# A Survey Of Automata And Logics Over Infinite Graphs

---

*Report submitted in Partial Fulfilment of the*

**Master of Science (M.Sc.) in Computer Science**

**by**


## Shambwaditya Saha

ssaha@cmi.ac.in

**CHENNAI MATHEMATICAL INSTITUTE**
**Plot H1, SIPCOT IT PARK**
**Padur Post, Siruseri**
**Tamilnadu - 603103, INDIA**


**JULY 2011**

# Certificate

This is to certify that the dissertation entitled *A Survey of Automata and Logic Over Infinite Graphs* is a bonafide record of work done by Shambwaditya Saha under my supervision.

This dissertation is to be submitted to Chennai Mathematical Institute for partial fulfillment of the requirements of M.Sc. (Computer Science) degree.

Dated:

K. Narayan Kumar
Professor
Chennai Mathematical Institute

# Contents

# 1 Introduction

Quite often in programs we use ubounded data stratures like stack and queues. These infinte state-based systems are modelled using infite transitions systems/graphs. Though these transition graphs are infinite in structure, for algorithmic applications they must be finitely representable. Among the many ways to generate finitely representable infinte straures, here we look at tree rewriting, where the basic object of rewriting are trees.

In Ground Tree Rewriting Systems(GTRS) we rewrite a subtree of a tree by another. A more generalized form of GTRS is Regular-GTRS (RGTRS) where rules are of the form $T \hookrightarrow T'$ where $T$ and $T'$ are regular sets. Here a subtree of a tree that appears in $T$ is replaced by any tree of $T'$. For RGTRS the FO theory and several of the reachabilty problems are known to be decidable while some are undecidable. In chapter 1 we look at the decidability proofs of FO theory, one-step reachabilty and rechabilty of RGTRS graphs.

Infinite graphs families are also linked to language theory. We know the chomsky hierarchy consists of regular, context free, context sensitive and enumerable languages. These families of lanugaes are characterised by infinte automata, which can be defined as an infinte graph along with two regular set initial and final vertices.

It is also known that regular languages are characterised by finite graphs, CFLs by pushdown graphs and enumerable languages by transition graphs of Turing Machines. In chapter 2, we study the traces of automatic graphs which is found exactly to be the family of context sensitive languages. We look at two different approaches to prove this fact. While one approach requires the use of Penttonen's Characterization of CSLs, the other uses Tiling systems to characterize CSLs.

In the next two section we start with the common definations which will be used in both the subsequent chapters.

## 1.1 Graphs and Automata

**Definition 1.1** (Graph). *[CM06][MR05] A labeled, directed and simple graph is a set $G \subseteq V \times \Sigma \times V$ where $\Sigma$ is a finite set of labels and $V$ a countable set of vertices. We denote by $s \xrightarrow[G]{a} t$ the existence of the edge $(s, a, t)$ in the graph or simply by $s \xrightarrow{a} t$ when there is no ambiguity. We also denote the set of edges by $(T_a)_{a\in\Sigma}$ where $T_a$ is the set of $a$-labeled edges. A graph is deterministic if it contains no pair of edges having same source and label.*

**Definition 1.2** (Path). *[CM06][MR05] A Path $s \xRightarrow[G]{\pi} t$ of a graph $G$ from a vertex $s$ to a vertex $t$ and of label $\pi$, is a finite sequence $s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n$ of edges of $G$ such that $\pi = a_1 \ldots a_n, \quad s = s_0$ and $t = s_n$ .*

**Definition 1.3** (Trace). *[CM06][MR05] A trace of a graph G is the language L(G,I,F), of path labels leading from the set I of initial vertices to the set F of final vertices:*

$$L(G, I, F) = \{\ u \mid \exists s \in I,\ \exists t \in F,\ s \xRightarrow{u} t\}$$

**Definition 1.4** (Automata). *[MR05] We can also define an Automata A by a triple $(G, I, F)$ where $G$ is a graph and $I$ and $F$ are initial and final set of states. The language, $L(A)$, recognized by $A$ is the trace $L(G, I, F)$.*

## 1.2 Transducers and Relations

**Definition 1.5** (Transducer). *[MR05] A transducer $T$ is a finite automaton where labels have been modified to recognize relations instead of sets of words. It is defined by a finite subset of $Q \times \Gamma^* \times \Gamma^* \times Q$ of labelled edges, where $Q$ is a finite set of states, by a set $I \subseteq Q$ of initial states, and by a set $F \subseteq Q$ of final states. So a transducer is labelled by pairs of words. Any transition $(p, u, v, q)$ of a transducer $T$ is denoted by $p \xrightarrow[T]{u/v} q$ or by $p \xrightarrow{u/v} q$ when $T$ is understood.*

**Definition 1.6** (Transducer path). *[MR05] A path $p_0 \xrightarrow{u_1/v_1} p_1 \ldots p_{n-1} \xrightarrow{u_n/v_n} p_n$ with $u = u_1 \ldots u_n$ and $v = v_1 \ldots v_n$ is labeled $u/v$ and is denoted by $p_0 \xRightarrow{u/v} p_n$ . A path is successful if it leads from an initial state to a $T$ final one. A pair $(u, v) \in \Gamma^* \times \Gamma^*$ is recognized by a transducer if there exists a successful path labelled $u/v$.*

**Definition 1.7** (Rational Relation)**.** *[MR05] A relation is rational if it is recognized by a transducer. We denote by* $Rat(\Gamma^* \times \Gamma^*)$ *the set of all binary rational relations.*

In general, there is no bound on the size difference between input and output in a transducer. Interesting classes of transducers/rational-relation are obtained by enforcing some form of synchronization.

**Definition 1.8** (Synchronous Transducer)**.** *[CM06] A tranducer is called synchronous if it have labels in* $\Gamma \times \Gamma$*. They recognize length-preserving rational relations which only pair words of the same size, and are called letter-to-letter relations.*

**Definition 1.9** ((Left)Synchronized Transducer)**.** *[CM06] (Also called Automatic Transducers) A transducer is called left synchronized if accept unequal sized words by padding* $\epsilon$ *at the right end of the smaller word to make their lengths equal. Formally a left synchronized relation is a finite union of relations of the form* $S.F$ *where* $S$ *is a synchronous relation (accepted by a synchronous transducer) and* $F$ *is either equal to* $\{\epsilon\} \times R$ *or* $R \times \{\epsilon\}$ *where* $R$ *is a rational language. Right synchronized transducers are defined similarly. Unless otherwise stated we will always refer to left-Synchronized transducers as synchronized transducers.*

**Definition 1.10** (Normalized Transducer)**.** *[MR05] A transducer is normalized if all its transitions are of the form* $p \xrightarrow{u/v} q$ *where* $|u| + |v| = 1$*. It is straight forward to see that any rational transducer can be simulated by a normalized transducer.*

**Proposition 1.1.** *[FS93] The family of synchronized relation contains the recognizable relations and the rational relations of bounded length difference.*

**Definition 1.11** (Rational Graph)**.** *[MR05] Using words as vertices, infinite graphs can be defined by the relations between the extremities of its edges. Given any graph* $G \subseteq \Gamma^* \times \Sigma \times \Gamma^*$ *, we denote by* $\xrightarrow[G]{a}$ *the relation* $\{(s,t) \mid (s,a,t) \in G\}$*. The graph* $G$ *is rational if for each* $a \in \Sigma$*, the relation* $\xrightarrow[G]{a}$ *rational.*

The rational graph with synchronized transducers were defined by [BG00] under the name automatic graphs and by [Ris02] under the name synchronized rational graphs.

We can get a Chomsky like hierarchy of graphs presented in [CK02]. It follows from the definition that the sequential synchronous, synchronous, synchronized and rational graphs form an inceasing hierarchy. This hierarchy is strict(up to isomorphism).

**Lemma 1.1** ([MR05]). *For every automatic graph $G$ with vertices in $\Gamma^*$ and rational sets $I$ and $F$, one can find two automatic graphs $H$ and $J$ and two symbols $i$ and $f \notin \Gamma$ , such that $L(G, I, F) = L(H, i^*, f^*)$ and $L(G, I, F) = L(J, \{i\}, \{f\})$*

*Proof Sketch.* Let $G$ be a automatic graph with vertices in $\Gamma^*$ and $i, f$ be two new distinct symbols. We define the automatic graph H as follows :

$$\begin{aligned} H = G \ \cup \ & \{ \ (i^n, a, v) \ | \exists a \in \Sigma, \exists u \in I, (u, a, v) \in G, |u| = n\} \\ \cup \ & \{(u, a, f^n) \ | \exists a \in \Sigma, \exists v \in F, (u, a, v) \in G, |v| = n\} \end{aligned}$$

Similarly we can also define the automatic graph $J$ by adding edges from the the vertex $i$ instead of $i^n$ and similarly adding edges to the vertex $f$ insted of $f^n$ □

# 2 Model checking of RGTS Graphs

In this chapter we consider the transition graphs of regular ground tree rewriting systems($RGTRS$). For $RGTRS$ , the first-order theory and several rechability problems are known to be decidable, while some are undecidable. Apart from the definition, we look at the decidable proofs of first-order theory, one-step reachability and reachability of $RGTR$ graphs and the undecidability proof of universal rechability.

## 2.1 Definitions

### 2.1.1 Ranked tree

**Definition 2.1** (Prefix Ordeing)**.** *[Löd06] We denote by $\sqsubseteq$ the prefix ordering on $\mathbb{N}^*$ and by $\sqsubset$ the strict prefix ordering. That is, $x \sqsubseteq y$ for $x, y \in \mathbb{N}^*$ if there is $z \in \mathbb{N}^*$ such that $y = xz$, and $x \sqsubset y$ if $x \sqsubseteq y \wedge x \neq y$.*

**Definition 2.2** (Ranked Alphabet)**.** *[Löd06] A ranked alphabet $A$ is a finite family of finite sets $(A_i)_{i\in[k]}$, where $[k] = \{0, \ldots, k\}$. Here, $k$ denotes the maximum rank in $A$. Alternatively we identify $A$ with the set $\bigcup_{i=0}^{k} A_i$. To specify a ranked alphabet one can list all the sets $A_i$ or simply specify the set of all symbols together with their ranks.*

**Definition 2.3** (Ranked Tree)**.** *[Löd06] Ranked tree $t$ over $A$ is a mapping $t : D_t \to A$ with $D_t \subseteq [k-1]^*$ such that*

- *$D_t$ is finite and prefix closed,*
- *$D_t \neq \emptyset$,*
- *for each $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$: if $xi \in D_t$, then $xj \in D_t$ for all $j \leq i$,*
- *if $x0, \ldots, x(i-1) \in D_t$ and $xi \notin D_t$, then $t(x) \in A_i$.*

The set $D_t$ is called the domain of $t$ and the elements of $D_t$ are called the locations of $t$. The function $t$, maps each valid position of the tree ($D_t$) to a letter ($A$). For $x, y \in D_t$ we call $x$ the predecessor of $y$ and $y$ a successor of $x$ if there is $i \in N$ such that $y = xi$. The set of all ranked trees over $A$ is denoted by $T_A$ .

For example, let us define a ranked alphabet A as $A_0 = \{a, c, d\}$, $A_1 = \{b, c\}$ and $A_2 = \{a\}$, and the set $D_t$ as $\{\epsilon, 0, 1, 00, 01, 10, 010\}$. Now if we consider the function $t$ as $t(\epsilon) = t(0) = t(10) = a$, $t(01) = b$, $t(1) = t(00) = c$ and $t(010) = d$, we get a ranked tree of the form $a(a(c, b(d)), c(a))$ over the ranked alphabet A.

**Definition 2.4** (Sub-Tree). *[Löd06] For $x \in \mathbb{N}^*$ we define $xD_t = \{xy \in \mathbb{N}^* \mid y \in D_t\}$ and $x^{-1}D_t = \{y \in \mathbb{N}^* \mid xy \in D_t\}$. For $x \in D_t$ the subtree $t^{\downarrow x}$ of $t$ at $x$ is the tree with domain $D_{t^{\downarrow x}} = x^{-1}D_t$ and $t^{\downarrow x}(y) = t(xy)$.*

**Definition 2.5** (Substitution). *[Löd06] Using substitution we replace the subtree $t^{\downarrow x}$ in $t$ by the tree $s$. It is denoted as a pair consisting of the location $x \in \mathbb{N}^*$ and the tree $s \in T_A$, written as $[x/s]$. A substitution $[x/s]$ can be applied to a tree $t$ if $x \in D_t$. The result $t[x/s]$ is the tree $t'$ with domain $D_{t'} = (D_t \backslash xD_{t^{\downarrow x}}) \cup xD_s$ and*

$$t'(y) = \begin{cases} t(y) & if\ y \in D_t \backslash xD_{t^{\downarrow x}} \\ s(z) & if\ y = xz\ with\ z \in D_s \end{cases}$$

In other words, by substitution $t[x/s]$ we mean to replace the subtree $t^{\downarrow x}$ in $t$ at position $x$ by the tree $s$.

### 2.1.2 Ground Tree Rewriting

**Definition 2.6** (GTRS). *[Löd06] A ground tree rewriting system ($GTRS$) is a tuple $\mathcal{R} = (A, \Sigma, R, t_{in})$, where $A = (A_i)_{i \in [k]}$ is a ranked alphabet, $\Sigma$ is an alphabet, $R$ is a finite set of rules of the form $s \overset{\sigma}{\hookrightarrow} s'$ with $s, s' \in T_A$, $\sigma \in \Sigma$, and $t_{in} \in T_A$ is the initial tree.*

Like rewriting a prefix of a word in prefix rewriting systems [Tho09] , here we rewrite an entire subtree of a tree. This results in a more flexible model than the Caucal hierarchy [Tho09].

The set of rules defines what kind of substitutions are compatible with $\mathcal{R}$, as follows:

- A substitution $[x/s']$ is $(\mathcal{R},\sigma)$-applicable to a tree $t \in T_A$ if $x \in D_t$ and if there is a rule $s \overset{\sigma}{\hookrightarrow} s' \in R$ with $s = t^{\downarrow x}$ .
- $[x/s']$ is $\mathcal{R}$-applicable to $t$ if it is $(\mathcal{R},\sigma)$-applicable to $t$ for some $\sigma \in \Sigma$.

and the notation for rewrites are as follows:

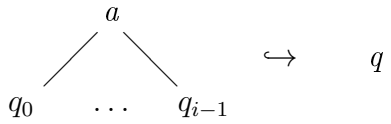
- $t \to^{\sigma}_{\mathcal{R}} t'$ if there is an $(\mathcal{R},\sigma)$-applicable substitution $[x/s']$ such that $t[x/s'] = t'$,
- $t \to_{\mathcal{R}} t'$ if there is $\sigma \in \Sigma$ with $t \to^{\sigma}_{\mathcal{R}} t'$, and
- $\to^{*}_{\mathcal{R}}$ for the transitive and reflexive closure of $\to_{\mathcal{R}}$.

### 2.1.3 Tree automata

Tree automata are devices with finite memory that read input trees and accept or reject them.

**Definition 2.7** (NTA). *[Löd06] A nondeterministic tree automaton (NTA) is a tuple $\mathcal{A} = (Q, A, \Delta, F)$, where $Q$ is a finite set of states, $A = (A_i)_{i\in[k]}$ is a ranked alphabet, $F \subseteq Q$ is a set of final states and $\Delta \subseteq (\bigcup_{i=0}^{k} Q^i \times A_i) \times Q$ is the transition relation.*

The set of trees that can be accepted by an NTA is called regular. Tree automata can also be viewed as a GTRS over the alphabet $A$. Let $q_0, \ldots, q_{i-1}, q \in Q$ and $a \in A$. The transition $(q_0, \ldots, q_{i-1}, a, q)$ corresponds to the (unlabeled) rewriting rule




Hence, a tree is accepted by NTA $\mathcal{A}$ if it can be transformed into a final state using the transition as rewriting-rules. We define the set $T(\mathcal{A})$ of trees accepted by NTA $\mathcal{A}$

$$T(\mathcal{A}) = \{\ t \in T_A \mid \exists q \in F :\ t \to^{*}_{\mathcal{A}} q\ \}$$

We give two arguments that are used repeatedly in connection with this notation.

**Remark 2.1.** *[Löd06] Let $\mathcal{A} = (Q, A, \Delta, F)$ be an NTA, $p, q \in Q$, $t, s \in T_A$, and $x \in D_t$.*

*i)* $t \to^{*}_{\mathcal{A}} t[x/q]$ *iff* $t^{\downarrow x} \to^{*}_{\mathcal{A}} q$.

*ii) If* $t[x/q] \to^{*}_{\mathcal{A}} p$ *and* $s \to^{*}_{\mathcal{A}} q$ *then* $t[x/s] \to^{*}_{\mathcal{A}} p$

**Definition 2.8** ($\epsilon - NTA$). *[Löd06] An $\epsilon - NTA$ is a tuple $\mathcal{A} = (Q, A, \Delta, F)$, where $Q$, $A$, $F$ are as for NTA and $\Delta \subseteq ((\bigcup_{i=0}^{k} Q^i \times A_i) \cup (Q \times Q)$ is the transition function.*

The only difference to NTA is that, we can have $\epsilon$-transitions of the form $(p, q)$ for $p, q \in Q$ which correspond to the rewrite rule $p \hookrightarrow q$. So in an $\epsilon$-NTA we allow the automata to change state without removing one of the input symbols, while in NTA the automata must replace one occurance of a symbol in each rewrite step.

**Remark 2.2.** *The class of regular tree languages are closed under Boolean opearations.*

### 2.1.4 Regular Ground Tree Rewriting

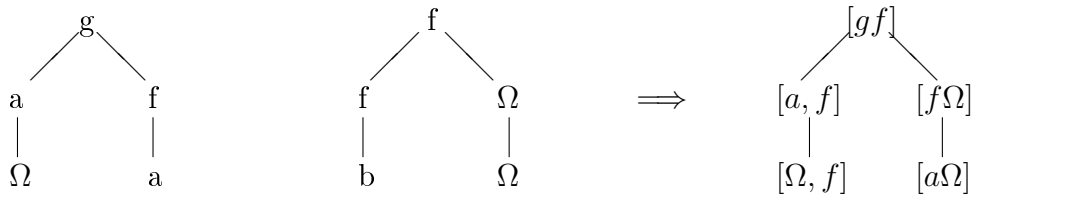
Regular Ground Tree Rewriting Systems ($RGTRS$) are defined in the same way as $GTRS$ but with regular sets instead of single trees in the rewriting rules.

**Definition 2.9** (RGTRS). *[Löd06] A regular ground tree rewriting system ($RGTRS$) is a tuple $\mathcal{R} = (A, \Sigma, R, t_{in})$, where $A$, $\Sigma$, and $t_{in}$ are the same as for $GTRS$ , and $R$ is a finite set of rewriting rules of the form $T \overset{\sigma}{\hookrightarrow} T'$ for regular sets $T, T \subseteq T_A$ of trees.*

A substitution $[x/s]$ is $(\mathcal{R}, \sigma)$-applicable to a tree $t$ if $x \in D_t$ and if there is a rule $T \overset{\sigma}{\hookrightarrow} T' \in R$ with $t^{\downarrow x} \in T$ and $s \in T'$. With this definition we can define $\mathcal{R}$-applicable, $\rightarrow_{\mathcal{R}}^{\sigma}$, $\rightarrow_{\mathcal{R}}$, and $\rightarrow_{\mathcal{R}}^{*}$ similarly as GTRS. The tree language generated by $\mathcal{R}$ is $T(\mathcal{R}) = \{t \in T_A \mid t_{in} \rightarrow_{\mathcal{R}}^{*} t \}$.

**Definition 2.10** (Regular Ground Tree Rewriting Graph). *[Löd06] A directed edge labeled graph $G_{\mathcal{R}} \subseteq V_{\mathcal{R}} \times E_{\mathcal{R}} \times V_{\mathcal{R}}$ generated by $\mathcal{R}$ is defined by $V_{\mathcal{R}} = T(\mathcal{R})$ and $(t, \sigma, t') \in E_{\mathcal{R}}$ if $t \rightarrow_{\mathcal{R}}^{\sigma} t$ .*

Graphs that are isomorphic to $G_{\mathcal{R}}$ for some RGTRS $\mathcal{R}$ are called regular ground tree rewriting graphs (RGTR graphs). Note that an RGTR graph only consists of trees that are reachable from the initial tree.




$g(a, f(a))$ $f(f(b))$ $[g, f](([a, f]([\Omega, b], [f, \Omega]([a, \Omega]))$

## 2.2 FO-Theory of RGTR gaphs

We now describe a proof of *FO is decidable for RGTR graphs* due to Dauchet and Tison [DT90]. We start by stating the the concept of RR relations, introduced in [DT90] using tree automata and tree transducers and then sketch the prove of *FO is decidable for RR relations*. Lastly we discuss that GTR/RGTR graphs are nothing but RR relations.

### 2.2.1 RR Relation

**Definition 2.11.** *[DT90] Let us denote $T_\Sigma$ as a free algebra generated by a finite alphabet $\Sigma$, $T_\Sigma(X)$ as the free algebra over $X$, $(T_\Sigma(X))_1$ as the set of terms of $T_\Sigma(X)$ whose arity is 1 and $Rec(\Sigma)$ as the set of recognizable tree languages on $\Sigma$.*

The main intuition is to encode multiple trees as a single tree (we also call this superposed tree as *term*) like transducers as shown in figure 2.2.1.

**Definition 2.12** (Alphabet $\Delta^n$)**.** *[DT90] The alphabet of the terms is $\Delta^n = (\Sigma \cup \Omega)^n = \{[\alpha_1 \dots \alpha_n] \mid \alpha_i \in \Sigma \cup \Omega\}$ where $\Omega$ is a new symbol of arity 0. The rank of $[\alpha_1 \dots \alpha_n]$ is maximal rank of the $\alpha_i$.*

**Definition 2.13** (Tree language $F_n(\Sigma)$)**.** *[DT90] The tree language $F_n(\Sigma)$ is the set of terms with valid configurations. A term $t$ belons to $F_n(\Sigma)$ iff*

- *for all node $\alpha$ in $t$ ,if $\beta_1 \dots \beta_p$ are successors of $\alpha$ then $rank(\pi_i(\alpha)) \leq k \Leftrightarrow \pi_i(\beta_{k+1}) = \dots \pi_i(\beta_p) = \Omega$*
- $\forall i, \pi_i(root(t)) \neq \Omega$

*For any $n$, $F_n(\Sigma)$ is a recognizable tree language.*

**Definition 2.14** (Projection). *[DT90] The projection operation on a particular node $\alpha = [\alpha_1 \dots \alpha_n]$ of a term is defined as $\pi_i(\alpha) = \alpha_i$. The projection $\pi_i(t) : F_n(\Sigma) \to T_\Sigma$ on a term returns the ith tree of the term. So the projection $\pi : F_n(\Sigma) \to T_\Sigma^n$ on a particular term $t$ returns the n-tupple of trees in $T_\Sigma$ ie. $\pi(t) = (\pi_1(t) \dots \pi_n(t)))$. Conversely we can define a function $can : T_\Sigma^n \to F_n(\Sigma)$ where given n-tuple of trees it returns a term, ie. $can(t_1 \dots t_n) = t$ .*

We also make sure that the encoding of the terms in $F_n(\Sigma)$ is unique ie, $\forall t, t' \in F_n(\Sigma), \quad \pi(t) = \pi(t') \Leftrightarrow t = t'$.

**Definition 2.15** (RR relation). *[DT90] Let $BF_n(\Sigma)$ be the set of all recognizable tree languages included in $F_n(\Sigma)$. For all $F \in BF_n(\Sigma)$ , we associate a n-ary relation $R$ defined as: $R(t_i, \dots, t_n) \Leftrightarrow \exists t \in F, \quad \forall i \in 1 \dots n, \ \pi_i(t) = t_i$. So the n-ary relation $R$ corresponds to the tree language $F$. The set of all these relations $R$ is $RR_n(\Sigma)$ and $RR = \bigcup_n RR_n(\Sigma)$. A relation is RR recognizable if it belongs to RR.*

### 2.2.2 Poperties of RR relations

**Proposition 2.1.** *[DT90] $RR_n$ is closed under boolean operation and the closure is effective.*

*Proof Sketch.* For every RR relation we get a corresponding tree lnguage. As tree languages are closed under boolean operations so are RR relations. □

**Proposition 2.2.** *[DT90] RR relatons are closed under the following operations (Let us consider $R \in RR_n$ and $R' \in RR_{n'}$):*

- *Permutation: If $R$ is a $RR_n$ and $\sigma$ be a permuation of $1 \dots n$ , then $\sigma(R) = \{(t_{\sigma(1)} \dots t_{\sigma(n)}) \ | (t_1 \dots t_n) \ \in R\}$ is a $RR_n$.*
- *Projection: If $R$ is a $RR_n$ then for all $i \in 1 \dots n$, $pr_i(R) = \{(t_1 \dots t_{i-1}, t_{i+1}, \dots t_n) \ | \exists t_i, \ (t_1 \dots t_n) \ \in R\}$ is a $RR_{n-1}$*
- *Cylindrification: If $R$ is a $RR_n$ the for all $i \in 1 \dots n$, $C_i(R) = \{(t_1 \dots t_{i-1}, t, t_{i+1}, \dots t_n) \mid (t_1 \dots t_{i-1}, t_{i+1}, \dots t_n) \in R\}$ is a $RR_{n+1}$*
- *Product: If $R$ is a $RR_n$ and $R'$ is a $RR_{n'}$ then $R \times R'$ is a $RR_{n+n'}$ .*
- *Composition: If $R$ and $R'$ are $RR_n$ relations then $R \circ R'$ is also a $RR_n$ relations.*

*and moreover, the properties Membership, Emptyness and Equality are decidable for RR relations.*

*Proof Sketch.* In all the above claims we consider the corresponding tree language(s). As all the above properties are closed/decidable for tree language(s), hence it is also closed/decidable for RR relations. □

### 2.2.3 Theory of RR relations

The FO theory of RR relations where constants are ground trems (trees/elements of $T_\Sigma$) and predicates are RR relations, is decidable. The set of terms is $T_\Sigma \cup X$, while atomic formulas and formulas are defined as usual. The domain is iterpreted as $T_\Sigma$, and for every predicate R we associate the corresponding function $(T_\Sigma^n \to \{T, F\})$.

**Proposition 2.3.** *[DT90] RR is decidable*

*Proof Sketch.* To decide the validity of a RR-formula, we transform the formula into a semantically equivalent formula of the form :

$$(Q_1 x_1) \dots (Q_p x_p)\ R(x_1 \dots x_p) \text{ where } Q_i = \exists \text{ or } \forall$$

by eliminating constants, binding free variable with existential quantifier, transforming the formula into prenex normal form and then evaluating the boolean operators. Now we eliminate the quantifiers $Q_i$ by projection if $Q_i = \exists$ or by decylindrification if $Q_i = \forall$. Lastly we get a formula of the form $Q_x R(x)$, if $Q_x = \exists(\forall)$ we check if $R(\neg R)$ is non-empty. □

### 2.2.4 FO decidable for GTR and RGTR graphs

**Definition 2.16.** *[DT90] A rewrite rule is ground iff no variable occurs in it. A tree rewrite system is GTRS if all rules are ground.*

So in the theory of GTRS , constants are ground terms(or elements on $T_\Sigma$). A binary predicates R is associated with every ground system $\mathcal{R}$ which indicates if a tree is one-step rechable from another considering the rewrite rules of $\mathcal{R}$. The domain of the interpretation is $T_\Sigma$, and the predicate R can be defined as:

$R(t,u)$ iff $\exists(l \to r) \in \mathcal{R},\ c \in T_\Sigma(X)_1$ st $t = c.l,\ v = c.r$ where $\Sigma' = \bigcup_{\alpha\in\Sigma}(\alpha,\alpha)$
Or in other words, the tree $t$ is one step rechable from the tree $u$ if there exists a digonal $RR_2$ relation $c$ and a rule $l \to r$ in $\mathcal{R}$ such that $t = c.l$ and $v = c.r$ .

Similarly we can introduce the concept of regularity by making the rules of $\mathcal{R}$ of the form $(T \to T')$ where $T$ and $T'$ are regular tree languages. We modify the definition of R as follows:

$$R(t,u) \text{ iff } \exists(T \to T') \in \mathcal{R},\ c \in T_\Sigma(X)_1,\ l \in T \text{ and } r \in T' \text{ st } t = c.l,\ v = c.r \text{ where } T, T' \in RR_1$$

Like the previous definition this one says that, the tree $t$ is one step reachable from the tree $u$ if there exist a digonal $RR_2$ relation $c$, a rule $T \to T'$ in $\mathcal{R}$ and two trees $l$ and $r$ in the regular tree languages $T$ and $T'$ respectively, such that $t = c.l$ and $v = c.r$ .

**Theorem 2.1.** *[DT90] FO-Theory of GTRS and RGTRS is decidable.*

*Proof Sketch.* It can be clearly seen that relation $R(t,u)$ for both GTRS and RGTRS can be encoded using the operations *can*, $\circ$(composition), $\times$(product) and $\cup$(union). As RR relations are closed under these operations and $T, T'$ are $RR_1$ relations, the relation R is also a RR relation in both the above two cases. Hence GTRS and RGTRS have decidable FO Theory. □

## 2.3 Reachability problems of RGTR graphs

In this section we are summarising the results of the basic reachability problems that can be expressed in temporal logic(CTL*), due to Loding [Löd02]. Specifically we look at the problems EX(One-step reachability) and EF(Reachability), both of which are shown to be decidale. We also state the Universal Reachability problem (AF) which is shown to be undecidable in [Tho09].

### 2.3.1 One-Step reachability : EX

**Theorem 2.2.** *[Löd02] [Löd02] Given a RGTRS $\mathcal{R} = (A, \Sigma, R, t_{in})$ and an $\epsilon$-NTA $\mathcal{A} = (Q, A, \Delta, F)$, one can construct an $\epsilon$-NTA $\mathcal{A}_{pre_\mathcal{R}}$ accepting the set $pre_\mathcal{R}(T(\mathcal{A})) = \{t \in T_A \mid t \underset{\mathcal{R}}{\to} T(\mathcal{A})\}$*

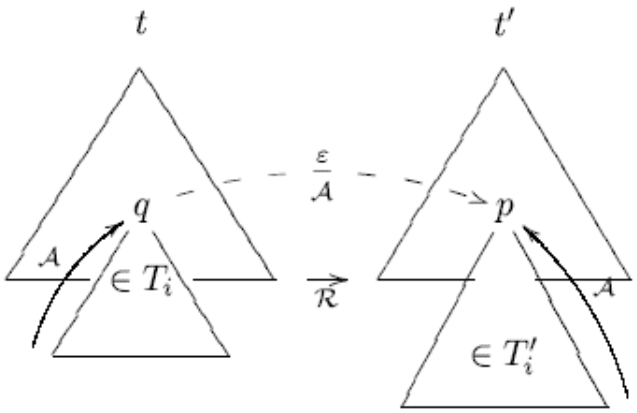



Figure 2.1: $\epsilon$-transition added to $\mathcal{A}$

*Proof Sketch.* In this proof the basic task is to modify the NTA $\mathcal{A}$ so that it can simulate exactly one 1-step rewrite rule by using a $\epsilon$-transition.

Let us assume, the following situation that there is a rewrite rule $T_i \hookrightarrow T_i' \in R$ , $t \in T_A$ and $t' \in T(\mathcal{A})$ such that the following properties hold:

- $t^{\downarrow x} \underset{\mathcal{A}}{\longrightarrow} q$ where q is a state of $\mathcal{A}$ that is only reachable via trees from $T_i$, in particular $t^{\downarrow x} \in T_i$
- $(t')^{\downarrow x} \in T_i'$ and $(t')^{\downarrow x} \underset{\mathcal{A}}{\rightarrow} p$, for some state $p$ of $\mathcal{A}$

From the above situation we can say $t \underset{\mathcal{R}}{\longrightarrow} t'$ holds. Now for $\mathcal{A}_{pre_{\mathcal{R}}}$ to accept $t$ , we must add $\epsilon$-edges from $q$ (the final state of $\mathcal{A}_i$, which accepts $T_i$) to $p$ (a state of $\mathcal{A}$ which can be reached via trees from $T_i'$). Thus the automata reduces $t^{\downarrow x}$ to $q$ using $\mathcal{A}_i$, takes the $\epsilon$-transition to jump to $p$ and pretends to read $(t')^{\downarrow x}$ instead of $t^{\downarrow x}$. This is depicted in the figure 2.3.1. So along with guessing the rewrite rule $T_i \hookrightarrow T_i'$, $\mathcal{A}_{pre_{\mathcal{R}}}$ must also guess a subtree $t^{\downarrow x} \in T_i$ such that $t^{\downarrow x}$ can be replaced by $t' \in T_i'$ yeilding a tree from $T(\mathcal{A})$.

It is evident from the description that for every rule $T_i \hookrightarrow T_i' \in R$, $\mathcal{A}_{pre_{\mathcal{R}}}$ contains the automata $\mathcal{A}_i$ . It also contains two copies of the automata $\mathcal{A}$ to keep track of the number of rewrites made (as we want exactly one rewrite). □

### 2.3.2 Reachability : EF

**Theorem 2.3.** *[Löd02] [Löd02] Given a RGTRS $\mathcal{R} = (A, \Sigma, R, t_{in})$ and an $\epsilon$-NTA $\mathcal{A} = (Q, A, \Delta, F)$, one can construct an $\epsilon$-NTA $\mathcal{A}_{pre^*_\mathcal{R}}$ accepting the set $pre^*_\mathcal{R}(T(\mathcal{A})) = \{t \in T_A \mid t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A})\}$*

*Proof Sketch.* The intution is similar to the construction of the previous proof, except that we do not have any constrain on the number of rewrites made. So $\mathcal{A}_{pre^*_\mathcal{R}}$ contains a single copy of the automata $\mathcal{A}$ along with the copy of $\mathcal{A}_i$ (which accepts $T_i$), for every rule $T_i \hookrightarrow T'_i \in R$.

As there can be multiple rewrites we add $\epsilon$-edges form the final states of $\mathcal{A}_i$, to any state of $\mathcal{A}_{pre^*_\mathcal{R}}$ that can be reached via a tree from $T'_i$. Since the state space is fixed, we only add finite number of $\epsilon$-transitions. Therefore, this method always terminates. □

### 2.3.3 Universal reachability : AF

**Theorem 2.4.** *[Tho09] [Löd02] Given a RGTRS$\mathcal{R} = (A, \Sigma, R, t_{in})$ and a regular set $T \subseteq T_A$ , the following problem is undecidable: Does every path through $G_\mathcal{R}$ that starts in $t_{in}$ infinetely often visit $T$?*

We ommit the proof sketch of this theorem. A detailed construction can be found at [Tho09] and [Löd02].

# Automatic Graphs and CSL

3

## 3.1 Context Sensitive Languages

In Chomsky hierarchy of languages [HU79], the family of context-sensitive languages(CSLs) is located between recursively enumerable and context-free language. There are many finite formalisms to characterize CSLs. In the following we look at three of these characterizations.

- Linear Bounded Automata(LBA): A linearly bounded automata is a Turing Machine such that the size of the tape is bounded linearly, by the length of the input.
  **Theorem 3.1.** *[Kur64] Context-Sensitive Languages are the set of languages recognized by linearly bounded Turing machines.*
- Penttonen's Characterization: A different characterization of CSLs due to Penttonen [Pen74], is based on rewriting systems.
- Bounded Tiling System :A less well-known acceptor of CSLs. However one can show these systems are equivalent.

As Penttonen's Characterization and Tiling systems are the heart of the proof techniques we sketch, we now give a detailed description of them in the following two subsections.

### 3.1.1 Penttonen's Characterization

This Characterization is based on a rewriting system using particular rules.

**Definition 3.1** (2-system)**.** *[MR05] A rewriting system $\Gamma = \Gamma_1 \cup \Gamma_2$ is a 2-system if every rule of $\Gamma_2$ is of the form $AB \to AC$ with $B \neq C$ and every rule of $\Gamma_1$ is of the form $A \to a$ where $A, B, C \in \Gamma$ (non-terminal alphabet) $\Gamma$ and $a \in \Sigma$ (terminal alphabet).*

**Definition 3.2** (Linear Language)**.** *[MR05] A language is linear if it can be generated by a grammar whose rules are of the form $Z \to W$ , where $Z$ is a non-terminal, $W$ is a word over terminal and non-terminal symbols, with at most one non-terminal.*

Context-sensitive languages are obtained by derivation of a 2-system from a linear language.

**Theorem 3.2.** *[Pen74] There exists a linear language $L_{Lin}$ such that every context-sensitive language is $\{v \in \Sigma^* \mid \exists u \in L_{Lin},\ u \xrightarrow[\Gamma]{*} v\}$ for some 2-system $\Gamma$.*

### 3.1.2 Tiling Systems

Tiling systems were used to specify picture languages i.e. two dimensional words on finite alphabet [GR96]. A $(n, m)$-picture over alphabet $\Gamma$ is a two-dimensional array of letters in $\Gamma$ with $n$ rows and $m$ columns. By $p(i, j)$ we denote the letter occuring in the $ith$ row and $jth$ column starting from the top-left corner, by $\Gamma^{n,m}$ the set of all $(n, m)$-pictures and by $\Gamma^{*,*}$ the set of all pictures (except the empty picture). A $(2, 2)$-picture is also called a Tile.

Given a $(n, m)$-picture $p$ over $\Gamma$ and a letter $\# \notin \Gamma$, $p\#$ is a (n+2,m+2) picture language over $\Gamma \cup \{\#\}$ such that

- $p_\#(i, 1) = p_\#(i, m + 2) = \# \ for\ i \in [1, n + 2]$,
- $p_\#(1, j) = p_\#(n + 2, j) = \# \ for\ j \in [1, m + 2]$,
- $p_\#(i + 1, j + 1) = p(i, j)\ for\ i \in [1, n]\ and\ j \in [1, m]$

**Definition 3.3.** *[CM06] Given any(n,m)-picture $p$ such that $n, m \geq 2$, T(p) is the set of all tiles appearing in p.*

**Definition 3.4** (Local Picture Lnaguage)**.** *[CM06] A picture language $K \subseteq \Gamma^{*,*}$ is local if there exists a symbol $\# \notin \Gamma$ and a finite set of tiles $\Delta$ such that $K = \{p \in \Gamma^{*,*} \mid T(p\#) \subseteq \Delta\}$.*

**Definition 3.5** (Frontier)**.** *[CM06] Frontier of a (n,m)-picture $p$ is the word $fr(p) = p(1, 1) \dots p(1, m)$ corresponding to the first row of the picture.*

**Definition 3.6** (Tiling System)**.** *[CM06] A tiling system is a tuple $S = (\Gamma, \Sigma, \#, \Delta)$, where $\Gamma$ is a finite alhabet , $\Sigma \subseteq \Gamma$ is a input alphabet, $\# \notin \Gamma$ is a frame symbol and $\Delta$ is a finite set of tiles over $\Gamma \cup \{\#\}$. It recognizes:*

- *Local Picture Language :* $P(S) = \{p \in \Gamma^{*,*} \mid T(p_{\#}) \subseteq \Delta\}$
- *Word Language :* $L(S) = fr(P(S)) \cap \Sigma^*$

A tiling system S recognizes a language $L \subseteq \Sigma^+$ in height $f(n)$ for some mapping $f : N \to N$ if for all $w \in L(S)$ there exists a $(n, m) - picture\ p$ in $P(S)$ such that $w = fr(p)$ and $n \leq f(m)$.

**Theorem 3.3.** *This theorem links tiling systems with linear bounded automata*

- *A linearly bounded Turing machine T working in f(n) reversals can be simulated by a tiling system of height f(n) + 2.*
- *A tiling system of height f(n) can be simulated by a linearly bounded Turing machine working in f(n) reversals.*

**Definition 3.7.** *[CM06] A tiling system is deterministic if we can infer from each row (in the picture) a single possible next row.*

**Definition 3.8.** *[CM06] A Context-Sensitive language is deterministic if it is accepted by a deterministic tiling system.*

## 3.2 Equivalence between automatic graphs and CSls

We now state the main theorem of this chapter.

**Theorem 3.4.** *[CM06] [MR05] Context-sensitive languages are exactly the traces of automatic graphs between rational sets.*

The Theorem has been proved using two different(but related) techniques. We sketch the details below:

- Proposition 3.1 and Proposition 3.2 establishes a tight bound between automatic graphs and tiling systems.
- Alternatively Proposition 3.3 and Proposition 3.4 establishes a tight bound between automatic graphs and CSLs using LBA and Penttonen's characterization.

### 3.2.1 Equivalence using tiling system

In the following two propositions, we use tiling systems to characterize context-sensitive languages.

**Proposition 3.1.** *[CM06] Given a Tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, there exists a automatic graph G and two rational sets I and F such that L(S)=L(G,I,F)*

*Proof Sketch.* For a given Tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, we can construct a Automatic Graph G such that $L(S) = L(G, \#^*, M)$ where M represents the set of all possible last column of the picture of P(S). M is a regular set as we will see soon.

For all picture $p \in P(S)$ , the vertices of the graph G comprises of $\#^*$ and all possible columns of p and edges corresponds to all possible frontiers of pictures p.

A pair of words (s,t) is accepted by the transducer $T_e$ if and only if e is the first letter of t, and either s and t are two adjacent columns of a picture in P(S) or $s \in \#^*$ and t is the first column of a picture in P(S). This is illustrated in the Figure.

$$\begin{array}{|cccccc|}\hline \# & \# & \# & \# & \# & \# \\ \# & a & a & b & b & \# \\ \# & a & \bot & \bot & b & \# \\ \# & \bot & \bot & \bot & \bot & \# \\ \# & \# & \# & \# & \# & \# \\ \hline \end{array} \qquad \begin{bmatrix} \# \\ \# \\ \# \end{bmatrix} \xrightarrow{a} \begin{bmatrix} a \\ a \\ \bot \end{bmatrix} \xrightarrow{a} \begin{bmatrix} a \\ \bot \\ \bot \end{bmatrix} \xrightarrow{b} \begin{bmatrix} b \\ \bot \\ \bot \end{bmatrix} \xrightarrow{b} \begin{bmatrix} b \\ b \\ \bot \end{bmatrix}$$

Figure 3.1: picture $p\#$, $p \in P(S)$ and the automatic graph corresponding to it.

As $\#^*$ and M are regular sets the trace of the graph (within the regular sets $\#^*$ and M) is precisely the language accepted by the tiling system.

*Construction of Automata for Language M:* As M accepts all possible last column of pictures of P(S), we build the automata $A = (Q, \Gamma, \delta, \#, F)$ as follows:

- $Q = \Gamma \cup \{\#\}$, $\Gamma$ = Input alphabet, $\#$ = Initial state.
- F : if $\begin{array}{|cc|}\hline a & \# \\ \# & \# \\ \hline \end{array} \in \Delta$ *then* $a \in F$
- $\delta$ : *if* $\begin{array}{|cc|}\hline \# & \# \\ a & \# \\ \hline \end{array} \in \Delta$ *then* $\# \xrightarrow[A]{a} a$ or if $\begin{array}{|cc|}\hline a & \# \\ b & \# \\ \hline \end{array} \in \Delta$ *then* $a \xrightarrow[A]{b} b$

Let L(A)=M then M = set of possible last columns of pictures of P(S).

(Note that this does not imply each word of M is actually a last column of a picture in P(S), it only says that M contains all words, which can be generated by aligning the right bordered tiles of $\Delta$ in all possible ways.)

*Construction of Automatic Graph G:* To build the Automatic graph G, we build the automatic transducers $T_e = (Q,\ (\Gamma \cup \{\#\} \times \Gamma \cup \{\#\}),\ \delta,\ (\#,\#),\ F)$ as follows:

- $(\Gamma \cup \{\#\}) \times (\Gamma \cup \{\#\})$ = input alphabet , $(\#,\#)$ = initial state
- $\delta$ : *if* $\begin{array}{|c|c|}\hline \# & \# \\ \hline a & b \\ \hline \end{array} \in \Delta$ *then* $(\#,\#) \xrightarrow[T_b]{a/b} (a,b)$ *for* $b \neq \#$

  *or if* $\begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline \end{array} \in \Delta$ *then* $(a,b) \xrightarrow[T_e]{c/d} (c,d)$ *for* $b,d \neq \#$, *for arbitrary* $e \in \Sigma$
- F : $(a,b) \in (\Gamma \cup \{\#\}) \times \Gamma)$ such that $\begin{array}{|c|c|}\hline a & b \\ \hline \# & \# \\ \hline \end{array} \in \Delta$

□

**Proposition 3.2.** *[CM06] For every automatic graph G and two rational sets I and F, there exist a tiling system S such that L(G,I,F)=L(S).*

*Proof Sketch.* Let $G = (T_a)_{a\in\Sigma}$ be an automatic graph with vertices in $\Gamma^*$ ($\Sigma \subseteq \Gamma$). Without loss of generality we assume $I = i^*$ and $F = f^*$ such that $i, f \notin \Gamma$ (by Lemma 1.2) and all transducers have disjoint state space.

A word $u(= a_1a_2 \ldots a_n) \in L(G, i^*, f^*)$ if there exist a path labeled $u$ from the vertex $i^n$ to the vertex $f^n$ for some $n$ . We can also say the composition of the transducers $(T_{a_1} \circ T_{a_2} \circ \ldots \circ T_{a_n})$ accepts the relation $(i^n, f^n)$. In this proof, we want to encode the run of these transducers corresponding to the labeled path $u$, as a picture whose frontier is precisely the word $u$.

To capture the run of all the transducers , we encode a one-step run of a transducer as a tile, which contains information such as the previous state, current state, input letter and output letter of that transducer.

Alternatively we can say that for a particular transition of a transducer we get a set of tiles (Note, all these tiles may not be actually used in the picture, it only says they are all comatible with this particular transition).

$$\begin{bmatrix} i \\ i \\ i \end{bmatrix} \xrightarrow{T_a} \begin{bmatrix} x \\ x' \\ x'' \end{bmatrix} \xrightarrow{T_b} \begin{bmatrix} y \\ y' \\ y'' \end{bmatrix} \xrightarrow{T_c} \begin{bmatrix} f \\ f \\ f \end{bmatrix}$$

| | | | | |
|---|---|---|---|---|
| # | # | # | # | # |
| # | $a$ | $b$ | $c$ | # |
| # | $xp$ | $yq$ | $fr$ | # |
| # | $x'p'$ | $y'q'$ | $fr'$ | # |
| # | $x''p''$ | $y''q''$ | $fr''$ | # |
| # | # | # | # | # |

Figure 3.2: From the Automatic graph (on the left), we can construct the picture $p\#$ (on the right), such that $p \in P(S)$ where $S = (\Gamma, \Sigma, \#, \Delta)$, $x, x', x'' \in Q_a$, $y, y', y'' \in Q_b$ and $z, z', z'' \in Q_c$

Formally, $Q_a$ is the set of states of $T_a$, $q_0^a \in Q_a$ is the unique initial state of $T_a$ and $Q_F$ is the set of final states of all $T_a$. Also let $a, b, c, d \in \Sigma$, $x, x', y, y', z, z' \in \Gamma$ and $p, p', q, q', r, r', s, s' \in \bigcup_{a\in\Sigma} Q_a$. We define a tiling system $S = (\Gamma, \Sigma, \#, \Delta)$, where $\Delta$ is constructed by using the following rules:

Tiles of the form $\begin{array}{|c|c|}\hline xp & yq \\ \hline x'p' & y'q' \\ \hline \end{array}$ corresponding to the one-step run $q \xrightarrow[T_b]{x'/y'} q'$ of the transducer $T_b$. Moreover the tiles which make up the first row of the picure (ie. tiles with hashes(#) in the first row) ,must encodes the path label $u$ and hene the second row contains letters from $\Sigma$. This indicates the initial guessing as we non-deterministically choose the word (or the path label) and are represented by tiles of the form $\begin{array}{|c|c|}\hline \# & \# \\ \hline \# & a \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline \# & \# \\ \hline a & b \\ \hline \end{array}$ and $\begin{array}{|c|c|}\hline \# & \# \\ \hline a & \# \\ \hline \end{array}$ .

Now depending on the value of $p, q$ and $x'$ we can have the following four cases:

- $q = q_0^b$ : If $q$ is the initial state of transducer $T_b$, we replace the top right cell of the tile by the letter $b$ (representing $T_b$). It is clear from the previous paragraph that the top left cell will contain any letter $a \in \Sigma$. This results into tiles of the form $\begin{array}{|c|c|}\hline a & b \\ \hline x'p' & y'q' \\ \hline \end{array}$
- $x' = i$ : The tiles that corresponds to the first column of the picture $p\#$, must contain hashes(#) as their left column, resulting into tiles of the form $\begin{array}{|c|c|}\hline \# & yq \\ \hline \# & y'q' \\ \hline \end{array}$.
- $x' = f$ : It is also clear that the tiles which encodes the last column must contain hashes(#) as its right column. Hence we have tiles of the form $\begin{array}{|c|c|}\hline b & \# \\ \hline fp' & \# \\ \hline \end{array}$

where $p' \in Q_b$ (indicating that the last transducer is $T_b$) and $\begin{array}{|c|c|}\hline fp & \# \\ \hline fp' & \# \\ \hline \end{array}$ where $p, p' \in Q_b$ for some $b \in \Sigma$.

- $p \in Q_F$ or $q \in Q_F$ or Both : To encode the final states of the transducers, tiles of the form $\begin{array}{|c|c|}\hline xp & \# \\ \hline \# & \# \\ \hline \end{array}$ , $\begin{array}{|c|c|}\hline \# & yq \\ \hline \# & \# \\ \hline \end{array}$ and $\begin{array}{|c|c|}\hline xp & yq \\ \hline \# & \# \\ \hline \end{array}$ (where $p, q \in Q_F$) are used respectively.

Since frontiers of Tiling Systems are exactly CSLs we conclude that Automatic Graphs recognize exactly the class of CLSs. □

### 3.2.2 Equivalence using LBA and Penttonen characterization

In the next theorem we use Linear bounded automata (LBA) to characterize context sensitive languages. Here we sketch the proof of a more general(stronger) form of the proposition *"Traces of Automatic graphs are context sensitve languages"*

**Proposition 3.3.** *[MR05] For a Rational graph G and two rational sets I and F, there exist a LBA A such that L(G,I,F) = L(A).*

*Proof Sketch.* The naive approach would be to simulate the graph using LBA , by storing a vertex in its tape and computing the next one. But this methods fail as the size of a vertex is totally unrelated to the size of the input word. The lengths of the vertices may grow exponentially, while the tape size is still linear in the order of its input.

Without loss of generality we assume all the transducers of the graph are normalized. We already know that, in a single move, a normalized transducer either consumes an input symbol or produces an output symbol, but not both. We also assume (by lemma 1.2 ) that $I = \{i\}$ and $F = \{f\}$.

So to overcome this difficulty we simulate all the transducers of G in parallel. For each transducer we only keep the current state and some information like what input it may consume and and what has it produced.

Let $u(= a_1 a_2 \dots a_n) \in L(G, \{i\}, \{f\})$, and hence the composition of the transducers $(T_{a_1} \circ T_{a_2} \circ \dots \circ T_{a_n})$ accepts the relation $(i, f)$.

Between every two consecutive transducer $T_{a_j}$ and $T_{a_{j+1}}$ we keep a buffer which indicates some letter (say $X$) that $T_{a_j}$ might generated using some transition $q \xrightarrow{\epsilon/X} r$. Thus we can activate $T_{a_{j+1}}$ knowing that X is the first non-empty label that $T_{a_{j+1}}$ might consume using some transition $q' \xrightarrow{X/\epsilon} r'$.

The trick is that there is just one buffer space in-between $T_{a_j}$ and $T_{a_{j+1}}$. So we create the LBA in such a way that $T_{a_j}$ doesent produce the next symbol unless all transducers $T_{a_k}$, $k > j$ , has read/used-up their input symbols and each of them($T_{a_k}$) is waiting for the previous ($T_{a_{k-1}}$) one to produce the next output symbol.

Hence we only need to store $n$ states, $n+1$ buffer positions including the initial vertex $i$ (which is the input of transducer $T_{a_1}$) and the final vertex $f$ (which is the output of transducer $T_{a_n}$) and some control markers, which makes the tape size linear in order of the input. Hence the proposition. □

As Automatic graphs are a strict subset of Rational Graphs [Tho09], we can claim that the proposition *"For an Automatic graph G and two rational sets I and F, there exist a LBA A such that L(G,I,F) = L(A)."* also holds true.

In order to show that traces of Automatic graphs are exactly the class of CSLs, we now state the following proposition:

**Proposition 3.4.** *[MR05] Let L be a context sensitive language , then there exist a automatic graph G and two rational sets I and F , such that L=L(G,I,F).*

The proof of this proposition requires Penttonen's characterization of CSLs. We ommit the proof sketch, a detailed construction can be found at [MR05].

# Bibliography


[BG00] Achim Blumensath and Erich Grädel. Automatic structures. In *LICS*, pages 51–62, 2000.

[CK02] Didier Caucal and Teodor Knapik. A chomsky-like hierarchy of infinite graphs. In *MFCS*, pages 177–187, 2002.

[CM06] Arnaud Carayol and Antoine Meyer. Context-sensitive languages, rational graphs and determinism. *Logical Methods in Computer Science*, 2(2), 2006.

[DT90] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. *Logics In Computer Science*, pages 242–248, 1990.

[FS93] Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993.

[GR96] Dora Giammarresi and Antonio Restivo. Two-dimensional languages. In *Handbook of Formal Languages*, volume 3. Springer Verlag, 1996.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Kur64] S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Computation/information and Control*, 7:207–223, 1964.

[Löd02] Christof Löding. *Infinite Graphs Generated by Tree Rewriting.* PhD thesis, RWTH Aachen, Germany, 2002.

[Löd06] Christof Löding. Reachability problems on regular ground tree rewriting graphs. *Theory Comput. Syst.*, 39(2):347–383, 2006.

[MR05] Christophe Morvan and Chloe Rispal. Families of automata characterizing context-sensitive languages. *Acta Inf.*, 41(4-5):293–314, 2005.

[Pen74] Martti Penttonen. One-sided and two-sided context in formal grammars. *Information and Control*, 25(4):371–392, 1974.

[Ris02] Chloe Rispal. The synchronized graphs trace the context-sensitive languages. *Electr. Notes Theor. Comput. Sci.*, 68(6):55–70, 2002.

[Tho09] Wolfgang Thomas. Finite automata and the analysis of infinite transition systems. 2009.