# Simulation of effective subshifts by two-dimensional subshifts of finite type

Nathalie Aubrun
LIGM, Université Paris-Est
aubrun@univ-mlv.fr

Mathieu Sablik
LATP, Université de Provence
sablik@latp.univ-mrs.fr

**Abstract**

In this article we study how a subshift can simulate another one, where the notion of simulation is given by operations on subshifts inspired by the dynamical systems theory (factor, projective subaction...). There exists a correspondence between the notion of simulation and the set of forbidden patterns. The main result of this paper states that any effective subshift of dimension $d$ – that is a subshift whose set of forbidden patterns can be generated by a Turing machine – can be obtained by applying dynamical operations on a subshift of finite type of dimension $d+1$ – a subshift that can be defined by a finite set of forbidden patterns. This result improves Hochman's [Hoc09].

## Introduction

A subshift of dimension $d$ is a closed and shift-invariant subset of $\mathcal{A}^{\mathbb{Z}^d}$ where $\mathcal{A}$ is a finite alphabet. A subshift can be characterized by either its language or by a set of forbidden patterns. With this last point of view, the simplest class is the set of subshifts of finite type, which are subshifts that can be characterized by a finite set of forbidden patterns. It is possible to apply dynamical transformations like factor or projective subaction on a subshift of dimension $d$, and it seems natural to wonder how they modify the set of forbidden patterns.

In dimension 1, the class of subshifts of finite type is well understood. In particular subshifts of finite type are exactly those whose language is accepted by a local automaton [Bea93]. Given this result, we are naturally interested in subshifts with a language given by a finite automaton without the locality condition. This class is entirely characterized in terms of dynamical operations: it is the class of sofic subshifts, which can all be obtained as a factor of a subshift of finite type [LM95]. Thus each sofic subshift is obtained by a dynamical transformation of a subshift of finite type.

Multidimensional subshifts of finite type are not well understood. For example, it is not easy to describe their languages. Moreover, in addition to factors, there exist other types of dynamical transformations on multidimensional subshifts: for example a subaction of a $d$-dimensional subshift consists in taking the restriction of a subshift to a subgroup of $\mathbb{Z}^d$. Hochman [Hoc09] showed that every $d$-dimensional subshift whose set of forbidden patterns is recursively enumerable can be obtained by subaction and factor of a $d+2$-subshift of finite type. The main result of this article states that any effective subshift of dimension $d$ can be obtained from a SFT of dimension $d+1$, thanks to a subaction and a factor operation. This result improves Hochman's [Hoc09] since our construction decreases the dimension. This problem is referenced in [Boy08] and independently of this work there is solution at this problem in [DRS10].

The idea of the proof in [Hoc09] and in this article is to construct $\mathbf{T}_{\texttt{Final}}$, a three dimensional subshift of finite type in [Hoc09] (resp. a two-dimensional subshift of finite type in this paper), which realizes a given effective subshift $\Sigma \subset {\mathcal{A}_\Sigma}^{\mathbb{Z}}$ in one direction (assume that $d=1$) after a projection. Thanks to product operation, $\mathbf{T}_{\texttt{Final}}$ is constituted by different layers, the first one is constituted by the alphabet $\mathcal{A}_\Sigma$ and can be obtained by a projection $\pi$. Then finite type conditions ensure that for any $x \in \mathbf{T}_{\texttt{Final}}$, one has $\pi(x)_{\mathbb{Z}\times\{(i,j)\}} \in \Sigma$ (resp. $\pi(x)_{\mathbb{Z}\times\{i\}} \in \Sigma$) and all these lines are equal; moreover conditions are not so restrictive and any configuration of $\Sigma$ can be realized by a configuration of $\mathbf{T}_{\texttt{Final}}$. We here briefly present the main ideas of the proof, so that the reader already has in mind the final goal of technical constructions presented in this article. The difficulty is to ensure that no forbidden pattern in $\Sigma$ appears.

Since $\Sigma$ is an effective subshift, its forbidden patterns can be enumerated by a Turing machine. There are classical techniques to simulate calculations of a Turing machine thanks to a finite type condition (see Section 2.6) and the key point of these techniques is that calculations are embedded into finite computation zones. Thus, we consider a Turing machine $\mathcal{M}_{\texttt{Forbid}}$ which has a double role: it both enumerates the forbidden patterns of $\Sigma$ and checks that none of these patterns appear in a particular zone around each computation zone, called the responsibility zone. However, when a Turing machine is constructed in a two-dimensional subshift of finite type, as in this article, computation zones are such that a computation is made on a fractured tape (see Section 2.2). Consequently forbidden patterns produced by $\mathcal{M}_{\texttt{Forbid}}$ are also written on a fractured tape, and comparing them with non fractured patterns that appear in ${\mathcal{A}_\Sigma}^{\mathbb{Z}^2}$ is not trivial. To do so, the machine $\mathcal{M}_{\texttt{Forbid}}$ calls for a second Turing machine $\mathcal{M}_{\texttt{Search}}$ (see Section 3.5). This machine $\mathcal{M}_{\texttt{Search}}$ is given by $\mathcal{M}_{\texttt{Forbid}}$ an address located in its responsibility zone, and answers back the letter of $\mathcal{A}_\Sigma$ that appears at this address. If a forbidden pattern is detected, the machine $\mathcal{M}_{\texttt{Forbid}}$ comes into a special state $q_{\text{stop}}$, whose presence is forbidden is the final subshift. This ensures that every row $x_{\mathbb{Z}\times\{i\}}$ in the final subshift is a configuration of $\Sigma$. So, the two final operations one has to apply in order to obtain the subshift $\Sigma$ consist first in taking the projective subaction on $\mathbb{Z}e_1$, where $e_1$ is the first vector of the canonical basis of $\mathbb{Z}^2$, and then to erase any information that do not concern $\Sigma$ – for instance the construction of computation zones or the SFTs simulating the behaviour of Turing machines – thanks to a well-chosen letter-to-letter factor.

The difficulty of this construction presented in this paper is to program Turing machines with different size of computation which exchange information in a two-dimensional subshift of finite type, similar arguments can be found in [Dal74, Han74, DLS01]. We note that the authors of [DRS10] prove a similar result based on Kleene's fixed-point theorem. In that other proof, they do not recourse to geometric arguments to describe the circulation of information between the different levels of computation.

The paper is organized as follows: in Section 1 we present five types of operations (product, factor, finite type, projective subaction and spatial extension) and we formulate classic results with this formalism. In Section 2, we present an important tool to define runs of a Turing machine with a sofic subshift in dimension 2, which is the construction of an aperiodic SFT that will contain calculations of a Turing machine and how to code communication between those different calculations of a Turing machine. These tools are used to prove our main result in Section 3. The main construction of the proof of Theorem 3.1 is built step by step and for a better understanding, at the end of each of these subsections the contribution to the final construction is summed up in a fact. We do not pretend to give a formal proof for these facts, but we hope it will clarify our intention.

# 1 Subshifts and operations on them

In this section we recall some basic definitions on subshifts inspired from symbolic dynamics. We also present some dynamical operations on subshifts, that were first introduced by Hochman [Hoc09] and then developed by the authors in [AS09].

## 1.1 Tilings and subshifts

Let $\mathcal{A}$ be a finite alphabet and $d$ be a positive integer. A *configuration* $x$ is an element of $\mathcal{A}^{\mathbb{Z}^d}$. Let $\mathbb{S}$ be a finite subset of $\mathbb{Z}^d$. Denote $x_{\mathbb{S}}$ the *restriction* of $x$ to $\mathbb{S}$. A *pattern* is an element $p \in \mathcal{A}^{\mathbb{S}}$ and $\mathbb{S}$ is the *support* of $p$, which is denoted by $\text{supp}(p)$. For all $n \in \mathbb{N}$, we call $\mathbb{S}_n^d = [-n; n]^d$ the *elementary support* of size $n$. A pattern with support $\mathbb{S}_n^d$ is an *elementary pattern*. We denote by $\mathcal{E}_{\mathcal{A}}^d = \bigcup_{n\in\mathbb{N}} \mathcal{A}^{[-n;n]^d}$ the set of $d$-dimensional elementary patterns. A *$d$-dimensional language* $\mathcal{L}$ is a subset of $\mathcal{E}_{\mathcal{A}}^d$. A pattern $p$ of support $\mathbb{S} \subset \mathbb{Z}^d$ *appears* in a configuration $x$ if there exists $i \in \mathbb{Z}^d$ such that for all $j \in \mathbb{S}$, $p_j = x_{i+j}$, we denote $p \sqsubset x$.

**Definition.** A *co-tile set* is a tuple $\tau = (\mathcal{A}, d, P)$ where $P$ is a subset of $\mathcal{E}_{\mathcal{A}}^d$ called the *set of forbidden patterns*.

A *generalized tiling* by $\tau$ is a configuration $x$ such that for all $p \in P$, $p$ does not appear in $x$. We denote by $\mathbf{T}_\tau$ the set of generalized tilings by $\tau$. If there is no ambiguity on the alphabet, we just denote it by $\mathbf{T}_P$.

*Remark.* If $P$ is finite, it is equivalent to define a generalized tiling by allowed patterns or forbidden patterns, the latter being the usual definition of tiling.

One can define a topology on $\mathcal{A}^{\mathbb{Z}^d}$ by endowing $\mathcal{A}$ with the discrete topology, and considering the product topology on $\mathcal{A}^{\mathbb{Z}^d}$. For this topology, $\mathcal{A}^{\mathbb{Z}^d}$ is a compact metric space on which $\mathbb{Z}^d$ acts by translation via $\sigma$ defined by:

$$\begin{array}{rccll} \sigma^i_{\mathcal{A}}: & \mathcal{A}^{\mathbb{Z}^d} & \longrightarrow & \mathcal{A}^{\mathbb{Z}^d} & \\ & x & \longmapsto & \sigma^i_{\mathcal{A}}(x) & \text{such that } \sigma^i_{\mathcal{A}}(x)_u = x_{i+u}\ \forall u \in \mathbb{Z}^d. \end{array}$$

for all $i$ in $\mathbb{Z}^d$. This action is called the *shift*.

**Definition.** A $d$-dimensional subshift on the alphabet $\mathcal{A}$ is a closed and $\sigma$-invariant subset of $\mathcal{A}^{\mathbb{Z}^d}$. We denote by $\mathcal{S}$ (resp. $\mathcal{S}_d$, $\mathcal{S}_{\leq d}$) the set of all subshifts (resp. $d$-dimensional subshifts, $d'$-dimensional subshifts with $d' \leq d$).

Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift. Denote $\mathcal{L}_n(\mathbf{T}) \subseteq \mathcal{A}^{[-n;n]^d}$ the set of elementary patterns of size $n$ which appear in some element of $\mathbf{T}$, and $\mathcal{L}(\mathbf{T}) = \bigcup_{n\in\mathbb{N}} \mathcal{L}_n(\mathbf{T})$ the *language* of $\mathbf{T}$ which is the set of elementary patterns which appear in some element of $\mathbf{T}$.

It is also usual to study a subshift as a dynamical system [LM95, Kit98], the next proposition shows the link between the two notions.

**Proposition 1.1.** *The set $\mathbf{T} \subset \mathcal{A}^{\mathbb{Z}^d}$ is a subshift if and only if $\mathbf{T} = \mathbf{T}_{\mathcal{L}(\mathbf{T})^c}$ where $\mathcal{L}(\mathbf{T})^c$ is the complement of $\mathcal{L}(\mathbf{T})$ in $\mathcal{E}^d_{\mathcal{A}}$.*

A set of patterns $P \subseteq \mathcal{E}^d_{\mathcal{A}}$ is *recursively enumerable* if there exists an effective procedure for listing the patterns of $P$ (see for instance [RJ87]).

**Definition.** It is possible to define different classes of subshifts according to the set of forbidden patterns:

- For a finite alphabet $\mathcal{A}$ and a dimension $d \in \mathbb{N}$, the subshift $\mathbf{T}_{(\mathcal{A},d,\emptyset)} = \mathcal{A}^{\mathbb{Z}^d}$ is the *full-shift* of dimension $d$ associated to $\mathcal{A}$. Denote $\mathcal{FS}$ the set of all full-shifts (for every finite alphabet $\mathcal{A}$ and dimension $d$).
- For a finite alphabet $\mathcal{A}$, a dimension $d \in \mathbb{N}$ and a finite set $P \subseteq \mathcal{E}^d_{\mathcal{A}}$, the subshift $\mathbf{T}_{(\mathcal{A},d,P)}$ is a *subshift of finite type*. Denote $\mathcal{SFT}$ the set of all subshifts of finite type. Subshifts of finite type correspond to the usual notion of tiling.
- For a finite alphabet $\mathcal{A}$, a dimension $d \in \mathbb{N}$ and a recursively enumerable set $P \subseteq \mathcal{E}^d_{\mathcal{A}}$, the subshift $\mathbf{T}_P$ is an *effective subshift*. Denote $\mathcal{RE}$ the set of all effective subshifts.

## 1.2 Operations on subshifts

In this section we describe five operations on subshifts and use them to define a notion of simulation of a subshift by another one. Operations are gathered in two groups depending on which part – the alphabet $\mathcal{A}$ or the group $\mathbb{Z}^d$ – of a subshift $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ they modify.

### 1.2.1 Simulation of a subshift by another one

An *operation op* on subshifts transforms a subshift or a $n$-tuple of subshifts into another one; it is a function $op : \mathcal{S} \to \mathcal{S}$ or $op : \mathcal{S} \times \cdots \times \mathcal{S} \to \mathcal{S}$ that can depend on a parameter. An operation is not necessarily defined for all subshifts. We remark that a subshift $\mathbf{T}$ (resp. a pair of subshifts $(\mathbf{T}', \mathbf{T}'')$) and its image by an operation $op(\mathbf{T})$ (resp. $op(\mathbf{T}', \mathbf{T}'')$) do not necessary have the same alphabet or dimension.

Let $Op$ be a set of operations on subshifts. Let $\mathcal{U} \subset \mathcal{S}$ be a set of subshifts. We define the *closure* of $\mathcal{U}$ under a set of operations $Op$, denoted by $\mathcal{C}l_{Op}(\mathcal{U})$, as the smallest set stable by $Op$ which contains $\mathcal{U}$.

We say that a subshift $\mathbf{T}$ *simulates* a subshift $\mathbf{T}'$ by $Op$ if $\mathbf{T}' \in \mathcal{C}l_{Op}(\mathbf{T})$. Thus there exists a finite sequence of operations chosen among $Op$, that transforms $\mathbf{T}$ into $\mathbf{T}'$. We note it by $\mathbf{T}' \leq_{Op} \mathbf{T}$. Remark that $\mathcal{C}l_{Op}(\mathbf{T}) = \{\mathbf{T}' \ : \ \mathbf{T}' \leq_{Op} \mathbf{T}\}$.

### 1.2.2 Local transformations

We describe three operations that locally modify a subshift $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$. The new subshift resulting from the operation will be a subset of $\mathcal{B}^{\mathbb{Z}^d}$, where $\mathcal{B}$ is a new alphabet.

**Product (Prod):** Let $\mathbf{T}_i \subseteq \mathcal{A}_i^{\mathbb{Z}^d}$ for any $i \in \{1, \dots, n\}$ be $n$ subshifts of the same dimension, define:

$$\mathbf{Prod}\left(\mathbf{T}_1, \dots, \mathbf{T}_n\right) = \mathbf{T}_1 \times \cdots \times \mathbf{T}_n \subseteq \left(\mathcal{A}_1 \times \cdots \times \mathcal{A}_n\right)^{\mathbb{Z}^d}.$$

One has $\mathcal{C}l_{\mathbf{Prod}}(\mathcal{FS}) = \mathcal{FS}$ and $\mathcal{C}l_{\mathbf{Prod}}(\mathcal{SFT}) = \mathcal{SFT}$.

**Finite type (FT):** These operations consist in adding a finite number of forbidden patterns to the initial subshift. Formally, let $\mathcal{A}$ be an alphabet, $P \subseteq \mathcal{E}_{\mathcal{A}}^d$ be a finite subset and let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift. By Proposition 1.1, there exists $P'$ such that $\mathbf{T} = \mathbf{T}_{P'}$. Define:

$$\mathbf{FT}_P\left(\mathbf{T}\right) = \mathbf{T}_{P \cup P'}.$$

Note that $\mathbf{FT}_P\left(\mathbf{T}\right)$ could be empty if $P$ prohibits too many patterns. By **FT**, one lists all operations on subshifts which are obtained by this type of transformation.

By definition of subshift of finite type, one has $\mathcal{C}l_{\mathbf{FT}}(\mathcal{FS}) = \mathcal{SFT}$ and $\mathcal{C}l_{\mathbf{FT}}(\mathcal{FS}) = \mathcal{SFT}$.

**Factor (Fact):** These operations allow to change the alphabet of a subshift by local modifications. Let $\mathcal{A}$ and $\mathcal{B}$ be two finite alphabets. A *morphism* $\pi : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{B}^{\mathbb{Z}^d}$ is a continuous function which commutes with the shift action (i.e. $\sigma^i \circ \pi = \pi \circ \sigma^i$ for all $i \in \mathbb{Z}^d$). In fact, such a function can be defined locally [Hed69]: that is to say, there exists $\mathbb{U} \subset \mathbb{Z}^d$ finite, called *neighborhood*, and $\overline{\pi} : \mathcal{A}^{\mathbb{U}} \to \mathcal{B}$, called *local function*, such that $\pi(x)_i = \overline{\pi}(\sigma^i(x)_{\mathbb{U}})$ for all $i \in \mathbb{Z}^d$.

Let $\pi : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{B}^{\mathbb{Z}^d}$ be a factor and $\mathbf{T} \subset \mathcal{A}^{\mathbb{Z}^d}$ be a subshift, define:

$$\mathbf{Fact}_\pi\left(\mathbf{T}\right) = \pi(\mathbf{T}).$$

By **Fact**, one lists all operations on subshifts which are obtained by this type of transformation.

Example 1.1 shows that $\mathcal{C}l_{\mathbf{Fact}}(\mathcal{SFT}) \neq \mathcal{SFT}$.

*Example* 1.1 ($\mathcal{C}l_{\mathbf{Fact}}(\mathcal{SFT}) \neq \mathcal{SFT}$). Consider the alphabet $\{0,1,2\}^{\mathbb{Z}}$ and define $\mathbf{T} = \mathbf{T}_{\{00,11,02,21\}}$. The factor $\pi$ such that $\pi(0) = \pi(1) = 0$ and $\pi(2) = 2$ transforms $\mathbf{T}$ into a subshift:

$$\pi(\mathbf{T}) = \{x \in \{0,2\}^{\mathbb{Z}} \ : \ \text{finite blocks of consecutive 0 are of even length } \}$$

which is called the *even shift*. It is known that the even shift is not a subshift of finite type (see Example 2.1.9 of [LM95]), since one need to exclude arbitrarily large blocks of consecutive 0's of odd lengths to describe it.

**Definition.** A sofic subshift is a factor of a subshift of finite type. Thus, the set of sofic subshifts is $\mathcal{S}ofic = \mathcal{C}l_{\mathbf{Fact}}(\mathcal{SFT})$.

In [LM95], it is shown that sofic subshifts of dimension 1 are subshift which can be defined with a language of forbidden patterns which is regular. The characterisation is unknown for multidimensional sofic subshifts.

### 1.2.3 Transformations of the group of the action

We describe an operation that modify the group on which the subshift is defined, thus we change the dimension of the subshift.

**Projective Subaction (SA):** These operations allow to take the restriction of a subshift of $\mathcal{A}^{\mathbb{Z}^d}$ according to a subgroup of $\mathbb{Z}^d$. Let $\mathbb{G}$ be a sub-group of $\mathbb{Z}^d$ freely generated by $u_1, u_2, \ldots, u_{d'}$ $(d' \leq d)$. Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift, define:

$$\mathbf{SA}_{\mathbb{G}}(\mathbf{T}) = \left\{ y \in \mathcal{A}^{\mathbb{Z}^{d'}} \ : \ \exists x \in \mathbf{T} \text{ such that } \forall i_1, \ldots, i_{d'} \in \mathbb{Z}^{d'}, y_{i_1,\ldots,i_{d'}} = x_{i_1 u_1 + \cdots + i_{d'} u_{d'}} \right\}.$$

It is easy to prove that $\mathbf{SA}_{\mathbb{G}}(\mathbf{T})$ is a subshift of $\mathcal{A}^{\mathbb{Z}^{d'}}$. One denotes by $\mathbf{SA}$ the set of all operations on subshifts which are obtained by this type of operation.

One verifies that $\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT}) \neq \mathcal{SFT}$ and $\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT}) \neq \mathcal{S}ofic$ (see respectively Example 1.2 and Example 1.3).

*Example* 1.2 ($\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT}) \neq \mathcal{SFT}$). We construct a subshift of finite type $\mathbf{T} \subset \{0,1,2\}^{\mathbb{Z}^2}$ such that the projective subaction of $\mathbf{T}$ on the sub-group $\Delta = \{(x,y) \in \mathbb{Z}^2 : y = x\} \subseteq \mathbb{Z}^2$ is not of finite type. In this example we want the subshift that appears on $\Delta$ to be

$$\left\{ x \in \{0,1,2\}^{\mathbb{Z}} \ : \ \text{finite blocks of consecutive 0's are of even length} \right\}.$$

Define $\overline{F}$ the following set of allowed patterns of size 4 (. symbol may be 1 or 2 but not 0, blank symbol may be 0,1 or 2):

| | | | |
|---|---|---|---|
| | | 2 | 0 |
| | 1 | 0 | 1 |
| 2 | 0 | 2 | |
| 0 | 1 | | |

;

| | | | |
|---|---|---|---|
| | | . | . |
| | 2 | 0 | . |
| 1 | 0 | 1 | |
| 0 | 2 | | |

;

| | | | |
|---|---|---|---|
| | | 1 | 0 |
| | 2 | 0 | 2 |
| . | 0 | 1 | |
| . | . | | |

;

| | | | |
|---|---|---|---|
| | | . | |
| | . | . | . |
| 2 | 0 | . | |
| 0 | 1 | | |

;

| | | | |
|---|---|---|---|
| | | . | |
| | . | | . |
| . | . | . | |
| 0 | . | | |

;

| | | | |
|---|---|---|---|
| | | 2 | 0 |
| | . | 0 | 1 |
| . | . | . | |
| | . | | |

;

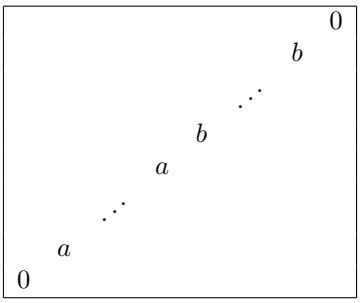
| | | | |
|---|---|---|---|
| | | . | 0 |
| | . | . | . |
| . | | . | |
| | . | | |

The alternation of 1 and 2 over and under the diagonal of 0 enables us to control the parity of 0 blocks. Define $F$ as the set of elementary patterns of size 4 that are not in $\overline{F}$. Then if we denote $\mathbf{T} = \mathbf{T}_F$:

$$\mathbf{SA}_{\Delta}(\mathbf{T}) = \left\{ x \in \{0,1,2\}^{\mathbb{Z}} \ : \ \text{blocks of consecutive 0's are of even length} \right\}$$

which is not a subshift of finite type as explained in Example 1.1.

*Example* 1.3 ($\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT}) \neq \mathcal{S}ofic$). The non finite type subshift constructed in Example 1.2 is sofic, but it is possible to obtain non sofic subshifts. We construct a subshift of finite type $\mathbf{T}$ such that the projection $\mathbf{SA}_{\Delta}(\mathbf{T})$ on the straight line $y = x$ is not sofic. It is well known that in dimension 1, sofic subshift are exactly subshifts whose language — see Definition 1.1 — is a regular language [LM95]. The language $\{a^n b^n : n \in \mathbb{N}\}$ is non-regular and so we construct a subshift of finite type $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^2}$ and a morphism $\pi : \mathcal{A}^{\mathbb{Z}^2} \to \{0,a,b\}^{\mathbb{Z}^2}$ such that the only allowed patterns in $\mathbf{T}' = \pi(\mathbf{T})$ containing finite blocks of consecutive $a$'s or $b$'s are those of the form $2n \times 2n$:




The principle is to construct patterns of even size and to localize the center of these patterns to distinguish the $a^n$ part from the $b^n$ part.

Denote $\mathcal{A} = \{*, a, b, 0, 1, 2, 3, 4\}$. We construct squares formed by any symbols except the symbol 0 which forms a background.The symbols $1, 2, 3$ and 4 help to draw the two diagonals of the square and to

distinguish in which quadrant we are. The symbol $*$ only appears on a diagonal of the square, and the other diagonal contains the $a^n b^n$ part. The presence of the symbol 0 everywhere around a finite figure ensures that the two diagonals cross in their middle, hence the figure pictured is a square. It is possible to describe a finite set of patterns where the only finite figures on the background formed by 0's which are allowed are even size squares of the form:

$$\begin{array}{|cccccccc|}\hline 0&0&0&0&0&0&0&0\\ 0&*&1&\dots&\dots&1&b&0\\ 0&4&\ddots&1&1&\iddots&2&0\\ 0&\vdots&4&*&b&2&\vdots&0\\ 0&\vdots&4&a&*&2&\vdots&0\\ 0&4&\iddots&3&3&\ddots&2&0\\ 0&a&3&\dots&\dots&3&*&0\\ 0&0&0&0&0&0&0&0\\ \hline\end{array} \quad (*)$$

We do not detail the entire set of allowed patterns, but the reader can easily deduce the missing patterns from those given below:

Squares center:
$$\begin{array}{|cccc|}\hline *&1&1&b\\ 4&*&b&2\\ 4&a&*&2\\ a&3&3&*\\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 0&0&0&0\\ 0&*&b&0\\ 0&a&*&0\\ 0&0&0&0\\ \hline\end{array}$$

Squares diagonals:
$$\begin{array}{|ccc|}\hline *&1&1\\ 4&*&1\\ 4&4&*\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 1&1&b\\ 1&b&2\\ b&2&2\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline *&2&2\\ 3&*&2\\ 3&3&*\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 4&4&a\\ 4&a&3\\ a&3&3\\ \hline\end{array}$$

Squares sides:
$$\begin{array}{|ccc|}\hline 0&0&0\\ 0&*&1\\ 0&4&*\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 0&0&0\\ 1&1&1\\ *&1&1\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 0&0&0\\ 1&1&1\\ 1&1&1\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 0&0&0\\ 1&1&1\\ 1&1&b\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 0&0&0\\ 1&b&0\\ b&2&0\\ \hline\end{array}$$

... and so on for the three other sides.

Uniform domains:
$$\begin{array}{|ccc|}\hline 0&0&0\\ 0&0&0\\ 0&0&0\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 1&1&1\\ 1&1&1\\ 1&1&1\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 2&2&2\\ 2&2&2\\ 2&2&2\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 3&3&3\\ 3&3&3\\ 3&3&3\\ \hline\end{array} \quad \begin{array}{|ccc|}\hline 4&4&4\\ 4&4&4\\ 4&4&4\\ \hline\end{array}$$

The only configurations one can construct with these allowed patterns are configurations of $\mathcal{A}^{\mathbb{Z}^2}$ with 0 everywhere except in some places where there are arbitrarily large blocks of the form $(*)$, and the configurations made of the infinite pattern $(*)$. We denote by $\mathbf{T}$ this subshift of finite type.

Let $\pi$ denote the letter-to-letter morphism defined by $\pi(x) = 0$ for $x \in \{*, 1, 2, 3, 4\}$ and $\pi(a) = a$, $\pi(b) = b$. Suppose that $\mathbf{SA}_\Delta(\mathbf{T})$ is sofic. Since $\mathcal{C}l_{\mathbf{Fact}}(\mathcal{S}ofic) = \mathcal{S}ofic$ then $\pi(\mathbf{SA}_\Delta(\mathbf{T}))$ would also be sofic, which is absurd since:

$$\pi(\mathbf{SA}_\Delta(\mathbf{T})) = \mathbf{T}_{\{ba;0a^m b^n 0 : m \neq n\}}.$$

So this construction proves that $\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT}) \neq \mathcal{S}ofic$.

The class of SFT is not stable under projective subaction and the class $\mathcal{C}l_{\mathbf{SA}}(\mathcal{SFT})$ is studied in [PS10]. Nevertheless a stable class for this operation is known, it is the class of effective subshifts. This follows from the fact that projective subactions are special cases of factors of subactions, and by Theorem 3.1 and Proposition 3.3 of [Hoc09] which establish that symbolic factors and subactions preserve effectiveness. That is to say $\mathcal{C}l_{\mathbf{SA}}(\mathcal{RE}) = \mathcal{RE}$.

With this formalism, the result of M. Hochman [Hoc09] can be written:

$$\mathcal{C}l_{\mathbf{Fact},\mathbf{SA}}(\mathcal{SFT}) = \mathcal{RE}.$$

More precisely, he proves that $\mathcal{C}l_{\mathbf{Fact},\mathbf{SA}}(\mathcal{SFT} \cap \mathcal{S}_{d+2}) \cap \mathcal{S}_{\leq d} = \mathcal{RE} \cap \mathcal{S}_{\leq d}$.

In Theorem 3.1, we show that $\mathcal{C}l_{\mathbf{Fact},\mathbf{SA}}(\mathcal{SFT} \cap \mathcal{S}_{d+1}) \cap \mathcal{S}_{\leq d} = \mathcal{RE} \cap \mathcal{S}_{\leq d}$. Moreover, there are examples of effective subshifts which are not sofic so $\mathcal{C}l_{\mathbf{Fact}}(\mathcal{SFT} \cap \mathcal{S}_d) = \mathcal{S}ofic \cap \mathcal{S}_d \neq \mathcal{RE} \cap \mathcal{S}_d$.

# 2 Computation zones for Turing machines

In this section we explain how to construct computation zones for a Turing machine and how to use them to simulate calculations. A Turing machine is a model of calculation composed by a finite automaton – the head of calculation – that moves on an infinite tape divided into boxes, each box containing a letter that can be modified by the head. A precise definition of Turing machine will be given in Subsection 2.1, and it will be explained how to code the behaviour of the machine thanks to local rules. The main problem is that this SFT is not enough to code calculations of the machine, since there is no rule that ensures the calculation is well initialized. So we need to embed calculations into specific zones. To make sure that the size of these computation zones is not a constraint and does not prematurely stop a calculation, we construct arbitrarily large computation zones with a sofic subshift in Subsection 2.2 and we implement the local rules of the Turing machine in these zones in Subsection 2.6.

## 2.1 Local rules to code the behaviour of a Turing machine

In this article, we consider Turing machines with some restrictions: the behaviour of the machine will be simulated only on the empty word (originally the tape only contains blank symbols $\sharp$). We also assume that the head cannot go to the left of the initial position. Note that we can impose these restrictions without loss of generality. First we recall the formal definition of a Turing machine. Remember that a Turing machine is a model of calculation composed by a finite automaton – the head of calculation – that can be in different states and moves on an infinite tape divided into boxes, each box containing a letter that can be modified by the head.

**Definition.** Let $\mathcal{M} = (Q, \Gamma, \sharp, q_0, \delta, Q_F)$ be a Turing machine, where:

- $Q$ is a finite set of states of the head of calculation; $q_0 \in Q$ is the initial state;
- $\Gamma$ is a finite alphabet;
- $\sharp \notin \Gamma$ is the blank symbol, with which the tape is initially filled; end of any enumerated word;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \cdot, \rightarrow\}$ is the transition function. Given the state of the head of calculation and the letter it can read on the tape — which thus depends on the position of the head of calculation on the tape — the letter on the tape is replaced or not by another one, the head of calculation moves or not to an adjacent box and changes or not of state;
- $F \subset Q_F$ is the set of final states — when a final state is reached, the calculation stops.

*Example* 2.1. We consider the Turing machine $\mathcal{M}_{\mathtt{ex}}$ that enumerates on its tape the words $ab, aabb, aaabbb, \ldots$ and never halts. This machine uses the three letters alphabet $\{a, b, \|\}$ and five states $Q = \{q_0, q_{\mathtt{a+}}, q_{\mathtt{b+}}, q_{\mathtt{b++}}, q_{\|}\}$. A separation symbol $\|$ is written at the end of each $a^n b^n$. The transition function $\delta_{\mathtt{ex}}$ is

$$\left.\begin{array}{l}\delta_{\mathtt{ex}}(q_0, \sharp) = (q_{\mathtt{b+}}, a, \rightarrow)\\ \delta_{\mathtt{ex}}(q_{\mathtt{b+}}, \sharp) = (q_{\|}, b, \rightarrow)\\ \delta_{\mathtt{ex}}(q_{\|}, \sharp) = (q_{\|}, \|, .)\end{array}\right\}$$ Initialization of the tape: the machine writes the first word $ab$ on the tape and place the head on the separation symbol $\|$ to the right of the word.

$$\left.\begin{array}{l}\delta_{\mathtt{ex}}(q_{\|}, \|) = (q_{\|}, \|, \leftarrow)\\ \delta_{\mathtt{ex}}(q_{\|}, b) = (q_{\|}, b, \leftarrow)\\ \delta_{\mathtt{ex}}(q_{\|}, a) = (q_{\mathtt{a+}}, a, \rightarrow)\end{array}\right\}$$ Suppose some word $a^n b^n \|$ is written on the tape, and that the head is on the $\|$ symbol in state $q_{\|}$. The machine looks for the rightmost symbol a in $a^n b^n$.

$$\left.\begin{array}{l}\delta_{\mathtt{ex}}(q_{\mathtt{a+}}, b) = (q_{\mathtt{b++}}, a, \rightarrow)\\ \delta_{\mathtt{ex}}(q_{\mathtt{b++}}, b) = (q_{\mathtt{b++}}, b, \rightarrow)\\ \delta_{\mathtt{ex}}(q_{\mathtt{b++}}, \|) = (q_{\mathtt{b+}}, b, \rightarrow)\end{array}\right\}$$ The machine replaces the leftmost symbol b by a symbol a and looks for the separation symbol $\|$ on the right of the word. Once it has found it, it is replaced by $bb\,\|$. The word $a^{n+1}b^{n+1}\,\|$ is now written on the tape and the head is on the $\|$ symbol in state $q_{\|}$.

A calculation of this machine will always go through the following configurations of the tape:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $a$ | $b$ | $(q_{\texttt{b++}}, b)$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $a$ | $(q_{\texttt{b++}}, b)$ | $b$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $(q_{\texttt{a+}}, b)$ | $b$ | $b$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $(q_{\|\|}, a)$ | $b$ | $b$ | $b$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $(q_{\|\|}, b)$ | $b$ | $b$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $(q_{\|\|}, b)$ | $b$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $b$ | $(q_{\|\|}, b)$ | ‖ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $b$ | $b$ | $(q_{\|\|}, \|\|)$ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $b$ | $b$ | $(q_{\|\|}, ♯)$ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $b$ | $(q_{\texttt{b+}}, ♯)$ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $b$ | $(q_{\texttt{b++}}, \|\|)$ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $a$ | $(q_{\texttt{b++}}, b)$ | ‖ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $(q_{\texttt{a+}}, b)$ | $b$ | ‖ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $(q_{\|\|}, a)$ | $b$ | $b$ | ‖ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $(q_{\|\|}, b)$ | $b$ | ‖ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $b$ | $(q_{\|\|}, b)$ | ‖ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $b$ | $b$ | $(q_{\|\|}, \|\|)$ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $b$ | $b$ | $(q_{\|\|}, ♯)$ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $b$ | $(q_{\texttt{b+}}, ♯)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $a$ | $(q_{\texttt{b++}}, \|\|)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $(q_{\texttt{a+}}, b)$ | ‖ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $(q_{\|\|}, a)$ | $b$ | ‖ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $(q_{\|\|}, b)$ | ‖ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $b$ | $(q_{\|\|}, \|\|)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $b$ | $(q_{\|\|}, ♯)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $a$ | $(q_{\texttt{b+}}, ♯)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |
| ... | ♯ | $(q_0, ♯)$ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ♯ | ... |

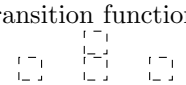
If an origin is given it is straightforward to describe the behaviour of a Turing machine with a set of two-dimensional patterns. The first dimension stands for the tape and second dimension for time evolution. We obtain the *space time diagram of computation of* $\mathcal{M}$ which can be construct locally by $3 \times 2$ allowed patterns:

- If the pattern codes a part of the tape on which the head of calculation does not act, the two line of allowed pattern are identical and for $x, y, z \in \Gamma$ one has:

| | | |
|---|---|---|
| $x$ | $y$ | $z$ |
| $x$ | $y$ | $z$ |

- If the head of calculation is present in the part of the tape coded, we code the modification given by the Turing machine. For example the rule $\delta(q_1, x) = (q_2, y, \leftarrow)$ will be coded by:

| | | |
|---|---|---|
| $(q_2, z)$ | $y$ | $z'$ |
| $z$ | $(q_1, x)$ | $z'$ |

Denote by $P_{\mathcal{M}}$ the set of forbidden patterns on the alphabet $\mathcal{A}_{\mathcal{M}} = \Gamma \cup (Q \times \Gamma)$ constructed according to the rules of $\mathcal{M}$ – patterns that cannot be seen as coming from the transition function as above. We can assume that the support of all patterns in $P_{\mathcal{M}}$ the following type:  . For example, with this assumption the rule $\delta(q_1, x) = (q_2, y, \leftarrow)$ becomes:

| | | |
|---|---|---|
| | $y$ | |
| $z$ | $(q_1, x)$ | $z'$ |

| | | |
|---|---|---|
| | $(q_2, z)$ | |
| $t$ | $z$ | $(q_1, x)$ |

| | | |
|---|---|---|
| | $z'$ | |
| $(q_1, x)$ | $z'$ | $z''$ |

Consider now the subshift of finite type $\mathbf{T}_{P_{\mathcal{M}}}$. It contains an element that is exactly the space time diagram of computation of $\mathcal{M}$, but also many other elements that are inconsistent. With the Turing machine $\mathcal{M}_{\texttt{ex}}$ of Example 2.1, the SFT $\mathbf{T}_{P_{\mathcal{M}_{\texttt{ex}}}}$ contains an element where the following configuration of the tape appears

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | ♯ | ... | ♯ | $a$ | $b$ | $b$ | $b$ | $b$ | $b$ | $(q_{\|\|}, \|\|)$ | ♯ | ... | ♯ | ... |

but this configuration is inconsistent since it is never reached by a calculation of $\mathcal{M}_{\texttt{ex}}$.

The problem comes from the lack of information about the beginning of a calculation. We need to specify a point in $\mathbb{Z}^2$ that stands for the origin of a calculation – the head of calculation is in the initial state $q_0$, and the row is filled with blank symbols ♯.

By compactness of the set of configurations of a subshift, it is impossible to impose that a special symbol appears exactly once in every configuration.

## 2.2 A substitutive sofic subshift as grid of computation

A classical problem in tiling theory is the construction of aperiodic tilings, that are sets of tiles that can only produce aperiodic configurations. A first example was initially given by Berger, who proved that the domino problem (is it possible to tile the whole plane with a given finite set of tiles?) is undecidable (see [Ber66] for the original proof by Berger and [Rob71] for Robinson's proof with a smaller set of tiles). Robinson reduces this problem to the Turing machine halting problem, which is known to be undecidable. The heart of the proof is the construction of an aperiodic tiling, which codes computation zones for Turing machines. These computation zones are all finite, but for any calculation of a Turing machine that stops, it is possible to find a zone large enough that contains it. Robinson entirely describes a finite set of tiles that produces the tiling, but there are many techniques to obtain it: Mozes [Moz89] gives a proof based on substitutions and Durand, Romashchenko and Shen [DRS08] propose a proof based on Kleene's fixed point theorem. We here define computation zones for Turing machine with a two dimensional substitution.

**Definition of the substitution $s_{\texttt{Grid}}$** Let $\mathcal{A}$ be a finite alphabet. A $(k,k')$-*two dimensional substitution* is a function $s : \mathcal{A} \to \mathcal{A}^{\mathbb{U}_{k,k'}}$ where $\mathbb{U}_{k,k'} = [0,k-1] \times [0,k'-1]$. We naturally extend $s$ to a function $s^{n,n'} : \mathcal{A}^{\mathbb{U}_{n,n'}} \to \mathcal{A}^{\mathbb{U}_{nk,n'k'}}$ by identifying $\mathcal{A}^{\mathbb{U}_{nk,n'k'}}$ with $(\mathcal{A}^{\mathbb{U}_{k,k'}})^{\mathbb{U}_{n,n'}}$. Starting from a letter placed in $(1,1) \in \mathbb{Z}^2$ and applying successively $s, s^{k,k'}, \dots, s^{k^{n-1},k'^{n-1}}$ we obtain a sequence of patterns in $\mathcal{A}^{\mathbb{U}_{k^i,k'^i}}$ for $i \in \{0,\dots,n\}$. Such patterns are called *s-patterns*. Note that the substitutions $s$ we define here are deterministic but one can imagine non deterministic substitutions replacing the function $s$ by a set of substitution rules, where a letter may have different images by the substitution. The definition of $s$-patterns naturally extends to non deterministic substitutions.

To describe the grid of computation, we consider two alphabets $\mathcal{G}_1$ and $\mathcal{G}_2$ (see Figure 1). The alphabet $\mathcal{G}_1 = \{$  $\}$ describes the zones of computation,  and  are called *computation boxes* where the computation holds and  are called *communication boxes* through which computation boxes can send information. More precisely,  and  are called *border computation boxes*. The alphabet $\mathcal{G}_2$ is constituted by lines which describe communication channels between the different zones of computation.

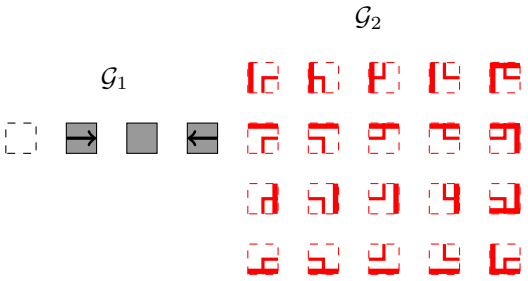



Figure 1: The alphabets $\mathcal{G}_1$ and $\mathcal{G}_2$ on which the substitution is defined.

We define two $(4,2)$-two dimensional substitutions, $s_1$ on $\mathcal{G}_1$ and $s_2$ on $\mathcal{G}_2$ (see Figure 2 for the substitution rules). Then, we define the product substitution $s_{\texttt{Grid}} = s_1 \times s_2$ on $\mathcal{G}_1 \times \mathcal{G}_2$. Iterations of $s_{\texttt{Grid}}$ on any pattern of $\mathcal{G}_1 \times \mathcal{G}_2$ produce arbitrarily large computation zones with communication channels between them (this will be detailed in Section 2.4 and Section 2.7). See Figure 5 for an example of an iteration of $s_{\texttt{Grid}}$.

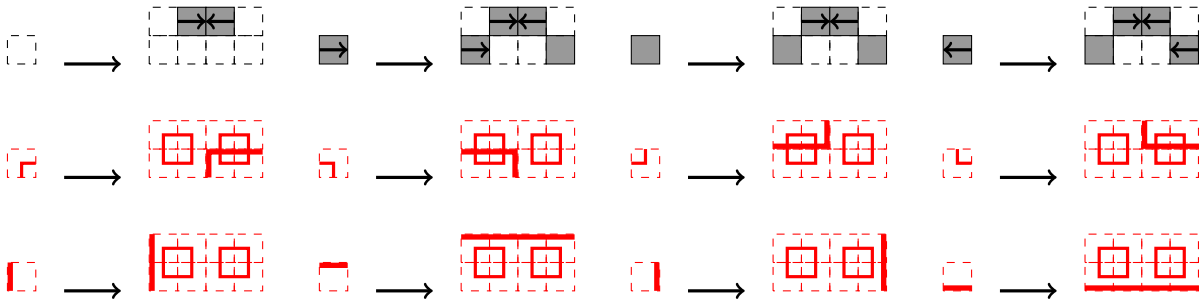


Figure 2: Basic elements to define the substitution rules of $s$. The first row lists the substitution rules of $s_1$ on the alphabet $\mathcal{G}_1$. The second and third rows contain substitution rules of $s_2$ on some of the letters of $\mathcal{G}_2$. All substitution rules of $s_2$ on $\mathcal{G}_2$ can be obtained by superimposing a substitution rule of the second row and a substitution rule of the third row. One can deduce substitution rules of $s_{\texttt{Grid}}$ on the alphabet $\mathcal{G}_1 \times \mathcal{G}_2$ by superimposing a rule of $s_1$ on $\mathcal{G}_1$ and a rule of $s_2$ on $\mathcal{G}_2$.

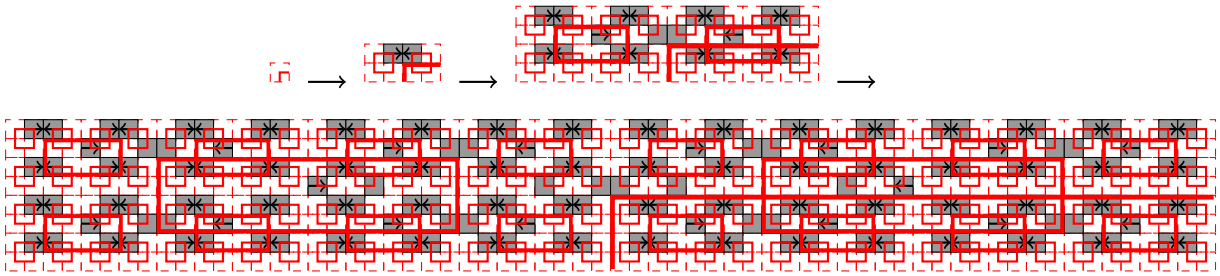


Figure 3: Four iterations of the substitution $s_{\texttt{Grid}}$ starting from an element of $\mathcal{G}_1 \times \mathcal{G}_2$.

We denote by $\pi_{\mathcal{G}_1}$ (resp. $\pi_{\mathcal{G}_2}$) the projection on $\mathcal{G}_1$ (resp. $\mathcal{G}_2$).

**Sofic subshift generated by the substitution** Given a substitution $s$, recall that a $s$-pattern is a pattern obtained by iteration of the substitution $s$ on a letter (for instance in Figure 5 are drawn $s$-patterns obtained after four iterations on the letter ). The *subshift generated by a substitution* $s$, denoted $\mathbf{T}_s$, is the set of configurations $x$ such that any pattern that appears in $x$ also appears in a $s$-pattern.

S. Mozes studied more general substitutions – non deterministic ones and substitution rules may be of different sizes – and proved that if the substitution $s$ satisfies some good property and has only strictly two-dimensional substitution rules, then $\mathbf{T}_s$ is a sofic subshift (see Theorem 4.1 of [Moz89]). In particular Mozes theorem can be applied for all deterministic substitutions, that is to say that all letter have only one image, like $s_{\texttt{Grid}}$. As a consequence, the following holds

**Fact 2.1.** *The subshift generated by* $s_{\texttt{Grid}}$*,*

$$\mathbf{T}_{\texttt{Grid}} = \mathbf{T}_{s_{\texttt{Grid}}} = \left\{ x \in (\mathcal{G}_1 \times \mathcal{G}_2)^{\mathbb{Z}^2} \ : \ \textit{for all } u \sqsubset x \textit{ there exists } n \in \mathbb{N} \textit{ such that } u \sqsubset s^n_{\texttt{Grid}}( ) \right\}$$

*is a two-dimensional sofic subshift.*

*Remark.* Note that $\pi_{\mathcal{G}_1}(\mathbf{T}_{\texttt{Grid}}) = \mathbf{T}_{s_1}$ and $\pi_{\mathcal{G}_2}(\mathbf{T}_{\texttt{Grid}}) = \mathbf{T}_{s_2}$ but $\mathbf{T}_{\texttt{Grid}}$ is different of $\mathbf{T}_{s_1} \times \mathbf{T}_{s_2}$.

A substitution $s : \mathcal{A} \to \mathcal{A}^{\mathbb{U}_{k,k'}}$ may be extended into an application $\widetilde{s} : \mathcal{A}^{\mathbb{Z}^2} \to \mathcal{A}^{\mathbb{Z}^2}$. This substitution has *unique derivation* if for every element $x \in \mathbf{T}_s$ there exists an unique $y \in \mathcal{A}^{\mathbb{Z}^2}$ and an unique $i \in \mathbb{U}_{k,k'}$ such that $\widetilde{s}(y) = \sigma^i(x)$.

Since the pattern appears in each rules of the substitutions $s_1$, for every configuration $x \in \mathbf{T}_{s_1}$, there exists $(i,j) \in [0,3] \times [0,1]$ such that $x_{\{n_1+i+1\}\times[n_2+j+1,n_2+j+2]} =$ for all $(n_1, n_2) \in \mathbb{N} \times \mathbb{N}$.

Moreover this pattern cannot appear in other position so $(i, j)$ is chosen in an unique way. Consider the plane partition $([n_1+i, n_1+i+3] \times [n_2+j, n_2+j+1])_{(n_1,n_2)\in\mathbb{N}\times\mathbb{N}}$ of the configuration $x$, since all boxes have different image by the substitution, this plane partition gives an unique antecedent y by $\widetilde{s_1}$. We deduce that $\widetilde{s_1}(y) = \sigma^{i,j}(x)$. Thus $\mathbf{T}_{s_1}$ has unique derivation. The same type of reasoning holds for $\widetilde{s_2}$.

**Fact 2.2.** *The substitutions $s_1$ and $s_2$ have unique derivation.*

## 2.3 Use of communication channels

A *communication channel* is a sequence of adjacent boxes marked by a special symbol – we call these marked boxes *channel boxes*. The channel begins and ends with two computation boxes. In our construction, the channel boxes are of two types which can appear in the same box:

- communication boxes from alphabet $\mathcal{G}_1$ that will be used for internal communication and communication between adjacent Turing machines;
- symbols from alphabet $\mathcal{G}_2$ that will be used for communication between non adjacent Turing machines.

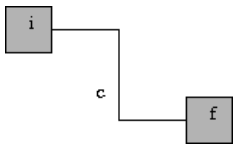



Figure 4: A communication channel between the computation boxes $i$ and $f$.

A *transfer of information* consists in three objects

- an *initial computation box* denoted $i$ and a *final computation box* denoted $f$
- local rules that determine the symbol transferred through a channel, depending on the direction of the channel starting from $i$ (resp.reaching $f$) and the symbol contained in the box $i$ (resp. in the box $f$)
- a communication channel $c$.

Two adjacent communication boxes carry the same symbol, which is transferred through the channel. Note that the computation boxes $i$ and $f$ are not necessary identical – for example a rule local may make a change at the end of the communication channel. The same computation box may be at the extremity of different communication channels.

Note that a communication box may belong to multiple communication channels, but this number must be bounded – in our construction the maximum number of channels going through a communication box will be 3 – internal communication inside a computation zone, communication between two adjacent computation zones of same level and communication between computation zones of different levels.

**Fact 2.3.** *Given a subshift $\Sigma$ that contains communication channels, it is possible to code transfers of information through these channels thanks to a product and a finite type operations, provided the symbols transferred locally depend on the symbol contained in the initial and final computation boxes of the channel.*

## 2.4 Description of computation zones

In this section we only consider the $\mathcal{G}_1$ part of the sofic subshift $\mathbf{T}_{\texttt{Grid}}$. We here describe the grid where computations hold for an element of $\pi_{\mathcal{G}_1}(\mathbf{T}_{\texttt{Grid}})$: horizontal dimension stands for the tape and vertical dimension for time evolution. On a horizontal line, a *zone of computation* is constituted by a group of computation boxes located between on the left and on the right. The *size* of a computation zone is the number of computation boxes which constitute the zone.

Consider a configuration $x \in \mathbf{T}_{\texttt{Grid}}$ and a computation zone in $x$. Since $s_1$ has unique derivation (see Fact 2.2) for any integer $n$, there exists a unique way to partition $x$ into $4^n \times 2^n$ rectangles so that each of these rectangles is a $s_1^n(a)$ for some $a \in \mathcal{G}_1$. So there exists a minimal integer $n$ such that the computation zone of $x$ appears in $s_1^n(a)$ for some $a \in \mathcal{G}_1$. We call this integer the *level* of the computation zone.

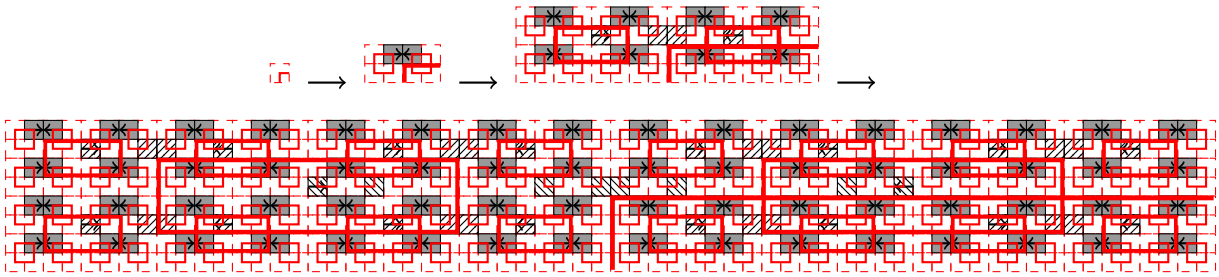


Figure 5: Four iterations of the substitution $s_{\texttt{Grid}}$. Computation zones of level 1,2 and 3 appear on the last pattern. The computation boxes of the first (resp. second and third) level are pictured with plain (resp. hashed with SW-NE lines and hashed with NW-SE lines) pattern.

At the iteration $n$ of the substitution on  , that is to say $s^n($$)$, we obtained a rectangle of size $4^n \times 2^n$. By induction, for all $m \in [1, n]$, we get $4^{n-m} * 2^{n-m} = 8^{n-m}$ zones of computation of level $m$ in $s^n($$)$, the size of these zones of computation is $2^m$. More precisely, if on the line $j \in [0, 2^n - 1]$ of $s^n($$)$ we find a zone of computation of level $m$, then in this line we have $4^{n-m}$ zones of computation of level $m$. Moreover for each computation box located at the coordinate $(i, j)$, the next computation box in the same column above the current one is separated by $2^m - 1$ communication boxes, so it is located at the coordinate $(i, j + 2^m)$, and this computation box is in a zone of computation of level $m$ at the same place that the box at the position $(i, j)$. There is the same phenomena if we look down. The set of zones of computation of the same size $2^m$ on a vertical line is called a *strip of computation* of size $2^m$.

For any other symbol $a \in \mathcal{G}_1$, the description is the same except for the bottom row.

*Remark.* Note that it is possible to have a  symbol on a row with no  symbol on its right – that is to say an infinite computation zone. In this case the computation zone has an infinite level.

**Fact 2.4.** *Consider $x \in \mathbf{T}_{\texttt{Grid}}$ and $C_n$ a computation zone of level $n$ of $x$. We assume that $C_n$ appears in the $i^{th}$ row of $x$, and we denote this row by $x_{\mathbb{Z}\times\{i\}}$. Then the following properties hold – remember that in this section we only consider the $\mathcal{G}_1$ part of the subshift $\mathbf{T}_{\texttt{Grid}}$:*

1. *$\mathcal{C}_n$ contains $2^n$ computation boxes, separated by communication boxes;*
2. *on $x_{\mathbb{Z}\times\{i\}}$ there are only computation zones of level $n$, separated by $2^{2n-1}$ communication boxes;*
3. *the row $x_{\mathbb{Z}\times\{i\}}$ is repeated vertically every $2^n$ rows, that is to say $x_{\mathbb{Z}\times\{i\}} = x_{\mathbb{Z}\times\{i+k\times 2^n\}}$ for any integer $k \in \mathbb{Z}$;*
4. *vertically, between every pair of consecutive computation boxes of $x_{\mathbb{Z}\times\{i\}}$ and $x_{\mathbb{Z}\times\{i+k\times 2^n\}}$, there are only $2^n - 1$ communication boxes.*

We now explain how it is possible for two computation boxes of the same computation zone and for two adjacent strips to communicate.

**Communication inside a strip** Two computation zones in the same computation strip of level $n$ communicate thanks to communication boxes of the $2^n - 1$ intermediate rows (vertical transfer of information), and inside a same computation zone communication between computation boxes occurs on the $2 * 4^{n-1} - 2^n$ communication boxes (horizontal transfer of information).

Figure 6 represents a computation grid where all zones of computation of the same size share the same color and are filled with the same pattern.

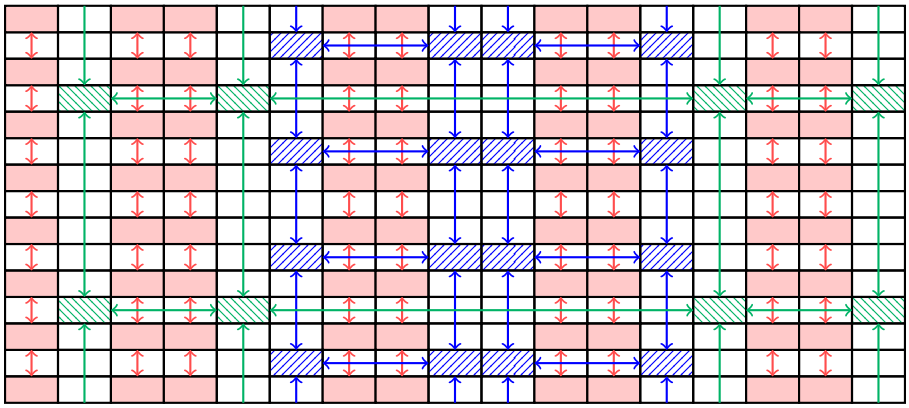


Figure 6: Computation grid with the communication between disconnected parts of the same computation zone. Computation zones of level 1,2 and 3 are pictured with three different colors and patterns. On a given row there are only computation zones of the same level, and there are $2^n$ rows between two rows with level $n$ computation zones. A row of a strip of computation of level $n$ is made of $2^n$ boxes arranged into a $2 * 4^{n-1}$ wide block of boxes. The two ways of communication (horizontal and vertical) are pictured with arrows whose color corresponds to the level of the computation zone or strip.

**Communication between two adjacent strips** Two strips of computation of same level can also communicate if they are adjacent – that is the leftmost computation box of the first strip and the rightmost computation box of the second strip are only separated by communication boxes.

**Fact 2.5.** *Communication boxes contain two communication channels – horizontal and vertical channels. Thanks to these channels, computation boxes into a same strip and two adjacent strips can communicate.*

## 2.5 Initialization of calculations : the clock

The computation strips described in the previous section are restricted in space but not in time, hence inconsistent configurations of a Turing machine may appear. To solve this problem, we equip each computation strip with a clock, that will be reinitialized periodically. At each step of calculation, the clock is increased and when it is reinitialized, the Turing machine starts a new calculation.

We use a four elements alphabet $\mathcal{C} = \{0, 1, \emptyset, \sim\}$ to construct a sofic subshift $\mathbf{T}_{\texttt{Clock}}$ obtained by adding finite type rules on $\mathbf{Prod}\left(\mathbf{T}_{\texttt{Grid}}, \mathcal{C}^{\mathbb{Z}^2}\right)$, where $\mathbf{T}_{\texttt{Grid}}$ is the sofic subshift described in Section 2.2. Denote $\pi_{\mathcal{C}}$ the projection on the second coordinate. The clock is actually a finite automaton that simulates binary addition modulo $2^{2^n}$ on a $2^n$ boxes tape — special symbol $\emptyset$ corresponds to the carry in binary addition, and symbol $\sim$ is used to synchronize adjacent computation zones of same level. To prevent the appearance of inconsistent states on the clock, we forbid the patterns $\boxed{\emptyset}\boxed{0}$, $\boxed{\emptyset}\boxed{1}$, $\boxed{0}\boxed{\emptyset}$, $\boxed{x}\boxed{\sim}$ and $\boxed{\sim}\boxed{x}$ where $x \in \{0, 1, \emptyset\}$ — we call this finite type condition **Consist**.

We describe the finite type conditions **Count** on the alphabet $\mathcal{G}_1 \times \mathcal{C}$ in Figure 7.

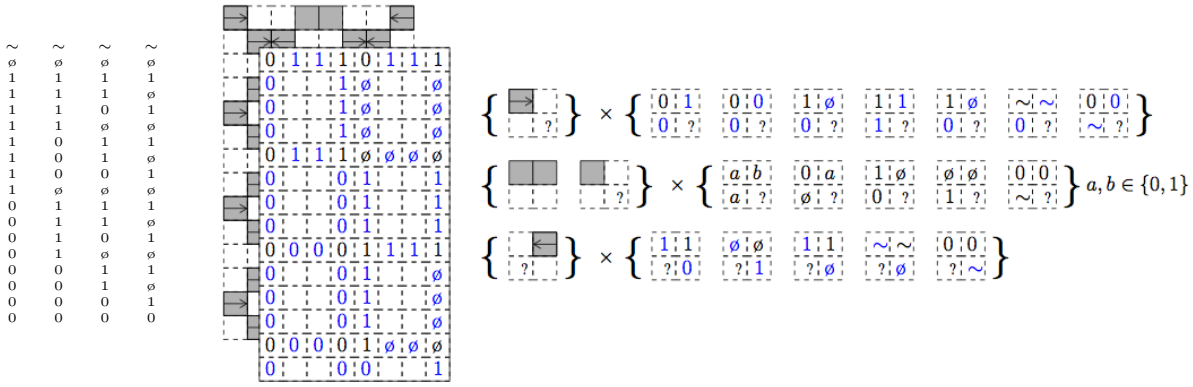


Figure 7: On the left, an example of the evolution of the clock for a computation zone of size $2^2$. On the middle the evolution of a part of this clock on a level 2 computation strip: on the tape are successively written 001ø, 0011, 01øø and 0101. And on the right, some of the finite type conditions **Count**, represented by the allowed patterns, added to the sofic subshift $\mathbf{T}_{\texttt{Grid}}$ to obtain the sofic subshift $\mathbf{T}_{\texttt{Clock}}$.

The clocks of different computation levels evolve according to the rules described in Figure 7, and when a symbol ø reaches the left most computation box , it is reinitialized. Before reinitialization, the clock passes through the configuration with only $\sim$ symbols on the tape. Thanks to this configuration, it is possible to synchronize a clock on a strip of level $n$ with its two neighbours of level $n$. For example the clock for a computation strip of level 1 will be $00, 01, 1ø, 11, øø, \sim\sim, 00, \dots$ Hence a clock for a computation strip of level $n$ is reinitialized after $2^{2^n} + 2$ steps.

To these local rules we add another finite type condition called **Synchro**, that ensures that clocks corresponding to computation zones on the same level are synchronized, that is they are in the same state at every calculation step – on a same row, all the clocks are in the same state. This can be easily done by the following way: a clock is in the configuration $\sim \cdots \sim$ only when its left and right neighbours are in the same configuration – a signal carrying symbol $\sim$ is sent through communication channel between neighbours. We thus obtain a sofic subshift

$$\mathbf{T}_{\texttt{Clock}} = \mathbf{FT}_{\mathbf{Count} \cup \mathbf{Consist} \cup \mathbf{Synchro}} \left( \mathbf{Prod} \left( \mathbf{T}_{\texttt{Grid}}, \mathcal{C}^{\mathbb{Z}^2} \right) \right)$$

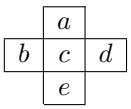
in which every computation strip of $\mathbf{T}_{\texttt{Grid}}$ is now equipped with a clock. Note that we do not impose clocks for different levels of computation zones to be somehow synchronized.

**Fact 2.6.** *Consider the sofic subshift $\mathbf{T}_{\texttt{Clock}}$, in the interior of a strip of computation of level $n$ which is of size $2^n$, the clock is initialized every $2^{2^n} + 2$ on computation zone of level $n$.*

## 2.6 A sofic subshift to describe Turing machines behaviour

We are going to use the subshift $\mathbf{T}_{\texttt{Clock}}$ constructed in Section 2.5 to construct a sofic subshift where the computation of $\mathcal{M}$ in a space $2^n$ holds on each strip of computation of size $2^n$, for all $n \in \mathbb{N}^*$. We want to apply the rules of $P_{\mathcal{M}}$ to adjacent computation boxes that may be separated by a sequel of communication boxes. As explained in Section 2.3 information may be transferred through communication boxes horizontally and vertically. The space of computation of $\mathcal{M}$ is restricted by  on the left and by  on the right. We start again with the sofic subshift $\mathbf{T}_{\texttt{Clock}}$ defined in Section 2.5, into the product subshift $\mathbf{Prod}\left(\mathbf{T}_{\texttt{Clock}}, \tilde{\mathcal{A}}^{\mathbb{Z}^2}\right)$ where $\tilde{\mathcal{A}} = \mathcal{A}_{\mathcal{M}} \cup (\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}})$. A symbol in $\tilde{\mathcal{A}}$ may be either a symbol of $\mathcal{A}_{\mathcal{M}}$ inside a computation box or three symbols of $\mathcal{A}_{\mathcal{M}}$ transferred – horizontally for the first and the second and vertically for the third – through a communication box. We have defined $\pi_{\mathcal{G}_1}$, $\pi_{\mathcal{G}_2}$ and $\pi_{\mathcal{C}}$ respectively the projections on $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{C}$ in the first coordinate of $\mathbf{Prod}\left(\mathbf{T}_{\texttt{Clock}}, \tilde{\mathcal{A}}^{\mathbb{Z}^2}\right)$. Moreover denote $\pi_{\tilde{\mathcal{A}}}$ the projection on the second coordinate of $\mathbf{Prod}\left(\mathbf{T}_{\texttt{Clock}}, \tilde{\mathcal{A}}^{\mathbb{Z}^2}\right)$, if we are in a communication box, we can write $\pi_{\tilde{\mathcal{A}}_1}$, $\pi_{\tilde{\mathcal{A}}_2}$ and $\pi_{\tilde{\mathcal{A}}_3}$ respecively for the first, second and third coordinate of $\mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}} \times \mathcal{A}_{\mathcal{M}}$.

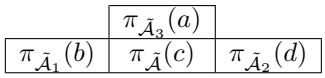
To the sofic-subshift $\mathbf{Prod}\left(\mathbf{T}_{\texttt{Clock}}, \tilde{\mathcal{A}}^{\mathbb{Z}^2}\right)$, we add the following finite conditions, the support of all forbidden patterns have the following form:



with $a, b, c, d, e \in \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{C} \times \tilde{\mathcal{A}}$

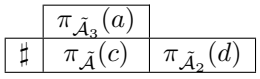
The conditions are:

- if the center box corresponds to a communication box in $\mathbf{T}_{\texttt{Clock}}$, that is to say $\pi_{\mathcal{G}_1}(c) =$ , one uses conditions **Transfer**: the first and second coordinates are constant along the central row, and the third coordinate is constant along the central comlumn – more precisely $\pi_{\tilde{\mathcal{A}}_1}(b) = \pi_{\tilde{\mathcal{A}}_1}(c) = \pi_{\tilde{\mathcal{A}}_1}(d)$, $\pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_2}(c) = \pi_{\tilde{\mathcal{A}}_2}(d)$ and $\pi_{\tilde{\mathcal{A}}_3}(a) = \pi_{\tilde{\mathcal{A}}_3}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$, these conditions hold if all boxes in the neighborhood are communication boxes, in fact, if there is a computation box, we just use the projection $\pi_{\tilde{\mathcal{A}}}$;
- if the center box corresponds to a computation box in $\mathbf{T}_{\texttt{Clock}}$, that is to say $\pi_{\mathcal{G}_1}(c) \in \{$ , ,  $\}$, one uses one of the followings conditions:
  - conditions **Init**: when the clock is in a initial state, there is the blank symbol $\sharp$ on each box and the tape is in the initial state on the left computation box  – more precisely
    - if $\pi_{\mathcal{C}}(c) = \sim$ and $\pi_{\mathcal{G}_1}(c) =$  then $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_1}(d) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_3}(a) = (q_0, \sharp)$,
    - if $\pi_{\mathcal{C}}(c) = \sim$ and $\pi_{\mathcal{G}_1}(c) \in \{$  ,  $\}$ then $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_1}(d) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_3}(a) = \sharp$;
  - conditions **Comp**: we use the rules described in $\mathcal{P}_{\mathcal{M}}$ if the clock is not in the initial state – more precisely
    - if $\pi_{\mathcal{C}}(c) \neq \sim$ and $\pi_{\mathcal{G}_1}(c) =$  then



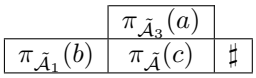
$\in P_{\mathcal{M}}$, $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d)$ and $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$,

    - if $\pi_{\mathcal{C}}(c) \neq \sim$, $\pi_{\mathcal{G}_1}(c) =$  and the third coordinate of $\delta(\pi_{\tilde{\mathcal{A}}}(c))$ is different from $\leftarrow$, that is to say the transition function of the Turing machine does not move the head toward the left, then



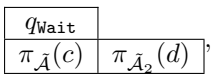
$\in P_{\mathcal{M}}$, $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d)$ and $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$,

    if $\pi_{\mathcal{C}}(c) \neq \sim$, $\pi_{\mathcal{G}_1}(c) =$  and the third coordinate of $\delta(\pi_{\tilde{\mathcal{A}}}(c))$ is different from $\rightarrow$, that is to say the transition function of the Turing machine does not move the head toward the right, then



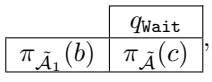
$\in P_{\mathcal{M}}$, $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d)$ and $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$;

  - conditions **Bound**: if the head wants to go to the left of the computation box  or to the right of the computation box , the head reaches a special state and the computation continues in an infinite loop until the computation is initiated by the clock – more precisely
    - if $\pi_{\mathcal{C}}(c) \neq \sim$, $\pi_{\mathcal{G}_1}(c) =$  and the third coordinate of $\delta(\pi_{\tilde{\mathcal{A}}}(c))$ is $\leftarrow$, then



$\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d)$ and $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$;

    - if $\pi_{\mathcal{C}}(c) \neq \sim$, $\pi_{\mathcal{G}_1}(c) =$  and the third coordinate of $\delta(\pi_{\tilde{\mathcal{A}}}(c))$ is $\rightarrow$, then



$\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d)$ and $\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(e)$;

* if $\pi_{\mathcal{C}}(c) \neq \sim$, $\pi_{\mathcal{G}_1}(c) \in \{$ ■ , → , ← $\}$ and $\pi_{\tilde{\mathcal{A}}} = q_{\texttt{Wait}}$, then

$$\pi_{\tilde{\mathcal{A}}}(c) = \pi_{\tilde{\mathcal{A}}_3}(a) = \pi_{\tilde{\mathcal{A}}_3}(e) = \pi_{\tilde{\mathcal{A}}_2}(b) = \pi_{\tilde{\mathcal{A}}_1}(d) = q_{\texttt{Wait}},$$

* if $\pi_{\mathcal{C}}(c) \neq \sim$ and $\pi_{\tilde{\mathcal{A}}_1}(b) = q_{\texttt{Wait}}$ or $\pi_{\tilde{\mathcal{A}}_2}(d) = q_{\texttt{Wait}}$ then $\pi_{\tilde{\mathcal{A}}}(c) = q_{\texttt{Wait}}$.

Define the sofic subshift $\mathbf{T}_{\mathcal{M}}$:

$$\mathbf{T}_{\mathcal{M}} = \mathbf{FT}_{\mathbf{Transfer} \cup \mathbf{Init} \cup \mathbf{Comp} \cup \mathbf{Bound}} \left( \mathbf{Prod} \left( \mathbf{T}_{\texttt{Clock}}, \tilde{\mathcal{A}}^{\mathbb{Z}^2} \right) \right).$$

For more convenience, we gather the local rules **Transfer**, **Init**, **Comp** and **Bound** in $\mathbf{Work}_{\mathcal{M}}$, and the construction is summed up by: $\mathbf{T}_{\mathcal{M}} = \mathbf{FT}_{\mathbf{Work}_{\mathcal{M}}} \left( \mathbf{Prod} \left( \mathbf{T}_{\texttt{Grid}}, \mathcal{A}_{Comp(\mathcal{M})}^{\mathbb{Z}^2} \right) \right)$ for any Turing machine $\mathcal{M}$.

On each strip of computation appears parts of the space time diagram of the calculation of $\mathcal{M}$ on the empty word. Each part of these space time diagrams are limited in space by the size of the strip of computation and the number of steps is bounded exponentially by the length of the strip. Thus we can find in $\mathbf{T}_{\mathcal{M}}$ arbitrary large part of space time diagram of $\mathcal{M}$.

**Fact 2.7.** *The subshift $\mathbf{T}_{\mathcal{M}}$ contains all calculations of the Turing machine $\mathcal{M}$ on space time diagram of size $2^n \times \left(2^{2^n} + 2\right) - 2^n$ boxes tape and $2^{2^n} + 2$ steps of calculation – starting with an empty entry word.*

*Example* 2.2. In this example the Turing machine $\mathcal{M}_{\texttt{ex}}$ starts its enumeration with the word $ab$. The picture describes how a run is coded on a computation grid. If one only considers computation boxes of level 2, they form a three by four computation zone (three steps of calculation on a four boxes tape).

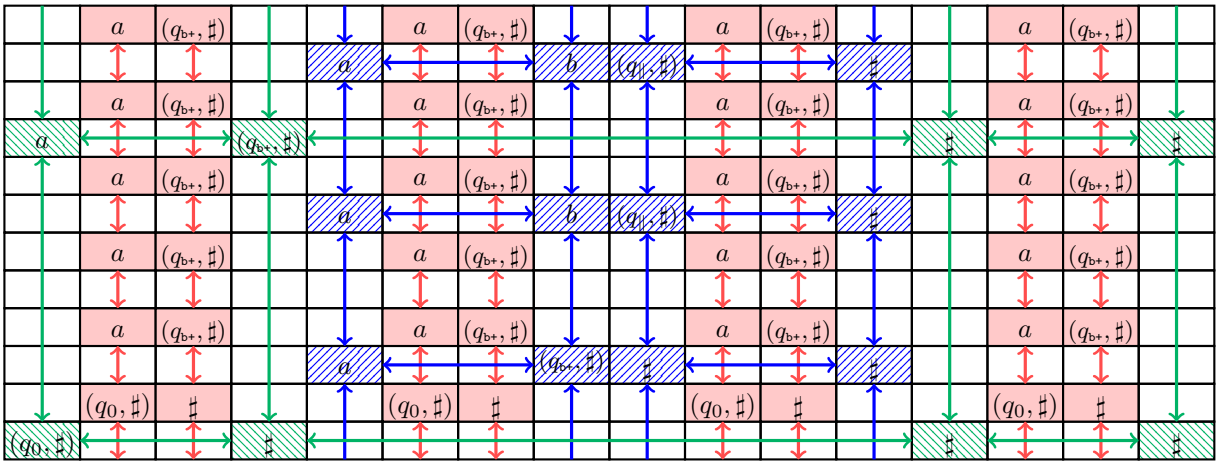


Figure 8: Calculation of a Turing machine on a computation grid with computation zones of levels 1, 2 and 3. Remark that each ↑ or ↔ arrow actually carries a symbol, but for more readability they are not pictured here. For the same reason the clock is also omitted.

## 2.7 Communication channels between Turing machine of different levels

In the sequel computation strips will need to communicate. For two strips of the same level communication it is easy since between two zones of computation of adjacent strips of level $n$, there are only communication boxes. Then one bit of information can be exchanged between two adjacent strips of level $n$ at each step of calculation (see Section 2.4). But if the two strips are not of the same level the problem is not as simple. We present in this section a communication grid that allows a strip of level $n$ to communicate with a strip of level $n-1$ and a strip of level $n+1$. This communication grid is based on the $\mathcal{G}_2$ part of the subshift $\mathbf{T}_{\texttt{Grid}}$.

The lines obtained with the alphabet $\mathcal{G}_2$ are called *communication lines*. Communication between computation zones of different levels are made through these lines. Under the action of $s_2$, communication

lines form rectangles. The two rectangles obtained after $n$ iteration of an element of $\mathcal{G}_2$ are called *communication rectangles of level* $n$. Each rectangle of level $n$ intersects two rectangles of level $n-1$ and it is intersected by a rectangle of level $n+1$.

If we consider a border computation box → (resp. ←) in a computation zone of level $n$, it is inside a communication rectangle of level $n$. Thus if we go horizontally on the left (resp. the right) of this box we meet the left border (resp. the right border) of this rectangle. On the bottom and top lines of this rectangle, we encounter two border computation boxes ( → and ← ) which are in two different computation zones of level $n-1$.

By local rules it is possible to construct *communications channel of level* $n$, that start from each border computation box ( → or ← ) of level $n$. The channel of communication goes on horizontally on the right and left branches until it meets the right or left border of a communication rectangle which is necessary of level $n$. Then the channel goes up and follows the border of the rectangle until it meets a border computation box. This box is necessarily of level $n-1$. Thus a computation zone can communicate with the four computation zones of the previous level which are included in itself (see Figure 9). These channels are used in Section 3.5 to ensure communication between computation zones of different levels. We remark that zones of a level $n$ repeat vertically with half the frequency of level $n-1$ zones. Therefore half the level $n-1$ zones do not incoming path from higher zones.

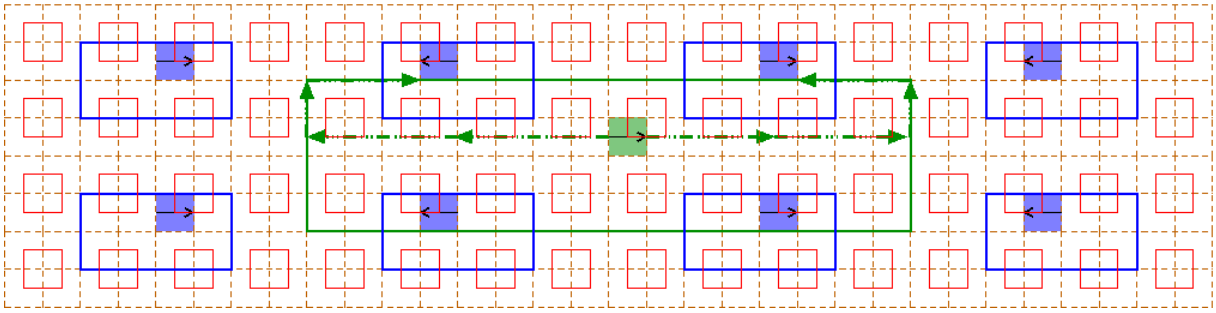


Figure 9: A computation grid with communication lines. The computation zone of level 3 communicates with level 2 computation zones it contains. This communication is made through the level 3 communication rectangle inside which the left border computation box is. Symmetrically, one can imagine that the right border communication box communicates with two other level 2 computation zones, this in not pictured here.

**Fact 2.8.** *For any computation strip of level $n$, there are two communication channels starting from each border computation box → or ← of level $n$ and ending at a border computation box of level $n-1$ – one → and one ← . Starting from a computation zone of level $n$, the four computation strips of level $n-1$ associated can be reached by this way.*

# 3 Proof of the main theorem

The ideas of the proof of the main result of this article were presented in the Introduction. We give here technical details that rely on constructions presented in the previous sections. We want to prove the following result.

**Theorem 3.1.** *Any effective subshift of dimension $d$ can be obtained with factor and projective subaction operations from a subshift of finite type of dimension $d+1$.*

Thanks to the formalism of Section 1 and since $\mathcal{RE}$ is stable under **Fact** and **SA** operations, we rewrite it:

$$\mathcal{C}l_{\mathbf{Fact},\mathbf{SA}}(\mathcal{SFT} \cap \mathcal{S}_{d+1}) \cap \mathcal{S}_{\leq d} = \mathcal{RE} \cap \mathcal{S}_{\leq d}.$$

This result improves Hochman's [Hoc09] since our construction decreases the dimension.

We here prove this statement in the particular case $d = 1$, but the proof can be easily extended to any dimension. Let $\Sigma$ be a one dimensional effective subshift, defined on an alphabet $\mathcal{A}_\Sigma$.

## 3.1 Construction of the four layers of SFT

We start with the two-dimensional fullshift $\mathcal{A}_\Sigma^{\mathbb{Z}^2}$ with a spatial extension operation, and thanks to factor, product and finite type operations we construct a sofic subshift $\mathbf{T}_{\texttt{Final}}$ such that after factor and projective subaction we obtain $\Sigma$. To do that, we eliminate configurations $x$ such that $x_{\mathbb{Z}\times\{0\}}$ contains a forbidden word of $\Sigma$. Then the projective subaction that consists in only keeping the first coordinate of a two-dimensional configuration $x$ gives the subshift $\Sigma$.

To resume the two-dimensional sofic subshift is made of four layers that are glued together thanks to product operations:

- first layer contains $\mathcal{A}_\Sigma^{\mathbb{Z}^2}$ and all horizontal lines are identical by finite condition **Align**, the other layers force the horizontal line to be an element of the effective subshift $\Sigma$, thus this subshift can be obtained after projective subaction (to keep horizontal line) and factor (to keep the first layer);
- layer 2 contains the computation zones for Turing machines equipped with the clock (this construction is described in Section 2), that will be used by both machines $\mathcal{M}_{\texttt{Forbid}}$ and $\mathcal{M}_{\texttt{Search}}$; but also the communication channels that will be used by the same machines to send requests (see Sections 3.4 and 3.5);
- layer 3 is devoted to Turing machines $\mathcal{M}_{\texttt{Forbid}}$, and communication with the Turing machines $\mathcal{M}_{\texttt{Search}}$ (this part is described in Section 3.4);
- layer 4 is devoted to Turing machines $\mathcal{M}_{\texttt{Search}}$ and internal communication between these machines (see Section 3.5).

Of course each of this layer depends on the others (for example layer 3 uses computation zones given by layer 2), and the dependences are coded thanks to finite type operations.

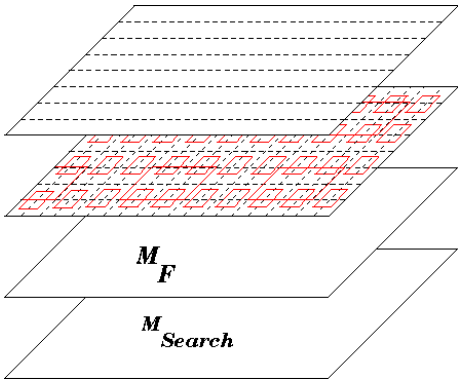



Figure 10: Four layers in the final construction.

## 3.2 Addresses in a strip

Since, on the first layer, each column is formed by one letter of $\mathcal{A}_\Sigma$, to check a word in an horizontal configuration, it is sufficient to check the first layer in the corresponding columns.

Let $C_n$ be a computation zone of level $n$ of an element $x \in \mathbf{T}_{\texttt{Grid}}$ and let $S_n$ be the computation strip associated. By Fact 2.2, there exists an unique $i \in [0, 4^n - 1] \times [0, 2^n - 1]$ and an unique $y \in \mathbf{T}_{\texttt{Grid}}$ such that $s^n_{\texttt{Grid}}(y) = \sigma^i(x)$ so there exists an unique $(j_1, j_2) \in \mathbb{Z}^2$ such that $C_n \sqsubset s^n_{\texttt{Grid}}(y_{(j_1,j_2)})$. One has $S_n \sqsubset \sigma^{-i}(s^n_{\texttt{Grid}}(y_{\{j_1\}\times\mathbb{Z}\}})) \sqsubset x$, the strip $\sigma^{-i}(s^n_{\texttt{Grid}}(y_{\{j_1\}\times\mathbb{Z}\}}))$ is the *dependency strip* associated with the computation strip $S_n$

In $\mathbf{T}_{\texttt{Grid}}$ the tape of a Turing machine in a strip of level $n$ is fractured. Thus a Turing machine of level $n$ cannot view all columns which are in its associated dependency strip. To get this information, this Turing machine communicates with a Turing machine of lower level (see Section 3.5) but both machines need to precisely identify a column.

Given a dependency strip associated with a computation strip of level $n$, it is possible to describe the coordinate relative to this strip of any column of the dependency strip by an address which contains $n$

letters in a four elements alphabet. Each $s_1^n(a)$ is horizontally decomposed into four (possibly different) $s_1^{n-1}(b)$ where $a, b \in \mathcal{G}_1$. The first letter of the address indicates in which of these dependency stripes of size $n-1$ the column is located. By iteration of this process the position of a column is exactly given with $n$ letters (see Figure 11).

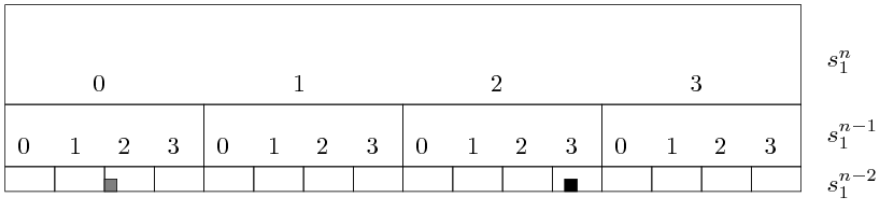



Figure 11: Addresses of two boxes inside a dependency strip associated with a computation zone of level 3. The address of the column of the black box is 231 and for the grey box, the address of the column is 020.

**Fact 3.2.** *For every dependency strip associated with a computation strip of level $n$, it is possible to describe the position of any column by an address of length $n$ on a four elements alphabet.*

## 3.3 Responsibility zones

On each computation zone a Turing machine makes calculations. The Turing machine $\mathcal{M}_{\texttt{Forbid}}$ described more precisely in Section 3.4 enumerates patterns and then checks that these patterns never appear. Since it takes an infinite number of steps of calculation to check that one pattern does not appear in the entire configuration, each Turing machine $\mathcal{M}_{\texttt{Forbid}}$ only checks a finite zone. The finite zone in which the machine ensures that no forbidden pattern it produces appears is called the *responsibility zone* of the machine.

We thus associate a responsibility zone with each strip of computation. For a strip of level $n$ this responsibility zone is $3 * (2 * 4^{n-1}) = 6 * 4^{n-1}$ wide and centered on the strip (see Figure 12), so that the responsibility zone of a strip starts at the end of the strip of same level on its left and ends at the beginning of the strip of same level on its right.

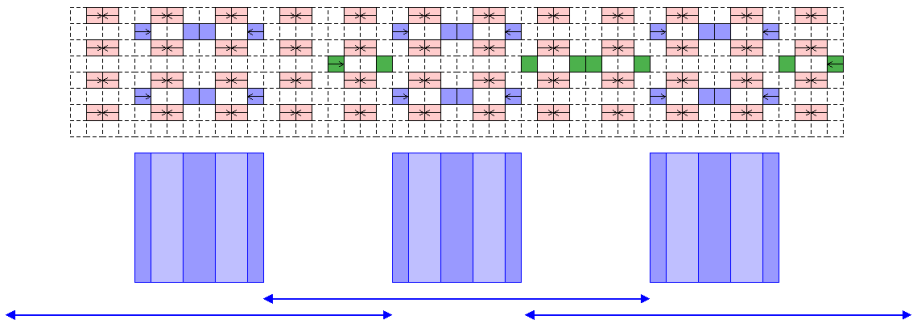


Figure 12: Responsibility zones for strips of level 2. These zones are 24 boxes wide and overlap on 8 boxes. The responsibility zone of the center strip starts at the end of the strip on its left and ends at the beginning of the strip on its right.

Responsibility zones defined in this way overlap: two adjacent responsibility zones of same level $n$ share $2 * 4^{n-1}$ boxes. These overlappings are essential: if they did not exist, one can imagine that a forbidden pattern not entirely included in any responsibility zone would not be detected. Moreover the non bounded size of overlappings ensures that any pattern is inside an infinite number of responsibility zones of increasing levels.

## 3.4 Generation and detection of forbidden patterns by $\mathcal{M}_{\texttt{Forbid}}$

Since $\Sigma$ is recursively enumerable, there exists a Turing machine that enumerates the forbidden patterns of $\Sigma$. We here describe a modified version of this Turing machine that also checks that no forbidden pattern appears inside its responsibility zone, on the first level of the construction ${\mathcal{A}_\Sigma}^{\mathbb{Z}^2}$. Computation zones are not connected (see Figure 8), so a calculation of $\mathcal{M}_{\texttt{Forbid}}$ on a strip of computation of size $2^n$ cannot access entirely its responsibility zone. The machine $\mathcal{M}_{\texttt{Forbid}}$ needs the help of a second Turing machine $\mathcal{M}_{\texttt{Search}}$ to obtain the patterns of ${\mathcal{A}_\Sigma}^{\mathbb{Z}}$ written in its responsibility zone. The behaviour of $\mathcal{M}_{\texttt{Forbid}}$ is the following: it enumerates as many forbidden patterns as the size of the computation zone allows, and each time such a pattern is generated, $\mathcal{M}_{\texttt{Forbid}}$ checks that it does not appear in its responsibility zone.

**Tapes of $\mathcal{M}_{\texttt{Forbid}}$** The machine $\mathcal{M}_{\texttt{Forbid}}$ uses three tapes:

- the first tape is the *calculation tape*;
- the second tape is a *writing tape*, where the forbidden patterns are successively written;
- the last tape is the *communication tape* and contains successively the addresses of letters from alphabet $\mathcal{A}_\Sigma$ needed by $\mathcal{M}_{\texttt{Forbid}}$ to check no forbidden pattern appears inside its responsibility zone; $\mathcal{M}_{\texttt{Forbid}}$ waits for the required $\mathcal{M}_{\texttt{Search}}$ machine of its neighbourhood (left, middle or right machine) to be available, then sends it the address of the letter it wants to access (see Section 3.5).

**Detection of the size of the responsibility zone associated** First, the Turing machine $\mathcal{M}_{\texttt{Forbid}}$ detects the size of the computation zone between $\boxed{\rightarrow}$ and $\boxed{\leftarrow}$. Thus, $\mathcal{M}_{\texttt{Forbid}}$ knows the size of its responsibility zone. This can be in linear time according to the size of the computation zone considered.

**Enumeration of forbidden patterns** Then, $\mathcal{M}_{\texttt{Forbid}}$ enumerates forbidden patterns and each time it encounter one, it checks if this forbidden pattern appears in the associated responsibility zone before to enumerate the following one.

**Check of the responsibility zone** Assume that the machine $\mathcal{M}_{\texttt{Forbid}}$ has written on its writing tape a forbidden pattern $f = f_0 f_1 \dots f_{k-1}$. Assume that $\mathcal{M}_{\texttt{Forbid}}$ must check a responsibility zone of level $n$ denoted $a_0 a_1 \dots a_{6*4^{n-1}-1}$. It asks $\mathcal{M}_{\texttt{Search}}$ for the first letter in its responsibility zone $a_0$ (the principe of a request is explained in Section 3.5), and compares it with $f_0$. If the letters coincide, then it is still possible that $f$ appears in position 0 in the responsibility zone, so the comparison of the two patterns $f$ and $a_0 \dots a_k$ continues. If $f_0 \neq a_0$ then we are sure that $f$ does not appear at this location. If $f = a_0 \dots a_k$, the Turing machine $\mathcal{M}_{\texttt{Forbid}}$ stops its computation and enter in a state which says that a forbidden patterns appears in the checked configuration. This state will be forbidden in the final subshift of finite type. When the word $a_0 \dots a_{k-1}$ is checked, $\mathcal{M}_{\texttt{Forbid}}$ continues the comparison with $a_1 \dots a_k$, $\dots$, $a_{6*4^{n-1}-k-1} \dots a_{6*4^{n-1}-1}$. At most, to check if $f$ appears in the responsibility zone of level $n$ , $\mathcal{M}_{\texttt{Forbid}}$ takes $6 * 4^{n-1} * k * t(n)$ where $t(n)$ is the time takes by $\mathcal{M}_{\texttt{Search}}$ to answer a request of $\mathcal{M}_{\texttt{Forbid}}$; the time $t(n)$ is estimated in Section 3.5.

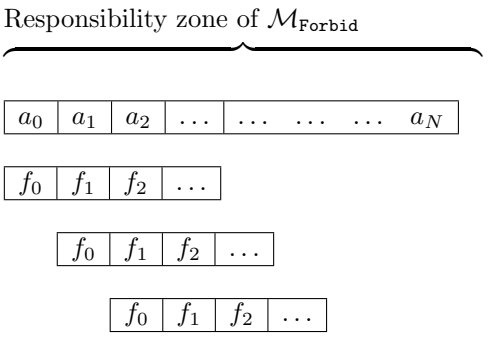



Figure 13: When a forbidden pattern $f = f_0 f_1 \dots f_k$ is generated by $\mathcal{M}_{\texttt{Forbid}}$, comparisons with the patterns appearing in the responsibility zone of $\mathcal{M}_{\texttt{Forbid}}$ are made in parallel.

## 3.5 Scan of the entire responsibility zone by $\mathcal{M}_{\texttt{Search}}$

The Turing machine $\mathcal{M}_{\texttt{Search}}$ is sent a *request* – that is to say a sequence of symbols which codes the address of a letter inside a responsibility zone of a $\mathcal{M}_{\texttt{Forbid}}$ machine – by $\mathcal{M}_{\texttt{Forbid}}$ each time an address is totally written on the communication tape (the third tape of $\mathcal{M}_{\texttt{Forbid}}$). The Turing machine $\mathcal{M}_{\texttt{Search}}$ must respond the letter corresponding to the address inside the responsibility zone, on the first level of the construction ${\mathcal{A}_\Sigma}^{\mathbb{Z}^2}$. Note that the responsibility zone of a $\mathcal{M}_{\texttt{Forbid}}$ machine of level $n$ does not exactly match with the communication network of $\mathcal{M}_{\texttt{Search}}$ machines of same level. Actually a $\mathcal{M}_{\texttt{Forbid}}$ machine shares its responsibility zone with three $\mathcal{M}_{\texttt{Search}}$ machines, and depending on the address of the bit requested, the $\mathcal{M}_{\texttt{Forbid}}$ sends its request to the appropriate $\mathcal{M}_{\texttt{Search}}$ machine (see Figure 14 for an example).

**Tapes of $\mathcal{M}_{\texttt{Search}}$** The machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ uses three tapes:

- the first tape is the *calculation tape*;
- the second tape is the *hierarchical request tape*; this is where the bits of an address transferred by the $\mathcal{M}_{\texttt{Search}}$ of level $n+1$ are written.
- the three last tapes are the *left request tape*, the *center request tape* and the *right request tape* which correspond to the addresses of the bits asked by the Turing machine $\mathcal{M}_{\texttt{Forbid}}$ of level $n$ localized respectively to the left, inside and to the right of the communication strip of the machine $\mathcal{M}_{\texttt{Search}}$ considered.

**Request sent by $\mathcal{M}_{\texttt{Forbid}}$** Each time that an address is written on the communication tape of a Turing machine $\mathcal{M}_{\texttt{Forbid}}$, this machine sends this request to the corresponding $\mathcal{M}_{\texttt{Search}}$ of the same level localized in the same communication strip or in communication strips directly to the left or to the right. $\mathcal{M}_{\texttt{Forbid}}$ sends one bit composing the address every step of calculation, so that a level $n$ Turing machine sends a bit every $2^n$ rows – if we implement Turing machines in the subshift of finite type described in Section 2.6. Adjacent strips of same level can communicate by communication channels described in Section 2.4 using the fact that in one row there is only computation zones of same level (see Fact 2.4). The bits of the address are sent one by one, hence the transfer takes $2^n * n$ rows since the size of the address of the request is $n$. The request is written on the corresponding request tape. $\mathcal{M}_{\texttt{Forbid}}$ waits for the answer of the corresponding $\mathcal{M}_{\texttt{Search}}$ before to continue the computation.

**Request sent by $\mathcal{M}_{\texttt{Search}}$** A Turing machine $\mathcal{M}_{\texttt{Search}}$ of level $n \geq 2$ can make a request at one of the four Turing machines $\mathcal{M}_{\texttt{Search}}$ of level $n-1$ localized in its dependency. The asking machine sends one bit composing the address every step of calculation, so that a level $n$ Turing machine sends a bit every $2^n$ rows and thus it takes $2^n * n$ rows to transfer the address of size $n$. The machine $\mathcal{M}_{\texttt{Search}}$ of level $n$

uses communication channels described in Fact 2.8 to communicate: each border computation box → and ← is surrounded by a rectangle of the same level $n$ which communicates with border computation box of the previous level $n-1$.

**Treatment of a request** A machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ successively responds to the different request tapes. The address of the request tape considered is copied on the computation tape, and the machine keeps in memory to which request tape it is responding. If the machine $\mathcal{M}_{\texttt{Search}}$ is of level 1, it directly reads the letter of $\mathcal{A}_\Sigma$. Otherwise the machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ transmits the address to the corresponding machine $\mathcal{M}_{\texttt{Search}}$ of level $n-1$: the first letter of the address indicates which channel $\mathcal{M}_{\texttt{Search}}$ must be used to send the continuation of the address, converted into a $n-1$ bits address by erasing the first bit of the address. Then the machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ waits for the answer, which is obtained when a machine of level 1 is reached (see Figure 14). This letter must be transferred back until it finds the machine which initially made the request.

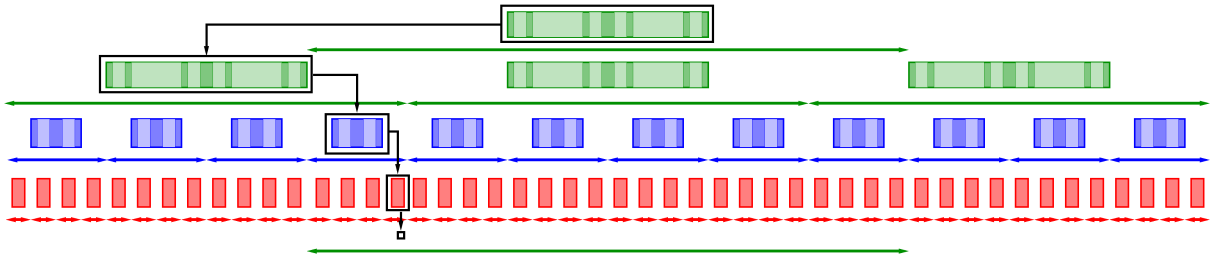


Figure 14: An example of request by a $\mathcal{M}_{\texttt{Forbid}}$ machine of level 3 – the computation zone on the top of the picture. Depending on the address of the letter requested, $\mathcal{M}_{\texttt{Forbid}}$ sends its request to either the left, center or right $\mathcal{M}_{\texttt{Search}}$ machine. On this example the $\mathcal{M}_{\texttt{Forbid}}$ machine sends its request to the left $\mathcal{M}_{\texttt{Search}}$ machine of level 3, which transmits it to a $\mathcal{M}_{\texttt{Search}}$ machine of level 2 and finally to a $\mathcal{M}_{\texttt{Search}}$ machine of level 1. This last machine can answer the request.

**Transfer back of the information** When a Turing machine $\mathcal{M}_{\texttt{Search}}$ obtains the bit corresponding to the request, it transfers it by the communication channel to the Turing machine which made the request via the request tapes. This operation is instantaneous for two reasons. First there is just one box of information to transmit. Secondly there is just one information on the channel since the corresponding Turing machine waits for an answer. A Turing machine $\mathcal{M}_{\texttt{Search}}$ eventually answers the request of the Turing machine $\mathcal{M}_{\texttt{Forbid}}$ of the same computation strip, since every $\mathcal{M}_{\texttt{Search}}$ alternately works for $\mathcal{M}_{\texttt{Forbid}}$ of same level and higher levels $\mathcal{M}_{\texttt{Search}}$ machines.

**Initialization of the computations** When the computation is initialized, it is important not to erase the addresses on the request tapes, because Turing machines of higher levels may be waiting for an answer. Requests are only made toward lower level, so they are answered even if the address does not correspond to a real request.

Another problem of initialization occurs when a Turing machine makes a request, but is initialized before to obtain its answer. Actually in this case we impose that once the Turing machine is initialized, it waits for the answer to its request from the previous computation before to begin a new one.

**Time taken by $\mathcal{M}_{\texttt{Search}}$ to answer at a request** Denote $t(n)$ the time that a machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ needs to answer a request from $\mathcal{M}_{\texttt{Forbid}}$. Since a machine of level $n$ makes a calculation step every $2^n$ rows, a machine $\mathcal{M}_{\texttt{Search}}$ of level $n$ needs $2^n * t(n)$ rows to answer a request from $\mathcal{M}_{\texttt{Forbid}}$.

A machine $\mathcal{M}_{\texttt{Search}}$ of level $n \geq 2$ needs the help of a machine $\mathcal{M}_{\texttt{Search}}$ of level $n-1$: it transfers one by one the $n-1$ bits of the address, one bit is transferred every $2^n$ rows, this takes $n * 2^n$ rows. Then it waits for the $\mathcal{M}_{\texttt{Search}}$ of level $n-1$ answer. It is possible that this $\mathcal{M}_{\texttt{Search}}$ of level $n-1$ is already busy, and the level $n$ machine has to wait – in the worst case three $\mathcal{M}_{\texttt{Forbid}}$ machines of level $n-1$ are already waiting for an answer. Hence the $\mathcal{M}_{\texttt{Search}}$ of level $n-1$ possibly works for the three neighbouring $\mathcal{M}_{\texttt{Forbid}}$ machines of level $n-1$, this takes $3 \times t(n-1)$ steps of calculation, before to work

for the $\mathcal{M}_{\texttt{Search}}$ of level $n$, this takes $t(n-1)$ steps of calculation. Thus the number of rows used to answer at a request is given by

$$2^n t(n) \leq n * 2^n + 4 * 2^{n-1} * t(n-1).$$

We deduce from the previous inequality that $t(n) \leq n^2 2^n$.

**Fact 3.3.** *All requests of $\mathcal{M}_{Forbid}$ of level $n$ are handled by the $\mathcal{M}_{Search}$ machine of same level in at most $n^2 2^n$ steps of calculation for large enough $n$.*

**Time taken by $\mathcal{M}_{\texttt{Forbid}}$ to check if a forbidden word appear** Assume that a Turing machine $\mathcal{M}_{\texttt{Forbid}}$ must check if a word $f$ of size $k$ appears in the responsibility zone associated. According to Section 3.4, this takes $6 * 4^{n-1} * k * t(n) \leq k * n^2 * 2^{3n+1}$ steps of calculation.

Let $(f_i)_{i\in\mathbb{N}}$ be the enumeration of forbidden patterns by $\mathcal{M}_{\texttt{Forbid}}$. Denote $t(f_0, \ldots, f_k)$ the time taken by $\mathcal{M}_{\texttt{Forbid}}$ to scan if the words $(f_i)_{i\in[0,k]}$ appear in the responsibility zone associated and denote $t'(f_0, \ldots, f_k)$ the time taken by $\mathcal{M}_{\texttt{Forbid}}$ to compute the words $(f_i)_{i\in[0,k]}$ without scaning the responsibility zone. Thus, the time taken by a Turing machine $\mathcal{M}_{\texttt{Forbid}}$ of level $n$ to scan the words $(f_i)_{i\in[0,k]}$ is given by

$$t(f_0, \ldots, f_k) \leq t'(f_0, \ldots, f_k) + (k+1) * \max\{|f_i| : i \in [0,k]\} * n^2 * 2^{3n+1}.$$

Since $t'(f_0, \ldots, f_k)$ does not depend of the level of $\mathcal{M}_{\texttt{Forbid}}$ and since by Fact 2.7 a machine $\mathcal{M}_{\texttt{Forbid}}$ of level $n$ could make $2^{2^n} + 2$ steps of calculation, there exists a level $n$ such that all Turing machines of level $n$ check that the words $(f_i)_{i\in[0,k]}$ does not appear in their responsibility zones.

**Fact 3.4.** *For all forbidden word of $\Sigma$, there exists $n \in \mathbb{N}$ such that every turing machine $\mathcal{M}_{Forbid}$ of level $n$ checks that the word does not appear in its responsibility zone.*

## 3.6 The final construction

We sum up the construction of the final subshift:

1. First, we construct the four layers: $\mathbf{T}_{\texttt{Level}} = \mathbf{Prod}\left(\mathcal{A}^{\mathbb{Z}^2}, \mathbf{T}_{\texttt{Grid}}, \mathcal{A}^{\mathbb{Z}^2}_{\mathrm{Comp}(\mathcal{M}_{\texttt{Forbid}})}, \mathcal{A}^{\mathbb{Z}^2}_{\mathrm{Comp}(\mathcal{M}_{\texttt{Search}})}\right)$;
2. then, we align all the letter of the first layer to obtain the same configuration horizontally $\mathbf{T}_{\texttt{Align}} = \mathbf{FT}_{\mathbf{Align}}\left(\mathbf{T}_{\texttt{Level}}\right)$;
3. finally, we include the working of $\mathcal{M}_{\texttt{Forbid}}$ and $\mathcal{M}_{\texttt{Search}}$ thanks to $\mathbf{Work}_{\mathcal{M}_{\texttt{Forbid}}} \cup \mathbf{Work}_{\mathcal{M}_{\texttt{Search}}}$ and we include the communication between the different layers thanks to **Com**. Moreover, we include the condition **Forbid** which exclude the configuration when $\mathcal{M}_{\texttt{Forbid}}$ encounters a forbidden pattern. We obtain:
$$\mathbf{T}_{\texttt{Final}} = \mathbf{FT}_{\mathbf{Work}_{\mathcal{M}_{\texttt{Forbid}}} \cup \mathbf{Work}_{\mathcal{M}_{\texttt{Search}}} \cup \mathbf{Com} \cup \mathbf{Forbid}}\left(\mathbf{T}_{\texttt{Align}}\right).$$

We denote by $\mathbf{T}$ the subshift $\mathbf{Fact}_{\pi}\left(\mathbf{SA}_{\mathbb{Z}e_1}\left(\mathbf{T}_{\texttt{Final}}\right)\right)$ where $\pi$ is a morphism that only keeps letters from alphabet $\mathcal{A}_{\Sigma}$ from the first layer. We want to compare $\Sigma$ and $\mathbf{T}$

**Any configuration in $\Sigma$ can be obtained ($\Sigma \subseteq \mathbf{T}$):** Let $x \in \Sigma$, by construction of $\mathbf{T}_{\mathrm{Final}}$ it is easy to construct a two-dimensional configuration $y$ such that $y \in \mathbf{T}_{\texttt{Final}}$ and $\pi(y_{|\mathbb{Z}e_1}) = x$.

**Any configuration constructed is in $\Sigma$ ($\mathbf{T} \subseteq \Sigma$):** Let $x \in \mathbf{T}$, we prove that $x \in \Sigma$. By definition there exists $y \in \mathbf{T}_{\mathrm{Final}}$ such that $\pi(y|_{\mathbb{Z}e_1}) = x$. It is sufficient to prove that every word in $x$ is in $\mathcal{L}(\Sigma)$. Let $w$ be a word that appears in $x$. Suppose that $w$ is not in $\mathcal{L}(\Sigma)$, by Fact 3.4, there exists $n \in \mathbb{N}$ such that in any computation strip of level $n$, the word $w$ is checked in the associated dependency strip. In particular the word $w$ will be compared with any word of length $|w|$ that appears in $x$. Since $w$ appears in $x$, there would be a computation strip of level $n$ in which the calculation of $\mathcal{M}_{\texttt{Forbid}}$ violates the finite type condition **Forbid**. This proves the inclusion $\mathbf{T} \subseteq \Sigma$.

### 3.7 Effective subshift as sub-action of a two-dimensional sofic

In fact the previous construction gives a more general result. If we consider

$$\begin{array}{rccc} \mathbf{SA}_{\mathbb{Z}e_1}: & \pi(\mathbf{T}_{\texttt{Final}}) & \longrightarrow & \Sigma \\ & x & \longmapsto & x_{\mathbb{Z}\times\{0\}} \end{array}$$

it is a continuous bijective map. Indeed, for all $x \in \pi(\mathbf{T}_{\texttt{Final}})$, one has $x_{(i,k)} = x_{(j,k)}$ for all $i, j, k \in \mathbb{Z}$ since by condition **Align** all columns contain the same symbol. Moreover, $\mathbf{SA}_{\mathbb{Z}e_1} \circ \sigma^{e_1} = \sigma_\Sigma \circ \mathbf{SA}_{\mathbb{Z}e_1}$, thus $\mathbf{SA}_{\mathbb{Z}e_1}$ realizes a conjugation between the dynamical system $(\pi(\mathbf{T}_{\texttt{Final}}), \sigma^{e_1})$ and $(\Sigma, \sigma_\Sigma)$. We deduce the following theorem:

**Theorem 3.5.** *Any effective subshift of dimension $d$ is conjugate to a sub-action of a sofic subshift of dimension $d+1$.*


## Acknowledgements

The authors are grateful to Michael Schraudner for useful discussions and important remarks about the redaction. We also want to thank the anonymous referee for his rigorous and detailed review which helped us to clarify the paper and Mike Boyle for some comments about sub-action concepts. Moreover, this research is partially supported by projects ANR EMC and ANR SubTile.


# References


[AS09] Nathalie Aubrun and Mathieu Sablik. An order on sets of tilings corresponding to an order on languages. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3, pages 99–110, 2009.

[Bea93] M.P. Beal. *Codage Symbolique*. Masson, 1993.

[Ber66] R. Berger. *The Undecidability of the Domino Problem*. American Mathematical Society, 1966.

[Boy08] M. Boyle. Open problems in symbolic dynamics. *Contemporary Mathematics*, 468:69–118, 2008.

[Dal74] Myers Dale. Nonrecursive tilings of the plane. ii. *The Journal of Symbolic Logic*, 39(2):286–294, 1974.

[DLS01] Bruno Durand, Leonid Levin, and Alexander Shen. Complex tilings. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 732–739, New York, NY, USA, 2001. ACM.

[DRS08] Bruno Durand, Andrei E. Romashchenko, and Alexander Shen. Fixed point and aperiodic tilings. In *Developments in Language Theory*, pages 276–288, 2008.

[DRS10] Bruno Durand, Andrei E. Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *CoRR abs/0910.2415, http://arxiv.org/abs/0910.2415*, 2010.

[Han74] William Hanf. Nonrecursive tilings of the plane. i. *The Journal of Symbolic Logic*, 39(2):283–285, 1974.

[Hed69] GA Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Theory of Computing Systems*, 3(4):320–375, 1969.

[Hoc09] M. Hochman. On the Dynamics and Recursive Properties of Multidimensional Symbolic Systems. *Inventiones Mathematicae*, 176(1):131–167, 2009.

[Kit98] B. Kitchens. *Symbolic dynamics*. Springer New York, 1998.

[LM95] D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

[Moz89] S. Mozes. Tilings, substitution systems and dynamical systems generated by them. *Journal d'analyse mathématique(Jerusalem)*, 53:139–186, 1989.

[PS10] R. Pavlov and M. Schraudner. Classification of sofic projective subdynamics of multidimensional shifts of finite type. *Submitted*, 2010.

[RJ87] H. Rogers Jr. *Theory of recursive functions and effective computability*. MIT Press Cambridge, MA, USA, 1987.

[Rob71] R.M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.