Fundamenta Informaticae XX (2000) 1–25
IOS Press

Thue Specifications, Infinite Graphs and Synchronized Product

Étienne Payet
IREMIA, Université de La Réunion,
BP 7151, 97715 Saint Denis Messageries Cedex 9, France
e-mail: epayet@univ-reunion.fr

Abstract. This paper presents some formal verification-oriented results about the class of graphs associated with Thue specifications. It is shown that this class is closed, up to isomorphism, under synchronized product. It is also established that every rational graph with no edge labeled by the empty word is isomorphic to the graph of a Thue specification. A consequence of this result is that the first-order theory of the graphs of Thue specifications is undecidable. Connections between graphs of Thue specifications and those of Turing machines are finally investigated. The main result is that the graph of each Thue specification is observationally equivalent to that of a Turing machine but the reverse does not hold.

Keywords: string-rewriting, infinite graphs, synchronized product, Turing machines

1. Introduction

An important step in formal verification consists in defining a mathematical model of a computer system to be verified. Labeled transition systems [AN82, Niv79] are currently used as such models, *e.g.* in methods like model checking [CE81]. They are made of a set of states and a set of labeled transitions. Each label represents an event that causes the transition of the computer system from one state to another. The synchronized product of labeled transition systems is an operation of paramount importance since it allows to model systems of interacting processes. It consists in considering some labeled transition systems modeling each component of the set of processes and a synchronization constraint, *i.e.* a set of tuples of labels, modeling the interactions. A tuple in the constraint is composed of events that are allowed to happen at the same time. The result of the synchronized product is a labeled transition system that consists of the composition of the initial ones and whose labels belong to the constraint.

At practical level, in many real cases, processes that are handled can only be represented by very large transition systems. Consequently, verification techniques that are based on such structures are known to encounter space-complexity problems. To face up these problems, many compression-like techniques, as *e.g.* binary decision diagrams [Bry86], have been developed and used in methods as symbolic model checking [McM92].

The problem of storage space for a representation of a process may be overcome using (possibly) infinite transition systems. Indeed, such systems having a strong regularity can be represented in a finite and compact way. Consequently, a process whose behaviour is modeled by a finite and very large, but strongly regular, transition system could be handled efficiently thanks to a compact presentation of an infinite transition system that would be an approximation of its behaviour. Such an approach could be used to avoid an enumeration of every state of the process. It could also be used to deal efficiently with processes whose number of states is really infinite (as *e.g.* any process with an unbounded queue).

Labeled transition systems can be seen as simple labeled graphs, *i.e.* as sets of labeled edges between vertices. Computer scientists started to study infinite graphs only recently. They discovered that some transition systems enjoy interesting properties such as having a decidable monadic second-order theory. This is the case for the transition graphs of pushdown machines [MS85], the regular graphs [Cou89] and the prefix-recognizable graphs [Cau96]. The proof of each of these results relies on Rabin's theorem, stating the decidability of the monadic second-order theory of the infinite binary tree [Rab69]. Unfortunately, none of the classes of these three kinds of graphs is closed under synchronized product. Nevertheless, the class of rational graphs [Mor00], that contains each of the three above classes, enjoy the closure property under this operation.

In this paper, we consider a class of infinite transition systems which has been introduced by Knapik and Calbrix [KC99]. The elements of this class are the graphs that are associated with Thue specifications. The latter have been introduced by Knapik in [Kna96] and they partly consist of a string-rewriting system $\mathcal{S}$ and a rational set R whose elements are irreducible by $\mathcal{S}$. The graph associated with such a specification is defined as follows. Its set of vertices is R and there is an edge labeled by a letter a between r_1 and r_2 if and only if r_1a reduces by $\mathcal{S}$ into r_2.

Originally, Thue specifications have been introduced as a specification and verification technique for infinite state processes. One of their interests is that they allow expression of dynamic properties such as deadlock freeness in a syntactic way thanks to language theory and string-rewriting systems. Moreover, automatic proof and string-rewriting techniques can be used within approaches based on Thue specifications to verify these properties. One such method is currently under investigation [Kna00] but it is limited to one Thue specification. In order to extend this method to sets of interconnected specifications, we introduce a construction of a composed Thue specification from given Thue specifications and a synchronization constraint. As explained above, each component of the system is specified separately and our construction yields a single specification which represents the behaviour of the whole system. Hence, we may apply to the latter the verification method of [Kna00] or its future extensions.

A major key point in Thue specifications is that they provide a simple and uniform formalism to describe several classes of infinite graphs. In [Cau00], Caucal introduces special kinds of word-rewriting systems: prefix/suffix and left/right systems. These systems yield a uniform characterization of prefix-recognizable (1) and of rational (2) graphs. Concerning (1), Caucal translates the prefix rewriting of systems into the rewriting of prefix systems. Concerning (2), he translates the mechanism of transducers into the rewriting of right systems.

In this paper, we present some formal verification-oriented results about the class of graphs of Thue specifications. First, we prove that this class is closed, up to isomorphism, under synchronized product. Then, we show that every rational graph with no edge labeled by the empty word is isomorphic to the graph of a Thue specification. A consequence of this result is that the first-order theory of the graphs of Thue specifications is undecidable. Finally, we consider connections between graphs of Thue specifications and those of Turing machines. We show that the graph of each Thue specification is observationally equivalent to that of a Turing machine. Nevertheless, the reverse does not hold.

This article is organized as follows. Section 2 introduces some basic notations and definitions from languages theory and about graphs, Thue specifications, Turing machines and rational graphs. Section 3 is devoted to the specification of systems of interacting processes by means of Thue specifications. We investigate the closure of the class of graphs of Thue specifications under synchronized product. Section 4 is related to the problem of verifying processes modeled by Thue specifications. We prove that the first-order theory of graphs of Thue specifications is undecidable. Section 5 establishes a connection between the class of graphs of Thue specifications and that of Turing machines. Brief conclusions close the paper.

2. Preliminaries

The reader is expected to have a smattering of formal languages and Turing machines. Nevertheless, the basic material for this paper is reviewed in the present section.

Throughout this article, if $n \in \mathbb{N}$, $[n]$ stands for the set $\{1, \ldots, n\}$ (with $[0] = \varnothing$). The domain of a binary relation R is written $\mathrm{Dom}(\mathrm{R})$ and its range is written $\mathrm{Ran}(\mathrm{R})$. If $\vec{t}$ is a tuple, $\pi_i(\vec{t})$ stands for the i^{th} coordinate of $\vec{t}$.

2.1. Alphabets, Words and Languages

In the scope of this paper, we consider finite alphabets, generally denoted by Σ or Γ. If Σ is a finite alphabet, we denote by Σ^* the free monoid generated by Σ. The neutral element of this monoid is the empty word, written ε and Σ^+ is the set $\Sigma^* \smallsetminus \{\varepsilon\}$. The set $\Sigma \cup \{\varepsilon\}$ is written $\widetilde{\Sigma}$. The length of a word w is denoted by $|w|$. The i^{th} character of w is written $w(i)$. The set of suffixes and the set of non empty suffixes of w respectively are

$$\mathrm{Suff}(w) = \{v \in \Sigma^* \mid \exists v' \in \Sigma^* \,.\, v'v = w\} \quad \text{and} \quad \mathrm{Suff}^+(w) = \mathrm{Suff}(w) \smallsetminus \{\varepsilon\} \;.$$

If $w = xyz$ (where x, y and z are elements of Σ^*), the word y is said to be a factor of w. Finally, an n-tuple, every coordinate of which is the empty word, is written $\vec{\varepsilon}_n$.

2.2. Graphs

A simple directed edge-labeled graph $\mathcal{G}$ (or more simply a *graph*) over alphabet Σ is a set of *edges*, *i.e.* a subset of $V \times \Sigma \times V$ where V is an arbitrary set, the elements of which are called the *vertices* of $\mathcal{G}$. In the sequel of this paper, we shall frequently consider graphs that are labeled by an alphabet extended by the empty word, *e.g.* $\widetilde{\Sigma}$. Given v and v' in V, an edge from

v to v' labeled by $c \in \widetilde{\Sigma}$ is written $v \xrightarrow[\mathcal{G}]{c} v'$. Thus, $\xrightarrow[\mathcal{G}]{c}$ is a binary relation on V for each $c \in \widetilde{\Sigma}$. In the following, $\overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$ stands for the reflexive-transitive closure of $\xrightarrow[\mathcal{G}]{\varepsilon}$ and $\overset{a}{\underset{\mathcal{G}}{\dashrightarrow}} = \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}} \circ \xrightarrow[\mathcal{G}]{a} \circ \overset{\varepsilon}{\underset{\mathcal{G}}{\dashrightarrow}}$ for any a in Σ.

A (finite) *path* in $\mathcal{G}$ from v to v' is a sequence of edges of the form $v_1 \xrightarrow[\mathcal{G}]{c_1} v_2, \dots, v_{n-1} \xrightarrow[\mathcal{G}]{c_{n-1}} v_n$ such that $n \in \mathbb{N}$, $v_1 = v$ and $v_n = v'$ (consequently, a path may be empty). The word $w = c_1 \dots c_n$ is then the *label* of the path. In this case, we may write $v \xrightarrow[\mathcal{G}]{w} v'$.

We shall sometimes consider graphs, the vertices of which are all accessible from some distinguished vertex. Thus, a graph $\mathcal{G}$ is said to be *rooted on a vertex* v if there exists a path from v to each vertex of $\mathcal{G}$.

Synchronized Product of Graphs. The synchronized product of graphs has been introduced by Arnold and Nivat [AN82, Niv79]. It is an essential part of the semantic of interacting processes. For more material, the reader may refer to [Arn94]. Given n alphabets $\Sigma_1, \dots, \Sigma_n$, a *synchronization constraint* $\mathcal{C}$ over $\Sigma_1, \dots, \Sigma_n$ is a subset of $\prod_{i \in [n]} \widetilde{\Sigma_i}$. Let $\mathcal{G}_i$, $i \in [n]$, be some graphs such that for each $i \in [n]$, the set of vertices of $\mathcal{G}_i$ is V_i and $\mathcal{G}_i$ is labeled by Σ_i. The *synchronized product* of the $\mathcal{G}_i$'s with respect to constraint $\mathcal{C}$, written $\prod_{i \in [n]}^{\mathcal{C}} \mathcal{G}_i$, is the graph

$$\mathcal{G} = \left\{ (v, c, v') \in V \times \mathcal{C} \times V \;\middle|\; \forall i \in [n],\ \pi_i(v) \xrightarrow[\mathcal{G}_i]{\pi_i(c)} \pi_i(v') \in \mathcal{G}_i \right\}$$

where V stands for the set $\prod_{i \in [n]} V_i$.

ε-Equivalence of Rooted Graphs. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two rooted graphs over Σ with set of vertices V_1 and V_2 respectively. Suppose that r_1 (resp. r_2) is a distinguished root of $\mathcal{G}_1$ (resp. $\mathcal{G}_2$). These graphs are said to be *ε-equivalent* if there exists a relation $\leftrightsquigarrow \subseteq V_1 \times V_2$ such that:

1. $\mathrm{Dom}(\leftrightsquigarrow) = V_1$ and $\mathrm{Ran}(\leftrightsquigarrow) = V_2$;
2. $r_1 \leftrightsquigarrow r_2$;
3. for each a in Σ, each path $v_1 \overset{a}{\underset{\mathcal{G}_1}{\dashrightarrow}} v_1'$ in $\mathcal{G}_1$ and each vertex v_2 of $\mathcal{G}_2$ such that $v_1 \leftrightsquigarrow v_2$, there exists a vertex v_2' in $\mathcal{G}_2$ such that $v_1' \leftrightsquigarrow v_2'$ and $v_2 \overset{a}{\underset{\mathcal{G}_2}{\dashrightarrow}} v_2'$;
4. for each a in Σ, each path $v_2 \overset{a}{\underset{\mathcal{G}_2}{\dashrightarrow}} v_2'$ in $\mathcal{G}_2$ and each vertex v_1 of $\mathcal{G}_1$ such that $v_1 \leftrightsquigarrow v_2$, there exists a vertex v_1' in $\mathcal{G}_1$ such that $v_1' \leftrightsquigarrow v_2'$ and $v_1 \overset{a}{\underset{\mathcal{G}_1}{\dashrightarrow}} v_1'$.

It should be noted that ε-equivalence is an observational equivalence (also called weak bisimulation) as defined by Milner [Mil80] if one considers ε to be a non-observable event.

2.3. Thue Specifications and Their Graphs

Semi-Thue Systems. A semi-Thue system $\mathcal{S}$ (an *sts* for short) on an alphabet Σ is a subset of $\Sigma^* \times \Sigma^*$. A pair (l, r) of $\mathcal{S}$ is called *(rewrite) rule* and is written $l \to r$, the word l (resp. r) is its left-hand (resp. right-hand) side. The *single-step reduction relation* induced by $\mathcal{S}$ on Σ^* is the binary relation

$$\underset{\mathcal{S}}{\rightarrow} = \big\{(u,v) \in \Sigma^* \times \Sigma^* \mid \exists x, y \in \Sigma^*,\ \exists (l,r) \in \mathcal{S},\ u = xly \text{ and } v = xry\big\}\ .$$

The *reduction relation* on Σ^* induced by $\mathcal{S}$ is the reflexive and transitive closure of $\underset{\mathcal{S}}{\rightarrow}$ and is denoted by $\overset{*}{\underset{\mathcal{S}}{\rightarrow}}$. A word u is *reducible* by $\mathcal{S}$ if u belongs to $\mathrm{Dom}(\underset{\mathcal{S}}{\rightarrow})$. Otherwise, u is *irreducible* by $\mathcal{S}$. The set of all words that are irreducible by $\mathcal{S}$, written $\mathrm{Irr}(\mathcal{S})$, is rational whenever $\mathrm{Dom}(\mathcal{S})$ is. Indeed, it is easy to see that $\mathrm{Irr}(\mathcal{S}) = \Sigma^* \setminus \Sigma^*\mathrm{Dom}(\mathcal{S})\Sigma^*$.

A word v is a *normal form* of u if v is irreducible and $u \overset{*}{\underset{\mathcal{S}}{\rightarrow}} v$. The set of normal forms of u is written $u{\downarrow_{\mathcal{S}}}$.

Particular Kinds of Semi-Thue Systems. Let $\mathcal{S}$ be an sts over alphabet Σ.

- $\mathcal{S}$ is *linear* if there exists a linear function f from $\mathbb{N}$ to $\mathbb{N}$ such that

$$\forall w, w' \in \Sigma^*,\ w \overset{*}{\underset{\mathcal{S}}{\rightarrow}} w' \Rightarrow |w'| \leq f\big(|w|\big)\ .$$

- $\mathcal{S}$ is *length-decreasing* if, for each $l \rightarrow r \in \mathcal{S}$, $|l| \geq |r|$. Obviously, any length-decreasing sts is linear.

Thue Specifications. A Thue specification is a four-tuple $(\Omega, \Delta, \mathcal{S}, R)$ where Ω is a finite *state alphabet*, Δ is a finite *event alphabet*, $\mathcal{S}$ is a semi-Thue system on $\Omega \cup \Delta$ and $R \subseteq \Omega^*$ is a rational subset of $\mathrm{Irr}(\mathcal{S})$.

A *rooted Thue specification* is a five-tuple $(\Omega, \Delta, \mathcal{S}, R, u)$ where $(\Omega, \Delta, \mathcal{S}, R)$ is a Thue specification and u is a word of R. Notice that a consequence of this definition is that $\mathcal{S}$ does not contain any rule, the left-hand side of which is the empty word (else, $\mathrm{Irr}(\mathcal{S}) = \varnothing$, $R = \varnothing$ and there cannot be a u satisfying the definition).

A *linear* (resp. *length-decreasing*) *Thue specification* is a Thue specification, the sts of which is linear (resp. length-decreasing).

Graph of a Thue Specification. Let $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R)$ be a Thue specification. The graph of $\mathcal{T}$, written $\mathcal{G}(\mathcal{T})$, is defined as follows. The vertices of $\mathcal{G}(\mathcal{T})$ are the words of R, the edges of $\mathcal{G}(\mathcal{T})$ are labeled by the letters of Δ and there is an edge labeled by a from v to w whenever w is a normal form of va.

The graph of a rooted Thue specification $(\Omega, \Delta, \mathcal{S}, R, u)$ is the maximal subgraph of Thue specification $(\Omega, \Delta, \mathcal{S}, R)$ that is rooted on vertex u.

The family of the graphs of rooted Thue specifications is denoted by $\mathcal{G}[\textsc{ts}]$. The subfamilies of $\mathcal{G}[\textsc{ts}]$ corresponding to special kinds of rooted Thue specifications are denoted as follows: $\mathcal{G}[\textsc{Lin-ts}]$ for the graphs of linear rooted Thue specifications and $\mathcal{G}[\textsc{Len-ts}]$ for the graphs of length-decreasing rooted Thue specifications.

2.4. Turing Machines and Their Graphs

Turing Machines. In the scope of this paper, we use a definition of a Turing machine that is inspired by that of an off-line Turing machine (see *e.g.* [MS97] for details). A Turing machine is made up of a read-only *input tape* and of *work-tapes*. It cannot write on its input tape and all computations are done on the work tapes that are infinite to the left and to the right. Initially, an *input word* is stored on the input tape and the work tapes only contain *blank characters*. Then, the input tape head reads the input word from the left to the right (with the possibility of stopping and then resuming its motion) and computations are made on the work tapes. Note that the input tape head of these Turing machines is one-way, from the left to the right, unlike the motion of an off-line Turing machine input tape head, which is two-ways. Moreover, there are no accepting states. Indeed, our motivation does not rely on any languages theory aspect. Actually, we are interested in Turing machines as an approach for modeling process behaviour that is defined as a graph associated to a machine.

Formally, a k work tape Turing machine $\mathcal{M}$ over alphabet Σ is a tuple $(Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ where Q is the finite set of *states*, Σ is the *input alphabet*, $\Gamma_1, \ldots, \Gamma_k$ are the *work tape alphabets*, q_0 is the *initial state*, and δ is the set of *transitions*:

$$\delta \subseteq Q \times \widetilde{\Sigma} \times \Gamma_1 \times \cdots \times \Gamma_k \times \Gamma_1 \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times \cdots \times \Gamma_k \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times Q$$

where $\blacktriangleleft$ (resp. $\blacktriangleright$ and $\blacksquare$) symbolizes a move to the left (resp. a move to the right and no move) and an input ε represents the case where the input tape head of $\mathcal{M}$ does not move and does not read any character. We assume that the blank character, written $\square$, belongs to each Γ_i.

An *ε-transition* is an element of the set

$$Q \times \{\varepsilon\} \times \Gamma_1 \times \cdots \times \Gamma_k \times \Gamma_1 \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times \cdots \times \Gamma_k \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times Q\ .$$

An *internal configuration* $(\mu_1 q \nu_1, \ldots, \mu_k q \nu_k)$ of $\mathcal{M}$ is an element of the set ${\Gamma_1}^*.Q.{\Gamma_1}^* \times \cdots \times {\Gamma_k}^*.Q.{\Gamma_k}^*$. This encodes the description of $\mathcal{M}$ at a time as follows: q is the current state, for all i in $[k]$, $\mu_i \nu_i$ is the content of the i^{th} work tape from the leftmost non-blank character to the rightmost non-blank character and for all i in $[k]$, the head of the i^{th} work tape is reading ν_i's first character or $\square$ if ν_i is empty. Note that for all i in $[k]$, both μ_i or ν_i may contain $\square$. An internal configuration, every coordinate of which equals q_0, is called *initial configuration* of $\mathcal{M}$.

Linear Bounded Machines. Turing machine $\mathcal{L} = (Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ is linear bounded if there exist k linear functions from $\mathbb{N}$ to $\mathbb{N}$ denoted by $S_1, \ldots, S_k$ such that, when computing an input, the length of which is n, $\mathcal{L}$ uses at most $S_i(n)$ cells of the i^{th} work tape.

Real-time Turing Machines. A Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma_1, \ldots, \Gamma_k, \delta, q_0)$ is said to be real-time if it never reads ε on its input tape *i.e.* if

$$\delta \subseteq Q \times \Sigma \times \Gamma_1 \times \cdots \times \Gamma_k \times \Gamma_1 \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times \cdots \times \Gamma_k \times \{\blacktriangleleft, \blacktriangleright, \blacksquare\} \times Q\ .$$

It is clear that $\mathcal{M}$ is also a linear bounded machine. Indeed, when computing an input, the length of which is n, $\mathcal{M}$ uses at most n cells on each work tape.

Finite Automata. A finite automaton is given by a finite set Q of states, a finite alphabet Σ, a subset δ of $Q \times \widetilde{\Sigma} \times Q$ called the set of transitions, an element q_0 of Q called the initial state and a subset F of Q called the set of final states. It is denoted by the five-tuple $(Q, \Sigma, \delta, q_0, F)$. The *language* of a finite automaton $\mathcal{A}$, written $\mathrm{Lang}(\mathcal{A})$, is the set of all words w such that there exists a sequence $(q_0, c_1, q_1), \dots, (q_{n-1}, c_n, q_n)$ of transitions satisfying $c_1 \dots c_n = w$ and $q_n \in F$.

Graph Associated to a Turing Machine. Let $\mathcal{M} = (Q, \Sigma, \Gamma_1, \dots, \Gamma_k, \delta, q_0)$ be a Turing machine. Let $\mathcal{G}$ be the graph defined as follows. The vertices of $\mathcal{G}$ are all the internal configurations of $\mathcal{M}$, the labels of $\mathcal{G}$ belong to $\widetilde{\Sigma}$ and $(\mu_1 q \nu_1, \dots, \mu_k q \nu_k) \xrightarrow[\mathcal{G}]{c} (\mu'_1 q' \nu'_1, \dots, \mu'_k q' \nu'_k)$ is an edge of $\mathcal{G}$ if and only if for each $i \in [k]$, there exist $X_i, Y_i \in \Gamma_i$ and $\blacklozenge_i \in \{\blacktriangleleft, \blacktriangleright, \blacksquare\}$ such that

$$(q, c, X_1, \dots, X_k, Y_1, \blacklozenge_1, \dots, Y_k, \blacklozenge_k, q') \in \delta$$

and either $\blacklozenge_i = \blacktriangleleft$ and one of the following holds:

- $\exists \alpha_i, \beta_i \in {\Gamma_i}^*$, $\exists Z_i \in \Gamma_i$, $\mu_i = \alpha_i Z_i$, $\nu_i = X_i \beta_i$, $\mu'_i = \alpha_i$ and, if $Z_i Y_i \beta_i \neq \Box\Box$, then $\nu'_i = Z_i Y_i \beta_i$, else $\nu'_i = \varepsilon$;
- $\mu_i = \varepsilon$, $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\mu'_i = \varepsilon$ and, if $Y_i \alpha_i \neq \Box$, then $\nu'_i = \Box Y_i \alpha_i$, else $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\exists \alpha_i \in {\Gamma_i}^*$, $\exists Z_i \in \Gamma_i$, $\mu_i = \alpha_i Z_i$, $\nu_i = \varepsilon$, $\mu'_i = \alpha_i$ and, if $Z_i Y_i \neq \Box\Box$, then $\nu'_i = Z_i Y_i$, else $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\mu_i = \varepsilon$, $\nu_i = \varepsilon$, $\mu'_i = \varepsilon$ and, if $Y_i \neq \Box$, then $\nu'_i = \Box Y_i$, else $\nu'_i = \varepsilon$;

or $\blacklozenge_i = \blacktriangleright$ and one of the following holds:

- $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\nu'_i = \alpha_i$ and, if $\mu_i Y_i \neq \Box$, then $\mu'_i = \mu_i Y_i$ else $\mu'_i = \varepsilon$;
- $X_i = \Box$, $\nu_i = \varepsilon$, $\nu'_i = \varepsilon$, and if $\mu_i Y_i \neq \Box$, then $\mu'_i = \mu_i Y_i$, else $\mu'_i = \varepsilon$;

or $\blacklozenge_i = \blacksquare$ and one of the following holds:

- $\exists \alpha_i \in {\Gamma_i}^*$, $\nu_i = X_i \alpha_i$, $\mu'_i = \mu_i$ and, if $Y_i \alpha_i \neq \Box$, then $\nu'_i = Y_i \alpha_i$, else $\nu'_i = \varepsilon$;
- $X_i = \Box$, $\nu_i = \varepsilon$, $\mu'_i = \mu_i$ and, if $Y_i \neq \Box$, then $\nu'_i = Y_i$, else $\nu'_i = \varepsilon$.

The graph of $\mathcal{M}$, written $\mathcal{G}(\mathcal{M})$, is the maximal subgraph of $\mathcal{G}$ that is rooted on the initial configuration of $\mathcal{M}$.

The family of the graphs of Turing machines is denoted by $\mathcal{G}[\textsc{tm}]$. The subfamilies of $\mathcal{G}[\textsc{tm}]$ corresponding to special kinds of Turing machines are denoted as follows: $\mathcal{G}[\textsc{Lin-tm}]$ for the graphs of linear bounded machines and $\mathcal{G}[\textsc{Real-tm}]$ for the graphs of real-time Turing machines.

2.5. Transducers and Rational Graphs

The machines that realize rational binary relations are called *transducers*. Formally, a transducer $\mathcal{K}$ is a six-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q is the finite set of states, Σ is the *input alphabet*, Γ is the *output alphabet*, $\delta \subseteq Q \times Z^* \times Z^* \times Q$ is the finite set of transitions, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. The relation $\textsc{r}$ that is *realized* by $\mathcal{K}$ is defined as: $\textsc{r} \subseteq \Sigma^* \times \Gamma^*$ and for each $w \in \Sigma^*$, for each $w' \in \Gamma^*$, $(w, w') \in \textsc{r}$ if and only if there exists a sequence $(q_0, x_1, y_1, q_1) \dots (q_{n-1}, x_n, y_n, q_n)$ of transitions satisfying $x_1 \dots x_n = w$, $y_1 \dots y_n = w'$ and $q_n \in F$.

Let Σ and Z be two finite alphabets and $\mathcal{G}$ be a graph, the vertices of which belong to Z^*, that is labeled by Σ. This graph is said to be *rational* if, for each $a \in \Sigma$, the relation $\xrightarrow[\mathcal{G}]{a}$ is rational, *i.e.* there exists a transducer that realizes $\xrightarrow[\mathcal{G}]{a}$.

3. Closure under Synchronized Product

Now, we prove that the class of graphs of Thue specifications is closed under synchronized product. For this aim, we construct a Thue specification, the graph of which is isomorphic to the synchronized product of the graphs of the original ones.

3.1. Synchronized Product of Thue Specifications

Let $n \geq 2$ be an integer and $\mathcal{T}_i = (\Omega_i, \Delta_i, \mathcal{S}_i, R_i)$, $i \in [n]$, be some Thue specifications. Let $\mathcal{C} \subseteq \prod_{i\in[n]} \Delta_i$ be a synchronization constraint. The *synchronized product* of the $\mathcal{T}_i$'s according to $\mathcal{C}$ is the Thue specification

$$\prod_{i\in[n]}^{\mathcal{C}} \mathcal{T}_i = \Big(\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}, \mathcal{C}, \mathcal{S}, R\Big)$$

where $\mathcal{S}$ and R are defined in this section. We first give a few definitions that will be useful in the sequel.

- $\mathcal{L}$ is the following subset of $\big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^*$:

$$\left\{ w \neq \varepsilon \;\middle|\; \begin{array}{l} \forall i \in [\,|w|-1],\ \forall j \in [n],\ \pi_j\big(w(i)\big) = \varepsilon \Rightarrow \pi_j\big(w(i+1)\big) = \varepsilon \text{ and} \\ \forall i \in [\,|w|\,],\ \exists j \in [n],\ \pi_j\big(w(i)\big) \neq \varepsilon \end{array} \right\} \bigcup \{\varepsilon\}\,.$$

- For each $i \in [n]$, $\overline{\pi}_i$ stands for the mapping π_i extended to $\big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^*$ in the following way:

$$\begin{array}{rccl} \overline{\pi}_i : & \big[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\big]^* & \longrightarrow & (\Omega_i \cup \Delta_i)^* \\ & w \neq \varepsilon & \longmapsto & \pi_i\big(w(1)\big) \dots \pi_i\big(w(\,|w|\,)\big) \\ & \varepsilon & \longmapsto & \varepsilon\,. \end{array}$$

- ϕ is the one-to-one correspondence:

$$\begin{array}{rccl} \phi : & \prod_{i\in[n]} (\Omega_i \cup \Delta_i)^* & \longrightarrow & \mathcal{L} \\ & \vec{w} & \longmapsto & w' \quad \text{such that } \forall i \in [n],\ \overline{\pi}_i\big(w'\big) = \pi_i\big(\vec{w}\big)\,. \end{array}$$

In the sequel of this section, for better readability, tuples $(a_1, \dots, a_n)$ will sometimes be represented in the vertical way $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$.

Example 3.1. Suppose that $n = 3$ and that for each $i \in [n]$, $\Omega_i = \Delta_i = \{a, b, c\}$. Then, $\mathcal{L}$ contains

$$\begin{pmatrix} a \\ a \\ b \end{pmatrix}\begin{pmatrix} b \\ b \\ b \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} a \\ b \\ c \end{pmatrix}\begin{pmatrix} b \\ \varepsilon \\ a \end{pmatrix}\begin{pmatrix} \varepsilon \\ \varepsilon \\ a \end{pmatrix}$$

but it does not contain

$$\begin{pmatrix}\varepsilon\\\varepsilon\\\varepsilon\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}a\\b\\c\end{pmatrix}\begin{pmatrix}b\\\varepsilon\\a\end{pmatrix}\begin{pmatrix}b\\a\\\varepsilon\end{pmatrix}.$$

On the other hand,

$$\phi\begin{pmatrix}\varepsilon\\\varepsilon\\\varepsilon\end{pmatrix} = \varepsilon \quad \text{and} \quad \phi\begin{pmatrix}ab\\b\\caa\end{pmatrix} = \begin{pmatrix}a\\b\\c\end{pmatrix}\begin{pmatrix}b\\\varepsilon\\a\end{pmatrix}\begin{pmatrix}\varepsilon\\\varepsilon\\a\end{pmatrix}.$$

The Semi-Thue System $\mathcal{S}$. This system is the union of a set $\mathcal{S}_a$ of *arrangement rules* and a set $\mathcal{S}_s$ of *simulation rules.* Arrangement rules are:

- each $\vec{t}_1\vec{t}_2 \to \vec{u}_1\vec{u}_2$ such that
  - $\vec{t}_1$, $\vec{t}_2$, $\vec{u}_1$ and $\vec{u}_2$ are some elements of $\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}$ and
  - $\vec{t}_1 \neq \vec{\varepsilon}_n$ and
  - if $S = \{i \in [n] \mid \pi_i(\vec{t}_1) = \varepsilon$ and $\pi_i(\vec{t}_2) \neq \varepsilon\}$, then $S \neq \varnothing$ and
    - $\forall i \in S$, $\pi_i(\vec{u}_2) = \varepsilon$ and $\pi_i(\vec{u}_1) = \pi_i(\vec{t}_2)$ and
    - $\forall i \notin S$, $\pi_i(\vec{u}_1) = \pi_i(\vec{t}_1)$ and $\pi_i(\vec{u}_2) = \pi_i(\vec{t}_2)$,
- and $\vec{\varepsilon}_n \to \varepsilon$.

The following lemma emphasizes the role of $\mathcal{S}_a$.

Lemma 3.1. *The rewriting system $\mathcal{S}_a$ is convergent. Moreover, for each word w in the language $\left[\prod_{i\in[n]} \widetilde{\Omega_i} \cup \widetilde{\Delta_i}\right]^* \setminus \mathcal{L}$, $w{\downarrow_{\mathcal{S}_a}}$ is the word w' in $\mathcal{L}$ satisfying: for each $i \in [n]$, $\pi_i(w') = \pi_i(w)$. Each word in $\mathcal{L}$ is irreducible by $\mathcal{S}_a$.*

The set $\mathcal{S}_s$ of simulation rules is the set of all rules $w \to w'$ such that

- $w \in \mathcal{L}$ and $w' \in \mathcal{L}$,
- $\exists i \in [n]$, $\overline{\pi}_i(w) \to \overline{\pi}_i(w') \in \mathcal{S}_i$ and $\forall j \in [\,|w|\,]$, $\pi_i(w(j)) \neq \varepsilon$,
- $\forall j \in [n] \setminus \{i\}$, $\overline{\pi}_j(w') = \overline{\pi}_j(w)$.

These rules allow to apply, for each i in $[n]$, the elements of $\mathcal{S}_i$ to the i^{th} projection of words of $\mathcal{L}$ without modifying the other projections.

Example 3.2. Suppose that $n = 2$, $\Omega_1 = \Delta_1 = \{a_1\}$, $\Omega_2 = \Delta_2 = \{a_2\}$, $\mathcal{S}_1 = \{a_1a_1 \to a_1\}$ and $\mathcal{S}_2 = \{a_2 \to a_2a_2\}$. The set $\mathcal{S}_a$ of arrangement rules is

$$\mathcal{S}_a = \left\{\binom{\varepsilon}{a_2}\binom{a_1}{\varepsilon} \to \binom{a_1}{a_2}\binom{\varepsilon}{\varepsilon},\ \binom{\varepsilon}{a_2}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2},\right.$$

$$\left.\binom{a_1}{\varepsilon}\binom{\varepsilon}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{\varepsilon},\ \binom{a_1}{\varepsilon}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{a_1}{\varepsilon}, \binom{\varepsilon}{\varepsilon} \to \varepsilon\right\}.$$

The set $\mathcal{S}_s$ of simulation rules is

$$\mathcal{S}_s = \left\{ \binom{a_1}{\varepsilon}\binom{a_1}{\varepsilon} \to \binom{a_1}{\varepsilon},\ \binom{a_1}{a_2}\binom{a_1}{\varepsilon} \to \binom{a_1}{a_2},\ \binom{a_1}{a_2}\binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2} \right\} \cup$$

$$\left\{ \binom{\varepsilon}{a_2} \to \binom{\varepsilon}{a_2}\binom{\varepsilon}{a_2}, \binom{a_1}{a_2} \to \binom{a_1}{a_2}\binom{\varepsilon}{a_2} \right\} .$$

The first (resp. second) part of this union corresponds to rules generated by $\mathcal{S}_1$ (resp. $\mathcal{S}_2$).

The Rational Language R. For each $i \in [n]$, let $\mathcal{A}_i = (Q_i, \Omega_i, \delta_i, q_{0i}, F_i)$ be a finite automaton with no ε-transition that recognizes R_i. Our goal is to prove that $\phi\big(\prod_{i\in[n]} R_i\big)$ is rational by constructing from the $\mathcal{A}_i$'s a finite automaton that recognizes this language. As every word of $\phi\big(\prod_{i\in[n]} R_i\big)$ is irreducible by each arrangement rule (because $\mathrm{Ran}(\phi) \subseteq \mathcal{L}$) and by each simulation rule (this is due to the way we have constructed $\mathcal{S}_s$ and to the fact that for each $i \in [n]$, $R_i \subseteq \mathrm{Irr}(\mathcal{S}_i)$), we could then define R as $R = \phi\big(\prod_{i\in[n]} R_i\big)$.

Here is a way to construct a finite automaton that recognizes $\phi\big(\prod_{i\in[n]} R_i\big)$. The idea is to make a product of the $\mathcal{A}_i$'s. Nevertheless, we have to add in a first way ε-transitions to each $\mathcal{A}_i$ if we want this product to recognize every word of $\phi\big(\prod_{i\in[n]} R_i\big)$. Effectively, if, for instance, $n = 2$, $R_1 = \{a\}$ and $R_2 = \{aaa\}$, the word $(a,a)(\varepsilon,a)(\varepsilon,a)$ is in $\phi\big(R_1 \times R_2\big)$ because it equals $\phi\big(a, aaa\big)$. Consequently, we associate to each $\mathcal{A}_i$ the automaton $\mathcal{B}_i$ defined as:

$$\mathcal{B}_i = \big(Q_i \cup \{f'_i\},\ \widetilde{\Omega_i},\ \delta_i \cup \delta'_i,\ q_{0i},\ F_i \cup \{f'_i\}\big) \text{ where:}$$

- f'_i is a new state that does not belong to Q_i and
- $\delta'_i = \big\{\big(f, \varepsilon, f'_i\big) \mid f \in F_i\big\} \cup \big\{(f'_i, \varepsilon, f'_i)\big\}$.

Let $\mathcal{B}$ be the product of the $\mathcal{B}_i$'s according to constraint $\prod_{i\in[n]} \widetilde{\Omega_i} \setminus \{\vec{\varepsilon}_n\}$, *i.e.*

$$\mathcal{B} = \Big(\prod_{i\in[n]} Q_i \cup \{f'_i\},\ \prod_{i\in[n]} \widetilde{\Omega_i} \setminus \{\vec{\varepsilon}_n\},\ \delta,\ (q_{01}, \ldots, q_{0n}),\ \prod_{i\in[n]} F_i \cup \{f'_i\} \Big)$$

where $\delta = \big\{(q, a, q') \mid a \neq \vec{\varepsilon}_n \text{ and } \forall i \in [n],\ \big(\pi_i(q), \pi_i(a), \pi_i(q')\big) \in \delta_i \cup \delta'_i\big\}$. It follows from construction of the $\mathcal{B}_i$'s that the language recognized by $\mathcal{B}$ is $\phi\big(\prod_{i\in[n]} R_i\big)$.

Consequently, $\phi\big(\prod_{i\in[n]} R_i\big)$ is rational and as it is also included in $\mathrm{Irr}(\mathcal{S})$, we define R as

$$R = \phi\big(\prod_{i\in[n]} R_i\big) \ .$$

Example 3.3. Consider rational languages $\{\varepsilon, x\}$ and a^*. Here are two finite automata with no ε-transition that recognize these languages:

$$\mathcal{A}_1 = \Big(\{q_0, q_1\}, \{x\}, \{(q_0, x, q_1)\}, q_0, \{q_0, q_1\}\Big) \text{ and } \mathcal{A}_2 = \Big(\{p_0\}, \{a\}, \{(p_0, a, p_0)\}, p_0, \{p_0\}\Big) \ .$$

From $\mathcal{A}_1$ and $\mathcal{A}_2$ we construct:

$$\mathcal{B}_1 = \Big(\{q_0, q_1, f_1'\}, \{x\}, \{(q_0, x, q_1), (q_0, \varepsilon, f_1'), (q_1, \varepsilon, f_1'), (f_1', \varepsilon, f_1')\}, q_0, \{q_0, q_1, f_1'\}\Big) \quad \text{and}$$

$$\mathcal{B}_2 = \Big(\{p_0, f_2'\}, \{a\}, \{(p_0, a, p_0), (p_0, \varepsilon, f_2'), (f_2', \varepsilon, f_2')\}, p_0, \{p_0, f_2'\}\Big)\ .$$

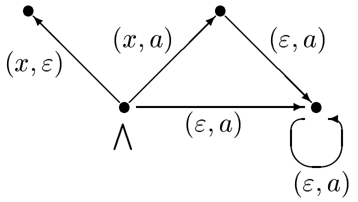
The product of $\mathcal{B}_1$ and $\mathcal{B}_2$ according to constraint $\big(\{x,\varepsilon\} \times \{a,\varepsilon\}\big) \smallsetminus \{(\varepsilon,\varepsilon)\}$ is such that

- its set of states is $\{(q_0,p_0),(q_1,p_0),(f_1',p_0),(q_0,f_2'),(q_1,f_2'),(f_1',f_2')\}$,
- its alphabet is $\{(\varepsilon,a),(x,\varepsilon),(x,a)\}$,
- its set of transitions is

$$\begin{array}{ll} \{ & \big((q_0,p_0),(\varepsilon,a),(f_1',p_0)\big), \big((q_0,p_0),(x,a),(q_1,p_0)\big), \big((q_0,p_0),(x,\varepsilon),(q_1,f_2')\big), \\ & \big((q_1,p_0),(\varepsilon,a),(f_1',p_0)\big), \big((f_1',p_0),(\varepsilon,a),(f_1',p_0)\big), \big((q_0,f_2'),(x,\varepsilon),(q_1,f_2')\big) \quad \}\ , \end{array}$$

- its initial state is (q_0,p_0),
- its set of accepting states is $\{(q_0,p_0),(q_1,p_0),(f_1',p_0),(q_0,f_2'),(q_1,f_2'),(f_1',f_2')\}$.

The accessible part of this automaton may be depicted as follows:



The language of the automaton is $(x,\varepsilon) + (\varepsilon,a)^* + (x,a).(\varepsilon,a)^*$ *i.e.* $\phi\big(\{x,\varepsilon\} \times a^*\big)$.

3.2. Graphs $\mathcal{G}(\prod^{\mathcal{C}}_{i\in[n]} \mathcal{T}_i)$ and $\prod^{\mathcal{C}}_{i\in[n]} \mathcal{G}(\mathcal{T}_i)$ are isomorphic

The main result of this section (Theorem 3.1) is established here. The following lemmas and propositions will be used in the proof.

Lemma 3.2. *Let $\vec{x},\vec{z} \in \prod_{i\in[n]}(\Omega_i \cup \Delta_i)^*$ such that there exists $i \in [n]$ satisfying $\pi_i(\vec{x}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{z})$ and for each $j \neq i$, $\pi_j(\vec{z}) = \pi_j(\vec{x})$. Then, $\phi(\vec{x}) \xrightarrow[\mathcal{S}]{*} \phi(\vec{z})$.*

Proof. By induction on the length of the rewriting $\pi_i(\vec{x}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{z})$. □

Proposition 3.1. *Let $\vec{x},\vec{y} \in \prod_{i\in[n]} R_i$ and $\vec{a} \in \mathcal{C}$ such that, for each $i \in [n]$, we have the rewriting $\pi_i(\vec{x}).\pi_i(\vec{a}) \xrightarrow[\mathcal{S}_i]{*} \pi_i(\vec{y})$. Then, $\phi(\vec{x}).\vec{a} \xrightarrow[\mathcal{S}]{*} \phi(\vec{y})$.*

Proof:
Let $\oplus$ be the usual concatenation operation for product of monoids, *i.e.* for each tuples $(w_1,\ldots,w_n)$ and $(w_1',\ldots,w_n')$ in $\prod_{i\in[n]}(\Omega_i \cup \Delta_i)^*$, we have

$$(w_1,\ldots,w_n) \oplus (w_1',\ldots,w_n') = (w_1.w_1',\ldots,w_n.w_n')\ .$$

Let $(\vec{x}_i)_{i\leq n}$ be the sequence defined as:

- $\vec{x}_0 = \vec{x} \oplus \vec{a}$ and
- for each $i \in [n]$, $\vec{x}_i$ is the n-tuple defined as $\pi_i(\vec{x}_i) = \pi_i(\vec{y})$ and, for each $j \neq i$, $\pi_j(\vec{x}_i) = \pi_j(\vec{x}_{i-1})$.

For each $i \in [n]$, $\pi_i(\vec{x}_{i-1}) = \pi_i(\vec{x}_0) = \pi_i(\vec{x} \oplus \vec{a}) = \pi_i(\vec{x}).\pi_i(\vec{a})$. Consequently, we have the rewriting $\pi_i(\vec{x}_{i-1}) \underset{\mathcal{S}_i}{\overset{*}{\rightarrow}} \pi_i(\vec{y})$ *i.e.* $\pi_i(\vec{x}_{i-1}) \underset{\mathcal{S}_i}{\overset{*}{\rightarrow}} \pi_i(\vec{x}_i)$. Moreover, for each $j \neq i$, we have $\pi_j(\vec{x}_{i-1}) = \pi_j(\vec{x}_i)$. Consequently, according to Lemma 3.2, for each $i \in [n]$, we have $\phi(\vec{x}_{i-1}) \underset{\mathcal{S}}{\overset{*}{\rightarrow}} \phi(\vec{x}_i)$. Finally, as $\vec{x}_0 = \vec{x} \oplus \vec{a}$ and $\vec{x}_n = \vec{y}$, we have $\phi(\vec{x} \oplus \vec{a}) \underset{\mathcal{S}}{\overset{*}{\rightarrow}} \phi(\vec{y})$. As $\phi(\vec{x}).\phi(\vec{a}) \underset{\mathcal{S}_a}{\overset{*}{\rightarrow}} \phi(\vec{x} \oplus \vec{a})$ and $\phi(\vec{a}) = \vec{a}$, we get $\phi(\vec{x}).\vec{a} \underset{\mathcal{S}}{\overset{*}{\rightarrow}} \phi(\vec{y})$. □

Proposition 3.2. *Let* $\vec{x}, \vec{y} \in \prod_{i \in [n]} R_i$ *and* $\vec{a} \in \mathcal{C}$ *such that* $\phi(\vec{x}).\vec{a} \underset{\mathcal{S}}{\overset{*}{\rightarrow}} \phi(\vec{y})$. *Then, for each* $i \in [n]$, $\pi_i(\vec{x}).\pi_i(\vec{a}) \underset{\mathcal{S}_i}{\overset{*}{\rightarrow}} \pi_i(\vec{y})$.

Proof:
Let $w_0 \underset{\mathcal{S}}{\rightarrow} w_1 \underset{\mathcal{S}}{\rightarrow} \dots \underset{\mathcal{S}}{\rightarrow} w_p$ be a rewriting of $\phi(\vec{x}).\vec{a}$ into $\phi(\vec{y})$ (*i.e.* $w_0 = \phi(\vec{x}).\vec{a}$ and $w_p = \phi(\vec{y})$). The application of a rule of $\mathcal{S}_s$ to a word w_k of the rewriting corresponds to the application of a rule of a $\mathcal{S}_i$ to $\overline{\pi}_i(w_k)$ and does not modify the words $\overline{\pi}_j(w_k)$, $j \in [n] \setminus \{i\}$. Moreover, the arrangement rules do not modify the words $\overline{\pi}_j(w_k)$, $j \in [n]$. Consequently, w_p is such that for each $i \in [n]$, $\overline{\pi}_i(w_0) \underset{\mathcal{S}_i}{\overset{*}{\rightarrow}} \overline{\pi}_i(w_p)$. But $\overline{\pi}_i(w_0) = \overline{\pi}_i(\phi(\vec{x}).\vec{a}) = \overline{\pi}_i(\phi(\vec{x})).\pi_i(\vec{a})$, $\overline{\pi}_i(\phi(\vec{x})) \overset{\text{def.}\phi}{=} \pi_i(\vec{x})$ and $\overline{\pi}_i(\phi(\vec{y})) \overset{\text{def.}\phi}{=} \pi_i(\vec{y})$. Consequently, the result of the proposition holds. □

In view of Prop. 3.1 and Prop. 3.2, the following theorem holds.

Theorem 3.1. *Graphs* $\mathcal{G}(\prod^{\mathcal{C}}_{i \in [n]} \mathcal{T}_i)$ *and* $\prod^{\mathcal{C}}_{i \in [n]} \mathcal{G}(\mathcal{T}_i)$ *are isomorphic.*

4. A Connection with the Class of Rational Graphs

Let $\mathcal{G}$ be a rational graph, the vertices of which belong to Z^* (where Z is any non-empty finite alphabet). Suppose that $\mathcal{G}$ is labeled by Σ, a finite alphabet that does not contain the empty word. As $\mathcal{G}$ is rational, for each $a \in \Sigma$, $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$ is a rational transduction. For each $a \in \Sigma$, let

$$\mathcal{K}_a = (Q_a, Z, Z, \delta_a, q_0^a, F_a)$$

be a transducer that realizes $\underset{\mathcal{G}}{\overset{a}{\rightarrow}}$. Suppose that $Q_a \cap Z = \varnothing$ and that for each $b \in \Sigma$, $b \neq a \Rightarrow Q_a \cap Q_b = \varnothing$.

Our goal is to construct a Thue specification, the graph of which is isomorphic to $\mathcal{G}$. To this aim, we define transducer $\mathcal{K}$ that realizes $\bigcup_{a \in \Sigma} \underset{\mathcal{G}}{\overset{a}{\rightarrow}}$:

$$\mathcal{K} = \Big(\{q_0\} \cup \bigcup_{a \in \Sigma} Q_a, Z, Z, \big\{(q_0, \varepsilon, \varepsilon, q_0^a)\big\} \cup \bigcup_{a \in \Sigma} \delta_a, q_0, \bigcup_{a \in \Sigma} F_a\Big)$$

where q_0 is a new state that does not belong to $Z \ \cup\ \bigcup_{a\in\Sigma} Q_a$. We define from $\mathcal{K}$ the Thue specification:

$$\mathcal{T} = \big(Q \cup Z \cup \{\pounds\}, \Sigma, \mathcal{S}, R\big) \text{ such that}$$

- $\pounds$ is a new symbol that does not belong to $Q \cup \Sigma \cup Z$,
- sts $\mathcal{S}$ is

$$\begin{aligned}\mathcal{S} = \ & \big\{\pounds a \to a\pounds \mid a \in \Sigma\big\} \ \cup\ \big\{za \to az \mid a \in \Sigma,\ z \in Z\big\} \cup \\ & \big\{q_0 a \to q_0^a \mid a \in \Sigma\big\} \cup \\ & \big\{qw \to w'q' \mid (q,w,w',q') \in \textstyle\bigcup_{a\in\Sigma} \delta_a\big\} \cup \\ & \big\{f\pounds \to q_0\pounds \mid f \in \textstyle\bigcup_{a\in\Sigma} F_a\big\} \cup \\ & \big\{zq_0 \to q_0 z \mid z \in Z\big\},\end{aligned}$$

- $R = q_0\ .\ Z^*\ .\ \pounds$

Let ϕ be the one-to-one correspondence from Z^* to R that maps each $w \in Z^*$ to $q_0 w\pounds \in R$. It is clear that $w \xrightarrow[\mathcal{G}]{a} w'$ if and only if $q_0 w\pounds \xrightarrow[\mathcal{G}(\mathcal{T})]{a} q_0 w'\pounds$. Consequently, graphs $\mathcal{G}$ and $\mathcal{G}(\mathcal{T})$ are isomorphic. So, we have the following theorem.

Theorem 4.1. *The class of rational graphs with no ε-edge is included, up to isomorphism, in the class of graphs of Thue specifications.*

Morvan establishes in [Mor00] that the first-order theory of rational graphs is not decidable. His proof consists in reducing the Post's correspondence problem to the decidability of a first-order formula of the form $\exists x\ \mathbf{s}_a(x,x)$ on a rational graph. This proof also works for rational graphs with no ε-edge. Consequently, the following corollary holds.

Corollary 4.1. *The first-order theory of graphs of Thue specifications is not decidable.*

Notice that it is possible to associate to rational graphs $\mathcal{G}$ together with a distinguished root r a rooted Thue specification $\mathcal{T}$, the graph of which is isomorphic to that of $\mathcal{G}$. It suffices to consider the above construction with the following additional definition for the initial word u of $\mathcal{T}$: $u = q_0 r\pounds$. As Morvan's undecidability proof also holds for rooted rational graphs, we have the following result.

Theorem 4.2. *The first-order theory of graphs of rooted Thue specifications is not decidable.*

Notice that those undecidability results could be obtain from rather simple formulas, as that of the following example.

Example 4.1. Let $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R)$ be a Thue specification. The following formula belongs to the first-order theory of the graph of $\mathcal{T}$ if and only if $\mathcal{S}$ is confluent on $\mathrm{Dom}(\xrightarrow[\mathcal{G}(\mathcal{T})]{}).\Delta$ where $\xrightarrow[\mathcal{G}(\mathcal{T})]{} = \bigcup_{a\in\Delta} \xrightarrow[\mathcal{G}(\mathcal{T})]{a}$:

$$\forall r_1, r_2 \in R,\ \forall a \in \Delta,\ r_1 \xrightarrow[\mathcal{G}(\mathcal{T})]{a} r_2 \implies \big(\forall r_3 \in R,\ r_1 \xrightarrow[\mathcal{G}(\mathcal{T})]{a} r_3 \Rightarrow r_2 = r_3\big)\ .$$

Consequently, as confluence is undecidable in general, the first-order theory of $\mathcal{G}(\mathcal{T})$ is not decidable.

5. Connections with the Class of Graphs of Turing Machines

In this section, we provide a construction that takes a rooted Thue specification as an input and produces a Turing machine, the graph of which is ε-equivalent to that of the specification. This construction is illustrated by a concrete example.

5.1. Turing Machine Associated to a Thue Specification

Let $\mathcal{T} = (\Omega, \Delta, \mathcal{S}, R, u)$ be a rooted Thue specification. We construct a Turing machine $\mathcal{M}$ that "simulates" $\mathcal{T}$, more precisely the graph of which is ε-equivalent to that of $\mathcal{T}$.

Machine $\mathcal{M}$ consists of two work tapes that we call B_1 and B_2. The first operations that are performed by $\mathcal{M}$ from its initial configuration (q_0, q_0) consist in writing the word u on B_1 and B_2 and to switch to configuration (uq'_0, uq'_0).

We could construct $\mathcal{M}$ in such a way that it then reads a character on its input tape: if it reads a, it could then write this letter at the end of the inscription u of B_1 and reduce the word ua using rules from $\mathcal{S}$. Nevertheless, such a conception encounters a problem if each rewriting of ua produces a word that is not in R. Effectively, in such a case, in the graph of $\mathcal{T}$ there is no outgoing edge from u that is labeled by a. But, as $\mathcal{M}$ has read a on its input tape, in its graph there is an outgoing edge from its initial configuration that is labeled by a. Consequently, graphs $\mathcal{G}(\mathcal{M})$ and $\mathcal{G}(\mathcal{T})$ are not ε-equivalent in this case.

The following note provides a way to bypass this problem. When a word wa (with $w \in R$ and $a \in \Delta$) is being rewritten, the first rule that is used is applied to a suffix la of wa (because w is irreducible by $\mathcal{S}$). Consequently the first rule that is used has the form $la \to r$. Moreover, if $w = w'l$, applying this rule to wa is equivalent to changing $w'l$ into $w'r$.

Consequently, we construct $\mathcal{M}$ in such a way that its behaviour implements the algorithm of Fig. 1. It is important to note that $\mathcal{M}$ reads a letter a on its input tape only in two precise cases (let w be the inscription of B_1 at the beginning of procedure Simulate):

1. when wa is in R, *i.e.* when there is an edge labeled by a from w to wa in $\mathcal{G}(\mathcal{T})$;
2. when a rewriting of wa, the first rule of which is $la \to r \in \mathcal{S}$, has produced a word $w' \in R$, *i.e.* when there is an edge labeled by a from w to w' in $\mathcal{G}(\mathcal{T})$.

In case the word being rewritten has no normal form with respect to $\mathcal{S}$, the loop **while** ... **do** of procedure Simulate does not terminate. Consequently, we construct $\mathcal{M}$ in such a way that it implements the loop **for each** $a \in \Delta$ **do** by an appropriate set of non-deterministic transitions.

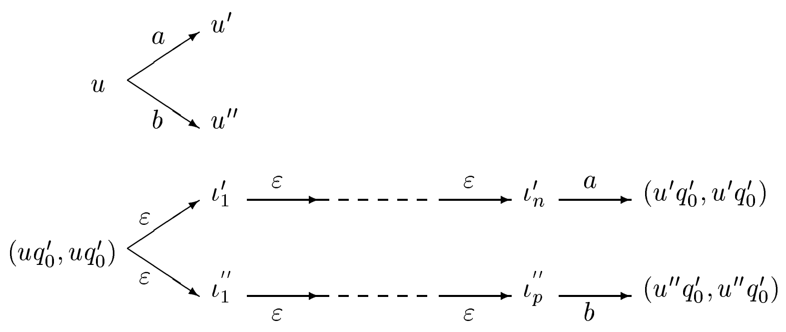
Nevertheless, the graph of $\mathcal{M}$ is not ε-equivalent to that of $\mathcal{T}$ yet. Effectively, consider the example below. We have represented, at the top line, the graph of a Thue specification and, at the bottom line, a part of the graph of the Turing machine that "simulates" the specification. Those graphs are not ε-equivalent because the only possible ε-equivalence relation is such that $u \leftrightsquigarrow (uq'_0, uq'_0)$, $u' \leftrightsquigarrow (u'q'_0, u'q'_0)$, $u'' \leftrightsquigarrow (u''q'_0, u''q'_0)$, for each $i \in [n]$, $\iota'_i \leftrightsquigarrow u$ and for each $i \in [p]$, $\iota''_i \leftrightsquigarrow u$. But $u \xrightarrow{b} u''$, $u \leftrightsquigarrow \iota'_1$ and there is no vertex ι such that $u'' \leftrightsquigarrow \iota$ and $\iota'_1 \overset{b}{\dashrightarrow} \iota$.

procedure $\mathcal{M}$
  switch to configuration (uq_0', uq_0');
  Simulate;

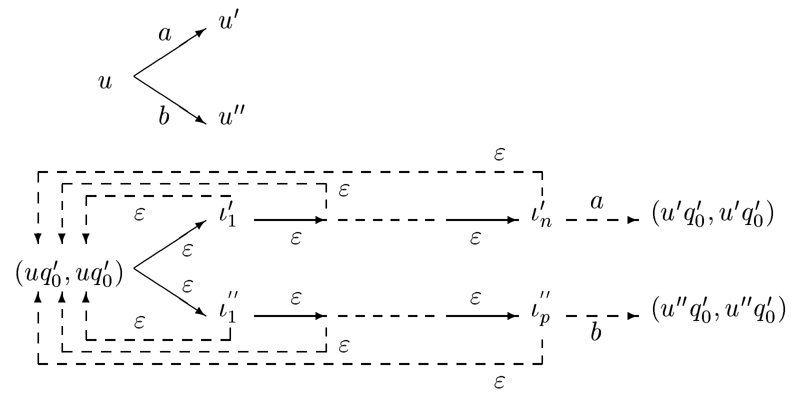
procedure Simulate
  $w :=$ inscription(B_1);
  for each $a \in \Delta$ **do**
    if $wa \in R$ **then** End$[a, wa]$;
    else
      for each $la \to r \in \mathcal{S}$ such that $w = w'l$ **do**
        change inscription $w'l$ of B_1 into $w'r$;
        while inscription(B_1) $\notin R$ **do**
          pick up a rule $l' \to r'$ of $\mathcal{S}$;
          apply $l' \to r'$ to the inscription of B_1;
        End$[a, \text{inscription}(\text{B}_1)]$;

procedure End$[a \in \Delta,\ w \in R]$
  read a on the input tape;
  switch to configuration (wq_0', wq_0');
  Simulate;

Figure 1. Working of $\mathcal{M}$



In order to face up this problem, $\mathcal{M}$ is constructed in such a way that at each step of any reduction, it can go back to the configuration it had before it started to reduce. This is done by means of the second work tape that stores the word that is on B_1 before the reduction. Note that if the reduction of a word wa ends with $w' \in R$, $\mathcal{M}$ has to write w' in the place of w on B_2. Consequently, the previous example turns into the following:



where transitions starting from ι'_{n+1} (resp. from ι''_{p+1}) correspond to the copying of u' (resp. u'') on B_2. It is easy to see that those graphs are ε-equivalent.

Finally, each time $\mathcal{M}$ rewrites a word $w_1 l w_2$ using a rule $l \to r$, it writes r in the place of l. If $l = l_1 l_2$, with $|l_1| = |r|$ and $|l_2| \neq \varepsilon$, it writes r in the place of l_1, then it erases l_2 and shifts to the left the portion of the work tape that corresponds to w_2. If $r = r_1 r_2$, with $|r_1| = |l|$ and $|r_2| \neq \varepsilon$, $\mathcal{M}$ writes r_1 on l then it shifts to the right the portion of B_1 corresponding to w_2 and it writes r_2 at the end of r_1. After performing these operations, the machine verifies if the word that is produced by the rewriting is in R.

Here is a more formal definition of $\mathcal{M}$. Let $\mathcal{A} = (Q_{\mathcal{A}}, \Omega, \delta_{\mathcal{A}}, p_0, F)$ be a finite, deterministic and complete automaton that recognizes R. Suppose that $\mathcal{A}$ has no ε-transition. We set

$$\mathcal{M} = (Q_{\mathcal{M}}, \Delta, \Omega \cup \Delta \cup \{\square\}, \Omega \cup \{\square\}, \delta_{\mathcal{M}}, q_0) .$$

The set of states $Q_{\mathcal{M}}$ is defined as

$$\begin{aligned} Q_{\mathcal{M}} = \;& \{q_0, q'_0\} \cup Q_u \cup Q_{\text{back}} \cup Q_{\text{replace}} \cup Q_{\text{shift}} \cup \\ & \big\{ [a], [a, \in] \bigm| a \in \Delta \big\} \cup \big\{ [a, p] \bigm| a \in \Delta,\ p \in Q_{\mathcal{A}} \big\} \cup \\ & \big\{ [\overline{a}, \in] \bigm| a \in \Omega \cap \Delta \big\} \cup \big\{ [[\overline{a}, p] \bigm| a \in \Omega \cap \Delta,\ p \in Q_{\mathcal{A}} \big\} \cup \\ & \big\{ [a, l \to r \mid w] \bigm| a \in \Delta,\ l \to r \in \mathcal{S},\ w \in \mathrm{Suff}^{+}(l) \big\} \cup \\ & \big\{ [a \mid w, h] \bigm| a \in \Delta,\ \exists l \to r \in \mathcal{S},\ w \in \mathrm{Suff}(l) \text{ and } h \in \mathrm{Suff}(r) \big\} \end{aligned}$$

where Q_u, Q_{back}, Q_{replace} and Q_{shift} are the set of states involved in δ_u, δ_{back}, δ_{replace} and δ_{shift} (see below).

The set of transitions is

$$\delta_{\mathcal{M}} = \delta_u \cup \delta_{\text{back}} \cup \delta_{\text{replace}} \cup \delta_{\text{shift}} \cup \delta_{\text{start}} \cup \delta_{\text{check-}wa} \cup \delta_{\text{rewrite}} \cup \delta_{\text{check}} \cup \delta_{\text{look-for}}$$

where δ_u, δ_{back}, δ_{replace} and δ_{shift} are the sets of ε-transitions that perform the operations described above: δ_u is the of ε-transitions that write u on B_1 and B_2 (when $\mathcal{M}$ has written u, it switches to configuration (uq'_0, uq'_0)), δ_{back} is the set of ε-transitions that allow to $\mathcal{M}$ to switch back to the configuration it had before a reduction, δ_{replace} is the set of ε-transitions that write on B_2 the inscription of B_1 when this inscription is the result of a rewriting and is in R and δ_{shift} is the set of ε-transitions that shift a portion of the inscription of B_1 when a rule is applied.

Without loss of generality, we may assume that δ_u and δ_{replace} are deterministic, *i.e.* for each configuration $(\mu_1 q \nu_1, \mu_2 q \nu_2)$ such that $q \in Q_u$ (resp. $q \in Q_{\text{replace}}$), if there exists in δ a transition of the form

$$(q, \varepsilon, X_1, X_2, Y_1, \blacklozenge_1, Y_2, \blacklozenge_2, p) \text{ where}$$

- X_1 is the first character of ν_1 if this word is not empty, else $X_1 = \Box$ and
- X_2 is the first character of ν_2 if this word is not empty, else $X_2 = \Box$,

then each transition of δ that has the form $(q', \varepsilon, X'_1, X'_2, Y'_1, \blacklozenge'_1, Y'_2, \blacklozenge'_2, p')$ with $q' \in Q_u$ (resp. $q' \in Q_{\text{replace}}$) is such that:

$$(Y'_1, \blacklozenge'_1, Y'_2, \blacklozenge'_2, p') \neq (Y_1, \blacklozenge_1, Y_2, \blacklozenge_2, p) \Rightarrow \ (q', X'_1, X'_2) \neq (q, X_1, X_2).$$

The sets of transitions δ_{start}, $\delta_{\text{check-}wa}$, δ_{rewrite}, δ_{check} and $\delta_{\text{look-for}}$ are defined as follows. The transitions of δ_{start} launch the first loop **for each** … **do** of procedure Simulate of algorithm of Fig. 1:

$$\begin{aligned}\delta_{\text{start}} \quad = \quad & \left\{ \left(q'_0, \varepsilon, \Box, \Box, \Box, \blacktriangleleft, \Box, \blacksquare, \left[\overline{a}, \in\right] \right) \;\middle|\; a \in \Omega \cap \Delta \right\} \bigcup \\ & \left\{ \left(q'_0, \varepsilon, \Box, \Box, \Box, \blacktriangleleft, \Box, \blacksquare, \left[a, q'_0\right] \right) \;\middle|\; a \in \Delta,\ a \notin \Omega \right\}.\end{aligned}$$

Let w be the inscription of B_1. It is clear that if an element a of Δ is not in Ω, wa cannot belong to R because $R \subseteq \Omega^*$. In this case, machine $\mathcal{M}$ directly tries to rewrite wa from state $[a, q'_0]$. Else, from state $\left[\overline{a}, \in\right]$, the machine verifies if $wa \in R$ by means of transitions of the set $\delta_{\text{check-}wa}$ defined as:

$$\begin{aligned}
&\delta_{\text{check-}wa} = \\
&\quad \left\{ \left(\left[\overline{a}, \in\right], \varepsilon, x, \Box, x, \blacktriangleleft, \Box, \blacksquare, \left[\overline{a}, \in\right] \right) \;\middle|\; a \in \Omega \cap \Delta,\ x \in \Omega \right\} \bigcup \\
&\quad \left\{ \left(\left[\overline{a}, \in\right], \varepsilon, \Box, \Box, \Box, \blacktriangleright, \Box, \blacksquare, \left[\overline{a}, p_0\right] \right) \;\middle|\; a \in \Omega \cap \Delta \right\} \bigcup \\
&\quad \left\{ \left(\left[\overline{a}, p\right], \varepsilon, x, \Box, x, \blacktriangleright, \Box, \blacksquare, \left[\overline{a}, p'\right] \right) \;\middle|\; a \in \Omega \cap \Delta,\ x \in \Omega,\ p, p' \in Q_{\mathcal{A}},\ (p, x, p') \in \delta_{\mathcal{A}} \right\} \bigcup \\
&\quad \left\{ \left(\left[\overline{a}, p\right], a, \Box, \Box, a, \blacktriangleright, a, \blacktriangleright, q'_0 \right) \;\middle|\; a \in \Omega \cap \Delta,\ p \in Q_{\mathcal{A}},\ \exists f \in F \text{ such that } (p, a, f) \in \delta_{\mathcal{A}} \right\} \bigcup \\
&\quad \left\{ \left(\left[\overline{a}, p\right], \varepsilon, \Box, \Box, \Box, \blacktriangleleft, \Box, \blacksquare, \left[a, q'_0\right] \right) \;\middle|\; \begin{array}{l} a \in \Omega \cap \Delta,\ p \in Q_{\mathcal{A}},\ \forall p' \in Q_{\mathcal{A}}, \\ (p, a, p') \in \delta_{\mathcal{A}} \Rightarrow p' \notin F \end{array} \right\}.
\end{aligned}$$

For any given a, if wa is an element of R, machine $\mathcal{M}$ is allowed to read a on its input tape, to write a at the end of the inscription of its work tapes and to start again from q'_0. Else (*i.e.* if wa

is reducible or if $wa \in \mathrm{Irr}(\mathcal{S}) \setminus R$), $\mathcal{M}$ switches to state $[a, q_0']$ because we have supposed that automaton $\mathcal{A}$ is deterministic and complete. From this state, $\mathcal{M}$ tries to rewrite wa by means of the rules of $\mathcal{S}$. This is performed by the transitions of δ_{rewrite}:

$$\delta_{\text{rewrite}} =$$

$$\Big\{\Big([a, q_0'], \varepsilon, x, \square, x, \blacktriangleright, \square, \blacksquare, [a \mid \varepsilon, r]\Big) \;\Big|\; a \to r \in \mathcal{S},\ a \in \Delta,\ x \in \Omega \cup \{\square\}\Big\} \cup$$

$$\Big\{\Big([a, q_0'], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, [a, l \to r \mid xa]\Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ l \to r \in \mathcal{S}, \\ x \in \Omega,\ xa \in \mathrm{Suff}(l) \end{array}\Big\} \cup$$

$$\Big\{\Big([a, l \to r \mid g], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, [a, l \to r \mid xg]\Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ l \to r \in \mathcal{S},\ x \in \Omega \cup \Delta, \\ g \in \mathrm{Suff}^+(l),\ xg \in \mathrm{Suff}(l) \end{array}\Big\} \cup$$

$$\Big\{\Big([a, l \to r \mid l], \varepsilon, z, \square, z, \blacktriangleright, \square, \blacksquare, [a \mid l, r]\Big) \;\Big|\; a \in \Delta,\ l \to r \in \mathcal{S},\ z \in \Omega \cup \Delta \cup \{\square\}\Big\} \cup$$

$$\left\{\Big([a \mid g, h], \varepsilon, x, \square, x', \blacktriangleright, \square, \blacksquare, [a \mid g', h']\Big) \;\middle|\; \begin{array}{l} a \in \Delta,\ x, x' \in \Omega \cup \Delta,\ \exists l \to r \in \mathcal{S}, \\ g \in \mathrm{Suff}(l),\ h \in \mathrm{Suff}(r), \\ g = xg',\ h = x'h' \end{array}\right\} \cup$$

$$\Big\{\Big([a \mid g, \varepsilon], \varepsilon, x, \square, x, \blacksquare, \square, \blacksquare, [a, \blacktriangleleft \mid g]\Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ \exists l \to r \in \mathcal{S},\ g \in \mathrm{Suff}^+(l), \\ x = g(1) \end{array}\Big\} \cup$$

$$\Big\{\Big([a \mid \varepsilon, h], \varepsilon, z, \square, z, \blacksquare, \square, \blacksquare, [a, \blacktriangleright \mid h]\Big) \;\Big|\; \begin{array}{l} a \in \Delta,\ z \in \Omega \cup \Delta \cup \{\square\}, \\ \exists l \to r \in \mathcal{S},\ h \in \mathrm{Suff}^+(r) \end{array}\Big\} \cup$$

$$\Big\{\Big([a \mid \varepsilon, \varepsilon], \varepsilon, z, \square, z, \blacktriangleleft, \square, \blacksquare, [a, \in\,]\Big) \;\Big|\; a \in \Delta,\ z \in \Omega \cup \Delta \cup \{\square\}\Big\} \quad .$$

The first four sets of this union contain the transitions that perform the search of a left-hand side of a rule. The other ones contain the transitions that perform the application of a rule. States $[a, \blacktriangleleft \mid g]$ and $[a, \blacktriangleright \mid g]$ launch the transitions of the set δ_{shift} that perform the shifting of a portion of the inscription of B_1 during the application of a rule. States $[a, \blacktriangleleft \mid g]$ are used in case of a shifting to the left and states $[a, \blacktriangleright \mid g]$ in case of a shifting to the right. Suppose that after any shifting from a state $[a, \blacktriangleleft \mid g]$ or $[a, \blacktriangleright \mid g]$, $\mathcal{M}$ switches to state $[a, \in\,]$.

Each time $\mathcal{M}$ has applied a rule, it has to verify if the word that is obtained is in R. This is done by means of the transitions of δ_{check}:

$$\delta_{\text{check}} = \Big\{\Big([a, \in\,], \varepsilon, x, \square, x, \blacktriangleleft, \square, \blacksquare, [a, \in\,]\Big) \;\Big|\; a \in \Delta,\ x \in \Omega\Big\} \cup$$

$$\Big\{\Big([a, \in\,], \varepsilon, x, \square, x, \blacksquare, \square, \blacksquare, [a]\Big) \;\Big|\; a \in \Delta,\ x \notin \Omega\Big\} \cup$$

$$\Big\{\Big([a, \in\,], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [a, p_0]\Big) \;\Big|\; a \in \Delta\Big\} \cup$$

$$\Big\{\Big(\big[a,p\big],\varepsilon,x,\square,x,\blacktriangleright,\square,\blacksquare,\big[a,p'\big]\Big)\;\Big|\;a\in\Delta,\ x\in\Omega,\ p,p'\in Q_{\mathcal{A}},\ (p,x,p')\in\delta_{\mathcal{A}}\Big\}\ \cup$$

$$\Big\{\Big(\big[a,p\big],a,\square,\square,\square,\blacktriangleleft,\square,\blacktriangleleft,\big[\text{replace}\big]\Big)\;\Big|\;p\in F,\ a\in\Delta\Big\}\ \cup$$

$$\Big\{\Big(\big[a,p\big],\varepsilon,\square,\square,\square,\blacktriangleleft,\square,\blacksquare,\big[a\big]\Big)\;\Big|\;p\in Q_{\mathcal{A}}\smallsetminus F,\ a\in\Delta\Big\}\ .$$

If the word w' that is produced by an application of a rule to the word wa is an element of R, the machine reads a on its input tape and replaces the inscription of B_2 by that of B_1 (*i.e.* w'). In order to perform this replacing, $\mathcal{M}$ switches to state $[\text{replace}]$ that launches the transitions of δ_{replace}. After any replacing, $\mathcal{M}$ switches to configuration $(w'q_0', w'q_0')$. If the word w' is not in R, $\mathcal{M}$ continues to rewrite, so it has to look for the left-hand side of a rule within the inscription of B_1:

$$\begin{aligned}\delta_{\text{look-for}} = \quad & \Big\{\Big(\big[a\big],\varepsilon,x,\square,x,\blacktriangleleft,\square,\blacksquare,\big[a\big]\Big)\;\Big|\;a\in\Delta,\ x\in\Omega\cup\Delta\Big\}\ \cup\\ & \Big\{\Big(\big[a\big],\varepsilon,x,\square,x,\blacktriangleright,\square,\blacksquare,\big[a\big]\Big)\;\Big|\;a\in\Delta,\ x\in\Omega\cup\Delta\Big\}\ \cup\\ & \Big\{\Big(\big[a\big],\varepsilon,x,\square,x,\blacktriangleleft,\square,\blacksquare,\big[a,l\to r\mid x\big]\Big)\;\Big|\;l\to r\in\mathcal{S},\ a\in\Delta,\ x\in\mathrm{Suff}(l)\Big\}\ .\end{aligned}$$

We insist that from any state of a transition in $\delta_{\text{check-}wa}$, δ_{rewrite}, δ_{shift}, δ_{check} and $\delta_{\text{look-for}}$, $\mathcal{M}$ is allowed to perform transitions of δ_{back} that bring it back to configuration (wq_0', wq_0'). This is why the graph of $\mathcal{M}$ is ε-equivalent to that of $\mathcal{T}$, as explained above.

Both lemmas below are direct consequences of the definition of $\mathcal{M}$.

Lemma 5.1. *Each vertex $(\mu q\nu, \mu' q\nu')$ of $\mathcal{G}(\mathcal{M})$ satisfies one of the following points.*

1. *Either $q = q_0'$, $\mu = \mu'$, $\mu \in R$ and $\nu = \nu' = \varepsilon$.*
2. *Either $q \in Q_{\text{replace}}$, $\mu\nu \in R$ and there exists a unique sequence $v_1, \ldots, v_m$ of vertices of $\mathcal{G}(\mathcal{M})$ such that*

$$(\mu q\nu, \mu' q\nu') \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_1 \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} \cdots \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_m \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} (\mu\nu q_0', \mu\nu q_0')$$

   (because δ_{replace} is a deterministic set of transitions). Moreover, for each vertex v of $\mathcal{G}(\mathcal{M})$, we have $(\mu q\nu, \mu' q\nu') \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v$ if and only if

$$(\mu q\nu, \mu' q\nu') \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_1 \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} \cdots \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} v_m \xrightarrow[\mathcal{G}(\mathcal{M})]{\varepsilon} (\mu\nu q_0', \mu\nu q_0') \overset{a}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} v\ .$$

3. *Either $q \in Q_u$. As δ_u is a deterministic set of transitions, this case is similar to the previous one with $(\mu\nu q_0', \mu\nu q_0')$ instead of (uq_0', uq_0').*
4. *Either $q \notin Q_u \cup Q_{\text{replace}}$ and there exists $w \in R$ such that $(wq_0', wq_0') \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} (\mu q\nu, \mu' q\nu')$. Moreover, for each vertex (wq_0', wq_0') of $\mathcal{G}(\mathcal{M})$, we have:*

$$(wq_0', wq_0') \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} (\mu q\nu, \mu' q\nu') \iff (\mu q\nu, \mu' q\nu') \overset{\varepsilon}{\underset{\mathcal{G}(\mathcal{M})}{\dashrightarrow}} (wq_0', wq_0')$$

   (this is due to the transitions of δ_{back}).

Lemma 5.2. *Let $w, w' \in R$ and $a \in \Delta$. Then, we have $(wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{a}{\dashrightarrow}} (w'q'_0, w'q'_0)$ if and only if $w' \in (wa)\downarrow_{\mathcal{S}}$. Consequently, (wq'_0, wq'_0) is a vertex of $\mathcal{G}(\mathcal{M})$ if and only if w is a vertex of $\mathcal{G}(\mathcal{T})$.*

The following result is a consequence of those lemmas.

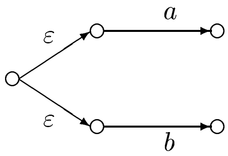
Proposition 5.1. *Graphs $\mathcal{G}(\mathcal{T})$ and $\mathcal{G}(\mathcal{M})$ are ε-equivalent.*

Proof:
It suffices to consider the relation $\leftrightsquigarrow \subseteq R \times \left[(\Omega \cup \Delta)^*.Q.(\Omega \cup \Delta)^* \times \Omega^*.Q.\Omega^*\right]$ defined as

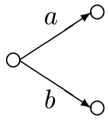
$$\begin{aligned} \leftrightsquigarrow \quad = \quad & \left\{\left(\mu\nu, (\mu q\nu, \mu' q\nu')\right) \;\middle|\; q \in Q_{\text{replace}},\ (\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu\nu q'_0, \mu\nu q'_0)\right\} \cup \\ & \left\{\left(u, (\mu q\nu, \mu' q\nu')\right) \;\middle|\; q \in Q_u,\ (\mu q\nu, \mu' q\nu') \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (uq'_0, uq'_0)\right\} \cup \\ & \left\{\left(w, (\mu q\nu, wq)\right) \;\middle|\; q \notin Q_u \cup Q_{\text{replace}},\ (wq'_0, wq'_0) \underset{\mathcal{G}(\mathcal{M})}{\overset{\varepsilon}{\dashrightarrow}} (\mu q\nu, wq)\right\}. \end{aligned}$$

□

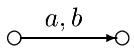
Note that there exist some rooted graphs that are ε-equivalent to no rooted graph with no ε-transition. Consider for example graph $\mathcal{G}$ depicted below:



The only possible candidates for a rooted graph with no ε-transition that would be ε-equivalent to $\mathcal{G}$ have one of the following forms:



or



It is clear that none of these two graphs is ε-equivalent to $\mathcal{G}$. Consequently, as there is no ε-transition in the graph of a Thue specification, there is no Thue specification whose graph is ε-equivalent to $\mathcal{G}$.

In case $\mathcal{T}$ is a linear rooted Thue specification, machine $\mathcal{M}$ constructed above is a linear bounded one because the working of $\mathcal{M}$ consists in the application of the rules of the semi-Thue system $\mathcal{S}$ of $\mathcal{T}$.

The following theorem sums up all of these results.

Theorem 5.1. *Graphs of Turing machines and those of rooted Thue specifications are connected in the following way.*

- *$\mathcal{G}[\text{TS}]$ is ε-equivalent to a strict subset of $\mathcal{G}[\text{TM}]$.*
- *$\mathcal{G}[\text{Lin-TS}]$ if ε-equivalent to a strict subset of $\mathcal{G}[\text{Lin-TM}]$.*

5.2. An Example

Consider the following Thue specification

$$\mathcal{T} = \Big(\{a\},\ \{a,b\},\ \{ab \to \varepsilon, ab \to b\},\ a^*,\ \varepsilon\Big)$$

and the following finite, deterministic and complete automaton with no ε-transition that recognizes a^*:

$$\mathcal{A} = \Big(\{p_0\},\ \{a\},\ \{(p_0,a,p_0)\},\ p_0,\ \{p_0\}\Big)\ .$$

The Turing machine associated with $\mathcal{T}$ is

$$\mathcal{M} = \Big(Q_{\mathcal{M}},\ \{a\},\ \{a,b,\square\},\ \{a,\square\},\ \delta_{\mathcal{M}},\ q_0\Big)$$

where $Q_{\mathcal{M}}$ is the set of states that are involved in $\delta_{\mathcal{M}}$ and

$$\delta_{\mathcal{M}} = \delta_u \cup \delta_{\text{start}} \cup \delta_{\text{check-}wa} \cup \delta_{\text{rewrite}} \cup \delta_{\text{shift}} \cup \delta_{\text{check}} \cup \delta_{\text{replace}} \cup \delta_{\text{look-for}} \cup \delta_{\text{back}}\ .$$

The sets of transitions composing $\delta_{\mathcal{M}}$ are the following.

- δ_u performs the inscription of u, *i.e.* of ε, on B_1 and B_2 and switches the machine to configuration (uq_0', uq_0'):

$$\delta_u = \Big\{\big(q_0, \varepsilon, \square, \square, \square, \blacksquare, \square, \blacksquare, q_0'\big)\Big\}.$$

- δ_{start} is the set:

$$\Big\{\big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, [\overline{a}, \in]\ \big),\ \big(q_0', \varepsilon, \square, \square, \square, \blacktriangleleft, \square, \blacksquare, [b, q_0']\ \big)\Big\}.$$

- $\delta_{\text{check-}wa}$ is the set:

$$\begin{aligned}\Big\{\quad &\big(\ [\overline{a},\in], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [\overline{a},\in]\ \big), \big(\ [\overline{a},\in], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacksquare, [\overline{a}, p_0]\ \big),\\ &\big(\ [\overline{a},p_0], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [\overline{a},p_0]\ \big), \big(\ [\overline{a},p_0], a, \square, \square, a, \blacktriangleright, a, \blacktriangleright, q_0'\big) \quad \Big\}.\end{aligned}$$

- δ_{rewrite} is the set:

$$\Big\{\ \big(\,[b,q'_0],\varepsilon,a,\Box,a,\blacktriangleleft,\Box,\blacksquare,[b,ab\to\varepsilon\mid ab]\,\big),\big(\,[b,q'_0],\varepsilon,a,\Box,a,\blacktriangleleft,\Box,\blacksquare,[b,ab\to b\mid ab]\,\big),$$

$$\big(\,[b,ab\to\varepsilon\mid b],\varepsilon,a,\Box,a,\blacktriangleleft,\Box,\blacksquare,[b,ab\to\varepsilon\mid ab]\,\big),$$

$$\big(\,[b,ab\to b\mid b],\varepsilon,a,\Box,a,\blacktriangleleft,\Box,\blacksquare,[b,ab\to b\mid ab]\,\big),$$

$$\big(\,[b,ab\to\varepsilon\mid ab],\varepsilon,a,\Box,a,\blacktriangleright,\Box,\blacksquare,[b\mid ab,\varepsilon]\,\big),\big(\,[b,ab\to b\mid ab],\varepsilon,a,\Box,a,\blacktriangleright,\Box,\blacksquare,[b\mid ab,b]\,\big),$$

$$\big(\,[b,ab\to\varepsilon\mid ab],\varepsilon,b,\Box,b,\blacktriangleright,\Box,\blacksquare,[b\mid ab,\varepsilon]\,\big),\big(\,[b,ab\to b\mid ab],\varepsilon,b,\Box,b,\blacktriangleright,\Box,\blacksquare,[b\mid ab,b]\,\big),$$

$$\big(\,[b,ab\to\varepsilon\mid ab],\varepsilon,\Box,\Box,\Box,\blacktriangleright,\Box,\blacksquare,[b\mid ab,\varepsilon]\,\big),\big(\,[b,ab\to b\mid ab],\varepsilon,\Box,\Box,\Box,\blacktriangleright,\Box,\blacksquare,[b\mid ab,b]\,\big),$$

$$\big(\,[b\mid ab,\varepsilon],\varepsilon,a,\Box,a,\blacksquare,\Box,\blacksquare,[b,\blacktriangleleft\ \mid ab]\,\big),\big(\,[b\mid ab,b],\varepsilon,a,\Box,b,\blacktriangleright,\Box,\blacksquare,[b\mid b,\varepsilon]\,\big),$$

$$\big(\,[b\mid b,\varepsilon],\varepsilon,b,\Box,b,\blacksquare,\Box,\blacksquare,[b,\blacktriangleleft\ \mid b]\,\big)\quad\Big\}.$$

  Note that we should have considered similar transitions with states of the form $[a\ldots]$ instead of states of the form $[b\ldots]$. Nevertheless, such transitions have no use because letter a is such that $R.a\subseteq R$, *i.e.* for each $w\in R$, wa is irreducible by $\mathcal{S}$ and transitions with states $[a\ldots]$ perform rewritings of wa.

- When $\mathcal{M}$ applies the rule $ab\to\varepsilon$ to an inscription of $\textsc{b}_1$ that has the form w_1abw_2, it changes this word into $w_1\Box\Box w_2$, then it shifts w_2 to the left so that the inscription of $\textsc{b}_1$ is changed into w_1w_2. These operations start from state $[b,\blacktriangleleft\ \mid ab]$ and end at state $[b,\in]$.

  Identically, when $\mathcal{M}$ applies the rule $ab\to b$ to an inscription of $\textsc{b}_1$ that has the form w_1abw_2, it changes this word into w_1bbw_2 (letter a is changed into the left-hand side of the rule), then into $w_1b\Box w_2$. It then shifts to the left the word w_2 so that the inscription of $\textsc{b}_1$ is changed into w_1bw_2. Operation $w_1bbw_2\to w_1b\Box w_2$ and shifting operations start from state $[b,\blacktriangleleft\ \mid b]$ and end at state $[b,\in]$.

  All of these operations are performed by the transitions of δ_{shift}. We do not detail this set here.

- δ_{check} is the set:

$$\Big\{\ \big(\,[b,\in],\varepsilon,a,\Box,a,\blacktriangleleft,\Box,\blacksquare,[b,\in]\,\big),\big(\,[b,\in],\varepsilon,\Box,\Box,\Box,\blacktriangleright,\Box,\blacksquare,[b,p_0]\,\big),$$

$$\big(\,[b,p_0],\varepsilon,a,\Box,a,\blacktriangleright,\Box,\blacksquare,[b,p_0]\,\big),\big(\,[b,p_0],b,\Box,\Box,\Box,\blacktriangleleft,\Box,\blacktriangleleft,[\text{replace}]\,\big),$$

$$\big(\,[b,p_0],\varepsilon,\Box,\Box,\Box,\blacktriangleleft,\Box,\blacksquare,[b]\,\big)\qquad\Big\}.$$

  As explained above, it is sufficient to only consider transitions of δ_{check}, the states of which have the form $[b\ldots]$.

- δ_{replace} performs the copying of the inscription of B_1 on B_2 and is the set:

$$\Big\{\ \big(\,[\text{replace}], \varepsilon, a, a, a, \blacktriangleleft, a, \blacktriangleleft, [\text{replace}]\,\big),$$

$$\big(\,[\text{replace}], \varepsilon, a, \square, a, \blacktriangleleft, a, \blacktriangleleft, [\text{replace}]\,\big),$$

$$\big(\,[\text{replace}], \varepsilon, \square, a, \square, \blacksquare, a, \blacksquare, [\text{erase}]\,\big),\ \big(\,[\text{replace}], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacktriangleright, [\text{go to end}]\,\big),$$

$$\big(\,[\text{erase}], \varepsilon, \square, a, \square, \blacksquare, \square, \blacktriangleleft, [\text{erase}]\,\big),\ \big(\,[\text{erase}], \varepsilon, \square, \square, \square, \blacktriangleright, \square, \blacktriangleright, [\text{go to end}]\,\big),$$

$$\big(\,[\text{go to end}], \varepsilon, a, a, a, \blacktriangleright, a, \blacktriangleright, [\text{go to end}]\,\big),\ \big(\,[\text{go to end}], \varepsilon, \square, \square, \square, \blacksquare, \square, \blacksquare, q_0'\big)\ \Big\}.$$

- $\delta_{\text{look-for}}$ is the set:

$$\Big\{\ \big(\,[b], \varepsilon, a, \square, a, \blacktriangleleft, \square, \blacksquare, [b]\,\big),\ \big(\,[b], \varepsilon, a, \square, a, \blacktriangleright, \square, \blacksquare, [b]\,\big),$$

$$\big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b]\,\big),\ \big(\,[b], \varepsilon, b, \square, b, \blacktriangleright, \square, \blacksquare, [b]\,\big),$$

$$\big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b, ab \to \varepsilon \mid b]\,\big),\ \big(\,[b], \varepsilon, b, \square, b, \blacktriangleleft, \square, \blacksquare, [b, ab \to b \mid b]\,\big)\ \Big\}.$$

  It is sufficient to only consider the transitions of $\delta_{\text{look-for}}$ whose states have the form $[b \ldots]$.
- δ_{back} brings back the machine to the configuration it had before starting to rewrite a word. Each state that is involved in $\delta_{\text{check-}wa}$, δ_{rewrite}, δ_{shift}, δ_{check} and $\delta_{\text{look-for}}$ is the origin of a transition of δ_{back}.

6. Conclusion

We have studied some formal verification-oriented properties of the class of graphs associated with Thue specifications. We have established the closure of this class, up to isomorphism, under synchronized product. Given any specifications $\mathcal{T}_1, \ldots, \mathcal{T}_n$ and a synchronization constraint $\mathcal{C}$, the proof consists in the construction of a Thue specification, written $\prod_{i\in[n]}^{\mathcal{C}} \mathcal{T}_i$, the graph of which is isomorphic to the synchronized product of those of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ with respect to $\mathcal{C}$. It can be proved that the number of rules of $\prod_{i\in[n]}^{\mathcal{C}} \mathcal{T}_i$ depends exponentially of the number n of given Thue specifications, exponentially of the size of the left-hand side of the rules of these specifications and polynomially of the size of their alphabets.

As a consequence of the closure result, and similarly to the Arnold-Nivat approach [AN82, Niv79], Thue specifications provide a uniform framework for the specification of communicating processes and their expressive power seems to be very satisfactory. However, the closure under synchronized product has a counterpart in the undecidability result: we have established that the first-order theory of the graphs of Thue specifications is undecidable. Consequently, it is important to develop automated or assisted proof techniques for Thue specifications. One such method is currently under investigation [Kna00].

We have also considered connections between the class $\mathcal{G}[\textsc{ts}]$ of graphs of rooted Thue specifications and the class $\mathcal{G}[\textsc{tm}]$ of graphs of Turing machines. We have provided a construction that takes a rooted Thue specification as an input and produces a Turing machine whose graph is observationally equivalent to that of the specification (considering ε-transitions as non-observable). We have shown that there cannot be any general construction in the other direction because there exist Turing machines whose graph is observationally equivalent to no graph with no ε-transition.

References

[AN82] André Arnold and Maurice Nivat. Comportements de processus. In *Colloque AFCET "Les mathématiques de l'Informatique"*, pages 35–68, 1982.

[Arn94] André Arnold. *Finite Transition Systems.* Prentice Hall Int., 1994.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[Cau96] Didier Caucal. On infinite transition graphs having a decidable monadic second–order theory. In F. Meyer auf der Heide and B. Monien, editors, *Proc. 23rd Inter. Col. on Automata Languages and Programming, Paderborn, Germany*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205, 1996.

[Cau00] Didier Caucal. On Word Rewriting Systems Having a Rational Derivation. In J. Tiuryn, editor, *Proc. 3rd Inter. Conf. on Foundations of Software Science and Computation Structures, Berlin, Germany*, volume 1784 of *Lecture Notes in Computer Science*, pages 48–62. Springer Verlag, 2000.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using Branching Time Temporal Logic. In *Proc. Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Berlin, 1981. Springer Verlag.

[Cou89] Bruno Courcelle. The monadic second–order logic of graphs, II: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21:187–221, 1989.

[KC99] Teodor Knapik and Hugues Calbrix. Thue Specification and Their Monadic Second–Order Properties. *Fundamenta Informaticae*, 39:305–325, 1999.

[Kna96] Teodor Knapik. Thue specifications. IREMIA, Université de la Réunion. Unpublished draft, 1996.

[Kna00] Teodor Knapik. Checking simple properties of transition systems defined by thue specifications. Technical Report INF/00/01/01/a, IREMIA Université de La Réunion, 2000. http://www.univ-reunion.fr/ knapik/publications/INF-00-01-01-a.ps.gz.

[McM92] Kenneth L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, 1992. Also published by Kluwer, *Symbolic Model Checking*, 1993.

[Mil80] Robin Milner. *Calculus of Communicating Systems*, volume 82 of *Lecture Notes in Computer Science.* Springer Verlag, 1980.

[Mor00] Christophe Morvan. On rational graphs. In J. Tiuryn, editor, *Proc. 3rd Inter. Conf. on Foundations of Software Science and Computation Structures, Berlin, Germany*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer Verlag, 2000.

[MS85] David E. Muller and Paul E. Schupp. The Theory of Ends, Pushdown Automata and Second–order Logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MS97] Alexandru Mateescu and Arto Salomaa. Aspects of Classical Language Theory. In Grzegorz Rozenberg and Arto Salomaa, editors, *Word, Language, Grammar*, volume 1 of *Handbook of Formal Languages*, pages 175–251. Springer Verlag, 1997.

[Niv79] Maurice Nivat. Sur la synchronisation des processus. *Revue technique Thomson-CSF*, 11:899–919, 1979.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, (141):1–35, 1969.