PIMP MY FIXPOINT: SOFIC REALIZATION OF MULTIDIMENSIONAL SUBSTITUTION-BASED SHIFT SPACES

ANTONIN CALLARD, LÉO PAVIET SALOMON, AND PASCAL VANIER

Abstract. In symbolic dynamics, the fixed point construction from [DRS12] defines shift spaces of finite type whose configurations embed infinite hierarchies of tilings. This article provides a "black box" abstraction of this method phrased in terms of substitutions and S-adic limit spaces operating over sequences of increasingly large alphabets. By quantifying the amount of information computed by the substitutions at each level, and using a suitable parallel model of computation, we provide a simple positive criterion of multidimensional soficity that generalizes classical examples from the literature.

1. Introduction

Shift spaces are sets of colorings (or "configurations") of a discrete space (usually $\mathbb{Z}^d$) with a finite alphabet of colors $\mathcal{A}$ that abide to some family of forbidden patterns. Most commonly, this family is composed of finitely many patterns, and thus defines a shift of finite type (SFT). Perhaps surprisingly, these shifts may already exhibit complex behaviors: it is undecidable whether they are empty [Ber64], they may have only non-computable configurations [Han74; Mye74], they may contain only maximally complex configurations in the sense of Kolmogorov complexity [DLS08]...

On the other side of the complexity spectrum are the effective shifts, whose forbidden constraints can be enumerated algorithmically. Classically, the most considered intermediate class of shifts is the class of *sofic shifts*, which are defined as the cellwise recolorings of the shifts of finite type (their *covers*). While they still admit a finite description, they form a superclass of SFTs that exhibit strictly more complex behaviors: for instance, the shift on the alphabet $\{\square, \blacksquare\}$ whose tilings contain at most a single $\blacksquare$ cell is sofic but not of finite type. In fact, sofic shifts can manifest computationally complex properties: as illustrated by a landmark result of M. Hochman [Hoc09], arbitrary effective shifts can be realized as subsystems of higher-dimensional sofic shifts.

While sofic shifts form a strict subset of effective shifts, and enjoy a simple algebraic characterization on $\mathbb{Z}$ [LM95, Chapter 3.2], they do not admit such a straightforward separation criterion in dimension $d \geq 2$. Indeed, the only arguments known to the authors proving that an effective shift is not sofic are based on information-theoretic tools quantifying the amount of information inside their patterns.

More precisely, since shifts of finite type are defined by finite families of forbidden patterns, the validity of patterns in an SFT is a local property. In a shift of finite type, the compatibility of a pattern of domain $\{0, \dots, n\}^d$ with its exterior is thus constrained by the size of their sufficiently thick border, which is only composed of $\mathcal{O}(n^{d-1})$ cells. With only a finite alphabet, this border cannot embed enough information to fully describe the interior of large enough patterns. As cellwise projections of SFTs, this $\mathcal{O}(n^{d-1})$ information bound still applies to sofic shifts.

The textbook example of this obstruction is the 2-dimensional mirror shift: on the alphabet $\{0, 1, \blacksquare\}$, only a single $\blacksquare$ symbol can appear on each line, all $\blacksquare$ symbols must constitute an entire column, and the two induced $\{0, 1\}$-colored half-planes must be mirrors of one another. The mirror shift defines 2^{n^2} binary $\{0, 1\}$-colored patterns of size $n \times n$, which must be mirrored across the $\blacksquare$-column. However large the alphabet of a cover for the mirror shift, these borders are too small to transmit the description of their interior to the other side with only SFT constraints.

Date: Version of April 10, 2026.

2020 *Mathematics Subject Classification.* Primary 37B10; Secondary 68Q09.

The authors want to acknowledge the many discussions they shared with Pierre Guillon about this work, ranging from his experience on the fixed point construction while collaborating with Charalampos Zinoviadis to potential properties of interests on the finite type covers generated by Lemma 5.2.

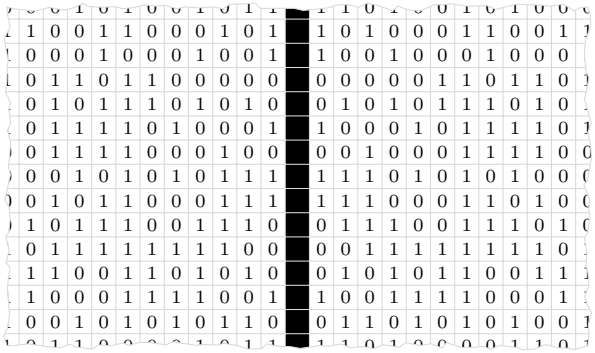


FIGURE 1. A typical tiling of the mirror shift.

The quest to find the exact separation between sofic and strictly effective shifts has found many examples of sofic multidimensional shifts: substitutive shifts [Moz89]; S-adic effective shifts [AS14]; seas of squares [Wes17]; all effective shifts of low density [Des21]...

All these examples can actually be recovered using ad-hoc modifications of the so called *fixed point construction* of tilings [DRS12]. Based on a fixed point theorem from computability theory, this construction defines shifts of finite type whose configurations simulate an infinite hierarchy of layers. Inside each layer, a simulation groups the tiles into finite blocks, called *macro-tiles*, in which the computations of a Turing machine emulate a tile of the next level.

Furthermore, this construction is entirely algorithmic: in order to change the tiling simulated at some level, one only needs to change its "programming" drawn in the macro-tiles of the previous layer. This versatility certainly explains the variety of shifts which have been proved sofic by this construction. For example, if a substitution is a map $\tau\colon \mathcal{A} \to \mathcal{A}^D$ that transforms individual cells into finite rectangular patterns, then the resulting *substitutive shift* (in which every tiling admits preimages for the sequence of substitutions τ, τ^2, τ^3 ...) is actually sofic [Moz89]. For every computable (or even Π^0_1-computable) set of integers $S \subseteq \mathbb{N}$, [Wes17] proves the soficity of the seas of squares of edge lengths S (where a sea of square is a tiling drawing independent ■-squares over a □ background). For every $\alpha < 1$, [Des21] proves the soficity of every effective shift $X \subseteq \{□, ■\}^{\mathbb{Z}^2}$ in which every $n \times n$ pattern contains at most $\mathcal{O}(n^\alpha)$ ■-colored cells.

Unfortunately, each new example has required (sometimes substantial) modifications of the original fixed point construction to fit its specificities. Nevertheless, a key argument in all these results appears to be sublinear bounds on the amount of "useful information" contained inside the patterns of the associated tilings.

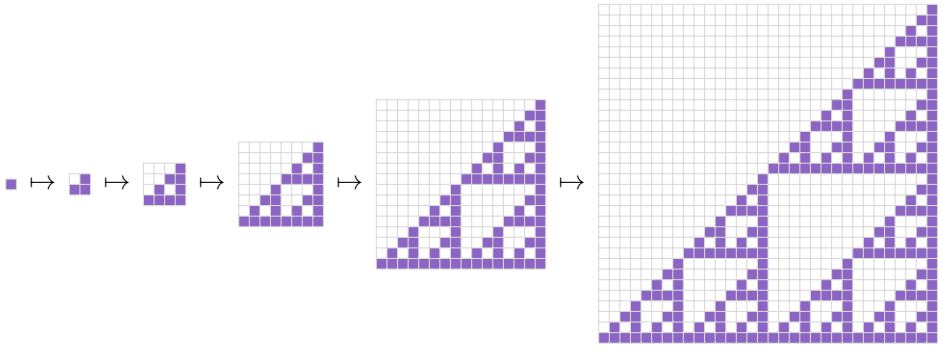


FIGURE 2. A few iterations of the substitution τ: ■ $\mapsto$, □ $\mapsto$ realizing the Sierpiński triangle.

In this article, we achieve a generalization of the fixed point theorem that covers all these previous applications while abstracting the underlying construction under the formalism of *substitutions* and *S-adic shifts* (where a shift space is *S-adic* if its tilings admit preimages for by an infinite sequence of substitutions τ_1, $\tau_1 \circ \tau_2$, $\tau_1 \circ \tau_2 \circ \tau_3 \dots$). Informally, we prove under some computability conditions that a sequence of substitutions operating on alphabets of sublinear sizes (with respect to pattern domains) results in an *S-adic shift space* that is actually sofic.

More precisely, our main results include:

- A "black box" lemma for the fixed point construction (Lemma 5.2) phrased in terms of a computable substitution operating on an infinite alphabet of Wang tiles and generating infinite sequences of valid tilings.
- Two applications of this lemma on limit shift spaces generated by infinite sequences of *dill maps*: Theorem 6.7 operating on square substitutions of possibly non-monotonous sizes, and Theorem 6.8 operating on rectangular substitutions of bounded sizes.

Dill maps [ST15] generalize both morphisms and classical substitutions by substituting symbols differently depending on the colors of their neighbors. A sequence of dill maps $(\tau_\ell)_{\ell \geq 1}$ can define a limit space in the same ways substitutions do, which we call a *D-adic shift space*. Informally, D-adic systems allow to consider S-adic configurations satisfying some intermediate validity conditions, *e.g.* limit configurations

$$x = x^{(0)} \xleftarrow{\tau_1} x^{(1)} \xleftarrow{\tau_2} x^{(2)} \xleftarrow{\tau_3} x^{(3)} \dots$$
$$\in X_0 \qquad \in X_1 \qquad \in X_2 \qquad \in X_3$$

in which all intermediate configurations $x^{(\ell)}$ are valid in some shifts of finite type X_ℓ.

While Theorems 6.7 and 6.8 are weaker versions of Lemma 5.2, they actually are both easier to manipulate and general enough to recover classical applications of the fixed point construction (*e.g.* [DRS12; AS14; Wes17; Des21]). Indeed, the structure of macro-tiles in the fixed point construction naturally induces a sequence of substitutions $\tau_\ell \colon \mathcal{A}_\ell \mapsto \mathcal{A}_{\ell-1}^{\{0,\dots,N_\ell\}^d}$, each simulating a tile from the ℓ^{th} level by a block of size $\{0, \dots, N_\ell\}^d$ on the previous layer. Lemma 5.2 admits the following technical improvements over previous applications from the literature:

- Classically, macro-tiles of domain $\{0, \dots, N_\ell\}^d$ embed the computations emulating the next layer of the simulation. Thus, the size N_ℓ must usually grow fast enough to embed increasingly more computations, *e.g.* $N_\ell = 2^{2^\ell}$.
  In Lemma 5.2, we actually allow a large variety of possibly non-monotonous sequences $(N_\ell)_{\ell \in \mathbb{N}}$, for example such that $N_\ell = 2$ for infinitely many ℓ's. To bypass the lack of computational space in the ℓ^{th} layer, we thus embed the computations for the layer of level $\ell + 1$ into some deeper layer $\ell' \leq \ell$, which increases the overall complexity of trans-layer communication in the construction.
- Classically, the fixed point construction is defined with square macro-tiles (*i.e.* square substitutions). Similarly to [Zin16], we relax this condition by allowing rectangular macro-tiles of possibly unbounded eccentricity;
- Classically, computability conditions on the next layer of the fixed point construction are usually given in terms of N_ℓ, *i.e.* the size of macro-tiles on the current level of index ℓ. Since Lemma 5.2 actually allows size $N_\ell = 2$ arbitrarily often, we must give more flexible conditions: we formulate them in terms of $\prod_{i=1}^{\ell} N_i$, *i.e.* in terms of the size of a tile t_ℓ after ℓ steps of substitutions $\tau_1 \circ \dots \circ \tau_\ell(t_\ell)$.
- Classically, the fixed point construction is defined in dimension $d = 2$. Generalizations to higher dimensions are often claimed possible, although it is not obvious how the classical computational embedding (space-time diagrams of Turing machines) should be adapted. In this article, we rely on another model of computation called *processor arrays* (see Section 3.2). By operating on multidimensional grids of processors, it can naturally process in time $\mathcal{O}(N)$ inputs of size $\mathcal{O}(N^{d-1})$. As far as we know, this is the first use of this computation model in multidimensional symbolic dynamics.

For convenience, we state the computational conditions of our results in the log-RAM model of computation, as the associated time complexity closely resembles those of "real-world" programming languages (and is thus easier to compute than with Turing machines).

§ Outline of the paper

- Section 2 provides classical background in symbolic dynamics, while Section 3 adapts computability theory results to the setting of log-RAM machines and processor arrays;
- Section 4 develops some necessary tools for the proof of Lemma 5.2, and in particular a notion of *wirings* and the associated routing lemma;
- Section 5 is dedicated to our main technical Lemma 5.2. The entire section motivates, states and proves this result. As our central result and proof, Section 5 spans most of this article.
- Section 6 introduces the notion of substitutions and dill maps; Theorems 6.7 and 6.8 relate Lemma 5.2 to other dynamical notions from the literature about shift spaces: namely, S-adic and D-adic limit spaces;
- Section 7 illustrates our results by considering their applications to classical examples [DRS12; Wes17; Des21] and some generalizations.

§ Notations and conventions

In dimension $d \in \mathbb{N}$, we will often denote $\boldsymbol{i} = (i_1, \dots, i_d) \in \mathbb{Z}^d$ an element of $\mathbb{Z}^d$, and $\boldsymbol{e}^1, \dots, \boldsymbol{e}^d$ the standard basis. For $n_1, \dots, n_d \in \mathbb{N}$, we denote by $[\![n_1, \dots, n_d]\!]$ the product of intervals $\{0, \dots, n_1 - 1\} \times \cdots \times \{0, \dots, n_d - 1\}$.

Rectangles. In dimension $d \in \mathbb{N}$, we denote by $\mathfrak{R}_0 = \{[\![n_1, \dots, n_d]\!] : (n_1, \dots, n_d) \in \mathbb{N}^d\}$ the set of rectangles of bottom-left corner $\mathbf{0}$; and $\mathfrak{R} = \bigcup_{\boldsymbol{i} \in \mathbb{Z}^d} \boldsymbol{i} + \mathfrak{R}_0$ the set of d-dimensional rectangles. We will often denote individual rectangles $\mathfrak{r} \in \mathfrak{R}$; and for any rectangle $\mathfrak{r} \in \mathfrak{R}$, we define $[\mathfrak{r}]$ the *normalization* of $\mathfrak{r}$, *i.e.* the unique translation of $\mathfrak{r}$ that belongs to $\mathfrak{R}_0$. For any position $\boldsymbol{i} = (i_1, \dots, i_d) \in \mathbb{Z}^d$ and rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$, we denote by $\boldsymbol{i} \bmod \mathfrak{r}$ the position $(i_1 \bmod n_1, \dots, i_d \bmod n_d) \in \mathfrak{r}$. For a given rectangle $\mathfrak{r} \in \mathfrak{R}_0$, a *cut* of $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$ is a rectangle $\mathfrak{c} = [\![n'_1, \dots, n'_d]\!]$ such that $n'_1 \leq n_1, \dots, n'_d \leq n_d$, *i.e.* a rectangle $\mathfrak{c} \in \mathfrak{R}_0$ such that $\mathfrak{c} \subseteq \mathfrak{r}$.

For a given rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$, we will denote $\lambda(\mathfrak{r})$ and $\mu(\mathfrak{r})$ the *time* and *space* of $\mathfrak{r}$, which respectively refer to the length $\lambda(\mathfrak{r}) = \max\{n_k : 1 \leq k \leq d\}$ of a longest edge and to the area $\mu(\mathfrak{r}) = \frac{1}{\lambda(\mathfrak{r})} \prod_{k=1}^{d} n_k$ of a smallest facet in $\mathfrak{r}$; and denote *volume* by $\|\mathfrak{r}\| = \prod_{k=1}^{d} n_k$.

Finally, we denote by $f_k^-(\mathfrak{r}) = \{\boldsymbol{i} \in \mathfrak{r} : \boldsymbol{i}_k = 0\}$ and $f_k^+(\mathfrak{r}) = \{\boldsymbol{i} \in \mathfrak{r} : \boldsymbol{i}_k = n_k - 1\}$ the $(d-1)$-dimensional facets of minimal and maximal index in $\mathfrak{r}$ along the direction $\boldsymbol{e}^k$ for $1 \leq k \leq d$. In this paper, we will often refer to *smallest facets*, *i.e.* facets of smallest area $\mu(\mathfrak{r})$. Since there are several of them, we arbitrarily pick a convention and denote by $f(\mathfrak{r})$ the facet $f_h^-(\mathfrak{r})$ of smallest index $1 \leq h \leq d$ that has area $\mu(\mathfrak{r})$.

Boustrophedon order. The *boustrophedon order*[1] on $[\![n_1, \dots, n_d]\!]$ is defined inductively: an array $a \in \mathbb{N}^{[\![n_1]\!]}$ is sorted in boustrophedon order if $a_i \leq a_{i+1}$ for all $i \in \{0, \dots, n_1 - 2\}$; and a d-dimensional array $a \in \mathbb{N}^{[\![n_1, \dots, n_d]\!]}$ is sorted in boustrophedon order if

i. Each subarray $a|_{\{i\} \times [\![n_2, \dots, n_d]\!]}$ is sorted in increasing (resp. decreasing) boustrophedon order if $i \in \{0, \dots, n_1 - 1\}$ is even (resp. odd);
ii. Adjacent subarrays are themselves sorted, *i.e.* $a_{i,\boldsymbol{j}} \leq a_{i+1,\boldsymbol{j'}}$ for every $i \in \{0, \dots, n_1 - 1\}$ and $\boldsymbol{j'}, \boldsymbol{j'} \in [\![n_2, \dots, n_d]\!]$.

We are especially interested in the boustrophedon order since it preserves locality: for any rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$ and for $n < \prod_{k=1}^{d} n_k$, denote by $\boldsymbol{i}$ and $\boldsymbol{j}$ the positions of index n and $n+1$ in the boustrophedon ordering of $\mathfrak{r}$; then $\boldsymbol{i}$ and $\boldsymbol{j}$ denote two adjacent positions in $\mathfrak{r}$.

Grids. We call *grid* a partition $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{Z}^d}$ of $\mathbb{Z}^d$ into facet-adjacent disjoint rectangles (*i.e.* for all $\boldsymbol{i} \in \mathbb{Z}^d$ and $1 \leq k \leq d$, the facets $f_k^+(\mathfrak{r}_{\boldsymbol{i}})$ and $f_k^-(\mathfrak{r}_{\boldsymbol{i}+\boldsymbol{e}^k})$ match). For $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{Z}^d}$ a grid and $\boldsymbol{j} \in \mathbb{Z}^d$ a position, there exists a unique $\boldsymbol{i} \in \mathbb{Z}^d$ such that $\boldsymbol{j} \in \mathfrak{r}_{\boldsymbol{i}}$; and we denote by $\boldsymbol{j} \bmod \mathfrak{g} \in \mathbb{N}^d$ the translation of $\boldsymbol{j} \in \mathfrak{r}_{\boldsymbol{i}}$ in the normalization $[\mathfrak{r}_{\boldsymbol{i}}] \in \mathfrak{R}_0$.

Generalizing the previous notions of time and space, the *time* of $\mathfrak{g}$ will denote the supremum $\lambda(\mathfrak{g}) = \sup_{\boldsymbol{i} \in \mathbb{Z}^d} \lambda(\mathfrak{r}_{\boldsymbol{i}})$ and the *space* of $\mathfrak{g}$ will denote the infimum $\mu(\mathfrak{g}) = \inf_{\boldsymbol{i} \in \mathbb{Z}^d} \mu(\mathfrak{r}_{\boldsymbol{i}})$.

[1] From the Greek "βουστροφηδόν" (lit. "in the way an ox turns [while plowing]"), it is also called the *snake* or the *shear order*; thus illustrating a mathematician's love for bucolic metaphors.

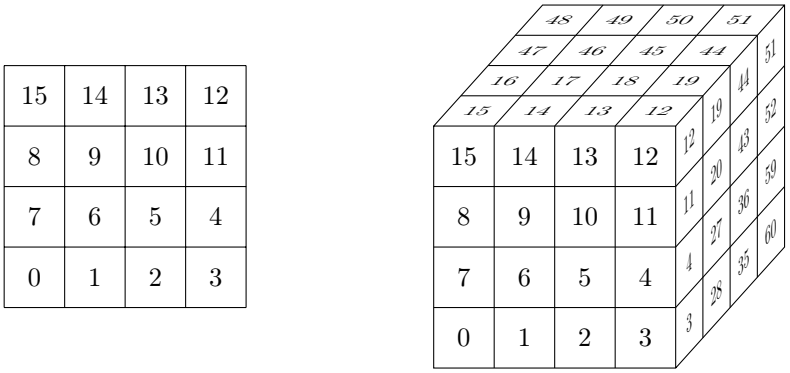


FIGURE 3. Arrays on $[\![4]\!]^d$ for $d = 2$ and $d = 3$ sorted in boustrophedon ordering.

We also define the following *composition*: for a rectangle $\mathfrak{r}' \in \mathfrak{R}$ and a family of rectangles $(\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathfrak{r}'}$ indexed by the positions $\boldsymbol{i} \in \mathfrak{r}'$, we define $\mathfrak{r}' \circ (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathfrak{r}'}$ as the rectangle $\tilde{\mathfrak{r}} = \bigcup_{\boldsymbol{i}\in\mathfrak{r}} \mathfrak{r}'_{\boldsymbol{i}}$. Generalizing from individual rectangles to complete grids, we define $\mathfrak{g}' \circ \mathfrak{g} = (\mathfrak{r}'_{\boldsymbol{i}} \circ (\mathfrak{r}_{\boldsymbol{j}})_{\boldsymbol{j}\in\mathfrak{r}'_{\boldsymbol{i}}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ for $\mathfrak{g}' = (\mathfrak{r}'_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ and $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{j}})_{\boldsymbol{j}\in\mathbb{Z}^d}$; in other words, $\mathfrak{g} \circ \mathfrak{g}'$ is the grid of individual rectangles $\bigcup_{\boldsymbol{j}\in\mathfrak{r}'_{\boldsymbol{i}}} \mathfrak{r}_{\boldsymbol{j}}$ for $\boldsymbol{i} \in \mathbb{Z}^d$. When composition multiple grids $(\mathfrak{g}_i)_{1\le i\le n}$, we denote by $\bigcirc_{i=1}^{n} \mathfrak{g}_i = \mathfrak{g}_n \circ \cdots \circ \mathfrak{g}_1$.

Notice that $\tilde{\mathfrak{g}} = \mathfrak{g}' \circ \mathfrak{g}$ is a *subgrid* of $\mathfrak{g}$ (we also say that $\mathfrak{g}$ is *nested* inside $\tilde{\mathfrak{g}}$), in the sense that $\tilde{\mathfrak{g}}$ is a coarser partition of $\mathbb{Z}^d$ than $\mathfrak{g}$. This elementary observation will become especially relevant when we will later consider *sequences of grids* $(\mathfrak{g}_i)_{1\le i\le n}$: the products $\tilde{\mathfrak{g}}_i = \bigcirc_{j=1}^{i} \mathfrak{g}_j$ define a sequence of *nested* grids, each $\tilde{\mathfrak{g}}_{i+1}$ being a subgrid of $\tilde{\mathfrak{g}}_i$.

2. SYMBOLIC DYNAMICS

Shift spaces. For a finite *alphabet* $\mathcal{A}$, and a *dimension* $d \in \mathbb{N}$, a *pattern* is a coloring $w\colon D \to \mathcal{A}$ of a *domain* $D \subseteq \mathbb{Z}^d$ (denoted $\mathrm{dom}(w) = D$) with symbols from $\mathcal{A}$; and we denote by $w_{\boldsymbol{p}} \in \mathcal{A}$ the symbol of $\mathcal{A}$ appearing at position $\boldsymbol{p} \in \mathrm{dom}(w)$. In case the domain $\mathrm{dom}(w)$ is finite, w is called a *finite* pattern; and in case the domain $\mathrm{dom}(w)$ equals the whole space $\mathbb{Z}^d$, w will often be called a *configuration*. For w a finite pattern over the alphabet $\mathcal{A}$ and for $a \in \mathcal{A}$, we denote $|w|_a = |\{\boldsymbol{i} \in \mathrm{dom}(w) : w_{\boldsymbol{i}} = a\}|$ the number of symbols a in w.

Abusing notations, we denote by $\mathcal{A}^{\mathfrak{R}}$ the set of d-dimensional rectangular finite patterns, *i.e.* $\mathcal{A}^{\mathfrak{R}} = \bigcup_{\mathfrak{r}\in\mathfrak{R}} \mathcal{A}^{\mathfrak{r}}$ (and $\mathcal{A}^{\mathfrak{R}_0} = \bigcup_{\mathfrak{r}\in\mathfrak{R}_0} \mathcal{A}^{\mathfrak{r}}$); and by $\mathcal{A}^{*d}$ (resp. $\mathcal{A}^{\circledast d}$) the set of arbitrarily-shaped d-dimensional finite (resp. possibly infinite) patterns. The set of all d-dimensional configurations over $\mathcal{A}$ is denoted $\mathcal{A}^{\mathbb{Z}^d}$.

A pattern u is said to *appear* in a pattern v (or, conversely, that v *contains* the pattern u), denoted $u \sqsubseteq v$, if there exists a position $\boldsymbol{p_0} \in \mathbb{Z}^d$ such that $(\boldsymbol{p_0} + \mathrm{dom}(u)) \subseteq \mathrm{dom}(v)$ and $\forall \boldsymbol{i} \in \mathrm{dom}(u)$, $u_{\boldsymbol{p}} = v_{\boldsymbol{p_0}+\boldsymbol{p}}$. For $w \in \mathcal{A}^{\circledast d}$ a pattern and $S \subseteq \mathrm{dom}(w)$, we denote by $w|_S \in \mathcal{A}^S$ the coloring it induces on S. For two patterns $u, v \in \mathcal{A}^{\circledast d}$, we denote $u = v$ if the two patterns are equal and $\mathrm{dom}(u) = \mathrm{dom}(v)$; and $u \equiv v$ if u and v are equal up to translation (*i.e.* $u \sqsubseteq v$ and $v \sqsubseteq u$). In particular, if $w \in \mathcal{A}^{\mathfrak{R}}$ is a rectangular pattern, we denote $[w]$ the unique pattern in $\mathcal{A}^{\mathfrak{R}_0}$ such that $w \equiv [w]$.

The space $\mathbb{Z}^d$ acts on patterns by translation. Equipping $\mathcal{A}$ with a discrete topology, and $\mathcal{A}^{\mathbb{Z}^d}$ with a product topology, the set of configurations $\mathcal{A}^{\mathbb{Z}^d}$ is a Cantor space; and its subsystems are:

Definition 2.1 (Shift space)**.** A *shift space* is a closed and translation-invariant subset of $\mathcal{A}^{\mathbb{Z}^d}$. Equivalently, a subset $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is a shift space if and only if there exists a (possibly infinite) family of forbidden finite patterns $\mathcal{F}$ such that $X = X_{\mathcal{F}}$, where

$$X_{\mathcal{F}} = \{x \in \mathcal{A}^{\mathbb{Z}^d} : \forall w \in \mathcal{F},\ w \not\sqsubseteq x\}.$$

A family of forbidden finite patterns $\mathcal{F}$ such that $X = X_{\mathcal{F}}$ may be considered as a *presentation* of the shift space X. As usual with presentations, two different families of forbidden finite patterns may define the same shift space.

Factor maps. The morphisms that preserve the structure of shift spaces as dynamical systems are the *factor maps*:

Definition 2.2 (Morphism)**.** For $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ and $Y \subseteq \mathcal{B}^{\mathbb{Z}^d}$ two shift spaces, a map $\varphi\colon X \to Y$ is a *morphism* if it is continuous and translation-equivariant map.

Equivalently, $\varphi\colon X \to Y$ is a morphism if it is the block map associated with a cellular automaton, *i.e.* if there exists a finite neighborhood $N \subseteq \mathbb{Z}^d$ and a local rule $f\colon \mathcal{A}^N \to \mathcal{B}$ such that $\varphi(x)_{\boldsymbol{p}} = f(x|_{\boldsymbol{p}+N})$ for every $x \in X$ and every $\boldsymbol{p} \in \mathbb{Z}^d$.

Definition 2.3 (Factor and conjugacy)**.** For $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ and $Y \subseteq \mathcal{B}^{\mathbb{Z}^d}$, a surjective morphism $\varphi\colon X \to Y$ is called a *factor map*; in which case, Y is said to be a *factor* of X (or, conversely, that X is a *cover* of Y).

If φ is a bijective factor map, φ^{-1} can also be shown to be a factor map [Hed69, Curtis-Hedlund-Lyndon theorem], and φ is then called a *conjugacy*; in which case, X and Y are said to be *conjugate*.

Classes of shift spaces. Shift spaces are traditionally classified depending on the complexity of their presentations:

- A shift space $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is *local* if there exists a set of local validity rules $V_1, \dots, V_d \subseteq \mathcal{A}^2$ such that $X = \{x \in \mathcal{A}^{\mathbb{Z}^d} : \forall \boldsymbol{p} \in \mathbb{Z}^d, \forall k \in \{1, \dots, d\}, (x_{\boldsymbol{p}}, x_{\boldsymbol{p}+\boldsymbol{e}^k}) \in V_k\}$;[2,3]
- A shift space $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is *of finite type* (Shift of Finite Type, SFT) if there exists a finite family of forbidden finite patterns $\mathcal{F}$ such that $X = X_{\mathcal{F}}$;
- A shift space $Y \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is *sofic* if it is the factor of a shift of finite type X, called an *SFT cover* of Y;
- A shift space $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is *effective* if there exists a computably enumerable family of forbidden finite patterns $\mathcal{F}$ such that $X = X_{\mathcal{F}}$.

The classes of shifts of finite type, sofic shifts and effective shifts are all closed under conjugacy, and strictly included in one another:

$$[\text{SFTs}] \subsetneq [\text{Sofic shifts}] \subsetneq [\text{Effective shifts}].$$

The local shifts are conjugated to shifts of finite type, which implies in particular that sofic shifts admit an equivalent definition: a shift $Y \subseteq \mathcal{A}^{\mathbb{Z}^d}$ is sofic if there exists a local shift space $X \subseteq \mathcal{B}^{\mathbb{Z}^d}$ (called a *local cover* of Y) and a letter-by-letter projection $\pi\colon \mathcal{B} \to \mathcal{A}$ such that $\pi(X) = Y$.

Wang tiles. For $\mathcal{C}$ a set of colors, a *(decorated) Wang tile* is a tuple $t = (c_0, \dots, c_{2d}) \in \mathcal{C}^{2d+1}$. We identify Wang tiles with colored unit cubes by defining $f_{\bullet}(t) = c_d$ the *decoration* of t, and for any $1 \leq k \leq d$ and direction $\pm$, denoting $f_k^{\pm}(t)$ the color $c_{d \pm k}$ of the facet $f_k^{\pm}$. A *Wang tileset* is then a set of tiles $T \subseteq \mathcal{C}^{2d+1}$.

Any Wang tileset T defines a local set of valid configurations in which any two adjacent tiles bear the same color on their common facet: $X_T = \{x \in T^{\mathbb{Z}^d} : \forall \boldsymbol{i} \in \mathbb{Z}^d, \forall 1 \leq k \leq d\, f_k^+(x_{\boldsymbol{i}}) = f_k^-(x_{\boldsymbol{i}+\boldsymbol{e}^k})\}$. If T is *finite*, then X_T is compact and thus defines a local shift space.

In this article, we consider *decorated* Wang tiles for two reasons. First, they allow some notational and writing convenience in the proof of our main Lemma 5.2. Second, the configurations of a sofic shift can actually be obtained by projecting the tilings on some decorated Wang tileset T to their decorations.

Substitutions. Recall that for $\mathcal{A}$ an alphabet, $\mathcal{A}^{\mathfrak{R}_0}$ is the set normalized d-dimensional rectangular patterns over the alphabet $\mathcal{A}$. We define *(rectangular) substitutions* as follows:

Definition 2.4 (Substitution)**.** Given two alphabets $\mathcal{A}$ and $\mathcal{B}$, a d-dimensional *rectangular substitution* is a non-deterministic map $\tau\colon \mathcal{A} \rightrightarrows \mathcal{B}^{\mathfrak{R}_0}$.
A substitution τ extends from individual symbols of $\mathcal{A}$ to whole configurations as follows: for $x \in \mathcal{A}^{\mathbb{Z}^d}$, we define $\tau(x)$ as the set of configurations $y \in \mathcal{B}^{\mathbb{Z}^d}$ for which there exists a grid $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{Z}^d}$ such that $[y|_{\mathfrak{r}_{\boldsymbol{i}}}] \in \tau(x_{\boldsymbol{i}})$.

[2]Recall that $\boldsymbol{e}^k$ is the k^{th} standard unit vector of $\mathbb{Z}^d$.

[3]Local shift spaces are also known as the *nearest neighbor* shifts of finite type.

Notice that our substitutions are non-deterministic, non-uniform (*i.e.* all image patterns do not necessarily have the same shape), and not necessarily total (that is, $\tau(x)$ might be empty for some configurations x). Traditionally, substitutions are extended from individual letters to whole configurations by assuming or providing some gluing/compatibility conditions on the images of adjacent letters (see *e.g.* [AS14]): in our definition, we circumvent this issue by only considering in $\tau(x)$ the configurations that explicitly admit a desubstitution along a grid.

Finally, we say that a substitution $\tau : \mathcal{A} \rightrightarrows \mathcal{B}^{\mathfrak{R}_0}$ is *growing* if, for every letter $a \in \mathcal{A}$ and pattern $w \in \tau(a)$, the domain $\mathrm{dom}(w) = [\![n_1, \dots, n_d]\!] \in \mathfrak{R}_0$ of w satisfies $n_k \geq 2$ for every $1 \leq k \leq d$. *In the rest of this paper, we always assume that a multidimensional substitution is growing.*

3. Computation models

Theoretic computations are usually abstracted from the specifics of a computation model, since most models (*e.g.* Turing machines, variations of register machines…) end up defining the same class of polynomial-time computable functions. Unfortunately, this article is not interested in time complexity as a theory, but rather as the precise number of operations performed by an algorithm, which depends heavily on the choice of computation model.

In these sections, we consider two computation models: log-*RAM machines*, which are very close to real-world programming languages and thus define a somewhat intuitive measure of time complexity; and *processor arrays*, which form a very geometric model of computation that appears especially suited for computation embeddings into multidimensional tilings.

§ 3.1. The RAM model

A *Random Access Machine* (RAM) is composed of a *memory* $\mathtt{M} = (\mathtt{M}[i])_{i\in\mathbb{N}}$, which consists of infinitely many *memory cells* holding a word[4] $\mathtt{M}[i] \in \{0,1\}^*$; and a finite *input* $\mathtt{I} \in \{0,1\}^{**}$ and *output* $\mathtt{O} \in \{0,1\}^{**}$. Such a machine executes a *program*, which consists of a finite set of *variables* $V = (\mathtt{var}_i)_{i\in I}$ (which each hold a word $\mathtt{var}_i \in \{0,1\}^*$) and a finite sequence of *instructions*. A *program counter* $\mathtt{PC} \in \mathbb{N}$ refers to (the index of) the current instruction of the program.

The instructions of a non-deterministic RAM machine are typically divided into:

- *Arithmetic operations* (such as add, subtract, copy, multiply, constant…) operating directly on the variables of V, typically $\mathtt{var}_i \leftarrow \mathtt{var}_j\ \mathtt{op}\ \mathtt{var}_k$ (resp. $\mathtt{var}_i \leftarrow \mathtt{op}\ \mathtt{var}_j$);
- A *non-deterministic* instruction $\mathtt{var}_i \leftarrow \mathtt{NDET}(\mathtt{var}_j)$ that non-deterministically sets the variable $\mathtt{var}_i$ to any integers in the interval $\{0, \dots, \mathtt{var}_j\}$;
- *Control flow instructions* (such as halt, conditional $\mathtt{GOTO}$…) whose instructions operate on the program counter $\mathtt{PC}$ to allow for (conditional) branching of the execution;
- *Memory input/output*, which consists in transferring data between the variables V and either the memory ($\mathtt{LOAD} : \mathtt{var}_i \leftarrow \mathtt{M}[\mathtt{var}_j]$ and $\mathtt{STORE} : \mathtt{M}[\mathtt{var}_i] \leftarrow \mathtt{var}_j$), the input or the output ($\mathtt{READ} : \mathtt{var}_i \leftarrow \mathtt{I}[\mathtt{var}_j][\mathtt{var}_k]$ and $\mathtt{WRITE} : \mathtt{O}[\mathtt{var}_i][\mathtt{var}_j] \leftarrow \mathtt{var}_k$).

We do not write the exact set of arithmetic and control flow instructions of our RAM model, as it would not yield much insight into the results of this paper. We informally assume that they allow to perform classical arithmetic operations (addition, multiplication, division…) and conditional branching instructions in constant time.

As with classical programming languages, *programs* will be represented as a binary word $e \in \{0,1\}^*$. By considering the associations of inputs and outputs obtained by running the program $e \in \{0,1\}^*$, we obtain a non-deterministic partial function:

Definition 3.1 (Computability)**.** Let $\varphi_e \subseteq: \{0,1\}^{**} \rightrightarrows \{0,1\}^{**}$ be the partial multifunction computed by the RAM program $e \in \{0,1\}^*$, *i.e.* the association of arrays $\mathtt{I} \in \{0,1\}^{**}$ and $\mathtt{O} \in \{0,1\}^{**}$ such that e admits a run on the input $\mathtt{I}$ that halts in a valid state and outputs $\mathtt{O}$.

For $t \in \mathbb{N}$, we denote by denote $\varphi_e(\mathtt{I})\!\downarrow_{[t]}$ the set of all output arrays $\mathtt{O}$ for which there exists a run of the program e on $\mathtt{I}$ of at most t steps halting in a valid state and outputting $\mathtt{O}$. We say that $f \subseteq: \{0,1\}^{**} \rightrightarrows \{0,1\}^{**}$ is *computable* (in time $t(s)$) if there exists a code $e \in \{0,1\}^*$ such that $f = \varphi_e$ (resp. for all $\mathtt{I} \in \{0,1\}^{**}$ of *bit length* $s = \sum_{i=1}^{\mathrm{len}(\mathtt{I})} |\mathtt{I}[i]|$, $f(\mathtt{I}) = \varphi_e(\mathtt{I})\!\downarrow_{[t(s)]}$).

[4]Our RAM model a is *word-RAM* model, in the sense that it operates on finite binary words. Nevertheless, using straightforward encoding, we will often consider these words as (binary expansions of) integers in $\mathbb{Z}$.

Remark 3.2 (Data types). In the same way we identified binary words and integers through their binary expansions, computations can actually be performed on arbitrary countable sets S through the choice of an *encoding* $\eta\colon S \to \{0,1\}^*$. Since most reasonable encodings of classical data types are computable from one another, this article will implicitly consider that RAM algorithms can manipulate integers of $\mathbb{N}$ and $\mathbb{Z}$, booleans $\{\top, \bot\}$, tuples $(\{0,1\}^*)^m$ (for arbitrary $m \in \mathbb{N}$), etc...

3.1.1. Word-RAM. RAM machines perform arithmetic operations on integers of arbitrary sizes in unit time. While this simplifies time complexities, it also has the unintended consequence of allowing to solve PSPACE-complete problems in polynomial time [HS74]. We will thus bound the size of words in variables and memory cells to avoid this issue [Сли78; AV79].

Definition 3.3 (word-RAM)**.** A RAM program $e \in \{0,1\}^*$ has *word length* $w(s)$ if, for any input $\mathtt{I} \in \{0,1\}^{**}$ of bit length $s = \sum_{i=0}^{\mathrm{len}(\mathtt{I})} |\mathtt{I}[i]|$ and for any execution on $\mathtt{I}$ halting in an accepting state, the word length of all memory cells and variables[5] is bounded at any time by $w(s)$.
In particular, if $w(s) = \mathcal{O}(\log s)$, e is said to be a log-*RAM program.*

As mentioned in the introduction, the log-RAM model is actually very close to real-world *machine code instructions* (which operate in unit time on integers of bounded lengths). By classical compilation processes (simulating a call stack, local variables and scopes, etc...), any C-like[6,7] program of polynomial time complexity $t(s)$ and word size $w(s)$ can be compiled into a RAM program with time $\mathcal{O}(t(s))$ and word size $w(s) + \mathcal{O}(\log t(s))$: in particular, if $t(s) = s^{\mathcal{O}(1)}$ and $w(s) = \mathcal{O}(\log s)$, then e is a log-RAM program. *We thus invite readers to consider the requirements in Lemma 5.2 and Section 7 about time complexity and variables of logarithmic length as they would in their favorite programming language.*

3.1.2. Classical results from computability. By considering RAM programs $e \in \{0,1\}^*$ as text/code, it is actually possible for RAM programs to read the code of other RAM programs, or even modify such codes. In what follows, we consider some classical computability results of this flavor and mention how they interact with time and word length restrictions. To do so, for any RAM program $e \in \{0,1\}^*$ and input array $\mathtt{I} \in \{0,1\}^{**}$, we denote by

$$T_e(\mathtt{I}) : \text{the maximal length of the halting and accepting runs of } e \text{ on } \mathtt{I};$$
$$W_e(\mathtt{I}) : \text{the maximal word length in of the halting and accepting runs of } e \text{ on } \mathtt{I}.$$

We first claim that there exists a (word-RAM preserving) universal interpreter:

Proposition 3.4. *There exists a RAM program $e_{\mathcal{U}}$ such that, for every RAM program $e \in \{0,1\}^*$ and every input $\mathtt{I} \in \{0,1\}^{**}$ of bit length n, we have*

$$\varphi_{e_{\mathcal{U}}}(e \cdot \mathtt{I}) = \varphi_e(\mathtt{I}).$$

Furthermore, time complexity and word length respectively satisfy $T_{e_{\mathcal{U}}}(e \cdot \mathtt{I}) = \mathcal{O}(T_e(\mathtt{I}))$ and $W_{e_{\mathcal{U}}}(e \cdot \mathtt{I}) = W_e(\mathtt{I}) + \mathcal{O}(\log |e|)$.

The construction of such an interpreter is a textbook result from computability that is proved by simulating the variables of e in the memory of $e_{\mathcal{U}}$. More precisely, the new machine $e_{\mathcal{U}}$ will contain at least a variable to simulate the program counter `PC` of e and three variables to perform arithmetic operations. Each instruction from e is decoded in time $\mathcal{O}(1)$ from the input of $e_{\mathcal{U}}$ using the simulated `PC`, and is then simulated in time $\mathcal{O}(1)$ (because of the additional `LOAD` and `STORE` instructions required to load the associated variables of e from the memory of $e_{\mathcal{U}}$).

Another classical result is the S_n^m theorem [Kle38], which allows to computably "hardcode" the value of some input words `I[0]`, ... directly in the of a program:

[5] In particular, we assume that $w(s) \geq \log |e|$ to include the program counter. However, since the `READ` instruction does not load a whole input word `I[`i`]` but operates one input bit at a time, this $w(s)$ bound does not apply to input words `I[`i`]`.

[6] Replace C by your favorite Turing-complete programming language: Common Lisp, Haskell, OCaml, Python...

[7] Since non-determinism is not a canonical feature of the C standard, we assume that some `NDET`-like instruction has been added. Similarly, we ignore any 64-bit lengths limitations.

Proposition 3.5 (S^m theorem)**.** *For all $m \in \mathbb{N}$, there exists a computable total function $S^m\colon \{0,1\}^* \times (\{0,1\}^*)^m \to \{0,1\}^*$ such that, for every code $e \in \{0,1\}^*$, every input $\mathtt{I} \in \{0,1\}^{**}$ and every words $(w_1, \dots, w_m) \in (\{0,1\}^*)^m$, we have*

$$\varphi_{S^m(e,w_1,\dots,w_m)}(\mathtt{I}) = \varphi_e(w_1 \cdot \ldots \cdot w_m \cdot \mathtt{I}).$$

Furthermore, time complexity and word length satisfy $T_{S^m(e,w_1\dots,w_m)}(\mathtt{I}) = \mathcal{O}(T_e(w_1 \cdot \ldots \cdot w_m \cdot \mathtt{I}))$ and $W_{S^m(e,w_1\dots,w_m)}(\mathtt{I}) = W_e(w_1 \cdot \ldots \cdot w_m \cdot \mathtt{I}) + \mathcal{O}(\log(|e| + \sum_{i=1}^m |w_i|))$.

Indeed, we can computably replace all `READ` instructions $\mathtt{var}_i \leftarrow \mathtt{I[var}_j\mathtt{][var}_k\mathtt{]}$ in the input program $e \in \{0,1\}^*$ by a `GOTO` to the end of the program e, where we add instructions to either read $\mathtt{I[var}_j - m\mathtt{]}$ or the hardcoded value of the corresponding $w_{\mathtt{var}_j}$. This can be implemented in time $\mathcal{O}(1)$ using arithmetic directly on the program counter. Since these modifications introduce new operations inside the code e, all the original `GOTO` instructions should be updated accordingly. The word length increases in order to index all these new instructions.

We now consider Kleene's fixed point theorems. Somewhat hidden in [Kle38], with modern phrasing from [Rog67, Chapter 11], the following theorem proves that all total computable function admit programs whose behavior they leave unchanged:

Proposition 3.6 (Fixed point theorem)**.** *For every computable total function $F\colon \{0,1\}^* \to \{0,1\}^*$, there exists a code $e \in \{0,1\}^*$ such that $\varphi_e = \varphi_{F(e)}$. Furthermore, time complexity and word length respectively satisfy*[8] *$T_e(\mathtt{I}) = \mathcal{O}(T_{F(e)}(\mathtt{I})) + \mathcal{O}(1)$ and $W_e(\mathtt{I}) = W_{F(e)}(\mathtt{I}) + \mathcal{O}(1)$.*

Proof. We follow the classical proof and analyze its time and word length. Let us fix $b \in \{0,1\}^*$ the following program: on a given input array $\mathtt{I} \in \{0,1\}^{**}$:

(1) Denoting $e' = \mathtt{I[0]}$ and $\mathtt{I}' = \mathtt{I[1:]}$, compute $\varphi_{e'}(e') \in \{0,1\}^{**}$; if this computation halts, denote $\mathtt{O} \in \{0,1\}^{**}$ the output result;
(2) If $\mathtt{O}$ is actually of length $\mathrm{len}(\mathtt{O}) = 1$ (*i.e.* $\mathtt{O} \in \{0,1\}^*$), compute $F(\mathtt{O})$ (otherwise, reject);
(3) Compute and return $\varphi_{F(\mathtt{O})}(\mathtt{I}') \in \{0,1\}^{**}$.

Applying the S^m theorem, there exists some total and computable $s\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ satisfying $\varphi_{s(b,e')}(\mathtt{I}) = \varphi_b(e' \cdot \mathtt{I})$ for all $e' \in \{0,1\}^*$ and $\mathtt{I} \in \{0,1\}^{**}$. Since s is computable, there exists a code $a \in \{0,1\}^*$ such that $\varphi_a(e') = s(b, e')$; and since s is total, so is $\varphi_a\colon \{0,1\}^* \to \{0,1\}^*$. Thus, the code $e = \varphi_a(a) \in \{0,1\}^*$ is well-defined; and for any input $\mathtt{I} \in \{0,1\}^{**}$, we have:

$$\varphi_e(\mathtt{I}) = \varphi_{s(b,a)}(\mathtt{I}) = \varphi_b(a \cdot \mathtt{I}) = \varphi_{F(\varphi_a(a))}(\mathtt{I}) = \varphi_{F(e)}(\mathtt{I}),$$

since $\varphi_a(a)$ halts and F is total.

We now consider the time and word length of e. To do so, we fix some code $\langle F \rangle$ for F:

- By the S^m theorem, the time complexity $T_e(\mathtt{I}) = T_{s(b,a)}(\mathtt{I})$ is $\mathcal{O}(T_b(a \cdot \mathtt{I}))$. By definition, $T_b(a \cdot \mathtt{I}) = T_{e_\mathcal{U}}(a \cdot a) + T_{\langle F \rangle}(e) + T_{e_\mathcal{U}}(F(e) \cdot \mathtt{I})$. Since $a \in \{0,1\}^*$ and $e \in \{0,1\}^*$ only depend on our choice of $\langle F \rangle$, $T_{e_\mathcal{U}}(a \cdot a)$ and $T_{\langle F \rangle}(e)$ are both constants for varying $\mathtt{I}$, and by our choice of universal machine $T_{e_\mathcal{U}}(F(e) \cdot \mathtt{I}) = \mathcal{O}(T_{F(e)}(\mathtt{I}))$. Thus:
$$T_e(\mathtt{I}) = \mathcal{O}(T_{F(e)}(\mathtt{I})) + \mathcal{O}(1).$$
- With the same reasoning, we obtain that $W_e(\mathtt{I}) = W_{s(b,a)}(\mathtt{I}) = W_b(a \cdot \mathtt{I}) + \mathcal{O}(\log |a|)$. Since $W_b(a \cdot \mathtt{I} = W_{e_\mathcal{U}}(a \cdot a) + W_{\langle F \rangle}(e) + W_{e_\mathcal{U}}(F(e) \cdot \mathtt{I})$, and that $W_{e_\mathcal{U}}(a \cdot a)$, $W_{\langle F \rangle}(e)$, $\log |a|$ and $\log |F(e)|$ are constants for varying $\mathtt{I}$, we deduce that:
$$W_e(\mathtt{I}) = W_{F(e)}(\mathtt{I}) + \mathcal{O}(1).$$ □

In conjunction with the S^m theorem, it is classically used to obtain programs that can access their own code (second fixed point theorem): for every computable function $\subseteq\colon \{0,1\}^{**} \rightrightarrows \{0,1\}^{**}$, there exists a code $e \in \{0,1\}^*$ such that, for every $\mathtt{I} \in \{0,1\}^{**}$, we have:

$$\varphi_e(\mathtt{I}) = f(e \cdot \mathtt{I}).$$

Nevertheless, the fixed point tiling construction in this article will be based upon the fixed point theorem from Proposition 3.6.

[8] All the following additive $\mathcal{O}(1)$ terms actually depend on the computable function F

§ 3.2. Grid-connected processor arrays

When embedding universal computations within tilings of domain $[\![n]\!]^2$, one usually draws the space-time diagram of a Turing machine: spatially, the tape occupies n cells on each line; and temporally, n lines allow for n steps of computations. However, we have rarely seen any consideration being given to higher-dimensional embeddings. In this direction, we consider the model of *processor arrays*: a highly parallel model of computation that appears especially suitable for tilings of d-dimensional rectangles $[\![n_1, \dots, n_d]\!]$.

3.2.1. Processor arrays. Processor arrays are a distributed model of computation in which the nodes of a graph, sometimes called *processors*, can perform some elementary operations and most importantly communicate with their neighbors to globally compute a function [Akl85, Chapter 5].

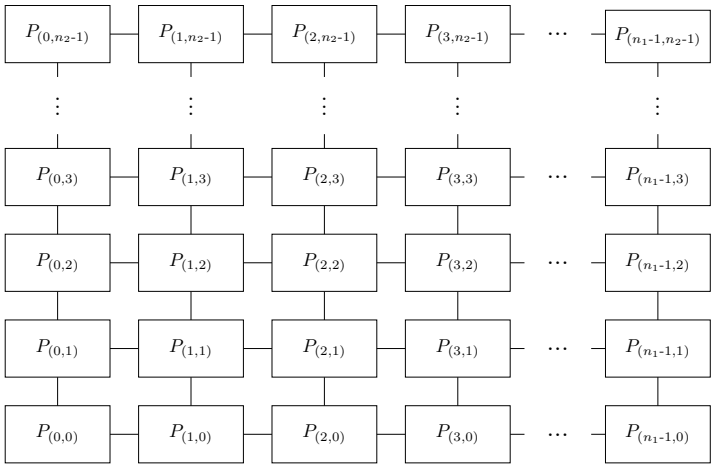


FIGURE 4. A processor array on the rectangle $[\![n_1, n_2]\!]$.

We call *processor* a register machine $((\mathtt{var}_i)_{i\in I}, \mathtt{PC})$ holding finitely many variables $\mathtt{var}_i \in \{0,1\}^*$ and a *program counter* $\mathtt{PC}$. In addition to performing (non-deterministic) arithmetic operations on its variables and control flow operations on its program counter, we assume that processors can be *connected* to their (at most) $2d$ neighbors and that an instruction $\mathtt{COMM} : \mathtt{var}_i \leftarrow P_{\mathtt{var}_j}(\mathtt{var}_k)$ allows a processor to read the variable of index $\mathtt{var}_k$ in its neighbor of index $\mathtt{var}_j$.

A *processor array* is then a pair $(\mathfrak{r}, e)$ consisting of a rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$ and a global program $e \in \{0,1\}^*$, which specifies the finite set of variables $(\mathtt{var}_i)_{i\in I}$ and a sequence of instructions. In order to perform computations, we place processors running the program e at each node in the graph of vertices $\mathfrak{r}$, and connect facet-adjacent processors with wires. *The word length of the variables of a processor is bounded by* $\mathcal{O}(\log(\sum_{k=1}^{d} n_k))$.

For a program $e \in \{0,1\}^*$ defining variables V, and for a variable $\mathtt{var} \in V$, we can define the projection $\pi_{\mathtt{var}} : (\{0,1\}^*)^V \times \mathbb{N} \to \{0,1\}^*$ projecting the *state* of a processor (*i.e.* the value of its variables and of its program counter $\mathtt{PC}$) to the value of its variable $\mathtt{var}$. This projection extends to the states of whole processor arrays $\pi_{\mathtt{var}} : ((\{0,1\}^*)^V \times \mathbb{N})^{\mathfrak{r}} \to (\{0,1\}^*)^{\mathfrak{r}}$.

In order to compute with a processor array $(\mathfrak{r}, e)$, we will assume that the set of variables V defined by e includes at least three variables $\mathtt{pos} = (\mathfrak{r}, \boldsymbol{i})$ (the position of the processor $\boldsymbol{i} \in \mathfrak{r}$), $\mathtt{input} \in \{0,1\}^*$ and $\mathtt{output} \in \{0,1\}^*$. A computation step of the global array then consists in executing a single instruction on each of its processors synchronously. By considering the mapping between the initial $\mathtt{input}$ and final $\mathtt{output}$ variables (*i.e.* resp. $\pi_{\mathtt{input}}$ and $\pi_{\mathtt{output}}$) after all processors have halted in an accepting state, a processor array defines a non-deterministic (partial) function $(\{0,1\}^*)^{\mathfrak{r}} \rightrightarrows (\{0,1\}^*)^{\mathfrak{r}}$.

3.2.2. The sorting problem. In the *sorting problem*, each processor in a processor array of domain $\mathfrak{r}$ is given an integer in its $\mathtt{input}$ variable: the goal is to sort the resulting array of $\mathbb{N}^{\mathfrak{r}}$ in boustrophedon order.

Proposition 3.7 ([TK77])**.** *There exists a deterministic program $e \in \{0,1\}^*$ such that, for any rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$, the processor array $(\mathfrak{r}, e)$ solves the sorting problem in time $\mathcal{O}(\sum_{k=1}^{d} n_k)$.*

3.2.3. Simulation of log*-RAM programs.* Proposition 3.7 provides a deterministic sorting algorithm that, combined with the non-deterministic capabilities of the processors, allows to efficiently simulate log-RAM programs.

For an input array $\mathtt{I} \in \{0,1\}^{**}$ of bit length s, we define $\mathrm{flat}_{\mathtt{I}} \colon \mathrm{dom}(\mathtt{I}) \to \{0, \dots, s-1\}$ the indexing function that is increasing for the lexicographic order, *i.e.* $\mathrm{flat}_{\mathtt{I}}(i,j) = \sum_{i'<i} \mathrm{len}(\mathtt{I}[i']) + j$. For a rectangle $\mathfrak{r} \in \mathfrak{R}_0$, we define the folding function $\mathrm{fold}_{\mathfrak{r}} \colon \{0,1\}^{**} \to (\mathbb{N}^2 \times \{0,1\})^{\mathfrak{r}}$ which maps an input array $\mathtt{I} \in \{0,1\}^{**}$ to the boustrophedon folding of its flattening (*i.e.* the pattern $w \in (\mathbb{N}^2 \times \{0,1\})^{\mathfrak{r}}$ such that, if $\boldsymbol{i} \in \mathfrak{r}$ is the position of index $\mathrm{flat}_{\mathtt{I}}(i,j)$ in the boustrophedon ordering of $\mathfrak{r}$, then $w_{\boldsymbol{i}} = (i, j, \mathtt{I}[i][j])$).

Recall that the *volume* of a rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$ is, by definition, $\|\mathfrak{r}\| = \prod_{k=1}^{d} n_k$. In what follows, we prove that there exists a program $e_{\mathcal{U}}$ such that processor arrays $(\mathfrak{r}, e_{\mathcal{U}})$ can simulate $\prod_{k=1}^{d} n_k$ steps of log-RAM computations in time $\mathcal{O}(\sum_{k=1}^{d} n_k)$:

Lemma 3.8 (Accelerated RAM simulations)**.** *There exists a program $e_{\mathcal{U}}$ such that, for any RAM program $e \in \{0,1\}^*$ of logarithmic word length $w(s) = \mathcal{O}(\log s)$, for every input array $\mathtt{I} \in \{0,1\}^{**}$, for every rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$ and for every $t \in \mathbb{N}$ such that $t \leq \|\mathfrak{r}\| = \prod_{k=1}^{d} n_k$, if the processor array $(\mathfrak{r}, e_{\mathcal{U}})$ is initialized as follows:*

- *The* `program` *variable in each processor contains the code $e \in \{0,1\}^*$;*
- *The* `word` *variable in each processor contains an upper bound on $w(\|\mathfrak{r}\|)$;*
- *The* `timeout` *variable in each processor contains the time bound $t \in \mathbb{N}$;*
- *The* `input` *variables of the processors (i.e. the projection $\pi_{\mathtt{input}}$) form[9] the pattern $\mathrm{fold}_{\mathfrak{r}}(\mathtt{I})$;*

then the outputs $w \in (\{0,1\}^*)^{\mathfrak{r}}$ yielded by accepting computations are exactly[9] the patterns $\mathrm{fold}_{\mathfrak{r}}(\mathtt{O})$ for $\mathtt{O} \in \varphi_e(\mathtt{I}){\downarrow_{[t]}}$, i.e. for $\mathtt{O}$ computed by e in time t.
Furthermore, the processor array $(\mathfrak{r}, e_{\mathcal{U}})$ halts in time $\mathcal{O}(\sum_{k=1}^{d} n_k)$; and the word length in each processor is bounded by $w(s) + \mathcal{O}(\log\|\mathfrak{r}\|) = \mathcal{O}(\log\|\mathfrak{r}\|)$ during accepting computations.

Proof. Let $e \in \{0,1\}^*$ be a RAM program, and assume that we are given a *trace* of the program e as a list of instructions performed in chronological order (*i.e.* for each operation, we record the addresses and values that were read and written). By sorting the trace in lexicographic order (first on memory addresses, and then on time), we obtain a new list in which all records operating on the same memory cell are adjacent: in other words, the validity of the global computation only depends on the consistency of such adjacent records.

Thus, if $e \in \{0,1\}^*$ is a log-RAM program, the word length is large enough for processors to each non-deterministically "guess" a computation step in $\mathcal{O}(1)$ steps. After checking in time $\mathcal{O}(1)$ that adjacent processors contain plausibly consecutive instructions (by considering the associated *program counters* `PC`), and sorting these records (by memory addresses and time) using Proposition 3.7, adjacent processors check the consistency of successive operations applied to the same memory cell. □

> *Details.* Let us consider more formally the *trace* of a RAM computation. A computation step of a RAM programs can be summarized as:
>
> (1) Reading a few variables/memory cells: for example, the instruction $\mathtt{ADD} : \mathtt{var}_1 \leftarrow \mathtt{var}_2 + \mathtt{var}_3$ reads the values of variables $\mathtt{var}_2$ and $\mathtt{var}_3$; the instruction $\mathtt{LOAD} : \mathtt{var}_1 \leftarrow \mathtt{M}[\mathtt{var}_2]$ reads the value of the variable $\mathtt{var}_2$ and the memory cell $\mathtt{M}[\mathtt{var}_2]$; etc...
> (2) (Optional) Performing a unit operation on the words read in the previous step (*e.g.* an addition, bit shift...);
> (3) Writing a few variables/memory cells: the instruction $\mathtt{ADD} : \mathtt{var}_1 \leftarrow \mathtt{var}_2 + \mathtt{var}_3$ changes the value of the variable $\mathtt{var}_1$; the instruction $\mathtt{STORE} : \mathtt{M}[\mathtt{var}_1] \leftarrow \mathtt{var}_2$ changes the value of the memory cell $\mathtt{M}[\mathtt{var}_1]$;
> (4) Updating the program counter `PC`: either increasing it by 1 for most instructions, or setting it to an arbitrary value by a `GOTO` instruction.

[9] Actually, $\mathrm{fold}_{\mathfrak{r}}$ returns a rectangular pattern over the alphabet $\mathbb{N}^2 \times \{0,1\}^*$ instead of $\{0,1\}^*$; thus, this equality holds only up to decoding binary strings of $\{0,1\}^*$ into tuples of $\mathbb{N}^2 \times \{0,1\}^*$.

We now detail the design of a program $e_{\mathcal{U}} \in \{0,1\}^*$ for processor arrays. Among its variables, it defines a `program` variable that is assumed to contain a whole RAM program $e \in \{0,1\}^*$, a `word` variable holding an upper bound of the allowed word length, a `timeout` variable and a `simPC` variable indexing an instruction of e. The program operates in two phases:

Part 1: Simulation. Reading its $\texttt{pos} = (\mathfrak{r}, \mathfrak{i})$ variable and checking with their neighbors that all processors share a common value $\texttt{timeout} \in \mathbb{N}$, each processor computes in time $\mathcal{O}(1)$ its index $n \in \{0, \dots, \|r\| - 1\}$ in the boustrophedon ordering of $\mathfrak{r}$. If $n \leq \texttt{timeout}$, it then non-deterministically "guesses" the n^{th} step of the RAM computation of e:

(1) First, if $n = 0$, then the processor sets its `simPC` variable to 0; otherwise, it uses non-determinism to fill a value `simPC` that indexes an instruction in e;

(2) Then, consider the instruction `INSTR` in e indexed by `simPC`:

- `INSTR` requires at most two variable/memory readings: using non-determinism, fill two words $u, v \in \{0,1\}^*$ of word length `word` to act as "guesses" for these readings; and in two variables `load1`, `load2`, store these guesses as records of the form

$$(\texttt{var}, \texttt{READ}, \texttt{time} = n, \texttt{address} = \texttt{var}_i, \texttt{value} = u)$$
$$(\texttt{M}, \texttt{READ}, \texttt{time} = n, \texttt{address} = u, \texttt{value} = v)$$

  if `INSTR` is an instruction $\texttt{LOAD} : \dots \leftarrow \texttt{M[var}_i\texttt{]}$; or of the form

$$(\texttt{var}, \texttt{READ}, \texttt{time} = n, \texttt{address} = \texttt{var}_i, \texttt{value} = u)$$
$$(\texttt{var}, \texttt{READ}, \texttt{time} = n, \texttt{address} = \texttt{var}_j, \texttt{value} = v)$$

  if `INSTR` is an arithmetic operation such as $\texttt{ADD} : \dots \leftarrow \texttt{var}_i + \texttt{var}_j$ or a memory writing operation $\texttt{STORE} : \texttt{M[var}_i\texttt{]} \leftarrow \texttt{var}_j$; etc...

  In the case of an input reading $\texttt{READ} : \dots \leftarrow \texttt{I[var}_i\texttt{][var}_j\texttt{]}$, we similarly non-deterministically guess words u, v and store records in variables `load1` and `load2`

$$(\texttt{var}, \texttt{READ}, \texttt{time} = n, \texttt{address} = \texttt{var}_i, \texttt{value} = u)$$
$$(\texttt{var}, \texttt{READ}, \texttt{time} = n, \texttt{address} = \texttt{var}_j, \texttt{value} = v),$$

  but we also guess an additional bit $b \in \{0,1\}$ stored as a record in a specific `read` variable

$$(\texttt{I}, \texttt{address} = (u, v), \texttt{value} = b).$$

- (Optional) If `INSTR` is an arithmetic operation (or any other requiring a computation), simulate this operation using `load1` and `load2` and store the result in a variable `res`;
- `INSTR` requires at most one memory writing: using non-determinism, we fill a variable `store` with either

$$(\texttt{var}, \texttt{WRITE}, \texttt{time} = n, \texttt{address} = \texttt{var}_i, \texttt{value} = u)$$

  if `INSTR` is an arithmetic operation such as $\texttt{ADD} : \texttt{var}_i \leftarrow \dots \texttt{op} \dots$, and where $u \in \{0,1\}^*$ was stored in `res` at the previous step; or

$$(\texttt{M}, \texttt{WRITE}, \texttt{time} = n, \texttt{address} = u, \texttt{value} = v)$$

  if `INSTR` was some $\texttt{STORE} : \texttt{M[var}_i\texttt{]} \leftarrow \dots$, and where $u, v \in \{0,1\}^*$ were respectively stored in a `load` variable as a guess of $\texttt{var}_i$, and v was stored in `res` at the previous step; etc...

  In the case of an output writing $\texttt{WRITE} : \texttt{O[var}_i\texttt{][var}_j\texttt{]} \leftarrow \dots$, we store in a `write` variable the record

$$(\texttt{O}, \texttt{address} = (u, v), \texttt{value} = b)$$

  where $u, v \in \{0,1\}^*$ were stored in some `load` variables as guesses for $\texttt{var}_i$ and $\texttt{var}_j$, and $b \in \{0,1\}$ was stored in `res` at the previous step.

- To conclude the simulation of `INSTR`, the processor stores in a variable $\texttt{simPC}'$ the new value of the instruction pointer, as either $\texttt{simPC} + 1$ or as written in `res`.

(3) Finally, check the *chronological continuity* of this simulated computation step: reading the `simPC` variable of its successor (in boustrophedon order) inside the processor array of domain $\mathfrak{r}$, the processor checks that it equals its own variable $\texttt{simPC}'$.

If we think about this simulation informally, the processors have simulated a plausible sequence of RAM instructions for the program e of length smaller than `timeout`; but memory readings/writings may have been guessed incorrectly. We solve this issue in the second "validation" phase:

Part 2: Validation.

(1) Verifying the non-deterministic input readings:
- Using the sorting procedure described in Proposition 3.7 [TK77] on the variables `input` and `read` mixed together, we sort records $\texttt{read} = (\texttt{I}, \texttt{address} = (u, v), \texttt{value} = b)$ and $\texttt{input} = (i, j, b)$ by addresses along the boustrophedon indexing of $\mathfrak{r}$.
- In the resulting array, all processors containing an input record for the address (i, j) are consecutive in the boustrophedon ordering of $\mathfrak{r}$, and thus adjacent; in parallel time $\mathcal{O}(1)$, they can check that all $\texttt{value} = b$ share the same value $b \in \{0, 1\}$, and that the latter is consistent with the actual input (i, j, b);

(2) Verifying the chronological consistency of variable/memory interactions:
- Using the sorting procedure described in Proposition 3.7 [TK77], we sort the `load` and `write` variables mixed together in lexicographic order (first by type `var` or `M`, then by `address`, then by `time`, and finally `READ` before `WRITE`) along the boustrophedon indexing of $\mathfrak{r}$.
- In the resulting array, all records about the same address $x \in \{0, 1\}^*$ are consecutive in the boustrophedon ordering of $\mathfrak{r}$, and these form a segment of memory interactions sorted in chronological order. In parallel time $\mathcal{O}(1)$, processors can communicate with their neighbor (in boustrophedon order) that these cords are chronologically consistent (successive `LOAD` records, or two consecutive `STORE` and `LOAD` records, should contain the same `value`, etc...).

Thus, at this point, the RAM computation simulated by the processor array is both chronologically consistent and correctly initialized.

(3) Correctly displaying the output:
- Using the sorting procedure described in Proposition 3.7 [TK77], we sort the `write` variables by addresses along the boustrophedon indexing of $\mathfrak{r}$.
- In parallel time $\mathcal{O}(1)$, processors communicate with their successor (in boustrophedon ordering) to check that two consecutive records for `O[`i`][`j`]` and `O[`i'`][`j'`]` are successors for the lexicographic order $(i, j) <_{\text{lex}} (i', j')$.
- Finally, each processor having a non-empty `write` record $(\texttt{O}, \texttt{address} = (i, j), \texttt{value} = b)$ copy its content to its variable $\texttt{output} = (i, j, b)$.

If any of these local checks has failed, the processor array rejects the computation. Otherwise, the computation halts in an accepting state. □

4. Rectangles and wirings in Wang tilings

The fixed point construction in Section 5 will be based on computational embeddings inside the rectangles of d-dimensional grids $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{Z}^d}$. This section will thus focus on drawing finite rectangles, rectangular grids, and transmitting information inside such rectangles.

§ 4.1. Drawing rectangles with Wang tiles

In this section, we define an infinite tileset $T_{\mathfrak{R}}$ to draw finite tilings of rectangles $\mathfrak{r} \in \mathfrak{R}_0$. To do so, consider $\mathcal{C}_{\mathfrak{R}} \subseteq (\mathbb{N}^d \times \mathbb{N}^d) \cup \{\texttt{B}\}$ the set of *positioning colors* defined as follows:

$$\mathcal{C}_{\mathfrak{R}} = \{((n_1, \dots, n_d), \boldsymbol{i}) : \boldsymbol{i} \in [\![n_1, \dots, n_d]\!]\} \cup \{\texttt{B}\}.$$

Abusing notations between $(n_1, \dots, n_d)$ and the rectangle $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$, we will always refer to a positioning color as a pair $(\mathfrak{r}, \boldsymbol{i})$ for $\boldsymbol{i} \in \mathfrak{r}$.

We now define $T_{\mathfrak{R}}$ as the set of tiles $t \in \mathcal{C}_{\mathfrak{R}}^{2d+1}$ such that there exists $\mathfrak{r} \in \mathfrak{R}_0$, $\boldsymbol{i} \in \mathfrak{r}$ and

- $f_{\bullet}(t) = (\mathfrak{r}, \boldsymbol{i})$;
- If $\boldsymbol{i} \notin f_k^-(\mathfrak{r})$, then $f_k^-(t) = (\mathfrak{r}, \boldsymbol{i})$; otherwise, $f_k^-(t) = \texttt{B}$ is left blank;
- If $\boldsymbol{i} \notin f_k^+(\mathfrak{r})$, then $f_k^+(t) = (\mathfrak{r}, \boldsymbol{i} + \boldsymbol{e}^k)$; otherwise, $f_k^+(t) = \texttt{B}$ is left blank;

We claim that the decorations of valid patterns on $T_{\mathfrak{R}}$ exactly draw the rectangles of $\mathfrak{R}_0$:

Proposition 4.1. *For any rectangle $\mathfrak{r} \in \mathfrak{R}_0$, there exists a valid pattern $w \in (T_{\mathfrak{R}})^{\mathfrak{r}}$ such that, for every position $\boldsymbol{i} \in \mathfrak{r}$, we have $f_{\bullet}(w_{\boldsymbol{i}}) = (\mathfrak{r}, \boldsymbol{i})$. Conversely: for any rectangle $\mathfrak{r} \in \mathfrak{R}_0$, for any valid pattern $w \in (T_{\mathfrak{R}})^{\mathfrak{r}}$ such that $f_{\bullet}(w_{\boldsymbol{0}}) = (\mathfrak{r}, \boldsymbol{0})$ and for any position $\boldsymbol{i} \in \mathfrak{r}$, we have $f_{\bullet}(w_{\boldsymbol{i}}) = (\mathfrak{r}, \boldsymbol{i})$.*

In fact, after adding a blank tile $\texttt{B}^{2d+1}$ to this tileset, the valid configurations over $T_{\mathfrak{R}} \cup \{\texttt{B}^{2d+1}\}$ are exactly the drawings of independent rectangles on $\mathbb{Z}^d$ over a blank background.

§ 4.2. Drawing grids with Wang tiles

If one aims at drawing rectangular grids instead, we define a similar set of tiles $T_{\mathfrak{G}}$ as the set of tiles $t \in \mathcal{C}_{\mathfrak{R}}^{2d+1}$ such that: there exists $\mathfrak{r} \in \mathfrak{R}_0$ and $\boldsymbol{i} \in \mathfrak{r}$ for which

- $f_{\bullet}(t) = (\mathfrak{r}, \boldsymbol{i})$;
- $f_k^-(t) = (\mathfrak{r}, \boldsymbol{i})$;
- If $\boldsymbol{i} \notin f_k^+(\mathfrak{r})$, then $f_k^+(t) = (\mathfrak{r}, \boldsymbol{i} + \boldsymbol{e}^k)$; otherwise, $f_k^+(t) = (\mathfrak{r}', \boldsymbol{i} + \boldsymbol{e}^k \bmod \mathfrak{r})$ for some rectangle $\mathfrak{r}' \in \mathfrak{R}_0$ such that $[f_k^+(\mathfrak{r})]$ and $[f_k^-(\mathfrak{r}')]$ are equal.

We claim that valid configurations on $T_{\mathfrak{G}}$ draw non-regular rectangular grids:

Proposition 4.2. *For any grid $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{p}})_{\boldsymbol{p} \in \mathbb{Z}^d}$, there exists a unique valid tiling $x \in (T_{\mathfrak{G}})^{\mathbb{Z}^d}$ such that, for any position $\boldsymbol{i} \in \mathbb{Z}^d$ appearing in the rectangle $\mathfrak{r}_{\boldsymbol{p}}$ from the grid $\mathfrak{g}$, we have $f_{\bullet}(x_{\boldsymbol{i}}) = ([\mathfrak{r}_{\boldsymbol{p}}], \boldsymbol{i} \bmod \mathfrak{g})$.*
Conversely: for any valid tiling $x \in (T_{\mathfrak{G}})^{\mathbb{Z}^d}$, there exists a unique grid $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{p}})_{\boldsymbol{p} \in \mathbb{Z}^d}$ such that, for any position $\boldsymbol{i} \in \mathbb{Z}^d$ appearing in the rectangle $\mathfrak{r}_{\boldsymbol{p}}$ from the grid $\mathfrak{g}$, we have $f_{\bullet}(x_{\boldsymbol{i}}) = ([\mathfrak{r}_{\boldsymbol{p}}], \boldsymbol{i} \bmod \mathfrak{g})$.

§ 4.3. Drawing wires with Wang tiles

Finally, we will often need to route information inside rectangles using continuous wires. For any set of colors $\mathcal{C}$, we define the *wiring colors* $\mathcal{C}_{\rightleftharpoons} = (\mathcal{C} \times \mathbb{N})^* \cup (\mathcal{C} \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$; and for a fixed constant $K_{\text{cross}} \in \mathbb{N}$, we define the *wiring tileset* $T_{\rightleftharpoons}(\mathcal{C})$ as the set of tiles $t \in (\mathcal{C}_{\rightleftharpoons})^{2d+1}$ such that there exists an index I such that $|I| \leq K_{\text{cross}}$ and colors $(c_i)_{i \in I} \in \mathcal{C}^I$ such that:

(1) $f_{\bullet}(t)$ is a list of pairs $((c_i, x_i))_{i \in I}$ (for $x_i \in \{\texttt{start}, \texttt{end}, \texttt{cross}\}$) encoding the list of wires traversing the tile t; intuitively, x_i represents whether the wire starts in t, ends in t, or crosses t;
(2) The color along each facet $f_k^{\pm}(t)$ is a list of pairs in $\mathcal{C} \times \mathbb{N}$, representing respectively the color and the length of the wires;
(3) There is a bijection between the wires $(c_i, \texttt{cross})$ in $f_{\bullet}(t)$ (*i.e.* the wires crossing t) and the pairs (c_i, n_i) and $(c_i, n_i + 1)$ (for some $x_i \in \mathbb{N}$) appearing across the facets $f_k^{\pm}(t)$;
(4) There is a bijection between the $(c_i, \texttt{start})$ in $f_{\bullet}(t)$ (resp. $(c_i, \texttt{end})$ in $f_{\bullet}(t)$) and the wiring tuples $(c_i, 0)$ (resp. wiring colors (c_i, n_i) without a matching $(c_i, n_i + 1)$ for $n_i > 0$) appearing across the facets $f_k^{\pm}(t)$.

Proposition 4.3. *For any valid pattern $w \in T_{\rightleftharpoons}(\mathcal{C})^{\mathfrak{r}}$ with blank borders*[10]*, there exists a bijection between the tuples $(c, \texttt{start})$ and $(c, \texttt{end})$ (for $c \in \mathcal{C}$) appearing across the decorations $f_{\bullet}(w_{\boldsymbol{i}})$ for $\boldsymbol{i}$ ranging over $\mathfrak{r}$.*
Furthermore, for each $(c, \texttt{start})$ appearing in some decoration $f_{\bullet}(w_{\boldsymbol{s}})$ for $\boldsymbol{s} \in \mathfrak{r}$, there exists a path of contiguous tiles starting at position $\boldsymbol{s}$ and ending at some position $\boldsymbol{e}$ such that $f_{\bullet}(w_{\boldsymbol{e}})$ contains the wiring tuple $(c, \texttt{end})$; and if $\boldsymbol{i}$ is the k^{th} (intermediate) position in this path, then $(c, \texttt{cross})$ appears in $f_{\bullet}(w_{\boldsymbol{i}})$, and there exists a pair of facets (f, f') such that $(c, k) \in f(w_{\boldsymbol{i}})$ and $(c, k+1) \in f'(w_{\boldsymbol{i}})$.

We call *wire* any such path of contiguous tiles in w, and *wiring* the bijection between the start and the end of each wire drawn by w. We made the wiring tiles t encode their position in the wires crossing them because of convenience (it helps to identify them when looking at patterns) and practical purposes: it eliminates cycling wires.

We now consider the condition upon which a wiring between a given matching of starting and ending positions can exist inside a rectangle $\mathfrak{r}$. Inside a facet $f_k^{\pm}(\mathfrak{r})$ of a rectangle $\mathfrak{r} \in \mathfrak{R}_0$, we say that a given set of positions $P \subseteq f_k^{\pm}(\mathfrak{r})$ is *greedy* if, for every $\boldsymbol{i} \in f_k^{\pm}(\mathfrak{r})$ such that $\boldsymbol{i} \in P$, all positions $\boldsymbol{j} \in f_k^{\pm}(\mathfrak{r})$ of boustrophedon index $k' < k$ (for k the boustrophedon index of $\boldsymbol{i}$ in $f_k^{\pm}(\mathfrak{r})$) also satisfy $\boldsymbol{j} \in P$. In other words, P fills the facet $f_k^{\pm}(\mathfrak{r})$ in boustrophedon order greedily.

For any $\mathfrak{r} \in \mathfrak{R}_0$, denote by $\partial\mathfrak{r} = \biguplus_{k=1}^{d} \biguplus_{s \in \{+,-\}} f_k^s(\mathfrak{r})$ the *border*[11] of $\mathfrak{r}$; and recall that $f(\mathfrak{r})$ refers to the smallest facet of $\mathfrak{r}$.

[10] *i.e.* for every position $\boldsymbol{i} \in \mathfrak{r}$ such that $\boldsymbol{i} + \boldsymbol{e}^k \notin \mathfrak{r}$ (resp. $\boldsymbol{i} - \boldsymbol{e}^k \notin \mathfrak{r}$), $f_k^+(w_{\boldsymbol{i}})$ (resp. $f_k^-(w_{\boldsymbol{i}})$) is the empty list.

[11] In this case, it is useful to consider this border as a disjoint union of facets: so that, if a position $\boldsymbol{i}$ appears at the intersection of two facets $f_k^{\pm}(\mathfrak{r})$ and $f_{k'}^{\pm}(\mathfrak{r})$, then it appears twice in $\partial\mathfrak{r}$.

Lemma 4.4 (Routing lemma)**.** *Let $\mathcal{C}$ be a set of colors and let $\mathfrak{r} \in \mathfrak{R}_0$ be a rectangle. For all subsets $S \subseteq \partial\mathfrak{r}$ and $D \subseteq \partial\mathfrak{r}$ of size $|S| = |D| \leq \|f(\mathfrak{r})\|$, and for all* source *and* destination *patterns $(s, d) \in \mathcal{C}^S \times \mathcal{C}^D$ satisfying the following conditions:*

- *For every color $c \in \mathcal{C}$, we have $|s|_c = |d|_c$;*
- *For every facet $f_k^\pm(\mathfrak{r})$, if $f_k^\pm(\mathfrak{r})$ is not a smallest facet of $\mathfrak{r}$, then both $S \cap f_k^\pm(\mathfrak{r})$ and $D \cap f_k^\pm(\mathfrak{r})$ are greedy in $f_k^\pm(\mathfrak{r})$;*

There exists a valid pattern $w \in (T_{\rightleftharpoons})^{\mathfrak{r}}$ with blank borders such that:

- *For every $\boldsymbol{i} \in \mathfrak{r}$, a tuple $(c, \texttt{start})$ appears in $f_{\blacklozenge}(w_{\boldsymbol{i}})$ if and only if $\boldsymbol{i} \in S$ and $s_{\boldsymbol{i}} = c$;*
- *For every $\boldsymbol{i} \in \mathfrak{r}$, a tuple $(c, \texttt{end})$ appears in $f_{\blacklozenge}(w_{\boldsymbol{i}})$ if and only if $\boldsymbol{i} \in D$ and $d_{\boldsymbol{i}} = c$;*
- *If $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$, then wires in w have length $\mathcal{O}(\sum_{k=1}^{d} n_k)$.*

In this paper, we now fix the value of the constant K_{cross} (bounding the number of wires that can appear in any given tile of $T_{\rightleftharpoons}$) in order to allow the tiles of $T_{\rightleftharpoons}$ to implement the geometric construction given in the following proof.

Proof. Fix $\mathfrak{r} = [\![n_1, \dots, n_d]\!]$. It is actually enough to consider a pair $(S, D) \subseteq f_S \times f_D$ for any facet $f_S = f_k^\pm(\mathfrak{r})$ and a smallest facet $f_D = f_h^\pm(\mathfrak{r})$. Indeed, starts and ends of wires are symmetric, and the routing lemma can be obtained by managing each pair of facets independently.

Since f_D is a smallest facet of $\mathfrak{r}$, the proof of Proposition 3.7 in [TK77] shows that any permutation of f_D can be realized with wires of length $\mathcal{O}(\lambda(\mathfrak{r}))$. Thus, instead of wiring S with exactly D, it is enough to realize a wire of S with any fixed $D' \subseteq f_D$. Since the case $f_S = f_D$ is already solved, we assume that $f_D \neq f_S$.

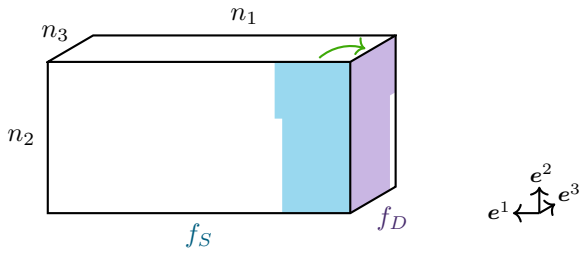
First, notice that the routing lemma is immediate in dimension $d = 2$. In the rest of this proof, we focus on the dimension $d = 3$, since the proof will generalize from dimension $d = 3$ to arbitrary $d > 3$ by keeping all other coordinates intact. By symmetry, we further assume that $f_D = f_1^-(\mathfrak{r})$ and that $f_S = f_3^-(\mathfrak{r})$, so that we can denote $f_S = [\![n_1, n_2, 1]\!]$, $f_D = [\![1, n_2, n_3]\!]$ and $n_1 = \max\{n_k\}$.

Since $S \subseteq f_S$ is greedy in f_S, let $\boldsymbol{e}^m$ refer to the most-significant direction of its boustrophedon indexing (denoting I_n the segment $I_n = \{(i_1, i_2, 0) \in f_S : i_m = n\}$, the points of S fill the segments I_n in increasing order of n). Since $|S| \leq |f(\mathfrak{r})| = |f_D| = n_2 \cdot n_3$, denoting

$$k_m = n_m \cdot \frac{n_3}{n_1}$$

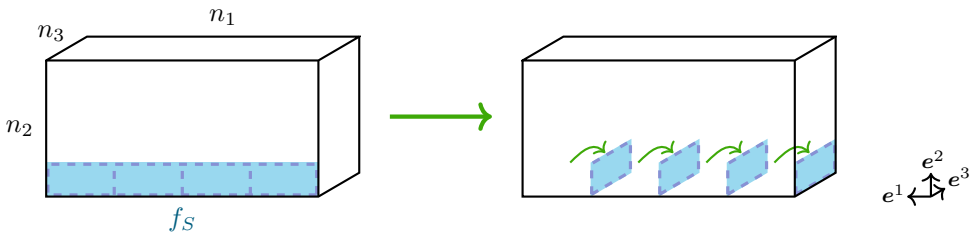
we notice that any segment I_n of index $n > k_m$ does not intersect S.

If $m = 1$, then $k_m = n_3$ and simple diagonal lines sending each position $x = (x_1, x_2, 0) \in S$ towards the position $(0, x_2, x_1) \in f_D$ draw a wiring between S and a subset $D' \subseteq f_D$:

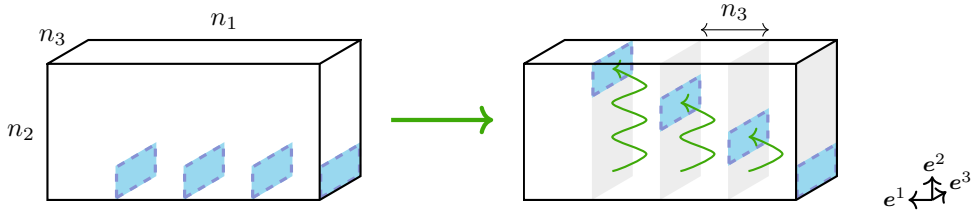


Otherwise, if $m = 2$, then $k_m = \frac{n_2 n_3}{n_1}$. We wire S and a subset $D' \subseteq f_D$ in three steps:

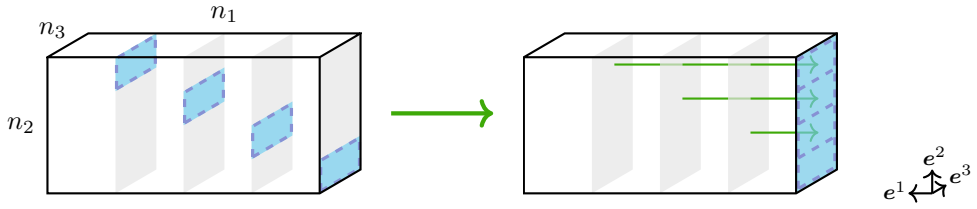
- We split the set S into blocks of length n_3 and turn them around using straight lines: each position $x = (kn_3 + i, i_2, 0)$ is sent to (kn_3, i_2, i) ($i < n_3$ and $k \leq \frac{n_1}{n_3}$);



- We then align these blocks by translating them using "zigzags" of amplitude k_m that send a position $x' = (kn_3, i_2, i)$ to $x'' = (kn_3, i_2 + k\frac{n_2 n_3}{n_1}, i))$.[12] As $n_2 \leq n_1$, we have that $k_m = \frac{n_2 n_3}{n_1} \leq n_3$, so that such zigzags can be drawn with at most $\mathcal{O}(1)$ superimposed wires at any position $\boldsymbol{i} \in \mathfrak{r}$;



- Finally, we send each of these positions in a straight line towards f_D: the position $x'' = (kn_3, i_2 + k\frac{n_2 n_3}{n_1}, i)$ is sent to $(0, i_2 + k\frac{n_2 n_3}{n_1}, i)$.



□

Remark 4.5. Notice that the wiring in this proof is actually constructive and "positional", in the sense that the direction of the wires appearing at a position $\boldsymbol{i} \in \mathfrak{r}$ only depends on said position. Thus, there exists a tileset $T_{\boldsymbol{\times}} \subseteq T_{\mathfrak{R}} \times T_{\overline{\rightarrow}}(\{0, \dots, 2d\} \times \mathcal{C})$ such that, if each decoration $(\boldsymbol{i}, (f, c, \texttt{start}))$ on the facet $f_k^{\pm}(\mathfrak{r})$ satisfies $f = d \pm k$, then the tileset $T_{\boldsymbol{\times}}$ satisfies the conclusion of the routing lemma while ensuring the *uniqueness* of the wiring pattern w.

5. The fixed point lemma

The fixed point construction was introduced in a series of articles [DRS08]–[DRS12]; themselves based on earlier work on self-simulating cellular automata [Gác86; Gác01]. For a more complete history of the construction, we refer to the extensive survey [Tör21].

This construction creates shifts of finite type that simulate an infinite hierarchy of configurations. It provides a flexible framework that can be adapted to a large variety of contexts (*e.g.* minimal shifts [DR21], projections and subdynamics of SFTs [DRS10], soficity of complex shifts [Wes17]...). In this section, we provide a technical abstraction of the construction under the formalism of non-deterministic substitutions on infinite tilesets.

Definition 5.1 (Computable local substitution)**.** For T_* the infinite tileset $T_* = (\{0,1\}^*)^{2d+1}$, consider a RAM program $\langle\tau\rangle \in \{0,1\}^*$ defining a function $\varphi_{\langle\tau\rangle} \colon \mathfrak{R}_0 \times \mathbb{Z}^d \times T_* \rightrightarrows T_*$. Then $\langle\tau\rangle$ defines a *computable local substitution* $\tau \colon T_* \rightrightarrows T_*^{\mathfrak{R}_0}$ as follows:

$$\forall a \in T_*,\ \tau(a) = \bigcup_{\mathfrak{r} \in \mathfrak{R}_0} \left\{ w \in T_*^{\mathfrak{r}} : \forall \boldsymbol{i} \in \mathfrak{r},\ w_{\boldsymbol{i}} \in \varphi_{\langle\tau\rangle}(\mathfrak{r}, \boldsymbol{i}, a) \right\}.$$

Furthermore, for $\mathfrak{r} \in \mathfrak{R}_0$ a rectangle, $a \in T_*$ a tile and $w \in T_*^{\mathfrak{r}}$ a pattern, we say that $a \xrightarrow{\tau} w$ is *computed in time* $t \in \mathbb{N}$ if $w_{\boldsymbol{i}} \in \varphi_{\langle\tau\rangle}(\mathfrak{r}, \boldsymbol{i}, a)\downarrow_{[t]}$ for all $\boldsymbol{i} \in \mathfrak{r}$. Similarly, for $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})$ a grid and $x, y \in T_*^{\mathbb{Z}^d}$ two configurations, the substitution $x \xrightarrow[\mathfrak{g}]{\tau} y$ is *computed in time* $t \in \mathbb{N}$ if each $x_{\boldsymbol{i}} \xrightarrow{\tau} y|_{\mathfrak{r}_{\boldsymbol{i}}}$ is computed in time t.

The notion of computable local substitution may appear quite arbitrary, but actually allows us to parallelize the computations of a substitution $w \in \tau(a)$ into independent pixels in the rectangular domain $\mathfrak{r} = \mathrm{dom}(w)$ of w. We discuss local substitutions in more details in Section 6.2.

[12] For rounding reasons, we might allow a small overlap between any two adjacent blocks.

We now introduce the main result of this article. Recall that for $\mathfrak{r} \in \mathfrak{R}_0$ a rectangle, we denote $\lambda(\mathfrak{r})$ and $\mu(\mathfrak{r})$ the length of its longest edges and the area of its smallest facets, respectively referred to as the *time* and *space* of $\mathfrak{r}$. We extend these quantities to infinite grid $\mathfrak{g}$ by defining $\lambda(\mathfrak{g}) = \sup_{\mathfrak{r}\in\mathfrak{g}} \lambda(\mathfrak{r})$ and $\mu(\mathfrak{g}) = \inf_{\mathfrak{r}\in\mathfrak{g}} \mu(\mathfrak{r})$. Finally, recall that T_* refers to the infinite set of d-dimensional Wang tiles with colors $\mathcal{C} = \{0,1\}^*$.

Lemma 5.2 (Fixed point lemma)**.** *Let $\langle\tau\rangle \in \{0,1\}^*$ be a* log*-RAM program defining a computable local growing substitution $\tau\colon T_* \rightrightarrows T_*^{\mathfrak{R}_0}$ on the infinite tileset T_*. For $\alpha < 1$ and $\delta < \frac{1}{2}$ two rationals, and $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ a sofic shift space on some finite alphabet $\mathcal{A} \subseteq \{0,1\}^*$, let $\overleftarrow{X}_{\langle\tau\rangle}$ denote the set of configurations $x \in X$ such that:*

(1) There exists a sequence of grids $(\mathfrak{g}_\ell)_{\ell\geq 1}$ such that, denoting $\tilde{\mu}_\ell = \prod_{i=1}^{\ell} \mu(\mathfrak{g}_\ell)$, we have:
  (i) $2 \leq \lambda(\mathfrak{g}_{\ell+1}) \leq 2^{\mathcal{O}(\tilde{\mu}_\ell^{\delta})}$ (upper bound on the size of the rectangles);
  (ii) $\mu(\mathfrak{g}_{\ell+1})^{\mathcal{O}(\log \tilde{\mu}_\ell)} \geq \lambda(\mathfrak{g}_{\ell+1})$ (upper bound on the eccentricity of the rectangles);

(2) There exists a sequence of valid Wang tilings $(x^{(\ell)})_{\ell\in\mathbb{N}}$ in $(T_*)^{\mathbb{Z}^d}$ such that:
  (i) $f_{\blacklozenge}(x^{(0)}) = x$, i.e. the decorations of $x^{(0)}$ project onto x;
  (ii) For every $\ell \geq 1$, the colors in $x^{(\ell)}$ have length at most $\|x^{(\ell)}\| = \mathcal{O}(\tilde{\mu}_{\ell-1}^{\alpha})$;
  (iii) For every $\ell \geq 1$, the substitution step $x^{(\ell)} \xrightarrow[\mathfrak{g}_\ell]{\tau} x^{(\ell-1)}$ is computed in time $\mathcal{O}(\tilde{\mu}_{\ell-1}^{\alpha})$.

Then $\overleftarrow{X}_{\langle\tau\rangle}$ is a sofic shift space.

Comment. Lemma 5.2 calls for some technical comments on the optimality of the parameters α, δ.

The bound $\delta < \frac{1}{2}$ on the sizes of the substitutions appears to be rather arbitrary, and is indeed a residue of the bound $\tilde{\mu}_\ell^{2\delta}$ appearing in the "staircase lemma" (Lemma 5.4). A fixed point construction for the more natural condition $\delta < 1$ is definitely possible with our proof, but such upper bounds would not allow grids to contain rectangles of arbitrary small sizes (*e.g.* $\mu(\mathfrak{g}_\ell) = 2$).

The bound on the color lengths $\|x^{(\ell)}\| = \mathcal{O}(\tilde{\mu}_{\ell-1}^{\alpha})$, which is less natural than the expected $\|x^{(\ell)}\| = \mathcal{O}(\tilde{\mu}_\ell^{\alpha})$, is also a proof artifact. It comes up when synchronizing information between consecutive levels of the construction. While results from [Wes17; Des21] rely on global assumptions on the density of information in their specific shifts to solve the communication between levels as a graph flow problem, such a solution cannot be applied to a generic statement such as ours[13].

We later discuss the differences between our version of the fixed point construction and existing applications from the literature in Section 6.3.3.

The rest of Section 5 is entirely dedicated to the proof of Lemma 5.2. Sections 6 and 7 will respectively focus on more intuitive corollaries of Lemma 5.2 and their applications.

§ 5.0. Overview of the fixed point construction

At its core, the fixed point construction is based on a notion of *block simulation*:

Definition 5.3 (Block simulation)**.** Given two sets of tiles S, T, a *simulation* of T by S is a substitution $\sigma\colon T \rightrightarrows S^{\mathfrak{R}_0}$ of T-tiles into valid rectangular patterns of S-tiles such that:

(i) σ is "injective": for $w \in S^{\mathfrak{r}}$, if there exists $t, t' \in T$ such that $w \in \sigma(t)$ and $w \in \sigma(t')$, then $t = t'$;
(ii) σ^{-1} preserves local validity: for all $t, t' \in T$, and for all $(w, w') \in \sigma(t) \times \sigma(t')$, if the patterns w and w' are compatible along the direction $\boldsymbol{e}^k$ (*i.e.* satisfy $f_k^+(w) = f_k^-(w')$), then so do the tiles t and t';
(iii) Unique structure: for all valid $y \in S^{\mathbb{Z}^d}$, there exists a unique grid $\mathfrak{g}$ and a unique valid $x \in T^{\mathbb{Z}^d}$ such that $x \xrightarrow[\mathfrak{g}]{\sigma} y$.

Intuitively, each S-tiling realizes a unique T-tiling up to a rectangular substitution. We call *macro-tiles* the rectangular S-patterns that are images of T by σ.

[13] Nevertheless, we still recover [Wes17; Des21] in Section 7: indeed, we argue that such density assumptions actually translate into alternative substitutions *of bounded sizes* computing the same space $\overleftarrow{X}_{\langle\tau\rangle}$.

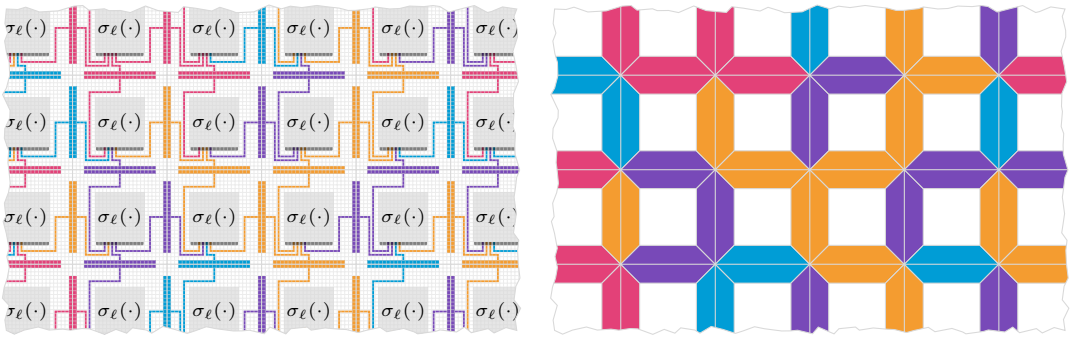


FIGURE 5. Schematics of a macro-tile, and a partial configuration of macro-tiles.

5.0.1. The classical construction. Usually defined on $\mathbb{Z}^2$, the fixed point construction creates a hierarchy of tilesets $(S_\ell)_{\ell\geq 1}$ such that each $S_{\ell-1}$ simulates S_ℓ with a deterministic simulation $\sigma_\ell \colon S_\ell \to S_{\ell-1}^{\mu_\ell\times\mu_\ell}$ of zoom $\mu_\ell \times \mu_\ell$ (for some size $\mu_\ell \in \mathbb{N}$).

To realize this hierarchy of simulations, the tiles of $S_{\ell-1}$ will arrange themselves into blocks of size $\mu_\ell \times \mu_\ell$ called *macro-tiles*, thus forming a regular grid $\mathfrak{g}_\ell$ such that all rectangles $\mathfrak{r} \in \mathfrak{g}_\ell$ satisfy $[\mathfrak{r}] = [\![\mu_\ell]\!]^2$. These macro-tiles then emulate the behavior of Wang tiles:

(1) If a tile of $S_{\ell-1}$ appears on the border of a macro-tile, it carries an additional bit;
(2) The concatenations of those bits on each side of a macro-tile form its four *macro-colors*;

When placing such macro-tiles next to one another along the grid $\mathfrak{g}_\ell$, adjacent macro-tiles share a common macro-color along their common facet; and thus, tilings of macro-tiles will behave like tilings of classical Wang tiles. In order to restrict the macro-tiles (whose macro-colors currently cover all binary strings of length μ_ℓ) to the tileset S_ℓ (which may only contain a subset of 4-tuples of macro-colors):

(3) We embed arbitrary computations by drawing, inside each macro-tile, the space-time diagram of a Turing machine computing the simulation σ_ℓ (and, thus, the tileset S_ℓ);

In particular, the zoom μ_ℓ of the simulation σ_ℓ between $S_{\ell-1}$ and S_ℓ *must be large enough* to provide enough time for those computations to terminate.

Notice that this method requires to already know the programming of the tileset S_ℓ *before* defining $S_{\ell-1}$. In order to define an infinite sequence of tilesets $(S_\ell)_{\ell\geq 1}$ such that $S_{\ell-1}$ simulates S_ℓ, we circumvent this dependency by using the fixed point theorem (Proposition 3.6), which defines a unique program upon which all these tilesets can be based.

5.0.2. Generalizing to non-uniform d-dimensional rectangles. Lemma 5.2 is stated for grids $(\mathfrak{g}_\ell)_{\ell\geq 1}$ whose rectangles $\mathfrak{r} \in \mathfrak{g}_\ell$ might draw various shapes and sizes. We thus generalize macro-colors and macro-tiles from 2-dimensional squares to d-dimensional rectangles, and:

- All computability conditions (*e.g.* time complexity) are stated for uniform bounds on "space" $\mu_{\ell+1} \leq \mu(\mathfrak{g}_{\ell+1}) = \inf_{\mathfrak{r}\in\mathfrak{g}_{\ell+1}} \mu(\mathfrak{r})$ and "time" $\lambda_{\ell+1} \geq \lambda(\mathfrak{g}_{\ell+1}) = \sup_{\mathfrak{r}\in\mathfrak{g}_{\ell+1}} \lambda(\mathfrak{r})$;
- We embed computations by drawing the space-time diagrams of a *processor array*, whose parallelism enables the d-dimensional geometry to operate in time $\mathcal{O}(n)$ on objects of size $\mathcal{O}(n^{d-1})$ (as opposed to Turing machines, in which space is always bounded by time);
- The transmission of information (*e.g.* macro colors) inside a macro-tile of rectangular domain now relies on the routing lemma (Lemma 4.4).

5.0.3. Computing the substitution τ. In order to compute the substitution τ given by Lemma 5.2, we are left with a major difficulty: while the fixed point construction classically requires the sizes $(\mu(\mathfrak{g}_\ell))_{\ell\geq 1}$ to grow significantly to provide enough time for the computations of the simulations $(\sigma^{(\ell)})_{\ell\geq 1}$ to terminate, there might exist a substitution step $x^{(\ell)} \xrightarrow[\mathfrak{g}_\ell]{\tau} x^{(\ell-1)}$ such that the grid $\mathfrak{g}_\ell$ contains arbitrarily small rectangles (*e.g.* $[\mathfrak{r}] = [\![2]\!]^d$).

Our main idea for this proof consists in compressing the computation of several substitution steps for τ at the same level of simulation. More precisely, we make a level of simulation ℓ compute several substitution steps

$$x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}, \quad x^{(\ell+2)} \xrightarrow[\mathfrak{g}_{\ell+2}]{\tau} x^{(\ell+1)}, \quad x^{(\ell+3)} \xrightarrow[\mathfrak{g}_{\ell+3}]{\tau} x^{(\ell+2)}, \quad \text{etc.}\dots$$

until the product $\mu(\mathfrak{g}_{\ell+1}) \cdot \mu(\mathfrak{g}_{\ell+2}) \cdot \dots$ becomes large enough to compute a simulation step. For a given sequence $(\mu(\mathfrak{g}_\ell))_{\ell \geq 1}$, we thus define a function $\kappa \colon \mathbb{N} \to \mathbb{N}$ that partitions $\mathbb{N}$ into segments $\{\kappa(n), \dots, \kappa(n+1) - 1\}$, which correspond to substitutions steps whose computations are actually implemented at the same level of simulation.

5.0.4. Interleaving computations of the substitution τ. Such segments $\{\kappa(n), \dots, \kappa(n+1) - 1\}$ might unfortunately define rectangles so large that the tileset $S_{\kappa(n)}$ can no longer simulate $S_{\kappa(n+1)}$ (for example, a macro-tile might become too big for its size to fit (in binary) inside the computation space of the tiles of $S_{\kappa(n)}$). To solve this issue, we actually interleave the computations of simulations and substitutions steps. More precisely, we introduce a second function $\iota \colon \mathbb{N} \to \mathbb{N}$ such that:

- We build a sequence of tilesets $(S_{\iota(n)})_{n \in \mathbb{N}}$ such that each $S_{\iota(n)}$ simulates $S_{\iota(n+1)}$ using the fixed point construction;
- For all $n \in \mathbb{N}$, we have $\iota(n) < \kappa(n)$ so that $S_{\iota(n)}$ can embed the computation of all the substitution steps $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$ for $\ell \in \{\kappa(n), \dots, \kappa(n+1) - 1\}$;
- And for all $n \in \mathbb{N}$, we have $\iota(n+1) \in \{\kappa(n), \dots, \kappa(n+1) - 1\}$ low enough so that $S_{\iota(n)}$ can simulate $S_{\iota(n+1)}$.

§ 5.1. Deciding the hierarchy of substitutions and simulations

The whole construction actually relies on the following lemma, which proves that such interleavings of simulations $\iota(n+1)$ inside substitution $\{\kappa(n), \dots, \kappa(n+1) - 1\}$ segments exist.

FIGURE 6. The functions $\iota \colon \mathbb{N} \to \mathbb{N}$ and $\kappa \colon \mathbb{N} \to \mathbb{N}$ from the "staircase lemma".[14]
In the proof, the levels $(\iota(n))_{n \in \mathbb{N}}$ will refer to successive levels of fixed point simulation. Each level $\iota(n)$ will also be responsible for computing the substitutions $x^{(\ell+1)} \xrightarrow{\tau} x^{(\ell)}$ for the block $\ell \in \{\kappa(n), \dots, \kappa(n+1)-1\}$.

Before we proceed with the formal statement Lemma 5.4, let us motivate the numerical quantities involved. If tiles of level $\iota(n)$ organize themselves into macro-tiles of level $\ell \geq \kappa(n)$, then:

- A macro-tile of level ℓ has space larger than $\prod_{i=\iota(n)+1}^{\ell} \mu_\ell$: more precisely, each facet in such a macro-tile contains at least that many tiles of level $\iota(n)$;
- We make macro-tiles of level ℓ embed $\mathcal{O}(\tilde{\mu}_\ell^{\gamma})$ steps of log-RAM computations for some $\gamma < 1$ by embedding the processor array simulations from Lemma 3.8: equation (2) ensures that each facet in such a macro-tile is large enough to embed this many processors;
- In a macro-tile of level ℓ, a tile of level $\iota(n)$ is actually contained in macro-tiles of intermediate levels $\iota(n) < i \leq \ell$. As these rectangles have maximal edge length λ_ℓ, encoding relative positions in binary (equation (3)) requires bit length $\mathcal{O}(\sum_{i=\iota(n)+1}^{\ell} \log \lambda_i)$;
- Since variables in a processor array have logarithmic bit length, and that $\mathcal{O}(\tilde{\mu}_\ell{}^{\gamma})$ processors are embedded in the macro-tiles of level ℓ, the tiles of level $\iota(n)$ require $\mathcal{O}(\log \tilde{\mu}_i)$ bits to encode a processor state for each intermediate level $\iota(n) < i \leq \ell$ (equation (4)).

[14]The staircase lemma was also named the « écluse » lemma, from the French word denoting the devices that can raise/lower boats in a waterway to transition between sections of different heights. To their dismay, the authors later learned that such devices are called "locks" in English, which looses all the intended meaning. We thus settled for the terminology "staircase", although this staircase is definitely broken.

Lemma 5.4 (Staircase lemma)**.** *Let $\delta, \gamma \in \mathbb{R}_+$ bet two real numbers such that $2\delta < \gamma < 1$, and let $(\mu_\ell, \lambda_\ell)_{\ell\in\mathbb{N}}$ be sequences of integers such that $\lambda_\ell^{d-1} \geq \mu_\ell$ and satisfying $2 \leq \lambda_{\ell+1} \leq 2^{\mathcal{O}(\tilde{\mu}_\ell^\delta)}$ and $\mu_{\ell+1}^{\mathcal{O}(\log \tilde{\mu}_\ell)} \geq \lambda_{\ell+1}$ for $\tilde{\mu}_\ell = \prod_{i\leq \ell} \mu_i$. For $K \in \mathbb{R}_+$, define $\iota\colon \mathbb{N} \to \mathbb{N}$ and $\kappa\colon \mathbb{N} \to \mathbb{N}$ as $\iota(0) = \kappa(0) = 0$, and:*

$$\kappa(n+1) = \min\big\{\ell > \kappa(n) : \prod_{i=\kappa(n)+1}^{\ell} \mu_i > K \cdot \tilde{\mu}_\ell^\gamma\big\}$$
$$\iota(n+1) = \max\{\ell < \kappa(n+1) : \prod_{i=\ell+1}^{\kappa(n+1)} \mu_i > K \cdot \tilde{\mu}_{\kappa(n+1)}^\gamma\big\}$$

Then if $K \in \mathbb{R}_+$ is large enough, ι and κ satisfy for all $n \in \mathbb{N}$:

$$\kappa(n) \leq \iota(n+1) < \kappa(n+1) \tag{1}$$

$$\prod_{\ell=\iota(n)+1}^{\kappa(n)} \mu_\ell \geq K \cdot \tilde{\mu}_{\kappa(n)}^\gamma \tag{2}$$

$$\sum_{\ell=1}^{\kappa(n+1)-1} \log \lambda_\ell \leq \operatorname{polylog} K \cdot \tilde{\mu}_{\iota(n)}^{2\delta} \tag{3}$$

$$\sum_{\ell=1}^{\kappa(n+1)-1} \log \tilde{\mu}_\ell \leq \operatorname{polylog} K \cdot \tilde{\mu}_{\iota(n)}^{2\delta} \tag{4}$$

Since $\gamma < 1$, (2) also implies that for every $\ell \in \{\kappa(n), \dots, \kappa(n+1)-1\}$, we have $\prod_{\ell'=\iota(n)+1}^{\ell} \mu_{\ell'} \geq K \cdot \tilde{\mu}_\ell^\gamma$.

Proof. First, the functions κ and ι are well-defined: since $\gamma < 1$ and that the product $\prod_{\ell=\kappa(n)+1}^{+\infty} \mu_\ell$ diverges, there must exist some ℓ such that $\prod_{i=\kappa(n)+1}^{\ell} \mu_i > (K \cdot \tilde{\mu}_{\kappa(n)}^\gamma) \cdot \prod_{i=\kappa(n)+1}^{\ell} \mu_i^\gamma$. In particular, $\kappa(n+1)$ is well-defined, and in turn so is $\iota(n+1)$. Furthermore, we have $\iota(n+1) < \kappa(n+1)$ by definition of $\iota(n+1)$; and the product $\prod_{i=\kappa(n)+1}^{\kappa(n+1)} \mu_i$ is large enough to ensure that $\iota(n+1) \geq \kappa(n)$.

We now prove that (2), (3) and (4) hold for all $K \in \mathbb{R}_+$ large enough. To do so, fix $K_\lambda \in \mathbb{R}_+$ and $K_\mu \in \mathbb{R}_+$ such that $2 \leq \lambda_{\ell+1} \leq 2^{K_\lambda \cdot \tilde{\mu}_\ell^\delta}$ and $\mu_{\ell+1}^{K_\mu \cdot \log \tilde{\mu}_\ell} \geq \lambda_{\ell+1}$ for all $\ell \in \mathbb{N}$. Then (2) holds by definition, whereas (3) and (4) follow from the following computations:

- First, we have by minimality of $\kappa(n+1)$ that
$$\prod_{\ell=\kappa(n)+1}^{\kappa(n+1)-1} \mu_\ell \leq K \cdot \tilde{\mu}_{\kappa(n+1)-1}^\gamma = K \cdot \tilde{\mu}_{\kappa(n)}^\gamma \cdot \prod_{\ell=\kappa(n)+1}^{\kappa(n+1)-1} \mu_\ell^\gamma;$$
so that, by dividing, taking a log and assuming that K is large enough, we deduce
$$\log \prod_{\ell=\kappa(n)+1}^{\kappa(n+1)-1} \mu_\ell \leq \frac{\gamma}{1-\gamma} \cdot \log \tilde{\mu}_{\kappa(n)} + \log K; \tag{$*$}$$
- Furthermore, by $(*)$, we obtain that for K large enough:
$$\log \tilde{\mu}_{\kappa(n+1)-1} = \log \tilde{\mu}_{\kappa(n)} + \log \prod_{\ell=\kappa(n)+1}^{\kappa(n+1)-1} \mu_\ell \leq \log \tilde{\mu}_{\kappa(n)} \cdot 2 \log K;$$
- And by maximality of $\iota(n)$, and since $\mu_{\ell+1} \leq \lambda_{\ell+1}^{d-1}$ and $\lambda_{\ell+1} \leq 2^{K_\lambda \cdot \tilde{\mu}_\ell^\delta}$ we have
$$\tilde{\mu}_{\kappa(n)} = \tilde{\mu}_{\iota(n)} \cdot \mu_{\iota(n)+1} \cdot \prod_{\ell=\iota(n)+2}^{\kappa(n)} \mu_\ell \leq \tilde{\mu}_{\iota(n)} \cdot 2^{K_\lambda \cdot (d-1) \cdot \tilde{\mu}_{\iota(n)}^\delta} \cdot K \cdot \tilde{\mu}_{\kappa(n)}^\gamma;$$
so that, by dividing, taking a log and assuming that K is large enough, we deduce that
$$\log \tilde{\mu}_{\kappa(n)} \leq (\log K) \cdot \tilde{\mu}_{\iota(n)}^\delta + \log K.$$

From these considerations, we obtain that (3) holds, *i.e.*:

$$\begin{aligned}\sum_{\ell=1}^{\kappa(n+1)-1} \log \lambda_\ell &\leq \sum_{\ell=1}^{\kappa(n+1)-1} K_\mu \cdot \log \tilde{\mu}_\ell \cdot \log \mu_\ell \\ &\leq K_\mu \cdot \log \tilde{\mu}_{\kappa(n+1)-1} \cdot \log \prod_{\ell=1}^{\kappa(n+1)-1} \mu_\ell \\ &\leq K_\mu \cdot (\log \tilde{\mu}_{\kappa(n+1)-1})^2 \\ &\leq K_\mu \cdot (2 \log K)^4 \cdot \tilde{\mu}_{\iota(n)}^{2\delta}\end{aligned}$$

if $K \in \mathbb{R}_+$ is large enough to satisfy $K \geq K_\mu \cdot (2 \log K)^4$. Furthermore, since all μ_ℓ satisfy $\mu_\ell \geq 2$, we deduce and obtain by $(*)$ that for K large enough:

$$\kappa(n+1) - 1 = \log 2^{\kappa(n+1)-1} \leq \log \prod_{\ell=1}^{\kappa(n+1)-1} \mu_\ell \leq \log \tilde{\mu}_{\kappa(n+1)-1} \leq \log \tilde{\mu}_{\kappa(n)} \cdot 2 \log K;$$

from which we deduce (4) if K is large enough again:

$$\begin{aligned}\sum_{\ell=1}^{\kappa(n+1)-1} \log \tilde{\mu}_\ell &\leq (\kappa(n+1)-1) \cdot \log \tilde{\mu}_{\kappa(n+1)-1} \\ &\leq \big(2 \log K \cdot \log \tilde{\mu}_{\kappa(n)}\big)^2 \\ &\leq (2 \log K)^4 \cdot \log \tilde{\mu}_{\iota(n)}^{2\delta}\end{aligned}$$

□

§ 5.2. Algorithm

We now begin the proof of Lemma 5.2. From any RAM program $e \in \{0,1\}^*$, we will define a new log-RAM program $F(e) \in \{0,1\}^*$ by computably transforming e, and such that $F(e)$ defines a non-deterministic function of the form:

$$\begin{aligned}\varphi_{F(e)}\big((K, K_{\mathrm{word}}), \iota, (\mu_\ell, \lambda_\ell)_{1 \leq \ell < \kappa}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa}, c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+\big) \\ \rightrightarrows \big(c_{\blacksquare}, \mathfrak{r}_\kappa, \boldsymbol{i}_\kappa, u\big)\end{aligned}$$

where[15]

- (K, K_{word}) is a pair of constants $K, K_{\mathrm{word}} \in \mathbb{N}$;
  Informally, they are used to hardcode $\mathcal{O}(...)$ constants from the staircase lemma and the word length of the program $e \in \{0,1\}^*$.
- $\iota \in \mathbb{N}$ is an integer;
  Informally, ι denotes the current level of the simulation.
- $(\mu_\ell, \lambda_\ell)_{1<\ell<\kappa} \in (\mathbb{N}^2)^{\kappa+1}$ is a family of integer bounds;
  Informally, $(\mu_\ell, \lambda_\ell) \in \mathbb{N}^2$ are respectively lower and upper bounds on the space and time of the grids $\mathfrak{g}_\ell$ ($\ell \leq \kappa$) defined by the previous steps of substitutions and simulations.
- $(c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+) \in (\{0,1\}^*)^{2d+1}$ is a $(2d+1)$-tuple of binary strings;
  Informally, $(c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+)$ represents a d-dimensional Wang tile t of the infinite tileset T_*.
- $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa}$ is a (possibly empty) family of rectangles $\mathfrak{r}_\ell \in \mathfrak{R}_0$ and positions $\boldsymbol{i}_\ell \in \mathfrak{r}_\ell$;
  Informally, $\boldsymbol{i}_\ell$ represents the position of the Wang tile t inside its rectangle $\mathfrak{r}_\ell = [\mathfrak{r}]$ for $\mathfrak{r} \in \mathfrak{g}_\ell$, where the grids $\mathfrak{g}_\ell$ ($\iota < \ell < \kappa$) were defined by the previous steps of substitutions and simulations.

and

- $c_{\blacksquare} \in \{\square, \blacksquare\}$ either refers to white or black;
  Informally, $c_{\blacksquare}$ returns the color of the tile t is a chessboard drawn at level κ.
- $(\mathfrak{r}_\kappa, \boldsymbol{i}_\kappa)$ is a rectangle $\mathfrak{r}_\kappa \in \mathfrak{R}_0$ and a position $\boldsymbol{i}_\kappa \in \mathfrak{r}_\kappa$;
  Informally, $\boldsymbol{i}_\kappa$ refers to the position of the Wang tile t inside its rectangle $\mathfrak{r}_\kappa = [\mathfrak{r}]$ for $\mathfrak{r} \in \mathfrak{g}_\kappa$, where $\mathfrak{g}_\kappa$ is one of the grids defined at the current level of simulation.
- $u \in \{0,1\}^*$ is either blank, or encodes a tuple $(i,j,b) \in (\mathbb{N} \times \{0,1\})$ for (i,j) a pair of integers and b a single bit.
  Intuitively, a non-blank tuple (i,j,b) corresponds to a bit $b \in \{0,1\}$ of index $(i,j) \in \{0,\dots,2d\} \times \mathbb{N}$ needed by the tile t from the previous level of simulation.

[15]Notice that most of these variables are not written as binary words in $\{0,1\}^*$, but as data types of the form $\mathbb{N}$, $(\{0,1\}^*)^{2d+1}$, etc... As mentioned in Remark 3.2, it is usual in computability theory to consider such elements both as data structures and binary words through implicit encodings.

Proof strategy. Given any RAM program $e \in \{0,1\}^*$, and for fixed parameters (K, K_{word}), ι and $(\mu_\ell, \lambda_\ell)_{1\leq \ell<\kappa}$, *the non-deterministic function $\varphi_{F(e)}$ will define a set of accepted Wang tiles* $(c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+) \in T_*$. Shortening notations, we denote by $S_\iota \subseteq T_*$ the set

$$S_\iota = \bigcup_{(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa}} \left\{(c_d^-, \dots, c_d^+) \in T_* \ : \ \varphi_{F(e)}((K, K_{\text{word}}), \iota, (\mu_\ell, \lambda_\ell)_{1\leq \ell<\kappa}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa}, c_d^-, \dots, c_d^+)\downarrow\right\}$$

of Wang tiles t for which there exists a family of rectangles and positions $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa}$ such that $\varphi_{F(e)}(\dots, t)$ admits an accepting computation; and by $\mathcal{C}_\iota \subseteq \{0,1\}^*$ the associated colors; although one should keep in mind that both sets S_ι and $\mathcal{C}_\iota$ strongly depend on the fixed parameters $(K, K_{\text{word}}) \in \mathbb{N}^2$ and $(\mu_\ell, \lambda_\ell) \in (\mathbb{N}^2)^*$.

Informally, we will design the tiles in S_ι to ensure that, for grids $(\mathfrak{g}_\ell)_{1\leq \ell<\kappa}$:

- Any tiling of S_ι defines some new levels of grids $(\mathfrak{g}_\ell)_{\kappa\leq \ell<\kappa'}$;
- Any tiling of S_ι simulates a tiling of $S_{\iota'}$ for some level ι' in the interval $\kappa \leq \iota' < \kappa'$;
- Any tiling of S_ι emulates valid T_*-tilings $x^{(\kappa)}, \dots, x^{(\kappa')}$ such that $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$.

The tileset $S_\iota \subseteq T_*$ will be progressively designed by defining restrictions on the tuples of colors $(c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+)$ accepted by the RAM algorithm $F(e) \in \{0,1\}^*$. By implicitly encoding binary strings into other data types, we will consider that valid colors of $\mathcal{C}_\iota \subseteq \{0,1\}^*$ actually represent a *record*[16], *i.e.* a collection of several data fields: for both the authors' and the reader's convenience, this proof will identify these fields by explicit names.

The proof below thus consists in listing the fields appearing in the colors $\mathcal{C}_\iota$, and the constraints a $(2d+1)$-tuple of them must satisfy to form a valid tile $(c_d^-, \dots, c_1^-, c_0, c_1^+, \dots, c_d^+) \in S_\iota$. The algorithm $F(e) \in \{0,1\}^*$ will thus be defined somewhat implicitly, as it mostly amounts to checking whether these constraints are satisfied.

Housekeeping. Before we begin the proof, let us introduce the objects given by Lemma 5.2: let $\mathcal{A} \subseteq \{0,1\}^*$ be a finite alphabet and $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ a sofic shift space on $\mathcal{A}$. Fix $\langle\tau\rangle \in \{0,1\}^*$ a log-RAM program, $\alpha < 1$ and $\delta < \frac{1}{2}$ two rationals satisfying the hypotheses of Lemma 5.2.

Before we begin the proof, let us introduce some integer constants K_λ, K_μ, $K^{(t)}_{\langle\tau\rangle}$ and $K^{(w)}_{\langle\tau\rangle} \in \mathbb{N}$ fixing the $\mathcal{O}(\dots)$ bounds from Lemma 5.2, *i.e.* such that

(1) (i) $2 \leq \lambda(\mathfrak{g}_{\ell+1}) \leq 2^{K_\lambda \cdot \tilde{\mu}_\ell^{\delta}}$;
(ii) $\mu_\ell^{K_\mu \cdot \log \tilde{\mu}_{\ell-1}} \geq \lambda_\ell$;

(2) (ii) The bit length in $x^{(\ell)}$ satisfies $\|x^{(\ell)}\| \leq K^{(t)}_{\langle\tau\rangle} \cdot \tilde{\mu}_{\ell-1}^{\alpha}$;
(iii) The substitution $x^{(\ell)} \xrightarrow[\mathfrak{g}_\ell]{\tau} x^{(\ell-1)}$ is computed in time at most $K^{(t)}_{\langle\tau\rangle} \cdot \tilde{\mu}_{\ell-1}^{\alpha}$;

and such that $\langle\tau\rangle \in \{0,1\}^*$ is a log-RAM program satisfying $W_{\langle\tau\rangle}(\mathtt{I}) \leq K^{(w)}_{\langle\tau\rangle} \log|\mathtt{I}|$. Furthermore, for the remainder of the proof, we fix a rational $\gamma < 1$ such that $\alpha < \gamma$, $2\delta < \gamma$ and $\frac{1}{2} < \gamma$.

§ 5.3. Hierarchical positioning

Let us fix $\iota \in \mathbb{N}$ an integer corresponding to the current level of tiles; (K, K_{word}) two integers, and a sequence of pairs $(\mu_\ell, \lambda_\ell)_{1\leq \ell<\kappa} \in (\mathbb{N}^2)^*$ as given as input to $\varphi_{F(e)}$. Notice that the latter determines a value $\kappa \in \mathbb{N}$, *via* its length.

The tileset S_ι resulting from these parameters will embed some "next" steps of substitutions

$$x^{\kappa'} \xrightarrow[\mathfrak{g}_{\kappa'}]{\tau} x^{\kappa'-1} \xrightarrow{\tau} \dots \xrightarrow{\tau} x^{(\kappa+1)} \xrightarrow[\mathfrak{g}_{\kappa+1}]{\tau} x^{(\kappa)}$$

for $\mathfrak{g}_{\kappa+1}, \dots, \mathfrak{g}_{\kappa'}$ some grids; and decide on a level $\iota' \geq \kappa$ for the next level of simulation. To proceed, the tileset S_ι will start by guessing the grids $\mathfrak{g}_\kappa, \dots, \mathfrak{g}_{\kappa'}$ and the uniform bounds on time $\lambda(\mathfrak{g}_\ell)$ and space $\mu(\mathfrak{g}_\ell)$ for these new grids.

[16] Also known as "compound data type" or `struct`, a record allows to define a collection of several variables (called *fields*), possibly of different data types, grouped together into a single value. The fields in a record are typically identified by some explicit names.

Uniform bounds. The tilings over S_ι start by "guessing" an integer $\kappa' > \kappa$ and draw a uniform family of bounds $(\mu_\ell, \lambda_\ell)_{\kappa \le \ell < \kappa'}$. To do so, we add a **bound field** to the colors of $\mathcal{C}_\iota$, which contains a family of bounds

$$(\mu_\ell, \lambda_\ell)_{\kappa \le \ell < \kappa'} \in (\mathbb{N}^2)^*$$

such that

$$\prod_{i=\iota+1}^{\kappa} \mu_i \ge K \cdot \tilde{\mu}_\kappa^\gamma \tag{b1}$$

and for every $\ell \in \{\kappa, \dots, \kappa' - 1\}$,

$$\lambda_\ell \le 2^{K_\lambda \cdot \tilde{\mu}_{\ell-1}^\delta} \quad ; \quad \mu_\ell^{K_\mu \cdot \log \tilde{\mu}_{\ell-1}} \ge \lambda_\ell \quad \text{and} \quad \prod_{i=\kappa+1}^{\ell} \mu_i < K \cdot \tilde{\mu}_\ell^\gamma \tag{b2}$$

for K_λ and K_μ the constants fixed in housekeeping paragraph, K the constant given as input to $\varphi_{F(e)}$ and $\tilde{\mu}_\ell$ the integer defined as $\tilde{\mu}_\ell = \prod_{i=1}^{\ell} \mu_i$. In particular, (b1) and (b2) ensure that, if the prefix $(\mu_\ell, \lambda_\ell)_{1 \le \ell < \kappa}$ satisfies the staircase lemma, then so does the larger segment $(\mu_\ell, \lambda_\ell)_{1 \le \ell < \kappa'}$; and that parameters κ and ι are consistent with the values given by said lemma[17].

> *Details.* For any tile in $t \in S_\iota$, the same data $(\mu_\ell, \lambda_\ell)_{\kappa \le \ell < \kappa'}$ must appear in the **bound fields** of all its facets $f_k^\pm(t)$, thus ensuring the uniformity of these bounds across entire configurations drawn by the tiles of S_ι. The same data is also copied to the decoration $f_\blacklozenge(t)$. □

Grids and positions. The tilings over S_ι must then draw the grids $\mathfrak{g}_\kappa, \dots, \mathfrak{g}_{\kappa'}$. To do so, we add a **position field** to the colors of $\mathcal{C}_\iota$ which contains a family of the form

$$(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'} \in (\mathfrak{R}_0, \mathbb{N}^d)^*$$

for $\mathfrak{r}_\ell \in \mathfrak{R}_0$ rectangles such that $\mu(\mathfrak{r}_\ell) \ge \mu_\ell$ and $\lambda(\mathfrak{r}_\ell) \le \lambda_\ell$; and $\boldsymbol{i}_\ell \in \mathfrak{r}_\ell$ positions in $\mathfrak{r}_\ell$. The prefix $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa}$ must equal the eponymous input parameters of $\varphi_{F(e)}$. In an "odometer-like" fashion, the **position fields** in an S_ι-tiling will uniquely determine a finitely many grids $\mathfrak{g}_{\iota+1}, \dots, \mathfrak{g}_{\kappa'-1}$.

Definition 5.5 (Odometer)**.** For $(\mathfrak{r}_\ell)_{\iota < \ell < \kappa'}$ a family of rectangles from $\mathfrak{R}_0$, the associated *partial odometer* defines, for every $1 \le k \le d$, an addition

$$(\boldsymbol{i}_\ell)_{\iota < \ell < \kappa'} + \boldsymbol{e}^k = (\boldsymbol{j}_\ell)_{\iota < \ell < \kappa'} \qquad \text{for } \boldsymbol{j}_\ell \in \mathfrak{r}_\ell \cup \{\Diamond\},$$

where

$$\boldsymbol{j}_\ell = \begin{cases} \boldsymbol{i}_\ell & \text{if there exists } \ell' < \ell \text{ such that } \boldsymbol{i}_{\ell'} \notin f_k^+(\mathfrak{r}_{\ell'}) \\ \boldsymbol{i}_\ell + \boldsymbol{e}^k & \text{if for all } \ell' < \ell \text{ we have } \boldsymbol{i}_{\ell'} \in f_k^+(\mathfrak{r}_{\ell'}), \text{ and } \boldsymbol{i}_\ell + \boldsymbol{e}^k \in \mathfrak{r}_\ell \\ \Diamond & \text{if for all } \ell' < \ell \text{ we have } \boldsymbol{i}_{\ell'} \in f_k^+(\mathfrak{r}_{\ell'}), \text{ but } \boldsymbol{i}_\ell + \boldsymbol{e}^k \notin \mathfrak{r}_\ell. \end{cases}$$

Similarly, one can define the subtraction $(\boldsymbol{i}_\ell)_{\iota < \ell < \kappa'} - \boldsymbol{e}^k$ using facets $f_k^-(\mathfrak{r}_\ell)$. Odometers intuitively generalize additions with carry in a d-dimensional setting, and having a result $\boldsymbol{i}'_\ell = \Diamond$ implies that an overflow occurred in the rectangle $\mathfrak{r}_\ell$.

> *Details.* For any tile in $t \in S_\iota$, let $(\mu_\ell, \lambda_\ell)_{\iota < \ell < \kappa'}$ be given by the **bound field** of the decoration $f_\blacklozenge(t)$. And let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'}$ be the **position field** of the decoration $f_\blacklozenge(t)$. Then for every direction $1 \le k \le d$:
>
> - The **position field** of the facet $f_k^-(t)$ is the same as the decoration's, *i.e.* is also $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'}$;
> - The **position field** of the facet $f_k^+(t)$ contains $(\mathfrak{r}'_\ell, \boldsymbol{i}'_\ell)_{\iota < \ell < \kappa'}$, where, if we denote $(\boldsymbol{j}_\ell)_{\iota < \ell < \kappa'}$ the result of the partial odometer operation $(\boldsymbol{i}_\ell)_{\iota < \ell < \kappa'} + \boldsymbol{e}^k$, then:
>   - If $\boldsymbol{j}_\ell \in \mathfrak{r}_\ell$, then $\mathfrak{r}'_\ell = \mathfrak{r}_\ell$ and $\boldsymbol{i}'_\ell = \boldsymbol{j}_\ell$;
>   - If $\boldsymbol{j}_\ell = \Diamond$, then $\mathfrak{r}'_\ell$ is any rectangle such that $[f_k^+(\mathfrak{r}_\ell)]$ and $[f_k^-(\mathfrak{r}'_\ell)]$ are equal, and $\boldsymbol{i}'_\ell = \boldsymbol{i}_\ell + \boldsymbol{e}^k \bmod \mathfrak{r}_\ell$. □

Claim 1. *In any valid tiling of the tileset S_ι, the **position fields** of the decorations uniquely determines a sequence of nested grids*

$$\left(\bigcirc_{i=\iota+1}^{\ell} \mathfrak{g}_\ell \right)_{\iota < \ell < \kappa'}$$

[17] Notice that the space bound μ_κ was not given as a parameter for S_ι, and was first introduced in the **bound fields** of S_ι. Thus, such consistency checks for the staircase lemma must be performed in S_ι, and could not have been done at the previous level of fixed point simulation.

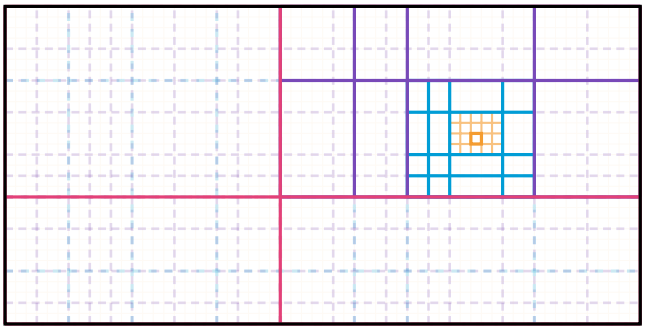


FIGURE 7. The information in the **position field** of $f_{\blacklozenge}(t)$, for t a tile in S_ι.
We draw five levels of grids: individual tiles of S_ι, $\mathfrak{g}_{\iota+1}, \dots, \mathfrak{g}_{\iota+4}$ (with respective colors —, - -, - -, - - and - -). Given the tile t (drawn □), the **position field** of $f_{\blacklozenge}(t)$ encodes the rectangles $\mathfrak{r}_{\iota+1}, \dots, \mathfrak{r}_{\iota+4}$ in which t appears (whose cells are respectively drawn —, —, — and —).

We will denote $\tilde{\mathfrak{g}}_\ell$ the grid $\tilde{\mathfrak{g}}_\ell = \bigcirc_{i=\iota+1}^{\ell} \mathfrak{g}_\ell$ for $\iota < \ell < \kappa'$. It is important to notice that, for $t \in S_\iota$ a tile in such a valid tiling, the **position field** of $f_{\blacklozenge}(t)$ contains the tile's position in the rectangle $[\mathfrak{r}_\ell]$ for the corresponding $\mathfrak{r}_\ell \in \mathfrak{g}_\ell$; but does *not* allow to deduce the exact position of the tile in (nor the size of) the rectangles $[\tilde{\mathfrak{r}}_\ell]$ for the corresponding $\tilde{\mathfrak{r}}_\ell \in \tilde{\mathfrak{g}}_\ell$.

> *Details.* While a single tile t cannot determine its exact position in the rectangle $[\tilde{\mathfrak{r}}_\ell]$, its **position fields** contains enough information to determine whether it appears on an external facet of $[\tilde{\mathfrak{r}}_\ell]$. Indeed, consider a valid S_ι-tiling in which t appears. Denoting $(\tilde{\mathfrak{g}}_\ell)_{\iota<\ell<\kappa'}$ the resulting grids, consider any level $\iota < \ell < \kappa'$. By definition of $\tilde{\mathfrak{g}}_\ell$, there exists some $\tilde{\mathfrak{r}} \in \tilde{\mathfrak{g}}_\ell$ such that t appears at position $\boldsymbol{i} \in [\tilde{\mathfrak{r}}]$. The tile t then appears on the facet $f_k^{\pm}([\tilde{\mathfrak{r}}])$ if and only if, denoting $(\mathfrak{r}_i, \boldsymbol{i}_i)_{\iota<i<\kappa'}$ the **position field** of $f_{\blacklozenge}(t)$, the operation $(\boldsymbol{i}_i)_{\iota<i\leq\ell} \pm \boldsymbol{e}^k$ in the partial odometer of rectangles $(\mathfrak{r}_i)_{\iota<i\leq\ell}$ returns a full $\lozenge$-sequence. □

> *Numerics.* Let $t \in S_\iota$ be a tile, and denote $(\mu_\ell, \lambda_\ell)_{1\leq\ell<\kappa+1}$ the sequence obtained by the **bound fields** and the parameters of $\varphi_{F(e)}$ fixed for S_ι. Then the staircase lemma (Lemma 5.4) ensures that the **bound fields** and the **position fields** of t have bit length bounded by:
>
> $$\sum_{\ell=1}^{\kappa'-1} \log \mu_\ell + \log \lambda_\ell \leq \operatorname{polylog} K \cdot \tilde{\mu}_\iota^{2\delta}.$$
>
> We chose to make the **position fields** encode positions in this "odometer-like" fashion (*i.e.* relatively to the rectangles $[\mathfrak{r}_\ell]$ from the grid $\mathfrak{g}_\ell$) instead of the more traditional "pixel-like" way (*i.e.* relatively to the rectangles $[\tilde{\mathfrak{r}}_\ell]$ from the product grid $\tilde{\mathfrak{g}}_\ell$) because it allows a better bound $\delta < \frac{1}{2}$ instead of $\delta < \frac{1}{3}$: by the staircase lemma the latter position scheme would require bit length:
>
> $$\sum_{i=\iota+1}^{\ell} \log \left(\prod_{j=\iota+1}^{i} \mu_j \cdot \lambda_i \right) \leq d \cdot \sum_{i=\iota+1}^{\ell} \sum_{j=\iota+1}^{i} \log \lambda_j = \mathcal{O}(\tilde{\mu_\iota}^{3\delta}).$$ □

Chessboards. Unfortunately, the growth bounds from Lemma 5.2 might allow grids $\mathfrak{g}_{\kappa'}$ whose rectangles are too large to fit the allowed word length of $o(\tilde{\mu}_\iota^\gamma)$ bits for the colors of $\mathcal{C}_\iota$. Thus, instead of making each tile of S_ι store a rectangle $\mathfrak{r}_{\kappa'} \in \mathfrak{R}_0$ and its position $\boldsymbol{i}_{\kappa'} \in \mathfrak{r}_{\kappa'}$ as with the previous **position field**, we instead add a **p-chessboard field** encoding a single bit of color

$$p_{\blacksquare} \in \{\square, \blacksquare\}$$

such that, in any valid tiling x of S_ι, the **p-chessboard fields** of the decorations *should* draw a rectangular grid whose rectangles are alternatively colored □ and ■ in a chessboard-like fashion, and thus define the last grid $\mathfrak{g}_{\kappa'}$ geometrically.

> *Details.* For any tile $t \in S_\iota$, the **p-chessboard field** of the decoration $f_{\blacklozenge}(t)$ should be a single bit of color $p_{\blacksquare} \in \{\square, \blacksquare\}$; and let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa'}$ be the **position field** of the decoration $f_{\blacklozenge}(t)$. Then for every direction $1 \leq k \leq d$, we have:
>
> - The **p-chessboard field** of the facet $f_k^-(t)$ is the same as the decoration's, *i.e.* is also $p_{\blacksquare}$;
> - If the partial odometer given by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell<\kappa'}$ yields $(\boldsymbol{i}_\ell)_{\iota<\ell<\kappa'} + \boldsymbol{e}^k \neq (\lozenge, \dots, \lozenge)$, then the **p-chessboard field** of the facet $f_k^+(t)$ is also $p_{\blacksquare}$; otherwise, it can be any of the two colors $\{\square, \blacksquare\}$.
>
> These conditions ensure that the **p-chessboard fields** will be constant inside a rectangle $\mathfrak{r}_{\kappa'-1}$. □

We will later ensure that the **p-chessboard fields** of a valid configuration actually draw a rectangular grid $\mathfrak{g}_{\kappa'}$, and not just a tiling of (possibly infinite) polyominoes, by checking its consistency at the next level of fixed point simulation ι' (which is currently undefined). Since this proof is a fixed point argument, this means that the tileset S_ι will also need to validate the chessboard drawn by the previous simulation level: to do so, we will need to have a second **chessboard field** encoding a single bit of color

$$c_{\blacksquare} \in \{\square, \blacksquare\}$$

drawing a chessboard-like tiling following the grid $\mathfrak{g}_\kappa$.

Details. For any tile $t \in S_\iota$, the **chessboard field** of the decoration $f_{\blacklozenge}(t)$ should be a single bit of color $c_{\blacksquare} \in \{\square, \blacksquare\}$. If $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa'}$ refers to the **position field** of the decoration $f_{\blacklozenge}(t)$, then for every direction $1 \le k \le d$ we have:

- The **chessboard field** of the facet $f_k^-(t)$ is the same as the decoration's, *i.e.* is also $c_{\blacksquare}$;
- If the partial odometer on the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\le\kappa}$ yields $(\boldsymbol{i}_\ell)_{\iota<\ell\le\kappa} + \boldsymbol{e}^k \neq (\Diamond, \dots, \Diamond)$, then the **chessboard field** of the facet $f_k^+(t)$ is also $c_{\blacksquare}$; otherwise, it must be the opposite color. □

§ 5.4. Macro-tiles

As any valid tiling $x \in S_\iota^{\mathbb{Z}^d}$ defines a sequence of nested grids $(\tilde{\mathfrak{g}}_\ell)_{\iota<\ell<\kappa'}$ (Claim 1), we define the *macro-tiles of level* ℓ (for $\iota \le \ell < \kappa'$) as the rectangular patterns $\tilde{t}^{(\ell)} = [x|_{\tilde{\mathfrak{r}}_\ell}]$ for $\tilde{\mathfrak{r}}_\ell \in \tilde{\mathfrak{g}}_\ell$.[18] In other words, a macro-tile of level ℓ is a valid pattern $\tilde{t}^{(\ell)} \in S_\iota^{\tilde{\mathfrak{r}}_\ell}$ (for some $\tilde{\mathfrak{r}}_\ell \in \mathfrak{R}_0$) such that the tiles t appearing in $\tilde{t}$ form a single complete rectangle $\mathfrak{r}_\ell$ (from their decorations' **position field**).

Let us make a few comments on the geometry of macro-tiles:

- Following from the definition of grid products and the induced nested structure of the grids $(\tilde{\mathfrak{g}}_\ell)_{\iota<\ell<\kappa'}$, any macro-tile of level ℓ is a union of macro-tiles of level $\ell - 1$;
- Let $\tilde{t}^{(\ell)}$ be a macro-tile of level ℓ: it uniquely determines the macro-tiles of level $\ell - 1$ whose union constitutes $\tilde{t}^{(\ell)}$. Inductively, it thus determines a unique structure of nested macro-tiles of level ℓ' for every $\ell' < \ell$.

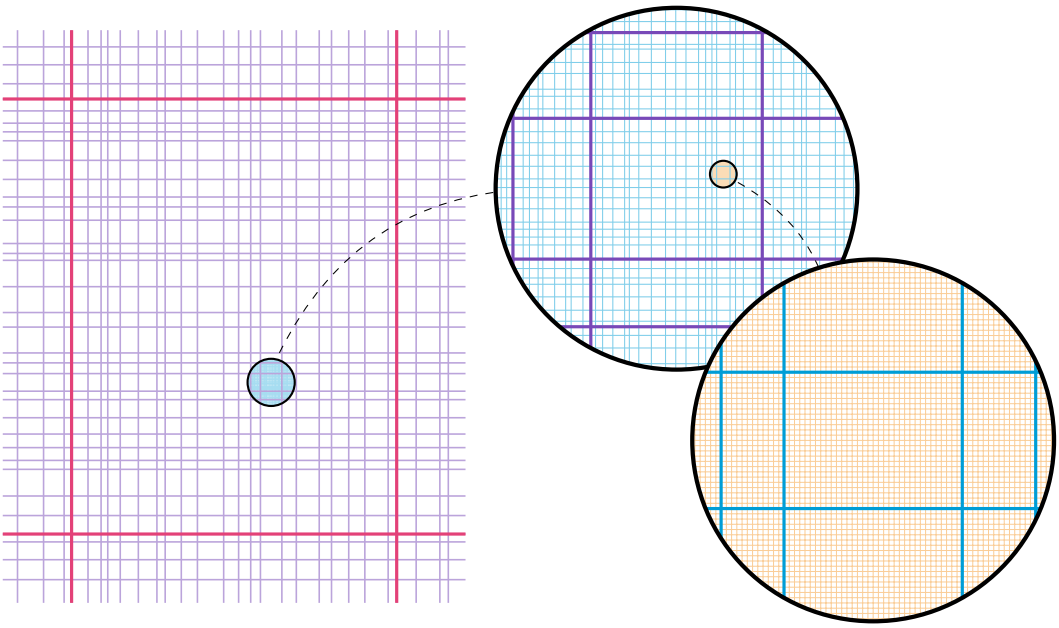


FIGURE 8. The nested structure of a macro-tile of level $\kappa' - 1$.
We represent a macro-tile of level $\kappa' - 1$ (as □) and its sub-macro-tiles of level ι' (as □). Zooming on the level ι', we draw the sub-macro-tiles of level κ (as □); and zooming again, we draw the tiles of S_ι (as □) composing the level κ. This figure implicitly assumes that $\iota' > \kappa$.

[18] Where $\tilde{\mathfrak{g}}_\iota = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ is the single cell grid of rectangles $\mathfrak{r}_{\boldsymbol{i}} = \{\boldsymbol{i}\}$.

§ 5.5. σ-macro-tiles (of level ι')

Since any tiling of S_ι defines grids $\tilde{\mathfrak{g}}_\ell$ for the next levels of indices $\kappa \leq \ell < \kappa'$, we will use one of these levels to implement the next level of fixed point simulation. We denote the latter ι' by analogy with the notations of the staircase lemma.

Level of simulation. We add a **p-level field** to the colors of $\mathcal{C}_\iota$, which contains a value

$$\iota' \in \{\kappa, \dots, \kappa' - 1\}$$

that is constant across any valid tiling of $\mathcal{C}_\iota$. Its role will be to guess the next level of simulation, which (a priori) can be any level in the interval $\{\kappa, \dots, \kappa' - 1\}$.[19]

> *Details.* For any tile $t \in S_\iota$, let κ and κ' be given by the **bound field** of $f_\blacklozenge(t)$. In this case, there exists an integer $\iota' \in \{\kappa, \dots, \kappa' - 1\}$ such that all the **p-level fields** of t (in the facets $f_k^{\pm}(t)$ and the decoration $f_\blacklozenge(t)$) are all equal to ι', thus ensuring the uniformity of the integer ι' across entire configurations drawn by the tiles of S_ι.
> Furthermore, when ranging across the tiles of S_ι whose decoration has a **bound field** determining the same values κ and κ', all values $\iota \in \{\kappa, \dots, \kappa' - 1\}$ appear in the **p-level fields** of the decorations. □

Macro-tiles. The rest of this section focuses on implementing this next level of fixed point simulation. Inside the macro-tiles of level ι', we will draw patterns whose role will be to emulate the behavior of Wang tiles, thus properly defining the tiling of the next simulation. For the rest of this proof, we will call *σ-macro-tile* a macro-tile of level ι' (where ι' matches the **p-level fields**).

5.5.1. Computation zone. We begin the construction of the σ-macro-tiles by designing a way to embed arbitrary computations inside them. To do so, we will define a *computation zone* inside each σ-macro-tile, that will embed the space-time diagram of the processor array defined in Lemma 3.8.

Delimiting the computation zone. For $\tilde{\mathfrak{r}} = [\![n_1, \dots, n_d]\!] \in \mathfrak{R}_0$, we define the *computation zone* of $\tilde{\mathfrak{r}}$ as the cut $\mathrm{Cut}_N(\tilde{\mathfrak{r}}) = [\![\min(n_1, N), \dots, \min(n_d, N)]\!]$, where we fix $N = \tilde{\mu}_{\iota'}/\tilde{\mu}_\iota = \prod_{\ell=\iota+1}^{\iota'} \mu_\ell$ (as read from a tile's **bound fields**).

Claim 2. *For any rectangle $\mathfrak{r} \in \mathfrak{R}_0$ and for any $N \in \mathbb{N}$, if $\mu(\mathfrak{r}) \geq N$, then $\mu(\mathrm{Cut}_N(\mathfrak{r})) \geq N$.*

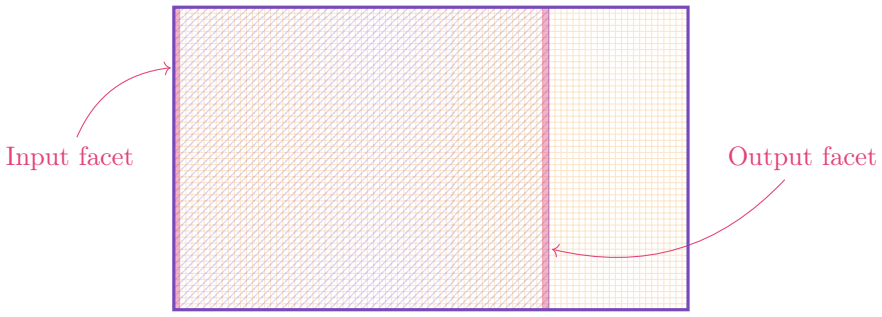


FIGURE 9. The computation zone (in ▨) of a macro-tile $\tilde{t}$ of level ι' (in □).
We draw the tiles of level S_ι composing $\tilde{t}$ as □. The computation zone is a rectangle $\tilde{\mathfrak{c}} = \mathrm{Cut}_N(\tilde{\mathfrak{r}})$ that forms a subset of $\mathrm{dom}(t)$. The area of its smallest facet $f(\tilde{\mathfrak{c}})$ is larger than $K \cdot \tilde{\mu}_{\iota'}^{\gamma}$, but $\tilde{\mathfrak{c}}$ is small enough for the positions $\boldsymbol{i} \in \tilde{\mathfrak{c}}$ to be of bit length $\mathcal{O}(\log \tilde{\mu}_{\iota'})$. The computation zone always contains the position $\mathbf{0} \in \mathrm{dom}(t)$, and may or may not span the whole rectangle $\mathrm{dom}(t)$.

To mark the positions in the computation zone of a σ-macro-tile $\tilde{t}$ of domain $\tilde{\mathfrak{r}}$, we introduce a **compzone field** to the colors of $\mathcal{C}_\iota$ based on the positioning tileset $T_{\mathfrak{R}}$ from Section 4.1, which will contain records of the form

$$(\tilde{\mathfrak{c}}, \boldsymbol{i}) \in \mathfrak{R}_0 \times \mathbb{N}^d$$

for $\tilde{\mathfrak{c}} = \mathrm{Cut}_{\tilde{\mu}_{\iota'}/\tilde{\mu}_\iota}(\tilde{\mathfrak{r}})$ the computation zone of $\tilde{\mathfrak{r}}$ and $\boldsymbol{i} \in \tilde{\mathfrak{c}}$ a position in $\tilde{\mathfrak{c}}$.

[19] The correctness of this choice for ι' will be checked at the next level of simulation, as a valid **bound field** (in $S_{\iota'}$) must satisfy conditions (b1).

Details. For any tile $t \in S_\iota$, denote $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa'}$ and $(\mu_\ell, \lambda_\ell)_{\iota<\ell<\kappa'}$ the content of the **position** and **bound fields** of $f_\blacklozenge(t)$. The **compzone fields** of t can either contain some pair $(\tilde{\mathfrak{c}}, \boldsymbol{i}) \in \mathfrak{R}_0 \times \mathbb{N}^d$, or be left entirely blank.
If the **compzone field** of $f_\blacklozenge(t)$ is blank, then so are the **compzone fields** of all facets $f_k^\pm(t)$. Otherwise, the **compzone field** of $f_\blacklozenge(t)$ is a pair $(\tilde{\mathfrak{c}}, \boldsymbol{i}) \in \mathfrak{R}_0 \times \mathbb{Z}^d$ and:

- Denoting $\tilde{\mathfrak{c}} = [\![n_1, \dots, n_d]\!]$, each n_k satisfies $n_k \leq \prod_{\ell=\iota+1}^{\iota'} \mu_\ell$; and $\boldsymbol{i}$ satisfies $\boldsymbol{i} \in \tilde{\mathfrak{c}}$;
- Furthermore, if $\boldsymbol{i}_\ell = \mathbf{0}$ for all $\iota < \ell \leq \iota'$ (*i.e.* the tile t appears at position $\mathbf{0}$ in σ-macro tiles), then $\boldsymbol{i} \in \tilde{\mathfrak{c}}$ is actually $\boldsymbol{i} = \mathbf{0}$;

Furthermore, for $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ the value of the **compzone field** of $f_\blacklozenge(t)$ and $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'}$ read from the **position field**, the facets $f_k^\pm(t)$ satisfy the following:

- If the partial odometer given by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\leq\iota'}$ defines an operation $(\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} \pm \boldsymbol{e}^k = (\Diamond, \dots, \Diamond)$ (*i.e.* the tile t appears on the facet $f_k^\pm$ of σ-macro-tiles), then the **compzone field** of $f_k^\pm(t)$ is left blank; in which case, $\boldsymbol{i}$ must satisfy $\boldsymbol{i} \in f_k^\pm(\tilde{\mathfrak{c}})$;
- Otherwise, the **compzone field** of the facet $f_k^-(t)$ is the same as the decoration's, *i.e.* $(\tilde{\mathfrak{c}}, \boldsymbol{i})$;
- And the **compzone field** of the facet $f_k^+(t)$ contains $(\tilde{\mathfrak{c}}, \boldsymbol{i} + \boldsymbol{e}^k)$ if $\boldsymbol{i} \notin f_k^+(\tilde{\mathfrak{c}})$ is not on the border of $\tilde{\mathfrak{c}}$; and is left blank otherwise. □

Claim 3. *Let $\tilde{t}$ be a σ-macro-tile of domain $\tilde{\mathfrak{r}}$. Then for any position $\boldsymbol{i} \in \tilde{\mathfrak{r}}$, the **compzone field** of $f_\blacklozenge(\tilde{t}_{\boldsymbol{i}})$ is non-blank if and only if $\boldsymbol{i}$ is a position in the computation zone $\tilde{\mathfrak{c}} = \mathrm{Cut}_{\tilde{\mu}_{\iota'}/\tilde{\mu}_\iota}(\tilde{\mathfrak{r}})$; in which case, said **compzone field** contains exactly the pair $(\tilde{\mathfrak{c}}, \boldsymbol{i})$.*

Numerics. In a tiling of S_ι, the volume of a σ-macro-tile of domain $[\tilde{\mathfrak{r}}] \in \mathfrak{R}_0$ may be larger than $\prod_{\ell=\iota+1}^{\iota'} \mu_\ell \cdot \lambda_\ell$. The computation zone restricts the size of the computations to a cut of $[\tilde{\mathfrak{r}}]$ whose volume is bounded by $\prod_{\ell=\iota+1}^{\iota'} (\mu_\ell)^d$, which limits the word length of the **compzone fields** to polylog $K \cdot \tilde{\mu}_\iota^\delta$. □

Embedding computations. We now use this computation zone to embed the space-time diagram of the RAM-simulating processor array of program $e_{\mathcal{U}} \in \{0,1\}^*$ from Lemma 3.8. Recall that, when used on a grid of processors of domain $[\![n_1, \dots, n_{d-1}]\!]$, the program $e_{\mathcal{U}}$ simulates $\prod_{k=1}^{d-1} n_k$ steps of log-RAM computations in time $\mathcal{O}(\sum_{k=1}^{d-1} n_k)$.

Let $\tilde{\mathfrak{c}} = [\![n_1, \dots, n_d]\!]$ be the computation zone of a σ-macro-tile. Denoting $f(\tilde{\mathfrak{c}})$ a smallest facet $f_k^-(\tilde{\mathfrak{c}})$ of minimal index $1 \leq k \leq d$, and denoting by $h \in \{1, \dots, d\}$ the index of this facet, longest edges in $\tilde{\mathfrak{c}}$ have length $\lambda(\tilde{\mathfrak{c}}) = n_h$. For notational convenience, we will assume that $h = d$; but the general case follows by permuting indices $1 \leq k \leq d$ in the proof below.

We use the facet $f(\tilde{\mathfrak{c}}) = [\![n_1, \dots, n_{d-1}]\!] \times \{0\}$ to draw the processor array $([\![n_1, \dots, n_{d-1}]\!], e_{\mathcal{U}})$ (the "space" of the computation); and use the remaining direction $\boldsymbol{e}^d$ to draw successive states of the array (the "time" of the computation[20]). To do so, we introduce a **processor field** to the colors of $\mathcal{C}_\iota$, which will either contain a tuple of the form

$$(\mathtt{PC}, (\mathtt{var}_i)_{i\in I}) \in \mathbb{N} \times (\{0,1\}^*)^I$$

where $\mathtt{PC}$ is a program counter indexing an instruction of $e_{\mathcal{U}}$, and I is a finite fixed set indexing the variables defined in $e_{\mathcal{U}}$; or two finite lists of adjacent processor communications of the form:

$$((\mathtt{COMM}_i^{\mathtt{in}})_{i\in I_1}, (\mathtt{COMM}_i^{\mathtt{out}})_{i\in I_2}) \in (\{0,1\}^*)^* \times (\{0,1\}^*)^*.$$

In the direction $\boldsymbol{e}^d$, colors $(\mathtt{PC}, (\mathtt{var}_i)_{i\in I})$ will denote consecutive states of a processor during a step of computation; and along the other directions, colors $((\mathtt{COMM}_i^{\mathtt{in}})_{i\in I_1}, (\mathtt{COMM}_i^{\mathtt{out}})_{i\in I_2})$ will draw the communications between adjacent processors. Globally, the **processor fields** in the computation zone $\tilde{\mathfrak{c}}$ will thus draw a space-time diagram of the processor array $[\![n_1, \dots, n_{d-1}]\!]$ for $n_d - 2$ steps of computation.

Remark 5.6 (Linear acceleration). The processor program $e_{\mathcal{U}}$ from Lemma 3.8 terminates in time $\mathcal{O}(n_1 + \cdots + n_{d-1})$ on processor arrays of size $\mathfrak{r} = [\![n_1, \dots, n_{d-1}]\!]$. Thus, there exists a constant $K_{\mathrm{sim}} \in \mathbb{N}$ such that $e_{\mathcal{U}}$ terminates in time $K_{\mathrm{sim}} \cdot \max_{1\leq k\leq d-1} n_k$.

Unfortunately, we only embed space-time diagrams of the processor array $([\![n_1, \dots, n_{d-1}]\!], e_{\mathcal{U}})$ over $n_d = \lambda(\tilde{\mathfrak{c}}) = \max_{1\leq k\leq d} n_k$ steps of computations. To allow enough time for the program $e_{\mathcal{U}}$ to terminate all its valid runs, we actually make each tile in the computation zone embed several (here, $2K_{\mathrm{sim}}$) computation steps of a processor (for a global running time $2K_{\mathrm{sim}} \cdot (\lambda(\tilde{\mathfrak{c}}) - 2)$).

[20]Hence naming *time* and *space* the longest edge length $\lambda(\mathfrak{r})$ and smallest facet area $\mu(\mathfrak{r})$ of a rectangle $\mathfrak{r}$.

Details. For any tile t, the **processor fields** of t are all blank if the **compzone field** of $f_{\diamond}(t)$ is blank. Otherwise, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ refer to the **compzone field** of $f_{\diamond}(t)$.

If $\boldsymbol{i}$ is a position in an external facet $f_k^{\pm}(\tilde{\mathfrak{c}})$, then the **processor field** of the corresponding $f_k^{\pm}(t)$ must be blank. Apart from this constraint, we now consider several cases:

Initializing the computations. Assume that $\boldsymbol{i}$ is part of the smallest facet $f(\tilde{\mathfrak{c}}) = f_d^-(\tilde{\mathfrak{c}})$:

- The **processor field** of the facet $f_d^+(t)$ is some $(\mathtt{PC}, (\mathtt{var}_i)_{i\in I}) \in \mathbb{N} \times (\{0,1\}^*)^I$ for $\mathtt{PC} = 0$ (the entry point of the program $e_{\mathcal{U}}$). Furthermore, the values of the variables $(\mathtt{var}_i)_{i\in I}$ must satisfy the following conditions:
  - The `pos` variable must be set to $\mathtt{pos} = (\mathfrak{r}, \boldsymbol{j})$ where $\mathfrak{r} = [\![n_1, \dots, n_{d-1}]\!]$ and $\boldsymbol{j} = (i_1, \dots, i_{d-1})$ (for $(i_1, \dots, i_{d-1}, 0) = \boldsymbol{i}$);
  - The `word` variable must be set to $\mathtt{word} = 2K_{\text{word}} \cdot \log \mu(\tilde{\mathfrak{c}})$, where K_{word} is the constant appearing in the parameters (K, K_{word}) of the global function $\varphi_{F(e)}$;
  - The `timeout` variable must be set to $\mathtt{timeout} = \mu(\tilde{\mathfrak{c}})$;
  - The `program` variable must be set to $\mathtt{program} = e$, where $e \in \{0,1\}^*$ is the eponymous argument given to F to define $F(e)$;
  - The `input` variables must be set to $\mathtt{input} = (i, j, b)$ for any bit $b \in \{0,1\}$ and any index[a] $(i,j) \in \{0, \dots, 2d+4\} \times \{0, \dots, \|\tilde{\mathfrak{c}}\|\}$. Restrictions on `input` variables will be added later;
  - Any other variable $\mathtt{var}_i$ of the program $e_{\mathcal{U}}$ is assumed to be the empty word $\mathtt{var}_i = \varepsilon$.
- The **processor field** of any other facet $f_k^{\pm}(t)$ $(k \neq d)$ is blank;

Computations. Assume that $\boldsymbol{i}$ is not part of $f_d^-(\tilde{\mathfrak{c}})$ nor its opposite facet $f_d^+(\tilde{\mathfrak{c}})$. Then:

- The **processor fields** of the facets $f_d^-(t)$ and $f_d^+(t)$ must respectively be some $(\mathtt{PC}, (\mathtt{var}_i)_{i\in I})$ and $(\mathtt{PC}', (\mathtt{var}'_i)_{i\in I})$ in $\mathbb{N} \times (\{0,1\}^*)^I$;
- The **processor fields** of the other facets $f_k^{\pm}(t)$ $(k \neq d)$ must contain two lists of length $2K_{\text{sim}}$ whose elements `COMM` are either blank, or encode a tuple (i, w) for $i \in I$ the index of a variable and $w \in \{0,1\}^*$ a value.
  On facets $f_k^-(t)$, the first list (resp. second) is referred to as the `in`-list (resp. `out`-list), and its elements are denoted $\mathtt{COMM}_n^{\mathtt{in}}$ (resp. $\mathtt{COMM}_n^{\mathtt{out}}$) for $0 \leq n < 2K_{\text{sim}}$; conversely, on facets $f_k^+(t)$, the first list and the second lists are respectively referred to as the `out`-list and the `in`-list;[b]

and there exists some computation steps $(\mathtt{PC}^{(n)}, (\mathtt{var}_i^{(n)})_{i\in I})_{0\leq n\leq 2K_{\text{sim}}}$ such that $(\mathtt{PC}^{(0)}, (\mathtt{var}_i^{(0)})_{i\in I}) = (\mathtt{PC}, (\mathtt{var}_i)_{i\in I})$ and $(\mathtt{PC}^{(2K_{\text{sim}})}, (\mathtt{var}_i^{(2K_{\text{sim}})})_{i\in I}) = (\mathtt{PC}', (\mathtt{var}'_i)_{i\in I})$ that satisfies, for all $0 \leq n < 2K_{\text{sim}}$:

- (Incoming communications) If $\mathtt{PC}^{(n)}$ points in the program $e_{\mathcal{U}}$ to a `COMM` instruction reading the variable of index $i \in I$ of the processor in direction $\pm e_k$, then the element $\mathtt{COMM}_n^{\mathtt{in}}$ from the `in`-list in the **processor field** of the corresponding[c] $f_k^{\pm}(t)$ must satisfy $\mathtt{COMM}_n^{\mathtt{in}} = (i, w)$ for some $w \in \{0,1\}^*$; otherwise, $\mathtt{COMM}_n^{\mathtt{in}}$ must blank;
- (Outgoing communications) On any facet $f_k^{\pm}(t)$ for $k \neq d$, if the entry $\mathtt{COMM}_n^{\mathtt{out}}$ in the `out`-list of the **processor field** is non-blank and contains some (i, w) for $i \in I$ and $w \in \{0,1\}^*$, then w must satisfy $w = \mathtt{var}_i^{(n)}$.
- (Computation step) $\mathtt{PC}^{(n+1)}$ and $(\mathtt{var}_i^{(n+1)})_{i\in I}$ must be consistent with a valid step of computation running the instruction of index $\mathtt{PC}^{(n)}$ in $e_{\mathcal{U}}$ on the values of variables $(\mathtt{var}_i^{(n)})_{i\in I}$ (and, in the case of a `COMM` instruction, the value w from the corresponding $\mathtt{COMM}_n^{\mathtt{in}}$ entry).

Finally, if the processor halts at some step $0 \leq n < 2K_{\text{sim}}$, then $\mathtt{PC}^{(n+1)}$ and $(\mathtt{var}_i^{(n+1)})_{i\in I}$ are just copies of $\mathtt{PC}^{(n)}$ and $(\mathtt{var}_i^{(n)})_{i\in I}$.

End of the computation. Assume that $\boldsymbol{i}$ is part of the opposite facet $f_d^+(\tilde{\mathfrak{c}}(c))$. Then:

- The **processor field** of the facet $f_d^-(t)$ must be some $(\mathtt{PC}, (\mathtt{var}_i)_{i\in I}) \in \mathbb{N} \times (\{0,1\}^*)^I$ such that `PC` denotes an accepting state of completed computation in $e_{\mathcal{U}}$;
- The **processor field** of any other facet $f_k^{\pm}(t)$ is blank.

Furthermore, all valid possibilities $(\mathtt{PC}, (\mathtt{var}_i)_{i\in I})$ and $(\mathtt{PC}', (\mathtt{var}'_i)_{i\in I})$ in $\mathbb{N}\times(\{0,1\}^*)^I$ for the **processor fields** of the facets $f_d^-(t)$ and $f_d^+(t)$ appear when ranging across the tiles S_ι with fixed **compzone field** $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ (with $\boldsymbol{i} \notin f_k^{\pm}(\tilde{\mathfrak{c}})$) in their decorations, thus ensuring that all valid computations can actually be drawn in the computation zones of a σ-macro-tile. □

---

[a] For our purposes, the function $\varphi_{F(e)}$ will define a fixed point operating on at most $2d+5$ arguments: we thus restrict i to range in $\{0, \dots, 2d+4\}$.

[b] This is a form of twisted pair communication!

[c] Notice that, in the general case where h can be any direction in the interval $\{1, \dots, d\}$, this requires a re-indexing of the facets between the d-dimensional Wang tiles and the $(d-1)$-dimensional processor array.

In a σ-macro-tile $\tilde{t}$ of computation zone $\tilde{\mathfrak{c}} = [\![n_1, \dots, n_d]\!]$, we respectively call *input facet* and *output facet* the facets $f(\tilde{\mathfrak{c}})$ and its opposite, as their **processor fields** draw the first and the last step of the embedded computation (*c.f.* Figure 9). Assuming that $f(\tilde{\mathfrak{c}}) = f_d^-(\tilde{\mathfrak{c}})$ for notational convenience, the *input facet* of $\tilde{\mathfrak{c}}$ is $[\![n_1, \dots, n_{d-1}]\!] \times \{0\}$; and the *output facet* is $[\![n_1, \dots, n_{d-1}]\!] \times \{n_d - 1\}$. We justify this terminology by the following claim:

Claim 4. *Let $\tilde{t}$ be a σ-macro-tile of computation zone $\tilde{\mathfrak{c}} = [\![n_1, \dots, n_d]\!]$ and smallest facet $f(\tilde{\mathfrak{c}}) = f_d^-(\tilde{\mathfrak{c}})$.*[21] *Then for all $0 \leq n < n_d - 1$, the **processor fields** of the facets $f_d^+(t)$ for tiles t at positions $[\![n_1, \dots, n_{d-1}]\!] \times \{n\}$ draw a state of the processor array $([\![n_1, \dots, n_{d-1}]\!], e_{\mathcal{U}})$ after $2K_{\mathrm{sim}} \cdot n$ steps of computations.*

Folding the arguments of the computation. The previous section embeds arbitrary computations of the log-RAM program $e \in \{0,1\}^*$ (given to F to define $F(e)$) in the form of space-time diagrams of the simulating processor array from Lemma 3.8.

Since said simulation from Lemma 3.8 requires its input and output bits `I[`i`][`j`]` and `O[`i`][`j`]` to be folded (in lexicographic order) along the boustrophedon indexing of the processors, we introduce an **arg field** to the colors of S_ι, which contains a tuple of the form

$$(i, j) \in \mathbb{N}^2.$$

It will force the input and output bits (appearing in the **processor fields**) of adjacent tiles from the input and output facets to follow their respective boustrophedon indexing.

> *Details.* For any tile t, the **arg fields** of t are all blank if the **compzone field** of $f_\blacklozenge(t)$ is blank. Otherwise, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ refer to the **compzone field** of $f_\blacklozenge(t)$, and denote $f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$ the input facet of $\tilde{\mathfrak{c}}$.
>
> If $\boldsymbol{i} \notin f_h^-(\tilde{\mathfrak{c}}) \cup f_h^+(\tilde{\mathfrak{c}})$ does not appear along the input or the output facet of $\tilde{\mathfrak{c}}$, then all the **arg fields** of t are blank.
>
> Otherwise, assume that $\boldsymbol{i} \in f_h^-(\tilde{\mathfrak{c}})$ belongs to the input facet of $\tilde{\mathfrak{c}}$:
>
> - If $\boldsymbol{i} = \mathbf{0}$, we set the **arg field** of $f_\blacklozenge(t)$ to be $(i, 0)$ for some $i \in \mathbb{N}$, or blank.
> - Otherwise, there exists some facet $f_p^\pm(t)$ pointing towards the predecessor of $\boldsymbol{i}$ along the boustrophedon indexing of the input facet $f(\tilde{\mathfrak{c}})$. Then the **arg field** of $f_p^\pm(t)$ and the **arg field** of the decoration $f_\blacklozenge(t)$ should be equal.
> - In any case, the `input` variable (from the **processor field** of $f_h^+(t)$) contains a non-blank (i, j, b) for some $(i, j) \in \mathbb{N}^2$ and bit $b \in \{0, 1\}$ if and only if the **arg field** of $f_\blacklozenge(t)$ is not blank and contains the same pair (i, j).
>
> Furthermore, if $\boldsymbol{i}$ is not the last position in the boustrophedon indexing of $f(\tilde{\mathfrak{c}})$, there exists some facet $f_s^\pm(t)$ pointing towards the successor of $\boldsymbol{i}$ in $f(\tilde{\mathfrak{c}})$. Then:
>
> - If the **arg field** of $f_\blacklozenge(t)$ is some $(i, j) \in \mathbb{N}^2$, then the **arg field** towards the successor $f_s^\pm(t)$ can either be $(i, j+1)$, or $(i', 0)$ for some $(i' > i)$, or blank;
> - If the **arg field** of $f_\blacklozenge(t)$ is blank, then the **arg field** of $f_s^\pm(t)$ is also blank.
>
> Finally, all other unmentioned facets $f_k^\pm(t)$ must have a blank **arg field**.
>
> We proceed similarly with tiles $\boldsymbol{i} \in f_h^+(\tilde{\mathfrak{c}})$ in the output facet of $\tilde{\mathfrak{c}}$, but synchronize with the `output` variable from the **processor field** of the facet $f_h^-(t)$. □

We then claim that `input` variables on the input facet are folded as in Lemma 3.8:

Claim 5. *In any σ-macro-tile $\tilde{t}$, let $\mathfrak{r} = f_h^-(\tilde{\mathfrak{c}})$ be the input facet of the computation zone $\tilde{\mathfrak{c}}$. Then the* `input` *variables of the **processor fields** of $f_h^+(\tilde{t}_{\boldsymbol{i}})$ for $\boldsymbol{i} \in \mathfrak{r}$ form the folded pattern $\mathrm{fold}_{\mathfrak{r}}(\mathtt{I}) \in (\mathbb{N}^2 \times \{0,1\})^{\mathfrak{r}}$ of some input array $\mathtt{I} \in \{0,1\}^{**}$.*
A similar claim holds for the `output` *variables on the output facet of $\tilde{t}$.*

> *Numerics.* Let $\tilde{t}$ be a σ-macro-tile of computation zone $\tilde{\mathfrak{c}}$, and let $\mathtt{I} \in \{0,1\}^{**}$ be the folded input array appearing on the input facet of $\tilde{t}$. Then for any tile t appearing in $\tilde{t}$, the bit length of:
>
> - The **processor fields** of t are bounded by $\mathcal{O}(W_e(\mathtt{I}) + \log\|\tilde{\mathfrak{c}}\|) = \mathcal{O}(W_e(\mathtt{I})) + \mathcal{O}(\tilde{\mu}_\iota^\delta)$;
> - The **arg fields** of t are bounded by $\log\|\tilde{\mathfrak{c}}\| = \mathcal{O}(\tilde{\mu}_\iota^\delta)$;
>
> where $W_e(\mathtt{I})$ is the maximal word length of the program $e \in \{0,1\}^*$ on the input array `I`, and the upper bounds $\mathcal{O}(\tilde{\mu}_\iota^\delta)$ are deduced from the proof of the staircase lemma. □

[21] Other cases $f(\tilde{\mathfrak{c}}) = f_h^-(\tilde{\mathfrak{c}})$ for $h \neq d$ follow from a permutation of the vectors $\boldsymbol{e}^k$.

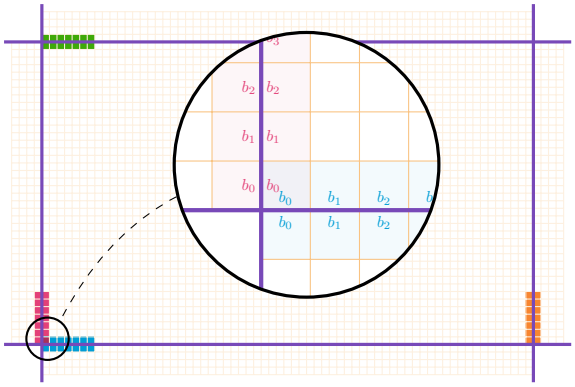


FIGURE 10. Macro-colors inside a σ-macro-tile (drawn as □).
Macro-colors appear as ■■■■■, ■■■■■, ■■■■■ and ■■■■■. Since the individual bits of the **color fields** are facing outwards, adjacent σ-macro-tiles must bear the same macro-color along their shared facet.

5.5.2. Macro-colors. In order to make σ-macro-tiles emulate the behavior of Wang tiles, we now implement in the tiles of S_ι a way for σ-macro-tiles to emulate colored facets.

Consider a σ-macro-tile $\tilde{t}$ of domain $\tilde{\mathfrak{r}}$. To emulate colored facets with $\tilde{t}$, we make the individual tiles appearing on the border of $\tilde{t}$ bear a single bit facing towards the exterior of $\tilde{t}$: along a facet $f_k^\pm(\tilde{\mathfrak{r}})$, the concatenation of those bits (folded along a path) will form a binary string $c \in \{0,1\}^*$ called a *macro-color*. Since these macro-colors are outward-facing, adjacent σ-macro-tiles will have to share the same macro-color along their adjacent facets; thus ensuring that tilings of σ-macro-tiles really simulate valid Wang tilings.

More precisely, we define a **color field** in the colors of $\mathcal{C}_\iota$, which consists of a single bit

$$b \in \{0,1\}.$$

Any tile t appearing in the interior of a σ-macro-tile $\tilde{t}$ will have blank color fields, and only the tiles t appearing on the border of σ-macro-tiles will bear an outward-facing **color field**.

> *Details.* Let t be a tile of S_ι, and let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'}$ be rectangles and positions read from the **position field** of its decoration $f_\blacklozenge(t)$. We make the **color field** of the decoration $f_\blacklozenge(t)$ always blank.
> For any direction $1 \leq k \leq d$, consider the results of the operations $\boldsymbol{j}^+ = (\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} + \boldsymbol{e}^k$ and $\boldsymbol{j}^- = (\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} - \boldsymbol{e}^k$ in the partial odometer defined by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\leq\iota'}$:
> - If $\boldsymbol{j}^+ = (\Diamond, \dots, \Diamond)$ (*i.e.* if t appears on the border of a σ-macro-tile), then the **color field** of $f_k^+(t)$ can either contain a bit $b \in \{0,1\}$ or be blank;
> - If $\boldsymbol{j}^- = (\Diamond, \dots, \Diamond)$, the **color field** of $f_k^-(t)$ can similarly contain a bit $b \in \{0,1\}$ or be blank;
> - In any other case, the **color field** of $f_k^\pm(t)$ is blank. □

Claim 6. *In any σ-macro-tile $\tilde{t}$, the* macro-colors *of $\tilde{t}$ define a tuple $(c_d^-, \dots c_1^-, c_1^+, \dots, c_d^+)$ of binary strings. Furthermore, in any valid S_ι-tiling x of $\mathbb{Z}^d$, and for each $\boldsymbol{i} \in \mathbb{Z}^d$, the macro-colors of the σ-macro-tile $x|_{\tilde{\mathfrak{r}}_i}$ describe a tile $t^{(\boldsymbol{i})} \in T_*$ with blank decoration such that the configuration $x' : \boldsymbol{i} \in \mathbb{Z}^d \mapsto t^{(\boldsymbol{i})}$ is a valid Wang tiling (where we denote $\tilde{\mathfrak{g}}_{\iota'} = (\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$).*

5.5.3. Wirings. To ensure that σ-macro-tiles actually emulate the tiles of $S_{\iota'}$ (instead of arbitrary tiles from T_*), we will use the embedded computations to restrict their tuples of possible macro-colors $(c_d^-, \dots, c_1^-, c_1^+, \dots, c_d^+)$. To do so, we will "route" the macro-colors with wires from the external facets of the σ-macro-tiles to the input facet of their computation zone.

To proceed, we add a **wire field** to the colors of S_ι, which is either of the form

$$b \in \{0,1\}$$

to draw wires outside of the computation zone transmitting bits in a straight line; or, inspired by the (possibly crossing) wires from Section 4.3, of the form

$$(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \mathbb{N})^* \quad \text{or} \quad (w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$$

to carry entries $(i, j, b) \in \mathbb{N}^2 \times \{0,1\}$ from the borders of the computation zone to the corresponding argument of its input facet, using the routing lemma (Lemma 4.4).

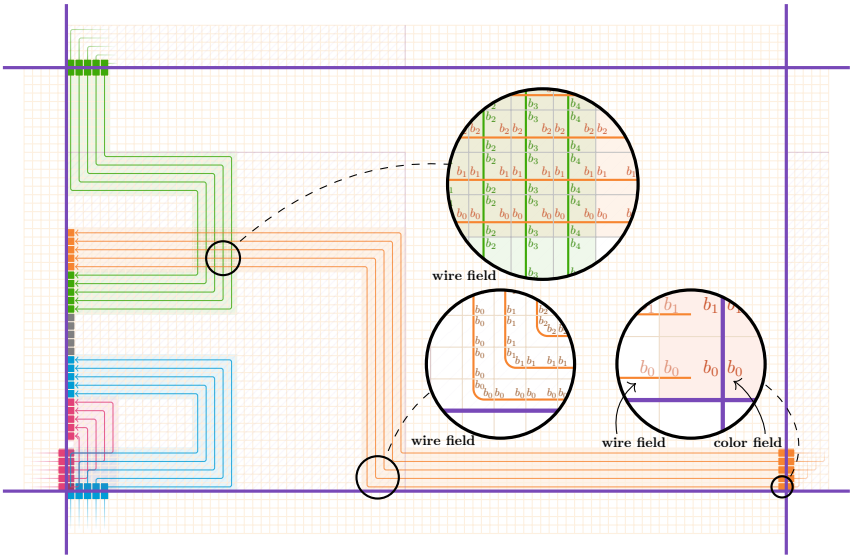


FIGURE 11. Macro-colors and wires inside a σ-macro-tile (drawn as □) and its computation zone (drawn with ▨).
Macro-colors appear as ■■■■■, ■■■■■, ■■■■■ and ■■■■■. Outside of the computation zone, wires carry the individual bits of macro-colors along straight lines. Inside the computation zones, the routing lemma wires these bits (indices (i, j) are not drawn) towards the input facet (on the left). On the input facet, the gray argument ■■■■■ corresponds to the macro-decoration (not wired) of the σ-macro-tile. For readability purposes, this figure draws a very neat wiring inside the computation zone, even though the wiring tileset may not always do so.

Details. For any tile t, let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'}$ be read from the **position field** of $f_{\blacklozenge}(t)$.

First, let $\boldsymbol{j}^+ = (\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} + \boldsymbol{e}^k$ and $\boldsymbol{j}^- = (\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} - \boldsymbol{e}^k$ denote the results of the partial odometer operations defined by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\leq\iota'}$. Then if $\boldsymbol{j}^+ = (\Diamond, \dots, \Diamond)$ (resp. $\boldsymbol{j}^+ = (\Diamond, \dots, \Diamond)$), then the **wire field** of $f_k^+(t)$ (resp. $f_k^-(t)$) must be blank. This ensures that the **wire fields** between adjacent σ-macro-tiles are completely isolated from one another.

Let us now consider whether t appears inside a computation zone or not. On the one hand, if the **compzone field** of $f_{\blacklozenge}(t)$ is blank:

- The **wire field** of $f_{\blacklozenge}(t)$ must be blank; and the **wire field** of any $f_k^\pm(t)$ can either be blank or a single bit $b \in \{0, 1\}$;
- The **wire field** of a facet $f_k^\pm(t)$ is a bit $b \in \{0, 1\}$ if and only if either the **wire field** of its opposite facet $f_k^\mp(t)$ contains the same bit $b \in \{0, 1\}$ (*i.e.* inside a σ-macro-tile), or if the **color field** of said $f_k^\mp(t)$ contains the same bit $b \in \{0, 1\}$ (*i.e.* on the border of a σ-macro-tile).

Otherwise, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ denote the **compzone field** of $f_{\blacklozenge}(t)$. Then the **wire fields** of t will implement the *routing tileset* from Section 4.3:

- The **wire field** of $f_{\blacklozenge}(t)$ contains a list of wires $(w_i^{\blacklozenge})_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$ of length $|I| \leq K_{\text{cross}}$, where $K_{\text{cross}} \in \mathbb{N}$ bounds the crossings required in the routing lemma;
- If $\boldsymbol{i} \in f_k^\pm(\tilde{\mathfrak{c}})$, then the **wire field** of $f_k^\pm(t)$ is either blank or a single bit $b \in \{0, 1\}$. Otherwise, $f_k^\pm(t)$ is a list of wires $(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \mathbb{N})^*$ of length $|I| \leq K_{\text{cross}}$;
- The **wire fields** of $f_{\blacklozenge}(t)$ and $f_k^\pm(t)$ must form a valid tile in the routing tileset $T_{\rightleftharpoons}(\mathbb{N}^2 \times \{0, 1\})$ (preservation of wires between facets…); we also bound the length of any wire $n \in \mathbb{N}$ by $\|\tilde{\mathfrak{c}}\|$;
- (Starts of wires) An entry $((i, j, b), \texttt{start})$ appears in the **wire field** of $f_{\blacklozenge}(t)$ if and only if $\boldsymbol{i}$ belongs to a facet $f_k^\pm(\tilde{\mathfrak{c}})$ for some $1 \leq k \leq d$, and the **color field** or the **wire field** of the corresponding facet $f_k^\pm(t)$ consists of a single bit $b \in \{0, 1\}$. In which case, we set $i = 4 + d \pm k$ and j to the index of the position $\boldsymbol{i}$ in the boustrophedon indexing of the facet $f_k^\pm(\tilde{\mathfrak{c}})$;
- (Ends of wires) An entry $((i, j, b), \texttt{end})$ appears in the **wire field** of $f_{\blacklozenge}(t)$ if and only if $\boldsymbol{i} \in f(\tilde{\mathfrak{c}}) = f_h^-(\tilde{\mathfrak{c}})$ belongs to the input facet of the computation zone, and the variable `input` in the **processor field** of the facet $f_h^+(t)$ contains the same tuple (i, j, b).

Finally, in order to prevent bits of macro-colors to traverse a σ-macro-tile in straight lines without ever crossing its computation zone, we enforce the following condition:

- If the **color field** of $f_k^-(t)$ is non-blank, then so is the **compzone field** of $f_{\blacklozenge}(t)$;

thus ensuring that macro-colors on negative facets must already appear inside the computation zone. □

We then claim that the macro-colors of a σ-macro-tile occupy, on its input facet, the arguments `I[4]` to `I[2d + 4]` of the simulated RAM program e:

Claim 7. *Let $\tilde{t}$ be a σ-macro-tile of macro-colors $(c_d^-, \dots c_1^-, c_1^+, \dots, c_d^+)$, and let $\mathtt{I} \in \{0,1\}^{**}$ be the input array that appears on the input facet of its computation zone. Then for every $1 \leq k \leq d$, we have $\mathtt{I}[d + 4 \pm k] = c_k^{\pm}$.*

> *Numerics.* Let $\tilde{t}$ be a σ-macro-tile of domain $\tilde{\mathfrak{r}}$ and computation zone $\tilde{\mathfrak{c}}$. Since the length of wires inside $\tilde{\mathfrak{c}}$ is bounded by $\|\tilde{\mathfrak{c}}\| \leq \mu_{\iota'}^d$, and that in the carried colors indices (i, j) belong to $\{4, \dots, 2d+4\} \times \{0, \dots, \|\tilde{\mathfrak{c}}\|\}$ (since they are mapped to the `input` variables of some **processor fields**), the **wire fields** in $\tilde{t}$ have bit length $\mathcal{O}(\log \mu_{\iota'}) = \operatorname{polylog} K \cdot \mathcal{O}(\tilde{\mu}_\iota^{\delta})$.
> (This also justifies why we use straight wires outside of the computation zone: otherwise, wires could be of length $\mathcal{O}(\lambda(\tilde{\mathfrak{r}}))$, which would result in **wire fields** of bit length $\operatorname{polylog} K \cdot \mathcal{O}(\tilde{\mu}_\iota^{2\delta})$.) □

5.5.4. Communication with the next level of the simulation. While the arguments `I[4]` to `I[2d+4]` of the input facets are *wired* from their macro-colors, the reader may have noticed that we have yet to mention anything about the arguments `I[0]`, ..., `I[3]`. These arguments will actually be used as a way to communicate information from the tiles of S_ι to the next level of the simulation ι', and are thus initialized differently.

The copying method. Assume that a **foo** field in the colors of $\mathcal{C}_\iota$ consists of a binary string that is constant across all the tiles $t \in S_\iota$ composing a σ-macro-tile $\tilde{t}$. We then claim that **foo** can be used to initialize some argument `I[`i`]` on the input facet of the computation zone of $\tilde{t}$.

Indeed, let $t \in S_\iota$ appearing in a σ-macro-tile $\tilde{t}$, and assume that $f_\blacklozenge(t)$ has a non-blank **compzone field** $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ and a **foo** field $u \in \{0,1\}^*$. Then we can ensure that, if t appears on the input facet of the computation zone $\tilde{\mathfrak{c}}$ (*i.e.* if $\boldsymbol{i} \in f(\tilde{\mathfrak{c}})$), then the `input` variable (i', j, b) of the corresponding **processor field** satisfies $b = u_j$ if $i = i'$.

Initializing the input facet. Using the copying method, we initialize the arguments `I[`i`]` of the input facets of σ-macro-tiles as follows:

- We initialize `I[0]` by copying the parameters (K, K_{word}) from the global function $\varphi_{F(e)}$;
- We initialize `I[1]` by copying the **p-level field** $\iota' \in \mathbb{N}$ of the decorations $f_\blacklozenge(t)$;
- We initialize `I[2]` by copying $(\mu_\ell, \lambda_\ell)_{1 \leq \ell < \kappa'}$, the concatenation of $(\mu_\ell, \lambda_\ell)_{1 \leq \ell < \kappa}$ from the function $\varphi_{F(e)}$ and the family $(\mu_\ell, \lambda_\ell)_{\kappa \leq \ell < \kappa'}$ from the **bound field** of $f_\blacklozenge(t)$;
- We initialize `I[3]` by copying $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota' < \ell < \kappa'}$, as extracted from the family $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'}$ from the **position field** of $f_\blacklozenge(t)$.

Reading the output facet. Instead of writing inside the input facet of the computation zone, it is also possible to use the copying method to read the values returned by the computation of the embedded processor array on the output facet. For example:

- We copy the argument `O[0]` from the output facet, which consists of a single bit $b \in \{0,1\}$, into the **p-chessboard field** of $f_\blacklozenge(t)$.

We will similarly use the other arguments `O[1]`, `O[2]` and `O[3]` later in the proof.

§ 5.6. τ-macro-tiles (of level $\kappa \leq \ell < \kappa'$)

Now that the next level of simulation is implemented in the σ-macro-tiles, we turn towards the embedding of the τ-substitution steps

$$x^{(\kappa')} \xrightarrow[\mathfrak{g}_{\kappa'}]{\tau} x^{(\kappa'-1)} \xrightarrow{\tau} \dots \xrightarrow{\tau} x^{(\kappa+1)} \xrightarrow[\mathfrak{g}_{\kappa+1}]{\tau} x^{(\kappa)}$$

for $\mathfrak{g}_{\kappa+1}, \dots, \mathfrak{g}_{\kappa'-1}$ the grids determined by the **position fields** of a valid tiling, and $\mathfrak{g}_{\kappa'}$ the grid determined by its **p-chessboard fields**.

In the rest of this proof, we will call τ-macro-tiles of level ℓ the macro-tiles of level $\kappa \leq \ell < \kappa'$.[22] In particular, τ-macro-tiles respect the nested hierarchy already mentioned in Section 5.4: we kindly refer the reader to Figure 8 again.

[22]The notions of σ-macro-tiles and τ-macro-tiles of level ι' actually coincide. When referring to a macro-tile of level ι' as a τ-macro-tile (instead of a σ-macro-tile), we will implicitly refer to the τ-variants of its fields.

The rest of this section will draw the computational embeddings associated with the substitution steps $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$ for $\kappa \leq \ell < \kappa'$ inside the τ-macro-tiles of level ℓ, using the nested structure of these macro-tiles. In terms of configurations, a valid tiling of S_ι will thus draw multiple levels of substitutions superimposed on top of one another.

Before we begin, recall that this substitution τ is obtained by a log-RAM program $\langle\tau\rangle \in \{0,1\}^*$ defining a function:

$$\varphi_{\langle\tau\rangle}\big(\mathfrak{r}, \boldsymbol{i}, a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+\big) \rightrightarrows \big(b_d^-, \dots, b_1^-, b_0, b_1^+, \dots, b_d^+\big)$$

where $\mathfrak{r} \in \mathfrak{R}_0$ is a rectangle, $\boldsymbol{i} \in \mathfrak{r}$ a position in $\mathfrak{r}$, and the $a_k^\pm$'s and the $b_k^\pm$'s respectively form Wang tiles $a, b \in T_*$. The code $\langle\tau\rangle$ thus implements the local function associated with the substitution:

$$\tau(a) = \bigcup_{\mathfrak{r}\in\mathfrak{R}_0} \Big\{ w \in T_*^{\mathfrak{r}} : \forall \boldsymbol{i} \in \mathfrak{r},\ w_{\boldsymbol{i}} \in \varphi_{\langle\tau\rangle}\big(\mathfrak{r}, \boldsymbol{i}, a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+\big) \Big\}.$$

5.6.1. τ-computation zones. These new τ-macro-tiles are built in the same way macro-tiles were in the previous part of the proof. More precisely, for every ℓ in the interval $\kappa \leq \ell < \kappa'$, we introduce a **$\boldsymbol{\tau}$-compzone field**, a **$\boldsymbol{\tau}$-processor field**, and a **$\boldsymbol{\tau}$-arg field of level $\boldsymbol{\ell}$** (in other words, there are $\kappa' - \kappa$ new **$\boldsymbol{\tau}$-processor fields** (resp. etc…), one per such level).

In a τ-macro-tile of level ℓ and domain $\tilde{\mathfrak{r}} \in \mathfrak{R}_0$, the **$\boldsymbol{\tau}$-compzone fields** of the form $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ (for $\tilde{\mathfrak{c}} \in \mathfrak{R}_0$ and $\boldsymbol{i} \in \tilde{\mathfrak{c}}$) delimit the computation zone $\tilde{\mathfrak{c}} = \mathrm{Cut}_N(\tilde{\mathfrak{r}})$ with bounds $N = \prod_{i=\iota+1}^{\ell} \mu_i = \tilde{\mu}_\ell / \tilde{\mu}_\iota$. In said computation zone, the **$\boldsymbol{\tau}$-processor fields** collectively draw the space-time diagram of a processor array simulating the substitution $\langle\tau\rangle$; and the **$\boldsymbol{\tau}$-arg fields** ensure that arguments of the input and output array are correctly folded along the Boustrophedon indexing of the input and output facets.

> *Details.* The adaptation of the previously defined fields into these new τ-counterparts is straightforward; for clarity, let us highlight a few important modifications on the **$\boldsymbol{\tau}$-processor fields**. For any tile t, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ denote the **$\boldsymbol{\tau}$-compzone field** of level ℓ of $f_\blacklozenge(t)$. Assuming that $\boldsymbol{i}$ is part of the input facet $f(\tilde{\mathfrak{c}}) = f_h^-(\tilde{\mathfrak{c}})$, the variables appearing in the **$\boldsymbol{\tau}$-processor field** of level ℓ of $f_h^+(t)$ satisfy:
>
> - The `program` variable must be set to $\texttt{program} = \langle\tau\rangle$, where $\langle\tau\rangle \in \{0,1\}^*$ is the code of the local program defining the substitution τ (*c.f.* Definition 5.1);
> - The `timeout` variable must be set to $\texttt{timeout} = K_{\langle\tau\rangle}^{(t)} \cdot \tilde{\mu}_\ell^\alpha$ for the constant $K_{\langle\tau\rangle}^{(t)} \in \mathbb{N}$ fixed in the housekeeping paragraph;
> - The `word` variable must be set to $\texttt{word} = \texttt{word} = 2K_{\mathrm{word}} \cdot \log \mu(\tilde{\mathfrak{c}})$, where K_{word} is the constant appearing in the parameters (K, K_{word}) of the global function $\varphi_{F(e)}$;
> - The `input` variable must be set to $\texttt{input} = (i, j, b)$ for any bit $b \in \{0,1\}$ and any index $(i,j) \in \{0, \dots, 2d+2\} \times \{0, \dots, K_{\langle\tau\rangle}^{(t)} \cdot \tilde{\mu}_\ell^\alpha\}$. □

> *Numerics.* Let $\tilde{t}$ be a τ-macro-tile of level ℓ. Using similar arguments than before, the **$\boldsymbol{\tau}$-compzone fields of level $\boldsymbol{\ell}$** and the **$\boldsymbol{\tau}$-processor fields of level $\boldsymbol{\ell}$** in $\tilde{t}$ have bit length $\mathrm{polylog}\, K \cdot \mathcal{O}(\tilde{\mu}_\iota^\delta)$.
> In particular, in a valid tile t, all the **$\boldsymbol{\tau}$-compzone fields** and the **$\boldsymbol{\tau}$-processor fields** of all levels $\kappa \leq \ell < \kappa'$ are of collective bit length $\mathrm{polylog}\, K \cdot \mathcal{O}(\tilde{\mu}_\iota^{2\delta})$. □

5.6.2. τ-colors and τ-wirings. Since τ substitutes Wang tilings, we create and enforce the local validity of the substituted Wang tilings by introducing, for every ℓ in the interval $\kappa \leq \ell < \kappa'$, a **$\boldsymbol{\tau}$-color field** and **$\boldsymbol{\tau}$-wire field of level $\boldsymbol{\ell}$**.

In a configuration, **$\boldsymbol{\tau}$-color fields** of adjacent τ-macro-tiles of level ℓ define a Wang tiling $x^{(\ell)} \in (T_*)^{\mathbb{Z}^d}$; and, inside each such macro-tile, **$\boldsymbol{\tau}$-wire fields** "route" the corresponding τ-macro-colors to the *output facet* of the computation zone (which contains the corresponding pixel of the substitution step $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$).

> *Details.* The adaptation of the previously defined fields into these new τ-counterparts is straightforward. For clarity, let us highlight one important modification on the **$\boldsymbol{\tau}$-wire fields**. For any tile t, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ denote the **$\boldsymbol{\tau}$-compzone field** of level ℓ of $f_\blacklozenge(t)$; and denote by $f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$ the *input facet* of $\tilde{\mathfrak{c}}$. Then the **$\boldsymbol{\tau}$-wire field of level $\boldsymbol{\ell}$** of $f_\blacklozenge(t)$ satisfies:
>
> - (Start of wires) An entry $((i,j,b), \texttt{start})$ appears in the **$\boldsymbol{\tau}$-wire field of level $\boldsymbol{\ell}$** of $f_\blacklozenge(t)$ if and only if $\boldsymbol{i} \in f_k^\pm(\tilde{\mathfrak{c}})$ for some facet $1 \leq k \leq d$ of the computation zone; and the **$\boldsymbol{\tau}$-color field** or the **$\boldsymbol{\tau}$-wire field of level $\boldsymbol{\ell}$** of the corresponding facet $f_k^\pm(t)$ consists of a single bit $b \in \{0,1\}$.

In which case, we set $i = d \pm k$ and j to the index of the position $\boldsymbol{i}$ in the boustrophedon indexing of the facet $f_k^{\pm}(\tilde{\mathfrak{c}})$;
- (End of wires) An entry $((i,j,b), \texttt{end})$ appears in the **τ-wire field of level ℓ** of $f_{\blacklozenge}(t)$ if and only if $\boldsymbol{i}$ belongs to the *output facet* $f_h^+(\tilde{\mathfrak{c}})$ of the computation zone; and the variable `output` in the **τ-processor field of level ℓ** of the facet $f_h^-(t)$ contains the same tuple (i,j,b). □

Numerics. Let $\tilde{t}$ be a τ-macro-tile of level ℓ and computation zone $\tilde{\mathfrak{c}}$. By the routing lemma, wires in the computation zone $\tilde{\mathfrak{c}}$ have length bounded by $\mathcal{O}(\lambda(\tilde{\mathfrak{c}}))$. Thus, the **$\tau$-wire fields of level ℓ** in $\tilde{t}$ have bit length $\operatorname{polylog} K \cdot \mathcal{O}(\tilde{\mu}_\iota^\delta)$. □

5.6.3. τ-consistency. For $\kappa \leq \ell < \kappa'$, an embedded substitution step $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$ must respect the structure of the grid $\mathfrak{g}_{\ell+1}$ (as determined by the **position fields** if $\ell < \kappa' - 1$, and the **p-chess fields** if $\ell = \kappa' - 1$). In particular, all local computations $\varphi_{\langle\tau\rangle}([\mathfrak{r}], \boldsymbol{i}, a)$ belonging to the same rectangle $\mathfrak{r}$ in $\mathfrak{g}_{\ell+1}$ must be initialized with the same tile $a \in T_*$.

For every $\kappa \leq \ell < \kappa'$, we thus add a **$\tau$-consistency field of level ℓ**. Based upon the already defined **wire field**, it ensures that τ-macro-tiles of level ℓ appearing in the same rectangle of the grid $\tilde{\mathfrak{g}}_{\ell+1}$ contain the same Wang tile $(a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+) \in T_*$ on their respective *input facets*. More precisely, these **τ-consistency fields of level ℓ** draw wires between τ-macro-tiles of level ℓ:

- Outside the computation zone of a τ-macro-tile, bits are carried in a straight line;
- Inside the computation zone of a τ-macro-tile, the routing lemma (Lemma 4.4) carries bits from the border of the computation zone to its input facet;
- Finally, wires are drawn between two adjacent τ-macro-tiles of level ℓ if and only if they belong to the same τ-macro-tile of level $\ell + 1$.

Details. Similar to **wire fields**, a **τ-consistency field of level ℓ** is either a single input bit

$$(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$$

to transmit bits in straight lines between the computation zones of adjacent macro-tiles; or of the form

$$(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \mathbb{N})^* \quad ; \text{ or } \quad (w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$$

to carry entries $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$ from the borders of the computation zone to its input facet. The adaptation from the already-defined **wire fields** is straightforward, but we highlight the important modifications for clarity.

Thus, let t be a tile and let $\boldsymbol{j}_k^+ = (\boldsymbol{i}_{\ell'})_{\iota<\ell'\leq\ell} + \boldsymbol{e}^k$ and $\boldsymbol{j}_k^- = (\boldsymbol{i}_\ell)_{\iota<\ell'\leq\ell} - \boldsymbol{e}^k$ denote the results of the partial odometer operations defined by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell'\leq\ell}$. If $\boldsymbol{j}_k^- = (\Diamond, \dots, \Diamond)$ (resp. $\boldsymbol{j}_k^+ = (\Diamond, \dots, \Diamond)$), *i.e.* if t appears on the border of a τ-macro-tile of level ℓ: as opposed to the previously defined **wire fields**, the **τ-consistency field** of $f_k^-(t)$ (resp. $f_k^+(t)$) is not always blank. More precisely:

- If $\boldsymbol{j}_k^+ = (\Diamond, \dots, \Diamond)$ and $\ell < \kappa' - 1$: the **τ-consistency field of level ℓ** of $f_k^+(t)$ can contain a non-blank entry $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$ if and only if $\boldsymbol{i}_\ell + \boldsymbol{e}^k \in \mathfrak{r}_{\ell+1}$;
- If $\boldsymbol{j}_k^+ = (\Diamond, \dots, \Diamond)$ and $\ell = \kappa' - 1$: the **τ-consistency field of level ℓ** of $f_k^+(t)$ can contain a non-blank entry $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$ if and only if the **p-chessboard field** of $f_{\blacklozenge}(t)$ and $f_k^+(t)$ are equal;
- If $\boldsymbol{j}_k^- = (\Diamond, \dots, \Diamond)$: the **$\tau$-consistency field of level ℓ** of $f_k^-(t)$ can contain a non-blank entry $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$ if and only if $f_{\blacklozenge}(t)$ has a non-blank **τ-compzone field of level ℓ**;

Otherwise, the **τ-consistency fields of level ℓ** of t behave similarly to the previously defined **wire fields**. We mostly manage the start and ends of wires inside the computation zones differently: if the **τ-compzone field of level ℓ** of $f_{\blacklozenge}(t)$ is some non-blank $(\tilde{\mathfrak{c}}, \boldsymbol{i})$, then

- (Start of wires) An entry $((d \pm k, i, j, b), \texttt{start})$ appears in the **τ-consistency field of level ℓ** of $f_{\blacklozenge}(t)$ if and only if $\boldsymbol{i} \in f_k^{\pm}(\tilde{\mathfrak{c}})$ is on some facet $1 \leq k \leq d$ of the computation zone, and the **τ-consistency field of level ℓ** of the corresponding $f_k^{\pm}(t)$ contains $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$;
- (End of wires) The **τ-consistency field of level ℓ** of $f_{\blacklozenge}(t)$ contains $2d$ entries of the form $((d \pm k, i, j, b), \texttt{end})$ (for $1 \leq k \leq d$ and $\pm \in \{+,-\}$) if and only if $\boldsymbol{i} \in f(\tilde{\mathfrak{c}}) = f_h^-(\tilde{\mathfrak{c}})$ appears on the input facet of the computation zone, and the variable `input` of the **τ-processor field of level ℓ** of the facet $f_h^+(t)$ contains the same (i,j,b) for $2 \leq i \leq 2d+2$; otherwise, it contains no entry of the form $((\cdot,\cdot,\cdot), \texttt{end})$.

By applying the routing lemma $2d$ times (one per value $d \pm k$ for $1 \leq k \leq d$), this defines a valid routing between the input facet of the computation zone and each facet $f_k^{\pm}(\tilde{\mathfrak{c}})$. □

Numerics. Similarly to the **τ-wire fields**, the **τ-wire fields of level ℓ** (for $\kappa \leq \ell < \kappa'$) in a valid tile $\tilde{t}$ have bit length $\mathrm{polylog}\, K \cdot \mathcal{O}(\tilde{\mu}_\iota^\delta)$. □

5.6.4. τ-pipes. We now initialize the input words `I[0]`, `I[1]` and `I[2]`, ..., `I[2d+2]` on the input facet of τ-macro-tiles of level ℓ. In a valid tiling, they should respectively encode a rectangle $[\mathfrak{r}_{\ell+1}]$ from the grid $\mathfrak{r}_{\ell+1} \in \mathfrak{g}_{\ell+1}$, a position $\boldsymbol{i}_{\ell+1} \in [\mathfrak{r}_{\ell+1}]$, and a tile $a_{\ell+1} = (a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+) \in T_*$. To enforce the consistency of this information across successive substitution steps $x^{(\ell+2)} \xrightarrow{\tau} x^{(\ell+1)} \xrightarrow{\tau} x^{(\ell)}$, the output facet of a τ-macro-tile of level $\ell+1$ and the input facets of its τ-macro-tiles of level ℓ need to be synchronized.

Before doing so, notice the following fact from the structure of nested τ-macro-tiles:

Claim 8. *Let $\ell' \leq \ell$, and consider a τ-macro-tile $\tilde{t}^{(\ell)}$ of level ℓ and domain $\tilde{\mathfrak{r}}$. Then:*

- *There exists a unique τ-macro-tile $\tilde{t}^{(\ell')}$ of level ℓ' in $\tilde{t}^{(\ell)}$ whose domain $\tilde{\mathfrak{r}}' \subseteq \tilde{\mathfrak{r}}$ contains the corner position $\mathbf{0} \in \tilde{\mathfrak{r}}$;*
- *The computation zone $\tilde{\mathfrak{c}}'$ of $\tilde{t}^{(\ell')}$ has a facet $f_k^-(\tilde{\mathfrak{c}}')$ that is included in the input facet $f(\tilde{\mathfrak{c}})$ of the computation zone $\tilde{\mathfrak{c}}$ of $\tilde{t}^{(\ell)}$. (Note that $f_k^-(\tilde{\mathfrak{c}}')$ has no reason to actually be the input facet of the computation zone $\tilde{\mathfrak{c}}'$.)*

Informally, this claim ensures that there exists a facet in $\tilde{t}^{(\ell')}$ through which the routing lemma will be able to transmit information between $\tilde{t}^{(\ell')}$ and $\tilde{t}^{(\ell)}$. By analogy with the chaining of standard input and output streams in Unix systems, we will call such information transmission across levels a *pipe*.

Substitution shapes. The first two arguments `I[0]` and `I[1]` of $\varphi_{\langle\tau\rangle}$, which appear on the input facet of every τ-macro-tile of level ℓ, should respectively be a rectangle $[\mathfrak{r}_{\ell+1}]$ from the grid $\mathfrak{g}_{\ell+1}$ and a position $\boldsymbol{i}_{\ell+1} \in [\mathfrak{r}_{\ell+1}]$.

If $\ell < \kappa' - 1$, these arguments `I[0]` and `I[1]` can be directly initialized from the entry $(\mathfrak{r}_{\ell+1}, \boldsymbol{i}_{\ell+1})$ in the **position field** of every tile t using the *copying method*.

Claim 9. *For any τ-macro-tile $\tilde{t}$ of level ℓ for $\ell < \kappa' - 1$, let $\mathtt{I} \in \{0,1\}^{**}$ be the input array appearing on the input facet of $\tilde{t}$, and let $(\mathfrak{r}_{\ell+1}, \boldsymbol{i}_{\ell+1})$ be the positions of level $\ell+1$ in any **position field** in $\tilde{t}$. Then `I[0]` $= \mathfrak{r}_{\ell+1}$ and `I[1]` $= \boldsymbol{i}_{\ell+1}$.*

If $\ell = \kappa' - 1$, the arguments `I[0]` and `I[1]` correspond to a pair $(\mathfrak{r}_{\kappa'}, \boldsymbol{i}_{\kappa'})$ as deduced from the grid $\mathfrak{g}_{\kappa'}$. While a τ-macro-tile $\tilde{t}^{(\kappa'-1)}$ of level $\kappa' - 1$ does not contain this information in the **position fields** of its tiles, it actually appears in the argument `O[1]` and `O[2]` from the output facet of every macro-tile $\tilde{t}$ (*c.f.* Section 5.5.4).

To transmit this information, we define a **τ-pos-pipe field** containing a pair (W_1, W_2), which informally correspond to two applications of the routing lemma transporting a color $(i, j, b) \in \mathbb{N}^2 \times \{0,1\}$ for $b =$ `O[i][j]`. Since by Claim 8, the computation zones of some macro-tile $\tilde{t}$ and of $\tilde{t}^{(\kappa'-1)}$ intersect along a common facet, we apply the routing lemma twice:

- Inside the σ-macro-tile $\tilde{t}$, we route the information from the output facet of the computation zone towards this common facet using the entries W_1 from **τ-pos-pipe fields**;
- Inside the τ-macro-tile $\tilde{t}^{(\kappa'-1)}$, we reorganize this information towards the corresponding arguments of the input array using the entries W_2 from **τ-pos-pipe fields**.

Details. Let $t \in S_\iota$ be a tile and let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'}$ be the **position field** of $f_\bullet(t)$. The **τ-pos-pipe fields** of t consist of tuples of the form (W_1, W_2), where W_1 and W_2 are lists $(w_i)_{i \in I}$ of the form:

$$(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \mathbb{N})^* \quad ; \text{ or } \quad (w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0,1\}) \times \{\mathtt{start}, \mathtt{end}, \mathtt{cross}\})^*$$

to carry entries $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0,1\}$ using the routing lemma.

Assume that $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ and $(\tilde{\mathfrak{c}}', \boldsymbol{i}')$ are the non-blank **compzone field** and **τ-compzone field of level $\kappa' - 1$** of t. Furthermore, assume that $(\boldsymbol{i}_\ell)_{\iota' < \ell < \kappa'} = (\mathbf{0}, \dots, \mathbf{0})$ (*i.e.* any σ-macro-tile containing t covers the position $\mathbf{0}$ in $\tilde{\mathfrak{c}}'$). By Claim 8, there exists a facet $f_k^-(\tilde{\mathfrak{c}})$ that is included in the input facet $f(\tilde{\mathfrak{c}}')$. Then the list of wires W_1 in the **τ-pos-pipe fields** of t is used to carry entries $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0,1\}$ from the output facet of $\tilde{\mathfrak{c}}$ towards the facet $f_k^-(\tilde{\mathfrak{c}})$. The adaptation from the already-defined **(τ)-wire fields** is straightforward, but we highlight the important modifications for clarity:

- (Border condition) If $\boldsymbol{i} \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{c}}$ (in the **compzone field** of t), then the list W_1 of the **τ-pos-pipe field** of $f_k^\pm(t)$ is empty;

- (Start of wires) Let $(W_1, \cdot)$ be the $\boldsymbol{\tau}$**-pos-pipe field** of $f_{\blacklozenge}(t)$. If the position $\boldsymbol{i}$ belongs to the output facet $f_h^+(\tilde{\mathfrak{c}})$ of $\tilde{\mathfrak{c}}$ (*i.e.* $f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$), then W_1 contains an entry $((i,j,b), \texttt{start})$ for some $b \in \{0,1\}$ if and only if the `output` variable of the **processor field** of $f_h^-(t)$ encodes the same tuple (i,j,b);
- (End of wires) Let $(W_1, \cdot)$ be the $\boldsymbol{\tau}$**-pos-pipe field** of $f_{\blacklozenge}(t)$. If the position $\boldsymbol{i}$ belongs to the common facet $f_k^-(\tilde{\mathfrak{c}})$, then W_1 may contain a single entry $((i,j,b), \texttt{end})$ for some $i \in \{1,2\}$, $j \in \{0, \dots, \|\tilde{\mathfrak{c}}\|\}$ and $b \in \{0,1\}$.

In any other case, the lists W_1 from the $\boldsymbol{\tau}$**-pos-pipe fields** of t are assumed to be empty.

Assume that $(\tilde{\mathfrak{c}}', \boldsymbol{i}')$ is a non-blank $\boldsymbol{\tau}$**-compzone field of level** $\boldsymbol{\kappa' - 1}$ of t. Then the list of wires W_2 in the $\boldsymbol{\tau}$**-pos-pipe fields** of t is used to carry entries $(i,j,b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0,1\}$ from the input facet of $\tilde{\mathfrak{c}}'$ to their correct positions in input array. We again highlight the important modifications from other wirings:

- (Border condition) If $\boldsymbol{i}' \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{c}}'$ (in the $\boldsymbol{\tau}$**-compzone field of level** $\boldsymbol{\kappa' - 1}$ of t), then the list W_2 of the $\boldsymbol{\tau}$**-pos-pipe field** of $f_k^{\pm}(t)$ is empty;
- (Start of wires) Let (W_1, W_2) be the $\boldsymbol{\tau}$**-pos-pipe field** of $f_{\blacklozenge}(t)$. Then W_2 contains an entry $((i,j,b), \texttt{start})$ if and only if W_1 contains the same entry $((i,j,b), \texttt{end})$;
- (End of wires) Let $(\cdot, W_2)$ be the $\boldsymbol{\tau}$**-pos-pipe field** of $f_{\blacklozenge}(t)$. If the position $\boldsymbol{i}'$ belongs to the input facet $f_h^-(\tilde{\mathfrak{c}}') = f(\tilde{\mathfrak{c}}')$, then W_2 contains an entry $((i,j,b), \texttt{end})$ for some $b \in \{0,1\}$ if and only if the `input` variable of the **processor field** of $f_h^+(t)$ encodes the tuple $(i-1, j, b)$. □

Claim 10. *For any τ-macro-tile $\tilde{t}^{(\kappa'-1)}$ of level $\kappa' - 1$ and domain $\tilde{\mathfrak{r}}$, let $\tilde{t}^{(\iota')}$ denote the σ-macro-tile in $\tilde{t}^{(\kappa'-1)}$ covering the position $\mathbf{0} \in \tilde{\mathfrak{r}}$. If $\mathfrak{r} \in \mathfrak{R}_0$ and $\boldsymbol{i} \in \mathfrak{r}$ are the respective arguments* `O[1]` *and* `O[2]` *from the output facet of $\tilde{t}^{(\iota')}$, then the arguments* `I[0]` *and* `I[1]` *on the input facet of $\tilde{t}^{(\kappa'-1)}$ satisfy* `I[0]` $= \mathfrak{r}$ *and* `I[1]` $= \boldsymbol{i}$.

Substitution symbols. The last $2d+1$ arguments `I[3]` to `I[2d+3]` of $\varphi_{\langle\tau\rangle}$ on the input facet of every τ-macro-tile of level ℓ should be initialized with the symbol $a_{\ell+1} = (a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+) \in T_*$ that appears on the output facet of the τ-macro-tile of the next level.

To transmit this information, we define a $\boldsymbol{\tau}$**-sym-pipe field of level** $\boldsymbol{\ell}$ containing a pair (W_1, W_2), which informally correspond to two applications of the routing lemma transporting a color $(i,j,b) \in \mathbb{N}^2 \times \{0,1\}$ for $b =$ `I[`i`][`j`]`. Since the $\boldsymbol{\tau}$**-consistency field of level** $\boldsymbol{\ell}$ ensures that all macro-tiles $\tilde{t}^{(\ell)}$ forming $\tilde{t}^{(\ell+1)}$ share the same input symbol, it is actually enough to synchronize the symbols $(a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+)$ from $\tilde{t}^{(\ell+1)}$ with the τ-macro-tile $\tilde{t}^{(\ell)}$ covering the corner position $\mathbf{0} \in \tilde{\mathfrak{r}}_{\ell+1}$.

This is actually achieved in a similar manner than for the $\boldsymbol{\tau}$**-pos-pipe field**.

Claim 11. *For any τ-macro-tile $\tilde{t}^{(\ell+1)}$ of level $\ell+1$ for $\ell < \kappa' - 1$, let $\tilde{t}^{(\ell)}$ denote any of the τ-macro-tile of level ℓ in $\tilde{t}^{(\ell+1)}$. If $(a_0, \dots, a_{2d}) \in T_*$ denotes the arguments* `I[3]`$, \dots,$ `I[`$2d+3$`]` *on the input facet of $\tilde{t}^{(\ell)}$, then the arguments* `O[0]`$, \dots,$ `O[`$2d$`]` *on the output facet of $\tilde{t}^{(\ell)}$ satisfy* `O[`i`]` $= a_i$ *for every $i \in \{0, \dots, 2d\}$.*

If $\ell = \kappa' - 1$, there is actually no τ-macro-tile of level κ' to synchronize with. Indeed, the result of the next substitution step $x^{(\kappa'+1)} \xrightarrow{\tau} x^{(\kappa')}$ is actually embedded in the next level of simulation, *i.e.* in tilings of $S_{\iota'}$ (and not S_ι). The symbol $a_{\ell+1} = (a_d^-, \dots, a_1^-, a_0, a_1^+, \dots, a_d^+) \in T_*$ must thus be *synchronized through another level of simulation*, which is slightly technical.

Details (pixel positions). In order to identify the shape (and, in particular, the smallest facet) of some rectangles $\tilde{\mathfrak{r}}_\ell$ in the respective grid $\tilde{\mathfrak{g}}_\ell$, we begin by defining three **pixel position fields** (respectively **of level** $\boldsymbol{\kappa - 1}$, $\boldsymbol{\kappa}$ and $\boldsymbol{\kappa' - 1}$. For $\ell \in \{\kappa - 1, \kappa, \kappa' - 1\}$, they encode tuples $(\tilde{\mathfrak{r}}_\ell, \boldsymbol{i}'_\ell)$ such that $\tilde{\mathfrak{r}}_\ell \in \mathfrak{R}_0$ and $\boldsymbol{i}'_\ell \in \tilde{\mathfrak{r}}_\ell$. For any tile $t \in S_\iota$, denote $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa'}$ the **position field** of $f_{\blacklozenge}(t)$; then the **pixel position field of level** $\boldsymbol{\ell}$ of $f_{\blacklozenge}(t)$ is always non-blank and encodes a pair $(\tilde{\mathfrak{r}}_\ell, \boldsymbol{i}'_\ell) \in \mathfrak{R}_0 \times \mathbb{N}^d$ such that:

- If $\boldsymbol{i}_j = \mathbf{0}$ for every $\iota < j \leq \ell$, then $\boldsymbol{i}'_\ell$ is actually $\boldsymbol{i}'_\ell = \mathbf{0}$;

Furthermore, if the **pixel position field of level** $\boldsymbol{\ell}$ contains $(\tilde{\mathfrak{r}}_\ell, \boldsymbol{i}'_\ell)$, the facets $f_k^{\pm}(t)$ satisfy:

- If the partial odometer given by the rectangles $(\mathfrak{r}_j)_{\iota < j \leq \ell}$ yields $(\boldsymbol{i}_j)_{\iota < j \leq \ell} \pm \boldsymbol{e}^k = (\Diamond, \dots, \Diamond)$ (*i.e.* t appears on the border of macro-tiles of level ℓ), then the **pixel position field of level** $\boldsymbol{\ell}$ of $f_k^{\pm}(t)$ is left blank; in which case, the position $\boldsymbol{i}'_\ell$ must satisfy $\boldsymbol{i}'_\ell \in f_k^{\pm}(\tilde{\mathfrak{r}}_\ell)$;
- Otherwise, the **pixel position field of level** $\boldsymbol{\ell}$ of the facet $f_k^-(t)$ is also $(\tilde{\mathfrak{r}}, \boldsymbol{i}')$;
- And the **pixel position field of level** $\boldsymbol{\ell}$ of the facet $f_k^+(t)$ contains $(\tilde{\mathfrak{r}}', \boldsymbol{i}' + \boldsymbol{e}^k)$. □

Claim 12. *For any $\ell \in \{\kappa - 1, \kappa, \kappa' - 1\}$, and for any macro-tile $\tilde{t}^{(\ell)}$ of level ℓ and domain $\tilde{\mathfrak{r}}_\ell$, the* ***pixel position field of level ℓ*** *of $f_\blacklozenge(\tilde{t}^{(\ell)}_{\boldsymbol{i}'})$ contains the pair $(\tilde{\mathfrak{r}}_\ell, \boldsymbol{i}')$ for every $\boldsymbol{i}' \in \tilde{\mathfrak{r}}_\ell$.*

Consider a τ-macro-tile $\tilde{t}^{(\kappa'-1)}$ of level $\kappa' - 1$ and domain $\tilde{\mathfrak{r}}_{\kappa'-1}$, and let $\tilde{t}$ be a macro-tile appearing in $\tilde{t}^{(\kappa'-1)}$. From now on, we assume that the macro-tile $\tilde{t}$ contains, on the argument `O[3]` of its output facet, a tuple $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0, 1\}$. Informally, the pair $(i, j) \in \{0, \dots, 2d\} \times \mathbb{N}$ represents the index of a single bit from the symbol $a_{\ell+1}$ whose value $b \in \{0, 1\}$ should be synchronized with the bit `I[`$i+2$`][`j`]` from the input facet of $\tilde{t}^{(\kappa'-1)}$.

To perform this synchronization, we introduce two ***τ*-in pipe fields**. The **first *τ*-in pipe field** is of the form $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0, 1\}$; and, in a σ-macro-tile $\tilde{t}$ appearing in $\tilde{t}^{(\kappa'-1)}$:

- If the σ-macro-tile $\tilde{t}$ touches the smallest facet $f(\tilde{\mathfrak{r}}_{\kappa'-1})$ of the τ-macro-tile $\tilde{t}^{(\kappa'-1)}$, the **first *τ*-in-pipe field** of the tiles t in $\tilde{t}$ contains some (i, j, b) that is is initialized with the *copying method* from the output argument `O[3]` of the σ-macro-tile $\tilde{t}$;
- If the σ-macro-tile $\tilde{t}$ does not touch the input facet of the $\tilde{t}^{(\kappa'-1)}$, the **first *τ*-in-pipe field** of any tile t in $\tilde{t}$ must be blank;

The **second *τ*-in pipe field** is very similar to the already-defined **wire fields**, and allows to apply the routing lemma from tiles with **first *τ*-in-pipe field** $(i, j, b), \cdot)$ towards the input arguments `I[`$i+2$`][`j`]` $= b$ of the input facet of $\tilde{t}^{(\kappa'-1)}$.

Details. Let $t \in S_\iota$ be a tile and let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa'}$ be the **position field** and $(\tilde{\mathfrak{r}}, \boldsymbol{i}')$ be the **pixel position field of level $\kappa' - 1$** of $f_\blacklozenge(t)$. The **first *τ*-in pipe field** of $f_\blacklozenge(t)$ satisfies the following:

- The **first *τ*-in pipe field** of $f_\blacklozenge(t)$ can either be blank, or contain a tuple (i, j, b) such that $i \in \{0, \dots, 2d\}$, $j \in \mathbb{N}$ and $b \in \{0, 1\}$;
- Furthermore, if $(\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} = (\mathbf{0}, \dots, \mathbf{0})$, but that the position $\boldsymbol{i}'$ does not belong to the smallest facet $f(\tilde{\mathfrak{r}})$, then the **first *τ*-in pipe field** of $f_\blacklozenge(t)$ must be blank.

Furthermore, if the **first *τ*-in pipe field** of $f_\blacklozenge(t)$ is blank, then so are the **first *τ*-in pipe fields** of $f_k^\pm(t)$. Otherwise, assuming that $f_\blacklozenge(t)$ encodes some tuple (i, j, b), then for any direction $1 \leq k \leq d$:

- If the partial odometer defined by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\leq\iota'}$ yields $(\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} \pm \boldsymbol{e}^k \neq (\Diamond, \dots, \Diamond)$, then the **first *$\tau$*-in pipe field** of $f_k^\pm(t)$ also contains (i, j, b); otherwise, it is blank.

This ensures that, in any macro-tile $\tilde{t}$ appearing inside a τ-macro tile $\tilde{t}^{(\kappa'-1)}$ of level $\kappa' - 1$, the **first *τ*-in pipe fields** of the decorations are constant; and can only be non-blank across the smallest facet of $\tilde{t}^{(\kappa'-1)}$. We can thus apply the *copying method*: if the **compzone field** of $f_\blacklozenge(t)$ contains some $(\tilde{\mathfrak{c}}, \boldsymbol{i})$, that $f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$ denotes the input facet of the computation zone $\tilde{\mathfrak{c}}$, and that $\boldsymbol{i} \in f_h^+(\tilde{\mathfrak{c}})$ belongs to its output facet then,

- If the `output` variable of **processor field** of $f_h^-(t)$ contains a tuple (i, j, b'), then $b = b'$.

We then use the **second *τ*-in pipe fields** to implement the routing lemma. A ***τ*-consistency field of level ℓ** is of the form

$$(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \mathbb{N})^* \quad ; \text{ or } \quad (w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$$

to carry entries $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0, 1\}$ from the borders of the computation zone to its input facet. The adaptation from the already-defined **(*τ*)-wire fields** is straightforward, but we highlight the important modifications for clarity:

- (Border condition) If $\boldsymbol{i}' \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{r}}$ (in the **pixel position field of level $\kappa' - 1$** of $f_\blacklozenge(t)$), then the **second *τ*-in pipe field** of $f_k^\pm(t)$ must be blank;
- (Start of wires) If $(\boldsymbol{i}_\ell)_{\iota<\ell\leq\iota'} = (\mathbf{0}, \dots, \mathbf{0})$ (in the **position field** of $f_\blacklozenge(t)$), then an entry $((i, j, b), \texttt{start})$ appears in the **second *τ*-in pipe field** of $f_\blacklozenge(t)$ if and only if its **first *τ*-in pipe field** encodes the same non-blank (i, j, b); otherwise, no entry $(\cdot, \texttt{start})$ appears;
- (End of wires) Denoting $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ the ***τ*-compzone field of level $\kappa' - 1$**, and assuming that $\boldsymbol{i} \in f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$: if the `input` variable of the ***τ*-processor field of level $\kappa' - 1$** of $f_h^+(t)$ encodes a tuple $((i + 2, j, b), \texttt{end})$, then either there appears a single wire ending $(\cdot, \texttt{end})$ of the form $((i, j, b), \texttt{end})$ in the **second *τ*-in pipe field** of $f_\blacklozenge(t)$, or there are none. Otherwise (*i.e.* if $\boldsymbol{i} \notin f(\tilde{\mathfrak{c}})$), there is no wire end entry $(\cdot, \texttt{end})$. □

Claim 13. *Let $\tilde{t}^{(\kappa'-1)}$ be a τ-macro-tile of level $\kappa' - 1$, and let $a_{\kappa'} = (a_0, \dots, a_{2d}) \in T_*$ be the symbol given by the input arguments $(\texttt{I[2]}, \dots, \texttt{I[}2d+2\texttt{]})$ of its computation zone. Then for any σ-macro-tile $\tilde{t}$ appearing in $\tilde{t}^{(\kappa'-1)}$ with non-blank output argument `O[3]` $= (i, j, b)$, we have:*

- *$\tilde{t}$ appears along the smallest facet of $\tilde{t}^{(\kappa'-1)}$;*
- *i ranges in $\{0, \dots, 2d\}$, j ranges in $\{0, \dots, |a_i| - 1\}$ and $b = a_i(j)$.*

Notice that, inside a τ-macro-tile of level $\kappa' - 1$ and domain $\tilde{\mathfrak{r}}$, only σ-macro-tiles touching the smallest facet $f(\tilde{\mathfrak{r}})$ can have a non-blank output argument `O[3]`. This limitation, imposed by the routing lemma from the **second τ-in pipe**, will result in some unfortunate complications below.

Since this proof relies on a fixed point argument, the next level of simulation ι' will output an argument `O[3]` $= (i, j, b)$ consistent with their symbol $a_{\kappa'} \in T_*$ only if the tiles S_ι (*i.e.* of level ι) do so themselves (with their symbol a_κ). Thus, we must perform another step of information synchronization inside each τ-macro-tile $\tilde{t}^{(\kappa)}$ of level κ:

- From the output bits `O[`i`][`j`]` of the output facet of $\tilde{t}^{(\kappa)}$;
- Towards some individual tiles of S_ι (*i.e.* of level ι), which intuitively correspond to entry points of **τ-in pipes** from the "previous level" of simulation.

Inside each macro-tile $\tilde{t}^{(\kappa-1)}$ of level $\kappa - 1$, the **τ-in pipes** of the "previous" level of simulation may impose to transmit information across any specific facet of $\tilde{t}^{(\kappa-1)}$ (*c.f.* Claim 13). With the worst case in mind, we will thus distribute the bits of the symbol a_κ to ensure that at most $\prod_{\ell=\iota+1}^{\kappa-1} \mu_\ell$ such bits[23] can appear inside any such macro-tile $\tilde{t}^{(\kappa-1)}$.

To do so, we first introduce a **out-destination field**, which encodes an integer $k_0 \in \{1, \dots, d\}$ each each macro-tile of level $\kappa - 1$ to non-deterministically guess the facet containing the **τ-in-pipes** of the "previous level" of the simulation.

Details. Let t be a macro-tile and let $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa'}$ be the **pixel position field** of $f_{\blacktriangledown}(t)$. The **out-destination field** of $f_{\blacktriangledown}(t)$ is always non-blank, and if it encodes an integer $k_0 \in \{1, \dots, d\}$, then for any direction $k \in \{1, \dots, d\}$:

- If the partial odometer given by the rectangles $(\mathfrak{r}_\ell)_{\iota<\ell\leq\kappa-1}$ yields $(\boldsymbol{i}_\ell)_{\iota<\ell\leq\kappa-1} \pm \boldsymbol{e}^k = (\Diamond, \dots, \Diamond)$ (*i.e.* t appears on the border of a macro-tile of level $\kappa - 1$), then the **out-destination field** of $f_k^{\pm}(t)$ is blank;
- Otherwise, the **out-destination field** of $f_k^{\pm}(t)$ also contains k_0. □

We then introduce three **τ-out-pipe fields**. Based on the already-defined **(τ)-wire fields**, they implement three iterations of the routing lemma: in a τ-macro-tile $\tilde{t}^{(\kappa)}$ of level κ,

- The **first τ-out-pipe fields** implement a routing of the bits $(i, j, b) \in \mathbb{N}^2 \times \{0, 1\}$ from the arguments `O[`i`][`j`]` on the output facet of $\tilde{t}^{(\kappa)}$ to its smallest facet $f(\tilde{t}^{(\kappa)})$;
- The **second τ-out-pipe fields** distributes the bits (i, j, b) by permuting them on the smallest facet $f(\tilde{t}^{(\kappa)})$, ensuring that each macro-tile of level $\kappa - 1$ receives at most $\prod_{\ell=\iota+1}^{\kappa-1} \mu_\ell$ such bits;
- The **third τ-out-pipe fields** permute, inside each macro-tile of level $\kappa-1$, these received bits towards the corresponding facet $f_{k_0}^-$.

Details. The **first**, **second** and **third τ-out-pipe fields** are of the form:

$$(w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \mathbb{N})^* \quad ; \text{ or } \quad (w_i)_{i\in I} \in ((\mathbb{N}^2 \times \{0, 1\}) \times \{\texttt{start}, \texttt{end}, \texttt{cross}\})^*$$

to carry entries $(i, j, b) \in \{0, \dots, 2d\} \times \mathbb{N} \times \{0, 1\}$ by following the routing lemma. Similar to the already defined **(τ)-wire fields**, we highlight, for each of them, the bounds, starting and ending points of their wires.

Let $t \in S_\iota$ be a tile. For the **first τ-out-pipe field**, let $(\tilde{\mathfrak{c}}, \boldsymbol{i})$ and $(\tilde{\mathfrak{r}}_\kappa, \boldsymbol{i}'_\kappa)$ denote the **τ-compzone field** and the **pixel position field of level κ** of $f_{\blacktriangledown}(t)$:

- (Border condition) If $\boldsymbol{i} \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{c}}$ (*i.e.* t appears on the border of the computation zone $\tilde{\mathfrak{c}}$), then the corresponding $f_k^{\pm}(t)$ has blank **first τ-out pipe field**;
- (Start of wires) If $f_h^-(\tilde{\mathfrak{c}}) = f(\tilde{\mathfrak{c}})$ denotes the input facet of the computation zone $\tilde{\mathfrak{c}}$, and $\boldsymbol{i}$ belongs to $f_h^+(\tilde{\mathfrak{c}})$, then the **first τ-out pipe field** of $f_{\blacktriangledown}(t)$ contains an entry $((i, j, b), \texttt{start})$ if and only if the `output` variable of $f_h^-(t)$ encodes the same (i, j, b). Otherwise (*i.e.* $\boldsymbol{i}$ is not part of the output facet), it must be blank;
- (End of wires) Let us denote $f_{k_1}^-(\tilde{\mathfrak{c}})$ the facet of $\tilde{\mathfrak{c}}$ that is included in the smallest facet $f(\tilde{\mathfrak{r}}_\kappa)$. If $\boldsymbol{i} \in f_{k_1}^-(\tilde{\mathfrak{c}})$, the **first τ-out pipe field** of $f_{\blacktriangledown}(t)$ can either contain a single entry $((i, j, b), \texttt{end})$ for any $i \in \{0, \dots, 2d\}$, $j \in \{0, \dots, [\tilde{\mathfrak{c}}]\}$ and $b \in \{0, 1\}$; or no entry $(\cdot, \texttt{end})$ at all.

For the **second τ-out-pipe field**, let $(\tilde{\mathfrak{r}}_\kappa, \boldsymbol{i}'_\kappa)$ and $(\tilde{\mathfrak{r}}_{\kappa-1}, \boldsymbol{i}'_{\kappa-1})$ denote the **pixel position fields of level κ** and **$\kappa - 1$** of $f_{\blacktriangledown}(t)$:

[23] Where $\prod_{\ell=\iota+1}^{\kappa-1} \mu_\ell$ is a lower bound on the size of the smallest facet of any macro-tile of level $\kappa - 1$.

- (Border condition) If $\boldsymbol{i} \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{r}}_\kappa$ (*i.e.* t appears on the border of a macro-tile of level κ), then the corresponding $f_k^\pm(t)$ has blank **second τ-out pipe field**;
- (Start of wires) The **first τ-out-pipe field** of $f_{\blacklozenge}(t)$ contains an entry $((i,j,b),\texttt{end})$ for $i,j \in \mathbb{N}^2$ and $b \in \{0,1\}$ if and only if the **second τ-out-pipe field** of $f_{\blacklozenge}(t)$ contains the same entry $((i,j,b),\texttt{start})$;
- (End of wires) If $\boldsymbol{i}'_\kappa$ belongs to the smallest facet $f^-_{k_1}(\tilde{\mathfrak{r}}_\kappa) = f(\tilde{\mathfrak{r}}_\kappa)$, then $\boldsymbol{i}'_{\kappa-1}$ belongs to the facet $f^-_{k_1}(\tilde{\mathfrak{r}}_{\kappa-1})$ (we thus denote $n \in \mathbb{N}$ its boustrophedon indexing in $f^-_{k_1}(\tilde{\mathfrak{r}}_{\kappa-1})$): in this case, if $n \leq \prod_{\ell=\iota+1}^{\kappa-1} \mu_\ell$, then the **second τ-out-pipe field** of $f_{\blacklozenge}(t)$ can either contain an entry $((i,j,b),\texttt{end})$ for some $(i,j,b) \in \{0,\dots,2d\} \times \{0,\dots,[\tilde{\mathfrak{r}}_\kappa]\} \times \{0,1\}$ or be blank. Otherwise (*i.e.* n is too large or $\boldsymbol{i}'_\kappa$ does not belong to $f(\tilde{\mathfrak{r}}_\kappa)$), it must be blank.

For the **third τ-out-pipe field**, let $(\tilde{\mathfrak{r}}_{\kappa-1}, \boldsymbol{i}'_{\kappa-1})$ denote the **pixel position fields of level $\kappa-1$** of $f_{\blacklozenge}(t)$:

- (Border condition) If $\boldsymbol{i} \pm \boldsymbol{e}^k \notin \tilde{\mathfrak{r}}_{\kappa-1}$ (*i.e.* t appears on the border of a macro-tile of level $\kappa-1$), then the corresponding $f_k^\pm(t)$ has blank **third τ-out pipe field**;
- (Start of wires) The **second τ-out-pipe field** of $f_{\blacklozenge}(t)$ contains en entry $((i,j,b),\texttt{end})$ for $i,j \in \mathbb{N}^2$ and $b \in \{0,1\}$ if and only if the **third τ-out pipe field** of $f_{\blacklozenge}(t)$ contains the same entry $((i,j,b),\texttt{start})$;
- (End of wires) Let $k_0 \in \{1,\dots,d\}$ denote the **out-destination field** of $f_{\blacklozenge}(t)$. If $\boldsymbol{i}'_{\kappa-1}$ belongs to the facet $f^-_{k_0}(\tilde{\mathfrak{r}}_{\kappa-1})$, then the **third τ-out pipe field** of $f_{\blacklozenge}(t)$ can either contain an entry $((i,j,b),\texttt{end})$ for some $(i,j,b) \in \{0,\dots,2d\} \times \{0,\dots,[\tilde{\mathfrak{r}}_\kappa]\} \times \{0,1\}$ or be blank. Otherwise (*i.e.* $\boldsymbol{i}'_{\kappa-1}$ does not belong to $f^-_{k_0}(\tilde{\mathfrak{r}}_{\kappa-1})$), it must be blank. □

Claim 14. *Let $\tilde{t}^{(\kappa)}$ be a τ-macro-tile of level κ, and let $a_\kappa = (a_0,\dots,a_{2d}) \in T_*$ be the symbol given by the output arguments $(\texttt{O[0]},\dots,\texttt{O[2}d\texttt{]})$ of its computation zone. Then for every $i \in \{0,\dots,2d\}$ and $j \in \{0,\dots,|a_i|-1\}$, there exists a unique Wang tile t in the τ-macro-tile $\tilde{t}^{(\kappa)}$ such that the **third τ-out-pipe field** of $f_{\blacklozenge}(t)$ contains an entry (i,j,b'); in which case:*

- *The bit b' satisfies $b' = a_i(j)$;*
- *Denoting $\tilde{t}^{(\kappa-1)}$ the macro-tile of level $\kappa-1$ containing t, the tile t appears on the facet $f^-_{k_0}$ of $\tilde{t}^{(\kappa-1)}$, where k_0 is the **out-destination field** of $f_{\blacklozenge}(t)$.*

*Furthermore, all alternative choices for the **out-destination fields** of the sub-macro-tiles $\tilde{t}^{(\kappa-1)}$ (of level $\kappa-1$) result in valid macro-tiles $\tilde{t}^{(\kappa)}$.*

Numerics. Similarly to already-defined wires, the **τ-in pipe fields** of a tile $t \in S_\iota$ are of bit length $\mathcal{O}(\tilde{\mu}_\iota^{2\delta})$, and its **$\tau$-out pipe fields** are of bit length $\operatorname{polylog} K \cdot \mathcal{O}(\tilde{\mu}_\iota^{2\delta})$. □

There ends the definition of the tileset S_ι. Going back to the definition of $\varphi_{F(e)}$, if we fix values (K, K_{word}), $\iota \in \mathbb{N}$, $(\mu_\ell, \lambda_\ell)_{1 \leq \ell < \kappa}$ and $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota < \ell < \kappa}$ for the arguments of $\varphi_{F(e)}$, we accept a tile $t = (c_d^-,\dots,c_1^-,c_0,c_1^+,\dots,c_d^+) \in T_*$ if and only if it belongs to the resulting tileset S_ι; in which case, we return a tuple $(c_{\blacksquare}, \mathfrak{r}_\kappa, \boldsymbol{i}_\kappa, u)$ where:

- $c_{\blacksquare} \in \{\square, \blacksquare\}$ is the **chessboard field** of $f_{\blacklozenge}(t)$;
- $\mathfrak{r}_\kappa$ and $\boldsymbol{i}_\kappa$ for the eponymous entry in the **position field** of $f_{\blacklozenge}(t)$;
- $u \in \{0,1\}^*$ encodes some non-blank $(i,j,b) \in \{0,\dots,2d\} \times \mathbb{N} \times \{0,1\}$ if and only if the **third τ-out-pipe field** of $f_{\blacklozenge}(t)$ contains an entry $((i,j,b),\texttt{end})$.

Otherwise, we assume that the computation of $\varphi_{F(e)}$ rejects and does not return anything.

§ 5.7. Fixed point theorem

5.7.1. Time complexity and word length of $F(e)$. Following the *numerics* paragraphs from the previous sections, we deduce that $F(e)$ can be implemented to run in time

$$K_{\text{word}} \cdot \operatorname{polylog} K \cdot \mathcal{O}(|\texttt{I}| \cdot \operatorname{polylog} |\texttt{I}| + |e|)$$

and with word length

$$\mathcal{O}(\log |\texttt{I}| + \log(|e| + K + K_{\text{word}}))$$

on input arrays $\texttt{I} \in \{0,1\}^{**}$, where we interpret $\texttt{I[0]}$ as a pair of integers $\texttt{I[0]} = (K, K_{\text{word}})$, and $|\texttt{I}|$ refers to the *bit size* $|\texttt{I}| = \sum_{i=0}^{\text{len}(\texttt{I})} |\texttt{I}[i]|$.

Indeed, log-RAM machines can implement regular programming operations (additions, multiplications, logarithms, copies, comparisons…) in quasi-linear time.

5.7.2. Fixed point theorem. From any RAM program $e \in \{0,1\}^*$, the previous section defined a program $F(e) \in \{0,1\}^*$ such that $\varphi_{F(e)}$ defines, for fixed parameters (K, K_{word}), $\iota \in \mathbb{N}$, $(\mu_\ell, \lambda_\ell)_{1\leq\ell<\kappa}$ and $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota<\ell<\kappa}$, a valid set of Wang tiles S_ι. Noticing that the transformation $e \mapsto F(e)$ is computable and total, we deduce from the fixed point theorem (Proposition 3.6) that there exists a program $e \in \{0,1\}^*$ such that $\varphi_e = \varphi_{F(e)}$.

Furthermore, the time complexity and the word length of this code $e \in \{0,1\}^*$ respectively satisfy $T_e(\mathtt{I}) = \mathcal{O}(T_{F(e)}(\mathtt{I})) + \mathcal{O}(1)$ and $W_e(\mathtt{I}) = W_{F(e)}(\mathtt{I}) + \mathcal{O}(1)$. In particular, since $F(e)$ is a log-RAM program, so is e.

5.7.3. Fixing constants. We now fix, once and for all, the values of the integer constants (K, K_{word}). Notice that, for the construction to be correct, we need the σ-macro-tiles and τ-macro-tiles to embed enough log-RAM computation steps, and with large enough word length, to allow the computations of (respectively) $\varphi_e(\dots)$ and $\varphi_{\langle\tau\rangle}(\dots)$ to run properly.

Recall that, in a valid tiling $x \in T_\iota^{\mathbb{Z}^d}$, a macro-tiles of level $\kappa \leq \ell < \kappa'$ embed, it a computation zone of domain $\tilde{\mathfrak{c}}$, at least $\mu(\tilde{\mathfrak{c}}) \geq K \cdot \tilde{\mu}_\ell^{\gamma}$ steps of RAM computations operating on variables of word length at most $2K_{\text{word}} \cdot \log\|\tilde{\mathfrak{c}}\| \geq K_{\text{word}} \cdot \log \tilde{\mu}_\ell$. Thus, we will determine (K, K_{word}) to be large enough for these inequalities to be satisfied.

σ-macro-tiles. By the *numerics* paragraphs from the previous section, macro-colors are of bit length $\mathcal{O}(\tilde{\mu}_{\iota'}^{2\delta})$ in macro-tiles of level $\kappa \leq \iota' < \kappa'$; and so do the remaining arguments `I[0]`, `I[1]`, `I[2]` and `I[3]` on their input facets. Thus, we need to ensure that:

$$K_{\text{word}} \cdot \operatorname{polylog} K \cdot \mathcal{O}(\tilde{\mu}_{\iota'}^{2\delta} \cdot \operatorname{polylog} \tilde{\mu}_{\iota'}) \leq K \cdot \tilde{\mu}_{\iota'}^{\gamma}$$

and

$$\mathcal{O}(\tilde{\mu}_{\iota'}^{2\delta}) + \mathcal{O}(\log(K + K_{\text{word}})) \leq K_{\text{word}} \cdot \log \tilde{\mu}_{\iota'}.$$

Since $2\delta < \gamma$, this indeed holds for any value of $\tilde{\mu}_{\iota'}$ if we set $K_{\text{word}} = \sqrt{K}$ and pick K large enough.

τ-macro-tiles. In the hypotheses of Lemma 5.2, we assume that for $\kappa \leq \ell < \kappa'$, the substitution step $x^{(\ell+1)} \xrightarrow{\tau} x^{(\ell)}$ is computed in time $K_{\langle\tau\rangle}^{(t)} \cdot \tilde{\mu}_\ell{}^{\alpha}$, that $\|x^{(\ell+1)}\| \leq K_{\langle\tau\rangle}^{(t)} \cdot \tilde{\mu}_\ell{}^{\alpha}$; and that $\langle\tau\rangle$ has word length $W_{\langle\tau\rangle}(I) \leq K_{\langle\tau\rangle}^{(w)} \log|\mathtt{I}|$ on input arrays of bit size $|\mathtt{I}|$ (*c.f.* housekeeping). Thus, we need to ensure that:

$$K_{\langle\tau\rangle}^{(t)} \cdot \tilde{\mu}_\ell^{\alpha} \leq K \cdot \tilde{\mu}_\ell^{\gamma}$$

and

$$K_{\langle\tau\rangle}^{(w)} \cdot \mathcal{O}(\log(2d \cdot \lambda_\ell + \tilde{\mu}_\ell^{\alpha})) \leq K_{\text{word}} \cdot \log \tilde{\mu}_\ell.$$

Since $2\delta < \gamma$ and $\log \lambda_\ell = \mathcal{O}(\log \tilde{\mu}_\ell)$, this indeed holds for any value of $\tilde{\mu}_\ell$ if we set $K_{\text{word}} = \sqrt{K}$ and pick K large enough.

In what follows, we thus fix such a value $K \in \mathbb{N}$ and $K_{\text{word}} = \sqrt{K} \in \mathbb{N}$.

§ 5.8. End of the proof

The previous section has thus defined a function $\varphi_e(\dots)$ that recognizes a set of valid tiles in T_* that, in a sense, implements an infinite hierarchy of simulations. We now create a shift of finite type X_{init} that implements the "level 0" of these simulations.

The shift X_{init}. Let $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be the initial sofic shift space, for a finite $\mathcal{A} \subseteq \{0,1\}^*$. Since X is sofic, there exists some shift of finite type $X' \subseteq \mathcal{B}^{\mathbb{Z}^d}$ on some finite alphabet $\mathcal{B}$ and a projection $\pi' \colon \mathcal{B} \to \mathcal{A}$ such that $\pi'(X') = X$.

We now define a shift space X_{init} whose symbols are of the form

$$(a, b, \iota', \kappa', (\mu_\ell, \lambda_\ell)_{0\leq\ell\leq\kappa'}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{1\leq\ell\leq\kappa'}, (a_\ell)_{0\leq\ell\leq\kappa'}, c_{\blacksquare}, t)$$

where:

- $a \in \mathcal{A}$ is a letter from the original shift X;
- $b \in \mathcal{B}$ is a letter from the cover of finite type X' of X, and $a_0 = \pi(b)$;
- ι', κ' are two integers $\iota' < \kappa'$; furthermore, we assume that $\kappa' \leq N$ for some constant $N \in \mathbb{N}$ to be determined later;

- $(\mu_\ell, \lambda_\ell)_{0\le\ell\le\kappa'}$ are pairs of integers satisfying the hypotheses of the staircase lemma, *i.e.* denoting $\tilde\mu_\ell = \prod_{i\le\ell}\mu_i$:
  - ▷ $\mu_0 = \lambda_0 = 1$;
  - ▷ $2 \le \lambda_{\ell+1} \le 2^{K_\lambda\cdot\tilde\mu_\ell^\delta}$; $\mu_{\ell+1}^{K_\mu\cdot\log\tilde\mu_{\ell-1}} \ge \lambda_{\ell+1}$ and $\prod_{i=1}^{\kappa'}\mu_i \ge K\cdot\tilde\mu_{\kappa'}^\gamma$;
  - ▷ The integer κ' is the minimal integer satisfying $\prod_{i=1}^{\kappa'}\mu_i \ge K\cdot\tilde\mu_{\kappa'}^\gamma$;
  - ▷ The integer ι' is the maximal integer in $\{1,\dots,\kappa'\}$ satisfying $\prod_{i=\iota'+1}^{\kappa'}\mu_i \ge K\cdot\tilde\mu_{\kappa'}^\gamma$;
- $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{1\le\ell\le\kappa'}$ are pairs of rectangles $\mathfrak{r}_\ell \in \mathfrak{R}_0$ and $\boldsymbol{i}_\ell \in \mathfrak{r}_\ell$ satisfying $\mu_\ell \le \mu(\mathfrak{r}_\ell)$ and $\lambda\mathfrak{r}_\ell \le \lambda_\ell$;
- $(a_\ell)_{0\le\ell\le\kappa'}$ is a sequence of symbols such that:
  - ▷ $f_{\blacklozenge}(a_0) = a$;
  - ▷ For every $1 \le \ell \le \kappa'$, the Wang tile $a_{\ell-1} \in T_*$ is of word length $\|a_{\ell-1}\| \le K_{\langle\tau\rangle}\cdot\tilde\mu_{\ell-1}^\alpha$ and is a valid output of $\varphi_{\langle\tau\rangle}(\mathfrak{r}_\ell, \boldsymbol{i}_\ell, a_\ell)$ computed in time $K_{\langle\tau\rangle}\cdot\tilde\mu_{\ell-1}^\alpha$.
- $c_{\blacksquare}$ is a valid color in $\{\square, \blacksquare\}$;
- $t \in T_*$ is a Wang tile such that:
  - ▷ $\varphi_e((K, K_{\mathrm{word}}), \iota', (\mu_\ell, \lambda_\ell)_{1\le\ell<\kappa'}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota'<\ell<\kappa'}, t)$ is an accepting computation of φ_e that returns $(c_{\blacksquare}, \mathfrak{r}_{\kappa'}, \boldsymbol{i}_{\kappa'}, u)$ in time $K\cdot\tilde\mu_{\iota'}^\gamma$ for some $u \in \{0,1\}^*$;
  - ▷ Let us denote $(c_0, \dots, c_{2d}) = a_{\kappa'}$. If u is non-blank, then u encodes a tuple (i, j, b) for some $i \in \{0,\dots,2d\}$, $j \in \{0,\dots,|c_i|-1\}$ and such that $b = c_i(j)$.

Notice that, since $N \in \mathbb{N}$ is fixed, the alphabet of the shift space X_{init} is finite. We actually set

$$N = \frac{1}{1-\gamma}\log K$$

to ensure that there always exists $\kappa' \le N$ such that $\tilde\mu_{\kappa'} \ge K\cdot\tilde\mu_{\kappa'}^\gamma$: indeed, for any sequence $(\mu_\ell, \lambda_\ell)_{0\le\ell\le N}$, we have $\tilde\mu_N \ge 2^N > K^{\frac{1}{1-\gamma}}$ and thus $\tilde\mu_N \ge K\cdot\tilde\mu_N^\gamma$.

The configurations of the shift space X_{init} then become straightforward:

- The integers ι', κ' and the sequence $(\mu_\ell, \lambda_\ell)_{0\le\ell\le\kappa'}$ are constant across any configuration;
- The sequences $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{1\le\ell\le\kappa'}$ should draw nested grids in an odometer-like fashion (*c.f.* the **position field** from Section 5.3). Thus, any valid configuration $x \in X_{\mathrm{init}}$ defines a unique sequence of grids $(\mathfrak{g}_\ell)_{1\le\ell\le\kappa'}$;
- In a valid configuration $x \in X_{\mathrm{init}}$ defining grids $(\mathfrak{g}_\ell)_{1\le\ell\le\kappa'}$, letters a_ℓ should be constant across adjacent position $\boldsymbol{i}_\ell$ in the same rectangle $\mathfrak{r}_\ell$ of $\mathfrak{g}_\ell$; furthermore, across two adjacent rectangles $\mathfrak{r} = \mathfrak{r}_{\boldsymbol{j}}$ and $\mathfrak{r}' = \mathfrak{r}_{\boldsymbol{j}+\boldsymbol{e}^k}$ in a grid $\mathfrak{g}_\ell$, we enforce adjacency conditions between the corresponding symbols a_ℓ and a'_ℓ, *i.e.* $f_k^+(a_\ell) = f_k^-(a'_\ell)$;
- In a valid configuration $x \in X_{\mathrm{init}}$ defining the grid $\mathfrak{g}_{\kappa'}$, letters $c_{\blacksquare}$ should be constant across adjacent position $\boldsymbol{i}_{\kappa'}$ in the same rectangle $\mathfrak{r}_{\kappa'}$ of $\mathfrak{g}_{\kappa'}$; furthermore, adjacent rectangles $\mathfrak{r} = \mathfrak{r}_{\boldsymbol{j}}$ and $\mathfrak{r}' = \mathfrak{r}_{\boldsymbol{j}+\boldsymbol{e}^k}$ in $\mathfrak{g}_{\kappa'}$ should have opposite colors $c_{\blacksquare} \ne c'_{\blacksquare}$;
- In a valid configuration $x \in X_{\mathrm{init}}$ defining a value $\iota' < N$ and a grid $\mathfrak{g}_{\iota'}$, the tile t should be constant across adjacent position $\boldsymbol{i}_{\iota'}$ in the same rectangle $\mathfrak{r}_{\iota'}$ of $\mathfrak{g}_{\iota'}$; furthermore, across two adjacent rectangles $\mathfrak{r} = \mathfrak{r}_{\boldsymbol{j}}$ and $\mathfrak{r}' = \mathfrak{r}_{\boldsymbol{j}+\boldsymbol{e}^k}$ in a grid $\mathfrak{g}_{\iota'}$, we enforce adjacency conditions between the corresponding symbols t and t', *i.e.* $f_k^+(t) = f_k^-(t')$;
- Finally, using the finite family of forbidden patterns of X', we ensure that the projection of a valid configuration $x \in X_{\mathrm{init}}$ to the alphabet $\mathcal{B}$ results in a valid configuration of X';

Since there exists a uniform bound (depending on N) on the largest possible cells in the grids $\mathfrak{g}_\ell$ for $\ell < N$, the shift space X_{init} is actually a shift of finite type.

End of the proof. Let us denote $\pi_{\mathcal{A}}$ the projection of X_{init} towards their first component in $\mathcal{A}^{\mathbb{Z}^d}$. We then claim that:

Claim 15. *The shift space X_{init} satisfies $\pi_{\mathcal{A}}(X_{\mathrm{init}}) = \overleftarrow{X}_{\langle\tau\rangle}$. In particular, $\overleftarrow{X}_{\langle\tau\rangle}$ is a sofic shift.*

We prove this result by double inclusions.

Proof ($\overleftarrow{X}_{\langle\tau\rangle} \subseteq \pi_{\mathcal{A}}(X_{\mathrm{init}})$). Let $x \in \overleftarrow{X}_{\langle\tau\rangle}$ be a valid configuration. By definition, there exists a sequence of grids $(\mathfrak{g}_\ell)_{\ell\ge1}$ such that $(\mu(\mathfrak{g}_\ell), \lambda(\mathfrak{g}_\ell))_{\ell\ge1}$ satisfies the hypotheses of the staircase lemma; and a sequence of configurations $(x^{(\ell)})_{\ell\ge0}$ such that $x^{(\ell)} \xrightarrow[\mathfrak{g}_\ell]{\tau} x^{(\ell-1)}$ is computed in time $K_{\langle\tau\rangle}^{(t)}\cdot\tilde\mu_{\ell-1}^\alpha$, that the bit length $\|x^{(\ell)}\|$ satisfies $\|x^{(\ell)}\| \le K_{\langle\tau\rangle}^{(t)}\cdot\tilde\mu_{\ell-1}^\alpha$ and that $f_{\blacklozenge}(x^{(0)}) = x$ (where, as usual, $\tilde\mu_\ell = \prod_{i=1}^{\ell}\mu(\mathfrak{g}_\ell)$).

We denote $\mathfrak{g}_\ell = (\mathfrak{r}_{\boldsymbol{i}}^{(\ell)})_{\boldsymbol{i}\in\mathbb{Z}^d}$. Up to re-indexing of the grids $\mathfrak{g}_\ell$ and shifting the configurations $x_{\ell\geq 1}^{(\ell)}$, we assume that $\mathbf{0} \in \mathfrak{r}_{\mathbf{0}}^{(\ell)}$ for every $\ell \geq 1$. In particular, if $\tilde{\mathfrak{g}}_\ell = (\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ refers to any product grid $\tilde{\mathfrak{g}}_\ell = \bigcirc_{i=1}^{\ell} \mathfrak{g}_i$, then $\mathbf{0} \in \tilde{\mathfrak{r}}_{\mathbf{0}}$.

We now prove that there exists a configuration $y \in X_{\text{init}}$ such that $\pi_{\mathcal{A}}(y) = x$. To do so, consider the sequence $(\iota(n), \kappa(n))_{n\geq 1}$ given by the staircase lemma. We build in parallel a sequence of configurations $(y^{(\iota(n))})_{n\geq 1}$ corresponding to the different levels of fixed point simulation, *i.e.* such that tiles t appearing in $y^{(\iota(n))}$ are accepted by a valid computation $\varphi_e(\dots, \iota(n), \dots, t)$.

Let $n \in \mathbb{N}$ and $\boldsymbol{i} \in \mathbb{Z}^d$, let us define a sequence $f_{\iota(n)}(\boldsymbol{i}) = (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\ell>\iota(n)}$ by induction: initializing $\boldsymbol{i}_{\iota+1} = \boldsymbol{i}$, there exists for each $\iota(n) < \ell < \kappa(n)$ a unique rectangle $\mathfrak{r}_\ell$ in the grid $\mathfrak{g}_\ell$ such that $\boldsymbol{i}_\ell \in \mathfrak{r}_\ell$; furthermore, we set $\boldsymbol{i}_{\ell+1}$ to the index of the rectangle $\mathfrak{r}_\ell$ in the grid $\mathfrak{g}_\ell$. Intuitively, $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\ell>\iota(n)}$ refers to the shapes and positions of higher-level macro-tiles containing the position $\boldsymbol{i} \in \mathbb{Z}^d$ from the level $\iota(n)$.

We begin by fixing the as many fields in $y^{(\iota(n))}$ as we can: for $\boldsymbol{i} \in \mathbb{Z}^d$, let t refer to the tile appearing at position $\boldsymbol{i}$ in $y^{(\iota(n))}$, and denote $f_{\iota(n)}(\boldsymbol{i}) = (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\ell>\iota(n)}$; then:

- The **bound fields** of $f_{\blacklozenge}(t)$ are set to $(\mu(\mathfrak{g}_\ell), \lambda(\mathfrak{g}_\ell))_{\kappa(n)\leq\ell<\kappa(n+1)}$;
- The **position fields** of $f_{\blacklozenge}(t)$ are set to $([\mathfrak{r}_\ell], \boldsymbol{i}_\ell \bmod \mathfrak{g}_\ell)_{\iota(n)<\ell<\kappa(n+1)}$;
- The **chessboard field** of $f_{\blacklozenge}(t)$ (resp. **p-chessboard field**) is set to □ if and only if $\sum_{k=1}^{d} i_k \bmod 2 = 0$, where $(i_1, \dots, i_d) = \boldsymbol{i}_{\kappa(n)}$ (resp. $\boldsymbol{i}_{\kappa(n+1)}$)
- The **p-level field** of $f_{\blacklozenge}(t)$ is set to $\iota(n+1)$.

The **bound** and **position fields** uniquely determine the **compzone fields** in $y^{(\iota(n))}$. Using our hypotheses on the configurations $(x^{(\ell)})_{\kappa(n)\leq\ell\leq\kappa(n+1)}$ and the computation of the substitution $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$ for $\kappa(n) \leq \ell < \kappa(n+1)$, we furthermore entirely fix all **τ-(...) fields** in $y^{(\iota(n))}$:

- The **position fields** of $y^{(\iota(n))}$ uniquely determine the τ-macro-tiles of level ℓ in $y^{(\iota(n))}$ for every $\kappa(n) \leq \ell < \kappa(n+1)$;
- Such a τ-macro-tile $\tilde{t}$ of level ℓ covering the position $\boldsymbol{i} \in \mathbb{Z}^d$ in $y^{(\iota(n))}$ should implement a valid computation $x^{(\ell+1)}_{\boldsymbol{i}_{\ell+1}} \xrightarrow{\tau} x^{(\ell)}|_{\mathfrak{r}^{(\ell+1)}_{\boldsymbol{i}_{\ell+1}}}$, where $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\ell>\iota(n)} = f_{\iota(n)}(\boldsymbol{i})$;
- In $y^{(\iota(n))}$, we arbitrarily fix a layout of valid **τ-wires** and **τ-consistency fields**; similarly, we arbitrarily fix a layout of **τ-in** and **τ-out pipes**, that is consistent with their counterparts in $y^{(\iota(n+1))}$ and $y^{(\iota(n-1))}$.

We have yet to fix the **processor**, **color** and **wire fields** in $y^{(\iota(n))}$. To fix these, let us first consider some level $\iota(N)$ and the associated (partially defined) configuration $y^{(\iota(N))}$. For any $\boldsymbol{i} \in \mathbb{Z}^d$, consider the tile t at position $\boldsymbol{i}$ in $y^{(\iota(N))}$. Assume that there exists a completion of the **processor**, **color** and **wire fields** of t such that

$$\varphi_e\big((K, K_{\text{word}}), \iota(N), (\mu(\mathfrak{g}_\ell), \lambda(\mathfrak{g}_\ell))_{1\leq\ell<\kappa(N)}, ([\mathfrak{r}_\ell], \boldsymbol{i}_\ell \bmod \mathfrak{g}_\ell)_{\iota(N)<\ell<\kappa(N)}, t\big)$$

admits an accepting computation of length at most $K \cdot \tilde{\mu}^{\gamma}_{\iota(N)}$ (*c.f.* Section 5.7.3). Then this computation can be used to fix the **processor**, etc... **fields** of the corresponding σ-macro-tile of level $\iota(N-1)$:

- More precisely, let $(\tilde{\mathfrak{r}}_{\boldsymbol{j}})_{\boldsymbol{j}\in\mathbb{Z}^d}$ denote the rectangles of the product grid $\bigcirc_{\ell=\iota(N-1)+1}^{\iota(N)} \mathfrak{g}_\ell$, and let $\tilde{t}$ refer to the σ-macro-tile $y^{\iota(N-1)}|_{\tilde{\mathfrak{r}}_{\boldsymbol{i}}}$;
- We fix the **color fields** of $\tilde{t}$ to draw the colors of the tile t;
- We fix the **processor fields** in $\tilde{t}$ to implement the accepting computation of t by $\varphi_e(\dots)$ of length $K \cdot \tilde{\mu}^{\gamma}_{\iota(N)}$; In particular, by design, the computation zone $\tilde{\mathfrak{c}}$ of $\tilde{t}$ is large enough to embed the whole computation;
- We fix any valid layout for the **wire fields** in $\tilde{t}$, which is possible by the routing lemma.

This inductively defines a completion of the **processor**, ... **fields** for increasingly large rectangles of tiles in the configurations $y^{(\iota(n))}$ for $\iota(n) < \iota(N)$.

We now conclude the construction of the configurations $y_{n\geq 1}^{(\iota(n))}$ by a compactness argument. Instead of considering a single Wang tile $y_{\boldsymbol{i}}^{\iota(N)}$ accepted by φ_e, consider a completion of the **processor**, **color** and **wire fields** of $y^{\iota(N)}|_{\{-1,0,1\}^d}$ (it is always possible to define a locally valid pattern in $y^{\iota(N)}$ of such a small size). Since we assumed that $\mathbf{0} \in \mathfrak{r}_{\mathbf{0}}^{(\ell)}$ for every $\ell \geq 1$, and that the substitution τ is *growing* (*i.e.* the rectangles in the grids $\mathfrak{g}_\ell$ of edge length at least 2), this

recursively defines for every $\iota(n) < \iota(N)$ a locally valid completion of the **processor**, … **fields** of $y^{(\iota(n))}$ in the domain $\{-2^{\iota(N)-\iota(n)}, \dots, 2^{\iota(N)-\iota(n)}\}^d$.

Using compactness and a diagonal argument, we deduce the existence of configurations $y^{\iota(n)}_{n\in\mathbb{N}}$ such that, for each $\boldsymbol{i} \in \mathbb{Z}^d$,

$$\varphi_e\big((K, K_{\mathrm{word}}), \iota(n), (\mu(\mathfrak{g}_\ell), \lambda(\mathfrak{g}_\ell))_{1\le\ell<\kappa(n)}, ([\mathfrak{r}_\ell], \boldsymbol{i}_\ell \bmod \mathfrak{g}_\ell)_{\iota(n)<\ell<\kappa(n)}, y^{\iota(n)}_{\boldsymbol{i}}\big)$$

results in an accepting computation, for $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\ell>\iota(n)} = f_{\iota(n)}(\boldsymbol{i})$. By considering a valid configuration $x' \in X_{\mathrm{init}}$ implementing the initialization of the configuration $y^{(\iota(1))}$ and such that $\pi_{\mathcal{A}}(x') = x$, we conclude that $x \in \pi_{\mathcal{A}}(X_{\mathrm{init}})$, and thus that $\overleftarrow{X}\langle\tau\rangle \subseteq \pi_{\mathcal{A}}(X_{\mathrm{init}})$. □

We now prove the converse inclusion:

Proof ($\overleftarrow{X}_{\langle\tau\rangle} \supseteq \pi_{\mathcal{A}}(X_{\mathrm{init}})$). Let $x \in X_{\mathrm{init}}$. By definition, $\pi_{\mathcal{A}}(x)$ is a valid configuration in the sofic shift space X. We now prove that it belongs to $\overleftarrow{X}_{\langle\tau\rangle}$ by exhibiting a sequence of configurations $(x^{(\ell)})_{\ell\ge0}$ such that $f_{\blacktriangleright}(x^{(0)}) = \pi_{\mathcal{A}}(x)$ and $x^{(\ell+1)} \xrightarrow{\tau} x^{(\ell)}$ satisfying the hypotheses of Lemma 5.2.

We build this sequence $(x^{(\ell)})_{\ell\ge0}$ by defining two sequences $(\iota(n), \kappa(n))_{n\ge1}$ as in the staircase lemma, and inductively defining the segment $(x^{(\ell)})_{\kappa(n)\le\ell\le\kappa(n+1)}$.

Initialization. By definition of X_{init}, there exists some integers $\iota' < \kappa'$, a sequence of bounds $(\lambda_\ell, \mu_\ell)_{1\le\ell<\kappa'}$, a sequence of grids $(\mathfrak{g}_\ell)_{1\le\ell\le\kappa'}$ and a sequence of configurations $(x^{(\ell)})_{0\le\ell\le\kappa'}$ such that $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$ in time $K^{(t)}_{\langle\tau\rangle} \cdot \tilde{\mu}_\ell^{\alpha}$ (where, as usual, $\tilde{\mu}_\ell = \prod_{i\le\ell}\mu_i$). Furthermore, it defines a configuration $y \in T_*^{\mathbb{Z}^d}$ such that each tile $y_{\boldsymbol{i}}$ is accepted by

$$\varphi_e\big((K, K_{\mathrm{word}}), \iota', (\mu_\ell, \lambda_\ell)_{\iota'<\ell<\kappa'}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota'<\ell<\kappa'}, y_{\boldsymbol{i}}\big)$$

for some consistent $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{1\le\ell<\kappa'}$. We denote $\iota(1) = \iota'$ and $\kappa(1) = \kappa'$.

Induction. Assume that $x \in T_*$ is a valid tiling whose every tiles are accepted by a computation of

$$\varphi_e\big((K, K_{\mathrm{word}}), \iota(n), (\mu_\ell, \lambda_\ell)_{1\le\ell<\kappa(n)}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota(n)<\ell<\kappa(n)}, y_{\boldsymbol{i}}\big),$$

where $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota(n)<\ell<\kappa(n)}$ is deduced from $f_{\iota(n)}(\boldsymbol{i})$ in the grids $(\mathfrak{g}_\ell)_{1\le\ell\le\kappa(n)}$. Then by construction of φ_e, we have:

- The **bound fields** and **p-level fields** in the tiling x uniquely determine two integers $\iota' < \kappa'$ and the next segment of bounds $(\mu_\ell, \lambda_\ell)_{\kappa(n)\le\ell<\kappa'}$. Furthermore, they ensure that the sequences $(\iota(m), \kappa(m))_{m\le n}$ satisfy the hypotheses of the staircase lemma;
- The **position fields** in the tiling x uniquely determine a extension of the grid sequence $(\mathfrak{g}_\ell)_{1\le\ell\le\kappa(n)}$ into $(\mathfrak{g}_\ell)_{1\le\ell<\kappa'}$;

Furthermore, the σ-macro-tiles in x define a grid $\tilde{\mathfrak{g}}_{\iota'} = (\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ and a configuration $y \in T_*^{\mathbb{Z}^d}$. More precisely, for every $\boldsymbol{i} \in \mathbb{Z}^d$, let $\tilde{t}$ be the σ-macro-tile $x|_{\tilde{\mathfrak{r}}_{\boldsymbol{i}}}$:

- The **color fields** in $\tilde{t}$ define the Wang tile $y_{\boldsymbol{i}} \in T_*$;
- The **processor fields** in $\tilde{t}$ embed an accepting computation of the form

  $$\varphi_e\big((K, K_{\mathrm{word}}), \iota', (\mu_\ell, \lambda_\ell)_{1\le\ell<\kappa'}, (\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota'<\ell<\kappa'}, y_{\boldsymbol{i}}\big)$$

  where $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)_{\iota'<\ell<\kappa'}$ is deduced from the **position fields** of the tiles in $\tilde{t}$;
- And this computation outputs some tuple $(c_{\blacksquare}, \mathfrak{r}_{\kappa'}, \boldsymbol{i}_{\kappa'}, u)$ for $c_{\blacksquare} \in \{\square, \blacksquare\}$, $\mathfrak{r}_{\kappa'} \in \mathfrak{R}_0$, $\boldsymbol{i}_{\kappa'} \in \mathfrak{r}_{\kappa'}$ and $u \in \{0,1\}^*$.

Furthermore, the pair $(\mathfrak{r}_{\kappa'}, \boldsymbol{i}_{\kappa'})$ output by the σ-macro-tile $x|_{\tilde{\mathfrak{r}}_{\boldsymbol{i}}}$ coincides with the eponymous pair from the **position field** of $y_{\boldsymbol{i}}$; thus, since y is a valid tiling whose every tile is accepted by $\varphi_e(\dots, \iota', \dots)$, these pairs collectively define a valid grid $\mathfrak{g}_{\kappa'}$.

Consider now the τ-macro-tiles of x. For every $\kappa(n) \le \ell < \kappa'$, the τ-macro-tiles of level ℓ define a grid $\tilde{\mathfrak{g}}_\ell = (\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ and two configurations $x^{(\ell)}$ and $\tilde{x}^{(\ell+1)}$. For every $\boldsymbol{i} \in \mathbb{Z}^d$, let $\tilde{t}$ be the corresponding τ-macro-tile $x|_{\tilde{\mathfrak{r}}_{\boldsymbol{i}}}$ of level ℓ:

- The ***τ*-wire fields** in $\tilde{t}$ define, on the border of $\tilde{t}$, a valid Wang tile $x^{(\ell)}_{\boldsymbol{i}} \in T_*$;
- The ***τ*-consistency fields** in $\tilde{t}$ define, on the border of $\tilde{t}$, a valid Wang tile $\tilde{x}^{(\ell+1)} \in T_*$;
- The ***τ*-processor fields** in $\tilde{t}$ embed an accepting computation of $\varphi_{\langle\tau\rangle}(\mathfrak{r}_\ell, \boldsymbol{i}_\ell, \tilde{x}^{(\ell+1)}_{\boldsymbol{i}}) \to x^{(\ell)}_{\boldsymbol{i}}$, where $(\mathfrak{r}_\ell, \boldsymbol{i}_\ell)$ is the eponymous pair from the **position fields** of the tiles in $\tilde{t}$;

Furthermore, the τ-wire fields ensure that every $x^{(\ell)}$ is a valid Wang tiling; and for every $\kappa(n) \leq \ell < \kappa' - 1$, the **$\tau$-pipes** ensure that for every $\boldsymbol{i} \in \mathbb{Z}^d$, any tile $t \sqsubseteq \tilde{x}^{(\ell+1)}|_{\mathfrak{r}_{\boldsymbol{i}}}$ must actually coincide with $x^{(\ell+1)}_{\boldsymbol{i}}$ (for $(\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d} = \mathfrak{g}_{\ell+1}$). Thus, we are left with comparing the configuration $\tilde{x}^{\kappa'}$ and the substitution steps implemented in y.

Let $(\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d} = \tilde{\mathfrak{g}}_{\kappa'}$ denote the product grid $\bigcirc_{\ell=\iota(n)+1}^{\kappa'} \mathfrak{g}_\ell$, and let $(\tilde{\mathfrak{r}}'_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d} = \tilde{\mathfrak{g}}'_{\kappa'}$ denote the product grid $\bigcirc_{\ell=\iota'+1}^{\kappa'} \mathfrak{g}_\ell$. Consider any pattern $w = x|_{\tilde{\mathfrak{r}}_{\boldsymbol{i}}}$ for some $\boldsymbol{i} \in \mathbb{Z}^d$;

- Applying the previous paragraph on y, the τ-macro-tiles of level κ' in y define a valid Wang tiling $x^{(\kappa')}$; we denote $(c_0, \dots, c_{2d}) = x^{(\kappa')}_{\boldsymbol{i}}$;
- By definition of the **τ-out pipes** in the τ-macro-tile $y|_{\tilde{\mathfrak{r}}'_{\boldsymbol{i}}}$: for every (i, j) such that $i \in \{0, \dots, 2d\}$ and $j \in \{0, \dots, |c_i| - 1\}$, there exists a unique position $\boldsymbol{p}_{i,j} \in \tilde{\mathfrak{r}}'_{\boldsymbol{i}}$ such that the **third τ-out pipe field** of $f_{\blacklozenge}(y_{\boldsymbol{p}_{i,j}})$ contains an entry $((i, j, b), \texttt{end})$ for some $b \in \{0, 1\}$; in which case, $b = c_i(j)$;
- By definition of $\varphi_e(\dots)$, for every (i, j) such that $i \in \{0, \dots, 2d\}$ and $j \in \{0, \dots, |c_i| - 1\}$, $\boldsymbol{p}_{i,j}$ is the index of the unique σ-macro-tile $x|_{\tilde{\mathfrak{r}}_{\boldsymbol{p}_{i,j}}}$ whose **second τ-in pipe fields** contain en entry $((i, j, b), \texttt{start})$; in which case, $b = c_i(j)$;

We conclude that for every $\boldsymbol{i} \in \mathbb{Z}^d$, any tile $t \sqsubseteq \tilde{x}^{(\kappa')}|_{\mathfrak{r}_{\boldsymbol{i}}}$ coincides with $x^{(\kappa')}_{\boldsymbol{i}}$ (for $(\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d} = \mathfrak{g}_{\kappa'}$). We conclude this induction step by setting $\iota(n+1) = \iota'$ and $\kappa(n+1) = \kappa'$.

By induction, we conclude that there exists a sequence $(x^{(\ell)})_{\ell\geq 0}$ such that $x^{(\ell+1)} \xrightarrow[\mathfrak{g}_{\ell+1}]{\tau} x^{(\ell)}$, and computed in time $K^{(t)}_{\langle\tau\rangle} \cdot \tilde{\mu}_\ell^\gamma$. Since the sequence $(\mu_\ell, \lambda_\ell)_{\ell\geq 1}$ satisfies the growth conditions of Lemma 5.2, so does the sequence $(\mu_\ell(\mathfrak{g}_\ell), \lambda_\ell(\mathfrak{g}_\ell))_{\ell\geq 1}$; and we conclude that $f_{\blacklozenge}(x^{(0)}) = x$ is a valid configuration in $\overline{X}_{\langle\tau\rangle}$. □

§ 5.9. Final word

We now make a few comments on Lemma 5.2 and its proof.

Pre-composition vs. parallel computations. One of the main difficulties of this proof (compared to previous fixed point constructions in the literature) comes from simulating several substitution steps $x^{(\kappa(n+1))} \xrightarrow{\tau} \dots \xrightarrow{\tau} x^{(\kappa(n))}$ at a single level of fixed point simulation $\iota(n)$. This is made necessary by the fact that the substitution τ can output arbitrarily small patterns, *e.g.* of domain $[\![2]\!]^d$.

Instead of drawing multiple levels of τ-macro-tiles in parallel, one could prefer to make single τ-macro-tiles compute several compositions of τ with itself algorithmically (informally, using a standard fixed point construction computing some power τ^n instead of τ), thus ensuring that the resulting patterns grow fast enough to apply a step of fixed point simulation. We claim that, while this choice is definitely possible, it results in significantly worse bounds on the possible growths of the grids $(\mathfrak{g}_\ell)_{\ell\geq 1}$ and the computational requirements on τ.

Indeed, consider a macro-tile $\tilde{t}$ of level ℓ and domain $\tilde{\mathfrak{r}}_\ell \in \mathfrak{R}_0$. If we are to compute some $n \in \mathbb{N}$ steps of τ inside $\tilde{t}$, the embedding of Lemma 3.8 only allows to draw $\mathcal{O}(\mu(\tilde{\mathfrak{r}}_\ell)^\alpha)$ steps of RAM computations in $\tilde{t}$. In particular, this limits the word length of the symbols in $x^{(\ell+n)}$ to $\|x^{(\ell+n)}\| = \mathcal{O}(\tilde{\mu}_\ell^\alpha)$, instead of the bound $\mathcal{O}(\tilde{\mu}^\alpha_{\ell+n-1})$ from Condition 2.(ii) in Lemma 5.2.

Algorithmic vs. geometric computations. One may also wonder why we actually consider a local computable substitution $\varphi_{\langle\tau\rangle}$ instead of a more classical notion of computability on substitutions. The answer is, once again, that it allows for better bounds on the possible growths of the grids $(\mathfrak{g}_\ell)_{\ell\geq 1}$ and the computational requirements on τ.

For fixed $n \in \mathbb{N}$, assume that we are designing a finite tileset T over the colors $\{0, 1\}^n$ in which individual tiles can embed t steps of RAM computations. In a finite cube $[\![N]\!]^d$ (for $N \leq 2^n$), there are two possibilities to embed computations in the patterns of $T^{[\![N]\!]^d}$:

- On the one hand, the individual tiles in $[\![N]\!]^d$ can perform independent computations (with, maybe, some small $\mathcal{O}(n)$ communications between adjacent tiles). Globally, such a pattern will embed

$$t \cdot N^d$$

steps of RAM computations, at the cost of distributing them as independent segments of length t;

- On the other hand, the tiles in $[\![N]\!]^d$ can collectively perform the RAM simulation of Lemma 3.8. This embedding is based upon drawing the *space-time diagram* of a processor array, thus requiring the use of a dimension to act as the "time" of the computation. Since this processor array is limited by its size $\mathcal{O}(N^{d-1})$ to embed RAM computations, such a pattern will only embed

$$t \cdot N^{d-1}$$

steps of RAM computations. Nevertheless, these computations may be performed by a single (non-distributed) machine, thus allowing for as many steps of sequential computations.

Entropy of the construction. In order to study the pattern complexity of the construction, we consider below the sources of non-determinism in the fixed point configurations:

- The substitution τ may be non-deterministic. This non-determinism is unavoidable in our proof, and comes from the non-determinism of the program $\langle\tau\rangle$ computing a local map for τ. This non-determinism can only be removed by restricting Lemma 5.2 to deterministic programs $\langle\tau\rangle$;
- If the substitution τ is deterministic, then the embedded log-RAM simulations are non-ambiguous, in the sense that there is exactly one computation embedding per accepted run of the function $\varphi_{\langle\tau\rangle}$;
- The routing lemma may result in several possible wirings for a given set of end points. This issue can actually be avoided by considering a deterministic version of the lemma, at the cost of synchronizing the wiring tiles with its position in each macro-tile (*c.f.* Remark 4.5).
- The **τ-out pipes** may also result in several wirings. This issue can also be fixed by implementing a fixed layout for these pipes to follow (*e.g.* depending on the position of the σ-macro-tiles in each τ-macro-tile of level κ).

6. Sofic realization of limit spaces

A substitution $\tau\colon \mathcal{A} \rightrightarrows \mathcal{A}^{\mathfrak{R}}$ naturally defines a *limit* shift space $\overleftarrow{X}_\tau \subseteq \mathcal{A}^{\mathbb{Z}^d}$ by considering the limit configurations obtained after infinitely many substitution steps. While substitutive words and shift spaces have been extensively studied in the one-dimensional setting [Fog02; AA20], they also frame one of the first major results in multidimensional symbolic dynamics: if $\mathcal{A}$ is a finite alphabet and τ has finite image, then $\overleftarrow{X}_\tau$ is actually a sofic shift [Moz89] in dimension $d \geq 2$.

S-adic systems extend the notion of substitutive shifts by allowing a more flexible structure of substitutions: instead of considering a single substitution τ, an S-adic configuration is obtained as a limit of the form

$$x = x_0 \xleftarrow{\tau_1} x_1 \xleftarrow{\tau_2} x_2 \xleftarrow{\tau_3} x_3 \dots$$

for $(\tau_\ell)_{\ell\geq 1}$ an infinite sequence of substitutions. One can naturally impose various restrictions on such sequences: operating on a fixed alphabet, ranging over a finite set of substitutions, etc… They enjoy a abundant literature in the one-dimensional setting [BD14], and the natural generalization of [Moz89]'s theorem to multidimensional S-adic systems has been proved in [AS14].

In this article, we are interested in the soficity of multidimensional shift spaces obtained as limit spaces of S-adic systems. While the intermediate $(x^{(\ell)})_{\ell\in\mathbb{N}}$ defining an S-adic configuration

$$x = x_0 \xleftarrow{\tau_1} x_1 \xleftarrow{\tau_2} x_2 \xleftarrow{\tau_3} x_3 \dots$$

must themselves be limit configurations of an S-adic system, the fixed point construction from Lemma 5.2 further allows to consider additional restrictions at each step: namely, we force each intermediate configuration $x^{(\ell)}$ to belong to some shift of finite type X_ℓ. To describe such local validity conditions, we consider substitutions "with context" under the formalism of *dill maps* [Ram23].

§ 6.1. Substitution, dill maps and limit spaces

6.1.1. Dill maps. Introduced in [ST15], dill maps[24] generalize substitutions and morphisms under a common formalism, thus defining substitutions of individual letters that are "context sensitive", *i.e.* whose images depend on the symbols borne by neighboring cells:

Definition 6.1 (Dill map)**.** Given two alphabets $\mathcal{A}$ and $\mathcal{B}$ and a finite neighborhood $V \subseteq \mathbb{Z}^d$, a d-dimensional *dill map* is a non-deterministic map $\tau\colon \mathcal{A}^V \rightrightarrows \mathcal{B}^{\mathfrak{R}_0}$.
A Dill map τ extends to whole configurations as follows: for $x \in \mathcal{A}^{\mathbb{Z}^d}$, we define $\tau(x)$ as the set of configurations $y \in \mathcal{B}^{\mathbb{Z}^d}$ for which there exists a grid $\mathfrak{g} = (\mathfrak{r}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d}$ such that $[y|_{\mathfrak{r}_{\boldsymbol{i}}}] \in \tau(x|_{\boldsymbol{i}+V})$. If $V \subseteq \{-r, \dots, r\}^d$, we say that τ has *radius* $r \in \mathbb{N}$.

In particular, substitutions form exactly the dill maps of radius 0; and morphisms are deterministic dill maps whose images consist of a single cell. For readability purposes, we will sometimes denote $x \xrightarrow{\tau} y$ if $y \in \tau(x)$. By analogy with substitutions, we say that a dill map $\tau\colon \mathcal{A}^V \to \mathcal{B}^{\mathfrak{R}_0}$ is *growing* if for every pattern $u \in \mathcal{A}^V$ and every image $w \in \tau(u)$, the domain $\mathrm{dom}(w) = [\![n_1, \dots, n_d]\!] \in \mathfrak{R}_0$ satisfies $n_k \geq 2$ for every $1 \leq k \leq d$. For more information about dill maps, we refer to [ST15; Ram23].

6.1.2. Limit spaces. In the same way that sequences of substitutions can define an S-adic configuration, dill maps can be used to define limit *D-adic* configurations. More precisely, for $(\tau_\ell)_{\ell\geq 1}$ a sequence of dill maps where $\tau_\ell\colon \mathcal{A}_\ell^{V_\ell} \to \mathcal{A}_{\ell-1}^{\mathfrak{R}_0}$, we say that a configuration $x \in \mathcal{A}_0^{\mathbb{Z}^d}$ is *D-adic* if there exists a sequence of configurations $(x^{(\ell)})_{\ell\in\mathbb{N}}$ such that $x^{(\ell)} \in \mathcal{A}_\ell^{\mathbb{Z}^d}$ and

$$x = x^{(0)} \xleftarrow{\tau_1} x^{(1)} \xleftarrow{\tau_2} x^{(2)} \xleftarrow{\tau_3} x^{(3)} \dots$$

The sequence $(\tau_\ell)_{\ell\geq 1}$ is called a *directive sequence* for x. A D-adic shift is then a set of D-adic configurations:

Definition 6.2 (D-adic limit space)**.** Let $(\mathcal{A}_\ell)_{\ell\in\mathbb{N}}$ be a sequence of finite alphabets, and let $(\mathcal{D}_\ell)_{\ell\geq 1}$ be a sequence of finite sets $\mathcal{D}_\ell \subseteq \left(\mathcal{A}_{\ell+1}^{V_\ell} \rightrightarrows \mathcal{A}_\ell^{\mathfrak{R}_0}\right)$ of dill maps. A shift space $X \subseteq \mathcal{A}_0^{\mathbb{Z}^d}$ and a set $\mathcal{D} \subseteq \prod_{\ell\geq 1} \mathcal{D}_\ell$ of *directive sequences* defines a *D-adic limit shift space*

$$\overleftarrow{X}_{\mathcal{D}} = \left\{x^{(0)} \in X : \exists (\tau_\ell)_{\ell\geq 1} \in \mathcal{D},\ \exists (x^{(\ell)})_{\ell\geq 1},\ x^{(0)} \xleftarrow{\tau_1} x^{(1)} \xleftarrow{\tau_2} x^{(2)} \xleftarrow{\tau_3} x^{(3)} \dots\right\}.$$

In case the initial shift X is unspecified, the limit space $\overleftarrow{X}_{\mathcal{D}}$ will implicitly be defined for $X = \mathcal{A}_0^{\mathbb{Z}^d}$.

The authors do not know of any previous study of D-adic limit spaces. Nevertheless, when all sets $\mathcal{D}_\ell$ in fact consist of substitutions, the resulting limit space is called an *S-adic shift*, which have been given much more consideration in the literature. For more details about S-adic shifts, we refer to the extensive survey [BD14], and to [AS14] for their multidimensional counterparts.

Dill maps and SFTs. Consider $(X_\ell)_{\ell\in\mathbb{N}}$ an arbitrary sequence of shifts of finite type $X_\ell \subseteq \mathcal{A}_\ell^{\mathbb{Z}^d}$. For any sequence $(\tau_\ell)_{\ell\geq 1}$ of substitutions $\tau_\ell\colon \mathcal{A}_\ell \rightrightarrows \mathcal{A}_{\ell-1}^{\mathfrak{R}_0}$, one can consider the limit configurations $x \in X_0$ such that there exists $(x^{(\ell)})_{\ell\geq 1}$ satisfying:

$$\begin{array}{ccccccccc} x = & x^{(0)} & \xleftarrow{\tau_1} & x^{(1)} & \xleftarrow{\tau_2} & x^{(2)} & \xleftarrow{\tau_3} & x^{(3)} & \dots \\ & \in & & \in & & \in & & \in & \\ & X_0 & & X_1 & & X_2 & & X_3 & \end{array}$$

i.e. such that each $x^{(\ell)}$ is furthermore a valid configuration in the SFT X_ℓ. In other words, the corresponding limit configurations are obtained by an alternation of infinitely many substitution steps and validity conditions of finite type.

Such limit configurations can actually be obtained by a corresponding directive sequence of dill maps. More precisely, let V_ℓ be the neighborhood of the SFT X_ℓ, *i.e.* a finite subset of $\mathbb{Z}^d$ large enough to contain a family of forbidden patterns that define it; and define $\tau'_\ell\colon \mathcal{A}_\ell^{V_\ell} \rightrightarrows \mathcal{A}_{\ell-1}^{\mathfrak{R}_0}$ by:

$$\tau'_\ell(u) = \begin{cases} \tau(u_{\mathbf{0}}) & \text{if } u \text{ is locally valid in } X_\ell \\ \emptyset & \text{otherwise.} \end{cases}$$

[24] According to [ST15], "dill" comes from Deterministic Interactive Lindenmayer systems on Long words.

Then the configurations of X_0 obtained as limit configurations by the dill maps $(\tau'_\ell)_{\ell\geq 1}$ exactly correspond to the limit configurations obtained by the substitutions $(\tau_\ell)_{\ell\geq 1}$ in which all intermediate configurations are valid in their respective SFTs. In particular, dill maps of radius 1 allow to consider substitutions of valid Wang tilings!

Alternative limit spaces. As is usual with substitutions, one can actually define two possibly different limit spaces: for $\mathcal{D}$ a set of directive sequences, one can consider

$$\overleftarrow{X}_{\mathcal{D}} = \big\{x \in X : \exists (\tau_\ell)_{\ell\geq 1} \in \mathcal{D},\, \forall n \in \mathbb{N}, \exists (x^{(\ell)})_{0\leq \ell\leq n},\; x = x^{(0)} \xleftarrow{\tau_1} \dots \xleftarrow{\tau_n} x^{(n)}\big\},$$

i.e. the limit shift space (as defined above) whose configurations can be infinitely de-substituted along a directive sequence of $\mathcal{D}$; or the shift space in which we only consider subpatterns of iterated substitions of "single symbols"[25] from the alphabets $\mathcal{A}_n$'s:

$$\overleftarrow{X}^{\sqsubseteq}_{\mathcal{D}} = \big\{x \in X : \exists (\tau_\ell)_{\ell\geq 1} \in \mathcal{D},\, \forall w \sqsubseteq x, \exists n \in \mathbb{N}, \exists (x^{(\ell)})_{0\leq \ell\leq n},\; x^{(0)} \xleftarrow[\mathfrak{g}_1]{\tau_1} \dots \xleftarrow[\mathfrak{g}_n]{\tau_n} x^{(n)} \text{ and } w \sqsubseteq x^{(0)}|_{\tilde{\mathfrak{r}}_{\mathbf{0}}} \ \big(\text{where } (\tilde{\mathfrak{r}}_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{Z}^d} = \bigcirc_{\ell=1}^{n} \mathfrak{g}_\ell\big)\big\}.$$

In this article, we only prove the soficity of $\overleftarrow{X}_{\mathcal{D}}$ for some sets of directive sequences $\mathcal{D}$. The soficity of the associated $\overleftarrow{X}^{\sqsubseteq}_{\mathcal{D}}$ is usually harder to prove, and might depend on technical properties of the dill maps: we claim that, in later Theorems 6.7 and 6.8, additionally respecting the *Property A* from [Moz89] (see also [AS14, Section 3.1]) is sufficient to prove the soficity of the limit space $\overleftarrow{X}^{\sqsubseteq}_{\mathcal{D}}$.

§ 6.2. Local dill maps and computability

As a way to parallelize the computations of a dill map τ, we will distribute the computations $w \in \tau(u)$ over the individual cells of the image rectangles $\mathfrak{r} = \mathrm{dom}(w)$. In order to ensure that the concatenation of these independent subcomputations results in a valid image of τ, we define a notion of local dill map:

Definition 6.3 (Local dill map)**.** A dill map $\tau\colon \mathcal{A}^V \rightrightarrows \mathcal{B}^{\mathfrak{R}_0}$ is *local* if there exists some $\varphi\colon \mathfrak{R}_0 \times \mathbb{Z}^d \times \mathcal{A}^V \rightrightarrows \mathcal{B}$ such that:

$$\forall u \in \mathcal{A}^V,\; \tau(u) = \bigcup_{\mathfrak{r}\in\mathfrak{R}_0} \{w \in \mathcal{B}^{\mathfrak{r}} : \forall \boldsymbol{i} \in \mathfrak{r},\, w_{\boldsymbol{i}} \in \varphi(\mathfrak{r}, \boldsymbol{i}, u)\}.$$

Remark 6.4. While all deterministic computable dill maps can always be realized locally, notice that for any rectangle $\mathfrak{r} \in \mathfrak{R}_0$ and input $u \in \mathcal{A}^V$, the patterns $\{w \in \mathcal{B}^{\mathfrak{r}} : w \in \tau(u)\} = \prod_{\boldsymbol{i}\in\mathfrak{r}} \varphi(\mathfrak{r}, \boldsymbol{i}, u)$ are a product of independent pixels ranges. As a consequence, even the simple substitution

is not local. Nevertheless, since we are only interested in *sofic* shifts, it will often be possible during later applications to change the "work alphabet" of the dill maps to realize a non-local D-adic limit space using an alternative well-chosen D-adic representation.

In order to prove the soficity of a D-adic shift space, we will consider additional computational restrictions on our directive sequences of dill maps. More precisely:

Definition 6.5. A local dill map $\tau\colon \mathcal{A}^V \rightrightarrows \mathcal{B}^{\mathfrak{R}_0}$ is said to be *computable* if there exists a RAM program $e \in \{0,1\}^*$ computing its local map, *i.e.* such that:

$$\forall u \in \mathcal{A}^V,\; \tau(u) = \bigcup_{\mathfrak{r}\in\mathfrak{R}_0} \{w \in \mathcal{B}^{\mathfrak{r}} : \forall \boldsymbol{i} \in \mathfrak{r},\, w_{\boldsymbol{i}} \in \varphi_e(\mathfrak{r}, \boldsymbol{i}, u)\}.$$

Furthermore, it is *computable in time* $t \in \mathbb{N}$ if $\varphi_e(\mathfrak{r}, \boldsymbol{i}, u)\!\downarrow_{[t]} = \varphi_e(\mathfrak{r}, \boldsymbol{i}, u)$.

Remark 6.6. As is usual in computability theory, a code $e \in \{0,1\}^*$ computing a partial function $f \subseteq\colon A \rightrightarrows B$ is only required to compute correct images on elements from the domain $\mathrm{dom}(f)$. We do not make any assumption of the behavior of $\varphi_e(a)$ on elements $a \notin \mathrm{dom}(f)$: in particular, the computability of φ_e in time $t \in \mathbb{N}$ does not allow to deduce the domain of f in said time t.

[25] In the case of substitutions instead of dill maps, the second condition can be rewritten: $\forall w \sqsubseteq x, \exists n \in \mathbb{N}, \exists a \in \mathcal{A}_n, w \sqsubseteq \tau_0 \circ \cdots \circ \tau_n(a)$.

As is usual in computability theory, any statement operating on multiple computable objects requires some *uniformity conditions*, *i.e.* that a single program $e \in \{0,1\}^*$ can enumerate them all. To this end, we say that:

- A sequence of local dill maps $(\tau_\ell)_{\ell\in\mathbb{N}}$ is *computable in time* $t(\ell)$ if there exists a program $e \in \{0,1\}^*$ such that, for all $\ell \in \mathbb{N}$, the function $(\mathfrak{r}, \boldsymbol{i}, u) \mapsto \varphi_e(\ell, \mathfrak{r}, \boldsymbol{i}, u)\downarrow_{[t(\ell)]}$ is a local map of τ_ℓ;
- A sequence $(\mathcal{D}_\ell)_{\ell\geq 1}$ of sets $\mathcal{D}_\ell = (\tau_{\ell,i})_{i\in I_\ell}$ of local dill maps is *computable in time* $t(\ell)$ if there exists a program $e \in \{0,1\}^*$ such that, for all $\ell \in \mathbb{N}$ and $i \in I_\ell$, the function $(\mathfrak{r}, \boldsymbol{i}, u) \mapsto \varphi_e(\ell, i, \mathfrak{r}, \boldsymbol{i}, u)\downarrow_{[t(\ell)]}$ is a local map of $\tau_{\ell,i}$.

§ 6.3. Main results

The statement of the fixed point lemma (Lemma 5.2) is rather technical, and might thus be difficult to parse on a first reading. To alleviate this issue, we rather invite to consider the following two theorems: while they are actually immediate corollaries of our main lemma, their phrasing in terms of D-adic limit spaces should help to understand and contextualize them.

6.3.1. Square dill maps of computable sizes. For the first corollary, we weaken the possibilities on the grids of Lemma 5.2: instead of allowing substitutions of level ℓ to generate a large variety of grid sizes and shapes, we consider a square dill map τ_ℓ of fixed computable size $N_\ell \in \mathbb{N}$:

Theorem 6.7. *For $d \geq 2$, let $(\tau_\ell)_{\ell\geq 1}$ be a sequence of local growing dill maps $\tau_\ell \colon \mathcal{A}_\ell^{V_\ell} \rightrightarrows \mathcal{A}_{\ell-1}^{[\![N_\ell]\!]^d}$ for $(N_\ell)_{\ell\geq 1}$ a computable sequence of integers, and let $e \in \{0,1\}^*$ be a* log*-RAM program uniformly computing $(\tau_\ell)_{\ell\geq 1}$. For $X \subseteq \mathcal{A}_0^{\mathbb{Z}^d}$ a sofic shift, and denoting $\mathcal{D} = \{(\tau_\ell)_{\ell\geq 1}\}$ and $L_\ell = \prod_{i\leq\ell} N_i$, assume that:*

(1) For every $\ell \geq 1$, we have $2 \leq N_\ell \leq 2^{\mathcal{O}(L_\ell^\delta)}$ for some $\delta < \frac{d-1}{2}$;
(2) For every $\ell \geq 1$, symbols in the alphabet $\mathcal{A}_\ell$ have bit length $\mathcal{O}(L_{\ell-1}^\alpha)$ for some $\alpha < d-1$;
(3) For every $\ell \geq 1$, the neighborhood V_ℓ of the dill map τ_ℓ has radius $\mathcal{O}(\operatorname{polylog} L_{\ell-1})$;
(4) For every $\ell \geq 1$, the RAM program $\varphi_e(\ell, ...)$ computes the dill map τ_ℓ in time $\mathcal{O}(L_{\ell-1}^\alpha)$.

Then the D-adic limit space $\overleftarrow{X}_{\mathcal{D}}$ is a sofic shift space.

Proof. Fix $K \in \mathbb{N}$ a constant fixing all $\mathcal{O}(...)$, *i.e.* such that for every ℓ:

- The squares of level ℓ have size $N_\ell \leq 2^{KL_\ell^\delta}$;
- Symbols in $\mathcal{A}_\ell$ have bit length $K \cdot L_{\ell-1}^\alpha$;
- The neighborhood V_ℓ satisfies $V_\ell \subseteq \{-K\cdot(\log L_{\ell-1})^K, ..., K\cdot(\log L_{\ell-1})^K\}^d$;
- The RAM program $\varphi_e(\ell, ...)$ computes the dill map τ_ℓ in time $K \cdot L_{\ell-1}^\alpha$.

Furthermore, let $\langle N\rangle \in \{0,1\}^*$ be a total computable function computing the sequence $(N_\ell)_{\ell\geq 1}$. We then define a code $\langle\tau\rangle$ satisfying the hypotheses of Lemma 5.2 and such that $\overleftarrow{X}_{\langle\tau\rangle} = \overleftarrow{X}_{\mathcal{D}}$. Up to encoding into binary strings, we consider colors of the form $(\ell, (N'_i)_{i\leq\ell}, u)$, where:

- $\ell \in \mathbb{N}$ is an integer called the **level** of the tuple;
- $(N'_i)_{i\leq\ell}$ is a sequence of integers such that $2 \leq N'_i \leq 2^{K\cdot L_\ell'^\delta}$ (where $L'_\ell = \prod_{i=1}^\ell N'_i$), called the **history** of the tuple;
- $u \in \mathcal{A}_\ell^{\mathfrak{R}_0}$ is a pattern of domain $\operatorname{dom}(u) \subseteq \{-K\cdot(\log L'_{\ell-1})^K, ..., K\cdot(\log L'_{\ell-1})^K\}^d$, called the **pattern** of the tuple.

We then define the code $\langle\tau\rangle$ defining $\varphi_{\langle\tau\rangle}(\mathfrak{r}, \boldsymbol{i}, t) \rightrightarrows t'$ as the following algorithm substituting Wang tiles of $\mathcal{C}^{2d+1}$:

- First, denote ℓ, $(N'_i)_{i\leq\ell}$ and u the color of $f_{\blacklozenge}(t)$; check that $\mathfrak{r} = [\![N'_\ell]\!]^d$, that $\boldsymbol{i} \in \mathfrak{r}$, and check that the pattern $u \in \mathcal{A}_\ell^{\mathfrak{R}_0}$ has domain $V'_\ell = \{-K\cdot(\log L'_{\ell-1})^K, ..., K\cdot(\log L'_{\ell-1})^K\}^d$;
- (Consistency) Check that all facets $f_k^\pm(t)$ have the same **level** ℓ and **history** $(N'_i)_{i\leq\ell}$ as the decoration $f_{\blacklozenge}(t)$;
- (Higher block code) Check that, if $f_k^\pm(t)$ has **pattern** $v \in \mathcal{A}_\ell^{\mathfrak{R}_0}$, then $v = [u|_{V'_\ell \setminus f_k^-(V'_\ell)}]$,
- (Growth check) For every $i \leq \ell$, we check if $\varphi_{\langle N\rangle}(i)$ halts in less than ℓ steps of computation. If it does, but that $N_i \neq N'_i$, the algorithm $\langle\tau\rangle$ rejects;
- (Dill map) Run the computation $\varphi_e(\ell, \mathfrak{r}, \boldsymbol{i}, u)$ with output $a \in \mathcal{A}_{\ell-1}$;

- (Return) Non-deterministically choosing a pattern $v \in \mathcal{A}_{\ell-1}^{V'_{\ell-1}}$ such that $v_{\mathbf{0}} = a$, we return $(\ell - 1, (N'_i)_{i<\ell}, v)$.

Then, in a sequence of simulations $x^{(0)} \xleftarrow{\tau} \dots \xleftarrow{\tau} x^{(\ell)} \xleftarrow{\tau} \dots$ along a sequence of regular grids of size $(N'_\ell)_{\ell\geq 1}$, each substitution step $x^{(\ell+1)} \xrightarrow{\tau} x^{(\ell)}$ is computed in time $\mathcal{O}(L'^{\alpha}_{\ell} \cdot \operatorname{polylog} L'_\ell)$. Furthermore, if $\varphi_{\langle N\rangle}(i)$ terminates in some time $t \in \mathbb{N}$, then in any configuration $x^{(\ell)}$ such that $\ell \geq t$ will have a **history** field containing N_i. This concludes the proof. □

6.3.2. Multidimensional D-adic shift spaces with effective directive sequences. For the second corollary, we again weaken the growth of the grids in Lemma 5.2: instead of allowing growing substitutions, the sizes of the rectangles are allowed any non-deterministic rectangular shape, but bounded by a global constant $K \in \mathbb{N}$:

Theorem 6.8. *For $d \geq 2$ and $K \in \mathbb{N}$, let $(\mathcal{D}_\ell)_{\ell\geq 1}$ be computably uniform sets $\mathcal{D}_\ell = (\tau_{\ell,i})_{1\leq i\leq K}$ of local growing dill maps $\tau_{\ell,i} \colon \mathcal{A}_\ell^{*d} \rightrightarrows \mathcal{A}_{\ell-1}^{\mathfrak{N}_0}$ uniformly computed by a* log*-RAM program $e \in \{0,1\}^*$ and uniformly bounded by K, i.e. :*

(1) For any $\ell \geq 1$ and $i \in \{1, \dots, K\}$, the dill map $\tau_{\ell,i}$ has radius K and every image pattern $w = \tau_{\ell,i}(u)$ has bounded domain $\operatorname{dom}(w) \subseteq [\![K]\!]^d$;

(2) For any $\ell \geq 1$, symbols in the alphabet $\mathcal{A}_\ell$ have bit length $\mathcal{O}(2^{\ell\cdot\alpha})$ for some $\alpha < d - 1$;

(3) For any $\ell \geq 1$ and $i \in \{1, \dots, K\}$, $\varphi_e(\ell, i, \dots)$ computes the dill map $\tau_{\ell,i}$ in time $\mathcal{O}(2^{\ell\cdot\alpha})$.

Then for any sofic shift $X \subseteq \mathcal{A}_0^{\mathbb{Z}^d}$ and any computably co-enumerable set of directive sequences $\mathcal{D} \subseteq \prod_{\ell\geq 1} \mathcal{D}_\ell$, the D-adic limit space $\overleftarrow{X}_{\mathcal{D}}$ is also sofic.

Proof. This proof is actually quite similar to the proof of Theorem 6.7. With **level** field n, the **history field** now contains a sequence of indices $(i_\ell)_{\ell\leq n}$ representing the choice of the dill map τ_{ℓ,i_ℓ} used when substituting from level ℓ to $\ell - 1$.

Furthermore, the growth check is replaced by a directive sequence check. For $\langle D\rangle$ a program enumerating the forbidden prefixes of the computably co-enumerable set of directive sequences $\mathcal{D}$:

- (Directive sequence check) The finite sequence $(i_\ell)_{\ell\leq n}$ from the **history field** is checked against the first ℓ computation steps of $\varphi_{\langle D\rangle}$. □

6.3.3. Comments. Both Theorems 6.7 and 6.8 substantially weaken our main fixed point lemma (Lemma 5.2) in two different ways: the first one reduces the shapes that can appear to uniform squares of fixed computable sizes along a single sequence of dill maps $(\tau_\ell)_{\ell\geq 1}$; the second to arbitrary rectangles of bounded size along branching successions of dill maps from $\prod_{\ell\geq 1} \mathcal{D}_\ell$. In both cases, the main lost feature is the non-uniformity of a level: in Lemma 5.2, the possible grids at level ℓ can actually be of any sizes $\lambda(\mathfrak{g}_\ell)$ ranging in $2 \leq \lambda(\mathfrak{g}_\ell) \leq 2^{\mathcal{O}(\tilde{\mu}_\ell^{\delta})}$, thus defining a wide range of computational capacities for the next levels of substitutions. Nevertheless, these two theorems together should offer good insight into the potential applications of the fixed point lemma, and are general enough to recover all examples from Section 7.

Limit spaces. By considering dill maps instead of the more classical substitutions, the resulting limit space $\overleftarrow{X}_{\mathcal{D}}$ is obtained as an alternation of substitution steps and local validity checks, where the latter can ensure some SFT conditions (see Section 6.1).

Soficity. As mentioned in the introduction, patterns of domain $[\![n]\!]^d$ define in sofic shift spaces an information flow bounded by $\mathcal{O}(n^{d-1})$. Theorems 6.7 and 6.8, which prove the soficity of shifts with information flow $\mathcal{O}(n^\alpha)$ for $\alpha < d - 1$, can be understood as a partial converse statement.

More precisely, the dill maps τ_ℓ operate on alphabets $\mathcal{A}_\ell$ of increasing cardinality. As such, the size of the alphabet $\mathcal{A}_\ell$ provides a bound on the amount of information that τ_ℓ can process. As such, a sequence of dill maps $(\tau_\ell)_{\ell\geq 1}$ defines a recursive hierarchical scheme that verifies the validity of patterns in the configurations of the limit space. This defines a quantification of the amount of information appearing in the patterns of $\overleftarrow{X}_{\mathcal{D}}$, which is somewhat orthogonal to more classical complexity measures[26] (*e.g.* Kolmogorov complexity).

[26]For example, a full shift $X = \{0,1\}^{\mathbb{Z}^d}$ contains many patterns of maximal Kolmogorov complexity, but can be defined using dill maps on a blank alphabet $\mathcal{A} = \{\square\}$.

Relation to existing literature. The fixed point construction depends on a notion of local *simulation* between tilings [DRS12, Section 2.1] that is actually a form a (de)substitution. While the similarity between simulations and substitutions was already noticed in [Tör21], our fixed point lemma is – to the best of our knowledge – the first abstraction of the construction phrased in terms of substitutions and dill maps.

The fixed point lemma generalizes already existing applications of the fixed point construction in the following ways:

(1) The fixed point construction traditionally requires the sequence $\mu(\mathfrak{g}_\ell)_{\ell\geq 1}$ to increase rather substantially (for example, $\mu(\mathfrak{g}_\ell) = 2^{2^{2^\ell}}$ in [Des21]); and in particular never allows for constant-shaped substitutions of size, say, $\mu(\mathfrak{g}_\ell) = 2$.[27]
(2) Computability conditions on the configuration $x^{(\ell)}$ are classically defined in terms of $\mu(\mathfrak{g}_\ell)$ (the size of the ℓ^{th} step of substitution) instead of $\tilde{\mu}_\ell$ (the product of all previous sizes). The latter is obviously more flexible, and made necessary here in order to allow for constant and non-monotonous sequences $(\mu(\mathfrak{g}_\ell))_{\ell\geq 1}$.
(3) The fixed point construction is usually applied on $\mathbb{Z}^2$, where its computations embed the space-time diagrams of a (multitape) Turing machine [DRS12; Wes17] or cellular automaton [Zin16; Des21], on which programming – and, hence, precise considerations about time complexity – are quite difficult to achieve. In our lemma, we actually embed the space-time diagram of a *processor array*, and rely on Lemma 3.8 to state our result in the more intuitive setting of log-RAM machines.

The fixed point lemma is also a multidimensional soficity result on D-adic limit spaces. To the best of our knowledge, the only similar statement is a characterization of some S-adic shifts in [AS14]. For their main differences, our results apply to sequences of dill maps/substitutions of possibly infinite support (*i.e.* the sequence may range over infinitely many different dill maps) and *infinite alphabet rank* (*i.e.* the alphabets $\mathcal{A}_\ell$ may be of increasing and unbounded cardinality).

7. Applications

§ 7.1. Low density

Another, qualitatively different example comes from [Des21]:

Theorem 7.1 (Low Density Shift Spaces)**.** *For $\alpha < d-1$ a rational, and $\mathcal{A} = \{0,1\}$ the binary alphabet, let $X \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be an effective shift space such that for any configuration $x \in X$ and any pattern $w \sqsubseteq x$ of domain $\mathrm{dom}(w) = [\![n]\!]^d$, w contains less than $|w|_1 \leq n^\alpha$ symbols 1.*
Then X is an effective shift.

Proof. Since X is an effective shift space, there exists a log-RAM code $\langle X\rangle \in \{0,1\}^*$ enumerating the forbidden patterns of X such that for $t \in \mathbb{N}$, the computation $\varphi_{\langle X\rangle}(t)$ outputs a list of forbidden patterns whose domain is bounded by $[\![t]\!]^d$.

For $\ell \in \mathbb{N}$, we define the alphabet $\mathcal{A}_\ell$ as the lists $(\boldsymbol{p}_i)_{i\in I} \in (\mathbb{N}^d)^*$ of length at most $2^{\ell\cdot\alpha}$ whose positions $\boldsymbol{p}_i$ actually satisfy $\boldsymbol{p}_i \in [\![2^\ell]\!]^d$. We then define a dill map

$$\tau_{\ell+1} \colon \mathcal{A}_{\ell+1}^{\{-1,0,1\}^d} \mapsto \mathcal{A}_\ell^{[\![2]\!]^d}$$

whose local map is implemented by the following algorithm: on input $(\mathfrak{r}, \boldsymbol{i}, u)$ such that $\mathfrak{r} = [\![2]\!]^d$, $\boldsymbol{i} \in \mathfrak{r}$ and $u \in \mathcal{A}_{\ell+1}^{\{-1,0,1\}^d}$:

- From the pattern $u \in \mathcal{A}_{\ell+1}^{\{-1,0,1\}^d}$, define the set of merged positions $S \in (\mathbb{N}^d)^*$ as
$$S = \bigcup_{\boldsymbol{j}\in\{-1,0,1\}^d} \{\boldsymbol{p}_i + 2^{\ell+1}\cdot\boldsymbol{j} : \boldsymbol{p}_i \in u_{\boldsymbol{j}}\}.$$
- Let $\mathcal{F}_\ell = (w_f)_{f\in F}$ be a list of at most t forbidden patterns of domain $\mathrm{dom}(w_f) \subseteq [\![\ell]\!]^d$ computed by $\varphi_{\langle X\rangle}(\ell)$ in time ℓ.
- Then for every pattern $w_f \in \mathcal{F}_\ell$:

[27] Not including here the first author's PhD thesis [Cal25], upon which this article is built. Notice that, conversely, [Cal25, Theorem 10.11] only considers substitutions of size $\mu(\mathfrak{g}_\ell) = 2$.

  - And for every position $\boldsymbol{p} \in S$:
    - Check if w_f appears around the position $\boldsymbol{p}$;
    - If it does, reject. Otherwise, continue.
- Denoting $L = u_{\mathbf{0}} \in \mathcal{A}_\ell$, L is a set of positions in the rectangle $[\![2^{\ell+1}]\!]$: with input position $\boldsymbol{i}$, we compute the set of positions $L_{\boldsymbol{i}} \subseteq \mathbb{N}^d$ of positions in L
$$L_{\boldsymbol{i}} = \left\{\boldsymbol{p} - 2^\ell \cdot \boldsymbol{i} : \boldsymbol{p} \in L \text{ and } \boldsymbol{p} \in 2^\ell \cdot \boldsymbol{i} + [\![2^\ell]\!]^d\right\}.$$
Then $L_{\boldsymbol{i}}$ is actually an element of $\mathcal{A}_\ell$.
- We accept and return $L_{\boldsymbol{i}}$.

Notice that checking the appearance in the set S of an individual pattern of domain $[\![\ell]\!]^d$ around a position $\boldsymbol{p}$ can be computed in time $\mathcal{O}(\ell^d \cdot \log(3^d \cdot 2^{\alpha\cdot\ell})) = \mathrm{poly}(\ell)$ if the set S is implemented using an efficient data structure, such as a balanced binary tree. Thus, we deduce that the local map of $\tau_{\ell+1}$ can be computed in time $\mathcal{O}(2^{\alpha\cdot\ell} \cdot \mathrm{poly}(\ell))$ for some $\alpha < 1$. By Theorem 6.7, this concludes the proof. □

§ 7.2. S-adic shift spaces

We state the main results of [AS14] in their original presentation, and show how our results immediately generalize them.

Theorem 7.2 ([AS14, Theorem 5])**.** *Let $\mathcal{S}_0$ be a finite set of non degenerate multidimensional substitutions operating on some finite alphabet $\mathcal{A}$, and let $\mathcal{S} \subset \mathcal{S}_0^{\mathbb{N}}$ be effectively closed set of directive sequences. Then the limit space $\overleftarrow{X}_{\mathcal{S}}$ is sofic. Moreover, if $\mathcal{S}_0$ has property A, then the limit space $\overleftarrow{X}_{\mathcal{S}}^{\sqsubset}$ is also sofic.*

Proof. Since $\mathcal{S}_0$ is a finite set of substitutions on a finite alphabet $\mathcal{A}$, there exists an alternative alphabet $\mathcal{A}'$ and an alternative set of local substitutions $\mathcal{S}_0'$ realizing (up to a projection $\pi\colon \mathcal{A}' \to \mathcal{A}$) the same limit space $\overleftarrow{X}_{\mathcal{S}}$. We then conclude immediately by Theorem 6.8, in the special case where all the sets of dill maps $\mathcal{D}_\ell$ are finite and equal. □

This in turn allows us to recover famous classical theorem from the literature:

Theorem 7.3 ([Moz89])**.** *Let $\tau\colon \mathcal{A} \rightrightarrows \mathcal{A}^{\mathfrak{R}_0}$ be a bounded substitution. Then $\overleftarrow{X}_\tau$ is sofic.*

Proof. This is a special case of the previous theorem where $\mathcal{S}_0 = \{\tau\}$. □

More applications to follow soon...

References

[AA20] Shigeki Akiyama and Pierre Arnoux, eds. *Substitution and tiling dynamics: introduction to self-inducing structures*. Vol. 2273. Lecture Notes in Mathematics. Springer, 2020. DOI: 10.1007/978-3-030-57666-0.

[Akl85] Selim G. Akl. *Parallel Sorting Algorithms*. Vol. 12. Notes and Reports in Computer Science and Applied Mathematics. Academic Press, 1985. DOI: 10.1016/C2013-0-10281-4.

[AS14] Nathalie Aubrun and Mathieu Sablik. "Multidimensional effective S-adic subshift are sofic". In: *Uniform Distribution Theory* 9.2 (2014), pp. 7–29. URL: https://pcwww.liv.ac.uk/~karpenk/JournalUDT/vol09/no2/02AubrunSablick.pdf.

[AV79] Dana C. Angluin and Leslie G. Valiant. "Fast probabilistic algorithms for Hamiltonian circuits and matchings". In: *Journal of Computer and System Sciences* 18.2 (1979), pp. 155–193. DOI: 10.1016/0022-0000(79)90045-X.

[BD14] Valérie Berthé and Vincent Delecroix. "Numeration and substitutions 2012". In: vol. B46. RIMS Kôkyûroku Bessatsu. Research Institute for Mathematical Sciences, Kyoto University, 2014. Chap. Beyond substitutive dynamical systems: S-adic expansions, pp. 81–123. URL: http://hdl.handle.net/2433/226205.

[Ber64] Robert Berger. "The undecidability of the Domino problem". PhD thesis. Harvard University, 1964, pp. x+62. Published as "The undecidability of the Domino problem". In: *Memoirs of the American Mathematical Society* 66 (1966), pp. 1–72.

[Cal25] Antonin Callard. "Soficity of multidimensional subshifts". PhD thesis. Université de Caen, France, 2025. HAL: tel-05295331.

[Des21] Juline Destombes. "Étude de la complexité algorithmique et du caractère sofique des shifts en dimension deux". PhD thesis (in French). Université de Montpellier, France, 2021. arXiv: 2309.12241 [cs.IT]. URL: https://theses.fr/2021MONTS129.

[DLS08] Bruno Durand, Leonid A. Levin, and Alexander K. Shen. "Complex tilings". In: *The Journal of Symbolic Logic* 73.2 (2008), pp. 593–613. DOI: 10.2178/jsl/1208359062.

[DR21] Bruno Durand and Andrei E. Romashchenko. "The expressiveness of quasiperiodic and minimal shifts of finite type". In: *Ergodic Theory and Dynamical Systems* 41.4 (2021), pp. 1086–1138. DOI: 10.1017/etds.2019.112.

[DRS08] Bruno Durand, Andrei E. Romashchenko, and Alexander K. Shen. "Fixed point and aperiodic tilings". In: *DLT 2008 (Developments in Language Theory)*. Vol. 5257. Lecture Notes in Computer Science. 2008, pp. 276–288. DOI: 10.1007/978-3-540-85780-8_22.

[DRS10] Bruno Durand, Andrei E. Romashchenko, and Alexander K. Shen. "Effective closed subshifts in 1D can be implemented in 2D". In: *Fields of Logic and Computation*. Vol. 6300. Lecture Notes in Computer Science. 2010, pp. 208–226. DOI: 10.1007/978-3-642-15025-8_12.

[DRS12] Bruno Durand, Andrei E. Romashchenko, and Alexander K. Shen. "Fixed-point tile sets and their applications". In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 731–764. DOI: 10.1016/j.jcss.2011.11.001.

[Fog02] N. Pytheas Fogg. *Substitutions in dynamics, arithmetics and combinatorics*. Vol. 1794. Lecture Notes in Mathematics. Springer, 2002. DOI: 10.1007/b13861.

[Gác01] Péter Gács. "Reliable cellular automata with self-organization". In: *Journal of Statistical Physics* 103.1-2 (2001), pp. 45–267. DOI: 10.1023/A:1004823720305.

[Gác86] Péter Gács. "Reliable computation with cellular automata". In: *Journal of Computer and System Sciences* 32.1 (1986), pp. 15–78. DOI: 10.1016/0022-0000(86)90002-4.

[Han74] William Hanf. "Nonrecursive tilings of the plane. I". In: *The Journal of Symbolic Logic* 39.2 (1974), pp. 283–285. DOI: 10.2307/2272640.

[Hed69] Gustav A. Hedlund. "Endomorphisms and automorphisms of the shift dynamical system". In: *Mathematical Systems Theory* 3 (1969), pp. 320–375. DOI: 10.1007/BF01691062.

[Hoc09] Michael Hochman. "On the dynamics and recursive properties of multidimensional symbolic systems". In: *Inventiones Mathematicae* 176.1 (2009), pp. 131–167. DOI: 10.1007/s00222-008-0161-7.

[HS74] Juris Hartmanis and János Simon. "On the power of multiplication in random access machines". In: *SWAT 1974*. Cornell University, 1974, pp. 13–23. DOI: 10.1109/SWAT.1974.20.

[Kle38] Stephen Cole Kleene. "On notation for ordinal numbers". In: *Journal of Symbolic Logic* 3.4 (1938), pp. 150–155. DOI: 10.2307/2267778.

[LM95] Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge University Press, 1995, pp. xvi+495. DOI: 10.1017/CBO9780511626302.

[Moz89] Shahar Mozes. "Tilings, substitution systems and dynamical systems generated by them". In: *Journal d'Analyse Mathématique* 53.1 (1989), pp. 139–186. DOI: 10.1007/bf02793412.

[Mye74] Dale Myers. "Nonrecursive tilings of the plane. II". In: *The Journal of Symbolic Logic* 39.2 (1974), pp. 286–294. DOI: 10.2307/2272641.

[Ram23] Firas Ben Ramdhane. "Symbolic dynamical systems in topological spaces defined via edit distances". PhD thesis. Université de Marseille, France, 2023. HAL: tel-04146898.

[Rog67] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT Press, 1967. ISBN: 978-0262680523.

[ST15] Ville Salo and Ilkka Törmä. "Block maps between primitive uniform and Pisot substitutions". In: *Ergodic Theory and Dynamical Systems* 35.7 (2015), pp. 2292–2310. DOI: 10.1017/etds.2014.29.

[TK77] Clark D. Thompson and Hsiang-Tsung Kung. "Sorting on a Mesh-Connected Parallel Computer". In: *Communications of the ACM (Association for Computing Machinery)* 20.4 (1977), pp. 263–271. DOI: 10.1145/359461.359481.

[Tör21] Ilkka Törmä. "Fixed point constructions in tilings and cellular automata". In: *AUTOMATA 2021 (International Workshop on Cellular Automata and Discrete Complex Systems)*. Vol. 90. Open Access Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 4:1–4:13. DOI: 10.4230/oasics.automata.2021.4.

[Wes17] Linda Brown Westrick. "Seas of squares with sizes from a Π^0_1 set". In: *Israel Journal of Mathematics* 222.1 (2017), pp. 431–462. DOI: 10.1007/s11856-017-1596-6.

[Zin16] Charalampos Zinoviadis. "Hierarchy and expansiveness in two-dimensional subshifts of finite type". PhD thesis. *Turun yliopisto*, Finland, 2016. arXiv: 1603.05464 [math.DS].

[Сли78] Анатоль Олесьевич Слисенко. «Методы вычислений, основанные на адресной организации памяти». В: *Всесоюзн. симпозиум « Искусств, интеллект и автоматизация исследова ний в матем. » Тезисы докладов и сообщений*. Інститут кібернетики, Киев, 1978, с. 94—96.

ENS de Lyon, UCBL, CNRS, Inria, LIP; F-69342 Lyon Cedex 07, France
Email address: contact@acallard.net

ENS de Lyon, UCBL, CNRS, Inria, LIP; F-69342 Lyon Cedex 07, France
Email address: mail@lpaviets.org

Université Caen Normandie, ENSICAEN, CNRS, Normandie Univ, GREYC; F-14000 Caen, France
Email address: pascal.vanier@unicaen.fr