INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Decidable Approximations of Sets of Descendants and Sets of Normal Forms

– extended version –

Thomas Genet

N° 3325

December 1997

THÈME 2

Rapport de recherche

ISSN 0249-6399

Decidable Approximations of Sets of Descendants and Sets of Normal Forms

– extended version –

Thomas Genet*

Thème 2 — Génie logiciel
et calcul symbolique
Projet PROTHEO

Rapport de recherche n° 3325 — December 1997 — 28 pages

Abstract: We present here decidable approximations of sets of descendants and sets of normal forms of Term Rewriting Systems, based on specific tree automata techniques. In the context of rewriting logic, a Term Rewriting System is a program, and a normal form is a result of the program. Thus, approximations of sets of descendants and sets of normal forms provide tools for analysing a few properties of programs: we show how to compute a superset of results, to prove the sufficient completeness property, or to find a criterion for proving termination under a specific strategy, the sequential reduction strategy.

Key-words: Term Rewriting, Program Verification, Normal Forms, Descendants, Tree Automata, Approximation, Sufficient Completeness, Reachability, Termination.

(Résumé : tsvp)

* Email: Thomas.Genet@loria.fr, http://www.loria.fr/equipe/protheo.html

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

Approximations calculables des ensembles de descendants et des ensembles de formes normales

– version étendue –

Résumé : A partir de techniques d'automates d'arbres, nous présentons des approximations calculables des ensembles de descendants et des ensembles de formes normales d'un système de réécriture. Dans le contexte de la logique de réécriture, un système de réécriture est un programme et une forme normale est un résultat du programme. Ainsi, l'approximation de l'ensemble des descendants et de l'ensemble des formes normales, fournit des outils pour la vérification des programmes: nous montrons en particulier comment calculer un sur-ensemble des résultats, comment montrer la complétude suffisante, ou encore comment prouver la terminaison sous une stratégie précise, la stratégie de réduction séquentielle.

Mots-clé : Systèmes de réécriture, Vérification de programmes, Descendants, Formes Normales, Automate d'arbre, Approximation, Complétude suffisante, Atteignabilité, Terminaison

Contents

Introduction

In the context of the programming language such as ELAN [KKV95], ASF+SDF [Kli93], MAUDE [CELM96], OBJ [GKK+87], a Term Rewriting System (TRS for short) is a program. We propose here to use tree automata techniques for proving various properties on TRSs and thus on programs. For a given TRS $\mathcal{R}$ and a set of terms E, these proofs are based on the computation of approximations of the set of $\mathcal{R}$-descendants of E and the set of $\mathcal{R}$-normal forms of E. For that, we build an approximation automaton which recognises a superset of the set of $\mathcal{R}$-descendants and $\mathcal{R}$-normal forms of terms in E. Considering $\mathcal{R}$ as a program and E as the set of possible inputs of the program, the set of $\mathcal{R}$-descendants of E represents all intermediate results of the program at every step of its execution on the given set of possible inputs. The set of $\mathcal{R}$-normal forms of E represents the set of all possible results obtained by executing the program $\mathcal{R}$ on the set of possible given inputs E, when the program stops. Thanks to those two sets, we show how to prove sufficient completeness of a program on a set of possible initial inputs, how to achieve some reachability testing on a program, and how to prove termination of a program represented by a TRS and a strategy of application of rewrite rules called sequential reduction strategy.

In Section 1, we recall basic definitions of terms, term rewriting systems, and tree automata. In Section 2, we briefly present sufficient completeness, reachability testing and termination proof under the sequential reduction strategy. Then, in Section 3, we recall some undecidability results on the set of descendants and the set of normal forms motivating our approach by approximation. We also detail the approximation construction which is based on specific matching and rewriting techniques on tree automata, schematising matching and rewriting on sets of terms. Feasibility of the approximation construction and its appropriateness for our purpose is shown in Section 4 on some examples. Some automatic proofs achieved by our prototype are also presented in Section 4. Finally we conclude on this work in Section 5.

1 Preliminaries

We now introduce some notations and basic definitions. Comprehensive surveys can be found in [DJ90] for term rewriting systems, in [GS84, CDG+97] for tree automata and tree language theory, and in [GT95] for connections between regular tree languages and term rewriting systems.

Terms, Substitutions, Rewriting systems

Let $\mathcal{F}$ be a finite set of symbols associated with an arity function denoted by $ar : \mathcal{F} \mapsto \mathbb{N}$, $\mathcal{X}$ be a countable set of variables, $\mathcal{T}(\mathcal{F}, \mathcal{X})$ the set of terms, and $\mathcal{T}(\mathcal{F})$ the set of ground terms (terms without variables). Positions in a term are represented as sequences of integers. The set of positions in a term t, denoted by $\mathcal{P}os(t)$, is ordered by lexicographic ordering $\prec$. The empty sequence ϵ denotes the top-most position. $\mathcal{R}oot(t)$ denotes the symbol at position ϵ in

t. For any term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we denote by $\mathcal{P}os_{\mathcal{F}}(s)$ the set of functional positions in s, i.e. $\{p \in \mathcal{P}os(s) \mid p \neq \epsilon$ and $\mathcal{R}oot(s|_p) \in \mathcal{F}\}$. If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. A *ground context* is a term of $\mathcal{T}(\mathcal{F} \cup \{\Box\})$ with only one occurrence of $\Box$, where $\Box$ is a special constant not occurring in $\mathcal{F}$. For any term $t \in \mathcal{T}(\mathcal{F})$, $C[t]$ denotes the term obtained after replacement of $\Box$ by t in the ground context $C[\,]$. The set of variables of a term t is denoted by $\mathcal{V}ar(t)$. A term is linear if any variable of $\mathcal{V}ar(t)$ has exactly one occurrence in t. A substitution is a mapping σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$, which can uniquely be extended to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Its domain $\mathcal{D}om(\sigma)$ is $\{x \in X \mid x\sigma \neq x\}$.

A term rewriting system $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if the left-hand side (resp. right-hand side) of the rule is linear. A rule is linear if it is both left and right-linear. A TRS $\mathcal{R}$ is linear (resp. left-linear, right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is linear (resp. left-linear, right-linear).

The relation $\to_{\mathcal{R}}$ induced by $\mathcal{R}$ is defined as follows: for any $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \to_{\mathcal{R}} t$ if there exist a rule $l \to r$ in $\mathcal{R}$, a position $p \in \mathcal{P}os(s)$ and a substitution σ such that $l\sigma = s|_p$ and $t = s[r\sigma]_p$. The transitive (resp. reflexive transitive) closure of $\to_{\mathcal{R}}$ is denoted by $\to_{\mathcal{R}}^{+}$ (resp. $\to_{\mathcal{R}}^{*}$). A term s is *reducible* by $\mathcal{R}$ if there exists t s.t. $s \to_{\mathcal{R}} t$.

A term s is in $\mathcal{R}$-normal form (or is *$\mathcal{R}$-irreducible*) if s is not reducible by $\mathcal{R}$. A term s *has a normal form* if there exists a term t in $\mathcal{R}$-normal form s.t. $s \to_{\mathcal{R}}^{*} t$. The set of all ground terms in $\mathcal{R}$-normal form is denoted by $IRR(\mathcal{R})$, and $s \to_{\mathcal{R}}^{*} t$ with $t \in IRR(\mathcal{R})$ is denoted by $s \to_{\mathcal{R}}^{!} t$. The set of $\mathcal{R}$-descendants of a set of ground terms E is denoted by $\mathcal{R}^*(E)$ and $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to_{\mathcal{R}}^{*} t\}$. The set of ground $\mathcal{R}$-normal forms of E is denoted by $\mathcal{R}^!(E)$ and $\mathcal{R}^!(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to_{\mathcal{R}}^{!} t\}$. Moreover, $\mathcal{R}^!(E) = \mathcal{R}^*(E) \cap IRR(\mathcal{R})$. A rewriting system $\mathcal{R}$ is

(1) *confluent* if for every $s, t, u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \to_{\mathcal{R}}^{*} t$ and $s \to_{\mathcal{R}}^{*} u$ implies that there exists a $v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $t \to_{\mathcal{R}}^{*} v$ and $u \to_{\mathcal{R}}^{*} v$,

(2) *terminating* or *strongly normalising* if there exists no infinite derivation $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \ldots$ where $s_1, s_2, \ldots \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,

(3) *weakly normalising* (WN for short) if every s of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ has a normal form,

(4) *weakly normalising on* $E \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ (WN on E) if every $s \in E$ has a normal form.

The set of function symbols $\mathcal{F}$ occurring in a TRS $\mathcal{R}$ can be partitioned into the set of *defined symbols* $\mathcal{D} = \{\mathcal{R}oot(l) \mid l \to r \in \mathcal{R}\}$ and the set of *constructors* $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. A *constructor term*, is a ground term with no defined symbol. The set of constructor terms is denoted by $\mathcal{T}(\mathcal{C})$. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be TRSs with respective sets of symbols $\mathcal{F}_1$ and $\mathcal{F}_2$, respective sets of defined symbols $\mathcal{D}_1$ and $\mathcal{D}_2$, and respective sets of constructors $\mathcal{C}_1$ and $\mathcal{C}_2$. TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ are *hierarchical* if $\mathcal{F}_2 \cap \mathcal{D}_1 = \emptyset$ and $\mathcal{R}_1 \subset \mathcal{T}(\mathcal{F}_1 \setminus \mathcal{D}_2, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_1, \mathcal{X})$.

Automata, Regular Tree Languages

Let $\mathcal{Q}$ be a finite set of symbols, with arity 0, called *states*. $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is called the set of *configurations*. A *transition* is a rewrite rule $c \to q$, where $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. A *normalised transition* is a transition $c \to q$ where $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$, $ar(f) = n$, and $q_1, \ldots, q_n \in \mathcal{Q}$. A bottom-up finite tree automaton (tree automaton for short) is a quadruple $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f \subseteq \mathcal{Q}$ and Δ is a set of normalised transitions. The rewriting relation induced by Δ is denoted by $\rightarrow_\Delta$. The tree language recognised by A is $\mathcal{L}(A) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \text{ s.t. } t \rightarrow_\Delta^* q\}$. For a given $q \in \mathcal{Q}$, the tree language recognised by A and q is $\mathcal{L}(A, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow_\Delta^* q\}$. A tree language (or a set of terms) E is *regular* if there exists a bottom-up tree automaton A such that $\mathcal{L}(A) = E$. The class of regular tree language is closed under boolean operations $\cup, \cap, \setminus$, and inclusion is decidable.

A $\mathcal{Q}$-substitution is a substitution σ s.t. $\forall x \in \mathcal{D}om(\sigma)$, $x\sigma \in \mathcal{Q}$. Let $\Sigma(\mathcal{Q}, \mathcal{X})$ be the set of $\mathcal{Q}$-substitutions. For every transition, there exists an equivalent set of normalised transitions. Normalisation consists in decomposing a transition $s \rightarrow q$, into a set $Norm(s \rightarrow q)$ of flat transitions $f(u_1, \ldots, u_n) \rightarrow q'$ where $u_1, \ldots, u_n$, and q' are states, by abstracting subterms $s' \notin \mathcal{Q}$ of s by states. We first define the abstraction function as follows:

Definition 1 *Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. For a given configuration $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$, an abstraction of s is a surjective mapping α:*

$$\alpha : \{s|_p \mid p \in \mathcal{P}os_{\mathcal{F}}(s)\} \mapsto \mathcal{Q}$$

The mapping α is extended on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ by defining α as identity on $\mathcal{Q}$.

Definition 2 *Let $\mathcal{F}$ be a set of symbols, $\mathcal{Q}$ a set of states, $s \rightarrow q$ a transition s.t. $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$, and α an abstraction of s. The set $Norm_\alpha(s \rightarrow q)$ of normalised transitions is inductively defined by:*

- *if $s = q$, then $Norm_\alpha(s \rightarrow q) = \emptyset$, and*
- *if $s \in \mathcal{Q}$ and $s \neq q$, then $Norm_\alpha(s \rightarrow q) = \{s \rightarrow q\}$, and*
- *if $s = f(t_1, \ldots, t_n)$, then $Norm_\alpha(s \rightarrow q) = \{f(\alpha(t_1), \ldots, \alpha(t_n)) \rightarrow q\} \cup \bigcup_{i=1}^n Norm_\alpha(t_i \rightarrow \alpha(t_i))$.*

Example 1 *Let $\mathcal{F} = \{f, g, a\}$ and $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_0\}$, and $\Delta = \{f(q_1) \rightarrow q_0, g(q_1, q_1) \rightarrow q_1, a \rightarrow q_1\}$.*

• *The languages recognised by q_1 and q_0 are the following: $\mathcal{L}(A, q_1) = \mathcal{T}(\{g, a\})$, and $\mathcal{L}(A, q_0) = \mathcal{L}(A) = \{f(x) \mid x \in \mathcal{L}(A, q_1)\}$.*

• *Let $s = f(g(q_1, f(a)))$, and α_1 be an abstraction of s, mapping any subterm $s|_p$ with $p \in \mathcal{P}os_{\mathcal{F}}(s)$, to distinct states in $\{q_2, q_3, q_4\}$. A possible normalisation of transition $f(g(q_1, f(a))) \rightarrow q_0$ with abstraction α_1 is the following: $Norm_{\alpha_1}(f(g(q_1, f(a))) \rightarrow q_0) = \{f(q_2) \rightarrow q_0, g(q_1, q_3) \rightarrow q_2, f(q_4) \rightarrow q_3, a \rightarrow q_4\}$.*

2 Applications of $\mathcal{R}^*(E)$ and $\mathcal{R}^!(E)$

In this section we present three applications of the set of descendants and the set of normal forms to program and system verification.

2.1 Sufficient Completeness

This property has already been much investigated [Com86, Kou85, NW83, KNZ87], in the context of algebraic specifications. We give here a definition of sufficient completeness of a TRS on a subset of the set of ground terms $E \subseteq \mathcal{T}(\mathcal{F})$.

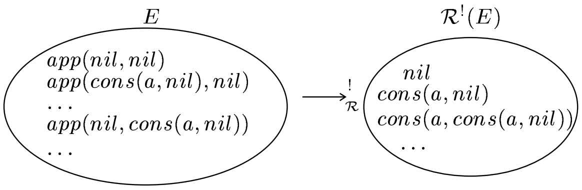
Definition 3 *A TRS $\mathcal{R}$ is sufficiently complete on $E \subseteq \mathcal{T}(\mathcal{F})$ if $\forall s \in E$, $\exists t \in \mathcal{T}(\mathcal{C})$ s.t. $s \rightarrow_{\mathcal{R}}^{*} t$, where $\mathcal{C}$ is the set of constructors in $\mathcal{F}$.*

Usual methods for checking this property on algebraic specifications are either based on enumeration and testing techniques [Kou85, NW83, KNZ87] or on disunification [Com86]. We propose, here, to check this property thanks to the set $\mathcal{R}^!(E)$.

Proposition 1 *If the TRS $\mathcal{R}$ is WN on $E \subseteq \mathcal{T}(\mathcal{F})$, and $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$, then $\mathcal{R}$ is sufficiently complete on E.*

This comes from the fact that since $\mathcal{R}$ is WN on E, for all terms $s \in E$, $\exists t \in IRR(\mathcal{R})$ s.t. $s \rightarrow_{\mathcal{R}}^{*} t$. Moreover, $t \in \mathcal{R}^!(E)$. Since $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$, we have $t \in \mathcal{T}(\mathcal{C})$.

Example 2 *Let $\mathcal{R} = \{app(nil, x) \rightarrow x, app(cons(x, y), z) \rightarrow cons(x, app(y, z))\}$, $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$, where $\mathcal{D} = \{app\}$ and $\mathcal{C} = \{cons, nil, a\}$.*



Since $\mathcal{R}$ is terminating, $\mathcal{R}$ is WN on E and since $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$, then $\mathcal{R}$ is sufficiently complete on E.

On the other hand, sufficient completeness on E does not necessarily imply that $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$. For example, let $\mathcal{R} = \{f(a) \rightarrow a, f(a) \rightarrow f(b)\}$, $\mathcal{C} = \{a, b\}$ and let $E = \{f(a)\}$. Then $\mathcal{R}$ is sufficiently complete on E, since $f(a) \rightarrow_{\mathcal{R}} a$, but $\mathcal{R}^!(E) = \{a, f(b)\} \not\subseteq \mathcal{T}(\mathcal{C})$.

2.2 Reachability Testing

Reachability testing consists in verifying if a term, or a term containing a pattern, can be reached by rewriting from an initial set E.

Definition 4 *Let $\mathcal{R}$ be a TRS, $E \subseteq \mathcal{T}(\mathcal{F})$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. The* pattern t is $\mathcal{R}$-reachable from E *if there exists a ground context $C[\,]$, a term $s \in E$, and a substitution σ s.t. $s \rightarrow_{\mathcal{R}}^{*} C[t\sigma]$.*

It is clear that

Proposition 2 *A pattern t is $\mathcal{R}$-reachable from E if an instance of t is a subterm of an element of $\mathcal{R}^*(E)$.*

Let us now show what can be the use of reachability testing on a simple example.

Example 3 *Assume that we want to compute $A_n^p = \frac{n!}{(n-p)!}$ with the following TRS:*

$$\mathcal{R} = \left\{ \begin{array}{ll} A(n,p) \to fact(n)/fact(n-p) & 0 * x \to x \\ fact(0) \to s(0) & s(x) * y \to (x * y) + y \\ fact(s(x)) \to s(x) * fact(x) & 0/s(y) \to 0 \\ x - 0 \to x & s(x)/s(y) \to s((x-y)/s(y)) \\ 0 - x \to 0 & x + 0 \to x \\ s(x) - s(y) \to x - y & x + s(y) \to s(x+y) \end{array} \right\}$$

on the domain $E = \{A(n,p) \mid n,p \in Nat\}$, where $Nat = \{0, s(0), \ldots\}$. Verifying if a division by 0 can occur is equivalent to check whether the pattern $div(x,0)$ is $\mathcal{R}$-reachable from E, i.e. whether $\exists C[\,]$, $\exists \sigma$, s.t. $C[div(x,0)\sigma] \in \mathcal{R}^*(E)$.

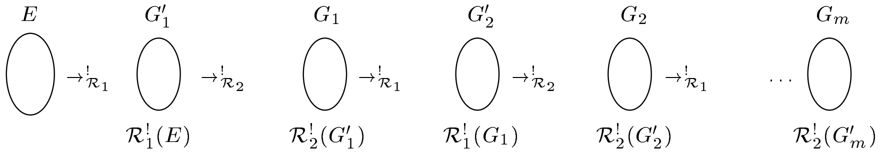
2.3 Termination under Sequential Reduction Strategy

Many works are devoted to automatising termination proofs of TRSs [AG97a, GG97]. On the other hand, it is interesting to study weaker forms of termination, since for many purposes weak normalisation is enough. In theorem provers and programming languages, rules are always applied under a specific strategy, and it is enough to ensure termination under this strategy. In addition, proving termination or WN on $\mathcal{T}(\mathcal{F},\mathcal{X})$ or on $\mathcal{T}(\mathcal{F})$ is not always needed. In practice, a TRS is often designed to rewrite terms from a subset $E \subseteq \mathcal{T}(\mathcal{F})$, for example logical formulas in disjunctive normal form, flattened lists, or well-typed terms. Moreover, some TRSs are WN on $E \subset \mathcal{T}(\mathcal{F})$, but not on $\mathcal{T}(\mathcal{F})$ [Gen97].

The strategy studied here is called the *Sequential Reduction Strategy* (SRS for short) and consists in separating a TRS $\mathcal{R}$ into several TRSs $\mathcal{R}_1, \ldots, \mathcal{R}_n$ s.t. $\mathcal{R} = \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n$ and in normalising terms successively w.r.t. $\mathcal{R}_1, \ldots, \mathcal{R}_n$. This rewriting relation under SRS is denoted by $\twoheadrightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n}$, and is based on *modular reduction relation* [KK90].

Definition 5 *Let $\mathcal{R} = \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n$ be TRSs. For $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $s \twoheadrightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n} t$ if s is reducible by $\mathcal{R}$ and $\exists s_1, \ldots, s_{n-1} \in \mathcal{T}(\mathcal{F},\mathcal{X})$ s.t. $s \to^!_{\mathcal{R}_1} s_1$ and $s_1 \to^!_{\mathcal{R}_2} s_2$ and $\ldots$ and $s_{n-1} \to^!_{\mathcal{R}_n} t$.*

This kind of strategy is of great interest when normalising terms w.r.t. a TRS splitted into several hierarchical TRSs (or modules) $\mathcal{R}_1, \ldots, \mathcal{R}_n$. In this situation, interleaving of rewriting steps w.r.t. to $\mathcal{R}_1, \ldots, \mathcal{R}_n$ is often not needed, and sometimes it is even not possible: for example when using rewriting with some built-in terms normalised at once by a built-in module. If modules $\mathcal{R}_1, \ldots, \mathcal{R}_n$ are WN and share only constructors, then $\twoheadrightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n}$ is terminating [Gen97], as a corollary of results of [KO91, Ohl94]. Now let us give an intuition on how to prove termination of $\twoheadrightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n}$ for WN TRSs $\mathcal{R}_1, \ldots, \mathcal{R}_n$ sharing function symbols. Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, for example. For proving termination of $\twoheadrightarrow_{\mathcal{R}_1;\mathcal{R}_2}$ on a set of initial terms $E \subseteq \mathcal{T}(\mathcal{F})$, we need to prove that for any term $s \in E$, there is no possible infinite derivation $s \rightarrow^!_{\mathcal{R}_1} s'_1 \rightarrow^!_{\mathcal{R}_2} s_1 \rightarrow^!_{\mathcal{R}_1} s'_2 \rightarrow^!_{\mathcal{R}_2} s_2 \rightarrow^!_{\mathcal{R}_1} \ldots$ In that case, a criterion for proving termination of $\rightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n}$ on E is the following: construct the sets $G_1 = \mathcal{R}^!_2(\mathcal{R}^!_1(E))$, $G_2 = \mathcal{R}^!_2(\mathcal{R}^!_1(G_1))$, $\ldots$ until we get a fixpoint G_m s.t. $G_m = \mathcal{R}^!_2(\mathcal{R}^!_1(G_m))$.



If $\mathcal{R}_1$ and $\mathcal{R}_2$ are WN on $E_1 \subseteq \mathcal{T}(\mathcal{F})$ and on $E_2 \subseteq \mathcal{T}(\mathcal{F})$, respectively, and $E \subseteq E_1$, $G_i \subseteq E_1$, $G'_i \subseteq E_2$ for all $i = 1 \ldots m-1$, then $\mathcal{R}_1$ is WN on E, G_i and $\mathcal{R}_2$ is WN on G'_i, for all $i = 1 \ldots m-1$. Furthermore, if $G_m \subseteq IRR(\mathcal{R}_1 \cup \mathcal{R}_2)$, then $\mathcal{R}$ is WN on E and $\mathcal{R}$ is terminating on E under SRS.

Proposition 3 *If $\mathcal{R}_1, \ldots, \mathcal{R}_n$ are WN resp. on subsets $E_1, \ldots, E_n$ of $\mathcal{T}(\mathcal{F})$, the rewriting relation under SRS $\rightarrow_{\mathcal{R}_1;\ldots;\mathcal{R}_n}$ is terminating on E if the iterated sequence of sets $G_{k+1} = \mathcal{R}^!_n(\ldots\mathcal{R}^!_1(G_k)\ldots)$, starting from $G_0 = E$, has a fixpoint which is a subset of $IRR(\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n)$, and for all $k \geq 0$, $G_k \subseteq E_1, \mathcal{R}^!_1(G_k) \subseteq E_2, \ldots$, and $\mathcal{R}^!_{n-1}(\ldots\mathcal{R}^!_1(G_k)\ldots) \subseteq E_n$.*

3 Approximating $\mathcal{R}^*(E)$ and $\mathcal{R}^!(E)$

First, recall that $\mathcal{R}^!(E) = \mathcal{R}^*(E) \cap IRR(\mathcal{R})$. $IRR(\mathcal{R})$ is a regular tree language if $\mathcal{R}$ is left-linear [GB85], and a procedure for building a regular tree grammar (resp. a tree automaton) producing (resp. recognising) $IRR(\mathcal{R})$ can be found in [CR87]. However, $\mathcal{R}^*(E)$ is not necessarily a regular tree language, even if E is. The language $\mathcal{R}^*(E)$ is regular if E is regular and if $\mathcal{R}$ is either a ground TRS [DT90], a right-linear and monadic TRS [Sal88], a linear and semi-monadic TRS [CDGV91] or an "inversely-growing" TRS [Jac96], where "inversely-growing" means that every right-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $ar(f) = n$, and $\forall i = 1, \ldots, n$, t_i is a variable, a ground term, or a term whose variables do not occur in the left-hand side. However, for a given regular language E, $\mathcal{R}^*(E)$ is not necessarily regular, even if $\mathcal{R}$ is a confluent and terminating linear TRS [GT95]. If $\mathcal{R}$ is not "inversely-growing", then $\mathcal{R}^*(E)$ is not necessarily regular [Jac96].

Since our purpose is to deal with TRSs representing programs, we cannot stick to the decidable class of "inversely-growing" TRSs which is not expressive enough. Our goal here is to define, an *approximation of* $\mathcal{R}^*(E)$ i.e. a regular superset of $\mathcal{R}^*(E)$ for left-linear TRSs and regular sets E. Then, since regular langages are closed by intersection, the intersection between the regular superset of $\mathcal{R}^*(E)$ and $IRR(\mathcal{R})$ gives a regular superset of $\mathcal{R}^!(E)$. Before going into details of the construction of the approximation itself, let us first show why an approximation is sufficient for proving properties addressed in Section 2. Let $\mathcal{R}$ be a TRS, and $E \subseteq \mathcal{T}(\mathcal{F})$. For any set G, let $super(G)$ be a regular superset of G.

- For sufficient completeness: if $super(\mathcal{R}^!(E)) \subseteq \mathcal{T}(\mathcal{C})$ then $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$,
- For reachability testing: if $C[t\sigma] \notin super(\mathcal{R}^*(E))$ then $C[t\sigma] \notin \mathcal{R}^*(E)$,

– For termination under SRS: if $super(\mathcal{R}_n^!(\ldots super(\mathcal{R}_1^!(G_k))\ldots\ldots))$ is a subset of $IRR(\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n)$ then so is $\mathcal{R}_n^!(\ldots \mathcal{R}_1^!(G_k)\ldots)$.

Now, starting from a tree automaton A s.t. $\mathcal{L}(A) = E$ and a left-linear TRS $\mathcal{R}$, we show how to build a tree automaton $\mathcal{T}_\mathcal{R}\!\uparrow\!(A)$ s.t. $\mathcal{L}(\mathcal{T}_\mathcal{R}\!\uparrow\!(A)) \supseteq \mathcal{R}^*(\mathcal{L}(A))$. The next proposition gives a sufficient condition for an automaton B to have such a property.

Proposition 4 *Let $\mathcal{R}$ be a left-linear TRS, $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$, and $B = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta'\rangle$ two tree automata. $\mathcal{R}^*(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$ if*

1. *$\Delta \subseteq \Delta'$, and*
2. *$\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma \in \Sigma(\mathcal{Q}', \mathcal{X})$, $l\sigma \to_{\Delta'}^* q$ implies $r\sigma \to_{\Delta'}^* q$.*

Proof (sketch) By definition, any term t of $\mathcal{R}^*(\mathcal{L}(A))$ is such that $\exists s \in \mathcal{L}(A)$ s.t. $s \to_\mathcal{R}^* t$. By induction on the size of the derivation $s \to_\mathcal{R}^* t$, we prove that if $s \to_\mathcal{R}^* t$ and $s \to_{\Delta'}^* q$ with $q \in \mathcal{Q}_f$ then $t \to_{\Delta'}^* q$, which implies that $t \in \mathcal{L}(B)$. See Appendix A, for a detailed proof. □

For building $\mathcal{T}_\mathcal{R}\!\uparrow(A)$, the algorithm we propose starts from the tree automaton A and incrementally adds to Δ the transitions necessary to ensure Condition 2, by computing critical peaks between rules of $\mathcal{R}$ and rules of Δ:

$$\begin{tikzcd}
l\sigma \arrow[r, "*", "\Delta"'] \arrow[d, "R"'] & q \\
r\sigma \arrow[ur, dashed, "*", "\Delta"'] &
\end{tikzcd}$$

If $r\sigma \not\to_\Delta^* q$, then it is necessary to add the transition $r\sigma \to q$ to Δ. If the transition $r\sigma \to q$ is not normalised, then it has to be normalised according to Definition 2. The choice of new states used to normalise $r\sigma \to q$ is guided by the approximation function γ defined below:

Definition 6 *Let $\mathcal{Q}$ be a set of states, $\mathcal{Q}_{new}$ be a set of new states s.t. $\mathcal{Q} \cap \mathcal{Q}_{new} = \emptyset$, and $\mathcal{Q}_{new}^*$ the set of sequences $q_1 \cdots q_k$ of states in $\mathcal{Q}_{new}$. An approximation function is a mapping $\gamma : \mathcal{R} \times (\mathcal{Q} \cup \mathcal{Q}_{new}) \times \Sigma(\mathcal{Q} \cup \mathcal{Q}_{new}, \mathcal{X}) \mapsto \mathcal{Q}_{new}^*$, such that $\gamma(l \to r, q, \sigma) = q_1 \cdots q_k$, where $k = \mathcal{C}ard(\mathcal{P}os_\mathcal{F}(r))$.*

In the following, for any sequence $S = q_1 \cdots q_k \in \mathcal{Q}_{new}^*$, and for all i s.t. $1 \le i \le k$, $\pi_i(S)$ denotes the i-th element of the sequence S, i.e. q_i.

Definition 7 *(Approximation Automaton) Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be a tree automaton, $\mathcal{R}$ a left-linear TRS, $\mathcal{Q}_{new}$ a set of new states s.t. $\mathcal{Q} \cap \mathcal{Q}_{new} = \emptyset$, and γ an approximation function. An approximation automaton $\mathcal{T}_\mathcal{R}\!\uparrow\!(A)$ is a tree automaton $\langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta'\rangle$ s.t.*

- *$\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}_{new}$, and*
- *$\Delta \subseteq \Delta'$, and*

- *$\forall l \rightarrow r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma \in \Sigma(\mathcal{Q}', \mathcal{X})$, $l\sigma \rightarrow^*_{\Delta'} q$ implies*

$$Norm_\alpha(r\sigma \rightarrow q) \subseteq \Delta'$$

where α is the abstraction of $r\sigma$ defined by: $\alpha(r\sigma|_{p_i}) = \pi_i(\gamma(l \rightarrow r, q, \sigma))$, for all $p_i \in \mathcal{P}os_{\mathcal{F}}(r) = \{p_1, \ldots, p_k\}$, s.t. $p_i \prec p_{i+1}$ for $i = 1 \ldots k-1$ (where $\prec$ is the lexicographic ordering).

By choosing specific approximation functions γ, we obtain specific approximations.

Theorem 1 *Given a tree automaton A and a left-linear TRS $\mathcal{R}$, every approximation automaton satisfies: for any approximation function γ,*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(A)) \supseteq \mathcal{R}^*(\mathcal{L}(A))$$

Proof (sketch) For proving $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(A)) \supseteq \mathcal{R}^*\mathcal{L}(A)$, it is enough to prove that the approximation automata verifies Conditions 1 and 2 of Proposition 4, for all approximation functions γ. By Definition 7, $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ trivially verifies Condition 1. Then, to prove that $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ also verifies Condition 2 of Proposition 4, it is enough to prove that $Norm_\alpha(r\sigma \rightarrow q) \subseteq \Delta'$ implies $r\sigma \rightarrow^*_{\Delta'} q$. See Appendix C for a detailed proof. $\Box$

For any rule $l \rightarrow r \in \mathcal{R}$, in order to find a $\mathcal{Q}$-substitution σ and a state $q \in \mathcal{Q}$ s.t. $l\sigma \rightarrow^*_\Delta q$, it is possible to enumerate every possible combination of σ and q and check whether $l\sigma \rightarrow^*_\Delta q$. However, this solution is not usable in practice, especially when $\mathcal{Q}$ is a large set, due to the huge number of possible σ and q to consider. In Appendix B, we detail a *matching algorithm* which starts from a *matching problem* $l \trianglelefteq q$ and a set of transitions Δ, and gives every solution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ s.t. $l\sigma \rightarrow^*_\Delta q$. This algorithm is used in our implementation.

However, adding transitions to Δ may not terminate, depending on the approximation function γ used, as in the following example.

Example 4 *Let A be a tree automaton where $\Delta = \{app(q_0, q_0) \rightarrow q_1, cons(q_2, q_1) \rightarrow q_0, nil \rightarrow q_0, nil \rightarrow q_1, a \rightarrow q_2\}$, $rl = app(cons(x, y), z) \rightarrow cons(x, app(y, z))$, $\mathcal{R} = \{rl\}$, and let γ be the approximation function mapping every tuple (rl, q, σ) to one new state (since $Card(\mathcal{P}os_{\mathcal{F}}(cons(x, app(y, z)))) = 1$).*

Step 1 *If we apply the matching algorithm on $app(cons(x, y), z) \trianglelefteq q_1$, we obtain a solution $\sigma = \{x \mapsto q_2, y \mapsto q_1, z \rightarrow q_0\}$, corresponding to the following critical peak:*

$$\begin{tikzcd}
app(cons(q_2, q_1), q_0) \arrow[r, "*", "\Delta"'] \arrow[d, "R"'] & q_1 \\
cons(q_2, app(q_1, q_0)) \arrow[ur, dashed, "*", "\Delta"'] &
\end{tikzcd}$$

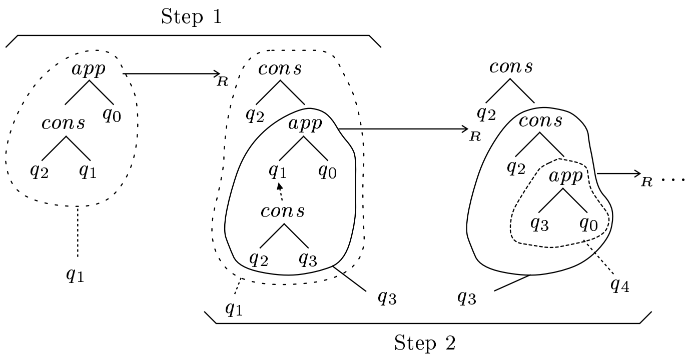
Let q_3 be the new state s.t. $\gamma(rl, q_1, \sigma) = q_3$. Then, since $\mathcal{P}os_{\mathcal{F}}(cons(q_2, app(q_1, q_0)) = \{p_1\} = \{2\}$, we have $\alpha(app(q_1, q_0)) = \pi_1(\gamma(rl, q_1, \sigma)) = q_3$, and the set of normalised transitions to be added to Δ is $Norm_\alpha(cons(q_2, app(q_1, q_0)) \rightarrow q_1) = \{cons(q_2, q_3) \rightarrow q_1, app(q_1, q_0) \rightarrow q_3\}$.

Step 2 *Applying the matching algorithm on* $app(cons(x, y), z) \trianglelefteq q_3$ *gives a solution* $\sigma' = \{x \mapsto q_2, y \mapsto q_3, z \mapsto q_0\}$, *corresponding to the following critical peak:*

$$\begin{tikzcd} app(cons(q_2, q_3), q_0) \arrow[r, "*", "\Delta"'] \arrow[d, "R"'] & q_3 \\ cons(q_2, app(q_3, q_0)) \arrow[ur, dashed, "*\ \Delta"'] & \end{tikzcd}$$

Let q_4 *be the new state s.t.* $\gamma(rl, q_3, \sigma') = q_4$. *Then,* $\alpha(app(q_3, q_0)) = q_4$, *and the set of normalised transitions to be added to* Δ *is* $Norm_\alpha(cons(q_2, app(q_3, q_0)) \to q_3) = \{cons(q_2, q_4) \to q_1, app(q_3, q_0) \to q_4\}$.

This process can go on forever and add infinitely many new states. This is due to the fact that we can apply recursively the rule $app(cons(x, y), z) \to cons(x, app(y, z))$ onto infinitely growing terms recognised by the automaton A (with transitions Δ), as shown on the following figure.



In order to have a finite automaton approximating the set $\mathcal{R}^*(\mathcal{L}(A))$, the intuition is to *fold recursive calls* into a unique new state. In the previous example, during Step 1, by applying the rule of $app(cons(x, y), z) \to cons(x, app(y, z))$ on $app(cons(q_2, q_1), q_0)$, we have obtained the configuration $cons(q_2, app(q_1, q_0))$, and we have created a new state q_3 recognising the subterm $app(q_1, q_0)$. During Step 2 we have applied *the same rule* on the subterm $app(q_1, q_0)$ recognised by q_3. In order to fold this recursive call in Step 2, we simply *re-use* the state q_3, instead of creating a new state q_4 for normalising the transition $cons(q_2, app(q_3, q_0)) \to q_3$ obtained in Step 2. Thus we obtain the set of normalised transitions $\{cons(q_2, q_3) \to q_3, app(q_3, q_0) \to q_3\}$ to be added to Δ. No more state nor transition needs to be further added and this automaton recognises a superset of $\mathcal{R}^*(\mathcal{L}(A))$. This is one of the basic idea of the ancestor approximation, which is formalised below.

Informally, every state $q \in \mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}_{new}$ has a unique ancestor $q_a \in \mathcal{Q}$. The ancestor of any state $q \in \mathcal{Q}$ is q itself, and the ancestor of every new state $q' \in \mathcal{Q}_{new}$ occuring in the sequence $\gamma(l \to r, q, \sigma)$ (used to normalise a new transition $r\sigma \to q$), is the ancestor of q. In the ancestor approximation, (1) the γ function does not depend on the σ parameter and,

(2) for any new state $q' \in \mathcal{Q}_{new}$, $\gamma(l \to r, q', \sigma) = \gamma(l \to r, q, \sigma)$, where $q \in \mathcal{Q}$ is the ancestor of q'.

Definition 8 *An approximation function γ is called ancestor approximation if*

1. $\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma_1, \sigma_2 \in \Sigma(\mathcal{Q}', \mathcal{X})$,

$$\gamma(l \to r, q, \sigma_1) = \gamma(l \to r, q, \sigma_2), \textit{ and}$$

2. $\forall l_1 \to r_1, l_2 \to r_2 \in \mathcal{R}, \forall q \in \mathcal{Q}', \forall q_1, \ldots, q_k \in \mathcal{Q}_{new}, \sigma_1, \sigma_2 \in \Sigma(\mathcal{Q}', \mathcal{X})$,

$$\gamma(l_1 \to r_1, q, \sigma_1) = q_1 \ldots q_k \Rightarrow \forall i = 1 \ldots k, \gamma(l_2 \to r_2, q_i, \sigma_2) = \gamma(l_2 \to r_2, q, \sigma_2).$$

Note that in the particular case of Example 4, using the ancestor approximation, we have $\gamma(rl, q_1, \sigma) = q_3$, and by case 2 of Definition 8 we get $\gamma(rl, q_3, \sigma') = \gamma(rl, q_1, \sigma')$, by case 1 we get that $\gamma(rl, q_1, \sigma') = \gamma(rl, q_1, \sigma) = q_3$, thus $\gamma(rl, q_3, \sigma') = q_3$, and the construction of $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(A)$ becomes finite.

Theorem 2 *Approximation automata built using ancestor approximation are finite automata.*

Proof (sketch) The automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(A)$ is finite if the set of new states $\mathcal{Q}_{new}$ is finite. Since $\mathcal{Q}$ is finite, $\mathcal{R}$ is finite, and γ does not depend on the σ parameter, there is a finite number of distinct sequences $\gamma(l \to r, q, \sigma)$ for $l \to r \in \mathcal{R}$, $q \in \mathcal{Q}$, and these sequences are finite. On the other hand, every state $q' \in \mathcal{Q}_{new}$ has a unique ancestor $q \in \mathcal{Q}$, and $\gamma(l \to r, q', \sigma) = \gamma(l \to r, q, \sigma)$. Thus, there is a finite number of distinct sequences $\gamma(l \to r, q', \sigma) = q'_1 \ldots q'_n$ with $q', q'_1, \ldots, q'_n \in \mathcal{Q}_{new}$. Hence, there is a finite number of states in $\mathcal{Q}_{new}$, used to normalise transitions. See Appendix D for a detailed proof. $\square$

4 Experiments

Working on tree automaton by hand is always a heavy task. In order to experiment and check feasibility of the method, we have implemented in ELAN [KKV95] a library of usual algorithms on tree automaton: union, intersection, cleaning, inclusion test, as well as algorithms for building the tree automata $\mathcal{T}_{\mathcal{R}}\!\uparrow\!(A)$, and $A_{IRR(\mathcal{R})}$ (the automaton recognising the set $IRR(\mathcal{R})$) for a given automaton A and a given left-linear TRS $\mathcal{R}$. In all the following examples, we use the same ancestor approximation method. We have experimented with several other approximations: if the γ function does not depend on the rule $l \to r$, on the state q or on the position p, then the approximation automaton is smaller, and faster to compute. However, the recognised language is bigger and sometimes not precise enough for our purpose. On the other hand, if for every σ, the γ function have distinct values, then the construction of the automaton is not necessarily terminating.

4.1 Reachability Testing

Let $\mathcal{R}_1$ be a TRS computing the function $A_n^p = \frac{n!}{(n-p)!}$, and $Aut(0)$ a tree automaton recognising the set $\mathcal{L}(Aut(0)) = \mathcal{L}(Aut(0), q_0) = \{A(n,p) \mid n,p \in \mathcal{L}(Aut(0), q_1)\}$ where $\mathcal{L}(Aut(0), q_1) = Nat = \{0, s(0), \ldots\}$. The TRS $\mathcal{R}_1$ and the automaton $Aut(0)$ are given as input to our prototype in the following syntax:

```
specification Anp

Vars    x y n p

Ops
    A:2 minus:2 div:2 o:0 s:1 fact:1 plus:2 mult:2

R1
    A(n, p) -> div(fact(n), fact(minus(n, p)))
    fact(s(x)) -> mult(s(x), fact(x))
    fact(o) -> s(o)
    mult(o, x) -> o
    mult(s(x), y) -> plus(mult(x, y), y)
    div(o, s(y)) -> o
    div(s(x), s(y)) -> s(div(minus(x, y), s(y)))
    plus(x, o) -> x
    plus(x, s(y)) -> s(plus(x, y))
    minus(x, o) -> x
    minus(o, x) -> o
    minus(s(x), s(y)) -> minus(x, y)
    nil

Automata
    Description of Aut(0)
    states q|0.q|1.nil
    final states q|0.nil
    transitions A(q|1, q|1) -> q|0
            o-> q|1
            s(q|1) -> q|1
            nil
    End of Description
    nil
```

Computing the automaton $\mathcal{T}_{\mathcal{R}_1}\!\uparrow(Aut(0))$ s.t. $\mathcal{L}(\mathcal{T}_{\mathcal{R}_1}\!\uparrow(Aut(0))) \supseteq \mathcal{R}_1^*(\mathcal{L}(Aut(0)))$, can be achieved by evaluating the following query:

```
[] start with term :
    T_up(R1) on (Aut(0))
```

And the result is the automaton $Aut(1)$:

```
[] result term:
    Description of Aut(1) states
    q|12.q|13.q|11.q|10.q|9.q|8.q|6.q|7.q|2.q|5.q|3.q|0.q|1.nil final states q|0.nil transitions
    s(q|11)->q|11.o->q|11.minus(q|10,q|10)->q|12.minus(q|8,q|10)->q|12. s(q|10)->q|13.s(q|10)->
    q|3.minus(q|10,q|8)->q|12.minus(q|8,q|8)->q|12.s(q|8)->q|13.div(q|12,q|13)->q|11.s(q|11)->
    q|0.plus(q|9,q|7)->q|3.s(q|8)->q|3.mult(q|6,q|7)->q|3.plus(q|9,q|10)->q|10.plus(q|9,q|10)
    ->q|12.s(q|10)->q|7.s(q|10)->q|10.s(q|10)->q|12.s(q|10)->q|9.plus(q|9,q|8)->q|10.
    plus(q|9,q|8)->q|12.s(q|10)->q|2.plus(q|9,q|7)->q|2.plus(q|9,q|7)->q|9.plus(q|9,q|7)->q|10.
    plus(q|9,q|7)->q|12.o->q|9.o->q|10.o->q|12.mult(q|1,q|7)->q|9.mult(q|1,q|7)->q|10.
    mult(q|1,q|7)->q|12.plus(q|9,q|7)->q|7.s(q|8)->q|2.o->q|8.s(q|8)->q|7.mult(q|6,q|7)->
    q|7.s(q|1)->q|6.fact(q|1)->q|7.mult(q|6,q|7)->q|2.fact(q|1)->q|2.minus(q|1,q|1)->q|5.
    fact(q|5)->q|3.div(q|2,q|3)->q|0.A(q|1,q|1)->q|0.o->q|1.o->q|5.s(q|1)->q|1.s(q|1)->q|5.nil
    End of Description
```

The pattern $div(x, 0)$ is not $\mathcal{R}_1$-reachable from $\mathcal{L}(Aut(0))$ if for all ground contexts $C[\,]$ and all substitutions σ, $C[dix(x, 0)\sigma] \notin \mathcal{L}(\mathcal{T}_{\mathcal{R}_1}\!\uparrow(Aut(0)))$. This is checked using the following query:

```
[] start with term :
    (div(x, o) ?= states) with (Aut(1))

[] result term:
    nil
```

The result is nil, meaning that there exists no substitution σ and no state $q \in \mathcal{Q}$ s.t. $div(x, 0)\sigma \rightarrow^*_\Delta q$, where $\mathcal{Q}$ and Δ are respectively the set of states and the set of transitions of $Aut(1)$.

An interesting aspect of this method is that the automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ is computed once for all, and the check itself is a simple and low cost operation. Another advantage is that for computing $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$, the TRS $\mathcal{R}$ is not supposed to be terminating nor even weakly normalising. This is of great interest when using TRS to encode non-terminating systems, like systems of communicating processes, for example. Note that such non-terminating TRS cannot be handled by induction proof techniques that need a well-founded ordering for proving termination of the TRS.

4.2 Sufficient Completeness

In order to prove sufficient completeness of $A(n, p)$ with $n, p \in Nat$, we first compute the intersection automaton between $Aut(1)$, computed previously, and the automaton recognising the set $IRR(\mathcal{R}_1)$, computed by the function `build_nf(R1)`.

```
[] start with term :
    simplify(Aut(1) inter build_nf(R1))

[] result term:
    Description of Aut(2) states q|0.q|1.nil final states q|1.nil
    transitions s(q|0)->q|1.s(q|0)->q|0.o->q|0.nil End of Description
```

Thus, the superset of $\mathcal{R}_1^!(\mathcal{L}(Aut(0)))$ recognised by $Aut(2)$ is $\mathcal{L}(Aut(2)) = \mathcal{L}(Aut(2), q_1) = \{s(x) \mid x \in \mathcal{L}(Aut(2), q_0)\}$, and $\mathcal{L}(Aut(2), q_0) = \{0, s(0), \ldots\}$. Thus $\mathcal{L}(Aut(2), q_0) = Nat$, and $\mathcal{L}(Aut(2)) = Nat^*$. Therefore, we trivially have $\mathcal{R}_1^!(\mathcal{L}(Aut(0))) \subseteq Nat^* \subseteq \mathcal{T}(\mathcal{C})$ and if $\mathcal{R}_1$ is weakly normalising on terms $A(n,p)$ with $n, p \in Nat$, then $\mathcal{R}_1$ is also sufficiently complete on those terms. Note that, if $Aut(2)$ is more complex, inclusion between automaton $Aut(2)$ and an automaton recognising exactly $\mathcal{T}(\mathcal{C})$ can also be verified automatically by our prototype.

4.3 Sequential Reduction Strategy

In this third example, we show that sequential reduction strategy is interesting for proving termination of programs combining different methods of termination proof. The following specification defines a function $make\_list(i,j)$, that constructs a list of naturals ($i!$, $(i+1)!$, ... , $(j-1)!$, $j!$). The module $\mathcal{R}_1$ constructs the list and the module $\mathcal{R}_2$ achieves the computation of the factorial function.

```
specification make_list2
Vars    x y z

Ops
        o:0 p:1 s:1 fact:1 plus:2 mult:2 cons:2 int:2 intlist:1 null:0
        fact_list:2 apply_fact:1

R1
        fact_list(x, y) -> apply_fact(int(x, y))
        apply_fact(null) -> null
        apply_fact(cons(x,y)) -> cons(fact(x),apply_fact(y))
        intlist(null) -> null
        intlist(cons(x, y)) -> cons(s(x), intlist(y))
        int(o,o) -> cons(o,null)
        int(o,s(y)) -> cons(o, int(s(o), s(y)))
        int(s(x),o) -> null
        int(s(x), s(y)) -> intlist(int(x, y))
        nil

R2
        p(s(x)) -> x
        mult(o, x) -> x
        mult(s(x), y) -> plus(mult(x, y), y)
        plus(x, o) -> x
        plus(x, s(y)) -> s(plus(x, y))
        fact(s(x)) -> mult(s(x), fact(p(s(x))))
        fact(o) -> s(o)
        nil

Automata
    Description of Aut(0)
    states q|0.q|1.nil
    final states q|0.nil
    transitions fact_list(q|1,q|1) -> q|0
```

```
          o -> q|1
          s(q|1) -> q|1
          nil
    End of Description
    nil
```

Note that neither termination of $\mathcal{R}_1$ nor termination $\mathcal{R}_2$ can be proven by a simplification ordering. However, termination of $\mathcal{R}_1$ can be proved by the dependency pair method [AG97b], and on the other hand, termination of $\mathcal{R}_2$ can be proved by GPO [DH95]. Instead of reconsidering the termination of the whole TRS $\mathcal{R}_1 \cup \mathcal{R}_2$, we can automatically verify that the (hierarchical) combination of those two systems is terminating under the sequential reduction strategy, for every initial term from the regular set $\mathcal{L}(Aut(0)) = \mathcal{L}(Aut(0), q_0) = \{fact\_list(n,p) \mid n, p \in Lang(Aut(0), q_1)\}$ where $\mathcal{L}(Aut(0), q_1) = \{0, s(0), \dots\} = Nat$. The query `start(Aut(0))` iterates the process described in Section 2.3, implemented with the `T_up` and `build_nf` operations, until we get a fixpoint. The result of this proof is the following:

```
[] result term:
   [true,Description of nil states q|0.q|1.q|2.q|3.q|4.nil final
   states q|4.nil transitions cons(q|2,q|3)->q|3.null->q|4.null->q|3.cons(q|2,q|3)->q|4.
   s(q|0)->q|1.o->q|0.s(q|1)->q|1.s(q|1)->q|2.s(q|0)->q|2.nil End of Description]
```

where the first field is `true` — the combination is terminating under the sequential reduction strategy — and the second field contains the automaton recognising the superset of the normal forms: lists (possibly empty) of strictly positive natural numbers, which is what was expected by definition of function $make\_list$, and which also proves sufficient completeness of $\mathcal{R}_1 \cup \mathcal{R}_2$ under sequential reduction strategy on $\mathcal{L}(Aut(0))$.

4.4 Testing co-domains of functions

This is a last example showing that computing a superset of the set of normal forms may be of great help also in debugging a functional program. Assume that you have the following program defining a function which reverses a list of elements.

```
specification reverse
Vars    x y z
Ops
    a:0 b:0 rev:1 cons:2 append:2 null:0
R1
    rev(null) -> null
    rev(cons(x, y)) -> append(rev(y), cons(x, null))
    append(null, x) -> null
    append(cons(x, y), z) -> cons(x, append(y, z))
    nil

Automata
    Description of Aut(0)
    states q|0.q|1.q|2.nil
```

```
final states q|0.nil
transitions rev(q|1) -> q|0.
        cons(q|2, q|1) -> q|1.
        null -> q|1.
        a -> q|2.
        b -> q|2.
        nil
End of Description
```

where $\mathcal{L}(Aut(0)) = \mathcal{L}(Aut(0), q_0) = \{rev(l) \mid l \in \mathcal{L}(Aut(0), q_1)\}$, $\mathcal{L}(Aut(0), q_1) = \{null, cons(x, y) \mid x \in \mathcal{L}(Aut(0), q_2), y \in \mathcal{L}(Aut(0), q_1)\}$, and $\mathcal{L}(Aut(0), q_2) = \{a, b\}$. In other words, $\mathcal{L}(Aut(0))$ is of the form $rev(l)$ where l is any flat list of a and b, possibly empty. If we compute the automaton recognising the superset of $\mathcal{R}_1^!(\mathcal{L}(Aut(0)))$, the superset of co-domain, by evaluating the query `simplify(T_up(R1) on(Aut(0)) inter build_nf(R1))`, we obtain:

```
[] result term:
   Description of Aut(1) states q|0.nil final
   states q|0.nil transitions null->q|0.nil End
   of Description
```

Thus $\mathcal{L}(Aut(1))$, the superset of $\mathcal{R}_1^!(\mathcal{L}(Aut(0)))$, is the singleton $\{null\}$, the empty list. That is clearly not what is expected from the reverse function. If you check TRS $\mathcal{R}_1$ in detail, you will notice that it is wrong: in the third rule of $\mathcal{R}_1$, the right-hand side should be x rather than $null$. The interesting remark here is that $\mathcal{R}_1$ has all usual good properties: it is terminating, confluent, and sufficiently complete on $\mathcal{L}(Aut(0))$. Note also that typing $\mathcal{R}$ would not detect any error. The main interest of the co-domain estimation is to be complementary to usual verification techniques used on TRSs: confluence, termination, sufficient completeness, and typing. After fixing the bug in $\mathcal{R}_1$, we obtain:

```
[] result term:
   Description of Aut(1) states q|0.q|1.q|2.q|3.nil final states q|3.nil
   transitions b->q|1.a->q|1.null->q|3.cons(q|1,q|0)->q|3.cons(q|1,q|0)
   ->q|2.null->q|0.cons(q|1,q|2)->q|2.cons(q|1,q|2)->q|3.nil
   End of Description
```

where $Aut(1)$ recognise any flat list of a and b, possibly empty.

5 Conclusion

We have shown in this work that the computation of regular supersets of $\mathcal{R}$-descendants and $\mathcal{R}$-normal forms using tree automata techniques can provide assistance for checking a few properties of TRSs seen as functional programs.

An important part of this work is devoted to the computation of a regular superset of the set of descendants $\mathcal{R}^*(E)$ for any left-linear TRS $\mathcal{R}$ and any regular set of terms E. The approach proposed here is based on the computation of an approximation automaton recognising a superset of $\mathcal{R}^*(E)$. This approximation seems to be sufficient for our purposes in many practical cases. Approximation of regular language is a notion that was already used in in [Jac96], but in a different way and for a different purpose. In [Jac96], Jacquemard approximates a TRS by another one for which the set of descendants is regular, whereas in our approach, we approximate the set of new states used for normalising transitions, in order to fold recursion when necessary. The set of descendants can be computed exactly thanks to the Tree Tuple Synchronised Grammars (TTSG) approach of non-regular langages proposed in [LR97]. However, this approach deals with more restricted classes of TRSs; namely linear confluent constructor systems. Moreover, in practice, efficiency of TTSGs for our purposes is not obvious.

A promising application area is the study of non-terminating TRSs encoding the behaviour of systems of communicating processes or systems of parallel processes sharing memory. In this framework, we can prove that there is no deadlock and also some general "reachability" properties: ensure mutual exclusion, ensure that a process never stops, etc. In further research, we intend to compute another regular approximation: a *subset of* $\mathcal{R}^*(E)$ in order to achieve some reachability testing in the other way: for instance to prove that a specific behaviour must occur, we may have to check that a specific pattern *does occur* in the set of $\mathcal{R}$-descendants. We also would like to get rid of the left-linear limitation in order to enlarge the class of programs to be checked, and to compute more precise approximations.

Acknowledgements

I would like to thank Hélène Kirchner, Christophe Ringeissen, Aart Middeldorp, Bernhard Gramlich, Isabelle Gnaedig for comments and discussion on this work, Florent Jacquemard, Marc Tommasi, and Gregory Kucherov for discussion on tree automata and regular tree langages.

References

[AG97a] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In M. Bidoit and M. Dauchet, editors, *Proceedings 22nd International Colloquium on Trees in Algebra and Programming, Lille (France)*, volume 1214 of *Lecture Notes in Computer Science*, pages 261–272. Springer-Verlag, 1997.

[AG97b] T. Arts and J. Giesl. Proving innermost termination automatically. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 1997.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and Tommasi. Tree automata techniques and applications. Preliminary Version, `http://l3ux02.univ-lille3.fr/tata/`, 1997.

[CDGV91] J.L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 287–298. Springer-Verlag, 1991.

[CELM96] Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In José Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[Com86] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer-Verlag, 1986.

[CR87] H. Comon and Jean-Luc Rémy. How to characterize the language of ground normal forms. Technical Report 676, INRIA-Lorraine, 1987.

[DH95] N. Dershowitz and C. Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, May 1995.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 242–248, June 1990.

[GB85] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science*, 37:123–150, 1985.

[Gen97] Thomas Genet. Proving termination of sequential reduction relation using tree automata. Technical report, Centre de Recherche en Informatique de Nancy, 1997. 97-R-091, 28 pages.

[GG97] T. Genet and I. Gnaedig. Termination proofs using gpo ordering constraints. In M. Dauchet, editor, *Proceedings 22nd International Colloquium on Trees in Algebra and Programming, Lille (France)*, volume 1214 of *Lecture Notes in Computer Science*, pages 249–260. Springer-Verlag, 1997.

[GKK+87] J. A. Goguen, Claude Kirchner, Hélène Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, July 1987. Also as internal report CRIN: 88-R-001.

[GS84] F. Gécseg and M. Steinby. *Tree automata*. Akadémiai Kiadó, Budapest, Hungary, 1984.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.

[KK90] M. Kurihara and I. Kaji. Modular term rewriting systems and the termination. *Information Processing Letters*, 34:1–4, February 1990.

[KKV95] Claude Kirchner, Hélène Kirchner, and Marian Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.

[Kli93] P. Klint. The ASF+SDF Meta-environment User's Guide. Technical report, CWI, 1993.

[KNZ87] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[KO91] M. Kurihara and A. Ohuchi. Modular term rewriting systems with shared constructors. *Journal of Information Processing of Japan*, 14(3):357–358, 1991.

[Kou85] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1985.

[LR97] S. Limet and P. Réty. E-unification by means of tree tuple synchronized grammars. In M. Dauchet, editor, *Proceedings 22nd International Colloquium on Trees in Algebra and Programming, Lille (France)*, volume 1214 of *Lecture Notes in Computer Science*, pages 429–440. Springer-Verlag, 1997.

[NW83] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, volume 145 of *Lecture Notes in Computer Science*, pages 257–268. Springer-Verlag, 1983.

[Ohl94] E. Ohlebusch. *Modular Properties of Composable Term Rewriting Systems.* PhD thesis, Universität Bielefeld, Bielefeld, 1994.

[Sal88] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *Journal of Computer and System Sciences*, 37:367–394, 1988.

A Proof of Proposition 4

Let us first recall the proposition:

Let $\mathcal{R}$ be a left-linear TRS, $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ and $B = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$ tree automata. $\mathcal{R}^*(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$ if

1. $\Delta \subseteq \Delta'$, and
2. $\forall l \rightarrow r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma \in \Sigma(\mathcal{Q}', \mathcal{X})$, $l\sigma \rightarrow^*_{\Delta'} q$ implies $r\sigma \rightarrow^*_{\Delta'} q$.

Proof By definition, any term t of $\mathcal{R}^*(\mathcal{L}(A))$ is such that $\exists s \in \mathcal{L}(A)$ s.t. $s \rightarrow^*_{\mathcal{R}} t$. By induction on the size of the derivation $s \rightarrow^*_{\mathcal{R}} t$, we prove that if $s \rightarrow^*_{\mathcal{R}} t$ and $s \rightarrow^*_{\Delta'} q$ with $q \in \mathcal{Q}_f$ then $t \rightarrow^*_{\Delta'} q$, which implies that $t \in \mathcal{L}(B)$.

1. if $t = s$ then, since $s \in \mathcal{L}(A)$, we have that $\exists q \in \mathcal{Q}_f$ s.t. $t = s \rightarrow^*_{\Delta} q$. Moreover, $\Delta \subseteq \Delta'$, hence $\exists q \in \mathcal{Q}_f$ s.t. $t \rightarrow^*_{\Delta'} q$,
2. if $s \rightarrow^+_{\mathcal{R}} t$, then $\exists s' \in \mathcal{T}(\mathcal{F})$ s.t. $s \rightarrow^*_{\mathcal{R}} s' \rightarrow_{\mathcal{R}} t$. By induction hypothesis applied to $s \rightarrow^*_{\mathcal{R}} s'$, we obtain that $\exists q \in \mathcal{Q}_f$ s.t. $s' \rightarrow^*_{\Delta'} q$. Moreover, since $s' \rightarrow_{\mathcal{R}} t$, there exists a rule $l \rightarrow r \in \mathcal{R}$, a substitution τ, and a position p in s' such that $l\tau = s'|_p$ and $t = s'[r\tau]_p$. By construction of bottom-up tree automata with normalised transitions, if $s' \rightarrow^*_{\Delta'} q$, then any subterm of s' is reducible by Δ' into a state of $\mathcal{Q}'$. Hence, since $l\tau = s'|_p$, we get that $\exists q' \in \mathcal{Q}'$ s.t. $l\tau \rightarrow^*_{\Delta'} q'$ and $s'[q']_p \rightarrow^*_{\Delta'} q$. Now, let us show that $r\tau \rightarrow^*_{\Delta'} q'$. Let $\mathcal{V}ar(l) = \{x_1, \ldots, x_k\}$. Since l is linear and $l\tau \rightarrow^*_{\Delta'} q'$, we get as before (still by construction of bottom-up tree automata) $\exists q_1, \ldots, q_k \in \mathcal{Q}'$ s.t. $x_i\tau \rightarrow^*_{\Delta'} q_i$ and $l\sigma \rightarrow^*_{\Delta'} q'$, where σ is the $\mathcal{Q}$-substitution $\{x_i \mapsto q_i \mid i = 1 \ldots k\}$. Then, since $x_i\tau \rightarrow^*_{\Delta'} q_i$ for $i = 1 \ldots k$, and $\sigma = \{x_i \mapsto q_i \mid i = 1 \ldots k\}$, we have $r\tau \rightarrow^*_{\Delta'} r\sigma$. And since $r\sigma \rightarrow^*_{\Delta'} q'$, we finally get $r\tau \rightarrow^*_{\Delta'} q'$.

□

B Matching in tree automata

In the following, a *matching problem* is a quantifier-free first order formula build on literals $\bot$, $s \trianglelefteq c$ where $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and closed by the connectives $\vee$ and $\wedge$. An empty conjunction $\bigwedge_\emptyset$ is a trivially true matching problem.

Definition 9 *Let ϕ, ϕ_1, ϕ_2 be matching problems, $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a linear term, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton. A* solution *to the matching problem ϕ is a $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that*

- *if $\phi = s \trianglelefteq c$, then $s\sigma \rightarrow^*_{\Delta} c$, or*

- *if $\phi = \phi_1 \wedge \phi_2$, then σ is a solution of ϕ_1 and a solution of ϕ_2, or*
- *if $\phi = \phi_1 \vee \phi_2$, then σ is a solution of ϕ_1 or a solution of ϕ_2.*

We assume that matching is applied on automata without epsilon-transitions. An epsilon transition is a transition of the form $q \to q'$ where q and q' are states. Any set of transition $\Delta \cup \{q \to q'\}$ can be equivalently replaced by $\Delta \cup \{c \to q' \mid c \to q \in \Delta\}$. Now let us give the matching algorithm.

Definition 10 *Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be a tree automaton, $f \in \mathcal{F}$, $ar(f) = n$, $g \in \mathcal{F}$, $ar(g) = m$, $q, q_1, \ldots, q_n \in \mathcal{Q}$, $q'_1, \ldots, q'_n \in \mathcal{Q}$, $c_1, \ldots, c_d \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $s, s_1, \ldots, s_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and ϕ_1, ϕ_2, ϕ_3 be non-empty matching problems. The matching algorithm consists in normalising any matching problem of the form $s \trianglelefteq q$ by the following set of rules.*

Decompose
$$\frac{f(s_1, \ldots, s_n) \trianglelefteq f(q_1, \ldots, q_n)}{s_1 \trianglelefteq q_1 \wedge \ldots \wedge s_n \trianglelefteq q_n}$$

Clash
$$\frac{f(s_1, \ldots, s_n) \trianglelefteq g(q'_1, \ldots, q'_m)}{\bot}$$

Configuration
$$\frac{s \trianglelefteq q}{s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d \vee \bot}$$

if $s \notin \mathcal{X}$, for all $c_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ $i = 1 \ldots d$ such that $c_i \to q \in \Delta$.

Moreover, after each application of any of these rules, matching problems are normalised by the following set of rules ξ:

$$\frac{\phi_1 \wedge (\phi_2 \vee \phi_3)}{(\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)} \qquad \frac{\phi_1 \vee \bot}{\phi_1} \qquad \frac{\phi_1 \wedge \bot}{\bot}$$

Correction, completeness and termination of the algorithm comes from the following theorem.

Theorem 3 *Given $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $q \in \mathcal{Q}$, every matching problem $s \trianglelefteq q$ has a normal form such that*

- *if it is $\bot$ then there is no $\mathcal{Q}$-substitution σ s.t. $s\sigma \to^*_{\Delta'} q$,*
- *if it is empty, then for all $\mathcal{Q}$-substitution σ, $s\sigma \to^*_{\Delta'} q$,*
- *otherwise, the normal form is a disjunction $\bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} x^i_j \trianglelefteq q^i_j$, where $x^i_j \in \mathcal{X}$ and $q^i_j \in \mathcal{Q}$, and $\sigma_1 = \{x^1_j \mapsto q^1_j \mid j = 1 \ldots n_1\}, \ldots, \sigma_k = \{x^k_j \mapsto q^k_j \mid j = 1 \ldots n_k\}$ are the only $\mathcal{Q}$-substitutions s.t. $s\sigma_i \to^*_{\Delta'} q$.*

Proof We first prove that any matching problem $s \trianglelefteq q$ has a normal form by the matching algorithm, i.e. matching algorithm is terminating on any initial problem $s \trianglelefteq q$. Rules of ξ are simply terminating. Assume that the matching algorithm is not terminating, then there exists an infinite chain of matching problems $\phi_1, \phi_2, \phi_3, \ldots$ s.t. every one is in ξ-normal form and ϕ_{i+1} is obtained by applying rule **Decompose**, **Clash**, or **Configuration** to ϕ_i and normalising by ξ. If we consider the size of terms in the left-hand side of matching problems, **Decompose** and **Clash** rules strictly decrease it, whereas **Configuration** rule preserves it. However, if we apply **Configuration** on a given matching problem $s \trianglelefteq q$, we obtain a finite disjunction of literals $s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d \vee \perp$. On every literal $s \trianglelefteq c_i$, the only rule that can be applied is **Decompose** or **Clash**. In other words, each step of **Configuration** rule, is necessarily followed by at least one step of the rule **Decompose** or **Clash**, and thus decreases the size of the left-hand side of literals. Hence the matching algorithm is terminating.

Secondly, we prove that the normal form is either $\perp$, an empty formula or of the form $\bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$, where $x_j^i \in \mathcal{X}$ and $q_j^i \in \mathcal{Q}$. Let Φ be a normal form s.t. Φ is not empty and $\Phi \neq \perp$. If Φ is not in disjunctive normal form or if there are some symbols $\perp$ in Φ then rules of ξ may apply contradicting the fact that Φ is in normal form. Thus, every matching problem Φ in ξ-normal form is of the form $\Phi = \bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} s_j^i \trianglelefteq c_j^i$ where $s_j^i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $c_j^i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. Moreover, right-hand sides of literals cannot be anything else than a state or a term of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ of depth 1. This comes from the fact that the matching process starts from a problem $s \trianglelefteq q$, and every right-hand side of every new literal is either a state, obtained by rule **Decompose**, or a left-hand side of a normalised transition, obtained by rule **Configuration**. On the other hand, left-hand side of literals are necessarily variables. Otherwise, rule **Decompose** or rule **Clash** may be applied which contradicts the fact that Φ is in normal form. Hence, we get that $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq c_j^i$ where $x_j^i \in \mathcal{X}$ and $c_j^i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and of depth at most 1. Now, note that there is no way to obtain a matching problem of the form $x \trianglelefteq s$ where $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$, since the initial matching problem is of the form $s \trianglelefteq q$, rule **Decompose** build literals of the form $s \trianglelefteq q$, and rule **Configuration** cannot be applied if the left-hand side is a variable. Finally, $\Phi = \bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$ where $x_j^i \in \mathcal{X}$ and $q_j^i \in \mathcal{Q}$.

Correctness and completeness result from the proof that, for each rule $\frac{N}{D}$, for all $\mathcal{Q}$-substitution σ, σ is solution of N if and only if σ is solution of D.

Decompose : assume that σ is a solution of the matching problem $s_1 \trianglelefteq q_1 \wedge \ldots \wedge s_n \trianglelefteq q_n$. By Definition 9, we get that $s_1\sigma \rightarrow_\Delta^* q_1$, and $\ldots$, and $s_n \rightarrow_\Delta^* q_n$. Then, we have $f(s_1, \ldots, s_n)\sigma = f(s_1\sigma, \ldots, s_n\sigma) \rightarrow_\Delta^* f(q_1, \ldots, q_n)$.

Conversely, if σ is a solution of $f(s_1, \ldots, s_n) \trianglelefteq f(q_1, \ldots, q_n)$, then we have $f(s_1, \ldots, s_n)\sigma = f(s_1\sigma, \ldots, s_n\sigma) \rightarrow_\Delta^* f(q_1, \ldots, q_n)$. By construction of the bottom-up tree automata, we get that for $i = 1 \ldots n$, we necessarily have $s_i\sigma \rightarrow_\Delta^* q_i$. Hence, σ is a solution of the matching problem $s_1 \trianglelefteq q_1 \wedge \ldots \wedge s_n \trianglelefteq q_n$.

Clash : for any $\mathcal{Q}$-substitution σ, by construction of bottom-up tree automata, we know that $f(s_1, \dots, s_n)\sigma \not\to^*_\Delta g(q'_1, \dots, q'_m)$.

Configuration : assume that σ is a solution of $s \trianglelefteq c_1 \vee \dots \vee s \trianglelefteq c_d \vee \perp$. Still by Definition 9, we get that $s\sigma \to^*_\Delta c_1$, or $\dots$, or $s\sigma \to^*_\Delta c_d$. Let l be the index s.t. $s\sigma \to^*_\Delta c_l$. Since $\forall i = 1 \dots d$, $c_i \to q \in \Delta$, we have $c_l \to q \in \Delta$, and we finally have $s\sigma \to^*_\Delta c_l \to^*_\Delta q$.

Conversely, if the only transitions of Δ leading to q are $c_1 \to q, \dots, c_d \to q$, then $s\sigma \to^*_\Delta q$ implies that there exists an index $l \in \{1 \dots d\}$ s.t. $s\sigma \to^*_\Delta c_l \to^*_\Delta q$. Finally, $s\sigma \to^*_\Delta c_l$ implies that σ is solution of the matching problem $s \trianglelefteq c_1 \vee \dots \vee s \trianglelefteq c_d$.

□

Thanks to this algorithm, for a given rule $l \to r$ and a given state q, it is possible to find every $\mathcal{Q}$-substitution σ s.t. $l\sigma \to^*_\Delta q$. During the construction of $\mathcal{T}_\mathcal{R}\!\uparrow(A)$, if $l\sigma \to^*_\Delta q$ and $r\sigma \not\to^*_\Delta q$, then it is necessary to add the transition $r\sigma \to q$ to Δ. If transition $r\sigma \to q$ is not normalised, then it has to be normalised (see Definition 2).

Example 5 *Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \to q_0, g(q_1) \to q_1, a \to q_1\}$. The language $\mathcal{L}(A) = \{f(g^*(a))\}$. Let $\mathcal{R} = \{f(g(x)) \to g(f(x))\}$. If we apply matching on $f(g(x)) \trianglelefteq q_0$, we obtain the following deductions, where the name of the applied rule is given on the right, and normalisation with simplification rules are omitted:*

$f(g(x)) \trianglelefteq q_0$	*rule* **Configuration**
$f(g(x)) \trianglelefteq f(q_1)$	*rule* **Decompose**
$g(x) \trianglelefteq q_1$	*rule* **Configuration**
$g(x) \trianglelefteq g(q_1) \ \vee \ g(x) \trianglelefteq a$	*rule* **Clash**
$g(x) \trianglelefteq g(q_1)$	*rule* **Decompose**
$x \trianglelefteq q_1$	

Let σ be the $\mathcal{Q}$-substitution $\sigma = \{x \mapsto q_1\}$. Thus, we deduced that $l\sigma = f(g(q_1)) \to^*_\Delta q_0$.

C Proof of Theorem 1

Let us recall the theorem to prove:

Every approximation automaton is complete, i.e. for all tree automata A, for all left-linear TRSs $\mathcal{R}$, and for all functions γ,

$$\mathcal{L}(\mathcal{T}_\mathcal{R}\!\uparrow(A)) \supseteq \mathcal{R}^*(\mathcal{L}(A))$$

Proof For proving $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\!\uparrow(A)) \supseteq \mathcal{R}^*\mathcal{L}(A)$, it is enough to prove that the approximation automata verifies Conditions 1 and 2 of Proposition 4, for all approximation functions γ. By Definition 7, $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ trivially verifies Condition 1. Now, to prove that $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ also verifies Condition 2 of Proposition 4, it is enough to prove that $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^*_{\Delta'} q$.

Let s' be any subterm of $r\sigma$ (possibly non-strict) and $q' \in \mathcal{Q}'$. By induction on the size of s', we show that $Norm_\alpha(s' \to q') \subseteq \Delta'$ implies that $s' \to^*_{\Delta'} q'$:

- if $s' = q'$, then we trivially have $s' \to^*_{\Delta'} q'$.
- if $s' = q'' \in \mathcal{Q}'$ s.t. $q'' \neq q'$ then, by case 2 of definition of $Norm$, we get that $Norm_\alpha(s' \to q') = \{s' \to q'\}$. Since $Norm_\alpha(s' \to q') \subseteq \Delta'$, we have $s' \to^*_{\Delta'} q'$.
- if $s' = g(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}')$, by applying case 3 of definition of $Norm$, we get that

  (a) $\{g(\alpha(t_1), \ldots, \alpha(t_m)) \to q'\} \subseteq \Delta'$, and

  (b) $\bigcup_{i=1}^n Norm_\alpha(t_i \to \alpha(t_i)) \subseteq \Delta'$,

  where $\forall i = 1 \ldots n$, $\alpha(t_i) \in \mathcal{Q}_{new} \subseteq \mathcal{Q}'$. By applying induction hypothesis to (b), we get that $\forall i = 1 \ldots n$, $t_i \to^*_{\Delta'} \alpha(t_i)$. On the other hand, (a) implies that $g(\alpha(t_1), \ldots, \alpha(t_m)) \to_{\Delta'} q'$. As a result, $g(t_1, \ldots, t_m) \to^*_{\Delta'} g(\alpha(t_1), \ldots, \alpha(t_m)) \to_{\Delta'} q'$.

Hence $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^*_{\Delta'} q$, and Condition 2 of Proposition 4 is satisfied by $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$. $\square$

D Proof of Theorem 2

Let us first recall the theorem:

Approximation automata built with ancestor approximation are finite automata.

Proof Since the approximation function γ used in the approximation does not depend on the σ parameter, in the following, we write $\gamma(l \to r, q)$ for $\gamma(l \to r, q, \sigma)$. First, note that if the arity of every symbol of $\mathcal{F}$ is finite, if $\mathcal{Q}$ is finite, and if the set of new states $\mathcal{Q}_{new}$ is finite, then $\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}_{new}$ is finite, the number of transitions that can be added to Δ' is also finite, and thus automaton $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ is finite. Since we only consider the case where $\mathcal{F}$ and $\mathcal{Q}$ are finite, to prove that $\mathcal{T}_{\mathcal{R}}\!\uparrow(A)$ is finite, it is enough to prove that $\mathcal{Q}_{new}$ is. In the particular case of the ancestor approximation, we have

(1) $\mathcal{Q}_{new} = \{\pi_i(\gamma(l \to r, q)) \mid l \to r \in \mathcal{R}, q \in \mathcal{Q}', 1 \leq i \leq \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r))\}$.

If we apply the fact that $\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}_{new}$ to (1), we get that $\mathcal{Q}_{new} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ where:

$$\begin{aligned}\mathcal{Q}_1 &= \{\pi_i(\gamma(l \to r, q)) \mid l \to r \in \mathcal{R}, q \in \mathcal{Q}, 1 \leq i \leq \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r))\}\\ \mathcal{Q}_2 &= \{\pi_i(\gamma(l \to r, q)) \mid l \to r \in \mathcal{R}, q \in \mathcal{Q}_{new}, 1 \leq i \leq \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r))\}\end{aligned}$$

Every state of $\mathcal{Q}_2$ is of the form

$$\pi_{i_1}(\gamma(l_1 \to r_1, \pi_{i_2}(\gamma(l_2 \to r_2, \ldots, \pi_{i_n}(\gamma(l_n \to r_n, q)) \ldots))))$$

where $q \in \mathcal{Q}$, $l_j \to r_j \in \mathcal{R}$, and $1 \leq i_j \leq \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r_j))$, for $j = 1 \ldots n$. On the other hand, Case 2 of Definition 8 is equivalent to:

$\forall l_1 \to r_1, l_2 \to r_2 \in \mathcal{R}, \forall q \in \mathcal{Q}', 1 \leq i \leq \mathcal{C}ard(\mathcal{P}os_{\mathcal{F}}(r_1))$:

$$\gamma(l_2 \to r_2, \pi_i(\gamma(l_1 \to r_1, q))) = \gamma(l_2 \to r_2, q).$$

Hence,

$$\gamma(l_1 \to r_1, \pi_{i_2}(\gamma(l_2 \to r_2, \ldots, \pi_{i_n}(\gamma(l_n \to r_n, q)) \ldots))) = \gamma(l_1 \to r_1, q)$$

and then,

$$\pi_{i_1}(\gamma(l_1 \to r_1, \pi_{i_2}(\gamma(l_2 \to r_2, \ldots, \pi_{i_n}(\gamma(l_n \to r_n, q)) \ldots)))) = \pi_{i_1}(\gamma(l_1 \to r_1, q))$$

Thus, $\mathcal{Q}_2 \subseteq \mathcal{Q}_1$ and $\mathcal{Q}_{new} = \mathcal{Q}_1$. Since $\mathcal{Q}$, $\mathcal{R}$, and $\mathcal{P}os_{\mathcal{F}}(r)$ are finite sets, $\mathcal{Q}_1$ is a finite set, and so is $\mathcal{Q}_{new}$. $\square$

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399