Asynchronous sliding block maps

Marie-Pierre Béal
Institut Gaspard Monge,
Université de Marne-la-Vallée
http://www-igm.univ-mlv.fr/~beal/

Olivier Carton
Institut Gaspard Monge and CNRS
http://www-igm.univ-mlv.fr/~carton/

March 11, 1999

Abstract

We define a notion of asynchronous sliding block map that can be realized by transducers labeled in $A^* \times B^*$. We show that, under some conditions, it is possible to synchronize this transducer by state splitting, in order to get a transducer which defines the same sliding block map and which is labeled in $A \times B^k$, where k is a constant integer. In the case of a transducer with a strongly connected graph, the synchronization process can be considered as an implementation of an algorithm of C. Frougny and J. Sakarovitch of synchronization of rational relations of bounded delay. The algorithm can be applied in the case where the transducer has a constant integer transmission rate on cycles and has a strongly connected graph. It keeps the locality of the input automaton of the transducer. We show that the size of the sliding window of the synchronous local map grows linearly during the process, but that the size of the transducer is intrinsically exponential. In the case of non strongly connected graphs, the algorithm of C. Frougny and J. Sakarovitch does not keep the locality of the input automaton of the transducer. We give another algorithm to solve this case without losing the good dynamic properties that guarantees the state splitting process.

1 Introduction

We define a notion of asynchronous sliding block map. The classical notion of sliding block map is the class of maps from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$, where A and B

are finite alphabets, which are continuous and invariant by the shift transformation. The image of a bi-infinite sequence can be obtained by shifting a window of fixed length along the sequence. We extend this definition to asynchronous sliding block maps. These maps still use a sliding window but may output a variable number of symbols for each input symbol. These maps can be realized by automata labeled in $A \times B^*$, called transducers. Furthermore, the input automaton can be chosen local, that is, it admits at most one bi-infinite path labeled by a given bi-infinite word.

We study here the problem of the synchronization of these transducers, that is, the construction of a synchronous transducer defining the same map. A synchronous transducer is a transducer labeled in $A \times B^k$, where k is a positive integer. A synchronization of an asynchronous sliding block map is a synchronous sliding block map which defines the same map between orbits of bi-infinite sequences. The goal of the paper is to synchronize transducers while keeping the local property of its input automaton.

The question of the synchronization of transducers goes back to the paper of Elgot and Mezei [10] about rational relations realized finite automata, and the result of Eilenberg and Schützenberger [9] which states that a length preserving rational relation of $A^* \times B^*$ is a rational subset of $(A \times B)^*$, or, equivalently, is realized by a synchronous automaton (labeled in $A \times B$). The proof of Eilenberg is effective put is done on regular expressions and not directly on automata. In [11], C. Frougny and J. Sakarovitch give an algorithm of synchronization of relations with bounded length difference, the relations being between finite words or between one-sided infinite words. This constitutes another proof of the previous result. Their algorithm operates directly on the transducer that realizes the relation.

Here we consider bi-infinite sequences recognized by automata that are without any initial or final states (or, more precisely, with all states both initial and final). A transducer is synchronizable if it has a constant integer transmission rate on cycles.

We show that it is possible to synchronize transducers with a strongly connected graph labeled in $A^* \times B^*$, while keeping the locality of the input automaton of the transducer. At the heart of the construction is the state splitting process. It is moreover possible to do only output state splitting, or to do only input state splitting. A state splitting is a transformation of a graph which is a an automorphism between the symbolic dynamic subshifts defined by the graph before and after the transformation. The notion of state splitting, appeared early in information theory, has been introduced to symbolic dynamics by R. Williams. It has been since widely used, for example to solve some coding problems (see for instance [1], [15] and [13]).

A state splitting keeps good properties of an automaton like the property of being local. The synchronization algorithm that we describe in the case of strongly connected graphs can be considered as an implementation, which guarantees the claimed property, of the algorithm sketched in [11] in Section 10. We use the notion of balance of a state introduced in [11], which controls the lookahead, i.e the number of symbols read in output (up to a division by a constant k) minus the number of symbols read in input.

The main interest of this algorithm is that the size of the sliding window grows linearly during the process. However, we give an example of an asynchronous map realized by a transducer such that any synchronized transducer with a local input automaton that realizes it has an exponential number of states. The synchronization is therefore intrinsically exponential in the number of states.

In the last section, we extend the result to the more general case of transducers with non strongly connected graphs. We give two algorithms to treat this case. The first one supposes that the transducer satisfies a stronger synchronization hypothesis. It guarantees the locality of the input of the transducer. We also mention a second one which is an implementation of the algorithm sketched in [11]. This algorithm is of less interest since it does not guarantee the locality of the input of the transducer. This makes the recovering of the synchronous map, defined by the way of a sliding block window, much more difficult.

2 Asynchronous maps and transducers

2.1 Asynchronous and synchronous sliding block maps

Let A be an alphabet. A bi-infinite word of $A^{\mathbb{Z}}$ is a bi-infinite sequence $(a_i)_{i\in\mathbb{Z}}$ of letters of A. The *shift* σ is the continuous bijection from $A^{\mathbb{Z}}$ to $A^{\mathbb{Z}}$ defined by

$$\sigma((a_i)_{i\in\mathbb{Z}}) = (a_{i+1})_{i\in\mathbb{Z}}.$$

The orbit of a word $x \in A^{\mathbb{Z}}$ is the set $\{\sigma^n(x) \mid n \in \mathbb{Z}\}$ and is denoted by $\dot{x}$. Two words are in the same orbit if they differ only in some shifting of the indices. Thus, an orbit may be seen as a bi-infinite word without explicit indexing. The set of all orbits is denoted by ${}^{\omega}A^{\omega}$. An element of ${}^{\omega}A^{\omega}$ is also called a word. To avoid ambiguity in the sequel, we refer to a word of $A^{\mathbb{Z}}$ or to a word of ${}^{\omega}A^{\omega}$.

We now come to the definition of sliding block maps also called local maps in the literature. We first recall the classical definition of sliding block maps from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$. Then we give the definition of asynchronous and synchronous sliding block maps from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$. We finally explain the connections between these definitions.

A function f from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$ is a *sliding block map* if there are integers m, a, (m is the memory and a is the anticipation), and a function $\bar{f} : A^l \to B$, where $l = m + a$, such that for all $x \in A^{\mathbb{Z}}$, the image $y = f(x)$ of x is the bi-infinite word of $B^{\mathbb{Z}}$ defined by $y_n = \bar{f}(x_{n-(m-1)} \cdots x_{n+a})$ for all $n \in \mathbb{Z}$. Thus the letter y_n only depends on the finite block $x_{n-(m-1)} \cdots x_{n+a}$ of x. The integer $l = m + a$ is called the size of the so called sliding window. A sliding block map f commutes with the shift, i.e., satisfies $f \circ \sigma = \sigma \circ f$. Actually, a function from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$ is a sliding block map if and only if it is continuous and commutes with the shift.

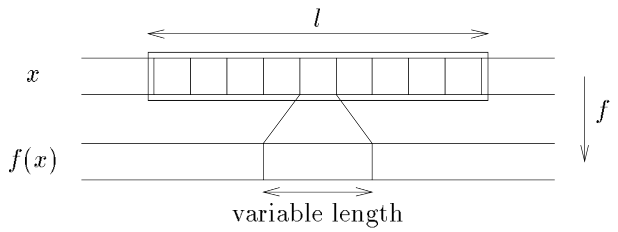

Figure 1: Asynchronous sliding block map

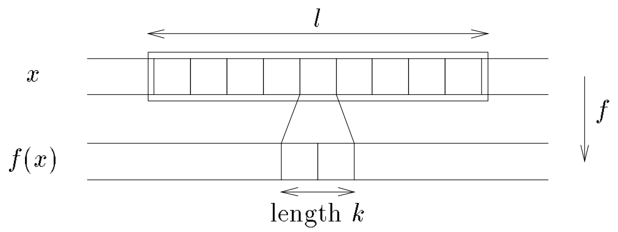

Figure 2: A k-synchronous sliding block map

A function f from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ is a *sliding block map* if there are an integer l and a function $\bar{f} : A^l \to B^*$ such that the image of a word x of ${}^{\omega}A^{\omega}$ is the concatenation of the finite words $\bar{f}(x_{n-(l-1)} \cdots x_n)$ for all $n \in \mathbb{Z}$ (see Figure 1). The integer l is also called the size of the sliding window. The function f is called a *k-synchronous sliding block map* if the function $\bar{f}$ is actually uniform, that is a function from A^l to B^k (see Figure 2) for some fixed integer k.

Both definitions are very similar but in the case of maps from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$, the image of a block by $\bar{f}$ is always a letter while it can be an arbitrary word in the case of functions from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$. If f is a sliding block map from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$ where $B = C^k$ for some integer k, it naturally induces a synchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}C^{\omega}$. This function maps a word $\dot{x}$ to the word obtained by concatenating the blocks of length k of the image of x by f. This function, which is also called f, is a k-synchronous sliding block map. Conversely, if f is a k-synchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}C^{\omega}$, it also defines a sliding block map from $A^{\mathbb{Z}}$ to $B^{\mathbb{Z}}$ where $B = C^k$.

If f is a sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$, there may exist several functions $\bar{f}$ and $\bar{f}'$ from A^l and $A^{l'}$ to B^* such that the image of a word x of ${}^{\omega}A^{\omega}$ is the concatenation of the finite words $\bar{f}(x_{n-(l-1)} \cdots x_n)$ or $\bar{f}'(x_{n-(l'-1)} \cdots x_n)$. In particular, one function, say $\bar{f}'$, may be uniform from $A^{l'}$ to B^k while the other, $\bar{f}$, may not be uniform.

The purpose of this paper is to explain how to find a uniform function $\bar{f}'$ which yields the same function f, when the asynchronous sliding block map f is described by a function $\bar{f}$ which is not uniform.

2.2 Transducers

In this section we consider automata labeled in $A^* \times B^*$. Automata considered in the literature are often labeled in $A \times B^*$ instead of $A^* \times B^*$ but most of the results that we present here does not require this assumption. The empty word is denoted by ε.

A *transducer* $\mathcal{T} = (V, E)$ is a finite state machine with V as set of vertices, and E as set of edges labeled in $A^* \times B^*$. Each edge is labeled by a pair of words whose first component is the input while the second is the output. Such a transducer defines a relation from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ made of all pairs (x, y) such that (x, y) is the label of a bi-infinite path of the transducer.

We always assume that transducers are ε-free, that is, have no $(\varepsilon, \varepsilon)$-labeled edges. Classical algorithms from automata theory are known to remove the $(\varepsilon, \varepsilon)$-labeled edges without changing the relation defined by the transducer (cf. [2]).

An automaton without any ε-transition is said to be (m, a)-*local*, where m and a are integers, iff two finite paths of length $n = m + a$ and with the same label: $((p_i, a_i, p_{i+1}))_{0 \le i \le n-1}$ and $((p'_i, a_i, p'_{i+1}))_{0 \le i \le n-1}$ satisfy $p_m = p'_m$. An automaton is said to be local if it is (m, a)-local for some m and a. This property is equivalent to the property of the existence of at most one bi-infinite path labeled by a given bi-infinite word. An automaton with

ε-transitions is said to be *local* if there is at most one bi-infinite path labeled by a given bi-infinite word.

The input automaton of the transducer is the automaton obtained by removing the second component of the edge label. The relation defined by a transducer might not be a function. This is however always true if the input automaton is a local automaton.

Each asynchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ can be defined by a transducer $\mathcal{T}$ labeled in $A \times B^*$. Let f be an asynchronous sliding block defined as above and let l be the size of its sliding window. Let m and a be non-negative integers such that $m + a = l - 1$. We define V as the set A^{l-1}. The edges of $\mathcal{T}$ are

$$(a_1 \ldots a_{m+a}) \xrightarrow{a_{m+1}|\bar{f}(a_1 \ldots a_{m+a} a_l)} (a_2 \ldots a_{m+a} a_l).$$

This transducer defines the function f. The input automaton of the transducer is actually a De Bruin graph. It is an (m, a)-local automaton.

For any asynchronous sliding block map, it is moreover possible to choose $m = l - 1$ and $a = 0$ in the definition of the transducer $\mathcal{T}$. In this case, the input automaton of $\mathcal{T}$ is a deterministic automaton.

Let k be a positive integer. A *k-synchronous transducer* is a transducer labeled in $A \times B^k$. On each edge, the number of output labels is k times the number of input ones: it has a constant transmission rate on each edge equal to the integer k. By the previous construction of the transducer $\mathcal{T}$, a k-synchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ can be defined by a k-synchronous transducer.

We have considered maps defined on ${}^{\omega}A^{\omega}$. Sometimes, maps are defined on the set $\dot{S}$ of orbits of a subshift of finite type S of $A^{\mathbb{Z}}$. A *subshift of finite type* is a subset of $A^{\mathbb{Z}}$ which can be characterized by a finite number of forbidden finite blocks. It is a closed subset of $A^{\mathbb{Z}}$ invariant by the shift σ. A subshift of finite type can be recognized by a local automaton. A canonical example of subshift of finite type is the set of bi-infinite paths of a finite automaton. It is included in $E^{\mathbb{Z}}$, where the alphabet E is the set of edges of the automaton. Equivalently, it is also the set of labels of bi-infinite paths of a finite automaton which has all its edges with distinct labels.

If f is an asynchronous sliding block map from $\dot{S}$ to ${}^{\omega}B^{\omega}$, it can be defined by an asynchronous transducer whose input automaton is a local automaton recognizing S.

Conversely any asynchronous transducer labeled in $A \times B^*$ with a local input automaton defines an asynchronous sliding block map. If the input automaton is (m, a)-local, one can define it with a sliding window of length

$m + a + 1$. If the transducer is k-synchronous, it defines a k-synchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$.

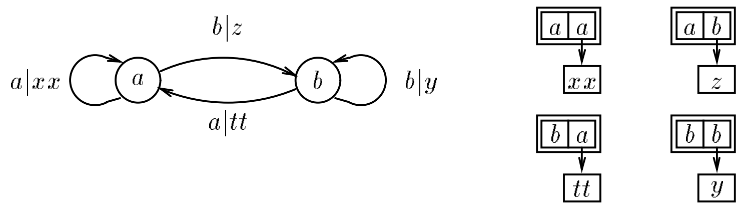

Figure 3: Asynchronous transducer and map

Example 1 Let $A = \{a, b\}$ and $B = \{x, y, z, t\}$. An example of an asynchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ is defined by the asynchronous transducer of Figure 3.

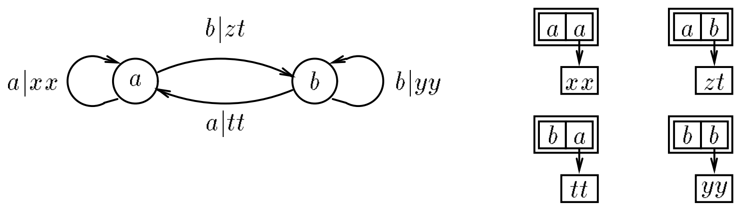

Figure 4: Synchronous transducer and map

Example 2 Let $A = \{a, b\}$ and $B = \{x, y, z, t\}$. An example of a 2-synchronous sliding block map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ is defined by the 2-synchronous transducer of Figure 4.

3 Synchronization of transitive transducers

In this section, we consider *transitive transducers*, that is transducers whose graphs are strongly connected. If the input automaton of the transducer is a local automaton, it recognizes a transitive shift of finite type. We describe an algorithm which synchronizes transducers with a constant transmission rate on cycles. This algorithms uses state splitting and thus keeps the local property of the input automaton. Non-transitive transducers are considered in Section 4.

3.1 Transmission rate

Let $\mathcal{T}$ be a transducer. We define the *transmission rate of a path* labeled by (u, v) as the ratio $|v|/|u|$. A transducer has a *constant transmission rate on cycles* if all cycles, that is paths beginning at and ending in a same state have the same transmission rate. This property can be checked on simple cycles only. A transducer has a *constant transmission rate on confluent paths* if for all states p, q, all paths beginning at p and ending in q have the same transmission rate (depending on p and q). If the transducer is transitive, a constant transmission rate on cycles is equivalent to a constant transmission rate on confluent paths.

We first give an algorithm to check if a transitive transducer has a constant integer transmission rate on confluent paths (or on cycles). This can be done by a depth first search. A first exploration can be done to find a cycle and get then an integer k candidate to be the constant transmission rate. We begin the exploration of the graph at some state i. We define a function balance from V to $\mathbb{Z}$. This function associates with any state q an integer balance(q) such that for any states p and q, the difference balance$(q) -$ balance(p) is equal to $|v| - k|u|$ for any path from p to q labeled (u, v). Since the the graph is strongly connected, this property defines the function balance up to an additive constant. The balances are completely defined if we fixed balance$(i) = 0$.

During the exploration of the graph, we can compute for each state q an integer balance(q) as follows:

- balance(i) is equal to 0
- balance(q) is equal to $|v| - k|u|$ for any path from i to q labeled by (u, v).

Here is the algorithm to compute the balances of the states. The main procedure BALANCE sets the value of balance(i) and the recursive function VISIT performs the depth first search. The boolean constant-rate initialized to TRUE indicates at the end if the transducer has a constant rate on cycles.

BALANCE
begin
  constant-rate:= TRUE ;
  for all states q **do** visited$[q]$:= FALSE ;
  balance$[i]$:= 0 ;
  VISIT(i) ;
end

```
VISIT(p)
begin
   visited[p] := TRUE ;
   for each edge (p, (u, v), q) do
       if visited[q] = FALSE then
           begin
           balance[q] := balance[p] + |v| - k|u| ;
           VISIT(q) ;
           end
       else if balance[q] ≠ balance[p] + |v| - k|u| then
           constant-rate := FALSE ;
end
```

The value of the balance is not important. Only the difference of two values is independent of the exploration order. If the transducer has n states and output labels of edges of length at most L, the difference of balances of any two states is bounded by Ln.

3.2 Description of the algorithm

We now describe the algorithm which synchronizes transitive transitive with a constant transmission rate on cycle. This algorithm uses state splitting that we now define. We first define the operation of *output state splitting* in a automaton $\mathcal{T} = (V, E)$. Let q be a vertex of $\mathcal{T}$ and let O (resp. I) be the set of edges going out of q (resp. coming in q). Let $O = O' + O''$ be a partition of O. The operation of *output state splitting* relative to the partition (O', O'') transforms $\mathcal{T}$ into the graph $\mathcal{T}' = (V', E')$ where $V' = (V \setminus \{q\}) \cup \{q', q''\}$ is obtained from V by splitting state q into two states q' and q'', and where E' is defined as follows.

- all edges of E that are not incident to q are left unchanged.
- we give to both q' and q'' copies of the input edges of q.
- we distribute the output edges of q between q' and q'' according to the partition of O into O' and O''. We denote U' and U'' the sets of output edges of q' and q'' respectively: $U' = \{(q', x, p) \mid (q, x, p) \in O'\}$ and $U'' = \{(q'', x, p) \mid (q, x, p) \in O''\}$, (see Figure 5).

The operation of input state splitting is obtained by reversing the roles played by input and output edges. It is well-known that if an automa-

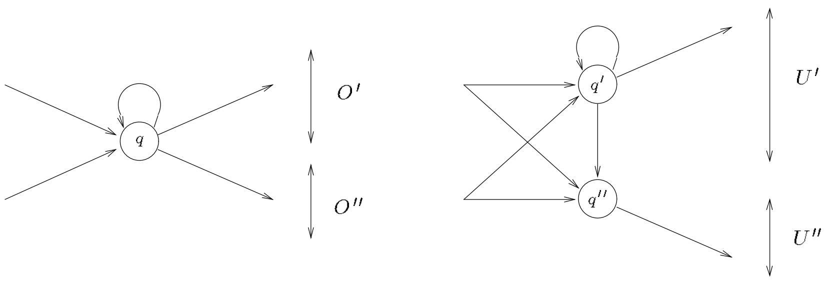

Figure 5: Output state splitting

ton is (m, a)-local, it is $(m, a+1)$-local after an output state splitting and $(m+1, a)$-local after an input state splitting. A deterministic (resp. codeterministic) automaton remains deterministic (resp. codeterministic) after an output (resp. input) splitting. The definitions can be generalized to define a multiple state splitting, when a state is split in more than two states according to a partition which has more than two parts.

We do input (resp. output) state splittings of states q of a transducer $\mathcal{T}$ only if the input (resp. output) edges of q have a non empty output labeling. An input state splitting of a state q is *admissible* if it is done according to a partition which is finer than the partition of the input edges defined by the *last* letter of their output label. An output state splitting of a state q is *admissible* if it is done according to a partition which is finer than the partition of the output edges defined by the *first* letter of their output label. Unless otherwise stated, we do admissible input (resp. output) state splitting corresponding to the partition defined by the last letter of the output label of input edges (resp. of the first letter of the output label of the output edges).

Examples of these two operations are described in Figure 6, where a, b and c are letters of B and u, u', v, v', w, w', r, r', t and t' are finite words of B^*. The state q is labeled by its balance, which remains unchanged after the transformation.

We now define another operation on a transducer $\mathcal{T}$. In order to synchronize the transducer, we are going to decrement or increment the balance of some states.

We first describe the operations called *incrementation* and *decrementation* in the case where all edges of the transducer are labeled by $A^+ \times B^*$. The general case is a bit more technical and it will described later. These two operations are local operations leaving the graph and the input labels

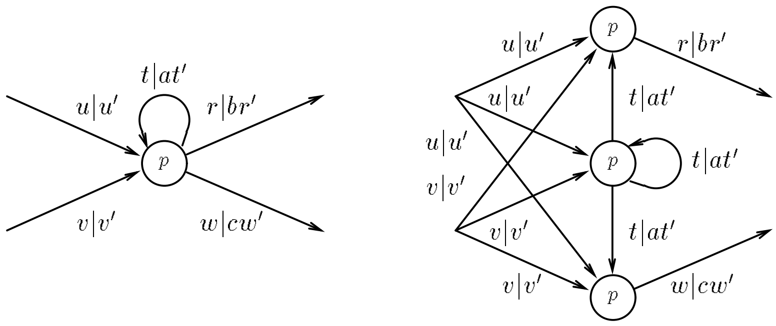

Figure 6: An admissible output state splitting

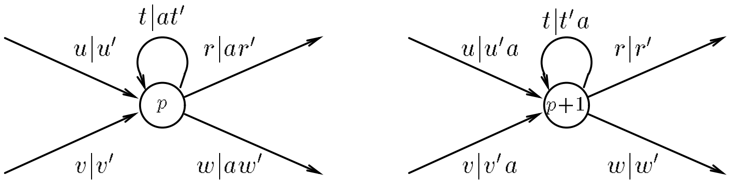

Figure 7: Incrementation of a state

unchanged. An incrementation of a state of balance p can be done if and only if all the output labels of its output edges begin with the same first letter. This letter is removed and put as last letter of the output label of the input edges. The balance is incremented by 1. The decrementation is defined similarly. The incrementation is illustrated in Figure 7. In the figure, the states are labeled with their balance.

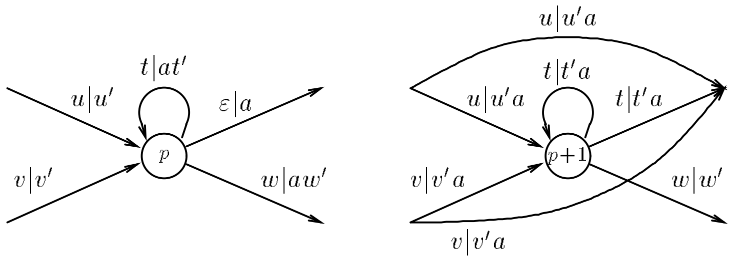

Figure 8: An ε-free incrementation

We now describe the incrementation in the case where some edges may be input labeled by ε. We have supposed that the transducer is ε-free but some $(\varepsilon, \varepsilon)$-labeled edges may appear in an incrementation made as described above. This can happen if an output edge is labeled by (ε, a). So, we describe a modified version which keeps the ε-free property of the transducer.

If an edge $(q, (\varepsilon, \varepsilon), q')$ appears in the incrementation of state q, it is removed and replaced by edges $(q'', (u, v), q')$, for each input edge $(q'', (u, v), q)$ of q. If the input automaton is local before the incrementation, it is still local after it. An ε-free incrementation is illustrated in Figure 8, where a is a letter of B and u, u', v, v', w, w', t and t' are finite words of B^*.

We now describe the synchronization algorithm by state splitting for a transitive transducer labeled in $A^* \times B^*$, and with a constant integer transmission rate k on cycles. A description of the input and output data is the following.

- INPUT:
  A transitive asynchronous transducer $\mathcal{T}$ labeled in $A^* \times B^*$ with a constant transmission rate k on cycles which defines an asynchronous sliding block map f from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$.

- OUTPUT:
  A transitive synchronous transducer $\mathcal{T}'$ labeled in $A \times B^k$. The transducer $\mathcal{T}'$ defines the same function f, which is k-synchronous. If the input automaton of $\mathcal{T}$ is local, the input automaton of $\mathcal{T}'$ is also local.

The transducer $\mathcal{T}'$ is obtained by state splitting. Furthermore, it is possible to do only output (resp. input) state splitting. Then, if the input automaton of $\mathcal{T}$ is deterministic (resp. codeterministic) and local, the input automaton of $\mathcal{T}'$ is also deterministic (resp. codeterministic) and local.

We denote DECREMENT(q) and INCREMENT(q) the procedures corresponding to the operations described above, applied to state q. We denote by INPUT-SPLIT$(q; q_1, q_2, \ldots, q_r)$ and OUTPUT-SPLIT$(q; q_1, q_2, \ldots, q_r)$ the corresponding procedures applied to a state q, split in states $q_1, q_2, \ldots, q_r$. The synchronization algorithm is the following.

SYNCHRONIZE$(\mathcal{T})$
begin
  for $i := L$ **downto** 1 **do**
    for each state q of balance i or $-i$ **do**
      if balance$(q) < 0$ **then**
        begin

```
      OUTPUT-SPLIT(q; q_1, q_2, ..., q_r) ;
      for all q_j (1 ≤ j ≤ r) do INCREMENT(q_j) ;
      end
    else if balance(q) > 0 then
      begin
      INPUT-SPLIT(q; q_1, q_2, ..., q_r) ;
      for all q_j (1 ≤ j ≤ r) do DECREMENT(q_j) ;
      end
end
```

The soundness of the algorithm is based on the following points:

- First, and this is the key point of the algorithm, a state with a negative balance, to be split and incremented, does not have outgoing edges with an empty output label. In fact, such an edge would arrive in a state with a strictly lower balance. This is not possible since states with lower balance are treated first. The same is true (mutatis mutandis) for states with positive balances.

- Second, decrementations (resp. incrementations) of states q or $q_1, q_2, \ldots, q_r$ are applied after an eventual admissible output (resp. input) state splitting, and they can actually be done. The transducer is synchronized when all balances are equal (to zero).

Remark 3 It is possible to synchronize the transducer by doing only output (or only input) state splittings. To do only output state splittings for example, we begin with a positive distribution of balances.

Remark 4 For each value of i of the outer loop, all states with balances equal to i (resp. $-i$) are split and decremented (resp. incremented). Actually all the splittings of these states are independent and can therefore be performed simultaneously. Such a step is called a *round of state splitting*. This also holds for incrementations and decrementations. When incrementations and decrementations are done simultaneously, the beginning and the end of the output labels can be modified in parallel since there is no concurrent write.

3.3 Evaluation of the complexity

In this section, we study the complexity of the procedure SYNCHRONIZE when the input automaton of the transducer is local. We first show that the size of the sliding window grows linearly. However, we exhibit examples showing that there is an exponential growth of the number of states.

This result can be compared to that obtained by J. Ashley in [4] (see also [3]) where he introduces a new construction of finite-state encoders for input constrained channels that guarantees an encoder with a window length that is linear in the number n of states of the smallest graph representing the constraint. His construction gives a specification of t rounds of state splitting to be performed on the graph, where t is linear in n, even if the number of states of the encoder is exponential. The same situation appears here: even if the number of states of the transducer that we get has an exponential number of states, it is possible to do a number of rounds of state splitting which is bounded by the maximal difference between the balances of the states. This result is interesting since the size of the window of the synchronized map that we get depends on the number of rounds of state-splitting that are performed, and not on the number of states of the transducer.

Let $\mathcal{T}$ be an asynchronous transducer whose input automaton is local. Let n be the number of states of $\mathcal{T}$. Let f be the asynchronous map from ${}^{\omega}A^{\omega}$ to ${}^{\omega}B^{\omega}$ defined by the transducer and let l be the size of its sliding window. It is known that $l = O(n^2)$ (see for example [5]). Let $\mathcal{T}'$ be the synchronized transducer. Let M be the maximal difference between the balances of states.

Proposition 5 *The size of the window of the synchronized map obtained is bounded by* $M + l$.

We point out that if the transducer $\mathcal{A}$ is labeled in $A \times B^*$ and the lengths of the output labels are bounded by K, the integer M is bounded by Kn. Indeed, the rate k is less than K, and for each edge $(p, (a, v), q)$, the difference between the balances of p and q is less than $|v| - k \leq K$.

Before proceeding to the proof of the proposition, we first state a lemma. Let M^+ be the maximum of positive balances of $\mathcal{T}$, and M^- the maximum of the absolute values of negative balances. Thus, the integer M is equal to $M^+ + M^-$.

Lemma 6 *If the input automaton of* $\mathcal{T}$ *is* (m, a)*-local, the input automaton of* $\mathcal{T}'$ *is* $(m + M^+, a + M^-)$*-local.*

Proof A single round of output state splitting increases the anticipation of a local automaton by at most one and a single round of input state splitting increases the memory of a local automaton by at most one. □

We now prove the proposition.

Proof We suppose that the input automaton of $\mathcal{T}$ is (m, a)-local. By the previous lemma, the input of $\mathcal{T}'$ is then a $(m+M^+, a+M^-)$-local automaton and the size of the window of the synchronized map is bounded by $M^+ + M^- + l = M + l$. $\square$

The following example shows that the number of states of the transducer grows exponentially when it is synchronized. It actually proves that this blow up is intrinsic to the synchronization. This does not depend of the algorithm used to construct the transducer.

Proposition 7 *There are n-state synchronizable transducers with a local input such that any synchronized transducer with a local input that defines the same map from* ${}^\omega A^\omega$ *to* ${}^\omega B^\omega$ *has an exponential number of states.*

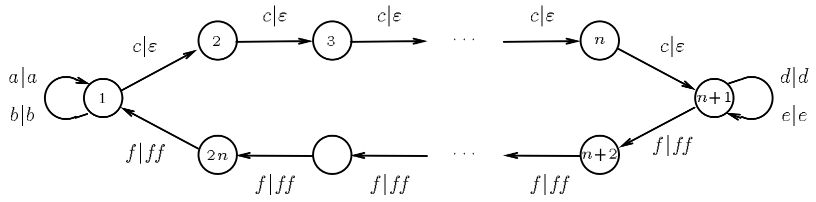

Figure 9: Transducer $\mathcal{T}$

Proof Let A be the alphabet $\{a, b, c, d, e, f\}$ and let us consider the transducer $\mathcal{T}$ of Figure 9. This transducer has $2n$ states and is synchronizable with rate 1. Let $\mathcal{T}'$ be any synchronized (or letter-to-letter) transducer with a local input automaton that defines the same map as $\mathcal{T}$ from ${}^\omega A^\omega$ to ${}^\omega A^\omega$.

We suppose that input automaton of $\mathcal{T}'$ is (m, a)-local. We can assume that $m = a$ and that m is greater than n. For each state q of $\mathcal{T}'$, we define the set E_q as the set of pairs (u_l, u_r) or words of length m such that $u_l u_r$ labels a path going through state q after reading u_l. Since the input automaton is (m, m)-local the sets E_q are pairwise disjoint. Furthermore, if both pairs (u_l, u_r) and (u'_l, u'_r) belong to E_q, both pairs (u_l, u'_r) and (u'_l, u_r) also belong to E_q.

Let w_l and w'_l be two different words of length $n/4$ over $\{a, b\}$ and let w_r and w'_r be two different words of length $n/4$ over $\{d, e\}$. Let us define the words u_l, u_r, u'_l and u'_r of length m by

$$u_l = a^{m-3n/4} w_l c^{n/2} \qquad u_r = c^{n/2} w_r d^{m-3n/4}$$
$$u'_l = a^{m-3n/4} w'_l c^{n/2} \qquad u'_r = c^{n/2} w'_r d^{m-3n/4}.$$

We suppose that both pairs (u_l, u_r) and (u'_l, u'_r) belong to E_q for some state q. There are four paths γ_1, γ_2, γ_3 and γ_4 as shown in Figure 10 where v_l, v_r,

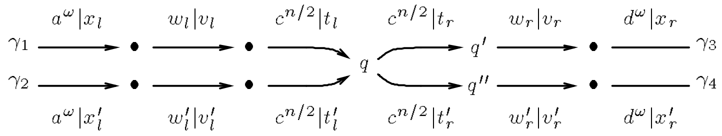

Figure 10: Paths γ_1, γ_2, γ_3 and γ_4

v'_l and v'_r are finite words of length $n/4$, t_l, t_r, t'_l and t'_r are finite words of length $n/2$, x_l and x'_l are left-infinite words and x_r and x'_r are right-infinite words. Paths γ_1 and γ_2 end in q while paths γ_3 and γ_4 start at q. Since the respective images by f of ${}^\omega a w_l c^n w_r d^\omega$, ${}^\omega a w'_l c^n w'_r d^\omega$, ${}^\omega a w_l c^n w'_r d^\omega$ and ${}^\omega a w'_l c^n w_r d^\omega$ are ${}^\omega a w_l w_r d^\omega$, ${}^\omega a w'_l w'_r d^\omega$, ${}^\omega a w_l w'_r d^\omega$ and ${}^\omega a w'_l w_r d^\omega$, we get the following equalities by considering the paths $\gamma_1\gamma_3$, $\gamma_2\gamma_4$, $\gamma_1\gamma_4$ and $\gamma_2\gamma_3$

$$x_l v_l t_l t_r v_r x_r = {}^\omega a w_l w_r d^\omega$$
$$x'_l v'_l t'_l t'_r v'_r x'_r = {}^\omega a w'_l w'_r d^\omega$$
$$x_l v_l t_l t'_r v'_r x'_r = {}^\omega a w_l w'_r d^\omega$$
$$x'_l v'_l t'_l t_r v_r x_r = {}^\omega a w'_l w_r d^\omega.$$

It follows that we have either:

$$t_r v_r x_r = x w_r d^\omega$$
$$t'_r v'_r x'_r = x w'_r d^\omega,$$

where x is a common suffix to w_l and w'_l, or:

$$x_l v_l t_l = {}^\omega a w_l y$$
$$x'_l v'_l t'_l = {}^\omega a w'_l y.$$

where y is a common prefix to w_r and w'_r. Since w_r and w'_r are different and the words w_l and w'_l are also different, the words t_r and t'_r must be different

different. This implies that the states q' and q'' are also different since the automaton is unambiguous (or lossless).

We finish the proof with a variant of the pigeon hole principle. Let us now choose $N = 2^{n/4}$ distinct words $w_{l,1}, v_{l,2}, \ldots w_{l,N}$ of length $n/4$ over $\{a, b\}$ and let us choose N distinct words $w_{r,1}, w_{r,2}, \ldots, w_{r,N}$ of length $n/4$ over $\{d, e\}$. Let $1 < \lambda < 2$. Let us assume that we have always less than $\lambda^{n/4}$ pairs $(w_{l,i}, w_{r,i})$ that belong to a same set E_q. Then the number of states of $\mathcal{T}'$ is $r > (2/\lambda)^{(n/4)}$, which is exponential. Otherwise, there is a state q such that E_q contains at least $\lambda^{n/4}$ pairs $(w_{l,i}, w_{r,i})$. We have proved that this implies that the $\lambda^{n/4}$ states q_{v_i} are all distinct. We find again an exponential number of states. □

3.4 Example

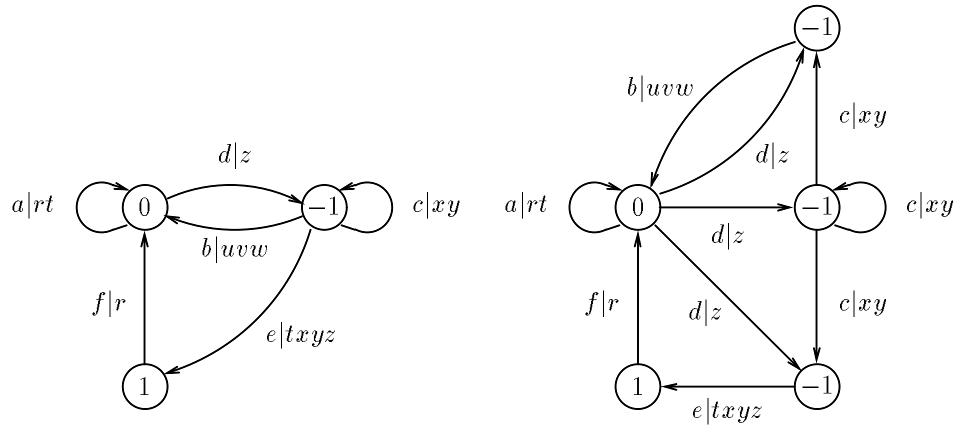

Figure 11: Transducer $\mathcal{T}$ and the output state splitting

We give an example of synchronization. We consider the transducer $\mathcal{T}$ pictured in the left of Figure 11. In the figure, each symbol represents one letter and states are labeled by their balance. This transducer is a candidate to be synchronized with $k = 2$. The state with balance -1 is output-split into three states (see the right of Figure 11). Then, each of the three new states are incremented and their balance becomes 0 (see left of Figure 12). Finally, the last state with balance 1 does not need to be split since it only has one incoming edge. It is just decremented. The synchronized transducer is pictured in the right of Figure 12). The input automaton is $(0, 1)$-local. The 2-synchronous sliding block map f defined has a window of length 2.

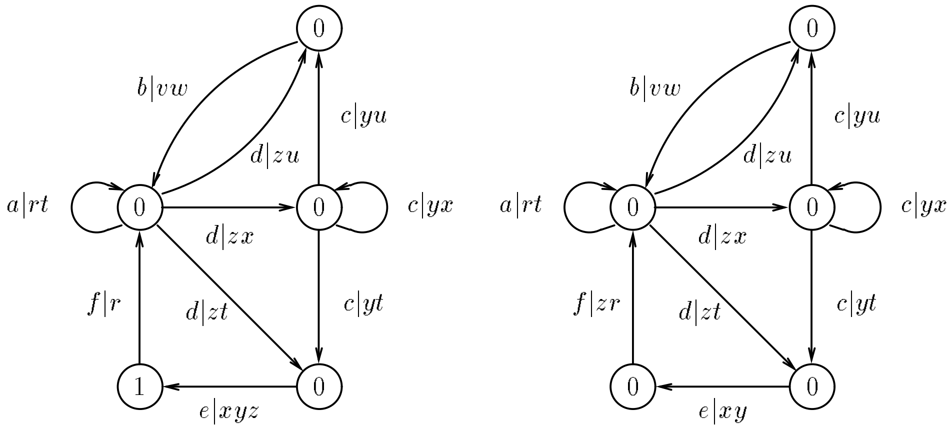

Figure 12: Incrementation and decrementation

4 Synchronization of non-transitive transducers

We finally consider the case of transducers labeled in $A^* \times B^*$ with a not necessarily strongly connected graph. We describe two algorithms that synchronize a non-transitive transducer. The first one needs a stronger hypothesis on the transducer but keeps the local property of the input automaton. The second one is an implementation of the algorithm given in [11]. Main (negative) point in that case is that the synchronization cannot be done in general without a step of duplication of states (which is not a state splitting process). This makes us loose the important property of locality of the input automaton of the transducer.

4.1 Method by state splitting

We give a new condition for a transducer to be synchronizable while keeping the local property of the input automaton of the transducer. As in the case of transitive transducer, we first suppose that the transducer has a constant transmission rate k on cycles. However, this condition is not sufficient for non-transitive transducers.

Let $\mathcal{T}$ be a transducer. An *undirected cycle* of $\mathcal{T}$ is a cycle in $\mathcal{T}$ viewed as an undirected graph. In such a cycle, each edge may be used in its usual direction or in the other direction. An undirected cycle is actually a cycle when all edges are used with the same orientation.

With each cycle c in the graph, we associate an integer $\mathrm{val}(c)$ called the valuation of the cycle and computed as follows. We fix some orientation for the cycle c and the valuation of the cycle is equal to the sum of the valuations of all edges of the cycle. The valuation of an edge $(p,(u,v),q)$ is equal to $|v| - k|u|$ if the orientation of the cycle coincide with those of the edge, and is equal to its opposite otherwise. The valuation of the cycle depends on the orientation chosen for the cycle.

It is well-known that the set of all undirected cycles of a graphs forms a vector space whose dimension is called the cyclomatic number of the graph. Our valuation is then a linear form from this vector space.

We suppose that the transducer $\mathcal{T}$ has a constant transmission rate k on cycles. The transducer $\mathcal{T}$ is said to have a *constant transmission rate on undirected cycle* if the following equality holds for any undirected cycle c

$$\mathrm{val}(c) = 0.$$

We first make some comments about this property. We first point out that if the transducer $\mathcal{T}$ is connected, it always has a constant transmission rate on undirected cycles if it already has a constant transmission rate k on cycles. Indeed, if the graph is connected, each undirected cycle can be decomposed as a sum of directed cycles. Second, it suffices to check this property on simple undirected cycles since the valuation is a linear form. This can be done by a straightforward adaptation of the algorithm BALANCE given in Section 3.1.

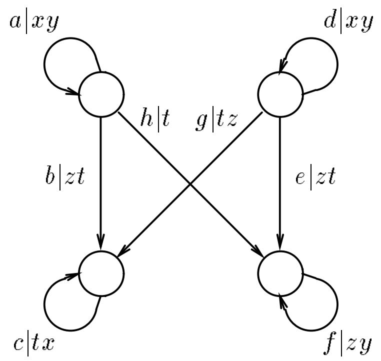

Figure 13: Transducer $\mathcal{T}$

Let γ and γ' are two paths from p to p' respectively labeled by $u|v$ and $u'|v'$. We can then consider the undirected cycle $\gamma\tilde{\gamma}'$ where $\tilde{\gamma}'$ is the path γ' in reverse direction. If the transducer has a constant transmission rate on undirected cycles, one has $|v| = |v'|$. However, the converse does not hold as shows the transducer pictured in Figure 13. This transducer has a constant transmission rate of 2 on cycles but it does not have a constant transmission rate on undirected cycles.

We claim that any transducer which has a constant transmission rate on undirected cycles can be synchronized using state splittings and incrementations and decrementations. We just sketch the procedure. It should be noticed that a constant transmission rate on undirected cycles is neither changed by a state splitting or by an incrementation. Thus, the property remains true along the procedure. The procedure treats successively each connected component of the transducer. A first connected component is synchronized using the algorithm SYNCHRONIZE given in Section 3.2. Then, at each step, a new connected component is also synchronized in the same way. However, it may happen that paths between the newly synchronized connected component and the already synchronized connected components do not have a transmission rate equal to k. In that case, all states of the treated connected component are split and incremented as many times as needed so that all paths have a transmission rate equal to k. The key point to be noticed is that the property of having a constant transmission rate on undirected cycles insures that all paths between the newly synchronized connected component and the old ones need the same number of incrementations. When all connected components have been treated that way, the transducer is completely synchronized.

4.2 Method by duplication

We still assume that the transducer $\mathcal{T} = (V, E)$ has a constant integer transmission rate k on cycles. In this case, the transducer can have a constant rate on cycles, without having a constant rate on confluent paths. Two paths with a same origin can arrive in a same state with a different balance.

To take this fact into account, a state can be duplicated as many times as the possible values of the balance given by all paths ending in this state. Only a finite number of duplications is needed since there is a finite number of simple paths in the graph. This is what is done in the algorithm of [11]. We choose a minimal set of states, I, from which is is possible to reach by a path any other state. We call them initial states. We assign a balance equal to zero to these initial states. We then define a new finite state transducer as the accessible part from I of the transducer whose states belong to $V \times \mathbb{Z}$, and whose set of edges is $\{((s,b),(u|v),(t,b+|v|-k|u|)) \mid (s,(u|v),t) \in E\}$. Here is the algorithm to compute the transducer. The main procedure

DUPLICATE calls the recursive procedure CONSTRUCT.

DUPLICATE
begin
  for each state $i \in I$ **do** CONSTRUCT$((i,0))$;
end

CONSTRUCT$((s,b))$
begin
  Create a state (s,b);
  for each edge $(s,(u|v),t)$ of $\mathcal{T}$ **do**
    begin
    if (state $(t, b+|v|-k|u|)$ has not been created) **then**
      CONSTRUCT$((t, b+|v|-k|u|))$;
    Create an edge $((s,b),(u|v),(t,b+|v|-k|u|))$;
    end
end

This new transducer, that we call the duplicated transducer, can be then synchronized by the procedure SYNCHRONIZE$(\mathcal{T})$ given in Section 3.2.

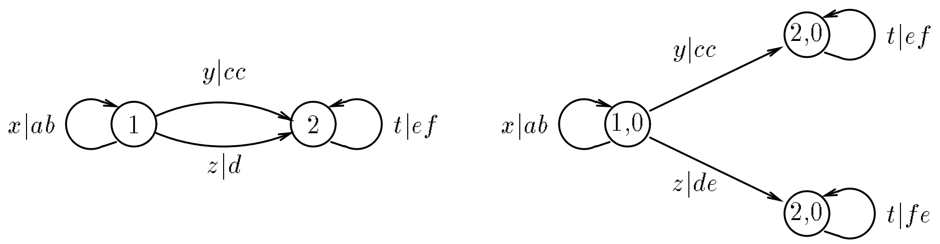

Figure 14: Transducer $\mathcal{T}$ and its synchronized version

Example 8 We consider the transducer pictured in the left part of Figure 14. The duplicated and synchronized transducer is pictured in the right part of the figure. The set I is equal to $\{1\}$. On can remark that the input automaton of the transducer is no more local since it has two loops labeled by t.

We now show that it is not possible to synchronize this transducer while keeping the local property of the input.

Let us assume that there is a transducer $\mathcal{T}'$ that represents the same map as $\mathcal{T}$ and which is both synchronous and local in input. Then $\mathcal{T}'$ is 2-synchronous, that is, labeled in $A \times B^2$. As its input is local, it admits a state p and a loop on p labeled either by $x|ab$ or by $x|ba$. It also admits a state q and a loop on q labeled either by $t|ef$ or by $t|fe$. As ${}^{\omega}xyt^{\omega}$ and ${}^{\omega}xzt^{\omega}$ are two input labels of bi-infinite paths of $\mathcal{T}'$, there are two finite paths going from p to q labeled $x^{n_1}yt^{n_2}$ and $x^{m_1}zt^{m_2}$, where n_1, n_2, m_1 and m_2 are nonnegative integers. This not possible since the output labels of the two edges of $\mathcal{T}$ labeled $(y|cc)$ and $(z|d)$ do not have the same length modulo 2.

References

[1] Adler, R. L., Coppersmith, D., and Hassner, M. Algorithms for sliding block codes. *IEEE Trans. Inform. Theory IT-29* (1983), 5–22.

[2] Aho, A., Sethi, R., and Ullman, J. *Compilers.* Addison-Wesley, 1986.

[3] Ashley, J. J. Factors and extensions of full shifts I. *IEEE Trans. Inform. Theory 34*, 3 (1988), 389–399.

[4] Ashley, J. J. A linear bound for sliding-block decoder window size, II. *IEEE Trans. Inform. Theory 42*, 6 (1996), 1913–1924.

[5] Béal, M.-P. *Codage Symbolique.* Masson, 1993.

[6] Béal, M.-P., and Perrin, D. Symbolic dynamics and finite automata. In *Handbook of Formal Languages*, G. Rosenberg and A. Salomaa, Eds., vol. 2. Springer, 1997, ch. 10.

[7] Berstel, J. *Transductions and Context-Free Languages.* Teubner, Stuttgart, 1979.

[8] Berstel, J., and Perrin, D. *Theory of Codes.* Academic Press, 1984.

[9] Eilenberg, S. *Automata, Languages and Machines*, vol. A. Academic Press, 1974.

[10] Elgot, C. C., and Mezei, J. E. On relations defined by generalized finite automata. *IBM Journal Res. and Dev. 9* (1965), 47–68.

[11] Frougny, C., and Sakarovitch, J. Synchronized relations of finite words. *Theoret. Comput. Sci. 108* (1993), 45–82.

[12] Frougny, C., and Sakarovitch, J. Synchronisation déterministe des automates à délai borné. *Theoret. Comput. Sci. 191* (1998), 61–77.

[13] Immink, K. A. S., Siegel, P. H., and Wolf, J. K. Codes for digital recorders. *IEEE Trans. Inform. Theory 44*, 6 (1998), 2260–2300.

[14] Lind, D., and Marcus, B. *An Introduction to Symbolic Dynamics and Coding.* Cambridge, 1995.

[15] Marcus, B. Factors and extensions of full shifts. *Monats. Math 88* (1979), 239–247.