Towards Static Analysis of Functional Programs using Tree Automata Completion

Thomas Genet

INRIA/IRISA, Université de Rennes, France
genet@irisa.fr

Abstract. This paper presents the first step of a wider research effort to apply tree automata completion to the static analysis of functional programs. Tree Automata Completion is a family of techniques for computing or approximating the set of terms reachable by a rewriting relation. The completion algorithm we focus on is parameterized by a set E of equations controlling the precision of the approximation and influencing its termination. For completion to be used as a static analysis, the first step is to guarantee its termination. In this work, we thus give a sufficient condition on E and $\mathcal{T}(\mathcal{F})$ for completion algorithm to always terminate. In the particular setting of functional programs, this condition can be relaxed into a condition on E and $\mathcal{T}(\mathcal{C})$ (terms built on the set of constructors) that is closer to what is done in the field of static analysis, where abstractions are performed on data.

1 Introduction

Computing or approximating the set of terms reachable by rewriting has more and more applications. For a Term Rewriting System (TRS) $\mathcal{R}$ and a set of terms $L_0 \subseteq \mathcal{T}(\mathcal{F})$, the set of reachable terms is $\mathcal{R}^*(L_0) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in L_0, s \rightarrow_{\mathcal{R}}^* t\}$. This set can be computed exactly for specific classes of $\mathcal{R}$ [10] but, in general, it has to be approximated. Applications of the approximation of $\mathcal{R}^*(L_0)$ are ranging from cryptographic protocol verification [1], to static analysis of various programming languages [5] or to TRS termination proofs [15]. Most of the techniques compute such approximations using tree automata as the core formalism to represent or approximate the (possibly) infinite set of terms $\mathcal{R}^*(L_0)$. Most of them also rely on a Knuth-Bendix completion-like algorithm completing a tree automaton $\mathcal{A}$ recognizing L_0 into an automaton $\mathcal{A}^*$ recognizing exactly, or over-approximating, the set $\mathcal{R}^*(L_0)$. As a result, these techniques can be refered as *tree automata completion* techniques [9, 22, 8, 4, 13, 19]. A strength of this algorithm, and at the same time a weakness, is that its precision is parameterized by a function [8] or a set of equations [13]. It is a strength because tuning the approximation function (or equations) permits to adapt the precision of completion to a specific goal to tackle. This is what made it successful for program and protocol verification. On the other hand, this is a weakness because it is difficult to guarantee its termination.

In this paper, we define a simple sufficient condition on the set of equations for the tree automata completion algorithm to terminate. This condition, which is strong in general, reveals to be natural and well adapted for the approximation of reachable terms when TRSs encode typed functional programs. We thus obtain a way to automatically over-approximate the set of all reachable program states of a functional program, or even restrict it to the set of all results. Thus we can over-approximate the image of a functional program.

2 Related work

Tree automata completion. With regards to most papers about completion [9, 22, 8, 4, 13, 19], our contribution is to give the first criterion *on the approximation* for the completion to terminate. Note that it is possible to guarantee termination of the completion by inferring an approximation adapted to the TRS under concern, like in [20]. In this case, given a TRS, the approximation is fixed and unique. Our solution is more flexible because it lets the user change the precision of the approximation while keeping the termination guarantee. In [22], T. Takai have a completion parameterized by a set of equations. He also gives a termination proof for its completion but only for some restricted classes of TRSs. Here our termination proof holds for any left-linear TRS provided that the set of equations satisfy some properties.

Static analysis of functional programs. With regards to static analysis of functional programs using grammars or automata, our contribution is in the scope of data-flow analysis techniques, rather than control-flow analysis. More precisely, we are interested here in predicting the results of a function [21], rather than predicting the control flow [18]. Those two papers, as well as many other ones, deal with higher order functions using complex higher-order grammar formalisms (PMRS and HORS). Higher-order functions are not in the scope of the solution we propose here. However, we obtained some preliminary results suggesting that an extension to higher order functions is possible and gives relevant results (see Section 6). Furthermore, using equations, approximations are defined in a more declarative and flexible way than in [21], where they are defined by a dedicated algorithm. Besides, the verification mechanisms of [21] use automatic abstraction refinement. This can be also performed in the completion setting [3] and adapted to the analysis of functional programs [14]. Finally, using a simpler (first order) formalism, *i.e.* tree automata, makes it easier to take into account some other aspects like: evaluation strategies and built-ins types (see Section 6) that are not considered by those papers.

3 Background

In this section, we introduce some definitions and concepts that will be used throughout the rest of the paper (see also [2, 7]). 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*.

$\mathcal{T}(\mathcal{F},\mathcal{X})$ 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 subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s.

A *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* if each variable of l occurs only once in l. A TRS $\mathcal{R}$ is left-linear if every rewrite rule $l \to r$ of $\mathcal{R}$ is left-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 $t = s[r\sigma]_p$. 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*. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\to^*_{\mathcal{R}}$ and $s \to^!_{\mathcal{R}} t$ denotes that $s \to^*_{\mathcal{R}} t$ and t is irreducible by $\mathcal{R}$. The set of irreducible terms w.r.t. a TRS $\mathcal{R}$ is denoted by $\textsc{Irr}(\mathcal{R})$. The set of $\mathcal{R}$-descendants of a set of ground terms I is $\mathcal{R}^*(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to^*_{\mathcal{R}} t\}$. A TRS $\mathcal{R}$ is sufficiently complete if for all $s \in \mathcal{T}(\mathcal{F})$, $(R^*(\{s\}) \cap \mathcal{T}(\mathcal{C})) \neq \emptyset$.

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 substitution σ we have $l\sigma =_E r\sigma$. 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 $\to^*_{\mathcal{R}/E}$ of $\to_{\mathcal{R}/E}$ 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 $s \to^*_{\mathcal{R}/E} t$. The set of $\mathcal{R}$-descendants modulo E of a set of ground terms I is $\mathcal{R}^*_E(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to^*_{\mathcal{R}/E} t\}$.

Let $\mathcal{Q}$ be a countably infinite set of symbols with arity 0, called *states*, such that $\mathcal{Q} \cap \mathcal{F} = \emptyset$. $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is called the set of *configurations*. A *transition* is a rewrite rule $c \to q$, where c is a configuration and q is state. A transition is *normalized* when $c = f(q_1,\dots,q_n)$, $f \in \mathcal{F}$ is of arity n, and $q_1,\dots,q_n \in \mathcal{Q}$. An *$\epsilon$-transition* is a transition of the form $q \to q'$ 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 $\to^*_{\Delta}$ (resp. $\to^{\not\epsilon *}_{\Delta}$). When Δ is attached to a tree automaton $\mathcal{A}$ we also note those two relations $\to_{\mathcal{A}}{}^*$ and $\to^{\not\epsilon *}_{\mathcal{A}}$, 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 $s \to_{\mathcal{A}}{}^{*} q$. The language (resp. $\not\epsilon$-language) recognized by $\mathcal{A}$ in a state q is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_{\mathcal{A}}^{*} q\}$ (resp. $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_{\mathcal{A}}^{\not\epsilon *} q\}$). A state q of an automaton $\mathcal{A}$ is *reachable* (resp. $\not\epsilon$-reachable) if $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$ (resp. $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) \neq \emptyset$). We define $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_F} \mathcal{L}(\mathcal{A}, q)$. A set of transitions Δ is $\not\epsilon$-deterministic if 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 transitions 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 $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q_1) \cap \mathcal{L}^{\not\epsilon}(\mathcal{A}, q_2) = \emptyset$.

4 Tree Automata Completion Algorithm

Tree Automata Completion algorithms were proposed in [16, 9, 22, 13]. They are very similar to a Knuth-Bendix completion except that they run on two distinct sets of rules: a TRS $\mathcal{R}$ and a set of transitions Δ of a tree automaton $\mathcal{A}$.

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 algorithm computes a tree automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ or $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. The algorithm iteratively computes tree automata $\mathcal{A}_{\mathcal{R}}^1$, $\mathcal{A}_{\mathcal{R}}^2$, ... such that $\forall i \geq 0 : \mathcal{L}(\mathcal{A}_{\mathcal{R}}^i) \subseteq \mathcal{L}(\mathcal{A}_{\mathcal{R}}^{i+1})$ until we get an automaton $\mathcal{A}_{\mathcal{R}}^k$ with $k \in \mathbb{N}$ and $\mathcal{L}(\mathcal{A}_{\mathcal{R}}^k) = \mathcal{L}(\mathcal{A}_{\mathcal{R}}^{k+1})$. For all $i \in \mathbb{N}$, if $s \in \mathcal{L}(\mathcal{A}_{\mathcal{R}}^i)$ and $s \to_{\mathcal{R}} t$, then $t \in \mathcal{L}(\mathcal{A}_{\mathcal{R}}^{i+1})$. Thus, if $\mathcal{A}_{\mathcal{R}}^k$ is a fixpoint then it also verifies $\mathcal{L}(\mathcal{A}_{\mathcal{R}}^k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. To construct $\mathcal{A}_{\mathcal{R}}^{i+1}$ from $\mathcal{A}_{\mathcal{R}}^i$, we achieve a *completion step* which consists in finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_{\mathcal{R}}^i}$. A critical pair is a triple $(l \to r, \sigma, q)$ where $l \to r \in \mathcal{R}$, $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and $q \in \mathcal{Q}$ such that $l\sigma \to_{\mathcal{A}_{\mathcal{R}}^i}^{*} q$ and $r\sigma \not\to_{\mathcal{A}_{\mathcal{R}}^i}^{*} q$. 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 $\mathcal{A}_{\mathcal{R}}^i$ to obtain $\mathcal{A}_{\mathcal{R}}^{i+1}$ such that $r\sigma \to_{\mathcal{A}_{\mathcal{R}}^{i+1}}^{*} q$. In [22, 13], critical pairs are joined in the following way:

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

From an algorithmic point of view, there remains two problems to solve: find all the critical pairs $(l \to r, \sigma, q)$ and find the transitions to add to $\mathcal{A}_{\mathcal{R}}^i$ to have $r\sigma \to_{\mathcal{A}_{\mathcal{R}}^{i+1}}^{*} q$. The first problem, called matching, can be efficiently solved using a specific algorithm [8, 10]. The second problem is solved using Normalization.

4.1 Normalization

The normalization function replaces subterms either by states of $\mathcal{Q}$ (using transitions of Δ) or by new states. A state q of $\mathcal{Q}$ is used to normalize a term t if $t \to_{\Delta}^{\not\epsilon} q$. Normalizing by reusing states of $\mathcal{Q}$ and transitions of Δ permits to preserve the $\not\epsilon$-determinism of $\to_{\Delta}^{\not\epsilon}$. Indeed, $\to_{\Delta}^{\not\epsilon}$ can be kept deterministic during completion though $\to_{\Delta}$ cannot.

Definition 1 (New state). *Given a set of transitions Δ, a new state (for Δ) is a state of $\mathcal{Q} \setminus \mathcal{Q}_f$ not occurring in left or right-hand sides of rules of Δ* [1].

We here define normalization as a bottom-up process. This definition is simpler and equivalent to top-down definitions [13]. In the recursive call, the choice of the context $C[]$ may be non deterministic but all the possible results are the equivalent modulo state renaming.

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

In the second case of the definition, if there are several states q' such that $f(q_1, \ldots, q_n) \to q' \in \Delta$, we arbitrarily choose one of them. We illustrate the above definition on the normalization of a simple transition.

Example 1. Given $\Delta = \{b \to q_0\}$, $Norm_\Delta(f(g(a), b, g(a)) \to q) = \{a \to q_1, g(q_1) \to q_2, b \to q_0, f(q_2, q_0, q_2) \to q\}$

4.2 One step of completion

A step of completion only consists in joining critical pairs. We first need to formally define the substitutions under concern: *state substitutions.*

Definition 3 (State substitutions, $\Sigma(\mathcal{Q}, \mathcal{X})$). *A* state substitution *over an automaton $\mathcal{A}$ with a set of states $\mathcal{Q}$ is a function $\sigma : \mathcal{X} \mapsto \mathcal{Q}$. We can extend this definition to a morphism $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{T}(\mathcal{F}, \mathcal{Q})$. We denote by $\Sigma(\mathcal{Q}, \mathcal{X})$ the set of state substitutions built over $\mathcal{Q}$ and $\mathcal{X}$.*

Definition 4 (Set of critical pairs). *Let a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$. The set of critical pairs between $\mathcal{R}$ and $\mathcal{A}$ is $CP(\mathcal{R}, \mathcal{A}) = \{(l \to r, \sigma, q) \mid l \to r \in \mathcal{R},\ q \in \mathcal{Q},\ \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}),\ l\sigma \to_{\mathcal{A}}^* q,\ r\sigma \not\to_{\mathcal{A}}^* q\}$.*

Recall that the completion process builds a sequence $\mathcal{A}_{\mathcal{R}}^0, \mathcal{A}_{\mathcal{R}}^1, \ldots, \mathcal{A}_{\mathcal{R}}^k$ of automata such that if $s \in \mathcal{L}(\mathcal{A}_{\mathcal{R}}^i)$ and $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{\mathcal{R}}^{i+1})$. One step of completion, *i.e.* the process computing $\mathcal{A}_{\mathcal{R}}^{i+1}$ from $\mathcal{A}_{\mathcal{R}}^i$, is defined as follows. Again, the following definition is a simplification of the definition of [13].

[1] Since $\mathcal{Q}$ is a countably infinite set of states, $\mathcal{Q}_f$ and Δ are finite, a new state can always be found.

Definition 5 (One step of completion). *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $\mathcal{R}$ be a left-linear TRS. The one step completed automaton is $\mathcal{C}_{\mathcal{R}}(\mathcal{A}) = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, Join^{CP(\mathcal{R},\mathcal{A})}(\Delta)\rangle$ where $Join^S(\Delta)$ is inductively defined by:*

– $Join^{\emptyset}(\Delta) = \Delta$
– $Join^{\{(l\to r,q,\sigma)\}\cup S}(\Delta) = Join^S(\Delta \cup \Delta')$ *where*
 $\Delta' = \{q' \to q\}$ *if there exists* $q' \in \mathcal{Q}$ *s.t.* $r\sigma \to_{\Delta}^{\not\epsilon *} q'$*, and otherwise*
 $\Delta' = Norm_{\Delta}(r\sigma \to q') \cup \{q' \to q\}$ *where* q' *is a new state for* Δ

Example 2. Let $\mathcal{A}$ be a tree automaton with $\Delta = \{f(q_1) \to q_0, a \to q_1, g(q_1) \to q_2\}$. If $\mathcal{R} = \{f(x) \to f(g(x))\}$ then $CP(\mathcal{R},\mathcal{A}) = \{(f(x) \to f(g(x)), \sigma_3, q_0)\}$ with $\sigma_3 = \{x \mapsto q_1\}$, because $f(x)\sigma_3 \to_{\mathcal{A}}{}^{*} q_0$ and $f(x)\sigma_3 \to_{\mathcal{R}} f(g(x))\sigma_3$. We have $f(g(x))\sigma_3 = f(g(q_1))$ and there exists no state q such that $f(g(q_1)) \to_{\mathcal{A}}^{\not\epsilon *} q$. Hence, $Join^{\{(f(x)\to f(g(x)),\sigma_3,q_0)\}}(\Delta) = Join^{\emptyset}(\Delta \cup Norm_{\Delta}(f(g(q_1)) \to q_3) \cup \{q_3 \to q_0\})$. Since $Norm_{\Delta}(f(g(q_1)) \to q_3) = \{f(q_2) \to q_3, q(q_1) \to q_2\}$, we get that $\mathcal{C}_{\mathcal{R}}(\mathcal{A}) = \langle \mathcal{F}, \mathcal{Q} \cup \{q_3\}, \mathcal{Q}_f, \Delta \cup \{f(q_2) \to q_3, q_3 \to q_0\}\rangle$.

4.3 Simplification of Tree Automata by Equations

In this section, we define the *simplification* of tree automata $\mathcal{A}$ 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 6 (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 7 (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 $s\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_a$, $t\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_b$, and $q_a \neq q_b$ then $\mathcal{A}$ can be* simplified *into $\mathcal{A}' = \mathcal{A}\{q_b \mapsto q_a\}$, denoted by $\mathcal{A} \leadsto_E \mathcal{A}'$.*

Example 3. Let $E = \{s(s(x)) = s(x)\}$ and $\mathcal{A}$ be the tree automaton with set of transitions $\Delta = \{a \to q_0, s(q_0) \to q_1, s(q_1) \to q_2\}$. We can perform a simplification step using the equation $s(s(x)) = s(x)$ because we found a substitution $\sigma = \{x \mapsto q_0\}$ such that: $s(s(x))\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_2$ and $s(x)\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_1$ Hence, $\mathcal{A} \leadsto_E \mathcal{A}' = \mathcal{A}\{q_2 \mapsto q_1\}$[2]

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

As stated in [13], simplification $\leadsto_E$ is a terminating relation (each step suppresses a state) and it enlarges the language recognized by a tree automaton, *i.e.* if $\mathcal{A} \leadsto_E \mathcal{A}'$ 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. In the following, $\mathcal{A} \leadsto^!_E \mathcal{A}'$ denotes that $\mathcal{A} \leadsto^*_E \mathcal{A}'$ and $\mathcal{A}'$ is irreducible by $\leadsto_E$. We denote by $\mathcal{S}_E(\mathcal{A})$ any automaton $\mathcal{A}'$ such that $\mathcal{A} \leadsto^!_E \mathcal{A}'$.

Theorem 1 (Simplified Tree Automata [13]). *Let $\mathcal{A}, \mathcal{A}'_1, \mathcal{A}'_2$ be tree automata and E be a set of equations. If $\mathcal{A} \leadsto^!_E \mathcal{A}'_1$ and $\mathcal{A} \leadsto^!_E \mathcal{A}'_2$ then $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are equivalent modulo state renaming.*

4.4 The full Completion Algorithm

Definition 8 (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* $n \geq 0$

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

In practice, checking if $CP(\mathcal{R}, \mathcal{A}^k_{\mathcal{R},E}) = \emptyset$ is sufficient to know that $\mathcal{A}^k_{\mathcal{R},E}$ is a fixpoint. However, a fixpoint cannot always be finitely reached[3]. To ensure termination, one can provide a set of approximating equations to overcome infinite rewriting and completion divergence.

Example 4. 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 $\mathcal{A}^0, \ldots, \mathcal{A}^{i-1}_{\mathcal{R},E}$. The automaton $\mathcal{A}^1_{\mathcal{R},E}$ is exactly $\mathcal{C}_\mathcal{R}(\mathcal{A}^0)$ since simplification by equations do not apply. Simplification has been applied on $\mathcal{C}_\mathcal{R}(\mathcal{A}^1_{\mathcal{R},E})$ to obtain $\mathcal{A}^2_{\mathcal{R},E}$.

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

Now, we recall the lower and upper bound theorems. Tree automata completion of automaton $\mathcal{A}$ with TRS $\mathcal{R}$ and set of equations E is lower bounded by $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ and upper bounded by $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. The lower bound theorem ensures that the completed automaton $\mathcal{A}^*_{\mathcal{R},E}$ recognizes all $\mathcal{R}$-reachable terms (but not all $\mathcal{R}/E$-reachable terms). The upper bound theorem guarantees that all terms recognized by $\mathcal{A}^*_{\mathcal{R},E}$ are only $\mathcal{R}/E$-reachable terms.

[3] See [10], for classes of $\mathcal{R}$ for which a fixpoint always exists.

Theorem 2 (Lower bound [13]). *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 $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.*

The upper bound theorem states the precision result of completion. It is defined using the $\mathcal{R}/E$-coherence property. 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.

Theorem 3 (Upper bound [13]). *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 $i \in \mathbb{N}$: $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$ and $\mathcal{A}^i_{\mathcal{R},E}$ is $\mathcal{R}/E$-coherent.*

5 Termination criterion for a given set of equations

Given a set of equations E, the effect of the simplification with E on a tree automaton is to merge two distinct states recognizing instances of the left and right-hand side for all the equations of E. In this section, we give a sufficient condition on E and on the completed tree automata $\mathcal{A}^i_{\mathcal{R},E}$ for the tree automata completion to always terminate. The intuition behind this condition is simple: if the set of equivalence classes for E, *i.e.* $\mathcal{T}(\mathcal{F})/_{=_E}$, is finite then so should be the set of new states used in completion. However, this is not true in general because simplification of an automaton with E does not necessarily merge all E-equivalent terms.

Example 5. Let $\mathcal{A}$ be the tree automaton with set of transitions $a \to q$, $\mathcal{R} = \{a \to c\}$ and let $E = \{a = b, b = c\}$. The set of transitions of $\mathcal{C}_\mathcal{R}(\mathcal{A})$ is $\{a \to q, c \to q', q' \to q\}$. We have $a =_E c$, $a \in \mathcal{L}^{\not\epsilon}(\mathcal{C}_\mathcal{R}(\mathcal{A}), q)$ and $c \in \mathcal{L}^{\not\epsilon}(\mathcal{C}_\mathcal{R}(\mathcal{A}), q')$ but on the automaton $\mathcal{C}_\mathcal{R}(\mathcal{A})$, no simplification situation (as described by Definition 7), can be found because the term b is not recognized by $\mathcal{C}_\mathcal{R}(\mathcal{A})$. Hence, the simplified automaton is $\mathcal{C}_\mathcal{R}(\mathcal{A})$ where a and c are recognized by different states.

There is no simple solution to have a simplification algorithm merging all states recognizing E-equivalent terms (see Section 6). Having a complete automaton $\mathcal{A}$ solve the above problem but leads to rough approximations (see [11]). In the next section, we propose to give some simple restrictions on E to ensure that completion terminates. In Section 5.2, we will see how those restrictions can easily be met for "functional" TRS, *i.e.* a typed first-order functional program translated into a TRS.

5.1 General criterion

What Example 5 shows is that, for a simplification with E to apply, it is necessary that both sides of the equation are recognized by the tree automaton. In the following, we will define a set E^c of *contracting* equations so that this property is true. What Example 5 does not show is that, by default, tree automata are not E-compatible. In particular, any non $\not\epsilon$-deterministic automaton does not satisfy the reflexivity of $=_E$. For instance, if an automaton $\mathcal{A}$ has two transitions $a \to q_1$ and $a \to q_2$, since $a =_E a$ for all E, for $\mathcal{A}$ to be E-compatible we should have $q_1 = q_2$. To enforce $\not\epsilon$-determinism by automata simplification, we define a set of *reflexivity equations* as follows.

Definition 9 (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.*

Note that for all set of equations E, the relation $=_E$ is trivially equivalent to $=_{E \cup E^r}$. Furthermore, simplification with E^r transforms all automaton into an $\not\epsilon$-deterministic automaton, as stated in the following lemma.

Lemma 1. *For all tree automaton $\mathcal{A}$ and all set of equation E, if $E \supseteq E^r$ and $\mathcal{A} \rightsquigarrow^!_E \mathcal{A}'$ then $\mathcal{A}'$ is $\not\epsilon$-deterministic.*

Proof. Shown by induction on the height of terms (see [11] for details). □

We now define sets of contracting equations. Such sets are defined for a set of symbols $\mathcal{K}$ which can be a subset of $\mathcal{F}$. This will be used later to restrict contracting equations to the subset of constructor symbols of $\mathcal{F}$.

Definition 10 (Sets of contracting equations for $\mathcal{K}$, $E^c_{\mathcal{K}}$). *Let $\mathcal{K} \subseteq \mathcal{F}$. A set of equations is contracting for $\mathcal{K}$, denoted by $E^c_{\mathcal{K}}$, if all equations of $E^c_{\mathcal{K}}$ are of the form $u = u|_p$ with $u \in \mathcal{T}(\mathcal{K}, \mathcal{X})$ a linear term, $p \neq \lambda$, and if the set of normal forms of $\mathcal{T}(\mathcal{K})$ w.r.t. the TRS $\overrightarrow{E^c_{\mathcal{K}}} = \{u \to u|_p \mid u = u|_p \in E^c_{\mathcal{K}}\}$ is finite.*

Contracting equations, if defined on $\mathcal{F}$, define an upper bound on the number of states of a simplified automaton.

Lemma 2. *Let $\mathcal{A}$ be a tree automaton and $E^c_{\mathcal{F}}$ a set of contracting equations for $\mathcal{F}$. If $E \supseteq E^c_{\mathcal{F}} \cup E^r$ then the simplified automaton $\mathcal{S}_E(\mathcal{A})$ is an $\not\epsilon$-deterministic automaton having no more states than terms in* $\textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}})$.

Proof. First, assume for all state q of $\mathcal{S}_E(\mathcal{A})$, $\mathcal{L}^{\not\epsilon}(\mathcal{S}_E(\mathcal{A}), q) \cap \textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}}) = \emptyset$. Then, for all terms s such that $s \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$, we know that s is not in normal form w.r.t. $\overrightarrow{E^c_{\mathcal{F}}}$. As a result, the left-hand side of an equation of $E^c_{\mathcal{F}}$ can be applied to s. This means that there exists an equation $u = u|_p$, a ground context C and a substitution θ such that $s = C[u\theta]$. Furthermore, since $s \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$, we know that $C[u\theta] \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$ and that there exists a state q' such that $C[q'] \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$

and $u\theta \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q'$. From $u\theta \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q'$, we know that all subterms of $u\theta$ are recognized by at least one state in $\mathcal{S}_E(\mathcal{A})$. Thus, there exists a state q'' such that $u|_p\theta \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q''$. We thus have a situation of application of the equation $u = u|_p$ in the automaton. Since $\mathcal{S}_E(\mathcal{A})$ is simplified, we thus know that $q' = q''$. As mentioned above, we know that $C[q'] \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$. Hence $C[u|_p\theta] \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} C[q'] \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$. If $C[u|_p\theta]$ is not in normal form w.r.t. $\overrightarrow{E^c_{\mathcal{F}}}$ then we can do the same reasoning on $C[u|_p\theta] \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$ until getting a term that is in normal form w.r.t. $\overrightarrow{E^c_{\mathcal{F}}}$ and recognized by the same state q. Thus, this contradicts the fact that $\mathcal{S}_E(\mathcal{A})$ recognizes no term of $\textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}})$.

Then, by definition of $E^c_{\mathcal{F}}$, $\textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}})$ is finite. Let $\{t_1, \ldots, t_n\}$ be the subset of $\textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}})$ recognized by $\mathcal{S}_E(\mathcal{A})$. Let $q_1, \ldots, q_n$ be the states recognizing $t_1, \ldots, t_n$ respectively. We know that there is a finite set of states recognizing $t_1, \ldots, t_n$ because $E \supseteq E^r$ and Lemma 1 entails that $\mathcal{S}_E(\mathcal{A})$ is $\not\epsilon$-deterministic. Now, for all terms s recognized by a state q in $\mathcal{S}_E(\mathcal{A})$, *i.e.* $s \rightarrow^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$, we can use a reasoning similar to the one carried out above and show that q is equal to one state of $\{q_1, \ldots, q_n\}$ recognizing normal forms of $\overrightarrow{E^c_{\mathcal{F}}}$ in $\mathcal{S}_E(\mathcal{A})$. Finally, there are at most $card(\textsc{Irr}(\overrightarrow{E^c_{\mathcal{F}}}))$ states in $\mathcal{S}_E(\mathcal{A})$. □

Now it is possible to state the Theorem guaranteeing the termination of completion if the set of equations E contains a set of contracting equations $E^c_{\mathcal{F}}$ for $\mathcal{F}$ and a set of reflexivity equations.

Theorem 4. *Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ a left linear TRS and E a set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{F}}$, then completion of $\mathcal{A}$ by $\mathcal{R}$ and E terminates.*

Proof. For completion to diverge it must produce infinitely many new states. This is impossible if E contains $E^c_{\mathcal{F}}$ and E^r (see Lemma 2). □

5.2 Criterion for Functional TRSs

Now, we consider functional programs viewed as TRSs. We assume that such TRSs are left-linear, which is a common assumption on TRSs obtained from functional programs [2]. In this section, we will restrict ourselves to sufficiently complete TRSs obtained from functional programs and will refer to them as *functional TRSs*. For TRSs representing functional programs, defining contracting equations of $E^c_{\mathcal{C}}$ on $\mathcal{C}$ rather than on $\mathcal{F}$ is enough to guarantee termination of completion. This is more convenient and also closer to what is usually done in static analysis where abstractions are usually defined on data and not on function applications. Since the TRSs we consider are sufficiently complete, any term of $\mathcal{T}(\mathcal{F})$ can be rewritten into a data-term of $\mathcal{T}(\mathcal{C})$. As above, using equations of $E^c_{\mathcal{C}}$ we are going to ensure that the data-terms of the computed languages will be recognized by a bounded set of states. To lift-up this property to $\mathcal{T}(\mathcal{F})$ it is enough to ensure that $\forall s, t \in \mathcal{T}(\mathcal{F})$ if $s \rightarrow_R t$ then s and t are recognized by equivalent states. This is the role of the set of equations E_R.

Definition 11 ($E_{\mathcal{R}}$). *Let $\mathcal{R}$ be a TRS, the set of $\mathcal{R}$-equations is $E_{\mathcal{R}} = \{l = r \mid l \to r \in \mathcal{R}\}$.*

Theorem 5. *Let $\mathcal{A}_0$ be a tree automaton, $\mathcal{R}$ a sufficiently complete left-linear TRS and E a set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{C}} \cup E_{\mathcal{R}}$ with $E^c_{\mathcal{C}}$ contracting then completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E terminates.*

Proof. Firstly, to show that the number of states recognizing terms of $\mathcal{T}(\mathcal{C})$ is finite we can do a proof similar to the one of Lemma 2 . Let $G \subseteq \mathcal{T}(\mathcal{C})$ be the finite set of normal forms of $\mathcal{T}(\mathcal{C})$ w.r.t. $\overrightarrow{E^c_{\mathcal{C}}}$. Since $E \supseteq E^r \cup E^c_{\mathcal{C}}$, like in the proof of Lemma 2, we can show that in any completed automaton, terms of $\mathcal{T}(\mathcal{C})$ are recognized by no more states than terms in G. Secondly, since $\mathcal{R}$ is sufficiently complete, for all terms $s \in \mathcal{T}(\mathcal{F}) \setminus \mathcal{T}(\mathcal{C})$ we know that there exists a term $t \in \mathcal{T}(\mathcal{C})$ such that $s \to_{\mathcal{R}}{}^* t$. The fact that $E \supseteq E_{\mathcal{R}}$ guarantees that s and t will be recognized by equivalent states in the completed (and simplified) automaton. Since the number of states necessary to recognize $\mathcal{T}(\mathcal{C})$ is finite, so is the number of states necessary to recognize terms of $\mathcal{T}(\mathcal{F})$. □

Finally, 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 $\mathcal{R}$. We still assume that the (sorted) TRS is sufficiently complete, which is defined in a similar way except that it holds only for well-sorted terms, *i.e.* for all $s \in \mathcal{T}(\mathcal{F})^{\mathcal{S}}$ there exists a term $t \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$ such that $s \to_{\mathcal{R}}{}^* t$. We slightly refine the definition of contracting equations as follows. For all sort S, if S has a unique constant symbol we note it c^S.

Definition 12 (Set $E^c_{\mathcal{K},\mathcal{S}}$ of contracting equations for $\mathcal{K}$ and $\mathcal{S}$). *Let $\mathcal{K} \subseteq \mathcal{F}$. The set of well-sorted equations $E^c_{\mathcal{K},\mathcal{S}}$ is* contracting *(for $\mathcal{K}$) if its equations are of the form (a) $u = u|_p$ with u linear and $p \neq \Lambda$, or (b) $u = c^S$ with u of sort S, and if the set of normal forms of $\mathcal{T}(\mathcal{K})^{\mathcal{S}}$ w.r.t. the TRS $\overrightarrow{E^c_{\mathcal{K},\mathcal{S}}} = \{u \to v \mid u = v \in E^c_{\mathcal{K},\mathcal{S}} \wedge (v = u|_p \vee v = c^S)\}$ is finite.*

The termination theorem for completion of sorted TRSs is similar to the previous one except that it needs $\mathcal{R}/E$-coherence of $\mathcal{A}_0$ to ensure that terms recognized by completed automata are well-sorted (see [11] for proof).

Theorem 6. *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 and $\mathcal{A}_0$ is $\mathcal{R}/E$-coherent then completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E terminates.*

5.3 Experiments

The objective of data-flow analysis is 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 $\mathcal{A}_{\text{IRR}(\mathcal{R})}$ recognizing $\text{IRR}(\mathcal{R})$. 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 $\mathcal{A}_{\text{IRR}(\mathcal{R})}$.

Here is an example of application of those theorems. Completions are performed using Timbuk. All the $\mathcal{A}_{\text{IRR}(\mathcal{R})}$ automata and intersections were performed using Taml. Details can be found in [14].

```
Ops append:2 rev:1 nil:0 cons:2 a:0 b:0    Vars X Y Z U Xs
TRS R
append(nil,X)->X      append(cons(X,Y),Z)->cons(X,append(Y,Z))
rev(nil)->nil         rev(cons(X,Y))->append(rev(Y),cons(X,nil))

Automaton A0 States q0 qla qlb qnil qf qa qb Final States q0 Transitions
rev(qla)->q0          cons(qb,qnil)->qlb    cons(qa,qla)->qla   nil->qnil
cons(qa,qlb)->qla     a->qa                 cons(qb,qlb)->qlb   b->qb

Equations E Rules    cons(X,cons(Y,Z))=cons(Y,Z)  %%% Ec
%%% E_R                                       %%% E^r
append(nil,X)=X                               rev(X)=rev(X)
append(cons(X,Y),Z)=cons(X,append(Y,Z))       cons(X,Y)=cons(X,Y)
rev(nil)=nil                                  append(X,Y)=append(X,Y)
rev(cons(X,Y))=append(rev(Y),cons(X,nil))     a=a  b=b  nil=nil
```

In this example, the TRS $\mathcal{R}$ encodes the classical *reverse* and *append* functions. The language recognized by automaton $\mathcal{A}_0$ is the set of terms of the form $rev([a, a, \ldots, b, b, \ldots])$. Note that there are at least one a and one b in the list. We assume that $\mathcal{S} = \{T, list\}$ and sorts for symbols are the following: $a : T$, $b : T$, $nil : list$, $cons : T \times list \mapsto list$, $append : list \times list \mapsto list$ and $rev : list \mapsto list$. Now, to use Theorem 6, we need to prove each of its assumptions. The set E of equations contains $E_{\mathcal{R}}$, E^r and $E^c_{\mathcal{C},\mathcal{S}}$. The set of Equations $E^c_{\mathcal{C},\mathcal{S}}$ is contracting because the automaton $\mathcal{A}_{\text{IRR}(\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}})}$ recognizes a finite language. This automaton can be computed using Taml: it is the intersection between the automaton $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$[4] recognising $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ and the automaton $\mathcal{A}_{\text{IRR}(\{cons(X,cons(Y,Z))\to cons(Y,Z)\})}$:

[4] Such an automaton has one state per sort and one transition per constructor. For instance, on our example $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$ will have transitions: $a \to qT$, $b \to qT$, $cons(qT, qlist) \to qlist$ and $nil \to qlist$.

States `q2 q1 q0` **Final States** `q0 q1 q2`
Transitions `b->q2 a->q2 nil->q1 cons(q2,q1)->q0`

The language of $\mathcal{A}_0$ is well-sorted and E and $\mathcal{R}$ are sort preserving. We can prove sufficient completeness of $\mathcal{R}$ on $\mathcal{T}(\mathcal{F})^{\mathcal{S}}$ using, for instance, Maude [6] or even Timbuk [9] itself. The last assumption of Theorem 6 to prove is that $\mathcal{A}_0$ is $\mathcal{R}/E$-coherent. This can be shown by remarking that each state q of $\mathcal{A}_0$ recognizes at least one term and if $s \rightarrow_{\mathcal{A}_0}^{\not\epsilon *} q$ and $t \rightarrow_{\mathcal{A}_0}^{\not\epsilon *} q$ then $s =_E t$. For instance $cons(b, cons(b, nil)) \rightarrow_{\mathcal{A}_0}^{\not\epsilon *} q_{lb}$ and $cons(b, nil) \rightarrow_{\mathcal{A}_0}^{\not\epsilon *} q_{lb}$ and $cons(b, cons(b, nil)) =_E cons(b, nil)$. Thus, completion is guaranteed to terminate: after 4 completion steps (7 ms) we obtain a fixpoint automaton $\mathcal{A}^*_{\mathcal{R},E}$ with 11 transitions. To restrain the language to normal forms it is enough to compute the intersection with $\textsc{Irr}(R)$. Since we are dealing with sufficiently complete TRSs, we know that $\textsc{Irr}(R) \subseteq \mathcal{T}(\mathcal{C})^{\mathcal{S}}$. Thus, we can use again $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$ for the intersection that is:

States `q3 q2 q1 q0` **Final States** `q3` **Transitions** `a->q0 nil->q1 b->q2`
`cons(q0,q1)->q3 cons(q0,q3)->q3 cons(q2,q1)->q3 cons(q2,q3)->q3`

which recognizes any (non empty) flat list of a and b. Thus, our analysis preserved the property that the result cannot be the empty list but lost the order of the elements in the list. This is not surprising because the equation `cons(X, cons(Y, Z))=cons(X, Z)` makes $cons(a, cons(b, nil))$ equal to $cons(a, nil)$. It is possible to refine by hand $E^c_{\mathcal{C},\mathcal{S}}$ using the following equations: `cons(a,cons(a,X))=cons(a,X)`, `cons(b,cons(b,X))=cons(b,X)`, `cons(a,cons(b,cons(a,X)))=cons(a,X)`. This set of equations avoids the previous problem. Again, E verifies the conditions of Theorem 6 and completion is still guaranteed to terminate. The result is the automaton $\mathcal{A}'^*_{\mathcal{R},E}$ having 19 transitions. This time, intersection with $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$ gives:

States `q4 q3 q2 q1 q0` **Final States** `q4` **Transitions** `a->q1 b->q3 nil->q0`
`cons(q1,q0)->q2 cons(q1,q2)->q2 cons(q3,q2)->q4 cons(q3,q4)->q4`

This automaton exactly recognizes lists of the form $[b, b, \ldots, a, a, \ldots]$ with at least one b and one a, as expected. Hopefully, refinement of equations can be automatized in completion [3] and can be used here, see [14] for examples. More examples can be found in the Timbuk 3.1 source distribution.

6 Conclusion and further research

In this paper we defined a criterion on the set of approximation equations to guarantee termination of the tree automata completion. When dealing with, so called, functional TRS this criterion is close to what is generally expected in static analysis and abstract interpretation: a finite model for an infinite set of data-terms. This work is a first step to use completion for static analysis of functional programs. There remains some interesting points to address.

Dealing with higher-order functions. Higher-order functions can be encoded into first order TRS using a simple encoding borrowed from [17]: defined symbols become constants, constructor symbols remain the same, and an additional *application* operator '@' of arity 2 is introduced. On all the examples of [21], completion and this simple encoding produces exactly the same results [14].

Dealing with evaluation strategies. The technique proposed here, as well as [21], over-approximates the set of results for all evaluation strategies. As far as we know, no static analysis technique for functional programs can take into account evaluation strategies. However, it is possible to restrict the completion algorithm to recognize only innermost descendants [14], *i.e.* call-by-value results. If the approximation is precise enough, any non terminating program with call-by-value will have an empty set of results. An open research direction is to use this to prove non termination of functional programs by call-by-value strategy.

Dealing with built-in types. Values manipulated by *real* functional programs are not always terms or trees. They can be numerals or be terms embedding numerals. In [12], it has been shown that completion can compute over-approximations of reachable terms embedding built-in terms. The structural part of the term is approximated using tree automata and the built-in part is approximated using lattices and abstract interpretation.

Besides, there remain some interesting theoretical points to solve. In section 5, we saw that having a finite $\mathcal{T}(\mathcal{F})/_{=_E}$ is not enough to guarantee the termination of completion. This is due to the fact that the simplification algorithm does not merge all states recognizing E-equivalent terms. Having a simplification algorithm ensuring this property is not trivial. First, the theory defined by E has to be decidable. Second, even if E is decidable, finding all the E-equivalent terms recognized by the tree automaton is an open problem. Furthermore, proving that $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite, is itself difficult. This question is undecidable in general [23], but can be answered for some particular E. For instance, if E can be oriented into a TRS $\mathcal{R}$ which is terminating, confluent and such that $\textsc{Irr}(\mathcal{R})$ is finite then $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite [23].

Acknowledgments Many thanks to the referees for their detailed comments.

References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In *CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3. Y. Boichut, B. Boyer, T. Genet, and A. Legay. Equational Abstraction Refinement for Certified Tree Regular Model Checking. In *ICFEM'12*, volume 7635 of *LNCS*. Springer, 2012.

4. Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko. Handling non left-linear rules when completing tree automata. *IJFCS*, 20(5), 2009.
5. Y. Boichut, T. Genet, T. Jensen, and L. Leroux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *RTA*, volume 4533 of *LNCS*, pages 48–62. Springer, 2007.
6. 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.
7. 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.
8. 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.
9. 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.
10. T. Genet. Reachability analysis of rewriting for software verification. Université de Rennes 1, 2009. Habilitation document, `http://www.irisa.fr/celtique/genet/publications.html`.
11. T. Genet. Towards Static Analysis of Functional Programs using Tree Automata Completion. Technical report, INRIA, 2013. `http://hal.archives-ouvertes.fr/hal-00921814/PDF/main.pdf`.
12. T. Genet, T. Le Gall, A. Legay, and V. Murat. A Completion Algorithm for Lattice Tree Automata. In *CIAA'13*, volume 7982 of *LNCS*, pages 134–145, 2013.
13. T. Genet and R. Rusu. Equational tree automata completion. *Journal of Symbolic Computation*, 45:574–597, 2010.
14. T. Genet and Y. Salmon. Tree Automata Completion for Static Analysis of Functional Programs. Technical report, INRIA, 2013. `http://hal.archives-ouvertes.fr/hal-00780124/PDF/main.pdf`.
15. A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. In *RTA'05*, volume 3467 of *LNCS*, pages 353–367. Springer, 2005.
16. 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.
17. N. D. Jones. Flow analysis of lazy higher-order functional programs. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 103–122. Ellis Horwood, Chichester, England, 1987.
18. Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20, 2013.
19. A. Lisitsa. Finite Models vs Tree Automata in Safety Verification. In *RTA'12*, volume 15 of *LIPIcs*, pages 225–239, 2012.
20. F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic Approximation for the Verification of Cryptographic Protocols. In *Proc. of FASE'03*, volume 2629 of *LNCS*, pages 34–48. Springer-Verlag, 2003.
21. L. Ong and S. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *POPL'11*, 2011.
22. T. Takai. A Verification Technique Using Term Rewriting Systems and Abstract Interpretation. In *Proc. 15th RTA Conf., Aachen (Germany)*, volume 3091 of *LNCS*, pages 119–133. Springer, 2004.
23. S. Tison. Finiteness of the set of E-equivalence classes is undecidable, 2010. Private communication.