Presentations of Constrained Systems with Unconstrained Positions

Marie-Pierre Béal, Maxime Crochemore and Gabriele Fici

Abstract— **We give a polynomial-time construction of the set of sequences that satisfy a finite-memory constraint defined by a finite list of forbidden blocks, with a specified set of bit positions unconstrained. Such a construction can be used to build modulation/error-correction codes (ECC codes) like the ones defined by the Immink-Wijngaarden scheme in which certain bit positions are reserved for ECC parity. We give a linear-time construction of a finite-state presentation of a constrained system defined by a periodic list of forbidden blocks. These systems, called periodic-finite-type systems, were introduced by Moision and Siegel. Finally, we present a linear-time algorithm for constructing the minimal periodic forbidden blocks of a finite sequence for a given period.**

Index Terms— **Directed acyclic word graph (DAWG), finite-memory systems, finite-state encoders, forbidden blocks, maximum transition run (MTR) codes, modulation codes, periodic-finite-type (PFT) systems, run-length limited (RLL) codes.**

I. 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) [1], 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 [2] (see also [3]), 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 [3], 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 subchannel [4]. Since these code constructions work on a presentation of the subchannel, it is worth to efficiently compute a small presentation of this subchannel.

In this correspondence, we 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 [3]. 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 [5]. This was already noticed in [3, 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 [3] is exponential in the general case, they cannot be compared directly for the following reasons. The algorithm described in [3] 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

Marie-Pierre Béal and Maxime Crochemore are with the Gaspard-Monge Institute, University of Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France (e-mail: beal@univ-mlv.fr, mac@univ-mlv.fr).

Gabriele Fici is with the Gaspard-Monge institut, University of Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France, and with the Department of Mathematics and Applications, University of Palermo, Via Archirafi 34, 90123 Palermo, Italy (e-mail: fici@univ-mlv.fr).

condition. The gap condition limits the number of unconstrained positions relatively to the memory of the system. An efficient algorithm is also proposed in [3] 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 [6]. 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 [3] is more efficient if both the input data are a presentation of the constraint and the gap condition is satisfied.

Our correspondence is organized as follows. In Section II, we recall some background regarding constrained systems with unconstrained positions, which are introduced in [3]. In Section III 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 IV, we combine the algorithm given in Section III to a preliminary treatment of the input trie presenting a list of forbidden blocks of the constrained channel. Although it is not directly related to modulation/error-correction codes construction, we added a final section which provides a linear space and time computation of minimal periodic forbidden blocks of a finite sequence for a given period. This algorithm extends a known algorithm from [7] for computing the minimal forbidden blocks of a finite word, which is used in a lossless data compression scheme [8]. We also believe that the notion of periodic list of forbidden blocks introduced by Moision and Siegel can be used in many areas other than modulation/error-correction codes.

II. Background and basic definitions

We recall definitions that can be found in [9]. 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 subblock $x_i \ldots x_j$ of a finite or infinite word x. A finite word w is a subblock 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 multigraph (or simply a graph). It is a tuple (Q, A, E), where Q is a finite set of states, A is the labeling 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.

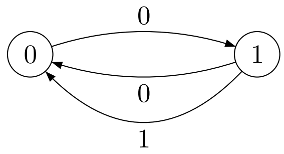

Fig. 1. An automaton presenting a periodic-finite-type shift X. The shift X admits the following list of periodic forbidden words, for $T = 2$, $\mathcal{F}^{(0)} = \{1\}, \mathcal{F}^{(1)} = \emptyset$.

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 [5]. 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 $\{0, 1 \ldots T - 1\}$, called the set of *phases*. The collection $\mathcal{F}$ is denoted by $\mathcal{F} = \{(w_1, n_1), \ldots, (w_{|\mathcal{F}|}, n_{|\mathcal{F}|})\}$ 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 1.

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

Constrained systems with unconstrained positions are defined in [3] as follows. Let S be a constrained system, T a positive period, and $U \subseteq \{0, \ldots, T - 1\}$, 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 [3]. 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 [3, 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 1: 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 $\mathcal{G}^{(k)} = \mathcal{G}'^{(k)}$ and $S^{\sigma}_{U,T} = \mathsf{X}_{\{\mathcal{G}, T\}}$.

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 be 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 2: 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 $0 \le i \le T-1$, $w \in \mathcal{F}^{(i)}$ 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 [7]). 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 [7]). 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 3: 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 $0 \le i \le T-1$.

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

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

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 [5] 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 III.

III. 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.* [7] 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 4, $\mathcal{D}(\mathcal{F})$ 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 $Q_k = \{(w,k) \mid w \text{ is a prefix of a word in } \mathcal{F}^{(k)}\}$,
- 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:

- $\mathsf{T} = \{((u,k), a, (v, k + r \bmod T)) \mid (u,k) \in Q_k \setminus F_k, a \in A, \text{ and } v \text{ is the longest suffix } (ua)[r\,..\,|ua|-1] \text{ of } ua \text{ such that } (v, k+r \bmod T) \in Q\}$, (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 1: 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 4: 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 1, $\delta(i,zu) = (v, r \bmod T)$, 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.

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 $0 \le k \le T-1$ (see Figure 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 $0 \le k \le T-1$, which justifies a specific treatment of the initial states in the algorithm. The failure function guarantees a good time complexity of the algorithm.

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, $0 \le k \le T-1$
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 $\mathcal{A} = (Q, A, i, Q \setminus F, \delta)$

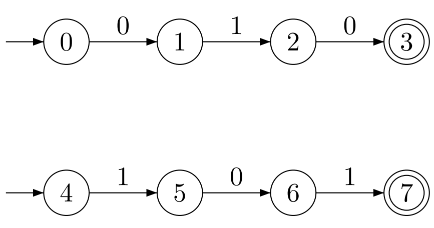

Fig. 2. Example of the two input tries for the collection $\mathcal{F}$ defined by $\mathcal{F}^{(0)} = \{010\}$, $\mathcal{F}^{(1)} = \{101\}$ and $T = 2$. Final states are doubled circled.

The shift $\mathsf{X}_{\{\mathcal{F},T\}}$, given in Figure 2, is presented by the deterministic automaton of Figure 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.

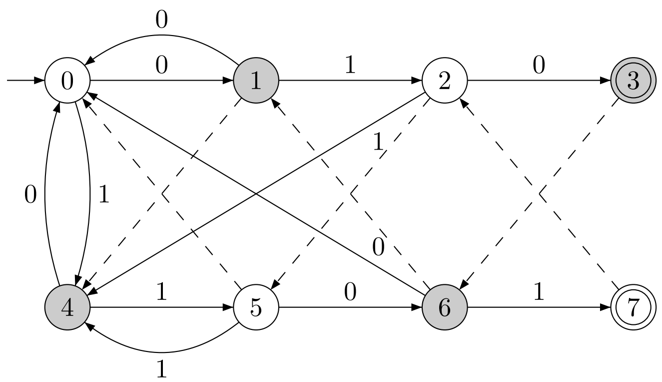

Fig. 3. Presentation of the shift $\mathsf{X}_{\{\mathcal{F},T\}}$, where $\{\mathcal{F}, T\}$ is defined by $\mathcal{F}^{(0)} = \{010\}$, $\mathcal{F}^{(1)} = \{101\}$ and $T = 2$.

Proposition 5: 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 $u[r \mathinner{..} |u|-1]$ 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 $(ua)[s \mathinner{..} |ua|-1]$ 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 $u'[r' \mathinner{..} |u'|-1]$ 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 $(v'a)[s' \mathinner{..} |v'a|-1]$ 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 $(u'a)[t \mathinner{..} |u'a|-1]$ of $u'a$ such that $(z, k + t \bmod T) \in Q$. Then $z = z'a$ and z' is a suffix $u'[t' \mathinner{..} |u'|-1]$ 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 $u[r \mathinner{.\,.} |u| - 1]$ 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 $(va)[s \mathinner{.\,.} |va| - 1]$ of va such that $(x, k + r + s \bmod T) \in Q$. Thus x is a suffix of ua. If y is a suffix $(ua)[t \mathinner{.\,.} |ua| - 1]$ of ua such that $(y, k+t \bmod T) \in Q$, then $y = y'a$ and y' is a suffix $u[t \mathinner{.\,.} |u| - 1]$ 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 $(ua)[t \mathinner{.\,.} |ua| - 1]$ 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 1: 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 6: 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 $0 \le k \le T - 1$.

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. ■

IV. PRESENTATION OF FINITE-MEMORY SYSTEMS WITH UNCONSTRAINED POSITIONS

In this section, we use results from Sections II and III 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 [3].

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 1 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 $\mathcal{T}_k$, $0 \le k \le T-1$, 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 III 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 1 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 T = (Q, A, i, F, δ),
     T' = (Q', A, i', F', δ'))
 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(T), (resp. l(T')) be the subtrie rooted at δ(i, 0)
    (respectively δ(i, 0))
 4. let r(T), (resp. r(T')) be the subtrie rooted at δ(i, 1)
    (respectively δ(i', 1))
 5. (such a subtrie is empty if the transition does not exist)
 6. set δ(i, 0) = PREFIX-FREE-UNION(l(T), l(T'))
 7. set δ(i, 1) = PREFIX-FREE-UNION(r(T), r(T'))
 8. return the trie T.
```

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

```
PERIODIC–TRIES(trie T = (Q, A, i, F, δ), integer T)
 1. make T copies T_k = (Q_k, A, i_k, F_k, δ_k) of T
 2. for each k ∈ {0, ..., T − 1}
 3.     for each state p of T_k at distance d from i_k
 4.     (for instance in a bottom-up order)
 5.         if (k + d mod T ∈ U) and p ∉ F_k
 6.             let l(T_k), (resp. r(T_k)) be the subtrie rooted
                by δ_k(p, 0) (resp. δ_k(p, 1)), eventually empty
                if the transition does not exist
 7.             remove δ_k(p, 0), if it exists
 8.             set δ_k(p, 1) = PREFIX-FREE-UNION(l(T_k), r(T_k))
 9. if k ∈ U, set δ_k(i_k, 0) = new sink state.
10. return the tries T_k
```

Proposition 7: 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 subtrie of p, *i.e.* the subtrie rooted by $\delta(p, 0)$. Similarly, we denote by $r(p)$ the (eventually empty) right subtrie 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 subtrie of a state p is small if it has the smallest size among the two subtries children of p. Then each state belongs to at most $\log_2 |Q|$ small subtries. 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 7 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 III 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 [3]. 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 [3] 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 [3, pp. 875]). Although our algorithm is polynomial and that given in [3] 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 [6]) 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 4. 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 7, $s = O(|Q|) = O(k)$. 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 [3]. Figure 5 displays an example for the constraint MTR(3). The presentation can be minimized with standard methods [9]. It leads to the minimal presentation displayed in Figure 6.

A condition similar to the gap condition of [3, 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

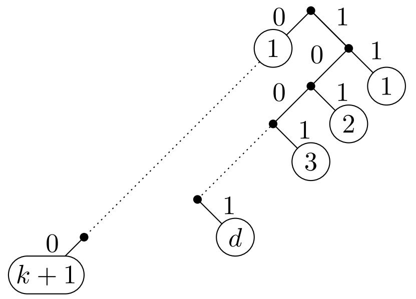

Fig. 4. The trie of the RLL (d, k)-constraint.

maximal length of a minimal forbidden word of the system. If this condition is satisfied, the complexity of our algorithm becomes linear, *i.e.* , $O(T \times |A| \times |\mathcal{T}|)$.

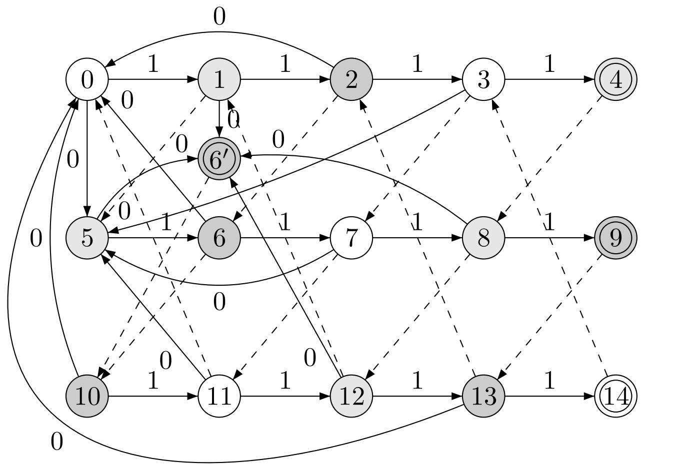

Fig. 5. A presentation of $S^{\sigma}_{U,T}$ for $S =$ MTR(3), $T = 3, U = \{1\}$. It is obtained by Algorithm PERIODIC–AUTOMATON on the input tries accepting the sets $\mathcal{G}^{(0)} = \{1111\}$, $\mathcal{G}^{(1)} = \{0, 1111\}$, $\mathcal{G}^{(2)} = \{1111\}$. States in phase 0, 1, and 2 are colored in white, light gray, and gray respectively.

V. PERIODIC FORBIDDEN WORDS OF A SINGLE FINITE WORD

Repetitions, and especially consecutive repetitions, play an important role in the analysis of molecular biology sequences. Some of them are even related to known deseases. 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.

In this section, we study the problem of computing the minimal periodic forbidden words of a given finite word. The problem of computing the minimal (non periodic) forbidden words of a single word has been solved in linear time in [7], see also [10, section 6.5 pp. 212]. We extend here the algorithm presented in [7] to the periodic case. The set of minimal forbidden words in a phase k of a word y, with $0 \leq k \leq T-1$, is the set of finite blocks w that never appear at a position $k \bmod T$ of y, and such that there is no strict factor w' of w with $w' \prec_i w$ appearing at a position $k + i \bmod T$ of y.

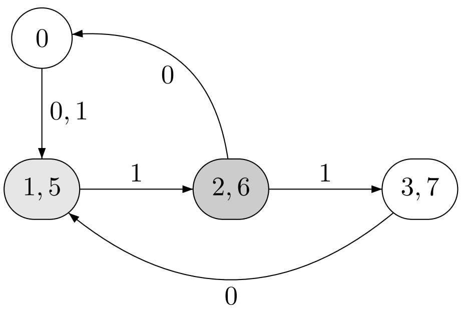

Fig. 6. The Shannon cover of $S^{\sigma}_{U,T}$ for $S =$ MTR(3), $T = 3$, $U = \{1\}$. It is the minimal presentation of that of Figure 5.

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

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

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

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

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

Thus, the above collection $\mathcal{MF}^{(k)}(y)$ is periodic and anti-factorial. It is minimal in the following sense: if $u \in \mathcal{F}^{(k)}(y)$, then u has a factor at some position i that belongs to $\mathcal{MF}^{(k+i \bmod T)}(y)$, and any other collection of finite sets of blocks $\mathcal{G}^{(k)}$ satisfying this condition verifies $\mathcal{MF}^{(k)}(y) \subseteq \mathcal{G}^{(k)}$. Although this notion of minimality refers to a finite word y, it is similar to the notion of periodic first offenders defined in [5] for constrained systems.

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

Proposition 8: The set $\mathcal{MF}^{(k)}(y)$ of minimal periodic forbidden words of y for a period T satisfies

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

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

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

We now describe an algorithm for computing the collection $\mathcal{MF}^{(k)}(y)$. The design of the algorithm is based on (1). A preliminary step of the algorithm consists in computing, for any $0 \leq k \leq T-1$, a minimal deterministic automaton accepting $\mathrm{Suff}^{(k)}(y)$. This operation can be performed in time $O(T \times |y| \times \log |A|)$.

First, the computation of a minimal deterministic automaton accepting $\mathrm{Suff}^{(k)}(y)$ is reduced to the computation of a minimal deterministic automaton accepting $\mathrm{Suff}^{(0)}(y[k\,..\,|y|-1])$. Hence, we will assume, without loss of generality, that $k = 0$. The computation of a minimal deterministic automaton accepting $\mathrm{Suff}^{(0)}(y)$ 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 [11], [12] or [10, section 5.4 pp. 179-192]).

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

Moreover, the automaton has a transition labelled by a from the class of a word u to the class of ua. If $u \in \mathrm{Fact}(y)$, 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 $F_y(v) \neq F_y(u)$. In this case, $F_y(u) \subseteq F_y(v)$.

It can be shown that if p is a state representing a class of congruent factors of y, and if u belongs to this class, all $s(u)$ belong to the same class of the congruence. Thus, one can define the *suffix link* of the state p, denoted by $s(p)$, as the class of $s(u)$.

In the description below, deterministic automata are denoted by (Q, A, i, F, δ), where Q is the set of states, A the alphabet, i the initial state, δ the partial transition function, and F the set of final states. A transition labelled by a from p to q is also denoted by the edge (p, a, q). The algorithm generating the minimal automaton accepting $\mathrm{Suff}^{(0)}(y)$ is described in Procedure PERIODIC-SUFFIX-AUTOMATON. It is an incremental algorithm that computes successively a minimal automaton accepting $\mathrm{Suff}^{(0)}(y[0\,..\,i])$, for i going from 0 to $|y|-1$. This procedure calls procedures EXTENSION and SPLIT. Procedure EXTENSION performs the transformations needed to get a minimal automaton accepting $\mathrm{Suff}^{(0)}(y[0\,..\,i])$ from a minimal automaton accepting $\mathrm{Suff}^{(0)}(y[0\,..\,i-1])$. 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 y, the non-dummy states of its suffix path are the final states of the automaton.

```
PERIODIC-SUFFIX-AUTOMATON (word y, 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 |y| − 1 do
 6.       p = EXTENSION(p, y_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 ≥ 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′
```

Proposition 9: Algorithm PERIODIC-SUFFIX-AUTOMATON computes the minimal deterministic automaton accepting $\mathrm{Suff}^{(0)}(y)$ 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 y (see [10, 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(y, T). Let us denote $y[0\,..\,i-1]$ 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 10: The size of the minimal automaton accepting $\mathrm{Suff}^{(0)}(y)$ for a given period T is linear in the size of y. Algorithm PERIODIC-SUFFIX-AUTOMATON runs in time linear in the size of y.

Proof: The proof is similar to the proof in the aperiodic case [10, section 5.4 pp. 192]. ■

Making all states final in the minimal deterministic automaton accepting $\mathrm{Suff}^{(k)}(y)$ gives a deterministic automaton accepting $\mathrm{Fact}^{(k)}(y)$. Note that this new automaton may not be the minimal one. An example is given in Figure 7.

We now describe the second step of the algorithm. We denote by $\mathcal{A}_k = (Q_k, A, i_k, Q_k, \delta_k)$ a deterministic automaton accepting $\mathrm{Fact}^{(k)}(y)$, that is the set of blocks, factors of y, 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 $\mathcal{MF}^{(k)}(y)$. An example of this computation is described in Figure 8.

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)
for each $a \in A$
  if $\delta_k(i_k, a)$ defined
    set $\delta(i_k, a) = \delta_k(i_k, a)$
    set $f(\delta(i_k, a)) = i_{k+1 \bmod T}$
  else
    set $\delta(i_k, a) =$ new sink
for each state $p \in Q_k$ in width-first search from $\cup_k\{i_k\}$
and each $a \in A$
  if $\delta_k(p, a)$ undefined **and** $\delta_{k+1 \bmod T}(f(p), a)$ defined
    set $\delta(p, a) =$ new sink
  else if $\delta_k(p, a) = q$ **and** q not already reached
    set $\delta(p, a) = q$
    set $f(\delta(p, a)) = \delta(f(p), a)$
return $(\mathcal{T}_k = (Q_k, A, i_k, \{\text{sinks}\}, \delta))_{0\le k\le T-1}$;

Proposition 11: Algorithm PERIODIC-MF-TRIES computes from the automata $\mathcal{A}_k$ accepting $\mathrm{Fact}^{(k)}(y)$ the set of tries accepting the minimal periodic forbidden words of y.

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 y (see [10, section 6.5 pp. 182] or [7]). ■

Proposition 12: Algorithm PERIODIC-SUFFIX-AUTOMATON followed by algorithm PERIODIC-MF-TRIES runs in time $O(|y| \times T \times \log|A|)$.

Proof: The complexity is straightforward. ■

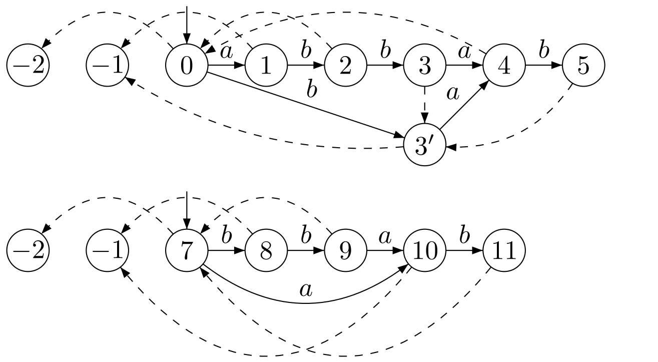

Fig. 7. Deterministic automata $\mathcal{A}_0$ and $\mathcal{A}_1$ accepting $\mathrm{Fact}^{(0)}(y)$ and $\mathrm{Fact}^{(1)}(y)$ respectively, with the period $T = 2$ and $y = abbab$. The suffix links are represented by dashed edges.

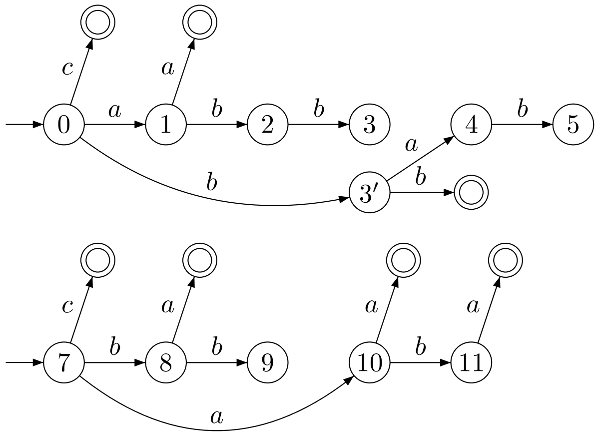

Fig. 8. Output tries $\mathcal{T}_0$ and $\mathcal{T}_1$ of Algorithm PERIODIC-MF-TRIES for the input tries $\mathcal{A}_0$ and $\mathcal{A}_1$ described in Figure 7. The sink states are double circled. The tries $\mathcal{T}_0$ and $\mathcal{T}_1$ accept $\mathcal{MF}^{(0)}(y) = \{c, aa, bb\}$ and $\mathcal{MF}^{(1)}(y) = \{c, ba, aa, aba\}$ respectively.

REFERENCES

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

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

[3] J. Campello de Souza, B. H. Marcus, R. New, and B. A. Wilson, "Constrained systems with unconstrained positions," *IEEE Trans. Inform. Theory*, vol. 48, no. 4, pp. 866–879, 2002.

[4] B. H. Marcus, R. M. Roth, and P. H. Siegel, "Constrained systems and coding for recording channels," in *Handbook of Coding Theory*, V. Pless and W. Huffman, Eds. North Holland, 1998, vol. II, ch. 20, pp. 1635–1764.

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

[6] M.-P. Béal, M. Crochemore, F. Mignosi, A. Restivo, and M. Sciortino, "Computing forbidden words of regular languages," *Fundamenta Informaticae*, vol. 56, no. 1,2, pp. 121–135, 2003.

[7] M. Crochemore, F. Mignosi, and A. Restivo, "Automata and forbidden words," *Inform. Process. Lett.*, vol. 67, pp. 111–117, 1998.

[8] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi, "Data compression using antidictionaries," in *Proceedings of the IEEE Lossless Data Compression*, J. Storer, Ed., 2000, pp. 1756–1768.

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

[10] M. Crochemore, C. Hancart, and Th. Lecroq, *Algorithmique du Texte*. Paris: Vuibert, 2001.

[11] 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.*, no. 21, pp. 12–20, 1983.

[12] ——, "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*. ACM Press, 1984, pp. 349–351.