Minimal Forbidden Words and Applications

Gabriele Fici

UNIVERSITÉ DE MARNE-LA-VALLÉE

École Doctorale ICMS
Information, Communication, Modélisation et Simulation

Institut Gaspard-Monge - UMR8049
Laboratoire d'Informatique

THÈSE

Minimal Forbidden Words and Applications

dirigée par

Marie-Pierre BÉAL et Filippo MIGNOSI

présentée par

Gabriele FICI

pour l'obtention du titre de

Docteur de l'Université de Marne-la-Vallée

specialté : Informatique

soutenue publiquement le 13 février 2006 devant le jury composé par :

Président :	Jean-Marc FÉDOU	-	Université de Nice - Sophia Antipolis
Rapporteurs :	Christiane FROUGNY	-	Université Paris 8
	Arturo CARPI	-	Università di Perugia
Directeurs :	Marie-Pierre BÉAL	-	Université de Marne-la-Vallée
	Filippo MIGNOSI	-	Università di Palermo
Examinateur :	Elena BARCUCCI	-	Università di Firenze

Acknowledgements

I am grateful to my parents who allowed me to make the realization of this thesis possible.

I want to thank Marie Pierre Béal and Filippo Mignosi for their guidance and their teaching during these years. I am also thankful to Antonio Restivo for his many suggestions and his support.

A special acknowledgement goes to the Institut Gaspard-Monge and their members for supporting my activity in Paris during these years.

A final thank goes to all people who supported me during these years and with which I shared a lot of marvellous moments, in particular Francesca Fiorenzi, Daniele and Fabio Otera, Giuseppe Cabibbo, Gianluca Fiandaca, Alireza Taheri, Damien Steciuk and all my friends of the Cité Universitaire de Paris.

Abstract

This thesis describes the theory and some applications of minimal forbidden words, that are the most little words that do not appear as factors of a given word.

In the first part of this thesis, we describe the properties of minimal forbidden words and we show some particular cases, as that of a finite word, a finite set of finite words, and a regular factorial language. We also present the procedures for the computation of the theoretical results.

Then we generalize the minimal forbidden words to the case of the existence of a period, which determines the positions of occurrences of the factors modulo a fixed integer. These are called minimal periodic forbidden words. We study their basic properties and give the algorithms for the computation in the case of a finite word and of a finite set of finite words.

In the second part we show two applications of minimal forbidden words.

The first one is related to constrained systems. We give a polynomial-time construction of the set of sequences that satisfy a constraint defined by a finite list of forbidden blocks, with a specified set of bit positions unconstrained. We also give a linear-time construction of a finite-state presentation of a constrained system defined by a periodic list of forbidden blocks.

The second one is a problem issued from biology: the reconstruction of a genomic sequence starting from a set of its fragments. We show that a theoretical formalization of this problem can be solved in linear time using minimal forbidden words. We also prove that our algorithm solves a special case of the Shortest Superstring Problem.

Résumé

Dans cette thèse nous traitons des mots interdits minimaux, qui sont les plus petits mots qui n'apparaissent pas comme facteur d'un mot donné, et de leurs applications.

Dans la première partie de la thèse nous exposons les propriétés des mots interdits minimaux, et nous considérons quelques cas particuliers, comme celui d'un mot fini, d'un ensemble fini de mots finis, et d'un langage factoriel régulier. Nous présentons aussi les procédures pour le calcul des objets considérés.

Ensuite, nous généralisons les mots interdits minimaux au cas de l'existence d'une période, qui détermine les positions des occurrences des facteurs modulo un entier fixé. Ceux-ci sont appelés mots interdits minimaux périodiques. Nous étudions leurs propriétés principales et avec des algorithmes de test de ces propriétés.

Dans la deuxième partie de la thèse nous montrons deux applications des mots interdits minimaux.

La première est reliée aux systèmes contraints. Nous donnons une construction en temps polynomial de l'ensemble des séquences qui satisfont la contrainte définie par une liste finie de blocs interdits, avec un ensemble spécifié de positions de bit sans contrainte. Nous donnons aussi une construction en temps linéaire d'une présentation à états finis d'un système contraint défini par une liste périodique de blocs interdits.

La deuxième application est relative à un problème de biologie : la reconstruction d'une séquence génomique à partir d'un ensemble de ses fragments. Nous donnons une formalisation théorique de ce problème qui le rend résoluble en temps linéaire en utilisant les mots interdits minimaux. Nous prouvons aussi que notre algorithme résout un cas particulier du " problème de la plus petite sur-séquence " (Shortest Superstring Problem).

Table of Contents

Abstract iii

Résumé v

Table of Contents vii

List of Figures 1

1 Introduction 3

I Minimal Forbidden Words Theory 9

2 Words, Languages and Automata 11
2.1 Words and Factors 12
2.2 Languages and Automata 13
2.3 The Suffix Automaton 14
2.4 The Factor Automaton 17

3 Minimal Forbidden Words 19
3.1 Definitions and Basic Properties 20
3.1.1 Definition of Minimal Forbidden Words 20
3.1.2 An Algebraic Characterization 21
3.1.3 The Bad Minimal Forbidden Words 21
3.2 The L-automaton 22
3.3 Minimal Forbidden Words of a Finite Word 23
3.3.1 The Value $m(w)$ 23
3.3.2 Computing the Minimal Forbidden Words for a Finite Word . 24
3.3.3 Retrieving a Word from its Set of Minimal Forbidden Words . 25
3.4 Minimal Forbidden Words of Regular Factorial Languages 26
3.4.1 An Algebraic Characterization 26

3.4.2 Computing the Minimal Forbidden Words for a Regular Factorial Language . . . 27
3.5 Minimal Forbidden Words of a Finite Set of Finite Words . . . 28
3.5.1 The Multi-Suffix-Automaton . . . 29
3.5.2 Computing the Minimal Forbidden Words for P . . . 32

4 Minimal Periodic Forbidden Words 35
4.1 Definition of Minimal Periodic Forbidden Word . . . 36
4.2 Minimal Periodic Forbidden Words of a Finite Word . . . 38
4.2.1 The Periodic-Suffix-Automaton . . . 38
4.2.2 Computing the Minimal Periodic Forbidden Words for a Finite Word . . . 41
4.3 Minimal Periodic Forbidden Words of a Finite Set of Finite Words . . 43
4.3.1 The Periodic-Multi-Suffix-Automaton . . . 44
4.3.2 Computing Minimal Periodic Forbidden Words for P . . . 44

II Applications of Minimal Forbidden Words 47

5 Applications to Constrained Systems 49
5.1 Background and Basic Definitions . . . 51
5.2 Computation of the Shift Defined by Periodic Forbidden Words . . . 56
5.3 Presentation of Finite-Memory Systems with Unconstrained Positions 63

6 Applications to the Word Assembly Problem 71
6.1 The Word Assembly Problem . . . 74
6.2 Uniqueness of the Reconstruction . . . 76
6.3 Finding the Minimal Forbidden Words for w . . . 81
6.4 A New Algorithm for the Word Assembly Problem . . . 86
6.5 Relation With the Shortest Superstring Problem . . . 92

A An example of calculus of Word Assembly 97

Conclusions 103

Bibliography 110

List of Figures

1.1 The Rubin's vase. 4

2.1 The suffix automaton of the word $w = aabbabb$. 17
2.2 The factor automaton of the word $w = aabbabb$ 18

3.1 The trie of minimal forbidden words for the word $w = aabbabb$. 25
3.2 The multi-suffix automaton of the set $P = \{abbab, abaab\}$. 30
3.3 The $DAWG$ of the set $P = \{abbab, abaab\}$. 33
3.4 The trie of minimal forbidden words for the set $P = \{abbab, abaab\}$. . 34

4.1 Periodic suffix automata of $w = abbab$ 42
4.2 Minimal periodic forbidden words of $w = abbab$ 43

5.1 An automaton presenting a periodic-finite-type shift X. 53
5.2 Example of input tries. 61
5.3 Presentation of the shift $\mathsf{X}_{\{\mathcal{F},T\}}$. 61
5.4 The trie of the RLL (d, k)-constraint. 68
5.5 The Shannon cover of $S^{\sigma}_{U,T}$ for $S = \mathrm{MTR}(3)$, $T = 3$, $U = \{1\}$. . . . 69
5.6 A presentation of $S^{\sigma}_{U,T}$ for $S = \mathrm{MTR}(3)$, $T = 3, U = \{1\}$. 70

A.1 The suffix automaton of the word $w_1 = \$baa\$aba\$$. 99
A.2 The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$. . . 100
A.3 The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$ cut at length l_1. 100
A.4 Building the L-automaton of $\mathcal{T}(l_2 - 1)$. 101
A.5 The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$ cut at length l_2. 101

A.6 Building the L-automaton of $\mathcal{T}(l_2 - 1)$. 102
A.7 The L-automaton of $\mathcal{T}(l_2 - 1)$. 102

Chapter 1

Introduction

A standard method in mathematics consists in investigating the properties of an object by looking at its complement. The result is often surprising. A lot of properties of an object can be described by setting out what that object does not be. It is almost the same marvellous surprise that one can experience by watching the Rubin's vase (Figure 1). In this way, the study of the complement of an object can lead to a vast theory. This is the case of the minimal forbidden words.

In the context of symbolic dynamics, the shift spaces (or symbolic dynamical systems) are sometimes described as the sets of bi-infinite sequences that avoid a set of forbidden blocks [34]. This is an example of an object defined by "what it does not be". Furthermore, in order to get a better description of the shift space, it can be useful to find the set of minimal forbidden blocks which defines it. A block is a minimal forbidden block (or minimal forbidden word) of the shift space if it is forbidden (*i.e.*, it never appears as a factor of a sequence of the shift), but all its proper factors are not forbidden.

Shift spaces described by minimal forbidden blocks (or words) model some physical systems [2], and also some constrained channels in the area of coding theory [34]. For instance, the 0-1 sequences of a track in a compact disk are sequences that have at least two and at most ten symbols 0 between two symbols 1. Hence, these sequences are sequences of a shift described by the following set of minimal forbidden blocks $\{11, 101, 00000000000\}$. These sequences are an example of a finite-state constrained system with finite memory. Such a system can be described by a finite set of forbidden blocks.

Figure 1.1: The Rubin's vase.

One can also define minimal forbidden words in the context of formal languages. Given a factorial language L over an alphabet A (that is, a subset of A^* such that each factor of a word in L belongs to L), the complement $A^* \setminus L$ is an ideal of A^*. The basis of this ideal is the set of minimal forbidden words for the language L. Minimal forbidden words have been studied for finite and infinite words [42, 45] and for regular languages [4].

Crochemore *et al.* [28], presented a text compression algorithm based, unlike many text compression algorithms, on an anti-dictionary of the text. This anti-dictionary is the set of minimal forbidden words of a single word (the text itself). This method uses an effective construction of the set of words that avoid a given anti-factorial set of finite words [26]. An anti-factorial set of words is a set of words such that each proper factor of a word of the set does not belong to the set. A set of minimal forbidden words is always an anti-factorial set. It has been shown [26] that the computation of the language of words avoiding a finite anti-factorial set of words can be done in a linear time in the size of the finite set. This language is regular and the result of the computation is a finite-state automaton accepting the language. It can happen that this language is exactly the set of factors of a single word. In this case, the minimal forbidden words are the minimal forbidden words of a single word w, and the result of the computation is the minimal deterministic automaton accepting the factors of w, called the *factor automaton* of w.

A generalization of the notion of minimal forbidden words consists in defining an analogous in higher dimension, called minimal forbidden pattern. It can be used,

for example, for describing new properties of multi-dimensional shift spaces [5].

Minimal forbidden words have also been used to construct a topological invariant for shift spaces [6].

In this thesis, we continue the theoretical developments of minimal forbidden words and give a survey of known results. Then, in a second part, we exhibit two applications of minimal forbidden words.

The first one is an application to coding for recording systems. Recently, Wijngaarden and Immink proposed a coding scheme to encode an unconstrained sequence of bits into a constrained sequence in which certain bit positions are reserved for error correcting code (ECC) parity [59] (see also [11]). The bit values in these positions can be flipped (or not flipped) independently without violating the constraint. These positions are called *unconstrained positions*. Therefore, ECC parity information can be inserted into the unconstrained positions of the modulation-encoded sequences without making them out of the constrained channel. In [11], the authors study different approaches to build such codes. One of them is based on the construction of the set of all constrained sequences such that every position in a given subset U of integers modulo some integer T (the period), is unconstrained. These sequences define a subsystem of the initial constrained system.

We focus on the construction of this subsystem for a finite-state constrained system with finite memory. It appears to be a natural example of *Periodic-Finite-Type* system (PFT) introduced by Moision and Siegel [47]. This new kind of constrained channels can be described by a finite list of periodic forbidden blocks for each position i modulo an integer T. This is a generalization of a set of forbidden blocks to the periodic case. A bi-infinite sequence belongs to the channel if, up to some shift, it contains no forbidden factors at any position i modulo T.

Our contributions are the following. We present a polynomial-time construction of the set of sequences which satisfy a finite-memory constraint defined by a finite list of forbidden blocks and have a specified set of bit positions unconstrained. This construction is based on a linear-time algorithm for constructing a finite-state automaton accepting the sequences avoiding a list of periodic forbidden words for a given period. This algorithm can be compared to the construction in [11] of the same set of sequences from a finite-state automaton defining the finite-memory constraint. Our algorithm runs in time $O(T \times n \log n)$ in general, where n is the size of

the list, while the latter is exponential in the size of the automaton representing the constraint. Hence the use of the list of forbidden blocks to describe the constraint provides an improvement of the time complexity of the channel construction.

If we fix a period T, we can also wonder what kind of factors appear in a position modulo T of a given finite word. Minimal periodic forbidden words for a single finite word w and a given position i modulo the period T are in fact those words that do not appear as factors of w in a position $i \mod T$, but such that their longest proper prefix appears in some position $i \mod T$ and their longest proper suffix appears in position $i+1 \mod T$. We present a linear-time algorithm for constructing the minimal periodic forbidden words of a finite sequence for a given period.

All these results are contained in [3].

A second problem in which minimal forbidden words have found an application is a problem issued from biology, called the *Fragment Assembly Problem*. It is known that it is not possible to read the entire sequence of basis of a DNA molecule, but only factors of small length. The reconstruction of the original DNA sequence starting from these factors can be formalized in a theoretical problem, called the Fragment Assembly Problem, and is the object of several results in bio-informatics (the most frequent technique used is linked to the search of eulerian paths in a graph, see [33], [50] and [51] for more details). A theoretical simplification of the problem consists in considering a finite word as the target of the reconstruction and a set of its factors as the input of the problem. This problem has been treated by Carpi *et al.* (cf. [12–23]), who showed that a finite word can be uniquely reconstructed starting from some particular sets of its factors.

An approach based on minimal forbidden words has been introduced by Mignosi *et al.* [43,44,46]. Let $m(w)$ denote the length of the longest minimal forbidden word of w. Starting from a set I of factors of a finite sequence w such that I contains all the factors of w up to the length $m(w)$, it is possible to reconstruct w under the condition that the value $m(w)$ if known [43, 44, 46].

We improve this result by removing the a-priori knowledge of the value $m(w)$, while keeping the linear-time complexity. The word w has to satisfy a condition called I-compatibility. Furthermore, it is decidable in linear time whether there exists a word that is I-compatible. Finally, we derive an application to the shortest superstring problem.

These results have been published in [30]. A preliminary version can be found in [31].

This thesis is organized as follows.

Part I is devoted to the minimal forbidden words theory. In Chapter 2 we introduce basic stuff on words, languages and automata, and describe the suffix automaton and the factor automaton. In Chapter 3 we introduce minimal forbidden words and their properties. We also describe some algorithms about minimal forbidden words in the case of a single word, a regular language and a finite set of finite words. In Chapter 4 we introduce minimal periodic forbidden words, and we describe some algorithms about minimal periodic forbidden words in the case of a single word and of a finite set of finite words. In Part II we describe some applications of minimal forbidden words. In Chapter 5 we show some applications to constrained systems and periodic finite type shift spaces. In Chapter 6 we deal with the Word Assembly Problem, that is a theoretical simplification of the Fragment Assembly Problem issued from biology. In an appendix section we present some examples of computation with the Word Assembly Algorithm presented in Chapter 6. The thesis ends with a conclusion section devoted to work in progress and open problems.

Part I

Minimal Forbidden Words Theory

Chapter 2

Words, Languages and Automata

We recall here basic stuff about combinatorics on words and basic language theory and automata theory. We also describe some important data structures, as the suffix automaton and the factor automaton.

Introduction

Formal languages theory is one of the most important area of theoretical computer science. To get an idea of the variety of different covered topics one can see the three volumes of the Handbook of Formal Languages [54–56].

We want to focus here on words, their basic combinatorics, and the data structures to handle them.

The first part of this chapter is devoted to the theory of words and their relations with languages and automata. We only introduce basic stuff. For further details on these topics one can see Lothaire's books [35–38], among the others.

In the second part of the chapter we introduce two important data structures for handling finite words: the suffix automaton and the factor automaton. They are useful in string comparison and pattern matching theory. For a more complete description of their properties one can see [24, 25, 33, 52].

In Section 2.1 we recall the background about basic combinatorics on words and we fix the notation.

In Section 2.2 we deal with languages and we recall the basic automata theory used in combinatorics on words.

In Section 2.3 we introduce the suffix automaton of a finite word, and we describe its construction.

In Section 2.4 we briefly recall another data structure for finite words: the factor automaton.

2.1 Words and Factors

An *alphabet*, denoted by A, is a finite set of symbols (called *letters*). The size of A is constant and it is denoted by $|A|$. A *word* over A is a sequence of letters from A. The *length* (or size) of a finite word w is denoted by $|w|$ and is the number of its letters. The set of all finite words over A is denoted by A^*; the set of all words over A having a length exactly equal to n is denoted by A^n, while the set of all words over A having a length smaller or equal to n is denoted by $A^{\leq n}$. The empty word has length zero and is denoted by ε.

Let $w = a_0 a_1 \dots a_n$ be a nonempty finite word.

A *prefix* of w is any word v such that $v = \varepsilon$ or v is of the form $v = a_0 a_1 \dots a_i$, with $0 \leq i \leq n$. A *suffix* of w is any word v such that $v = \varepsilon$ or v is of the form $v = a_i a_{i+1} \dots a_n$, with $0 \leq i \leq n$. A *factor* (or *substring*) of w is a prefix of a suffix of w (or, equivalently, a suffix of a prefix of w). It is straightforward that the empty word ε is a prefix and a suffix (so even a factor) of every finite word w. We say that a factor v of a word w is a *strict factor* of w if $v \neq w$.

We denote by $\mathrm{Pref}(w)$, $\mathrm{Suff}(w)$ and $\mathrm{Fact}(w)$ respectively the set of all prefixes, suffixes and factors of the word w.

We denote by $w[i]$, for $i = 0, 1, \dots |w| - 1$, the letter at the position i in the word w.

A *period* for the word w is a positive integer p, with $0 < p \leq |w|$, such that $w[i] = w[i + p]$ for every $i = 0, 1, \dots |w| - p - 1$. Since $|w|$ is a period for w, we have that every nonempty word has at least one period. We can unambiguously define *the* period of the word w as the shortest of its periods. For example the period of $w = aabaaba$ is 3.

A *border* u of a finite word w is a strict factor of w (*i.e.* $u \neq w$) such that u is a prefix and a suffix of w. We can unambiguously define *the* border of the word w as the longest of its borders. For example the border of the word $w = aabaaabaab$

is *aab*.

There is a close relation between the notions of border and period of a word, as it is shown in the Proposition 1.4 of [25]:

Proposition 1 *Let w be a nonempty finite word and p an integer such that $0 \leq p \leq |w|$. Then the following statements are equivalent:*

1. *The integer p is a period of w.*
2. *There exists two unique words $m \in A^*$ and $r \in A^+$ and an integer $k > 0$ such that $w = (mr)^k m$ and $|mr| = p$.*
3. *There exists three words z, u, and v such that $w = zu = uv$ and $|z| = |v| = p$ (in particular u is a border of w).*

In particular Proposition 1 shows that the border of w has length $|w| - p$, where p is the period of w.

2.2 Languages and Automata

A language L is a subset of A^*, *i.e.* a collection of finite words over A. A language is finite if it contains a finite number of words. For a finite language L the *size* of L is the sum of the lengths of the words in L.

A language L is said *factorial* if it contains all factors of its words, *i.e.* it satisfies

$$\forall u, v \in A^* \quad uv \in L \Rightarrow u, v \in L.$$

A language M is said to be *anti-factorial* if for every $u, v \in M$ such that $u \neq v$, one has that u is not factor of v.

An automaton over the alphabet A is composed of a set Q of *states*, a set $E \subset Q \times A \times Q$ of *edges* or *transitions* and two sets $I, T \subseteq Q$ of *initial* and *terminal* (or *final*) states. The *transition function* of an automaton $\mathcal{A}$ is a partial function $\delta : Q \times A \to Q$, which associates to a state $p \in Q$ and a letter $a \in A$ the state $q \in Q$ if there exists a transition (p, a, q). A *path* in the automaton $\mathcal{A}$ is a sequence

$$(p_0, a_1, p_1), (p_1, a_2, p_2), \ldots, (p_{n-1}, a_n, p_n)$$

of consecutive transitions. Its *label* is the word $w = a_0a_1 \cdots a_n$. The path starts at p_0 and ends at p_n.

An automaton is *deterministic* if for each state p and for each letter a there exists at most one state q for which the transition (p, a, q) is defined.

An automaton is *finite* if its set of states is finite.

The set of initial states of an automaton can be reduced to a single *initial state* denoted by i. So we denote an automaton by $\mathcal{A} = (Q, A, i, T, \delta)$.

The language *recognized* by the automaton $\mathcal{A} = (Q, A, i, T, \delta)$ is the set of labels of the path starting at i and ending at a state $q \in T$.

A language $L \subseteq A^*$ is *recognizable* (or *regular*) if it can be recognized by a finite automaton.

A language $L \subseteq A^*$ is *rational* if it can be obtained from the finite subsets of A and a finite number of operations of union, product and star.

The well-known Theorem of Kleene asserts that, over a finite alphabet, a language is rational if and only if it is recognizable.

2.3 The Suffix Automaton

The *suffix automaton* of a finite word w over the alphabet A is the minimal deterministic automaton $\mathcal{A}(w)$ that recognizes the language $\mathrm{Suff}(w)$. We briefly recall its construction and its basic properties (more details can be found in [9, 25, 52]).

We denote by $u^{-1}\,\mathrm{Suff}(w)$ the right context (or future) of the word u in the language $\mathrm{Suff}(w)$, that is the language $u^{-1}\,\mathrm{Suff}(w) = \{u^{-1}y : y \in \mathrm{Suff}(w)\}$.

We define the equivalence relation $\equiv_{\mathrm{Suff}(w)}$ by

$$u \equiv_{\mathrm{Suff}(w)} v \Leftrightarrow u^{-1}\,\mathrm{Suff}(w) = v^{-1}\,\mathrm{Suff}(w).$$

Its equivalence classes are the states of $\mathcal{A}(w)$. We can also identify the states of $\mathcal{A}(w)$ with the sets of indices of the right positions of equivalent factors in w.

The function $s_w : \mathrm{Fact}(w) \setminus \{\varepsilon\} \to \mathrm{Fact}(w)$ defined by

$$s_w(v) = \text{the longest } u \in \mathrm{Suff}(v) \text{ such that } u \not\equiv_{\mathrm{Suff}(w)} v$$

is called the *suffix function* for the word w. One can prove that if $u, v \in \text{Fact}(w) \setminus \{\varepsilon\}$ are such that $u \equiv_{\text{Suff}(w)} v$, then $s_w(u) = s_w(v)$.

Let p be a state (different from the initial one) of $\mathcal{A}(w)$, and let u be a word in the class of p. The *suffix link* of p is the equivalence class of $s_w(u)$. The function $s : Q \setminus \{i\} \to Q$ which associates to a state of $\mathcal{A}(w)$ different from the initial one its suffix target is called *suffix function* of the suffix automaton $\mathcal{A}(w)$. The transitions belonging to some longest path from the initial state to some other state are called *solid* while the other ones are called *weak*.

If p is a state of the automaton, the sequence of states $p, s(p), s(s(p)), \ldots$ is finite and ends at the initial state i of $\mathcal{A}(w)$. This sequence is called the *suffix path* of p. If p is the class of w, the states of its suffix path are the terminal states of the automaton.

The algorithm Suffix-Automaton builds the suffix automaton of a finite word w over the alphabet A in linear time $O(|w| \times \log |A|)$. It is an incremental algorithm that computes successively a minimal automaton accepting $\text{Suff}(w[0\,..\,i])$, for i going from 0 to $|w| - 1$. This procedure calls procedures Extension and Split. Procedure Extension performs the transformations needed to get a minimal automaton accepting $\text{Suff}(w[0\,..\,i])$ from a minimal automaton accepting $\text{Suff}(w[0\,..\,i-1])$.

```
Suffix-Automaton (word w)
 1. create an initial state 0
 2. set s(0) = nil
 3. let p = 0
 4. for i from 0 to |w| − 1 do
 5.       p = Extension(p, w_i)
 6. let f = p
 7. while  f ≠ nil do
 8.       set f final
 9.       set f = s(f)
10. return automaton (Q, A, 0, {final}, δ)
```

```
EXTENSION (state p, letter a)
 1. create a new state q
 2. create a new solid transition (p, a, q)
 3. let r = s(p)
 4. while r ≠ NIL and there is no transition a going out of r do
 5.       create a weak transition (r, a, q)
 6.       set r = s(r)
 7. if r = NIL
 8.       set s(q) = 0
 9. else
10.       let s = δ(r, a)
11.       if the transition (r, a, s) is solid
12.             set s(q) = s
13.       else
14.             set s(q) = SPLIT(r, a, s)
15. return q
```

```
SPLIT (state p, letter a, state q)
 1. create a new state q'
 2. for each transition (q, a, r)
          create a weak transition (q', a, r)
 3. change the (weak) transition (p, a, q) into a solid
    transition (p, a, q')
 4. set s(q') = s(q)
 5. set s(q) = q'
 6. let t = s(p)
 7. while t ≠ NIL and the transition (t, a, q) is weak do
 8.       change (t, a, q) into (t, a, q')
 9.       set t = s(t)
10. return q'
```

Example 1 The suffix automaton of the word $w = aabbabb$ over the alphabet $A = \{a, b\}$ is shown in Figure 2.1.

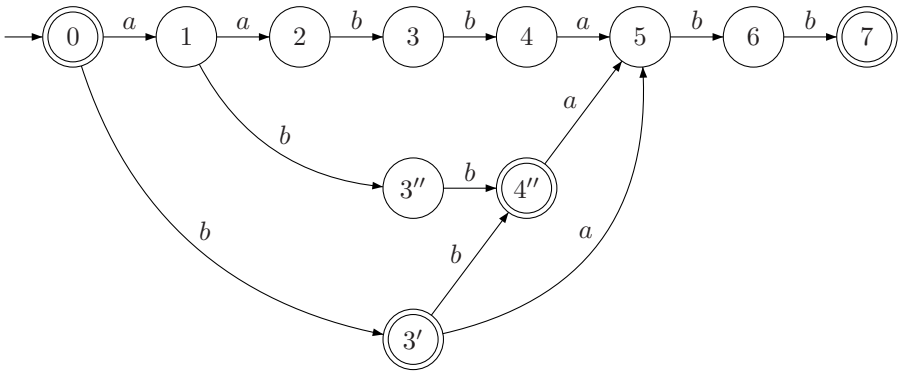

Figure 2.1: The suffix automaton of the word $w = aabbabb$. There have been three splits during the construction.

2.4 The Factor Automaton

The factor automaton of a finite word $w \in A^*$ is the minimal deterministic automaton recognizing the language of the factors of w.

Making all states of the suffix automaton of a word w terminal ones we obtain an automaton that recognizes the factors of w, but it may not be the minimal one, since the states of the factor automaton are in fact the equivalence classes of the right syntactic congruence associated with $\mathrm{Fact}(w)$, while in the suffix automaton it was associated with $\mathrm{Suff}(w)$.

Nevertheless, the factor automaton of the word w can be obtained from the suffix automaton of w, by a standard minimization procedure.

Example 2 The factor automaton of the word $w = aabbabb$ is shown in Figure 2.2

For a word w over A and a symbol \$ not belonging to A, we have that the factor automaton of the word $w\$$ is in fact the suffix automaton of $w\$$. This suggests a method for building the suffix automaton of w from its factor automaton (see [24]). Actually, one can first build the factor automaton of $w\$$ on the alphabet $A \cup \{\$\}$, then set as terminal those states from which an edge labelled by \$ outgoes, and finally remove all edges labelled by \$ and the states they reach.

Deeper discussions about the factor automaton, its construction, and its links

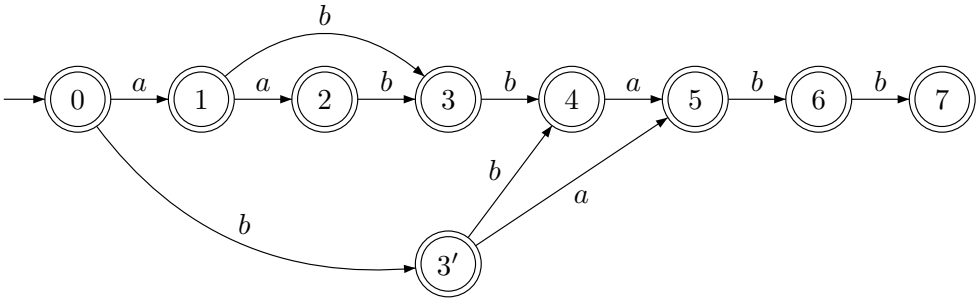

Figure 2.2: The factor automaton of the word $w = aabbabb$. States $3''$ and 3 and states $4''$ and 4 of the suffix automaton have been merged, and all states are terminal.

with the suffix automaton can be found in [24] and [10]. We only recall, here, that the factor automaton of the word w can be constructed in linear time on the size of w, and that one can obtain the factor automaton of a word w from its suffix automaton and *vice versa* always in linear time on the size of w.

Chapter 3

Minimal Forbidden Words

This chapter is devoted to the presentation of the minimal forbidden words theory. We introduce the minimal forbidden words and their basic theory. Then we show recent theoretical results about minimal forbidden words.

Introduction

Minimal forbidden words have been introduced as a tool for the investigation of combinatorial properties of words. The basic idea is the following: in order to understand what kind of factors do appear in a word, determinate the "most little object" that determines what does not appear in that word.

Words and their factors are the basic stuff of the formal language theory, including the related area of string processing, symbolic dynamics, genomic sequences treatment, and so on. Hence, minimal forbidden words can appear in different context of theoretical computer science.

A necessary condition for obtaining a consistent theory is to build a solid algebraic frame on the background. Hopefully, we have got a solid algebraic theory from which to start: the formal language theory. This will be our start lane.

Another suitable aspect of a combinatorial theory is the fastness of the procedures and of the algorithms created for their practical implementation. This problem is related to (and often a consequence of) the right choice of the data structures. We will show in this chapter how powerful are in fact the data structures we introduced in the previous chapter.

For all these reasons the introduction of the notion of minimal forbidden words (or minimal forbidden factors) is followed by the algebraic characterization of the set of the minimal forbidden words for a factorial language (that is the natural context in which words appear).

Then, some basic algorithms are described. First, we must be able to pass from the "positive" point of view (the words and their factors) to the "negative" one (the minimal forbidden words), and *vice versa*.

Moreover, these two operations can be realized in a "fast" way (*i.e.* in linear time on the input size of the algorithms).

Finally, we describe the methods and the procedures for the computation of the minimal forbidden words in some specifical case, as a single word, a regular language, and a finite set of finite words.

In Section 3.1 we give the definition of minimal forbidden word and we set the algebraic frame.

In Section 3.2 we introduce the L-automaton, which builds the language of the words that avoid a given set of words.

In Section 3.3 we investigate the properties of the minimal forbidden words for a single finite word and we give the automata for constructing the set of the minimal forbidden words for a word and for retrieving a word from its set of minimal forbidden words.

In Section 3.4 we generalize the minimal forbidden words to the case of a factorial regular language, and we give the automaton for constructing the set of the minimal forbidden words for a factorial regular language.

Finally, in Section 3.5, we extend previous algorithms to the case of a finite set of finite words.

3.1 Definitions and Basic Properties

3.1.1 Definition of Minimal Forbidden Words

Let w be a word over an alphabet A. A finite nonempty word $v = a_0a_1 \dots a_n$ is a *minimal forbidden word for* w (or a *minimal forbidden factor of* w) if

- the word v is not a factor of w,

- the strict prefix of maximal length of v, $a_0a_1 \dots a_{n-1}$, and the strict suffix of maximal length of v, $a_1a_2 \dots a_n$, are factors of w.

Equivalently, one can say that a finite word v is a minimal forbidden word for the word w if and only if v is not a factor of w but every strict factor of v is factor of w.

We denote by $\mathcal{MF}(w)$ the set of all minimal forbidden words for w. By the minimality of its words we have that $\mathcal{MF}(w)$ is an anti-factorial language.

Example. Let $w = aabbbaa$ over the alphabet $A = \{a, b\}$. The minimal forbidden words for w are those of the set $\mathcal{MF}(w) = \{aaa, bbbb, aba, abba, bab, baab\}$.

Example. Let $w = 0000$ over the alphabet $A = \{0, 1\}$. The minimal forbidden words for w are those of the set $\mathcal{MF}(w) = \{00000, 1\}$.

3.1.2 An Algebraic Characterization

From an algebraic point of view the set $\mathcal{MF}(w)$ of the minimal forbidden words for a finite word w over the alphabet A is uniquely characterized by the equation:

$$\mathcal{MF}(w) = A\,\mathrm{Fact}(w) \cap \mathrm{Fact}(w)A \cap (A^* \setminus \mathrm{Fact}(w)).$$

Conversely, every finite word in A^* that do not contain any factor of $\mathcal{MF}(w)$ is a factor of w, *i.e.*

$$\mathrm{Fact}(w) = A^* \setminus A^*\mathcal{MF}(w)A^*,$$

so $\mathcal{MF}(w)$ uniquely characterizes $\mathrm{Fact}(w)$ and $\mathrm{Fact}(w)$ uniquely characterizes $\mathcal{MF}(w)$.

3.1.3 The Bad Minimal Forbidden Words

We introduce here a new definition that will be useful in Chapter 6. A minimal forbidden word v for a finite word w is called a *bad minimal forbidden word* for w if

1. the strict prefix of maximal length of v appears just once as factor of w, and it is a suffix of w,

2. the strict suffix of maximal length of v appears just once as factor of w, and it is a prefix of w.

Example. Let $w = aabbbaa$. Then $v = baab$ is a bad minimal forbidden word for w.

3.2 The L-automaton

We now introduce an algorithm which will be useful in almost all the further procedures. This is the reason why it is described in a stand-alone section.

We denote by $L(M)$ the (factorial) language avoiding a given finite anti-factorial language M, *i.e.* the set of all the words that do not contain any word of M as factor.

For any finite anti-factorial language M, the *L-automaton* of M is the minimal deterministic automaton that recognizes the language $L(M)$.

```
L-AUTOMATON (trie T = (Q, A, i, T, δ'))
 1. for each a ∈ A
 2.       if δ'(i, a) defined
 3.             set δ(i, a) = δ'(i, a);
 4.             set f(δ(i, a)) = i;
 5.       else
 6.             set fδ(i, a) = i;
 7. for each state p ∈ Q \ {i} in width-first search and each a ∈ A
 8.       if δ'(p, a) defined
 9.             set δ(p, a) = δ'(p, a);
10.             set f(δ(p, a)) = δ(f(p), a);
11.       else if p ∉ T
12.             set δ(p, a) = δ(f(p), a);
13.       else
14.             set δ(p, a) = p;
15. return (Q, A, i, Q \ T, δ);
```

We describe below the algorithm L-AUTOMATON that builds the L-automaton of an anti-factorial language M. It has been introduced by Crochemore *et al.* in [26]. It runs in linear time on the size of M. If M is the set of the minimal forbidden words for a finite word w, then the L-automaton of the language M is the minimal

deterministic automaton accepting the set Fact(w) of the factors of w. The input of L-AUTOMATON is the trie[1] recognizing the anti-factorial language M.

The following two results are proved in [26].

Proposition 2 *Let $\mathcal{T}$ be the trie of an anti-factorial language M. Algorithm* L-AUTOMATON *builds a complete deterministic automaton accepting $L(M)$.*

Proposition 3 *Algorithm* L-AUTOMATON *runs in time $O(|Q| \times |A|)$ on input $\mathcal{T}$ if transition functions are implemented by transition matrices.*

3.3 Minimal Forbidden Words of a Finite Word

3.3.1 The Value $m(w)$

For a finite word w, we denote by $m(w)$ the length of the longest minimal forbidden word for w. Mignosi *et al.* have proved (see [44]) that for a word w randomly generated by a memoryless source, the parameter $m(w)$ approximates $O(\log_d(n))$, where n is the length of the word w and d is the cardinality of the alphabet A.

The largest value that $m(w)$ can take is $|w|+1$, since the prefixes and the suffixes of a minimal forbidden word for a word w are factors of w. The words w having a minimal forbidden word of length $|w| + 1$ are all and the only ones of the form $w = a^n$ for a symbol $a \in A$ and a positive integer n. Indeed, if a minimal forbidden word u for w has length $|w| + 1$, it must be $u = aw = wb$ for some $a, b \in A$. But in this case it is well known by the elementary theory of combinatorics on words that the only possibility is $a = b$ and $w = a^{|w|}$.

On the other hand, it is shown in [45, Theorem 13]) that

$$m(w) \geq \lceil \log_{|A|}(|w| + 1) \rceil.$$

Moreover, if the word w has period p, the following inequality holds (see [45, Theorem 14]):

$$m(w) \geq |w| - p + 2.$$

[1]Recall that a *trie* is a tree for storing a set of words in which there is one node for every common prefix and in which the words are stored in the leaves.

For a finite word w, the *repetition index* $r(w)$ is the length of the longest factor of w that has at least two occurrences in w. For example the word $w = aabbbaa$ has $r(w) = 2$. There is a close relation between repetition index and minimal forbidden words. Actually, we will prove in Chapter 6 that

$$r(w) = m(w) - 2.$$

3.3.2 Computing the Minimal Forbidden Words for a Finite Word

Now, we describe the procedures for computing the minimal forbidden words of a finite word, and for retrieving a finite word from the set of its minimal forbidden words.

Given a finite word w, one can construct the set $\mathcal{MF}(w)$ of the minimal forbidden words for w in linear time on the size of w. Actually, algorithm MF-TRIE, described in [26], builds the trie of the set $\mathcal{MF}(w)$, having as input the factor automaton of w, that is the minimal deterministic automaton accepting the factors of w. Algorithm MF-TRIE runs in linear time $O(|w| \times |A|)$. Moreover, the states of the trie of the set $\mathcal{MF}(w)$ are the same as those of the factor automaton of w, plus some *sink* states, that are the terminal states of the minimal forbidden words.

MF-TRIE (factor automaton $\mathcal{A} = (Q, A, i, T, \delta)$ and its suffix function s)
1. **for** each state $p \in Q$ in width-first search from i **and** each $a \in A$
2. **if** $\delta(p, a)$ undefined **and** ($p = i$ **or** $\delta(s(p), a)$ defined)
3. $\delta'(p, a) \leftarrow$ new sink;
4. **else**
5. **if** $\delta(p, a) = q$ **and** q not already reached
6. $\delta'(p, a) \leftarrow q$;
7. **return** $(Q, A, i, \{sinks\}, \delta')$;

The following two propositions are proved in [26].

Proposition 4 *Let $\mathcal{A}(w)$ be the factor automaton of a word $w \in A^*$. Algorithm* MF-TRIE *builds the trie accepting the set of minimal forbidden words for* Fact(w)*, that is $\mathcal{MF}(w)$.*

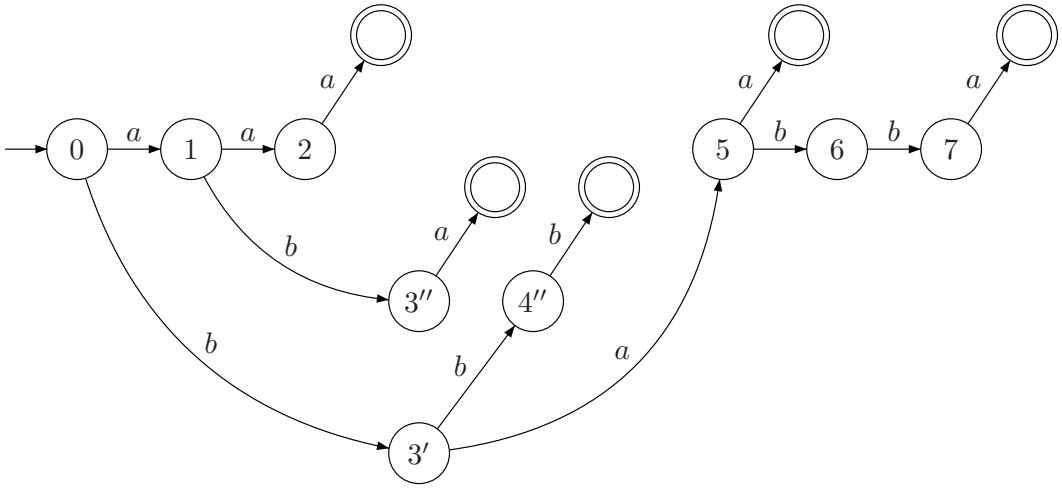

Figure 3.1: The trie of minimal forbidden words for the word $w = aabbabb$.

Proposition 5 *Algorithm* MF-TRIE *runs in time* $O(|w| \times |A|)$ *on input word* w *if transition functions are implementd by transition matrices.*

Example. Let $w = aabbabb$. We have $\mathcal{MF}(w) = \{aaa, aba, baa, bbb, babba\}$. The trie of minimal forbidden words for w is shown in Figure 3.1. Note that the states of the trie are the same as those of the suffix automaton of w (we do not represent here states 3 and 4 because they are not accessible nor co-accessible) plus the sink states representing the minimal forbidden words for w, which are in number of $|\mathcal{MF}(w)| = 5$.

3.3.3 Retrieving a Word from its Set of Minimal Forbidden Words

Conversely, given a finite set $\mathcal{MF}(w)$ representing the set of the minimal forbidden words for a finite word w, we can reconstruct the word w in linear time on the size of the trie representing the set $\mathcal{MF}(w)$. The algorithm performing this operation is W-RECONSTRUCTION and it is described in [44] and [45] (see also 6.1).

We only outline here its description. The algorithm first constructs the L-automaton of the trie of the minimal forbidden words for w, that is the trie recognizing the set $\mathcal{MF}(w)$. After deleting the sink states of the obtained automaton, it

finds the longest path starting from the initial state, by using a classical topological sort procedure. This path corresponds to the word w.

This construction is a consequence of the following result (Theorem 7 of [26]):

Theorem 6 *If the input of the algorithm* L-AUTOMATON *is the set of the minimal forbidden words for a single finite word w, then its output is the minimal deterministic finite automaton accepting the factors of w, i.e. the factor automaton of w.*

3.4 Minimal Forbidden Words of Regular Factorial Languages

In [4] authors give a quadratic-time algorithm to compute the set of minimal forbidden words of a factorial regular language. We give here an outline of this result.

3.4.1 An Algebraic Characterization

Let $L \subseteq A^*$ be a factorial language. The complement language $L^c = A^* \setminus L$ is an ideal of A^*. By denoting by $\mathcal{MF}(L)$ the base of this ideal we have that $L^c = A^*\mathcal{MF}(L)A^*$. The set $\mathcal{MF}(L)$ is called the set of *minimal forbidden words for L*. A word $v \in A^*$ is forbidden for L if $v \notin L$, which is equivalent to say that v occurs in no word of L. In addition, v is minimal if it has non proper factor that is forbidden. Remark that

$$L = A^* \setminus A^*\mathcal{MF}(L)A^*. \tag{3.1}$$

It is also straightforward that a word $v = a_1a_2 \dots a_n$ belongs to $\mathcal{MF}(L)$ if and only if the two following conditions hold:

- v is forbidden.
- both $a_1a_2 \dots a_{n-1} \in L$ and $a_2a_3 \dots a_n \in L$.

The set $\mathcal{MF}(L)$ is an anti-factorial language, and by the definition one can see that

$$\mathcal{MF}(L) = AL \cap LA \cap (A^* \setminus L). \tag{3.2}$$

As a consequence of both equalities 3.1 and 3.2 we obtain the following:

Proposition 7 *Given a factorial language L, L is regular if and only if $\mathcal{MF}(L)$ is regular.*

3.4.2 Computing the Minimal Forbidden Words for a Regular Factorial Language

The following algorithm MF-AUTOMATON presented in [4] computes in polynomial time a deterministic finite state automaton recognizing the set $\mathcal{MF}(L)$ from a deterministic finite state automaton recognizing the set L. This generalizes the algorithm MF-TRIE of [26], in which L was the set of the factors of a finite word, that we presented in the previous section.

We recall that an automaton is *trim* if each state is accessible from the initial state and coaccessible from a final state.

Assume that a trim automaton $\mathcal{A}$ recognizing the factorial language L is given. The automaton $\mathcal{A}$ is in the form (Q, A, i, Q, δ) because all states are necessarily terminal states. The algorithm MF-AUTOMATON below computes from $\mathcal{A}$ a deterministic automaton $\mathcal{A}' = (Q', A, i', T', \delta')$ recognizing $\mathcal{MF}(L)$ where

- the set Q' of states is the set $Q \times Q \cup \{(-, i)\} \cup \{p, \$) | p \in Q\}$, ("$-$","\$" are symbols that do not belong to A),
- A is the current alphabet,
- the initial state i' is the state $(-, i)$,
- the set T' of terminal states is the set of states $(p, \$)$ for $p \in Q$.

The pairs (p, q) considered in the algorithm are such that $p = \delta(i, w)$, $q = \delta(i, aw)$ for some letter $a \in A$ and some word $w \in A^*$. Note that $\delta(p, a)$ is well defined at line 10 when $\delta(q, a)$ is defined since it is assumed that the language L is factorial and $\mathcal{A}$ is trim.

MF-AUTOMATON (automaton $\mathcal{A} = (Q, A, i, Q, \delta)$)
1. $i' \leftarrow (-, i)$, $T' \leftarrow \{(p, \$) \mid p \in Q\}$;
2. $Q' \leftarrow Q \times Q \cup \{i'\} \cup \{T'\}$;
3. **for** each $a \in A$
4. **if** $\delta(i, a)$ defined
5. $\delta'((-, i), a) \leftarrow (i, \delta(i, a))$;
6. **else**
7. $\delta'((-, i), a) \leftarrow (i, \$)$;
8. **for** each pair of states (p, q) accessible by δ' from i' with $p, q \in Q$, $p \neq q$ **and** each $a \in A$
9. **if** $\delta(q, a)$ defined
10. $\delta'((p, q), a) \leftarrow (\delta(p, a), \delta(q, a))$;
11. **else**
12. **if** $\delta(p, a)$ defined
13. $\delta'((p, q), a) \leftarrow (\delta(p, a), \$)$;
14. **return** automaton $\mathcal{A}' = (Q', A, i', T', \delta')$;

The following two propositions are proved in [4].

Proposition 8 *Let $\mathcal{A}$ be a deterministic trim automaton which recognizes a factorial language L. Algorithm* MF-AUTOMATON *computes from $\mathcal{A}$ a deterministic automaton that recognizes $\mathcal{MF}(L)$.*

Proposition 9 *If the transition function is implemented by a transition matrix, then the algorithm* MF-AUTOMATON *runs in time $O(|Q|^2 \times |A|)$ on input automaton $\mathcal{A} = (Q, A, i, Q, \delta)$.*

For a discussion about the tightness of the quadratic bound in the last Proposition 9 see [4].

3.5 Minimal Forbidden Words of a Finite Set of Finite Words

In [4] authors also give a linear-time algorithm computing the set of minimal forbidden words for a finite set of finite words. This is another extension of algorithm MF-TRIE of [26].

3.5.1 The Multi-Suffix-Automaton

Let $P = \{w^1, w^2, \ldots, w^r\}$ be a finite set of words of cardinal r. We denote by $|P|$ the sum of the lengths of the words of P and by $\mathrm{Suff}(P)$ and $\mathrm{Fact}(P)$ the set of suffixes and factors of P respectively, *i.e.* the union, for $1 \leq j \leq r$, of the sets $\mathrm{Suff}(w^j)$ and of the sets $\mathrm{Fact}(w^j)$ respectively.

The construction of a suffix automaton of a single finite word can be extended to a finite set of finite words P (see [8, 9], and also [52]). This construction can be performed in linear time on the size $|P|$ of the set P. The automaton obtained is not necessarily minimal, but it can be minimized in linear time by using classical minimization algorithms, as that of [53]. A direct construction of the minimal deterministic finite automaton recognizing $\mathrm{Suff}(P)$ is not known so far.

As in the single-word case, one can obtain an automaton recognizing the factors of the words in the set P, that is the set $\mathrm{Fact}(P)$, by setting all the states terminal in the suffix automaton of P.

Let us denote by $\mathcal{A}(P) = (Q, A, i, T, \delta)$ the suffix automaton of P. The states of $\mathcal{A}(P)$ are the equivalence classes of the right invariant equivalence $\equiv_{\mathrm{Suff}(P)}$ defined as follows. If $u, v \in \mathrm{Fact}(P)$,

$$u \equiv_{\mathrm{Suff}(P)} v \;\text{ iff }\; \forall i,\ 1 \leq i \leq r,\ u^{-1}\,\mathrm{Suff}(p^i) = v^{-1}\,\mathrm{Suff}(p^i),$$

and there is a transition labelled by a from the class of a word u to the class of ua. The automaton $\mathcal{A}(P)$ has a unique initial state, which is the class of the empty word. Note that the syntactic congruence $\sim$ defining the minimal automaton of the language is

$$u \sim v \;\text{ iff }\; \bigcup_{i=1}^{r} u^{-1}\,\mathrm{Suff}(p^i) = \bigcup_{i=1}^{r} v^{-1}\,\mathrm{Suff}(p^i),$$

which is not the same as the above equivalence. This explains why $\mathcal{A}(P)$ is not always the minimal automaton recognizing the suffixes of the words in P.

The algorithm building the suffix automaton of the set P is called Multi-Suffix-Automaton and is described below. The words of the set P are added in a sequential way. Adding the word w^j consists in adding its letters w_i^j one after the other. The procedure realizing this operation is called Multi-Extension and is

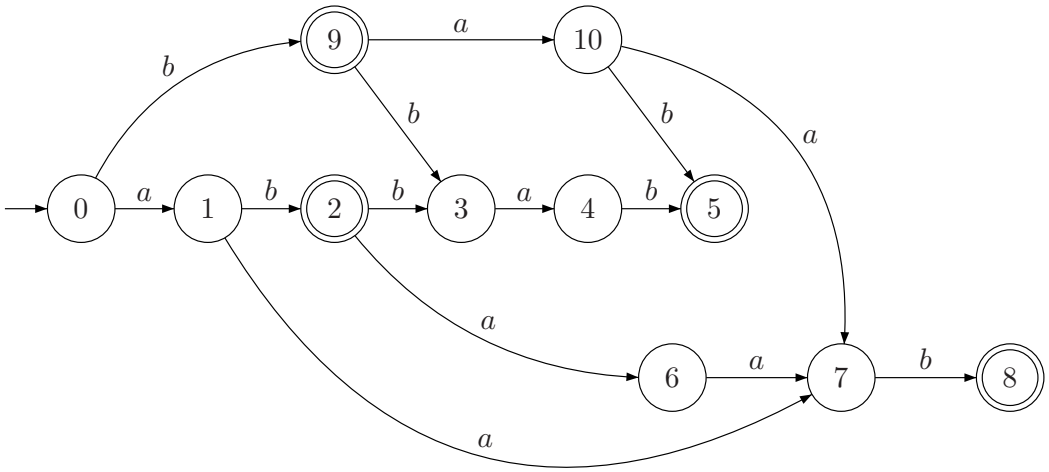

Figure 3.2: The multi-suffix automaton of the set $P = \{abbab, abaab\}$.

slightly different from the EXTENSION one because it must consider the edges and the suffix links builded for the words previously added. Nevertheless it calls the same procedure SPLIT as in the construction of the suffix automaton for a single word.

Example. Let $P = \{abbab, abaab\}$. The suffix automaton $\mathcal{A}(P)$ of the set P is shown in Figure 3.2.

Final states are determined only at the and of the whole procedure, when all words of the set P have been added, by following the suffix links relative to the set P.

The following Proposition is straightforward.

Proposition 10 *Algorithm* MULTI-SUFFIX-AUTOMATON *runs in time* $O(|P|)$*, where* $|P|$ *denotes the sum of the lengths of the words in* P*.*

```
MULTI-SUFFIX-AUTOMATON (set P = {w^1, w^2, ..., w^r})
 1. create an initial state 0
 2. set s(0) = NIL
 3. for j from 1 to r do
 4.       set p = 0;
 5.       for i from 0 to |w^j| − 1 do
 6.            p = MULTI-EXTENSION(p, w_i^j)
 7.       let f = p
 8.       while  f ≠ NIL do
 9.       set f final
10.       set f = s(f)
11. return automaton A(P) = (Q, A, 0, {final}, δ)
```

```
MULTI-EXTENSION (state p, letter a)
 1. if δ(p, a) is defined
 2.       set q = δ(p, a)
 3.       if the transition (p, a, q) is weak
 4.            q ← SPLIT(p,a,q)
 5. else
 6.       create a new state q
 7.       create a new solid transition (p, a, q)
 8.       let r = s(p)
 9.       while r ≠ NIL and there is no transition a going out of r do
10.            create a weak transition (r, a, q)
11.            set r = s(r)
12.       if r = NIL
13.            set s(q) = 0
14.       else
15.            let s = δ(r, a)
16.            if the transition (r, a, s) is solid
17.                 set s(q) = s
18.            else
19.                 set s(q) = SPLIT(r, a, s)
20. return q
```

3.5.2 Computing the Minimal Forbidden Words for P

Now, we can construct a procedure for computing the minimal forbidden words for the set P.

We denote by $DAWG(P)$ (Direct Acyclic Word Graph) the automaton recognizing the factors of the words in P obtained from $\mathcal{A}(P)$ by setting all states terminal. We recall that $DAWG(P)$ may not be the minimal deterministic automaton recognizing Fact(P) (see 2.4), but the important fact, in this context, is that $DAWG(P)$ comes out with the suffix function s for the set P, defined during the construction of $\mathcal{A}(P)$.

Since the graph of $\mathcal{A}(P)$ is an acyclic graph, so is the graph of $DAWG(P)$. Thus algorithm MF-TRIE of [26] can run on the input $DAWG(P)$, obtaining the trie recognizing the set of the minimal forbidden words for the language Fact(P), that is $\mathcal{MF}(\text{Fact}(P))$.

MF-TRIE (automaton $DAWG(P) = (Q, A, i, Q, \delta)$ and its suffix function s)

1. **for** each state $p \in Q$ in width-first search from i **and** each $a \in A$
2. **if** $\delta(p, a)$ undefined **and** ($p = i$ **or** $\delta(s(p), a)$ defined)
3. $\delta'(p, a) \leftarrow$ new sink;
4. **else**
5. **if** $\delta(p, a) = q$ **and** q not already reached
6. $\delta'(p, a) \leftarrow q$;
7. **return** $(Q, A, i, \{sinks\}, \delta')$;

The following two propositions are proved in [4].

Proposition 11 *Algorithm* MF-TRIE *builds the trie recognizing the set of minimal forbidden words for* Fact(P)*, that is denoted by* $\mathcal{MF}(P)$*.*

Proposition 12 *Algorithms* MULTI-SUFFIX-AUTOMATON *and* MF-TRIE *together run in time* $O(|P| \times |A|)$ *on the input* P*, if the transition functions are implemented by transition matrices.*

Example. Let $P = \{abbab, abaab\}$. The automaton $DAWG(P)$ is shown in Figure 3.3.

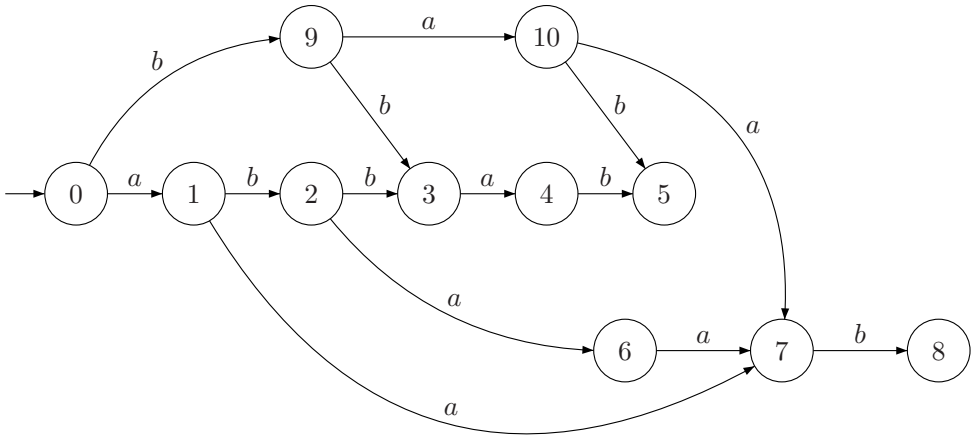

Figure 3.3: The $DAWG$ of the set $P = \{abbab, abaab\}$.

Example. The set of the minimal forbidden words for P over the alphabet $A = \{a, b, c\}$ is $\mathcal{MF}(P) = \{c, aaa, bbb, aaba, aabb, abab, baba, babb, bbaa\}$. The trie representing $\mathcal{MF}(P)$ is shown in Figure 3.4.

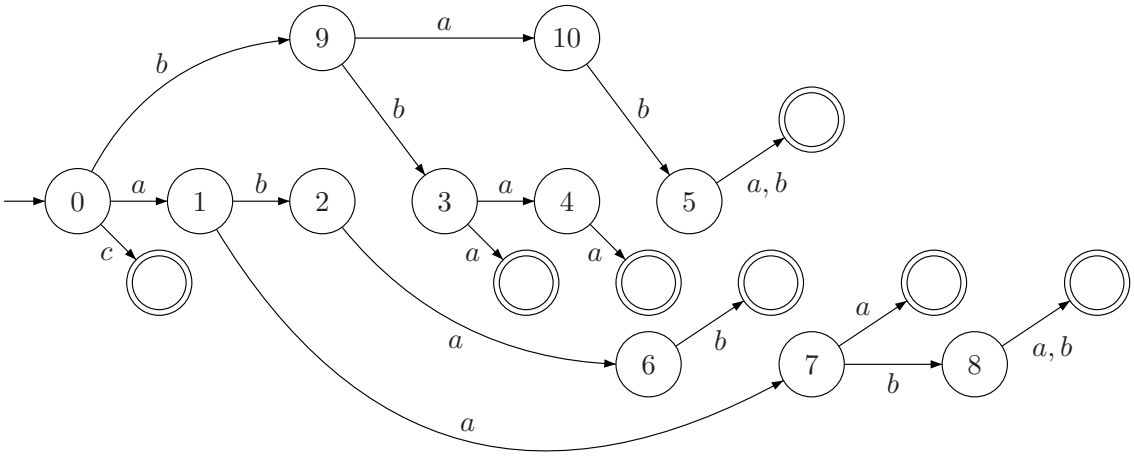

Figure 3.4: The trie of minimal forbidden words for the set $P = \{abbab, abaab\}$.

Chapter 4

Minimal Periodic Forbidden Words

In this chapter we define the notion of minimal periodic forbidden words and we study the problem of computing the minimal periodic forbidden words of a finite word and of a finite set of finite words.

Introduction

Repetitions, and especially consecutive repetitions, play an important role in the analysis of molecular biology sequences. Some of them are even related to known diseases. From this point of view it is interesting to consider periodic forbidden words according to a single word. This may be used to discover combinatorial properties of the sequence and identify subsequence motifs either in coding regions and in "junk DNA", and then to derive statistical features on them.

We introduce the definition of the minimal periodic forbidden words of a given finite word. We give a linear-time algorithm to compute the set of the minimal periodic forbidden words of a finite word. This represents an extension to the periodic case of algorithm MF-TRIE of [26] presented in 3.3.

As in the non-periodic case, we can generalize this construction to the case of a finite set of finite words. We are able to compute the set of the minimal periodic forbidden words for a finite set of finite words P with a fixed period in linear time on the size of P.

These results have been obtained very recently, and we hope that they will be the basis for further developments. One can try, for instance, to generalize other results obtained in the non-periodic case.

In Section 4.1 we introduce the minimal periodic forbidden words and we describe their basic algebraic properties.

In Section 4.2 we give a method to compute in linear time the set of minimal periodic forbidden words for a finite word with a fixed period.

Finally, in Section 4.3, we generalize previous algorithms to the case of a finite set of finite words with a fixed period. We describe an algorithm to compute in linear time the set of minimal periodic forbidden words in this case.

4.1 Definition of Minimal Periodic Forbidden Word

Let w be a finite word over a finite alphabet A of fixed constant size. Let T be a positive integer, called the period. The set of minimal forbidden words in a phase k of the word w, with $0 \leq k \leq T-1$, is the set of finite blocks v that never appear at a position $k \bmod T$ of w, and such that there is no strict factor v' of v with $v' \prec_i v$ appearing at a position $k+i \bmod T$ of w.

In the sequel, we fix a positive integer T as period. If w is a finite word we denote by $\mathrm{Suff}^{(k)}(w)$, for $0 \leq k \leq T-1$, the set of suffixes of w beginning at a position of w equal to k modulo T. Thus,

$$\mathrm{Suff}^{(k)}(w) = \{w[i \,..\, |w|-1] \mid i = k \bmod T\}.$$

We denote by $\mathrm{Fact}^{(k)}(w)$ the set of prefixes of $\mathrm{Suff}^{(k)}(w)$, that is, the set of factors of w that occur in w at positions k modulo T. In this section, we also denote by $\mathcal{F}^{(k)}(w)$ the set of finite blocks that are not factors of w at a position k modulo T. Thus $\mathcal{F}^{(k)}(w) = A^* - \mathrm{Fact}^{(k)}(w)$.

The collection of minimal periodic forbidden words of w for a period T is defined as the finite collection of sets $\mathcal{MF}^{(k)}(w)$, with $0 \leq k \leq T-1$, where

$$\begin{aligned}\mathcal{MF}^{(k)}(w) &= \mathcal{F}^{(k)}(w) - \mathcal{F}^{(k)}(w)A^+ - (A^T)^+\mathcal{F}^{(k)}(w)A^* \\ &\quad - \bigcup_{i=1}^{T-1}(A^T)^*A^i\mathcal{F}^{(k+i \bmod T)}(w)A^*.\end{aligned}$$

Thus, the above collection $\mathcal{MF}^{(k)}(w)$ is periodic and anti-factorial. It is minimal in the following sense: if $u \in \mathcal{F}^{(k)}(w)$, then u has a factor at some position i that belongs to $\mathcal{MF}^{(k+i \bmod T)}(w)$, and any other collection of finite sets of blocks $\mathcal{G}^{(k)}$ satisfying this condition verifies $\mathcal{MF}^{(k)}(w) \subseteq \mathcal{G}^{(k)}$. Although this notion of minimality refers to a finite word w, it is similar to the notion of periodic first offenders defined in [47] for constrained systems.

We now give a simpler expression of the set $\mathcal{F}^{(k)}(w)$ used to derive the next algorithm.

Proposition 13 *The set $\mathcal{MF}^{(k)}(w)$ of minimal periodic forbidden words of w for a period T satisfies*

$$\begin{aligned}\mathcal{MF}^{(k)}(w) = (A\,\mathrm{Fact}^{(k+1 \bmod T)}(w)) \\ \cap\,(\mathrm{Fact}^{(k)}(w)A) \cap (A^* - \mathrm{Fact}^{(k)}(w)).\end{aligned} \tag{4.1}$$

Proof Let u be a block of $(A\,\mathrm{Fact}^{(k+1 \bmod T)}(w)) \cap (\mathrm{Fact}^{(k)}(w)A) \cap (A^* - \mathrm{Fact}^{(k)}(w))$. Then $u \in \mathcal{F}^{(k)}(w)$. Since $u \in \mathrm{Fact}^{(k)}(w)A$, then $u \notin \mathcal{F}^{(k)}(w)A^+$. Since $u \in A\,\mathrm{Fact}^{(k+1 \bmod T)}(w)$, then $u \in A^i\,\mathrm{Fact}^{(k+i \bmod T)}(w)$ for $1 \leq i \leq |u|$. Hence $u \notin (A^T)^+\mathcal{F}^{(k)}(w)A^*$, and u does not belong to $\bigcup_{i=1}^{T-1}(A^T)^*A^i\mathcal{F}^{(k+i \bmod T)}(w)A^*$ either.

Conversely, let u be a block of $\mathcal{MF}^{(k)}(w)$. Then $u \in \mathcal{F}^{(k)}(w)$. If $u \notin \mathrm{Fact}^{(k)}(w)A$, then $u = va$, with $a \in A$, and $v \in \mathcal{F}^{(k)}(w)$. Hence $u \in \mathrm{Fact}^{(k)}(w)A$. Let us now assume that $u \notin A\,\mathrm{Fact}^{(k+1 \bmod T)}(w)$. Then $u = av$, with $a \in A$ and $v \in \mathcal{F}^{(k+1 \bmod T)}(w)$. Then v has a factor at position i belonging to $\mathcal{MF}^{(k+1+i \bmod T)}(w)$. This contradicts the fact that u does not belong to $\bigcup_{i=1}^{T-1}(A^T)^*A^i\mathcal{F}^{(k+i \bmod T)}(w)A^* \cup (A^T)^+\mathcal{F}^{(k)}(w)A^*$. Hence $\mathcal{MF}^{(k)}(w)$ satisfies (4.1). □

Thus we can think of the elements of $\mathcal{MF}^{(k)}(w)$ as the finite words $v = a_1a_2\ldots a_n$ such that:

1. $v = a_1a_2\ldots a_n \notin \mathrm{Fact}^{(k)}(w)$,

2. $a_1 a_2 \dots a_{n-1} \in \mathrm{Fact}^{(k)}(w)$,

3. $a_2 \dots a_n \in \mathrm{Fact}^{(k+1 \bmod T)}(w)$.

4.2 Minimal Periodic Forbidden Words of a Finite Word

We now describe an algorithm for computing the collection $\mathcal{MF}^{(k)}(w)$ of the minimal periodic forbidden words in phase k of the word w. The design of the algorithm is based on (4.1).

4.2.1 The Periodic-Suffix-Automaton

A preliminary step of the algorithm consists in computing, for any $0 \leq k \leq T-1$, a minimal deterministic automaton accepting $\mathrm{Suff}^{(k)}(w)$. This operation can be performed in time $O(T \times |w| \times \log |A|)$.

First, the computation of a minimal deterministic automaton accepting the set $\mathrm{Suff}^{(k)}(w)$ is reduced to the computation of a minimal deterministic automaton accepting the set $\mathrm{Suff}^{(0)}(w[k\,..\,|w|-1])$. Hence, we will assume, without loss of generality, that $k = 0$. The computation of a minimal deterministic automaton accepting $\mathrm{Suff}^{(0)}(w)$ is an extension of the known computation of the minimal automaton of the suffixes of a word, also called the directed acyclic word graph (DAWG) of a word (see for instance [7, 8] or [25, section 5.4 pp. 179-192]).

The states of this automaton are the equivalence classes of the syntactic congruence associated with the language $\mathrm{Suff}^{(0)}(w)$ defined as follows: if $u \in \mathrm{Fact}(w)$, we denote by $F_w(u)$ the *future* of u relative to $\mathrm{Suff}^{(0)}(w)$. Thus $F_w(u) = \{v \mid uv \in \mathrm{Suff}^{(0)}(w)\}$. Note that $F_w(w)$ is reduced to the empty word, and that $F_w(u)$ is the empty set if $u \notin \mathrm{Fact}^{(0)}(w)$. The words u and v are equivalent if and only $F_w(u) = F_w(v)$.

Moreover, the automaton has a transition labelled by a from the class of a word u to the class of ua. If $u \in \mathrm{Fact}(w)$, we define its image $s(u)$ by the *suffix function* s as the longest suffix v of u in $\mathrm{Suff}^{(0)}(u)$ such that $F_w(v) \neq F_w(u)$. In this case, $F_w(u) \subseteq F_w(v)$.

The algorithm generating the minimal deterministic automaton accepting the set $\mathrm{Suff}^{(0)}(w)$ is described in Procedure Periodic-Suffix-Automaton. It is an incremental algorithm that computes successively a minimal automaton accepting $\mathrm{Suff}^{(0)}(w[0\,..\,i])$, for i going from 0 to $|w|-1$. This procedure calls procedures Extension and Split. Procedure Extension performs the transformations needed to get a minimal automaton accepting $\mathrm{Suff}^{(0)}(w[0\,..\,i])$ from a minimal automaton accepting $\mathrm{Suff}^{(0)}(w[0\,..\,i-1])$. Some dummy states are added during the construction. The suffix link is not defined for these dummy states. The transitions belonging to some longest path from the initial state to some other state are called *solid* while the others are called *weak*. If p is a state of the automaton, the sequence of states $p, s(p), s(s(p)), \ldots$ is finite and ends with a dummy state. This sequence is called the *suffix path* of p. If p is the class of w, the non-dummy states of its suffix path are the final states of the automaton.

Proposition 14 *Algorithm* Periodic-Suffix-Automaton *computes the minimal deterministic automaton accepting* $\mathrm{Suff}^{(0)}(w)$ *for a given period* T.

Proof The proof is an extension to the periodic case of the correctness proof of the computation of the minimal deterministic automaton accepting the set of all suffixes of w (see [25, section 5.4 pp. 179-192]). We omit the proof but we mention below the main differences needed to take the period into account. If p is a state such that $l(p) < T$, the suffix link $s(p)$ is the dummy state $-T + l(p)$. Let us assume that we are at step i, lines 5-6 of Procedure Periodic-Suffix-Automaton(w, T). Let us denote $w[0\,..\,i-1]$ by w. Let r be the state obtained at the end of the loop in lines 4-6 of Procedure Extension(p, a). If r is a dummy state, for any word u in $\mathrm{Suff}^{(0)}(w)$, either $F_w(u) = F_w(w) = \{\varepsilon\}$ or $ua \notin \mathrm{Fact}^{(0)}(w)$. □

Proposition 15 *The size of the minimal automaton accepting* $\mathrm{Suff}^{(0)}(w)$ *for a given period* T *is linear in the size of* w. *Algorithm* Periodic-Suffix-Automaton *runs in time linear in the size of* w.

Proof The proof is similar to the proof in the aperiodic case [25, section 5.4 pp. 192]. □

Making all states final in the minimal deterministic automaton accepting the set $\mathrm{Suff}^{(k)}(w)$ gives a deterministic automaton accepting $\mathrm{Fact}^{(k)}(w)$. Note that this new

```
PERIODIC-SUFFIX-AUTOMATON (word w, period T)
 1. create T dummy states −1, −2, ... − T
 2. create an initial state 0
 3. set s(0) = −T
 4. let p = 0
 5. for i from 0 to |w| − 1 do
 6.       p = EXTENSION(p, w_i)
 7. let f = p
 8. while  f ≥ 0 do
 9.       set f final
10.       set f = s(f)
11. return automaton (Q, A, 0, E, {final})
```

```
EXTENSION (state p, letter a)
 1. create a new state q
 2. create a new solid transition (p, a, q)
 3. let r = s(p)
 4. while r ≥ 0 and there is no transition a going out of r do
 5.       create a weak transition (r, a, q)
 6.       set r = s(r)
 7. if r < 0
 8.       set s(q) = r + 1
 9. else
10.       let s = δ(r, a)
11.       if the transition (r, a, s) is solid
12.            set s(q) = s
13.       else
14.            set s(q) = SPLIT(r, a, s)
15. return q
```

SPLIT (state p, letter a, state q)
1. create a new state q'
2. **for** each transition (q, a, r) create a weak transition(q', a, r)
3. change the (weak) transition (p, a, q) into a solid transition (p, a, q')
4. set $s(q') = s(q)$
5. set $s(q) = q'$
6. let $t = s(p)$
7. **while** $t \geq 0$ **and** the transition (t, a, q) is weak **do**
8. change (t, a, q) into (t, a, q')
9. set $t = s(t)$
10. **return** q'

automaton may not be the minimal one. An example is given in Figure 4.1.

4.2.2 Computing the Minimal Periodic Forbidden Words for a Finite Word

We now describe the PERIODIC-MF-TRIES algorithm, that computes the minimal periodic forbidden words for a word w, with a fixed period T.

We denote by $\mathcal{A}_k = (Q_k, A, i_k, Q_k, \delta_k)$ a deterministic automaton accepting $\mathrm{Fact}^{(k)}(w)$, that is the set of blocks, factors of w, beginning at a position equal to k modulo T. From the automata $\mathcal{A}_k$, the algorithm outputs the tries $\mathcal{T}_k$ accepting the sets $\mathcal{MF}^{(k)}(w)$.

An example of this computation is described in Figure 4.2.

Proposition 16 *Algorithm* PERIODIC-MF-TRIES *computes from the automata* $\mathcal{A}_k$ *accepting* $\mathrm{Fact}^{(k)}(w)$ *the set of tries accepting the minimal periodic forbidden words of* w.

Proof Again, the proof is an extension of the correctness proof of the computation of the minimal forbidden words of a word from the factor automaton of w (see [25, section 6.5 pp. 182] or [26]). □

Proposition 17 *Algorithm* PERIODIC-SUFFIX-AUTOMATON*, followed by algorithm* PERIODIC-MF-TRIES*, runs in time* $O(|w| \times T \times \log |A|)$.

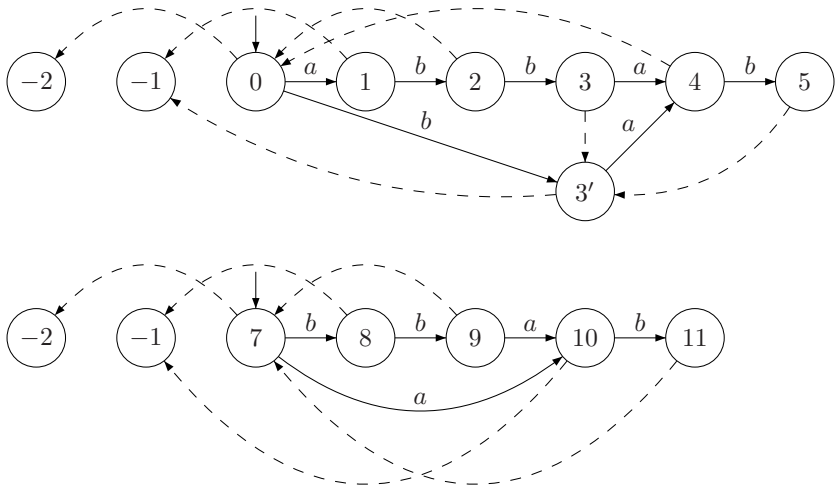

Figure 4.1: Deterministic automata $\mathcal{A}_0$ and $\mathcal{A}_1$ accepting $\mathrm{Fact}^{(0)}(w)$ and $\mathrm{Fact}^{(1)}(w)$ respectively, with the period $T = 2$ and $w = abbab$. The suffix links are represented by dashed edges.

PERIODIC-MF-TRIES (factor automata
$\mathcal{A}_k = (Q_k, A, i_k, F_k, \delta_k))_{0 \le k \le T-1}$, integer T)
1. **for** each $a \in A$
2. **if** $\delta_k(i_k, a)$ defined
3. set $\delta(i_k, a) = \delta_k(i_k, a)$
4. set $f(\delta(i_k, a)) = i_{k+1 \bmod T}$
5. **else**
6. set $\delta(i_k, a) =$ new sink
7. **for** each state $p \in Q_k$ in width-first search from $\cup_k \{i_k\}$ **and** each $a \in A$
8. **if** $\delta_k(p, a)$ undefined **and** $\delta_{k+1 \bmod T}(f(p), a)$ defined
9. set $\delta(p, a) =$ new sink
10. **else if** $\delta_k(p, a) = q$ **and** q not already reached
11. set $\delta(p, a) = q$
12. set $f(\delta(p, a)) = \delta(f(p), a)$
13. **return** $(\mathcal{T}_k = (Q_k, A, i_k, \{\text{sinks}\}, \delta))_{0 \le k \le T-1}$;

Proof The complexity is straightforward. □

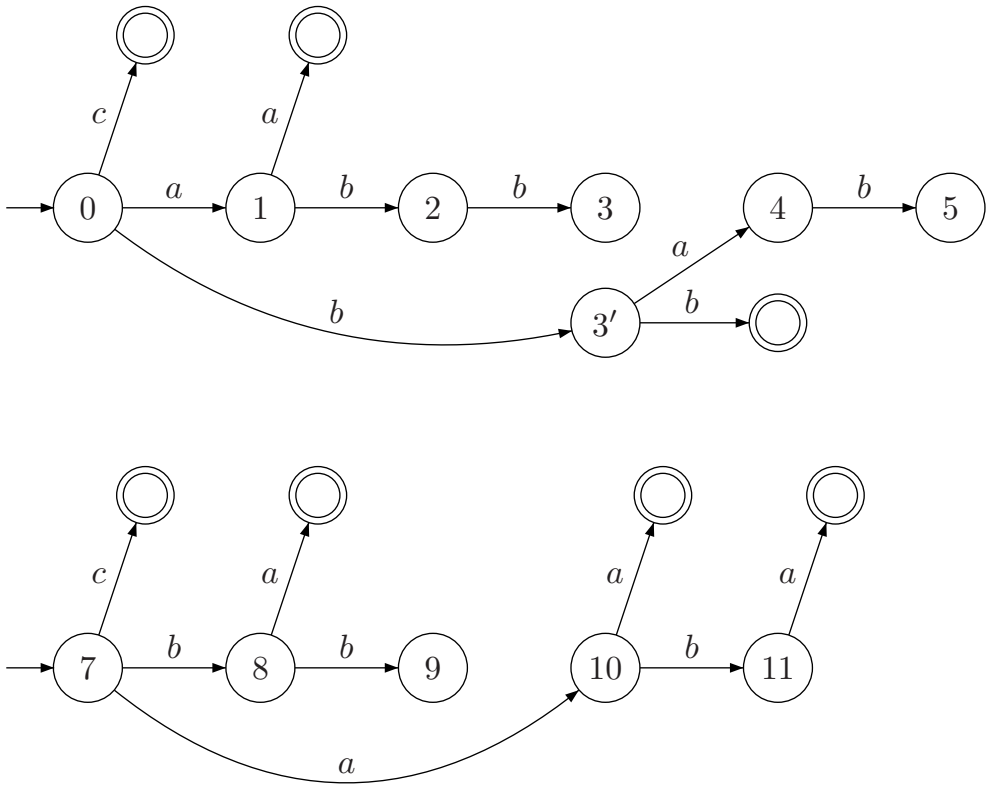

Figure 4.2: Output tries $\mathcal{T}_0$ and $\mathcal{T}_1$ of Algorithm PERIODIC-MF-TRIES for the input tries $\mathcal{A}_0$ and $\mathcal{A}_1$ described in Figure 4.1. The sink states are double circled. The tries $\mathcal{T}_0$ and $\mathcal{T}_1$ accept $\mathcal{MF}^{(0)}(w) = \{c, aa, bb\}$ and $\mathcal{MF}^{(1)}(w) = \{c, ba, aa, aba\}$ respectively.

4.3 Minimal Periodic Forbidden Words of a Finite Set of Finite Words

In this section we generalize the results of the previous section to the case of a finite set of finite words.

4.3.1 The Periodic-Multi-Suffix-Automaton

Let $P = \{w^1, \ldots, w^r\}$ be a set of finite words and let T be a positive integer, called the period. We note by $\mathrm{Suff}^{(k)}(P)$ the set of suffixes in phase k mod T of all words in P, that is:

$$\mathrm{Suff}^{(k)}(P) = \bigcup_{j=1}^{r} \mathrm{Suff}^{(k)}(w^j).$$

The following algorithm Periodic-Multi-Suffix-Automaton computes the set $\mathrm{Suff}^{(0)}(P)$. It is an incremental algorithm that computes successively an automaton accepting $\mathrm{Suff}^{(0)}(w^j[0\,..\,i])$, for i going from 0 to $|w^j|-1$, and from j going from 1 to r.

This procedure calls procedures Periodic-Multi-Extension and Split. Procedure Periodic-Multi-Extension performs the transformations needed to get an automaton accepting the set $\mathrm{Suff}^{(0)}(w^j[0\,..\,i])$ from an automaton accepting the set $\mathrm{Suff}^{(0)}(w^j[0\,..\,i-1])$. It calls procedure Split, which is the same as in the single-word case.

```
Periodic-Multi-Suffix-Automaton (P = {w^1, ..., w^r}, period T)
 1. create T dummy states -1, -2, ..., -T
 2. create an initial state 0
 3. set s(0) = -T
 4. for every word w^i, from i = 1 to i = r, do
 5.       let p = 0
 6.       for j from 0 to |w^i| - 1 do
 7.            p = Multi-Extension(p, w^i_j)
 8. let f = p
 9. while  f ≥ 0 do
10.       set f final
11.       set f = s(f)
12. return automaton (Q, A, 0, E, {final})
```

4.3.2 Computing Minimal Periodic Forbidden Words for P

In an analogous way as we did in the non-periodic case, we shall denote by $DAWG^{(k)}(P)$ the automata obtained from $\mathcal{A}_k^P$ by setting all states terminal. These

```
PERIODIC-MULTI-EXTENSION (state p, letter a)
 1. if there is a transition a going out of p
 2.       set q = δ(p, a)
 3.       if the transition (p, a, q) is weak
 4.             set p = SPLIT(p, a, q)
 5. else
 6.       create a new state q
 7.       create a new solid transition (p, a, q)
 8.       let r = s(p)
 9.       while r ≥ 0 and there is no transition a going out of r do
10.             create a weak transition (r, a, q)
11.             set r = s(r)
12.       if r < 0
13.             set s(q) = r + 1
14.       else
15.             let s = δ(r, a)
16.             if the transition (r, a, s) is solid
17.                   set s(q) = s
18.             else
19.                   set s(q) = SPLIT(r, a, s)
20. return q
```

automata recognize the sets $\mathrm{Fact}^{(k)}(P)$, even if they may be not the minimal ones.

Now, we can extend algorithm Periodic-MF-tries to the set P, simply by giving it as input the automata $DAWG^{(k)}(P)$. This is possible since the graphs of the automata $\mathcal{A}_k^P$, and hence those of $DAWG^{(k)}(P)$, are in fact acyclic graphs. So, on the inputs $DAWG^{(k)}(P)$, the algorithm Periodic-MF-tries outputs the tries $\mathcal{T}_k^P$ accepting the minimal periodic forbidden words in phase k for the set P, *i.e.* the sets $\mathcal{MF}^{(k)}(P)$.

```
Periodic-MF-tries (factor automata
    A_k^P = (Q_k, A, i_k, F_k, δ_k))_{0≤k≤T−1}, integer T)
 1. for each a ∈ A
 2.     if δ_k(i_k, a) defined
 3.         set δ(i_k, a) = δ_k(i_k, a)
 4.         set f(δ(i_k, a)) = i_{k+1 mod T}
 5.     else
 6.         set δ(i_k, a) = new sink
 7. for each state p ∈ Q_k in width-first search from ∪_k{i_k}
    and each a ∈ A
 8.     if δ_k(p, a) undefined and δ_{k+1 mod T}(f(p), a) defined
 9.         set δ(p, a) = new sink
10.     else if δ_k(p, a) = q and q not already reached
11.         set δ(p, a) = q
12.         set f(δ(p, a)) = δ(f(p), a)
13. return (T_k^P = (Q_k, A, i_k, {sinks}, δ))_{0≤k≤T−1};
```

Proposition 18 *Algorithm* Periodic-MF-tries *applied on the set P computes from the automata $DAWG^{(k)}(P)$, accepting the sets $\mathrm{Fact}^{(k)}(w)$, the set of tries accepting the minimal periodic forbidden words for the set P.*

Proof The proof directly follows from propositions 16 and 11. □

Proposition 19 *Algorithm* Periodic-Multi-Suffix-Automaton *followed by algorithm* Periodic-MF-tries *on the input set P runs in time $O(|P| \times T \times \log |A|)$.*

Proof The complexity is straightforward. □

Part II

Applications of Minimal Forbidden Words

Chapter 5

Applications to Constrained Systems

In this chapter we show an application of minimal forbidden words to coding for constrained systems, in particular for constrained systems with unconstrained positions.

Introduction

Recording systems often use combined modulation/error-correction codes (ECC codes). While error-correction codes enable the correction of a certain number of channel errors, modulation codes encode the sequences into a constrained channel that is supposed to reduce the likelihood of errors. Well known examples of such channels are the maximum transition run systems MTR(j) [48], where the maximum run of consecutive 1's is j, or the run length limited systems RLL (d, k), where the maximum run of consecutive 0's is k and the minimum run of consecutive 0's is d. Among various schemes proposed to construct both error-correction codes and modulation codes, one of them, called the Wijngaarden-Immink scheme [59] (see also [11]), proposes to encode an unconstrained sequence of bits into a constrained sequence in which certain bit positions are reserved for ECC parity. The bit values in these positions can be flipped (or not flipped) independently without violating the constraint. These positions are called *unconstrained positions*. Therefore, ECC parity information can be inserted into the unconstrained positions of the

modulation-encoded sequences without making them out of the constrained channel.

In [11], the authors study different approaches to build such codes, one of them being based on the construction of the unique maximal subsystem of a constrained system S such that any position modulo T in U is unconstrained, where U is a given subset of integers modulo some integer T. We call this system the (U, T)-unconstrained subsystem of S. The knowledge of this maximal subsystem enables the computation of the maximal possible rate of a code that both satisfies a given constraint and is unconstrained in a specified set of positions. Indeed, this maximal rate is the Shannon capacity of the maximal subsystem. It also enables to apply standard modulation code constructions to this sub-channel [41]. Since these code constructions work on a presentation of the sub-channel, it is worth to efficiently compute a small presentation of this sub-channel.

We want to focus on the construction of this maximal subsystem for a finite-state constrained system with finite memory. Our goal is to reduce the time and space complexities of the general solution proposed in [11]. We consider a finite-memory constrained system S defined by a finite list of forbidden blocks. Given such a system and a subset U of integers modulo some integer T, we construct in a polynomial amount of time and space a finite-state graph that presents the (U, T)-unconstrained subsystem of S. The maximal subsystem appears to be a natural example of *periodic-finite-type systems* (PFT) introduced by Moision and Siegel in [47]. This was already noticed in [11, pp. 869].

In our process, we start with the construction of a periodic list of forbidden blocks that defines the maximal subsystem from a finite list of forbidden blocks of the finite-memory system. More precisely, if the input data is a trie $\mathcal{T}$ representing a finite prefix-free list of forbidden blocks, the algorithm works in space and time $O(T \times |A| \times |\mathcal{T}| \times \log |\mathcal{T}|)$, where $|\mathcal{T}|$ is the size (the number of states) of the trie and A is the alphabet. In a second step, we construct in linear time and space a finite-state presentation of a periodic-finite-type shift defined by a periodic list of forbidden blocks. The whole two-step process computes a finite-state presentation of the maximal unconstrained subsystem. Moreover, our algorithm becomes linear if the input trie has itself a linear structure. For instance, it runs in $O(j)$ time for the MTR(j) constraint, and in $O(k)$ time for the RLL (d, k) constraint with the input data $d, k\,(d \leq k)$, if the period T of the unconstrained positions is naturally assumed

to be constant. We restrict ourselves to binary systems, but the results carry over easily to constrained finite-memory systems over any finite set of symbols.

While our algorithm is polynomial and the algorithm given in [11] is exponential in the general case, they cannot be compared directly for the following reasons. The algorithm described in [11] works in an exponential amount of space and time for all finite-state systems given by a finite-state presentation, and in quadratic space and time for finite-memory systems with an additional condition called the gap condition. The gap condition limits the number of unconstrained positions relatively to the memory of the system. An efficient algorithm is also proposed in [11] for the special case of MTR systems. We point out that, although our algorithm has a better complexity for finite-memory systems, and no restriction similar to the gap condition, it works with different input data. Indeed, it is possible to compute in polynomial time an automaton accepting a list of forbidden blocks of a finite-memory system given by a deterministic automaton with a single initial state [4]. But it is not possible to do it in polynomial time from a presentation where all states are initial ones. Thus our algorithm runs faster if the input data are a list of forbidden blocks while the one presented in [11] is more efficient if both the input data are a presentation of the constraint and the gap condition is satisfied.

In Section 5.1, we recall some background regarding constrained systems with unconstrained positions, which are introduced in [11].

In Section 5.2 we give a linear construction of a finite-state presentation of a periodic-finite-type shift defined by a periodic list of forbidden blocks.

In Section 5.3, we combine the algorithm given in Section 5.2 to a preliminary treatment of the input trie presenting a list of forbidden blocks of the constrained channel.

5.1 Background and Basic Definitions

We recall definitions that can be found in [34]. Let $A = \{0, 1, \ldots, k\}$ be a finite alphabet, with $k \geq 1$. We denote by A^* the set of finite words on A, by $A^{\mathbb{Z}}$ the set of bi-infinite sequences $x = \cdots x_{-3}x_{-2}x_{-1}x_0x_1x_2x_3\cdots$ drawn from A, and by $A^{\mathbb{N}}$ the set of right-infinite ones.

The shift map σ transforms a sequence $(x_i)_{i\in\mathbb{Z}}$ into the sequence $(x_{i+1})_{i\in\mathbb{Z}}$. If

$i \leq j$ are integers, we denote by $x[i\,..\,j]$ the factor or sub-block $x_i \dots x_j$ of a finite or infinite word x. A finite word w is a sub-block of a finite or infinite word x at position i if $w = x[i\,..\,i + |w| - 1]$, where $|w|$ is the length of w. We denote this fact by $w \prec_i x$. note that $w = w[0\,..\,|w| - 1]$.

An automaton is a finite labelled multi-graph (or simply a graph). It is a tuple (Q, A, E), where Q is a finite set of states, A is the labelling alphabet, and E is a finite set of edges labelled with elements in the alphabet A. An automaton *accepts* a set of finite words when initial and final states are specified. A finite word is then accepted if it is the label of a finite path from an initial state to a final one. The set of bi-infinite labels of paths in an automaton is called a *constrained system*, or also a *sofic shift* in the symbolic dynamics terminology. The automaton is then called a *presentation* of the shift. In that case, the initial and final states may not be specified since all states are supposed to be both initial and final.

An automaton is *deterministic* if for any given state and any given symbol, there is at most one outgoing edge labelled by a given symbol. A sofic shift is *irreducible* if it has a presentation with a strongly connected graph. In an *essential presentation* all states have at least one outgoing edge and one incoming edge. An automaton has *finite memory* M (or also is M*-local* or M*-definite*) if whenever any two paths of the automaton of length M have the same label sequence, they end at the same state. *Finite-memory systems* or *finite-type systems* or *shifts of finite type* (SFT) have a finite-memory presentation. Examples of such systems include the RLL and MTR constraints.

Finite-type shifts are characterized by a finite collection of *forbidden blocks*. If $\mathcal{F}$ is a finite subset of A^*, we denote by $\mathsf{X}_{\mathcal{F}}$ the shift of finite type defined by the set of forbidden words $\mathcal{F}$. A bi-infinite word x belongs to $\mathsf{X}_{\mathcal{F}}$ if and only if $w \prec_i x$, for some index i, implies $w \notin \mathcal{F}$. Any irreducible sofic shift has a unique minimal deterministic presentation called the *right Shannon cover* of the shift.

Periodic-finite-type shifts are constrained systems with a time-varying constraint. They have been introduced by Moision and Siegel in [47]. They provide suitable representations of constrained systems that forbid the appearance of certain patterns in a periodic manner.

Let T be a positive integer, called the period. Let $\mathcal{F}$ be a finite collection of finite words over A, where each $w_i \in \mathcal{F}$ is associated with an integer n_i in

the set $\{0, 1 \dots T-1\}$, called the set of *phases*. The collection $\mathcal{F}$ is denoted by $\mathcal{F} = \{(w_1, n_1), \dots, (w_{|\mathcal{F}|}, n_{|\mathcal{F}|})\}$ and called a collection of *periodic forbidden words*. For $0 \leq k < T$, $\mathcal{F}^{(k)}$ denotes the subset of $\mathcal{F}$ associated with the phase k. We denote by $\mathsf{X}_{\{\mathcal{F},T\}}$ the shift defined as the set of bi-infinite sequences having a shifted sequence that does not contain a word $(w_j, n_j) \in \mathcal{F}$ starting at any index $i = n_j \bmod T$. More precisely, a bi-infinite word x belongs to $\mathsf{X}_{\{\mathcal{F},T\}}$ if and only if there is an integer k such that $\sigma^k(x) = y$ and, for each integer i, one has $w \prec_i y \Rightarrow w \notin \mathcal{F}^{(i \bmod T)}$. A *periodic-finite-type shift for a period* T (PFT(T)) is a constrained system S such that there is a collection of periodic forbidden words $\mathcal{F}$ with $S = \mathsf{X}_{\{\mathcal{F},T\}}$. A *periodic-finite-type shift* (PFT) is a PFT(T) for some period T. An example is given in Figure 5.1.

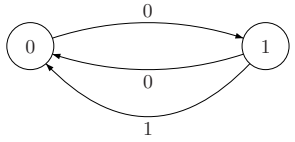

Figure 5.1: An automaton presenting a periodic-finite-type shift X. The shift X admits the following list of periodic forbidden words, for $T = 2$, $\mathcal{F}^{(0)} = \{1\}, \mathcal{F}^{(1)} = \emptyset$.

Note that a shift of finite type is of periodic-finite-type for any period.

Constrained systems with unconstrained positions are defined in [11] as follows. Let S be a constrained system, T a positive period, and $U \subseteq \{0, \dots, T-1\}$, called the set of unconstrained positions. For any finite (resp. right-infinite, bi-infinite) word x, a *U-flip* of x is a finite (resp. right-infinite, bi-infinite) word y such that $y_i = x_i$ whenever $i \bmod T \notin U$. If A is the two-letter alphabet $\{0, 1\}$, a U-flip is obtained by flipping (or not) the bit values in the unconstrained positions. The set of all U-flips of words of a set X is called the *U-closure* of X.

We denote by $S_{U,T}$ the set of all infinite (right-infinite or bi-infinite according to the context) sequences x of S such that

- all U-flips of x belong to S,
- $x_i = 1$ for all positions i such that $i \bmod T \in U$.

The unconstrained positions are forced to be 1 in order to fix a leader in each U-flip class of a word. The important fact is that one can independently change the values in the unconstrained positions without violating the constraint defined by S. Note that the shifted sequence of a sequence in $S_{U,T}$ may not be in $S_{U,T}$ (*i.e.*$S_{U,T}$ is not a shift). We denote the set of all these shifted sequences by $S^{\sigma}_{U,T}$. We also denote by $\overline{S_{U,T}}^{\sigma}$ the set of all bi-infinite shifted sequences of $S^{\sigma}_{U,T}$.

Note that $S^{\sigma}_{U,T} \subseteq \overline{S_{U,T}}^{\sigma}$.

An algorithm to compute a presentation of $S_{U,T}$ from a presentation of S is given in [11]. The result is a deterministic automaton $G_{U,T}$ whose graph has a period that is a multiple of T with the following properties:

- states of $G_{U,T}$ are partitioned according to T phases $\{0, \ldots, T-1\}$ in such a way that if a state has phase k, its successors have phase $k+1 \bmod T$.
- the transitions beginning in a state of a phase in U are labelled by 1.
- $S_{U,T}$ is the set of right-infinite sequences of $G_{U,T}$ that are labels of a path starting in a state of phase 0.

The link between constrained systems with unconstrained positions and periodic-finite-type shifts is given in the proposition below which is stated in [11, p. 869] without proof. We use the following notation: if U is a subset of $\{0, \ldots, T-1\}$, and k is an integer, we denote by $U+k$ the set $\{u+k \bmod T \mid u \in U\}$.

Proposition 20 *Let S be a finite-type shift, T a period and U a set of unconstrained positions. The shifts $\overline{S_{U,T}}^{\sigma}$ and $S^{\sigma}_{U,T}$ are PFT(T) shifts.*

Proof Let $\mathcal{F}$ be a finite collection of finite forbidden words such that $S = \mathsf{X}_{\mathcal{F}}$. We define two collections of periodic forbidden words $\mathcal{G}$ and $\mathcal{G}'$ as follows. If $k \in \{0, \ldots, T-1\}$, then $\mathcal{G}'^{(k)}$ is the $(U-k)$-closure of $\mathcal{F}$ and $\overline{S_{U,T}}^{\sigma} = \mathsf{X}_{\{\mathcal{G}', T-1\}}$. If $k \in U$, then $\mathcal{G}^{(k)} = \{0\} \cup \mathcal{G}'^{(k)}$. If $k \in \{0, \ldots, T-1\} \setminus U$, then $\mathcal{G}^{(k)} = \mathcal{G}'^{(k)}$. Then $S^{\sigma}_{U,T} = \mathsf{X}_{\{\mathcal{G}, T\}}$.

Let us detail for instance the equality $\overline{S_{U,T}}^{\sigma} = \mathsf{X}_{\{\mathcal{G}', T\}}$. Let x be a bi-infinite word of $\overline{S_{U,T}}^{\sigma}$. Thus, there is an integer i with $\sigma^i(x) = y$, and y belongs to the U-closure of $S_{U,T}$. Thus y has a U-flip z in S. Let w a finite block with $w \prec_k y$. There is a $(U-k)$-flip of w that is not in $\mathcal{F}$. Thus w does not belong to the $(U-k)$-closure

of $\mathcal{F}$. This proves that $\overline{S_{U,T}}^{\sigma} \subseteq \mathsf{X}_{\{\mathcal{G}',T\}}$. Conversely, let x be a bi-infinite word of $\mathsf{X}_{\{\mathcal{G}',T\}}$. There is an integer i with $\sigma^i(x) = y$, and, for each integer k, one has $w \prec_k y \Rightarrow w \notin \mathcal{G}'^{(k \bmod T)}$. Let z be a U-flip of y. Let w' be the block obtained from w with the same U-flip. Then $w' \prec_k z$. Since w does not belong to the $(U-k)$-closure of $\mathcal{F}$, w' does not belong either. It follows that $w' \notin \mathcal{F}$. Thus any U-flip of y belongs to S. Hence y belongs to the U-closure of $S_{U,T}$, and $\mathsf{X}_{\{\mathcal{G}',T\}} \subseteq \overline{S_{U,T}}^{\sigma}$. □

Note that the result of the previous proposition extends as follows if S is a periodic-finite-type system for a period that is a multiple of T.

Proposition 21 *Let S be a PFT(T) shift, and U a set of unconstrained positions. The shifts $\overline{S_{U,T}}^{\sigma}$ and $S^{\sigma}_{U,T}$ are unions of PFT(T) shifts.*

Proof Suppose $S = \mathsf{X}_{\{\mathcal{F},T\}}$. We first fix $k_0 \in \{0, 1, \ldots, T-1\}$, and define the two collections of periodic forbidden words $\mathcal{G}_{k_0}$ and $\mathcal{G}'_{k_0}$ as follows. If $k \in \{0, \ldots, T-1\}$, then $\mathcal{G}'^{(k)}_{k_0}$ is the $(U-k)$-closure of $\mathcal{F}^{(k+k_0 \bmod T)}$. Hence $\overline{S_{U,T}}^{\sigma} = \bigcup_{k_0 \in \{0,1,\ldots,T-1\}} \mathsf{X}_{\{\mathcal{G}'_{k_0},T\}}$.

If $k \in U$, then $\mathcal{G}^{(k)}_{k_0} = \{0\} \cup \mathcal{G}'^{(k)}_{k_0}$. If $k \in \{0, \ldots, T-1\} \setminus U$, then $\mathcal{G}^{(k)}_{k_0} = \mathcal{G}'^{(k)}_{k_0}$. Hence $S^{\sigma}_{U,T} = \bigcup_{k_0 \in \{0,1,\ldots,T-1\}} \mathsf{X}_{\{\mathcal{G}_{k_0},T\}}$. □

Let $\mathcal{F}$ be a list of periodic forbidden words of a shift X for a given positive period T. We say that $\mathcal{F}$ is *periodic anti-factorial* if for any $0 \leq i \leq T-1$, $w \in \mathcal{F}^{(i)}$ implies that, for any proper factor u of w with $u \prec_j w$, $u \notin \mathcal{F}^{(i+j \bmod T)}$. The notion of periodic anti-factorial list generalizes the notion of anti-factorial language (see for instance [26]). In the aperiodic case, an anti-factorial language means a language where no word is the factor of another one, while a factorial language is a language where each factor of a word of the language also belongs to the language (see [26]). In particular, the sets $\mathcal{F}^{(i)}$ of an anti-factorial list $\mathcal{F}$ of periodic forbidden words, are prefix-free codes, *i.e.*sets of words where no word is a proper prefix of another word of the set. The empty word never belongs to any $\mathcal{F}^{(i)}$.

Example 1 The list $\mathcal{F}^{(0)} = \{00, 11\}, \mathcal{F}^{(1)} = \{00, 11, 010\}$ with $T = 2$ is periodic anti-factorial while the list $\mathcal{F}^{(0)} = \{00, 11, 010\}, \mathcal{F}^{(1)} = \{00, 10\}$ with $T = 2$ is not. Indeed, in the latter list, $10 \in \mathcal{F}^{(1)}$, $010 \in \mathcal{F}^{(0)}$, and $10 \prec_1 010$.

Proposition 22 *Let $\mathcal{F}$ be a list of periodic forbidden words of a PFT(T) shift X. Then there is an anti-factorial list of periodic forbidden words $\mathcal{F}'$ of X with the same period, such that $\mathcal{F}'^{(i)} \subseteq \mathcal{F}^{(i)}$ for any $0 \leq i \leq T-1$.*

Proof We define the list $\mathcal{F}'$ by

$$\mathcal{F}'^{(i)} = \mathcal{F}^{(i)} - \mathcal{F}^{(i)}A^+ - (A^T)^+\mathcal{F}^{(i)}A^* - \bigcup_{j=1}^{T-1}(A^T)^*A^j\mathcal{F}^{(i+j \bmod T)}A^*,$$

where A^* denotes the set of all finite words over A and A^+ the set of all non-empty ones. Note that $\mathcal{F}'^{(i)}$ is obtained from $\mathcal{F}^{(i)}$ by removing all words that contain a strict factor in position k belonging to $\mathcal{F}^{(k+i \bmod T)}$. By construction $\mathcal{F}'$ is periodic and anti-factorial, and $X = \mathsf{X}_{\{\mathcal{F}',T\}}$. □

The notion of anti-factorial list is weaker than the notion of minimal list of periodic forbidden words (see [47] for a notion of minimality, where minimal periodic forbidden words are called *periodic first offenders*). This notion is however a key point in the algorithms described in Section 5.2.

5.2 Computation of the Shift Defined by Periodic Forbidden Words

In this section, we describe an algorithm that computes the shift $\mathsf{X}_{\{\mathcal{F},T\}}$ from a finite list of periodic forbidden words $\mathcal{F}$ with period T. This algorithm extends to the periodic case an algorithm of Crochemore *et al.* [26] that computes the language avoiding the blocks defined by an anti-factorial language. We first assume that the periodic forbidden list is anti-factorial, and show later how to remove this restriction.

We denote by $\mathcal{B}^0(\mathcal{F},T)$ the set of finite blocks w such that, for any integer $0 \leq i \leq |w|$, $u \prec_i w \Rightarrow u \notin \mathcal{F}^{(i \bmod T)}$. The set of finite blocks or factors of $\mathsf{X}_{\{\mathcal{F},T\}}$ is denoted by $\mathcal{B}(\mathsf{X}_{\{\mathcal{F},T\}})$. Note that $\mathcal{B}^0(\mathcal{F},T) \subseteq \mathcal{B}(\mathsf{X}_{\{\mathcal{F},T\}})$. The inclusion is strict in general. For instance, if $\mathcal{F}^{(0)} = \{010\}$, $\mathcal{F}^{(1)} = \{101\}$ and $T = 2$, $010 \notin \mathcal{B}^0(\mathcal{F},T)$ since $010 \in \mathcal{F}^{(0)}$, and $010 \in \mathcal{B}(\mathsf{X}_{\{\mathcal{F},T\}})$.

Moreover, if $w \in \mathcal{B}(\mathsf{X}_{\{\mathcal{F},T\}})$, there is a finite block u such that $uw \in \mathcal{B}^0(\mathcal{F},T)$. Hence $\mathcal{B}(\mathsf{X}_{\{\mathcal{F},T\}})$ is included in the set of factors of $\mathcal{B}^0(\mathcal{F},T)$.

Let $\mathcal{F}$ be an anti-factorial list of periodic forbidden words with period T. We associate with $\mathcal{F}$ the finite deterministic automaton $\mathcal{D}(\mathcal{F})$ described below. A finite word is accepted by this automaton if it is the label of a path from an initial state to a final one. As shown in Proposition 24, $\mathcal{D}(\mathcal{F})$ accepts the set $\mathcal{B}^0(\mathcal{F},T)$ of finite

blocks of $\mathsf{X}_{\{\mathcal{F},T\}}$ appearing in phase 0. An essential presentation of the PFT shift $\mathsf{X}_{\{\mathcal{F},T\}}$ is obtained from $\mathcal{D}(\mathcal{F})$ by removing the states that have no outgoing edges or no incoming edges.

The automaton $\mathcal{D}(\mathcal{F})$ is defined by the tuple (Q, A, i, F, δ) as follows:

- the set Q of states is $\bigcup_{0\leq k\leq T-1} Q_k$, where $Q_k = \{(w,k) \mid w$ is a prefix of a word in $\mathcal{F}^{(k)}\}$,
- A is the current alphabet,
- the initial state i corresponds to the empty word $(\epsilon, 0)$,
- the set F of final states is $Q \setminus \bigcup_{0\leq k\leq T-1} F_k$, where $F_k = \{(w,k) \mid w \in \mathcal{F}^{(k)}\}$.

The states of $\bigcup_{0\leq k\leq T-1} F_k$ are called *sink states*. The set of transitions T is defined as follows:

- $\mathsf{T} = \{((u,k), a, (v, k + r \bmod T)) \mid (u,k) \in Q_k \setminus F_k, a \in A$, and v is the longest suffix $(ua)[r\,..\,|ua|-1]$ of ua such that $(v, k + r \bmod T) \in Q\}$, (transitions $((u,k), a, (ua,k))$ such that $(ua,k) \in Q_k$ are called forward edges while the others are called backward edges).

The partial transition function defined by transitions is denoted by δ. If w is a finite word and q a state, $\delta(q,w)$ is defined if and only if there is a path starting at q with label w. In that case, this path is unique and $\delta(q,w)$ is its ending state. Note that there is no transition going out of a sink state, but $\delta(q,a)$ is defined for any letter a and any state q that is not a sink state.

Remarks One can easily prove from the definitions that

- If $q \in Q \setminus (F \cup \bigcup_{0\leq k\leq T-1}(\varepsilon, k))$, all transitions arriving on state q have the same label.
- If $q \in Q$, there is a path from q to a sink state in the automaton.

Lemma 23 *Let w be a finite word. If $\delta(i,w)$ is defined, then $\delta(i,w) = (v, r \bmod T)$, where v is the longest suffix $w[r\,..\,|w|-1]$ of w such that $(v, r \bmod T)$ is a state of Q.*

Proof We prove the lemma by induction on the length of w. If w is the empty word, the claim is trivially satisfied. Otherwise $w = ua$, where a is a letter. Hence, $\delta(i, w) = \delta(\delta(i, u), a)$. By inductive hypothesis, $\delta(i, u) = (u', k \bmod T)$, where u' is the longest suffix $u[k \,..\, |u| - 1]$ of u such that $(u', k \bmod T)$ is a state of Q. Since $\delta(i, ua)$ is defined, $\delta(i, u)$ is not a sink state and $(\delta(i, u), a, \delta(i, ua))$ is a transition of T.

If $\delta(i, u) = (u', k \bmod T)$, $\delta(i, ua) = (v, k + r \bmod T)$, where v is the longest suffix $(u'a)[r \,..\, |u'a| - 1]$ of $u'a$ such that $(v, k + r \bmod T)$ is a state of Q. Let v' be a nonempty suffix $(ua)[r' \,..\, |ua| - 1]$ of ua such that $(v', r' \bmod T)$ is a state of Q. Then $v' = w'a$, and w' is a suffix $u[r' \,..\, |u| - 1]$ of u such that $(w', r' \bmod T)$ is a state of Q. From the inductive hypothesis, we get that w' is a suffix of u', and thus $v' = w'a$ is a suffix of $u'a$. Then v is the longest suffix $(ua)[r \,..\, |ua| - 1]$ of ua such that $(v, r \bmod T)$ is a state of Q. □

Proposition 24 *Let $\mathcal{F}$ be a finite anti-factorial list of periodic forbidden words with period T. The automaton $\mathcal{D}(\mathcal{F})$ accepts $\mathcal{B}^0(\mathcal{F}, T)$. It is also a presentation of $\mathsf{X}_{\{\mathcal{F},T\}}$ after removing the sink states.*

Proof We first prove that $\mathcal{B}^0(\mathcal{F}, T)$ is included in the language accepted by $\mathcal{D}(\mathcal{F})$. Let w be a finite block of $\mathcal{B}^0(\mathcal{F}, T)$. If w is not accepted by $\mathcal{D}(\mathcal{F})$, $\delta(i, w)$ is not defined. Thus there is a prefix u of w such that $\delta(i, u) = (v, k)$ is a sink state. Hence v is a suffix $u[n \,..\, |u| - 1]$ of u, with $k = n \bmod T$, which belongs to $\mathcal{F}^{(k)}$. This implies that $v \prec_n w$, and $w \notin \mathcal{B}^0(\mathcal{F}, T)$.

Conversely, let us assume that $w \notin \mathcal{B}^0(\mathcal{F}, T)$. There is an integer k with $0 \leq k \leq |w|$, and a finite block $u \in \mathcal{F}^{(k \bmod T)}$, such that $u \prec_k w$. We denote by z the word $w[0 \,..\, k - 1]$. Hence zu is a prefix of w. If w is accepted by $\mathcal{D}(\mathcal{F})$, $\delta(i, zu)$ is defined. By Lemma 23, $\delta(i, zu) = (v, r \bmod T)$, where v is the longest suffix $(zu)[r \,..\, |zu| - 1]$ of zu such that $(v, r \bmod T)$ is a state of Q. Since $(u, k \bmod T)$ is a state of Q, $|v| \geq |u|$. Since u, v are suffixes of zu, $u \in \mathcal{F}^{(k \bmod T)}$ is a suffix of v that is a prefix of a word in $\mathcal{F}^{(r \bmod T)}$. The anti-factoriality of $\mathcal{F}$ implies that $k = r \bmod T$, and $u = v$. Thus $\delta(i, zu)$ is a sink state, and therefore w is not accepted by $\mathcal{D}(\mathcal{F})$, which is a contradiction. □

The above definition of the automaton $\mathcal{D}(\mathcal{F})$ turns into the algorithm below called Periodic–automaton that produces it. We first consider the code of this

algorithm without the lines 3.a, 3.b, 3.c and the lines 11.a, 11.b, 11.c. It builds the automaton $\mathcal{D}(\mathcal{F})$ from a finite anti-factorial collection of finite words. With all lines included, it builds the automaton from any finite collection of finite words. The input is thus a collection of T finite sets of finite words. Each finite set of words is represented by a tree-like deterministic automaton, called a *trie*, defined as follows.

PERIODIC–AUTOMATON (tries $\mathcal{T}_k = (Q_k, A, i_k, F_k, \delta_k)$ accepting $\mathcal{F}^{(k)}$, integer T)
1. set $Q = \bigcup_k Q_k$, $F = \bigcup_k F_k$, $i = i_0$.
2. **for** each $a \in A$ **and** each k, $0 \leq k \leq T-1$
3.a **if** $i_k \in F$, remove transition $\delta_k(i_k, a)$ in $\mathcal{T}_k$
3.b **if** $\delta_k(i_k, a)$ is defined **and** $i_{k+1 \bmod T} \in F$
3.c remove transition $\delta_k(i_k, a)$ in $\mathcal{T}_k$
4. **if** $\delta_k(i_k, a)$ is defined
5. set $\delta(i_k, a) = \delta_k(i_k, a)$
6. set $f(\delta(i_k, a)) = i_{k+1 \bmod T}$
7. **else**
8. set $\delta(i_k, a) = i_{k+1 \bmod T}$
9. **for** each k, each $p \in Q_k \setminus \{i_k\}$ in width-first search from $\bigcup_k i_k$
10. **and for** each $a \in A$
11.a **if** $p \in F$, remove transition $\delta_k(p, a)$ in $\mathcal{T}_k$
11.b **if** $\delta_k(p, a)$ is defined **and** $\delta(f(p), a) \in F$
11.c remove transition $\delta_k(p, a)$ in $\mathcal{T}_k$
12. **if** $\delta_k(p, a)$ is defined
13. set $\delta(p, a) = \delta_k(p, a)$
14. set $f(\delta(p, a)) = \delta(f(p), a)$
15. **else if** $p \notin \bigcup_k F_k$
16. set $\delta(p, a) = \delta(f(p), a)$
17. **else**
18. set $\delta(p, a)$ is undefined (or equal to p)
19. **return** automaton $\mathcal{A} = (Q, A, i, Q \setminus F, \delta)$

Let L be a finite language of finite words, a *trie* representing L is a finite deterministic automaton accepting L, where

- the set of states is the set of prefixes of words in L,
- the initial state is the empty word ε,
- the set of final states is F,

- the set of transitions is $\{(u, a, ua) \mid a \in A\}$.

The *size* of a trie $\mathcal{T}$ is defined as its number of states and it is denoted by $|\mathcal{T}|$.

The input of our algorithm is the set of tries $\mathcal{T}_k = (Q_k, A, i_k, F_k, \delta_k)$ that accept the finite sets $\mathcal{F}^{(k)}$, for $0 \leq k \leq T-1$ (see Figure 5.2). The output is the deterministic automaton accepting $\mathcal{D}(\mathcal{F})$. It is denoted by (Q, A, i, T, δ). An essential representation of $\mathsf{X}_{\{\mathcal{F},T\}}$ is obtained from it by removing the states that have no outgoing edges or no incoming edges, and by setting all states both initial and final.

The key point for the final efficiency is the use of a function f called a *failure function* and defined on the set Q, the union of the sets Q_k of states of the tries $\mathcal{T}_k$, as follows. A state of the trie $\mathcal{T}_k$ is identified with a pair (u, k), where u is a prefix of a word in $\mathcal{F}^{(k)}$. For a state $(au, k) \in Q$, $f(au, k)$ is $\delta(i_{k+1 \bmod T}, u)$. Note that $f(i_k)$ is undefined for any k such that $0 \leq k \leq T-1$, which justifies a specific treatment of the initial states in the algorithm. The failure function guarantees a good time complexity of the algorithm.

The shift $\mathsf{X}_{\{\mathcal{F},T\}}$, given in Figure 5.2, is presented by the deterministic automaton of Figure 5.3. The doubled circled states can be removed. For each state p, the value of the failure function is represented as the target of the dashed edge starting at p. States can be divided into two subsets, the set of states in phase 0 (in white) and the set of states in phase 1 (in gray). Note that all transitions go from a state in phase 0 to a state in phase 1 or conversely.

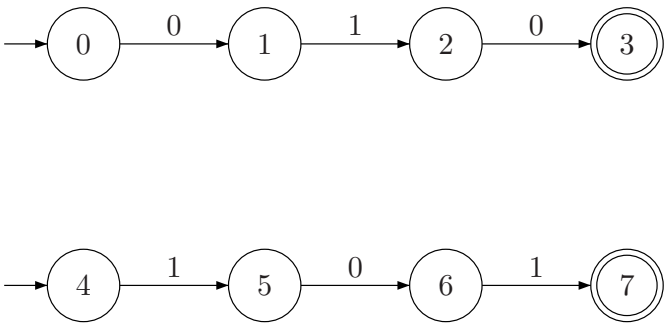

Figure 5.2: Example of the two input tries for the collection $\mathcal{F}$ defined by $\mathcal{F}^{(0)} = \{010\}, \mathcal{F}^{(1)} = \{101\}$ and $T = 2$. Final states are doubled circled.

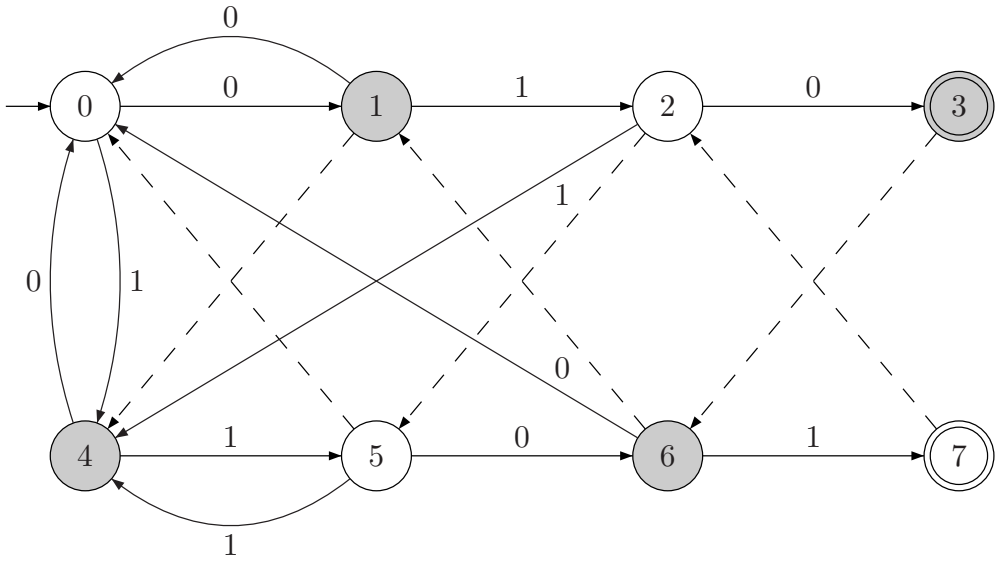

Figure 5.3: Presentation of the shift $\mathsf{X}_{\{\mathcal{F},T\}}$, where $\{\mathcal{F},T\}$ is defined by $\mathcal{F}^{(0)} = \{010\}$, $\mathcal{F}^{(1)} = \{101\}$ and $T = 2$.

Proposition 25 *Let $(\mathcal{T}_k)_{0\le k\le T-1}$ be the tries of a finite anti-factorial list $\mathcal{F}$ of periodic forbidden words for the period T. Algorithm* Periodic-automaton *builds the deterministic automaton $\mathcal{D}(\mathcal{F})$.*

Proof Since we assume that $\mathcal{F}$ is anti-factorial, we skip the lines 3 and 11 of the code of the algorithm. The automaton computed by the algorithm has a set of states Q which is the union of the set of states of the input tries. The automaton is deterministic, by construction.

Let $p = (u, k)$ be a state of Q_k. We prove by induction on the length of u that:

1. if $u \neq \varepsilon$, $f(p) = (v, k + r \bmod T)$, where v is the longest suffix $u[r\,..\,|u| - 1]$ of u, distinct from u, such that $(v, k + r \bmod T) \in Q$,

2. if a is a letter of A, and $\delta(p, a)$ is defined, $\delta(p, a) = (w, k + s \bmod T)$, where w is the longest suffix $(ua)[s\,..\,|ua| - 1]$ of ua such that $(w, k + s \bmod T) \in Q$.

Property 1 is trivially satisfied when u is a letter. Property 2 is trivially satisfied when u is the empty word.

Let u be a nonempty finite word, $p = (u, k) \in Q$. Hence $u = u'a$, where a is a letter, and we denote by p' the state (u', k) of Q_k.

By the inductive hypothesis of 1, since $|u'| < |u|$, either $u' = \varepsilon$ and Property 1 is satisfied for the state p, or $u' \neq \varepsilon$, and $f(p') = (v', k + r' \bmod T)$, where v' is the longest suffix $u'[r' \,..\, |u'| - 1]$ of u', distinct from u', such that $(v', k + r' \bmod T) \in Q$. By the inductive hypothesis of 2, since $|v'| < |u'| < |u|$, $\delta(f(p'), a) = (w', k + r' + s' \bmod T)$, where w' is the longest suffix $(v'a)[s' \,..\, |v'a| - 1]$ of $v'a$ such that $(w', k + r' + s' \bmod T) \in Q$. Then $f(p) = \delta(f(p'), a) = (w', k + r' + s' \bmod T)$. Thus, the block w' is a proper suffix of $u'a = u$. Let z be a proper suffix $(u'a)[t \,..\, |u'a| - 1]$ of $u'a$ such that $(z, k + t \bmod T) \in Q$. Then $z = z'a$ and z' is a suffix $u'[t' \,..\, |u'| - 1]$ of u' distinct from u', with $(z', k + t \bmod T) \in Q$. This implies that z' is a suffix of v', and that $z = z'a$ is a suffix of w'. Then Property 1 is satisfied for the state p.

We now consider two cases to prove property 2. Let a be a letter of the alphabet. Let us assume first that there is a transition $\delta_k(p, a)$. Then $\delta(p, a)$ is defined as $\delta_k(p, a) = (ua, k)$ and Property 2 is satisfied. Otherwise, $\delta(p, a)$ is defined as $\delta(f(p), a)$. Since Property 1 is satisfied for the state p, $f(p)$ is the state $(v, k + r \bmod T)$, where v is the longest suffix $u[r \,..\, |u| - 1]$ of u distinct from u such that $(v, k + r \bmod T) \in Q$. Hence $|v| < |u|$. Then, by inductive hypothesis of 2, $\delta(f(p), a) = (x, k + r + s \bmod T)$, where x is the longest suffix $(va)[s \,..\, |va| - 1]$ of va such that $(x, k + r + s \bmod T) \in Q$. Thus x is a suffix of ua. If y is a suffix $(ua)[t \,..\, |ua| - 1]$ of ua such that $(y, k + t \bmod T) \in Q$, then $y = y'a$ and y' is a suffix $u[t \,..\, |u| - 1]$ of u such that $(y', k + t \bmod T) \in Q$. Thus either $y' = u$ or y' is a suffix of v. The former case implies $t = 0$ and $\delta_k(p, a)$ exists, which is excluded. The latter case implies that $y = y'a$ is a suffix of va, and thus a suffix of x. It follows that $\delta(p, a) = (x, k + t \bmod T)$, where x is the longest suffix $(ua)[t \,..\, |ua| - 1]$ of ua such that $(x, k + t \bmod T) \in Q$. Since $\delta(p, a)$ is defined as $\delta(f(p), a)$, Property 2 is satisfied for the state p.

Therefore, assuming that $\mathcal{F}$ is anti-factorial, it remains to check that the instructions implement the definition of $\mathcal{D}(\mathcal{F})$. □

Corollary 26 *Let $(\mathcal{T}_k)_{0 \le k \le T-1}$ be the tries of a finite list $\mathcal{F}$ of periodic forbidden words for the period T. Algorithm* Periodic-automaton *builds a deterministic automaton accepting $\mathcal{B}^0(\mathcal{F}, T)$. It is also a presentation of $\mathsf{X}_{\{\mathcal{F},T\}}$ after removing the sink states.*

Proof Now $\mathcal{F}$ is no longer anti-factorial. We keep the lines 3 and 11 of the code of the algorithm. The algorithm detects in lines 3.a, 3.b, 3.c and 11.a, 11.b, 11.c, a violation of the anti-factorial property of the collection $\mathcal{F}$. Moreover, when $\mathcal{F}$ is not anti-factorial, it builds a new anti-factorial collection $\mathcal{F}'$ with $\mathcal{B}^0(\mathcal{F}, T) = \mathcal{B}^0(\mathcal{F}', T)$, by eliminating the words w in a set $\mathcal{F}^{(i)}$ that have strict factors $u \prec_j w$ in $\mathcal{F}^{(i+j \bmod T)}$. □

Proposition 27 *If transition functions are implemented by transition matrices, algorithm* PERIODIC–AUTOMATON *runs in time* $O((\sum_k |Q_k|) \times |A|)$ *on input* $\mathcal{T}_k = (Q_k, A, i_k, F_k, \delta_k)$, *for* $0 \leq k \leq T-1$.

Proof If transition functions δ_k and δ are implemented by transition matrices, access to or definition of $\delta_k(p, a)$ or $\delta(p, a)$ (p state, $a \in A$) are realized in constant amount of time. The result follows immediately. □

5.3 Presentation of Finite-Memory Systems with Unconstrained Positions

In this section, we use results from Sections 5.1 and 5.2 to derive an algorithm for constructing presentation of a finite-memory system with unconstrained positions from a finite list of forbidden words characterizing the constraint. This construction is an alternative to the construction given in [11].

Let S be a finite-memory system (or finite type shift), T a period and U a set of unconstrained positions. Let $\mathcal{F}$ be a set of forbidden blocks such that $S = \mathsf{X}_{\mathcal{F}}$. We know from Proposition 20 that the shift $S^{\sigma}_{U,T}$ is a periodic-finite-type system defined by the collection $\mathcal{G}$ as follows. For $k \in \{0, \ldots, T-1\}$,

- if $k \in U$, $\mathcal{G}^{(k)}$ is the $(U-k)$-closure of $\mathcal{F}$ plus the word 0,
- if $k \notin U$, $\mathcal{G}^{(k)}$ is the $(U-k)$-closure of $\mathcal{F}$.

We assume that the input data of our construction are the period T and the trie $\mathcal{T}$ accepting a prefix-free set of forbidden blocks $\mathcal{F}$ of S. The construction of a presentation of $S^{\sigma}_{U,T}$ is composed of two steps. In the first step, we build T tries $\mathcal{T}_k$, $0 \leq k \leq T-1$, accepting finite sets $\mathcal{G}^{(k)}$ such that $\mathsf{X}_{\{\mathcal{G},T\}} = S^{\sigma}_{U,T}$. In the

second step, we compute a presentation of $S^{\sigma}_{U,T}$ from the tries $\mathcal{T}_k$ accepting $\mathcal{G}^{(k)}$. Algorithm PERIODIC–AUTOMATON of Section 5.2 performs this second step.

We describe the first step for a two-letter alphabet $A = \{0, 1\}$, but the results carry over easily to larger alphabets. In order to reduce the complexity of the construction, we slightly change the sets $\mathcal{G}^{(k)}$ defined in Proposition 20 to avoid the generation of all U-flips of words in $\mathcal{F}$.

If L is a set of finite words, we call *prefix part of* L the subset $L - LA^+$ of L, where A^+ is the set of nonempty words over A. Hence, the prefix part of L is obtained from L by removing the words that have a strict prefix in L itself.

If $k \notin U$, we define $\mathcal{G}^{(k)}$ as the set of words obtained by setting all symbols at positions i, with $i + k \bmod T \in U$, to 1 in the words of $\mathcal{F}$, and by keeping the prefix part of this set. If $k \in U$, $\mathcal{G}^{(k)}$ is obtained by adding the word 0 to the above defined set, and by keeping again only its prefix part. It is easy to verify that $S^{\sigma}_{U,T} = \mathsf{X}_{\{\mathcal{G},T\}}$. The result is a collection of prefix-free sets but it may not be an anti-factorial collection.

Example 2 The RLL (2,7)-constraint is defined by the set of forbidden blocks $\mathcal{F} = \{11, 101, 00000000\}$. For $T = 3$ and $U = \{1\}$ we have to construct three sets $\mathcal{G}^{(k)}$, for $k = 0, 1$ and 2.

First, for every word of $\mathcal{F}$, we flip the symbols 0 in positions i such that $i + k \bmod T \in U$. Hence, for $k = 0$, we get the words $\{11, 111, 01001001\}$, for $k = 1$ the words $\{11, 101, 10010010\}$, and for $k = 2$ the words $\{11, 101, 00100100\}$. The sets $\mathcal{G}^{(k)}$ are obtained by taking the prefix part of the sets above, and by adding the word 0 to those $\mathcal{G}^{(k)}$ such that $k \bmod T \in U$. We obtain:

$$\begin{aligned} \mathcal{G}^{(0)} &= \{11, 01001001\}, \\ \mathcal{G}^{(1)} &= \{0, 11, 101, 10010010\}, \\ \mathcal{G}^{(2)} &= \{11, 101, 00100100\}. \end{aligned}$$

Example 3 The constrained system MTR(3) is defined by the set of forbidden

blocks $\mathcal{F} = \{1111\}$. For $T = 3$, $U = \{1\}$, we obtain

$$\begin{aligned} \mathcal{G}^{(0)} &= \{1111\}, \\ \mathcal{G}^{(1)} &= \{0, 1111\}, \\ \mathcal{G}^{(2)} &= \{1111\}. \end{aligned}$$

We will use the following operation on tries accepting prefix-free sets of words. If $\mathcal{T}$ and $\mathcal{T}'$ are two tries accepting prefix-free sets of words L and L' respectively, we denote by PREFIX-FREE-UNION$(\mathcal{T}, \mathcal{T}')$ a procedure that computes a trie accepting the prefix part of $L \cup L'$.

PREFIX-FREE-UNION (tries $\mathcal{T} = (Q, A, i, F, \delta)$,
$\mathcal{T}' = (Q', A, i', F', \delta')$)

1. **if** one of the tries is empty **return** the other trie
2. **if** one of the tries is reduced to a final state **return** this trie
3. let $l(\mathcal{T})$, (resp. $l(\mathcal{T}')$) be the sub-trie rooted at $\delta(i, 0)$ (respectively $\delta(i, 0)$)
4. let $r(\mathcal{T})$, (resp. $r(\mathcal{T}')$) be the sub-trie rooted at $\delta(i, 1)$ (respectively $\delta(i', 1)$)
5. (such a sub-trie is empty if the transition does not exist)
6. set $\delta(i, 0) =$ PREFIX-FREE-UNION$(l(\mathcal{T}), l(\mathcal{T}'))$
7. set $\delta(i, 1) =$ PREFIX-FREE-UNION$(r(\mathcal{T}), r(\mathcal{T}'))$
8. **return** the trie $\mathcal{T}$.

The construction of the tries $\mathcal{T}_k$ accepting $\mathcal{G}^{(k)}$ is then performed through Algorithm PERIODIC–TRIES below.

PERIODIC–TRIES(trie $\mathcal{T} = (Q, A, i, F, \delta)$, integer T)
1. make T copies $\mathcal{T}_k = (Q_k, A, i_k, F_k, \delta_k)$ of $\mathcal{T}$
2. **for** each $k \in \{0, \ldots, T-1\}$
3. **for** each state p of $\mathcal{T}_k$ at distance d from i_k
4. (for instance in a bottom-up order)
5. **if** $(k + d \bmod T \in U)$ **and** $p \notin F_k$
6. let $l(\mathcal{T}_k)$, (resp. $r(\mathcal{T}_k)$) be the sub-trie rooted by $\delta_k(p, 0)$ (resp. $\delta_k(p, 1)$), eventually empty if the transition does not exist
7. remove $\delta_k(p, 0)$, if it exists
8. set $\delta_k(p, 1) =$ PREFIX-FREE-UNION$(l(\mathcal{T}_k), r(\mathcal{T}_k))$
9. **if** $k \in U$, set $\delta_k(i_k, 0) =$ new sink state.
10. **return** the tries $\mathcal{T}_k$

Proposition 28 *Algorithm* PERIODIC–TRIES *runs in time* $O(|Q| \log |Q| \times T \times |A|)$ *on the input trie* $\mathcal{T} = (Q, A, i, F, \delta)$ *and the input period* T.

Proof The procedure PREFIX-FREE-UNION$(\mathcal{T} = (Q, A, i, F, \delta),\ \mathcal{T}' = (Q', A,\ i', F', \delta'))$ runs in time $O(\min(|Q|, |Q'|))$. If p is a state of the trie $\mathcal{T}$, we denote by $l(p)$ the (eventually empty) left sub-trie of p, *i.e.*the sub-trie rooted by $\delta(p, 0)$. Similarly, we denote by $r(p)$ the (eventually empty) right sub-trie of p. Thus Algorithm PERIODIC–TRIES$(\mathcal{T} = (Q, A, i, F, \delta))$ runs in time $O(T \times |A| \times \sum_{p \in Q} \min(|l(p)|, |r(p)|)$. We now evaluate the sum $s = \sum_{p \in Q} \min(|l(p)|, |r(p)|)$. We say that a sub-trie of a state p is small if it has the smallest size among the two sub-tries children of p. Then each state belongs to at most $\log_2 |Q|$ small sub-tries. It follows that $s \leq |Q| \log_2 |Q|$. □

We mention that other simplifications may be added in the procedure PERIODIC–TRIES. For instance, if we are interested in computing bi-infinite words or right-infinite words, any two words $u0$ and $u1$ accepted by a trie may be removed and replaced by u. Indeed, in the case of infinite words, if $u0$ and $u1$ are forbidden in a position i, then u is also forbidden. Nevertheless, this simplification does not reduce the overall asymptotic complexity of the process.

Note that, if one considers $|A|$ and T as constants, the $|Q| \log |Q|$ time-complexity obtained in Proposition 28 becomes linear in $|Q|$ when the input trie $\mathcal{T}$ is linear,

*i.e.*accepts a single word. Note also that each output periodic trie has a size not larger than the size of the input trie.

The second step of the construction uses Algorithm PERIODIC–AUTOMATON of Section 5.2 for computing a presentation of $S^{\sigma}_{U,T}$ from tries $\mathcal{T}_k$ accepting sets $\mathcal{G}^{(k)}$ such that $\mathsf{X}_{\{\mathcal{G},T\}} = S^{\sigma}_{U,T}$. The output is an automaton $\mathcal{A} = (Q, A, i, Q \setminus F, \delta)$ accepting $S_{U,T}$. If the sink states (*i.e.*states of F) are removed, one gets a presentation of the shift $S^{\sigma}_{U,T}$.

We now evaluate the overall time-complexity of the process and compare it with the time-complexity of the construction given in [11]. Algorithm PERIODIC–AUTOMATON runs on tries $\mathcal{T}_k$ in time $O((\sum_k |\mathcal{T}_k|) \times |A|)$. Since $|\mathcal{T}_k| \leq |\mathcal{T}|$, it is $O(T \times |\mathcal{T}| \times |A|)$. Then the overall time-complexity for the input data T and a trie $\mathcal{T}$ accepting a prefix-free set of forbidden blocks of S, is $O(T \times |A| \times |\mathcal{T}| \log |\mathcal{T}|)$. It becomes linear for linear tries. The evaluation of the space complexity is similar and gives $O(T \times |A| \times |\mathcal{T}|)$.

The construction of [11] enables the computation of a presentation of $S^{\sigma}_{U,T}$ from a presentation of finite-state constrained system S in an exponential amount of time in general, and in quadratic time with a particular condition, called the gap condition (see [11, pp. 875]). Although our algorithm is polynomial and that given in [11] is exponential, the two algorithms compute similar presentations. But the input data are different. In particular, the minimal set of forbidden words of a finite-memory system can be computed in quadratic-time (see [4]) from a deterministic presentation of the system when this presentation has a unique initial state. If the system is given by a deterministic presentation where all states are initial, with Q states and memory M, it can take in the worst case $O(|A|^M)$ amount of time to compute a deterministic presentation that has a unique initial state. Thus the complexities of the two algorithms cannot be compared directly and one can choose one or the other depending on the way the constraint is defined.

Some constraints may be naturally defined by a list of forbidden blocks. For instance, an MTR constraint is defined by a single forbidden block. The RLL (d, k)-constraint is defined by d forbidden blocks of length at most $d + 1$ and one block of length $k + 1$. With $(d, k) = (2, 7)$ one gets the forbidden blocks $\{11, 101, 00000000\}$. A trie accepting a finite set is built in time linear in the sum of the lengths of the words of the set. In the particular case of the set of forbidden blocks of the (d, k)-

constraint, the trie is built in time linear in $d+k$, *i.e.*since $d \leq k$, in time $O(k)$ from the inputs d and k. Moreover the trie has a size that is also $O(k)$. Indeed, the trie has the particular linear structure described in Figure 5.4.

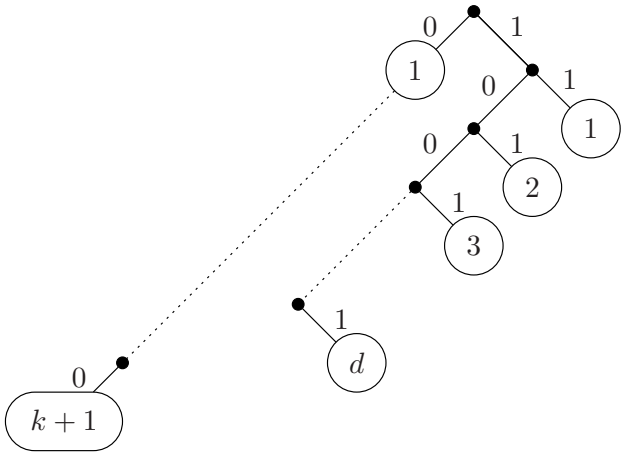

Figure 5.4: The trie of the RLL (d, k)-constraint.

It follows that Algorithm Periodic–automaton runs in time $O(k)$ on this input trie. Indeed, in the analysis of the complexity in the proof of Proposition 28, $s = O(|Q|) = O(k)$. Thus our algorithm works linearly on the MTR constraints, and on the RLL constraints. An efficient algorithm for the MTR constraints is also given in [11]. Figure 5.6 displays an example for the constraint MTR(3). The presentation can be minimized with standard methods [34]. It leads to the minimal presentation displayed in Figure 5.5.

A condition similar to the gap condition of [11, pp. 875] can be stated as follows. We assume that there is at most one unconstrained position in $\{0, \ldots, M-1\}$, where M is the maximal length of a minimal forbidden word of the system. If this condition is satisfied, the complexity of our algorithm becomes linear, *i.e.*, $O(T \times |A| \times |\mathcal{T}|)$.

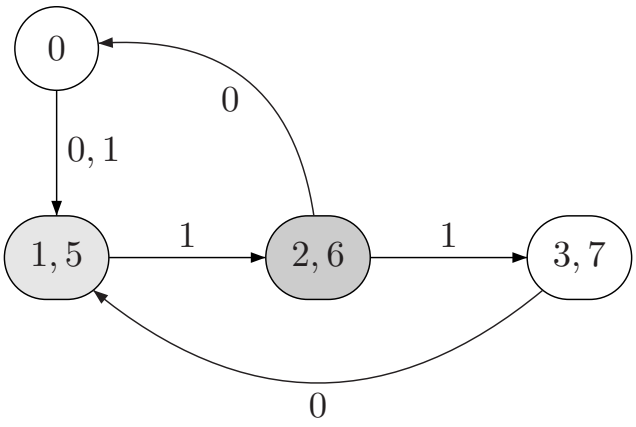

Figure 5.5: The Shannon cover of $S^{\sigma}_{U,T}$ for $S = \mathrm{MTR}(3)$, $T = 3$, $U = \{1\}$. It is the minimal presentation of that of Figure 5.6.

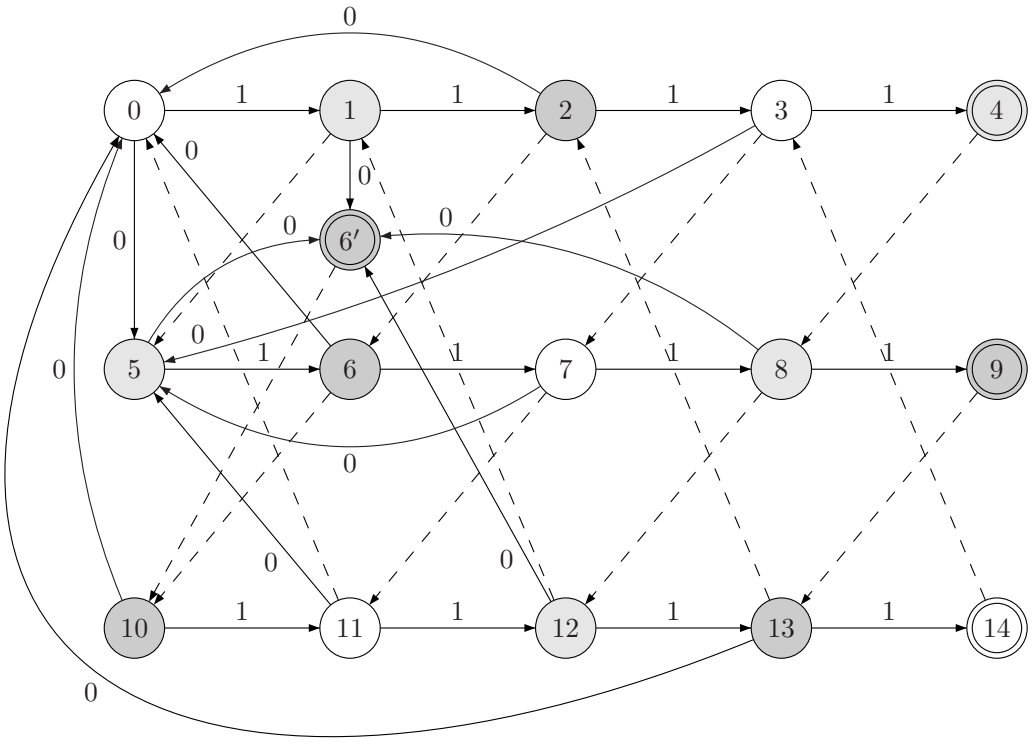

Figure 5.6: A presentation of $S^{\sigma}_{U,T}$ for $S = \mathrm{MTR}(3)$, $T = 3, U = \{1\}$. It is obtained by Algorithm Periodic–automaton on the input tries accepting the sets $\mathcal{G}^{(0)} = \{1111\}$, $\mathcal{G}^{(1)} = \{0, 1111\}$, $\mathcal{G}^{(2)} = \{1111\}$. States in phase 0, 1, and 2 are colored in white, light gray, and gray respectively.

Chapter 6

Applications to the Word Assembly Problem

In this chapter we show an application of the minimal forbidden words to the Word Assembly Problem, that is to reconstruct a finite word w over a finite alphabet of constant size A starting from a set of factors of w.

Introduction

The problem of the reconstruction of a word from a set of its factors arises from several fields, as biology or cryptography. An example is the mathematical formalization of the problem of a genomic sequence reconstruction. It is known, for instance, that it is not possible to read the entire sequence of basis of a DNA molecule, but only factors of small length. The reconstruction of the original DNA sequence is complicated by other constraints, as read-errors or unknown orientation of the factors. This problem is known as the Fragment Assembly Problem.

A theoretical simplification of the problem consists in considering a finite word as target of the reconstruction and a set of its factors as input of the problem. In general, in order to reconstruct in a unique way a word from its fragments, one has to introduce further hypothesis. We will deal with this theoretical problem, and we will call it the Word Assembly Problem.

Carpi et al. [17, 23], showed that a finite word can be uniquely reconstructed starting from a particular set of its factors. The factors needed for the reconstruction

are called maximal boxes of the word.

Mignosi et al. in [43, 44, 46] introduced a hypothesis of non-repetitiveness and gave two different algorithms for the sequence assembly that work in linear time. Such algorithms avoid one of the most common step used in solving fragment assembly problem that is the overlap phase in which every fragment is compared to each other, giving rise to a quadratic number of comparisons.

One of these algorithms is based on the notion of minimal forbidden word. Given a word w over a finite alphabet A, a minimal forbidden word for w is a finite word v that is not a factor of w but such that every proper factor of v is a factor of w. The length of the longest minimal forbidden word for w is noted by $m(w)$ and it is involved in the previously mentioned hypothesis of non-repetitiveness. Starting from a set I of factors of w containing all the factors of w having length $m(w)$, it was described an algorithm able to retrieve w from the set I under the condition that the value of $m(w)$ is known. The authors showed in [43] that such a hypothesis on the elements of I is statistically reasonable. Actually, they proved for a word w randomly generated by a memoryless source with identical symbol probabilities, that the probability that $m(w)$ is $O(\log(|w|))$ converges to 1 as $|w|$ leads to infinity, so it is very likely that any factor of w of length $m(w)$ is covered by at least one element of I.

In this chapter we introduce the definition of I-compatibility for a finite word. Given an arbitrary finite set of finite words I we say that a finite word w is I-compatible if all the words in I are factors of w and if I contains all the factors of w having length $m(w)$. By using this definition algorithms in [43, 44, 46] work under the assumptions that there exists a word w that is I-compatible and that $m(w)$ is known.

Here we improve previous results by removing the *a-priori* knowledge of the value $m(w)$, *i.e.* we show that the only existence of a I-compatible word is a sufficient condition for its unique reconstruction. Such a reconstruction can be done in linear time in the size of the set I.

As another improvement, we show that it is possible to decide in linear time whether there exists a word w that is I-compatible.

At the end of the chapter we study the well known Shortest Superstring Problem, that consists, given a finite set of finite words I, in finding the shortest word that

contains all the words in I as factors. This problem is NP-hard in general.

A greedy algorithm, that is called here the Greedy Algorithm, running in quadratic time on the size of I, is used in order to find in a fast way words that are "quite close" to the shortest superstring. It repeatedly merges the two fragments of the set I having maximum overlap until only one word remains. If there exists more than one couple of fragments having maximum overlap it randomly chooses one of these couples for the merging.

The ratio between the length of the shortest superstring and the length of the Greedy superstring is the object of a deep study. More details about this discussion can be found in [58].

We show that, under the hypothesis of the existence of a I-compatible word, the Greedy algorithm outputs the shortest solution, *i.e.* it is deterministic. Moreover our algorithm retrieves the same solution. This shows that our Word Assembly procedure outputs in fact the shortest superstring for the set I and thus the shortest superstring can be reconstructed in linear time in this case.

In Section 5.1 we recall all the needed background and we introduce some new definitions.

In Section 6.1 we state the Word Assembly Problem and we recall the techniques used in [44].

In Section 6.2 we show that for a given set of finite words I there exists at most one I-compatible word.

In Section 6.3 we give a method that allows to retrieve the set of the minimal forbidden words for the target word w starting from the input set I.

In Section 6.4 we give an algorithm for the reconstruction of the word w from the set I under the hypothesis that there exists a I-compatible word and a linear algorithm that decides whether there exists a word w that is I-compatible.

Finally, in Section 6.5 we compare the Word Assembly Problem with the Shortest Superstring Problem, and we compare our algorithms with Greedy ones used for solving this second problem. We show that under our hypothesis of existence of a I-compatible word, these two problems have the same solution, and so we can solve the problem in a deterministic way and in linear time.

6.1 The Word Assembly Problem

Let $I = \{i_1, \ldots, i_n\}$ be a set of fragments, *i.e.* a finite set of finite words over a given finite alphabet A.

We say that a finite word w is I-compatible if

1. $I \subseteq \mathrm{Fact}(w)$,
2. for every $u \in \mathrm{Fact}(w)$ such that $|u| \leq m(w)$ there exists at least a fragment $i_j \in I$ such that $u \in \mathrm{Fact}(i_j)$.

If I is a set of factors of a finite word w, we have the following definition:

Definition 6.1 *A set of factors I of a finite word w is a k-cover for w, for $0 \leq k \leq |w|$, if every factor of w of length k is a sub-factor of at least one word in I. The covering index of I, denoted $C(I)$, is the largest value of k such that I is a k-cover of w.*

In general any set of fragments I can be a set of factors for many different words. It will have, then, a cover index for any such word.

The point 2. of the definition of I-compatibility is equivalent to the fact that the condition $C(I) \geq m(w)$ is verified for the word w (see [44]).

The *Word Assembly Problem* is here formulated as follows:

Word Assembly Problem. Given a finite set of fragments I, decide whether there exists an I-compatible word w, and, if it exists, reconstruct it.

To end this section, we briefly recall the construction of the Assembly algorithm given in [44]. The inputs of the algorithm are the set I and the value $m(w)$, so the algorithm works under the assumption that the value $m(w)$ is known.

Starting from the set of fragments $I = \{i_1, \ldots, i_n\}$ over the finite alphabet A, the first step is the construction of the concatenation word w_1 over the alphabet $A \cup \{\$\}$, that is the concatenation of all the strings in I, interspersed with the symbol \$, that is a special symbol not belonging to A, *i.e.* $w_1 = \$i_1\$i_2\$ \ldots \$i_n\$$.

The second goal of the Assembly algorithm consists in the construction of the trie of the minimal forbidden words for w_1 having length smaller than or equal to

$m(w)$ and not containing the symbol \$. Such a construction is consequence of the following result (see [44] Proposition 5.3):

Proposition 29 *Let w be a word over a fixed alphabet A and let I a set of substring of w such that*

$$m(w) \leq C(I).$$

Then the set of minimal forbidden words for the word w is exactly the set of all the minimal forbidden words for w_1 that do not contain the symbol \$ and that have length smaller than or equal to $m(w)$, i.e.

$$\mathcal{MF}(w) = \mathcal{MF}(w_1) \cap A^{\leq m(w)}.$$

So we can retrieve the trie of the minimal forbidden words for w starting from the factor automaton of w_1 (coming with its suffix function h) and the value $m(w)$. This is operation is performed in linear time $O(||I||)$ by the CREATE-TRIE algorithm, that computes the trie of the minimal forbidden words for w_1 and keeps those having length smaller or equal to $m(w)$ and not containing the symbol \$.

CREATE-TRIE (factor automaton $\mathcal{F}(w_1) = (Q, A \cup \{\$\}, i, T, \delta)$,
suffix function h, value $m(w)$)
1. **for** each state $p \in Q$ in breadth-first search from i **and** each $a \in A$
2. **if** $\delta(p, a)$ undefined **and** ($p = i$ **or** $\delta(h(p), a)$ defined)
3. $\delta'(p, a) \leftarrow$ new sink;
4. **else if** $\delta(p, a) = q$ **and** q is distant from i more than p
5. $\delta'(p, a) \leftarrow q$;
6. In a depth-first search with respect to δ' prune all branches of the trie $\mathcal{T}(w)$ not ending in a state that is sink and has depth smaller than or equal to $m(w)$;
7. **return** $\mathcal{T}(w) = (Q', A, i', \{\text{sinks}\}, \delta')$;

Finally, a linear algorithm, w-RECONSTRUCTION, which reconstructs the word w from the set $\mathcal{MF}(w)$, is used. This algorithm calls a procedure BUILDWORD which finds the longest path in a DAG (Directed Acyclic Graph) by using a topological sort.

The overall ASSEMBLY algorithm is thus

w-RECONSTRUCTION (Trie $\mathcal{T}(w)$ representing the set $\mathcal{MF}(w)$)
1. $\mathcal{A}(w) \leftarrow$ L-AUTOMATON($\mathcal{T}(w)$);
2. Let $\mathcal{F}(w)$ be the automaton obtained by removing sink states of $\mathcal{A}(w)$;
3. $w \leftarrow$ BUILDWORD($\mathcal{F}(w)$);
4. **return** w;

ASSEMBLY (set of fragments $I = \{i_1, i_2, \ldots, i_n\}$, value $m(w)$)
1. $w_1 \leftarrow \$i_1\$i_2\$ \ldots \$i_n\$$;
2. $\mathcal{F}(w_1) = (Q, A \cup \{\$\}, i, T, \delta) \leftarrow$FACTOR-AUTOMATON($w_1$);
3. $\mathcal{T}(w) = (Q', A, i, \{\text{sinks}\}, \delta') \leftarrow$CREATE-TRIE ($\mathcal{F}(w_1), h$);
4. $w \leftarrow w$-RECONSTRUCTION ($\mathcal{T}(w)$);
5. **return** w;

This algorithm runs in linear time $O(||I||)$, where $||I||$ denotes the sum of the lengths of all the strings in I.

6.2 Uniqueness of the Reconstruction

The main result of this section is the following theorem:

Theorem 30 *Given a finite alphabet A, and a set I of fragments over A, if there exists a word w that is I-compatible, then w is unique.*

We start with the following definition:

Definition 6.2 *Given two finite sets of finite words M and M', we say that M is strongly included in M' if $M \subsetneqq M'$ and moreover there exists at least one word $v \in M'$ such that $|v| > max\ \{|u| : u \in M\}$. If M is strongly included in M' we note $M \precneqq M'$.*

In the rest of this section we suppose that there exists a word w that is I-compatible. Let w' be another I-compatible word.

By the construction of the word w_1, concatenation of the fragments of I, and by the Proposition 29, we have that

- $\mathcal{MF}(w_1) \cap A^{\leq m(w)} = \mathcal{MF}(w)$

- $\mathcal{MF}(w_1) \cap A^{\leq m(w')} = \mathcal{MF}(w')$

If $m(w) = m(w')$, then $\mathcal{MF}(w) = \mathcal{MF}(w')$, and so $\mathrm{Fact}(w) = \mathrm{Fact}(w')$. Therefore $w = w'$.

If instead $m(w) \neq m(w')$, we have a situation in which either $\mathcal{MF}(w) \precneqq \mathcal{MF}(w')$ or $\mathcal{MF}(w') \precneqq \mathcal{MF}(w)$. The next Corollaries 33 and 34 show that this situation is impossible.

We start with a Lemma whose proof is straightforward and which will be used in the proof of the next Theorem 32.

Lemma 31 *Let M be a finite anti-factorial set of words over a finite alphabet A, and let l be the length of the longest word in M. If a finite word z over A has the property that there exists an $l' \geq l$ such that every factor of z of length l' does not contain any word of M, then z does not contain any word of M,* i.e. *$z \in L(M)$.*

The following theorem shows that the set of the minimal forbidden words for a finite word has a very rigid structure:

Theorem 32 *Let w be a finite nonempty word over a finite alphabet A, and $X = \{v \in \mathcal{MF}(w) : |v| = m(w)\}$ the set of the longest minimal forbidden words for w. Then the L-automaton that recognizes the language $L = L(\mathcal{MF}(w) \setminus X)$ of the finite words avoiding the anti-factorial language $\mathcal{MF}(w) \setminus X$ has some loops, so it cannot be the factor automaton of a single finite word.*

Proof We show that there always exists a non-empty factor of w that can be iterated an arbitrary number of times without violating the constraints of the language $L(\mathcal{MF}(w) \setminus X)$. Thus this language is infinite and then the L-automaton that recognizes it must contain some loops.

Let $v = aub \in X$, with a, b symbols in a finite alphabet A (we can have $a = b$). By the definition of minimal forbidden word we know that au and ub must appear as factors of w, so we have three cases:

Case 1. The factors au and ub appear at the same position. In this situation $au = ub$, so the only possibility is $a = b$ and $au = ub = a^n$, for a positive integer n, hence $aub = a^{n+1}$. We can then write $w = s_1 a^n s_2$, with $s_1, s_2 \in A^*$.

In this case we can iterate the factor a^n an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1(a^n)^+s_2 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1(a^n)^+s_2$ are exactly the factors of length $m(w)$ of w plus the factor $a^{n+1} = v$, that belongs to X, every factor of length $m(w)$ of $s_1(a^n)^+s_2$ does not contain any factor of $\mathcal{MF}(w) \setminus X$, so it belongs to $L(\mathcal{MF}(w) \setminus X)$.

Case 2. The factor ub appears before the factor au.

We can consider several configurations:

i) If ub and au do not overlap, we can then write $w = s_1ubs_2aus_3$, with $s_1, s_2, s_3 \in A^*$. In this case we can iterate the factor bs_2au an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1u(bs_2au)^+s_3 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1u(bs_2au)^+s_3$ are exactly the factors of length $m(w)$ of w plus the factor $aub = v$, that belongs to X, none of the factors of length $m(w)$ of $s_1u(bs_2au)^+s_3$ contains any factors of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

ii) If ub and au overlap over a single letter $a = b$, we can write $w = s_1uaus_2$, with $s_1, s_2 \in A^*$. In this case we can iterate the factor ua an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1(ua)^+us_2 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1(ua)^+us_2$ are exactly the factors of length $m(w)$ of w plus the factor $aua = v$, that belongs to X, none of the factors of length $m(w)$ of $s_1(au)^+us_2$ contains any factor of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

iii) If ub and au overlap over a two-letters factor (in particular u is not empty), we have that u starts with the letter b and ends with the letter a, *i.e.* $u = b\overline{u}a$, for a $\overline{u} \in A^*$.

Note that we must have $|u| > 1$, because if $|u| = 1$, ub and au appear at the same position, against the hypothesis that ub appears before au.

So we can write $w = s_1b\overline{u}ab\overline{u}as_2$, with $s_1, s_2 \in A^*$. In this case we can iterate the factor $u = b\overline{u}a$ an arbitrary number of times without violating the constraints

of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1(b\overline{u}a)^+s_2 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1(b\overline{u}a)^+s_2$ are exactly the factors of length $m(w)$ of w plus the factor $ab\overline{u}ab = v$, that belongs to X, none of the factors of length $m(w)$ of $s_1(b\overline{u}a)^+s_2$ contains any factor of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

iv) If ub and au overlap over a sub-factor u' of u, such that $|u'| > 0$ (in particular u is not empty), then u' is a border of u. Note that $|u| > 1$, because if not ub and au should appear at the same position.

So we can write $w = s_1w's_2$, where $s_1, s_2 \in A^*$, and w' is the non-empty factor of w that starts with ub and ends with au. In particular w' can be written as $w' = zu = uv$, with $|z| = |v| = p$, for a positive integer $p < |u|$.

By the Proposition 1, p is a period of the word w'. Moreover, there exists two unique words $m \in A^*$ and $r \in A^+$, and an integer $k > 0$, such that $w' = (mr)^km$ and $|mr| = p$.

So we have $w = s_1w's_2 = s_1(mr)^kms_2$.

Since u is a border of w' longer than the half of the length of w' (because $|z| = |v| < |u|$), we can suppose $k > 1$. Actually, if $k = 1$, since $p = |mr| < |u|$, we should have

$$|u| + p = |w'| = |mr| + |m| < 2|mr| = 2p < |u| + p,$$

that is a contradiction.

Note that since $w' = zu = (mr)^km$ and $|z| = |mr| = p$, we have $z = mr$, hence $u = (mr)^{k-1}m$. Let us note $r = ctc'$, with $t \in A^*$ and $c, c' \in A \cup \{\varepsilon\}$, not both empty (if $|r| = 1$ we assume $t = \varepsilon$ and $c = c'$).

Thus we can iterate the factor mr an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.*
$s_1(mr)^k(mr)^+ms_2 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1(mr)^k(mr)^+ms_2$ are exactly the factors of length $m(w)$ of w plus the factor $c'(mr)^{k-1}mc = c'uc$ and since $c'u$ and uc are factors of w, we have that if $c'uc$ is a forbidden factor for w, then it is a minimal one, and so it belongs to X. Hence none of the factors of length $m(w)$ of $s_1u(au)^+s_2$ contains any factors of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

Case 3. The factor ub appears after the factor au.

We can consider several configurations.

i) If the factors au and ub do not overlap, we can then write $w = s_1auys_2xubs_3$, with $s_1, s_2, s_3 \in A^*$ and $x, y \in A \cup \{\varepsilon\}$. Suppose that $|ys_2x| > 0$ (in particular if $|ys_2x| = 1$ we assume $s_2 = \varepsilon$ and $y = x$). In this case we can iterate the factor ys_2xu an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1au(ys_2xu)^+bs_3 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1au(ys_2xu)^+bs_3$ are exactly the factors of length $m(w)$ of w plus the factor xuy and since xu and uy are factors of w, we have that if xuy is a forbidden factor for w, then it is a minimal one, and so it belongs to X. Hence, none of the factors of length $m(w)$ of $s_1u(bs_2au)^+s_3$ contains any factor of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

If instead $|ys_2x| = 0$, then $w = s_1auubs_3$. Observe that in this case $|u| > 0$, because ab cannot be at the same time a forbidden factor and a factor of w. So let us note $u = dzd'$, with $z \in A^*$ and $d, d' \in A \cup \{\varepsilon\}$ but not both empty (if $|u| = 1$ we assume $z = \varepsilon$ and $d = d'$). In this case we can iterate the factor u an arbitrary number of times without violating the constraints of the language $L = L(\mathcal{MF}(w) \setminus X)$, *i.e.* $s_1au(u)^+bs_3 \subseteq L(\mathcal{MF}(w) \setminus X)$.

Actually, by the Lemma 31, with $M = \mathcal{MF}(w) \setminus X$, since the factors of length $m(w)$ of the language $s_1au(u)^+bs_3$ are exactly the factors of length $m(w)$ of w plus the factor $d'dzd'd = d'ud$ and since $d'u$ and ud are factors of w, we have that if $d'ud$ is a forbidden factor for w, then it is a minimal one, and so it belongs to X. Hence, none of the factors of length $m(w)$ of $s_1au(u)^+bs_3$ contains any factor of $\mathcal{MF}(w) \setminus X$, so all these factors belong to $L(\mathcal{MF}(w) \setminus X)$.

ii) If au and ub overlap over a sub-factor u' of u, then u' is a border of u (note that $|u'| < |u|$ because aub is a forbidden word for w), and the proof is the same as that of Case 2, iv. □

Example. Let $w = abababa$ over the alphabet $A = \{a, b\}$. We have $\mathcal{MF}(w) = \{aa, bb, bababab\}$, so $X = \{bababab\}$ and than $\mathcal{MF}(w) \setminus X = \{aa, bb\}$. The minimal forbidden word $v = bababab$ has a strict maximal prefix and a strict maximal suffix that overlap as factors of w. We are in the situation described in Case 2, iv.

We observe that the words $(ab)^n$ belongs to $L(\mathcal{MF}(w) \setminus X)$ for every positive integer n.

Corollary 33 *Let w be a finite nonempty word over a finite alphabet A, and $M = \mathcal{MF}(w)$ the set of its minimal forbidden words. Then for every anti-factorial finite set of finite words M' such that $M' \precneqq M$, the L-automaton that recognizes the language $L(M')$ has some loops, so it cannot be the factor automaton of a single finite word.*

Proof If $M' \precneqq M$, in particular we have $M' \subset M \setminus X$, then every finite word avoiding the set $M \setminus X$ must avoid *a fortiori* the set M', so $L(M \setminus X) \subseteq L(M')$. Thus, since by the Theorem 32 $L(M \setminus X)$ is infinite, then $L(M')$ is infinite too, and so the L-automaton of M' must contain some loops. □

Corollary 34 *Let $M = \mathcal{MF}(w)$ be the (anti-factorial) set of the minimal forbidden words for a finite nonempty word w over a finite alphabet A. Then for every anti-factorial finite set of finite words M' such that $M \precneqq M'$, the L-automaton that recognizes the language $L(M')$ cannot be the factor automaton of a single finite word.*

Proof If the L-automaton of $L(M')$ were the factor automaton of a single finite word, it should exist a finite word w' such that $M' = \mathcal{MF}(w')$. But in this case, by the Theorem 32, the L-automaton that recognizes the language $L(M)$ could not be the factor automaton of a single finite word, against the hypothesis. □

6.3 Finding the Minimal Forbidden Words for w

We now suppose to have a set of fragments I and that there exists a (unique) I-compatible word w. So we know that there exists a finite word w such that the fragments of I are factors of w and the condition $C(I) \geq m(w)$ is verified, and we want to reconstruct the word w.

In particular, we are interested in finding the minimal forbidden words for the word w. This will allow us to reconstruct the word w using the w-RECONSTRUCTION procedure.

In this section we find a way to deduce the set $\mathcal{MF}(w)$ from the set $\mathcal{MF}(w_1)$ without the explicit knowledge of the value $m(w)$.

The following two Propositions are given in [44] without proof (Remarks 3.3 and 5.4).

Proposition 35 *Let w be a finite word over a finite alphabet A. Then*

$$r(w) = m(w) - 2$$

where $r(w)$ is the repetition index of w and $m(w)$ is the length of the longest minimal forbidden word for w.

Proof Let $m = avb$ be a minimal forbidden word for w of maximal length $|m| = m(w)$. So av and vb are factors of w.

If av is a suffix of w, then, since vb is a factor of w, we have that v is a factor of w appearing at least twice (maybe overlapping).

If av is not a suffix of w then avx is a factor of w for a letter $x \in A$ different from b. Since vb is a factor of w, v is a factor of w appearing at least twice (maybe overlapping).

So there exists a factor v of length $m(w) - 2$ of the word w appearing at least twice, and thus $r(w) \geq m(w) - 2$.

Conversely, if v is a factor of the word w having length $r(w)$ (so it appears at least twice in w, maybe overlapping), then we want to prove that $|v| \leq m(w) - 2$. To do this, it is sufficient to show that there exists a minimal forbidden word for w of the form avb, with $a, b \in A$. We have several cases:

Case 1. The factor v is not a prefix nor a suffix of w. So the word w contains at least two factors containing v as central factor, say x_1vy_1, with $x_1, y_1 \in A$, and x_2vy_2, with $x_2, y_2 \in A$.

In this case we have that $x_1 \neq x_2$ and $y_1 \neq y_2$. If not there should exist a factor of w longer than v appearing at least twice, against the hypothesis that $|v| = r(w)$. Moreover, for the same reason, x_1vy_2 cannot be a factor of w. So, since x_1v and vy_2 are factors of w, x_1vy_2 is a minimal forbidden word for w.

Case 2. If v appears as prefix and as suffix of w, let b be the letter following the prefix v and a the letter preceding the suffix v (note that such a and b must

exist since $|v| < |w|$). Then av and vb are factors of w. Moreover, they appear only once as factors of w, if not there should exist a factor longer than v appearing at least twice as factor of w. So avb cannot be a factor of w, and then it is a minimal forbidden word for w.

Case 3. If v appears as prefix but not as suffix of w, let a be the letter following the prefix v. The second time that v appears in w it must be followed by a letter b different from a, if not there should exist a factor longer than v appearing at least twice in w. Let x be the letter preceding the factor vb. So xv is a factor of w and it appears only once in w, since it is longer than v. Thus xva is a minimal forbidden word for w.

Case 4. If v appears as suffix but not as prefix of w, the proof is analogous to that of the Case 3. □

We focus now on the structure of the set $(\mathcal{MF}(w_1) \cap A^*) \setminus \mathcal{MF}(w)$, *i.e.* of the minimal forbidden words for w_1 not containing the symbol \$ that are not minimal forbidden words for $\mathcal{MF}(w)$.

Remark. Since by the Proposition 29 we have that $\mathcal{MF}(w_1) \cap A^{\leq m(w)} = \mathcal{MF}(w)$, every word belonging to $(\mathcal{MF}(w_1) \cap A^*) \setminus \mathcal{MF}(w)$ has a length greater than $m(w)$.

Let S be the set of the minimal forbidden words for w_1 not containing the symbol \$, of the form $v = aub$, such that

- the words $au\$$ and $\$ub$ are factors of w_1,
- the words aux and xub are not factors of w_1, for every $x \in A$.

We will show that the knowledge of this set allows us to retrieve the set $\mathcal{MF}(w)$ from the set $\mathcal{MF}(w_1)$.

Proposition 36 *If w is a I-compatible word, then the following equality holds:*

$$\mathcal{MF}(w) \cup S = \mathcal{MF}(w_1) \cap A^*.$$

Proof The inclusion $S \subseteq \mathcal{MF}(w_1) \cap A^*$ follows by the definition of S. In order to prove that $\mathcal{MF}(w) \cup S \subseteq \mathcal{MF}(w_1) \cap A^*$, it remains to show that $\mathcal{MF}(w) \subseteq$

$\mathcal{MF}(w_1)$. Actually, let $v = aub$ be a minimal forbidden word for w. By definition, it follows that both au and ub are factors of w and hence, by the second condition of I-compatibility, they appear in some fragment of I. Therefore they appear in w_1. If $v = aub$ was in w_1, then it should appear in some fragments of I and so, by the first condition of I-compatibility, it should be a factor of w.

Let us consider now a word $v = aub$ in $\mathcal{MF}(w_1) \cap A^*$. We show that if it does not belong to $\mathcal{MF}(w)$, then it has to belong to S. Actually, aub is not a factor of w_1, while au and ub are, and so, by the construction of the word w_1, au and ub are factors of w too. If moreover $aub \notin \mathcal{MF}(w)$, then we know by the previous Remark that $|aub| > m(w)$, and so $|au| = |ub| > m(w) - 1$. Since aub is not a minimal forbidden word for w, but au and ub are factors of w, we conclude that aub must be a factor of w. By the Proposition 35, we know that au and ub appear exactly once as factors of w. So, in the word w, au can only be followed by b and ub can only be preceded by a. Thus, in the word w_1, the factor au can only be followed by \$ and ub can only be preceded by \$, aub being a minimal forbidden word for w_1. □

The following two propositions show that the sets $\mathcal{MF}(w)$ and S are "almost" disjoint.

Proposition 37 *Let w be a I-compatible word, and $B(w)$ be the set of the bad minimal forbidden words for w. Then the following equality holds:*

$$\mathcal{MF}(w) \cap S = B(w).$$

Proof The set $B(w)$ is included in $\mathcal{MF}(w)$ by definition. Let $v = aub$ be a bad minimal forbidden word for w. Then, by the second condition of I-compatibility, it is a minimal forbidden word for w_1. Moreover, since au is a suffix of w and it does not appear elsewhere in w, and since $|au| \leq m(w)$, it appears in w_1 and every time it appears it must be followed by the symbol \$. By the same reasoning ub appears in w_1 and every time it appears it must be preceded by the symbol \$.

We now show the other inclusion. Let $v = aub$ be a minimal forbidden word for w. By the second condition of I-compatibility, it is a minimal forbidden word for w_1 too. Suppose that it also belongs to S. This implies that au is a suffix of w and it does not appear elsewhere in w. Actually, if aux was a factor of w for some letter $x \in A$, then, since $|aux| \leq m(w)$, aux should be a factor of w_1 too, against

the hypothesis that it belongs to S. Analogously one can prove that ub is a prefix of w and it does not appear elsewhere in w. Thus $v = aub$ belongs to $B(w)$. □

Proposition 38 *For a finite word w over a finite alphabet A it can exist at most one bad minimal forbidden word,* i.e. $|B(w)| \leq 1$.

Proof Let $v = aub$ be a bad minimal forbidden word for w, and suppose that $v' = xu'y$ is another bad minimal forbidden word for w longer than v, *i.e.* $|u'| > |u|$. By the definition of bad minimal forbidden word, u' and ub are prefixes of w, so $u' = ubs$ for some $s \in A^*$. But $xu' = xubs$ must be suffix of w, and ub cannot appear as central factor of w. The only possible case is therefore $s = \varepsilon$ and so $u' = ub$. In this case ub is a suffix of w. Since v is bad minimal forbidden word, au is a suffix of w too, so the unique possibility is $a = b$ and $u = a^n$ for some integer n, and then $v = a^{n+2}$ (and $v' = xa^{n+1}y$, with x and y different from a).

So, if w is the word a^{n+1}, for a letter $a \in A$, then for w it exists just one bad minimal forbidden word, that is $v = a^{n+2}$; else, if $w \neq a^{n+1}$, the factor a^{n+1} appears twice as factor of w, that is a contradiction since v is a bad minimal forbidden word for w and so, by the definition, its prefix and its suffix must appear just once as factors of w.

Thus, it cannot exist a bad minimal forbidden word for w longer than v. □

Remark. The previous propositions allow to find out the elements of $\mathcal{MF}(w)$ starting from the set $\mathcal{MF}(w_1)$. In fact, we only need to remove from the set $\mathcal{MF}(w_1) \cap A^*$ the elements of $S \setminus B(w)$, *i.e.*

$$\mathcal{MF}(w) = (\mathcal{MF}(w_1) \cap A^*) \setminus (S \setminus B(w)).$$

Unfortunately, since we do not know the word w *a priori* we cannot know whether the set $B(w)$ is empty. However we can solve the problem by taking in account the lengths of the shortest elements of the set S. In the next section we will show that the set $\mathcal{MF}(w)$ is completely determined by the knowledge of the following two values without the explicit knowledge of the value $m(w)$:

$$l_1 = \min\{|v|,\ v \in S\},$$

$$l_2 = \min\{|v|,\ v \in S \ \text{ and } |v| > l_1\}.$$

We conventionally define $l_1 = l_2 = \infty$ if $S = \emptyset$ and $l_2 = \infty$ if S contains just one element.

6.4 A New Algorithm for the Word Assembly Problem

We start with the construction of a procedure that computes in linear time $O(|A|^2 \times ||I||)$ the values l_1 and l_2 starting from the factor automaton of w_1 and its trie of the minimal forbidden words.

We recall that if S contains a bad minimal forbidden word for w, it is the only element in S having length smaller than or equal to $m(w)$. So, if there exists a bad minimal forbidden word v for the word w, then its length is l_1. Thus, every word in $\mathcal{MF}(w_1) \cap A^*$ having length smaller than l_2 is a minimal forbidden word for w. If instead no bad minimal forbidden word for w exists, then every word in $\mathcal{MF}(w_1) \cap A^*$ having length smaller than l_1 is a minimal forbidden word for w.

The first algorithm is the S-CONSTRUCTION. It labels the sink states of the trie of the minimal forbidden words for w_1.

The algorithm uses a FIFO (First In First Out) file F, of which the entries are couples of states, the first one corresponding to a breadth-first-search on the factor automaton of w_1, and the second one corresponding to a breadth-first-search on the trie of the minimal forbidden words for w_1.

The behavior of the algorithm is the following. First it fixes a symbol $x \in A \cup \{\$\}$. Then it fixes a letter $a \in A$ and starts a breadth-first-searches on the trie of the minimal forbidden words for the word w_1. At the end of the exploration of the trie it labels with the symbol x the sink states corresponding to the minimal forbidden words $v = aub$ verifying:

1. v does not contain the symbol \$
2. xub is a factor of w_1
3. in the word w_1 the factor au can only be followed by the symbol \$

To perform this operation, the algorithm does at the same time a breadth-first-search on the factor automaton of w_1, to ensure that the factor xub is indeed a factor of w_1.

At the end of the procedure, the sink states labelled only by the symbol \$ are the states corresponding to the words of S.

In the worst theoretical case, if we note $k = |A|$, the algorithm does $k(k+1)$ breadth-first-searches ($k+1$ possibilities for the letter x at line 1., and k for the letter a at line 2.) on the trie $\mathcal{MF}(w_1)$. Since the set of the states that are not sink of the trie $\mathcal{MF}(w_1)$ is a subset of the set of the states of the factor automaton $\mathcal{F}(w_1)$, and since a breadth first search is a linear standard procedure, the algorithm is linear on $||I||$, where $||I||$ denotes the sum of the lengths of all the strings in I.

```
S-CONSTRUCTION (Factor Automaton F(w1) = (Q, A ∪ {$}, i, F, δ),
MF-Trie T(w1) = (Q', A ∪ {$}, i', {sinks}, δ'), FIFO file F)
 1. for each x ∈ A ∪ {$} do
 2.       for each a ∈ A do
 3.            if δ(i, x) and δ'(i', a) both defined and δ'(i', a) not sink then
 4.                 F ← {(δ(i, x), δ'(i', a))};
 5.                 while F is not empty do
 6.                      (p, p') ← HEAD (F);
 7.                      DEQUEUE (F);
 8.                      for each b ∈ A do
 9                            if δ'(p', b) is a sink state then
 10.                             if δ(p', $) defined but for every
                                 c ∈ A δ(p', c) is not defined then
 11.                                 LABEL (δ'(p', b), x);
 12.                           else if δ'(p', b) and δ(p, b) both defined then
 13.                                ENQUEUE (F, (δ(p, b), δ'(p', b)));
 14. return T'(w1) = (Q', A, i', {labeled sinks}, δ');
```

The second algorithm, l_1, l_2-FINDING, does a breadth-first-search on the labeled trie $\mathcal{T}'(w_1)$ using a FIFO file F having two entries: the first one is a state and second one is the distance of this state from the initial state.

The algorithm returns the lengths l_1 and l_2 respectively of the shortest and of the second shortest minimal forbidden words for w_1 whose label in the trie $\mathcal{T}'(w_1)$ is only the symbol \$, *i.e.* the shortest and the second shortest elements of S. If S is empty, then the algorithm sets both l_1 and l_2 equals to infinity. If S contains juste

one element, then the algorithm sets l_2 equal to infinity.

The linear time complexity of the l_1, l_2-FINDING procedure follows from the linear time complexity of the standard breadth-first-search procedure on a finite graph.

```
l_1, l_2-FINDING (labeled MF-Trie T'(w_1) = (Q', A, i', {labeled sinks}, δ'),
FIFO file F)
 1. l_1 ← ∞;
 2. l_2 ← ∞;
 3. F ← {i', 1};
 4. while F is not empty do
 5.       (p', d) ← HEAD (F);
 6.       DEQUEUE (F);
 7.       for each a ∈ A do
 8.            if δ'(p', a) is a sink state and its label is {$} then
 9.                 if d < l_1 then l_1 ← d
10.                 else if d < l_2 then l_2 ← d
11.            else if δ'(p', a) defined but not sink then
12.                 ENQUEUE (F, (δ'(p', a), d + 1));
13. return l_1, l_2;
```

Now, we describe the WORD ASSEMBLY 1 algorithm. It reconstructs the word w from a set of fragments I under the hypothesis that there exists a I-compatible word w.

The first step of the procedure is the construction of the concatenation word w_1, that can be easily done in linear time $O(||I||))$.

Then we can construct the factor automaton of w_1. Remember that the factor automaton of a word v over the alphabet A can be computed in linear time $O(|v| \times |A|)$ and has no more than $2|v|$ states, see for instance [26].

Now, we can construct in linear time $O(|w_1|) = O(||I||)$ the trie $\mathcal{T}(w_1)$ of the set $\mathcal{MF}(w_1)$, by using the MF-TRIE algorithm.

Once we have constructed both the factor automaton and the trie of the minimal forbidden words for w_1, we can use the two algorithms S-CONSTRUCTION and l_1, l_2-FINDING to find the values l_1 and l_2.

Now we apply the CREATE-TRIE algorithm with the value $l_1 - 1$ instead of the value $m(w)$ at the line 6, and we obtain the trie $\mathcal{T}(l_1 - 1)$ that represents the set $M_1 = \mathcal{MF}(w_1) \cap A^{\leq l_1 - 1}$ (we use the convention that $A^{\leq \infty} = A^*$). If no bad minimal forbidden word exists for the I-compatible word w, then $M_1 = S = \mathcal{MF}(w)$, so we

can easily reconstruct w by using the w-RECONSTRUCTION procedure applied to the trie $\mathcal{T}(l_1 - 1)$.

If instead there exists a bad minimal forbidden word for the I-compatible word w, then the L-automaton applied to the trie $\mathcal{T}(l_1 - 1)$ will give, by the Corollary 33, an automaton $\mathcal{F}(l_1 - 1)$ with some loops [1]. Note that checking whether a finite directed graph contains loops or not is a standard linear procedure (by using for example a depth-first-search).

So, if $\mathcal{F}(l_1 - 1)$ contains some loops, and since by the hypothesis there exists a I-compatible word w, then we can state that there exists a bad minimal forbidden word for w, and so, by the Remark at the end of Section 6.3, the CREATE-TRIE algorithm, with the value $l_2 - 1$ instead of the value $m(w)$ at line 6, will produce the trie of the minimal forbidden words for the word w, that so can be reconstructed by using the w-RECONSTRUCTION procedure applied to the trie $\mathcal{T}(l_2 - 1)$.

WORD ASSEMBLY 1 (set of fragments $I = \{i_1, i_2, \ldots, i_n\}$,
existence of a I-compatible word)

1. $w_1 \leftarrow \$i_1\$i_2\$ \ldots \$i_n\$$;
2. $\mathcal{F}(w_1) = (Q_1, A \cup \{\$\}, i_1, Q_1, \delta_1) \leftarrow$FACTOR-AUTOMATON$(w_1)$;
3. $\mathcal{T}(w_1) = (Q_2, A \cup \{\$\}, i_2, \{\text{sinks}\}, \delta_2) \leftarrow$MF-TRIE $(\mathcal{F}(w_1), h)$;
4. $\mathcal{T}'(w_1) = (Q_2, A, i_2, \{\text{labeled sinks}\}, \delta_2) \leftarrow S$-CONSTRUCTION$(\mathcal{F}(w_1), \mathcal{T}(w_1))$;
5. $(l_1, l_2) \leftarrow l_1, l_2$-FINDING $(\mathcal{T}'(w_1))$;
6. $\mathcal{T}(l_1 - 1) = (Q_3, A, i_2, \{\text{sinks}\}_1, \delta_3) \leftarrow$CREATE-TRIE$(\mathcal{F}(w_1), l_1 - 1)$;
7. $\mathcal{F}(l_1 - 1) = (Q_4, A, i_4, Q_4 = Q_3 \setminus \{\text{sinks}\}_1, \delta_4) \leftarrow$L-AUTOMATON $(\mathcal{T}(l_1 - 1))$;
8. **if** $\mathcal{F}(l_1 - 1)$ contains loops **then**
9. $\mathcal{T}(l_2 - 1) = (Q_5, A, i_5, \{\text{sinks}\}_2, \delta_5) \leftarrow$CREATE-TRIE$(\mathcal{F}(w_1), l_2 - 1)$;
10. $w \leftarrow w$-RECONSTRUCTION$(\mathcal{T}(l_2 - 1))$;
11. **return** w;
12. **else**
13. $w \leftarrow w$-RECONSTRUCTION$(\mathcal{T}(l_1 - 1))$;
14. **return** w;

We are now quite close to the solution of the Word Assembly Problem as formulated in Section 6.1. The last step consists in eliminating the hypothesis on the existence of a I-compatible word.

[1] In the general construction the L-automaton has the same states of its input trie, but we always suppose to delete the sink states of the L-automaton after its construction. So we do not consider loops on the sink states.

So we start only with an arbitrary set I of finite words over a finite alphabet A.

The following WORD ASSEMBLY 2 algorithm completely answers to the Word Assembly Problem, and it produces its output in linear time $O(||I||)$.

The first steps of the algorithm are the same as those of the WORD ASSEMBLY 1 algorithm. Once we have constructed the automaton $\mathcal{F}(l_1 - 1)$ we have to check whether it is (after deleting the sink states) the factor automaton of a I-compatible word.

If it contains loops, clearly it cannot be the factor automaton of a single finite word.

If it does not contain loops, how can we decide whether it is the factor automaton of a I-compatible word? First, the factor automaton of a single finite word always contains a unique longest path from the initial state, and it is the path corresponding to the longest factor of the word, that is the word itself. So if $\mathcal{F}(l_1 - 1)$ contains two or more paths of maximal length (one can check this in linear time by using a simple adaptation of the topological sort procedure on a directed acyclic graph), we can state that no I-compatible word exists; otherwise, by the Corollary 33, $\mathcal{F}(l_1 - 1)$ should contain loops. If instead $\mathcal{F}(l_1 - 1)$ contains just one path of maximal length, set w this path. Now, one can construct in linear time on the size of w (and so on $||I||$) the factor automaton of w, noted by $\mathcal{F}(w)$. We can now compare the automata $\mathcal{F}(w)$ and $\mathcal{F}(l_1 - 1)$ (it is well known that checking the equality between two finite deterministic automata can be done in linear time).

If $\mathcal{F}(w) \neq \mathcal{F}(l_1 - 1)$, then we can state that no I-compatible word exists, if not we should retrieve its factor automaton with $\mathcal{F}(l_1 - 1)$ (or $\mathcal{F}(l_2 - 1)$ if $\mathcal{F}(l_1 - 1)$ contains loops, that is not the case here).

If $\mathcal{F}(w) = \mathcal{F}(l_1 - 1)$ then we have found a I-compatible word, w, as the following Theorem shows.

Theorem 39 *If $\mathcal{F}(l_1 - 1)$, the automaton that recognizes the set $L(\mathcal{MF}(w_1) \cap A^{\leq l_1 - 1})$, is the factor automaton of a finite word w, then w is a I-compatible word.*

Proof The first condition to prove is that $I \subseteq \text{Fact}(w)$. Let $v \in I$, then $v \in \text{Fact}(w_1) \cap A^*$, so $v \notin \mathcal{MF}(w_1) \cap A^*$. Therefore $v \in L(\mathcal{MF}(w_1) \cap A^*)$ and thus *a fortiori* $v \in L(\mathcal{MF}(w_1) \cap A^{\leq l_1 - 1})$. Thus v is recognized by $\mathcal{F}(l_1 - 1) = F(w)$.

The second condition to prove is that $C(I) \geq m(w)$. Suppose that $k = C(I) <$

```
WORD ASSEMBLY 2 (set of fragments I = {i_1, i_2, ..., i_n})
 1. w_1 ← $i_1$i_2$...$i_n$;
 2. F(w_1) = (Q_1, A ∪ {$}, i_1, Q_1, δ_1) ←FACTOR-AUTOMATON(w_1);
 3. T(w_1) = (Q_2, A ∪ {$}, i_2, {sinks}, δ_2) ←MF-TRIE (F(w_1), h);
 4. T'(w_1) = (Q_2, A, i_2, {labeled sinks}, δ_2) ←S-CONSTRUCTION(F(w_1), T(w_1));
 5. (l_1, l_2) ←l_1, l_2-FINDING (T'(w_1));
 6. T(l_1 − 1) = (Q_3, A, i_3, {sinks}_1, δ_3) ←CREATE-TRIE(F(w_1), l_1 − 1);
 7. F(l_1 − 1) = (Q_4, A, i_4, Q_4 = Q_3 \ {sinks}_1, δ_4) ←L-AUTOMATON (T(l_1 − 1));
 8.     if F(l_1 − 1) contains loops then
 9.         T(l_2 − 1) = (Q_5, A, i_5, {sinks}_2, δ_5) ←CREATE-TRIE(F(w_1), l_2 − 1);
10.         F(l_2 − 1) = (Q_6, A, i_6, Q_6 = Q_5 \ {sinks}_2, δ_6) ←L-AUTO-
            MATON (T(l_2 − 1));
11.             if F(l_2 − 1) contains loops then
12.                 return "No I-compatible word exists";
13.             else
14.                 if F(l_2 − 1) has a unique path of maximal length
                    from the initial state, labelled by w, then
15.                     F(w) ←FACTOR-AUTOMATON(w);
16.                         if F(w) = F(l_2 − 1) then
17.                             return w;
18.                         else
19.                             return "No I-compatible word exists";
20.     else
21.         if F(l_1 − 1) has a unique path of maximal length
            from the initial state, labelled by w, then
22.             F(w) ←FACTOR-AUTOMATON(w);
23.                 if F(w) = F(l_1 − 1) then
24.                     return w;
25.                 else
26.                     return "No I-compatible word exists";
27.         else
28.             return "No I-compatible word exists";
```

$m(w)$. This implies that there exists at least one factor $v \in \text{Fact}(w)$ such that $|v| = k + 1$ but $v \notin \text{Fact}(I)$, so $v \notin \text{Fact}(w_1) \cap A^*$. Let $v = a_0 a_1 \dots a_k$. Since $k = C(I)$ we have that $a_0 a_1 \dots a_{k-1}$ and $a_1 a_2 \dots a_k$ belong to $\text{Fact}(w_1) \cap A^*$, so $v \in \mathcal{MF}(w_1) \cap A^*$.

We thus have two possibilities for v.

If $|v| < l_1$, then $v \in \mathcal{MF}(w_1) \cap A^{\leq l_1 - 1}$, therefore $v \notin L(\mathcal{MF}(w_1) \cap A^{\leq l_1 - 1})$, so it should not be recognized by $\mathcal{F}(l_1 - 1) = F(w)$, that is a contradiction.

If instead $|v| \geq l_1$, we should have $m(w) > l_1 - 1$. But this is impossible, because in this case $L(\mathcal{MF}(w_1) \cap A^{\leq l_1 - 1})$ could not avoid the minimal forbidden words for w having length $m(w)$, against the hypothesis that $\mathcal{F}(l_1 - 1) = \mathcal{F}(w)$. □

The previous Theorem also shows that the algorithm Word Assembly 2 cannot retrieve a finite word w that is not I-compatible.

If $\mathcal{F}(l_1 - 1)$ contains loops, we can try with the automaton $\mathcal{F}(l_2 - 1)$, obtained from the Create-trie procedure with the value l_2.

If it contains loops, we can state that no I-compatible word exists, otherwise we should obtain an automaton without loops, as in the Word Assembly 1 algorithm. If instead it does not contain loops, we can apply the same test as that we did for $\mathcal{F}(l_1 - 1)$.

The linear time complexity of the whole Word Assembly 2 algorithm follows from the linear time complexity of all the used procedures.

6.5 Relation With the Shortest Superstring Problem

In this section we want to show that if w is a I-compatible word, then w is solution of the Shortest Superstring Problem for the set I, that is finding the shortest word w such that all the fragments in I are factors of w. This implies that for those sets I for which there exists a (unique) I-compatible word, we are able to find the shortest superstring for I in linear time on the size of I.

In the sequel we suppose that the input set of fragments I is an anti-factorial set, *i.e.* there exists not two distinct fragments i_j and i_k in I such that i_j is a factor of i_k.

Given an arbitrary set I, it is possible to retrieve its anti-factorial part I' (that is the set obtained from I by eliminating the strings that are factors of another string of I) in linear time on the size of I. For instance one can perform this task by using a generalized suffix tree (see [33]) of the fragments in I and eliminating fragments that are factors of other ones.

It is easy to see that our algorithms Word Assembly 1 and Word Assembly 2 give the same output on the input I and on the input I'.

Definition 6.3 *For any couple of fragments i_j, i_k in I, an* overlap *between i_j and i_k is a word $u_{j,k} \in \mathrm{Suff}(i_j) \cap \mathrm{Pref}(i_k)$. The* maximum overlap *between i_j and i_k is the longest overlap between i_j and i_k.*

Definition 6.4 *An* arrangement *$A_\sigma(I)$ of $I = \{i_1, \ldots, i_n\}$ is a word w obtained by concatenating the fragments in I in an order given by a permutation σ over n elements; so*

$$A_\sigma(I) = i_{\sigma(1)} \cdots i_{\sigma(n)}.$$

Definition 6.5 *An* arrangement with overlap *$\overline{A}_\sigma(I)$ of I is an arrangement in which two consecutive factors can overlap. An* arrangement with maximum overlap *$\overline{A}^m_\sigma(I)$ is an arrangement with overlap in which every overlap between two consecutive factors is a maximum overlap.*

Note that for every fixed permutation σ there exists a unique arrangement with maximum overlap $\overline{A}^m_\sigma(I)$.

It is straightforward that a shortest word w such that the fragments in I are all factors of w, *i.e.* a Shortest Superstring of I, is an arrangement with maximum overlap $\overline{A}^m_\sigma(I)$ for a permutation σ which minimizes the length of the arrangement, *i.e.* which maximizes the sum of the lengths of the overlaps.

Note that if w is a I-compatible word, then w in particular corresponds to an arrangement with overlap of I, so we can associate to the word w a permutation σ over $|I|$ elements such that $w = \overline{A}_\sigma(I)$.

Proposition 40 *If w is a I-compatible word corresponding to a permutation σ, then all the overlaps between consecutive factors of I in the arrangement $w = \overline{A}_\sigma(I)$ have length greater than $m(w) - 2$. In particular they appear just once as factors of w.*

Proof From the definition of I-compatible word, we know that every factor of w having a length smaller than or equal to $m(w)$ is contained in some fragment in I.

Suppose that in the arrangement corresponding to σ there exists two consecutive fragments $i_{\sigma(j)}, i_{\sigma(j)+1}$ such that their overlap $u_{\sigma(j),\sigma(j)+1}$ has a length $|u_{\sigma(j),\sigma(j)+1}| \leq m(w) - 2$. Let $a \in A$ be the letter preceding $u_{\sigma(j),\sigma(j)+1}$ in w, and $b \in A$ be the letter following $u_{\sigma(j),\sigma(j)+1}$ in w. Then $au_{\sigma(j),\sigma(j)+1}b$ is a factor of w having length $|au_{\sigma(j),\sigma(j)+1}b| \leq m(w)$ which is not contained in any fragment of I, that is a contradiction.

So every overlap u between consecutive fragments of I in the arrangement corresponding to σ has length greater than $m(w) - 2$. By the proposition 35 we know that u appears just once as factor of w. □

Lemma 41 *If w is a I-compatible word then w is an arrangement with maximum overlap of I, for a permutation σ over $|I|$ elements.*

Proof Let w be an arrangement with overlap $\overline{A}_\sigma(I)$ of I, for a permutation σ over $|I|$ elements. We want to prove that there exists not another choice of the overlaps giving an arrangement with overlap $\overline{A'}_\sigma(I)$ shorter than $\overline{A}_\sigma$. Actually, if there exists such an arrangement, then there should exist at least two consecutive fragments $i_{\sigma(j)}, i_{\sigma(j)+1}$ with overlap $u'_{\sigma(j),\sigma(j)+1}$ longer than the overlap $u_{\sigma(j),\sigma(j)+1}$ which they have in the arrangement $\overline{A}_\sigma$.

Since both $u'_{\sigma(j),\sigma(j)+1}$ and $u_{\sigma(j),\sigma(j)+1}$ are prefixes of $i_{\sigma(j)+1}$ by the definition, and since $u'_{\sigma(j),\sigma(j)+1}$ is longer than $u_{\sigma(j),\sigma(j)+1}$, we conclude that $u_{\sigma(j),\sigma(j)+1}$ is a (strict) prefix of $u'_{\sigma(j),\sigma(j)+1}$. Moreover, since $u'_{\sigma(j),\sigma(j)+1}$ and $u_{\sigma(j),\sigma(j)+1}$ are by the definition suffixes of $i_{\sigma(j)}$, we can state that the overlap $u_{\sigma(j),\sigma(j)+1}$ appears in (at least) two different positions as factor of the fragment $i_{\sigma(j)}$, and so in particular as factor of the word $w = \overline{A}_\sigma(I)$. The contradiction follows then by the Proposition 40. □

Theorem 42 *If w is a I-compatible word then w is a solution of the Shortest Superstring Problem for the set I.*

Proof Let w be a I-compatible word, where $I = \{i_1, \ldots, i_n\}$. We have to prove that w is the shortest word such that $I \subset \mathrm{Fact}(w)$, *i.e.* the shortest arrangement with maximum overlap of I.

So let $w = \overline{A}^m_\sigma(I)$. We want to prove that for every other permutation over n elements τ one has $|\overline{A}^m_\sigma(I)| < |\overline{A}^m_\tau(I)|$.

We claim that for any couple of fragments i_k and i_l which are consecutive in $\overline{A}^m_\tau(I)$ but not in $\overline{A}^m_\sigma(I)$ one has that their (maximum) overlap $u_{k,l}$ in $\overline{A}^m_\tau(I)$ has length smaller than or equal to $m(w)-2$. Actually, since i_k and i_l are not consecutive in $\overline{A}^m_\sigma(I) = w$, $u_{k,l}$ is a factor which appears at least twice in w and the claim follows from Proposition 35. Since τ is different from σ there exists two fragments which are consecutive in $\overline{A}^m_\tau(I)$ but not in $\overline{A}^m_\sigma(I)$. The thesis follows then by the Proposition 40.

□

We end this section with a comparison between our algorithm Word Assembly 2 and the well known Greedy one. The Greedy algorithm is an approximation algorithm that repeatedly merges the two fragments of the set I having maximum overlap until only one word remains. If there exists more than one couple of fragments having maximum overlap it randomly chooses one of these couples for the merging. Greedy is not linear, since it must compare every couple of fragments to determinate the maximum overlaps.

Lemma 43 *If there exists a I compatible word w for the set I, then for every fragment i_j in I which is not suffix of w there exists just one fragment i_k in I such that*

$$|u_{j,k}| > \max_{l \neq k} |u_{j,l}|.$$

Proof Let i_j be in I, and suppose that there exists two distinct fragments i_k and $i_{k'}$ in I such that the maximum overlaps $u_{j,k}$ between i_j and i_k and $u_{j,k'}$ between i_j and $i_{k'}$ have the same length. Since we supposed that I is an anti-factorial set we have that $u_{j,k} = u_{j,k'}$ is a strict suffix of i_j and a strict prefix of i_k and $i_{k'}$. Thus, in the I-compatible word w, the factor $u_{j,k} = u_{j,k'}$ should appear at least twice, and this is impossible because we have shown in the Lemma 40 that the maximum overlap between two consecutive fragments in w appears just once as factor of w. □

Corollary 44 *If I is a set for which there exists a (unique) I compatible word w, then the Greedy algorithm applied on the set I is deterministic, therefore it gives the shortest superstring for the set I.*

Remark. The output of the Greedy algorithm is the same of our algorithm Word Assembly 2, since we showed that our algorithm too gives the shortest

superstring for the set I. So WORD ASSEMBLY 2 (that runs in linear time on the size of I) improves the Greedy algorithm for those sets I for which there exists a (unique) I-compatible word.

Appendix A

An example of calculus of Word Assembly

We show here an example of how algorithm Word Assembly 2 works. We start with the set $I = \{baa, aba\}$.

1. Construction of the concatenation word $w_1 = \$baa\$aba\$$ (line 1).

2. Construction of the factor automaton of w_1, shown in Figure A.1, (line 2).

3. Construction of the trie of minimal forbidden words for w_1, shown in Figure A.2, line 3.

4. Construction of the set S, line 4.

5. Computation of the values l_1, and l_2, line 5. Here we have $l_1 = 3$ and $l_2 = 4$.

6. Pruning the trie of minimal forbidden words for w_1 at length $l_1 - 1$, gives the trie $\mathcal{T}(l_1 - 1)$ shown in Figure A.3, line 6.

7. Computation of the L-automaton of the trie $\mathcal{T}(l_1 - 1)$, as shown in Figure A.4, line 7.

8. Test of loops on $L(\mathcal{T}(l_1 - 1))$, line 8. The automaton does contain a loop on state 0.

9. Pruning the trie of minimal forbidden words for w_1 at length $l_2 - 1$, gives the trie $\mathcal{T}(l_2 - 1)$ shown in Figure A.5, line 9.

10. Computation of the L-automaton of the trie $\mathcal{T}(l_2 - 1)$, as shown in Figure A.7, line 10.

11. Test of loops on $L(\mathcal{T}(l_2 - 1))$, line 11. The automaton does not contain any loops.

```
WORD ASSEMBLY 2 (set of fragments I = {i_1, i_2, ..., i_n})
 1. w_1 ← $i_1$i_2$...$i_n$;
 2. F(w_1) = (Q_1, A ∪ {$}, i_1, Q_1, δ_1) ←FACTOR-AUTOMATON(w_1);
 3. T(w_1) = (Q_2, A ∪ {$}, i_2, {sinks}, δ_2) ←MF-TRIE (F(w_1), h);
 4. T'(w_1) = (Q_2, A, i_2, {labeled sinks}, δ_2) ←S-CONSTRUCTION(F(w_1), T(w_1));
 5. (l_1, l_2) ←l_1, l_2-FINDING (T'(w_1));
 6. T(l_1 − 1) = (Q_3, A, i_3, {sinks}_1, δ_3) ←CREATE-TRIE(F(w_1), l_1 − 1);
 7. F(l_1 − 1) = (Q_4, A, i_4, Q_4 = Q_3 \ {sinks}_1, δ_4) ←L-AUTOMATON (T(l_1 − 1));
 8.     if F(l_1 − 1) contains loops then
 9.         T(l_2 − 1) = (Q_5, A, i_5, {sinks}_2, δ_5) ←CREATE-TRIE(F(w_1), l_2 − 1);
10.         F(l_2 − 1) = (Q_6, A, i_6, Q_6 = Q_5 \ {sinks}_2, δ_6) ←L-AUTO-
            MATON (T(l_2 − 1));
11.             if F(l_2 − 1) contains loops then
12.                 return "No I-compatible word exists";
13.             else
14.                 if F(l_2 − 1) has a unique path of maximal length
                    from the initial state, labelled by w, then
15.                     F(w) ←FACTOR-AUTOMATON(w);
16.                         if F(w) = F(l_2 − 1) then
17.                             return w;
18.                         else
19.                             return "No I-compatible word exists";
20.     else
21.         if F(l_1 − 1) has a unique path of maximal length
            from the initial state, labelled by w, then
22.             F(w) ←FACTOR-AUTOMATON(w);
23.                 if F(w) = F(l_1 − 1) then
24.                     return w;
25.                 else
26.                     return "No I-compatible word exists";
27.         else
28.             return "No I-compatible word exists";
```

12. Test of the unique path of maximal length on $L(\mathcal{T}(l_2 - 1))$, line 14. The automaton has got a unique path of maximal length, which label is $w = abaa$.

13. Construction of the factor automaton of $w = abaa$, shown in Figure A.7, line 15.

14. Comparison between $L(\mathcal{T}(l_2 - 1))$ and the factor automaton of $w = abaa$, line 23. They are the same, so there exists a (unique) I-compatible word, that is $w = abaa$.

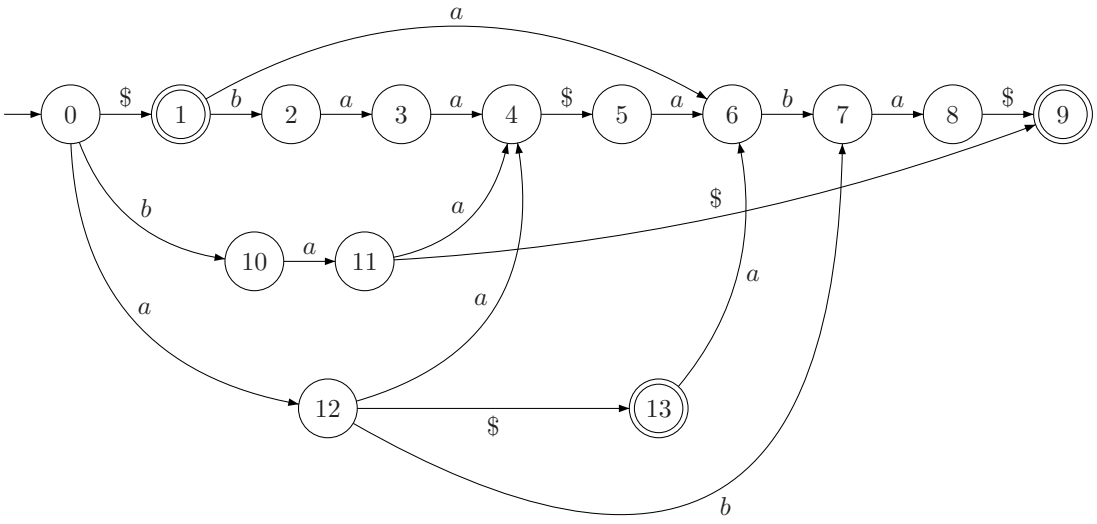

Figure A.1: The suffix automaton of the word $w_1 = \$baa\$aba\$$.

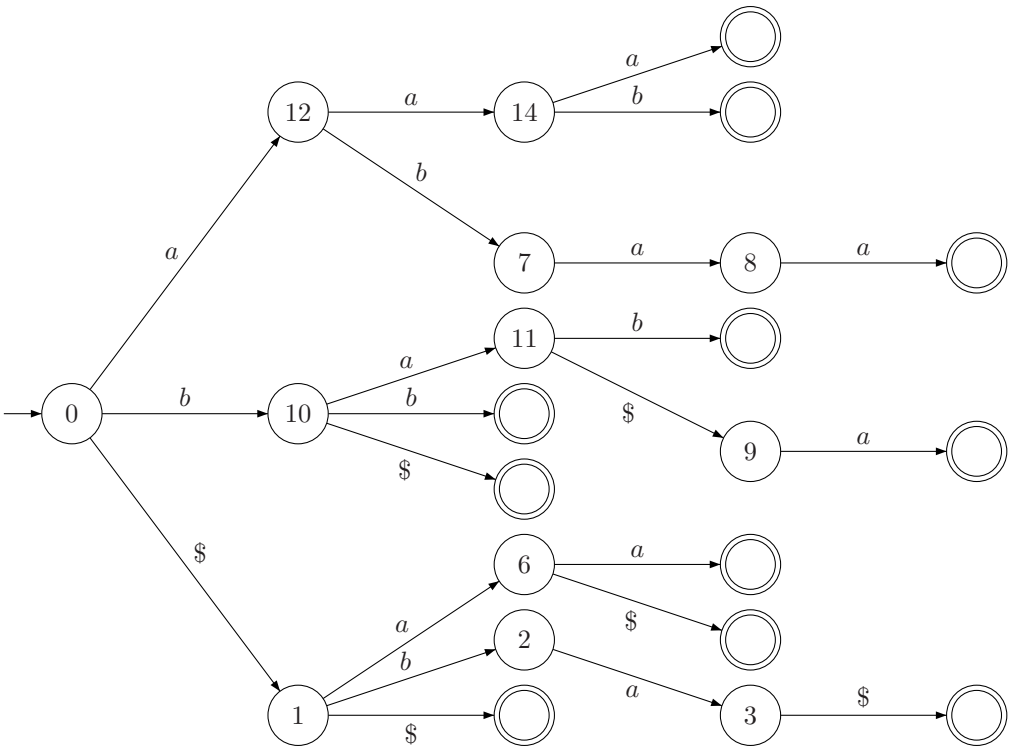

Figure A.2: The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$.

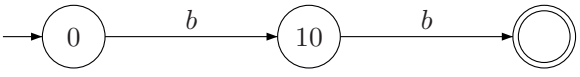

Figure A.3: The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$ cut at length l_1.

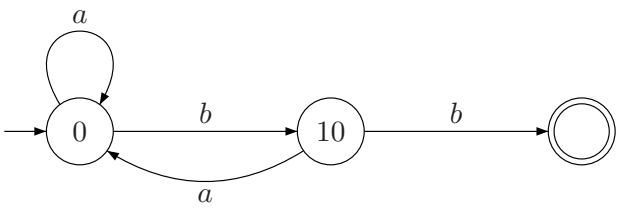

Figure A.4: Building the L-automaton of $\mathcal{T}(l_2 - 1)$.

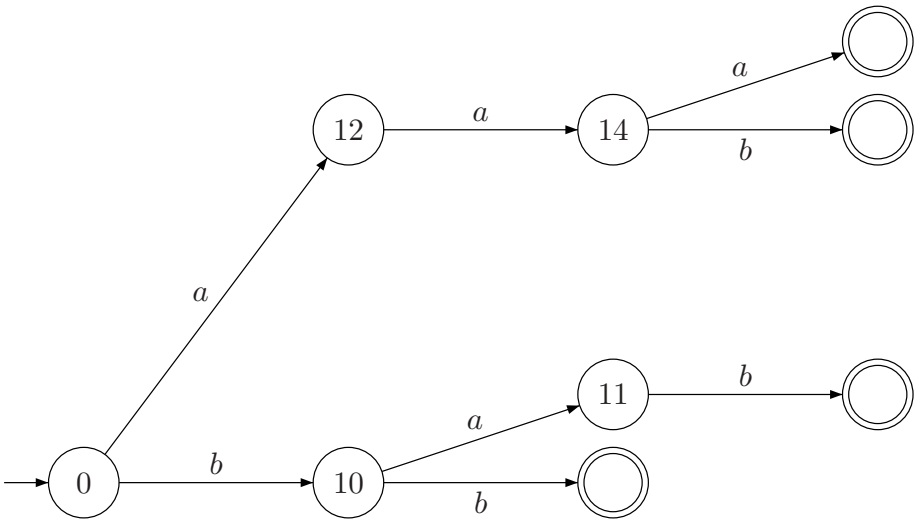

Figure A.5: The trie of minimal forbidden words for the word $w_1 = \$baa\$aba\$$ cut at length l_2.

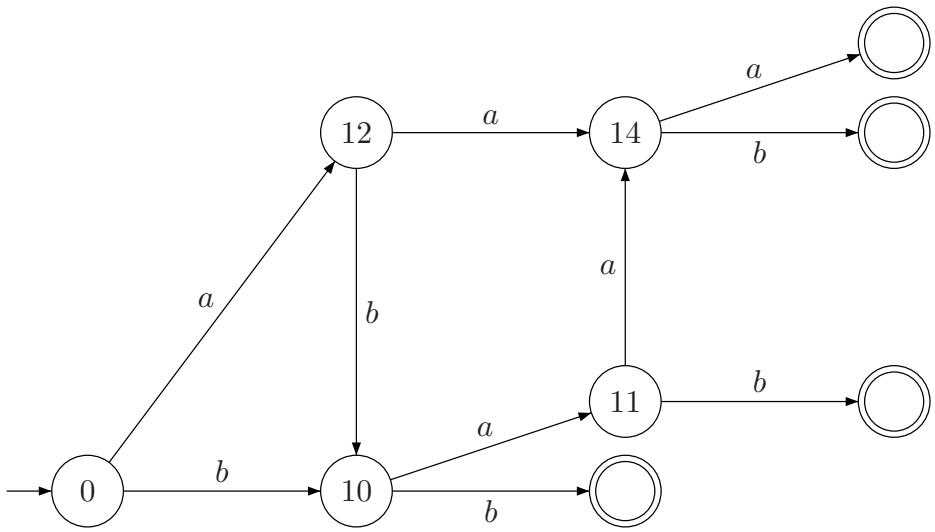

Figure A.6: Building the L-automaton of $\mathcal{T}(l_2 - 1)$.

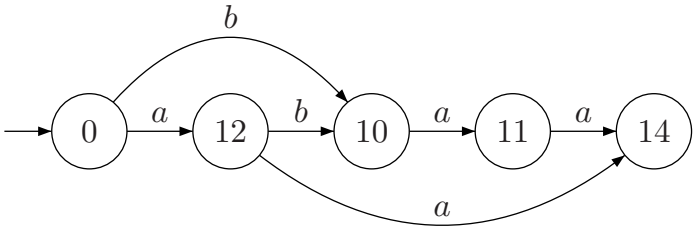

Figure A.7: The L-automaton of $\mathcal{T}(l_2 - 1)$. It is the the same automaton as in figure A.6 but without sink states.

Conclusions

In this thesis we explained different theoretical aspects of minimal forbidden words and showed some applications to practical problems.

We now want to give an outline of possible further works on the subjects we treated.

First, one can wonder whether it is possible to mix the results of Chapter 5 and Chapter 6 in order to obtain an algorithm for the reconstruction of a finite word in the periodic case, that is, fixed a period T, starting from T sets of minimal periodic forbidden words, one for each phase. This is in fact a work in progress, but some technical difficulties arose during the generalization of some theorems in 6.2, and hence we decided to not include in this thesis the partial results we obtained so far.

Another work in progress is the extension of minimal forbidden words to the bi-dimensional case. Even if some results have been obtained in the context of symbolic dynamics [5], we hope that an analogous of minimal forbidden words for a finite word shall be defined for bi-dimensional words. This could lead to new algorithms for bi-dimensional pattern matching, and to other generalizations of the results obtained in the single-word case to higher dimensions.

In a more general frame, we want to improve the results obtained in Chapter 6 and give a more realistic algorithm for the Fragment Assembly Problem based on minimal forbidden words. Actually, we stress that the results presented in Chapter 6 are theoretical and concern the reconstruction of a finite word from a set of its factors. But our results could be extended by including some other constraints, as the orientation of the input fragments or the read-errors obtained during the shotgun sequencing procedure (for example, the work of Gabriele *et al.* (cf. [32]) could be

useful to treat the case of read-errors). An algorithm of practical interest should output a solution (or a range of possible solutions) on every possible input (see [49,57] for more technical details), for example even when the fragments obtained from the shotgun do not cover the whole sequence.

Bibliography

[1] A. V. Aho, J.E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Prentice-Hall, 1983.

[2] R. Badii and A. Politi. *Complexity, Hierarchical Structures and Scaling in Physics*. Cambridge University Press, 1997.

[3] M.P. Béal, M. Crochemore, and G. Fici. Presentation of constrained systems with unconstrained positions. *IEEE Trans. Inform. Theory*, 51:1891–1900, May 2005.

[4] M.P. Béal, M. Crochemore, F. Mignosi, A. Restivo, and M. Sciortino. Computing forbidden words of regular languages. *Fund. Inform.*, 56(1-2):121–135, 2003. Special issue on computing patterns in strings.

[5] M.P. Béal, F. Fiorenzi, and F. Mignosi. Minimal forbidden patterns of multidimensional shifts. *Internat. J. Algebra Comput.*, 15:73–93, 2005.

[6] M.P. Béal, F. Mignosi, A. Restivo, and M. Sciortino. Forbidden words in symbolic dynamics. *Adv. in Appl. Math.*, 25(2):163–193, 2000.

[7] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnell. Linear size finite automata for the set of all subwords of a word : an outline of results. *Bull. European Assoc. Theoret. Comput. Sci.*, 21:12–20, 1983.

[8] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnell. Building a complete inverted file for a set of text files in linear time. In *Proceedings of the 16th ACM symposium on the Theory of Computing*, pages 349–351. ACM Press, 1984.

[9] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987.

[10] D. Breslauer and R. Hariharan. Optimal parallel construction of minimal suffix and factor automata. *Parallel Processing Letter*, 6(1):35–44, 1996.

[11] J. Campello de Souza, B. H. Marcus, R. New, and B. A. Wilson. Constrained systems with unconstrained positions. *IEEE Trans. Inform. Theory*, 48(4):866–879, 2002.

[12] A. Carpi and A. de Luca. Repetitions and boxes in words and pictures. In J. Karhumäki, H. Maurer, and G. Rozenberg, editors, *Jewels are Forever*, pages 295–306. Springer, Berlin, 1999.

[13] A. Carpi and A. de Luca. Special factors, periodicity, and an application to Sturmian words. *Acta Inform.*, 36:983–1006, 2000.

[14] A. Carpi and A. de Luca. Periodic-like words, periodicity and boxes. *Acta Inform.*, 37:597–618, 2001.

[15] A. Carpi and A. de Luca. Some generalizations of periodic words. *Bull. Belg. Math. Soc.*, 8:257–275, 2001.

[16] A. Carpi and A. de Luca. Words and repeated factors, *Séminaire Lotharingien de Combinatoire* **B421** (1999). In D. Foata and G.N. Han, editors, *The Andrews Festschrift*, pages 231–251. Springer, Berlin, 2001.

[17] A. Carpi and A. de Luca. Words and special factors. *Theoret. Comput. Sci.*, 259(1–2):145–182, may 2001.

[18] A. Carpi and A. de Luca. A combinatorial property of the factor poset of a word. *Inform. Process. Lett.*, 81:35–39, 2002.

[19] A. Carpi and A. de Luca. On the distribution of characteristic parameters of words. *RAIRO Theoret. Inform. Appl.*, 36:67–96, 2002.

[20] A. Carpi and A. de Luca. On the distribution of characteristic parameters of words II. *RAIRO Theoret. Inform. Appl.*, 36:97–127, 2002.

[21] A. Carpi and A. de Luca. Semiperiodic words and root-conjugacy. *Theoret. Comput. Sci.*, 292:111–130, 2003.

[22] A. Carpi, A. de Luca, and S. Varricchio. Special factors and uniqueness conditions in rational trees. *Theory of Comput. Syst.*, 34:375–395, 2001.

[23] A. Carpi, A. de Luca, and S. Varricchio. Words, univalent factors and boxes. *Acta Inform.*, 38(6):409–436, 2002.

[24] M. Crochemore and C. Hancart. Automata for matching patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2 Linear Modeling: Background and Application, chapter 9, pages 399–462. Springer-Verlag, Berlin, 1997.

[25] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithmique du Texte*. Vuibert, 2001.

[26] M. Crochemore, F. Mignosi, and A. Restivo. Automata and forbidden words. *Inform. Process. Lett.*, 67(3):111–117, 1998.

[27] M. Crochemore, F. Mignosi, and A. Restivo. Minimal forbidden words and factor automata. In *Mathematical Foundations of Computer Science, 1998 (Brno)*, volume 1450 of *Lect. Notes Comput. Sci.*, pages 665–673. Springer, Berlin, 1998.

[28] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Data compression using antidictionaries. In J. Storer, editor, *Proceedings of the IEEE Lossless Data Compression*, pages 1756–1768. 2000.

[29] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[30] G. Fici, F. Mignosi, M. Sciortino, and A. Restivo. Word assembly through minimal forbidden words. *Theoret. Comput. Sci.* to appear.

[31] G. Fici, F. Mignosi, M. Sciortino, and A. Restivo. Fragment assembly through minimal forbidden words. In *X Journées Montoises d'Informatique Théorique*. Liège, Sept. 2004.

[32] A. Gabriele, F. Mignosi, A. Restivo, and M. Sciortino. Indexing structures for approximate string matching. In R. Silvestri R. Petreschi, G. Persiano, editor, *Algorithms and Complexity: 5th Italian Conference, CIAC 2003, Rome, Italy, May 28-30, 2003. Proceedings*, volume 2653, pages 140–151. Lect. Notes Comput. Sci., 2003.

[33] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[34] D. A. Lind and B. H. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge, 1995.

[35] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, 1983.

[36] M. Lothaire. *Combinatorics on Words*. Cambridge Math. Library, 1997. Second edition.

[37] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge Univ. Press, 2002.

[38] M. Lothaire. *Applied Combinatorics on Words*. Cambridge Univ. Press, 2005.

[39] B. Marcus, P. Chaichanavong, and T.L̃. Poo. Trade-off functions for constrained systems with unconstrained positions. *IEEE Trans. Inform. Theory*. to appear.

[40] B. H. Marcus. Sofic systems and encoding data. *IEEE Trans. Inform. Theory*, 31:266–377, 1985.

[41] B. H. Marcus, R. M. Roth, and P. H. Siegel. Constrained systems and coding for recording channels. In V.S. Pless and W.C. Huffman, editors, *Handbook of Coding Theory*, volume II, chapter 20, pages 1635–1764. North Holland, 1998.

[42] F. Mignosi, A. Restivo, and M. Sciortino. Forbidden factors in finite and infinite words. In *Jewels are Forever*, pages 339–350. Springer, Berlin, 1999.

[43] F. Mignosi, A. Restivo, and M. Sciortino. Forbidden Factors and Fragment Assembly. *RAIRO Theoret. Inform. Appl.*, 35(6):565–577, 2001. (Journal version).

[44] F. Mignosi, A. Restivo, and M. Sciortino. Forbidden factors and fragment assembly. In *Developments in Language Theory (Vienna, 2001)*, volume 2295 of *Lect. Notes Comput. Sci.*, pages 349–358. Springer, Berlin, 2002.

[45] F. Mignosi, A. Restivo, and M. Sciortino. Words and forbidden factors. *Theoret. Comput. Sci.*, 273(1-2):99–117, 2002. WORDS (Rouen, 1999).

[46] F. Mignosi, A. Restivo, M. Sciortino, and J. Storer. On Sequence Assembly. Technical Report cs-00-210, Brandeis University, 2000.

[47] B. E. Moision and P. H. Siegel. Periodic-finite-type shift spaces. In *IEEE Int. Symp. Information Theory*, page 65, Washington, DC, June 24-29 2001.

[48] J. Moon and B. Brickner. Maximum transition run codes for data storage. *IEEE Trans. Magn.*, 32:3992–3994, Sept. 1996.

[49] E. W. Myers. Adavances in sequence assembly. In M.D. Adams, C. Fields, and J.C. Venter, editors, *Automata DNA Sequencing and Analysis*, pages 231–238. Academic Press, New-York, 1994.

[50] P. A. Pevzner, H. Tang, and M. S. Waterman. A new approach to fragment assembly in DNA sequencing. In *RECOMB*, pages 256–267, 2001.

[51] P. A. Pevzner, H. Tang, and M.S. Waterman. An Eulerian path approach to DNA fragment assembly. In *Proc. Nat. Acad. Sci.*, volume 98, pages 9748–9753, Aug. 2001.

[52] M. Raffinot. *Structures pour la localisation de motifs*. PhD thesis, University of Marne-la-vallée, France, 1999.

[53] D. Revuz. Minimization of acyclic deterministic automata in linear time. *Theoret. Comput. Sci.*, 92(1):181–189, 1992.

[54] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages vol.1, Word Language Grammar*. Springer, 1997.

[55] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages vol.2, Linear Modeling: Background and Application*. Springer, 1997.

[56] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages vol.3, Linear Modeling: Beyond Words*. Springer, 1997.

[57] M. T. Tammi. The principles of shotgun sequencing and automated fragment assembly. Technical report, Center for Genomics and Bioinformatics, Karolinska Institutet, Stockholm - Sweden, April 2003.

[58] M. Weinard and G. Schnitger. On the Greedy Superstring Conjecture. In *Foundations of Software Technology and Theoretical Computer Science (23rd Conference, Mumbai India, December 15-17, 2003)*, volume 2914 of *Lect. Notes Comput. Sci.*, pages 387–398. Springer, Berlin, 2003.

[59] A. Wijngaarden and K. S. Immink. Maximum run-length limited codes with error control properties. *IEEE J. Select. Areas Commun.*, 19:602–611, Apr. 2001.