On external presentations of infinite graphs

Christophe Morvan

On external presentations of infinite graphs

Christophe Morvan
Université Paris-Est
INRIA Centre Rennes - Bretagne Atlantique
Campus de Beaulieu, 35042 Rennes, France
christophe.morvan@irisa.fr

The vertices of a finite state system are usually a subset of the natural numbers. Most algorithms relative to these systems only use this fact to select vertices.

For infinite state systems, however, the situation is different: in particular, for such systems having a finite description, each state of the system is a configuration of some machine. Then most algorithmic approaches rely on the structure of these configurations. Such characterisations are said *internal*. In order to apply algorithms detecting a structural property (like identifying connected components) one may have first to transform the system in order to fit the description needed for the algorithm. The problem of internal characterisation is that it hides structural properties, and each solution becomes *ad hoc* relatively to the form of the configurations.

On the contrary, *external* characterisations avoid explicit naming of the vertices. Such characterisation are mostly defined via graph transformations.

In this paper we present two kind of external characterisations: deterministic graph rewriting, which in turn characterise regular graphs, deterministic context-free languages, and rational graphs. Inverse substitution from a generator (like the complete binary tree) provides characterisation for prefix-recognizable graphs, the Caucal Hierarchy and rational graphs. We illustrate how these characterisation provide an efficient tool for the representation of infinite state systems.

1 Introduction

Infinite graphs are a very general way to define infinite state systems. There are several means to define such infinite graphs: internal characterisations which relies on some machine: pushdown systems [19, 21], higher order pushdown systems [5], Petri nets [20], automatic and rational graphs [3, 16]. These internal characterisations are very efficient to prove properties of these systems, but they provide many restrictions on the names and definition of the states of these systems. For example, the set of vertices of a rational graph is a rational set of words, still, having a context-free set of vertices does not affect the structure of a graph.

In order to have a more direct access to the structure of such graph families, external characterisations have been introduced. These characterisation avoid explicit definition of the vertices. From a general perspective these characterisation are based on graph transformations. Meaning that it is simpler to introduce a suitable naming for the vertices depending on the problem. Also these approaches often allow nice proofs for structural properties.

There are mainly two kind of approach to externally define graph families: algebraic graph transformation (like inverse rational substitution) from an original graph (like the complete binary tree). This technique was first used by Caucal, [6], it allowed him to prove in a very elegant way the decidability of the monadic second order of the prefix-recognizable graph (a nice reformulation of this result in terms of monadic transduction is presented in [15]). This technique has given rise to the so-called Caucal hierarchy: [7]. Graph unfolding (the operation of transforming a graph into a tree) preserves the decidability of MSO theory (see [13]) and Caucal proved that unfolding prefix-recognizable graphs produces

Axel Legay and Azadeh Farzan(Ed.) : International Workshop on Verification of Infinite-State Systems (INFINITY 2009)
EPTCS 13, 2009, pp. 23–35, doi:10.4204/EPTCS.13.4

trees which are not prefix-recognizable graphs, and applying inverse rational substitution and unfolding alternatively generates a strict hierarchy of infinite graphs families having decidable MSO theories. Recently Carayol and Wöhrle showed that this hierarchy coincides in precise sense to the graphs of higher order pushdown automata: [5]. The rational graphs are a family of infinite graphs characterising context-sensitive languages, and defined by labelled rational transducers [16, 18]. They are characterised by similar external characterisation. One of which is by inverse *finite* substitution from some very general rational graph whose first order theory is undecidable.
The second kind of external characterisation is done by inductive graph transformations, and more precisely graph rewriting. The graph grammars are a classical tool to define infinite families of finite graphs. In [12], Courcelle employed deterministic hyperhedge replacement graph grammars (HR-grammars) to define the regular graphs. It turns out that these graphs are very close to graphs of pushdown automata [10], but enable very elegant proofs for structural properties like accessibility. [8] provides very thorough survey for these graphs. Interestingly the deterministic graphs generated by such grammars correspond precisely to deterministic context-free languages, this enables a generalisation of visibly pushdown languages (see [1]) defined in [9]: every deterministic context-free language belongs to a Boolean algebra of deterministic context-free languages which contains every regular languages. Earlier Colcombet, in [11], defined the class of graphs generated by vertex replacement grammars with product. These graphs have a decidable first order theory with accessibility. In [17], the author introduces contextual graph grammars characterising rational graphs, and thus context-sensitive languages.
In this paper we propose a detailed survey of these results as well as a couple of enlightening examples of the interest of working with external characterisations. The first part of the paper examines graph families defined from a generator: prefix-recognizable graphs, the Caucal hierarchy and rational graphs. The second part examines graph rewriting systems: regular graphs, synchronised graphs and again, rational graphs.

2 Preliminaries

2.1 Mathematical notations

For any set E, its powerset is denoted by 2^E; if it is finite, its size is denoted by $|E|$. Let the set of non-negative integers be denoted by $\mathbb{N}$, and $\{1,2,3,\ldots,n\}$ be denoted by $[n]$. A monoid M is a set equipped with an associative operation (denoted $\cdot$) and a (unique) neutral element (denoted ε). A monoid M is *free* if there exist a finite subset A of M such that $M = A^* := \bigcup_{n\in\mathbb{N}} A^n$ and for each $u \in M$ there exists a unique finite sequence of elements of A, $(u(i))_{i\in[n]}$, such that $u = u(1)u(2)\cdots u(n)$. Elements of a free monoid will be called words. Let u be a word in M, $|u|$ denotes the length of u and $u(i)$ denotes its ith letter.
In order to define formally graph grammars, we recall some elements on hypergraphs. Let F be an alphabet ranked by a mapping $\rho : F \to \mathbb{N}$, this mapping associates to each element of F its *arity*. Furthermore, for a ranked alphabet F, we denote by F_n the set of symbols of arity n. Now given V an arbitrary set, a *hypergraph* G is a subset of $\cup_{n\geqslant 1} F_n V^n$. The vertex set of such a hypergraph is the set $V_G = \{v \in V \mid FV^*vV^* \cap G \neq \emptyset\}$, in our setting, this set is either finite or countable. A hyperarc of arity n is denoted by $f\ v_1\ v_2\ \cdots\ v_n$.

Graphs

A (simple oriented labelled) *graph* G over V with arcs labelled in F_2 is a subset of F_2VV. An element *ast* in G is an *arc* of *source s, target t* and *label a* (s and t are *vertices* of G). We denote by $Dom(G)$, $Im(G)$

and V_G the sets respectively of sources, targets and vertices of G. Each arc *ast* of G is identified with the labelled transition $s \xrightarrow[G]{a} t$ or simply $s \xrightarrow{a} t$ if G is understood.

A graph G is *deterministic* if distinct arcs with same source have distinct label: $r \xrightarrow{a} s \ \wedge \ r \xrightarrow{a} t \ \Rightarrow \ s = t$. The set $2^{{F_2}^+ VV}$ of graphs with vertices in V, labelled by elements of ${F_2}^+$, is a semigroup for the *composition relation*: $G \cdot H := \{r \xrightarrow{a \cdot b} t \mid \exists s, r \xrightarrow[G]{a} s \wedge s \xrightarrow[H]{b} t\}$ for any $G, H \ \subseteq V \times {F_2}^+ \times V$. The relation $\xrightarrow[G^+]{u}$ denoted by $\overset{u}{\underset{G}{\Longrightarrow}}$ or simply $\overset{u}{\Longrightarrow}$ if G is understood, is the existence of a *path* in G labelled u in ${F_2}^+$. A vertex $q \in V_G$ is *reachable* from a vertex $p \in V_G$ if $p \overset{u}{\underset{G}{\Longrightarrow}} q$ for some $u \in {F_2}^+$.

For any subset L of ${F_2}^+$, we denote by $s \overset{L}{\Rightarrow} t$ that there exists u in L such that $s \overset{u}{\Rightarrow} t$.

A *graph morphism* g is a mapping from a graph G to a graph G' such that if there is an arc $u \xrightarrow[G]{a} v$, then there is an arc $g(u) \xrightarrow[G']{a} g(v)$. A graph *isomorphism* is a graph morphism which is a bijection between the vertex sets.

In the following we will consider first algebraic transformations from a generator, then we will examine graphs rewriting system defining infinite families of graphs.

3 Algebraic graph transformations

Inverse substitution

A *substitution* over a free monoid X^* is a morphism $\varphi : \Sigma^* \to 2^{X^*}$, which associates to each letter in Σ a language in X^*. For a class $\mathscr{C}$ of languages in * (for example finite languages or regular languages), a $\mathscr{C}$ substitution is such that the image of each element of Σ is a language in $\mathscr{C}$.

We denote by $\overline{X}$ the set $\{\overline{a} \mid a \in X\}$, and we say that $x \xrightarrow{\overline{a}} y$ if $y \xrightarrow{a} x$. Now, given a graph $G \in XVV$, and $\varphi : \Sigma^* \to 2^{(X \cup \overline{X})^*}$ a substitution, we define the graph: $\varphi^{-1}(G)$ in the following way:

$$\varphi^{-1}(G) = \{x \xrightarrow{d} y \mid d \in \Sigma \wedge \ x \overset{\varphi(d)}{\underset{G}{\Longrightarrow}} y\}$$

This graph is a subset of ΣVV

3.1 Prefix-recognizable graphs

In this section, let Λ be the complete binary tree over $X = \{a, b\}$, whose vertices are in V.

Given a language L in X^*, let us denote by $L_\Lambda = \{s \mid r \overset{L}{\underset{\Lambda}{\Rightarrow}} s\}$, the set of vertices in Λ that are reached by a path in L.

Definition 3.1. A graph in ΣVV is prefix-recognizable if it is the image of the complete binary tree, Λ, by an inverse regular substitution followed by a regular restriction:

$$\varphi^{-1}(\Lambda)_{|L_\Lambda}$$

With φ a regular substitution ($\varphi : \Sigma^* \to 2^{X^*}$), and L a regular language in X^*.

Example 3.2. The Figure 3.1 represents a classical example of prefix-recognizable graph, it is an infinite ladder (L_0) labelled by b's on the ascending side, by c's on the descending side, and with a's connecting the ascending and descending branches.

On this figure, the complete binary tree Λ is composed of A arcs (dotted) and B arcs (dashed). The graph L_0 is obtained using a restriction to vertices reached by a path in $L = A + B^* + B^*A$, and using the following rational substitution: $h(a) = \{A\}$, $h(b) = \{B\}$, $h(c) = \{\overline{A}\overline{B}A\}$
For example, in L_0, there is an arc labelled c, between BBA and BA (here we identify each vertex of Λ with the single path from the root leading to it), because there is a path labelled $\overline{A}\overline{B}A$ between them. We could also consider the same graph with the transitive closure for c arcs, in this case, the substitution for c would be: $h'(c) = \{\overline{A}(\overline{B})^+A\}$ (which is simply the iteration of $h(c)$, simplified by $A\overline{A} = \varepsilon$).

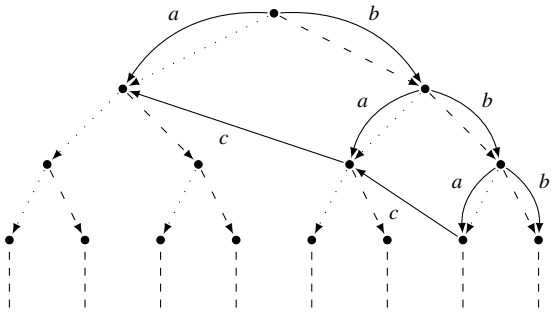


Figure 3.1: The infinite ladder L_0

Proposition 3.3. [6] *Inverse regular substitution and regular restriction preserves the decidability of the MSO theory of graphs.*

This proposition derives from the inductive definition of regular languages and MSO formula. From the decidability of the MSO theory of the complete binary tree we have the following.

Theorem 3.4. [6] *The monadic second order theory of prefix-recognizable graphs is decidable.*

3.2 Caucal hierarchy

Another operation preserving monadic second order theory is the unfolding. Using this operation Caucal has defined a hierarchy of graphs and terms having decidable MSO theories.

Theorem 3.5. [13] *Unfolding preserves the decidability of the MSO theory of graphs.*

The unfolding of a prefix-recognizable graph graph in general produces a graph which is not a prefix-recognizable graph. Let us denote these trees $\mathbf{tree}_2$. Applying inverse regular substitution followed by regular restriction to these trees produces graphs (that we denote $\mathbf{graph}_2$). Iterating this process defines $\mathbf{tree}_n$ and $\mathbf{graph}_n$ for each integer $n \geqslant 2$.
All theses graphs (and trees) families are defined by graph transformations from the complete binary tree.
By construction the MSO theory of each element in $\mathbf{tree}_n$ and $\mathbf{graph}_n$ is decidable.
And important result is the following:

Theorem 3.6. [7] *The hierarchy formed by the* $\mathbf{graph}_n$ *(resp.* $\mathbf{tree}_n$*) is strict:*

$$\mathbf{graph}_n \subsetneq \mathbf{graph}_{n+1}$$

$$\mathbf{tree}_n \subsetneq \mathbf{tree}_{n+1}$$

This theorem may be summarised in the following table:

Level	Trees	graphs
0	Finite trees	Finite graphs
1	regular trees	Prefix-recognizable graphs
2	algebraic trees	$\mathbf{graph}_2$
	...	
n	$\mathbf{tree}_n$	$\mathbf{graph}_n$

This external characterisation corresponds to an internal characterisation which is higher order pushdown automata: each n-graph is the ε-closure of the *configuration graph* of a n-pushdown automaton (see [5]).

3.3 Context-sensitive languages and rational graphs

In this section we present an external characterisation for context-sensitive languages.

3.3.1 Definitions

In this section we recall the classical definition of context-sensitive languages. Then we present the definition of the family of rational graphs. These graphs are very general, and provide a *graph* characterisation of these languages. More details can be found in [16, 18].
Context-sensitive languages are defined as the level 1 of the Chomsky hierarchy (0 being recursively enumerable sets). Which means they are characterised by growing word grammars. Another popular characterisation of these languages is by linear bounded Turing machines [14].
This family of languages is very expressive, for example, the sets of words of the form ww, or $a^nb^nc^n$, with n a natural number are context-sensitive sets of words. The set of a^p where p is a prime number is context-sensitive as well. One of the most stunning property of these languages is that they are closed under complementation.
The family of rational subsets of a monoid $(M,\cdot)$ is the least family containing the finite subsets of M and closed under union, concatenation and iteration.
A transducer is a finite automaton labelled by pairs of words over a finite alphabet X, see for example [2]. A transducer accepts a relation in $X^*\times X^*$; these relations are called rational relations as they are rational subsets of the product monoid $(X^*\times X^*,\cdot)$.
Now, let us consider the graphs of $X^*\times\Sigma\times X^*$. Rational graphs, denoted by $Rat(X^*\times\Sigma\times X^*)$, are extensions of rational relations, which are defined by *labelled rational transducers*.

Definition 3.7. A *labelled rational transducer* $T=(Q,I,F,E,L)$ over X and Σ, is composed of a finite set of states Q, a set of initial states $I\subseteq Q$, a set of final states $F\subseteq Q$, a finite set of transitions (or edges) $E\subseteq Q\times X^*\times X^*\times Q$ and a mapping L from F into 2^Σ.

An arc $u\xrightarrow{a}v$ is *accepted* by a labelled transducer T if there is a path from a state in I to a state f in F labelled by (u,v) and such that $a\in L(f)$.

Definition 3.8. A graph in $X^*\times\Sigma\times X^*$ is *rational* if it is accepted by a labelled rational transducer.

Let G be a rational graph, for each a in Σ we denote by G_a the restriction of G to arcs labelled by a (it defines a rational relation between vertices); let u be a vertex in X^*, we denote by $G_a(u)$ the set of all vertices v such that $u\xrightarrow{a}v$ is an arc of G.

Example 3.9. In Figure 3.2, the graph on the right-hand side is generated by the labelled transducer on the left-hand side.
The path $p\xrightarrow{0/0}q_1\xrightarrow{0/1}r_2\xrightarrow{1/1}r_2$ accepts the couple $(001,011)$, the final state r_2 is labelled by b thus there is a arc $001\xrightarrow{b}011$ in the graph.

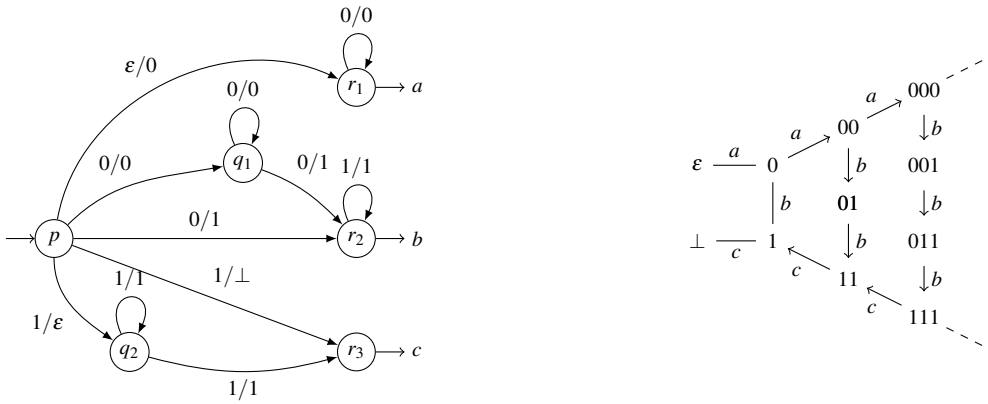


Figure 3.2: A rational graph and its labelled transducer

Rational graphs have been introduced in order to extend existing families of graphs. They provide a very general family of graphs. They have few decidable properties, but they characterise context-sensitive languages [18]. If we only consider trees (rooted connected acyclic-graphs such that each vertex has at most one predecessor) these trees have a decidable first order theory [4].

Using transducers to characterise a family of graphs induce that each graph is defined in a very precise way. In particular, each vertex is a word, and thus each arc is defined between two precise words, which are not interchangeable.

Finally, we recall the characterisation of context-sensitive languages by rational graphs:

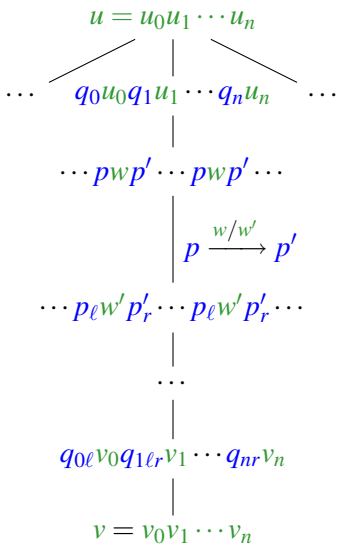
Theorem 3.10. [18] *The sets of path between regular sets of vertices of rational graphs corresponds precisely to context-sensitive languages.*

3.3.2 An external characterisation for rational graphs

There are at least two characterisations of rational graphs in terms of inverse substitution: the first one is presented in [16], it extends directly Proposition 3.3 to rational graphs. It uses linear context-free languages, and restricts the use of symbols of $\overline{X}$ to left-hand side of productions, and symbols of X to right-hand side, it is ad-hoc. Here, we present a second such characterisation build with unrestricted finite substitutions. Furthermore the generator is no longer the complete binary tree, but a complex rational graph built on purpose.

Example 3.11. Let $X = \{0,1\}$ be a fixed alphabet. Let G_{gen} be the rational graph labelled on X, defined as follows. First, this graph will be used to refine *any* graph in $Rat(X^* \times \Sigma \times X^*)$, many path in G_{gen} will correspond to path in *any* transducer. Each state of such transducer will be encoded in X^*. In fact, G_{gen} will encode 0 into 000, 1 into 001 for ordinary elements of X^*, and 0 into 010, 1 into 011 for elements of X^* which represents states. So some vertices will be elements of $\{000,001\}^*$, and some of $\{000,001,010,011\}^*$. Furthermore, 110, 101 and 100 will be used to mark states, and 111 will be used along a computation like a reading head (and thus some vertices of G_{gen} will be in $\{000,001,010,011,100,101,110,111\}^*$).

Now each pair of elements of $\{000,001\}^*$ are connected to each-other via a infinite set of paths of this form:



The first branching corresponds to *non-deterministically* guessing a path in the transducer that eventually will connect u and v in the transducer. Each intermediate step corresponds in applying a transition: each state are copied, except that a special marker is added: ℓ or r representing that the left-hand side (respectively right-hand side) have been checked (they are encoded by $110, 101$ and 100, representing respectively that ℓ is present, r or both). This path is reflected on the labels of the transitions. Furthermore the marker 111 is used inside each pwp' to reflect that this part is checked, and also where the progression is. The last sequence reaching state v is obtained by removing each state that has been checked.

The key aspect to observe is that along a path of the form $p \xrightarrow{w/w'} p'$ each occurrence of pwp' in the *vertex* $q_0u_0q_1u_1\cdots q_nu_n$ is processed, simultaneously.
Expressed differently: each path in G_{gen} from a vertex u to a vertex v reflects the individual transitions of the transducer T such that (u,v) belongs to the rational relation generated by T. And the first intermediate state $q_0u_0q_1u_1\cdots q_nu_n$ reflect an actual *path* in T, recognising (u,v).

Now, from this definition of G_{gen}, it is possible to express the following result:

Proposition 3.12. *Rational graphs are obtained from G_{gen} by* finite *inverse substitution and regular restriction.*

Proof sketch. Given a rational graph G in $Rat(X^* \times \Sigma \times X^*)$, and a in Σ, let T_a be the rational transducer corresponding to $G_{|a}$. We define the finite substitution

$$h(a) = \left\{ p_0 \xrightarrow{u_0/v_0} p_1 \dots p_{m-1} \xrightarrow{u_{m-1}/v_{m-1}} p_m \mid p_0 \in I(T_a) \wedge p_m \in F(T_a) \wedge \textbf{simple}(p_0 \Rightarrow p_m) \right\}$$

With **simple** denoting the fact that each transition appears at most once in the path. T_a is finite, so $h(a)$ is finite. Now, from the construction of G_{gen}, each path labelled by an element of $h(a)$ connects two vertices which are in the relation defined by T_a, and thus legitimately connected by an arc labelled a.
To ensure that only vertices not involving "states" are in G, we add the regular restriction to vertices in $\{000, 001\}^*$. □

Now, from Proposition 2.14 in [16] we know that the first order theory of rational graphs is undecidable. Obviously finite inverse substitution preserve the decidability of first-order logic. Thus following Corollary is straightforward.

Corollary 3.13. *First order theory of G_{gen} is undecidable.*

Even if the construction of G_{gen} is built on an explicit naming of the vertices. This is an external characterisation: every other rational graph is obtain from an algebraic transformation.
We have seen external characterisation for three families of graph defined by algebraic transformations from a generator. In the following section we will examine external characterisations obtained by recursive graph transformations.

4 Graph rewriting systems

In this section we examine graphs defined by graph rewriting systems. The characterisation are external as they do not provide explicit naming for the vertices. They are constructed as recursive application of finite graph transformations. We focus on HR-grammar, and their contextual counterparts.

4.1 Deterministic Graph grammars

Deterministic (hyperhedge replacement) graph grammars are another very nice example of external characterisation of infinite graphs. These grammars were initially defined to be an extension to graphs of word grammars. Indeed such a graph grammar derived, from an axiom, an infinite family of *finite* graphs. Courcelle in [12] used the deterministic form of these grammars to obtain a single infinite graph as the least solution of a finite set of deterministic graph equations. In 2007 Caucal made a very in-depth survey on deterministic graphs grammars [8]. In particular he devised several techniques which allowed the presentation of these results in a very unified manner.

Definition 4.1 (Hypergraph grammar)**.** A *hypergraph grammar* (HR-grammar for short) G, is a 4-tuple (N,T,R,H_0), where N and T are two ranked alphabets of respectively *non-terminals* and *terminals* symbols; H_0 is the axiom, a finite graph formed by hyperarcs labelled by $N \cup T$, and R is a set of rules of the form $f\ x_1\ \cdots\ x_{\rho(f)} \to H$ where $f\ x_1\ \cdots\ x_{\rho(f)}$ is an hyperarc joining disjoint vertices and H is a finite hypergraph.

Remark 4.2. In this paper, we consider graphs, therefore, the terminal symbols will have either rank one, or two. Furthermore, we see such a graph as a simple subset of $T_2VV \cup T_1V$. Rank 1 symbols will be called *colours* rather than labels (we use label to identify (hyper) arcs). A single vertex may have several colours.

A grammar is deterministic if there is a single rewriting rule per non-terminal:

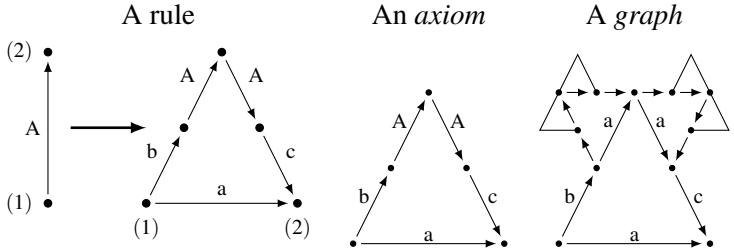
$$(X_1,H_1),(X_2,H_2) \in R \wedge X_1(1) = X_2(1) \Rightarrow (X_1,H_1) = (X_2,H_2)$$

Now, given a set of rules R, the *rewriting* $\underset{R}{\to}$ is the binary relation between hypergraphs defined as follows: M rewrites into N, written $M \underset{R}{\to} N$ if there is a non-terminal hyperarc $X = Av_1v_2\dots v_p$ in M and a rule $Ax_1x_2\dots x_p \to H$ in R such that N is obtained by replacing X by H in M: $N = (M-X) \cup h(H)$ for some injection h, mapping v_i to x_i for each i, and every other vertices of H to vertices outside of M. This rewriting is denoted by $M \underset{R,X}{\longrightarrow} N$. Now, this rewriting obviously extends to sets of non-terminal, for E such a set, this rewriting is denoted: $M \underset{R,E}{\longrightarrow} N$. The *complete parallel rewriting* $\underset{R}{\Rightarrow}$ is the rewriting relative to the set of *all* non-terminal hyperarcs of R.

Now given a deterministic graph grammar $G = (N, T, R, H_0)$, and a hypergraph H, we denote by $[H] := H \cap (T\ V_H\ V_H \cup T\ V_H)$ the set of terminal arcs, and colours of H. A graph H is *generated* by G, if it belongs to the following set of isomorphic graphs:

$$G^{\omega} = \left\{ \cup_{n \geqslant 0} [H_n] \mid \forall n \geqslant 0, H_n \underset{R}{\Rightarrow} H_{n+1} \right\}$$

Example 4.3. We present here a simple example of deterministic graph grammar and propose a representation of the resulting graph. An important observation on this graph is that it does not provide any naming scheme for the vertices. But there is of course an obvious connection between the vertices and the sequence of graph rewriting producing them.



Graph grammars characterise *regular graphs*. This external characterisation is very efficient to extend to these infinite graphs techniques which work for finite graphs (for example computing the connected components of a regular graph is very simple from the grammar). Furthermore these graphs correspond (in a precise sense) to transition graphs of pushdown automata. Nonetheless, algorithms which only depend on the structure of these graphs often make technical assumptions on the form of the automaton: for example that the states carry some information, such as the configuration belongs to a certain regular set. These assumptions only affect the internals of the automaton, it does not affect the structure of its configuration graph. In such case, grammars are very efficient as there is no assumption on vertices identification, only the structure is explicit.
Following structural operations on graphs preserve the regularity of graphs, meaning that given a graph grammar G there is an effective procedure to produce a grammar G' producing the desired graph.

Proposition 4.4. *Accessible colouring preserves regularity.*

This proposition relies on the fact that there are only finitely many right-hand side in any grammar so computing local accessibility and iterating eventually finishes.

Proposition 4.5. *The restriction of a regular graph to vertices having some colour is a regular graph.*

This result is obvious from the definition. And implies that restriction to a regular set of configuration for a pushdown automaton is a pushdown automaton, which seems less obvious.

4.2 Synchronised graph grammars

These grammars generate deterministic regular graphs. They correspond to deterministic context-free languages, and enable the extension of visibly pushdown languages to every deterministic context-free language. This topic is discussed in [9]. It presents a nice way to *synchronise* deterministic regular graphs.
From this synchronisation, closure properties are defined (mainly under product) and enables a nice extension of visibly pushdown automata.

4.3 Contextual graph rewriting systems

In this section we present a second external characterisation of rational graphs. This graph rewriting characterisation is the most general in some sense: each natural more general rewriting system fails to produce recursive graphs.

4.3.1 The general setting

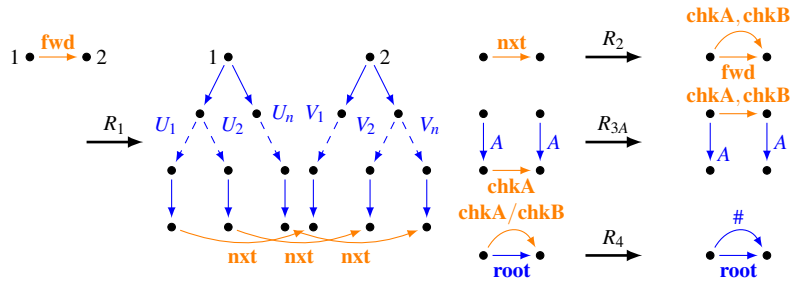
We recall mainly results from [17]
Let N_R be a finite ranked set of *non-terminals*, and T_R a finite ranked set of *terminals*.
We propose here a natural definition of contextual graph rewriting system.

Definition 4.6 (Contextual graph rewriting system)**.** A *contextual graph rewriting system* S, is a set of rules of the form $H_c \cup f\ x_1\ \cdots\ x_{\rho(f)} \to H_c \cup H$ where $f\ x_1\ \cdots\ x_{\rho(f)}$ is a non-terminal hyperarc, H_c is a finite *context* graph, and H is a finite hypergraph, that can share some vertices with H_c and f. Furthermore, H_c is composed only of terminal hyperarcs, and $H_c \cup f\ x_1\ \cdots\ x_{\rho(f)}$ forms a connected hypergraph.

Proposition 4.7. *Given $(U_i,V_i)_{i\in[n]}$ an instance of PCP, there exists a graph obtained from a finite axiom A by a contextual graph rewriting system which possesses an arc labelled # between the two vertices v_0 and v_1 of A if and only if $(U_i,V_i)_{i\in[n]}$ is a positive instance.*

The following example illustrates this proposition.

Example 4.8. But the construction is straightforward, and illustrated by this example. Consider $((U_i,V_i))_{i\in[n]}$ an instance of PCP, and observe the following contextual rewriting system:



The axiom is simply the following finite graph: $\{\mathbf{root}\ v_0\ v_1, \mathbf{fwd}\ v_0\ v_1\}$. Furthermore there is a rule R_{3B} similar to R_{3A} for the rewriting of **chkB**.
Now, the rule R_1 uses arc **fwd** to produce two partial binary trees corresponding to the U_i's and V_i's. For each sequence of indexes $(k_j)_{j\in[m]}$, the extremity of the path $(U_{k_j})_{j\in[m]}$ is connected to the extremity of $(V_{k_j})_{j\in[m]}$ by an non-terminal arc **nxt**. Then the rules R_{3A} and R_{3B} will ultimately reach the arc **root** if and only if $(U_i,V_i)_{i\in[n]})$ is a positive instance of PCP.

The most direct consequence of this proposition is the following:

Corollary 4.9. *Graphs generated by deterministic contextual graph rewriting systems are not recursive.*

4.3.2 Contextual hyper-edge-replacement graph grammars

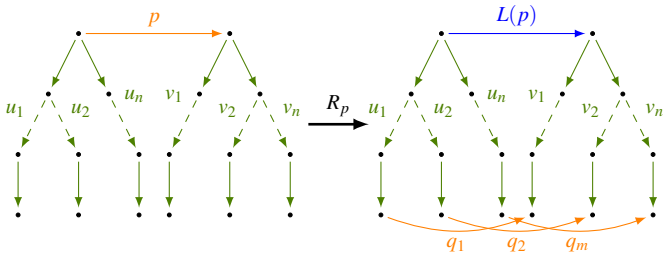
In this section we present a more restrictive contextual rewriting system which will be used to characterise context-sensitive languages.

Definition 4.10. A *contextual hyper-edge-replacement hypergraph grammar* (CHR-grammar for short) is a tuple (C,N,T,R_c,H_0), where C,N and T are finite ranked alphabets of respectively contextual, non-terminal and terminal symbols; R_c is a finite set of contextual rules (for each rule $H_c \cup fx_1 \dots x_{\rho(f)} \to H_c \cup H$, the graph H_c is formed only by arcs labelled in C, and H by arcs labelled in $T \cup N$); and H_0 is the axiom: a *deterministic* regular graph formed by arcs with labels in C, and a single non-terminal hyperarc.

This definition imposes that the axiom is a deterministic regular graph. This restriction ensures that for each rule R, of non-terminal A, and each occurrence of A in the graph, there is at most a single morphism which maps the context of the left-hand side of R to the neighbourhood of A.
First we will show that using a n-ary tree as axiom is sufficient to obtain all the rational graphs up to isomorphism, achieving the goal of containing the context-sensitive languages.

Proposition 4.11. *Any rational graph on $X^* \times \Sigma \times X^*$ is obtained from a* CHR*-grammar.*

Example 4.12. Like for Proposition 4.7, the proof is in the full paper. But the construction is straightforward, and illustrated by this example. Let G be a rational graph in $X^* \times \Sigma \times X^*$ (and T a transducer representing it), let H_0 be the complete n-ary tree labelled on X (with a non-terminal p_0 on the root). For each state p of T, we have the following rule R_p.



Here, we suppose that there are transitions $p \xrightarrow{u_i/v_i} q_i$ for some states $(q_i)_{i\in[m]}$, and also $L(p)$ represent all labels produced at state p (if p is a terminal state). Now each pair of path in H_0 correspond to a pair of paths in T. Thus the graph obtained from the contextual rewriting system is the same as the graph obtained from the transducer.

Corollary 4.13. *Any context-sensitive language L is the set of paths between two colours in a graph obtained from a* CHR*-grammar.*

4.3.3 Graphs obtained from a tree-separated contextual grammar are rational graphs

In this section we examine restrictions in order to obtain a converse to Proposition 4.11.
First, we designate interesting restrictions of CHR-grammar. A CHR-grammar (C,N,T,R_c,H_0) is called a *tree-*CHR*-grammar* if the axiom H_0 is a tree, and left-hand side of each rule of R_c is formed by trees *rooted* in the vertices of the non-terminal (some vertices of this non-terminal may be non-root vertices of theses trees). Furthermore, if each such tree possesses a single vertex belonging to the non-terminal (its root) this grammar is called a *tree-separated-*CHR*-grammar*. These grammars are captured by rational graphs:

Proposition 4.14. *Any graph obtained from a tree-separated-*CHR*-grammar, is isomorphic to a rational graph on $X^* \times \Sigma \times X^*$.*

Now combining this result with Theorem 3.10 and Corollary 4.13 we obtain the desired result.

Theorem 4.15. *The set of paths (between colours) of any graph obtained from a tree-separated-*CHR*-grammar, is a context-sensitive language. And conversely, any context-sensitive language can be obtained as the set of paths of such a graph.*

Now we show that the natural extension of the previous result by allowing the non-terminal (of the left-hand side) to be set anywhere in the context produces another non-recursive family of graphs.

Proposition 4.16. *There is a graph obtained from a* CHR*-grammar, such that the axiom is a deterministic tree, and having a loop on the root of the axiom if and only if a given instance of PCP has a solution.*

Unfortunately, at the moment, there are few applications illustrating the potential of this characterisation. A nice one, would be to provide a new demonstration of the closure under complementation of context-sensitive languages. Unfortunately the most obvious proof of this result would require determinism for these graphs, and we have some indications that deterministic rational graphs do not characterise all context-sensitive languages.

5 Discussion

In this paper we have presented several external characterisations of infinite graphs families. These characterisations falls into two categories: either algebraic transformations from a generator, or recursive application of finite graph transformations.

Our statement is that these characterisations are essential in order to grasp structural properties of graphs. And also provide an elegant way to extend to infinite graphs techniques used for finite graphs. In particular HR-grammar enable many simplifications in proofs relatively to those using pushdown automata.

References

[1] R. Alur & P. Madhusudan (2004): *Visibly pushdown languages*. In: *STOC 04*. ACM, pp. 202–211.

[2] J. Berstel (1979): *Transductions and context-free languages*. Teubner.

[3] A. Blumensath & E. Grädel (2000): *Automatic Structures*. In: *15th IEEE Symposium on Logic in Computer Science LICS 2000*. pp. 51–62. Available at `http://www-mgi.informatik.rwth-aachen.de/Publications/pub/graedel/BlGr-lics00.ps`.

[4] A. Carayol & C. Morvan (2006): *On rational trees*. In: Zoltán Ésik, editor: CSL 06, *LNCS* 4207. pp. 225–239.

[5] A. Carayol & S. Woehrle (2003): *The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata*. In: P. K. Pandya & J. Radhakrishnan, editors: FSTTCS 03, *LNCS* 2914. pp. 112–123.

[6] D. Caucal (1996): *On transition graphs having a decidable monadic theory*. In: Icalp 96, *LNCS* 1099. pp. 194–205.

[7] D. Caucal (2002): *On infinite terms having a decidable monadic theory*. In: MFCS 02, *LNCS* 2420. pp. 165–176.

[8] D. Caucal (2007): *Deterministic graph grammars*. In: *Texts in logics and games 2*. Amsterdam University Press, pp. 169–250.

[9] D. Caucal & S. Hassen (2008): *Synchronization of Grammars*. In: *CSR 08*, *LNCS* 5010. pp. 110–121.

[10] D. Caucal & T. Knapik (2001): *An internal presentation of regular graphs by prefix-recognizable ones*. *Theory of Computing Systems* 34(4).

[11] T. Colcombet (2002): *On Families of Graphs Having a Decidable First Order Theory with Reachability*. In: *Proceedings of the 29th International Conference on Automata, Languages, and Programming*, *LNCS* 2380. Springer-Verlag, pp. 98–109.

[12] B. Courcelle (1990): *Handbook of Theoretical Computer Science*, chapter Graph rewriting: an algebraic and logic approach. Elsevier.

[13] C. Courcelle & I. Walukiewicz (1998): *Monadic Second-Order Logic, Graph Coverings and Unfoldings of Transition Systems*. Ann. Pure Appl. Logic 92(1), pp. 35–62.

[14] S.-Y. Kuroda (1964): *Classes of Languages and Linear-Bounded Automata*. Information and Control 7(2), pp. 207–223.

[15] Martin Leucker (2001): *Prefix-Recognizable Graphs and Monadic Logic*. In: Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors: Automata, Logics, and Infinite Games, Lecture Notes in Computer Science 2500. Springer, pp. 263–284. Available at `http://link.springer.de/link/service/series/0558/bibs/2500/25000263.htm`.

[16] C. Morvan (2000): *On rational graphs*. In: J. Tiuryn, editor: Fossacs 00, *LNCS* 1784. pp. 252–266.

[17] C. Morvan (2009): *Contextual graph grammars characterizing context-sensitive languages*. Rapport de recherche 1926, IRISA.

[18] C. Morvan & C. Stirling (2001): *Rational graphs trace context-sensitive languages*. In: A. Pultr & J. Sgall, editors: *MFCS*, *LNCS* 2136. pp. 548–559.

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

[20] J. L. Peterson (1981): *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR.

[21] I. Walukiewicz (2000): *Model Checking* CTL *Properties of Pushdown Systems*. In: *FSTTCS*. pp. 127–138. Available at `http://link.springer.de/link/service/series/0558/bibs/1974/19740127.htm`.