Electronic Notes in Theoretical Computer Science 68 No. 6 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 15 pages

Rewriting in the partial algebra of typed terms modulo AC

Thomas Colcombet

IRISA *Campus universitaire de Beaulieu, 35042 Rennes, France*

Abstract

We study the partial algebra of typed terms with an associative commutative and idempotent operator (typed AC-terms). The originality lies in the representation of the typing policy by a graph which has a decidable monadic theory.

In this paper we show on two examples that some results on AC-terms can be raised to the level of typed AC-terms. The examples are the results on rational languages (in particular their closure by complement) and the property reachability problem for ground rewrite systems (equivalently process rewrite systems).

1 Introduction

Exact verification of programs is known for long to be undecidable. To bypass this limit, a solution is to abstract programs into weaker formal models on which decision procedures are possible. A system is such a formalism: it is defined by a given set (which can be infinite) of *states* and labelled *transitions* between those states describe the possible evolutions of the system.

The first study of a family of infinite systems has been presented by Muller and Schupp [14] on the family of pushdown processes. Each state is a word and the transitions are defined by a finite set of prefix rewriting rules. Pushdown processes model faithfully the control flow and the stack mechanism of programs. Highly complex properties can be automatically checked on such systems (e.g. monadic second order theory [14] and model checking of the μ-calculus). However those systems lack parallelism features. More general systems extend this approach with infinite number of rules. The first family of such systems were the HR-equational graphs introduced by Courcelle[4]. Caucal extended them to prefix recognizable graphs[1]. Those two family of graphs share the same strong decidability properties as pushdown automata.

Independently, studies were pursued on Petri nets. Petri nets can be defined as systems of addition of vectors. Each state is a vector of naturals and each transition adds a fixed vector of integers to the state. Another equivalent definition of petri nets is by rewriting rules applied on a multiset. As what

happens in each dimension of the vector is dissociated from the other components, petri nets possess subtle parallelism facilities. On the other hand the expressiveness over control flow is quiet weak (e.g. it is impossible to encode a stack). The fundamental result over petri nets is the problem of reachability [12,9] (given two states, is it possible to find a path of transitions leading from the first to the second one). However the properties decidable over Petri nets are far more simple than the one decidable for pushdown systems (for instance checking whether the set of states reachable by two Petri nets coincide on some given dimensions is not decidable[7]).

Despite those fundamental differences, pushdown processes and Petri nets have been combined by Mayr into process rewrite systems [13](PRS). PRS are defined by rewrite systems in an algebra of closed terms with an associative and commutative operator (which can be understood as the + in systems of addition of vectors, or as the $\cup$ operator when dealing with the multiset point of view) and an associative operator for sequence (which can be seen as the concatenation of words: the basic operation over stacks). The PRS have a decidable reachability problem as well as Petri nets.

Separately, in the context of formal languages as well as compilation of λ-calculus, pushdown systems were extended to higher order pushdown systems (processes with stacks of stacks of stacks ...). This definition lead to a hierarchy of systems (or graphs) having a decidable monadic second order theory [8,2] as well as pushdown systems. It is natural to try to combine those ¡¡higher order systems¿¿ with Petri nets and thus obtain ¡¡higher order process rewrite systems.¿¿ The present paper is devoted to such a study.

To perform this extension, we slightly transform the original definition of PRS and use the unifying notion of ground rewriting: each state is a closed term and each transition replaces a subterm by another according to a set of rewrite rules (it amounts to replace the sequence operator by the composition of symbols). The main parameter left in this description is the algebra of term itself and the congruence relation allowed. Here, we allow an associative and commutative operator (written +). To obtain an higher order PRS, we restrict the language of branches of the terms by an infinite deterministic top down automaton (called the typing policy). In this work, we show that the reachability and property reachability remain decidable when lifted to the higher order. The proof techniques share many common points with the original study of PRS, however some simplifications are obtained by an extensive use of the notion of rationality (regularity). For this reason, a great part of the paper is devoted to the notion of rationality and culminates with the proof of closure by complement of rational languages in algebras containing an associative and commutative operator.

The remaining of the paper is divided as follows. In Section 2 we study the untyped AC terms and the corresponding rational languages. In Section 3 the corresponding study is performed for typed terms. In Section 4 we study the typed process rewrite systems.

2 Untyped AC terms

2.1 Terms with an AC symbol

In this section, we introduce typed terms with an associative commutative symbol.

From now on, A stands for a finite set of *symbols*. We consider the algebra $\mathcal{T}(A)$ of (finite) terms built with the constant 0, the unary symbols $a \in A$ and the binary operator $+$:

$$t ::= 0 \ \mid a(t) \ \mid t + t \ .$$

The terms are considered modulo associativity and commutativity of the $+$ operator and neutrality of 0 for $+$:

$$t + t' = t' + t, \quad t + (t' + t'') = (t + t') + t'' \quad \text{and} \quad t = 0 + t \ .$$

For any unary symbol $a \in A$ and any term $t \in \mathcal{T}(A)$, we write a instead of $a(0)$ and call such terms *constants*. We say that a term t is *rooted* if it has a symbol of A at its root (and not 0 or $+$). We also use the notation $\Sigma_{i=1}^n t_i$ to denote $t_1 + \ldots + t_n$ and the notation nt for $\Sigma_{i=1}^n t$. With this notations, it is natural to decompose a term t as $\Sigma_{i=1}^n t_i$ where all the t_i are rooted. This decomposition is at the core of most inductive proofs presented in this paper.

The *branches* of a term t are the words in $br(t)$ defined inductively by $br(0) = \{\epsilon\}$, $br(t + t') = br(t) \cup br(t')$ and $br(a(t)) = \{aw \mid w \in br(t)\} \cup \{\epsilon\}$. One can notice that the branches are by definition prefix closed and do not rely on any choice of associativity, commutativity and neutrality of 0 for $+$ (this is why the $\cup\{\epsilon\}$ is present in the definition). The *height* of a term is the greatest length of its branches.

2.2 Rational sets of terms

In this section, we study rational (or equivalently regular) languages of the algebra $\mathcal{T}(A)$. In [3], many equivalent definitions of rational languages of terms in free algebras (without associativity or commutativity) are given: regular grammars, regular expressions and their automata-theoretic counterparts. Those equivalences remain valid when the algebra is not free anymore, and in particular in the case of $\mathcal{T}(A)$. The only result which does not hold anymore in the $\mathcal{T}(A)$ case concerns recognizable languages: recognizable languages are strictly contained in rational ones (see Example 2.1) while both notion are equivalent in free algebras. The main contribution of this work concerning rational languages of $\mathcal{T}(A)$ is their closure by complementation (Corollary 2.4).

A (bottom-up tree) *automaton* is a triple (Q, δ, F) where Q is a finite set of states, δ is a set of transitions of the form $0 \to q$ or $a(q') \to q$ or $q + q' \to q''$

(where q,q',q'' belong to Q and a to A) and F is a subset of Q of *accepting states*.

We define $\xrightarrow{\delta}$ as the smallest relation between states and terms closed by the following deduction rules:

$$\frac{}{0 \xrightarrow{\delta} q} \text{ if } 0 \to q \in \delta \ , \qquad \frac{t \xrightarrow{\delta} q \quad t' \xrightarrow{\delta} q'}{t + t' \xrightarrow{\delta} q''} \text{ if } q + q' \to q'' \in \delta \ ,$$

$$\frac{t \xrightarrow{\delta} q}{a(t) \xrightarrow{\delta} q'} \text{ if } a(q) \to q' \in \delta \ .$$

A term t of $\mathcal{T}(A)$ is *accepted* by the automaton if there exists an accepting state $q \in F$ such that $t \xrightarrow{\delta} q$. A subset of $\mathcal{T}(A)$ is *rational* if it is the language of terms accepted by an automaton.

The main goal of this part is to prove the closure of rational languages of $\mathcal{T}(A)$ by complement (and consequently by all boolean operations). Following the classical approach, the proof gives a deterministic and complete representation to rational languages: an automaton is deterministic and complete if the set of rules δ is a mapping (in this case, from $\{0\} \cup \{a(q) \mid q \in Q\} \cup \{q + q' \mid q, q' \in Q\}$ into Q).

However, determinism defined in this way strictly diminishes the expressiveness of automata (the languages of terms accepted is lowered from rationality to recognizability). The following example illustrates this fact.

Example 2.1 Let us suppose that there is a deterministic and complete automaton $\mathcal{A} = (Q, \delta, F)$ accepting the language of terms $L = \{na+nb \mid n \in \mathbb{N}\}$. Let us write $\overline{\delta}(t)$ the mapping which associates to a term t the only state q such that $t \xrightarrow{\delta} q$. As there is a finite number of states, there exists two distinct naturals n and n' such that $\overline{\delta}(na) = \overline{\delta}(n'a)$. By consequence $\overline{\delta}(na + nb) = \overline{\delta}(n'a + nb)$. Let q be this state. As $na + nb \in L$, $q \in F$, but similarly, as $n'a + nb \notin L$, $q \notin F$. Thus, L cannot be accepted by a deterministic and complete automaton, though it is obviously rational.

This problem is inherited from the associativity and commutativity of $+$ and arise already in commutative monoids: recognizable languages are strictly included in rational ones. Even though, rational languages of finitely generated commutative monoids are closed by all boolean operations as shown by Ginsburg and Spanier[15]. The idea of the present proof of closure by complement of rational languages of $\mathcal{T}(A)$ is to use a proper definition of automata including the notion of rationality over commutative monoids: multiset automata. Multiset automata are automata with multiset as states (thus having an infinite number of states and an infinite number of transitions). Multiset automata can be seen as an extension of automata with arithmetic constraints working on flat trees [10]: flat trees correspond to the algebra $\mathcal{T}(A)$ (extended with non unary symbols), however automata with arithmetic constraints are

intrinsically restricted to the power of recognizability.

Given a finite set Q, we consider $\mathcal{M}(Q)$ the set of multisets with elements in Q. The union operation is simply written $+$ and the neutral element is denoted 0. Elements of $\mathcal{M}(Q)$ can naturally be written $\Sigma_{i=1}^n q_i$ for $q_i \in Q$. The rational languages of $\mathcal{M}(Q)$ have the property to be closed by all the boolean operations (and in particular complementation [15]).

A *multiset automaton* is a triple $\mathcal{A} = (Q, (R_{q,a})_{q\in Q, a\in A}, F)$ where Q is a finite set of *state constants*, for any state constant q and any symbol $a \in A$, $R_{q,a}$ is a rational language of $\mathcal{M}(Q)$, and F is a rational language of $\mathcal{M}(Q)$ of *accepting states*. We call *state* of the automaton the elements of $\mathcal{M}(Q)$.

We define the relation $\xrightarrow{\mathcal{A}}$ between a state u in $\mathcal{M}(Q)$ and a term t in $\mathcal{T}(A)$ by the following inference rules:

$$\frac{t_1 \xrightarrow{\mathcal{A}} q_1 \quad \dots \quad t_n \xrightarrow{\mathcal{A}} q_n}{\Sigma_{i=1}^n t_i \xrightarrow{\mathcal{A}} \Sigma_{i=1}^n q_i} \ , \tag{1}$$

$$\frac{u \in R_{q,a} \qquad t \xrightarrow{\mathcal{A}} u}{a(t) \xrightarrow{\mathcal{A}} u} \ . \tag{2}$$

The multiset automaton accepts a term t if $t \xrightarrow{\mathcal{A}} u$ for some $u \in F$. We write as usual $L_{\mathcal{A}}$ the set of terms accepted by the automaton. It is important to notice that for $t \xrightarrow{\mathcal{A}} u$ then t is a rooted term iff u is a singleton. More generally, if $\Sigma_{i=1}^k t_i \xrightarrow{\mathcal{A}} \Sigma_{i=1}^{k'} q_i$ for t_i rooted terms, then $k = k'$ and, up to permutation of the indices, $t_i \xrightarrow{\mathcal{A}} q_i$ for all i.

The following lemma justifies the use of those automata.

Lemma 2.2 *A subset of $\mathcal{T}(A)$ is rational iff it is accepted by a multiset automaton.*

We say that a multiset automaton is *deterministic* if for a fixed, the sets $R_{q,a}$ are disjoint. The direct consequence of determinism is that for all term $t \in \mathcal{T}(A)$ there is at most one state $u \in \mathcal{M}(Q)$ such that $t \xrightarrow{\mathcal{A}} u$. A multiset automaton is *complete* if for all a, $\cup_{q\in Q} R_{q,a} = \mathcal{M}(Q)$. The direct consequence of completeness is that for any $t \in \mathcal{T}(A)$, there is some $u \in \mathcal{M}(Q)$ such that $t \xrightarrow{\mathcal{A}} u$. Under both constraints, $\xrightarrow{\mathcal{A}}$ is a mapping from $\mathcal{T}(A)$ to $\mathcal{M}(Q)$.

Lemma 2.3 *Every rational language in $\mathcal{T}(A)$ is accepted by a deterministic and complete multiset automaton.*

Proof Let (Q, R, F) be a multiset automaton. We construct a deterministic and complete multiset automaton (Q', R', F') which accepts exactly the same terms.

As for determinizing finite automata, we set $Q' = 2^Q$.

For R a subset of $\mathcal{M}(Q)$, we define $R\uparrow \subseteq \mathcal{M}(Q')$ by

$$R\uparrow = \{(\Sigma_{i=1}^n s_i) \in \mathcal{M}(Q') \mid \exists (\Sigma_{i=1}^n q_i) \in R, \quad \forall i, q_i \in s_i\} \ .$$

Notice that if R is rational then $R\uparrow$ also is.

For $s \in Q'$ a state constant and $a \in A$ a symbol, we set the corresponding transition by:

$$R'_{s,a} = \{v \in \mathcal{M}(Q') \mid \forall\ q \in Q, \quad q \in s \Leftrightarrow v \in R_{q,a}\uparrow\} \ . \tag{3}$$

The set of accepting states is $F' = F\uparrow$.

Validity: To make this construction valid, we need to ensure that the subsets used are rational. If the set R is rational then $R\uparrow$ also is [1] . Hence, F' is rational. Using the equality (equivalent to the definition)

$$R'_{s,a} = \bigcap_{q \in s} R_{q,a}\uparrow \quad - \quad \bigcup_{q \in Q-s} R_{q,a}\uparrow \ ,$$

it follows that $R'_{s,a}$ is rational.

Determinism of $\mathcal{A}'$: Let v be in $R'_{s,a}$ and $R'_{s',a}$, then for all $q \in Q$, $q \in s$ iff $v \in R_{q,a}\uparrow$ (definition of $R'_{s,a}$) and $q \in s'$ iff $v \in R_{q,a}\uparrow$ (definition of $R'_{s',a}$). Thus, $s = s'$.

Completeness of $\mathcal{A}'$: Let v be a state in $\mathcal{M}(Q')$ and a a symbol. Let s be $\{q \in Q \mid v \in R_{q,a}\uparrow\}$, then $v \in R_{s,a}$ (definition of $R_{s,a}$).

Correctness: We prove the two following properties simultaneously by induction on the height h of t:

- for s a constant state of Q' and t a rooted term, then

$$t \xrightarrow{\mathcal{A}'} s \quad \Leftrightarrow \quad s = \{q \in Q \mid t \xrightarrow{\mathcal{A}} q\} \ , \tag{4}$$

- for a state v in $\mathcal{M}(Q')$ and a term t such that $t \xrightarrow{\mathcal{A}'} v$,

$$\forall R \subseteq \mathcal{M}(Q), \quad v \in R\uparrow \Leftrightarrow \ (\exists u \in R, \ t \xrightarrow{\mathcal{A}} u) \ . \tag{5}$$

For $h = 0$, there is no rooted terms: (4) is satisfied. By (1), $0 \xrightarrow{\mathcal{A}} v$ iff $v = 0$. By definition of $\uparrow$, $0 \in R\uparrow$ iff $0 \in R$. Hence (5) is satisfied.

Let $h \geq 1$ be an integer, we suppose that properties (4) and (5) are satisfied by all terms of height $< h$.

(4) Let $a(t)$ be a rooted term of height h. By completeness, there is a $s \in Q'$ such that $a(t) \xrightarrow{\mathcal{A}'} s$. Let v be such that $v \in R'_{s,a}$ and $t \xrightarrow{\delta'} v$. Such a v exists (2). By definition (3), for all q, $q \in s$ iff $v \in R_{q,a}\uparrow$. Applying hypothesis of induction (5), $q \in s$ iff there is some $u \in R_{q,a}$ verifying $t \xrightarrow{\mathcal{A}} q$. Thus by rule (2), $q \in s$ iff $t \xrightarrow{\mathcal{A}} q$.

(5) Let t be a term. It can be written $\Sigma_{i=1}^n t_i$ with the t_i rooted. Let v be such that $t \xrightarrow{\mathcal{A}'} v$. By (1), there is n state constants $s_1,\ldots,s_n$ such that $v = \Sigma_{i=1}^n s_i$

[1] It can be shown using e.g. a formula of the arithmetic of Pressburger.

and for all i, $t_i \xrightarrow{\mathcal{A}'} s_i$. By definition of $\uparrow$, $v \in R\uparrow$ iff there is $\Sigma_{i=1}^n q_i \in R$ with for all i, $q_i \in s_i$. But by (4), $s_i = \{q \mid t_i \xrightarrow{\mathcal{A}} q\}$. Hence, $v \in R\uparrow$ iff there is $\Sigma_{i=1}^n q_i \in R$ with for all i, $t_i \xrightarrow{\mathcal{A}} q_i$, or equivalently by (1), iff there is $u \in R$ with $t \xrightarrow{\mathcal{A}} u$.

Property (5) together with the definition of F' gives $L_{\mathcal{A}'} = L_{\mathcal{A}}$. □

Corollary 2.4 *The rational languages in $\mathcal{T}(A)$ are closed by complement.*

According to (5), it is sufficient to replace the set of accepting states by its complementary in a deterministic and complete multiset automaton accepting a rational language to obtain a multiset automaton accepting the complement of the rational subset.

3 Typed AC terms

3.1 Partial algebra of typed terms with an AC symbol

A *typing policy* is described by a set (which can be infinite) of *types* Θ, and a typing function τ_a from types to types for all symbol $a \in A$. The intended meaning is that the symbol a, when applied to an argument of type $\tau_a(\theta)$ has type θ. We will often refer to the typing policy as a graph: the vertices are the types, and there is an edge between type θ and type θ' labelled by a if $\tau_a(\theta) = \theta'$. Notice that it does not correspond to the arrow notation used in classical function types. As τ is a function, the graph is deterministic.

We say that a term t has type θ for the typing policy (Θ, τ), written $(\Theta, \tau) \vdash t : \theta$ if one can derive the judgment $(\Theta, \tau) \vdash t : \theta$ by the following rules:

$$\frac{\theta \in \Theta}{(\Theta,\tau) \vdash 0 : \theta}, \qquad \frac{(\Theta,\tau) \vdash t : \tau_a(\theta) \quad a \in A}{(\Theta,\tau) \vdash a(t) : \theta}, \qquad \frac{(\Theta,\tau) \vdash t : \theta \quad (\Theta,\tau) \vdash t' : \theta}{(\Theta,\tau) \vdash t + t' : \theta}.$$

The second rule can only be applied if τ_a is defined. For this reason, some terms may not be typable. We call *typed term* the couple of a term t and a type θ, and we write it $t{:}\theta$. The set of typed terms $t : \theta$ such that $(\Theta, \tau) \vdash t : \theta$ is written $\mathcal{T}^{\Theta,\tau}(A)$. We write $\mathcal{T}_\theta^{\Theta,\tau}(A)$ the restriction of $\mathcal{T}^{\Theta,\tau}(A)$ to typed terms of type θ.

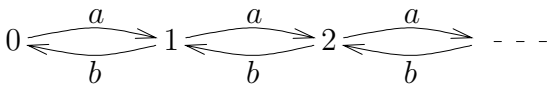
The following property gives a simple graphical interpretation of typing.

Proposition 3.1 *The terms in $\mathcal{T}_\theta^{\Theta,\tau}(A)$ are the terms such that all branches corresponds to a path of origin θ in the graph (Θ, τ)* [2].

Example 1: If $\Theta = \{\theta\}$ and $\tau_a(\theta) = \theta$ for all $a \in A$ then $\mathcal{T}_\theta^{\Theta,\tau}(A) = \mathcal{T}(A)$ (up to type removal). It corresponds to the domain of (untyped) process rewrite systems.

[2] Notice that, as the graph is deterministic, there is no ambiguity about those paths.

Example 2: Let Θ be the natural integers, A be $\{a, b\}$ and τ be such that $\tau_a(n) = n + 1$, $\tau_b(n + 1) = n$. Graphically,



The paths of origin 0 in this graph contain more a's than b's. Reciprocally, to any word over $\{a, b\}$ such that all prefixes contain more a's than b's corresponds a path of origin 0 in the graph. Thus, according to Proposition 3.1, terms of type 0 are such that all the branches contain more a's than b's (recall the branches are prefix closed). This is an example of an infinite typing policy.

From now and on, the typing policy is fixed. Thus, we simplify slightly the notation $(\Theta, \tau) \vdash t : \theta$ by writing $\vdash t : \theta$. Furthermore, by convention, we will suppose that there is a type $\bullet \in \Theta$ of out-degree 0 (no term but 0 has type $\bullet$), and for any newly introduced symbol q, $\tau_q(\theta) = \bullet$ for $\theta \in \Theta - \{\bullet\}$. Thus, all newly introduced symbol behaves like a constant of any type (but $\bullet$).

We need now a way to perform computations on the typing policy, even if there is an infinite number of types. The monadic second order logic presented in the next section serves this purpose.

3.2 Monadic second order logic

In this part, we briefly recall the basis of monadic second order logic (MSO).

The MSO logic expresses properties on graphs. Our purpose is to apply it to the typing policy. We thus adapt slightly the usual notations to fit with this use. A *MSO formula* follows the syntax:

$$\begin{aligned}\Phi ::= {} & \exists \theta, \Phi \quad | \forall \theta,\ \Phi \\ & | \exists X, \Phi \quad | \forall X,\ \Phi \\ & | \tau_a(\theta) = \theta \quad | \theta \in X \\ & | \Phi \wedge \Phi \quad | \Phi \vee \Phi \quad | \neg \Phi \quad | \mathit{true} \quad | \mathit{false}\ .\end{aligned}$$

The *first order variables* (θ,θ',...) range over types while the *monadic second order variables* (written in capital letters X,Y,...) range over sets of types. All the boolean connectives are allowed as well as any quantification over first and second order variables. Atomic predicates allow to test the typing policy ($\theta = \tau_a(\theta')$) and the membership of a first order variable to a second order variable ($\theta \in X$). We allow ourselves to use some more complex notations. For instance we can use variables over known finite domains (e.g the states or the rules of an automaton) and use quantification over them. In any case those extensions can be encoded into standard MSO formulas.

Whenever a typing policy (Θ, τ) satisfies a MSO formula Φ, we say that (Θ, τ) is a *model* of Φ and write $(\Theta, \tau) \models \Phi$. A typing policy is said to have

a *decidable MSO theory* if one can decide, given a MSO formula, whether the typing policy is a model of the formula. The typing policy being fixed, we just write $\models \Phi$ instead of $(\Theta, \tau) \models \Phi$.

As a useful example, we define for any term t the predicate $hastype_t$ such that for any type θ, $\models hastype_t(\theta)$ iff $\vdash t : \theta$. The predicate is built by induction:

$$hastype_{\Sigma_{i=1}^n a_i(t_i)}(\theta) \quad \equiv \quad \forall i \in [1,n],\ \exists \theta',\ \tau_{a_i}(\theta) = \theta' \wedge hastype_{t_i}(\theta')$$

The MSO logic has been extensively studied and many classes of (possibly infinite) graphs are known to have a decidable MSO theory [14,5,1,8,2]. The infinite typing policy of Example 2 belongs to the simplest of those families: pushdown graphs.

3.3 MSO-guarded rational languages of typed terms

In this section we extend the notion of rational languages of $\mathcal{T}(A)$ to MSO-guarded rational languages of $\mathcal{T}^{\Theta,\tau}(A)$. To this purpose, we introduce MSO-guarded automata: automata such that transitions can be applied if and only if a certain MSO formula is satisfied by the type of the terms involved in the transition. Apart from this distinction, the techniques involved in this section are very similar to the ones of Section 2.2.

Formally, a *MSO-guarded automaton* over $\mathcal{T}^{\Theta,\tau}(A)$ is a triple (Q, δ, F) where Q is a finite set of states, δ is a finite set of transitions of the form $0 \to_\Phi q$, $q + q' \to_\Phi q''$ or $a(q) \to_\Phi q'$ with q, q' and q'' states and Φ a MSO formula, and $F \subseteq Q$ is the set of accepting states. The MSO formulas Φ are called *guards*. Each guard Φ has precisely one free variable and we can bind it by using a functional notation: $\Phi(\theta)$. The MSO-guarded automata behave like standard automata, the only difference being that transitions can be applied if and only if the guard is satisfied by the type of the term involved.

We define the relation $\xrightarrow{\delta}$ between typed terms and states as the smallest relation satisfying:

$$\frac{}{0 : \theta \xrightarrow{\delta} q} \text{ if } 0 \to_\Phi q \in \delta \text{ and } \models \Phi(\theta)\ ,$$

$$\frac{t : \tau_a(\theta) \xrightarrow{\delta} q}{a(t) : \theta \xrightarrow{\delta} q'} \text{ if } a(q) \to_\Phi q' \in \delta \text{ and } \models \Phi(\theta)\ ,$$

$$\frac{t : \theta \xrightarrow{\delta} q \quad t' : \theta \xrightarrow{\delta} q'}{t + t' : \theta \xrightarrow{\delta} q''} \text{ if } q + q' \to_\Phi q'' \in \delta \text{ and } \models \Phi(\theta)\ .$$

A typed term $t : \theta$ is *accepted* by the automaton if there is an accepting state $q \in F$ such that $q \xrightarrow{\delta} t : \theta$. We will write $L_\delta(q)$ the set of typed terms $t : \theta$ such that $q \xrightarrow{\delta} t : \theta$. A set of typed term $R \subseteq \mathcal{T}^{\Theta,\tau}(A)$ is MSO-guarded rational if there is a MSO-guarded automaton accepting exactly the typed

terms of R. A term can be present with different types in the same rational language, but, as the type information is always kept along with the term, no confusion arises. On the contrary, the states of the automaton are not typed: it is not necessary. The important point is that the satisfaction of the guard depends only of the type of the term but not at all of the term itself. We will furthermore suppose that the rules enforce the typing of terms: if there is a rule of the form $q \xrightarrow{\delta}_\Phi a(q')$, then for all type θ, $\models \Phi(\theta) \Rightarrow \mathit{hastype}_a(\theta)$. Thus, this rule can be applied only if it is sensible to consider a term of root a.

The following theorem extends naturally the previous one to MSO-guarded rational sets.

Theorem 3.2 *The MSO-guarded rational languages of $\mathcal{T}^{\Theta,\tau}(A)$ are closed by complementation.*

The proof works as in the untyped case. One defines similarly MSO-guarded multiset automata. We first remark that there is only a finite number of possible valuations for $(\Phi_1(\theta), \dots, \Phi_k(\theta))$ for a given type θ (where $\Phi_1, \dots, \Phi_k$ are the guards of an automaton). Given such a valuation v, by combining together the guards using the boolean connectives, it is easy to obtain a new guard Φ_v such that $\Phi_v(\theta)$ is satisfied if and only if $(\Phi_1(\theta), \dots, \Phi_k(\theta)) = v$. It is then sufficient to apply the techniques of the previous demonstration for each of this new guards.

Lemma 3.3 *If R is a MSO-guarded rational language of $\mathcal{T}^{\Theta,\tau}(A)$, then there is a MSO formula empty_R such that:*

$$\models \mathit{empty}_R(\theta) \quad \textit{iff} \quad R \cap \mathcal{T}_\theta^{\Theta,\tau}(A) = \emptyset \ .$$

Proof Let (Q, δ, F) be a MSO-guarded automaton. The principle is to compute the sets of types X_q for $q \in Q$ such that $\theta \in X_q$ if and only if $t : \theta \xrightarrow{\delta} q$ for some t.

$$\begin{aligned}
&\mathit{empty}_R(\theta_0) \equiv \forall (X_q)_{q\in Q},\\
&\quad \forall 0 \to_\Phi q \in \delta, \qquad \forall \theta, \qquad \Phi(\theta) \Rightarrow \theta \in X_q \qquad (a)\\
&\wedge\ \forall q + q' \to_\Phi q'' \in \delta, \quad \forall \theta, \qquad (\Phi(\theta) \wedge \theta \in X_q \wedge \theta \in X_{q'}) \Rightarrow \theta \in X_{q''} \qquad (b)\\
&\wedge\ \forall a(q) \to_\Phi q' \in \delta, \quad \forall \theta, \theta', (\Phi(\theta) \wedge \tau_a(\theta) = \theta' \wedge \theta' \in X_q) \Rightarrow \theta \in X_{q'} \qquad (c)\\
&\qquad \Rightarrow \exists q \in F,\ \theta_0 \in X_q
\end{aligned}$$

The sets (X_q) such that $\theta \in X_q$ iff $t : \theta \xrightarrow{\delta} q$ for some t are the smallest sets solution of constraints (a), (b) and (c). For instance constraint (a) can be read: ¡¡if there is a transition $0 \to_\Phi q$ in the automaton, then, for all type θ such that the guard is satisfied, there is a term of type θ in $L_\delta(q)$.¿¿ As the constraints are continuous (in the meaning of complete partial orders), testing if $\theta_0 \in X_q$ for the smallest solution of (a), (b) and (c) amounts to verify

that $\theta_0 \in X_q$ for all solution of (a), (b) and (c). This is what performs the universal quantification over (X_q). □

Corollary 3.4 *If the typing policy has a decidable monadic theory then the emptyness of R is decidable.*

4 Ground rewrite systems

4.1 Typed process rewrite systems

In this section, we introduce MSO-guarded rational process rewrite systems. It is a natural extension to types of process rewrite systems. Following the scheme of Mayr's proof [13], we then state a normalization lemma.

Definition 4.1 A MSO-guarded rational process rewrite system over $\mathcal{T}^{\Theta,\tau}(A)$ labelled by E is a finite set Δ of rules of the form $R \xrightarrow{e} R'$ where R and R' are MSO-guarded rational languages of $\mathcal{T}^{\Theta,\tau}(A)$ and e is a label in E.

A transition of the process rewrite systems corresponds to the replacement of a subterm by another according to the set of rules: we define inductively between typed terms of same type $t_1 : \theta$ and $t_2 : \theta$ the rewrite judgment $t_1 \overset{e}{\Rightarrow}_\Delta t_2 : \theta$ by

$$\frac{t_1 : \theta \in R_1 \quad t_2 : \theta \in R_2 \quad R_1 \xrightarrow{e} R_2 \in \Delta}{t_1 \overset{e}{\Rightarrow}_\Delta t_2 : \theta},$$

$$\frac{\vdash t : \theta \qquad t_1 \overset{e}{\Rightarrow}_\Delta t_2 : \theta}{t + t_1 \overset{e}{\Rightarrow}_\Delta t + t_2 : \theta}, \qquad \frac{t_1 \overset{e}{\Rightarrow}_\Delta t_2 : \tau_a(\theta)}{a(t_1) \overset{e}{\Rightarrow}_\Delta a(t_2) : \theta}.$$

We also define the reflexive and transitive closure $\overset{*}{\Rightarrow}_\Delta$ of $\overset{e}{\Rightarrow}_\Delta$ by:

$$\frac{\vdash t : \theta}{t \overset{*}{\Rightarrow}_\Delta t : \theta}, \qquad \frac{t \overset{*}{\Rightarrow}_\Delta t' : \theta \quad t' \overset{e}{\Rightarrow}_\Delta t'' : \theta}{t \overset{*}{\Rightarrow}_\Delta t'' : \theta}.$$

When $t \overset{*}{\Rightarrow}_\Delta t' : \theta$, we will say that there is a path of type θ between t and t' for the rules Δ. We will omit to specify the set of rules when there is no ambiguity about it.

Definition 4.2 A MSO-guarded rational process rewrite system Δ is normalized if for all rule $R_1 \xrightarrow{e} R_2 \in \Delta$ there is a MSO formula Φ and two terms t_1, t_2 such that $R_1 = \{t_1 : \theta \mid \ \models \Phi(\theta)\}$, $R_2 = \{t_2 : \theta \mid \ \models \Phi(\theta)\}$ and (t_1, t_2) has one of the following forms:

- sequential rules: $(a, b(c))$ or $(a(b), c)$
- parallel rules: $(a, b + c)$, $(a + b, c)$, $(0, a)$, $(a, 0)$ or (a, b).

In this case, we write Δ^{seq} the set of sequential rules and Δ^{par} the set of parallel rules. We also use notations similar to MSO-guarded automata for rules: $a \xrightarrow{e}_\Phi b(c) \in \Delta$, $a(b) \xrightarrow{e}_\Phi c \in \Delta$, ...

Proposition 4.3 *Given Δ a MSO-guarded rational process rewrite system over $\mathcal{T}^{\Theta,\tau}(A)$, there is a MSO-guarded normalized process rewrite system Δ' over $\mathcal{T}^{\Theta,\tau}(A \cup C)$ (where C is a finite set of new constants of any type) such that for all typed terms $t:\theta, t':\theta \in \mathcal{T}^{\Theta,\tau}(A)$, $t \stackrel{*}{\Rightarrow}_\Delta t':\theta$ iff $t \stackrel{*}{\Rightarrow}_{\Delta'} t':\theta$.*

Proof Let $R_1 \stackrel{e_1}{\to} R'_1$, ..., $R_k \stackrel{e_k}{\to} R'_k$ be the rules in Δ with $R_i = L_{(Q_i,\delta_i,F_i)}$ and $R'_i = L_{(Q'_i,\delta'_i,F'_i)}$. We assume, without loss of generality, that the Q_i's, the Q'_i's and A are all disjoint. Let C be $Q_1 \cup \cdots \cup Q_k \cup Q'_1 \cup \cdots \cup Q'_k$. We define now Δ' by:

$$\begin{aligned}\Delta' = &\{t \stackrel{\$}{\to}_\Phi q \mid \exists i,\ q \stackrel{\delta}{\to}_\Phi t \in \delta_i\}\\ &\cup \{q \stackrel{e}{\to}_{true} q' \mid \exists i,\ q \in F_i,\ e = e_i,\ q' \in F'_i\}\\ &\cup \{q \stackrel{\$}{\to}_\Phi t \mid \exists i,\ q \stackrel{\delta}{\to}_\Phi t \in \delta'_i\}\end{aligned}$$

The normalized process rewrite system mimics the behavior of the automata by using exactly the same rules (labelled by a dummy symbol \$). Obviously, if $t \stackrel{*}{\Rightarrow}_\Delta t':\theta$ then $t \stackrel{*}{\Rightarrow}_{\Delta'} t':\theta$. The other direction is more technical, we do not show it here. □

4.2 Reachability in typed process rewrite systems

In this section we show that (providing that the typing policy has a decidable monadic theory) the reachability problem between constants is decidable for typed process rewrite systems.

Many proofs of reachability in infinite state systems rely on the rationality of the set of reachable states (for instance [6] for ground rewrite systems or [11] for PA processes). Such an approach cannot be used in the case of process rewrite systems (typed or not): the closure properties of rational language would give the decidability of the equivalence of reachable sets for process rewrite systems and this problem is not decidable (it has been proved for the subclass of Petri Nets [7]).

The core of our approach is a direct translation to typed terms of [13]. The idea is to obtain a representation of the (potentially infinite) set:

$$\Lambda_\Delta = \{(a, b, \theta) \mid a \stackrel{*}{\Rightarrow}_\Delta b:\theta\}$$

To this purpose, we use the following lemma:

Lemma 4.4 *The set Λ_Δ is the smallest set satisfying:*

$$\frac{a \stackrel{e}{\to}_\Phi b(c) \in \Delta \quad \models \Phi(\theta) \quad (c, c', \tau_b(\theta)) \in \Lambda_\Delta \quad b(c') \stackrel{e'}{\to}_{\Phi'} a' \in \Delta \quad \models \Phi'(\theta)}{(a, a', \theta) \in \Lambda_\Delta},$$

$$\frac{a \stackrel{*}{\Rightarrow}_{\Delta'} b:\theta \qquad \Delta' = \Delta^{par}_\theta \cup \{c \stackrel{\$}{\to} d : \theta \mid (c, d, \theta) \in \Lambda_\Delta\}}{(a, b, \theta) \in \Lambda_\Delta}.$$

The first rule states that a path of type θ between a and b can start with a rule $a \xrightarrow{e}_\Phi b(c)$ and end with a rule $b(c') \xrightarrow{e'}_{\Phi'} a'$ providing that the guard are satisfied and that there is a path of type $\tau_b(\theta)$ between c and c'. The second rule states that if there is a path of type θ between a and b using the parallel rules and what is already known in Λ_Δ, then there is a path of type θ between a and b. As a consequence, the second rule inserts all the paths of length 0 (i.e. (a, a, θ) with $\vdash a : \theta$). The test $a \overset{*}{\Rightarrow}_{\Delta'} b : \theta$ performed by the second rule is handled by the following lemma:

Lemma 4.5 *If Δ contains only parallel rules then $a \overset{*}{\Rightarrow}_\Delta b : \theta$ is decidable.*

This problem is an instance of the reachability problem for Petri nets for each possible valuation of the guards. It is decidable [12,9].

In Mayr's proof, the set Λ_Δ is finite (there is no type information) and a saturation algorithm is sufficient for computing it. In our case, we can define this set by monadic formulas:

Lemma 4.6 *Given a normalized MSO-guarded process rewrite system Δ over $\mathcal{T}^{\Theta,\tau}(A)$, there is $|A|^2$ monadic formulas $\lambda_{a,b}$ for a, b in A such that:*

$$(a, b, \theta) \in \Lambda_\Delta \quad \textit{iff} \quad \models \lambda_{a,b}(\theta)$$

The set Λ_Δ can be represented by $|A|^2$ second order variables $(X_{a,b})_{a,b\in A}$

$$\text{such that} \quad \theta \in X_{a,b} \quad \text{iff} \quad (a, b, \theta) \in \Lambda_\Delta \ .$$

Lemma 4.4 describes the set Λ_Δ as the smallest set satisfying some MSO-expressible constraints. Thus, a technique similar to the proof of Lemma 3.3 can be used. The test $a \overset{*}{\Rightarrow}_{\Delta'} b : \theta$ with $\Delta' = \Delta_\theta^{par} \cup \{c \xrightarrow{\$} d : \theta \mid (c, d, \theta) \in \Lambda_\Delta\}$ can be performed using Lemma 4.5 and the remark that there is only a finite number of possible parallel rules for a given set of symbol A. Thus all the cases can be treated in a MSO formula of exponential size (one for each of the possible sets of parallel rules).

4.3 Property reachability

In this section we show that (providing that the typing policy has a decidable monadic theory), the reachable property problem is decidable (Theorem 4.8).

Given a MSO-guarded rational process rewrite system Δ with labels in E, a property has the following syntax:

$$\phi \quad ::= \quad e \ \mid \neg\phi \ \mid \phi \vee \phi \ \mid \phi \wedge \phi \ ,$$

with $e \in E$. An atomic property e is satisfied by a typed term $t : \theta$ if there is some $t' : \theta$ such that $t \overset{e}{\Rightarrow}_\Delta t' : \theta$. The boolean connectives have their standard semantic. An instance of the reachability property problem is: ¡¡given a MSO-guarded rational process rewrite system Δ, a typed term $t : \theta$ and a property ϕ,

is it possible to reach a typed term satisfying ϕ from $t:\theta$ by using the rewriting rules in Δ?¿¿.

To show this result, it is sufficient to prove that the set of typed terms satisfied by a property is MSO-guarded rational. This can be reduced to atomic properties according to the closure by boolean connectives of MSO-guarded rational languages.

Lemma 4.7 *Given a label $e \in E$, the set of typed terms $t:\theta$ such that $t \stackrel{e}{\Rightarrow}_\Delta t':\theta$ for some term t' is MSO-guarded rational.*

Proof Thanks to the closure of MSO-guarded rational languages by union, it is sufficient to show the result for only one rule labelled by e. Let $R_1 \stackrel{e}{\to} R_2$ be such a rule. One can apply this rule on a typed term $t:\theta$ if and only if it contains a subterm $t':\theta'$ in R_1 and there is a term of type θ' in R_2. Let $\mathcal{A} = (Q, \delta, F)$ be the automaton accepting R_1. We construct the automaton (Q', δ', F') accepting the language of typed terms such that $t \stackrel{e}{\Rightarrow}_\Delta t':\theta$ for some term t' by (ϵ-transitions are used here only for convenience):

$$
\begin{aligned}
Q' &= Q \cup \{q_0, q_1\} \\
\delta' &= \delta \\
&\quad \cup \{q \to_\phi q_1 \mid q \in F,\ \Phi(\theta) = \neg \mathit{empty}_{R_2}(\theta)\} \\
&\quad \cup \{q_0 + q_1 \to_{\mathit{true}} q_1\} \\
&\quad \cup \{a(q_0) \to_{\mathit{true}} q_0 \mid a \in A\} \cup \{0 \to_{\mathit{true}} q_0,\ q_0 + q_0 \to_{\mathit{true}} q_0\} \\
F' &= \{q_1\}
\end{aligned}
$$

The states in Q perform the same computation as in $\mathcal{A}$. The state q_0 accepts any term of $\mathcal{T}^{\Theta,\tau}(A)$. Notice the use of the *empty* predicate for checking that there is a term of type θ in R_2. □

Theorem 4.8 *If the typing policy has a decidable monadic theory then the reachable property problem of typed process rewrite systems is decidable.*

Proof Given a tree t and a type θ, then the set $R_1 = \{t:\theta\}$ is MSO-guarded rational. According to Lemma 4.7, the set R_2 of typed terms satisfying the property is MSO-guarded rational. We add to the system the new symbols a_1 and a_2 and the transitions $a_1 \to_{\mathit{true}} R_1$ and $R_2 \to_{\mathit{true}} a_2$. Then, the satisfaction of the property reachability problem amounts to check the satisfaction of $\lambda_{a_1,a_2}(\theta)$. □

References

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

[2] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, 2002.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.

[4] B. Courcelle. The monadic second-order logic of graphs, ii: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21:187–222, 1989.

[5] B. Courcelle. The monadic second order logic of graphs ix: Machines and their behaviours. In *Theoretical Computer Science*, volume 151, pages 125–162, 1995.

[6] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.

[7] M. Hack. The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems. In *15th annual symposium on switching and automata theory*, New York, 1974.

[8] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen, editor, *FOSSACS'02*, LNCS, 2002.

[9] S. Kosaraju. Decidability of reachability in vector addition systems. In *14th Annual Symposium on Theory of Computing*, 1982.

[10] D. Lugiez and J.-L. Moysset. Complement problems and tree automata in ac-like theories. In P. Enjalbert, A. Finkel, and K.W. Wagner, editors, *STACS'93*, volume 665, pages 515–524, Würzburg (Germany), February 1993. LNCS.

[11] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, volume 1466, pages 50–66. Springer, 1998.

[12] E. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on computing*, 13:441–460, 1981.

[13] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

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

[15] E. Spanier S. Ginsburg. Semigroups, presburger formulas and languages. *Pacific J. Maths 16*, pages 285–296, 1966.