Automatic Graphs and D0L-Sequences of Finite Graphs*

Olivier Ly†

Advanced Research – Smart Card & Terminals, SchlumbergerSema
University Bordeaux I, LaBRI

Version: May 14, 2003

The notion of end is a classical mean to understand the behavior of a graph at infinity. In this respect, we show that the problem of deciding whether an infinite automatic graph has more than one end is recursively undecidable. The proof involves the analysis of some global topological properties of the configuration graph of a self-stabilizing Turing machine. This result is applied to the graph transformation system theory: It allows us to set a picture of decidability issues concerning logic of $D0L$-languages of finite graphs. We first show that the first-order theory of a $D0L$-sequence of finite graphs is decidable. Secondly, we show that the first-order theory, when it is upgraded with a closure operator, becomes undecidable for $D0L$-sequences of finite graphs.

Key Words: Infinite graphs, graph transformations, automata, logic.

*This article is a full and enhanced version of [27]

†e-mail: olivier.ly@louveciennes.sema.slb.com

INTRODUCTION

The concept of *automatic graph* [5, 23, 31, 37] intends to define infinite graphs in a constructive way in terms of finite state automata. The set of vertices of such a graph is the quotient of a rational language by an equivalence relation which is itself described by a finite state letter-to-letter automaton recognizing pairs of words. Two vertices are connected by an edge if any pair of representatives of them is recognized by some fixed finite state letter-to-letter automaton recognizing pairs of words.

To understand how these graphs fit in infinite graph theory, let us mention that *equational graphs* in the sense of [9] are automatic. One can actually be more precise seeing that the deterministic automatic graphs are definable up to isomorphism by weak monadic second-order logical formulæ (cf. [37]). Indeed, this fact together with results of [9], implies that deterministic automatic graphs of bounded tree-width are equational (see [32] for the definition of tree-width). Therefore, the deterministic automatic graphs of bounded tree-width are exactly the deterministic equational graphs. Let us note that this implies that the deterministic automatic graphs of bounded tree-width and of bounded degree are exactly the *context-free* graphs in the sense of [29]. But automatic graphs are not of bounded tree-width in general, neither of bounded degree. In this respect, let us mention that *transition graphs* in the sense of [6] are automatic. In other respects, automatic graphs are instances of *automatic relational structures* which have been studied in [37, 31, 23, 4] from the point of view of model theory. In this respect, let us mention the result of [23] according to which the first-order theory of an automatic logical structure is decidable (see [31, 4] for extensions and variations).

Here we deal with the notion of *end* of an infinite graph which is intended to capture the concept of "way to infinity" (see [1, 13, 20, 28]). In this respect, we prove the following result:

THEOREM 1. *The problem of deciding whether an automatic graph has more than one end is recursively undecidable.*

This result contributes in clearing up the decidability boundary of the problem of determining the number of ends of an infinite graph. In the case of the Cayley graphs of context-free groups in the sense of [29], it was proved in [38] that the number of ends is computable. The group structure hypothesis is actually useless for this result: One can show that the space of ends of a *context-free graph* in the sense of [29] is homeomorphic to the boundary of a computable rational language. Hence, the number of ends of a context-free graph is effectively computable (cf. [26]).

Theorem 1 applies to two domains: Combinatorial group theory and graph transformation systems.

Concerning graph transformation systems, it allows us to set a picture of decidability issues concerning logic of graph substitution systems, i.e. $D0L$-languages of finite graphs. To do that, we introduce a connection between graph $D0L$-systems and automatic graphs: $D0L$-sequences of finite graphs, i.e. the sequences of finite graphs produced by iterating a deterministic graph $D0L$-systems, can be described in terms of finite state automata by automatic presentations. More precisely, any $D0L$-sequence of finite graphs actually is the sequence of the spheres of an automatic graph. The converse is true for a large class of automatic graphs to be defined in the text. Firstly, this fact together with the decidability of the first-order theories of automatic graphs (see [23]), implies the following fact:

THEOREM 2. *The first-order theory of a $D0L$-sequence of finite graphs is decidable.*

As we will see in the text, the graph $D0L$-systems to be considered here generalize vertex replacement systems [11, 15, 10], of which monadic second-order theories are decidable. One can then ask how further Theorem 2 can be extended. In this respect, we

will see that graph $D0L$-systems can generate arbitrary large squared grids, which implies that monadic second-order theories of $D0L$-sequences of graphs are undecidable (see e.g. [10, Prop. 5.2.2]). Theorem 1 actually allows us to give a more precise answer to this question: It turns out to be equivalent to the following result, which implies that the satisfiability problem of first-order formulæ upgraded with the closure operator is undecidable for $D0L$-sequences of finite graphs.

THEOREM 3. *The problem of deciding whether all the finite graphs produced by iterating a graph $D0L$-system are connected is recursively undecidable.*

In other respects, the question of determining the number of ends of an automatic graph arises through the research of algorithms in combinatorial group theory. Let us note that the concept of automatic graph actually comes from the notion of automatic groups (see [16] for basics about it) which is by now classical. Indeed, the property of being automatic for a finitely generated group can be directly expressed as a property of its Cayley graph. In fact, an automatic presentation for a group basically defines its Cayley graph in terms of finite state automata. The concept of automatic graph consists in considering automatic presentations as defining infinite graphs which are not necessarily Cayley graphs of some groups, dropping the symmetry properties implied by the group structure hypothesis.

By Stallings's Theorem [39, 13], the computation of the number of ends of a group, i.e. of its Cayley graph, is a crucial step towards its decomposition as amalgamed products or HNN-extensions over finite subgroups. The study of effectiveness of Stallings's Theorem led to a decomposition algorithm for context-free groups in [38]. This algorithm can actually be extended to word hyperbolic groups in view of the recent result of [17] according to which the number of ends of such a group can be effectively computed. Note that context-free groups are word hyperbolic and that word hyperbolic groups are automatic [16, 18]. A computer program was developed to compute the number of ends of an automatic group from an automatic presentation [7]. However, there is no known halting proof for this program; and identifying one-ended automatic groups in an effective way remains an open problem. Theorem 1 states that this task is not possible if one drops the group structure hypothesis.

The paper is organized as follows:
Section 1.1 deals with the presentation of the automatic graph concept. Besides the definition, several examples are provided. And in order to see how automatic graphs fit in infinite graph theory, we also provide proofs that equational graphs and transition graphs are automatic (Section 1.2). The main tool of the proof of Theorem 1 is the *configuration graph* of a Turing machine, to be defined in Section 2.2. We show how this graph can be equipped with an automatic presentation, which is deserved in [5] as well. As a direct consequence of that, we get that connectivity of automatic graphs is undecidable. Basics about the concept of end of infinite graphs are then given in Section 2.1. We will see that the property of having one and only one end can be expressed by a counting monadic second-order formula. Section 2.2 is devoted to the proof of Theorem 1. It involves a construction inspired by the notion of *self-stabilizing* algorithm. The aim is to get a machine which simulates the computation of a given Turing machine, and which can detect by itself that it is in a "wrong" configuration, i.e. a configuration which cannot be reached from the initial one. This property allows a control of the global shape of the configuration graph, which is needed to deal with its ends. The proof of Theorem 1 is a reduction of the Turing machine halting problem. Starting with a given Turing machine, one considers the associated self-stabilizing Turing machine, and a slight variation of the configuration graph of this last one. Lemma 10 states that this graph has exactly one end if and only if the initial Turing machine does not halt, which leads directly to Theorem 1. In some sense, the use of a self-stabilizing machine is explained by the fact that having only one end is a global property of a graph. And, in order to achieve

such a property for the configuration graph of a Turing machine, one needs to control the behavior of the machine on any instantaneous descriptions and not only on those produced by the computation from the initial state. Note that an automatic presentation cannot distinguish the instantaneous descriptions coming from the initial one from the other ones. Section 3 establishes the relationship between automatic graphs and graph $D0L$-systems. Graph $D0L$-systems are introduced in Section 3.1. In Section 3.2, we show how the sequence of finite graphs obtained by iterating a graph $D0L$-system can be seen as the sequence of the spheres of an automatic graph, and conversely. Firstly, we show how this fact, together with the decidability of first-order theory of automatic graphs, drives to Theorem 2. We then show that Theorem 1 implies that the simultaneous connectivity of all the elements of such sequences is undecidable, i.e. Theorem 3. Finally, Section 3.3 deals with the relationships between graph $D0L$-systems and other graph transformation formalism. First, we show that graph $D0L$-systems emulate vertex replacement systems (see e.g. [15]). Second, we show that graph $D0L$-systems are emulated by edge grammars (see [3]).

ACKNOWLEDGMENTS

The author is greatly indebted to D. B. A. Epstein for suggesting the problem and for many helpful discussions. The author also wants to thank G. Sénizergues for pointing out the notion of self-stabilizing machine and also P. Narbel for improving Section 3.

1. AUTOMATIC GRAPHS

1.1. Definition

We deal with *labeled directed graphs* as relational structures whose relations are all of arity 1 or 2 (see e.g. [9]). Formally, that are tuples of form $(V, L, \{R_\ell\}_{\ell \in L})$ where V is the set of vertices; L is the finite set of labels. And for each $\ell \in L$, $R_\ell \subset V$ or $R_\ell \subset V \times V$. The R_ℓ's which are subsets of V are supposed to be pairwise disjoint; they encode the vertex labeling mappings: a given vertex is ℓ-labeled for some $\ell \in L$ if and only if it belongs to R_ℓ. Edges are defined according to the R_ℓ's which are subsets of $V \times V$: two vertices v_1 and v_2 are connected by an ℓ-labeled edge if and only if (v_1, v_2) belongs to R_ℓ. As the R_ℓ's which are subsets of $V \times V$ are not supposed to be pairwise disjoint, two given vertices can be connected by several edges of distinct labels.

A *non directed path* in such a graph is a finite sequence of vertices $v_1...v_n$ such that for each $i = 1, ..., n-1$, there is an edge from v_i to v_{i+1} or from v_{i+1} to v_i; such a path is said to be *directed* if for each $i = 1, ..., n-1$, there is an edge from v_i to v_{i+1}. We consider the *traces* of paths which are some words over $L \cup \overline{L}$ where $\overline{L}$ is an *inverse alphabet* of L; it is defined as follows: $L \cap \overline{L} = \emptyset$ and $\overline{L}$ is given with a one-to-one mapping $\tau : L \to \overline{L}$; for each $\ell \in L \cup \overline{L}$, let $\overline{\ell}$ denote $\tau(\ell)$ if $\ell \in L$ and $\tau^{-1}(\ell)$ if $\ell \in \overline{L}$. A *trace* of a path $v_1...v_n$ is a word $\ell_1...\ell_{n-1}$ such that for each $i = 1, ..., n-1$, if $\ell_i \in L$, then there is a ℓ_i-labeled edge from v_i to v_{i+1}, and if $\ell_i \in \overline{L}$, then there is an $\overline{\ell_i}$-labeled edge from v_{i+1} to v_i.

For basics about formal languages and automata theory, the reader is refered to [22]. The concept, to be used here, of finite state automata recognizing directed pairs of words over an alphabet A, i.e. letter-to-letter automata, is presented in [16, Chapter 1] (see also [23, 4]). Such a device consists in a finite state automaton over the alphabet $(A \cup \{\$\}) \times (A \cup \{\$\})$, where \$ is a *end-of-string* or *padding* symbol not belonging to A, whose language consists in elements of $A^*.\$^* \times A^*.\$^*$ of the form (w_1, w_2) with $|w_1| = |w_2|$ and $w_1 \in A^*$ or $w_2 \in A^*$. By abuse of notations, here we consider the subset of $A^*.\$^* \times A^*.\$^*$ which consists in pairs of words of the same length as a subset of $((A \cup \{\$\}) \times (A \cup \{\$\}))^*$. We say by extension that such a device recognizes a directed pair of words $(u, v) \in A^* \times A^*$ if it actually recognizes $(u.\$^{\max\{0,|v|-|u|\}}, v.\$^{\max\{0,|u|-|v|\}})$.

DEFINITION 1 (Automatic Graphs).
A possibly infinite graph $\Gamma = (V, L, \{R_\ell\}_{\ell \in L})$ is said to be *automatic* if there exists a tuple $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ where

- W is a finite state automaton over a finite alphabet A. And $\nu : \mathcal{L}(W) \to V$ is a surjective application. $\mathcal{L}(W)$ denotes the language which W recognizes.
- M_0 is a letter-to-letter automaton recognizing pairs of words of A^* such that for any two words w_1 and w_2 of $\mathcal{L}(W)$, $\nu(w_1) = \nu(w_2)$ if and only if $(w_1, w_2) \in \mathcal{L}(M_0)$.
- Let $\ell \in L$. If the arity of R_ℓ is equal to 1, then M_ℓ is an automaton over A such that for any $w \in \mathcal{L}(W)$, $\nu(w) \in R_\ell$ if and only if $w \in \mathcal{L}(M_\ell)$. If the arity of R_ℓ is equal to 2, M_ℓ is a letter-to-letter automaton over A such that for any two words $w_1, w_2 \in \mathcal{L}(W)$, $(\nu(w_1), \nu(w_2)) \in R_\ell$ if and only if $(w_1, w_2) \in \mathcal{L}(M_\ell)$.

$(\nu, W, M_0, (M_\ell)_{\ell \in L})$ is called an *automatic graph presentation* of Γ.

EXAMPLE 1. The concept of automatic graph can be illustrated by the example of the infinite square grid.

Let us consider the graph G represented in Figure 1, i.e. the infinite square grid, defined as follows:

- The set of vertices of G is $V_G = \mathbb{Z} \times \mathbb{Z}$.
- The set of labels is $L_G = \{a, b\}$; and edges are defined by the following binary relations (vertices are not labeled):

$$R_a^G = \{((n, m), (n+1, m)) \mid n, m \in \mathbb{Z}\}$$

$$R_b^G = \{((n, m), (n, m+1)) \mid n, m \in \mathbb{Z}\}$$

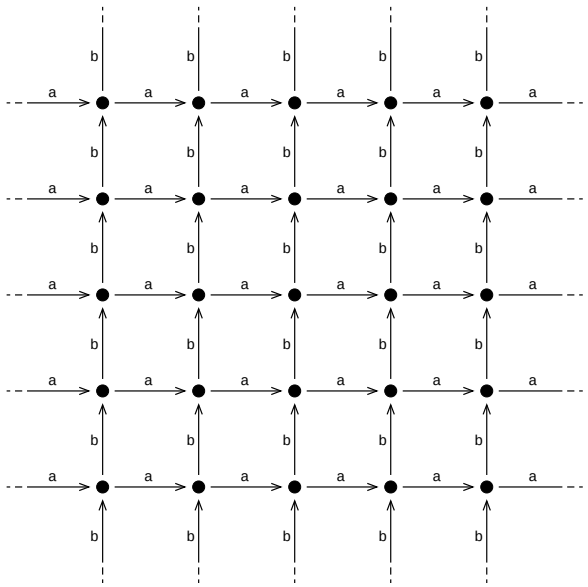


FIG. 1 The infinite square grid is an automatic graph

Now, let us consider the automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ for G defined as follows: We shall not define the automaton W but only the language it recognizes:

its alphabet is $L_G \cup \{\bar{a}, \bar{b}\} = \{a, \bar{a}, b, \bar{b}\}$ and $\mathcal{L}(W)$ is defined by the following rational expression:

$$(a^* + \bar{a}^*)(b^* + \bar{b}^*).$$

In a natural way, the mapping $\nu : \mathcal{L}(W) \to V_G$ is defined to be the morphism of monoïd such that $\nu(a) = (1, 0)$, $\nu(\bar{a}) = (-1, 0)$, $\nu(b) = (0, 1)$ and $\nu(\bar{b}) = (0, -1)$, when considering $V_G = \mathbb{Z} \times \mathbb{Z}$ provided with the term by term addition. One gets that for all $n, m \in \mathbb{N}$, $\nu(a^n b^m) = (n, m)$, $\nu(a^n \bar{b}^m) = (n, -m)$, $\nu(\bar{a}^n b^m) = (-n, m)$, $\nu(\bar{a}^n \bar{b}^m) = (-n, -m)$. Let us note that the mapping ν is one-to-one; this leads us to define

$$\mathcal{L}(M_0) = \{(w, w) \mid w \in \mathcal{L}(W)\}$$

The alphabet of the automata M_a and M_b is $(\{a, \bar{a}, b, \bar{b}\} \cup \{\$\}) \times (\{a, \bar{a}, b, \bar{b}\} \cup \{\$\})$. We shall not describe them but the languages they recognize.
$\mathcal{L}(M_a)$ is defined by the following rational expression:

$$(a, a)^*((\$, a) + (b, a)(b, b)^*(\$, b) + (\bar{b}, a)(\bar{b}, \bar{b})^*(\$, \bar{b}))$$

$$+(\bar{a}, \bar{a})^*((\bar{a}, \$) + (\bar{a}, b)(b, b)^*(b, \$) + (\bar{a}, \bar{b})(\bar{b}, \bar{b})^*(\bar{b}, \$))$$

And $\mathcal{L}(M_b)$ is defined by the following rational expression:

$$((a, a)^* + (\bar{a}, \bar{a})^*)((b, b)^*(\$, b) + (\bar{b}, \bar{b})^*(\bar{b}, \$))$$

Let us look at the correspondence between M_a and R_a^G. The rational expression defining the language recognized by M_a can be decomposed as follows:

$$(a, a)^*(\$, a) + (a, a)^*(b, a)(b, b)^*(\$, b) + (a, a)^*(\bar{b}, a)(\bar{b}, \bar{b})^*(\$, \bar{b})$$

$$+(\bar{a}, \bar{a})^*(\bar{a}, \$) + (\bar{a}, \bar{a})^*(\bar{a}, b)(b, b)^*(b, \$) + (\bar{a}, \bar{a})^*(\bar{a}, \bar{b})(\bar{b}, \bar{b})^*(\bar{b}, \$).$$

The first term of this expression addresses the edges of the form $((n, 0), (n + 1, 0))$ with $n \geq 0$. The second one addresses the edges of the form $((n, m), (n + 1, m))$ with $n \geq 0$ and $m > 0$. The third one addresses the edges of the form $((n, m), (n + 1, m))$ with $n \geq 0$ and $m < 0$. The fourth one addresses the edges of the form $((n - 1, 0), (n, 0))$ with $n \leq 0$. The fifth one addresses the edges of the form $((n - 1, m), (n, m))$ with $n \leq 0$ and $m > 0$. And the sixth one addresses the edges of the form $((n - 1, m), (n, m))$ with $n \leq 0$ and $m < 0$. This shows that the a-labeled edges of G are actually those that M_a recognizes. A similar analysis applies to edges labeled by b.

Remark 1. Though any finite graph is obviously automatic, an automatic graph is in general infinite. However, the description of a graph by an automatic graph presentation is *constructive* in the following sense: Let us consider an automatic graph provided with an automatic graph presentation. We can get a set of unique representatives of vertices as a rational language: Given a total ordering of A, let us consider the *shortlex* ordering of $\mathcal{L}(W)$ (see [22, 16]). Each equivalence class of $\mathcal{L}(W)$ according to $\mathcal{L}(M_0)$ has a unique minimal word according to this ordering which is called the shortlex minimal representative of the class in question. The language of shortlex minimal representatives is rational (see [16]).

Definition 2 (Regularly Accessible Automatic Graphs).
A *regularly accessible automatic graph* is an automatic graph G for which there exists an automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ such that

- $\mathcal{L}(W)$ is a prefix closed subset of A^*.
- For each $w \in \mathcal{L}(W)$ and $a \in A$ such that $w.a \in \mathcal{L}(W)$, there is an a-labeled edge in G from $\nu(w)$ to $\nu(w.a)$.

Edges of the previous type are said to be *radial*, the other ones are said to be *transversal*. Note that these terms get some geometric justification when for instance the regular equivalence relation used to define vertices only associates pairs of words of the same length and transversal edges only connect pairs of vertices represented by words of the same length.

EXAMPLE 2. The concept of regularly accessible automatic graph can be illustrated by a variation of Example 1.

First, let us note that the graph G described in Example 1 is not regularly accessible. Indeed, if it was, then there would exist a vertex which all the other vertices would be accessible from via a directed path of radial edges. This is not true: for each vertex (n, m) of G, there are vertices that are not accessible from it, for instance $(n-1, m)$.

So, let us consider a new graph G' obtained from the graph G of Example 1 by adding some new edges. Its definition is as follows.

- $V_{G'} = V_G$.
- $L_{G'} = \{a, \bar{a}, b, \bar{b}\}$. $R_a^{G'}$ and $R_b^{G'}$ respectively are equal to R_a^G and R_b^G, which are defined in Example 1. And we add two new binary relations defining edges labeled by $\bar{a}$ and $\bar{b}$:

$$R_{\bar{a}}^{G'} = \{((n, m), (n-1, m)) \mid n, m \in \mathbb{Z}\}$$
$$R_{\bar{b}}^{G'} = \{((n, m), (n, m-1)) \mid n, m \in \mathbb{Z}\}$$

Actually, $\bar{a}$-labeled edges and $\bar{b}$-labeled edges respectively are the inverses of a-labeled edges and b-labeled edges.

G' is represented in Figure 2.

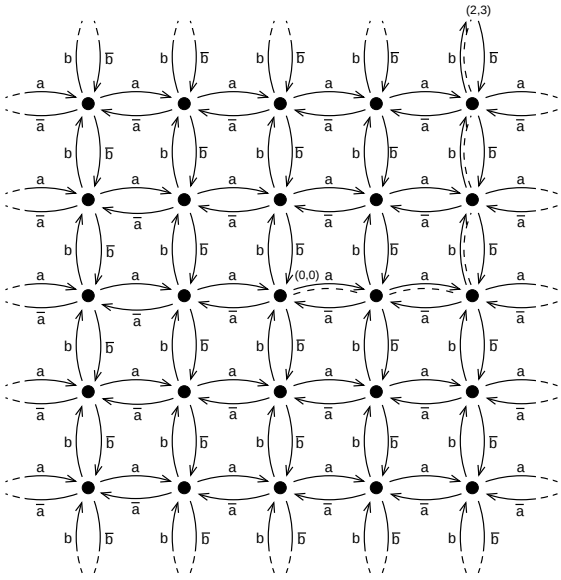


FIG. 2 The infinite square grid as a regularly accessible automatic graph. The dashed line represents a directed path of radial edges from the vertex $(0, 0)$ to the vertex $(2, 3)$.

The automatic presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ for G' is defined from those of G defined in Example 1 by adding the automata associated to $R_{\bar{a}}^{G'}$ and $R_{\bar{b}}^{G'}$. As in Example 1, we actually shall not describe them but the languages they recognize.

$\mathcal{L}(M_{\bar{a}})$ is defined by the following rational expression:

$$(a,a)^*((a,\$) + (a,b)(b,b)^*(b,\$) + (a,\bar{b})(\bar{b},\bar{b})^*(\bar{b},\$))$$
$$+(\bar{a},\bar{a})^*((\$,\bar{a}) + (b,\bar{a})(b,b)^*(\$,b) + (\bar{b},\bar{a})(\bar{b},\bar{b})^*(\$,\bar{b}))$$

$\mathcal{L}(M_{\bar{b}})$ is defined by the following rational expression:

$$((a,a)^* + (\bar{a},\bar{a})^*)((b,b)^*(b,\$) + (\bar{b},\bar{b})^*(\$,\bar{b})).$$

This automatic graph presentation actually defines G', and actually fulfills the conditions of Definition 2.

Remark 2. The concepts of automatic graph and regularly accessible automatic graph are generalizations of Cayley graphs of automatic groups. This provides a lot of examples of automatic graphs, for instance, free groups and free abelian groups. To this respect, let us note that the graph defined in Example 1 is the Cayley graph of $\mathbb{Z} \times \mathbb{Z}$, the free abelian group of rank 2. Let us also mention word hyperbolic groups which also are automatic.

But to some extends, regularly accessible graphs are in between the automatic graphs and the Cayley graphs of automatic groups (see [5]): Let us consider an automatic finitely generated group G together with a finite set of generators S. Then it is classical to consider the surjective group morphism from the free group over S onto G. And this morphism can be extended into a graph morphism between the respective Cayley graphs. This property has a counterpart in the framework of regularly accessible automatic graphs: Let us be given a regularly accessible automatic graph Γ together with an automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ as in Definition 2. Then the mapping $\nu : \mathcal{L}(W) \to V_G$ can be extended into a vertex-surjective graph morphism from a regular tree, i.e. a tree with finitely many sub-trees up to isomorphism, into Γ in the following way: let us consider the tree $T_{\mathcal{L}(W)}$ associated to $\mathcal{L}(W)$, i.e. the tree whose vertices are the words of $\mathcal{L}(W)$ and whose edges are defined as follows: a word w_1 is connected to a word w_2 by a a-labeled edge if and only if $w_2 = w_1.a$. Any prefix-closed regular language gives rise to such a tree which is regular. Then $\nu : \mathcal{L}(W) \to V_\Gamma$ maps nodes of $T_{\mathcal{L}(W)}$ onto vertices of Γ. And this mapping actually is a graph morphism. Let us note that the radial edges are the edges which are reached by this morphism; the transversal edges are the other ones. This graph morphism is the counterpart in the regularly accessible automatic graph framework of the surjective morphism of the free group over S, whose Cayley graph obviously is a regular tree, onto G. Let us note that automatic graphs can not have such a property seeing that they possibly are not connected.

1.2. Constructive Infinite Graphs

This section is devoted to the relationship between the family of automatic graphs and other families of constructive graphs. We shall see that *context-free* graphs, *equational graphs* and *transition* graphs are all automatic.

Context-free Graphs.

The context-free graphs, introduced in [29], are the graphs of configurations of pushdown automata. The vertices of such a graph are the configurations of a given pushdown automaton and its edges represent the transitions of the automaton. Let us be given a pushdown automaton; a given configuration consists of the state which the automaton is in together with the stack content. It can be encoded by a word obtained by concatenating a letter encoding the state to a word representing the content of the stack. On the one hand, the language of words encoding configurations is obviously rational; it has the form $S.A^*$ where S is the set of states of the automaton and A is its stack alphabet. On the other hand, the transitions between configurations of the automaton can be computed by a letter-to-letter automaton from the words which encode them. So, *context-free graphs are automatic.*

Equational Hypergraphs.

The concept of equational hypergraph extends those of context-free graphs. It has been introduced in [8].

A *hypergraph*[1] is a tuple of the form $(V, L, \{R_\ell\}_{\ell \in L})$ where V is the set of vertices; L is the finite set of labels. And for each $\ell \in L$, R_ℓ is a relation over V, i.e. a subset of V^n for some strictly positive integer n. An element of R_ℓ of form $(v_1, ..., v_n)$ defines a *hyperedge* whose vertex list is $v_1, ..., v_n$; n is called the arity of the hyperedge. If the v_i are pairwise distinct, the hyperedge is said to be *simple*. Graphs, as defined in Section 1.1, are the hypergraphs whose hyperedges are all of arity 1 or 2. Here we actually deal with hypergraphs with *sources* which generalize pointed graphs; such a hypergraph is a hypergraph provided with a finite ordered list of pairwise distinct vertices. These distinguished vertices are called the sources of the hypergraph, the other ones are said to be internal. The number of sources of a hypergraph is called its *type*.

An equational hypergraph[2] is defined according to a deterministic *hyperedge replacement graph grammar* (see [14]). Such a grammar is defined according to a finite set of symbols L which is divided into two subsets L_1 and L_2 whose elements are respectively called *terminal* symbols and *non-terminal* symbols. A deterministic hyperedge replacement grammar is a tuple $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ where $\ell_0 \in L_2$ is the initial symbol and for each $\ell \in L_2$, H_ℓ is a finite L-labeled hypergraph with source, such that for each $\ell \in L$, the arities of all the ℓ-labeled hyperedges of any $H_{\ell'}$ are equal; and if $\ell \in L_2$, this arity must be equal to the type of H_ℓ.

The construction of the equational hypergraph associated to such a grammar takes several steps. First, for each $\ell \in L_2$, let us denote by $e_{\ell 1}, ..., e_{\ell n_\ell}$ the non-terminal hyperedges of H_ℓ, i.e. the hyperedges labeled by non-terminal symbols. Then, let us consider the ordered regular tree t, whose nodes are labeled over L_2, constructed inductively as follows: the root of t is labeled by ℓ_0; and for each $\ell \in L_2$, each ℓ-labeled node of t has exactly n_ℓ sons and for $i = 1...n_\ell$, its i-th son has the same label than $e_{\ell i}$. Eventually the equational hypergraph Γ associated to the grammar $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ is defined as follows: for each $\ell \in L_2$ and each ℓ-labeled node μ of t, let us consider the hypergraph H_μ obtained from a copy of H_ℓ by deleting all its non-terminal hyperedges. Let $\overline{\Gamma} = \bigcup_\mu H_\mu$ be the disjoint union of the H_μ's. Let us consider the binary relation R over vertices of $\overline{\Gamma}$ defined as follows: for each pair (x, x') of vertices of $\overline{\Gamma}$, $(x, x') \in R$ if and only if there exist two nodes μ and μ' of t such that μ' is the j-th son of μ for some j, $x \in H_\mu$, $x' \in H_{\mu'}$ and there exists i such that x corresponds to the i-th vertex of $e_{\ell j}$ in H_ℓ, where ℓ denotes the label of μ, and x' is the i-th source of $H_{\mu'}$. Then the equational hypergraph Γ associated to $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ is defined as a quotient of $\bigcup_\mu H_\mu$ by the transitive symmetric closure of R. For each vertex x of $\overline{\Gamma}$, the vertex of Γ represented by x shall be denoted by $\nu(x)$. The sources of Γ are defined to be those of H_{μ_0} where μ_0 denotes the root of t.

Remark 3. Equational graphs can also be defined as inductive limits of some sequences of finite graphs. These sequences of finite graphs are defined in terms of hyperedge replacement process. Let us consider a deterministic hyperedge replacement graph grammar $(\ell_0, \{H_\ell\}_{\ell \in L_2})$. Let us consider a hypergraph H_ℓ of this grammar, together with a ℓ_e-labeled hyperedge e of H_ℓ. *Gluing* H_{ℓ_e} in place of e in H_ℓ consists in deleting e and then gluing H_{ℓ_e} in place of it by identifying the i-th source of H_{ℓ_e} with the i-th vertex of e in H_ℓ for all i less or equal to the rank of H_{ℓ_e}, which is supposed to be equal to the arity of e. This process is called the *replacement* of e. Now, let us start from H_{ℓ_0}. First, we replace all its non-terminal hyperedges by their associated hypergraphs (according to the grammar). We obtain a graph Γ_1. Inductively, let us suppose that $\Gamma_1, ..., \Gamma_k$ have been already constructed. Then Γ_{k+1} is obtained from Γ_k by replacing all its non-terminal hyperedges by their associated hypergraphs. In a natural way, the graph obtained from

[1] also called a *relational structure*

[2] the equational hypergraphs which are graphs, i.e. whose (terminal) hyperedges are all of arity 2, are also called *regular graphs* (see [6])

Γ_k by deleting its non-terminal hyperedges (let us denote it by $\overline{\Gamma}_k$) can be embedded into Γ_{k+1}, and actually into $\overline{\Gamma}_{k+1}$. So, we get an infinite nested sequence of finite graphs $\overline{\Gamma}_1 \hookrightarrow \overline{\Gamma}_2 \hookrightarrow ... \hookrightarrow \overline{\Gamma}_k \hookrightarrow ....$ This sequence defines an infinite graph as inductive limit which is the equational graph defined by the graph grammar we deal with. The proof that both definitions of equational graphs are equivalent can be found e.g. in [9].

In order to prove Proposition 1 below, we need the two following lemmas.

LEMMA 1. *Let Γ be an equational hypergraph. There exists a deterministic hyperedge replacement graph grammar such that $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ such that all the non-terminal hyperedges of each H_ℓ are simple.*

Proof. First, let us remark that, starting from some deterministic hyperedges replacement graph grammar defining Γ, when a hypergraph has to be glued on an other one according to a non-simple non-terminal hyperedge, then the fact that the hyperedge is non-simple means that some sources of the hypergraph to be glued are identified during the gluing process. The principle of this proof simply is to define some new hypergraphs obtained from the old ones by gluing in advance the sources in question. Then, non-simple hyperedges can be replaced by simple ones, which encode the gluing of the new hypergraphs in which sources have been already glued.

Let $(\ell'_0, \{H'_\ell\}_{\ell \in L'_2})$ be a deterministic hyperedge replacement graph grammar defining Γ. We shall construct a new grammar $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ from the H'_ℓ's such that all the non-terminal hyperedges of each H_ℓ are simple.

For each integer n, let us consider the set Λ_n of ordered partitions of $[1, n]$, i.e. the set of ordered sequences of pairwise disjoint subsets of $[1, n]$ of form $(p_1, ..., p_k)$ such that $\bigcup p_i = [1, n]$. Let $L_2 = \bigcup_{\ell \in L'_2} \{\ell\} \times \Lambda_{\tau_\ell}$ where τ_ℓ is the type of H'_ℓ. For each $(\ell, \lambda) \in L_2$, one defines $H_{(\ell,\lambda)}$ from H'_ℓ as follows:

- For each pair (i, j) of integers such that there exists a subset in λ which contains both i and j, one glues the i-th source of H'_ℓ and its j-th one.
- Let e be a ℓ_e-labeled non-terminal hyperedge. Let $(v_1, ..., v_{\ell_e})$ be the vertex sequence of e. In a natural way, this sequence defines an ordered partition $\lambda_e = (p_1, ..., p_k)$ of $[1, \tau_{\ell_e}]$:
  $p_1 = \{i \mid v_i = v_1\}$
  $p_2 = \{i \mid v_i = v_{i_2}\}$ where $i_2 = \min\{j \mid v_j \notin p_1\}$
  ...
  $p_k = \{i \mid v_i = v_{i_k}\}$ where $i_k = \min\{j \mid v_j \notin p_1 \cup p_2 \cup ... \cup p_{k-1}\}$.
  Eventually, one deletes e, and adds a new (ℓ_e, λ_e)-labeled non-terminal hyperedge of vertex list $(v_1, v_{i_2}, ..., v_{i_k})$. We do that for all the non-terminal hyperedges.

After these two transformations, we get $H_{(\ell,\lambda)}$. As initial symbol of the new grammar, we choose $\ell_0 = (\ell'_0, (\{1\}, \{2\}, ..., \{\tau_{\ell_0}\}))$. Therefore, up to non-terminal hyperedges, H_{ℓ_0} is equal to $H'_{\ell'_0}$.

Now, gluing $H_{(\ell_e,\lambda_e)}$ in place of e' in $H_{(\ell,\lambda)}$ leads to the same hypergraph than gluing H_{ℓ_e} in place of e in H_ℓ, up to non-terminal hyperedges and the gluing of sources. So, starting from H_{ℓ_0}, the replacement of non-terminal hyperedges by the $H_{(\ell,\lambda)}$'s step after step as defined in Remark 3 leads to the same equational graph Γ.

The proof that this new grammar indeed defines Γ is more tedious than difficult. It shall be omitted here. ∎

LEMMA 2 (Composition).
Let A, B, C be finite alphabets. Let $\mathcal{L}_1$ (respectively $\mathcal{L}_2$) be a subset of $A^* \times B^*$ (respectively $B^* \times C^*$) which is recognizable by a letter-to-letter automaton. Then there exists a letter-to-letter automaton which recognizes the set $\mathcal{L}$ of pairs (w_1, w_2) of $A^* \times C^*$ such that there exists $w_3 \in B^*$ such that $(w_1, w_3) \in \mathcal{L}_1$ and $(w_3, w_2) \in \mathcal{L}_2$. $\mathcal{L}$ shall be denoted by $\mathcal{L}_1 \square \mathcal{L}_2$.

Proof. Let us consider $\mathcal{A}_1 = (Q_1, \Delta_1, q_1^{\text{init}}, F_1)$ (respectively $\mathcal{A}_2 = (Q_2, \Delta_2, q_2^{\text{init}}, F_2)$) a letter-to-letter automaton recognizing $\mathcal{L}_1$ (respectively $\mathcal{L}_2$). Q_i is the set of the states of $\mathcal{A}_i$, Δ_i is the set of its transitions, q_i^{init} its initial state, and F_i the set of its final states. Transitions of Δ_1 (respectively Δ_2) are encoded by tuples of form $(q, (s,t), q')$ where $q, q' \in Q_1$, $s \in A \cup \{\$\}$ (respectively $s \in B \cup \{\$\}$) and $t \in B \cup \{\$\}$ (respectively $t \in C \cup \{\$\}$).

We add to $\mathcal{A}_i$ some new transitions: for each final state q_f of $\mathcal{A}_i$, we add a $(\$,\$)$-labeled transition from q_f to itself. The resulting automaton still is denoted by $\mathcal{A}_i$.

We construct an automaton $\mathcal{A} = (Q, \Delta, q^{\text{init}}, F)$ recognizing $\mathcal{L}$ in the following way: $Q = Q_1 \times Q_2$. Δ is the set of tuples $((q_1, q_2), (a, c), (q_1', q_2'))$ where $q_i, q_i' \in Q_i$, $a \in A \cup \{\$\}$ and $c \in C \cup \{\$\}$ are such that there exists $b \in B \cup \{\$\}$ such that $(q_1, (a,b), q_1') \in Q_1$ and $(q_2, (b,c), q_2') \in Q_2$. The initial state q^{init} of $\mathcal{A}$ is defined to be $(q_1^{\text{init}}, q_2^{\text{init}})$. And eventually $F = F_1 \times F_2$. The verification of the fact that $\mathcal{A}$ actually recognizes $\mathcal{L}$ is straightforward. ∎

PROPOSITION 1. *Equational graphs are automatic.*

Proof. Let us consider an equational graph Γ, here we keep the notations of the definition of equational graphs. According to Lemma 1, we can assume that Γ is defined by a deterministic hyperedge replacement graph grammar such that $(\ell_0, \{H_\ell\}_{\ell \in L_2})$ such that all the non-terminal hyperedge of each H_ℓ are simple.

The main idea for the construction of the automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ for a given equational graph Γ consists in defining the language of W as the set of vertices of $\overline{\Gamma}$ (according to the notations of the definition of equational hypergraphs previously given).

Vertices of $\overline{\Gamma}$ are encoded by words over the alphabet $[1, ..., \max\{n_\ell\}_{\ell \in L_2}] \cup \bigcup_\ell V_{H_\ell}$: a vertex v of $\overline{\Gamma}$ is encoded by a word of the form $i_1 ... i_n w$ where $1 \leq i_j \leq \max_{\ell \in L_2}\{n_\ell\}$ and w is a vertex of some H_ℓ; $i_1 ... i_n$ encodes the path from the root of t to the node μ such that $v \in H_\mu$ and w is the vertex of H_ℓ corresponding to v, ℓ denoting the label of μ. Since t is a regular tree, the language made of the words which encode the vertex of Γ is rational. By definition, this is the language of W, from now on it shall be denoted by $\mathcal{L}(W)$.

Let $\overline{R}$ denote the transitive symmetric closure of R (R has been defined in the definition of equational hypergraphs). For each vertex $x = i_1 ... i_n w$ of $\overline{\Gamma}$, let C_x be the set of words over $\mathcal{L}(W)$ of form $i_1' ... i_m' w'$ such that $i_1 ... i_n i_1' ... i_m' w'$ is equivalent to x according to $\overline{R}$. Let us note that w belongs to C_x. For each $z \in \overline{\Gamma}$, let μ_z be the node of t such that $z \in V_{H_{\mu_z}}$. Then a vertex y of $\overline{\Gamma}$ belongs to C_x if and only if y is $\overline{R}$-equivalent to x and μ_y belongs to the subtree of t of root μ_x.

Let z and z' be two $\overline{R}$-equivalent vertices of $\overline{\Gamma}$. Then by definition of $\overline{R}$ there exists a chain $z_0 = z, z_1, ..., z_n = z'$ of pairwise vertices of $\overline{\Gamma}$ such that for each $i = 0, ..., n-1$, (z_i, z_{i+1}) or (z_{i+1}, z_i) belongs to R. Such a sequence is called a R-chain. Then by definition of R, the sequence $\mu_{z_0}, \mu_{z_1}, ..., \mu_{z_n}$ is a path in t.

Let z and z' be two $\overline{R}$-equivalent vertices of $\overline{\Gamma}$ such that $\mu_z = \mu_{z'}$, then $z = z'$. Indeed, let us suppose that there exist two $\overline{R}$-equivalent distinct vertices z and z' of $\overline{\Gamma}$ such that $\mu_z = \mu_{z'}$; we can assume that z and z' have been chosen in such a way that μ_z is minimal (according to the partial order induced by t). Let us consider a chain $z_0 = z, z_1, ..., z_n = z'$ as above of minimal length. First, let us note that by definition of R, if two vertices s and s' are such that $(s, s') \in R$, then $\mu_s \neq \mu_{s'}$. Therefore, for each $i = 0, ..., n-1$, μ_{z_i} and $\mu_{z_{i+1}}$ are distinct. Let us consider $i_{\max}$ such that $\mu_{z_{i_{\max}}}$ is maximal among the μ_{z_i}. Let us suppose that $i_{\max} \neq 0$ and $i_{\max} \neq n$. Then $\mu_{z_{i_{\max}-1}} = \mu_{z_{i_{\max}+1}}$ is the father of $\mu_{z_{i_{\max}}}$. Since the chain $z, z_1, ..., z_n$ have been chosen to be of minimal length, $z_{i_{\max}-1}$ and $z_{i_{\max}+1}$ are distinct. But, by definition of R, there is no pair of distinct vertices of $\overline{\Gamma}$ belonging to the same H_μ which both are in R-relation with a vertex of $H_{\mu'}$ for some son μ' of μ. So, this is a contradiction. Therefore $i_{\max} = 0$ or $i_{\max} = n$, i.e. $\mu_{z_0} = \mu_{z_n}$ is maximal among the μ_{z_i}. Thus, if $n > 1$, $\mu_1 = \mu_{n-1}$ is the father of μ_{z_0}. Let us suppose that $z_1 \neq z_{n-1}$. Then z_1 and z_{n-1} are $\overline{R}$-equivalent and $\mu_{z_1} = \mu_{z_{n-1}}$ is less that μ_{z_0}. This is

a contradiction with the fact that the pair (z, z') has been chosen in such a way that μ_z is minimal. Therefore, $z_1 = z_{n-1}$. Thus, z_1 is in R-relation with z and z'. For each $\mu \in t$, let ℓ_μ be the label of μ. By definition of R, we get that z and z', as vertices of $H_{\ell_{\mu_0}}$, are some sources, say the j-th one and the j'-th one, and that z_1, as a vertex of $H_{\ell_{\mu_1}}$, appears in a ℓ_{μ_0}-labeled hyperedge of $H_{\ell_{\mu_1}}$ at position j and j'. Thus, this hyperedge is not simple. This contradicts the choice of $(\ell_0, \{H_\ell\}_{\ell \in L_2})$. Eventually, we deduce that $n = 1$, this means that z and z' are in R-relation, but they both belong to the same H_μ, this contradicts the definition of R. Eventually, $z = z'$.

C_x only depends on w. Indeed, let us consider $x' = i'_1 ... i'_{m'} w$. We shall show that $C_x \subset C_{x'}$, which shall imply by symmetry that $C_{x'} = C_x$. Let $y \in C_x$. This means that $i_1 ... i_m y$ is $\overline{R}$-equivalent to x. Let us consider a R-chain $z_0 = x, z_1, ..., z_n = i_1 ... i_m y$ of minimal length from x to $i_1 ... i_m y$. Since it is of minimal length, the z_i's are pairwise distinct. And by the above remark, the μ_{z_i}'s are also pairwise distinct. Therefore, for each $i = 1, ..., n$, $\mu_{z_i} \neq \mu_x$. This implies that all the μ_{z_i} are in the subtree of t of root x. This means that each z_i has the form $i_1 ... i_m j^i_1 ... j^i_{m_i} w_i$. Since for each $i = 1, ..., n$, $\mu_{z_i} \neq \mu_x$, we have that for each $i = 1, ..., n$, $m_i > 0$, i.e. at least j^i_1 exists. By definition of R, the fact that z_i and z_{i+1} are R-related only depends on $j^i_{m_i}, w_i, j^{i+1}_{m_{i+1}}$ and w_{i+1} if $i > 0$, and on $w, j^{i+1}_{m_{i+1}}$ and w_{i+1} if $i = 0$. So, for each $i = 1...n$, $i'_1 ... i'_m j^i_1 ... j^i_{m_i} w_i$ and $i'_1 ... i'_m j^{i+1}_1 ... j^{i+1}_{m_{i+1}} w_{i+1}$ are R-related. And $x' = i'_1 ... i'_m w$ is also related to $i'_1 ... i'_m j^1_1 w_1$. For each $i = 1...n$, let $z'_i = i'_1 ... i'_m j^i_1 ... j^i_{m_i} w_i$. Then $z'_0 z'_1 ... z'_n$ is a R-chain from x' to $i'_1 ... i'_m j^n_1 ... j^n_{m_n} w_n$. This prove that $y = j^n_1 ... j^n_{m_n} w_n$ belongs to $C_{x'}$, which is what we wanted. Since C_x only depends on w, from now on, it shall be denoted by C_w.

For each $w \in \bigcup_\ell V_{H_\ell}$, we have

$$C_w = \{w\} \cup \sum_{sw' \in C_w, |sw'|=2} s.C_{w'} \tag{1}$$

Indeed, let us consider $y \in sC_{w'}$ for some s and w' such that $sw' \in C_w$ and $|sw'| = 2$. Let x be a vertex of $\overline{\Gamma}$ of form $i_1 ... i_m w$. By construction, x is $\overline{R}$-equivalent to $i_1 ... i_m sw'$. Let $y' \in C_{w'}$ be such that $y = sy'$. For any vertex x' of $\overline{\Gamma}$ of form $i'_1 ... i'_{m'} w'$, we have that $i'_1 ... i'_{m'} y'$ is $\overline{R}$-equivalent to x'. By choosing $m' = m + 1$ and $i'_j = i_j$ for $j < m'$ and $i'_{m'} = s$, we get that $i_1 ... i_m sw'$ is $\overline{R}$-equivalent to $i_1 ... i_m sy' = i_1 ... i_m y$; y indeed belongs to C_w. Conversely, let us consider $y \in C_w$, let us assume that $y \neq w$. Let x be a vertex of $\overline{\Gamma}$ of form $i_1 ... i_m w$. By definition, x is $\overline{R}$-equivalent to $i_1 ... i_m y$. Let us consider a R-chain $z_0 = x, z_1, ..., z_n = i_1 ... i_m y$ of minimal length from x to $i_1 ... i_m y$. Since it is minimal, μ_{z_1} is a son of μ_x. Therefore, z_1 is of form $i_1 ... i_m sw'$ for some s and w'. And each for $i = 1...n$, μ_{z_i} belongs to the subtree of t of root z_1. This means that for each $i = 1...n$, z_i is of form $i_1 ... i_m s j^i_1 ... j^i_{m_i} w_i$. Therefore, for each $i = 1...n$, $j^i_1 ... j^i_{m_i} w_i$ belongs to $C_{w'}$; in particular, $y = sj^n_1 ... j^n_{m_n} w_n$ indeed belongs to the right member of Equation 1, which is what we wanted. In a classical way, Equation 1 implies that for each w, C_w is a rational language.

Let $x = i_1 ... i_m w$ be a vertex of $\overline{\Gamma}$. If w is an internal vertex of H_ℓ, then all the vertices of $\overline{\Gamma}$ which are $\overline{R}$-equivalent to x are in $i_1 ... i_m C_w$. Indeed, let us consider a vertex y of $\overline{\Gamma}$ which is $\overline{R}$-equivalent to x. Let us consider a R-chain $z_0 = y, z_1, ..., z_n = x$ of minimal length. Since it is of minimal length, the z_i's are pairwise distinct. And by the above remark, the μ_{z_i}'s are also pairwise distinct. Let us suppose that μ_y does not belong to the subtree of t of root x. Thus $\mu_{z_{n-1}}$ is the father of μ_x. And z_{n-1} and x are R-related. But w is an internal vertex of H_ℓ. This contradicts the definition of R. Therefore, μ_y belongs to the subtree of t of root μ_x, and thus $y \in C_x$.

We say that a vertex x of $\overline{\Gamma}$ is *$\overline{R}$-minimal* if for each vertex y of $\overline{\Gamma}$ which is $\overline{R}$-equivalent to x, μ_y is in the subtree of t of root μ_x. From the above discussion, we get that if x has the form $i_1 ... i_m w$ with w internal, then it is the unique $\overline{R}$-minimal element of its $\overline{R}$-equivalence class. And in particular, any vertex of $\overline{\Gamma}$ is $\overline{R}$-equivalent to at most one vertex of form $i_1 ... i_m w$ with w internal.

Let us consider now a vertex x of $\overline{\Gamma}$ whose $\overline{R}$-equivalence class does not contains any vertex of form $i_1 ... i_m w$ with w internal. Each vertex which is $\overline{R}$-equivalent to x, including

x itself, is of form $i_1...i_m w$ where w is a source. Let $x' = i_1...i_m w$ be such a vertex. By definition of R, x' is R-related to a vertex of H_{μ_f} where μ_f is the father of $\mu_{x'}$, unless $\mu_{x'}$ is the root of t. Therefore, any $\overline{R}$-minimal element of such the $\overline{R}$-equivalence class of x is a source of H_{μ_0} where μ_0 is the root of t. Let us note that if x is a vertex of $\overline{\Gamma}$ which is a source of H_{μ_0}, then it has the form $x = w$ for a source w of H_{ℓ_0}, and all the vertex which are $\overline{R}$-equivalent to x are in C_w. In particular x is the unique $\overline{R}$-minimal element of its $\overline{R}$-equivalence class.

Eventually, each $\overline{R}$-equivalence class has a unique $\overline{R}$-minimal element which is a source of H_{μ_0} or has the form $i_1...i_m w$ with w internal. Conversely, a source of H_{μ_0} or a vertex having the form $i_1...i_m w$ with w internal is the unique $\overline{R}$-minimal element of its R-equivalence class. For each vertex x, let $\min_x$ denote the unique $\overline{R}$-minimal of its $\overline{R}$-equivalence class.

The language $\mathcal{L}_{\min} \subset \mathcal{L}(W) \times \mathcal{L}(W)$ made of all the pairs of form $(x, \min_x)$ is recognizable by a letter-to-letter automaton. Indeed, by the above discussion we get:

$$\mathcal{L}_{\min} = \sum_{w \text{ source of } H_{\mu_0}} \{(x, w) \mid x \in C_w\} + \sum_{\rho.w \in \mathcal{L}(W) \mid w \text{ internal}} \{(\rho.x, \rho.w) \mid x \in C_w\}$$

We have seen that for each w, C_w is a rational language, therefore $\mathcal{L}_{\min}$ is recognizable by a letter-to-letter automaton.

By Lemma 2, the language of pairs (x_1, x_2) such that x_1 and x_2 represent the same vertex of Γ is recognizable by a letter-to-letter automaton. Indeed, this language can be written as follows :

$$\{(x_1, x_2) \mid \min_{x_1} = \min_{x_2}\}$$

which is equal to

$$\{(x_1, x_2) \mid \exists m \mid (x_1, m) \in \mathcal{L}_{\min} \text{ and } (x_2, m) \in \mathcal{L}_{\min}\}$$

which, by Lemma 2, is recognizable by a letter-to-letter automaton. This automaton is by definition M_0.

The construction of M_ℓ is similar. Let us note that Γ is a graph, i.e. this is a hypergraph whose hyperedges are all of arity 2. This means that the terminal hyperedges of the H_ℓ's are all of arity 2. Two vertices v and v' of Γ are connected by a ℓ-labeled hyperedge if they have representatives in $\overline{\Gamma}$ which are themselves connected in $\overline{\Gamma}$ by a terminal ℓ-labeled hyperedge of arity 2. Two such representatives both belong to the same H_μ. The language made of such pairs of representative is recognizable by a letter-to-letter automaton seeing its form:

$$\mathcal{L}_\ell = \{(\rho.w, \rho.w') \in \mathcal{L}(W)^2 \mid \exists \mu \mid w, w' \in V_{H_\mu} \text{ and } w \text{ and } w' \text{ are } \ell\text{-connected in } H_\mu\}$$

Therefore, by Lemma 2, the language of pairs of form (x, x') of vertices of $\overline{\Gamma}$ is also recognizable by a letter-to-letter automaton, which shall actually be M_ℓ, since it can be written as $\mathcal{L}(M_0) \square \mathcal{L}_\ell \square \mathcal{L}(M_0)$. ∎

Remark 4. Equational graphs are not regularly accessible automatic graphs in general. Indeed, regularly accessible automatic graphs are connected though equational graphs are not in general.

Remark 5. Conversely, deterministic automatic graphs are definable up to isomorphism by monadic second-order logical formulæ (see [37]). Together with Theorem 6.5 of [9], this implies that *deterministic automatic graphs of bounded tree width are equational.* Let us note that equational graphs are not of bounded tree width in general. Indeed, the graphs defined in Example 1 and 2 are not of bounded tree width (see [32]).

Transition Graphs.

A *transition graph* (see [6]) is defined according to a rational language R over a finite alphabet L and a *label-mapping* $h : L \to \mathrm{Rat}(L \cup \overline{L})$ mapping elements of L to some rational languages over $L \cup \overline{L}$, where $\overline{L}$ is an inverse alphabet of L.

We consider *the complete deterministic tree* over L, i.e. the graph whose vertex set if L^* and for each word w of L^* and each symbol ℓ of L, there is an ℓ-labeled edge from w to $w.\ell$. Vertices are not labeled.

The transition graph T associated to R and h is a graph whose vertices are some words of L^* and whose edges are labeled over L. The vertex set of T is R. Edges are defined as follows: Two vertices v and v' are linked by a ℓ-labeled edge from v to v' if and only if there exists a path in the complete deterministic tree over L from v to v' whose trace is a word of $h(\ell)$.

PROPOSITION 2. *Transition graphs are automatic.*

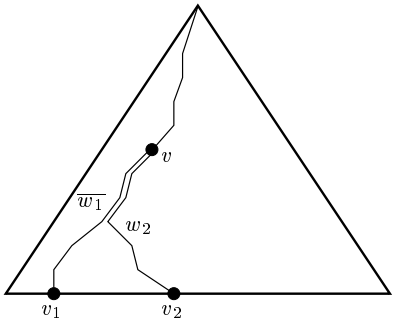
Proof. Let us consider a transition graph T defined by a tuple (R, L, h) as above. First, we transform each $h(\ell)$ into an other rational language $h'(\ell)$ such that the tuple (R, L, h') defines the same transition graph, and each word of each $h'(\ell)$ has no sub-word of the form $\ell\overline{\ell}$ for some $\ell \in L$. The transformation is defined as follows: let us consider an automaton M which defines $h(\ell)$ for some fixed ℓ. Let us pick a pair of states (s, s') of M such that there exists $\ell' \in L$ such that there exists a path of length 2 in M from s to s' which is labeled $\ell'\overline{\ell'}$. Let us consider the automaton M^1 obtained from M by adding a ε-transition from s to s'. We claim that replacing $h(\ell)$ by the language of M^1 does not modify T. Indeed, on the one hand, if a word w' is recognized by M^1 then, either it is also recognized by M, or else there exists $w_1, ..., w_n$ such that $w' = w_1 w_2 ... w_n$ and $w_1 \ell'\overline{\ell'} w_2 \ell'\overline{\ell'} ... \ell'\overline{\ell'} w_n$ is recognized by M. So, if two nodes of the complete deterministic tree over L are connected by a path whose trace is recognized by M^1, then there exists a path connecting them whose trace is recognized by M. The reciprocal is true as well seeing that the language of M is included in those of M^1. We do this operation inductively until there is no more ε-transition to add. This process finish seeing that the number of iterations is bounded by the number of pair of state of M_ℓ. We get at the end an automaton M'. And replacing $h(\ell)$ by the language of M' does not modify T.

But now, if two nodes v and v' are connected by a path whose trace is recognized by M', then there also exists a path connecting v and v' whose trace is also recognized by M' but has no sub-word of form $\ell'\overline{\ell'}$ for some $\ell' \in L$. To finish the transformation, we replace the language of M' by its intersection with the language of words over $L \cup \overline{L}$ which have no sub-word of form $\ell'\overline{\ell'}$ for some $\ell' \in L$. The language we obtain remains rational seeing that the intersection of two rational languages is a rational language. Eventually we get that T can be defined by a tuple (R, L, h') such that the words of each $h'(\ell)$ have no sub-word of form $\ell'\overline{\ell'}$ for some $\ell' \in L$.

We claim that any word of any $h'(\ell)$ actually is of form $w_1.w_2$ where $w_1 \in \overline{L}^*$ and $w_2 \in L^*$. Indeed, let us suppose that there exists a word w in some $h'(\ell)$ which can not be decomposed like that. This means that there exists $w_1 \in (L \cup \overline{L})^*$, $\ell_1 \in L$, $\overline{\ell}_2 \in \overline{L}$ and $w_2 \in (L \cup \overline{L})^*$ such that $w = w_1.\ell_1\overline{\ell}_2.w_2$. But this word is the trace of a path in the complete deterministic tree over L; this implies that $\ell_2 = \ell_1$ and thus $w = w_1.\ell_1\overline{\ell}_1.w_2$; this is in contradiction with the construction of $h'(\ell)$.

Now, let us consider a pair of vertices v_1 and v_2 of T connected by a ℓ-labeled edge. This means that, in the complete deterministic tree over L, there exists a path from v_1 to v_2 whose trace is a word w of $h'(\ell)$. According to the above discussion, there exist $w_1 \in \overline{L}^*$ and $w_2 \in L^*$ such that $w = w_1.w_2$. Then there exists a word $v \in L^*$ such that $v_1 = v.\overline{w_1}$ and $v_2 = v.w_2$ where $\overline{w_1}$ is defined as follows : if $\ell_1, \ell_2, ..., \ell_n$ are such that $w_1 = \ell_1\ell_2...\ell_n$, then $\overline{w_1} = \overline{\ell}_n ... \overline{\ell}_2\overline{\ell}_1$ (see Figure); let us note that $\overline{w_1} \in L^*$. Let us note that for any other $v' \in L^*$, the vertices $v'.\overline{w_1}$ and $v'.w_1$ are connected by a ℓ-labeled edge in T.

This decomposition allows us to construct a letter-to-letter automaton which recognize the pair of words over L which are connected in T. First, there exists a letter-to-letter



automaton $\mathcal{A}_\ell$ which recognizes the pairs of words $(\overline{w_1}, w_2)$ of $L^* \times L^*$ such that $w_1.w_2 \in h'(\ell)$. $\mathcal{A}_\ell$ is constructed from an automaton M recognizing $h'(\ell)$: A pair (v_1, v_2) of $L^* \times L^*$ is recognized by $\mathcal{A}_\ell$ if there exists a state s of M such that M has a run for v_2 from s to a final state, and a run for $\overline{v_1}$ from an initial state of M to s. Let us note that this last condition corresponds to the existence of a run going backward through transitions labeled by inverse letters. The set of states of $\mathcal{A}_\ell$ is the product of those of M by it-self. A pair of states (s_1, s_1') is connected to a pair (s_2, s_2') by a transition of label $(\ell, \ell') \in (L \cup \{\$\})^2 \backslash \{(\$, \$)\}$ if and only if $\ell = \$$ or M has a $\bar{\ell}$-labeled transition from s_2 to s_1, and $\ell' = \$$ or M has a ℓ'-labeled transition from s_1' to s_2'. The initial states of $\mathcal{A}_\ell$ are those of form (s, s) for some state s of M. And final states of $\mathcal{A}_\ell$ are those of form (s_1, s_2) where s_1 is an initial state of M and s_2 a final one. Now, the construction of a letter-to-letter automaton M_ℓ recognizing pairs of words over L which are connected in T by ℓ-labeled edges combines $\mathcal{A}_\ell$ with a letter-to-letter automaton $\mathcal{A}$ recognizing the trivial diagonal language $\{(v, v) \mid v \in L^*\}$: M_ℓ is the disjoint union of $\mathcal{A}$ and $\mathcal{A}_\ell$ with additional ε-transitions from final states of $\mathcal{A}$ to initial states of $\mathcal{A}_\ell$. The initial states of M_ℓ are those of $\mathcal{A}$, its final ones are those of $\mathcal{A}_\ell$.

Eventually, we get an automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell \in L})$ for T by considering the M_ℓ's defined above, and by defining W to be an automaton recognizing R, ν to be the identity over R, and M_0 to be a letter-to-letter recognizing the trivial diagonal language $\{(v, v) \mid v \in L^*\}$. ∎

2. THE PROBLEM OF ENDS

2.1. Ends

Basics.

The concept of *end* of a locally compact Hausdorff space can be understood as expressing the idea of "way going to infinity"; basics about it can be found e.g. in [28, 1, 13, 20]. Let X be a locally compact Hausdorff space; a *region* of X is a non empty connected open subset; a *closed region* is the closure of a region; we say that a subset of X is *bounded* if its closure is compact. Let us consider nested sequences $P_1 \supset ... \supset P_n \supset ...$ of unbounded regions of X such that: The boundary of P_n is compact for all n and for any bounded subset A of X, $A \cap P_n = \emptyset$ for n sufficiently large. Two such sequences $\{P_i\}_{i \geq 0}$ and $\{P_i'\}_{i \geq 0}$ are said to be equivalent if and only if for all n there exists n' such that $P_{n'}' \subset P_n$ and *vice versa*. An equivalent class of such sequences is called an *end* of X.

We consider graphs as metric spaces with the usual metric: edges are of length 1 and the distance between any two points is the length of a shortest non-directed path connecting them. Connected graphs of finite degree are locally compact Hausdorff spaces; and thus, the concept of ends makes sense for them. But in their case, one can use an other characterization of ends which is in somewhat more constructive. Let G be an

infinite connected graph and v_0 be a vertex of G. The *subgraph* $G[A]$ of G *induced* by a subset A of V_G is defined to be the graph whose vertex set is A and such that two vertices are connected in $G[A]$ if and only if they are in G. The *complement* of a subgraph B of G is the subgraph of G which is induced by the complement in V_G of V_B. A *ball* of G of radius ρ centered at a vertex v is the subgraph of G which is induced by the set of vertices which are within a distance less of equal than ρ from v. A non-directed infinite path[3] $\rho = v_0 v_1 v_2 ...$ in G is called an *infinite branch* if the distance between x_0 and v_n is strictly increasing and thus tends to infinity. Two infinite branches are said to be equivalent if they are connected in the complement of any ball centered at v_0. The set of ends of G is in one-to-one correspondence with the set of equivalent classes of infinite branches. In order to sketch the construction of this correspondence, we need some notations; let us consider $\mathfrak{U} = \{U_i\}_{i \in I}$, the family of subgraphs of G which are the infinite connected components of the complements in G of the balls centered at v_0; note that for any pair U_i, $U_{i'}$ of elements of $\mathfrak{U}$, $U_i \cap U_{i'}$ is equal to $\emptyset$, U_i or else $U_{i'}$. One shows that any end p of G is represented by a unique nested sequence $(U_{i_k})_{i_k}$ of elements of $\mathfrak{U}$ such that for all k, U_{i_k} is a connected component of the ball of radius k centered at v_0. One then associates to p an infinite branch constructed as following: for all k, let us choose a vertex w_k of U_{i_k} at distance k from v_0, together with a minimal path α_k from v_0 to w_k. The infinite branch $\alpha = v_0 v_1 v_2 \ldots$ which is associated to p is constructed by induction as following: v_1 is chosen among the vertices of G at distance 1 from v_0 which infinitely many α_k go through, such a vertex exists as G is of finite degree; one do no longer consider the α_k's which do not go through v_1. We then choose v_2 among the vertices at distance 2 from v_0 which infinitely many α_k go through. And so on, the construction follows the König lemma method. We shall omit to show that the equivalence class of α does not depend on the choices of the α_k, neither on the choices of the v_k. We shall also omit to verify that the map we have constructed is a one-to-one correspondence.

For example, the set of ends of the complete binary tree is an infinite non-countable set, actually a Cantor set[4]. An other example is given by the infinite squared grid which has only one end. This can be proved by using the following criteria.

LEMMA 3. *Let G be an infinite connected graph of finite degree. G has one and only one end if and only if the complement in G of any finite connected subgraph has one and only one infinite connected component.*

Proof. We first restrict ourself to balls, showing that the complement in G of any finite connected subgraph has one and only one infinite connected component if and only if the complement in G of any ball centered at a fixed vertex v_0 has one and only one infinite connected component. The direct way is obvious; let us look at the converse. Suppose that there exists in G a finite connected subgraph A whose complement has at least two disjoint infinite connected components U and V. Let us consider a ball B centered at v_0 which contains A; let us note that, since G is of finite degree, B is itself finite. We shall see that U and V give rise to at least two infinite connected components of the complement of B. Indeed, $U \backslash B$ is an infinite set. And, because G is of finite degree, $U \backslash B$ has finitely many connected component. So, one of them is infinite, let us denoted it by U'. Let us note that U' also is a component of $G \backslash B$. In the same way, V allows us to construct an infinite connected component V' of $V \backslash B$, which actually is an infinite connected component of $G \backslash B$ and which is disjoint with U'. That is what we wanted.

Now, it remains to see that G has one and only one end if and only if the complement of any ball centered at v_0 contains only one infinite connected component. We keep the notations of the previous construction. We saw that the set of ends of G is in one-to-one correspondence with the set of nested sequences $(U_{i_k})_k$ of elements of $\mathfrak{U}$ such that U_{i_k}

[3] Note that when we deal with the topology of graphs, the orientation of edges does not matter. One says that two vertices are connected by a path if they are connected by a path of non-directed edge.

[4] The set of ends is usually provided with a topology, cf. e.g. [28]

is an infinite connected component of the complement of the ball of radius k centered at v_0. Since the complement of such a ball is assumed to have only one infinite connected component, U_{i_k} is uniquely determined k and there is only one nested sequence $(U_{i_k})_k$. The conclusion follows then : G has only one end. ∎

As a corollary we get the following result:

LEMMA 4. *Let G be an infinite connected graph of finite degree. If the complement in G of any ball centered at a fixed vertex v_0 is connected, then G has one and only one end.*

Proof. Indeed, let A be a finite connected subgraph of G. Since G is the union of all the balls centered in v_0, A is contained in a ball, say B. Let U be an infinite connected component of the complement in G of A. Then, since B is finite, the intersection of U and the complement of B is non empty. Moreover, since the complement of B is assumed to be connected, it is entirely contained in U. Let us consider U' an other infinite connected component of the complement of A. Similarly, U' contains the complement of B, and thus, its intersection with U is non empty; which is a contradiction. Eventually, the complement of A has only one infinite component. And thus, by Lemma 3, G has one and only one end. ∎

Let us now come back to the infinite square grid as defined in Example 1. Keeping the notations of Example 1, the distance between two vertices (x, y) and (x', y') of G is equal to $|x' - x| + |y' - y|$. So, the ball of radius ρ centered at $(0, 0)$ is the subgraph of G induced by the following subset of vertices : $\{(x, y) \mid |x| + |y| \leq \rho\}$. Its complement is obviously connected. Therefore, by Lemma 4, G has one and only one end.

One End Graphs and Counting Monadic Second-Order Logic.

Before turning to the case of automatic graphs, let us look at the problem of ends for context-free graphs in the sense of [29]. Let us note that since only locally compact spaces are considered, we only are interested in graphs of finite degree. And equational graphs, and also transition graphs, when they are of finite degree are all context-free graphs.

Here we use a result concerning *counting monadic second-order logical formulæ* [19, 10]. Such formulæ are constructed by using individual and set variables, usual boolean operators, atomic formulæ which express adjacency, labels, inclusion, and by adding the unary predicate $\texttt{fin}(X)$ which is true if and only if X is finite. Quantifications are done over vertices and vertex sets only.

LEMMA 5. *For infinite connected graphs of finite degree, the property of having one and only one end can be expressed by a formula of the counting monadic second-order language.*

Proof. According to Lemma 3, an infinite connected graph G of finite degree has one and only one end if and only if the complement in G of any finite connected subgraph has one and only one infinite connected component.

First, let us remark that connectivity can be expressed by a monadic second-order formula (cf. e.g. [15, Chap. 5]). Therefore there is a predicate which expresses that a subset of vertex induces a finite and connected subgraph. The complement of a definable subset of vertex also is definable. So, there exists a predicate which expresses that the complement of a given subset of vertex induces a finite connected subgraph.

Second, let us consider two subsets H and U of V_G such that $U \subset H$. Then there exists a predicate expressing that U is a connected component of H. Indeed, the connected components of H are the maximal connected subsets of H. So, there exists a predicate expressing that H has one and only one infinite connected component.

Eventually, one can express the fact that the complement in G of any finite connected subgraph has one and only one infinite connected component, which is what we wanted. ∎

The counting monadic second-order theories of context-free graphs are decidable (cf. [8]). We thus get the following result:

PROPOSITION 3. *There exists an algorithm to decide whether a context-free graph has one and only one end or not.*

One can actually show that the space of ends of a context-free graph is homeomorphic to the adherence of a rational language, which is computable (see [26]). This gives a constructive way to catch it.

2.2. Undecidability of the Problem of Ends for Automatic Graphs

This section is devoted to the undecidability of the problem of ends for automatic graphs, i.e. Theorem 1. We shall actually prove the following stronger result:

THEOREM 4. *The problem of deciding whether a regularly accessible automatic graph has more than one end is recursively undecidable.*

Proof of Theorem 4 is a reduction to the Turing machines halting problem. We start with a Turing machine which we slightly modify as described below. We then consider the *self-stabilizing machine* associated to it, to be defined below, and finally the *configuration graph* of this self-stabilizing machine which we also slightly modify. Lemma 10 below eventually states that this graph has only one end if and only if the initial Turing machine does not stop, which obviously implies Theorem 4.

This section is organized as follows: we start by defining Turing machine configuration graphs; then we define the concept of self-stabilizing Turing machine; And eventually we address the proof of Theorem 4.

Turing Machine Configuration Graphs

In this article, we consider one tape deterministic Turing machines with *several* tape heads; a particularity of our model is that they can detect whether the tape heads are scanning the first cell of the tape. Basics about Turing machines can be found e.g. in [22, 34]. Formally, a deterministic Turing machine with $n > 0$ tape heads is a tuple $T = (Q, \Gamma, \delta, q_{\mathsf{init}}, F, \Box)$ where Q is the finite set of the *states* of T; Γ is the *tape alphabet*; δ is a partial function from $Q \times \Gamma^n \times \{0,1\}^n$ into $Q \times \{1, ..., n\} \times (\Gamma \backslash \{\Box\}) \times \{\mathsf{left}, \mathsf{stay}, \mathsf{right}\}$; $q_{\mathsf{init}} \in Q$ is the *initial state*; $F \subset Q$ is the set of *final states* and $\Box \in \Gamma$ is the blanc symbol. T is provided with an infinite tape. At the beginning, the tape only contains blank symbols $\Box$; T is in the state q_{init} and all its tape heads are scanning the first cell. We explain the meaning of δ: Let $\delta(q, a_1, ..., a_n, b_1, ..., b_n) = (q', i', a', D')$; this means that if the machine is in the state q, and for each $i \in \{1, ..., n\}$: the i-th tape head is reading a_i and it is scanning the first cell if and only if $b_i = 1$, then the machine goes into the state q', the i'-th tape head write $a' \in \Gamma \backslash \{\Box\}$ and moves to the left (respectively keeps its position, moves to the right) if $D' = \mathsf{left}$ (respectively $D' = \mathsf{stay}$, $D' = \mathsf{right}$). Note that T can not write any blank symbol on the tape.

An *instantaneous description* is a word over the alphabet $(Q \times \{1, ..., n\}) \cup (\Gamma \backslash \{\Box\})$ of the form $w_0 q^{i_1} w_1 q^{i_2} ... w_{n-1} q^{i_n} w_n$ where w_j is a word over $\Gamma \backslash \{\Box\}$; (q, i_j) is denoted by q^{i_j} to simplify notations and $\{i_1, ..., i_n\} = \{1, ..., n\}$. The meaning of $w_0 q^{i_1} w_1 q^{i_2} ... w_{n-1} q^{i_n} w_n$ is that the tape contains $w_0.w_1...w_{n-1}.w_n \Box\Box...$; the machine is in the state q and for each $j \in \{1, ..., n\}$, the i_j-th tape head is scanning the first symbol of w_j. Let $i' > i$; if the i-th tape head and the i'-th one scan the same tape position, to get uniqueness the instantaneous description is written $wq^i q^{i'} w'$; the word $wq^{i'} q^i w'$ is not regarded as an instantaneous description. Let Id_1 and Id_2 be two instantaneous descriptions of T. We write $\mathsf{Id}_1 \underset{T}{\rightarrow} \mathsf{Id}_2$ (respectively $\mathsf{Id}_1 \overset{*}{\underset{T}{\rightarrow}} \mathsf{Id}_2$, $\mathsf{Id}_1 \overset{+}{\underset{T}{\rightarrow}} \mathsf{Id}_2$) to say that Id_2 is the instantaneous description describing T after a one-step computation (respectively a finite computation, a

finite computation of length greater than zero) from Id_1.

From the classical point of view (see e.g. [30]), the configuration graph of T is defined to be the graph whose vertices are the instantaneous descriptions of T, and whose edges encode the transitions of T between instantaneous descriptions. One of the main tools involved in the proof of Theorem 4 is a slight variation of this concept. The graph we shall use is defined as follows: Let $A = (Q \times \{1, ..., n\}) \cup (\Gamma \backslash \square)$. Let $\mathcal{L} \subset A^*$ denote the language of instantaneous descriptions of T. Let us consider $\mathcal{L}.\square^*$ i.e. the language of words of the form $\mathsf{Id}.\square\square...\square$ with $\mathsf{Id} \in \mathcal{L}$. Note that a word of $\mathcal{L}.\square^*$ also encodes an instantaneous description of T in a non ambiguous way. Let $\overline{\mathcal{L}.\square^*}$ denote the prefix closure of $\mathcal{L}.\square^*$. We consider the graph G_T defined as follows: Vertices of G_T are the elements of $\overline{\mathcal{L}.\square^*}$. For all $w_1, w_2 \in \overline{\mathcal{L}.\square^*}$ and $a \in A$ such that $w_2 = w_1 a$ there is an edge labeled by a connecting w_1 to w_2. And for any two words $w_1, w_2 \in \overline{\mathcal{L}.\square^*}$ of the same length defining respectively two instantaneous descriptions Id_1 and Id_2 such that $\mathsf{Id}_1 \xrightarrow{T} \mathsf{Id}_2$, there is an edge connecting w_1 to w_2. Note that as $\overline{\mathcal{L}.\square^*}$ is the prefix closure of $\mathcal{L}.\square^*$, some vertices of G_T do not define any instantaneous description.

Let v be a vertex of G_T. The distance between it and ε i.e. the minimal length of a non-directed path connecting them, is denoted by $|v|$. Note that it is simply the length of v as an element of $\overline{\mathcal{L}.\square^*}$.

LEMMA 6. *Let T be a Turing machine. Then the configuration graph of T is an automatic graph. And G_T is a regularly accessible automatic graph. Moreover, automatic graph presentations defining them can be effectively constructed from T.*

Proof. The fact that the configuration graph is automatic is deserved in [5]. Here we look at the second assertion. Firstly, $\overline{\mathcal{L}.\square^*}$ is obviously rational; we define W to be an automaton recognizing it. M_0 recognizes the trivial diagonal language $\{(v, v) \in (\overline{\mathcal{L}.\square^*})^2 \mid v \in \overline{\mathcal{L}.\square^*}\}$. The transversal edge automaton has to recognize pairs of words of the form $(v_1, v_2) \in (\overline{\mathcal{L}.\square^*})^2$ which encode one-step computations of T; to do that, it first has to guess the transition of T which is applied to v_1 to obtain v_2. This is possible by constructing a non deterministic automaton. And then it has to check that this transition actually drives T from v_1 to v_2, which is possible with a finite amount of memory. Indeed, the modification of the instantaneous description is local. ∎

COROLLARY 1. *The problem of connectivity is undecidable for automatic graphs.*

Proof. Let us consider a Turing machine T with one tape head. Let us consider the graph G which is obtained from the configuration graph of T by connecting all the final instantaneous descriptions, i.e. instantaneous descriptions with final states, to the initial one, i.e. q_0. This graph is connected if and only if the computation of T is finite from every instantaneous description. Indeed, if the computation of T is finite from every instantaneous description, this means that there is in G a path from any vertex to a final instantaneous description, which is itself connected to q_0. G is thus connected. Conversely, let us suppose that G is not connected. This means that there exists an instantaneous description Id which is not connected to q_0. This means that the computation of T from Id never reaches any final instantaneous description. The problem of deciding whether a given Turing machine halts on any instantaneous description is called the *immortality problem* for Turing machines; it is known to be undecidable (see [21]). ∎

Remark 6. It has been shown in [23] that the first order satisfiability problem is decidable for automatic relational structures, and thus for automatic graphs. This result have been extended in [4] to first order formulæ with quantifiers of the form $\exists^\infty x$, which means "there exists infinitely many x". The fact that the connectivity problem is undecidable for automatic graphs shows that if one adds a closure operator, which is classical, the satisfiability problem for automatic graphs becomes undecidable.

Self-Stabilizing Turing Machines.

Let $T = (Q, \Gamma, \delta, q_{\text{init}}, \emptyset, \Box)$ be a one tape head Turing machine without final state such that for all $q \in Q$, $a \in \Gamma$, $b \in \{0,1\}$, $\delta(q,a,b)$ is defined. We describe in the following a Turing machine $\widetilde{T}$ with two tape heads which simulates the computation of T; $\widetilde{T}$ will be called the *self-stabilizing* machine associated to T.

One of the aims of this construction is to achieve the property to be described in Lemma 8; we want indeed to control the behavior of $\widetilde{T}$ on all its instantaneous descriptions and not only on those produced by a normal computation from the initial state.

The idea for the construction of $\widetilde{T}$ is as follows: The simulation is done by computing step by step the successive instantaneous descriptions of T during its computation. They are stocked on the tape without repetition. At each global loop, $\widetilde{T}$ verifies that the list of instantaneous descriptions which are stocked on its tape well describes the real computation of T; if it does not, $\widetilde{T}$ goes into a special state called the `BUG` state. When one deals with some Turing machine, one is usually not able to anticipate its behavior on any instantaneous description. Here we want the machine to drive itself aware that it is not in the right computation, i.e. the computation from the initial state. This is the role of the `BUG` state.

Here we describe the working of $\widetilde{T}$. A more formal construction is done in great details in Appendix.

Step 1. $\widetilde{T}$ starts by putting the initial instantaneous description of T on the tape. More precisely, the first tape head writes the string $\#q_{\text{init}}$ at the beginning of the tape. The symbol # stands for a separation between instantaneous descriptions of T stoked on the tape. Note that the initial instantaneous description of T is simply q_{init}.

Step 2. $\widetilde{T}$ verifies that the tape contains the sequence of successive instantaneous descriptions of T during a finite computation from q_{init}. It also verifies that no instantaneous description appears twice. Rigorously, it checks that the content of the tape is of the form

$$\#\mathsf{Id}_0\#\mathsf{Id}_1\#\ldots\#\mathsf{Id}_k \tag{2}$$

where $\mathsf{Id}_0 = q_{\text{init}}$; for all $i \in \{0, k-1\}$, $\mathsf{Id}_i \underset{T}{\rightarrow} \mathsf{Id}_{i+1}$ and for all $j \neq i$, $\mathsf{Id}_i \neq \mathsf{Id}_j$. $\widetilde{T}$ verifies that $\mathsf{Id}_i \underset{T}{\rightarrow} \mathsf{Id}_{i+1}$ by using the first tape head to scan Id_i while the second one reads and checks Id_{i+1}. This can be done as this operation is local in the sense that it is sufficient to check correspondence of symbols of Id_i and of Id_{i+1} three by three. If these conditions are not fulfilled, $\widetilde{T}$ goes into the `BUG` state, which means that $\widetilde{T}$ failed. When it is in the `BUG` state, $\widetilde{T}$ stays in it permanently without modifying the content of the tape.

Step 3. $\tilde{T}$ computes the instantaneous description following the last one written on the tape. If it does not already appear in the list stocked on the tape, it is written in last position. To do that, the first tape head starts at the beginning of the last instantaneous description Id_k (here we keep the notations of (2) of the description of Step 2) and the second one goes at the beginning of the tape. Let Id_{k+1} be the instantaneous description of T just following Id_k. The first tape head scans Id_k while the second one checks that Id_{k+1} is distinct from Id_0. After that, the first tape head return to the beginning of Id_k and the second one goes to the beginning of Id_1. And so on; this operation is done for all the Id_i's. Finally, if Id_{k+1} has not been found on the tape, it is written next to Id_k.
$\widetilde{T}$ then returns to Step 2.

Note that $\widetilde{T}$ does not simulate T strictly. An instantaneous description of T cannot indeed appear twice on the tape. So, if there is a loop in the sequence of successive instantaneous descriptions of T, then $\widetilde{T}$ writes the first ones of them until it reaches the

first repetition; after this point, $\widetilde{T}$ does never add any more instantaneous description of T. In this case, the content of its tape stops growing.

LEMMA 7 (Normal behavior of $\widetilde{T}$).
Let $\widetilde{\mathsf{Id}}$ be an instantaneous description describing $\widetilde{T}$ at the beginning of Step 2 with tape content of the form:

$$\#\mathsf{Id}_0\#\mathsf{Id}_1\#...\#\mathsf{Id}_k$$

where the Id_i's are instantaneous descriptions of T, $\mathsf{Id}_0 = q_{\mathsf{init}}$, for all $i = 0, ..., k-1$, $\mathsf{Id}_i \underset{T}{\rightarrow} \mathsf{Id}_{i+1}$, and for all $i \neq j$, $\mathsf{Id}_i \neq \mathsf{Id}_j$.

Let Id_{k+1} be the instantaneous description of T following Id_k. If Id_{k+1} appears in $\widetilde{\mathsf{Id}}$, i.e. if there exists $0 \leq i \leq k$ such that $\mathsf{Id}_{k+1} = \mathsf{Id}_i$, then there exists an instantaneous description $\widetilde{\mathsf{Id}}'$ with the same tape contains as the one of Id such that $\widetilde{\mathsf{Id}} \overset{+}{\underset{\widetilde{T}}{\rightarrow}} \widetilde{\mathsf{Id}}'$. Otherwise $\widetilde{\mathsf{Id}} \overset{+}{\underset{\widetilde{T}}{\rightarrow}} \widetilde{\mathsf{Id}}'\#\mathsf{Id}_{k+1}$.

The preceding lemma actually describes the idea which the construction of $\widetilde{T}$ follows. The proof is a tedious induction on k, using the complete description of the machine given in Appendix.

The following lemma states the main property of $\widetilde{T}$: From any instantaneous description, either $\widetilde{T}$ catches up with the normal behavior, i.e. those coming from its initial state or it goes into the BUG state after a finite computation.

LEMMA 8 (Main Property of Self-Stabilizing Machines).
Let $\widetilde{\mathsf{Id}}$ be an instantaneous description of $\widetilde{T}$. Then

- *either there exists an instantaneous description $\widetilde{\mathsf{Id}}'$ of $\widetilde{T}$ such that $\widetilde{\mathsf{Id}} \overset{*}{\underset{\widetilde{T}}{\rightarrow}} \widetilde{\mathsf{Id}}'$ and $\widetilde{q}^1_{\mathsf{init}}\widetilde{q}^2_{\mathsf{init}} \overset{*}{\underset{\widetilde{T}}{\rightarrow}} \widetilde{\mathsf{Id}}'$ where $\widetilde{q}_{\mathsf{init}}$ is the initial state of $\widetilde{T}$,*
- *or $\widetilde{\mathsf{Id}}$ drives $\widetilde{T}$ into the BUG state after a finite computation.*

This lemma is a direct consequence of Lemma 14 of Appendix. The proof is omitted. Let us note that the instantaneous descriptions being a matter for the first case are not only those coming from the initial configuration of $\widetilde{T}$. For instance, at the beginning of Step 2, one can modify the positions of the tape heads without modifying the content of the tape in such a way that the new configuration does not occur in the normal computation of $\widetilde{T}$. What happens is that anyway the tape heads go back to the beginning of the tape (see the full description of $\widetilde{T}$ given in Appendix). And then, the behavior of $\widetilde{T}$ is not affected.

Proof of Theorem 4.

Let $T = (Q, \Gamma, \delta, q_{\mathsf{init}}, F, \square)$ be a one head Turing machine. We first modify T as following. First of all, states of F are no longer considered as final states, i.e. we let F be the empty set but its states are not deleted; instead of that, we add new transitions in order to ensure that once T has reached such a state, it stays in it permanently without stopping, leaving the tape and the head position unchanged. We also want T to be never jammed by a missing transition; to achieve that, we add a new state called the "well" state in which T goes every times it is in such a state, reading such a symbol that no transition is defined. The third step of this modification aims to transform T in such a way that, before it has reached a final state, i.e. a state considered as final in the original definition of T, it takes more and more space on the tape. To achieve this property, it suffices to insist that T, between any two transitions, goes to the end of the tape and writes a new symbol. Details of this transformation are left to the reader. We get then a machine still denoted by T *which never stops and which takes a bounded space on the tape if and only if the initial machine stops after a finite computation.*

Let us consider $\widetilde{T}$, the self-stabilizing Turing machine associated to T as in Section 2.2 and $G_{\widetilde{T}}$, the automatic graph associated to $\widetilde{T}$ as in Section 2.2. We also modify $G_{\widetilde{T}}$.

First, we add an infinite branch of radial edges labeled by a new symbol also denoted by BUG which starts at ε (see Figure 3). The new symbol BUG must not be confused with the symbols $\mathtt{BUG}^1$ and $\mathtt{BUG}^2$ which refer to the pairs $(\mathtt{BUG}, 1)$ and $(\mathtt{BUG}, 2)$ used to describe instantaneous descriptions of $\widetilde{T}$ with BUG state. We then add to the automatic presentation of $G_{\widetilde{T}}$ a new edge automaton $M_{\mathtt{BUG}}$ which connects any vertex representing an instantaneous description with BUG state to the new branch $\mathtt{BUG}^\omega$. More precisely, $M_{\mathtt{BUG}}$ connects any word of the form $w.q^i.w'.q^j.w''.\Box\Box...\Box$ of length $l > 0$ where $q = \mathtt{BUG}$, with the word $\underbrace{\mathtt{BUG.BUG...BUG}}_{l}$. The graph defined by this new automatic presentation is still denoted by $G_{\widetilde{T}}$.

LEMMA 9. *Let v and v' be two vertices of $G_{\widetilde{T}}$. Suppose there exists two instantaneous descriptions $\widetilde{\mathsf{Id}}$ and $\widetilde{\mathsf{Id}}'$ of which v and v' respectively represent some prefixes, such that $\widetilde{\mathsf{Id}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}'$. Then there exists a path in $G_{\widetilde{T}}$ connecting v and v' and remaining at distance greater than $\min\{|v|, |v'|\}$ from ε.*

Proof. Let $\widetilde{\mathsf{Id}} = \widetilde{\mathsf{Id}_0} \to \widetilde{\mathsf{Id}_1} \to ... \to \widetilde{\mathsf{Id}_k} = \widetilde{\mathsf{Id}}'$ be the computation sequence of $\widetilde{T}$ from $\widetilde{\mathsf{Id}}$ to $\widetilde{\mathsf{Id}}'$. Let $\ell_{\max} = \max_{0 \le i \le k} \{|\widetilde{\mathsf{Id}_i}|\} = |\widetilde{\mathsf{Id}}'|$; note that $\ell_{\max} \ge \min\{|v|, |v'|\}$. For $i \in \{0, k\}$, let $v_i = \widetilde{\mathsf{Id}_i}\Box^{\ell_{\max} - |\widetilde{\mathsf{Id}_i}|}$. Then by definition of $G_{\widetilde{T}}$, for each $i = 0, ..., k-1$ there is a transversal edge connecting v_i to v_{i+1} and the path $v_0 v_1 ... v_k$ remains at distance greater than $\min\{|v|, |v'|\}$. On the other hand, v (respectively v') is connected to v_0 (respectively v_k) by a path of radial edges. ∎

Theorem 4 follows directly from the following fact.

LEMMA 10. *If T reachs a final state after a finite computation then $G_{\widetilde{T}}$ has two ends. Otherwise $G_{\widetilde{T}}$ has one end.*

Proof.
Suppose that T does never reach any final state. Our aim is to prove that $G_{\widetilde{T}}$ has only one end. Let $k > 0$ and let v be vertex of $G_{\widetilde{T}}$ at a distance greater than k from ε. We prove that v is connected to the infinite branch $\mathtt{BUG}^\omega$ by a path remaining at distance greater than k from ε. Let $\widetilde{\mathsf{Id}}$ be an instantaneous description of $\widetilde{T}$ of which v represents a prefix. By Lemma 8, there are two cases:

1. either there exists an instantaneous description $\widetilde{\mathsf{Id}}'$ such that $\widetilde{\mathsf{Id}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}'$ and $\widetilde{q}^1_{\mathsf{init}}\widetilde{q}^2_{\mathsf{init}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}'$ where $\widetilde{q}_{\mathsf{init}}$ is the initial state of $\widetilde{T}$,
2. or $\widetilde{\mathsf{Id}}$ drives $\widetilde{T}$ into the BUG state after a finite computation.

Case 1: Recall that T has been modified in such a way that it takes more and more space on the tape. This implies that the sequence of instantaneous descriptions it defines during its computation cannot contain any loop. Thus, the sequence stocked by $\widetilde{T}$ increases strictly and we can choose $\widetilde{\mathsf{Id}}'$ to be of length greater than k. A look at Step 2 in the definition of self-stabilizing machines shows that we can actually choose $\widetilde{\mathsf{Id}}'$ such that the two tape heads are on the right of the k-th cell of the tape.
Let v' be the vertex associated to $\widetilde{\mathsf{Id}}'$. By Lemma 9, there exists a path from v to v' remaining at distance greater than k from ε.
Let w_1, w_2, w_3 be words over the tape-alphabet of $\widetilde{T}$, let q be a state of $\widetilde{T}$ and $i \in \{1, 2\}$ such that $\widetilde{\mathsf{Id}}' = w_1 q^i w_2 q^{\bar{i}} w_3$. Since the two tape heads are on the right of the k-th cell, $|w_1| > k$. Let v'' be the vertex associated to w_1. Trivially, v' is connected to v'' by covering backward a path of radial edges remaining at distance greater than k from ε. Let $\widetilde{\mathsf{Id}}'' = w_1 \mathtt{BUG}^1\mathtt{BUG}^2$. By covering two radial edges in the opposite direction of ε, we connect v'' to the vertex representing $\widetilde{\mathsf{Id}}''$ which is directly connected to $\mathtt{BUG}^{|\widetilde{\mathsf{Id}}''|}$ by a transversal edge. We eventually have connected v to the branch $\mathtt{BUG}^\omega$, remaining at distance greater than k from ε (see Figure 3).

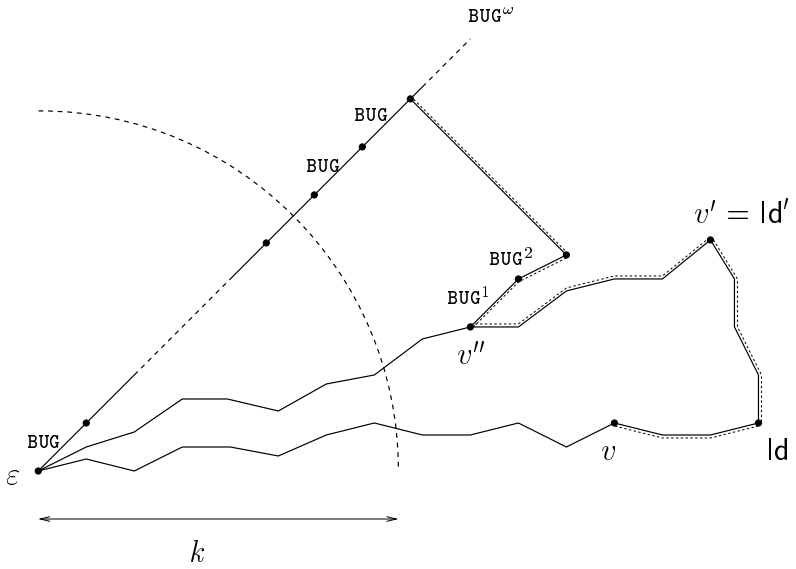


FIG. 3 Case 1

Case 2: Let $\widetilde{\mathsf{Id}}_{\mathtt{BUG}}$ be the first instantaneous description with BUG state which is accessible from $\widetilde{\mathsf{Id}}$. Then by Lemma 9, v is connected to the vertex associated to $\widetilde{\mathsf{Id}}_{\mathtt{BUG}}$ by a path remaining at a distance greater than k from ε. And by construction, there is a transversal edge between $\widetilde{\mathsf{Id}}_{\mathtt{BUG}}$ and $\mathtt{BUG}^{|\widetilde{\mathsf{Id}}_{\mathtt{BUG}}|}$.

The complement of the ball of radius k centered at ε is thus connected, which proves together with Lemma 9 that $G_{\widetilde{T}}$ has only one end.

Suppose that T reaches a final state after a finite computation. Let $k > 0$ and let v be a vertex of $G_{\widetilde{T}}$ at a distance greater than k from ε. Let $\widetilde{\mathsf{Id}}$ be an instantaneous description of which v represents a prefix.
As above, let us consider the two cases foreseen by Lemma 8. If $\widetilde{\mathsf{Id}}$ drives $\widetilde{T}$ to the BUG state, then v is connected to $\mathtt{BUG}^\omega$ by a path remaining at distance greater than k from ε. Suppose then that $\widetilde{T}$ will never go into the BUG state from $\widetilde{\mathsf{Id}}$. By the assumption, T reaches a final state and by construction, it stays in it indefinitely, leaving the tape and the head position unchanged: The sequence of instantaneous descriptions describing the computation of T enters into a loop. Let Id be the first instantaneous description of T to be repeated.
Once $\widetilde{T}$ has written Id on the tape, it does not write any more instantaneous description (Lemma 7). The size of the content of the tape is therefore bounded and the sequence of instantaneous descriptions describing the computation of $\widetilde{T}$ also enters into a loop. Let $\widetilde{\mathsf{Id}}''$ be the first instantaneous description of $\widetilde{T}$ to be repeated.
We prove that v is connected to the branch $\widetilde{\mathsf{Id}}''\Box^\omega$. By Lemma 8 there exists $\widetilde{\mathsf{Id}}'$ such that $\widetilde{\mathsf{Id}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}'$ and $\widetilde{q}^1_{\mathsf{init}}\widetilde{q}^2_{\mathsf{init}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}'$. So $\mathsf{Id}' \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}''$ and thus $\widetilde{\mathsf{Id}} \xrightarrow[\widetilde{T}]{*} \widetilde{\mathsf{Id}}''$. By Lemma 9, v is connected to $\widetilde{\mathsf{Id}}''$ by a path remaining at distance greater than k from ε.

We have thus shown that any vertex is connected either to the branch $\mathtt{BUG}^\omega$ or to the branch $\widetilde{\mathsf{Id}}''\Box^\omega$ by path remaining far from ε. It remains to show that these two infinite branches represent different ends.
Let $k > |\widetilde{\mathsf{Id}}''|$. Suppose that there exists a vertex v of the form $\widetilde{\mathsf{Id}}''\Box^\ell$ with $\ell > 0$ which is

connected to $\mathtt{BUG}^\omega$ by a non-directed path α remaining at distance greater than k from ε. Let $\mathcal{I}$ denote the set of instantaneous descriptions of $\widetilde{T}$ which drive to $\widetilde{\mathsf{Id}}''$. Note that this set is finite since $\widetilde{T}$ cannot write the blank symbol $\Box$. Let v' be the first vertex of α which represents a prefix of an instantaneous description $\widetilde{\mathsf{Id}}_{v'}$ not belonging to $\mathcal{I}$. Note that $v' \neq v$. Let v'' be the vertex of α just preceding v'. By construction of v'', any instantaneous description of which v'' represents a prefix is in $\mathcal{I}$. But $\mathcal{I}$ is finite; and if a vertex is prefix of several instantaneous descriptions then it is prefix of infinitely many different ones. This implies that v'' is of the form $\widetilde{\mathsf{Id}}_{v''}\Box^{\ell'}$ where $\widetilde{\mathsf{Id}}_{v''}$ is a complete instantaneous description of $\widetilde{T}$ and $\ell' > 0$. Moreover, by definition of $\mathcal{I}$, $\widetilde{\mathsf{Id}}_{v''} \to \widetilde{\mathsf{Id}}''$ and then, $|\widetilde{\mathsf{Id}}_{v''}| \leq \widetilde{\mathsf{Id}}'' < k$. Suppose now that v' and v'' are connected by a radial edge; as $|v'| \geq k$, $\widetilde{\mathsf{Id}}_{v''}\Box$ is prefix of v'; thus $\widetilde{\mathsf{Id}}_{v'} = \widetilde{\mathsf{Id}}_{v''}$ which contradicts the fact that $\widetilde{\mathsf{Id}}_{v'}$ does not lead to $\widetilde{\mathsf{Id}}''$. Therefore, v' and v'' are connected by a transversal edge which means that $\widetilde{\mathsf{Id}}_{v'} \underset{\widetilde{T}}{\to} \widetilde{\mathsf{Id}}_{v''}$ or $\widetilde{\mathsf{Id}}_{v''} \underset{\widetilde{T}}{\to} \widetilde{\mathsf{Id}}_{v'}$; since $\widetilde{\mathsf{Id}}_{v''} \in \mathcal{I}$, both cases implies that $\widetilde{\mathsf{Id}}_{v'} \underset{\widetilde{T}}{\to} \widetilde{\mathsf{Id}}''$, which is a contradiction. ∎

3. GRAPH $D0L$-SYSTEMS

3.1. Definition

Some efforts have been made to extend L-systems (see [35]) to n-dimensional structures, under the name of *map generating systems*, i.e. *$M0L$-systems* (cf. [25, 12, 24]). They was initially intended to simulate the process of growing generations of tissue layers in biology. In an other framework, a parallel graph generation formalism, edge grammars, has been introduced in [2] to simulate parallel computations (see also [3]). Parallel graph transformations have been also investigated in [33], extending the classical node label controlled systems into a parallel process of vertex replacement called *parallel BNLC grammars*.

The graph transformations to be introduced here is a generalization of the concept of L-systems to graphs. It consists in the simultaneous deterministic context-free replacement of all the vertices and all the edges.

More precisely, it consists in firstly, replacing each vertex by a finite graph according to its label and then, replacing each edge of the initial graph by finitely many edges connecting some vertices of the graph stemming from its origin with some vertices of the graph stemming from its target. The replacement of a given edge is defined according to its label. Substitutions of all the vertices and then of all the edges are done *simultaneously*.

Definition 3 (Graph $D0L$-Systems).
In this paper, a *deterministic graph $D0L$-system* δ is defined according to a tuple $(\ell_0, L, \Delta_V, \Delta_E)$ where:

- $L = L_V \cup L_E$ is a set of labels. L_V is the set of *vertex labels* and L_E is the set of *edge labels*. And $\ell_0 \in L_V$ is the *initial symbol* or *axiom*.
- Δ_V is a mapping associating to each vertex label a finite graph whose vertices (respectively edges) are labeled over L_V (respectively L_E). These graphs are provided with an additional structure: A linear ordering of the vertex set.
- Δ_E is a mapping associating to each edge label a finite bipartite graph each vertex part of which is linearly ordered; the pair of vertex parts is directed itself. Edges are labeled on L_E and vertices are not labeled.

A graph is said to be a *δ-graph* if its vertices are labeled by elements of L_V and its edges by elements of L_E. Performing δ on a δ-graph $\Gamma = (V, L, \{R_\ell\}_{\ell \in L})$ consists in doing the following parallel transformations on vertices and edges:

- Each vertex $v \in V$ is replaced by $\Delta_V(lab(v))$.

- Let us consider an ℓ-labeled edge of Γ from an ℓ_1-labeled vertex v_1 to an ℓ_2-labeled vertex v_2. First, if the cardinality of the first vertex part of $\Delta_E(\ell)$ (respectively its second one) is different from the number of vertices of $\Delta_V(\ell_1)$ (respectively $\Delta_V(\ell_2)$), then this edge simply is deleted without generating any new edge. Otherwise, each edge e' of $\Delta_E(\ell)$ gives rise to a new connection: Let ℓ' be the label of e'. Let us assume that the origin of e' is the i-th vertex of the first vertex part of $\Delta_E(\ell)$ (respectively of its second one) and that its target is the j-th vertex of the second vertex part of $\Delta_E(\ell)$ (respectively of its first one). Then we add an ℓ'-labeled edge from the i-th vertex of the graph stemming from v_1 (respectively v_2) to the j-th vertex of the graph stemming from v_2 (respectively v_1).

Once simultaneous substitutions of all the vertices and all the edges are done, orderings become useless and are forgotten. A sequence of graphs $(\Gamma_n)_{n\geq 0}$ is said to be a *D0L-sequence* if there exists a graph *D0L*-system $\delta = (\ell_0, L, \Delta_V, \Delta_E)$ such that Γ_0 is made of only one ℓ_0-labeled vertex without loop and for all $n \geq 0$, $\Gamma_{n+1} = \delta(\Gamma_n)$.

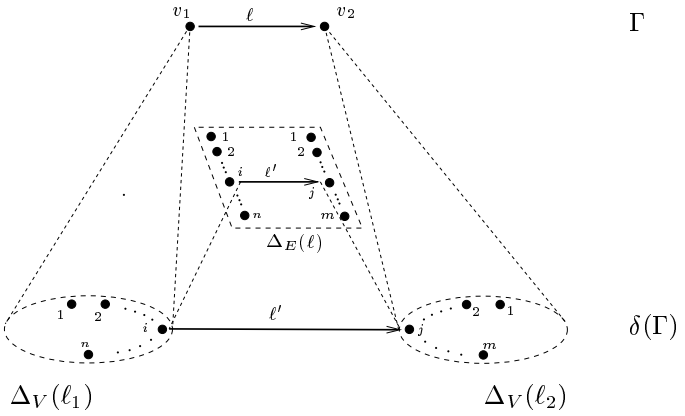


FIG. 4 Example of *D0L*-system rules

Figure 4 describes an example of graph *D0L*-system allowing to produce a sequence of binary trees. Let $L_V = \{i, e\}$ and $L_E = \{i\}$; note that $L = L_V$. The two rules described on the left hand side are the vertex replacement rules; the right hand side describes the unique edge replacement rule. A vertex labeled by i (like **i**nternal) is replaced by one vertex also labeled by i, i.e. this kind of vertex is not modified. A vertex labeled by e (like **e**xternal) splits in 3 vertices; one of them (the first) is labeled by i and the other ones by e. Concerning the edge replacement rule, it specifies that an edge is replaced by an edge of the same label, i.e. i, which connects the first vertex stemming from the origin to the first vertex stemming from the target. The first iterations are shown in Figure 6.

FIG. 5 Example of *D0L*-system rules

Figures 7 and 8 describe another example of *D0L*-system which generates square grids.

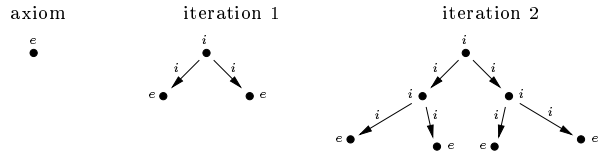


FIG. 6 First iterations of the $D0L$-system described in Figure 5

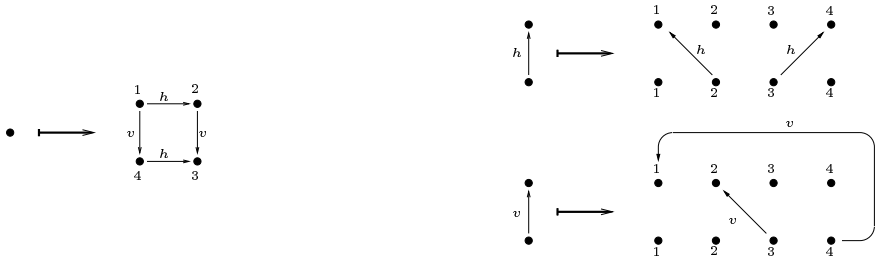


FIG. 7 A $D0L$-system which generates a sequence of grids. Note that vertex labels are not indicated as there is only one type of vertex

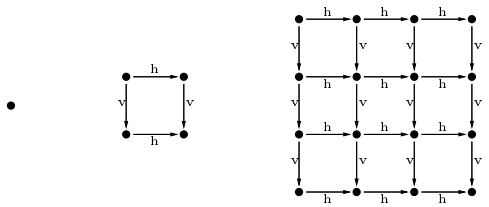


FIG. 8 The $D0L$-sequence of grids generated by the $D0L$-system described in Figure 7

3.2. Graph $D0L$-systems and Automatic Graphs

This section is devoted to the construction of the relationship between the notions of graph $D0L$-systems and of automatic graphs. This will lead us to show the equivalence between statements of Theorem 1 and Theorem 3. We deal with a special class of regularly accessible automatic graphs, which shall be called *equilibrate automatic graphs*. Such a graph is a regularly accessible automatic graph provided with an automatic graph presentation such that:

- The rational equivalence relation defining vertices is trivial, i.e. diagonal.
- Transversal edges only connect pairs of words of the same lengths.
- All the states of the transversal edge automata are final.

The sphere of radius n centered at ε of an automatic graph satisfying these conditions is simply made of the vertices represented by words of length n together with the transversal edges which connect them. The third condition guarantees that transversal edges leaves some traces, i.e. some edges which can be loops, in the spheres of lower radii.

The following result shows how automatic graphs arise as a natural mean to describe $D0L$-sequences of finite graphs. This is a crucial point in making the connection between Theorem 1 and Theorem 3, and for the proof of Theorem 2 as well.

PROPOSITION 4. *Every $D0L$-sequence of finite graphs is the sequence of the spheres of an automatic graph. Conversely, up to a modification of the labels, the sequence of the spheres of an equilibrate automatic graph is equal to a $D0L$-sequence, except possibly for the sphere of radius* 0.

Proof. Let G be an equilibrate automatic graph defined by an automatic presentation $(\nu, W, M_0, (M_\ell)_{\ell\in L})$ satisfying the above conditions. By using the usual determinizing method, we can assume without lost of generality that the automata defining G are deterministic. We construct a $D0L$-system δ_G such that for all $n \geq 0$, the $(n+1)$-th sphere of G is the result of performing δ_G on the n-th one. The set of vertex labels of δ_G is defined to be Q_W, the set of states of W; the set of edge labels is defined to be $Q_W \times (\bigcup_\ell Q_{M_\ell}) \times Q_W$.

We first modify the labeling of G: The reading of a vertex v drives W to a state q_v; let us set the new label of v to be q_v. In the same way, we define the new labels of the edges among the elements of $Q_W \times (\bigcup_\ell Q_{M_\ell}) \times Q_W$: Let e be a transversal edge of G initially labeled by ℓ; this edge connects a pair of vertices (v_1, v_2); for $i = 1, 2$, let q_i be the state to which the reading of v_i drives W; and let q_ℓ be the state to which the reading of (v_1, v_2) drives M_ℓ; we then define the label of e to be the 3-tuple (q_1, q_ℓ, q_2).

Splitting rules of δ_G are now defined according to the transitions of W and of the M_ℓ's. Let us look at vertex splitting rules: Let us fix an arbitrary linear ordering of the alphabet A of W; let q be an element of Q_W; it is now a vertex label of δ_G. Since W is deterministic, the ordering of A defines an ordering of the set of transitions of W with origin q. $\Delta_V(q)$ is then defined to be a graph with as many vertices as there are transitions with origin q in W; the ordering of these vertices is given by the ordering of these transitions as defined above; their labels, which are states of W, are defined to be the targets of the transitions with which they are associated. There are no edges in $\Delta_V(q)$.

Construction of edge splitting rules follows the same idea. Let (q_1, q_ℓ, q_2) be an edge label, where $q_i \in Q_W$ and $q_\ell \in Q_{M_\ell}$ for some ℓ. $\Delta_E(q_1, q_\ell, q_2)$ is a bipartite graph; the first part (respectively the second part) of the vertex set of $\Delta_E(q_1, q_\ell, q_2)$ is made of as many vertices as there are transitions with origin q_1 (respectively q_2) in W; as above, the ordering of these vertices is given by the ordering of these transitions. The vertices of $\Delta_E(q_1, q_\ell, q_2)$ are not labeled. Its edges are defined as following: Let (v_1, v_2) be a pair of vertices of $\Delta_E(q_1, q_\ell, q_2)$, v_1 belonging to the first part and v_2 to the second one; to these vertices correspond two symbols a_1 and a_2 of A which are the symbols associated to the transitions of W which gave rise respectively to v_1 and v_2, i.e a_i is the last symbol of the word which encodes v_i. If there is in M_ℓ a transition with origin q_ℓ which is associated to

the pair (a_1, a_2), we create an edge in $\Delta_E(q_1, q_\ell, q_2)$ from v_1 to v_2 labeled by the 3-tuple (q'_1, q'_ℓ, q'_2) where q'_1 (respectively q'_2) is the state to which a_1 (respectively a_2) drives W from q_1 (respectively q_2), and $q'_\ell \in Q_{M_\ell}$ is the state to which the pair (a_1, a_2) drives M_ℓ from q_ℓ.

We then consider the $D0L$-sequence $(\Gamma_n)_n$ generated by δ_G from the graph Γ_0 made of only one vertex labeled by the initial state q_0 of W, and for each ℓ, an edge which actually is a loop, labeled by $(q_0, q_{\ell 0}, q_0)$ where $q_{\ell 0}$ is the initial state of M_ℓ. We shall omit to verify that, up to a modification of the labels, Γ_n is exactly the n-sphere of G; this can be done by an induction which is more tedious than difficult.

Let us turn to the converse statement. Let $\delta = (\ell_0, L, \Delta_V, \Delta_E)$ be the graph $D0L$-system associated a $D0L$-sequence of finite graphs $(\Gamma_n)_n$. We define an automatic graph presentation describing this sequence in the following sense. Let us consider Γ_0, the first graph of the sequence; applying δ on Γ_0 gives rise to Γ_1. We then consider the graph made of the union of Γ_0 and Γ_1 with some additional edges which connect the vertex of Γ_0 to all the vertices of Γ_1; we then add to this graph a disjoint copy of $\Gamma_2 = \delta(\Gamma_1)$ and we link each vertex of Γ_1 to the vertices of Γ_2 stemming from it via δ; these new edges are said to be *radial*, the other ones are said to be *transversal*. By carrying on this process, we define by inductive limit an infinite graph G which is automatic. The unique vertex coming from Γ_0 in G is called the *origin* of G. We can construct an automatic graph presentation $(\nu, W, M_0, \{M_\ell\}_{\ell \in L})$ which describes it.

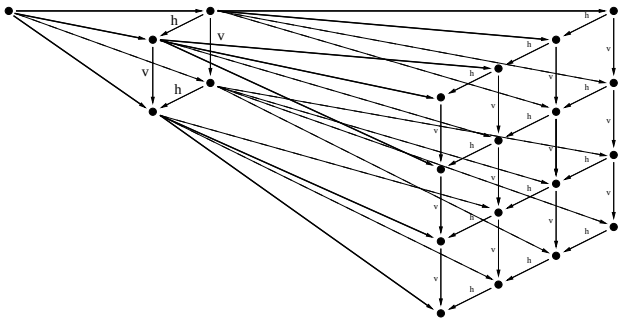


FIG. 9 The automatic graph encoding the $D0L$-sequence of grids described in Figure 8

Let us turn to the construction of W. First, let us note that there is a unique directed path of radial edges in G from the origin to a given vertex v. This path goes through all the ancestors of v. Although radial edges are not labeled, we associate to each radial edge connecting a vertex v_1 to a vertex v_2 the rank of the v_2 in $\Delta_V(\ell_1)$ where ℓ_1 is the label of v_1. Therefore, we can consider the trace of an directed path of radial edge, which is a word over the integers less than the maximum of vertices arising by splitting a vertex via δ. The trace of a path from the origin of G to a given vertex characterizes the path in question, and thus the vertex. $\mathcal{L}(W)$ is defined to be the language of such traces, and ν associates to such a trace the target vertex of the path it encodes. $\mathcal{L}(W)$ is a rational language, W is defined as follows: the states of W are defined to be the vertex labels of δ. The transitions of W are defined according to the vertex splitting rules of δ. Their definition is the converse of the one described above. We define the alphabet of W to be the set of all the integers less than the maximum of vertices arising by splitting a vertex via δ; let ℓ_v be a vertex label of δ, i.e. a state of W now; we create as many transitions of origin ℓ_v as there are vertices in the graph associated to ℓ_v by δ, i.e. $\Delta_V(\ell_v)$; such a transition is labeled by the integer which is the rank of the vertex of $\Delta_V(\ell_v)$ to which it

is associated; its target is the label of this vertex. All the states of W are final.

Since the traces of the paths of radial edges starting at the origin are in one-to-one correspondence with vertices of G, M_0 is defined to be trivial, i.e. it defines the diagonal equivalence relation. On the other hand, for each $\ell \in L_V$, M_ℓ is defined to be a copy of W with ℓ as final state.

Let us turn to the construction of M_ℓ for $\ell \in L_E$. Let us note that we shall not deal with the radial edges for which there obviously exists some letter-to-letter automaton which recognizes them.

The construction of the transversal edge automata relies on the following notion of ancestor: let us consider an ℓ-labeled transversal edge e of G connecting a vertex v_1 to a vertex v_2. Let us consider v_1' (respectively v_2') the father of v_1 (respectively v_2), i.e. the vertex which v_1 (respectively v_2) stems from. Then by construction of G, $v_1' = v_2'$ or else there is a transversal edge e' between v_1' and v_2'. We define the father of e to be v_1' if $v_1' = v_2'$ or else e'. The sequence of ancestors of e has the form $v_0, ..., v_n, e_{n+1}, ..., e_m$ where the v_i's are some vertices and the e_i's are some transversal edges; v_0 is the origin of G; let $a_i = v_i$ if $i \leq n$ and $a_i = e_i$ otherwise. Let us now consider the trace $x_1^1...x_{m+1}^1$ (respectively $x_1^2...x_{m+1}^2$) of the path of radial edges defining v_1 (respectively v_2). Let us also consider the sequence of labels $\ell_0, ..., \ell_m$ of the ancestors of e. Then, by construction of G, the existence of a_i only depends on ℓ_{i-1}, x_i^1 and x_i^2. And the existence of an ℓ-labeled transversal edge e of G from v_1 to v_2 depends only on ℓ_m, x_{m+1}^1 and x_{m+1}^1. Therefore, the principle of the construction of M_ℓ is to compute the ℓ_i's step by step by reading synchronously the x_i^1's and the x_i^2's. According to the above remark, during this computation, one just has to memorize the last ℓ_i, which allows us to construct a finite state automaton. The precise definition of M_ℓ is as follows:

First, we shall modify δ in such a way that each edge of each $\Delta_E(\ell)$ is oriented from the first part to the second one. Such a transformation is done by adding to the edge label set L_E of δ, a disjoint copy of it, $\overline{L_E} = \{\bar{\ell} \mid \ell \in L_E\}$ in order to encode edges to be inverted. In each $\Delta_E(\ell)$, each edge oriented from the second part to the first one is inverted and its label ℓ_e is replaced by $\bar{\ell}_e$. Δ_E associates to each $\bar{\ell} \in \overline{L_E}$, a bipartite graph which is a copy of $\Delta_E(\ell)$ for which the orientation of the pair of vertex parts is inverted and each edge oriented from the (new) second vertex part to the first one is inverted together with its label. We then get a new graph $D0L$-system which is denoted by δ' such that all the edges of each $\Delta_E'(\ell)$ are oriented from the first part to the second one. For each δ-graph H, we get the original $\delta(H)$ from $\delta'(H)$ by in-versing all the edges labeled by elements of $\overline{L_E}$ together with their labels.

We will now construct the family of transversal edge automata $\{M_\ell' \mid \ell \in L_E \cup \overline{L_E}\}$ associated to δ'.

We first consider a graph M constructed as following: The vertex set of M is defined to be the set of edge labels of δ', i.e. $L_E \cup \overline{L_E}$. Let us consider such an edge label ℓ_e; we add in M an edge labeled by a directed pair of integer (i, j) from ℓ_e to ℓ_e' each time that there is in $\Delta_E(\ell_e)$ an edge labeled by ℓ_e' from the i-th vertex of the first vertex part of $\Delta_E(\ell_e)$ to j-th vertex of the second vertex part: M thus encodes the edge splitting rules. We then consider the graph $\widetilde{M}$ made of one copy of M and one copy of W in which we replace the label of each edge labeled by an integer i by the pair (i, i); for each edge e of each $\Delta_V(\ell_v)$, we add an edge in $\widetilde{M}$ from $\ell_v \in Q_W$ to $\ell_e \in V_M$, where ℓ_e is the label of e, which is labeled by the pair (i, j), where i (respectively j) is the rank of the origin of e (respectively its target) in $\Delta_V(\ell_v)$. Finally, for each $\ell \in L_E \cup \overline{L_E}$, we define M_ℓ' to be a copy of $\widetilde{M}$ with the initial state of W as initial state and ℓ as final state. For each $\ell \in L_E$, it is now easy, from M_ℓ' and $M_{\bar{\ell}}'$, to construct a transversal edge automaton M_ℓ associated to the original edge label of δ: M_ℓ is defined to be the disjoint union of M_ℓ' and $M_{\bar{\ell}}'$; the label of each transition of $M_{\bar{\ell}}'$ which is a directed pair of integers (i, j) is replaced by (j, i). ∎

First-Order Logic Theories of D0L-Sequences.

Here we deal with *first order logical formulæ*. Such formulæ are constructed by using individual variables, usual boolean operators and atomic formulæ which express adjacency and labels. The first order theory of a family $\mathcal{F}$ of graphs is the set of first order formulæ which are true for all the graphs of $\mathcal{F}$.

The first order theory of a given automatic graph, i.e. the set of first order formulæ which are satisfied by it, is decidable. More generally, it has been proved in [23] that the first order theory of an *automatic relational structure* is decidable (see also [4] for extensions and complexity issues). The domain of such a logical structure is defined to be the quotient of a rational language by an equivalence relation which is recognized by a letter-to-letter automaton, as we used in Definition 1. Each relation of such a logical structure is recognized by an automaton as well. More precisely, a k-ary relation is defined by an automaton recognizing k-tuples of words belonging to the language which encodes the domain (cf. [23] for precise definitions). Automatic graphs actually are instances of automatic logical structures.

Proposition 4 together with the decidability of the first-order theory for automatic graphs turns out to implies the following result.

THEOREM 2. *The first-order theory of a D0L-sequence of finite graphs is decidable.*

Proof. Let $(\Gamma_n)_n$ be a $D0L$-sequence of finite graphs. The proof of Proposition 4 shows how to construct a regularly accessible automatic graph G whose sequence of sphere is equal to $(\Gamma_n)_n$.

Let us note that G satisfies the first and the second conditions of the definition of equilibrate automatic graphs. On the other hand, if one deletes all its radial edges, it remains automatic, but not regularly accessible. We get that the graph G' defined to be the disjoint union of all the Γ_n's is an automatic graph. Moreover, if one consider the automatic graph presentation $(\nu, W, M_0, (M_\ell)_{\ell\in L})$ for G' which is constructed from the one of G, we have that two vertices of G' stem from the same Γ_n for some n if and only if they belong to the same sphere of G, if and only if they are respectively represented by two words of $\mathcal{L}(W)$ of the same length. Such a relation obviously is recognizable by a finite letter-to-letter automaton. So, if ρ denotes a new label, we get that the graph G'' obtained from G' by adding a ρ-labeled edge between any two vertices stemming from the same Γ_n still is automatic. In other words, the disjoint union of all the graphs of $(\Gamma_n)_n$, together with the binary relation expressing for any two vertices the belonging to the same Γ_n, is an automatic relational structure.

Let us consider a first-order closed formula φ, i.e. without free variables. We shall transform φ into a formula denoted by $\Lambda(\varphi)(x)$ with one free variable such that for any vertex $\bar{x}$ of G'', $(G'', \bar{x}) \models \Lambda(\varphi)(x)$ if and only if $\Gamma_n \models \varphi$ where n is such that $\bar{x}$ stems from Γ_n. We define the Λ transformation on formulæ with free variables as follows. Let us consider a first-order formula φ possibly with free variables. Without lost of generality, we can assume that φ is in prefix form, i.e. $\varphi = Q_1 y_1 \ \ldots \ Q_k y_k \ \varphi_k$ where $Q_i = \exists$ of $\forall$ and φ_k is quantifier free. Λ is defined by induction on the number of quantifiers : If φ is quantifier free ; then $\Lambda(\varphi) = \varphi$. If there is at least one quantifier in φ, then we can assume that φ has the form $\varphi = \exists y \,.\, \varphi'(y)$ or $\varphi = \forall y \,.\, \varphi'(y)$ for some formula φ'. If $\varphi = \exists y \,.\, \varphi'(y)$ then we define inductively $\Lambda(\varphi)(x) = \exists y \,.\, \rho(x,y) \wedge \Lambda(\varphi'(y))(x)$, where the atomic formula $\rho(x,y)$ expresses that x and y both belong to the same Γ_n, i.e. they are connected by a ρ-labeled edge in G''. If $\varphi = \forall y . \varphi'(y)$ then we define $\Lambda(\varphi)(x) = \forall y . \rho(x,y) \Rightarrow \Lambda(\varphi'(y))(x)$. The verification that Λ has the good properties is an induction on k which is more tedious than difficult; it shall be omitted here.

The conclusion comes as follows: We have that : $\forall n . \Gamma_n \models \varphi$ if and if $G'' \models \forall x . \Lambda(\varphi)(x)$. Seeing that the first-order theory of automatic graphs is decidable, this last sentence is decidable. ∎

Connectivity Problem.

Let G be an equilibrate automatic graph defined by an automatic graph presentation $(W, M_0, \{M_\ell\}_{\ell \in L})$. It is easy to see that if all the spheres of G are connected then G has only one end. In order to get the converse statement, i.e. Lemma 11 to be stated bellow, we need to cut the *finite branches* of G, i.e. the paths of radial edges in G which are not prefix of any infinite branch. Indeed, it may happen that some paths made of radial edges do not lead to any infinite branch. We can avoid this phenomenon by deleting all the states of W through which there is no infinite run. This can be done in an effective way by checking the loops in W and deleting all the states of W which do not lead to any loop. Let $\overline{G}$ denote the graph obtained after this transformation. It still is automatic; and now, we can state the following fact:

LEMMA 11. *Keeping the above notations, G has one and only one end if and only if all the spheres of $\overline{G}$ are connected.*

Proof. First, $\overline{G}$ has as many ends as G has; indeed, the infinite branches of $\overline{G}$ are exactly those of G. What could happen is that some pairs of them which are equivalent in G become no longer equivalent in $\overline{G}$. Let us consider two infinite branches $\alpha_1 = v_0^1 v_1^1 ... v_k^1 ...$ and $\alpha_2 = v_0^2 v_1^2 ... v_k^2 ...$ in G, where $v_0^1 = v_0^2 = \varepsilon$. We will verify that if α_1 and α_2 are equivalent regarding G, i.e. they are connected in the complementary in G of any ball centered at ε, then for all k, there exists a path in S_k, the sphere of G centered at ε of radius k, between v_k^1 and v_k^2, which takes only vertices which are prefixes of some infinite branches. So, let us suppose that α_1 and α_2 are equivalent. For all $k' > k$, let us choose a path $p'_{k'}$ which connects $v_{k'}^1$ and $v_{k'}^2$ and which remains at a distance greater than k' from ε; we associate to $p'_{k'}$ a path $p_{k'}$ in S_k between v_k^1 and v_k^2 defined as following: We first consider the path of S_k made of the ancestors of the vertices of $p'_{k'}$ and then, we delete all the loops to get $p_{k'}$; note that $p_{k'}$ still is a path because all the states of the M_ℓ's are supposed to be final. Now, since S_k is finite, the number of paths without repetition between v_k^1 and v_k^2 is also finite; therefore, one of them appears to be a $p_{k'}$ for infinitely many k'; this implies that each of its vertices is the ancestor of infinitely many vertices and then, it is a prefix of an infinite branch. Therefore, this path is also a path in $\overline{G}$. $\overline{G}$ and G have the same number of ends.

It remains to check that $\overline{G}$ has one and only one end if and only if all its spheres are connected. We just prove that if $\overline{G}$ has only one end, then all its spheres are connected, the converse statement being easy. Let us then suppose that a sphere S'_k of $\overline{G}$ is disconnected; let z_1 and z_2 be two vertices of S'_k which are not connected in S'_k. Let us choose two infinite branches α_1 and α_2 of which z_1 and z_2 respectively are prefixes. Then these two infinite branches represent different ends; indeed, one can check as above that any path connecting them outside the ball of radius k centered at ε would define a path in S'_k. ∎

It is easy to see that the automatic graph constructed from a Turing machine in the proof of Lemma 10 satisfies the two first conditions of the definition of equilibrate automatic graphs. The third can be achieved by setting all the states of all the automata of the automatic graph presentation to be final; one can easily verify that this does not modify the set of its ends. From this, we then construct the graph needed to apply Lemma 11; we finally get Theorem 3 stated below, which was announced in Introduction. Note that only one way of Lemma 11 is needed for that; nevertheless the full equivalence shows that the statements of Theorem 1 and Theorem 3 are in fact equivalent.

THEOREM 3. *The problem of deciding whether all the finite graphs produced by iterating a graph D0L-system are connected is undecidable.*

Remark 7. This result implies that the first-order theory, when it is upgraded with a closure operator, is undecidable for graph $D0L$-systems.

3.3. Graph $D0L$-Systems in the Graph Transformation Theory

This section investigates relationships between graph $D0L$-systems and some other graph transformation formalisms. First, we show that graph $D0L$-systems emulate vertex replacement systems (see e.g. [15]). Second, we show that graph $D0L$-systems are emulated by edge grammars (see [3]).

Emulation of VR grammars by graph D0L-systems

Let us turn to VR-grammars (see [15, Chap. 1, Def. 1.3.5]).

Vertex replacement. Let $\Gamma = (V, L, \{R_\ell\}_{\ell \in L})$ be a graph and $v \in V$. Let us assume that L is the disjoint union of two sets L_V and L_E which respectively are the vertex label set and the edge label set. Following [15, Chap. 1.3], by *graph with embedding* we mean a pair (D, C) where D is a finite graph and $C \subset L_V \times L_E \times L_E \times V_D \times \{in, out\}$; C is called the *connection relation* of (D, C). The *replacement* of v by D according to C is then defined as following: v is deleted from Γ and replaced by a copy of D; for each connection instruction $(\sigma, \alpha, \beta, x, out) \in C$, any α-labeled edge in Γ from v to a σ-labeled vertex $w \neq v$ of Γ give rise to an β-labeled edge from x to w; and similarly for "in" instead of "out", where "in" refers to incoming edges while "out" refers to outgoing edges of v.

Deterministic VR grammars. Following [15, Chap. 1, Def. 1.3.5], a *edNCE-grammar* is a tuple (L_V, L'_V, L_E, P, S) where L_V is the set of vertex labels; $L'_V \subset L_V$ is the set of *terminal* vertex labels; L_E is the set of edge label; P is a set of *production*, and $S \in L_V \setminus L'_V$ is the *initial non-terminal*. A production is of the form $X \to (D, C)$ with $X \in L_V \setminus L'_V$ and (D, C) is a graph with embedding with D labeled over $L_V \cup L_E$. Let Γ and Γ' be two graphs labeled over $L_V \cup L_E$, let $p : X \to (D, C)$ be a production, and let $v \in V_\Gamma$ be a X-labeled vertex of Γ; we then write $\Gamma \Rightarrow_{v,p} \Gamma'$ if Γ' is the graph obtained from Γ by replacing v by D according to C. As usual in formal language theory, we consider $\mathcal{L}(G)$, the *graph language generated by* G, to be the set of all the graphs produced by G from the initial non-terminal S. A edNCE-grammar will be called *VR-grammar* if it is *confluent*, i.e. if $H \Rightarrow_{v_1 p_1} H_1 \Rightarrow_{v_2 p_2} H_{12}$ and $H \Rightarrow_{v_2 p_2} H_2 \Rightarrow_{v_1 p_1} H_{21}$ are two derivation chains with $v_1, v_2 \in V_H$ and $v_1 \neq v_2$, then $H_{12} = H_{21}$ (see [15, Chap. 1, Prop. 1.3.6]). Here we actually deal with a particular kind of VR-grammar: the *deterministic* VR-grammar, i.e. grammars containing at most one production associated to each non-terminal.

VR sequences. Let $G = (L_V, L'_V, L_E, P, S)$ be a deterministic VR-grammar. We consider the sequence $(\Gamma_n)_n$ of graphs defined by induction as following: Γ_0 is the graph made of only one S-labeled vertex without loop. And for each $n \geq 0$, Γ_{n+1} is obtained from Γ_n by replacing each vertex labeled by a non-terminal for which a production is defined. As G is confluent, the ordering of the derivation steps in this transformation does not matter. A sequence of graphs obtained in such a way is called a *VR-sequence* of graphs.

LEMMA 12. *Let G be a deterministic VR-grammar; let $(\Gamma_n)_n$ be the VR-sequence associated to it. Then all the graphs of $\mathcal{L}(G)$ are connected if and only if for all $n \geq 0$, Γ_n is connected.*

Proof. Seeing that for all n, $\Gamma_n \in \mathcal{L}(G)$, the direct way is trivial. For the converse statement, let us note that any graph of $\mathcal{L}(G)$ can be obtained from Γ_n with n big enough by collapsing some subgraphs in single vertices, and possibly adding some edges and modifying some labels. Indeed, let $\Gamma \in \mathcal{L}(G)$. Let us suppose that Γ can be obtained from the initial non-terminal through n derivations. Then G can be obtained from Γ_n by inversing some derivations, i.e. by collapsing some subgraphs into single vertices, adding some new edges and modifying some labels. Collapsing a subgraph of a connected graph in a single vertex leads to a connected graph. Adding some new edge to a connected graph or else modifying some labels also leads to a connected graph. Thus, if Γ_n is connected,

then it remains connected after a finite sequence of such transformations; and Γ is indeed connected. ∎

One of the main algorithmic property of VR-languages of graphs is that their monadic second-order logic theories, with quantifications over vertex sets only, are decidable (cf. [15, Chap. 1.4.2]). As we saw in the proof of Proposition 5, the connectivity can be expressed by a monadic second-order logical formula whose quantifications only are over vertex sets. This implies in particular that *there exists an algorithm to decide whether all the graphs of a VR-sequence are connected or not.*

Emulation.

PROPOSITION 5. *Up to a modification of the labels, any VR-sequence of graphs is a D0L-sequence.*

Proof. Let $(\Gamma_n)_n$ be the VR-sequence associated to a deterministic VR-grammar $G = (L_V, L'_V, L_E, P, S)$. We will construct a graph $D0L$-system $\delta = (\ell_0, L, \Delta_V, \Delta_E)$ which generates $(\Gamma_n)_n$.

First, the vertex label set of δ is the one of G, i.e. L_V. And $\ell_0 = S$. Δ_V associates to each terminal symbol ℓ a graph made of only one vertex labeled by ℓ without any edge; and to each non-terminal one, it associates the graph defined by the associated production of G, with a fixed linear ordering of its vertex set.

The edge label set of δ is defined to be $L_V \times L_E \times L_V$; for each edge e of each Γ_n, we define a new label to be (ℓ_1, ℓ_e, ℓ_2) where ℓ_1 is the label of the origin of e, ℓ_2 is the label of its target, and ℓ_e is the initial label of e; in the same way, we modify the labeling of the edges of the graphs defined by Δ_V. Let us turn to edge splitting rules; let (ℓ_1, ℓ_e, ℓ_2) be an edge label of δ; let us consider the graph made of two vertices v_1 and v_2, respectively labeled by ℓ_1 and ℓ_2 and a ℓ_e-labeled edge from v_1 to v_2. We then consider the graph obtained by replacing v_1 and v_2 according to G. We delete from it all the edges connecting pairs of vertices stemming from the same vertex v_1 or v_2. We then get a bipartite graph; each vertex part of it gets a linear ordering from the definition of $\Delta_V(\ell_1)$ and $\Delta_V(\ell_2)$. The labels of edges are modified as above and the labels of the vertices are deleted. The graph we obtain in this way is by definition $\Delta_E(\ell_1, \ell_e, \ell_2)$.

We shall omit to check that δ generates $(\Gamma_n)_n$ which is more tedious than difficult. ∎

In view of Theorem 3 there is no hope of converse for this result. Let us note that we could also see that from the fact that graph $D0L$-systems generate grids of unbounded width.

Remark 8. VR grammars can emulate *hyperedge replacement grammars* (see Section 1.2 or else [14]). And so, HR-sequences of finite graphs, i.e. a sequence of finite graphs which is generated by a deterministic hyperedge replacement grammar, can also be emulated by $D0L$-sequences.

Remark 9. Deterministic graph $D0L$-systems as considered here do not emulate parallel BNLC grammars in general (see [33] for the definition of parallel BNLC grammars), because these last ones are non-deterministic. But deterministic graph $D0L$-systems emulate *deterministic parallel BNLC grammars*, i.e. parallel BNLC grammars which have at most one production per non-terminal. Indeed, graph languages produced by deterministic parallel BNLC grammars can be generated by *BNLC grammars with parallel rewriting* (see [33] for the definition of BNLC grammars with parallel rewriting). And the graph languages produced by this last ones can also be generated by classical BNLC grammars (see [33], see also [36] for an introduction to BNLC grammars), which VR grammars emulate. According to Proposition 5, graph $D0L$-systems thus emulate deterministic parallel BNLC grammars.

Emulation of graph D0L-Systems by Edge Grammars

Edge Grammars have been introduced in [2] in the context of graph generating formalism for parallel computations (see also [3]). This section is devoted to the emulation of graph $D0L$-systems by edge grammars.

Edge Grammars. An *edge grammar* is a 4-tuple (N, T, S, P), where N is a finite set of non-terminals, $T = \{(v, w) \mid v, w \in \Sigma^*\}$ is a finite set of terminal pairs, where Σ is a finite alphabet, $S \in N$ is the start symbol, and $P = \{\alpha \to \beta \mid \alpha, \beta \in (N \cup T)^+\}$ is a finite set of productions.

We define the concatenation of pairs by the pair of concatenates: $(v, w)(x, y) = (vx, wy)$. A pair (v, w) is *derived* in an edge grammar $G = (N, T, S, P)$ if (v, w) results from S by a sequence of legal applications of productions in P and $|v| = |w|$ (see [3] for examples). This shall be denoted by $S \hookrightarrow^* (v, w)$.

The *n-th directed[5] graph $G_n(G)$ derived by G* is an directed graph with vertex set $V_n(G)$ and edge set $E_n(G)$, where

$$V_n(G) = \{v \mid |v| = n \text{ and } \exists w. S \hookrightarrow^* (u, w) \text{ or } S \hookrightarrow^* (w, u)\}$$

$$E_n(G) = \{(v, w) \mid v, w \in V_n(G) \text{ and } S \hookrightarrow^* (v, w), v \neq w\}.$$

The sequence $(G_n(G))_n$ is called the *graph family* derived by G.

Emulation.

PROPOSITION 6. *Let $(\Gamma_n)_n$ be a graph $D0L$-sequence such that for all n, Γ_n has no isolated vertex neither loop, i.e. an edge connecting a vertex to itself, then, up to a modification of the labels, $(\Gamma_n)_n$ can be generated by an edge grammar.*

Proof. Let $\delta = (\ell_0, L = L_V \cup L_E, \Delta_V, \Delta_E)$ be a graph $D0L$-system which generates $(\Gamma_n)_n$. Let $G = (N, T, S, P)$ be the edge grammar defined as follows:

- $N = L$
- $\Sigma = [1, \max_{\ell \in L_V} |\Delta_V(\ell)|]$
- $S = \ell_0$
- $T = \Sigma^2 \cup \{(\varepsilon, \varepsilon)\}$
- P is made of the following families of productions:

  (i) For each $\ell \in L_V$ and for each i, ℓ_v such that the i-th vertex of $\Delta_V(\ell)$ is labeled by ℓ_v, we add the rule:
  $$\ell \to (i, i)\ell_v$$

  (ii) For each $\ell \in L_V$ and for each i, j, ℓ_e such that $\Delta_V(\ell)$ has an ℓ_e-labeled edge from its i-th vertex to its j-th one, we add the rule:
  $$\ell \to (i, j)\ell_e$$

  (iii) For each $\ell \in L_E$ and for each i, j, ℓ_e such that $\Delta_E(\ell)$ has an ℓ_e-labeled edge from the i-th vertex of its first vertex part to the j-th vertex of its second vertex part, we add the rule:
  $$\ell \to (i, j)\ell_e$$

[5] In [3], the graphs generated by edge grammars are undirected. For our purpose, we need to consider directed graphs. The reader can verify the undirected graphs generated by an edge grammar is the symmetrized of the directed graphs generated by the same edge grammar according to this definition.

(iv) For each $\ell \in L_E$, we add the rule:

$$\ell \to (\varepsilon, \varepsilon)$$

The us consider $(G_n)_n$, the graph family derived by G.

Let us consider the following induction hypothesis:

- $V_{\Gamma_n} = V_{G_n}$, *which we suppose to live in* Σ^*.
- $E_{\Gamma_n} = E_{G_n}$.
- *For each* $v \in V_{G_n}$ *of label* ℓ_v, *we have* $\ell_0 \hookrightarrow^* (v, v)\ell_v$.
- *For each* $v_1, v_2 \in V_{G_n}$, *if there is an* ℓ_e*-labeled edge from* v_1 *to* v_2, *then we have* $\ell_0 \hookrightarrow^* (v_1, v_2)\ell_e$.

Base Case. Let us recall that Γ_0 is defined to be the graph with only one ℓ_0-labeled vertex without any edge. So up to a modification of the label, $\Gamma_0 = G_0$. This proves the two first conditions. The third condition becomes $\ell_0 \hookrightarrow^* \ell_0$ which is trivially true. The fourth condition is also trivially verified since there is no edge.

Induction Case. Let us assume the induction hypothesis for some integer n. According to δ, each ℓ-labeled vertex v of Γ_n is replaced by $\Delta_V(\ell)$. According to the induction hypothesis, we have $\ell_0 \hookrightarrow^* (v, v)\ell$; so, applying the rules of type (i) gives $\ell_0 \hookrightarrow^* (vi, vi)\ell_i$ for each $i \in [1, |\Delta_V(\ell)|]$, where ℓ_i is the label of the i-th vertex of $\Delta_V(\ell)$. The i-th vertex of the copy of $\Delta_V(\ell)$ replacing v is encoded by the word vi. This proves the third condition for $n+1$.

We can divide the edge set of Γ_n into two parts: edges coming from the replacement of a vertex and edges coming from the replacement of an edge. The first ones are produced by the productions of type (ii) applied on words of form $(v, v)\ell$ for $\ell \in L_V$, followed by application of production of type (iv) to delete non-terminals. The second ones are produced by the productions of type (iii) applied on words of form $(v_1, v_2)\ell_e$, followed by application of production of type (iv) to delete non-terminals. Indeed, by induction, for each edge of Γ_n, there is such a word; thus applying productions of type (iii) produces all the edges stemming from it. Let us note that we get the fourth condition of the induction hypothesis for $n+1$. So, we have that that all the edges of Γ_{n+1} are indeed in G_{n+1}. Conversely, the last operations are the only ones which produce edges connecting words of length $n+1$. So, there is no more edge in G_{n+1} than in Γ_{n+1}.

The assumption according to which Γ_{n+1} has no isolated vertex ensures that all the vertex of Γ_{n+1} are in G_{n+1}, seeing that the vertices of G_n are defined to be the words belonging to some edge produced by the grammar, and that all the edges of Γ_{n+1} have been produced. This proves the first and the second conditions of the induction hypothesis for $n+1$. ∎

Remark 10. Let us note that the proof of Proposition 6 uses an edge grammar of type 3 (see [3]); This is the weaker one according to the power of expression. So, graph $D0L$-systems are located at the bottom of the edge grammar hierarchy (see [3]).

CONCLUSION

We have seen how an infinite automatic graph or an infinite automatic structure encodes the whole structure of a sequence of finite graphs which is generated by some graph transformation, i.e. HR grammars, VR grammars or else $D0L$-systems (Lemma 4). This fact allowed us to prove the decidability of first-order theory for these sequences (Theorem 2) which was already known for HR grammars and VR grammars. On the other hand, it also allowed us to apply the undecidability of the problem of ends for automatic graphs to clarify the question of the possible extension of this result, i.e. the satisfiability of first-order formululæ when they are upgraded with a closure operator, becomes undecidable

for $D0L$-sequences (Theorem 3).

This study leaves opened some questions:

- Can a $D0L$-sequence of finite graphs of *uniformly bounded tree-width* (respectively *uniformly bounded clique-width*, see [10] for a definition of this) be generated by a HR grammar (respectively a VR grammar) ?
- Is it possible to characterize in term of clique-width the $D0L$-sequences of finite graphs for which monadic second-order logic is decidable ?
- One can show that the graph transformation defined by a $D0L$-system can be encoded by a monadic second-order definable transduction. Conversely, is it possible to characterize the definable transductions which are equivalent to some $D0L$-system ?

REFERENCES

[1] L. W. Ahlfors and L. Sario. *Riemann Surfaces.* Princeton University Press, 1960.

[2] F. Berman. Edge Grammars and Parallel Computation. In *Proceedings of the 21st Annual Allerton Conference on Communication, Control and Computing*, pages 214–223, 1983.

[3] F. Berman and G. Shannon. Representing graph families with edge grammars. *Information Sciences*, 70:241–269, 1993.

[4] A. Blumensath and E. Grädel. Automatic Structures. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000*, pages 51–62, 2000.

[5] A. Carbone and S. Semmes. *A Graphic Apology for Symmetry and Implicitness.* Oxford University Press, 2000.

[6] D. Caucal. On infinite transition graphs having decidable monadic theory. In *ICALP'96 - LNCS*, volume 1099, pages 194–205, 1996.

[7] A. D. M. Clow. *Ends of Groups – A Computational Approach.* PhD thesis, Warwick University, UK, 2000.

[8] B. Courcelle. The Monadic Second-order Logic of Graphs II : Infinite Graphs of Bounded Width. *Math. Syst. Theory*, 21:187–221, 1989.

[9] B. Courcelle. The Monadic Second-order Logic of Graphs IV : Definability Properties of Equational Graphs. *Annals of Pure and Applied Logic*, 49:193–255, 1990.

[10] B. Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.

[11] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle Rewriting Hypergraph Grammars. *JCSS*, 46:218–270, 1993.

[12] M. De Does and A. Lindenmayer. Algorithms for the Generation and Drawing of Maps Representing Cell Clones. In *GGACS'82*, volume 153 of *Lect. Notes in Comp. Sci.*, pages 39–57. Springer-Verlag, 82.

[13] W. Dicks and M.J. Dunwoody. *Groups Acting on Graphs*, volume 17 of *Cambridge Studies in Advanced Mathematics.* Cambridge University Press, 1989.

[14] F. Drewes, H.-J. Kreowski, and Habel A. Hyperedge Replacement Graph Grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 95–162. World Scientific, 1997.

[15] J. Engelfriet and G. Rozenberg. Node Replacement Graph Grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 1–94. World Scientific, 1997.

[16] D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson, and W.P. Thurston. *Word processing in groups*. Jones and Bartlett, 1992.

[17] V. Gerasimov. Detecting Connectedness of the Boundary of a Hyperbolic Group. Preprint, 1999.

[18] E. Ghys and P. de la Harpe. *Sur les groupes hyperboliques d'après Mikhael Gromov*, volume 83 of *Progress in Mathematics*. Birkhäuser, 1990.

[19] Y. Gurevich. Monadic Second-Order Theories. In J. Barwise and Feferman; S., editors, *Model Theoretic Logic*, pages 479–506. Springer, 1985.

[20] R. Halin. Über Unendliche Wege in Graphen. *Math. Ann.*, 157:125–137, 1964.

[21] P. K. Hooper. The undecidability of the turing machine immortality problem. *J. Symbolic Logic*, 31:219–234, 1966.

[22] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.

[23] B. Khoussainov and A. Nerode. Automatic Presentations of Structures. In *Logic and Computational Complexity (Indianapolis, IN, 1994)*, volume 960 of *Lecture Notes in Comput. Sci*, pages 367–392. Springer, Berlin, 1995.

[24] A. Lindenmayer. An Introduction to Parallel Map Generating Systems. In *GGACS'86*, volume 291 of *Lect. Notes in Comp. Sci.*, pages 27–40. Springer-Verlag, 1986.

[25] A. Lindenmayer and G. Rozenberg. Parallel generation of maps: developmental systems systems for cell layers. In *GGACS'78*, volume 73 of *Lect. Notes in Comp. Sci.*, pages 301–316. Springer-Verlag, 1978.

[26] O. Ly. On Effective Decidability of the Homeomorphism Problem for Non-Compact Surfaces. *Contemporary Mathematics - Amer. Math. Soc.*, 250:89–112, 1999.

[27] O. Ly. Automatic Graphs and Graph $D0L$-systems. In M. Nielsen and B. Rovan, editors, *MFCS'2000, Lect. Notes in Comp. Sci.*, volume 1893, pages 539–548, 2000.

[28] W. S. Massey. *Algebraic Topology : An Introduction*, volume 56 of *Graduate Text in Mathematics*. Springer, 1967.

[29] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[30] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[31] L. Pelecq. *Isomorphismes et automorphismes des graphes context-free, équationnels et automatiques*. PhD thesis, Bordeaux I, 1997.

[32] N. Robertson and P. Seymour. Some New Results on the Well-Quasi Ordering of Graphs. *Annals of Dicrete Math.*, 23:343–354, 1984.

[33] Röder, H.-J. Parallel BNLC graph grammars. In G. Rozenberg and A. Salomaa, editors, *Development in Language Theory*, pages 438–449. World Scientific, Singapore, 1994.

[34] H. Jr Rogers. *Theory of Recursive Functions and Effective Computability*. Series in Higher Mathematics. McGraw-Hill, 1967.

[35] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L-Systems*. Academic Press, 1980.

[36] G Rozenberg and E. Welzl. Boundary nlc graph grammars. *Inform. and Control*, 69:136–167, 1986.

[37] G. Sénizergues. Definability in weak monadic second-order logic of some infinite graphs. In *Dagstuhl seminar on Automata theory: Infinite computations. Wadern, Germany*, volume 28, pages 16–16, 1992.

[38] G. Sénizergues. An effective version of Stallings' theorem in the case of context-free groups. In *ICALP'93*, pages 478–495. Lect. Notes Comp. Sci. 700, 1993.

[39] J.R. Stallings. On torsion-free groups with infinitely many ends. *Ann. of Math.*, 88:312–334, 1968.

APPENDIX A: SELF-STABILIZING MACHINES

Here we describe the details of the working of $\widetilde{T}$. And we state some technical lemmas. $\widetilde{T}$ has two tape heads which are denoted by t_1 and t_2; its tape alphabet is $\Gamma_{\widetilde{T}} = \Gamma_T \cup \{\#, \$\} \cup Q_T \cup \{\square\}$ where Q_T is the set of state of T, and Γ_T is its alphabet exept its blank symbol; recall that $\widetilde{T}$ cannot write the blank symbol. For relative simplicity of the description of the behavior of $\widetilde{T}$, we need to define a basic language. We first define some basic instructions:

- `read`(t_i) denotes the character read by t_i. The position of t_i does not change.
- `write`(t_i, u) where $u \in (\Gamma_{\widetilde{T}} \backslash \{\square\})^*$ means that t_i write u on the tape and moves to the cell next to the last character it has written.
- `move-left`(t_i) means that t_i moves to the left. If t_i is scanning the first cell of the tape then it does not move
- `move-right`(t_i) means that t_i moves to the right.
- BEGINNING$_i$ is a boolean value meanning that t_i is scanning the first cell.

We will also use some simple control structures:

- `if` ... ... `else` ...
- `switch` ...
  - `case` ...
    ...
  - `case` ...
    ...
  - `default`
    ...
- `goto` ...

Some comments signalized by two slashs (//) are written through the algorithm.

The algorithm performed by $\widetilde{T}$ is parted in three procedures. The description of it uses variables varying in some finite sets. For instance, some of them take their values in Q_T. We make an extensive use of macros which admit sometimes parameters. Macros are supposed to be replaced by their codes each time they appear. The reader will see that the actual definition of $\widetilde{T}$ (i.e. in terms of set of states, transitions, initial state and final states) can be set up step by step, translating line after line. Each line gives rise to several states according to possible values of each variables. We do not give any rigorous method to do such a construction which actually is very tedious but assuredly possible.

Main algorithm:

Step 1:
// $\widetilde{T}$ starts by putting the initial instantaneous description of T on the tape.
`Initialisation`();
Step 2:
// $\widetilde{T}$ verifies that the content of the tape is of the form $\#\mathsf{Id}_0\#\mathsf{Id}_1\#...\#\mathsf{Id}_k\square\square...$
// where $\mathsf{Id}_0 = q_0$ and $\mathsf{Id}_i \rightarrow \mathsf{Id}_{i+1}$. It also verifies that there is no repetition
// of instantaneous description.
`Verify-History`();
Step 3:
// Then it computes the instantaneous description following the last one written on the tape.
// If it does not already appear, it write it on the tape in last position.
`Next-ID`();

goto Step 2;

Step 1 - **Initialisation:**
$\widetilde{T}$ starts by writing the initial instantaneous description on the tape.

```
Initialisation():
        write(t_1, #q_init);
        Back-to-the-Beginning(t_1);
        end;
```

Step 2 - **Verify History:**
This procedure verifies that the tape contains the sequences of successive instantaneous descriptions of T during a finite computation from q_{init}. It also verifies that no instantaneous description appears twice. Note that after the execution of this procedure, if it does not lead to the BUG state, t_1 is scanning the final instantaneous description i.e. the separator symbol # just before it while t_2 is at the beginning of the tape.

```
Verify-History():
        Back-to-the-Beginning(t_1);
        if (Deal-With(t_1, #q_init, VERIFY-MODE) = FALSE)     // one verifies that the beginning of the
            BUG;                                              // tape is of the form
        if (read(t_1) ≠ □, #)                                 // #q_init#... or #q_init□...
            BUG;
        Back-to-the-Beginning(t_1);
        Back-to-the-Beginning(t_2);
        move-right(t_2); move-right(t_2);
(i)     if (read(t_2) = □)
            Back-to-the-Beginning(t_2);
            end;
        Look-Forward();
        if (Simulation(VERIFY-MODE) = FALSE)
            BUG;
        goto (i);
```

This procedure verifies that the instantaneous description scanned by t_1 which is denoted by Id_1 does not appear again after the position of t_2.

```
Look-Forward():
        write(t_2, $);                                        // t_2 puts a mark to keep its position
(i)     move-right(t_1);                                      // in memory
        switch
            • case (read(t_1) = read(t_2) and read(t_1) ∈ Q_T ∪ Γ_T)
                    move-right(t_2);
                    goto (i);
            • case (read(t_1) ∈ {#, □, $} and read(t_2) ∈ {#, □})   // t_1 and t_2 have read the same
                    BUG;                                      // instantaneous description
        move-left(t_1); GotoPreviousID(t_1);                  // t_1 returns to the beginning of Id_1.
        GotoNextID(t_2);                                      // t_2 moves to the next
        if (read(t_2) = #)                                    // instantaneous description.
            move-right(t_2);
            goto (i);
```

```
(ii)    move-left(t2);                  // t2 has reached the end of the tape.
        if (t2 ≠ $)                     // It returns to the mark and replace it
            goto (ii);                  //by #.
        write(t2, #);
        move-left(t2);
        end;
```

Step 3 - **Next instantaneous description:**

The aim of this procedure is to compute the next instantaneous description of T and add it to the tape if it is not already stocked. One assume that the tape contains a sequence of instantaneous descriptions of T separated by separator symbols #. One also assume that t_1 is scanning a separator symbol # followed by an instantaneous description Id_1. Let Id_2 be the result of a one-step computation of T from Id_1. The procedure searchs Id_2 among the instantaneous descriptions following the position of t_2. If it does not find it, then it writes it at the end of the list.

```
Next-ID():
(i)     if (read(t2) = □)               // when t2 has reached the end of the
            Simulation(WRITE-MODE);     // tape, one writes Id2 and the
            end;                        // procedure terminates.
        if (Simulation(VERIFY-MODE) = TRUE)   // Otherwise one checks that
            end;                        // Id2 is not already on the tape.
                                        // If it is, then there is nothing to do.
        move-left(t1);                  // t1 return to the beginning of Id1.
        GotoPreviousID(t1);
        GotoNextID(t2);
        goto (i);
```

The Main Macro: Simulation of the Computation of T.

In this macro, $\widetilde{T}$ simulates the computation of T. The parameter state can take as values VERIFY-MODE or WRITE-MODE.

One suppose that t_1 is scanning a separator symbol # which is followed by an instantaneous description of T denoted by Id_1. If this is not the case, the macro returns FALSE.

In the VERIFY-MODE, one suppose that the symbol scanned by t_2 which have to be # is followed by an instantaneous description of T denoted by Id_2. $\widetilde{T}$ verifies that Id_2 is the result of a one-step computation of T from Id_1. The macro returns TRUE or FALSE whether it is true of false. If the result is TRUE, t_1 (t_2 respectively) is scanning the cell next to Id_1 (Id_2 respectively) which have to be a blank symbol or a separator symbol.

In the WRITE-MODE, t_2 just writes $\#\mathsf{Id}_2$.

```
Simulation(state):
        if (read(t1) ≠ #)
            return FALSE;
        move-right(t1);
        if (Deal-With(t2, #, state) = FALSE);
            return FALSE;

        c1 ← read(t1); move-right(t1);      // The three variables c1, c2 and c3
        if (c1 ∉ QT ∪ ΓT)                   // represent a window of length 3
            return FALSE;                   // which scans Id1.
        c2 ← read(t1); move-right(t1);      // This window scans Id1 from left to right.
        if (c2 ∉ QT ∪ ΓT ∪ {#, □})          //t2 checks or writes Id2.
```

```
        return FALSE;

(i)     c3 ← read(t1); move-right(t1);
        if (c3 ∉ QT ∪ ΓT ∪ {#, □})
            return FALSE;
        switch
            • case c1 ∈ QT and c2 ∉ QT:                    // The state of Id1 occurs in first position
                    if (c2 = #)
                            c2 ← □;
                    (c, q, D) ← δT(c1, c2, 1);
                    if (D = left, stay)
                            if (Deal-With(t2, qc, state) = FALSE)
                                    return FALSE;
                    else
                            if (Deal-With(t2, cq, state) = FALSE)
                                    return FALSE;
                    switch
                            • case c2 = □:
                                    move-left(t1); move-left(t1);
                                    if (read(t2) ≠ #, □)
                                            return FALSE;
                                    return TRUE;
                            • case c2 ∈ ΓT:
                                    if (c3 ∈ {#, □})
                                            move-left(t1);
                                            if (read(t2) ≠ #, □)
                                                    return FALSE;
                                            return TRUE;
                                    else
                                            if (Deal-With(t2, c3, state) = FALSE)
                                                    return FALSE;
                                            goto (ii);
            • case c1 ∈ ΓT and c2 ∈ QT and c3 ∉ QT:        // The state of Id1 is c2.
                    if (c3 = #)
                            c3 ← □;
                    (c, q, D) ← δT(c2, c3, 0);
                    switch
                            • case D = left:
                                    if (Deal-With(t2, qc1c, state) = FALSE)
                                            return FALSE;
                            • case D = stay:
                                    if (Deal-With(t2, c1qc, state) = FALSE)
                                            return FALSE;
                            • case D = right:
                                    if (Deal-With(t2, c1cq, state) = FALSE)
                                            return FALSE;
                    switch
                            • case c3 = □:
                                    move-left(t1);
                                    if (read(t2) ≠ #, □)
                                            return FALSE;
                                    return TRUE;
                            • case c3 ∈ ΓT:
                                    goto (ii);
```

```
        • case c1, c2 ∈ Γ_T:                          // The state of Id1 still is on the right.
              if (Deal-With(t2, c1, state) = FALSE)    // Then one shifts the window
                    return FALSE;                      //to the right.
              c1 ← c2; c2 ← c3;
              goto (i);
        • default:
              return FALSE;

(ii)    if (read(t1) ∉ #, □)                           // Once the state of Id1 has been found,
            if (read(t1) ∉ Γ_T)                        // it remains to copy the following
                  return FALSE;                        // symbols, taking care they are in Γ_T.
            if (Deal-With(t2, read(t1)), state) = FALSE)
                  return FALSE;
            move-right(t1);
            goto (ii);
        if (read(t2) ≠ #, □)
            return FALSE;
        return TRUE
```

Macros:

The following macro moves t_i at the beginning of the tape.

```
Back-to-the-Beginning(t_i):
(i)     if (not BEGINNING_i)
            move-left(t_i);
            goto (i);
        end;
```

The following macro moves t_i to the first separator symbol or blanc symbol occuring on the right hand side of its position. Note that if t_i is already scanning a separator symbol or a blanc symbol then it does not move.

```
GotoNextID(t_i):
(i)     if (read(t_i) ≠ #, □)
            move-right(t_i);
            goto (i);
        end;
```

The following macro moves t_i to the first separator symbol occuring on the left hand side of its position. If t_i does not find any separator symbol before reaching the beginning of the tape, then $\tilde{T}$ goes into the BUG state. Note that if t_i is already scanning a separator symbol then it does not move.

```
GotoPreviousID(t_i):
(i)     if (BEGINNING_i)
            BUG;
        if (read(t_i) ≠ #)
            move-left(t_i);
            goto (i);
        end;
```

The following macro verifies that the word u is written on the tape from the position of t_i or writes u on the tape whether the parameter state is equal to VERIFY-MODE or WRITE-MODE. After a normal execution i.e. an execution returning TRUE, the macro leaves t_i scanning the cell next to u.

```
Deal-With(t_i, u, state):                          // Let u = u_0...u_n ∈ Γ*_T̃.
    switch
    • case (state = VERIFY-MODE)
        if (read(t_i) ≠ u_0)
              return FALSE;
        move-right(t_i);
        ...
        if (read(t_i) ≠ u_n)
              return FALSE;
        move-right(t_i);
    • case (state = WRITE-MODE)
        write(t_i, u);
    return TRUE;
```

The proofs of following lemmas are easy but rather tedious. They can be checked by a rigorous examination of the description of $\widetilde{T}$.

LEMMA 13 (Behavior of Simulation).
The procedure Simulation(VERIFY-MODE) *returns* TRUE *if and only if, at the beginning of its execution, t_1 (t_2 respectively) scans a separator symbol # preceding an instantaneous description Id_1 (Id_2 respectively) which is ended by a separator symbol # or a blank symbol $\square$, and such that $\mathsf{Id}_1 \underset{T}{\rightarrow} \mathsf{Id}_2$. After an execution returning* TRUE, *t_1 (t_2 respectively) scans the cell next to Id_1 (Id_2 respectively).*

The procedure Simulation(WRITE-MODE) *returns* TRUE *if and only if, at the beginning of its execution, t_1 scans a separator symbol # preceding an instantaneous description Id_1 which is ended by a separator symbol # or a blank symbol $\square$. In this case t_2 writes $\#\mathsf{Id}_2$ on the tape where Id_2 is the instantaneous description produced from Id_1 by a one-step computation of T.*

LEMMA 14 (Behavior of Verify-History).
Let $\widetilde{\mathsf{Id}}$ be an instantaneous description of $\widetilde{T}$. After a finite computation from $\widetilde{\mathsf{Id}}$, $\widetilde{T}$ eventually enters into the procedure Verify-History. *Let $\widetilde{\mathsf{Id}}'$ be the instantaneous description describing T at this moment. And if $\widetilde{T}$ succeeds in performing* Verify-History *on $\widetilde{\mathsf{Id}}'$ without going into the* BUG *state, then the tape content described by $\widetilde{\mathsf{Id}}'$ is of the form $\#\mathsf{Id}_0\#\mathsf{Id}_1\#...\#\mathsf{Id}_k$ where $\mathsf{Id}_0 = q_{\mathsf{init}}$ is the initial instantaneous description of T; for all $i \in \{0, k-1\}$: $\mathsf{Id}_i \underset{T}{\rightarrow} \mathsf{Id}_{i+1}$ and for all $i \neq j$: $\mathsf{Id}_i \neq \mathsf{Id}_j$.*