Applied Combinatorics on Words

Lothaire

June 23, 2004

Contents

Presentation

A series of important applications of combinatorics on words has emerged with the development of computerized text and string processing, especially in biology and in linguistics. The aim of this volume is to present, in a unified treatment, some of the major fields of applications. The main topics that are covered in this book are

1. Algorithms for manipulating text, such as string searching, pattern matching, and testing a word for special properties.
2. Efficient data structures for retrieving information on large indexes, including suffix trees and suffix automata
3. Combinatorial, probabilistic and statistical properties of patterns in finite words, and more general pattern, under various assumptions on the sources of the text.
4. Inference of regular expressions.
5. Algorithms for repetitions in strings, such as maximal run or tandem repeats.
6. Linguistic text processing, especially analysis of the syntactic and semantic structure of natural language. Applications to language processing with large dictionaries.
7. Enumeration, generation and sampling of complex combinatorial structures by their encodings in words.

This book is actually the third of a series of books on combinatorics on words. Lothaire's "Combinatorics on Words" appeared in its first printing in 1984 as Volume 17 of the Encyclopedia of Mathematics. It was based on the impulse of M. P. Schützenberger's scientific work. Since then, the theory developed to a large scientific domain. It was reprinted in 1997 in the Cambridge Mathematical Library. Lothaire is a *nom de plume* for a group of authors initially constituted of former students of Schützenberger. Along the years, it has enlarged to a broader community coordinated by the editors. A second volume of Lothaire's series, entitled "Algebraic Combinatorics on Words" appeared in 2002. It contains both complements and newx developments that emerged since the publication of the first volume.

The content of this volume is quite applied, in comparison with the two previous ones. However, we have tried to follow the same spirit, namely to present introductory expositions, with full descriptions and numerous examples. Refinements are frequently deferred to problems, or mentioned in Notes. There is presently no similar book that covers these topics in this way.

Although each chapter has a different author, the book is really a cooperative work. A set of common notation has been agreed upon. Algorithms are presented in a consistent way using transparent conventions. There is also a common general index, and a common list of bibliographic references.

This book is independent of Lothaire's other books, in the sense that no knowledge of the other volumes is assumed.

The book has been written with the objective of being readable by a large audience. The prerequisits are those of a general scientific education. Some chapters may require a more advanced preparation. A graduate student in science or engineering should have no difficulty in reading all the chapters. A student in linguistics should be able to read part of it with profit and interest.

Outline of contents.

The general organisations is described below.

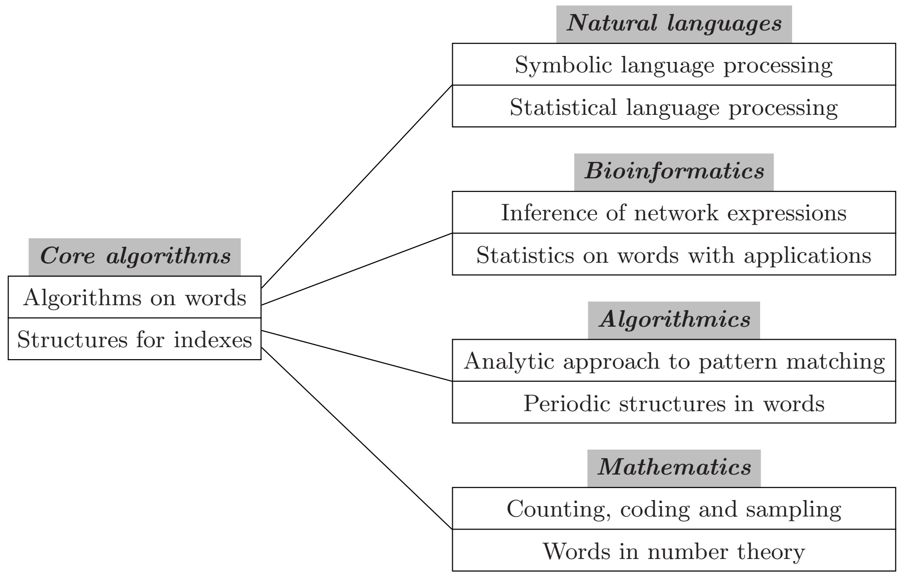


Figure 0.1. Overall structure of "Applied Combinatorics on Words".

The two first chapters are devoted to core algorithms. The first, "Algorithms on words", is is quite general, and is used in all other chapters. The second chapter, "Structures for indexes", is fundamental for all advanced algorithmic treatment, and more technical.

Among the applications, a first domain is linguistics, represented by two chapters entitled "Symbolic language processing" and "Statictical language processing".

A second application is biology. This is covered by two chapters, entitled "Inference of network expressions", and "Statistics on Words with Applications to Biological Sequences".

The next block is composed of two chapters dealing with algorithmics, a subject which is of interest for its own in theoretical computer science, but also related to biology and linguistics One chapter is entitled "Analytic approach to pattern matching" and deals with generalized pattern matching algorithms. A chapter entitled "Periodic structures in words" describes algorithms used for discovering and enumerating repetitions in words.

A final block is devoted to applications to mathematics (and theoretical physics). It is represented by two chapters. The first chapter, entitled "Counting, coding and sampling with words" deals with the use of words for coding combinatorial structures. Another chapter, entitled "Words in number theory" deals with transcendence, fractals and dynamical systems.

Description of contents.

Basic algorithms, as needed later, and notation are given in Chapter 1 "Algorithms on words", written by Jean Berstel and Dominique Perrin. This chapter also contains basic concepts on automata, grammars, and parsing. It ends with an exposition of probability distribution on words. The concepts and methods introduced are used in all the other chapters.

Chapter 2, entitled "Structures for indexes" and written by Maxime Crochemore, presents data structures for the compact representation of the suffixes of a text. These are used in several subsequent chapters. Compact suffix trees are presented, and construction of these trees in linear time is carefully described. The theory and algorithmics for suffix automata are presented next. The main application, namely the construction of indexes, is described next. Many other applications are given, such as detection of repetitions or forbidden words in a text, use as a pattern matching machine, and search for conjugates.

The first domain of applications, linguistics, is represented by Chapter 3 and Chapter 4. Chapter 3, entitled "Symbolic language processing" is written by Eric Laporte. In language processing, a text or a discourse is a sequence of sentences; a sentence is a sequence of words; a word is a sequence of letters. The most universal levels are those of sentence, word and letter (or phoneme), but intermediate levels exist, and can be crucial in some languages, between word and letter: a level of morphological elements (e.g. suffixes), and the level of syllables. The discovery of this piling up of levels, and in particular of word level and phoneme level, delighted structuralist linguists in the 20th century. They termed this inherent, universal feature of human language as "double articulation".

This chapter is organized around the main levels of any language modelling: first, how words are made from letters; second, how sentences are made from words. It surveys the basic operations of interest for language processing, and for each type of operation, it examines the formal notions and tools involved. The main originality of this presentation is the systematic and consistent use of finite state automata at every level of the description. This point of view is reflected in some practical implementations of natural language processing systems.

Chapter 4, entitled "Statistical language processing" is written by Mehryar Mohri. It presents the use of statistical methods to natural language processing. The main tool developed is the notion of weighted transducers. The weights are numbers in some semiring that can represent probabilities. Applications to speech processing are discussed.

The block of applications to biology is concerned with analysis of word occurences, pattern matching, and connections with genome analysis. It is covered by the next two chapters, and to some extent also by the alogrithmics bloc.

Chapter 5, "Inference of network expressions", is written by Nadia Pisanti and Marie-France Sagot. This chapter introduces various mathematical models and algorithms for inferring regular expression without Kleene star that appear repeated in a word or are common to a set of words. Inferring a network expression means to discover such expressions which are initially unknown, from the word(s) where the repeated (or common) expressions will be sought. This is in contrast with the string searching problem considered in other chapters. This has many applications, notably in molecular biology, system security, text mining etc. Because of the richness of the mathematical and algorithmical problems posed by molecular biology, we concentrate on applications in this area. Applications to biology motivate us also to consider network expressions that appear repeated not exactly but approximately.

Chapter 6 is written by Gesine Reinert, Sophie Schbath and Michael Waterman, and entitled "Statistics on Words with Applications to Biological Sequences". Properties of words in sequences have been of considerable interest in many fields, such as coding theory and reliability theory, and most recently in the analysis of biological sequences. The latter will serve as the key example in this chapter.

Two main aspects of word occurrences in biological sequences are: where do they occur and how many times do they occur? An important problem, for instance, was to determine the statistical significance of a word frequency in a DNA sequence. The naive idea is the following: a word may be significantly rare in a DNA sequence because it disrupts replication or gene expression, (perhaps a negative selection factor), whereas a significantly frequent word may have a fundamental activity with regard to genome stability. Well-known examples of words with exceptional frequencies in DNA sequences are certain biological palindromes corresponding to restriction sites avoided for instance in *E. coli*, and the Cross-over Hotspot Instigator sites in several bacteria.

Statistical methods to study the distribution of the word locations along a sequence and word frequencies have also been an active field of research; the goal of this chapter is to provide an overview of the state of this research.

Because DNA sequences are long, asymptotic distributions were proposed first. Exact distributions exist now, motivated by the analysis of genes and protein sequences. Unfortunately, exact results are not adapted in practice for long sequences because of heavy numerical calculation, but they allow the user to assess the quality of the stochastic approximations when no approximation error can be provided. For example, BLAST is probably the best-known algorithm for DNA matching, and it relies on a Poisson approximation. This is another motivation for the statistical analysis given in this chapter.

The algorithmics block is composed of two chapters. In Chapter 7, entitled "Analytic approach to pattern matching", and written by Philippe Jacquet and Wojciech Szpankowski, pattern matching is considered for various types of patterns, and for various types of sources. Single patterns, sequences of patterns, and sequences of patterns with sepatation conditions are considered. The sources are Bernoulli and Markov, and also more general sources arising from dynamical systems. The derivation of the equations is heavily based on combinatorics on words and formal languages.

Chapter 9, written by Roman Kolpakov and Gregory Koucherov and entitled "Periodic structures in words", deals with the algorithmic problem of detecting, counting and enumeration repetitions in a word. The interest for this is in text processing, compression and genome analysis, where tandem repeats may have a particular signification. Linear time algorithm exist for detecting tandem repeats, but since there may be quadratically many repetitions, maximal repetitions or "runs" are of importance, and are considered in this chapter.

A final block is concerned with applications to mathematics. Chapter 8, written by Dominique Poulalhon and Gilles Schaeffer, is entitled "Counting, coding and sampling with words". Its aim is to give typical descriptions of the interaction of combinatorics on words with the treatments of combinatorial structures. The chapter is focused on three aspects of enumeration: counting elements of a family according to their size, generating them uniformly at random, and coding them as compactly as possible by binary words. These aspects are respectively illustrated on examples taken from classical combinatorics (walks on lattices), from statistical physics (convex polyominoes and directed animals), and from graph algorithmics (planar maps). The rationale of the chapter is that nice enumerative properties are the visible traces of structural properties, and that making the latters explicit in terms of words of simple languages is a way to solve simultaneously and simply the three problems above.

Chapter 10 is written by Jean-Paul Allouche and Valérie Berthé. It is entitled "Words in number theory". This chapter is concerned with the interconnection between combinatorial properties of infinite words, such as repetitions, and transcendental numbers. A second part considers a famous infinite word, called the Tribonacci word, to investigate and illustrate connections between combinatorics on words and dynamical systems, quasicrystals, the Rauzy fractal, rotation on the torus, etc. Relations to the cut and project method are described, and an application to simultaneous approximation is given.

Acknowledgements.

Gesine Reinert, Sophie Schbath and Michael Waterman would like to thank Simon Tavaré for many helpful comments. Thanks goes also to Xueying Xie for pointing out inconsistencies in a previous version concerning testing for the order of a Markov chain.

Their work was supported in part by Sandia National Laboratories, operated by Lockheed Martin for the U.S. Department of Energy under contract No. DE-AC04-94AL85000, and by the Mathematics, Information, and Computational Science Program of the Office of Science of the U.S. Department of Energy. Gesine Reinert was supported in part by EPSRC grant aGR/R52183/01. Michael Waterman was partially supported by Celera Genomics.

An earlier and shorter version of chapter 6 appeared as *Probabilistic and statistical properties of words: an overview* in the *Journal of Computational Biology*, Vol. 7 (2000), pp. 1–46. The authors thank Mary Ann Liebert, Inc. Publishers for permission to include that material here.

Philippe Jacquet and Wojciech Szpankowski thank J. Bourdon, P. Flajolet, M. Régnier and B. Vallée for collaborating on pattern matching problems, co-authoring papers, and commenting on this chapter. They also thank M. Drmota and J. Fayolle for reading the chapter and providing useful comments.

W. Szpankowski acknowledges NSF and NIH support through grants CCR-0208709 and R01 GM068959-01.

J.-P. Allouche and Valérie Berthé would like to express their gratitude to P. Arnoux, A. Rémondière, D. Jamet and A. Siegel for their careful reading and their numerous suggestions.

Jean Berstel
Dominique Perrin

Marne-la-Vallée, June 23, 2004

CHAPTER 1

Algorithms on Words

Version June 23, 2004

1.0. Introduction

This chapter is an introductory chapter to the book. It gives general notions, notation and technical background. It covers, in a tutorial style, the main notions in use in algorithms on words. In this sense, it is a comprehensive exposition of basic elements concerning algorithms on words, automata and transducers, and probability on words.

The general goal of "stringology" we pursue here is to manipulate strings of symbols, to compare them, to count them, to check some properties and perform simple transformations in an effective and efficient way.

A typical illustrative example of our approach is the action of circular permutations on words, because several of the aspects we mentioned above are present in this example. First, the operation of circular shift is a transduction which can be realized by a transducer. We include in this chapter a section (Section 1.5) on transducers. Transducers will be used in Chapter 3. The orbits of the transformation induced by the circular permutation are the so-called conjugacy classes. Conjugacy classes are a basic notion in combinatorics on words. The minimal element in a conjugacy class is a good representative of a class. It can be computed by an efficient algorithm (actually in linear time). This is one of the algorithms which appear in Section 1.2. Algorithms for conjugacy are again considered in Chapter 2. These words give rise to Lyndon words which have remarkable combinatorial properties already emphasized in Lothaire 1997. We describe in Section 1.2.5 the Lyndon factorization algorithm.

The family of algorithms on words has features which make it a specific field within algorithmics. Indeed, algorithms on words are often of low complexity but intricate and difficult to prove. Many algorithms have even a linear time complexity corresponding to a single pass scanning of the input word. This contrasts with the fact that correctness proofs of these algorithms are frequently complex. A well-known example of this situation is the Knuth–Morris–Pratt string searching algorithm (see Section 1.2.3). This algorithm is compact, and apparently simple but the correctness proof requires an sophisticated loop invariant.

The field of algorithms on words still has challenging open problems. One of them is the minimal complexity of the computation of a longest common subword of two words which is still unknown. We present in Section 1.2.4 the classic quadratic dynamic programming algorithm. A more efficient algorithm is mentioned in the Notes.

The field of algorithms on words is intimately related to formal models of computation. Among those models, finite automata and context-free grammars are the most used in practice. This is why we devote a section ((Section 1.3) to finite automata and another one to grammars and syntax analysis (Section 1.6). These models, and especially finite automata, regular expressions and transducers, are ubiquous in the applications. They appear in almost all chapters.

The relationship between words and probability theory is an old one. Indeed, one of the basic aspects of probability and statistics is the study of sequences of events. In the elementary case of a finite sample space, like in coin tossing, the sequence of outcomes is a word. More generally, a partition of an arbitrary probability space into a finite number of classes produces sequences over a finite set. Section 1.8 is devoted to an introduction to these aspects. They are developed later in Chapters 6 and 7.

We have chosen to present the algorithms and the related properties in a direct style. This means that there are no formal statements of theorems, and consequently no formal proofs. nevertheless, we give precise assertions and enough arguments to show the correctness of algorithms and to evaluate their complexity. In some cases, we use results without proof and we give bibliographic indications in the Notes.

For the description of algorithms, we use a kind of programming language close to usual programming languages. It gives more flexibility and improves the readability to do so instead of relying on a precise programming language.

The syntactic features of our programs make it similar to a language like Pascal, concerning the control structure and the elementary instructions. We take some liberty with real programs. In particular, we often omit declarations and initializations of variables. The parameter handling is C–like (no call by reference). Besides arrays, we use implicitly data structures like sets and stacks and pairs or triples of variables to simplify notation. All functions are global, and there is nothing like classes or other features of object-oriented programming. However, we use overloading for parsimony. The functions are referenced in the text and in the index by their name, like LongestCommonPrefix() for example.

1.1. Words

We briefly introduce the basic terminology on words. Let $\mathcal{A}$ be a finite set usually called the *alphabet*. In practice, the elements of the alphabet may be characters from some concrete alphabet, but also more complex objects. They may be themselves words on another alphabet, as in the case of syllables in natural language processing, as presented in Chapter 3. In information processing, any kind of record can be viewed as a symbol in some huge alphabet. This has as consequence that some apparently elementary operations on symbols, like the test for equality, often needs a careful definition and may require a delicate implementation.

We denote as usual by $\mathcal{A}^*$ the set of words over $\mathcal{A}$ and by ε the empty word. For a word w, we denote by $|w|$ the length of w. We use the notation $\mathcal{A}^+ = \mathcal{A}^* - \{\varepsilon\}$. The set $\mathcal{A}^*$ is a monoid. Indeed, the concatenation of words is associative, and the empty word is a neutral element for concatenation. The set $\mathcal{A}^+$ is sometimes called the *free semigroup* over $\mathcal{A}$, while $\mathcal{A}^*$ is called the *free monoid*.

A word w is called a *factor* (resp. a *prefix*, resp. a *suffix*) of a word u if there exist words x, y such that $u = xwy$ (resp. $u = wy$, resp. $u = xw$). The factor (resp. the prefix, resp. the suffix) is *proper* if $xy \neq \varepsilon$ (resp. $y \neq \varepsilon$, resp. $x \neq \varepsilon$). The prefix of length k of a word w is also denoted by $w[0..k-1]$.

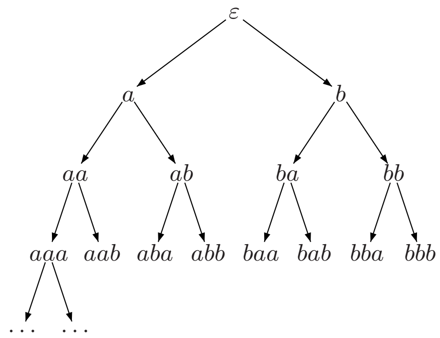


Figure 1.1. The tree of the free monoid on two letters.

The set of words over a finite alphabet $\mathcal{A}$ can be conveniently seen as a tree. Figure 1.1 represents $\{a, b\}$ as a binary tree. The vertices are the elements of $\mathcal{A}^*$. The root is the empty word ε. The sons of a node x are the words xa for $a \in \mathcal{A}$. Every word x can also be viewed as the path from leading from the root to the node x. A word x is a prefix of a word y if it is an ancestor in the tree. Given two words x and y, the longest common prefix of x and y is the nearest common ancestor of x and y in the tree.

A word x is a *subword* of a word y if there are words $u_1, \ldots, u_n$ and $v_0, v_1, \ldots, v_n$ such that $x = u_1 \cdots u_n$ and $y = v_0 u_1 v_1 \cdots u_n v_n$. Thus, x is obtained from y by erasing some factors in y.

Given two words x and y, a *longest common subword* is a word z of maximal length that is both a subword of x and y . There may exist several longest common subwords for two words x and y. For instance, the words abc and acb have the common subwords ab and ac.

We denote by $\operatorname{alph} w$ the set of letters having at least one occurrence in the word w.

The set of factors of a word x is denoted $F(x)$. We denote by $F(\mathcal{X})$ the set of factors of words in a set $\mathcal{X} \subset \mathcal{A}^*$. The *reversal* of a word $w = a_1 a_2 \cdots a_n$, where $a_1, \ldots, a_n$ are letters, is the word $\tilde{w} = a_n a_{n-1} \cdots a_1$. Similarly, for $\mathcal{X} \subset \mathcal{A}^*$, we denote $\widetilde{\mathcal{X}} = \{\tilde{x} \mid x \in \mathcal{X}\}$. A *palindrome word* is a word w such that $w = \tilde{w}$. If $|w|$ is even, then w is a palindrome if and only if $w = x\tilde{x}$ for some word x. Otherwise w is a palindrome if and only if $w = xa\tilde{x}$ for some word x and some letter a.

An integer $p \geq 1$ is a *period* of a word $w = a_1 a_2 \cdots a_n$ where $a_i \in \mathcal{A}$ if $a_i = a_{i+p}$ for $i = 1, \ldots, n-p$. The smallest period of w is called *the* period the *minimal* period of w.

A word $w \in \mathcal{A}^+$ is *primitive* if $w = u^n$ for $u \in \mathcal{A}^+$ implies $n = 1$.

Two words x, y are *conjugate* if there exist words u, v such that $x = uv$ and $y = vu$. Thus conjugate words are just cyclic shifts of one another. Conjugacy is thus an equivalence relation. The conjugacy class of a word of length n and period p has p elements if p divides n and has n elements otherwise. In particular, a primitive word of length n has n distinct conjugates.

1.1.1. Ordering

There are three order relations frequently used on words. We give the definition of each of them.

The *prefix order* is the partial order defined by $x \leq y$ if x is a prefix of y.

Two other orders, the *radix order* and the *lexicographic order* are refinements of the prefix order which are defined for words over an ordered alphabet $\mathcal{A}$. Both are total orders.

The *radix order* is defined by $x \leq y$ if $|x| < |y|$ or $|x| = |y|$ and $x = uax'$ and $y = uby'$ with a, b letters and $a \leq b$. If integers are represented in base k without

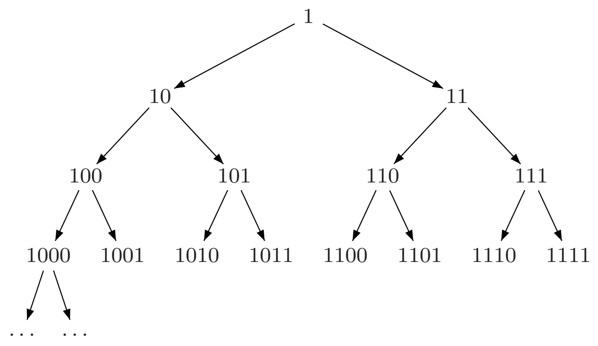


Figure 1.2. The tree of integers in binary notation.

leading zeroes, then the radix order on their representations corresponds to the natural ordering of the integers. If we allow leading zeroes, the same holds provided the words have the same length (which always can be achieved by padding).

For $k = 2$, the tree of words without leading zeroes is given in Figure 1.2. The radix order corresponds to the order in which the vertices are met in a breadth-first traversal. The index of a word in the radix order is equal to the number represented by the word in base 2.

The *lexicographic order*, also called *alphabetic order*, is defined as follows. Given two words x, y, we have $x < y$ if x is a proper prefix of y or if there exist factorizations $x = uax'$ and $y = uby'$ with a, b letters and $a < b$. This is the usual order in a dictionary. Note that $x < y$ in the radix order if $|x| < |y|$ or if $|x| = |y|$ and $x < y$ in the lexicographic order.

1.1.2. Distances

A *distance* over a set E is a function d which assigns to each element of E a nonnegative number such that:

(i) $d(u, v) = d(v, u)$,

(ii) $d(u, w) \leq d(u, v) + d(v, w)$ (triangular inequality)

(iii) $d(u, v) = 0$ iff $u = v$.

Several distances between words are used. The most common is the *Hamming distance*. It is only defined on words of equal length. For two words $u = a_0 \cdots a_{n-1}$ and $v = b_0 \cdots b_{n-1}$, where a_i, b_i are letters, it is the number $d_H(u, v)$ of indices i with $0 \leq i \leq n-1$ such that $a_i \neq b_i$. In other terms

$$d_H(u, v) = \mathrm{Card}\{i \mid 0 \leq i \leq n-1 \text{ and } a_i \neq b_i\}\,.$$

Thus the Hamming distance is the number of *mismatches* between u and v. It can be verified that d_H is indeed a distance. Observe that $d_H(u, v) = n - p$ where p is the number of positions where u nd v coincide. In a more general setting, a distance between letters is used instead of just counting for 1 each mismatch.

The Hamming distance takes into account the differences at the same position. In this way, it can be used as a measure of modifications or errors caused by a modification of a symbol by another one, but not of a deletion or an insertion. Another distance is the subword distance which is defined as follows. Let

(a) Hamming distance (b) Subword distance

Figure 1.3. The Hamming distance is 3 and the subword distance is 2.

u be a word of length n and v be a word of length m, and p be the length of a longest common subword of u and v. The *subword distance* between u and v is defined as $d_S(u, v) = n + m - 2p$. It can be verified that $d_S(u, v)$ is the minimal number of insertions and suppressions that changes u into v. The name *indel* (for *in*sertions and *del*etions) is used to qualify the transformation consisting in either an insertion or a deletion.

A common generalization of the Hamming distance and the subword distance is the *edit distance*. It takes into account the substitutions of a symbol by another in additions to indels (see Problem 1.1.2).

A related distance is the *prefix distance*. It is defined as $d(u, v) = n+m-2p$ where $n = |u|$, $m = |v|$ and p is the length of the longest common prefix of u and v. It can be verified that the prefix distance is actually the length of the shortest path from u to v in the tree of the free monoid.

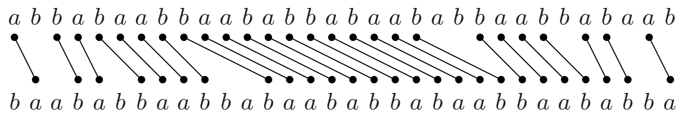


Figure 1.4. The Hamming distance of these two Thue-Morse blocks of length 32 is equal to their length, their subword distance is only 6.

1.2. Elementary algorithms

In this section, we treat algorithmic problems related to the basic notions on words: prefixes, suffixes, factors.

1.2.1. Prefixes and suffixes

Recall that a word x is a *prefix* of a word y if there is a word u such that $y = xu$. It is said to be *proper* if u is non empty. Checking whether x is a prefix of y is straightforward. Algorithm LongestCommonPrefix below returns the length of the longest common prefix of two words x and y.

```
LongestCommonPrefix(x, y)
  ▷ x has length m, y has length n
  i ← 0
  while i < m and i < n and x[i] = y[i] do
        i ← i + 1
  return i
```

In the tree of a free monoid, the length of the longest common prefix of two words is the height of the least common ancestor.

As mentioned earlier, the conceptual simplicity of the above algorithm hides implementation details such as the computation of equality between letters.

1.2.2. Overlaps and borders

We introduce first the notion of overlap of two words x and y. It captures the amount of possible overlap between the end of x and the beginning of y. To avoid trivial cases, we rule out the case where the overlap would be the whole word x or y. Formally, the *overlap* of x and y is the longest proper suffix of x that is also a proper prefix of y. For example, the overlap of *abacaba* and *acabaca* has length 5. The *border* of a non empty word w is the overlap of w and itself. Thus it is the longest word u which is both a proper prefix and a proper suffix of w. The overlap of x and y is denoted by $\mathrm{overlap}(x, y)$, and the border of x by $\mathrm{border}(x)$. Thus $\mathrm{border}(x) = \mathrm{overlap}(x, x)$.

As we shall see, the computation of the overlap of x and y is intimately related to the computation of the border. This is due to the fact that the overlap of x and y involves the computation of the overlaps of the prefixes of x and y. Actually, one has $\mathrm{overlap}(xa, y) = \mathrm{border}(xa)$ whenever x is a prefix of y and a is a letter. Next, the following formula allows the computation of the overlap of xa and y, where x, y are words and a is a letter. Let $z = \mathrm{overlap}(x, y)$.

$$\mathrm{overlap}(xa, y) = \begin{cases} za & \text{if } za \text{ is a prefix of } y, \\ \mathrm{border}(za) & \text{otherwise.} \end{cases}$$

Observe that $\mathrm{border}(za) = \mathrm{overlap}(za, y)$ because z is a prefix of y. The computation of the border is an interesting example of a non trivial algorithm on words. A naive algorithm would consist in checking for each prefix of w whether it is also a suffix of w and to keep the longest such prefix. This would obviously require a time proportional to $|w|^2$. We will see that it can be done in time proportional to the length of the word. This relies on the following recursive formula allowing to compute the border of xa in terms of the border of x, where x is a word and a is a letter. Let $u = \mathrm{border}(x)$ be the border of x. Then for each letter a,

$$\mathrm{border}(xa) = \begin{cases} ua & \text{if } ua \text{ is a prefix of } x, \\ \mathrm{border}(ua) & \text{otherwise.} \end{cases} \tag{1.2.1}$$

The following algorithm (Algorithm Border) computes the length of the border of a word x of length m. It outputs an array b of $m+1$ integers such that $b[j]$ is the length of the border of $x[0..j-1]$. In particular, the length of $\mathrm{border}(x)$ is $b[m]$. It is convenient to set $b[0] = -1$. For example, if $x = abaababa$, the array b is

	0	1	2	3	4	5	6	7	8
b :	-1	0	0	1	1	2	3	2	3

BORDER(x)
1 ▷ x has length m, b has size $m+1$
2 $i \leftarrow 0$
3 $b[0] \leftarrow -1$
4 **for** $j \leftarrow 1$ **to** $m-1$ **do**
5     $b[j] \leftarrow i$
6   ▷ Here $x[0..i-1] = \text{border}(x[0..j-1])$
7   **while** $i \geq 0$ and $x[j] \neq x[i]$ **do**
8         $i \leftarrow b[i]$
9     $i \leftarrow i+1$
10 $b[m] \leftarrow i$
11 **return** b

This algorithm is an implementation of Formula (1.2.1). Indeed, the body of the loop on j computes, in the variable i, the length of the border of $x[0..j]$. This value will be assigned to $b[j]$ at the next increase of j. The inner loop is a translation of the recursive formula.

The algorithm computes the border of x (or the table b itself) in time $O(|x|)$. Indeed, the execution time is proportional to the number of comparisons of symbols performed at line 7. Each time a comparison is done, the expression $2j - i$ increases strictly. In fact, either $x[j] = x[i]$ and i, j increase both by 1. Or $x[j] \neq x[i]$, and j remains constant while i decreases strictly (since $b[i] < i$). Since the value of the expression is initially 0 and is bounded by $2|x|$, the number of comparisons is at most $2|x|$.

The computation of the overlap of two words x, y will be done in the next section.

1.2.3. Factors

In this section, we consider the problem of checking whether a word x is a factor of a word y. This problem is usually referred to as a *string matching problem*. The word x is called the *pattern* and y is the *text*. A more general problem, referred to as *pattern matching*, occurs when x is replaced by a *regular expression* $\mathcal{X}$ (see Section 1.4. The evaluation of the efficiency of string matching or pattern matching algorithms depends on which parameters are considered. In particular, one may consider the pattern to be fixed (because several occurrences of the same pattern are looked for in an unknown text), or the text to be fixed (because several different pattern will be searched in this text). When the pattern or the text is fixed, it may be subject to a preprocessing. Moreover, the evaluation of the complexity can take into account either only the computation time, or both time and space. This may make significant difference on very large texts and patterns.

We begin by a naive quadratic string searching algorithm. To check whether a word x is a factor of a word y, it is clearly enough to test for each index $j = 0, \ldots, n-1$ if x is a prefix of the word $y[j..n-1]$.

```
NaiveStringMatching(x, y)
  ▷ x has length m, y has length n
  (i, j) ← (0, 0)
  while i < m and j < n do
      if x[i] = y[j] then
          (i, j) ← (i + 1, j + 1)
      else j ← j − i + 1
          i ← 0
  return i = m
```

The number of comparisons required in the worst case is $O(|x||y|)$. The worst case is reached for $x = a^m b$ and $y = a^n$. The number of comparisons performed is in this case $m(n - m - 1)$.

We shall see now that it is possible to search a word x inside another word y in linear time, that is in time $O(|x|+|y|)$. The basic idea is to use a finite automaton recognizing the words ending with x. If we can compute some representation of it in time $O(|x|)$, then it will be straightforward to process the word y in time $O(|y|)$.

The wonderfully simple solution presented below uses the notion of border of a word. Suppose that we are in the process of identifying x inside y, the position i in x being placed in front of position j in y, as in the naive algorithm. We can set then $x = ubt$ where $b = x[i]$ and $y = wuaz$ where $a = y[j]$. If a=b, the process goes on with $i + 1, j + 1$. Otherwise, instead of just shifting x to the right one place (i.e. $j = j - i + 1, i = 0$), we can take into account that the next possible position for x is determined by the border of u. Indeed, we must have $y = w'u'az$ and $x = u'ct'$ with u' both a prefix of u and a suffix of u since $w'u' = wu$. Hence the next comparison to perform is between $y[j]$ and $x[k]$ where $k - 1$ is the length of the border of u.

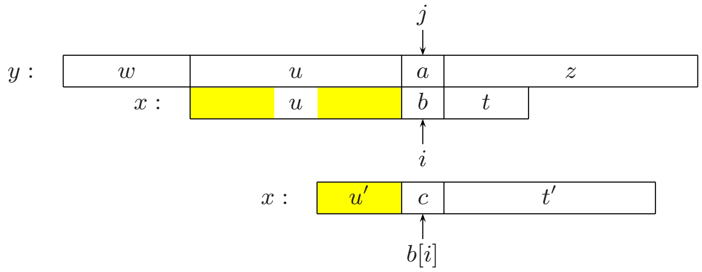


Figure 1.5. Checking $y[j]$ against $x[i]$: if they are different, $y[j]$ is checked against $x[b[i]]$.

The algorithm is realized by the following program (Algorithm SearchFactor). It returns the starting position of the first occurrence of the word x inside the word y, and $|y|$ if x is not a factor of y. It uses an array b of $|x|+1$ integers such that $b[i]$ is the length of the border of $x[0..i-1]$.

```
SearchFactor(x, y)
▷ x has length m, y has length n
▷ b is the array of length of borders of the prefixes of x
b ← Border(x)
(i, j) ← (0, 0)
while i < m and j < n do
      while i ≥ 0 and x[i] ≠ y[j] do
            i ← b[i]
      (i, j) ← (i + 1, j + 1)
return i = m
```

The time complexity is $O(|x| + |y|)$. Indeed, the computation of the array b can be done in time $O(|x|)$ as in Section 1.2.2. Further, the analysis of the algorithm given by the function SearchFactor is the same as for the function Border. The expression $2j - i$ increases strictly at each comparison of two letters, and thus the number of comparisons is bounded by $2|y|$. Thus, the complete time required to check whether x is a factor of y is $O(|x| + |y|)$ as announced.

Computing the overlap of two word x, y can be done as follows. We may suppose $|x| < |y|$. The value of $\mathrm{overlap}(x, y)$ is the final value of the variable i in the algorithm SearchFactor applied to the pair (y, x)

1.2.4. Subwords

We now consider the problem of looking for subwords. The following algorithm checks whether x is a subword of y. In contrast to the case of factors, a greedy algorithm suffices to perform the check in linear time.

```
IsSubword(x, y)
▷ x has length m, y has length n
(i, j) ← (0, 0)
while i < m and j < n do
      if x[i] = y[j] then
            i ← i + 1
      j ← j + 1
return i = m
```

We denote by $\mathrm{lcs}(x, y)$ the set of *longest common subwords* (also called longest common subsequences) of two words x and y. The computation of the longest common subwords is a classical algorithm with many practical uses. We present below a quadratic algorithm. It is based on the following formula.

$$\mathrm{lcs}(xa, yb) = \begin{cases} \mathrm{lcs}(x, y)a & \text{if } a = b, \\ \max(\mathrm{lcs}(xa, y), \mathrm{lcs}(x, yb)) & \text{otherwise.} \end{cases}$$

where max() stands for the union of the sets if their elements have equal length, and for the set with the longer words otherwise.

In practice, one computes the length of the words in $\mathrm{lcs}(x, y)$. For this, define an array $M[i, j]$ by $M[i, j] = k$ if the longest common subwords to the prefixes of length i of x and j of y have length k. The previous formula then translates into

$$M[i+1, j+1] = \begin{cases} M[i,j]+1 & \text{if } a = b, \\ \max(M[i+1,j], M[i,j+1]) & \text{otherwise.} \end{cases}$$

For instance, if $x = abba$ and $y = abab$, the array M is the following.

		a	b	a	b
	0	0	0	0	0
a	0	1	1	1	1
b	0	1	2	2	2
b	0	1	2	2	3
a	0	1	2	3	3

The first row and the first column of the array M are initialized at 0. The following function computes the array M.

```
LcsLengthArray(x, y)
▷ x has length m and y has length n
for i ← 0 to m − 1 do
    for j ← 0 to n − 1 do
        if x[i] = y[j] then
            M[i + 1, j + 1] ← M[i, j] + 1
        else M[i + 1, j + 1] ← max(M[i + 1, j], M[i, j + 1])
return M
```

The above algorithm has quadratic time and space complexity. Observe that the length of the longest common subwords, namely the value $M[m, n]$, can be computed in linear space (but quadratic time) by computing the matrix M row by row or column by column. To recover the a word in $\mathrm{lcs}(x, y)$, it is enough to walk backwards through the array M.

```
Lcs(x, y)
▷ result is a longest common word w
M ← LcsLengthArray(x, y)
(i, j, k) ← (m − 1, n − 1, M[m, n] − 1)
while k ≥ 0 do
    if x[i] = y[j] then
        w[k] ← x[i]
        (i, j, k) ← (i − 1, j − 1, k − 1)
    else if M[i + 1, j] < M[i, j + 1] then
            i ← i − 1
        else j ← j − 1
    return w
```

1.2.5. Conjugacy and Lyndon words

Two words x, y are said to be *conjugate* if $x = uv, y = vu$, for some words u, v. Thus two words are conjugate if they differ only by a cyclic permutation of their letters.

To check whether x and y are conjugate, we can compare all possible cyclic permutations of x with y. This requires $O(|x||y|)$ operations. Actually we can do much better as follows. Indeed, x and y are conjugate if and only if $|x| = |y|$ and if x is a factor of yy. Indeed, if $|x| = |y|$ and $yy = uxv$, we have $|y| \leq |ux|, |xv|$ and thus there are words u', v' such that $x = v'u'$ and $y = uv' = u'v$. Since $|x| = |y|$, we have $|u'| = |u|$, whence $u = u'$ and $v = v'$. This shows that $x = vu, y = uv$.

Hence, using the linear time algorithm SearchFactor of Section 1.2.3, we can check in $O(|x| + |y|)$ whether two words x, y are conjugate.

Recall that a *Lyndon word* is a word which is strictly smaller than any of its conjugates for the alphabetic ordering. In other terms, a word x is a Lyndon word if for any factorization $x = uv$ with u, v non empty, one has $uv < vu$. A Lyndon word is in particular primitive.

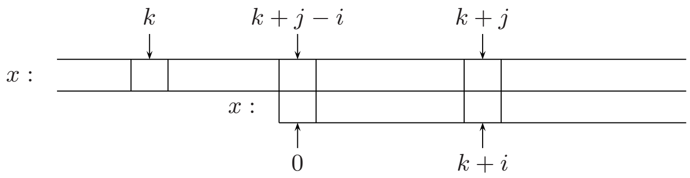


Figure 1.6. Checking whether $x[k..k + m - 1]$ is the least circular conjugate of x.

Any primitive word has a conjugate which is a Lyndon word, namely its least conjugate. Computing the smallest conjugate of a word is a practical way to compute a standard representative of the conjugacy class of a word (this is sometimes called *canonization*). This can be done in linear time by the following algorithm, which is a modification of the algorithm Border of Section 1.2.2. It is applied to a word x of length m. We actually use an array containing x^2, and called this array x. Of course, an array of length m would suffice provided the indices are computed mod m.

```
CircularMin(x)
 1  (i, j, k) ← (0, 1, 0)
 2  b[0] ← −1
 3  while k + j < 2m do
 4      ▷ Here x[k..k + i − 1] = border(x[k..k + j − 1])
 5      if j − i = m then
 6          return k
 7      b[j] ← i
 8      while i ≥ 0 and x[k + j] ≠ x[k + i] do
 9          if x[k + j] < x[k + i] then
10              (k, j) ← (k + j − i, i)
11          i ← b[i]
12      (i, j) ← (i + 1, j + 1)
```

Algorithm CircularMin looks like Algorithm Border. Indeed, if we discard lines 5–6 and lines 9–10 in algorithm CircularMin, the variable k remains 0 and we obtain an essentially equivalent algorithm (with a **while** loop replacing the **for** loop). The key assertion of this algorithm is that $x[k..k+i-1] = \mathrm{border}(x[k..k+j-1])$, as indicated at line 4. This is the same as the assertion in Algorithm Border for $k = 0$. The array b contains the information on borders, in the sense that $b[j]$ is the length of $\mathrm{border}(x[k..k+j-1])$.

The value of k is the index of the beginning of a candidate for a least conjugate of x (see Figure 1.6). If the condition at line 9 holds, a new candidate has been found. The assignment at line 10 shifts the value of k by $j - i$, and j is adjusted in such a way that the value of $k + j$ is not modified. The modifications of the value of k does not require the entire re-computation of the array b. Indeed, the values $b[j']$ for $0 \le j' < i$ serve both for the old and the new candidate. For the same reason as for Algorithm Border, the time complexity is linear in the size of x.

Any word admits a unique factorization as a non increasing product of Lyndon words. In other words, for any word x, there is a factorization

$$x = \ell_1^{n_1} \cdots \ell_r^{n_r}$$

where $r \ge 0$, $n_1, \ldots, n_r \ge 1$, and $\ell_1 > \cdots > \ell_r$ are Lyndon words. We discuss now an algorithm to compute this factorization.

The following program computes the pair (ℓ_1, n_1) for x in linear time. By iteration, it allows to compute the Lyndon factorization in linear time.

```
LyndonFactorization(x)
▷ x has length m
(i, j) ← (0, 1)
while j < m and x[i] ≤ x[j] do
      if x[i] < x[j] then
            i ← 0
      else i ← i + 1
      j ← j + 1
return (j − i, ⌊j/(j − i)⌋)
```

The idea of the algorithm is the following. Assume that at some step, we have $x = \ell^n py$, where ℓ is a Lyndon word, $n \geq 1$ and p is a proper prefix of ℓ. The pair (ℓ, n) is a candidate for the value (ℓ_1, n_1) of the factorization. The relation with the values i, j of the program is given by $j = |\ell^n p|$, $j - i = |\ell|$, $n = \lfloor j/(j-i) \rfloor$. Let $a = x[i]$, $b = x[j]$. Then $\ell = paq$ for some word q, and $x = \ell^n pbz$. If $a < b$, then $\ell' = \ell^n pb$ is a Lyndon word. The pair $(\ell', 1)$ becomes the new candidate. If $a = b$, then pb replaces p. Finally, if $a > b$ the pair (ℓ, n) is the correct value of (ℓ_1, n_1).

The above algorithm can also be used to compute the Lyndon word ℓ in the conjugacy class of a primitive word x. Indeed, ℓ is the only Lyndon word of length $|x|$ that appears in the Lyndon factorization of xx. Thus, Algorithm LyndonFactorization gives an alternative to Algorithm CircularMin.

1.3. Tries and automata

In this section, we consider sets of words. These sets arise in a natural way in applications. Dictionaries in natural language processing, or more generally text processing in data bases are typical examples. Another situation is when one considers properties of words, and the sets satisfying such a property, for example the set of all words containing a given pattern. We are interested in the practical representation for retrieval and manipulation of sets of words.

The simplest case is the case of finite, but possibly very large sets. General methods for manipulation of sets may be used. This includes hash functions, bit vectors, and various families of search trees. These general methods are sometimes available in programming packages. We will be interested here in methods that apply specifically to sets of words.

Infinite sets arise naturally in pattern matching. The natural way to handle them is by means of two equivalent notions: regular expressions and finite automata. We describe here in some detail these approaches.

1.3.1. Tries

A *trie* is the simplest non trivial structure allowing to represent a finite set $\mathcal{X}$ of words. It has both the advantage of reducing the space required to store the set of words and to allow a fast access to each element.

A trie R is a rooted tree. Each edge is labelled with a letter. The labels have the property that two distinct edges starting in the same vertex have distinct labels. A subset T of the set of vertices is called the set of *terminal vertices*. The set $\mathcal{X}$ of words *represented* by the trie R is the set of labels of paths from the root to a vertex in T. It is convenient to assume that every vertex is on a path from the root to some vertex in T (since otherwise the vertex could be removed). In particular, every leaf of the tree is a terminal vertex.

Example 1.3.1. The trie represented on Figure 1.3.1 represents the set

$$\mathcal{X} = \{\text{leader, let, letter, sent}\}.$$

The terminal vertices are doubly circled.

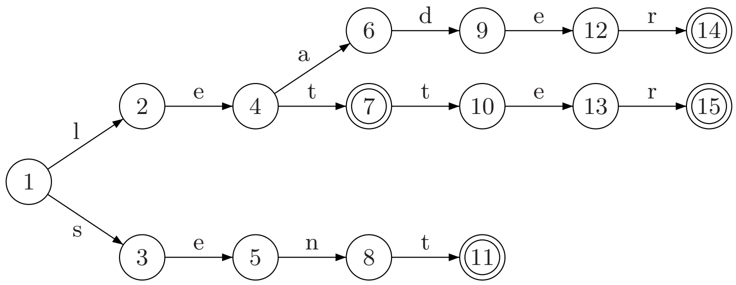


Figure 1.7. A trie

To implement a trie, we use a function $\textsc{Next}(p, a)$ which gives the vertex q such that the edge (p, q) is labeled a. We assume that $\textsc{Next}(p, a) = -1$ if the function is not defined. The root of the tree is the value of $\textsc{Root}()$.

```
IsInTrie(w)
  ▷ checks if the word w of length n is in the trie
  (i, p) ← LongestPrefixInTrie(w)
  return i = n and p is a terminal vertex
```

Function IsInTrie returns true if the word w is in the set represented by the trie. It uses the function LongestPrefixInTrie() to compute the pair (i, p) where i is the length of the longest prefix of w which is the label of a path in the trie, and p is the vertex reached by this prefix. For future use, we give a slightly more general version of this function. It computes the pair (i, p) where i is the length of the longest prefix of the suffix of w starting in position j.

```
LongestPrefixInTrie(w, j)
▷ returns the length of the longest prefix of w[j..n − 1]
▷ in the trie, and the vertex reached by this prefix.
q ← Root()
for i ← j to n − 1 do
      p ← q
      q ← Next(q, w[i])
      if q is undefined then
            return (i − j, p)
return (n − j, q)
```

Searching a word in a trie is done in linear time with respect to the length of the word. It does not depend on the size of the trie. This is the main advantage of this data structure. However, this is only true under the assumption that the function Next can be computed in constant time. In practice, if the alphabet is of large size, this might not be longer true.

To add a word to a trie amounts to the following simple function.

```
AddToTrie(w)
1  ▷ adds the word w to the trie.
2  (i, p) ← LongestPrefixInTrie(w, 0)
3  for j ← i to n − 1 do
4        q ← NewVertex()
5        Next(p, w[j]) ← q
6        p ← q
7  Add q to the set of terminal vertices
```

We use a function NewVertex() to create a new vertex of the trie. The function AddToTrie() is linear in the length of w, provided Next() is in constant time.

To remove a word from a trie is easy if we have a function Father() giving the father of a vertex. The function can be tabulated during the construction of the trie (by adding the instruction Father(q) $\leftarrow$ p just after line 5 in Algorithm AddToTrie). The function Father() can also be computed on the fly during the computation of LongestPrefixInTrie() at line 2 of Algorithm RemoveFromTrie(.) Another possibility, avoiding the use of the function Father(), is to write the function RemoveFromTrie() recursively. We also use a boolean function IsLeaf() to test whether a vertex is a leaf or not.

```
RemoveFromTrie(w)
  ▷ removes the word w of length n from the trie
  (i, p) ← LongestPrefixInTrie(w, 0)
  ▷ i should be equal to n
  Remove p from the set of terminal vertices
  while IsLeaf(p) and p is not terminal do
        (i, p) ← (i − 1, Father(p))
        Next(p, w[i]) ← −1
```

The use of a trie structure reduce the space needed to represent a set of words, compared with a naive representation. If one wishes to further reduce the size, it is possible to use an acyclic graph instead of a tree. The result is an acyclic graph with labeled edges, an initial vertex and a set of terminal vertices. This is sometimes called a *directed acyclic word graph* abbreviated as DAWG.

Example 1.3.2. We represent below a DAWG for the set

$$\mathcal{X} = \{\text{leader, let, letter, sent}\}$$

of Example 1.3.1.

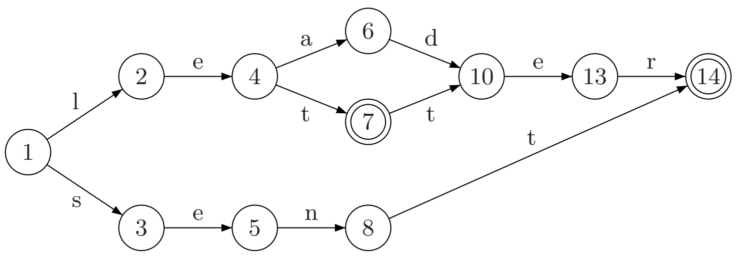


Figure 1.8. A directed acyclic word graph (DAWG).

For a given finite set $\mathcal{X}$ of words, there is a unique minimal DAWG representing $\mathcal{X}$. This is a particular case of a statement concerning finite automata that we shall see in the next section. The minimal DAWG is actually the minimal deterministic automaton recognizing $\mathcal{X}$, and standard algorithm exist to compute it.

1.3.2. Automata

An *automaton* over an alphabet $\mathcal{A}$ is composed of a set Q of *states*, a finite set $E \subset Q \times \mathcal{A}^* \times Q$ of *edges* or *transitions* and two sets $I, T \subset Q$ of *initial* and *terminal* states. For an edge $e = (p, w, q)$, the state p is the *origin*, w is the *label*, and q is the *end*.

The automaton is often denoted $\mathfrak{A} = (Q, E, I, T)$, or also (Q, I, T) when E is understood, or even $\mathfrak{A} = (Q, E)$ if $Q = I = T$.

A *path* in the automaton $\mathfrak{A}$ is a sequence

$$(p_0, w_1, p_1), (p_1, w_2, p_2), \ldots, (p_{n-1}, w_n, p_n)$$

of consecutive edges. Its label is the word $x = w_1 w_2 \cdots w_n$. The path *starts* at p_0 and *ends* at p_n. The path is often denoted

$$p_0 \xrightarrow{x} p_n$$

A path is *successful* if it starts in an initial state and ends in a terminal state. The set *recognized* by the automaton is the set of labels of its successful paths.

A set is *recognizable* or *regular* if it is the set of words recognized by some automaton.

The family of regular sets is both the simplest family of sets that admits an algorithmic description. It is also the most widely used one, because of its numerous closure properties.

A state p is *accessible* if there is a path starting in an initial state and ending in p. It is *coaccessible* if there is a path starting in p and ending in a terminal state. An automaton is *trim* if every state is accessible and coaccessible.

An automaton is *unambiguous* if, for each pair of states p, q, and for each word w, there is at most one path from p to q labeled with w. An automaton is represented as a labelled graph. Initial (final) states are distinguished by an incoming (outgoing) arrow.

Example 1.3.3. The automaton given in Figure 1.9 recognizes the set of words on the alphabet $\{a, b\}$ ending with aba. It is unambiguous and trim.

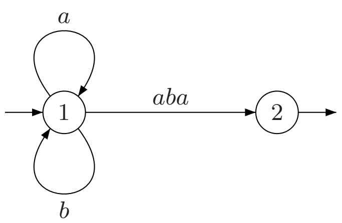


Figure 1.9. A nondeterministic automaton.

The definition of an automaton given above is actually an abstraction which went up from circuits and sequential processes. In this context, an automaton is frequently called a *state diagram* to mean that the states represent possible values of time changing variables.

In some situations, this representation is not adequate. In particular, the number of states can easily become too large. Indeed, the number of states is in general exponential in the number of variables. A typical example is the automaton which memorizes the n last input symbols. It has 2^n states on a binary alphabet but can be represented simply with n binary variables. Observe however that this situation is not general. In particular, automata occurring in linguistics or in bioinformatics cannot in general be represented with such parsimony.

We have introduced here a general model of automata which allows edges labelled by words. This allows in particular edges labelled by the empty word. Such an edge is usually called an ε-*transition*. We will use here two particular cases of this general definition. The first is that of a *synchronous* automaton in which all edges are labelled by letters. In this case, the length of a path equals the length if its label.

An automaton which is not synchronous is called *asynchronous*. Among asynchronous automata, we use *literal* automata as a second class. These have labels that are either letters or the empty word. In this case, the length of a path is always at least equal to the length of its label.

EXAMPLE 1.3.4. The automaton $\mathfrak{A}$ of Figure 1.10 is asynchronous but literal. It recognizes the set a^*b^*.

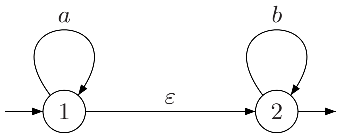


Figure 1.10. A literal automaton for the set a^*b^*.

An automaton is *deterministic* if it is synchronous, it has a unique initial state, and if, for each state p and each letter a, there is at most one edge which starts at p and is labeled by a. The end state of the edge is denoted by $p \cdot a$. Clearly, a deterministic automaton is unambiguous. For any word w, there is at most one path starting in p and labeled w. The end state of this is denoted $p \cdot w$. Clearly, for any state p and any words u, v, one has

$$p \cdot uv = (p \cdot u) \cdot v$$

provided the paths exist.

An automaton is *complete* if for any state p and any letter a there exists an edge starting in p and labelled with a. Any automaton can be *completed*, that is transformed into a complete automaton by adding one state (frequently called the sink) and by adding transitions to this state whenever they do no exist in the original automaton.

EXAMPLE 1.3.5. The automaton given in Figure 1.11 is deterministic. It recognizes the set of words having no occurrence of the factor aa. It is frequently called the *Golden mean* automaton, because the number of words of length n it recognizes is the Fibonacci number F_n (with the convention $F_0 = 0$ and $F_1 = 1$).

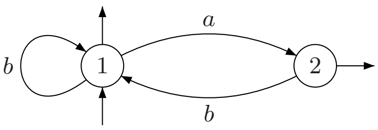


Figure 1.11. The golden mean automaton.

An automaton is *finite* if its set of states is finite. Since the alphabet is usually assumed to be finite, this means that the set of edges is finite.

A set of words $\mathcal{X}$ over $\mathcal{A}$ is *recognizable* if it can be recognized by a finite automaton.

The implementation of a deterministic automaton with a finite set of states Q, and over finite alphabet $\mathcal{A}$, uses the *next-state* function which is the partial function $\text{Next}(p, a) = p \cdot a$. In practice, the states are identified with integers, and the next-state function is given either by an array or by a set of edges (a, q) for each state p. The set may be either hashed, or listed, or represented in some balanced tree structure. Other representations exist with the aim of reducing the space while preserving the efficiency of the access.

The next-state function is extended to a function again called Next and defined by $\text{Next}(p, w) = p \cdot w$, for a word w. A practical implementation has to choose a convenient way to represent the case where the function is undefined.

```
Next(p, w)
▷ w has length n
for i ← 0 to n − 1 do
      p ← Next(p, w[i])
      if p is undefined then
            break
return p
```

Example 1.3.6. For the Golden mean automaton, the next-state function is represented by the following table (observe that $2 \cdot a$ is undefined)

	a	b
1	2	1
2		1

For the implementation of nondeterministic automata, we restrict ourselves to the case of a literal automaton which is the most frequent one. For each state, the set of outgoing edges is represented by sets $\text{Next}(p, a)$ for each letter a, and

NEXT(p, ε) for the ε-transitions. By definition NEXT$(p, a) = \{q \mid (p, a, q) \in E\}$, and NEXT$(p, \varepsilon) = \{q \mid (p, \varepsilon, q) \in E\}$, where E denotes the set of edges. We denote by INITIAL the set of initial states, and by TERMINAL the set of terminal states.

In order to check whether a word is accepted by a nondeterministic automaton, one performs a search in the graph controlled by the word to be processed. We treat this search in a breadth-first manner in the sense that, for each prefix p of the word, we compute the set of states reachable by p.

For this, we start with the implementation of the next-state function for a set of states. We give a function NEXT(S, a) that computes the set of states reachable from a state in S by a path consisting of an edge labelled by the letter a followed by a path labelled ε. An other possible choice consists in grouping the ε-transitions before the edge labelled by a. This will be seen in the treatment of the computation of a word.

```
NEXT(S, a)
  ▷ S is a set of states, and a is a letter
  T ← ∅
  for q ∈ S do
        T ← T ∪ NEXT(q, a)
  return CLOSURE(T)
```

The function CLOSURE(T) computes the set of states accessible from states in T by paths labelled ε. This is just a search in a graph, and it can be performed either depth-first or breadth-first. The time complexity of the function NEXT(S, a) is $O(d \cdot \mathrm{Card}(S))$, where d is the maximal out-degree of a state.

The function NEXT() extends to words as follows.

```
NEXT(S, w)
  ▷ S is a set of states, and w is a word of length n
  T ← CLOSURE(S)
  for i ← 0 to n − 1 do
        T ← NEXT(T, w[i])
  return T
```

In order to test whether a word w is accepted by an automaton, it suffices to compute the set $S = $ NEXT(INITIAL$, w)$, and to check whether S meets the set of final states. This is done by the following function.

```
ISACCEPTED(w)
  ▷ S is a set of states
  S ← NEXT(INITIAL, w)
  return S ∩ TERMINAL ≠ ∅
```

The time complexity of the function ACCEPT(w) is $O(nmd)$, where m is the number of states and d is the maximal out-degree of a state. Thus, in all cases, the time complexity is $O(nm^2)$.

1.3.3. Determinization algorithm

Instead of exploring a nondeterministic automaton, one may compute an equivalent deterministic automaton and perform the acceptance test on the resulting deterministic automaton. This preprocessing is especially interesting when the same automaton is going to be used on several inputs. However, the size of the deterministic automaton may be exponential in the size of the original, non deterministic one, and the direct search may be the unique realistic option.

We now show how to compute an equivalent deterministic automaton. The states of the deterministic automaton are sets of states, namely the sets computed by the function Next(). A practical implementation of the algorithm will use an appropriate data structure for a collection of sets of states. This can be a linked list or an array of sets. We only need to be able to add elements, and to test membership.

The function Explore() consists essentially in searching, in the automaton $\mathfrak{B}$ under construction, the states that are accessible. As for any exploration, several strategies are possible. We use a depth-first search realized by recursive calls of the function Explore().

```
Explore(𝒯, S, 𝔅)
▷ 𝒯 is a collection of sets of states of 𝔄
▷ 𝒯 is also the set of states of 𝔅
▷ S is an element of 𝒯
for each letter c do
    U ← Next_𝔄(S, c)
    Next_𝔅(S, c) ← U
    if U ≠ ∅ and U ∉ 𝒯 then
        𝒯 ← 𝒯 ∪ U
        (𝒯, 𝔅) ← Explore(𝒯, U, 𝔅)
return (𝒯, 𝔅)
```

We can now write the determinization algorithm.

```
NFAtoDFA(𝔄)
▷ 𝔄 is a nondeterministic automaton
𝔅 ← NewDFA()
I ← Closure(Initial_𝔄)
Initial_𝔅 ← I
▷ 𝒯 is a collection of sets of states of 𝔄
𝒯 ← I
(𝒯, 𝔅) ← Explore(𝒯, I, 𝔅)
Terminal_𝔅 ← {U ∈ 𝒯 | U ∩ Terminal_𝔄 ≠ ∅}
return 𝔅
```

The result of Algorithm NFAtoDFA is the deterministic automaton $\mathfrak{B}$. Its set of states is the set $\mathcal{T}$. In practice, it can be represented by a set of integers coding the elements of $\mathcal{T}$, as the collection $\mathcal{T}$ itself is not needed any more. The complexity of Algorithm NFAtoDFA is proportional to the size of the resulting deterministic automaton times the complexity of testing membership in line 7.

Example 1.3.7. We show in a first example the computation of a deterministic automaton equivalent to a nondeterministic one. We start with the automaton $\mathfrak{A}$ given in Figure 1.12. We have $\text{Initial}_{\mathfrak{A}} = \{1, 2\}$ and $\text{Terminal}_{\mathfrak{A}} = \{1\}$.

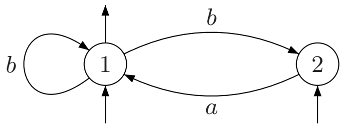


Figure 1.12. The nondeterministic automaton $\mathfrak{A}$.

The next-state function is given by the following table

	a	b
1	$\emptyset$	$\{1, 2\}$
2	$\{1\}$	$\emptyset$

The collection T of sets of states of the resulting automaton computed by Algorithm NFAtoDFA is $T = \{\{1, 2\}, \{1\}\}$. The automaton is represented in Figure 1.13. It is actually the Golden mean automaton of Example 1.3.5.

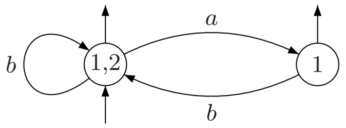


Figure 1.13. The deterministic version $\mathfrak{B}$ of $\mathfrak{A}$.

Example 1.3.8. As a second example, we consider the automaton $\mathfrak{A}$ of Example 1.3.4. We have $\text{Initial}_{\mathfrak{A}} = \{1\}$, and $\text{Closure}(\text{Initial}_{\mathfrak{A}}) = \{1, 2\}$. The resulting deterministic automaton is given in Figure 1.14

Example 1.3.9. For any set $\mathcal{K}$ of words, let $F(\mathcal{K})$ denote the set of factors of the words in $\mathcal{K}$. We are going to verify a formula involving the shuffle of two sets of words. Formally, the *shuffle operator* $\sqcup\!\sqcup$ is defined inductively on words by $u \sqcup\!\sqcup \epsilon = \epsilon \sqcup\!\sqcup u = u$ and

$$ua \sqcup\!\sqcup vb = \begin{cases} (u \sqcup\!\sqcup v)a & \text{if } a = b \\ (ua \sqcup\!\sqcup v)b \cup (u \sqcup\!\sqcup vb)a & \text{otherwise.} \end{cases}$$

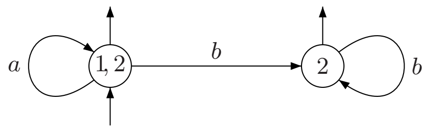


Figure 1.14. A deterministic automaton for the set a^*b^*.

The shuffle of two sets is the union of the shuffles of the words in the sets.

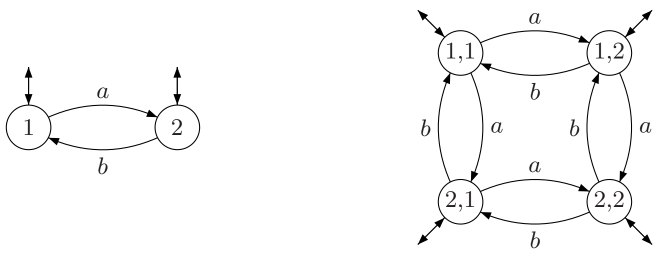
The formula is the following

$$F((ab)^*) \sqcup\!\sqcup F((ab)^*) = F((ab + ba)^*)\,. \tag{1.3.1}$$

This equality is the basis of a card trick known as Gilbreath's card trick (see Notes).

In order to prove this formula, we apply a general principle that is valid for regular sets. It consists in computing deterministic automata for each side of the equation and to check that they are equivalent.

The set $F((ab)^*)$ is recognized by the automaton on the left of Figure 1.15. It is easy to see that the set $F((ab)^*) \sqcup\!\sqcup F((ab)^*)$ is recognized by the nondeterministic automaton on the right of Figure 1.15, realized by forming pairs of states of the first automaton with action on either component.



(a) $F((ab)^*)$

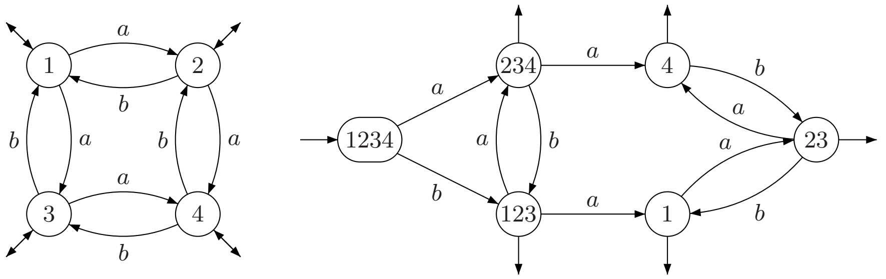
(b) $F((ab)^*) \sqcup\!\sqcup F((ab)^*)$

Figure 1.15. Two automata, recognizing $F((ab)^*)$ and $F((ab)^*) \sqcup\!\sqcup F((ab)^*)$.

To compute a deterministic automaton, we first renumber the states as indicated on the left of Figure 1.16. The result of the determinization is shown on the right.

Next, an automaton recognizing $(ab+ba)^*$ is shown on the left of Figure 1.17.

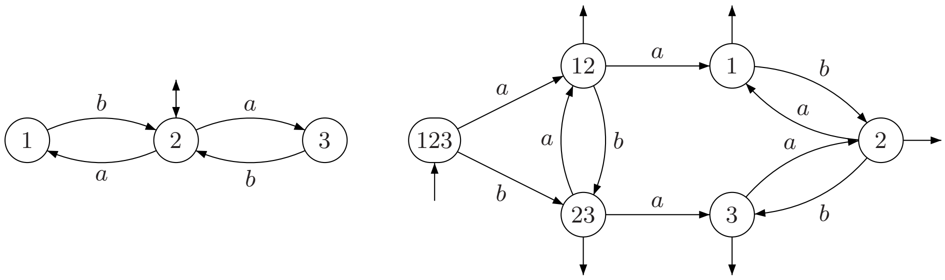
To recognize the set $F((ab + ba)^*)$, we make all states initial and terminal in this automaton. The determinization algorithm is then applied with the new initial state $\{1, 2, 3\}$. The result is shown on the right of Figure 1.17. This automaton is clearly equivalent to the automaton of Figure 1.16. This proves Formula 1.3.1.



(a) After renaming states.

(b) Deterministic automaton

Figure 1.16. On the right, a deterministic automaton recognizing the set $F((ab)^*) ш F((ab)^*)$ which is recognized by the automaton on the left.



(a) An automaton recognizing the set $F((ab+ba)^*)$.

(b) A deterministic automaton for this set.

Figure 1.17. Two automata recognizing the set $F((ab+ba)^*)$.

1.3.4. Minimization algorithms

A given regular language $\mathcal{S} \subset \mathcal{A}^*$ may be recognized by several different automata. There is however a unique one with a minimal number of states, called the *minimal automaton* of S. We will give a description of the minimal automaton and several algorithms allowing to compute it.

The abstract definition is quite simple: the states are the nonempty sets of the form $x^{-1}\mathcal{S}$ for $x \in \mathcal{A}^*$ where

$$x^{-1}\mathcal{S} = \{y \in \mathcal{A}^* \mid xy \in \mathcal{S}\}\,.$$

The initial state is the set $\mathcal{S}$ itself (corresponding to $x = \varepsilon$) and the final states are the sets $x^{-1}\mathcal{S}$ with $x \in \mathcal{S}$ (or, equivalently, such that $\varepsilon \in x^{-1}\mathcal{S}$). There is a transition from the state $x^{-1}\mathcal{S}$ by letter $a \in \mathcal{A}$ to the state $(xa)^{-1}\mathcal{S}$.

Example 1.3.10. Let us consider the set $\mathcal{S}_n$ of words over $\mathcal{A} = \{a, b\}$ that have a symbol a at the $(n+1)$th position before the end for some $n \geq 0$.

Formally, $\mathcal{S}_n = \mathcal{A}^* a \mathcal{A}^n$. For any $x = a_0 a_1 \cdots a_m \in \mathcal{A}^*$, one has

$$x^{-1}\mathcal{S}_n = \cup_{i \in P(x)} \mathcal{A}^{n-i}$$

where $P(x) = \{i \mid 0 \le i \le n \text{ and } a_{m-i} = a\}$. Thus the minimal automaton of $\mathcal{S}_n$ has 2^{n+1} states since its set of states is the set of all subsets of $\{0, , 1, \ldots, n\}$. The set $\mathcal{S}$ is also recognized by the nondeterministic automaton of Figure 1.18. This example shows that the size of the minimal automaton can be exponential,

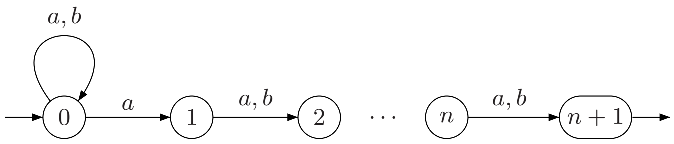


Figure 1.18. Recognizing the words which have the letter a at the $n+1$-th position before the end.

compared with the size of a nondeterministic one.

A general method for computing the minimal automaton consists in three steps.

(i) Compute a nondeterministic automaton (e.g. by the method explained in the next section),

(ii) Apply the determinization algorithm of the preceding section 1.3.3 and remove all states that are not accessible or coaccessible. The resulting automaton is deterministic and trim.

(ii) Apply a minimization algorithm, as described below.

To minimize a deterministic automaton, one uses a sequence of refinements of equivalence relations $\pi_0 \ge \pi_1 \ge \cdots \ge \pi_n$ in such a way that the classes of π_n are the states of the minimal automaton.

The equivalence relation π_n is called the *Nerode equivalence* of the automaton. It is characterized by

$$p \sim q \text{ if and only if } \mathcal{L}_p = \mathcal{L}_q\,,$$

where $\mathcal{L}_p$ is the set of words recognized by the automaton with initial state p.

The sequence starts with the partition π_0 in two classes separating the terminal states from the other ones. Further, one has $p \equiv q \bmod \pi_{k+1}$ if and only if

$$p \equiv q \bmod \pi_k \text{ and } p \cdot a \equiv q \cdot a \bmod \pi_k \text{ for all } a \in A.$$

In the above condition, it is understood that $p \cdot a = \emptyset$ if and only $q \cdot a = \emptyset$. A partition of a set with n elements can be simply represented by a function assigning to each element x its class $c(x)$.

The computation of the final partition is realized by the following algorithm known as Moore's algorithm.

```
MooreMinimization()
f ← InitialPartition()
do  e ← f
      ▷ e is the old partition, f is the new one
      f ← Refine(f)
while e ≠ f
return e
```

The refinement is realized by the following function in which we denote by $a^{-1}e$ the equivalence $p \equiv q \bmod a^{-1}e$ if and only if $p \cdot a \equiv q \cdot a \bmod e$. Again, it is understood that $p \cdot a$ is defined if and only if $q \cdot a$ is defined.

```
Refine(e)
for a ∈ 𝒜 do
      g ← a⁻¹e
      e ← Intersection(e, g)
return e
```

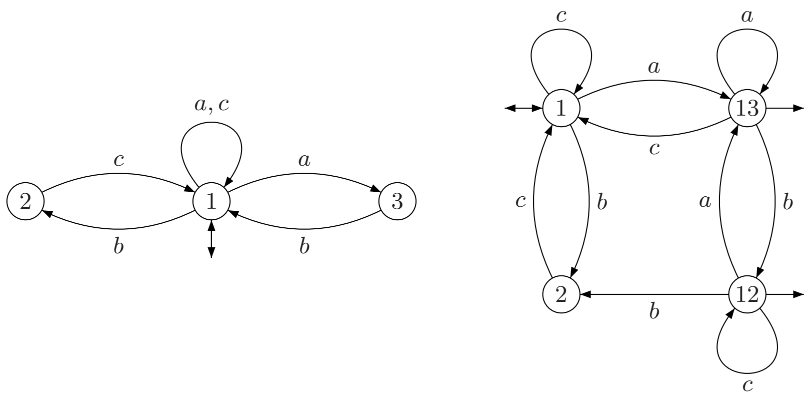
The computation of the intersection of two equivalence relations on a n-element set can be done in time $O(n^2)$ by brute force. A refinement using a radix sort of the pairs of classes improves the running time to $O(n)$. Thus, the function Refine() runs in time $O(nk)$ on an automaton with n states on an alphabet with k symbols. The loop in the function Partition() is executed at most n times since the sequence of successive partitions is strictly decreasing. Moore's algorithm itself thus computes in time $O(n^2k)$ the minimal automaton equivalent to a given automaton with n states and k letters.

Example 1.3.11. Let us consider the set $\mathcal{S} = (a + bc + ab + c)^*$. A nondeterministic automaton recognizing $\mathcal{S}$ is represented on the left of Figure 1.19. The determinization algorithm produces the automaton on the right of the figure.

Applying a renumbering of the states, we obtain the automaton on the left of Figure 1.20. The minimization procedure starts with the partition $e = \{1, 3, 4\}\{2\}$. Since $a^{-1}e = e$, the action of letter a does not refine e. On the contrary, $b^{-1}e = \{1, 4\}\{2\}\{3\}$ and thus e is refined to $f = \{1, 4\}\{2\}\{3\}$ which is found to be stable. Thus we obtain the minimal automaton represented on Figure 1.20 on the right.

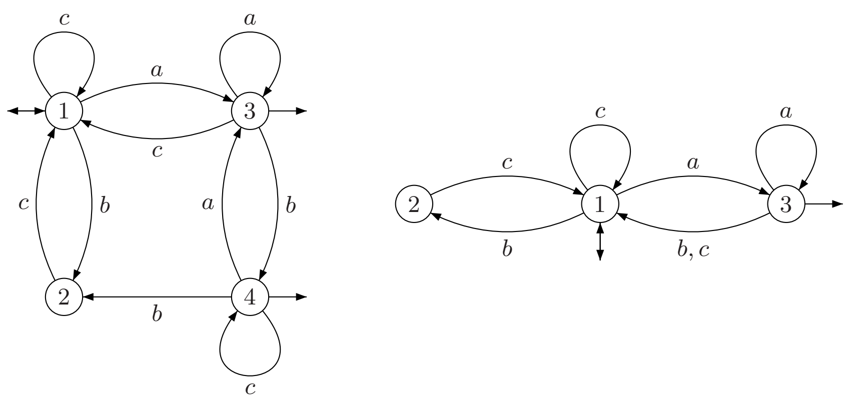
There is a more complicated but more efficient algorithm, known as Hopcroft's algorithm, which can be used to minimize deterministic automata. We assume that the automaton is complete.

The idea is to replace the global operation of intersection of two partitions by the refinement of a partition by a single block. Let P be a set of states, and let a be a letter. Let $a^{-1}P = \{q \mid q \cdot a \in P\}$. A set B of states is *refined* into B' and B'' by the pair (P, a) if the sets $B' = B \cap a^{-1}P$ and $B'' = B \setminus B'$ are both non empty. Otherwise, B is said to be *stable* by the pair (P, a).



(a) A nondeterministic automaton. (b) The determinized version.

Figure 1.19. Recognizing the set $(a + bc + ab + c)^*$



(a) Renaming the states. (b) The minimal automaton.

Figure 1.20. The minimization algorithm

The algorithm starts with the partition composed of the set T of terminal states and its complement T^c. It maintains a set S of pairs (P, a) formed of a set of states and a letter.

The main loop consists in selecting a pair (P, a) from the set S. Then for each block B of the current partition which is refined by (P, a) into B', B'', one performs the following steps

1. replace B by B' and B'' in the current partition,
2. for each letter b,
   (a) if (B, b) is in S, then replace (B, b) by (B', b) and (B'', b) in S,

(b) otherwise add to S the pair (C, b) where C is the smaller of the sets B' and B''.

If, instead of choosing the smaller of the sets B' and B'', one adds both sets (B', b) and (B'', b) to S, the algorithm becomes a complicated version of Moore's algorithm. The reason why one may dispense with one of the two sets is that when a block B is stable by (P, a) and when P is partitioned into P' and P'', then the refinement of B by (P', a) is the same as the refinement by (P'', a). The choice of the smaller one is the essential ingredient to the improvement of the time complexity from $O(n^2)$ to $O(n \log n)$.

This is described in Algorithm HopcroftMinimization() below.

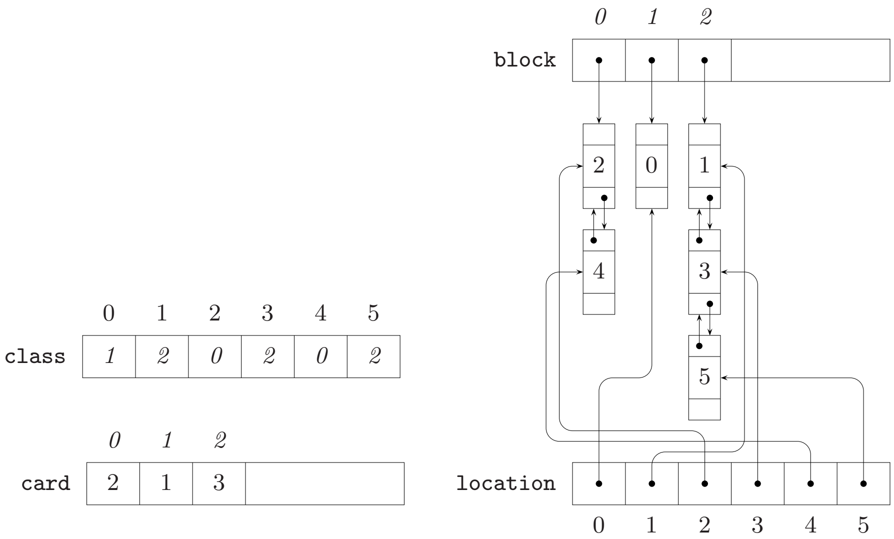
```
HopcroftMinimization()
e ← {T, T^c}
C ← the smaller of T and T^c
for a ∈ 𝒜 do
    Add((C, a), S)
while S ≠ ∅ do
    (P, a) ← First(S)
    for B ∈ e such that B is refined by (P, a) do
        B', B'' ← Refine(B, P, a)
        BreakBlock(B, B', B'', e)
        ▷ breaks B into B', B'' in the partition e
        Update(S, B, B', B'')
```

where Update() is the function that updates the set of pairs used to refine the partition, defined as follows.

```
Update(S, B, B', B'')
C ← the smaller of B' and B''
for b ∈ 𝒜 do
    if (B, b) ∈ S then
        Replace((B, b), S, (B', b), (B'', b))
    else Add((C, b), S)
```

A careful implementation of the algorithm leads to a time complexity in $O(kn \log n)$ on an automaton with n states over k letters. One of the key points is the implementation of the function BreakBlock(B, B', B'', e) which has to be implemented so as to run in time $O(\mathrm{Card}(B))$. The function actually replaces B by $B \setminus B'$ and adds a new block B'. For this, one traverses B (in linear time) and removes each element which is in B' from B in constant time and adds it to the new block, also in constant time.

The states of a class are represented by a doubly linked list, one list for each class of the partition. This representation allows to remove the element from the list, and so also from the class, in constant time. An array of pointers



(a) The classes and their size (b) The blocks of the partition

Figure 1.21. A partition of $Q = \{0, \ldots, 5\}$. The class of a state is the integer in the array `class`. The size of a class is given in the array `card`. The elements of a block are chained in a doubly linked list pointed to by the entry in the array `block`. Each cell in these lists can be retrieved in constant time by its state using the pointer in the array `location`.

indexed by the states allows to retrieve the location of a state in its block in the partition.

In order to be able to check whether a block B is refined by a pair (P, a), one maintains an array that counts, for each block B the number of states of $a^{-1}P$ that are found to be in B. The test whether B is actually refined consists in checking whether this number is both nonzero and strictly less than $\operatorname{Card} B$. This requires to maintain a table containing the number of elements of the blocks in the current partition.

To summarize, an arbitrary deterministic finite automaton with n states can be minimized in time $O(n \log n)$.

A trim automaton recognizing a *finite set* of words can be minimized in linear time with respect to the size of the automaton. Let $\mathfrak{A}$ be a finite automaton with set of states Q recognizing a finite set of words. Since the automaton is trim, it is acyclic. Thus we are faced again with DAWG's already seen in Section 1.3.1.

The *height* $h(q)$ of a state q is the length of the longest path in $\mathfrak{A}$ starting in q. Equivalently, it is the length of a longest word in the language $\mathcal{L}_q$ of words recognized by the automaton with initial state q. Of course, for any edge (p, a, q) one has $h(p) > h(q)$. Since the automaton is trim, its initial state is the unique state of maximal height. The heights satisfy the formula

$$h(p) = \begin{cases} 0 & \text{if } p \text{ has no outgoing edge,} \\ 1 + \max_{(p,a,q)} h(q) & \text{otherwise.} \end{cases}$$

In the second case, the maximum is taken over all edges starting in p. Observe that this formula leads to an effective algorithm for computing heights because the automaton has no cycle.

The parameters in the algorithm are the number n of states of $\mathfrak{A}$, the number m of transitions, and the size k of the underlying alphabet. Of course, $m \leq n \cdot k$. In practical situations like large dictionaries, the number m is much smaller than the product. As we will see, the minimization algorithm can beimplemented in time $O(n + m + k)$.

A word about the representation of $\mathfrak{A}$. Since there are only few edges, a convenient representation is to have, for each state p, a list of outgoing edges, each represented by the pair (a, q) such that (p, a, q) is a transition. States are numbered, so traversal, marking, copying, sorting is done by integers. Also, terminal states are represented in such a way that one knows in constant time whether a state is terminal.

It is easily seen that two states q and q' can be merged into a single state in the minimal automaton only if they have the same height. Therefore, the Nerode equivalence is a refinement of the partition into states of equal height.

Recall that the Nerode equivalence is defined by

$$p \sim q \text{ if and only if } \mathcal{L}_p = \mathcal{L}_q\,.$$

Recall also that

$$p \sim q \text{ if and only if } (p \in T \Leftrightarrow q \in T) \text{ and } p \cdot a \sim q \cdot a \text{ for all } a \in \mathcal{A} \tag{1.3.2}$$

This formula shows that if the equivalence is known for all states up to some height $h - 1$, it can be computed, by this formula, for states of height h. To describe this in more detail, we associate, to each state q, a sequence of data called it *signature*. It has the form

$$\sigma(q) = (s, a_1, \nu(q_1), a_2, \nu(q_2), \ldots, a_r, \nu(q_r))$$

where $s = 0$ if q is a nonterminal state and $s = 1$ if q is a terminal state, where $(q, a_1, q_1), \ldots, (q, a_r, q_r)$ are the edges starting in q, and where $\nu(p)$ is the class of the state p. We consider that classes of states are represented by integers. We assume moreover that $a_1, \ldots, a_r$ are in increasing order. This can be realized by a bucket sort in time $O(n + m + k)$.

Then Equation 1.3.2 means that

$$p \sim q \text{ if and only if } \sigma(p) = \sigma(q)\,.$$

Thus, a signature is a sequence of integers of length at most $1+2k$, $(k = \operatorname{Card} \mathcal{A})$ and each element in this sequence has a value bounded by $\max(2, k, n)$. Observe that the sum of the lengths of all signatures is bounded by $2m+n$, where m is the number of transitions. In fact, the signature of state p merely a representation of the transitions in the minimal automaton starting in the state $\nu(p)$.

For computing the Nerode equivalence of the set Q_h of states of height h, one computes the set of signatures of states in Q_h. This set is sorted by a radix sort according to their signatures, viewed as vectors over integers. Then states with equal signatures are consecutive in the sorted list and the test $\sigma(p) = \sigma(q)$ for equivalence can be done in linear time.

Here is the algorithm

AcyclicMinimization()
1 $\triangleright$ $\nu[p]$ is the state corresponding to p in the minimal automaton
2 $(Q_0, \ldots, Q_H) \leftarrow$ PartitionByHeight(Q)
3 **for** p in Q_0 **do**
4     $\nu[p] \leftarrow 0$
5 $k \leftarrow 0$
6 **for** $h \leftarrow 1$ **to** H **do**
7     $S \leftarrow$ Signatures(Q_h, ν)
8     $P \leftarrow$ RadixSort(Q_h, S) $\triangleright$ P is the sorted sequence Q_h
9     $p \leftarrow$ first state in P
10     $\nu[p] \leftarrow k$
11     $k \leftarrow k+1$
12   **for** each q in $P \setminus p$ in increasing order **do**
13   **if** $\sigma(q) = \sigma(p)$ **then**
14             $\nu[q] \leftarrow \nu[p]$
15   **else** $\nu[q] \leftarrow k$
16             $(k, p) \leftarrow (k+1, q)$
17 **return** ν

A usual topological sort can implement PartitionByHeight(Q) in time $O(n+m)$.

Each signature is then computed in time proportional to its size, so the whole set of signatures is computed in time $O(n+m)$. Each radix sort can be done in time proportional to the sum of the sizes of the signatures, with an overhead of one $O(k)$ initialization of the buckets. So the total time for the sort is also $O(n+m+k)$.

Observe that the test at line 13 is linear in the length of the signatures, so the whole algorithm is in time $O(k+n+m)$.

Example 1.3.12. Consider the automaton of Figure 1.22. The computation of the heights gives the follow partition:

$$Q_0 = \{4, 8\},\ Q_1 = \{3, 7\},\ Q_2 = \{2, 6, 10, 11\},\ Q_3 = \{1, 5, 9\},\ Q_4 = \{0\}\,.$$

States of height 0 are always final states, and are merged into a class numbered *0.*

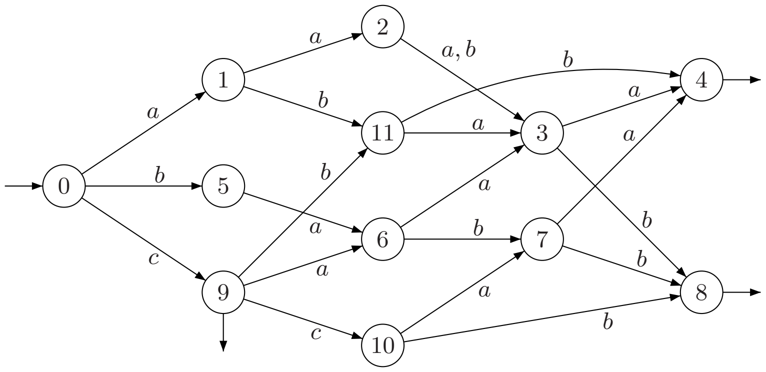


Figure 1.22. A trim automaton recognizing a finite set.

3 : 0*a0b0*
7 : 0*a0b0*

(a) Signatures of states of height 1.

	0	1	2	3	4	5	6	7	8	9	10	11
ν				*1*	*0*			*1*	*0*			

(b) The corresponding states of the minimal automaton.

The states of height 1 have the signatures given below. Observe that in a signature, the next state appearing in an edge is replaced by its class. This can be done because the algorithm works by increasing height. These states are merged into a class numbered *1*.

The radix sort of the four states of height 2 gives the sequence $(10, 11, 2, 6)$, so $10, 11$ are grouped into a class *2* and $2, 6$ are grouped into a class *3*.

2 : 0*a1b1*
6 : 0*a1b1*
10 : 0*a1b0*
11 : 0*a1b0*

(c) Signatures of states of height 2.

	0	1	2	3	4	5	6	7	8	9	10	11
ν			*3*	*1*	*0*		*3*	*1*	*0*		*2*	*2*

(d) The corresponding states of the minimal automaton.

The states of height 3 all give singleton classes, because the signatures are different. This is already clear because they have distinct lengths. In other term, a refinement of the algorithm could consist in partitioning the states of same height into subclasses according to their *width*, that is the number of edges starting in each state.

Thus, the minimal automaton has 8 states. It is given in Figure 1.23.

$1 : 0a3b2$
$5 : 0a3$
$9 : 1a3b2c2$

(e) Signatures of states of height 3

	0	1	2	3	4	5	6	7	8	9	10	11
ν	*7*	*5*	*3*	*1*	*0*	*4*	*3*	*1*	*0*	*6*	*2*	*2*

(f) The final state vector of the minimal automaton.

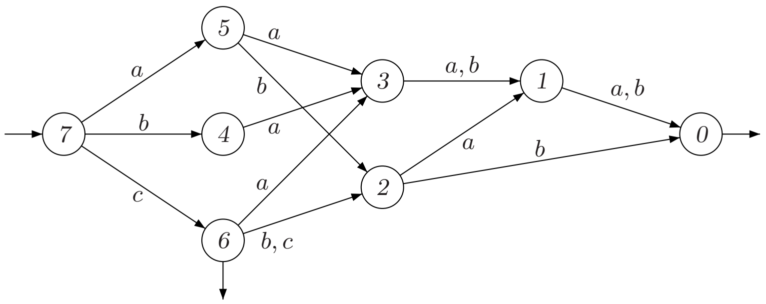


Figure 1.23. The corresponding minimal automaton.

1.4. Pattern matching

The specification of simple patterns on words uses the notion of a *regular expression.* It is an expression build using letters and a symbol representing the empty word, and three operators:

- *union*, denoted by the symbol '+',
- *product*, denoted by mere concatenation,
- *star* denoted by '*'.

These operators are used to denote the usual operations on sets of words. The operations are the set union, set product

$$\mathcal{X}\mathcal{Y} = \{xy \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$$

and the star operation

$$\mathcal{X}^* = \{x_1 \cdots x_n \mid n \geq 0, x_1, \ldots, x_n \in \mathcal{X}\}\,.$$

A regular expression defines a set of words $W(e)$, by using recursively the operations of union, product and star.

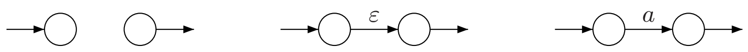
$$W(e+f) = W(e) \cup W(f),\ W(ef) = W(e)W(f),\ W(e^*) = W(e)^*\,.$$

Words in $W(e)$ are said to *match* the expression e. The problem of checking whether a word matches a regular expression is called a *pattern matching problem.*

For instance, $e = (a+b)^*abaab(a+b)^*$ is a regular expression. The set $W(e)$ is the set of words on $\mathcal{A} = \{a, b\}$ having $abaab$ as a factor. More generally, for any word w, the words matching the regular expression $\mathcal{A}^*w\mathcal{A}^*$ are those having w as a factor. Thus, the problem of checking whether a word is a factor of another is a particular case of a pattern matching problem. The same holds for subwords.

For each regular expression e, there exists a finite automaton recognizing the set of words $W(e)$. In other terms, $W(e)$ is a regular set. A proof of this assertion consists in an algorithm for building such a finite automaton, inductively on the structure of the expression. Several constructions exist that use slightly different normalizations of automata or of expressions. The main variations concern the use of ε-transitions. We present below a construction which makes extensive use of ε-transitions. The main advantage is its simplicity, and the small size of the resulting automaton.

One starts with simple automata recognizing respectively ε and a, for any letter a. They are represented in Figure 1.24. One further uses a recursive con-



(a) Empty set. (b) Empty word. (c) Letter a.

Figure 1.24. Automata for the empty set, for the empty word, and for a letter.

struction on automata with three constructs implementing union product and star. The construction is indicated below. This construction has the property to

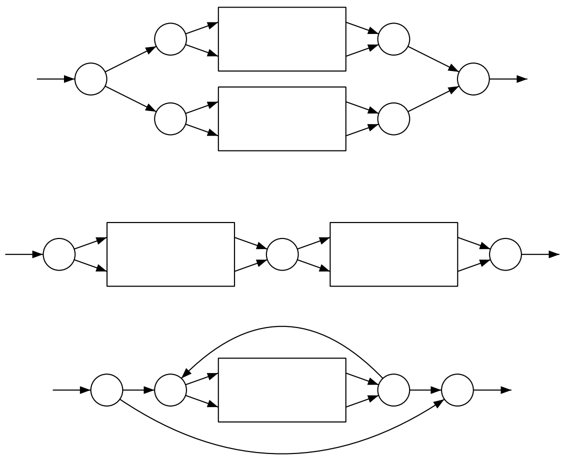


Figure 1.25. Automata for union, product and star.

construct finite automata with several particular properties. First, each state has at most two edges leaving it. If there are two edges, they have each an empty label. Also, there is a unique initial state i and a unique terminal state t. Finally, there is no edge entering i and no edge leaving t. We call such an automaton a *pattern matching automaton*.

We use a specific representation of nondeterministic automata tailored to the particular automata constructed by the algorithm. A conversion to the representation described above is straightforward. First, an automaton $\mathfrak{A}$ has a state INITIAL (the initial state) and a state TERMINAL (the terminal state). Then, there are two functions NEXT1() and NEXT2(). For each state p, NEXT1(p) = (a, q) if there is an edge (p, a, q). If there is an edge (p, ε, q), then NEXT1(p) = (ε, q). If there is a second edge (p, ε, q'), then NEXT2(p) = (ε, q'). If no edge starts from p, then NEXT(p) is undefined.

We use a function NEWAUTOMATON() to create an automaton with just one initial state and one terminal state and no edges. The function creating an automaton recognizing a is given in Algorithm AUTOMATONLETTER.

AUTOMATONLETTER(a)
$\mathfrak{A} \leftarrow$ NEWAUTOMATON()
NEXT1(INITIAL$_{\mathfrak{A}}$) $\leftarrow (a,$ TERMINAL$_{\mathfrak{A}})$
return $\mathfrak{A}$

The automata recognizing the union, the product and the star are depicted in Figure 1.25. Boxes represent automata, up to their initial and terminal state that are drawn separately. All drawn edges are ε-transitions. The implementation of the corresponding three functions AUTOMATAUNION(), AUTOMATAPRODUCT() and AUTOMATONSTAR() is straightforward.

AUTOMATAUNION($\mathfrak{A}, \mathfrak{B}$)
$\mathfrak{C} \leftarrow$ NEWAUTOMATON()
NEXT1(INITIAL$_{\mathfrak{C}}$) $\leftarrow (\varepsilon,$ INITIAL$_{\mathfrak{A}})$
NEXT2(INITIAL$_{\mathfrak{C}}$) $\leftarrow (\varepsilon,$ INITIAL$_{\mathfrak{B}})$
NEXT1(TERMINAL$_{\mathfrak{A}}$) $\leftarrow (\varepsilon,$ TERMINAL$_{\mathfrak{C}})$
NEXT1(TERMINAL$_{\mathfrak{B}}$) $\leftarrow (\varepsilon,$ TERMINAL$_{\mathfrak{C}})$
return $\mathfrak{C}$

The function AUTOMATAPRODUCT() uses a function MERGE() that merges two states into a single one.

AUTOMATAPRODUCT($\mathfrak{A}, \mathfrak{B}$)
$\mathfrak{C} \leftarrow$ NEWAUTOMATON()
INITIAL$_{\mathfrak{C}} \leftarrow$ INITIAL$_{\mathfrak{A}}$
TERMINAL$_{\mathfrak{C}} \leftarrow$ TERMINAL$_{\mathfrak{B}}$
MERGE(TERMINAL$_{\mathfrak{A}}$, INITIAL$_{\mathfrak{B}}$)
return $\mathfrak{C}$

```
AutomatonStar(𝔄)
𝔅 ← NewAutomaton()
Next1(Initial_𝔅) ← (ε, Initial_𝔄)
Next2(Initial_𝔅) ← (ε, Terminal_𝔅)
Next1(Terminal_𝔄) ← (ε, Initial_𝔄)
Next1(Terminal_𝔄) ← (ε, Terminal_𝔅)
return ℭ
```

The practical implementation of these algorithms on a regular expression is postponed to the next section. As an example, consider the automaton in Figure 1.26. It has 21 states and 27 edges. The size of the pattern matching

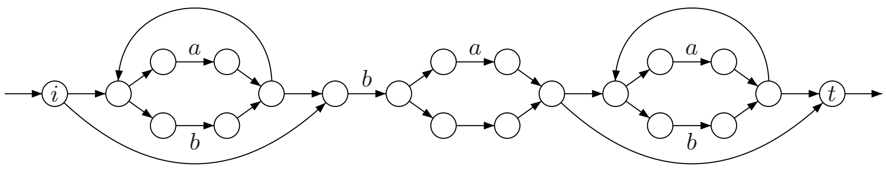


Figure 1.26. The automaton for the expression $(a + b)^*b(a + 1)(a + b)^*$.

automaton recognizing the set of words matching a regular expression is linear in the size of the expression. Indeed, denote by $n(e)$ the number of states of the pattern matching automaton corresponding to the expression e. Then

$$\begin{aligned} n(a) &= 2 \quad \text{for each letter } a \\ n(\varepsilon) &= 2 \\ n(e + f) &= n(e) + n(f) + 2 \\ n(ef) &= n(e) + n(f) - 1 \\ n(e^*) &= n(e) + 2 \end{aligned}$$

Thus $n(e) \leq 2|e|$, where $|e|$ is the length of the expression e (discarding the left and right parentheses). The number of edges is at most twice the number of states. Thus the space complexity of the resulting algorithm is linear in the size of the expression.

To realize the run of such an automaton on a word w, one uses Algorithm IsAccepted. We observe that in a pattern matching automaton, the out-degree of a state is at most 2. Therefore, the time complexity of a call IsAccepted(w) is $O(nm)$, where n is the size of the regular expression and $m = |w|$.

In some particular cases, the quadratic complexity $O(nm)$ can be replaced by $O(n + m)$. This is the case in particular for the string matching problem treated in Algorithm SearchFactor.

1.5. Transducers

Beyond formal languages, *relations* between words are a very natural concept. We consider relations over words, but most of the general notions work for relations over arbitrary sets.

Formally, a relation ρ between words over the alphabet $\mathcal{A}$ and words over the alphabet $\mathcal{B}$ is just a subset of the Cartesian product $\mathcal{A}^* \times B^*$. We call it a relation from $\mathcal{A}^*$ to $\mathcal{B}^*$. Actually, such a relation can be viewed as a partial function f_ρ from $\mathcal{A}^*$ to the set $\mathfrak{P}(\mathcal{B}^*)$ of subsets of $\mathcal{B}^*$ defined by

$$f_\rho(x) = \{y \in \mathcal{B}^* \mid (x, y] \in \rho\}, \qquad x \in \mathcal{A}^* .$$

The *inverse* of a relation σ from $\mathcal{A}^*$ to $\mathcal{B}^*$ is the relation σ^{-1} from $\mathcal{B}^*$ to $\mathcal{A}^*$ defined by

$$\sigma^{-1} = \{(v, u) \mid (u, v) \in \sigma\}.$$

The *composition* of a relation σ from $\mathcal{A}^*$ to $\mathcal{B}^*$ and a relation τ from $\mathcal{B}^*$ to $\mathcal{C}^*$ is the relation from $\mathcal{A}^*$ to $\mathcal{C}^*$ defined by $(x, z) \in \sigma \circ \tau$ if and only if there exists $y \in \mathcal{B}^*$ such that $(x, y) \in \sigma$ and $(y, z) \in \tau$. The reader should be aware that the composition of relations goes the other way round than the usual composition of functions. The function $f_{\sigma\circ\tau}$ defined by the relation $\sigma\circ\tau$ is $f_{\sigma\circ\tau}(x) = f_\tau(f_\sigma(x))$. One can overcome this unpleasant aspect by writing the function symbol on the right of the argument.

A particular case of a relation ρ from $\mathcal{A}^*$ to $\mathcal{B}^*$ is that of a partial function from $\mathcal{A}^*$ to $\mathcal{B}^*$. In this case, f_ρ is a (partial) function from $\mathcal{A}^*$ into $\mathcal{B}^*$.

Example 1.5.1. Consider the relation $\gamma \subset \mathcal{A}^* \times \mathcal{A}^*$ defined by $(x, y) \in \gamma$ if and only if x and y are conjugate. Clearly, $\gamma = \gamma^{-1}$. The image of a word x is the set of conjugates of x.

Example 1.5.2. Consider the relation $\mu \subset \mathcal{A}^* \times \mathcal{A}^*$ defined by

$$\mu = \{(a_1 a_2 \cdots a_n, a_n a_{n-1} \cdots a_1) \mid a_1, \ldots, a_n \in \mathcal{A}\} .$$

Clearly, $\mu = \mu^{-1}$ and $\mu \circ \mu$ is the identity relation.

Example 1.5.3. For the relation $\rho \subset \mathcal{A}^* \times \mathcal{A}^*$ defined by doubling each letter:

$$\rho = \{(a_1 a_2 \cdots a_n, a_1^2 a_2^2 \cdots a_n^2) \mid a_1, \ldots, a_n \in \mathcal{A}\}$$

the image of a word $x = a_1 a_2 \cdots a_n$ is $a_1^2 a_2^2 \cdots a_n^2$. The inverse is only defined on words of the form $a_1^2 a_2^2 \cdots a_n^2$.

The set of relations on words is subject to several additional operations. The *union* of two relations $\rho, \sigma \subset \mathcal{A}^* \times \mathcal{B}^*$ is the set union $\rho \cup \sigma$. The *product* of ρ and $\sigma \subset \mathcal{A}^* \times \mathcal{B}^*$ is the relation

$$\rho\sigma = \{(ur, vs) \mid (u, v) \in \rho, (r, s) \in \sigma\}.$$

The *star* of $\sigma \subset \mathcal{A}^* \times \mathcal{B}^*$ is the relation

$$\sigma^* = \{(u_1u_2\cdots u_n, v_1v_2\cdots v_n) \mid (u_i, v_i) \in \sigma, n \geq 0\}.$$

A relation from $\mathcal{A}^*$ to $\mathcal{B}^*$ is *rational* if it can be obtained from subsets of $(\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{B} \cup \{\varepsilon\})$ by a finite number of operations of union, product and star.

A rational relation that is a (partial) function is called a *rational function*.

Example 1.5.4. The doubling relation is rational since it can be written, e.g. on the alphabet $\{a, b\}$ as $((a, aa) \cup (b, bb))^*$. More generally, for any morphism f from $\mathcal{A}^*$ to $\mathcal{B}^*$, the relation $\rho = \{(x, f(x) \mid x \in \mathcal{A}^*\}$ is rational. Indeed, $\rho = (\cup_{a\in\mathcal{A}}(a, f(a)))^*$. Thus morphisms are rational functions.

Just like regular expressions correspond to automata, rational relations correspond to a kind of automata called *transducers* which are just automata with output. Formally, a *transducer* over the alphabets $\mathcal{A}$, $\mathcal{B}$ is an automaton in which the edges are elements of $Q \times \mathcal{A}^* \times \mathcal{B}^* \times Q$. Thus each edge (p, u, v, q) has an *input label* u which is a word over the alphabet $\mathcal{A}$ and an *output label* v which is a word over the output alphabet $\mathcal{B}$. The transducer is denoted (Q, E, I, T) where Q is the set of states, E the set of edges, I the set of initial states and T the set of final states.

There are two "ordinary" automata corresponding to a given transducer. The *input automaton* is obtained by using only the input label of each edge. The *output automaton* is obtained by using only the output labels.

The terminology introduced for automata extends naturally to transducers. In particular, a path is labeled by a pair (x, y) formed of its input label x and its output label y. Such a path from p to q is often denoted $p \xrightarrow{x|y} q$. Just as a finite automaton recognizes a set of words, a transducer recognizes or *realizes* a relation. The algorithm of Section 1.4 can be easily adapted to build a transducer corresponding to a given rational relation.

As for automata, we allow in the definition of transducers the input and output labels to be arbitrary, possibly empty, words. The behavior of the transducer can be viewed as a machine reading an input word and writing an output word through two "heads" (see Figure 1.27). The mechanism is asynchronous in the sense that the two heads may move at different speeds.

The particular case of *synchronous* transducers is important. A transducer is said to be synchronous if for each edge, the input label and the output label are letters. Not every rational relation can be realized by a synchronous transducer. Indeed, if ρ is realized by a synchronous transducer, then ρ is length-preserving. This means that whenever $(x, y) \in \rho$, then $|x| = |y|$.

A transducer is *literal* if for each edge the input label and the output label are letters or the empty word. It is not difficult to show that any transducer can be replaced by a literal one.

Example 1.5.5. The relation between a word written in lower-case letters $a, b, c, \ldots$ and the corresponding upper-case letters $\mathcal{A}, B, C, \ldots$ is rational. Indeed, it is described by the expression $((a, A) \cup (b, B) \cup \ldots)^*$. This relation is

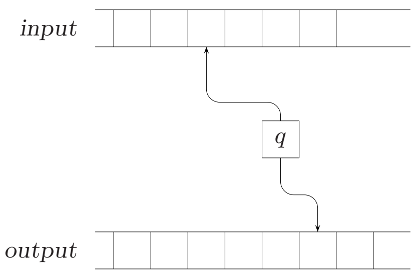


Figure 1.27. A transducer reads the input and writes the output.

realized by the transducer of Figure 1.28. This transducer is both literal and

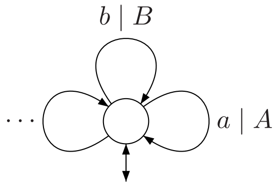


Figure 1.28. From lower case to upper case.

synchronous.

Example 1.5.6. The Fibonacci morphism defined by $a \to ab$, $b \to a$ is realized by the transducer on the left of Figure 1.29. The transducer on the right of

Figure 1.29. The Fibonacci morphism.

Figure 1.29 realizes the same morphism. It is literal.

Example 1.5.7. The transducer represented on the left of Figure 1.30 realizes the *circular right shift* on a word on the alphabet $\{a, b\}$ ending with letter a. The transformation consists in shifting cyclically each symbol one place to the right. For example

$$\begin{array}{cccccc} a & b & b & a & b & a \\ a & a & b & b & a & b \end{array}$$

The restriction to words ending with letter a is for simplicity (and corresponds to the choice of state 0 as initial and final state in the automaton on the left of Figure 1.30). The inverse of the right shift is the left shift which shifts

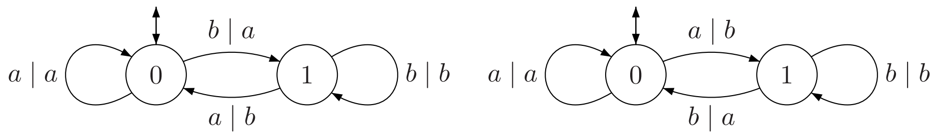


Figure 1.30. The circular right shift on words ending with a and its inverse.

all symbols cyclically one place to the left. Its restriction to words beginning with a is represented on the right of Figure 1.30. The composition of both transformations is the identity restricted to words ending with the letter a plus the empty word.

An important property of rational relations is that *the composition of two rational relations is again a rational relation*. The construction of a transducer realizing the composition is the following. We start with a transducer $\mathfrak{S} = (Q, E, I, T)$ over $\mathcal{A}$, $\mathcal{B}$ and a transducer $\mathfrak{S}' = (Q', E', I', T')$ over $\mathcal{B}$, $\mathcal{C}$. We suppose that $\mathfrak{S}$ and $\mathfrak{S}'$ are literal (actually we shall only need that the output automaton of $\mathfrak{S}$ is literal and that the input automaton of $\mathfrak{S}'$ is literal). We build a new transducer $\mathfrak{U}$ as follows. The set of states of $\mathfrak{U}$ is $Q \times Q'$. The set of edges is formed of three kinds of edges.

1. the set of edges $(p, p') \xrightarrow{a|c} (q, q')$ for all edges $p \xrightarrow{a|b} q$ in E and $p' \xrightarrow{b|c} q'$ in E'.
2. the set of edges $(p, p') \xrightarrow{\varepsilon|c} (p, q')$ for $p' \xrightarrow{\varepsilon|c} q'$ in E'.
3. the set of edges $(p, p') \xrightarrow{a|\varepsilon} (q, p')$ for $(p \xrightarrow{a|\varepsilon} q)$ in E.

The set of initial states of $\mathfrak{U}$ is $I \times I'$ and the set of terminal states is $T \times T'$. The definition of the edges implies that

$$(p, r) \xrightarrow{x|z} (q, s) \iff \exists y \ : \ p \xrightarrow{x|y} q \ \text{ and } \ r \xrightarrow{y|z} s\,.$$

This allows to prove that the composed transducer realizes the composition of the relations.

Example 1.5.8. The composition of the circular right shift of Example 1.5.7 with itself produces the circular right 2-shift which consists in cyclically shifting the letters two places to the right for words ending with aa.

For the implementation of transducers we use a function Next(p) which associates to each state p the set of edges beginning at p, and two sets Initial and Terminal to represent the initial and terminal states.

The algorithm computing the composition of two transducers is easy to write.

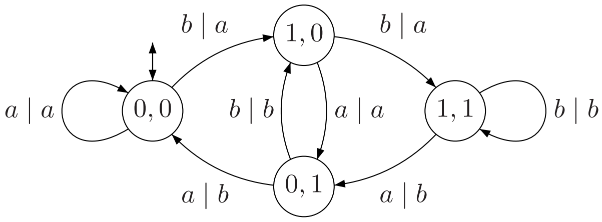


Figure 1.31. The right 2-shift.

ComposeTransducers($\mathfrak{S}, \mathfrak{T}$)

$\triangleright$ $\mathfrak{S}$ and $\mathfrak{T}$ are literal transducers
$\mathfrak{U} \leftarrow$ NewTransducer()
for each edge (p, a, b, q) of $\mathfrak{S}$ **do**
  for each edge (r, b, c, s) of $\mathfrak{T}$ **do**
    add $((p, r), a, c, (q, s))$ to the edges of $\mathfrak{U}$
for each edge (p, a, ε, q) of $\mathfrak{S}$ **do**
  for each state r of $\mathfrak{T}$ **do**
    add $((p, r), a, \varepsilon, (q, r))$ to the edges of $\mathfrak{U}$
for each edge (r, ε, c, s) of $\mathfrak{T}$ **do**
  for each state p of $\mathfrak{S}$ **do**
    add $((p, r), \varepsilon, c, (p, s))$ to the edges of $\mathfrak{U}$
Initial$_{\mathfrak{U}} \leftarrow$ Initial$_{\mathfrak{S}} \times$ Initial$_{\mathfrak{T}}$
Terminal$_{\mathfrak{U}} \leftarrow$ Terminal$_{\mathfrak{S}} \times$ Terminal$_{\mathfrak{T}}$
return $\mathfrak{U}$

The composition can be used to compute an automaton that recognizes the image of a word (and more generally of a regular set) by a rational relation. Indeed, let ρ be a rational relation from $\mathcal{A}^*$ to $\mathcal{B}^*$, let x be a word over $\mathcal{A}$. Let $\mathfrak{R}$ be a literal transducer realizing ρ, and let $\mathfrak{A}$ be a literal transducer realizing the relation $\{(\varepsilon, x)\}$. Let $\mathfrak{T} = $ Compose$(\mathfrak{A}, \mathfrak{R})$ be the composition of $\mathfrak{A}$ and $\mathfrak{R}$. The image $f(x) = \{y \in \mathcal{B}^* \mid (x, y) \in \rho\}$ is recognized by the output automaton of $\mathfrak{T}$.

Example 1.5.9. Consider the word $x = ab$ and the Fibonacci morphism of Example 1.5.6. On the left of Figure 1.32 is a transducer realizing $\{(\varepsilon, x)\}$, and on the right the transducer obtained by composing it with the literal transducer of Figure 1.29. The composition of the transducers contains actually an additional edge $(0,1) \xrightarrow{\varepsilon \mid b} (0,0)$ which is useless because the state $(0,1)$ is inaccessible from the initial state.

A *sequential transducer* over $\mathcal{A}$, $\mathcal{B}$ is a triple (Q, i, T) together with a partial function

$$Q \times \mathcal{A} \to \mathcal{B}^* \times Q$$

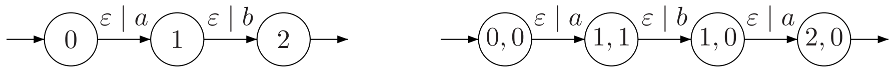


Figure 1.32. The image $f(x) = aba$ of $x = ab$ by the Fibonacci morphism.

which breaks up into a *next state* function $Q \times \mathcal{A} \to Q$ and an output function $Q \times \mathcal{A} \to \mathcal{B}^*$. As usual, the next state function is denoted $(q, a) \mapsto q \cdot a$ and the output function $(q, a) \mapsto q * a$. In addition, the initial state $i \in Q$ has attached a word λ called the *initial prefix* and T is actually a (partial) function $T : Q \to B^*$ called the *terminal function*. Thus, an initial prefix and additional suffix can be added to all outputs.

The next state and the output functions are extended to words by $p \cdot (xa) = (p \cdot x) \cdot a$ and $p * (xa) = (p * x)(p \cdot x) * a$. The second formula means that the output $p * (xa)$ is actually the product of the words $p * x$ and $q * a$ where $q = p \cdot x$. The (partial) function f from $\mathcal{A}^*$ to $\mathcal{B}^*$ *realized* by the sequential transducer is defined by $f(x) = \lambda v \tau$ where u is the initial prefix, $v = i * x$ and $\tau = T(i \cdot x)$. A function from $\mathcal{A}^*$ to $\mathcal{B}^*$ that is realized by a sequential transducer is called a *sequential function*.

Example 1.5.10. The circular left shift on words over $\{a, b\}$ beginning with a is realized, on the right of Figure 1.30, by a transducer which is not sequential (two edges with input label a leave state 0). It can also be computed by the sequential transducer of Figure 1.33 with the initial pair $(\varepsilon, 0)$ and the terminal function $T(1) = a$.

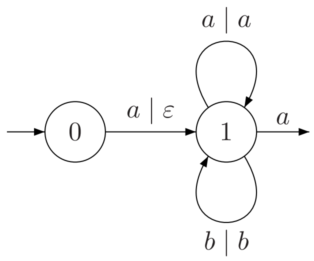


Figure 1.33. A sequential transducer for the circular left shift on words beginning with a.

The composition of two sequential functions is again a sequential function. This is actually a particular case of the composition of rational functions. The same construction is used to compose sequential transducers and it happens to produce a sequential transducer. We give explicitly the form of the composed transducer.

Let $\mathfrak{S} = (Q, i, T)$ be a sequential transducer over $\mathcal{A}$, $\mathcal{B}$ and let $\mathfrak{S}' = (Q', i', T')$ be a sequential transducer over $\mathcal{B}$, $\mathcal{C}$. The *composition* of $\mathfrak{S}$ and $\mathfrak{S}'$ is the sequential transducer $\mathfrak{S} \circ \mathfrak{S}'$ with set of states $Q' \times Q$, initial state (i', i) and terminal states $T'' = T' \times T$. Observe that we reverse the order for notational convenience. The next state function and the output function are given by

$$\begin{aligned}(p', p) \cdot x &= (p' \cdot (p * x), p \cdot x)\\(p', p) * x &= p' * (p * x)\end{aligned}$$

The initial prefix of the composed transducer is the word $\lambda'' = \lambda'(i' * \lambda)$, and the terminal function T'' is defined by

$$T''(q', q) = (q' * T(q))T'(q' \cdot T(q)) .$$

The value of the terminal function T'' on (q', q) is indeed obtained by first computing the value of the terminal function $T(q)$ and then fitting this word in the transducer $\mathfrak{S}'$ at state q'.

For the implementation of sequential transducers we use a partial function $\textsc{Next}(p, a) = (p * a, p \cdot a)$ grouping the output function and the next state function. There is also a pair $\textsc{Initial} = (\lambda, i) \in B^* \times Q$ for the initial prefix and the initial state and a partial function $\textsc{Terminal}(q)$ returning the terminal suffix for each terminal state $q \in T$.

1.5.1. Determinization of transducers

Contrary to ordinary automata, it is not true that any finite transducer is equivalent to a finite sequential one. It can be verified that a transducer is equivalent to a sequential one if and only if it realizes a partial function and if it satisfies a condition called the twinning property defined as follows. Consider a pair of paths with the same input label and of the form

$$i \xrightarrow{u|u'} q \xrightarrow{v|v'} q$$

$$i' \xrightarrow{u|u''} q' \xrightarrow{v|v''} q'$$

where i and i' are initial states. Two paths as above are called *twin*. The *twinning property*) is that for any pair of twin paths, the output is such that v', v'' are conjugate and $u'v'v' \cdots = u''v''v'' \cdots$.

Example 1.5.11. The circular right shift on all words over $\{a, b\}$ is realized by the transducer of Figure 1.34. It is not a sequential function because the last letter cannot be guessed before the end of the input. Formally, this is visible because of the twin paths

$$0 \xrightarrow{b|a} 1 \xrightarrow{ab|ba} 1$$

and

$$3 \xrightarrow{b|b} 3 \xrightarrow{ab|ba} 3.$$

with distinct outputs $ababa\cdots$ and $bbababa\cdots$.

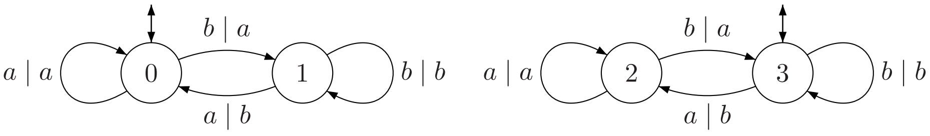


Figure 1.34. The circular right shift.

The computation of an equivalent sequential transducer is a variant of the determinization algorithm of automata. The main difference is that it may fail to terminate since, as we have seen before, it cannot be always performed successfully. We start with a transducer $\mathfrak{A}$ which is supposed to be equivalent to a sequential one. We suppose that $\mathfrak{A}$ is literal (or, at least, that its input automaton is literal) and trim. The states of the equivalent sequential transducer $\mathfrak{B}$ are sets of pairs $(u, q) \in \mathcal{B}^* \times Q$. A pair $(u, q) \in \mathcal{B}^* \times Q$ is called a *half-edge*. The states are computed by using in a first step a function $\textsc{Next}()$ represented below. The value of $\textsc{Next}(S, a)$ on a set S of half-edges and a letter a is the union, for $(u, p) \in S$ of the set of half-edges (uvw, r) such that there are, in $\mathfrak{A}$,

(i) an edge $p \xrightarrow{a|v} q$
(ii) and a path $q \xrightarrow{\varepsilon|w} r$

We use a function $\textsc{Next}_{\mathfrak{A}}(p, a)$ returning the set of half-edges (v, q) such that (p, a, v, q) is an edge of the transducer $\mathfrak{A}$.

```
Next(S, a)
▷ S is a set of half-edges (u, q) ∈ B* × Q, and a is a letter
T ← ∅
for (u, p) ∈ S do
      for (v, q) ∈ Next_𝔄(p, a) do
            T ← T ∪ (uv, q)
return Closure(T)
```

The set $\textsc{Closure}(T)$ is the set of half-edges (uw, r) such that there is a path $q \xrightarrow{\varepsilon|w} r$ in $\mathfrak{A}$ for some half-edge $(u, q) \in T$. If the transducer is equivalent to a deterministic one, this set is finite. The computation of $\textsc{Closure}(T)$ uses as usual an exploration of the graph composed of the edges of the form (q, ε, v, r). A test can be added to check that this graph has no loop whose label is nonempty word, i.e. that the set $\textsc{Closure}(T)$ is finite.

As an auxiliary step, we compute the following function

```
Lcp(U)
  ▷ U is a set of half-edges
  v ← LongestCommonPrefix(U)
  U′ ← Erase(v, U)
  return (v, U′)
```

The function LongestCommonPrefix(U) returns the longest common prefix of the words u such that there is a pair $(u, q) \in U$. The function Erase(v, U) returns the set of half-edges obtained by erasing the prefix v of the words u appearing in the half-edges $(u, q) \in U$.

In a second step, we build the set of states and the next state function of the resulting sequential transducer $\mathfrak{B}$. As for automata, we use a function Explore() which operates on the fly.

```
Explore(T, S, B)
  ▷ T is a collection of sets of half-edges
  ▷ S is an element of T
  for each letter a do
      (v, U) ← Lcp(Next(S, a))
      Next_B(S, a) ← (v, U)
      if U ≠ ∅ and U ∉ T then
          T ← T ∪ U
          (T, B) ← Explore(T, U, B)
  return (T, B)
```

We can finally write the function realizing the determinization of a transducer into a sequential one.

```
ToSequentialTransducer(A)
  ▷ A is a transducer
  B ← NewSequentialTransducer()
  I ← Closure({ε} × Initial_A)
  Initial_B ← I
  ▷ T is a collection of sets of half-edges
  T ← I
  (T, B) ← Explore(T, I, B)
  for S ∈ T do
      for (u, q) ∈ S do
          if q ∈ Terminal_A then
              Terminal_B(S) ← u
  return B
```

Example 1.5.12. The application of the determinization algorithm to the transducer on the right of Figure 1.30 produces the sequential transducer of Figure 1.33 as obtained on Figure 1.35

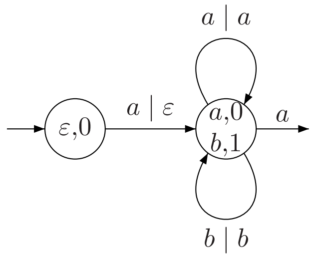


Figure 1.35. A sequential transducer for the circular left shift on words beginning with a obtained by the determinization algorithm.

A test can be added to the determinization algorithm to stop the computation in case of failure, that is if one of the folllowing situations occur, implying that the transducer $\mathfrak{A}$ is not equivalent to a sequential one. First, one may check at line 4 in algorithm Explore() that the half edges appearing in a state of $\mathfrak{B}$ have a label of bounded length. Indeed, it can be shown that there exists a constant K, depending on $\mathfrak{A}$ such that for each half-edge (u, q) appearing in a state of $\mathfrak{B}$, the length of u is bounded by K (otherwise $\mathfrak{A}$ does not satisfy the twinning property, see Problem 1.5.1). Second, a test can be added at line 10 of algorithm ToSequentialTransducer() to check that if a state of $\mathfrak{B}$ contains two half-edges (u, q) and (v, r) with q, r terminal, then $u = v$ (if this condition fails to hold, then $\mathfrak{A}$ does not realize a function).

1.5.2. Minimization of transducers

Just as there is a unique minimal deterministic automaton equivalent to a given one, there is also a unique minimal sequential transducer equivalent to a given one. The minimization of sequential transducers consists in two steps. A preliminary one, called *normalization* allows to produce output as soon as possible. The second step is quite similar to the minimization of finite automata.

Let $\mathfrak{A} = (Q, i, T)$ be a sequential transducer. For each state $p \in Q$, let us denote by $\mathcal{X}_p$ the subset of $\mathcal{B}^*$ recognized by the output automaton corresponding to $\mathfrak{A}$ with p as initial state. The normalization consists in computing for each state $p \in Q$, the longest common prefix π_p of all words in $\mathcal{X}_p$.

The normalized transducer is obtained by modifying the output function and terminal function of $\mathfrak{A}$. We set

$$\lambda' = \lambda\pi_i\,, \quad p *' a = \pi_p^{-1}(p * a)\pi_{p.a}\,, \quad T'(p) = \pi_p^{-1}T(p)$$

The computation of the words π_p can be performed as follows. It uses the binary operation associating to two words their longest common prefix. This operation is associative and commutative and will be denoted in this section by a +, like a sum. We consider the set $K = \mathcal{B}^* \cup 0$ formed of $\mathcal{B}^*$ augmented with 0 as ordered by the relation $x \le y$ if x is a prefix of y or $y = 0$. For $p, q \in Q$, we denote by $M_{p,q}$ the element of K which is the longest common prefix of all words v such that there is an edge $p \xrightarrow{a|v} q$ (and $M_{pq} = 0$ if this set is empty). We also consider the Q-vector N defined by $N_p = T(p)$, where T is the terminal function, and $N_p = 0$ if $T(p)$ is empty. For a Q-vector X of elements of K, we consider the vector $Y = MX + N$ which is defined for $p \in Q$ by

$$Y_p = \sum_{q \in Q} M_{p,q} X_q + N_p$$

Recall that all sums are in fact longest common prefixes and that the right-hand side of the equation above is the longest common prefix of the words $M_{p,q}X_q$, for $q \in Q$, and N_p. It can be checked that the function f defined by $f(X) = MX + N$ is order preserving for the partial order considered on the set K. Thus, there is a unique maximal fix-point which satisfies $X = MX + N$. This is precisely the vector of words $P = (\pi_p)$ we are looking for. It can be computed as the limit of the decreasing sequence $f^k(0)$ for $k = 1, 2, \ldots$..

Example 1.5.13. Consider the transducer realizing the Fibonacci morphism represented on the right of Figure 1.29. The determinization of this transducer produces the sequential transducer on the left of Figure 1.36. The computation

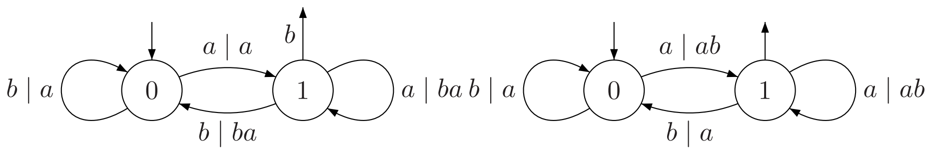


Figure 1.36. The normalization algorithm.

of the vector P uses the transformation $Y = MX + N$ with

$$\begin{aligned} Y_0 &= aX_1 + aX_0 + \varepsilon \\ Y_1 &= baX_0 + baX_1 + b \end{aligned}$$

The successive values of the vector P are $P = \begin{bmatrix} 0 \; 0 \end{bmatrix}$, $P = \begin{bmatrix} \varepsilon \; b \end{bmatrix}$. The last value satisfies $P = MP + N$ and thus it is the final one. The normalized transducer is shown on the right of Figure 1.36.

The algorithm to compute the array P is easy to write.

LongestCommonPrefixArray($\mathfrak{A}$)
▷ P, P' are arrays of strings initially null
▷ M is the matrix of transitions of $\mathfrak{A}$ and N the vector of terminals
do $P \leftarrow P'$
    $P' \leftarrow MP + N$
while $P \neq P'$
return P

The expression $MP+N$ should be evaluated using the longest common prefix for the sum, including those appearing in the product MP. The normalized transducer can now be computed by the following function.

NormalizeTransducer($\mathfrak{A}$)
$P \leftarrow$ LongestCommonPrefixArray($\mathfrak{A}$)
$(\lambda, i) \leftarrow$ Initial
Initial $\leftarrow (\lambda P[i], i)$
for $(p, a) \in Q \times A$ **do**
    $(u, q) \leftarrow$ Next(p, a)
    Next$(p, a) \leftarrow P[p]^{-1}uP[q]$
for $p \in Q$ **do**
    $T[p] \leftarrow P[p]^{-1}T[p]$

The last step of the minimization algorithm consists in applying the minimization algorithm to the input automaton, starting from the initial partition which is defined by $p \equiv q$ if $T(p) = T(q)$ and if $p * a = q * a$ for each $a \in A$. Any one of the minimization algorithms presented in Section 1.3.4 applies.

Example 1.5.14. We apply the minimization algorithm to the transducer obtained after normalization on the right of Figure 1.37. The two states are found to be equivalent. The result is the sequential transducer on the right of Fig-

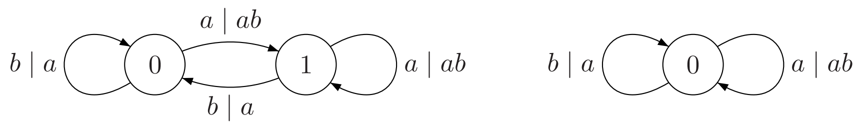


Figure 1.37. The minimization algorithm.

ure 1.37 which is of course identical to the transducer on the left of Figure 1.29.

1.6. Parsing

There are other ways, beyond regular expressions, to specify properties of words. In particular, context-free grammars offer a popular way to describe words satisfying constraints. These constraints often appear as the syntactic constraints defining programming languages or also natural languages. The patterns specified by regular expressions can also be expressed in this way, but grammars are strictly more powerful.

The problem of parsing or syntax analysis consists in computing the derivation tree of a word, given a grammar.

A *grammar* $\mathfrak{G}$ on an alphabet $\mathcal{A}$ is given by a finite set $\mathcal{V}$ and a finite set $\mathcal{R} \subset \mathcal{V} \times (\mathcal{A} \cup \mathcal{V})^*$. The elements of $\mathcal{V}$ are called *variables* and the elements of $\mathcal{R}$ are called the *productions* of the grammar. A production (v, w) is often written $v \to w$. One fixes moreover a particular variable $i \in \mathcal{V}$ called the *axiom*. The grammar is denoted by $\mathfrak{G} = (\mathcal{A}, \mathcal{V}, \mathcal{R}, i)$.

Given two words $x, y \in (\mathcal{A} \cup \mathcal{V})^*$, one writes $x \to y$ if y is obtained from x by replacing some occurrence of v by w for some production (v, w) in $\mathcal{R}$, i.e. if $x = pvq$, $y = pwq$. One denotes by $\xrightarrow{*}$ the reflexive and transitive closure of the relation $\to$. Thus $x \xrightarrow{*} y$ if there exists a sequence $w_0 = x, w_1, \ldots, w_n = y$ of words $w_h \in (\mathcal{A} \cup \mathcal{V})^*$ such that $w_h \to w_{h+1}$ for $0 \le h < n$. Such a sequence is called a *derivation* from x to y. The *language generated* by the grammar $\mathfrak{G}$ is the set

$$L(\mathfrak{G}) = \{x \in \mathcal{A}^* \mid i \xrightarrow{*} x\}.$$

One may more generally consider the language generated by any variable v, denoted by $L(\mathfrak{G}, v) = \{x \in \mathcal{A}^* \mid v \xrightarrow{*} x\}$.

A grammar $\mathfrak{G} = (\mathcal{A}, \mathcal{V}, \mathcal{R}, i)$ can usefully be viewed as a system of equations, where the unknowns are the variables. Consider indeed the system of equations

$$v = W_v \quad (v \in \mathcal{V}) \tag{1.6.1}$$

where $W_v = \{w \mid (v, w) \in \mathcal{R}\}$. If each variable v is replaced by the set $L(\mathfrak{G}, v)$, one obtains a solution of the system of equations which is always the smallest solution (with respect to set inclusion) of the system.

A variant of the definition of a grammar is often used, where the sets W_v of Equation 1.6.1 are regular sets. In this case, these sets are usually described by regular expressions. This is equivalent to the first definition but often more compact. We give two fundamental examples of languages generated by a grammar.

Example 1.6.1. As a first example, let $\mathcal{A} = \{a, b\}$, $\mathcal{V} = \{v\}$ and $\mathcal{R}$ be composed of the two productions

$$v \to avv, \quad v \to b.$$

The language generated by the grammar $\mathfrak{G} = (\mathcal{A}, \mathcal{V}, \mathcal{R}, v)$ is known as the *Lukasiewicz language*. Its elements can be interpreted as arithmetic expressions in prefix notation, with a as an operator symbol and b as an operand symbol. The first words of $L(\mathfrak{G})$ in radix order are $b, abb, aabbb, ababb, aaabbbb, aababbb, aabbabb, abaabbb, \ldots$. In alphabetic order (with $a < b$) the last words are $\ldots, abb, b$.

Example 1.6.2. The second fundamental example is the *Dyck language* generated by the grammar $\mathfrak{G}$ with the same sets $\mathcal{A}, \mathcal{V}$ as above and the productions

$$v \to avbv, \quad v \to \epsilon.$$

Let $\mathcal{M}$ be the language generated by this grammar. Then $\mathcal{M} = a\mathcal{M}b\mathcal{M} + \varepsilon$. Set $\mathcal{D} = a\mathcal{M}b$. Then $\mathcal{M} = \mathcal{D}\mathcal{M} + \varepsilon$. This shows that $\mathcal{M} = \mathcal{D}^*$, and thus $\mathcal{D} = a\mathcal{D}^*b$. The set $\mathcal{M}$ is called the *Dyck language*, and $\mathcal{D}$ is the set of *Dyck primes*. The words in $\mathcal{M}$ can be viewed as well-formed sequences of parentheses with a as left parenthesis and b as right parenthesis. The words of $\mathcal{D}$ are the words in $\mathcal{M}$ which are not products of two nonempty words of $\mathcal{M}$. The first words in radix order in $\mathcal{D}$ and in $\mathcal{D}^*$ are respectively $ab, aabb, aababb, \ldots$, and $\epsilon, ab, aabb, abab, aabbab$. A basic relation between the Lukasiewicz set $\mathcal{L}$ and the Dyck language $\mathcal{M}$ is the equation

$$\mathcal{L} = \mathcal{M}b\,.$$

This is easy to verify, provided one uses the equational form of the grammar. The set $\mathcal{L}$ is indeed uniquely defined as the solution of the equation

$$\mathcal{L} = a\mathcal{L}\mathcal{L} + b \tag{1.6.2}$$

Since $\mathcal{M} = a\mathcal{M}b\mathcal{M} + \varepsilon$, multiplying both sides by b on the right, we obtain

$$\mathcal{M}b = a\mathcal{M}b\mathcal{M}b + b\,.$$

which is Equation 1.6.2, whence $\mathcal{M}b = \mathcal{L}$. There is a simple combinatorial interpretation of this identity. Let $\delta(x)$ denote the difference of the number of occurrences of a and of b in the word x. One can verify that a word x is in $\mathcal{M}$ if and only if $\delta(x) = 0$ and $\delta(p) \geq 0$ for each prefix p of x. Similarly, a word x is in $\mathcal{L}$ if and only if $\delta(x) = -1$ and $\delta(p) \geq 0$ for each proper prefix p of x.

A *derivation tree* for a word w is a tree T labeled by elements of $\mathcal{A} \cup \mathcal{V} \cup \{\varepsilon\}$ such that

1. the root of T is labeled by i.
2. for each interior node n, the pair (v, x) formed by the label v of n and the word x obtained by concatenating the labels of the children of n in left to right order is an element of $\mathcal{R}$.
3. A leaf is labeled ε only if it is the unique child of its parent.
4. The word w is obtained by concatenating the labels of the leaves of T in increasing order.

A derivation tree is a useful shorthand for representing a set of derivations. Indeed, any traversal of the derivation tree produces a derivation represented by this tree, and conversely.

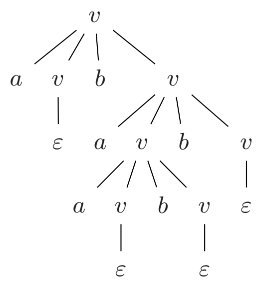


Figure 1.38. A derivation tree for the word *abaabb* in the Dyck grammar.

We present now in an informal manner two strategies for syntax analysis. Given a grammar $\mathfrak{G}$ and a word x, we want to be able to check whether x is in $L(\mathfrak{G})$. This amounts to build a derivation $i \xrightarrow{*} x$ from the axiom i of $\mathfrak{G}$ to x. There are two main options for doing this. The first one, called top-down parsing, builds the derivation from left to right (from i to x). This corresponds to constructing the derivation tree from the root to the leaves. The second one, called bottom-up parsing, builds the derivation from right to left (from x backwards to i). This corresponds to constructing the derivation tree from the leaves to the root.

1.6.1. Top-down parsing

The idea of top-down parsing is to build the derivation tree from the root. This is done by trying to build a derivation $i \xrightarrow{*} x$ and from left to right. The current situation in a top-down parsing is as follows (see Figure 1.39). A derivation $i \xrightarrow{*} yw$ has already been constructed. It has produced the prefix y of $x = yz$. It remains to build the derivation $w \xrightarrow{*} z$. We may assume that w starts with a variable v, that is $w = vs$. The key point for top-down parsing to work is that the grammar fulfills the following requirement. The pair (v, a), where a is the first letter of z, uniquely determines the production $v \to \alpha$ to be used, i. e. such that there exists a derivation $\alpha s \xrightarrow{*} z$. Grammars having this property for all x usually are called $LL(1)$ grammars.

We illustrate this method on two examples. The first one is the example of arithmetic expressions, and the second one concerns regular expressions already considered in Section 1.4. We consider the following grammar defining arithmetic expressions with operators $+$ and $*$ and parenthesis. The grammar allows unambiguous parsing of these expressions by introducing a hierarchy (expressions > terms > factors) reflecting the usual precedence of arithmetic operators $(* > +)$.

$$\begin{aligned} &E \to E + T \mid T \\ &T \to T * F \mid F \\ &F \to (E) \mid c \end{aligned} \tag{1.6.3}$$

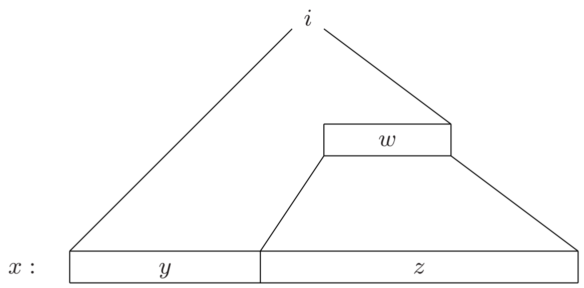


Figure 1.39. Top down parsing.

where c is any simple character.

We want to write a program to evaluate such an expression using top-down parsing. The idea is to associate to each variable of the grammar a function which acts according to the right side of the corresponding production in the grammar. To manage the word to be analyzed, a function Current() gives the first letter of the suffix of the input word that remains to be analyzed. In syntax analysis, the value of the function Current() is called the *lookahead* symbol.

A function Advance() allows to progress on the input word. The value of Current() allows one to choose the production of the grammar that should be used.

As already said, this method will work provided one may uniquely select, with the help of the value of Current(), which production should be applied. However, we are already faced with this problem with the productions $E \to E + T$ and $E \to T$, because the first letter of the input word does not allow to know whether there is a + sign following the first term. This phenomenon is called *left recursion*. To eliminate this feature, we transform the grammar and replace the two rules above by the equivalent form $E = T(+T)^*$. This shows that every expression starts with a term, and the continuation of the derivation is postponed to the end of the analysis of the first term.

The function corresponding to the variables E is given in Algorithm EvalExp. It returns the numerical value of the expression.

```
EvalExp()
v ← EvalTerm()
while Current() = '+' do
    Advance()
    v ← v + EvalTerm()
return v
```

The functions EvalTerm() and EvalFact() corresponding to T and F are similar.

```
EvalTerm()
1  v ← EvalFact()
2  while Current() = '*' do
3       Advance()
4       v ← v * EvalFact()
5  return v
```

```
EvalFact()
1  if Current() = '(' then
2       Advance()
3       v ← EvalExp()
4       Advance()
5  else v ← Current()
6       Advance()
7  return v
```

The instruction at line 5 of the function EvalFact() assigns to v the numerical value corresponding to the current symbol.

The evaluation of an expression, involving the parsing of its structure, is realized by the calling EvalExp().

As a second example, we show that the syntax of regular expressions can also be defined by a grammar. This is quite similar to the previously seen grammar of arithmetic expressions.

$$\begin{aligned} E &\to E + T \mid T \\ T &\to TF \mid F \\ F &\to G \mid G^* \\ G &\to (E) \mid c \end{aligned} \tag{1.6.4}$$

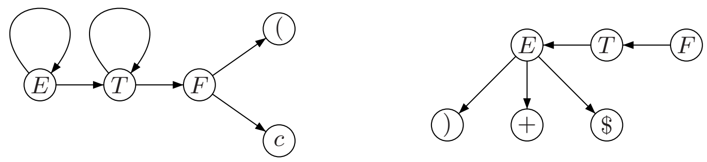
The symbol c stands for a letter or the symbol representing the empty word. A top-down parser for this grammar allows one to implement the constructions of the previous section that produce a finite automaton from a regular expression.

We have just seen top-down parsing developed on two examples. These examples show how easy it is to write a top-down analyzer. The drawback of this method is that it assumes that the grammar defining the language has a rather restricted form. In particular, it should not be left recursive, although there exist standard procedures to eliminate left recursion. However, there exist grammars that cannot be transformed into equivalent $LL(1)$ grammars that allow top-down parsing. The letters L in the acronym $LL(1)$ refer to left to right processing (on both the text and the derivation), and the number 1 refers to the number of lookahead symbols.

The precise definition of $LL(1)$ grammars uses two functions called First() and Follow() that associate to each variable a set of terminal symbols. For a variable $x \in \mathcal{V}$, First(x) is the set of terminal symbols $a \in \mathcal{A}$ such that there is a derivation of the form $x \xrightarrow{*} au$. The function First() is extended to words in a natural way: First(w) is the set of terminal symbols a such that $w \xrightarrow{*} au$.

For each variable $x \in \mathcal{V}$, Follow(x) is the set of terminal symbols $a \in \mathcal{A}$ such that there is a derivation $u \xrightarrow{*} vxaw$ with a "following" x.

To compute First(), we build a graph with $\mathcal{A} \cup \mathcal{V}$ as vertices and edges the pairs $(x, y) \in \mathcal{V} \times (\mathcal{A} \cup \mathcal{V})$ such that there is a production of the form $x \rightarrow uyw$ with $u \xrightarrow{*} \varepsilon$. Then $a \in \text{First}(x)$ iff there is a path from x to a in this graph. The graph corresponding to the grammar of arithmetic expressions is shown on Figure 1.40(a).



(a) First(). (b) Follow().

Figure 1.40. The graphs of First() and of Follow().

The algorithm used to compute First() is given more precisely below. We begin with an algorithm (Epsilon()) which computes a boolean array *epsilon* indicating whether a symbol v is nullable, i.e. whether $v \xrightarrow{*} \varepsilon$. The array *epsilon* has of size $n + k$, where n is the number of variables in the grammar and k is the number of terminals.

Epsilon()
for each production $v \rightarrow \varepsilon$ **do**
  $\mathit{epsilon}[v] \leftarrow$ **true**
for $i \leftarrow 0$ **to** $n - 1$ **do**
  for each production $v \rightarrow x_1 \cdots x_m$ **do**
    $\mathit{epsilon}[v] \leftarrow \mathit{epsilon}[v] \vee (\mathit{epsilon}[x_1] \wedge \cdots \wedge \mathit{epsilon}[x_m])$
return *epsilon*

It is easy to compute a function IsNullable(w) for $w = x_1 \cdot x_n$ as the conjunction of the boolean values $\mathit{epsilon}[x_i]$. The computation of First() consists in several steps. We first compute the graph defined above. The graph is represented by the set FirstChild(v) of successors of each variable v. The function First() is computed after a depth-first exploration of the graph has been performed.

FirstChild(v)
$\triangleright$ S is the set of successors of v
$S \leftarrow \emptyset$
for each production $v \rightarrow x_1 \cdots x_m$ **do**
  for $i \leftarrow 1$ **to** m **do**
    $S \leftarrow S \cup x_i$
    if $\mathit{epsilon}[x_i] =$ **false then**
      break
return S

We mark vertices in the graph by a standard depth-first exploration.

```
ExploreFirstChild(v)
firstmark[v] ← true
for each x ∈ FirstChild(v) do
      if firstmark[x] = false then
            ExploreFirstChild(x)
```

The array *firstmark* is used for exploration of the graph of First(). Finally, we compute First().

```
First(v)
▷ mark is an array initialized to false
ExploreFirstChild(v)
S ← ∅
for each terminal c do
      if firstmark[c] then
            S ← S ∪ c
return S
```

The values of the function First() could of course be stored in an array *first*. The extension of First to words is straightforward.

```
First(w)
S ← ∅
for i ← 1 to n do                ▷ w has length n
      S ← S ∪ First(w[i])
      if epsilon[w[i]] = false then
            break
return S
```

There is an alternative way to present the computation of First(), by means of a system of mutually recursive equations. For this, observe that for each variable x, First(x) is the union of the sets First(y) over the the set $S(x)$ of successors of x in the graph of First(). Thus, the function First() is the least solution of the system of equations

$$\text{First}(x) = \cup_{y \in S(x)} \text{First}(y) \qquad (x \in V)$$

such that First$(a) = a$ for each letter $a \in \mathcal{A}$. For example, the equations for the grammar 1.6.3 are

$$\begin{aligned} \text{First}(E) &= \text{First}(E) \cup \text{First}(T) \\ \text{First}(T) &= \text{First}(T) \cup \text{First}(F) \\ \text{First}(F) &= \{(, c\} \end{aligned}$$

To compute the function Follow(), we build a similar graph. There are two rules to define the edges.

1. if there is a production $x \to uvw$ with a terminal symbol a in $\text{First}(w)$, then (v, a) is an edge.
2. if there is a production $z \to uxw$ with $w \xrightarrow{*} \varepsilon$, then (x, z) is an edge (notice that we use the productions backwards).

The graph of Follow() for the grammar of arithmetic expressions is shown on Figure 1.40(b). The computation of the function Follow is analogous. It begins with the computation of the graph Sibling(x).

```
Sibling(x)
S ← ∅
for each production z → uxw do
    S ← S ∪ First(w)
    if IsNullable(w) then
        S ← S ∪ z
return S
```

The depth-first exploration ExploreSibling(v) is then performed as before. It produces an array *followmark* which is used to compute the function Follow().

```
Follow(v)
▷ followmark is an array initialized to false
ExploreSibling(v)
S ← ∅
for each terminal c do
    if followmark[c] then
        S ← S ∪ c
return S
```

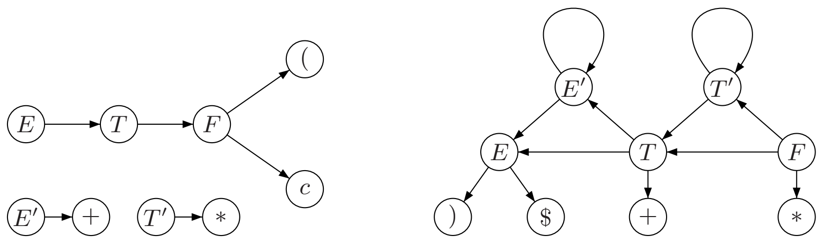
As for the function First(), the function Follow() can also be computed by solving a system of equations. The precise definition of an $LL(1)$ grammar can now be formulated. It is a grammar such that

1. for each pair of distinct productions $x \to u$, $x \to v$, with the same left side and $u, v \neq \varepsilon$, one has
$$\text{First}(u) \cap \text{First}(v) = \emptyset.$$
2. For each pair of distinct productions of the form $x \to u$, $x \to \varepsilon$, one has
$$\text{First}(u) \cap \text{Follow}(x) = \emptyset.$$

Observe that our grammar for arithmetic expressions violates the first condition, since for instance $\text{First}(E) = \text{First}(T)$, although we have two productions $E \to E + T$ and $E \to T$ with the same left hand side. We have already met this problem of left recursion, and solved it by transforming the grammar. The solution that we described is actually equivalent to consider the grammar

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \varepsilon \\ T &\to FT' \\ T' &\to *FT' \mid \varepsilon \\ F &\to (E) \mid c \end{aligned} \tag{1.6.5}$$

This grammar is equivalent to grammar 1.6.3. It meets the two conditions for being $LL(1)$. Indeed, the functions First() and Follow() are given in Figure 1.41.



(a) First(). (b) Follow().

Figure 1.41. The graphs of First() and Follow() for the grammar 1.6.5.

For example, consider the productions $E' \to \varepsilon$ and $E' \to +TE'$. The symbol $+$ is not in Follow(E'), and thus the second condition is satisfied for this pair of productions. The characterization allows us to fill the entries of a table called the *parsing table* given in 1.1. This is an equivalent way to define the mutually recursive functions we defined above (for the wise : this is also a way to convince oneself that the programs are correct !)

	c	$+$	$*$	(	)	\$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to c$			$F \to (E)$		

Table 1.1. The parsing table of grammar 1.6.5.

The computation of the $LL(1)$ parsing table uses the following algorithm.

LLTable()
▷ computes the $LL(1)$ parsing table M
for each production $p : v \rightarrow w$ **do**
    for each terminal $c \in \mathrm{First}(w)$ **do**
        $M[v][c] \leftarrow p$
    if $w = \varepsilon$ **then**
        for each terminal $c \in \mathrm{Follow}(v)$ **do**
            $M[v][c] \leftarrow p$
return M

The above algorithm as written supposes the grammar to be $LL(1)$. Error messages to inform that the grammar is not $LL(1)$ can easily be added.

1.6.2. Bottom-up parsing

We now describe bottom-up parsing which is a more complicated but more powerful method for syntax analysis.

The idea of bottom-up parsing is to build the derivation tree from the leaves to the root. This method is more complicated to program, but is more powerful than top-down parsing.

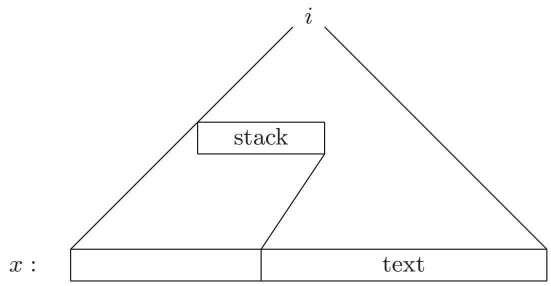


Figure 1.42. Bottom up parsing.

The current situation of bottom-up parsing is pictured in Figure 1.42. The left part of the text which has already been analyzed has been reduced, using the productions backwards, to a string that is kept in a stack. We will see below that this actually corresponds to a last-in first-out strategy.

We present bottom-up parsing on the example of arithmetic expressions already used above.

$$\begin{aligned} &1 : E \rightarrow E + T \\ &2 : E \rightarrow T \\ &3 : T \rightarrow T * F \\ &4 : T \rightarrow F \\ &5 : F \rightarrow (E) \\ &6 : F \rightarrow c \end{aligned} \tag{1.6.6}$$

We reproduce the grammar 1.6.3 with productions numbered from 1 to 6.

Programming a bottom-up analyzer involves the management of a stack containing the part of the text that has already been analyzed. The evolution of the stack and of the text is pictured below (Figure 1.43) to be read from top to bottom.

	Stack	Text
1		$(1+2)*3$
2	$($	$1+2)*3$
3	$(c$	$+2)*3$
4	$(F$	$+2)*3$
5	$(T$	$+2)*3$
6	$(E$	$+2)*3$
7	$(E+$	$2)*3$
8	$(E+c$	$)*3$
9	$(E+F$	$)*3$
10	$(E+T$	$)*3$
11	$(E$	$)*3$
12	(E)	$*3$
13	F	$*3$
14	T	$*3$
15	$T*$	3
16	$T*c$	
17	$T*F$	
18	T	
19	E	

Figure 1.43. Evolution of the stack and of the text during the bottom-up analysis of the expression $(1+2)*3$.

At the beginning, the stack is empty. Each step consists either in

1. transferring a new symbol from the text to the stack (this operation is called a *shift*);
2. reducing the top part of the stack according to a rule of the grammar (this is a *reduction*).

As an example, the second and third row in Figure 1.43 are the results of shifts, while the three following rows are the results of reductions by rules number 6, 4, and 2 respectively.

To be able to choose between shift and reduction, one uses a finite automaton called *LR automaton*. This automaton keeps track of the information concerning the presence of the right side of a rule at the top of the stack. In our example, the automaton is given in Figure 1.44.

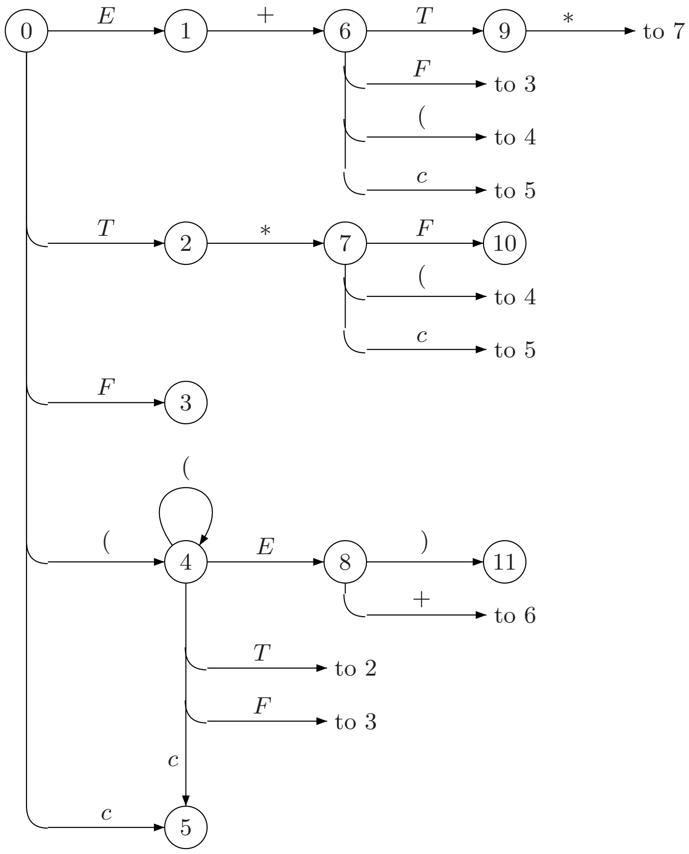


Figure 1.44. The LR automaton.

The input to the *LR* automaton is the content of the stack. According to the state reached, and to the lookahead symbol, the decision can be made whether to shift or to reduce, and in the latter case by which rule. The fact that this is possible is a property of the grammar. These grammars are called *SLR*–grammars.

In practice, instead of pushing the symbols on the stack, one rather pushes the states of the *LR* automaton. The result on the expression $(1+2)*3$ is shown on Figure 1.45.

The decision made at each step uses two arrays S and R, represented on Figure 1.46.

The array S is the transition table of the *LR* automaton. Thus $S[p][c]$ is the state reached from state p by reading c. The table R indicates which reduction to perform. The value $R[p][c]$ indicates the number of the production to be used backwards to perform a reduction when the state p is on top of the stack

Stack	Text
0	$(1+2)*3\$$
0 4	$1+2)*3\$$
0 4 5	$+2)*3\$$
0 4 3	$+2)*3\$$
0 4 2	$+2)*3\$$
0 4 8	$+2)*3\$$
0 4 8 6	$2)*3\$$
0 4 8 6 5	$)*3\$$
0 4 8 6 3	$)*3\$$
0 4 8 6 9	$)*3\$$
0 4 8	$)*3\$$
0 4 8 11	$*3\$$
0 3	$*3\$$
0 2	$*3\$$
0 2 7	$3\$$
0 2 7 5	$\$$
0 2 7 10	$\$$
0 2	$\$$
0 1	$\$$

Figure 1.45. The stack of states of the LR automaton during the bottom-up analysis of the expression $(1+2)*3$.

	c	$+$	$*$	$($	$)$	$\$$	E	T	F
0	5			4			1	2	3
1		6				*Acc*			
2			7						
3									
4	5			4			8	2	3
5									
6	5			4				9	3
7	5			4					10
8		6			11				
9			7						
10									
11									

(a) The array S.

	c	$+$	$*$	$($	$)$	$\$$	E	T	F
0									
1									
2		2			2	2			
3		4	4		4	4			
4									
5		6	6		6	6			
6									
7									
8									
9		1			1	1			
10		3	3		3	3			
11		5	5		5	5			

(b) The array R.

Figure 1.46. The arrays S and R

and the symbol c is the lookahead symbol. Empty entries in tables S and R

correspond to non existing transitions. A special state *Accept*, abbreviated as *Acc* is the accepting state ending the computation with a successful analysis. The tables S and R could be superposed because their nonempty entries are disjoint. Actually, this is necessary for the LR-algorithm to work!

The implementation of the algorithm is given in the function LRParse(). It uses, on the input, the two functions Current(), Advance() already described earlier, and the symbol '\$' to mark the end of the text. The function Top() and Push() are the usual functions on stacks. The function Reduce() operates in three steps. The call Reduce(n), where n is the index of the production $r \to u$, consists in the following

1. it erases from the stack the number of states equal to the length of u,
2. it computes the new value $p =$ Top() and the state $q = T[p][r]$,
3. it pushes q on the stack.

In the implementation, the value -1 represents non existing transitions. The function returns the boolean value **true** if the analysis was successful, and **false** otherwise. There are three cases of failure

1. there is no legal shift nor legal reduction, this is checked at lines 5 and 9. This happens for instance if the input is x')'.
2. the text has not been exhausted at the end of the analysis, for instance if $x =$ '('; this leads to the same situation as above, because the state *Accept* can only be accessed by the end marker.
3. the text has been exhausted before the end of the analysis; in this case, the end marker leads to an empty entry in the tables.

```
LRParse(x)
 1  while Top() ≠ Accept do
 2        p ← Top()
 3        c ← Current()
 4        q ← T[p][c]
 5        if q ≠ −1 then
 6              Push(q)
 7              Advance()
 8        else n ← R[p][c]
 9              if n ≠ −1 then
10                    Reduce(n)
11              else return false
12  return true
```

There remains to explain how to compute the LR automaton and the corresponding tables from the grammar. We work with an end marker '\$'. Accordingly, we add to the grammar an additional rule which, in our running example, is $E' \to E\$$. The LR automaton recognizes the content of the stack and its state allows one to tell whether the right side of some production is present on the top of the stack. The set of possible stack contents (sometimes called the *viable prefixes*) is the set

$$\mathcal{X} = \{p_1 p_2 \cdots p_n \mid p_i \in P, n \geq 0\}$$

where P is the set of prefixes of the right sides of the productions and where, for each i, $1 \leq i \leq n-1$, there is a production (x_i, v_i) such that $p_i x_{i+1}$ is a prefix of v_i, and x_1 is the axiom of the grammar. One may verify this description of $\mathcal{X}$ by working on the bottom-up analysis backwards. It is easy to build a non deterministic automaton recognizing the set $\mathcal{X}$ above. It is built from the automata recognizing the right sides of the productions and adding ε-transitions from each position before a variable y to the initial positions of the productions with left side y.

The result is represented on Figure 1.47. The circled states correspond to full right hand sides and thus to productions of the grammar.

To be complete, we should add the transitions corresponding to the rule $E' \to E\$$. The states 0 and 4 which correspond to the productions with left side E. The automaton of Figure 1.44 is just the result of the determinization algorithm applied to the non deterministic automaton obtained. This explains how the LR automaton and thus the table S, which is just its transition table, are built. There still remains to explain how table R is built. We have $R[p][c] = n$ if and only if the reduction by production $n : x \to v$ is possible in state p, and provided the lookahead symbol c is in $\textsc{Follow}(x)$. This solves the conflicts between shift and reduce.

Suppose for example that the variable T is on top of the stack, as at lines $5, 14, 18$ of Figure 1.43. At each of these lines, we can either reduce by production 2 or shift. Similarly, at line 10 we can either reduce by production 1 or 2, or shift. We should reduce only if the lookahead symbol is in $\textsc{Follow}(E)$. This is why we choose to reduce by production 2 at lines 5 and 18. At line 14, we choose to shift, because the symbol $*$ is not in $\textsc{Follow}(E)$. At line 10, we reduce by production 1 because the corresponding state 9 allows this reduction and the lookahead symbol ')' is in $\textsc{Follow}(E)$.

A grammar such that the method used above to fill the table R works is called $SLR(1)$. A word on this terminology. The acronym LR refers to a left to right analysis of the text and a rightmost derivation (corresponding to a bottom-up analysis). A grammar is said to be $LR(0)$ if no shift-reduce conflict appears on the LR automaton. The 0 means that no lookahead is needed to make the decisions.This is not the case of Grammar 1.6.6, as we have seen. The acronym SLR means 'simple LR' and the integer 1 refers to the length of the lookahead. Formally, a grammar is said to be $SLR(1)$ if for any state p of the LR automaton and each terminal symbol c, at most one of the two following cases arise.

1. There is a transition from p by c in the automaton.

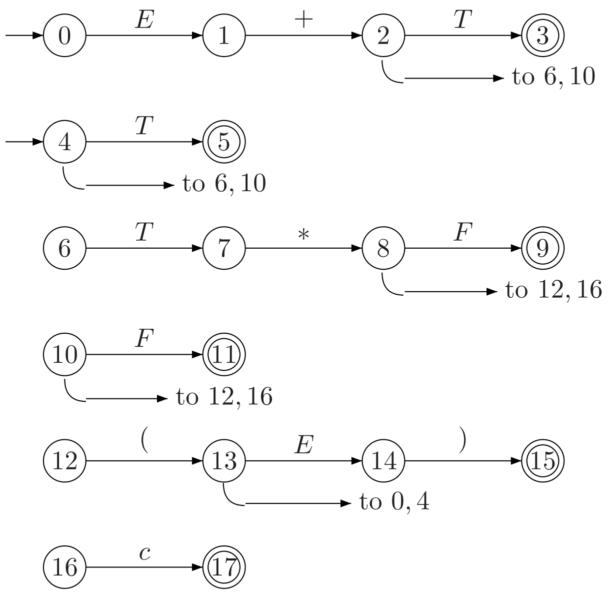


Figure 1.47. A non deterministic LR automaton

2. There is a possible reduction in state p by production $n : x \to v$ such that $c \in \textsc{Follow}(x)$.

In practice, this condition is equivalent to the property that the sets of non empty entries of the tables S and R are disjoint.

More complicated methods exist, either with lookahead 1 or with a larger lookahead, although a lookahead of size larger than 1 is rarely used in practice. With lookahead 1, the class of $LR(1)$ grammars uses an automaton called the $LR(1)$ automaton to keep track of the pair (s, c) of the stack content s and the lookahead symbol c to be expected at the next reduction. The main drawback is that the number of states is much larger than with the $LR(0)$ automaton.

1.7. Word enumeration

One often has to compute the number of words satisfying some property. This can be done using finite automata or grammars as illustrated in the following examples.

1.7.1. Two illustrative examples

The first example illustrates the case of a property defined by a finite automaton.

Example 1.7.1. The number u_n of words of length n on the binary alphabet $\{a, b\}$ which do not contain two consecutive a's satisfies the recurrence formula $u_{n+1} = u_n + u_{n-1}$. Indeed, a nonempty word of length n can either terminate with a or b. In the first case, it has to terminate with ba unless it is the word a. Since $u_0 = 1$ and $u_1 = 2$, the number u_n is the Fibonacci number F_{n+2}.

This argument can be used quite generally when the corresponding set of words is recognized by a finite automaton. In the present case, the set $\mathcal{S}$ without factor bb is recognized by the Golden mean automaton of Figure 1.11. Let $\mathcal{S}_q$ be the set of words recognized by the automaton with initial state 1 and final state q. We derive from the automaton the following set of equations

$$\begin{aligned} \mathcal{S}_1 &= \mathcal{S}_1 b + \mathcal{S}_2 b + \varepsilon \\ \mathcal{S}_2 &= \mathcal{S}_1 a \end{aligned}$$

Since $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$, summing up the equations gives

$$\mathcal{S} = \mathcal{S}b + \mathcal{S}_1 a + \varepsilon = \mathcal{S}(b + ab) + \varepsilon\,.$$

This gives the expected recurrence relation.

A second example concerns the Dyck language.

Example 1.7.2. Recall from Example 1.6.2 that the Dyck language $\mathcal{D}^*$ is related to the Lukasiewicz language $\mathcal{L}$ by the relation $\mathcal{D}^* b = \mathcal{L}$. Let f_n be the number of words of length n in $\mathcal{D}$ and let u_n be the number of words of length n in $\mathcal{D}^*$.

It can be verified, using the function δ of Example 1.6.2, that each word x of length $2n+1$ with $\delta(x) = -1$ is primitive and has exactly one conjugate in $\mathcal{L}$. Since u_{2n} is also the number of words of length $2n+1$ in $\mathcal{L}$, one gets

$$u_{2n} = \frac{1}{2n+1}\binom{2n+1}{n} = \frac{1}{n+1}\binom{2n}{n}\,.$$

Since $\mathcal{D} = a\mathcal{D}^* b$, it follows that

$$f_{2n} = \frac{1}{n}\binom{2n-2}{n-1}\,.$$

The sequence (u_{2n}) is the sequence of *Catalan numbers*.

The combinatorial method used to compute the numbers f_n and u_n can be frequently generalized in the case of more complicated grammars (see Chapter 9). In the present case, the relation is the following.

We start with the relation $\mathcal{D} = a\mathcal{D}^* b$. This implies that the generating function $D(z) = \sum_{n \geq 0} f_n z^n$ satisfies the equation

$$D^2 - D + z^2 = 0\,.$$

It follows that

$$D(z) = \frac{1 - \sqrt{1 - 4z^2}}{2}\,.$$

An elementary application of the binomial formula gives the the expected expression for the coefficient f_n.

1.7.2. The Perron–Frobenius theorem

Several enumeration problems on words involve the spectral properties of non-negative matrices. The *Perron–Frobenius theorem* describes some of these properties and constitutes a very important tool in this framework. We shall see in the next section several applications of this theorem.

Let Q be a set of indices (we have of course in mind the set of states of a finite automaton). For two Q-vectors v, w with real coordinates, one writes $v \leq w$ if $v_q \leq w_q$ for all $q \in Q$ and $v < w$ if $v_q < w_q$ for all $q \in Q$. A vector v is said to be *nonnegative* (resp. *positive*) if $v \geq 0$ (resp. $v > 0$). In the same way, for two $Q \times Q$-matrices M, N with real coefficients, one writes $M \leq N$ when $M_{p,q} \leq N_{p,q}$ for all $p, q \in Q$ and $M < N$ when $M_{p,q} < N_{p,q}$ for all $p, q \in Q$. The $Q \times Q$-matrix M is said to be *nonnegative* (resp. *positive*) if $M \geq 0$ (resp. $M > 0$). We shall use often the elementary fact that if $M > 0$ and $v \geq 0$ with $v \neq 0$, then $Mv > 0$.

A nonnegative matrix M is said to be *irreducible* if for all indices p, q, there is an integer k such that $M^k_{p,q} > 0$, where M^k denotes the k–th power of M. It is easy to verify that M is irreducible if and only if $(I + M)^n > 0$ where n is the dimension of M. It is also easy to prove that M is reducible (i.e. M is not irreducible) if there is a reordering of the indices such that M is block triangular, i.e. of the form

$$M = \begin{bmatrix} U & V \\ 0 & W \end{bmatrix} \tag{1.7.1}$$

with U, W of dimension > 0.

A nonnegative matrix M is called *primitive* if there is an integer k such that $M^k > 0$. A primitive matrix is irreducible but the converse is not true.

A nonnegative matrix M is called *aperiodic* if the greatest common divisor of the integers k such that $M^k_{i,i} > 0$ for some i is equal to 1 (including the case where the set of integers k is empty). It can be verified that a matrix is primitive if and only if it is aperiodic and irreducible.

The Perron–Frobenius Theorem asserts that for any nonnegative matrix M, the following holds

1. The matrix M has a real eigenvalue ρ_M such that $|\lambda| \leq \rho_M$ for any eigenvalue λ of M.
2. If $M \leq N$ with $M \neq N$, then $\rho_M < \rho_N$.
3. There corresponds to ρ_M a nonnegative eigenvector v and ρ_M is the only eigenvalue with a nonnegative eigenvector.
4. If M is irreducible, the eigenvalue ρ_M is simple and there corresponds to ρ_M a positive eigenvector v.
5. If M is primitive, all other eigenvalues have modulus strictly less than ρ_M. Moreover, $\frac{1}{\rho^n_M} M^n$ converges to a matrix of the form vw, where v (w) is a right (left) eigenvector corresponding to ρ, i.e. $Mv = \rho v$ ($wM = \rho w$) and $wv = 1$.

We shall give a sketch of a proof of this classical theorem. Let us first show that one may reduce to the case where M is irreducible. Indeed, if M is reducible, we may consider a triangular decomposition as in Equation 1.7.1 above. Applying by induction the theorem to U and W, we obtain the result with ρ_M equal to the maximal value of the moduli of eigenvalues of U and W. The corresponding eigenvector is completed with zeroes (and thus condition 4 fails to hold).

We suppose from now on that M is irreducible. For a nonnegative Q-vector v, let

$$r_M(v) = \min\{(Mv)_i/v_i \mid 1 \le i \le n, v_i \ne 0\}$$

Thus $r_M(v)$ is the largest real number r such that $Mv \ge rv$. The function r_M is known as the *Wielandt* function. One has $r_M(\lambda v) = r_M(v)$ for all real number $\lambda \ge 0$. Moreover, r_M is continuous on the set of nonnegative vectors.

The set X of nonnegative vectors v such that $\|v\| = 1$ is compact. Since a continuous function on a compact set reaches its maximum on this set, there is an $x \in X$ such that $r_M(x) = \rho_M$ where $\rho_M = \max\{r_M(w) \mid w \in X\}$. Since $r_M(v) = r_M(\lambda v)$ for $\lambda \ge 0$, we have $\rho_M = \max\{r_M(w) \mid w \ge 0\}$.

We show that $Mx = \rho_M x$. By the definition of the function r_M, we have $Mx \ge \rho_M x$.

Set $y = Mx - \rho_M x$. Then $y \ge 0$. Assume $Mx \ne \rho_M x$. Then $y \ne 0$. Since $(I + M)^n > 0$, this implies that the vector $(I + M)^n y$ is positive. But

$$(I+M)^n y = (I+M)^n(Mx-\rho_M x) = M(I+M)^n x-\rho_M(I+M)^n x = Mz-\rho_M z\,,$$

with $z = (I + M)^n x$. This shows that $Mz > \rho_M z$, which implies that $r_M(z) > \rho_M$, a contradiction with the definition of r_M. This shows that ρ_M is an eigenvalue with a nonnegative eigenvector.

Let us show that $\rho_M \ge |\lambda|$ for each real or complex eigenvalue λ of M. Indeed, let v be an eigenvector corresponding to λ. Then $Mv = \lambda v$. Let $|v|$ be the nonnegative vector with coordinates $|v_i|$. Then $M|v| \ge \lambda|v|$ by the triangular inequality. By the definition of the Wielandt function, this implies $r_M(|v|) \ge |\lambda|$ and consequently $\rho_M \ge |\lambda|$. This completes the proof of assertion 1.

We have already seen that there corresponds to ρ_M a nonnegative eigenvector x. Let us now verify that $x > 0$. But this is easy since $(I+M)^n x = (1+\rho_M)^n x$, which implies that $(1 + \rho_M)^n x > 0$ and thus $x > 0$.

In order to prove assertion 2, let us consider N such that $M \le N$. Then obviously $\rho_M \le \rho_N$. Let us show that $\rho_M = \rho_N$ implies $M = N$. Let $v > 0$ be such that $Mv = \rho_M v$. Then $Nv \ge \rho_M v$ and we conclude as above that $Nv = \rho_M v$. From $Mv = Nv$ with $v > 0$, we conclude that $M = N$ as asserted.

We now complete the proof of assertion 3. Let $Mv = \lambda v$ with $v \ge 0$. Since, as above, $(I + M)^n v = (1 + \lambda)^n v$, we have actually $v > 0$. Let D be the diagonal matrix with coefficients $v_1, v_2, \ldots, v_n$ and let $N = D^{-1}MD$. Since $n_{i,j} = m_{i,j}v_j/v_i$, we have $\sum_j b_{i,j} = \lambda$ for $1 \le i \le n$. Let w be a nonnegative eigenvector of N for the eigenvalue ρ_M. We normalize w in such a way that $w_i \le 1$ for all i and $w_t = 1$ for one index t. Then, $\rho_M = \sum_j n_{t,j}w_j \le \sum_j n_{t,j} = \lambda$. Thus $\lambda = \rho_M$ as asserted. This completes the proof of assertion 3.

We further have to prove that ρ_M is simple. Let $p(\lambda) = \det(\lambda I - M)$ be the characteristic polynomial of M. We have $p'(\lambda) = \sum_i \det(\lambda I - M_i)$ where

M_i is the matrix obtained from M by replacing the i-th row and column by 0. Indeed,

$$\det(\lambda I - M) = \det(\lambda e_1 - v_1, \ldots, \lambda e_n - v_n)$$

where e_i is the i-th unit vector and v_i is the i-th column of M. Since the determinant is a multilinear function, this gives the desired formula for $p'(\lambda)$. One has $M_i \leq M$ and $M_i \neq M$ for each i, because an irreducible matrix cannot have a null row. By assertion 2, $\rho_{M_i} < \rho_M$ and thus $\det(\rho_M I - M_i) > 0$, whence $p'(\rho_M) > 0$. This shows that the root ρ_M is simple.

Let us finally prove assertion 5. Let λ be an eigenvalue of M such that $|\lambda| = \rho_M$. Let v be an eigenvector for the eigenvalue λ. Then, from $Mv = \rho_M v$, we obtain $M|v| \geq \rho_M|v|$ whence $M|v| = \rho_M|v|$ by the same argument as above. Let k be such that $M^k > 0$. Then, from $|M^k v| = M^k|v|$, we deduce that v is collinear to a nonnegative real vector. This shows that λ is real and thus that $\lambda = \rho_M$.

Since M has a simple eigenvalue ρ_M strictly greater than every other eigenvalue, the sequence $\frac{1}{\rho_M^n} M^n$ converges to a matrix of rank one, which is thus of the indicated form.

This completes the proof of the Perron–Frobenius theorem. For an indication of another proof, see Problem 1.7.1.

The practical computation of the maximal eigenvalue of a primitive matrix M can be done using the following algorithm. It is based on the fact that, by Assertion 5 the sequence defined by $x^{(n+1)} = \frac{1}{r(x^{(n)})} M x^{(n)}$ converges to an eigenvector corresponding the maximal eigenvalue, and thus $r(x^{(n)})$ converges to an eigenvector. The starting value $x^{(0)}$ can be an arbitrary positive vector.

DominantEigenvalue(M, x)
$y \leftarrow x$
do $(y, x) \leftarrow (Mx, y)$
  $r \leftarrow \min_{1 \leq i \leq n} y_i / x_i$
  $y \leftarrow \frac{1}{r} y$
while $y \not\approx x$
return r

where $y \approx x$ means that y is numerically close to x.

The vector computed by this algorithm is called an *approximate eigenvector*. The definition is the following. Let M be nonnegative matrix. Let r be such that $r \leq \rho_M$. Then a vector v such that $Mv \geq rv$ is called an approximate eigenvector relative to r.

Example 1.7.3. Let

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

The matrix M is nonnegative and irreducible. The eigenvalues of M are $\varphi = (1 + \sqrt{5})/2$ and $\hat{\varphi} = (1 - \sqrt{5})/2$. The vector $x = \begin{bmatrix} \varphi \\ 1 \end{bmatrix}$ is an eigenvector relative to φ. The vector $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an approximate eigenvector relative to $r = 1$ and Mv is an approximate eigenvector relative to $r = 3/2$.

1.8. Probability distributions on words

In this section, we consider the result of randomly selecting the letters composing a word. We begin with the formal definition of a probability law ruling this selection.

1.8.1. Information sources

Given an alphabet $\mathcal{A}$, a *probability distribution* on the set of words on $\mathcal{A}$ is a function $\pi : \mathcal{A}^* \to [0, 1]$ such that $\pi(\epsilon) = 1$ and for each word $x \in \mathcal{A}^*$,

$$\sum_{a \in \mathcal{A}} \pi(xa) = \pi(x).$$

The definition implies that $\sum_{x \in \mathcal{A}^n} \pi(x) = 1$ for all $n \geq 0$. Thus a probability distribution on words does not make the set of all words a probability space but it does for each set $\mathcal{A}^n$.

Probability distributions on words are sometimes defined with a different vocabulary. One considers a sequence of random variables $(X_1, X_2, \ldots, X_n, \ldots)$ with values in the set $\mathcal{A}$. Such a sequence is often called a discrete time information *source* or also a stochastic *process*. For $x = a_1 \cdots a_n$ with $a_i \in \mathcal{A}$, set

$$\pi(x) = \mathbf{P}(X_1 = a_1, \ldots, X_n = a_n)$$

Then π is a probability distribution in the previous sense. Conversely, if π is a probability distribution, this formula defines the n-th order joint distribution of the sequence $(X_1, X_2, \ldots, X_n, \ldots)$. We will say that $\mathbf{P}$ and π *correspond* to each other.

Two particular cases are worth mentioning: Bernoulli distributions and Markov chains.

First, a *Bernoulli distribution* corresponds to successively independent choices of the symbols in a word, with a fixed distribution on letters. Thus it is given by a probability distribution on the set $\mathcal{A}$ extended by simple multiplication. For $a_1, a_2, \ldots, a_n \in \mathcal{A}$, one has $\pi(a_1 a_2 \cdots a_n) = \pi(a_1)\pi(a_2) \cdots \pi(a_n)$. In the terminology of information sources, a Bernoulli distribution corresponds to a sequence of independent, identically distributed (i.i.d.) random variables.

For example, if the alphabet has two letters a and b with probabilities $\pi(a) = p$ and $\pi(b) = q = 1 - p$, then $\pi(w) = p^{|w|_a} q^{|w|_b}$. The random variable X whose value is the number of b's in a word w of length n has the distribution

$$\mathbf{P}(X = m) = \binom{n}{m} p^{n-m} q^m$$

This distribution is called the *binomial distribution*. Its expectation and variance are

$$\mathbf{E}(X) = np, \quad \mathrm{Var}(X) = npq\,.$$

Second, a *Markov chain* corresponds to the case where the probability of choosing a symbol depends on the previous choice, but not on earlier choices. Thus, a Markov chain is given by an initial distribution π on $\mathcal{A}$ and by an $\mathcal{A} \times \mathcal{A}$ stochastic matrix P of conditional probabilities $P(a, b)$, i.e. such that for all $a \in \mathcal{A}$, $\sum_{b \in \mathcal{A}} P(a, b) = 1$. Then

$$\pi(a_1 a_2 \cdots a_n) = \pi(a_1) P(a_1, a_2) \cdots P(a_{n-1}, a_n).$$

In terms of stochastic processes, $P(a, b)$ is the conditional probability given for all $n \geq 2$ by $P(a, b) = \mathbf{P}(X_n = b \mid X_{n-1} = a)$. The powers of the matrix P allow to compute the probability $\mathbf{P}(X_n = a)$. Indeed, one has $\mathbf{P}(X_n = a) = (\pi P^n)(a)$.

Example 1.8.1. Consider the Markov chain over $\mathcal{A} = \{a, b\}$ given by the matrix

$$\begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$$

and the initial distribution $\pi(a) = \pi(b) = 1/2$. For example, one has $\pi(aab) = \pi(aaba) = 1/8$. This distribution assigns probability 0 to any word containing two consecutive b's because $P(b, b) = 0$.

A distribution π on $\mathcal{A}^*$ is said to be *stationary* if for all $x \in \mathcal{A}^*$, one has $\pi(x) = \sum_{a \in \mathcal{A}} \pi(ax)$. In terms of stochastic processes, this means that the joint distribution does not depend on the choice of time origin, that is,

$$\mathbf{P}(X_i = a_i, m \leq i \leq n) = \mathbf{P}(X_{i+1} = a_i, m \leq i \leq n)$$

A Bernoulli distribution is a stationary distribution. A Markov chain is stationary if and only if $\pi P = \pi$, i.e. if π is an eigenvector of the matrix P for the eigenvalue 1. The distribution π on $\mathcal{A}$ is itself called stationary.

A Markov chain is *irreducible* if, for all $a, b \in \mathcal{A}$, there exists an integer $n \geq 0$ such that $P^n(a, b) > 0$. This is exactly the definition of an irreducible matrix.

Similarly, a Markov chain is *aperiodic* if the matrix P is aperiodic.

The fundamental theorem of Markov chains says that for any irreducible Markov chain, there is a unique stationary distribution π, and whatever be the initial distribution, $\mathbf{P}(X_n = a)$ tends to $\pi(a)$. The proof uses the Perron–Frobenius theorem.

Example 1.8.2. Consider the Markov chain over $\mathcal{A} = \{a, b\}$ given by the same matrix

$$P = \begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$$

and the initial distribution $\pi(a) = 2/3$, and $\pi(b) = 1/3$. This Markov chain is irreducible, and π is its unique stationary distribution.

A Markov chain is actually a particular case of a more general concept which is a probability distribution on words given by a finite automaton. Let $\mathfrak{A} = (Q, \mathcal{A})$ be a finite deterministic automaton. Let π be a probability distribution on Q. For each state $q \in Q$, consider a probability distribution on the set of edges starting in q. This is again denoted by π. Thus

$$\sum_{q \in Q} \pi(q) = 1, \quad \sum_{a \in \mathcal{A}} \pi(q, a) = 1 \text{ for all } q \in Q$$

This defines a probability distribution on the set of paths in $\mathfrak{A}$: given a path $\gamma : q_0 \stackrel{*}{\to}^{a_0} q_1 \stackrel{*}{\to}^{a_1} \cdots$, we set $\pi(\gamma) = \pi(q_0)\pi(q_0, a_0)\pi(q_1, a_1)\cdots$. This in turn defines a probability distribution on the set of words as follows: for a word w, $\pi(w)$ is the sum of $\pi(\gamma)$ over all paths (γ) with label w. The probability on words obtained in this way is a transfer of a Markov chain on the edges of the automaton.

We now give two examples of probability distributions on words. The first one is a distribution given by a finite automaton, the second one is more general.

Example 1.8.3. Consider the automaton given in Figure 1.48. Let $\pi(1) = 1$, $\pi(2) = \pi(3) = 0$, and let $\pi(1, a, 2) = \pi(1, b, 2) = 1/2$, $\pi(2, a, 2) = \pi(2, b, 2) = 1/2$, and $\pi(3, a, 3) = 0$, $\pi(2, b, 2) = 1$. The probability π induced on words by

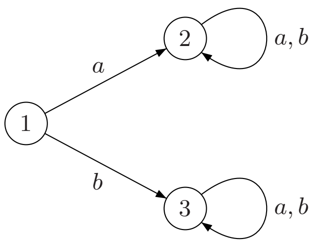


Figure 1.48. A finite automaton.

this distribution on the automaton is such that $\pi(b^n) = 1/2$ for any word $n \geq 1$. This distribution keeps an unbounded memory of the past, and is therefore not a Markov distribution.

Example 1.8.4. Let $t = abbabaabbaababba \cdots$ be the Thue–Morse word which is the fixed point of the morphism μ which maps a to ab and b to ba. Let $\mathcal{S}$ be the set of factors of t. Define a function δ on words in $\mathcal{S}$ of length at least 4 as follows. For $w \in \mathcal{S}$ with $|w| \geq 4$, set $\delta(w) = v$, where v is the unique word of $\mathcal{S}$ such that

$$\mu(v) = \begin{cases} w \text{ or } \ xwx & \text{if } |w| \text{ is even,} \\ wx \text{ or } \ xw & \text{otherwise,} \end{cases}$$

for some $x \in \{a, b\}$.

For $w \in \mathcal{S}$, define $\pi(w)$ recursively by $\pi(w) = \delta(\pi(w))/2$ if $|w| \geq 4$, and by the value given in Figure 1.49 otherwise. It is easy to verify that π is an invariant probability distribution on $\mathcal{S}$. Indeed, one has $\pi(\varepsilon) = 1$ and, for each $w \in \mathcal{S}$,

$$\pi(wa) + \pi(wb) = \pi(w), \quad \pi(aw) + \pi(bw) = \pi(w)\,.$$

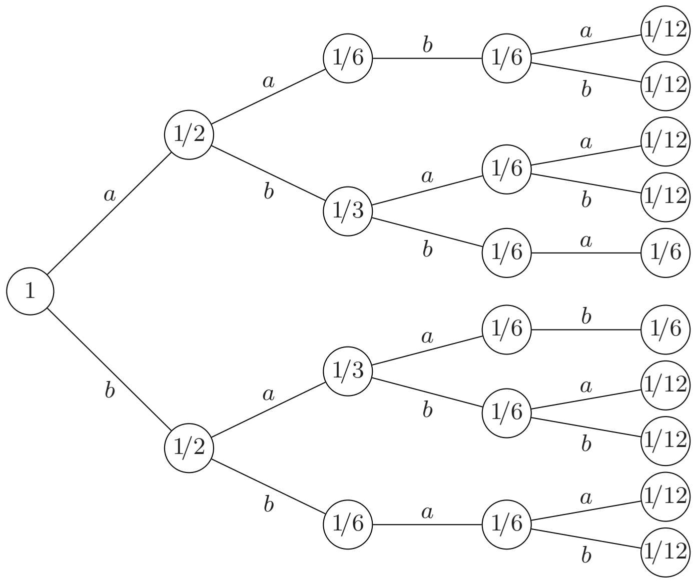


Figure 1.49. A probability distribution on the factors of the Thue-Morse infinite word.

The notion of a probability distribution on words leads naturally to the definition of a probability measure on the set $\mathcal{A}^\omega$ of infinite words. This allows to obtain a real probability distribution, instead of the distribution on each set $\mathcal{A}^n$.

Let $\mathfrak{C}$ be the set of thin cylinders, that is $\mathfrak{C} = \{w\mathcal{A}^\omega \mid w \in \mathcal{A}^*\}$, and let Σ be the σ-algebra generated by $\mathfrak{C}$. Recall that the σ-algebra generated by $\mathfrak{C}$ is the smallest family of sets containing $\mathfrak{C}$ and closed under complements and countable unions. A function μ from a σ-algebra Σ to the real numbers is said to be *σ-additive* if

$$\mu(\bigcup_n E_n) = \sum_n \mu(E_n)$$

for any family E_n of pairwise disjoint sets from Σ. A *probability measure* μ on $(\mathcal{A}^\omega, \Sigma)$ is a real valued function on Σ such that $\mu(\mathcal{A}^\omega) = 1$ and which is σ-additive.

By a classical theorem due to Kolmogorov, for each probability distribution π there exists a unique probability measure μ on $(\mathcal{A}^\omega, \Sigma)$ such that $\mu(x\mathcal{A}^\omega) = \pi(x)$.

Example 1.8.5. Let us consider again the distribution π on $\mathcal{A}^*$ with $\mathcal{A} = \{a, b\}$ of Example 1.8.3. The corresponding probability measure μ on $\mathcal{A}^\omega$ is such that $\mu(b^\omega) = 1/2$. Indeed, since $\mathcal{A}^\omega = \cup_{i\geq 0} b^i a\mathcal{A}^\omega \cup b^\omega$ one has by the property of σ-additivity

$$\mu(b^\omega) = \mu(\mathcal{A}^\omega) - \sum_{i\geq 0}\mu(b^i a\mathcal{A}^\omega) = 1 - \sum_{i\geq 0}\pi(b^i a) = 1 - \pi(a) = 1/2.$$

1.8.2. Entropy

Let U be a finite set, and let X be a random variable with values in U. Set $p(u) = \mathbf{P}(X = u)$. We define the *entropy* of X as

$$H(X) = -\sum_{u\in U} p(u)\log p(u)$$

We use the convention that $0 \log 0 = 0$. We also use the convention that the logarithm is taken in base 2. In this way, when the set U has two elements 0 and 1, with $p(0) = p(1) = 1/2$, then $H(X) = 1$. More generally, if U has n elements, then

$$H(X) \leq \log n \tag{1.8.1}$$

and the equality $H(X) = \log n$ holds if and only if $p(u) = 1/n$ for $u \in U$.

To prove this statement, we first establish the following assertion: Let p_i, q_i, for $(1 \leq i \leq n)$ be two finite probability distributions with $p_i, q_i > 0$. Then

$$\sum p_i \log p_i \geq \sum p_i \log q_i \tag{1.8.2}$$

with equality if and only if $p_i = q_i$ for $i = 1, \ldots, n$.

Indeed, observe first that $\log_e(x) \leq x - 1$ for $0 < x$ with equality if and only if $x = 1$. Thus for $1 \leq i \leq n$

$$\log_e(q_i/p_i) \leq q_i/p_i - 1$$

and consequently

$$\sum p_i \log_e(q_i/p_i) \leq \sum q_i - 1 = 0$$

This shows the inequality (1.8.2) for the logarithm in base e. Multiplying by an appropriate constant gives the general inequality. Equality holds if and only if $p_i = q_i$ for all i.

If we choose $q_i = 1/n$ for all i, inequality (1.8.2) becomes inequality (1.8.1).

If (X, Y) is a two-dimensional random variable with values in $U \times V$, we set $p(u, v) = \mathbf{P}(X = u, Y = v)$. Thus

$$H(X, Y) = -\sum_{(u,v)\in U\times V} p(u, v)\log p(u, v)$$

Finally, set $p(u|v) = \mathbf{P}(X = u|Y = v)$. Then we first define

$$H(X|v) = -\sum_{u \in U} p(u|v) \log p(u|v)$$

and for two random variables X, Y, we set

$$H(X|Y) = \sum_{v \in V} H(X|v) p(v)$$

It is easy to check that

$$H(X|Y) = -\sum_{(u,v) \in U \times V} p(u,v) \log p(u|v)$$

It can be checked that

$$H(X,Y) = H(Y) + H(X|Y) \tag{1.8.3}$$

Indeed

$$\begin{aligned} H(X,Y) &= -\sum_{u,v} p(u,v) \log p(u,v) \\ &= -\sum_{u,v} p(u,v) \log(p(u|v)p(v)) \\ &= -\sum_{u,v} p(u,v) \log p(u|v) - \sum_{u,v} p(u,v) \log p(v) \\ &= H(X|Y) + H(Y) \end{aligned}$$

It can also be verified that

$$H(X,Y) \leq H(X) + H(Y) \tag{1.8.4}$$

and that the equality holds if and only if X and Y are independent. Indeed,

$$\begin{aligned} H(X) + H(Y) &= -\sum_{u} p(u) \log p(u) - \sum_{v} p(v) \log p(v) \\ &= -\sum_{u,v} p(u,v) \log p(u) - \sum_{u,v} p(u,v) \log p(v) \\ &= -\sum_{u,v} p(u,v) \log(p(u)p(v)) \\ &\geq -\sum_{u,v} p(u,v) \log p(u,v) \end{aligned}$$

where the last inequality follows from Inequality (1.8.2).

More generally, if $(X_1, \ldots, X_n)$ is an information source, then $H_n = H(X_1, \ldots, X_n)$ is defined as the entropy of the random variable $(X_1, \ldots, X_n)$. In terms of a probability distribution π, we have

$$H_n = -\sum_{x \in \mathcal{A}^n} \pi(x) \log \pi(x)$$

Thus H_n is the entropy of the finite probability space $U = \mathcal{A}^n$.

Assume now that the source $(X_1, \ldots, X_n, \ldots)$ is stationary. The entropy of the source is defined as

$$H = \lim_{n\to\infty} \frac{1}{n} H_n$$

According to the context, we write indistinctly H or $H(X)$. We show that this limit exists. First, observe that, by Inequality (1.8.4), for all $m, n \geq 1$

$$H(X_1, , \ldots, X_{m+n}) \leq H(X_1, , \ldots, X_m) + H(X_{m+1}, , \ldots, X_{m+n})$$

Since the source is stationary, $H(X_{m+1}, , \ldots, X_{m+n}) = H(X_1, , \ldots, X_n)$. This implies

$$H_{m+n} \leq H_m + H_n$$

Thus the sequence (H_n) is a subadditive sequence of positive numbers. This implies that H_n/n has a limit.

We now give expressions for the entropy of particular sources. The entropy of a Bernoulli distribution π is

$$H = -\sum_{a\in\mathcal{A}} \pi(a)\log(\pi(a))$$

Indeed, $H_n = nH_1$ since the random variables $X_1, \ldots, X_n$ are independent and identical. As a particular case, if $\pi(a) = 1/q$ for all $a \in \mathcal{A}$, then $H = \log q$.

The entropy of an irreducible Markov chain with matrix P and stationary distribution π is

$$H = \sum_{a\in\mathcal{A}} \pi(a)H^a$$

where $H^a = -\sum_{b\in\mathcal{A}} P(a, b)\log P(a, b)$. Indeed, by Formula (1.8.3), one has

$$H(X_1, \ldots, X_n) = H(X_1) + \sum_{k=1}^{n-1} H(X_{k+1} \mid X_k)$$

By definition,

$$H(X_{n+1} \mid X_n) = \sum_{a\in\mathcal{A}} H(X_{n+1} \mid X_n = a)\mathbf{P}(X_n = a)$$

and $H(X_{n+1} \mid X_n = a) = H^a$. Since $\mathbf{P}(X_n = a)$ tends to $\pi(a)$, $H(X_{n+1} \mid X_n)$ tends to $\sum_{a\in\mathcal{A}} H^a\pi(a)$. This implies

$$\lim \frac{1}{n} H(X_1, \ldots, X_n) = \sum_{a\in\mathcal{A}} H^a\pi(a)\,.$$

Example 1.8.6. Consider again the Markov chain given by

$$P = \begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$$

and the initial distribution $\pi(a) = 2/3$, and $\pi(b) = 1/3$. Then $H^a = 1$, $H^b = 0$ and $H = 2/3$.

1.8.3. Topological entropy

For a set $\mathcal{S}$ of words, one defines the *topological entropy* of $\mathcal{S}$ as the limit

$$h(\mathcal{S}) = \limsup \frac{1}{n} \log s_n$$

where s_n is the number of words of length n in $\mathcal{S}$. This entropy is called the topological entropy to distinguish it from the entropy defined above.

Let π be a stationary distribution. Let $\mathcal{S}$ be the set of words $x \in \mathcal{A}^*$ such that $\pi(x) > 0$. Then

$$H(\pi) \leq h(\mathcal{S}).$$

Indeed, in view of Inequality (1.8.1), one has for each $n \geq 1$,

$$H_n \leq \log s_n$$

where s_n is the number of words of length n in $\mathcal{S}$. The inequality follows by taking the limit. Thus the topological entropy of $\mathcal{S}$ is an upper bound to the value of possible entropies related to a stationary probability distribution supported by $\mathcal{S}$.

In the case of a regular set $\mathcal{S}$, the entropy $h(\mathcal{S})$ can be easily computed using the Perron–Frobenius theorem. Indeed, let $\mathfrak{A}$ be a deterministic automaton recognizing $\mathcal{S}$, and let M be the adjacency matrix of the underlying graph. By the Perron–Frobenius theorem, there is a real positive eigenvalue λ which is the maximum of the moduli of all eigenvalues. One has the formula

$$h(\mathcal{S}) = \log \lambda\,.$$

This formula expresses the fact that the number s_n of words of length n in $\mathcal{S}$ grows as λ^n.

Example 1.8.7. Consider again the golden mean automaton of Example 1.3.5 which we redraw for convenience. It recognizes the set $\mathcal{S}$ of words without two consecutive a's. We have

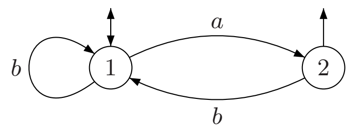


Figure 1.50. The golden mean automaton.

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

and $\lambda = (1+\sqrt{5})/2$; Thus $h(\mathcal{S}) = \log(1+\sqrt{5})/2$.

1.8.4. Distribution of maximal entropy

As we have seen in the previous section, the topological entropy of a set $\mathcal{S}$ is an upper bound to the value of possible entropies related to a stationary probability distribution supported by $\mathcal{S}$. A probability distribution π supported by $\mathcal{S}$ such that $H(\pi) = h(\mathcal{S})$ is called a *distribution of maximal entropy*. Intuitively, a distribution of maximal entropy on a set of words $\mathcal{S}$ is such that all words of $\mathcal{S}$ of given length have approximately the same probability.

We are going to show that for each rational set $\mathcal{S}$, there exists a distribution π supported by $\mathcal{S}$ of maximal entropy, i.e. such that $H(\pi) = h(\mathcal{S})$.

We consider a deterministic automaton $\mathfrak{A}$ recognizing $\mathcal{S}$. Let G be the underlying graph of $\mathfrak{A}$ with labels removed. We assume that G is strongly connected. Let Q be the set of vertices of G and let M be its adjacency matrix. The fact that G is strongly connected is equivalent to the property of M to be irreducible.

By the Perron–Frobenius Theorem, an irreducible matrix M has a real positive simple eigenvalue λ larger than or equal to the modulus of any other eigenvalue.

The number of paths of length n in G is asymptotically equivalent to λ^n. We prove that there is a labelling of G by positive real numbers which results in a Markov chain on Q of entropy $\log \lambda$. This produces a probability distribution on the words of S exactly which has maximal entropy $\log \lambda$.

Again by the Perron–Frobenius theorem, there exist a right eigenvector v and a left eigenvector w for the eigenvalue λ such that all v_i and w_i are strictly positive. We normalize v and w such that $v \cdot w = 1$. Let

$$P_{ij} = (v_j/\lambda v_i) m_{i,j}$$

and let $\pi_i = v_i w_i$. The matrix P is stochastic since

$$\sum_j P_{i,j} = \sum_j (v_j/\lambda v_i) m_{i,j} = \sum_j v_j m_{i,j}/\lambda v_i = 1\,.$$

The Markov chain with transition matrix P and initial distribution π is stationary. Indeed,

$$\sum_i \pi_i P_{i,j} = \sum_i v_i w_i (v_j/\lambda v_i) m_{i,j} = (v_j/\lambda) \sum_i w_i m_{i,j} = (v_j/\lambda)\lambda w_j = \pi_j\,.$$

The entropy of the Markov chain is $\log \lambda$. Indeed, the probability of any path γ of length n from i to j is

$$p(\gamma) = \frac{w_i v_j}{\lambda^n}.$$

This proves the existence of a distribution with maximal entropy on $\mathcal{S}$ when the graph of the automaton is strongly connected. In this case, the uniqueness can also be proved. The existence in the general case can be shown to reduce to this one.

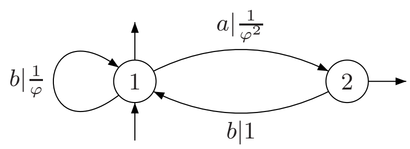


Figure 1.51. The golden mean automaton with transition probabilities.

EXAMPLE 1.8.8. Let us consider again the golden mean automaton of Example 1.3.5 which recognizes the set $\mathcal{S}$ of words without aa. We have

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad w = \begin{bmatrix} \varphi & 1 \end{bmatrix}, \quad (1+\varphi^2)v = \begin{bmatrix} \varphi \\ 1 \end{bmatrix},$$

$$P = \begin{bmatrix} \frac{1}{\varphi} & \frac{1}{\varphi^2} \\ 1 & 0 \end{bmatrix}, \quad \pi = \begin{bmatrix} \frac{\varphi^2}{1+\varphi^2} & \frac{1}{1+\varphi^2} \end{bmatrix}.$$

The values of the transition probabilities are represented on the the automaton of Figure 1.51. The probability distribution on words induced by this Markov chain is pictured in Figure 1.52.

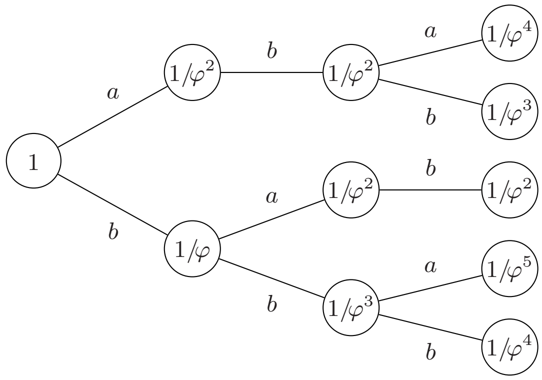


Figure 1.52. The tree of the golden mean.

As a consequence of the above construction, the distribution of maximal entropy associated with a rational set $\mathcal{S}$ is given by a finite automaton. It is even more remarkable that it can be given by the *same* automaton than the set $\mathcal{S}$ itself. This appears clearly in the above example where the automaton of Figure 1.51 is the same as the golden mean automaton of Figure 1.11.

1.8.5. Ergodic sources and compressions

Consider a source $X = (X_1, X_2, \ldots, X_n, \ldots)$ on the alphabet $\mathcal{A}$ associated to a probability distribution π. Given a word $w = a_1 \cdots a_n$ on $\mathcal{A}$, denote by

$f_N(w)$ the frequency of occurrences of the word w in the first N terms of the sequence X.

We say that the source X is *ergodic* if for any word w, the sequence $f_N(w)$ tends almost surely to $\pi(w)$. An ergodic source is stationary. The converse is not true, as shown by the following example.

EXAMPLE 1.8.9. Let us consider again the distribution of Example 1.8.3. This distribution is stationary. We have $f_N(b) = 1$ when the source outputs only b's, although the probability of b is $1/2$. Thus, this source is not ergodic.

EXAMPLE 1.8.10. Consider the distribution of Example 1.8.4. This source is ergodic. Indeed, the definition of π implies that the frequency $f_N(w)$ of any factor w in the Thue–Morse word tends to $\pi(w)$.

It can be proved that any Bernoulli source is ergodic. This implies in particular the statement known as the strong law of large numbers: if the sequence $X = (X_1, X_2, \ldots, X_n, \ldots)$ is independent and identically distributed then, setting $S_n = X_1 + \cdots + X_n$, the sequence $\frac{1}{n}S_n$ converges almost surely to the common value $\mathbf{E}(X_i)$.

More generally, any irreducible Markov chain equipped with its stationary distribution as initial distribution is an ergodic source.

Ergodic sources have the important property that typical messages of the same length have approximately the same probability, which is 2^{-nH} where H is the entropy of the source. Let us give a more precise formulation of this property, known as the *asymptotic equirepartition property.* Let $(X_1, X_2, \ldots)$ be an ergodic source with entropy H. Then for any $\epsilon > 0$ there is an N such that for all $n \geq N$, the set of words of length n is the union of two sets $\mathcal{R}$ and $\mathcal{T}$ satisfying

(i) $\pi(\mathcal{R}) < \epsilon$

(ii) for each $w \in \mathcal{T}$,
$$2^{-n(H+\epsilon)} < \pi(w) < 2^{-n(H-\epsilon)}$$

where π denotes the probability distribution on $\mathcal{A}^n$ defined by $\pi(a_1a_2\cdots a_n) = \mathbf{P}(X_1 = a_1, \ldots, X_n = a_n)$. Thus, the set of messages of length n is partitioned into a set $\mathcal{R}$ of negligible probability and a set $\mathcal{T}$ of "typical" messages having all approximately probability 2^{-nH}.

Since $\pi(w) \geq 2^{-n(H+\epsilon)}$ for $w \in \mathcal{T}$, the number of typical messages satisfies $\mathrm{Card}(\mathcal{T}) \leq 2^{n(H+\epsilon)}$. This observation allows us to see that the entropy gives a lower bound for the compression of a text. Indeed, if the messages of length n are coded unambiguously by binary messages of average length ℓ, then $\ell/n \geq H - \epsilon$ since otherwise two different messages would have the same coding. On the other hand, any coding assigning different binary words of length $n(H + \epsilon)$ to the typical messages and arbitrary values to the other messages will give a coding of compression rate approximately equal to H.

It is interesting in practice to have compression methods which are universal in the sense that they do not depend on a particular source. Some of these methods however achieve asymptotically the theoretical lower bound given by the entropy for all ergodic sources. We sketch here the presentation of one of these methods among many, the *Ziv–Lempel encoding algorithm*. This algorithm fits well in our selection of topics because it is combinatorial in nature.

We consider for a word w the factorization

$$w = x_1 x_2 \cdots x_m u$$

where

1. for each $i = 1, \ldots, m$, the word x_i is chosen the shortest possible not the set $\{x_0, x_1, x_2, \ldots, x_{i-1}\}$, with the convention $x_0 = \epsilon$.
2. the word u is a prefix of some x_i.

This factorization is called the *Ziv–Lempel factorization* of w. It appears again in Chapter 8. For example, the Fibonacci word has the factorization

$$(a)(b)(aa)(ba)(baa)(baab)(ab)(aab)(aba)\cdots$$

The coding of the word w is the sequence $(n_1, a_1), (n_2, a_2), \ldots, (n_m, a_m)$ where $n_1 = 0$ and $x_1 = a_1$, and for each $i = 2, \ldots, n$, we have $x_i = x_{n_i} a_i$, with $n_i < i$ and a_i a letter. Writing each integer n_i in binary gives a coding of length approximately $m \log m$ bits. It can be shown that for any ergodic source, the quantity $m \log m/n$ tends almost surely to the entropy of the source. Thus this coding is an optimal universal coding.

Practically, the coding of a word w uses a set D called the *dictionary* to maintain the set of words $\{x_1, \ldots, x_i\}$. We use a trie (see Section 1.3.1) to represent the set D. We also suppose that the word ends with a final symbol to avoid coding the last factor u.

```
ZLEncoding(w)
▷ returns the Ziv–Lempel encoding c of w
T ← NewTrie()
(c, i) ← (ε, 0)
while i < |w| do
      (ℓ, p) ← LongestPrefixInTrie(w, i)
      a ← w[i + ℓ]
      q ← NewVertex()
      Next(p, a) ← q              ▷ updates the trie T
      c ← c · (p, a)              ▷ appends (p, a) to c
      i ← i + ℓ + 1
return c
```

The result is a linear time algorithm. The decoding is also simple. The important point is that there is no need to transmit the dictionary. Indeed, one builds it in the same way as it was built in the encoding phase. It is convenient this time to represent the dictionary as an array of strings.

```
ZLDECODING(c)
(w, i) ← (ε, 0)
D[i] ← ε
while c ≠ ε do
    (p, a) ← CURRENT()        ▷ returns the current pair in c
    ADVANCE()
    y ← D[p]
    i ← i + 1
    D[i] ← ya                 ▷ adds ya to the dictionary
    w ← wya
return w
```

The functions CURRENT() and ADVANCE() manage the sequence c, considering each pair as a token. The practical details of the implementation are delicate. In particular, it is advised not to let the size of the dictionary grow too much. One strategy consists in limiting the size of the input, encoding it by blocks. Another one is to reset the dictionary once it has exceeded some prescribed size. In either case, the decoding algorithm must of course also follow the same strategy.

1.8.6. Unique ergodicity

We have seen that in some cases, given a formal language $\mathcal{S}$, there exists a unique invariant measure with entropy equal the topological entropy of the set $\mathcal{S}$. In particular, it is true in the case of a regular set $\mathcal{S}$ recognized by an automaton with a strongly connected graph. In this case, the measure is also ergodic since it is the invariant measure corresponding to an irreducible Markov chain. There are even cases in which there is a unique invariant measure supported by $\mathcal{S}$. This is the so-called property of *unique ergodicity* . We will see below that this situation arises for the factors of fixed points of primitive morphisms.

Example 1.8.4 is one illustration of this case. We got the result by an elementary computation. In the general case, one considers a morphism $f : \mathcal{A}^* \to \mathcal{A}^*$ that admits a fixed point $u \in \mathcal{A}^\omega$. Let M be the $\mathcal{A} \times \mathcal{A}$–matrix defined by

$$M_{a,b} = |f(a)|_b$$

where $|x|_a$ is the number of occurrences of the symbol a in the word x. We suppose the morphism f to be primitive, which by definition means that the matrix M itself is primitive. It is easy to verify that for any n, the entry $M^n_{a,b}$ is the number of occurrences of b in the word $f^n(a)$.

Since the matrix M associated to the morphism f is primitive it is also irreducible. By the Perron–Frobenius theorem, there is a unique real positive eigenvalue λ and a real positive eigenvector v such that $vM = \lambda v$. We normalize v by $\sum_{a \in \mathcal{A}} v_a = 1$.

Using the fact that M is primitive, again by the Perron–Frobenius theorem, $\frac{1}{\lambda^n} M^n_{a,b}$ tends to a matrix with rows proportional to v_b when n tends to ∞. This shows that the frequency of a symbol b in u is equal to v_b.

The value of the distribution of maximal entropy on the letters is given by $\pi(a) = v_a$. For words of length ℓ larger than 1, a similar computation can be carried out, provided one passes to the alphabet of overlapping words of length ℓ, as shown in the following example.

Example 1.8.11. Let us consider again the set $\mathcal{S}$ of factors of the Thue-Morse infinite word t (Example 1.8.4). The matrix of the morphism $\mu : a \to ab, b \to ba$ is

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} .$$

The left eigenvector is $v = [1/2\ 1/2]$ an the maximal eigenvalue is 2. Accordingly, the probability of the symbols are $\pi(a) = \pi(b) = 1/2$. To compute by this method the probability of the words of length 2, we replace the alphabet $\mathcal{A}$ by the alphabet $\mathcal{A}_2 = \{x, y, z, t\}$ with $x = aa$, $y = ab$, $z = ba$ and $t = bb$. We replace μ by the morphism μ_2 obtained by coding successively the overlapping blocks of length 2 appearing in $f(\mathcal{A}^2)$.

It is enough to truncate at length 2 in order to get a morphism that has as unique fixed point the infinite word t_2 obtained by coding overlapping blocks of length 2 in t. Thus

$$\mu_2 : \begin{array}{r@{\ }l} x & \mapsto yz \\ y & \mapsto yt \\ z & \mapsto zx \\ t & \mapsto zy \end{array}$$

has the fixed point

$$t_2 = ytzyzxytzxyz \cdots .$$

The matrix associated with μ_2 is

$$M^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} .$$

The left eigenvector is $v_2 = [1/6\ 1/3\ 1/3\ 1/6]$, consistently with the values of π given in Figure 1.49.

1.8.7. Practical estimate of the entropy

The entropy of a source given by an experiment and not by an abstract model (like a Markov chain for example) can usefully be estimated. This occurs in practice in the context of natural languages or for sources producing signals recorded by some physical measure.

The case of natural languages is of practical interest for the purpose of text compression. An estimate of the entropy H of a natural language like English implies for example that an optimal compression algorithm can encode using H bits per character in the average. The definition of a quantity which can be called 'entropy of English' deserves some commentary. First we have to clarify the nature of the sequences considered. A reasonable simplification is to assume that the alphabet is composed of the 26 ordinary letters (and thus without the upper/lower case distinction) plus possibly a blank character to separate words. The second convention is of different nature. If one wants to consider a natural language as an information source, an assumption has to be made about the nature of the source. The good approximation obtained by finite automata for the description of natural languages makes it reasonable to assume that a natural language like English can be considered as an irreducible Markov chain and thus as an ergodic source. Thus it makes sense to estimate the probabilities by the frequencies observed on a text or a corpus of texts and to use these approximations to estimate the entropy H by $H \approx H_n/n$ where

$$H_n = -\sum_k p_k \log p_k$$

and where the p_k are the probabilities of the n-grams. One has actually $H \leq H_n/n$. It is of interest to remark that the approximation thus obtained is much better than by using $H \approx h_n/n$ with

$$h_n = \log s_n$$

where s_n is the number of possible n-grams in correct English sentences. For small n the approximation is bad because some n-grams are far more frequent than others, and for large n the computation is not feasible because the number of correct sentences is too large.

One has $H \leq \log_2(26) \approx 4.7$ when considering only 26 symbols and $H \leq \log_2(27) \approx 4.76$ on 27 symbols. Further values are given in the table below leading to an upper bound $H \leq 3$. An algorithm to compute the frequencies

number of symbols	26	27
H_1	4.14	4.03
$H_2/2$	3.56	3.32
$H_3/3$	3.30	3.10

Table 1.2. Entropies of n-grams on an alphabet of 26 or 27 letters

of n-grams is easy to implement. It uses a buffer s which is initialized to the initial n symbols of the text and which is updated by shifting the symbols one place to the left and adding the current symbol of the text at the last place. This is done by the function CURRENT(). The algorithm maintains a set S of n-grams together with a map FREQ() containing the frequencies of each n-gram. A practical implementation should use a representation of sets like a hashtable, allowing to store the set in a space proportional to the size of S (and not to the number of all possible n-grams which grows too fast).

```
Entropy(n)
▷ returns the n-th order entropy H_n
S ← ∅                          ▷ S is the set of n-grams in the text
do   s ← Current()             ▷ s is the current n-gram of the text
     if s ∉ S then
          S ← S ∪ s
          Freq(s) ← 1
     else Freq(s) ← Freq(s) + 1
while there are more symbols
for s ∈ S do
     Prob(s) ← Freq(s)/Card S
return (1/n) ∑_{s∈S} Prob(s) log Prob(s)
```

Another approach leads to a better estimate of H. It is based on an experiment which uses a human being as an oracle. The idea is to scan a text through a window of $n-1$ consecutive characters and to ask a subject to guess the symbol following the window contents, repeating the question until the answer is correct. The average number of probes is an estimate of the conditional entropy $H(X_n|X_1,\dots,X_{n-1})$. The values obtained are shown in Table 1.3.

n	1	2	3	4	5	6	7
upper bound	4.0	3.4	3.0	2.6	2.1	1.9	1.3
lower bound	3.2	2.5	2.1	1.8	1.2	1.1	0.6

Table 1.3. Experimental bounds for the entropy of English

1.9. Statistics on words

In this section, we consider the problem of computing the probability of appearance of some properties on words defined using the concepts introduced at the beginning of the chapter. In particular, we shall study the average number of factors or subwords of a given type in a regular set.

1.9.1. Occurrences of factors

For any integer valued random variable X with probability distribution $p_n = \mathbf{P}(X=n)$, one introduces the generating series $f(z) = \sum_{n\geq 0} p_n z^n$. If we denote $q_n = \sum_{m\geq n} p_m$, then the generating series $g(z) = \sum_{n\geq 0} q_n z^n$ is given by the formula

$$g(z) = \frac{1-f(z)}{1-z}.$$

This implies in particular that the expectation $\mathbf{E}(X) = \sum_{n\geq 0} np_n$ of X has also the expression $\mathbf{E}(X) = g(1)$. These general observations about random variables have an important interpretation when the random variable X is the length of a prefix in a given prefix code.

Let π be a probability distribution on $\mathcal{A}^*$. For a prefix code $\mathcal{C} \subset \mathcal{A}^*$, the value $\pi(\mathcal{C}) = \sum_{x\in\mathcal{C}} \pi(x)$ can be interpreted as the probability that a long enough word has a prefix in $\mathcal{C}$. Accordingly, we have $\pi(\mathcal{C}) \leq 1$.

Let $\mathcal{C}$ be a prefix code such that $\pi(\mathcal{C}) = 1$.The average length of the words of $\mathcal{C}$ is

$$\lambda(\mathcal{C}) = \sum_{x\in\mathcal{C}} |x|\pi(x).$$

One has the useful identity

$$\lambda(\mathcal{C}) = \pi(\mathcal{P})$$

where $\mathcal{P} = \mathcal{A}^* - \mathcal{C}\mathcal{A}^*$ is the set of words which do not have a prefix in $\mathcal{C}$. Indeed, let $p_n = \pi(\mathcal{C} \cap \mathcal{A}^n)$ and $q_n = \sum_{m\geq n} p_m$. Then, $\lambda(\mathcal{C}) = \sum_{n\geq 1} np_n = \sum_{n\geq 1} q_n$. Since $\pi(\mathcal{P} \cap \mathcal{A}^n) = q_n$, this proves the claim.

The generating series $C(z) = \sum_{n\geq 0} p_n z^n$ is related to $P(z) = \sum_{n\geq 0} q_n z^n$ by

$$C(z) - 1 = P(z)(1 - z).$$

When π is a Bernoulli distribution, one may use unambiguous expressions on sets to compute probability of events definable in this way. Indeed, the unambiguous operations translate to operations on probability generating series. If $\mathcal{W}$ is set of words, we set

$$W(z) = \sum_{n\geq 0} \pi(\mathcal{W} \cap \mathcal{A}^n) z^n.$$

Then, if $\mathcal{U} + \mathcal{V}$, $\mathcal{U}\mathcal{V}$ and $\mathcal{U}^*$ are unambiguous expressions, we have

$$(U + V)(z) = U(z) + V(z), \quad (UV)(z) = U(z)V(z), \quad (U^*)(z) = \frac{1}{1 - U(z)}.$$

We give below two examples of this method.

Consider first the problem of finding the expected waiting time $T(w)$ before seeing a word w. We are going to show that it is given by the formula

$$T(w) = \frac{\pi(\mathcal{Q})}{\pi(w)} \tag{1.9.1}$$

where $\mathcal{Q} = \{q \in \mathcal{A}^* \mid wq \in \mathcal{A}^* w \text{ and } |q| < |w|\}$. Thus $\mathcal{Q}$ is the set of (possibly empty) words q such that $w = sq$ with s a nonempty suffix of w.

Let indeed $\mathcal{C}$ be the prefix code formed of words that end with w for the first time. Let $\mathcal{V}$ be the set of prefixes of $\mathcal{C}$, which is also the set of words which do not contain w as a factor. We can write

$$\mathcal{V}w = \mathcal{C}\mathcal{R}. \tag{1.9.2}$$

Moreover both sides of this equality are unambiguous. Thus, since $\pi(\mathcal{C}) = 1$, $\pi(\mathcal{V})\pi(w) = \pi(\mathcal{Q})$, whence Formula (1.9.2). Formula (1.9.2) can also be used to obtain an explicit expression for the generating series $C(z)$. Indeed, using (1.9.2), one obtains $V(z)\pi(w)z^m = C(z)Q(z)$, where m is the length of w. Replacing $V(z)$ by $(1-C(z))/(1-z)$, one obtains

$$C(z) = \frac{\pi(w)z^m}{\pi(w)z^m + Q(z)(1-z)} \tag{1.9.3}$$

The polynomial $Q(z)$ is called the *autocorrelation polynomial* of w. Its explicit expression is

$$Q(z) = 1 + \sum_{p \in P(w)} \pi(w_{n-p} \cdots w_{n-1}) z^p$$

where $P(w)$ is the set of periods of the word $w = w_0 \cdots w_{n-1}$, and w_i denotes the i-th letter of w. A slightly more general definition is given in Chapter 6.

Example 1.9.1. In the particular case of $w = a^m$ and $A = \{a, b\}$ with $\pi(a) = p$, $\pi(b) = q = 1 - p$, the autocorrelation polynomial of w is

$$R(z) = \frac{1 - p^m z^m}{1 - pz} .$$

Consequently, $\pi(\mathcal{R}) = (1-p^m)/q$ and formulas (1.9.1) and (1.9.3) become

$$T(a^m) = \frac{1-p^m}{qp^m}, \quad C(z) = \frac{(1-pz)p^n z^m}{1 - z + qp^m z^{m+1}} ,$$

so that for $p = q = 1/2$,

$$T(w) = 2^{m+1} - 2, \quad C(z) = \frac{(1-z/2)z^m/2^m}{1 - z + z^{m+1}/2^{m+1}}$$

Formula (1.9.1) can be considered as a paradox. Indeed, it asserts that with $\pi(a) = \pi(b) = 1/2$, the waiting time for the word $w = aa$ is 6 while it is 4 for $w = ab$.

Formula (1.9.1) is related with the automaton recognizing the words ending with w and consequently with Algorithm SearchFactor. We illustrate this on an example. Let $w = abaab$. The minimal automaton recognizing the words on $\{a, b\}$ ending with w for the first time is represented in Figure 1.53. The transitions of the automaton can actually be computed using the array b introduced in algorithm Border.

	0	1	2	3	4	5
b :	-1	0	0	1	1	2

For example, the transition from state 3 by letter b is to state 2 because $b[3] = 1$ and $w[1] = b$. The set R can also be read on the array b. Actually, we have $R = \{\epsilon, aab\}$ since the the border of w has length 2 ($b[5] = 2$) and $b[2] = 0$.

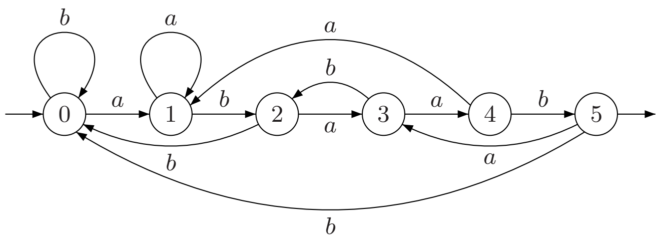


Figure 1.53. The minimal automaton recognizing the words ending with *abaab*.

As a second example, we now consider the problem of finding the probability f_n that the number of a equals the number of b for the first time in a word of length n on $\{a, b\}$ starting with a, with $\pi(a) = p$, $\pi(b) = q = 1 - p$. This is the classical problem of return to 0 in a random walk on the line.

The set of words starting with a and having as many a as b for the first time is the Dyck set $\mathcal{D}$ already studied in Section 1.6. We have already seen that $\mathcal{D} = a\mathcal{D}^*b$. Thus, the generating series $D(z) = \sum_{n\geq 0} f_{2n}z^{2n}$ satisfies

$$D^2 - D + pqz^2 = 0.$$

$$D(z) = \frac{1 - \sqrt{1 - 4pqz^2}}{2}.$$

This formula shows in particular that for $p = q$, $\pi(\mathcal{D}) = 1/2$ since $\pi(\mathcal{D}) = D(1)$. But for $p \neq q$, $\pi(\mathcal{D}) < 1/2$. An elementary application of the binomial formula gives the coefficient f_n of $D(z) = \sum_{n\geq 0} f_n z^n$

$$f_{2n} = \frac{1}{n}\binom{2n-2}{n-1}p^n q^n.$$

1.9.2. Extremal problems

We consider here the problem of computing the average value of several maxima concerning words. We assume here that the source is Bernoulli, i.e. that the successive letters are drawn independently with a constant probability distribution π.

We begin with the case of *longest run of successive occurrences of some letter* a with $\pi(a) = p$. The probability of seeing a run of k consecutive a's beginning at some given position in a word of length n is p^k. So the average number of runs of length k is approximately np^k. Let K_n be the average value of the maximal length of a run of a's in the words of length n.

Intuitively, since the longest run is likely to be unique, we have $np^{K_n} = 1$. This equation has the solution $K_n = \log_{1/p} n$. One can elaborate the above intuitive reasoning to prove that

$$\lim_{n\to\infty} \frac{K_n}{\log_{1/p} n} = 1\,. \tag{1.9.4}$$

This formula shows that, in the average, the maximal length of a run of a's is logarithmic in the length of the word.

A simple argument shows that the same result holds when runs are extended to be words over some fixed subset $\mathcal{B}$ of the alphabet $\mathcal{A}$. In this case, p is replaced by the sum of the probabilities of the letters in $\mathcal{B}$.

Another application of the above result is the computation of the average length of the longest common factor starting at the same position in two words of the same length. Such a factor x induces in two words w and w' the factorizations $w = uxv$ and $w' = yuxv'$ with $|u| = |u'|$. A factor is just a run of symbols (a, a) in the word (w, w') written over the alphabet of pairs of letters. The value of p for Equation 1.9.4 is

$$p = \sum_{a\in\mathcal{A}} \pi(a)^2\,. \tag{1.9.5}$$

The *average length of the longest repeated factor* in a word is also logarithmic in the length of the word. It is easily seen that over a q letter alphabet, the length k of the longest repeated factor is at least $\lfloor \log_q n \rfloor$ and thus the average length of the longest repeated factor is at least $\log_q n$. It can be proved that it is also $O(\log n)$.

The longest common factor of two words can be computed in linear time. An algorithm (Lengths-of-factors) is given in Chapter 2. The *average length of the longest common factor* of two words of the same length is also logarithmic in the length. More precisely, let C_n denote the average length of the longest common factor of two words of the same length n. Then

$$\lim_{n\to\infty} \frac{C_n}{\log_{1/p} n} = 2\,.$$

The intuitive argument used to derive Formula 1.9.4 can be adapted to this case to explain the value of the limit. Indeed, the the average number of common factors of length k in two words of length n is approximately n^2p^k. Solving the equation $n^2p^k = 1$ gives $k = \log_{1/p} n^2 = 2\log_{1/p} n$.

The case of subwords contrasts with the case of factors. We have already given in Section 1.2.4 an algorithm (LcsLengthArray(x, y)) which allows to compute the length of the longest common subwords of two words. The essential result concerning subwords is that the average length $c(k, n)$ of the longest common subwords of two words of length n on k symbols is $O(n)$. More precisely, there is a constant c_k such that

$$\lim_{k\to\infty} \frac{c(k, n)}{n} = c_k.$$

This result is easy to prove, even if the proof does not give a formula for c_k. Indeed, we have $c(k, n+m) \geq c(k, n) + c(k, m)$ since this inequality holds for the length of the longest common subwords of any pair of words. This implies that the sequence $c(k, n)/n$ converges (we have already met this argument in Section 1.8.2). There is no known formula for c_k but only estimates given in Table 1.4.

k	lower bound	upper bound
2	0.76	0.86
3	0.61	0.77
10	0.39	0.54
15	0.32	0.46

Table 1.4. Some upper and lower bounds for c_k

Problems

Section 1.1

1.1.1 Show that the number of words of length n on q letters with a given subword of length k is

$$\sum_{i=0}^{n-k} \binom{n-i-1}{k-1} q^i (q-1)^{n-i-k}.$$

In particular, this number does not depend on the particular word chosen as a subword. (*Hint* Consider the automaton recognizing the set of words having a given word as subword.)

1.1.2 Let $c : (A \cup \varepsilon) \times (A \cup \varepsilon) \to \mathbb{R} \cup \infty$ be a function assigning a *cost* to each pair of elements equal to a symbol or to the empty word. Assume that

(i) the restriction of c to $A \times A$ is a distance.

(ii) $c(\varepsilon, a) = c(a, \varepsilon) > 0$ for all $a \in A$.

Each transformation on a word is assigned a cost using the cost c as follows. A substitution of a symbol a by a symbol b adds a cost $c(a, b)$. An insertion of a symbol a counts for $c(\varepsilon, a)$ and a deletion for $c(a, \varepsilon)$. Let $d(u, v)$ be the distance defined as the minimal cost of a sequence of transformations that changes u into v.

Show that d is a distance on A^*. Show that d coincides with

1. the Hamming distance if $c(a, b) = 1$ for $a \neq b$ and $c(a, \varepsilon) = c(\varepsilon, a) = \infty$.
2. the subword distance if $c(a, b) = \infty$ for $a, b \in A$ and $a \neq b$, and $c(a\varepsilon) = c(\varepsilon, a) = 1$ for all $a \in A$.

Section 1.2

1.2.1 The sharp border array of a word x of length m is the array sb of size $m+1$ such that $sb[m] = b[m]$ and for $1 \le j \le m-1$, $sb[j]$ is the largest integer i such that $x[0..i-1] = x[j-i..j-1]$ and $x[j] \neq x[i]$. By convention, $sb[j] = -1$ if no such integer i exists. For example, if $x = abaababa$, the array sb is

	0	1	2	3	4	5	6	7	8
b :	-1	0	-1	1	0	-1	3	-1	3

Show that the following variant of Algorithm Border computes the array sb in linear time.

BorderSharp(x)
$\triangleright$ x has length m, sb has size $m+1$
$i \leftarrow 0$
$sb[0] \leftarrow -1$
for $j \leftarrow 1$ **to** $m-1$ **do**
    $\triangleright$ Here $x[0..i-1] = \text{border}(x[0..j-1])$
    if $x[j] = x[i]$ **then**
        $sb[j] \leftarrow sb[i]$
    else $sb[j] \leftarrow i$
        do $i \leftarrow sb[i]$
        while $i \ge 0$ and $x[j] \neq x[i]$
        $i \leftarrow b[i]$
    $i \leftarrow i+1$
$sb[m] \leftarrow i$
return b

Show that, in Algorithm SearchFactor, one may use the table sb of sharp borders instead of the table b of borders, resulting in a faster algorithm.

Section 1.3

1.3.1 This exercise shows how to answer the following questions: what is the minimal Hamming distance between a word w and the words of a regular set X and how to compute a word of X which realizes the minimum? These questions are solved by the following algorithm known as *Viterbi algorithm.*
Let $\mathcal{A} = (Q, i, T)$ be a finite automaton over the alphabet A and, for each $p \in Q$, let X_p be the set recognized by the automaton (Q, i, p).
Let $w = a_0 \cdots a_{n-1}$ be a word of length n. For a symbol $a \in A$ and $0 \le i < n$ we denote $c(a,i) = 0$ if $a = a_i$ and $c(a,i) = 1$ otherwise. We compute a function $d : Q \times \mathbb{N} \to \mathbb{N}$ defined by $d(p,i)$ is the minimal Hamming distance of the words in $X_p \cap A^i$ to the word $a_0 \cdots a_{i-1}$.

VITERBI(w)
for $i \leftarrow 0$ **to** $n-1$ **do**
    for each edge (p, a, q) **do**
        if $d(p, i-1) + c(a, i) < d(q, i)$ **then**
            $d(q, i) \leftarrow d(p, i-1) + c(a, i)$
return $\min_{t \in T} d(t, n-1)$

Show how to modify this algorithm to return a word in X that is closest to w.

1.3.2 Prove that the minimal automaton recognizing the set $\mathcal{S}(w)$ of suffixes of a word of length n has at most $2n$ states. Hint: show that for any p, q, the sets $p^{-1}\mathcal{S}(w)$ and $q^{-1}\mathcal{S}(w)$ are either disjoint or comparable. Conclude that the states of the automaton can be identified with the internal nodes of a tree with n leaves corresponding to the elements of S.

Section 1.5

1.5.1 Let $\mathfrak{A} = (Q, I, T)$ be a transducer over $\mathcal{A}$, $\mathcal{B}$ with n states. Let M be the maximal length of output labels in the edges of $\mathfrak{A}$. Suppose that $\mathfrak{A}$ is equivalent to a sequential transducer $\mathfrak{B}$, obtained by the determinization algorithm. Let $(u, q) \in B^* \times Q$ be a pair appearing in a state of $\mathfrak{B}$. Show that $|u| \leq 2n^2 M$.

Section 1.7

1.7.1 A *rational function* is a function of the form $f(z) = \sum_{n \geq 0} a_n z^n$ such that $f(z)q(z) = p(z)$ for two polynomials p, q with $q(0) = 1$. It is said to be nonnegative if $a_n \geq 0$ for all $n \geq 0$. We shall use the fact that if $f(z)$ is a nonnegative rational function such that $f \neq 0$ and $f(0) = 0$, then the radius of convergence σ of $f^*(z) = 1/(1-f(z))$ is a simple pole of f^* such that $\sigma \leq |\pi|$ for any other pole π (see the Notes Section for a reference). Moreover σ is the unique real number such that $f(\sigma) = 1$. Let M be an $n \times n$ irreducible matrix and let

$$M = \begin{bmatrix} u & v \\ w & N \end{bmatrix}$$

with N of dimension $n-1$. Let $f(z) = uz + v(I - zN)^{-1}wz^2$. Show that

$$1/(1 - f(z)) = (I - Mz)^{-1}_{1,1}.$$

Use the result quoted above on nonnegative rational functions to prove that

1. the spectral radius ρ_M of M is $1/\sigma$ where σ is such that $f(\sigma) = 1$.

2. each row of the matrix $[(\sigma - z)(I - Mz)^{-1}]_{z=\sigma}$ is a positive eigenvector of M corresponding to $1/\sigma$.

(hint: use the relation $I + (I - Mz)^{-1}M = (I - Mz)^{-1}$).

Section 1.8

1.8.1 Consider a primitive morphism $f : \mathcal{A}^* \to \mathcal{A}^*$ with a fixpoint $u \in \mathcal{A}^\omega$. We indicate here a method to compute the frequency of the factors of length ℓ in u by a faster method than the one used in Section 1.8.6. Let F_ℓ be the set of factors of length ℓ of u. Let $M^{(\ell)}$ be the $F_\ell \times F_\ell$–matrix defined by $M^{(\ell)}_{x,y} = |f(x)|_y$. Let p be an integer such that $f^p(a) > \ell - 2$ for all $a \in \mathcal{A}$. Let U be the $F_2 \times F_\ell$–matrix defined as follows. For $a, b \in \mathcal{A}$ such that $ab \in F_2$ and $y \in F_\ell$, $U_{ab,y}$ is the number of occurrences of y in $f^p(ab)$ that begin in the prefix $f^p(a)$. Show that

$$UM^{(\ell)} = M^{(2)}U,$$

that $M^{(2)}$ and $M^{(\ell)}$ have the same dominant eigenvalue ρ and that if v_2 is an eigenvector of $M^{(2)}$ corresponding to ρ, then $v_\ell = v_2 U$ is an eigenvector of $M^{(\ell)}$ corresponding to ρ.

1.8.2 Let $\mu : a \to ab, b \to ba$ be the morphism with fixpoint the Thue-Morse word. Show that for $\ell = 5$, $p = 3$, the matrix U of the previous problem (with the 12 factors of length 5 of the Thue-Morse word listed in alphabetic order) is

$$U = \begin{bmatrix} 1&0&1&1&0&1&1&0&1&1&0&1 \\ 0&1&1&0&1&1&1&1&0&1&1&0 \\ 1&1&0&1&0&1&1&0&1&1&0&1 \\ 1&0&1&1&0&1&1&0&1&1&0&1 \end{bmatrix}$$

and that the vector v_2U with $v_2 = \begin{bmatrix} 1 & 2 & 2 & 1 \end{bmatrix}$ is the vector with all components equal to 4. Deduce that the 12 factors of length 5 of the Thue-Morse word have the same frequency (see Example 1.8.4).

1.8.3 Consider the following transformation T on words: a word w is replaced by the word $T(w)$ of the same length obtained as follows: list the cyclic shitfs of w in alphabetic order as the rows $w_1, w_2, \ldots, w_n$ of an array. Then $T(w)$ is the last column of the array. For example, let $w = abracadabra$. The list of conjugates of w sorted in alphabetical order is represented below.

	1	2	3	4	5	6	7	8	9	10	11
1	a	a	b	r	a	c	a	d	a	b	r
2	a	b	r	a	a	b	r	a	c	a	d
3	a	b	r	a	c	a	d	a	b	r	a
4	a	c	a	d	a	b	r	a	a	b	r
5	a	d	a	b	r	a	a	b	r	a	c
6	b	r	a	a	b	r	a	c	a	d	a
7	b	r	a	c	a	d	a	b	r	a	a
8	c	a	d	a	b	r	a	a	b	r	a
9	d	a	b	r	a	a	b	r	a	c	a
10	r	a	a	b	r	a	c	a	d	a	b
11	r	a	c	a	d	a	b	r	a	a	b

The word $T(w)$ is the last column of the array. Thus in our example $T(w) = rdarcaaaabb$. Show that $w \mapsto T(w)$ is a bijection.

1.8.4 Let $u_{\mathcal{S}}(z)$ be the generating series of the number of words of length n in $\mathcal{S}$, that is

$$u_{\mathcal{S}}(z) = \sum_{w \in \mathcal{S}} z^{|w|} .$$

Show that

$$u_{\mathcal{S}}(z) = S(qz),$$

where $S(z)$ is the generating series for the uniform Bernoulli distribution on q symbols.

1.8.5 Show that the generating series of the set $\mathcal{F}$ of words over $\{a, b\}$ without factor $w = a^n$ is

$$u_{\mathcal{F}}(z) = \frac{1 - z^n}{1 - 2z + z^{n+1}}.$$

Notes

Several textbooks treat the subject of algorithms on words in much more detail than we did here. In the first place, several general textbooks on algorithms like Aho, Hopcroft, and Ullman (1975) or Sedgewick (1983) include automata and pattern matching algorithms among many other topics. In the second place, several books like Crochemore and Rytter (1994), Gusfield (1997) or Baeza-Yates and Ribero-Neto (1999) are entirely dedicated to word algorithms.

Words. The distance introduced in Problem 1.1.2 is known as the *edit distance* or also the *alignment distance*. It has been introduced first by Levenshtein (1965) and it is used in many ways in bioinformatics (see Sankoff and Kruskal 1983).

The algorithm VITERBI in Problem 1.3.1 is used in the context of convolutional error-correcting codes (see McEliece 2002). It appears again in Chapter 4.

Elementary algorithms. The algorithm Border computing the border of a word in linear time and the linear time algorithm SearchFactor that checks whether a word is a factor of another (Algorithm SearchFactor) are originally due to Knuth, Morris, and Pratt (1977). This algorithm is the first one of a large family of algorithms constituting the field of *pattern matching* algorithms. See Crochemore, Hancart, and Lecroq (2001) for a general presentation. The algorithm BorderSharp of Problem 1.2.1 is from Knuth et al. (1977).

The quadratic algorithm to compute a longest common subword (Algorithm Lcs) is usually credited to Hirschberg (1977), although many authors discovered it independently (see Sankoff and Kruskal 1983). It is not known whether there exists or not a linear algorithm. An algorithm working in time $O(p \log n)$ on two words of length n with p pairs of matching positions is due to Hunt and Szymanski (1977).

The linear algorithm CircularMin thato computes the least conjugate of a word is due to Booth (1980). Several refinements where proposed, see Shiloach (1981), Duval (1983), Apostolico and Crochemore (1991). The algorithm giving the factorization in Lyndon words (Algorithm LyndonFactorization) is due to Fredricksen and Maiorana (1978), see also Duval (1983).

Tries and automata. Tries are treated in many textbooks on algorithms (e.g. Aho, Hopcroft, and Ullman 1983). Our treatment of the implementation of automata and pattern matching is also similar to that of most textbooks (see for example Aho et al. 1975).

The exact complexity of the minimisation problem for deterministic finite automata is not yet known. Moore's algorithm appears in a historical paper (Moore 1956). Hopcroft's minimization algorithm appears first in Hopcroft (1971). The linear minimization algorithm for DAWG's is from Revuz (1992). It can be considered as an extension of the tree isomorphism algorithm in Aho et al. (1975).

Gilbreath's card trick (Example 1.3.9) is described as follows in Chapter 9 of Gardner (1966): *Consider a deck of* $2n$ *cards ordered in such a way that red and black cards alternate. Cut the deck into two parts and give it a riffle shuffle. Cut it once more, this time not completely arbitrarily but at a place where two cards of the same colour meet. Square up the deck.*

Then for every $i = 1, \ldots, n$ *the pair consisting of the* $(2i-1)$*-th and the* $2i$*-th card is of the form* (red, black) *or* (black, red). The property of binary sequences underlying the card trick is slightly less general than Formula 1.3.1.

The source of Exercise 1.3.2 is Blumer, Blumer, Haussler, Ehrenfeucht, Chen, and Seiferas (1985). The automaton can be used in several contexts, including as a transducer called the *suffix transducer* (see Chapter 2).

Pattern Matching. The equivalence of regular expressions and finite automata is a classical result known as *Kleene's theorem.* We present here only one direction of this result, namely the construction of finite automata from regular expressions. This transformation is used in many situations. Actually, regular expressions are often used as a specification of some pattern and the equivalent finite automaton can be considered as an implementation of this specification. The converse transformation gives rise to algorithms that are less frequently used. One case of use is for the computation of generating series (see Section 1.9).

There is basically only one method to transform a regular expression into a finite automaton which operates by induction on the structure of the regular expression. However, several variants exist. The one presented here is due to Thompson Thompson 1968. It uses ε-transitions and produces a normalized automaton that has a unique initial state with no edge entering it and a unique terminal state with no edge going out of it. Another variant produces an automaton without ε-transitions (see Eilenberg 1974 for example). The resulting automaton has in general fewer states than the one obtained by Thomson's algorithm. Yet another variant produces directly a deterministic automaton (see Berry and Sethi 1986 or Aho, Sethi, and Ullman 1986).

Transducers. The notion of a rational relation and of a transducer dates back to the origins of automata theory although there are few books treating extensively this aspect of the theory. Eilenberg's book (Eilenberg 1974) represents a significant date in the clarification of the concepts and notation. Later books treating transducers include Berstel (1979) and Sakarovich (2004). A word on the terminology concerning what we call here sequential transducers. The term "sequential machine" (also called "Mealy machine") is in general used only in the case of sequential letter-to-letter transducers. The version using (possibly empty) word outputs is often called a "generalized sequential machine" (or gsm). A further generalization, used by Schützenberger (1977), introduces a class of tranducers called *subsequential* which allow the additional use of a terminal suffix. We simply call here *sequential* these subsequential transducers.

Any sequential function is a rational function but the converse is of course not true. Several characterizations of rational functions, in particular special classes of transducers realize rational fuctions. Among these, so-called *bimachines* used in Chapter 3. The determinization algorithm has been first studied in Schützenberger (1977) and Choffrut (1979). In particular, the characterization of sequential transducers by the twinning property is in Choffrut (1977, 1979). See also Reutenauer (1990).

The source of Problem 1.5.1 is Béal and Carton (2002). It can be checked in polynomial time whether a transducer is equivalent to a sequential one (see Weber and Klemm 1995 or Béal and Carton 2002). The normalization algorithm has been first considered by Choffrut (1979) and subsequently by Mohri (1994) and by Béal and Carton (2001).

A quite different algorithm relying on shortest paths algorithms has been proposed by Breslauer (1998). For recent developments, see the survey on minimization algorithms of transducers in Choffrut (2003).

Parsing. The section on parsing follows essentially Aho et al. (1986). The Dyck language is named after the group theorist Walther von Dyck (Dyck 1882). Context-free grammars are an important model for modelling hierarchically structured data. They appear in various equivalent forms, such recursive transition networks (RTN) used in natural language processing (see Chapter 3). Parsers are ubiquous is data processing systems, including natural language processing. The abstract model of a parser is a pushdown automaton which is a particular case of the general model of Turing machine.

It is a remarkable fact that a large class of grammars can be parsed in one pass, from left to right, and in linear time. This was first established by Knuth (1965) who introduced in particular the LR analysis described here.

Word enumeration. A detailed proof of the Perron–Frobenius theorem can be found in Gantmacher (1959). The proof given here is due to Wielandt, whence the name of "Wielandt function", also called Collatz–Wielandt function (see Allouche and Shallit (2003)).

Problem 1.7.1 presents the connection between the theorem of Perron–Frobenius and related statements concerning the poles of nonnegative rational functions. For a proof of the statement appearing at the beginning of the problem, see Eilenberg (1974) or Berstel and Reutenauer (1984). This approach gives a proof of the Perron–Frobenius theorem that differs from the one given in Section 1.7.2. See McCluer (2000) for a survey on several possible proofs of this theorem.

Probability. Our presentation of probability distributions on words is inspired by Welsh (1988), Szpankowski (2001) and Shields (1969). A proof of the fundamental theorem of Markov chains can be found in most textbooks on probability theory (see e. g. Feller (1968), chap. XV). The theorem of Kolmogorov on probability measures on infinite words can be found in Feller 1971, chap. IV. The notion of entropy is due to Shannon (1948). Many textbooks contain a presentation of the main properties of entropy (see e. g. Ash 1990). The computation of the distribution of maximal entropy (Section 1.8.4) is originally due to Shannon. Our presentation follows Lind and Marcus 1996 (chap. 13).

The computation of the frequencies of factors in fixpoints of substitutions is reproduced from Queffélec (1987). The method described in Problem 1.8.1 is also from Queffélec (1987).

The asymptotic equirepartition property of ergodic sources is known as the Shannon–McMillan theorem. See Shields (1969) for a proof. The Ziv–Lempel coding originally appears in Ziv and Lempel (1977). A complete presentation of this popular coding can be found in Bell, Cleary, and Witten (1990) with several variants.

The entropy of english has been studied by Shannon. In particular, tables 1.2 and 1.3 are from Shannon (1951). They are reproduced in several manuals on text compression (see e.g. Welsh 1988 or Bell et al. 1990).

Statistics on words. Events defined by prefix codes (Section 1.9.1) are presented in Feller (1968) under the name of recurrent events. Formula (1.9.1) appears already in the paper Schützenberger (1964). The name of autocorrelation polynomial appears in Guibas and Odlyzko (1981b). Formula (1.9.4) is due to Erdős and Rényi (1970) . For a proof, see Waterman (1995) (chap. 11). Formula (1.9.5) is due to Arratia, Morris, and Waterman (1988) (see Waterman 1995, chap. 11). Table 1.4 is from Sankoff and Kruskal (1983).

The transformation described in Problem 1.8.3 is known as the Burrows–Wheeler transformation (see Manzini (2001)). It is the basis of a text compression method. Indeed, the idea is that adjacent rows of the table of cyclic shifts will often begin by a long common prefix and $T(w)$ will therefore have long runs of identical symbols. For example, in a text in english, most rows beginning with 'nd' will end with 'a'.

CHAPTER 2

Structures for Indexes

2.0. Introduction

The chapter presents data structures used to memorize the suffixes of a text and some of their applications. These structures are designed to give a fast access to all factors of the text, and this is the reason why they have a fairly large number of applications in text processing.

Two types of objects are considered in this chapter, digital trees and automata, together with their compact versions. Trees put together common prefixes of the words in the set. Automata gather in addition their common suffixes. The structures are presented in order of decreasing size.

The representation of all the suffixes of a word by an ordinary digital tree called a suffix trie (Section 2.1) has the advantage of being simple but can lead to a memory size that is quadratic in the length of the considered word. The compact tree of suffixes (Section 2.2) is ensured to hold in linear memory space.

The minimization (related to automata) of the suffix trie gives the minimal automaton accepting the suffixes and is described in Section 2.4. Compaction and minimization yields the compact suffix automaton of Section 2.5.

Most algorithms that build the structures presented in the chapter work in time $O(n \times \log \operatorname{Card} \mathcal{A})$, for a text of length n, assuming that there is an ordering on the alphabet $\mathcal{A}$. Their execution time is thus linear when the alphabet is finite and fixed. Locating a word of length m in the text then takes $O(m \times \log \operatorname{Card} \mathcal{A})$ time.

The main application of presented techniques is to provide the basis for implementing indexes, which is described in Section 2.6. But the direct access to factors of a word authorizes a great number of other applications. We briefly mention how to detect repetitions or forbidden words in a text (Section 2.7). Structures can also be used to search for fixed patterns in texts because they can be regarded as pattern matching machines (see Section 2.8). This method is extended in a particularly effective way for searching conjugates (or rotations) of a pattern in Section 2.8.3.

2.1. Suffix trie

The tree of suffixes of a word, called its *suffix trie*, is a deterministic automaton that accepts the suffixes of the word and in which there is a unique path from the initial state to any state. It can be viewed as a digital tree which represents the set of suffixes of the word. Standard methods can be used to implement these automata, but its tree structure authorizes a simplified representation.

Considering a tree implies that the terminal states of the tree are in one-to-one correspondence with the words of the accepted language. The tree is thus finite only if its language is also finite. Consequently, the explicit representation of the tree has an algorithmic interest only for finite languages.

Sometimes one forces the tries to have for terminal states only the external nodes of the tree. With this constraint, a language $\mathcal{L}$ is representable by a trie only if no proper prefix of a word of $\mathcal{L}$ is in $\mathcal{L}$. It results from this remark that if y is a nonempty word, only $\operatorname{Suff}(y) \setminus \{\varepsilon\}$ is representable by a trie having this property, and this takes place only when the last letter of y appears only once in y. For this reason one frequently adds for this purpose a marker at the end of the word. We prefer to attach an output to the nodes of the tree, which is

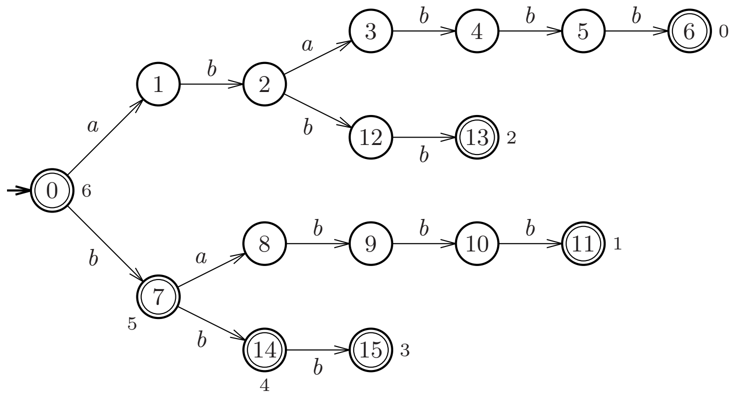


Figure 2.1. Trie $\mathfrak{T}(ababbb)$ of suffixes of *ababbb*. Terminal states are marked by double circles. The output associated with a terminal state is the position of the corresponding suffix on the word *ababbb*. The empty suffix, by convention, is associated with the length of the word.

in conformity with the concept used, that of automaton. Only the nodes whose output is defined are regarded as terminals. In addition, there are only very slight differences between the implementations of the two features.

The suffix trie of a word y is denoted by $\mathfrak{T}(y)$. Its nodes are the factors of y, ε is the initial state, and the suffixes of y are the terminal states. The transition function δ of $\mathfrak{T}(y)$ is defined by $\delta(u, a) = ua$ if ua is a factor of y and $a \in \mathcal{A}$. The output of a terminal state, which is then a suffix, is the position of this suffix in y. An example of suffix trie is displayed in Figure 2.1.

A classical construction of $\mathfrak{T}(y)$ is carried out by adding successive suffixes of y in the tree under construction, from the longest suffix, y itself, until the shortest, the empty word.

The current operation consists in inserting $y[i \mathinner{..} n-1]$, the suffix at position i, in the structure which contains already all the longer suffixes. It is illustrated by Figure 2.2. We call *head* of a suffix its longest prefix common to a suffix occurring at a smaller position. It is also the longest prefix of $y[i \mathinner{..} n-1]$ that is the label of some path starting at the initial state of the automaton in construction. The target state of the path is called a *fork* (two divergent paths start from this state). If $y[i \mathinner{..} k-1]$ is the head of the suffix at position i ($y[i \mathinner{..} n-1]$) the word $y[k \mathinner{..} n-1]$ is called the *tail* of the suffix.

More precisely, one calls fork any state of the automaton which is of (out-degree) degree at least 2, or which is both of degree 1 and terminal.

Algorithm SuffixTrie builds the suffix trie of y. Its code is given below. It is supposed that the automaton is represented by lists of successors (adjacency list). The list associated with state p is denoted by $adj[p]$ and contains pairs of the form (a, q) where a is a letter and q a state. The function Target implements transitions of the automaton, so Target(p, a) is q when $(a, q) \in$

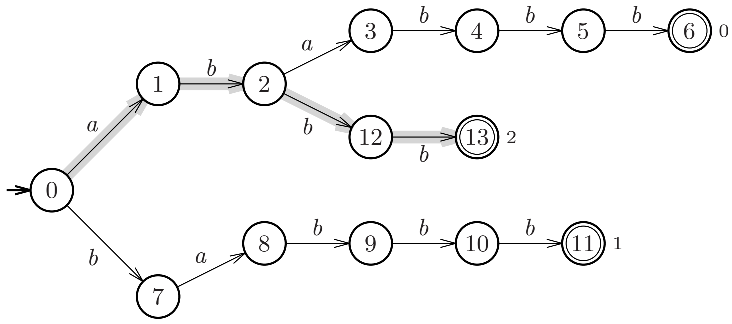


Figure 2.2. The trie $\mathfrak{T}(ababbb)$ (see Figure 2.1) during its construction, just after the insertion of suffix *abbb*. The fork, state 2, corresponds to the head, *ab*, of the suffix. It is the longest prefix of *abbb* that appears before the concerned position. The tail of the suffix is *bb*, label of the path grafted at this stage from the fork and leading to states 12 and 13.

$adj[p]$ (or more generally when $(au, q) \in adj[p]$ for some word u, as considered in next sections). States of the automaton have the attribute *output* whose value is a position. When creating a state, the procedure NewState allocates an empty adjacency list and set as undefined the value of the attribute *output*. Only the output of terminal states is set by the algorithm. The procedure NewAutomaton creates a new automaton, say M, with only one state, its initial state $initial(M)$.

In the algorithm, the insertion of the current suffix $y[i \,.\,.\, n-1]$ in the automaton M under construction, starts with the computation of its head, $y[i \,.\,.\, k-1]$, and of the associated fork, $p = \delta(initial(M), y[i \,.\,.\, k-1])$, from which is grafted the tail of the suffix (denoting by δ the transition function of M). The value of the function SlowFind applied to the pair $(initial(M), i)$ is precisely the sought pair (p, k). The creation of the path of label $y[k \,.\,.\, n-1]$ from p together with the definition of the output of its target is carried out at lines 5–9.

The last step of the execution, insertion of the empty suffix, just consists in defining the output of the initial state, which value is $n = |y|$ by convention (line 10).

```
SuffixTrie(y, n)
 1  M ← NewAutomaton()
 2  for i ← 0 to n − 1 do
 3      (fork, k) ← SlowFind(initial(M), i)
 4      p ← fork
 5      for j ← k to n − 1 do
 6          q ← NewState()
 7          adj[p] ← adj[p] ∪ {(y[j], q)}
 8          p ← q
 9      output[p] ← i
10  output[initial(M)] ← n
11  return M
```

```
SlowFind(p, i)
for k ← i to n − 1 do
    if Target(p, y[k]) is undefined then
        return (p, k)
    p ← Target(p, y[k])
return (p, n)
```

Proposition 2.1.1. *Algorithm* SuffixTrie *builds the suffix trie of a word of length n in time $\Theta(n^2)$.*

Proof. The correctness is easy to check on the code of the algorithm.

For the evaluation of execution time, let us consider stage i. Let us suppose that $y[i \mathinner{.\,.} n-1]$ has head $y[i \mathinner{.\,.} k-1]$ and has tail $y[k \mathinner{.\,.} n-1]$. The call to SlowFind (line 3) performs $k-i$ operations and the **for** loop at lines 5–8 does $n-k$ ones, which gives a total of $n-i$ operations. Thus the **for** loop indexed by i at lines 2–9 executes $n+(n-1)+\cdots+1$ operations, which gives a total execution time $\Theta(n^2)$. ∎

2.1.1. Suffix links

It is possible to accelerate the preceding construction by improving the search for forks. The technique described here is used in the following section where it leads to an actual gain in the execution time that is measurable with the asymptotic evaluation.

Let av be a suffix of y with a nonempty head az ($a \in \mathcal{A}$). The prefix z of v thus appears in y before the considered occurrence. This implies that z is a prefix of the head of suffix v. The search for this head and the corresponding fork can thus be done by starting in state z instead of starting systematically with the initial state as done in the preceding algorithm. However, this supposes that, the state az being known, one has a fast access to state z. For that, one introduces a function defined on the states of the automaton and called the *suffix link function*. It is denoted by s_y and defined, for each state az ($a \in \mathcal{A}$,

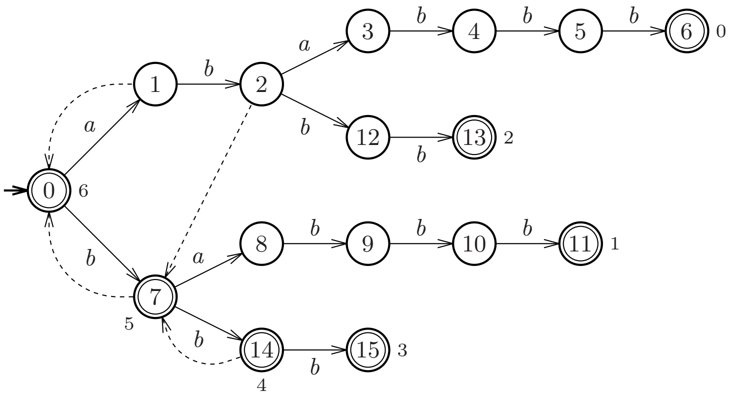


Figure 2.3. The trie $\mathfrak{T}(ababbb)$ with suffix links of forks and of their ancestors indicated by dashed arrows.

$z \in \mathcal{A}^*$), by $s_y(az) = z$. State z is called the *suffix link* of state az. Figure 2.3 displays in dashed arrows the suffix link function of the trie of Figure 2.1.

The algorithm SlowFind-bis uses the suffix link function for the computation of the suffix trie of y. The function is implemented by a table named $s\ell$. Suffix links are actually computed there only for the forks and their ancestors, except for the initial state. The rest is just an adaptation of algorithm SlowFind that includes the definition of the suffix link table $s\ell$. The new algorithm is called SlowFind-bis.

```
SlowFind-bis(p, k)
while k < n and Target(p, y[k]) is defined do
    q ← Target(p, y[k])
    (e, f) ← (p, q)
    while e ≠ initial(M) and sℓ[f] is undefined do
        sℓ[f] ← Target(sℓ[e], y[k])
        (e, f) ← (sℓ[e], sℓ[f])
    if sℓ[f] is undefined then
        sℓ[f] ← initial(M)
    (p, k) ← (q, k + 1)
return (p, k)
```

Algorithm SuffixTrie-bis is an adaptation of SuffixTrie. It uses the function SlowFind-bis instead of SlowFind.

```
SuffixTrie-bis(y, n)
 1  M ← NewAutomaton()
 2  sℓ[initial(M)] ← initial(M)
 3  (fork, k) ← (initial(M), 0)
 4  for i ← 0 to n − 1 do
 5      k ← max{k, i}
 6      (fork, k) ← SlowFind-bis(sℓ[fork], k)
 7      p ← fork
 8      for j ← k to n − 1 do
 9          q ← NewState()
10          adj[p] ← adj[p] ∪ {(y[j], q)}
11          p ← q
12      output[p] ← i
13  output[initial(M)] ← n
14  return M
```

Proposition 2.1.2. *Algorithm* SuffixTrie-bis *builds the suffix trie of y in time $\Theta(\operatorname{Card} Q)$, where Q is the set of states of $\mathfrak{T}(y)$.*

Proof. The operations of the main loop, apart from line 6 and the **for** loop at lines 8–11, are carried out in constant time, which gives a time $O(n)$ for their total execution.

Each operation of the internal loop of the algorithm SlowFind-bis, which is called at line 6, leads to create a suffix link. The total number of links being bounded by $\operatorname{Card} Q$, the cumulated time of all the executions of line 6 is $O(\operatorname{Card} Q)$.

The execution time of the loop 8–11 is proportional to the number of states that it creates. The cumulated time of all the executions of lines 8–11 is thus still $O(\operatorname{Card} Q)$.

Consequently, the total time of the construction is $\Theta(\operatorname{Card} Q)$, which is the announced result. ∎

The size of $\mathfrak{T}(y)$ can be quadratic. This is the case for example for a word whose letters are pairwise distinct. For this category of words algorithm SuffixTrie-bis is in fact not faster than SuffixTrie.

For certain words, it is enough to prune the hanging branches (below the forks) of $\mathfrak{T}(y)$ to obtain a structure of linear size. This kind of pruning gives the tree called the position tree of y (see Figure 2.4), which represents the shortest factors occurring only once in y or the suffixes that identify other positions. However, considering the position tree does not completely solve the question of memory space for the structure that can still have a quadratic size. It can be checked for example that the word $a^k b^k a^k b^k$ ($k \in \mathbb{N}$) of length $4k$ has a pruned suffix trie that contains more k^2 nodes.

The structure of compact tree of the following section is a solution to obtain a structure of linear size. The automata of Sections 2.4 and 2.5 provide another type of solution.

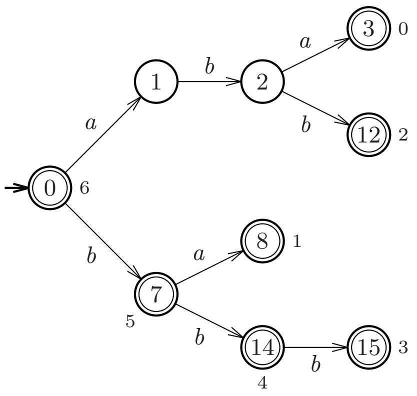


Figure 2.4. Position tree of *ababbb*. It accepts the shortest factors which identify positions on the word, and some suffixes.

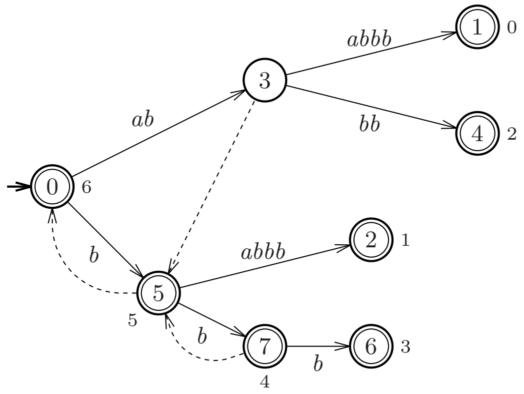
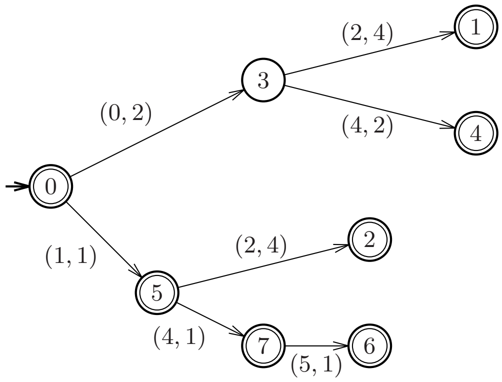


Figure 2.5. The (compact) suffix tree $\mathfrak{S}(ababbb)$ with its suffix links.

2.2. Suffix tree

The compact suffix trie of word y, simply called its *suffix tree* and denoted by $\mathfrak{S}(y)$, is obtained by removing the nodes of degree 1 which are not terminal in its suffix trie. This operation is called the compaction of the trie. The compact tree preserves only the forks and the terminal nodes of the suffix trie. Labels of edges then become words of positive variable length. Observe that if two edges starts from a same node and are labeled by the words u and v then the first letters of these words are distinct, *i.e.*, $u[0] \neq v[0]$. This comes from the fact that the suffix trie is a deterministic automaton.

Figure 2.5 shows the compact suffix tree obtained by compaction of the suffix trie of Figure 2.1.



i	0	1	2	3	4	5
$y[i]$	a	b	a	b	b	b

Figure 2.6. Representation of labels in the (compact) suffix tree $\mathfrak{S}(ababbb)$. (To be compared with the tree in Figure 2.5.) Label $(2, 4)$ of edge $(3, 1)$ represents the factor of length 4 at position 2 in y, *i.e.*, the word $abbb$.

PROPOSITION 2.2.1. *The compact suffix tree of a word of length $n > 0$ has between $n + 1$ and $2n$ nodes. The number of forks of the tree is between 1 and n.*

Proof. The tree contains $n + 1$ distinct terminal nodes corresponding to the $n + 1$ suffixes they represent. This gives the lower bound.

Each fork that is not terminal has at least two children. For a fixed number of external nodes, the maximum number of these forks is obtained when each one has exactly two children. In this case, one obtains at most n forks (terminal or not). As for $n > 0$ the initial state is both a fork and a terminal node one obtains the bound $(n + 1) + n - 1 = 2n$ on the total number of nodes. ■

The fact that the compact suffix tree has a linear number of nodes does not imply the linearity of its representation, because this also depends on the total size of labels of the edges. The example of a word of length n that has n distinct letters shows that this size can well be quadratic. Nevertheless, labels of edges being all factors of y, each one can be represented by a pair position-length (or also starting position-end position), provided that the word y resides in memory with the tree to allow an access to the labels. A word u that is the label of an edge (p, q) is represented by the pair $(i, |u|)$ where i is the position of some occurrence of u in y. We write $label(p, q) = (i, |u|)$ and assume that the implementation of the tree provides a direct access to this label. This representation of labels is illustrated in Figure 2.6 for the tree of Figure 2.5.

PROPOSITION 2.2.2. *Representing labels of edges by pairs of integers, the total size of the compact suffix tree of a word is linear in its length, i.e., the size of $\mathfrak{S}(y)$ is $\Theta(|y|)$.*

Proof. The number of nodes of $\mathfrak{S}(y)$ is $\Theta(|y|)$ according to Proposition 2.2.1. The number of edges of $\mathfrak{S}(y)$ is one unit less than the number of nodes. The assumption on the representation of labels of edges implies that each edge occupies a constant space. What gives the result. ■

The suffix link function introduced in the preceding section finds its complete usefulness in the construction of compact suffix tries. It allows a fast construction when, moreover, the algorithm SlowFind of the preceding section is replaced by the algorithm FastFind hereafter that has a similar function. The possibility of retaining only the forks of the tree, in addition to terminal states, rests on the following lemma.

Proposition 2.2.3. *In a suffix trie, the suffix link of a nonempty fork is a fork.*

Proof. For a nonempty fork, there are two cases to consider according to whether the fork, say au ($a \in \mathcal{A}$, $u \in \mathcal{A}^*$) has degree at least 2, or has degree 1 and is terminal.

Let us suppose first that the degree of au is at least 2. For two distinct letters b and c, the words aub and auc are factors of y. The same property then holds for $u = s_y(au)$ which is thus of degree at least 2 and therefore is a fork.

If the fork au has degree 1 and is terminal, then aub is a factor of y for some letter b and simultaneously au is a suffix of y. Thus, ub is a factor of y and u is a suffix of y, which shows that $u = s_y(au)$ is also a fork. ■

The following property is used as a basis for the computation of suffix links in the algorithm SuffixTree that builds the suffix tree. We denote by δ the transition function of $\mathfrak{S}(y)$.

Lemma 2.2.4. *Let (p, q) be an edge of $\mathfrak{S}(y)$ and $y[j\,..\,k-1]$, $j < k$, be its label. If q is a fork of the tree, then*

$$s_y(q) = \begin{cases} \delta(p, y[j+1\,..\,k-1]) & \textit{if } p \textit{ is the initial state,} \\ \delta(s_y(p), y[j\,..\,k-1]) & \textit{otherwise.} \end{cases}$$

Proof. As q is a fork, $s_y(q)$ is defined according to Proposition 2.2.3. If p is the initial state of the tree, *i.e.*, if $p = \varepsilon$, one has $s_y(q) = \delta(\varepsilon, y[j+1\,..\,k-1])$ by definition of s_y.

In the other case, there is a single path from the initial state ending at p because $\mathfrak{S}(y)$ is a tree. Let av be the nonempty label of this path with $a \in \mathcal{A}$ and $v \in \mathcal{A}^*$ (*i.e.*, $p = av$). One has $\delta(\varepsilon, v) = s_y(p)$ and $\delta(\varepsilon, v \cdot y[j\,..\,k-1]) = s_y(q)$. It follows that $s_y(q) = \delta(s_y(p), y[j\,..\,k-1])$ (the automaton is deterministic), as announced. ■

2.2.1. Construction

The strategy we select to build the suffix tree of y consists in successively inserting the suffixes of y in the structure, from the longest to the shortest, as in

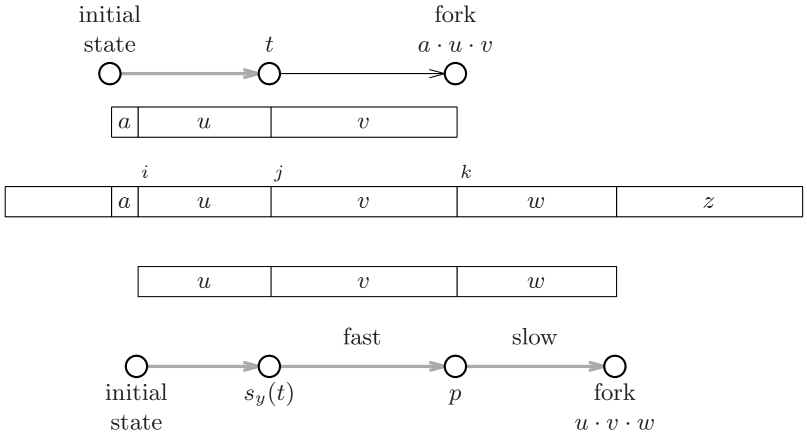


Figure 2.7. Schema for the insertion of the suffix $y[i\,.\,.\,n-1] = u \cdot v \cdot w \cdot z$ of y in the (compact) suffix tree during its construction, when the suffix link is not defined on the fork $a \cdot u \cdot v$. Let t be the parent of this fork and v be the label of the associated edge. One first computes $p = \delta(s_y(t), v)$ using FastFind, then the fork of the suffix using SlowFind as in Section 2.1.

the preceding section. As for the algorithm SuffixTrie-bis, the insertion of the tail of the running suffix is done after a slow find starting from the suffix link of the current fork. When this link does not exist it is created (lines 6–11 of SuffixTree) by using the equality of the preceding statement. Calculation is performed by the algorithm FastFind that satisfies

$$\text{FastFind}(r, j, k) = \delta(r, y[j\,.\,.\,k-1])$$

for r state of the tree and j, k positions on y for which

$$r \cdot y[j\,.\,.\,k-1] \text{ is a factor of } y.$$

The diagram for the insertion of one suffix inside the tree in construction is presented in Figure 2.7.

```
SuffixTree(y, n)
M ← NewAutomaton()
sℓ[initial(M)] ← initial(M)
(fork, k) ← (initial(M), 0)
for i ← 0 to n − 1 do
    k ← max{i, k}
    if sℓ[fork] is undefined then
        t ← parent of fork
        (j, ℓ) ← label(t, fork)
        if t = initial(M) then
            ℓ ← ℓ − 1
        sℓ[fork] ← FastFind(sℓ[t], k − ℓ, k)
    (fork, k) ← SlowFindC(sℓ[fork], k)
    if k < n then
        q ← NewState()
        adj[fork] ← adj[fork] ∪ {((k, n − k), q)}
    else q ← fork
    output[q] ← i
output[initial(M)] ← n
return M
```

Algorithm SlowFindC is merely adapted from algorithm SlowFind to take into account the fact that labels of edges are words. However, when the sought target falls in the middle of an edge it is now necessary to cut this edge. Let us notice that Target(p, a), if it exists, is the state q for which a is the first letter of the label of the edge (p, q). Labels can be words of length strictly more than 1; thus, it is not true in general that Target$(p, a) = \delta(p, a)$.

```
SlowFindC(p, k)
while k < n and Target(p, y[k]) is defined do
    q ← Target(p, y[k])
    (j, ℓ) ← label(p, q)
    i ← j
    do  i ← i + 1
        k ← k + 1
    while i < j + ℓ and k < n and y[i] = y[k]
    if i < j + ℓ then
        adj[p] ← adj[p] \ {((j, ℓ), q)}
        r ← NewState()
        adj[p] ← adj[p] ∪ {((j, i − j), r)}
        adj[r] ← adj[r] ∪ {((i, ℓ − i + j), q)}
        return (r, k)
    p ← q
return (p, k)
```

The improvement on the execution time of the construction of a suffix tree by the algorithm SuffixTree rests, in addition to the compaction of the data structure, on an additional algorithmic element: the implementation of FastFind. Resorting to the particular algorithm describes by the code below is essential to obtain the execution time stated in Theorem 2.2.7.

The algorithm FastFind is used to compute a fork. It is applied to state r and word $y[j\,..\,k-1]$ only when

$$r \cdot y[j\,..\,k-1] \text{ is a factor of } y.$$

In this case, from state r there is a path whose label is prefixed by $y[j\,..\,k-1]$. Moreover, as the automaton is deterministic, the shortest of these paths is unique. The algorithm uses this property to determine edges of the path by only checking the first letter of their label. The code below, or at least its main part, implements the recurrence relation given in the proof of Lemma 2.2.5.

The algorithm FastFind is used more precisely for computing the value $\delta(r, y[j\,..\,k-1])$ (or that of $\delta(r, v)$ with the notations of the lemma). When the end of the traversed path is not the sought state, a state p is created and inserted between the last two states met.

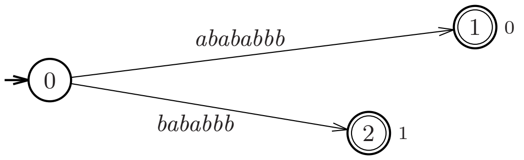
```
FastFind(r, j, k)
▷ Computation of δ(r, y[j . . k − 1])
if j ≥ k then
      return r
else q ← Target(r, y[j])
      (j′, ℓ) ← label(r, q)
      if j + ℓ ≤ k then
            return FastFind(q, j + ℓ, k)
      else adj[r] ← adj[r] \ {((j′, ℓ), q)}
            p ← NewState()
            adj[r] ← adj[r] ∪ {((j′, k − j), p)}
            adj[p] ← adj[p] ∪ {((j′ + k − j, ℓ − k + j), q)}
            return p
```

The work of algorithms SlowFindC and FastFind is illustrated by Figures 2.8 and 2.9.

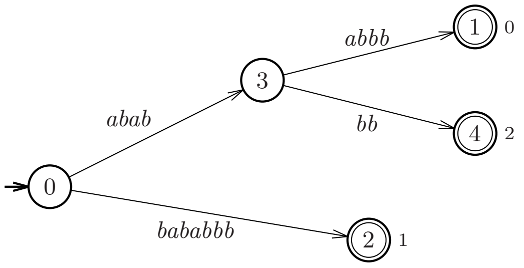
2.2.2. Complexity

The lemma which follows is used for the evaluation of the execution time of FastFind(r, j, k). It is an element of the proof of the theorem 2.2.7. It indicates that the computing time is proportional (up to a multiplicative coefficient that comes from the computing time of transitions) to the number of nodes of the traversed path, and not to the length of the label of the path, result which one would obtain immediately by applying algorithm SlowFind (Section 2.1).

For a state r of $\mathfrak{S}(y)$ and a word v for which $r \cdot v$ is a factor of y, we denote by $\mathit{end}(r, v)$ the final vertex of the shortest path having origin r and whose label has v as a prefix. Observe that $\mathit{end}(r, v) = \delta(r, v)$ only if v is the label of the path.



(a) After insertion of *abababbb* and *bababbb*.



(b) Suffix *ababbb* is added.

Figure 2.8. During the construction of $\mathfrak{S}(abababbb)$, insertion of suffixes *ababbb* and *babbb*. **(a)** Automaton obtained after the insertion of suffixes *abababbb* and *bababbb*. The current fork is the initial state 0. **(b)** Suffix *ababbb* is added using letter-by-letter comparisons (slow find) and starting from state 0. This results in creating fork 3. The suffix link of 3 is not yet defined.

Lemma 2.2.5. *Let r be a node of $\mathfrak{S}(y)$ and let v be a word such that $r \cdot v$ is a factor of y. Let $\langle r, r_1, \ldots, r_\ell \rangle$ be the path having origin r and end $r_\ell = end(r, v)$ in $\mathfrak{S}(y)$. The computation of $end(r, v)$ can be carried out in time $O(\ell \times \log \operatorname{Card} \mathcal{A})$ in the comparison model.*

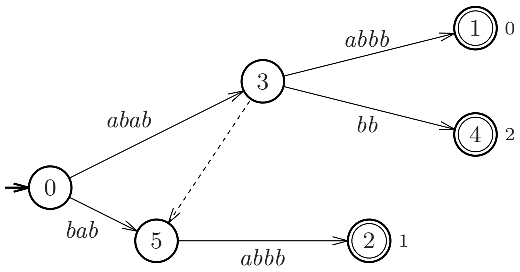
Proof. It is noticed that the path $\langle r, r_1, \ldots, r_\ell \rangle$ exists by the condition "$r \cdot v$ is a factor of y" and is unique because the tree is a deterministic automaton. If $v = \varepsilon$ one has $end(r, v) = r$. If not, let $r_1 = \textsc{Target}(r, v[0])$ and let v' be the label of edge (r, r_1). Note that

$$end(r, v) = \begin{cases} r_1 & \text{if } |v| \le |v'| \text{ (\textit{i.e.}, } v \text{ is a prefix of } v'\text{)}, \\ end(r_1, v'^{-1}v) & \text{otherwise.} \end{cases}$$

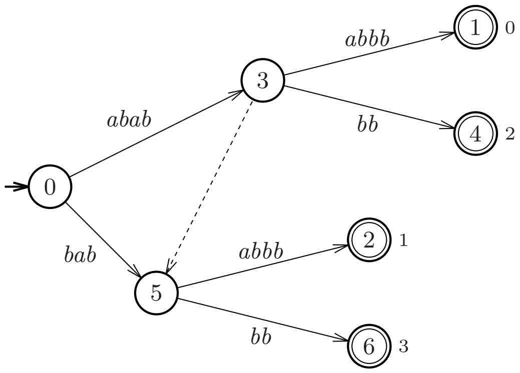
This relation shows that each stage of the computation takes time $\alpha + \beta$ where α is a constant and β is the computing time of $\textsc{Target}(r, v[0])$. This gives time $O(\log \operatorname{Card} \mathcal{A})$ in the comparison model.

The computation of r_ℓ which includes traversing the path $\langle r, r_1, \ldots, r_\ell \rangle$ thus takes time $O(\ell \times \log \operatorname{Card} \mathcal{A})$ as announced. ■

Corollary 2.2.6. *Let r be a node of $\mathfrak{S}(y)$ and j, k two positions on y, $j < k$, such that $r \cdot y[j \,..\, k-1]$ is a factor of y. Let ℓ be the number of states of the tree*



(c) Definition of the suffix link of state 3.



(d) Insertion of $babbb$.

Figure 2.9. During the construction of $\mathfrak{S}(abababbb)$ (continued). **(c)** The first step of the insertion of suffix $babbb$ starts with the definition of the suffix link of state 3, which is state 5. This is a fast find process from state 0 by word bab. **(d)** The second step of the insertion of $babbb$ leads to the creation of state 6. State 5, which is the fork of suffix $babbb$, becomes the current fork to continue the construction.

traversed during the computation of FastFind(r, j, k). *Then, the execution time of* FastFind(r, j, k) *is* $O(\ell \times \log \operatorname{Card} \mathcal{A})$ *in the comparison model.*

Proof. Let $v = y[j\,..\,k-1]$ and let $\langle r, r_1, \ldots, r_\ell \rangle$ be the path ending at $end(r, v)$. The computation of $end(r, v)$ is done by FastFind that implements the recurrence relation of the proof of Lemma 2.2.5. It thus takes time $O(\ell \times \log \operatorname{Card} \mathcal{A})$. During the last recursive call, a state p may be created and related edges modified. This operation takes time $O(\log \operatorname{Card} \mathcal{A})$. What gives the total time $O(\ell \times \log \operatorname{Card} \mathcal{A})$ of the statement. ∎

Theorem 2.2.7. *The computation of* SuffixTree$(y) = \mathfrak{S}(y)$ *takes* $O(|y| \times \log \operatorname{Card} \mathcal{A})$ *time in the comparison model.*

Proof. The fact that SuffixTree$(y) = \mathfrak{S}(y)$ is based mainly on Lemma 2.2.4 by checking that the algorithm uses again the elementary technique of Section 2.1.

The evaluation of the running time rests on the following observations (see Figure 2.7):

- Each stage of the computation done by FastFind, except perhaps the last stage, leads to traversing a state and strictly increases the value of $k - \ell$ (j on the figure), which never decreases.
- Each stage of the computation done by SlowFind, except perhaps the last stage, strictly increases the value of k, which never decreases.
- Each other instruction of the **for** loop leads to incrementing variable i, which never decreases.

The number of stages done by FastFind is thus bounded by $|y|$, which gives $O(|y| \times \log \operatorname{Card} \mathcal{A})$ time for these stages according to Corollary 2.2.6. The same reasoning applies to the number of stages carried out by SlowFind, and also for the other stages, still giving time $O(|y| \times \log \operatorname{Card} \mathcal{A})$.

Therefore, one obtains a total execution time $O(|y| \times \log \operatorname{Card} \mathcal{A})$. ■

2.3. Contexts of factors

We present in this section the formal basis for the construction of the minimal automaton which accepts the suffixes of a word, and called the *suffix automaton* of the word. Some properties contribute to the proof of the construction of the automaton (Theorems 2.3.10 and 2.4.7 below).

The suffix automaton is denoted by $\mathfrak{A}(y)$. Its states are classes of the (right) syntactic equivalence associated with $\operatorname{Suff}(y)$, *i.e.*, are the sets of factors of y having the same right context within y. These states are in one-to-one correspondence with the (right) contexts of the factors of y in y itself. Let us recall that the (right) context of a word u is $\mathcal{R}_y(u) = u^{-1}\operatorname{Suff}(y)$. We denote by $\equiv_y$ the syntactic congruence which is defined, for $u, v \in \mathcal{A}^*$, by

$$u \equiv_y v$$

if and only if

$$\mathcal{R}_y(u) = \mathcal{R}_y(v).$$

One can also identify the states of $\mathfrak{A}(y)$ to sets of indices on y which are end positions of occurrences of equivalent factors.

The right contexts satisfy some properties stated below that are used later in the chapter. The first remark concerns the link between the relation "is a suffix of" and the inclusion of contexts. For any factor u of y, one denotes by

$$\textit{end-pos}(u) = \min\{|wu| \mid wu \text{ is a prefix of } y\} - 1,$$

the right position of the first occurrence of u in y. Note that $\textit{end-pos}(\varepsilon) = -1$.

Lemma 2.3.1. *Let* $u, v \in \mathrm{Fact}(y)$ *with* $|u| \le |v|$. *Then,*

$$u \text{ is a suffix of } v \text{ implies } \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u)$$

and

$$\mathcal{R}_y(u) = \mathcal{R}_y(v) \text{ implies both } \textit{end-pos}(u) = \textit{end-pos}(v) \text{ and } u \text{ is a suffix of } v.$$

Proof. Let us suppose that u is a suffix of v. Let $z \in \mathcal{R}_y(v)$. By definition, vz is a suffix of y and, since u is a suffix of v, the word uz is also a suffix of y. Thus, $z \in \mathcal{R}_y(u)$, which proves the first implication.

Let us now suppose $\mathcal{R}_y(u) = \mathcal{R}_y(v)$. Let w, z be such that $y = w \cdot z$ with $|w| = \textit{end-pos}(u) + 1$. By definition of *end-pos*, u is suffix of w. Therefore, z is the longest word in $\mathcal{R}_y(u)$. The assumption implies that z is also the longest word in $\mathcal{R}_y(v)$, which yields $|w| = \textit{end-pos}(v) + 1$. The words u and v are thus both suffixes of w, and as u is shorter than v one obtains that u is a suffix of v. This finishes the proof of the second implication and the whole proof. ∎

Another very useful property of the congruence is that it partitions the suffixes of a factor of y into intervals according to their length.

Lemma 2.3.2. *Let* $u, v, w \in \mathrm{Fact}(y)$. *If* u *is a suffix of* v, v *is a suffix of* w *and* $u \equiv_y w$, *then* $u \equiv_y v \equiv_y w$.

Proof. By Lemma 2.3.1, the assumption implies

$$\mathcal{R}_y(w) \subseteq \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u).$$

Then, the equivalence $u \equiv_y w$ which means $\mathcal{R}_y(u) = \mathcal{R}_y(w)$ gives the conclusion. ∎

A consequence of the following property is that inclusion induces a tree structure on the right contexts. In this tree, the parent link is related to the proper inclusion of sets. This link, important for the fast construction of the automaton, corresponds to the suffix function defined then.

Corollary 2.3.3. *Let* $u, v \in \mathcal{A}^*$. *Then, the contexts of* u *and* v *are comparable for inclusion or are disjoint, i.e., at least one of the three following conditions is satisfied:*

1. $\mathcal{R}_y(u) \subseteq \mathcal{R}_y(v)$,
2. $\mathcal{R}_y(v) \subseteq \mathcal{R}_y(u)$,
3. $\mathcal{R}_y(u) \cap \mathcal{R}_y(v) = \emptyset$.

Proof. One proves the property by showing that the condition

$$\mathcal{R}_y(u) \cap \mathcal{R}_y(v) \neq \emptyset$$

implies

$$\mathcal{R}_y(u) \subseteq \mathcal{R}_y(v) \text{ or } \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u).$$

Let $z \in \mathcal{R}_y(u) \cap \mathcal{R}_y(v)$. Then, uz, vz are suffixes of y, and u, v are suffixes of yz^{-1}. Consequently, among u and v one is suffix of the other. One obtains finally the conclusion by Lemma 2.3.1. ∎

2.3.1. Suffix function

On the set $\mathrm{Fact}(y)$ we consider the function s_y called the *suffix function* of y. It is defined, for all $v \in \mathrm{Fact}(y) \setminus \{\varepsilon\}$, by

$$s_y(v) = \text{longest suffix } u \text{ of } v \text{ such that } u \not\equiv_y v.$$

After Lemma 2.3.1, one deduces the equivalent definition:

$$s_y(v) = \text{ longest suffix } u \text{ of } v \text{ such that } \mathcal{R}_y(v) \subset \mathcal{R}_y(u).$$

Note that, by definition, $s_y(v)$ is a proper suffix of v (*i.e.*, $|s_y(v)| < |v|$). The following lemma shows that the suffix function s_y induces a failure function on states of $\mathfrak{A}(y)$.

LEMMA 2.3.4. *Let $u, v \in \mathrm{Fact}(y) \setminus \{\varepsilon\}$. If $u \equiv_y v$, then $s_y(u) = s_y(v)$.*

Proof. By Lemma 2.3.1 one can suppose without loss of generality that u is a suffix of v. The word u cannot be a suffix of $s_y(v)$ because Lemma 2.3.2 would imply $s_y(v) \equiv_y v$, which contradicts the definition of $s_y(v)$. Consequently, $s_y(v)$ is a suffix of u. Since, by definition, $s_y(v)$ is the longest suffix of v which is not equivalent to itself, it is also $s_y(u)$. Thus, $s_y(u) = s_y(v)$. ∎

LEMMA 2.3.5. *Let $y \in \mathcal{A}^+$. The word $s_y(y)$ is the longest suffix of y that appears at least twice in y itself.*

Proof. The context $\mathcal{R}_y(y)$ is $\{\varepsilon\}$. As y and $s_y(y)$ are not equivalent, $\mathcal{R}_y(s_y(y))$ contains some non empty word z. Then, $s_y(y)z$ and $s_y(y)$ are suffixes of y, which shows that $s_y(y)$ appears twice at least in y.

Any suffix w of y, longer than $s_y(y)$, is equivalent to y by definition of $s_y(y)$. It thus satisfies $\mathcal{R}_y(w) = \mathcal{R}_y(y) = \{\varepsilon\}$. Which shows that w appears only once in y and finishes the proof. ∎

The following lemma shows that the image of a factor of y by the suffix function is a word of maximum length in its equivalence class.

LEMMA 2.3.6. *Let $u \in \mathrm{Fact}(y) \setminus \{\varepsilon\}$. Then, any word equivalent to $s_y(u)$ is a suffix of it.*

Proof. Let $w = s_y(u)$ and $v \equiv_y w$. We show that v is a suffix of w. The word w is a proper suffix of u. If the conclusion of the statement is false, according to Lemma 2.3.1 one obtains that w is a proper suffix of v. Let then $z \in \mathcal{R}_y(u)$. As w is a suffix of u equivalent to v, we have $z \in \mathcal{R}_y(w) = \mathcal{R}_y(v)$. Then, u and v are both suffixes of yz^{-1}, which implies that one is a suffix of the other. But this contradicts either the definition of $w = s_y(u)$ or the conclusion of Lemma 2.3.2, and proves that v is a suffix of $w = s_y(u)$. ∎

The preceding property is considered in Section 2.8 where the automaton is used as a pattern searching engine. One can check that the property of s_y is not satisfied in general on the minimal automaton which accepts the factors (and not only suffixes) of a word, or, more exactly, is not satisfied on the similar function defined from the right congruence defined from $\mathrm{Fact}(y)$ (instead of $\mathrm{Suff}(y)$).

2.3.2. Evolution of the congruence

The on-line construction of suffix automata relies on the relationship between $\equiv_{wa}$ and $\equiv_w$ which we examine here. By doing this, we consider that the generic word y is equal to wa for some letter a. The properties detailed below are also used to derive precise bounds on the size of the automaton in the following section.

The first relation states that $\equiv_{wa}$ is a refinement of $\equiv_w$.

LEMMA 2.3.7. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. The congruence $\equiv_{wa}$ is a refinement of $\equiv_w$, i.e.,, for all words $u, v \in \mathcal{A}^*$, $u \equiv_{wa} v$ implies $u \equiv_w v$.*

Proof. Let us assume that $u \equiv_{wa} v$, that is, $\mathcal{R}_{wa}(u) = \mathcal{R}_{wa}(v)$, and show that $u \equiv_w v$, that is, $\mathcal{R}_w(u) = \mathcal{R}_w(v)$. We show $\mathcal{R}_w(u) \subseteq \mathcal{R}_w(v)$ only because the opposite inclusion results by symmetry.

If $\mathcal{R}_w(u) = \emptyset$ the inclusion is clear. If not, let $z \in \mathcal{R}_w(u)$. Then uz is a suffix of w, which implies that uza is a suffix of wa. The assumption gives that vza is a suffix of wa, and thus vz is a suffix of w, or $z \in \mathcal{R}_w(v)$, which finishes the proof. ∎

The congruence $\equiv_w$ partitions $\mathcal{A}^*$ in classes. Lemma 2.3.7 amounts to saying that these classes are unions of classes according to $\equiv_{wa}$ ($a \in \mathcal{A}$). It proves that only one or two classes with respect to $\equiv_w$ are divided into two subclasses to give the partition induced by $\equiv_{wa}$. One of these two classes consists of words not appearing in w. It contains the word wa itself which produces a new class and a new state of the suffix automaton (see lemma 2.3.8). Theorem 2.3.10 and its corollaries give conditions for the division of another class and indicate how this is done.

LEMMA 2.3.8. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. If u is a suffix of wa strictly longer than z, then the equivalence $u \equiv_{wa} wa$ holds.*

Proof. It is a direct consequence of Lemma 2.3.5 because z occurs at least twice in wa. ∎

Before going to the main theorem we state an additional relation concerning right contexts.

LEMMA 2.3.9. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Then, for each word $u \in \mathcal{A}^*$,*

$$\mathcal{R}_{wa}(u) = \begin{cases} \{\varepsilon\} \cup \mathcal{R}_w(u)a & \text{if } u \text{ is a suffix of } wa, \\ \mathcal{R}_w(u)a & \text{otherwise.} \end{cases}$$

Proof. First notice that $\varepsilon \in \mathcal{R}_{wa}(u)$ is equivalent to: u is a suffix of wa. It is thus enough to show $\mathcal{R}_{wa}(u) \setminus \{\varepsilon\} = \mathcal{R}_w(u)a$.

Let z be a nonempty word of $\mathcal{R}_{wa}(u)$. We get that uz is a suffix of wa. The word uz can be written $uz'a$ with uz' a suffix of w. Consequently, $z' \in \mathcal{R}_w(u)$, and thus $z \in \mathcal{R}_w(u)a$.

Conversely, let z be a (nonempty) word in $\mathcal{R}_w(u)a$. It can be written $z'a$ for $z' \in \mathcal{R}_w(u)$. Thus, uz' is a suffix of w, which implies that $uz = uz'a$ is a suffix of wa, that is, $z \in \mathcal{R}_{wa}(u)$. This proves the converse statement and ends the proof. ∎

Theorem 2.3.10. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. Let z' be the longest factor of w for which $z' \equiv_w z$. Then, for each $u, v \in \mathrm{Fact}(w)$,*

$$u \equiv_w v \text{ and } u \not\equiv_w z \text{ imply } u \equiv_{wa} v.$$

Moreover, for each word u such as $u \equiv_w z$,

$$u \equiv_{wa} \begin{cases} z & \text{if } |u| \leq |z|, \\ z' & \text{otherwise.} \end{cases}$$

Proof. Let $u, v \in \mathrm{Fact}(w)$ be such that $u \equiv_w v$. By definition of the equivalence we get $\mathcal{R}_w(u) = \mathcal{R}_w(v)$. We suppose first that $u \not\equiv_w z$ and show that $\mathcal{R}_{wa}(u) = \mathcal{R}_{wa}(v)$, which is equivalent to $u \equiv_{wa} v$.

According to Lemma 2.3.9, we have just to show that u is a suffix of wa if and only if v is a suffix of wa. Indeed, it is enough to show that if u is a suffix of wa then v is a suffix of wa since the opposite implication results by symmetry.

So, let us suppose that u is a suffix of wa. We deduce from the fact that u is a factor of w and the definition of z that u is a suffix of z. We can thus consider the greatest integer $j \geq 0$ for which $|u| \leq |{s_w}^j(z)|$. Let us note that ${s_w}^j(z)$ is a suffix of wa (like z is), and that Lemma 2.3.2 ensures that $u \equiv_w {s_w}^j(z)$. From which we get $v \equiv_w {s_w}^j(z)$ by transitivity.

Since $u \not\equiv_w z$, we have $j > 0$. Lemma 2.3.6 implies that v is a suffix of ${s_w}^j(z)$, and thus also of wa as wished. This proves the first part of the statement.

Let us consider now a word u such as $u \equiv_w z$.

When $|u| \leq |z|$, to show $u \equiv_{wa} z$ by using the above argument, we have only to check that u is a suffix of wa because z is a suffix of wa. This, in fact, is a simple consequence of Lemma 2.3.1.

Let us suppose $|u| > |z|$. The existence of such a word u implies $z' \neq z$ and $|z'| > |z|$ (z is a proper suffix of z'). Consequently, by the definition of z, u and z' are not suffixes of wa. Using the above argument again, this proves $u \equiv_{wa} z'$ and finishes the proof. ∎

The two corollaries of the preceding theorem stated below refer to situations simple to manage during the construction of suffix automata.

Corollary 2.3.11. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. Let z' be the longest word such as $z' \equiv_w z$. Let us suppose $z' = z$. Then, for each $u, v \in \mathrm{Fact}(w)$,*

$$u \equiv_w v \text{ implies } u \equiv_{wa} v.$$

Proof. Let $u, v \in \mathrm{Fact}(w)$ be such that $u \equiv_w v$. We prove the equivalence $u \equiv_{wa} v$. The conclusion comes directly from Theorem 2.3.10 if $u \not\equiv_w z$. Else, $u \equiv_w z$; by the assumption made on z and Lemma 2.3.1, we get $|u| \leq |z|$. Finally, Theorem 2.3.10 gives the same conclusion as above. ∎

Corollary 2.3.12. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. If the letter a does not appear in w, then, for each $u, v \in \mathrm{Fact}(w)$,*

$$u \equiv_w v \text{ implies } u \equiv_{wa} v.$$

Proof. Since a does not appear in w, the word z of Corollary 2.3.11 is the empty word. It is of course the longest of its class, which makes it possible to apply Corollary 2.3.11 and gives the same conclusion. ∎

2.4. Suffix automaton

The *suffix automaton* of a word y is the minimal automaton that accepts the set of suffixes of y. It is denoted by $\mathfrak{A}(y)$. The structure is intended to be used as an index on the word (see Section 2.6) but also constitutes a device to search for factors of y within another text (see Section 2.8). The most surprising property of the automaton is that its size is linear in the length of y although the number of factors of y can be quadratic. The construction of the automaton also takes a linear time on a fixed alphabet. Figure 2.10 shows an example of such an automaton to be compared with trees in Figures 2.1 and 2.5.

As we do not force the automaton to be complete, the class of words which do not appear in y, whose right context is empty, is not a state of $\mathfrak{A}(y)$.

2.4.1. Size of suffix automata

The size of an automaton is expressed both by the number of its states and the number of its edges. We show that $\mathfrak{A}(y)$ has less than $2|y|$ states and less than $3|y|$ edges, for a total size $O(|y|)$. This result is based on Theorem 2.3.10 of the preceding section. Figure 2.11 shows an automaton that has the maximum number of states for a word length 7.

Proposition 2.4.1. *Let $y \in \mathcal{A}^*$ be a word length n and let $st(y)$ be the number of states of $\mathfrak{A}(y)$. For $n = 0$, we have $st(y) = 1$; for $n = 1$, we have $st(y) = 2$; for $n > 1$ finally, we have*

$$n + 1 \leq st(y) \leq 2n - 1,$$

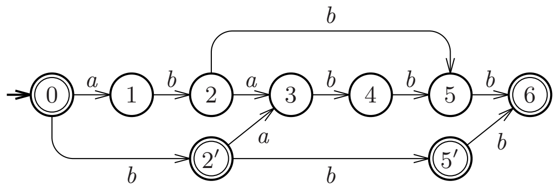


Figure 2.10. The (minimal) suffix automaton of *ababbb*.

and the upper bound is reached if and only if y is of the form ab^{n-1}, for two distinct letters a, b.

Proof. The equalities concerning short words can be checked directly including $st(y) = 3$ when $|y| = 2$. Let us suppose $n > 2$. The minimal number of states of $\mathfrak{A}(y)$ is obviously $n + 1$ (otherwise the path labeled by y would contain a cycle yielding an infinite number of words recognized by the automaton), minimum which is reached with $y = a^n$ ($a \in \mathcal{A}$).

Let us show the upper bound. By Theorem 2.3.10, each letter $y[i]$, $2 \leq i \leq n - 1$, increases by at most two the number of states of $\mathfrak{A}(y[0 \,.\,.\, i-1])$. As the number of states of $\mathfrak{A}(y[0]y[1])$ is 3, it follows that

$$\begin{aligned} st(y) &\leq 3 + 2(n-2) \\ &= 2n - 1, \end{aligned}$$

as announced.

The construction of a word of length n whose suffix automaton has $2n - 1$ states is still a simple application of the Theorem 2.3.10 by noting that each letter $y[2]$, $y[3]$, ..., $y[n-1]$ must effectively lead to the creation of two states during the construction. Notice that after the choice of the first two letters that must be different, there is no choice for the other letters. This produces the only possible form given in the statement. ∎

LEMMA 2.4.2. *Let $y \in \mathcal{A}^+$ and let $ed(y)$ be the number of edges of $\mathfrak{A}(y)$. Then,*

$$ed(y) \leq st(y) + |y| - 2.$$

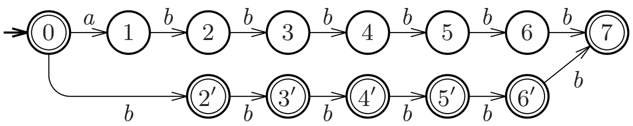


Figure 2.11. A suffix automaton with the maximum number of states.

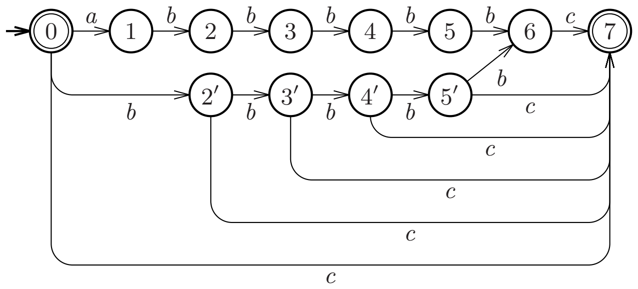


Figure 2.12. A suffix automaton with the maximum number of edges.

Proof. Let us call q_0 the initial state of $\mathfrak{A}(y)$, and consider the spanning tree of longest paths starting at q_0 in $\mathfrak{A}(y)$. The tree contains $st(y) - 1$ edges of $\mathfrak{A}(y)$ because it arrives exactly one edge on each state except on the initial state.

With each other edge (p, a, q) we associate the suffix uav of y defined as follows: u is the label of the path starting at q_0 and ending at p; v is the label of the longest path from q arriving on a terminal state. Doing so, we get an injection from the set of concerned edges to the set $\mathrm{Suff}(y)$. The suffixes y and ε are not concerned because they are labels of paths in the spanning tree. This shows that there is at most $\mathrm{Card}(\mathrm{Suff}(y) \setminus \{y, \varepsilon\}) = |y| - 1$ additional edges.

Summing up the numbers of edges of the two types, we get a maximum of $st(y) + |y| - 2$ edges in $\mathfrak{A}(y)$. ∎

Figure 2.12 shows an automaton that has the maximum number of edges for a word length 7.

Proposition 2.4.3. *Let $y \in \mathcal{A}^*$ be a word of length n and let $ed(y)$ be the number of edges of $\mathfrak{A}(y)$. For $n = 0$, we have $ed(y) = 0$; for $n = 1$, we have $ed(y) = 1$; for $n = 2$, we have $ed(y) = 2$ or $ed(y) = 3$; finally, for $n > 2$, we have*

$$n \le ed(y) \le 3n - 4,$$

and the upper bound is reached if y is of the form $ab^{n-2}c$, where a, b and c are three pairwise distinct letters.

Proof. We can directly check the results on short words. Let us consider $n > 2$. The lower bound is immediate and is reached by the word $y = a^n$ $(a \in \mathcal{A})$.

Let us examine then the upper bound. By Proposition 2.4.1 and Lemma 2.4.2 we obtain

$$\begin{aligned} ed(y) &\le (2n - 1) + n - 2 \\ &= 3n - 3. \end{aligned}$$

The $2n-1$ quantity is the maximum number of states obtained only if $y = ab^{n-1}$ $(a, b \in \mathcal{A},\ a \ne b)$. But for a word in this form the number of edges is only $2n-1$. Thus, $ed(y) \le 3n - 4$.

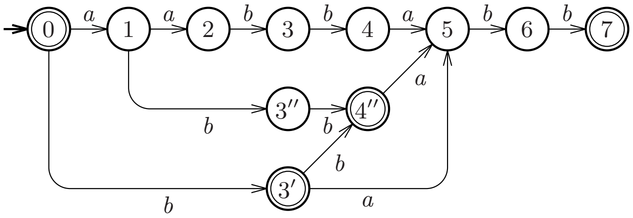


Figure 2.13. The suffix automaton $\mathfrak{A}(aabbabb)$. Suffix links on states are : $f[1]=0$, $f[2]=1$, $f[3]=3''$, $f[3'']=3'$, $f[3']=0$, $f[4]=4''$, $f[4'']=3'$, $f[5]=1$, $f[6]=3''$, $f[7]=4''$. The suffix path of 7 is $\langle 7,4'',3',0\rangle$, which includes all the terminal states of the automaton (see Corollary 2.4.6).

It can be checked that the automaton $\mathfrak{A}(ab^{n-2}c)$ (where $a,b,c \in \mathcal{A}$ with $\mathrm{Card}\{a,b,c\}=3$) has $2n-2$ states and $3n-4$ edges. ■

The following statement summarizes Propositions 2.4.1 and 2.4.3.

Theorem 2.4.4. *The total size of the suffix automaton of a word is linear in the length of the word.* ■

2.4.2. Suffix links and suffix paths

Theorem 2.3.10 and its two consecutive corollaries provide the frame of the on-line construction of the suffix automaton $\mathfrak{A}(y)$. The algorithm controls the conditions which appear in these statements by means of a function defined on the states of the automaton, the suffix link function, and of a classification of the edges in solid and non-solid edges. We define these two concepts hereafter.

Let p be a state of $\mathfrak{A}(y)$, different from the initial state. State p is a class of factors of y that are equivalent with respect to equivalence $\equiv_y$. Let u be any word in the class ($u \neq \varepsilon$ because p is not the initial state). We define the suffix link of p, denoted by $f_y(p)$, as the congruence class of $s_y(u)$. The function f_y is called the *suffix link* function of the automaton. According to Lemma 2.3.4 the value of $s_y(u)$ is independent of the word u chosen in the class of p, which makes the definition coherent. The suffix link function is also called a failure function and used with this meaning in Section 2.8. An example is given in Figure 2.13.

For a state p of $\mathfrak{A}(y)$, we denote by $lg_y(p)$ the maximum length of words u in the congruence class of p. It is also the length of the longest path starting from the initial state and ending at p. The longest paths starting at the initial state form a spanning tree for $\mathfrak{A}(y)$ (consequence of Lemma 2.3.1). Edges which belong to this tree are qualified as *solid*. In an equivalent way,

$$\text{edge } (p,a,q) \text{ is solid}$$

if and only if

$$lg_y(q) = lg_y(p) + 1.$$

This notion is used in the construction of the automaton.

Suffix links induce by iteration what we call *suffix paths* in $\mathfrak{A}(y)$ (see Figure 2.13). One can note that

$$q = f_y(p) \text{ implies } lg_y(q) < lg_y(p).$$

So, the sequence

$$\langle p, f_y(p), {f_y}^2(p), \ldots \rangle$$

is finite and ends at the initial state (which does not have a suffix link). It is called the suffix path of p in $\mathfrak{A}(y)$, and is denoted by $SP(p)$.

Let *last* be the state of $\mathfrak{A}(y)$ that is the class of word y itself. This state is characterized by the fact that it is not the origin of any edge. The suffix path of *last*,

$$\langle last, f_y(last), {f_y}^2(last), \ldots, {f_y}^{k-1}(last) = q_0 \rangle,$$

where q_0 is the initial state of the automaton, plays an important part in the on-line construction. It is used to effectively test conditions of Theorem 2.3.10 and its corollaries. We denote by δ the transition function of $\mathfrak{A}(y)$.

Proposition 2.4.5. *Let $u \in \mathrm{Fact}(y) \setminus \{\varepsilon\}$ and let $p = \delta(q_0, u)$. Then, for each integer $j \geq 0$ for which ${s_y}^j(u)$ is defined,*

$${f_y}^j(p) = \delta(q_0, {s_y}^j(u)).$$

Proof. We prove the result by recurrence on j. If $j = 0$, ${f_y}^j(p) = p$ and ${s_y}^j(u) = u$, therefore the equality is satisfied by assumption.

Let $j > 0$ such as ${s_y}^j(u)$ is defined and suppose by recurrence assumption that ${f_y}^{j-1}(p) = \delta(i, {s_y}^{j-1}(u))$. By definition of f_y, $f_y({f_y}^{j-1}(p))$ is the congruence class of the word $s_y({s_y}^{j-1}(u))$. Consequently, ${f_y}^j(p) = \delta(q_0, {s_y}^j(u))$, which completes the recurrence and the proof. ∎

Corollary 2.4.6. *The terminal states of $\mathfrak{A}(y)$ are the states of the suffix path of last, $SP(last)$.*

Proof. First, we prove that states of the path suffix are terminal. Let p be any state of $SP(last)$. One has $p = {f_y}^j(last)$ for some $j \geq 0$. Since $last = \delta(q_0, y)$, Proposition 2.4.5 implies $p = \delta(q_0, {s_y}^j(y))$. And as ${s_y}^j(y)$ is a suffix of y, p is a terminal state.

Conversely, let p be a terminal state of $\mathfrak{A}(y)$. Let then u be a suffix of y such that $p = \delta(q_0, u)$. Since u is a suffix of y, we can consider the greatest integer $j \geq 0$ for which $|u| \leq |{s_y}^j(y)|$. By Lemma 2.3.2 one obtains $u \equiv_y {s_y}^j(y)$. Thus, $p = \delta(q_0, {s_y}^j(y))$ by definition of $\mathfrak{A}(y)$. Therefore, Proposition 2.4.5 applied to y implies $p = {f_y}^j(last)$, which proves that p appears in $SP(last)$. This ends the proof. ∎

2.4.3. On-line construction

It is possible to build the suffix automaton of y by applying to the suffix trie of Section 2.1 standard algorithms that minimize automata. But the suffix trie can be of quadratic size what gives the time and space complexity of this approach. We present an on-line construction algorithm that avoids this problem and works in linear space with an execution time $O(|y| \times \log \mathrm{Card}\,\mathcal{A})$.

The algorithm treats the prefixes of y from the shorter, ε, to the longest, y itself. At each stage, just after having treated prefix w, the following information is available:

- The suffix automaton $\mathfrak{A}(w)$ with its transition function δ.
- The table f, defined on the states of $\mathfrak{A}(w)$, which implements the suffix function f_w.
- The table L, defined on the states of $\mathfrak{A}(w)$, which implements the function length, lg_w.
- The state *last*.

Terminal states of $\mathfrak{A}(w)$ are not explicitly marked, they are given implicitly by the suffix path of *last* (Corollary 2.4.6). The implementation of $\mathfrak{A}(w)$ with these additional elements is discussed just before the analysis of complexity of the computation.

Algorithm SuffixAutomaton that builds the suffix automaton of y relies on the procedure Extension given further. This procedure treats the next letter of word y. It transforms the suffix automaton $\mathfrak{A}(w)$ already built into the suffix automaton $\mathfrak{A}(wa)$ (wa is a prefix of y, $a \in \mathcal{A}$). After all extensions, terminal states are eventually marked explicitly (lines 7 to 10).

```
SuffixAutomaton(y, n)
 1  M ← NewAutomaton()
 2  L[initial(M)] ← 0
 3  last[M] ← initial(M)
 4  for each letter a of y, sequentially do
 5          ▷ Extension of M by letter a
 6          Extension(a)
 7  p ← last[M]
 8  do   terminal(p) ← true
 9          p ← f[p]
10  while p is defined
11  return M
```

Contrary to what happens for the construction of suffix trees, a state-splitting operation is necessary in some circumstances. It is realized by the algorithm Clone below.

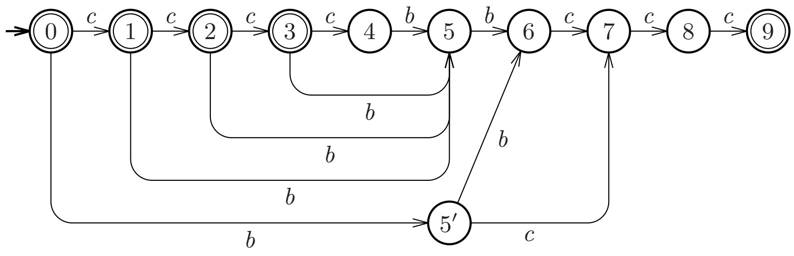


Figure 2.14. Automaton $\mathfrak{A}(ccccbbccc)$ on which is illustrated in Figures 2.15, 2.16, and 2.17 the procedure Extension(a) according to three cases.

```
Extension(a)
new ← NewState()
L[new] ← L[last[M]] + 1
p ← last[M]
do   adj[p] ← adj[p] ∪ {(a, new)}
     p ← f[p]
while p is defined and Target(p, a) is undefined
if p is undefined then
     f[new] ← initial(M)
else q ← Target(p, a)
     if (p, a, q) is a solid edge, i.e., L[p] + 1 = L[q] then
          f[new] ← q
     else clone ← Clone(p, a, q)
          f[new] ← clone
last[M] ← new
```

```
Clone(p, a, q)
clone ← NewState()
L[clone] ← L[p] + 1
for each (b, q′) ∈ adj[q] do
     adj[clone] ← adj[clone] ∪ {(b, q′)}
f[clone] ← f[q]
f[q] ← clone
do   adj[p] ← adj[p] \ {(a, q)}
     adj[p] ← adj[p] ∪ {(a, clone)}
     p ← f[p]
while p is defined and Target(p, a) = q
return clone
```

Figures 2.14, 2.15, 2.16, and 2.17 illustrate how the procedure Extension works.

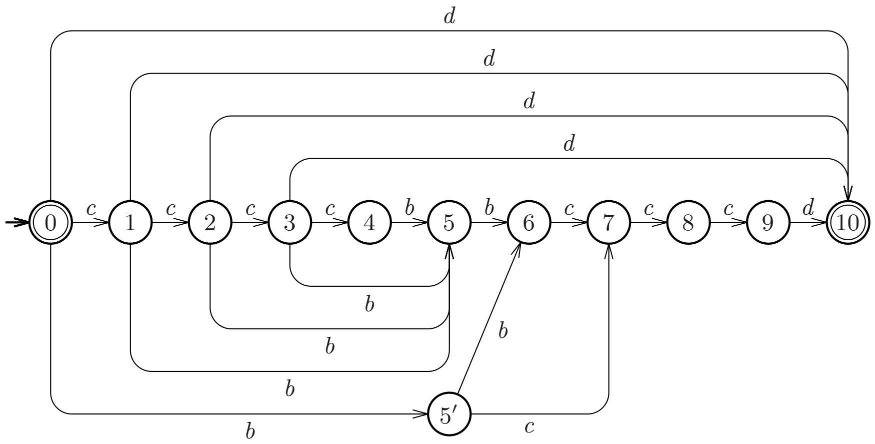


Figure 2.15. Suffix automaton $\mathfrak{A}(ccccbbcccd)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.14 by letter d. During the execution of the first loop of Extension(d), state p traverses the suffix path $\langle 9, 3, 2, 1, 0\rangle$. At the same time, edges labeled by letter d are created, starting from these states and leading to 10, the last created state. The loop stops at the initial state. This situation corresponds to Corollary 2.3.12.

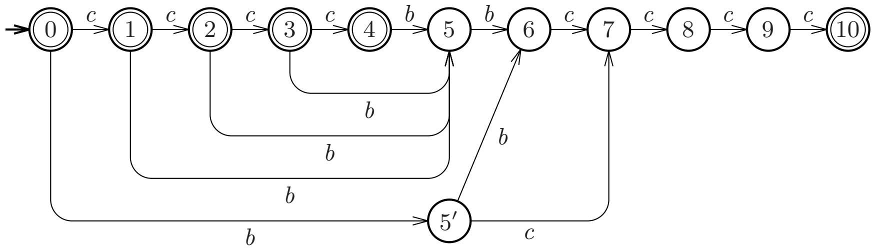


Figure 2.16. Suffix automaton $\mathfrak{A}(ccccbbcccc)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.14 by letter c. The first loop of the procedure Extension(c) stops at state $3 = f[9]$ because an edge labeled by c starts from this state. Moreover, the edge $(3, c, 4)$ is solid. We obtain directly the suffix link of the new state created: $f[10] = \delta(3, c) = 4$. There is nothing else to do according to Corollary 2.3.11.

Theorem 2.4.7. *Algorithm* SuffixAutomaton *builds a suffix automaton, that is* SuffixAutomaton(y) *is the automaton* $\mathfrak{A}(y)$, *for* $y \in \mathcal{A}^*$.

Proof. We show by recurrence on $|y|$ that the automaton is computed correctly, as well as tables L and f and state *last*. It is shown then that terminal states are computed correctly.

If $|y| = 0$, the algorithm builds an automaton consisting of only one state which is both an initial and terminal state. No transition is defined. The

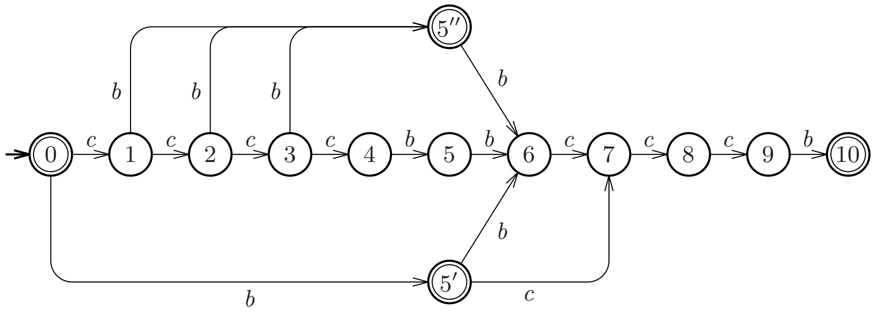


Figure 2.17. Suffix automaton $\mathfrak{A}(ccccbbcccb)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.14 by letter b. The first loop of the procedure Extension(b) stops at state $3 = f[9]$ because an edge labeled by b starts from this state. In the automaton $\mathfrak{A}(ccccbbccc)$ edge $(3, b, 5)$ is not solid. The word *cccb* is a suffix of *ccccbbcccb* but *ccccb* is not, although they both lead to state 5. This state is duplicated into the final state $5''$ that is the class of factors *cccb*, *ccb* and *cb*. Edges $(3, b, 5)$, $(2, b, 5)$ and $(1, b, 5)$ of $\mathfrak{A}(ccccbbccc)$ are redirected onto $5''$ according to Theorem 2.3.10.

automaton thus recognizes the language $\{\varepsilon\}$ which is $\text{Suff}(y)$. Elements f and *last* as well as tables L and f are also correctly calculated.

We consider now that $|y| > 0$ and that $y = wa$, for $a \in \mathcal{A}$ and $w \in \mathcal{A}^*$. We suppose, by recurrence, that the current automaton M is $\mathfrak{A}(w)$ with its transition function δ_w, that $q_0 = initial(M)$, that $last = \delta_w(q_0, w)$, that the table L satisfies $L[p] = lg_w(p)$ for any state p, and that the table f satisfies $f[p] = f_w(p)$ for any state p different from the initial state.

We first show that the procedure Extension carries out correctly the transformation of the automaton M, of the variable *last*, and of the tables L and f.

The variable p of procedure Extension runs through the states of the suffix path $SP(last)$ of $\mathfrak{A}(w)$. The first loop creates transitions labeled by a targeted at the new state *new* in agreement with Lemma 2.3.8. We have also the equality $L[new] = lg_y(new)$.

When the first loop stops, three disjoint cases arise:

1. p is not defined,
2. (p, a, q) is a solid edge,
3. (p, a, q) is a non-solid edge.

Case 1. This situation occurs when the letter a does not occur in w; one has then $f_y(new) = q_0$. Thus, after the instruction at line 8 the equality $f[new] = f_y(new)$ holds. For the other states r, one has $f_w(r) = f_y(r)$ according to Corollary 2.3.12. Which gives the equalities $f[r] = f_y(r)$ at the end of the execution of the procedure Extension.

Case 2. Let u be the longest word for which $\delta(q_0, u) = p$. By recurrence and Lemma 2.3.6, we have $|u| = lg_w(p) = L[p]$. The word ua is the longest suffix of y which is a factor of w. Thus, $f_y(new) = q$, which shows that $f[new] = f_y(new)$ after the instruction of line 11.

Since edge (p, a, q) is solid, by recurrence again, we have $|ua| = L[q] = lg_y(q)$, which shows that the words equivalent to ua according to $\equiv_w$ are not longer than ua. Corollary 2.3.11 applies with $z = ua$. And as in the case 1, $f[r] = f_y(r)$ for all the states different from *new*.

Case 3. Let u be the longest word for which $\delta(q_0, u) = p$. The word ua is the longest suffix of y which is a factor of w. So, $f_y(new) = q$, and thus $f[new] = f_y(new)$. Since edge (p, a, q) is not solid, ua is not the longest word in its congruence class according to $\equiv_w$. Theorem 2.3.10 applies with $z = ua$, and z' the longest word for which $\delta(q_0, z') = q$. The class of ua according to $\equiv_w$ is divided into two subclasses with respect to $\equiv_{wa}$. They correspond to states q and *clone*.

Words v no longer than ua and such as $v \equiv_w ua$ are of the form $v'a$ with v' a suffix of u (consequence of Lemma 2.3.1). Before the execution of the last loop, all these words v satisfy $q = \delta_w(q_0, v)$. Consequently, just after the execution of the loop, they satisfy $clone = \delta_y(q_0, v)$, as required by Theorem 2.3.10. Words v longer than ua and such as $v \equiv_w ua$ satisfy $q = \delta_y(q_0, v)$ after the execution of the loop as required by Theorem 2.3.10 again. On can check that table f is updated correctly.

For each of the three cases, one can check that the value of *last* is correctly computed at the end of the execution of the procedure Extension.

Finally, the recurrence shows that automaton M, state *last*, tables L and f are correct after the execution of procedure Extension.

It remains to be checked that terminal states are correctly marked during the execution of the last loop of algorithm SuffixAutomaton. But this is a straight consequence of Corollary 2.4.6 because variable p runs through the suffix path of *last*. ∎

2.4.4. Complexity

To analyze the complexity of the algorithm SuffixAutomaton we first describe a possible implementation of the elements necessary for the construction.

We suppose that the automaton is represented by lists of successors. By doing this, operations of addition, update, and access concerning an edge are performed in time $O(\log \operatorname{Card} \mathcal{A})$ with an efficient implementation of the lists. Function f_y is realized by table f which gives access to $f_y(p)$ in constant time.

To implement the solidity of edges table L is used. It represents the function lg_y, as the description of the procedure Extension suggests (line 10). Another way of doing it consists in using a Boolean value per edge of the automaton. This induces a slight modification of the procedure which we describe as follows: each first edge created during the execution of the loops at lines 4–6 and lines 7–10 must be marked as solid; the other created edges are marked as non solid.

This type of implementation does not require the use of table L which can then be eliminated, reducing the memory space used. Nevertheless, table L finds its utility in applications like those of Section 2.8. We retain that the two types of implementation provide a constant-time access to the quality (solid or not solid) of an edge.

Theorem 2.4.8. *Algorithm* SuffixAutomaton *can be implemented so that the construction of* $\mathfrak{A}(y)$ *takes time* $O(|y| \times \log \operatorname{Card} \mathcal{A})$ *in a memory space* $O(|y|)$.

Proof. We choose an implementation by lists of successors for the transition function. States of $\mathfrak{A}(y)$ and tables f and L require a space $O(st(y))$, lists of edges a space $O(ed(y))$. Thus, the complete implementation takes a space $O(|y|)$, as a consequence of Propositions 2.4.1 and 2.4.3.

Another consequence of these propositions is that all the operations carried out either once per state or once per edge of the final automaton take a total time $O(|y| \times \log \operatorname{Card} \mathcal{A})$. The same result applies to the operations which are performed once per letter of y. It thus remains to be shown that the time spent for the execution of the two loops at lines 4–6 and lines 7–10 of the procedure Extension is of the same order, namely $O(|y| \times \log \operatorname{Card} \mathcal{A})$.

We examine initially the case of the first loop. Let us consider the execution of the procedure Extension during the transformation of $\mathfrak{A}(w)$ into $\mathfrak{A}(wa)$ (wa is a prefix of y, $a \in \mathcal{A}$). Let u be the longest word of state p during the test at line 6. The initial value of u is $s_w(w)$, and its final value satisfies $ua = s_{wa}(wa)$ (if p is defined). Let $k = |w| - |u|$, position of the suffix occurrence of u in w. Then, each test strictly increases the value of k during a call to the procedure. Moreover, the initial value of k at the beginning of the execution of the next call is not smaller than its final value reached at the end of the execution of the current call. So, k is never decreased and thus, tests and instructions of this loop are done in $O(|y|)$.

A similar argument applies to the second loop at lines 7–10 of the procedure Extension. Let v be the longest word of p during the test of the loop. The initial value of v is ${s_w}^j(w)$, for $j \geq 2$, and its final value satisfies $va = {s_{wa}}^2(wa)$ (if p is defined). Then, the position of v as a suffix of w increases strictly at each test during successive calls to the procedure. Thus, again, tests and instructions of the loop are done in $O(|y|)$ time.

Consequently, the cumulated time of the executions of the two loops is $O(|y| \times \log \operatorname{Card} \mathcal{A})$, which finishes the proof. ∎

On a small alphabet, one can still choose an implementation of the automaton that is even more efficient than that by lists of successors, to the detriment of memory space however. It is enough to use a transition matrix within $O(|y| \times \operatorname{Card} \mathcal{A})$ memory space and managed it like a sparse table. With this particular management, any operation on edges is done in constant time, which leads to the following result.

Theorem 2.4.9. *When the alphabet is fixed, algorithm* SuffixAutomaton *can be implemented so that the construction of $\mathfrak{A}(y)$ takes time $O(|y|)$ in a memory space $O(|y| \times \mathrm{Card}\,\mathcal{A})$.*

Proof. One can use, to implement the transition matrix, the technique for representing sparse tables which gives a direct access to each one of its entries while avoiding initializing the complete matrix. ■

2.5. Compact suffix automaton

In this section, we describe briefly how to build a *compact suffix automaton* denoted by $\mathfrak{A}^{\mathrm{C}}(y)$ for $y \in \mathcal{A}^*$. This automaton can be seen as the compact version of the suffix automaton of the preceding section, *i.e.*, it is obtained by removal of the states having only one outgoing transition and that are not terminal. It is the process used on the suffix trie of Section 2.1 to produce a structure of linear size.

The compact suffix automaton is also the minimized version, in the sense of automata theory, of the (compact) suffix tree of Section 2.2. It is obtained by identifying subtrees which recognize the same words.

Figure 2.18 shows the compact suffix automaton of *ababbb* that can be compared to the compact tree of Figure 2.5 and to the automaton of Figure 2.10.

Exactly as for the tree $\mathfrak{T}(y)$, in the automaton $\mathfrak{A}(y)$ we call *fork* any state that is of (outgoing) degree at least 2, or that is both of degree 1 and terminal. Forks of suffix automata satisfy the same property as forks of suffix trees, property which allows the compaction of the automaton. The proof of the next proposition is an immediate adaptation of that of Proposition 2.2.3.

Proposition 2.5.1. *In the suffix automaton of a word, the suffix link of a fork (different from the initial state) is a fork.* ■

When one removes non fork states in $\mathfrak{A}(y)$, edges of the automaton must be labeled by (not empty) words and not only by letters. To get a structure of size linear in the length of y, labels of edges must not be stored explicitly.

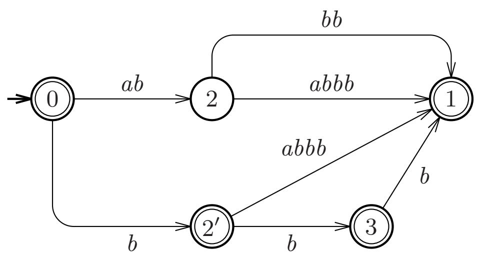
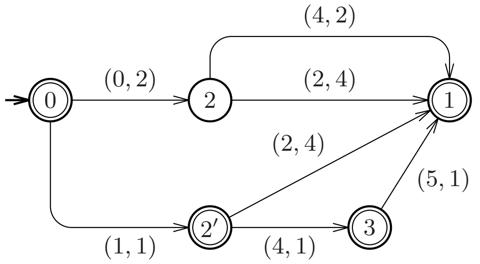


Figure 2.18. The compact suffix automaton $\mathfrak{A}^{\mathrm{C}}(ababbb)$.



i	0	1	2	3	4	5
$y[i]$	a	b	a	b	b	b

Figure 2.19. Representation of labels in the compact suffix automaton $\mathfrak{A}^{\mathrm{C}}(ababbb)$. (To be compared with the automaton in Figure 2.18.)

One represents them in constant space by means of a couple of integers. If the word u is label of an edge (p, q), it is represented by the pair $(i, |u|)$ for which i is the position of an occurrence of u in y. We denote the label by $label(p, q) = (i, |u|)$ and suppose that the implementation of the automaton provides a direct access to it. This forces to store the word y together with the data structure. Figure 2.19 indicates how labels of the compact suffix automaton of *ababbb* are represented.

The size of compact suffix automata evaluates rather directly from sizes of compact suffix trees and of suffix automata.

PROPOSITION 2.5.2. *Let $y \in \mathcal{A}^*$ be a word of length n and let $e_c(y)$ be the number of states of $\mathfrak{A}^{\mathrm{C}}(y)$. For $n = 0$, we have $e_c(y) = 1$; for $n > 0$, we have*

$$2 \leq e_c(y) \leq n + 1,$$

and the upper bound is reached for $y = a^n$, $a \in \mathcal{A}$.

Proof. The result can be checked directly for the empty word.

Let us suppose $n > 0$. Let c be a letter, $c \notin \mathcal{A}$, and let us consider the tree $\mathfrak{S}(y \cdot c)$. This tree has exactly $n + 1$ external nodes on each one of those arrives an edge whose label ends by letter c. The tree has at most n internal nodes because they have at least two outgoing edges. When minimized to get a compact automaton, all external nodes are identified in only one state, which reduces the number of state to $n + 1$ at most. Removal of letter c does not increase this value, which gives the upper bound. It is immediate to check that $\mathfrak{A}^{\mathrm{C}}(a^n)$ has $n + 1$ states exactly and that the obvious lower bound is reached when the alphabet of y has size n. ∎

PROPOSITION 2.5.3. *Let $y \in \mathcal{A}^*$ be a word of length n and let $f_c(y)$ be the number of edges of $\mathfrak{A}^{\mathrm{C}}(y)$. For $n = 0$, we have $f_c(y) = 0$; for $n = 1$, we have $f_c(y) = 1$; for $n > 1$, we have*

$$f_c(y) \leq 2(n - 1),$$

and the upper bound is reached for $y = a^{n-1}b$*, where* a, b *are two distinct letters.*

Proof. After checking the results for the short words, one notes that if x is of the form a^n, $n > 1$, one has $f_c(y) = n-1$, quantity that is smaller than $2(n-1)$.

Let us suppose now that $\mathrm{Card}\, alph(y) \geq 2$. We continue the proof of the preceding lemma by still considering the word $y \cdot c$, $c \notin \mathcal{A}$. Its suffix tree has at most $2n$ nodes. Thus it has at most $2n-1$ edges, which after compaction gives $2n-2$ edges since the edges labeled by c disappear. This gives the announced upper bound. The automaton $\mathfrak{A}^{\mathrm{C}}(a^{n-1}b)$ has n states and $2n-2$ edges, as can be directly checked. ∎

The construction of $\mathfrak{A}^{\mathrm{C}}(y)$ can be carried out starting from the tree $\mathfrak{S}(y)$ or from the automaton $\mathfrak{A}(y)$ (see exercises 2.5.1 and 2.5.2). However, to save memory space at construction time one rather takes advantage of a direct construction. It is the schema of this construction that is sketched here.

The construction borrows elements from the algorithms SuffixTree and SuffixAutomaton. Thus, the edges of the automaton are marked as solid or not solid. The created edges targeted at new leaves of the tree become edges to state *last*. We also use the concepts of slow and fast traversal from the construction of suffix trees. It is on these two procedures that the changes are essential, and that are added duplications of states and redirections of edges like for the construction of suffix automata.

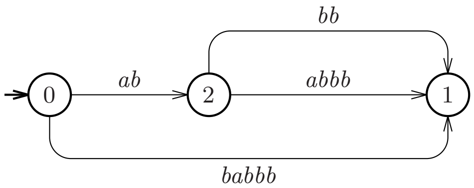
During the execution of a slow traversal, the attempt at crossing a non-solid edge leads to cloning its target, with a duplication similar to what is done during the execution of procedure Extension at line 6. One can note that certain edges can be redirected by this process.

The second important point in the adaptation of the algorithms of the preceding sections relates to the fast traversal procedure. The main algorithm calls it for the definition of a suffix link as in the algorithm SuffixTree. The difference comes when the target of a suffix link for a last-created fork (see lines 8–11 in procedure FastFind) is created. If a new state has to be created in the middle of a solid edge, the same process applies. But, if the edge is not solid, during a first step the edge is only redirected towards the concerned fork, and its label is updated accordingly. This leaves the suffix link undefined and leads to an iteration of the same process.

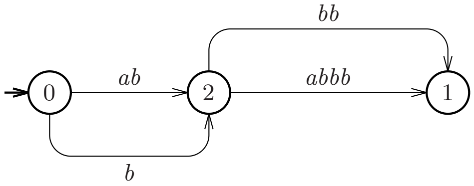
Phenomena that have been just described intervene in any sequential construction of this type of automaton. Taking them into account is necessary for a correct sequential computation of $\mathfrak{A}^{\mathrm{C}}(y)$. They are present in the construction of $\mathfrak{A}^{\mathrm{C}}(ababbb)$ (see Figure 2.18) for which three stages are detailed in Figure 2.20.

To conclude the section, we state the complexity of the direct construction of the compact suffix automaton. The formal description and the proof of the algorithm are left to the reader.

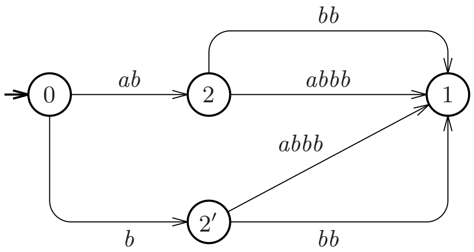
Proposition 2.5.4. *The computation of the compact suffix automaton* $\mathfrak{A}^{\mathrm{C}}(y)$ *can be done in time* $O(|y| \times \log \mathrm{Card}\, \mathcal{A})$ *in a space* $O(|y|)$*.* ∎



(a) After three insertions.



(b) Suffix link of state 2 is defined as state 0.



(c) Duplication of state 2.

Figure 2.20. Three steps of the construction of $\mathfrak{A}^{\mathrm{C}}(ababbb)$. **(a)** Automaton right after the insertion of the three longest suffixes of the word *ababbb*. The suffix link of state 2 is still undefined. **(b)** Computation by fast find of the suffix link of state 2, which results in transforming the edge $(0, babbb, 1)$ into $(0, b, 2)$. At same time, the suffix *bbb* is inserted. **(c)** Insertion of the next suffix, *bb*, is done by slow find starting from state 0. Since edge $(0, b, 2)$ is not solid, its target, state 2, is duplicated as $2'$ that has the same transitions as 2. To finish the insertion of suffix *bb* it remains to cut the edge $(2', bb, 1)$ to insert state 3. Finally, the rest of the construction amounts to determining final states, and we get the automaton of Figure 2.18.

2.6. Indexes

Techniques introduced in the preceding sections find immediate applications to the design of indexes on textual data. The utility to consider the suffixes of a text for this kind of application comes from the obvious remark that any factor of a word is a prefix of some suffix of the text. Using suffix structures thus provides a kind of direct access to all the factors of a word or a language, and it is certainly the main interest of these techniques. This property gives rise to an implementation of an index on a text or a family of texts, with efficient algorithms for the basic operations (Section 2.6.2) such as questions of membership, location and computation of lists of occurrences of patterns. Section 2.6.3 gives a solution in the form of a transducer.

2.6.1. Implementation of indexes

The aim of an index is to provide an efficient mechanism for answering certain questions concerning the contents of a fixed text. This word is denoted by y ($y \in \mathcal{A}^*$) and its length is n ($n \in \mathbb{N}$). An *index* on y can be regarded as an abstract data type whose basic set is the set of factors of y, $\mathrm{Fact}(y)$, and that includes operations for accessing information related to factors of y. The concept is similar to the index of a book which provides pointers to pages from a set of selected keywords. We rather consider what can be called a *generalized index*, in which all the factors of the text are present. We describe indexes for only one word, but extending methods to a finite number of words is in general a simple matter.

We consider four principal operations on the index of a text. They are related to a word x, the query, to be searched for in y: membership, position, number of occurrences and list of positions. This set of operations is often extended in real applications, in connection with the nature of data represented by y, to yield information retrieval systems. But the four operations we consider constitute the technical basis from which can be developed broader systems of queries.

For implementing indexes, we choose to treat the main method that leads to efficient and sometimes optimal algorithms. It is based on one of the data structures that represent suffixes of y and that are described in previous sections. The choice of the structure produces variations of the method. In this section we recall the elements of the data structures that must be available to execute the index operations. The operations themselves are treated in the next section.

The implementation of an index is built on automata of the preceding sections. Let us recall the data structures necessary to use the suffix tree, $\mathfrak{S}(y)$, of y. They are composed of:

- The word y itself stored in a table.
- An implementation of the automaton in the form of a transition matrix or list of edges per state, to represent the transition function δ, the access to the initial state, and a table of terminal states, for example.
- The table $s\ell$, defined on states, which represents the suffix link function of the tree.

Note that the word y itself must be maintained in memory for the labeling of edges refers to it (see Section 2.2). The suffix link is useful for certain applications only, it can of course be eliminated when the implemented operations do not make use of it.

One can also consider the suffix automaton of y, $\mathfrak{A}(y)$, which produces in a natural way an index on factors of the text y. The structure includes:

- an implementation of the automaton as for the tree above,
- the table f that implements the failure function defined on states,
- the table L that indicates for each state the maximum length of the words reaching this state.

For this automaton it is not necessary to store the word y in memory. It appears in the automaton as the label of the longer path starting from the initial state. Tables f and L can be omitted if non useful for the set of selected operations.

Lastly, the compact version of the suffix automaton can be used in order to reduce even more the memory capacity necessary to the store the structure. Its implementation uses in a standard way the same elements as for the suffix automaton (in non-compact version) with, in addition, the word y in order to access to labels of edges, as for the suffix tree. One gets a noticeable space reduction in using this structure rather than the two preceding ones.

In the rest of the section we examine several types of solutions for realizing basic operations on indexes.

2.6.2. Basic operations

We considers in this section four operations related to factors of text y: membership (in $\mathrm{Fact}(y)$), first position, number of occurrences, and list of the positions. The algorithms are presented after the global description of these four operations.

The first operation on an index is the membership of word x to the index, *i.e.*, the question of knowing if x is a factor of y. This question can be specified in two complementary ways according to whether one expects to find an occurrence of x in y or not. If x does not appear in y, it is often interesting in practice to find the longest prefix of x which is a factor of y. It is usually the type of response necessary to realize sequential searches in text editors.

Membership Given $x \in \mathcal{A}^*$, find the longest prefix of x that belongs to $\mathrm{Fact}(y)$.

In the contrary case ($x \in \mathrm{Fact}(y)$), methods produce without much modification the position of an occurrence of x, and even the position of the first or last occurrence of x in y.

Position Given x a factor of y, find the (left) position of its first (respectively last) occurrence in y.

Knowing that x is in the index another relevant information consists of its number of occurrences in y. This information can drive later researches differently.

Number of occurrences Given x a factor of y, find how many time x appears in y.

Lastly, under the same assumption as before, complete information on the location of x in y is provided by the list of positions of its occurrences.

List of positions Given x a factor of y, produce the list of positions of the occurrences of x in y.

We describe solutions obtained by using the above data structures. It should be noticed that the structures sometimes require to be enriched to guarantee an efficient execution of the algorithms.

PROPOSITION 2.6.1. *Given one of the automata $\mathfrak{S}(y)$, $\mathfrak{A}(y)$ or $\mathfrak{A}^{C}(y)$, computing the longest prefix u of x that is a factor of y can be carried out in time $O(|u| \times \log \operatorname{Card} \mathcal{A})$ within memory space $O(|y|)$.*

Proof. By means of $\mathfrak{A}(y)$, to determine the word u, it is enough to follow a path labeled by a prefix of x starting from the initial state of the automaton. The traversal stops when a transition misses or when x is exhausted. This produces the longest prefix of x which is also prefix of the label of a path starting at the initial state, *i.e.*, which appears in y since all the factors of y are labels of these paths. On the overall, this is done after $|u|$ successful transitions and possibly one unsuccessful transition (when u is a proper prefix of x) at the end of the test. As each transition takes a time $O(\log \operatorname{Card} \mathcal{A})$ for an implementation in space $O(|y|)$ (by lists of successors), we obtain a total time $O(|u| \times \log \operatorname{Card} \mathcal{A})$.

The same process works with $\mathfrak{S}(y)$ and $\mathfrak{A}^{C}(y)$. Taking into account the representation of these structures, certain transitions are done by simple letter comparisons, but the maximum execution time is unchanged. ∎

Position

We now examine the operations for which it is supposed that x is factor of y. The test of membership which can be carried out separately as in the preceding proposition, can also be integrated into the solutions of the other problems that interest us here. The use of transducers, which extend suffix automata for this type of question, is considered in the following section.

Finding the position $pos_y(x)$ of the first occurrence of x in y amounts to calculating its right position $end\text{-}pos_y(x)$ (see Section 2.3) because

$$pos_y(x) = end\text{-}pos_y(x) - |x| + 1.$$

Moreover, this is also equivalent to computing the maximum length of right contexts of x in y,

$$lc_y(x) = \max\{|z| \mid z \in \mathcal{R}_y(x)\},$$

because

$$pos_y(x) = |y| - lc_y(x) - |x|.$$

In a symmetrical way, in order to find the position $\textit{last-pos}_y(x)$ of the last occurrence of x in y, it remains to calculate the minimal length $sc_y(x)$ of its right contexts because

$$\textit{last-pos}_y(x) = |y| - sc_y(x) - |x|.$$

To be able to quickly answer requests related to the first or last positions of factors of y, structures of index are not sufficient alone, at least if one seeks to obtain optimal execution times. Consequently, one precomputes two tables indexed by the states of the selected automaton and that represent functions lc_y and sc_y. One thus obtains the following result.

PROPOSITION 2.6.2. *Automata $\mathfrak{S}(y)$, $\mathfrak{A}(y)$ and $\mathfrak{A}^{\mathrm{C}}(y)$ can be preprocessed in time $O(|y|)$ so that the first (or last) position on y of a factor x of y, as well as the number of occurrences of x, can be computed in time $O(|x| \times \log \operatorname{Card} \mathcal{A})$ within memory space $O(|y|)$.*

Proof. Let us call M the selected structure, δ its transition function, F its set of edges, and T its terminal states.

To begin let us consider the computation of $pos_y(x)$. The preprocessing of M relates to the computation of a table LC defined on states of M and aimed at representing the function lc_y. For a state p and a word $u \in \mathcal{A}^*$ with $p = \delta(\textit{initial}(M), u)$, we define

$$LC[p] = lc_y(u),$$

quantity that is independent of the word u that labels a path from the initial state to p, according to Lemma 2.3.1. This value is also the maximum length of paths starting at p and ending at a terminal state in the automaton $\mathfrak{A}(y)$. For $\mathfrak{S}(y)$ and $\mathfrak{A}^{\mathrm{C}}(y)$ this consideration still applies by defining the length of an edge as that of its label.

The table LC satisfies the recurrence relation:

$$LC[p] = \begin{cases} 0 & \text{if } \deg(p) = 0, \\ \max\{\ell + LC[q] \mid (p, v, q) \in F \text{ and } |v| = \ell\} & \text{otherwise.} \end{cases}$$

The relation shows that the computation of values $LC[p]$, for all the states of M, is done by a simple depth-first traversal of the graph of the structure. As its number of states and its number of edges are linear (see sections 2.2, 2.4 and 2.5) and since the access to the label length of an edge is done in constant time according to the representation described in Section 2.2, the computation of the table takes a time $O(|y|)$ (independent of the alphabet).

Once the precomputation of the table LC is performed, the computation of $pos_y(x)$ is done by searching for $p = \delta(\textit{initial}(M), x)$ and then by computing $|y| - LC[p] - |x|$. We then obtain the same asymptotic execution time as for the membership problem, namely $O(|x| \times \log \operatorname{Card} \mathcal{A})$. Let us note that if

$$\textit{end}(\textit{initial}(M), x) = \delta(\textit{initial}(M), xw)$$

with w non empty, the value of $pos_y(x)$ is then $|y| - LC[p] - |xw|$, which does not modify the asymptotic evaluation of the execution time.

The computation of the position of the last occurrence of x in y is solved in a similar way by considering the table SC defined by

$$SC[p] = sc_y(u),$$

with the notations above. The relation

$$SC[p] = \begin{cases} 0 & \text{if } p \in T, \\ \min\{\ell + SC[q] \mid (p,v,q) \in F \text{ and } |v| = \ell\} & \text{otherwise,} \end{cases}$$

shows that the precomputation of SC takes a time $O(|y|)$, and that the computation of $\mathit{last\text{-}pos}_y(x)$ takes then $O(|x| \times \log \operatorname{Card} \mathcal{A})$ time.

Lastly, for accessing the number of occurrences of x one precomputes a table NB defined by

$$NB[p] = \operatorname{Card}\{z \in \mathcal{A}^* \mid \delta(p,z) \in T\},$$

which is precisely the sought quantity when $p = \mathit{end}(\mathit{initial}(M), x)$. The linear precomputation results from the relation

$$NB[p] = \begin{cases} 1 + \sum_{(p,v,q) \in F} NB[q] & \text{if } p \in T, \\ \sum_{(p,v,q) \in F} NB[q] & \text{otherwise.} \end{cases}$$

Then, the number of occurrences of x is obtained by computing the state $p = \mathit{end}(\mathit{initial}(M), x)$ and by accessing to $NB[p]$, which is done in the same time as for the above operations.

This ends the proof. ■

An argument similar to the last element of the preceding proof allows an effective computation of the number of factors of y, *i.e.*, of the size of $\operatorname{Fact}(y)$. For that, one evaluates the quantity $CS[p]$, for all states p of the automaton, by using the relation

$$CS[p] = \begin{cases} 1 & \text{if } \deg(p) = 0, \\ 1 + \sum_{(p,v,q) \in F} (|v| - 1 + CS[q]) & \text{otherwise.} \end{cases}$$

If $p = \delta(\mathit{initial}(M), u)$ for some factor u of y, $CS[p]$ is the number of factors of y starting with u. This gives a linear-time computation of $\operatorname{Card}\operatorname{Fact}(y) = CS[q_0]$ (q_0 initial state of the automaton), *i.e.*, in time $O(|y|)$ independent of the alphabet, given the automaton.

List of positions

PROPOSITION 2.6.3. *Given the tree $\mathfrak{S}(y)$ or the automaton $\mathfrak{A}^{\mathrm{C}}(y)$, the list L of positions of the occurrences of a factor x of y can be computed in time $O(|x| \times \log \operatorname{Card} \mathcal{A} + k)$ within memory space $O(|y|)$, where k is the number of elements in L.*

Proof. The tree $\mathfrak{S}(y)$ is first considered. Let us point out from Section 2.1 that a state q of the tree is a factor of y, and that, if it is terminal, its output is the position of the suffix occurrence of q in y (in this case q is a suffix of y and $output[q] = |y| - |q|$). The positions of occurrences of x in y are the positions of suffixes prefixed by x. One thus obtains these positions by seeking terminal states of the subtree rooted at $p = end(initial(M), x)$ (see section 2.2). Exploration of this subtree takes a time proportional to its size and indeed to its number of terminal nodes since each node that is not terminal has at least two children by definition of the tree. Finally, the number of terminal nodes is precisely the number k of elements of the list L.

In short, the computation of the list require that of p and then the traversal of the subtree. The first phase is carried out in time $O(|x| \times \log \operatorname{Card} \mathcal{A})$, the second in time $O(k)$, which gives the announced result when $\mathfrak{S}(y)$ is used.

A similar reasoning applies to $\mathfrak{A}^{\mathrm{C}}(y)$. Let $p = end(initial(M), x)$ and let w be such that $\delta(initial(M), xw) = p$. Starting from p, we explore the automaton by memorizing the length of the current path (the length of an edge is that of its label). A terminal state q that is reached by a path of length ℓ corresponds to a suffix of length ℓ which therefore occurs at position $|y| - \ell$. Then, $|y| - \ell - |xw|$ is the position of an occurrence of x in y. The complete traversal takes a time $O(k)$ as its equivalent traversal of the subtree of $\mathfrak{S}(y)$ describes above. We thus obtain the same running time as with the compact suffix tree. ∎

Notice that the computation of the lists of positions is obtained without preprocessing the automata. By the way, using the (non-compact) suffix automaton of y requires a preprocessing which consists in creating shortcuts to superimpose the structure of $\mathfrak{A}^{\mathrm{C}}(y)$ to it, if one wishes to obtain the same running time.

2.6.3. Transducer of positions

Some of the questions of locating factors within the word y can be described in terms of transducers, *i.e.*, automata in which edges have an output in addition to outputs on states. As an example, the function pos_y is realized by the transducer of positions of y, denoted by $\mathcal{T}(y)$. Figure 2.21 gives an illustration of it.

The transducer $\mathcal{T}(y)$ is built upon $\mathfrak{A}(y)$ by adding outputs to edges and by modifying the outputs associated with the terminal states. Edges of $\mathcal{T}(y)$ are of the form $(p, (a, s), q)$ where p, q are states and (a, s) the label of the edge. Letter $a \in \mathcal{A}$ is its input and integer $s \in \mathbb{N}$ is its output. The path

$$(p_0, (a_0, s_0), p_1), (p_1, (a_1, s_1), p_2), \ldots, (p_{k-1}, (a_{k-1}, s_{k-1}), p_k)$$

of the transducer has as input label the word $a_0 a_1 \cdots a_{k-1}$, concatenation of input labels of edges of the path, and for output the sum $s_0 + s_1 + \cdots + s_{k-1}$.

The transformation of $\mathfrak{A}(y)$ into $\mathcal{T}(y)$ is done as follows. When (p, a, q) is an edge of $\mathfrak{A}(y)$ it becomes the edge $(p, (a, s), q)$ of $\mathcal{T}(y)$ with output

$$s = end\text{-}pos_y(q) - end\text{-}pos_y(p) - 1,$$

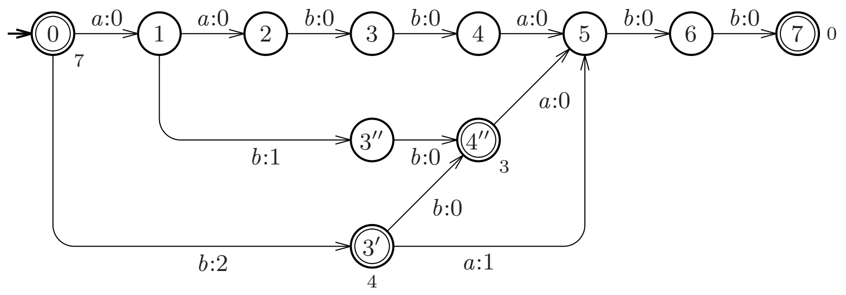


Figure 2.21. Transducer that realizes in a sequential way the function pos_y relative to $y = aabbabb$. Each edge is labeled by a pair (a, s) denoted by $a{:}s$, where a is the input of the edge and s its output. When scanning abb, the transducer produces the value 1 $(= 0+1+0)$, which is the position of the first occurrence of abb in y. The last state having output 3, one deduces that abb is a suffix at position 4 $(= 1 + 3)$ of y.

which is also

$$LC[p] - LC[q] - 1$$

with the notation LC used in the proof of Proposition 2.6.2. The output associated with a terminal state p is defined as $LC[p]$. It is shown how to compute the table LC in the proof of Proposition 2.6.2, from which one deduces a computation of outputs associated with edges and terminal states. The transformation is thus carried out in linear time.

Proposition 2.6.4. *Let u be the input label of a path starting at the initial state of the transducer $\mathcal{T}(y)$. Then, the output of the path is $pos_y(u)$.*

Moreover, if the end of the path is a terminal state having output t, u is a suffix of y and the position of this occurrence of u in y is $pos_y(u)+t$ $(= |y|-|u|)$.

Proof. We prove it by recurrence on the length of u. The first step of the recurrence, for $u = \varepsilon$, is immediate. Let us suppose that $u = va$ with $v \in \mathcal{A}^*$ and $a \in \mathcal{A}$. The output of the path having input label va is $r + s$, where r and s are respectively the outputs corresponding to inputs v and a. By the recurrence hypothesis, we have $r = pos_y(v)$. By definition of labels in $\mathcal{T}(y)$, we have

$$s = \textit{end-pos}_y(u) - \textit{end-pos}_y(v) - 1.$$

Therefore the output associated with u is

$$pos_y(v) + \textit{end-pos}_y(u) - \textit{end-pos}_y(v) - 1,$$

or also, since $\textit{end-pos}_y(w) = pos_y(w) + |w| - 1$,

$$pos_y(u) + |u| - |v| - 1,$$

which is $pos_y(u)$ as expected. This finishes the proof of the first part of the statement.

If the end of the considered path is a terminal state, its output t is, by definition, $LC[u]$, which is $|y| - \textit{end-pos}_y(u) - 1$ or $|y| - pos_y(u) - |u|$. Therefore $pos_y(u) + t = |y| - |u|$, which is the position in y of the suffix u as announced. ■

The existence of the transducer of positions described above shows that the position of a factor in y can be computed sequentially, while reading the factor. The computation is even done in real time when transitions are performed in constant time.

2.7. Finding regularities

2.7.1. Repetitions

In this section we examine two questions concerning repetitions of factors within the text y. There are two dual problems that are solved efficiently by using a suffix tree or suffix automaton:

- Compute longest repeated factors of y.
- Find shortest factors having few occurrences in y.

These questions are parameterized by an integer k which bounds the number of occurrences.

Longest repetition Given an integer k, $k > 1$, find a longest word occurring at least k times in y.

Let $\mathfrak{A}(y)$ be the suffix automaton of y. If the table NB defined in the proof of Proposition 2.6.2 is available, the problem of the longest repetition remains to find the states p of $\mathfrak{A}(y)$ which are the deepest in the automaton and for which $NB[p] \geq k$. The labels of longest paths from the initial state to p's are then solutions of the problem.

Indeed the solution comes without the use of table NB because values in the table do not need to be stored. We show how this is done for the instance of the problem with $k = 2$. One simply seeks a state (or all states), as deep as possible, that satisfies one of the two conditions:

- at least two edges leave p,
- an edge leaves p and p is a terminal state.

State p is then a fork and it is found by a mere traversal of the automaton. Proceeding in this way, no preliminary treatment of $\mathfrak{A}(y)$ is necessary and nevertheless the linear computing time is preserved. One can note that the execution time does not depend on the branching time in the automaton because no transition is executed, the search only traverses existing edges.

The two descriptions above are summarized in the following proposition.

Proposition 2.7.1. *Given one of the automata $\mathfrak{S}(y)$, $\mathfrak{A}(y)$ or $\mathfrak{A}^{\mathrm{C}}(y)$, computing a longest repeated factor of y can be done in time and space $O(|y|)$.* ∎

The second problem deals with searching for a marker. A factor of y is so called when it marks a small number of positions on y.

Marker Given an integer k, $k > 1$, find a shortest word having less than k occurrences in y.

The use of a suffix automaton provides a solution to the problem of the same vein as that of the solution to the longest repetition problem. It amounts to find, in the automaton, a state that is as closest as possible to the initial state and that is the origin of less than k paths to a terminal state. Contrary to the above situation however, a state associated with a marker is not necessarily a fork, but this has no effect on the solution. Again, a simple traversal of the automaton solves the question, which gives the following result.

Proposition 2.7.2. *Given one of the automata $\mathfrak{S}(y)$, $\mathfrak{A}(y)$ or $\mathfrak{A}^{\mathrm{C}}(y)$, the computation of a marker in y can be carried out in time and space $O(|y|)$.* ∎

2.7.2. Forbidden words

Searching for forbidden words is a reverse question to finding repetitions. It intervenes in the description of a certain type of text compression algorithms.

A word $u \in \mathcal{A}^*$ is called a *forbidden word* in the word $y \in \mathcal{A}^*$ if it is not factor of y. And u is called a *minimal forbidden word* if in supplement all its own proper factors are factors of y. In other words, the minimality relates to the ordering “is a factor of”. This concept is in fact more relevant than the preceding one. We denote by $I(y)$ the set of minimal forbidden words in y.

One can notice that

$$u = u[0\,..\,k-1] \in I(y)$$

if and only if

$$u \text{ is not a factor of } y \text{ but } u[0\,..\,k-2] \text{ and } u[1\,..\,k-1] \text{ are factors of } y,$$

which results in the equality

$$I(y) = (\mathcal{A} \cdot \mathrm{Fact}(y)) \cap (\mathrm{Fact}(y) \cdot \mathcal{A}) \cap (\mathcal{A}^* \setminus \mathrm{Fact}(y)).$$

The equality shows in particular that the language $I(y)$ is finite. It is thus possible to represent $I(y)$ by a trie in which the external nodes only are terminal because of the minimality of words.

The algorithm ForbiddenWords, which code is given below, built the trie accepting $I(y)$ from the automaton $\mathfrak{A}(y)$. Figure 2.22 shows the example of the trie of forbidden words of *aabbabb*, obtained from the automaton of Figure 2.13. In the algorithm, the queue is used to traverse the automaton $\mathfrak{A}(y)$ in a width-first manner.

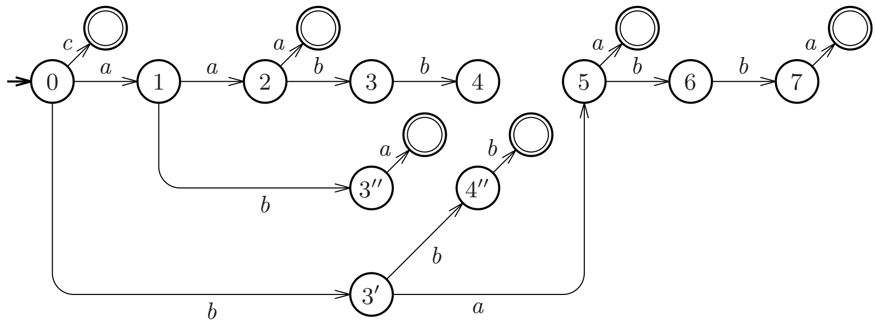


Figure 2.22. Trie of minimal forbidden words of the word *aabbabb* on the alphabet $\{a, b, c\}$, such as it is built by algorithm Forbidden. Non-terminal states are those of automaton $\mathfrak{A}(aabbabb)$ of Figure 2.13. Note that states 3 and 4 as well as the edges reaching them can be removed. The forbidden word *babba*, recognized by the tree, is minimal because *babb* and *abba* are factors of *aabbabb*.

```
ForbiddenWords(𝔄(y))
 1  M ← NewAutomaton()
 2  L ← EmptyQueue()
 3  Enqueue(L, (initial(𝔄(y)), initial(M)))
 4  while not FileIsEmpty(L) do
 5      (p, p′) ← Dequeue(L)
 6      for each a ∈ 𝒜 do
 7          if Target(p, a) is undefined and
              (p = initial(𝔄(y)) or Target(f[p], a) is defined) then
 8              q′ ← NewState()
 9              terminal(q′) ← true
10              adj[p′] ← adj[p′] ∪ {(a, q′)}
11          elseif Target(p, a) is defined and
              Target(p, a)notreachedyet then
12              q′ ← NewState()
13              adj[p′] ← adj[p′] ∪ {(a, q′)}
14              Enqueue(L, (Target(p, a), q′))
15  return M
```

Proposition 2.7.3. *For $y \in \mathcal{A}^*$, the algorithm* ForbiddenWords *produces, from the automaton $\mathfrak{A}(y)$, a tree that accepts the language $I(y)$. The execution can be done in time $O(|y| \times \log \operatorname{Card} \mathcal{A})$.*

Proof. It is noticed that edges created at line 13 duplicate the edges of the spanning tree of shortest paths of the graph of $\mathfrak{A}(y)$, because the automaton is traversed in width-first order (the queue L is aimed at that). Other edges are created at line 10 and are of the form (p', a, q') with $p', q' \in T'$, denoting by T' the set of terminal states of M. Let us denote by δ' the transition function associated with the edges of M created by the algorithm. By construction, the word u for which $\delta'(\mathit{initial}(M), u) = p'$ is the shortest word that reaches the state $p = \delta(\mathit{initial}(\mathfrak{A}(y)), u)$ in $\mathfrak{A}(y)$.

We start by showing that any word recognized by the tree that the algorithm produces is a minimal forbidden word. Let ua be such a word, necessarily nonempty ($u \in \mathcal{A}^*$, $a \in \mathcal{A}$). By assumption, the edge (p', a, q') was created at line 10 and $q' \in T'$. If $u = \varepsilon$, we have $p' = \mathit{initial}(M)$ and we notice that, by construction, $a \notin \mathit{alph}(y)$; therefore ua is effectively a minimal forbidden word. If $u \neq \varepsilon$, let us write it bv with $b \in \mathcal{A}$ and $v \in \mathcal{A}^*$. The state

$$s = \delta(\mathit{initial}(\mathfrak{A}(y)), v)$$

satisfies $s \neq p$ because both $|v| < |u|$ and, by construction, u is the shortest word that satisfies $p = \delta(\mathit{initial}(M), u)$. Therefore $f[p] = s$, by definition of suffix links. Then, again by construction, $\delta(s, a)$ is defined, which implies that va is a factor of y. The word $ua = bva$ is thus minimal forbidden since bv, va are factors of y but ua is not a factor of y.

It is then shown conversely that any forbidden word is recognized by the tree built the algorithm. Let ua such a word, necessarily nonempty ($u \in \mathrm{Fact}(y)$, $a \in \mathcal{A}$). If $u = \varepsilon$, the letter a does not appear in y, and thus $\delta(\mathit{initial}(\mathfrak{A}(y)), a)$ is not defined. The condition at line 7 is met and causes to create an edge which leads to the recognition of the word ua by the automaton M. If $u \neq \varepsilon$, let us write it bv with $b \in \mathcal{A}$ and $v \in \mathcal{A}^*$. Let

$$p = \delta(\mathit{initial}(\mathfrak{A}(y)), u).$$

As v is a proper suffix of u and va is a factor of y while ua is not a factor of y, if we consider the state

$$s = \delta(\mathit{initial}(\mathfrak{A}(y)), v),$$

we have necessarily $p \neq s$ and thus $s = f[p]$ by definition of suffix links. The condition at line 7 is thus still satisfied in this case, and this has the same effect as above. In conclusion, ua is recognized by the tree created by the algorithm, which finishes the proof. ■

An unexpected consequence of the preceding construction is an upper bound on the number of minimal forbidden words in a word.

Proposition 2.7.4. *A word $y \in \mathcal{A}^*$ of length $|y| \geq 2$ has no more than $\mathrm{Card}\,\mathcal{A} + (2|y| - 3) \times (\mathrm{Card}\,\mathit{alph}(y) - 1)$ minimal forbidden words. It has $\mathrm{Card}\,\mathcal{A}$ of them if $|y| < 2$.*

Proof. According to the preceding proposition the number of minimal forbidden words in y is equal to the number of terminal states of the trie $I(y)$, which is also the number of edges entering these states.

There is exactly $\mathrm{Card}\,\mathcal{A} - \alpha$ such edges coming out from the initial state, by noting $\alpha = \mathrm{Card}\,\mathit{alph}(y)$. There is at most α outgoing edges from the unique state of $\mathfrak{A}(y)$ having no outgoing transition. From other states there is at most

$\alpha - 1$ outgoing edges. Since, for $|y| \geq 2$, $\mathfrak{A}(y)$ has at most $2|y| - 1$ states (Proposition 2.4.1), we obtain

$$\operatorname{Card} I(y) \leq (\operatorname{Card} \mathcal{A} - \alpha) + \alpha + (2|y| - 3) \times (\alpha - 1),$$

which gives

$$\operatorname{Card} I(y) \leq \operatorname{Card} \mathcal{A} + (2|y| - 3) \times (\alpha - 1),$$

as announced.

Finally, we have $I(\varepsilon) = \mathcal{A}$ and, for $a \in \mathcal{A}$, $I(a) = (\mathcal{A} \setminus \{a\}) \cup \{aa\}$. Therefore $\operatorname{Card} I(y) = \operatorname{Card} \mathcal{A}$ when $|y| < 2$. ∎

2.8. Pattern matching machine

Suffix automata can be used like machines to locate occurrences of patterns. We consider in this section the suffix automaton $\mathfrak{A}(x)$ to implement the search for x (length m) in a word y (length n). The other structures, compact tree $\mathfrak{S}(x)$ and compact automaton $\mathfrak{A}^{\mathrm{C}}(x)$, can be used as well.

The searching algorithm rests on considering a transducer with a failure function. The transducer computes sequentially the lengths ℓ_i defined below. It is built upon the automaton $\mathfrak{A}(x)$, and the failure function, used to cope with non-explicitly defined transitions of the searching automaton, is nothing else than the suffix link f defined on states of the automaton. The principle of the searching method is standard. The search is carried out sequentially along the word y. Adaptation and analysis of the algorithm with the tree $\mathfrak{S}(x)$ are immediate although the suffix link function of this structure is not a failure function according to the precise meaning of this concept (see Exercise 2.2.4).

The advantage brought by the algorithm on other methods based on failure functions lies in a bounded amount of time to treat a letter of y, together with a more direct analysis of its time complexity. The price for this improvement is a more important need of memory capacity intended to store the automaton instead of a simple table, although the space remains linear.

2.8.1. Lengths of common factors

The search for x is based on computing lengths of factors of x appearing at any position on y. More precisely, the algorithm computes, at any position i on y, $0 \leq i < n$, the length

$$\ell_i = \max\{|u| \mid u \in \operatorname{Fact}(x) \cap \operatorname{Suff}(y[0 \,..\, i])\}$$

of the longest factor of x ending at this position. The detection of occurrences of x follows the obvious remark:

$$x \text{ occurs at position } i - |x| + 1 \text{ on } y$$

if and only if

$$\ell_i = |x|.$$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$y[i]$	a	a	a	b	b	b	a	b	b	a	a	b	b	a	b	b	b
ℓ_i	1	2	2	3	4	2	3	4	5	4	2	3	4	5	6	7	2
p_i	1	2	2	3	4	$4''$	5	6	7	5	2	3	4	5	6	7	$4''$

Figure 2.23. Using the automaton $\mathfrak{A}(aabbabb)$ (see Figure 2.13), algorithm Length-of-factors determines the factors common to *aabbabb* and y. Values ℓ_i and p_i are the respective values of variables ℓ and p of the algorithm related to position i. At position 8 for example, $\ell_8 = 5$ indicates that the longest factor of *aabbabb* ending there has length 5; it is *bbabb*; the current state is 7. An occurrence of the pattern is detected when $\ell_i = 7 = |aabbabb|$, as it is at position 15.

The algorithm which computes the lengths $\ell_0, \ell_1, \ldots, \ell_{n-1}$ is given below. It uses the table L, defined on states of the automaton (Section 2.4), to reset the length of the current factor, after a traversal through a suffix link (line 8). The correction of this instruction is a consequence of Lemma 2.3.6. A simulation of the computation is given in Figure 2.23.

LengthsOfFactors($\mathfrak{A}(x), y$)
1 $(\ell, p) \leftarrow (0, \mathit{initial}(\mathfrak{A}(x)))$
2 **for** $i \leftarrow 0$ **to** $n-1$ **do**
3   **if** Target($p, y[i]$) is defined **then**
4         $(\ell, p) \leftarrow (\ell + 1,$ Target$(p, y[i]))$
5   **else do** $p \leftarrow f[p]$
6   **while** p is defined and Target($p, y[i]$) is undefined
7   **if** p is defined **then**
8             $(\ell, p) \leftarrow (L[p] + 1,$ Target$(p, y[i]))$
9   **else** $(\ell, p) \leftarrow (0, \mathit{initial}(\mathfrak{A}(x)))$
10   output ℓ

Theorem 2.8.1. *The algorithm* LengthsOfFactors *applied to the automaton $\mathfrak{A}(x)$ and the word y ($x, y \in \mathcal{A}^*$) produces the lengths $\ell_0, \ell_1, \ldots, \ell_{|y|-1}$. It makes less than $2|y|$ transitions in $\mathfrak{A}(x)$ and runs in time $O(|y| \times \log \operatorname{Card} \mathcal{A})$ and space $O(|x|)$.*

Proof. The proof of correctness of the algorithm is done by recurrence on the length of prefixes of y. We show more exactly than the equalities

$$\ell = \ell_i$$

and

$$p = \delta(\mathit{initial}(\mathfrak{A}(x)), y[i - \ell + 1 \,..\, i])$$

are invariants of the **for** loop, by noting δ the transition function of $\mathfrak{A}(x)$.

Let $i \geq 0$. The already-treated prefix has length i and the current letter is $y[i]$. It is supposed that the condition is satisfied for $i-1$. Thus, $u = y[i-\ell \mathinner{.\,.} i-1]$ is the longest factor of x ending at position $i-1$ and $p = \delta(\mathit{initial}(\mathfrak{A}(x)), u)$.

Let w be the suffix of length ℓ_i of $y[0 \mathinner{.\,.} i]$. Let us first suppose $w \neq \varepsilon$; therefore w rewrites $v \cdot y[i]$ with $v \in \mathcal{A}^*$. Note that v cannot be longer than u because this would contradict the definition of u, and thus v is a suffix of u.

If $v = u$, $\delta(p, y[i])$ is defined and provides the next value of p. Moreover, $\ell_i = \ell + 1$. These two points correspond to the update of (ℓ, p) carried out at line 4, which shows that the condition is satisfied for i in this situation.

When v is a proper suffix of u, we consider the greatest integer k, $k > 0$, for which v is a suffix of $s_x^k(u)$ where s_x is the suffix function relative to x (Section 2.3). Lemma 2.3.6 implies that $v = s_x^k(u)$ and that the length of this word is $L_x(q)$ where $q = \delta(\mathit{initial}(\mathfrak{A}(x)), v)$. The new value of p is thus $\delta(q, y[i])$, and that of ℓ is $L_x(q) + 1$. It is done so by the instruction at line 8, since f and L respectively implement the suffix function and the length function of the automaton, and according to Proposition 2.4.5 which establishes the relation with function s_x.

When $w = \varepsilon$, this means that letter $y[i] \notin \mathit{alph}(x)$. It is thus necessary to re-initialize the pair (ℓ, p), which is done at line 9.

Finally, it is noted that the proof is also valid for the treatment of the first letter of y, which finishes the proof of the invariant condition and proves the correctness of the algorithm.

For the complexity, one notices that each transition done, successfully or not, leads to incrementing i or to strictly increasing the value of $i - \ell$. As each one of these two expressions varies from 0 to $|y|$, we deduces that the number of transitions done by the algorithm is no more than $2|y|$. Moreover, as the execution time of all the transitions is representative of the total execution time, it is $O(|y| \times \log \operatorname{Card} \mathcal{A})$.

The memory space necessary to run the algorithm is used mainly to store the automaton $\mathfrak{A}(x)$ which has size $O(|x|)$ according to the theorem 2.4.4. This gives the last stated result, and finishes the proof. ∎

2.8.2. Optimization of suffix links

Since the algorithm LengthsOfFactors works in a sequential way, it is natural to consider its delay, that is, the maximum time spent on a letter of y. One realizes immediately that it is possible to modify the suffix function in order to reduce this time.

Optimization is based on sets of letters that label edges going out a state. We define, for p state of $\mathfrak{A}(x)$,

$$\mathit{Next}(p) = \{a \in \mathcal{A} \mid \delta(p, a) \text{ is defined}\}.$$

Then, the new suffix link $\hat{f}$ is defined, for p state of $\mathfrak{A}(x)$, by the relation:

$$\hat{f}[p] = \begin{cases} f[p] & \text{if } \mathit{Next}(p) \subset \mathit{Next}(f[p]), \\ \hat{f}[f[p]] & \text{else, if this value is defined.} \end{cases}$$

Note that the relation can leave the value of $\hat{f}[p]$ undefined. The idea of this definition comes from the fact that the link is used as a failure function: there is no need to go to $f[p]$ if $Next(f[p]) \subseteq Next(p)$.

Note that in the automaton $\mathfrak{A}(x)$ one always has

$$Next(p) \subseteq Next(f[p]).$$

So, we can reformulate the definition of $\hat{f}$ as:

$$\hat{f}[p] = \begin{cases} f[p] & \text{if } \deg(p) \neq \deg(f[p]), \\ \hat{f}[f[p]] & \text{else, if this value is defined.} \end{cases}$$

The computation of $\hat{f}$ can thus be performed in linear time by considering outgoing degrees (deg) of states in the automaton.

The optimization of the suffix link leads to a reduction of the delay of algorithm LengthsOfFactors. The time can be evaluated as the number of executions of the instruction at line 5. We get the following result, which shows that the algorithm treat the letters of y in real time when the alphabet is fixed.

Proposition 2.8.2. *When the algorithm* LengthsOfFactors *makes use of the suffix link $\hat{f}$ in place of f, the treatment of each letter of y takes a time* $O(\text{Card}\, alph(x))$.

Proof. The result is an immediate consequence of inclusions

$$Next(p) \subset Next(\hat{f}[p]) \subseteq \mathcal{A}$$

for each state p for which $\hat{f}[p]$ is defined. ∎

2.8.3. Search for conjugates

The sequence of lengths $\ell_0, \ell_1, \ldots, \ell_{n-1}$ of the preceding section is a very rich information on resemblances between the words x and y. It can be exploited in various ways by algorithms comparing words. It authorizes for example an efficient computation of $LCF(x, y)$, the maximum length of factors common to x and y. This is done in linear time on a bounded alphabet. This quantity intervenes for example in the definition of the distance between words:

$$d(x, y) = |x| + |y| - 2LCF(x, y).$$

We are interested in searching for conjugates (or rotation) of a word within a text. The solution put forward in this section is another consequence of the length computation described in the previous section. Let us recall that a conjugate of word x is a word of the form $v \cdot u$ for which $x = u \cdot v$.

Searching for conjugates Let $x \in \mathcal{A}^*$. Locate all the occurrences of conjugates of x (of length m) occurring in a word y (of length n).

A first solution consists in applying a classical algorithm for searching a finite set of words after having built the trie of conjugates of x. The search time is then proportional to n (on a fixed alphabet), but the trie can have a quadratic size $O(n^2)$, as can be the size of the (non compact) suffix trie of x.

The solution based on the use of a suffix automaton does not have this disadvantage while preserving an equivalent execution time. The technique is derived from the computation of lengths done in the preceding section. For this purpose, we consider the suffix automaton of the word $x \cdot x$, by noting that every conjugate of x is a factor of $x \cdot x$. One could even consider the word $x \cdot w\mathcal{A}^{-1}$ where w is the primitive root of x, but that does not change the following result.

Proposition 2.8.3. *Let $x, y \in \mathcal{A}^*$. Locating the conjugates of x in y can be done in time $O(|y| \times \log \mathrm{Card}\,\mathcal{A})$ within a memory space $O(|x|)$.*

Proof. We consider a variant of algorithm LengthsOfFactors that produces the positions of the occurrences of factors having a length not smaller than a given integer k. The transformation is immediate since at each stage of the algorithm the length of the current factor is stored in variable ℓ.

The modified algorithm is applied to the automaton $\mathfrak{A}(x^2)$ and the word y with parameter $k = |x|$. The algorithm thus determines factors of length $|x|$ of x^2 which appear in y. The conclusion follows, noting that factors of length $|x|$ of x^2 are conjugate of x, and that all conjugates x appear in x^2. ∎

The concept of index is strongly used in questions related to data retrieval techniques. One can refer to the book of Baeza-Yates and Ribero-Neto 1999 to go deeper into the subject, or to that of Salton 1989. Apostolico 1985 describes several algorithmic applications of suffix trees that applies often to other suffix structures.

Personal searching systems, or indexes used by search engines, often use simpler techniques like the constitution of lexicons of rare words or k-grams (*i.e.*, factors of length k) with k relatively small.

The majority of topics covered in this chapter is classical in string algorithmics. The book of Gusfield 1997 contains a good number of problems, and especially those grounded on questions in computational molecular biology, whose algorithmic solutions rest on the use of data structures for indexes, including questions related to repetitions.

Forbidden words of Section 2.7.2 are used in the DCA compression method of Crochemore, Mignosi, Restivo, and Salemi 2000.

The use of suffix automata as searching machines is due to Crochemore 1987. Using suffix trees for this purpose produces an immediate but less efficient solution (see exercise 2.2.4).

Problems

Section 2.2

2.2.1 Check that the execution of SuffixTree(a^n) ($a \in \mathcal{A}$) takes a time $O(n)$. Check that the execution time of SuffixTree(y) is $\Omega(n \log n)$ when $\mathrm{Card}\, alph(y) = |y| = n$.

2.2.2 How many nodes are there in the compact suffix tree of a de Bruijn word? How many for a Fibonacci word? Same question for their compact and non compact suffix automata.

2.2.3 Let $\mathcal{T}_{k,l}(y)$ be the compact trie that accepts the factors of word y that have a length ranging between the two natural integers k and ℓ ($0 \le k \le \ell \le |y|$). Design an algorithm to build $\mathcal{T}_{k,l}(y)$ and that uses a memory space proportional to the size of the tree (and not $O(|y|)$) and that runs in the same asymptotic time as the construction of the suffix tree of y.

2.2.4 Design an algorithm for the computation of $LCF(x, y)$ ($x, y \in \mathcal{A}^*$), maximum length of factors common to x and y, based on the tree $\mathfrak{S}(x \cdot c \cdot y)$, where $c \in \mathcal{A}$ and $c \notin alph(x \cdot y)$. What is the time and space complexity of the computation? Compare with the solution in Section 2.8.

2.2.5 Give a bound on the number of cubes of primitive words occurring in a word of length n. Same question for squares. (Hint: use the suffix tree of the word.)

2.2.6 Design an algorithm for the fusion of two suffix trees.

2.2.7 Describe a linear time and space algorithm (on a fixed alphabet) for the construction of the suffix tree of a finite set of words.

Section 2.3

2.3.1 Let y be a word in which the last letter does not appear elsewhere. Show that $\mathfrak{F}(y)$, the minimal deterministic automaton accepting the factors of y, has the same states and same edges as $\mathfrak{A}(y)$ (only the terminal states differ).

2.3.2 Give the precise number of states and edges in the factor automaton $\mathfrak{F}(y)$.

Section 2.4

2.4.1 Design an on-line algorithm for the construction of the factor automaton $\mathfrak{F}(y)$. The algorithm should run in linear time and space on a finite and fixed alphabet.

2.4.2 Design a linear-time algorithm (on a fixed alphabet) for the construction of the suffix automaton of a finite set of words.

Section 2.5

2.5.1 Describe an algorithm for constructing $\mathfrak{A}^{\mathrm{C}}(y)$ from $\mathfrak{S}(y)$.

2.5.2 Describe an algorithm for constructing $\mathfrak{A}^{\mathrm{C}}(y)$ from $\mathfrak{A}(y)$.

2.5.3 Write in details the code of the algorithm for the direct construction of $\mathfrak{A}^{\mathrm{C}}(y)$.

2.5.4 Design an on-line algorithm for constructing $\mathfrak{A}^{\mathrm{C}}(y)$.

Section 2.7

2.7.1 Let $k > 0$ be an integer. Implement an algorithm, based on one of the automata of suffixes of $y \in \mathcal{A}^*$, which determines factors that appear at least k times in y.

2.7.2 For $y \in \mathcal{A}^*$, design an algorithm for computing the maximum length of factors of y which have two non-overlapping occurrences (*i.e.*, if u is such a factor, it appears in y at two positions i and j such as $i+|u| \le j$).

2.7.3 It is said that a language $M \subseteq \mathcal{A}^*$ avoids a word $u \in \mathcal{A}^*$ if u is not factor of any word of M. Let M be the language of words that avoid all the words of a finite set $I \subseteq \mathcal{A}^*$. Show that M is accepted by a finite automaton. Give an algorithm that builds an automaton accepting M given the trie of I.

2.7.4 Design a construction of the automaton $\mathfrak{F}(y)$ given the trie of forbidden words $I(y)$.

Section 2.8

2.8.1 Provide an infinite family of words for which each word has a trie of its conjugates that is of quadratic size.

2.8.2 Design an algorithm for locating conjugates of x in y ($x, y \in \mathcal{A}^*$), given the tree $\mathfrak{S}(x \cdot x \cdot c \cdot y)$, where $c \in \mathcal{A}$ and $c \notin alph(x \cdot y)$. What is the complexity of the computation?

Notes

The concept of position tree is due to Weiner 1973 who presented an algorithm to compute its compact version. The algorithm of Section 2.2 is from McCreight 1976. A strictly sequential version of the suffix tree construction was described by Ukkonen 1995.

For questions referring to formal languages, like concepts of syntactic congruences and minimal automata, one can refer to the books of Berstel 1979 and Pin 1986.

The suffix automaton of a text with unmarked terminal states is also known as the suffix DAWG, *Directed Acyclic Word Graph*. Its linearity was discovered by Blumer, Blumer, Ehrenfeucht, Haussler, and McConnel 1983 who gave a linear construction of it on a fixed alphabet (see also Blumer et al. 1985). The minimality of the structure as an automaton is due to Crochemore 1986, who showed how to build within the same complexity the factor automaton of a text (see exercises 2.3.1, 2.3.2 and 2.4.1).

The notion of compact suffix automaton appears in Blumer, Ehrenfeucht, and Haussler 1989. An algorithm for compacting suffix automata, as well as a direct construction of compact suffix automata, is presented in Crochemore and Vérin 1997. An on-line construction of compact suffix automata has been designed by Inenaga, Hoshino, Shinohara, Takeda, Arikawa, Mauri, and Pavesi 2001.

For the average analysis of sizes of the various structures presented in the chapter one can refer to Szpankowski 1993b and to Jacquet and Szpankowski 1994, who corrected a previous analysis by Blumer et al. 1989, extended by Raffinot 1997. These analyses rely on methods described in the book of Sedgewick and Flajolet 1995.

On special integer alphabets, Farach 1997 has designed a linear time construction of suffix trees.

Indexes can also be realized efficiently with the use of suffix arrays. This data structure may be viewed as an implementation of a suffix tree. The notion has been introduced by Manber and Myers 1993 who designed the first efficient algorithms for its construction and use. On special integer alphabets, a suffix array can be built in linear time by three independent algorithms provided by Kärkkäinen and Sanders 2003, Kim, Sim, Park, and Park 2003, and Ko and Aluru 2003.

CHAPTER 3

Symbolic Natural Language Processing

3.0. Introduction

Fundamental notions of combinatorics on words underlie natural language processing. This is not surprising, since combinatorics on words can be seen as the formal study of sets of strings, and sets of strings are fundamental objects in language processing.

Indeed, language processing is obviously a matter of strings. A text or a discourse is a sequence[1] of sentences; a sentence is a sequence of words; a word is a sequence of letters. The most universal levels are those of sentence, word and letter (or phoneme), but intermediate levels exist, and can be crucial in

[1] In this chapter, we will not use the term "word" to denote a sequence of symbols, in order to avoid ambiguity with the linguistic meaning.

some languages, between word and letter: a level of morphological elements (e.g. suffixes), and the level of syllables. The discovery of this piling up of levels, and in particular of word level and phoneme level, delighted structuralist linguists in the 20th century. They termed this inherent, universal feature of human language as "double articulation".

It is a little more intricate to see how *sets* of strings are involved. There are two main reasons. First, at a point in a linguistic flow of data being processed, you must be able to predict the set of possible continuations after what is already known, or at least to expect any continuation among some set of strings that depends on the language. Second, natural languages are ambiguous, i.e. a written or spoken portion of text can often be understood or analyzed in several ways, and the analyses are handled as a set of strings as long as they cannot be reduced to a single analysis. The notion of set of strings covers the two dimensions that linguists call the syntagmatic axis, i.e. that of the chronological sequence of elements in a given utterance, and the paradigmatic axis, i.e. the "or" relation between linguistic forms that can substitute for one another.

The connection between language processing and combinatorics on words is natural. Historically, linguists actually played a part in the beginning of the construction of theoretical combinatorics on words. Some of the terms in current use originate from linguistics: word, prefix, suffix, grammar, syntactic monoid... However, interpenetration between the formal world of computer theory and the intuitive world of linguistics is still a love story with ups and downs. We will encounter in this chapter, for example, terms that specialists of language processing use without bothering about what they mean in mathematics or in linguistics.

This chapter is organized around the main levels of any language modeling: first, how words are made from letters; second, how sentences are made from words. We will survey the basic operations of interest for language processing, and for each type of operation we will examine the formal notions and tools involved.

3.1. From letters to words

All the operations in the world between letters and words can be collectively denoted by the term "lexical analysis". Such operations mainly involve finite automata and transducers. Specialists in language processing usually refer to these formal tools with the term "finite-state" tools, because they have a finite number of states.

3.1.1. Normalization of encoding

The computer encoding of the 26 letters of the Latin alphabet is fairly standardized. However, almost all languages need additional characters for their writing. European languages use letters with diacritics: accents (*é, è*), cedilla (*ç*), tilde (*ñ*), umlaut (*ü*)... There are a few ligatures, the use of some of them being standard in some conditions: *æ*, *œ*, *ß*, others are optional variants: *ff*, *fl*. The encoding of these extensions of 7-bit ASCII is by no means normalized: constructors of computers and software editors have always tended to propose divergent encodings in order to hold users captive and so faithful. Thus, *é* is encoded as 82 and 8E in two common extended ASCII codes, as 00E9 in UCS-2 Unicode, as C3A9 in UTF-8 Unicode, and named "é" by ISO 8879:1986 standard. The situation of other alphabets (Greek, Cyrillic, Korean, Japanese...) is similar. The encoding systems for the Korean national writing system are based on different levels: in KSC 5601-1992, each symbol represents a syllable; in "n-byte" encodings, each symbol represents a segment of a syllable, often a phoneme.

Thus, generally speaking, when an encoding is transliterated into another, a symbol may be mapped to a sequence of several symbols, or the reverse. Transliteration implies (i) cutting up input text into a concatenation of segments, and (ii) translating each segment. Both aspects depend on input and output encodings.

Transliteration is simple whenever it is unambiguous, i.e. when source encoding and target encoding convey exactly the same information in two different forms. The underlying formal objects are very simple. The set of possible segments in input text is a finite code (the input code). It is often even a prefix code, i.e. no segment is a prefix of another. Here is an example of an input code that is not prefix: consider transliterating a phoneme-based Korean encoding into a syllable-based encoding. A 5-symbol input sequence *kilto* must be segmented as *kil/to* in order to be translated into a 2-symbol output sequence, but *kilo* must be segmented as *ki/lo*.

In any case, encodings are designed so that transliteration can be performed by a sequential transducer.

For the reader's convenience, we will recall a few of the definitions of section 1.5. A finite transducer over the alphabets A, B is a finite automaton in which all edges have an input label $u \in A^*$ and an output label $v \in B^*$. The input alphabet A can be different from the output alphabet B, but they frequently have a nonempty common subset. The notation we will use is convenient when a transducer is considered as an automaton over a finite alphabet of the form $X \subset A^\star \times B^\star$, as in section 3.1.5, and when we define a formal notion of alignment, as in section 3.1.7. Elements of X will be denoted $(u\!:\!v)$ or $\begin{pmatrix} u \\ v \end{pmatrix}$ as in Fig. 3.1; edges will be denoted $(p, u\!:\!v, q)$. The label of a successful path of a transducer consists of a pair of sequences $(w\!:\!x) \in A^\star \times B^\star$. Corresponding input and output sequences may be of different lengths in number of symbols, and some of the edges may have input and output labels of different lengths. A transducer over A and B is input-wise literal if and only if all input labels are in $A|\varepsilon$, and input-wise synchronous if and only if they are in A. The set of labels of successful paths of a transducer is the transduction realized by the transducer. A transduction over A and B is a relation between $A^\star$ and $B^\star$. A transduction over A and B can be specified by a regular expression in the monoid $A^\star \times B^\star$ if and only if it is realized by a finite transducer.

A sequential transducer is a finite transducer with additional output labels attached to the initial and terminal states, and with the following properties:

- it has at most one initial state,
- it is input-wise synchronous,
- for each state p and input label $a \subset A$, there is at most one edge $(p, a : u, q) \in E$.

The output string for a given input string is obtained by concatenating the initial output label, the output label of the path defined by the input string, and the terminal output label attached to the terminal state that ends the path. With a sequential transducer, input sequences can be mapped into output sequences through input-wise deterministic traversal. All transductions realized by sequential transducers are word functions. Sequential transducers can be minimized (cf. section 1.5.2).

In practice, the output labels attached to terminal states are necessary for transliteration when input code is not prefix. The second and third properties above are obtained by adapting the alignment between input labels and output labels, i.e. by making them shorter or longer and by shifting parts of labels between adjacent edges. Fig. 3.1 shows a sequential transducer that transliterates *é* and *è* from their ISO 8879 names, "é" and "è", to their codes in an extended ASCII encoding, 82 and 8A.

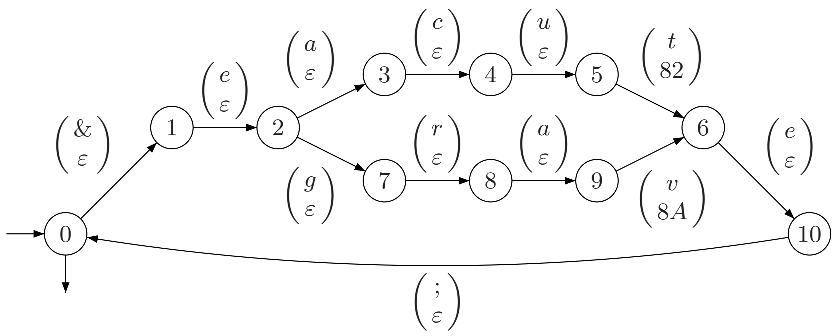


Figure 3.1. A sequential transducer that substitutes "82" for "é" and "8A" for "è".

The number of edges of transducers for normalization of character encoding is of the same order of magnitude as the sum of the lengths of the elements of the input code, say 30 if only letters are involved and 3000 if syllables are involved.

Transliteration from one encoding to another is ambiguous when the target system is more informative than the source system. For example, 7-bit ASCII encoding, frequently used in informal communication, does not make any difference between *e* and *é*, or between *oe* and the ligature *œ*. In a more elaborate encoding, these forms are not equivalent: *œ* is not a free variant for *oe*; it can be used in *cœur* but not in *coexiste*. Transliteration from 7-bit ASCII to an extended ASCII encoding involves recognizing more complex linguistic elements, like words. It cannot be performed by small sequential transducers.

The situation is even more complex in Korean and Japanese. In these languages, text can be entirely written in national writing systems, but Chinese characters are traditionally substituted for part of it, according to specific rules. In Japan, the use of Chinese characters in written text is standard in formal communication; in Korea, this traditional substitution is not encouraged by the authorities and is on the waning. Let us consider text with and without Chinese characters as two encodings. The version with Chinese characters is usually more informative than the one without: when a word element is ambiguous, it may have several transcriptions in Chinese characters, according to its respective meanings. However, the reverse also happens. For instance, an ambiguous Chinese character that evokes "music", "pleasure" or "love" in Korean words is pronounced differently, and transcribed *ak*, *lak*, *nak* or *yo* in the national writing system, depending on the words in which it occurs.

3.1.2. Tokenization

The first step in the processing of written text is helped by the fact that words are delimited by spaces. During Antiquity, this feature was exclusive to unvowelled script of Semitic languages; it developed in Europe progressively during the early Middle Ages (Saenger, 1997) and is now shared by numerous languages in the world.

Due to word delimitation, a simple computer program can segment written text into a sequence of words without recognizing them, e.g. without a dictionary. This process is called tokenization. Once it has been performed, words become directly available for further operations: statistics, full text indexation, dictionary lookup...

The formal basis of delimiter-based tokenization is the unambiguous use of certain characters as delimiters.

The alphabet of letters, A, and the alphabet of delimiters, D, are disjoint. A text is a sequence of letters and delimiters. After tokenization, it is a sequence of tokens. Word tokens are maximal occurrences of elements of $A^\star$ in the text. Delimiter tokens can be defined either as single delimiters:

Why/?/ /1/./ /Because/ /of/ /temperature/.

or as sequences of delimiters:

Why/? 1. /Because/ /of/ /temperature/.

Some symbols, like dash (-) and apostrophe (') in English, can be considered either as letters or as delimiters. In the first case, *trade-off* and *seven-dollar* are tokens; otherwise they are sequences of tokens. In any case, tokenization can be performed by simulating the two-state automaton of Fig. 3.2, and by registering a new word token whenever control shifts from state 1 to state 0.

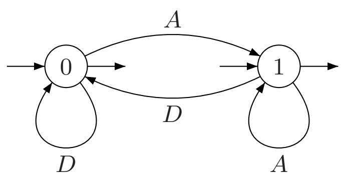


Figure 3.2. An automaton for written text tokenization.

In this section, we used the term "word" in its everyday sense; I would even say in its visual sense: a word in written text is something visibly separated by spaces. However, this naive notion of word does not always give the best results if we base further processing on it, because visual words do not always behave as units conveying a meaning. For example *white* does in *white trousers*, but not in *white wine*. We will return to this matter in section 3.1.4.

Delimiter-based tokenization is not applicable to languages written without delimitation between words, like Arabic, Chinese or Japanese. In these languages, written text cannot be segmented into words without recognizing the words. The problem is exactly the same with spoken text: words are not audibly delimited.

However, in some cases, another type of tokenization consists in identifying all the positions in the text where words are liable to begin. These positions cut up text into tokens. After that, words can be recognized as certain sequences of tokens. For instance, in Thai language, words can only begin and end at syllable boundaries, and syllable boundaries cannot be preceded or followed by any patterns of phonemes. These patterns can be recognized by a transducer.

3.1.3. Zipf's law

During the tokenization of a text or of a collection of texts, it is easy to build the list of all the different tokens in the text, to count the occurrences of each different token, and to rank them by decreasing number of occurrences. What is the relation between rank r and number of occurrences n_r? Zipf observed that the following law is approximately true:

$$n_r = n_1/r^a \tag{3.1.1}$$

with $a \approx 1$. As a matter of fact, there are few frequent tokens, and many infrequent tokens. In experiments on French text, 1 token out of 2 was found to belong to the most frequent 139 tokens. In fact, for $20 \leq r \leq 2000$, n_r is a little higher than predicted by (3.1.1).

Several equations can be derived from Zipf's law. The number r_n of different tokens that occur at least n times is such that $n = n_1/r_n^a$, so:

$$r_n = \left(\frac{n_1}{n}\right)^{1/a}$$

The number of different tokens that occur between n and $n+1$ times is:

$$\left(\frac{n_1}{n}\right)^{1/a} - \left(\frac{n_1}{n+1}\right)^{1/a} \tag{3.1.2}$$

For large values of n and $a = 1$, this is approximately n_1/n^2, which is confirmed experimentally.

According to (3.1.2), the number of tokens that occur once (hapaxes) is proportional to $n_1^{1/a}$. It is easy to observe that the number of occurrences of a very frequent token is approximately proportional to the size of the text, i.e. n_1/N depends on the language but not on the text. This means that all texts comprise roughly the same proportion of hapaxes.

Can Zipf's law be used to predict the relation between the size of a text and the size of its vocabulary? The size of the text is the total number of occurrences of tokens,

$$N = n_1 + n_2 + \ldots + n_R$$

where R is the size of the vocabulary, i.e. the number of different tokens. With $a = 1$, we have:

$$N = n_1 \sum_{r=1}^{R} 1/r \approx n_1 \ln R$$

However, the relation between N and n_1 in this equation is not confirmed experimentally. Firstly, n_1 is proportional to N. Secondly, the growth of R with respect to N tends to slow down, because of the tokens that occur again, whereas this equation implies that it would speed up. Thirdly, if this law were accurate, R would grow unbounded with N, which means that the vocabulary of a language would be infinite. What is surprising and counter-intuitive is that a steady growth of R with respect to N is maintained for texts up to several million different tokens.

In other words, Zipf's law correctly predicts that a collection of texts needs to be very large and diverse to encompass the complete vocabulary of a language, because new texts will contain new words for a very long time. Experience shows, for example, that the proportion of vocabulary which is shared by one year's production of a newspaper and another year's production is smaller than simple intuition would suggest.

3.1.4. Dictionary compression and lookup

Most operations on text require information about words: their translation into another language, for example. Since such information cannot in general be computed from the form of words, it is stored in large databases, in association with the words. Information about words must be formal, precise, systematic and explicit, so that it can be exploited for language processing. Such information is encoded into word tags or lexical tags. Examples of word tags are given in Fig. 3.3. The tags in this figure record only essential information:

fit	*fit*	*A*
fit	*fit*	*N:s*
fit	*fit*	*V:W:P1s:P2s:P1p:P2p:P3p*
fitter	*fit*	*A:C*
fitting	*fit*	*V:G*
hop	*hop*	*N:s*
hop	*hop*	*V:W:P1s:P2s:P1p:P2p:P3p*
hope	*hope*	*N:s*
hope	*hope*	*V:W:P1s:P2s:P1p:P2p:P3p*
hoping	*hope*	*V:G*
hopping	*hop*	*V:G*
hot	*hot*	*A*
hot air	*hot air*	*N:s*
hotter	*hot*	*A:C*
open	*open*	*A*
open	*open*	*N:S*
open	*open*	*V:W:P1s:P2s:P1p:P2p:P3p*
open air	*open air*	*N:S*

Figure 3.3. The word tags for a few English words.

- the lemma, which is the corresponding form with default inflectional features, e.g. the infinitive, in the case of verbs,
- the part of speech: *A*, *N*, *V*...,
- the inflectional features.

Lemmas are necessary for nearly all applications, because they are indexes to properties of words. If all the vocabulary is taken into account, the tag set used in Fig. 3.3 has many thousands of elements, due to lemmas. Size of tag sets is a measure of the informative content of tags.

The operation of assigning tags to words in a text is called lexical tagging. It is one of the main objectives of lexical analysis. The reverse operation is useful in text generation: words are first generated in the form of lexical tags, then you have to spell them. In many languages, it is feasible to construct a list of roughly all words that can occur in texts. Such a list, with unambiguous word tags, is called an electronic dictionary[2], or a dictionary. The strange term "full-form dictionary" is also in use. An electronic dictionary is in the order of a million words. Such a list is always an approximation, due to the fact that new words continuously come into use: proper nouns, foreign borrowings, new derivatives of existing words...

[2]The term "electronic dictionary" emphasizes the fact that entries are designed for programs, whereas the content of "conventional dictionaries" is meant for human readers, no matter whether they are stored on paper or on electronic support.

In inflectional languages like English, the construction of an electronic dictionary involves generating inflected forms, like conjugated verbs or plurals. This operation is usually carried out with tables of suffixes, prefixes or infixes, or with equivalent devices.

What is considered as a word is not always clear, because words sometimes appear as combinations of words, e.g. *hot air* "meaningless talk", *open air* "outdoors space", *white wine*, which are called compound words. The situation is less clear with numerals, e.g. *sixty-nine*: linguistically, each of them is equivalent to a determiner, which is a word; technically, if we include them in the dictionary, they are another million words; syntactically, they are made of elements combined according to rules, but these rules are entirely specific to numerals and are not found anywhere in the syntax of the language. The status of such forms and of other examples like dates is not easy to assign. If they are considered as words, then the simplest form of description for them is a finite automaton. We will refer to such automata in section 3.2.2 by the term "local grammars".

The most repetitive operation on an electronic dictionary is lookup. The input of this operation is word forms, and the output, word tags. Natural and efficient data structures for them are tries, with output associated to leaves, and transducers. In both cases, lookup is done in linear time with respect to the length of the word, and does not depend on the size of the dictionary.

Consider representing the dictionary in the form of a transducer. The dictionary is viewed as a finite set of word form/word tag pairs, i.e. a transduction. Alignment between input and output is based on the similarity between word forms and the lemmas included in word tags. This transduction is not a word function, since many word forms in a dictionary are associated with several word tags, like *fit* in Fig. 3.3:

The shoes are fit for travel
Max had a fit of fever
These shoes fit me

Due to this universal phenomenon, known as lexical ambiguity or homography, the transduction cannot be represented by a sequential transducer. A p-sequential transducer is a generalization of sequential transducers with at most p terminal output strings at each terminal state. A p-sequential transducer for the words in Fig. 3.3 is shown in Fig. 3.4. In this transducer, the symbol # stands for a space character. The notion of p-sequential transducer allows for representing a transduction that is not a word function without resorting to an ambiguous transducer. A transducer is ambiguous if and only if it has distinct paths with the same input label. In a p-sequential transducer, there are no distinct paths with the same input label; any difference between output labels of the same path must occur in terminal output strings.

In order to make the transducer p-sequential, lexically ambiguous word forms must be processed in a specific way: any difference between the several word tags for such a word form must be postponed to terminal output strings, by shifting parts of labels to adjacent edges. This operation may change the natural alignment between input and output, and increase the number of states and

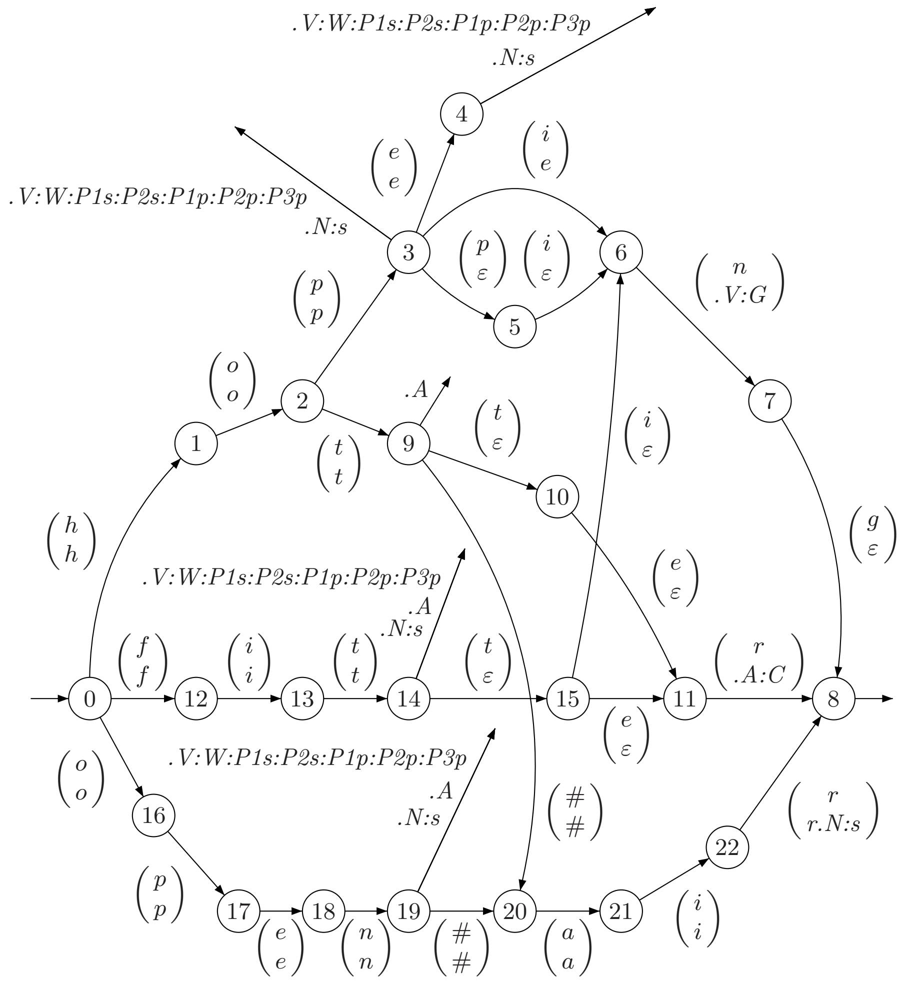


Figure 3.4. A p-sequential transducer for the words and tags in Fig. 3.3.

edges of the transducer, but the increase in size remains within reasonable proportions because inflectional suffixes are usually short. After this operation, a variant of algorithm ToSequentialTransducer (section 1.5) can be applied.

A dictionary represented as a transducer can be used to produce a dictionary for generation, by swapping input and output. The resulting transducer can be processed so as it becomes p-sequential too, provided that the dictionary is finite.

Fig. 3.5 shows an approximation of the preceding transducer by an acyclic automaton or DAWG. Most of the letters in the word form are identical to letters in the lemma and are not explicitly repeated in the output. The end of the output is shifted to the right and attached to terminal states, with an integer indicating how many letters at the end of the word form are not part of the lemma. When several output strings are possible for the same word, they are concatenated and the result is attached to a terminal state. During minimization of the DAWG, terminal states can be merged only if the output strings attached to them are identical. For the tag set used in Fig. 3.3, and for all the vocabulary, there are only about 2000 different output strings. The practical advantage of this solution is that output strings are stored in a table that need not be compressed and is easy to search for word tags.

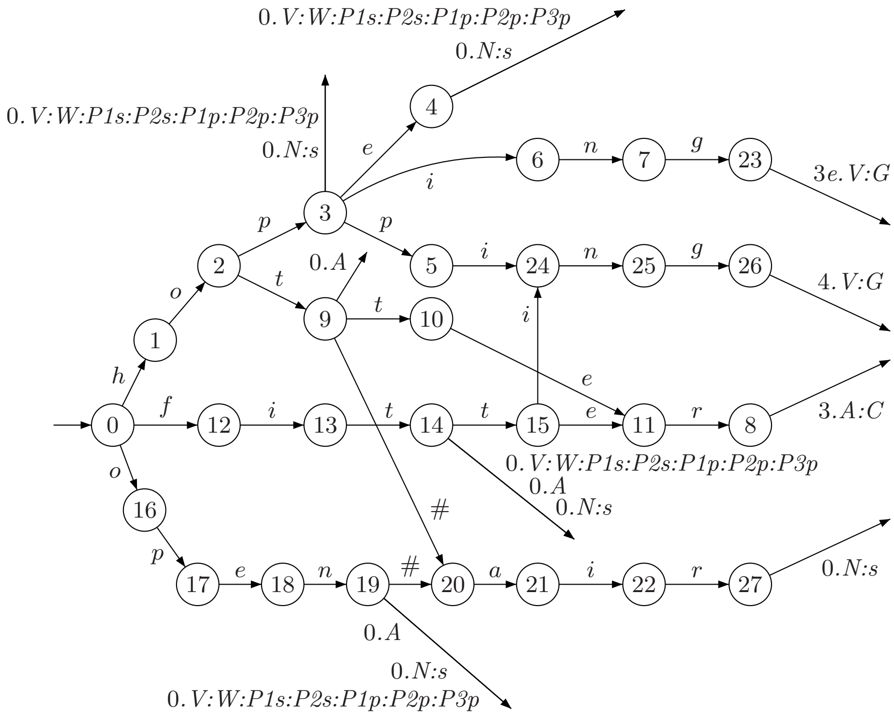


Figure 3.5. The DAWG for the words and tags in Fig. 3.3.

In the previous figures, we have presented the same dictionary in different forms. The form containing most redundancy is the list (Fig. 3.3): parts of words are repeated, not only in lemmas and inflected forms, but also across different entries. The DAWG (Fig. 3.5) is virtually free of this redundancy, but it is unreadable and cannot be updated directly. In fact, linguistic maintenance must be carried out on yet another form, the dictionary of lemmas used to generate the list of Fig. 3.3. The dictionary of lemmas is readable and presents little redundancy, two fundamental features for linguistic maintenance. But the only way to exploit it computationally is to generate the list – a form with huge redundancy – and then the DAWG. The flexibility of finite automata is essential to this practical organization.

The main difficulties with dictionary-based lexical tagging are lexical lacunae, errors and ambiguity.

Lexical lacunae, i.e. words not found in a dictionary, are practically impossible to avoid due to the continuous creation and borrowing of new words. Simple stopgaps are applicable by taking into account the form of words: for example, in English, a capitalized token not found in the dictionary is often a proper noun.

Lexical errors are errors producing forms which do not belong to the vocabulary of the language, e.g. *coronre* for *coroner*[3]. Lexical errors are impossible to distinguish from lexical lacunae. A few frequent errors can be inserted in dictionaries, but text writers are so creative that this solution cannot be implemented systematically. In order to deal with errors (find suggestions for corrections, retrieve lexical information about correct forms), an electronic dictionary can be used. By looking up in an error-tolerant way, we find correct forms that are close to the erroneous form.

Lexical ambiguity refers to the fact that many words should be assigned distinct tags in relation to context, like *fit*. About half the forms in a text are lexically ambiguous. Lexical ambiguity resolution is dealt with in section 3.2.4.

In some languages, sequences of words are written without delimiter in certain conditions, even if the sequence is not frozen. In German, *ausschwimmen* "to swim out" is the concatenation of *aus* "out" and *schwimmen* "swim". Obviously, dictionary lookup has to take a special form in cases where a token comprises several words.

Performing the lexical analysis of a text with a set of dictionaries requires adapted software, like the open-source system Unitex. Fig. 3.6 shows the result of the lexical analysis of an English text by Unitex. This system can also be used for the management of the dictionaries in their different forms, and for the operations on words that we will present in section 3.2.

3.1.5. Morphological analysis

Given a word in a written text, represented by a sequence of letters, how do you analyse it into a sequence of underlying morphological elements? This problem is conveniently solved by the dictionary methods of the preceding section, except when the number of morphological elements that make up words is too large. This happens with agglutinative languages. English and other Indo-European languages are categorized as inflected languages. A few agglutinative languages are spoken in Europe: Turkish, Hungarian, Finnish, Basque... and many others are from all other continents. In such languages, a word is a concatenation of morphological elements, usually written without delimiters[4]. For

[3] Errors can also produce words which belong to the vocabulary, like *corner*.

[4] When morphological elements are delimited by spaces, like in Sepedi, an African agglutinative language, the problem of recognizing their combinations is quite different.

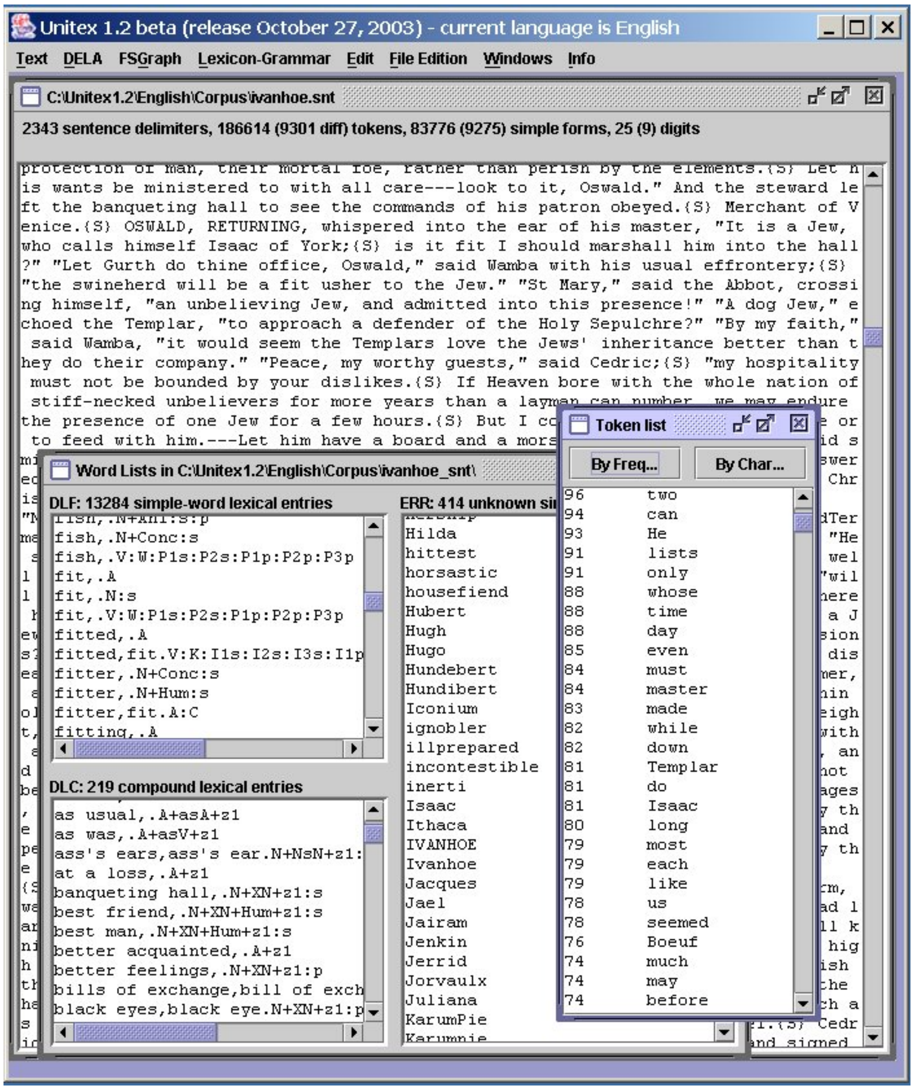


Figure 3.6. Lexical analysis of an English text by Unitex.

example, the following Korean sequence, transliterated into the Latin alphabet: *manasiôs'takojocha* "even that (he) met", comprises 6 elements:

- *mana* "meet"
- *si* (honorification of grammatical subject)
- *ôs'* (past)
- *ta* (declarative)

- *ko* "that"
- *jocha* "even"

and can be used in a sentence meaning "(The Professor) even (thought) that (he) met (her yesterday)". The form of each element can depend on its neighbors, so each element has a canonical form or lemma and morphological variants. There are two types of morphological elements: stems, which are lexical entries, like "meet" in the Korean example, and grammatical affixes, like tense, mood or case markers. Morphological analysis consists of segmenting the word and finding the lemma and grammatical tag of each underlying morphological element. The converse problem, morphological generation, is relevant to machine translation in case of an agglutinative target language: words are constructed as sequences of morphological elements, but you have to apply rules to spell the resulting word correctly.

Finite transducers are usually convenient for representing the linguistic data required for carrying out morphological analysis and generation. For example, Fig. 3.7 represents a part of English morphology as if it were agglutinative. This transducer analyses *removably* as the combination of three morphological elements, *remove.V*, *able.A* and *ly.ADV*, and inserts plus signs in order to delimit them. The transducer roughly respects a natural alignment between written forms and underlying analyses. It specifies two types of information: how written forms differ from underlying forms, and which combinations of morphological elements are possible. Grammatical codes are assigned to morphological elements: verb, adjective, tense/mood suffix, adverb. Some other examples of words analyzed by this transducer are *remove*, *removable*, *removed*, *removing*, *accept*, *acceptable*, *acceptably*, *accepted*, *accepting*, *emphatic*, *emphatically*, *famous* and *famously*. The four initial states should be connected to parts of the dictionary representing the stems that accept the suffixes represented in the transducer.

In this toy example, it would have been simpler to make a list of all suffixed forms with their tags. However, combinations of morphological elements are more numerous and more regular in agglutinative languages than in English, and they justify the use of a transducer.

Transducers of this kind obviously have to be manually constructed by linguists, which implies the use of a convenient, readable graphic form, so that errors are easily detected and maintenance is possible. A widely used set of conventions consists in attaching labels to states and not to edges. States are not explicitly numbered. This graphic form is sometimes called a "graph". For example, Fig. 3.8 shows the same transducer as Fig. 3.7 but with this presentation. The expressive power is the same. When the transducer is used in an operation on text or with another transducer, it is compiled into the more traditional form. During this compilation, states are assigned arbitrary numbers.

The main challenge with algorithmic tools for morphological processing is the need to observe two constraints: manually constructed data must be presented in a readable form, whereas data directly used to process text must be coded in adapted data structures. When no format is simultaneously readable and adapted to efficient processing, the data in the readable form must be auto-

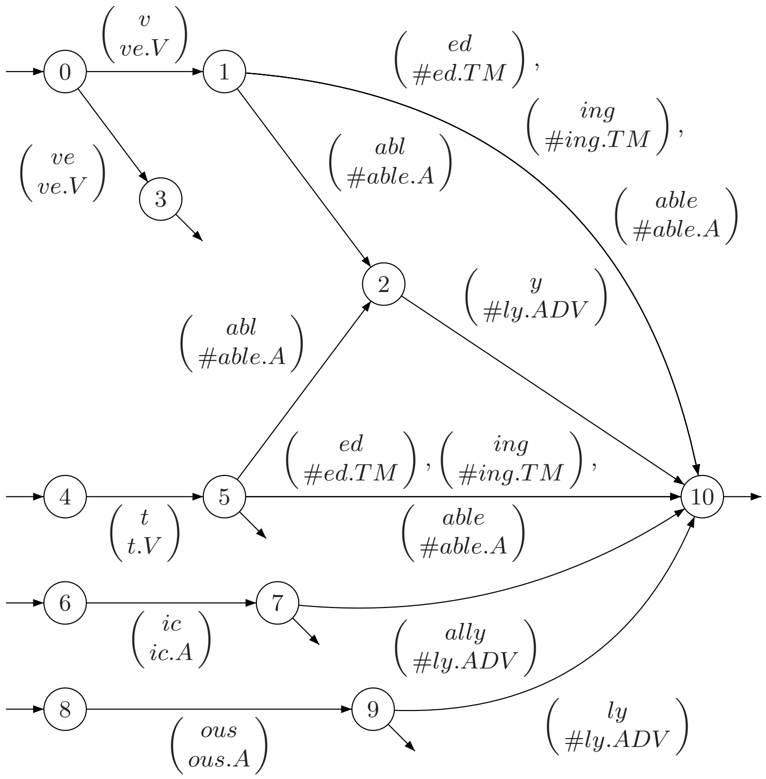


Figure 3.7. Morphological analysis in English.

matically compiled into the operation-oriented form. This organization should not be given up as soon as operation-oriented data are available: linguistic maintenance, i.e. correction of errors, inclusion of new words, selection of vocabulary for applications etc., can only be done in the readable form.

Transducers for morphological analysis are usually ambiguous. This happens when a written word has several morphological analyses, like *flatter*, analyzable as *flatter.V* in *Advertisements flatter consumers*; and as *flat.A+er.C* in *The ground is flatter here*. The fact that transducers are ambiguous is not a problem for linguistic description, since ambiguous transducers are as readable as unambiguous ones. However, it can raise algorithmic problems: in general, an ambiguous transducer cannot be traversed in an input-wise deterministic way. In inflected languages, this problem is avoided by substituting p-sequential transducers to ambiguous transducers, but this solution is no longer valid for most agglutinative languages. When ambiguity affects the first element in a long sequence of morphological elements, shifting output labels to terminal output strings would change the natural alignment between input and output to such

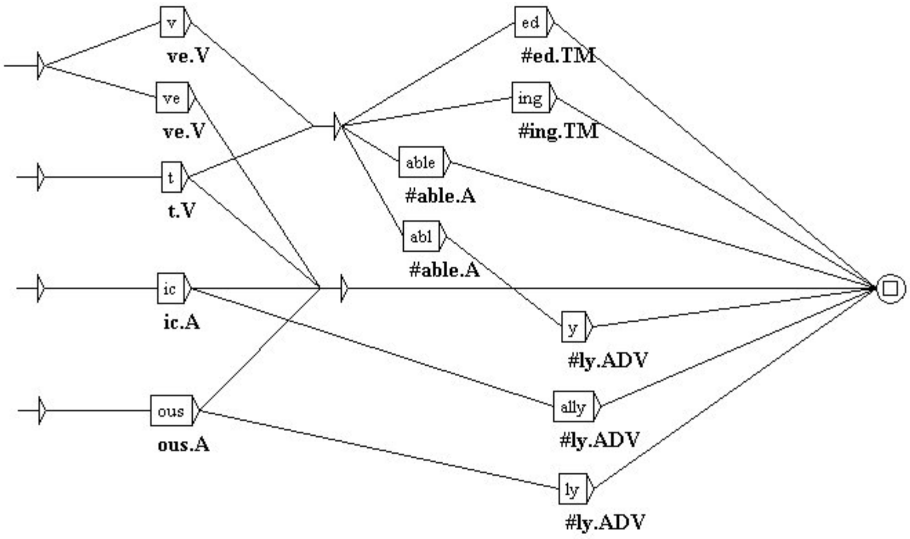


Figure 3.8. Morphological analysis in English.

an extent that the number of states and edges of the transducer would explode.

Therefore, algorithm ToSequentialTransducer is not applicable: ambiguous transducers have to be actually used. There are several ways of automatically reducing the degree of input-wise nondeterminism of an ambiguous transducer. We will see two methods which can be applied after the alignment of the transducer has been tuned so as to be input-wise synchronous (see section 3.1.1). Both methods will be exemplified on the transducer of Fig. 3.8, which has 4 initial states. These distinct initial states encode dependencies between stems and suffixes, as we will see in the last page of this section. For simplicity's sake, the stems are not included in this figure: thus, we will consider it as a collection of 4 transducers, and artificially maintain the 4 initial states.

The first method consists in determinizing (algorithm NFAtoDFA, section 1.3.3) and minimizing (section 1.3.4) the ambiguous transducer, considering it as an automaton over a finite alphabet $X \subset A^\star \times B^\star$. In general, the resulting transducer is still ambiguous: distinct edges can have the same origin, the same input label, and distinct ends, $(p, a : u, q)$ and $(p, a : v, r)$, but only if their output labels u and v are distinct. The transducer of Fig. 3.9 is the result of the application of this method to the transducer of Fig. 3.8. Applying the resulting transducer to a word involves a variant of the nondeterministic search of section 1.3.2 (algorithm IsAccepted), but the search is quicker than with the original transducer, because algorithm NFAtoDFA reduces the nondeterminism of the transducer.

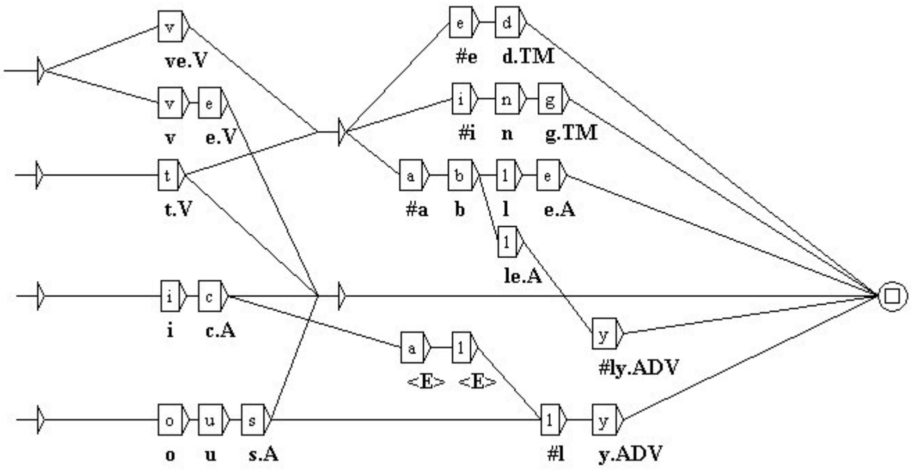


Figure 3.9. An ambiguous transducer determinized as an automaton.

In order to introduce the second method, we define a new generalization of p-sequential transducers. We will allow differences between output labels of the same path to occur at any place as long as they remain strictly local. Formally, a generalized sequential transducer is a finite transducer with a finite set of output labels $I(i)$ for the initial state i, a finite set of output labels $T(q)$ for each terminal state q, and with the following properties:

- it has at most one initial state,
- it is input-wise synchronous,
- for each pair of edges $(p, a\!:\!u, q), (p, a\!:\!v, r)$ with the same origin and the same input label, $q = r$.

A transduction is realized by a generalized sequential transducer if and only if it is the composition of a sequential transduction with a finite substitution. Thus, such a transduction is not necessarily a word function: two edges can have the same origin, the same input label, the same end and distinct output labels, $(p, a\!:\!u, q)$ and $(p, a\!:\!v, q)$. However, given the input label of a path, a generalized sequential transducer can be traversed in an input-wise deterministic way, even if it is ambiguous.

The second method constructs a generalized sequential transducer equivalent to the ambiguous transducer. When two edges with the same origin and the same input label have different output labels and different ends, output labels are shifted to adjacent edges to the right, but not necessarily until a terminal state is reached. The condition for ceasing shifting a set of output strings to the right is the following. Consider the set $E_{p,a}$ of all edges with origin p and input label a. Each edge $e \in E_{p,a}$ has an output label $u_e \in B^*$ and an end $q_e \in Q$. Consider the finite language $L_{p,a} \subset B^*Q$ over the alphabet $B \cup Q$ defined by $L_{p,a} = \{u_e q_e | e \in E_{p,a}\}$. If we can write $L_{p,a} = MN$ with $M \subset B^*$

and $N \subset B^*Q$, then

- create a new state r; let r be terminal if and only if at least one of the states q_e is terminal;
- substitute a new set of edges for $E_{p,a}$: the edges $(p, a\!:\!v, r)$ for all $v \in M$;
- shift the rest of output labels further to the right by replacing each edge $(q_e, b : w, s)$ with the edges $(r, b : xw, s)$ for all $x \in N$; for each terminal state among the states q_e, substitute $NT(q_e)$ for $T(q_e)$.

There can be several ways of writing $L_{p,a} = MN$: in such a case, the longer the elements of M, the better.

If the transduction realized by the ambiguous transducer is finite, this algorithm terminates; otherwise it is not certain to terminate. If it does, we obtain an equivalent generalized sequential transducer like that of Fig. 3.10.

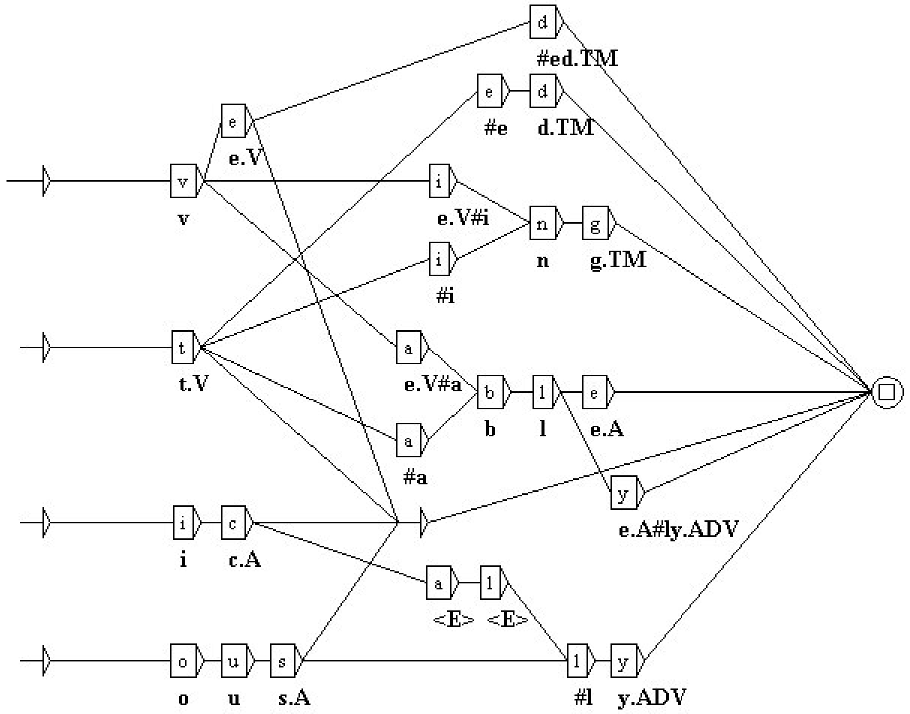


Figure 3.10. A generalized sequential transducer.

Transducers for morphological analysis like those of Fig. 3.7–3.10 can be used to produce transducers for morphological generation, by swapping input and output. The resulting transducer can be processed with the same methods as above in order to reduce nondeterminism.

When observable forms and underlying lemmas are very different, the description of morphology becomes complex. At the same time, it must still be hand-crafted by linguists, which requires that it is made of simple, readable parts, which are combined through some sort of compilation. For example, if both morphological variations and combinatorial constraints are complex, they are better described separately. Combinatorial constraints between morphological elements are described in an automaton at the underlying level, i.e. of lemmas and grammatical codes, as in Fig. 3.11.

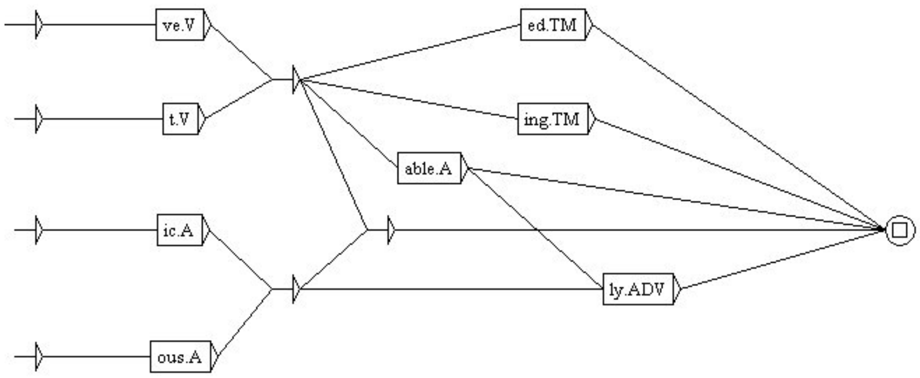


Figure 3.11. Combinatorial constraints between morphological elements.

Morphological changes are described in a transducer, with input at the level of written text and output at the underlying level. This is done in Fig. 3.12, which is more complex than Fig. 3.8, but also more general: it allows for more combinations of suffixes, i.e. *-ingly*, which was not included in Fig. 3.8 because it is not acceptable combined with *remove*.

How can we use these two graphs for morphological analysis? There are two solutions. The simpler solution applies the two graphs separately. When we apply the transducer of Fig. 3.12 to a word, we obtain, in general, an automaton. The automaton has several paths if several analyses are possible, as with *flatter*. Then when we compute the intersection of this automaton with that of Fig. 3.11, this operation selects those analyses that obey the combinatorial constraints. The algorithm of intersection of finite automata is based on the principle that the set of states of the resulting automaton is the Cartesian product of the sets of states of the input automata.

A more elaborate solution consists in performing part of the computation in advance. The automaton of Fig. 3.11 and the transducer of Fig. 3.12 do not depend on input text; they can be combined into the transducer of Fig. 3.8. If the automaton recognizes a set L and the transducer realizes a relation R, the operation consists in computing a transducer that realizes the relation R with its output restricted to L. This can be implemented, for instance, by applying algorithm ComposeTransducers (section 1.5) to the transducer of R and a transducer realizing the identity of L. Note that this algorithm is a variant of

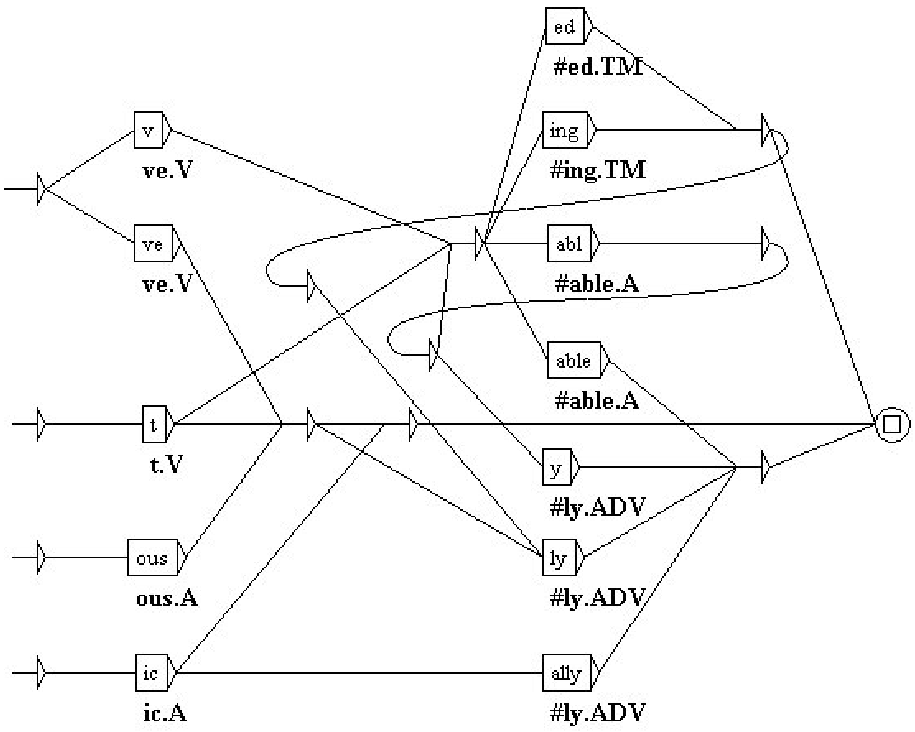


Figure 3.12. Morphological changes.

the algorithm of intersection of finite automata.

Morphological analysis and generation are not independent of the dictionary of stems: combinations of stems with affixes obey compatibility constraints, e.g. the verb *fit* does not combine with the suffix *-able*; stems undergo morphological variations, like *remove* in *removable*. Due to such dependencies, morphological analysis, in general, cannot be performed without vocabulary recognition. A dictionary of stems is manually created in the form of a list of many thousands of items and then compiled, so the interface with a transducer for morphological analysis requires practical organization. Combinatorial constraints between stems and affixes are represented by assigning marks to stems to indicate to which initial states of the automaton each stem must be connected. During compilation, the dictionary of stems and the automaton of combinatorial constraints are combined into an automaton. Morphological variations of stems are taken into account in the transducer; if analogous stems behave differently in an unpredictable way, like *fit/fitted* and *profit/profited*, marks are assigned to stems and the transducer refers to these marks in its output. If these provisions are taken, the operation on the automaton of constraints and the transducer of variations can be performed as above and produces a satisfactory result.

In this case, the description is distributed over two data sets: an automaton and a transducer, and the principle of the combination between them is that the automaton is interpreted as a restriction on the output part of the transducer.

It is often convenient to structure manual description in the form of more than two separate data sets: for example, one for the final *e* of verbs like *remove*, another for the final *e* of *-able*, another for variations between the forms *-ly*, *-ly*, *-y* of the adverbial suffix etc. This strategy can be implemented in three ways, depending on the formal principle adopted to combine the different elements of description: composition of transductions, intersection of transducers, and commutative product of bimachines.

3.1.6. Composition of transductions

The simplest of these three techniques involves the composition of transductions. Specialists in language processing usually refer to this operation by the bucolic term "cascade". The principle is simple. The data for morphological analysis or generation consists of a specification of a transduction between input strings and output strings. This transduction can be specified with several transducers. The first transducer is applied to input strings, the next transducer to the output of the first, and so on. The global transduction is defined as the composition of all the transductions realized by the respective transducers.

For example, Fig. 3.8 is equivalent to the composition of the transductions specified by Figs. 3.13–3.16. Fig. 3.13 delimits and tags morphological elements,

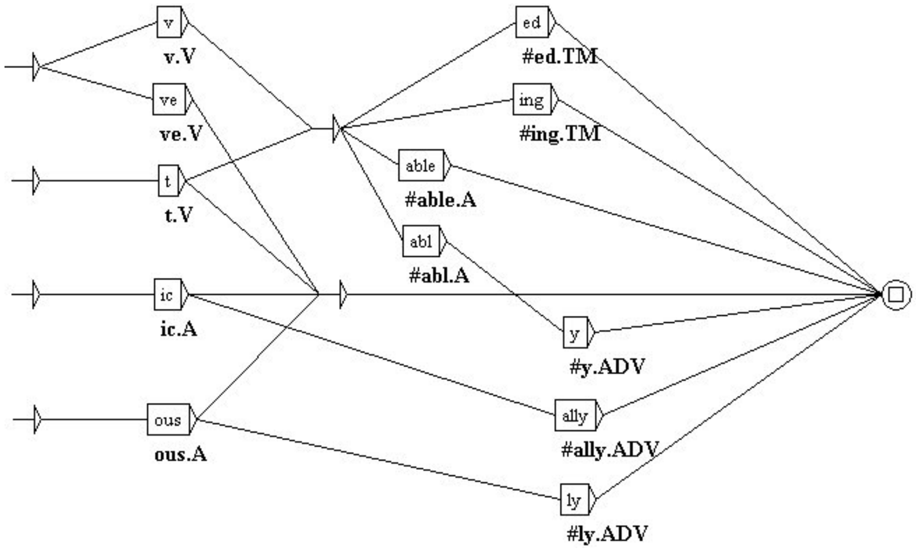


Figure 3.13. A cascade: first transducer.

but does not substitute canonical forms for variants. Fig. 3.14 inserts the final *e* of the canonical form of *remove*. In Fig. 3.14, the input label @ stands for

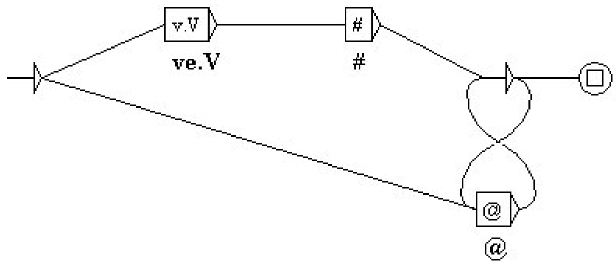


Figure 3.14. A cascade: second transducer.

a default input symbol: it matches the next input symbol if, at this point of the transducer, no other symbol matches. The output label @ means an output symbol identical to the corresponding input symbol. Fig. 3.15 inserts the final *e* of the canonical form of *-able*. Fig. 3.16 assigns the canonical form to the

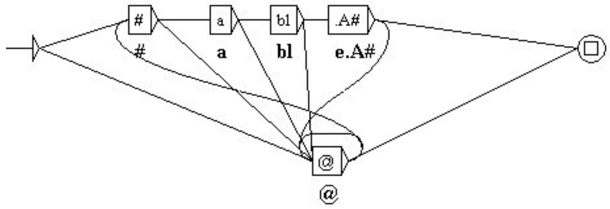


Figure 3.15. A cascade: third transducer.

variants of the adverbial suffix *-ly*.

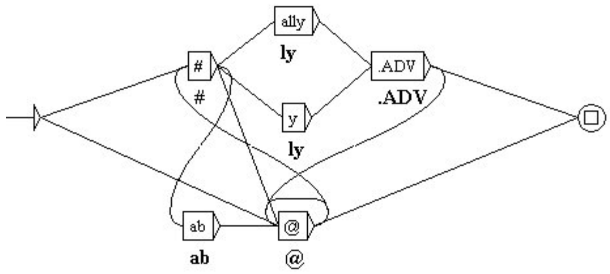


Figure 3.16. A cascade: fourth transducer.

During the application of a transducer, the input string is segmented according to the input labels of the transducer, and the output string is a concatenation of output labels. When transducers are applied as a cascade, the segmentation of the output string of a transducer is not necessarily identical to the segmentation induced by the application of the next. The global transduction is not changed if we modify the alignment of one of the transducers, provided that it realizes the same transduction.

As an alternative to applying several transducers in sequence, one can precompute an equivalent transducer by algorithm ComposeTransducers, but the application of the resulting transducer is not necessarily quicker, depending on the number, size and features of the original transducers.

The principle of composition of rules was implemented for the first time in... the 5th century B.C., in Panini's Sanskrit grammar, in order to define Sanskrit spelling, given that the form of each element depends on its neighbors.

Composition of relations is not a commutative operation. In our example of a cascade, the transductions of Figs. 3.14–3.16 can be permuted without changing the result of the composition, but they must be applied after Fig. 3.13, because they use the boundaries of morphological elements in their input, and these boundaries are inserted by the transduction of Fig. 3.13. In general, simple transductions read and write only in a few regions of a string, but interactions between different transductions are observed when they happen to read or write in the same region.

The principle of defining a few levels in a determined order between the global input level and the global output level is often natural and convenient. The alphabet of each intermediate level is a subset of $A \cup B$. In morphological generation, the level of underlying morphological elements may have something to do with a previous state of the language, the sequence of levels being connected to successive periods of time in the history of language changes.

However, in a language with complex morphological variations represented by dozens of rules, the exclusive use of composition involves dozens of ordered levels. This complicates the task of the linguist, because he has to form a mental image of each level and of their ordering.

Intuitively, when two morphological rules are sufficiently simple and unrelated, one feels that it should be possible to implement them independently, without even determining in which order they apply: hence the term "simultaneous combination". In spite of this intuition, rules cannot be formalized without specifying how they are interpreted in case of an overlap between the application sites of several rules (or even of the same rule): if rules apply to two sites uv and vw, the value of v taken into account for uvw can involve the input or the output level, or both. Various formal ways of combining formal rules have been investigated. Two main forms of simultaneous combination are presently in use.

3.1.7. Intersection of transducers

The intersection of finite transducers can be used to specify and implement morphological analysis and generation. The alignment between input and output strings is an essential element of this model. This alignment must be literal, i.e. each individual input or output symbol must be aligned either with a single symbol or with ε. Several alignments are usually acceptable, e.g.

$$\binom{u}{u}\binom{s}{s}\binom{\varepsilon}{e}\binom{\varepsilon}{.V}\binom{\varepsilon}{\#}\binom{e}{e}\binom{d}{d}\binom{\varepsilon}{.TM}$$

and

$$\binom{u}{u}\binom{s}{s}\binom{e}{e}\binom{\varepsilon}{.V}\binom{\varepsilon}{\#}\binom{\varepsilon}{e}\binom{d}{d}\binom{\varepsilon}{.TM}$$

but one must be chosen arbitrarily.

Formally, an alignment over A and B is a subset of the free monoid X^*, where X is a finite subset of $A^\star \times B^\star$. An alignment is literal if it is a subset of $((A \mid \varepsilon) \times (B \mid \varepsilon))^*$.

The alignment is determined in order to specify explicitly the set of all pairs $(u\!:\!v) \in (A \mid \varepsilon)\times(B \mid \varepsilon)$ that will be allowed in aligned input/output pairs for all words of the language. Since all elements in the alignment will be concatenations of elements in this set, we can call it X. In the English example above, this set can comprise letters copied to output:

$$\binom{v}{v},\binom{e}{e},\binom{d}{d},\binom{i}{i},\binom{n}{n},\binom{g}{g},\binom{a}{a},\binom{b}{b},$$

$$\binom{l}{l},\binom{t}{t},\binom{c}{c},\binom{y}{y},\binom{o}{o},\binom{u}{u},\binom{s}{s},$$

plus a few insertions:

$$\binom{\varepsilon}{e},\binom{\varepsilon}{.V},\binom{\varepsilon}{\#},\binom{\varepsilon}{.TM},\binom{\varepsilon}{.A},\binom{\varepsilon}{l},\binom{\varepsilon}{.ADV},$$

and two deletions of letters:

$$\binom{a}{\#},\binom{l}{\varepsilon}$$

The set of aligned input/output pairs for all words of the language is viewed as a language over the alphabet X. This language is specified as the intersection of several regular languages. Each of these languages expresses a constraint that all input/output pairs must obey, and the intersection of the languages is the set of pairs that obey simultaneously all the constraints. Since these regular languages share the same alphabet $X \subset A^* \times B^*$, they can be specified by transducers over A and B. For example, the transducers in Figs. 3.17–3.20 specify necessary conditions of occurrence for some of the elements of X. In Fig. 3.17, the label @ denotes a default symbol. It matches the next member of X if and only if no other label explicitly present at this point of the graph

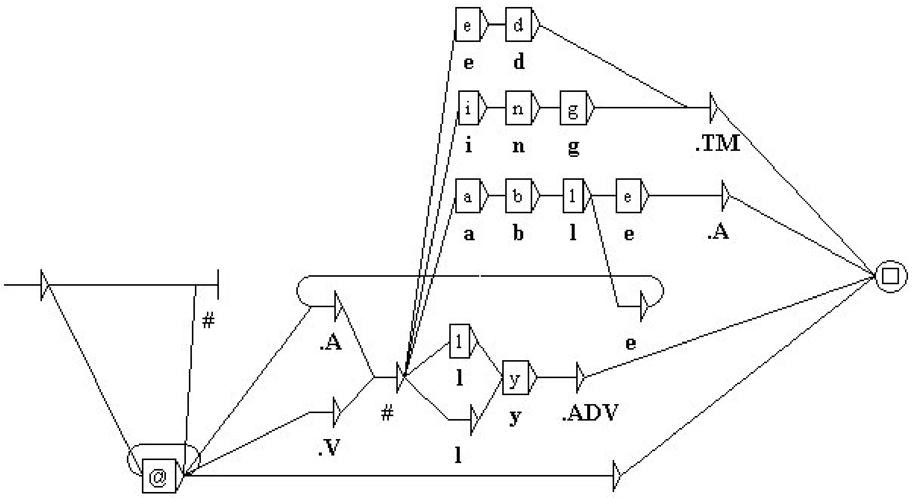


Figure 3.17. Conditions of occurrence of $(\varepsilon : \#)$.

does. One of the states has no outgoing edge and is not terminal: it is a sink state which is used to rule out the occurrence of $(\varepsilon : \#)$ when it is not preceded by $(\varepsilon : .A)$ or $(\varepsilon : .V)$.

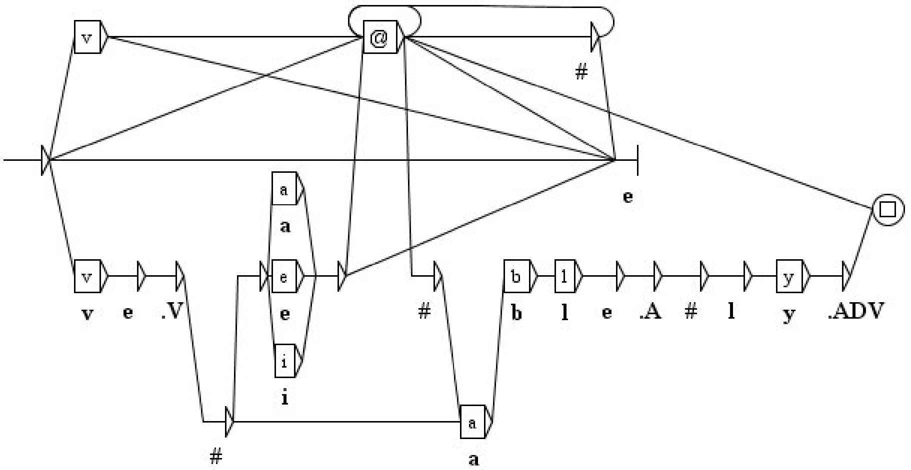


Figure 3.18. Conditions of occurrence of $(\varepsilon : e)$.

In order to be complete, we should add transducers to specify the conditions

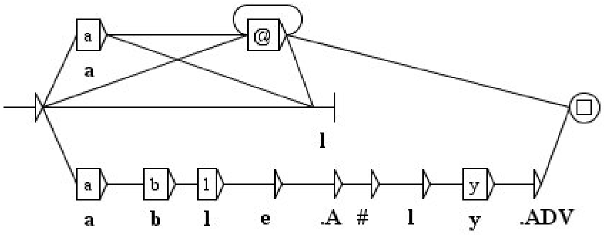


Figure 3.19. Conditions of occurrence of $(\varepsilon : l)$.

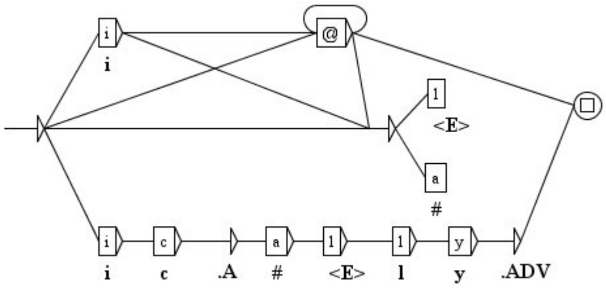


Figure 3.20. Conditions of occurrence of $(a : \#)$ and $(l : \varepsilon)$.

of occurrence of $(\varepsilon : .V)$, $(\varepsilon : .TM)$, $(\varepsilon : .A)$ and $(\varepsilon : .ADV)$.

The intersection of transducers is computed with the algorithm of intersection of automata, considering transducers as automata over X. The resulting transducer checks all the constraints simultaneously. This operation of intersection of transducers is equivalent to the intersection of languages in the free monoid $X^\star$, but not to the intersection of relations in $A^\star \times B^\star$, because the intersection of relations does not take into account alignment. (In addition, an intersection of regular relations is not necessarily regular.)

As opposed to the framework of composition of transductions, all the transducers describe correspondences between the same input level and the same output level. This is why this model is called "two-level morphology". Composition of transductions and intersection of transducers are orthogonal formalisms, and they can be combined: several batches of two-level rules are composed in a definite order.

Two-level constraints expressed as transducers are hardly readable, and expressing them as regular expressions over X would be even more difficult and error-prone. In order to solve this problem of readability, specialists in two-level morphology have designed an additional level of compilation. Rules are expressed in a special formalism and compiled into transducers. These transducers are then intersected together. The formalism of expression of two-level rules involves logical operations and regular expressions over X. For example, the following rule is equivalent to Fig. 3.17:

$$\binom{\varepsilon}{\#} \Longrightarrow \left(\binom{\varepsilon}{.A} \mid \binom{\varepsilon}{.V}\right) \underline{\quad} \left(\binom{a}{a}\binom{b}{b}\binom{l}{l}\binom{\varepsilon}{e}\binom{\varepsilon}{.A}\right)^{*}$$
$$\Bigg(\Bigg(\bigg(\binom{e}{e}\binom{d}{d} \mid \binom{i}{i}\binom{n}{n}\binom{g}{g}\bigg)\bigg)\binom{\varepsilon}{.TM} \mid \binom{a}{a}\binom{b}{b}\binom{l}{l}\binom{e}{e}\binom{\varepsilon}{.A} \mid$$
$$\bigg(\bigg(\binom{l}{l} \mid \binom{\varepsilon}{l}\bigg)\bigg)\binom{y}{y}\binom{\varepsilon}{.ADV}\Bigg)\Bigg)$$

This type of rule is more readable than a transducer, because it is structured in three separate parts: the symbol involved in the rule, here $(\varepsilon : \#)$, the left context (before ____), and the right context.

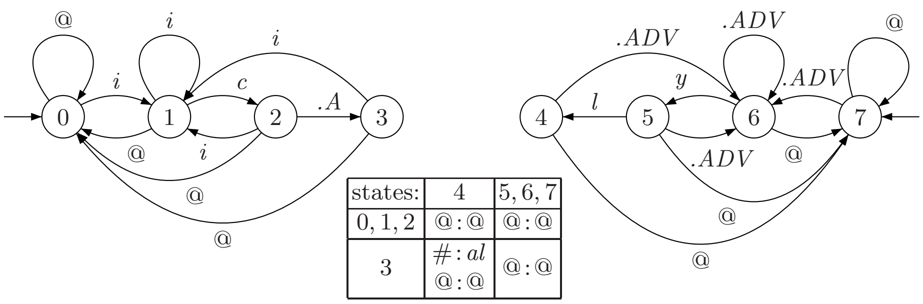
In this model, input and output are completely symmetrical: the same description is adapted for morphological analysis and generation.

3.1.8. Commutative product of bimachines

A bimachine is structured in three parts:

- a description of the left context required for the rule to apply,
- a similar description of the right context, and
- a mapping table that specifies a context-dependent mapping of input symbols to output symbols.

As opposed to two-level rules, left and right context are described only at input level. Fig. 3.21 is a representation of a bimachine that generates the variant *-ally* of the adverbial suffix *-ly* in *emphatically*.



states:	4	5, 6, 7
0, 1, 2	@ : @	@ : @
3	# : *al* @ : @	@ : @

Figure 3.21. Bimachine generating the variant *-ally* of the adverbial suffix *-ly*.

In this figure, the automaton on the left represents the left context and recognizes occurrences of the sequence $ic.A$. Whenever this sequence occurs, the automaton enters state 3. In the automaton, the label @ represents a default symbol: it matches the next input symbol if no other label at this point of the automaton does. The automaton on the right similarly recognizes occurrences of *ly.ADV*, but from right to left. Whenever this sequence occurs, the automaton enters state 4. The table specifies the mapping of input symbols to output symbols. The alphabets A and B have a nonempty common subset. In the table, @ : @ represents a default mapping: any input symbol not explicitly specified in the table is mapped onto itself. The symbol # is mapped to *al* when its left and right context is such that the respective automata are in states 3 and 4, i.e. when it is preceded by *ic.A* and followed by *ly.ADV*. Other symbols in such a context, and all symbols in other contexts, are copied to output. Thus, the bimachine maps occurrences of *ic.A#ly.ADV* to *ic.Aally.ADV* and leaves everything else unchanged. The input/output alignment that underlies the bimachine is always input-wise synchronous.

Formally, a bimachine over alphabets A and B is defined by

- two deterministic automata over A; let $\overrightarrow{Q}$ and $\overleftarrow{Q}$ be the sets of states of the two automata; the distinction between terminal vs. non-terminal states is not significant;
- a function $\gamma : \overrightarrow{Q} \times A \times \overleftarrow{Q} \longrightarrow B^*$, which is equivalent to the mapping table in Fig. 3.21.

The transduction realized by a bimachine is defined as follows. One performs a search in the left automaton controlled by the input word $u = u_1 u_2 \cdots u_n$. If this search is possible right until the end of the word, a sequence $\overrightarrow{q_0}\overrightarrow{q_1} \cdots \overrightarrow{q_n}$ of states of the left automaton is encountered, where $\overrightarrow{q_0}$ is the initial state. A similar search in the right automaton is controlled by $u_n \cdots u_2 u_1$. If the search can be completed too, states $\overleftarrow{q_n} \cdots \overleftarrow{q_1}\overleftarrow{q_0}$ of the right automaton are encountered, where $\overleftarrow{q_n}$ is the initial state.

The output string for the symbol u_i of u is $\gamma(\overrightarrow{q_{i-1}}, u_i, \overleftarrow{q_i})$ and the output for u is the concatenation of these output strings. If one of the searches could not be completed, or if one of the output strings for the letters is undefined, then the output for u is undefined.

A transduction is realized by a bimachine if and only if it is regular and a function.

The use of bimachines for specifying and implementing morphological analysis or generation requires that they can be combined to form complete descriptions. In the mapping table of Fig. 3.21, the default pair @ : @ occurs in all four cases; the bimachine specifies an output string for some occurrences of #, and copies all other occurrences of input symbols. We will say that the bimachine "applies" to these occurrences of #, and "does not apply" to other occurrences of input symbols. In morphology, separate rules belonging to the same description are complementary in so far as they do not "apply" to the same occurrences of input symbols. This idea can be used to define a notion of combination of bimachines over the same alphabets A and B.

Formally, we say that a bimachine "applies" to an input symbol a in a given context, represented by two states $\overrightarrow{q}$ and $\overleftarrow{q}$, if and only if $\gamma(\overrightarrow{q}, a, \overleftarrow{q})$ either is undefined or is not equal to a. It "does not apply" if and only if

$\gamma(\overrightarrow{q}, a, \overleftarrow{q}) = a$. If two bimachines never apply to the same symbol in the same input sequence, a new bimachine over the same input and output alphabets A and B can be defined so that the output for a given input symbol is specified by the bimachine that applies. The output is a copy of the input symbol if none of them applies. (Each automaton of the new bimachine is constructed from the corresponding automata of the two bimachines, with the algorithm of intersection of automata.) This operation on bimachines is commutative and associative; its neutral element is a bimachine that realizes the identity of A. We call this operation "commutative product".

The commutative product of a finite number of bimachines is defined if and only if it is defined for any two of them.

With this operation, linguists can manually construct separate bimachines, or rules, and combine them. These manually constructed rules must also be readable. This can be achieved by ensuring that the rules are presented according to the following conventions and have the following properties.

- Final states are specified in the two automata. The content of the mapping table does not depend on the particular states reached when exploring the context, but only on whether these states are terminal or not. For example, in Fig. 3.21, states 3 and 4 would be specified as terminal.
- In the mapping table, whenever at least one of the two states representing the context is non-terminal, input symbols are automatically copied to output, as in Fig. 3.21. When both states are terminal, only the input/output pairs for which the output string is different from the input symbol are specified. Let I be the set of input symbols that occur in the input part of these pairs: if both states are terminal and the input symbol is in I, the rule applies; otherwise, it does not apply and input is copied to output.
- The languages recognized by the two automata are of the form A^*L and A^*R, as in Fig. 3.21. Therefore, it suffices to specify L and R; automata for A^*L and A^*R can be automatically computed. In addition, the mirror image of R is specified instead of R itself, for the sake of readability.

The bimachine of Fig. 3.21 has these properties and is represented with these conventions in Fig. 3.22.

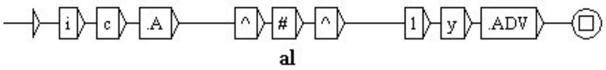


Figure 3.22. The bimachine of Fig. 3.21 with the conventions for manually constructed rules.

This figure represents L, R and the input/output pairs for which the rule applies. These three parts are separated by the states labeled $\wedge$.

The commutative product of two rules is defined if and only if $A^*L_1 \cap A^*L_2$, $A^*R_1 \cap A^*R_2$ and $I_1 \cap I_2$ are not simultaneously nonempty. This condition is tested automatically on all pairs in a set of rules written to be combined by commutative product. If the three intersections are simultaneously nonempty for a pair of rules, the linguist is provided with the set of left contexts, right contexts and input symbols for which the two rules conflict, and he/she can modify them in order to resolve the conflict. (A hierarchy or priorities between rules would theoretically be possible but would probably make the system more complex and its maintenance more difficult.)

The advantages of bimachines for specifying and implementing morphological analysis and generation are their readability and the fact that only differences between input and output need to be specified.

Bimachines are equivalent to regular word functions and, in principle, cannot represent ambiguous transitions. They have to be adapted in order to allow for limited variations in output. Take, for example, the generation of the preterite of *dream*: for a unique underlying form, *dream.V#ed.TM*, where *#ed.TM* is an underlying tense/mood suffix, there are two written variants: *dreamed* and *dreamt*. Such variations are limited; in agglutinative languages, they can occur at any point of a word, not necessarily just at the end. This problem is easily solved in the same way as we did for minimizing ambiguous transducers in section 3.1.5: by composition with finite substitutions. Bimachines realize transductions; several of these transductions can be composed in a definite order together or with finite substitutions.

In the example of *dream.V#ed.TM*, the two variants can be generated by introducing 3 new symbols *1*, *2* and *3*, and

- a bimachine that produces *dream.V#1ed.TM*,
- a finite substitution producing *dream.V#2ed.TM* and *dream.V#3ed.TM*, and
- a second bimachine that outputs *dreamed* for *dream.V#2ed.TM* and the variant *dreamt* for *dream.V#3ed.TM*.

However, a bimachine is an essentially deterministic formalism. It is adequate for the direct description of morphological generation, because the underlying level is more informative and less ambiguous than the level of written text: thus, for an input string at the level of underlying morphological elements, there will often be a unique output string or limited variations in output. For instance, *flatter* has two representations at the underlying level, but one spelling.

It is possible to do morphological analysis with bimachines, but one has to carry out linguistic description for morphological generation, and automatically derive morphological analysis from it. The method consists in compiling each bimachine (or commutative product of bimachines) into a transducer, and swapping input and output in the transducer. During the compilation of a bimachine into a transducer, the set of states of the transducer is constructed as the Cartesian product of the sets of states of the two automata.

3.1.9. Phonetic variations

Morphological analysis and generation of written text have an equivalent for speech: analysis and generation of phonetic forms. Phonetic forms are represented by strings of phonetic symbols. They describe how words are pronounced, taking into account contextual variants and free variants. An example of contextual phonetic variation in British English is the pronunciation of *more*, with r in *more ice* and without in *more tea*. Free variation is exemplified by *can* which can be either stressed or reduced in *He can see*. The input of analysis is thus a phonetic representation of speech. The output is some underlying representation of pronunciation, which is either conventional spelling, or a specific representation if additional information is needed, such as grammatical information.

The analysis of phonetic forms is useful for speech recognition. Their generation is useful for speech synthesis. A combination of both is a method for spelling correction: generate the pronunciation(s) of a misspelled word, then analyze the phonetic forms obtained.

A difference between phonetic processing and morphological processing is that a text can usually be pronounced in many ways, whereas spelling is much more standardized. In other aspects, the analysis and generation of phonetic forms is similar to morphological analysis and generation. The computational notions and tools involved are essentially the same.

The complexity of the task depends on the writing systems of languages. When all information needed to deduce phonetic strings, including information about phonetic variants, is encoded in spelling, then phonetic forms can be derived from written text without any recognition of the vocabulary. This is approximately the case of Spanish. Most Spanish words can be converted to phonetic strings by transducers, two-level rules or bimachines that do not comprise lexical information. Fig. 3.23 converts the letter c into the phonetic symbol θ before the vowels e and i.

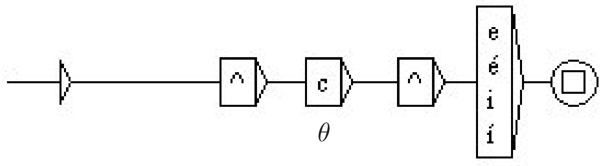


Figure 3.23. A phonetic conversion rule in Spanish.

In most of other languages, spelling is ambiguous: the pronunciation of a sequence of letters depends on the word in which it occurs in an unpredictable way. For example, *ea* between consonants is pronounced differently in *bead, head, beatific, creation, react*; in *read*, the pronunciation depends on the grammatical tense of the verb; in *lead*, it depends on the part of speech of the word: noun or verb. Due to such dependencies, which are most frequent in English and in French, phonetic forms cannot be generated from written texts accurately without vocabulary recognition. In other words, phonetic conversion requires a dictionary, which can be implemented in the form of a transducer and adapted for quick lookup into a generalized sequential transducer like that of Fig. 3.10.

However, even in languages with a disorderly writing system like English or French, the construction of such a dictionary can be partially automated. Transducers, two-level rules or bimachines can be used to produce tentative phonetic forms which have to be reviewed and validated or corrected by linguists.

A transducer that recognizes the vocabulary of a language is larger than a transducer that does not. They also differ in the way they delete word boundaries. In many languages, words are delimited in written text; they are not in phonetic strings, because speech is continuous and there is no audible evidence that a word ends and the next begins. In a transducer that recognizes the vocabulary, edges that delete word boundaries, e.g. edges labelled $(\# : \varepsilon)$, can be associated with ends of words. When the transducer is reversed by swapping input and output, the resulting transducer not only converts phonetics into spelling but also delimits words. The same cannot be done in a transducer that does not recognize vocabulary: since certain edge(s) erase word boundaries independently of context, the reversed transducer will generate optional word boundaries everywhere.

Phonetic strings are usually very ambiguous, and the result of their analysis consists of several hypotheses with different word delimitation, as in Fig. 3.24.

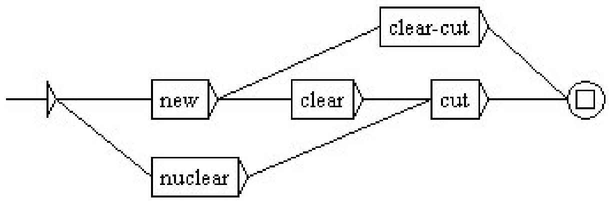


Figure 3.24. Acyclic automaton of the analyses of a phonetic form.

The result of the analysis of ambiguous input is naturally represented in an acyclic automaton like that of Fig. 3.24. We will call it an automaton of analyses, because it represents a set of mutually exclusive analyses. In language engineering, most specialists call such an automaton a "lattice"[5]. The output of a purely acoustic-to-phonetic phase of speech recognition is also an automaton of analyses: a segment of speech signal, i.e. the equivalent of a vowel or a consonant in acoustic signal, cannot always be definitely identified as a single phone (phonetic segment).

[5]This term has a precise mathematical meaning: an ordered set where each pair has a greatest lower bound and a least upper bound. As a matter of fact, in an acyclic graph, edges induce an ordering among the set of states. But the ordered set of states of an acyclic graph is not necessarily a lattice in the mathematical sense. In the acyclic automaton of Fig. 3.24, for instance, *cut* has no greatest lower bound and *new* has no least upper bound. Consequently we will avoid using the term "lattice" for denoting automata of analyses.

3.1.10. Weighted automata

The notions of automata and transducers exemplified in the preceding sections can be extended to weighted automata and transducers. In a weighted automaton, each transition has a weight which is an element of a semiring K; the set of terminal states is replaced with a terminal weight function from the set of states to K. The weight of a path is the product of the weights of its transitions. A Markov chain is a particular case of a weighted automaton.

In such models, weights approximate probabilities of occurrence of symbols in certain contexts, and the semiring is often $\mathbb{R}^+$. For example, in an automaton of analyses which contains phones recognized in a speech signal, weights can be assigned to each transition in order to represent the plausibility of the phone given the acoustic signal. The weighted automaton is exploited by selecting the path that maximizes the product of the weights.

Another example can be derived from Fig. 3.11: the plausibility of occurrence of a morphological element after a given left context could be added to this figure by assigning weights to boxes. The only known method of setting the value of these weights is based on statistics about occurrences of symbols or sequences in a sample of texts, a learning corpus.

Weighted automata are also used to compensate for the lack of accurate linguistic data. Weights are assigned to transitions in function of observable hints as to the occurrence of specific linguistic elements. During the analysis of a text, the weights are used to recognize those elements. For example, an initial uppercase letter is a hint of a proper name; the word ending *-ly* is a hint of an adverb like *shyly*. Weights are derived from statistics computed in a learning corpus. Results are inferior to those obtained with word lists of sufficient lexical coverage, e.g. lists of proper names or of adverbs: for instance, *bodily* ends in *-ly* but is usually an adjective. Word lists tend to be more and more used, but the two approaches are complementary, and the weighted-automaton method can make systems more robust when sufficiently extensive word lists are not available.

3.2. From words to sentences

3.2.1. Engineering approaches

The simplest model of the meaning of a text is the "word bag" model. Each word in the text represents an element of meaning, and the meaning of the text is represented by the set of the words that occur at least once in the text. The number of occurrences is usually attached to each word. The "word bag" model is used to perform tasks like content-based classification and indexation.

In order to implement the same tasks in a more elaborate way, or to implement other tasks, the sequential order of words must be taken into account. Translation is an example of an operation for which word order is obviously relevant: in many target languages, *The fly flies* and *The flies fly* should be translated differently. A model of text for which not only the value of words, but also their order, is relevant can be called a syntactic model. The formal and algorithmic tools involved in such a model depend entirely on the form of the linguistic data required. The most rational approach consists in constructing and using data similar to those mentioned in sections 3.1.4 to 3.1.9, but specifying ordered combinations of words. These data take the form of manually constructed lists or automata; some of them are automatically compiled into forms more adapted to computational operations. This approach is a long-term one. The stage of manual construction of linguistic data implies even more skill and effort than in the examples of section 3.1 (From letters to words), basically because there are many more words than letters. In addition, engineers feel uneasy with such data, that are largely outside their domain of competence; linguists feel uneasy with the necessary formal encoding; and little of the task can be automated. A consequence of this situation is a lack of linguistic resources that has been widely recognized, since 1990, as a major bottleneck in the development of language processing.

In order to avoid such work, alternative engineering techniques have been implemented and have had a dramatic development in recent years. The commonest of these techniques rely on weighted automata. (They are the most popular techniques based on weighted automata in language processing.) Weighted automata can be used to approximate various aspects of the grammar and syntax of languages: they can, for instance, guess at the part of speech of a word if the parts of speech of neighboring words are known. Weights are automatically derived from statistics about occurrences of symbols or sequences in a sample of texts, the learning corpus. The idea is similar to that with adverbs in *-ly* in section 3.1.10, but works even less well, for the same reason: there are more words than letters; there is a higher degree of complexity. As a matter of fact, in complex applications like translation and continuous speech recognition, results are still disappointing. Algorithms are well-known, but weights must be learnt for all words, and the only way of obtaining weights producing satisfactory results implies

- numerous occurrences of each word; therefore very large learning corpora (cf. section 3.1.1 about Zipf's law),
- statistics about sufficiently large contexts,
- sufficiently fine-grained tag sets.

The first constraint correctly predicts that if the learning corpus is too small, results are inadequate. When the size of the learning corpus increases, performances usually reach a maximum which is the best possible approximation in this framework. The last two constraints would lead to an explosion of the size of weighted automata and computational complexity. In practice, implementations of this method require considerable simplification of fundamental objects of the model: there is no serious attempt at processing compound words or ambiguity; the size of contexts is limited to two words to the left, and the size of tag sets to a few dozen tags, which is less than the tags et of Fig. 3.3. Finally, taking into consideration the third constraint would increase the cost of the manual tagging of the learning corpus, or require resorting to automatic tagging, with a corresponding output of inferior quality.

Resorting to such statistical approximations of grammar, syntax and the lexicon of languages is natural in so far as sufficiently accurate and comprehensive data seem out of reach. However, this is a short-term approach: it does not contribute to the enhancement of knowledge in these areas, and the technologies required for gathering exploitable and maintainable linguistic data have little in common with example-based learning. We can draw a parallel with meteorology: future weather depends on future physical data, or on physical data all around the world, including in marine areas where they are not measured with sufficient accuracy and frequency. Thus, weather is forecast on the basis of statistics about examples of past observations. However, designing weather forecast programs does not contribute to the advance of thermodynamics.

We will now turn to the linguistic approach. In order to relate formal notions with applications, we will refer primarily to translation, which is not a successfully automated operation yet, but which involves many of the basic operations in language processing.

3.2.2. Pattern definition and matching

Defining and matching patterns are two of these basic operations. In order to be able to translate a technical term like *microwave oven*, we must have a description of it, a method to locate occurrences in texts, and a link to a translation. The methods of description and location of such linguistic forms must take into account the existence of variants like the plural, *microwave ovens*, and possibly abbreviations like *MWO* if they are in use in relevant source texts. Thus, many linguistic forms are in fact sets of variants, and the actual form of all variants cannot always be computed from a canonical form. For example, the abbreviation *MWO* cannot be predicted from *microwave oven* by capitalizing initials, which would yield *MO*; the equivalence between *MWO* and the full form cannot be automatically inferred, even if the acronym occurs in a sample of source texts, because an explicit link between them, like *microwave oven (MWO)*, may be absent and, if present, would be ambiguous; etc. Thus the set of equivalent variants must often be manually constructed by linguists who are familiar with the field – a category of population which is often hard to find.

We can associate in a natural way *microwave oven* and its variants in the finite automaton of Fig. 3.25. When several lines are included in the same state, like *oven* and *ovens* here, they label parallel paths.

This type of automaton is more usual when there are more variants than with *microwave oven*. It is also used when the forms described are not equivalent, but constitute a small system which follows specific rules instead of general grammar rules of the language (Fig. 3.26). Such a system is called a local grammar.

In very restricted domains, the vocabulary and the syntactic constructions used in actual texts can be so stereotyped that all variability can be described in this form. This is the case of short stock exchange reports, weather forecast reports, sport scores etc. Local grammars can be used for translation, but this implies linking two monolingual local grammars together, one for the source language and another for the target language. Individual phrases of a grammar

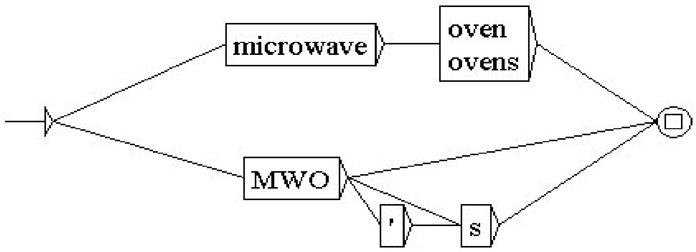


Figure 3.25. Definition of a simple linguistic pattern.

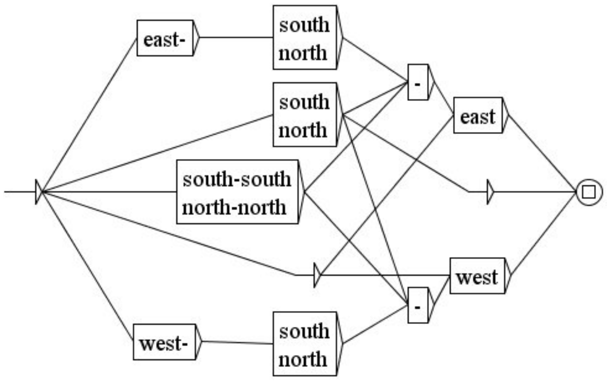


Figure 3.26. A local grammar.

must be specifically linked with phrases of the other, because they are not equivalent.

Finite automata defining linguistic patterns can be used to locate occurrences of the patterns in texts. When automata are as small as in the preceding instances, simple algorithms are sufficient: automata are compiled into the more traditional format with labelled edges and numbered states; they are determinized; they are matched against each point of the text.

A local grammar can be a representation of a subject of interest for a user in a text, for example one or several particular types of microwave ovens. In such a case, the local grammar can be used for text filtering, indexing and classification. Weights can be assigned to transitions in order to indicate the relevancy of paths with respect to the user's interest.

Comprehensive descriptions accounting for general language can reach impressive sizes. A complete grammar of dates, including informal dates, e.g. *before Christmas*, recognizes thousands of sequences. To be readable, such a description is necessarily organized into several automata. ¿From the formal point of view, the principle of such an organization is simple: a general finite automaton invokes sub-automata by special labels. Sub-automata, in turn, can equally invoke other sub-automata. Recursiveness may be allowed or not. In Fig. 3.27, the general automaton for numbers from 1 to 999 written in letters

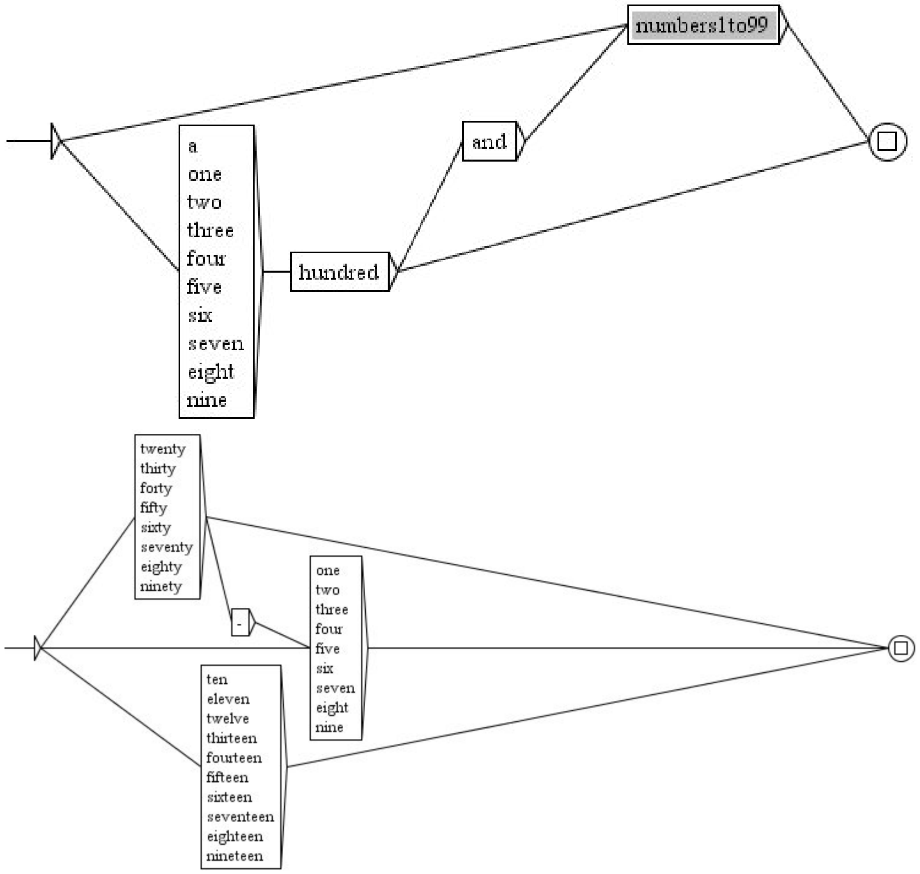


Figure 3.27. An automaton invokes another.

invokes the automaton for numbers from 1 to 99. The label for the second automaton is shown in grey. The use of labels for automata facilitates linguistic description for another reason: the same automaton can be invoked from several points and thus shared. Invoking an automaton via a label is thus equivalent to substituting it for the label. With patterns like terms, dates or numbers, invocations usually do not make up cycles: actual substitution is theoretically possible; it makes the set of automata equivalent to one finite automaton. However, with large grammars, actual substitution can lead to an explosion in size. For example, M. Gross's grammar of dates in French, which is organized into about 100 automata, becomes a 50-Mb automaton if sub-automata are systematically substituted. In the case of large grammars, the algorithms for locating occurrences in texts efficiently are therefore different: sub-automata are kept distinct and the matching algorithm is nondeterministic.

If cycles of invocations are allowed, the language recognized by the set of automata can be defined by reference to an equivalent context-free grammar (cf. section 1.6). The labels invoking sub-automata are the counterparts of variables, including the label of the general automaton which corresponds to the axiom of the grammar. Each of the automata is translated into a finite number of productions of the grammar. Such a set of automata is called a "recursive transition network" (RTN).

3.2.3. Parsing

If we consider more and more complex local grammars, we reach a point where the identification of a linguistic form depends on the identification of free constituents. Free constituents are syntactic constructs, like sentences or noun phrases, which involve open categories, like verbs or nouns, in their content. For example, recognizing the phrase *take into account* may imply identifying:

- its subject, which cannot be any noun, e.g. not *air*, and
- its free complement, which can occur before or after *into account*.

Both are free constituents. The subject is a noun phrase, which comprises at least an open category, a noun. The free complement can be a noun phrase or a sentential clause: *Max took into account that Mary was early.* The identification of these free and frozen constituents is required for complex applications like translation.

Several features of RTNs make them adequate for the formal description of such phrases.

- Free constituents can be represented by labels invoking other parts of the grammar. In the example of *take into account*, these labels will represent types of noun phrases, of sentences and of sentential clauses. Obviously, the labels are reusable from other points of the grammar, because other phrases or verbs will accept the same types of subjects or of complements.
- Small lexical variations and alternative constructions are described in parallel paths of the automata, as in Fig. 3.28.
- Recursiveness can be used for embeddings between syntactic constructs. In the example of Fig. 3.28, the phrase and the free constituents around it make up a sentence; the label S included in the automaton represents sentences. Thus, the rule is recursive.

A large variety of syntactic constructions in natural languages can be expressed in that way. A complete description of *take into account*, for example, should include passive, interrogative forms etc., and would be much larger than

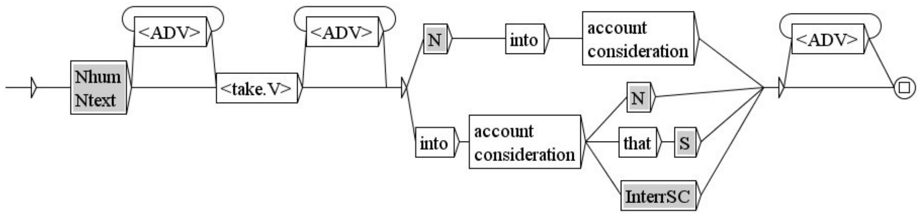


Figure 3.28. A sample of a grammar of *take into account*.

this figure. In addition, the number of grammatical constructions in a language is in some way multiplied by the size of the lexicon, since different words do not enter into the same grammatical constructions. However, the construction of large grammars for thousands of phrases and verbs can be partially automated. General grammars are manually constructed in the form of parameterized RTNs, then they are adapted to specific lexical items like *take into account* by setting the values of the parameters. These values are encoded for each lexical item in tables of syntactic properties. A large proportion of the parameters must be at the level of specific lexical items, and not of classes of items (e.g. transitive verbs), because syntactic properties are incredibly dependent on actual lexical items.

Here are two examples of open problems in the construction of grammars[6]: selectional constraints between predicates (i.e. verbs, nouns and adjectives) and their arguments (i.e. subject and essential complements):

*(Max + *The air) took into account that Mary was early*

and selectional constraints between predicates and adverbs:

*Max took the delay into account (last time + *by plane)*

Present grammars either overgenerate or undergenerate when such constraints come into play.

Even so, the construction of grammars of natural languages in the form of RTNs now appears to be within reach.

This situation provides partial answers for a classical controversy about the most popular two formal models of syntax: finite automata and context-free grammars. The issue of the adequacy of these two models dates back to the time of their actual definition and is still going on. Infrequent constructions have been used to argue that both were inadequate, but they can be conveniently dealt with as exceptions. ¿From 1960 to 1990, the folklore of the domain held that it was reasonable practice to use context-free grammars, and a heresy to use automata. Since then, investigation results suggested that the RTN model, which is equivalent to grammars but relies heavily on the automaton form, is

[6]In the next two examples, the star * marks that a sequence is not acceptable as a sentence.

convenient for the manual description of syntax as well as for automatic parsing. It is an open question as to whether the non-recursive counterpart of RTNs, which is equivalent to finite automata, will be better. Recursiveness can surely be eliminated from RTNs through an automatic compilation process, by substituting cycles for terminal embeddings and by limiting central embeddings to a fixed maximal depth. But even without recursiveness, RTN-based parsing is not necessarily more similar to automaton-based parsing than context-free parsing... In any case, the issue now appears less theoretical than computational.

3.2.4. Lexical ambiguity reduction

We mentioned lexical tagging in section 3.1.4. This operation consists of assigning tags to words. Word tags record linguistic information. Lexical tagging is not an application in itself, since word tags contain encoded information not directly exploitable by users. However, lexical tagging is required for enhancing the results of nearly all operations on texts: translation, spelling correction, location of index terms etc. Section 3.1.4 shows how dictionary lookup contributes to lexical tagging, but many words should be assigned distinct tags in relation to context, like *record*, a noun or a verb. Such forms are said to be lexically ambiguous. Syntactic parsing often resolves all lexical ambiguity. Sentences like the following are rare:

The newspapers found out some record

This ambiguous sentence has two syntactic analyses: *some record* is a noun phrase or a sentential clause, and *record* is accordingly a noun or a verb.

Syntactic parsing is not a mature technique yet, and there is a need for procedures that can work without complete syntactic grammars of languages, even if they resolve less lexical ambiguity than syntactic parsing.

Such a procedure can be designed on the following basis. After dictionary lookup, a text can be represented as an acyclic automaton of analyses like that of Fig. 3.29. Syntactic constraints can be represented as an automaton over the

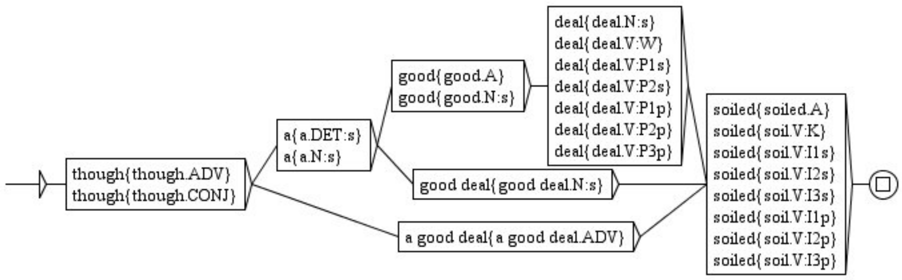


Figure 3.29. The automaton of analyses of *though a good deal soiled.*

same alphabet. Fig. 3.30 states that when the word *good* is a noun, it cannot follow the indefinite determiner *a*. The label @ stands for a default symbol:

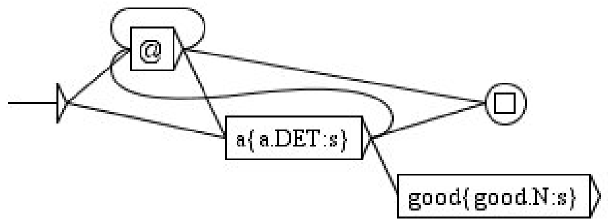


Figure 3.30. An automaton stating a syntactic constraint.

it matches the next input symbol if, at this point of the automaton, no other symbol matches. The intersection of the two automata is shown in Fig. 3.31; it

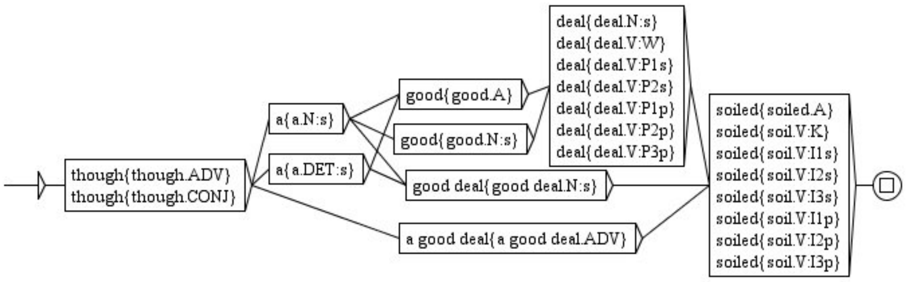


Figure 3.31. The intersection of the two automata.

represents those analyses of the text that obey the constraints. The intersection of two automata is an automaton that recognizes the intersection of the two languages recognized. It is constructed by a simple algorithm. Different syntactic constraints can be represented by different automata: since intersection is associative and commutative, the automata can be intersected in any order without changing the result. Thus, various syntactic constraints can be formalized independently and accumulated in order to reduce progressively more lexical ambiguity. However, this approach needs a convenient interface to allow linguists to express the constraints in the form of automata. Automata like that of Fig. 3.30 can be directly constructed only in very simple cases.

An alternative approach combines dictionary lookup and ambiguity resolution in another way. It considers that the relevant data are (i) the probability for a given word to occur with a given tag, and (ii) the probability of occurrence of a sequence of words (or tags). Such probabilities are estimated on the basis of statistics in a tagged corpus. The resulting values are inserted into a weighted automaton to make up a model of language. This technique has been applied to small tag sets, and the possibility of tagging compound words has not been seriously investigated.

Notes

The notion of formal model in linguistics emerged progressively. We will mention a few milestones on this path. During the first half of the twentieth century, Saussure stated clearly that language is a system and that form/meaning associations are arbitrary. This was a first step towards the separation between syntax and semantics. The translation of this idea into practice owes much to the study of native American languages by Sapir 1921. During the second half of the century, Harris incorporated the information aspect into the study of the forms of language. In particular, he introduced the notion of transformation (Harris 1952, Harris 1970). Gross 1975, Gross 1979 originated the construction of tables of syntactic properties. The parameterized graphs of section 3.2.3 are used in Senellart 1998 and Paumier 2001.

The theory of formal languages developed in parallel (Schützenberger and Chomsky 1963; Gross and Lentin 1967). Discussions arose during the same period of time about the adequacy of formal models for representing the behavior of speakers (Miller and Chomsky 1963) or the syntax of natural languages. Chomsky 1956, Chomsky 1957 mathematically "proved" that neither finite automata nor context-free languages were adequate for syntax, but he used infrequent constructions that can be conveniently dealt with as exceptions (Gross 1995). Gross gave an impulse to the actual production of extensive descriptions of lexicon and syntax with finite automata.

The observations that led to the statement of Zipf's law (Zipf 1935) were not restricted to language. The results exposed in section 3.1.3 about Zipf's law applied to written texts are based on Senellart 1999.

Johnson 1972 investigated various ways of combining formal rules and established whether the result of combination can be represented as a finite automaton. The notion of sequential transducer originates from Schützenberger 1977. Two algorithms of minimization of sequential transducers are known (Breslauer 1998; Béal and Carton 2001); the second one is based on successive contributions by Choffrut 1979, Reutenauer 1990 and Mohri 1994 (see also Chapter 1). The definition of p-sequential transducers was proposed by Mohri 1994. The algorithm of construction of generalized sequential transducers is adapted from Roche 1997.

The representation of finite automata as graphs with labels attached to states was introduced into language processing by Gross 1989 and Silberztein 1994 (`http://acl.ldc.upenn.edu/C/C94/C94-1095.pdf`). The Unitex system (`http://www-igm.univ-mlv.fr/~unitex`), implemented by Sébastien Paumier at the University of Marne-la-Vallée, is an open-source environment for language processing with automata and dictionaries.

The use of the intersection of finite transducers for specifying and implementing morphological analysis and generation, and for lexical ambiguity resolution, was first suggested by Koskenniemi 1983. Bimachines were introduced by Schützenberger 1961. The adaptation of bimachines to morphology and phonetics comes from Laporte 1997.

Weighted automata and transducers are defined by Paz 1971 and Eilenberg 1974. The FSM library (Mohri, Pereira, and Riley 2000) offers consistent tools related to weighted automata.

Algorithms for deriving weights from statistics about occurrences of symbols or sequences in a learning corpus are available in handbooks, e.g. Jurafsky and Martin 2000.

CHAPTER 4

Statistical Natural Language Processing

4.0. Introduction

The application of statistical methods to natural language processing has been remarkably successful over the past two decades. The wide availability of text and speech corpora has played a critical role in their success since, as for all learning techniques, these methods heavily rely on data. Many of the components of complex natural language processing systems, e.g., text normalizers, morphological or phonological analyzers, part-of-speech taggers, grammars or language models, pronunciation models, context-dependency models, acoustic Hidden-Markov Models (HMMs), are statistical models derived from large data sets using modern learning techniques. These models are often given as *weighted automata* or *weighted finite-state transducers* either directly or as a result of the approximation of more complex models.

Weighted automata and transducers are the finite automata and finite-state

Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Table 4.1. *Semiring examples.* $\oplus_{\log}$ *is defined by:* $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

transducers described in Chapter 1 Section 1.5 with the addition of some weight to each transition. Thus, weighted finite-state transducers are automata in which each transition, in addition to its usual input label, is augmented with an *output label* from a possibly different alphabet, and carries some *weight*. The weights may correspond to probabilities or log-likelihoods or they may be some other costs used to rank alternatives. More generally, as we shall see in the next section, they are elements of a *semiring* set. Transducers can be used to define a mapping between two different types of information sources, e.g., word and phoneme sequences. The weights are crucial to model the uncertainty of such mappings. Weighted transducers can be used for example to assign different pronunciations to the same word but with different ranks or probabilities.

Novel algorithms are needed to combine and optimize large statistical models represented as weighted automata or transducers. This chapter reviews several recent weighted transducer algorithms, including composition of weighted transducers, determinization of weighted automata and minimization of weighted automata, which play a crucial role in the construction of modern statistical natural language processing systems. It also outlines their use in the design of modern real-time speech recognition systems. It discusses and illustrates the representation by weighted automata and transducers of the components of these systems, and describes the use of these algorithms for combining, searching, and optimizing large component transducers of several million transitions for creating real-time speech recognition systems.

4.1. Preliminaries

This section introduces the definitions and notation used in the following.

A system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a *semiring* if $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$, $(\mathbb{K}, \otimes, \overline{1})$ is a monoid with identity element $\overline{1}$, $\otimes$ distributes over $\oplus$, and $\overline{0}$ is an annihilator for $\otimes$: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. Thus, a semiring is a ring that may lack negation. Table 4.1 lists some familiar semirings. In addition to the Boolean semiring, and the probability semiring used to combine probabilities, two semirings often used in text and speech processing applications are the *log semiring* which is isomorphic to the probability semiring via the negative-log morphism, and the *tropical semiring* which is derived from the log semiring using the *Viterbi approximation*. A *left semiring* is a system that verifies all the axioms of a semiring except from the right ditributivity. In the following definitions, $\mathbb{K}$ will be used to denote a left semiring or a semiring.

A semiring is said to be *commutative* when the multiplicative operation $\otimes$ is commutative. It is said to be *left divisible* if for any $x \neq \overline{0}$, there exists $y \in \mathbb{K}$ such that $y \otimes x = \overline{1}$, that is if all elements of $\mathbb{K}$ admit a left inverse. $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *weakly left divisible* if for any x and y in $\mathbb{K}$ such that $x \oplus y \neq \overline{0}$, there exists at least one z such that $x = (x \oplus y) \otimes z$. The $\otimes$-operation is *cancellative* if z is unique and we can write: $z = (x \oplus y)^{-1}x$. When z is not unique, we can still assume that we have an algorithm to find one of the possible z and call it $(x \oplus y)^{-1}x$. Furthermore, we will assume that z can be found in a consistent way, that is: $((u \otimes x) \oplus (u \otimes y))^{-1}(u \otimes x) = (x \oplus y)^{-1}x$ for any $x, y, u \in \mathbb{K}$ such that $u \neq \overline{0}$. A semiring is *zero-sum-free* if for any x and y in $\mathbb{K}$, $x \oplus y = \overline{0}$ implies $x = y = \overline{0}$.

A *weighted finite-state transducer* $\mathfrak{T}$ over a semiring $\mathbb{K}$ is an 8-tuple $\mathfrak{T} = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ where: $\mathcal{A}$ is the finite input alphabet of the transducer; $\mathcal{B}$ is the finite output alphabet; Q is a finite set of states; $I \subseteq Q$ the set of initial states; $F \subseteq Q$ the set of final states; $E \subseteq Q \times (\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{B} \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions; $\lambda : I \to \mathbb{K}$ the initial weight function; and $\rho : F \to \mathbb{K}$ the final weight function mapping F to $\mathbb{K}$. $E[q]$ denotes the set of transitions leaving a state $q \in Q$. $|\mathfrak{T}|$ denotes the sum of the number of states and transitions of $\mathfrak{T}$.

Weighted automata are defined in a similar way by simply omitting the input or output labels. Let $\Pi_1(\mathfrak{T})$ ($\Pi_2(\mathfrak{T})$) denote the weighted automaton obtained from a weighted transducer $\mathfrak{T}$ by omitting the input (resp. output) labels of $\mathfrak{T}$.

Given a transition $e \in E$, let $p[e]$ denote its origin or previous state, $n[e]$ its destination state or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. n, p, and w can be extended to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$ and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. More generally, w is extended to any finite set of paths R by setting: $w[R] = \bigoplus_{\pi \in R} w[\pi]$. Let $P(q, q')$ denote the set of paths from q to q' and $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$. These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, y, R') = \cup_{q \in R,\, q' \in R'} P(q, x, y, q')$. A transducer $\mathfrak{T}$ is *regulated* if the weight associated by $\mathfrak{T}$ to any pair of input-output string (x, y) given by:

$$[\![\mathfrak{T}]\!](x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]] \tag{4.1.1}$$

is well-defined and in $\mathbb{K}$. $[\![\mathfrak{T}]\!](x, y) = \overline{0}$ when $P(I, x, y, F) = \emptyset$. In particular, when it does not have any ε-cycle, $\mathfrak{T}$ is always regulated.

4.2. Algorithms

4.2.1. Composition

Composition is a fundamental algorithm used to create complex weighted transducers from simpler ones. It is a generalization of the composition algorithm presented in Chapter 1 Section 1.5 for unweighted finite-state transducers. Let $\mathbb{K}$ be a commutative semiring and let $\mathfrak{T}_1$ and $\mathfrak{T}_2$ be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet of $\mathfrak{T}_2$ coincides with the output alphabet of $\mathfrak{T}_1$. Assume that the infinite sum $\bigoplus_z \mathfrak{T}_1(x,z)\otimes\mathfrak{T}_2(z,y)$ is well-defined and in $\mathbb{K}$ for all $(x,y) \in \mathcal{A}^* \times \mathcal{C}^*$. This condition holds for all transducers defined over a *closed semiring* such as the Boolean semiring and the tropical semiring and for all acyclic transducers defined over an arbitrary semiring. Then, the result of the composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ is a weighted transducer denoted by $\mathfrak{T}_1 \circ \mathfrak{T}_2$ and defined for all x, y by:

$$[\![\mathfrak{T}_1 \circ \mathfrak{T}_2]\!](x,y) = \bigoplus_z \mathfrak{T}_1(x,z) \otimes \mathfrak{T}_2(z,y) \tag{4.2.1}$$

Note that we use a *matrix notation* for the definition of composition as opposed to a *functional notation*. There exists a general and efficient composition algorithm for weighted transducers. States in the composition $\mathfrak{T}_1 \circ \mathfrak{T}_2$ of two weighted transducers $\mathfrak{T}_1$ and $\mathfrak{T}_2$ are identified with pairs of a state of $\mathfrak{T}_1$ and a state of $\mathfrak{T}_2$. Leaving aside transitions with ε inputs or outputs, the following rule specifies how to compute a transition of $\mathfrak{T}_1 \circ \mathfrak{T}_2$ from appropriate transitions of $\mathfrak{T}_1$ and $\mathfrak{T}_2$:

$$(q_1, a, b, w_1, q_2) \text{ and } (q_1', b, c, w_2, q_2') \Longrightarrow ((q_1, q_1'), a, c, w_1 \otimes w_2, (q_2, q_2')) \tag{4.2.2}$$

The following is the pseudocode of the algorithm in the ε-free case.

```
WEIGHTED-COMPOSITION(𝔗1, 𝔗2)
Q ← I1 × I2
S ← I1 × I2
while S ≠ ∅ do
      (q1, q2) ← HEAD(S)
      DEQUEUE(S)
      if (q1, q2) ∈ I1 × I2 then
            I ← I ∪ {(q1, q2)}
            λ(q1, q2) ← λ1(q1) ⊗ λ2(q2)
      if (q1, q2) ∈ F1 × F2 then
            F ← F ∪ {(q1, q2)}
            ρ(q1, q2) ← ρ1(q1) ⊗ ρ2(q2)
      for each (e1, e2) ∈ E[q1] × E[q2] such that o[e1] = i[e2] do
            if (n[e1], n[e2]) ∉ Q then
                  Q ← Q ∪ {(n[e1], n[e2])}
                  ENQUEUE(S, (n[e1], n[e2]))
            E ← E ∪ {((q1, q2), i[e1], o[e2], w[e1] ⊗ w[e2], (n[e1], n[e2]))}
return 𝔗
```

The algorithm takes as input $\mathfrak{T}_1 = (\mathcal{A}, \mathcal{B}, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $\mathfrak{T}_2 = (\mathcal{B}, \mathcal{C}, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$, two weighted transducers, and outputs a weighted

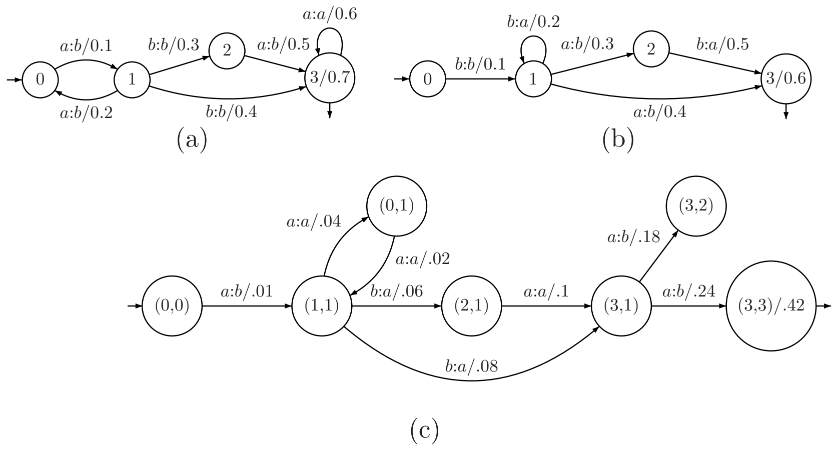


Figure 4.1. (a) Weighted transducer $\mathfrak{T}_1$ over the probabilityl semiring. (b) Weighted transducer $\mathfrak{T}_2$ over the probability semiring. (c) Composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$. Initial states are represented by an incoming arrow, final states with an outgoing arrow. Inside each circle, the first number indicates the state number, the second, at final states only, the value of the final weight function ρ at that state. Arrows represent transitions and are labeled with symbols followed by their corresponding weight.

transducer $\mathfrak{T} = (\mathcal{A}, \mathcal{C}, Q, I, F, E, \lambda, \rho)$ realizing the composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$. E, I, and F are all assumed to be initialized to the empty set.

The algorithm uses a queue S containing the set of pairs of states yet to be examined. The queue discipline of S can be arbitrarily chosen and does not affect the termination of the algorithm. The set of states Q is originally reduced to the set of pairs of the initial states of the original transducers and S is initialized to the same (lines 1-2). Each time through the loop of lines 3-16, a new pair of states (q_1, q_2) is extracted from S (lines 4-5). The initial weight of (q_1, q_2) is computed by $\otimes$-multiplying the initial weights of q_1 and q_2 when they are both initial states (lines 6-8). Similar steps are followed for final states (lines 9-11). Then, for each pair of matching transitions (e_1, e_2), a new transition is created according to the rules specified earlier (line 16). If the destination state $(n[e_1], n[e_2])$ has not been found before, it is added to Q and inserted in S (lines 14-15).

In the worst case, all transitions of $\mathfrak{T}_1$ leaving a state q_1 match all those of $\mathfrak{T}_2$ leaving state q_1', thus the space and time complexity of composition is quadratic: $O(|\mathfrak{T}_1||\mathfrak{T}_2|)$. However, a lazy implementation of composition can be used to construct just the part of the composed transducer that is needed. Figures 4.1(a)-(c) illustrate the algorithm when applied to the transducers of Figures 4.1(a)-(b) defined over the probability semiring.

More care is needed when $\mathfrak{T}_1$ admits output ε labels and $\mathfrak{T}_2$ input ε labels. Indeed, as illustrated by Figure 4.2, a straightforward generalization of the ε-

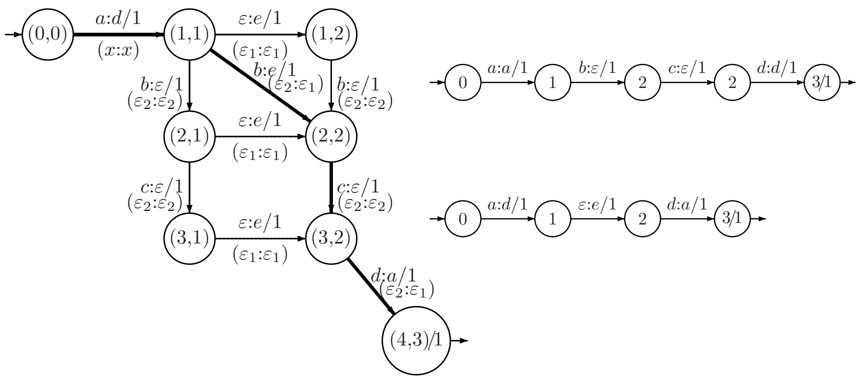


Figure 4.2. Redundant ε-paths. A straightforward generalization of the ε-free case could generate all the paths from $(1,1)$ to $(3,2)$ when composing the two simple transducers on the right-hand side.

free case would generate redundant ε-paths and, in the case of non-idempotent semirings, would lead to an incorrect result. The weight of the matching paths of the original transducers would be counted p times, where p is the number of redundant paths in the result of composition.

To cope with this problem, all but one ε-path must be filtered out of the composite transducer. Figure 4.2 indicates in boldface one possible choice for that path, which in this case is the shortest. Remarkably, that filtering mechanism can be encoded as a finite-state transducer.

Let $\tilde{\mathfrak{T}}_1$ ($\tilde{\mathfrak{T}}_2$) be the weighted transducer obtained from $\mathfrak{T}_1$ (resp. $\mathfrak{T}_2$) by replacing the output (resp. input) ε labels with ε_2 (resp. ε_1), and let $\mathfrak{F}$ be the filter finite-state transducer represented in Figure 4.3. Then $\tilde{\mathfrak{T}}_1 \circ \mathfrak{F} \circ \tilde{\mathfrak{T}}_2 = \mathfrak{T}_1 \circ \mathfrak{T}_2$. Since the two compositions in $\tilde{\mathfrak{T}}_1 \circ \mathfrak{F} \circ \tilde{\mathfrak{T}}_2$ do not involve ε's, the ε-free composition already described can be used to compute the resulting transducer.

Intersection (or Hadamard product) of weighted automata and composition of finite-state transducers are both special cases of composition of weighted transducers. Intersection corresponds to the case where input and output labels of transitions are identical and composition of unweighted transducers is obtained simply by omitting the weights.

In general, the definition of composition cannot be extended to the case of non-commutative semirings because the composite transduction cannot always be represented by a weighted finite-state transducer. Consider for example, the case of two transducers $\mathfrak{T}_1$ and $\mathfrak{T}_2$ accepting the same set of strings $(a, a)^*$, with $[\![\mathfrak{T}_1]\!](a, a) = x \in \mathbb{K}$ and $[\![\mathfrak{T}_2]\!](a, a) = y \in \mathbb{K}$ and let τ be the composite of the transductions corresponding to $\mathfrak{T}_1$ and $\mathfrak{T}_2$. Then, for any non-negative integer n, $\tau(a^n, a^n) = x^n \otimes y^n$ which in general is different from $(x \otimes y)^n$ if x and y

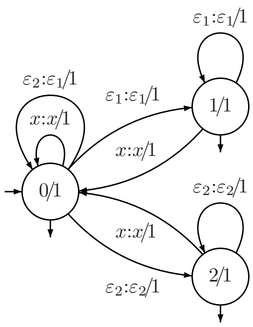


Figure 4.3. Filter for composition $\mathfrak{F}$.

do not commute. An argument similar to the classical Pumping lemma can then be used to show that τ cannot be represented by a weighted finite-state transducer.

When $\mathfrak{T}_1$ and $\mathfrak{T}_2$ are acyclic, composition can be extended to the case of non-commutative semirings. The algorithm would then consist of matching paths of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ directly rather than matching their constituent transitions. The termination of the algorithm is guaranteed by the fact that the number of paths of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ is finite. However, the time and space complexity of the algorithm is then exponential.

The weights of matching transitions and paths are $\otimes$-multiplied in composition. One might wonder if another useful operation, $\times$, can be used instead of $\otimes$, in particular when $\mathbb{K}$ is not commutative. The following proposition proves that that cannot be.

Proposition 4.2.1. *Let $(\mathbb{K}, \times, e)$ be a monoid. Assume that $\times$ is used instead of $\otimes$ in composition. Then, $\times$ coincides with $\otimes$ and $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a commutative semiring.*

Proof. Consider two sets of consecutive transitions of two paths: $\pi_1 = (p_1, a, a, x, q_1)(q_1, b, b, y, r_1)$ and $\pi_2 = (p_2, a, a, u, q_2)(q_2, b, b, v, r_2)$. Matching these transitions using $\times$ result in the following:

$$((p_1, p_2), a, a, x \times u, (q_1, q_2)) \quad \text{and} \quad ((q_1, q_2), b, b, y \times v, (r_1, r_2)) \tag{4.2.3}$$

Since the weight of the path obtained by matching π_1 and π_2 must also correspond to the $\times$-multiplication of the weight of π_1, $x \otimes y$, and the weight of π_2, $u \otimes v$, we have:

$$(x \times u) \otimes (y \times v) = (x \otimes y) \times (u \otimes v) \tag{4.2.4}$$

This identity must hold for all $x, y, u, v \in \mathbb{K}$. Setting $u = y = e$ and $v = \overline{1}$ leads to $x = x \otimes e$ and similarly $x = e \otimes x$ for all x. Since the identity element of $\otimes$ is unique, this proves that $e = \overline{1}$.

With $u = y = \overline{1}$, identity 4.2.4 can be rewritten as: $x \otimes v = x \times v$ for all x and v, which shows that $\times$ coincides with $\otimes$. Finally, setting $x = v = \overline{1}$ gives $u \otimes y = y \times u$ for all y and u which shows that $\otimes$ is commutative. ■

4.2.2. Determinization

This section describes a generic determinization algorithm for weighted automata. It is thus a generalization of the determinization algorithm for unweighted finite automata. When combined with the (unweighted) determinization for finite-state transducers presented in Chapter 1 Section 1.5, it leads to an algorithm for determinizing weighted transducers.[1]

A weighted automaton is said to be *deterministic* or *subsequential* if it has a unique initial state and if no two transitions leaving any state share the same input label. There exists a natural extension of the classical subset construction to the case of weighted automata over a weakly left divisible left semiring called *determinization*.[2] The algorithm is generic: it works with any weakly left divisible semiring. The pseudocode of the algorithm is given below with Q', I', F', and E' all initialized to the empty set.

Weighted-Determinization($\mathfrak{A}$)
$i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
$\lambda'(i') \leftarrow \overline{1}$
$S \leftarrow \{i'\}$
while $S \neq \emptyset$ **do**
    $p' \leftarrow$ Head(S)
    Dequeue(S)
    for each $x \in i[E[Q[p']]]$ **do**
        $w' \leftarrow \bigoplus \{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
        $q' \leftarrow \{(q, \bigoplus \{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) :$
            $q = n[e], i[e] = x, e \in E[Q[p']]\}$
        $E' \leftarrow E' \cup \{(p', x, w', q')\}$
        if $q' \notin Q'$ **then**
            $Q' \leftarrow Q' \cup \{q'\}$
            if $Q[q'] \cap F \neq \emptyset$ **then**
                $F' \leftarrow F' \cup \{q'\}$
                $\rho'(q') \leftarrow \bigoplus \{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
            Enqueue(S, q')
return $\mathfrak{T}'$

[1] In reality, the determinization of unweighted and that of weighted finite-state transducers can both be viewed as special instances of the generic algorithm presented here but, for clarity purposes, we will not emphasize that view in what follows.

[2] We assume that the weighted automata considered are all such that for any string $x \in \mathcal{A}^*$, $w[P(I, x, Q)] \neq \overline{0}$. This condition is always satisfied with trim machines over the tropical semiring or any zero-sum-free semiring.

A *weighted subset* p' of Q is a set of pairs $(q, x) \in Q \times \mathbb{K}$. $Q[p']$ denotes the set of states q of the weighted subset p'. $E[Q[p']]$ represents the set of transitions leaving these states, and $i[E[Q[p']]]$ the set of input labels of these transitions.

The states of the output automaton can be identified with (weighted) subsets of the states of the original automaton. A state r of the output automaton that can be reached from the start state by a path π is identified with the set of pairs $(q, x) \in Q \times \mathbb{K}$ such that q can be reached from an initial state of the original machine by a path σ with $i[\sigma] = i[\pi]$ and $\lambda[p[\sigma]] \otimes w[\sigma] = \lambda[p[\pi]] \otimes w[\pi] \otimes x$. Thus, x can be viewed as the *residual* weight at state q. When it terminates, the algorithm takes as input a weighted automaton $\mathfrak{A} = (\mathcal{A}, Q, I, F, E, \lambda, \rho)$ and yields an equivalent subsequential weighted automaton $\mathfrak{A}' = (\mathcal{A}, Q', I', F', E', \lambda', \rho')$.

The algorithm uses a queue S containing the set of states of the resulting automaton $\mathfrak{A}'$, yet to be examined. The queue discipline of S can be arbitrarily chosen and does not affect the termination of the algorithm. $\mathfrak{A}'$ admits a unique initial state, i', defined as the set of initial states of $\mathfrak{A}$ augmented with their respective initial weights. Its input weight is $\overline{1}$ (lines 1-2). S originally contains only the subset i' (line 3). Each time through the loop of lines 4-16, a new subset p' is extracted from S (lines 5-6). For each x labeling at least one of the transitions leaving a state p of the subset p', a new transition with input label x is constructed. The weight w' associated to that transition is the sum of the weights of all transitions in $E[Q[p']]$ labeled with x pre-$\otimes$-multiplied by the residual weight v at each state p (line 8). The destination state of the transition is the subset containing all the states q reached by transitions in $E[Q[p']]$ labeled with x. The weight of each state q of the subset is obtained by taking the $\oplus$-sum of the residual weights of the states p $\otimes$-times the weight of the transition from p leading to q and by *dividing* that by w'. The new subset q' is inserted in the queue S when it is a new state (line 15). If any of the states in the subset q' is final, q' is made a final state and its final weight is obtained by summing the final weights of all the final states in q', pre-$\otimes$-multiplied by their residual weight v (line 14).

Figures 4.4(a)-(b) illustrate the determinization of a weighted automaton over the tropical semiring. The worst case complexity of determinization is exponential even in the unweighted case. However, in many practical cases such as for weighted automata used in large-vocabulary speech recognition, this blow-up does not occur. It is also important to notice that just like composition, determinization admits a natural lazy implementation which can be useful for saving space.

Unlike the unweighted case, determinization does not halt on all input weighted automata. In fact, some weighted automata, non *subsequentiable* automata, do not even admit equivalent subsequential machines. But even for some subsequentiable automata, the algorithm does not halt. We say that a weighted automaton $\mathfrak{A}$ is *determinizable* if the determinization algorithm halts for the input $\mathfrak{A}$. With a determinizable input, the algorithm outputs an equivalent subsequential weighted automaton.

There exists a general *twins property* for weighted automata that provides a

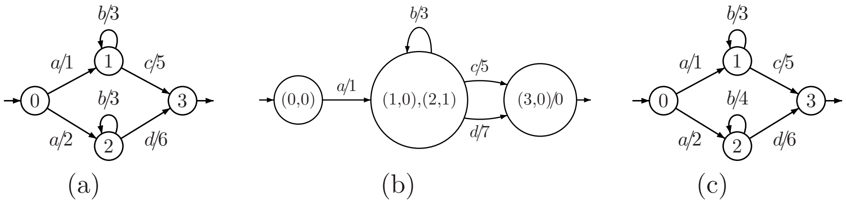


Figure 4.4. Determinization of weighted automata. (a) Weighted automaton over the tropical semiring $\mathfrak{A}$. (b) Equivalent weighted automaton $\mathfrak{B}$ obtained by determinization of $\mathfrak{A}$. (c) Non-determinizable weighted automaton over the tropical semiring, states 1 and 2 are non-twin siblings.

characterization of determinizable weighted automata under some general conditions. Let $\mathfrak{A}$ be a weighted automaton over a weakly left divisible left semiring $\mathbb{K}$. Two states q and q' of $\mathfrak{A}$ are said to be *siblings* if there exist two strings x and y in $\mathcal{A}^*$ such that both q and q' can be reached from I by paths labeled with x and there is a cycle at q and a cycle at q' both labeled with y. When $\mathbb{K}$ is a commutative and cancellative semiring, two sibling states are said to be *twins* iff for any string y:

$$w[P(q, y, q)] = w[P(q', y, q')] \tag{4.2.5}$$

$\mathfrak{A}$ has *the twins property* if any two sibling states of $\mathfrak{A}$ are twins. Figure 4.4(c) shows an unambiguous weighted automaton over the tropical semiring that does not have the twins property: states 1 and 2 can be reached by paths labeled with a from the initial state and admit cycles with the same label b, but the weights of these cycles (3 and 4) are different.

Theorem 4.2.2. *Let $\mathfrak{A}$ be a weighted automaton over the tropical semiring. If $\mathfrak{A}$ has the twins property, then $\mathfrak{A}$ is determinizable.*

With trim unambiguous weighted automata, the condition is also necessary.

Theorem 4.2.3. *Let $\mathfrak{A}$ be a trim unambiguous weighted automaton over the tropical semiring. Then the three following properties are equivalent:*
1. *$\mathfrak{A}$ is determinizable.*
2. *$\mathfrak{A}$ has the twins property.*
3. *$\mathfrak{A}$ is subsequentiable.*

There exists an efficient algorithm for testing the twins property for weighted automata, which cannot be presented briefly in this chapter. Note that any acyclic weighted automaton over a zero-sum-free semiring has the twins property and is determinizable.

4.2.3. Weight pushing

The choice of the distribution of the total weight along each successful path of a weighted automaton does not affect the definition of the function realized by that automaton, but this may have a critical impact on the efficiency in many applications, e.g., natural language processing applications, when a heuristic pruning is used to visit only a subpart of the automaton. There exists an algorithm, *weight pushing*, for normalizing the distribution of the weights along the paths of a weighted automaton or more generally a weighted directed graph. The transducer normalization algorithm presented in Chapter 1 Section 1.5 can be viewed as a special instance of this algorithm.

Let $\mathfrak{A}$ be a weighted automaton over a semiring $\mathbb{K}$. Assume that $\mathbb{K}$ is zero-sum-free and weakly left divisible. For any state $q \in Q$, assume that the following sum is well-defined and in $\mathbb{K}$:

$$d[q] = \bigoplus_{\pi \in P(q,F)} (w[\pi] \otimes \rho[n[\pi]]) \tag{4.2.6}$$

$d[q]$ is the *shortest-distance* from q to F. $d[q]$ is well-defined for all $q \in Q$ when $\mathbb{K}$ is a k-closed semiring. The weight pushing algorithm consists of computing each shortest-distance $d[q]$ and of *reweighting* the transition weights, initial weights and final weights in the following way:

$$\forall e \in E \text{ s.t. } d[p[e]] \neq \overline{0}, w[e] \leftarrow d[p[e]]^{-1} \otimes w[e] \otimes d[n[e]] \tag{4.2.7}$$

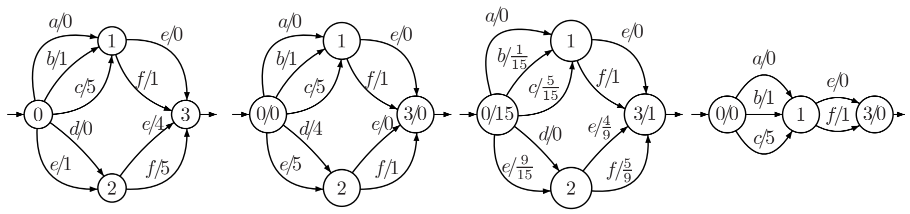
$$\forall q \in I, \lambda[q] \leftarrow \lambda[q] \otimes d[q] \tag{4.2.8}$$

$$\forall q \in F, \text{ s.t. } d[q] \neq \overline{0}, \rho[q] \leftarrow d[q]^{-1} \otimes \rho[q] \tag{4.2.9}$$

Each of these operations can be assumed to be done in constant time, thus reweighting can be done in linear time $O(\mathfrak{T}_\otimes|\mathfrak{A}|)$ where $\mathfrak{T}_\otimes$ denotes the worst cost of an $\otimes$-operation. The complexity of the computation of the shortest-distances depends on the semiring. In the case of k-closed semirings such as the tropical semiring, $d[q]$, $q \in Q$, can be computed using a generic shortest-distance algorithm. The complexity of the algorithm is linear in the case of an acyclic automaton: $O(\mathrm{Card}(Q)+(\mathfrak{T}_\oplus+\mathfrak{T}_\otimes)\,\mathrm{Card}(E))$, where $\mathfrak{T}_\oplus$ denotes the worst cost of an $\oplus$-operation. In the case of a general weighted automaton over the tropical semiring, the complexity of the algorithm is $O(\mathrm{Card}(E)+\mathrm{Card}(Q)\log\mathrm{Card}(Q))$.

In the case of closed semirings such as $(\mathbb{R}_+, +, \times, 0, 1)$, a generalization of the Floyd-Warshall algorithm for computing all-pairs shortest-distances can be used. The complexity of the algorithm is $\Theta(\mathrm{Card}(Q)^3(\mathfrak{T}_\oplus+\mathfrak{T}_\otimes+\mathfrak{T}_*))$ where $\mathfrak{T}_*$ denotes the worst cost of the closure operation. The space complexity of these algorithms is $\Theta(\mathrm{Card}(Q)^2)$. These complexities make it impractical to use the Floyd-Warshall algorithm for computing $d[q]$, $q \in Q$, for relatively large graphs or automata of several hundred million states or transitions. An approximate version of a generic shortest-distance algorithm can be used instead to compute $d[q]$ efficiently.

Roughly speaking, the algorithm *pushes the weights* of each path as much as possible towards the initial states. Figures 4.5(a)-(c) illustrate the application of the algorithm in a special case both for the tropical and probability semirings.



(a) (b) (c) (d)

Figure 4.5. Weight pushing algorithm. (a) Weighted automaton $\mathfrak{A}$. (b) Equivalent weighted automaton $\mathfrak{B}$ obtained by weight pushing in the tropical semiring. (c) Weighted automaton $\mathfrak{C}$ obtained from $\mathfrak{A}$ by weight pushing in the probability semiring. (d) Minimal weighted automaton over the tropical semiring equivalent to $\mathfrak{A}$.

Note that if $d[q] = \overline{0}$, then, since $\mathbb{K}$ is zero-sum-free, the weight of all paths from q to F is $\overline{0}$. Let $\mathfrak{A}$ be a weighted automaton over the semiring $\mathbb{K}$. Assume that $\mathbb{K}$ is closed or k-closed and that the shortest-distances $d[q]$ are all well-defined and in $\mathbb{K}-\{\overline{0}\}$. Note that in both cases we can use the distributivity over the infinite sums defining shortest distances. Let e' (π') denote the transition e (path π) after application of the weight pushing algorithm. e' (π') differs from e (resp. π) only by its weight. Let λ' denote the new initial weight function, and ρ' the new final weight function.

Proposition 4.2.4. *Let $\mathfrak{B} = (\mathcal{A}, Q, I, F, E', \lambda', \rho')$ be the result of the weight pushing algorithm applied to the weighted automaton $\mathfrak{A}$, then*

1. *the weight of a successful path π is unchanged after application of weight pushing:*

$$\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] = \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]] \tag{4.2.10}$$

2. *the weighted automaton $\mathfrak{B}$ is* stochastic, *i.e.*

$$\forall q \in Q, \bigoplus_{e' \in E'[q]} w[e'] = \overline{1} \tag{4.2.11}$$

Proof. Let $\pi' = e'_1 \ldots e'_k$. By definition of λ' and ρ',

$$\begin{aligned}\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] &= \lambda[p[e_1]] \otimes d[p[e_1]] \otimes d[p[e_1]]^{-1} \otimes w[e_1] \otimes d[n[e_1]] \otimes \cdots \\ &\quad \otimes d[p[e_k]]^{-1} \otimes w[e_k] \otimes d[n[e_k]] \otimes d[n[e_k]]^{-1} \otimes \rho[n[\pi]] \\ &= \lambda[p[\pi]] \otimes w[e_1] \otimes \cdots \otimes w[e_k] \otimes \rho[n[\pi]]\end{aligned}$$

which proves the first statement of the proposition. Let $q \in Q$,

$$\begin{aligned}\bigoplus_{e' \in E'[q]} w[e'] &= \bigoplus_{e \in E[q]} d[q]^{-1} \otimes w[e] \otimes d[n[e]] \\ &= d[q]^{-1} \otimes \bigoplus_{e \in E[q]} w[e] \otimes d[n[e]]\end{aligned}$$

$$
\begin{aligned}
&= d[q]^{-1} \otimes \bigoplus_{e \in E[q]} w[e] \otimes \bigoplus_{\pi \in P(n[e],F)} (w[\pi] \otimes \rho[n[\pi]]) \\
&= d[q]^{-1} \otimes \bigoplus_{e \in E[q], \pi \in P(n[e],F)} (w[e] \otimes w[\pi] \otimes \rho[n[\pi]]) \\
&= d[q]^{-1} \otimes d[q] = \overline{1}
\end{aligned}
$$

where we used the distributivity of the multiplicative operation over infinite sums in closed or k-closed semirings. This proves the second statement of the proposition. ■

These two properties of weight pushing are illustrated by Figures 4.5(a)-(c): the total weight of a successful path is unchanged after pushing; at each state of the weighted automaton of Figure 4.5(b), the minimum weight of the outgoing transitions is 0, and at at each state of the weighted automaton of Figure 4.5(c), the weights of outgoing transitions sum to 1. Weight pushing can also be used to test the equivalence of two weighted automata.

4.2.4. Minimization

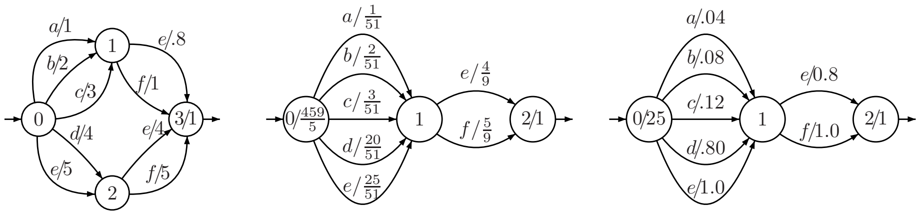
A deterministic weighted automaton is said to be *minimal* if there exists no other deterministic weighted automaton with a smaller number of states and realizing the same function. Two states of a deterministic weighted automaton are said to be *equivalent* if exactly the same set of strings with the same weights label paths from these states to a final state, the final weights being included. Thus, two equivalent states of a deterministic weighted automaton can be merged without affecting the function realized by that automaton. A weighted automaton is minimal when it admits no two distinct equivalent states after any redistribution of the weights along its paths.

There exists a general algorithm for computing a minimal deterministic automaton equivalent to a given weighted automaton. It is thus a generalization of the minimization algorithms for unweighted finite automata. It can be combined with the minimization algorithm for unweighted finite-state transducers presented in Chapter 1 Section 1.5 to minimize weighted finite-state transducers.[3] It consists of first applying the weight pushing algorithm to normalize the distribution of the weights along the paths of the input automaton, and then of treating each pair (label, weight) as a single label and applying the classical (unweighted) automata minimization.

THEOREM 4.2.5. *Let $\mathfrak{A}$ be a deterministic weighted automaton over a semiring $\mathbb{K}$. Assume that the conditions of application of the weight pushing algorithm hold, then the execution of the following steps:*

1. *weight pushing,*
2. *(unweighted) automata minimization,*

[3]In reality, the minimization of both unweighted and weighted finite-state transducers can be viewed as special instances of the algorithm presented here, but, for clarity purposes, we will not emphasize that view in what follows.



(a) (b) (c)

Figure 4.6. Minimization of weighted automata. (a) Weighted automaton $\mathfrak{A}'$ over the probability semiring. (b) Minimal weighted automaton $\mathfrak{B}'$ equivalent to $\mathfrak{A}'$. (c) Minimal weighted automaton $\mathfrak{C}'$ equivalent to $\mathfrak{A}'$.

lead to a minimal weighted automaton equivalent to $\mathfrak{A}$.

The complexity of automata minimization is linear in the case of acyclic automata $O(\text{Card}(Q) + \text{Card}(E))$ and in $O(\text{Card}(E) \log \text{Card}(Q))$ in the general case. Thus, in view of the complexity results given in the previous section, in the case of the tropical semiring, the total complexity of the weighted minimization algorithm is linear in the acyclic case $O(\text{Card}(Q) + \text{Card}(E))$ and in $O(\text{Card}(E) \log \text{Card}(Q))$ in the general case.

Figures 4.5(a), 4.5(b), and 4.5(d) illustrate the application of the algorithm in the tropical semiring. The automaton of Figure 4.5(a) cannot be further minimized using the classical unweighted automata minimization since no two states are equivalent in that machine. After weight pushing, the automaton (Figure 4.5(b)) has two states (1 and 2) that can be merged by the classical unweighted automata minimization.

Figures 4.6(a)-(c) illustrate the minimization of an automaton defined over the probability semiring. Unlike the unweighted case, a minimal weighted automaton is not unique, but all minimal weighted automata have the same graph topology, they only differ by the way the weights are distributed along each path. The weighted automata $\mathfrak{B}'$ and $\mathfrak{C}'$ are both minimal and equivalent to $\mathfrak{A}'$. $\mathfrak{B}'$ is obtained from $\mathfrak{A}'$ using the algorithm described above in the probability semiring and it is thus a stochastic weighted automaton in the probability semiring.

For a deterministic weighted automaton, the first operation of the semiring can be arbitrarily chosen without affecting the definition of the function it realizes. This is because, by definition, a deterministic weighted automaton admits at most one path labeled with any given string. Thus, in the algorithm described in theorem 4.2.5, the weight pushing step can be executed in any semiring $\mathbb{K}'$ whose multiplicative operation matches that of $\mathbb{K}$. The minimal weighted automata obtained by pushing the weights in $\mathbb{K}'$ is also minimal in $\mathbb{K}$ since it can be interpreted as a (deterministic) weighted automaton over $\mathbb{K}$.

In particular, $\mathfrak{A}'$ can be interpreted as a weighted automaton over the semiring $(\mathbb{R}_+, \max, \times, 0, 1)$. The application of the weighted minimization algorithm to $\mathfrak{A}'$ in this semiring leads to the minimal weighted automaton $\mathfrak{C}'$ of Figure 4.6(c). $\mathfrak{C}'$ is also a *stochastic* weighted automaton in the sense that, at any state, the maximum weight of all outgoing transitions is one.

This fact leads to several interesting observations. One is related to the complexity of the algorithms. Indeed, we can choose a semiring $\mathbb{K}'$ in which the complexity of weight pushing is better than in $\mathbb{K}$. The resulting automaton is still minimal in $\mathbb{K}$ and has the additional property of being stochastic in $\mathbb{K}'$. It only differs from the weighted automaton obtained by pushing weights in $\mathbb{K}$ in the way weights are distributed along the paths. They can be obtained from each other by application of weight pushing in the appropriate semiring. In the particular case of a weighted automaton over the probability semiring, it may be preferable to use weight pushing in the $(\max, \times)$-semiring since the complexity of the algorithm is then equivalent to that of classical single-source shortest-paths algorithms. The corresponding algorithm is a special instance of the generic shortest-distance algorithm.

Another important point is that the weight pushing algorithm may not be defined in $\mathbb{K}$ because the machine is not zero-sum-free or for other reasons. But an alternative semiring $\mathbb{K}'$ can sometimes be used to minimize the input weighted automaton.

The results just presented were all related to the minimization of the number of states of a deterministic weighted automaton. The following proposition shows that minimizing the number of states coincides with minimizing the number of transitions.

Proposition 4.2.6. *Let $\mathfrak{A}$ be a minimal deterministic weighted automaton, then $\mathfrak{A}$ has the minimal number of transitions.*

Proof. Let $\mathfrak{A}$ be a deterministic weighted automaton with the minimal number of transitions. If two distinct states of $\mathfrak{A}$ were equivalent, they could be merged, thereby strictly reducing the number of its transitions. Thus, $\mathfrak{A}$ must be a minimal deterministic automaton. Since, minimal deterministic automata have the same topology, in particular the same number of states and transitions, this proves the proposition.

4.3. Application to speech recognition

Much of the statistical techniques now widely used in natural language processing were inspired by early work in speech recognition. This section discusses the representation of the component models of an automatic speech recognition system by weighted transducers and describes how they can be combined, searched, and optimized using the algorithms described in the previous sections. The methods described can be used similarly in many other areas of natural language processing.

4.3.1. Statistical formulation

Speech recognition consists of generating accurate written transcriptions for spoken utterances. The desired transcription is typically a sequence of words, but it may also be the utterance's phonemic or syllabic transcription or a transcription into any other sequence of written units.

The problem can be formulated as a maximum-likelihood decoding problem, or the so-called *noisy channel* problem. Given a speech utterance, speech recognition consists of determining its most likely written transcription. Thus, if we let o denote the observation sequence produced by a signal processing system, w a (word) transcription sequence over an alphabet $\mathcal{A}$, and $\mathbf{P}(w \mid o)$ the probability of the transduction of o into w, the problem consists of finding $\hat{w}$ as defined by:

$$\hat{w} = \operatorname*{argmax}_{w \in \mathcal{A}^*} \mathbf{P}(w \mid o) \tag{4.3.1}$$

Using Bayes' rule, $\mathbf{P}(w \mid o)$ can be rewritten as: $\frac{\mathbf{P}(o|w)\mathbf{P}(w)}{\mathbf{P}(o)}$. Since $\mathbf{P}(o)$ does not depend on w, the problem can be reformulated as:

$$\hat{w} = \operatorname*{argmax}_{w \in \mathcal{A}^*} \mathbf{P}(o \mid w)\,\mathbf{P}(w) \tag{4.3.2}$$

where $\mathbf{P}(w)$ is the a priori probability of the written sequence w in the language considered and $\mathbf{P}(o \mid w)$ the probability of observing o given that the sequence w has been uttered. The probabilistic model used to estimate $\mathbf{P}(w)$ is called a *language model* or a *statistical grammar*. The generative model associated to $\mathbf{P}(o \mid w)$ is a combination of several knowledge sources, in particular the *acoustic model*, and the *pronunciation model*. $\mathbf{P}(o \mid w)$ can be decomposed into several intermediate levels e.g., that of phones, syllables, or other units. In most large-vocabulary speech recognition systems, it is decomposed into the following probabilistic models that are assumed independent:

- $\mathbf{P}(p \mid w)$, a pronunciation model or lexicon transducing word sequences w to phonemic sequences p;
- $\mathbf{P}(c \mid p)$, a context-dependency transduction mapping phonemic sequences p to context-dependent phone sequences c;
- $\mathbf{P}(d \mid c)$, a context-dependent phone model mapping sequences of context-dependent phones c to sequences of distributions d; and
- $\mathbf{P}(o \mid d)$, an acoustic model applying distribution sequences d to observation sequences.[4]

Since the models are assumed to be independent,

$$\mathbf{P}(o \mid w) = \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w) \tag{4.3.3}$$

[4] $\mathbf{P}(o \mid d)\mathbf{P}(d \mid c)$ or $\mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)$ is often called an *acoustic model*.

Equation 4.3.2 can thus be rewritten as:

$$\hat{w} = \operatorname*{argmax}_{w} \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.4}$$

The following sections discuss the definition and representation of each of these models and that of the observation sequences in more detail. The transduction models are typically given either directly or as a result of an approximation as weighted finite-state transducers. Similarly, the language model is represented by a weighted automaton.

4.3.2. Statistical grammar

In some relatively restricted tasks, the language model for $\mathbf{P}(w)$ is based on an unweighted rule-based grammar. But, in most large-vocabulary tasks, the model is a weighted grammar derived from large corpora of several million words using statistical methods. The purpose of the model is to assign a probability to each sequence of words, thereby assigning a ranking to all sequences. Thus, the parsing information it may supply is not directly relevant to the statistical formulation described in the previous section.

The probabilistic model derived from corpora may be a probabilistic context-free grammmar. But, in general, context-free grammars are computationally too demanding for real-time speech recognition systems. The amount of work required to expand a recognition hypothesis can be unbounded for an unrestricted grammar. Instead, a regular approximation of a probabilistic context-free grammar is used. In most large-vocabulary speech recognition systems, the probabilistic model is in fact directly constructed as a weighted regular grammar and represents an n-gram model. Thus, this section concentrates on a brief description of these models.[5]

Regardless of the structure of the model, using the Bayes's rule, the probability of the word sequence $w = w_1 \cdots w_k$ can be written as the following product of conditional probabilities:

$$\mathbf{P}(w) = \prod_{i=1}^{k} \mathbf{P}(w_i \mid w_1 \cdots w_{i-1}) \tag{4.3.5}$$

An n-gram model is based on the Markovian assumption that the probability of the occurrence of a word only depends on the $n-1$ preceding words, that is, for $i = 1 \ldots n$:

$$\mathbf{P}(w_i \mid w_1 \cdots w_{i-1}) = \mathbf{P}(w_i \mid h_i) \tag{4.3.6}$$

where the conditioning history h_i has length at most $n-1$: $|h_i| \leq n-1$. Thus,

$$\mathbf{P}(w) = \prod_{i=1}^{k} \mathbf{P}(w_i \mid h_i) \tag{4.3.7}$$

[5]Similar probabilistic models are designed for biological sequences (see Chapter 6).

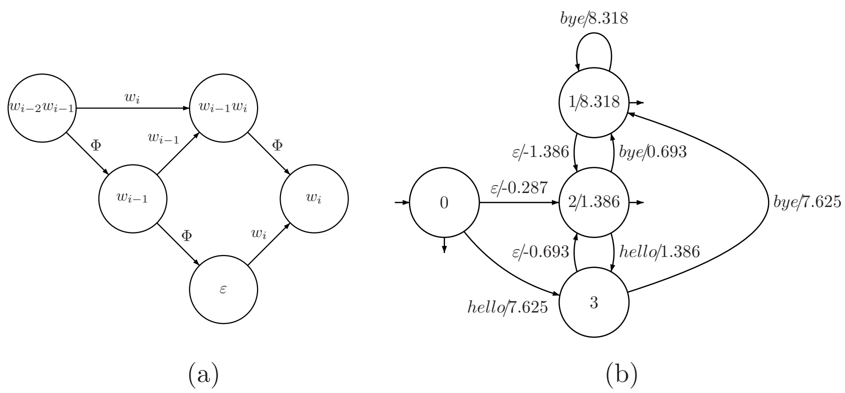


Figure 4.7. Katz back-off n-gram model. (a) Representation of a trigram model with failure transitions labeled with Φ. (b) Bigram model derived from the input text *hello bye bye*. The automaton is defined over the log semiring (the transition weights are negative log-probabilities). State 0 is the initial state. State 1 corresponds to the word *bye* and state 3 to the word *hello*. State 2 is the back-off state.

Let $c(w)$ denote the number of occurrences of a sequence w in the corpus. $c(h_i)$ and $c(h_i w_i)$ can be used to estimate the conditional probability $\mathbf{P}(w_i \mid h_i)$. When $c(h_i) \neq 0$, the maximum likelihood estimate of $\mathbf{P}(w_i \mid h_i)$ is:

$$\hat{\mathbf{P}}(w_i \mid h_i) = \frac{c(h_i w_i)}{c(h_i)} \tag{4.3.8}$$

But, a classical data sparsity problem arises in the design of all n-gram models: the corpus, no matter how large, may contain no occurrence of h_i ($c(h_i) = 0$). A solution to this problem is based on *smoothing techniques*. This consists of adjusting $\hat{\mathbf{P}}$ to reserve some probability mass for unseen n-gram sequences.

Let $\tilde{\mathbf{P}}(w_i \mid h_i)$ denote the adjusted conditional probability. A smoothing technique widely used in language modeling is the Katz back-off technique. The idea is to "back-off" to lower order n-gram sequences when $c(h_i w_i) = 0$. Define the backoff sequence of h_i as the lower order n-gram sequence suffix of h_i and denote it by h'_i. $h_i = u h'_i$ for some word u. Then, in a Katz back-off model, $\mathbf{P}(w_i \mid h_i)$ is defined as follows:

$$\mathbf{P}(w_i \mid h_i) = \begin{cases} \tilde{\mathbf{P}}(w_i \mid h_i) & \text{if } c(h_i w_i) > 0 \\ \alpha_{h_i} \mathbf{P}(w_i \mid h'_i) & \text{otherwise} \end{cases} \tag{4.3.9}$$

where α_{h_i} is a factor ensuring normalization. The Katz back-off model admits a natural representation by a weighted automaton in which each state encodes a

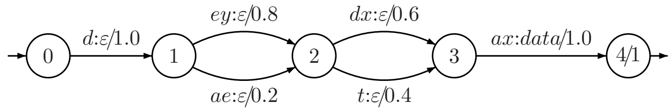


Figure 4.8. Section of a pronunciation model of English, a weighted transducer over the probability semiring giving a compact representation of four pronunciations of the word *data* due to two distinct pronunciations of the first vowel a and two pronunciations of the consonant t (flapped or not).

conditioning history of length less than n. As in the classical de Bruijn graphs, there is a transition labeled with w_i from the state encoding h_i to the state encoding $h'_i w_i$ when $c(h_i w_i) \neq 0$. A so-called *failure transition* can be used to capture the semantic of "otherwise" in the definition of the Katz back-off model and keep its representation compact. A failure transition is a transition taken at state q when no other transition leaving q has the desired label. Figure 4.3.2(a) illustrates that construction in the case of a trigram model ($n = 3$).

It is possible to give an explicit representation of these weighted automata without using failure transitions. However, the size of the resulting automata may become prohibitive. Instead, an approximation of that weighted automaton is used where failure transitions are simply replaced by ε-transitions. This turns out to cause only a very limited loss in accuracy.[6].

In practice, for numerical instability reasons negative-log probabilities are used and the language model weighted automaton is defined in the log semiring. Figure 4.3.2(b) shows the corresponding weighted automaton in a very simple case. We will denote by $\mathfrak{G}$ the weighted automaton representing the statistical grammar.

4.3.3. Pronunciation model

The representation of a pronunciation model $\mathbf{P}(p \mid w)$ (or lexicon) with weighted transducers is quite natural. Each word has a finite number of phonemic transcriptions. The probability of each pronunciation can be estimated from a corpus. Thus, for each word x, a simple weighted transducer $\mathfrak{T}_x$ mapping x to its phonemic transcriptions can be constructed.

Figure 4.8 shows that representation in the case of the English word *data*. The closure of the union of the transducers $\mathfrak{T}_x$ for all the words x considered gives a weighted transducer representation of the pronunciation model. We will denote by $\mathfrak{P}$ the equivalent transducer over the log semiring.

[6]An alternative when no offline optimization is used is to compute the explicit representation on-the-fly, as needed for the recognition of an utterance. There exists also a complex method for constructing an exact representation of an n-gram model which cannot be presented in this short chapter.

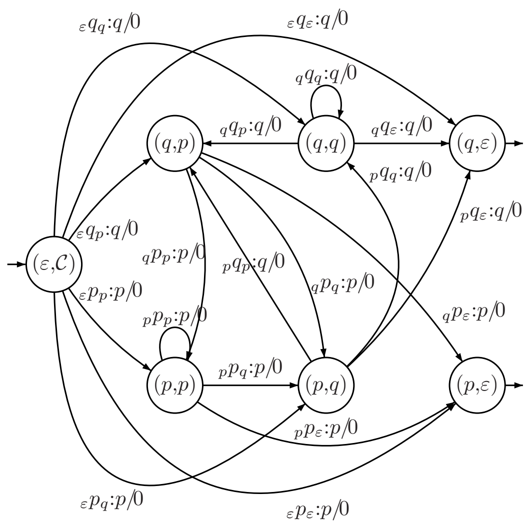


Figure 4.9. Context-dependency transducer restricted to two phones p and q.

4.3.4. Context-dependency transduction

The pronunciation of a phone depends on its neighboring phones. To design an accurate acoustic model, it is thus beneficial to model a *context-dependent phone*, i.e., a phone in the context of its surrounding phones. This has also been corroborated by empirical evidence. The standard models used in speech recognition are n-phonic models. A context-dependent phone is then a phone in the context of its n_1 previous phones and n_2 following phones, with $n_1 + n_2 + 1 = n$. Remarkably, the mapping $\mathbf{P}(c \mid d)$ from phone sequences to sequences of context-dependent phones can be represented by finite-state transducers. This section illustrates that construction in the case of triphonic models ($n_1 = n_2 = 1$). The extension to the general case is straightforward.

Let $\mathcal{P}$ denote the set of context-independent phones and let $\mathcal{C}$ denote the set of triphonic context-dependent phones. For a language such as English or French, $\mathrm{Card}(\mathcal{P}) \approx 50$. Let ${}_{p_1}p_{p_2}$ denote the context-dependent phone corresponding to the phone p with the left context p_1 and the right context p_2.

The construction of the context-dependency transducer is similar to that of the language model automaton. As in the previous case, for numerical instability reasons, negative log-probabilities are used, thus the transducer is defined in the log semiring. Each state encodes a history limited to the last two phones. There is a transition from the state associated to (p, q) to (q, r) with input label the context-dependent phone ${}_{p}q_r$ and output label q. More precisely, the transducer $\mathfrak{T} = (\mathcal{C}, \mathcal{P}, Q, I, F, E, \lambda, \rho)$ is defined by:

- $Q = \{(p, q) : p \in \mathcal{P}, q \in \mathcal{P} \cup \{\varepsilon\}\} \cup \{(\varepsilon, \mathcal{C})\}$;
- $I = \{(\varepsilon, \mathcal{C})\}$ and $F = \{(p, \varepsilon) : p \in \mathcal{P}\}$;

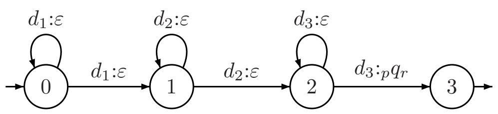


Figure 4.10. Hidden-Markov Model transducer.

- $E \subseteq \{((p, Y),\ {}_pq_r, q, 0, (q, r)) : Y = q \text{ or } Y = \mathcal{C}\}$

with all initial and final weights equal to zero. Figure 4.9 shows that transducer in the simple case where the phonemic alphabet is reduced to two phones ($\mathcal{P} = \{p, q\}$). We will denote by $\mathfrak{C}$ the weighted transducer representing the context-dependency mapping.

4.3.5. Acoustic model

In most modern speech recognition systems, context-dependent phones are modeled by three-state Hidden Markov Models (HMMs). Figure 4.10 shows the graphical representation of that model for a context-dependent model ${}_pq_r$. The context-dependent phone is modeled by three states ($0, 1$, and 2) each modeled with a distinct distribution (d_0, d_1, d_2) over the input observations. The mapping $\mathbf{P}(d \mid c)$ from sequences of context-dependent phones to sequences of distributions is the transducer obtained by taking the closure of the union of the finite-state transducers associated to all context-dependent phones. We will denote by $\mathfrak{H}$ that transducer. Each distribution d_i is typically a mixture of Gaussian distributions with mean μ and covariance matrix σ:

$$\mathbf{P}(\omega) = \frac{1}{(2\pi)^{N/2}|\sigma|^{1/2}} e^{-\frac{1}{2}(\omega-\mu)^T\sigma^{-1}(\omega-\mu)} \tag{4.3.10}$$

where ω is an observation vector of dimension N. Observation vectors are obtained by local spectral analysis of the speech waveform at regular intervals, typically every 10 ms. In most cases, they are 39-dimensional feature vectors ($N = 39$). The components are the 13 *cepstral coefficients*, i.e., the energy and the first 12 components of the *cepstrum* and their first-order (*delta cepstra*) and second-order differentials (*delta-delta cepstra*). The cepstrum of the (speech) signal is the result of taking the inverse-Fourier transform of the log of its Fourier transform. Thus, if we denote by $x(\omega)$ the Fourier transform of the signal, the 12 first coefficients c_n in the following expression:

$$\log|x(\omega)| = \sum_{n=-\infty}^{\infty} c_n e^{-in\omega} \tag{4.3.11}$$

are the coefficients used in the observation vectors. This truncation of the Fourier transform helps smooth the log magnitude spectrum. Empirically, cepstral coefficients have shown to be excellent features for representing the speech

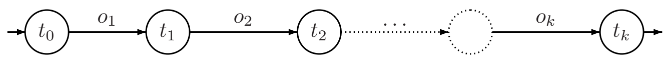


Figure 4.11. Observation sequence $O = o_1 \cdots o_k$. The time stamps t_i, $i = 0, \ldots k$, labeling states are multiples of 10 ms.

signal.[7] Thus the observation sequence $o = o_1 \cdots o_k$ can be represented by a sequence of 39-dimensional feature vectors extracted from the signal every 10 ms. This can be represented by a simple automaton as shown in figure 4.11, that we will denote by $\mathfrak{O}$.

We will denote by $\mathfrak{O} \star \mathfrak{H}$ the weighted transducer resulting from the application of the transducer $\mathfrak{H}$ to an observation sequence $\mathfrak{O}$. $\mathfrak{O} \star \mathfrak{H}$ is the weighted transducer mapping O to sequences of context-dependent phones, where the weights of the transitions are the negative log of the value associated by a distribution d_i to an observation vector O_j, -log $d_i(O_j)$.

4.3.6. Combination and search

The previous sections described the representation of each of the components of a speech recognition system by a weighted transducer or weighted automaton. This section shows how these transducers and automata can be combined and searched efficiently using the weighted transducer algorithms previously described, following Equation 4.3.4.

A so-called *Viterbi approximation* is often used in speech recognition. It consists of approximating a sum of probabilities by its dominating term:

$$\hat{w} = \operatorname*{argmax}_{w} \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.12}$$

$$\approx \operatorname*{argmax}_{w} \max_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.13}$$

This has been shown to be empirically a relatively good approximation, though, most likely, its introduction was originally motivated by algorithmic efficiency. For numerical instability reasons, negative-log probabilities are used, thus the equation can be reformulated as:

$$\hat{w} = \operatorname*{argmin}_{w} \min_{d,c,p} -\log \mathbf{P}(o \mid d) - \log \mathbf{P}(d \mid c) - \log \mathbf{P}(c \mid p) - \log \mathbf{P}(p \mid w) - \log \mathbf{P}(w)$$

As discussed in the previous sections, these models can be represented by weighted transducers. Using the composition algorithm for weighted transducers, and by definition of the $\star$-operation and projection, this is equivalent

[7] Most often, the spectrum is first transformed using the Mel Frequency bands, which is a non-linear scale approximating the human perception.

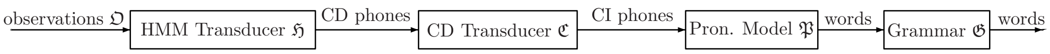


Figure 4.12. Cascade of speech recognition transducers.

to:[8]

$$\hat{w} = \operatorname*{argmin}_{w} \Pi_2(\mathfrak{O} \star \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G}) \tag{4.3.14}$$

Thus, speech recognition can be formulated as a cascade of composition of weighted transducers illustrated by Figure 4.12. $\hat{w}$ labels the path of $\mathfrak{W} = \Pi_2(\mathfrak{O} \star \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G})$ with the lowest weight. The problem can be viewed as a classical single-source shortest-paths algorithm over the weighted automaton $\mathfrak{W}$. Any single-source shortest paths algorithm could be used to solve it. In fact, since $\mathfrak{O}$ is finite, the automaton $\mathfrak{W}$ could be acyclic, in which case the classical linear-time single-source shortest-paths algorithm based on the topological order could be used.

However, this scheme is not practical. This is because the size of $\mathfrak{W}$ can be prohibitively large even for recognizing short utterances. The number of transitions of $\mathfrak{O}$ for 10s of speech is 1000. If the recognition transducer $\mathfrak{T} = \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G}$ had in the order of just 100M transitions, the size of $\mathfrak{W}$ would be in the order of 1000×100M transitions, i.e., about 100 billion transitions!

In practice, instead of visiting all states and transitions, a heuristic pruning is used. A pruning technique often used is the *beam search*. This consists of exploring only states with tentative shortest-distance weights within a *beam* or threshold of the weight of the best *comparable* state. Comparable states must roughly correspond to the same observations, thus states of $\mathfrak{T}$ are visited in the order of analysis of the input observation vectors, i.e. chronologically. This is referred to as a *synchronous beam search*. A synchronous search restricts the choice of the single-source shortest-paths problem or the relaxation of the tentative shortest-distances. The specific single-source shortest paths algorithm then used is known as the *Viterbi Algorithm*, which is presented in Exercise 1.3.1.

The $\star$-operation, the Viterbi algorithm, and the beam pruning techniques are often combined into a *decoder*. Here is a brief description of the decoder. For each observation vector o_i read, the transitions leaving the current states of $\mathfrak{T}$ are expanded, the $\star$-operation is computed on-the-fly to compute the acoustic weights given by the application of the distributions to o_i. The acoustic weights are added to the existing weight of the transitions and out of the set of states

[8]Note that the Viterbi approximation can be viewed simply as a change of semiring, from the log semiring to the tropical semiring. This does not affect the topology or the weights of the transducers but only their interpretation or use. Also, note that composition does not make use of the first operation of the semiring, thus compositions in the log and tropical semiring coincide.

reached by these transitions those with a tentative shortest-distance beyond a pre-determined threshold are pruned out. The beam threshold can be used as a means to select a trade-off between recognition speed and accuracy. Note that the pruning technique used is non-admissible. The best overall path may fall out of the beam due to local comparisons.

4.3.7. Optimizations

The characteristics of the recognition transducer $\mathfrak{T}$ were left out of the previous discussion. They are however key parameters for the design of real-time large-vocabulary speech recognition systems. The search and decoding speed critically depends on the size of $\mathfrak{T}$ and its non-determinism. This section describes the use of the determinization, minimization, and weight pushing algorithm for constructing and optimizing $\mathfrak{T}$.

The component transducers described can be very large in speech recognition applications. The weighted automata and transducers we used in the North American Business news (NAB) dictation task with a vocabulary of just 40,000 words (the full vocabulary in this task contains about 500,000 words) had the following attributes:

- $\mathfrak{G}$: a *shrunk* Katz back-off trigram model with about 4M transitions;[9]
- $\mathfrak{P}$: pronunciation transducer with about $70,000$ states and more than 150,000 transitions;
- $\mathfrak{C}$: a triphonic context-dependency transducer with about 1,500 states and 80,000 transitions.
- $\mathfrak{H}$: an HMM transducer with more than 7,000 states.

A full construction of $\mathfrak{T}$ by composition of such transducers without any optimization is not possible even when using very large amounts of memory. Another problem is the non-determinism of $\mathfrak{T}$. Without prior optimization, $\mathfrak{T}$ is highly non-deterministic, thus, a large number of paths need to be explored at the search and decoding time, thereby considerably slowing down recognition.

Weighted determinization and minimization algorithms provide a general solution to both the non-determinism and the size problem. To construct an optimized recognition transducer, weighted transducer determinization and minimization can be used at each step of the composition of each pair of component transducers. The main purpose of the use of determinization is to eliminate non-determinism in the resulting transducer, thereby substantially reducing recognition time. But, its use at intermediate steps of the construction also helps improve the efficiency of composition and reduce the size of the resulting transducer. We will see later that it is in fact possible to construct offline the recognition transducer and that its size is practical for real-time speech recognition!

[9] Various *shrinking methods* can be used to reduce the size of a statistical grammar without affecting its accuracy excessively.

However, as pointed out earlier, not all weighted automata and transducers are determinizable, e.g., the transducer $\mathfrak{P} \circ \mathfrak{G}$ mapping phone sequences to words is in general not determinizable. This is clear in presence of homophones. But even in the absence of homophones, $\mathfrak{P} \circ \mathfrak{G}$ may not have the twins property and be non-determinizable. To make it possible to determinize $\mathfrak{P} \circ \mathfrak{G}$, an auxiliary phone symbol denoted by $\#_0$ marking the end of the phonemic transcription of each word can be introduced. Additional auxiliary symbols $\#_1 \ldots \#_{k-1}$ can be used when necessary to distinguish homophones as in the following example:

`r eh d` $\#_0$	*read*
`r eh d` $\#_1$	*red*

At most D auxiliary phones, where D is the maximum degree of homophony, are introduced. The pronunciation transducer augmented with these auxiliary symbols is denoted by $\tilde{\mathfrak{P}}$. For consistency, the context-dependency transducer $\mathfrak{C}$ must also accept all paths containing these new symbols. For further determinizations at the context-dependent phone level and distribution level, each auxiliary phone must be mapped to a distinct context-dependent phone. Thus, self-loops are added at each state of $\mathfrak{C}$ mapping each auxiliary phone to a new auxiliary context-dependent phone. The augmented context-dependency transducer is denoted by $\tilde{\mathfrak{C}}$.

Similarly, each auxiliary context-dependent phone must be mapped to a new distinct distribution. D self-loops are added at the initial state of $\mathfrak{H}$ with auxiliary distribution input labels and auxiliary context-dependency output labels to allow for this mapping. The modified HMM transducer is denoted by $\tilde{\mathfrak{H}}$.

It can be shown that the use of the auxiliary symbols guarantees the determinizability of the transducer obtained after each composition. Weighted transducer determinization is used at several steps of the construction. An n-gram language model $\mathfrak{G}$ is often constructed directly as a deterministic weighted automaton with a back-off state – in this context, the symbol ε is treated as a regular symbol for the definition of determinism. If this does not hold, $\mathfrak{G}$ is first determinized. $\tilde{\mathfrak{P}}$ is then composed with $\mathfrak{G}$ and determinized: $det(\tilde{\mathfrak{P}} \circ \mathfrak{G})$. The benefit of this determinization is the reduction of the number of alternative transitions at each state to at most the number of distinct phones at that state (≈ 50), while the original transducer may have as many as V outgoing transitions at some states where V is the vocabulary size. For large tasks where the vocabulary size can be more than several hundred thousand, the advantage of this optimization is clear.

The inverse of the context-dependency transducer might not be deterministic.[10] For example, the inverse of the transducer shown in Figure 4.9 is not deterministic since the initial state admits several outgoing transitions with the same input label p or q. To construct a small and efficient integrated transducer, it is important to first determinize the inverse of $\mathfrak{C}$.[11]

[10] The inverse of a transducer is the transducer obtained by swapping input and output labels of all transitions.

[11] Triphonic or more generally n-phonic context-dependency models can also be constructed directly with a deterministic inverse.

$\tilde{\mathfrak{C}}$ is then composed with the resulting transducer and determinized. Similarly $\tilde{\mathfrak{H}}$ is composed with the context-dependent transducer and determinized. This last determinization increases sharing among HMM models that start with the same distributions: at each state of the resulting integrated transducer, there is at most one outgoing transition labeled with any given distribution name. This leads to a substantial reduction of the recognition time.

As a final step, the auxiliary distribution symbols of the resulting transducer are simply replaced by ε's. The corresponding operation is denoted by Π_ε. The sequence of operations just described is summarized by the following construction formula:

$$\mathfrak{N} = \Pi_\varepsilon(det(\tilde{\mathfrak{H}} \circ det(\tilde{\mathfrak{C}} \circ det(\tilde{\mathfrak{P}} \circ \mathfrak{G})))) \tag{4.3.15}$$

where parentheses indicate the order in which the operations are performed. Once the recognition transducer has been determinized, its size can be further reduced by minimization. The auxiliary symbols are left in place, the minimization algorithm is applied, and then the auxiliary symbols are removed:

$$\mathfrak{N} = \Pi_\varepsilon(min(det(\tilde{\mathfrak{H}} \circ det(\tilde{\mathfrak{C}} \circ det(\tilde{\mathfrak{P}} \circ \mathfrak{G}))))) \tag{4.3.16}$$

Weighted minimization can also be applied after each determinization step. It is particularly beneficial after the first determinization and often leads to a substantial size reduction. Weighted minimization can be used in different semirings. Both minimization in the tropical semiring and minimization in the log semiring can be used in this context. The results of these two minimizations have exactly the same number of states and transitions and only differ in how weight is distributed along paths. The difference in weights arises from differences in the definition of the key pushing operation for different semirings.

Weight pushing in the log semiring has a very large beneficial impact on the pruning efficacy of a standard Viterbi beam search. In contrast, weight pushing in the tropical semiring, which is based on lowest weights between paths described earlier, produces a transducer that may slow down beam-pruned Viterbi decoding many fold.

The use of pushing in the log semiring preserves a desirable property of the language model, namely that the weights of the transitions leaving each state be normalized as in a probabilistic automaton. Experimental results also show that pushing in the log semiring makes pruning more effective. It has been conjectured that this is because the acoustic likelihoods and the transducer probabilities are then synchronized to obtain the optimal likelihood ratio test for deciding whether to prune. It has been further conjectured that this reweighting is the best possible for pruning. A proof of these conjectures will require a careful mathematical analysis of pruning.

The result $\mathfrak{N}$ is an integrated recognition transducer that can be constructed even in very large-vocabulary tasks and leads to a substantial reduction of the recognition time as shown by our experimental results. Speech recognition is thus reduced to the simple Viterbi beam search described in the previous section applied to $\mathfrak{N}$.

In some applications such as for spoken-dialog systems, one may wish to modify the input grammar or language model $\mathfrak{G}$ as the dialog proceeds to exploit the context information provided by previous interactions. This may be to activate or deactivate certain parts of the grammar. For example, after a request for a location, the date sub-grammar can be made inactive to reduce alternatives.

The offline optimization techniques just described can sometimes be extended to the cases where the changes to the grammar $\mathfrak{G}$ are pre-defined and limited. The grammar can then be factored into sub-grammars and an optimized recognition transducer is created for each. When deeper changes are expected to be made to the grammar as the dialog proceeds, each component transducer can still be optimized using determinization and minimization and the recognition transducer $\mathfrak{N}$ can be constructed on-demand using an on-the-fly composition. States and transitions of $\mathfrak{N}$ are then expanded as needed for the recognition of each utterance.

This concludes our presentation of the application of weighted transducer algorithms to speech recognition. There are many other applications of these algorithms in speech recognition, including their use for the optimization of the word or phone lattices output by the recognizer that cannot be covered in this short chapter.

We presented several recent weighted finite-state transducer algorithms and described their application to the design of large-vocabulary speech recognition systems where weighted transducers of several hundred million states and transitions are manipulated. The algorithms described can be used in a variety of other natural language processing applications such as information extraction, machine translation, or speech synthesis to create efficient and complex systems. They can also be applied to other domains such as image processing, optical character recognition, or bioinformatics, where similar statistical models are adopted.

Notes

Much of the theory of weighted automata and transducers and their mathematical counterparts, rational power series, was developed several decades ago. Excellent reference books for that theory are Eilenberg (1974), Salomaa and Soittola (1978), Berstel and Reutenauer (1984) and Kuich and Salomaa (1986).

Some essential weighted transducer algorithms such as those presented in this chapter, e.g., composition, determinization, and minimization of weighted transducers are more recent and raise new questions, both theoretical and algorithmic. These algorithms can be viewed as the generalization to the weighted case of the composition, determinization, minimization, and pushing algorithms described in Chapter 1 Section 1.5. However, this generalization is not always straightforward and has required a specific study.

The algorithm for the composition of weighted finite-state transducers was given by Pereira and Riley (1997) and Mohri, Pereira, and Riley (1996). The composition filter described in this chapter can be refined to exploit information about the composition states, e.g., the finality of a state or whether only ε-transitions or only non ε-transitions leave that state, to reduce the number of non-coaccessible states created by composition.

The generic determinization algorithm for weighted automata over weakly left divisible left semirings presented in this chapter as well as the study of the determinizability of weighted automata are from Mohri (1997). The determinization of (unweighted) finite-state transducers can be viewed as a special instance of this algorithm. The definition of the twins property was first formulated for finite-state transducers by Choffrut (see Berstel (1979) for a modern presentation of that work). The generalization to the case of weighted automata over the tropical semiring is from Mohri (1997). A more general definition for a larger class of semirings, including the case of finite-state transducers, as well as efficient algorithms for testing the twins property for weighted automata and transducers under some general conditions is presented by Allauzen and Mohri (2003).

The weight pushing algorithm and the minimization algorithm for weighted automata were introduced by Mohri 1997. The general definition of shortest-distance and that of k-closed semirings and the generic shortest-distance algorithm mentioned appeared in Mohri (2002). Efficient implementations of the weighted automata and transducer algorithms described as well as many others are incorporated in a general software library, AT&T FSM Library, whose binary executables are available for download for non-commercial use (Mohri et al. (2000)).

Bahl, Jelinek, and Mercer 1983 gave a clear statistical formulation of speech recognition. An excellent tutorial on Hidden Markov Model and their application to speech recognition was presented by Rabiner (1989). The problem of the estimation of the probability of unseen sequences was originally studied by Good 1953 who gave a brilliant discussion of the problem and provided a principled solution. The back-off n-gram statistical modeling is due to Katz (1987). See Lee (1990) for a study of the benefits of the use of context-dependent models in speech recognition.

The use of weighted finite-state transducers representations and algorithms in statistical natural language processing was pioneered by Pereira and Riley (1997) and Mohri (1997). Weighted transducer algorithms, including those described in this chapter, are now widely used for the design of large-vocabulary speech recognition systems. A detailed overview of their use in speech recognition is given by Mohri, Pereira, and Riley (2002). Sproat 1997 and Allauzen, Mohri, and Riley 2004 describe the use of weighted transducer algorithms in the design of modern speech synthesis systems. Weighted transducers are used in a variety of other applications. Their recent use in image processing is described by Culik II and Kari (1997).

CHAPTER 5

Inference of Network Expressions

5.0. Introduction

This chapter introduces various mathematical models and combinatorial algorithms for inferring network expressions that appear repeated in a word or are common to a set of words, where by *network expression* is meant a regular expression without Kleene closure on the alphabet of the input word(s). A network expression on such an alphabet is therefore any expression built up of concatenation and union operators. For example, the expression $\mathtt{A}(\mathtt{C}+\mathtt{G})\mathtt{T}$ concatenates $\mathtt{A}$ with the union $(\mathtt{C}+\mathtt{G})$ and with $\mathtt{T}$. Inferring network expressions means *discovering* such expressions which are initially unknown. The only input is the word(s) where the repeated (or common) expressions will be sought for. This is in contrast with another problem we shall not be concerned with, that consists in *searching* for a known expression in a word(s) both of which are in this case part of the input. The inference of network expressions has many applications, notably in molecular biology, system security, text mining etc. Because of the richness of the mathematical and algorithmic Al problems posed by molecular biology, we concentrate on applications in this area. The network expressions considered may therefore contain spacers where by *spacer* is meant any number of don't care symbols (a don't care is a symbol that matches anything). *Constrained spacers* are consecutive don't care symbols whose number ranges over a fixed interval of values. Network expressions with don't care symbols but no spacers are called "simple" while network expressions with spacers are called "flexible" if the spacers are unconstrained, and "structured" otherwise. Both notions are important in molecular biology. Applications to biology motivate us also to consider network expressions that appear repeated not exactly but approximately.

Only exact combinatorial methods that are non-trivial (that is, are not simple brute-force schemes which enumerate all possible network expressions) will be mentioned. In most cases, the network expressions that have been considered in the literature present some constraint that generally applies to the union operator. Indeed, the operands concerned by the union operation will most often be elements of the alphabet $\mathcal{A}$ and not arbitrary words in $\mathcal{A}^+$ as is the case with unrestricted network expressions. For instance, we do not allow expressions such as $\mathtt{A}(\mathtt{CG}+\mathtt{G})\mathtt{T}$.

The literature on the inference of regular expressions, also called grammatical inference, is vast, and predates computational biology by many years. The inference problems addressed in this chapter present special characteristics in relation to such general problems. The most important ones are that, although the expressions considered here are simpler in the sense indicated above, their occurrences are not exact and come hidden inside often very large texts. To use the terms commonly adopted in the grammatical inference community, we work with (positive) examples that have first to be fished from a sea of other textual information, consisting mostly in noise. Most often, there is not one regular expression only, and thus one set of examples, but various distinct ones hidden in the same text.

5.1. Inferring simple network expressions: models and related problems

5.1.1. The star model

A *star expression* X is an expression of the form $X = e_1e_2\cdots e_m$ where each e_i is the union of elements of the alphabet, i.e. $e_i = a_{i,1} + a_{i,2} + \cdots + a_{i,n_i}$ with $a_{i,j} \in \mathcal{A}$ for $1 \le j \le n_i \le \mathrm{Card}(\mathcal{A})$, $1 \le i \le m$. Star expressions are therefore words on the alphabet $\mathcal{P}(\mathcal{A})$ of all non empty subsets of $\mathcal{A}$, that is, they are elements of $\mathcal{P}(\mathcal{A})^+$. This includes the set $\mathcal{A}$ which we denote by $\bullet$ and call the *don't care* symbol[1]. Let $F(w)$ denote the set of factors of a word w. A star expression denotes also a finite set of words of length m. A star expression

[1]One also finds in the literature the terms *wild card* and *joker*.

$X \in \mathcal{P}(\mathcal{A})^+$ is said to occur *exactly* in a word $w \in \mathcal{A}^+$ if there exists $v \in F(w)$ such that $v \in X$. The factor v is said to be an *occurrence* of X in w.

The notion of approximate occurrence relies on the notion of a distance between two words u and v on $\mathcal{A}$. In biology, like in a number of other text applications, a natural distance measures the effort required to transform one word into the other given certain allowed operations. The operations that model best the mutational events that may happen during replication and survive are the *substitution* (*i.e.* replacement) of a letter of $\mathcal{A}$ by another, the *deletion* of a letter in one of the two words, and the *insertion* of a letter. Finally, a *match* is an operation that leaves the letter unchanged. These are called *edit operations*.

Let S, D, I, M denote the four edit operations described above. An *edit transcript* is a string over the alphabet S, D, I, M that describes a transformation of a word u into another v. An equivalent way of describing such transformation is through a *global alignment* of u and v. This is obtained by inserting spaces in both u and v, transforming them into u' and v' defined over $\mathcal{A} \cup \{-\}$ where $\{-\}$ denotes a space and $|u'| = |v'|$. An edit transcript can be easily converted into a global alignment and vice-versa.

A cost c may be attributed to each operation where c is a function $(\mathcal{A} \cup \{-\}) \times (\mathcal{A} \cup \{-\}) \to \mathbb{R}$. The cost of a global alignment (and thus of the associated edit transcript) of two words u' and v' of length n is then $\sum_{i=0}^{n-1} c(u'[i], v'[i])$. Not all cost functions define a distance. This will be the case if the function c is symmetric and if $c(a, b)$ for $a, b \in \mathcal{A} \cup \{-\}$ is strictly greater than 0 if $a \neq b$ and is 0 otherwise.

Two types of distances have attracted special attention. These are the *Hamming* and the *edit* distance (this last is also called the *Levenshtein* distance).

In the case of the edit distance, the cost function is

$$c(a, b) = \begin{cases} 1 & \text{if } \ a \neq b; \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } a, b \in \mathcal{A} \cup \{-\}$$

The edit distance is thus the minimum number of substitutions, insertions and deletions required to transform u into v. The Hamming distance applies only to words of same length, and is restricted to substitution/match operations. The cost function is the same as for the edit distance applied to $a, b \in \mathcal{A}$. It counts the minimum number of substitutions needed to obtain v from u assuming they have the same length. The Hamming distance will be denoted by $dist_H$ and the edit distance by $dist_E$.

Given a positive integer d, a d-occurrence of X in a word w is a word $v \in F(w)$ such that there exists a word $u \in X$ with $dist(u, v) \leq d$ for some fixed d. An expression $X \in \mathcal{P}(\mathcal{A})^+$ is thus said to appear *approximately* in w if it has a d-occurrence in w. Where there is no possible ambiguity, reference to d will be dropped in all such notations.

The distances considered between words v and u are, as suggested, usually Hamming or edit, although any other may be used. For ease of exposition, we consider Hamming distance exclusively. Issues related to the use of the edit distance instead are left as open problems in Section 5.5.1.

The above definitions of approximate occurrence of an expression lead to the following inference problem statements. They call upon the concept of *quorum*. This is the minimum number of times an expression must appear repeated in the input word(s). In the case of a set of words, the quorum is the number of distinct words where the expression appears.

Expression inference problem for a star expression

Expression X repeated in a word

INPUT: A word $w \in \mathcal{A}^+$, a quorum q and a distance d.

OUTPUT: All star expressions $X \in \mathcal{P}(\mathcal{A})^+$ that occur d-approximately in w at least q times.

Expression X common to a set of words

INPUT: N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a quorum q, a distance d.

OUTPUT: All star expressions $X \in \mathcal{P}(\mathcal{A})^+$ that occur d-approximately in at least q of the N words $w_1, w_2, \ldots, w_N$.

Observe that nothing, except algorithmic Al concerns, would forbid to consider in all the above definitions and problem statements, network expressions without constraints on the union operator.

These definitions and statements have been adopted in a number of exact algorithms, namely Combi, Poivre, Speller, Smile, Pratt, and MitraCount. The main difference between the algorithms has been in the type of further constraints put on the network expressions allowed. In Combi, the expressions are indeed elements of $\mathcal{P}(\mathcal{A})^+$ with just a constraint on the number of times the don't care symbol $\mathcal{A}$ may be used in the expression. This last constraint is used by all algorithms. In Poivre as in Pratt, the expressions are over an alphabet $\mathcal{S}$ where $\mathcal{S}$ is a proper subset of $\mathcal{P}(\mathcal{A})$. In Pratt furthermore, expressions must appear exactly in a word ($d = 0$). In Speller and MitraCount, the expressions are elements of $\mathcal{A}^+$. Speller was later extended to handle elements of $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$ as is the case for Smile.

The model described in this section is called a *star* model because the expressions X given as output may be viewed as the center of star trees whose edges have as length the distance between X and each of its occurrences in the input word(s). A special case of the problem of finding such expressions has been called the *Closest substring problem.* This is stated in the following way, where by $F_k(w)$ we denote the set $F(w) \cap \mathcal{A}^k$ of w having length k.

Closest substring problem

INPUT: N words $w_1, w_2, \ldots, w_N$ over the alphabet $\mathcal{A}$ and integers d and k.

OUTPUT: A word x of length k over $\mathcal{A}$ such that there exist $u_i \in F_k(w_i)$ with $dist_H(x, u_i) \leq d$ for all $1 \leq i \leq N$.

It is interesting to observe the relation between an expression within the star model and another well known mathematical object: Steiner strings. Given a set of words $x_1, \ldots, x_N$ of same length in $\mathcal{A}^+$, a *Steiner string* is a word $\bar{x}$ also in $\mathcal{A}^+$ which minimizes $\sum_{i=0}^{N} dist_H(\bar{x}, x_i)$. One may then wonder whether an expression X that solves the *Expression inference problem for star expressions* leads to a Steiner string, that is, are the star expressions found also solutions of the Steiner string problem for their sets of occurrences? The answer is negative. A simple counter-example is the star expression `ACAA` repeated in the word $w =$ `AAAAAACAC` with $d = 1$ and $q = 4$. The expression `ACAA` of length 4 is indeed a solution of the star expression inference problem since it has 4 occurrences in w, namely at positions $0, 1, 2, 5$ (positions in a word start at 0). Expression `AAAC` is also a solution with 5 occurrences, at positions $0, 1, 2, 3, 5$. Neither expression is a Steiner string of its set of occurrences. In both cases, this is the word `AAAA`. Notice that `AAAA` is *not* itself a solution of the problem.

The star-model admits a variant that is interesting for some biological applications. The variant applies to the case of expressions common to a set of words and assumes a phylogenetic tree is given as input together with the words. This is a binary tree that represents the speciation events that have led to the different species currently existing (each represented by a word in the input words set and associated to a leaf of the tree) from the ancestors that are not known. The tree may be unrooted, or rooted if the order of the events in time is known. We assume here that the tree is rooted. It is this tree, and not a star-tree, that is used to compute the distances.

In fact, each edge in the tree is labeled by the Hamming distance between the words at its extremities. Given a phylogenetic tree, a set of words placed at the leaves, and an integer k, factors of length k, one for each input word, and intermediate expressions corresponding to internal nodes have to be inferred such that the sum of the labels over all edges of the tree is minimized. Such minimal sum is called *parsimony score*. The expressions are elements of $\mathcal{A}^+$, the problem (in its decision version) as addressed by the FOOTPRINTER algorithm is as follows.

Substring parsimony problem

> **INPUT:** N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a phylogenetic tree $\mathcal{F}$ for the words, a length k, and an integer d.
>
> **OUTPUT:** Factors of length k for the leaves and words of length k for all internal nodes of the tree that have a parsimony score at most d.

Another variant that has been considered constrains the expressions given as solutions to the expression inference problem to satisfy an *uniform property*. Suppose expressions of a length k are sought and let X be a solution of the problem. Let also two positive integers, $d' < d$ and $k' < k$, be given. The following must then be true: for all i such that $0 \leq i \leq |X|$, $dist_H(X[i..i + k' - 1], v[i..i + k' - 1]) \leq d'$. Intuitively, this constraint imposes that the possible differences between an expression X solution of the problem and each of its occurrences be uniformly spread, hence the name given to the property. A further variant has been used in WEEDER where the constraint that must be satisfied is: for all i such that $0 \leq i \leq |X|$, $dist_H(X[0..i], v[0..i]) \leq \lfloor \frac{i \times d}{k} \rfloor$. The inconvenient of a constraint of this latter type is that an asymmetry is introduced: differences may accumulate at the end of the occurrences of an expression.

5.1.2. The clique model

In this model, given an alphabet $\mathcal{A}$, the network expressions on $\mathcal{A}$ that are considered have the constraint that union operators may this time be applied to elements of $\mathcal{A}^k$ only, where k is the length of the expressions sought. Such expressions are therefore collections of words $w \in \mathcal{A}^k$, and the notion of occurrence is associated to both a distance and a quorum. The definition of occurrence in this case is thus operational and includes within it the problem statement.

Such an alternative model has been used by some authors, most notably in the algorithms WINNOWER, MITRA-GRAPH and KMRC. MITRA-GRAPH uses in fact both models: the clique model and the star. WINNOWER and MITRA-GRAPH formalize the model and problem in graph-theoretical terms.

We state the problem in the case of a set of words. Given a set of N words $w_1, w_2, \ldots, w_N$ over the alphabet $\mathcal{A}$ and two non negative integers d and k, let $\mathcal{G} = (V_1 \cup \cdots \cup V_N, E)$ be an N-partite graph where $V_i = F(w_i) \cap \mathcal{A}^k$ (*i.e*, it is the set of all factors of length k of w_i for all i) and there is an edge between $v_i \in V_i, v_j \in V_j$ for $i \neq j$ if $dist_H(w_i, w_j) \leq 2d$. Given d and k as above, and given a quorum q, we say that a network expression X is a (d, q)-clique if and only if it is a set of q words of length k such that $(u, v) \in E$ for all $u, v \in X$.

The problem is then formulated as follows.

Expression inference as a clique problem

INPUT: N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a quorum q, a length k, and a distance d; the associated N-partite graph $\mathcal{G}$.

OUTPUT: All (d, q)-cliques of $\mathcal{G}$.

The output cliques correspond to expressions X having occurrences that are all pairwise $2d$-approximations[2] of each other in at least q of the N words $w_1, w_2, \ldots, w_N$. Expression X is the union of its occurrences.

In POIVRE instead, the graph is built upon a relation R between the letters of the alphabet $\mathcal{A}$ that is also part of the input. Typically, the relation will be non transitive and model the degree to which shared physico-chemical properties between the biological units denoted by the letters (nucleic or amino acids) enables them to perform equivalent functions in a molecule. The relation R on the letters of $\mathcal{A}$ is straightforwardly extended to a relation R on words of the same length in $\mathcal{A}^+$ as follows: two factors u, v of length k are in relation by R extended if $u[i]$ is in relation with $v[i]$ by R, for $0 \leq i \leq k - 1$. Relation R extended to factors of length k is denoted by R_k. The problem is then

[2]It will be explained later in this section why a $2d$ threshold is used instead of simply d.

expressed in a way that resembles the formulation given above except that graph $\mathcal{G} = (V_1 \cup \ldots \cup V_N, E)$ is now such that there is an edge between nodes $v_i \in V_i, v_j \in V_j$ for $i \neq j$ if the corresponding factors in w_i, w_j are in relation by R_k. In POIVRE, the idea was used to infer contiguous motifs in protein structures previously coded into a string of pairs of angles whose values are discretized into integers by means of a grid.

A natural question is whether for each solution X of the expression inference as a clique problem in the case of WINNOWER, there exists an expression Y such that $|Y| = |X|$ and Y is a solution under the star model for the collection of words that is the set of occurrences of X, distance d and same quorum. The answer is no. Let us assume expressions in $\mathcal{A}^+$ only are considered. Let the input words be the set $\{w_1 = \texttt{ACAC}, w_2 = \texttt{AGAG}, w_3 = \texttt{ATAT}\}$, $d = 1$ and $q = 3$. The expressions in $\mathcal{A}^+$ of length 4, $X_1 = \texttt{ACAC}$, $X_2 = \texttt{AGAG}$ and $X_3 = \texttt{ATAT}$ are solutions of the *Expression inference as a clique problem* as the (non proper) factors $w_1 = \texttt{ACAC}, w_2 = \texttt{AGAG}, w_3 = \texttt{ATAT}$ in the three input words w_i form a clique of the 3-partite graph $\mathcal{G}$. Yet no expression Y in $\mathcal{A}^4$ exists that is a solution of the problem as formulated in the star model for $\frac{d}{2}$. Obviously, there are solutions for distance d. In that case however, there may be more solutions in the star model, some of which have more occurrences. Consider this time the following set of input words $\{w_1 = \texttt{ACAC}, w_2 = \texttt{AGAG}, w_3 = \texttt{ATAT}, w_4 = \texttt{TCTG}\}$ and d, q as before. Expression $X = \texttt{ACAC}|\texttt{AGAG}|\texttt{ATAT}$ remains a solution within the clique model for quorum 3. Within the star model and with $\frac{d}{2} = 1$, any expression of the type $\texttt{AbAc}$, with $\texttt{b}, \texttt{c} \in \mathcal{A}$, $\texttt{b} \neq \texttt{c}$, is a solution with three occurrences while expression $Y = \texttt{ACAG}$ is also a solution but with four occurrences (w_1, w_2, w_3, w_4).

5.1.3. Other models

Other models have been used, in general for inferring expressions that are either elements of $\mathcal{A}^+$ or sets of elements of $\mathcal{A}^k$ for a given positive integer k. In the case of expressions in $\mathcal{A}^+$, those sought are the "most surprising" ones in the statistical sense. They correspond to expressions whose probability of occurring (exactly or approximately) in the input word(s) is lower than expected assuming a certain statistical model that is in general a Markov model of order p of the input word(s) for $p < (k-1)$. The algorithms for computing the "most surprising" expressions either perform brute-force enumeration of all possible expressions and then sophisticated exact or approximate statistical evaluation (described in detail in Chapter 6), or are heuristic (*e.g* PROFILE). We therefore do not speak of this model further.

In the case of expressions that are sets of elements of $\mathcal{A}^k$ for a given positive integer k, it is worth mentioning that another measure has been used to decide whether a set of factors in some input word(s) should be grouped. The measure corresponds to what has been called the *relative entropy* of a set of words or *Kullback-Leibler information number*. This measure is not a distance (triangular inequality is not satisfied) and is global: it is not built upon a pairwise relation between the factors and therefore it does not lead to a graph-theoretical formulation as above. Algorithms that seek network expressions that are elements of $\mathcal{A}^+$ can use the relative entropy of the sets of occurrences of the expressions given as output as a measure of "surprise" that will be different from the measure given by the probability of occurrence of the expression under the same conditions as it was inferred.

5.2. Algorithms

5.2.1. Inference in the star model

5.2.1.1. Preliminaries: suffix tree and generalized suffix tree

Long words, specially when they are defined over a small alphabet, may contain many exact repetitions. From this observation follows the idea of using an indexed representation of the input word(s). Using a representation that has a unique pointer for identical factors enables to avoid comparing such factors more than once with the expressions as these are inferred. The index used by the algorithm whose description follows is a suffix tree.

Details on the suffix tree construction may be found in 2. We just recall below that the suffix tree of an input word w, denoted by $\mathcal{T}_w$ or simply $\mathcal{T}$ when the input word is clear from the context, has the following properties:

1. each edge of the suffix tree is labelled by a non empty factor of the input word w;
2. each internal node of the suffix tree has at least two edges leaving it;
3. the factors labelling distinct edges that leave a same node start with distinct letters;
4. the label of each root-to-leaf path in the suffix tree represents a suffix of the input word and the label of each root-to-node path represents a factor;
5. each suffix of w is associated with the label of a (unique) root-to-leaf path in the tree.

Furthermore, an edge links the node spelling ax to the node spelling x for every $a \in \mathcal{A}$ and $x \in \mathcal{A}^*$. Such edges are called *suffix links* and are what allows the tree to be built in time linear with respect to the length of the word.

When the input consists in a set of words $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$, a tree called *generalized suffix tree* is used to represent in a compact way all the suffixes of the set of words. A generalized suffix tree is constructed in a way very similar to the suffix tree for a single word. We denote such generalized trees by $\mathcal{GT}_\mathcal{W}$ or simply $\mathcal{GT}$ when the input words are clear from the context. Generalized suffix trees have properties similar to those of a suffix tree with word w substituted by the set of words $\mathcal{W}$. In particular, a generalized suffix tree $\mathcal{GT}$ satisfies the fact that every suffix of every word w_i in the set leads to a distinct leaf. When several words share a suffix, the generalized suffix tree must have as many leaves corresponding to the suffix, each associated with a different word. To achieve this property during construction requires simply concatenating to each word w_i a symbol that is not in $\mathcal{A}$ and that is specific to that word.

5.2.1.2. Speller algorithm

We describe in this section the *original* Speller algorithm. The algorithm was later extended to allow for more general network expressions (over $\mathcal{P}(\mathcal{A})^+$ and not just $\mathcal{A}^+$) and its performance was improved (see Section 5.3.2.1).

For ease of exposition, the algorithm is first described for inferring expressions of fixed length that are repeated in a single word. In fact, the length of the expressions output by the algorithm may range over an interval (k_{min}, k_{max}) with possibly $k_{max} = \infty$. In this last case, the longest expressions output will be those still satisfying the quorum. When $k_{min} = k_{max} = \infty$, only the longest expressions satisfying the quorum are output. It is relatively straightforward to modify the algorithm to treat any of these cases, or to infer expressions common to a set of words, as will be briefly indicated.

Speller uses a suffix tree representation of the input word. Actually, it builds (at the same cost) a suffix tree with an additional information attached to each non-leaf node indicating the number of leaves in the subtree rooted at that node. This is also the number of occurrences in w of the factor spelled by the path from the root to the node. Denoting both node and factor by v, the information added to node v is denoted by $\ell(v)$.

Candidate expressions in $\mathcal{A}^+$ are for convenience processed in lexicographical order, starting from the empty word ε. For each candidate expression, say x, all pointers to nodes spelling d-approximate occurrences of x are kept (we say the nodes themselves are (node-)occurrences of x). Let

$$\mathrm{occ}(x, i) = \{y \in F(w) \mid d_H(y, x) = i\}$$

and let

$$\mathrm{occ}_x = \bigcup_{i=1}^{d} \mathrm{occ}(x, i)$$

be the set of occurrences of x. Possibly, some such sets are empty. Let $\ell(x) = \sum_{y \in \mathrm{occ}_x} \ell(y)$. The candidate expression x is processed as long as $\ell(\mathrm{occ}_x) \geq q$. If x has reached the length k, it is output, otherwise its possible extensions are considered. Let xa be its first extension (recall that extensions are attempted in lexicographical order) for $a \in \mathcal{A}$. The occurrences of x belonging to $\mathrm{occ}(x, 0) \cup \mathrm{occ}(x, 1) \cup \mathrm{occ}(x, 2) \ldots \cup \mathrm{occ}(x, d-1)$ are also occurrences of xa. On the other hand, among the occurrences of x that belong to the set $\mathrm{occ}(x, d)$ (their Hamming distance from x is already the maximum allowed), only those followed by a in w may be occurrences of xa. The procedure of extension of a candidate expression is applied recursively. Clearly, if a given candidate expression x does not satisfy the quorum anymore, it is useless to extend it.

A pseudo code for the algorithm speller is given below. It assumes the suffix tree $\mathcal{T}$ of w has already been built. We define:

$$\mathrm{occ}(x, i)_a = \{y \in \mathrm{occ}(x, i) \mid ya \in F(w)\}$$

(it is the subset of $\mathrm{occ}(x, i)$ followed by the letter a).

INITIALIZESPELLER()

$x \leftarrow \varepsilon$
$\triangleright$ by convention, the empty word occurs everywhere in w
$\triangleright$ with Hamming distance 0
$\text{occ}(\varepsilon, 0) \leftarrow \{0, 1, \ldots, |w| - 1\}$
for $i \leftarrow 1$ **to** d **do**
    $\text{occ}(\varepsilon, i) \leftarrow \emptyset$

SPELLER(x, w, q, k, d)

1 **if** $\ell(x) \geq q$ **then**
2   **if** $|x| = k$ **then**
3         $OUTPUT \leftarrow OUTPUT \cup \{x, \text{occ}_x\}$
4   **else for** a in $\mathcal{A}$ **do**
5   **for** $i \leftarrow d$ **downto** 0 **do**
6                 $\text{occ}(xa, i) \leftarrow \text{occ}(x, i)_a$
7   **if** $i \geq 1$ **then**
8                     $\text{occ}(xa, i) \leftarrow \text{occ}(x, i)_a \cup$
                        $(\text{occ}(x, i-1) \setminus \text{occ}(x, i-1)_a)$
9   SPELLER(xa, w, q, k, d)
10 **return** $OUTPUT$

The time complexity of SPELLER is in $O(nV_H(d, k))$ where n is $|w|$ and $V_H(d, k)$ is the size of the set containing all words of length k at Hamming distance d from another of length k. We have that $V_H(d, k) \leq k^d \,\text{Card}(\mathcal{A})^d$. Therefore, SPELLER is linear in the input size, but possibly exponential with respect to d. It has linear space complexity. When $d = 0$, SPELLER has linear (optimal) time complexity.

When the length of the expressions sought is given as a range of values (k_{min}, k_{max}), the algorithm continues extending candidates as long as they do not reach k_{max}. Any candidate expression x having already reached length k_{min} that satisfies the quorum is output.

In the case where SPELLER is extended to handle expressions in $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$, $a \in \mathcal{A}$ in line 4 just needs to be replaced by $S \in \mathcal{S}$ while x and xa in lines 6, 8, and 9 are replaced, respectively, by X and XS.

SPELLER can also be applied to infer expressions in $\mathcal{A}^+$ common to a set of words. As mentioned, a *generalized suffix tree* $\mathcal{GT}$ is used in this case for representing all the suffixes of the input words. When we are dealing with N words, it is not enough anymore to know the value of $\ell(v)$ for each node v in $\mathcal{GT}$ in order to be able to check whether an expression satisfies the quorum. Indeed, for each node v, we need this time to know, not the number of leaves in the subtree of $\mathcal{GT}$ having v as root, but that number for each different word the leaves refer to.

In order to do that, we associate to each node v in $\mathcal{GT}$ a boolean array bit_v of size N, that is defined by:

$$bit_v[i] = \begin{cases} 1, & \text{if at least one leaf in the subtree rooted at } x \\ & \text{represents a suffix of } w_i \\ 0, & \text{otherwise} \end{cases}$$

for $1 \leq i \leq N$.

Let $\ell'(x)$ be the total number of cells set to 1 in the boolean array that results from the OR of bit_v for all nodes v that are occurrences of x in $\mathcal{GT}$. This corresponds to the number of distinct input words where x occurs. The algorithm then changes only in that the condition to be satisfied now is $\ell'(x) \geq q$.

The time complexity in this case is in $O(nN^2V_H(d,k))$ if n is the length of each input word (assuming to simplify that they have same length). The space complexity is $O(nN^2k)$.

5.2.1.3. Mitra-count algorithm

The Mitra-count algorithm proceeds in exactly the same way as Speller except that Mitra-count works directly on the input word(s) and not on an index of the word(s) in the form of a (generalized) suffix tree. The time and space cost of building the suffix tree is thus saved. Another advantage of the approach is that the positions of the occurrences of the expressions can be kept naturally ordered as the expressions are recursively extended. This is also a characteristic of earlier algorithms like Combi and Poivre which work in essentially the same manner as Mitra-count. On the other hand, the inference step is less efficient both in terms of time (if a factor has multiple copies in the input word(s), it will be processed as many times as it has copies) and of space (for the same reason, factors with multiple copies that are occurrences of an expression will need an equal number of pointers to them).

5.2.1.4. Footprinter algorithm

Footprinter has a completely different approach from Speller, or from the other approaches that will be described in this chapter. It can address only the problem of inferring expressions common to a set of words. Unique among all the approaches, it also needs as input a phylogenetic tree besides a set of words. In a simplified way, a phylogenetic tree, that we denote by $\mathcal{F}$, is a tree describing the speciation events that have lead to the species currently observed, or to those having existed in the past. It is a tree with values attached to the edges whose topology represents the evolutionary relations between the species (current or ancestral) and whose nodes correspond to the species. The value of an edge indicates the evolutionary distance separating the species labelling the nodes at the edge's extremities. Each possible set of factors, one taken from each input word, will be considered, that is, placed at the leaves of the input phylogenetic tree. The parsimony score of the tree is then calculated before deciding whether the set, and the expression at the root of the tree, are a solution of the *substring parsimony problem*. The problem is known to be NP-hard.

Only expressions of a single fixed length k are addressed by Footprinter. Extension to a range of length values is not straightforward: in practice, the algorithm has to be run again for each different length required for the output expressions.

The algorithm couples a straightforward dynamic programming technique with the use of a table tab_v containing $\text{Card}(\mathcal{A})^k$ entries for each node v of $\mathcal{F}$, including the leaves. All sets of factors are thus treated together. Each entry x (with $x \in \mathcal{A}^k$) in the table corresponds to one possible word of length k to be assigned to node v, and contains the value of the best parsimony score that can be achieved for the subtree rooted at v, if node v is labeled with x. Denote by $C(v)$ the set of children of node v in $\mathcal{F}$. Then table tab_v can be computed for all nodes v of $\mathcal{F}$ starting from the leaves can be computed by performing the steps indicated in the algorithm FOOTPRINTER below. The quorum is assumed to be N.

FOOTPRINTER$(\mathcal{F}, w_1, w_2, \ldots, w_N, k, d)$
for all nodes $v \in \mathcal{F}$ starting from the leaves **do**
  for all $x \in \mathcal{A}^k$ **do**
    if v is a leaf of $\mathcal{F}$ **then**
      $\triangleright$ let w_v be the input word placed at leaf v in $\mathcal{F}$
      if x is a factor of w_v **then**
        $tab_v[x] \leftarrow 0$
      else $tab_v[x] \leftarrow +\infty$
    else $tab_v[x] \leftarrow \sum_{u \in C(v)} \min_{y \in \mathcal{A}^k}(tab_u[y] + dist_H(x, y))$
return $\{x \in \mathcal{A}^k \mid tab_{root}[x] \leq d\}$

The algorithm has a structure that resembles the structure of the Fitch algorithm for the so-called *small parsimony problem*. It proceeds from the leaves up to the root looking for the optimum at each level up, and then, once the root has been reached from all leaves, goes down the phylogenetic tree again to recover the values at each internal node and leaf that actually produced all optimal parsimony solutions that are below d.

It is possible to use a quorum lower than N, giving rise to the so called *substring parsimony problem with losses*. The basic idea is the following. One assumes the evolution time along the edges of the phylogenetic tree is also known, and the quorum is in this case expressed as a minimum total evolutionary time summed over all edges in the subtree containing as leaves the factors that are occurrences of a same expression. An expression may therefore have less than N occurrences, but the occurrences must then span a "wide-enough" evolutionary time, that is, they must concern organisms that are "distant enough" in terms of evolution.

5.2.2. Inference as a clique detection problem

5.2.2.1. WINNOWER algorithm

The algorithm WINNOWER allows to infer expressions that are collections of words in $\mathcal{A}^k$ for a given positive integer k that is the length of the expression. Like FOOTPRINTER (and unlike SPELLER or similar algorithms which do not use an index), there is no efficient way of handling a range of values for the length of the expressions sought.

The method was elaborated for inferring expressions common to a set of N input words but may easily be adapted to find expressions repeated in a single input word. It can as easily be modified to handle a quorum lower than N although in what follows, the method is presented for a quorum of N only.

Given N input words, $w_1, w_2, \ldots, w_N$ of the same length n, an integer d and a length k, Winnower starts by building the graph $\mathcal{G} = (V_1 \cup \ldots \cup V_N, E)$ as indicated in Section 5.1.2. The graph has $O(nN)$ nodes and $O(n^2N)$ edges.

The goal is then to find all cliques of size N in $\mathcal{G}$, which is an NP-complete problem. The idea of Winnower is to remove edges that cannot belong to cliques. This makes the graph sparse enough that clique detection is easier to perform.

This is achieved by incrementally eliminating what are called *spurious* edges. An edge is spurious if it does not belong to any extendable clique of a given size where by *extendable* clique of size c is meant a clique contained in *all* other possible cliques of size $c+1$. By observing that every edge belonging to a clique of size N also belongs to at least $\binom{N-2}{c-2}$ extendable cliques of size c and through a suitable choice of c, it is possible to eliminate spurious edges. This is recursively done as long as possible. At the end, one expects the graph will contain only cliques of size N, or that at least detecting cliques of size N will have become very easy to do in the graph that remains.

The pseudo-code is not given in this case as the core ideas are those described above. Care with implementation is required for the efficiency of some essential parts of the algorithm but these are not given in enough detail that we feel we can reproduce their essence with perfect fidelity. They are therefore omitted.

The time complexity of Winnower is claimed by the authors to be in $O((nN)^{c+1})$ which is the cost of eliminating spurious edges (for $c = 3$, eliminating spurious edges takes on average $O(N^4 n^{2.66})$ time according to them.) If $d = 0$, Winnower takes exponential time and is therefore, like Footprinter, not optimal.

There are interesting instances with critical values of d that cannot be efficiently handled by Winnower because too few edges can be eliminated and the clique detection step must thus be performed in a dense graph. This is the case in what the authors called *the challenge problem*: for instance, for $k = 15$, $d = 4$, and $\mathrm{Card}(\mathcal{A}) = 4$, it is already not feasible to apply Winnower to an instance as small as 20 words of length 600 each.

5.2.2.2. Mitra-graph algorithm

Mitra-graph is an algorithm that mixes the ideas behind Mitra-count (that is, behind Speller) and Winnower. It thus works within both the star and clique model. The solutions produced are those that would be obtained with Mitra-count for expressions in $\mathcal{A}^k$ with a distance of d and, originally, a quorum of N. Extending it to expressions of a length covering a range of values, or to a quorum less than N is more straightforward and less costly to do than for WINNOWER.

Like WINNOWER, MITRA-GRAPH builds a graph and looks for cliques of size N in it. The big difference is that the graph depends now at each step on the candidate expression currently considered. The graph is thus denoted by $\mathcal{G} = (x, V_1 \cup \ldots \cup V_N, E)$, or $\mathcal{G} = (x, V, E)$ for short. The set of nodes of $\mathcal{G}$ are defined as in WINNOWER. It is in the set of edges that the two graphs differ. For each node v_i, set $v_i = p_i s_i$ with $|p_i| = |x|$ (and $|s_i| = k - |x|$). In MITRA-GRAPH, there is an edge between v_i and v_j if and only if the three inequalities $dist_H(x, p_i) \leq d$, $dist_H(x, p_j) \leq d$, and $dist_H(x, p_i) + dist_H(x, p_j) + dist_H(s_i, s_j) \leq 2d$ hold. The condition of existence of an edge is therefore stronger with MITRA-GRAPH than with WINNOWER. Finding cliques in this graph is also much easier to do than in the graph used by WINNOWER (it basically consists in eliminating all edges that enter nodes with degree less than $N - 1$), while the pruning ideas of both MITRA-COUNT (when an expression does not satisfy the quorum any longer) and WINNOWER allow this in theory to be a more efficient approach than MITRA-COUNT alone.

The algorithm presents an additional cost due to the fact that the graph has to be updated continuously as viable expressions are recursively explored (in lexicographic order like in MITRA-COUNT). The key idea in this case comes from the observation that once expression xy with $y \in \mathcal{A}^+, x \in \mathcal{A}^*$ has been treated, either expression xya with $a \in \mathcal{A}$ will be considered or, if xy did not satisfy the quorum and $y[1], \ldots, y[|y| - 1]$ were all equal to the last letter in the alphabet, it is expression xb with $b \in \mathcal{A}$ and different from the first letter in y (it will, in fact, be the next letter in the alphabet) that will be considered. From the graph $\mathcal{G}(xy, V, E)$, it is easy to obtain $\mathcal{G}(xb, V, E)$ if the values of $dist_H(x, v_i[0..(|x|-1)]$, $dist_H(x, v_j[0..(|x| - 1)]$, $dist_H(v_i[|x|..(k - 1), v_j[|x|..(k - 1)])$ are kept for each edge (v_i, v_j).

As for WINNOWER and for the same reason (not enough detail is presented in the literature indicating how the algorithm is actually implemented), a pseudo-code for MITRA-GRAPH is omitted.

5.3. Inferring network expressions with spacers

5.3.1. Mathematical models and related inference problem

In biology, network expressions with spacers are a first approach to model sequences along a molecule, typically DNA, that function in a cooperative way in the sense that they need to simultaneously bind a same or different molecular complexes so that a given biological process may be initiated. In the case of so-called "higher" organisms, the sites may even come in big clusters. The relative positions along the molecule of the sites that are inside a cluster are in general not random, either because they are recognized by the same complex and cannot therefore stand too much apart, or because they are recognized by different complexes that interact between them. In this last case, the distances between sites along the molecule may be longer but is often quite constrained.

Finally, not all positions within a site are equally important for the binding to happen. In particular for binding sites in proteins where recognition is strongly connected to the 3D structure of the molecule, even a single binding site (single in the sense that it binds a unique site in the other molecule) may concern a sequence of non contiguous positions at variable distances one from another that correspond to amino acids close in 3D space.

Given an alphabet $\mathcal{A}$, a network expression X with spacers is an ordered sequence of simple network expressions $X_1, \ldots, X_p$ with $X_1, \ldots, X_p \in \mathcal{P}(\mathcal{A})^+$ and $p \geq 2$.

The expression X is said to appear exactly in a word w if there exist factors $u_1, \ldots, u_p$ of w such that $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$ and $u_i \in X_i$ for $1 \leq i \leq p$. Given $d = (d_1, \ldots, d_p)$ non negative integers, X is said to appear d-approximately in w if $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$ and, for all $i \in [1, p]$, there exists $v_i \in X_i$ such that $dist_H(u_i, v_i) \leq d_i$.

Finally, given a network expression X with constrained spacers, that is, given a sequence of simple network expressions $X_1, \ldots, X_p$, positive integers $d_1, \ldots, d_p$, and intervals $[min_1, max_1], \ldots, [min_{p-1}, max_{p-1}]$ with $min_i \leq max_i$ non negative integers, X is said to appear approximately in a word w if $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$, u_i a d_i-approximate occurrence of X_i for all $i \in [1, p]$ and $|t_j| \in [min_j, max_j]$ for all $j \in [1, p-1]$. The case of intervals containing negative values may also be considered but has not been treated in the literature.

From now on, network expressions X with constrained spacers and composed of p simple network expressions $X_1, \ldots, X_p$ separated by distances within the intervals $[min_1, max_1], \ldots, [min_{p-1}, max_{p-1}]$ will be denoted by $X = X_1$ $[min_1, max_1]$ $X_2 \ldots X_{p-1}$ $[min_{p-1}, max_{p-1}] X_p$. Network expressions with unconstrained spacers and composed of p simple network expressions $X_1, \ldots, X_p$ will be denoted by $X = X_1 * \ldots * X_p$.

5.3.2. Algorithms

5.3.2.1. Inferring network expressions with constrained spacers

Mitra-dyad **algorithm.** Mitra-dyad infers network expressions with constrained spacers for the case $p = 2$ only, that is expressions of the type $X = X_1[min, max]X_2$. The reason is that the inference is performed in a but containing $O(max - min + 1)$ times more nodes and potentially $O((max - min + 1)^2)$ more edges. Indeed, supposing $|X_1| = |X_2| = k$, each factor u of length k of the input words $w_1, \ldots, w_N$, which corresponded to a node in the original graph, gives now rise to $O(max - min + 1)$ nodes, each one corresponding to the factor u followed by the factor v starting i positions after the end of u, when such position exists, for i between min and max. Nodes are then linked under the same conditions as for Mitra-graph; in particular, the existence of an edge between two nodes remains dependent on the expression $X = X_1[min, max]X_2$ that is being currently considered. Once this graph is built, Mitra-dyad runs Mitra-graph on it. The way the graph is built ensures that the solutions found in this way correspond to the required network expressions with constrained spacers.

Smile **algorithm.** There are in fact two versions of the Smile algorithm. Both versions call the Speller algorithm given in Section 5.2.1 as an internal subroutine. The basic algorithm for a single input word is shown below. It is straightforward to adapt it to the case of N input words. For ease of exposition, we assume that, in the expression $X = X_1[min_1, max_1]X_2 \ldots X_{p-1}[min_{p-1}, max_{p-1}]X_p$, all expressions X_i have the same length k and maximum number of differences allowed d. The way the search space is considered is what makes the main difference between the two versions of Smile. It is worth observing that besides being able to handle a different distance d for each simple expression in X, Smile can handle a global distance, something Mitra-dyad cannot.

The Smile algorithm shown below assumes that $p = 2$ and that the suffix tree $\mathcal{T}$ of w has been previously built. The notations X_1 and X_2 stand for candidate motifs for, respectively, the first and second expressions in the network expression with spacer that is being searched for.

Smile$(w, q, k, d, (min_1, max_1), \ldots, (min_p, max_p))$
1 **for** $i \leftarrow 1$ **to** $p - 1$ **do**
2   **for** each solution of Speller(X_1, w, q, k, d) **do**
3   consider only the search space of all factors of w
4   that start from min_i to max_i positions after
5   occurrences of X_1 in w
6 **return** Speller(X_2, w, q, k, d)

The extension of Smile to the case where $p > 2$ is straightforward. The difference between the two versions of the Smile algorithm are in how lines 3 to 5 are dealt with. We explain it in the simple case where $p = 2$ and $|X_1| = |X_2| = k$. Generalization to different lengths (or range of lengths) for each simple expression in X, or to a general p, is straightforward for the first version and more elaborated for the second. Details may be found in the literature indicated in the notes at the end of the chapter.

The first version of Smile proceeds as follows. For each expression X_1 of length k satisfying the quorum that is obtained, together with its set of node-occurrences in $\mathcal{T}$, that we denote by occ_{X_1}, all simple expressions X_2 are sought. The search starts (using Speller) with the expression $X_2 = \varepsilon$ and occ_{X_2} the set of words v which have an ancestor u in occ_{X_1} with $min \leq level(v) - level(u) \leq max$, where $level(v)$ indicates the length of the label of the path from the root to node v in $\mathcal{T}$. From a node-occurrence u in occ_{X_1}, a jump is therefore made in $\mathcal{T}$ to all potential start node-occurrences v of X_2. These nodes are the min to max-generation descendants of u in $\mathcal{T}$.

The second version of Smile initially proceeds like the first. For each simple expression X_1 inferred, and for each node-occurrence u of X_1 considered in turn, a jump is made in $\mathcal{T}$ down to the descendants of u located at lower levels. This time however, the algorithm just passes through the nodes at these lower levels, grabs some information the nodes contain and jumps back up to level k again. The information grabbed in passing is used to temporarily and partially modify $\mathcal{T}$ and start, *from the root of* $\mathcal{T}$, the inference of all possible companions X_2 for X_1 that are located at the required distance (min, max). Once this operation has ended, the part of $\mathcal{T}$ that was modified is restored to its previous state. The inference of another simple expression X_1 then follows. The whole process unwinds in a recursive way until all expressions X satisfying the initial conditions are inferred.

More precisely, the operation between the spelling of X_1 and X_2 locally alterates $\mathcal{T}$ up to level k into a tree $\mathcal{T}'$ that contains only the prefixes of length k of suffixes of w starting at a position between min and max from the end position in w of an occurrence of X_1. Tree $\mathcal{T}'$ is, in a sense, the union of all the subtrees t of depth at most k rooted at nodes that represent start occurrences of a potential companion X_2 for X_1. SPELLER can then be applied directly to $\mathcal{T}'$. The information that is grabbed in passing is the one required to modify $\mathcal{T}$ into $\mathcal{T}'$: it corresponds to the boolean arrays indicating to which factors of w belong the leaves of all potential end node-occurrences of companions for X_1 in the tree.

The complexity of the first version of SMILE for a single input word is $O(n + n_{2k+max}V_H^2(d,k))$ where n_{2k+max} is the number of nodes at level $2k + max$ in the suffix tree. Its space complexity is $O(n(2k + max))$.

The complexity of the second version of SMILE for a single input word is $O(n + \min\{n_k^2, n_{2k+max}\}V_H^2(d,k) + n_{2k+max}V_H(d,k)))$ and its space complexity $O(n(2k + max) + n_k)$.

5.3.2.2. Inferring network expressions with unconstrained spacers

SMILE **algorithm revisited.** Extensions of the SMILE algorithm enable also to deal with flexible spacers.

The first extension concerns what is called "meta-differences". Given a non negative integer D, a network expression $X = X_1[min_1, max_1]X_2 \ldots X_{p-1}$ $[min_{p-1}, max_{p-1}]X_p$ is said to appear exactly in a word w if there exist factors $u_{j_1}, \ldots, u_{j_q}$ of w such that $p - D \leq q \leq p$ and $w = t_0u_{j_1}t_1 \cdots t_{q-1}u_{j_q}t_q$, with $t_0, \ldots, t_q \in \mathcal{A}^*$, $1 \leq j_1 < \cdots < j_q \leq p$ and $u_{j_i} \in X_{j_i}$ for $i = 1, \ldots, q$. An equivalent definition may be derived for approximate occurrences of X.

The second extension allows SMILE to handle restricted intervals of distances between the simple network expressions $X_1, \ldots, X_p$, exploring in a same run a wide range of possibilities for the middle value of the interval. The expressions that may be inferred in this case are of the type $X = X_1[m_1 \pm \varepsilon_1]X_2$ $\ldots X_{p-1}[m_{p-1} \pm \varepsilon_{p-1}]X_p$ where, for $1 \leq i < p$, ε_i is a non negative integer and $m_i \in [Min_i, Max_i]$ with $Max_i - Min_i$ as large as desirable.

PRATT **algorithm.** Trying to infer network expressions with completely unconstrained spacers would in most situations lead to trivial solutions besides being a computationally harder problem. PRATT therefore imposes some constraints on the amount and distribution of don't care symbols that are allowed.

The expressions treated may however be more flexible than the constrained spacers of SMILE as presented in Section 5.3.2.1. We shall see in a moment the extensions of SMILE which enable to deal with spacers that are as unconstrained as in PRATT, although in a different way.

The constraints that PRATT puts on the spacers are specified as input parameters. Some of the main ones are:

1. a maximum number of spacer regions, that is of regions that are composed of a contiguous sequence of don't care symbols;

2. a maximum length for spacer regions;

3. a maximum number of overall don't care in the expressions sought;

4. a maximum length of the network expression.

Other possible constraints are omitted for the sake of simplicity.

takes in general as input N words, that is, it infers common network expressions, but it can easily be modified to treat the case of a single input word. It works basically like SPELLER for expressions in $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$. Unlike SPELLER, PRATT does not use a suffix tree representation of the input word(s) but a simple queue or file data structure like COMBI or POIVRE. The don't care symbol is treated in a way similar to another element of $\mathcal{S}$, with counters enabling to check whatever spacer constraint was given as input.

The version of SMILE that allows intervals for the distances between single network expressions results in performances analogous to those of PRATT as far as spacers are concerned, although in a slightly different way. SMILE can be more flexible and it further allows for differences in the inference process.

5.4. Related issues

5.4.1. The concept of basis

Given some input word(s), the number of even simpler expressions $X \in \mathcal{A}(\mathcal{A} \cup \bullet)^*\mathcal{A}$ can be exponential with the length of the input, so that it is infeasible to list all of them along with their occurrences in the word(s). Fixing the Hamming distance to 0 or using a high quorum does not avoid the explosive growth in the number of such expressions. Several researchers are working to alleviate this drawback.

Among the many methods proposed to select expressions, one can single out those based on the notion of *maximality* or *specificity*. We assume $d = 0$. Since the expressions may contain don't cares, approximate occurrences are in a certain sense still allowed. Informally, an expression X in $\mathcal{A}(\mathcal{A}\cup\bullet)^*\mathcal{A}$ is maximal if it cannot be extended to the left or to the right by adding further symbols and/or if none of its don't care symbols can be replaced by an alphabet letter, without losing any occurrences. In other words, specifying more a maximal expression causes a loss of information, while this is not true for non-maximal expressions. While the notion of maximality reduces significantly the number of expressions, their number may still be exponential.

A significant step in reducing the number of maximal expressions is the introduction of the notion of *basis*. Informally speaking, a basis is a set of (maximal) expressions that can generate all the (other) maximal expressions by simple mechanical rules. The maximal expressions in the basis are representative of the information content of the words in that they can generate all the other repeated or common expressions. A notion of basis called the set of *tiling motifs* was introduced. It has size linear in the length n of the input word(s) and it is able to generate the repeated expressions (possibly exponential in number) that appear at least twice with don't care symbols in such input over an alphabet $\mathcal{A}$. This basis has some interesting features such as being (a) a subset of previously defined bases, (b) truly linear as its expressions are less than n in number and appear in the word for a total of $2n$ times at most; (c) symmetric as the basis of the reversed word is the reverse of the basis; (d) computable in polynomial time, namely, in $O(n^2 \log n \log \mathrm{Card}(\mathcal{A}))$ time. As an example, the basis of tiling motifs for repeats in the word $w = \mathtt{ATATACTATCAT}$ contains three elements, namely $x_1 = \mathtt{ATA{\bullet}{\bullet}{\bullet}TAT}$, $x_2 = \mathtt{ATAT{\bullet}{\bullet}T}$, and $x_3 = \mathtt{TATA{\bullet}{\bullet}AT}$ that are able to generate (through a suitable operation that takes also into account the positions where the motifs in the basis occur) all other repeated motifs such as $\mathtt{TAT}, \mathtt{TA}, \mathtt{AT}, \mathtt{ATA{\bullet}{\bullet}{\bullet}T}$ etc that appear at least twice in w. For instance, the motif $\mathtt{ATA{\bullet}{\bullet}{\bullet}T}$ can be obtained by the overlap of the occurrences of x_1 and of x_2 at position 0.

A more general and flexible framework is required for repeated or common expressions when $d > 0$ and the notion of a basis may perhaps not be extended in this case. Some fuzzy form of clustering should then be considered.

5.4.2. Inferring tandem network expressions

5.4.2.1. Problem definition

Tandem arrays (called *tandem repeats* when there are only two units) are approximate powers (squares) of a word, that is, a sequence of approximate repeats that appear adjacent in a word. The inference of tandem arrays may proceed in much the same way as for simple expressions appearing repeated a number of times in a word (using for instance Speller or Mitra-count). Checking that the expression appears *tandemly* repeated can then be done *a posteriori*. This however can be a very inefficient approach as many expressions will be generated whose occurrences have no chance of forming a tandem array. It is therefore more interesting to develop a method that allows to check the tandem condition of a repeat as it proceeds with the inference, that is, simultaneously with it. The use of a suffix tree is not interesting when approximate matches are sought because a suffix tree does not allow the positions of the occurrences to be kept ordered for easy processing of the tandem condition. An approach like the one adopted by Mitra-count, that was also used earlier in Combi or Poivre, is the most appropriate in this case.

Before sketching the main ideas of the algorithm, called SATELLITE, we need to introduce the more complex models required by tandem arrays. There are in fact two definitions related to a tandem array model, one called *prefix model* and the other *consensus model*. This latter concerns tandem array models strictly speaking while prefix models are in fact models for approximately periodic repeats that are not necessarily (yet) tandem. They correspond to the prefixes of a consensus model.

Formally, a *prefix model* of a tandem array is a word $x \in \mathcal{A}^+$ (x could also belong to $\mathcal{P}(\mathcal{A})^+$) that approximately matches a *train of wagons*. A *wagon* of x is a factor u in w such that $dist_E(x, u) \leq d$ for d a non negative integer (observe that in this case, it is the edit distance that has been considered). A *train* of a prefix model x is a collection of wagons $u_1, u_2, \ldots, u_p$ ordered by their starting positions in w and satisfying the following properties:

(P_1) $p \geq q$ where q is again a quorum indicating this time the minimum number of units the sought tandem arrays must have;

(P_2) $left_{u_{i+1}} - left_{u_i} \in [min\_period, max\_period]$ is the position of the left end of wagon u in w and min_period, max_period are the minimum and maximum period of the repeat.

A consensus model must further satisfy the following property:

(P_3) $left_{u_{i+1}} - right_{u_i} = 0$

where $right_u$ is the position of the right-end of wagon u. The property checks that the occurrences of consensus models are indeed tandem. This is verified only when $|x| \in [min\_period, max\_period]$, that is when the length of the repeat has reached the value specified as input.

The tandem array inference problem is then the following.

Inference of tandem array problem

INPUT: A word $w \in \mathcal{A}^+$, a quorum q, an edit distance d and a minimum and maximum period min_period and max_period.

OUTPUT: All expressions $x \in \mathcal{A}^+$ that are consensus models for tandem arrays (that is, properties (P_1), (P_2) and (P_3) are satisfied).

5.4.2.2. SATELLITE algorithm

Expressions for tandem arrays are inferred by increasing length. The algorithm keeps track of individual wagons, and at each step determines, on the fly, if they can be combined into at least one train (observe that a wagon can belong to more than one train). The latter corresponds to checking, for each wagon, whether it belongs to at least one set of wagons satisfying properties (P_1) and (P_2/P_3) above.

For each expression x that is a prefix model for a tandem array, a list of the wagons of x that belong to at least one train of x is kept. When the expression x is extended into the expression $x' = xa$, two tasks must be performed:

1. determine which wagons of x can be extended to become wagons of x';

2. among these newly determined wagons of x', keep only those that belong to at least a train of x'. This requires effectively assembling wagons into trains.

The trains do not need to be enumerated in the second step, it must only be determined if a wagon is part of one. This allows to perform an extension step in time linear with the length of the input word.

Consider the directed graph $G = (V, E)$ where V is the set of all wagons of x and there is an edge from wagon u to wagon v if $\mathit{left}_v - \mathit{left}_u \in [\mathit{min\_period}, \mathit{max\_period}]$. A wagon u is then part of a train if it is in a path of length q or more in G. Determining this is quite simple as the graph is clearly acyclic.

For each expression x of length between *min_period* and *max_period*, it remains to check whether x satisfies the properties of a consensus model for a tandem array. Consider now a directed bipartite graph $G_x = (L_x \cup R_x, E)$ whose vertices are the positions at which, respectively, the left- and right-ends of wagons of x occur. Edges $i \to j$ with $i \in L_x, j \in R_x$ are *wagon edges* and edges $j \to i$ with $i \in L_x, j \in R_x$ are *gap edges*. There is a wagon edge $i \to j$ if and only if $w[i..j-1]$ is a wagon, and there is a gap edge $j \to i$ if and only if $i = j$. Thus, there is an edge sequence $i \to j \to k$ occurs in G_x if and only if there are wagons u and v such that $u = w_i w_{i+1} \cdots w_{j-1}$, $\mathit{left}_v = k$, and $\mathit{left}_v - \mathit{right}_u = 0$. It follows that a position/node which is on a path of length $2q$ or more is part of a train satisfying properties (P_1), (P_2) and (P_3). Such a position is called a *final position* or *final node*. Let G'_x be the graph induced by the set, F_x, of all final nodes. If G'_x is non-empty, then x is a consensus model for a tandem array having the characteristics specified in the input.

The complexity of SATELLITE is $O(n\ \mathit{max\_period}\ M_E(d,k))$ where n is, as before, $|w|$ and $M_E(d,k)$ is the size of the set containing all words of length k at edit distance d from another word of length k. This is actually the complexity of SPELLER multiplied by the term *max_period* because of the need to check for the tandem condition. An extended version of SATELLITE allows to deal with tandem arrays that may miss a period, meaning that the repeat may contain some units that have accumulated more differences than allowed. Such units are called *bad wagons*. A number of them may be authorized in a train.

5.5. Open problems

5.5.1. Inference of network expressions using edit distance

In theory, all algorithms presented in this chapter may be modified to handle edit instead of Hamming distance. Indeed, edit distance is already an integral part of POIVRE (and of SATELLITE). Thus MITRA-COUNT which behaves much as POIVRE can easily be extended to use an edit distance. The same is true of SPELLER and such a modification was suggested and quickly sketched by the authors. A more recent approach using a suffix tree like SPELLER introduces what appears to be an algorithm producing a different solution from SPELLER given the same instance.

There has been also a theoretical discussion on how to introduce edit distance into FOOTPRINTER which includes the time complexity that the resulting algorithm would have. However, FOOTPRINTER is not suitable for dealing with expressions or occurrences of variable length which appear when insertions and deletions are allowed. The reason comes from the data structure used (the table at each node of the tree).

WINNOWER could also theoretically handle an edit distance, but the number of edges in the graph would grow as would the number of spurious edges. MITRA-GRAPH would have the same type of problem but the filtering of spurious edges is easier to perform and therefore the algorithm might be able to handle the situation a lot better than WINNOWER.

Finally, the first version of SMILE is, like SPELLER, easily modifiable to handle an edit distance. Although theoretically not impossible, introducing such distance into the second version of SMILE might be more tricky.

In all cases, it is worth exploring more compact ways of representing the occurrences of an expression once insertions and deletions are allowed. One possible way extends ideas for pattern matching in a long text with edit distance.

5.5.2. Minimal covering set

The concept of *minimal covering set* of expressions may enable to address two difficulties encountered by currently existing combinatorial algorithms for network expression inference in a set of words. These difficulties are, first how to fix *a priori* the quorum, and second (an even harder problem) how to efficiently identify weak and rare expressions? To solve the second problem one can increase the value of d while simultaneously decreasing the value of q. However, this may lead to a huge number of solutions, many of which are uninteresting. A minimal covering set extends the concept of individual expressions, with or without spacers, to that of a family of expressions which "completely explains" a set of words. In a more precise way, the problem could be expressed in the following (informal) way. Given a set of input words, one must find a minimal set of $r \geq 1$ expression(s) (the value of r is unknown at start) such that:

- each expression has an occurrence in at least one input word;
- distinct expressions among the r may have occurrences in the same input word but the number of times this may happen is smaller than a threshold value t (possibly t may be 0: there is no "word overlap" of the expressions);
- all words are covered by (at least) one expression in the family (strictly one in case the threshold t is 0).

Notes

The term *network expression* to denote repeated regular expressions without the Kleene closure was introduced for the first time in (Mehldau and Myers 1993).

The literature on grammatical inference is large. Three papers have been influential on the theory of learning grammars. The first by Gold (Gold 1967) introduced the notion of "language identification in the limit", the second by Wharton (Wharton 1974) relaxed the condition of an exact identification by allowing for various descriptions of the correct solution, while the third by Valiant (Valiant 1984) relaxed such condition by allowing for a solution to be only approximately correct. The earliest and main expository of inference problems for regular grammars is Angluin (Angluin 1982, 1987).

The definitions and statements concerning the star model have been adopted in several exact algorithms, namely COMBI (Sagot and Viari 1996), POIVRE (Sagot, Viari, and Soldano 1997), SPELLER (Sagot 1998), SMILE (Marsan and Sagot 2000b, 2001), PRATT (Jonassen, Collins, and Higgins 1995), and MITRA-COUNT (Eskin and Pevzner 2002). The uniform property variant of the star model has been introduced in (Sagot 1996) and a similar idea in a previous paper (Sagot, Soldano, and Viari 1995).

The closest substring problem has been defined and proved to be NP-complete in (Fellows, Gramm, and Niedermeier 2002). It remains an open problem whether it is parameter-tractable for constant size alphabet when either d alone or d and N are fixed (Fellows et al. 2002).

The definitions of the *substring parsimony problem* and the proof of its NP-hardness are given in (Blanchette, Schwikowski, and Tompa 2000). The *small parsimony problem* was introduced in (Fitch 1975) and the *substring parsimony problem with losses* in (Blanchette 2001, Blanchette, Schwikowski, and Tompa 2002). The FOOTPRINTER algorithm was presented and analyzed in (Blanchette et al. 2000, Blanchette 2001, Blanchette and Tompa 2002, Blanchette et al. 2002). The time complexity of FOOTPRINTER is $O(Nk\,\mathrm{Card}(\mathcal{A})^k + nNk)$ where n is the length of each input word (assuming they have same length). The highest term was in fact $Nk\,\mathrm{Card}(\mathcal{A})^k$ in a first paper (Blanchette et al. 2000). This came from the fact that the computation of the Hamming distance between two words of length k (which takes $O(k)$ time) is done for each of the $O(N)$ edges in $\mathcal{F}$, each of the $O(\mathrm{Card}(\mathcal{A})^k)$ possible values for x and each of the $O(\mathrm{Card}(\mathcal{A})^k)$ possible values for y. In (Blanchette 2001), an improvement of the original algorithm described in (Blanchette et al. 2000) was introduced which enabled to get the exponent k instead of $2k$. The improvement is achieved by means of an auxiliary table for each edge in $\mathcal{F}$. Details may be found in (Blanchette 2001). FOOTPRINTER is thus linear with the size of the input words but exponential with the length k of the expressions sought. If $d = 0$, FOOTPRINTER still takes exponential time and is therefore not optimal. The algorithm WINNOWER was described in (Pevzner and Sze 2000). One must observe that if a quorum lower than N is used, the size of the cliques sought is the only thing that changes in WINNOWER. In practice however, the smaller the quorum, the less spurious edges there will be that can be safely eliminated. Winnower's descendant, , was presented in (Eskin and Pevzner 2002, Eskin, Gelfand, and Pevzner 2003). The two algorithms that are similar to Winnower and Mitra-graph, namely KMRC and Poivre, appeared in (Sagot, Viari, Pothier, and Soldano 1995, Soldano, Viari, and Champesme 1995).

Informations concerning the Profile data base can be found in (Buhler and Tompa 2001).

Some examples of the use of relative entropy for the evaluation of network expressions can be found in (Vanet, Marsan, and Sagot 1999, Pavesi, Mauri, and Pesole 2001b).

The algorithm Speller was introduced in (Sagot 1998), while the two variants it inspired, Mitra-count and Weeder, where described in, respectively, (Eskin and Pevzner 2002) and (Pavesi, Mauri, and Pesole 2001a). Detailed information about the generalized suffix tree data structure can be found in (Bieganski, Riedl, Carlis, and Retzel 1994, Hui 1992). A recent approach using a suffix tree as in Speller but working with the edit distance is given in (Adebiyi, Jiang, and Kaufmann 2001, Adebiyi and Kaufmann 2002). The approach seems to produce a different solution from the one that would result from an application of an extension of Speller enabling to work with the edit distance.

Concerning algorithms for inferring network expressions with constrained spacers, the various versions of the Smile algorithm are described in (Marsan and Sagot 2000b, 2001), and Mitra-dyad is presented in (Eskin and Pevzner 2002, Eskin et al. 2003).

For the case of unconstrained spacers, Pratt is introduced in (Brazma, Jonassen, Vilo, and Ukkonen 1998c, 1998b, Brazma, Jonassen, Eidhammer, and Gilbert 1998a, Jonassen et al. 1995). A few years after Pratt was conceived, an algorithm that is roughly equivalent to Pratt in terms of its output was elaborated which uses a lazy implementation of the suffix tree (Giegerich, Kurtz, and Stoye 1999) to represent the patterns as these are produced. The lazy suffix tree construction as adapted by the authors to their needs takes quadratic time but is claimed to be efficient in most practical situations (Brazma et al. 1998c).

The notion of basis of repeated motifs was introduced in (Parida, Rigoutsos, Floratos, Platt, and Gao 2000, Parida, Rigoutsos, and Platt 2001). The more recent notion of tiling motifs was described in (Pisanti, Crochemore, Grossi, and Sagot 2003).

Finally, the Satellite algorithm for inferring tandem network expressions can be found in (Sagot and Myers 1998).

Readers interested in approaches to the inference, in biological applications, of simple network expressions or of network expressions with spacers using heuristics or statistical methods may consult (Durbin, Eddy, Krogh, and Mitchison 1998), (Pevzner 2000) and (Waterman 1995). Machine learning techniques have also long been in use for inferring patterns or grammars. References to some of these techniques as applies to biology may be found in (Baldi and Brunak 1998).

CHAPTER 6

Statistics on Words with Applications to Biological Sequences

6.0. Introduction

Statistical and probabilistic properties of words in sequences have been of considerable interest in many fields, such as coding theory and reliability theory, and most recently in the analysis of biological sequences. The latter will serve as the key example in this chapter. We only consider finite words.

Two main aspects of word occurrences in biological sequences are: where do they occur and how many times do they occur? An important problem, for instance, was to determine the statistical significance of a word frequency in a DNA sequence. The naive idea is the following: a word may be significantly rare in a DNA sequence because it disrupts replication or gene expression, (perhaps a negative selection factor), whereas a significantly frequent word may have a fundamental activity with regard to genome stability. Well-known examples of words with exceptional frequencies in DNA sequences are certain biological palindromes corresponding to restriction sites avoided for instance in *E. coli*, and the Cross-over Hotspot Instigator sites in several bacteria. Identifying over- and under-represented words in a particular genome is a very common task in genome analysis.

Statistical methods to study the distribution of the word locations along a sequence and word frequencies have also been an active field of research; the goal of this chapter is to provide an overview of the state of this research.

Because DNA sequences are long, asymptotic distributions were proposed first. Exact distributions exist now, motivated by the analysis of genes and protein sequences. Unfortunately, exact results are not adapted in practice for long sequences because of heavy numerical calculation, but they allow the user to assess the quality of the stochastic approximations when no approximation error can be provided. For example, BLAST is probably the best-known algorithm for DNA matching, and it relies on a Poisson approximation. Approximate p-values can be given; yet the applicability of the Poisson approximation needs to be justified.

Statistical properties of words only make sense with respect to some underlying probability model. DNA sequences are commonly modeled as stationary random sequences. Typical models are homogeneous m-order Markov chains (model Mm) in which the probability of occurrence of a letter at a given position depends only on the m previous letters in the sequence (and not on the position); the independent case is a particular case with $m = 0$. Hidden Markov models (HMMs) reveal however that the composition of a DNA sequence may vary over the sequence. However, no statistical properties of words have been yet derived in such heterogeneous models. DNA sequences code for amino acid sequences (proteins) by non-overlapping triplets called *codons*. The three positions of the codons have distinct statistical properties, so that for coding DNA we naturally think of three sequences where the successive letters come from the three codon positions, respectively. The three chains and their transition matrices are denoted as Mm-3. In this chapter, we will focus on the homogeneous models Mm and give existing results for Mm-3.

Because these probabilistic models have to be fitted to the observed biological sequence, we will pay attention to the influence of the model parameter estimation on the statistical results. Some asymptotic results take care of this problem but the exact results require that the true model driving the observed sequence is known.

The choice of the Markov model order depends on the sequence length, because of the data requirements in estimation. One might be able to test hierarchical models using Chi-square tests to assign which order of Markovian dependence is appropriate for the underlying sequence. From a practical point of view, it also depends on the composition of the biological sequence one wants to take into account. Indeed, if the sequence was generated from an m-order Markov chain, then the model Mm provide a good prediction for the $(m + 1)$-letter words.

In this chapter, we are concerned firstly with the occurrences of a single pattern in a sequence. To begin, we discuss the underlying probabilistic models (Section 6.1). The main complication for word occurrences arises from overlaps of words. One might be interested either in overlapping occurrences or in particular non-overlapping ones (Section 6.2). After presenting results for the statistical distribution of word locations along the sequence (Section 6.3), we focus on the distribution of the number of overlapping occurrences (Section 6.4) and the number of renewals (Section 6.5). In Section 6.6, we will study the occurrences of multiple patterns. Section 6.7 gives two applications on how probabilistic and statistical considerations come into play for DNA sequence analysis. Firstly, we look for words with unexpected counts in some DNA sequences. The focus will be on the importance of the order m of the Markov model used and on the interest of using a model of the type Mm-3 (with three transition matrices), when analyzing a coding DNA sequence. We will also take the opportunity to compare exact and asymptotic results on the word count distributions. Secondly, we describe how to analyze so-called SBH chips, a fast and effective method for determining a DNA sequence. These chips provide the ℓ-tuple contents of a DNA sequence, where typically $\ell = 8, 10$ or 12. A non-trivial combinatorial problem arises when determining the probability that a randomly chosen DNA sequence can be uniquely reconstructed from its ℓ-tuple contents. Finally, Section 6.8, meant as an appendix, gives a compilation of more general techniques that are applied in this chapter.

Due to the abundance of literature, the present chapter has no intention of being a complete literature survey (indeed even just a list of references would take up all the space designated to this chapter), but rather to introduce the reader to the major aspects of this field, to provide some techniques and to warn of major pitfalls associated with the analysis of words. For the same reason we completely omit the algorithmic aspect.

6.1. Probabilistic models for biological sequences

In this chapter, a biological sequence is either a DNA sequence or a protein sequence, that is, a finite sequence of letters either in the 4-letter DNA alphabet $\{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$ or the 20-letter amino-acid alphabet. To model a biological sequence, we will consider models for random sequences of letters. Even if we observed a finite biological sequence $\underline{S} = s_1 s_2 \cdots s_n$, we consider for convenience in the whole chapter an infinite random sequence $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ on a finite alphabet $\mathcal{A}$, where $\mathbb{Z}$ is the set of integers. We present below two classes of Markov models widely used to analyze biological sequences and how to estimate their parameters according to the observed sequence. Then we give a classical Chi-square test to choose the appropriate order of the Markov model for a given sequence.

However, we will see in Section 6.7.1 that the choice of the model also has to take biological considerations on the sequence composition into account.

6.1.1. Markovian models for random sequences of letters

The simplest model assumes that the letters X_i are independent and take on the value $a \in \mathcal{A}$ with probability $\mu(a) = 1/\operatorname{Card}(\mathcal{A})$, where $\operatorname{Card}(\mathcal{A})$ denotes the size of the alphabet. To refine this model, we can simply assume independent letters taking values in $\mathcal{A}$ with probabilities $(\mu(a))_{a\in\mathcal{A}}$ such that $\sum_{a\in\mathcal{A}} \mu(a) = 1$. This is called model M0. Typically for DNA sequences, this model is not very accurate. Therefore, we consider a much more general homogeneous model, the model Mm: an ergodic stationary m-order Markov chain on a finite alphabet $\mathcal{A}$ with transition matrix $\Pi = (\pi(a_1 \cdots a_m, a_{m+1}))_{a_1,\ldots,a_{m+1}\in\mathcal{A}}$ such that

$$\pi(a_1 \cdots a_m, a_{m+1}) = \mathbf{P}(X_i = a_{m+1} \mid X_{i-1} = a_m, \ldots, X_{i-m} = a_1).$$

In general, a stationary distribution μ of an ergodic stationary Markov chain with transition matrix Π is defined as a solution of $\mu = \mu\Pi$. This implies that the above Markov chain has a unique stationary distribution μ on $\mathcal{A}^m$ defined by

$$\mu(a_1 \cdots a_m) = \mathbf{P}(X_i \cdots X_{i+m-1} = a_1 \cdots a_m), \quad \forall i \in \mathbb{Z}$$

such that the equation

$$\mu(a_1 \cdots a_m) = \sum_{b\in\mathcal{A}} \mu(b a_1 \cdots a_{m-1}) \pi(b a_1 \cdots a_{m-1}, a_m)$$

is satisfied for all $(a_1 \cdots a_m) \in \mathcal{A}^m$. The model where the letters $\{X_i\}_{i\in\mathbb{Z}}$ are chosen independently with probabilities $p_1, p_2, \ldots, p_{|\mathcal{A}|}$ corresponds to the transition matrix Π with identical rows $(p_1\ p_2\ \cdots\ p_{|\mathcal{A}|})$ and stationary distribution $\mu = (p_1, p_2, \ldots, p_{|\mathcal{A}|})$.

A coding DNA sequence is naturally read as successive non-overlapping 3-letter words called codons. These codons are then translated into amino acids via the genetic code to produce a protein sequence. Several different codons can code for the same amino acid, and often the first two letters of a codon suffice to determine the corresponding amino acid. Therefore, letters may have different importance depending on their position with respect to the codon partition. To distinguish the letter probabilities according to their position modulo 3 in the coding DNA sequence, we consider a stationary Markov chain with three distinct transition matrices Π_1, Π_2 and Π_3 such that, for $a_1, \ldots, a_{m+1} \in \mathcal{A}$ and $k \in \{1, 2, 3\}$

$$\pi_k(a_1 \cdots a_m, a_{m+1}) = \mathbf{P}(X_{3j+k} = a_{m+1} \mid X_{3j+k-1} = a_m, \ldots, X_{3j+k-m} = a_1).$$

This is model Mm-3. The index $k \in \{1, 2, 3\}$ is called *phase* and represents the position of a letter inside a codon. By convention, the phase of a word is the phase of its last letter in the sequence; codons are then 3-letter words in phase 3.
The stationary distribution μ on $\mathcal{A}^m \times \{1, 2, 3\}$ is given by

$$\mu(a_1 \cdots a_m, k) = \mathbf{P}(X_{3j+k-m+1} \cdots X_{3j+k} = a_1 \cdots a_m), \quad \forall j \in \mathbb{Z}$$

such that the equation

$$\mu(a_1 \cdots a_m, k) = \sum_{b\in\mathcal{A}} \mu(ba_1 \cdots a_{m-1}, k-1)\pi_k(ba_1 \cdots a_{m-1}, a_m)$$

is satisfied for all $(a_1 \cdots a_m, k) \in \mathcal{A}^m \times \{1, 2, 3\}$.

Some general results for Markov chains will be used in the exposition. For simplicity we concentrate here on the case of a 1-order Markov chain.

The stationary distribution of a Markov chain can be obtained from its transition matrix. For a 1-order Markov chain we diagonalize the transition matrix as follows. Let $(\alpha_t)_{t=1,\ldots,|\mathcal{A}|}$ be the eigenvalues of Π such that $|\alpha_1| \geq |\alpha_2| \geq \cdots \geq |\alpha_{|\mathcal{A}|}|$. The Perron–Frobenius Theorem ensures that $\alpha_1 = 1$ and $|\alpha_2| < 1$; we abbreviate

$$\alpha := \alpha_2. \tag{6.1.1}$$

Then $(1, 1, \ldots, 1)^T$ is a right eigenvector of Π for the eigenvalue 1 whereas the vector of stationary distribution $(\mu(a), a \in \mathcal{A})$ is a left-eigenvector of Π for the eigenvalue 1. Let $D = \mathrm{Diag}(1, \alpha, \alpha_3, \cdots, \alpha_{|\mathcal{A}|})$. We decompose $\Pi = PDP^{-1}$ such that the first column of P is $(1, 1, \ldots, 1)^T$; then the first row of P^{-1} is the vector of stationary distribution $(\mu(a), a \in \mathcal{A})$. For all $t \in \{1, \ldots, |\mathcal{A}|\}$,

I_t denotes the $|\mathcal{A}| \times |\mathcal{A}|$ matrix such that all its entries are equal to 0 except $I_t(t,t) = 1$, and we define

$$Q_t := P I_t P^{-1}. \tag{6.1.2}$$

We shall use the following decomposition of the h-step transition matrix Π^h

$$\Pi^h = P D^h P^{-1} = \sum_{t=1}^{|\mathcal{A}|} \alpha_t^h Q_t \tag{6.1.3}$$

and that

$$Q_1(a,b) = \mu(b),\ \forall a, b \in \mathcal{A}. \tag{6.1.4}$$

In the exposition, we shall also refer to the *reversed* Markov chain, for a 1-order chain. Its h-step transition probabilities are given by

$$\pi_R^{(h)}(b,a) = \frac{\mu(a)\pi^{(h)}(a,b)}{\mu(b)}.$$

where the $\big(\pi^{(h)}(a,b)\big)$'s are the h-step transition probabilities for the chain itself. Another useful quantity is

$$\rho = 1 - \min\left\{\sum_{b\in\mathcal{A}} \min_{a\in\mathcal{A}} \pi(a,b), \sum_{b\in\mathcal{A}} \min_{a\in\mathcal{A}} \pi_R(a,b)\right\}. \tag{6.1.5}$$

These quantities can easily be generalized to m-order Markov chains, using the following embedding. Let us now assume that the sequence $(X_i)_{i\in\mathbb{Z}}$ is a m-order Markov chain on the alphabet $\mathcal{A}$, with transition probabilities $\pi(a_1 \cdots a_m, a_{m+1})$, $a_1, \cdots, a_{m+1} \in \mathcal{A}$. Rewrite the sequence over the alphabet $\mathcal{A}^m$ by defining

$$\mathbb{X}_i = X_i X_{i+1} \cdots X_{i+m-1}, \tag{6.1.6}$$

so that the sequence $(\mathbb{X}_i)_{i\in\mathbb{Z}}$ is a first-order Markov chain on $\mathcal{A}^m$ with transition probabilities, for $\mathbb{A} = a_1 \cdots a_m \in \mathcal{A}^m$ and $\mathbb{B} = b_1 \cdots b_m \in \mathcal{A}^m$,

$$\Pi(\mathbb{A},\mathbb{B}) = \begin{cases} \pi(a_1 \cdots a_m, b_m) & \text{if } a_2 \cdots a_m = b_1 \cdots b_{m-1} \\ 0 & \text{otherwise.} \end{cases}$$

6.1.2. Estimation of the model parameters

Modeling a biological sequence consists of choosing a probabilistic model (see previous paragraph) and then estimating the model parameters according to the unique realization that is the biological sequence. In the case of model Mm, it means to estimate the transition probabilities $\pi(a_1 \cdots a_m, a_{m+1})$; their estimators are classically denoted by $\widehat{\pi}(a_1 \cdots a_m, a_{m+1})$.

We now derive the estimators that maximize the likelihood of the M1 model given the observed sequence; we will then give the maximum-likelihood estimators in models Mm and Mm-3.

Assume $X_1 \cdots X_n$ is a stationary Markov chain on $\mathcal{A}$ with transition matrix $\Pi = (\pi(a,b))_{a,b\in\mathcal{A}}$ and stationary distribution $(\mu(a))_{a\in\mathcal{A}}$. The likelihood L of the model is

$$L(\pi(a,b), a, b \in \mathcal{A}) = \mu(X_1) \prod_{a,b\in\mathcal{A}} (\pi(a,b))^{N(ab)}$$

where $N(ab)$ denotes the number of occurrences of the 2-letter word ab in the random sequence $X_1 \cdots X_n$. To find the transition probabilities that maximize the likelihood, one maximizes the log likelihood

$$\log L(\pi(a,b), a, b \in \mathcal{A}) = \log \mu(X_1) + \sum_{a,b\in\mathcal{A}} N(ab) \log \pi(a,b).$$

One can separately maximize $\sum_{b\in\mathcal{A}} N(ab) \log \pi(a,b)$ for $a \in \mathcal{A}$, keeping in mind that $\sum_{b\in\mathcal{A}} \pi(a,b) = 1$. Let $a \in \mathcal{A}$ and choose $c \in \mathcal{A}$; we have

$$\sum_{b\in\mathcal{A}} N(ab) \log \pi(a,b) = \sum_{b\neq c} N(ab) \log \pi(a,b) + N(ac) \log \left(1 - \sum_{b\neq c} \pi(a,b)\right)$$

and for $b \neq c$

$$\frac{\partial}{\partial \pi(a,b)} \left(\sum_{b\in\mathcal{A}} N(ab) \log \pi(a,b)\right) = \frac{N(ab)}{\pi(a,b)} - \frac{N(ac)}{\pi(a,c)}.$$

All the partial derivatives equal to zero means that

$$\frac{N(ab)}{\pi(a,b)} = \frac{N(ac)}{\pi(a,c)} \quad \forall b \in \mathcal{A};$$

this implies in particular that

$$\frac{N(ab)}{\pi(a,b)} = \frac{\sum_{d\in\mathcal{A}} N(ad)}{\sum_{d\in\mathcal{A}} \pi(a,d)} = \sum_{d\in\mathcal{A}} N(ad) := N(a\bullet) \quad \forall b \in \mathcal{A}.$$

It follows that

$$\widehat{\pi}(a,b) = \frac{N(ab)}{N(a\bullet)} \quad \forall b \in \mathcal{A}.$$

Note that the second partial derivatives of the likelihood function are negative, assuring that we have indeed determined a maximum.

Remark 6.1.1. For notational convenience, the estimators mainly used in the remainder of the chapter will be $\widehat{\pi}(a,b) = N(ab)/N(a)$ since $N(a\bullet) = N(a)$ except for the last letter of the sequence for which the counts differ by 1.

It is important to note that the estimators $\widehat{\pi}(a,b)$ are random variables. Assuming that the biological sequence is a realization of the random sequence, one can calculate a numerical value for the estimator of $\pi(a,b)$; that is

$$\widehat{\pi}^{\mathrm{obs}}(a,b) = \frac{N^{\mathrm{obs}}(ab)}{N^{\mathrm{obs}}(a\bullet)},$$

where $N^{\mathrm{obs}}(\cdot)$ denotes the observed count in the biological sequence. As we will see, some results are obtained assuming that the true parameters $\pi(a,b)$ are known and equal, in practice, to $N^{\mathrm{obs}}(ab)/N^{\mathrm{obs}}(a\bullet)$, and do not take care of the estimation. It is indeed a common practice to substitute the estimator for the corresponding parameter in distributional results, but sometimes it changes the distribution being studied, as we will see later.

In the model Mm, the maximum-likelihood estimator of $\pi(a_1 \cdots a_m, a_{m+1})$, $a_1, \ldots, a_{m+1} \in \mathcal{A}$, is

$$\widehat{\pi}(a_1 \cdots a_m, a_{m+1}) = \frac{N(a_1 \cdots a_m a_{m+1})}{N(a_1 \cdots a_m \bullet)},$$

and in model Mm-3, we have $\forall a_1, \ldots, a_{m+1} \in \mathcal{A}$, $\forall k \in \{1,2,3\}$,

$$\widehat{\pi}_k(a_1 \cdots a_m, a_{m+1}) = \frac{N(a_1 \cdots a_m a_{m+1}, k)}{\displaystyle\sum_{b \in \mathcal{A}} N(a_1 \cdots a_m b, k)}.$$

6.1.3. Test for the appropriate order of the Markov model

To test which Markov model would be appropriate for a given sequence of length n, the most straightforward test is a Chi-square test, which can be viewed as a generalized likelihood ratio test. Most well-known is the Chi-square test for independence.

Suppose we have a sample of size n cross-classified in a table with U rows and V columns. For instance, we could have four rows labeled $\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}$, and four columns labeled $\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}$, and we count how often a letter from the row is followed by a letter from the column in the sequence. First we test whether we may assume the sequence to consist of independent letters. To this purpose, recall that $N(ab)$ denotes the count in cell (a,b), whereas $N(a\bullet)$ is the ath row count, and let $N(\bullet b)$ is the bth column count. Thus $N(ab)$ counts how often letter a is followed by letter b in the sequence. Let $\pi(a,b)$ be the probability of cell (a,b), let $\pi(a,\bullet)$ be the ath row marginal probability, and let $\pi(\bullet,b)$ be the bth column marginal probability. We test the null hypothesis of independence

$$H_0 : \pi(a,b) = \pi(\bullet, b)$$

against the alternative that the $\pi(a,b)$'s are free. Under H_0, the maximum-likelihood estimate of $\pi(a,b)$ is

$$\hat{\pi}(a,b) = \hat{\pi}(\bullet,b) = \frac{N(\bullet b)}{n-1}.$$

The Pearson chi-square statistic is the sum of the square difference between observed and estimated expected counts, divided by the estimated expected count, where expectations are taken assuming that the null hypothesis is true. Thus, under H_0, for the count $N(ab)$ we expect $(n-1)\mu(a)\pi(\bullet, b)$, estimated by $N(a\bullet)\hat{\pi}(\bullet, b)$, and the chi-square statistic is

$$\chi^2 = \sum_{a\in\mathcal{A}}\sum_{b\in\mathcal{A}} \frac{(N(ab) - N(a\bullet)N(\bullet b)/(n-1))^2}{N(a\bullet)N(\bullet b)/(n-1)}.$$

Under the null hypothesis, χ^2 follows asymptotically a chi-square distribution with $(\mathrm{Card}(\mathcal{A})-1)^2$ degrees of freedom. Thus we would reject the null hypothesis when χ^2 is too large, compared to the corresponding chi-square distribution. As a rule of thumb, this test is applicable when the expected count in each row and column is at least 5. Applying this test to DNA counts, we thus would have to compare χ^2 to a chi-square distribution with $(4-1)^2 = 9$ degrees of freedom. A typical cutoff level would be 5%, or, if one would like to be conservative, 1%. The corresponding critical values are 16.92 for 5 %, and 21.67 for 1 %. Thus, if $\chi^2 > 16.92$, we would reject the null hypothesis of independence at the 5 % level (meaning that, if we repeated this experiment many times, in about 5% of the cases we would reject the null hypothesis when it is true). If $\chi^2 > 21.67$, we could reject the null-hypothesis at the 1 % level (so in only about 1 % of all trials would we reject the null hypothesis when it is true). Otherwise we would not reject the null hypothesis.

If the null hypothesis of independence cannot be rejected at an appropriate level (say, 5 %), then one would fit an independent model. However, if the null hypothesis is rejected, one would test for a higher-order dependence. The next step would thus be to test for a first-order Markov chain. We describe here the general case.

Suppose we know that our data come from a Markov chain of order at most m. Let $N(a_1a_2\ldots a_{m+1})$ be the count of the vector $(a_1, a_2, \ldots, a_{m+1})$ in the sequence $(X_1, \ldots, X_n)$, let $N(a_1a_2\ldots a_m\bullet)$ be the count of the vector $(a_1, a_2, \ldots, a_m)$ in the sequence $(X_1, \ldots, X_{n-1})$, let $N(\bullet a_{m-r+1}\ldots a_m\bullet)$ be the count of the vector $(a_{m-r+1}, \ldots, a_m)$ in the sequence $(X_{r+1}, \ldots, X_{n-1})$, $r < m$, and let $N(\bullet a_{m-r+1}\ldots a_{m+1})$ be the count of the vector $(a_{m-r+1}, \ldots, a_{m+1})$ in the sequence $(X_{r+1}, \ldots, X_n)$. Put

$$\hat{\pi}(a_1\ldots a_m, a_{m+1}) = \frac{N(\bullet a_{m-r+1}\ldots a_{m+1})}{N(\bullet a_{m-r+1}\ldots a_m\bullet)}.$$

Then under the null hypothesis of having a Markov chain of order r against the alternative that it is a Markov chain of order higher than r, the test statistic

$$\chi^2 = \sum_{a_1,\ldots,a_{m+1}\in\mathcal{A}} \frac{\big(N(a_1a_2\ldots a_{m+1}) - N(a_1a_2\ldots a_m\bullet)\hat{\pi}(a_1\ldots a_m, a_{m+1})\big)^2}{N(a_1a_2\ldots a_m\bullet)\hat{\pi}(a_1\ldots a_m, a_{m+1})}$$

is asymptotically chi-square distributed; the degrees of freedom are given by $(\mathrm{Card}(\mathcal{A})^{m+1} - \mathrm{Card}(\mathcal{A})^m) - (\mathrm{Card}(\mathcal{A})^{r+1} - \mathrm{Card}(\mathcal{A})^r)$.

Although this test can be carried out for arbitrary orders, caution is advised: for higher order, a longer sequence of observations is required.

6.2. Overlapping and non-overlapping occurrences

Statistical inference is often based on independence assumptions. Even if the sequence letters are independent and identically distributed, the different random indicators of word occurrences are not independent due to overlaps. For example, if $w = \texttt{atat}$ occurs at position i in the sequence, then another occurrence of w is much more likely to occur at position $i+2$ than if w did not occur at position i, and an occurrence of w at position $i+1$ is not possible. Many of the arguments needed for a probabilistic and statistical analysis of word occurrences deal with disentangling this overlapping structure.

Let $w = w_1 \cdots w_\ell$ be a word of length ℓ on a finite alphabet $\mathcal{A}$. Two occurrences of w may overlap in a sequence if and only if w is periodic, meaning that there exists a period $p \in \{1, \ldots, \ell-1\}$ such that $w_i = w_{i+p}$, $i = 1, \ldots, \ell-p$. A word may have several periods: for instance `gtgtgtg` admits three periods, 2, 4, 6, and `aacaa` has the periods 3 and 4. The set $\mathcal{P}(w)$ of the periods of w is defined by

$$\mathcal{P}(w) := \{p \in \{1, \ldots, \ell-1\} : w_i = w_{i+p}, \forall i = 1, \ldots, \ell-p\}.$$

A word w is not periodic if and only if $\mathcal{P}(w)$ is empty. As we will see later, not all periods of a word will have the same importance; we distinguish the multiples of the minimal period $p_0(w)$ of w from the so-called *principal* periods of w, namely the periods that are not strictly multiples of the minimal period. We denote by $\mathcal{P}'(w)$ the set of the *principal* periods of w. For instance, $\mathcal{P}'(\texttt{gtgtgtg}) = \{2\}$ and $\mathcal{P}'(\texttt{aacaa}) = \{3, 4\}$.

Occurrences of periodic words tend to overlap in a sequence. There are 4 occurrences of `aacaa` in the sequence $\texttt{tg}\underline{\texttt{aacaaacaacaa}}\texttt{tag}\underline{\texttt{aacaa}}\texttt{aa}$, starting respectively at positions 3, 7, 10 and 18. The first 3 occurrences overlap and form a clump. A *clump* of w in a sequence is a maximal set of overlapping occurrences of w in the sequence. By definition two clumps of w in a sequence cannot overlap. A clump composed of exactly k overlapping occurrences of w is called a k-clump of w. There are 2 clumps of `aacaa` in the previous sequence, the first one is a 3-clump starting at position 3 and the second one is a 1-clump starting at position 18. Let $\mathcal{C}_k(w)$ be the set of the concatenated words composed of exactly k overlapping occurrences of w. For example, $\mathcal{C}_1(\texttt{aacaa}) = \{\texttt{aacaa}\}$ and $\mathcal{C}_2(\texttt{aacaa}) = \{\texttt{aacaacaa}, \texttt{aacaaacaa}\}$.

For a word $w = w_1 \cdots w_\ell$ we use the following prefix and suffix notation:

$$\begin{aligned} w^{(p)} &= w_1 \cdots w_p \quad \text{denotes the prefix of } w \text{ of length } p \\ w_{(q)} &= w_{\ell-q+1} \cdots w_\ell \quad \text{denotes the suffix of } w \text{ of length } q, \end{aligned} \tag{6.2.1}$$

and $w^{(p)}w = w_1 \cdots w_p w_1 \cdots w_\ell$ is the concatenated word obtained by two overlapping occurrences starting p positions apart. If $p \in \mathcal{P}(w)$ then $w^{(p)}$ is called a *root* of w; if $p \in \mathcal{P}'(w)$, $w^{(p)}$ is called a *principal root* of w.

Related to the set of periods is the autocorrelation polynomial $\mathcal{Q}(z)$ associated with w defined by

$$\mathcal{Q}(z) = 1 + \sum_{p \in \mathcal{P}(w)} \frac{\mu(w)}{\mu(w^{(\ell-p)})}\, z^p. \tag{6.2.2}$$

Renewals are another type of non-overlapping occurrences of interest that require scanning the sequence from one end to the other: the first occurrence of w in the sequence is a renewal and a given occurrence of w is a renewal if and only if it does not overlap a previous renewal. Renewals of w do not overlap in a sequence. In the above example, there are 3 renewals of `aacaa` starting at position 3, 10 and 18.

Depending on the problem, one could be interested in studying the overlapping occurrences of w in a sequence, or in restricting attention to non-overlapping occurrences: the beginnings of clumps, the beginnings of k-clumps or the renewals. We now introduce notation related to occurrences of a word $w = w_1 \cdots w_\ell$, of a clump of w, of a k-clump of w, of a renewal of w in a sequence, and to the corresponding counts.

Occurrence and number of overlapping occurrences An occurrence of w starts at position i in the sequence $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ if and only if $X_i \cdots X_{i+\ell-1} = w_1 \cdots w_\ell$. Let $Y_i(w)$ be the associated random indicator

$$Y_i(w) := 1\!\!1\{w \text{ starts at position } i \text{ in } \underline{X}\}. \tag{6.2.3}$$

For convenience in some sections, $Y_i(w)$ will be the random indicator that an occurrence of w ends at position i in $\underline{X}$; it will be made precise in that case.

In the stationary m-order Markovian model, the expectation of $Y_i(w)$, that is, the probability that an occurrence of w occurs at a given position in the sequence, is denoted by $\mu_m(w)$ and is given by

$$\mu_m(w) = \mu(w_1 \cdots w_m)\pi(w_1 \cdots w_m, w_{m+1}) \cdots \pi(w_{\ell-m} \cdots w_{\ell-1}, w_\ell). \tag{6.2.4}$$

When there is no ambiguity, the index m referring to the order of the model will be omitted.

The number of overlapping occurrences of w in the sequence $(X_i)_{i=1,\ldots,n}$, simply called *count* of w in this chapter, is defined by $N(w) = N_n(w) = \sum_{i=1}^{n-\ell+1} Y_i(w)$ (or $N(w) = \sum_{i=\ell}^{n} Y_i(w)$ if $Y_i(w)$ is associated with an occurrence of w ending at position i).

Clump and declumped counts A clump of w starts at position i in the infinite sequence $\underline{X}$ if and only if there is an occurrence of w starting at position i that does not overlap a previous occurrence of w. It follows that

$$\begin{aligned} \widetilde{Y}_i(w) &:= 1\!\!1\{\text{a clump of } w \text{ starts at position } i \text{ in } \underline{X}\} \\ &= Y_i(w)(1 - Y_{i-1}(w)) \cdots (1 - Y_{i-\ell+1}(w)). \end{aligned} \tag{6.2.5}$$

Often $\widetilde{Y}_i(w)$ is zero, depending on the overlapping structure of w. Using the principal periods, it turns that

$$\widetilde{Y}_i(w) = Y_i(w) - \sum_{p \in \mathcal{P}'(w)} Y_{i-p}(w^{(p)}w) \tag{6.2.6}$$

with the notation from (6.2.1). Equation (6.2.6) is obtained from the two following steps: (i) note that an occurrence of w starting at position i overlaps a previous occurrence of w if and only if it is directly preceded by an occurrence of a principal root of w, meaning that a principal root $w^{(p)}$, $p \in \mathcal{P}'(w)$, occurs at position $i-p$, (ii) note that the events $E_p = \{Y_{i-p}(w^{(p)}) = 1\}$, $p \in \mathcal{P}'(w)$, are disjoint. To prove (ii), we assume that two different principal roots $w^{(p)}$ and $w^{(q)}$ occur simultaneously at position $i-p$ and $i-q$. If so, the minimal root $w^{(p_0)}$ of w could be decomposed into $w^{(p_0)} = xy = yx$ where x and y are two nonempty words. Now, two words commute if and only if they are powers of the same word. Thus, we would obtain the contradiction that the minimal root is not minimal.

It follows from Equation (6.2.6) that the probability $\widetilde{\mu}(w)$ that a clump of w starts at a given position in $\underline{X}$ is given by

$$\begin{aligned}\widetilde{\mu}(w) &= \mu(w) - \sum_{p \in \mathcal{P}'(w)} \mu(w^{(p)}w) \\ &= (1 - A(w))\mu(w) \end{aligned} \tag{6.2.7}$$

where $A(w)$ is the probability for an occurrence of w to be overlapped from the left by a previous occurrence of w:

$$A(w) = \sum_{p \in \mathcal{P}'(w)} \frac{\mu(w^{(p)}w)}{\mu(w)}. \tag{6.2.8}$$

The number $\widetilde{N}(w)$ of clumps of w in the finite sequence $X_1 \cdots X_n$ (or the declumped count) may be different from the sum $\widetilde{N}_{\text{inf}}(w) = \sum_{i=1}^{n-\ell+1} \widetilde{Y}_i(w)$ because of a possible clump of w that would start in $\underline{X}$ before position 1 and would stop after position $\ell - 1$. The difference $\widetilde{N}(w) - \widetilde{N}_{\text{inf}}(w)$ is either equal to 0 or equal to 1. In fact, it can be shown that $\mathbf{P}(\widetilde{N}(w) \neq \widetilde{N}_{\text{inf}}(w)) \leq (\ell - 1)(\mu(w) - \widetilde{\mu}(w))$.

k-clump and number of k-clumps A *k-clump* of w starts at position i in $\underline{X}$ if and only if there is an occurrence of a concatenated word $c \in \mathcal{C}_k(w)$ starting at position i that does not overlap any other occurrence of w in the sequence $\underline{X}$. As we proceeded for a clump occurrence, an occurrence of $c \in \mathcal{C}_k(w)$ is a k-clump of w in $\underline{X}$ if and only if it is not directly preceded by any principal root $w^{(p)}$ of w and it is not directly followed by any suffix $w_{(q)} = w_{\ell-q+1} \cdots w_\ell$ with $q \in \mathcal{P}'(w)$. Some straightforward calculation yields the expression

$$\widetilde{Y}_{i,k}(w) := \mathbb{1}\{\text{a } k\text{-clump of } w \text{ starts at position } i \text{ in } \underline{X}\} \tag{6.2.9}$$

$$= \sum_{c\in\mathcal{C}_k(w)} \left(Y_i(c) - \sum_{p\in\mathcal{P}'(w)} Y_{i-p}(w^{(p)}c) - \sum_{q\in\mathcal{P}'(w)} Y_i(cw_{(q)}) \right.$$
$$\left. + \sum_{p,q\in\mathcal{P}'(w)} Y_{i-p}(w^{(p)}cw_{(q)}) \right),$$

with the notation (6.2.1). It follows that the probability for a k-clump to start at a given position is given by

$$\widetilde{\mu}_k(w) = \sum_{c\in\mathcal{C}_k(w)} \mu(c) - 2 \sum_{c'\in\mathcal{C}_{k+1}(w)} \mu(c') + \sum_{c''\in\mathcal{C}_{k+2}(w)} \mu(c'').$$

This formula can be simplified. Note that $\mathcal{C}_{k+1}(w) = \{w^{(p)}c, c \in \mathcal{C}_k(w), p \in \mathcal{P}'(w)\}$ and $\mu(w^{(p)}c) = \mu(c)\frac{\mu(w^{(p)}c)}{\mu(c)} = \mu(c)\frac{\mu(w^{(p)}w)}{\mu(w)}$. By using the overlap probability $A(w)$ given in (6.2.8), we have that

$$\sum_{c'\in\mathcal{C}_{k+1}(w)} \mu(c') = A(w) \sum_{c\in\mathcal{C}_k(w)} \mu(c)$$

and it follows that

$$\begin{aligned}
\widetilde{\mu}_k(w) &= (1 - A(w))^2 \sum_{c\in\mathcal{C}_k(w)} \mu(c) \\
&= (1 - A(w))^2 A(w) \sum_{c\in\mathcal{C}_{k-1}(w)} \mu(c) \\
&\;\vdots \\
&= (1 - A(w))^2 A(w)^{k-1} \mu(w). && (6.2.10)
\end{aligned}$$

As for the declumped count, the number of k-clumps of w in the finite sequence may be different from the sum $\widetilde{N}^{(k)}_{\text{inf}}(w) = \sum_{i=1}^{n-\ell+1} \widetilde{Y}_{i,k}(w)$ because of possible end effects. The probability that these counts are not equal can be explicitly bounded, see (6.4.10), (6.4.11) below. Moreover, possible end effects may lead to a difference between the count $N(w)$ and $\sum_{k>0} k\widetilde{N}^{(k)}_{\text{inf}}(w)$, but this can also be controlled.

Renewal and renewal count A renewal of w starts at position i in $X_1 \cdots X_n$ if and only if there is an occurrence of w starting at position i that either is the first one or does not overlap a previous renewal of w. Let $\mathbb{I}_i(w)$ be the associated random indicator:

$$\begin{aligned}
\mathbb{I}_i(w) &= \mathbb{1}\{\text{a renewal of } w \text{ starts at position } i \text{ in } X_1 \cdots X_n\} \\
&= Y_i(w) \prod_{j=i-\ell+1}^{i-1} (1 - \mathbb{I}_j(w)) && (6.2.11)
\end{aligned}$$

with the convention that $\mathbb{I}_j(w) = 0$ if $j < 1$. Thus, for $i \leq \ell$, a renewal occurrence of w at position i is exactly a clump occurrence of w at i in the finite sequence. The renewal count makes extensive use of the linear ordering in the sequence: it is defined by $R(w) = R_n(w) = \sum_{i=1}^{n-\ell+1} \mathbb{I}_i(w)$.

6.3. Word locations along a sequence

Here we are concerned with the length of the gaps between word occurrences. First we describe how to obtain the exact distribution of the distance between successive occurrences of a word, and then we give asymptotic results.

6.3.1. Exact distribution of the distance between word occurrences

Let $w = w_1 \cdots w_\ell$ be a word of length ℓ on a finite alphabet $\mathcal{A}$. We assume that $X_1 \cdots X_n$ is a stationary first-order Markov chain on $\mathcal{A}$ with transition matrix $\Pi = (\pi(a,b))_{a,b\in\mathcal{A}}$ and stationary distribution $(\mu(a))_{a\in\mathcal{A}}$. Here we are interested in the statistical distribution of the distance D between two successive occurrences of w and more precisely in the probabilities

$$\begin{aligned} f(d) &= \mathbf{P}(D = d) \\ &= \mathbf{P}(w \text{ occurs at } i+d \text{ and there is no occurrence of } w \\ &\qquad \text{between } i+1 \text{ and } i+d-1 \mid w \text{ occurs at } i), \quad d \geq 1. \end{aligned}$$

In this section, we say that a word w occurs at position i if an occurrence of w ends at position i; it happens with probability $\mu(w)$ given in (6.2.4).

The probability $f(d)$ can be obtained via a recursive formula as follows. It is clear that, if $1 \leq d \leq \ell - 1$ and $d \notin \mathcal{P}(w)$, then $f(d) = 0$. If $d \in \mathcal{P}(w)$ or if $d \geq \ell$ then we decompose the event

$$E = \{w \text{ occurs at } i+d\}$$

into the disjoint events

$$\begin{aligned} E_1 = \{&w \text{ occurs at } i+d \text{ and there is no occurrence of } w \text{ between } i+1 \\ &\text{and } i+d-1\} \end{aligned}$$

and

$$\begin{aligned} E_2 = \{&w \text{ occurs at } i+d \text{ and there are some occurrences of } w \text{ between } i+1 \\ &\text{and } i+d-1\}. \end{aligned}$$

Thus $\{E_1 \mid w \text{ at } i\}$ has probability $f(d)$. Moreover E_2 is itself decomposed as $E_2 = \cup_{h=1}^{d-1} E_2(h)$, where

$$\begin{aligned} E_2(h) = \{&\text{there is no occurrence of } w \text{ between } i+1 \text{ and } i+h-1, \\ &w \text{ occurs at } i+h \text{ and } i+d\} \end{aligned}$$

are again disjoint events.

If $1 \le d \le \ell - 1$ and $d \in \mathcal{P}(w)$, then $\mathbf{P}(E \mid w \text{ at } i) = \mu(w)/\mu(w^{(\ell-d)})$. Moreover, if there are occurrences at positions $i+h$ and $i+d$, for some $h < d$, then the occurrences necessarily overlap, and this is only possible for $d - h \in \mathcal{P}(w)$; in this case, $\mathbf{P}(E_2(h) \mid w \text{ at } i) = f(h)\mu(w)/\mu(w^{(\ell-d+h)})$. Thus, we have

$$\frac{\mu(w)}{\mu(w^{(\ell-d)})} = f(d) + \sum_{\substack{1 \le h \le d-1 \\ d-h \in \mathcal{P}(w)}} f(h) \frac{\mu(w)}{\mu(w^{(\ell-d+h)})}.$$

If $d \ge \ell$, then $\mathbf{P}(E \mid w \text{ at } i) = \Pi^{d-\ell+1}(w_\ell, w_1)\mu(w)/\mu(w_1)$. If there is an occurrence at positions $i+h$ and $i+d$, for some $h < d$, then we distinguish two cases depending on the possible overlap between the occurrences at $i+h$ and $i+d$: if $d - \ell + 1 \le h \le d - 1$, they overlap and we use previous calculation; if $1 \le h \le d - \ell$, they do not overlap and $\mathbf{P}(E_2(h) \mid w \text{ at } i) = f(h)\Pi^{d-\ell-h+1}(w_\ell, w_1)\mu(w)/\mu(w_1)$. Thus, from

$$\mathbf{P}(E \mid w \text{ at } i) = \mathbf{P}(E_1 \mid w \text{ at } i) + \sum_{h=1}^{d-1} \mathbf{P}(E_2(h) \mid w \text{ at } i)$$

we get

$$\begin{aligned}\Pi^{d-\ell+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} &= f(d) + \sum_{1 \le h \le d-\ell} f(h)\Pi^{d-\ell-h+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} \\ &\quad + \sum_{\substack{d-\ell+1 \le h \le d-1 \\ d-h \in \mathcal{P}(w)}} f(h)\frac{\mu(w)}{\mu(w^{(\ell-d+h)})}.\end{aligned}$$

This is the proof of the next theorem.

Theorem 6.3.1. *The distribution $f(d) = \mathbf{P}(D = d)$ of the distance D between two successive occurrences of a word w in a Markov chain is given by the following recursive formula:*

If $1 \le d \le \ell - 1$ and $d \notin \mathcal{P}(w)$, then $f(d) = 0$.
If $1 \le d \le \ell - 1$ and $d \in \mathcal{P}(w)$,

$$f(d) = \frac{\mu(w)}{\mu(w^{(\ell-d)})} - \sum_{\substack{1 \le h \le d-1 \\ d-h \in \mathcal{P}(w)}} f(h)\frac{\mu(w)}{\mu(w^{(\ell-d+h)})}.$$

If $d \ge \ell$,

$$\begin{aligned}f(d) &= \Pi^{d-\ell+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} - \sum_{1 \le h \le d-\ell} f(h)\Pi^{d-\ell-h+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} \\ &\quad - \sum_{\substack{d-\ell+1 \le h \le d-1 \\ d-h \in \mathcal{P}(w)}} f(h)\frac{\mu(w)}{\mu(w^{(\ell-d+h)})}.\end{aligned}$$

Since D is the distance between two successive occurrences of w, note that, even if $d \in \mathcal{P}(w)$, $f(d)$ can be null. For instance, by taking $w = \mathtt{aaa}$, we have $\mathcal{P}(\mathtt{aaa}) = \{1, 2\}$, and $f(1) = \mu(\mathtt{aaa})/\mu(\mathtt{aa}) = \pi(\mathtt{a}, \mathtt{a})$, $f(2) = \pi^2(\mathtt{a}, \mathtt{a}) - f(1)\pi(\mathtt{a}, \mathtt{a}) = 0$.

Note that the recurrence formula on $f(d)$ is not a "finite" recurrence since calculating $f(d)$ requires the calculation of $f(d-1), \ldots, f(1)$, involving substantial numerical calculations for large d. One can approach this computation problem by using the generating function defined by $\Phi_D(t) := \mathbf{E}(t^D) = \sum_{d \geq 1} f(d)t^d$. The key argument is that the $\Phi_D(t)$ expression is a rational function of the form $P(t)/Q(t)$, and hence the coefficient $f(d)$ of t^d can be expressed by a recurrence formula whose order is the degree of the polynomial $Q(t)$ (see Section 6.8.4).

Theorem 6.3.2. *The generating function of D is*

$$\Phi_D(t) = 1 - \mu^{-1}(w) \left(\sum_{\substack{u=0 \\ u \in \mathcal{P}(W) \cup \{0\}}}^{\ell-1} \frac{t^u}{\mu(w^{(\ell-u)})} + \frac{1}{\mu(w_1)} \sum_{u \geq 1} \Pi^u(w_\ell, w_1) t^{\ell+u-1} \right)^{-1}.$$

Remark 6.3.3. If the transition matrix Π is diagonalizable, there exists δ_i, $\beta_i \in \mathbb{C}$, $i = 2 \cdots |\mathcal{A}|$, such that

$$\frac{1}{\mu(w_1)} \sum_{u \geq 1} \Pi^u(w_\ell, w_1) t^{\ell+u-1} = \frac{t^\ell}{1-t} \left(1 + \frac{1-t}{\mu(w_1)} \sum_{i=2}^{|\mathcal{A}|} \frac{\delta_i}{1 - t\beta_i} \right)$$

implying that the above expression is a rational function with a pole at $t = 1$.

Remark 6.3.4. Since $\Phi_D(t) = \sum_{d \geq 1} f(d)t^d$, we have the general following properties:

$$\begin{aligned} \mathbf{E}(D) &= \Phi'_D(1) = \mu^{-1}(w) \\ \mathrm{Var}(D) &= \Phi''_D(1) + \Phi'_D(1)(1 - \Phi'_D(1)). \end{aligned}$$

Successive derivatives of $\Phi_D(t)$ are obtained using the decomposition stated in the previous remark.

Proof

The proof of Theorem 6.3.2 is not complicated since one just has to develop the sum $\sum_{d \geq 0} f(d)t^d$ with $f(d)$ given by Theorem 6.3.1, but it is very technical. We thus only give the main lines of the calculation. By replacing $f(d)$ given by Theorem 6.3.1 in $\sum_{d \geq 0} f(d)t^d$, we obtain a sum of five term

$$\Phi_D(t) = K_1 - K_2 + K_3 - K_4 - K_5$$

with

$$
\begin{aligned}
K_1 &= \sum_{\substack{d=1\\ d\in\mathcal{P}(W)}}^{\ell-1} \frac{\mu(w)}{\mu(w^{(\ell-d)})} t^d \\
K_2 &= \sum_{\substack{d=1\\ d\in\mathcal{P}(W)}}^{\ell-1} \sum_{\substack{h=1\\ d-h\in\mathcal{P}(W)}}^{d-1} f(h)\frac{\mu(w)}{\mu(w^{(\ell-d+h)})} t^d \\
&= \sum_{h=1}^{\ell-2} f(h) \sum_{\substack{u=1\\ u\in\mathcal{P}(W)}} \frac{\mu(w)}{\mu(w^{(\ell-u)})} t^{h+u} \\
K_3 &= \sum_{d\geq\ell} \Pi^{d-\ell+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} t^d \\
&= \frac{\mu(w)}{\mu(w_1)} t^{\ell-1} \sum_{u\geq 1} \Pi^u(w_\ell, w_1) t^u \\
K_4 &= \sum_{d\geq\ell} \sum_{h=1}^{d-\ell} f(h)\Pi^{d-\ell-h+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)} t^d \\
&= \frac{\mu(w)}{\mu(w_1)} t^{\ell-1} \sum_{h\geq 1} f(h) t^h \sum_{z\geq h} \Pi^{z-h+1}(w_\ell, w_1) t^{z-h+1} \\
&= \frac{\mu(w)}{\mu(w_1)} t^{\ell-1} \Phi_D(t) \sum_{u\geq 1} \Pi^u(w_\ell, w_1) t^u
\end{aligned}
$$

and

$$
\begin{aligned}
K_5 &= \sum_{d\geq\ell} \sum_{\substack{h=d-\ell+1\\ d-h\in\mathcal{P}(W)}}^{d-1} f(h)\frac{\mu(w)}{\mu(w^{(\ell-d+h)})} t^d \\
K_5 &= \sum_{h=1}^{\ell-1} f(h) \sum_{\substack{z=1\\ z+\ell-h-1\in\mathcal{P}(W)}}^{h} \frac{\mu(w)}{\mu(w^{(h-z+1)})} t^{z+\ell-1} \\
&\quad + \sum_{h\geq\ell} f(h) t^h \sum_{\substack{z=h-\ell+2\\ z+\ell-h-1\in\mathcal{P}(W)}}^{h} \frac{\mu(w)}{\mu(w^{(h-z+1)})} t^{z-h+\ell-1} \\
&= \sum_{h=1}^{\ell-1} f(h) \sum_{\substack{u=\ell-h\\ u\in\mathcal{P}(W)}}^{\ell-1} \frac{\mu(w)}{\mu(w^{(\ell-u)})} t^{h+u} + \sum_{h\geq\ell} f(h) t^h \sum_{\substack{u=1\\ u\in\mathcal{P}(W)}}^{\ell-1} \frac{\mu(w)}{\mu(w^{(\ell-u)})} t^u.
\end{aligned}
$$

Grouping $K_1 - K_2 - K_5$ and $K_3 - K_4$ leads to

$$\Phi_D(t) = (1 - \Phi_D(t)) \left(\sum_{\substack{u=1 \\ u \in \mathcal{P}(W)}}^{\ell-1} \frac{\mu(w)}{\mu(w^{(\ell-u)})} t^u + \frac{\mu(w)}{\mu(w_1)} t^{\ell-1} \sum_{u \geq 1} \Pi^u(w_\ell, w_1) t^u \right),$$

hence

$$\Phi_D(t) = 1 - \left(1 + \sum_{\substack{u=1 \\ u \in \mathcal{P}(W)}}^{\ell-1} \frac{\mu(w)}{\mu(w^{(\ell-u)})} t^u + \frac{\mu(w)}{\mu(w_1)} t^{\ell-1} \sum_{u \geq 1} \Pi^u(w_\ell, w_1) t^u \right)^{-1}$$

Using $\mu(w)/\mu(w^{(\ell)}) = 1$ establishes the theorem.

■

The distance D between two successive occurrences of w can be seen as the distance between the j-th and $(j+1)$-th occurrence of w in the sequence, since we use a homogeneous model. It may be useful to study the distance $D^{(r)}$ between the j-th and $(j+r)$-th occurrence of w, the so-called r-scan. The distance $D^{(r)}$ is the sum of r independent and identically distributed random variables with same distribution as D. Hence we have

$$\Phi_{D^{(r)}}(t) = \big(\Phi_D(t)\big)^r.$$

We obtain the exact distribution of $D^{(r)}$ from the Taylor expansion of $\Phi_{D^{(r)}}(t)$: the probability $\mathbf{P}(D^{(r)} = d)$ is the coefficient of t^d in the series.

6.3.2. Asymptotic distribution of r-scans

In the preceding paragraph, we presented how to obtain the exact distribution of an r-scan $D^{(r)}$, the distance between a word occurrence and the $(r-1)$-th next one, in a stationary Markov chain of first order. Often one is interested in the occurrence of any element of a subset of words; such a subset is called a *motif*. When analyzing a biological sequence, assume we observe $(h+1)$ occurrences of a given motif, so that we observe h distances $D_1, \ldots, D_h$ between occurrences of the motif. Thus we observe $(h-r+1)$ so-called r-scans $D_i^{(r)} = \sum_{j=i}^{i+r-1} D_j$. To detect poor and rich regions with this motif, one is interested in studying the significance of the smallest and the largest r-scans, or more generally the kth smallest r-scan, denoted by m_k, and the kth largest r-scan, denoted by M_k. In this section, we present a Poisson approximation for the statistical distribution of the extreme value m_k using the Chen-Stein method. A similar result is available for M_k by following an identical setup, so it will not be explained in detail here.

We begin by defining the Bernoulli variables that will be used in the Chen-Stein method (see Section 6.8.2):

$$W_i^-(d) := 1\!\!1\{D_i^{(r)} \leq d\},\ d \geq 0.$$

Denote by

$$W^-(d) = \sum_{i=1}^{h-r+1} W_i^-(d)$$

the number of r-scans less or equal to d. Note the duality principle

$$\{W^-(d) < k\} = \{m_k > d\},\ d \geq 0.$$

We now use Theorem 6.8.2 to get a Poisson approximation for the distribution of $W^-(d)$. To apply this theorem, we first need to choose a neighborhood of dependence for each indicator variable; ideally the indicator variables with indices not from the neighborhood of dependence are independent of that indicator variable. Secondly there are three quantities to bound, called b_1, b_2, and b_3, given in (6.8.1), (6.8.2), and (6.8.3). Piecing this together gives a bound on the total variation distance between the distributions. Here we proceed as follows.

For $i \in \{1, \ldots, h-r+1\}$, we choose the neighborhood $B_i = \{j \mid |i-j| < r\}$, so that $D_i^{(r)}$ is independent of $D_j^{(r)}$ if $j \notin B_i$ (recall the distances $D_1, \ldots, D_h$ are independent). Let Z_{λ^-} be the Poisson variable with expectation λ^-, where

$$\begin{aligned}
\lambda^- &= \mathbf{E}(W^-(d)) \\
&= (h-r+1)\mathbf{E}(W_i^-(d)) \\
&= (h-r+1)\mathbf{P}(D^{(r)} \leq d).
\end{aligned}$$

Theorem 6.8.2 gives that

$$\begin{aligned}
d_{\mathrm{TV}}\left(\mathcal{L}(W^-(d)), \mathcal{L}(Z_{\lambda^-})\right) \leq \frac{1-\mathrm{e}^{-\lambda^-}}{\lambda^-} & \left(\sum_{i=1}^{h-r+1} \sum_{j \in B_i} \mathbf{E}(W_i^-(d))\mathbf{E}(W_j^-(d)) \right. \\
& \left. + \sum_{i=1}^{h-r+1} \sum_{j \in B_i \setminus \{i\}} \mathbf{E}(W_i^-(d) W_j^-(d)) \right).
\end{aligned}$$

Indeed the neighborhood B_i is chosen so that $W_i^-(d)$ is independent of $W_j^-(d)$, $\forall j \notin B_i$, leading to $b_3 = 0$. For $j > i$, we have

$$\begin{aligned}
\mathbf{E}(W_i^-(d) W_j^-(d)) &= \mathbf{P}(D_i^{(r)} \leq d, D_j^{(r)} \leq d) \\
&= \mathbf{P}(D_j^{(r)} \leq d \mid D_i^{(r)} \leq d)\mathbf{P}(D_i^{(r)} \leq d) \\
&= \mathbf{P}(D_{j-i+1}^{(r)} \leq d \mid D_1^{(r)} \leq d)\mathbf{P}(D^{(r)} \leq d).
\end{aligned}$$

Therefore,

$$\sum_{i=1}^{h-r+1} \sum_{j \in B_i \setminus \{i\}} \mathbf{E}(W_i^-(d) W_j^-(d))$$

$$\leq 2(h-r+1)\mathbf{P}(D^{(r)} \leq d) \sum_{s=2}^{r} \mathbf{P}(D_s^{(r)} \leq d \mid D_1^{(r)} \leq d)$$
$$\leq 2\lambda^- \sum_{s=2}^{r} \mathbf{P}(D_s^{(r)} \leq d \mid D_1^{(r)} \leq d).$$

It can be shown that

$$\mathbf{P}(D_s^{(r)} \leq d \mid D_1^{(r)} \leq d) \leq \mathbf{P}\left(\sum_{i=r+1}^{s+r-1} D_i \leq d\right) = \mathbf{P}(D^{(s-1)} \leq d).$$

We finally get

$$d_{\mathrm{TV}}\left(\mathcal{L}(W^-(d)), \mathcal{L}(Z_{\lambda^-})\right) \leq \left((2r-1)\mathbf{P}(D^{(r)} \leq d) + 2\sum_{s=1}^{r-1} \mathbf{P}(D^{(s)} \leq d)\right) \times (1 - \mathrm{e}^{-\lambda^-}).$$

From the duality principle,

$$|\mathbf{P}(m_k > d) - \mathbf{P}(Z_{\lambda^-} < k)| \leq \left((2r-1)\mathbf{P}(D^{(r)} \leq d) + 2\sum_{s=1}^{r-1} \mathbf{P}(D^{(s)} \leq d)\right) \times (1 - \mathrm{e}^{-\lambda^-}).$$

This approximation is very useful for the comparison between the expected distribution of the r-scans and the one observed in the biological sequence.

6.4. Word count distribution

Let again $w = w_1 \cdots w_\ell$ be a word of length ℓ on a finite alphabet $\mathcal{A}$ and $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ be a random sequence on $\mathcal{A}$. This section is devoted to the statistical distribution of the count $N(w)$ of w in the sequence $X_1 \cdots X_n$. First we state how to compute the exact distribution in the model M1, using recursion techniques. For long sequences, however, asymptotic results are obtainable, and, in general, easier to handle. Here the appropriate asymptotic regime depends crucially on the length ℓ of the target word relative to the sequence length n. For very short words, the law of large numbers can be applied to approximate the word count by the expected word count. This being a very crude estimate, one can easily improve on it by employing the Central Limit Theorem, stating that the word count distribution is asymptotically normal. This approximation will be satisfactory when the words are not too long. For rare words, as a rule of thumb words of length $\ell \asymp \log n$, a compound Poisson approximation will give better results. For the latter, the error made in the approximation can be bounded in terms of the sequence length, the word length, and word probabilities, so that it is possible to assess when a compound Poisson approximation will be a good choice. Moreover, the error bound can be incorporated to give conservative confidence intervals, as will be explained below.

6.4.1. Exact distribution

If $\underline{X}$ is a stationary first-order Markov chain, the exact distribution of the count $N(w)$ can be easily obtained using the distribution of the successive positions $(T_j)_{j\geq 1}$ of the j-th occurrence of w in $X_1 \cdots X_n$, using the duality principle

$$\{N(w) \geq j\} = \{T_j \leq n\}.$$

The exact distribution of T_j can be obtained as in Section 6.3.1, by deriving the Taylor expansion of the generating function $\Phi_{T_j}(t)$ of T_j. If $j = 1$, the generating function $\Phi_{T_1}(t)$ can be obtained as $\Phi_D(t)$ (see Theorem 6.3.2). We just state the result:

$$\Phi_{T_1}(t) = \frac{t^\ell}{1-t} \left(\sum_{\substack{u=0 \\ u \in \mathcal{P}(W) \cup \{0\}}}^{\ell-1} \frac{t^u}{\mu(w^{(\ell-u)})} + \frac{1}{\mu(w_1)} \sum_{u \geq 1} \Pi^u(w_\ell, w_1) t^{\ell+u-1} \right)^{-1}.$$

As $T_j - T_1$ is a sum of $j-1$ independent and identically distributed random variables with the same distribution as D, we have $\Phi_{T_j}(t) = \Phi_{T_1}(t)\big(\Phi_D(t)\big)^{j-1}$. Now $\mathbf{P}(T_j = a) = g_j(a)$ is equal to the coefficient of t^a in the Taylor expansion of $\Phi_{T_j}(t)$. Using the duality principle, we obtain

$$\mathbf{P}(N(w) = j) = \sum_{a=\ell}^{n} g_j(a) - g_{j+1}(a).$$

6.4.2. The weak law of large numbers

As a crude first approximation, the weak law of large numbers states that the observed counts will indeed converge towards the expected counts. Indeed we may use Chebyshev's inequality to bound the expected deviation of the observed counts from the expected number of occurrences. This approximation is valid only for relatively short words, and in this case a normal approximation gives more information. Such an approximation will be derived in the following subsection.

6.4.3. Asymptotic distribution: the Gaussian regime

We assume that $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ is a stationary m-order Markov chain on $\mathcal{A}$, $0 \leq m \leq \ell - 2$, with transition probabilities $\pi(a_1 \cdots a_m, a_{m+1})$ and stationary distribution $\mu(a_1 \cdots a_m)$, $a_1, \ldots, a_{m+1} \in \mathcal{A}$. For convenience in this particular subsection, we consider $N(w) = \sum_{i=\ell}^{n} Y_i(w)$ and

$$Y_i = Y_i(w) = \mathbb{1}\{w \textit{ ends} \text{ at position } i \text{ in } \underline{X}\}.$$

If the model is known, the asymptotic normality of $(N(w) - \mathbf{E}(N(w)))/\sqrt{n}$ directly follows from a Central Limit Theorem for Markov chains. When

$m = 1$, the expectation and variance of $N(w)$ are

$$\begin{aligned}\mathbf{E}(N(w)) &= (n - \ell + 1)\mu_1(w)\\ \mathrm{Var}(N(w)) &= \mathbf{E}(N(w)) + 2 \sum_{p \in \mathcal{P}(w)} \mathbf{E}(N(w^{(p)}w)) - \mathbf{E}(N(w))^2\\ &\quad + \frac{2}{\mu(w_1)} \mu_1^2(w) \sum_{d=1}^{n-2\ell+1} (n - 2\ell + 2 - d)\Pi^d(w_\ell, w_1) \end{aligned} \tag{6.4.1}$$

where $\mu_1(w)$ is given in Eq. (6.2.4).

In the problem of finding exceptional words in biological sequences, the model is unknown and its parameters are estimated from the observed sequence. The expected mean of $N(w)$ is not available and is approximated by an estimator $\widehat{N}_m(w)$. In this paragraph, we derive both the asymptotic normality of $(N(w) - \widehat{N}_m(w))/\sqrt{n}$ and the asymptotic variance. This is not a trivial problem since the estimation changes the variance expression fundamentally.

The expected mean of $N(w)$ is given by $\mathbf{E}(N(w)) = (n - \ell + 1)\mu(w)$ where $\mu(w) = \mu_m(w)$ is the probability that an occurrence of w ends at a given position in the sequence (see Eq. (6.2.4)). Estimating each parameter by its maximum likelihood estimator (with the simplification from Remark 6.1.1) gives an estimator $\widehat{N}_m(w)$ of $\mathbf{E}(N(w))$:

$$\widehat{N}_m(w) = \frac{N(w_1 \cdots w_{m+1}) \cdots N(w_{\ell-m} \cdots w_\ell)}{N(w_2 \cdots w_{m+1}) \cdots N(w_{\ell-m} \cdots w_{\ell-1})}. \tag{6.4.2}$$

Maximal model Let us first consider the maximal model ($m = \ell - 2$), which is mainly used to find exceptional words. To shorten the formulas, we introduce the notation

$$\begin{aligned} w^- &:= w_1 \cdots w_{\ell-1} && \text{first } \ell - 1 \text{ letters of } w\\ {}^-w &:= w_2 \cdots w_\ell && \text{last } \ell - 1 \text{ letters of } w\\ {}^-w^- &:= w_2 \cdots w_{\ell-1} && \ell - 2 \text{ central letters of } w. \end{aligned}$$

Under the maximal model, the estimator of $N(w)$ is

$$\widehat{N}_{\ell-2}(w) = \frac{N(w_1 \cdots w_{\ell-1})N(w_2 \cdots w_\ell)}{N(w_2 \cdots w_{\ell-1})} = \frac{N(w^-)N({}^-w)}{N({}^-w^-)};$$

moreover, the asymptotic normality of $(N(w) - \widehat{N}_{\ell-2}(w))/\sqrt{n}$ and the asymptotic variance can be obtained in an elegant way using martingale techniques. Indeed, $\widehat{N}_{\ell-2}(w)$ is a natural estimator of $N(w^-)\pi({}^-w^-, w_\ell)$, and $N(w) - N(w^-)\pi({}^-w^-, w_\ell)$ is approximately a martingale as it is shown below.

We introduce the martingale $M_n = \sum_{i=\ell}^{n} (Y_i - \mathbf{E}(Y_i \mid \mathcal{F}_{i-1}))$ with $\mathcal{F}_i = \sigma(X_1, \ldots, X_i)$; it is easy to verify that $\mathbf{E}(M_n \mid \mathcal{F}_{n-1}) = M_{n-1}$. Moreover, we have

$$\begin{aligned}\mathbf{E}(Y_i \mid \mathcal{F}_{i-1}) &= \mathbf{P}(w^- \text{ ends at } i-1 \text{ and } w_\ell \text{ occurs at } i \mid \mathcal{F}_{i-1})\\ &= \mathbb{1}\{w^- \text{ ends at } i-1\}\pi({}^-w^-, w_\ell),\end{aligned}$$

and

$$\sum_{i=\ell}^{n} \mathbf{E}(Y_i \mid \mathcal{F}_{i-1}) = \big(N(w^-) - \mathbb{I}\{w^- \text{ ends at } n\}\big)\pi({}^-w^-, w_\ell).$$

Therefore,

$$\frac{1}{\sqrt{n}} M_n = \frac{1}{\sqrt{n}}\big(N(w) - N(w^-)\pi({}^-w^-, w_\ell)\big) \\ -\frac{1}{\sqrt{n}}\mathbb{I}\{w^- \text{ ends at } n\}\pi({}^-w^-, w_\ell). \tag{6.4.3}$$

Note that $n^{-1/2}\mathbb{I}\{w^- \text{ ends at } n\}\pi({}^-w^-, w_\ell)$ tends to zero as $n \to \infty$. The next proposition establishes the asymptotic normality of $M_n/\sqrt{n}$.

PROPOSITION 6.4.1. *Let $V = \mu(w^-)\pi({}^-w^-, w_\ell)(1 - \pi({}^-w^-, w_\ell))$. We have*

$$\frac{1}{\sqrt{n}} M_n \xrightarrow{\mathcal{D}} \mathcal{N}(0, V) \text{ as } n \to \infty.$$

Proof

This is an application of Theorem 6.8.7 for the one-dimensional random variable $\xi_{n,i} = n^{-1/2}(Y_i - \mathbf{E}(Y_i \mid \mathcal{F}_{i-1}))$. Three conditions have to be satisfied. Condition (i) holds from $\mathbf{E}(\xi_{n,i} \mid \mathcal{F}_{i-1}) = 0$. We then have to check that $\sum_{i=\ell}^{n} \mathrm{Var}(\xi_{n,i} \mid \mathcal{F}_{i-1})$ converges to V as $n \to \infty$. Since Y_i is a 0-1 random variable, we have

$$\begin{aligned}\mathrm{Var}(Y_i \mid \mathcal{F}_{i-1}) &= \mathbf{E}(Y_i \mid \mathcal{F}_{i-1}) - \big(\mathbf{E}(Y_i \mid \mathcal{F}_{i-1})\big)^2 \\ &= \mathbb{I}\{w^- \text{ ends at } i-1\}\pi({}^-w^-, w_\ell)\big(1 - \pi({}^-w^-, w_\ell)\big).\end{aligned}$$

We thus obtain

$$\begin{aligned}\sum_{i=\ell}^{n} \mathrm{Var}(\xi_{n,i} \mid \mathcal{F}_{i-1}) &= \frac{1}{n}\sum_{i=\ell}^{n} \mathrm{Var}(Y_i \mid \mathcal{F}_{i-1}) \\ &= \frac{1}{n} N(w^-)\pi({}^-w^-, w_\ell)(1 - \pi({}^-w^-, w_\ell)) \\ &\quad -\frac{1}{n}\mathbb{I}\{w^- \text{ ends at } i-1\}\pi({}^-w^-, w_\ell)(1 - \pi({}^-w^-, w_\ell)) \\ &\longrightarrow V \text{ as } n \to \infty;\end{aligned}$$

the convergence follows from the Law of Large Numbers: $N(w^-)/n \to \mu(w^-)$. Finally, $|\xi_{n,i}| \le \frac{2}{\sqrt{n}}$, so that $\forall \varepsilon > 0$, $\forall n > 4/\varepsilon^2$, $\mathbf{P}(|\xi_{n,i}| > \varepsilon) = 0$, establishing condition (iii). Using Theorem 6.8.7 proves the proposition. ∎

Proposition 6.4.1 and Equation (6.4.3) also yield that

$$\frac{1}{\sqrt{n}}\big(N(w) - N(w^-)\pi({}^-w^-, w_\ell)\big) \xrightarrow{\mathcal{D}} \mathcal{N}(0, V) \text{ as } n \to \infty.$$

We want to prove such convergence for

$$T_n = \frac{1}{\sqrt{n}}\big(N(w) - N(w^-)\widehat{\pi}({}^-w^-, w_\ell)\big),$$

where

$$\widehat{\pi}({}^-w^-, w_\ell) = \frac{N({}^-w)}{N({}^-w^-)}.$$

To this purpose, we decompose T_n as follows:

$$\begin{aligned}
T_n &= \frac{1}{\sqrt{n}}\big(N(w) - N(w^-)\pi({}^-w^-, w_\ell)\big) \\
&\quad - \frac{1}{\sqrt{n}}N(w^-)\big(\widehat{\pi}({}^-w^-, w_\ell) - \pi({}^-w^-, w_\ell)\big) \\
&= \frac{1}{\sqrt{n}}\big(N(w) - N(w^-)\pi({}^-w^-, w_\ell)\big) \\
&\quad - \frac{1}{\sqrt{n}}\frac{N(w^-)}{N({}^-w^-)}\big(N({}^-w) - N({}^-w^-)\pi({}^-w^-, w_\ell)\big) \\
&= \frac{1}{\sqrt{n}}M_n - \frac{1}{\sqrt{n}}\frac{N(w^-)}{N({}^-w^-)}M'_n + o(1), \qquad (6.4.4)
\end{aligned}$$

where M'_n is the martingale $M'_n = \sum_{i=\ell}^n \left(Y_i({}^-w) - \mathbf{E}(Y_i({}^-w) \mid \mathcal{F}_{i-1})\right)$. Now, using Theorem 6.8.7 gives

$$\frac{1}{\sqrt{n}}\begin{pmatrix} M_n \\ M'_n \end{pmatrix} \longrightarrow \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}; \begin{pmatrix} V & V_{12} \\ V_{21} & V_{22} \end{pmatrix}\right) \qquad (6.4.5)$$

with

$$V_{21} = V_{12} = \lim_{n\to\infty}\frac{1}{n}\sum_{i=\ell}^{n}\mathbf{E}\Big(\big(Y_i - \mathbf{E}(Y_i \mid \mathcal{F}_{i-1})\big)\big(Y_i({}^-w) - \mathbf{E}(Y_i({}^-w) \mid \mathcal{F}_{i-1})\big)\Big)$$

and

$$V_{22} = \lim_{n\to\infty}\frac{1}{n}\sum_{i=\ell}^{n}\mathrm{Var}(Y_i({}^-w) \mid \mathcal{F}_{i-1}).$$

With the same technique as for the derivation of V, as $Y_iY_i({}^-w) = Y_i$, we get $V_{21} = V_{12} = V$ and $V_{22} = \mu({}^-w^-)\pi({}^-w^-, w_\ell)(1 - \pi({}^-w^-, w_\ell))$. Note that the Law of Large Numbers guarantees that, almost surely,

$$\frac{N(w^-)}{N({}^-w^-)} \to \frac{\mu(w^-)}{\mu({}^-w^-)} \text{ as } n \to \infty. \qquad (6.4.6)$$

From (6.4.4)–(6.4.6), we are now able to deduce that T_n converges in distribution to $\mathcal{N}(0, \sigma^2_{\ell-2}(w))$ with

$$\sigma^2_{\ell-2}(w) = V_{11} - 2\frac{\mu(w^-)}{\mu({}^-w^-)}V_{12} + \left(\frac{\mu(w^-)}{\mu({}^-w^-)}\right)^2 V_{22}$$

$$
\begin{aligned}
&= \mu(w^-)\left(1 - \frac{\mu(w^-)}{\mu(^-w^-)}\right)\pi(^-w^-, w_\ell)(1 - \pi(^-w^-, w_\ell)) \\
&= \frac{\mu(w)}{\mu(^-w^-)}\big(\mu(^-w^-) - \mu(w^-)\big)(1 - \pi(^-w^-, w_\ell)) \\
&= \frac{\mu(w)}{\mu(^-w^-)}\big(\mu(^-w^-) - \mu(w^-) - \mu(^-w) + \mu(w)\big) \\
&= \frac{\mu(w)}{\mu(^-w^-)^2}\big(\mu(^-w^-) - \mu(^-w)\big)\big(\mu(^-w^-) - \mu(w^-)\big).
\end{aligned}
$$

We have just proved the following theorem.

THEOREM 6.4.2. *As $n \to \infty$, we have*

$$
\frac{1}{\sqrt{n}}\left(N(w) - \widehat{N}_{\ell-2}(w)\right) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2_{\ell-2}(w))
$$

with

$$
\sigma^2_{\ell-2}(w) = \frac{\mu(w)}{\mu(^-w^-)^2}(\mu(^-w^-) - \mu(^-w))(\mu(^-w^-) - \mu(w^-))
$$

and

$$
\frac{N(w) - \widehat{N}_{\ell-2}(w)}{\sqrt{n\widehat{\sigma}^2_{\ell-2}(w)}} \xrightarrow{\mathcal{D}} \mathcal{N}(0, 1)
$$

where $n\widehat{\sigma}^2_{\ell-2}(w)$ is the plug-in estimator of $n\sigma^2_{\ell-2}(w)$:

$$
n\widehat{\sigma}^2_{\ell-2}(w) = \frac{\widehat{N}_{\ell-2}(w)}{N(^-w^-)^2}\big(N(^-w^-) - N(^-w)\big)\big(N(^-w^-) - N(w^-)\big).
$$

Non-maximal model In the non-maximal models ($m < \ell - 2$), it is straightforward to extend the previous martingale approach to prove the asymptotic normality of $(N(w) - \widehat{N}_m(w))/\sqrt{n}$ and to derive the asymptotic variance. Indeed, for each value of $\ell - m$, the difference $N(w) - \widehat{N}_m(w)$ can be decomposed as a linear combination of martingales, exactly as for T_n. For instance, if $w = abcde$ and $m = 1$, write

$$
\begin{aligned}
N(abcde) - \widehat{N}_1(abcde) &= N(abcde) - \frac{N(ab)N(bc)N(cd)N(de)}{N(b)N(c)N(d)} \\
&= N(abcde) - N(abcd)\frac{N(de)}{N(d)} \\
&\quad + \frac{N(de)}{N(d)}\left(N(abcd) - N(abc)\frac{N(cd)}{N(c)}\right) \\
&\quad + \frac{N(de)N(cd)}{N(d)N(c)}\left(N(abc) - N(ab)\frac{N(bc)}{N(b)}\right).
\end{aligned}
$$

Another approach uses the δ-method. The idea is to consider $N(w)-\widehat{N}_m(w)$ as $f(\underline{N})$, where $\underline{N}$ is the count vector

$$\begin{aligned}\underline{N} = (&N(w), N(w_1\cdots w_{m+1}), \ldots, N(w_{\ell-m}\cdots w_\ell),\\ &N(w_2\cdots w_{m+1}), \ldots, N(w_{\ell-m}\cdots w_{\ell-1}))\end{aligned}$$

(see Eq. (6.4.2)). There exists a covariance matrix Σ such that

$$\frac{1}{\sqrt{n}}(\underline{N}-\mathbf{E}(\underline{N})) \xrightarrow{\mathcal{D}} \mathcal{N}(0,\Sigma).$$

The next step is to use the δ-method (Theorem 6.8.5) to transfer this convergence to $f(\underline{N})$:

$$\frac{1}{\sqrt{n}}(f(\underline{N})-f(\mathbf{E}(\underline{N}))) \xrightarrow{\mathcal{D}} \mathcal{N}(0,\nabla\Sigma\nabla^t),$$

where $\nabla = \left(\frac{\partial f(x_1,\ldots,x_{2(\ell-m)})}{\partial x_j}\,|_{\mathbf{E}(\underline{N})}\right)_{j=1,\ldots,2(\ell-m)}$ is the partial derivative vector of f. Since $f(\mathbf{E}(\underline{N}))=0$, we finally obtain

$$\frac{1}{\sqrt{n}}\left(N(w)-\widehat{N}_m(w)\right) \xrightarrow{\mathcal{D}} \mathcal{N}(0,\nabla\Sigma\nabla^t).$$

However, this method does not easily provide an explicit formula for the asymptotic variance since the function f and its derivative depends on $\ell-m$.

An alternative method is given by the conditional approach. The principle is to work conditionally on the sufficient statistic $\mathcal{S}_m$ of the model Mm, namely the collection of counts $\{N(a_1\cdots a_{m+1}),\ a_1,\ldots,a_{m+1}\in\mathcal{A}\}$ and the first m letters of the sequence. One can derive both the conditional expectation $\mathbf{E}(N(w)\mid\mathcal{S}_m)$ and the conditional variance of $N(w)$. The key arguments are first that the conditional expectation is asymptotically equivalent to $\widehat{N}_m(w)$, leading to the asymptotic normality of $(N(w)-\mathbf{E}(N(w)\,|\,\mathcal{S}_m))/\sqrt{n}$, and second, that $n^{-1}\mathrm{Var}(N(w)\,|\,\mathcal{S}_m)$ has the limiting value $\sigma_m^2(w)$ with

$$\sigma_m^2(w) = \mu(w) + 2 \sum_{p \in \mathcal{P}(w),\, p \le \ell - m - 1} \mu(w^{(p)} w) + \mu(w)^2 \left(\sum_{a_1,\ldots,a_m} \frac{n(a_1 \cdots a_m \bullet)^2}{\mu(a_1 \cdots a_m)} \right.$$
$$\left. - \sum_{a_1,\ldots,a_{m+1}} \frac{n(a_1 \cdots a_{m+1})^2}{\mu(a_1 \cdots a_{m+1})} + \frac{1 - 2n(w_1 \cdots w_m \bullet)}{\mu(w_1 \cdots w_m)} \right), \tag{6.4.7}$$

where $n(\cdot)$ denotes the number of occurrences inside w, and $n(a_1 \cdots a_m \bullet)$ stands for $\sum_{b \in \mathcal{A}} n(a_1 \cdots a_m b)$. Since the conditional moment of order 4 of $N(w)/\sqrt{n}$ is bounded, it follows that

$$\frac{1}{\sqrt{n}} \left(N(w) - \widehat{N}_m(w) \right) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma_m^2(w)).$$

The overlapping structure of w clearly appears in the limiting variance. It is an exercise to verify that the limiting variances given by Theorem 6.4.2 and Equation (6.4.7) with $m = \ell - 2$ are identical.

Taking the phase into account Both the martingale approach and the conditional approach can be extended to the Mm-3 model (see Section 6.1 for definition and notation). When one wants to distinguish the occurrences of w in a coding DNA sequence according to a particular phase $k \in \{1, 2, 3\}$ (k represents the position of the word with respect to the codons), one is interested in the count $N(w, k)$ of w in phase k in $X_1 \cdots X_n$; recall that the word phase is the phase of its last letter. Here we state the result in the maximal model.

Theorem 6.4.3. *Assume $\underline{X} = (X_i)_{i \in \mathbb{Z}}$ is a stationary $(\ell - 2)$-order Markov chain on $\mathcal{A}$ with transition probabilities $\pi_k(a_1 \cdots a_{\ell-2}, b)$ and stationary distribution $\mu(a_1 \cdots a_{\ell-2}, k)$, $a_1, \ldots, a_{\ell-2}, b \in \mathcal{A}$, $k \in \{1, 2, 3\}$. As $n \to \infty$, we have*

$$\frac{1}{\sqrt{n}} \left(N(w, k) - \frac{N(w^-, k-1) N({}^-w, k)}{N({}^-w^-, k-1)} \right) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma_{\ell-2}^2(w, k))$$

with

$$\sigma_{\ell-2}^2(w, k) = \frac{\mu(w, k)}{\mu({}^-w^-, k-1)^2} \Big(\mu({}^-w^-, k-1) - \mu({}^-w, k) \Big)$$
$$\times \Big(\mu({}^-w^-, k-1) - \mu(w^-, k-1) \Big)$$

and

$$\mu(w^-, k-1) = \mu(w_1 \cdots w_{\ell-2}, k-2) \pi_{k-1}(w_1 \cdots w_{\ell-2}, w_{\ell-1})$$
$$\mu({}^-w, k) = \mu({}^-w^-, k-1) \pi_k({}^-w^-, w_\ell)$$
$$\mu(w, k) = \mu(w^-, k-1) \pi_k({}^-w^-, w_\ell).$$

Error bound for the approximation Using Stein's method for normal approximations, namely Theorem 6.8.1, provides a bound on the distance to the normal distribution; however, it does not take the estimation of parameters into account.

Recall $v^2 = \mathrm{Var}(N(w))$ from (6.4.1), and α given in (6.1.1). One has the following result.

THEOREM 6.4.4. *Assume $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ is a stationary 1-order Markov chain. Let w be a word of length ℓ and $Z \sim \mathcal{N}((n-\ell+1)\mu(w), v^2)$. There are constants c and C_1, C_2, C_3 such that*

$$|\mathbf{P}(N(w) \leq x) - \mathbf{P}(Z \leq x)| \leq c \min_{\ell \leq s \leq \frac{n}{2}} B_s,$$

where

$$\begin{aligned} B_s = & \, 2(4s-3)v^{-1} + 2n(2s-1)(4s-3)v^{-3}(|\log v^{-1}| + \log n) \\ & + C_1 n v^{-1}\mu(w)|\alpha|^{s-\ell+1} \\ & + C_2(|\log v^{-1}| + \log n)(2s-1)|\alpha|^{s-\ell+1} \\ & + C_3(|\log v^{-1}| + \log n)(n-2s+1)n\mu^2(w)v^{-2}|\alpha|^{s-\ell+1}. \end{aligned}$$

The multivariate generalization will be presented in Theorem 6.6.1, where the explicit forms of the constants C_1, C_2, and C_3 will be given.

6.4.4. Asymptotic distribution: the Poisson regime

In the previous section, we showed that the count $N(w)$ of a word w in a random sequence of length n can be approximated by a Gaussian distribution for large n. This Gaussian approximation is in fact not good when the expected count $(n-\ell+1)\mu(w)$ is very small, meaning that w is a rare word. Poisson approximations are appropriate for counts of rare events. As an illustration, it is well-known that a sum of independent Bernoulli variables can be either approximated by a Gaussian distribution or a Poisson distribution, depending on the asymptotic behavior of the expected value.

When the sequence letters are independent, Poisson and compound Poisson approximations for $N(w)$ have been widely studied in the literature. As we will see, a Poisson distribution is not satisfactory for periodic words because of possible overlaps; a compound Poisson distribution is proposed. Two classes of tools can be used: generating functions, which do not provide any approximation error, and the Chen-Stein method, which gives a bound for the total variation distance between the two distributions (see Section 6.8.2 for details). In this section, we chose to present the Chen-Stein approach under a first-order Markovian model with known parameters; generalizations to higher order and to estimated parameters are presented at the end of the section. No assumption is made on the overlapping structure of the word w.

We assume that $\underline{X} = (X_i)_{i\in\mathbb{Z}}$ is a stationary first-order Markov chain on $\mathcal{A}$, with transition probabilities $\pi(a, b)$ and stationary distribution $\mu(a)$,

$a, b \in \mathcal{A}$. Let $w = w_1 \cdots w_\ell$ be a word of length ℓ on $\mathcal{A}$. Here, $Y_i = Y_i(w) = \mathbb{I}\{w$ starts at position i in $\underline{X}\}$ and $\mu(w) = \mathbf{E}(Y_i(w))$. Moreover, we make the *rare word assumption* $n\mu(w) = O(1)$. Note that $n\mu(w) = O(1)$ also means $\ell = O(\log n)$.

Applying Theorem 6.8.2 to the Bernoulli variables Y_i, we obtain a bound $b_1 + b_2 + b_3$ for the total variation distance between the distribution of $N(w)$ and the Poisson distribution with mean $(n - \ell + 1)\mu(w)$ that does not converge to 0 under the rare word assumption. The problem comes from the b_2 term and the possible overlaps of periodic words. Indeed, let w be a periodic word; its set of periods $\mathcal{P}(w)$ is not empty. Take $B_i = \{i - 2\ell + 1, \ldots, i + 2\ell - 1\}$ for the neighborhood of $i \in I = \{1, \ldots, n - \ell + 1\}$; then b_1 and b_3 tend to 0 as $n \to +\infty$. We obtain

$$b_2 := \sum_{i \in I} \sum_{j \in B_i \setminus \{i\}} \mathbf{E}(Y_i Y_j) = 2(n - \ell + 1) \sum_{p \in \mathcal{P}(w)} \mu(w^{(p)} w) + O(n\ell\mu^2(w));$$

this quantity can be of order $O(1)$ if $\mathcal{P}(w)$ contains small periods p. The Poisson approximation is however valid for the count of non-periodic words because the set of periods is empty. For periodic words, the crucial argument is to consider clumps, as by definition they cannot overlap. We first prove that the declumped count $\widetilde{N}(w)$ can be approximated by a Poisson distribution with mean $(n - \ell + 1)\widetilde{\mu}(w)$ (see Eq. (6.2.7)) by applying Theorem 6.8.2 to the Bernoulli variables $\widetilde{Y}_i(w)$ defined in (6.2.5). For simplicity, the variables $\widetilde{Y}_i(w)$ are denoted by $\widetilde{Y}_i$. In the next section we prove a compound Poisson approximation for $N(w)$.

Poisson approximation for the declumped count Our aim is to approximate the vector $\underline{\widetilde{Y}} = (\widetilde{Y}_i(w))_{i \in I}$ of Bernoulli variables by a vector $\underline{Z} = (Z_i)_{i \in I}$ with independent Poisson coordinates of mean $\mathbf{E}(Z_i) = \mathbf{E}(\widetilde{Y}_i(w)) = \widetilde{\mu}(w)$, where $\widetilde{\mu}(\cdot)$ is defined in (6.2.7). To apply Theorem 6.8.2, we choose the following neighborhood of $i \in I$:

$$B_i := \{j \in I : |j - i| \leq 3\ell - 3\}.$$

The neighborhood is such that, for j not in B_i, there are no letters X_h common to $\widetilde{Y}_i$ and $\widetilde{Y}_j$, and moreover, the X_h's defining $\widetilde{Y}_i$ and those defining $\widetilde{Y}_j$ are separated by at least ℓ positions. It is important to consider a lag converging to infinity with n since it leads to the exponential decay of the b_3 term given by Theorem 6.8.2 as we will see below. Deriving a bound for the total variation distance between $\underline{\widetilde{Y}}$ and $\underline{Z}$ consists of bounding the quantities b_1, b_2 and b_3 given in (6.8.1), (6.8.2) and (6.8.3). Bounding b_1 presents no difficulty:

$$b_1 := \sum_{i \in I} \sum_{j \in B_i} \mathbf{E}(\widetilde{Y}_i)\mathbf{E}(\widetilde{Y}_j) \leq (n - \ell + 1)(6\ell - 5)\widetilde{\mu}^2(w) = O\left(\frac{\log n}{n}\right).$$

Since clumps of w do not overlap in the sequence, $\widetilde{Y}_i\widetilde{Y}_j = 0$ for $|j-i| < \ell$. Therefore, we get

$$b_2 := \sum_{i\in I}\sum_{j\in B_i\setminus\{i\}} \mathbf{E}(\widetilde{Y}_i\widetilde{Y}_j) \le 2\sum_{i\in I}\sum_{j=i+\ell}^{i+3\ell-3} \mathbf{E}(\widetilde{Y}_i\widetilde{Y}_j)$$

using the symmetry of B_i. Now we have

$$\mathbf{E}(\widetilde{Y}_i\widetilde{Y}_j) \le \mathbf{E}(\widetilde{Y}_iY_j) = \widetilde{\mu}(w)\Pi^{j-i-\ell+1}(w_\ell, w_1)\frac{\mu(w)}{\mu(w_1)}$$

and

$$b_2 \le \frac{2}{\mu(w_1)}(n-\ell+1)\widetilde{\mu}(w)\mu(w)\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell, w_1) = O\left(\frac{\log n}{n}\right).$$

Bounding b_3 is a little more involved but we give all the steps because the same technique is used for the compound Poisson approximation of the count and will not be described in detail there. By definition we have

$$b_3 := \sum_{i\in I}\mathbf{E}|\mathbf{E}(\widetilde{Y}_i - \mathbf{E}(\widetilde{Y}_i) \mid \sigma(\widetilde{Y}_j, j\notin B_i))|\,.$$

Since $\sigma(\widetilde{Y}_j, j\notin B_i) \subset \sigma(X_1,\ldots,X_{i-2\ell+1}, X_{i+2\ell-1},\ldots,X_n)$, properties of conditional expectation and the Markov property give

$$\begin{aligned} b_3 &\le \sum_{i\in I}\mathbf{E}|\mathbf{E}(\widetilde{Y}_i - \mathbf{E}(\widetilde{Y}_i) \mid X_{i-2\ell+1}, X_{i+2\ell-1})| \\ &\le \sum_{i\in I}\sum_{x,y\in\mathcal{A}} |\mathbf{E}(\widetilde{Y}_i - \mathbf{E}(\widetilde{Y}_i) \mid X_{i-2\ell+1}=x, X_{i+2\ell-1}=y)| \\ &\qquad\qquad \times\mathbf{P}(X_{i-2\ell+1}=x, X_{i+2\ell-1}=y). \end{aligned}$$

To evaluate the right-hand term, we introduce the set of possible words of length $\ell-1$ preceding a clump of w:

$$\mathcal{G}(w) = \{g = g_1\cdots g_{\ell-1} : \text{ for all } p\in\mathcal{P}(w), g_{\ell-p}\cdots g_{\ell-1}\ne w^{(p)}\}\,. \tag{6.4.8}$$

Thus a clump of w starts at position i in $(X_i)_{i\in\mathbb{Z}}$ if and only if one of the words gw, $g\in\mathcal{G}(w)$, starts at position $i-\ell+1$. Therefore, we can write

$$\widetilde{Y}_i(w) = \sum_{g\in\mathcal{G}(w)} Y_{i-\ell+1}(gw). \tag{6.4.9}$$

This gives

$$\begin{aligned} &b_3 \\ &\le \sum_{i\in I}\sum_{x,y\in\mathcal{A}}\sum_{g\in\mathcal{G}(w)} |\mathbf{E}(Y_{i-\ell+1}(gw) - \mathbf{E}(Y_{i-\ell+1}(gw)) \mid X_{i-2\ell+1}=x, X_{i+2\ell-1}=y)| \end{aligned}$$

$$
\begin{aligned}
&\qquad\qquad \times\mathbf{P}(X_{i-2\ell+1}=x, X_{i+2\ell-1}=y) \\
&= \sum_{i\in I}\sum_{x,y\in\mathcal{A}}\sum_{g\in\mathcal{G}(w)} |\mathbf{P}(X_{i-2\ell+1}=x, Y_{i-\ell+1}(gw)=1, X_{i+2\ell-1}=y) \\
&\qquad\qquad -\mu(gw)\mathbf{P}(X_{i-2\ell+1}=x, X_{i+2\ell-1}=y)| \\
&= \sum_{i\in I}\sum_{x,y\in\mathcal{A}}\sum_{g\in\mathcal{G}(w)} \left|\mu(x)\Pi^\ell(x,g_1)\frac{\mu(gw)}{\mu(g_1)}\Pi^\ell(w_\ell,y) - \mu(gw)\mu(x)\Pi^{4\ell-2}(x,y)\right|.
\end{aligned}
$$

We now use the diagonalization (6.1.3) and (6.1.4), with α given in (6.1.1), yielding

$$
\begin{aligned}
b_3 &\le (n-\ell+1)|\alpha|^\ell \sum_{g\in\mathcal{G}(w)}\mu(gw)\sum_{x,y\in\mathcal{A}}\mu(x)\left|\frac{1}{\mu(g_1)}\sum_{(t,t')}\frac{\alpha_t^\ell\alpha_{t'}^\ell}{\alpha^\ell}Q_t(x,g_1)Q_{t'}(w_\ell,y)\right. \\
&\qquad\qquad \left. -\sum_{t=1}^{|\mathcal{A}|}\frac{\alpha_t^{4\ell-2}}{\alpha^\ell}Q_t(x,y)\right| \\
&= (n-\ell+1)|\alpha|^\ell \sum_{g\in\mathcal{G}(w)}\mu(gw)\sum_{x,y\in\mathcal{A}}\mu(x)\left|\frac{1}{\mu(g_1)}\sum_{(t,t')\ne(1,1)}\frac{\alpha_t^\ell\alpha_{t'}^\ell}{\alpha^\ell}Q_t(x,g_1)Q_{t'}(w_\ell,y)\right. \\
&\qquad\qquad \left. -\sum_{t=2}^{|\mathcal{A}|}\frac{\alpha_t^{4\ell-2}}{\alpha^\ell}Q_t(x,y)\right| \\
&\le (n-\ell+1)|\alpha|^\ell \sum_{g\in\mathcal{G}(w)}\mu(gw)\gamma(\ell,w_\ell),
\end{aligned}
$$

where

$$
\gamma(\ell,a) = \max_{b\in\mathcal{A}}\sum_{x,y\in\mathcal{A}}\mu(x)\left|\frac{1}{\mu(b)}\sum_{(t,t')\ne(1,1)}\frac{\alpha_t^\ell\alpha_{t'}^\ell}{\alpha^\ell}Q_t(x,b)Q_{t'}(a,y) - \sum_{t=2}^{|\mathcal{A}|}\frac{\alpha_t^{4\ell-2}}{\alpha^\ell}Q_t(x,y)\right|.
$$

Note that $\gamma(\ell,w_\ell)=O(1)$. From (6.4.8) we have $\sum_{g\in\mathcal{G}(w)}\mu(gw)=\widetilde{\mu}(w)$ and

$$
b_3 \le (n-\ell+1)\widetilde{\mu}(w)\gamma(\ell,w_\ell)|\alpha|^\ell = O(|\alpha|^\ell).
$$

We have proved the next theorem.

THEOREM 6.4.5. *Let $\underline{Z}=(Z_i)_{i\in I}$ be independent Poisson variables with expectation $\mathbf{E}(Z_i)=\mathbf{E}(\widetilde{Y}_i(w))=\widetilde{\mu}(w)$. We have*

$$
\begin{aligned}
d_{\mathrm{TV}}\left(\mathcal{L}(\underline{\widetilde{Y}}),\mathcal{L}(\underline{Z})\right) \le (n-\ell+1)\widetilde{\mu}(w)\Bigg\{&(6\ell-5)\widetilde{\mu}(w)+\gamma(\ell,w_\ell)|\alpha|^\ell \\
&+\frac{2}{\mu(w_1)}\mu(w)\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell,w_1)\Bigg\}.
\end{aligned}
$$

The declumped count $\widetilde{N}(w)$ can be approximated by $\widetilde{N}_{\mathrm{inf}}(w) := \sum_{i\in I} \widetilde{Y}_i(w)$ since

$$d_{\mathrm{TV}}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{L}(\widetilde{N}_{\mathrm{inf}}(w))\right) \le \mathbf{P}(\widetilde{N}(w) \ne \widetilde{N}_{\mathrm{inf}}(w)) \\ \le (\ell-1)(\mu(w)-\widetilde{\mu}(w)). \qquad (6.4.10)$$

Using the triangle inequality leads to the following corollary.

COROLLARY 6.4.6. *Let Z be a Poisson variable with expectation $\mathbf{E}(Z) = (n-\ell+1)\widetilde{\mu}(w)$. We have*

$$d_{TV}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{L}(Z)\right) \le (n-\ell+1)\widetilde{\mu}(w)\Bigg\{(6\ell-5)\widetilde{\mu}(w) + \gamma(\ell, w_\ell)|\alpha|^\ell \\ + \frac{2}{\mu(w_1)}\mu(w)\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell, w_1)\Bigg\} \\ +(\ell-1)(\mu(w)-\widetilde{\mu}(w)).$$

Estimation of the parameters When the transition probabilities are unknown and can only be estimated from the observed sequence, we need to evaluate the total variation distance between the word count distribution and the distribution of $\sum_{k\ge1} kZ'_k$, the Z'_k's being independent Poisson variables with expectation $(n-\ell+1)\widehat{\widetilde{\mu}}_k(w)$, where $\widehat{\widetilde{\mu}}_k(w)$ is the observed value of the plug-in maximum likelihood estimator of $\widetilde{\mu}_k(w)$. Similarly, we want to know the total variation distance between the declumped count, $\widetilde{N}(w)$, and the Poisson variable with expectation $(n-\ell+1)\widehat{\widetilde{\mu}}(w)$. For this we use the triangle inequality and the fact that the total variation distance between two Poisson variables with expectation λ and λ' is less than $|\lambda-\lambda'|$:

$$d_{\mathrm{TV}}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{P}\mathrm{o}((n-\ell+1)\widehat{\widetilde{\mu}}(w))\right) \le d_{\mathrm{TV}}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{P}\mathrm{o}((n-\ell+1)\widetilde{\mu}(w))\right) \\ +(n-\ell+1)|\widehat{\widetilde{\mu}}(w)-\widetilde{\mu}(w)|.$$

Using the Law of Iterated Logarithm for Markov chains and Equation (6.2.4), one can show that

$$\widehat{\mu}(w) = \mu(w)\left(1+O\left(\frac{\ell\sqrt{\log\log n}}{\sqrt{n}}\right)\right) \qquad \text{almost surely (a.s.)}$$

Under the rare word condition $n\mu(w) = O(1)$, we get

$$n\widehat{\mu}(w) - n\mu(w) = O\left(\frac{\ell\sqrt{\log\log n}}{\sqrt{n}}\right) \qquad \text{a.s.}$$

Now, using Equation (6.2.7), we obtain

$$n\widehat{\widetilde{\mu}}(w) - n\widetilde{\mu}(w) = O\left(\frac{\ell^2\sqrt{\log\log n}}{\sqrt{n}}\right) \qquad \text{a.s.}$$

This quantity converges to zero as $n \to \infty$, because the rare word condition implies that $\ell = O(\log n)$. Thus,

$$\begin{aligned} &d_{\mathrm{TV}}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{P}\mathrm{o}\big((n-\ell+1)\widehat{\widetilde{\mu}}(w)\big)\right) \\ &\leq d_{\mathrm{TV}}\left(\mathcal{L}(\widetilde{N}(w)), \mathcal{P}\mathrm{o}\big((n-\ell+1)\widetilde{\mu}(w)\big)\right) + O\left(\frac{\ell^2\sqrt{\log\log n}}{\sqrt{n}}\right). \end{aligned}$$

The approximation follows from Corollary 6.4.8.

We do not have an explicit bound for this additional error term. However, for long sequences the error term due to the maximum-likelihood estimation will be small compared to the bound on the Poisson approximation error.

6.4.5. Asymptotic distribution: the Compound Poisson regime

Here we present two approaches for a compound Poisson approximation for the count. Firstly, such an approximation can be derived using a Poisson process approximation for the Bernoulli variables $\widetilde{Y}_{i,k}(w)$ defined in (6.2.9) and by using that $N(w)$ is asymptotically equivalent to $\sum_{i\in I}\sum_{k\geq 1} k\widetilde{Y}_{i,k}(w)$ in probability. For simplicity, the variables $\widetilde{Y}_{i,k}(w)$ are denoted by $\widetilde{Y}_{i,k}$. Secondly, a direct approximation for $N(w)$ can be obtained using Stein's method for compound Poisson approximation. The second method yields better bounds on the approximation, whereas the first method is easier to generalize to multivariate results, as will be shown in Section 6.6.

Compound Poisson approximation via Poisson process To approximate the distribution of the count $N(w)$, we first use that $N(w)$ is asymptotically equivalent to $N_{\mathrm{inf}}(w) := \sum_{i=1}^{n-\ell+1}\sum_{k\geq 1} k\widetilde{Y}_{i,k}$ in probability:

$$\begin{aligned} d_{\mathrm{TV}}\left(\mathcal{L}(N(w)), \mathcal{L}(N_{\mathrm{inf}}(w))\right) &\leq \mathbf{P}(N(w) \neq N_{\mathrm{inf}}(w)) \\ &\leq 2(\ell-1)(\mu(w) - \widetilde{\mu}(w)). \qquad (6.4.11) \end{aligned}$$

Our goal is now to approximate the vector $(\widetilde{Y}_{i,k})_{(i,k)\in I}$, $I = \{1,\ldots,n-\ell+1\}\times\{1,2,\ldots\}$, of Bernoulli variables by a vector $(Z_{i,k})_{(i,k)\in I}$ with independent Poisson coordinates of expectation $\mathbf{E}(Z_{i,k}) = \mathbf{E}(\widetilde{Y}_{i,k}) = \widetilde{\mu}_k(w)$ where $\widetilde{\mu}_k(\cdot)$ is given in Equation (6.2.10). The neighborhood $B_{i,k}$ of (i,k) is such that, for (j,k') not in $B_{i,k}$, the letters X_h's defining $\widetilde{Y}_{i,k}$ and those defining $\widetilde{Y}_{j,k}$ are separated by at least ℓ positions. Since $\widetilde{Y}_{i,k}$ can be described by at most $X_{i-\ell+1},\ldots,X_{i+(k+1)(\ell-1)}$, we consider

$$B_{i,k} := \left\{(j,k') \in I : -(k'+3)(\ell-1) \leq j-i \leq (k+3)(\ell-1)\right\}.$$

We bound successively the quantities given in (6.8.1), (6.8.2) and (6.8.3). By definition

$$b_1 := \sum_{(i,k)\in I}\ \sum_{(j,k')\in B_{i,k}} \mathbf{E}(\widetilde{Y}_{i,k})\mathbf{E}(\widetilde{Y}_{j,k'})$$

$$\leq \sum_{i=1}^{n-\ell+1} \sum_{k\geq 1} \sum_{k'\geq 1} \sum_{j=i-(k'+3)(\ell-1)}^{i+(k+3)(\ell-1)} \widetilde{\mu}_k(w)\widetilde{\mu}_{k'}(w)$$
$$\leq (n-\ell+1) \sum_{k\geq 1} \sum_{k'\geq 1} \big((k+k'+6)(\ell-1)+1\big)\widetilde{\mu}_k(w)\widetilde{\mu}_{k'}(w).$$

From (6.2.7) and (6.2.10), we use that

$$\sum_{k\geq 1} \widetilde{\mu}_k(w) = \widetilde{\mu}(w)\,, \tag{6.4.12}$$

$$\sum_{k\geq 1} k\widetilde{\mu}_k(w) = \mu(w)\,, \tag{6.4.13}$$

to obtain

$$b_1 \leq (n-\ell+1)\left(2(\ell-1)\widetilde{\mu}(w)\mu(w) + (6\ell-5)\widetilde{\mu}(w)^2\right).$$

The b_2 term involves products such as $\widetilde{Y}_{i,k}\widetilde{Y}_{j,k'}$ with $(j,k') \in B_{i,k}$. Since a k-clump of w at position i cannot overlap a k'-clump of w, many of these products are zero. To identify them, we need to describe in more detail the compound words $c \in \mathcal{C}_k(w)$ and $c' \in \mathcal{C}_{k'}(w)$ that may occur at positions i and j. For this purpose, we introduce the set of words of length $\ell-1$ that can follow a clump of w:

$$\mathcal{D}(w) = \{d = d_1 \cdots d_{\ell-1} : \forall p \in \mathcal{P}(w), d_1 \cdots d_p \neq w_{\ell-p+1} \cdots w_\ell\}.$$

Therefore, we can write

$$\widetilde{Y}_{i,k}(w) = \sum_{g\in\mathcal{G}(w), c\in\mathcal{C}_k(w), d\in\mathcal{D}(w)} Y_{i-\ell+1}(gCd). \tag{6.4.14}$$

For convenience, we write $\sum_{gcd}$ for the sum over $g \in \mathcal{G}(w)$, $c \in \mathcal{C}_k(w)$, $d \in \mathcal{D}(w)$, and, similarly, $\sum_{g'c'd'}$ for the sum over $g' \in \mathcal{G}(w)$, $c' \in \mathcal{C}_{k'}(w)$ and $d' \in \mathcal{D}(w)$. This gives

$$b_2 := \sum_{(i,k)\in I} \sum_{(j,k')\in I\setminus\{(i,k)\}} \mathbf{E}(\widetilde{Y}_{i,k}\widetilde{Y}_{j,k'})$$
$$= \sum_{i=1}^{n-\ell+1} \sum_{k\geq 1} \sum_{k'\geq 1} \sum_{gcd} \sum_{g'c'd'} \sum_{j=i-(k'+3)(\ell-1)}^{i+(k+3)(\ell-1)} \mathbf{E}(Y_{i-\ell+1}(gcd)Y_{j-\ell+1}(g'c'd')).$$

For $i-|c'| < j < i+|c|$, we have that $Y_{i-\ell+1}(gcd)Y_{j-\ell+1}(g'c'd') = 0$ because clumps do not overlap. We distinguish two cases:

(1) $g'c'd'$ at position $j-\ell+1$ overlaps gcd at position $i-\ell+1$ (this is only possible over at most $2(\ell-1)$ letters); that is, for

$$j \in \{i-|c'|-2\ell+3, \ldots, i-|c'|\} \cup \{i+|c|, \ldots, i+|c|+2\ell-3\}\,;$$

let b_{21} denote the associated term.

(2) $g'c'd'$ at position $j-\ell+1$ does not overlap gcd at position $i-\ell+1$; that is, for

$j \in \{i-(k'+3)(\ell-1), \ldots, i-|c'|-2\ell+2\} \cup \{i+|c|+2\ell-2, \ldots, i+(k+3)(\ell-1)\}$;

let b_{22} denote the associated term.

By symmetry, we have

$$b_{21} \le 2 \sum_{i=1}^{n-\ell+1} \sum_{k\ge 1} \sum_{k'\ge 1} \sum_{gcd} \sum_{g'c'd'} \sum_{j=i+|c|}^{i+|c|+2\ell-3} \mathbf{E}(Y_{i-\ell+1}(gCd) Y_{j-\ell+1}(g'C'd')) .$$

Summing over k', g', c' and d' gives

$$b_{21} \le 2 \sum_{i=1}^{n-\ell+1} \sum_{k\ge 1} \sum_{gcd} \sum_{j=i+|c|}^{i+|c|+2\ell-3} \mathbf{E}(Y_{i-\ell+1}(gcd) \widetilde{Y}_j(w)) ;$$

now, summing over d and using that $\widetilde{Y}_j(w) \le Y_j(w)$ leads to

$$b_{21} \le 2 \sum_{i=1}^{n-\ell+1} \sum_{k\ge 1} \sum_{gc} \sum_{j=i+|c|}^{i+|c|+2\ell-3} \mathbf{E}(Y_{i-\ell+1}(gc) Y_j(w)) .$$

An occurrence of gc at position $i-\ell+1$ does not overlap an occurrence of w at position $j \ge i+|c|$; thus it follows that

$$\mathbf{E}(Y_{i-\ell+1}(gc) Y_j(w)) = \mu(gc) \Pi^{j-i-|c|+1}(w_\ell, w_1) \frac{\mu(w)}{\mu(w_1)} ,$$

and

$$b_{21} \le 2(n-\ell+1) \frac{\mu(w)}{\mu(w_1)} \sum_{s=1}^{2\ell-2} \Pi^s(w_\ell, w_1) \sum_{k\ge 1} \sum_{gc} \mu(gc).$$

Finally, note that

$$\sum_{k\ge 1} \sum_{gc} \mu(gc) = \sum_{k\ge 1} \sum_{k^*\ge k} \widetilde{\mu}_{k^*}(w) = \sum_{k^*\ge 1} k^* \widetilde{\mu}_{k^*}(w) = \mu(w) ,$$

which leads to

$$b_{21} \le 2(n-\ell+1) \frac{\mu^2(w)}{\mu(w_1)} \sum_{s=1}^{2\ell-2} \Pi^s(w_\ell, w_1) = O\left(\frac{\log n}{n}\right) .$$

The b_{22} term is easier to bound and we get

$$b_{22} \le 2(n-\ell+1) \frac{\widetilde{\mu}(w)}{\mu_{\min}} ((\ell-2)\mu(w) + \widetilde{\mu}(w)) = O\left(\frac{\log n}{n}\right) ,$$

where $\mu_{\min}$ is the smallest value of $\{\mu(a), a \in \mathcal{A}\}$.

Combining these bounds, we have

$$b_2 \leq 2(n-\ell+1)\frac{\mu^2(w)}{\mu(w_1)}\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell, w_1) + 2(n-\ell+1)\frac{\widetilde{\mu}(w)}{\mu_{\min}}\left((\ell-2)\mu(w) + \widetilde{\mu}(w)\right).$$

Bounding b_3 consists of following the different steps previously detailed for the declumped count and using the decomposition (6.4.14) instead of (6.4.9). Since there is no interest in repeating this technical part, we just give the bound of b_3 and state the theorem:

$$b_3 \leq (n-\ell+1)\widetilde{\mu}(w)\gamma_2(\ell)|\alpha|^\ell$$

with

$$\gamma_2(\ell) = \sum_{x,y\in\mathcal{A}}\mu(x)\max_{a,b\in\mathcal{A}}\left(\frac{1}{\mu(b)}\sum_{(t,t')\neq(1,1)}\left|\frac{\alpha_t^\ell\alpha_{t'}^\ell}{\alpha^\ell}Q_t(x,b)Q_{t'}(a,y)\right| + \sum_{t=2}^{|\mathcal{A}|}\left|\frac{\alpha_t^{5\ell-3}}{\alpha^\ell}Q_t(x,y)\right|\right).$$

THEOREM 6.4.7. *Let $(Z_{i,k})_{(i,k)\in I}$ be independent Poisson variables with expectation $\mathbf{E}(Z_{i,k}) = \mathbf{E}(\widetilde{Y}_{i,k}(w)) = \widetilde{\mu}_k(w)$. With the previous notation, we have*

$$\begin{aligned} &d_{\mathrm{TV}}\Big(\mathcal{L}\big((\widetilde{Y}_{i,k}(w))_{(i,k)\in I}\big), \mathcal{L}\big((Z_{i,k})_{(i,k)\in I}\big)\Big) \\ &\leq (n-\ell+1)\widetilde{\mu}(w)\Big(2(\ell-1)\mu(w) + (6\ell-5)\widetilde{\mu}(w) + \gamma_2(\ell)|\alpha|^\ell\Big) \\ &\quad +2(n-\ell+1)\left\{\frac{\mu^2(w)}{\mu(w_1)}\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell, w_1) + \frac{\widetilde{\mu}(w)}{\mu_{min}}\big((\ell-2)\mu(w) + \widetilde{\mu}(w)\big)\right\}. \end{aligned}$$

From the total variation distance properties, we have

$$d_{\mathrm{TV}}\Big(\mathcal{L}\big(\sum_{(i,k)\in I}k\widetilde{Y}_{i,k}\big), \mathcal{L}\big(\sum_{(i,k)\in I}kZ_{i,k}\big)\Big) \leq d_{\mathrm{TV}}\Big(\mathcal{L}\big((\widetilde{Y}_{i,k}(w))_{(i,k)\in I}\big), \mathcal{L}\big((Z_{i,k})_{(i,k)\in I}\big)\Big).$$

Since the $Z_{i,k}$'s are independent Poisson variables, $\sum_{(i,k)\in I}kZ_{i,k}$ has the same distribution as $\sum_{k\geq1}kZ_k$, where the Z_k's are independent Poisson variables with expectation $(n-\ell+1)\widetilde{\mu}_k(w)$. Note that the latter has a compound Poisson distribution with parameters $\big((n-\ell+1)\widetilde{\mu}(w), (\widetilde{\mu}_k(w)/\widetilde{\mu}(w))_k\big)$. Because of the expressions of $\widetilde{\mu}(w)$ and $\widetilde{\mu}_k(w)$ given by (6.2.7) and (6.2.10), this compound Poisson distribution reduces to a Polýa-Aeppli distribution. Using the triangle inequality leads to the following corollary.

Corollary 6.4.8. *Let $(Z_k)_{k\geq 1}$ be independent Poisson variables with expectation $\mathbf{E}(Z_k) = (n-\ell+1)\widetilde{\mu}_k(w)$. Let*

$$CP = CP\left((n-\ell+1)\mu(w)(1-A(w)), \left((1-A(w))A^{k-1}(w)\right)_{k\geq 1}\right) \quad (6.4.15)$$

denote the compound Poisson distribution of $\sum_{k\geq 1} kZ_k$. With the previous notation, we have

$$\begin{aligned} d_{TV}(\mathcal{L}(N(w)), CP) \leq \\ & (n-\ell+1)\widetilde{\mu}(w)\Big(2(\ell-1)\mu(w) + (6\ell-5)\widetilde{\mu}(w) + \gamma_2(\ell)|\alpha|^\ell\Big) \\ & +2(n-\ell+1)\left\{\frac{\mu^2(w)}{\mu(w_1)}\sum_{s=1}^{2\ell-2}\Pi^s(w_\ell, w_1) + \frac{\widetilde{\mu}(w)}{\mu_{min}}\big((\ell-2)\mu(w) + \widetilde{\mu}(w)\big)\right\} \\ & +2(\ell-1)(\mu(w) - \widetilde{\mu}(w)) \\ = O\left(\frac{\log n}{n}\right). \end{aligned}$$

Such a bound on the total variation distance between, for instance, the word count distribution and the associated compound Poisson distribution has the great advantage of providing confidence intervals (see Section 6.8.2). Indeed, using notation from Corollary 6.4.8, for all $t \in \mathbb{R}$, we have

$$\left|\mathbf{P}(N(w) \geq t) - \mathbf{P}\left(\sum_{k\geq 1} kZ_k \geq t\right)\right| \leq d_{\mathrm{TV}}\left(\mathcal{L}(N(w)), \mathcal{L}\left(\sum_{k\geq 1} kZ_k\right)\right).$$

Direct compound Poisson approximation Empirically, often a compound Poisson approximation also gives good results when the underlying words are not so rare, indicating that the theoretical bounds are not sharp. Using the direct compound Poisson approximation Theorem 6.8.4, it is possible to obtain improved bounds for $N(w)$. For this, choose as neighborhoods in Theorem 6.8.4

$$B(i,k) = \{(j,k') : -(k'-2)(\ell-1) - r + 1 \leq j - i \leq (k+2)(\ell-1) + r - 1\},$$

where $r \geq 1$ can be chosen. In Theorem 6.4.5 we had $r = \ell$. Recall (6.2.10), ρ from (6.1.5), Γ from (6.5.5), and CP from (6.4.15). One obtains the following result.

Theorem 6.4.9. *If $A(w) \leq \frac{1}{5}$, then*

$$d_{TV}(\mathcal{L}(N(w)), CP) \leq \frac{1-A(w)}{1-5A(w)}\left(\Delta_1 + \sqrt{(n-\ell+1)\mu(w)}\Delta_0\right) + 2(\ell-1)\mu(w),$$

where

$$\Delta_0 = 2\rho^r \left(2 + 3\rho^{3(\ell-1)+r} + 2\rho^r\right),$$

$$\Delta_1 = 2\mu(w) \left\{ 3(\ell-1) + r + 2(\ell-1)\frac{A(w)}{1-A(w)} + \Gamma\left(\frac{2A(w)(\ell-1-p_0)}{(1-A(w))^3} + \frac{2(\ell-1)+r-1}{(1-A(w))^2}\right)\right\},$$

and p_0 is the shortest period of w. The value r can be chosen to minimize the estimates.

Estimation of the parameters When estimating the parameters, as in Subsection 6.4.4, the total variation distance between the two compound Poisson distributions is bounded by

$$d_{\mathrm{TV}}\left(\mathcal{L}\left(\sum_{k\geq 1} kZ_k\right), \mathcal{L}\left(\sum_{k\geq 1} kZ'_k\right)\right) \leq \sum_{k\geq 1} |n\widehat{\widetilde{\mu}}_k(w) - n\widetilde{\mu}_k(w)|.$$

Using Equation (6.2.10), this quantity tends to zero as $n \to \infty$ when $n\mu(w) = O(1)$.

Again, for long sequences the error term due to the maximum-likelihood estimation will be small compared to the bound on the compound Poisson approximation error.

Generalization to Mm Let us now assume that the sequence $(X_i)_{i\in\mathbb{Z}}$ is a m-order Markov chain on the alphabet $\mathcal{A}$, with transition probabilities $\pi(a_1 \cdots a_m, a_{m+1})$, $a_1, \cdots, a_{m+1} \in \mathcal{A}$. The basic idea is to rewrite the sequence over the alphabet $\mathcal{A}^m$ using the embedding (6.1.6),

$$\mathbb{X}_i = X_i X_{i+1} \cdots X_{i+m-1},$$

so that the sequence $(\mathbb{X}_i)_{i\in\mathbb{Z}}$ is a first-order Markov chain on $\mathcal{A}^m$ with transition probabilities ($\mathbb{A} = a_1 \cdots a_m \in \mathcal{A}^m$, $\mathbb{B} = b_1 \cdots b_m \in \mathcal{A}^m$)

$$\Pi(\mathbb{A}, \mathbb{B}) = \begin{cases} \pi(a_1 \cdots a_m, b_m) & \text{if } a_2 \cdots a_m = b_1 \cdots b_{m-1} \\ 0 & \text{otherwise.} \end{cases}$$

Denote by $\mathbb{W} = \mathbb{W}_1 \cdots \mathbb{W}_{\ell-m+1}$ the word $w = w_1 \ldots w_\ell$ written using the alphabet $\mathcal{A}^m$, so that $\mathbb{W}_j = w_j \ldots w_{j+m-1}$. The results presented below are valid for the number $N(\mathbb{W})$ of overlapping occurrences and the number $\widetilde{N}(\mathbb{W})$ of clumps of $\mathbb{W}$ in $\mathbb{X}_1 \cdots \mathbb{X}_{n-m+1}$. Since an occurrence of w at position i in $X_1 \cdots X_n$ corresponds to an occurrence of $\mathbb{W}$ at position $i - m + 1$ in $\mathbb{X}_1 \cdots \mathbb{X}_{n-m+1}$, we simply have $N(w) = N(\mathbb{W})$. In contrast, clumps of $\mathbb{W}$ in $\mathbb{X}_1 \cdots \mathbb{X}_{n-m+1}$ are different from clumps of w in $X_1 \cdots X_n$ because $\mathbb{W}$ is less periodic than w, leading to $\widetilde{N}(\mathbb{W}) \neq \widetilde{N}(w)$. Let us take a simple example: $w = \mathtt{ata}$ and

$m = 2$. Put $\mathbb{A} = \texttt{at} \in \mathcal{A}^2$ and $\mathbb{B} = \texttt{ta} \in \mathcal{A}^2$; we then have $\mathbb{W} = \mathbb{AB}$. The sequence tatatatat contains a unique clump of at whereas the associated sequence $\mathbb{BABABABA}$ contains 3 clumps of $\mathbb{AB}$. Indeed, $\mathbb{AB}$ has no period and ata has one period. In fact, the periods of $\mathbb{W}$ are those periods of w that are strictly less than $\ell - m + 1$. Therefore, the Poisson approximation for the declumped count in a m-order Markov chain does not follow immediately from the case $m = 1$; a rigorous proof would require applying the Chen-Stein theorem with an adapted neighborhood and to bound the new quantities b_1, b_2 and b_3 in Mm, but this has not yet been carried out.

Since $N(w) = N(\mathbb{W})$, Corollary 6.4.8 ensures that $N(w)$ can be approximated by a sum $\sum_{k\geq 1} kZ_k$, where Z_k is a Poisson variable whose expectation is $(n - \ell + 1)$ times the probability that a k-clump of $\mathbb{W}$ starts at a given position in $\mathbb{X}_1 \cdots \mathbb{X}_{n-m+1}$. From Equation (6.2.10), we obtain

$$\mathbf{E}(Z_k) = (n - \ell + 1)(1 - A'(w))^2 A'(w)^{k-1}\mu(w)$$

with

$$A'(w) = \sum_{p\in\mathcal{P}'(w)\cup\{1,\ldots,\ell-m\}} \frac{\mu(w^{(p)}w)}{\mu(w)}.$$

An important consequence is that, in Mm, the compound Poisson approximation for words that cannot overlap on more than $m - 1$ letters becomes a single Poisson approximation.

6.4.6. Large deviation approximations

For long sequences, the probability that a given word occurs more than a certain number of times can be approximated using a Gaussian or a compound Poisson distribution (Sections 6.4.3 and 6.4.5). The aim of this section is to show that large deviation techniques can also be used to approximate the probability that a given word frequency deviates from its expected value by more than a certain amount. Let $w = w_1 \cdots w_\ell$ be a word of length ℓ; recall that $\mu(w)$ denotes the probability that w occurs at a given position in $X_1 \cdots X_n$. We aim to provide good approximations for $\mathbf{P}(\frac{1}{n-\ell+1}N(w) \geq \mu(w) + b)$ and $\mathbf{P}(\frac{1}{n-\ell+1}N(w) \leq \mu(w) - b)$ with $0 < b < 1$.

We assume that $X_1 \cdots X_n$ is a stationary first-order Markov chain on a finite alphabet $\mathcal{A}$ with transition probabilities $\pi(a, b) > 0$, $a, b \in \mathcal{A}$. (Generalization to Mm follows the same setup as in Section 6.4.5, using (6.1.6).) To use Theorem 6.8.6 for $\frac{1}{n-\ell+1}N(w)$, we need to consider the irreducible Markov chain $\mathbb{X}_1$, $\ldots$, $\mathbb{X}_{n-\ell+1}$ on $\mathcal{A}^\ell$ where $\mathbb{X}_i = X_i \cdots X_{i+\ell-1}$, with transition matrix $\mathbb{\Pi} = (\Pi(u, v))_{u,v\in\mathcal{A}^\ell}$ such that

$$\Pi(u_1 \cdots u_\ell, v_1 \cdots v_\ell) = \begin{cases} \pi(u_\ell, v_\ell) & \text{if } u_{j+1} = v_j,\ j = 1 \cdots \ell - 1, \\ 0 & \text{otherwise.} \end{cases}$$

The count $N(w)$ can then be written as

$$N(w) = \sum_{i=1}^{n-\ell+1} 1\!\!1\{X_i \cdots X_{i+\ell-1} = w_1 \cdots w_\ell\}$$
$$= \sum_{i=1}^{n-\ell+1} 1\!\!1\{\mathbb{X}_i = w\}$$

Let I be the function

$$I(x) = \sup_{\theta \in \mathbb{R}} (\theta x - \log \lambda(\theta)),$$

$x \in \mathbb{R}$, where $\lambda(\theta)$ is the largest eigenvalue of the matrix $\mathbb{\Pi}_\theta = (\Pi_\theta(u,v))_{u,v \in \mathcal{A}^\ell}$ defined by

$$\Pi_\theta(u,v) = \begin{cases} \mathrm{e}^\theta \Pi(u,v) & \text{if } v = w, \\ \Pi(u,v) & \text{otherwise.} \end{cases}$$

Let $0 < b < 1$; applying Theorem 6.8.6 with the function $f(u) = 1\!\!1\{u = w\}$ to the closed subset $[\mu(w)+b, +\infty]$ and the open subset $(\mu(w)+b, +\infty)$, we obtain

$$\lim_{n \to +\infty} \frac{1}{n-\ell+1} \log \mathbf{P}\left(\frac{1}{n-\ell+1} N(w) \geq \mu(w) + b\right) = -I(\mu(w)+b)\,;$$

indeed, the rate function I is convex and minimal at $\mathbf{E}(f(\mathbb{X}_i)) = \mu(w)$. Similarly we have

$$\lim_{n \to +\infty} \frac{1}{n-\ell+1} \log \mathbf{P}\left(\frac{1}{n-\ell+1} N(w) \leq \mu(w) - b\right) = -I(\mu(w)-b).$$

Denoting the observed count of w in the biological sequence by $N^{\text{obs}}(w)$, as a consequence we have for large n:

if $N^{\text{obs}}(w) > (n-\ell+1)\mu(w)$ and $b := \frac{N^{\text{obs}}(w)}{n-\ell+1} - \mu(w)$, then

$$\mathbf{P}(N(w) \geq N^{\text{obs}}(w)) \simeq \exp\left(-(n-\ell+1) I\left(\frac{N^{\text{obs}}(w)}{n-\ell+1}\right)\right),$$

if $N^{\text{obs}}(w) < (n-\ell+1)\mu(w)$ and $b := \mu(w) - \frac{N^{\text{obs}}(w)}{n-\ell+1}$, then

$$\mathbf{P}(N(w) \leq N^{\text{obs}}(w)) \simeq \exp\left(-(n-\ell+1) I\left(\frac{N^{\text{obs}}(w)}{n-\ell+1}\right)\right).$$

Note that this approximation is obtained assuming the transition probabilities $\pi(a,b)$, $a, b \in \mathcal{A}$ are known. Moreover, since $\lambda(\theta)$ is an eigenvalue of a $|\mathcal{A}|^\ell \times |\mathcal{A}|^\ell$ matrix, the word length ℓ is a limiting factor for the numerical calculation, even if $|\mathcal{A}| = 4$.

6.5. Renewal count distribution

As a particular case of non-overlapping occurrence counts, in this section we count renewals of a word $w = w_1 w_2 \dots w_\ell$ in a random sequence $X_1 \cdots X_n$ as defined in Section 6.2. We then consider the renewal count $R_n(w) = \sum_{i=1}^{n-\ell+1} \mathbb{I}_i(w)$, where $\mathbb{I}_i(w)$ is the random indicator that a renewal of w starts at position i in $X_1 \cdots X_n$ (see (6.2.11)).

Exact results for the distribution of R_n have been proposed using a combinatorial approach and language decompositions. Because those tools are very different from the ones used in this chapter, we only present asymptotic results. First we derive the expected renewal count.

Expected renewal count If the random indicators $\mathbb{I}_i(w)$ had the same expectation, say $\mu_R(w)$, then $\mathbf{E}(R_n(w)) = (n-\ell+1)\mu_R(w)$. This is the commonly used expectation, but it ignores the end effect. For $i > \ell$, the $\mathbb{I}_i(w)$'s are effectively identically distributed by stationarity of the Markov process, but it is not the case for $1 \leq i \leq \ell$.

We start with the calculation of $\mu_R(w)$. Recall that $\mathcal{P}(w)$ is the set of periods of w and that $w^{(p)} = w_1 w_2 \cdots w_p$ denotes the word composed of the first p letters of w. When the Markov process is in stationarity, we have from renewal theory that

$$\mu_R(w) = \frac{\mu(w)}{\mathcal{Q}(1)} \tag{6.5.1}$$

with $\mathcal{Q}$ given in (6.2.2). To understand this formula, note that we can decompose the event {there is an occurrence of w starting at position i}, $i > \ell$, as the disjoint union of {there is a renewal of w starting at position i} and {there is a renewal of w starting at position j directly followed by the letters $w_{\ell-i+j+1} \cdots w_\ell$ and $j - i$ is a period of w}, for $j \in \{i - \ell + 1, \dots, i - 1\}$. This can be written as follows

$$\begin{aligned} Y_i(w) &= \sum_{j=i-\ell+1}^{i} \mathbb{I}_j(w) Y_{j+\ell}(w_{\ell-i+j+1} \cdots w_\ell) \mathbb{1}\{i - j \in \mathcal{P}(w) \cup \{0\}\} \\ &= \sum_{p \in \mathcal{P}(w) \cup \{0\}} \mathbb{I}_{i-p}(w) Y_{i+\ell-p}(w_{\ell-p+1} \cdots w_\ell). \end{aligned}$$

Taking expectations on both sides thus gives

$$\mu(w) = \sum_{p \in \mathcal{P}(w) \cup \{0\}} \mu_R(w) \mu(w_{\ell-p} \cdots w_\ell) \frac{1}{\mu(w_{\ell-p})}.$$

Hence

$$\mu_R(w) = \frac{\mu(w)}{1 + \sum_{p \in \mathcal{P}(w)} \pi(w_{\ell-p}, w_{\ell-p+1}) \cdots \pi(w_{\ell-1}, w_\ell)},$$

which gives the result (6.5.1).

As previously noted, the first variables $\mathbb{I}_1(w)$, ..., $\mathbb{I}_\ell(w)$ are not identically distributed because of boundary effects. For the asymptotic results we are interested in in this section, this end effect may be ignored.

6.5.1. Gaussian approximation

Once the asymptotic variance is established, the normal approximation follows from the Markov Renewal Central Limit Theorem. Calculating the asymptotic variance is a little more involved than calculating the mean, relying much on the autocorrelation polynomial. To this purpose, we define $\mathbf{1}$ as the Card($\mathcal{A}$) $\times$ Card($\mathcal{A}$) matrix where all the entries equal 1. With Π denoting the Markovian transition matrix, put

$$Z = \sum_{k=1}^{\infty} (\Pi - \underline{\mu}\mathbf{1})^k. \tag{6.5.2}$$

Put

$$\sigma^2 = \mu_R^2(w) \left((1 - 2\ell) + 2\frac{\mathcal{Q}'(1)}{\mathcal{Q}(1)} + \frac{2Z(w_\ell, w_1)}{\mu(w_1)} \right).$$

We then have the following Central Limit Theorem.

THEOREM 6.5.1. *We have that, as $n \to \infty$,*

$$\frac{R_n(w) - n\mu_R(w)}{\sqrt{n}} \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2).$$

The main technique to prove this theorem being generating functions, no bound on the rate of convergence is obtained. Note also that we do not have a corresponding result when mean and standard deviation are estimated.

6.5.2. Poisson approximation

Similarly as with the declumped count, we can also derive a Poisson approximation for the renewal count under the rare word condition $n\mu(w) = O(1)$. Indeed this is very simple. Recall (6.2.3)

$$Y_i(w) := \mathbb{1}\{w \text{ starts at position } i \text{ in } \underline{X}\}.$$

We can write, for $i > \ell$,

$$\begin{aligned} \mathbb{I}_i(w) &= Y_i(w) \prod_{j=i-\ell+1}^{i-1} (1 - \mathbb{I}_j(w)) \\ &= Y_i(w) \prod_{j=i-\ell+1}^{i-1} (1 - Y_j(w)) \end{aligned}$$

$$+Y_i(w)\left(\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w))-\prod_{j=i-\ell+1}^{i-1}(1-Y_j(w))\right)$$

$$=\widetilde{Y}_i(w)+Y_i(w)\left(\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w))-\prod_{j=i-\ell+1}^{i-1}(1-Y_j(w))\right)\tag{6.5.3}$$

whereas $\mathbb{I}_i(w) = Y_i(w)\prod_{j=1}^{i-1}(1 - Y_j(w))$ if $1 \le i \le \ell$. Note that a renewal occurrence in the first ℓ positions is a clump occurrence observed in the finite sequence, and conversely. Thus we have

$$\begin{aligned} R_n(w) &= \sum_{i=1}^{n-\ell+1} \mathbb{I}_i(w) \\ &= \widetilde{N}(w) + \sum_{i=\ell+1}^{n-\ell+1} Y_i(w)\left(\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w))-\prod_{j=i-\ell+1}^{i-1}(1-Y_j(w))\right). \end{aligned}$$

We have already derived a Poisson approximation for the number of clumps $\widetilde{N}(w)$ (see Section 6.4.5). Let us consider the difference

$$R_n(w)-\widetilde{N}(w) = \sum_{i=\ell+1}^{n-\ell+1} Y_i(w)\left(\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w))-\prod_{j=i-\ell+1}^{i-1}(1-Y_j(w))\right).$$

For a summand to be nonzero, firstly we need that $Y_i(w) = 1$. Note that a renewal always implies an occurrence, so that

$$\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w)) \ge \prod_{j=i-\ell+1}^{i-1}(1-Y_j(w)).$$

The product being always 0 or 1, the two products are different if and only if $\prod_{j=i-\ell+1}^{i-1}(1-\mathbb{I}_j(w)) = 1$ and $\prod_{j=i-\ell+1}^{i-1}(1-Y_j(w)) = 0$. This implies that there is no renewal between the positions $i-\ell+1$ and $i-1$, but that there must be an occurrence not only at position i but also at some position j between $i-\ell+1$ and $i-1$. This occurrence again cannot be a renewal, so that it must be part of a larger clump; repeating this argument we see that the occurrence at i must be part of a clump that started before position $i-\ell+1$. This implies that there had to be an occurrence of w somewhere between $i-2\ell+2$ and $i-\ell$, and this occurrence is in the same clump as the occurrence at i. Thus

$$\begin{aligned} \mathbf{P}(R_n(w) \ne \widetilde{N}(w)) &\le \sum_{i=\ell+1}^{n-\ell+1} \sum_{j=i-2\ell+2}^{i-\ell} \mathbf{E}(Y_i(w)Y_j(w)) \\ &\le (n-2\ell+1)(\ell-1)\mu(w)^2\frac{1}{\mu(w_1)}. \end{aligned}\tag{6.5.4}$$

This quantity will be small under the asymptotic framework $n\mu(w) = O(1)$. Thus we may use the Poisson bound for the number of clumps derived above, and just add an error term of order $\log n/n$.

A different type of bound is also available. Put

$$\Gamma = \Gamma(w) = \sup_{t \geq 1} \frac{\pi^{(t)}(w_\ell, w_1)}{\mu(w_1)}. \tag{6.5.5}$$

Recall ρ given in (6.1.5), and $\mathbf{E}(R_n(w))$ is given in (6.5.1). Using the Chen-Stein method, it is possible to prove the following theorem (see Section 6.9).

THEOREM 6.5.2. *We have that*

$$d_{TV}(\mathcal{L}(R_n(w)), \mathcal{P}o(\mathbf{E}(R_n(w))) \leq \left(1 - e^{-\mathbf{E}(\widetilde{N}(w))}\right) D_1 + \min\left\{1, \sqrt{\frac{2}{e\mathbf{E}(\widetilde{N}(w))}}\right\} D_2 + D_3,$$

where

$$\begin{aligned}
D_1 &= (2\ell - 5)(\widetilde{\mu}(w) + \Gamma\mu(w)) - \Gamma(2\ell - 1)\mu(w), \\
D_2 &= 2\mathbf{E}(R_n(w))\rho^\ell \left(2 + 2\rho^\ell + \rho^{3\ell-2}\right), \\
D_3 &= \left(1 + \min\left\{1, (\mathbf{E}(R_n(w)))^{-\frac{1}{2}}\right\}\right)(\mathbf{E}(R_n(w)) - \mathbf{E}(\widetilde{N}(w))).
\end{aligned}$$

It is also of interest to consider the case that $n \to \infty$, for a sequence of words $w^{(n)}$ of length $\ell^{(n)}$, where $\ell^{(n)}$ may grow with n. Indeed, under the conditions

(i) $\lim_{n\to\infty} \mathbf{E}(R_n(w^{(n)})) = \lambda < \infty$

(ii) $\lim_{n\to\infty} \frac{\ell^{(n)}}{n} = 0$,

the bound in Theorem 6.5.2 is of order $O\left(\frac{\ell^{(n)}}{n}\right)$, which converges to zero for $n \to \infty$. Thus $R_n(w^{(n)})$ converges in distribution to a Poisson variable with mean λ.

6.6. Occurrences and counts of multiple patterns

In biological sequence analysis often the distribution of the joint occurrences of multiple patterns rather than that of single words is of relevance, for example when characterizing protein families via short motifs, or when assessing the statistical significance of the count of degenerated words such as `a`(`c` or `g`)`g`(`a` or `t`), describing the family of words {`acga`, `agga`, `acgt`, `aggt`}.

Since the exact distribution of the counts of multiple words is not easily calculated in practice, we will focus in this section on the asymptotic point of view.

Indeed, asymptotic results, similar to the above approximations, are available for the distribution of joint occurrences and joint counts of multiple patterns and we will present them in this section. As we will see, the main new feature one has to consider are the possible overlaps between different words from the target family.

Consider the family of q words $\{w^1, \dots, w^q\}$, where $w^r = w_1^r w_2^r \cdots w_{\ell_r}^r$. For two words $w^1 = w_1^1 w_2^1 \cdots w_{\ell_1}^1$ and $w^2 = w_1^2 w_2^2 \cdots w_{\ell_2}^2$ on $\mathcal{A}$, we describe the possible overlaps between w^1 and w^2 by defining

$$\mathcal{P}(w^1, w^2) := \{p \in \{1, \dots, \ell_1 - 1\} : w_i^2 = w_{i+p}^1, \forall i = 1, \dots, (\ell_1 - p) \wedge \ell_2\}.$$

Thus $\mathcal{P}(w^1, w^2) \neq \emptyset$ means that an occurrence of w^2 can overlap an occurrence of w^1 from the right, and $\mathcal{P}(w^2, w^1) \neq \emptyset$ means that w^2 can overlap w^1 from the left. Note the lack of symmetry; for example, if $w^1 =$ `aaagaagaa` and $w^2 =$ `aagaatca`, we have $\mathcal{P}(w^1, w^2) = \{4, 7, 8\}$ and $\mathcal{P}(w^2, w^1) = \{7\}$. To avoid trivialities, we make the following assumption.

(A1) $\qquad \forall r \neq r'$, w^r is not a substring of $w^{r'}$.

Thus $\{w^1, \dots, w^q\}$ is a *reduced* set of words. Again we model the sequence $\{X_i\}_{i \in \mathbb{Z}}$ as a stationary ergodic Markov chain.

We introduce the notation

$$\begin{aligned} \ell &= \max_{1 \le r \le q} \ell_r \\ \ell_{\min} &= \min_{1 \le r \le q} \ell_r. \end{aligned} \tag{6.6.1}$$

6.6.1. Gaussian approximation for the joint distribution of multiple word counts

We assume the general model Mm, $m \le \ell_{\min} - 2$. We will show the asymptotic normality of the vector $n^{-1/2}(N(w^r) - \widehat{N}_m(w^r))_{r=1,\dots,q}$:

$$\frac{1}{\sqrt{n}} \big(N(w^r) - \widehat{N}_m(w^r)\big)_{r=1,\dots,q} \xrightarrow{\mathcal{D}} \mathcal{N}(0, \Sigma_m).$$

To prove this result, we use a multivariate martingale central limit theorem. The estimated count $\widehat{N}_m(w^r)$ is given by (6.4.2). The novelty consists here of deriving the asymptotic covariance matrix $\Sigma_m = (\Sigma_m(w^r, w^{r'}))_{r,r'=1,\dots,q}$.

Suppose all the words w^r have the same length ℓ and $m = \ell - 2$ (the maximal model) then the martingale technique (see Section 6.4.3) leads to

$$\begin{aligned} \Sigma_{\ell-2}(w^r, w^{r'}) = \mu(w^r)\mu(w^{r'}) \Bigg(& \frac{\mathbb{I}\{w^r = w^{r'}\}}{\mu(w^r)} - \frac{\mathbb{I}\{(w^r)^- = (w^{r'})^-\}}{\mu((w^r)^-)} \\ & - \frac{\mathbb{I}\{^-(w^r) = ^-(w^{r'})\}}{\mu(^-(w^r))} + \frac{\mathbb{I}\{^-(w^r)^- = ^-(w^{r'})^-\}}{\mu(^-(w^r)^-)} \Bigg). \end{aligned}$$

Note that when $r = r'$, this formula reduces to the asymptotic variance $\sigma^2_{\ell-2}(w^r)$ of Section 6.4.3.

More generally, for $r \neq r'$, the conditional approach (see Section 6.4.3) leads to

$$\begin{aligned}
\Sigma_m(w^r, w^{r'}) = & \sum_{\substack{p \in \mathcal{P}(w^r, w^{r'}) \\ p \le \ell_r - m - 1}} \mu\Big((w^r)^{(p)} w^{r'} \Big) + \sum_{\substack{p \in \mathcal{P}(w^{r'}, w^r) \\ p \le \ell_{r'} - m - 1}} \mu\Big((w^{r'})^{(p)} w^r \Big) \\
& + \mu(w^r)\mu(w^{r'}) \left(\sum_{a_1, \ldots, a_m} \frac{n(a_1 \cdots a_m \bullet) n'(a_1 \cdots a_m \bullet)}{\mu(a_1 \cdots a_m)} \right. \\
& - \sum_{a_1, \ldots, a_{m+1}} \frac{n(a_1 \cdots a_{m+1}) n'(a_1 \cdots a_{m+1})}{\mu(a_1 \cdots a_{m+1})} - \frac{n(w_1^{r'} \cdots w_m^{r'} \bullet)}{\mu(w_1^{r'} \cdots w_m^{r'})} \\
& \left. + \frac{\mathbb{I}\{w_1^r \cdots w_m^r = w_1^{r'} \cdots w_m^{r'}\} - n'(w_1^r \cdots w_m^r \bullet)}{\mu(w_1^r \cdots w_m^r)} \right),
\end{aligned}$$

where $n(\cdot)$ denotes the number of occurrences inside w^r and $n'(\cdot)$ denotes the number of occurrences inside $w^{r'}$. (When $r = r'$, the formula reduces to Equation (6.4.7).)

Note that, if one wants to study the total number of occurrences of a word family $\{w^r, r = 1, \ldots, q\}$, we have

$$\frac{1}{\sqrt{n}} \left(\sum_{r=1}^{q} N(w^r) - \sum_{r=1}^{q} \widehat{N}_m(w^r) \right) \xrightarrow{\mathcal{D}} \mathcal{N}\left(0, \sum_{r, r'} \Sigma_m(w^r, w^{r'}) \right).$$

Error bound for the normal approximation Similarly to Theorem 6.4.4, it is possible to give a bound on the approximation when the parameters do not have to be estimated. Let $\mathbf{w} = \{w^1, \ldots, w^q\}$ be the word set and

$$\mathbf{N}(\mathbf{w}) = (N(w^1), \ldots, N(w^q))$$

be the vector of word counts. Denote its covariance matrix by

$$\mathcal{L}_n = \mathcal{L}_n(\mathbf{w}) = \mathrm{Cov}(\mathbf{N}(\mathbf{w})) = \big(\mathrm{Cov}(N(w^i), N(w^j) \big)_{i,j=1,\ldots,q}.$$

A calculation similar to (6.4.1) shows that, for two different words u and v of length ℓ_u and ℓ_v such that u is not a substring of v and v is not a substring of u:

$$\begin{aligned}
& \mathrm{Cov}(N(u), N(v)) = \\
& \sum_{p \in \mathcal{P}(u,v)} \mathbf{E}(N(u^{(p)} v)) + \sum_{p \in \mathcal{P}(v,u)} \mathbf{E}(N(v^{(p)} u)) - \mathbf{E}(N(u))\mathbf{E}(N(v)) \\
& + \mu_1(u)\mu_1(v) \sum_{d=1}^{n - \ell_u - \ell_v + 1} (n - \ell_u - \ell_v + 2 - d) \left[\frac{\Pi^d(u_{\ell_u}, v_1)}{\mu_1(v_1)} + \frac{\Pi^d(v_{\ell_v}, u_1)}{\mu_1(u_1)} \right].
\end{aligned}$$

In particular, $\mathcal{L}_n$ is invertible.

Some more notation is needed. Let $\mathcal{H}$ denote the collection of convex sets in $\mathbb{R}^q$, and let

$$\beta = \beta(\mathbf{w}) = \max_{1 \le r \le q} \mu(w^r).$$

Recall, from the transition matrix diagonalization, α given in (6.1.1) and Q_t given in (6.1.2). Using Theorem 6.8.1, it is possible to derive the following result.

THEOREM 6.6.1. *Assume the Markov model M1. Let $\mathbf{Z} \sim \mathcal{N}(\mathbf{E}(\mathbf{N}(w), \mathcal{L}_n)$. There are constants c and C_1, C_2, C_3 such that, for any set $\mathbf{w}$ of q words with maximal length ℓ,*

$$\sup_{A \in \mathcal{H}} |\mathbf{P}(\mathbf{N}(\mathbf{w}) \in A) - \mathbf{P}(\mathbf{Z} \in A)| \le c \min_{\ell \le s \le \frac{n}{2}} B_s,$$

where

$$\begin{aligned} B_s = {} & 2q^{\frac{3}{2}}(4s-3)\left|\mathcal{L}_n^{-1}\right|^{\frac{1}{2}} \\ & +2q^{\frac{1}{2}}n(2s-1)(4s-3)\left(q^2\sqrt{\left|\mathcal{L}_n^{-1}\right|}\right)^3\left(|\log(q^2\sqrt{\left|\mathcal{L}_n^{-1}\right|})| + \log n\right) \\ & +C_1 n\left|\mathcal{L}_n^{-1}\right|^{\frac{1}{2}}\beta|\alpha|^{s-\ell+1} \\ & +C_2\left(|\log(q^2\sqrt{\left|\mathcal{L}_n^{-1}\right|})| + \log n\right)(2s-1)|\alpha|^{s-\ell+1} \\ & +C_3\left(|\log(q^2\sqrt{\left|\mathcal{L}_n^{-1}\right|})| + \log n\right)(n-2s+1)nq^4\beta^2\left|\mathcal{L}_n^{-1}\right||\alpha|^{s-\ell+1}. \end{aligned}$$

Here,

$$\begin{aligned} C_1 &= \max\left\{\sum_{a,b \in \mathcal{A}} \mu(a)C_{1,1}(a,b), \sum_{a \in \mathcal{A}} C_{1,2}(a), \sum_{a \in \mathcal{A}} C_{1,3}(a)\right\} \\ C_2 &= n\left|\mathcal{L}_n^{-1}\right| q^4\beta(2\beta+1)C_1 \\ C_3 &= \max_{a,b \in \mathcal{A}}\left\{\sum_{t \ge 2} \frac{|Q_t(a,b)|}{\mu(a)}\right\} \end{aligned}$$

and

$$\begin{aligned} C_{1,1}(a,b) &= \max_{x,y \in \mathcal{A}}\left\{\sum_{t \ge 2 \text{ or } t' \ge 2} \frac{|Q_t(a,x)Q_{t'}(b,y)|}{\mu(x)}\right\} + \sum_{t \ge 2}|Q_t(a,b)| \\ C_{1,2}(a) &= \max_{b \in \mathcal{A}}\left\{\sum_{t \ge 2}|Q_t(b,a)|\right\} \\ C_{1,3}(a) &= \max_{b \in \mathcal{A}}\left\{\sum_{t \ge 2} \frac{|Q_t(a,b)|}{\mu(b)}\right\}. \end{aligned}$$

The constant c is not easy to describe. Note that convergence on the class of convex sets is not as strong as convergence in total variation. Indeed, approximating discrete counts by a continuous multivariate variable might not be expected to be very good in total variation distance.

6.6.2. Poisson and compound Poisson approximations for the joint distribution of declumped counts and multiple word counts

We assume the model M1 since generalization to Mm follows the single pattern case. To give a bound on the error for a Poisson process approximation for overlapping counts, define the following quantities for all r and r' in $\{1, \ldots, q\}$, and for all $a \in \mathcal{A}$:

$$
\begin{aligned}
\Omega_r &= \sum_{s=1}^{3\ell-\ell_r-2} \Pi^s, \\
\Omega_{r,r'} &= \sum_{s=1}^{\ell_r+\ell_{r'}-2} \Pi^s, \\
M(w^r, w^{r'}) &= \begin{cases} \displaystyle\sum_{p \in \mathcal{P}(w^r, w^{r'})} \frac{1}{\mu((w^{r'})^{(\ell_r - p)})} & \text{if } r \neq r' , \\ 0 & \text{if } r = r' , \end{cases} \\
T_1(w^r, w^{r'}) &= (2n - \ell_r - \ell_{r'} + 2)\mu(w^r)\widetilde{\mu}(w^{r'}) \left(\frac{\Omega_{r'}(w^{r'}_{\ell_{r'}}, w^r_1)}{\mu(w^r_1)} + M(w^{r'}, w^r) \right), \\
T_2(w^r, w^{r'}) &= (n - \ell_r + 1)\Big((\ell - 1)(\widetilde{\mu}(w^r)\mu(w^{r'}) + \mu(w^r)\widetilde{\mu}(w^{r'})) \\
&\quad + (6\ell - 5)\widetilde{\mu}(w^r)\widetilde{\mu}(w^{r'}) \Big), \\
T_3(w^r, w^{r'}) &= (n - \ell_r + 1)\mu(w^r)\mu(w^{r'}) \left(\frac{\Omega_{r,r'}(w^r_{\ell_r}, w^{r'}_1)}{\mu(w^{r'}_1)} + \frac{\Omega_{r,r'}(w^{r'}_{\ell_{r'}}, w^r_1)}{\mu(w^r_1)} \right) \\
&\quad + \frac{(n - \ell_r + 1)(6\ell - 3\ell_r - 3\ell_{r'} + 2)}{\mu_{\min}} \widetilde{\mu}(w^r)\widetilde{\mu}(w^{r'}) \qquad (6.6.2) \\
&\quad + \frac{(n - \ell_r + 1)(\ell - 2)}{\mu_{\min}} \left(\mu(w^r)\widetilde{\mu}(w^{r'}) + \mu(w^{r'})\widetilde{\mu}(w^r) \right) \\
&\quad + (n - \ell_r + 1)\mu(w^r)\mu(w^{r'}) \left(M(w^r, w^{r'}) + M(w^{r'}, w^r) \right) , \\
\gamma_1(\ell_r, \ell, a) &= \sum_{x,y \in \mathcal{A}} \mu(x) \max_{b \in \mathcal{A}} \Bigg| \frac{1}{\mu(b)} \sum_{(t,t') \neq (1,1)} \frac{\alpha_t^{2\ell-\ell_r}\alpha_{t'}^{2\ell-\ell_r}}{\alpha^\ell} Q_t(x, b)Q_{t'}(a, y) \\
&\quad - \sum_{t=2}^{|\mathcal{A}|} \frac{\alpha_t^{4\ell-2}}{\alpha^\ell} Q_t(x, y) \Bigg| ,
\end{aligned}
$$

$$\gamma_2(\ell_r,\ell) = \sum_{x,y\in\mathcal{A}} \mu(x) \max_{a,b\in\mathcal{A}} \left(\frac{1}{\mu(b)} \sum_{(t,t')\neq(1,1)} \left| \frac{\alpha_t^{2\ell-\ell_r}\alpha_{t'}^{2\ell-\ell_r}}{\alpha^\ell} Q_t(x,b)Q_{t'}(a,y) \right| \right.$$
$$\left. + \sum_{t=2}^{|\mathcal{A}|} \left| \frac{\alpha_t^{5\ell-3}}{\alpha^\ell} Q_t(x,y) \right| \right).$$

Here we choose as index set $I = \{1,2,\ldots,q(n+1)-\sum_{s=1}^q \ell_s\}$; it can be written as the disjoint union $I = \bigcup_{r=1}^q I_r$ with

$$I_r = \left\{ (r-1)(n+1) - \sum_{s=1}^{r-1} \ell_s + 1, \ldots, r(n+1) - \sum_{s=1}^{r} \ell_s \right\}. \tag{6.6.3}$$

We define $[i]$ by

$$[i] := i - (r-1)(n+1) + \sum_{s=1}^{r-1} \ell_s \quad \text{with } r = r(i) \text{ such that } i \in I_r\,. \tag{6.6.4}$$

Joint distribution of declumped counts To apply Theorem 6.8.2, the Bernoulli process $\underline{\widetilde{Y}} = (\widetilde{\mathbb{Y}}_i)_{i\in I}$ and the Poisson process $\underline{Z} = (Z_i)_{i\in I}$ are given by

$$\begin{aligned} \widetilde{\mathbb{Y}}_i &= \widetilde{Y}_{[i]}(w^r), \\ Z_i &\sim \mathrm{Po}(\widetilde{\mu}(w^r)), \end{aligned} \tag{6.6.5}$$

where r is such that $i \in I_r$. For $i \in I$, we choose the neighborhood $B_i := \{j \in I : |[j]-[i]| \le 3\ell - 3\}$.

Then the following results can be proven. Recall the notation (6.6.2), (6.6.5), and (6.6.1).

Theorem 6.6.2. *Under assumption (A1) we have*

$$\begin{aligned} &d_{TV}\left(\mathcal{L}(\underline{\widetilde{Y}}), \mathcal{L}(\underline{Z})\right) \\ &\le (n-\ell_{min}+1)(6\ell-5)\left(\sum_{r=1}^q \widetilde{\mu}(w^r)\right)^2 + \sum_{1\le r,r'\le q} T_1(w^r, w^{r'}) \\ &\quad + |\alpha|^\ell \sum_{r=1}^q \gamma_1(\ell_r,\ell,w^r_{\ell_r})(n-\ell_r+1)\widetilde{\mu}(w^r)\,. \end{aligned}$$

Corollary 6.6.3. *Let $(Z_r)_{r=1,\ldots,m}$ be independent Poisson variables with $\mathbf{E}(Z_r) = (n-\ell_r+1)\widetilde{\mu}(w^r)$. With the previous notation and under assumption (A1), we have*

$$
\begin{aligned}
& d_{TV}\Big(\mathcal{L}\big((\widetilde{N}(w^r))_{r=1,\ldots,q}\big), \mathcal{L}\big((Z_r)_{r=1,\ldots,q}\big)\Big) \\
& \leq (n-\ell_{min}+1)(6\ell-5)\left(\sum_{r=1}^{q}\widetilde{\mu}(w^r)\right)^2 + \sum_{1\leq r,r'\leq q} T_1(w^r, w^{r'}) \\
& \qquad +|\alpha|^\ell \sum_{r=1}^{q}\gamma_1(\ell_r,\ell,w^r_{\ell_r})(n-\ell_r+1)\widetilde{\mu}(w^r) + \sum_{r=1}^{q}(\ell_r-1)\big(\mu(w^r)-\widetilde{\mu}(w^r)\big)\,.
\end{aligned}
$$

The proof is a direct application of Theorem 6.8.2, similar as in Section 6.4.

Distribution of multiple word counts In a similar way a compound Poisson approximation for the numbers of occurrences can be obtained. Choose as index set

$$
I = \left\{1,2,\ldots,q(n+1)-\sum_{s=1}^{q}\ell_s\right\}\times\{1,2,\ldots\}.
$$

To apply Theorem 6.8.2, the Bernoulli process $\underline{\widetilde{\mathbb{Y}}} = (\widetilde{\mathbb{Y}}_{i,k})_{(i,k)\in I}$ and the Poisson process $\underline{\mathbb{Z}} = (Z_{i,k})_{(i,k)\in I}$ are now defined as

$$
\begin{aligned}
\widetilde{\mathbb{Y}}_{i,k} &= \widetilde{Y}_{[i],k}(w^r),\\
Z_{i,k} &\sim \mathrm{Po}(\widetilde{\mu}_k(w^r)),
\end{aligned}
$$

where $r=r(i)$ is such that $i\in I_r$; I_r and $[i]$ are given by (6.6.3) and (6.6.4). For $(i,k)\in I$, the neighborhood is still $B_{i,k} := \{(j,k')\in I : -(k'+3)(\ell-1)\leq [j]-[i]\leq (k+3)(\ell-1)\}$.
We make the following weak assumption on the overlap structure.

(A2) $\qquad \forall r\neq r'$, w^r is not a substring of any composed word in $\mathcal{C}_2(w^{r'})$.

THEOREM 6.6.4. *Under assumptions (A1), (A2) and with the notation (6.6.2), we have*

$$
\begin{aligned}
d_{TV}\Big(\mathcal{L}(\underline{\widetilde{\mathbb{Y}}}), \mathcal{L}(\underline{\mathbb{Z}})\Big) \leq & \sum_{1\leq r,r'\leq q} T_2(w^r,w^{r'}) + \sum_{1\leq r,r'\leq q} T_3(w^r,w^{r'}) \\
& +|\alpha|^\ell \sum_{r=1}^{q}\gamma_2(\ell_r,\ell)(n-\ell_r+1)\widetilde{\mu}(w^r)\,.
\end{aligned}
$$

The following corollary is easily obtained.

COROLLARY 6.6.5. *Let $(Z_k)_{k\geq1}$ be independent Poisson variables with expectation $\mathbf{E}(Z_k)=\sum_{r=1}^{q}(n-\ell_r+1)\widetilde{\mu}_k(w^r)$; CP denotes the (compound Poisson) distribution of $\sum_{k\geq1} kZ_k$. With the notation (6.6.2) and under assumptions*

(A1), (A2), we have

$$d_{\mathrm{TV}}\left(\mathcal{L}\left(\sum_{r=1}^{q} N(w^r)\right), \mathrm{CP}\right) \leq \sum_{1\leq r,r'\leq q} T_2(w^r, w^{r'}) + \sum_{1\leq r,r'\leq q} T_3(w^r, w^{r'})$$
$$+|\alpha|^\ell \sum_{r=1}^{q} \gamma_2(\ell_r, \ell)(n-\ell_r+1)\widetilde{\mu}(w^r)$$
$$+2\sum_{r=1}^{q}(\ell_r-1)\left(\widetilde{\mu}(w^r)-\mu(w^r)\right).$$

Again, empirically the compound Poisson approximation may perform better than the bound suggests, in the case of not so rare words.

Expected count of mixed clumps For the family $\mathbf{w} = (w^1, \ldots, w^q)$ of words it is also interesting to consider the number of *mixed* clumps of occurrences. Let

$$Y_i^c(\mathbf{w}) = \sum_{r=1}^{q} Y_i(w^r) \prod_{j=i-\ell_r+1}^{i-1} \left\{1 - \sum_{r'=1}^{q} Y_j(w^{r'})\right\},$$

that is, $Y_i^c(\mathbf{w}) = 1$ if there is an occurrence of a word from the family $\mathbf{w}$ at i, and if there is no previous occurrence of any word in $\mathbf{w}$ that overlaps position i. Thus the mixed clumps can be composed of any words from $\mathbf{w}$, whereas for $\widetilde{Y}_i$ the clumps are composed of the same word. Note that, for $q = 1$, $Y_i^c(\mathbf{w}) = \widetilde{Y}_i(w^1)$. Let

$$N^c(\mathbf{w}) = \sum_{i=1}^{n-\ell_{\min}+1} Y_i^c(\mathbf{w}).$$

be the number of mixed clumps in the sequence. To calculate $\mathbf{E}(N^c(\mathbf{w}))$, introduce the quantities

$$e_{r,s} = \sum_{p\in\mathcal{P}(w^r,w^s)} \frac{\mu\left(w^s_{(\ell_s-\ell_r+p+1)}\right)}{\mu(w^r_{\ell_r})},$$

where the summands are the probabilities of observing the last $(\ell_s - \ell_r + p)$ letters of w^s successively given that the last letter of w^r has just occured. It can be shown that

$$\mathbf{E}(N^c) = (n-\ell+1)\sum_{r=1}^{q} y_r, \tag{6.6.6}$$

where $(y_1, \ldots, y_q)$ is the solution of the $q \times q$ linear system of equations

$$\sum_{r=1}^{q} y_r e_{r,s} = \mu(w^s), \quad s = 1, \ldots, q.$$

6.6.3. Competing renewal counts

Related results to the above for renewal counts are available. We consider non-overlapping occurrences in competition with each other. For example, in the sequence `cgtatattaaaaatattaga`, the set of words `tat`, `tta` and `aa` has renewal occurrences of `tat` at position 3 and 14, of `tta` at position 7, and of `aa` at positions 10 and 12. The occurrences of `tat` at position 5, of `tta` at position 16, and of `aa` at positions 9 and 11 are not counted because they overlap with some already counted words.

Let

$$\mathbb{I}_i^c(w^r) = \mathbb{1}\{\text{a competing renewal of } w^r \text{ starts at position } i \text{ in } X_1 \cdots X_n\},$$

and let

$$R_n^c(w^r) = \sum_{i=1}^{n-\ell_r+1} \mathbb{I}_i^c(w^r)$$

be the number of competing renewals of w^r in the sequence $X_1X_2\cdots X_n$.

For the mean $\mu_R^c(w^r) = \mathbf{E}(R_n^c(w^r))$, some more notation is needed. For a matrix A denote its transposed matrix by A^T, and, if A is a square matrix, $Diag(A)$ represents the vector of the diagonal elements of A. Define the probabilities of ending a word for $1 \le j \le \ell_r - 1$ as

$$\begin{aligned}\mathbf{P}_r(j) &= \mathbf{P}(\text{ collect final } j \text{ letters of } w^r | \text{ start with correct } \ell_r - j \text{ initial} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{letters of } w^r) \\ &= \frac{\mu(w^r)}{\mu((w^r)^{(\ell_r-j)})}.\end{aligned}$$

Then, in analogy to (6.2.2), the correlation polynomials are defined as

$$\mathcal{Q}_{r,r'}(z) = 1 + \sum_{p \in \mathcal{P}(w^r, w^{r'})} z^p \mathbf{P}_{r'}(p).$$

Define the $q \times q$ matrix

$$\Delta(z) = (\mathcal{Q}_{r,r'}(z))_{r,r'=1,\dots,q}$$

and

$$\Lambda(z) = (\Delta^{-1})(z)^T$$
$$\Lambda = \Lambda(1).$$

Moreover put $K_r = \mu(w_1^r)\mathbf{P}_r(\ell_r - 1)$ and define the vector

$$K = (K_1, \dots, K_q)^T.$$

Then the means $\mu_R^c(w^r), r = 1, \dots, q$, are given by

$$(\mu_R^c(w^1), \dots, \mu_R^c(w^q))^T = \Lambda K.$$

Gaussian approximation for the joint distribution of competing renewal counts The main problem in the multivariate normal approximation is to specify the covariance structure. To state the result, quite a bit of notation is needed. Define

$$\widetilde{K}_r(z) = z^{\ell_r - 1}\mathbf{P}_r(\ell_r - 1)$$

and the vector

$$\widetilde{K}(z) = (\widetilde{K}_1(z), \ldots, \widetilde{K}_q(z))^T.$$

Denote by

$$Diag(\widetilde{K}(z))$$

the $q \times q$ diagonal matrix with the components of $\widetilde{K}(z)$ as diagonal elements. Put

$$\begin{aligned} \widetilde{K} &= \widetilde{K}(1) \\ H(z) &= \frac{d}{dz}\Lambda(z) \\ H &= H(1). \end{aligned}$$

Define the vector

$$L = (\ell_1 K_1, \ldots, \ell_q K_q),$$

and the matrix

$$\widetilde{Z} = Z_{[\psi]},$$

where Z is defined in (6.5.2), and for a matrix A the matrix $A_{[\psi]}$ is the $q \times q$ matrix whose (r, r') entry is the element of A at the row corresponding to the last letter $w^r_{\ell_r}$ of the word w^r, and at the columns corresponding to the first letter $w^{r'}_1$ of $w^{r'}$. Define the variance-covariance matrix

$$\begin{aligned} C = \frac{1}{2}\big(&\Lambda K(\Lambda K - 2HK - 2\Lambda L)^T + (\Lambda K - 2HK - 2\Lambda L)(\Lambda K)^T\big) \\ &+ Diag(\Lambda K)\widetilde{Z}Diag(\widetilde{K})\Lambda^T + \Lambda Diag(\widetilde{K})\widetilde{Z}^T Diag(\Lambda K) + Diag(\Lambda K). \end{aligned}$$

Now we have all the ingredients to state the normal approximation.

THEOREM 6.6.6. *Under Assumption (A1) we have*

$$\left(\frac{R^c_n(w^r) - n\mu^c_R(w^r)}{\sqrt{n}}\right)_{r=1,\ldots,q} \xrightarrow{\mathcal{D}} \mathcal{N}(0, C).$$

In the case of a single pattern, this theorem reduces to Theorem 6.5.1.

Poisson approximation for the renewal count distribution For a Poisson approximation, the problem can be reduced to declumped counts, as in the case of a single word. For a Poisson process approximation (and, following from that, a Poisson approximation for the counts), we want to assess $\mathbf{P}(\mathbb{I}_i^c(w^r) \neq \widetilde{Y}_i(w^r))$. First consider $\mathbf{P}(\mathbb{I}_i^c(w^r) = 1, \widetilde{Y}_i(w^r) = 0)$. Note that, from (6.5.3), for $i > \ell_r$, to have $\mathbb{I}_i^c(w^r) = 1, \widetilde{Y}_i(w^r) = 0$, there must be an occurrence of w^r at position i, and this occurrence cannot be the start of a clump of w^r, so that there must be an overlapping occurrence of w^r at some position $j = i - \ell_r + 1, \ldots, i-1$. Moreover, this occurrence cannot be a competing renewal, so there must be another word $w^{r'}$ overlapping this occurrence. Hence we may bound

$$\begin{aligned}
&\mathbf{P}(\mathbb{I}_i^c(w^r) = 1, \widetilde{Y}_i(w^r) = 0) \\
&\leq \mu^2(w^r) \sum_{p \in \mathcal{P}(w^r)} \frac{1}{\mu((w^r)^{(\ell_r - p)})} \sum_{r'=1}^{q} \mu(w^{r'}) M(w^{r'}, w^r),
\end{aligned}$$

with M given in (6.6.2). For $i \leq \ell_r$ the above bound is still valid (the probability is even smaller since there is not always enough space for these clumps to occur). Secondly, consider $\mathbf{P}(\mathbb{I}_i^c(w^r) = 0, \widetilde{Y}_i(w^r) = 1)$. For $\mathbb{I}_i^c(w^r) = 0, \widetilde{Y}_i(w^r) = 1$ to occur, there must be an occurrence of w^r at position i, overlapped by an occurrence of a different word $w^{r'}$, so that we may bound

$$\mathbf{P}(\mathbb{I}_i^c(w^r) = 0, \widetilde{Y}_i(w^r) = 1) \leq \mu(w^r) \sum_{r'=1}^{q} \mu(w^{r'}) M(w^{r'}, w^r).$$

Again, for $i \leq \ell_r$ the above bound remains valid. Thus we have

$$\begin{aligned}
\mathbf{P}(\underline{\mathbb{I}^c(w^r)} \neq \underline{\widetilde{Y}(w^r)}) \leq (n - \ell_r + 1)\mu(w^r) &\sum_{r'=1}^{q} \mu(w^{r'}) M(w^{r'}, w^r) \\
&\left(1 + \mu(w^r) \sum_{p \in \mathcal{P}(w^r)} \frac{1}{\mu((w^r)^{(\ell_r - p)})}\right).
\end{aligned}$$

Hence

$$\begin{aligned}
\mathbf{P}(\underline{\mathbb{I}^c} \neq \underline{\widetilde{Y}}) \leq \sum_{r=1}^{q} &(n - \ell_r + 1)\mu(w^r) \sum_{r'=1}^{q} \mu(w^{r'}) M(w^{r'}, w^r) \\
&\left(1 + \mu(w^r) \sum_{p \in \mathcal{P}(w^r)} \frac{1}{\mu((w^r)^{(\ell_r - p)})}\right).
\end{aligned}$$

Thus we obtain as a corollary of Theorem 6.6.2

Corollary 6.6.7. *Under assumption (A1) and with the notation (6.6.2) and (6.6.5), we have*

$$d_{TV}(\mathcal{L}(\underline{\mathbb{I}^c}), \mathcal{L}(\underline{Z})) \le (n - \ell_{min} + 1)(6\ell - 5)\left(\sum_{r=1}^{q} \widetilde{\mu}(w^r)\right)^2 + \sum_{1\le r,r'\le q} T_1(w^r, w^{r'})$$
$$+ |\alpha|^\ell \sum_{r=1}^{q} \gamma_1(\ell_r, \ell, w^r_{\ell_r})(n - \ell_r + 1)\widetilde{\mu}(w^r)$$
$$+ \sum_{r=1}^{q}(n - \ell_r + 1)\mu(w^r) \sum_{r'=1}^{q} \mu(w^{r'})M(w^{r'}, w^r)$$
$$\left(1 + \mu(w^r) \sum_{p\in\mathcal{P}(w^r)} \frac{1}{\mu((w^r)^{(\ell_r-p)})}\right).$$

Note that the order of the approximation is the same as in Theorem 6.6.2; the additional error terms are comparable to T_1 and T_2, respectively. A Poisson approximation for the competing renewal counts follows immediately.

Poisson approximation for competing renewal counts Alternatively to the above approach, a Poisson approximation similar to Theorem 6.5.2 for the number of competing renewals can be derived. Recall $\mathbf{E}(N^c(\mathbf{w}))$ from (6.6.6), and Γ from (6.5.5).

Theorem 6.6.8. *We have that*

$$d_{TV}\left(\mathcal{L}(\sum_{r=1}^{q} R^c_n(w^r)), \mathcal{P}o\left(\sum_{r=1}^{q} \mathbf{E}(R^c_n(w^r))\right)\right)$$
$$\le \left(1 - e^{-\mathbf{E}(N^c(\mathbf{w}))}\right) D_1 + \min\left\{1, \sqrt{\frac{2}{e\mathbf{E}(N^c(\mathbf{w}))}}\right\} D_2 + D_3,$$

where

$$D_1 = (2\ell - 5)\left(\mathbf{E}(Y^c_i(\mathbf{w})) + \Gamma\sum_{r=1}^{q} \mu(w^r)\right) - \Gamma(2\ell_{min} - 1)\sum_{r=1}^{q} \mu(w^r),$$
$$D_2 = 2\mathbf{E}(N^c(\mathbf{w}))\rho^\ell \left(2 + 2\rho^\ell + \rho^{3\ell-2}\right),$$
$$D_3 = \left(1 + \min\left\{1, (\mathbf{E}(N^c(\mathbf{w})))^{-\frac{1}{2}}\right\}\right)\left(\sum_{r=1}^{q} \mathbf{E}(R^c_n(w^r)) - \mathbf{E}(N^c(\mathbf{w}))\right).$$

It is again interesting to consider the case that $n \to \infty$, for a sequence of words $\mathbf{w}^{(n)} = (w^{1,n}, \ldots, w^{q,n})$ of maximal length $\ell^{(n)}$, where $\ell^{(n)}$ may grow with n. It is possible to show that under the conditions

(i) $\lim_{n\to\infty} \sum_{r=1}^{q} \mathbf{E}(R_n^c(w^{r,n})) = \lambda < \infty$

(ii) $\lim_{n\to\infty} \frac{\ell(n)}{n} = 0$,

the bound in Theorem 6.5.2 is of order $O\left(\frac{\ell(n)}{n}\right)$, so that $\sum_{r=1}^{q} R_n^c(w^{r,n})$ converges in distribution to a Poisson variable with mean λ.

6.7. Some applications to DNA sequences

6.7.1. Detecting exceptional words in DNA sequences

We call *exceptional* word a word w that appears in an observed sequence with a significantly high or low frequency. This significance is measured under a given probabilistic model by the p-value $\mathbf{P}(N(w) \geq N_{\text{obs}}(w))$ using the distribution of the count $N(w)$. Depending on the sequence length and on the expected count of the word it is often not realistic to use the exact distribution of the count since it is time consuming to calculate In this section, we will first give some elements of comparison between the p-values obtained using the exact distribution (Section 6.4.1) and the ones obtained using the Gaussian approximation (Section 6.4.3) or the compound Poisson approximation (Section 6.4.5) or using the large deviation techniques (Section 6.4.6). For convenience, we will manipulate scores from $\mathbb{R}$ of the form $\phi^{-1}(\text{p}-value)$ rather than the p-values, where ϕ is the cumulative distribution function of the standard Gaussian distribution (probit normalization). Exceptionally frequent words would then have high positive scores whereas exceptionally rare words would have high negative scores.

Quality of the approximate p-values For each word of length 3, 6 and 9 of the complete genome of the *Lambda* phage ($\ell = 48\,502$), we can compare the exact scores under the Bernoulli model M0 with the approximate ones using either the Gaussian approximation or the compound Poisson distribution (the parameters are assumed to be known). The results are presented on Figure 6.1 together with the approximate scores obtained with the large deviation approach: the x-axis of each plot is for the exact scores of 3-words (first row), 6-words (second row) and 9-words (last row). The y-axis is for the scores approximated with the Gaussian approximation (first column), the compound Poisson distribution (second column) and the large deviation approach (last column). Due to numerical errors the exact score of 5 words of length 3 have not been calculated successfully. We observe that the accuracy of the Gaussian approximation decreases as the length of the words increases (rare words). The compound Poisson approximation is surprisingly satisfactory even for short (frequent) words. This agrees with the evolution of the total variation distance between the exact distribution of the count and both approximate distributions; when the expected count of the word is close to 100 or greater then the accuracy of the Gaussian approximation is very good. The large deviation approach seems also to provide a good approximation for the exceptional words. However, it cannot manage with words having an estimated expected count too close to the observed one.

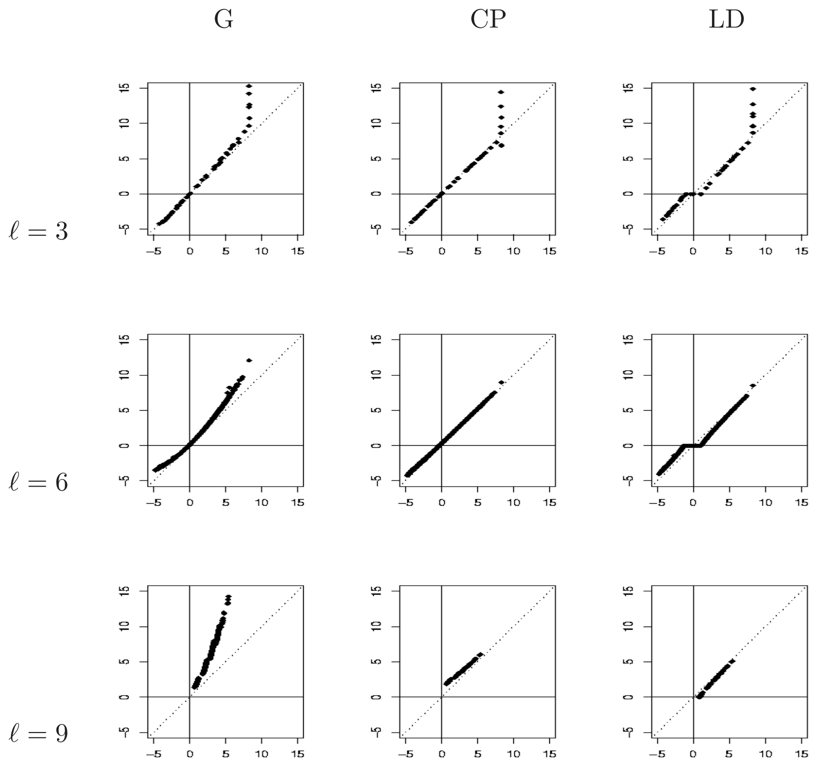


Figure 6.1. Normalized p-values of the counts of all the words of length 3, 6 and 9 in the genome of the *Lambda* phage ($\ell = 48\,502$). Comparison of the Gaussian, the compound Poisson and the large deviation approximations (y-axis) with the exact scores (x-axis).

The p-value is then set to 1/2 in this case and the flatness of the curves is an artifact. An important feature is that every method to calculate or to approximate the p-values seems to classify the words in the same way; the score ranks are almost the same. Moreover, in this example, the three methods agree on the fact that there are no exceptionally rare words of length 9.

Influence of the model Whatever the word count distribution used to calculate the normalized p-value, the choice of the model, in particular the order m of the Markov chain, is important to interpret the exceptionality of a given word. Using the model Mm means taking into account the 1- to $(m+1)$-letter word composition of the sequence. Therefore, the greater the order m of the model, the closer the random sequences will be to the observed sequence, and fewer unexpected words will be found. As an example, Figure 6.2 shows the discrepancy of the scores for the 8-letter words in the complete genome of *E. coli* ($\ell = 4\,638\,858$) under models M0 to M6. For each model, the box contains half of the 65536 scores, the horizontal line is drawn through the box at the median of the data, the upper and lower ends of the box are at upper and lower quartiles (25% and 75%) and vertical lines go up and down from the box to the extremes of the data, except for the outliers, which are plotted by themselves. Here the outliers are the scores that are separated from the box by at least 3 times the inter-quartile range (height of the box). In models M7 and higher, all the 8-letter words have a null score since their counts are included in these models: they are expected as they occur. M6 is then the maximal model for words of length 8.

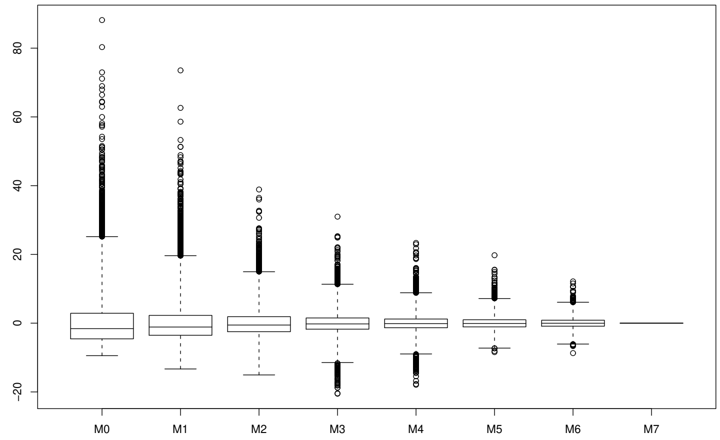


Figure 6.2. Boxplots of the 8-letter word scores in the complete genome of *E. coli* under models M0 to M7.

To analyze the frequency of a ℓ-letter word, the maximal model is of order $m = \ell - 2$; in this model the exceptionality of a word of length ℓ cannot be explained by an unexpected sub-word, since all the sub-word frequencies are included into the maximal model. On the contrary, in small models such contamination by exceptional sub-words may occur. As an illustration let us consider the following example: Figure 6.3 compares the scores (using the Gaussian approximation) of all the 256 4-letter words in the complete *Lamdba* genome under the models M1 (x-axis) and M2 (y-axis). The most over-represented 4-word under M1 is `ccgg`, and it remains significantly over-represented under M2 while taking into account the counts of `ccg` and `cgg`. However, many words lose their exceptionality when the order of the model increases. For example, `gctg` loses its exceptionality as soon as one takes into account the fact that `ctg` occurs 1169 times and is thus a significantly frequent 3-letter word (see

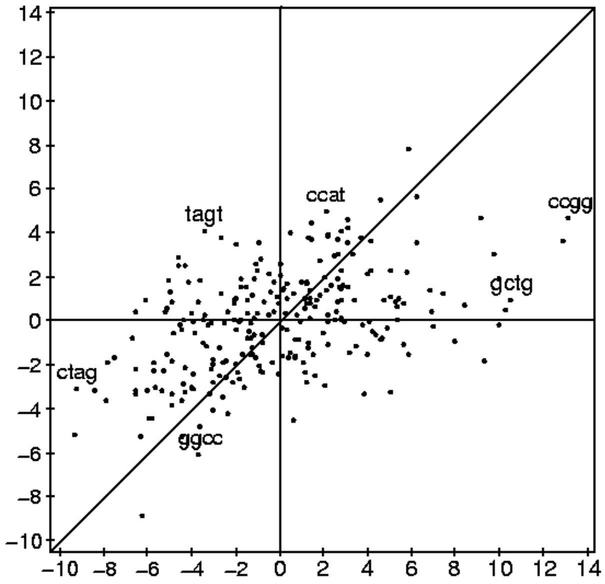


Figure 6.3. Scores of the 4-letter words in the *Lambda* genome under M1 (x-axis) and M2 (y-axis).

Table 6.2). The model M2 says that the 406 occurrences of `gctg` are expected according to the 3-letter word composition of the sequence: `gctg` is expected 394 times under M2 (see Table 6.1). Its exceptionality under M1 (expected only 255 times) is an artifact due to the important over-representation of its sub-word `ctg`. The number of times that we see `gctg` is not surprising given the number of occurrences of `ctg`. This is what we call a contamination. Another such example is `ctag`: it is exceptionally rare under M1 but not under M2. On the other hand, some exceptionality may be hidden in small models and be revealed in higher models, leading to very interesting interpretations. As an example, `ccat` is not exceptional under M1 and becomes one of the most over-represented word under M2. If we look at its two sub-words of length 3, `cca` and `cat` are slightly under-represented (see Table 6.2). Given their low frequency, `ccat` is expected only 191 times under M2, which is is significantly less than the 218 observed occurrences. So, `cca` and `cat` are slightly avoided in the sequence but they are preferentially overlapping in the sequence. This is more pronounced for `tagt` which is composed of the most avoided 3-word `tag` and is declared under-represented under M1 (contamination in fact), but it seems that there is an important constraint for these occurrences of `tag` to be followed by a `t`.

w	$N(w)$	Model M1				Model M2			
		$\widehat{N}_1(w)$	$\sigma_1(w)$	score	rank	$\widehat{N}_2(w)$	$\sigma_2(w)$	score	rank
ctag	14	101.8	9.5	-9.21	2	28.7	4.7	-3.10	27
tagt	71	104.0	9.6	-3.42	57	47.3	5.8	4.07	246
ccat	218	191.1	12.6	2.12	180	168.6	10.0	4.94	253
gctg	406	255.2	14.3	10.52	254	394.6	11.9	0.96	170
ccgg	328	169.7	12.0	13.16	256	273.5	11.6	4.68	252

Table 6.1. Statistics of some 4-letter words in the *Lambda* genome under the models M1 and M2. The rank of the scores are obtained while sorting the 256 scores by increasing order.

w	$N(w)$	$\widehat{N}_1(w)$	$\sigma_1(w)$	score	rank
tag	217	481.2	17.6	-15.04	1
cat	803	869.4	21.6	-3.07	18
cca	675	706.5	19.9	-1.58	25
agt	595	590.2	19.1	0.25	34
gct	856	806.6	20.7	2.39	46
cgg	963	772.1	21.0	9.10	60
ccg	884	684.3	19.7	10.15	61
ctg	1169	802.4	20.8	17.63	63

Table 6.2. Statistics of some 3-letter words in the *Lambda* genome under model M1. The rank of the scores are obtained while sorting the 64 scores by increasing order.

Utility of models Mm_3 Coding DNA sequences are composed of successive trinucleotides called codons. Each base in the sequence is associated to a phase k in $\{1, 2, 3\}$ depending on its position in the associated codon. In the general model Mm_3, the transition probabilities of a letter depend on its phase and word occurrences can be analyzed separately for each phase or for all phases together (see p. 277); note that $N(w) = \sum_k N(w, k)$. Recall that the phase of an occurrence is by convention in this chapter the phase of its last letter. It is well-known to biologists that there exists a bias in the codon usage: codons that code for the same amino acid are not used uniformly. The following analysis illustrates the importance of taking the 3-letter word composition on each phase into account, in particular the codon composition (3-words on phase 3). Let us consider 36 genes of *E. coli* ($\ell = 44\,856$) and analyze the trinucleotide frequency. Figure 6.4 shows that the majority of the trinucleotides have the same behavior under M1 or M1_3; however, some trinucleotides are less exceptional when one takes the phase into account. If we now calculate the scores of the trinucleotides on phase 1, on phase 2 and on phase 3 under M1_3, we see that the main exceptional trinucleotides are the ones on phase 3: the codons. Figure 6.5 presents the discrepancy of theses scores: codons are much more exceptional than the trinucleotides on phase 1 and 2.

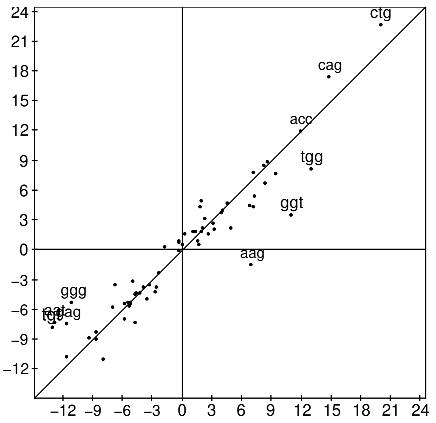


Figure 6.4. Scores of the 3-letter words in 36 genes of *E. coli* under the models M1 (x-axis) and M1_3 (y-axis).

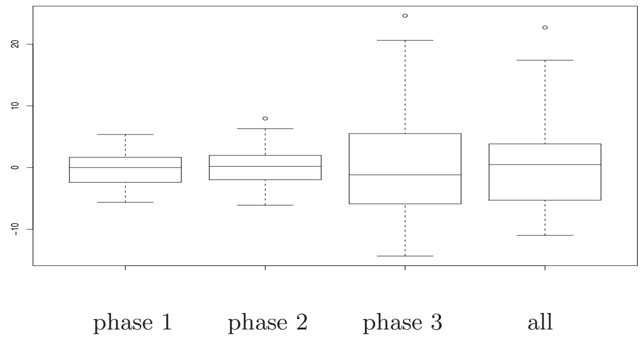


Figure 6.5. Boxplots of the scores of the 3-letter words for each phase and for all phases in 36 genes of *E. coli* under the model M1_3.

Figure 6.6 compares the scores of the 4-words on phase 1 under M1_3 (the codon composition is not taken into account) and M2_3 (the codon composition is taken into account). Note that a 4-word on phase 1 starts with a codon. The 3 most over-represented codons are `ctg`, `cag` and `tgg`. This over-representation is responsible of the exceptionality of `ctgg`, `tggt`, `tggc` and `cagc`. The over-representation of `cagg` seems to be a strong constraint since it is still exceptional given the high frequency of `cag`. When analyzing coding sequences, to be sure to find exceptional words that are not contaminated by the codon usage, the minimal model to use is the model M2_3.

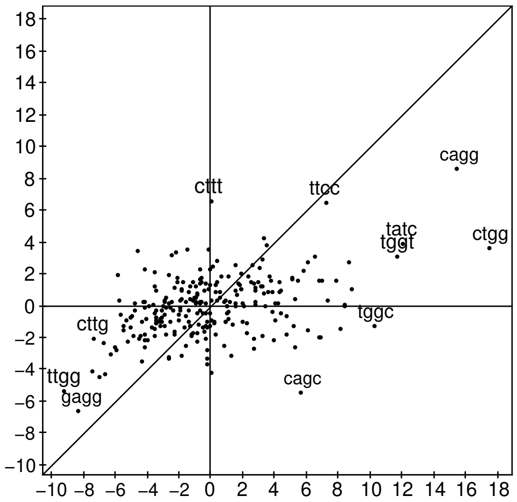


Figure 6.6. Scores of the 4-letter words on phase 1 in 36 genes of *E. coli* under the models M1_3 (x-axis) and M2_3 (y-axis).

6.7.2. Sequencing by hybridization

As a slightly more involved example of how statistics and probability on words are applied in DNA sequence analysis, we describe a problem related to sequencing by hybridization. Sequencing by hybridization is an approach to determine a DNA sequence from the unordered list of all ℓ-tuples contained in this sequence; typical numbers for ℓ are $\ell = 8, 10, 12$. It is based on the fact that DNA nucleotides bind or hybridize with each other: `a` and `t` hybridize, and `c` and `g` hybridize. DNA strands have a polarity (5', 3'), and hybridizing sequences must be of opposite polarity. To avoid introducing notation to show polarity, we present complementary strands written in reverse direction. For example, the sequence `tgtgtgagtg` hybridizes with `acacactcac`. In a sequencing chip, all 4^ℓ possible oligonucleotides ("probes") of length ℓ are attached to the surface of a substrate, each fragment at a distinct location.

To use an SBH chip, the single-stranded target DNA is amplified, labeled by a fluorescent, and exposed to the sequencing chip. The probes on the chip will hybridize to a copy of the single-stranded target DNA if the substring complementary to the probe exists in the target. These probes are then detected with a spectroscopic detector. For example, if $\ell = 4$, the sequence `tgtgtgagtg` will hybridize to the probes `acac`, `actc`, `caca`, `cact`, `ctca` and `tcac`.

As chips can be washed and used again, and due to automatization, this method is not only fast but also inexpensive. There are still technical difficulties in producing an error-free chip; moreover the SBH image may be difficult to read. We remark that the microarray industry grew out of attempts to make SBH technology practical. However, even if these sources of errors are eliminated, a major drawback of the SBH procedure is that more than one sequence may produce the same SBH data. For example, if $\ell = 4$, the sequence `acactcacac` will hybridize to the same probes as the sequence `acacactcac`.

To control this error resulting from non-unique recoverability, we are interested in an estimate for the probability that a sequence is uniquely recoverable. This probability will depend on the probe length ℓ, on the length n of the target sequence, and on the frequencies of the different nucleotides, `a`, `c`, `g` and `t`, in the sequence. Furthermore we need to bound the error made in estimating the probability of unique recoverability in order to make assertions about the reliability of the chip.

As a simplification, we assume that we not only know the set of all ℓ-tuples in the sequence but also their multiplicity (but not the order in which they occur). This multiset is called the ℓ-spectrum of the sequence. In the sequel, unique recoverability is understood to mean unique recoverability of a sequence from its ℓ-spectrum.

Unique recoverability from the ℓ-spectrum can be characterized using the de Bruijn graph whose vertices are the $(\ell-1)$-tuples in the sequence. Two vertices v and w are joined by a directed edge from v to w if the ℓ-spectrum contains an ℓ-tuple for which the first $(\ell - 1)$ nucleotides coincide with v and the last $(\ell - 1)$ nucleotides coincide with w. A sequence is uniquely recoverable from its ℓ-spectrum if and only if there is a unique (Eulerian) path connecting all the vertices. It was shown that there are exactly three structures that prevent unique recoverability:

1. Rotation. The sequence starts and ends with the same $(\ell - 1)$-tuple. In this case, the de Bruijn graph is a cycle, and any vertex could be chosen as the starting point.

2. Transposition with a three-way repeat. If an $(\ell-1)$-tuple occurs three times in the sequence, then the de Bruijn graph has two loops at this vertex, and the order in which these loops are passed is not fixed.

3. Transposition with two interleaved pairs of repeats. There are two "interleaved" pairs of $(\ell - 1)$-tuple repeats, i.e. in the de Bruijn graph there are two vertices x and y connected by a path of the form $\ldots c \ldots y \ldots x \ldots y \ldots$, where we described a path connecting all the vertices by listing the vertices in the order they are used in the path. This implies that there are two ways of going from x to y in the graph.

Example 6.7.1. The sequence `acacactcac` possesses as 4-spectrum the multiset {`acac`, `acac`, `caca`, `cact`, `actc`, `ctca`, `tcac`}. The competing sequence `acactcacac` has the same 4-spectrum. The de Bruijn graph for the sequence acacactcac has as vertices aca, cac, act, ctc and tca. There are two directed edges from aca to cac, and one directed edge each from cac to aca, from cac to act, from act to ctc, from ctc to tca, and from tca to cac. The competing sequence acactcacac has the same de Bruijn graph. For the sequence acacactcac, a path connecting all vertices is

$$\texttt{aca}, \texttt{cac}, \texttt{aca}, (\texttt{cac}, \texttt{act}, \texttt{ctc}, \texttt{tca}), \texttt{cac}.$$

The alternate path

$$\texttt{aca}, (\texttt{cac}, \texttt{act}, \texttt{ctc}, \texttt{tca}), \texttt{cac}, \texttt{aca}, \texttt{cac}$$

also connecting all the vertices, corresponds to the sequence, acactcacac, with the same 4-spectrum.

Thus unique recoverability can be described in terms of possibly overlapping repeats of $(\ell-1)$-tuples within a single sequence. We use the model M0. For a sequence to be uniquely recoverable, the event of an $(\ell-1)$-tuple repeat should be rare. This implies that we consider the occurrence of $(\ell-1)$-tuples under a Poisson regime. (Note that we are interested in the configuration in which the repeats occur; hence we need a Poisson process approximation for the process of repeats rather than a Poisson approximation for the number of repeats.) If repeats are rare, then three-way repeats are negligible, and so is the probability that a sequence starts and ends with the same $(\ell-1)$-tuple. After bounding these probabilities, we thus restrict our attention to interleaved pairs of repeats. Under the Poisson regime, if there are k pairs of repeats, then the occurrences of these repeats are discrete uniform. Additional randomization makes the position of the repeats continuously uniform, so that all orderings of these pairs will be approximately equally likely. This allows the application of a combinatorial argument using Catalan numbers to obtain that the number of interleaved pairs of repeats, if k repeats are present, is approximately $2^k/(k+1)!$. If λ is the expected number of repeats of ℓ-tuples in a single sequence, we hence get, for the probability P_ℓ that $X_1 X_2 \dots X_n$ is uniquely recoverable from its ℓ-spectrum,

$$P_\ell \approx e^{-\lambda} \sum_{k \geq 0} \frac{(2\lambda)^k}{k!(k+1)!}.$$

The Chen-Stein method for Poisson approximation provides explicit bounds for the error terms in this approximation, as follows.

In the sequence $X_1 \dots X_n$ of independent identically distributed letters, let $p = \sum_{a \in \mathcal{A}} \mu^2(a)$ be the probability that two random letters match. We write t for $\ell - 1$, as we are interested in $(\ell-1)$-repeats. Again we have to declump: Define $Y_{i,i} = 0$ for all i, and

$$Y_{i,j} = \begin{cases} \mathbb{1}\{X_1 \cdots X_t = X_{j+1} \cdots X_{j+t}\} & \text{if } i = 0 \\ (1 - \mathbb{1}\{X_i = X_j\})\mathbb{1}\{X_{i+1} \cdots X_{i+t} = X_{j+1} \cdots X_{j+t}\} & \text{otherwise.} \end{cases}$$

Thus $Y_{i,j} = 1$ if and only if there is a leftmost repeat starting after i and j. Put $I = \{(i,j), 1 \leq i, j \leq n - \ell + 1\}$. A careful analysis yields that the process

$\underline{Y} = (Y_\alpha)_{\alpha\in I}$ is sufficient to decide whether a sequence is uniquely recoverable from its ℓ-spectrum (although $\underline{Y}$ contains strictly less information than the process of indicators of occurrences).

For a Poisson process approximation, we first identify the expected number λ of leftmost repeats. If $\alpha = (i, j)$ does not have self-overlap, that is, if $j - i > t$, then

$$\mathbf{E}(Y_\alpha) = \begin{cases} p^t & \text{if } i = 0 \\ (1-p)p^t & \text{otherwise.} \end{cases}$$

Hence the expected number λ^* of repeats without self-overlap is

$$\lambda^* = \binom{n-2t}{2}(1-p)p^t + (n-2t)p^t.$$

If α does have self-overlap, then, in order to have a leftmost repeat at α, for indices in the overlapping set, two matches are required, and for indices in the non-overlapping set, one match is required. Let $d = j - i$; then $E(Y_\alpha)$ depends on the decomposition of $t + d$ into a quotient q of d and a remainder r (such that $t + d = qd + r$): if p_q is the probability that q random letters match, then

$$\mathbf{E}(Y_\alpha) = \begin{cases} p_{q+1}^r p_q^{d-r} & \text{if } i = 0 \\ (p_q - p_{q+1})^r p_q^{d-r} & \text{otherwise.} \end{cases}$$

If λ^* is bounded away from 0 and infinity, which corresponds to having $t = 2log_{1/p(}n) + c$ for some constant c, then it can be seen that

$$\lambda \approx \frac{n^2}{2}(1-p)p^t.$$

Under the regime that λ is bounded away from 0 and infinity, here is a general result. Let $\mu_{\max} = \max_a \mu(a)$ be the probability of the most likely letter.

THEOREM 6.7.2. *Let $\underline{Z} \equiv (Z_\alpha)_{\alpha\in I}$ be a process with independent Poisson distributed coordinates Y_α, with $\mathbf{E}(Z_\alpha) = \mathbf{E}(Y_\alpha), \alpha \in I$. Then*

$$d_{TV}(\underline{Y}, \underline{Z}) \le b(n, t),$$

where the error term $b(n, t)$ is such that

$$b(n,t) \sim \begin{cases} 16\lambda^2 \frac{t}{n} & \textit{in the uniform case} \\ n\mu_{max}^t & \textit{in the nonuniform case.} \end{cases}$$

6.8. Some probabilistic and statistical tools

6.8.1. Stein's method for normal approximation

Stein's method for the normal approximation makes it possible to obtain multivariate normal approximations with a bound on the error in the distance of suprema over convex sets, as follows.

Let $\mathcal{H}$ denote the class of convex sets in $\mathbb{R}^d$. Let $\mathbf{Y}_j, j = 1, \ldots, n$ be random vectors taking values in $\mathbb{R}^d$, and let $\mathbf{W} = \sum_{j=1}^n \mathbf{Y}_j$ be the vector of sums. Assume there is a constant B such that $|\mathbf{Y}_j| := \sum_{i=1}^d |Y_{(j,i)}| \leq B$. Let $\mathbf{Z} \sim \mathcal{N}(0, \mathbf{I}_d)$ have the $d-$dimensional standard multivariate normal distribution.

THEOREM 6.8.1. *Let $\mathcal{S}_i$ and $\mathcal{N}_i$ be subsets of $\{1, \ldots, n\}$, such that $i \in \mathcal{S}_i \subset \mathcal{N}_i, i = 1, \ldots, n$. Assume that there exist constants $D_1 \leq D_2$ such that*

$$\max\{\mathrm{Card}(\mathcal{S}_i), i = 1, \ldots, n\} \leq D_1$$

and

$$\max\{\mathrm{Card}(\mathcal{N}_i), i = 1, \ldots, n\} \leq D_2.$$

Then, for $d = 1$ there exists a universal constant c such that

$$\begin{aligned}&\sup_{x \in \mathbb{R}} |\mathbf{P}(\mathbf{W} \leq x) - \mathbf{P}(\mathbf{Z} \leq x)| \\ &\leq c\{2D_2B + n(2 + \sqrt{\mathbf{E}(W^2)})D_1D_2B^3 + \chi_1 + \chi_2 + \chi_3\}.\end{aligned}$$

For $d \geq 1$ there exists a constant c depending only on the dimension d such that

$$\begin{aligned}\sup_{A \in \mathcal{H}} |\mathbf{P}(\mathbf{W} \in A) - \mathbf{P}(\mathbf{Z} \in A)| \leq c\{&2\sqrt{d}D_2B + 2\sqrt{d}nD_1D_2B^3(|\log B| + \log n) \\ &+\chi_1 + (|\log B| + \log n)(\chi_2 + \chi_3)\},\end{aligned}$$

where

$$\chi_1 = \sum_{j=1}^n \mathbf{E}\left|\mathbf{E}(\mathbf{Y}_j | \sum_{k \notin \mathcal{S}_j} \mathbf{Y}_k)\right|$$

$$\chi_2 = \sum_{j=1}^n \mathbf{E}\left|\mathbf{E}(\mathbf{Y}_j (\sum_{k \in \mathcal{S}_j} \mathbf{Y}_k)^T) - \mathbf{E}(\mathbf{Y}_j (\sum_{k \in \mathcal{S}_j} \mathbf{Y}_k)^T | \sum_{l \notin \mathcal{N}_j} \mathbf{Y}_l)\right|$$

$$\chi_3 = \left|I - \sum_{j=1}^n \mathbf{E}(\mathbf{Y}_j (\sum_{k \in \mathcal{S}_j} \mathbf{Y}_k)^T)\right|.$$

Note that there are no explicit assumptions on the mean vector and the variance-covariance matrix; however, for a good approximation it would be desirable to have the mean vector close to zero, and the variance-covariance matrix close to the identity.

6.8.2. The Chen-Stein method for Poisson approximation

The Chen-Stein method is a powerful tool for deriving Poisson approximations and compound Poisson approximations in terms of bounds on the total variation distance. For any two random processes $\underline{Y}$ and $\underline{Z}$ with values in the same space

E, the total variation distance between their probability distributions is defined by

$$\begin{aligned} d_{\mathrm{TV}}(\mathcal{L}(\underline{Y}), \mathcal{L}(\underline{Z})) &= \sup_{B \subset E, \text{measurable}} |\mathbf{P}(\underline{Y} \in B) - \mathbf{P}(\underline{Z} \in B)| \\ &= \sup_{h: E \to [0,1], \text{measurable}} |\mathbf{E}(h(\underline{Y})) - \mathbf{E}(h(\underline{Z}))| . \end{aligned}$$

The following general bound on the distance to a Poisson distribution is available.

Theorem 6.8.2. *Let I be an index set. For each $\alpha \in I$, let Y_α be a Bernoulli random variable with $p_\alpha = \mathbf{P}(Y_\alpha = 1) > 0$. Suppose that, for each $\alpha \in I$, we have chosen $B_\alpha \subset I$ with $\alpha \in B_\alpha$. Let Z_α, $\alpha \in I$, be independent Poisson variables with mean p_α. The total variation distance between the dependent Bernoulli process $\underline{Y} = (Y_\alpha, \alpha \in I)$ and the Poisson process $\underline{Z} = (Z_\alpha, \alpha \in I)$ satisfies*

$$d_{TV}\left(\mathcal{L}(\underline{Y}), \mathcal{L}(\underline{Z})\right) \leq b_1 + b_2 + b_3,$$

where

$$b_1 = \sum_{\alpha \in I} \sum_{\beta \in B_\alpha} p_\alpha p_\beta \tag{6.8.1}$$

$$b_2 = \sum_{\alpha \in I} \sum_{\beta \in B_\alpha, \beta \neq \alpha} \mathbf{E}(Y_\alpha Y_\beta) \tag{6.8.2}$$

$$b_3 = \sum_{\alpha \in I} \mathbf{E} \left| \mathbf{E} \left\{ Y_\alpha - p_\alpha | \sigma(Y_\beta, \beta \notin B_\alpha) \right\} \right| . \tag{6.8.3}$$

Moreover, if $W = \sum_{\alpha \in I} Y_\alpha$ and $\lambda = \sum_{\alpha \in I} p_\alpha < \infty$, then

$$d_{TV}(\mathcal{L}(W), \mathrm{Po}(\lambda)) \leq \frac{1 - e^{-\lambda}}{\lambda}(b_1 + b_2) + \min\left(1, \sqrt{\frac{2}{\lambda e}}\right) b_3.$$

Note that $b_3 = 0$ if Y_α is independent of $\sigma(Y_\beta, \beta \notin B_\alpha\}$. We think of B_α as a neighborhood of strong dependence of Y_α.

One consequence of this theorem is that for any indicator of an event, i.e. for any measurable functional h from E to $[0, 1]$, there is an error bound of the form $|\mathbf{E}(h(\underline{Y})) - \mathbf{E}(h(\underline{Z}))| \leq d_{\mathrm{TV}}(\mathcal{L}(\underline{Y}), \mathcal{L}(\underline{Z}))$. Thus, if $T(\underline{Y})$ is a test statistic then, for all $t \in \mathbb{R}$,

$$\left| \mathbf{P}\left(T(\underline{Y}) \geq t\right) - \mathbf{P}\left(T(\underline{Z}) \geq t\right) \right| \leq b_1 + b_2 + b_3 ,$$

which can be used to construct confidence intervals and to find p-values for tests based on this statistic.

Note that this method can also be used to prove compound Poisson approximations. For multivariate compound Poisson approximations it is very convenient. For univariate compound Poisson approximations, better bounds are at hand, as will be illustrated in the next subsection.

6.8.3. Stein's method for direct compound Poisson approximation

A drawback of the point process approach to compound Poisson approximation is that the bounds are not very accurate. Instead it is possible to set up a related method for obtaining a compound Poisson approximation directly, in the univariate case.

Denote by $CP(\lambda, \underline{\nu})$ the *compound Poisson distribution* with parameters λ and $\underline{\nu}$, that is, the distribution of the random variable $\sum_{k\geq 1} kM_k$, where $(M_k)_{k\geq 1}$ are independent, and $M_k \sim \mathcal{P}o(\lambda\nu_k), k = 1, 2, \ldots$.

The particular case where $\lambda = n\phi(1-p)$ and $\nu_k = p^{k-1}(1-p)$ for some $\phi > 0$ and $0 < p < 1$, is called the *Polya-Aeppli* distribution.

Again, let I be an index set, and let

$$W = \sum_{\alpha \in I} V_\alpha,$$

where $(V_\alpha)_{\alpha\in I}$ are nonnegative integer-valued random variables. Similarly to the Poisson case, for each $\alpha \in I$ decompose the index set into disjoint sets as

$$I = \alpha \cup \mathcal{S}_\alpha \cup \mathcal{W}_\alpha \cup \mathcal{U}_\alpha.$$

Here, $\mathcal{S}_\alpha$ would correspond to the set of indices with strong influence on α, $\mathcal{W}_\alpha$ would correspond to the set of indices with weak influence on α, and $\mathcal{U}_\alpha$ collects the remaining indices. Put

$$\begin{aligned} S_\alpha &= \sum_{\beta\in\mathcal{S}_\alpha} V_\alpha \\ W_\alpha &= \sum_{\beta\in\mathcal{W}_\alpha} V_\alpha \\ U_\alpha &= \sum_{\beta\in\mathcal{U}_\alpha} V_\alpha. \end{aligned}$$

Then, for $\alpha \in I$,

$$W = V_\alpha + S_\alpha + W_\alpha + U_\alpha.$$

Define the *canonical* parameters $(\lambda, \underline{\nu})$ of the corresponding compound Poisson distribution by

$$\begin{aligned} \lambda\nu_k &= \frac{1}{k}\sum_{\alpha\in I} \mathbf{E}\{V_\alpha \mathbb{I}(V_\alpha + S_\alpha = k)\}, \quad , k \geq 1 \\ \lambda &= \sum_{k\geq 1} k\nu_k. \end{aligned} \tag{6.8.4}$$

Put

$$q_{jk}^{(\alpha)} = \frac{\mathbf{P}(V_\alpha = j, S_\alpha = k)}{m_{i,1}}, \quad \geq 1, k \geq 0,$$

and

$$m_{1,k} = \mathbf{E}(V_\alpha)$$
$$m_1 = \mathbf{E}(W) = \sum_{\alpha\in I} m_{1,\alpha}.$$

Similarly to the Poisson case, we shall need the quantities

$$\delta_1 = \sum_{\alpha\in I} m_{1,\alpha} \sum_{j\geq 1}\sum_{k\geq 1} q_{jk}^{(\alpha)} \mathbf{E}\left|\frac{\mathbf{P}(V_\alpha = j, S_\alpha = k|W_\alpha)}{\mathbf{P}(V_\alpha = j, S_\alpha = k)} - 1\right|$$
$$\delta_2 = 2\sum_{\alpha\in I} \mathbf{E}\left\{V_\alpha d_{\mathrm{TV}}(\mathcal{L}(W_\alpha|V_\alpha, S_\alpha); \mathcal{L}(W_\alpha)\right\}$$
$$\delta_3 = \sum_{\alpha\in I} \left\{\mathbf{E}(V_\alpha U_\alpha) + \mathbf{E}(V_\alpha)\mathbf{E}(V_\alpha + S_\alpha + U_\alpha)\right\}.$$

Then, roughly, δ_3 corresponds to $b_1 + b_2$ in the Poisson case, whereas δ_1 and δ_2 correspond to b_3 in the Poisson case.

The following result can be shown to hold.

Theorem 6.8.3. *There exist constants $H_0 = H_0(\lambda, \underline{\nu})$, $H_1 = H_1(\lambda, \underline{\nu})$, independent of W, such that, with $(\lambda, \underline{\nu})$ given in (6.8.4),*

$$d_{TV}(\mathcal{L}(W), CP(\lambda, \underline{\nu})) \leq H_0 \min(\delta_1, \delta_2) + H_1\delta_3,$$

and

$$H_0, H_1 \leq \min(1, (\lambda\nu_1)^{-1})e^\lambda.$$

If in addition

$$k\nu_k \geq (k+1)\nu_{k+1}, \quad k \geq 1, \tag{6.8.5}$$

then, with $\gamma = \lambda(\nu_1 - 2\nu_2)$,

$$H_0 \leq \min\left\{1, \frac{1}{\sqrt{\gamma}}\left(2 - \frac{1}{\sqrt{\gamma}}\right)\right\}$$
$$H_1 \leq \min\left\{1, \frac{1}{\sqrt{\gamma}}\left(\frac{1}{4\gamma} + \log^+(2\gamma)\right)\right\}.$$

An important special case is the *declumped* situation, that is, W can be written as

$$W = \sum_{\alpha\in I}\sum_{k\geq 1} k 1\!\!1_{\alpha k},$$

where

$$1\!\!1_{\alpha k} = 1\!\!1(\alpha \text{ is the index of the representative of a clump of size } k).$$

For $\alpha \in I, k \in \mathbb{N}$, let $B(\alpha, k) \subset I \times \mathbb{N}$ contain $\{\alpha\} \times \mathbb{N}$; this set can be viewed intuitively as the neighborhood of strong dependence of (α, k).

The canonical parameters are now

$$\lambda = \sum_{\alpha \in I} \sum_{k \geq 1} \mathbf{E}(1\!\!1_{\alpha k})$$
$$\nu_k = \lambda^{-1} \sum_{\alpha \in I} \mathbf{E}(1\!\!1_{\alpha k}). \tag{6.8.6}$$

For example, if $1\!\!1_{\alpha k} = \widetilde{Y}_{i,k}$, then $W = N(w)$, and the canonical parameters are $(n - \ell + 1)\widetilde{\mu}_k, k \geq 1$, and $\tilde{\lambda} = \mathbf{E}(\widetilde{N}(w))$, so that the approximating distribution is as before, $\mathcal{L}(\sum_{k \geq 1} kZ_k)$ with Z_k's independent Poisson variables with parameters $(n - \ell + 1)\widetilde{\mu}_k$. Thus it is the same distribution as in Corollary 6.4.8.

Similarly as in the Poisson case, we shall need the quantities

$$b_1^* = \sum_{(\alpha,k) \in I \times \mathbb{N}} \sum_{(\beta,k') \in B(\alpha,k)} k'k\mathbf{E}(1\!\!1_{\alpha k})\mathbf{E}(1\!\!1_{\beta k'})$$
$$b_2^* = \sum_{(\alpha,k) \in I \times \mathbb{N}} \sum_{(\beta,k') \in B(\alpha,k) \setminus \{(\alpha,k)\}} k'k\mathbf{E}(1\!\!1_{\alpha k} \; 1\!\!1_{\beta k'})$$
$$b_3^* = \sum_{(\alpha,k) \in I \times \mathbb{N}} k\mathbf{E}\left|\mathbf{E}\{1\!\!1_{\alpha k} - \mathbf{E}(1\!\!1_{\alpha k})|\sigma(1\!\!1_{\beta k'}, (\beta, k') \notin B(\alpha, k)\}\right|.$$

The following result holds.

THEOREM 6.8.4. *With the parameters as in (6.8.6), we have that*

$$d_{TV}(\mathcal{L}(W), CP(\lambda, \underline{\nu})) \leq b_3^* H_0 + (b_1^* + b_2^*)H_1.$$

6.8.4. Moment-generating function

Here is a short outline of moment-generating functions. The *moment-generating function M* of a random variable X is defined as

$$\Phi_X(t) = \mathbf{E}(e^{tX}).$$

So, if X has a discrete distribution p, we have that

$$\Phi_X(t) = \sum_x e^{tx} p(x).$$

If the moment-generating function exists for all t in an open interval containing zero, it uniquely determines the probability distribution.

In particular, under regularity conditions the moments of a random variable can be obtained via the moment-generating function using differentiation. Namely, if $\Phi_X(t)$ is finite, we have

$$\Phi'_X(t) = \frac{d}{dt}\mathbf{E}(e^{tX}) = \mathbf{E}(Xe^{tX}).$$

Thus

$$\Phi'_X(0) = \mathbf{E}(X)$$

if both sides of the equation exist. Similarly, differentiating r times we obtain

$$\Phi_X^{(r)}(0) = \mathbf{E}(X^r).$$

A special case is when the moment-generating function $\Phi_X(t)$ is rational, that is, when $\Phi_X(t)$ can be written as

$$\Phi_X(t) = \frac{p_0 + p_1 t + \ldots + p_r t^r}{q_0 + q_1 t + \ldots + q_s t^s} = \sum_d f(d) t^d,$$

for some r, s and coefficients $p_1, \ldots, p_r, q_1, \ldots, q_s$. By normalization we may assume $q_0 = 1$. Then

$$p_0 + p_1 t + \ldots + p_r t^r = \sum_d f(d) t^d (1 + q_1 t + \ldots + q_s t^s).$$

Identification of the coefficients of t^i on both sides yields

$$\begin{aligned} p_i &= \sum_{d=0}^{i} f(d) q_{i-d} \text{ for } i \le r \\ 0 &= \sum_{d=0}^{i} f(d) q_{i-d} \text{ for } i > r. \end{aligned}$$

This gives a recurrence formula for the coefficients $f(d)$; we have

$$\begin{aligned} f(0) &= p_0 \\ f(d) &= p_d - \sum_{i=1}^{\min(d,s)} f(d-i) q_i, \quad d \ge 1 \end{aligned}$$

where $p_d = 0$ for $d > r$.

6.8.5. The δ-method

In general, the δ-method, or *propagation of error*, is a linear approximation (Taylor expansion) of a nonlinear function of random variables. Here we are particularly interested in the validity of a normal approximation for functions of random vectors.

THEOREM 6.8.5. *Let $\underline{X}_n = (X_{n1}, X_{n2}, \ldots, X_{nk})$ be a sequence of random vectors satisfying*

$$b_n(\underline{X}_n - \underline{\mu}) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \Sigma)$$

with $b_n \to \infty$. *The vector valued function* $\underline{g}(\underline{x}) = (g_1(\underline{x}), \ldots, g_\ell(\underline{x}))$ *has real valued* $g_i(\underline{x})$ *with non-zero differential*

$$\frac{\partial g_i}{\partial g_{\underline{x}}} = \left(\frac{\partial g_i}{\partial g_{x_1}}, \ldots, \frac{\partial g_i}{\partial g_{x_k}} \right).$$

Define $\mathbf{D} = (d_{i,j})$ *where* $d_{i,j} = \frac{\partial g_i}{\partial g_{x_j}}(\underline{\mu})$. *Then*

$$b_n(g(\underline{X}_n) - g(\underline{\mu})) \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, \mathbf{D\Sigma D^T}).$$

6.8.6. A large deviation principle

Assume $X_1 \cdots X_n$ is an irreducible Markov chain on a finite alphabet $\mathcal{A}$ with transition probabilities $\pi(a, b)$, $a, b \in \mathcal{A}$. Large deviations from the mean can be described as follows.

Theorem 6.8.6 (Miller). *Let* f *be a function mapping* $\mathcal{A}$ *into* $\mathbb{R}$. *Then,* $n^{-1} \sum_{i=1}^n f(X_i)$ *obeys a large deviation principle with rate function* I *defined below: for every closed subset* $F \subset \mathbb{R}$ *and every open subset* $O \subset \mathbb{R}$,

$$\limsup_{n \to +\infty} \frac{1}{n} \log \mathbf{P} \left(\frac{1}{n} \sum_{i=1}^n f(X_i) \in F \right) \le - \inf_{x \in F} I(x),$$
$$\liminf_{n \to +\infty} \frac{1}{n} \log \mathbf{P} \left(\frac{1}{n} \sum_{i=1}^n f(X_i) \in O \right) \ge - \inf_{x \in O} I(x).$$

The rate function I *is positive, convex, uniquely equal to zero at* $x = \mathbf{E}(f(X_1))$ *and given by*

$$I(x) = \sup_\theta (\theta x - \log \lambda(\theta))$$

where $\lambda(\theta)$ *is the largest eigenvalue of the matrix* $\left(e^{\theta f(b)} \pi(a, b)\right)_{a,b \in \mathcal{A}}$.

6.8.7. A CLT for martingales

For martingales, the following central limit theorem is available.

Theorem 6.8.7. *Let* $(\xi_{n,i})_{i=1,\ldots,n}$ *be a triangular array of* d*-dimensional random vectors such that* $\mathbf{E}||\xi_{n,i}||_2^2 < \infty$, *and* V *be a positive* $d \times d$ *matrix. Put* $\mathcal{F}_{n,i} = \sigma(\xi_{n,1}, \ldots, \xi_{n,i})$; $\mathbf{E}(\xi_{n,i} \,|\, \mathcal{F}_{n,i-1})$ *denotes the conditional expectation vector of* $\xi_{n,i}$ *and* $Cov(\xi_{n,i} \mid \mathcal{F}_{n,i-1})$ *denotes the conditional covariance matrix of* $\xi_{n,i}$. *If as* $n \to \infty$

(i) $\sum_{i=1}^n \mathbf{E}(\xi_{n,i} \mid \mathcal{F}_{n,i-1}) \xrightarrow{\mathbf{P}} 0$,

(ii) $\sum_{i=1}^n Cov(\xi_{n,i} \mid \mathcal{F}_{n,i-1}) \to V$,

$$\textit{(iii)}\ \forall \varepsilon > 0, \sum_{i=1}^{n} \mathbf{P}(|\xi_{n,i}| > \varepsilon \mid \mathcal{F}_{n,i-1}) \xrightarrow{\mathbf{P}} 0,$$

then $\sum_{i=1}^{n} \xi_{n,i} \xrightarrow{\mathcal{D}} \mathcal{N}(0, V)$.

Notes

The material in this chapter can be viewed as an updated version of Reinert et al. (2000). Recent progress on exact distributional results, as well as on compound Poisson approximation and on multivariate normal approximation, is included.

This chapter does not deal with the algorithmic issues; an excellent starting point would be Waterman (1995) or Gusfield (1997). For a particular example see also Apostolico et al. (1998), and for a recent exposition see Lonardi (2001).

Number of clumps. Equations (6.2.6) and (6.2.9) that characterize the occurrence of a clump, or a k-clump, of the word w at a given position with respect to the periods of w are due to Schbath (1995a).

Word locations. The recursive formula for the exact distribution of the distance D between two word occurrences (Theorem 6.3.1) is from Robin and Daudin (1999). It was first proposed for independent and uniformly distributed letters by Blom and Thorburn (1982). The moment-generating function of the distance D, expressed as a rational function and given in Theorem 6.3.2, also comes from Robin and Daudin (1999). Recently, Stefanov (2003) obtained another expression for the generating function that avoids the calculation of the "infinite" sum of the Π^u's.

Similar results are derived in Robin and Daudin (2001) and Stefanov (2003) for the probability distribution of the distance between any word in a given set. They are not presented in Section 6.6 but are useful for instance for the purpose of calculating the occurrence probability of a structured motif (Robin et al. (2002)). These motifs are of particular interest since they are candidate promotors for transcription. Indeed, a structured motif is of the form $v(d_1 : d_2)w$, denoting a word v separated from a word w by a distance between d_1 and d_2; where v and w can be approximate patterns. Efficient algorithms exist to find such structured motifs (Marsan and Sagot (2000a)).

A related problem concerns the position T_1 of the first occurrence of a word; it is treated in Rudander (1996) and more recently in Stefanov (2003). The moment generating function of T_1 given on page 271 is taken from Robin and Daudin (1999).

The Poisson approximation for the statistical distribution of the k-smallest r-scan presented on page 268 is due to Dembo and Karlin (1992). This approximation is very useful for the comparison between the expected distribution of the r-scans and the one observed in the biological sequence. It has been first applied in Karlin and Macken (1991) to the *E. coli* genome by approximating the r-scan distribution given in Section 6.3.1 by a sum of $r-1$ independent exponential random variables. Robin and Daudin (2001) refined this approximation using the exact distribution of the r-scans. Related work is presented by Robin (2002) but using a compound Poisson model for the word occurrences rather than a Markov model for the sequence of letters. This new approach has the advantage of taking the eventual unexpected frequency of the word of interest into account when analyzing its location along a sequence. See Glaz et al. (2001) for more material and applications of scan statistics.

Word count distribution. The method of obtaining the exact distribution of the word count presented here generalizes the result that Gentleman and Mullin (1989) obtained for the case that the sequence is composed of i.i.d. letters, where each letter occurs with equal probability. In this case, Gentleman (1994) also gives an algorithm for calculating the word frequency distribution. Moreover, in the Markov case the exact distribution of the count can also be obtained by other techniques: Kleffe and Langbecker (1990) as well as Nicodème et al. (2002) used an automaton built on the pattern structure matrix, whereas Régnier (2000) used a language decomposition approach to obtain the generating function of the count (see Chapter 7).

The variance (6.4.1) of the count $N(w)$ is inspired by Kleffe and Borodovsky (1992).

Gaussian approximation. The asymptotic normality of the difference between the word count and its estimator was first proposed by Lundstrom (1990) using the δ-method. For an exposition, see Waterman (1995). The two alternative approaches presented in this chapter, the martingale and the conditional ones, have the advantage to provide explicit formulas for the asymptotic variance. They are both due to Prum et al. (1995) for the first order Markov chain model, and to Schbath (1995b) for higher order models and phased models. The conditional expectation of the count is originally due to Cowan (1991).

The bound Theorem 6.4.4 on the distance to the normal distribution was obtained by Huang (2002). This paper, and references therein, discusses also the constant c which appears in the bound. The result in the independent case was first presented in Reinert et al. (2000).

Poisson and compound Poisson approximations. When the sequence letters are independent, Poisson and compound Poisson approximations for $N(w)$ have been widely studied in the literature (Chryssaphinou and Papastavridis (1988a), Chryssaphinou and Papastavridis (1988b), Arratia et al. (1990), Godbole (1991), Hirano and Aki (1993), Godbole and Schaffner (1993), Fu (1993)). Markovian models under different conditions have then been considered (Rajarshi (1974), Godbole (1991), Godbole and Schaffner (1993), Hirano and Aki (1993), Geske et al. (1995), Schbath (1995a), Erhardsson (1997)), but few works concern general periodic words and provide explicit parameters of the limiting distribution. Our two basic references in this chapter are Arratia et al. (1990) and Schbath (1995a).

For the compound Poisson and Poisson approximation error term due to the estimation of the transition probabilities, refer to Schbath (1995b). Reinert and Schbath (1998) showed that the end effects due to the finite sequence are negligible for the count (Equation (6.4.11)) and the count of clumps.

The special case of runs of 1 in a random sequence of letters in the binary alphabet $\{0, 1\}$ is extensively studied: Erdős and Rényi (1970) gave the asymptotic behavior of the longest run in a sequence of Bernoulli trials, and of the length of the longest segment that contains a proportion of 1 greater than a predescribed level α. Their result was refined by Guibas and Odlyzko (1980), Deheuvels et al. (1986), and Gordon et al. (1986). The compound Poisson approximation for counts of runs in the case where the sequence letters are independent was considered by Eichelsbacher and Roos (1999), also employing the Chen-Stein method using results by Barbour and Utev (1998) (the limiting distribution is the same as the one given in (6.4.15), reduced to this special case). Barbour and Xia (1999) obtained a more accurate limiting approximation for the case of runs of length 2; this approximation is based on a perturbation of a Poisson distribution.

Direct compound Poisson approximation. Theorem 6.4.9, which presents a direct compound Poisson approximation of the count, is due to Barbour et al. (2001). They give a more general form of the result, and also a bound for the Kolmogorov distance. Using the approach by Erhardsson (1999), they also derive a slightly less explicit but asymptotically better bound in terms of stopping times for a Markov chain.

Indeed, in Erhardsson (1997), Erhardsson (1999) and Erhardsson (2000), a different approach based on the direct compound Poisson approximation Theorem 6.8.3 is developed. The idea is to express counts of events as numbers of visits of a certain Markov chain to a rare set, and to use regeneration cycles for suitable couplings. It results in bounds that are formulated in terms of stopping times of Markov chains. Results of this type are less explicit, but they have asymptotic order $O(n^{-1})$ under the typical regime $n\mu(w) = O(1)$, see also Barbour et al. (2001) and Gusto (2000), whereas the bounds in Theorem 6.4.9 and in Corollary 6.4.8 (which is from Schbath (1995a)) are of order $O(n^{-1} \log n)$ under the same regime.

Numerical experiments in Barbour et al. (2001) display that the bound in Theorem 6.4.9 and the bound from the Erhardsson (1997)-approach perform better than the bound in Corollary 6.4.8 for the word `acgacg` in the bacteriophage *Lambda* ($n = 48, 502$) under three different transition matrices. In contrast, Gusto (2000) compared the result from Erhardsson (1999) to the one in Schbath (1995a) and did not observe any marked improvement for all words of length 8 in the bacteriophage *Lambda*. This may illustrate that, whereas the compound Poisson approximation via a Poisson process approximation works well in the case of rare words, it does not yield the best bounds in the case of not so rare words.

Approximation using a large deviation principle. Section 6.4.6 is inspired by

Schbath (1995b). Nuel (2001) obtained a better approximation using a large deviation principle for the empirical distribution of the ℓ-letter words. This empirical distribution is defined as the random measure $L_{n,\ell}$ on $\mathcal{A}^\ell$ such that, for $w \in \mathcal{A}^\ell$,

$$L_{n,\ell}(w) = \frac{1}{n-\ell+1} \sum_{i=1}^{n-\ell+1} Y_i(w),$$

so that $L_{n,\ell}(w) = N(w)$. However, the definition of the large deviation rate funtion and its mathematical treatment are more involved than the one given in Section 6.4.6.

Renewal count distribution. For a classical introduction to renewals, see Chapter 13 in Feller (1968). Exact results for the distribution of R_n can be found in Régnier (2000). When the letters $X_1, \ldots, X_n$ are independent and identically distributed, the asymptotic distribution of the renewal count was studied by Breen et al. (1985) and Tanushev and Arratia (1997). The Central Limit Theorem 6.5.1 in the Markovian case is due to Tanushev (1996). He also proved a multivariate approximation. The theorem is much easier to prove in the i.i.d. case, see Waterman (1995). The main technique being generating functions, no bound on the rate of convergence is obtained.

The Poisson approximation for renewals based on the Poisson approximation for the number of clumps is the idea behind the proof of Geske et al. (1995), although Geske et al. (1995) prove the result only for words having at most one principal period. Related results have been obtained by Chryssaphinou and Papastavridis (1988b). Theorem 6.5.2 is due to Chryssaphinou et al. (2001); they also derive the stated conditions under which convergence to a Poisson distribution holds.

Occurrences of multiple patterns. The multivariate generating function of the counts of multiple words can be found in Régnier (2000) and can be derived from Robin and Daudin (2001). The methods used are extensions of the ones presented in Subsection 6.4.1.

The covariance was also calculated in Lundstrom (1990), in a different form. Theorem 6.6.1 is proven in Huang (2002); there it is also shown that $\mathcal{L}_n$ is invertible as well as a discussion of the constant c; see also references therein. As in Rinott and Rotar (1996), Huang (2002) considers more general classes of test functions as well, but not as general as to cover total variation.

The Poisson and compound Poisson approximations for the joint distribution of declumped counts and multiple word counts presented here are due to Reinert and Schbath (1998). Recently, Chen and Xia (pear) obtained a much improved bound for the independent model, in the Wasserstein metric (which is weaker than the total variation metric), for the Poisson approximation of counts of palindromes, assuming the four-letter alphabet $\mathcal{A} = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$ and that $p_\mathtt{a} = p_\mathtt{t}, p_\mathtt{c} = p_\mathtt{g}$. Formula (6.6.6) is due to Chryssaphinou et al. (2001).

Tanushev (1996) studied non-overlapping occurrences in competitions with each other, including the derivation of the mean for the number of competing renewal counts, and, most notably, the normal approximation Theorem 6.6.6. The mean of the total number of competing renewals, $\sum_{r=1}^{q} R_n^c(w^r)$, has recently been presented in a slightly simpler form by Chryssaphinou et al. (2001). Also the alternative approach for a Poisson approximation for competing renewal counts is given in Chryssaphinou et al. (2001).

Some applications to DNA sequences. The quality of the approximate p-values was extensively studied in Robin and Schbath (2001); their results here are combined with the approximate scores obtained with the large deviation approach of Nuel (2001).

The details on the treatment of sequencing by hybridization as presented here are given in Arratia et al. (1996). The characterization of unique recoverability from the ℓ-spectrum is due to Pevzner (1989); Ukkonen (1992) conjectured and Pevzner (1995) proved that there are exactly three structures that prevent unique recoverability. De Bruijn graphs are described in van Lint and Wilson (1992). Theorem 6.7.2 is from Arratia et al. (1996), where more detailed versions are also given. This bound is improved by Shamir and Tsur (2001). In Arratia et al. (1996), a more general result is derived for general alphabets, and explicit bounds are obtained. These bounds can be used to approximate the probability of unique recoverability. Arratia et al. (2000) have obtained results on the number of possible reconstructions for a given sequence (when the reconstruction is not unique).

Some probabilistic and statistical tools. Stein's method for the normal approximation was first published by Stein (1972). Rinott and Rotar (1996) applied it to obtain multivariate normal approximations with a bound on the error in the distance of suprema over convex sets, which yields Theorem 6.8.1. Indeed, Rinott and Rotar (1996) derive the result for more general classes of test functions.

First published by Chen (1975) as the Poisson analog to Stein's method for normal approximations (Stein (1972)), the Chen-Stein method for Poisson approximation has found widespread application; word counts being just one of them. A friendly exposition is found in Arratia et al. (1989) and a description with many examples can be found in Arratia et al. (1990) and Barbour et al. (1992). The key theorem for word counts in stationary Markov chains is Theorem 1 in Arratia et al. (1990) with an improved bound by Barbour et al. (1992) (Theorem 1.A and Theorem 10.A), giving Theorem 6.8.2.

Much of the subsection on direct compound Poisson approximation is based on the overview of Barbour and Chryssaphinou (2001). This approach started with Barbour et al. (1992); see also Roos (1993), Barbour and Utev (1998). Recently much attention has been given to this problem, and the reader is referred to the references in Barbour and Chryssaphinou (2001).

For δ_3, in Barbour and Chryssaphinou (2001) there is an additional, alternative quantity given in terms of the Wasserstein distance between two distributions. Instead of Condition (6.8.5), improved bounds on H_0 and H_1 are also available under the condition that $m^{-1}(m_2 - m_1) < 1/2$, where $m_2 = \sum_{k \geq 1} k^2 \nu_k$, see Barbour and Chryssaphinou (2001). Barbour and Chryssaphinou (2001) also obtain Theorem 6.8.3, which in their paper is also phrased in the Kolmogorov distance, and slightly more general, and Theorem 6.8.4. Barbour and Chryssaphinou (2001) also provide refined versions of this approach as well as results in Kolmogorov distance. Barbour and Mansson (2002) give related results in Wasserstein distance.

A short outline of moment-generating functions can be found e.g., in Rice (1995). Theorem 6.8.5 on the delta method can be found for example on p.313 in Waterman (1995). The large deviation principle Theorem 6.8.6 for Markov chains can be found on p.78 in Bucklew (1990). The martingale central limit theorem 6.8.7 is in Dacunha-Castelle and Duflo (1983) p.80.

General tools. The autocorrelation polynomial was introduced by Guibas and Odlyzko (1980); see also Li (1980), Biggins and Cannings (1987). The result that two words commute if and only if they are powers of the same word can be found in Lothaire (1997). The Perron–Frobenius Theorem used on page 255 is classical; see for example Karlin and Taylor (1975). The Chi-square test for independence is textbook material in statistics; Rice (1995) gives a good exposition. The case of general order Markov chains is reviewed in Billingsley (1961). However, for higher order, a longer sequence of observations is required (see Guthrie and Youssef (1970)). For an introduction to martingales, see, e.g., Chung (1974). The Law of Iterated Logarithm for Markov chains is due to Senoussi (1990).

Genome analysis. The first analysis of the restriction sites in *E. coli* was carried out by Churchill et al. (1990) while analyzing the distance between those sites. Avoidance of restriction sites in *E. coli* was first presented by Karlin et al. (1992). The Cross-over Hotspot Instigator sites are very important for several bacteria (see Biaudet et al. (1998), Chedin et al. (1998), Sourice et al. (1998)). Their significant abundances have been first showed in Schbath (1995b) for *E. coli* and then in El Karoui et al. (1999) for other bacteria. Several papers aim at identifying over- and under-represented words in a particular genome (for instance, Leung et al. (1996), Rocha et al. (1998)). They usually use the maximal Markov model (see also Brendel et al. (1986)). The Poisson approximation used in BLAST to approximate the p-value of a sequence alignment was first proposed in Altschul et al. (1990), and proven in Karlin and Dembo (1992). The variational composition of a genome have been studied with HMMs by Churchill (1989), Muri (1998), Durbin et al. (1998).

CHAPTER 7

Analytic Approach to Pattern Matching

7.0. Introduction

Repeated patterns and related phenomena in words are known to play a central role in many facets of computer science, telecommunications, coding, data compression, and molecular biology. One of the most fundamental questions arising in such studies is the frequency of pattern occurrences in another string known as the *text*. Applications of these results include gene finding in biology, code synchronization, user search in wireless communications, detecting signatures of an attacker in intrusion detection, and discovering repeated strings in the Lempel-Ziv schemes and other data compression algorithms.

The basic *pattern matching* is to find for a given (or random) pattern w or a set of patterns $\mathcal{W}$ and text X how many times $\mathcal{W}$ occurs in the text and how long it takes for $\mathcal{W}$ to occur in X for the first time. These two problems are not unrelated as we have already seen in Chapter 6. Throughout this chapter we allow patterns to overlap and we count overlapping occurrences separately. For example, $w = abab$ occurs three times in the text $= b\mathtt{abababab}b$.

We consider pattern matching problems in a probabilistic framework in which the text is generated by a probabilistic source while the pattern is given. In Chapter 1 various probabilistic sources were discussed. Here we succinctly summarize assumptions adopted in this chapter. In addition, we introduce a new general source known as a *dynamic source* recently proposed by Vallée. In Chapter 2 algorithmic aspects of pattern matching and various efficient algorithms for finding patterns were discussed. In this chapter, as in Chapter 6, we focus on *analysis*. However, unlike Chapter 6, we apply here analytic tools of combinatorics and analysis of algorithms to discover general laws of pattern occurrences. An immediate consequence of our results is the possibility to set *thresholds* at which a pattern in a text starts being (statistically) meaningful.

The approach we undertake to analyze pattern matching problems is through a formal description by means of regular languages. Basically, such a description of *contexts* of one, two, or several occurrences gives access to expectation, variance, and higher moments, respectively. A systematic translation into *generating functions* of a complex variable z is available by methods of analytic combinatorics deriving from the original Chomsky-Schützenberger theorem. Then, the structure of the implied generating functions at a pole, usually at $z = 1$, provides the necessary asymptotic information. In fact, there is an important phenomenon of *asymptotic simplification* where the essentials of combinatorial-probabilistic features are reflected by the singular forms of generating functions. For instance, variance coefficients come out naturally from this approach together with a suitable notion of correlation. Perhaps the originality of the present approach lies in such a joint use of combinatorial-enumerative techniques and of analytic-probabilistic methods.

There are various pattern matching problems. In its simplest form, the pattern $\mathcal{W} = w$ is a single string w and one searches for some/all occurrences of w as a block of consecutive symbols in the text. This problem is known as the *exact string matching* and its analysis is presented in Section 7.2 (cf. also Chapter 6). We adopt a symbolic approach, and first describe a language that contains all occurrences of w. Then we translate this language into a generating function that will lead to precise evaluation of the mean and the variance of the number of occurrences of the pattern. Finally, we prove the central and local limit laws, and large deviations.

In the *generalized string matching* problem the pattern $\mathcal{W}$ is a set rather than a single pattern. In its most general formulation, the pattern is a pair $(\mathcal{W}_0, \mathcal{W})$ where $\mathcal{W}_0$ is the so called *forbidden set.* If $\mathcal{W}_0 = \emptyset$, then $\mathcal{W}$ appears in the text whenever a word from $\mathcal{W}$ occurs as a string with overlapping allowed. When $\mathcal{W}_0 \neq \emptyset$ one studies the number of occurrences of strings in $\mathcal{W}$ under the condition that there is no occurrence of a string from $\mathcal{W}_0$ in the text X. This could be called a *restricted* string matching since one restricts the text to those strings that do not contain strings from $\mathcal{W}_0$. Finally, setting $\mathcal{W} = \emptyset$ (with $\mathcal{W}_0 \neq \emptyset$) we search for the number of text strings that do not contain any pattern from $\mathcal{W}_0$. In particular, for $\ell \leq k$ if $\mathcal{W}_0$ consists of runs of zeros of length at least ℓ and at most k, then we deal with the so called (ℓ, k) sequences that find application in magnetic recoding.

We shall present a complete analysis of the generalized string matching problem in Section 7.3. We first consider the so called *reduced set of patterns* in which a string in $\mathcal{W}$ cannot be a substring of another string in $\mathcal{W}$. We shall generalize our combinatorial language approach from Section 7.2 to derive the mean, variance, central and local limit laws, and large deviations. Then we analyze the generalized string pattern matching with $\mathcal{W}_0 = \emptyset$ and adopt a different approach. We shall construct an automaton to recognize the pattern $\mathcal{W}$ that turns out to be a de Bruijn graph. The generating function of the number of occurrences will have a matrix form with the main matrix representing the transition matrix of the associated de Bruijn graph. Finally, we consider the (ℓ, k) sequences and enumerate them leading to the Shannon capacity.

In Section 7.4 we discuss a new pattern matching problem called the *subsequence pattern matching* or the *hidden pattern matching.* In this case the pattern $\mathcal{W} = a_1 a_2 \cdots a_m$, where a_i is a symbol of the underlying alphabet, is to occur as a *subsequence* rather than a string (consecutive symbols) in a text. We say that $\mathcal{W}$ is hidden in the text. For example, `date` occurs as a subsequence in the text `hidden pattern`, in fact four times, but not even once as a string. The gaps between occurrences of $\mathcal{W}$ may be bounded or unrestricted. The extreme cases are: *fully unconstrained* problem where all gaps are unbounded; and the *fully constrained* problem where all gaps are bounded. We analyze these and mixed cases.

In Section 7.5 we generalized all of the above pattern matching problems and analyze the *generalized subsequence problem.* In this case, the pattern is $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_d)$ where $\mathcal{W}_i$ is a collection of strings (a language). We say that the generalized pattern $\mathcal{W}$ occurs in the text X if X contains $\mathcal{W}$ as a *subsequence* $(w_1, w_2, \ldots, w_d)$ where $w_i \in \mathcal{W}_i$. Clearly, it includes all the problems discussed so far. We shall analyze this generalized pattern matching for general probabilistic dynamic sources (which include among others Markov sources and mixing sources). The novelty of the analysis lies in translating probabilities into composition of operators. Under a mild decomposability assumption, these operators entertain spectral representations that allows us to derive precise asymptotic behavior for quantities of interest.

Finally, in the last section we study a different pattern matching, namely the one in which the pattern is part of the (random) text. We coin the term *self-repetitive pattern matching.* More precisely, we look for the longest substring of the text occurring at a given position that has another copy in the text. This new quantity, when averaged over all possible positions of the text, is actually the typical *depth* in a suffix trie (cf. Chapter 2) built over (randomly generated) text. We analyze it using analytic techniques such as generating functions and the Mellin transform. We reduce its analysis to the exact pattern matching; thus we call the technique the *string-ruler* method. In fact, we prove that the probability generating function of the depth in a suffix trie is asymptotically close to the probability generating function of the depth in a trie that is built over n *independently* generated texts. Such tries have been extensively studied in the past and we have pretty good understanding of their probabilistic behaviors. This allows us to conclude that the depth in a suffix trie is asymptotically normal.

7.1. Probabilistic models

We study here pattern matching in a probabilistic framework in which the text is generated randomly. Let us first introduce some general probabilistic models of generating sequences. The reader is also referred to Chapter 1 for a brief introduction to probabilistic models. For the convenience of the reader, we repeat here some definitions.

Throughout we shall deal with sequences of discrete random variables. We write $(X_k)_{k=1}^{\infty}$ for a one-sided infinite sequence of random variables; however, we often abbreviate it as X provided it is clear from the context that we are talking about a sequence, not a single variable. We assume that the sequence $(X_k)_{k=1}^{\infty}$ is defined over a finite alphabet $\mathcal{A} = \{a_1, \ldots, a_V\}$ of size V. A partial sequence is denoted as $X_m^n = (X_m, \ldots, X_n)$ for $m < n$. Finally, we shall always assume that a probability measure exists, and we write $P(x_1^n) = \mathbf{P}(X_k = x_k,\ 1 \leq k \leq n,\ x_k \in \mathcal{A})$ for the probability mass, where we use lowercase letters for a realization of a stochastic process.

Sequences are generated by information sources, usually satisfying some constraints. We also call them *probabilistic models*. Throughout, we assume the existence of a stationary probability distribution, that is, for any string w the probability that the text X contains an occurrence of w at position k is equal to $P(w)$ independently of the position k. For $P(w) > 0$, we denote by $P(u \mid w)$ the conditional probability equals $P(wu)/P(w)$.

The most elementary source is a *memoryless source* also known as the *Bernoulli source*.

(B) Memoryless or Bernoulli Source

Symbols of the alphabet $\mathcal{A} = \{a_1, \ldots, a_V\}$ occur independently of one another; thus $X = X_1X_2X_3\ldots$ can be described as the outcome of an infinite sequence of Bernoulli trials in which $\mathbf{P}(X_j = a_i) = p_i$ and $\sum_{i=1}^{V} p_i = 1$. Throughout, we assume that at least for one i we have $0 < p_i < 1$.

In many cases, assumption (B) is not very realistic. When this is the case, assumption (B) may be replaced by:

(M) Markov Source of order one

There is a Markovian dependency between consecutive symbols in a string; that is, the probability $p_{ij} = \mathbf{P}(X_{k+1} = a_j | X_k = a_i)$ describes the conditional probability of sampling symbol a_j immediately after symbol a_i. We denote by $\mathrm{P} = \{p_{ij}\}_{i,j=1}^{V}$ the transition matrix, and by $\mu = (\pi_1, \ldots, \pi_V)$ the stationary vector satisfying $\mu\mathrm{P} = \mu$. (Throughout, we assume that the Markov chain is irreducible and aperiodic.) A general Markov source of order r is characterized by the transition matrix $V^r \times V$ with coefficients being $P(j \in \mathcal{A} \mid u)$ for $u \in \mathcal{A}^r$.

In some situations more general sources must be considered (for which one still can obtain reasonably precise analysis). Recently, Vallée introduced new sources called *dynamic sources* that we briefly describe here and use in the analysis of the generalized subsequence problem in Section 7.5. To introduce such sources we start with a description of a *dynamic system* defined by:

- A topological partition of the unit interval $\mathcal{I} := (0, 1)$ into a disjoint set of open intervals $\mathcal{I}_a, a \in \mathcal{A}$.
- An encoding mapping χ which is constant and equal to $a \in \mathcal{A}$ on each $\mathcal{I}_a$.
- A shift mapping $T: \mathcal{I} \to \mathcal{I}$ whose restriction to $\mathcal{I}_a$ is a bijection of class $\mathcal{C}^2$ from $\mathcal{I}_a$ to $\mathcal{I}$. The local inverse of T restricted to $\mathcal{I}_a$ is denoted by h_a.

Observe that such a dynamic system produces infinite words of $\mathcal{A}^\infty$ through the encoding χ. For an initial $x \in \mathcal{I}$ the source outputs a word, say $w(x) = (\chi x, \chi T x, \ldots)$.

(DS) Probabilistic Dynamic Source

A source is called a *probabilistic dynamic source*, if the unit interval of a dynamic system is endowed with a density f.

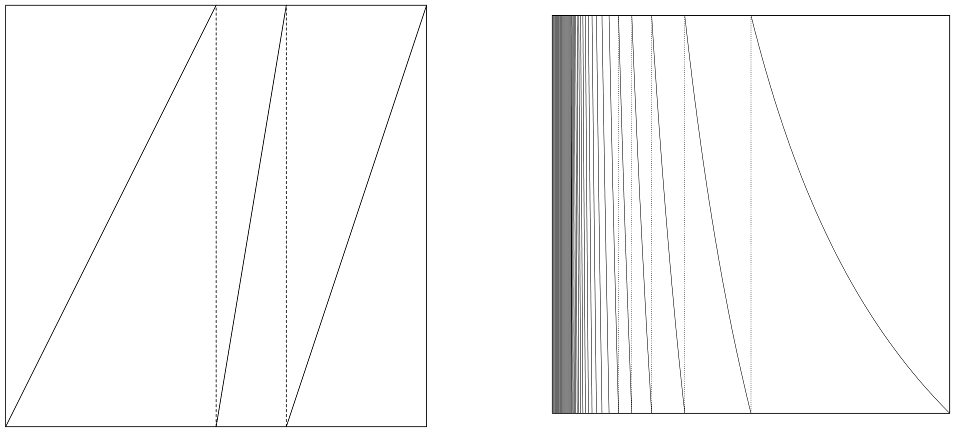
Example 7.1.1. A memoryless source associated with the probability distribution $\{p_i\}_{i=1}^{V}$ (where V can be finite or infinite) is modeled by a dynamic source in which the components $w_k(x) = \chi T^k x$ are independent and the corresponding topological partition of $\mathcal{I}$ is defined as

$$\mathcal{I}_m := (q_m, q_{m+1}], \qquad q_m = \sum_{j<m} p_j.$$

In particular, a symmetric V-ary memoryless source can be described as

$$T(x) = \{Vx\}, \qquad \chi(x) = \lfloor Vx \rfloor,$$

where $\lfloor x \rfloor$ is the integer part of x and $\{x\} = x - \lfloor x \rfloor$ is the fractional part of x (cf. Figure 7.1(a)).



(a) (b)

Figure 7.1. Dynamic Sources discussed in Example 7.1.1: (a) memoryless with the shift mapping $T_m(x) = \langle (x - q_m)/p_{m+1} \rangle$ (b) continued fraction source with $T_m(x) = 1/x - m = \langle 1/x \rangle$.

Here is another example of a source with memory related to continued fractions. The alphabet $\mathcal{A}$ is the set of all natural numbers and the partition of $\mathcal{I}$ is defined as $\mathcal{I}_m = (1/(m+1), 1/m)$. The restriction of T to $\mathcal{I}_m$ is the decreasing linear fractional transformation $T(x) = 1/x - m$, that is,

$$T(x) = \{1/x\}, \qquad \chi(x) = \lfloor 1/x \rfloor.$$

Observe that the inverse branches h_m are defined as $h_m(x) = 1/(x+m)$ (cf. Figure 7.1(b)).

Let us observe that a word of length k, say $w = w_1 w_2 \cdots w_k$ is associated with the mapping $h_w := h_{w_1} \circ h_{w_2} \circ \cdots \circ h_{w_k}$ which is an inverse branch of T^k. In fact, all words that begin with the same prefix w belong to the same *fundamental interval* defined as $\mathcal{I}_w = (h_w(0), h_w(1))$. Furthermore, for probabilistic dynamic sources with the density f, one easily computes the probability of w as the measure of the interval $\mathcal{I}_w$.

The probability $P(w)$ of a word w can be explicitly computed through the special *generating operator* $\mathbf{G}_w$ define as follows

$$\mathbf{G}_w[f](t) := |h'_w(t)| f \circ h_w(t). \tag{7.1.1}$$

One recognizes in $\mathbf{G}_w[f](t)$ a density mapping, that is, $\mathbf{G}_w[f](t)$ is the density of f mapped over $h_w(t)$. The probability of w can then be computed as

$$P(w) = \left| \int_{h_w(0)}^{h_w(1)} f(t)dt \right| = \int_0^1 |h'_w(t)| f \circ h_w(t) dt = \int_0^1 \mathbf{G}_w[f](t)dt. \tag{7.1.2}$$

Let us now consider a concatenation of two words w and u. For memoryless sources $P(w \cdot u) = P(w)P(u)$. For Markov sources one still obtains the product of *conditional* probabilities. Dynamic sources replaces the product of probabilities by the product (composition) of generating operators. To see this, we observe that

$$\mathbf{G}_{w \cdot u} = \mathbf{G}_u \circ \mathbf{G}_w, \tag{7.1.3}$$

where we write $\mathbf{G}_w := \mathbf{G}_w[f](t)$. Indeed, $h_{wu} = h_w \circ h_u$ and $\mathbf{G}_{w \cdot u} = h'_w \circ h_u \cdot h'_u \cdot f \circ h_w \circ h_u$ while $\mathbf{G}_w = h'_w \cdot f \circ h_w$ and then $\mathbf{G}_u \circ \mathbf{G}_w = h_u \cdot h'_w \circ h_u \cdot f \circ h_w \circ h_u$, as desired.

7.2. Exact string matching

In the *exact string matching* problem the pattern $w = w_1 w_2 \cdots w_m$ of length m is *given* while the text $X = X_1^n = X_1 \ldots X_n$ of length n is generated by a random source. Observe that since the pattern $\mathcal{W}$ is given, its length m will *not* vary with n when $n \to \infty$ (asymptotic analysis).

There are several parameters of interest in the string matching, but two of them stand out. Namely, the number of times w occurs in X which we denote as $N_n := N_n(w)$ and define formally by

$$N_n(w) = \mathrm{Card}(\{i : \; X_{i-m+1}^i = w, \; m \le i \le n\}).$$

We can write $N_n(w)$ in an equivalent form as follows

$$N_n(w) = I_m + I_{m+1} + \cdots + I_n \tag{7.2.1}$$

where $I_i = 1$ if w occurs at position i and $I_i = 0$ otherwise.

The second parameter is the *waiting time* T_w defined as the first time w occurs in the text X, that is,

$$T_w := \min\{n : \; X_{n-m+1}^n = w\}.$$

One can also define T_j as the minimum length of the text in which the pattern w occurs j times. Clearly, $T_w = T_1$. These parameters are not independent since

$$\{T_w > n\} = \{N_n(w) = 0\}. \tag{7.2.2}$$

More generally,

$$\{T_j \le n\} = \{N_n(w) \ge j\}. \tag{7.2.3}$$

Relation (7.2.3) is called the *duality principle* in Chapter 6.

Our goal is to estimate the frequency of pattern occurrences N_n in a text generated by a Markov source. We allow patterns to overlap when counting occurrences (e.g., if $w = abab$, then it occurs twice in $X = abababb$ when overlapping is allowed; it occurs only once if overlapping is not allowed). We study probabilistic behavior of N_n through two generating functions, namely:

$$N_r(z) = \sum_{n\geq 0} \mathbf{P}(N_n(w) = r) z^n,$$
$$N(z,u) = \sum_{r=1}^{\infty} N_r(z) u^r = \sum_{r=1}^{\infty}\sum_{n=0}^{\infty} \mathbf{P}(N_n(w) = r) z^n u^r$$

that are defined for $|z| \leq 1$ and $|u| \leq 1$.

Throughout this section we adopt a combinatorial approach to string matching, that is, we use combinatorial calculus to find combinatorial relationships between sets of words satisfying certain properties (i.e., languages). Alternatively, we could start with the representation (7.2.1) and use probabilistic tools along the lines already discussed in Chapter 6.

7.2.1. Languages representations

We start our combinatorial analysis with some definitions. For any language $\mathcal{L}$ we define its *generating function* $L(z)$ as

$$L(z) = \sum_{u\in\mathcal{L}} P(u) z^{|u|},$$

where $P(u)$ is the stationary probability of u occurrence, $|u|$ is the length of u, and we assume that $P(\varepsilon) = 1$. Notice that $L(z)$ is defined for all complex z such that $|z| < 1$. In addition, we define the *w-conditional generating function* of $\mathcal{L}$ as

$$L_w(z) = \sum_{u\in\mathcal{L}} P(u|w) z^{|u|} = \sum_{u\in\mathcal{L}} \frac{P(wu)}{P(w)} z^{|u|},$$

Since we allow overlaps, the structure of the pattern has a profound impact on the number of occurrences. To capture this, we introduce the autocorrelation language and the autocorrelation polynomial. Given a string w, we define the *autocorrelation set* $\mathcal{S}$ as:

$$\mathcal{S} = \{w_{k+1}^m : \; w_1^k = w_{m-k+1}^m\}. \tag{7.2.4}$$

By $\mathcal{P}(w)$ we denote the set of positions $k \geq 1$ satisfying $w_1^k = w_{m-k+1}^m$. In other words, if $w = vu$ and $w = ux$ for some words v, x and u, then x belongs to $\mathcal{S}$ and $|u| \in \mathcal{P}(w)$. Notice that $\varepsilon \in \mathcal{S}$. The generating function of the language $\mathcal{S}$ is denoted as $S(z)$ and we call it the *autocorrelation polynomial*. Its w-conditional generating function is denoted $S_w(z)$. In particular, for Markov sources (of order one)

$$S_w(z) = \sum_{k\in\mathcal{P}(w)} P(w_{k+1}^m \mid w_k^k) z^{m-k}. \tag{7.2.5}$$

Before we proceed, let us present a simple example illustrating the definitions introduced so far.

Example 7.2.1. Let us assume that $w = aba$ over a binary alphabet $\mathcal{A} = \{a, b\}$. Observe that $\mathcal{P}(w) = \{1, 3\}$ and $\mathcal{S} = \{\varepsilon, ba\}$, where ε is the empty word. Thus, for the unbiased memoryless source we have $S(z) = 1 + \frac{z^2}{4}$, while for the Markovian model of order one, we obtain $S_{aba}(z) = 1 + p_{ab}p_{ba}z^2$.

Our goal is to estimate the number of pattern occurrences in a text. Alternatively, we can seek the generating function of a language that consists of all words containing some occurrences of w. Given a pattern w, we introduce the following languages:

(i) $\mathcal{T}_r$ as a set of words containing exactly r occurrences of w.

(ii) $\mathcal{R}$ as a set of words containing only one occurrence of w, located at the right end.

(iii) $\mathcal{U}$ defined as

$$\mathcal{U} = \{u : \ w \cdot u \in \mathcal{T}_1\}, \tag{7.2.6}$$

that is, a word $u \in \mathcal{U}$ if $w \cdot u$ has exactly one occurrence of w at the left end of $w \cdot u$.

(iv) $\mathcal{M}$ defined as

$$\mathcal{M} = \{v : \ w \cdot v \in \mathcal{T}_2 \text{ and } w \text{ occurs at the right end of } w \cdot v\},$$

that is, $\mathcal{M}$ is a language such that any word in $w \cdot \mathcal{M}$ has exactly two occurrences of w at the left and right end.

Example 7.2.2. Let $\mathcal{A} = \{a, b\}$ and $w = abab$. Then $r = aa\mathtt{abab} \in \mathcal{R}$ since there is only one occurrence of w at the right end of r. Also, $u = bbbb \in \mathcal{U}$ since wu has only one occurrence of w at the left end; but $v = abbbb \notin \mathcal{U}$ since $wv = abababbbb$ has two occurrences of w. Furthermore, $ab \in \mathcal{M}$ since $wm = \mathtt{ababab} \in \mathcal{T}_2$ has two occurrences of w at the left and the right ends. Finally, $t = bb\mathtt{ababab}bb\mathtt{abab}bb \in \mathcal{T}_3$ and one observes that $t = rm_1m_2u$ where $r = bbabab \in \mathcal{R}$, $m_1 = ab \in \mathcal{M}$, $m_2 = bbabab \in \mathcal{M}$, and $u = bb \in \mathcal{U}$.

We now describe languages $\mathcal{T}_{\geq 1} = \bigcup_{r \geq 1} \mathcal{T}_r$ (set of words containing at least once occurrence of w) and $\mathcal{T}_r$ in terms of $\mathcal{R}$, $\mathcal{M}$, and $\mathcal{U}$. Recall that $\mathcal{M}^r$ denotes the concatenation of r languages $\mathcal{M}$, and $\mathcal{M}^0 = \{\varepsilon\}$. Also, $\mathcal{M}^+ = \cup_{r \geq 1} M^r$ and $\mathcal{M}^* = \cup_{r \geq 0} M^r$.

Theorem 7.2.3. *The languages $\mathcal{T}_r$ for $r \geq 1$ and $\mathcal{T}_{\geq 1}$ satisfy the relations*

$$\mathcal{T}_r = \mathcal{R} \cdot \mathcal{M}^{r-1} \cdot \mathcal{U} \ , \tag{7.2.7}$$

and therefore

$$\mathcal{T}_{\geq 1} = \mathcal{R} \cdot \mathcal{M}^* \cdot \mathcal{U}. \tag{7.2.8}$$

In addition, we have:

$$\mathcal{T}_0 \cdot w = \mathcal{R} \cdot \mathcal{S}. \tag{7.2.9}$$

Proof. To prove (7.2.7), we obtain our decomposition of $\mathcal{T}_r$ as follows: The first occurrence of w in a word belonging to $\mathcal{T}_r$ determines a prefix $p \in \mathcal{T}_r$ that is in $\mathcal{R}$. After concatenating a nonempty word v we create the second occurrence of w provided $v \in \mathcal{M}$. This process is repeated $r-1$ times. Finally, after the last w occurrence we add a suffix u that does not create a new occurrence of w, that is, wu is such that $u \in \mathcal{U}$. Clearly, a word belongs to $\mathcal{T}_{\geq 1}$ if for some $1 \leq r < \infty$ it is in $\mathcal{T}_r$.

The derivation of (7.2.9) is left to the reader as Exercise 7.2.1.

■

Example 7.2.4. Let $w = TAT$. The following string belongs to $\mathcal{T}_3$:

$$\overbrace{CCTAT}^{\mathcal{R}}\underbrace{AT}_{\mathcal{M}}\underbrace{GATAT}_{\mathcal{M}}\overbrace{GGA}^{\mathcal{U}}.$$

We now prove the following result that summarizes relationships between the languages $\mathcal{R}$, $\mathcal{M}$, and $\mathcal{U}$.

Theorem 7.2.5. *The languages $\mathcal{M}$, $\mathcal{R}$, and $\mathcal{U}$ satisfy*

$$\mathcal{M}^* = \mathcal{A}^* \cdot w + \mathcal{S}, \tag{7.2.10}$$

$$\mathcal{U} \cdot \mathcal{A} = \mathcal{M} + \mathcal{U} - \{\varepsilon\}, \tag{7.2.11}$$

$$w(\mathcal{M} - \varepsilon) = \mathcal{A} \cdot \mathcal{R} - \mathcal{R}. \tag{7.2.12}$$

Proof. We first deal with (7.2.10). Clearly, $\mathcal{A}^*w$ contains at least one occurrence of w on the right, hence $\mathcal{A}^*w \subset \mathcal{M}^*$. Furthermore, a word v in $\mathcal{M}^*$ is not in $\mathcal{A}^* \cdot w$ if and only if its size $|v|$ is smaller than $|w|$ (e.g., think of $v = ab \in \mathcal{M}$ for $w = abab$). Then the second w occurrence in wv overlaps with w, which means that v is in $\mathcal{S}$.

Let us turn now to (7.2.11). When one adds a character $a \in \mathcal{A}$ right after a word u from $\mathcal{U}$, two cases may occur. Either wua still does not contain a second occurrence of w (which means that ua is a nonempty word of $\mathcal{U}$) or a new w appears, clearly at the right end. Hence $\mathcal{U} \cdot \mathcal{A} \subseteq \mathcal{M} + \mathcal{U} - \varepsilon$. Let now $v \in \mathcal{M} - \varepsilon$, then by definition $wv \in \mathcal{T}_2 \subseteq \mathcal{U}\mathcal{A} - \mathcal{U}$ which proves (7.2.11).

We now prove (7.2.12). Let now $x = ar$ be a word in $w \cdot (\mathcal{M} - \varepsilon)$ where $a \in \mathcal{A}$. As x contains exactly two occurrences of w located at its left and right ends, r is in $\mathcal{R}$ and x is in $\mathcal{A} \cdot \mathcal{R} - \mathcal{R}$, hence $w(\mathcal{M} - \varepsilon) \subseteq \mathcal{A} \cdot \mathcal{R} - \mathcal{R}$. To prove $\mathcal{A} \cdot \mathcal{R} - \mathcal{R} \subseteq w(\mathcal{M} - \varepsilon)$, we take a word arw from $\mathcal{A} \cdot \mathcal{R}$ that is not in $\mathcal{R}$. Then arw contains a second w occurrence starting in ar. As rw is in $\mathcal{R}$, the only possible position is at the left end, and then x is in $w(\mathcal{M} - \varepsilon)$. This proves (7.2.12).

■

7.2.2. Generating functions

The next step is to translate the relationships between languages into the associated generating functions. Therefore, we must now select the probabilistic model according to which the text is generated. We derive our results for a Markov model of order one. We adopt the following notation: To extract a particular element, say with index (i, j), from a matrix, say P, we shall write $[\mathrm{P}]_{i,j} = p_{i,j}$. We also recall that $(\mathrm{I} - \mathrm{P})^{-1} = \sum_{k\geq 0} \mathrm{P}^k$ provided $||\mathrm{P}|| < 1$ for a matrix norm $||\cdot||$. We also write Π for the stationary matrix that consists of V identical rows equal to μ. Finally, by Z we denote the *fundamental matrix* $\mathrm{Z} = (\mathrm{I} - (\mathrm{P} - \Pi))^{-1}$ where I is the identity matrix.

The next lemma translates the relationships between languages (7.2.10)–(7.2.12) into generating functions $M_w(z)$, $U_w(z)$ and $R(z)$ of languages $\mathcal{M}$, $\mathcal{U}$ and $\mathcal{R}$ (we recall that the first two generating function are *conditioned* on w appearing just before any word from $\mathcal{M}$ and $\mathcal{U}$). We define a function $F(z)$ by

$$F(z) = \frac{1}{\mu_{w_1}}[\sum_{n\geq 0}(\mathrm{P} - \Pi)^{n+1}z^n]_{w_m,w_1} = \frac{1}{\mu_{w_1}}[(\mathrm{P} - \Pi)(\mathrm{I} - (\mathrm{P} - \Pi)z)^{-1}]_{w_m,w_1} \tag{7.2.13}$$

for $|z| <\| \mathrm{P} - \Pi \|^{-1}$, where μ_{w_1} is the stationary probability of the first symbol w_1 of w. For memoryless sources $F(z) = 0$.

LEMMA 7.2.6. *For Markov sources (of order one), the generating functions associated with languages $\mathcal{M}$,$\mathcal{U}$, and $\mathcal{R}$ satisfy*

$$\frac{1}{1 - M_w(z)} = S_w(z) + P(w)z^m\left(\frac{1}{1-z} + F(z)\right), \tag{7.2.14}$$

$$U_w(z) = \frac{M_w(z) - 1}{z - 1}, \tag{7.2.15}$$

$$R(z) = P(w)z^m \cdot U_w(z), \tag{7.2.16}$$

provided the underlying Markov chain is aperiodic and ergodic.

Proof. We first prove (7.2.15). Let us consider language relationship (7.2.11) from Theorem 7.2.5, which we rewrite as $\mathcal{U} \cdot \mathcal{A} - \mathcal{U} = \mathcal{M} - \varepsilon$. Observe that $\sum_{b\in\mathcal{A}} p_{ab}z = z$. Hence, set $\mathcal{U} \cdot \mathcal{A}$ yields (conditioning on the left occurrence of w)

$$\sum_{u\in\mathcal{U}}\sum_{b\in\mathcal{A}} P(ub|w)z^{|ub|} = \sum_{a\in\mathcal{A}}\sum_{u\in\mathcal{U},\ell(u)=a} P(u|w)z^{|u|}\sum_{b\in\mathcal{A}} p_{ab}z = U_w(z)\cdot z,$$

where $\ell(u)$ denotes the last symbol of the word u. Of course, $\mathcal{M} - \varepsilon$ and $\mathcal{U}$ translate into $M_w(z) - 1$ and $\mathcal{U}_w(z)$, and (7.2.15) is proved.

We now turn our attention to (7.2.16), and we use relationship (7.2.12) $w\mathcal{M} - w = \mathcal{A}\mathcal{R} - \mathcal{R}$ of Theorem 7.2.5. In order to compute the *conditional* generating function of $\mathcal{A} \cdot \mathcal{R}$ we proceed as follows

$$\sum_{ab\in\mathcal{A}^2}\sum_{bv\in\mathcal{R}} P(abv)z^{|abv|} = z^2\sum_{a\in\mathcal{A}}\sum_{b\in\mathcal{A}}\mu_a p_{ab}\sum_{bv\in\mathcal{R}} P(v|v_{-1} = b)z^{|v|}.$$

But due to the stationarity of the underlying Markov chain $\sum_a \mu_a p_{ab} = \mu_b$. As $\mu_b P(v|v_{-1} = b) = P(bv)$, we get $zR(z)$. Furthermore, $w \cdot \mathcal{M} - w$ translates into $P(w)z^m \cdot (M_w(z) - 1)$. By just proved (7.2.15), this is $P(w)z^m \cdot \mathcal{U}_w(z)(z-1)$, and after a simplification, we obtain (7.2.16).

Finally, we deal with (7.2.14), and prove it using (7.2.10) from Theorem 7.2.5. The left-hand side of (7.2.10) involves language $\mathcal{M}$, hence we must condition on the left occurrence of w. In particular, $\bigcup_{r\geq 1} \mathcal{M}^r + \varepsilon$ of (7.2.10) translates into $\frac{1}{1-\mathcal{M}_w(z)}$. Now we deal with $\mathcal{A}^* \cdot w$ of the right-hand side of (7.2.10). *Conditioning* on the left occurrence of w, the generating function $A_w(z)$ of $\mathcal{A}^* \cdot w$ is

$$\begin{aligned} A_w(z) &= \sum_{n\geq 0} \sum_{|u|=n} z^{n+m} P(uw|u_{-1} = w_m) \\ &= \sum_{n\geq 0} \sum_{|u|=n} z^n P(uw_1|u_{-1} = w_m) P(w_2 \dots w_m|w_1) z^m. \end{aligned}$$

We have $P(w_2 \dots w_m|w_1)z^m = \frac{1}{\mu_{w_1}} z^m P(w)$, and for $n \geq 0$:

$$\sum_{|u|=n} P(uw_1|u_{-1} = w_m) = [\mathrm{P}^{n+1}]_{w_m,w_1}$$

where, we recall, w_m is the last character of w. In summary, the language $\mathcal{A}^* \cdot w$ contributes $P(w)z^m \left[\frac{1}{\mu_{w_1}} \sum_{n\geq 0} \mathrm{P}^{n+1} z^n\right]_{w_m,w_1}$, while the language $\mathcal{S} - \{\varepsilon\}$ introduces $S_w(z) - 1$. Using the equality $\mathrm{P}^{n+1} - \Pi = (\mathrm{P} - \Pi)^{n+1}$ (which follows from a consecutive application of the identity $\Pi\mathrm{P} = \Pi$), and observing that for any symbols a and b

$$\left[\frac{1}{\mu_b} \sum_{n\geq 0} \Pi z^n\right]_{ab} = \sum_{n\geq 0} z^n = \frac{1}{1-z}.$$

we finally obtain the sum in (7.2.14). This completes the proof of the theorem. ■

The lemma above together with Theorem 7.2.3 suffice to derive generating functions $N_r(z)$ and $N(z,u)$ in an explicit form.

THEOREM 7.2.7. *Let w be a given pattern of size m, and X be a random text of length n generated according to an ergodic and aperiodic Markov chain with the transition probability matrix* P. *Define*

$$D_w(z) = (1-z)S_w(z) + z^m P(w)(1 + (1-z)F(z)). \tag{7.2.17}$$

Then

$$N_0(z) = \frac{1-R(z)}{1-z} = \frac{S_w(z)}{D_w(z)}, \tag{7.2.18}$$

$$N_r(z) = R(z) M_w^{r-1}(z) U_w(z)\ , \quad r \geq 1, \tag{7.2.19}$$

$$N(z,u) = R(z) \frac{u}{1-uM_w(z)} U_w(z), \tag{7.2.20}$$

where

$$M_w(z) = 1 + \frac{z-1}{D_w(z)} \;, \tag{7.2.21}$$

$$U_w(z) = \frac{1}{D_w(z)}, \tag{7.2.22}$$

$$R(z) = z^m P(w) \frac{1}{D_w(z)}. \tag{7.2.23}$$

We recall that for memoryless sources, $F(z) = 0$, and hence

$$D(z) = (1-z)S(z) + z^m P(w). \tag{7.2.24}$$

Proof. We only comment on the derivation of $N_0(z)$ since the rest follows directly from our previous results. Observe that

$$N_0(z) = \sum_{n \ge 0} \mathbf{P}(N_n = 0) z^n = \sum_{n \ge 0} (1 - \mathbf{P}(N_n > 0)) z^n = \frac{1}{1-z} - \sum_{r=1}^{\infty} N_r(z),$$

thus the first expression follows form (7.2.19). The second expression is a direct translation of $\mathcal{T}_0 \cdot w = \mathcal{R} \cdot \mathcal{A}$ (cf. (7.2.9)) which reads $N_0(z)P(w)z^m = R(z)S_w(z)$ in terms of the appropriate generating functions. ∎

7.2.3. Moments and limit laws

In the previous section we derived an explicit formula for the generating function $N(z,u) = \sum_{n \ge 0} \mathbf{E}(u^{N_n}) z^n$ and $N_r(z)$. These formulas can be used to obtain explicit and asymptotic expressions for moments of N_n (cf. Theorem 7.2.8), the central limit theorem (cf. Theorem 7.2.11), and large deviations (cf. Theorem 7.2.12). We start with derivation of the mean and the variance of N_n.

Theorem 7.2.8. *Under the assumptions of Theorem 7.2.7 and $nP(w) \to \infty$, one has, for $n \ge m$:*

$$\mathrm{E}[N_n(w)] = P(w)(n - m + 1) \;, \tag{7.2.25}$$

and

$$\mathrm{Var}[N_n(w)] = nc_1 + c_2 + O(R^{-n}), \;\; \textit{for } R > 1 \tag{7.2.26}$$

where

$$c_1 = P(w)(2S_w(1) - 1 - (2m-1)P(w) + 2P(w)E_1)), \tag{7.2.27}$$

$$\begin{aligned} c_2 &= P(w)((m-1)(3m-1)P(w) - (m-1)(2S_w(1) - 1) - 2\mathcal{S}'_w(1)) \\ &\quad - 2(2m-1)P(w)^2 E_1 + 2E_2 P(w)^2, \end{aligned} \tag{7.2.28}$$

and the constants E_1, E_2 are

$$E_1 = \frac{1}{\mu_{w_1}} [(\mathrm{P} - \Pi)\mathrm{Z}]_{w_m, w_1}, \quad E_2 = \frac{1}{\mu_{w_1}} [(\mathrm{P}^2 - \Pi)\mathrm{Z}^2]_{w_m, w_1},$$

Proof. Notice that first moment estimate can be derived directly from the definition of the stationary probability of w. In order to grasp higher moments we will use analytic tools applied to generating functions. We compute the first two moments of N_n from $N(z,u)$ since $\mathbf{E}(N_n) = [z^n]N_u(z,1)$ and $\mathbf{E}(N_n(N_n - 1)) = [z^n]N_{uu}(z,1)$ where $N_u(z,1)$ and $N_{uu}(z,1)$ are the first and the second derivatives of $N(z,u)$ with respect to variable u at $(z,1)$. By Theorem 7.2.7 we find

$$\begin{aligned} N_u(z,1) &= \frac{z^m P(w)}{(1-z)^2}, \\ N_{uu}(z,1) &= \frac{2z^m P(w) M_w(z) D_w(z)}{(1-z)^3}. \end{aligned}$$

Now we observe that both expressions admit as a numerator a function that is analytic beyond the unit circle. Furthermore, for a positive integer $k > 0$

$$[z^n](1-z)^{-k} = \binom{n+k-1}{k-1} = \frac{\Gamma(n+k)}{\Gamma(k)\Gamma(n+1)}, \tag{7.2.29}$$

(where $\Gamma(x)$ is the Euler gamma function), we find for $n \geq m$

$$\mathbf{E}(N_n) = [z^n]N_u(z,1) = P(w)[z^{n-m}](1-z)^{-2} = (n-m+1)P(w).$$

In order to estimate variance, we introduce

$$\Phi(z) = 2z^m P(w) M_w(z) D_w(z),$$

and observe that

$$\Phi(z) = \Phi(1) + (z-1)\Phi'(1) + \frac{(z-1)^2}{2}\Phi''(1) + (z-1)^3 f(z),$$

where $f(z)$ is the remainder of the Taylor expansion of $\Phi(z)$ up to order 3 at $z = 1$. For *memoryless sources*, $\Phi(z)$ and thus $f(z)$ are polynomials of degree $2m-2$ and $[z^n](z-1)f(z)$ is 0 for $n \geq 2m-1$. Hence, by (7.2.29) we arrive at

$$\mathbf{E}(N_n(N_n-1)) = [z^n]N_{uu}(z,1) = \Phi(1)\frac{(n+2)(n+1)}{2} - \Phi'(1)(n+1) + \frac{1}{2}\Phi''(1).$$

But $\mathcal{M}_w(z)D_w(z) = D_w(z) + (1-z)$ and taking into account formula (7.2.24) for $D(z)$, we finally obtain (7.2.26).

For *Markov sources*, $D_w(z)$ has an additional term, namely

$$[z^n]\frac{2(z^{2m}P(w)^2F(z))}{(1-z)^2},$$

where $F(z)$, defined in (7.2.13), is analytic beyond the unit circle for $|z| \leq R$, with $R > 1$. The Taylor expansion of $F(z)$ is $E_1 + (1-z)E_2$, and applying (7.2.29) again yields the result. ∎

Recall that $\mathrm{P} = \Pi$ for memoryless sources, so $E_1 = E_2 = 0$ and (7.2.26) reduces to an equality for $n \geq 2m-1$. Thus

$$\mathrm{Var}[N_n(w)] = nc_1 + c_2 \tag{7.2.30}$$

with

$$\begin{aligned}
c_1 &= P(w)(2S(1) - 1 - (2m-1)P(w)),\\
c_2 &= P(w)((m-1)(3m-1)P(w) - (m-1)(2S(1)-1) - 2S'(1)).
\end{aligned}$$

In passing we should notice that from the generating function $N(z,u)$ we can compute all moments of N_n. Instead, however, we present some limit laws for $\mathbf{P}(N_n = r)$ for different values of r: We consider $r = O(1)$, $r = \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)}$ (central and local limit regime), and $r = (1+\delta)\mathbf{E}(N_n)$ (large deviations). From the central limit theorem (cf. Theorem 7.2.11 below) we conclude that the normalized random variable $(N_n - \mathbf{E}(N_n))/\sqrt{\mathrm{Var}(N_n)}$ converges also in moments to the moments of the standard normal distribution. This follows from the fact that in the theorem below we prove the convergence of the normalized generating function to an analytic function, namely $e^{u^2/2}$ for u complex in the vicinity of zero. Since an analytic function has well defined derivatives, convergence in moments follows. We shall leave a formal proof to the reader (cf. Exercise 7.2.3).

THEOREM 7.2.9. *Under the assumptions of Theorem 7.2.8, let ρ_w be the root of $D_w(z) = 0$ of the smallest modulus and multiplicity one. Then, ρ_w is real such that $\rho_w > 1$, and there exists $\rho > \rho_w$ such that for $r = O(1)$*

$$\mathbf{P}(N_n(w) = r) = \sum_{j=1}^{r+1} (-1)^j a_j \binom{n}{j-1} \rho_w^{-(n+j)} + O(\rho^{-n}) , \tag{7.2.31}$$

where

$$a_{r+1} = \frac{\rho_w^m P(w) \left(\rho_w - 1\right)^{r-1}}{\left(D_w'(\rho_w)\right)^{r+1}} , \tag{7.2.32}$$

and the remaining coefficients can be computed according to

$$a_j = \frac{1}{(r+1-j)!} \lim_{z \to \rho_w} \frac{d^{r+1-j}}{dz^{r+1-j}} \left(N_r(z)(z-\rho_w)^{r+1}\right) \tag{7.2.33}$$

with $j = 1, 2, \ldots, r$.

In order to prove Theorem 7.2.9, we need the following simple result.

LEMMA 7.2.10. *The equation $D_w(z) = 0$ has at least one root, and all its roots are of modulus greater than 1.*

Proof. Poles of $D_w(z) = (1-z)/(1-M_w(z))$ are clearly poles of $\frac{1}{1-M_w(z)}$. As $\frac{1}{1-M_w(z)}$ is the generating function of a language, it converges for $|z| < 1$ and has no pole of modulus smaller than 1. Since $D_w(1) \neq 0$, then $z = 1$ is a simple pole of $1/(1 - M_w(z))$. As all its coefficients are real and non negative, there is no other pole of modulus $|z| = 1$. It follows that all roots of $D_w(z)$ are of modulus greater than 1. The existence of a root is guaranteed since $D_w(z)$ is either a polynomial (Bernoulli model) or a ratio of polynomials (Markov model). ■

Proof of Theorem 7.2.9. We first re-write the formula on $N_r(z)$ as follows

$$N_r(z) = \frac{z^m P(w)(D_w(z) + z - 1)^{r-1}}{D_w^{r+1}(z)} . \tag{7.2.34}$$

Observe that $\mathbf{P}(N_n(w) = r)$ is the coefficient at z^n of $N_r(z)$. By Hadamard's theorem, asymptotics of the coefficients of a generating function depend on the singularities of the underlying generating function. In our case, the generating function $N_r(z)$ is a rational function, thus we can only expect poles (for which the denominator $D_w(z)$ vanishes). Lemma 7.2.10 above establishes the existence and properties of such a pole. Therefore, the generating function $N_r(z)$ can be expanded around its root of smallest modulus, let ρ_w be this smallest modulus, in Laurent's series:

$$N_r(z) = \sum_{j=1}^{r+1} \frac{a_j}{(z-\rho_w)^j} + \widetilde{N}_r(z) \tag{7.2.35}$$

where $\widetilde{N}_r(z)$ is analytical in $|z| < \rho'$ and ρ' is defined as $\rho' = \inf\{|\rho| : \ \rho > \rho_w \text{ and } D_w(\rho) = 0\}$. The constants a_j satisfy (7.2.33). This formula simplifies into (7.2.32) for the leading constant a_{-r-1}. As a consequence of analyticity we have for $1 < \rho_w < \rho < \rho'$: $[z^n]\widetilde{N}^{(r)}(z) = O(\rho^{-n})$. Hence, the term $\widetilde{N}_r(z)$ contributes only to the lower terms in the asymptotic expansion of $N_r(z)$. After some algebra, and noting that $[z^n]1/(1-z)^{k+1} = \binom{n+k}{n}$, we prove Theorem 7.2.9. ■

In the next theorem we establish the central limit theorem in its strong form (i.e., local limit theorem).

Theorem 7.2.11. *Under the same assumption as in Theorem 7.2.8 we have*

$$\mathbf{P}(N_n(w) \le \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)}) = \left(1 + O\left(\frac{1}{\sqrt{n}}\right)\right) \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt. \tag{7.2.36}$$

If, in addition, $p_{ij} > 0$ for all $i, j \in \mathcal{A}$, then for any bounded real interval B

$$\sup_{x \in B} \left| \mathbf{P}(N_n(w) = \lfloor \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)} \rfloor) - \frac{1}{\sqrt{2\pi \mathrm{Var}(N_n)}} e^{-\frac{1}{2}x^2} \right| = o\left(\frac{1}{\sqrt{n}}\right) \tag{7.2.37}$$

as $n \to \infty$.

Proof. Let $r = \lfloor \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)} \rfloor$ with $x = O(1)$. We compute $\mathbf{P}(N_n(w) \le r)$ (central limit theorem) and $\mathbf{P}(N_n(w) = r)$ (local limit theorem) for $r = \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)}$ when $x = O(1)$. Let $\nu_n = \mathbf{E}(N_n(w)) = (n-m+1)P(w)$ and $\sigma_n^2 = \mathrm{Var}(N_n(w)) = c_1 n + O(1)$. To establish normality of $(N_n(w) - \nu_n)/\sigma_n$, it suffices, according to Lévy's continuity theorem, to prove the following

$$\lim_{n\to\infty} e^{-\tau\nu_n/\sigma_n} N_n(e^{\tau/\sigma_n}) = e^{\tau^2/2} \tag{7.2.38}$$

for complex τ (actually, $\tau = iv$ suffices). Again, by Cauchy's theorem

$$N_n(u) = \frac{1}{2\pi i}\oint \frac{N(z,u)}{z^{n+1}}dz = \frac{1}{2\pi i}\oint \frac{uP(w)}{D_w^2(z)(1-uM_w(z))z^{n+1-m}}dz \ ,$$

where the integration is along a circle around the origin. The evaluation of this integral is standard and it appeals to the Cauchy residue theorem. Namely, we enlarge the circle of integration to a bigger one, say $R > 1$, such that the bigger circle contains the dominant pole of the integrand function. Observe that the Cauchy integral over the bigger circle is $O(R^{-n})$. Let us now substitute $u = e^t$ and $z = e^\rho$. Then, the poles of the integrand are the roots of the equation

$$1 - e^t M_w(e^\rho) = 0. \tag{7.2.39}$$

This equation implicitly defines in some neighborhood of $t = 0$ a unique C^∞ function $\rho(t)$, satisfying $\rho(0) = 0$. Notably, all other roots ρ satisfy $\inf|\rho| = \rho' > 0$. Then, the residue theorem with $e^{\rho'} > R > e^\rho > 1$ leads to

$$N_n(e^t) = C(t)e^{-(n+1-m)\rho(t)} + O(R^{-n}) \tag{7.2.40}$$

where

$$C(t) = \frac{P(w)}{D_w^2(\rho(t))M_w'(\rho(t))}.$$

To study properties of $\rho(t)$, we observe that the cumulant formula implies $\mathbf{E}(N_n(w)) = [t]\log N_n(e^t)$ and $\sigma_n^2 = [t^2]\log N_n(e^t)$ where, we recall, $[t^r]f(t)$ denotes the coefficient of $f(t)$ at t^r. In our case, $\nu_n \sim -n\rho'(0)$ as well as $\sigma_n^2 \sim -n\rho''(0)$. Set now in (7.2.40) $t = \tau/\sigma_n \to 0$ for some complex τ. Since uniformly in t we have $\rho(t) = t\rho'(0) + \rho''(0)t^2/2 + O(t^3)$ for $t \to 0$, our estimate (7.2.40) leads to

$$\begin{aligned} e^{-\tau\nu_n/\sigma_n} N_n(e^{\tau/\sigma_n}) &= \exp\left(\frac{\tau^2}{2} + O(n\tau^3/\sigma_n^3)\right) \\ &= e^{\tau^2/2}\left(1 + O(1/\sqrt{n})\right), \end{aligned}$$

which proves (7.2.36) after applying the Berry-Essen inequality that allows to derive the error term $O(1/\sqrt{n})$ for the probability distribution.

To establish the local limit theorem, we observe that if $p_{ij} > 0$ for all $i, j \in \mathcal{A}$, then $\rho(t) > 0$ for $t \neq 0$ (cf. Exercise 7.2.4). We can obtain much more refined local limit result. Indeed, we find for $x = o(n^{1/6})$

$$\mathbf{P}(N_n = \mathbf{E}(N_n) + x\sqrt{nc_1}) = \frac{1}{\sqrt{2\pi nc_1}} e^{-\frac{1}{2}x^2} \left(1 - \frac{\kappa_3}{2c_1^{3/2}\sqrt{n}} \left(x - \frac{x^3}{3}\right)\right) + O(n^{-3/2}) \ , \tag{7.2.41}$$

where κ_3 a constant (i.e., the third cumulant). This completes the proof of Theorem 7.2.11. ■

Finally, we establish precise large deviations for N_n. Large deviations play a central role in many applications, most notably in data mining and molecular biology, since it allows to establish a threshold for overrepresented and underrepresented patterns.

Theorem 7.2.12. *Let $r = a\mathrm{E}[N_n]$ with $a = (1+\delta)P(w)$ for $\delta \neq 0$. For complex t, define $\rho(t)$ to be the root of*

$$1 - e^t M_w(e^\rho) = 0 \ , \tag{7.2.42}$$

and define ω_a and σ_a by

$$-\rho'(\omega_a) = a, \quad -\rho''(\omega_a) = \sigma_a^2.$$

Then

$$\mathbf{P}(N_n(w) = (1+\delta)\mathbf{E}(N_n)) \sim \frac{1}{\sigma_a\sqrt{2\pi(n-m+1)}} e^{-(n-m+1)I(a)+\theta_a} \tag{7.2.43}$$

where $I(a) = a\omega_a + \rho(\omega_a)$ and

$$\theta_a = \log \frac{P(w)e^{m\rho(\omega_a)}}{D_w(e^{\rho(\omega_a)}) + (1-e^{\rho(\omega_a)})D'_w(e^{\rho(\omega_a)})}, \tag{7.2.44}$$

and $D_w(z)$ is defined in (7.2.17).

Proof. From (7.2.40) we conclude that

$$\lim_{n\to\infty} \frac{\log N_n(e^t)}{n} = -\rho(t) \ .$$

By the Gärtner-Ellis theorem we find

$$\lim_{n\to\infty} \frac{\log \mathbf{P}(N_n > na)}{n} = -I(a) \ ,$$

where

$$I(a) = a\omega_a + \rho(\omega_a)$$

with ω_a being a solution of $-\rho'(t) = a$, A stronger version of the above result is possible and we derive it in the sequel. In fact, we use (7.2.41) and the "shift of mean" technique.

As in the local limit regime, we could use Cauchy's formula to compute the probability $\mathbf{P}(N_n = r)$ for $r = \mathbf{E}(N_n) + xO(\sqrt{n})$. But, formula (7.2.41) is only good for $x = O(1)$ while we need $x = O(\sqrt{n})$ for the large deviations. To expand its validity, we shift the mean of the generating function $N_n(u)$ to a new value, say $m = an = (1+\delta)P(w)(n-m+1)$, so we can again apply the central limit formula (7.2.41) around the new mean. To accomplish this, let us re-write (7.2.40) as for any $R > 0$

$$N_n(e^t) = C(t)[g(t)]^{n-m+1} + O(R^{-n})$$

where $g(t) = e^{-\rho(t)}$. (In the derivation below, for simplicity we dropped $O(R^{-n})$ term.) The above suggests that $N_n(e^t)$ is the moment generating function of a sum S_n of $n-m+1$ "almost" independent random variables $X_1, \ldots, X_{n-m+1}$ having moment generating function equal to $g(t)$ and Y whose moment generating function is $C(t)$. Observe that $\mathbf{E}(S_n) = (n-m+1)P(w)$ while we need to estimate the tail of S_n around $(1+\delta)(n-m+1)P(w)$. To achieve it, we introduce a new random variable $\widetilde{X}_i$ whose moment generating function $\widetilde{g}(t)$ is

$$\widetilde{g}(t) = \frac{g(t+\omega)}{g(\omega)}$$

where ω will be chosen later. Then, the mean and the variance of the new variable $\widetilde{X}$ is

$$\mathbf{E}(\widetilde{X}) = \frac{g'(\omega)}{g(\omega)} = -\rho'(\omega) \; ,$$

$$\mathrm{Var}(\widetilde{X}) = \frac{g''(\omega)}{g(\omega)} - \left(\frac{g'(\omega)}{g(\omega)}\right)^2 = -\rho''(\omega) \; .$$

Let us now choose ω_a such that

$$-\rho'(\omega_a) = \frac{g'(\omega_a)}{g(\omega_a)} = a = P(w)(1+\delta) \; .$$

Then, the new sum $\widetilde{S}_n - Y = \widetilde{X}_1 + \ldots + \widetilde{X}_{n-m+1}$ has a new mean $(1+\delta)P(w)(n-m+1) = a(n-m+1)$, and hence we can apply to $\widetilde{S}_n - Y$ the central limit result (7.2.41). To translate from $\widetilde{S_n} - Y$ to S_n we use the following simple formula

$$[e^{tM}]\,(g^n(t)) = \frac{g^n(\omega)}{e^{\omega M}}[e^{tM}]\left(\frac{g^M(t+\omega)}{g^M(\omega)}\right) \tag{7.2.45}$$

where $M = a(n-m+1)$ and $[e^{tn}]g(t)$ denotes the coefficient of $g(t)$ at $z^n = e^{tn}$ (where $z = e^t$). Now, we can apply (7.2.41) to the right-hand side of the above to obtain

$$[e^{tM}]\left(\frac{g^M(t+\omega)}{g^M(\omega)}\right) \sim \frac{1}{\sigma_a\sqrt{2\pi(n-m+1)}}.$$

To obtain the final result we must take into account the effect of Y whose moment generating function is $C(t)$. This leads to replacing $a = 1+\delta$ by $a =$

Table 7.1. Z score vs p-value of tandem repeats in *A.thaliana*.

Oligomer	Obs.	p-val (large dev.)	Z-sc.
AATTGGCGG	2	8.059×10^{-4}	48.71
TTTGTACCA	3	4.350×10^{-5}	22.96
ACGGTTCAC	3	2.265×10^{-6}	55.49
AAGACGGTT	3	2.186×10^{-6}	48.95
ACGACGCTT	4	1.604×10^{-9}	74.01
ACGCTTGG	4	5.374×10^{-10}	84.93
GAGAAGACG	5	0.687×10^{-14}	151.10

$1+\delta+C'(0)/n$ resulting the the correction term $e^{\theta_a} = e^{C'(0)\omega_a}$. Theorem 7.2.12 is proved. ■

We illustrate the above results on an example taken from molecular biology.

Example 7.2.13. Biologists apply the so called Z-score and p-value to determine whether biological sequences such as DNA or protein contain a biological signal, that is, an underrepresented or overrepresented patterns. These quantities are defined as

$$Z(w) = \frac{\mathbf{E}(N_n) - N_n(w)}{\sqrt{\mathrm{Var}(N_n(w))}},$$
$$pval(r) = P(N_n(w) > r).$$

Z-score indicates by how many standard deviations the observed value $N_n(w)$ is away from the mean. Clearly, this score makes sense only if one can prove, as we did in Theorem 7.2.11, that Z satisfies (at least asymptotically) the Central Limit Theorem (CLT). On the other hand, p-value is used for rare occurrences, far away from the mean where one needs to apply the large deviations as in Theorem 7.2.12.

The range of validity of Z-score and p-value are important as illustrated in Table 7.2.13 where results for 2008 nucleotides long fragments of *A.thaliana* (a plant genome) are presented. In the table for each 9-mer the number of observations is presented in the first column following by the large deviations probability computed from Theorem 7.2.12 and Z-score. We observe that for *AATTGGCGG* and *AAGACGGTT* the Z-scores are about 48 while p-values differ by two order of magnitudes. In fact, occurrences of these 9-mers are very rare, and therefore Z-score is not an adequate measure.

7.2.4. Waiting times

We shall now discuss the waiting times T_w and T_j, where $T_w = T_1$ is the first time w occurs in the text, while T_j is the minimum length of the text in which

w occurs j times. Fortunately, we do not need re-derive generating function of T_j since, as we have already indicated in (7.2.3), the following *duality principle* holds

$$\{N_n \geq j\} = \{T_j \leq n\},$$

and in particular, $\{T_w > n\} = \{N_n = 0\}$. Therefore, if

$$T(u,z) = \sum_{n\geq 0}\sum_{j\geq 0} \mathbf{P}(T_j = n) z^n u^j,$$

then by the duality principle we have

$$(1-u)T(u,z) + u(1-z)N(z,u) = 1,$$

and one obtains $T(u,z)$ from Theorem 7.2.7. Waiting times were analyzed in depth in Chapter 6.

Finally, observe that the above duality principle implies

$$\mathbf{E}(T_w) = \sum_{n\geq 0} \mathbf{P}(N_n = 0) = N_0(1).$$

In particular, for *memoryless sources*, from Theorem 7.2.7 we conclude that

$$N_0(z) = \frac{z^m S(z)}{(1-z)S(z) + z^m P(w)}.$$

Hence

$$\begin{aligned}\mathbf{E}(T_w) &= \sum_{n\geq 0} \mathbf{P}(N_n(w) = 0) = N_0(1) = \frac{S(1)}{P(w)} \\ &= \sum_{k\in\mathcal{P}(w)} \frac{1}{P(w_1^k)} = \frac{1}{P(w)} + \sum_{k\in\mathcal{P}(w)-\{m\}} \frac{1}{P(w_1^k)} \qquad (7.2.46)\end{aligned}$$

7.3. Generalized string matching

In this section we consider generalized pattern matching in which a set of patterns (rather than a single pattern) is given. We assume that the pattern is a pair of sets of words $(\mathcal{W}_0, \mathcal{W})$ where $\mathcal{W} = \bigcup_{i=1}^d \mathcal{W}_i$ consists of sets $\mathcal{W}_i \subset \mathcal{A}^{m_i}$ (i.e., all words in $\mathcal{W}_i$ have a fixed length equal to m_i). The set $\mathcal{W}_0$ is called the *forbidden set*. For $\mathcal{W}_0 = \emptyset$ one is interested in the number of pattern occurrences, $N_n(\mathcal{W})$, defined as the *number of patterns* from $\mathcal{W}$ occurring in the text X_1^n generated by a (random) source. Another parameter of interest may be the *number of positions* in X_1^n where *a pattern* from $\mathcal{W}$ appears (clearly, some patterns may occur more than once at some positions). The latter quantity we denote as Π_n. If we define $\Pi_n^{(i)}$ as the number of positions where a word from $\mathcal{W}_i$ occurs, then

$$N_n(\mathcal{W}) = \Pi_n^{(1)} + \cdots + \Pi_n^{(d)}.$$

Notice that at any given position of the text and for a given i only one word from $\mathcal{W}_i$ can occur.

For $\mathcal{W}_0 \neq \emptyset$ one studies the number of occurrences $N_n(\mathcal{W})$ under the condition that $N_n(\mathcal{W}_0) := \Pi_n^{(0)} = 0$, that is, there is no occurrence of a pattern from $\mathcal{W}_0$ in the text X_1^n. This could be called a *restricted* pattern matching since one restricts the text to those strings that do not contain strings from $\mathcal{W}_0$.

Finally, we may set $\mathcal{W}_i = \emptyset$ for $i = 1, \dots, d$ with $\mathcal{W}_0 \neq \emptyset$ and count the number of text strings that do not contain any pattern from $\mathcal{W}_0$. (Alternatively, we can estimate the probability that a randomly selected text X_1^n does not contain any pattern from $\mathcal{W}_0$.) In particular, define for $\ell \leq k$

$$\mathcal{W}_0 = \{\underbrace{0\dots0}_{\ell}, \dots, \underbrace{0\dots0}_{k}\}, \tag{7.3.1}$$

that is, $\mathcal{W}_0$ consists of runs of zeros of length at least ℓ and at most k. A text satisfying the property that no pattern from $\mathcal{W}_0$ defined in (7.3.1) occurs in it is called a (ℓ, k) sequence. Such sequences are used for magnetic coding.

In this section, we first present an analysis of the generalized pattern matching with $\mathcal{W}_0 = \emptyset$ and $d = 1$ that we call the *reduced pattern set* (i.e., no pattern is a substring of another pattern) followed by a detailed analysis of the generalized pattern matching. We describe two methods of analysis. First, we generalize our language approach from the previous section, and then for the general pattern matching case we apply de Bruijn's automaton and spectral analysis of matrices. Finally, we enumerate (ℓ, k) sequences and compute the so called Shannon capacity for such sequences.

Throughout this section we assume that the text is generated by a (non-degenerate) memoryless source (B), as defined in Section 7.1.

7.3.1. String matching over reduced set of patterns

We analyze here a special case of the generalized pattern matching with $\mathcal{W}_0 = \emptyset$ and $d = 1$. In this case we shall write $\mathcal{W}_1 := \mathcal{W} = \{w_1, \dots, w_K\}$ where w_i $(1 \leq i \leq K)$ are given patterns with fixed length $|w_i| = m$. We shall generalize the results from the exact pattern matching section, but we omit most of the proofs or move them to exercises.

As before, let $\mathcal{T}_{\geq 1}$ be a language of words containing at least one occurrence from the set $\mathcal{W}$, and for any nonnegative integer r, let $\mathcal{T}_r$ be the language of words containing exactly r occurrences from $\mathcal{W}$. In order to characterize $\mathcal{T}_r$ we introduce some additional languages for any $1 \leq i, j \leq K$:

- $\mathcal{M}_{ij} = \{v : \ w_i v \in \mathcal{T}_2$ *and* w_j occurs at the right end of $v\}$;
- $\mathcal{R}_i$ defined as the set of words containing only one occurrence of w_i, located at the right end;
- $\mathcal{U}_i = \{u : \ w_i u \in \mathcal{T}_1\}$, that is, a set of words u such that the only occurrence of $w_i \in \mathcal{W}$ in $w_i u$ is on the left.

We also need to generalize the autocorrelation set and the autocorrelation polynomial to a set of patterns. For any given two strings w and u, let

$$\mathcal{S}_{w,u} = \{u_{k+1}^m : w_{m-k+1}^m = u_1^k\}$$

be the *correlation set*. The set of positions k satisfying $u_1^k = w_{m-k+1}^m$ is denoted as $\mathcal{P}(w,u)$. If $w = x \cdot v$ and $u = v \cdot y$ for some words x, y, v, then $y \in \mathcal{S}_{w,u}$ and $|v| \in \mathcal{P}(w,u)$. The correlation polynomial, $S_{w,u}(z)$, of w and u is the associated generating function of $\mathcal{S}_{w,u}$, that is,

$$S_{w,u}(z) = \sum_{k \in \mathcal{P}(w,u)} P(u_{k+1}^m) z^{m-k}.$$

In particular, for $w_i, w_j \in \mathcal{W}$ we define $\mathcal{S}_{i,j} := \mathcal{S}_{w_i,w_j}$. The *correlation matrix* of $\mathcal{W}$ is denoted as $\mathrm{S}(z) = \{S_{w_i w_j}(z)\}_{i,j=1,K}$.

Example 7.3.1. Consider a DNA sequence over the alphabet $\mathcal{A} = \{A, C, G, T\}$ generated by a memoryless source with $P(A) = \frac{1}{5}$, $P(C) = \frac{3}{10}$, $P(G) = \frac{3}{10}$ and $P(T) = \frac{1}{5}$. Let $w_1 = ATT$ and $w_2 = TAT$. Then the correlation matrix $\mathrm{S}(z)$ is

$$\mathrm{S}(z) = \begin{pmatrix} 1 & 1 + \frac{z^2}{25} \\ 1 + \frac{z}{5} & 1 + \frac{z^2}{25} \end{pmatrix}.$$

In order to analyze the number of occurrences $N_n(\mathcal{W})$ and its generating functions we first generalize the language relationships discussed in Theorem 7.2.3. Observe that

$$\mathcal{T}_r = \sum_{1 \le i,j \le K} \mathcal{R}_i \mathcal{M}_{ij}^{r-1} \mathcal{U}_j,$$

$$\mathcal{T}_{\ge 1} = \sum_{r \ge 1} \sum_{1 \le i,j \le K} \mathcal{R}_i \mathcal{M}_{ij}^{r-1} \mathcal{U}_j,$$

where $\sum$ denotes disjoint union of sets. As in Theorem 7.2.5, one finds the following relationships between just introduced languages

$$\bigcup_{k \ge 1} \mathcal{M}_{i,j}^k = \mathcal{A}^* \cdot w_j + \mathcal{S}_{ij} - \varepsilon \qquad 1 \le i,j \le K,$$

$$\mathcal{U}_i \cdot \mathcal{A} = \bigcup_j \mathcal{M}_{ij} + \mathcal{U}_i - \varepsilon, \qquad 1 \le i \le K,$$

$$\mathcal{A} \cdot \mathcal{R}_j - (\mathcal{R}_j - w_j) = \bigcup_i w_i \mathcal{M}_{ij}, \qquad 1 \le j \le K,$$

$$\mathcal{T}_0 \cdot w_j = \mathcal{R}_j + \mathcal{R}_i(\mathcal{S}_{ij} - \varepsilon), \qquad 1 \le i,j \le K.$$

Let us now analyze $N_n(\mathcal{W})$ in a probabilistic framework. To simplify our presentation, we assume that the text is generated by a memoryless source. Then the above language relationships translate directly into generating functions, as discussed the last section.

Before we proceed, we adopt the following notations. Lower-case letters are reserved for vectors which are assumed to be column vectors (e.g., $\mathrm{x}^t = (x_1, \ldots, x_K)$) except for vectors of generating functions which we denote by uppercase letters (e.g., $\mathrm{U}^t(z) = (U_1(z), \ldots, U_K(z))$ where $U_i(z)$ is the generating function of a language $\mathcal{U}_{w_i}$). In the above the upper index "t" denotes transpose. We shall use upper-case letters for matrices (e.g., $\mathrm{S}(z) = \{S_{w_i w_j}(z)\}_{i,j=1,K}$). In particular, we write I for the identity matrix, and $\vec{1}^t = (1, \ldots, 1)$ for the vector of all ones.

Now we are ready to present exact formulas for the generating function $N_r(z) = \sum_{n\geq 0} \mathbf{P}(N_n(\mathcal{W}) = r)z^n$ and $N(z,u) = \sum_{k\geq 0} N_r(z)u^r$. The following theorem is a direct consequences of our definitions and language relationships.

Theorem 7.3.2. *Let $\mathcal{W} = \{w_1, \ldots, w_K\}$ be a given set of reduced patterns each of length m, and X be a random text of length n generated by a memoryless source. The generating functions $N_r(z)$ and $N(z,u)$ can be computed as follows:*

$$N_r(z) = \mathrm{R}^t(z)\mathrm{M}^{r-1}(z)\mathrm{U}(z) \tag{7.3.2}$$

$$N(z,u) = \mathrm{R}^t(z)u(\mathrm{I} - u\mathrm{M}(z))^{-1}\mathrm{U}(z)\ , \tag{7.3.3}$$

where, denoting $\mathrm{w}^t = (P(w_1), \ldots, P(w_K))$ and $\vec{1}^t = (1, 1, \ldots, 1)$, we have

$$\mathrm{M}(z) = (\mathrm{D}(z) + (z-1)\mathrm{I})\mathrm{D}(z)^{-1}, \tag{7.3.4}$$

$$(\mathrm{I} - \mathrm{M}(z))^{-1} = \mathrm{S}(z) + \frac{z^m}{1-z}\vec{1} \cdot \mathrm{w}^t, \tag{7.3.5}$$

$$\mathrm{U}(z) = \frac{1}{1-z}(\mathrm{I} - \mathrm{M}(z)) \cdot \vec{1}, \tag{7.3.6}$$

$$\mathrm{R}^t(z) = \frac{z^m}{1-z}\mathrm{w}^t \cdot (\mathrm{I} - \mathrm{M}(z)), \tag{7.3.7}$$

and

$$\mathrm{D}(z) = (1-z)\mathrm{S}(z) + z^m\vec{1} \cdot \mathrm{w}^t.$$

Using these results and following footsteps of our analysis for the exact pattern matching, we arrive at the following asymptotic results.

Theorem 7.3.3. *Let the text X be generated by a memoryless source with $P(w_i) > 0$ for $i = 1, \ldots, K$ and $P(\mathcal{W}) = \sum_{w_i\in\mathcal{W}} P(w_i) = \mathrm{w}^t \cdot \vec{1}$.*

(i) *The following holds*

$$\mathbf{E}(N_n(\mathcal{W})) = (n-m+1)P(\mathcal{W}),$$

$$\begin{aligned}\mathrm{Var}(N_n(\mathcal{W})) &= (n-m+1)\left(P(\mathcal{W}) + P^2(\mathcal{W}) - 2mP^2(\mathcal{W}) + 2\mathrm{w}^t(\mathrm{S}(1) - \mathrm{I})\vec{1}\right)\\ &\quad + m(m-1)P^2(\mathcal{W}) - 2\mathrm{w}^t\dot{\mathrm{S}}(1) \cdot \vec{1}\ ,\end{aligned}$$

where $\dot{\mathrm{S}}(1)$ denotes the derivative of the matrix $\mathrm{S}(z)$ at $z = 1$.

(ii) *Let $\rho_{\mathcal{W}}$ be the smallest root of multiplicity one of* $\det \mathrm{D}(z) = 0$ *outside the unit circle* $|z| \leq 1$. *There exists $\rho > \rho_{\mathcal{W}}$ such that for $r = O(1)$*

$$\mathbf{P}(N_n(\mathcal{W}) = r) = (-1)^{r+1} \frac{a_{r+1}}{r!} (n)_r \rho_{\mathcal{W}}^{-(n-m+r+1)} + \sum_{j=1}^{r} (-1)^j a_j \binom{n}{j-1} \rho_{\mathcal{W}}^{-(n+j)} + O(\rho^{-n}) ,$$

where a_r are computable constants.

(iii) *Let B be a bounded real interval and $r = \lfloor \mathbf{E}(N_n) + x\sqrt{\mathrm{Var}(N_n)} \rfloor$. Then*

$$\sup_{x \in B} \left| \mathbf{P}(N_n(\mathcal{W}) = r) - \frac{1}{\sqrt{2\pi \mathrm{Var}(N_n)}} e^{-\frac{1}{2}x^2} \right| = o\left(\frac{1}{\sqrt{n}} \right),$$

as $n \to \infty$.

(iv) *Let $r = (1+\delta)\mathbf{E}(N_n)$ with $\delta \neq 0$, and let $a = (1+\delta)P(\mathcal{W})$. Define $\tau(t)$ to be the root of*

$$\det(\mathrm{I} - e^t \mathrm{M}(e^\tau)) = 0 ,$$

and ω_a and σ_a to be

$$-\tau'(\omega_a) = -a, \qquad -\tau''(\omega_a) = \sigma_a^2.$$

Then

$$\mathbf{P}(N_n(\mathcal{W}) = r) \sim \frac{1}{\sigma_a \sqrt{2\pi(n-m+1)}} e^{-(n-m+1)I(a)+\theta_a}$$

where $I(a) = a\omega_a + \tau(\omega_a)$ and θ_a is a computable constant (cf. Exercise 7.3.3).

Proof. We only sketch the derivation of part (iii) but we present two proofs. Our starting point is

$$N(z,u) = \mathrm{R}^t(z) u (\mathrm{I} - u\mathrm{M}(z))^{-1} \mathrm{U}(z)$$

shown in Theorem 7.3.2 to hold for $|z| < 1$ and $|u| < 1$. We may proceed in two different ways.

Method A: Determinant Approach.

Observe that

$$(\mathrm{I} - u\mathrm{M}(z))^{-1} = \frac{\mathrm{B}(z,u)}{\det(\mathrm{I} - u\mathrm{M}(z))}$$

where $\mathrm{B}(z,u)$ is a complex matrix. Let

$$Q(z,u) := \det(\mathrm{I} - u\mathrm{M}(z)),$$

and let $z_0 := \rho(u)$ be the smallest root of

$$Q(z,u) = \det(\mathrm{I} - u\mathrm{M}(z)) = 0.$$

Observe that $\rho(1) = 1$ by (7.3.5).

For our central limit result, we restrict out interest to $\rho(u)$ in a vicinity of $u = 1$. Such a root exists and is unique since for real z the matrix $\mathrm{M}(z)$ has all positive coefficients. The Perron–Frobenius theorem implies that all other roots $\rho_i(u)$ are of smaller modulus. Finally, one can analytically continue $\rho(u)$ to a complex neighborhood of u. Thus Cauchy's formula yields for some $A < 1$

$$\begin{aligned} N_n(u) := [z^n]N(z,u) &= \frac{1}{2\pi i}\oint \frac{\mathrm{R}^t(z)\mathrm{B}(z,u)\mathrm{U}(z)}{Q(z,u)}\frac{dz}{z^{n+1}} \\ &= C(u)\rho^{-n}(u)(1+O(A^n)) \end{aligned}$$

where $C(u) = -\mathrm{R}^t(\rho(u))\mathrm{B}(\rho(u),u)\mathrm{U}(\rho(u))\rho^{-1}(u)/Q'(\rho(u),u)$. As in the proof of Theorem 7.2.11, we recognize a quasi-power form for $N_n(u)$ that directly leads to the central limit theorem. An application of a saddle point method completes the proof of the local limit theorem.

Method B: Eigenvalue Approach

We apply now the Perron–Frobenius theorem for positive matrices together with a matrix spectral representation to obtain even more precise asymptotics. Our starting point is the following formula

$$[\mathrm{I} - u\mathrm{M}(z)]^{-1} = \sum_{k=0}^{\infty} u^k \mathrm{M}^k(z). \tag{7.3.8}$$

Now, observe that $\mathrm{M}(z)$ for real z, say x, is a positive matrix since each element $M_{ij}(x)$ is the generating function of the language $\mathcal{M}_{ij}$ and for any $v \in \mathcal{M}_{ij}$ we have $P(v) > 0$ for memoryless sources. Let then $\lambda_1(x), \lambda_2(x), \ldots, \lambda_K(x)$ are eigenvalues of $\mathrm{M}(x)$. By Perron–Frobenius result we know that $\lambda_1(x)$ is simple, real and $\lambda_1(x) > |\lambda_i(x)|$ for $i \geq 2$. (To simplify our further derivation, we also assume that $\lambda_i(x)$ are simple but this assumption will not have any significant impact on our asymptotics, as we shall see below.) Let l_i and r_i, $i = 1, \ldots, K$ are left and right eigenvectors corresponding to $\lambda_1(x), \lambda_2(x), \ldots, \lambda_K(x)$ eigenvalues, respectively. We set $\langle \mathrm{l}_1, \mathrm{r}_1 \rangle = 1$ where $\langle \mathrm{x}, \mathrm{y} \rangle$ is the scalar product of the vectors x and y. Since r_i is orthogonal to the left eigenvector r_j for $j \neq i$, we can write for any vector x

$$\mathrm{x} = \langle \mathrm{l}_1, \mathrm{x} \rangle \mathrm{r}_1 + \sum_{i=2}^{K} \langle \mathrm{l}_i, \mathrm{x} \rangle \mathrm{r}_i.$$

This yields

$$\mathrm{M}(x)\mathrm{x} = \langle \mathrm{l}_1, \mathrm{x} \rangle \lambda_1(x) \mathrm{r}_1 + \sum_{i=2}^{K} \langle \mathrm{l}_i, \mathrm{x} \rangle \lambda_i(x) \mathrm{r}_i.$$

Since $\mathrm{M}^k(x)$ has eigenvalues $\lambda_1^k(x), \lambda_2^k(x), \ldots, \lambda_K^k(x)$, then — dropping even the assumption about eigenvalues $\lambda_2, \ldots, \lambda_K$ being simple — we arrive at

$$\mathrm{M}^k(x)\mathrm{x} = \langle \mathrm{l}_1, \mathrm{x} \rangle \mathrm{r}_1 \lambda_1^k(x) + \sum_{i=2}^{K'} q_i(k) \langle \mathrm{l}_i, \mathrm{x} \rangle \mathrm{r}_i \lambda_i^k(x) \tag{7.3.9}$$

where $q_i(k)$ is a polynomial in k ($q_i(k) \equiv 1$ when the eigenvalues $\lambda_2, \ldots, \lambda_K$ are simple). Finally, we observe that we can analytically continue $\lambda_1(x)$ to complex plane due to separation of $\lambda_1(x)$ form other eigenvalues leading to $\lambda_1(z)$.

Applying now (7.3.9) to (7.3.8) and using it in the formula for $N(z,u)$ derived in Theorem 7.3.2 we obtain

$$
\begin{aligned}
N(z,u) &= \mathrm{R}^t(z)u[\mathrm{I} - u\mathrm{M}(z)]^{-1}\mathrm{U}(z) \\
&= u\mathrm{R}^t(z)\left(\sum_{k=0}^{\infty} u^k \lambda_1^k(z)\langle \mathrm{l}_1(z), \mathrm{U}(z)\rangle \mathrm{r}_1(z)\right. \\
&\qquad \left. + \sum_{i=2}^{K'} u^k \lambda_i^k(z)\langle \mathrm{l}_i(z), \mathrm{U}(z)\rangle \mathrm{r}_i(z)\right) \\
&= \frac{uC_1(z)}{1-u\lambda_1(z)} + \sum_{i=2}^{K'} \frac{uC_i(z)}{1-u\lambda_i(z)}
\end{aligned}
$$

for some polynomials $C_i(z)$. This representation entails to apply the Cauchy formula yielding, as before, for $A < 1$ and a polynomial $B(u)$

$$N_n(u) := [z^n]N(z,u) = B(u)\rho^{-n}(u)(1 + O(A^n))$$

where $\rho(u)$ is the smallest root of $1 - u\lambda(z) = 0$ which coincides with the smallest root of $\det(\mathrm{I} - u\mathrm{M}(u)) = 0$. In the above $A < 1$ since $\lambda_1(z)$ dominates all the other eigenvalues. In the next section we return to this method and discuss it in some more depth. ∎

7.3.2. Analysis of the generalized string matching

In this section we deal with a general pattern matching problem where words in $\mathcal{W}$ are not of the same length, that is, $\mathcal{W} = \bigcup_{i=1}^{d} \mathcal{W}_i$ such that $\mathcal{W}_i$ is a subset of $\mathcal{A}^{m_i}$ with all m_i being different. We still keep $\mathcal{W}_0 = \emptyset$ (i.e., there are no forbidden words). In the next section, we consider the case $\mathcal{W}_0 \neq \emptyset$. We present here a powerful method based on a finite automata (i.e., de Bruijn graph). This approach is very versatile, but unfortunately is not as insightful as the combinatorial approach discussed so far.

Our goal is to derive the probability generating function $N_n(u) = \mathbf{E}(u^{N_n(\mathcal{W})})$ of the number of pattern $\mathcal{W}$ occurrences in the text. We start with building an automaton that scans the text $X_1X_2\cdots X_n$ and recognizes occurrences of patterns from the set $\mathcal{W}$. As a matter of fact, our automaton is a de Bruijn graph that we describe in the sequel: Let $M = \max\{m_1, \ldots, m_d\} - 1$ and $\mathcal{B} = \mathcal{A}^M$. The de Bruijn automaton is built over the state space $\mathcal{B}$. Let $b \in \mathcal{B}$ and $a \in \mathcal{A}$. Then a transition from a state b upon scanning symbol a of the text is to $\hat{b} \in \mathcal{B}$ such that

$$\hat{b} = b_2b_3\cdots b_M a,$$

that is, the leftmost symbol of b is erased and symbol a is appended on the right. We shall denote such a transition as $ba \mapsto \hat{b}$ or $ba \in \mathcal{A}\hat{b}$ since the first symbol of b has been deleted when scanning symbol a. When scanning a text of length $n-M$ one constructs an associated path of length $n-M$ in the de Bruijn automaton that begins at a state formed by the first M symbols of the text, that is, $b = X_1 X_2 \cdots X_M$.

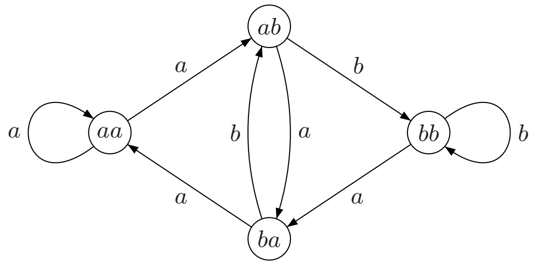


Figure 7.2. The de Bruijn graph for $\mathcal{W} = \{ab, aab, aba\}$.

To record the *number* of pattern occurrences we equip the automaton with a counter $\phi(b,a)$. When a transition occurs, we increment $\phi(b,a)$ by the number of occurrences of patterns from $\mathcal{W}$ in the text ba. Since all occurrences of patterns from $\mathcal{W}$ that end at a are contained in the text of the form ba, we realize that

$$\phi(b,a) = N_{M+1}(\mathcal{W}, ba) - N_M(\mathcal{W}, b)$$

where $N_k(\mathcal{W}, x)$ is the number of pattern occurrences in the text x of length k. Having built such an automaton, we construct a transition $V^M \times V^M$ matrix $\mathrm{T}(u)$ as a function of a complex variable u and indexed by $\mathcal{B} \times \mathcal{B}$ such that

$$\begin{aligned}[\mathrm{T}(u)]_{b,\hat{b}} &:= P(a) u^{\phi(b,a)} [\![\, ba \in \mathcal{A}\hat{b} \,]\!] \\ &= P(a) u^{N_{M+1}(\mathcal{W},ba) - N_M(\mathcal{W},b)} [\![\, \hat{b} = b_2 b_3 \cdots b_M a \,]\!]\end{aligned} \tag{7.3.10}$$

where Iverson's bracket convention is used:

$$[\![B]\!] = \begin{cases} 1 \text{ if the property } B \text{ holds,} \\ 0 \text{ otherwise.} \end{cases}$$

Example 7.3.4. Let $\mathcal{W} = \{ab, aab, aba\}$. Then $M = 2$, the de Bruijn graph is presented in Figure 7.2, and the matrix $\mathrm{T}(u)$ is shown below

$$\mathrm{T}(u) = \begin{array}{c c} & \begin{array}{cccc} aa & ab & ba & bb \end{array} \\ \begin{array}{c} aa \\ ab \\ ba \\ bb \end{array} & \begin{pmatrix} P(a) & P(b)\,u & 0 & 0 \\ 0 & 0 & P(a)\,u^2 & P(b) \\ P(a) & P(b) & 0 & 0 \\ 0 & 0 & P(a) & P(b) \end{pmatrix} \end{array}.$$

Next, we extend the above construction to scan a text of length $k \geq M$. By combinatorial properties of matrix products, the entry of index $b, \hat{b}$ of the power $\mathrm{T}^k(u)$ cumulates all terms corresponding to starting in state b, ending in state $\hat{b}$, and recording the total number of occurrences of patterns $\mathcal{W}$ found upon scanning the last k letters of the text. Therefore,

$$\left[\mathrm{T}^k(u)\right]_{b,\hat{b}} = \sum_{v \in \mathcal{A}^k} P(v)\, u^{N_{M+k}(\mathcal{W},bv)-N_M(\mathcal{W},b)}. \tag{7.3.11}$$

Define now a vector $\mathrm{x}(u)$ indexed by b as

$$\left[\mathrm{x}(u)\right]_b = P(b)\, u^{N_M(\mathcal{W},b)}.$$

Then, the summation of all the entries of the row vector $\mathrm{x}(u)^{\mathbf{t}}\mathrm{T}^k(u)$ is achieved by means of the vector $\vec{1} = (1, \dots, 1)$ so that the quantity $\mathrm{x}(u)^{\mathbf{t}}\mathrm{T}(u)^k\vec{1}$ represents the probability generating function of $N_{k+M}(\mathcal{W})$ taken over all texts of length $M+k$. By setting $n = M+k$ we prove the following theorem.

Theorem 7.3.5. *Consider a general pattern $\mathcal{W} = (\mathcal{W}_1, \dots, \mathcal{W}_d)$ with $M = \max\{m_1, \dots, m_d\} - 1$. Let $\mathrm{T}(u)$ be the transition matrix defined as*

$$\left[\mathrm{T}(u)\right]_{b,\hat{b}} := P(a) u^{N_{M+1}(\mathcal{W},ba)-N_M(\mathcal{W},b)} [\![\, \hat{b} = b_2 b_3 \cdots b_M a \,]\!]$$

where $b, \hat{b} \in \mathcal{A}^M$ and $a \in \mathcal{A}$. Then

$$N_n(u) = \mathbf{E}(u^{N_n(\mathcal{W})}) = \mathrm{b}^t(u)\mathrm{T}^n(u)\vec{1} \tag{7.3.12}$$

where $\mathrm{b}^t(u) = \mathrm{x}^t(u)\mathrm{T}^{-M}(u)$. Also,

$$N(z,u) = \sum_{n \ge 0} N_n(z) z^n = \mathrm{b}^t(u)(\mathrm{I} - z\mathrm{T}(u))^{-1}\vec{1} \tag{7.3.13}$$

for $|z| < 1$.

Let us now return for a moment to the reduced pattern case discussed in the previous section and compare expression (7.3.13) derived here with (7.3.3) of Theorem 7.3.2 that we repeat below

$$N(z,u) = \mathrm{R}^t(z) u (\mathrm{I} - u\mathrm{M}(z))^{-1}\mathrm{U}(z).$$

Although there is a striking resemblance of these formulas they are quite different. In (7.3.3) $\mathrm{M}(z)$ is a matrix of z representing generating functions of languages $\mathcal{M}_{ij}$, while $\mathrm{T}(u)$ is a function of u and it is the transition matrix of the associated de Bruijn graph. Nevertheless, the eigenvalue method discussed in the proof of Theorem 7.3.3 can be directly applied to derive limit laws of $N_n(\mathcal{W})$ for general set of patterns $\mathcal{W}$. We shall discuss it next.

To study asymptotics of $N_n(\mathcal{W})$ we need to estimate the growth of $T^n(u)$ which is governed by the growth of the largest eigenvalue, as we have already seen in the previous sections. Here, however, the situation is a little more complicated since the matrix $\mathrm{T}(u)$ is irreducible but not necessary primitive (cf. Chapter 1 for in depth discussion). To be more precise, $\mathrm{T}(u)$ is *irreducible* if its associated de Bruijn graph is strongly connected, while for *primitivity* of $\mathrm{T}(u)$ we require that the greatest common divisor of the cycle weights of the de Bruijn graph is equal to one.

Let us first verify irreducibility of $\mathrm{T}(u)$. As easy to check the matrix is irreducible since for any $g \geq M$ and $b, \hat{b} \in \mathcal{A}^M$ there are two words $w, v \in \mathcal{A}^g$ such that $bw = v\hat{b}$ (e.g., for $g = M$ one can take $w = \hat{b}$ and $v = b$). Thus $\mathrm{T}^g(u) > 0$ for $u > 0$ which is sufficient for irreducibility.

Let us now have a closer look at the primitivity of $\mathrm{T}(u)$. We start with a precise definition. Let $\psi(b, \hat{b}) := \phi(ba)$ where $ba \mapsto \hat{b}$ be the counter value when transitioned form b to $\hat{b}$. Let also $\mathcal{C}$ be a cycle in the associated de Bruijn graph. Define the total weight of the cycle $\mathcal{C}$ as

$$\psi(\mathcal{C}) = \sum_{b, \hat{b} \in \mathcal{C}} \psi(b, \hat{b}).$$

Finally, we set $\psi_{\mathcal{W}} = \gcd(\psi(\mathcal{C}) : \ \mathcal{C} \text{ cycle})$. If $\psi_{\mathcal{W}} = 1$, then we say $\mathrm{T}(u)$ is primitive.

Example 7.3.4 (*continued*). Consider again the matrix $\mathrm{T}(u)$ and its associated graph shown in Figure 7.2. There are six cycles of respective weights $0, 3, 2, 0, 0, 1$, therefore $\psi_{\mathcal{W}} = 1$ and $\mathrm{T}(u)$ is primitive.

Consider now another matrix

$$\mathrm{T}(u) = \begin{pmatrix} P(a) & P(b)\, u^4 \\ P(a)\, u^2 & P(b)\, u^3 \end{pmatrix}.$$

This time there are three cycles of weights $0, 6$ and 3 and $\psi_{\mathcal{W}} = 3$. The matrix is not primitive. Observe that the characteristic polynomial $\lambda(u)$ of this matrix is a polynomial in u^3.

Observe that the diagonal elements of $\mathrm{T}(u)^k$ (i.e., its trace) are polynomials in u^ℓ if and only if ℓ divides $\psi_{\mathcal{W}}$; therefore, the characteristic polynomial $\det(z\mathrm{I} - \mathrm{T}(u))$ of $\mathrm{T}(u)$ is a polynomial in $u^{\psi_{\mathcal{W}}}$. Indeed, it is known that for any matrix A

$$\det(\mathrm{I} - A) = \exp\left(\sum_{k \geq 0} - \frac{\mathrm{Tr}[A^k]}{k} \right)$$

where $\mathrm{Tr}[A]$ is the trace of A.

Asymptotic behavior of the generating function $N_n(u) = \mathbf{E}(u^{N_n(\mathcal{W})})$, hence $N_n(\mathcal{W})$, depends on the growth of $\mathrm{T}^n(u)$. The next lemma summarizes some useful properties of $\mathrm{T}(u)$ and its eigenvalues. For the matrix $\mathrm{T}(u)$ of dimension $|\mathcal{A}|^M \times |\mathcal{A}|^M$ we denote by $\lambda_j(u)$ for $j = 1, \ldots, R = |\mathcal{A}^M|$ its eigenvalues and we assume that $|\lambda_1(u)| \geq |\lambda_2(u)| \geq \cdots \geq |\lambda_R(u)|$. To simplify notation, we often drop the index of the largest eigenvalue, that is, $\lambda(u) := \lambda_1(u)$. Observe that $\varrho(u) = |\lambda(u)|$ is known as the *spectral radius* and it is equal to

$$\varrho(u) = \lim_{n \to \infty} ||T^n(u)||^{1/n}$$

where $||\cdot||$ is any matrix norm.

LEMMA 7.3.6. *Let $\mathcal{G}_M(\mathcal{W})$ and $\mathrm{T}(u)$ denote, respectively, the de Bruijn graph and its associated matrix defined in (7.3.10) for general pattern $\mathcal{W}$. Assume $P(\mathcal{W}) > 0$.*

(i) *For $u > 0$ the matrix $\mathrm{T}(u)$ has a unique dominant eigenvalue $\lambda(u)$ ($> \lambda_j(u)$ for $j = 2, \ldots, |\mathcal{A}|^M$) that is strictly positive and a dominant eigenvector $\mathrm{r}(u)$ whose all entries are strictly positive. Furthermore, there exists a complex neighborhood of the real positive axis on which the mappings $u \to \lambda(u)$ and $u \to \mathrm{r}(u)$ are well-defined and analytic.*

(ii) *Define $\Lambda(s) := \log \lambda(e^s)$ for s complex. For real s the function $s \to \Lambda(s)$ is strictly increasing and strictly convex. In addition,*

$$\Lambda(0) = 1, \qquad \Lambda'(0) = P(\mathcal{W}) > 0, \qquad \Lambda''(0) := \sigma^2(\mathcal{W}) > 0.$$

(iii) *For any $\theta \in (0, 2\pi)$ and x real $\varrho(xe^{i\theta}) \le \varrho(x)$.*

(iv) *For any $\theta \in (0, 2\pi)$, if $\psi_{\mathcal{W}} = 1$, then for x real $\varrho(xe^{i\theta}) < \varrho(x)$; otherwise $\psi_{\mathcal{W}} = d > 1$ and $\varrho(xe^{i\theta}) = \varrho(x)$ if and only if $\theta = 2k\pi/d$.*

Proof. We first prove (i). Take $u > 0$ real positive. Then the matrix $\mathrm{T}(u)$ has positive entries, and for any exponent $g \ge M$ the gth power of $\mathrm{T}(u)$ has strictly positive entries, as shown above (see irreducibility of $\mathrm{T}(u)$). Therefore, by the Perron–Frobenius theorem (cf. also Chapter 1) there exists an eigenvalue $\lambda(u)$ that dominates strictly all the others. Moreover, it is simple and strictly positive. In other words, one has

$$\lambda(u) := \lambda_1(u) > |\lambda_2(u)| \ge |\lambda_3(u)| \ge \cdots.$$

Furthermore, the corresponding eigenvector $r(u)$ has all its components strictly positive. Since the dominant eigenvalue is separated from other eigenvalues, by perturbation theory there exists a complex neighborhood of the real positive axis where the functions $u \to \lambda(u)$ and $u \to \mathrm{r}(u)$ are well-defined and analytic. Moreover, $\lambda(u)$ is an algebraic function since it satisfies the characteristic equation $\det(\lambda\mathrm{I} - \mathrm{T}(u)) = 0$.

We now prove part (ii). The increasing property for $\lambda(u)$ (and thus for $\Lambda(s)$) is a consequence of the fact that if A and B are nonnegative irreducible matrices such that $A_{i,j} \ge B_{i,j}$ for all (i, j), then the spectral radius of A is larger than the spectral radius of B.

For convexity of $\Lambda(s)$, it is sufficient to prove that for $u, v > 0$

$$\lambda(\sqrt{uv}) \le \sqrt{\lambda(u)}\sqrt{\lambda(v)}.$$

Since eigenvectors are defined up to a constant, one can always choose the eigenvectors $\mathrm{r}(\sqrt{uv})$, $\mathrm{r}(u)$, and $\mathrm{r}(v)$ such that

$$\max_i \frac{r_i(\sqrt{uv})}{\sqrt{r_i(u)\, r_i(v)}} = 1.$$

Suppose that this maximum is attained at some index i. We denote by P_{ij} the coefficient at u in $\mathrm{T}(u)$, that is, $P_{ij} = [u^{\psi}][\mathrm{T}(u)]_{ij}$. By the Cauchy-Schwarz inequality we have

$$\begin{aligned}
\lambda(\sqrt{uv})r_i(\sqrt{uv}) &= \sum_j P_{ij}\,(\sqrt{uv})^{\psi(i,j)}\,r_j(\sqrt{uv}) \\
&\le \sum_j P_{ij}(\sqrt{uv})^{\psi(i,j)}\,\sqrt{r_j(u)\,r_j(v)} \\
&\le \left(\sum_j P_{ij}\,u^{\psi(i,j)}\,r_j(u)\right)^{1/2} \left(\sum_j P_{ij}\,v^{\psi(i,j)}\,r_j(v)\right)^{1/2} \\
&= \sqrt{\lambda(u)}\sqrt{\lambda(v)}\;\sqrt{r_i(u)\,r_i(v)},
\end{aligned}$$

which implies convexity of $\Lambda(s)$. To show that $\Lambda(s)$ is strictly convex, we argue as follows: Observe that for $u = 1$ the matrix $\mathrm{T}(u)$ is stochastic, hence $\lambda(1) = 1$ and $\Lambda(0) = 0$. As we shall see below, the mean and the variance of $N_n(\mathcal{W})$ are equal asymptotically to $n\Lambda'(0)$ and $n\Lambda''(0)$, respectively. From the problem formulation, we conclude that $\Lambda'(0) = P(\mathcal{W}) > 0$ and $\Lambda''(0) = \sigma^2(\mathcal{W}) > 0$. Therefore, $\Lambda'(s)$ and $\Lambda''(s)$ cannot be always 0 and (since they are analytic) they cannot be zero on any interval. This implies that $\Lambda(s)$ is strictly increasing and strictly convex.

We now establish part (iii). For $|u| = 1$, and x real positive, consider two matrices $\mathrm{T}(x)$ and $\mathrm{T}(xu)$. From (i) we know that for $\mathrm{T}(x)$ there exist a dominant strictly positive eigenvalue $\lambda := \lambda(x)$ and a dominant eigenvector $\mathrm{r} := \mathrm{r}(x)$ whose all entries r_j are strictly positive. Consider an eigenvalue ν of $\mathrm{T}(xu)$ and its corresponding eigenvector $s := s(u)$. Denote by v_j the ratio s_j/r_j. One can always choose r and s such that $\max_{1\le j\le R} |v_j| = 1$. Suppose that this maximum is attained for some index i. Then

$$|\nu s_i| = |\sum_j P_{ij}\,(xu)^{\psi(i,j)}\,s_j| \le \sum_j P_{ij}\,x^{\psi(i,j)}\,r_j = \lambda r_i. \tag{7.3.14}$$

We conclude that $|\nu| \le \lambda$, and part (iii) is proven.

Finally we deal with part (iv). Suppose now that the equality $|\nu| = \lambda$ holds. Then, all the previous inequalities in (7.3.14) become equalities. First, for all indices ℓ such that $P_{i,\ell} \ne 0$, we deduce that $|s_\ell| = r_\ell$, and v_ℓ has modulus 1. For these indices ℓ, we have the same equalities in (7.3.14) as for i. Finally, the transitivity of the de Bruijn graph entails that that each complex v_j is of modulus 1. Now, the converse of the triangular inequality shows that for every edge $(i, j) \in \mathcal{G}_M(\mathcal{W})$ we have

$$u^{\psi(i,j)}v_j = \frac{\nu}{\lambda}v_i,$$

and for any cycle of length L we conclude that

$$\left(\frac{\nu}{\lambda}\right)^L = u^{\psi(\mathcal{C})}.$$

However, for any pattern $\mathcal{W}$ there exists a cycle $\mathcal{C}$ of length one with weight $\psi(\mathcal{C}) = 0$, as easy to see. This proves that $\nu = \lambda$ and that $u^{\psi(\mathcal{C})} = 1$ for any cycle $\mathcal{C}$. If $\psi_{\mathcal{W}} = \gcd(\psi(\mathcal{C}),\ \mathcal{C} \text{ cycle}) = 1$, then $u = 1$ and $\varrho(xe^{i\theta}) < \varrho(x)$ for $\theta \in (0, 2\pi)$.

Suppose now that $\psi_{\mathcal{W}} = d > 1$. Then, the characteristic polynomial and the dominant eigenvalue $\lambda(v)$ are functions of v^d. The lemma is proved. ∎

Lemma 7.3.6 provides the main technical support to prove the forthcoming results; in particular, to establish asymptotic behavior of $\mathrm{T}^n(u)$ for large n. Indeed, our starting point is (7.3.13) to which we apply the spectral decomposition as in (7.3.9) to conclude that

$$N(z, u) = \frac{c(u)}{1 - z\lambda(u)} + \sum_{i \geq 2} \frac{c_i(u)}{(1 - z\lambda_i(u))^{\alpha_i}}.$$

where $\alpha_i \geq 1$ are some integers. In the above, $\lambda(u)$ is the dominant eigenvalue, while $\lambda_i(u) < \lambda(u)$ are other eigenvalues. The numerator has the expression $c(u) = \mathrm{b}^t(u)\langle \mathrm{l}(u), \vec{1}\rangle \mathrm{r}(u)$ where $\mathrm{l}(u)$ and $\mathrm{r}(u)$ are the left and the right dominant eigenvectors and $\mathrm{b}^t(u)$ is defined after (7.3.12). Then Cauchy's coefficient formula implies

$$N_n(u) = c(u)\lambda^n(u)(1 + O(A^n)) \tag{7.3.15}$$

for some $A < 1$. Equivalently, the moment generating function for $N_n(\mathcal{W})$ is given by the following uniform approximation in a neighborhood of $s = 0$

$$\mathbf{E}(e^{sN_n(\mathcal{W})}) = d(s)\lambda^n(e^s)(1 + O(A^n)) = d(s)\exp\left(n\Lambda(s)\right)\left(1 + O(A^n)\right) \tag{7.3.16}$$

where $d(s) = c(e^s)$ and $\Lambda(s) = \log \lambda(e^s)$.

There is another, more general, derivation of (7.3.15). Observe that the spectral decomposition of $\mathrm{T}(u)$ when u lies in a sufficiently small complex neighborhood of any compact subinterval of $(0, +\infty)$ is of the form

$$\mathrm{T}(u) = \lambda(u)\mathrm{Q}(u) + \mathrm{R}(u) \tag{7.3.17}$$

where $\mathrm{Q}(u)$ is the projection under the dominant eigensubspace and $\mathrm{R}(u)$ a matrix whose spectral radius equals $|\lambda_2(u)|$. Therefore,

$$\mathrm{T}(u)^n = \lambda(u)^n Q(u) + R(u)^n,$$

entails the estimate (7.3.15). The next results follows immediately from (7.3.16).

Theorem 7.3.7. *Let $\mathcal{W} = (\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_d)$ be a generalized pattern with $\mathcal{W}_0 = \emptyset$ generated by a memoryless source. For large n*

$$\mathbf{E}(N_n(\mathcal{W})) = n\Lambda'(0) + O(1) = nP(\mathcal{W}) + O(1), \tag{7.3.18}$$

$$\mathrm{Var}(N_n(\mathcal{W})) = n\Lambda''(0) + O(1) = n\sigma^2(\mathcal{W}) + O(1) \tag{7.3.19}$$

where $\Lambda(s) = \log \lambda(e^s)$ and $\lambda(u)$ is the largest eigenvalue of $\mathrm{T}(u)$. Furthermore,

$$\mathbf{P}(N_n(\mathcal{W}) = 0) = C\lambda^n(0)(1 + O(A^n))$$

where $C > 0$ is a constant and $A < 1$.

Now we establish limit laws, starting with the central limit law and its local limit law.

THEOREM 7.3.8. *Under the same assumption as for Theorem 7.3.7, the following holds*

$$\sup_{x\in B}\left|\mathbf{P}\left(\frac{N_n(\mathcal{W})-nP(\mathcal{W})}{\sigma(\mathcal{W})\sqrt{n}}\le x\right)-\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}e^{-t^2/2}\,dt\right|=O\left(\frac{1}{\sqrt{n}}\right) \quad (7.3.20)$$

where B is a bounded real interval.

Proof. The uniform asymptotic expansion (7.3.16) of a sequence of moment generating functions is known as a "quasi-powers approximation". Then an application of the classical Levy continuity theorem leads to the Gaussian limit law. An application of the Berry-Essen inequality provides the speed of convergence which is $O(1/\sqrt{n})$. This proves the theorem. ∎

Finally, we deal with the large deviations.

THEOREM 7.3.9. *Under the same assumption as before, Let ω_a be a solution of*

$$\omega\lambda'(\omega)=a\lambda(\omega)$$

for some $a\ne P(\mathcal{W})$, where $\lambda(u)$ is the largest eigenvalue of $\mathrm{T}(u)$. Define

$$I(a)=a\log\omega_a-\log\lambda(\omega_a). \quad (7.3.21)$$

Then there exists a constant $C>0$ such that $I(a)>0$ for $a\ne P(\mathcal{W})$ and

$$\lim_{n\to\infty}\frac{1}{n}\log\mathbf{P}\left(N_n(\mathcal{W})\le an\right)=-I(x) \qquad \text{if } \ 0<x<P(\mathcal{W}) \quad (7.3.22)$$

$$\lim_{n\to\infty}\frac{1}{n}\log\mathbf{P}\left(N_n(\mathcal{W})\ge na\right)=-I(x) \qquad \text{if } \ P(\mathcal{W})<x<C. \quad (7.3.23)$$

Proof. We consider now large deviations and establish (7.3.22). The variable $N_n(\mathcal{W})$ is by definition of at most linear growth, and there exists a a constant C such that $N_n(\mathcal{W})\le Cn+O(1)$. Let $0<x<P(\mathcal{W})$. Cauchy's coefficient formula provides

$$\mathbf{P}\left(N_n(\mathcal{W})\le k\right)=\frac{1}{2i\pi}\int_{|u|=r}\frac{N_n(u)}{u^k}\,\frac{du}{u(1-u)}.$$

For ease of exposition, we first discuss the case of a primitive pattern. We recall that a pattern is primitive if $\psi_{\mathcal{W}}=\gcd(\psi(\mathcal{C}),\ \mathcal{C}\text{ cycle})=1$. The strict domination property expressed in Lemma 7.3.6(iv) for primitive patterns implies that the above integrand is strictly maximal at the intersection of the circle $|u|=r$ and the positive real axis. Near the positive real axis, where the contribution of the integrand is concentrated, the following uniform approximation holds, with $k=na$:

$$\frac{N_n(u)}{u^k}=\exp\left(n\left(\log\lambda(u)-a\log u\right)\right)(1+o(1)) \quad (7.3.24)$$

The saddle point equation is then obtained by cancelling the first derivative yielding

$$F(\omega) := \frac{\omega\lambda'(\omega)}{\lambda(\omega)} = a. \tag{7.3.25}$$

Note that the function F is exactly the derivative of $\Lambda(s)$ at point $s := \log \omega$. Since $\Lambda(s)$ is strictly convex, the left side is an increasing function of its argument as proved in Lemma 7.3.6(ii). Also, we know form this lemma that the value $F(0) = 0$, $F(1) = P(\mathcal{W})$ while we set $F(\infty) = C$. Thus, for any real a in $(0, C)$, equation (7.3.25) always admits a unique positive solution that we denote by $\omega \equiv \omega_a$. Moreover, for $a \neq P(\mathcal{W})$, one has $\omega_a \neq 1$. Since the function

$$u \to -\log \frac{\lambda(u)}{u^a}$$

admits a strict maximum at $u = \omega_a$, hence this maximum $I(a)$ is strictly positive. Finally, the usual saddle point approximation applies and one finds

$$\mathbf{P}\left(\frac{N_n(\mathcal{W})}{n} \le a\right) = \left(\frac{\lambda(\omega_a)}{\omega_a^a}\right)^n \Theta(n),$$

where $\Theta(n)$ is of the order of $n^{-1/2}$. In summary, the large deviation rate is

$$I(a) = -\log \frac{\lambda(\omega_a)}{\omega_a^a} \qquad \text{with} \qquad \frac{\omega_a\lambda'(\omega_a)}{\lambda(\omega_a)} = a.$$

as shown in the theorem.

In the general case when the pattern is not primitive, the strict inequality of Lemma 7.3.6(iv) is not satisfied, and several saddle points may be present on the circle $|u| = r$, which will lead to some oscillations. We must, in this case, use the weaker inequality of Lemma 7.3.6, namely, $\varrho(xe^{i\theta}) \le \varrho(x)$, which replaces the strict inequality. However, the factor $(1-u)^{-1}$ present in the integrand of (7.3.24) attains its maximum modulus on $|u| = r$ solely at $u = r$. Thus, the contribution of possible saddle points can only affect a fraction of the contribution from $u = r$. Consequently, (7.3.22) and (7.3.21) continue to be valid. A similar reasoning provides the right tail estimate, with $I(a)$ still given by (7.3.21). This completes the proof of (7.3.22). ∎

We complete this analysis with a local limit law.

Theorem 7.3.10. *If* $\mathrm{T}(u)$ *is primitive, then*

$$\sup_{x\in B}\left|\mathbf{P}\left(N_n = nP(\mathcal{W}) + x\sigma(\mathcal{W})\sqrt{n}\right) - \frac{1}{\sigma(\mathcal{W})\sqrt{n}}\frac{e^{x^2/2}}{\sqrt{2\pi}}\right| = o\left(\frac{1}{\sqrt{n}}\right) \tag{7.3.26}$$

where B *is a bounded real interval. Furthermore, under the above additional assumption, one can find constants* σ_a *and* δ_a *such that*

$$\mathbf{P}(N_n(\mathcal{W}) = a\mathbf{E}(N_n)) \sim \frac{1}{\sigma_a\sqrt{2\pi n}} e^{-nI(a)+\theta_a} \tag{7.3.27}$$

where $I(a)$ *is defined in (7.3.21) above.*

Proof Stronger "regularity conditions" are needed in order to obtain local limit estimates. Roughly, one wants to exclude the possibility that the discrete distribution is of a lattice type, being supported by a nontrivial sublattice of the integers. (For instance, we need to exclude the possibility for $N_n(\mathcal{W})$ to be always odd, or of the parity of n, and so on.) Observe first that positivity and irreducibility of the matrix $\mathrm{T}(u)$ is not enough as shown in Example 7.3.4.

By Lemma 7.3.6, one can estimate the probability distribution of $N_n(\mathcal{W})$ by the classical saddle point method in the case when $\mathcal{W}$ is primitive. Again, one starts from Cauchy's coefficient integral,

$$\mathbf{P}(N_n(\mathcal{W}) = k) = \frac{1}{2i\pi} \int_{|u|=1} N_n(u) \frac{du}{u^{k+1}}, \tag{7.3.28}$$

where k is of the form $k = nP(\mathcal{W})n + x\sigma(\mathcal{W})\sqrt{n}$. Property (iv) of Lemma 7.3.6 grants us precisely the fact that any closed arc of the unit circle not containing $u = 1$ brings an exponentially negligible contribution. A standard application of the saddle point technique does the job. In this way, the proof of the local limit law of Theorem 7.3.10 is completed. Finally, the precise large deviations follows from the local limit result and an application of the method of shift discussed in the proof of Theorem 7.2.12. ∎

7.3.3. Forbidden words and (ℓ, k) sequences

Finally, consider the general pattern $\mathcal{W} = (\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_d)$ with nonempty forbidden set $\mathcal{W}_0$. In this case, we study the number of occurrences $N_n(\mathcal{W}|\mathcal{W}_0 = 0)$ of patterns $\mathcal{W}_1, \ldots \mathcal{W}_d$ under the condition that there is no occurrence in the text of any pattern from $\mathcal{W}_0$.

Fortunately, we can recover almost all results from our previous analysis after re-defining the matrix $\mathrm{T}(u)$ and its de Bruijn graph. We now change (7.3.10) to

$$[\mathrm{T}(u)]_{b,\hat{b}} := P(a)u^{\phi(b,a)}[\![\, ba \in \mathcal{A}\hat{b} \text{ and } ba \not\subset \mathcal{W}_0]\!] \tag{7.3.29}$$

where $ba \subset \mathcal{W}_0$ means that any subword of ba belongs to $\mathcal{W}_0$. In words, we force the matrix $\mathrm{T}(u)$ to be zero at any position that leads to a word containing patterns from $\mathcal{W}_0$, that is, we eliminate from the de Bruijn graph any transition that contains a forbidden word. Having matrix $\mathrm{T}(u)$ constructed, we can repeat all previous results except that it is much harder to find explicit formulas even for the mean and the variance (cf. Exercise 7.3.4)

Finally, we consider a degenerated general pattern in which $\mathcal{W}_i = \emptyset$ for all $i = 1, \ldots, d$ except nonempty $\mathcal{W}_0$. In this case, we count the number of sequences that do not contain a pattern from $\mathcal{W}_0$. We only consider the special case of this problem, that of (ℓ, k) sequences for which $\mathcal{W}_0$ is defined in (7.3.1). In particular, we compute the so called *Shannon capacity* $C_{\ell,k}$ defined as

$$C_{\ell,k} = \lim_{n\to\infty} \frac{\log(\text{number of } (\ell,k) \text{ sequence of length } n)}{n}.$$

We first compute the ordinary generating function $T_{\ell,k}(z) = \sum_{w\in\mathcal{T}_{\ell,k}} z^{|w|}$ of all (ℓ,k) words denoted as $\mathcal{T}_{\ell,k}$. To enumerate $\mathcal{T}_{\ell,k}$ we define $\mathcal{D}_{\ell,k}$ as the set of all words consisting only of runs of 0's whose length is between ℓ and k. The generating function $D(z)$ is clearly equal to

$$D(z) = z^\ell + z^{\ell+1} + \cdots + z^k = z^\ell \frac{1-z^{k-\ell+1}}{1-z}.$$

We now observe that $\mathcal{T}_{\ell,k}$ can be symbolically written as

$$\mathcal{T}_{\ell,k} = \mathcal{D}_{\ell,k}\left(\{1\}\times\varepsilon + \bar{\mathcal{D}}_{\ell,k} + \bar{\mathcal{D}}_{\ell,k}\times\bar{\mathcal{D}}_{\ell,k} + \cdots + \bar{\mathcal{D}}_{\ell,k}^k + \cdots\right), \tag{7.3.30}$$

where $\bar{\mathcal{D}}_{\ell,k} = \{1\}\times\mathcal{D}_{\ell,k}$. Above basically says that the collection of (ℓ,k) sequences, $\mathcal{T}_{\ell,k}$, is a concatenation of $\{1\}\times\mathcal{D}_{\ell,k}$. Thus (7.3.30) translates into the generating functions $T_{\ell,k}(z)$ as follows

$$\begin{aligned} T_{\ell,k}(z) = D(z)\frac{1}{1-zD(z)} &= \frac{z^\ell(1-z^{k+1-\ell})}{1-z-z^{\ell+1}+z^{k+2}} \\ &= \frac{z^\ell + z^{\ell+1}+\cdots+z^k}{1-z^{\ell+1}-z^{\ell+2}-\cdots-z^{k+1}}. \end{aligned} \tag{7.3.31}$$

Then Shannon capacity $C_{\ell,k}$ is

$$C_{\ell,k} = \lim_{n\to\infty}\frac{\log[z^n]T_{\ell,k}(z)}{n}.$$

If ρ is the smallest root in absolute value of $1-z^{\ell+1}-z^{\ell+2}-\cdots-z^{k+1}=0$, then clearly

$$C_{\ell,k} = -\log\rho.$$

Example 7.3.11. In this example, we show that one can enumerate more precisely (ℓ,k) sequences. In fact, since the function $T_{\ell,k}(z)$ is rational we can compute $[z^n]T_{\ell,k}(z)$ exactly. Let us consider a particular case, namely, $\ell=1$ and $k=3$. Then the denumerator in (7.3.31) becomes $1-z^2-z^3-z^4$, and its roots are

$$\rho_{-1}=-1,\quad \rho_0 = 0.682327\ldots,\quad \rho_1 = -0.341164\ldots + i1.161541\ldots,\quad \rho_2=\bar{\rho}_1.$$

Computing residues we obtain

$$\begin{aligned} [z^n]T_{1,3}(z) = &\frac{\rho_0+\rho_0^2+\rho_0^3}{(\rho_1+1)(\rho_0-\rho_1)(\rho_0-\bar{\rho}_1)}\rho_0^{-n-1} \\ &+(-1)^{n+1}\frac{1}{(\rho_0+1)(\rho_1+1)(\bar{\rho}_1+1)} + O(r^{-n}), \end{aligned}$$

where $r\approx 0.68$. More specifically,

$$[z^n]T_{1,3}(z) = 0.594(1.465)^{n+1} + 0.189(-1)^{n+1} + O(0.68^n).$$

for large n.

7.4. Subsequence pattern matching

In string matching problem, given a pattern $\mathcal{W}$ one searches for some/all occurrences of $\mathcal{W}$ as a block of consecutive symbols in a text. We analyzed various string matching problems in the previous sections. Here we concentrate on *subsequence pattern matching*. In this case we search for a given pattern $\mathcal{W} = w_1 w_2 \dots w_m$ in the text $X = x_1 x_2 \dots x_n$ as a *subsequence*, that is, we look for indices $1 \le i_1 < i_2 < \cdots < i_m \le n$ such that $x_{i_1} = w_1$, $x_{i_2} = w_2$, $\cdots$, $x_{i_m} = w_m$. We also say that the word $\mathcal{W}$ is *"hidden"* in the text; thus we call this the *hidden pattern* problem. For example, `date` occurs as a subsequence in the text `hidden pattern`, in fact four times, but not even once as a string.

More specifically, we allow the possibility of imposing an additional set of constraints $\mathcal{D}$ on the indices $i_1, i_2, \dots, i_m$ to record a valid subsequence occurrence. For a given family of integers d_j ($d_j \ge 1$, possibly $d_j = \infty$), one should have $(i_{j+1} - i_j) \le d_j$. More formally, the hidden pattern specification is determined by a pair $(\mathcal{W}, \mathcal{D})$ where $\mathcal{W} = w_1 \cdots w_m$ is a word of length m and the *constraint* $\mathcal{D} = (d_1, \dots, d_{m-1})$ is an element of $(\mathbf{N}^+ \cup \{\infty\})^{m-1}$.

EXAMPLE 7.4.1. With # representing a 'don't-care-symbol' and the subscript denoting a strict upper bound on the length of the associated gap, a typical pattern may look like

$$\texttt{ab}\#_2\texttt{r}\#\texttt{ac}\#\texttt{a}\#\texttt{d}\#_4\texttt{a}\#\texttt{br}\#\texttt{a} \tag{7.4.1}$$

where $\# = \#_\infty$ and $\#_1$ is omitted; That is '`ab`' should occur first contiguously, followed by '`r`' with a gap of < 2 symbols, followed anywhere later in the text by '`ac`', etc.

The case when all the d_j's are infinite is called the (fully) *unconstrained problem*. When all the d_j's are finite, then we speak of the (fully) *constrained* problem. In particular, the case where all d_j are equal to one reduces to the exact string matching problem. Furthermore, observe that when all $d_j < \infty$ (fully constrained pattern), the problem can be treated as the generalized string matching discussed in Section 7.3. In this case, the general pattern $\mathcal{W}$ is a set consisting of all words satisfying the constraint $\mathcal{D}$. However, if at least one d_j is infinite, then the techniques discussed so far are not well suited to handle it. Therefore, in this section, we develop new methods that make the analysis possible.

If an m-tuple $I = (i_1, i_2, \dots, i_m)$ $(1 \le i_1 < i_2 < \cdots < i_m)$ satisfies the constraint $\mathcal{D}$ with $i_{j+1} - i_j \le d_j$, then it is called a *position tuple*. Let $\mathcal{P}_n(\mathcal{D})$ be the set of all positions subject to the separation constraint $\mathcal{D}$, satisfying furthermore $i_m \le n$. Let also $\mathcal{P}(\mathcal{D}) = \bigcup_n \mathcal{P}_n(\mathcal{D})$. An *occurrence* of pattern $\mathcal{W}$ subject to the constraint $\mathcal{D}$ is a pair (I, X) formed with a position $I = (i_1, i_2, \dots, i_m)$ of $\mathcal{P}_n(\mathcal{D})$ and a text $X = x_1 x_2 \cdots x_n$ for which $x_{i_1} = w_1, x_{i_2} = w_2, \dots, x_{i_m} = w_m$. Thus, what we call an occurrence is a text augmented with the distinguished positions at which the pattern occurs. The number Ω of occurrences of pattern

$\mathcal{W}$ in text X as a subsequence subject to the constraint $\mathcal{D}$ is then a sum of characteristic variables

$$\Omega(X) = \sum_{I \in \mathcal{P}_{|X|}(\mathcal{D})} Z_I(X), \tag{7.4.2}$$

where $Z_I(X) := [\![\mathcal{W}$ occurs at position I in $X]\!]$. When the text X is of length n, then we often write $\Omega_n := \Omega(X)$.

In order to proceed we need to introduce important notion of *blocks* and *aggregates*. In the general case, we assume that the subset $\mathcal{F}$ of indices j for which d_j is finite ($d_j < \infty$) has cardinality $m - b$ with $1 \leq b \leq m$. The two extreme values of b, namely, $b = m$ and $b = 1$, describe the (fully) unconstrained and the (fully) constrained problem, respectively. Thus, the subset $\mathcal{U}$ of indices j for which d_j is unbounded ($d_j = \infty$) has cardinality $b - 1$. It then separates the pattern $\mathcal{W}$ into b independent subpatterns that are called the *blocks* and are denoted by $\mathcal{W}_1, \mathcal{W}_2, \ldots \mathcal{W}_b$. All the possible d_j "inside" any $\mathcal{W}_r$ are finite and form the subconstraint $\mathcal{D}_r$, so that a general hidden pattern specification $(\mathcal{W}, \mathcal{D})$ is equivalently described as a b-tuple of fully constrained hidden patterns $((\mathcal{W}_1, \mathcal{D}_1), (\mathcal{W}_2, \mathcal{D}_2), \ldots, (\mathcal{W}_b, \mathcal{D}_b))$.

Example 7.4.1 (*continued*). Consider again

$$\mathtt{ab}\#_2\mathtt{r}\#\mathtt{ac}\#\mathtt{a}\#\mathtt{d}\#_4\mathtt{a}\#\mathtt{br}\#\mathtt{a},$$

in which one has $b = 6$, the six blocks being

$$\mathcal{W}_1 = \mathtt{a}\#_1\mathtt{b}\#_2\mathtt{r},\ \mathcal{W}_2 = \mathtt{a}\#_1\mathtt{c},\ \ \mathcal{W}_3 = \mathtt{a},\ \mathcal{W}_4 = \mathtt{d}\#_4\mathtt{a},\ \ \mathcal{W}_5 = \mathtt{b}\#_1\mathtt{r},\ \ \mathcal{W}_6 = \mathtt{a}.$$

In the same way, an occurrence position $I = (i_1, i_2, \ldots, i_m)$ of $\mathcal{W}$ subject to constraint $\mathcal{D}$ gives rise to b suboccurrences, $I^{[1]}, I^{[2]}, \ldots I^{[b]}$, the rth term $I^{[r]}$ representing an occurrence of $\mathcal{W}_r$ subject to constraint $\mathcal{D}_r$. The rth *block* $B^{[r]}$ is the closed segment whose end points are the extremal elements of $I^{[r]}$, and the *aggregate* of position I, denoted by $\alpha(I)$, is the collection of these b blocks.

Example 7.4.1 (*continued*). Taking the pattern of Example 7.4.1, the position tuple

$$I = (6, 7, 9, 18, 19, 22, 30, 33, 50, 51, 60)$$

satisfies the constraint $\mathcal{D}$ and gives rise to six subpositions,

$$\overbrace{(6,7,9)}^{I^{[1]}},\quad \overbrace{(18,19)}^{I^{[2]}},\quad \overbrace{(22)}^{I^{[3]}},\quad \overbrace{(30,33)}^{I^{[4]}},\quad \overbrace{(50,51)}^{I^{[5]}},\quad \overbrace{(60)}^{I^{[6]}};$$

accordingly, the resulting aggregate $\alpha(I)$,

$$\overbrace{[6,9]}^{B^{[1]}},\quad \overbrace{[18,19]}^{B^{[2]}},\quad \overbrace{[22]}^{B^{[3]}},\quad \overbrace{[30,33]}^{B^{[4]}},\quad \overbrace{[50,51]}^{B^{[5]}},\quad \overbrace{[60]}^{B^{[6]}},$$

is formed with six blocks.

7.4.1. Mean and variance analysis

Hereafter, we assume that $\mathcal{W}$ is given and the text X is generated by a (non-degenerate) *memoryless* source. The first moment of the number of occurrences, $\Omega(X)$, is easily obtained by describing the collection of all occurrences in terms of formal languages, as already discussed in previous sections. We consider the collection of position-text pairs

$$\mathcal{O} := \{(I, X) \;\; ; \;\; I \in \mathcal{P}_{|X|}(\mathcal{D})\},$$

with the size of an element being by definition the length n of the text X. The weight of an element of $\mathcal{O}$ is taken to be equal to $Z_I(X)P(X)$, where $P(X)$ is the probability of the text. In this way, $\mathcal{O}$ can also be regarded as the collection of all occurrences weighted by probabilities of the text. The corresponding generating function of $\mathcal{O}$ equipped with this weight is

$$O(z) = \sum_{(I,X)\in\mathcal{O}} Z_I(X)P(X)\, z^{|X|} = \sum_X \left(\sum_{I\in\mathcal{P}_{|X|}(\mathcal{D})} Z_I(X) \right) P(X)z^{|X|}, \quad (7.4.3)$$

and, with the definition of Ω,

$$O(z) = \sum_X \Omega(X)P(X)\, z^{|X|} \; = \sum_n \mathbf{E}(\Omega_n)z^n. \quad (7.4.4)$$

As a consequence, one has $[z^n]O(z) = \mathbf{E}(\Omega_n)$, so that $O(z)$ serves as the generating function of the sequence of expectations $\mathbf{E}(\Omega_n)$.

On the other hand, each occurrence can be viewed as a "context" with an initial string, then the first letter of the pattern, then a separating string, then the second letter, etc. The collection $\mathcal{O}$ is therefore described combinatorially by

$$\mathcal{O} \; = \mathcal{A}^\star \times \{w_1\} \times \mathcal{A}^{<d_1} \times \{w_2\} \times \mathcal{A}^{<d_2} \times \ldots \times \{w_{m-1}\} \times \mathcal{A}^{<d_{m-1}} \times \{w_m\} \times \mathcal{A}^\star. \quad (7.4.5)$$

There, for $d < \infty$, $\mathcal{A}^{<d}$ denotes the collection of all words of length strictly less d, i.e., $\mathcal{A}^{<d} := \bigcup_{i<d} \mathcal{A}^i$, whereas, for $d = \infty$, $\mathcal{A}^{<\infty}$ denotes the collection of all finite words, i.e., $\mathcal{A}^{<\infty} := \mathcal{A}^\star = \bigcup_{i<\infty} \mathcal{A}^i$. Since the source is memoryless, the rules discussed at the end of the last section can be applied, and they give access to $O(z)$ from the description (7.4.5). The generating function functions associated to $\mathcal{A}^{<d}$ and $\mathcal{A}^{<\infty}$ are

$$A^{<d}(z) = 1+z+z^2+\cdots+z^{d-1} = \frac{1-z^d}{1-z}, \qquad A^{<\infty}(z) = 1+z+z^2+\cdots = \frac{1}{1-z}.$$

Thus, the description (7.4.5) of occurrences automatically translates into

$$O(z) \equiv \sum_{n\ge 0} \mathrm{E}[\Omega_n]\; z^n = \left(\frac{1}{1-z}\right)^{b+1} \times \left(\prod_{i=1}^m p_{w_i} z\right) \times \left(\prod_{i\in\mathcal{F}} \frac{1-z^{d_i}}{1-z}\right). \quad (7.4.6)$$

One finally finds

$$\mathbf{E}(\Omega_n) = [z^n]O(z) = \frac{n^b}{b!}\left(\prod_{i\in\mathcal{F}} d_i\right) P(\mathcal{W})\left(1 + O\left(\frac{1}{n}\right)\right), \tag{7.4.7}$$

and a complete asymptotic expansion could be easily obtained.

For the analysis of variance and especially of higher moments, it is essential to work with a centered random variable Ξ defined, for each n, as

$$\Xi_n := \Omega_n - \mathbf{E}(\Omega_n) = \sum_{I\in\mathcal{P}_n(\mathcal{D})} Y_I, \tag{7.4.8}$$

where $Y_I := Z_I - \mathbf{E}(Z_I) = Z_I - P(\mathcal{W})$. The second moment of the centered variable Ξ equals the variance of Ω_n and with the centered variables defined above by (7.4.8), one has

$$\mathbf{E}(\Xi_n^2) = \sum_{I,J\in\mathcal{P}_n(\mathcal{D})} \mathbf{E}(Y_I Y_J). \tag{7.4.9}$$

From this last equation, we need to analyze *pairs of positions* $(I, X), (J, X) = (I, J, X)$ relative to a common text X. We denote by $\mathcal{O}_2$ this set, that is,

$$\mathcal{O}_2 := \{(I, J, X) \ \ ; \ \ I, J \in \mathcal{P}_{|X|}(\mathcal{D})\},$$

and we weight each element (I, J, X) by $Y_I(X)Y_J(X)P(X)$. The corresponding generating function, which enumerates pairs of occurrences, is

$$\begin{aligned} O_2(z) &:= \sum_{(I,J,X)\in\mathcal{O}_2} Y_I(X)Y_J(X)P(X)\, z^{|X|} \\ &= \sum_X \left(\sum_{I,J\in\mathcal{P}_{|X|}(\mathcal{D})} Y_I(X)Y_J(X)\right) P(X)z^{|X|} \end{aligned}$$

and, with (7.4.9),

$$O_2(z) = \sum_{n\geq 0}\ \sum_{I,J\in\mathcal{P}_n(\mathcal{D})} \mathbf{E}(Y_I Y_J)\, z^n = \sum_{n\geq 0} \mathbf{E}(\Xi_n^2)\, z^n.$$

The process entirely parallels the derivation of (7.4.3) and (7.4.4), and, one has $[z^n]O_2(z) = \mathbf{E}(\Xi_n^2)$, so that $O_2(z)$ serves as the generating function (in the usual sense) of the sequence of moments $\mathbf{E}(\Xi_n^2)$.

There are two kinds of pairs (I, J) depending whether they intersect or not. When I and J do not intersect, the corresponding random variables Y_I and Y_J are independent, and the corresponding covariance $E[Y_I Y_J]$ reduces to 0. As a consequence, one may restrict attention to pairs of occurrences I, J that intersect at one place at least. Suppose that there exist two occurrences of pattern $\mathcal{W}$ at positions I and J which intersect at ℓ distinct places. We then

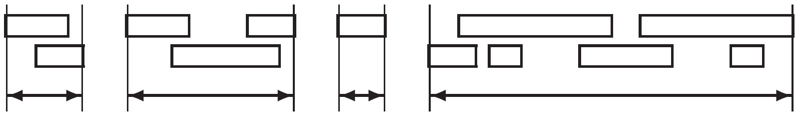


Figure 7.3. A pair of position tuples I, J with $b = 6$ blocks each and the joint aggregates; the number of degrees of freedom is here $\beta(I, J) = 4$.

denote by $\mathcal{W}_{I\cap J}$ the subpattern of $\mathcal{W}$ that occurs at position $I \cap J$, and by $P(\mathcal{W}_{I\cap J})$ the probability of this subpattern. Since the expectation $\mathbf{E}(Z_I Z_J)$ equals $P(\mathcal{W})^2/P(\mathcal{W}_{I\cap J})$ provided that $\mathcal{W}$ agrees on every position of $I \cap J$, the expectation $\mathbf{E}(Y_I Y_J) = P(\mathcal{W})^2 e(I, J)$ involves a *correlation number* $e(I, J)$

$$e(I, J) = \frac{[\![\mathcal{W} \text{ agree } I \cap J]\!]}{P(\mathcal{W}_{I\cap J})} - 1. \tag{7.4.10}$$

Remark that this relation remains true even if the pair (I, J) is not intersecting, since, in this case, one has $P(\mathcal{W}_{I\cap J}) = P(\varepsilon) = 1$.

The asymptotic behavior of variance is driven by the overlapping of blocks involved in I and J, rather than plainly by the cardinality of $I \cap J$. In order to formalize this, define first the (joint) *aggregate* $\alpha(I, J)$ to be the system of blocks obtained by merging together all intersecting blocks of the two aggregates $\alpha(I)$ and $\alpha(J)$. The number of blocks $\beta(I, J)$ of $\alpha(I, J)$ plays a fundamental rôle here, since it measures the *degree of freedom* of pairs; we also call $\beta(I, J)$ the degree of pair (I, J). Figure 7.3 illustrates graphically this notion.

Example 7.4.2. Consider the pattern $\mathcal{W} = \boxed{\texttt{a}\#_3\texttt{b}\#_4\texttt{r}}\,\#\,\boxed{\texttt{a}\#_4\texttt{c}}$ composed of two blocks. Then the text `aarbarbccaracc` contains several valid occurrences of $\mathcal{W}$ including two at positions $I = (2, 4, 6, 10, 13)$ and $J = (5, 7, 11, 12, 13)$. The individual aggregates are $\alpha(I) = \{[2, 6], [10, 13]\}, \alpha(J) = \{[5, 11], [12, 13]\}$ so that the joint quantities are: $\alpha(I, J) = [2, 13]$ and $\beta(I, J) = 1$. This pair has exactly degree 1.

When I and J intersect, there exists at least one block of $\alpha(I)$ that intersects a block of $\alpha(J)$, so that the degree $\beta(I, J)$ is at most equal to $2b - 1$. Next, we partition $\mathcal{O}_2$ according to the value of $\beta(I, J)$ and write

$$\mathcal{O}_2^{[p]} := \{(I, J, X) \in \mathcal{O}_2 \ \ ; \ \ \beta(I, J) = 2b - p\}$$

for the collection of intersecting pairs (I, J, X) of occurrences for which the degree of freedom equals $2b - p$. From the preceding discussion, only $p \geq 1$ needs to be considered and

$$O_2(z) = O_2^{[1]}(z) + O_2^{[2]}(z) + O_2^{[3]}(z) + \cdots + O_2^{[2b]}(z).$$

As we see next, it is only the first term of this sum that matters asymptotically.

In order to conclude the discussion, we need the notion of *full pairs*: a pair (I,J) of $\mathcal{P}_q(\mathcal{D}) \times \mathcal{P}_q(\mathcal{D})$ is *full* if the joint aggregate $\alpha(I,J)$ completely covers the interval $[1,q]$; see Figure 7.4. (Clearly, the possible values of length q are finite, since q is at most equal to 2ℓ, where ℓ is the length of the constraint $\mathcal{D}$.)

EXAMPLE 7.4.3. Consider the pattern $\mathcal{W} = \mathtt{a}\#_3\mathtt{b}\#_4\mathtt{r}\#\mathtt{a}\#_4\mathtt{c}$. The text `aarbarbccaracc` also contains two other occurrences of $\mathcal{W}$, at positions $I' = (1,4,6,12,13)$ and $J' = (5,7,11,12,14)$. Now, I' and J' are intersecting, and the aggregates are $\alpha(I') = \{[1,6],[12,13]\}, \alpha(J') = \{[5,11],[12,14]\}$ so that $\alpha(I',J') = \{[1,11],[12,14]$. We have here an example of a full pair of occurrences with a number of blocks $\beta(I',J') = 2$.

There is a fundamental translation invariance due to the independence of symbols in the Bernoulli model that entails a combinatorial isomorphism ($\cong$ represents combinatorial isomorphism)

$$\mathcal{O}_2^{[p]} \cong (\mathcal{A}^\star)^{2b-p+1} \times \mathcal{B}_2^{[p]},$$

where $\mathcal{B}_2^{[p]}$ is the subset of $\mathcal{O}_2$ formed of full p airs such that $\beta(I,J)$ equals $2b-p$. In essence, the gaps can be all grouped together (their number is $2b-p+1$, which is translated by the prefactor $(\mathcal{A}^\star)^{2b-p+1}$), while what remains constitutes a full occurrence. The generating function of $\mathcal{O}_2^{[p]}$ is accordingly

$$O_2^{[p]}(z) = \left(\frac{1}{1-z}\right)^{2b-p+1} \times B_2^{[p]}(z)$$

where $B_2^{[p]}(z)$ is the generating function of the collection $\mathcal{B}_2^{[p]}$. From our earlier discussion, it is a *polynomial*. Now, an easy dominant pole analysis entails that $[z^n]O_2^{[p]} = O(n^{2b-p})$. This proves that the dominant contribution to the variance is given by $[z^n]O_2^{[1]}$, which is of order $O(n^{2b-1})$.

The variance $\mathbf{E}(\Xi_n^2)$ involves the constant $B_2^{[1]}(1)$ that is the total weight of the collection $\mathcal{B}_2^{[1]}$. Recall that this collection is formed of intersecting full pairs of occurrences of degree $2b-1$. The polynomial $B_2^{[1]}(z)$ is itself the generating function of the collection $\mathcal{B}_2^{[1]}$, and it is conceptually an extension of Guibas and Odlyzko's autocorrelation polynomial. We shall later make precise the relation between both polynomials.

We summarize our findings in the following theorem.

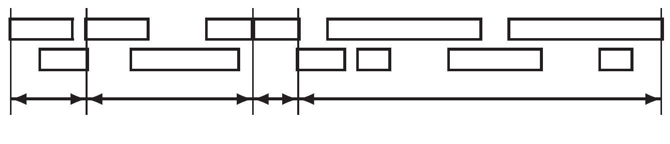


Figure 7.4. A full pair of position tuples I, J with $b = 6$ blocks each.

THEOREM 7.4.4. *Consider a general constraint $\mathcal{D}$ with a number of blocks equal to b. The mean and the variance of the number of occurrences Ω_n of a pattern $\mathcal{W}$ subject to constraint $\mathcal{D}$ satisfy*

$$\mathbf{E}(\Omega_n) = \frac{P(\mathcal{W})}{b!}\left(\prod_{j\,:\,d_j<\infty} d_j\right) n^b\left(1+O(n^{-1})\right),$$
$$\mathrm{Var}(\Omega_n) = \sigma^2(\mathcal{W})n^{2b-1}\left(1+O(n^{-1})\right),$$

where the "variance coefficient" $\sigma^2(\mathcal{W})$ involves the autocorrelation $\kappa(\mathcal{W})$

$$\sigma^2(\mathcal{W}) = \frac{P^2(\mathcal{W})}{(2b-1)!}\,\kappa^2(\mathcal{W}) \qquad \text{with} \quad \kappa^2(\mathcal{W}) := \sum_{(I,J)\in\mathcal{B}_2^{[1]}} e(I,J) \tag{7.4.11}$$

The set $\mathcal{B}_2^{[1]}$ is the collection of all pairs of position tuple (I,J) that satisfy three conditions: (i) they are full; (ii) they are intersecting; (iii) there is a single pair (r,s) with $1 \le r,s \le b$ for which the rth *block $B^{[r]}$ of $\alpha(I)$ and the* sth *block $C^{[s]}$ of $\alpha(J)$ intersect.*

The computation of the autocorrelation $\kappa(\mathcal{W})$ reduces to b^2 computations of correlations $\kappa(\mathcal{W}_r,\mathcal{W}_s)$, relative to pairs $(\mathcal{W}_r,\mathcal{W}_s)$ of blocks. Note that each correlation of the form $\kappa(\mathcal{W}_r,\mathcal{W}_s)$ involves a totally constrained problem and is discussed below. Let $D(\mathcal{D}) := \prod_{i:\ d_i<\infty} d_i$. Then, one has

$$\kappa^2(\mathcal{W}) = D^2(\mathcal{D}) \sum_{1\le r,s\le b} \frac{1}{D(\mathcal{D}_r)D(\mathcal{D}_s)}\binom{r+s-2}{r-1}\binom{2b-r-s}{b-r}\kappa(\mathcal{W}_r,\mathcal{W}_s), \tag{7.4.12}$$

where $\kappa(\mathcal{W}_r,\mathcal{W}_s)$ is the sum of the $e(I,J)$ taken over all full intersecting pairs (I,J) formed with an position tuple I of block $\mathcal{W}_r$ subject to constraint $\mathcal{D}_r$ and an position tuple J of block $\mathcal{W}_s$ subject to constraint $\mathcal{D}_s$. Let us explain the formula (7.4.12) in words: for a pair (I,J) of the set $\mathcal{B}_2^{[1]}$, there is a single pair (r,s) of indices with $1 \le r,s \le b$ for which the rth block $B^{[r]}$ of $\alpha(I)$ and the sth block $C^{[s]}$ of $\alpha(J)$ intersect. Then, there exist $r+s-2$ blocks before the block $\alpha(B^{[r]},C^{[s]})$ and $2b-r-s$ blocks after it. We then have three different degrees of freedom: (i) the relative order of blocks $B^{[i]}(i<r)$ and blocks $C^{[j]}(j<s)$, and similarly the relative order of blocks $B^{[i]}(i>r)$ and blocks $C^{[j]}(j>s)$; (ii) the lengths of the blocks (there are D_j possible lengths for the jth block); (iii) finally the relative positions of the blocks $B^{[r]}$ and $C^{[s]}$.

In particular, in the unconstrained case, the parameter b equals m, and each block $\mathcal{W}_r$ is reduced to the symbol w_r. Then the "correlation coefficient" $\kappa^2(\mathcal{W})$ simplifies to

$$\kappa^2(\mathcal{W}) := \sum_{1\le r,s\le m}\binom{r+s-2}{r-1}\binom{2m-r-s}{m-r}[\![w_r=w_s]\!]\left(\frac{1}{p_{w_r}}-1\right). \tag{7.4.13}$$

In words, once you fix the position of the intersection, called pivot, then amongst the $r+s-2$ elements smaller than the pivot one assigns freely $r-1$ to the first occurrence and the remaining $s-1$ to the second. One proceeds similarly for the $2m-r-s$ elements larger than the pivot.

7.4.2. Autocorrelation polynomial revisited

Finally, we compare the autocorrelation coefficient $\kappa(\mathcal{W})$ with the autocorrelation polynomial $S_w(z)$ introduced in the last section for the exact string matching problem. Let now $w = w_1 w_2 \dots w_m$ be again a string of length m, and all the symbols of w must occur at consecutive places, so that a valid position I is an interval of length m. We recall that the autocorrelation set $\mathcal{P}(w) \subset [1..m]$ involves all indices k such that the prefix w_1^k coincides with the suffix w_{m-k+1}^m. Here, an index $k \in \mathcal{P}(w)$ is relative to a intersecting pair of positions (I, J) and one has $w_1^k = w_{I \cap J}$.

In the previous section, we introduced the autocorrelation polynomial $S_w(z)$ as

$$S_w(z) = \sum_{k \in \mathcal{P}_w} P(w_{k+1}^m) z^{m-k} = P(w) \sum_{k \in \mathcal{P}(w)} \frac{1}{P(w_1^k)} z^{m-k}.$$

We also define

$$C_w(z) = \sum_{k \in \mathcal{P}(w)} z^{m-k}.$$

Since the polynomial $B_2^{[1]}$ involves coefficients $e(I, J)$ this polynomial can be written as function of the two autocorrelations polynomials A_w and C_w,

$$B_2^{[1]}(z) = P(w) z^m \, [A_w(z) - P(w)\, C_w(z)].$$

Put simply, the variance coefficient of the hidden pattern problem extends the classical autocorrelation quantities associated with strings.

7.4.3. Central limit laws

Our goal is to prove that the sequence Ω_n appropriately centered and scaled tends to the normal distribution. We consider the following standardized random variable $\tilde{\Xi}_n$ which is defined for each n by

$$\tilde{\Xi}_n := \frac{\Xi_n}{n^{b-1/2}} = \frac{\Omega_n - \mathbf{E}(\Omega_n)}{n^{b-1/2}}, \tag{7.4.14}$$

where b is the number of blocks of the constraint $\mathcal{D}$. We shall show that $\tilde{\Xi}_n$ behaves asymptotically as a normal variable with mean 0 and standard deviation σ. By the classical *moment convergence theorem* this is established once all moments of $\tilde{\Xi}_n$ are known to converge to the appropriate moments of the standard normal distribution.

We remind the reader that if G is a standard normal variable (i.e., a Gaussian distributed variable with mean 0 and standard deviation 1), then for any integral $s \geq 0$

$$\mathbf{E}(G^{2s}) = 1 \cdot 3 \cdots (2s-1), \qquad \mathbf{E}(G^{2s+1}) = 0. \tag{7.4.15}$$

We shall accordingly distinguish two cases based on the parity of r, $r = 2s$ and $r = 2s + 1$, and prove that

$$\mathrm{E}[\Xi_n^{2s+1}] = o(n^{(2s+1)(b-1/2)}), \qquad \mathbf{E}(\Xi_n^{2s}) \sim \sigma^{2s}\,(1 \cdot 3 \cdots (2s-1))\,n^{2sb-s}, \tag{7.4.16}$$

which implies Gaussian convergence of $\tilde{\Xi}_n$.

Theorem 7.4.5. *The random variable Ω_n over a random text of length n generated by a memoryless source asymptotically obeys a* Central Limit Law *in the sense that its distribution is asymptotically normal: for all $x = O(1)$, one has*

$$\lim_{n\to\infty} \mathbf{P}\left(\frac{\Omega_n - \mathbf{E}(\Omega_n)}{\sqrt{\mathrm{Var}(\Omega_n)}} \le x\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,dt. \tag{7.4.17}$$

Proof. The proof below is combinatorial; it basically reduces to grouping and enumerating adequately the various combinations of indices in the sum that expresses $\mathbf{E}(\Xi_n^r)$. Once more, $\mathcal{P}_n(\mathcal{D})$ is formed of all the sets of positions in $[1, n]$ subject to the constraint $\mathcal{D}$ and we set $\mathcal{P}(\mathcal{D}) := \bigcup_n \mathcal{P}_n(\mathcal{D})$. Then totally distributing the terms in Ξ^r yields

$$\mathbf{E}(\Xi_n^r) = \sum_{(I_1,\dots,I_r)\in\mathcal{P}_n^r(\mathcal{D})} \mathbf{E}(Y_{I_1}\cdots Y_{I_r}). \tag{7.4.18}$$

An r-tuple of sets $(I_1, \dots, I_r)$ in $\mathcal{P}^r(\mathcal{D})$ is said to be *friendly* if each I_k intersects at least one other I_ℓ, with $\ell \neq k$ and we let $\mathcal{Q}^{(r)}(\mathcal{D})$ be the set of all friendly collections in $\mathcal{P}^r(\mathcal{D})$. For $\mathcal{P}^r$, $\mathcal{Q}^{(r)}$, and their derivatives below, we add the subscript n each time the situation is particularized to texts of length n. If $(I_1, \dots, I_r)$ does not lie in $\mathcal{Q}^{(r)}(\mathcal{D})$, then $\mathbf{E}(Y_{I_1}\cdots Y_{I_r}) = 0$, since at least one of the Y_I's is independent of the other factors in the product and the Y_I's have been centered, $\mathbf{E}(Y_I) = 0$. One can thus restrict attention to friendly families and get the basic formula

$$\mathbf{E}(\Xi_n^r) = \sum_{(I_1,\dots,I_r)\in\mathcal{Q}_n^{(r)}(\mathcal{D})} \mathbf{E}(Y_{I_1}\cdots Y_{I_r}), \tag{7.4.19}$$

where the expression involves fewer terms than in (7.4.18). From there, we proceed in two stages. First, restrict attention to friendly families that give rise to the dominant contribution and introduce a suitable subfamily $\mathcal{Q}_\star^{(r)} \subset \mathcal{Q}^{(r)}$; in so doing, moments of odd order appear to be negligible. Next, for even order r, the family $\mathcal{Q}_\star^{(r)}$ involves a symmetry and it suffices to consider another smaller subfamily $\mathcal{Q}_{\star\star}^{(r)} \subset \mathcal{Q}_\star^{(r)}$ that corresponds to a "standard" form of position tuple intersection; this last reduction precisely gives rise to the even Gaussian moments.

Odd moments. Given $(I_1, \dots, I_r) \in \mathcal{Q}^{(r)}$, the aggregate $\alpha(I_1, I_2, \dots, I_r)$ is defined as the aggregation (in the sense of the variance calculation above) of $\alpha(I_1) \cup \cdots \cup \alpha(I_r)$. Next, the *number of blocks* of $(I_1, \dots, I_r)$ is the number of blocks of the aggregate $\alpha(I_1, \ldots, I_r)$; if p is the total number of intersecting blocks of the aggregate $\alpha(I_1, \ldots, I_r)$, the aggregate $\alpha(I_1, I_2, \ldots I_r)$ has $rb - p$ blocks. Like previously, we say that the family $(I_1, \ldots, I_r)$ of $\mathcal{Q}_q^{(r)}$ is *full* if the aggregate $\alpha(I_1, I_2, \ldots I_r)$ completely covers the interval $[1, q]$. In this case, the length of the aggregate is at most $rd(m-1)+1$, and the generating function of full families is a polynomial $P_r(z)$ of degree at most $rd(m-1)+1$ with $d = \max_{j \in \mathcal{F}} d_j$. Then, the generating function of families of $\mathcal{Q}^{(r)}$ whose block number equals k is of the form

$$\left(\frac{1}{1-z}\right)^{k+1} \times P_r(z),$$

so that the number of families of $\mathcal{Q}_n^{(r)}$ whose block number equals k is $O(n^k)$. This observation proves that the dominant contribution to (7.4.19) arises from friendly families with a maximal block number. It is clear that the minimum number of intersecting blocks of any element of $\mathcal{Q}^{(r)}$ equals $\lceil r/2 \rceil$, since it coincides exactly with the minimum number of edges of a graph with r vertices which contains no isolated vertex. Then the maximum block number of a f friendly family equals $rb - \lceil r/2 \rceil$. In view of this fact and the remarks above regarding cardinalities, we immediately have

$$\mathrm{E}\left[\Xi_n^{2s+1}\right] = O\left(n^{(2s+1)b-s-1}\right) = o\left(n^{(2s+1)(b-1/2)}\right)$$

which establishes the limit form of odd moments in (7.4.16).

Even Moments. We are thus left with estimating the even moments. The dominant term is relative to friendly families of $\mathcal{Q}^{(2s)}$ with an intersecting block number equal to s, whose set we denote by $\mathcal{Q}_\star^{(2s)}$. In such a family, each subset I_k intersects one and only one other subset I_ℓ. Furthermore, if the blocks of $\alpha(I_h)$ are denoted by $B_h^{[u]}, 1 \le u \le b$, there exists only one block $B_k^{[u_k]}$ of $\alpha(I_k)$ and only one block $B_\ell^{[u_\ell]}$ that contains the points of $I_k \cap I_\ell$. This defines an involution τ such that $\tau(k) = \ell$ and $\tau(\ell) = k$ for all pairs of indices (ℓ, k) for which I_k and I_ℓ intersect. Furthermore, given the symmetry relation $\mathbf{E}(Y_{I_1} \cdots Y_{I_{2s}}) = \mathbf{E}(Y_{I_{\rho(1)}} \cdots Y_{I_{\rho(2s)}})$ it suffices to restrict attention to friendly families of $\mathcal{Q}_\star^{(2s)}$ for which the involution τ is the standard one with cycles $(1,2)$, $(3,4)$, etc; for such "standard" families whose set is denoted by $\mathcal{Q}_{\star\star}^{(2s)}$, the pairs that intersect are thus (I_1, I_2), …, (I_{2s-1}, I_{2s}). Since the set $\mathcal{K}_{2s}$ of involutions of $2s$ elements has cardinality $K_{2s} = 1 \cdot 3 \cdot 5 \cdots (2s-1)$, the equality

$$\sum_{\mathcal{Q}_{\star n}^{(2s)}} \mathbf{E}(Y_{I_1} \cdots Y_{I_{2s}}) = K_{2s} \sum_{\mathcal{Q}_{\star\star n}^{(2s)}} \mathbf{E}(Y_{I_1} \cdots Y_{I_{2s}}), \tag{7.4.20}$$

entails that we can work now solely with standard families.

The class of position tuples relative to standard families is $\mathcal{A}^\star \times (\mathcal{A}^\star)^{2sb-s-1} \times \mathcal{B}_{2s}^{[s]} \times \mathcal{A}^\star$; this class involves the collection $\mathcal{B}_{2s}^{[s]}$ of all full friendly $2s$-tuples of position tuples with a number of blocks equal to s. Since $\mathcal{B}_{2s}^{[s]}$ is exactly a shuffle of s copies of $\mathcal{B}_2^{[1]}$ (as introduced in the study of the variance), the associated generating function is

$$\left(\frac{1}{1-z}\right)^{2sb-s+1} \quad (2sb-s)! \left(\frac{B_2^{[1]}(z)}{(2b-1)!}\right)^s,$$

where $B_2^{[1]}(z)$ is the already introduced autocorrelation polynomial. Upon taking coefficients, we obtain the estimate

$$\sum_{\mathcal{Q}_{\star\star n}^{(2s)}} \mathbf{E}(Y_{I_1} \cdots Y_{I_{2s}}) \sim n^{(2b-1)s}\sigma^{2s}. \tag{7.4.21}$$

In view of the formulæ (7.4.18), (7.4.19), (7.4.20), and (7.4.21) above, this yields the estimate of even moments and leads to the second relation of (7.4.16). This completes the proof of Theorem 7.4.5. ∎

The even Gaussian moments eventually come out as the number of involutions, which corresponds to a fundamental asymptotic symmetry present in the problem. In this perspective the specialization of the proof to the fully unconstrained case is reminiscent of the derivation of the usual central limit theorem (dealing with sums of independent variables) by moments methods.

7.4.4. Limit laws for fully constrained pattern

In this section, we strengthen our results for *fully constrained pattern* in which all gaps d_j are finite. We set $D = \prod_j d_j$, and $\ell = \sum_j d_j$. Observe that in this case, we can reduced the subsequence problem to a generalized string matching problem with the generalized pattern $\mathcal{W}$ consisting of all words that satisfy $(\mathcal{W}, \mathcal{D})$. Thus our previous results apply, in particular, Theorems 7.3.8 and 7.3.10. This leads to the following result.

THEOREM 7.4.6. *Consider a fully constrained pattern with mean and variance found in Theorem 7.4.4 for $b = 1$.*

(i) *The random variable Ω_n satisfies a* Central Limit Law *with speed of convergence $1/\sqrt{n}$:*

$$\sup_x \left| \mathbf{P}\left(\frac{\Omega_n - DP(\mathcal{W})n}{\sigma(\mathcal{W})\sqrt{n}} \le x\right) - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt \right| = O\left(\frac{1}{\sqrt{n}}\right). \tag{7.4.22}$$

(ii) Large deviations *from the mean value have exponentially small probability: there exist a constant $\eta > 0$ and a nonnegative function $I(x)$ defined throughout $(0, \eta)$ such that $I(x) > 0$ for $x \ne DP(\mathcal{W})$ and*

$$\begin{cases} \lim_{n\to\infty} \frac{1}{n} \log \mathbf{P}\left(\frac{\Omega_n}{n} \le x\right) = -I(x) \text{ if } 0 < x < DP(\mathcal{W}) \\ \lim_{n\to\infty} \frac{1}{n} \log \mathbf{P}\left(\frac{\Omega_n}{n} \ge x\right) = -I(x) \text{ if } DP(\mathcal{W}) < x < \eta \end{cases}, \tag{7.4.23}$$

except for at most a finite number of exceptional values of x. More precisely,

$$I(x) = -\log \frac{\lambda(\zeta)}{\zeta^x} \qquad \textit{with } \zeta \equiv \zeta(x) \textit{ defined by } \quad \frac{\zeta\lambda'(\zeta)}{\lambda(\zeta)} = x \tag{7.4.24}$$

where $\lambda(u)$ is the largest eigenvalue of the matrix $\mathrm{T}(u)$ *of the associate de Bruijn graph constructed for $\mathcal{W} = \{v: \ v = w_1u_1w_2\cdots w_{m-1}u_{m-1}w_m$, where $u_i \in \mathcal{A}^{d_i-1},\ 1 \le i \le m-1\}$.*

(iii) *For primitive patterns (cf. Section 7.3.2)* a Local Limit Law *holds:*

$$\sup_k \left| \mathbf{P}\left(\Omega_n = k\right) - \frac{1}{\sigma(\mathcal{W})\sqrt{n}} \frac{e^{x(k)^2/2}}{\sqrt{2\pi}} \right| = o\left(\frac{1}{\sqrt{n}}\right), \tag{7.4.25}$$

where

$$x(k) = \frac{k - DP(\mathcal{W})n}{\sigma(\mathcal{W})\sqrt{n}}$$

for $n \to \infty$.

Example 7.4.7. We motivated our desire to study the subsequence problem by an example form computer security. Indeed, if one wants to detect "suspicious" activities (e.g., signatures viewed as subsequences in an audit file), it is important to set up a threshold in order to avoid false alarms. This problem can be rephrased as one of finding a threshold $\alpha_0 = \alpha_0(\mathcal{W}; n, \beta)$ such that

$$\mathbf{P}(\Omega_n > \alpha_0) \le \beta,$$

for small given β (say $\beta = 10^{-5}$). Based on frequencies of letters and the assumption that a memoryless model is (at least roughly) relevant, one can estimate the mean value and the standard deviation coefficients $P(\mathcal{W}), \sigma(\mathcal{W})$ as discussed above. The Gaussian limits granted by Theorems 7.4.5 and 7.3.8 then reduce the problem to solving an approximate system, which in the (fully) constrained case reads

$$\alpha_0 = nP(\mathcal{W}) + x_0\sigma(\mathcal{W})\sqrt{n}, \qquad \beta = \frac{1}{\sqrt{2\pi}} \int_{x_0}^{\infty} e^{-t^2/2}\, dt.$$

This system admits the approximate solution

$$\alpha_0 \approx n\pi(\omega) + \sigma(\mathcal{W})\sqrt{2n\log(1/\beta)}. \tag{7.4.26}$$

for small β.

7.5. Generalized subsequence problem

In the *generalized subsequence problem* the pattern is $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_d)$ where $\mathcal{W}_i$ is a set of strings (a language). We say that the generalized pattern $\mathcal{W}$

occurs in the text X if X contains $\mathcal{W}$ as a *subsequence* $(w_1, w_2, \ldots, w_d)$ where $w_i \in \mathcal{W}_i$. An occurrence of the pattern in X is a sequence

$$(u_0, w_1, u_1, \ldots, w_d, u_d)$$

such that $X = u_0 w_1 u_1 \cdots w_d u_d$. We shall study the associated language $\mathcal{L}$ that can be described as

$$\mathcal{L} = \mathcal{A}^* \cdot \mathcal{W}_1 \cdot \mathcal{A}^* \cdots \mathcal{W}_d \cdot \mathcal{A}^*. \tag{7.5.1}$$

More precisely, an occurrence of $\mathcal{W}$ is a sequence of d disjoint intervals $I = (I_1, I_2, \ldots I_d)$ such that $I_j := [k_j^1, k_j^2]$ where $1 \le k_j^1 \le k_j^2 \le n$ is a portion of text X_1^n where $w_j \in \mathcal{W}_j$ occurs. We denote by $\mathcal{P}_n := \mathcal{P}_n(\mathcal{W})$ the set of all valid occurrences I. The number of occurrences Ω_n of $\mathcal{W}$ in the text X of size n is then

$$\Omega_n = \sum_{I \in \mathcal{P}_n(\mathcal{L})} Z_I, \tag{7.5.2}$$

where $Z_I(X) := [\![\mathcal{W} \text{ occurs at position } I \text{ in } X]\!]$.

In passing, we observe that the generalized subsequence problem is the most general pattern matching considered so far. It contains the exact string matching (cf. Section 7.2), generalized string matching (cf. Section 7.3), and the subsequence pattern matching known also as hidden patterns (cf. Section 7.4). In this section we present an analysis of the first two moments of Ω_n for the generalized subsequence pattern matching problem for dynamic sources discussed in Section 7.1.

7.5.1. Generating operators for dynamic sources

In Section 7.1 we have introduced a general probabilistic source known as a dynamic source. In this section we analyze the generalized subsequence model for such sources.

We start with a brief description of the methodology of generating operators that are used in the analysis of dynamic sources. We recall from Section 7.1 that the *generating operator* $\mathbf{G}_w$ is defined as $\mathbf{G}_w[f](t) := |h'_w(t)| f \circ h_w(t)$ for a density function f and a word w. In particular, in (7.1.2) we proved that $P(w) \int_0^1 f(t)dt = \int_0^1 \mathbf{G}_w[f](t)dt$ for any function $f(t)$, which implies that $P(w)$ is an eigenvalue of the operator $\mathbf{G}_w$. Furthermore, the generating operator for $w \cdot u$ is $\mathbf{G}_{w \cdot u} = \mathbf{G}_u \circ \mathbf{G}_w$, where w and u are words (cf. (7.1.3)) and $\circ$ is the composition of operators.

Consider now a language $\mathcal{B} \subset \mathcal{A}^*$. Its *generating operator* $\mathbf{B}(z)$ is then defined as

$$\mathbf{B}(z) := \sum_{w \in \mathcal{B}} z^{|w|} \, \mathbf{G}_w.$$

We observe that the ordinary generating function of a language $\mathcal{B}$ is related to the generating operators. Indeed,

$$B(z) := \sum_{w \in \mathcal{B}} z^{|w|} P(w) = \sum_{w \in \mathcal{B}} z^{|w|} \int_0^1 \mathbf{G}_w[f](t)dt = \int_0^1 \mathbf{B}(z)[f](t)dt. \tag{7.5.3}$$

If $\mathbf{B}(z)$ is well defined at $z = 1$, then $\mathbf{B}(1)$ is called the *normalized operator* of $\mathcal{B}$. In particular, using (7.1.1) we can compute

$$P(\mathcal{B}) = \sum_{w \in \mathcal{B}} P(w) = \int_0^1 \mathbf{B}(1) dt.$$

Furthermore, the operator

$$\mathbf{G} := \sum_{a \in \mathcal{A}} \mathbf{G}_a, \tag{7.5.4}$$

is the normalized operator of the alphabet $\mathcal{A}$ and plays a fundamental role in the analysis.

From the product formula (7.1.3) of the generating operators $\mathbf{G}_w$ we conclude that unions and Cartesian products of languages translates into sums and compositions of the associated operators. For instance, the operator associated with $\mathcal{A}^\star$ is

$$(I - z\mathbf{G})^{-1} := \sum_{i \geq 0} z^i \mathbf{G}^i.$$

where $\mathbf{G}^i = \mathbf{G} \circ \mathbf{G}^{i-1}$.

In order to proceed, we must restrict our attention to a class of dynamic sources called *decomposable* that satisfy two properties: (i) there exists a unique positive dominant eigenvalue λ and a dominant eigenvector denoted as φ (which is unique under the normalization $\int \varphi(t) dt = 1$); (ii) there is a spectral gap between the dominant eignevalue and other eignevalues. These properties entail the separation of the operator $\mathbf{G}$ into two parts

$$\mathbf{G} = \lambda \mathbf{P} + \mathbf{N} \tag{7.5.5}$$

such that the operator $\mathbf{P}$ is the projection relative to the dominant eigenvalue λ while $\mathbf{N}$ is the operator relative to the remainder of the spectrum (cf. Section 7.3). Furthermore (cf. Exercise 7.5.1)

$$\mathbf{P} \circ \mathbf{P} = \mathbf{P}, \tag{7.5.6}$$

$$\mathbf{P} \circ \mathbf{N} = \mathbf{N} \circ \mathbf{P} = 0. \tag{7.5.7}$$

The last property implies that for any $i \geq 1$

$$\mathbf{G}^i = \lambda^i \mathbf{P} + \mathbf{N}^i. \tag{7.5.8}$$

In particular, for the *density* operator $\mathbf{G}$ the dominant eigenvalue $\lambda = P(\mathcal{A}) = 1$ and φ is the unique stationary distribution. The function 1 is the left eigenvector. Then using (7.5.8) we arrive at

$$(I - z\mathbf{G})^{-1} = \frac{1}{1-z} \mathbf{P} + \mathbf{R}(z), \tag{7.5.9}$$

where

$$\mathbf{R}(z) := (I - z\mathbf{N})^{-1} - \mathbf{P} = \sum_{k \geq 0} z^k (\mathbf{G}^k - \mathbf{P}). \tag{7.5.10}$$

Observe that the first part of (7.5.9) has a pole at $z = 1$ and due to the spectral gap the operator $\mathbf{N}$ has spectral radius $\nu < \lambda = 1$. Furthermore, the operator $\mathbf{R}(z)$ is analytic in $|z| < (1/\nu)$ and again thanks to the existence of the spectral gap, the series $\mathbf{R}(1)$ is of geometric type. We shall point out below that the speed of convergence of $\mathbf{R}(z)$ is closely related to the decay of the correlation between two consecutive symbols. Finally, we list some additional properties of just introduced operators (cf. Exercise 7.5.2) true for any function $g(t)$ defined between 0 and 1.

$$\mathbf{N}[\varphi] = 0, \qquad \mathbf{P}[g](t) = \varphi(t) \int_0^1 g(t')dt' \tag{7.5.11}$$

$$\int_0^1 \mathbf{P}[g](t)dt = \int_0^1 g(t)dt, \qquad \int_0^1 \mathbf{N}[g](t)dt = 0, \tag{7.5.12}$$

where φ is the stationary density.

Theory built so far allows us, among others, to define precisely the correlation between languages in terms of the generating operators. From now on we restrict our analysis to the so called *nondense* languages $\mathcal{B}$ for which the associated generating operator $\mathbf{B}(z)$ is analytic in a disk $|z| > 1$. First, observe that for a nondense language $\mathcal{B}$, the normalized generating operator $\mathbf{B}$ satisfies

$$\int_0^1 \mathbf{P} \circ \mathbf{B} \circ \mathbf{P}[g](t) = P(\mathcal{B}) \left(\int_0^1 g(t)dt \right). \tag{7.5.13}$$

Let us now define the correlation coefficient between two languages, say $\mathcal{B}$ with the generating operator $\mathbf{B}$ and $\mathcal{C}$ with generating operator $\mathbf{C}$. Two types of correlations may occur between such languages. If $\mathcal{B}$ and $\mathcal{C}$ do not overlap, then $\mathcal{B}$ may be before $\mathcal{C}$, or after $\mathcal{C}$. We define the *correlation coefficient* $c(\mathcal{B}, \mathcal{C})$ (and in an analogous way $c(\mathcal{C}, \mathcal{B})$) as

$$\begin{aligned} P(\mathcal{B})P(\mathcal{C})c(\mathcal{B}, \mathcal{C}) &:= \sum_{k \geq 0} \left[P(\mathcal{B} \times \mathcal{A}^k \times \mathcal{C}) - P(\mathcal{B})P(\mathcal{C}) \right] \\ &= \int_0^1 \mathbf{C} \circ \mathbf{R}(1) \circ \mathbf{B}[\varphi](t). \end{aligned} \tag{7.5.14}$$

To see this we observe, using (7.5.5)–(7.5.13),

$$\begin{aligned} \int_0^1 \mathbf{C} \circ \mathbf{R}(1) \circ \mathbf{B}[\varphi](t)dt &= \int_0^1 \mathbf{C} \circ \left(\sum_{k \geq 0} (\mathbf{G}^k - \mathbf{P}) \right) \circ \mathbf{B}[\varphi](t)dt \\ &= \sum_{k \geq 0} \left(\int_0^1 \mathbf{C} \circ \mathbf{G}^k \circ \mathbf{B}[\varphi](t)dt - \int_0^1 \mathbf{C} \circ \mathbf{P} \circ \mathbf{B}[\varphi](t) \right) \\ &= \sum_{k \geq 0} \left(P(\mathcal{B} \times \mathcal{A}^k \times \mathcal{B}) - P(\mathcal{B})P(\mathcal{C}) \right). \end{aligned}$$

We say that $\mathcal{B}$ and $\mathcal{C}$ overlap if there exist words b, u and c such that $u \neq \varepsilon$ and $(bu, uc) \in (\mathcal{B} \times \mathcal{C}) \cup (\mathcal{C} \times \mathcal{B})$. Then we denote by $\mathcal{B} \uparrow \mathcal{C}$ the set of words that be obtained by overlapping words from $\mathcal{B}$ and $\mathcal{C}$. The correlation coefficient of the overlapping languages $\mathcal{B}$ and $\mathcal{C}$ is defined as

$$d(\mathcal{B},\mathcal{C}) := \frac{P(\mathcal{B} \uparrow \mathcal{C})}{P(\mathcal{B})P(\mathcal{C})} \tag{7.5.15}$$

Finally, the total correlation coefficient $m(\mathcal{B},\mathcal{C})$ between $\mathcal{B}$ and $\mathcal{C}$ is defined as

$$m(\mathcal{B},\mathcal{C}) = c(\mathcal{B},\mathcal{C}) + c(\mathcal{C},\mathcal{B}) + d(\mathcal{B},\mathcal{C}), \tag{7.5.16}$$

that is,

$$\begin{aligned} P(\mathcal{B})P(\mathcal{C})m(\mathcal{B},\mathcal{C}) &= P(\mathcal{B} \uparrow \mathcal{C}) \\ &+ \sum_{k\geq 0} \left[P(\mathcal{B} \times \mathcal{A}^k \times \mathcal{C}) + P(\mathcal{C} \times \mathcal{A}^k \times \mathcal{B}) - 2P(\mathcal{B})P(\mathcal{C}) \right]. \end{aligned}$$

We shall need these coefficients in the analysis of the generalized subsequence problem for dynamic sources.

7.5.2. Mean and variance

In this section we shall derive the mean and the variance of the number of occurrences $\Omega_n(\mathcal{W})$ of the generalized pattern as a subsequence for a dynamic source.

We first give a sketch of the forthcoming proof:

- We first describe the generating operators of the language $\mathcal{L}$ defined in (7.5.1) that we repeat it here

  $$\mathcal{L} = \mathcal{A}^* \times \mathcal{W}_1 \times \mathcal{A}^* \cdots \mathcal{W}_d \times \mathcal{A}^*.$$

  It turns out that the quasi-inverse $(I - z\mathbf{G})^{-1}$ operator is involved in such a generating operator.
- We then decompose the operator with the help of (7.5.9). We obtain a term related to $(1-z)^{-1}\mathbf{P}$ that gives the main contribution to the asymptotics, and another term coming from the operator $\mathbf{R}(z)$.
- We then compute the generating function of $\mathcal{L}$ using (7.5.3).
- Finally, we extract asymptotic behavior from the generating function.

The main finding of this section is summarized in the next theorem.

Theorem 7.5.1. *Consider a decomposable dynamical source endowed in the stationary density φ and let $\mathcal{W} = (\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_d)$ be a generalized nondense pattern.*
(i) *The expectation $\mathbf{E}(\Omega_n)$ of the number of occurrences of the generalized pattern $\mathcal{W}$ in a text of length n satisfies asymptotically*

$$\mathbf{E}(\Omega_n(\mathcal{W})) = \binom{n+d}{d} P(\mathcal{W}) + \binom{n+d-1}{d-1} P(\mathcal{W}) \, [C(\mathcal{W}) - T(\mathcal{W})] \, + O(n^{d-2}),$$

where

$$T(\mathcal{W}) = \sum_{i=1}^{d} \sum_{w \in \mathcal{W}_i} \frac{|w| P(w)}{P(\mathcal{W}_i)} \tag{7.5.17}$$

is the average length, and the correlation coefficient $C(\mathcal{W})$ *is the sum of the correlations* $c(\mathcal{W}_{i-1}, \mathcal{W}_i)$ *between languages* $\mathcal{W}_i$ *and* $\mathcal{W}_{i+1}$ *as defined in (7.5.14).*
(ii) *The variance of* Ω_n *is asymptotically equal to*

$$\mathrm{Var}(\Omega_n(\mathcal{W})) = \sigma^2(\mathcal{W})\ n^{2d-1}\left(1 + O(n^{-1})\right), \tag{7.5.18}$$

where the coefficient

$$\sigma^2(\mathcal{W}) = P^2(\mathcal{W}) \left[\frac{d - 2T(\mathcal{W})}{d!(d-1)!} + \frac{m(\mathcal{W})}{(2d-1)!} \right]$$

and the total correlation–coefficient $m(\mathcal{W})$ *can be computed as*

$$m(\mathcal{W}) := \sum_{1 \le i,j \le d} \binom{i+j-2}{i-1} \binom{2d-i-j}{d-i} m(\mathcal{W}_i, \mathcal{W}_j).$$

where $m(\mathcal{W}_i, \mathcal{W}_{i+1})$ *are defined in (7.5.16).*

Proof. We only prove part (i) leaving the proof of part (ii) as an exercise (cf. Exercise 7.5.3). We shall start with the language representation $\mathcal{L}$ defined in (7.5.1) that we recalled above. Its generating operator is

$$\mathbf{L}(z) = (I - z\mathbf{G})^{-1} \circ \mathbf{L}_r(z) \circ (I - z\mathbf{G})^{-1} \circ \cdots \circ \mathbf{L}_1(z) \circ (I - z\mathbf{G})^{-1}. \tag{7.5.19}$$

After applying the transformation (7.5.8) to $\mathbf{L}(z)$, we obtain an operator $\mathbf{M}_1(z)$

$$\mathbf{M}_1(z) = \left(\frac{1}{1-z}\right)^{d+1} \mathbf{P} \circ \mathbf{L}_r(z) \circ \mathbf{P} \circ \cdots \circ \mathbf{P} \circ \mathbf{L}_1(z) \circ \mathbf{P}$$

that has a pole of order $r+1$ at $z = 1$. Near $z = 1$, each operator $\mathbf{L}_i(z)$ is analytic and admits the expansion

$$\mathbf{L}_i(z) = \mathbf{L}_i + (z-1)\mathbf{L}_i'(1) + O(z-1)^2.$$

Therefore, the leading term of the expansion is

$$\left(\frac{1}{1-z}\right)^{d+1} \mathbf{P} \circ \mathbf{L}_r \circ \mathbf{P} \circ \cdots \circ \mathbf{P} \circ \mathbf{L}_1 \circ \mathbf{P}. \tag{7.5.20}$$

The second main term is a sum of r terms, each of them obtained by replacing the operator $\mathbf{L}_i(z)$ by its derivative $\mathbf{L}_i'(1)$ at $z = 1$. The corresponding generating function $M_1(z)$ satisfies near $z = 1$

$$M_1(z) = \left(\frac{1}{1-z}\right)^{d+1} P(\mathcal{W}) - \left(\frac{1}{1-z}\right)^{d} P(\mathcal{W})T(\mathcal{W}) + O\left(\frac{1}{1-z}\right)^{d-1}. \tag{7.5.21}$$

where the average length $T(\mathcal{W})$ is defined in (7.5.17).

After applying (7.5.8) in $\mathbf{L}(z)$, we obtain an operator $\mathbf{M}_2(z)$ that has a pole of order r at $z = 1$. This is a sum of $d+1$ terms, each of the term containing an occurrence of the operator $\mathbf{R}(z)$ between two generating operators of languages $\mathcal{W}_{i-1}, \mathcal{W}_i$. The corresponding generating function $M_2(z)$ has also a pole of order r at $z = 1$ and satisfies near $z = 1$

$$M_2(z) = \left(\frac{1}{1-z}\right)^d P(\mathcal{W}) \sum_{i=2}^{d} c(\mathcal{L}_{i-1}, \mathcal{L}_i) + O\left(\frac{1}{1-z}\right)^{d-1}.$$

Here, the correlation number $c(\mathcal{B}, \mathcal{C})$ between $\mathcal{B}$ and $\mathcal{C}$ is defined in (7.5.14). To complete the proof we need only to extract the coefficients of $P(z)/(1-z)^d$, as already discussed in previous sections. ■

7.6. Self-repetitive pattern matching

In this last section of the chapter, we change the model. So far we postulated the pattern w is given. Hereafter, we make the pattern part of the text, which is still randomly generated. To simplify our presentation, we assume that the text is emitted by a memoryless source. We should point out that the quantity analyzed here is in fact the typical depth in a (compact) suffix trie built over the suffixes of a randomly generated text.

7.6.1. Formulation of the problem

Let i be an arbitrary integer smaller than or equal to n. We define $D_n(i)$ to be the largest value of $k \le n$ such that X_i^{i+k-1} occurs at least twice in the text X_1^n of length n; in other words, such that $N_n(X_i^{i+k-1}) \ge 2$. We recall that $N_n(w)$ is the number of times pattern w occurs in the text X_1^n. Clearly, $N_n(X_i^{i+k-1}) \ge 1$. Our goal is to determine probabilistic behavior of a "typical" $D_n(i)$, that is, we define D_n to be equal to $D_n(i)$ when i is randomly and uniformly selected between 1 and n. More precisely,

$$\mathbf{P}(D_n = \ell) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{P}(D_n(i) = \ell)$$

for any $1 \le \ell \le n$.

Let $w \in \mathcal{A}^k$ be an arbitrary word of size k. Observe that

$$\mathbf{P}(D_n(i) \ge k \ \&\ X_i^{i+k-1} = w) = \mathbf{P}(N_n(w) \ge 2 \ \&\ X_i^{i+k-1} = w),$$

and

$$\sum_{i=1}^{n} \mathbf{P}(N_n(w) = r \ \&\ X_i^{i+k-1} = w) = r\mathbf{P}(N_n(w) = r).$$

Recall that $N_n(u) = \mathbf{E}(u^{N_n(w)}) = \sum_{r\geq 0} \mathbf{P}(N_n(w) = r)u^r$ is the probability generating function of $N_n(w)$. We sometimes shall write $N_{n,w}(u)$ to underline the fact that the pattern w is given. From above we conclude that

$$\begin{aligned}
\mathbf{P}(D_n \geq k) &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{P}(D_n(i) \geq k) \\
&= \sum_{w\in\mathcal{A}^k} \frac{1}{n}\sum_{i=1}^{n} \mathbf{P}(D_n(i) \geq k \ \&\ X_i^{i+k-1} = w) \\
&= \frac{1}{n}\sum_{w\in\mathcal{A}^k}\sum_{r\geq 2} r\mathbf{P}(N_n(w) = r) \\
&= \sum_{w\in\mathcal{A}^k}\left(\mathbf{P}(w) - \frac{1}{n}N'_{n,w}(0)\right) \\
&= 1 - \frac{1}{n}\sum_{w\in\mathcal{A}^k} N'_{n,w}(0),
\end{aligned}$$

where $N'_{n,w}(0)$ denotes the derivative of $N_n(u)$ at $u = 0$

Let now $D_n(u) = \mathbf{E}(u^{D_n}) = \sum_k \mathbf{P}(D_n = k)u^k$ be the probability generating function of D_n. Then above implies

$$D_n(u) = \frac{1}{n}\frac{(1-u)}{u}\sum_{w\in\mathcal{A}^*} u^{|w|}N'_{n,w}(0),$$

and the bivariate generating function $D(z,u) = \sum_n nD_n(u)z^n$ becomes

$$D(z,u) = \frac{1-u}{u}\sum_{w\in\mathcal{A}^*} u^{|w|}\frac{\partial}{\partial u}N_w(z,0) \tag{7.6.1}$$

where $N_w(z,u) = \sum_{n=0}^{\infty}\sum_{r=0}^{\infty} \mathbf{P}(N_n(w) = r)z^n u^r$. In Section 7.2 we worked with $\sum_{n=0}^{\infty}\sum_{r=1}^{\infty} \mathbf{P}(N_n(w) = r)z^n u^r$ and in (7.2.20) of Theorem 7.2.7 we provided a formula for it. Adding the term $N_0(z) = S_w(z)/D_w(z)$ we finally arrive at

$$N_w(z,u) = \frac{z^{|w|}\mathbf{P}(w)}{D_w^2(z)}\frac{u}{1-uM_w(z)} + \frac{S_w(z)}{D_w(z)},$$

where $M_w(z)$ is defined in (7.2.21) and $D_w(z) = (1-z)S_w(z) + z^{|w|}\mathbf{P}(w)$ (cf. 7.2.24) with $S_w(z)$ being the autocorrelation polynomial for w. Since

$$\frac{\partial}{\partial u}N_w(z,0) = z^{|w|}\frac{\mathbf{P}(w)}{D_w^2(z)},$$

we finally arrive at the following lemma that is the starting point of the subsequent analysis.

LEMMA 7.6.1. *The bivariate generating function for D_n is*

$$D(z,u) = \frac{1-u}{u} \sum_{w \in \mathcal{A}^*} (zu)^{|w|} \frac{\mathbf{P}(w)}{((1-z)S_w(z) + z^{|w|}\mathbf{P}(w))^2} \tag{7.6.2}$$

for $|u| < 1$ and $|z| < 1$, where $S_w(z)$ is the autocorrelation polynomial for w.

In this section, we prove the following result for a random text generated by a memoryless source over a finite alphabet $\mathcal{A}$ of size V with p_i being the probability of emitting symbol $i \in \mathcal{A}$. We denote by $h = -\sum_{i=1}^{V} p_i \log p_i$ the entropy rate of the source, and $h_2 = \sum_{i=1}^{V} p_i \log^2 p_i$. The reader is asked in Exercise 7.6.1 to extend below theorem to Markov sources.

THEOREM 7.6.2. (i) *For a biased memoryless source (i.e., $p_i \neq p_j$ for some $i \neq j$) and any $\varepsilon > 0$*

$$\mathbf{E}(D_n) = \frac{1}{h}\log n + \frac{\gamma}{h} + \frac{h_2}{h^2} + P_1(\log n) + O(n^{-\varepsilon}), \tag{7.6.3}$$

$$\mathrm{Var}(D_n) = \frac{h_2 - h^2}{h^3}\log n + O(1) \tag{7.6.4}$$

where $P_1(\cdot)$ is a periodic function with small amplitude in the case where the tuple $(\log p_1, \ldots,$
$\log p_V)$, is collinear with a rational tuple (i.e., $\log p_j / \log p_1 = r/s$ for some integers r and s) and converges to zero otherwise.
Furthermore, $(D_n - \mathbf{E}(D_n))/\mathrm{Var}(D_n)$ is asymptotically normal with mean zero and variance one that is, for fixed $x \in R$

$$\lim_{n\to\infty} \mathbf{P}\{D_n \le \mathbf{E}(D_n) + x\sqrt{\mathrm{Var}(D_n)}\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt \ ,$$

and for all integer m

$$\lim_{n\to\infty} \mathbf{E}\left[\frac{D_n - \mathbf{E}(D_n)}{\sqrt{\mathrm{Var} D_n}}\right]^m = \begin{cases} 0 & \text{when } m \text{ is odd} \\ \frac{m!}{2^{m/2}(\frac{m}{2})!} & \text{when } m \text{ is even.} \end{cases}$$

(ii) *For the unbiased source (i.e., $p_1 = \cdots = p_V = 1/V$), $h_2 = h^2$, the expected value $\mathbf{E}(D_n)$ is given by (7.6.3) above, and for any $\varepsilon > 0$*

$$\mathrm{Var}(D_n) = \frac{\pi^2}{6\log^2 V} + \frac{1}{12} + P_2(\log n) + O(n^{-\varepsilon})$$

where $P_2(\log n)$ is a periodic function with small amplitude The limiting distribution of D_n does not exist, but one finds

$$\lim_{n\to\infty} \sup_x \mid \mathbf{P}(D_n \le x) - \exp(-nV^{-x}) \mid = 0$$

for any fixed real x.

In passing we observe that the quantity D_n is also the depth of a randomly selected suffix in a compact suffix trie. Such a trie is a compacted version of suffix tries defined in Chapter 2. In a compact suffix trie one deletes all unary nodes at the *bottom* of the non-compact suffix trie. Observe that in a compact suffix trie, which we further call simply a suffix trie, the path from the root to node i (representing the ith suffix) is the shortest suffix that distinguishes it from all other suffixes. The quantity $D_n(i)$ defined above represents the depth of the i suffix in the associated suffix trie, while D_n is the *typical depth*, that is, the depth of a randomly selected terminal node in the suffix trie. Theorem 7.6.2 tells us that the typical depth is normally distributed with the average depth asymptotically equal to $\frac{1}{h}\log n$ and variance $\Theta(\log n)$ for biased memoryless source. In the unbiased case variance is $O(1)$ and the (asymptotic) distribution is of the extreme distribution type. Interestingly, as proved below, the depth in a suffix trie (built over *one* sequence generated by a memoryless source) is asymptotically equivalent to the depth in a trie built over n *independently* generated strings. Thus suffix tries resemble tries!

7.6.2. Random tries resemble suffix tries

The proof of Theorem 7.6.2 hinges on establishing asymptotic equivalence between D_n introduced above and a new random variable D_n^T defined as follows: First, for n *independently* generated texts (by the same memoryless source as for D_n) we denote by $D_n^T(i)$ for an integer $i \le n$ the length of the longest prefix of the ith text that is also a prefix of another text, say the jth text, $j \ne i$. Then the random variable D_n^T is defined by selecting integer i uniformly between 1 and n. We also define $D_n^T(u) = \sum_k \mathbf{P}(D_n^T = k)u^k$ and $D^T(z,u) = \sum_n nD_n^T(u)z^n$. Observe that D_n^T is in fact the typical depth in a trie built over these n independent texts.

It is relatively easy to derive the generating function of D_n^T, as shown below.

Lemma 7.6.3. *For all $n \ge 1$*

$$D_n^T(u) = \frac{1-u}{u}\sum_{w\in\mathcal{A}^*} u^{|w|}\mathbf{P}(w)(1-\mathbf{P}(w))^{n-1},$$

$$D^T(z,u) = \frac{1-u}{u}\sum_{w\in\mathcal{A}^*} u^{|w|}\frac{z\mathbf{P}(w)}{(1-z+\mathbf{P}(w)z)^2}$$

for all $|u| \le 1$ and $|z| < 1$.

Proof. It suffices to observe that

$$\mathbf{P}(D_n^T(i) < k) = \sum_{w\in\mathcal{A}^k} \mathbf{P}(w)(1-\mathbf{P}(w))^{n-1}.$$

Indeed, $D_n^T(i) < k$ if there is a word $w \in \mathcal{A}^k$ such the a prefix of the ith string is equal to w and none of the other text prefixes are equal to w. ∎

Our goal now is to prove that $D_n(u)$ and $D_n^T(u)$ are asymptotically close as $n \to \infty$. This requires several preparatory steps outlined below that will lead to

$$D_n^T(u) - D_n(u) = (1-u)O(n^{-\varepsilon}) \tag{7.6.5}$$

for some $\varepsilon > 0$ and all $|u| < \beta$ for $\beta > 1$. Consequently,

$$|\mathbf{P}(D_n \le k) - \mathbf{P}(D_n^T \le k)| = O(n^{-\varepsilon}\beta^{-k})$$

for all positive integers k. In Lemma 7.6.11 we shall prove that D_n^T is asymptotically normal, hence D_n is normal. This will prove Theorem 7.6.2.

We start with a lemma indicating that for most words w the autocorrelation polynomial $S_w(z)$ is very close to 1 for z non-negative. This lemma provides information about analytical properties of the autocorrelation polynomial.

Lemma 7.6.4. *There exist $\delta < 1$, $\theta > 0$ and $\rho > 1$ such that $\rho\delta < 1$ and*

$$\sum_{w\in\mathcal{A}^k} [\![|S_w(\rho) - 1| \le (\rho\delta)^k\theta]\!] \mathbf{P}(w) \ge 1 - \theta\delta^k. \tag{7.6.6}$$

Proof. To simply notations, let P_k be the probability measure on $\mathcal{A}^k$ such that $P_k(A) = \sum_{w\in\mathcal{A}^k} [\![w \in A]\!] \mathbf{P}(w)$. Thus we need to prove that $P_k(S_w(\rho) \le 1 + (\rho\delta)^k\theta) \ge 1 - \theta\delta^k$.

Let i be an integer smaller than $k \in \mathcal{P}(w)$, where $\mathcal{P}(w)$ is the autocorrelation set for w. It is easy to see that (cf. Exercise 7.6.2)

$$P_k(k - i \in \mathcal{P}(w)) = \left(\sum_{j=1}^{V} p_j^{\lfloor k/i\rfloor+1}\right)^r \left(\sum_{j=1}^{V} p_j^{\lfloor k/i\rfloor}\right)^{i-r} \tag{7.6.7}$$

where $r = k - \lfloor k/i\rfloor i$. Denoting $p = \max_i p_i$ we have

$$P_k(k - i \in \mathcal{P}(w)) \le p^{k-i}.$$

Thus $P_k(\max(\mathcal{P}(w) - \{k\}) \ge k/2) \le \sum_{i=1}^{\lfloor k/2\rfloor} P_k(i+1 \in \mathcal{P}(w)) \le \frac{p^{k/2}}{1-p}$. Now, if the word w is such that $\max(\mathcal{P}(w) - \{k\}) < k/2$, then $S_w(\rho) \le \sum_{i>\lfloor k/2\rfloor}^{k} \rho^i p^i \le \rho^k \frac{p^{k/2}}{1-p}$. Therefore, it suffices for (7.6.6) to select $\delta = \sqrt{p}$, $\theta = (1-p)^{-1}$ and $\rho > 1$ such that $\rho\delta < 1$. ∎

In the next lemma we show that $D(z,u)$ can be analytically continued above the unit disk, that is, for $|u| > 1$.

Lemma 7.6.5. *The generating function $D(z,u)$ can be analytically continued for all $|u| < \delta^{-1}$ and $|z| < 1$ where $\delta < 1$.*

Proof. Let $|u| < 1$ and $|z| < 1$. Consider the following identity

$$\sum_w (uz)^{|w|} \frac{\mathbf{P}(w)}{(1-z)^2} = \frac{1}{(1-uz)(1-z)^2}.$$

Therefore, for $|z| < 1$

$$uD(z,u) - \frac{(1-u)}{(1-uz)(1-z)^2} = (1-u)\sum_w (zu)^{|w|}\mathbf{P}(w)\left(\frac{1}{D_w^2(z)} - \frac{1}{(1-z)^2}\right)$$
$$= (u-1)\sum_w (zu)^{|w|}\mathbf{P}(w)\frac{1}{D_w^2(z)(1-z)^2}(D_w(z)-(1-z))(D_w(z)+(1-z)),$$

where we recall $D_w(z)-(1-z) = (1-z)(S_w(z)-1)+\mathbf{P}(w)z^{|w|}$. By Lemma 7.6.4

$$P_k(|D_w(z)-(1-z)| \le (|1-z|+1)\delta^{|w|}) \ge 1 - O(\delta^{|w|})$$

for all w such that $|w| = k$. Moreover, for any bounded function $f(w)$ such that $f(w) \le f_{max}$ for all w with $|w| = k$, we also have the following estimate for all y:

$$\sum_{|w|=k} \mathbf{P}(w)f(w) \le y + f_{\max}P_k(f(w) > y)\ . \tag{7.6.8}$$

In particular, we take $f(w) = D_w(z) - (1-z)$ and we have $f_{\max} = O(1)$ since $|S_w(z)| < (1-p)^{-1}$ (p as defined as in proof of lemma 7.6.4). Now taking $y = (|1-z|+1)\delta^k$, using the above we obtain

$$uD(z,u) - \frac{1-u}{(1-uz)(1-z)^2} = (u-1)\sum_{k=0}^{\infty}(zu)^k O((|1-z|+1)\delta^k + \delta^k)$$

for all w. In conclusion,

$$uD(z,u) - \frac{(1-u)}{(1-uz)(1-z)^2} = O\left(\frac{u-1}{1-\delta|u|}\right)$$

for $\delta < 1$ and $|z| < 1$, which completes the proof. ∎

Before we proceed, we need two technical lemmas.

LEMMA 7.6.6. *There exists K, a constant $\rho' > 1$ and $\alpha > 0$ such that for all w with $|w| \ge K$ we have*

$$|S_w(z)| \ge \alpha$$

for $|z| \le \rho'$ with $\rho' > 1$ such that $p\rho' < 1$.

Proof. Let ℓ be an integer and $\rho' > 1$ such that $p\rho' + (p\rho')^\ell < 1$. Let $k > \ell$ and let w such that $|w| > k$. Let $i = \max(\mathcal{P} - \{k\})$. If $i \le \ell$, then for all z such that $|z| \le \rho'$ we have

$$|S_w(z)| \ge 1 - \frac{(p\rho')^\ell}{1-p\rho'}.$$

If $i > \ell$, let $q = \lfloor k/i \rfloor$, then $w = u^q v$ where u is the prefix of length i of word w, and v is the suffix of length $k - iq$. Thus

$$S_w(z) = \frac{1 - (\mathbf{P}(u)z^i)^q}{1 - \mathbf{P}(u)z^i} + (\mathbf{P}(u)z^i)^q S_{uv}(z),$$

where $S_{uv}(z)$ is the autocorrelation polynomial of uv. This implies

$$|S_w(z)| \geq \frac{1 - (p\rho')^{qi}}{1 + (p\rho')^i} - \frac{(p\rho')^{iq} - (p\rho')^k}{1 - p\rho'}.$$

But since $i > \ell$, we obtain

$$|S_w(z)| \geq \frac{1 - (p\rho') - 3(p\rho')^{k-\ell}}{1 + p\rho'} > 0,$$

which completes the proof. ∎

LEMMA 7.6.7. *There exists an integer K' such that for $|w| \geq K'$ there is only one root of $D_w(z)$ in the disk $|z| \leq \rho'$ for $\rho' > 1$.*

Proof. Let K_1 be such that $(p\rho')^{K_1} < \alpha(\rho' - 1)$ holds for the α and ρ' as in Lemma 7.6.6. Denote $K' = \max\{K, K_1\}$, where K is defined above. Note also that the above condition implies that for all w such that $|w| = k > K'$ we have $\mathbf{P}(w)(\rho')^k < \alpha(\rho - 1)$. Hence, for $|w| > K'$ we have $|\mathbf{P}(w)z^k| < |(z-1)S_w(z)|$ on the circle $|z| = \rho' > 1$. Therefore, by Rouché's theorem the polynomial $D_w(z)$ has the same number of roots as $(1 - z)S_w(z)$ in the disk $|z| \leq \rho'$. But, the polynomial $(1-z)S_w(z)$ has only a single root in this disk since by Lemma 7.6.6 we have $|S_w(z)| > 0$ in $|z| \leq \rho'$. ∎

We just establish that there exists the smallest root of $D_w(z) = 0$ that we denote as A_w. Let also C_w and D_w be the first and the second derivatives of $D_w(z)$ at $z = A_w$, respectively. Using bootstrapping, one easily obtains the following expansions

$$\begin{aligned}
A_w &= 1 + \frac{1}{S_w(1)}\mathbf{P}(w) + O(\mathbf{P}(w)^2), \\
C_w &= -S_w(1) + \left(k - \frac{2S'_w(1)}{S_w(1)}\right)\mathbf{P}(w) + O(\mathbf{P}(w)^2), \\
E_w &= -2S'_w(1) + \left(k(k-1) - \frac{3S''_w(1)}{S_w(1)}\right)\mathbf{P}(w) + O(\mathbf{P}(w)^2),
\end{aligned}$$

where $S'_w(1)$ and $S''_w(1)$, respectively, denote the first and the second derivatives of $S_w(z)$ at $z = 1$.

Finally, we are ready to compare $D_n(u)$ with $D_n^T(u)$ to conclude that they do not differ too much as $n \to \infty$. Let us define two new generating functions

$Q_n(u)$ and $Q(z,u)$ that represent the difference between $D_n(u)$ and $D_n^T(u)$, that is,

$$Q_n(u) = \frac{u}{1-u}\left(D_n(u) - D_n^T(u)\right),$$

and

$$Q(z,u) = \sum_{n=0}^{\infty} n\, Q_n(u) z^n = \frac{u}{1-u}\left(D(z,u) - D^T(z,u)\right).$$

Then

$$Q(z,u) = \sum_w u^{|w|}\mathbf{P}(w)\left(\frac{z^{|w|}}{D_w(z)^2} - \frac{z}{(1-z+\mathbf{P}(w)z)^2}\right).$$

It is not difficult to establish asymptotics of $Q_n(u)$ by appealing to the Cauchy theorem. This is done in the following lemma.

Lemma 7.6.8. *There exists $B > 1$ such that for all $|u| \le \beta$ the following evaluation holds*

$$Q_n(u) = \frac{1}{n}\sum_w u^{|w|}\mathbf{P}(w)\left(A_w^{|w|-n-1}\left(\frac{n+1-|\mathcal{W}|}{C_w^2 A_w} + \frac{E_w}{C_w^3}\right) - n(1-\mathbf{P}(w))^{n-1}\right) + O(B^{-n})$$

for some $\beta > 1$.

Proof. By Cauchy's formula

$$nQ_n(u) = \frac{1}{2i\pi}\oint Q(z,u)\frac{dz}{z^{n+1}},$$

where the integration is along a loop contained in the unit disk that encircles the origin. Let w be such that $|w| \ge K'$, where K' is defined in Lemma 7.6.7. From the proof of Lemma 7.6.7 we conclude that $D_w(z)$ and $(1 - z + \mathbf{P}(w)z)$ have only one root in $|z| \le \rho$ for some $\rho > 1$. Applying Cauchy's residue theorem we obtain

$$\frac{1}{2i\pi}\oint u^{|w|}\mathbf{P}(w)\frac{dz}{z^{n+1}}\left(\frac{z^{|w|}}{D_w(z)^2} - \frac{z}{(1-z+\mathbf{P}(w)z)^2}\right) =$$
$$= u^{|w|}\mathbf{P}(w)\left(\frac{A_w^{|w|-n-1}}{u}\left(\frac{n+1-|w|}{C_w^2 A_w} + \frac{E_w}{C_w^3}\right) - n(1-\mathbf{P}(w))^{n-1}\right) + I_w(\rho,u),$$

where

$$I_w(\rho,u) = \frac{\mathbf{P}(w)}{2i\pi}\int_{|z|=\rho} u^{|w|}\frac{dz}{z^{n+1}}\left(\frac{z^{|w|}}{D_w(z)^2} - \frac{z}{(1-z+\mathbf{P}(w)z)^2}\right).$$

To establish a bound for $I_w(\rho,u)$ we argue exactly in the same manner as in the proof of Theorem 7.6.5. This leads for $|w| > K'$ to

$$\sum_{|w|=k} I_w(\rho,u) = O((\delta\rho u)^k \rho^{-n})$$

since for all w we also have $S_w(\rho) \le 1/(1-p\rho)$ and $D_w(z) = O(\rho^k)$ in the circle $|z| \le \rho$. Set now $\beta = (\delta\rho)^{-1} > 1$. Then, for $|u| < \beta$ we have

$$\sum_{\{w:\ |w|>K'\}} I_w(\rho, u) = O(\sum_w \mathbf{P}(w)\rho^{|w|-n}) = O(\rho^{-n}).$$

This proves our bound since the other terms ($|w| < K'$) contribute only B^{-n} for some $B > 1$ due to the fact that all roots of $D_w(z)$ have magnitudes greater than 1. ∎

In the next lemma we show that $Q_n(u) \to 0$ as $n \to \infty$.

Lemma 7.6.9. *For all $1 < \beta < \delta^{-1}$, there exists $\varepsilon > 0$ such that $Q_n(u) = (1-u)O(n^{-\varepsilon})$ uniformly for $|u| \le \beta$.*

Proof. The expansion of E_w with respect to $\mathbf{P}(w)$, and Lemma 7.6.4 show that as $n \to \infty$ the following holds $\sum_w u^{|w|}\mathbf{P}(w)A_w^{-n}E_w/C_w^3 = O(1)$. Therefore, by Lemma 7.6.8 we have

$$Q_n(u) = \sum_w u^{|w|}\mathbf{P}(w)\left(\frac{A_w^{|w|-n-2}}{C_w^2} - (1-\mathbf{P}(w))^{n-1}\right) + O(1/n)\ .$$

Let now $f_w(x)$ be a function defined for x real by

$$f_w(x) = \frac{A_w^{|w|-x-2}}{C_w^2} - (1-\mathbf{P}(w))^{x-1}.$$

By the same arguments as used in proving (7.6.8) in Theorem 7.6.5, we note that $\sum_w u^{|w|}\mathbf{P}(w)f_w(x)$ is absolutely convergent for all x and u such that $|u| \le \beta$. The function $\bar{f}_w(x) = f_w(x) - f_w(0)e^{-x}$ is exponentially decreasing when $x \to +\infty$ and is $O(x)$ when $x \to 0$; therefore its Mellin transform defined as

$$\bar{f}_w^*(s) = \int_0^\infty \bar{f}_w(x)x^{s-1}dx$$

is well defined for $\Re(s) > -1$. In this region we obtain

$$\bar{f}_w^*(s) = \Gamma(s)\left(A_w^{|w|-1}\frac{(\log A_w)^{-s} - 1}{A_w C_w^2} - \frac{(-\log(1-\mathbf{P}(w))^{-s} - 1}{1-\mathbf{P}(w)}\right)\ ,$$

where $\Gamma(s)$ is the gamma function. Let $g^*(s,u)$ be the Mellin transform of the series $\sum_w u^{|w|}\mathbf{P}(w)\bar{f}_w(x)$ which exists at least in the strip $(-1, 0)$. Formally, we have

$$g^*(s,u) = \sum_w u^{|w|}\mathbf{P}(w)\bar{f}_w^*(s)\ .$$

We can reverse the Mellin transform $g^*(s,u)$ provided that the following holds.

Lemma 7.6.10. *The function $g^*(s,u)$ is analytical in $\Re(s) \in (-1, c)$ for some $c > 0$.*

Assuming Lemma 7.6.10 is granted, we have

$$Q_n(u) = \frac{1}{2i\pi}\int_{\varepsilon-i\infty}^{\varepsilon+i\infty} g^*(s,U)n^{-s}ds + O(1/n) \ + \sum_w u^{|w|}\mathbf{P}(w)f_w(0)e^{-n},$$

for some $\varepsilon \in (0,c)$. Notice that the last term of the above contributes $O(e^{-n})$, and can be safely ignored. Furthermore, a simple majorization under the integral gives the evaluation $Q_n(u) = O(n^{-\varepsilon})$ which completes the proof. ∎

Proof of Lemma 7.6.10: We establish the absolute convergence of $g^*(s,u)$ for all s such that $\Re(s) \in (-1,c)$ and $|u| \le \beta$. Let us define $h^*(s,u) = \frac{g^*(s,u)}{\Gamma(s)}$. Note that for any fixed s we have the following

$$(\log A_w)^{-s} = \left(\frac{\mathbf{P}(w)}{1+S_w(1)}\right)^{-s}(1+O(\mathbf{P}(w)))\ ,$$
$$(-\log(1-\mathbf{P}(w)))^{-s} = \mathbf{P}(w)^{-s}(1+O(\mathbf{P}(w)))\ .$$

Thus

$$\frac{(\log A_w)^{-s}-1}{A_w^{2-|w|}C_w^2} - \frac{(-\log(1-\mathbf{P}(w)))^{-s}-1}{1-\mathbf{P}(w)} =$$
$$= \mathbf{P}(w)^{-s}\left[(1+a_w(1))^s(1+O(|w|\mathbf{P}(w)) - (1+O(\mathbf{P}(w))\right] + O(|w|\mathbf{P}(w))\ .$$

By Lemma 7.6.4, $P_k(S_w(1) \le 1+\theta\delta^k) \ge 1 - O(\delta^k)$, and hence

$$h^*(s,u) = \sum_{k=0}^{\infty}\left(\sup\{p^{-\Re(s)}, q^{-\Re(s)}\}|u|\delta\right)^k O(1)$$

that absolutely converges for all values of s such that $\Re(s) < c$ where c satisfies $\sup\{p^{-c}, q^{-c}\} < (\delta\beta)^{-1}$. Since $h^*(0,u) = 0$ by definition, the pole of $\Gamma(s)$ at $s = 0$ is canceled in $g^*(s,u)$, and therefore $h^*(s,u)$ does not show any singularities in the strip $\Re(s) \in (-1,c)$. ∎

To complete the proof of our main Theorem 7.6.2, we need an asymptotic analysis of $D_n^T(u)$ which is presented next. We recall that D_n^T represents also the typical depth in a trie built from n independently generated strings.

Lemma 7.6.11. *There exists $\varepsilon > 0$ such that*

$$D_n^T(u) = (1-u)n^{\kappa(u)}(\Gamma(\kappa(u)) + P(\log n, u))) + O(n^{\varepsilon}),$$

where

$$u\sum_{i=1}^{V} p_i^{1-\kappa(u)} = 1$$

and $P(\log n, u)$ is periodic function with small amplitude in the case where the vector $(\log p_1,$
$\dots, \log p_V)$ *is collinear with a rational tuple, and converges to zero when $n \to \infty$ otherwise.*

Proof. We begin with the identity

$$D_n^T(u) = \frac{1-u}{u} \sum_{w \in \mathcal{A}^*} u^{|w|} \mathbf{P}(w)(1 - \mathbf{P}(w))^{n-1}.$$

We argue exactly in the same manner as in the proof of Lemma 7.6.8. We find the Mellin transform $T^*(s,u) = \int_0^\infty x^{s-1} dx u/(1-u) D_x^T(u) dx$ to be

$$T^*(s,u) = \sum_{w \in \mathcal{A}^*} u^{|w|} \mathbf{P}(w)(-\log(1 - \mathbf{P}(w)))^{-s} \Gamma(s).$$

Using the fact that for s bounded $(-\log(1-\mathbf{P}(w)))^{-s} = \mathbf{P}(w)^{-s}(1+O(s\mathbf{P}(w)))$, we conclude

$$T^*(s,u) = \Gamma(s) \left(\frac{u \sum_{i=1}^V p_i^{1-s}}{1 - u \sum_{i=1}^V p_i^{1-s}} + g(s,u) \right),$$

where

$$g(s,u) = O\left(\frac{us \sum_{i=1}^V p_i^{2-\Re(s)}}{1 - |u| \sum_{i=1}^V p_i^{2-\Re(s)}} \right).$$

Let $\kappa(u)$ be the main root of $1 = u \sum_{i=1}^V p_i^{1-s}$. The other roots of $1 = u \sum_{i=1}^V p_i^{1-s}$, are countable and we denote them as $\kappa_k(u)$ for $k \neq 0$ integer. For all integers k we have $\Re(\kappa_k(u)) \geq \kappa(u)$. Using the inverse Mellin we find

$$D_n^T = \frac{1-u}{2i\pi u} \int_{-i\infty}^{+i\infty} T^*(s,u) n^{-s} ds.$$

We now consider $|u| < \delta^{-1}$ for $\delta < 1$. Then there exists ε such that for $\Re(s) \leq \varepsilon$ the function $g(s,u)$ has no singularity. Moving the integration path to the left of $\Re(s) = \varepsilon$, and applying the reside theorem we find the following estimate

$$D_n^T(u) = (1-u) \frac{\Gamma(\kappa(u))}{h(u)} n^{\kappa(u)} + (1-u) \sum_k \frac{\Gamma(\kappa_k(u))}{h_k(u)} n^{\kappa_k(u)} + O(n^{-\varepsilon}) \quad (7.6.9)$$

with $h(u) = -\sum_i p_i^{1-\kappa(u)} \log p_i$ and $h_k(u) = -\sum_i p_i^{1-\kappa_k(u)} \log p_i$. When $\log p_i$'s are collinear with a rational vector, then there is subset of $\kappa_k(u)$ that have the same real part as $\kappa(u)$ and also equally spaced on the vertical line $\Re(s) = \Re(\kappa(u))$. In this case their contribution to (7.6.9) is

$$n^{\kappa(u)} \sum_k \frac{\Gamma(\kappa_k(u))}{h(u)} \exp((\kappa_k(u) - \kappa(u)) i \log n).$$

When the $\log p_i$'s are not collinear with a rational vector the contribution of the $\kappa_k(u)$ divided by $n^{\kappa(u)}$ converges to zero when $n \to \infty$. ∎

The last lemma completes the proof of Theorem 7.6.2. Indeed, it suffices to observe that for $t \to 0$

$$\kappa(e^t) = c_1 t + \frac{c_2}{2} t^2 + O(t^3) \quad (7.6.10)$$

where $c_1 = 1/h$ and $c_2 = (h_2 - h^2)/h^3$. We concentrate first on the asymmetric case. From the expression of $D_n^T(u)$ we find immediately the first and the second moments via the first and the second derivatives of $D_n^T(u)$ at $u = 1$ with the appropriate asymptotic expansion in $c_1 \log n$ and in $c_2 \log n$. In order to obtain the limiting normal distribution we prove

$$e^{-tc_1 \log n/\sqrt{c_2 \log n}} D_n^T\left(e^{t/\sqrt{c_2 \log n}}\right) \to e^{t^2/2}$$

using $n^{\kappa(u)} = \exp(\kappa(u) \log n)$ and referring to expansion (7.6.10).

For the symmetric case there is no normal limiting distribution since variance is $O(1)$. However, there are oscillation due to the fact that all $\kappa_k(u)$ are aligned on a vertical line. This completes the proof of Theorem 7.6.2.

Problems

Section 7.2

7.2.1 Prove (7.2.9).

7.2.2 In Theorem 7.2.8 we prove that for irreducible aperiodic Markov chain the variance $\mathrm{Var}(N_n) = nc_1 + c_2$ (cf. (7.2.26)). Prove that $c_1 > 0$.

7.2.3 Prove that $(N_n - \mathbf{E}(N_n))/\sqrt{\mathrm{Var}(N_n)}$ converges in moments to the appropriate moments of the standard normal distribution.

7.2.4 Let $\rho(t)$ be a root of $1 - e^t M_{\mathcal{W}}(e^\rho) = 0$. Observe that $\rho(0) = 0$. Prove that $\rho(t) > 0$ for $t \neq 0$ for $p_{ij} > 0$ for all $i, j \in \mathcal{A}$.

7.2.5 Prove the expression (7.2.44) for θ_a of Theorem 7.2.12 (cf. Denise and Régnier (2004)).

Section 7.3

7.3.1 Extend the analysis of Section 7.3 to multisets $\mathcal{W}$, that is, a word w_i may occur several times in $\mathcal{W}$.

7.3.2 Prove language relationships (7.3.2)–(7.3.2).

7.3.3 Derive explicit formulas for θ_a appearing in Theorem 7.3.3(iv).

7.3.4 Find explicit formulas for the values of the mean $\mathbf{E}(N_n(\mathcal{W}))$ and of the variance $\mathrm{Var}(N_n(\mathcal{W}))$ for the generalized pattern matching discussed in Section 7.3.2 for $\mathcal{W}_0 = \emptyset$ and $\mathcal{W}_0 \neq \emptyset$.

7.3.5 Derive explicit formulas for σ_a and θ_a in (7.3.27) appearing in Theorem 7.3.10.

7.3.6 Enumerate (ℓ, k) sequences over a non binary alphabet (i.e., generalize the analysis of Section 7.3.3).

Section 7.4

7.4.1 Find an explicit formula for the generating function $B_2^{[p]}(z)$ of the collection $\mathcal{B}_2^{[p]}$.

7.4.2 Design a dynamic programming algorithm to compute the correlation algorithm, $\kappa^2(\mathcal{W})$.

7.4.3 Establish the rate of convergence for the Gaussian law from Theorem 7.4.5.

7.4.4 For the fully unconstrained subsequence problem establish the large deviations (cf. Janson (pear)).

7.4.5 Provide details of the proof for Theorem 7.4.6.

7.4.6 Let $\mathcal{W} = \{w_1, \ldots, w_d\}$ be a set of patterns w_i. The pattern $\mathcal{W}$ occurs as a subsequence in the text if any of w_i occurs as a subsequence. Analyze this generalization of the subsequence pattern matching.

7.4.7 Let w be a pattern. Set W to be a window size with $|w| \leq W \leq n$. Consider the *windowed subsequence pattern matching* in which w must appear as a subsequence within the window W. Analyze the number of windows that has at least one occurrence of w as a subsequence within the window (cf. Gwadera, Atallah, and Szpankowski 2003).

Section 7.5

7.5.1 Prove the generating operators identities (7.5.5)–(7.5.8).

7.5.2 Prove (7.5.11)–(7.5.13).

7.5.3 Prove the second part of Theorem 7.5.1, that is, derive formula (7.5.18) for variance of $\Omega_n(\mathcal{W})$.

7.5.4 Does the central limit theorem holds for the generalized subsequence problem discussed in Section 7.5? What about large deviations?

Section 7.6

7.6.1 Extend Theorem 7.6.2 for Markov sources.

7.6.2 Prove (7.6.7) and extend it to Markov sources (cf. Apostolico and Szpankowski 1992).

7.6.3 Let $\kappa(u)$ be the main root of $1 = u\sum_{i=1}^{V} p_i^{1-s}$, and $\kappa_k(u)$ for $k \neq 0$ integer are other roots of $1 = u\sum_{i=1}^{V} p_i^{1-s}$. Prove that for all integers k we have $\Re(\kappa_k(u)) \geq \kappa(u)$.

Notes

Algorithmic aspects of pattern matching are presented in numerous books. We mention here Crochemore and Rytter (1994) and Gusfield (1997) (cf. also Apostolico (1985)). Public domain utilities like `agrep`, `grappe`, `webglimpse` for finding general patters were recently developed by Wu and Manber (1995), Kucherov and Rusinowitch (1997), and others. Various data compression schemes are studied in Wyner and Ziv (1989), Wyner (1997), Yang and Kieffer (1998), Ziv and Lempel (1978), Ziv and Merhav (1993)). Prediction based on pattern matching is discussed in Jacquet, Szpankowski, and Apostol (2002). Algorithmic aspect of pattern matching can also be found in Chapter 2 and Chapter 8 of this book.

In this chapter the emphasis is on analysis of pattern matching problems by analytic methods in a probabilistic framework. Probabilistic models are discussed in Section 7.1 and Chapter 1. Markov models are presented in many standard books (cf. Karlin and Ost (1987)). Dynamic sources were introduced by Vallée (2001) (cf. also Clement, Flajolet, and Vallée (2001), Bourdon and Vallée (2002)). General stationary ergodic sources are discussed in Shields (1969).

In this chapter analytic tools are used to investigate combinatorial pattern matching problems. The reader is referred to Alon and Spencer (1992), Szpankowski (2001), Waterman (1995) (cf. also Arratia and Waterman (1989, 1994)) for in-depth discussion of probabilistic tools. Analytic techniques are thoroughly explained in Sedgewick and Flajolet (1995) and Szpankowski (2001). The reader may also consult Atallah, Jacquet, and Szpankowski (1993), Bender (1973), Clement et al. (2001), Hwang (1996), Jacquet and Szpankowski (1994, 1998). The Perron–Frobenius theory and the spectral decomposition of matrices can be found in Gantmacher (1959), Karlin and Taylor (1975), Kato (1980), Szpankowski (2001). Operator theory is discussed in Kato (1980).

Exact string matching is presented in Section 7.2. There are numerous references. Our approach is founded in the work of Guibas and Odlyzko (1981a) and Guibas and Odlyzko (1981b). The presentation of this section follows very closely recent work of Régnier and Szpankowski (1998a) and Régnier (2000). More probabilistic approach is adopted in Chapter 2 and in Prum et al. (1995). Example 7.2.13 is taken from Denise, Régnier, and Vandenbogaert (2001).

Generalized string matching problem discussed in Section 7.3 was introduced in Bender and Kochman (1993). The analysis of string matching over reduced set of patterns appears in Régnier and Szpankowski (1998b) (cf. also Guibas and Odlyzko (1981b)). An automaton approach to motif finding was proposed in Nicodème et al. (2002). The general string matching was first dealt with in Bender and Kochman (1993), however, our presentation follows a different path simplifying previous analyses. It is closely related to the subsequence pattern matching analysis presented in Flajolet, Guivarc'h, Szpankowski, and Vallée (2001). The (ℓ, k) sequence analysis is taken from Szpankowski (2001).

The subsequence pattern matching or the hidden pattern matching discussed in Section 7.4 is based on Flajolet et al. (2001). Proceeding along different tracks, Janson (pear) has related this particular case to his treatment of U–statistics via Gaussian Hilbert spaces; see Chapter XI of Janson's book Janson (1997) for the type of method employed. Example 7.4.7 was fully developed in Gwadera et al. (2003).

The generalized subsequence pattern matching discussed in Section 7.5 is taken from Bourdon and Vallée (2002). The operator generating function approach for dynamic sources was developed by Vallée (2001).

In Section 7.6 we present some results for the self-repetitive pattern matching. Theorem 7.6.2 was proved in Jacquet and Szpankowski (1994), however, our proof in this section is somewhat simplified. In particular, proof of the crucial Lemma 7.6.1 is new and based on results presented in Section 7.2. Lemma 7.6.11 is due to Jacquet and Régnier (1986) (for an extension to Markov sources see Jacquet and Szpankowski (1991)). Mellin transform is explained in depth in Flajolet, Gourdon, and Dumas (1995), Szpankowski (2001). Tries are treated in depth in Mahmoud (1992) and Szpankowski (2001). As mentioned, the quantity D_n analyzed in the section is also the typical depth in a suffix tries introduced in Chapter 2 (cf. also Apostolico (1985)). Probabilistic analysis of suffix tries can be found in Apostolico and Szpankowski (1992), Devroye, Szpankowski, and Rais (1992), Szpankowski (1993a, 1993b). As discussed in the section, suffix tries are often appear in analysis of data compression schemes (cf. Wyner and Ziv (1989), Wyner (1997), Yang and Kieffer (1998), Ziv and Lempel (1978), Ziv and Merhav (1993)).

CHAPTER 8

Periodic Structures in Words

8.0. Introduction

Repetitions (periodicities) in words are important objects that play a fundamental role in combinatorial properties of words and their applications to string processing, such as compression or biological sequence analysis. Using properties of repetitions allows one to speed up pattern matching algorithms.

The problem of efficiently identifying repetitions in a given word is one of the classical pattern matching problems. Recently, searching for repetitions in strings received a new motivation, due to the biosequence analysis. In DNA sequences, successively repeated fragments often bear an important biological information and their presence is characteristic for many genomic structures (such as telomer regions for example). From a practical viewpoint, satellites and alu-repeats are involved in chromosome analysis and genotyping, and thus are of major interest to genomic researchers. Thus, different biological studies based on the analysis of tandem repeats have been done, and even databases of tandem repeats in certain species have been compiled.

In this chapter, we present a general efficient approach to computing different periodic structures in words. It is based on two main algorithmic techniques – a special factorization of the word and so-called longest extension functions – described in Section 8.3. Different applications of this method are described in Sections 8.4, 8.5, 8.6, 8.7, and 8.8. These sections are preceded by section 8.2 devoted to a combinatorial enumerative properties of repetitions. Bounding the maximal number of repetitions is necessary for proving complexity bounds of corresponding search algorithms.

8.1. Definitions and preliminary results

Consider a word $w = a_1 \cdots a_n$. A *position* w is any integer ℓ with $1 \le \ell \le n$. Any word x such that there exist $i \le j$ with $x = a_i \cdots a_j$ is called a *factor* of w. If a specific pair of integers (i, j) with this property is meant, we speak about an *occurrence* of the factor x. We say that this occurrence starts at position i and ends at position j in w and denote it $w[i..j]$. We say that a factor occurrence $v = w[i..j]$ *contains* a position ℓ of w, if $i \le \ell \le j$.

Recall (see Chapter 1) that an integer p is called *a period* of w if $a_i = a_{i+p}$, for all i such that $1 \le i, i+p \le n$. Equivalently, p is a period of w iff $w[1..n-p] = w[p+1..n]$. If p is a period of w, any factor u of w of length p is called a *root* of w. In other words, u is a root of w iff w is a factor of u^n for some natural n.

Each word w has a *minimal period* that we will denote $p(w)$. The roots u of w such that $|u| = p(w)$ are called *cyclic roots*. We also call the cyclic root $w[1..p(w)]$ the *prefix cyclic root* of w, and the cyclic root $w[n - p(w) + 1..n]$ the *suffix cyclic root* of w.

The rational number $e(w) = |w|/p(w)$ is called *the exponent* of w. If $e(w) \ge 2$, then w is called a *repetition* (*periodicity*). If $k = e(w)$ is an integer greater than 1, w can be written as $u^k = uu \cdots u$ (k times) and is called an *integer power* (k-power, or *tandem array* in biological literature). A word which is not an integer power is called *primitive*. An integer power of even exponent is commonly called a *square* (or a *tandem repeat*). These are words of the form uu for some word u. An integer power of exponent 2 is called a *primitively-rooted square*, as it corresponds to a square uu where u is primitive. In general, any word w of minimal period p and exponent e can be written as $u^k v$, where u is a primitive word, $|u| = p$, v is a proper prefix of u and $e = k + \frac{|v|}{|u|}$.

The following proposition specifies some properties of repetitions.

Proposition 8.1.1. *A word r of length m is a repetition of minimal period $p \le m/2$ if and only if one of the following conditions holds:*

(i) $r[1..m-p] = r[p+1..m]$, and p is the minimal number with this property,

(ii) any factor of r of length $2p$ is a square, and p is the minimal number with this property.

From now on, we will be interested in repetitions occurring as factors of some word, that is in factor occurrences $r = w[i..j]$ with $e(r) \geq 2$. A *maximal repetition* in a word w, is a repetition $r = w[i..j]$ such that
(i) if $i > 1$, then $p(w[i-1..j]) > p(w[i..j])$, and
(ii) if $j < n$, then $p(w[i..j+1]) > p(w[i..j])$.
In other words, a maximal repetition is a repetition $r = w[i..j]$ such that no factor of w that contains r as a proper factor has the same minimal period as r. For example, the factor 10101 in the word $w = 1011010110110$ is a maximal repetition (with period 2), while the factor 1010 is not. Other maximal repetitions of w are prefix 10110101101 (period 5), suffix 10110110 (period 3), prefix 101101 (period 3), and the three occurrences of 11 (period 1). (Note that different repetitions can be equal words.)

Any repetition in a word can be extended to a unique maximal repetition, that we will call the *corresponding* maximal repetition. For example, the repetition 1010 in word $w = 1011010110110$ corresponds to the maximal repetition 10101 obtained by the extension by one letter to the right.

A basic result about periods is the Fine and Wilf's theorem:

Theorem 8.1.2 (Fine and Wilf). *If w has periods p_1, p_2, and $|w| \geq p_1 + p_2 - gcd(p_1, p_2)$, then $gcd(p_1, p_2)$ is also a period of w.*

Two factor occurrences $w[i..j]$ and $w[k..\ell]$ are said to *overlap* if their intervals $[i..j]$ and $[k..\ell]$ have a non-empty intersection. The *overlap* of the two factors is then the factor $w[r..s]$ where $[r..s]$ is the intersection of $[i..j]$ and $[k..\ell]$. The following lemma will be used in the sequel.

Lemma 8.1.3. *Two distinct maximal repetitions with the same period p cannot have an overlap of length greater than or equal to p.*

Proof. From a case analysis of relative positions of two repetitions of period p, it follows that if they intersect on at least p letters, at least one of them is not maximal. ∎

A repetition r occurring in a word w is said to *have a root* in some factor occurrence of w, if r overlaps with this occurrence by at least $p(r)$ letters. Also, we say that a repetition r *has a root on the right (respectively on the left) of a position ℓ of w* with the meaning that r overlaps by at least $p(r)$ letters with the suffix $w[\ell+1..n]$ (respectively, the prefix $w[1..\ell-1]$).

The following reformulation of Lemma 8.1.3 will be also useful.

Corollary 8.1.4. *If two maximal repetitions of w contain a position ℓ and have a root on the right (on the left) of ℓ, then either these maximal repetitions have different minimal periods or they coincide.*

We will also use the following proposition which is again a consequence of the Fine and Wilf theorem.

PROPOSITION 8.1.5. *If u is a primitive word, then u cannot be an internal factor of uu (that is, a factor which is not a prefix or suffix).*

Proof. If u is an internal factor of uu, then $u = xy = yx$, where x, y are nonempty words. We show that x and y are powers of the same word, by induction on the length of xy. The claim holds if $|x| = |y|$. Otherwise, assume $|x| > |y|$. Then $x = yz$ for some nonempty word z. It follows that $zy = yz$. By induction, y and z are powers of the same word, and so are x and y. Thus, u is not primitive. ∎

8.2. Counting maximal repetitions

Before considering algorithmic issues related to repetitions, we briefly study in this section the combinatorial question of the number of repetitions occurring in a word. The main point here is to show that considering maximal repetitions leads to a compact linear-space representation of all repetitions. This result will motivate building an efficient algorithm for computing all maximal repetitions, and, on the other hand, will be used to obtain bounds on its algorithmic complexity.

8.2.1. Counting squares: an upper bound

We start with the question of how many square occurrences can a word contain. Clearly, if all squares are counted, a word can contain a quadratic number of those (e.g. 1^n). If we restrict our attention to primitively-rooted squares, the following result holds.

THEOREM 8.2.1. *The number of occurrences of primitively-rooted squares in a word of length n is $O(n \log n)$.*

The proof is based on the following "lemma of three squares":

LEMMA 8.2.2. *Consider three squares x^2, y^2, z^2 and assume that z^2 is a prefix of y^2 and y^2 is a prefix of x^2. Assume that z is a primitive word. Then $|z| < |x|/2$.*

Proof. Denote $w = x^2$. Assume that $|z| \geq |x|/2$. Let $p = |y| - |z|$ and $k = |z| - p = 2|z| - |y|$. Observe that prefix $w[1..k]$ of z is also a suffix of z, and therefore $w[1..k] = w[p+1..k+p] = w[p+1..|z|]$. We show by induction that for every i, $k+1 \leq i \leq k+p$, we have $w[i] = w[i+p]$. We have $w[i] = w[i+|y|] = w[i+|y|-|x|]$, as $i+|y| > 2|z| \geq |x|$. By induction, the latter is equal to $w[i+|y|-|x|+p] = w[i+|y|+p] = w[i+p]$. We showed that z (and even z^2) occurs at position $p+1$ in w. By Proposition 8.1.5, this contradicts that z is primitive. ∎

Proof of Theorem 8.2.1 Lemma 8.2.2 implies that the number of primitively-rooted squares occurring as prefixes of a word w is less than $2\log_2 |w|$. Indeed, from Lemma 8.2.2, it follows that if w has as its prefixes i primitively-rooted squares, then $|w| > 2^{i/2}$. Theorem 8.2.1 follows. ∎

8.2.2. Repetitions in Fibonacci words

Fibonacci words are words over the binary alphabet $\{0, 1\}$ defined recursively by $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1}f_{n-2}$ for $n \geq 2$. The length of f_n, denoted F_n, is the n-th Fibonacci number. Fibonacci words have numerous interesting combinatorial properties and often provide a good example to test conjectures and analyze algorithms on words.

The following lemma summarizes some properties of Fibonacci words that we will need in this section.

Lemma 8.2.3. *(i) For $n \geq 2$, f_n and the word $f_{n-2}f_{n-1}$ share a common prefix of length $F_n - 2$.*
(ii) For every n, f_n is a primitive word.
(iii) Every repetition occurring in Fibonacci words has the minimal period F_k for some k, and has a cyclic root f_k.
(iv) Fibonacci words contain no repetition of exponent 4.

Proof. (i) is easily proved by induction. To prove (ii), observe that for $n \geq 2$, f_n contains F_{n-1} occurrences of 1, as can be easily proved by induction. If f_n is not primitive that is $f_n = w^k$ for a non-empty word w and $k \geq 2$, then both $F_n = |f_n|$ and F_{n-1} (the number of 1's in f_n) are divisible by k which is a contradiction as F_n and F_{n-1} are mutually prime, which can be again easily proved by induction. Proving (iii) and (iv) is beyond the scope of this chapter (see Notes). ∎

We now prove that Fibonacci words realize the asymptotically largest number of square occurrences, according to Theorem 8.2.1. This shows, in particular, that the $O(n \log n)$ bound of Theorem 8.2.1 is asymptotically tight.

Theorem 8.2.4. *Fibonacci word f_n contains $\Theta(F_n \log F_n)$ occurrences of primitively-rooted squares.*

Proof. Let S_n be the number of occurrences of primitively-rooted squares in f_n. By induction on n, we will show that $S_n \geq \frac{1}{6}F_n \log_2 F_n$, for $n \geq 5$. For $n = 5$ and $n = 6$, the inequality is verified directly. Assume now that $n \geq 7$. Consider the decomposition $f_n = f_{n-1}f_{n-2}$ and call the position between f_{n-1} and f_{n-2} the *frontier*. Clearly, all squares in f_n are divided into those which lie entirely in f_{n-1} or f_{n-2} and those which *cross the frontier*, i.e. overlap both with f_{n-1} and with f_{n-2}. Therefore,

$$S_{n+1} = S_n + S_{n-1} + \widehat{S}_{n+1},$$

where $\widehat{S}_{n+1}$ is the number of primitively-rooted squares crossing the frontier. By Lemma 8.2.3(i), the prefix of f_{n+1} of length $(F_{n+1}-2)$ is also a prefix of the word $f_{n-1}f_n = f_{n-1}f_{n-1}f_{n-2}$. Since f_{n-2} is a prefix of f_{n-1}, then f_{n+1} contains $(F_{n-2}-1)$ squares of period F_{n-1} that cross the frontier. Since f_{n-1} is a primitive word (Lemma 8.2.3(ii)), all those squares are primitively-rooted. Since

$$f_{n+1} = f_n f_{n-1} = f_{n-1}f_{n-2}f_{n-2}f_{n-3}$$

and f_{n-3} is a prefix of f_{n-2}, then the suffix $f_{n-2}f_{n-2}f_{n-3}$ of f_{n+1} contains $F_{n-3}+1$ squares of period F_{n-2} that cross the frontier. Since f_{n-2} is a primitive word, all those squares are primitively-rooted too.

We obtain that $\widehat{S}_{n+1} \geq (F_{n-2}-1)+(F_{n-3}+1) = F_{n-1}$. Therefore,

$$\begin{aligned} S_{n+1} &\geq S_n + S_{n-1} + F_{n-1} \geq \frac{1}{6}F_n \log_2 F_n + \frac{1}{6}F_{n-1}\log_2 F_{n-1} + F_{n-1} \\ &= \frac{F_{n+1}}{6}\Big[\frac{F_n}{F_{n+1}}(\log_2 \frac{F_n}{F_{n+1}} + \log_2 F_{n+1}) + \frac{F_{n-1}}{F_{n+1}}(\log_2 \frac{F_{n-1}}{F_{n+1}} + \log_2 F_{n+1}) \\ &\quad + \frac{6F_{n-1}}{F_{n+1}}\Big] = \frac{F_{n+1}}{6}(\log_2 F_{n+1} + \Delta), \end{aligned}$$

where

$$\Delta = \frac{F_n}{F_{n+1}}\log_2 \frac{F_n}{F_{n+1}} + \frac{F_{n-1}}{F_{n+1}}\log_2 \frac{F_{n-1}}{F_{n+1}} + \frac{6F_{n-1}}{F_{n+1}}.$$

Both the first and the second term of the expression Δ is greater than or equal to $\min_{0<x<1}(x\log_2 x) = -\frac{\log_2 e}{e}$. The third term is greater than $3/2$, since $F_{i+1} < 2F_i$ for every i. We derive that

$$\Delta > -\frac{2\log_2 e}{e} + \frac{3}{2} > -\frac{3}{e} + \frac{3}{2} > 0.$$

We conclude that $S_{n+1} \geq \frac{F_{n+1}}{6}\log_2 F_{n+1}$. The upper bound follows from Theorem 8.2.1. ■

In view of Theorem 8.2.1, one might want to estimate the maximal number of occurrences of primitively-rooted integer powers in a word, and conjecture that this number is asymptotically smaller than the number of square occurrences, since each primitively-rooted integer power of exponent e represents $e-1$ primitively-rooted squares. In particular, one might want to count the occurrences of *non-extensible* primitively-rooted integer powers, that is those primitively-rooted integer powers u^k, $k \geq 2$, which are not followed or preceded by another occurrence of u. Fibonacci words provide a counter-example to this hypothesis, as by Lemma 8.2.3, all integer powers contained in them are of exponent two or three, and therefore the maximal number of their occurrences is still $\Theta(F_n \log F_n)$, as implied by Theorem 8.2.4.

What happens if we count maximal repetitions instead of occurrences of integer powers or just squares? Note that a word can contain much less maximal repetitions than integer powers: e.g. if v is a square-free word over a three-letter alphabet, then word $v\$v\$v$ contains $|v|+1$ non-extensible integer powers (here squares) but only one maximal repetition. What is the number of maximal repetitions in Fibonacci words? The following theorem gives the exact answer.

THEOREM 8.2.5. *Let R_n be the number of maximal repetitions in f_n. Then for all $n \geq 4$, $R_n = 2F_{n-2} - 3$.*

Proof. As in the proof of Theorem 8.2.4, we divide all maximal repetitions in f_n into those which lie entirely in f_{n-1} or f_{n-2} and those which cross the frontier, i.e. overlap with f_{n-1} and with f_{n-2}. We call such repetitions *crossing repetitions* of f_n. Overlaps of a crossing repetition with f_{n-1} and f_{n-2} are called the *left part* and the *right part* respectively. Note first that the left part and the right part of a maximal repetition cannot be both of exponent greater than or equal to 2, since Fibonacci words don't have factors of exponent 4 (Lemma 8.2.3(iv)). If either the left or the right part is of exponent at least 2, then this repetition is an extension of a maximal repetition of respectively f_{n-1} or f_{n-2}. This implies that the only new maximal repetitions of f_n that should be counted are crossing maximal repetitions with both right and left part of exponent smaller than 2. We call those repetitions *composed* repetitions of f_n. Let $c(n)$ be their number. The following lemma allows to compute $c(n)$.

LEMMA 8.2.6. *Let R_n be the number of occurrences of maximal repetitions in the Fibonacci word f_n and set $R_n = R_{n-1} + R_{n-2} + c(n)$. Then for all $n \geq 8$, $c(n) = c(n-2)$.*

Proof. Consider the representation

$$f_n = f_{n-1}|f_{n-2} = f_{n-2}f_{n-3}|f_{n-3}f_{n-4} = f_{n-2}[f_{n-3}|f_{n-4}]f_{n-5}f_{n-4} \qquad (8.2.1)$$

where $|$ denotes the frontier, $n \geq 5$, and square brackets delimit the occurrence of f_{n-2} with the same frontier as for the whole word f_n (we call it the *central occurrence* of f_{n-2}). By Lemma 8.2.3(iii), the minimal period of every repetition in Fibonacci words is equal to F_k for some k. Since $F_{n-3} > F_{n-4} > 2F_{n-6}$, it follows from (8.2.1) that if a composed repetition of f_n has the minimal period F_k for $k \leq n-6$, then it is also a composed repetition of f_{n-2} and therefore is counted in $c(n-2)$. Vice versa, every composed repetition of f_{n-2} with the period F_k for $k \leq n-6$, is also a composed repetition of f_n. We now examine crossing maximal repetitions of f_n with minimal periods F_{n-2}, F_{n-3}, F_{n-4}, F_{n-5}.

Crossing repetitions with the minimal period F_{n-2}. The last term of (8.2.1) shows that square $(f_{n-2})^2$ is a prefix of f_n that crosses the frontier. As $F_{n-1} < 2F_{n-2}$, the corresponding maximal repetition does not have a square in its left or right part and therefore is composed for f_n. Since $F_{n-2} > F_n/3$, any two maximal repetitions of f_n with the period F_{n-2} overlap by more than F_{n-2} letters. By Lemma 8.1.3, f_n has only one maximal repetition with the period F_{n-2}. Trivially, the repetition under consideration is not a repetition for the central occurrence of f_{n-2}.

Crossing repetitions with the minimal period F_{n-3}. From the decomposition $f_n = f_{n-2}f_{n-3}|f_{n-3}f_{n-4}$ (see (8.2.1)), there is a square $(f_{n-3})^2$ of period F_{n-3} crossing the frontier. The corresponding maximal repetition does not extend to the left of the left occurrence of f_{n-3}, as the last letters of f_{n-3} and f_{n-2} are different (the last letters of f_i's alternate). Therefore, this repetition does not have a square in its left or right part, and thus is composed for f_n. As this maximal repetition has a root both on the left and on the right of the frontier, it is the only repetition with the period F_{n-3} crossing the frontier (see Corollary 8.1.4). Again, from length considerations, it is not a maximal repetition for the central occurrence of f_{n-2}.

Crossing repetitions with the minimal period F_{n-4}. Since

$$\begin{aligned} f_n &= f_{n-1}|f_{n-4}f_{n-5}f_{n-4} = f_{n-1}|f_{n-4}f_{n-5}f_{n-5}f_{n-6} \\ &= f_{n-1}|f_{n-4}f_{n-5}f_{n-6}f_{n-7}f_{n-6} = f_{n-1}|(f_{n-4})^2 f_{n-7}f_{n-6}, \end{aligned}$$

there is a square $(f_{n-4})^2$ on the right of the frontier. However, this maximal repetition does not extend to the left (i.e. does not cross the frontier), since the last letters of f_{n-4} and f_{n-1} are different.

On the other hand,

$$\begin{aligned} f_n &= f_{n-2}[f_{n-3}|f_{n-4}]f_{n-5}f_{n-4} = f_{n-3}f_{n-4}[f_{n-4}f_{n-5}|f_{n-5}f_{n-6}]f_{n-5}f_{n-4} \\ &= f_{n-3}f_{n-4}[f_{n-4}\underbrace{f_{n-5}|f_{n-6}}_{f_{n-4}}f_{n-7}f_{n-6}]f_{n-5}f_{n-4}, \text{ for } n \geq 6. \end{aligned}$$

This reveals a maximal repetition with the period F_{n-4} which crosses the frontier. However, this is not a composed repetition of f_n, as it has a square on the left of the frontier. On the other hand, the restriction of this repetition to the central occurrence of f_{n-2} is a composed repetition for f_{n-2}.

There is no other repetition with the period F_{n-4} crossing the frontier, since such would overlap with one of the two above by more than one period, which would contradict Lemma 8.1.3. In conclusion, there is one composed repetition with the period F_{n-4} in the central occurrence of f_{n-2} and no such repetition in f_n.

Crossing repetitions with the minimal period F_{n-5}. Rewrite

$$f_n = f_{n-2}[f_{n-4}f_{n-5}|f_{n-5}f_{n-6}]f_{n-5}f_{n-4}$$

which shows that there is a square of period F_{n-5} crossing the frontier. Since the frontier is the center of this square, the latter corresponds to the only crossing repetition with the period F_{n-5}. However, this repetition is not a composed repetition for f_n, as it has a square in its right part, as shown by the following transformation:

$$\begin{aligned} f_n &= f_{n-2}[f_{n-4}f_{n-5}|f_{n-5}f_{n-6}]f_{n-6}f_{n-7}f_{n-4} \\ &= f_{n-2}[f_{n-4}f_{n-5}|f_{n-5}\underbrace{f_{n-6}]f_{n-7}}_{f_{n-5}}f_{n-8}f_{n-7}f_{n-4}, \text{ for } n \geq 8. \end{aligned}$$

On the other hand, the restriction of this repetition to the central occurrence of f_{n-2} is a composed repetition for f_{n-2}. Thus, there is one composed maximal repetition with the period F_{n-5} in the central occurrence of f_{n-2} and no such repetition in f_n.

In conclusion, two new composed repetitions arise in f_n in comparison to f_{n-2}, but two composed maximal repetitions of the central occurrence of f_{n-2} are no more composed in f_n, as they extend in f_n to form a square in its right or left part. This shows that $c(n) = c(n-2)$ for $n \geq 8$. ■

Proof of Theorem 8.2.5 (continued): A direct counting shows that $R_0 = 0,\ R_1 = 0,\ R_2 = 0,\ R_3 = 0,\ R_4 = 1,\ R_5 = 3,\ R_6 = 7,\ R_7 = 13$. Therefore, $c(3) = 0,\ c(4) = 1,\ c(5) = 2,\ c(6) = 3,\ c(7) = 3$. Since $c(n) = c(n-2)$ for all $n \geq 8$, then $c(n) = 3$ for all $n \geq 6$. We then have the recurrence relation $R_n = R_{n-1} + R_{n-2} + 3$ for $n \geq 6$ with boundary conditions $R_4 = 1,\ R_5 = 3$. Substituting $R_n = 2R'_{n-2} - 3$, we get the relation $R'_n = R'_{n-1} + R'_{n-2}$ for $n \geq 4$, where $R'_2 = 2,\ R'_3 = 3$. This defines exactly the Fibonacci numbers. Thus, $R'_n = F_n$ for $n \geq 2$, and $R_n = 2F_{n-2} - 3$ for $n \geq 4$. ■

Note that by Lemma 8.2.3(iv), any maximal repetition in a Fibonacci word has the exponent smaller than 4, and therefore not only the number of maximal repetitions is linearly bounded, but also the sum of their exponents is linearly bounded on the word length.

8.2.3. Counting maximal repetitions

The results on Fibonacci words suggest a conjecture that arbitrary words contain only a linear number of maximal repetitions and moreover, that the sum of their exponents is linearly-bounded too. In this section we confirm these conjectures. Later in Section 8.4, this result will allows us to derive a linear-time algorithm for identifying all maximal repetitions in a word.

THEOREM 8.2.7. *Let $\mathcal{R}(w)$ be the set of all maximal repetitions in a word w of length n (over an arbitrary alphabet), and let $E(w) = \sum_{r \in \mathcal{R}(w)} e(r)$. Then $E(w) = O(n)$.*

The existing proof of Theorem 8.2.7 is very technical and is done by a tedious case analysis. We don't include it here and refer the reader to Kolpakov and Kucherov 2000b. Finding a simple proof of Theorem 8.2.7 remain an open problem.

As a corollary of Theorem 8.2.7, we obtain that the number of maximal repetitions in a word over an arbitrary alphabet is linearly-bounded in the length of the word.

THEOREM 8.2.8. *Let $\mathcal{R}(w)$ be the set of all maximal repetitions in a word w of length n. Then* $\mathrm{Card}\,(\mathcal{R}(w)) = O(n)$.

Together with the previous results of Section 8.2, this confirms that the set of maximal repetitions is a more compact representation of all repetitions than

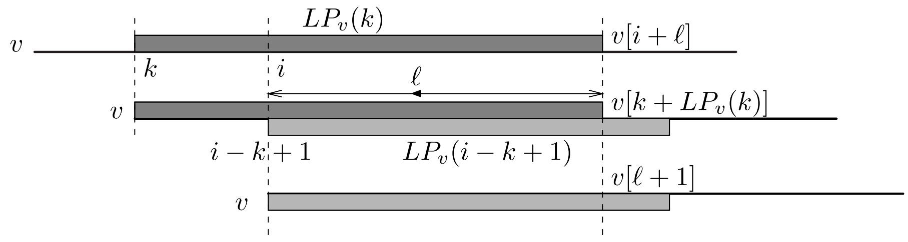


Figure 8.1. Case $LP_v(i-k+1) > \ell$

the set of primitively-rooted squares or primitively-rooted non-extensible integer powers.

8.3. Basic algorithmic tools

From now on, we will be interested in the algorithmic question: How to efficiently compute different types of repetitions in a given word? In the following sections, we present algorithms that allow to compute efficiently all maximal repetitions, as well as other types of repetitions. All those algorithms are based on a common general technique, and therefore a "secondary goal" of this chapter is to demonstrate the power of this approach. The technique is based on two main tools that we describe in this section.

8.3.1. Longest extension functions

The first tool is longest extension functions. Computing those functions (which are basically arrays of integer values, indexed by word positions) is an important component of the algorithms to be presented. Longest extension functions come in different variants – here we present a basic formulation and we will refer to it afterwards.

Consider a word v of length n. For each position i of v, we want to compute the longest factor of v which starts at position i and is also a prefix of v. Formally, we want to compute, for all $i \in [1..n]$, the value $LP_v(i)$ defined as maximal $\ell > 0$ such that $v[1..\ell] = v[i..i+\ell-1]$ and $LP_v(i) = 0$ if no such positive ℓ exists. Note that $LP_v(1) = n$ and, by convention, we always set $LP_v(n+1) = 0$. For example, for $v = 101101011011$, the values of $LP_v(i)$ for $i = 1, \ldots, 13$ are respectively $12, 0, 1, 3, 0, 6, 0, 1, 4, 0, 1, 1, 0$.

We now describe an algorithm that computes LP_v in $O(n)$ time. The algorithm processes v from left to right and computes $LP_v(i)$ for all positions i successively. The computation is based on the following idea. Assume we have computed $LP_v(j)$ for all $j < i$, and we are about to compute $LP_v(i)$. Assume we have stored a position $k < i$ that maximizes $k + LP_v(k)$, and assume that $k + LP_v(k) > i$. Set $\ell = k + LP_v(k) - i$ and consider the value $LP_v(i-k+1)$. If

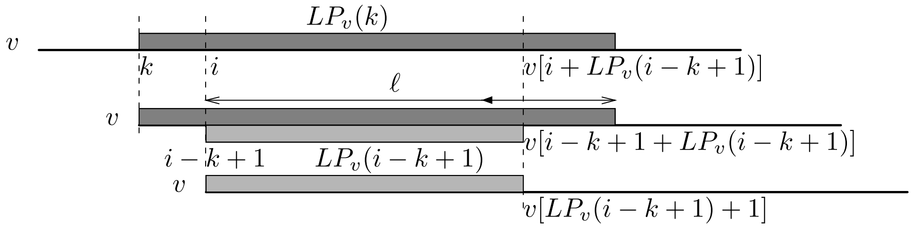


Figure 8.2. Case $LP_v(i - k + 1) < \ell$

$LP_v(i-k+1) > \ell$ (see Figure 8.1), then we claim that $LP_v(i) = \ell$. Indeed, it is easily seen that $v[i..i+\ell-1] = v[1..\ell]$. On the other hand, $v[\ell+1] \neq v[i+\ell]$, as $v[i+\ell] = v[k+LP_v(k)] \neq v[LP_v(k)+1]$, and $v[LP_v(k)+1] = v[\ell+1]$. Therefore, $LP_v(i) = \ell$.

If $LP_v(i-k+1) < \ell$ (see Figure 8.2), then we show that $LP_v(i) = LP_v(i-k+1)$. Again, $v[i..i+LP_v(i-k+1)-1] = v[1..LP_v(i-k+1)]$, but $v[i+LP_v(i-k+1)] \neq v[LP_v(i-k+1)+1]$, since $v[i+LP_v(i-k+1)] = v[i-k+1+LP_v(i-k+1)]$ and $v[i-k+1+LP_v(i-k+1)] \neq v[LP_v(i-k+1)+1]$.

Putting together the two cases above, if $LP_v(i-k+1) \neq \ell$, then $LP_v(i) = \min\{\ell, LP_v(i-k+1)\}$. The only case when $LP_v(i)$ cannot be computed immediately is $LP_v(i-k+1) = \ell$. In this case, we keep on reading letters $v[i+\ell], v[i+\ell+1], \dots$ while $v[i+\ell+j] = v[\ell+j]$, and thus compute the value $LP_v(i)$. Position i is then stored as the new value of k. The resulting linear-time algorithm Longest-Prefix-Extension is shown below.

Function LP_v can be generalized to compute, for all positions of v, the longest factor that starts at this position and is a prefix of *another* fixed word. Formally, for two words $v[1..n]$ and $w[1..m]$, we define $LP_{v|w}$ to be the function associating with every position $i \in [1..n]$ of v the maximal length of a factor of word vw which starts at position i and is a prefix of w. Note that this factor starts inside v but can overlap with w. All values of $LP_{v|w}$ can be computed in time $O(m+n)$.

```
Longest-Prefix-Extension(v[1..n])
LP(1) ← n
j ← 0
while j ≤ n − 2 and v[j + 1] = v[j + 2] do
    j ← j + 1
LP(2) ← j
k ← 2
for i ← 3 to n do
    ℓ ← k + LP(k) − i
    if LP(i − k + 1) ≠ ℓ and ℓ ≥ 0 then
        LP(i) ← min(ℓ, LP(i − k + 1))
    else j ← max(0, ℓ)
        while i + j ≤ n and v[i + j] = v[j + 1] do
            j ← j + 1
        LP(i) ← j
        k ← i
return LP
```

Symmetrical functions can be defined with respect to suffixes instead of prefixes. For a word $v[1..n]$, we define $LS_v(i)$ to be the length of the longest factor of v that ends at position i and is a suffix of v. For $v[1..n]$ and $w[1..m]$, $LS_{w|v}(i)$, $i \in [1..n]$ is the maximal length of a factor of wv that ends at position $m + i$ in wv and is a suffix of w. Both these functions can be computed in time linear in the involved words, using an algorithm similar to Longest-Prefix-Extension. In particular, the function computing $LS_{w|v}$ will be called Longest-Suffix-Extension(w, v) in the sequel.

8.3.2. s-factorization and Lempel-Ziv factorization

The second basic algorithmic tool is a special factorization of the word, that will allow to speed up our repetition-finding algorithms. There are several variants of this factorization that we will use in the following sections. The difference is of technical nature and the choice of the definition will be basically guided by the convenience in describing the algorithm. We distinguish two factorizations that we call *s-factorization* and *Lempel-Ziv factorization*, following the terms under which they have been introduced in the literature. Each of those factorizations comes in two variants, thus yielding four different definitions.

Let w be an arbitrary word. The *s-factorization* of w (respectively, *s-factorization with non-overlapping copies*) is the factorization $w = f_1 f_2 \cdots f_k$, where f_i's are defined inductively as follows:

- $f_1 = w[1]$, and if letter a occurring in w immediately after $f_1 f_2 \cdots f_{i-1}$ does not occur in $f_1 f_2 \cdots f_{i-1}$, then $f_i = a$.
- otherwise, f_i is the longest factor occurring in w immediately after $f_1 f_2 \cdots f_{i-1}$ that occurs in $f_1 f_2 \cdots f_{i-1} f_i$ other than as a suffix (respectively, that occurs in $f_1 f_2 \cdots f_{i-1}$).

The *Lempel-Ziv factorization of* w (respectively, *Lempel-Ziv factorization with non-overlapping copies*) is the factorization $w = f_1 f_2 \cdots f_k$, where f_i's are defined inductively as follows:

- $f_1 = w[1]$,
- for $i \geq 2$, f_i is the shortest factor occurring in w immediately after $f_1 f_2 \cdots f_{i-1}$ that does not occur in $f_1 f_2 \cdots f_{i-1} f_i$ other than as a suffix (respectively, that does not occur in $f_1 f_2 \cdots f_{i-1}$ at all).

In the s-factorization of the word w, we look for the longest factor f_i starting after $f_1 f_2 \cdots f_{i-1}$ which has a copy on the left. In the s-factorization with non-overlapping copies, this copy is required to be non-overlapping with f_i. In the Lempel-Ziv factorization, we look for the shortest word that does not have an occurrence on the left. In other words, we extend by one letter the longest word which does have a copy on the left. If the Lempel-Ziv factorization with non-overlapping copies is considered, then the copy is required not to overlap with f_i.

As an example, consider the word $w = 1100101010000$. Its s-factorization is $1|1|0|0|10|1010|000$, and the s-factorization with non-overlapping copies of w is $1|1|0|0|10|10|100|00$. The Lempel-Ziv factorization of w is $1|10|01|010100|00$ and the Lempel-Ziv factorization without copy overlap is $1|10|01|010|1000|0$.

If $w = f_1 f_2 \cdots f_k$ is the s-factorization (respectively, Lempel-Ziv factorization), we call each f_i an *s-factor* (respectively, *LZ-factor*) of w.

A remarkable feature of all considered factorizations is that each of them can be computed in a time linear in the length of the word. This can be done in different ways, using a data structure like the suffix tree or the DAWG (Directed Acyclic Word Graph). A possible algorithm consists in computing the factorization along with constructing the data structure in an on-line fashion. A factorization provides a very useful information about the structure of repeated factors and the possibility to compute the factorization in linear time makes of it a key algorithmic tool that we will use throughout the rest of this chapter.

8.4. Finding all maximal repetitions in a word

According to the results of Section 8.2, maximal repetitions are important structures, as they encode, in a most compact way, all repetitions occurring in the word. If the set of maximal repetitions is known, repetitions of any other type can be extracted from it: primitively- or non-primitively rooted squares, cubes, etc.

In this section, we show that the set of all maximal repetitions can be computed very efficiently, namely in a time linear in the length of the word. The linear time bound is supported by Theorems 8.2.7, 8.2.8 that guarantee that the output itself is of linear size (assuming that each maximal repetition is represented in constant space, e.g. by the start position, the period and the total length).

We first consider the following auxiliary problem. Assume we are given two words $x = x[1..m]$, $y = y[1..n]$, and consider their concatenation $v = xy =$

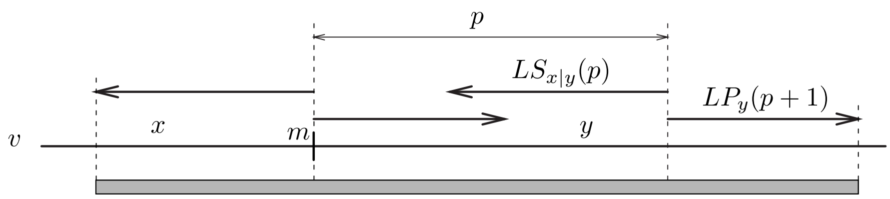


Figure 8.3. Illustration to Theorem 8.4.1

$v[1..m+n]$. We want to find all maximal repetitions $r = v[i..j]$ in the word v that *contain the frontier between* x *and* y, i.e. such that $i \leq m+1$ and $j \geq m$. Every such repetition belongs (non-exclusively) to one of two classes: the repetitions which have a root in y and those which have a root in x. Note that by Corollary 8.1.4, for every p, $1 \leq p \leq n$, there is at most one maximal repetition with a period p that contains the frontier between x and y and has a root in y. This shows, in particular, that the number of such repetitions is not greater than n. Similarly, the number of repetitions that contain the frontier and have a root in x is not greater than m, and thus, the number of maximal repetitions in $v = xy$ which contain the frontier between x and y is bounded by $(m+n)$.

Let us focus on maximal repetitions r which have a root in y, those which have a root in x are found similarly. Consider longest extension functions LP_y and $LS_{x|y}$ (see Section 8.3). The following theorem holds.

Theorem 8.4.1. *For $1 \leq p \leq n$, there exists a maximal repetition with a period p in $v = xy$ that contains the frontier between x and y and has a root in y iff*

$$LS_{x|y}(p) + LP_y(p+1) \geq p. \tag{8.4.1}$$

If the inequality holds, this maximal repetition is $v[m - LS_{x|y}(p)+1..m+p+LP_y(p+1)]$ (see Figure 8.3).

Proof. Assume there is a square $s = v[\ell..\ell+2p-1]$ of period p that contains the frontier between x and y, i.e. such that $\ell \leq m+1$ and $\ell+2p-1 \geq m$. Assume that s has a root in y, i.e. $\ell+p > m$. Observe that prefix $y[1..\ell+p-m-1]$ of y is equal to $y[p+1..\ell+2p-m-1]$. This implies that $LP_y(p+1) \geq \ell+p-m-1$. On the other hand, suffix $x[\ell..m]$ of x is equal to $v[\ell+p..m+p]$, and then $LS_{x|y}(p) \geq m-\ell+1$. Inequality (8.4.1) follows.

On the other hand, if (8.4.1) holds, then any factor $v[\ell..\ell+2p-1]$ with $m - LS_{x|y}(p)+1 \leq \ell \leq m+LP_y(p+1)-p+1$ is a square of period p. Therefore, $r = v[m-LS_{x|y}(p)+1..m+p+LP_y(p+1)]$ is a repetition. It remains to see that r is maximal, i.e. extended to the right and left as far as possible. This follows from the definition of the longest extension functions LS and LP. ∎

Theorem 8.4.1 yields an algorithm for computing all maximal repetitions which contain the frontier between x and y and have a root in y. The algorithm, called Right-Repetitions, is shown below. It assumes that each repetition is represented by a pair of its start and end positions.

Right-Repetitions(x, y)
$LP_y \leftarrow$ Longest-Prefix-Extension(y)
$LS_{x|y} \leftarrow$ Longest-Suffix-Extension(x, y)
$\mathcal{R} \leftarrow \emptyset$
for $p \leftarrow 1$ **to** $|y|$ **do**
  if $LS_{x|y}(p) + LP_y(p+1) \geq p$ **then**
    $r \leftarrow (m - LS_{x|y}(p) + 1, m + p + LP_y(p+1))$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
return $\mathcal{R}$

The repetitions containing the frontier between x and y and having a root in x are computed similarly (function Left-Repetitions hereafter). As the functions Longest-Prefix-Extension(y) and Longest-Suffix-Extension(y) run in time $O(|y|)$ and $O(|x|+|y|)$ respectively, all maximal repetitions in $v = xy$ which contain the frontier between x and y can be computed in time $O(|v|)$.

Let us now come back to the problem of computing all maximal repetitions in a word. The s-factorization is another useful tool for building a linear-time algorithm for this problem, due to the following theorem.

Theorem 8.4.2. *Let $w = f_1 f_2 \cdots f_k$ be the s-factorization of w. Let r be a maximal repetition in w that contains the frontier between f_{i-1} and f_i and ends inside f_i. Then the prefix of r which is a suffix of $f_1 \cdots f_{i-1}$ is smaller than $|f_i| + 2|f_{i-1}|$.*

Proof. Assume $r = w[\ell..m]$ and denote b_i the position of the last letter of f_i, i.e. $b_i = |f_1 f_2 \cdots f_i|$. The theorem asserts that if $\ell \leq b_{i-1}+1$ and $b_{i-1}+1 \leq m \leq b_i$, then $b_{i-1} + 1 - \ell \leq |f_i| + 2|f_{i-1}|$.

Consider the suffix cyclic root $r' = w[m - p(r) + 1..m]$ of r. Observe that r' has a copy $p(r)$ letters to the left. If r' starts at a position before the start of f_{i-1}, i.e. $m - p(r) \leq b_{i-2}$, then it includes entirely the factor f_{i-1} and at least the first letter of f_i. This contradicts that f_{i-1} is the *longest* factor occurring on the left, according to the definition of s-factorization. Therefore, $m - p(r) > b_{i-2}$ and then $p(r) = |r'| < |f_{i-1}| + |f_i|$.

On the other hand, r cannot extend to the left of $b_{i-2} + 1$ by $p(r)$ letters or more, as this would again contradict the definition of f_{i-1}. Thus, the part of r before the start of f_{i-1} is bounded by $|f_{i-1}| + |f_i|$. The theorem follows. ■

For the simplicity of presentation, we assume that the last letter of w does not occur elsewhere in w. Given the s-factorization $w = f_1 f_2 \cdots f_k$, we consider, for each s-factor f_i, $i \in [2..k]$, all maximal repetitions that end either at the last position of f_{i-1}, or at some position of f_i except the last one. Formally, these are repetitions $r = w[\ell..m]$ such that $b_{i-1} \le m < b_i$, where $b_i = |f_1 f_2 \cdots f_i|$. Cleary, each maximal repetition of w belongs to exactly one such class. Note that since the last letter of w is unique, the last s-factor f_k consists of one letter and there is no maximal repetitions that occur as suffixes of w.

We further split all considered repetitions into two sets that we call *repetitions of type 1 and 2*:

repetitions of type 1: maximal repetitions r that contain the frontier between f_{i-1} and f_i and end at a position strictly smaller than the end of f_i,

repetitions of type 2: maximal repetitions r that occur properly inside f_i.

Repetitions of type 1 are repetitions $r = w[\ell..m]$ such that either $m = b_{i-1}$, or $b_{i-1} + 1 \le m \le b_i - 1$ and $\ell \le b_{i-1} + 1$. By Theorem 8.4.2, the former cannot extend by more than $|f_{i-1}| + 2|f_{i-2}|$ to the left of f_{i-1}, and therefore its length is bounded by $2|f_{i-1}| + 2|f_{i-2}|$, and the latter cannot extend by more than $|f_i| + 2|f_{i-1}|$ to the left of f_i. Joining both cases together, a repetition of type 1 cannot extend by more than $\max\{2|f_{i-1}| + 2|f_{i-2}|, |f_i| + 2|f_{i-1}|\} = 2|f_{i-1}| + \max\{2|f_{i-2}|, |f_i|\}$ to the left of f_i.

Therefore, to find all repetitions of type 1, we consider the word $t_i f_i$, where t_i is the suffix of $f_1 \cdots f_{i-1}$ of length $2|f_{i-1}| + \max\{2|f_{i-2}|, |f_i|\}$ (t_i is the whole word $f_1 \cdots f_{i-1}$ if its length is smaller than $2|f_{i-1}| + \max\{2|f_{i-2}|, |f_i|\}$). We then have to find in $t_i f_i$ all maximal repetitions that contain the frontier between t_i and f_i and don't include the last letter of f_i. This can be done in time $O(|f_{i-2}| + |f_{i-1}| + |f_i|)$ using longest extension functions, as described above. Summing up over all s-factors, all repetitions of type 1 in w can be found in time $O(n)$.

Every repetition of type 2 occurs entirely inside some s-factor f_i of the s-factorization, and each f_i has an earlier occurrence in w. Therefore, each maximal repetition of type 2 has another occurrence on the left. This implies, in particular, that finding all repetitions of type 1 guarantees finding all *distinct* maximal repetitions, and in particular all *leftmost* occurrences of distinct maximal repetitions.

We are left with the problem of finding all repetitions of type 2. Here is how this can be done.

During the computation of the s-factorization we store, for each s-factor f_i, a pointer to an earlier occurrence of f_i in w. Computing such a pointer does not affect the linear time complexity of s-factorization. Let v_i be this earlier occurrence of f_i, and let Δ_i be the difference between the position of f_i and the position of v_i. Obviously, each repetition of type 2 occurring inside f_i is a copy of a maximal repetition occurring inside v_i shifted by Δ_i to the right.

We first sort, using bucket sort, all maximal repetitions of type 1, found at the first stage, into n lists $end[1], ..., end[n]$ such that list $end[j]$ contains all maximal repetitions with end position j. Then we process all lists $end[j]$ in the increasing order of j and sort the repetitions again, using bucket sort, into n lists $start[1], ..., start[n]$ according to their start position. After this double sorting, the repetitions with the same start position j are sorted inside the list $start[j]$

in the increasing order of their end positions. As there is a linear number of repetitions of type 1, both sorting procedures take a linear time.

We will use the same lists $start[j]$ to store maximal repetitions of type 2. For each s-factor f_i and for each internal position j inside f_i, we have to find all maximal repetitions starting at this position and ending strictly inside f_i. We then have to find all maximal repetitions from the list $start[j - \Delta_i]$ which end inside v_i, and then shift them by Δ_i to the right. Note that these repetitions may be either of type 1, or previously found repetitions of type 2. We look through the list $start[j - \Delta_i]$ and retrieve its prefix consisting of those maximal repetitions which end inside v_i. Then we shift each of these maximal repetitions by Δ_i and append a modified copy of this prefix to the head of the list $start[j]$. Note that the data structure is preserved, as all appended repetitions must end before any of repetitions of type 1 previously stored in the list $start[j]$. Since we process f_i's from left to right, no maximal repetition can be missed. Thus, we recover all repetitions of type 2 and after all f_i's have been processed, the data structure contains all maximal repetitions of both types.

Note that when we retrieve a prefix of the list corresponding to some position in v_i, each repetition in this prefix results in a new maximal repetition of type 2 in f_i. This shows that the time spent on processing the lists is proportional to the number of newly found maximal repetitions. Theorem 8.2.7 states that the number of all maximal repetitions is linear in the length of w. This proves that the whole algorithm takes a linear time.

The whole algorithm for computing all maximal repetitions is summarized below.

```
Maximal-Repetitions(w[1..n])
(f_1, ..., f_k) ← s-factorization of w
▷ first stage
R ← ∅
for i ← 1 to k do
    t_i ← suffix of f_1 ··· f_{i-1} of length 2|f_{i-1}| + max{2|f_{i-2}|, |f_i|}
    R'_i ← Right-Repetitions(t_i, f_i)
    R''_i ← Left-Repetitions(t_i, f_i)
    R ← R ∪ R'_i ∪ R''_i
▷ second stage
for each r = (j, ℓ) ∈ R do
    add r to list end[ℓ]
for ℓ ← 1 to n do
    for each r = (j, ℓ) from list end[ℓ] do
        add r to list start[j]
for i ← 1 to k do
    for j ← b_{i-1} + 1 to b_i do
        for each r from list start[j − Δ_i] such that |r| < b_i − j + 1 do
            add repetition r' = (j, j + |r| − 1) to list start[j]
            R ← R ∪ {r'}
return R
```

The complexity of MAXIMAL-REPETITIONS follows from the above discussion:

THEOREM 8.4.3. MAXIMAL-REPETITIONS *finds all maximal repetitions in a word of length n in time $O(n)$.*

The set of all maximal repetitions provides an exhaustive information about the repetition structure of the word. It allows to extract all repetitions of other types, such as (primitively- or non-primitively-rooted) squares, cubes, or integer powers. Thus, all these tasks can be done in time $O(n+T)$ where T is the output size.

As another application, the set of maximal repetitions allows to determine, in linear time, the number $d_k(i)$ of primitively-rooted integer powers of a given exponent k, starting at each position i of the word. Here is how this can be done. For each position $i \in [1..n]$ of the input word w, we create two counters $b(i)$ and $c(i)$, initially set to 0. For each repetition $r = w[\ell..m]$, we increment $b(\ell)$ and $c(m - kp(r) + 1)$ by 1 ($[\ell..m - kp(r) + 1]$ is the interval, where primitively-rooted k-powers induced by repetition r start). By Theorem 8.2.8, the number of updates is linear. To compute the numbers $d_k(i)$, we scan all positions from left to right applying the following iterative procedure: $d_k(1) = b(1)$, $d_k(i+1) = d_k(i) + b(i) - c(i-1)$, $i = 2..|w|$. Note that the algorithm can be extended to all (not necessarily primitively-rooted) k-powers. In this case, we increment $b(\ell)$ by $\lfloor e(r)/k \rfloor$, and we increment by 1 each $c(j)$, for $j = m - kp(r) + 1, m - 2kp(r) + 1, \ldots, m - \lfloor e(r)/k \rfloor kp(r) + 1$. Here, Theorem 8.2.7 guarantees that the number of updates is linear. Finally, note that the procedure can be easily modified in order to count fractional repetitions of a given exponent, as well as to repetitions ending (or centered) at each position.

8.5. Finding quasi-squares in two words

The linear-time algorithm for computing all maximal repetitions presented in the previous section allows us to compute all squares in a word of length n in time $O(n + S)$, where S is the number of those squares. In this section we consider a problem of finding *quasi-squares* that generalizes the problem of computing usual squares. Besides that the problem of quasi-squares is interesting in its own, it will be used later in Section 8.6.1.

Assume we are given two words u, v of equal length, $|u| = |v| = n$, $n \geq 2$. We say that words u, v contain a *quasi-square* iff for some $1 \leq \ell \leq n$ and $p > 0$, we have $u[\ell..\ell+p-1] = v[\ell+p..\ell+2p-1]$. p is called the *period* of the quasi-square, and words $u[\ell..\ell+p-1], v[\ell+p..\ell+2p-1]$ are called respectively its *left root* and *right root*. Obviously, if $u = v$, then the quasi-squares are usual squares.

Denote $\mathcal{QS}(u, v)$ the set of all quasi-squares of words u, v. We show that $\mathcal{QS}(u, v)$ can be computed in time $O(n \log n + S)$, where $S = \mathrm{Card}\,(\mathcal{QS}(u, v))$. The algorithm we propose is based only on longest extension functions and is similar to the algorithm based on Theorem 8.4.1 for finding maximal repetitions containing a given position. An advantage of the proposed solution is that the output quasi-squares are naturally grouped into runs of quasi-squares of the same period starting at successive positions in the word. This runs are analogous to repetitions for usual non-gapped squares. We will use this feature of the algorithm later in Section 8.6.

Assume $n = 2m$, and denote $\mathcal{QS}_m(u,v)$ the subset of $\mathcal{QS}(u,v)$ consisting of those quasi-squares $(u[\ell..\ell+p-1], v[\ell+p..\ell+2p-1])$ which contain the frontier in the middle of u and of v, more precisely such that $\ell \leq m+1$ and $\ell+2p-1 \geq m$. $\mathcal{QS}_m(u,v)$ is the union of two subsets $\mathcal{QS}^l_m(u,v)$ and $\mathcal{QS}^r_m(u,v)$ consisting of quasi-squares containing the middle frontier in their left root or right root respectively. Consider the set $\mathcal{QS}^l_m(u,v)$ ($\mathcal{QS}^r_m(u,v)$ is treated similarly). $\mathcal{QS}^l_m(u,v)$ consists of quasi-squares $(u[\ell..\ell+p-1], v[\ell+p..\ell+2p-1])$ verifying $m+1-p \leq \ell \leq m+1$.

Consider the following longest extension functions defined on all positions $i \in [1..m]$ of $v[m+1..n]$:

$$\widehat{LP}(i) = \min\{LP_{v[m+1..n]|u[m+1..n]}(i), m-i+1\}$$
$$\widehat{LS}(i) = \min\{LS_{u[1..m]|v[m+1..n]}(i), i\}$$

In words, $\widehat{LP}(i)$ is the length of the longest common prefix of $v[m+i..n]$ and $u[m+1..n]$, and $\widehat{LS}(i)$ is the length of the longest common suffix of $u[1..m]$ and $v[m+1..m+i]$.

Let $u[\ell..\ell+p-1], v[\ell+p..\ell+2p-1]$ be a quasi-square of period p belonging to $\mathcal{QS}^l_m(u,v)$. By an argument similar to Theorem 8.4.1, we have

$$\widehat{LS}(p) + \widehat{LP}(p+1) \geq p. \tag{8.5.1}$$

Vice versa, if for some $p \in [1..m]$, inequality (8.5.1) holds, then there exists a quasi-square of period p from $\mathcal{QS}^l_m(u,v)$. More precisely, the following lemma holds.

LEMMA 8.5.1. *For $p \in [1..m]$, there exists a quasi-square of $\mathcal{QS}^l_m(u,v)$ of period p iff inequality (8.5.1) holds. When (8.5.1) holds, there is a family of quasi-squares of period p from $\mathcal{QS}^l_m(u,v)$, with the left roots starting at each position of the interval*

$$[m+1-\widehat{LS}(p) \;..\; m+1+\min\{\widehat{LP}(p+1)-p, 0\}]. \tag{8.5.2}$$

To use Lemma 8.5.1 as an algorithm for computing $\mathcal{QS}^l_m(u,v)$, we have to compute the values $\widehat{LP}(i)$, $\widehat{LS}(i)$ for $i \in [1..m]$. All these values can be computed in time $O(m)$, as explained in Section 8.3.1.

We conclude that all quasi-squares of $\mathcal{QS}^l_m(u,v)$ can be computed in time $O(m + \mathrm{Card}\left(\mathcal{QS}^l_m(u,v)\right))$. Similarly, all quasi-squares of $\mathcal{QS}^r_m(u,v)$ can be computed in time $O(m + |\mathcal{QS}^r_m(u,v)|)$, and thus all quasi-squares of $\mathcal{QS}_m(u,v)$ are computed in time $O(m + \mathrm{Card}\,(\mathcal{QS}_m(u,v)))$. A straightforward divide-and-conquer algorithm gives the running time $O(n \log n + \mathrm{Card}\,(\mathcal{QS}(u,v)))$ for finding all quasi-squares in u, v.

THEOREM 8.5.2. *The set $\mathcal{QS}(u, v)$ of all quasi-squares in words u, v of length n can be found in time $O(n \log n + \mathrm{Card}\,(\mathcal{QS}(u, v)))$.*

8.6. Finding repeats with a fixed gap

In this section, we are interested in the problem of computing all factors of a given word repeated within a specified distance, rather then contiguously. In other words, we want to find all factor occurrences uvu, where the size of v, called the *gap*, is equal to a pre-specified constant δ. We will show that using the algorithm of the previous sections, all such occurrences can be found in time $O(n \log \delta + S)$, where S is the number of them. Thus, if δ is considered constant, we obtain an $O(n + S)$ time bound.

8.6.1. Algorithm for finding repeats with a fixed gap

Let $\delta > 0$ be an integer, called the *gap*. An occurrence in w of a factor $r = uvu$, where $|u| > 0$ and $|v| = \delta$, is called a *δ-gapped repeat* (for short, *δ-repeat*) in w. The left occurrence of u is called the *left copy* of r, and the right one the *right copy*. For a δ-repeat r, the length $|u|$ is called the *copy length*. The problem is to find all δ-repeats in a given word.

Let $w = a_1 \cdots a_n$ be a word of length n. Without loss of generality, we assume that a_n does not occur elsewhere in w. In this section, we use the Lempel-Ziv factorization (see Section 8.3.2). Let $w = f_1 \cdots f_k$ be the Lempel-Ziv factorization of w.

To describe the algorithm, we need some notation. Let $b_0, b_1, \ldots, b_{k-1}, b_k$ be the end positions of f_i's, that is $b_0 = 0$, and $b_i = |f_1 \cdots f_i|$ for $1 \le i \le k$. We also denote $\ell_i = |f_i|$, $i = 1, \ldots, k$. For every $i = 1, \ldots, k-1$, if $\ell_i > \delta$, then we decompose $f_i = f_i' f_i''$, where $|f_i'| = \delta$.

Let us split the set $\mathcal{GR}$ of all δ-repeats into the set $\mathcal{GR}'$ of those δ-repeats which contain a frontier between LZ-factors and the set $\mathcal{GR}''$ of the δ-repeats located properly inside LZ-factors. We now concentrate on the δ-repeats of $\mathcal{GR}'$, and further split $\mathcal{GR}'$ into (disjoint) subsets $\mathcal{GR}_i'$, $i = 1, \ldots, k-1$, where $\mathcal{GR}_i'$ consists of those δ-repeats which contain the frontier between f_i and f_{i+1} but don't contain the frontier between f_{i+1} and f_{i+2}. Furthermore, each $\mathcal{GR}_i'$ is split into the following subsets:

(a) $r \in \mathcal{GR}_i^{lrt}$ iff the left copy of r contains the frontier between f_i and f_{i+1},
(b) $r \in \mathcal{GR}_i^{rrt}$ iff the right copy of r contains the frontier between f_i and f_{i+1},
(c) $r \in \mathcal{GR}_i^{mrt}$ iff $\ell_{i+1} > \delta$ and the right copy of r contains the frontier between f_{i+1}' and f_{i+1}'' but does not contain the frontier between f_i and f_{i+1},
(d) $r \in \mathcal{GR}_i^{mid}$ iff the right copy of r contains neither the frontier between f_i and f_{i+1}, nor, if $\ell_{i+1} > \delta$, the frontier between f_{i+1}' and f_{i+1}''.

Cases (a) and (b) cover the situation when the frontier between f_i and f_{i+1} is contained respectively in the left and right copy of r. Otherwise, this frontier is contained in the gap between the copies. Cases (c) and (d) distinguish whether the right copy contains the frontier between f_{i+1}' and f_{i+1}'' or not, provided that $\ell_{i+1} > \delta$.

We now consider each of cases (a)-(d) separately and show how to find the corresponding set of δ-repeats.

(a) Finding δ-repeats of $\mathcal{GR}_i^{lrt}$. Let $r \in \mathcal{GR}_i^{lrt}$ be a δ-repeat with copy length p. Since r does not contain the frontier between f_{i+1} and f_{i+2}, then $\delta + p < \ell_{i+1}$, and therefore $p \leq \ell_{i+1} - 1 - \delta$. In particular, $\mathcal{GR}_i^{lrt}$ is empty whenever $\ell_{i+1} \leq \delta + 1$. Assume now that $\ell_{i+1} > \delta + 1$.

Let t_i be the suffix of $f_1 \cdots f_i$ of length $\ell_{i+1} - 1 - \delta$. We define the following longest extension functions on all positions $i \in [1..\ell_{i+1} - 1]$ of f_{i+1}:

$$\widehat{LP}(i) = LP_{f_{i+1}[1..\ell_{i+1}-1]}(i)$$
$$\widehat{LS}(i) = LS_{t_i|f_{i+1}[1..\ell_{i+1}-1]}(i)$$

Similarly to Theorem 8.4.1, an occurrence of a δ-repeat $r \in \mathcal{GR}_i^{lrt}$ implies

$$\widehat{LS}(\delta + p) + \widehat{LP}(\delta + p + 1) \geq p. \tag{8.6.1}$$

Conversely, if for some $p \in [1..\ell_{i+1} - 1 - \delta]$, inequation (8.6.1) holds, then there exists a δ-repeat of $\mathcal{GR}_i^{lrt}$ with copy length p. To summarize, the following lemma holds.

Lemma 8.6.1. *For $p \in [1..\ell_{i+1} - 1 - \delta]$, there exists a δ-repeat of $\mathcal{GR}_i^{lrt}$ with copy length p iff inequality (8.6.1) holds. When (8.6.1) holds, there is a family of δ-repeats of $\mathcal{GR}_i^{lrt}$ with copy length p, starting at each position of the interval*

$$[b_i + 1 - \min\{\widehat{LS}(\delta + p), p\} \;..\; b_i + 1 + \min\{\widehat{LP}(\delta + p + 1) - p, 0\}]. \tag{8.6.2}$$

Lemma 8.6.1 gives a method of computing $\mathcal{GR}_i^{lrt}$. Compute the longest extension functions $\widehat{LS}$ and $\widehat{LP}$. According to Section 8.3.1, this computation can be done in linear time in the length of involved words, that is in time $O(\ell_{i+1})$. Then, all δ-repeats of $\mathcal{GR}_i^{lrt}$ can be computed in time $O(\ell_{i+1} + \mathrm{Card}\left(\mathcal{GR}_i^{lrt}\right))$.

(b) Finding δ-repeats of $\mathcal{GR}_i^{rrt}$. Consider a δ-repeat $r \in \mathcal{GR}_i^{rrt}$ with copy length p. From the definition of Lempel-Ziv factorization, it follows that the right copy of r starts strictly after the start of f_i. On the other hand, from the definition of $\mathcal{GR}_i^{rrt}$, it ends strictly before the end of f_{i+1}. Therefore, $p \leq \ell_i + \ell_{i+1} - 2$.

We then proceed similarly to case (a). Using appropriate longest extension functions computed in time $O(\ell_i + \ell_{i+1})$, all δ-repeats of $\mathcal{GR}_i^{rrt}$ can be reported in time $O(\ell_i + \ell_{i+1} + \mathrm{Card}\left(\mathcal{GR}_i^{rrt}\right))$.

(c) Finding δ-repeats of $\mathcal{GR}_i^{mrt}$. Note that this case is defined only when $\ell_{i+1} > \delta$. Consider a δ-repeat $r \in \mathcal{GR}_i^{mrt}$ with copy length p. The right copy of r occurs inside $w[b_i + 2..b_i + \ell_{i+1} - 1]$, and therefore $p \leq \ell_{i+1} - 2$.

Again, using appropriate longest extension functions, all δ-repeats of $\mathcal{GR}_i^{mrt}$ can be reported in time $O(\ell_{i+1} + \mathrm{Card}\left(\mathcal{GR}_i^{mrt}\right))$.

(d) Finding δ-repeats of $\mathcal{GR}_i^{mid}$. Consider now a δ-repeat $r \in \mathcal{GR}_i^{mid}$ with copy length p. Denote $m_i = \min\{\delta, \ell_{i+1}\}$. The right copy of r occurs inside $f_{i+1}[2..m_i - 1]$, and therefore $p \leq m_i - 2$.

This case differs from cases (a)-(c) in that we cannot *a priori* select a position contained in the right or left copy of r. Therefore, we cannot apply directly the technique of longest extension functions. We reduce this case to the problem of finding quasi-squares, considered in Section 8.5.

Since the start position of the right copy belongs the interval $[b_i + 2..b_i + m_i - 1]$, the end position of the left copy belongs to the interval $w[b_i + 1 - \delta..b_i + m_i - 2 - \delta]$. Since $p \leq m_i - 2$, the left copy of r is contained in the word $w[b_i - \delta - m_i + 4..b_i + m_i - 2 - \delta]$.

Consider the word $w' = w[b_i - \delta - m_i + 4..b_i + m_i - 1 - \delta]$. The length of w' is $(2m_i - 4)$. Let $\#$ be another fresh letter. Denote by w'' the word $\#^{m_i-2}w[b_i + 2..b_i + m_i - 1]$.

Lemma 8.6.2. *There exists a δ-repeat $r \in \mathcal{GR}_i^{mid}$ iff there exists a quasi-square in words w', w''. Each such quasi-square corresponds to a δ-repeat $r \in \mathcal{GR}_i^{mid}$.*

Proof. Consider a δ-repeat $r \in \mathcal{GR}_i^{mid}$ with the left copy $w[\ell..\ell+p-1]$ and the right copy $w[\ell+\delta+p..\ell+\delta+2p-1]$. The right copy is a factor of $w[b_i+2..b_i+m_i-1]$, and therefore starts at position $(\ell + \delta + p - b_i + m_i - 3)$ in w''. The left copy starts at position $\ell - (b_i - \delta - m_i + 4) + 1 = \ell + \delta - b_i + m_i - 3$ in w' and therefore this forms a quasi-square of period p in w' and w''.

Inversly, assume there is a quasi-square $w'[j..j+p-1] = w''[j+p..j+2p-1]$. We must have $[j + p..j + 2p - 1] \subseteq [m_i - 1..2m_i - 4]$. This implies that $w[b_i - \delta - m_i + j + 3..b_i - \delta - m_i + j + p + 2] = w[b_i - m_i + j + 3..b_i - m_i + j + p + 2]$, and hence a δ-repeat starting at position $(b_i - \delta - m_i + 3 + j)$ in w. ∎

In view of Theorem 8.5.2, all quasi-squares in w', w'' can be found in time $O(m_i \log m_i)$. We conclude that all δ-repeats of $\mathcal{GR}_i^{mid}$ can be reported in time $O(m_i \log m_i + \mathrm{Card}\left(\mathcal{GR}_i^{mid}\right))$. Using $m_i = \min\{\delta, \ell_{i+1}\}$, rewrite this bound as $O(\ell_{i+1} \log \delta + \mathrm{Card}\left(\mathcal{GR}_i^{mid}\right))$.

Putting together cases (a)-(d), all δ-repeats of $\mathcal{GR}'_i$ can be found in time $O(\ell_i) + O(\ell_{i+1} \log \delta) + O(\mathrm{Card}\left(\mathcal{GR}'_i\right))$. Summing up over all $i = 1..k$, we obtain that all δ-repeats of $\mathcal{GR}'$ can be found in time $O(n \log \delta + \mathrm{Card}\left(\mathcal{GR}'\right))$.

Finding δ-repeats of $\mathcal{GR}''$ can be done using a technique similar to the second stage of Maximal-Repetitions from Section 8.4. The key observation here is that each δ-repeat of $\mathcal{GR}''$ occurs inside some factor f_i (i.e. does not contain positions $b_i + 1$ and b_{i+1}). By definition of the factorization, each such δ-repeat is a copy of another δ-repeat occurring to the left. When constructing the Lempel-Ziv factorization, we store, for each factor $f_i = va$, a reference to an earlier occurrence of v. δ-repeats occurring in f_i are located inside v, and are retrieved from its copy by the method used in the second stage of Maximal-Repetitions. The running time of this stage is $O(n + \mathrm{Card}\left(\mathcal{GR}''\right))$.

We conclude with the final result of this section.

THEOREM 8.6.3. *The set $\mathcal{GR}$ of all δ-repeats in a word of length n can be found in time $O(n \log \delta + \mathrm{Card}\,(\mathcal{GR}))$.*

8.6.2. Finding δ-repeats with fixed gap word

The algorithm of Section 8.6.1 can be modified in order to find all δ-repeats with a fixed word between the two copies. Assume v is a fixed word of length δ. Denote by $\mathcal{GR}_v$ the set of δ-repeats of the form uvu, where $|u| \geq 1$. We show that all those repeats can be found in time $O(n \log \delta + \mathrm{Card}\,(\mathcal{GR}_v))$. To do that, we first find, using any linear-time string matching algorithm (for example, the Knuth-Morris-Pratt algorithm) all start occurrences of v in w. For each position i of v, we compute the position $next(i)$, defined as the closest start position of v to the right of i.

From the algorithm of Section 8.6.1 for finding the set $\mathcal{GR}'$, it should be clear that all the δ-repeats of $\mathcal{GR}'$ can be represented by $O(n \log \delta)$ families each consisting of δ-repeats with a given copy length and starting at all positions from a given interval. In other words, each family can be specified by an interval $[i..j]$ and a number p, and encodes all δ-repeats with copy length p starting at positions from $[i..j]$.

From this description, using function $next(i)$, we can easily extract all δ-repeats of $\mathcal{GR}_v$ in time proportional to the number of those. For that, we first assume that each family is specified by the interval of start positions of the gap between the left and right copies (as the copy length p is known for each family, the translation can be trivially computed by just adding p to the interval of start positions). Then we process all the families and extract from each interval those positions which are start positions of an occurrence of v. Using function $next$, this can be easily done in time proportional to the number of such positions.

After processing all families, we have found all δ-repeats from the set $\mathcal{GR}'_v = \mathcal{GR}_v \cap \mathcal{GR}'$ in time $O(n \log \delta + \mathrm{Card}\left(\mathcal{GR}'_v\right))$. Then, using a procedure for finding δ-repeats from $\mathcal{GR}''$, described in Section 8.6.1, we find all δ-repeats from $\mathcal{GR}''_v = \mathcal{GR}_v \cap \mathcal{GR}''$ in time $O(n + \mathrm{Card}\left(\mathcal{GR}''_v\right))$. As $\mathcal{GR}_v = \mathcal{GR}'_v \cup \mathcal{GR}''_v$, all δ-repeats from $\mathcal{GR}_v$ are found in time $O(n \log \delta + \mathrm{Card}\,(\mathcal{GR}_v))$.

8.7. Computing local periods of a word

In this section we focuse on the important notion of *local periods*, that characterize a local periodic structure at each location of the word. The local period at a given position is the root size of the smallest square centered at this position. An importance of local periods is evidenced by the fundamental Critical Factorization Theorem that asserts that there exists a position in the word (and a corresponding factorization), for which the local period is equal to the global period of the word.

Consider a word $w = a_1 \cdots a_n$ over a finite alphabet. Let $w = uv$ be a factorization of w such that $|u| = i$. We say that a non-empty square xx *is* *centered at position* i *of* w (or *matches* w *at central position* i) iff the following conditions hold:

(i) x is a suffix of u, or u is a suffix of x,
(ii) x is a prefix of v, or v is a prefix of x.

In the case when x is a suffix of u and x is a prefix of v, we have a square occurring inside w. We call it an *internal square*. If v is a proper prefix of x (respectively, u is a proper suffix of x), the square is called *right-external* (respectively, *left-external*).

The smallest square centered at a position i of w is called the *minimal local* square (hereafter simply *minimal*, for shortness). The *local period* at position i of w, denoted $MLP_w(i)$, is the period of the minimal square centered at this position[1].

Note that for each position i of w, $MLP_w(i)$ is well-defined, and $1 \leq MLP_w(i) \leq |w|$. The relation between local periods and the (global) period of the word is established by the fundamental Critical Factorization Theorem.

THEOREM 8.7.1 (Critical Factorization Theorem). *For each word w, there exists a position i (and the corresponding factorization $w = uv$, $|u| = i$) such that $MLP_w(i) = p(w)$. Moreover, such a position exists among any $p(w)$ consecutive positions of w.*

In this section, we show how the techniques of the previous sections can be used to compute *all* local periods in a word in time $O(n)$, assuming a constant-size alphabet. The method consists of two parts. We first show, in Section 8.7.1, how to compute all *internal* minimal squares. Then, in Section 8.7.2 we show how to compute left- and right-external minimal squares, in particular for those positions for which no internal square has been found. Both computations will be shown to be linear-time, and therefore computing all local periods can be done within linear time too.

8.7.1. Computing internal minimal squares

Finding internal minimal squares amounts to compute, for each position of the word, the smallest square that is centered at this position and occurs entirely inside the word, provided that such a square exists. Thus, throughout this section we will be considering only squares occurring inside the word and, for the sake of brevity, omit the adjective "internal".

The general approach is to use the algorithm for computing maximal repetitions from Section 8.4 in order to retrieve squares which are minimal for some position. One modification of MAXIMAL-REPETITIONS we make here is that we use the s-factorization *with non-overlapping copies* (see Section 8.3.2) instead of the regular s-factorization, used in Section 8.4. This modification, however, does not affect any properties of MAXIMAL-REPETITIONS, including its linear time bound.

[1] Note that the period of a square xx is $|x|$ and not the minimal period of word xx which can be smaller.

Let us now focus on the first stage of MAXIMAL-REPETITIONS. At this step, we find, for each s-factor $f_j = w[b_{j-1}+1..b_j]$, all maximal repetitions that start before b_{j-1} and end inside f_j. According to Corollary 8.1.4, for each possible period p, there can be at most two such repetitions. Each maximal repetition is a run of squares occurring at successive positions in the word. For our purpose here, it will be convenient for us to think of a repetition as an interval of center positions of squares it contains.

In this section, we will need to compute, for a maximal repetition, a *subrun* of squares it contains, that is a subinterval of the corresponding interval of center positions. To illustrate this, consider Theorem 8.4.1. The maximal repetition found according to the theorem corresponds to squares centered at positions $[m + p - LS_{x|y}(p) \,..\, m + LP_y(p+1)]$. If we want to compute only squares centered at positions greater than or equal to m and starting at positions less than or equal to m (as it will be the case below), the interval of centers should be restricted to $[m + \max\{p - LS_{x|y}(p), 0\} \,..\, m + \min\{LP_y(p+1), p\}]$. A similar interval restriction has been done in the previous section for computing δ-repeats.

We present now a linear-time algorithm for computing all internal minimal squares in a given word w. The general description of the algorithm is as follows. First, we compute, in linear time, the s-factorization of w with non-overlapping copies and keep, for each factor f_j, a reference to its non-overlapping left copy. Then we process all factors from left to right and compute, for each factor f_j, all minimal squares ending in this factor. For each computed minimal square, centered at position i, the corresponding value $MLP_w(i)$ is set. After the whole word has been processed, positions i for which values $MLP_w(i)$ have not been assigned are those for which no internal square centered at i exists. For those positions, minimal squares are external, and they will be computed at the second stage, presented in Section 8.7.2.

Let $f_j = w[b_{j-1}+1..b_j]$ be the current factor, $\ell_j = b_j - b_{j-1}$, and let $w[b_{j-1}+1-\Delta_j \,..\, b_j - \Delta_j]$ be its non-overlapping left copy (i.e. $\Delta_j \geq \ell_j$). If for some position $b_{j-1}+i$, $1 \leq i < \ell_j$, the minimal square centered at $b_{j-1}+i$ occurs entirely inside f_j, that is $MLP_w(b_{j-1}+i) \leq \min\{i, \ell_j - i\}$, then $MLP_w(b_{j-1}+i) = MLP_w(b_{j-1}+i-\Delta_j)$. Note that $MLP_w(b_{j-1}+i-\Delta_j)$ has been computed before, as the minimal square centered at $b_{j-1}+i$ ends before the beginning of f_j. Based on this observation, we retrieve, in time $O(|f_j|)$, all values $MLP_w(b_{j-1}+i)$ which correspond to squares occurring entirely inside f_j. Therefore, it remains to find those values $MLP_w(b_{j-1}+i)$ which correspond to minimal squares that end in f_j and extend to the left beyond the frontier between f_j and f_{j-1}.

To do this, we use the technique of computing runs of squares from Section 8.4. The idea is to compute all candidate squares and test which of them are minimal. However, this should be done carefully as this can break down the linear time bound, due to a possible super-linear number of all squares. The main trick is to keep squares in runs and to show that there is only a linear number of individual squares which need to be tested for minimality.

We are interested in squares starting at positions less than or equal to b_{j-1}

and ending inside f_j. All these squares are divided into those which are centered inside f_j and those centered to the left of f_j. Two cases are symmetrical and therefore we concentrate on squares centered at positions $[b_{j-1}..b_{j-1}+\ell_j-1]$. We compute all such squares *in the increasing order of periods*. For each $p \in [1..\ell_j]$ we compute the run of all squares of period p centered at positions belonging to the interval $[b_{j-1}..b_{j-1}+\ell_j-1]$, starting at a position less than or equal to b_{j-1}, and ending inside f_j, as explained above. Assume we have computed a run of such squares of period p, and assume that $q < p$ is the maximal period value for which squares have been previously found. If $p \geq 2q$, then we check each square of the run whether it is minimal or not by checking the value $MLP_w(b_{j-1}+i)$. If this square is not minimal, then $MLP_w(b_{j-1}+i)$ has been already assigned a positive value before. Indeed, if a smaller square centered at $b_{j-1}+i$ exists, it has necessarily been already computed by the algorithm (recall that squares are computed in the increasing order of periods). If no positive value $MLP_w(b_{j-1}+i)$ has yet been set, then we have found the minimal square centered at $b_{j-1}+i$. Since there is at most p considered squares of period p (their centers belong to the interval $[b_{j-1}..b_{j-1}+p-1]$), checking all of them takes at most $2(p-q)$ individual checks (as $q \leq p/2$ and $p-q \geq p/2$).

Now assume $p < 2q$. Consider a square $s_q = w[c_q-q+1..c_q+q]$ of period q and center c_q, which has been previously found by the algorithm (square of period q in Figure 8.4). We now prove that we need to check for minimality only those squares s_p of period p which have their center c_p verifying one of the following inequalities :

$$|c_p - c_q| \leq p - q, \text{ or} \tag{8.7.1}$$

$$c_p \geq c_q + q \tag{8.7.2}$$

In words, c_p is located either within distance $p-q$ from c_q, or beyond the end of square s_q.

LEMMA 8.7.2. *Let $s_p = w[c_p-p+1..c_p+p]$ be the minimal square centered at some position c_p. Let $s_q = w[c_q-q+1..c_q+q]$ be another square with $q < p$. Then one of inequations (8.7.1),(8.7.2) holds.*

Proof. By contradiction, assume that neither of them holds. Consider the case $c_p > c_q$, case $c_p < c_q$ is symmetric. The situation with $c_p > c_q$ is shown in Figure 8.4. Now observe that word $w[c_q+1..c_p]$ has a copy $w[c_q-q+1..c_p-q]$ (shown with empty strip in Figure 8.4) and that its length is (c_p-c_q). Furthermore, since $c_p-c_q > p-q$ (as inequation (8.7.1) does not hold), this copy overlaps by $p-q$ letters with the left root of s_p. Consider this overlap $w[c_p-p+1..c_p-q]$ (shadowed strip in Figure 8.4). It has a copy $w[c_p+1..c_p+(p-q)]$ and another copy $w[c_p-(p-q)+1..c_p]$ (see Figure 8.4). We thus have a smaller square centered at c_p, which proves that square s_p is not minimal. ■

By Lemma 8.7.2, we need to check for minimality only those squares s_p which verify, with respect to s_q, one of inequations (8.7.1),(8.7.2). Note that there are at most $2(p-q)$ squares s_p verifying (8.7.1), and at most $p-q$ squares

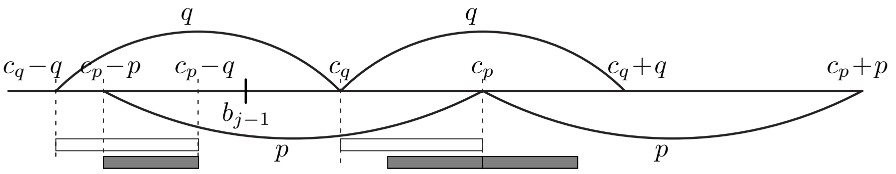


Figure 8.4. Case where neither of inequations (8.7.1),(8.7.2) holds (subcase $c_p > c_q$)

s_p verifying (8.7.2), the latter because s_p must start before the current factor, i.e. $c_p \le b_{j-1} + p$. We conclude that there are at most $3(p-q)$ squares of period p to check for minimality, among all squares found for period p. Summing up the number of all individual checks results in a telescoping sum, and we obtain that processing all squares centered in the current factor can be done in time $O(|f_j|)$.

The Right-Local-Squares algorithm for computing minimal squares centered inside f_j is given below. It is based on algorithms Right-Repetitions and Maximal-Repetitions from Section 8.4. In particular, t_i is defined as in the Maximal-Repetitions algorithm.

A similar algorithm applies to the squares centered on the left of f_j. Note that after processing f_j, all minimal squares ending in f_j have been computed.

To summarize, we need to check for minimality only $O(|f_{j-1}|+|f_j|)$ squares, among those containing the frontier between f_j and f_{j-1}, each check taking a constant time. We also need $O(|f_j|)$ time to compute minimal squares occurring inside f_j. Processing f_j takes then time $O(|f_{j-1}| + |f_j|)$ overall, and processing the whole word takes time $O(n)$.

Theorem 8.7.3. *All internal minimal squares in a word of length n can be computed in time $O(n)$.*

```
Right-Local-Squares(f_i)
▷ computing minimal squares centered inside f_i
LS_{t_i|f_i} ← Longest-Suffix-Extension(t_i, f_i)
LP_{f_i} ← Longest-Prefix-Extension(f_i)
q ← 0
for p ← 1 to ℓ_i do
    t_i ← suffix of f_1 ··· f_{i-1} of length (2|f_{i-1}| + max{2|f_{i-2}|, |f_i|})
    \widehat{LS}(p) ← min{LS_{t_i|f_i}(p), p}
    \widehat{LP}(p+1) ← min{LP_{f_i}(p+1), p}
    if \widehat{LS}(p) + \widehat{LP}(p+1) ≥ p then
        I ← [p − \widehat{LS}(p)..\widehat{LP}(p+1)]        ▷ interval of square centers
        if p < 2q then
            c'_q ← c_q − b_{j-1}
            I ← I ∩ ([c'_q − (p − q)..c'_q + (p − q)] ∪ [c'_q + q..ℓ_j − 1])
        for each i ∈ I do
            if MLP_w(b_{j-1} + i) is undefined then
                MLP_w(b_{j-1} + i) ← p
        q ← p
        c_q ← b_{j-1} + p − \widehat{LS}(p)
```

8.7.2. Computing external minimal squares

The algorithm of the previous section allows to compute all *internal* minimal squares of a word. Here we show how to compute *external* minimal squares for those positions which don't have internal squares centered at them.

Consider a word w of length n. We first consider squares which are right-external but not left-external. Those squares are centered at positions in the right half of the word. The case of squares which are left-external but not right-external is symmetrical.

For each position i in the right half of w, we compute a value $RS(i)$ equal to the period of the smallest right-external and not left-external square centered at i, provided such a square exists. We show that all values $RS(i)$ can be computed in linear time using longest extension functions (Section 8.3.1).

Consider a right-external square of period p centered at some position $i \in [\lceil n/2 \rceil..n-1]$, where $n-i < p \leq i$. Observe that $w[i-p+1..n-p] = w[i+1..n]$. This implies that $LS_w(n-p) \geq n-i$. Conversly, if for some $p \in [1..n-1]$, $LS_w(n-p) > 0$, then there exists a family of squares of period p centered at positions $i \in [n - LS_w(n-p)..n-1]$. For $i > n-p$, the square is right-external, otherwise it is internal.

This implies the following algorithm for computing minimal right-external squares. Compute LS_w for all positions of w. For each $j \in [1..n]$, set $LNS_w(j) = LS_w(j)$ if $LS_w(j) < n-j$, and $LNS_w(j) = n-j-1$ otherwise. For each center position $i \in [\lceil n/2 \rceil..n-1]$, we need to compute the minimal p such that $LNS_w(n-p) \geq n-i$.

Consider all pairs $(j, LNS_w(j))$ for $j \in [1..n]$. If for some pair $(j, LNS_w(j))$, there exists a pair $(j', LNS_w(j'))$ such that $LNS_w(j') \geq LNS_w(j)$ and $j' > j$, then $(j, LNS_w(j))$ carries no useful information for computing minimal right-external squares. We then delete all such pairs $(j, LNS_w(j))$ from consideration by looping through all j from n to 1 and deleting those for which the value $LNS_w(j)$ is smaller than or equal to $\max_{j'>j}\{LNS_w(j')\}$. We then sort the remaining pairs $(j, LNS_w(j))$ in the decreasing order of $LNS_w(j)$. Using bucket sort, this can be done in $O(n)$ time and space.

We now set the values $RS(i)$ as follows. For the first element $(j_0, LNS_w(j_0))$ of the list, we set $RS(i) = 0$ for all $i \in [\lceil n/2 \rceil..n - LNS_w(j_0) - 1]$. We then scan through the ordered list of pairs and for each element $(j, LNS_w(j))$, look at the next element $(j', LNS_w(j'))$, $LNS_w(j) > LNS_w(j')$. For all $i \in [n - LNS_w(j)..n - LNS_w(j') - 1]$, set $RS(i) = n - j$. We then have the following

Lemma 8.7.4. *For each $i \in [\lceil n/2 \rceil..n-1]$, $RS(i)$ is the smallest period of a right-external square centered at i if such a square exists, and $RS(i) = 0$ otherwise.*

We now turn to squares that are both right-external and left-external. Consider such a square of period p, centered at some position i. Observe that $w[1..n-p] = w[p+1..n]$. Therefore, there exists a *border* of w of size $n-p < n/2$. The largest border corresponds to smallest square. On the other hand, the period of this square is equal to the minimal period $p(w)$ of w. Note that, in general, each local period cannot be greater than $p(w)$, and all minimal squares which are both right-external and left-external have the period equal to $p(w)$.

It is well known that $p(w)$ can be easily computed in linear time (see Chapter 1). We then obtain an $O(n)$ algorithm for computing all minimal squares: first, using Theorem 8.7.3 we compute all minimal internal squares; then, using Lemma 8.7.4 and the above remark, we compute the minimal external squares for those positions for which no internal square has been found at the first stage. This proves the main result.

Theorem 8.7.5. *For a word of length n, all local periods $MLP_w(i)$ can be computed in time $O(n)$.*

8.8. Finding approximate repetitions

In many practical applications, such as DNA sequence analysis, considered repetitions admit a certain variation between copies of the repeated pattern. In other words, repetitions under interest are *approximate repetitions* and not necessarily exact repetitions only. Computing approximate repetitions is the subject of this section.

The simplest notion of approximate repetition is an *approximate square*. An approximate square in a word is a factor uv, where u and v are within a given distance k and the distance measure could be one of those usually used in practical applications, such as Hamming distance or Levenshtein (or edit) distance. Here we focus on the Hamming distance, when the variation between repeated copies can be only letter replacements. An important motivation here is to define structures encoding families of approximate squares, analogous to maximal repetitions in the exact case. In Section 8.8.1, we define two basic structures that we call *K-repetitions* and *K-runs*, where K the number of allowed errors. In Section 8.8.2, we show that all K-repetitions can be found in time $O(nK \log K + S)$, where S is their number. In Section 8.8.3 we show that the same bound holds for K-runs: all of them can be found in time $O(nK \log K + R)$, where R is their number. The latter result implies, in particular, that all approximate squares can be found in time $O(nK \log K + T)$ (T their number). All those algorithms require only $O(n)$ of working space.

8.8.1. K-repetitions and K-runs

Let $h(\cdot, \cdot)$ be the Hamming distance between two words of equal length, that is $h(u, v)$ is the number of mismatches (letter differences at corresponding positions) between u and v. For example, $h(baaacb, bcabcb) = 2$.

A word $s = uv$, such that $|u| = |v|$, is called a *K-square* iff $h(u, v) \leq K$. Reusing the terminology of the exact case, we call $p = |u| = |v|$ the *period* of s, and words u, v the *left* and *right root* of s respectively.

We now want to define a more global structure which would be able to capture "long approximate repetitions", generalizing repetitions of arbitrary exponent in the exact case. As opposed to the exact case, Conditions (i)-(ii) of Proposition 8.1.1 generalize to different notions of approximate repetition. Condition (i) gives rise to the strongest of them: A word $r[1..n]$ is called a *K-repetition* of period p, $p \leq n/2$, iff $h(r[1..n-p], r[p+1..n]) \leq K$.

Equivalently, a word $r[1..n]$ is a K-repetition of period p, if the number of i such that $r[i] \neq r[i+p]$ is at most K. For example, *abaa abba cbba cb* is a 2-repetition of period 4. *abc abc abc abd abd abd abd abd* is a 1-repetition of period 3 but *abc abc abc abb abc abc abc abb* is not.

Another point of view, expressed by Condition (ii) of Proposition 8.1.1, considers a repetition as an encoding of squares it contains. Projecting this to the approximate case, we come up with the notion of run of approximate squares: A word $r[1..n]$ is called a *run of K-squares*, or a *K-run*, of period p, $p \leq n/2$, iff for every $i \in [1..n-2p+1]$, the factor $s = r[i..i+2p-1]$ is a K-square of period p.

Similarly to the exact case, when we are looking for approximate repetitions occurring in a word, it is natural to consider *maximal* approximate repetitions. Those are repetitions extended to the right and left as much as possible provided that the corresponding definition is still verified. Note that the notion of maximality applies both to K-repetitions and to K-runs: in both cases we can extend each repetition to the right and left as long as it verifies the corresponding definition. We will always be interested in maximal K-repetitions and K-runs, without mentioning it explicitly. Note that for both definitions, the maximality requirement implies that if $r = w[i..j]$ is an approximate repetition of period p in $w[1..n]$, then $w[j+1] \neq w[j+1-p]$ (provided $j < n$) and $w[i-1] \neq w[i-1+p]$ (provided $i > 1$). Furthermore, if $w[i..j]$ is a maximal K-repetition, it contains

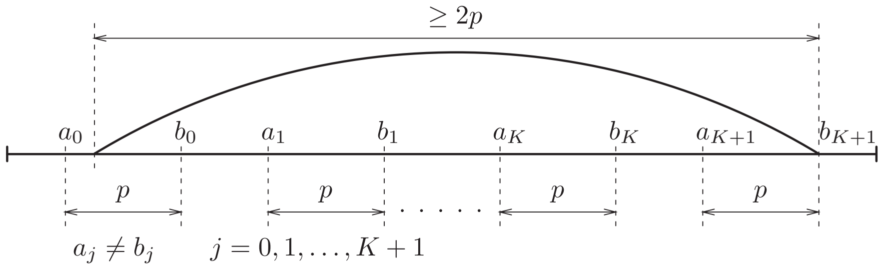


Figure 8.5. Maximal K-repetition

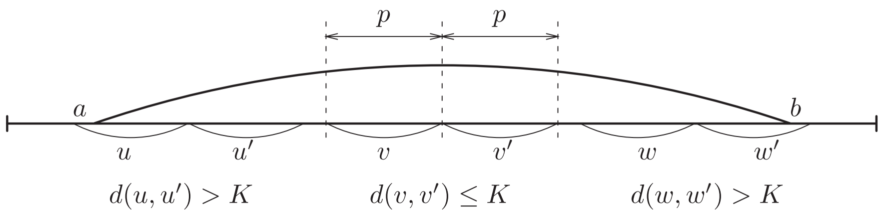


Figure 8.6. Maximal K-run

exactly K mismatches $w[\ell] \neq w[\ell + p]$, $i \leq \ell, \ell + p \leq j$, unless the whole word w contains less than K mismatches (to simplify the presentation, we exclude this latter case from consideration).

Figure 8.5 illustrates the definition of (maximal) K-repetitions and Figure 8.6 that of (maximal) K-run.

Example 8.8.1. The following Fibonacci word contains three 3-runs of of period 6. They are shown in regular font, in positions aligned with their occurrences. Two of them are identical, and contain each four 3-repetitions, shown in italic for the first run only. The third run is a 3-repetition in itself.

010010 100100 101001 010010 010100 1001

10010 100100 101001
10010 100100 10
0010 100100 101
10 100100 10100
0 100100 101001
1001 010010 010100 1
10 010100 1001

In general, each K-repetition is a factor of a K-run of the same period. On the other hand, a K-run in a word is the union of all K-repetitions it contains. Observe that a K-run can contain as many as a linear number of K-repetitions with the same period. For example, the word $(000\,100)^n$ of length $6n$ is a 1-run of period 3, which contains $(2n-1)$ 1-repetitions.

In general, the following lemma holds.

LEMMA 8.8.2. *Let $w[1..n]$ be a K-run of period p and let s be the number of mismatches $w[i] \neq w[i+p]$, $1 \leq i, i+p \leq n$ (equivalently, $s = h(w[1..n-p], w[p+1..n])$). Then w contains $(s-K+1)$ K-repetitions of period p.*

K-runs and K-repetitions provide respectively the weekest and strongest notions of repetitions with K mismatches, and therefore “embrace” all practically relevant repetitions.

8.8.2. Finding K-repetitions

In this subsection we describe how to find, in a given word w, all maximal K-repetitions occurring in w (K is a given constant).

We assume we fixed a minimal bound p_0 for the period of repetitions we are looking for. For example, p_0 can be taken to be $K+1$ having in mind that if a period $p \leq K$ is allowed, then any factor of length $2p$ would be a K-square. This assumption, however, is pragmatic and does not affect the method nor the complexity bounds.

From a general point of view, we are going to apply the same approach as the one used in Sections 8.4-8.7. However, the case of approximate repetitions is more complex and requires a number of modifications.

We start with describing the modification of the basic problem of finding repetitions containing a given position of the word (Theorem 8.4.1 of Section 8.4). Recall that a factor $w[i..j]$ of w is said to *contain* a position ℓ of w iff $i \leq \ell \leq j$. A factor $w[i..j]$ is said to *touch* a position ℓ of w iff $i-1 \leq \ell \leq j+1$. Here, it will be convenient for us to specify the problem as follows: Given a word $w[1..n]$ and a distinguished position $\ell \in [2..n-1]$, find all K-repetitions in w that touch ℓ. Similar to Section 8.4, we distinguish two (non-disjoint) classes of K-repetitions according to whether they have a root on the right or on the left of position ℓ. We focus on K-repetitions of the first class, those of the second class are found similarly.

To apply a method similar to the one of Theorem 8.4.1, we need a generalisation of longest extension functions (Section 8.3.1) that compares factors up to a given Hamming distance. Formally, given a word w and a position ℓ, for every $k = 0, \ldots, K$, we compute the following functions on $p \in [p_0..n-\ell]$:

$$LP^{(k)}_{w,\ell}(p) = \max\{j \mid h(w[\ell+p..\ell+p+j-1], w[\ell+1..\ell+j]) \leq k\}, \quad (8.8.1)$$

$$LS^{(k)}_{w,\ell}(p) = \max\{j \mid h(w[\ell+p-j+1..\ell+p], w[\ell-j+1..\ell]) \leq k\}. \quad (8.8.2)$$

$LP^{(k)}_{w,\ell}(p)$ is the length of the longest factor of w starting at position $\ell+p$ and equal, within k mismatches, to the factor of the same length starting at $\ell+1$.

$LS_{w,\ell}^{(k)}(p)$ is the length of the longest factor ending at position $\ell + p$ and equal, within k mismatches, to the factor ending at position ℓ. These functions are generalizations of longest extension functions considered in Section 8.3.1 and can be computed in time $O(nK)$ using suffix trees combined with the lowest common ancestor computation in a tree (see Gusfield 1997).

Consider now a K-repetition r of period p that has a root on the right of a position ℓ. Note that position $\ell+p$ of w is contained in r, and that r is uniquely defined by the number of mismatches $w[i+p] \neq w[i]$, $i \geq \ell + 1$, occurring in r. Let k be the number of those mismatches. The following theorem is a generalization of Theorem 8.4.1.

THEOREM 8.8.3. *Let w be a word of length n and let ℓ, $1 < \ell < n$, be a distinguished position of w. There exists a K-repetition of period p which touches position ℓ, and has a root on the right of ℓ, iff for some $k \in [0..K]$,*

$$LS_{w,\ell}^{(K-k)}(p) + LP_{w,\ell}^{(k)}(p+1) \geq p. \tag{8.8.3}$$

When (8.8.3) holds, this repetition starts at position $\left(\ell - LS_{w,\ell}^{(K-k)}(p) + 1\right)$ and ends at position $\left(\ell + p + LP_{w,\ell}^{(k)}(p)\right)$.

Theorem 8.8.3 provides an $O(nK)$ algorithm for finding all considered K-repetitions: compute longest extension functions (8.8.1), (8.8.2) (this takes time $O(nK)$) and then check inequation (8.8.3), for each $k = 0, \ldots, K$ and all $p \in [p_0..\ell-1]$ (this takes time $O(nK)$ too). Every time the inequation is verified, a K-repetition is identified. The computation is summarized in the following algorithm:

MISMATCH-RIGHT-REPETITIONS(w, ℓ)

$\triangleright$ Find K-repetitions of w which have a root on the right of position ℓ
for all $k = 0, \ldots, K$ **do**
    $\triangleright$ compute longest extension functions (8.8.1), (8.8.2)
    $LP_{w,\ell}^{(k)} \leftarrow$ MISMATCH-PREFIX-EXTENSION(w, ℓ, k)
    $LP_{w,\ell}^{(k)} \leftarrow$ MISMATCH-SUFFIX-EXTENSION(w, ℓ, k)
$\mathcal{R} \leftarrow \emptyset$
for $p \leftarrow p_0$ **to** $\min\{n - \ell + 1, n/2\}$ **do**
    for $k \leftarrow 0$ **to** K **do**
        if $LP_{w,\ell}^{(k)}(p+1) + LS_{w,\ell}^{(k)}(p) \geq p$ **then**
            $r \leftarrow (\ell - LS_{w,\ell}^{(K-k)}(p) + 1, \ell + p + LP_{w,\ell}^{(k)}(p))$
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
return $\mathcal{R}$

Finding repetitions having a root on the left of position ℓ is a symmetric problem that can be solved within the same time bound (hereafter referred to as algorithm MISMATCH-LEFT-REPETITIONS).

We are now ready to describe an extension of the algorithm Maximal-Repetitions from Section 8.4 to compute all K-repetitions in a word w. Consider the Lempel-Ziv factorization $w = f_1 f_2 \cdots f_m$ (Section 8.3.2). The last position of an LZ-factor f_i will be called the *head* of f_i. The algorithm consists of three stages. The **first stage** is based on the following two lemmas.

Lemma 8.8.4. *The suffix of length p of a K-repetition of period p cannot contain $K + 1$ consecutive LZ-factors.*

Proof. Each LZ-factor contained in the suffix of length p of a K-repetition must contain at least one mismatch with the letter located p positions to the left. Indeed, if it does not contain a mismatch, it has an exact copy occurring earlier, which contradicts the definition of the Lempel-Ziv factorization. All those mismatches belong to the repetition and there are at most K of them. Therefore, the suffix of length p contains at most K LZ-factors. ∎

Divide w into consecutive *blocks* of $(K + 2)$ LZ-factors. Let $w = B_1 \cdots B_{m'}$ be the partition of w into such blocks. The last letter of block B_i will be called the *head* of this block. At the first stage, we find, for each block B_i, those K-repetitions which touch the head of B_{i-1} but do not touch that of B_i. The following lemma is analogous to Theorem 8.4.2.

Lemma 8.8.5. *Assume that a K-repetition r touches the head of B_{i-1} but not that of B_i. Then the length of the prefix of r which is a suffix of $B_1 \cdots B_{i-1}$ is bounded by $|B_i| + 2|B_{i-1}|$.*

Proof. Lemma 8.8.4 implies that the suffix of r of period length cannot start before the first letter of B_{i-1}. Therefore, the period of r is bounded by $|B_{i-1}B_i|$. On the other hand, by an argument similar to Lemma 8.8.4, r cannot extend by more than one period to the left of B_{i-1}. This is because otherwise each of the LZ-factors of B_{i-1}, except possibly the last one, would correspond to a mismatch in r, and thus r would contain at least $(K + 1)$ mismatches which is a contradiction. Therefore, the length of the prefix of r which is a suffix of $B_1 \cdots B_{i-2}$ is at most $|B_{i-1}| + |B_i|$. ∎

Based on Lemma 8.8.5, we apply Mismatch-Right-Repetitions and Mismatch-Left-Repetitions algorithms: Consider the word $w_i = v_i B_i$, where v_i is the suffix of $B_1 \cdots B_{i-1}$ of length $(2|B_{i-1}| + |B_i|)$. Then find, by Mismatch-Right-Repetitions and Mismatch-Left-Repetitions, all K-repetitions in w_i touching the head of B_{i-1} and discard those which touch the head of B_i. The resulting complexity is $O(K(|B_{i-1}| + |B_i|))$.

After processing all blocks, we find all repetitions touching block heads. Observe that repetitions resulting from processing different blocks are distinct. Summing up over all blocks, the resulting complexity of the first stage is $O(nK)$. The repetitions which remain to be found are those which lie entirely within a block – this is done at the next two stages.

At the **second stage** we find all K-repetitions inside each block B_i which touch factor heads other than the block head (i.e. the head of the last LZ-factor of the block). For each B_i, we proceed by the following divide-and-conquer procedure:

Process-Block(B)
$\triangleright$ Compute K-repetitions which touch factor heads inside a block B
$\mathcal{R} \leftarrow \emptyset$
divide $B = f_i f_{i+1} \cdots f_{i+s}$ into two sub-blocks
  $B' = f_i \cdots f_{\lfloor s/2 \rfloor}$ and $B'' = f_{\lfloor s/2 \rfloor+1} \cdots f_{i+s}$
$\triangleright$ compute K-repetitions which touch the head of B'
$h \leftarrow$ position of the head of B'
$\mathcal{R}_1 \leftarrow$ Mismatch-Right-Repetitions(B, h)
$\mathcal{R}_2 \leftarrow$ Mismatch-Left-Repetitions(B, h)
$\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1 \cup \mathcal{R}_2$
$\triangleright$ process recursively B' and B''
$\mathcal{R}' \leftarrow$ Process-Block(B')
$\mathcal{R}'' \leftarrow$ Process-Block(B'')
$\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}' \cup \mathcal{R}''$
return $\mathcal{R}$

The algorithm Process-Block has $\lceil \log_2 K \rceil$ levels of recursion, and since at each step the word is split into disjoint sub-blocks, the whole complexity of the second stage is $O(nK \log K)$.

Finally, at the **third stage**, it remains to compute the K-repetitions which occur entirely inside each Lempel-Ziv factor, i.e. don't contain its first position and don't touch its head. By definition of Lempel-Ziv factorization, each LZ-factor without its head has a (possibly overlapping) copy on the left. Therefore, each of these K-repetitions has another occurrence in that copy. Using this observation, these K-repetitions can be found using the same technique as at the second stage of Maximal-Repetitions: during the construction of the Lempel-Ziv factorization we keep, for each LZ-factor wa, a pointer to a copy of w on the left. Then process all LZ-factors from left to right and recover repetitions occurring inside each LZ-factor from its left copy in the same way that it was done at the second stage of Maximal-Repetitions. The complexity of this stage is $O(n + S)$, where S is the number of repetitions found.

The following theorem summarizes this section.

Theorem 8.8.6. *All K-repetitions in a word of length n can be found in time $O(nK \log K + S)$ where S is the number of K-repetitions found.*

The algorithm K-Repetitions given below summarizes the three stages of the computation of K-repetitions.

K-Repetitions(w)
$(f_1, \cdots, f_m) \leftarrow$ Lempel-Ziv factorization of w
partition $f_1, \cdots, f_m$ into blocks $B_1 \cdots B_{m'}$ of $K+2$ consecutive factors
$\triangleright$ **first stage**
$\mathcal{R} \leftarrow \emptyset$
for $i \leftarrow 2$ **to** m' **do**
    $v_i \leftarrow$ suffix of $B_1 \cdots B_{i-1}$ of length $(2|B_{i-1}| + |B_i|)$
    $\ell \leftarrow |v_i|$
    $\mathcal{R}'_i \leftarrow$ Mismatch-Right-Repetitions($v_i B_i, \ell$)
    $\mathcal{R}''_i \leftarrow$ Mismatch-Left-Repetitions($v_i B_i, \ell$)
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}'_i \cup \mathcal{R}''_i$
$\triangleright$ **second stage**
for $i \leftarrow 1$ **to** m' **do**
    $\mathcal{R}_i \leftarrow$ Process-Block(B_i)
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_i$
$\triangleright$ **third stage**
for each factor f_j **do**
    retrieve all K-repetitions which occur entirely inside f_j using a procedure similar to the **second stage** of Maximal-Repetitions

8.8.3. Finding K-runs

We now describe an algorithm for finding all K-runs in a word. The general structure of this algorithm is the same as algorithm K-Repetitions – it has three stages playing similar roles. However, the case of K-runs will require a considerable modification and additional algorithmic techniques, especially at the third stage.

At the first and second stages, the key difference is the type of objects we are looking for: instead of computing K-repetitions we now compute *subruns* of K-squares. Formally, a K-subrun is a family of K-squares occurring at successive positions. In other words, a K-subrun is a K-run which is not necessarily maximal. In this section, we *identify a subrun with the interval of end positions of the squares it contains* (note that a different convention has been adopted in Section 8.7.1 where we identified a subrun with the interval of central positions of its squares).

At each point of the first and the second stages when we search for repetitions touching some head position ℓ, we now compute subruns containing those K-squares which touch ℓ, i.e. K-subruns belonging to the interval $[\ell - 1..\ell + 2p]$. As in the case of K-repetitions, we split those squares into those having a root on the left of ℓ (belonging to the interval $[\ell - 1..\ell + p - 1]$) and those having a root on the right of ℓ (belonging to the interval $[\ell + p..\ell + 2p]$). Note that here, however, these two cases are disjoint. The modification of the Mismatch-Right-Repetitions algorithm is the algorithm Mismatch-Right-Subruns below. It computes the subruns of K-squares touching $w[\ell]$ and having a root on the right of it.

MISMATCH-RIGHT-SUBRUNS(w, ℓ)

1 $\triangleright$ Find subruns of K-squares touching position ℓ in w
$\triangleright$ and having the right root on the right of ℓ
2 **for** all $k = 0, \ldots, K$ **do**
3 $\triangleright$ compute longest extension functions defined by (8.8.1), (8.8.2)
4 $LP^{(k)}_{w,\ell} \leftarrow$ MISMATCH-PREFIX-EXTENSION(w, ℓ, k)
5 $LP^{(k)}_{w,\ell} \leftarrow$ MISMATCH-SUFFIX-EXTENSION(w, ℓ, k)
6 $\mathcal{L} \leftarrow$ empty list
7 **for** $p \leftarrow p_0$ **to** $\min\{n - \ell + 1, n/2\}$ **do**
8 **for** $k \leftarrow 0$ **to** K **do**
9 **if** $LP^{(k)}_{w,\ell}(p+1) + LS^{(k)}_{w,\ell}(p) \geq p$ **then**
10 $lbound(p,k) \leftarrow \max\{\ell + 2p - LS^{(k)}_{w,\ell}(p), \ell + p\}$
11 $rbound(p,k) \leftarrow \min\{\ell + p + LP^{(k)}_{w,\ell}(p), \ell + 2p\}$
12 $r \leftarrow$ subrun $(lbound(p,k), rbound(p,k))$ of period p
13 **if** $rbound(p, k-1)$ is defined **and**
$lbound(p,k) \leq rbound(p,k-1) + 1$ **then**
14 merge r with the subrun computed for $k-1$
15 **else** add r to $\mathcal{L}$
16 **else** $rbound(p,k) \leftarrow$ undefined
17 **return** $\mathcal{L}$

A major additional difficulty in computing K-runs is that we have to *assemble* them from subruns. To perform the assembling, we need to store subruns in an additional data structure that allows to maintain links between subruns and to merge adjacent subruns into bigger runs. Finally, we have to ensure that the number of subruns we come up with and the work spent on processing them do not increase the resulting complexity bound.

The assembling occurs already in the function MISMATCH-RIGHT-SUBRUNS, as intervals (subruns) found for different values of k (**for**-loop at line 8) may overlap or immediately follow each other, in which case we join them into a bigger subrun (lines 13-14). Moreover, those intervals can be disjoint, in which case we organize them in a linked list. A more formal description of the data structure will be given later.

The list of subruns of K-squares that touch $w[l]$ and have a root on the left of it is computed similarly. Once computed, it has to be concatenated with the list computed by MISMATCH-RIGHT-SUBRUNS and the rightmost subrun of left-rooted squares has to be merged with the leftmost subrun of right-rooted squares, if those subruns are adjacent.

We now describe the three stages of the algorithm in more details. Given an input word w, we compute the Lempel-Ziv factorization $w = f_1 \cdots f_k$. Unlike the case of K-repetitions, we use here the Lempel-Ziv factorization *with non-overlapping copies* (see Section 8.3.2). We then divide the factorisation into blocks $B_1, \ldots, B_{m'}$, each containing $(K+2)$ consecutive LZ-factors. **At the first stage**, we compute subruns of those K-squares which touch the heads of all blocks $B_1, \ldots, B_{m'}$. For each block B_i, we find the subruns of

K-squares which touch the head of B_i but not that of B_{i+1}. This is done using algorithm MISMATCH-RIGHT-SUBRUNS and its couterpart for the left-rooted squares, together with Lemma 8.8.5. Let $\widehat{b}_i$ be the head position of B_i. Then for each period p, K-subruns of period p found at this step belong to the interval $[\widehat{b}_i - 1 .. \min\{\widehat{b}_i + 2p, \widehat{b}_{i+1} - 2\}]$. We call this interval the *explored interval* for $\widehat{b}_i$ and p. For each p, K-subruns found at this step can be seen as non-intersecting subintervals of this explored interval. These K-subruns are stored into a double-linked list, say $\mathcal{L}(i,p)$, in the increasing order of positions. (We leave it to the reader to check that such a list can be easily computed by the algorithm MISMATCH-RIGHT-SUBRUNS by making at each step a constant amount of extra work.) For $p > (\widehat{b}_i - \widehat{b}_{i-1})/2 - 1$, the explored interval for $\widehat{b}_{i-1}$ has to be joined with the explored interval for $\widehat{b}_i$, thus forming a bigger explored interval. Accordingly, lists $\mathcal{L}(i-1,p)$ and $\mathcal{L}(i,p)$ are joined. Note that if the last subrun of $\mathcal{L}(i-1,p)$ turns out to be adjacent to the first subrun of $\mathcal{L}(i,p)$, then those two subruns are merged into a single one. All additional operations take a constant time, and the resulting complexity of the first stage is still $O(nK)$.

The second stage is modified in a similar way. Let b_i denote the head position of factor f_i. Recall that at each call of PROCESS-BLOCK we are searching for K-squares occurring between some factor head $b_{j'}$ and another factor head $b_{j''}$, and touching some factor head b_i ($j' < i < j''$). Moreover, no factor head between $b_{j'}$ and $b_{j''}$ has been processed yet. In this case, the explored interval is $[\max\{b_{j'} + 2p + 1, b_i - 1\} .. \min\{b_i + 2p, b_{j''} - 2\}]$, and we may have to merge it either with the previous explored interval, or with the next one, or both.

After the first and the second stages, we have computed lists of subruns of K-squares that touch all factor heads. Each list stores subruns of K-squares of some fixed period p touching heads of some successive factors $f_i, f_{i+1}, ..., f_j$. Note that for each factor and each period, the corresponding list exists but can be empty. Each such list is accessed through two pointers, associated with the corresponding leftmost (f_i) and rightmost (f_j) factors. We denote these pointers $\mathit{left}_i(p)$ and $\mathit{right}_j(p)$ respectively. These pointers are needed, in particular, for merging explored intervals at the second stage. An important remark is that at each moment there are only $O(n)$ pointers that need to be stored. The key observation is that for each factor head b_i, pointer $\mathit{left}_i(p)$ should be defined only for periods $p \leq (b_i - b_{i'})/2 - 1$, where $b_{i'}$ is the closest head on the left of b_i that has been processed before. Similarly, pointer $\mathit{right}_i(p)$ is defined only for periods $p \leq (b_{i''} - b_i)/2 - 1$, where $b_{i''}$ is the closest head to the right of b_i that has been processed before. On the other hand, all pointer manipulations add only a constant amount of work to each step of the second stage, and then the time complexity of the second stage stays $O(nK \log K)$.

At **the third stage**, we have to find those K-subruns which lie entirely inside LZ-factors. For each period, potential occurrences of these K-subruns correspond precisely to the gaps between explored intervals. Thus, the third stage can be also seen as closing up, for each period, the gaps between explored intervals. The goal of the third stage is to construct, for each period p, a single list $\mathcal{L}[p]$ of all K-subruns of period p occurring in the word. In the beginning of the third stage, $\mathcal{L}[p]$ is initialized to $left_1(p)$.

As before, the key observation here is the fact that each LZ-factor without its head has a copy on the left (here required to be non-overlapping), and the idea is again to process w from left to right and to retrieve the K-subruns occurring inside each LZ-factor from its copy. However, the situation here is different in comparison to the previous section. One difference is that here subruns have to be "copied forward" when we process a factor copy, rather than to be retrieved at the time of processing the factor itself, as done at the second stage of Maximal-Repetitions. Moreover, we may have to "cut out", from a longer list, a sublist of K-subruns belonging to a factor copy and then to "fit" it into the gap between two explored intervals. The "cutting out" may entail splitting K-subruns which span over the borders of the factor copy, and "fitting into" may entail merging those K-subruns with K-subruns from the neighboring explored intervals. Below we describe the algorithm for the third stage, which copes with these difficulties. The algorithm Runs-Third-Stage given below provides a detailed description of the third stage.

During the computation of the Lempel-Ziv factorization, for each LZ-factor $f_i = va$ we choose a copy of v occurring earlier and point from the end position of this copy to the head position of f_i. It may happen that one position has to have several pointers, in which case we organize them in a list. We traverse w from left to right and maintain the rightmost K-run, of each period, which starts before the current position. This K-run is called the *active run* and is denoted $A[p]$ in the Runs-Third-Stage algorithm. To this purpose, we also maintain the following invariant: at the moment we arrive at a position i, we have the list, denoted $S[i]$, of all K-subruns which start at this position. The lists $S[i]$ are maintained according to the following general rule: for each K-subrun starting at the current position, we assign the start position of the next K-subrun in the list provided that this K-subrun exists (instructions 16-19 of Runs-Third-Stage). If the next subrun does not exist, we set a special flag *islast* in order to do it later.

When we arrive at the end position of a copy of a LZ-factor, we have to copy "into the factor" all the K-subruns which this copy contains. Therefore, we scan *backwards* the K-subruns contained in the copy and copy them into the factor (instructions 24-29). After copying these K-subruns, we bridged two explored intervals into one interval, and linked together the two corresponding lists of K-subruns, possibly inserting a new list of K-runs in between (line 30). Copying K-subruns in the backward direction is important for the correction of the algorithm – this guarantees that no K-subruns are missed. It is also for this reason that we need the copy to be non-overlapping with the factor.

The final part of the algorithm (lines 31-40) treats the situation when before executing line 30, $right_{s-1}(p)$ actually refers to the current list $\mathcal{L}[p]$. If, in addition, $\mathcal{L}[p]$ was empty before but became non-empty after the execution of line 30, we have to add the first subrun of $\mathcal{L}[p]$ to the corresponding list $S[i']$ (lines 31-35). If the active run $A[p]$ was the last run in $\mathcal{L}[p]$ before the execution of line 30 but is not the last one after this execution, we have to update the list $S[i']$ for the start position i' of the next subrun (instructions 36-40).

```
Runs-Third-Stage()
▷ A[p] is maintained to be the last considered run of period p
▷ S[i] is maintained to be the list of runs starting at i
for each period p do
    L[p] ← left_1(p)
    islast[p] ← false
    if L[p] is not empty then
        r ← first run of L[p]
        i ← start position of r
        add r to S[i]
        isempty[p] ← false
    else isempty[p] ← true
for each position i ∈ [1..n] do
    for each run r ∈ S[i] do
        p ← period of r
        A[p] ← r
        if r is not the last run in L[p] then
            r' ← run next to r in L[p]
            i' ← first position of r'
            add r' to S[i']
        else islast[p] ← true
    for each factor copy v ending at position i do
        s ← index of the factor corresponding to v
        for each period p ≤ |v|/2 do
            r ← A[p]
            while r is defined and r contains K-squares inside v do
                r' ← subrun of all these K-squares
                r'' ← copy of r' in f_s
                add/merge r'' to/with the head of list left_s(p)
                r ← predecessor of r in L[p]
            link/merge right_{s-1}(p) to/with left_s(p)
            if isempty[p] and L[p] is no more empty then
                r' ← first run of L[p]
                i' ← start position of r'
                add r' to S[i']
                isempty[p] ← false
            if islast[p] and A[p] is no more the last run in L[p] then
                r' ← run next to A[p] in L[p]
                i' ← start position of r'
                add r' to S[i']
                islast[p] ← false
```

After the whole word has been traversed, no more gaps between explored intervals exist anymore. This means that for each period p, $\mathcal{L}[p]$ is the list of all K-subruns of period p occurring in the word, which are actually the searched runs.

The complexity of the third stage is $O(n+S)$, where S is the number of resulting K-runs. We show this by an amortized analysis of the Runs-Third-Stage algorithm. Specifically, we show that the *total* number of iterations of each loop in Runs-Third-Stage is either $O(n)$ or $O(S)$. Each iteration of the **for**-loop at line 13 processes a new K-run starting at position i. Therefore there are $O(S)$ iterations of this loop during the whole execution. Each iteration of the **for**-loop at line 21 treats a copy of a distinct Lempel-Ziv factor. Furthermore, the number of iterations of the nested **for**-loop at line 23 is the half of the length of the corresponding factor. Therefore, the total number of iterations of both **for**-loops is $O(n)$. On the other hand, the **while**-loop at line 25 iterates $O(n+S)$ times, as at each iteration, except possibly the first and the last one, it computes a new K-subrun, which becomes a completed K-run at that point. Thus, the *overall* time spent by all internal loops is $O(n+S)$. The main **for**-loop (line 3) makes obviously $O(n)$ iterations, and this completes the proof that the whole complexity of the third stage is indeed $O(n+S)$.

Putting together the three stages, we obtain the main result of this section.

Theorem 8.8.7. *All K-runs can be found in time $O(nK\log K+S)$ where n is the word length and S is the number of K-runs found.*

Once all K-runs have been found, we can easily output all K-squares. We then have the following result.

Corollary 8.8.8. *All K-squares can be found in time $O(nK\log K+S)$ where n is the word length and S is the number of K-squares found.*

8.9. Notes

Section 8.0. We refer to Storer 1988 for applications of repetitions to compression techniques. Galil and Seiferas 1983, Crochemore and Rytter 1995, Cole and Hariharan 1998 illustrate how repetitions are used in pattern matching. Kolpakov et al. 2003 discusses the role of repetitions in DNA sequences. More on biological origin and function of repeated sequences in genome sequences can be learnt, e.g., in Brown 1999.

Section 8.1. Basic definitions and results of word combinatorics, including Proposition 8.1.1 and Theorem 8.1.2 and Proposition 8.1.5, can be found in Lothaire 1997. The exponent is called the *order* in Chapter 8 of Lothaire 2002. Maximal repetitions were called *maximal periodicities* in Main 1989 and *runs* in Iliopoulos et al. 1997.

Section 8.2. The proof of Lemma 8.2.2 is attributed to D. Hickerson and was communicated to us by M. Crochemore and D. Gusfield. The lemma is a weaker and easier-to-prove version of a result from Crochemore and Rytter 1995 asserting that under conditions of Lemma 8.2.2, the stronger inequality $|y|+|z|\leq|x|$

holds. The latter implies that the number of primitively-rooted squares occurring as prefixes of a word w is less than $\log_\varphi |w|$ (φ is the golden ratio), which is a better bound than $\log_{\sqrt{2}} |w|$ implied by Lemma 8.2.2 (cf Theorem 8.2.1). Moreover, the bound $\log_\varphi |w|$ is asymptotically tight, as realized by Fibonacci words.

Lemma 8.2.3(i) is a "folklore result" (see e.g. Pirillo 1997), together with the fact that the common prefix of $f_{n-1}f_{n-2}$ and $f_{n-2}f_{n-1}$ of length $F_n - 2$ is a palindrome. Lemma 8.2.3(ii) appeared in De Luca 1981. The proof given here was communicated to us by J. Berstel. Theorem 8.2.4 was proved in Crochemore 1981.

Lemma 8.2.3(iii) was proved in the PhD thesis of P. Séébold (1985). Lemma 8.2.3(iv) appeared in Karhumäki 1983. Later, repetitions in Fibonacci words have been extensively studied in Mignosi and Pirillo 1992, Pirillo 1997 where it was proved, in particular, that they contain no repetition of exponent greater than $2+\varphi = 3.618..$ but do contain repetitions of exponent greater than $2+\varphi-\varepsilon$ for every $\varepsilon > 0$.

An exact formula for the number of squares Fibonacci word f_n was obtained in Fraenkel and Simpson 1999. It implies that this number is asymptotically $\frac{2}{5}(3-\varphi)nF_n + O(F_n) \approx 0.7962 \cdot F_n \log_2 F_n + O(F_n)$.

It is interesting to note that if we count *distinct* squares rather than square occurrences, their maximal number is asymptotically linearly bounded on the word length. In Fraenkel and Simpson 1999, it has been shown that Fibonacci word f_n contains $2(F_{n-2}-1) = 2(2-\varphi)F_n + O(1)$ distinct squares. In Fraenkel and Simpson 1998, it has been proved that the number of distinct squares in general words of length n is bounded by $2n$ (for an arbitrary alphabet). It is conjectured that this number is actually smaller than n. Thus, in contrast to square occurrences, the maximal number of distinct squares is linear.

A linear bound on the number of maximal repetitions in Fibonacci words was first obtained in Iliopoulos et al. 1997 by presenting a linear-time algorithm enumerating all of those. The direct formula given here was obtained in Kolpakov and Kucherov 2000b. Since Fibonacci words don't contain exponents greater than $(2+\varphi)$, Theorem 8.2.5 implies that the sum of exponents of all maximal repetitions in f_n is no greater than $(2+\varphi)(2F_{n-2}-3) = 2(2-\varphi)(2+\varphi)F_n+O(1) = 2(3-\varphi)F_n+O(1) \approx 2.764 \cdot F_n$. While an exact formula for the sum of exponents is not known, a more precise estimate was obtained in Kolpakov and Kucherov 2000b, where it was shown that the sum of exponents of all maximal repetitions in Fibonacci word f_n is $(C \cdot F_n + o(F_n))$, where $1.922 \leq C \leq 1.926$.

A complete proof of Theorem 8.2.7 can be found in Kolpakov and Kucherov 2000b.

On a different but related topic, several studies have been done on the *minimal*, rather than maximal, number of repetitions in words. In particular, it is well-known (Lothaire 1997) that an arbitrary long word with no squares (and therefore no repetitions at all) can be constructed on a three-letter alphabet. In Fraenkel and Simpson 1995 it was shown that for the binary alphabet, there exists an infinite word containing three *distinct squares* (e.g. 00, 11, 0101) and three is the minimal bound. Complementary, in Kucherov et al. 2003 it was shown that the minimal number of square occurrences in an infinite binary word is, in the limit, a constant fraction of the word length, and that this constant is 0.55080....

Section 8.3. Longest extension functions have been introduced in Main and Lorentz 1984; a closely related idea was used independently in Crochemore 1983. The algorithm presented here for computing longest extension functions is a refinement of the algorithm of Main and Lorentz 1984.

Using s-factorization (called f*-factorization* in Crochemore and Rytter 1994) for finding repetitions was first proposed in Crochemore 1983. The Lempel-Ziv factorization is directly related to the Lempel-Ziv compression algorithm (Ziv and Lempel 1977) and to the underlying definition of complexity of a string (Lempel and Ziv 1976). A discussion on two types of factorization can be found in Gusfield 1997. The linear-time computation of Lempel-Ziv factorization was used in Rodeh et al. 1981. Linear-time construction algorithms for the suffix tree are described in McCreight 1976, Ukkonen 1995 and for the DAWG in Blumer et al. 1985, Crochemore 1986.

Section 8.4. First papers on finding repetitions in words are Crochemore 1981, Slisenko 1983. Crochemore 1981 proposed an $O(n \log n)$ algorithm for finding all occurrences of non-extensible primitively-rooted integer powers in a word. Using a suffix tree technique, Apostolico and Preparata 1983 described an $O(n \log n)$ algorithm for finding all *right-maximal* repetitions, which are repetitions that cannot be extended *to the right* without increasing the period. Main and Lorentz 1984 proposed another algorithm that finds all maximal repetitions in $O(n \log n)$ time. They also pointed out the optimality of this bound under the assumption of unbounded alphabet and under the restriction that the algorithm is based only on letter comparisons.

Crochemore 1983 described a simple and elegant *linear-time* algorithm for finding a square in a word (and thus checking if a word is repetition-free). Another linear algorithm checking whether a word contains a square was proposed in Main and Lorentz 1985.

Using s-factorization, Main 1989 proposed a *linear-time* algorithm which finds all *leftmost* occurrences of distinct maximal repetitions in a word. This algorithm basically corresponds to the first stage of MAXIMAL-REPETITIONS. In particular, Theorems 8.4.2 and 8.4.1 are from Main 1989. The linear-time algorithm presented here is from Kolpakov and Kucherov 1999.

As far as other related works are concerned, Kosaraju 1994 described an $O(n)$ algorithm which, given a word, finds for each position the shortest square starting at this position. Stoye and Gusfield 1998 proposed several algorithms that are based on a unified suffix tree framework. Their results are based on an algorithm which finds in time $O(n \log n)$ all *branching tandem repeats*.

Section 8.5, 8.6. The results of those sections are from Kolpakov and Kucherov 2000a. A more general problem has been considered in Brodal et al. 2000: find all gapped repeats with a gap size belonging to a specified interval. The proposed algorithm has the time complexity $O(n \log n + S)$, where S is the size of the output.

Section 8.7. We refer to Duval 1998, Duval et al. 2001 for studies of properties of local periods. The Critical Factorization Theorem is presented in Lothaire 1997, Choffrut and Karhumäki 1997, Lothaire 2002. For recent developments of the Critical Factorization Theorem see Mignosi et al. 1995.

The results of Section 8.7 are based on Duval et al. 2003.

Section 8.8. The problem of finding approximate squares for both Hamming and edit distances has been first studied in Landau and Schmidt 1993. Computing generalized longest extension functions in time $O(nK)$ can be done by a method based on the suffix tree and the computation of the *nearest common ancestor* described in Gusfield 1997.

The results presented in the section are from Kolpakov and Kucherov 2003.

Algorithms of Sections 8.4 and 8.8 have been implemented in the `mreps` software for finding tandem repeats in DNA sequences. For more information about the software and experimental results obtained with it, we refer to the Web-site `http://www.loria.fr/mreps/` and the publication Kolpakov et al. 2003.

CHAPTER 9

Counting, Coding and Sampling with Words

9.0. Introduction

This chapter illustrates the use of words to derive enumeration results and algorithms for sampling and coding.

Given a family $\mathcal{C}$ of combinatorial structures, endowed with a size such that the subset $\mathcal{C}_n$ of objects of size n is finite, we consider three problems:

– *Counting:* determine for all $n \geq 0$, the cardinal $\mathrm{Card}(\mathcal{C}_n)$ of the set $\mathcal{C}_n$ of objects with size n.

– *Sampling:* design an algorithm $\textsc{Rand}\mathcal{C}$ that, for any n, produces a random object uniformly chosen in $\mathcal{C}_n$: in other terms, the algorithm must satisfy, for any object $O \in \mathcal{C}_n$, $\mathbf{P}(\textsc{Rand}\mathcal{C}(n) = O) = 1/\operatorname{Card}(\mathcal{C}_n)$.

– *Optimal coding:* construct a function φ that maps injectively objects of $\mathcal{C}$ on words of $\{0,1\}^*$ in such a way that an object O of size n is coded by a word $\varphi(O)$ of length roughly bounded by $\log_2 \operatorname{Card}(\mathcal{C}_n)$.

These three problems have in common an *enumerative* flavour, in the sense that they are immediately solved if a list of all objects of size n is available. However, since in general there is an exponential number of objects of size n in the families we are interested in, this solution is by no way satisfying. For a wide class of so-called *decomposable* combinatorial structures, including non ambiguous algebraic languages, algorithms with polynomial complexity can be derived from the rather systematic recursive method. Our aim is to explore classes of structures for which an even tighter link exists between counting, sampling and coding.

For a number of natural families of combinatorial structures, the counting problem has indeed a "nice" solution: by nice could be intended that there is a simple formula for $\operatorname{Card}(\mathcal{C}_n)$, that the generating series $\sum_{n\geq 0} \operatorname{Card}(\mathcal{C}_n)x^n$ is an algebraic function, etc. The rationale of this chapter is that these nice enumerative properties are the visible "traces" of deeper structural properties, and that making the latters explicit is a way to solve simultaneously and simply the three problems above.

The enumeration of walks on lattices (Section 9.1) is an inextinguishable source of nice counting formulas. These formulas can often be given simple interpretations by viewing walks as words on an alphabet of steps, and using ingredients of the combinatorics of words. In particular we shall consider some rational and algebraic languages, shuffles and the cycle lemma.

Convex or directed polyominoes (Section 9.2) illustrate the idea that nice combinatorial properties help for sampling. Since enumeration and random generation of general polyominoes appear intractable, it was proposed in statistical physics to study subclasses like convex or directed polyominoes, that display better enumerative properties. These objects can be described in terms of simple languages, often algebraic, and this leads to efficient random generators.

The family of planar maps (Section 9.3) is a further example of class with unexpectedly nice enumerative properties. Maps are the natural combinatorial abstraction for embeddings of graphs in the plane and for polygonal meshes in computational geometry, and maps were also largely studied in theoretical physics. Toy models of statistical physics, like percolation or the Ising model, are often studied on regular lattices, but also on random maps. The uniform distribution indeed appears to give, at the discrete level, the right notion of distribution of probability on possible universes as prescribed by quantum gravity. In these various contexts, results have been obtained independently on counting, sampling and coding problems. Again we rely on a combinatorial explanation of the enumerative properties of planar maps to approach these three problems.

Most of the time, we state and prove results for some particularly simple structures, while they are valid for more generic families (*e.g.* walks with more general steps, polyominoes on other lattices, maps with constraints). We made this choice to maintain the chapter relatively short, but also because on these simple structures the "traces" are more visible, and the underlying combinatorics appears more explicitly.

All the objects that are considered in this chapter have nice geometric interpretations in the plane. We have chosen to rely on the geometric intuition of the reader to support these interpretations, and concentrate the proofs on the combinatorial aspects.

9.1. Counting: walks in sectors of the plane

A *(nearest neighbor) walk* on the square lattice $\mathbb{Z}^2$ is a finite sequence of vertices $w = (w_0, w_1, \ldots, w_n)$ in $\mathbb{Z}^2$ such that each *step* $w_i - w_{i-1}$, for $1 \leq i \leq n$, belongs to the set $\mathcal{S} = \{(0,1), (0,-1), (-1,0), (1,0)\}$. The number of steps n is the *length* of w; w_0 and w_n are respectively its startpoint and endpoint. The *reverse* walk of w is the walk $\bar{w} = (w_n, w_{n-1}, \ldots, w_1, w_0)$. A *loop* is a walk with identical startpoint and endpoint.

Elements of $\mathcal{S}$ are also denoted u, d, l, r – standing for *up, down, left* and *right*. Unless explicitly specified, we consider walks up to translation, or equivalently, we assume that they start from the origin $(0,0)$. A walk can thus be seen as a word on the alphabet $\mathcal{S} = \{u, d, l, r\}$ and we identify the set of walks with the language $\{u, d, l, r\}^*$, making no distinction between both of them.

In the rest of this section, we study families of walks with various boundary constraints: on a line, a half line, a half plane, a quarter plane, and finally, on the slitplane. This is the occasion to introduce enumerative tools that will be of use in later sections.

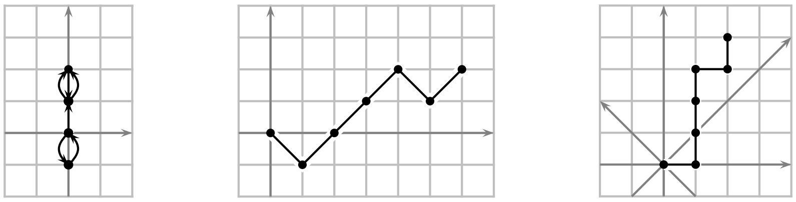
9.1.1. Unconstrained walks and rational series

Let us first consider walks that use only vertical steps (*i.e.* u or d), and hence stay on the axis $(x = 0)$. These walks are sometimes called *one-dimensional simple symmetric walks*, and are often considered in their "time stretched" version: each step u or d is replaced by a $(1,1)$ or $(1,-1)$ step, in order to give an unambiguous representation in the plane, as illustrated by Figure 9.1. Up to a $\pi/4$-rotation, these walks are in one-to-one correspondence with walks with steps in $\{u, r\}$ and as such, are sometimes called *staircase walks*, or *directed two-dimensional walks*.

Counting these walks with respect to their length ℓ amounts to counting words on $\{u, d\}$ of length ℓ, and there are 2^ℓ of those. Restricting them to end at ordinate j, with $\ell = 2n + |j|$ for some nonnegative n, is hardly more difficult: for $j \geq 0$, the corresponding words are arbitrary shuffles of $n + j$ letters u and n letters d, and similarly for $j \leq 0$, they are shuffles of n letters u and $n - j$ letters d. Hence the number of walks of length $2n + |j|$ ending at ordinate j is

$$\binom{2n + |j|}{n}.$$

It will be convenient to express enumerative results in terms of languages and generating functions. In this case, the language $\mathcal{V}$ of walks on the vertical



(a) On an axis, (b) stretched, (c) and rotated.

Figure 9.1. Three representations of the one-dimensional walk *duuudu*.

axis is just $\{u,d\}^*$. Equivalently, in the algebra $\mathbb{Q}\langle\langle u,d\rangle\rangle$ of formal power series in non commuting variables, the language $\mathcal{V}$ (viewed as the formal sum of its words) is uniquely defined by the linear equation:

$$\mathcal{V} = \varepsilon + (u+d)\mathcal{V}, \tag{9.1.1}$$

which corresponds to the non ambiguous decomposition: "a walk is either the empty walk or made of a step u or d followed by a walk".

Define now $\delta(w) = |w|_u - |w|_d$ for any word w on $\mathcal{S}$, so that $\delta(w)$ is the *final ordinate* of the walk w. The *generating function* of the language $\mathcal{V}$ with respect to the length (variable t) and the final ordinate (variable y) is

$$V(t;y) = \sum_{w\in\mathcal{V}} t^{|w|}y^{\delta(w)},$$

which is an element of the algebra $\mathbb{Q}(y)[[t]]$ of formal power series in the variable t with coefficients that are rational functions in y.

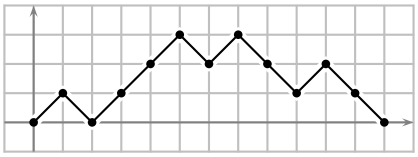
Observe that $|.|$ and δ are morphisms of monoïds $(\mathcal{S}^*,\cdot)\to(\mathbb{Z},+)$, so that $V(t;y)$ can be viewed as the commutative image of $\mathcal{V}$ by the morphism of algebra $w\mapsto t^{|w|}y^{\delta(w)}$ from $\mathbb{Q}\langle\langle u,d\rangle\rangle$ to $\mathbb{Q}(y)[[t]]$. Taking the commutative image of Equation 9.1.1, the generating function $V(t;y)$ satisfies:

$$V(t;y) = 1 + (ty + ty^{-1})V(t;y).$$

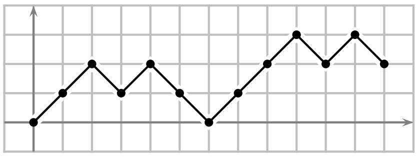
An explicit expression of $V(t;y)$ follows, and its expansion of course agrees with the previous direct enumeration:

$$V(t;y) \;=\; \frac{1}{1-(y+y^{-1})t} \;=\; \sum_{m=0}^{+\infty}\sum_{k=0}^{m}\binom{m}{k}t^m y^{m-2k}.$$

The commutative image mechanism produces *a priori* a formal power series of $\mathbb{Q}(y)[[t]]$, but, as in the present example, it retains properties of the initial language: the series $V(t;y)$ of the rational language $\{u,d\}^*$ is a rational function of t and y, *i.e.* belongs to $\mathbb{Q}(t,y)$. Walks with more general steps are dealt with in a similar way: for instance the language $\mathcal{W}$ associated to walks in $\mathbb{Z}^2$ is $\mathcal{S}^*$



(a) a Dyck word,



(b) a Dyck prefix.

Figure 9.2. The family of Dyck words (stretched representations).

and the generating function of these walks with respect to the length and the coordinates of the endpoint is:

$$W(t;x,y) = \frac{1}{1-(x+x^{-1}+y+y^{-1})t}.$$

Another illustration is given by the family of walks that never immediately undo a step they have just done. Their language is the set of words avoiding the factors $\{ud, du, lr, rl\}$ which is well known to be rational. Accordingly their generating function with respect to the length and the coordinate of the endpoints belongs to $\mathbb{Q}(t,x,y)$. Conversely, when the generating function of a set of objects is rational, it is natural to try to encode them by words of a rational language.

9.1.2. Walks on a half line and Catalan's factorization

We shall now consider walks that stay on the upper half axis ($x = 0,\ y \geq 0$). More precisely let the *depth* of w be the absolute value of the minimal ordinate $\delta(v)$ for all prefixes v of w. Walks that stay on the upper half axis are exactly the walks with depth zero, and this condition is called the *nonnegative prefix condition*. Loops satisfying the nonnegative prefix condition are often called *Dyck words* on the alphabet $\{u,d\}$. In turn, walks satisfying the nonnegative prefix condition are sometimes referred to as *Dyck prefixes*, since any of them can be completed into a Dyck word. See Figure 9.2 for examples. Let $\mathcal{D}$ denote the language of Dyck words and $\mathcal{D}_n$ the set of Dyck words of length $2n$. The following lemma gives a central role to Dyck words.

Lemma 9.1.1 (Catalan's factorization). *The language $\{u,d\}^*$ of one-dimensional walks admits the following non ambiguous decomposition:*

$$\{u,d\}^* = (\mathcal{D}d)^*\mathcal{D}(u\mathcal{D})^*.$$

More precisely, the language of walks with depth ℓ and ending at ordinate j is

$$(\mathcal{D}d)^\ell\mathcal{D}(u\mathcal{D})^{j+\ell}$$

Proof. For any word w on the alphabet $\{u,d\}$ with depth ℓ and final ordinate j, such a factorization is obtained at first passages from ordinate $i+1$ to i for

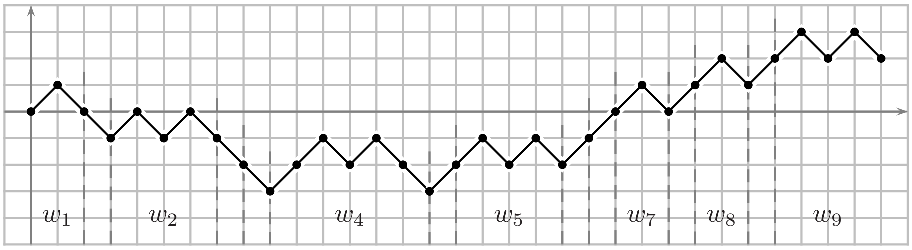


Figure 9.3. Catalan's factorization of a walk in $(\mathcal{D}d)^3\mathcal{D}(u\mathcal{D})^5$.

$i = -1, \dots, -\ell$ and last passages from ordinate i to $i+1$ for $i = -\ell, \dots, j-1$. The uniqueness of the decomposition follows from the fact that any strict prefix v of a word in $\mathcal{D}d$ satisfies $\delta(v) \geq 0$ by definition of $\mathcal{D}$, and hence does not belong to $\mathcal{D}d$. ■

Catalan's factorization immediately allows us to derive the total number of walks on the half line.

Proposition 9.1.2. *The number of Dyck prefixes of length m is*

$$\binom{m}{\lfloor \frac{m}{2} \rfloor}.$$

Proof. A Dyck prefix of even length is a walk with depth zero and even final ordinate $2j$ for some integer $j \geq 0$. According to Lemma 9.1.1, the language of these words is $\mathcal{D}(u\mathcal{D})^{2j}$. Upon changing the j first factors u in factors d, words of length $2n$ in this language are in bijection with words of length $2n$ in the language $(\mathcal{D}d)^j\mathcal{D}(u\mathcal{D})^j$, *i.e.* with words of the language of loops with depth j. Hence Dyck prefixes of length $2n$ are in bijection with loops of the same length, and their number is $\binom{2n}{n}$.

Similarly, a Dyck prefix of odd length ends at ordinate $2j+1$, for some $j \geq 0$. But words of equal length in the languages $\mathcal{D}(u\mathcal{D})^{2j+1}$ and $(\mathcal{D}d)^j\mathcal{D}(u\mathcal{D})^{j+1}$ are in bijection. The union of the last languages for all $j \geq 0$ is the set of words w with $\delta(w) = 1$, $\binom{2n+1}{n}$ of which have length $2n+1$. ■

The previous proof can be summarized as follows: find a factorization into Dyck factors separated by some specific steps (typically first or last passages), and then reorganize the factorization without modifying the Dyck factors. We shall apply this principle again to give a bijective enumeration of Dyck words.

Proposition 9.1.3. *The number of loops of length $2n$ that stay on the half axis $(x = 0,\ y \geq 0)$ is the n-th* Catalan number*:*

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Proof (as a corollary of Proposition 9.1.2). Removing the last step of a Dyck prefix of length $2n+1$ yields a prefix of length $2n$. In this way every Dyck prefix of length $2n$ is obtained twice, except for Dyck paths that are obtained only once. Hence $\binom{2n+1}{n} = 2\binom{2n}{n} - \operatorname{Card} \mathcal{D}_n$, and the formula follows. ∎

Proof (direct bijection). We prove the relation $(n+1) \operatorname{Card} \mathcal{D}_n = \binom{2n}{n}$ by giving a bijection between the set of pairs (v, v') with $vv' \in \mathcal{D}_n$ and v empty or ending with a letter u, and the set of loops of length $2n$. To do that we first state two factorizations that follow from Lemma 9.1.1:

- the set of pairs (v, v') as above with $\delta(v) = \ell$ is $(\mathcal{D}u)^\ell \times \mathcal{D}(d\mathcal{D})^\ell$;
- the set of loops with depth ℓ is $(\mathcal{D}d)^\ell \mathcal{D}(u\mathcal{D})^\ell$.

Exchanging u and d factors in these decompositions leads to the announced bijection. ∎

The same idea allows to refine the enumeration of Dyck prefixes.

Proposition 9.1.4. *The number of Dyck prefixes of length $2n+j$ and final ordinate $j \geq 0$ is*

$$\frac{j+1}{n+j+1}\binom{2n+j}{n}.$$

Proof. We prove the formula by giving a bijection between pairs (w, i) where w is a walk with $\delta(w) = j$ and $i \in \{0, \ldots, j\}$, and pairs (w', k) where w' is a Dyck prefix with $\delta(w') = j$ and $k \in \{0, \ldots, n+j\}$:

- to any pair (w, i) as above, associate $(w_i, \ldots, w_j, w_0, \ldots, w_{i-1})$ where w_0 is the loop and the other w_ℓ are the Dyck paths such that $w = w_0 u w_1 \cdots u w_j$ (this is the decomposition at the last passages at level $0, \ldots, j$).
- to any pair (w', k) as above, associate $(w'_0, \ldots, \widehat{w}'_i, \ldots, w'_j)$, where the w'_ℓ are the Dyck words such that $w' = w'_0 u w'_1 \cdots u w'_j$, i is the index of the w'_i containing or following the kth letter u in the word uw', and $\widehat{w}'_i = (v, v')$ is the factorization of w'_i after this letter.

The bijection in the second proof of Proposition 9.1.3 allows to transform the pair $\widehat{w}'_i = (v, v')$ in a loop, so that both sets are associated to the same set of sequences of $j+1$ walks. ∎

9.1.3. Walks on a half plane and algebraic series

Walks in the half plane $(y \geq 0)$ are hardly more complicated to enumerate than walks on the half line. Indeed, as words on the alphabet $\mathcal{S}$, these walks are completely characterized by the fact that all their prefixes v contain at least as many letters u as letters d. Hence the associated language is the set of shuffles of vertical Dyck prefixes with sequences of horizontal steps. Various formulas can be derived from this characterization: for instance, the number of loops of length $2n$ that stay in the half plane $(y \geq 0)$ is

$$\sum_{k=0}^{n} \binom{2n}{2k}\binom{2k}{k} C_{n-k}.$$

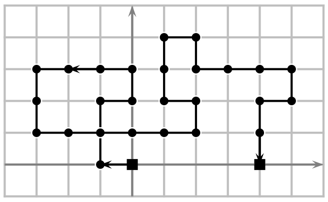


Figure 9.4. An excursion in the half plane.

Rather than going further in this direction, we shall observe that the set of these walks is an algebraic language and return to generating functions. Consider the alphabet $\mathcal{A}_k = \{u, d, x_1, \dots, x_k\}$, and the monoïd morphism δ defined as previously by $\delta(w) = |w|_u - |w|_d$. The language $\mathcal{M}^{(k)}$ of *k-colored Motzkin words* is the set of words w on the alphabet $\mathcal{A}_k$ satisfying $\delta(w) = 0$ and the nonnegative prefix property. For $k = 0$ this is the Dyck language. For $k = 2$, upon setting $x_1 = l$, $x_2 = r$, bicolored Motzkin words are *excursions in the half plane*, *i.e.* walks in the half plane ($y \geq 0$) that finish on the axis ($y = 0$).

The language of k-colored Motzkin words admits an algebraic description:

$$\mathcal{M}^{(k)} = \varepsilon + (x_1 + \ldots + x_k)\mathcal{M}^{(k)} + u\mathcal{M}^{(k)} d\mathcal{M}^{(k)}, \tag{9.1.2}$$

which derives immediately from the non ambiguous decomposition of any non empty Motzkin word at its smallest non empty prefix v such that $\delta(v) = 0$. Taking the commutative image, the generating function $M^{(k)}(t) = \sum_{w \in \mathcal{M}^{(k)}} t^{|w|}$ of the Motzkin language with respect to the length satisfies the equation:

$$M^{(k)}(t) = 1 + ktM^{(k)}(t) + t^2 M^{(k)}(t)^2. \tag{9.1.3}$$

Observe that this equation completely determines $M^{(k)}(t)$, since it has a unique solution in the space of formal power series in the variable t (as can be checked by induction, extracting the coefficient of t^n on both sides).

Any additive parameter can be taken into consideration in the commutative image. For instance the previous algebraic decomposition yields the following proposition in the case of bicolored Motzkin words.

Proposition 9.1.5. *The generating function for walks in the half plane returning to the axis ($y = 0$), with respect to their length, abscissa of the endpoint and number of vertical steps, is:*

$$M^{(2)}(t; x, z) = \frac{1 - t(x + \frac{1}{x}) - \sqrt{[1 - t(x + \frac{1}{x} + 2z)][1 - t(x + \frac{1}{x} - 2z)]}}{2t^2 z^2}.$$

Proof. Taking the commutative image with the map $w \to t^{|w|} x^{|w|_r - |w|_l} z^{|w|_u + |w|_d}$ yields the equation

$$M^{(2)}(t; x, z) = 1 + t(x + \frac{1}{x})M^{(2)}(t; x, z) + t^2 z^2 M^{(2)}(t; x, z)^2.$$

The discriminant of this equation is

$$\Delta(t;x,z) = [t(x+\frac{1}{x})-1]^2 - 4t^2z^2,$$

and among the two roots of the quadratic equation, only the one of the proposition is a formal power series in t. ∎

Equation (9.1.3) shows that the series $M^{(k)}(t)$ satisfies a relation of the form $P(M^{(k)}(t),t) = 0$ with P a polynomial, which means that it is an *algebraic* formal power series. This illustrates the fact that algebraic languages that admit a non ambiguous algebraic description naturally have algebraic generating functions with respect to additive parameters. Conversely, when the generating function of a set of objects is algebraic, one would like to obtain it from an algebraic description of the objects (or more formally from an encoding of the objects by the words of an algebraic language with a non ambiguous description). In this sense, Equation (9.1.2) is more satisfying than Catalan's factorization, even though the commutative image of the latter also induces an algebraic equation.

Expanding the generating function $M^{(2)}(t,1,1) = (1-2t-\sqrt{1-4t})/2t^2$ in powers of t, one observe the following amusing result (*cf.* Problem 9.1.5).

Corollary 9.1.6. *The number of bicolored Motzkin words of length n is given by the Catalan number C_{n+1}.*

Loops in the up diagonal quadrant ($x+y \geq 0,\ y \geq x$) are simple to describe: let w be such a loop of length $2n$, and consider the projections of the walk on the two diagonals ($x = y$) and ($x = -y$). Let $\{a,b\}$ be the elementary steps on these two axes, with a corresponding to up steps and b to down steps. Steps in $\mathbb{Z}^2$ have the following projections:

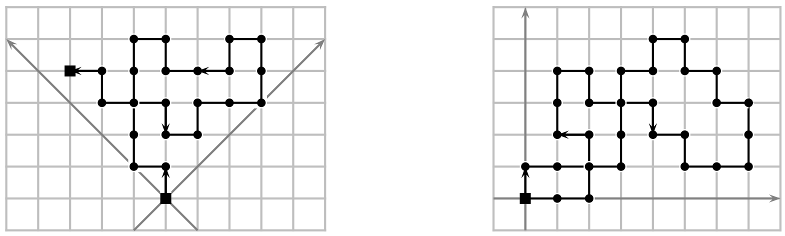
$$u \longrightarrow (a,a) \qquad d \longrightarrow (b,b) \qquad l \longrightarrow (b,a) \qquad r \longrightarrow (a,b)$$

and the projections of w on the diagonals are Dyck words of length $2n$ on $\{a,b\}$; reciprocally any pair of Dyck words of same length over this alphabet corresponds to a loop in the up diagonal quadrant. Hence:

Proposition 9.1.7. *The number of loops of length $2n$ that stay in the diagonal quadrant $(x+y \geq 0,\ y \geq x)$ is given by:*

$$C_n^2.$$

More generally, any walk of length $2n+|i|+j$ and endpoint (i,j) in the up diagonal quadrant is described by its projections on the two diagonal axes; these projections are decoupled Dyck prefixes of length $2n+|i|+j$ with respective ordinate of the endpoint $i+j$ and $j-i$. Hence:



(a) A walk in the diagonal up quadrant. (b) A loop in the first quadrant.

Figure 9.5. Walks in quadrants.

PROPOSITION 9.1.8. *The number of walks of length $2n+|i|+j$ and endpoint (i,j) that stay in the diagonal quadrant $(x+y\geq 0,\ y\geq x)$ is given by:*

$$\frac{(j+i+1)(j-i+1)}{(n+j+|i|+1)(n+j+1)}\binom{2n+|i|+j}{n+|i|}\binom{2n+|i|+j}{n},$$

and the total number of walks of length n that stay in the diagonal quadrant $(x+y\geq 0,\ y\geq x)$ is given by

$$\binom{n}{\lfloor\frac{n}{2}\rfloor}^2.$$

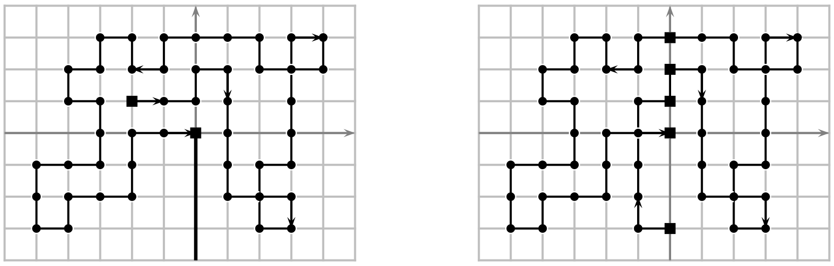
The case of loops in the first quadrant $(x\geq 0,\ y\geq 0)$ is quite similar. These loops are words w on $\mathcal{S}$ such that both restrictions of w to $\{u,d\}$ and to $\{l,r\}$ are Dyck words; hence the language of loops in the first quadrant is the shuffle of the Dyck languages on $\{u,d\}$ and $\{l,r\}$.

PROPOSITION 9.1.9. *The number of loops of length $2n$ that stay in the quadrant $(x\geq 0,\ y\geq 0)$ is given by:*

$$\sum_{k=0}^{n}\binom{2n}{2k}C_kC_{n-k}=\frac{1}{(2n+1)(2n+2)}\binom{2n+2}{n+1}^2.$$

The general case of walks with given length and endpoint or with given length is similar to the case of the diagonal quadrant and left to the reader.

A remarkable consequence of these formulas is that the languages of walks in the diagonal (or in the standard quadrant) cannot be an algebraic language: on the one hand the asymptotic number of walks of length n in the diagonal quadrant, $\binom{n}{\lfloor n/2\rfloor}^2$, grows like $4^n/n$ when n goes to infinity; on the other hand, the possible asymptotic behaviors of the Taylor coefficients of an algebraic series are classified, and do not include the form $\rho^n n^{-i}$ for i a positive integer; therefore the generating function of walks in the diagonal quadrant is not algebraic, and the associated language cannot be algebraic either.



(a) A walk on the slitplane. (b) The factorization of a walk.

Figure 9.6. On the slitplane.

9.1.4. Walks on the slitplane and the cycle lemma

We call *slitplane* the complement of the half axis ($x = 0, y \leq 0$) in the square lattice $\mathbb{Z}^2$. Walks on the slitplane are defined as walks that do not touch this half axis except maybe at their startpoint or endpoint, as shown in Figure 9.6(a).

The tool we shall use to enumerate walks on the slitplane is the so-called *cycle lemma*. For any alphabet $\mathcal{A}$ endowed with a morphism $\delta : (\mathcal{A}, \cdot) \to (\mathbb{Z}, +)$, a word w in $\mathcal{A}^*$ is said to have the *Łukasiewicz property* if every strict prefix v of w satisfies $\delta(v) > \delta(w)$.

Lemma 9.1.10 (Cycle lemma). *Let $\mathcal{A}$ be an alphabet endowed with a morphism $\delta : (\mathcal{A}, \cdot) \to (\mathbb{Z}, +)$. Then a word w in $\mathcal{A}^*$ such that $\delta(w) = -1$ admits a unique factorization $w_1 w_2$ with w_1 non empty such that $w_2 w_1$ has the Łukasiewicz property.*

Proof. Let w_1 be the shortest prefix of w with $\delta(w_1)$ equal to the depth of w. Then $w_2 w_1$ has the Łukasiewicz property. Moreover, let us verify that there is no other such factorization. First assume that w'_1 is a prefix of w shorter than w_1. Then the prefix w'' of w'_2 of length $|w_1| - |w'_1|$ satisfies $\delta(w'') < 0$ and is also a strict prefix of $w'_2 w'_1$. Hence $w'_2 w'_1$ has not the Łukasiewicz property. It remains to consider the case of a prefix w'_1 of w longer than w_1. The suffix w'' of w'_1 of length $|w'_1| - |w_1|$ satisfies $\delta(w) \geq 0$ and is also a suffix of $w'_2 w'_1$. Since moreover $\delta(w'_2 w'_1) = -1$, $w'_2 w'_1$ has not the Łukasiewicz property. ∎

Corollary 9.1.11. *Consider the alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, endowed with a morphism δ, and let $n_1, n_2, \ldots, n_k$ be nonnegative integers such that,*

$$\sum_{i=1}^{k} n_i \delta(a_i) = -1.$$

Then the number of words with n_i letters a_i for any $1 \leq i \leq k$ that have the

Łukasiewicz property is equal to:

$$\frac{1}{n_1+\ldots+n_k}\binom{n_1+\ldots+n_k}{n_1,\ldots,n_k}.$$

crossref to chapter 1 ?

Proof. For any word w as above, $\delta(w)=-1$, so that the conjugacy class of w contains $|w|$ different words. According to the cycle lemma exactly one of these $n_1+\cdots+n_k$ words has the Łukasiewicz property. The formula follows. ■

For $\mathcal{A}=\{u,d\}$ with $\delta(u)=1$, $\delta(d)=-1$, the set of words enumerated by the previous corollary is the Dyck-Łukasiewicz language $\mathcal{D}d$, and we recover Proposition 9.1.3.

def of code needed?

Corollary 9.1.12. *Let $\mathcal{C}$ be a code for a set of words on the alphabet $\mathcal{A}$. Then the generating function (with respect to the length) for Łukasiewicz words w in $\mathcal{C}^*$ such that $\delta(w)=-1$ is equal to*

$$[y^{-1}]\log\frac{1}{1-C(t;y)},$$

where $C(t;y)$ is the generating function of the code $\mathcal{C}$ with respect to the length (variable t) and to δ (variable y).

Proof. The generating function of words on the alphabet $\mathcal{A}$ with k factors in $\mathcal{C}$ is $C(t;y)^k$. Restricting the generating function to words w with $\delta(w)=-1$ is done by taking the coefficients of y^{-1} in the series. The fraction of these words that have the Łukasiewicz property is then $1/k$, so that their generating function is

$$\sum_{k\geq 1}\frac{1}{k}[y^{-1}]C(t;y)^k=[y^{-1}]\log\frac{1}{1-C(t;y)}.$$

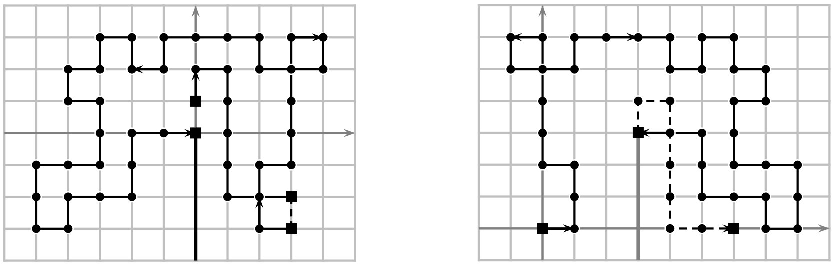
■

To study walks on the slitplane, it is natural to decompose them at points where they touch the vertical axis ($x=0$), as shown in Figure 9.6: any walk w on the plane that finishes on the vertical axis can be uniquely factored into vertical steps on this axis and primitive excursions in the left or right half plane; in other terms, the language of these walks is

$$(u+d+l\mathcal{M}^{(l)}r+r\mathcal{M}^{(r)}l)^*$$

where $\mathcal{M}^{(l)}$ and $\mathcal{M}^{(r)}$ respectively denote the set of excursions in the left half plane ($x<0$) and in the right one ($x>0$). Hence the set $\{u,d\}\cup l\mathcal{M}^{(l)}r\cup r\mathcal{M}^{(r)}l$ forms a code $\mathcal{C}$ for walks on the plane ending on the vertical axis: these walks can thus be viewed as walks on the axis ($x=0$) with the infinite set of steps $\mathcal{C}$.

To apply the cycle lemma to walks on the slitplane, we consider again the morphism $\delta(w)=|w|_u-|w|_d$. Let us single out the class of walks on the slitplane that start at position $(0,1)$ and end on the half axis at position $(0,0)$: these walks are exactly the Łukasiewicz words w in $\mathcal{C}^*$ such that $\delta(w)=-1$.



(a) A walk w on the slitplane. (b) The half plane excursion $\varphi(w)$.

Figure 9.7. On the slitplane.

PROPOSITION 9.1.13. *The number of walks on the slitplane with startpoint* $(0,0)$, *endpoint* $(0,1)$ *and length* $2n+1$ *is:*

$$C_{2n+1} = \frac{1}{2n+2}\binom{4n+2}{2n+1}.$$

Proof. Let $C(t;y)$ be the commutative image of $\mathcal{C}$, so that $1/(1-C(t;y))$ is the generating function of words on the code $\mathcal{C}$. Observe that a $\pi/2$- (respectively $-\pi/2$-) rotation maps bijectively words of length n in $\mathcal{M}^{(l)}$ (resp. $\mathcal{M}^{(r)}$) on words of length n in the bicolored Motzkin language $\mathcal{M}^{(2)}$, hence Proposition 9.1.5 yields:

$$\begin{aligned}\log\frac{1}{1-C(t;y)} &= \frac{1}{2}\left(\log\frac{1}{1-t(y+\frac{1}{y}+2)}+\log\frac{1}{1-t(y+\frac{1}{y}-2)}\right)\\ &= \frac{1}{2}\sum_{n\geq 1}\frac{t^n}{n}\left((y+\tfrac{1}{y}+2)^n+(y+\tfrac{1}{y}-2)^n\right).\end{aligned}$$

The formula follows by extracting the coefficient of y^{-1} and resumming. ■

The above proof does not yield an interpretation of the occurrence of Catalan numbers in Proposition 9.1.13. We conclude this section with a more direct derivation.

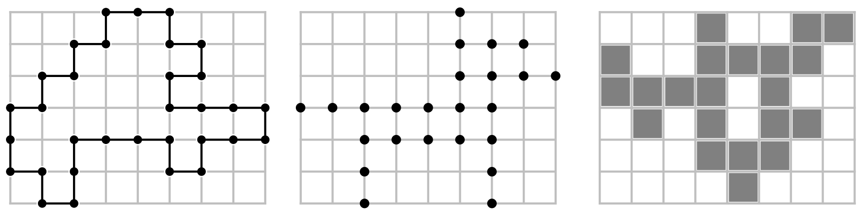
Proof of Proposition 9.1.13 (bis). We are interested in walks w such that

– $|w|_l = |w|_r$, and $|w|_d = |w|_u + 1$,

– and for any strict prefix v of w, either $|v|_l \neq |v|_r$, or $|v|_u \geq |v|_d$.

The first condition accounts for the displacement between the startpoint and endpoint, while the second one ensures that the walks stay in the slitplane. Let us describe a one-to-one correspondence φ between these walks and excursions of even length in the half plane (bicolored Motzkin words). The result then follows from Corollary 9.1.6.

Let w be a walk as in the proposition. Since $|w|_d = |w|_u + 1$, Lemma 9.1.10 yields a unique factorization of w in $w_1 d w_2$ such that each proper prefix v of



(a) A polygon, (b) an animal, (c) and a polyomino.

Figure 9.8. Three related classes of objects.

$w_2 w_1 d$ satisfies $|v|_u \geq |v|_d$: this is the factorization at the first arrival to the lowest level. Let $\bar{w_2}$ be the walk that is symmetric to w_2 with respect to the vertical axis ($x = 0$), and $\varphi(w)$ be equal to $\bar{w_2} w_1$. Then $\varphi(w)$ is a bicolored Motzkin word, corresponding to an excursion in the half plane ($y \geq 0$) of length $2n$. Moreover the factorization $\bar{w_2} w_1$ of $\varphi(w)$ is the factorization at the first passage on the lowest point on the vertical line of equidistance between the startpoint and endpoint of $\varphi(w)$.

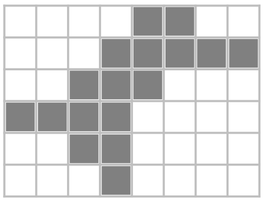
Conversely, given a bicolored Motzkin word w', let $w'_1 w'_2$ be its factorization at the first passage on the lowest point on the vertical line of equidistance between its startpoint and endpoint. Let $\psi(w') = w'_2 d \bar{w}'_1$. The walk $\psi(w')$ is clearly a walk in the slitplane from $(0, 1)$ to $(0, 0)$, and $\varphi(\psi(w')) = w'$. Moreover, $\psi(\varphi(w)) = w$ for any walk w as in the proposition, and this concludes the proof. ■

As discussed in Section 9.1.3, the language of bicolored Motzkin words has a very natural algebraic decomposition. However this decomposition does not carry very well through the bijection.

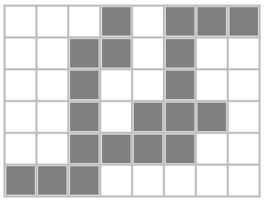
9.2. Sampling: polygons, animals and polyominoes

A walk on the square lattice $\mathbb{Z}^2$ is called a *self-avoiding walk*, or a *path*, if it visits at most once each vertex of the lattice. A *self-avoiding polygon*, or simply in this text, a *polygon*, is a self-avoiding loop.

An *animal* is a set A of vertices of the lattice such that any two vertices of A are connected by a path visiting only vertices of A. Animals are considered up to translations of the lattice. Placing a unit square centered on each vertex of A, we obtain a *polyomino*. The latter are however more naturally defined as edge-connected sets of squares of the lattice. These definitions are illustrated by Figure 9.8. Each polygon is the *contour* (or the boundary) of a simply-connected polyomino, and in the plane this is a one-to-one correspondence (see Figures 9.10, 9.11 and 9.12). In particular the *length* of a polygon corresponds to the *perimeter* of the polyomino. A polygon has moreover *dimension* (p, q) if the smallest rectangle in which it can be inscribed has horizontal width p



(a) A convex polyomino.



(b) A directed polyomino.

Figure 9.9. Subclasses of polyominoes.

and vertical width q. Finally the area of a polyomino is its number of cells, corresponding for animals to the number of vertices.

Little can be said from the enumerative point of view on animals, polygons or polyominoes in general. Two ideas have however been particularly successful for defining subclasses amenable to mathematical study and still of interest: restriction to convex or to directed objects. A polygon of dimension (p, q) is *convex* if its length is $2p+2q$. This definition stresses the fact that convex polygons are in some sense the most extended polygons, and do not make meanders. An equivalent, but maybe more appealing interpretation is in terms of polyominoes: a polyomino is *convex* if its intersection with any horizontal or vertical line is connected. A polyomino (respectively an animal) is *directed* if there is a cell (resp. a vertex) from which every cell (resp. vertex) can be reached by a path going up or right inside the object. These definitions are illustrated by Figure 9.9.

9.2.1. Generalities on sampling

Together with the enumerative questions, much interest has been given to the properties of random animals, polyominoes and polygons. By random is meant here the uniform distribution: objects of equal size are given equal probability to appear. We illustrate this trend by concentrating on the derivation of random generators. In order to describe these algorithms, we assume that we have at our disposal a perfect random number generator $\textsc{Rand}(m, n)$ that outputs an integer of the interval $[m, n]$ chosen with uniform probability: for all $m \leq i \leq n$,

$$\mathbf{P}(\textsc{Rand}(m, n) = i) \;=\; 1/(n - m + 1).$$

We assume unit cost for arithmetic operations and for calls to the generator $\textsc{Rand}()$. These randomness and complexity models are justified by the fact that our algorithms only sample and compute on integers that are polynomially bounded in the size of the objects generated.

We shall need a random sampler for elements of $\mathfrak{S}(w)$, the set of permutations of the letters of a fixed word w. The following algorithm does this by applying a random permutation to the letters of w.

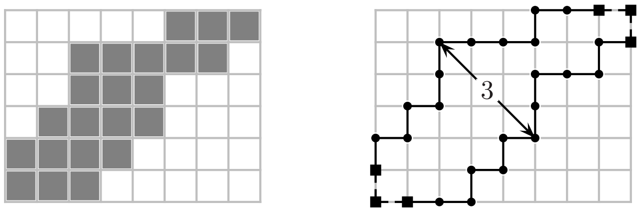


Figure 9.10. A parallelogram polyomino and its contour.

```
RandPerm(w)
for i ← 2 to |w| do
    Swap(w[i], w[Rand(1, i)])
return w
```

Lemma 9.2.1. RandPerm(w) *returns in linear time a random element of* $\mathfrak{S}(w)$ *under the uniform distribution: for all* $w' \in \mathfrak{S}(w)$,

$$\mathbf{P}(\text{RandPerm}(w) = w') \;=\; \frac{1}{\mathrm{Card}(\mathfrak{S}(w))}.$$

Proof. A permutation σ on the set $\{1, \ldots, n\}$ has a unique decomposition as a product $\sigma = \tau_n \ldots \tau_2$ of transpositions of the form $\tau_i = (j_i, i)$ with $1 \leq j_i \leq i$, and conversely any such decomposition provides a permutation. Therefore, the call RandPerm(w) on a word w with distinct letters generates a uniform random permutation of the letters. Upon labelling identical letters by their initial place, we conclude that uniformity is also preserved in the general case. ∎

In the rest of this part, we describe random sampling algorithms for convex polygons and directed animals.

9.2.2. Parallelogram polyominoes and the cycle lemma

A convex polyomino P is a *parallelogram polyomino* if its contour contains the bottom left and top right corners of its bounding box. Equivalently, its contour must be a *staircase polygon*, *i.e.* a polygon made of two up-right directed paths, meeting only at their extremities. These upper and lower paths, being directed, can be coded with two letters. For later purpose, it will be convenient to code them on the alphabet $\{h, v\}$, with h standing for a horizontal step and v standing for a vertical step. Starting from the bottom left corner, let vw_1h be the word coding the upper path, and hw_2v be the word coding the lower path (there is no choice for first and last letters). If P has dimension $(p+1, q+1)$ then $|w_1|_h = |w_2|_h = p$ and $|w_1|_v = |w_2|_v = q$. The *reduced code* of a staircase polygon w is the word on the alphabet $\mathcal{A} = \left\{\binom{v}{h}, \binom{v}{v}, \binom{h}{h}, \binom{h}{v}\right\}$ obtained by stacking the two words w_1 and w_2. In the example of Figure 9.10, the two paths are respectively $vw_1h = v \cdot vhvhvvhhhvhh \cdot h$ and $hw_2v = h \cdot hhvhvhvvhhvh \cdot v$.

Words on $\mathcal{A}$ that code for staircase polygons are characterized by the facts that they have an equal number of letters h in both rows, and that their prefixes contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$: indeed, the morphism δ induced by $\{\delta\binom{v}{h} = 1,\ \delta\binom{h}{v} = -1,\ \delta\binom{h}{h} = \delta\binom{v}{v} = 0\}$ measures the distance between the upper and lower paths along diagonals, and the positive prefix property expresses the condition that the upper and lower paths do not meet before their endpoint. Codes of staircase polygons are thus essentially bicolored Motzkin words.

This characterization suggests to construct staircase polygons by applying the cycle lemma to words of the set $S(p,q)$ of words of length $p+q+1$ on $\mathcal{A}$ with $p+1$ letters h and q letters v in the first row, and p letters h and $q+1$ letters v in the second row:

Staircase(p,q)

$w'_1 \leftarrow$ RandPerm$(h^{p+1}v^q)$ $\quad\triangleright$ generate $w' = \binom{w'_1}{w'_2} \in S(p,q)$
$w'_2 \leftarrow$ RandPerm$(h^p v^{q+1})$
$(m, \delta_m) \leftarrow (0,0)$ $\quad\triangleright$ seek the position m of the
$\delta \leftarrow 0$ $\quad\triangleright$ leftmost minimum w.r.t δ
for $i \leftarrow 1$ **to** $p+q+1$ **do**
    if $(w'_1[i], w'_2[i]) = (v,h)$ **then**
        $\delta \leftarrow \delta + 1$
    elseif $(w'_1[i], w'_2[i]) = (h,v)$ **then**
        $\delta \leftarrow \delta - 1$
        if $\delta < \delta_m$ **then**
            $(m, \delta_m) \leftarrow (i, \delta)$
$(w_1 h, w_2 v) \leftarrow$ Shift$((w'_1, w'_2), m)$ $\quad\triangleright$ get the conjugate at position m
return $(v w_1 h, h w_2 v)$

Proposition 9.2.2. Staircase(p,q) *produces the code of a random uniform staircase polygon with dimension* $(p+1, q+1)$ *in linear time.*

Proof. Let us first use the cycle lemma to derive the number of staircase polygons. The number of words in $S(p,q)$ is $\mathrm{Card}(S(p,q)) = \binom{p+q+1}{q}\binom{p+q+1}{p}$. Then among the $p+q+1$ cyclic shifts of any word $w' \in S(p,q)$, exactly one is of the form $w\binom{h}{v}$ with w having the positive prefix property. Hence the number of staircase polygons with dimension $(p+1,q+1)$ is $\frac{1}{p+q+1}\binom{p+q+1}{q}\binom{p+q+1}{p}$.

The algorithm Staircase() generates a word uniformly at random in the set $S(p,q)$, and computes its unique cyclic shift coding for a staircase polygon. The probability to get the code of a given polygon P is thus the sum of the probability to get each of its cyclic shifts. But the code of P admits $p+q+1$ distinct cyclic shifts, and each of these word has probability $1/\mathrm{Card}(S(p,q))$ to be obtained. Thus the probability to get P is $(p+q+1)/\mathrm{Card}(S(p,q))$, *i.e.* depends only on the dimension of P: uniformity is preserved through the cycle lemma. ∎

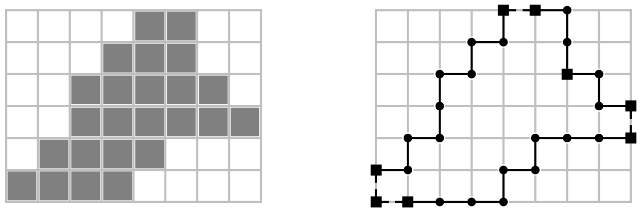


Figure 9.11. A directed convex polyomino and its contour.

9.2.3. Directed convex polyominoes and Catalan's factorization

Directed convex polyominoes are characterized among convex polyominoes by the property that their contour contains the bottom left corner of their bounding box. In other terms contours of directed convex polyominoes are *unimodal polygons*, *i.e.* shuffles of a word of the language u^*d^* and a word of the language r^*l^*. Let us consider an unimodal polygon with dimension $(p+1, q+1)$, and decompose it into an upper path and a lower path both starting from the bottom left corner and of length $p+q+2$, and respectively obtained in clockwise and counterclockwise direction. Let w'_1 and w'_2 be the codes of these two paths on the alphabet $\{h, v\}$. In the example of Figure 9.11, the two paths are respectively $w'_1 = vhvhvvhvhvhhvv$ and $w'_2 = hhhhvhvhhhvhvh$. The following properties of w'_1 are immediate consequences of the definition of unimodal polygons:

1. the word w'_1 starts with a letter v;
2. it contains at least $q+1$ letters v;
3. the first $q+1$ letters v code up steps, the other ones down steps;
4. the $(q+1)$th letter v is followed by a letter h.

The last property accounts for the right turn that the path has to make when reaching the upper boundary. Define the *reduced code* w_1 as obtained from w'_1 by deleting the two redundant letters given by Properties 1 and 4 above. Similarly the reduced code w_2 is obtained by deleting from w'_2 the first letter (that is a letter h) and the letter following the $(p+1)$th letter h (that is a letter v). Let w be the word on $\mathcal{A}$ obtained by stacking w_1 and w_2. Then again all prefixes of w contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$. It turns out that this condition is sufficient for w to code an unimodal polygon: this is expressed by the following lemma, the proof of which is left to the reader.

Lemma 9.2.3. *A word w on $\mathcal{A}$ is the stacked reduced code of an unimodal polygon with dimension $(p+1, q+1)$ if and only if all its prefixes contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$, and, viewed as a pair of words on $\{h, v\}$, it contains $2p$ letters h and $2q$ letters v.*

In terms of the morphism δ of the previous section, Lemma 9.2.3 implies that a word of $\mathcal{A}^*$ is the code of an unimodal polygon if and only if it is a prefix of Motzkin word on $(\mathcal{A}, \delta)$. These prefixes are similar to prefixes of Dyck words with δ even, and the proof of Proposition 9.1.2 suggests the following algorithm.

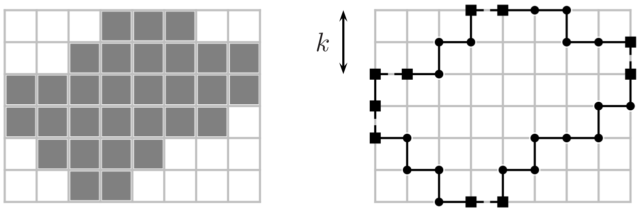


Figure 9.12. A convex polyomino and its contour.

```
Unimodal(p, q)
 1  w1 ← RandPerm(h^p v^q)                  ▷ generate w = (w1 / w2) with δ(w) = 0
 2  w2 ← RandPerm(h^p v^q)
 3  δ ← 0
 4  δm ← 0
 5  for i ← 1 to p + q do
 6      if (w1[i], w2[i]) = (v, h) then
 7          δ ← δ + 1
 8      elseif (w1[i], w2[i]) = (h, v) then
 9          δ ← δ − 1
10          if δ < δm then                        ▷ leftmost minimum found
11              (δm, w1[i], w2[i]) ← (δ, v, h)    ▷ down step to up step
12  return (w1, w2)
```

Proposition 9.2.4. Unimodal(p, q) *produces the reduced code of a random uniform unimodal polygon with dimension* $(p+1, q+1)$ *in linear time.*

Proof. Lines 1, 2 of the algorithm construct a word $\binom{w_1}{w_2}$ satisfying $\delta\binom{w_1}{w_2} = 0$. A straightforward adaptation of the bijection used for Proposition 9.1.2 shows that these words are in one-to-one correspondence with prefixes of Motzkin words: for the current δ, steps $\binom{v}{h}$ play the role of up steps, steps $\binom{h}{v}$ that down steps, and Motzkin factors replace Dyck factors. The algorithm implements the inverse bijection, replacing leftmost down steps at negative levels by up steps.

Since the word $\binom{w_1}{w_2}$ is taken uniformly in the set of words with p letters h and q letters v in both lines, its image is uniform in the set of bicolored Motzkin prefixes with $2p$ letters h and $2q$ letters v. ∎

As a corollary of the previous proof, we also see that the number of unimodal polygons of dimension $(p+1, q+1)$ is $\binom{p+q}{p}^2$.

9.2.4. Convex polyominoes and rejection sampling

The contour of a convex polyomino with dimension $(p+1, q+1)$ can be coded as follows by a pair (w', k): start from the upper point of the contour on the left boundary, and code the path in clockwise direction by a word w' with letters h and v as previously; let moreover k be the distance of the startpoint to the top border of the bounding box (see Figure 9.12). From the geometry, the following properties of the word w' are immediate:

1. there are $2p+2$ letters h and $2q+2$ letters v; moreover $0 \leq k \leq q$;
2. the first $p+1$ letters h code right steps, the other $p+1$ left steps;
3. the first k letters v code up steps, the next $q+1$ down steps, and the final $q+1-k$ up steps again;
4. the first letter is a letter h;
5. if $k > 0$ then the kth letter v is followed by a letter h;
6. the $(p+1)$th letter h is followed by a letter v;
7. the $(k+q+1)$th letter v is followed by a letter h;
8. the $(2p+2)$th letter h is followed by a letter v;
9. the letters singled out in 4, 5, 6, 7, and 8 above appear in this order.

These properties do not completely characterize the codes of convex polygons, but this is almost the case, as the reader will verify:

LEMMA 9.2.5. *A pair (w', k) satisfying the nine properties above is the code of a convex polygon if and only if the corresponding walk is a polygon, that is, if it does not visit twice the same point. This property can be checked in linear time by the following algorithm.*

```
CheckSimple(w′, k)
(i1, δ1, ε1) ← (1, q + 1 − k, +1)         ▷ traversal of w′ from the left
(i2, δ2, ε2) ← (2p + 2q + 3, q − k, −1)   ▷ traversal of w′ from the right
for ℓ ← 1 to p + 1 do                     ▷ ℓ counts horizontal steps
    while w′[i1] = v do                   ▷ vertical move on top
        (i1, δ1) ← (i1 + 1, δ1 + ε1)
    while w′[i2] = v do                   ▷ vertical move on bottom
        (i2, δ2) ← (i2 − 1, δ2 + ε2)
    if δ1 ≤ δ2 then                       ▷ self-intersection detected
        return FALSE
    if δ1 = q + 1 then                    ▷ top reached
        ε1 ← −1
    if δ2 = 0 then                        ▷ bottom reached
        ε2 ← +1
    (i1, i2) ← (i1 + 1, i2 − 1)           ▷ next column
return TRUE
```

The reduced code (w, k) of a convex polygon is obtained by deleting the redundant letters given by Properties 4, 6, 7, 8, and if $k > 0$ by Property 5. The reduced word w has thus, if $k = 0$, $2p$ letters h and $2q$ letters v, or, if $k > 0$, $2p-1$ letter h and $2q$ letters v. Given the reduced word w and the index k there is an immediate algorithm INSERTREDUNDANTLETTERS(w, k) that reconstructs w' by inserting the missing letters from left to right.

The following generator is based on the rejection principle: words of a superset of the set of codes are generated uniformly at random until a proper code is obtained.

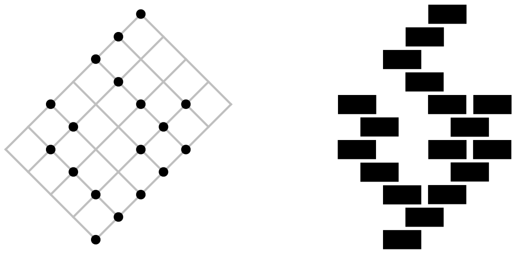


Figure 9.13. A directed animal and the equivalent strict pyramid.

```
Convex(p, q)
do   k ← Rand(0, q)
     w ← RandPerm(h^{2p} v^{2q})
     if k = 0 or w[2p + 2q] = h then
          w' ← InsertRedundantLetters(w, k)
          if CheckSimple(w', k) = true then
               return (w', k)
while true
```

Proposition 9.2.6. Convex(p, q) *produces the code of a random uniform convex polygon with dimension* $(p + 1, q + 1)$.

Proof. The fact that the output is uniform follows from the following standard rejection argument: when the algorithm stops, the probability to output a given code is proportional to the probability to get this code as an element of the superset; but elements of the superset are sampled uniformly, *i.e.* have the same probability to be generated. ■

The expected complexity of the algorithm Convex() depends on the comparison between the size $(q+1)\binom{2p+2q}{2p}$ of the superset $S_{p,q}$ in which k and w are sampled, and the size of the set $P_{p,q}$ of convex polygons with dimension $(p+1, q+1)$. More precisely, each loop takes linear time, the probability of success of a loop is $s_{p,q} = \operatorname{Card}(P_{p,q}) / \operatorname{Card}(S_{p,q})$, and the number of loops is a geometric random variable with expectation $1/s_{p,q}$. The explicit computation of $\operatorname{Card}(P_{p,q})$ shows that this last value is bounded by a constant, but we do not include the details here (see Problem 9.2.2).

Proposition 9.2.7. *The call* Convex(p, q) *has expected linear complexity.*

9.2.5. Directed animals

Upon rotating the lattice counterclockwise by $\pi/4$, directed animals can be given an elegant interpretation in terms of *heaps of bricks*: cells are viewed as bricks exposed to the gravity law with the bottom brick lying on the floor; the condition that animals are directed, *i.e.* that there always exists a path downward to the bottom cell, is equivalent to the fact that every brick leans on one brick below and cannot fall.

To be more precise, let us give a definition of heaps of bricks. The alphabet of bricks is $\mathcal{B} = \{(i, i+1), i \in \mathbb{Z}\}$. Two bricks b, b' of $\mathcal{B}$ commute if and only if, as subsets of $\mathbb{Z}$, $b \cap b' = \emptyset$. Two words are equivalent, $w \equiv w'$, if one can be obtained from the other by a sequence of commutations of adjacent commuting bricks. A heap of bricks is an element of the associated partially commutative monoïd, *i.e.* an equivalence class for the relation $\equiv$. The set of minimal bricks of a heap w is the set $\min(w) = \{b \mid \exists w', w \equiv bw'\}$. A *pyramid* at abscissa i is a heap such that $\min(w) = \{(i, i+1)\}$.

The canonical geometric representation of a heap induced by the gravity law corresponds to the standard Cartier-Foata normal form of the heap: reading a heap from left to right in lines from bottom to top yields a word w of the form $w_1 \cdots w_k$ with each block w_i made of commuting letters and such that for each letter b of w_{i+1} there is a letter b' of w_i with $b \cap b' \neq \emptyset$. A heap is *strict* if moreover no two consecutive blocks of the normal form have a brick in common: in other terms in a strict heap a brick $(i, i+1)$ always lean on a brick $(i-1, i)$ or $(i+1, i+2)$, not on another brick $(i, i+1)$.

From the geometric interpretation of pyramids of bricks and the initial discussion of this paragraph, the following lemma is immediate.

LEMMA 9.2.8. *Directed animals "are" strict pyramids of bricks.*

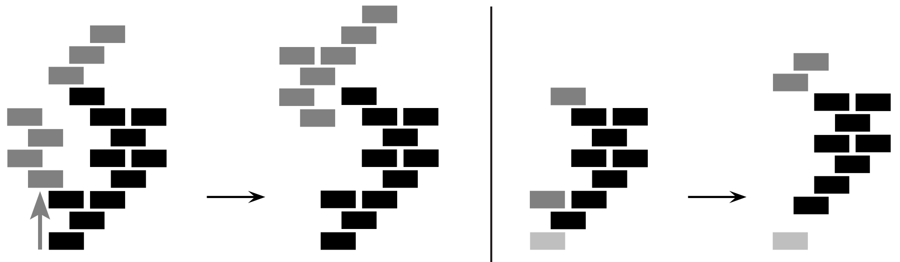
This interpretation of directed animals in terms of pyramids of bricks allows to perform decompositions that would otherwise be very difficult to explain. First define a *semi-pyramid* to be a pyramid without bricks on the left hand side of the bottom brick. Then the following two decompositions are obtained by pushing upward a brick and all the bricks that lay above it, or indirectly lean on it:

- a strict pyramid of bricks is either a strict semi-pyramid, or can be factored, by pushing upward the lowest brick with abscissa -1, into a strict pyramid at abscissa -1 stacked over a strict semi-pyramid;
- a strict semi-pyramid is reduced to a brick, or to a strict semi-pyramid at abscissa 1 over a brick, or can be factored, by pushing upward the second lowest brick with abscissa 0, into a strict semi-pyramid at abscissa 0 stacked over a strict semi-pyramid at abscissa 1 over a brick.

This joint decomposition is isomorphic to the joint decomposition of prefixes of words and of words of the Motzkin language on the alphabet $\{a, b, x_1\}$:

- a prefix of Motzkin word is either a Motzkin word or can be decomposed as uav with u a Motzkin word and v a prefix of Motzkin word.
- a Motzkin word is reduced to the empty word ε, or is of the form $x_1 u$ with u a Motzkin word, or can be decomposed as $aubv$ with u and v two Motzkin words.

These isomorphic decompositions induce a bijection between strict pyramids of n bricks and prefixes of Motzkin words of length $n-1$.



(a) Factorization of a pyramid into a pyramid and a semi-pyramid.

(b) Factorization of a semi-pyramid into a brick and two semi-pyramids.

Figure 9.14. Decomposition of pyramids.

COROLLARY 9.2.9. *Prefixes of Motzkin words can be bijectively transformed into strict pyramids of bricks in linear time.*

The Motzkin language being algebraic, uniform random generation could be done using a recursive approach. We describe instead another application of the rejection principle which is both more elegant and more efficient for this specific problem. Let us consider again the alphabet $\mathcal{A}_k = \{u, d, x_1, \ldots, x_k\}$ and the associated k-colored Motzkin words of Section 9.1.3. A naive algorithm to generate uniform random prefixes of k-colored Motzkin words of length n consists in generating uniform random words of $(\mathcal{A}_k)^n$ and rejecting. However a simple calculation shows that the probability of success is of order $O(n^{-1/2})$ thus giving an algorithm with expected complexity $O(n^{3/2})$. A slight refinement on this idea is to observe that rejection can be decided on the fly. This turns out to be surprisingly efficient.

```
FlorentineRejection(n, k)
do  w ← ε
    for i ← 1 to n do              ▷ generate from left to right
        w[i] ← Rand(1, k + 2)
        if w[i] = k + 1 then
            δ ← δ + 1
            w[i] ← u
        elseif w[i] = k + 2 then
            δ ← δ − 1
            w[i] ← d
            if δ < 0 then          ▷ if a negative prefix is detected
                break              ▷ restart from scratch
while i ≠ n + 1                    ▷ until w is a valid n letters word
return w
```

This algorithm obviously produces a prefix of k-colored Motzkin word.

LEMMA 9.2.10. *The function* FLORENTINEREJECTION(n,k) *generates a random uniform prefix of k-Motzkin word of length n in expected linear time.*

Proof. For simplicity the analysis is presented in the case $k=0$ but the same strategy of analysis applies to the general case (using generating functions instead of elementary counting). It will be convenient to consider that when the construction fails at the ith step of the inner loop, we finish the loop and generate $n-i$ more letters at no cost. This modification of the algorithm do not affect the final result or the cost, but allow us to think at each iteration as producing a uniform random word of $(\mathcal{A}_k)^n$. From this point of view, the Florentine rejection behaves like standard rejection and therefore it is uniform on prefixes.

The probability of success of the inner loop is $p_n = \binom{2n}{n}2^{-2n} = p_n$, and the number of aborted loops is a geometric random variable with expected value $1/p_n = O(n^{1/2})$. Let us now compute the expected cost of a failure: a failure with cost $2i+1$ is obtained for a word w of the form ubv with u a Dyck word of length $2i$ and v in $\{a,b\}^{2n-2i-1}$. Hence the cumulated cost for all these $2^{2n}-\binom{2n}{n}$ words is $\sum_{i=0}^{n-1}(2i+1)C_i 2^{2n-2i-1} = 2^{2n-1}\sum_{i=0}^{n-1}\binom{2i+1}{i}2^{-2i} = O(2^{2n}n^{1/2})$. With $O(n^{1/2})$ aborted loops with cost $O(n^{1/2})$ each, and one successful loop with cost n, the total expected cost is linear as announced. ∎

Florentine rejection thus uses on average a linear number of random bits. As opposed to this a call to RANDPERM(w) for a word w of length n uses about $n\log n$ bits, and this is in general suboptimal from a theoretical point of view. For instance for $w = a^n b^n$, $\log\binom{2n}{n} \sim 2n$ bits should suffice. In this case an optimal solution (on average) is obtained using FLORENTINEREJECTION$(n,0)$ to get a prefix of Dyck words and Catalan's factorization (Proposition 9.1.2) to transform it into a word of $\mathfrak{S}(a^n b^n)$. As opposed to this, it is an open problem in general to sample in linear time from $\mathfrak{S}(w)$ using $O(\log \mathrm{Card}(\mathfrak{S}(w)))$ random bits.

9.3. Coding: trees and maps

A *planar map*[1] is a proper embedding of a connected graph in the plane. Multiple edges and loops are allowed, and proper means that edges are smooth simple arcs which meet only at their endpoints. The faces of a planar map are the connected components of the complement of the graph in the plane: apart from one *infinite face*, all faces are bounded and homeomorphic to disks. All the planar maps we consider are *rooted*: they have an oriented edge, called the *root*, which is incident to the infinite face on its right-hand side. Examples of rooted maps are presented in Figure 9.15.

From now on we shall consider that two planar maps are the same if one can be mapped onto the other (including roots) by an homeomorphism of the plane. However there are still many more planar maps than planar graphs, as illustrated by Figure 9.15. Indeed homeomorphisms of the plane respect the

[1] The word *map* is intended here in its geographic sense, like in road-map.

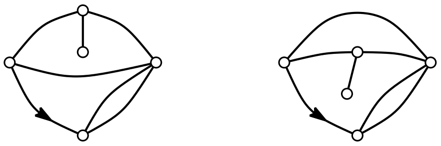


Figure 9.15. Two rooted planar maps with the same underlying graph.

neighborhood of each vertex, so that the circular order of edges around vertices is fixed.

From a combinatorial point of view, a planar map can in fact entirely be specified as follows: label half-edges (or *darts*) and for each half-edge give the names of the opposite half-edge, and of the next half-edge around its origin in counterclockwise direction. As a consequence the number of planar maps with n edges is finite. Moreover these labeled maps capture exactly the level at which algorithms on maps are implemented in computational geometry, using darts as elementary data structures. Carrying on with labeled maps, one could also reach a purely combinatorial setting and eliminate the geometry (at least at the formal level of proofs). However for the sake of conciseness it appears more efficient to keep higher level geometric arguments.

Examples of specific families of planar maps are numerous. A *triangulation of a k-gon* is a planar map without multiple edges such that all bounded faces have degree 3 and the infinite face has degree k (the degree of a face is the number of sides of edges to which it is incident). A *k-valent map* is a planar map such that all vertices have degree k (the degree of a vertex is the number of half-edges to which it is incident).

9.3.1. Plane trees and generalities on coding

A *rooted plane tree*, or hereafter simply a *plane tree* is a planar map with one face. A *planted plane tree* is a plane tree such that the root vertex has degree 1. A *binary tree* is a planted plane tree with vertices of degree 3 and 1 only, respectively called *nodes* and *leaves*. These definitions agree with classical recursive definitions of plane trees: for instance a plane tree can be decomposed as an ordered sequence of subtrees attached to the root.

The *contour traversal* of a planar map is the walk on the vertices and edges of the map that starts from (the right-hand side of) the root edge, and turns around the map in counterclockwise direction so as to visit the boundary of the infinite face. (The reader is encouraged to imagine an ant walking around the map.) The contour traversal of a plane tree visits in particular twice every edge: the first time away from the root vertex, and the second time toward the root vertex. The preorder on the vertices of a planted plane tree is defined by ordering vertices according to the first passage of the contour traversal.

The *Dyck code* of a planted plane tree with $n+1$ edges is the word of length $2n$ on the alphabet $\{u, d\}$ obtained during a contour traversal of the tree by

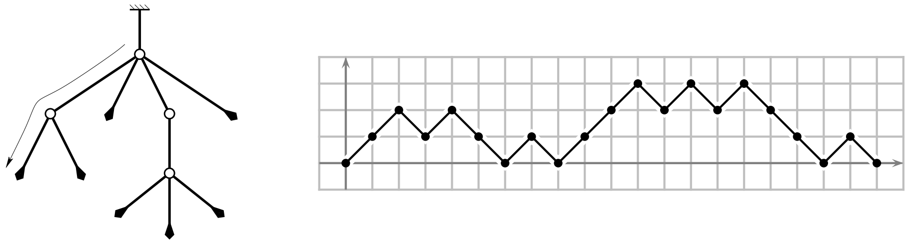


Figure 9.16. A planted plane tree and its Dyck code.

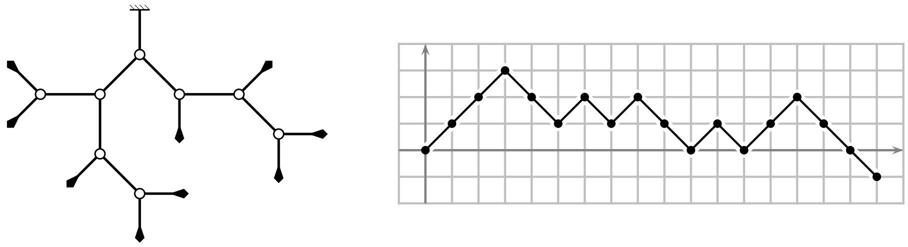


Figure 9.17. A planted binary tree and its prefix code.

writing a letter u each time a non-root edge is visited for the first time (away from the root vertex), and a letter d each time a non-root edge is visited for the second time (toward the root vertex). The reader should convince himself that the Dyck code of a tree characterizes it.

Lemma 9.3.1. *Dyck encoding is a bijection between planted plane trees with $n+1$ edges and Dyck words of length $2n$. In particular the number of planted plane trees with $n+1$ edges is the nth Catalan number.*

The *prefix* or *Łukasiewicz code* of a planted plane tree with n edges is the word of length n on the alphabet $\{x_i, i \geq 0\}$ obtained during a contour traversal of the tree by writing a letter x_i each time a non-root vertex with degree $i+1$ is visited for the first time. Let us define the morphism δ by $\delta(x_i) = i - 1$. Then the prefix code w of a planted plane tree has the Łukasiewicz property (*i.e.* for each strict prefix v of w, $\delta(v) > \delta(w)$). In particular, upon setting $x_2 = u$ and $x_0 = d$, we obtain the following lemma for the case of binary trees:

Lemma 9.3.2. *Prefix encoding is a bijection between binary trees with n nodes, (and thus $n+2$ leaves and $2n+1$ edges) and words of length $2n+1$ of the Dyck-Łukasiewicz language $\mathcal{D}d$. In particular the number of binary trees with n nodes is the nth Catalan number.*

Recall that the optimal coding problem for a family $\mathcal{C}$ of combinatorial structures consists in finding a function φ that maps injectively objects of $\mathcal{C}$ on words of $\{0,1\}^*$ in such a way that an object O of size n is coded by a word $\varphi(O)$ of length roughly bounded by $\log_2 \mathrm{Card}(\mathcal{C}_n)$, with $\mathcal{C}_n$ the set of objects of size n. Since the nth Catalan number satisfies $\log C_n \sim 2n$ as n goes to infinity, Dyck codes and prefix codes respectively solve the optimal coding problem for plane trees and for binary trees. On the other hand, the Dyck code of a binary tree with n nodes has length $4n+2$, so that Dyck codes are far from optimality with respect to the family of binary trees: the optimality of a code is relative to the entropy $\log \mathcal{C}_n$ of the set $\mathcal{C}_n$ under consideration.

More generally, consider the set of planted plane trees with d_i nodes of degree i (and thus $\ell = 1+\sum(i-2)d_i$ non-root leaves). Prefix encoding defines a bijection between this set of trees and the subset of words of $S(x_0^\ell x_1^{d_1} \ldots x_k^{d_k})$ that have the Łukasiewicz property. But according to the cycle lemma, the fraction of such words of length n among words of same length in $S(x_0^\ell x_1^{d_1} \ldots x_k^{d_k})$ is $1/n$. Now words on a finite alphabet with fixed proportion of letters can be encoded optimally by the so-called entropy coder. Hence prefix encoding combined with entropy encoding yields optimal coding for plane trees with a fixed proportion of nodes of each degree.

9.3.2. Conjugacy classes of trees

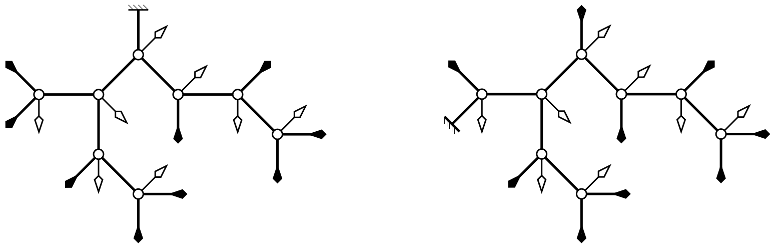
From now on, we consider planted plane trees with two types of vertices of degree 1, respectively called *buds* and *leaves*. Vertices of higher degree are called *nodes*. In particular, a *blossoming tree* is a planted plane tree such that each node has degree 4 and is adjacent to exactly one bud; a blossoming tree with n nodes has thus $n+2$ leaves and n buds. Examples of blossoming trees are given in Figure 9.18.

LEMMA 9.3.3. *The number of blossoming trees that are planted on a leaf and have n nodes is $\frac{3^n}{n+1}\binom{2n}{n}$. The number of blossoming trees that are planted on a bud and have n nodes is $\frac{3^n}{n+2}\binom{2n}{n-1}$.*

Proof. Let $\mathcal{B}'_n$ and $\mathcal{B}''_n$ denote these two sets of blossoming trees. A blossoming tree of the first type can be uniquely obtained from a binary tree with n nodes by attaching a bud to each node in one of the three possible ways. Together with Lemma 9.3.2, this proves the first formula.

Now let us consider the set of doubly planted blossoming trees, one root being a leaf and the second one a bud. Such a tree with n nodes can be considered either as a blossoming tree in $\mathcal{B}'_n$ with a marked bud, or as a blossoming tree in $\mathcal{B}''_n$ with a marked leaf. Hence doubly planted blossoming trees with n nodes are either counted by $n\,\mathrm{Card}(\mathcal{B}'_n)$ or by $(n+2)\,\mathrm{Card}(\mathcal{B}''_n)$. As a consequence, $\mathrm{Card}(\mathcal{B}''_n) = \frac{n}{n+2}\cdot\frac{3^n}{n+1}\binom{2n}{n}$, which proves the second formula. ∎

Let T be a planted plane tree with n nodes. During a contour traversal of T, its buds and leaves are visited in a sequence (by convention the root vertex is visited at the end of the traversal). Accordingly the *border word* is the word with letters $\{b,\ell\}$ obtained along the contour traversal by writing a letter b each time a bud is visited and a letter ℓ each time a leaf is visited. For example, the border



(a) A blossoming tree, (b) and a balanced one.

Figure 9.18. Two conjugate blossoming trees.

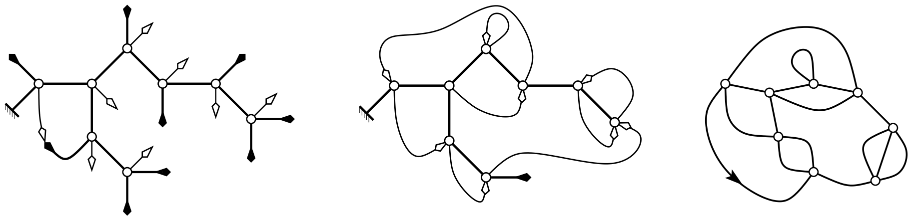
words of the blossoming trees of Figure 9.18 are respectively $\ell\ell b\ell b\ell\ell bb\ell b\ell\ell b\ell bb\ell$ and $b\ell b\ell\ell bb\ell b\ell\ell b\ell bb\ell\ell\ell$.

Two planted plane trees T and T' are *conjugate* if one is obtained from the other by re-rooting. In other terms, two planted plane trees are in the same *conjugacy class of trees* if they share the same underlying unrooted plane tree. This terminology is motivated by the remark that conjugate planted plane trees have conjugate border words. Taking $\delta(b) = +1$ and $\delta(\ell) = -1$, the cycle lemma suggests the following definition: a planted plane tree is *balanced* if its border word has the Łukasiewicz property. With this definition, and remembering that blossoming trees have two more leaves than buds, the cycle lemma for those trees reads: a blossoming tree has exactly two canonical leaves such that the conjugate trees rooted at these leaves are balanced.

Lemma 9.3.4. *There are $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$ balanced blossoming trees with n nodes.*

Proof. The first proof is again based on a double counting argument. Let $\mathcal{B}_n^*$ be the set of balanced blossoming trees with n nodes. The number of balanced blossoming trees with a secondary root leaf is $(n+2)\,\mathrm{Card}(\mathcal{B}_n^*)$. Upon exchanging the role of the two roots, these trees are also blossoming trees with a secondary root leaf taken among the two canonical leaves: their number is thus $2\cdot\frac{3^n}{n+1}\binom{2n}{n}$. The result follows. ∎

Proof (bis). An alternative proof is based on the following remark: the number of balanced re-rootings of any blossoming tree is equal to the difference between its numbers of leaves and buds, so that, in each conjugacy class of trees, the number of balanced trees is exactly the difference between the number of trees rooted on a leaf and the number of trees rooted on a bud. Hence the number of balanced blossoming trees with n nodes is the difference $\frac{3^n}{n+1}\binom{2n}{n} - \frac{3^n}{n+2}\binom{2n}{n-1}$. ∎



(a) A fusion, (b) the partial closure (c) and the complete closure.

Figure 9.19. The closure of the balanced blossoming tree of Figure 9.18(b).

9.3.3. The closure of a plane tree

The *closure* of a planted plane tree with two more leaves than buds is obtained by repeating the following construction until only two leaves remain: perform a contour traversal, and each time a leaf follows a bud in the sequence of vertices of degree 1 met by the walk, match them, *i.e.* fuse the two corresponding dangling edges in the unique way that creates a bounded face with no vertex of degree 1 inside (see Figure 9.19(a)).

LEMMA 9.3.5. *The closure of a plane tree with n nodes and two more leaves than buds terminates and produces a planar map with the same n nodes and two leaves, which are both incident to the infinite face. In particular the closure of a blossoming tree has n vertices of degree four, plus two of degree one in the infinite face.*

If moreover the tree is balanced, then its root vertex is one of the two remaining leaves.

Proof. At each iteration all factors $b\ell$ of the border word are detected, and deleted since the corresponding pairs of bud and leaf are matched. In particular at least one pair is matched at each iteration, so that the construction terminates. Vertices of degree at least two remain unchanged while all buds and leaves are eliminated but the two canonical roots. ∎

As described above the closure could require a quadratic number of operations. The following algorithm takes a planted plane tree with two more leaves than buds and computes its closure in linear time. It uses the following items:

- a local stack with functions PUTINSTACK(), POPFROMSTACK() and ISSTACKEMPTY(),
- a function NEXTFREEVERTEX(*vertex*) that starts a contour traversal after the vertex of degree 1 *vertex* and returns the first vertex of degree 1 found,
- a function TYPE(*vertex*) that tells whether *vertex* is a bud or a leaf,
- a function FUSEINTOEDGE(*bud*, *leaf*) that realizes the fusion of a bud *bud* and a leaf *leaf* into an edge.

```
Closure(T)
 1  n ← NumberOfLeaves(T)
 2  vertex ← RootOf(T)
 3  (ℓ1, ℓ2) ← (vertex, vertex)
 4  while n > 2 do
 5      vertex ← NextFreeVertex(vertex)
 6      if Type(vertex) = bud then
 7          PutInStack(vertex)
 8      elseif IsStackEmpty() then
 9          (ℓ1, ℓ2) ← (ℓ2, vertex)
10      else bud ← PopFromStack()
11          FuseIntoEdge(bud, vertex)
12          n ← n − 1
13  if ℓ1 = ℓ2 then
14      ℓ2 ← NextFreeVertex(vertex)
15  return (T, ℓ1, ℓ2)
```

Remark 9.3.6. Lines 13 and 14 only treat the special case of a balanced blossoming tree in which the second free leaf is the last one of the border word.

The *complete closure* of a balanced blossoming tree is obtained from its closure by fusing the two remaining vertices of degree 1 and the incident dangling edges into a root edge. Lemma 9.3.5 implies that the complete closure of a blossoming tree with n nodes is a 4-valent map with n vertices. The following more precise theorem will be proved in the next section.

Theorem 9.3.7. *The complete closure is one-to-one between balanced blossoming trees with n nodes and 4-valent maps with n vertices. In particular the number of these maps is $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$.*

As a corollary we already have the complete description of a random sampling algorithm for 4-valent maps with n vertices. Apart from the function Closure(), it uses the random generator FlorentineRejection() defined in Section 9.2 and the following items:

- a function PrefixDecode(w) that constructs the binary tree encoded by a Dyck-Łukasiewicz word w,
- a function AddBud(n, i) that adds a bud to a node n in one of the three possible manners,
- a function AddRoot(M, ℓ_1, ℓ_2) that roots the map M by fusing its two leaves ℓ_1 and ℓ_2 into an oriented edge.

```
RandMap(n)
w ← FlorentineRejection(n, 0)
T ← PrefixDecode(wd)
for node ∈ T do
    AddBud(node, Rand(1, 3))
(M, ℓ1, ℓ2) ← Closure(T)
if Rand(1, 2) = 1 then
    AddRoot(M, ℓ1, ℓ2)
else AddRoot(M, ℓ2, ℓ1)
return M
```

Corollary 9.3.8. Randmap(n) *outputs a uniform random 4-valent map with n vertices in linear time.*

9.3.4. The opening of a 4-valent map

The *dual* of a planar map M is the planar map M^* defined as follows: in each face of M put a vertex, and join these new vertices by edges dual to the edges of M. By construction the vertices, edges and faces of M^* are respectively in bijection with faces, edges and vertices of M. This construction is illustrated by Figure 9.20(a). The proof of the following property of duality in planar maps is left to the reader.

Lemma 9.3.9. *Let (E_1, E_2) be a partition of the set of edges of a planar map M. Then E_1 is a spanning tree of M if and only if E_2^* is a spanning tree of M^*. When this case we call (E_1, E_2)* a spanning tree decomposition *of M.*

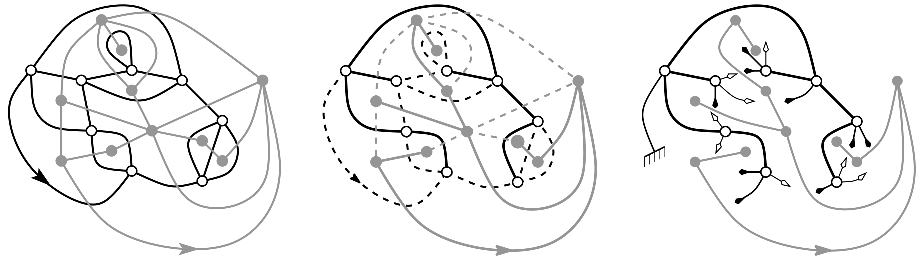
From now on, let M be a planar map, and (E_1, E_2) be a spanning tree decomposition of M. For e an edge of E_2, *opening e with respect to* (E_1, E_2) will mean: orienting e so that the cycle it induces with the tree E_1 is counter-clockwise, and then replacing e by two dangling edges, the one attached to the origin of e holding a bud $b(e)$, the other one holding a leaf $\ell(e)$. We shall always assume moreover that the root r of M belongs to E_2. Then, the *opening of M with respect to* (E_1, E_2) is the tree T defined as follows: (see Figure 9.20(c))

– open each edge $e \in E_2$ with respect to (E_1, E_2),
– replace the bud $b(r)$ by a leaf and plant the tree on it.

The tree T thus consists of the edges of the spanning tree E_1 together with pairs of dangling edges associated to edges of E_2. More precisely, these edges contribute to one bud and one leaf except for the root which contributes to two leaves. By construction, the opening T of a 4-valent planar map M with n vertices has n nodes of degree 4, n buds and $n + 2$ leaves.

Lemma 9.3.10. *The complete closure of the opening of a planar map M with respect to any spanning tree decomposition is the planar map M itself.*

Proof. The opening of an edge merges the two faces incident to it. Since E_2^* forms a spanning tree of M^*, the openings can be performed sequentially so that



(a) A map and its dual, (b) a spanning tree decomposition, (c) and the corresponding opening.

Figure 9.20. An opening of the map of Figure 9.19(c).

one of the two merged faces is always the infinite face. It is then immediate at each step that the pair of bud and leaf created by the opening of an edge corresponds to a matched pair in the closure. ∎

There are in general many spanning tree decompositions of M, and the right one must be chosen to invert the closure. To explain how this is done we need to introduce the distance in the dual map M^*: two faces of M are adjacent if they share a common edge, and the distance between two faces f and f' is the length k of the shortest path $(f_0, \ldots, f_k)$ where $f_0 = f$, $f_k = f'$ and for all i, the two faces f_i and f_{i-1} are adjacent. Observe that the dual M^* of a 4-valent map has only faces with even degrees (in fact degree 4), so that it does not contain any cycle of odd length, and the distances of a face f to two adjacent faces f' and f'' always differ by 1.

To each face f of M, associate the face $r(f)$ incident to the root edge r and closest to f for the distance in M^*. The set $\mathcal{P}(f)$ of paths of minimal length from $r(f)$ to f forms a bundle of paths bounded by two paths $P_0(f)$ and $P_1(f)$, with $P_0(f)$ having the bundle on its right hand side. We shall call $P_0(f)$ the *leftmost minimal path from the root to* f. The union of r^* and of the edges of the paths $P_0(f)$ for all faces f of M forms a spanning tree of M^*: the existence of a cycle would prevent one of the paths from being leftmost. This tree is called the *leftmost breadth first search tree of* M^* *starting from* r^*, because it is also given by a breadth first search traversal with the left hand rule. As stated in the following proposition, it is the spanning tree we are looking for.

Proposition 9.3.11. *Let M be a 4-valent map with root edge r and (E_1, E_2) be a spanning tree decomposition such that $r \in E_2$. Then the opening of M with respect to (E_1, E_2) is a blossoming tree if and only if E_2^* is the leftmost breadth first search tree of M^* starting from r^*.*

The proof of this proposition is based on two lemmas. The first one is a characterization of blossoming trees.

LEMMA 9.3.12. *A tree T with n buds, $n+2$ leaves and n nodes of degree 4 is a blossoming tree if and only if, for every inner edge e, the two components of $T \setminus e$ both contain one more leave than buds.*

Proof. The characterization is trivial for $n = 1$, and remains true when a further node with two leaves and a bud is attached in place of a leaf. The lemma thus follows by induction since every tree can be obtained by adding new nodes incrementally. ∎

For the second lemma it is useful to view the spanning tree E_2^* as rooted on r^*, with the convention that the infinite face of M is the origin of the root.

LEMMA 9.3.13. *Let e be an edge of E_1 separating two faces f, f', with f before f' in the leftmost depth first order on the tree E_2^*. Consider the paths P and P' from f and f' to their common ancestor in E_2^*, which define with e^* a cycle separating a bounded region B of the plane from an unbounded one U. Then,*

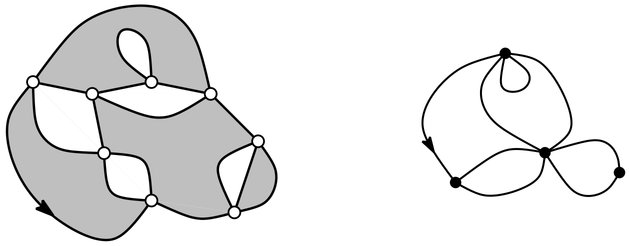
– *the opening of an edge of P with respect to (E_1, E_2) creates a leaf in B and a bud in U,*
– *and the opening of an edge of P' with respect to (E_1, E_2) creates a bud in B and a leaf in U.*

Proof. The result is immediate upon comparing the orientation used in the definition of the opening of an edge and the orientation of the cycle going from e^* up the path P and down the path P'. ∎

Proof of Proposition 9.3.11. First assume that E_2^* is the leftmost breadth first search tree of M^* starting from r^*, and let T be the opening of M with respect to E_2^*. According to Lemma 9.3.12, it suffices to check that for any edge e of E_1, both components of $T \setminus e$ contain one more leaves than buds. Let us consider the paths P and P' of Lemma 9.3.13. The breadth first search condition on E_2^* implies that the length of these two paths differ at most by 1, hence exactly by 1, in view of the discussion of distances in M^*. The leftmost condition on E_2^* moreover implies that the shortest path of the two must be P'. Finally observe that two components of $T \setminus e$ are separated by the dual cycle of Lemma 9.3.13, so that this lemma can be used to count buds and leaves in the two regions. This can be done easily upon distinguishing whether r^* is on P or not.

Let now E_2^* be a spanning tree of M^* different from the leftmost breadth first search tree $E_2'^*$. Then there are leftmost minimal paths that do not appear in E_2^*. Among the shortest of them let $P_0(f)$ be the leftmost one, connecting the root to a face f. Since $P_0(f)$ is minimal, all its edges but the last one e belong to E_2^*. Moreover, by definition of $P_0(f)$, this path is to the left and no longer than the path $P(f)$ connecting the root to f in E_2^*. Applying Lemma 9.3.13 to e, $P \subset P_0(f)$ and $P' \subset P(f)$ and comparing the length of these two paths shows that P' is longer than P, so that the two components of $T \setminus e$ have not the expected number of buds and leaves. ∎

The opening of M with respect to (E_1, E_2) with E_2^* the leftmost breadth first search tree of M^* at r^* will be called simply the *opening of M*. In view



(a) Bicoloration of faces. (b) Corresponding map.

Figure 9.21. Inverse of the edge-map construction.

of Lemma 9.3.10, Proposition 9.3.11 completes the proof of Theorem 9.3.7: the opening is the inverse of the closure. Moreover it induces a linear time algorithm Opening(M) that recovers the unique balanced blossoming tree T such that Closure(T)$=M$:

Opening(M)
1 Perform a leftmost bfs traversal of the dual map M^* starting from r^*.
2 Open the edges of the resulting tree to create buds and leaves.
3 Return the resulting balanced blossom tree.

9.3.5. A code for planar maps

Theorem 9.3.7 deals with a specific family of planar maps, namely 4-valent ones. It turns out however that 4-valent maps play for planar maps the role that edge-graph play for graphs. More precisely, define the *edge-map* of a planar map M as the 4-valent map M^φ having as vertex set the set of edges of M and having an edge c^φ for each corner c of the map M.

Proposition 9.3.14. *The edge-map construction is a bijection between planar maps with n edges and 4-valent maps with n vertices. In particular the number of planar maps with n edges is* $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$.

Proof. The inverse construction follows from the remark that the faces of a 4-valent map F can be colored in two colors, black and white, so that adjacent faces have different colors. The planar map M is obtained by putting a vertex into each black face of F and joining these vertices by an edge across each vertex of F. ∎

The edge-map construction thus allows us to deduce from Theorem 9.3.7 a code for the family of planar map.

ENCODEMAP(M)
$F \leftarrow$ EDGEMAP(M)
$T \leftarrow$ OPENING(F)
for *node* $\in T$ **do**
    $w'[node] \leftarrow$ POSITIONOFBUD(*node*)
    $T \leftarrow$ REMOVEBUD(T)
$w \leftarrow$ BINARYCODE(T)
return (w, w')

THEOREM 9.3.15. *The algorithm* ENCODEMAP() *encodes a planar map with n edges by a pair of words respectively in $\{a,b\}^{2n}$ and $\{0,1,2\}^n$. In view of the number of planar maps, this code is optimal.*

Problems

Section 9.1

9.1.1 Show that the generating function of a rational language with respect to the length is rational.

9.1.2 Compute the generating function with respect to the length of walks that never immediately undo a step they have just done.

9.1.3 Define the *area under a Dyck word* as the number of integer points between the horizontal axis and the associated walk. Use Catalan's factorization to show that the sum of the area under all Dyck words of length $2n$ is 4^n.

(Chottin and Cori 1982)

9.1.4 Show that an algebraic language that can be generated by a non ambiguous context free grammar has an algebraic generating function with respect to the length.

9.1.5 Give a bijective proof of the fact that the number of bicolored Motzkin words of length n is equal to the number of Dyck words of length $2n+2$.

9.1.6 Give a bijective proof of the right hand side formula in Proposition 9.1.9 for the number of loops of length $2n$ that stay in the quadrant $(x \geq 0, y \geq 0)$.

(Guy et al. 1992)

Section 9.2

9.2.1 What is the number of staircase and unimodal polygons with semi-perimeter n?

*9.2.2 Show bijectively that the number of convex polyominoes with bounding box (p,q) is

$$\binom{2p+2q}{2p} + q\binom{2p+2q-1}{2p-1} - 2(p+q)\binom{p+q-1}{q}\binom{p+q-1}{p}.$$

What is the number of convex polyominoes with semi-perimeter n?
(Bousquet-Mélou and Guttmann 1997,Gessel 2000)

9.2.3 An animal on the square lattice has compact source if there exists k such that every vertex of the animal can be reached from one of the vertices $(i, k - i)$ with $0 \leq i \leq k$ by a path going north or east inside the animal. In particular directed animals are exactly the animals with compact source for $k = 0$.
Prove that there are 3^{n-1} animals of size n with compact source.
(Gouyou-Beauchamps and Viennot 1988)

*9.2.4 Give a bijection between bilateral Dyck paths of length n and (non necessarily strict) pyramids of n bricks such that the number of pairs of steps connecting levels i and $i + 1$ is mapped onto the number of bricks in position $(i, i + 1)$.
(Viennot 1986)

**9.2.5 Give a uniform random sampling algorithm of expected linear complexity for the set of words of length n on an arbitrary fixed finite alphabet that have the Łukasiewicz property.

Section 9.3

9.3.1 Give a direct bijection between plane trees with n edges and binary trees with n nodes.

*9.3.2 What is the number of rooted planar maps with d_i vertices of degree $2i$ for all $i \geq 0$ and no odd degree vertex?
(Schaeffer 1997)

**9.3.3 Compute the generating function of rooted planar maps according to the distribution of degrees.
(Bouttier et al. 2002)

**9.3.4 Show that planted plane trees with two leaves per inner vertices are in one-to-one correspondence with rooted triangulations with a marked face.
(Poulalhon and Schaeffer 2003)

Notes

Although this chapter can be read independently, it is intended as a companion to Chapter 11, Words and trees, in Lothaire 1997. Systematic approaches to enumeration, in particular using generating functions, are described in the books Goulden and Jackson 1983, Bergeron et al. 1998 and in the more recent Stanley 1999, Flajolet and Sedgewick 2002. In particular the relevance of rational, algebraic and D-finite series to enumeration is emphasized in the last two ones.

The enumeration of walks in the plane, in the half plane and in the quarter plane has become part of the combinatorial folklore, as well as Dyck walks and Catalan's factorization. The cycle lemma is attributed in the combinatorial literature to Dvoretzky and Motzkin 1947, where it is used to derive Proposition 9.1.4. As first shown by Raney 1960 (see also Chapter 11 of Lothaire 1997), the cycle lemma is a combinatorial version of the Lagrange inversion formula, which has numerous applications in enumerative combinatorics. More detailed historical accounts can be found in Pitman 1998 and Stanley 1999.

The classification of the possible asymptotic behaviors of the Taylor coefficients of an algebraic series can be found in Flajolet 1987. The generating function of walks on the slitplane according to the length and the coordinates of the extremities was first shown algebraic and computed in Bousquet-Mélou and Schaeffer 2002. This is one in a series of results obtained recently by writing and solving linear equations with catalytic variables, see Banderier and Flajolet 2002, Bousquet-Mélou 2002 (these references are also good entry points to the literature on counting walks on lattices). The first proof we present illustrates a very general approach developed in Bousquet-Mélou 2001. The second proof is taken from Barcucci et al. 2001.

The foundation of combinatorial random generation was laid in Nijenhuis and Wilf 1978 with the recursive method. As shown in Flajolet et al. 1994, this approach leads systematically to polynomial algorithms for decomposable combinatorial structures. The (much more specialized) application of the cycle lemma to random generation is discussed in Dershowitz and Zaks 1990 and Alonso et al. 1997. The Florentine rejection algorithm is taken from Barcucci et al. 1995. A systematic utilisation of mixed probabilistic/combinatorial arguments for sampling was recently proposed in Duchon et al. 2002.

General references on polyominoes are Klarner 1997, van Rensburg 2000. Exact enumerative results are surveyed in Bousquet-Mélou 1996. The algorithms to sample convex and directed convex polyominoes are adapted from Hochstättler et al. 1996 and Del Lungo et al. 2001. From the enumeration point of view, these results are encompassed by Bousquet-Mélou and Guttmann 1997, which deals with convex polygons in any dimension. Our treatment of directed animals and heaps of bricks is adapted from Bétréma and Penaud 1993. These results built on the combinatorial intepretation of the commutation monoïd of Cartier and Foata 1969 in terms of heaps of pieces due to Viennot 1986.

Starting from the seminal work of Tutte 1962, the literature on combinatorial maps has grown almost independently in combinatorics and in physics. Some surveys are Cori and Machì 1992 (combinatorial point of view), Ambjørn et al. 1997 (physical point of view) and Di Francesco 2001 (mixed points of view).

A more detailed description of codes for plane trees appear in Chapter 11 of Lothaire 1997. The idea to use algebraic languages to encode maps already appeared in Cori 1975, and plane trees are explicitly used in Cori and Vauquelin 1981. Conjugacy classes of trees were introduced in Schaeffer 1997, as well as the bijection between balanced trees and planar maps. Applications to coding and sampling are discussed in Poulalhon and Schaeffer 2003.

CHAPTER 10

Words in Number Theory

Version June 23, 2004

10.0. Introduction

This chapter shows some examples of applications of combinatorics on words to number theory with a brief incursion into physics. These examples have a common feature: the notion of morphism of the free monoid. Such morphisms have been widely studied in combinatorics on words; they generate infinite words which can be considered as highly ordered, and which occur in an ubiquitous way in mathematics, theoretical computer science, and theoretical physics.

The first part of this chapter is devoted to the notion of automatic sequences and uniform morphisms, in connection with transcendence of formal power series with coefficients in a finite field. Namely it is possible to characterize algebraicity of these series in a simple way: a formal power series is algebraic if and only if the sequence of its coefficients is automatic, i.e., if it is the image by a letter-to-letter map of a fixed point of a uniform morphism. This criterion is known as Christol's theorem. A central tool in the study of automatic sequences is the notion of kernel of an infinite word (sequence) over a finite alphabet: this is the set of subsequences obtained by certain decimations. A rephrasing of Christol's theorem is that transcendence of a formal power series over a finite field is equivalent to infiniteness of the kernel of the sequence of its coefficients: this will be illustrated in this chapter.

Examples of applications of the properties of automatic sequences to transcendence results for power series over the rationals, and for real numbers whose base b-expansion is automatic are also given.

Then, in a second part, this chapter uses a famous infinite word, the Tribonacci word as a guideline to introduce various applications in Diophantine approximation and in simultaneous approximation. The Tribonacci word was introduced as a generalization of the celebrated Fibonacci word. It is defined as the fixed point of a non-uniform primitive morphism, called the Tribonacci morphism. We first associate in a natural way a numeration system with this morphism, that leads us to the definition of a compact subset of the plane with fractal boundary, called the Rauzy fractal. By closely studying its topological properties, we show that this compact set can be considered as a fundamental domain for a lattice of the plane, and that a particular geometric transformation, namely an exchange of pieces, can be performed on it. This transformation can furthermore be factored as a translation on the two-dimensional torus. The goal of this chapter is then to show how to deduce arithmetic properties of this translation from combinatorial properties of the Tribonacci word. In particular, it is shown how to associate with some prefixes of this infinite word best approximations for a given norm of the corresponding vector of translation. Relations to tilings and quasicrystals via the cut and project method are also mentioned.

10.1. Morphic and automatic sequences: definitions and generalities

In this section we define morphisms, uniform morphisms, morphic sequences and automatic sequences.

10.1.1. Topology and distance on the set of finite and infinite words

Let $\mathcal{A}$ be a finite alphabet. The set $\mathcal{A}$ is equipped with the discrete topology (i.e., every subset is open), and the set $\mathcal{A}^\omega$ of infinite words (that we also call here *infinite sequences*) on $\mathcal{A}$ is equipped with the corresponding product topology. It is well-known and not hard to prove that the product topology can also be defined by the following distance:

$$d((u_n)_{n\geq 0},(v_n)_{n\geq 0}) := 2^{-\min\{j\in\mathbb{N},\ u_j\neq v_j\}}.$$

The topology on $\mathcal{A}^\omega$ can be extended to the set $\mathcal{A}^*\cup\mathcal{A}^\omega$ of all finite and infinite words on $\mathcal{A}$ as follows: let $\sharp$ be a symbol not in $\mathcal{A}$. The set $\mathcal{A}\cup\{\sharp\}$ is equipped with the discrete topology and the set $(\mathcal{A}\cup\{\sharp\})^\omega$ is equipped with the product topology. Finally the set $\mathcal{A}^*$ is naturally embedded in $(\mathcal{A}\cup\{\sharp\})^\omega$ by identifying the word $u_0u_1\cdots u_d$ in $\mathcal{A}^*$ and the infinite word $u_0u_1\cdots u_d(\sharp)^\omega$ in $(\mathcal{A}\cup\{\sharp\})^\omega$ (where $(\sharp)^\omega$ stands for the infinite word whose terms are all equal to $\sharp$).

Remark 10.1.1. Note that the distance defined above can be informally described by saying that two words are close to each other if they coincide on their first letters. Also note that the set $\mathcal{A}^\omega$ is a compact set.

10.1.2. Morphisms and uniform morphisms

Let $\mathcal{A}$ and $\mathcal{B}$ be two alphabets. Let us recall that a *morphism* $h:\mathcal{A}^*\to\mathcal{B}^*$ is a map from $\mathcal{A}^*$ to $\mathcal{B}^*$ such that for all $u,v\in\mathcal{A}^*$, the relation $h(uv)=h(u)h(v)$ holds. (In other words h is a homomorphism of monoids.)

Remark 10.1.2.

- A morphism $h:\mathcal{A}^*\to\mathcal{B}^*$ is defined by its values on the elements of $\mathcal{A}$.
- The iterates of a morphism $h:\mathcal{A}^*\to\mathcal{A}^*$ are denoted h^j, $j\geq 0$, and defined by $h^0(a)=a$ for all $a\in\mathcal{A}$ and $h^{j+1}:=h\circ h^j$.

The morphism $h:\mathcal{A}^*\to\mathcal{B}^*$ is called *uniform* if all the words $h(a)$, $a\in\mathcal{A}$, have the same length. Let d be this common length, the morphism is called a *morphism of length d* or a *d-uniform morphism* or a *d-morphism*.

10.1.3. Fixed points of morphisms, morphic sequences and automatic sequences

Proposition 10.1.3. *Let $\mathcal{A}$ be an alphabet. Let $h:\mathcal{A}^*\to\mathcal{A}^*$ be a morphism such that there exists $a\in\mathcal{A}$ and $x\in\mathcal{A}^*$ with the properties:*

(i) $h(a) = ax$,
(ii) $\forall j \geq 0$, $h^j(x) \neq \varepsilon$.
Then, the sequence of words a, $h(a)$, $h^2(a)$, $\ldots, h^n(a)$, $\ldots$ *converges to an infinite word denoted* $h^\omega(a)$. *This infinite word is a fixed point of the extension of* h *by continuity to infinite words.*

Proof. The hypotheses easily imply that $h^{j+1}(a) = axh(x)h^2(x)\cdots h^j(x)$, for $j \geq 0$. Hence the word $h^j(a)$ is a nontrivial prefix of the word $h^{j+1}(a)$, which gives the convergence of the sequence of words $h^j(a)$ to an infinite word $h^\omega(a)$. Since $h^{j+1}(a) = h(h^j(a))$, letting j go to infinity establishes the claim. ∎

Remark 10.1.4. In the sequel we will say that an *infinite word* u *on the alphabet* $\mathcal{A}$ *is a fixed point of a morphism* $h : \mathcal{A}^* \to \mathcal{A}^*$ if and only if it can be obtained as in Proposition 10.1.3 above. One thus has $h(u) = u$.

The fixed points (in the sense of Remark 10.1.4) of a uniform morphism have a simple property that we give now.

Proposition 10.1.5. *An infinite word* $(u_n)_{n\geq 0}$ *on the alphabet* $\mathcal{A}$ *is a fixed point of the* d*-morphism* $h : \mathcal{A}^* \to \mathcal{A}^*$ *(in the sense of Remark 10.1.4) if and only if there exist* d *maps* $h_r : \mathcal{A} \to \mathcal{A}$, $r \in [0, d-1]$, *such that*

$$\forall n \geq 0,\ \forall r \in [0, d-1],\ u_{dn+r} = h_r(u_n).$$

Proof. Suppose that the infinite word $(u_n)_{n\geq 0}$ is a fixed point of the d-morphism $h : \mathcal{A}^* \to \mathcal{A}^*$, i.e., the limit when j goes to infinity of the sequence of words a, $h(a)$, $h^2(a)$, ..., $h^k(a)$ ..., with $a \in \mathcal{A}$ and $h(a) = ax$, with $x \in \mathcal{A}^*$ and $h^j(x) \neq \varepsilon$ for all $j \geq 0$. Since h is d-uniform, for each letter $e \in \mathcal{A}$, the word $h(e)$ can be written as $h(e) = \alpha_{e,0}\alpha_{e,1}\cdots\alpha_{e,d-1}$. We define the maps $h_r : \mathcal{A} \to \mathcal{A}$, $r \in [0, d-1]$, by: for each $e \in \mathcal{A}$, $h_r(e) := \alpha_{e,r}$. Now, for each $k \geq 0$, the length of the word $h^k(a)$ is equal to d^k hence

$$h^k(a) = u_0u_1\cdots u_{d^k-1}.$$

We thus have

$$\begin{aligned} u_0u_1\cdots u_{d^{k+1}-1} &= h^{k+1}(a) = h(h^k(a)) \\ &= h(u_0u_1\cdots u_{d^k-1}) \\ &= h(u_0)h(u_1)\cdots h(u_{d^k-1}). \end{aligned}$$

This thus gives:

$$\forall n \in [0, d^k-1],\ \forall r \in [0, d-1],\ u_{dn+r} = h_r(u_n).$$

Since this holds for all $k \geq 0$, we thus have

$$\forall n \geq 0,\ \forall r \in [0, d-1],\ u_{dn+r} = h_r(u_n).$$

Conversely suppose that there exist d maps $h_r : \mathcal{A} \to \mathcal{A}$, $r \in [0, d-1]$, such that

$$\forall n \geq 0,\ \forall r \in [0, d-1],\ u_{dn+r} = h_r(u_n).$$

Taking $n = r = 0$, we get $u_0 = h_0(u_0)$. Define the morphism $h : \mathcal{A}^* \to \mathcal{A}^*$, by

$$\forall e \in \mathcal{A},\ h(e) := h_0(e)h_1(e) \cdots h_{d-1}(e).$$

Furthermore let $a := u_0$. The morphism h is clearly uniform. We have

$$h(a) = h(u_0) = h_0(u_0)h_1(u_0) \cdots h_{d-1}(u_0) = u_0 h_1(u_0) \cdots h_{d-1}(u_0) = ax,$$

where $x := h_1(u_0) \cdots h_{d-1}(u_0)$. For all $j \geq 0$ we clearly have $|h^j(x)| = d^j(d-1)$, hence $h^j(x) \neq \varepsilon$. Thus Conditions (i) and (ii) of Proposition 10.1.3 are satisfied. It is then easy to check that $h^\omega(a)$ is precisely the word $(u_n)_{n\geq 0}$. ∎

A word $(u_n)_{n\geq 0}$ on the alphabet $\mathcal{A}$ is called a *morphic sequence* (or *substitutive sequence*) if there exists an alphabet $\mathcal{C}$, a word $(v_n)_{n\geq 0}$ on $\mathcal{C}$, a morphism $h : \mathcal{C}^* \to \mathcal{C}^*$, and a map $\varphi : \mathcal{C} \to \mathcal{A}$ such that

(i) word $(v_n)_{n\geq 0}$ is a fixed point of the morphism h (see Remark 10.1.4),

(ii) for all $n \geq 0$, one has $u_n = \varphi(v_n)$.

A word $(u_n)_{n\geq 0}$ on the alphabet $\mathcal{A}$ is called an *automatic sequence* if there exists an alphabet $\mathcal{C}$, a word $(v_n)_{n\geq 0}$ on $\mathcal{C}$, a *uniform* morphism $h : \mathcal{C}^* \to \mathcal{C}^*$, and a map $\varphi : \mathcal{C} \to \mathcal{A}$ such that

(i) the word $(v_n)_{n\geq 0}$ is a fixed point of the uniform morphism h (see Remark 10.1.4),

(ii) for all $n \geq 0$, one has $u_n = \varphi(v_n)$.

If the morphism h has length d, the word $(u_n)_{n\geq 0}$ is called *d-automatic*.

Remark 10.1.6. An automatic sequence is in particular morphic. The denomination "automatic" comes from the fact that such an infinite word can be generated by a finite automaton.

10.1.4. Examples of morphic and automatic sequences

10.1.4.1. The Fibonacci word

The (binary) Fibonacci word is defined as the fixed point (in the sense of Remark 10.1.4) of the morphism $0 \to 01$, $1 \to 0$, on the alphabet $\{0, 1\}$. The first few terms of this word are

$$0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,0 \cdots$$

The name of this word comes from the fact that iterating the morphism starting from 0 gives words whose lengths are equal to the Fibonacci numbers

1, 2, 3, 5, 8, $\cdots$:

$$\begin{array}{l} 0 \\ 0\ 1 \\ 0\ 1\ 0 \\ 0\ 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0 \\ \ldots \end{array}$$

It can be shown that this word is a Sturmian word, i.e., that the number of blocks of consecutive letters of length n occurring in the word is equal to $n+1$ for each $n \geq 1$ (see Problem 10.7.1).

10.1.4.2. The Tribonacci word

The Tribonacci word is defined as the fixed point (in the sense of Remark 10.1.4) of the morphism $1 \to 12$, $2 \to 13$, $3 \to 1$ on the alphabet $\{1, 2, 3\}$. The first few terms of this word are

$$1\ 2\ 1\ 3\ 1\ 2\ 1\ 1\ 2\ 1\ 3\ 1\ 2\ 1\ 2\ 1\ \cdots$$

Both words share many properties and the Tribonacci word can be considered as a generalization of the Fibonacci word, hence the terminology. We study in more details the Tribonacci word in Section 10.7–10.9.

10.1.4.3. The Thue–Morse word

Let us recall (see Example 1.8.4) that the (Prouhet)-Thue–Morse word is defined as the fixed point (in the sense of Remark 10.1.4) beginning with 0 of the morphism $0 \to 01$, $1 \to 10$. The first few terms of this word are

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ \cdots$$

The n-th term (starting from index 0) of this word is 0 if the sum of the binary digits of n is even, and 1 if this sum is odd. This property can easily been deduced from the results of Section 10.2.3.

10.1.4.4. The Rudin-Shapiro word

We consider on the alphabet $\{a, b, c, d\}$ the morphism

$$\begin{array}{l} a \to ab \\ b \to ac \\ c \to db \\ d \to dc \end{array}$$

Iterating this morphism starting from a gives the following fixed point

$$a\ b\ a\ c\ a\ b\ d\ b\ a\ b\ a\ c\ d\ c\ \cdots$$

The image of this infinite word by the map $a \to 1$, $b \to 1$, $c \to -1$, $d \to -1$, is called the Rudin-Shapiro word. This word begins as follows

$$+1 \quad +1 \quad +1 \quad -1 \quad +1 \quad +1 \quad -1 \quad +1 \quad +1 \quad +1 \quad +1 \quad -1 \quad -1 \quad -1 \quad \ldots$$

Denoting by $a(n)$ the number of (possibly overlapping) blocks 11 in the binary expansion of n, it can be shown that the n-th term of the Rudin-Shapiro word is equal to $(-1)^{a(n)}$. Here again, this property can easily been deduced from the results of Section 10.2.3.

10.1.4.5. The regular paperfolding word

We consider on the alphabet $\{a, b, c, d\}$ the morphism

$$\begin{aligned} a &\to ab \\ b &\to cb \\ c &\to ad \\ d &\to cd \end{aligned}$$

Iterating this morphism starting from a gives the following fixed point

$$a\ b\ c\ b\ a\ d\ c\ b\ a\ b\ c\ d\ a\ d\ c\ b\ \ldots$$

The image of this infinite word by the map $a \to 0$, $b \to 0$, $c \to 1$, $d \to 1$, is called the (regular) paperfolding word. This word begins as follows

$$0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ \cdots$$

Denoting this word by $(z_n)_{n\geq 0}$, it is easy to show that

$$z_{4n} = 0,\ z_{4n+2} = 1,\ z_{2n+1} = z_n$$

(which gives an alternative definition of the paperfolding word).

The proof of the following property of the paperfolding word is left to the reader. For any word w on the alphabet $\{0, 1\}$, define the word w^R as the word obtained by reading w backwards (in other words $(w_0 w_1 \cdots w_\ell)^R := w_\ell w_{\ell-1} \cdots w_0$). Also define the word $\overline{w}$ as the word obtaining from w by replacing 0's by 1's and 1's by 0's (in other words $\overline{w} := (1 - w_0)(1 - w_1) \cdots (1 - w_\ell)$. Define the map P on $\{0, 1\}^*$ by $P(w) := w0\overline{w^R}$. (The map P is called *perturbed symmetry*.) Then the paperfolding word is equal to $\lim_{j\to\infty} P^j(0)$.

10.2. d-Kernels and properties of automatic sequences

10.2.1. d-Kernels

Let $(u_n)_{n\geq 0}$ be an infinite word defined on the alphabet $\mathcal{A}$. Let $d \geq 2$ be an integer. The d-kernel of the word $(u_n)_{n\geq 0}$, denoted $\mathcal{K}(d, (u_n)_n)$, is the set of subsequences of the word $(u_n)_{n\geq 0}$ defined by

$$\mathcal{K}(d, (u_n)_n) := \{(u_{d^k n + r})_{n\geq 0},\ k \geq 0,\ r \in [0, d^k - 1]\}.$$

Remark 10.2.1. It is easy to prove that the d-kernel of an infite sequence $(u_n)_{n\geq 0}$ is stable under the maps D_j, $j \in [0, d-1]$, defined on the set of sequences on $\mathcal{A}$ by

$$\forall (z_n)_{n\geq 0} \in \mathcal{A}^\omega,\ D_j((z_n)_{n\geq 0}) := (z_{dn+j})_{n\geq 0}.$$

Furthermore $\mathcal{K}(d, (u_n)_n)$ is the smallest set that contains the sequence $(u_n)_{n\geq 0}$ and is stable by the maps D_j, $j \in [0, d-1]$.

10.2.2. Combinatorial characterization of automatic sequences

The notion of d-kernel permits to give a simple combinatorial characterization of automatic sequences.

Proposition 10.2.2. *Let $(u_n)_{n\geq 0}$ be an infinite sequence defined on the alphabet $\mathcal{A}$. Let $d \geq 2$ be an integer. Then, the following properties are equivalent:*

(i) the sequence $(u_n)_{n\geq 0}$ is d-automatic,

(ii) the d-kernel $\mathcal{K}(d, (u_n)_n)$ is a finite set,

(iii) there exists a finite set of sequences $\mathcal{F}$ that contains the sequence $(u_n)_{n\geq 0}$ and such that, if the sequence $(v_n)_{n\geq 0}$ belongs to $\mathcal{F}$ then, for every $j \in [0, d-1]$, the sequence $D_j((v_n)_{n\geq 0}) := (v_{dn+j})_{n\geq 0}$ belongs to $\mathcal{F}$.

Proof. (i) $\Rightarrow$ (ii). We suppose that the sequence $(u_n)_{n\geq 0}$ is d-automatic. Then there exists an alphabet $\mathcal{C}$, a sequence $(v_n)_{n\geq 0}$ on $\mathcal{C}$, a *uniform* morphism $h : \mathcal{C}^* \to \mathcal{C}^*$, and a map $\varphi : \mathcal{C} \to \mathcal{A}$ such that the sequence $(v_n)_{n\geq 0}$ is a fixed point of the uniform morphism h and for all $n \geq 0$, one has $u_n = \varphi(v_n)$.
In order to prove that the set $\mathcal{K}(d, (u_n)_n)$ is finite, it thus suffices to prove that $\mathcal{K}(d, (v_n)_n)$ is finite. We know from Proposition 10.1.5 that there exist d maps $h_r : \mathcal{A} \to \mathcal{A}$, $r \in [0, d-1]$, such that

$$\forall n \geq 0,\ \forall r \in [0, d-1],\ v_{dn+r} = h_r(v_n).$$

An easy induction on k shows the following: let $t \in [0, d^k - 1]$; write its base d expansion (possibly with leading zeros) as $t_{k-1} \ldots t_0$; then

$$\forall n \geq 0,\ v_{d^k n+t} = h_{t_0}((h_{t_1} \ldots (h_{t_{k-1}}(v_n))) \ldots).$$

In other words there exists a map f_t from $\mathcal{A}$ into itself such that $\forall n \geq 0$, we have $v_{d^k n+t} = f_t(v_n)$. The set $\mathcal{A}$ is finite, hence the set of maps from $\mathcal{A}$ to itself is also finite. This implies that there are only finitely many sequences $(v_{d^k n+t})_{n\geq 0}$, with $k \geq 0$, $t \in [0, d^k - 1]$.

(ii) $\Rightarrow$ (iii). This is an easy consequence of Remark 10.2.1.

(iii) $\Rightarrow$ (i). Let $\mathcal{F} = \{(u_n^{(1)})_{n\geq 0}, (u_n^{(2)})_{n\geq 0}, \cdots, (u_n^{(t)})_{n\geq 0}\}$ be a finite set of sequences, with $(u_n^{(1)})_{n\geq 0} = (u_n)_{n\geq 0}$ such that $\mathcal{F}$ is stable by the maps D_j, for $j \in [0, d-1]$. Define the vector $V(n)$ by

$$V(n) := \begin{pmatrix} u_n^{(1)} \\ u_n^{(2)} \\ \vdots \\ u_n^{(t)} \end{pmatrix}$$

Let $\mathcal{C} \subset \mathcal{A}^t$ be the (finite) set of values of $V(n)$. The fact that the set $\mathcal{F}$ is stable under the maps D_j for $j \in [0, d-1]$ implies that for each $j \in [0, d-1]$ there exists a matrix Θ_j of 0's and 1's, having exactly one 1 on each row, such that

$$\forall n \geq 0,\ V(dn+j) = \Theta_j V(n).$$

Using Proposition 10.1.5 we see that the sequence $(V(n))_{n\geq 0}$ is a fixed point of the d-morphism h of $\mathcal{A}^*$ defined by

$$\forall \alpha \in \mathcal{C}^*,\ h(\alpha) := (\Theta_1\alpha)\ (\Theta_2\alpha)\ ...\ (\Theta_t\alpha).$$

Now the sequence $(u_n)_{n\geq 0} = (u_n^{(1)})_{n\geq 0}$ is the (point wise) image of the sequence $(V(n))_{n\geq 0}$ by the restriction to $\mathcal{C}$ of the first projection $\mathcal{A}^t \to \mathcal{A}$. ∎

Remark 10.2.3. We have spoken in the proof of Proposition 10.2.2 (iii) above of vectors and matrices, although there is no vector space (nor module): the reader will be easily convinced that this is only a practical terminology (recall the special form of the matrices Θ_j).

10.2.3. Examples of kernels of automatic sequences

The Thue–Morse word, the Rudin-Shapiro word, and the paperfolding word are 2-automatic (see their definitions in Section 10.1.4). Namely their 2-kernels are finite:

– the definition of the Thue–Morse word $(u_n)_{n\geq 0}$ shows that $u_{2n} = u_n$ and $u_{2n+1} = 1 + u_n$ for every $n \geq 0$; hence the 2-kernel of the Thue–Morse word is

$$\mathcal{K}(2, (u_n)_n) = \{(u_n)_{n\geq 0}, (1+u_n)_{n\geq 0}\};$$

one deduces that the n-th term (starting from index 0) of the Thue–Morse word is 0 if the sum of the binary digits of n is even, and 1 if this sum is odd;

– the property of the Rudin-Shapiro word $(v_n)_{n\geq 0}$ that $v_n = (-1)^{a(n)}$, where $a(n)$ counts the number of possibly overlapping blocks 11 in the binary expansion of the integer n, shows that $v_{2n} = v_n$, $v_{4n+1} = v_n$, $v_{4n+3} = -v_{2n+1}$, for every $n \geq 0$; hence the 2-kernel of the Rudin-Shapiro word is

$$\mathcal{K}(2, (v_n)_n) = \{(v_n)_{n\geq 0}, (v_{2n+1})_{n\geq 0}, (-v_n)_{n\geq 0}, (-v_{2n+1})_{n\geq 0}\};$$

denoting by $a(n)$ the number of (possibly overlapping) blocks 11 in the binary expansion of n, one deduces that the n-th term of the Rudin-Shapiro word is equal to $(-1)^{a(n)}$;

– since the regular paperfolding word $(z_n)_{n\geq 0}$ satisfies $z_{4n} = 0$, $z_{4n+2} = 1$, $z_{2n+1} = z_n$ for every $n \geq 0$, its 2-kernel is

$$\mathcal{K}(2, (z_n)_n) = \{0, 1, (z_n)_{n\geq 0}, (z_{2n})_{n\geq 0}\}.$$

10.2.4. Properties of automatic sequences

We give below some properties, in particular closure properties, of automatic sequences.

PROPOSITION 10.2.4. *Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ be a d-automatic sequence on the alphabet $\mathcal{A}$. Then the sequences $(u_{q^n})_{n\geq 0}$ and $(u_{q^n-1})_{n\geq 0}$ are periodic from some point on.*

Proof. We prove only the second assertion, the first one is proved analogously. Since the d-kernel of the sequence $(u_n)_{n\geq 0}$ is finite (Proposition 10.2.2), the set of subsequences $\{(u_{q^k n+q^k-1})_{n\geq 0},\ k \geq 0\}$, is finite. In particular there exist $k \geq 0$ and $j \geq 1$ such that the sequences $(u_{q^k n+q^k-1})_{n\geq 0}$ and $(u_{q^{k+j} n+q^{k+j}-1})_{n\geq 0}$ are equal. In other words, the sequences $(u_{q^k n-1})_{n\geq 1}$ and $(u_{q^{k+j} n-1})_{n\geq 1}$ are equal. Replacing n by $q^j n$, $q^{2j} n$, ..., shows that the sequences $(u_{q^k n-1})_{n\geq 1}$ and $(u_{q^{k+\alpha j} n-1})_{n\geq 1}$ are equal for all $\alpha \geq 0$. Taking $n = 1$ concludes the proof. ∎

PROPOSITION 10.2.5.

Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ and $(v_n)_{n\geq 0}$ be two d-automatic sequences defined respectively on the alphabets $\mathcal{A}$ and $\mathcal{B}$. Then the sequence $(u_n, v_n)_{n\geq 0}$ defined on the alphabet $\mathcal{A} \times \mathcal{B}$ is d-automatic.

Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ be a d-automatic sequence defined on the alphabet $\mathcal{A}$. Let $\mathcal{B}$ be an alphabet and f be a map $f : \mathcal{A} \to \mathcal{B}$. Then the sequence $(f(u_n))_{n\geq 0}$ is d-automatic.

Proof. The proofs of both assertions are straightforward using the characterization of automatic sequences given in Proposition 10.2.2. ∎

PROPOSITION 10.2.6. *Let $(u_n)_{n\geq 0}$ be a sequence on the alphabet $\mathcal{A}$ that is ultimately periodic (i.e., periodic from some point on). Then, the sequence $(u_n)_{n\geq 0}$ is d-automatic for every $d \geq 1$.*

Proof. Since the sequence $(u_n)_{n\geq 0}$ is ultimately periodic, there exist two integers $n_0 \geq 0$ and $T > 1$, such that $\forall n \geq n_0$, $u_{n+T} = u_n$. Now, for $d \geq 2$, take a sequence in the d-kernel of $(u_n)_{n\geq 0}$, say $(u_{d^k n+\ell})_{n\geq 0}$, with $k \geq 0$ and $\ell \in [0, d^k - 1]$. We have for all $n \geq n_0$

$$u_{d^k(n+T)+\ell} = u_{d^k n+\ell+d^k T} = u_{d^k n+\ell}.$$

In other words all sequences $(v_n)_{n\geq 0}$ in $\mathcal{K}(d,(u_n)_n)$ satisfy $\forall n \geq n_0$, $v_{n+T} = v_n$. Hence the d-kernel of $(u_n)_{n\geq 0}$ is finite with at most $(\operatorname{Card}\mathcal{A})^{n_0+T}$ elements. ■

PROPOSITION 10.2.7. *Let $(u_n)_{n\geq 0}$ be a d-automatic sequence defined on the alphabet $\mathcal{A}$. Then*

(i) for all $a, b \in \mathbb{N}$, the sequence $(u_{an+b})_{n\geq 0}$ is d-automatic;

(ii) the sequence $(v_n)_{n\geq 0}$ defined by $v_0 = a \in \mathcal{A}$ and $v_n = u_{n-1}$ for all $n \geq 1$, is d-automatic.

Proof. (i) We may assume, from Proposition 10.2.6, that $a \geq 1$. The d-kernel $\mathcal{K}(d,(u_n)_n)$ of the sequence $(u_n)_{n\geq 0}$ is finite. Let

$$\mathcal{K}(d,(u_n)_n) := \{(u_n^{(1)})_{n\geq 0}, (u_n^{(2)})_{n\geq 0}, \ldots, (u_n^{(t)})_{n\geq 0}\},$$

with $(u_n^{(1)})_{n\geq 0} = (u_n)_{n\geq 0}$. Let us then define the set $\mathcal{L}$ of sequences by

$$\mathcal{L} := \{(u_{an+b'}^{(i)})_{n\geq 0},\ i \in [1,t],\ b' \in [0, a+b-1]\}.$$

The set $\mathcal{L}$ is clearly finite with at most $t(a+b)$ elements. It thus suffices to prove that the d-kernel of the sequence $(u_{an+b})_{n\geq 0}$ is a subset of $\mathcal{L}$. Let $(u_{a(d^k n+\ell)+b})_{n\geq 0}$ be a sequence in the d-kernel of $(u_{an+b})_{n\geq 0}$, where $k \geq 0$ and $\ell \in [0, d^k-1]$. We write $a\ell + b = xd^k + y$, with $y \in [0, d^k - 1]$. Thus,

$$u_{a(d^k n+\ell)+b} = u_{d^k(an+x)+y} = u_{an+x}^{(i)}$$

for some i that does not depend on n. Furthermore we have

$$xd^k \leq xd^k + y = a\ell + b \leq a(d^k-1) + b < ad^k + b \leq (a+b)d^k.$$

Hence $x < a+b$, and the sequence $(u_{a(d^k n+\ell)+b})_{n\geq 0}$ belongs to $\mathcal{L}$.

(ii) Let us write, as above, the (finite) d-kernel of the sequence $(u_n)_{n\geq 0}$ as

$$\mathcal{K}(d,(u_n)_n) := \{(u_n^{(1)})_{n\geq 0}, (u_n^{(2)})_{n\geq 0}, \ldots, (u_n^{(t)})_{n\geq 0}\},$$

with $(u_n^{(1)})_{n\geq 0} = (u_n)_{n\geq 0}$. Let us then define t sequences $(v_n^{(i)})_{n\geq 0}$, $i \in [1,t]$, by: $v_0^{(i)} := a$ and $v_n^{(i)} := u_{n-1}^{(i)}$ for $n \geq 1$. Note that $(v_n^{(1)})_{n\geq 0} = (v_n)_{n\geq 0}$. Consider the (finite) set $\mathcal{M}$ defined by

$$\mathcal{M} := \{(u_n^{(1)})_{n\geq 0}, (u_n^{(2)})_{n\geq 0}, \ldots, (u_n^{(t)})_{n\geq 0}, (v_n^{(1)})_{n\geq 0}, (v_n^{(2)})_{n\geq 0}, \ldots, (v_n^{(t)})_{n\geq 0}\}.$$

It suffices to prove that $\mathcal{K}(d,(v_n)_n) \subset \mathcal{M}$. Let $(v_{d^k n+\ell})_{n\geq 0}$ with $k \geq 0$ and $\ell \in [0, d^k-1]$ be an element of $\mathcal{K}(d,(v_n)_n)$.

• If $\ell \geq 1$, then, $v_{d^k n+\ell} = u_{d^k n+(\ell-1)}$. Since $(\ell-1) \in [0, d^k-1]$, the sequence $(u_{d^k n+(\ell-1)})_{n\geq 0}$ is equal to $(u_n^{(i)})_{n\geq 0}$ for some $i \in [1,t]$ hence it belongs to $\mathcal{M}$.

• If $\ell = 0$, then $(v_{d^k n+\ell})_{n\geq 0} = (v_{d^k n})_{n\geq 0}$. If $n \geq 1$, let $m = n-1 \geq 0$. We have:

$$v_{d^k n} = u_{d^k n-1} = u_{d^k m+d^k-1} = u_m^{(i)} = u_{n-1}^{(i)}$$

for some i that does not depend on n. Hence

$$v_{d^k n} = \left\{ \begin{array}{ll} a & \text{if } n = 0 \\ u_{n-1}^{(i)} & \text{if } n \geq 1 \end{array} \right\} = v_n^{(i)}.$$

■

REMARK 10.2.8. This proposition implies in particular, using (i), that shifting a d-automatic sequence gives a d-automatic sequence. Using this remark and (ii) shows that finite modifications of a d-automatic sequence give a d-automatic sequence.

PROPOSITION 10.2.9. *Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ be a sequence on an alphabet $\mathcal{A}$, such that there exists $a \in \mathbb{N} \setminus \{0\}$ for which all subsequences $(u_{an+b})_{n\geq 0}$ are d-automatic, for $b \in [0, a-1]$. Then the sequence $(u_n)_{n\geq 0}$ is d-automatic.*

Proof. In order to prove that the d-kernel of the sequence $(u_n)_{n\geq 0}$ is finite, it suffices to prove that the set of sequences of the form $(u_{d^k(an+b)+\ell})_{n\geq 0}$ for $k \geq 0$, $\ell \in [0, d^k-1]$ and $b \in [0, a-1]$ is finite: namely interspersing these sequences produces the sequences $(u_{d^k n+\ell})_{n\geq 0}$ for $k \geq 0$, $\ell \in [0, d^k-1]$.

Now, for $k \geq 0$, $\ell \in [0, d^k-1]$, and $b \in [0, a-1]$, let $d^k b + \ell = ar + s$ with $s \in [0, a-1]$. This implies

$$ar \leq ar + s = d^k b + \ell \leq d^k b + d^k - 1 < d^k(b+1) \leq d^k a$$

hence $r \in [0, d^k-1]$. Then, for $n \geq 0$,

$$u_{d^k(an+b)+\ell} = u_{a(d^k n+r)+s}.$$

This shows that the sequence $(u_{d^k(an+b)+\ell})_{n\geq 0}$ belongs to the d-kernel of the sequence $(u_{an+s})_{n\geq 0}$, hence to the (finite) set

$$\bigcup_{s\in[0,a-1]} \mathcal{K}(d, (u_{an+s})_n).$$

■

COROLLARY 10.2.10. *Let $(u_n)_{n\geq 0}$ be a sequence defined on the alphabet $\mathcal{A}$. Let $d \geq 2$ be an integer. Then the following properties are equivalent:*

(i) the sequence $(u_n)_{n\geq 0}$ is d-automatic;

(ii) there exists an integer $\alpha \geq 1$ such that the sequence $(u_n)_{n\geq 0}$ is d^α-automatic;

(iii) for every integer $\alpha \geq 1$ the sequence $(u_n)_{n\geq 0}$ is d^α-automatic.

Proof. The implication (iii) $\Rightarrow$ (ii) is trivial. Furthermore, we clearly have, for any integer $\alpha \geq 1$, the inclusion $\mathcal{K}(d^\alpha, (u_n)_n) \subset \mathcal{K}(d, (u_n)_n)$, which shows that (i) $\Rightarrow$ (iii).

It remains to prove that (ii) $\Rightarrow$ (i). Suppose that the sequence $(u_n)_{n\geq 0}$ is d^α-automatic, for some $\alpha \geq 1$. If $\alpha = 1$ we are done. Hence we can suppose that $\alpha \geq 2$. Define $d' := d^{\alpha-1}$. Fix $j \in [0, d'-1]$, and define the sequence $(v_n)_{n\geq 0}$ by: $v_n := u_{d'n+j}$. This sequence $(v_n)_{n\geq 0}$ is d-automatic: namely for each $i \in [0, d-1]$ we have $v_{dn+i} = u_{d'dn+d'i+j} = u_{d^\alpha n+d'i+j}$, hence the sequence $(v_{dn+i})_{n\geq 0}$ belongs to the finite set $\mathcal{K}(d^\alpha, (u_n)_n)$ (note that $d'i + j \leq d^\alpha - 1$). Applying now Proposition 10.2.9 with $a = d' - 1$ ends the proof. ∎

Corollary 10.2.11. *Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ be a d-automatic sequence defined on the alphabet $\mathcal{A}$. Let $\mathcal{B}$ be an alphabet and let h be a uniform morphism $h : \mathcal{A}^* \to \mathcal{B}^*$. Then the sequence $(h(u_n))_{n\geq 0}$ is d-automatic.*

Proof. We recall that the morphism h is extended by continuity to infinite sequences. Let suppose that the length of the morphism h is d'. Hence, for each letter $e \in \mathcal{A}$, the word $h(e)$ can be written as $h(e) = \alpha_{e,0}\alpha_{e,1} \cdots \alpha_{e,d'-1}$. We define the maps $h_i : \mathcal{A} \to \mathcal{A}$, $i \in [0, d'-1]$, by: for each $e \in \mathcal{A}$, $h_i(e) := \alpha_{e,i}$.

We thus can write the sequence $(h(u_n))_{n\geq 0}$ as

$$h_0(u_0)h_1(u_0)\ldots h_{d'}(u_0)h_0(u_1)h_1(u_1)\ldots h_{d'}(u_1)\ldots$$

In other words we have, for all $n \geq 0$ and for all $i \in [0, d'-1]$,

$$u_{d'n+i} = h_i(u_n).$$

But the sequences $(h_i(u_n))_{n\geq 0}$, for $i \in [0, d'-1]$ are d-automatic, from Proposition 10.2.5, hence the sequence $(u_n)_{n\geq 0}$ is d-automatic from Proposition 10.2.9. ∎

Proposition 10.2.12. *Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ and $(v_n)_{n\geq 0}$ be two d-automatic sequences defined on the alphabet $\mathcal{A}$.*

(i) If $\mathcal{A}$ is a module over a commutative ring $\mathcal{R}$, then the sequences $((u + v)_n)_{n\geq 0} := (u_n + v_n)_{n\geq 0}$ and $((xu)_n)_{n\geq 0} := (xu_n)_{n\geq 0}$, where $x \in \mathcal{R}$, are d-automatic.

(ii) If $\mathcal{A}$ is a finite commutative ring, then the (ordinary) product of the sequences $(u_n)_{n\geq 0}$ and $(v_n)_{n\geq 0}$, i.e., the sequence $((uv)_n)_{n\geq 0} := (u_n v_n)_{n\geq 0}$ is d-automatic.

(iii) If $\mathcal{A}$ is a finite commutative ring, then the Cauchy product of the sequences $(u_n)_{n\geq 0}$ and $(v_n)_{n\geq 0}$, i.e., the sequence $(\sum_{0\leq j\leq n} u_j v_{n-j})_{n\geq 0}$, is d-automatic.

Proof. Assertions in (i) and (ii) are easy consequences of Propositions 10.2.5 and 10.2.6. Let us prove assertion (iii). Let $k \geq 0$ and $\ell \in [0, d^k - 1]$. We first note that, for $n \geq 1$, writing any $i \in [0, d^k n + \ell]$ as $i = d^k m + j$, with $j \in [0, d^k - 1]$, we have

$$d^k m \leq d^k m + j = i \leq d^k n + \ell \leq d^k n + d^k - 1 < d^k(n+1).$$

This implies $m < n + 1$, hence $m \leq n$. Hence the inclusion

$$[0, d^k n + \ell] \subset \{d^k m + j,\ m \leq n - 1,\ 0 \leq j \leq d^k - 1\} \cup \{d^k n + j,\ j \in [0, \ell]\}.$$

The reverse inclusion is clear, hence

$$[0, d^k n + \ell] = \{d^k m + j,\ m \leq n - 1,\ 0 \leq j \leq d^k - 1\} \cup \{d^k n + j,\ j \in [0, \ell]\}.$$

This equality clearly implies

$$[0, d^k n+\ell] = \{d^k m+j,\ m \leq n-1,\ \ell \leq j \leq d^k-1\} \cup \{d^k m+j,\ m \leq n,\ j \in [0, \ell]\}.$$

Now let us consider our two d-automatic sequences and let us take an element in the d-kernel of the sequence $(\sum_{0 \leq i \leq n} u_i v_{n-j})_{n \geq 0}$, i.e., let $k \geq 0$ and $\ell \in [0, d^k - 1]$, then

$$\sum_{0 \leq i \leq d^k n + \ell} u_i v_{d^k n + \ell - i} = S_1(n) + S_2(n)$$

where

$$S_1(n) := \sum_{\ell < j \leq d^k - 1} \left(\sum_{0 \leq m \leq n-1} u_{d^k m + j} v_{d^k n + \ell - d^k m - j} \right)$$

and

$$S_2(n) := \sum_{0 \leq j \leq \ell} \left(\sum_{0 \leq m \leq n} u_{d^k m + j} v_{d^k n + \ell - d^k m - j} \right).$$

Writing $S_1(n)$, for $n \geq 1$, as

$$S_1(n) := \sum_{\ell < j \leq d^k - 1} \left(\sum_{0 \leq m \leq n-1} u_{d^k m + j} v_{d^k (n-1-m) + d^k + \ell - j} \right)$$

we see that, for $n \geq 1$, $S_1(n)$ is a finite sum of sequences of the type

$$\sum_{0 \leq m \leq n-1} u_m^{(r)} v_{n-1-m}^{(s)}$$

where the sequence $(u_n^{(r)})_{n \geq 0}$ (resp. $(v_n^{(s)})_{n \geq 0}$) belongs to the d-kernel of the sequence $(u_n)_{n \geq 0}$ (resp. to the d-kernel of the sequence $(v_n)_{n \geq 0}$).

We also see that $S_2(n)$ is a finite sum of sequences of the type

$$\sum_{0 \le m \le n} u_m^{(r)} v_{n-m}^{(s)}$$

where the sequence $(u_n^{(r)})_{n\ge0}$ (resp. $(v_n^{(s)})_{n\ge0}$) belongs to the d-kernel of the sequence $(u_n)_{n\ge0}$ (resp. to the d-kernel of the sequence $(v_n)_{n\ge0}$).

Hence the sequence $(S_1(n) + S_2(n))_{n\ge1}$ belongs to a finite set of sequences. Since $S_1(0) + S_2(0)$ can take only finitely many values, we are done. ∎

10.2.5. A density property for "automatic" sets of integers

This section is devoted to proving a density property of sets of integers defined by automatic sequences. Before stating it we need two definitions and a lemma.

A subset $\mathbb{M}$ of the integers is said to have a *density* if the limit

$$\lim_{x\to\infty} \frac{1}{x} \operatorname{Card}\{n \le x,\ n \in \mathbb{M}\}$$

exists. The value of this limit is called the density of the set $\mathbb{M}$.

A factor w of an infinite word x is said to have a *density* if the set of indices of occurrence of this factor in x admits a density, that is, if the limit of the number of occurrences of this factor in the first k terms of the word divided by k exists. The value of this limit, that we denote by $\pi(w)$, is called the *probability* (or the *frequency*) of occurrence of the factor w in x.

The following lemma is a direct consequence of the Perron–Frobenius theorem (for more details, see Section 1.7.2).

Lemma 10.2.13. *Let M be a positive stochastic matrix, i.e., such that all its entries are nonnegative and all the entries in any column sum up to* 1. *Then the sequence of matrices M^n converges and all the entries in the limit are rational numbers.*

Let h be a morphism on the alphabet $\mathcal{C} := \{c_1, c_2, \cdots c_t\}$. The *incidence matrix* (also called the *transition matrix* or *substitution matrix*) of h is the $t \times t$-matrix $M = (M_{ij})_{i,j}$ defined by

$$M_{ij} = |h(c_j)|_{c_i} := \text{number of occurrences of } c_i \text{ in } h(c_j).$$

Remark 10.2.14. The incidence matrix is the transpose of the matrix introduced in Section 1.8.6. If the incidence matrix of h is M, it is easy to see that the incidence matrix for h^n is M^n. If h is a d-morphism, it is clear that the entries in any column of its incidence matrix M sum up to d. We introduce this matrix also in this chapter in order to deduce probabilities of occurrence of letters.

Proposition 10.2.15. *Let $d \geq 2$ be an integer. Let $(u_n)_{n\geq 0}$ be a d-automatic sequence on the set $\mathcal{A}$. Let a belongs to $\mathcal{A}$. If the set $\{n \geq 0,\ u_n = a\}$ has a density, this density (which is the probability $\pi(a)$ of occurrence of the letter a) must be a rational number.*

Proof. Since the sequence $(u_n)_{n\geq 0}$ is d-automatic, there exists an alphabet $\mathcal{C}$, a sequence $(v_n)_{n\geq 0}$ on $\mathcal{C}$, a d-morphism $h : \mathcal{C}^* \to \mathcal{C}^*$, and a map $\varphi : \mathcal{C} \to \mathcal{A}$ such that: the sequence $(v_n)_{n\geq 0}$ is a fixed point of the d-morphism h, and for all $n \geq 0$, one has $u_n = \varphi(v_n)$.

We first note that, for each letter $c \in \mathcal{C}$ the limit

$$\lim_{n\to\infty} \frac{1}{d^n} \operatorname{Card}\{m \leq d^n - 1,\ v_m = c\}$$

exists. Namely, let $M = (M_{ij})_{i,j}$ be the incidence matrix of the d-morphism h, and let $M^n = (M_{ij}^{(n)})_{i,j}$. Then

$$\frac{1}{d^n} \operatorname{Card}\{m \leq d^n - 1,\ v_n = c\} = \frac{1}{d^n} |h^n(v_0)|_c = \frac{M_{ij}^{(n)}}{d^n} \text{ for some } i, j.$$

Since the matrix M/d is clearly positive and stochastic, Proposition 10.2.15 shows that $\lim_{n\to\infty} \frac{M_{ij}^{(n)}}{d^n}$ exists and is rational. Hence, if $\mathcal{C}' = \varphi^{-1}(a)$ is the subset of $\mathcal{C}$ consisting of the elements of $\mathcal{C}$ whose image by φ is equal to a, the limit

$$\lim_{n\to\infty} \frac{1}{d^n} \operatorname{Card}\{m \leq d^n - 1,\ u_m = a\}$$

is the sum over $\mathcal{C}'$ of rational numbers, hence a rational number itself. Since the density of the set $\{m,\ u_m = a\}$ exists, it must be equal to the previous quantity, hence rational. ∎

10.3. Christol's algebraic characterization of automatic sequences

10.3.1. Formal power series

We recall that the ring $K[[X]]$ of formal power series with coefficients in a field K is defined by

$$K[[X]] := \left\{ \sum_{n\geq 0} u_n X^n,\ u_n \in K \right\},$$

where addition and multiplication of the series $F := \sum_{n\geq 0} u_n X^n$ and $G := \sum_{n\geq 0} b_n X^n$ are defined by

$$F + G := \sum_{n\geq 0} (u_n + b_n) X^n, \quad FG := \sum_{n\geq 0} \left(\sum_{i+j=n} u_i b_j \right) X^n.$$

The ring $K[[X]]$ is a subring of the field $K((X))$ of formal Laurent series

$$K((X)) := \left\{ \sum_{n \geq -n_0} u_n X^n,\ n_0 \in \mathbb{Z},\ u_n \in K \right\},$$

where addition and multiplication are defined analogously.

Note that the field of rational functions $K(X)$ is a subfield of $K((X))$. Hence we can define algebraicity over $K(X)$ for an element belonging to $K((X))$.

The formal power series $F = F(X) = \sum_{n \geq -n_0} u_n X^n$ is said to be *algebraic (over the field $K(X)$)*, if there exist an integer $d \geq 1$ and polynomials $A_0(X)$, $A_1(X)$, …, $A_d(X)$, with coefficients in K and not all zero, such that

$$A_0 + A_1 F + A_2 F^2 + \cdots + A_d F^d = 0.$$

Remark 10.3.1.

- Any element of $K(X)$ is algebraic over $K(X)$.
- The sum and product of algebraic elements are algebraic.
- Let $F = \sum_{n \geq -n_0} u_n X^n$ be an algebraic power series. Its derivative $F' := \sum_{n \geq -n_0} n u_n X^{n-1}$ is also algebraic. Namely take an equation as above with minimal degree d.

  $$A_0 + A_1 F + A_2 F^2 + \cdots + A_d F^d = 0.$$

  Taking the derivative gives

  $$A_0' + A_1' F + A_2' F^2 + \cdots + A_d' F^d + F'(A_1 + 2A_2 F + \cdots + dA_d F^{d-1}).$$

  The coefficient of F' cannot be zero (d is minimal and the A_j's are not all zero). Hence F' is the quotient of two elements that are algebraic over $K(X)$, thus it is algebraic over $K(X)$.

10.3.2. A simple example

Let $F(X) := \sum_{n \geq 0} u_n X^n$ where $(u_n)_{n \geq 0}$ is the Thue–Morse sequence. We have

$$\begin{aligned} F(X) &= \sum_{n \geq 0} u_{2n} X^{2n} + \sum_{n \geq 0} u_{2n+1} X^{2n+1} = \sum_{n \geq 0} u_n X^{2n} + X \sum_{n \geq 0} (u_n + 1) X^{2n} \\ &= F(X^2) + XF(X^2) + X \frac{1}{1 - X^2}. \end{aligned}$$

Hence we have, over the two-element field $\mathbb{F}_2$,

$$(1 + X)^3 F(X)^2 + (1 + X)^2 F(X) + X = 0.$$

In other words the series $F(X)$ is algebraic (actually quadratic) over the field $\mathbb{F}_2(X)$.

10.3.3. Christol's theorem

The example given in Section 10.3.2 above is actually a particular case of a general property of algebraic formal power series over a finite field $\mathbb{F}_q(X)$, which is a characterization of these series. We begin with a definition and a lemma.

Let $q = p^t$ be a positive power of a prime integer p. Let $\mathbb{F}_q$ be the finite field of cardinality q (the characteristic of $\mathbb{F}_q$ is p). For $0 \leq r < q$, we define the linear map λ_r on $\mathbb{F}_q[[X]]$ by

$$\text{if } F = F(X) = \sum_{i \geq 0} u_i X^i, \text{ then } \lambda_r(F) := \sum_{i \geq 0} u_{qi+r} X^i.$$

Lemma 10.3.2. *Let $A = A(X)$ and $B = B(X)$ be two formal power series in $\mathbb{F}_q[[X]]$. Then $A = \sum_{0 \leq r < q} X^r \lambda_r(A)^q$, and $\lambda_r(A^q B) = A \lambda_r(B)$.*

Proof. The proof is left to the reader who might want to remember that we have in $\mathbb{F}_q[[X]]$ the equality

$$\left(\sum_{n \geq 0} u_n X^n\right)^q = \sum_{n \geq 0} u_n X^{qn}. \qquad \blacksquare$$

We will also need a proposition proving that, in positive characteristic, any algebraic formal power series satisfies a "special" algebraic equation.

Proposition 10.3.3. *Let p be a prime number, let $\alpha \geq 1$ be an integer, and $q := p^t$. Let $F(X)$ be a formal power series with coefficients in $\mathbb{F}_q$. Then F is algebraic over $\mathbb{F}_q(X)$ if and only if there exist polynomials $B_0(X), \ldots, B_t(X)$ in $\mathbb{F}_q[X]$ not all equal to zero, such that*

$$B_0 F + B_1 F^q + B_2 F^{q^2} + \cdots + B_t F^{q^t} = 0.$$

Furthermore we can suppose that $B_0 \neq 0$.

Proof. If the formal power series $F(X)$ satisfies

$$B_0 F + B_1 F^q + B_2 F^{q^2} + \cdots + B_t F^{q^t} = 0,$$

where the polynomials $B_j(X)$ are not all equal to zero, then F is clearly algebraic over $\mathbb{F}_q(X)$. Now, if F is algebraic, the series F, F^q, F^{q^2}, $\ldots$, cannot be all linearly independent. Hence there exists a nontrivial linear relation

$$B_0 F + B_1 F^q + B_2 F^{q^2} + \cdots + B_t F^{q^t} = 0.$$

Let us prove that there exists such a relation with $B_0 \neq 0$. Suppose that

$$B_0 F + B_1 F^q + B_2 F^{q^2} + \cdots + B_t F^{q^t} = 0$$

with t minimal, and let j be the smallest non-negative integer such that $B_j \neq 0$. We will prove that $j = 0$. Since

$$B_j = \sum_{0 \le r < q} X^r (\lambda_r(B_j))^q$$

by Lemma 10.3.2, it follows that there exists r with $\lambda_r(B_j) \neq 0$. Now, since $\sum_{j \le i \le t} B_i F(X)^{q^i} = 0$, we have

$$\sum_{j \le i \le t} \lambda_r(B_i F^{q^i}) = 0$$

and, using (10.3.2), we see that, if $j \neq 0$, then

$$\sum_{j \le i \le t} \lambda_r(B_i) F^{q^{i-1}} = 0,$$

which gives a new relation with the coefficient of $F^{q^{j-1}} \neq 0$, a contradiction, hence $j = 0$. We thus have the relation

$$\sum_{0 \le i \le t} B_i F^{q^i} = 0,$$

with $B_0 \neq 0$. ∎

We now state Christol's theorem.

Theorem 10.3.4. *Let $\mathcal{A}$ be a non-empty alphabet, and let $(u_n)_{n \ge 0}$ be a sequence of elements of $\mathcal{A}$. Let p be a prime number. Then the sequence $(u_n)_{n \ge 0}$ is p-automatic if and only if there exists an integer $\alpha \ge 1$ and an injective map $\iota : \mathcal{A} \to \mathbb{F}_{p^\alpha}$ such that the formal power series $\sum_{n \ge 0} \iota(u_n) X^n$ is algebraic over $\mathbb{F}_{p^\alpha}(X)$.*

Proof. Let us first suppose that the sequence $(u_n)_{n \ge 0}$ is p-automatic. Choose α such that $|\mathcal{A}| \le p^\alpha$, and choose an injective map $\iota : \mathcal{A} \to \mathbb{F}_{p^\alpha}$. Up to notations we may suppose that $\mathcal{A} \subset \mathbb{F}_{p^\alpha}$ and that ι is the identity map. We thus want to prove that the formal power series $\sum_{n \ge 0} u_n X^n$ is algebraic over $\mathbb{F}_{p^\alpha}[X]$. Since the sequence $(u_n)_{n \ge 0}$ is p-automatic, it is also p^α-automatic from Corollary 10.2.10. Hence $\mathcal{K}(p^\alpha, (u_n)_n)$ is finite, say

$$\mathcal{K}(p^\alpha, (u_n)_n) = \{(u_n^{(1)})_{n \ge 0}, (u_n^{(2)})_{n \ge 0}, \cdots, (u_n^{(t)})_{n \ge 0}\}$$

with $(u_n^{(1)})_{n \ge 0} = (u_n)_{n \ge 0}$. Let us define

$$F_j(X) := \sum_{n \ge 0} u_n^{(j)} X^n \text{ for } j \text{ in } [1, t].$$

Then, for j such that $1 \leq j \leq t$, we have

$$F_j(X) = \sum_{0 \leq r \leq p^\alpha - 1} \left(\sum_{m \geq 0} u^{(j)}_{p^\alpha m + r} X^{p^\alpha m + r} \right) = \sum_{0 \leq r \leq p^\alpha - 1} X^r \sum_{m \geq 0} u^{(j)}_{p^\alpha m + r} X^{p^\alpha m}.$$

But the sequence $(u^{(j)}_{p^\alpha m + r})_{m \geq 0}$ is one of the sequences $(u^{(i)}(m))_{m \geq 0}$, hence $F_j(X)$ is a linear combination, with coefficients in the field $\mathbb{F}_{p^\alpha}(X)$, of the power series $F_i(X^{p^\alpha})$. In other words, for all $j \in [1, t]$, the formal power series $F_j(X)$ belongs to the $\mathbb{F}_{p^\alpha}(X)$-vector space generated by the t series $F_i(X^{p^\alpha})$, $i \in [1, t]$:

$$F_j(X) \in \langle F_1(X^{p^\alpha}), F_2(X^{p^\alpha}), \ldots, F_t(X^{p^\alpha}) \rangle.$$

This implies that for all $j \in [1, t]$

$$F_j(X^{p^\alpha}) \in \langle F_1(X^{p^{2\alpha}}), F_2(X^{p^{2\alpha}}), \ldots, F_t(X^{p^{2\alpha}}) \rangle,$$

and thus that for all $j \in [1, t]$

$$F_j(X) \in \langle F_1(X^{p^{2\alpha}}), F_2(X^{p^{2\alpha}}), \ldots, F_t(X^{p^{2\alpha}}) \rangle.$$

Hence, for all $j \in [1, t]$,

$$F_j(X) \text{ and } F_j(X^{p^\alpha}) \in \langle F_1(X^{p^{2\alpha}}), F_2(X^{p^{2\alpha}}), \ldots, F_t(X^{p^{2\alpha}}) \rangle.$$

This implies that, for all $j \in [1, t]$,

$$F_j(X^{p^\alpha}) \text{ and } F_j(X^{p^{2\alpha}}) \in \langle F_1(X^{p^{3\alpha}}), F_2(X^{p^{3\alpha}}), \ldots, F_t(X^{p^{3\alpha}}) \rangle.$$

Hence, for all $j \in [1, t]$,

$$F_j(X),\ F_j(X^{p^\alpha}) \text{ and } F_j(X^{p^{2\alpha}}) \in \langle F_1(X^{p^{3\alpha}}), F_2(X^{p^{3\alpha}}), \ldots, F_t(X^{p^{3\alpha}}) \rangle.$$

Iterating, we have, for all $j \in [1, t]$ and for all $k \in [0, t]$,

$$F_j(X^{p^{k\alpha}}) \in \langle F_1(X^{p^{(t+1)\alpha}}), F_2(X^{p^{(t+1)\alpha}}), \ldots, F_t(X^{p^{(t+1)\alpha}}) \rangle.$$

But the dimension of a finitely generated vector space is at most the number of its generators. Hence the dimension of the $\mathbb{F}_{p^\alpha}(X)$-vector space

$$\langle F_1(X^{p^{(t+1)\alpha}}), F_2(X^{p^{(t+1)\alpha}}), \ldots, F_t(X^{p^{(t+1)\alpha}}) \rangle$$

is at most t. Hence for any $j \in [1, t]$, there must exist a nontrivial linear relation between the formal power series

$$F_j(X), F_j(X^{p^\alpha}), \ldots, F_j(X^{p^{t\alpha}})$$

over $\mathbb{F}_{p^\alpha}(X)$. Taking $j = 1$, and remembering that $F_j(X^{p^{k\alpha}}) = F_j^{p^{k\alpha}}(X)$ (the ground field is $\mathbb{F}_{p^\alpha}$) this gives that $F(X) = F_1(X) = \sum_{n \geq 0} u^{(1)}_n X^n$ is algebraic over $\mathbb{F}_{p^\alpha}(X)$.

Let us now suppose that there exist an integer $\alpha \geq 1$ and an injective map $\iota : \mathcal{A} \to \mathbb{F}_{p^\alpha}$ such that the formal power series $\sum_{n\geq 0} \iota(u_n)X^n$ is algebraic over $\mathbb{F}_{p^\alpha}(X)$. The sequence $(u_n)_{n\geq 0}$ is p-automatic if and only if the sequence $(\iota(u_n))_{n\geq 0}$ is p-automatic. Up to renaming we can suppose that $\mathcal{A} \subset \mathbb{F}_{p^\alpha}$ and that the formal power series $F := \sum_{n\geq 0} u_n X^n$ is algebraic over $\mathbb{F}_{p^\alpha}(X)$. Then, from Proposition 10.3.3, there exist polynomials $B_0(X), \ldots, B_t(X)$ with $B_0 \neq 0$ such that

$$\sum_{0\leq i\leq t} B_i(X)F(X)^{q^i} = 0.$$

Define $G = G(X) := \frac{F(X)}{B_0(X)}$. Then

$$\sum_{0\leq i\leq t} B_i(X)B_0(X)^{q^i}G(X)^{q^i} = 0,$$

i.e.,

$$G(X) = \sum_{1\leq i\leq t} C_i(X)G(X)^{q^i} \quad \text{where } C_i(X) := -B_i(X)B_0^{q^i-2}(X).$$

Now let $N = \max(\deg B_0, \max\{\deg C_i\})$, and define $\mathcal{H}$ by

$$\mathcal{H} := \left\{ H \in \mathbb{F}_{p^\alpha}[[X]],\ H = \sum_{0\leq i\leq t} D_i G^{q^i} \text{ with } D_i \in \mathbb{F}_{p^\alpha}[X] \text{ and } \deg D_i \leq N \right\}.$$

It is clear that $\mathcal{H}$ is a finite set and that $F = B_0 G$ belongs to $\mathcal{H}$. We now prove that $\mathcal{H}$ is mapped into itself by λ_r. Let $H \in \mathcal{H}$. Then

$$\begin{aligned}\lambda_r(H) = \lambda_r\left(D_0G + \sum_{1\leq i\leq t} D_iG^{q^i}\right) &= \lambda_r\left(\sum_{1\leq i\leq t}(D_0C_i + D_i)G^{q^i}\right)\\ &= \sum_{1\leq i\leq t}\lambda_r(D_0C_i + D_i)G^{q^{i-1}}.\end{aligned}$$

Since $\deg D_0, \deg D_i, \deg C_i \leq N$, we have $\deg(D_0C_i + D_i) \leq 2N$, and hence

$$\deg(\lambda_r(D_0C_i + D_i)) \leq \frac{2N}{q} \leq N.$$

Hence $\mathcal{H}$ is a finite set that contains F and that is stable under the maps λ_r for $r \in [0, p^\alpha - 1]$. This clearly implies that the p^α-kernel of the sequence $(u_n)_{n\geq 0}$ is finite. The sequence $(u_n)_{n\geq 0}$ is thus p^α-automatic, hence p-automatic (Corollary 10.2.10). ∎

10.4. An application to transcendence in positive characteristic

The Christol theorem is a combinatorial criterion that can be used as a tool to prove the transcendence of formal power series over a finite field. We give here an automata-based proof of transcendence for the Carlitz formal power series Π.

Let p be a prime number. Let α be an integer ≥ 1 and let $q := p^\alpha$. The Carlitz formal power series Π_q is defined by

$$\Pi_q := \prod_{k\geq 1}\left(1 - \frac{X^{q^k} - X}{X^{q^{k+1}} - X}\right).$$

Remark 10.4.1. Note that Π_q belongs to $\mathbb{F}_q((X^{-1}))$.

Theorem 10.4.2. *The formal power series Π_q is transcendental over the field $F_q(X)$.*

Proof. We first compute Π'_q/Π_q, where Π'_q is the derivative of Π_q (with respect to X). It is easy to obtain:

$$\frac{\Pi'_q}{\Pi_q} = \left(\sum_{k\geq 1}\frac{1}{X^{q^k} - X}\right) - \frac{1}{X^q - X}.$$

If Π_q were algebraic over $\mathbb{F}_q(X)$, then Π'_q would also be algebraic in view of Remark 10.3.1. Hence Π'_q/Π_q would be algebraic. Since $1/(X^q - X)$ is rational, this would imply that $\sum_{k\geq 1}\frac{1}{X^{q^k}-X}$ would be algebraic over $\mathbb{F}_q(X)$. We then write

$$\begin{aligned}\sum_{k\geq 1}\frac{1}{X^{q^k} - X} &= \frac{1}{X}\sum_{k\geq 1}\frac{1}{X^{q^k-1}}\sum_{n\geq 0}\left(\frac{1}{X}\right)^{n(q^k-1)} \\ &= \frac{1}{X}\sum_{\substack{k\geq 1\\ n\geq 0}}\left(\frac{1}{X}\right)^{(n+1)(q^k-1)} = \frac{1}{X}\sum_{\substack{k\geq 1\\ n\geq 1}}\left(\frac{1}{X}\right)^{n(q^k-1)} \\ &= \frac{1}{X}\sum_{m\geq 1}\left(\frac{1}{X}\right)^m c(m),\end{aligned}$$

where

$$c(m) := \sum_{\substack{k,n\geq 1\\ n(q^k-1)=m}} 1 = \sum_{\substack{k\geq 1\\ q^k-1\mid m}} 1 = \sum_{\substack{k\geq 1\\ q^k-1\mid m}} 1.$$

We then note that $\mathbb{F}_q(X) = \mathbb{F}_q(X^{-1})$. Hence, replacing X by X^{-1} in Christol's theorem, we see that the algebraicity of Π_q would imply the q-automaticity of the sequence $(c(m))_{m\geq 1}$.

Now, if the sequence $(c(m))_{m\geq 1}$ were q-automatic, then the subsequence $(c(q^n-1))_{n\geq 0}$ would be ultimately periodic by Proposition 10.2.4. But

$$c(q^n-1) = \sum_{\substack{k\geq 1\\ q^k-1|q^n-1}} 1 = \sum_{\substack{k\geq 1\\ k|n}} 1 = d(n)$$

by Problem 10.4.1, where $d(n)$ is the number of positive integral divisors of n.

Since $q = p^k$ for some $k \geq 1$, where p is a prime, we would have that $(d(n) \bmod p)_{n\geq 1}$ is ultimately periodic. Hence there would exist integers $t \geq 1, n_0 \geq 0$ such that, for all $n \geq n_0$ and $k \geq 1$,

$$d(n+kt) \equiv d(n) \pmod{p}.$$

Take $k = nk'$. Then

$$d(n(1+k't)) \equiv d(n) \pmod{p}$$

for all $k' \geq 1$. Now by Dirichlet's theorem we can find $k' \geq 1$ such that $p' = 1 + k't$ is a prime. Take $n = p'$. We get

$$d({p'}^2) \equiv d(p') \pmod{p}$$

and hence $3 \equiv 2 \pmod{p}$, hence the desired contradiction. ∎

10.5. An application to transcendental power series over the rationals

We recall the following definition.

A word w on the alphabet $\mathcal{A}$ is called *primitive* if it cannot be written as $w = v^\alpha$ for some word v in $\mathcal{A}$ and some integer $\alpha \geq 2$.

PROPOSITION 10.5.1. *Let $\psi_k(n)$ be the number of primitive words of length n over the alphabet $\mathcal{A}$ with* $\operatorname{Card}\mathcal{A} = k \geq 2$. *Then the formal power series $R(X) = \sum_{k\geq 1} \psi_k(n)X^n$ is transcendental over $\mathbb{Q}(X)$.*

Proof. We recall that $\psi_k(n) = \sum_{d|n} \mu(d)k^{n/d}$, where μ is the Möbius function (see Problem 10.5.1). If the series $R(X) = \sum_{k\geq 1} \psi_k(n)X^n$ were algebraic over $\mathbb{Q}(X)$, then the series $\widetilde{R}(X) := \sum_{n\geq 1} \frac{\psi_k(n)}{k} X^n$ would also be algebraic over $\mathbb{Q}(X)$. Thus (note that the number $\frac{\psi_k(n)}{k}$ is an integer for every $n \geq 1$) for any prime number p the series

$$\widetilde{R}_p(X) = \sum_{n\geq 1} (\frac{\psi_k(n)}{k} \bmod p) X^n$$

would be algebraic over the field $\mathbb{F}_p(X)$ from Problem 10.5.2. Take any prime number p dividing k (recall that $k \geq 2$). We see that $\psi_k(n)/k \equiv \mu(n) \bmod p$. Hence the series

$$\sum_{n\geq 1} (\mu(n) \bmod p) X^n$$

would be algebraic over $\mathbb{F}_p(X)$. It follows, using Theorem 10.3.4. that the sequence $(\mu(n) \bmod p)_{n\geq 0}$ would be p-automatic. From Proposition 10.2.15 this implies that, if the set

$$\{n \geq 1,\ \mu(n) \equiv 0 \bmod p\} = \{n \geq 1,\ \mu(n) = 0\}$$

has a density, this density would be rational. But this set has a density equal to $1 - 6/\pi^2$ (see Problem 10.5.1), which gives the desired contradiction. ■

Remark 10.5.2. Proposition 10.5.1 and the Chomsky-Schützenberger theorem imply the following result: if the language of primitive words over an alphabet of size ≥ 2 is context-free, it must be inherently ambiguous.

10.6. An application to transcendence of real numbers

We will prove in this section a theorem of transcendence (over the rationals) of real numbers whose base b-expansion is the fixed point of a morphism satisfying some extra hypotheses. This theorem is a consequence of a combinatorial version of a theorem of Ridout. We first give Ridout's theorem without proof.

Theorem 10.6.1. *Let $\xi \neq 0$ be a real algebraic number. Let ρ, c_1, c_2, c_3 be positive constants, and let λ and μ satisfy $0 \leq \lambda, \mu \leq 1$. Let $r', r'' \geq 0$ be integers, and suppose $\omega_1, \omega_2, \ldots, \omega_{r'+r''}$ are finitely many distinct primes. Assume there exist infinitely many fractions p_n/q_n such that*

$$\left|\frac{p_n}{q_n} - \xi\right| \leq c_1 \left|q_n\right|^{-\rho}.$$

Furthermore, suppose that p_n and q_n are not zero and can be written in the form

$$p_n = p'_n \prod_{j=1}^{r'} \omega_j^{e_j}, \qquad q_n = q'_n \prod_{j=r'+1}^{r'+r''} \omega_j^{e_j},$$

where the e_i are non-negative integers that may depend on n, and the (p'_n)'s and (q'_n)'s are positive integers that may depend on n. Finally, suppose that

$$0 < |p'_n| \leq c_2 \left|p_n\right|^{\lambda}, \quad 0 < |q'_n| \leq c_3 \left|q_n\right|^{\mu}.$$

for all $n \geq 0$. Then

$$\rho \leq \lambda + \mu.$$

Corollary 10.6.2. *Let ξ be an irrational number. Suppose that, for every integer $n \geq 0$, the base-k expansion of ξ begins by $0.U_nV_nV_nV_n'$, where U_n belongs to $\{0,1,\cdots,k-1\}^*$, V_n belongs to $\{0,1,\cdots,k-1\}^+$, and the word V_n' is a prefix of V_n. Furthermore suppose that $\lim_{n\to\infty}|V_n| = \infty$, and that there exist real numbers $0 \leq \alpha < \infty$ and $\beta > 0$ such that for all $n \geq 0$ we have $|U_n| \leq \alpha|V_n|$ and $|V_n'| \geq \beta|V_n|$. Then ξ is a transcendental number.*

Proof. Let $r_n = |U_n|$, $s_n = |V_n|$, and $s_n' = |V_n'|$, so, for all $n \geq 0$, we have $r_n \leq \alpha s_n$ and $s_n' \geq \beta s_n$. Define t_n to be the rational number whose base-k expansion is $t_n = 0.U_nV_nV_nV_n\cdots$. Hence $t_n = \dfrac{p_n}{k^{r_n}(k^{s_n}-1)}$, for some integer p_n. Note that

$$|\xi - t_n| < \frac{1}{k^{r_n+2s_n+s_n'}}.$$

Now,

$$\frac{s_n}{r_n+s_n} \geq \frac{1}{1+\alpha}$$

and

$$\frac{r_n+2s_n+s_n'}{r_n+s_n} \geq 1 + \frac{1+\beta}{1+\alpha}.$$

Hence there exist two positive real numbers μ, ρ such that

$$1 + \frac{s_n}{r_n+s_n} < 1+\mu < \rho < \frac{r_n+2s_n+s_n'}{r_n+s_n}$$

for infinitely many n. With this choice of μ and ρ, let us take $p_n' = p_n$, $\lambda = 1$, $c_2 = 1$, $q_n' = k^{s_n} - 1$. Let us choose the primes $\omega_{r'+1}, \cdots, \omega_{r'+r''}$ to be the prime divisors of k. Finally, defining $e_{r'+1}, \cdots, e_{r'+r''}$ by $k^{r_n} = \prod_{i=r'+1}^{r'+r''} \omega_j^{e_j}$, we can apply Ridout's theorem if ξ were algebraic irrational, and deduce that $\rho \leq \lambda + \mu$, which gives a contradiction. Hence ξ is transcendental. (Note that the t_n's are not necessarily in their irreducible forms, but there is an infinite number of them, since the sequence $(t_n)_n$ converges to ξ, which is irrational from the hypothesis.) ∎

We deduce a theorem on transcendence of certain "automatic" real numbers. By abuse of notation with respect to Section 1.2.2, we define here an *overlap* as a word of the form wwa where a is the first letter of w (in other words an overlap is the beginning of a cube just longer than a square).

Theorem 10.6.3. *If the expansion of the real number $\xi \in (0,1)$ in some integer base $b \geq 2$ is a non-ultimately periodic fixed point of a d-morphism h for some $d \geq 2$, and if furthermore this expansion contains an overlap, then the number ξ is transcendental.*

Proof. We write the base-k expansion of ξ as $\xi = 0.UVVa\cdots$, where U and V are finite words, and a is the first letter of V. Since the expansion of ξ

is a fixed point of the d-morphism h, then this expansion also begins with $h^n(U)h^n(V)h^n(V)h^n(a)$ for every $n \geq 1$. We can apply the previous corollary with $U_n = h^n(U)$, $V_n = h^n(V)$, and $V'_n = h^n(a)$: namely $|h^n(U)| = d^n|U|$, $|h^n(V)| = d^n|V|$, and $|h^n(a)| = d^n$. ■

10.7. The Tribonacci word

The aim of this section is to use the Tribonacci word as a guideline to introduce various applications of combinatorics on words and symbolic dynamics to arithmetics.

10.7.1. Definitions and notation

Let us recall that the *Tribonacci word* is defined as the fixed point (in the sense of Remark 10.1.4) of the *Tribonacci morphism* $\sigma : \{1,2,3\}^* \to \{1,2,3\}^*$ defined on the letters of the alphabet $\{1,2,3\}$ as follows: $\sigma : 1 \mapsto 12$, $2 \mapsto 13$, $3 \mapsto 1$. Let us observe that the Tribonacci morphism admits a unique one-sided fixed point u in $\{1,2,3\}^\omega$.

The *incidence matrix* of the Tribonacci morphism σ is $M_\sigma = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$. This matrix is easily seen to be primitive. Hence the Perron–Frobenius theorem applies (for more details, see Section 1.7.2).

Indeed the characteristic polynomial of M_σ is $X^3 - X^2 - X - 1$; this polynomial admits one positive root $\beta > 1$ (the dominant eigenvalue) and two complex conjugates α and $\overline{\alpha}$, with $|\alpha| < 1$; in particular, one has $1/\beta = \alpha\overline{\alpha}$. Hence β is a *Pisot number*, that is, an algebraic integer with all Galois conjugates having modulus less than 1.

In particular, the incidence matrix M_σ admits as eigenspaces in $\mathbb{R}^3$ one *expanding eigenline* (generated by the eigenvector with positive coordinates $v_\beta = (1/\beta, 1/\beta^2, 1/\beta^3)$ associated with the eigenvalue β) and a *contracting eigenplane* $\mathcal{P}$; we denote by v_α and $v_{\overline{\alpha}}$ the eigenvectors in $\mathbb{C}^3$ associated with α and $\overline{\alpha}$, normalized in such a way that the sum of their coordinates equals 1.

One associates with the Tribonacci word $u = (u_n)_{n \geq 0}$ a broken line starting from 0 in $\mathbb{Z}^3$ and approximating the expanding line v_β as follows. Let us first introduce the *abelianization map* f of the free monoid $\{1,2,3\}^*$ defined by

$$f : \{1,2,3\}^* \to \mathbb{Z}^3, \; f(w) = |w|_1 e_1 + |w|_2 e_2 + |w|_3 e_3,$$

where $|w|_i$ denotes the number of occurrences of the letter i in the word w, and (e_1, e_2, e_3) denotes the canonical basis of $\mathbb{R}^3$. Note that for every finite word w, we have

$$f(\sigma(w)) = M_\sigma f(w).$$

The *Tribonacci broken line* is defined as the broken line which joins with segments of length 1 the points $f(u_0 u_1 \cdots u_{N-1})$, $N \in \mathbb{N}$ (see Figure 10.1). In other words we describe this broken line by starting from the origin, and then by reading successively the letters of the Tribonacci word u, going one step in direction e_i if one reads the letter i.

We will see in Section 10.7.3 that the vectors $f(u_0u_1 \ldots u_N)$, $N \in \mathbb{N}$, stay within bounded distance of the expanding line, which is exactly the direction given by the vector of probabilities of occurrence $(\pi(1), \pi(2), \pi(3))$ of the letters $1, 2, 3$ in u. It is then natural to try to represent these points by projecting them along the expanding direction onto a transverse plane, that we chose here to be the plane $x+y+z=0$. The closure of the set of projected vertices of the broken line is called the *Rauzy fractal* and is represented on Figure 10.2. We detail this construction in Section 10.8.1. We then study the arithmetic and topological properties of the Rauzy fractal in Section 10.8.2 and 10.8.4, respectively, which leads to the proof of the main theorem of this section: Theorem 10.8.16 states that the Tribonacci word codes the orbit of the point 0 under the action of the toral translation in $\mathbb{T}^2$: $x \mapsto x + (\frac{1}{\beta}, \frac{1}{\beta^2})$. We discuss in Section 10.9 some applications of this theorem to simultaneous approximations: it is proved that the points of the broken line corresponding to $\sigma^n(1)$, $n \in \mathbb{N}$, produce best approximations for the vector $(\frac{1}{\beta}, \frac{1}{\beta^2})$ for a given norm associated with the matrix M_σ.

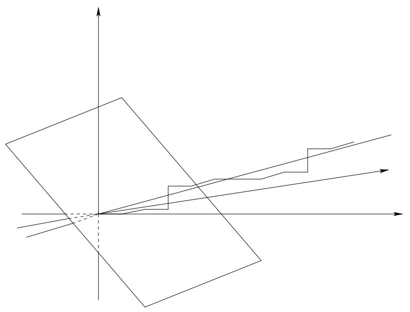


Figure 10.1. The Rauzy broken line.

10.7.2. Numeration in Tribonacci base

We now introduce two numeration systems which will be used to expand here either natural integers or finite factors of the Tribonacci word.

The sequence of lengths $T = (T_n)_{n\geq 0}$ of the words $\sigma^n(1)$ is called the *sequence of Tribonacci numbers*. One has $T_0 = 1$, $T_1 = 2$, $T_2 = 4$ and for all $n \in \mathbb{N}$, $T_{n+3} = T_{n+2} + T_{n+1} + T_n$. Indeed, one has for $n \in \mathbb{N}$

$$\sigma^{n+3}(1) = \sigma^{n+2}(12) = \sigma^{n+2}(1)\sigma^{n+1}(13) = \sigma^{n+2}(1)\sigma^{n+1}(1)\sigma^n(1).$$

Let us observe that this sequence is increasing, and thus tends to infinity.

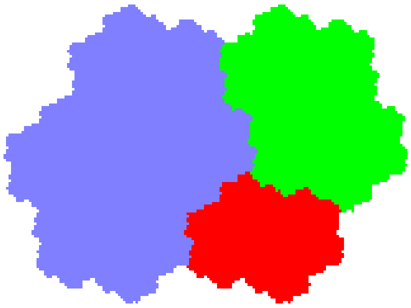


Figure 10.2. The Rauzy fractal.

A *greedy* or *normal representation* in the system T of a nonnegative integer N is a finite sequence of digits $(\varepsilon_i)_{0\le i\le k}$ where for all i, $\varepsilon_i \in \{0,1\}$ and $\varepsilon_{i+2}\varepsilon_{i+1}\varepsilon_i = 0$, $\varepsilon_k \neq 0$ such that

$$N = \sum_{i=0}^{k} \varepsilon_i T_i.$$

Lemma 10.7.1. *Every nonnegative integer admits a unique normal T-representation.*

Proof. Let us first prove the existence of the decomposition by induction. We consider the following induction property: for any integer $0 \le N < T_k$ (with $k \ge 1$), there exists a decomposition $N = \sum_{i=0}^{k-1} \varepsilon_i T_i$, where for all i, $\varepsilon_i \in \{0,1\}$ and $\varepsilon_{i+2}\varepsilon_{i+1}\varepsilon_i = 0$. This property holds for $k = 1, 2$.

Suppose that the induction hypothesis holds for the integer $k \ge 2$. Let $T_k \le N < T_{k+1} = T_k + T_{k-1} + T_{k-2}$; we have $N - T_k < T_k$ and by hypothesis, $N - T_k = \sum_{i=0}^{k-1} \varepsilon_i T_i$, hence $N = T_k + \sum_{i=0}^{k-1} \varepsilon_i T_i$. Assume that $\varepsilon_{k-1} = 1$. Since $N < T_{k+1} = T_k + T_{k-1} + T_{k-2}$, then $\varepsilon_{k-2} = 0$, and the property holds for $k+1$.

The unicity of a normal T-expansion is a direct consequence of the following observation: one has $\sum_{i=0}^{k} \varepsilon_i T_i < T_{k+1}$, where for all i, $\varepsilon_i \in \{0,1\}$ and $\varepsilon_{i+2}\varepsilon_{i+1}\varepsilon_i = 0$.

This can be easily be proved by induction. Indeed if $\varepsilon_k = \varepsilon_{k-1} = 1$, then $\varepsilon_{k-2} = 0$ and $\sum_{i=0}^{k} \varepsilon_i T_i = T_k + T_{k-1} + \sum_{i=0}^{k-3} \varepsilon_i T_i$. By induction hypothesis, $\sum_{i=0}^{k-2} \varepsilon_i T_i < T_{k-2}$, hence we get that $\sum_{i=0}^{k} \varepsilon_i T_i < T_k + T_{k-1} + T_{k-2} = T_{k+1}$. ∎

Lemma 10.7.2. *Every prefix w of the Tribonacci word u can be uniquely expanded as*

$$w = \sigma^n(p_n)\sigma^{n-1}(p_{n-1}) \cdots p_0,$$

where the finite words p_i are either equal to the empty word ε or to the letter 1, $p_n \neq \varepsilon$, and if $p_i = p_{i-1} = 1$, then $p_{i-2} = \varepsilon$; furthermore, $|w|$ admits as normal

T-representation $|w| = \sum_{i=0}^{k} \varepsilon_i T_i$, with $\varepsilon_i = 1$ if $p_i = 1$, and $\varepsilon_i = 0$, otherwise. Conversely every finite word that can be decomposed under this form is a prefix of the Tribonacci word.

Such a representation is called normal Tribonacci representation.

Proof. The proof works exactly in the same way as the proof of Lemma 10.7.1. Let us prove by induction on $n \geq 1$ that every prefix w of length $|w| < |\sigma^n(1)|$ can be decomposed as

$$w = \sigma^{n-1}(p_{n-1})\sigma^{n-2}(p_{n-2}) \cdots p_0,$$

where the finite words p_i are either equal to the empty word ε or to the letter 1, $p_{n-1} \neq \varepsilon$, and if $p_i = p_{i-1} = 1$, then $p_{i-2} = 0$. The induction property holds for $n = 1, 2$.

Let w be a prefix of length at least 4 of the Tribonacci word. Then there exists a positive integer $n \geq 2$ such that $|\sigma^n(1)| \leq |w| < |\sigma^{n+1}(1)|$. One has $\sigma^{n+1}(1) = \sigma^n(1)\sigma^{n-1}(1)\sigma^{n-2}(1)$. Put $p_n = 1$; put $p_{n-1} = 1$, if $|w| \geq |\sigma^n(1)| + |\sigma^{n-1}(1)|$, and $p_{n-1} = \varepsilon$, otherwise.

Let v be such that $w = \sigma^n(p_n)\sigma^{n-1}(p_{n-1})v$ (v may be equal to the empty word); v is a prefix either of $\sigma^{n-1}(1)$ or of $\sigma^{n-2}(1)$. If $p_{n-1} = 1$, then $|v| < |\sigma^{n-2}(1)|$. We conclude by applying the induction hypothesis on v.

The unicity of such an expansion, as well as the corresponding normal T-representation for $|w|$, is a direct consequence of the fact that $|\sigma^n(1)| = T_n$ and of the unicity of normal T-representations (Lemma 10.7.1).

Let us prove by induction on n that every finite word of the form

$$\sigma^{n-1}(p_{n-1})\sigma^{n-2}(p_{n-2}) \cdots p_0,$$

where the finite words p_i are either equal to the empty word ε or to the letter 1, $p_n \neq \varepsilon$, and if $p_i = p_{i-1} = 1$, then $p_{i-2} = 0$, is a prefix of the word $\sigma^{n+1}(1)$. This property holds for $n = 0, 1$. Assume that the hypothesis holds for every integer $k \leq n - 1$. Let $w = \sigma^n(p_n)\sigma^{n-1}(p_{n-1}) \cdots p_0$, with the above mentioned conditions on the "digits" p_i (and in particular $p_n = 1$).

One has

$$\begin{aligned}\sigma^{n+1}(1) &= \sigma^n(1)\sigma^n(2) = \sigma^n(1)\sigma^{n-1}(1)\sigma^{n-1}(3)\\ &= \sigma^n(1)\sigma^{n-1}(1)\sigma^{n-2}(1).\end{aligned}$$

Assume $p_{n-1} = 1$, then one has $p_{n-2} = \varepsilon$. By induction hypothesis the word $\sigma^{n-3}(p_{n-3}) \cdots p_0$ is a prefix of $\sigma^{n-2}(1)$, which implies that w is a prefix of $\sigma^n(1)\sigma^{n-1}(1)\sigma^{n-2}(1)$ and thus of $\sigma^{n+1}(1)$.

Assume now $p_{n-1} = \varepsilon$. Then $\sigma^{n-2}(p_{n-2}) \cdots p_0$ is a prefix of $\sigma^{n-1}(1)$, and w is a prefix of $\sigma^{n+1}(1)$, which ends the proof. ∎

Remark 10.7.3. Such a numeration system on finite factors of the Tribonacci word can similarly be introduced for fixed points of morphisms in the sense of Remark 10.1.4 (see Problem 10.7.3).

10.7.3. Density properties: statistics on letters

Since the Tribonacci morphism is primitive, we know from Section 1.7.2 and 1.8.6 that, by applying the Perron–Frobenius theorem, the letters admit densities in the Tribonacci word and the vector of probabilities of occurrence of letters is equal to the normalized positive (right) eigenvector $v_\beta = (1/\beta, 1/\beta^2, 1/\beta^3)$ associated with the dominant eigenvalue β (let us recall that the incidence matrix is the transpose of the matrix introduced in Section 1.8.6). We give below a direct proof of this result and prove even a stronger result of convergence towards the probabilities of letters.

Proposition 10.7.4. *Each of the letters $1, 2, 3$ admits a density in the Tribonacci word. The probabilities of letters are positive. More precisely, the vector of probabilities $(\pi(1), \pi(2), \pi(3))$ is equal to the normalized positive eigenvector $v_\beta = (1/\beta, 1/\beta^2, 1/\beta^3)$ associated with the dominant eigenvalue β of the incidence matrice of the Tribonacci morphism. Furthermore, there exists $C > 0$ such that*

$$\forall N,\ \big|\, |u_0 u_1 \cdots u_{N-1}|_i - \pi(i) N\big| \leq C.$$

Proof. Let $u_0 u_1 \cdots u_{N-1}$ be a prefix of the Tribonacci word; according to Lemma 10.7.2, let us decompose it as

$$u_0 \cdots u_{N-1} = \sigma^n(p_n)\sigma^{n-1}(p_{n-1}) \cdots p_0,$$

where the finite words p_i are either equal to the empty word ε or to the letter 1, $p_n \neq \varepsilon$, and if $p_k = p_{k-1} = 1$, then $p_{k-2} = 0$. Then for $i = 1, 2, 3$

$$|u_0 \cdots u_{N-1}|_i = < f(u_0 \cdots u_{N-1}), e_i >,$$

where $< >$ denotes the Hermitian scalar product in $\mathbb{C}^3$.

Let us write $e_1 = a_\beta v_\beta + a_\alpha v_\alpha + a_{\overline{\alpha}} v_{\overline{\alpha}}$, where $a_\beta,\ a_\alpha,\ a_{\overline{\alpha}} \in \mathbb{C}$. We have

$$f(\sigma^k(1)) = M_\sigma^k e_1 = a_\beta \beta^k v_\beta + a_\alpha \alpha^k v_\alpha + a_{\overline{\alpha}} \overline{\alpha}^k v_{\overline{\alpha}}.$$

Furthermore,

$$f(u_0 \cdots u_{N-1}) = \sum_{k=0}^{n} f(\sigma^k(p_k)),$$

which implies for $i = 1, 2, 3$

$$\begin{aligned} |u_0 \cdots u_{N-1}|_i = a_\beta (\textstyle\sum_{k=0}^n |p_n| \beta^k) < v_\beta, e_i > + a_\alpha (\sum_{k=0}^n |p_n| \alpha^k) < v_\alpha, e_i > + \\ + a_{\overline{\alpha}} (\textstyle\sum_{k=0}^n |p_n| \overline{\alpha}^k) < v_{\overline{\alpha}}, e_i > . \end{aligned}$$

Let us recall that $|\alpha| < 1$. We have proved that the vectors $f(\sigma^k(1))$ converge exponentially fast to the expanding line, whereas the vectors $f(u_0 \cdots u_{N-1})$ stay within bounded distance of this line (Figure 10.1).

One has

$$\begin{aligned} N &= \textstyle\sum_{i=1,2,3} |u_0 \cdots u_{N-1}|_i \\ &= a_\beta \textstyle\sum_{k=0}^n |p_n| \beta^k + a_\alpha \sum_{k=0}^n |p_n| \alpha^k + a_{\overline{\alpha}} \sum_{k=0}^n |p_n| \overline{\alpha}^k, \end{aligned}$$

since $< v_\beta, e_1 + e_2 + e_3 >=< v_\alpha, e_1 + e_2 + e_3 >=< v_{\overline{\alpha}}, e_1 + e_2 + e_3 >= 1$, according to our conventions of normalization.

Hence there exists $C > 0$ such that

$$\forall i = 1, 2, 3, \ | < f(u_0 \cdots u_{N-1}), e_i > -N < v_\beta, e_i > | \leq C,$$

which implies in particular that $< v_\beta, e_i >= \pi(i) = 1/\beta^i$, $i = 1, 2, 3$. ■

Remark 10.7.5. Proposition 10.7.4 holds more generally for Pisot morphisms (Problem 10.7.4) and is strongly connected to the balance properties of their fixed points (Problem 10.7.5). Let us observe that the statement in Proposition 10.7.4 is stronger than Assertion 5 of the Perron–Frobenius theorem.

10.8. The Rauzy fractal

10.8.1. A discrete approximation of the line

The Tribonacci broken line stays within a bounded distance of the expanding line (Proposition 10.7.4 and Figure 10.1). Let us project its vertices $f(u_0 \cdots u_{N-1})$ along the expanding direction v_β, in order to obtain in particular some information on the quality of approximation of the expanding line by the points $f(\sigma^k(1))$, $k \in \mathbb{N}$. We thus choose here to project onto the plane $x + y + z = 0$; this allows us to express the coordinates of the projected points in the basis $(e_3 - e_1, e_2 - e_1)$ of the plane $x + y + z = 0$ in terms of the convergence towards the probabilities of occurrence of the letters, as explained below (Equation (10.8.1)).

Let π_0 denote the projection in $\mathbb{R}^3$ onto the plane $\mathcal{P}_0$ of equation $x+y+z = 0$ along the expanding line generated by the vector v_β. One has

$$\forall\, P = (x, y, z) \in \mathbb{R}^3, \ \pi_0(P) = (x, y, z) - < (x, y, z), (1, 1, 1) > v_\beta,$$

that is,

$$\pi_0(P) = (\frac{1}{\beta}(x + y + z) - x)(e_3 - e_1) + (\frac{1}{\beta^2}(x + y + z) - y)(e_3 - e_2).$$

In particular, if $P = f(u_0 \cdots u_{N-1})$, for some $N \in \mathbb{N}$, then

$$\pi_0(P) = (\frac{N}{\beta} - |u_0 \cdots u_{N-1}|_1)(e_3 - e_1) + (\frac{N}{\beta^2} - |u_0 \cdots u_{N-1}|_2)(e_3 - e_2). \quad (10.8.1)$$

We define the set $\mathcal{R}$ as the closure of the projections of the vertices of the Tribonacci broken line:

$$\mathcal{R} := \overline{\{\pi_0(f(u_0 \ldots u_{N-1})); \ N \in \mathbb{N}\}},$$

where $u_0 \ldots u_{N-1}$ stands for the empty word when $N = 0$. The set $\mathcal{R}$ is called the *Rauzy fractal* associated with the Tribonacci morphism σ (see Figure 10.2).

We now introduce a lattice in the plane $\mathcal{P}_0$ which will play a key rôle in the following. Let $\mathcal{L}_0 := \mathbb{Z}^3 \cap \mathcal{P}_0$; $\mathcal{L}_0$ is equal to the lattice $\mathbb{Z}(e_3 - e_1) + \mathbb{Z}(e_3 - e_2)$.

Proposition 10.8.1. *The set $\mathcal{R}$ is compact. The translates of the Rauzy fractal by the vectors of the lattice $\mathcal{L}_0$ cover the contracting plane $\mathcal{P}_0$, that is,*

$$\cup_{\gamma\in\mathcal{L}_0}(\mathcal{R}+\gamma) = \mathcal{P}_0. \tag{10.8.2}$$

The interior of R is not empty.

Proof. We first deduce from (10.8.1) and Proposition 10.7.4 that the Rauzy fractal is bounded, and hence compact.

We then need the following lemma to prove that one has a covering of the plane $\mathcal{P}_0$ by the translates of the Rauzy fractal.

Lemma 10.8.2. *The translates along the lattice $\mathcal{L}_0$ of the vertices of the broken line $f(u_0u_1\dots u_{N-1})$, $N\in\mathbb{N}$, cover the following upper half space:*

$$\{f(u_0u_1\dots u_{N-1})+\gamma;\ N\in\mathbb{N},\ \gamma\in\mathcal{L}_0\} = \{(x,y,z)\in\mathbb{Z}^3;\ x+y+z\geq 0\}.$$

Let $(x,y,z)\in\mathbb{Z}^3$ with $x+y+z\geq 0$; let $N=x+y+z$; one has $N = |u_0u_1\dots u_{N-1}|_1+|u_0u_1\dots u_{N-1}|_2+|u_0u_1\dots u_{N-1}|_3$. Let

$$\gamma = (x-|u_0u_1\dots u_{N-1}|_1, y-|u_0u_1\dots u_{N-1}|_2, z-|u_0u_1\dots u_{N-1}|_3);$$

then $\gamma\in\mathbb{Z}(e_1-e_3)+\mathbb{Z}(e_2-e_3)=\mathcal{L}_0$. ■

Let us end the proof of Proposition 10.8.1. We need the following theorem known as Kronecker's theorem that we recall here without a proof (a proof of this theorem can be found for instance in Cassels 1957).

Theorem 10.8.3 (Kronecker's theorem). *Let $r\geq 1$ and let $\alpha_1,\cdots,\alpha_r$ be r real numbers such that $1,\alpha_1,\cdots,\alpha_r$ are rationally independent. For every $\eta>0$ and for every $(x_1,\cdots,x_r)\in\mathbb{R}^r$, there exist $N\in\mathbb{N}$, $(p_1,\cdots,p_r)\in\mathbb{Z}^r$ such that*

$$\forall i=1,\cdots,r,\ |N\alpha_i-p_i-x_i|<\eta.$$

Let us apply Kronecker's theorem to $1,\frac{1}{\beta},\frac{1}{\beta^2}$ (which are rationally independent). Let us fix $\eta>0$ and let P be given in $\mathcal{P}_0$ with coordinates (x,y) say, in the basis (e_3-e_1,e_3-e_2). There exist $p,q\in\mathbb{Z}$, $N\in\mathbb{N}$ such that $|N\frac{1}{\beta}-p-x|<\eta$ and $|N\frac{1}{\beta^2}-q-y|<\eta$. Take $r=N-(p+q)$. Then the coordinates in the basis (e_3-e_1,e_3-e_2) of $\pi_0(p,q,r)$ and P differ by at most η. We thus have proved that $\pi_0(\{(p,q,r)\in\mathbb{Z}^3,\ p+q+r\geq 0\})$ is dense in $\mathcal{P}_0$. Consequently, given any point P of $\mathcal{P}_0$, there exists a sequence of points $(\pi_0(f(u_0u_1\dots u_{N_k-1}))+\gamma_k)_k$ with γ_k in the lattice $\mathcal{L}_0$ which converges to P in $\mathcal{P}_0$. Since $\mathcal{R}$ is bounded, there are infinitely many k for which the points γ_k of the lattice $\mathcal{L}_0$ take the same value, say γ; we thus get $P\in\mathcal{R}+\gamma$, which implies (10.8.2). Since $\mathcal{L}_0$ is countable, we deduce from Baire's theorem that the interior of $\mathcal{R}$ is not empty. ■

Remark 10.8.4. In fact, we have more than a covering by translates of the Rauzy fractal. We have in fact a periodic tiling of the plane up to sets of zero Lebesgue measure, that is, the union in (10.8.2) is disjoint up to sets of zero measure, as illustrated in Figure 10.8.3. We prove it in Section 10.8.5.

10.8.2. Arithmetic expression

In order to study more carefully the topological properties of the Rauzy fractal, which is the aim of Section 10.8.4, we introduce some more notation to express the coordinates of the vectors $f(u_0 \cdots u_{N-1})$ in the basis $(e_3 - e_1, e_3 - e_2)$ of the plane $\mathcal{P}_0$.

Let $\delta : \mathbb{N} \to \mathbb{R}^2$, $N \mapsto \delta(N)$, where $\delta(N)$ denotes the vector of coordinates of $\pi_0(f(u_0 u_1 \cdots u_{N-1}))$ in the basis $(e_3 - e_1, e_3 - e_2)$ of the plane $x + y + z = 0$. One has according to (10.8.1), for $N \in \mathbb{N}$,

$$\delta(N) = N \cdot (1/\beta, 1/\beta^2) - (|u_0 u_1 \cdots u_{N-1}|_1, |u_0 u_1 \cdots u_{N-1}|_2). \qquad (10.8.3)$$

Let $B = \begin{bmatrix} -1/\beta & -1/\beta \\ 1 - (1/\beta)^2 & -(1/\beta)^2 \end{bmatrix}$. One easily checks that for every word $w \in \{1, 2, 3\}^*$, then the vector of coordinates of $\pi_0(f(\sigma(w)))$ in the basis $(e_3 - e_1, e_3 - e_2)$ is equal to the matrix B applied to the vector of coordinates of $\pi_0(f(w))$ in the same basis. We thus get that if N has for normal T-representation, $N = \sum_{i=0}^{n} \varepsilon_i T_i$, that is, $u_0 \cdots u_{N-1} = \sigma^n(p_n)\sigma^{n-1}(p_{n-1}) \cdots p_0$, with $|p_i| = \varepsilon_i$, then

$$\delta(N) = \sum_{i=0}^{k} \varepsilon_i B^i z, \text{ where we set } z = \delta(1) = (1/\beta - 1, 1/\beta^2).$$

The eigenvalues of the matrix B are of modulus smaller than 1, hence the series $\sum_{i=0}^{\infty} \varepsilon_i B^i z$ are convergent in $\mathbb{R}^2$. The following proposition is thus an immediate consequence of this:

Proposition 10.8.5. *The Rauzy fractal is the set of points of the plane $\mathcal{P}_0$ with coordinates in the basis $(e_3 - e_1, e_3 - e_2)$ in*

$$R := \{\sum_{i=0}^{\infty} \varepsilon_i B^i z;\ (\varepsilon_i)_{i \geq 0} \in \{0, 1\}^{\omega}, \forall i\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0\}.$$

Remark 10.8.6. We will mostly study the set R to deduce topological properties of the Rauzy fractal $\mathcal{R}$; indeed both sets are by definition in one-to-one correspondence, this bijection being the restriction of a topological isomorphism. Let us observe that similarly, the Rauzy fractal and

$$\{\sum_{i=0}^{\infty} \varepsilon_i \alpha^i \in \mathbb{C};\ (\varepsilon_i)_{i \geq 0} \in \{0, 1\}^{\omega}, \forall i\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0\}$$

are also easily seen to be in one-to-one correspondence. Indeed the matrix B admits as characteristic polynomial $(X-\alpha)(X-\overline{\alpha})$, and it is thus similar in $\mathbb{C}^2$ to the matrix $\begin{bmatrix} \alpha & 0 \\ 0 & \overline{\alpha} \end{bmatrix}$.

10.8.3. An exchange of pieces

Let us introduce the following division of the Rauzy fractal into three sets according to which letter was lastly read before projecting. For $i \in \{1,2,3\}$ let

$$\mathcal{R}_i = \overline{\{\pi_0(f(u_0 \dots u_{N-1}));\ N \in \mathbb{N},\ u_N = i\}}.$$

We similarly define the subsets R_i of $\mathbb{R}^2$, $i = 1,2,3$, as, respectively, the sets of coordinates of elements of $\mathcal{R}_i$ (in the basis $(e_3 - e_1, e_3 - e_2)$).

Lemma 10.8.7. *One has*

$$\begin{aligned}
R_1 &= \left\{ \textstyle\sum_{i \geq 0} \varepsilon_i B^i z;\ \forall i,\ \varepsilon_i \in \{0,1\};\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0;\ \varepsilon_0 = 0 \right\}, \\
R_2 &= \left\{ \textstyle\sum_{i \geq 0} \varepsilon_i B^i z;\ \forall i,\ \varepsilon_i \in \{0,1\};\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0;\ \varepsilon_0 \varepsilon_1 = 10 \right\}, \\
R_3 &= \left\{ \textstyle\sum_{i \geq 0} \varepsilon_i B^i z;\ \forall i,\ \varepsilon_i \in \{0,1\};\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0;\ \varepsilon_0 \varepsilon_1 = 11 \right\},
\end{aligned}$$

and

$$R_1 = BR,\ R_2 = z + B^2 R,\ R_3 = z + Bz + B^3 R,$$

that is,

$$R_1 = B(R_1 + R_2 + R_3),\ R_2 = z + BR_1,\ R_3 = z + BR_2.$$

Proof. It is sufficient to check that if $u_0 \cdots u_{N-1}$ admits for normal Tribonacci representation $\sigma^n(p_n) \cdots \sigma^0(p_0)$, then

$$\begin{cases} p_0 = \varepsilon \text{ implies } u_N = 1, \\ p_0 = 1,\ p_1 = \varepsilon \text{ implies } u_N = 2, \\ p_0 = 1,\ p_1 = 1 \text{ implies } u_N = 3. \end{cases}$$

- Assume that $p_0 = \varepsilon$. Then $u_0 \cdots u_{N-1} = \sigma^n(p_n) \cdots \sigma(p_1)$, and $u_0 \cdots u_N = \sigma^n(p_n) \cdots \sigma(p_1) u_N$. Hence u_N needs to be equal to 1, since the images of letters under σ begin with 1, and u is fixed under σ.
- Assume that $p_0 = 1$ and $p_1 = \varepsilon$. One has $u_0 \cdots u_{N-1} = \sigma^n(p_n) \cdots \sigma^2(p_2) 1$. The word $\sigma^n(p_n) \cdots \sigma^2(p_2)\sigma(1)$ has length $N+1$. If either p_2 or p_3 equals ε, then this expansion is a normal Tribonacci representation, and thus a prefix of the Tribonacci word (according to Lemma 10.7.2), which gives $u_N = 2$. Otherwise it can also be represented as $\sigma^n(p_n) \cdots \sigma^2(1) = \sigma^n(p_n) \cdots \sigma^4(1)$. One shows by induction that the last term of the normal Tribonacci representation of this expansion is of the form $\sigma^{3k+1}(1)$, which admits as last letter 2.

- Assume that $p_0 = 1$ and $p_1 = 1$, and thus $p_2 = \varepsilon$. Then $u_0 \cdots u_{N-1} = \sigma^n(p_n) \cdots \sigma^3(p_3)\sigma(1)1$. The word $\sigma^n(p_n) \cdots \sigma^2(1)$ has length $N+1$. If either p_3 or p_4 equals ε, then this expansion is a normal Tribonacci representation, and thus a prefix of the Tribonacci word (according to Lemma 10.7.2), which gives $u_N = 3$. Otherwise it can also be represented as $\sigma^n(p_n) \cdots \sigma^2(1) = \sigma^n(p_n) \cdots \sigma^5(1)$. One shows by induction that the last term of the normal Tribonacci representation of this expansion is of the form $\sigma^{3k+2}(1)$, which admits as last letter 3. ■

The sets $\mathcal{R}_i$, $i = 1, 2, 3$ are represented in Figure 10.2. Figure 10.8.3 illustrates Lemma 10.8.8 below, that is, one can reorganize the division of $\mathcal{R}$ into these three pieces up to translations.

Lemma 10.8.8. *The following exchange of pieces E is well-defined*

$$E : \text{Int } \mathcal{R}_1 \cup \text{Int } \mathcal{R}_2 \cup \text{Int } \mathcal{R}_3 \to \mathcal{R}, \; x \mapsto x + \pi_0(e_i), \text{ when } x \in \text{Int } \mathcal{R}_i.$$

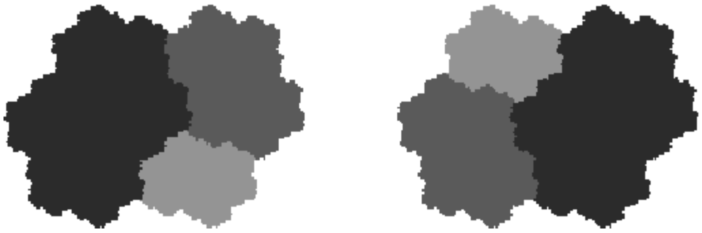


Figure 10.3. The exchange map E.

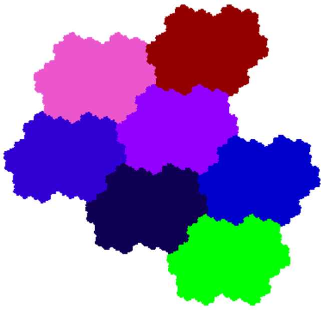


Figure 10.4. A piece of a periodic tiling by the Rauzy fractal.

Proof. Let us first prove that the sets $\mathcal{R}_i$, for $i = 1, 2, 3$, are two-by-two disjoint in measure, and hence that their interiors Int $\mathcal{R}_i$ are two-by-two disjoint.

Since $\mathcal{R}$ is compact, then it is measurable for the Lebesgue measure and its Lebesgue measure $\mu(\mathcal{R})$ is finite and nonzero since its interior is not empty according to Proposition 10.8.1.

One has $\mu(\mathcal{R}) \leq \sum_{i=1}^{3} \mu(\mathcal{R}_i)$. Since the determinant of the matrix B equals $1/\beta$, then according to Lemma 10.8.7

$$\mu(\mathcal{R}_1) = 1/\beta\ \mu(\mathcal{R}),\ \mu(\mathcal{R}_2) = (1/\beta)^2\mu(\mathcal{R}),\ \mu(\mathcal{R}_3) = (1/\beta)^3\mu(\mathcal{R}).$$

Hence one gets $\mu(\mathcal{R}) = \sum_{i=1}^{3} \mu(\mathcal{R}_i)$. This implies in particular that $\mu(\mathcal{R}_i \cap \mathcal{R}_j) = 0$ for $i \neq j$. The same holds for their interiors Int $\mathcal{R}_i$, that is, $\mu(\text{Int } \mathcal{R}_i \cap \text{Int } \mathcal{R}_j) = 0$ for $i \neq j$, which implies that they are two-by-two disjoint.

One easily sees that for $i = 1, 2, 3$, $\mathcal{R}_i + \pi_0(e_i) = \overline{\{\pi(f(u_0 \ldots u_N));\ u_N = i\}}$, which implies $\mathcal{R}_i + \pi_0(e_i) \subset \mathcal{R}$. We thus deduce that the map E is well-defined. ■

Remark 10.8.9. The sets $\mathcal{R}_i$, $i = 1, 2, 3$ are not disjoint. Indeed a vector with coordinates in the basis $(e_3 - e_1, e_3 - e_2)$ having several expansions as $\sum_{i=0}^{\infty} \varepsilon_i B^i z$, (with $(\varepsilon_i)_{i \geq 0} \in \{0,1\}^\omega$ and $\forall i,\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0$) can belong simultaneously to several of these sets. This is the case in particular of the vector with coordinates $\sum_{i=1}^{\infty} B^{3i} z$. Since $B^3 = B^2 + B + 1$, then $\sum_{i=1}^{\infty} B^{3i} z = \sum_{i=0}^{\infty} B^i z$, and it admits the following three admissible expansions

$$\sum_{i=1}^{\infty} B^{3i} z = z + \sum_{i=0}^{\infty} B^{3i+1} z = Z + Bz + \sum_{i=0}^{\infty} B^{3i+2} z.$$

10.8.4. Some topological properties

We need now to introduce a suitable norm on $\mathbb{R}^2$ associated with the matrix B that will be crucial for the statement of the first topological properties of the Rauzy fractal, from which the arithmetic properties of Section 10.9 will be deduced.

Let us recall that the matrix B is similar in $\mathbb{C}^2$ to the matrix $\begin{bmatrix} \alpha & 0 \\ 0 & \overline{\alpha} \end{bmatrix}$. Let

$$M = \begin{bmatrix} \overline{\alpha} + 1/\beta & 1/\beta \\ -(\alpha + 1/\beta) & -1/\beta \end{bmatrix}.$$

One easily checks that $MBM^{-1} = \begin{bmatrix} \alpha & 0 \\ 0 & \overline{\alpha} \end{bmatrix}$.

The *Rauzy norm* $||\,||$ is defined for $x \in \mathbb{R}^2$ as the Euclidean norm of Mx. Hence, for every $x \in \mathbb{R}^2$

$$||Bx|| = |\alpha|||x|| = \sqrt{1/\beta}||x||.$$

We denote by $|||\ |||$ the distance to the nearest point with integer coordinates. One checks that

$$||z|| = ||\delta(1)| = |\alpha|^4 = 1/\beta^2 \text{ and } ||\delta(T_n)|| = |\alpha|^n ||z|| = |\alpha|^{n+4}.$$

We will mainly work in this section with the set of coordinates R, rather than with $\mathcal{R}$ itself. The following lemma states that if one takes an element in R of sufficiently small norm which is equal modulo $\mathbb{Z}^2$ to the coordinates $\delta(N)$ of $\pi_0(f(u_0 \cdots u_{N-1}))$, then it has to be exactly equal to $\delta(N)$. The proof is based on the fact that the set R is contained in the square $\{(x, y) \in \mathbb{R}^2;\ |x|, |y| < 1\}$ located at the origin. In particular, 0 is the only element with integer coordinates contained in R. This lemma is fundamental and is a first step toward the fact that if two points of R differ by a vector with integer coordinates, then these two points do coincide.

Lemma 10.8.10. *There exists $C > 0$ such that*

$$\forall N \geq 1,\ \forall v \in \mathbb{Z}^2,\ ||N \cdot (1/\beta, 1/\beta^2) - v|| < C \Longrightarrow v = N \cdot (1/\beta, 1/\beta^2) - \delta(N).$$

Remark 10.8.11. This lemma implies in particular that if the norm of $\delta(N)$ is smaller than C, then $(|u_0 \cdots u_{N-1}|_1, u_0 \cdots u_{N-1}|_2)$ is the nearest point with integer coordinates to $N \cdot (1/\beta, 1/\beta^2)$. For instance, for n large enough,

$$|||T_n \cdot (1/\beta, 1/\beta^2)||| = ||\delta(T_n)||,$$

since $||\delta(T_n)|| = |\alpha|^{n+4} < C$. In other words, the projections of the points $f(\sigma^n(1))$ approximate very well the points with coordinates $T_n \cdot (1/\beta, 1/\beta^2)$.

Proof. Let $N \geq 1$ with normal T-representation $N = \sum_{i=0}^{k} \varepsilon_i T_i$. One can write $\delta(N) = \sum_{i=0}^{k} \varepsilon_i B^i z$ as $\delta(N) = \sum_{i \geq 0} \varepsilon_{3i} B^{3i} y_i$, where y_i belongs to the following set F:

$$F := \{0, z, Bz, B^2 z, z + Bz, z + B^2 z, Bz + B^2 z\}.$$

Hence

$$||\delta(N)|| \leq \sum_{i \geq 0} |\alpha|^{3i} \max_{y \in F} ||y||.$$

One checks that $\max_{y \in F} ||y|| = ||z|| = 1/\beta^2$. We thus get

$$||\delta(N)|| \leq \frac{1}{\beta^2(1 - |\alpha^3|)} < 1/2. \tag{10.8.4}$$

One also checks that the set of points $x \in \mathbb{R}^2$ such that $||x|| < 0,53$ is a domain delimited by an ellipse strictly included in the square $\{(x, y) \in \mathbb{R}^2;\ |x|, |y| < 1\}$. Hence R is also included in this square, following (10.8.4).

Let $v \in \mathbb{Z}^2$. Take $C = 0,03$ for instance. Let $N \geq 1$ such that

$$||N \cdot (1/\beta, 1/\beta^2) - v|| < C.$$

Hence according to (10.8.3)

$$||(|u_0 \cdots u_{N-1}|_0, |u_0 \cdots u_{N-1}|_1) - v|| \leq ||\delta(N)|| + ||N \cdot (1/\beta, 1/\beta^2) - v|| < 0,53,$$

which implies that $(|u_0 \cdots u_{N-1}|_0, |u_0 \cdots u_{N-1}|_1) - v$ belongs to the square $\{(x, y) \in \mathbb{R}^2;\ |x|, |y| < 1\}$ and thus $v = (|u_0 \cdots u_{N-1}|_0, |u_0 \cdots u_{N-1}|_1)$, since both vectors have integer coordinates. ■

Proposition 10.8.12. *The point 0 belongs to the interior of the Rauzy fractal $\mathcal{R}$. Furthermore, for all $N \in \mathbb{N}$, $\delta(N)$ belongs to the interior of R_{u_N}. Consequently, the Rauzy fractal is the closure of its interior.*

Proof. Let us prove that 0 is an interior point of the set R. Let C be the constant of Lemma 10.8.13. The sequence $(N \cdot (1/\beta, 1/\beta^2))_{N \geq 0}$ is dense in $\mathbb{R}^2$ modulo $\mathbb{Z}^2$ by Kronecker's theorem (Theorem 10.8.3), since $1, 1/\beta, 1/\beta^2$ are linearly independent over $\mathbb{Q}$. In particular, it is dense in the set $\{x \in \mathbb{R}^2;\ ||x|| < C\}$. This implies, according to Lemma 10.8.10, that the points $\delta(N)$ are also dense in this same set. Hence $\{x \in \mathbb{R}^2;\ ||x|| < C\}$ is included in the closure R of $\{\delta(N);\ N \in \mathbb{N}\}$. This proves that 0 is an interior point.

One easily deduces that for every $N \in \mathbb{N}$, $\delta(N)$ belongs to the interior of R_{u_N}. Indeed let us consider a given N with normal T-representation $\sum_{i=0}^{k} \varepsilon_i T_i$; by definition, $\delta(N) \in R_{u_N}$; for any $(\varepsilon_i)_{i \geq k+2} \in \{0,1\}^\omega$, with the admissibility condition that no three consecutive 1's occur in this sequence, then $\delta(N) + \sum_{i \geq k+2} \varepsilon_l T_l \in R_{u_N}$, which implies that $\delta(N) + B^{k+2} R$ is still included in R_{u_N}, and thus $\delta(N)$ belongs to the interior of R_{u_N}, since 0 belongs to the interior of R. This easily implies that $\mathcal{R}$ is the closure of its interior. ∎

One can even get more information on the first coefficients of N in its normal T-representation if the distance between $N \cdot (1/\beta, 1/\beta^2)$ and $\mathbb{Z}^2$ is small enough; this provides some knowledge on the repartition of the sequence $(N \cdot (1/\beta, 1/\beta^2))_{N \geq 0}$.

Lemma 10.8.13. *Let $N \geq 1$ with normal T-representation $N = \sum_{i \geq 0} \varepsilon_i T_i$. Then*

$$\forall v \in \mathbb{Z}^2,\ \forall m \in \mathbb{N},\ \Big(||N \cdot (1/\beta, 1/\beta^2) - v|| < C\beta^{-m/2} \Rightarrow \forall i < m,\ \varepsilon_i = 0 \Big).$$

Proof. Let $N \geq 1$ with normal T-representation $N = \sum_{i \geq 0} \varepsilon_i T_i$ and let $v \in \mathbb{Z}^2$ such that there exists $m \geq 1$ with $||N \cdot (1/\beta, 1/\beta^2) - v|| < C\beta^{-m/2}$. Since $||N \cdot (1/\beta, 1/\beta^2) - v|| < C$, and according to Lemma 10.8.10, then $\delta(N) = N \cdot (1/\beta, 1/\beta^2) - v$. Furthermore one has $B^{-m}\delta(N) \in R$; indeed $||B^{-m}\delta(N)|| = \beta^{m/2} ||\delta(N)|| < C$, and we have seen in the proof of Proposition 10.8.12 that $\{x;\ ||x|| < C\}$ is included in R.

It remains to prove that if N satisfies $\delta(N) \in B^m R$, then its normal T-representation verifies $N = \sum_{i \geq m} \varepsilon_i$. For that purpose, we introduce the following notation in order to refine the partition of R into the three pieces R_i, $i = 1, 2, 3$. Let us consider the three following three maps $\psi_i : \mathbb{R}^2 \to \mathbb{R}^2$, $i = 1, 2, 3$, as follows (recall that $z = \delta(1)$):

$$\psi_1 : v \mapsto Bv,\ \psi_2 : v \mapsto z + B^2 v,\ \psi_3 : v \mapsto z + Bz + B^3 v.$$

For $a_1 \cdots a_r \in \{1,2,3\}^r$, let $R_{a_1 \cdots a_r} = \psi_{a_0} \circ \cdots \circ \psi_{a_r}(R)$. Let us observe that $R_{a_1 \cdots a_r} \subset R$.

One proves by induction that for all N, and for all r, there exists $a_1 \cdots a_r$ such that $\delta(N)$ belongs to $R_{a_1 \cdots a_r}$. Indeed, let $v = \sum_{i=0}^{k} \varepsilon_i B^i z \in R$; if $v \in R_i$, then there exists $w \in R$ such that $v = \psi_i(w)$. Furthermore, the same argument as in the proof of Proposition 10.8.12 implies that $\delta(N)$ belongs to the interior of $R_{a_1 \cdots a_r}$.

Let us prove by induction on r that the interiors of the sets $R_{a_1 \cdots a_r}$ are two-by-two disjoint. The induction property holds for $r = 1$ according to Lemma 10.8.8. Assume it is true for $k \leq r$, with $r \geq 1$. Let $a_1 \cdots a_r \in \{1,2,3\}^r$. One has $\mu(R_{a_1 \cdots a_r i}) = (1/\beta)^i \mu(R_{a_1 \cdots a_r})$, which implies similarly as in the proof of Lemma 10.8.8 that the interiors of the sets $R_{a_1 \cdots a_r i}$, $i = 1,2,3$, are two-by-two disjoint in measure, as well as the interiors of the sets $R_{a_1 \cdots a_r i}$, for $i = 1,2,3$, and $a_1 \cdots a_r \in \{1,2,3\}^r$.

Hence for every N, and for every r, there exists a unique $a_1 \cdots a_r$ such that $\delta(N)$ belongs to the interior of $R_{a_1 \cdots a_r}$. Furthermore it is easily seen that if there exists k such that $a_k \neq 1$, then there exists a coefficient ε_i equal to 1, with $i < r$, in the normal T-representation of N. This implies that if $\delta(N) \in B^m R = \psi_1^m(R)$, then all the coefficients ε_i for $i < m$ are equal to 0 in its normal T-representation. ∎

10.8.5. Tiling and Tribonacci translation

We are now able to prove that the covering of the plane $\mathcal{P}_0$ stated in Proposition 10.8.1, that is, $\cup_{\gamma \in \mathcal{L}_0} \mathcal{R} + \gamma$, is in fact a periodic tiling (up to sets of zero measure).

Lemma 10.8.14. *The sets Int $\mathcal{R} + \gamma$, for $\gamma \in \mathcal{L}_0$, are disjoint, that is,*

$$\text{if } x, y \in \text{Int } R, \text{ with } x - y \in \mathbb{Z}^2, \text{ then } x = y.$$

Proof. Let $x, y \in \text{Int } R$ with $x - y \in \mathbb{Z}^2$. By density of the sequence $(\delta(N))_{N \geq 0}$, there exists a point $\delta(M)$ close enough to x so that $\delta(M) + y - x$ is close enough to y, and thus, still belongs to R.

Let us choose an integer m large enough so that the coefficients ε_i in the normal T-representation of M are equal to 0 for $i \geq m$. One gets $M = \sum_{i=0}^m \varepsilon_i T_i$. By density of the sequence $(\delta(N))_N$, there exists $N > M$ such that

$$||\delta(N) - (\delta(M) + y - x)|| < C \left(\frac{1}{\beta}\right)^{(m+2)/2}.$$

There exists $h \in \mathbb{Z}^2$ such that $\delta(N) - (\delta(M) + y - x) = (N - M) \cdot (1/\beta, 1/\beta^2) - h$. We thus can apply Lemma 10.8.13, and get that the normal T-representation $\sum_{i=0}^k \varepsilon_i' T_i$ of $N - M$ satisfies $\varepsilon_i' = 0$ for $i \leq m + 1$. This implies that N admits as normal T-representation $\sum_{i=0}^m \varepsilon_i T_i + \sum_{i \geq m+2} \varepsilon_i' T_i$, and hence $\delta(N) - \delta(M) = \delta(N - M)$. Since $||(N - M) \cdot (1/\beta, 1/\beta^2) - h|| < C \left(\frac{1}{\beta}\right)^{(m+2)/2} < C$, it follows from Lemma 10.8.10 that

$$\delta(N - M) = (N - M) \cdot (1/\beta, 1/\beta^2) - h = \delta(N) - \delta(M) = \delta(N) - \delta(M) + x - y,$$

which implies $y = x$. ∎

Remark 10.8.15. The domain R is thus a fundamental domain of the torus $\mathbb{T}^2 = \mathbb{R}^2/\mathbb{Z}^2$, that is,

$$\mathbb{R}^2 = \cup_{v\in\mathbb{Z}^2} R + v,\ \mathcal{P}_0 = \cup_{\gamma\in\mathbb{Z}^2} \mathcal{R} + \gamma,$$

both unions being disjoint up to sets of zero measure.

This tiling property has the following arithmetic formulation: the translation by $(1/\beta, 1/\beta^2)$ in $\mathbb{R}^2/\mathbb{Z}^2 = \mathbb{T}^2$, which is the quotient map of the exchange map E defined in Lemma 10.8.8 with respect to the lattice $\mathcal{L}_0$, is coded by the Tribonacci word:

Theorem 10.8.16. *The Tribonacci word codes the orbit of the point* 0 *under the action of the translation*

$$R_\beta : \mathbb{T}^2 \to \mathbb{T}^2,\ x \mapsto x + (1/\beta, 1/\beta^2)$$

with respect to the partition of the fundamental domain R of $\mathbb{T}^2$ by the sets (R_1, R_2, R_3), that is,

$$\forall N \in \mathbb{N},\ \forall i = 1, 2, 3,\ u_N = i \Longleftrightarrow R_\beta^N(0) \in R_i.$$

Proof. According to Proposition 10.8.12, for every N, there exists $i = 1, 2, 3$ such that $\delta(N)$ belongs to the interior of R_i; hence $R_\beta^N(0)$ (which is congruent modulo $\mathbb{Z}^2$ to $\delta(N)$) also belongs modulo $\mathbb{Z}^2$ to R_i. Furthermore, such an integer i is unique according to Lemma 10.8.14. This implies that the coding of the orbit of 0 under R_β is well-defined.

Let E be the exchange of pieces introduced in Lemma 10.8.8. Let us prove by induction on N that $E^N(0) = \pi_0(f(u_0 \cdots u_{N-1}))$. The induction property holds for $N = 0$. Suppose that the induction property holds for N. One has $\pi_0(f(u_0 \cdots u_{N-1})) \in \text{ Int } \mathcal{R}_{u_N}$. Hence $E^{N+1}(0) = E(\pi_0(f(u_0 \cdots u_{N-1}))) = \pi_0(f(u_0 \cdots u_{N-1})) + \pi_0(e_{u_N}) = \pi_0(f(u_0 \cdots u_N))$, which ends the induction proof.

One thus deduces that for all $N \in \mathbb{N}$, for all $i = 1, 2, 3$, $E^N(0) = \pi_0(f(u_0 \cdots u_{N-1})) \in \mathcal{R}_i$ if and only if $u_N = i$. In other words, we have proved that the Tribonacci word codes the orbit of 0 under the action of the map E with respect to the partition $(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3)$, that is,

$$\forall N \in \mathbb{N},\ \forall i = 1, 2, 3,\ u_N = i \Longleftrightarrow E^N(0) = i.$$

It remains to check that for all $N \in \mathbb{N}$, for all $i = 1, 2, 3$, $E^N(0) \in \mathcal{R}_i$ if and only if $R_\beta^N(0) \in R_i$. By definition, the coordinates of $E^N(0)$ in the basis $(e_3 - e_1, e_3 - e_2)$ are equal to $\delta(N)$, which is congruent to $R_\beta^N(0)$ modulo $\mathbb{Z}^2$, which ends the proof. ■

10.8.6. A cut and project scheme

The aim of this section is to reformulate the previous reults in terms of "cut and project scheme": Theorem 10.8.18 below states that the vertices of the broken line are exactly the points of $\mathbb{Z}^3$ selected by shifting the Rauzy fractal (considered as an "acceptance window"), along the eigendirection v_β.

A *cut and project scheme* consists of a direct product $\mathbb{R}^k \times H$, $k \geq 1$, where H is a locally compact Abelian group, and a lattice D in $\mathbb{R}^k \times H$, such that with respect to the natural projections $p_0 : \mathbb{R}^k \times H \to H$ and $p_1 : \mathbb{R}^k \times H \to \mathbb{R}^k$:

1. $p_0(D)$ is dense in H;
2. p_1 restricted to D is one-to-one onto its image $p_1(D)$.

This cut and project scheme is denoted $(\mathbb{R}^k \times H, D)$.

A subset Γ of $\mathbb{R}^k$ is a *model set* if there exists a cut and project scheme $(\mathbb{R}^k \times H, D)$ and a relatively compact set (i.e., a set such that its closure is compact) Ω of H with nonempty interior such that

$$\Gamma = \{p_1(P);\ P \in D,\ p_0(P) \in \Omega\}.$$

The set Γ is called the *acceptance window* of the cut and project scheme.

A *Meyer set* S is a subset of some model set of $\mathbb{R}^k$, for some $k \geq 1$, which is *relatively dense*, that is, there exists $R > 0$ such that for all $P \in \mathbb{R}^k$, there exists $M \in S$ such that the ball of radius R located at P contains M.

Remark 10.8.17. The locally abelian compact group which usually occur in the previous definition are either Euclidean or p-adic spaces.

Let π_1 denote the projection in $\mathbb{R}^3$ on the expanding line generated by v_β along the plane $\mathcal{P}_0$. Let us recall that π_0 denotes the projection on the plane $\mathcal{P}_0$ along the expanding line.

Theorem 10.8.18. *The subset $\pi_1(\{f(u_0 \cdots u_{N-1});\ N \in \mathbb{N}\})$ of the expanding eigenline obtained by projecting under π_1 the vertices of the Tribonacci broken line is a Meyer set associated with the cut and project scheme $(\mathbb{R} \times \mathbb{R}^2, \mathbb{Z}^3)$, with acceptance window the interior of the set R of coordinates of the Rauzy fractal. In other words,*

$$\{f(u_0 \cdots u_{N-1});\ N \in \mathbb{N}\} = \{P = (x, y, z) \in \mathbb{Z}^3;\ x + y + z \geq 0; \pi_0(P) \in Int\mathcal{R}\}. \tag{10.8.5}$$

Proof. Let $H = \mathbb{R}^2$, $D = \mathbb{Z}^3$, $k = 1$. The set $H = \mathbb{R}^2$ is in one-to-one correspondence with the plane $\mathcal{P}_0$, whereas $\mathbb{R}$ is in one-to-one correspondence with the expanding eigenline. Up to these two bijections, the natural projections become respectively π_0 and π_1 and are easily seen to satisfy the required conditions (the density has been proved in the proof of Proposition 10.8.1). It remains to prove (10.8.5) to conclude.

According to Lemma 10.8.12, for every N, $\pi_0(f(u_0 \cdots u_{N-1})) \in Int\ \mathcal{R}$. Conversely, let $P = (x, y, z) \in \mathbb{Z}^3$ with $x + y + z \geq 0$ such that $\pi_0(P) \in Int\ \mathcal{R}$. Let $N = x + y + z$. According to Lemma 10.8.2, there exists $\gamma \in \mathcal{L}_0$ such that $P = f(u_0 \cdots u_{N-1}) + \gamma$. Since $\pi_0(P) = \pi_0(f(u_0 \cdots u_{N-1})) + \pi_0(\gamma) = \pi_0(f(u_0 \cdots u_{N-1})) + \gamma$, one gets $P = f(u_0 \cdots u_{N-1})$, following Lemma 10.8.14. ■

Remark 10.8.19. Cut and project schemes are used to modelize quasicrystals and to generate aperiodic tilings, as illustrated in Problem 10.8.6.

10.9. An application to simultaneous approximation

We end this chapter with a section devoted to the study of some Diophantine approximation properties of the vector of translation $(1/\beta, 1/\beta^2)$ of the Tribonacci translation. In particular, the sequence of Tribonacci numbers is shown to be the sequence of best approximations of this vector for the Rauzy norm. Indeed, the vertices of the broken line of the form $f(\sigma^n(1))$, $n \in \mathbb{N}$, provide (after projection) very good approximations of the vector $(1/\beta, 1/\beta^2)$, and even, the best approximations for the Rauzy norm.

Let v be vector and $|| \, ||_0$ a norm in $\mathbb{R}^2$. The increasing sequence of positive integers (q_n) is said to be the sequence of *best approximations* of the vector v for the norm $|| \, ||_0$ if there exists a sequence of vectors (v_n) such that for each integer n and for every $w \in \mathbb{Z}^2$

$$||q_{n+1}v - v_{n+1}||_0 < ||q_n v - w||_0,$$

and for every $q < q_{n+1}$, $q \neq q_n$, and for every $w \in \mathbb{Z}^2$ then

$$||q_n v - v_n||_0 < ||qv - w||_0.$$

Theorem 10.9.1. *1. The vector $(1/\beta, 1/\beta^2)$ is badly approximable by the rational numbers, that is, there exists $K > 0$ such that for every positive integer N, then*

$$\sqrt{N}\, |||N \cdot (1/\beta, 1/\beta^2)||| \geq K.$$

2. For every norm, the sequence $(\frac{q_{n+1}}{q_n})$ is bounded, where (q_n) denotes the sequence of best approximations of the vector $(1/\beta, 1/\beta^2)$.

3. The Tribonacci sequence (T_n) is the sequence of best approximations of the vector $(1/\beta, 1/\beta^2)$ for the Rauzy norm.

4. Furthermore

$$\lim_{n\to\infty} \sqrt{T_n}\, |||T_n \cdot (1/\beta, 1/\beta^2)||| = \frac{1}{\sqrt{\beta^2 + 2\beta + 3}}.$$

Proof.

1. Let $n \geq 1$ and let $m \geq 1$ such that $T_{m-1} \leq N < T_m$. Hence the normal T-representation of $N = \sum_{i \geq 0} \varepsilon_i T_i$ satisfies $\varepsilon_{m-1} = 1$. According to Lemma 10.8.13, then $|||N \cdot (1/\beta, 1/\beta^2)||| \geq C(1/\beta)^{m/2}$. There exist two constants C_1, C_2 such that the Tribonacci sequence satisfies: $\forall N \in \mathbb{N}$, $C_1\beta^n \leq T_n \leq C_2\beta^n$. Hence
$$|||N \cdot (1/\beta, 1/\beta^2)||| \geq C\sqrt{\frac{C_1}{T_m}} \geq \frac{CC_1}{\sqrt{C_2\beta N}},$$
which ends the proof of the first assertion. Let us observe that such a statement (up to the choice of the positive constant K) also holds for every norm, by equivalence of the norms.

2. Let $|| \; ||_0$ be a norm in $\mathbb{R}^2$ and let (q_n) be the sequence of best approximations of $(1/\beta, 1/\beta^2)$ associated with this norm.

   Let n and m such that $T_m \leq q_n < T_{m+1}$. For all $q < q_{n+1}$, one has by definition $|||q \cdot (1/\beta, 1/\beta^2)|||_0 \geq |||q_n \cdot (1/\beta, 1/\beta^2)|||_0$, where $||| \; |||_0$ denotes the distance to the nearest integer for the norm $||| \; |||_0$. We just have seen (proof of Assertion 1) that $|||q_n \cdot (1/\beta, 1/\beta^2)||| \geq K{q_n}^{-1/2}$. Hence
$$|||q_n \cdot (1/\beta, 1/\beta^2)||| > KT_{m+1}^{-1/2}.$$
   On the other hand, one has for l large enough, according to Lemma 10.8.10, that $|||T_{m+1+l} \cdot (1/\beta, 1/\beta^2)||| = ||\delta(T_{m+1+l})||$. Since the norms $|| \; ||_0$ and $|| \; ||$ are equivalent, then $|||T_{m+1+l} \cdot (1/\beta, 1/\beta^2)|||_0 = ||\delta(T_{m+1+l})||_0$ also holds for l large enough, still following Lemma 10.8.10. By equivalence of the norms, there exists a constant C_3 such that for l large enough
$$\begin{aligned} |||T_{m+1+l} \cdot (1/\beta, 1/\beta^2)|||_0 &= ||\delta(T_{m+1+l})||_0 \leq C_3||\delta(T_{m+1+l})|| \\ &= C_3|\alpha|^{m+l+5} \leq C_3\sqrt{C_2}|\alpha|^{l+4}T_{m+1}^{-1/2}. \end{aligned}$$
   Hence there exists l_0 large enough such that
$$|||T_{m+1+l_0} \cdot (1/\beta, 1/\beta^2)|||_0 < |||q_n \cdot (1/\beta, 1/\beta^2)|||_0,$$
   which implies that $T_{m+1+l_0} \geq q_{n+1}$. Hence one has
$$\frac{q_{n+1}}{q_n} \leq \frac{Tm+1+l_0}{T_m} \leq C_2/C_1\beta^{l_0+1}.$$

3. The sequence $(\delta(T_n))_n$ which satisfies $\delta(T_n) = |\alpha|^{n+4}$ is a decreasing sequence. Furthermore, for $n \geq 8$, $||\delta(T_n)|| \leq |\alpha|^{12} < C$, which implies that $|||T_n \cdot (1/\beta, 1/\beta^2)||| = ||\delta(T_n)||$. One checks by considering a finite number of cases that when $n < T_8$, then the properties of good approximation hold for the Tribonacci sequence.

   Let us assume from now on that $N \geq T_8$. We want to prove that if $N < T_{n+1}$ and $N \neq T_n$, then for every $v \in \mathbb{Z}^2$
$$||\delta(T_n)|| < ||N \cdot (1/\beta, 1/\beta^2) - v||.$$

Since $||N \cdot (1/\beta, 1/\beta^2) - v|| < C$ implies that $N \cdot (1/\beta, 1/\beta^2) - v = \delta(N)$, it is sufficient to check that $||\delta(T_n)|| < ||\delta(N)||$.

Let $n \in \mathbb{N}$ and let $N < T_{n+1}$, $N \neq T_n$, with normal T-representation $N = \sum_{0 \leq i \leq k} \varepsilon_i T_i$. Let $i_0 = \min\{i | \ \varepsilon_i \neq 0\}$ ($i_0 \neq n$ since $N \neq T_n$); hence $N = \sum_{i_0 \leq n} \varepsilon_i T_i$. One has

$$||\delta(N)|| = || \sum_{i_0 \leq i \leq n} \varepsilon_i B^i z|| \geq ||B^{i_0} z + \varepsilon_{i_0+1} B^{i_0+1} z|| - || \sum_{i_0+2 \leq i \leq n} \varepsilon_i B^i z||.$$

Let us prove that $||B^{i_0} z + \varepsilon_{i_0+1} B^{i_0+1} z|| - || \sum_{i_0+2 \leq i \leq n} \varepsilon_i B^i z|| > |\alpha|^{12+i_0}$.

- Assume first that $\varepsilon_{i_0+1} = 0$. Then

  $$|| \sum_{i \geq i_0+2}^{n} \varepsilon_i B^i z|| \leq ||B^{i_0+2} \sum_{i \geq 0} \varepsilon_{i_0+2+i} B^i z|| \leq \frac{|\alpha|^{i_0+2} ||z||}{1 - |\alpha|^3} = \frac{|\alpha|^{i_0+6}}{1 - |\alpha|^3}.$$

  Hence $||\delta(N)|| \geq |\alpha|^{i_0+4} \left(\frac{1-|\alpha|^2-|\alpha|^3}{1-|\alpha^3|} \right) > 0$.

- Assume now that $\varepsilon_{i_0+1} = 1$, and thus $\varepsilon_{i_0+2} = 0$. One has

  $$||\delta(N)|| \geq |\alpha|^{i_0} (||z + Bz|| - \frac{|\alpha|^7}{1 - |\alpha|^3}).$$

  It remains to check that $|\alpha|^4 \left(\frac{1-|\alpha|^2-|\alpha|^3}{1-|\alpha^3|} \right), ||z + Bz|| - \frac{|\alpha|^7}{1-|\alpha|^3} > |\alpha|^{12}$ to conclude.

If $n - i_0 \geq 8$, then $\delta(N) > |\alpha|^{i_0+12} \geq |\alpha|^{n+4} = ||\delta(T_n)||$.

In the case where $n - i_0 \leq 7$, one checks by considering a finite number of cases that $|| \sum_{i=0}^{m} \varepsilon_i B^i z|| > |\alpha|^{m+4}$, for $0 \leq m \leq 7$, which implies

$$\delta(N) = |\alpha|^{i_0} || \sum_{i=0}^{n-i_0} \varepsilon_i B^i z|| > |\alpha|^{i_0} |\alpha|^{n-i_0+4} \geq ||\delta(T_n)||.$$

4. Let K_0 be defined as the smallest real number such that there exist infinitely many integers N satisfying

   $$\sqrt{N} \, |||N \cdot (1/\beta, 1/\beta^2)||| < K_0.$$

   ¿From Assertion 1, one deduces that K_0 is finite. In fact, K_0 is the smallest real number such that there exist infinitely many integers N satisfying $\sqrt{T_n} \, ||\delta(T_n)|| < K_0$, since (T_n) is the sequence of best approximations of $(1/\beta, 1/\beta^2)$. The following limit exists and equals:

   $$\lim_{n \to +\infty} \sqrt{T_n} \, ||\delta(T_n)|| = \frac{1}{\sqrt{\beta^2 + 2\beta + 3}},$$

   hence the result. ∎

Problems

Section 10.4

10.4.1 (gcd). Let $q \geq 2$ be an integer. Prove that for all integers $m, n \geq 1$

$$\gcd(q^m - 1, q^n - 1) = \gcd(m, n).$$

(*Hint*. If $m \geq n$ and $m = \alpha n + \beta$ with $\beta \in [0, n-1]$, prove that an integer divides both $q^m - 1$ and $q^n - 1$ if and only if it divides both $q^m - 1$ and $q^\beta - 1$. Then use the Euclidean algorithm to compute gcd's).
Deduce that $q^m - 1$ divides $q^n - 1$ if and only if m divides n.

Section 10.5

10.5.1 (Möbius function). Define the Möbius function μ on the integers ≥ 1 by

$$\mu(n) := \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{if there exists } k \geq 2 \text{ such that } k^2 \text{ divides } n, \\ (-1)^r & \text{if } n = p_1 p_2 \cdots p_r, \text{ where the } p_i's \text{ are distinct primes.} \end{cases}$$

a. Prove that, for every $n \geq 1$,

$$\sum_{d|n} \mu(d) = \begin{cases} 1 \text{ if } n = 1, \\ 0 \text{ if } n \geq 2. \end{cases}$$

(*Hint*. Note that, if $n = \prod_{1 \leq j \leq r} p_j^{\alpha_j}$ is the decomposition of $n \geq 2$ into primes, then

$$\sum_{d|n} \mu(d) = \sum_{d|p_1 \cdots p_r} \mu(d) = \sum_{0 \leq j \leq r} \binom{r}{j} (-1)^j = (1-1)^r = 0.)$$

b. Prove the Möbius inversion formula: if f and g are two maps defined on the positive integers, then

$$\forall n \geq 1,\ g(n) = \sum_{d|n} f(d) \Rightarrow \forall n \geq 1,\ f(n) = \sum_{d|n} \mu(d) g(n/d).$$

c. Prove that, if F and G are two maps defined on the real numbers, then (summing over $n \leq x$ means that the summation is over the integers n such that $n \leq x$)

$$\forall x \geq 0,\ G(x) = \sum_{n \leq x} F(n) \Rightarrow \forall x \geq 0,\ F(x) = \sum_{n \leq x} \mu(n) G(\frac{x}{n}).$$

d. Define a *square-free* number as an integer that is not divisible by any square of an integer ≥ 2. Prove that for each integer $n \geq 1$ there exists a unique square-free number q and a unique integer a such that $n = a^2 q$.

e. Let $\lfloor x \rfloor$ be the integral part of the real x. Let $Q(x)$ be the number of square-free numbers smaller than x. Prove that, for each real number $x \geq 0$,
$$Q(x) = \sum_{n \leq \sqrt{x}} \mu(n) \lfloor \frac{x}{n^2} \rfloor.$$
(*Hint.* Start from
$$\lfloor x \rfloor = \sum_{n \leq x} 1 = \sum_{\substack{a^2 q \leq x \\ a \geq 1 \\ q \text{ squarefree}}} 1 = \sum_{a \leq \sqrt{x}} Q(\frac{x}{a^2}).$$
Deduce that
$$\lfloor x^2 \rfloor = \sum_{n \leq x} Q(\frac{x^2}{n^2})$$
and use Part b. above.)

f. Prove that the density of the square-free numbers exists and is equal to $\sum_{n \geq 1} \mu(n)/n^2$.
(*Hint.* Write
$$\begin{aligned} Q(x) &= \textstyle\sum_{n \leq \sqrt{x}} \mu(n) \lfloor \frac{x}{n^2} \rfloor = x \sum_{n \leq \sqrt{x}} \frac{\mu(n)}{n^2} + O(\sqrt{x}) \\ &= x \textstyle\sum_{n \geq 1} \frac{\mu(n)}{n^2} + O(\sqrt{x}).) \end{aligned}$$

g. Prove that $\sum_{n \geq 1} \mu(n)/n^2 = 6/\pi^2$.
(*Hint.* Write
$$\sum_{m \geq 1} \frac{\mu(m)}{m^2} \sum_{n \geq 1} \frac{1}{n^2} = \sum_{\ell \geq 1} \frac{1}{\ell^2} \sum_{m \mid \ell} \mu(m)$$
and use that $\sum_{n \geq 1} 1/n^2 = \pi^2/6$.)

h. Let $\psi_k(n)$ denote the number of primitive words of length n over an alphabet of size k. Prove that
$$k^n = \sum_{d \mid n} \psi_k(d).$$
(*Hint.* Every word w can be written in a unique way as $w = v^d$, where v is a primitive word, and d is an integer ≥ 1. Of course d must divide the length of w.)
Using the inversion formula b. above deduce that
$$\psi_k(n) = \sum_{d \mid n} \mu(d) k^{n/d}.$$

10.5.2 (Algebraicity). Prove that, if the formal power series $\sum a_n X^n$ has integral coefficients and is algebraic over the field $\mathbb{Q}(X)$, then the formal power series $\sum(a_n \bmod p)X^n$ is algebraic over $\mathbb{F}_p(X)$.

Section 10.7

10.7.1 (Complexity function of the Tribonacci word). First observe that the letters 2 and 3 are only followed or preceded by the letter 1 in the Tribonacci word u. Second, prove that every factor w distinct from the empty word ε of the Tribonacci word u can be uniquely written as follows: $w = r_1\sigma(v)r_2$, where v is a factor of u, $r_1 \in \{\varepsilon, 2, 3\}$, and $r_2 = 1$ if the last letter of w is 1, and r_2 is the empty word, otherwise. Deduce from this the following combinatorial properties:

a. Prove that the Tribonacci word u is not ultimately periodic, that is, periodic from some rank on.
b. A factor w of a word x is said *right special* if there exist two distinct letters a and b such that both wa and wb are factors of x. Prove that the Tribonacci word admits exactly one right special factor of each length.
c. The *complexity function* of an infinite word s is defined as the function $P(s,n)$ which counts the number of distinct factors of length n of s. Deduce that the complexity function $P(u,n)$ of the Tribonacci word satisfies: $\forall n \in \mathbb{N},\ P(u,n) = 2n+1$.
d. Prove that the Tribonacci word is uniformly recurrent, i.e., every factor appears infinitely often with bounded gaps.
e. Use the same method to prove that the Fibonacci word (defined in Section 10.1.4) admits exactly $n+1$ factors of length n.
f. Prove that the topological entropy (as defined in Section 1.8.3) of the set of factors of the Tribonacci word as well as the topological entropy of the set of factors of the Fibonacci word are equal to 0.

10.7.2 Prove that the Tribonacci word is not an automatic sequence, by considering the probabilities of occurrence of the letters. Deduce from Problem 1.8.1 and Section 1.8.6 the values of the probabilities of occurrence of the factors of length 2 of the Tribonacci word.

10.7.3 (Dumont-Thomas numeration system on words). The aim of this problem is to extend the statement of Lemma 10.7.2 to more general morphisms following Dumont and Thomas 1989, 1993, and Rauzy 1990.
Let τ be a morphism on the alphabet $\mathcal{A}$ satisfying the assumptions of Proposition 10.1.3. The *prefix automaton* of τ is defined as follows: its edges are the letters of $\mathcal{A}$; there is an edge from a to b labeled by $p \in \mathcal{A}^*$ if $\tau(a) = pas$, where $s \in \mathcal{A}^*$. For instance the prefix automaton of the Fibonacci morphism : $0 \to 01$, $b \to 0$ is the *Golden mean automaton* as defined in Example 1.3.5, where the label a has to be replaced by 1 and b by 0. Let v be the fixed point of τ having a as first letter, in the sense of Remark 10.1.4. Prove that every finite prefix of v can be uniquely

expanded as

$$\tau^n(p_n)\tau^{n-1}(p_{n-1})\cdots p_0,$$

where $p_n \neq \varepsilon$, and $p_n \cdots p_0$ is the sequence of labels of a path in the prefix automaton starting from the letter a. Conversely, prove that any such sequence of labels generates a finite prefix of v.

10.7.4 (Statistics on letters for Pisot morphisms). A morphism $\tau : \mathcal{A}^* \to \mathcal{A}^*$ is said of *Pisot type* if first it satisfies the assumptions of Proposition 10.1.3, and second, the eigenvalues of its incidence matrix satisfy the following: there exists a dominant eigenvalue α such that for every other eigenvalue λ, one gets $\alpha > 1 > |\lambda| > 0$. Deduce from Problem 10.7.3 that the results of Proposition 10.7.4 hold for any fixed point of a Pisot type morphism.

10.7.5 (Uniform balance). An infinite word $v \in \mathcal{A}^\omega$ is said *uniformly balanced* if there exists $C > 0$ such that for any two factors w, w' of the same length of v, and for any letter $i \in \mathcal{A}$, then

$$|\,|w|_i - |w'|_i| \leq C.$$

An infinite word $v \in \mathcal{A}^\omega$ is said to have *bounded remainder letters* if first, for every letter i, its probability of occurrence $\pi(i)$ in v exists, and second, there exists C' such that

$$\forall N,\ |\,|v_0 v_1 \cdots v_{N-1}|_i - \pi(i)N| \leq C'.$$

Prove that a sequence is uniformly balanced if and only if it has bounded remainder letters. Deduce from Problem 10.7.4 that a fixed point of a Pisot morphism is uniformly balanced.
For more results on the balance properties of fixed points of morphisms, see Adamczewski 2003.

Section 10.8

10.8.1 (Bounded remainder sets). A measurable set X with respect to the Lebesgue measure $\mu(\,)$ in $\mathbb{T}^2$ is said to be a *bounded remainder set* for the translation R_β if there exists $C > 0$ such that

$$\forall N, \quad |\,\mathrm{Card}\{i;\ 0 \leq i \leq N,\ R_\beta^n(0) \in X\} - \mu(X)| \leq C.$$

Deduce from Proposition 10.7.4 and Theorem 10.8.16 that the sets $\mathcal{R}_i$, $i = 1, 2, 3$, are bounded remainder sets.
Bounded remainder sets have been widely studied, see for instance Ferenczi 1992.

10.8.2 (Generalized Rauzy fractal and self-similarity). Let τ be a primitive morphism of Pisot type over the alphabet $\{1, \cdots, d\}$. Define similarly as in Section 10.8.1 a generalized Rauzy fractal $\mathcal{R}(\tau)$ as well as its

division into the pieces $\mathcal{R}_i(\tau)$, $i = 1, \cdots, d$. Prove that the statement of Proposition 10.8.1 still holds.
Prove that for $i = 1, \cdots, d$, that

$$M_\tau^{-1}(\mathcal{R}_i(\tau)) = \cup_{1\leq j\leq d} \cup_{pis,\ \tau(j)=pis} (\mathcal{R}_j(\tau) + M_\tau^{-1}(\pi_0 \circ f(p))).$$

(*Hint.* Apply M_σ to this equality.) This equality means that the pieces $\mathcal{R}_i(\tau)$ of the Rauzy fractal are self-similar (and more precisely self-affine), that is, they can be inflated under the expanding action of M_τ^{-1}, the image of each piece $\mathcal{R}_i(\tau)$, $i = 1, \cdots, d$ being redivided into translates of the sets $\mathcal{R}_j(\tau)$. This result is a generalization of the statement of Lemma 10.8.7. This self-similarity property is considered for instance in Holton and Zamboni 1998, Arnoux and Ito 2001, Sirvent and Wang 2002.

10.8.3 Deduce from the proof of Proposition 10.8.1 an upper bound on the diameter of the Rauzy fractal $\mathcal{R}$.

10.8.4 (β-numeration). Prove that every positive real number can be expanded as

$$x = \sum_{i=-d}^{+\infty} \varepsilon_i \beta^{-i}, \text{ where } d \in \mathbb{Z},\ \forall i,\ \varepsilon_i \in \{0,1\},\ \varepsilon_i\varepsilon_{i+1}\varepsilon_{i+2} = 0, \tag{10.10.1}$$

by introducing the β- *transformation* map $T_\beta : [0,1[\to [0,1[$, $x \mapsto \{\beta x\}$; such an expansion (with the above admissibility conditions (10.10.1)) is called a β-*expansion*; is there unicity of such an expansion? For more details on the β-numeration, see for instance Lothaire 2002.

10.8.5 (F-property). The aim of this problem is to prove that the set $\mathrm{Fin}(\beta)$ of positive real numbers having a finite β-expansion (see Problem 10.8.4) coincides with the set $(\mathbb{Z}[\beta^{-1}])_+$ of positive polynomials in $1/\beta$ with integer coefficients. This property is called the F-property and has been introduced in Frougny and Solomyak 1992, see also Akiyama 1999.

a. Let $\mathbb{Z}_+[\beta^{-1}]$ denote the set of polynomials in $1/\beta$ with non-negative integer coefficients. Prove that $\mathrm{Fin}(\beta)$ is included in $\mathbb{Z}_+[\beta^{-1}]$. (Use the fact that $1 = \beta^3 + \beta^2 + \beta$.)

b. The aim of this question is to prove that $\mathbb{Z}_+[\beta^{-1}] = (\mathbb{Z}[\beta^{-1}])_+$. Let $x \in (\mathbb{Z}[\beta^{-1}])_+$. Prove that there exists $s \in \mathbb{N}$ and $(x_0, x_1, x_2) \in \mathbb{Z}^3$ such that

$$x = \frac{1}{\beta^s}(x_0 + x_1\beta^{-1} + x_2\beta^{-2}) = \frac{1}{\beta^{s+1}} < (x_0, x_1, x_2), v_\beta >,$$

(we consider here the Euclidean scalar product in $\mathbb{R}^3$). Deduce that for all $n \in \mathbb{N}$, $x = \frac{1}{\beta^{s+n}} <{}^t M_\sigma^n(x_0, x_1, x_2), v_\beta >$. Apply the Perron Frobenius theorem to conclude.

c. The aim of this question is to prove that $\mathbb{Z}_+[\beta^{-1}] \cap [0,1[$ is included in $\mathrm{Fin}(\beta)$. For that purpose we introduce an algorithm consisting in

the repetition of the action of two steps A_1 and A_2, that transforms a finite β-representation of x (with digits not necessarily satisfying the admissible conditions (10.10.1)) into the β-expansion of x.
Let $x = \sum_{i=1}^{d} x_i \beta^{-i} \in \mathbb{Z}_+[\beta^{-1}] \cap [0, 1[$, where $\forall i,\ x_i \in \mathbb{N}$.

Step A_1. Assume that there exists an integer $k \geq 1$ such that $x_{k+1} \geq 1$, $x_{k+2} \geq 1$, and $x_{k+3} \geq 1$. Let A_1 be the algorithm which maps $(x_i)_{i \geq 1}$ (where we set $x_i = 0$ for $i > d$) to

$$(x'_i) = x_1 \cdots (x_k + 1)(x_{k+1} - 1)(x_{k+2} - 1)(x_{k+3} - 1)x_{k+4} \cdots$$

Prove that $\sum_i x'_i < \sum_i x_i$ and that $\sum_i x_i \beta^{-i} = \sum_i x'_i \beta^{-i}$.

Step A_2. Assume that there exists an index k such that $x_k \geq 2$. Prove that $k \geq 2$. Let l be the smallest integer such that $x_l \geq 2$. Let A_2 be the algorithm which sends $(x_i)_{i \geq 1}$ to

$$(x'_i) = x_1 \cdots (x_l - 1)(x_{l+1} + 1)(x_{l+2} + 1)(x_{l+3} + 1) \cdots$$

Let $k \geq 1$ be the largest integer such that $k \leq l$ and $x'_{k+1} \geq 1$, $x'_{k+2} \geq 1$, $x'_{k+3} \geq 1$. Then, the algorithm A_2 sends (x'_i) to

$$(x''_i) = x'_1 \cdots (x'_k + 1)(x'_{k+1} - 1)(x'_{k+2} - 1)(x'_{k+3} - 1).$$

The sequence (x''_i) is the image of (x_i) under the action of A_2. Prove that $\sum_i x''_i = \sum_i x_i$ and that $\sum_i x_i \beta^{-i} = \sum_i x''_i \beta^{-i}$.
We now apply repeatedly steps A_1 and A_2 to x, defining a sequence $(x^{(j)})$ such that for all j, $x^{(j)}$ takes finitely but all zero values. If for some value j_0, $x^{(j_0)}$ satisfies the admissibility conditions (10.10.1), then we set $x^{(j)} = x^{(j_0)}$, for $j \geq j_0$, and we apply no step anymore. Prove that for j large enough, step A_1 cannot be performed any more.
Let us assume that A_2 can be applied indefinitely. Let J be such that for $j > J$, step A_1 cannot be performed any more. Let l_j denote, for $j > J$, the smallest index l such that $x_l^{(j)} \geq 2$. Prove that (l_j) tends to infinity and that the sequence $(x^{(j)})$ is convergent. Find a contradiction.

d. Conclude.

We have followed here the proof of Frougny and Solomyak 1992.

10.8.6 (A tiling of the line). We have seen that $\pi_1(\{f(u_0 \cdots u_{N-1};\ N \in \mathbb{N}\}))$ is a Meyer set in Section 10.8.6. We associate here in a natural way to this set of points a tiling of the line.
Prove that $\pi_1(\{f(u_0 \cdots u_{N-1};\ N \in \mathbb{N}\}))$ defines a tiling $\mathcal{T}$ of the half-line generated by v_β in the positive octant $\{(x, y, z);\ x, y, z > 0\}$ by segments of three distinct lengths, say l_1, l_2, l_3, In other words, prove that the distance between two successive points of $\pi_1(\{f(u_0 \cdots u_{N-1};\ N \in \mathbb{N}\}))$ (with respect to the orientation on the half-line provided by v_β) equals either l_1, l_2 or l_3.

Prove that under a suitable choice of a unit vector on the half-line, then $\pi_1(\{f(u_0 \cdots u_{N-1};\ N \in \mathbb{N}\}))$ is in one-to-one correspondence with the set of *β-integers*

$$\mathbb{Z}_\beta^+ = \{\sum_{i=0}^{d} \varepsilon_i \beta^i;\ d \in \mathbb{N},\ \forall i,\ \varepsilon_i \in \{0,1\},\ \varepsilon_i \varepsilon_{i+1} \varepsilon_{i+2} = 0\}.$$

Let $(t_n)_{n \geq 0}$ denote the set of elements of $\pi_1(\{f(u_0 \cdots u_{N-1};\ N \in \mathbb{N}\}))$ ordered in increasing order (still with respect to the orientation on the half-line provided by v_β). One can code the tiling $\mathcal{T}$ as follows: for $n \geq 0$, for $i = 1, 2, 3$, then $v_n = i$ if and only if $t_{n+1} - t_n = l_i$. Prove that $(v_n)_{n \geq 0}$ is equal to the Tribonacci word.

Notes

For general references on substitutive sequences and substitutive dynamical systems, see for instance Queffélec 1987 and Fogg 2002.

The examples in Section 10.1.4 are famous. For more on Sturmian words, one can read for example Lothaire 2002 and Fogg 2002. For more on the Thue–Morse word, its history, and its many occurrences in the literature, see for example Allouche and Shallit 1999. The Rudin-Shapiro word was first introduced in Shapiro 1952. For all these words and for the paperfolding word one can read the notes of Allouche and Shallit 2003.

In Section 10.2.5, Lemma 10.2.13 is a classical result in Perron–Frobenius theory (see for example Gantmacher 1959). The main theorem of Section 10.2.5 (Theorem 10.2.15) is due to Cobham 1972.

The main theorem of Section 10.3.3 (Theorem 10.3.4) was proved in Christol 1979, see also Christol, Kamae, Mendès France, and Rauzy 1980. More generally, it is also possible to give a simple combinatorial characterization of primitive substitutive sequences (see Durand 1998, Holton and Zamboni 1999).

The first proof of Theorem 10.4.2 in Section 10.4 is due to Wade 1941. The proof we give here is adapted from a proof given in Allouche 1990.

The proof of Proposition 10.5.1 that we give in Section 10.5 comes from Allouche 1997. The first proof was given in Petersen 1994 and Petersen 1996. For a proof of the theorem of Chomsky-Schützenberger, see Chomsky and Schützenberger 1963.

The theorem of Ridout given without proof in Section 10.6 was given in Ridout 1957. Corollary 10.6.2 is due to Ferenczi and Mauduit 1997. Theorem 10.6.3 is also due to Ferenczi and Mauduit 1997 under a more general form. A slightly more precise result in the case of binary alphabets is given in Allouche and Zamboni 1998. For more results on the transcendence of "automatic" real numbers, see for example Allouche and Shallit 2003.

We do not claim here for exhaustivity in our choice of applications of the Rauzy fractal in number theory. We have chosen the more representative properties which also motivated G. Rauzy in its study of the Tribonacci word in

Rauzy 1982. All the results of Section 10.8 follow carefully the approach of the seminal paper Rauzy 1982, from which come the proofs of Theorem 10.8.16, Lemma 10.8.10 and 10.8.13, as well as the introduction of the matrix B, whereas the proof of Theorem 10.9.1 is due to Chekhova, Hubert, and Messaoudi 2001. The fact that the vector $(1/\beta, 1/\beta^2)$ is badly approximable by the rationals (Assertion 1 of Theorem 10.9.1) is a classical statement for elements of a totally real field number (see for instance Cassels 1957).

Arnoux–Rauzy words. The Tribonacci translation first occurred in Arnoux 1988, where the Tribonacci morphism was used to model an interval exchange map of 6 intervals and to build explicitly a continuous and surjective conjugacy between this interval exchange map and the Rauzy translation (see also Arnoux and Yoccoz 1981); these results have led to the introduction of the family of *Arnoux–Rauzy words* in Arnoux and Rauzy 1991, to which the Tribonacci word belongs, as a generalization of the family of Sturmian words.

Arnoux–Rauzy words are defined as the one-sided words x with complexity $P(x, n) = 2n+1$ for all n which are recurrent and which have for every length a unique right special factor and a unique left special factor, each of these special factors being extendable in three different ways. Let us note that they can be similarly defined over any alphabet of larger size, say d; one thus obtains infinite words of complexity $(d-1)n+1$. Contrary to the Sturmian case, these words are not characterized by their complexity function any more. For instance, codings of non-degenerated three-interval exchanges have also complexity $2n+1$. Let us observe that Arnoux–Rauzy words can be described as exchanges of six intervals of the unit circle (Arnoux and Rauzy 1991).

The combinatorial properties of the Arnoux–Rauzy words are well-understood and are perfectly described by a two-dimensional continued fraction algorithm defined over a subset of zero measure of the simplex introduced in Arnoux and Rauzy 1991, Risley and Zamboni 2000, Zamboni 1998 and in Chekhova 2000. By using this algorithm, one can express in an explicit way the probabilities of occurrence of factors of given length (Wozny and Zamboni 2001), one can count the number of all the factors of the Arnoux–Rauzy words (Mignosi and Zamboni 2002), or prove that the associated dynamical system has always simple spectrum (Chekhova 2000). See also Castelli, Mignosi, and Restivo 1999, and Justin 2000 for the connections with a generalization of the Fine and Wilf's theorem for three periods. The family of Arnoux–Rauzy words has been itself extended to the family of episturmian words (Justin and Pirillo 2002b, 2002a).

Rauzy fractal. The study of the topological properties of the Rauzy fractal is mainly due to Rauzy 1982, 1988, where the Rauzy fractal $\mathcal{R}$ is shown to be connected with simply connected interior (and so do the three pieces of the Rauzy fractal $\mathcal{R}_i$, $i = 1, 2, 3$). See also Messaoudi 1998 and Messaoudi 2000a for a parametrisation of its boundary, the points which have several expansions being studied in details (see also Remark 10.8.9). For a study of its fractal boundary, see Ito and Kimura 1991, where it is proved to be a Jordan curve generated by Dekking's fractal generation method (Dekking 1982), from which a computation of its Hausdorff dimension is deduced.

Theorem 10.8.16 states that the translation $v \mapsto v+(1/\beta, 1/\beta^2)$ on $\mathbb{T}^2$ can be coded using the Tribonacci morphism. In dynamical terms, this theorem extends to the fact that the symbolic dynamical system generated by the Tribonacci word is measure-theoretically isomorphic to a translation of the torus $\mathbb{T}^2$, the isomorphism being a continuous onto map. Furthermore it is also possible to construct a Markov partition for the toral automorphism of $\mathbb{T}^3$ of matrix given by the incidence matrix of the Tribonacci morphism, this construction being based on the Rauzy fractal.

More generally, it is possible to associate a generalized Rauzy fractal to any Pisot unimodular morphism (see Problem 10.8.2). (A morphism is said *unimodular* if the determinant of its incidence matrix equals ± 1.) There are several definitions associated with several methods of construction for such Rauzy fractals. We have given here a definition based on formal power series inspired by the seminal paper Rauzy 1982, by Messaoudi 1998, 2000a, and by Canterini and Siegel 2001a, 2001b. A different approach via iterated function systems and generalized substitutions has been developed following ideas from Ito and Kimura 1991, and Arnoux and Ito 2001, Sano, Arnoux, and Ito 2001. Indeed, Rauzy fractals can be described as the attractor of some graph iterated function system (IFS), as in Holton and Zamboni 1998 where one can find a study of the Hausdorff dimension of various sets related to Rauzy fractals, and as in Sirvent 2000a, 2000b, Sirvent and Wang 2002 with special focus on the self-similar properties of Rauzy fractals (see Lemma 10.8.7 and Problem 10.8.2). For more details on both approaches, see Chap. 7 and 8 of Fogg 2002. Both methods apply to unimodular morphisms of Pisot type.

More generally, for any unimodular morphism of Pisot type the measure-theoretical isomorphism with a translation on the torus (or equivalently the existence of a periodic tiling of the plane by the Rauzy fractal) is conjectured to hold. A large literature is devoted to this question, which is surveyed in Fogg 2002, Chap.7. Inspired by Bedford 1986, Ito and Ohtsuki 1993 extends Rauzy's approach in order to produce Markov partitions for toral automorphisms produced by the modified Jacobi-Perron algorithm. See also Praggastis 1999.

In particular, Arnoux–Rauzy words which are fixed points of primitive morphisms (which are thus unimodular and of Pisot type following Arnoux and Ito 2001) also generate symbolic dynamical systems which are measure-theoretically isomorphic to toral translations. It was believed that all Arnoux–Rauzy words originated from toral translations, and more precisely, that they were natural codings of translations over $\mathbb{T}^2$. This conjecture was disproved in Cassaigne, Ferenczi, and Zamboni 2000.

Tribonacci numeration system. The Tribonacci numeration system is the canonical numeration system associated with the positive root β of $X^3 = X^2 + X + 1$. More generally, for a given $\beta > 1$, one can expand real numbers in $[0, 1[$ as powers of the number β using the greedy algorithm: $x = \sum_{k=1}^{\infty} b_k \beta^{-k}$, with some conditions on the nonnegative integers b_k (see also Problem 10.8.4); such expansions are called β-*expansions* and are generated by the β-*transformation* $x \mapsto \beta x - [\beta x]$ which also generates as a dynamical system the β-shift (for more details, see for instance Lothaire 2002). One can also represent natural integers in a base given by an infinite sequence of integers (which generalizes Lemma 10.7.1) canonically associated with the β-numeration: the set of factors of greedy representations of natural integers in this base and the factors of the β-shift are the same. Similar compact sets with fractal boundary are considered as geometrical representations of the β-shift when β is a Pisot unit, in Thurston 1989, in Akiyama 1999 and in Praggastis 1999, where topological or tiling properties such as Proposition 10.8.12 or Lemma 10.8.14 are studied in connection with the so-called F-property (Frougny and Solomyak 1992) (see also Problem 10.8.5). Generalized Rauzy fractals issued from the β-numeration are also closely related to canonical number systems (see for instance Akiyama and Pethö 2002).

There are also some close connections between the dynamical properties of the Rauzy fractal and the extension of the Fibonacci multiplication (introduced in Knuth 1988) to the Tribonacci recurrence relation, as studied for instance in Arnoux 1989 and Messaoudi 2000b, 2002.

Rauzy fractals can be used to characterize the numbers that have a purely periodic β-expansion, producing a kind of generalized Galois' theorem on classical continuous fractions. It is known following Schmidt 1980 and Bertrand 1977 that elements of $\mathbb{Q}(\beta)$ have a ultimately periodic expansion when β is a Pisot number. A characterization of those points having an immediately periodic expansion is given in Sano 2002, see also Ito and Sano 2001, by introducing a realization of the natural extension of the β-transformation acting on the associated generalized Rauzy fractal for β being a Pisot unit which is a simple β-number. See also Ito 2000 (and more generally Gambaudo et al. 2000) for closely related results for elements of cubic fields. Let us observe furthermore that the results of Section 10.9 can be extended following the same ideas to other cubic numbers (Ito 1996, Ito, Fujii, and Yasutomi 2003). Such results can also be proved using algebraic geometry following Adams 1969.

Rauzy tilings have also been studied in theoretical physics and quasycristal theory in Vidal and Mosseri 2000, 2001 as outlined in Section 10.8.6, where we have followed the terminology of Moody 1997. For more on mathematical quasicrystals, see for instance Baake and Moody 2000. See also Burdík et al. 1998, 2000, Verger Gaugry and Gazeau 2004 for connected results in the framework of beta-numeration.

References

Adamczewski, B. (2003). Balances for fixed points of primitive substitutions, *Theoret. Comput. Sci.*, *307*, 47–75.

Adams, W. W. (1969). Simultaneous asymptotic diophantine approximations to a basis of a real cubic number field, *J. Number Theory*, *1*, 179–194.

Adebiyi, E. F., Jiang, T., and Kaufmann, M. (2001). An efficient algorithm for finding short approximate non-tandem repeats, In *Intelligent Systems for Molecular Biology (ISMB)*, pp. 5–12.

Adebiyi, E. F. and Kaufmann, M. (2002). Extracting common motifs under the Levenshtein measure: Theory and experimentation, In Guigó, R. and Gusfield, D. (Eds.), *Algorithms in Bioinformatics (WABI 2002)*, Vol. 2452 of *Lect. Notes Comp. Sci.*, pp. 140–156. Springer-Verlag.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1975). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1983). *Data Structures and Algorithms*. Addison-Wesley.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques and Tools*. Addison-Wesley.

Akiyama, S. (1999). Self affine tiling and Pisot numeration system, In *Number Theory and its Applications (Kyoto, 1997)*, Vol. 2 of *Dev. Math.*, pp. 7–17. Kluwer.

Akiyama, S. and Pethö, A. (2002). On canonical number systems, *Theoret. Comput. Sci.*, *70*, 921–933.

Allauzen, C. and Mohri, M. (2003). Efficient algorithms for testing the twins property, *J. Autom. Lang. Comb.*, *8*(2), 117–144.

Allauzen, C., Mohri, M., and Riley, M. (2004). Statistical modeling for unit selection in speech synthesis, In 42*nd Meeting Assoc. Computational Linguistics (ACL 2004)*, Barcelona, Spain.

Allouche, J.-P. (1990). Sur la transcendance de la série formelle π, *Sém. Théor. Nombres Bordeaux*, *2*, 103–117.

Allouche, J.-P. (1997). Note on the transcendence of a generating function, In Laurincikas, A., Manstavicius, E., and Stakenas, V. (Eds.), *Proc. Palanga Conference*, Vol. 4 of *New Trends in Probability and Statistics*, pp. 461–465. VSP Science.

Allouche, J.-P. and Shallit, J. (1999). The ubiquitous Prouhet-Thue-Morse sequence, In Ding, C., Helleseth, T., and Niederreiter, H. (Eds.), *Sequences and Their Applications, Proceedings of SETA '98*, pp. 1–16. Springer-Verlag.

Allouche, J.-P. and Shallit, J. (2003). *Automatic Sequences*. Cambridge University Press.

Allouche, J.-P. and Zamboni, L. Q. (1998). Algebraic irrational binary numbers cannot be fixed points of non-trivial constant-length or primitive morphisms, *J. Number Theory*, *69*, 119–124.

Alon, N. and Spencer, J. (1992). *The Probabilistic Method*. J. Wiley and Sons.

Alonso, L., Rémy, J. L., and Schott, R. (1997). A linear-time algorithm for the generation of trees, *Algorithmica*, *17*(2), 162–182.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool, *J. Mol. Biol.*, *215*, 403–410.

Ambjørn, J., Durhuus, B., and Jonsson, T. (1997). *Quantum Geometry*. Cambridge Monographs on Mathematical Physics. Cambridge University Press.

Angluin, D. (1982). Inference of reversible languages, *J. Assoc. Comput. Mach.*, *29*(3), 741–765.

Angluin, D. (1987). Learning regular sets from queries and counterexamples, *Inform. Comput.*, *75*(2), 87–106.

Apostolico, A. (1985). The myriad virtues of suffix trees, In Apostolico, A. and Galil, Z. (Eds.), *Combinatorial Algorithms on Words*, Vol. 12 of *NATO Advanced Science Institutes, Series F*, pp. 85–96. Springer-Verlag.

Apostolico, A., Bock, M., and Xuyan, X. (1998). Annotated statistical indices for sequence analysis, In *Compression and Complexity of Sequences 97*, pp. 215–229. IEEE Computer Society Press.

Apostolico, A. and Crochemore, M. (1991). Optimal canonization of all substrings of a string, *Inform. Comput.*, *95*(1), 76–95.

Apostolico, A. and Preparata, F. (1983). Optimal off-line detection of repetitions in a string, *Theoret. Comput. Sci.*, *22*(3), 297–315.

Apostolico, A. and Szpankowski, W. (1992). Self-alignments in words and their applications, *J. Algorithms*, *13*, 446–467.

Arnoux, P. (1988). Un exemple de semi-conjugaison entre un échange d'intervalles et une translation sur le tore, *Bull. Soc. Math. France*, *116*(4), 489–500.

Arnoux, P. (1989). Some remarks about Fibonacci multiplication, *Appl. Math. Lett.*, *2*(4), 319–320.

Arnoux, P. and Ito, S. (2001). Pisot substitutions and Rauzy fractals, *Bull. Belg. Math. Soc. Simon Stevin*, *8*(2), 181–207.

Arnoux, P. and Rauzy, G. (1991). Représentation géométrique de suites de complexité $2n+1$, *Bull. Soc. Math. France*, *119*(2), 199–215.

Arnoux, P. and Yoccoz, J.-C. (1981). Construction de difféomorphismes pseudo-Anosov, *C. R. Acad. Sci. Paris Sér. I Math.*, *292*(1), 75–78.

Arratia, R., Bollobas, B., Coppersmith, D., and Sorkin, G. (2000). Euler circuits and DNA sequencing by hybridization, *Discr. Appl. Math.*, *104*, 63–69.

Arratia, R., Goldstein, L., and Gordon, L. (1989). Two moments suffice for Poisson approximations : the Chen-Stein method, *Ann. Probab.*, *17*, 9–25.

Arratia, R., Goldstein, L., and Gordon, L. (1990). Poisson approximation and the Chen-Stein method, *Statist. Sci.*, *5*, 403–434.

Arratia, R., Martin, D., Reinert, G., and Waterman, M. S. (1996). Poisson approximation for long repeats in a random sequence with application to sequencing by hybridization, *J. Comput. Biol.*, *3*, 425–463.

Arratia, R., Morris, P., and Waterman, M. S. (1988). Stochastic Scrabble: large deviations for sequences with scores, *J. Appl. Probab.*, *25*, 106–119.

Arratia, R. and Waterman, M. (1994). A phase transition for the score in matching random sequences allowing deletions, *Ann. Appl. Probab.*, *4*, 200–225.

Arratia, R. and Waterman, M. S. (1989). The Erdös-Rényi strong law for pattern matching with a given proportion of mismatches, *Ann. Probab.*, *17*, 1152–1169.

Ash, R. B. (1990). *Information Theory*. Dover. Corrected reprint of the 1965 original edition.

Atallah, M., Jacquet, P., and Szpankowski, W. (1993). A probabilistic analysis of a pattern matching problem, *Random Structures & Algorithms*, *4*, 191–213.

Baake, M. and Moody, R. V. (Eds.) (2000). *Directions in Mathematical Quasicrystals*, Vol. 13 of *CRM Monograph Series*. Amer. Math. Soc.

Baeza-Yates, R. and Ribero-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley.

Bahl, L. R., Jelinek, F., and Mercer, R. (1983). A maximum likelihood approach to continuous speech recognition, *IEEE Trans. Pattern Anal. Machine Intell.*, *5*(2), 179–190.

Baldi, P. and Brunak, S. (1998). *Bioinformatics. The Machine Learning Approach*. The MIT Press.

Banderier, C. and Flajolet, P. (2002). Basic analytic combinatorics of directed lattice paths, *Theoret. Comput. Sci.*, *281*(1-2), 37–80.

Barbour, A. D., Chen, L., and Loh, W.-L. (1992). Compound Poisson approximation for nonnegative random variables via Stein's method, *Ann. Probab.*, *20*, 1843–1866.

Barbour, A. D. and Chryssaphinou, O. (2001). Compound Poisson approximation: a user's guide, *Ann. in Appl. Probab.*, *11*, 964–1002.

Barbour, A. D., Chryssaphinou, O., and Vaggelatou, E. (2001). Applications of compound Poisson approximation, In Charalambides, C. A., Koutras, M. V., and Balakrishnan, N. (Eds.), *Probability and Statistical Models with Applications*, pp. 41–62. Chapman and Hall.

Barbour, A. D., Holst, L., and Janson, S. (1992). *Poisson Approximation*. Oxford University Press.

Barbour, A. D. and Mansson, M. (2002). Compound Poisson process approximation, *Ann. Probab.*, *30*, 1492–1537.

Barbour, A. D. and Utev, S. (1998). Solving the Stein equation in compound Poisson approximation, *Adv. in Appl. Probab.*, *30*, 449–475.

Barbour, A. D. and Xia, A. (1999). Poisson perturbations, *ESAIM Probab. Statist.*, *3*, 131–150.

Barcucci, E., Pergola, E., Pinzani, R., and Rinaldi, S. (2001). A bijection for some paths on the slit plane, *Adv. in Appl. Math.*, *26*(2), 89–96.

Barcucci, E., Pinzani, R., and Sprugnoli, R. (1995). The random generation of underdiagonal walks, *Discrete Math.*, *139*(1-3), 3–18.

Béal, M.-P. and Carton, O. (2001). Computing the prefix of an automaton, *Theoret. Inform. Appl.*, *34*(6), 503–515.

Béal, M.-P. and Carton, O. (2002). Determinization of transducers over finite and infinite words, *Theoret. Comput. Sci.*, *289*(1), 225–251.

Bedford, T. (1986). Generating special Markov partitions for hyperbolic toral automorphisms using fractals, *Ergodic Theory Dynam. Systems*, *6*(3), 325–333.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression*. Prentice Hall.

Bender, E. (1973). Central and local limit theorems applied to asymptotic enumeration, *J. Combin. Th. A*, *15*, 91–111.

Bender, E. and Kochman, F. (1993). The distribution of subword counts is usually normal, *European J. Combin.*, *14*, 265–275.

Bergeron, F., Labelle, G., and Leroux, P. (1998). *Combinatorial Species and Tree-like Structures*, Vol. 67 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Berry, G. and Sethi, R. (1986). From regular expressions to deterministic automata, *Theoret. Comput. Sci.*, *48*, 117–126.

Berstel, J. (1979). *Transductions and Context-Free Languages*. Teubner.

Berstel, J. and Reutenauer, C. (1984). *Rational Series and Their Languages*. Springer-Verlag.

Bertrand, A. (1977). Développements en base de Pisot et répartition modulo 1, *C. R. Acad. Sci. Paris, Sér. A*, *285*, 419–421.

Bétréma, J. and Penaud, J.-G. (1993). Modèles avec particules dures, animaux dirigés et séries en variables partiellement commutatives, available at `arXiv:math.CO/0106210`.

Biaudet, V., El Karoui, M., and Gruss, A. (1998). Codon usage can explain GT-rich islands surrounding Chi sites on the Escherichia coli genome, *Mol. Microbiol.*, *29*, 666–669.

Bieganski, P., Riedl, J., Carlis, J. V., and Retzel, E. (1994). Generalized suffix trees for biological sequence data: applications and implementations, In *27th Hawai Int. Conf. on Systems Science*, pp. 35–44. IEEE Computer Society Press.

Biggins, J. and Cannings, C. (1987). Markov renewal processes, counters and repeated sequences in Markov chains, *Ann. in Appl. Probab.*, *19*, 521–545.

Billingsley, P. (1961). Statistical methods in Markov chains, *Ann. Math. Statist.*, *2*, 12–40.

Blanchette, M. (2001). Algorithms for phylogenetic footprinting, In *5th Research in Computational Molecular Biology (RECOMB)*, pp. 49–58. ACM Press.

Blanchette, M., Schwikowski, B., and Tompa, M. (2000). An exact algorithm to identify motifs in orthologous sequences from multiple species, In *Intelligent Systems for Molecular Biology (ISMB)*, pp. 37–45.

Blanchette, M., Schwikowski, B., and Tompa, M. (2002). Algorithms for phylogenetic footprinting, *J. Comput. Biol.*, *9*(2), 211–223.

Blanchette, M. and Tompa, M. (2002). Discovery of regulatory elements by a computational method for phylogenetic footprinting, *Genome Research*, *12*, 739–748.

Blom, G. and Thorburn, D. (1982). How many random digits are required until given sequences are obtained?, *J. Appl. Probab.*, *19*, 518–531.

Blumer, A., Blumer, J., Ehrenfeucht, A., Haussler, D., and McConnel, R. (1983). Linear size finite automata for the set of all subwords of a word: an outline of results, *Bull. Eur. Assoc. Theoret. Comput. Sci.*, *21*, 12–20.

Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M. T., and Seiferas, J. (1985). The smallest automaton recognizing the subwords of a text, *Theoret. Comput. Sci.*, *40*(1), 31–55.

Blumer, A., Ehrenfeucht, A., and Haussler, D. (1989). Average size of suffix trees and DAWGS, *Discr. Appl. Math.*, *24*, 37–45.

Booth, K. S. (1980). Lexicographically least circular substrings, *Inform. Proc. Letters*, *10*(4-5), 240–242.

Bourdon, J. and Vallée, B. (2002). Generalized pattern matching statistics, In *Mathematics and Computer Science II (Versailles, 2002)*, Trends. Math., pp. 249–265. Birkhäuser.

Bousquet-Mélou, M. (1996). *Habilitation à diriger des recherches*. Université Bordeaux I, available at `http://www.labri.fr/~bousquet/`.

Bousquet-Mélou, M. (2001). Walks on the slit plane: other approaches, *Adv. in Appl. Math.*, *27*(2-3), 243–288.

Bousquet-Mélou, M. (2002). Counting walks in the quarter plane, In *Mathematics and Computer Science II (Versailles, 2002)*, Trends Math., pp. 49–67. Birkhäuser.

Bousquet-Mélou, M. and Guttmann, A. J. (1997). Three-dimensional self-avoiding convex polygons, *Phys. Rev. E (3)*, *55*(6, part A), R6323–R6326.

Bousquet-Mélou, M. and Schaeffer, G. (2002). Walks on the slit plane, *Probab. Theory Related Fields*, *124*(3), 305–344.

Bouttier, J., Di Francesco, P., and Guitter, E. (2002). Census of planar maps: from the one-matrix model solution to a combinatorial proof, *Nuclear Phys. B*, *645*(3), 477–499.

Brazma, A., Jonassen, I., Eidhammer, I., and Gilbert, D. (1998a). Approaches to the automatic discovery of patterns in biosequences, *J. Comput. Biol.*, *5*(2), 277–304.

Brazma, A., Jonassen, I., Vilo, J., and Ukkonen, E. (1998b). Pattern discovery in biosequences, In Honavar, V. and Slutzki, G. (Eds.), *Grammatical Inference (ICGI-98)*, Vol. 1433 of *Lect. Notes Comp. Sci.*, pp. 255–270. Springer-Verlag.

Brazma, A., Jonassen, I., Vilo, J., and Ukkonen, E. (1998c). Predicting gene regulatory elements *in silico* on a genomic scale, *Genome Research*, *8*(11), 1202–1215.

Breen, S., Waterman, M. S., and Zhang, N. (1985). Renewal theory for several patterns, *J. Appl. Probab.*, *22*, 228–234.

Brendel, V., Beckmann, J. S., and Trifonov, E. N. (1986). Linguistics of nucleotide sequences: Morphology and comparison of vocabularies, *J. Biomol. Struct. Dynamics*, *4*, 11–21.

Breslauer, D. (1998). The suffix tree of a tree and minimizing sequential transducers, *Theoret. Comput. Sci.*, *191*(1-2), 131–144.

Brodal, G., Lyngsø, R., Pedersen, C., and Stoye, J. (2000). Finding maximal pairs with bounded gap, *J. Discrete Algorithms*, *1*(1), 77–104.

Brown, T. A. (1999). *Genomes*. BIOS Scientific Publishers.

Bucklew, J. A. (1990). *Large Deviation Techniques in Decision, Simulation, and Estimation*. J. Wiley and Sons.

Buhler, J. and Tompa, M. (2001). Finding motifs using random projections, In *5th Research in Computational Molecular Biology (RECOMB)*, pp. 69–76. ACM Press.

Burdík, Č., Frougny, C., Gazeau, J. P., and Krejcar, R. (1998). Beta-integers as natural counting systems for quasicrystals, *J. Phys. A*, *31*(30), 6449–6472.

Burdík, Č., Frougny, C., Gazeau, J.-P., and Krejcar, R. (2000). Beta-integers as a group, In Gambaudo, J.-M., Hubert, P., Tisseur, P., and Vaienti, S. (Eds.), *Dynamical Systems (Luminy-Marseille, 1998)*, pp. 125–136. World Scientific.

Canterini, V. and Siegel, A. (2001a). Automate des préfixes-suffixes associé à une substitution primitive, *J. Théor. Nombres Bordeaux*, *13*(2), 353–369.

Canterini, V. and Siegel, A. (2001b). Geometric representation of substitutions of Pisot type, *Trans. Amer. Math. Soc.*, *353*(12), 5121–5144.

Cartier, P. and Foata, D. (1969). *Problèmes combinatoires de commutation et réarrangements*, Vol. 85 of *Lect. Notes Math.* Springer-Verlag.

Cassaigne, J., Ferenczi, S., and Zamboni, L. Q. (2000). Imbalances in Arnoux-Rauzy sequences, *Ann. Inst. Fourier (Grenoble)*, *50*(4), 1265–1276.

Cassels, J. W. S. (1957). *An Introduction to Diophantine Approximation*. Cambridge University Press.

Castelli, M. G., Mignosi, F., and Restivo, A. (1999). Fine and Wilf's theorem for three periods and a genereralization of Sturmian words, *Theoret. Comput. Sci.*, *218*, 83–94.

Chedin, F., Noirot, P., Biaudet, V., and Ehrlich, S. (1998). A five-nucleotide sequence protects DNA from exonucleolytic degradation by AddAB, the RecBCD analogue of *Bacillus subtilis*, *Mol. Microbiol.*, *31*, 1369–1377.

Chekhova, N. (2000). Covering numbers of rotations, *Theoret. Comput. Sci.*, *230*(1-2), 97–116.

Chekhova, N., Hubert, P., and Messaoudi, A. (2001). Propriétés combinatoires, ergodiques et arithmétiques de la substitution de Tribonacci, *J. Théor. Nombres Bordeaux*, *13*(2), 371–394.

Chen, L. H. Y. (1975). Poisson approximation for dependent trials, *Ann. Probab.*, *3*, 534–545.

Chen, L. H. Y. and Xia, A. (to appear). Stein's method, Palm theory and Poisson process approximation, *Ann. Probab.*

Choffrut, C. (1977). Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles, *Theoret. Comput. Sci.*, *5*(3), 325–337.

Choffrut, C. (1979). A generalization of Ginsburg and Rose's characterization of G-S-M mappings, In Maurer, H. A. (Ed.), *6th Automata, Languages and Programming (ICALP)*, Vol. 71 of *Lect. Notes Comp. Sci.*, pp. 88–103. Springer-Verlag.

Choffrut, C. (2003). Minimizing subsequential transducers: a survey, *Theoret. Comput. Sci.*, *292*(1), 131–143.

Choffrut, C. and Karhumäki, J. (1997). Combinatorics of words, In Rozenberg, G. and Salomaa, A. (Eds.), *Handbook of Formal Languages*, Vol. I, pp. 329–438. Springer-Verlag.

Chomsky, N. (1956). Three models for the description of language, *IRE Trans. Inform. Theory*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures.* Mouton, The Hague.

Chomsky, N. and Schützenberger, M. P. (1963). The algebraic theory of context-free languages, In Braffort, P. and Hirschberg, D. (Eds.), *Computer Programming and Formal Systems*, pp. 118–161. North Holland, Amsterdam.

Chottin, L. and Cori, R. (1982). Une preuve combinatoire de la rationalité d'une série génératrice associée aux arbres, *RAIRO Inform. théor.*, *16*(2), 113–128.

Christol, G. (1979). Ensembles presque périodiques k-reconnaissables, *Theoret. Comput. Sci.*, *9*, 141–145.

Christol, G., Kamae, T., Mendès France, M., and Rauzy, G. (1980). Suites algébriques, automates et substitutions, *Bull. Soc. Math. France*, *108*, 401–419.

Chryssaphinou, O. and Papastavridis, S. (1988a). A limit theorem for the number of non-overlapping occurrences of a pattern in a sequence of independent trials, *J. Appl. Probab.*, *25*, 428–431.

Chryssaphinou, O. and Papastavridis, S. (1988b). A limit theorem on the number of overlapping appearances of a pattern in a sequence of independent trials, *Probab. Theory Related Fields*, *79*, 129–143.

Chryssaphinou, O., Papastavridis, S., and Vaggelatou, E. (2001). Poisson approximation for the non-overlapping appearances of several words in Markov chains, *Combin. Probab. Comput.*, *10*, 293–308.

Chung, K. L. (1974). *A Course in Probability Theory.* Academic Press. 2nd edition.

Churchill, G. A. (1989). Stochastic models for heterogeneous DNA sequences, *Bull. Math. Biol.*, *51*, 79–94.

Churchill, G. A., Daniels, D. L., and Waterman, M. S. (1990). The distribution of restriction enzyme sites in *Escheria coli*, *Nucl. Acids Res.*, *18*, 589–597.

Clement, J., Flajolet, P., and Vallée, B. (2001). Dynamic sources in information theory: a general analysis of trie structures, *Algorithmica*, *29*, 307–369.

Cobham, A. (1972). Uniform tag sequences, *Math. Systems Theory*, *6*, 164–192.

Cole, R. and Hariharan, R. (1998). Approximate String Matching: A Simpler Faster Algorithm, In *9th SIAM Symposium on Discrete Algorithms (SODA)*, pp. 463–472.

Cori, R. (1975). *Un code pour les graphes planaires et ses applications.* Société Mathématique de France. Astérisque, No. 27.

Cori, R. and Machì, A. (1992). Maps, hypermaps and their automorphisms: a survey. I, II, III, *Exposition. Math.*, *10*(5), 403–427, 429–447, 449–467.

Cori, R. and Vauquelin, B. (1981). Planar maps are well labeled trees, *Canad. J. Math.*, *33*(5), 1023–1042.

Cowan, R. (1991). Expected frequencies of DNA patterns using Whittle's formula, *J. Appl. Probab.*, *28*, 886–892.

Crochemore, M. (1981). An optimal algorithm for computing the repetitions in a word, *Inform. Proc. Letters*, *12*, 244–250.

Crochemore, M. (1983). Recherche linéaire d'un carré dans un mot, *Comptes Rendus Acad. Sci. Paris Sér. I Math.*, *296*, 781–784.

Crochemore, M. (1986). Transducers and repetitions, *Theoret. Comput. Sci.*, *45*(1), 63–86.

Crochemore, M. (1987). Longest common factor of two words, In Ehrig, H., Kowalski, R., Levi, G., and Montanari, U. (Eds.), *2nd Theory and Practice of Software Development (TAPSOFT), Pisa, Italy*, Vol. 249 of *Lect. Notes Comp. Sci.*, pp. 26–36. Springer-Verlag.

Crochemore, M., Hancart, C., and Lecroq, T. (2001). *Algorithmique du texte*. Vuibert.

Crochemore, M., Mignosi, F., Restivo, A., and Salemi, S. (2000). Data compression using antidictonaries, *Proceedings of the IEEE*, *88*(11), 1756–1768. Special issue *Lossless data compression* edited by J. Storer.

Crochemore, M. and Rytter, W. (1994). *Text Algorithms*. The Clarendon Press Oxford University Press.

Crochemore, M. and Rytter, W. (1995). Squares, cubes, and time-space efficient string searching, *Algorithmica*, *13*, 405–425.

Crochemore, M. and Vérin, R. (1997). On compact directed acyclic word graphs, In Mycielski, J., Rozenberg, G., and Salomaa, A. (Eds.), *Structures in Logic and Computer Science*, Vol. 1261 of *Lect. Notes Comp. Sci.*, pp. 192–211. Springer-Verlag.

Culik II, K. and Kari, J. (1997). Digital Images and Formal Languages, In Rozenberg, G. and Salomaa, A. (Eds.), *Handbook of Formal Languages*, Vol. 3, pp. 599–616. Springer-Verlag.

Dacunha-Castelle, D. and Duflo, M. (1983). *Probabilités et statistiques 2.Problèmes à temps mobile*. Masson.

De Luca, A. (1981). A combinatorial property of the Fibonacci words, *Inform. Proc. Letters*, *12*(4), 193–195.

Deheuvels, P., Devroye, L., and Lynch, J. (1986). Exact convergence rate in the limit theorems of Erdös-Rényi and Shepp, *Ann. Probab.*, *14*, 209–223.

Dekking, F. M. (1982). Recurrent sets, *Adv. in Math.*, *44*(1), 78–104.

Del Lungo, A., Mirolli, M., Pinzani, R., and Rinaldi, S. (2001). A bijection for directed-convex polyominoes, In Cori, R., Mazoyer, J., Morvan, M., and Mosseri, R. (Eds.), *Discrete Models: Combinatorics, Computation, and Geometry, DM-CCG 2001*, Vol. AA of *DMTCS Proceedings*, pp. 133–144.

Dembo, A. and Karlin, S. (1992). Poisson approximations for r-scan processes, *Ann. Appl. Probab.*, *2*, 329–357.

Denise, A. and Régnier, M. (2004). Rare events and conditional events on random strings, *Discrete Math. Theor. Comput. Sci.*, *6*, 191–214.

Denise, A., Régnier, M., and Vandenbogaert, M. (2001). Assessing the statistical significance of overrepresented oligonucleotides, In Gascuel, O. and Moret, B. M. E. (Eds.), *Algorithms in Bioinformatics (WABI 2001)*, Vol. 2149 of *Lect. Notes Comp. Sci.*, pp. 85–97. Springer-Verlag.

Dershowitz, N. and Zaks, S. (1990). The cycle lemma and some applications, *European J. Combin.*, *11*, 35–40.

Devroye, L., Szpankowski, W., and Rais, B. (1992). A note of the height of suffix trees, *SIAM J. Comput.*, *21*, 48–53.

Di Francesco, P. (2001). Matrix model combinatorics: applications to folding and coloring, In *Random Matrix Models and Their Applications*, Vol. 40 of *Math. Sci. Res. Inst. Publ.*, pp. 111–170. Cambridge University Press.

Duchon, P., Flajolet, P., Louchard, G., and Schaeffer, G. (2002). Random sampling from Boltzmann principles, In Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., and Conejo, R. (Eds.), *29th Automata, Languages and Programming (ICALP '02)*, Vol. 2380 of *Lect. Notes Comp. Sci.*, pp. 501–513. Springer-Verlag.

Dumont, J.-M. and Thomas, A. (1989). Systèmes de numération et fonctions fractales relatifs aux substitutions, *Theoret. Comput. Sci.*, *65*(2), 153–169.

Dumont, J.-M. and Thomas, A. (1993). Digital sum moments and substitutions, *Acta Arith.*, *64*, 205–225.

Durand, F. (1998). A characterization of substitutive sequences using return words, *Discrete Math.*, *179*(1-3), 89–101.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press.

Duval, J.-P. (1983). Factorizing words over an ordered alphabet, *J. Algorithms*, *4*(4), 363–381.

Duval, J.-P. (1998). Périodes locales et propagation de périodes dans un mot, *Theoret. Comput. Sci.*, *204*(1-2), 87–98.

Duval, J.-P., Kolpakov, R., Kucherov, G., Lecroq, T., and Lefebvre, A. (2003). Linear-time computation of local periods, In Rovan, B. and Vojtas, P. (Eds.), *28th Mathematical Foundations of Computer Science (MFCS 2003)*, Vol. 2747 of *Lect. Notes Comp. Sci.*, pp. 388–397. Springer-Verlag.

Duval, J.-P., Mignosi, F., and Restivo, A. (2001). Recurrence and periodicity in infinite words from local periods, *Theoret. Comput. Sci.*, *262*(1), 269–284.

Dvoretzky, A. and Motzkin, T. (1947). A problem of arrangements, *Duke Math. J.*, *14*, 305–313.

Dyck, W. (1882). Gruppentheoretische Studien, *Math. Ann.*, *20*, 1–44.

Eichelsbacher, P. and Roos, M. (1999). Compound Poisson approximation for dissociated random variables via Stein's method, *Combin. Probab. Comput.*, *8*, 335–346.

Eilenberg, S. (1974). *Automata, Languages, and Machines. Vol A.* Academic Press.

El Karoui, M., Biaudet, V., Schbath, S., and Gruss, A. (1999). Characteristics of Chi distribution on several bacterial genomes, *Res. Microbiol.*, *150*, 579–587.

Erdős, P. and Rényi, A. (1970). On a new law of large numbers, *J. Anal. Math.*, *23*, 103–111.

Erhardsson, T. (1997). *Compound Poisson approximation for Markov chains.* Ph.D. thesis, Royal Institute of Technology, Stockholm.

Erhardsson, T. (1999). Compound Poisson approximation for Markov chains using Stein's method, *Ann. Probab.*, *27*, 565–596.

Erhardsson, T. (2000). Compound Poisson approximation for counts of rare patterns in Markov chains and extreme sojourns in birth-death chains, *Ann. in Appl. Probab.*, *10*, 573–591.

Eskin, E., Gelfand, M. S., and Pevzner, P. A. (2003). Genome-wide analysis of bacterial promoter regions, In *Pacific Symposium on Biocomputing (PSB)*, pp. 29–40.

Eskin, E. and Pevzner, P. A. (2002). Finding composite regulatory patterns in DNA sequences, In *Intelligent Systems for Molecular Biology (ISMB)*, pp. 354–363.

Farach, M. (1997). Optimal suffix tree construction with large alphabets, In *38th Foundations of Computer Science (FOCS)*, pp. 137–143, Miami Beach, FL.

Feller, W. (1968). *An Introduction to Probability Theory and its Applications*, Vol. I. J. Wiley and Sons. Third edition.

Feller, W. (1971). *An Introduction to Probability Theory and its Applications*, Vol. II. J. Wiley and Sons. Second edition.

Fellows, M. R., Gramm, J., and Niedermeier, R. (2002). On the parameterized intractability of CLOSEST SUBSTRING and related problems, In Alt, H. and Ferreira, A. (Eds.), *Theoretical Aspects of Computer Science (STACS '02)*, Vol. 2285 of *Lect. Notes Comp. Sci.*, pp. 262–273. Springer-Verlag.

Ferenczi, S. (1992). Bounded remainder sets, *Acta Arith.*, *61*, 319–326.

Ferenczi, S. and Mauduit, C. (1997). Transcendence of numbers with a low complexity expansion, *J. Number Theory*, *67*, 146–161.

Fitch, W. (1975). Toward defining the course of evolution: minimum change for a specified tree topology, *Systematic Zoology*, *20*, 406–416.

Flajolet, P. (1987). Analytic models and ambiguity of context-free languages, *Theoret. Comput. Sci.*, *49*(2-3), 283–309.

Flajolet, P., Gourdon, X., and Dumas, P. (1995). Mellin transforms and asymptotics: harmonic sums, *Theoret. Comput. Sci.*, *144*, 3–58.

Flajolet, P., Guivarc'h, Y., Szpankowski, W., and Vallée, B. (2001). Hidden pattern statistics, In Oreijas, F., Spirakis, P., and Leeuwen, J. van (Eds.), *Automata, Languages and Programming (ICALP 2001)*, Vol. 2076 of *Lect. Notes Comp. Sci.*, pp. 152–165. Springer-Verlag.

Flajolet, P. and Sedgewick, R. (2002). *Analytic Combinatorics – Symbolic Combinatorics*. Technical report, INRIA.

Flajolet, P., Zimmerman, P., and Van Cutsem, B. (1994). A calculus for the random generation of labelled combinatorial structures, *Theoret. Comput. Sci.*, *132*(1-2), 1–35.

Fogg, N. P. (2002). *Substitutions in Dynamics, Arithmetics and Combinatorics*, Vol. 1794 of *Lect. Notes Math.* Springer-Verlag. Edited by V. Berthé, S. Ferenczi, C. Mauduit and A. Siegel.

Fraenkel, A. and Simpson, J. (1995). How many squares must a binary sequence contain?, *Electronic J. Comb.*, *2*(R2), 9pp.

Fraenkel, A. and Simpson, J. (1998). How many squares can a string contain?, *J. Combin. Th. A*, *82*, 112–120.

Fraenkel, A. and Simpson, J. (1999). The exact number of squares in Fibonacci words, *Theoret. Comput. Sci.*, *218*(1), 83–94.

Fredricksen, H. and Maiorana, J. (1978). Necklaces of beads in k colors and k-ary de Bruijn sequences, *Discrete Math.*, *23*(3), 207–210.

Frougny, C. and Solomyak, B. (1992). Finite beta-expansions, *Ergodic Theory Dynam. Systems*, *12*(4), 713–723.

Fu, J. C. (1993). Poisson convergence in reliability of a large linearly connected system as related to coin tossing, *Statist. Sinica*, *3*, 261–275.

Galil, Z. and Seiferas, J. (1983). Time-space optimal string matching, *J. Comput. System Sci.*, *26*(3), 280–294.

Gambaudo, J.-M., Hubert, P., Tisseur, P., and Vaienti, S. (Eds.) (2000). *Dynamical Systems (Luminy-Marseille, 1998)*. World Scientific.

Gantmacher, F. R. (1959). *The Theory of Matrices* Vols 1, 2. Chelsea. Translated from the Russian original.

Gardner, M. (1966). *New Mathematical Diversions from Scientific American*. Simon and Schuster.

Gentleman, J. (1994). The distribution of the frequency of subsequences in alphabetic sequences, as exemplified by deoxyribonucleic acid, *Appl. Statist.*, *43*, 404–414.

Gentleman, J. and Mullin, R. (1989). The distribution of the frequency of occurrence of nucleotide subsequences, based on their overlap capability, *Biometrics*, *45*, 35–52.

Geske, M. X., Godbole, A. P., Schaffner, A. A., Skolnick, A. M., and Wallstrom, G. L. (1995). Compound Poisson approximations for word patterns under Markovian hypotheses, *J. Appl. Probab.*, *32*, 877–892.

Gessel, I. M. (2000). On the number of convex polyominoes, *Ann. Sci. Math. Québec*, *24*(1), 63–66.

Giegerich, R., Kurtz, S., and Stoye, J. (1999). Efficient implementation of lazy suffix trees, In *3rd Workshop on Algorithmic Engineering (WAE99)*, Vol. 1668 of *Lect. Notes Comp. Sci.*, pp. 30–42. Springer-Verlag.

Glaz, J., Naus, J., and Wallenstein, S. (2001). *Scan Statistics*. Springer-Verlag.

Godbole, A. P. (1991). Poisson approximations for runs and patterns of rare events, *Adv. in Appl. Probab.*, *23*, 851–865.

Godbole, A. P. and Schaffner, A. A. (1993). Improved Poisson approximations for word patterns, *Adv. in Appl. Probab.*, *25*, 334–347.

Gold, E. M. (1967). Language identification in the limit, *Inform. and Control*, *10*(5), 447–474.

Good, I. (1953). The population frequencies of species and the estimation of population parameters, *Biometrica*, *40*, 237–264.

Gordon, L., Schilling, M., and Waterman, M. (1986). An extreme value theory for long head runs, *Probab. Theory Related Fields*, *72*, 279–287.

Goulden, I. P. and Jackson, D. M. (1983). *Combinatorial Enumeration*. J. Wiley and Sons.

Gouyou-Beauchamps, D. and Viennot, G. (1988). Equivalence of the two-dimensional directed animal problem to a one-dimensional path problem, *Adv. in Appl. Math.*, *9*(3), 334–357.

Gross, M. (1975). *Méthodes en syntaxe*. Hermann, Paris.

Gross, M. (1979). On the failure of generative grammar, *Language*, *55*(4), 859–885.

Gross, M. (1989). The use of finite automata in the lexical representation of natural language, In Gross, M. and Perrin, D. (Eds.), *Electronic Dictionaries and Automata in Computational Linguistics*, Vol. 377 of *Lect. Notes Comp. Sci.*, pp. 34–50. Springer-Verlag.

Gross, M. (1995). Representation of finite utterances and the automatic parsing of texts, *Language Research*, *31*(2), 291–307. Seoul National University.

Gross, M. and Lentin, A. (1967). *Notions sur les grammaires formelles*. Gauthier-Villars, Paris. 2nd edition: 1970.

Guibas, L. and Odlyzko, A. (1980). Long repetitive patterns in random sequences, *Z. Wahrsch. Verw. Gebiete*, *53*, 241–262.

Guibas, L. and Odlyzko, A. (1981a). Periods in strings, *J. Combin. Th. A*, *30*, 19–43.

Guibas, L. and Odlyzko, A. (1981b). String overlaps, pattern matching, and nontransitive games, *J. Combin. Th. A*, *30*, 183–208.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

Gusto, G. (2000). Approximation par une loi de Poisson composée de la loi du comptage d'un mot rare dans une chaîne de Markov, Master's thesis, DEA Modélisation Stochastique et Statistique, Université Paris XI, Orsay.

Guthrie, D. and Youssef, M. (1970). Empirical evaluation of some chi-square tests for the order of a Markov chain, *J. Amer. Statist. Soc.*, *65*, 631–634.

Guy, R. K., Krattenthaler, C., and Sagan, B. E. (1992). Lattice paths, reflections, & dimension-changing bijections, *Ars Combin.*, *34*, 3–15.

Gwadera, R., Atallah, M., and Szpankowski, W. (2003). Reliable detection of episodes in event sequences, In *3rd IEEE Conf. on Data Mining*, pp. 67–74. IEEE Computer Soc.

Harris, Z. S. (1952). Discourse analysis, *Language*, *28*, 1–30.

Harris, Z. S. (1970). *Papers in Structural and Transformational Linguistics.* D. Reidel, Dordrecht/Holland.

Hirano, K. and Aki, S. (1993). On number of occurrences of success runs of specified length in a two-state Markov chain, *Statist. Sinica*, *3*, 313–320.

Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem, *J. Assoc. Comput. Mach.*, *24*(4), 664–675.

Hochstättler, W., Loebl, M., and Moll, C. (1996). Generating convex polyominoes at random, *Discrete Math.*, *153*, 165–176.

Holton, C. and Zamboni, L. Q. (1998). Geometric realizations of substitutions, *Bull. Soc. Math. France*, *126*(2), 149–179.

Holton, C. and Zamboni, L. Q. (1999). Descendants of primitive substitutions, *Theory Comput. Syst.*, *32*(2), 133–157.

Hopcroft, J. (1971). An $n \log n$ algorithm for minimizing states in a finite automaton, In *Theory of Machines and Computations*, pp. 189–196. Academic Press.

Huang, H. (2002). Error bounds on multivariate normal approximations for word count statistics, *Adv. in Appl. Probab.*, *34*, 559–587.

Hui, L. (1992). Color set size problem with applications to string matching, In *Combinatorial Pattern Matching*, Vol. 644 of *Lect. Notes Comp. Sci.*, pp. 230–243. Springer-Verlag.

Hunt, J. W. and Szymanski, T. G. (1977). A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, *20*, 350–353.

Hwang, H.-K. (1996). Large deviations for combinatorial distributions I: Central limit theorems, *Ann. in Appl. Probab.*, *6*, 297–319.

Iliopoulos, C., Moore, D., and Smyth, W. (1997). A characterization of the squares in a Fibonacci string, *Theoret. Comput. Sci.*, *172*, 281–291.

Inenaga, S., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S., Mauri, G., and Pavesi, G. (2001). On-line construction of compact directed acyclic word graphs, In Amir, A. and Landau, G. M. (Eds.), *12th Combinatorial Pattern Matching,* Jerusalem, Vol. 2089 of *Lect. Notes Comp. Sci.*, pp. 169–180. Springer-Verlag.

Ito, S. (1996). Simultaneous approximations and dynamical systems (on the simultaneous approximation of (α, α^2) satisfying $\alpha^3 + k\alpha - 1 = 0$), *Sūrikaisekikenkyūsho Kōkyūroku*, *958*, 59–61. Analytic number theory (Japanese) (Kyoto, 1994).

Ito, S. (2000). On periodic expansions of cubic numbers and Rauzy fractals, In Gambaudo, J.-M., Hubert, P., Tisseur, P., and Vaienti, S. (Eds.), *Dynamical Systems (Luminy-Marseille, 1998)*, pp. 144–164. World Scientific.

Ito, S., Fujii, J.and Higashino, H., and Yasutomi, S.-i. (2003). On simultaneous approximation to (α, α^2) with $\alpha^3 + k\alpha - 1 = 0$, *J. Number Theory*, *99*(2), 255–283.

Ito, S. and Kimura, M. (1991). On Rauzy fractal, *Japan J. Indust. Appl. Math.*, *8*(3), 461–486.

Ito, S. and Ohtsuki, M. (1993). Modified Jacobi-Perron algorithm and generating Markov partitions for special hyperbolic toral automorphisms, *Tokyo J. Math.*, *16*(2), 441–472.

Ito, S. and Sano, Y. (2001). On periodic β-expansions of Pisot numbers and Rauzy fractals, *Osaka J. Math.*, *38*(2), 349–368.

Jacquet, P. and Régnier, M. (1986). Trie partitioning process: limiting distributions, In *CAAP '86 (Nice, 1986)*, Vol. 214 of *Lect. Notes Comp. Sci.*, pp. 196–210. Springer-Verlag.

Jacquet, P. and Szpankowski, W. (1991). Analysis of digital tries with Markovian dependency, *IEEE Trans. Inform. Theory*, *37*, 1470–1475.

Jacquet, P. and Szpankowski, W. (1994). Autocorrelation on words and its applications: analysis of suffix trees by string-ruler approach, *J. Combin. Th. A*, *66*, 237–269.

Jacquet, P. and Szpankowski, W. (1998). Analytical de-Poissonization and its applications, *Theoret. Comput. Sci.*, *201*, 1–62.

Jacquet, P., Szpankowski, W., and Apostol, I. (2002). A universal predictor based on pattern matching, *IEEE Trans. Inform. Theory*, *48*, 1462–1472.

Janson, S. (1997). *Gaussian Hilbert Spaces*. Cambridge University Press.

Janson, S. (to appear). Large deviations for sums of partially dependent random variables, *Random Structures & Algorithms*.

Johnson, C. D. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague.

Jonassen, I., Collins, J. F., and Higgins, D. (1995). Finding flexible patterns in unaligned protein sequences, *Protein Science*, *4*(8), 1587–1595.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall.

Justin, J. (2000). On a paper by M. Castelli, F. Mignosi, A. Restivo, *Theor. Inform. Appl.*, *34*(5), 373–377.

Justin, J. and Pirillo, G. (2002a). Episturmian words and episturmian morphisms, *Theoret. Comput. Sci.*, *276*(1-2), 281–313.

Justin, J. and Pirillo, G. (2002b). On a characteristic property of Arnoux-Rauzy sequences, *Theor. Inform. Appl.*, *36*(4), 385–388 (2003).

Karhumäki, J. (1983). On cube-free ω-words generated by binary morphisms, *Discr. Appl. Math.*, *5*(3), 279–297.

Kärkkäinen, J. and Sanders, P. (2003). Simple linear work suffix array constructio, In Baeten, J. C. M., Lenstra, J. K., Parrow, J., and Woeginger, G. J. (Eds.), *30th Automata, Languages and Programming (ICALP '03)*, Vol. 2719 of *Lect. Notes Comp. Sci.*, pp. 943–955. Springer-Verlag.

Karlin, S., Burge, C., and Campbell, A. M. (1992). Statistical analyses of counts and distributions of restriction sites in DNA sequences, *Nucl. Acids Res.*, *20*, 1363–1370.

Karlin, S. and Dembo, A. (1992). Limit distributions of maximal segmental score among Markov dependent partial sums, *Adv. in Appl. Probab.*, *24*, 113–140.

Karlin, S. and Macken, C. (1991). Assessment of inhomogeneities in an *E. coli* physical map, *Nucl. Acids Res.*, *19*, 4241–4246.

Karlin, S. and Ost, F. (1987). Counts of long aligned word matches among random letter sequences, *Adv. in Appl. Probab.*, *19*, 293–351.

Karlin, S. and Taylor, H. (1975). *A First Course in Stochastic Processes* (Second edition). Academic Press.

Kato, T. (1980). *Perturbation Theory for Linear Operators*. Springer-Verlag.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer, *IEEE Trans. Acoust. Speech Signal Process.*, *35*, 400–401.

Kim, D. K., Sim, J. S., Park, H., and Park, K. (2003). Linear-time construction of suffix arrays, In Baeza-Yates, R., Chávez, E., and Crochemore, M. (Eds.), *Combinatorial Pattern Matching (CPM 2003)*, Vol. 2676 of *Lect. Notes Comp. Sci.*, pp. 186–199. Springer-Verlag.

Klarner, D. A. (1997). Polyominoes, In *Handbook of Discrete and Computational Geometry*, CRC Press Ser. Discrete Math. Appl., pp. 225–240. CRC.

Kleffe, J. and Borodovsky, M. (1992). First and second moment of counts of words in random texts generated by Markov chains, *Comput. Appl. Biosci.*, *8*, 433–441.

Kleffe, J. and Langbecker, U. (1990). Exact computation of pattern probabilities in random sequences generated by Markov chains, *Comput. Appl. Biosci.*, *6*, 347–353.

Knuth, D. E. (1965). On the translation of languages from left to right, *Inform. and Control*, *8*, 607–639.

Knuth, D. E. (1988). Fibonacci multiplication, *Appl. Math. Lett.*, *1*(1), 57–60.

Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977). Fast pattern matching in strings, *SIAM J. Comput.*, *6*, 323–350.

Ko, P. and Aluru, S. (2003). Space-efficient linear-time construction of suffix arrays, In Baeza-Yates, R., Chávez, E., and Crochemore, M. (Eds.), *Combinatorial Pattern Matching (CPM 2003)*, Vol. 2676 of *Lect. Notes Comp. Sci.*, pp. 200–210. Springer-Verlag.

Kolpakov, R., Bana, G., and Kucherov, G. (2003). `mreps`: efficient and flexible detection of tandem repeats in DNA, *Nucl. Acids Res.*, *31*(13), 3672–3678.

Kolpakov, R. and Kucherov, G. (1999). Finding maximal repetitions in a word in linear time, In *40th Symp. Foundations of Computer Science (FOCS)*, pp. 596–604. IEEE Computer Society Press.

Kolpakov, R. and Kucherov, G. (2000a). Finding repeats with fixed gap, In *7th Symp. String Processing and Information Retrieval (SPIRE)*, pp. 162–168. IEEE Computer Society Press.

Kolpakov, R. and Kucherov, G. (2000b). On maximal repetitions in words, *J. Discrete Algorithms*, *1*(1), 159–186.

Kolpakov, R. and Kucherov, G. (2003). Finding approximate repetitions under Hamming distance, *Theoret. Comput. Sci.*, *33*(1), 135–156.

Kosaraju, S. (1994). Computation of squares in string, In Crochemore, M. and Gusfield, D. (Eds.), *5th Combinatorial Pattern Matching*, Vol. 807 of *Lect. Notes Comp. Sci.*, pp. 146–150. Springer-Verlag.

Koskenniemi, K. (1983). *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*. University of Helsinki, Department of General Linguistics. Publications, no. 11.

Kucherov, G., Ochem, P., and Rao, M. (2003). How many square occurrences must a binary sequence contain, *Electronic J. Comb.*, *10*(1), 11pp.

Kucherov, G. and Rusinowitch, M. (1997). Matching a set of strings with variable length don't cares, *Theoret. Comput. Sci.*, *178*, 129–154.

Kuich, W. and Salomaa, A. (1986). *Semirings, Automata, Languages.* Springer-Verlag.

Landau, G. and Schmidt, J. (1993). An algorithm for approximate tandem repeats, In Apostolico, A., Crochemore, M., Galil, Z., and Manber, U. (Eds.), *4th Combinatorial Pattern Matching (Padova)*, Vol. 684 of *Lect. Notes Comp. Sci.*, pp. 120–133. Springer-Verlag, Padova, Italy.

Laporte, E. (1997). Rational transductions for phonetic conversion and phonology, In Roche, E. and Schabès, Y. (Eds.), *Finite-state Language Processing*, Language, Speech, and Communication Series, chap. 14, pp. 407–428. MIT Press.

Lee, K.-F. (1990). Context dependent phonetic hidden Markov models for continuous speech recognition, *IEEE Trans. Acoust. Speech Signal Process.*, *38*(4), 599–609.

Lempel, A. and Ziv, J. (1976). On the complexity of finite sequences, *IEEE Trans. Inform. Theory*, *IT-22*, 75–81.

Leung, M. Y., Marsh, G. M., and Speed, T. P. (1996). Over and underrepresentation of short DNA words in Herpesvirus genomes, *J. Comput. Biol.*, *3*, 345–360.

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals, *Soviet Physics Dokl.*, *10*, 707–710.

Li, S.-Y. (1980). A martingale approach to the study of occurrence of sequence patterns in repeated experiments, *Ann. Probab.*, *8*, 1171–1176.

Lind, D. and Marcus, B. (1996). *An Introduction to Symbolic Dynamics and Coding.* Cambridge University Press.

Lint, J. H. van and Wilson, R. M. (1992). *A Course in Combinatorics*. Cambridge University Press.

Lonardi, S. (2001). *Global detectors of unusual words design, implementation and applications to pattern discovery in biosequences*. Ph.D. thesis, Purdue University, West Lafayette, Indiana.

Lothaire, M. (1997). *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press. Reprint with corrections of the 1983 original edition.

Lothaire, M. (2002). *Algebraic Combinatorics on Words*, Vol. 90 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Lundstrom, R. (1990). *Stochastic models and statistical methods for DNA sequence data*. Ph.D. thesis, University of Utah.

Mahmoud, H. (1992). *Evolution of Random Search Trees*. J. Wiley and Sons.

Main, M. (1989). Detecting leftmost maximal periodicities, *Discr. Appl. Math.*, *25*, 145–153.

Main, M. and Lorentz, R. (1984). An $O(n \log n)$ algorithm for finding all repetitions in a string, *J. Algorithms*, *5*(3), 422–432.

Main, M. and Lorentz, R. (1985). Linear time recognition of square free strings, In Apostolico, A. and Galil, Z. (Eds.), *Combinatorial Algorithms on Words*, Vol. 12 of *NATO Advanced Science Institutes, Series F*, pp. 272–278. Springer-Verlag.

Manber, U. and Myers, G. (1993). Suffix arrays: a new method for on-line string searches, *SIAM J. Comput.*, *22*(5), 935–948.

Manzini, G. (2001). An analysis of the Burrows-Wheeler transform, *J. Assoc. Comput. Mach.*, *48*(3), 407–430.

Marsan, L. and Sagot, M.-F. (2000a). Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification, *J. Comput. Biol.*, *7*, 345–362.

Marsan, L. and Sagot, M.-F. (2000b). Extracting structured motifs using a suffix tree. Algorithms and application to consensus identification, In *4th Research in Computational Molecular Biology (RECOMB)*, pp. 210–219. ACM Press.

Marsan, L. and Sagot, M.-F. (2001). Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory consensus identification, *J. Comput. Biol.*, *7*, 345–360.

McCluer, C. R. (2000). The many proofs and applications of Perron's theorem, *SIAM Review*, *42*, 487–498.

McCreight, E. M. (1976). A space-economical suffix tree construction algorithm, *J. Assoc. Comput. Mach.*, *23*(2), 262–272.

McEliece, R. J. (2002). *The Theory of Information and Coding* (2nd edition)., Vol. 86 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Mehldau, G. and Myers, E. (1993). A system for pattern matching applications on biosequences, *Computer Applied Bioscience (CABIOS)*, *9*(3), 299–314.

Messaoudi, A. (1998). Propriétés arithmétiques et dynamiques du fractal de Rauzy, *J. Théor. Nombres Bordeaux*, *10*(1), 135–162.

Messaoudi, A. (2000a). Frontière du fractal de Rauzy et système de numération complexe, *Acta Arith.*, *95*(3), 195–224.

Messaoudi, A. (2000b). Généralisation de la multiplication de Fibonacci, *Math. Slovaca*, *50*(2), 135–148.

Messaoudi, A. (2002). Tribonacci multiplication, *Appl. Math. Lett.*, *15*(8), 981–985.

Mignosi, F. and Pirillo, G. (1992). Repetitions in the Fibonacci infinite word, *Theoret. Inform. Appl.*, *26*(3), 199–204.

Mignosi, F., Restivo, A., and Salemi, S. (1995). A periodicity theorem on words and applications, In *20th Mathematical Foundations of Computer Science (MFCS)*, Vol. 969 of *Lect. Notes Comp. Sci.*, pp. 337–348. Springer-Verlag.

Mignosi, F. and Zamboni, L. Q. (2002). On the number of Arnoux-Rauzy words, *Acta Arith.*, *101*(2), 121–129.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users, In Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. 2, chap. 13, pp. 419–491. Wiley, New York.

Mohri, M. (1994). Minimization of sequential transducers, In Crochemore, M. and Gusfield, D. (Eds.), *Combinatorial Pattern Matching 94*, Vol. 807 of *Lect. Notes Comp. Sci.*, pp. 151–163. Springer-Verlag.

Mohri, M. (1997). Finite-state transducers in language and speech processing, *Comput. Linguistics*, *23*(2), 269–311.

Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems, *J. Autom. Lang. Comb.*, *7*(3), 321–350.

Mohri, M., Pereira, F., and Riley, M. (2000). The design principles of a weighted finite-state transducer library, *Theoret. Comput. Sci.*, *231*(1), 17–32.

Mohri, M., Pereira, F. C. N., and Riley, M. (1996). Weighted automata in text and speech processing, In *12th European Conf. Artificial Intelligence (ECAI 1996), Workshop on Extended finite state models of language, Budapest, Hungary*. J. Wiley and Sons.

Mohri, M., Pereira, F. C. N., and Riley, M. (2002). Weighted finite-state transducers in speech recognition, *Computer Speech and Language*, *16*(1), 69–88. Available at `http://www.research.att.com/~mohri/postscript/csl01.ps`.

Moody, R. V. (1997). Meyer sets and their duals, In Moody, R. V. (Ed.), *The Mathematics of Long-Range Aperiodic Order*, Vol. 13 of *CRM Monograph Series*, pp. 403–441. Kluwer.

Moore, E. F. (1956). Gedanken-experiments on sequential machines, In *Automata Studies*, Vol. 34 of *Annals of Mathematics Studies*, pp. 129–153. Princeton University Press.

Muri, F. (1998). Modelling Bacterial Genomes using Hidden Markov Models, In Payne, R. and Green, P. (Eds.), *Compstat'98*, pp. 89–100. Physica-Verlag.

Nicodème, P., Salvy, B., and Flajolet, P. (2002). Motif statistics, *Theoret. Comput. Sci.*, *287*(2), 593–617. Algorithms (Prague, 1999).

Nijenhuis, A. and Wilf, H. S. (1978). *Combinatorial Algorithms* (2nd edition). Academic Press.

Nuel, G. (2001). *Grandes déviations et chaînes de Markov pour l'étude des mots exceptionnels dans les séquences biologiques.* Ph.D. thesis, Université d'Evry Val d'Essonne.

Parida, L., Rigoutsos, I., Floratos, A., Platt, D., and Gao, Y. (2000). Pattern discovery on character sets and real-valued data: Linear bound on irredundant motifs and efficient polynomial time algorithm, In *11th SIAM Symposium on Discrete Algorithms (SODA)*, pp. 297–308. ACM Press.

Parida, L., Rigoutsos, I., and Platt, D. (2001). An output-sensitive flexible pattern discovery algorithm, In Amir, A. and Landau, G. (Eds.), *12th Combinatorial Pattern Matching*, Vol. 2089 of *Lect. Notes Comp. Sci.*, pp. 131–142. Springer-Verlag.

Paumier, S. (2001). Some remarks on the application of a lexicon-grammar, *Lingvisticae Investigationes*, *24*(2), 245–256.

Pavesi, G., Mauri, G., and Pesole, G. (2001a). An algorithm for finding signals of unknown length in DNA sequences, *Bioinformatics*, *17*(Suppl. 1), 207–214.

Pavesi, G., Mauri, G., and Pesole, G. (2001b). Methods for pattern discovery in unaligned biological sequences, *Briefings in Bioinformatics*, *2*(4), 417–430.

Paz, A. (1971). *Introduction to Probabilistic Automata.* Academic Press.

Pereira, F. C. N. and Riley, M. D. (1997). Speech recognition by composition of weighted finite automata, In *Finite-State Language Processing*, pp. 431–453. MIT Press.

Petersen, H. (1994). The ambiguity of primitive words, In Enjalbert, P., Mayr, E. W., and Wagner, K. W. (Eds.), *Theoretical Aspects of Computer Science (STACS '94)*, Vol. 775 of *Lect. Notes Comp. Sci.*, pp. 679–690. Springer-Verlag.

Petersen, H. (1996). On the language of primitive words, *Theoret. Comput. Sci.*, *161*, 141–156.

Pevzner, P. A. (1989). l-tuple DNA sequencing: computer analysis, *J. Biomol. Struct. Dynamics*, *7*, 63–73.

Pevzner, P. A. (1995). DNA physical mapping and alternating Eulerian cycles in colored graphs, *Algorithmica*, *13*, 77–105.

Pevzner, P. A. (2000). *Computational Molecular Biology. An Algorithmic Approach.* The MIT Press.

Pevzner, P. A. and Sze, S.-H. (2000). Combinatorial approaches to finding subtle signals in DNA sequences, In *Intelligent Systems for Molecular Biology (ISMB)*, pp. 269–278.

Pin, J.-E. (1986). *Varieties of Formal Languages*. North Oxford.

Pirillo, G. (1997). Fibonacci numbers and words, *Discrete Math.*, *173*(1-3), 197–207.

Pisanti, N., Crochemore, M., Grossi, R., and Sagot, M.-F. (2003). A basis of tiling motifs for generating repeated patterns and its complexity for higher quorum, In Rovan, B. and Vojtás, P. (Eds.), *Mathematical Foundations of Computer Science (MFCS 2003)*, Vol. 2747 of *Lect. Notes Comp. Sci.*, pp. 622–631. Springer-Verlag.

Pitman, J. (1998). Enumerations of trees and forests related to branching processes and random walks, In Aldous, D. and Propp, J. (Eds.), *Microsurveys in Discrete Probability*, Vol. 41 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.* Amer. Math. Soc.

Poulalhon, D. and Schaeffer, G. (2003). Optimal coding and sampling of triangulations, In Baeten, J. C. M., Lenstra, J. K., Parrow, J., and Woeginger, G. J. (Eds.), *30th Automata, Languages and Programming (ICALP '03)*, Vol. 2719 of *Lect. Notes Comp. Sci.*, pp. 1080–1094. Springer-Verlag.

Praggastis, B. (1999). Numeration systems and Markov partitions from self-similar tilings, *Trans. Amer. Math. Soc.*, *351*(8), 3315–3349.

Prum, B., Rodolphe, F., and Turckheim, É. (1995). Finding words with unexpected frequencies in deoxyribonucleic acid sequences, *J. Roy. Statist. Soc. Ser. B*, *57*, 205–220.

Queffélec, M. (1987). *Substitution Dynamical Systems–Spectral Analysis*, Vol. 1294 of *Lect. Notes Math.* Springer-Verlag.

Rabiner, L. (1989). A tutorial on hidden Markov models, *Proceedings of the IEEE*, *77*(2), 257–286.

Raffinot, M. (1997). Asymptotic estimation of the average number of terminal states in DAWGs, In Baeza-Yates, R. (Ed.), *4th South American Workshop on String Processing*, pp. 140–148. Carleton University Press, Valparaiso, Chile.

Rajarshi, M. B. (1974). Success runs in a two-state Markov chain, *J. Appl. Probab.*, *11*, 190–192.

Raney, G. N. (1960). Functional composition patterns and power series reversion, *Trans. Amer. Math. Soc.*, *94*, 441–451.

Rauzy, G. (1982). Nombres algébriques et substitutions, *Bull. Soc. Math. France*, *110*(2), 147–178.

Rauzy, G. (1988). Rotations sur les groupes, nombres algébriques, et substitutions, In *Sém. Théor. Nombres (Talence, 1987–1988)*. Univ. Bordeaux I. Exp. No. 21.

Rauzy, G. (1990). Sequences defined by iterated morphisms, In *Sequences (Naples/Positano, 1988)*, pp. 275–286. Springer, New York.

Régnier, M. (2000). A unified approach to word occurrence probabilities, *Discr. Appl. Math.*, *104*, 259–280.

Régnier, M. and Szpankowski, W. (1998a). On pattern frequency occurrences in a Markovian sequence, *Algorithmica*, *22*, 631–649.

Régnier, M. and Szpankowski, W. (1998b). On the approximate pattern occurrences in a text, In *Compression and Complexity of Sequences 97*, pp. 253–264. IEEE Computer Society Press.

Reinert, G. and Schbath, S. (1998). Compound Poisson and Poisson process approximations for occurrences of multiple words in Markov chains, *J. Comput. Biol.*, *5*, 223–253.

Reinert, G., Schbath, S., and Waterman, M. (2000). Probabilistic and statistical properties of words: an overview, *J. Comput. Biol.*, *7*, 1–46.

Rensburg, E. J. J. van (2000). *The Statistical Mechanics of Interacting Walks, Polygons, Animals and Vesicles*, Vol. 18 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press.

Reutenauer, C. (1990). Subsequential functions: characterizations, minimization, examples, In Dassow, J. and Kelemen, J. (Eds.), *6th International Meeting of Young Computer Scientists*, Vol. 464 of *Lect. Notes Comp. Sci.*, pp. 62–79. Springer-Verlag.

Revuz, D. (1992). Minimisation of acyclic deterministic automata in linear time, *Theoret. Comput. Sci.*, *92*(1), 181–189.

Rice, J. (1995). *Mathematical Statistics and Data Analysis*. Duxbury Press.

Ridout, D. (1957). Rational approximations to algebraic numbers, *Mathematika*, *4*, 125–131.

Rinott, Y. and Rotar, V. (1996). On coupling constructions and rates in the CLT for dependent summands with applications to the antivoter model and weighted U-statistics, *J. Multivariate Anal.*, *56*, 333–350.

Risley, R. N. and Zamboni, L. Q. (2000). A generalization of Sturmian sequences: combinatorial structure and transcendence, *Acta Arith.*, *95*(2), 167–184.

Robin, S. (2002). A compound Poisson model for words occurrences in DNA sequences, *J. Roy. Statist. Soc. Ser. C*, *51*, 437–451.

Robin, S. and Daudin, J.-J. (1999). Exact distribution of word occurrences in a random sequence of letters, *J. Appl. Probab.*, *36*, 179–193.

Robin, S. and Daudin, J.-J. (2001). Exact distribution of the distances between any occurrences of a set of words, *Ann. Inst. Statist. Math.*, *36*(4), 895–905.

Robin, S., Daudin, J.-J., Richard, H., Sagot, M.-F., and Schbath, S. (2002). Occurrence probability of structured motifs in random sequences, *J. Comput. Biol.*, *9*, 761–773.

Robin, S. and Schbath, S. (2001). Numerical comparison of several approximations of the word count distribution in random sequences, *J. Comput. Biol.*, *8*, 349–359.

Rocha, E., Viari, A., and Danchin, A. (1998). Oligonucleotide bias in *Bacillus subtilis*: general trends and taxonomic comparisons, *Nucl. Acids Res.*, *26*, 2971–2980.

Roche, E. (1997). Compact factorization of finite-state transducers and finite-state automata, *Nordic J. Comput.*, *4*(2), 187–216.

Rodeh, M., Pratt, V. R., and Even, S. (1981). Linear algorithm for data compression via string matching, *J. Assoc. Comput. Mach.*, *28*(1), 16–24.

Roos, M. (1993). *Stein-Chen method for compound Poisson approximation*. Ph.D. thesis, University of Zurich.

Rudander, J. (1996). *On the first occurrence of a given pattern in a semi-Markov process*. Ph.D. thesis, Uppsala, Sweden.

Sagot, M.-F. (1996). *Ressemblance lexicale et structurale entre macromolécules – Formalisation et approches combinatoires*. Ph.D. thesis, Université de Marne-la-Vallée, Noisy le Grand, France. Thèse de doctorat.

Sagot, M.-F. (1998). Spelling approximate repeated or common motifs using a suffix tree, In Lucchesi, C. and Moura, A. (Eds.), *LATIN'98: Theoretical Informatics: Third Latin American Symposium*, Vol. 1380 of *Lect. Notes Comp. Sci.*, pp. 111–127. Springer-Verlag.

Sagot, M.-F. and Myers, E. (1998). Identifying satellites and periodic repetitions in biological sequences, *J. Comput. Biol.*, *5*(3), 539–554.

Sagot, M.-F., Soldano, H., and Viari, A. (1995). A distance-based block searching algorithm, In *Intelligent Systems for Molecular Biology (ISMB)*, pp. 322–331.

Sagot, M.-F. and Viari, A. (1996). A double combinatorial approach to discovering patterns in biological sequences, In Hirschberg, D. and Myers, G. (Eds.), *7th Combinatorial Pattern Matching*, Vol. 1075 of *Lect. Notes Comp. Sci.*, pp. 186–208. Springer-Verlag.

Sagot, M.-F., Viari, A., Pothier, J., and Soldano, H. (1995). Finding flexible patterns in a text - an application to 3D molecular matching, *Computer Applied Bioscience (CABIOS)*, *11*(1), 59–70.

Sagot, M.-F., Viari, A., and Soldano, H. (1997). Multiple sequence comparison - A peptide matching approach, *Theoret. Comput. Sci.*, *180*(1-2), 115–137.

Sakarovich, J. (2004). *Éléments de théorie des automates*. Vuibert.

Salomaa, A. and Soittola, M. (1978). *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag.

Salton, G. (1989). *Automatic Text Processing*. Addison-Wesley.

Sankoff, D. and Kruskal, J. B. (Eds.) (1983). *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley.

Sano, Y. (2002). On purely periodic β-expansions of Pisot numbers, *Nagoya Math. J.*, *166*, 183–207.

Sano, Y., Arnoux, P., and Ito, S. (2001). Higher dimensional extensions of substitutions and their dual maps, *J. Anal. Math.*, *83*, 183–206.

Sapir, E. (1921). *Language: An Introduction to the Study of Speech*. Harcourt, Brace and World.

Schaeffer, G. (1997). Bijective census and random generation of Eulerian planar maps with prescribed vertex degrees, *Electronic J. Comb.*, *4*(1), Research Paper 20, 14 pp.

Schbath, S. (1995a). Compound Poisson approximation of word counts in DNA sequences, *ESAIM Probab. Statist.*, *1*, 1–16. (`http://www.emath.fr/ps/`).

Schbath, S. (1995b). *Étude asymptotique du nombre d'occurrences d'un mot dans une chaîne de Markov et application à la recherche de mots de fréquence exceptionnelle dans les séquences d'ADN*. Ph.D. thesis, Université René Descartes, Paris V.

Schmidt, K. (1980). On periodic expansions of Pisot numbers and Salem numbers, *Bull. London Math. Soc.*, *12*, 269–278.

Schützenberger, M.-P. (1961). A remark on finite transducers, *Inform. and Control*, *4*, 185–196.

Schützenberger, M.-P. (1964). On the synchronizing properties of certain prefix codes, *Inform. and Control*, *7*, 23–36.

Schützenberger, M.-P. (1977). Sur une variante des fonctions séquentielles, *Theoret. Comput. Sci.*, *4*(1), 47–57.

Schützenberger, M.-P. and Chomsky, N. (1963). The algebraic theory of context-free languages, In *Computer Programming and Formal Systems*, pp. 118–161. North Holland.

Sedgewick, R. (1983). *Algorithms*. Addison-Wesley.

Sedgewick, R. and Flajolet, P. (1995). *An Introduction to the Analysis of Algorithms*. Addison-Wesley.

Senellart, J. (1998). Reconnaissance automatique des entrées du lexique-grammaire des phrases figées, *Travaux de linguistique*, *37*, 109–125.

Senellart, J. (1999). *Outils de reconnaissance d'expressions linguistiques complexes dans de grands corpus*. Ph.D. thesis, LADL, University of Paris 7.

Senoussi, R. (1990). Statistique asymptotique presque-sûre de modèles statistiques convexes, *Ann. Inst. Henri Poincaré*, *26*, 19–44.

Shamir, R. and Tsur, D. (2001). Large scale sequencing by hybridization, In *5th Research in Computational Molecular Biology (RECOMB)*, pp. 269–277. ACM Press.

Shannon, C. E. (1948). A mathematical theory of communication, *Bell System Tech. J.*, *27*, 379–423, 623–656.

Shannon, C. E. (1951). Prediction and entropy of printed English, *Bell System Tech. J.*, *30*, 50–64.

Shapiro, H. S. (1952). Extremal problems for polynomials and power series, Master's thesis, MIT.

Shields, P. C. (1969). *The Ergodic Theory of Discrete Sample Paths*. Amer. Math. Soc.

Shiloach, Y. (1981). Fast canonization of circular strings, *J. Algorithms*, *2*(2), 107–121.

Silberztein, M. (1994). INTEX: a corpus processing system, In *15th Conf. Comput. Linguistics (COLING'94)*, Vol. 1, pp. 579–583, Kyoto, Japan. Available at `http://acl.ldc.upenn.edu/C/C94/C94-1095.pdf`.

Sirvent, V. F. (2000a). The common dynamics of the Tribonacci substitutions, *Bull. Belg. Math. Soc. Simon Stevin*, *7*(4), 571–582.

Sirvent, V. F. (2000b). Geodesic laminations as geometric realizations of Pisot substitutions, *Ergodic Theory Dynam. Systems*, *20*(4), 1253–1266.

Sirvent, V. F. and Wang, Y. (2002). Self-affine tiling via substitution dynamical systems and Rauzy fractals, *Pacific J. Math.*, *206*(2), 465–485.

Slisenko, A. (1983). Detection of periodicities and string matching in real time, *J. Soviet Math.*, *22*, 1316–1386. translation from the Russian ordiginal.

Soldano, H., Viari, A., and Champesme, M. (1995). Searching for flexible repeated patterns using a non transitive similarity relation, *Pattern Recognition Letters*, *16*(3), 233–246.

Sourice, S., Biaudet, V., El Karoui, M., Ehrlich, S., and Gruss, A. (1998). Identification of the Chi site of *Haemophilus influenzae* as several sequences related to the *Escherichia coli* Chi site, *Mol. Microbiol.*, *27*, 1021–1029.

Sproat, R. (1997). Multilingual text analysis for text-to-speech synthesis, *J. Natural Language Engineering*, *2*(4), 369–380.

Stanley, R. P. (1999). *Enumerative Combinatorics. Vol. II*, Vol. 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.

Stefanov, V. (2003). The intersite distances between pattern occurrences in strings generated by general discrete - and continuous- time models: an algorithmic approach, *J. Appl. Probab.*, *40*.

Stein, C. (1972). A bound for the error in the normal approximation to the distribution of a sum of dependent random variables, In *Proc. Sixth Berkeley Symp. Math. Statist. Probab.*, Vol. 2, pp. 583–602. Univ. California Press, Berkeley.

Storer, J. A. (1988). *Data Compression: Methods and Theory*. Computer Science Press.

Stoye, J. and Gusfield, D. (1998). Simple and flexible detection of contiguous repeats using a suffix tree, In Farach-Colton, M. (Ed.), *9th Combinatorial Pattern Matching*, No. 1448 in Lect. Notes Comp. Sci., pp. 140–152. Springer-Verlag.

Szpankowski, W. (1993a). Asymptotic properties of data compression and suffix trees, *IEEE Trans. Inform. Theory*, *39*, 1647–1659.

Szpankowski, W. (1993b). A generalized suffix tree and its (un)expected asymptotic behaviors, *SIAM J. Comput.*, *22*, 1176–1198.

Szpankowski, W. (2001). *Average Case Analysis of Algorithms on Sequences*. J. Wiley and Sons.

Tanushev, M. S. (1996). Central limit theorem for several patterns in a Markov chain sequence of letters,. Preprint.

Tanushev, M. S. and Arratia, R. (1997). Central limit theorem for renewal theory for several patterns, *J. Comput. Biol.*, *4*, 35–44.

Thompson, K. (1968). Regular expression search algorithm, *Comm. Assoc. Comput. Mach.*, *11*, 419–422.

Thurston, W. P. (1989). Groups, tilings and finite state automata, Lectures notes distributed in conjunction with the Colloquium Series, in *AMS Colloquium lectures*.

Tutte, W. T. (1962). A census of planar triangulations, *Canad. J. Math.*, *14*, 21–38.

Ukkonen, E. (1992). Approximate string-matching with q-grams and maximal matches, *Theoret. Comput. Sci.*, *92*, 191–211.

Ukkonen, E. (1995). On-line construction of suffix trees, *Algorithmica*, *14*(3), 249–260.

Valiant, L. G. (1984). A theory of the learnable, *Comm. Assoc. Comput. Mach.*, *27*(11), 1134–1142.

Vallée, B. (2001). Dynamical sources in information theory: fundamental intervals and word prefixes, *Algorithmica*, *29*, 262–306.

Vanet, A., Marsan, L., and Sagot, M.-F. (1999). Promoter sequences and algorithmical methods for identifying them, *Res. Microbiol.*, *150*(1), 1–21.

Verger Gaugry, J.-L. and Gazeau, J.-P. (2004). Geometric study of the beta-integers for a Perron number and mathematical quasicrystals, to appear in *J. Théor. Nombres Bordeaux*.

Vidal, J. and Mosseri, R. (2000). Generalized Rauzy tilings: construction and electronic properties, *Materials Science and Engineering A*, *294–296*, 572–575.

Vidal, J. and Mosseri, R. (2001). Generalized quasiperiodic Rauzy tilings, *J. Phys. A*, *34*(18), 3927–3938.

Viennot, G. X. (1986). Heaps of pieces. I. Basic definitions and combinatorial lemmas, In *Combinatoire énumérative (Montréal, 1985)*, Vol. 1234 of *Lect. Notes Math.*, pp. 321–350. Springer-Verlag.

Wade, L. I. (1941). Certain quantities transcendental over $GF(p^n, x)$, *Duke Math. J.*, *8*, 701–720.

Waterman, M. S. (1995). *Introduction to Computational Biology. Maps, Sequences and Genomes*. Chapman and Hall.

Weber, A. and Klemm, R. (1995). Economy of description for single-valued transducers, *Inform. and Comput.*, *118*(2), 327–340.

Weiner, P. (1973). Linear pattern matching algorithm, In *14th IEEE Symposium on Switching and Automata Theory (SWAT)*, pp. 1–11, Washington, DC.

Welsh, D. (1988). *Codes and Cryptography*. The Clarendon Press Oxford University Press.

Wharton, R. M. (1974). Approximate language identification, *Inform. and Control*, *26*, 236–255.

Wozny, N. and Zamboni, L. Q. (2001). Frequencies of factors in Arnoux-Rauzy sequences, *Acta Arith.*, *96*(3), 261–278.

Wu, S. and Manber, U. (1995). Fast text searching allowing errors, *Comm. Assoc. Comput. Mach.*, *35*, 983–991.

Wyner, A. and Ziv, J. (1989). Some asymptotic properties of the entropy of a stationary ergodic dat a source with applications to data compression, *IEEE Trans. Inform. Theory*, *35*, 1250–1258.

Wyner, A. J. (1997). The redundancy and distribution of the phrase lengths of the fixed-da tabase Lempel-Ziv algorithm, *IEEE Trans. Inform. Theory*, *43*, 1439–1465.

Yang, E. and Kieffer, J. (1998). On the performance of data compression algorithms based upon string matching, *IEEE Trans. Inform. Theory*, *44*, 47–65.

Zamboni, L. Q. (1998). Une généralisation du théorème de Lagrange sur le développement en fraction continue, *C. R. Acad. Sci. Paris, Série I*, *327*, 527–530.

Zipf, G. K. (1935). *Psycho-Biology of Languages.* Houghton-Mifflin. Republished by MIT Press, 1965.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression, *IEEE Trans. Inform. Theory*, *IT-23*, 337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding, *IEEE Trans. Inform. Theory*, *24*, 530–536.

Ziv, J. and Merhav, N. (1993). A measure of relative entropy between individual sequences with application to universal classification, *IEEE Trans. Inform. Theory*, *39*, 1270–1279.

General Index

B

C