Proving Termination of Sequential Reduction Relation using Tree Automata

Thomas Genet

INRIA Lorraine & CRIN CNRS - BP 101
54602 Villers-lès-Nancy Cedex FRANCE
Phone: (+33) 3-83-59-30-18 - Fax: (+33) 3-83-27-83-19
E-mail: Thomas.Genet@loria.fr
http://www.loria.fr/equipe/protheo.html/

Abstract

The sequential reduction relation is a specific rewrite relation, rewriting terms by separate normalisations w.r.t. several systems, or modules. When considering simple combinations of systems, like disjoint or constructor sharing, termination (or weak termination) of each system implies termination of the sequential reduction relation. In the case of more complex combinations, like hierarchical or general combination, there is no general result for termination of the sequential reduction relation. We propose here a criterion for termination of the sequential reduction relation in the hierarchical (and general) case. This criterion is based on decidable approximations of sets of normal forms, computed thanks to specific tree automata techniques. Some examples of application to modular termination proof of programs are also presented.

Keywords: Term Rewriting, Termination, Modularity, Sequential Reduction Relation, Tree Automata, Approximation.

1 Introduction

In the context of rewriting logic, a Term Rewrite System (TRS for short) is a program, and the *normalisation* of a term by a TRS is the execution of the program. Thus, we can take advantage of methods and tools developed in rewriting to prove termination of programs. Unfortunately, proving termination of a TRS is undecidable [17] and automatising TRS termination proof methods is hard. Moreover, it is difficult to apply a "divide and conquer" approach to termination proof of TRSs, since termination is not modular in general[31].

Termination of the union of two TRSs is ensured only for restricted classes of disjoint [29, 26, 14, 22] or constructor sharing TRSs [24].

On the other hand, for termination of rewrite programs, we often do not need to prove *universal termination* of the TRS, i.e. termination on any argument, but only *existential termination*, i.e. termination on a set of "possible" arguments for which the program has been designed. Similarly, we do not need to prove termination of rewriting under any strategy of application of the rules, but only under a "normalising strategy" that rewrites any term to a normal form. However, existential termination and termination of rewriting under strategies are difficult to capture with orderings – as far as we know, there exists only two methods for proving termination of a TRS under a strategy: forward closures [8, 12], and dependency pairs [2, 1], both for proving innermost termination.

We propose here another approach to termination proofs of rewrite programs. We start from a restricted relation enjoying good modularity properties for termination: the *modular reduction relation* defined by Kurihara and Kaiji [21]. The modular reduction relation preserves termination and completeness for disjoint systems [21, 27] and even for constructor sharing systems [23, 28]. The intuition behind this relation is to normalise a term w.r.t. a set of TRS by separate normalisations w.r.t. to each TRS.

When considering a *hierarchical combination* of TRSs, i.e. a constructor sharing combination where some defined symbols of a TRS (the base system) may appear in the right hand side of rules of another TRS (the extension), termination is not necessarily preserved by modular reduction relation. We choose here to define the *sequential reduction relation*, which is a deterministic version of the modular reduction relation, and to study a sufficient criterion for its termination. Since this relation is a particular case of modular reduction relation, we show that it also preserves termination for disjoint and constructor sharing systems. For the hierarchical and the general combination of systems, we propose a criterion based on a schematisation of all possible derivations on $T(F)$ with R, thanks to tree automata. We propose a specific matching and rewriting technique on tree automata, schematising matching and rewriting on sets of terms. When termination cannot be proven on the whole set $T(F)$, it is also possible to start the termination proof of the sequential reduction relation from a set $G \subseteq T(F)$ of initial terms we want to prove termination on.

2 Preliminaries

We now introduce some notations and basic definitions. Comprehensive surveys can be found in [9] for term rewriting systems, in [27, 28, 16] for modular properties of term rewriting systems, in [11] for tree automata and tree language theory, and in [13] for connections between regular tree languages and term rewriting systems.

Terms, Substitutions, Rewrite systems

Let F be a finite set of ranked symbols. For $0 \leq i \leq \alpha$, F^i denotes the set of symbols of rank i, where α is a non negative integer. Let X be a countable set of variables. The set of *terms* $T(F,X)$ is inductively defined by:

1. $X \subseteq T(F,X)$, and
2. if $f \in F^n$ and $s_1, \ldots, s_n \in T(F,X)$, then $f(s_1, \ldots, s_n) \in T(F,X)$.

We write a instead of $a()$ for every $a \in F^0$. The set of positions in a term t, denoted by $\mathcal{P}os(t)$, is defined inductively by $\mathcal{P}os(x) = \{\epsilon\}$ if $x \in X$, and $\mathcal{P}os(f(s_1, \ldots, s_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(s_i)\}$ if $0 \leq n \leq \alpha$, $f \in F^n$ and $s_1, \ldots, s_n \in T(F,X)$, where ϵ denotes the empty sequence. 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. For a term t, $\mathcal{R}oot(t)$ denotes the symbol at position ϵ in t. The set of variables of a term t will be denoted by $\mathcal{V}ar(t)$. A term is linear if any variable of $\mathcal{V}ar(t)$ has exactly one occurrence in t. Terms without variables are called *ground* terms. The set of all ground terms is denoted by $T(F)$. A substitution is a mapping from X into $T(F,X)$, which can uniquely be extended into an homomorphism $\sigma : T(F,X) \mapsto T(F,X)$.

A term rewriting system (TRS for short) is a pair (F,R) where F is a signature and R is a set of *rewrite rules* $l \to r$, where $l, r \in T(F,X)$, $l \notin X$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A TRS (F,R) will also be noted R when F is clear from the context. 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 R is linear if every rewrite rule $l \to r$ of R is linear.

The relation $\to_R$ induced by R is defined as follows: for any $s, t \in T(F,X)$, $s \to_R t$ if and only if there exist a rule $l \to r$ in 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 subterm $t|_p$ is called a *redex*. The transitive (resp. reflexive transitive) closure of $\to_R$ will be denoted by $\to_R^+$ (resp. $\to_R^*$). A term s is *reducible* by R if there exists t s.t. $s \to_R t$.

A term s is in R-normal form (or is *R-irreducible*) if s is not reducible by R. A term *s has a normal form* if there exists a term t in R-normal form s.t. $s \to_R^* t$. The set of all ground terms in R-normal form will be denoted by $NF(R)$, and $s \to_R^* t$ with $t \in NF(R)$ will be denoted by $s \xrightarrow{!}_R t$. Let E be a set of ground terms, the set of R-descendants of set E is denoted by $R^*(E)$ and $R^*(E) = \{t \in T(F) \mid \exists s \in E \text{ s.t. } s \to_R^* t\}$. The set of ground R-normal forms of set E is denoted by $R^!(E)$ and $R^!(E) = \{t \in NF(R) \mid \exists s \in E \text{ s.t. } s \to_R^* t\}$.

A binary relation $\to$ on a set A is

(1) *confluent* if for every $s, t, u \in A$, $s \to^* t$ and $s \to^* u$ implies that there exists a $v \in A$ such that $t \to^* v$ and $u \to^* v$,

(2) *terminating* if there exists no infinite derivation $s_1 \to s_2 \to \ldots$ where $s_1, s_2, \ldots \in A$,

(3) *weakly terminating* if every s of A has a normal form,

(4) *existentially terminating on B*, where $B \subseteq A$, if there exists no s_1 in B starting a infinite derivation $s_1 \to s_2 \to s_3 \to \ldots$ where $s_2, s_3, \ldots \in A$,

(5) *complete* if $\rightarrow$ is confluent and terminating,
(6) *semi-complete* if $\rightarrow$ is confluent and weakly terminating.

A signature of a TRS (F, R) can be partitioned into the set of *defined symbols* $\mathcal{D} = \{\mathcal{R}oot(l) \mid l \rightarrow r \in R\}$ and the set of *constructor* $\mathcal{C} = F \setminus \mathcal{D}$. Let (F_1, R_1) and (F_2, R_2) be TRSs with 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 (F_1, R_1) and (F_2, R_2) are *disjoint* if $F_1 \cap F_2 = \emptyset$, TRSs are *constructor sharing* if $F_1 \cap F_2 = \mathcal{C}_1 \cap \mathcal{C}_2$, TRSs are *composable* if $\mathcal{C}_1 \cap \mathcal{D}_2 = \mathcal{C}_2 \cap \mathcal{D}_1 = \emptyset$ and both systems contain all rewrite rules defining a defined symbol f if f is shared, i.e. $\{l \rightarrow r \in R \mid \mathcal{R}oot(l) \in \mathcal{D}_1 \cap \mathcal{D}_2\} \subseteq R_1 \cap R_2$, and TRSs are *hierarchical* if $F_1 \cap \mathcal{D}_2 = \emptyset$ and $R_2 \subset \mathcal{T}(F_2 \setminus \mathcal{D}_1, X) \times \mathcal{T}(F_2, X)$.

Automata, Regular Tree Languages

Let F be a finite set of ranked symbols, and Q be a finite set of symbols of rank 0 called *states*. We denote by F^n the set of symbols of rank n, where $0 \leq n \leq \alpha$ and α is a non-negative integer. The set of *configurations* $\mathcal{T}(F \cup Q)$ is inductively defined by:

1. $Q \subseteq \mathcal{T}(F \cup Q)$,
2. if $f \in F^n$ where $0 \leq n \leq \alpha$ and $s_1, \ldots, s_n \in \mathcal{T}(F \cup Q)$ then $f(s_1, \ldots, s_n) \in \mathcal{T}(F \cup Q)$.

A *transition* is a rewrite rule $c \rightarrow q$, where $c \in \mathcal{T}(F \cup Q)$ and $q \in Q$. A *normalised transition* is a transition $c \rightarrow q$ where $c = f(q_1, \ldots, q_n)$ with $0 \leq n \leq \alpha$, $f \in F^n$, and $q_1, \ldots, q_n \in Q$. A bottom-up finite tree automaton (tree automaton for short) is a quadruple $A = \langle F, Q, Q_f, \Delta \rangle$, where $Q_f \subseteq Q$ and Δ is a set of normalised transitions. The rewrite relation induced by Δ will be denoted by $\rightarrow_\Delta$. The tree language recognizee by A is $\mathcal{L}(A) = \{t \in \mathcal{T}(F) \mid \exists q \in Q_f \text{ s.t. } t \rightarrow_\Delta^* q\}$. A tree language 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.

For every transition, there exists an equivalent set of normalised transitions. We define here a generalised form of normalisation using transitions with variables i.e. rewrite rules $c' \rightarrow d$ where $c' \in \mathcal{T}(F \cup Q, X)$ and $d \in Q \cup X$. For every transition $c \rightarrow q$ with $c \in \mathcal{T}(F \cup Q)$ and $q \in Q$ there exists a set of normalised transitions with variables denoted by $Norm(c \rightarrow q)$ we now define. For any term $s \in \mathcal{T}(F \cup Q)$, we denote by $\mathcal{P}os_F(s)$ the set $\{p \in \mathcal{P}os(s) \mid p \neq \epsilon \text{ and } \mathcal{R}oot(s|_p) \in F\}$. A Q-substitution is a map σ: $X \mapsto Q$, $\mathcal{D}om(\sigma)$, the domain of σ is defined as $\mathcal{D}om(\sigma) = \{x \in X \mid x\sigma \in Q\}$. Let $\Sigma(Q, X)$ be the set of Q-substitutions.

Definition 1 *Let $A = \langle F, Q, Q_f, \Delta \rangle$ be a tree automaton, $f \in F^n$, $g \in F^m$, $c, s, t_1, \ldots, t_n \in \mathcal{T}(F \cup Q)$, $\alpha_1, \ldots, \alpha_n \in Q \cup X$, $q, q' \in Q$, $p \in \mathcal{P}os_F(c)$, $x_p \in X$ for all $p \in \mathcal{P}os_F(c)$, and a natural $i > 0$. The set $Norm(s \rightarrow q)$ is defined by:*

- *if $s \in Q$ and $s = q'$, $Norm(q' \rightarrow q) = \{c \rightarrow q \mid c \rightarrow q' \in \Delta\}$, and*

- *if $s \notin Q$, $Norm(s \to q)$ is the smallest set s.t.*

  1. $\{f(\alpha_1, \ldots, \alpha_n) \to q\} \subseteq Norm(s \to q)$, *if* $s = f(t_1, \ldots, t_n)$, *where* $\forall i = 1 \ldots n$, *if* $t_i \in Q$ *then* $\alpha_i = t_i$, *and* $\alpha_i = x_i$ *otherwise, and*
  2. $\forall p \in \mathcal{P}os_F(s)$, $\{g(\alpha_1, \ldots, \alpha_m) \to x_p\} \subseteq Norm(s \to q)$, *if* $s|_p = g(t_1, \ldots, t_m)$, *where* $\forall i = 1 \ldots m$, *if* $t_i \in Q$ *then* $\alpha_i = t_i$, *and* $\alpha_i = x_{p.i}$ *otherwise.*

For all transitions with variable $c' \to d$ where $c' \in T(F \cup Q, X)$ and $d \in Q \cup X$, for all Q-substitution σ, $(c' \to d)\sigma = c'\sigma \to d\sigma$. For every transition $c \to q$, the equivalent set of normalised transitions is $Norm(c \to q)\sigma$ where σ is the Q-substitution mapping each variable of $\{x_p \mid p \in \mathcal{P}os_F(c)\}$ to a new state.

Example 1 *Let* $F = \{f, g, a\}$ *and* $A = \langle F, Q, Q_f, \Delta\rangle$, *where* $Q = \{q_0, q_1\}$, $Q_f = \{q_0\}$, *and* $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.

$Norm(q_1 \to q_0) = \{g(q_1, q_1) \to q_0, a \to q_0\}$, *and* $Norm(f(g(q_1, f(a))) \to q_0) = \{f(x_1) \to q_0, g(q_1, x_{1.2}) \to x_1, f(x_{1.2.1}) \to x_{1.2}, a \to x_{1.2.1}\}$. *A set of normalised transitions equivalent to* $f(g(q_1, f(a))) \to q_0$ *is thus* $Norm(f(g(q_1, f(a))) \to q_0)\sigma$ *where* $\sigma = \{x_1 \mapsto q_2, x_{1.2} \mapsto q_3, x_{1.2.1} \mapsto q_4\}$, *and* q_2, q_3, q_4 *are new states.*

Lemma 1 *Let* $A = \langle F, Q, Q_f, \Delta\rangle$, $s \in T(F \cup Q) \setminus Q$, $q \in Q$, *a natural* $k = Card(\mathcal{P}os_F(s))$, *and* $\mathcal{P}os_F(s) = \{p_1, \ldots, p_k\}$. *If* $\exists q_1, \ldots, q_k \in Q$ *and a Q-substitution* $\nu = \cup_{i=1}^k \{x_{p_i} \mapsto q_i\}$ *s.t.* $Norm(s \to q)\nu \subseteq \Delta$, *then* $s \to_\Delta^* q$.

Proof See Appendix A $\square$

3 Modular Reduction Relation

We briefly define the modular reduction relation and recall the main results. Definitions and theorems presented here are extracted from [21, 28, 27, 23] to which the reader is invited to refer for more details.

Instead of proposing a restricted class of disjoint systems for which termination of the general rewrite relation is modular, Kurihara and Kaji [21] chose to define a restricted rewrite relation, called modular reduction relation, for which termination is a modular property for disjoint TRSs. The modular reduction relation is denoted by $\Rightarrow$.

Definition 2 *Let* (F, R) *be a rewrite system and* $s, t \in T(F, X)$. *We write* $s \Rightarrow_R t$ *if* $s \to_R^+ t$ *and* $t \in NF(R)$.

Definition 3 *Let* $(F_1, R_1), \ldots, (F_n, R_n)$ *be TRSs. Let* $F = \bigcup_{j=1}^n F_j$. *For* $s, t \in T(F, X)$, $s \Rightarrow t$ *if and only if* $s \Rightarrow_{R_j} t$ *for some* $j \in \{1, \ldots, n\}$.

The intuition behind the relation $\Rightarrow$ is to avoid arbitrary interleaving of $R_1, \ldots, R_n$ reduction steps, and always rewrite as long as possible w.r.t. to the same TRS.

Theorem 1 *(Kurihara & Kaji [21]) Let $(F_1, R_1), \ldots, (F_n, R_n)$ be pairwise disjoint TRSs. Then the modular reduction relation $\Rightarrow$ is terminating on $T(F, X)$.*

An alternative proof of this result can be found in [27]. Note that the theorem holds even if $(F_1, R_1), \ldots, (F_n, R_n)$ are not terminating or not weakly terminating. Another result of [21] is that $\Rightarrow$ is complete if $(F_1, R_1), \ldots, (F_n, R_n)$ are at least semi-complete. Note, however, that $\Rightarrow$ is not necessarily confluent if $(F_1, R_1), \ldots, (F_n, R_n)$ are. Termination and completeness results have been extended to the constructor sharing case.

Theorem 2 *(Kurihara & Ohuchi [23]) Let $(F_1, R_1), \ldots, (F_n, R_n)$ be pairwise constructor-sharing TRSs. Then the modular reduction relation $\Rightarrow$ is terminating on $T(F, X)$.*

There is also an alternative proof of this result in [28], with the additional hypothesis that $(F_1, R_1), \ldots, (F_n, R_n)$ should be at least weakly terminating. For completeness, similarly, $\Rightarrow$ is complete if $(F_1, R_1), \ldots, (F_n, R_n)$ are pairwise constructor-sharing and at least semi-complete [28].

The modular reduction relation is an important step to bring TRS and termination proof methods closer to what is needed in the context of programming. Roughly, a combination of pairwise constructor sharing TRS corresponds, from the programmer's point of view, to a system constituted by modules sharing no function symbols but working on a common data structure. If for evaluation, the modular reduction relation is used, then the whole system is terminating if each part is. However, in programming, a module often define functions called from other modules or from a main program. This corresponds to hierarchical systems. But there is a simple counter example to the termination of $\Rightarrow$ for hierarchical systems: $R_1 = \{f(a) \to f(b)\}$ and $R_2 = \{b \to a\}$ where R_1 and R_2 are clearly terminating and hierarchical, but there exist an infinite $\Rightarrow$-derivation: $f(a) \Rightarrow_{R_1} f(b) \Rightarrow_{R_2} f(a) \Rightarrow_{R_1} \ldots$ Composable systems offer also some sharing of function symbols, however, rules defining shared symbols have to appear in both systems, which is conflicting with the "divide and conquer" approach we want to apply to termination proofs.

4 Modular Termination Criterion

Since for hierarchical terminating TRSs, the modular reduction relation is not necessarily terminating, we propose to investigate a sufficient criterion for the termination of a deterministic version of the modular reduction relation called *sequential reduction relation*, not to be confused with Sequential TRSs [18]. In the following, we will also denote sequential reduction relation by *combination of TRSs under sequential strategy*. The sequential reduction relation will be denoted by $\to_{R_1; \ldots; R_n}$.

Definition 4 *Let $(F_1, R_1), \ldots, (F_n, R_n)$ be TRSs. Let $F = \bigcup_{j=1}^{n} F_j$. For $s, t \in T(F, X)$,*

$$s \to_{R_1; \ldots; R_n} t$$

if and only if s is reducible by the TRS $(F, R_1 \cup \ldots \cup R_n)$ and $\exists s_1, \ldots, s_{n-1} \in T(F,X)$ s.t. $s \xrightarrow{!}_{R_1} s_1$ and $s_1 \xrightarrow{!}_{R_2} s_2$ and $\ldots$ and $s_{n-1} \xrightarrow{!}_{R_n} t$.

Since we are here interested in the termination of the transitive closure of the sequential reduction relation, let us first show that the sequential reduction relation preserves termination for disjoint and constructor sharing TRSs. This is a consequence of the fact that the sequential reduction relation is a particular case of modular reduction relation.

Lemma 2 *Let $(F_1, R_1), \ldots, (F_n, R_n)$ be TRSs.*

$$\rightarrow_{R_1;\ldots;R_n} \subseteq \Rightarrow^+$$

Proof Let $s, t \in T(F,X)$ s.t. $s \rightarrow_{R_1;\ldots;R_n} t$. By definition of $\rightarrow_{R_1;\ldots;R_n}$, we get that $\exists s_1, \ldots, s_{n-1} \in T(F,X)$ s.t. $s \xrightarrow{!}_{R_1} s_1$, $s_1 \xrightarrow{!}_{R_2} s_2$, $\ldots$, and $s_{n-1} \xrightarrow{!}_{R_n} t$. Let $s_0 = s$, $s_n = t$, and S be the set of indices i s.t. R_i has been applied in $s \rightarrow_{R_1;\ldots;R_n} t$, i.e. $S = \{i \mid 1 \leq i \leq n \text{ s.t. } s_{i-1} \rightarrow^+_{R_i} s_i\}$. Hence, we have $\forall i \in S$ $s_{i-1} \rightarrow^+_{R_i} s_i$ and s_i is in normal form w.r.t. to R_i. By definition of $\Rightarrow$, we get that $\forall i \in S$ $s_{i-1} \Rightarrow s_i$. Moreover $\forall i \in \{1 \ldots n\} \setminus S$ we have $s_{i-1} = s_i$. Thus, $\forall i \in \{1 \ldots n\}$ we have $s_{i-1} \Rightarrow s_i$ or $s_{i-1} = s_i$, hence, we have $s \Rightarrow^* t$. Note that the set S cannot be empty since by definition of $\rightarrow_{R_1;\ldots;R_n}$, $s \rightarrow_{R_1;\ldots;R_n} t$ implies that s is reducible by at least one R_i in $R_1, \ldots, R_n$. Hence, we have at least one step of $\Rightarrow$, and finally we obtain $s \Rightarrow^+ t$. $\Box$

Theorem 3 *Let $(F_1, R_1), \ldots, (F_n, R_n)$ be pairwise disjoint (or constructor sharing) TRSs. Then the sequential reduction relation $\rightarrow_{R_1;\ldots;R_n}$ is terminating on $T(F,X)$.*

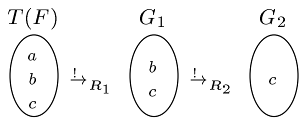
Proof Assume that there exists an infinite derivation $s_1 \rightarrow_{R_1;\ldots;R_n} s_2 \rightarrow_{R_1;\ldots;R_n} \ldots$ Then we can construct the infinite derivation $s_1 \Rightarrow^+ s_2 \Rightarrow^+ \ldots$ which contradicts termination of the modular reduction relation. $\Box$

In order to find a criterion for termination of hierarchical (or general) combination of TRSs, we construct sets of terms representing all possible derivations with $\rightarrow_{R_1;\ldots;R_n}$ on $T(F)$. Note first that if TRS R_1 and R_2 are terminating (resp. weakly terminating) on F_1 and F_2 respectively, termination (resp. weak termination) of both R_1 and R_2 on $F = F_1 \cup F_2$ is ensured by the following proposition.

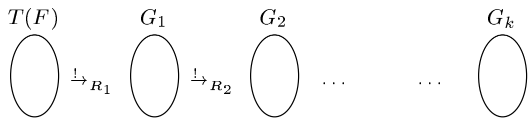
Proposition 1 *Extension of signature preserves termination [15] and weak termination (corollary of modularity of weak termination for disjoint systems [27])*

In order to sketch the criterion on a very simple example (involving only a finite set of terms), let us consider $R_1 = \{a \rightarrow b\}$ et $R_2 = \{b \rightarrow a, a \rightarrow c\}$, two terminating TRSs defined on $F_1 = \{a, b\}$ and on $F_2 = \{a, b, c\}$, respectively. Let $F = F_1 \cup F_2 = F_2$. Since $F = F_2$, R_2 is terminating on $T(F)$. From Proposition 1, we get that R_1 is terminating on $T(F)$. Termination of $\rightarrow_{R_1;R_2}$ on

$T(F)$ can be proved in the following way. By rewriting each term of $T(F)$ with R_1 into a normal form, we obtain the set $G_1 = R_1^!(T(F)) = NF(R_1)$. Then, if we rewrite each term of G_1 with R_2 into a normal form, we obtain the set $G_2 = R_2^!(G_1)$. From G_2, neither R_1 or R_2 applies, hence $\rightarrow_{R_1;R_2}$ terminates on $T(F)$.



When considering signatures with some m-ary symbols ($m > 0$), and then, possibly infinite sets of terms, we aim at generalising this approach using tree automata. However, we can replace each possibly infinite set G_i by a finite tree automaton A_i recognising G_i, only if G_i is regular.



From the tree automata theory, it is well known that $G_1 = NF(R_1)$ is a regular tree language if R_1 is left-linear [10], and a procedure for building a regular tree grammar (or a tree automaton) producing $NF(R_1)$ can be found in [5], generalised into conditional tree grammars for the case of non-left linear TRSs in [4]. For the next steps, $G_2, G_3, \ldots$, we have $G_2 = R_2^!(G_1)$, etc. However the G_i's are not necessarily regular tree languages. Let G be a regular tree language and R be a TRS. The set $R^!(G)$ is not necessarily regular even if R is a complete (or convergent) linear TRS [13].

There exists several classes of TRS for which $R^*(G)$ is regular when G is, like ground TRSs [7], right linear and monadic TRSs [30] and linear semi-monadic TRSs [6].[1] We know that $R^!(G) = R^*(G) \cap NF(R)$, and since $NF(R)$ is regular if R is left-linear, and since regular tree langages are closed by set intersection, then $R^!(G)$ is a regular tree language if G is regular and R is ground, linear monadic or linear semi-monadic.

In this paper, we deal with left-linear TRSs, and consider *approximations* of sets $G_i = R_j^!(G_{i-1})$. An approximation of a (possibly non-regular) set of normal forms G_i is a tree automaton A_i such that the regular tree language recognizee by A_i is a superset of G_i, i.e. $\mathcal{L}(A_i) \supseteq G_i$. The first automaton, A_1, is the automaton recognising the set $NF(R_1)$ and can be built from the procedure described in [5]. In the following section, we show how to deduce the approximation automaton A_i from A_{i-1} using a specific rewriting technique on automata. Then, if there exists a natural k and an automaton A_k such that every term of $\mathcal{L}(A_k) \supseteq G_k$ is irreducible by $R_1 \cup \ldots \cup R_n$, then every term of G_k is also irreducible and thus, $\rightarrow_{R_1;\ldots;R_n}$ is terminating.

[1] and also, the class of inversely-growing linear TRS as a corollary of [19]

If termination cannot be proven on $T(F)$, it is also possible to consider a subset of terms we want to prove termination on. In that case, the subset of initial terms is described by a tree automaton A_0 and A_1 is an approximation to be deduced from A_0.

5 Approximation Automaton

In this section, starting from a tree automaton A and a left-linear TRS R, we show how to build a tree automaton A' s.t. $\mathcal{L}(A') \supseteq R^!(\mathcal{L}(A))$. The method we propose consists in building a finite tree automaton $\mathcal{T}_R \uparrow (A)$, the approximation, such that $\mathcal{L}(\mathcal{T}_R \uparrow (A)) \supseteq R^*(\mathcal{L}(A))$ for any tree automaton A and any left-linear TRS R. Then, we obtain that $A' = \mathcal{T}_R \uparrow (A) \cap A_{NF(R)}$, where $A_{NF(R)}$ is a tree automaton such that $\mathcal{L}(A_{NF(R)}) = NF(R)$. The tree automaton $A_{NF(R)}$ can be constructed by one of the procedures proposed in [5, 4]. We now show how to deduce the finite approximation automaton $\mathcal{T}_R \uparrow (A)$. Let $A = \langle F, Q, Q_f, \Delta\rangle$ be a tree automaton and let X be a set of variables. We recall that a Q-substitution is a map σ: $X \mapsto Q$, and that $\Sigma(Q, X)$ is the set of Q-substitutions. The next proposition gives a sufficient condition for an automaton B to be a candidate for approximation automaton.

Proposition 2 *Let R be a left-linear TRS, $A = \langle F, Q, Q_f, \Delta\rangle$, $B = \langle F, Q', Q_f, \Delta'\rangle$ two tree automata, and σ a Q-substitution $X \mapsto Q'$.*

$$R^*(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$$

if

1. $\Delta \subseteq \Delta'$, *and*
2. $\forall l \to r \in R$, *s.t.* $r \notin X$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, $l\sigma \to^*_{\Delta'} q$ *implies* $r\sigma \to^*_{\Delta'} q$, *and*
3. $\forall l \to r \in R$, *s.t.* $r \in X$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, $l\sigma \to^*_{\Delta'} q$ *implies* $\{c \to q \mid c \to r\sigma \in \Delta'\} \subseteq \Delta'$.

Proof See Appendix B. □

Thus, for building an automaton $\mathcal{T}_R \uparrow (A)$ satisfying $R^*(\mathcal{L}(A)) \subseteq \mathcal{L}(\mathcal{T}_R \uparrow (A))$, it is enough to ensure Conditions 1, 2 and 3 of Proposition 2. For building $\mathcal{T}_R \uparrow (A)$, the algorithm we propose starts from the tree automaton A and adds to Δ the transitions necessary to ensure Condition 2 and 3: $r\sigma \to^*_\Delta q$ for any rule $l \to r$, any Q-substitution σ and any $q \in Q$ s.t. $l\sigma \to^*_\Delta q$. Since the algorithm starts from A and only adds transitions to Δ, Condition 1 is trivially verified. In order to find a Q-substitution σ and a state $q \in Q$ s.t. $l\sigma \to^*_\Delta q$, we define a *matching* algorithm.

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

Definition 5 *Let ϕ, ϕ_1, ϕ_2 be matching problems, $s \in T(F,X) \setminus X$ be a linear term, $c \in (T \cup Q)$, and $A = \langle F, Q, Q_f, \Delta\rangle$ a tree automaton. A* solution *to the matching problem ϕ is a Q-substitution $\sigma \in \Sigma(Q,X)$ 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.*

The matching algorithm is defined as follows

Definition 6 *Let $A = \langle F, Q, Q_f, \Delta\rangle$ be a tree automaton, $f \in F^n$, $g \in F^m$, $q, q_1, \ldots, q_n \in Q$, $q'_1, \ldots, q'_n \in Q$, $c_1, \ldots, c_d \in T(F \cup Q)$, $s, s_1, \ldots, s_n \in T(F,X)$ 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
$$\frac{f(s_1, \ldots, s_n) \trianglelefteq g(q'_1, \ldots, q'_m)}{\perp}$$

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

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

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

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

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

Theorem 4 *Given $s \in T(F,X)$, and $q \in Q$, every matching problem $s \trianglelefteq q$ has a normal form such that*

- *if it is $\perp$ then there is no Q-substitution σ s.t. $s\sigma \rightarrow^*_{\Delta'} q$,*
- *if it is empty, then for all Q-substitution σ, $s\sigma \rightarrow^*_{\Delta'} q$,*

- *otherwise, the normal form is a disjunction $\bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$, where $x_j^i \in X$ and $q_j^i \in Q$, and $\sigma_1 = \{x_j^1 \mapsto q_j^1 \mid j = 1 \dots n_1\}, \dots, \sigma_k = \{x_j^k \mapsto q_j^k \mid j = 1 \dots n_k\}$ are the only Q-substitutions s.t. $s\sigma_i \to_{\Delta'}^* q$.*

Proof We first prove that the algorithm is terminating, then we prove that every normal form is either $\bot$, empty, or of the form $\bigvee_{i=1}^{k} \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$. Then we prove that the algorithm is correct and complete. See Appendix C, for a complete proof. $\square$

Thanks to this algorithm, for a given rule $l \to r$ and a given state q, it is possible to find every Q-substitution σ s.t. $l\sigma \to_{\Delta}^* q$. If $r\sigma \not\to_{\Delta}^* q$, it is necessary to add the transition $r\sigma \to q$ to Δ. If transition $r\sigma \to q$ is not normalised, it has to be normalised (see Definition 1). However, adding transitions to Δ may not terminate, as we see in the following example.

Example 2 *Let $A = \langle F, Q, Q_f, \Delta \rangle$, where $F = \{f, g, a\}$, $Q = \{q_0, q_1\}$, $Q_f = \{q_0\}$ 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 $R = \{f(g(x)) \to g(f(x))\}$. 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 Q-substitution $\sigma = \{x \mapsto q_1\}$. We deduced that that $l\sigma = f(g(q_1)) \to_{\Delta}^* q_0$. To ensure condition 2, since $r\sigma = g(f(q_1)) \not\to_{\Delta}^* q_0$, we must add some normalised transitions to Δ s.t. $g(f(q_1)) \to_{\Delta}^* q_0$. We have $Norm(g(f(q_1)) \to q_0) = \{g(x_1) \to q_0, f(q_1) \to x_1\}$. If we add a new state q_2 in Q, and add $Norm(g(f(q_1)) \to q_0)\{x_1 \mapsto q_2\}$ to Δ, we obtain $\Delta = \{f(q_1) \to q_0, g(q_1) \to q_1, a \to q_1, g(q_2) \to q_0, f(q_1) \to q_2\}$. Thus $\mathcal{L}(A)$ becomes $\{f(g^*(a)), g(f(g^*(a)))\}$. Now, if we apply matching again on $f(g(x)) \trianglelefteq q_0$, we obtain the same Q-substitution $\sigma = \{x \mapsto q_1\}$. But, now we have $g(f(q_1)) \to_{\Delta}^* q_0$. Nevertheless, we can apply matching to state q_2, and we obtain:

$f(g(x)) \trianglelefteq q_2$	*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 $\sigma' = \{x \mapsto q_1\}$. With the new solution σ', we have $l\sigma' = f(g(q_1)) \to^*_\Delta q_2$, but $r\sigma' = g(f(q_1)) \not\to^*_\Delta q_2$. Hence, we need to add a new state q_3 to Q, and the transitions $\{g(q_3) \to q_2, f(q_1) \to q_3\}$ to Δ. Thus $\mathcal{L}(A)$ becomes $\{f(g^*(a)), g(f(g^*(a))), g(g(f(g^*(a))))\}$. Then we can find another solution with the matching problem $f(g(x)) \trianglelefteq q_3$, and so on.

In the previous example, the process comes into an infinite loop, because applying a rule $f(g(x)) \to g(f(x))$ on an infinite set of terms of the form $f(g^*(a))$, recursively add terms $g(f(g^*(a))), g(g(f(g^*(a))))$, and recursively add new states $q_2, q_3, \ldots$ and so on to the automaton. In order to obtain a finite automaton, in the approximation construction, we avoid such an infinite loop by adding all terms of the form $g^*(f(g^*(a)))$ at once.

The intuition is to apply all possible recursive rewriting steps at the same time, by capturing infinite recursive rewriting in a finite set of recursive automaton transitions. In our last example, $f(g(q_1)) \to^*_\Delta q_0$ implies that $g(f(q_1))$ should also rewrite to q_0 by transitions of Δ. Thus, by normalisation of $g(f(q_1)) \to q_0$, we obtain the set $\{g(q_2) \to q_0, f(q_1) \to q_2\}$ of normalised transitions. However, adding those transitions to Δ gives us a new match and a new deduction: $f(g(q_1)) \to^*_\Delta q_2$ implies that $g(f(q_1))$ should also rewrite to q_2. Instead of normalising transition $g(f(q_1)) \to q_2$ with new states, it is possible to *re-use* state q_2 (since we already have added transition $f(q_1) \to q_2$ to Δ) and normalise $g(f(q_1)) \to q_2$ into $\{g(q_2) \to q_2, f(q_1) \to q_2\}$. If we add those transitions to Δ, we obtain an automaton A such that $\mathcal{L}(A) = \{g^*(f(g^*(a)))\}$, and thus, we are done. This is one of the basic idea of the approximation automaton, which is formalised below.

Definition 7 *(Approximation Automaton) Let $A = \langle F, Q, Q_f, \Delta\rangle$ be a tree automaton, R a left-linear TRS, Q_{new} a set of new states, $\mathcal{P} = \bigcup_{l \to r \in R} \mathcal{P}os_F(r)$, and γ a function s.t. $\gamma :\ R \times (Q \cup Q_{new}) \times \Sigma(Q \cup Q_{new}, X) \times \mathcal{P} \mapsto Q_{new}$. An approximation automaton $\mathcal{T}_R \uparrow (A)$ is a tree automaton $\langle F, Q', Q_f, \Delta'\rangle$ s.t.*

- *$Q' = Q \cup Q_{new}$, and*
- *$\Delta \subseteq \Delta'$, and*
- *$\forall l \to r \in R$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, $l\sigma \to^*_{\Delta'} q$ implies $\Delta' \supseteq Norm(r\sigma \to q)\nu$, where $k = \mathcal{C}ard(\mathcal{P}os_F(r))$, $\mathcal{P}os_F(r) = \{p_1, \ldots, p_k\}$, $\nu = \{x_{p_i} \mapsto q_i\}$ for $i = 1 \ldots k$, and $q_i \in Q_{new}$ and $q_i = \gamma(l \to r, q, \sigma, p_i)$.*

By choosing specific γ function, we obtain specific approximations.

Theorem 5 *Every approximation automaton is complete, i.e. for all tree automaton A, for all left-linear TRS R, and for all function γ,*

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

Proof See Appendix D. $\square$

Although every approximation is complete, it is not necessary finite. We now focus on finite approximations. For example, a (trivially) finite approximation is the approximation where Q_{new} is a singleton $\{q\}$ and γ is a constant function, i.e. $\forall l \to r \in R, \forall q' \in Q', \forall \sigma \in \Sigma(Q', X), \forall p \in \bigcup_{l \to r \in R} \mathcal{P}os_F(r)$, we have $\gamma(l \to r, q', \sigma, p) = q$. This approximation is complete (by Theorem 5), and it is also finite since it generates at most one new state: q. However, this approximation, is not precise enough to prove termination of sequential reduction relation in interesting cases. We now propose a finite approximation, more precise, which is called the *ancestor approximation*. Informally, every new state $q' \in Q_{new}$, used to normalise a transition $t \to q$ where $q \in Q$, has a unique ancestor: q itself. On the other hand, every new state $q' \in Q_{new}$, used to normalise a transition $t \to q$ where $q \in Q_{new}$, has a unique ancestor which is the ancestor of q. In the ancestor approximation, the γ function does not depend on the σ parameter and, for any $q' \in Q_{new}$, we have $\gamma(l \to r, q', \sigma, p) = \gamma(l \to r, q, \sigma, p)$, where q is the ancestor of q'.

Definition 8 *(Ancestor approximation) The ancestor approximation is defined by:*

1. $\forall l \to r \in R$, $\forall q \in Q'$, $\forall \sigma_1, \sigma_2 \in \Sigma(Q', X)$, $\forall p \in \bigcup_{l \to r \in R} \mathcal{P}os_F(r)$, *we have* $\gamma(l \to r, q, \sigma_1, p) = \gamma(l \to r, q, \sigma_2, p)$, *and*

2. $\forall l_1 \to r_1, l_2 \to r_2 \in R$, $\forall q \in Q'$, $\forall \sigma_1, \sigma_2 \in \Sigma(Q', X)$, $\forall p_1, p_2 \in \bigcup_{l \to r \in R} \mathcal{P}os_F(r)$, *we have* $\gamma(l_1 \to r_1, \gamma(l_2 \to r_2, q, \sigma_2, p_2), \sigma_1, p_1) = \gamma(l_1 \to r_1, q, \sigma_1, p_1)$

Since for this specific approximation values of γ does not depend on the σ parameter, we will use $\gamma(l \to r, q, p)$ as a shorthand for $\gamma(l \to r, q, \sigma, p)$.

Theorem 6 *Approximation automata built with ancestor approximation are finite automata.*

Proof See Appendix E. $\square$

6 Experiments

Working on tree automaton by hand is always a heavy task. We chose, here, to implement every needed algorithm in ELAN [20, 3], in order to experiment on interesting cases as well as to check the feasibility of the method. Thus, we have implemented a library of usual algorithms on tree automaton: union, intersection, cleaning, inclusion test, as well as algorithms for building the tree automaton $A_{NF(R)}$ and $\mathcal{T}_R \uparrow (A)$ for a given automaton A and a given left-linear TRS R. Given two TRSs R_1 and R_2, the prototype is able to verify automatically the criterion for termination of the sequential reduction. The prototype is also able to handle a subset $G_0 \subseteq T(F)$ on which termination has to be proved. In that case, the subset G_0 is given as input, as a tree automaton $A(0)$.

6.1 Introducing Example

Let us propose a first simple example, in order to show how the prototype works. Assume that we have two terminating TRSs R_1 and R_2, and we want to prove termination of their combination under a sequential strategy. Let R_2 be the TRS defining the *div* function, dividing any natural by 2. Let R_1 be the TRS defining the function *mapdiv*, applying *div* to every element of a list of naturals.

```
specification mapdiv
Vars    x y z

Ops
        o:0 s:1 cons:2 null:0 mapdiv:1 div:1

R1
        mapdiv(null) -> null
        mapdiv(cons(x,y)) -> cons(div(x), mapdiv(y))
        nil
R2
        div(o) -> o
        div(s(o)) -> o
        div(s(s(x))) -> s(div(x))
        nil

end of specification
```

This example is trivial since combination of the two TRSs is simply terminating. However, our purpose here is just to give a first idea of the prototype interface. To prove that the combination of those two TRSs is terminating under sequential strategy on any term of $T(F)$, the query is just the following:

```
start
```

and the obtained result is:

```
[] result term:
   [true,Description of nil states q|0.q|1.q|2.q|3.nil final states
q|3.nil transitions div(q|1)->q|2.mapdiv(q|2)->q|2.s(q|3)->q|2.
o->q|2.div(q|1)->q|1.div(q|1)->q|3.div(q|1)->q|0.mapdiv(q|2)->
q|1.mapdiv(q|2)->q|3.mapdiv(q|2)->q|0.null->q|1.null->q|3.
null->q|0.cons(q|3,q|3)->q|1.cons(q|3,q|3)->q|3.cons(q|3,q|3)->
q|0.s(q|0)->q|1.s(q|3)->q|3.o->q|3.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 (because of approximation) of the set of normal forms.

It is also possible to prove existential termination of the combination, giving a set of terms on which we want to prove termination, thanks to a tree automaton called $A(0)$. For example, in the following call, we are interested in proving termination of the combination under sequential reduction strategy, on every term of $\mathcal{L}(A(0))$, i.e. on every term of the form: *mapdiv*(l) where l is any list of naturals.

```
start(Description of A(0)
                states q|0.q|1.q|2.nil
                final states q|0.nil
                transitions
                  mapdiv(q|1) -> q|0.
                  null-> q|1.
                  cons(q|2, q|1)-> q|1.
                  o->q|2.
                  s(q|2)-> q|2.
                  nil
      End of Description)
```

and the obtained result is:

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

where `true` still means that combination is terminating, and the second field is still the automaton recognising the superset of normal forms. In this example, the terms recognised by this automaton are lists (possibly empty) of natural numbers, which is what was expected by definition of function *mapdiv*.

6.2 Example where combination of proof methods is hard

In this second example, we show that sequential reduction relation is interesting for proving termination of programs, by combining different methods of termination proof. The following specification defines a function *make_list*(i, j), that constructs a list of naturals ($i!$, $(i+1)!$, $\ldots$, $(j-1)!$, $j!$). The module R_1 constructs the list and the module R_2 achieves the computation of the factorial function.

```
specification make_list1
Vars    x y z

Ops
        o:0 p:1 s:1 fact:1 plus:2 mult:2 cons:2 make_list:2 null:0
        list_fact:1

R1
        list_fact(null) -> null
```

```
        list_fact(cons(x,y)) -> cons(fact(s(x)),list_fact(y))
        make_list(o,o) -> cons(o,null)
        make_list(o,s(y)) -> cons(o, make_list(s(o), s(y)))
        make_list(s(x),o) -> null
        make_list(s(x), s(y)) -> list_fact(make_list(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

end of specification
```

Note neither TRSs R_1 nor R_2 are simply terminating. However, termination of R_1 can be proved by a dependency pair method [2], and on the other hand, termination of R_2 can be proved by GPO [8]. Instead of reconsidering the termination of the whole TRS $R_1 \cup R_2$, and achieve by hand a proof that is not straightforward, 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 $\mathcal{L}(A(0))$, where $A(0)$ is:

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

The transitions of this automaton rewrite every natural number represented by $\{o, s(o), s(s(o)), \dots\}$ into the state q_1. The language $\mathcal{L}(A(0))$ consists in any ground term that can be rewritten into q_0 by the transitions, i.e. any term that can be rewritten into $make\_list(q_1, q_1)$. Thus, roughly speaking, we try to prove termination of the program on terms of the form: $make\_list(i, j)$ where $i, j \in \mathbb{N}$. 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.q|5.q|6.
q|7.q|8.nil final states q|8.nil transitions null->q|5.
```

```
cons(q|2,q|3)->q|8.cons(q|2,q|3)->q|3.null->q|3.cons(q|2,q|3)->
q|5.null->q|8.o->q|4.cons(q|4,q|5)->q|8.o->q|6.null->q|7.
cons(q|6,q|7)->q|8.s(q|0)->q|1.o->q|0.s(q|1)->q|2.s(q|1)->q|1.
nil End of Description]
```

where the first field is `true` — the combination is terminating with the sequential reduction strategy — and the second field contains the automaton recognising the superset of the set of normal forms: the terms in normal form recognised by this automaton are all lists (possibly empty) of natural numbers, which is what was expected by definition of function *make_list*.

6.3 Example: Strategy

This second example concerns a combination that terminates *thanks to* the sequential strategy. Let R_1 be the TRS defining functions *plus* and *mult* optimised on a compressed representation of naturals: constructors are o, s and ds, where $ds(x)$ stands for "double-successor" and is equivalent to $s(s(x))$. Now, assume that we want to use a TRS R_2 that is not able to deal with ds constructors. A simple solution consists in adding the conversion rules transforming $ds(x)$ into $s(s(x))$, to the system $R2$. That is what is done in the following specification, where R_2 defines the "lesser or equal" predicate denoted by leq.

```
specification leq
Vars    x y

Ops
        o:0 s:1 ds:1 plus:2 mult:2 leq:2 true:0 false:0
R1
        s(s(o)) -> ds(o)
        s(s(ds(x))) -> ds(ds(x))
        plus(o, x) -> x
        plus(s(x), y) -> s(plus(x, y))
        plus(ds(x), y) -> ds(plus(x, y))
        mult(o, x) -> o
        mult(s(x), y) -> plus(mult(x, y), y)
        mult(ds(x), y) -> plus(y, plus(y, mult(x, y)))
        nil

R2
        ds(o) -> s(s(o))
        ds(x) -> s(s(x))
        leq(o, x) -> true
        leq(s(x), o) -> false
        leq(s(x), s(y)) -> leq(x, y)
        nil

end of specification
```

Note that this combination is not hierarchical but general, since s is a functional symbol in R_1 and a constructor in R_2 and ds is a functional symbol in R_2 and a constructor in R_1. Note also that every system is simply terminating, but the union is not terminating, nor even weakly terminating since there is a trivial infinite loop $s(s(o)) \to ds(o) \to s(s(o)) \to \ldots$. Similarly, the combination of those two systems under sequential strategy is not terminating on $T(F)$, since there is a similar infinite loop: $s(s(o)) \xrightarrow{!}_{R_1} ds(o) \xrightarrow{!}_{R_2} s(s(o)) \xrightarrow{!}_{R_1} \ldots$. Hence, all we can do here is prove termination on a subset of $T(F)$ that is not containing the term $s(s(o))$.

```
start(Description of A(0)
                states q|0.q|1.nil
                final states q|0.nil
                transitions
                  leq(q|1, q|1) -> q|0.
                  o -> q|1.
                  s(q|1) -> q|1.
                  mult(q|1, q|1) -> q|1.
                  plus(q|1, q|1) -> q|1.
                nil
      End of Description)
```

The language $\mathcal{L}(A(0))$ we start from is the set of terms of the form $leq(s,t)$ where s,t are any terms built on $\{o, s, plus, mult\}$. The result of this proof is the following:

```
[] result term:
   [true,Description of nil states q|0.nil final states q|0.nil
transitions true->q|0.false->q|0.nil End of Description]
```

where the field `true` ensures that the combination is terminating on $\mathcal{L}(A(0))$, under sequential reduction strategy, and the second field shows that the result of any computation, starting from any term of $\mathcal{L}(A(0))$, is necessarily $true$ or $false$.

6.4 Example: For Fun

Here is a well known combination of two TRSs [31]

```
specification f01x
Vars    x y z

Ops
        a:0 b:0 f:3 g:2
R1
        f(a,b,x) -> f(x,x,x)
        nil
```

```
R2
        g(x,y) -> x
        g(x,y) -> y
        nil
end of specification
```

Since those two systems are disjoint, we already know that their combination under sequential reduction relation is terminating, by Theorem 3. However, we can prove it automatically, in the following way:

```
start
```

And the result is:

```
[] result term:
   [true,Description of nil states q|0.q|1.q|2.q|3.q|4.q|5.q|6.nil
final states q|6.nil transitions f(q|0,q|1,q|6)->q|4.f(q|6,q|2,q|6)->
q|4.f(q|0,q|1,q|6)->q|3.f(q|6,q|2,q|6)->q|3.f(q|3,q|6,q|6)->q|4.
f(q|4,q|5,q|6)->q|4.f(q|3,q|6,q|6)->q|3.f(q|4,q|5,q|6)->q|3.b->q|4.
b->q|3.f(q|0,q|1,q|6)->q|1.f(q|6,q|2,q|6)->q|1.f(q|0,q|1,q|6)->q|2.
f(q|6,q|2,q|6)->q|2.f(q|3,q|6,q|6)->q|1.f(q|4,q|5,q|6)->q|1.
f(q|3,q|6,q|6)->q|2.f(q|4,q|5,q|6)->q|2.a->q|1.a->q|2.f(q|0,q|1,q|6)->
q|6.f(q|6,q|2,q|6)->q|6.f(q|0,q|1,q|6)->q|5.f(q|6,q|2,q|6)->q|5.
f(q|0,q|1,q|6)->q|0.f(q|6,q|2,q|6)->q|0.f(q|0,q|1,q|6)->q|1.
f(q|6,q|2,q|6)->q|1.f(q|0,q|1,q|6)->q|2.f(q|6,q|2,q|6)->q|2.
f(q|0,q|1,q|6)->q|4.f(q|6,q|2,q|6)->q|4.f(q|0,q|1,q|6)->q|3.
f(q|6,q|2,q|6)->q|3.f(q|3,q|6,q|6)->q|6.f(q|4,q|5,q|6)->q|6.
f(q|3,q|6,q|6)->q|5.f(q|4,q|5,q|6)->q|5.f(q|3,q|6,q|6)->q|0.
f(q|4,q|5,q|6)->q|0.f(q|3,q|6,q|6)->q|1.f(q|4,q|5,q|6)->q|1.
f(q|3,q|6,q|6)->q|2.f(q|4,q|5,q|6)->q|2.f(q|3,q|6,q|6)->q|4.
f(q|4,q|5,q|6)->q|4.f(q|3,q|6,q|6)->q|3.f(q|4,q|5,q|6)->q|3.b->q|6.
b->q|0.b->q|1.b->q|4.b->q|3.a->q|6.a->q|5.a->q|1.a->q|2.a->q|4.nil
End of Description]
```

7 Conclusion

This work was first concerned with termination of hierarchical combination of TRSs, which is an important topic when considering TRSs as programs. In the particular case of combination under the sequential strategy, termination is modular up to the constructor sharing case. For the hierarchical case, since termination is not modular in general, we investigated a criterion based on tree automata. Thanks to this criterion, we are able to prove modularity of termination of some combination of TRSs under the sequential strategy on $T(F)$, but also on subsets of $T(F)$ described by tree automata. Interesting subsets of $T(F)$ used in the examples of this paper are mainly: sets of well typed terms, and sets excluding terms initiating infinite loops, which are restriction difficult to take into account with orderings. Thus, further research may be to focus on termination proof of sorted rewriting with or without sort inclusion, and also on conditional rewriting.

An important part of this work is devoted to giving a regular superset of the set of descendants $R^*(E)$ for any left-linear TRS R and any regular set E. The approach proposed here is based on the computation of an approximation automaton recognising a superset of $R^*(E)$. This approximation seems to be sufficient for our criterion in many cases. Approximation of regular language is a notion that was already used in in [19], but in a different way and for a different purpose. In [19], Jacquemard approximates a TRS by another 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 create recursive transitions when necessary. The set of descendants can be computed exactly for the class of constructor TRSs thanks to the TTSG approach of non-regular langages proposed in [25]. TTSGs give a more precise criterion in the case of constructor systems. However, efficency of TTSGs for this criterion has still to be investigated in practice.

The initial purpose of this work was termination, but it seems that approximation automaton may also be interesting for studying sufficient completeness of definitions of TRSs on sets of terms. In section 6.3, the superset of normal forms: $\{true, false\}$ gives us the information that the symbol leq has disappeared, and thus the predicate leq is sufficiently defined on the initial set of terms $\mathcal{L}(A(0))$. Note also, that we have a precise information on the form of the result itself, for instance in the example of leq, the result of any call on $\mathcal{L}(A(0))$ is not any constructor term but only $true$ of $false$.

Acknowledgements

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

References

[1] T. Arts. *Automatically proving termination and innermost normalisation of term rewriting systems.* PhD thesis, Universiteit Utrecht, Utrecht, 1997.

[2] T. Arts and J. Giesl. New criteria for strong and innermost normalisation. Technical Report IBN 96/38, Technische Hochschule Darmstadt, 1996.

[3] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Marian Vittek. ELAN: A logical framework based on computational systems. In José Meseguer, editor, *Proceedings of the first inter-*

national workshop on rewriting logic, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[4] H. Comon. An effective method for handling initial algebras. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proc. 1st ALP Workshop*, pages 108–118. Akademie Verlag, 1988. Also in Springer-Verlag, LNCS, volume 343.

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

[6] J.L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In R. V. Book, editor, *Proc. 4th RTA Conf., Como (Italy)*, volume 488 of *LNCS*, pages 287–298. Springer-Verlag, 1991.

[7] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*, pages 242–248, June 1990.

[8] N. Dershowitz and C. Hoot. Natural termination. *TCS*, 142(2):179–207, May 1995.

[9] 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.

[10] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *TCS*, 37:123–150, 1985.

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

[12] O. Geupel. Overlap closure and termination of term rewriting systems. Technical Report MIP-8922, Universität Passau (Germany), 1989.

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

[14] B. Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. Technical report, Fachbereich Informatik, Universität Kaiserslautern, 1993.

[15] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computation*, 5:131–158, 1994.

[16] Bernhard Gramlich. *Termination and Confluence Properties of Structured Rewrite Systems*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, January 1996.

[17] G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, Laboria, France, 1978.

[18] G. Huet and J.-J. Lévy. Call by need computations in non-ambiguous linear term rewriting systems. Research report 359, INRIA, August 1979.

[19] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proc. 7th RTA Conf., New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.

[20] 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.

[21] M. Kurihara and I. Kaji. Modular term rewriting systems and the termination. *IPL*, 34:1–4, February 1990.

[22] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems. *Journal of IPS Japan*, 31(5):633–642, 1990.

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

[24] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *TCS*, 103(2):273–282, 1992.

[25] S. Limet and P. Réty. E-unification by means of tree tuple synchronized grammars. In M. Dauchet, editor, *Proc. 22nd CAAP Conf., Lille (France)*, volume 1214 of *LNCS*. Springer-Verlag, 1997.

[26] A. Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proc. 4th LICS Symp., Pacific Grove*, pages 396–401, 1989.

[27] A. Middeldorp. *Modular Properties of Term Rewriting Systems.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

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

[29] M. Rusinowitch. On termination of the direct sum of term rewriting systems. *IPL*, 26(2):65–70, 1987.

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

[31] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. Technical report, NTT Electrical Communications Laboratories Japan, 1987.

A Proof of lemma 1

We first recall the lemma to be proven:

Let $A = \langle F, Q, Q_f, \Delta\rangle$, $s \in T(F \cup Q) \setminus Q$, $q \in Q$, a natural $k = Card(\mathcal{P}os_F(s))$, and $\mathcal{P}os_F(s) = \{p_1, \ldots, p_k\}$. If $\exists q_1, \ldots, q_k \in Q$ and a Q-substitution $\nu = \cup_{i=1}^{k}\{x_{p_i} \mapsto q_i\}$ s.t. $Norm(s \to q)\nu \subseteq \Delta$, then $s \to_\Delta^* q$.

Proof We first prove that for any position $p \in \mathcal{P}os_F(s)$, if $\exists q_1, \ldots, q_k \in Q$ and a Q-substitution $\nu = \cup_{i=1}^{k}\{x_{p_i} \mapsto q_i\}$ s.t. $Norm(s \to q)\nu \subseteq \Delta$, then $s|_p \to_\Delta^* x_p\nu$. By induction on the size of term $s|_p$, we have:

1. if $s|_p = a \in F^0$, by case 2 of the definition of $Norm$, we have that $\{a \to x_p\} \subseteq Norm(s \to q)$. By hypothesis, $\exists q_1, \ldots, q_k \in Q$ and a Q-substitution $\nu = \cup_{i=1}^{k}\{x_{p_i} \mapsto q_i\}$, hence $x_p\nu \in Q$. On the other hand, $a\nu = a$. Finally, we have $\{a \to x_p\}\nu = \{a \to x_p\nu\} \subseteq \Delta$, hence $s|_p = a \to_\Delta^* x_p\nu$.

2. if $s|_p = g(t_1, \ldots, t_m) \in T(F \cup Q)$, by case 2 of the definition of $Norm$, we have that $\{g(\alpha_1, \ldots, \alpha_m) \to x_p\} \subseteq Norm(s \to q)$. Now, by case on t_i, we show that $t_i \to_\Delta^* \alpha_i\nu$:

   - either $t_i = q \in Q$, and $\alpha_i = t_i$. By reflexivity of $\to_\Delta^*$, we get $t_i \to_\Delta^* \alpha_i$. Since $\alpha_i \in Q$, there is no variable in α_i, and $t_i \to_\Delta^* \alpha_i\nu$,
   - or $t_i \in T(F \cup Q) \setminus Q$, and $\alpha_i = x_{p.i}$. We have $t_i \notin Q$ and $t_i = s|_{p.i}$, hence $p.i \in \mathcal{P}os_F(s)$. Hence, we can apply induction to $s|_{p.i} = t_i$, and we obtain that $s|_{p.i} = t_i \to_\Delta^* x_{p.i}\nu$. Since $\alpha_i = x_{p.i}$, this is equivalent to $t_i \to_\Delta^* \alpha_i\nu$.

   Since by hypothesis, $\exists q_1, \ldots, q_k \in Q$ and a Q-substitution $\nu = \cup_{i=1}^{k}\{x_{p_i} \mapsto q_i\}$, we have $x_p\nu \in Q$, and $\alpha_1\nu, \ldots, \alpha_m\nu \in Q$. Hence, $g(\alpha_1\nu, \ldots, \alpha_m\nu) \to x_p\nu$ is a valid transition. Moreover $g(\alpha_1\nu, \ldots, \alpha_m\nu) \to x_p\nu$ is in Δ. Finally, since $\forall i = 1 \ldots m$, $t_i \to_\Delta^* \alpha_i\nu$, we have $g(t_1, \ldots, t_m) \to_\Delta^* x_p\nu$.

Thus, we have proven that for any position $p \in \mathcal{P}os_F(s)$, we have $s|_p \to_\Delta^* x_p\nu$. Now, if $s = f(t_1, \ldots, t_n)$, thanks to Case 1 of definition of $Norm$, we get that the transition $f(\alpha_1\nu, \ldots, \alpha_n\nu) \to q$ is in Δ. Similarly to the previous cases, for every $i = 1 \ldots n$,

- either $\alpha_i = t_i \in Q$ and we trivially have $t_i \to_\Delta^* \alpha_i$, and $\alpha_i = \alpha_i\nu$,
- or $t_i \in T(F \cup Q) \setminus Q$, and $\alpha_i = x_i$. Thanks to the previous proof, we have $t_i = s|_i \to_\Delta^* x_i\nu$. Since $\alpha_i = x_i$, we get $t_i \to_\Delta^* \alpha_i\nu$.

Finally, $f(\alpha_1\nu, \ldots, \alpha_n\nu) \to q$ is in Δ, and for all $i = 1 \ldots n$, $t_i \to_\Delta^* \alpha_i\nu$, hence $s \to_\Delta^* q$. $\Box$

B Proof of Proposition 2

Let us first recall the proposition:

Let R be a left-linear TRS, $A = \langle F, Q, Q_f, \Delta \rangle$ $B = \langle F, Q', Q_f, \Delta' \rangle$ tree automata, and σ a Q-substitution $X \mapsto Q'$,

$$R^*(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$$

if

1. $\Delta \subseteq \Delta'$, and
2. $\forall l \to r \in R$, s.t. $r \notin X$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, $l\sigma \to_{\Delta'}^* q$ implies $r\sigma \to_{\Delta'}^* q$, and
3. $\forall l \to r \in R$, s.t. $r \in X$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, $l\sigma \to_{\Delta'}^* q$ implies $\{c \to q \mid c \to r\sigma \in \Delta'\} \subseteq \Delta'$.

Proof By definition of $R^*(\mathcal{L}(A))$, any term t of $R^*(\mathcal{L}(A))$ is such that $\exists s \in \mathcal{L}(A)$ s.t. $s \to_R^* t$. By induction on the size of the derivation $s \to_R^* t$, we prove that $s \to_R^* t$ implies that $\exists q \in Q_f$ s.t. $t \to_{\Delta'}^* q$, which clearly implies that $t \in \mathcal{L}(B)$.

1. if $t = s$ then, since $s \in \mathcal{L}(A)$, we have that $\exists q \in Q_f$ s.t. $t \to_\Delta^* q$. Moreover, $\Delta \subseteq \Delta'$, hence $\exists q \in Q_f$ s.t. $t \to_{\Delta'}^* q$,
2. if $s \to_R^+ t$, then $\exists s' \in T(F)$ s.t. $s \to_R^* s' \to_R t$. By induction hypothesis applied to $s \to_R^* s'$, we obtain that $\exists q \in Q_f$ s.t. $s' \to_{\Delta'}^* q$. Moreover, since $s' \to_R t$, there exists a rule $l \to r \in R$, 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 $s' \to_{\Delta'}^* q$, then any subterm of s' is reducible by Δ' into a state of Q'. Hence, since $l\tau = s'|_p$, we get that $\exists q' \in Q'$ s.t. $l\tau \to_{\Delta'}^* q'$ and $s'[q']_p \to_{\Delta'}^* q$. Now, let us show that $r\tau \to_{\Delta'}^* q'$. Let $\mathcal{V}ar(l) = \{x_1, \ldots, x_k\}$. Since l is linear and $l\tau \to_{\Delta'}^* q'$, we get as before (still by construction of bottom-up tree automata) $\exists q_1, \ldots, q_k \in Q'$ s.t. $x_i\tau \to_{\Delta'}^* q_i$ and $l\sigma \to_{\Delta'}^* q'$, where σ is the Q-substitution $\{x_i \mapsto q_i \mid i = 1 \ldots k\}$. Then by case on r:
   - if $r \notin X$ then, $l\sigma \to_{\Delta'}^* q'$ and Condition 2 of the proposition, implies that $r\sigma \to_{\Delta'}^* q'$. Since $x_i\tau \to_{\Delta'}^* q_i$ for $i = 1 \ldots k$, and σ replace every x_i by q_i in r, from $r\sigma \to_{\Delta'}^* q'$ we get that $r\tau \to_{\Delta'}^* q'$. And finally, since $s[q']_p \to_{\Delta'}^* q$, $r\tau \to_{\Delta'}^* q'$, and $t = s'[r\tau]_p$, we get that $t \to_{\Delta'}^* q$.
   - if $r = x_j \in X$, $j \in \{1, \ldots, k\}$, we have $r\tau \in T(F)$, and $x_j\tau \to_{\Delta'}^* q_j$, hence $r\tau \to_{\Delta'}^* q_j$. Let $t_1, \ldots, t_n \in T(F)$ and $r\tau = f(t_1, \ldots, t_n)$. Since $r\tau \to_{\Delta'}^* q_j$, there exists $q'_1, \ldots, q'_n \in Q'$ s.t. $t_1 \to_{\Delta'}^* q'_1, \ldots, t_n \to_{\Delta'}^* q'_n$ and $\exists f(q'_1, \ldots, q'_n) \to q_j \in \Delta'$. Since, $l\sigma \to_{\Delta'}^* q'$ and $r\sigma = q_j$, Condition 3 of the proposition implies that $f(q'_1, \ldots, q'_n) \to q' \in \Delta'$.

Finally, we have $r\tau = f(t_1, \ldots, t_n) \rightarrow^*_{\Delta'} q'$ and since $s'[q']_p \rightarrow^*_{\Delta'} q$, we get that $t = s'[r\tau]_p \rightarrow^*_{\Delta'} q$.

□

C Proof of Theorem 4

Let us first recall the theorem:

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

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

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 $s \trianglelefteq q$. Rules of ξ are simply terminating. Assume that the matching algorithm is not terminating, then there exists an infinite chain of matching problems ϕ_1, ϕ_2, ϕ_3, … 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 $x_j^i \in X$ and $q_j^i \in Q$. 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 $s_j^i \in T(F, X)$ and $c_j^i \in T(F \cup Q)$. Moreover, right-hand sides of literals cannot be anything else than a state or a term of $T(F \cup Q)$ 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 $x_j^i \in X$ and $c_j^i \in T(F \cup Q)$ 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 $s \in T(F \cup Q) \setminus Q$, 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 $x_j^i \in X$ and $q_j^i \in Q$.

Correctness and completeness result from the proof that, for each rule $\frac{N}{D}$, for all 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 5, we get that $s_1\sigma \to_\Delta^* q_1$, and $\ldots$, and $s_n \to_\Delta^* q_n$. Then, we have $f(s_1, \ldots, s_n)\sigma = f(s_1\sigma, \ldots, s_n\sigma) \to_\Delta^* f(q_1, \ldots, q_n)$.

Conversely, if σ is a solution of $f(s_1, \ldots, s_n) \trianglelefteq f(q_1, \ldots, q_n)$, then we have $f(s_1, \ldots, s_n)\sigma = f(s_1\sigma, \ldots, s_n\sigma) \to_\Delta^* f(q_1, \ldots, q_n)$. By construction of the bottom-up tree automata, we get that for $i = 1 \ldots n$, we necessarily have $s_i\sigma \to_\Delta^* q_i$. 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 Q-substitution σ, by construction of bottom-up tree automata, we know that $f(s_1, \ldots, s_n)\sigma \not\to_\Delta^* g(q'_1, \ldots, q'_m)$.

Configuration : assume that σ is a solution of $s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d \vee \perp$. Still by Definition 5, we get that $s\sigma \to_\Delta^* c_1$, or $\ldots$, or $s\sigma \to_\Delta^* c_d$. Let l be the index s.t. $s\sigma \to_\Delta^* c_l$. Since $\forall i = 1 \ldots d$, $c_i \to q \in \Delta$, 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 $c_1 \to q, \ldots, c_d \to q$, then $s\sigma \to_\Delta^* q$ implies that there exists an index $l \in \{1 \ldots d\}$ 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 $s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d$.

□

D Proof of Theorem 5

We first recall the theorem to be proven:

Every approximation automaton is complete, i.e. for all tree automaton A, for all left-linear TRS R, and for all function γ,

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

Proof For proving $\mathcal{L}(\mathcal{T}_R \uparrow (A)) \supseteq R^*\mathcal{L}(A)$, it is enough to prove that the approximation automata verifies Conditions 1, 2 and 3 of Proposition 2, for all function γ. By definition 7, $\mathcal{T}_R \uparrow (A)$ trivially verifies Condition 1. We now prove that $\mathcal{T}_R \uparrow (A)$ also verifies Condition 2 and 3 of Proposition 2.

Condition 2: let us recall condition 2 that has to be verified on $\mathcal{T}_R \uparrow (A)$.

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

Since $r \notin X$, we have $r\sigma \notin Q'$. Let $k = Card(\mathcal{P}os_F(r))$ and $\mathcal{P}os_F(r) = \{p_1, \dots, p_k\}$. Thanks to Lemma 1, we get that $r\sigma \to^*_{\Delta'} q$, if $\exists q_1, \dots, q_k \in Q'$ s.t. $Norm(r\sigma \to q)\nu \subseteq \Delta'$, where $\nu = \cup_{i=1}^k \{x_{p_i} \mapsto q_i\}$. Let us denote by $\delta(q_1, \dots, q_k)$ the set of normalised transitions $Norm(r\sigma \to q)\nu$ parametrised by $q_1, \dots, q_k$. Thus Condition 2 is implied by:

$$\begin{array}{l} \forall l \to r \in R, r \notin X, \forall q \in Q, \forall \sigma \in \Sigma(Q', X), \\ l\sigma \to^*_{\Delta'} q \implies \exists q_1, \dots, q_k \in Q \text{ s.t. } \delta(q_1, \dots, q_k) \subseteq \Delta' \end{array} \quad (1)$$

Let $\Gamma_1, \dots, \Gamma_k$ be Skolem functions. Formula (1) can be skolemised into:

$$\begin{array}{l} \forall l \to r \in R, r \notin X, \forall q \in Q', \forall \sigma \in \Sigma(Q', X), \\ l\sigma \to^*_{\Delta'} q \implies \delta(\Gamma_1(l \to r, q, \sigma), \dots, \Gamma_k(l \to r, q, \sigma,)) \subseteq \Delta' \end{array} \quad (2)$$

Thanks to properties of skolemisation, we know that (2) implies (1). On the other hand, by definition of $\mathcal{T}_R \uparrow (A)$, if $\mathcal{P}os_F(r) = \{p_1, \dots, p_k\}$, we have

$$\begin{array}{l} \forall l \to r \in R, r \notin X, \forall q \in Q', \forall \sigma \in \Sigma(Q', X), \\ l\sigma \to^*_{\Delta'} q \implies Norm(r\sigma \to q)\nu \subseteq \Delta' \text{ where } \nu = \bigcup_{i=1}^k \{x_{p_i} \mapsto q_i\} \\ \quad \text{and } q_i = \gamma(l \to r, q, \sigma, p_i) \text{ for } i = 1 \dots k \end{array} \quad (3)$$

Since $\delta(q_1, \dots, q_k)$ denotes the set $Norm(r\sigma \to q)\nu$ parametrised by $q_1, \dots, q_k$, we can rewrite (3) into

$$\begin{array}{l} \forall l \to r \in R, r \notin X, \forall q \in Q', \forall \sigma \in \Sigma(Q', X), \\ l\sigma \to^*_{\Delta'} q \implies \delta(q_1, \dots, q_k) \subseteq \Delta' \text{ where} \\ q_i = \gamma(l \to r, q, \sigma, p_i) \text{ for } i = 1 \dots k \end{array} \quad (4)$$

If we replace every q_i by its value, we obtain

$$\begin{array}{l} \forall l \to r \in R, r \notin X, \forall q \in Q', \forall \sigma \in \Sigma(Q', X), \\ l\sigma \to^*_{\Delta'} q \implies \delta(\gamma(l \to r, q, \sigma, p_1), \dots, \gamma(l \to r, q, \sigma, p_k)) \subseteq \Delta' \end{array} \quad (5)$$

Then, if for all $l \to r \in R$, for all $q \in Q'$, and for all $\sigma \in \Sigma(Q', X)$, we instantiate the Skolem functions Γ_i, $i = 1 \dots k$ by:

$$\begin{array}{l} \Gamma_1(l \to r, q, \sigma) = \gamma(l \to r, q, \sigma, p_1), \text{ and} \\ \dots \\ \Gamma_k(l \to r, q, \sigma) = \gamma(l \to r, q, \sigma, p_k) \end{array}$$

then, we trivially have (5) $\implies$ (2) for any γ function. Thus,

$$(3) \iff (5) \implies (2) \implies (1) \implies \text{Condition 2.}$$

Hence, Condition 2 is verified.

Condition 3: by definition of $\mathcal{T}_R \uparrow (A)$, $\forall l \to r \in R$ s.t. $r \in X$, $\forall q \in Q'$, $\forall \sigma \in \Sigma(Q', X)$, if $l\sigma \to^*_{\Delta'} q$ we have $\Delta' \supseteq Norm(r\sigma \to q)\nu$. If $r \in X$, $r\sigma = q' \in Q'$ and by definition 1, we have $Norm(q' \to q) = \{c \to q \mid c \to q' \in \Delta'\}$. Moreover, since there is no variable in $Norm(q' \to q)$, we have $Norm(q' \to q)\nu = Norm(q' \to q)$. Finally, $Norm(q' \to q)\nu \subseteq \Delta'$ implies that $\{c \to q \mid c \to r\sigma \in \Delta'\} \subseteq \Delta'$.

$\square$

E Proof of Theorem 6

Let us first recall the theorem:

Approximation automata built with ancestor approximation are finite automata.

Proof First, note that, if the arity of every symbol of F is finite, if Q is finite, and if the set of new states Q_{new} is finite, then Q' is finite, the number of transitions that can be added to Δ' is also finite, and thus automaton $\mathcal{T}_R \uparrow (A)$ is finite. Since we only consider the case where F and Q are finite, to prove that $\mathcal{T}_R \uparrow (A)$ is finite, it is enough to prove that Q_{new} is. In the particular case of the ancestor approximation, we have $Q' = Q \cup Q_{new}$ and $Q_{new} = \{\gamma(l \to r, q, p) \mid l \to r \in R,\, q \in Q',\, p \in \bigcup_{l \to r \in R} \mathcal{P}os_F(r)\}$. Hence, Q_{new} can be divided into two sets of states $Q_{new} = Q_1 \cup Q_2$ where

$$\begin{aligned} Q_1 =& \{\gamma(l \to r, q, p) \mid l \to r \in R,\, q \in Q,\, p \in \textstyle\bigcup_{l \to r \in R} \mathcal{P}os_F(r)\} \\ Q_2 =& \{\gamma(l \to r, q, p) \mid l \to r \in R,\, q \in Q_{new},\, p \in \textstyle\bigcup_{l \to r \in R} \mathcal{P}os_F(r)\} \end{aligned}$$

Every state of Q_2 is of the form $\gamma(l_1 \to r_1, \ldots, \gamma(l_2 \to r_2, q, p_2), \ldots, p_1)$, where $q \in Q$, $l_1 \to r_1, l_2 \to r_2 \in R$, and $p_1, p_2 \in \bigcup_{l \to r \in R} \mathcal{P}os_F(r)$. Because of case 2 of definition 8, we have

$$\gamma(l_1 \to r_1, \ldots, \gamma(l_2 \to r_2, q, p_2), \ldots, p_1) = \gamma(l_1 \to r_1, q, p_1)$$

Hence, $Q_2 \subseteq Q_1$ and thus $Q_{new} = Q_1$. Since Q, R, and $\bigcup_{l \to r \in R} \mathcal{P}os_F(r)$ are finite sets, Q_1 is a finite set, and so is Q_{new}. $\square$