A Note on the Precision of the Tree Automata Completion

Thomas Genet

A Note on the Precision of the Tree Automata Completion

Thomas Genet

IRISA/Université de Rennes 1, Campus Beaulieu, 35042 Rennes Cedex, France

Abstract. Tree Automata Completion is an algorithm over-approximating sets of terms reachable by rewriting. Precision of completion is conditioned by the, so-called, $\mathcal{R}/E$-coherence property of the initial tree automaton. This paper shows that, in the particular case of functional TRS, this restriction can easily be removed. First, we prove that there always exists an equivalent $\mathcal{R}/E$-coherent tree automaton. Second, we show how to approximate this equivalent tree automaton using completion itself and the Timbuk tool.

1 Introduction

2 Background

In this section, we introduce some definitions and concepts that will be used throughout the rest of the paper (see also [1,4]). Let $\mathcal{F}$ be a finite set of symbols, each associated with an arity function, and let $\mathcal{X}$ be a countable set of *variables*. For brevity, $f : 2$ stands for: f is a symbol of arity 2. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the set of *terms* and $\mathcal{T}(\mathcal{F})$ denotes the set of *ground terms* (terms without variables). The set of variables of a term t is denoted by $\mathcal{V}ar(t)$. A *substitution* is a function σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$, which can be uniquely extended to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A *position* p for a term t is a finite word over $\mathbb{N}$. The empty sequence λ denotes the top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by $\mathcal{P}os(t) = \{\lambda\}$ if $t \in \mathcal{X}$ or t is a constant and $\mathcal{P}os(f(t_1, \dots, t_n)) = \{\lambda\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\}$ otherwise. If $p \in \mathcal{P}os(t)$, then $t(p)$ denotes the symbol at position p in t, $t|_p$ denotes the subterm of t at position p, and $t[s]_p$ denotes the term obtained by replacing the subterm $t|_p$ at position p by the term s. The top symbol of a term is $Root(t) = t(\lambda)$

A *term rewriting system* (TRS) $\mathcal{R}$ 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 each variable occurs only once in l (resp. r). A TRS $\mathcal{R}$ is left-linear (resp. right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is left-linear (resp right-linear). A TRS $\mathcal{R}$ is said to be linear iff $\mathcal{R}$ is left-linear and right-linear. The TRS $\mathcal{R}$ induces a rewriting relation $\to_{\mathcal{R}}$ on terms as follows. Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $l \to r \in \mathcal{R}$, $s \to_{\mathcal{R}} t$ denotes that there exists a position $p \in \mathcal{P}os(s)$ and a substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. The set of term irreducible by a TRS $\mathcal{R}$ is $\textsc{Irr}(\mathcal{R})$. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\to^*_{\mathcal{R}}$. The set of $\mathcal{R}$-descendants of a set of ground terms I is $\mathcal{R}^*(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to^*_{\mathcal{R}} t\}$. Given a TRS $\mathcal{R}$, $\mathcal{F}$ can be split into two disjoint sets $\mathcal{C}$ and $\mathcal{D}$. All symbols occurring at the root position of left-hand sides of rules of $\mathcal{R}$ are in $\mathcal{D}$. $\mathcal{D}$ is the set of defined symbols of $\mathcal{R}$, $\mathcal{C}$ is the set of constructors. Terms in $\mathcal{T}(\mathcal{C})$ are called *data-terms.*

An *equation set* E is a set of *equations* $l = r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. The relation $=_E$ is the smallest congruence such that for all equations $l = r$ of E and for all substitutions σ we have $l\sigma =_E r\sigma$. The set of equivalence classes defined by $=_E$ on $\mathcal{T}(\mathcal{F})$ is noted $\mathcal{T}(\mathcal{F})/_{=_E}$. Given a TRS $\mathcal{R}$ and a set of equations E, a term $s \in \mathcal{T}(\mathcal{F})$ is rewritten modulo E into $t \in \mathcal{T}(\mathcal{F})$, denoted $s \to_{\mathcal{R}/E} t$, if there exist $s' \in \mathcal{T}(\mathcal{F})$ and $t' \in \mathcal{T}(\mathcal{F})$ such that $s =_E s' \to_{\mathcal{R}} t' =_E t$. The reflexive transitive closure $\rightarrow^*_{\mathcal{R}/E}$ of $\rightarrow_{\mathcal{R}/E}$ is defined as usual except that reflexivity is extended to terms equal modulo E, i.e. for all $s, t \in \mathcal{T}(\mathcal{F})$ if $s =_E t$ then $s \rightarrow^*_{\mathcal{R}/E} t$. The set of $\mathcal{R}$-descendants modulo E of a set of ground terms I is $\mathcal{R}^*_E(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \rightarrow^*_{\mathcal{R}/E} t\}$.

Let $\mathcal{Q}$ be a countably infinite set of symbols with arity 0, called *states*, such that $\mathcal{Q} \cap \mathcal{F} = \emptyset$. Terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ are called *configurations*. A *transition* is a rewrite rule $c \rightarrow q$, where c is a configuration and q is state. A transition is *normalized* when $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$ is of arity n, and $q_1, \ldots, q_n \in \mathcal{Q}$. An *$\epsilon$-transition* is a transition of the form $q \rightarrow q'$ where q and q' are states. A bottom-up non deterministic finite tree automaton (tree automaton for short) over the alphabet $\mathcal{F}$ is a tuple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f$ is a finite subset of $\mathcal{Q}$, Δ is a finite set of normalized transitions and ϵ-transitions. The transitive and reflexive *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ induced by the set of transitions Δ (resp. all transitions except ϵ-transitions) is denoted by $\rightarrow^*_\Delta$ (resp. $\rightarrow^{\not\epsilon *}_\Delta$). When Δ is attached to a tree automaton $\mathcal{A}$ we also note those two relations $\rightarrow_{\mathcal{A}}{}^*$ and $\rightarrow^{\not\epsilon *}_{\mathcal{A}}$, respectively. A tree automaton $\mathcal{A}$ is complete if for all $s \in \mathcal{T}(\mathcal{F})$ there exists a state q of $\mathcal{A}$ such that $s \rightarrow_{\mathcal{A}}{}^* q$. The language (resp. $\not\epsilon$-language) recognized by $\mathcal{A}$ in a state q is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow^*_{\mathcal{A}} q\}$ (resp. $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow^{\not\epsilon *}_{\mathcal{A}} q\}$). A state q of an automaton $\mathcal{A}$ is *reachable* (resp. $\not\epsilon$-reachable) if $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$ (resp. $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) \neq \emptyset$). An automaton is *reduced* (resp. $\not\epsilon$-reduced) if all its states are reachable ($\not\epsilon$-reachable). We define $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q)$. An automaton $\mathcal{A}$ is deterministic if for all ground terms $s \in \mathcal{T}(\mathcal{F})$ and all states q, q' of $\mathcal{A}$, if $s \rightarrow_{\mathcal{A}}{}^* q$ and $s \rightarrow_{\mathcal{A}}{}^* q'$ then $q = q'$. A set of transitions Δ is $\not\epsilon$-deterministic there are no two normalized transitions in Δ with the same left-hand side. A tree automaton $\mathcal{A}$ is $\not\epsilon$-deterministic if its set of transition is $\not\epsilon$-deterministic. Note that if $\mathcal{A}$ is $\not\epsilon$-deterministic then for all states q_1, q_2 of $\mathcal{A}$ such that $q_1 \neq q_2$, we have $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q_1) \cap \mathcal{L}^{\not\epsilon}(\mathcal{A}, q_2) = \emptyset$.

3 Tree Automata Completion

3.1 Tree Automata Completion General Principle

Starting from a tree automaton $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_0 \rangle$ and a left-linear TRS $\mathcal{R}$, the aim of the completion algorithm is to compute a tree automaton $\mathcal{A}^*$ such that $\mathcal{L}(\mathcal{A}^*) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ or $\mathcal{L}(\mathcal{A}^*) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. Tree automata completion successively computes tree automata $\mathcal{A}^1_{\mathcal{R}}$, $\mathcal{A}^2_{\mathcal{R}}$, …such that $\forall i \geq 0 : \mathcal{L}(\mathcal{A}^i_{\mathcal{R}}) \subseteq \mathcal{L}(\mathcal{A}^{i+1}_{\mathcal{R}})$ and if $s \in \mathcal{L}(\mathcal{A}^i_{\mathcal{R}})$, such that $s \rightarrow_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}^{i+1}_{\mathcal{R}})$, until we get an automaton $\mathcal{A}^k_{\mathcal{R}}$ with $k \in \mathbb{N}$ such that $\mathcal{L}(\mathcal{A}^k_{\mathcal{R}}) = \mathcal{L}(\mathcal{A}^{k+1}_{\mathcal{R}})$. Thus, $\mathcal{A}^k_{\mathcal{R}}$ is a fixpoint, we note it $\mathcal{A}^*$, and $\mathcal{A}^k_{\mathcal{R}}$ also verifies $\mathcal{L}(\mathcal{A}^k_{\mathcal{R}}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. To construct $\mathcal{A}^{i+1}_{\mathcal{R}}$ from $\mathcal{A}^i_{\mathcal{R}}$, we achieve a *completion step* which consists in finding *critical pairs* between $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{A}^i_{\mathcal{R}}}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \rightarrow r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \rightarrow^*_{\mathcal{A}^i_{\mathcal{R}}} q$ and $r\sigma \not\rightarrow^*_{\mathcal{A}^i_{\mathcal{R}}} q$. For $r\sigma$ to be recognized by the same state and thus model the rewriting of $l\sigma$ into $r\sigma$, it is enough to add the necessary transitions to $\mathcal{A}^i_{\mathcal{R}}$ to obtain $\mathcal{A}^{i+1}_{\mathcal{R}}$ such that $r\sigma \rightarrow^*_{\mathcal{A}^{i+1}_{\mathcal{R}}}$. In [12,10], critical pairs are joined in the following way:

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}^i_{\mathcal{R}}"'] & r\sigma \arrow[d, "\mathcal{A}^{i+1}_{\mathcal{R}}"] \\
q & q' \arrow[l, "\mathcal{A}^{i+1}_{\mathcal{R}}"]
\end{tikzcd}$$

From an algorithmic point of view, there remains two problems to solve: find all the critical pairs $(l \rightarrow r, \sigma, q)$ and find the transitions to add to $\mathcal{A}^i_{\mathcal{R}}$ to have $r\sigma \rightarrow^*_{\mathcal{A}^{i+1}_{\mathcal{R}}} q$. The first problem, called matching, can be efficiently solved using a specific algorithm [5,7]. The second problem is solved using Normalization.

Definition 1 (Normalization). *Let Δ be a set of transitions defined on a set of states $\mathcal{Q}$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$. Let $C[\,]$ be a non empty context of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$, $f \in \mathcal{F}$ of arity n, and $q, q', q_1, \ldots, q_n \in \mathcal{Q}$. The normalization function is inductively defined by:*

1. $Norm_\Delta(f(q_1, \ldots, q_n) \rightarrow q) = \{f(q_1, \ldots, q_n) \rightarrow q\}$
2. $Norm_\Delta(C[f(q_1, \ldots, q_n)] \rightarrow q) = \{f(q_1, \ldots, q_n) \rightarrow q'\} \cup Norm_{\Delta \cup \{f(q_1,\ldots,q_n) \rightarrow q'\}}(C[q'] \rightarrow q)$
   where $f(q_1, \ldots, q_n) \rightarrow q' \in \Delta$ or (q' is a new state for Δ and $\forall q'' \in Q : f(q_1, \ldots, q_n) \rightarrow q'' \notin \Delta$).

3.2 Simplification of Tree Automata by Equations

Since completion adds new transitions and new states to the automaton to join critical pairs, it may diverge. Divergence is limited or even avoided by *simplifying* the tree automaton to complete w.r.t. a set of equations E. This operation permits to over-approximate languages that cannot be recognized *exactly* using tree automata completion, *e.g.* non regular languages. The simplification operation consists in finding E-equivalent terms recognized in $\mathcal{A}$ by different states and then by merging those states together. The merging of states is performed using renaming of a state in a tree automaton.

Definition 2 (Renaming of a state in a tree automaton). *Let $\mathcal{Q}, \mathcal{Q}'$ be set of states, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, and α a function $\alpha : \mathcal{Q} \mapsto \mathcal{Q}'$. We denote by $\mathcal{A}\alpha$ the tree automaton where every occurrence of q is replaced by $\alpha(q)$ in $\mathcal{Q}$, $\mathcal{Q}_f$ and in every left and right-hand side of every transition of Δ.*

If there exists a bijection α such that $\mathcal{A} = \mathcal{A}'\alpha$ then $\mathcal{A}$ and $\mathcal{A}'$ are said to be *equivalent modulo renaming.* Now we define the *simplification relation* which merges states in a tree automaton according to an equation. Note that it is not required for equations of E to be linear.

Definition 3 (Simplification relation). *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and E be a set of equations. For $s = t \in E$, $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$, $q_a, q_b \in \mathcal{Q}$ such that $s\sigma \rightarrow_{\mathcal{A}}^{\not\epsilon *} q_a$, $t\sigma \rightarrow_{\mathcal{A}}^{\not\epsilon *} q_b$, i.e.*

$$\begin{tikzcd}
s\sigma \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & t\sigma \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_a & q_b
\end{tikzcd}$$

and $q_a \neq q_b$ then $\mathcal{A}$ can be simplified *into $\mathcal{A}' = \mathcal{A}\{q_b \mapsto q_a\}$, denoted by $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$.* ⋄

Example 1. Let $E = \{s(s(x)) = s(x), a = b\}$ and $\mathcal{A}$ be the tree automaton with $\mathcal{Q}_f = \{q_2, q_4\}$ and set of transitions $\Delta = \{a \rightarrow q_0, s(q_0) \rightarrow q_1, s(q_1) \rightarrow q_2, b \rightarrow q_3, s(q_3) \rightarrow q_4\}$. Hence $\mathcal{L}(\mathcal{A}) = \{s(s(a)), s(b)\}$. We can perform a first simplification step using the equation $s(s(x)) = s(x)$[1], because we found a substitution $\sigma = \{x \mapsto q_0\}$ such that:

$$\begin{tikzcd}
s(s(q_0)) \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & s(q_0) \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_2 & q_1
\end{tikzcd}$$

[1] Note that we could have begun to simplify $\mathcal{A}$ w.r.t. equation $a = b$, but as we will see below, this makes no difference.

Hence, $\mathcal{A} \rightsquigarrow_E \mathcal{A}' = \mathcal{A}\{q_2 \mapsto q_1\}$[2]. Thus, $\mathcal{A}'$ is the automaton with $\mathcal{Q}'_f = \{q_1, q_4\}$, $\Delta = \{a \to q_0, s(q_0) \to q_1, s(q_1) \to q_1, b \to q_3, s(q_3) \to q_4\}$, and $\mathcal{L}(\mathcal{A}') = \{s^*(s(a)), s(b)\}$. Then, we can perform a second simplification step using the equation $a = b$, because we found a substitution $\sigma' = \emptyset$ such that:

$$\begin{tikzcd}
a \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & b \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_0 & q_3
\end{tikzcd}$$

We can thus simplify $\mathcal{A}'$ in this way: $\mathcal{A}' \rightsquigarrow_E \mathcal{A}'' = \mathcal{A}'\{q_0 \mapsto q_3\}$ where $\mathcal{A}''$ is the tree automaton such that $\mathcal{Q}''_f = \mathcal{Q}'_f$ and $\Delta'' = \{a \to q_3, s(q_3) \to q_1, s(q_1) \to q_1, b \to q_3, s(q_3) \to q_4\}$. A last step of simplification can be performed using $s(s(x)) = s(x)$ and leads to the automaton $\mathcal{A}''' = \mathcal{A}''\{q_4 \mapsto q_1\}$ with $\mathcal{Q}'''_f = \{q_1\}$ and $\Delta''' = \{a \to q_3, s(q_3) \to q_1, s(q_1) \to q_1, b \to q_3\}$. Automaton $\mathcal{A}'''$ cannot be simplified, is thus a normal form of $\rightsquigarrow_E$ and $\mathcal{L}(\mathcal{A}''') = \{s^*(s(a|b))\}$.

As stated in [10] and to no one's surprise, simplification $\rightsquigarrow_E$ is a terminating relation (each step suppresses a state) and it enlarges the language recognized by a tree automaton, i.e. if $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$. Furthermore, no matter how simplification steps are performed, the obtained automata are equivalent modulo state renaming, *i.e.* $\rightsquigarrow_E$ is confluent. In the following, $\mathcal{A} \rightsquigarrow^!_E \mathcal{A}'$ denotes that $\mathcal{A} \rightsquigarrow^*_E \mathcal{A}'$ and $\mathcal{A}'$ is irreducible by $\rightsquigarrow_E$, i.e. no simplification by E can be performed on $\mathcal{A}'$.

Theorem 1 (Simplified Tree Automata [10]). *Let $\mathcal{A}, \mathcal{A}'_1, \mathcal{A}'_2$ be tree automata and E be a set of equations. If $\mathcal{A} \rightsquigarrow^!_E \mathcal{A}'_1$ and $\mathcal{A} \rightsquigarrow^!_E \mathcal{A}'_2$ then $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are equivalent modulo state renaming.*

In the following, we note $\mathcal{S}_E(\mathcal{A})$ the unique automaton (modulo renaming) $\mathcal{A}'$ such that $\mathcal{A} \rightsquigarrow^!_E \mathcal{A}'$. Now, we can define the full equational completion algorithm.

3.3 The full Completion Algorithm

Definition 4 (Automaton completion). *Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ a left-linear TRS and E a set of equations.*

- $\mathcal{A}^0_{\mathcal{R},E} = \mathcal{A}$,
- $\mathcal{A}^{n+1}_{\mathcal{R},E} = \mathcal{S}_E\left(\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})\right)$, *for $n \geq 0$ where $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})$ is a tree automaton such that all critical pairs of $\mathcal{A}^n_{\mathcal{R},E}$ are joined.*

If there exists $k \in \mathbb{N}$ such that $\mathcal{A}^k_{\mathcal{R},E} = \mathcal{A}^{k+1}_{\mathcal{R},E}$, then we note $\mathcal{A}^*_{\mathcal{R},E}$ for $\mathcal{A}^k_{\mathcal{R},E}$.

Example 2. Let $\mathcal{R} = \{f(x, y) \to f(s(x), s(y))\}$, $E = \{s(s(x)) = s(x)\}$ and $\mathcal{A}^0$ be the tree automaton with set of transitions $\Delta = \{f(q_a, q_b) \to q_0), a \to q_a, b \to q_b\}$, i.e. $\mathcal{L}(\mathcal{A}^0) = \{f(a, b)\}$. The completion ends after two completion steps on $\mathcal{A}^2_{\mathcal{R},E}$ which is a fixpoint. Completion steps are summed up in the following table. To simplify the presentation, we do not repeat the common transitions: $\mathcal{A}^i_{\mathcal{R},E}$ and $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^i)$ columns are supposed to contain all transitions of $\mathcal{A}^0, \ldots, \mathcal{A}^{i-1}_{\mathcal{R},E}$.

$\mathcal{A}^0$	$\mathcal{C}_{\mathcal{R}}(\mathcal{A}^0)$	$\mathcal{A}^1_{\mathcal{R},E}$	$\mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})$	$\mathcal{A}^2_{\mathcal{R},E}$
$f(q_a, q_b) \to q_0$	$f(q_1, q_2) \to q_3$	$f(q_1, q_2) \to q_3$	$f(q_4, q_5) \to q_6$	$f(q_1, q_2) \to q_6$
$a \to q_a$	$s(q_a) \to q_1$	$s(q_a) \to q_1$	$s(q_1) \to q_4$	$s(q_1) \to q_1$
$b \to q_b$	$s(q_b) \to q_2$	$s(q_b) \to q_2$	$s(q_2) \to q_5$	$s(q_2) \to q_2$
	$q_3 \to q_0$	$q_3 \to q_0$	$q_6 \to q_3$	

[2] or $\{q_1 \mapsto q_2\}$, any of q_1 or q_2 can be used for renaming.

The automaton $\mathcal{A}^1_{\mathcal{R},E}$ is exactly $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^0)$ since simplification by equations do not apply. Then $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})$ contains all the transitions of $\mathcal{A}^1_{\mathcal{R},E}$ plus those obtained by the resolution of the critical pair $f(q_1, q_2) \rightarrow_{\mathcal{A}}{}^* q_3$ and $f(q_1, q_2) \rightarrow_{\mathcal{R}} f(s(q_1), s(q_2))$. On $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})$ simplification using the equation $s(s(x)) = s(x)$ can be applied on following instances: $s(s(q_a)) = s(q_a)$ and $s(s(q_b)) = q_b$ which results in merging states q_4 with q_1 and q_5 with q_2. Thus, $\mathcal{A}^2_{\mathcal{R},E} = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})\{q_4 \mapsto q_1, q_5 \mapsto q_2\}$. This last automaton is a fixed point because $CP(\mathcal{R}, \mathcal{A}^2_{\mathcal{R},E}) = \emptyset$.

3.4 Three Theorems on Completion

Tree automata completion enjoys three theorems defining its main properties. The first theorem is about *termination*. It ensures that if the set of equations E satisfies some constraints then completion always terminates. The second is a *sound approximation* theorem guaranteeing that completion always compute a tree automaton recognizing an over-approximation of reachable terms. This is the lower bound theorem. The third one, a *precision* theorem, guarantees that the computed automaton recognizes no more terms than $\mathcal{R}/E$ reachable terms. This is the upper bound theorem. We first recall the lower bound theorem.

Theorem 2 (Lower bound [10]). *Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of equations. If completion terminates on $\mathcal{A}^*_{\mathcal{R},E}$ then*

$$\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

To state the upper bound theorem, we need the notion of $\mathcal{R}/E$-coherent tree automaton we now define.

Definition 5 (Coherent automaton). *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton, $\mathcal{R}$ a TRS and E a set of equations. The automaton $\mathcal{A}$ is said to be* $\mathcal{R}/E$-coherent *if $\forall q \in \mathcal{Q} : \exists s \in \mathcal{T}(\mathcal{F})$:*

$$s \rightarrow_{\mathcal{A}}^{\not\epsilon *} q \wedge [\forall t \in \mathcal{T}(\mathcal{F}) : (t \rightarrow_{\mathcal{A}}^{\not\epsilon *} q \implies s =_E t) \wedge (t \rightarrow_{\mathcal{A}}{}^* q \implies s \rightarrow^*_{\mathcal{R}/E} t)].$$

The intuition behind $\mathcal{R}/E$-coherence is the following: in the tree automaton ϵ-transitions represent rewriting steps and normalized transitions recognize E-equivalence classes. More precisely, in a $\mathcal{R}/E$-coherent tree automaton, if two terms s, t are recognized into the same state q using only normalized transitions then they belong to the same E-equivalence class. Otherwise, if at least one ϵ-transition is necessary to recognize, say, t into q then at least one step of rewriting was necessary to obtain t from s.

Example 3. Let $\mathcal{R} = \{a \rightarrow b\}$, $E = \{c = d\}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ with $\Delta = \{a \rightarrow q_0, b \rightarrow q_1, c \rightarrow q_2, d \rightarrow q_2, q_1 \rightarrow q_0\}$. The automaton $\mathcal{A}$ is $\mathcal{R}/E$-coherent because all states recognize at least one term and the state q_2 recognizes with $\rightarrow_{\Delta}^{\not\epsilon}$ two terms c and d but they satisfy $c =_E d$. Finally, $a \rightarrow^*_{\Delta} q_0$ and $b \rightarrow^*_{\Delta} q_0$ but $a \rightarrow_{\Delta}^{\not\epsilon} q_0$ and $a \rightarrow_{\mathcal{R}} b$.

Theorem 3 (Upper bound [10]). *Let $\mathcal{R}$ be a left-linear TRS, E a set of equations and $\mathcal{A}$ a $\mathcal{R}/E$-coherent tree automaton. For any $i \in \mathbb{N}$:*

$$\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A})) \qquad \textit{and} \qquad \mathcal{A}^i_{\mathcal{R},E} \textit{ is } \mathcal{R}/E\textit{-coherent}$$

The fact that those two theorems apply on different sets, namely $\mathcal{R}^*$ and $\mathcal{R}^*_E$ is important to use this technique for software verification. Indeed, if $\mathcal{R}$ models the program and E defines the approximation then it is natural to focus the theorem on the over-approximation of $\mathcal{R}$-reachable terms rather than on $\mathcal{R}/E$-reachable ones. In the context of verification, $\mathcal{R}/E$-reachable terms that are not $\mathcal{R}$-reachable are not interesting: they are necessarily part of the approximation defined by E. Computing exactly or over-approximating $\mathcal{R}/E$-reachable terms is nevertheless possible for some well identified classes of E [10].

Several termination theorems can be found in [9]. Here, we focus on the theorem which is tailored for functional programs encoded as TRS. We assume that such TRSs are left-linear, which is a common assumption on TRSs obtained from functional programs [1]. Furthermore, to exploit the types of the functional program, we now see $\mathcal{F}$ as a many-sorted signature whose set of sorts is $\mathcal{S}$. Each symbol $f \in \mathcal{F}$ is associated to a profile $f : S_1 \times \ldots \times S_k \mapsto S$ where $S_1, \ldots, S_k, S \in \mathcal{S}$ and k is the arity of f. Well-sorted terms are inductively defined as follows: $f(t_1, \ldots, t_k)$ is a well-sorted term of sort S if $f : S_1 \times \ldots \times S_k \mapsto S$ and $t_1, \ldots, t_k$ are well-sorted terms of sorts $S_1, \ldots, S_k$, respectively. We denote by $\mathcal{T}(\mathcal{F}, \mathcal{X})^{\mathcal{S}}$, $\mathcal{T}(\mathcal{F})^{\mathcal{S}}$ and $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ the set of well-sorted terms, ground terms and constructor terms, respectively. Note that we have $\mathcal{T}(\mathcal{F}, \mathcal{X})^{\mathcal{S}} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathcal{T}(\mathcal{F})^{\mathcal{S}} \subseteq \mathcal{T}(\mathcal{F})$ and $\mathcal{T}(\mathcal{C})^{\mathcal{S}} \subseteq \mathcal{T}(\mathcal{C})$. We assume that $\mathcal{R}$ and E are *sort preserving*, *i.e.* that for all rule $l \to r \in R$ and all equation $u = v \in E$, $l, r, u, v \in \mathcal{T}(\mathcal{F}, \mathcal{X})^{\mathcal{S}}$, l and r have the same sort and so do u and v. Note that well-typedness of the functional program entails the well-sortedness of $\mathcal{R}$. Finally, we restrict ourselves to sufficiently complete sorted TRSs obtained from typed functional programs and will refer to them as *functional TRSs*. Sufficient completeness in a typed framework is defined as follows: for all $s \in \mathcal{T}(\mathcal{F})^{\mathcal{S}}$ there exists a term $t \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$ such that $s \to_{\mathcal{R}}{}^{*} t$.

For tree automata completion to terminate on functional TRSs, the set E of equations has to contain $E^c_{\mathcal{C},\mathcal{S}} \cup E_{\mathcal{R}} \cup E^r$, where $E^c_{\mathcal{C},\mathcal{S}}$ is a set of contracting equations, $E_{\mathcal{R}}$ is a set where each rewrite rule of $\mathcal{R}$ is mirrored by an equation and E^r is a set of reflexivity equations ensuring $\not\epsilon$-determinism of the completed automaton.

Definition 6 (Set of reflexivity equations E^r). *For a given set of symbols $\mathcal{F}$, $E^r = \{f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \mid f \in \mathcal{F},$ and arity of f is $n\}$, where $x_1 \ldots x_n$ are pairwise distinct variables.*

Definition 7 ($E_{\mathcal{R}}$). *Let $\mathcal{R}$ be a TRS, the set of $\mathcal{R}$-equations is $E_{\mathcal{R}} = \{l = r \mid l \to r \in \mathcal{R}\}$.*

Definition 8 (Set $E^c_{\mathcal{C},\mathcal{S}}$ of contracting equations for $\mathcal{C}$ and $\mathcal{S}$). *The set of well-sorted equations $E^c_{\mathcal{C},\mathcal{S}}$ is* contracting *(for $\mathcal{C}$) if its equations are of the form $u = u|_p$ with u a linear term of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ and $p \neq \lambda$ and if the set of normal forms of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ w.r.t. the TRS $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}} = \{u \to u|_p \mid u = u|_p \in E^c_{\mathcal{C},\mathcal{S}}\}$ is finite.*

Theorem 4 (Termination [9]). *Let $\mathcal{A}_0$ be a tree automaton recognizing well-sorted terms, $\mathcal{R}$ a sufficiently complete sort-preserving left-linear TRS and E a sort-preserving set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{C},\mathcal{S}} \cup E_{\mathcal{R}}$ with $E^c_{\mathcal{C},\mathcal{S}}$ contracting then completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E terminates.*

4 The precision problem: ensuring $\mathcal{R}/E$-coherence in practice

Now we explain what is the precision problem with completion on examples of functional programs encoded as TRSs. To simplify the explanations, assume that we have a notation, inspired by regular expressions, to define regular languages of lists. Let us denote by `[a*]` the language of lists having 0 or more occurrences of symbol `a`. Let us note `[a*,b*]` the language of lists having 0 or more `a` followed by 0 or more `b`. We note `[(a|b)*]` any list with 0 or more occurrence or `a` and `b` (in any order). Now, in OCaml, we define a function deleting all the occurrences of an element in a list:

```
let rec delete x l= match l with
  | [] -> []
  | h::t -> if h=x then (delete x t) else h::(delete x t);;
```

If we encode this functional program into a functional TRS, we can use tree automata completion to answer this kind of questions: what is the set of the results obtained by applying `delete` to `a` and to any list of `a` and `b`. In particular, we would like to prove that `delete(a,[(a|b)*])=[b*]`, *i.e.* prove that `delete` deletes all occurrences of `a` in the list. This is a form of data-flow analysis: try to predict the set of all program states reachable from a language of initial function calls, *i.e.* to over-approximate $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ where $\mathcal{R}$ represents the functional program and $\mathcal{A}$ the language of initial function calls. In this setting, we automatically compute an automaton $\mathcal{A}^*_{\mathcal{R},E}$ over-approximating $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$. But we can do more. Since we are dealing with left-linear TRS, it is possible to build $\mathcal{A}_{\textsc{Irr}(\mathcal{R})}$ recognizing $\textsc{Irr}(\mathcal{R})$. Finally, since tree automata are closed under all boolean operations, we can compute an approximation of all the results of the function calls by computing the tree automaton recognizing the intersection between $\mathcal{A}^*_{\mathcal{R},E}$ and $\mathcal{A}_{\textsc{Irr}(\mathcal{R})}$. The above problem can be encoded into the following TRS, automaton $\mathcal{A}_0$ and set of equations E. This is a Timbuk[11] specification:

```
Ops delete:2 cons:2 nil:0 a:0 b:0 ite:3 true:0 false:0 eq:2
Vars X Y Z

TRS R
eq(a,a)->true        eq(a,b)->false     eq(b,a)->false     eq(b,b)->true
delete(X,nil)->nil   ite(true,X,Y)->X   ite(false,X,Y)->Y
delete(X,cons(Y,Z))->ite(eq(X,Y),delete(X,Z),cons(Y,delete(X,Z)))

Automaton A0  States qf qa qb qlb qlab qnil  Final States qf
Transitions  delete(qa,qlab)->qf  a->qa  b->qb  nil->qlab
cons(qa,qlab)->qlab    cons(qb,qlab)->qlab

Equations E
Rules
%%%%%% Ec
cons(a, cons(a,X)) = cons(a,X)
cons(b, cons(b,X)) = cons(b,X)
cons(a, cons(b, cons(a,X))) = cons(b,cons(a,X))

%%%%%% ER
eq(a,a)=true        eq(a,b)=false     eq(b,a)=false     eq(b,b)=true
delete(X,nil)=nil   ite(true,X,Y)=X   ite(false,X,Y)=Y
delete(X,cons(Y,Z))=ite(eq(X,Y),delete(X,Z),cons(Y,delete(X,Z)))

%%%%%% Er
delete(X,Y)=delete(X,Y)  cons(X,Y)=cons(X,Y)  nil=nil  a=a  b=b
```

We can check that $\mathcal{A}_0$, E and $\mathcal{R}$ respect all conditions of Theorems 2, 3 and 4, except one: the $\mathcal{R}/E$-coherence condition of $\mathcal{A}_0$. The language of $\mathcal{A}_0$ is well-sorted and E and $\mathcal{R}$ are sort preserving. The system $\mathcal{R}$ is left-linear and sufficiently complete (using Maude [3] or even Timbuk [6] itself) and E contains all the necessary equations including a set of contracting equations on $\mathcal{C}$. Thus, we are sure that completion is going to terminate and build an over-approximation of reachable terms (Theorems 2 and 4). However, since $\mathcal{A}_0$ is not $\mathcal{R}/E$-coherent, we have no precision guarantee: the completed tree automaton may recognize terms that are not $\mathcal{R}/E$-reachable. Note that $\mathcal{A}_0$ is not $\mathcal{R}/E$-coherent because there exists terms recognized by the same state in $\mathcal{A}_0$ and that are not E-equivalent. For instance, $cons(a,nil) \rightarrow^*_{\mathcal{A}_0} qlab$ and $nil \rightarrow^*_{\mathcal{A}_0} qlab$ but $cons(a,nil) \neq_E nil$. Indeed, if we run completion on $\mathcal{R}$, $\mathcal{A}_0$ and E, the completed final automaton is not precise: it recognizes lists of the form `[(a|b)*]`. However, for the kind of initial language we consider for the analysis of functional TRS, we now show that there always exists a $\mathcal{R}/E$-coherent tree automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}_0)$.

5 Getting rid of the $\mathcal{R}/E$-coherence condition for functional TRSs

For the of analysis we are interested in, the initial language $\mathcal{A}_0$ is of the form $\{C[t_1,\dots,t_n] \mid t_1,\dots,t_n \in \mathcal{L} \subseteq \mathcal{T}(\mathcal{C})^{\mathcal{S}}\}$ where $C[\,]$ is a finite context composed of defined symbols (symbols of $\mathcal{D}^{\mathcal{S}}$). More precisely, terms of the initial language can only consist of a *finite* defined context embedding *unbounded* constructor terms. For instance, in the above section, we try to compute all reachable terms from an initial language `delete(a,[(a|b)*])`, where `delete(□,□)` is the finite context defined symbol `delete`$\in \mathcal{D}^{\mathcal{S}}$ and `a` and `[(a|b)*]` are (possibly unbounded) languages of constructor terms.

Let $\mathcal{A}_0$ be an initial tree automaton fulling the above restrictions. For simplicity, we also assume that $\mathcal{A}_0$ has no epsilon transitions. Let $E^c_{\mathcal{C},\mathcal{S}}$ be a set of contracting equations for $\mathcal{C}$ and $\mathcal{S}$. We can prove that $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$ is finite.

Lemma 1 (Finiteness of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$). *If $E^c_{\mathcal{C},\mathcal{S}}$ is a set of contracting equations for $\mathcal{C}$ and $\mathcal{S}$ then $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$ is a finite set of equivalence classes.*

Proof. We make a proof by contradiction. Assume that $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$ has an infinite set of equivalence classes. By definition, of contracting equations, we know that the TRS $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}} = \{u \to u|_p \mid u = u|_p \in E^c_{\mathcal{C},\mathcal{S}}\}$ has a finite set of normal forms, say n normal forms. Furthermore, $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ is obviously terminating since each of its rule is of the form $u \to u|_p$ where $p \neq \lambda$, *i.e.* $u|_p$ is a strict subterm of u. Since $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$ has an infinite set of equivalence classes, we can find $n+1$ terms, $s_1,\dots,s_{n+1} \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$, that are all different w.r.t. $E^c_{\mathcal{C},\mathcal{S}}$, *i.e.* that are in distinct equivalence classes. Since $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ is terminating we can rewrite $s_1,\dots,s_{n+1}$ with $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ up to normal forms. Let $t_1,\dots,t_{n+1}$ be their normal forms. Since $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ has n normal forms, there are at least two indexes $i,j \in \{1,\dots,n+1\}$ such that $i \neq j$ and $t_i = t_j$. We thus have $s_i \rightarrow^*_{\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}} t_i$ and $s_j \rightarrow^*_{\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}} t_i$. By definition of rewriting and equation application, we can thus replace every rewrite step by the application of one equation. As a result, we have $s_i =_{E^c_{\mathcal{C},\mathcal{S}}} t_i$ and $s_j =_{E^c_{\mathcal{C},\mathcal{S}}} t_i$, and finally $s_i =_{E^c_{\mathcal{C},\mathcal{S}}} s_j$ which is a contradiction. □

Now, given a set E of equations defining a finite set of equivalence classes, we can build a tree automaton where each state recognizes one of them. This is a straightforward application of the Myhill-Nerode Theorem for trees [4]. For the particular case of $E^c_{\mathcal{C},\mathcal{S}}$, let us call $\mathcal{A}^E$ such an automaton. In $\mathcal{A}^E$, we set $\mathcal{Q}^E_F = \mathcal{Q}^E$, *i.e.* every state is final. Thus, in $\mathcal{A}^E$ every state of $\mathcal{Q}^E$ recognizes an equivalence class of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C},\mathcal{S}}}$ and $\mathcal{L}(\mathcal{A}^E) = \mathcal{T}(\mathcal{C})^{\mathcal{S}}$. Then by defining a simple product between $\mathcal{A}_0$ and a simple extension of $\mathcal{A}^E$, we obtain a tree automaton that is $\mathcal{R}/E$-coherent and whose language is equivalent to the language recognized by $\mathcal{A}_0$. Let us first recall what is a product automaton.

Definition 9 (Product automaton [4]). *Let $\mathcal{A} = (\mathcal{F}, Q, Q_F, \Delta_{\mathcal{A}})$ and $\mathcal{B} = (\mathcal{F}, P, P_F, \Delta_{\mathcal{B}})$ be two automata. The product automaton of $\mathcal{A}$ and $\mathcal{B}$ is $\mathcal{A} \times \mathcal{B} = (\mathcal{F}, Q \times P, Q_F \times P_F, \Delta)$ where $\Delta = \{f((q_1,p_1),\dots,(q_k,p_k)) \to (q',p') \mid f(q_1,\dots,q_k) \to q' \in \Delta_{\mathcal{A}}$ and $f(p_1,\dots,p_k) \to p' \in \Delta_{\mathcal{B}}\}$.*

Let $\mathcal{A}_0$ be an initial tree automaton such that $\mathcal{L}(\mathcal{A}_0) = \{C[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in \mathcal{L} \subseteq \mathcal{T(C)}^{\mathcal{S}}\}$ where $C[\,]$ is a finite context built with defined symbols. Let $E^c_{\mathcal{C},\mathcal{S}}$ be a set of contracting equations. Let $\mathcal{A}^E$ be the automaton defined as above using the Myhill-Nerode Theorem, $\mathcal{Q}^E$ be its set of states and Δ^E be its set of transitions. States of $\mathcal{Q}^E$ recognize terms of $\mathcal{T(C)}^{\mathcal{S}}$. To have a tree automaton $\mathcal{A}'$ recognizing terms of the form $C[t_1, \ldots, t_n]$ for all $t_1, \ldots, t_n \in \mathcal{T(C)}^{\mathcal{S}}$, we have to add some states and transitions to $\mathcal{A}^E$. Let $\mathcal{A}' = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}'_F, \Delta' \rangle$ be the tree automaton where $\mathcal{Q}'_F = \{q_f\}$ and Δ' be Δ plus the necessary transitions to have $C[q_1, \ldots, q_n] \rightarrow^*_{\mathcal{A}'} q_f$ for all states $q_1, \ldots, q_n \in \mathcal{Q}$. More precisely, $\Delta' = \Delta \cup \bigcup_{q_1,\ldots,q_n \in \mathcal{Q}} Norm_{\Delta^E}(C[q_1, \ldots, q_n] \rightarrow q_f)$.

Theorem 5. *Let $E^c_{\mathcal{C},\mathcal{S}}$ be a set of contracting equations for $\mathcal{C}$ and $\mathcal{S}$, and $E \supseteq E^c_{\mathcal{C},\mathcal{S}}$. The automaton $\mathcal{A}' \times \mathcal{A}_0$ is $\mathcal{R}/E$-coherent and $\mathcal{L}(\mathcal{A}' \times \mathcal{A}_0) = \mathcal{L}(\mathcal{A}_0)$.*

Proof. By construction of $\mathcal{A}'$, the language recognized by $\mathcal{A}'$ is $\mathcal{L}(\mathcal{A}') = \{C[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in \mathcal{T(C)}^{\mathcal{S}}\}$. By definition of the product automaton, the language recognized by $\mathcal{A}' \times \mathcal{A}_0$ is $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}_0)$. Recall that $\mathcal{L}(\mathcal{A}_0) = \{C[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in \mathcal{L} \subseteq \mathcal{T(C)}^{\mathcal{S}}\}$. Thus, $\mathcal{L}(\mathcal{A}' \times \mathcal{A}_0) = \{C[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in (\mathcal{L} \cap \mathcal{T(C)}^{\mathcal{S}})\}$. Since $\mathcal{L} \cap \mathcal{T(C)}^{\mathcal{S}} = \mathcal{L}$, $\mathcal{L}(\mathcal{A}' \times \mathcal{A}_0) = \mathcal{L}(\mathcal{A}_0)$. Now, we prove that $\mathcal{A}' \times \mathcal{A}_0$ is $\mathcal{R}/E^c_{\mathcal{C},\mathcal{S}}$-coherent. Let $\mathcal{B} = \mathcal{A}' \times \mathcal{A}_0$, q a state of $\mathcal{B}$ and $s, t \in \mathcal{T(F)}^{\mathcal{S}}$ such that $s \neq t$, $s \rightarrow^*_{\mathcal{B}} q$ and $t \rightarrow^*_{\mathcal{B}} q$. Our aim is thus to prove that $s =_{E^c_{\mathcal{C},\mathcal{S}}} t$. Since $\mathcal{B}$ is a product automaton, we know that there exists states q_1 of $\mathcal{A}'$ and q_2 of $\mathcal{A}_0$ such that $q = (q_1, q_2)$. Thus $s \rightarrow^*_{\mathcal{B}} (q_1, q_2)$ and $t \rightarrow^*_{\mathcal{B}} (q_1, q_2)$. By definition of the product automaton, we know that we necessarily have $s \rightarrow^*_{\mathcal{A}'} q_1$ and $t \rightarrow^*_{\mathcal{A}'} q_1$. Recall that $\mathcal{A}'$ is obtained by adding states and transitions to $\mathcal{A}^E$. If q_1 is a state of $\mathcal{A}^E$ then, by definition of $\mathcal{A}^E$, we know that terms of the same equivalence class w.r.t. $E^c_{\mathcal{C},\mathcal{S}}$ are recognized by the same state. This implies that $s =_{E^c_{\mathcal{C},\mathcal{S}}} t$. If q_1 is not a state of $\mathcal{A}^E$ this means that it is used to recognize a subterm of the initial context built with defined symbols: $C[q'_1, \ldots, q'_n]$ where $q'_1, \ldots, q'_n$ are states of $\mathcal{A}^E$. Hence, s (resp. t) is a terms of the form $C[s_1, \ldots, s_n]|_p$ (resp. $C[t_1, \ldots, t_n]|_p$) for a position p. Thus they can only differ because of terms $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$. However, $s_i \rightarrow^*_{\mathcal{A}^E} q_i$ and $t_i \rightarrow^*_{\mathcal{A}^E} q_i$ for all $i = 1 \ldots n$. As above, since we know that in $\mathcal{A}^E$ terms of the same equivalence class w.r.t. $E^c_{\mathcal{C},\mathcal{S}}$ are recognized by the same state, we get that $s_i =_{E^c_{\mathcal{C},\mathcal{S}}} t_i$ for all $i = 1 \ldots n$. As a result $s =_{E^c_{\mathcal{C},\mathcal{S}}} t$ and $\mathcal{B} = \mathcal{A}' \times \mathcal{A}_0$ is $\mathcal{R}/E^c_{\mathcal{C},\mathcal{S}}$-coherent. Finally, since $E \supseteq E^c_{\mathcal{C},\mathcal{S}}$, we know that for all terms s, t such that $s \rightarrow^*_{\mathcal{B}} q$ and $t \rightarrow^*_{\mathcal{B}} q$ then $s =_{E^c_{\mathcal{C},\mathcal{S}}} t$ which entails that $s =_E t$. The automaton $\mathcal{B} = \mathcal{A}' \times \mathcal{A}_0$ is thus $\mathcal{R}/E$-coherent.

Starting from an automaton $\mathcal{A}_0$ and a set of equations including a set of contracting equations $E^c_{\mathcal{C}}$, we just proved that there always exists a $\mathcal{R}/E$-coherent automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}_0)$. This proof relies on the construction of $\mathcal{A}^E$ from equations of $E^c_{\mathcal{C},\mathcal{S}}$. As far as we know there is no known algorithm to do such a construction. We conjecture that it should be possible to adapt a congruence closure algorithm like [2] to this particular setting. We can enumerate terms t of $\mathcal{T(C)}^{\mathcal{S}}$, starting from constants and iteratively raising the height of considered terms. For each term, we can run an algorithm close to the congruence closure algorithm with $\overrightarrow{E^c_{\mathcal{C}}}$ to determine in which class t belongs. Since, $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ has a finite set of normal forms, this is guaranteed to terminate. Note that $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ may not be confluent and, thus, normal forms may not exactly coincide with equivalence classes of $E^c_{\mathcal{C},\mathcal{S}}$. However, we are sure to obtain a finite automaton anyway, which is the objective. Iteratively running this algorithm on terms of $\mathcal{T(C)}^{\mathcal{S}}$, we incrementally build a tree automaton where each class corresponds to a distinct state of the automaton. After adding one term, we can check whether the current automaton recognizes all $\mathcal{T(C)}^{\mathcal{S}}$ or not. Since we know that the set of equivalence classes of $\mathcal{T(C)}^{\mathcal{S}}/_{=_{E^c_{\mathcal{C},\mathcal{S}}}}$ is finite, the algorithm is guaranteed to terminate.

Example 4. Let $\mathcal{C} = \{A : 0,\ B : 0,\ cons : 2,\ nil : 0\}$ and $E^c_{\mathcal{C},\mathcal{S}} = \{cons(A, cons(A, X)) = cons(A, X), cons(B, cons(B, X)) = cons(B, X), cons(A, cons(B, cons(A, X))) = cons(B, cons(A, X))\}$. The set $E^c_{\mathcal{C},\mathcal{S}}$ is a contracting set of equations for $\mathcal{C}$ and $\mathcal{S}$. Furthermore, we can use $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ to normalize terms since it is terminating and, by definition, has a finite set of normal forms. In the following, let us denote by ①, ②, ... the equivalence classes, like they are used in congruence closure algorithms. Now we start enumerating terms of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$.

- A cannot be normalized and lies into its own class ①;
- B cannot be normalized and lies into its own class ②;
- *nil* cannot be normalized and lies into its own class ③;
- $cons(A, nil)$ is equal to *cons*(①, ③). It cannot be normalized and lies into its own class ④;
- $cons(B, nil)$ is equal to *cons*(②, ③). It cannot be normalized and lies into its own class ⑤;
- $cons(A, cons(A, nil))$ is equal to *cons*(①, ④). Term $cons(A, cons(A, nil))$ can be normalized into $cons(A, nil)$ which belongs to ④. Thus *cons*(①, ④) lies into class ④;
- $cons(B, cons(A, nil))$ is equal to *cons*(②, ④). It cannot be normalized and lies into its own class ⑥;
- $cons(A, cons(B, nil))$ is equal to *cons*(①, ⑤). It cannot be normalized and lies into its own class ⑦;
- $cons(A, cons(A, cons(A, nil)))$ is equal to *cons*(①, ④) which is already in the class ④;
- $cons(B, cons(A, cons(A, nil)))$ is equal to *cons*(②, ④) which is already in the class ⑥;
- $cons(A, cons(B, cons(A, nil)))$ is equal to *cons*(①, ⑥) it can be normalized into $cons(B, cons(A, nil))$ which is in class ⑥. Thus, *cons*(①, ⑥) is in the class ⑥;
- $cons(B, cons(B, cons(A, nil)))$ is equal to *cons*(②, ⑥) it can be normalized into $cons(B, cons(A, nil))$ which is in class ⑥. Thus, *cons*(②, ⑥) is in the class ⑥;
- $cons(A, cons(A, cons(B, nil)))$ is equal to *cons*(①, ⑦) it can be normalized into $cons(A, cons(B, nil))$ which is in class ⑦. Thus, *cons*(①, ⑦) is in the class ⑦;
- $cons(B, cons(A, cons(B, nil)))$ is equal to *cons*(②, ⑦). It cannot be normalized. Thus, *cons*(②, ⑦) lies into its own class ⑧;

Now the only terms that are not covered by those classes are terms of the form $cons(A, l)$ and $cons(B, l)$ where l belongs to class ⑧. We consider those two last cases:

- $cons(A, cons(B, cons(A, cons(B, nil))))$ is equal to *cons*(①, ⑧) it can be normalized into $cons(B, cons(A, cons(B, nil)))$ which is in class ⑧. Thus, *cons*(①, ⑧) is in the class ⑧;
- $cons(B, cons(B, cons(A, cons(B, nil))))$ is equal to *cons*(②, ⑧) it can be normalized into $cons(B, cons(A, cons(B, nil)))$ which is in class ⑧. Thus, *cons*(②, ⑧) is in the class ⑧.

After adding each term it is possible to check if the built automaton recognizes $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$. To produce the tree automaton $\mathcal{A}^E$ it is enough to associate a state q_i to each class ⓘ, to define a transition $cons(q_i, q_j) \rightarrow q_k$ for every term *cons*(ⓘ, ⓙ) belonging to a class ⓚ and transitions $A \rightarrow q_1$, $B \rightarrow q_2$, $nil \rightarrow q_3$. We define the set of final states $\mathcal{Q}_f$ as the set containing all states of the automaton. Here, if we consider the tree automaton computed above, $\mathcal{L}(\mathcal{A}^E) = \mathcal{T}(\mathcal{C})^{\mathcal{S}}$. Thus the algorithm can be stopped.

The theoretical complexity of this algorithm is exponential since, for a given tree automaton $\mathcal{A}$ deciding if $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\mathcal{C})^{\mathcal{S}}$ is exponential [4]. However, it should be possible to have a simpler construction by considering only terms of the form $f(c_1, \ldots, c_n)$ for all equivalence classes c_i, $i = 1 \ldots n$ containing terms of sort S_i and symbols $f : S_1 \times \ldots \times S_n \mapsto S$. Since the set of equivalence class is finite, the number of terms to consider is also finite and the algorithm should naturally stop without performing costly tests to ensure that $\mathcal{L}(\mathcal{A}^E)$ contains $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$.

6 An alternative way to ensure $\mathcal{R}/E$-coherence with Timbuk

An alternative (yet approximated) way to build a $\mathcal{R}/E$-coherent tree automaton from $\mathcal{A}_0$ is to use tree automata completion and simplification itself. Instead of starting from a tree automaton $\mathcal{A}_0$ recognizing an (infinite) initial language, we can start from a tree automaton $\mathcal{A}^G$ recognizing a finite set of non-terminals and generate the language using an additional set of rules $\mathcal{R}^G$. This technique has already been used in [8], to tackle precision problems. First, like above, assume that we are interested in an initial automaton $\mathcal{A}_0$ whose language is of the form $\{C_{in}[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in \mathcal{L} \subseteq \mathcal{T}(\mathcal{C})^{\mathcal{S}}\}$ where $C_{in}[\,]$ is a context that may contain defined symbols. Let $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$. Let $\mathcal{Q}_{\mathcal{L}}$ be the subset of states of $\mathcal{Q}$ recognizing only terms of $\mathcal{L} \subseteq \mathcal{T}(\mathcal{C})^{\mathcal{S}}$, *i.e.* that contain no defined symbols of $C_{in}[\,]$. We now define the shifting operation. It uses an infix symbol $\bullet$, which is a standard symbol, but whose behavior (defined by $\mathcal{R}^G$) is associative and commutative. Thus, terms built on $\bullet$ can be recognized by a standard tree automaton: we do not need AC tree automata. The associative commutative behavior will be taken into account at the completion level.

Definition 10 (Shifting automaton and TRS). *Let* $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *be a tree automaton and* $\mathcal{F}_{\mathcal{Q}} = \{\_ \bullet \_ : 2, \perp : 0\} \cup \{\#_q : 1 \mid q \in \mathcal{Q}_{\mathcal{L}}\}$ *a set of symbols such that* $\mathcal{F} \cap \mathcal{F}_Q = \emptyset$*. The* shifting automaton $\mathcal{A}^G$ *and the* shifting TRS $\mathcal{R}^G$ *are defined as follows:*

– $\mathcal{A}^G = \langle \mathcal{F} \cup \mathcal{F}_{\mathcal{Q}}, \mathcal{Q}, \mathcal{Q}_f, \Delta^G \rangle$ *with* $\mathcal{Q} = \{q_G\}$, $\mathcal{Q}_f = \{q_G\}$, *and*
$\Delta^G = \{\perp \to q_\perp\}$

– $\mathcal{R}^G$ *is the set of rewrite rules containing the rules:*
- $\{x \bullet y \to y \bullet x,\ x \bullet (y \bullet z) \to (x \bullet y) \bullet z,\ (x \bullet y) \bullet z \to x \bullet (y \bullet z)\}$;
- $\{\perp \to \perp \bullet \#_q(a) \mid a \to q \in \Delta\}$
- $\{\#_{q_1}(X_1)\bullet\ldots\bullet\#_{q_n}(X_n) \to \#_q(f(X_1, \ldots, X_n))\bullet\#_{q_1}(X_1)\bullet\ldots\bullet\#_{q_n}(X_n) \mid f(q_1, \ldots, q_n) \to q \in \Delta\}$
- $\{\#_{q_1}(X_1) \bullet \ldots \bullet \#_{q_n}(X_n) \to C_{in}[X_1, \ldots, X_n] \mid C_{in}[q_1, \ldots, q_n] \to^*_{\mathcal{A}_0} q_f$ *and* $q_f \in \mathcal{Q}_f\}$

The intuition behind the shifting operation is to generate, using rewrite rules, all the terms recognized by $\mathcal{A}_0$. The principle is simply to revert each recognition rule $f(q_1, \ldots, q_n) \to q$ of $\mathcal{A}_0$ into a production rule $\#_{q_1}(X_1) \bullet \ldots \bullet \#_{q_n}(X_n) \to \#_q(f(X_1, \ldots, X_n))$ in $\mathcal{R}^G$. In particular, $\mathcal{R}^G$ generates all the terms $\#_q(t)$, for $t \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$, such that $t \to^*_{\mathcal{A}_0} q$. This language is infinite, but since $E^c_{\mathcal{C},\mathcal{S}}$ is contracting for $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$, there is only a finite number of equivalence classes for those terms. Then, the last set of rules of $\mathcal{R}^G$ will generate all terms $C_{in}[t_1, \ldots, t_n]$ as defined in $\mathcal{A}_0$. Starting from $\mathcal{A}^G$, $\mathcal{R}^G$ and $E^c_{\mathcal{C},\mathcal{S}}$, completion generates a tree automaton that recognizes a language that closely over-approximate $\mathcal{L}(\mathcal{A}_0)$ and which is $\mathcal{R}/E^c_{\mathcal{C},\mathcal{S}}$-coherent.

Since completion runs with $E^c_{\mathcal{C},\mathcal{S}}$, the generation of terms $\#_q(t)$, for $t \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$, such that $t \to^*_{\mathcal{A}_0} q$ can be finitely done by completion. What needs a little bit more attention is the first set of rules defining the associative and commutative behavior of the symbol $\bullet$. For this set, termination of completion can be guaranteed using two additional approximation equations merging together all terms built with a bullet symbol: $x \bullet y = x$ and $x \bullet y = y$. Those two equations ensure that any term containing a $\bullet$ is recognized by a unique state. Since $\mathcal{R}^G$ exactly encode the production of the language recognized by $\mathcal{A}_0$, it can be shown using Theorems 2 and 3 that the language recognized by the automaton $\mathcal{A}^*_{\mathcal{R},E}$ produced by completion of $\mathcal{A}^G$ by $\mathcal{R}^G$ has the following properties: $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{L}(\mathcal{A}_0)$ and $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq (\mathcal{R}^G/_{E^c_{\mathcal{C},\mathcal{S}}})^*(\mathcal{L}(\mathcal{A}^G))$. In particular, we can use Theorem 3 because $\mathcal{A}^G$ is trivially $\mathcal{R}^G/E^c_{\mathcal{C},\mathcal{S}}$-coherent since it recognizes exactly one term. Having $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq (\mathcal{R}^G/_{E^c_{\mathcal{C},\mathcal{S}}})^*(\mathcal{L}(\mathcal{A}^G))$ means that the terms recognized by

$\mathcal{A}^*_{\mathcal{R},E}$ which are not part of $\mathcal{L}(\mathcal{A}_0)$ (if any) are due to the application of $E^c_{\mathcal{C},\mathcal{S}}$ to $\mathcal{L}(\mathcal{A}_0)$. Thus, in any way, they would have appeared in the completion of $\mathcal{A}_0$ with simplification with $E^c_{\mathcal{C},\mathcal{S}}$.

We can use the above shifting operation on our introductory example. This results into the following Timbuk[11] specification:

```
Ops delete:2 G1:1 G2:1 cons:2 nil:0 A:0 B:0 ite:3 true:0 false:0 eq:2
     U:2 bot:0
Vars X Y Z V W J
TRS R1
eq(a,a)->true        eq(a,b)->false     eq(b,a)->false     eq(b,b)->true
delete(X,nil)->nil   ite(true,X,Y)->X   ite(false,X,Y)->Y
delete(X,cons(Y,Z))->ite(eq(X,Y),delete(X,Z),cons(Y,delete(X,Z)))

U(X,Y) -> U(Y,X)
U(X,U(Y,Z)) -> U(U(X,Y),Z)
U(U(X,Y),Z) -> U(X,U(Y,Z))

bot -> U(bot,G1(A))
bot -> U(bot,G1(B))
bot -> U(bot,G2(nil))
U(G1(X),G2(Y)) -> U(G2(cons(X,Y)),U(G1(X),G2(Y)))
G2(Y) -> delete(A,Y)

Automaton A0  States qf  Final States qf
Transitions  bot -> qf

Equations E
Rules
%%%%%% approximation for associative commutative behavior of U
U(X,Y)=X
U(X,Y)=Y

Equations E
Rules
%%%%%% Ec
cons(a, cons(a,X)) = cons(a,X)
cons(b, cons(b,X)) = cons(b,X)
cons(a, cons(b, cons(a,X))) = cons(b,cons(a,X))

%%%%%% ER
eq(a,a)=true        eq(a,b)=false     eq(b,a)=false     eq(b,b)=true
delete(X,nil)=nil   ite(true,X,Y)=X   ite(false,X,Y)=Y
delete(X,cons(Y,Z))=ite(eq(X,Y),delete(X,Z),cons(Y,delete(X,Z)))

%%%%%% Er
delete(X,Y)=delete(X,Y)  cons(X,Y)=cons(X,Y)  nil=nil  a=a  b=b
```

Using this encoding, Timbuk is able to compute an initial tree automaton that is $\mathcal{R}/E$-coherent and to continue completion with the definition of `delete`. Completion terminates and ends on a tree automaton permitting to prove that `delete(a,[(a|b)*])=[b*]`.

7 Conclusion

Tree automata completion comes with several theorems. One of them is a precision theorem which uses $\mathcal{R}/E$-coherence of the initial automaton as an hypothesis. In this paper, we have shown this technical restriction can be discarded as soon as the TRS under concern is a functional TRS: *i.e.* a TRS encoding a typed functional program. This result can, probably, be easily lifted to the general case of left-linear TRSs, as soon as there exists a contracting set of equations for $\mathcal{T}(\mathcal{F})$ (see [9] for the definition).

For a functional TRS and an initial tree automaton $\mathcal{A}_0$, we gave a constructive proof of the existence of a $\mathcal{R}/E$-coherent tree automaton $\mathcal{A}'$ recognizing the same language as $\mathcal{A}_0$. Since the complexity of this first algorithm is exponential, we also gave an alternative algorithm producing a precise over-approximation, and which can easily be implemented using completion itself. This has been demonstrated on an example using Timbuk the tree automaton completion tool.

With this result, we can use the full power of the precision theorem of tree automata completion and get the most precise approximations of sets of reachable terms for a given set of approximation equations.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
2. L. Bachmair, A. Tiwari, and L. Vigneron. Abstract congruence closure. *JAR*, 31(2):129–168, 2003.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude, A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
4. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.fr`, 2008.
5. G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasonning*, 33 (3-4):341–383, 2004.
6. T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
7. T. Genet. Reachability analysis of rewriting for software verification. Université de Rennes 1, 2009. Habilitation document, http://www.irisa.fr/celtique/genet/publications.html.
8. T. Genet. Termination Criteria for Tree Automata Completion. *Journal of Logical and Algebraic Methods in Programming*, 2014. 53 pages, Submitted.
9. T. Genet. Towards Static Analysis of Functional Programs using Tree Automata Completion. In *Proceedings of WRLA'14*, volume 8663 of *LNCS*. Springer, 2014.
10. T. Genet and R. Rusu. Equational tree automata completion. *Journal of Symbolic Computation*, 45:574–597, 2010.
11. T. Genet and V. Viet Triem Tong. Timbuk – a Tree Automata Library. IRISA / Université de Rennes 1, 2001. http://www.irisa.fr/celtique/genet/timbuk/.
12. T. Takai, Y. Kaji, and H. Seki. Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability. In *Proc. 11th RTA Conf., Norwich (UK)*, volume 1833 of *LNCS*. Springer-Verlag, 2000.