Termination criteria for tree automata completion

Thomas Genet

Termination Criteria for Tree Automata Completion

Thomas Genet

IRISA/INRIA/Université de Rennes 1
France
Tel.: +33-2-99-84-73-44
Fax: +33-2-99-84-71-71
E-mail: genet@irisa.fr

Abstract

This paper presents two criteria for the termination of tree automata completion. Tree automata completion is a technique for computing a tree automaton recognizing or over-approximating the set of terms reachable w.r.t. a term rewriting system. The first criterion is based on the structure of the term rewriting system itself. We prove that for most of the known classes of linear rewriting systems preserving regularity, the tree automata completion is terminating. Moreover, it outputs a tree automaton recognizing exactly the set of reachable terms. When the term rewriting system is outside of such classes, the set of reachable terms can be approximated using a set of equations defining an abstraction. The second criterion, which holds for any left-linear term rewriting system, defines sufficient restrictions on the set of equations for the tree automata completion to terminate. We then show how to take advantage of this second criterion to use completion as a new static analysis technique for functional programs. Some examples are demonstrated using the Timbuk completion tool.

Keywords: Term Rewriting, Regular Tree Languages, Tree Automata, Regularity Preservation, Tree Automata Completion, Static Analysis, Functional Programming.

1. Introduction

Tree automata completion is an algorithm for computing or approximating the set of reachable terms for a Term Rewriting System $\mathcal{R}$ (TRS for short). Computing or approximating the reachability of TRSs have found various applications ranging from static analysis [1, 2] to cryptographic protocol verification [3, 4] and to termination proofs of TRS [5]. Given an initial language recognized by a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, tree automata completion is able to produce a tree automaton $\mathcal{A}^*$ that recognizes or over-approximates $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$, *i.e.* all terms that can be reached by rewriting terms of $\mathcal{L}(\mathcal{A})$ with $\mathcal{R}$. Then, $\mathcal{A}^*$ can be used to show that terms recognized by another tree automaton $\mathcal{A}_{bad}$ are not reachable by rewriting terms of $\mathcal{L}(\mathcal{A})$ with $\mathcal{R}$, i.e. $\forall s \in \mathcal{L}(\mathcal{A}), \forall t \in \mathcal{L}(\mathcal{A}_{bad})$:

$s \not\to_{\mathcal{R}}{}^* t$. To show this, it is enough to check that $\mathcal{L}(\mathcal{A}^*) \cap \mathcal{L}(\mathcal{A}_{bad}) = \emptyset$, *i.e.* to compute the automaton recognizing the intersection between languages recognized by $\mathcal{A}^*$ and $\mathcal{A}_{bad}$ and check that it recognized an empty language. This has been used to verify the safety of protocols [3, 4] and of Java programs [1].

A strength of the completion algorithm, and at the same time a weakness, is that its precision is parameterized by a function [6] or a set of equations [7]. 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 if the approximation is not strong enough termination is not guaranteed.

1.1. Contributions

In this paper, we define two sufficient conditions for the tree automata completion algorithm to terminate. This paper builds upon preliminary versions of two termination criteria defined in [8] and [9]. This is the result of a large research effort to make tree automata completion competitive with other program verification techniques. Tree automata completion was shown to efficiently handle the verification of complex infinite state systems [3, 4, 1] but there were few results about its termination. Thus, it was essentially used as a semi-algorithm for program verification, except in [4]. In this paper, we define *two conditions* for tree automata completion to terminate. Furthermore, we present some experiments showing that those two results open new ways to *precisely* analyze functional programs.

Condition on the TRS. The first termination condition is on the TRS. For most of the known classes of linear TRS for which the set of reachable terms is regular, completion is guaranteed to terminate and computes exactly the set of reachable terms. We here extend the previous results of [6] with classes defined by Toshinori Takai [10] and Pierre Réty [11]. We also show that completion produces equivalent, but more compact, tree automata than the other known algorithm of [12, 10]. With regards to verification, this first result guarantees the precision of the verification: without approximation, tree automata completion is as precise as possible.

Condition on the set of equations. When approximations are used, they are defined using a set of equations E and a theorem of [7] guarantees that completion introduces no more approximation than what E defines. The second termination condition is on the set of equations E. This condition is essentially syntactic and simple to check.

Alternative static analysis technique for functional programs. Finally, we show that all these results can be used to achieve efficient and *precise* static analysis of functional programs translated into TRS. Using Timbuk [13], which implements completion and equational simplification, we give some examples showing that the obtained technique provides an interesting alternative static analysis technique for functional programs.

1.2. Outline

The outline of this paper is the following. We first introduce some background material on TRS and tree automata. In Section 3, we recall the tree automata completion algorithm and prove a new precision result on it in Section 4. In Section 5, we survey most of the known TRS classes preserving regularity. Then, we show that for most of the known linear classes, tree automata completion terminates and computes exactly the set of reachable terms. This is the first termination criterion: if the TRS is inside one of those classes the tree automata completion terminates. For TRS laying outside of those classes, in Section 6, we define a second criterion based on the structure of the set of approximation equations. This condition is quite restrictive but can be refined into a nicer criterion when the TRS under consideration encodes a functional program. We conclude this part with some experiments showing that this last criterion transforms tree automata completion into an alternative static analysis technique for functional programs. In Section 7, we compare the results presented in this paper with related work. Finally, in Section 8 we conclude and give some further research directions.

2. Background

In this section, we introduce some definitions and concepts that will be used throughout the rest of the paper (see also [14, 15]). 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 $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\lambda\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\}$ 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 $Root(t) = t(\lambda)$

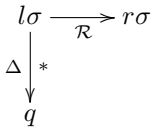
A *term rewriting system* (TRS) $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. right-linear) if each variable occurs 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 $\mathcal{R}^*(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to^*_{\mathcal{R}} t\}$.

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 $\to^*_{\mathcal{R}/E}$ 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 $\mathcal{Q} \cap \mathcal{F} = \emptyset$. Terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ are called *configurations*. A *transition* is a rewrite rule $c \to q$, where c is a configuration and q is state. A transition is *normalized* when $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$ is of arity n, and $q_1, \ldots, q_n \in \mathcal{Q}$. An *$\epsilon$-transition* is a transition of the form $q \to q'$ 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. $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to^{\not\epsilon *}_{\mathcal{A}} q\}$). 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 $s \to_{\mathcal{A}}{}^* q$ and $s \to_{\mathcal{A}}{}^* q'$ then $q = q'$. 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 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 Algorithm

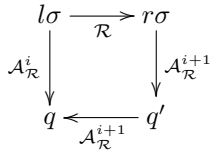
Tree Automata Completion algorithms were proposed in [16, 17, 12, 7, 18]. Tree automata completion is very similar to a Knuth-Bendix completion except that it runs on two distinct sets of rules: a TRS $\mathcal{R}$ and a set of transitions Δ of a tree automaton $\mathcal{A}$. A critical pair is a triple $(l \to r, \sigma, q)$ where $l \to r \in \mathcal{R}$, σ is a substitution mapping variables to states $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and $q \in \mathcal{Q}$ such that



then we know that $l\sigma$ is rewritten by Δ into q (recognized by $\mathcal{A}$) and that $l\sigma$ is rewritten into $r\sigma$ by $\mathcal{R}$. If $r\sigma \not\to^*_\Delta q$ then the critical pair has to be solved for $r\sigma$ to be recognized by Δ into state q. Hence, we need to add the necessary transitions to Δ to have $r\sigma \to^*_\Delta q$. Note that, contrary to Knuth-Bendix completion, there is no choice to make w.r.t. the orientation of the rewrite rule to add since having $q \to^*_\Delta r\sigma$ is not compatible with the standard transition relation of a tree automaton. Then, as in Knuth-Bendix completion, this process is iterated until all critical pairs between $\mathcal{R}$ and Δ can be joined. The complete process is described in the following section.

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 $s \to_\mathcal{R} t$ 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 satisfies $\mathcal{L}(\mathcal{A}^k_\mathcal{R}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. 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 $\to_\mathcal{R}$ and $\to_{\mathcal{A}^i_\mathcal{R}}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to^*_{\mathcal{A}^i_\mathcal{R}} q$ and $r\sigma \not\to^*_{\mathcal{A}^i_\mathcal{R}} q$. 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 $r\sigma \to^*_{\mathcal{A}^{i+1}_\mathcal{R}} q$. In [12, 7], critical pairs are joined in the following way:



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 $\mathcal{A}^i_\mathcal{R}$ to have $r\sigma \to^*_{\mathcal{A}^{i+1}_\mathcal{R}} q$. The first problem, called matching, can be efficiently solved using a specific algorithm [6, 8]. The second problem is solved using Normalization.

3.2. Normalization

The normalization function associates a term with a state appearing in Δ or with a new state, *i.e.* a state of $\mathcal{Q}$ not appearing in Δ. A state q appearing in Δ

is used to normalize a term t if $t \to_{\Delta}^{\not\varepsilon *} q$. Normalizing by reusing transitions of Δ preserves the determinism of $\to_{\Delta}^{\not\varepsilon}$. As we will see, $\to_{\Delta}^{\not\varepsilon}$ can be kept deterministic during completion but $\to_{\Delta}$ cannot.

Definition 1 (New state). Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$. A new state (for Δ) is a state of $\mathcal{Q} \setminus \mathcal{Q}_f$ not occurring in any left or right-hand side of any rule of Δ [1].

We here define normalization as a bottom-up process. This definition is simpler and equivalent to top-down definitions [7]. 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 $f(q_1, \ldots, q_n) \to q' \in \Delta$ or (q' is a new state for Δ and $\forall q'' \in Q$: $f(q_1, \ldots, q_n) \to q'' \notin \Delta$).

We illustrate the above definition on the normalization of a simple transition.

Example 3. *Given* $\Delta = \{b \to q_0\}$, $Norm_{\Delta}(f(g(a), b, g(a)) \to q) = \{a \to q_1, g(q_1) \to q_2, b \to q_0, f(q_2, q_0, q_2) \to q\}$ *where* q_1 *and* q_2 *are new states.*

3.3. One step of completion

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

Definition 4 (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 5 (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 will build a sequence $\mathcal{A}_{\mathcal{R}}^0, \mathcal{A}_{\mathcal{R}}^1, \ldots, \mathcal{A}_{\mathcal{R}}^k$ of automata such that $\mathcal{A}_{\mathcal{R}}^0 = \mathcal{A}$ and 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 [7].

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

Definition 6 (One step automaton 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 7. *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 $\mathcal{R} = \{f(a) \to g(a)\}$ then $CP(\mathcal{R}, \mathcal{A}) = \{(f(a) \to g(a), \sigma_1, q_0)\}$ with $\sigma_1 = \emptyset$ because $f(a)\sigma_1 \to_\mathcal{A}{}^* q_0$ and $f(a)\sigma_1 \to_\mathcal{R} g(a)\sigma_1$. Besides, we have $g(a) \to_\mathcal{A}^{\not\epsilon *} q_2$. Hence $Join^{\{(f(a) \to g(a), \sigma_1, q_0)\}}(\Delta) = Join^\emptyset(\Delta \cup \{q_2 \to q_0\}) = \Delta \cup \{q_2 \to q_0\}$. Thus, $\mathcal{C}_\mathcal{R}(\mathcal{A}) = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \cup \{q_2 \to q_0\} \rangle$;*
- *If $\mathcal{R} = \{f(x) \to x\}$ then $CP(\mathcal{R}, \mathcal{A}) = \{(f(x) \to x, \sigma_2, q_0)\}$ with $\sigma_2 = \{x \mapsto q_1\}$ since $f(x)\sigma_2 \to_\mathcal{A}{}^* q_0$ and $f(x)\sigma_2 \to_\mathcal{R} x\sigma_2 = q_1$. Similarly, from $q_1 \to_\mathcal{A}^{\not\epsilon *} q_1$ we get that $Join^{\{(f(x) \to x, \sigma_2, q_0)\}}(\Delta) = Join^\emptyset(\Delta \cup \{q_1 \to q_0\}) = \Delta \cup \{q_1 \to q_0\}$ and $\mathcal{C}_\mathcal{R}(\mathcal{A}) = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \cup \{q_1 \to q_0\} \rangle$;*
- *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 this time, 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 $\mathcal{C}_\mathcal{R}(\mathcal{A}) = \langle \mathcal{F}, \mathcal{Q} \cup \{q_3\}, \mathcal{Q}_f, \Delta \cup \{f(q_2) \to q_3, q_3 \to q_0\} \rangle$.*

3.4. 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 8 (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 9 (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 $s\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_a$, $t\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_b$, i.e.

$$\begin{array}{ccc} s\sigma & = & t\sigma \\ {\scriptstyle \mathcal{A},\not\epsilon}\downarrow & & \downarrow{\scriptstyle \mathcal{A},\not\epsilon} \\ q_a & & q_b \end{array}$$

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

Example 10. *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 $\Delta = \{a \to q_0, s(q_0) \to q_1, s(q_1) \to q_2, b \to q_3, s(q_3) \to q_4\}$. Hence $\mathcal{L}(\mathcal{A}) = \{s(s(a)), s(b)\}$. We can perform a first simplification step using the equation $s(s(x)) = s(x)$[2], because we found a substitution $\sigma = \{x \mapsto q_0\}$ such that:*

$$\begin{array}{ccc} s(s(q_0)) & \underset{E}{=} & s(q_0) \\ {\scriptstyle \mathcal{A},\not\epsilon}\downarrow & & \downarrow{\scriptstyle \mathcal{A},\not\epsilon} \\ q_2 & & q_1 \end{array}$$

Hence, $\mathcal{A} \leadsto_E \mathcal{A}' = \mathcal{A}\{q_2 \mapsto q_1\}$[3]. 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:

$$\begin{array}{ccc} a & \underset{E}{=} & b \\ {\scriptstyle \mathcal{A},\not\epsilon}\downarrow & & \downarrow{\scriptstyle \mathcal{A},\not\epsilon} \\ q_0 & & q_3 \end{array}$$

We can thus simplify $\mathcal{A}'$ in this way: $\mathcal{A}' \leadsto_E \mathcal{A}'' = \mathcal{A}'\{q_0 \mapsto q_3\}$ 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 $\leadsto_E$ and $\mathcal{L}(\mathcal{A}''') = \{s^*(s(a|b))\}$.

[2]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.

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

As stated in [7] 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 11 (Simplified Tree Automata [7])**.** *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.

3.5. The full Completion Algorithm

Definition 12 (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 note $\mathcal{A}^*_{\mathcal{R},E}$ for $\mathcal{A}^k_{\mathcal{R},E}$.

A good criterion to know that $\mathcal{A}^k_{\mathcal{R},E}$ is a fixpoint is when $CP(\mathcal{R}, \mathcal{A}^k_{\mathcal{R},E}) = \emptyset$. However, a fixpoint cannot always be finitely reached. This objective of this paper is precisely to define sufficient conditions, either on $\mathcal{R}$ or on E, for the fixpoint to be finitely reached.

Example 13. *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.*

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

The automaton $\mathcal{A}^1_{\mathcal{R},E}$ is exactly $\mathcal{C}_\mathcal{R}(\mathcal{A}^0)$ since simplification by equations does 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 $f(q_1, q_2) \to_{\mathcal{A}}{}^* q_3$ and $f(q_1, q_2) \to_\mathcal{R} f(s(q_1), s(q_2))$. 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, $\mathcal{A}^2_{\mathcal{R},E} = \mathcal{C}_\mathcal{R}(\mathcal{A}^1_{\mathcal{R},E})\{q_4 \mapsto q_1, q_5 \mapsto q_2\}$. The last automaton is a fixed point because $CP(\mathcal{R}, \mathcal{A}^2_{\mathcal{R},E}) = \emptyset$.

4. Proving precision of tree automata completion

This section investigates what is the precision of the full completion algorithm. We start from theorems of [7] and show that when completion stops it exactly computes the set of reachable terms. On the one side, we use the fact that completion produces a tree automaton recognizing an over-approximation of reachable terms (Lower bound). On the other side, we recall that the language recognized by the completed tree automaton is upper bounded by the set of $\mathcal{R}/E$ reachable terms. To state this last theorem, we need to first recall the key notion of $\mathcal{R}/E$-coherence.

4.1. Lower and upper bound theorems of completion

Definition 14 (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 $\forall q \in \mathcal{Q} : \exists s \in \mathcal{T}(\mathcal{F})$:

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

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

Now, we can state 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.

Theorem 16 (Lower bound [7])**.** *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*

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

Note that, in the literature, there exists some solutions to remove or weaken the left-linearity condition on $\mathcal{R}$ [12, 6, 19]. If we are interested in the *precision* of the approximation, it can be estimated using the $\mathcal{R}/E$-coherence notion. This will be used in the following to build automata that *exactly* recognize sets of reachable terms.

Theorem 17 (Upper bound [7])**.** *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 the TRS $\mathcal{R}$ models the program and equations E define 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 [7].

4.2. Using completion to compute sets of reachable terms

Our objective is to prove this result by taking advantage of theorems 16 and 17. Theorem 16 could be applied rather straightforwardly on $\mathcal{R}$, $\mathcal{A}$ and $E = \emptyset$ and would give us $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},\emptyset}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$. However, Theorem 17 cannot be applied as is because $\mathcal{A}$ is not necessarily $\mathcal{R}/E$-coherent. Indeed, for $\mathcal{A}$ to be $\mathcal{R}/E$-coherent it is necessary to have $s \to^*_{\mathcal{R}/E} t$ or $t \to^*_{\mathcal{R}/E} s$ for all two terms s, t and all state q such that $s \to_{\mathcal{A}}{}^* q$ and $t \to_{\mathcal{A}}{}^* q$. However, this is not possible in general. As soon as $\mathcal{A}$ recognizes an infinite language with a finite set of states, we necessarily have an infinite number of terms recognized by a state. For instance, assume that $\mathcal{R} = \{f(x) \to g(x)\}$ and $\mathcal{L}(\mathcal{A}) = \{f^*(a)\}$ whatever the transition set of $\mathcal{A}$ may be there is necessarily at least one state q of $\mathcal{A}$ recognizing $f^i(a)$ and $f^j(a)$ with $i \neq j$. However, we do not have $f^i(a) \to^*_{\mathcal{R}/\emptyset} f^j(a)$ nor $f^j(a) \to^*_{\mathcal{R}/\emptyset} f^i(a)$. Many solutions to have an initial $\mathcal{A}$ that is $\mathcal{R}/E$-coherent are possible, but most of them are not satisfactory. For instance, it would be possible to use the Myhill-Nerode theorem of [15] in order to define a set of equations E' such that $\mathcal{A}$ is $\mathcal{R}/E'$-coherent. However, E' may yield an approximation of the set of reachable terms in such a way

that $\mathcal{R}^*_{E'}(\mathcal{L}(\mathcal{A})) \supset \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$[4]. The solution we propose here relies on a simple observation. If the initial language is finite, then it is trivial to define a tree automaton recognizing it and that is $\mathcal{R}/E$-coherent (with $E = \emptyset$). In such an automaton, every state recognizes *exactly* one term (or subterm) of the initial language. If the initial language is not finite, we can *generate* the missing terms by additional rewrite rules. This is the precisely the idea behind the *shifting* automaton $\mathcal{A}^G$ and TRS $\mathcal{R}^G$, that are defined from a tree automaton $\mathcal{A}$ (possibly not $\mathcal{R}/E$-coherent). The automaton $\mathcal{A}^G$ recognizes a finite language which is the set $\{\#_q \mid q \in \mathcal{A}\}$, where the symbols $\#_q$ are new constants, one for each state q of $\mathcal{A}$. Thus $\mathcal{A}^G$ is trivially $\mathcal{R}/E$-coherent. In $\mathcal{R}^G$, we find all the transitions of $\mathcal{A}$ but oriented in the other direction, and thus generating the language $\mathcal{L}(\mathcal{A})$.

Definition 18 (Shifting automaton and TRS). Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and $\mathcal{F}_\mathcal{Q} = \{\#_q \mid q \in \mathcal{Q}\}$ a set of additional symbols such that $\mathcal{F} \cap \mathcal{F}_\mathcal{Q} = \emptyset$. 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 $\Delta^G = \{\#_q \to q \mid q \in \mathcal{Q}\}$
- $\mathcal{R}^G = \{\#_q \to f(\#_{q_1}, \ldots, \#_{q_n}) \mid f(q_1, \ldots, q_n) \to q \in \Delta\} \cup \{\#_{q_2} \to \#_{q_1} \mid q_1 \to q_2 \in \Delta\}$

Lemma 19 ($\mathcal{R}^G$ and $\mathcal{A}^G$ produce $\mathcal{L}(\mathcal{A})$)**.** *For all tree automata $\mathcal{A}$ and state q of $\mathcal{A}$, if $\mathcal{A}^G$, $\mathcal{R}^G$ are the shifting automaton and shifting TRS of $\mathcal{A}$ then $\mathcal{L}(\mathcal{A}, q) = (\mathcal{R}^G)^*(\{\#q\}) \cap \mathcal{T}(\mathcal{F})$ and $\mathcal{L}(\mathcal{A}) = (\mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) \cap \mathcal{T}(\mathcal{F})$.*

Proof. (Sketch) It is enough to show that for all $s \in \mathcal{T}(\mathcal{F})$ and all $q \in \mathcal{Q}$, a derivation $s \to_\mathcal{A}{}^* q$ exists iff there exists $\#_q$ in $\mathcal{L}(\mathcal{A}^G)$ and $\#_q \to^*_{\mathcal{R}^G} s$. □

Lemma 20 (Switching $\mathcal{R}$ and $\mathcal{R}^G$ steps)**.** *Let $\mathcal{R}$ be a linear TRS defined on $\mathcal{F}$, $\mathcal{R}^G$ be a shifting TRS defined on $\mathcal{F} \cup \mathcal{F}_\mathcal{Q}$ and $s, t, u \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$. If $s \to_\mathcal{R} t \to_{\mathcal{R}^G} u$ then there exists $t' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$ such that $s \to_{\mathcal{R}^G} t' \to_\mathcal{R} u$.*

Proof. Assume that $s \to_\mathcal{R} t \to_{\mathcal{R}^G} u$. From $s \to_\mathcal{R} t$ we get that there exists a context $C[\,] \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$, a rewriting rule $l \to r \in \mathcal{R}$ and a substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$ such that $s = C[l\sigma] \to_\mathcal{R} C[r\sigma] = t$. Since $t \to_{\mathcal{R}^G} u$, we know that $C[r\sigma] \to_{\mathcal{R}^G} u$. Since the left-hand sides of rules of $\mathcal{R}^G$ are ground terms of $\mathcal{T}(\mathcal{F}_\mathcal{Q})$, they cannot match r or any subterm of r. As a result the rewriting position for the $\to_{R^G}$ step is either:

[4]For instance, if $\mathcal{A}$ has transitions $\{a \to q_1, f(q_1) \to q_1, g(q_1) \to q_2\}$, E' will have to be defined so that $a = f(a)$, $a = f(f(a))$, ... If $\mathcal{R} = \{g(x) \to h(a), g(x) \to h(f(a))\}$ then $h(a)$ and $h(f(a))$ will be in $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ but E' will also add $h(f^2(a)), h(f^3(a)), \ldots$ that are not in $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

- in $C[\,]$. Then, we have $C[r\sigma] \to_{\mathcal{R}^G} C'[r\sigma]$ with $C'[\,] \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$. Since rules of $\mathcal{R}^G$ are ground rules on $\mathcal{T}(\mathcal{F}_\mathcal{Q})$ they cannot interfere with $r\sigma$ (or $l\sigma$) in $C[r\sigma]$. As a result, it is possible to do $s = C[l\sigma] \to_{\mathcal{R}^G} C'[l\sigma] \to_\mathcal{R} C'[r\sigma]$;
- in a term v substituted by σ. Let x be the variable and $C'[\,]$ the context such that $r = C'[x]$ and $x\sigma = v$. Let v' be the term s.t. $v \to_{\mathcal{R}^G} v'$. We thus have $s = C[l\sigma] \to_\mathcal{R} C[r\sigma] = C[C'[v]\sigma] \to_{\mathcal{R}^G} C[C'[v']\sigma] = u$. Let σ' be the substitution s.t. $\sigma'(y) = \sigma(y)$ for $x \neq y$ and $\sigma'(x) = v'$. Since $\mathcal{R}$ is linear, there is only one x in l and we thus have $s = C[l\sigma] \to_{\mathcal{R}^G} C[l\sigma']$. Finally, since $\mathcal{R}$ is linear, there is only one x in r, and $C[l\sigma'] \to_\mathcal{R} C[r\sigma'] = C[C'[v']\sigma] = u$. We thus have $s = C[l\sigma] \to_{\mathcal{R}^G} C[l\sigma'] \to_\mathcal{R} C[r\sigma'] = u$.

□

Using the shifting operation and completion, computed automata recognizes terms over $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$. However, such automata can be transformed so as to recognize only terms of $\mathcal{T}(\mathcal{F})$: it is enough to remove all transitions with symbols outside of $\mathcal{F}$. This is the simple projection operation we now define.

Definition 21 (The $\mathcal{F}$-Projection, $\Pi_\mathcal{F}$). Let $\mathcal{A} = \langle \mathcal{F}', \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton defined on $\mathcal{F}'$ and $\mathcal{F} \subseteq \mathcal{F}'$. The $\mathcal{F}$-projection of $\mathcal{A}$, denoted by $\Pi_\mathcal{F}(\mathcal{A})$ is the automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_p \rangle$ where $\Delta_p = \{q \to q' \mid q \to q' \in \Delta \text{ and } q, q' \in \mathcal{Q}\} \cup \{f(q_1, \ldots, q_n) \to q \mid f(q_1, \ldots, q_n) \to q \in \Delta \text{ and } f \in \mathcal{F},\ q_1, \ldots, q_n, q \in \mathcal{Q}\}$.

Lemma 22 ($\mathcal{F}$-Projection realizes intersection with $\mathcal{T}(\mathcal{F})$)**.** *If $\mathcal{A}$ is a tree automaton defined on $\mathcal{F}' \supseteq \mathcal{F}$,* i.e. *$\mathcal{A} = \langle \mathcal{F}', \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, then $\mathcal{L}(\Pi_\mathcal{F}(\mathcal{A})) = \mathcal{L}(\mathcal{A}) \cap \mathcal{T}(\mathcal{F}))$.*

Proof. This is proven by an easy induction on the height of terms recognized by $\mathcal{A}$. We also use the fact that projection does not remove epsilon transitions. □

In the following, for a better readability we denote composed sets of reachable terms *e.g.* $\mathcal{R}_1^*(\mathcal{R}_2^*(S))$ using square brackets for the outermost operator, *i.e.* $\mathcal{R}_1^*[\mathcal{R}_2^*(S)]$. The next lemma states that, when using shifting on a tree automaton $\mathcal{A}$, it is equivalent to perform the intersection with $\mathcal{T}(\mathcal{F})$ before and after the application of rules of $\mathcal{R}$ provided that $\mathcal{A}$ is reduced, *i.e.* that all its states are reachable. We first show on an example why $\mathcal{A}$ needs to be reduced.

Example 23. *Assume that $\mathcal{A}$ is the tree automaton with set of transitions $\Delta = \{f(q_1, q_2) \to q, a \to q_2\}$ and $\mathcal{R} = \{f(x, y) \to g(y)\}$. Then $\mathcal{A}^G$ has transitions $\{\#q_1 \to q_1, \#q_2 \to q_2\}$ and $\mathcal{R}^G = \{\#q_2 \to a, \#q \to f(\#q_1, \#q_2)\}$. $\mathcal{R}^*[\mathcal{R}^{G*}(\{\#q\})] \cap \mathcal{T}(\mathcal{F}) = \{g(a)\}$ but $\mathcal{R}^*[\mathcal{R}^{G*}(\{\#q\}) \cap \mathcal{T}(\mathcal{F})] = \emptyset$, because $\#q \to_G^* f(\#q_1, a)$ and $f(\#q_1, a)$ does not belong to $\mathcal{T}(\mathcal{F})$ but $f(\#q_1, a) \to_\mathcal{R} g(a)$ that belongs to $\mathcal{T}(\mathcal{F})$.*

Lemma 24. *Let $\mathcal{R}$ be a TRS and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton. Let $\mathcal{A}^G$ be the shifting automaton of $\mathcal{A}$, $\mathcal{R}^G$ its shifting TRS, and $\mathcal{F}_\mathcal{Q}$ the set of additional symbols. Let $S \subseteq \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$. If all states of $\mathcal{A}$ are reachable then*

$$\mathcal{R}^*[\mathcal{R}^{G*}(S)] \cap \mathcal{T}(\mathcal{F}) = \mathcal{R}^*[\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})]$$

Proof. The fact that $\mathcal{R}^*[\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})] \subseteq \mathcal{R}^*[\mathcal{R}^{G*}(S)] \cap \mathcal{T}(\mathcal{F})$ trivially holds. Let us prove that the opposite inclusion is also true. We make a proof by contradiction. Assume that there exists a term $t \in \mathcal{R}^*(\mathcal{R}^{G*}(S)) \cap \mathcal{T}(\mathcal{F})$ such that $t \notin \mathcal{R}^*[\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})]$. Such a term exists if there is a term $s \in S$ such that $s \to_{\mathcal{R}^G}^* s'$, $s' \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_\mathcal{Q})$, $s' \notin \mathcal{T}(\mathcal{F})$, $s' \to_\mathcal{R}{}^* t$ and $t \in \mathcal{T}(\mathcal{F})$. We know that $s' \notin \mathcal{T}(\mathcal{F})$ because, otherwise, it would belong to $\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})$ and, since $s' \to_\mathcal{R}{}^* t$, t would be in $\mathcal{R}^*[\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})]$. Since s' does not to belong to $\mathcal{T}(\mathcal{F})$, we know that it is of the form $C[\#q_1, \ldots, \#q_n]$ where $C[\,]$ is a ground context of $\mathcal{T}(\mathcal{F})$ and $\#q_1, \ldots, \#q_n \in \mathcal{F}_\mathcal{Q}$. Since $s' = C[\#q_1, \ldots, \#q_n] \to_\mathcal{R}{}^* t \in \mathcal{T}(\mathcal{F})$ and rules of $\mathcal{R}$ are defined on $\mathcal{F}$, we can infer that this rewriting derivation is independent of $\#q_1, \ldots, \#q_n \in \mathcal{F}_\mathcal{Q}$ and would be the same for any term replacing $\#q_1, \ldots, \#q_n$. Let us choose, in particular, $t_i \in \mathcal{L}(\mathcal{A}, q_i)$ ($t_i \in \mathcal{T}(\mathcal{F})$). Such terms exist because all states of $\mathcal{A}$ are reachable. We thus have $C[t_1, \ldots, t_n] \to_\mathcal{R}{}^* t$ and $C[t_1, \ldots, t_n] \in \mathcal{T}(\mathcal{F})$. From Lemma 19 and $t_i \in \mathcal{L}(\mathcal{A}, q_i)$, we get that $t_i \in (\mathcal{R}^G)^*(\{\#q_i\})$. Thus $C[\#q_1, \ldots, \#q_n] \to_{\mathcal{R}^G}^* C[t_1, \ldots, t_n] \in \mathcal{T}(\mathcal{F})$ which implies that $C[t_1, \ldots, t_n] \in \mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})$. Finally, since $C[t_1, \ldots, t_n] \to_\mathcal{R}{}^* t$, we obtain that $t \in \mathcal{R}^*[\mathcal{R}^{G*}(S) \cap \mathcal{T}(\mathcal{F})]$ which is a contradiction. □

The following theorem gives a sufficient condition for completion to build a tree automaton recognizing the set of reachable terms for any linear TRS $\mathcal{R}$ and any regular set of initial terms recognized by a tree automaton $\mathcal{A}$. Note that all states of $\mathcal{A}$ have to be reachable. This property is easy to check or to enforce while keeping the same recognized language by reducing $\mathcal{A}$ to its set of reachable states [15]. Note also that this theorem assumes that completion terminates. Several conditions on $\mathcal{R}$ and $\mathcal{A}$ sufficient for completion to terminate will be given in the next section. For the next precision theorem to hold, $\mathcal{R}$ needs to be linear. We first illustrate the need for left and right linearity of $\mathcal{R}$ with an example.

Example 25. *Let $\mathcal{R} = \{f(x, x) \to g(x)\}$ and $\mathcal{A}$ be the tree automaton with (non deterministic) set of transitions $\{f(q_1, q_2) \to q_0, a \to q_1, a \to q_2\}$. When running completion on this TRS no critical pair is found because there is no substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and no state q such that $f(x, x)\sigma \to_\mathcal{A}{}^* q$. However, $\mathcal{A}$ recognizes $f(a, a)$ that rewrites to $g(a)$ which is not recognized by the completed automaton (which remains $\mathcal{A}$). For right-linearity, if $\mathcal{R} = \{f(x) \to g(x, x)\}$ and $\mathcal{A}$ is the tree automaton with set of transitions $\{f(q_1) \to q_0, a \to q_1, b \to q_1\}$, then running completion on $\mathcal{A}$ we will add the transitions $g(q_1, q_1) \to q_2$ and $q_2 \to q_0$ to $\mathcal{A}$ to obtain the completed automaton. Using those transitions it is*

thus possible to recognize terms $g(a,a)$ and $g(b,b)$ which are valid successors of $f(a)$ and $f(b)$ respectively, but also $g(a,b)$ and $g(b,a)$ which are not.

Theorem 26 (Precision)**.** *Let $\mathcal{R}$ be a linear TRS and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton such that all states of $\mathcal{Q}$ are reachable. If completion of $\mathcal{A}^G$, $\mathcal{R} \cup \mathcal{R}^G$ and $E = \emptyset$ stops on automaton $(\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}$, then*

$$\mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

Proof. By applying Theorem 16, on automaton $\mathcal{A}^G$, TRS $\mathcal{R}\cup\mathcal{R}^G$ and $E = \emptyset$, we get that $\mathcal{L}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}) \supseteq (\mathcal{R}\cup\mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G))$. Besides, the shifting automaton $\mathcal{A}^G$ is $\mathcal{R}/E$-coherent for any $\mathcal{R}$ and E since its recognized language is a finite set of constants of $\mathcal{F}_\mathcal{Q}$ and each of them is recognized by a distinct state. We can thus apply Theorem 17 on automaton $\mathcal{A}^G$, TRS $\mathcal{R} \cup R^G$ and $E = \emptyset$ and obtain that $\mathcal{L}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}) \subseteq (\mathcal{R}\cup\mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G))$. Combining the two inequalities, we thus have that

$$(1)\ \mathcal{L}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}) = (\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G))$$

Then, we can restrict both sides of this inequality to $\mathcal{T}(\mathcal{F})$ and obtain:

$$(2)\ (\mathcal{L}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}) \cap \mathcal{T}(\mathcal{F})) = ((\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) \cap \mathcal{T}(\mathcal{F})$$

Using Lemma 22 on automaton $(\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}$, we get that $\mathcal{L}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}) \cap \mathcal{T}(\mathcal{F}) = \mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}))$. The equality thus becomes:

$$(3)\ \mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})) = ((\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) \cap \mathcal{T}(\mathcal{F})$$

To simplify the right-hand side, we can first remark that by definition of sets of reachable terms $(\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) \supseteq \mathcal{R}^*(\mathcal{R}^{G*}(\mathcal{L}(\mathcal{A}^G)))$. Moreover, using Lemma 20, we can iteratively move all the $\mathcal{R}^G$ steps at the beginning of all the rewriting derivations and get that $(\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) \subseteq \mathcal{R}^*(\mathcal{R}^{G*}(\mathcal{L}(\mathcal{A}^G)))$. Combining the two we get that $(\mathcal{R} \cup \mathcal{R}^G)^*(\mathcal{L}(\mathcal{A}^G)) = \mathcal{R}^*(\mathcal{R}^{G*}(\mathcal{L}(\mathcal{A}^G)))$. The equality (3) can thus be simplified into:

$$(4)\ \mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})) = \mathcal{R}^*[\mathcal{R}^{G*}(\mathcal{L}(\mathcal{A}^G))] \cap \mathcal{T}(\mathcal{F})$$

Provided that states of $\mathcal{A}$ are reachable, we can apply Lemma 24 and simplify (4) by moving the intersection with $\mathcal{T}(\mathcal{F})$ into the brackets:

$$(5)\ \mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})) = \mathcal{R}^*[\mathcal{R}^{G*}(\mathcal{L}(\mathcal{A}^G)) \cap \mathcal{T}(\mathcal{F})]$$

Finally, Lemma 19 ensures that $(\mathcal{R}^{G\,*}(\mathcal{L}(\mathcal{A}^G)) \cap \mathcal{T}(\mathcal{F})) = \mathcal{L}(\mathcal{A})$ and thus:

$$\mathcal{L}(\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

□

5. Termination for TRS classes preserving regularity

In this section, we first review known classes of TRSs for which the image $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ of a regular language $\mathcal{L}(\mathcal{A})$ by a TRS $\mathcal{R}$ is regular. These classes are also known as *classes of TRS preserving the regularity*. In [6], it was shown that completion, when used with a specific normalization strategy, covers 3 of those classes. In this work, we extend this result to 7 of the 11 known classes. Contrary to [6], we choose to keep the completion algorithm unchanged and, instead, use the shifting transformation defined in the previous section. We have shown above that, when completion terminates on those transformed automaton and TRS, it computes exactly the set of reachable terms (*i.e.* $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$). Now, we additionally show that when $\mathcal{R}$ belongs to all linear classes of TRS preserving regularity then it terminates. We first recall some of the known TRS classes preserving regularity.

5.1. Term rewriting systems preserving the regularity

Given a TRS $\mathcal{R}$ and a regular language $\mathcal{S}$, deciding whether $\mathcal{R}^*(\mathcal{S})$ is regular is not possible in general, even if $\mathcal{R}$ is a confluent and terminating linear TRS [20]. Thus, many results aim at defining classes of TRS $\mathcal{R}$ so that $\mathcal{R}^*(\mathcal{S})$ is regular. We now survey most of those classes. Note that we choose not to consider classes where regularity is preserved only if the TRS is applied under a specific strategy such as [21, 22]. First, we present the classes which can be defined with simple syntactic restrictions on $\mathcal{R}$.

5.1.1. Classes defined by a simple syntactical criterion

G: $\mathcal{R}$ is a ground TRS, i.e. all rules are of the form $l \to r$ where l and r are ground terms [23, 24].

RL-M: $\mathcal{R}$ is a right-linear and monadic TRS [25], i.e. right-hand sides of the rules of $\mathcal{R}$ are either variables or terms of the form $f(x_1, \ldots, x_n)$ where $f \in \mathcal{F}$ and $x_1, \ldots, x_n$ are variables.

L-SM: $\mathcal{R}$ is a linear and semi-monadic TRS [26], i.e. rules are linear and their right-hand sides are of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ and t_i $(1 \leq i \leq n)$ is either a variable or a ground term.

L-G: In [16], F. Jacquemard defines the class **L-G** of linear "growing" TRS, where "growing" means that every left-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $\mathcal{A}r(f) = n$, and for all $i = 1, \ldots, n$ the term t_i is a variable, a ground term, or a term whose variables do not occur in the right-hand side. F. Jacquemard also shows that if $\mathcal{R}$ is growing then $(\mathcal{R}^{-1})^*(\mathcal{S})$ was regular. Those classes are essentially used for needness analysis of redex in rewriting, see for instance [27]. In order to compare this class with all the others on $\mathcal{R}^*(\mathcal{S})$, we can define the **L-G**$^{-1}$ class using the restrictions in the other direction. The **L-G**$^{-1}$ class corresponds to linear TRS where the *right-hand side* is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $\mathcal{A}r(f) = n$, and for all $i = 1, \ldots, n$

the term t_i is a variable, a ground term, or a term whose variables do not occur in the *left-hand side*. Thus, note that in this class the usual variable restriction on rewrite rules, i.e. $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$ does not hold.

RL-G^{-1}: similar to **L-G^{-1}** except that left-linearity is not required. This result was proved by Nagaya and Toyama in [28].

L-IOSLT: $\mathcal{R}$ is a linear I/O separated layered transducing TRS [29]. Those TRS are defined on sets of symbols $\mathcal{F}_i$, $\mathcal{F}_o$ and P such that $\forall p \in P' : \mathcal{A}r(p) = 1$ and $\mathcal{F}_i$, $\mathcal{F}_o$ and P are disjoint. Symbols of $\mathcal{F}_i$ are input symbols and those of $\mathcal{F}_o$ are output symbols. In the TRS, all the rewrite rules are linear and of the form:

- $f_i(p_1(x_1), \ldots, p_n(x_n)) \to p(t_o)$, or
- $p'_1(x_1) \to p'(t'_o)$

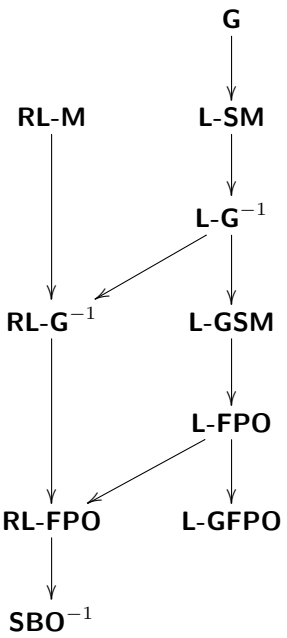
where $f_i \in \mathcal{F}_i$, $p_1, \ldots, p_n, p, p'_1, p' \in P$, $x_1, \ldots, x_n$ are disjoint variables, $t_o, t'_o \in \mathcal{T}(\mathcal{F}_o, \mathcal{X})$ such that $\mathcal{V}ar(t_o) \subseteq \{x_1, \ldots, x_n\}$ and $\mathcal{V}ar(t'_o) \subseteq \{x_1\}$. This class corresponds to linear tree transducers.

There exists more general classes such as **L-GSM** linear generalized semi-monadic TRS [30], **RL-FPO** right-linear and finite-path overlapping TRS, **L-FPO** linear finite-path overlapping TRS [12], **L-GFPO** linear generalized finite-path overlapping TRS [10] and more recently bounded TRS **SBO^{-1}** [31]. However, the regularity criteria used in those classes are more sophisticated and cannot be expressed as a simple syntactic restriction like above. They are based on a careful inspection of the syntactic structure of rewrite rules so that recursive application of rewrite rules are guaranteed to preserve regularity. Since we are going to use the **RL-FPO** and **RL-GFPO** criteria, we detail them in Section 5.1.2.

P. Réty [11] proposed another way of considering the problem and defined a class where restrictions are weaker on the TRS and stronger on the regular language $\mathcal{S}$. Since this class imposes restrictions on the language $\mathcal{S}$ it is thus incomparable with previous ones. We call this class **Constructor**. The alphabet $\mathcal{F}$ is separated into a set of *defined symbols* $\mathcal{D} = \{f \mid \exists l \to r \in \mathcal{R} \text{ s.t. } \mathcal{R}oot(l) = f\}$ and *constructor symbols* $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. The restriction on $\mathcal{S}$ is the following: $\mathcal{S}$ is a regular set of ground constructor instances of a linear term t, i.e. $\mathcal{S} = \{t\sigma \mid \sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear. The restrictions on $\mathcal{R}$ are the following: for each rule $l \to r$

1. r is linear, and
2. for each position $p \in \mathcal{P}os_{\mathcal{F}}(r)$ such that $r|_p = f(t_1, \ldots, t_n)$ and $f \in \mathcal{D}$ we have that for all $i = 1 \ldots n$, t_i is a variable or a ground term, and
3. there is no nested function symbol in r

The expressiveness of all the above mentioned classes can be ordered like in Figure 1.



and **L-IOSLT** / **Constructor** are incomparable with others.

Figure 1: Expressiveness TRS classes preserving regularity, where $A \to B$ means that class A is included in class B.

5.1.2. The FPO and GFPO criteria and algorithms

As seen above, **RL-FPO** and **L-GPFO** are some of the most general classes of TRS $\mathcal{R}$ such that $\mathcal{R}^*(\mathcal{S})$ is regular if $\mathcal{S}$ is. The criteria and algorithms associated to those classes are complex. However, since we want to prove that those classes are covered by completion, we need to present their principles in details. The **FPO** and **GFPO** criteria are based on the notion of *sticking-out*. Roughly speaking, a term s sticks out of a term t if they have in common a position p such that (1) all symbols encountered on the path from ϵ to p are the same in s and t, and (2) $t|_p$ is a variable and $s|_p$ is either a variable or a non ground term. This is formally defined as follows [12].

Definition 27. A term s *sticks-out* of a term t at position p with $p \in (\mathcal{P}os_{\mathcal{X}}(t) \cap \mathcal{P}os(s) \setminus \{\lambda\})$ if

- $\forall o : \lambda \preceq o \prec p \wedge o \in \mathcal{P}os(s) \implies s(o) = t(o)$, and
- $s|_p \notin \mathcal{T}(\mathcal{F})$.

Additionally, a term s *properly sticks-out* of a term t at position p it $s|_p$ is not a variable.

Example 28. *Assume that* $\mathcal{F} = \{f, g, a, b\}$ *and* $\mathcal{X} = \{x, y, z, u\}$. *The term* $f(x, g(b))$ *sticks out of term* $f(y, a)$ *at position* $1.\lambda$. *Note that* $1.\lambda$ *is a variable*

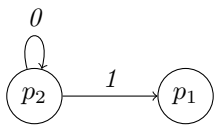
position of $f(y,a)$ and a position of $f(x,g(b))$ and all symbols on the path from positions λ to $1.\lambda$ are similar (here f only). Similarly, $f(f(a,x),y)$ properly sticks-out of $f(z,g(u))$ at position $1.\lambda$ because it sticks-out and because subterm $f(a,x)$ is not a variable.

The definition of sticking-out graphs [12] does not make the usual assumption that $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$ for rewrite rules $l \to r$. In their paper, this makes sense because rewrite systems can be used in the reverse direction to compute the ancestors. Though in the following we only consider TRS such that $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$, we choose to stick to their original definition.

Definition 29 (Sticking-out graph). The *sticking-out graph* of a TRS $\mathcal{R}$ is a directed graph $G = (V, E)$ where $V = \mathcal{R}$, the vertices are the rules of $\mathcal{R}$, and the set E is defined as follows. Let v_1 and v_2 be, possibly identical, vertices which corresponds to rewrite rules $l_1 \to r_1$ and $l_2 \to r_2$ respectively. For i=1,2, replace each variable in $\mathcal{V}ar(r_i) \setminus \mathcal{V}ar(l_i)$ by a fresh constant symbol $\blacklozenge$.

1. If r_2 properly sticks-out of a subterm of l_1, then E contains an edge from v_2 to v_1 with weight one.
2. If a subterm of r_2 properly sticks-out of l_1, then E contains an edge from v_2 to v_1 with weight one.
3. If a subterm of l_1 sticks-out of r_2, then E contains an edge from v_2 to v_1 with weight zero.
4. If l_1 sticks-out of a subterm of r_2, then E contains an edge from v_2 to v_1 with weight zero.

Example 30. *Let $\mathcal{F} = \{f, g, a, b\}$, $\mathcal{X} = \{x, y\}$ and $\mathcal{R} = \{p_1 = f(x,a) \to f(h(y),x), p_2 = g(y) \to f(g(y),b)$ where for $i = 1, 2$ l_i (resp. r_i) denotes the left (resp. right)-hand side of p_i. Since y occurs in r_1 but not in l_1 it is replaced by the $\blacklozenge$ constant. Since $r_2 = f(g(y),b)$ properly sticks out of $f(x,a)$ at position $1.\lambda$, in E we have an edge between p_2 and p_1 of weight 1. Then, since $l_2 = g(y)$ sticks-out of $r_2 = f(g(y),b)$ at position $1.\lambda$, in E we have an cyclic edge of weight 0 on p_2. Note that there is no cyclic edge on p_1 because $r_1 = f(h(\blacklozenge),x)$ does not sticks-out of $l_1 = f(x,a)$ at position $1.\lambda$ because $r_1|_{1.\lambda} = h(\blacklozenge)$ which is a ground term. The complete sticking-out graph is thus the following:*



Definition 31. A TRS is *Finite Path Overlapping* (FPO) if its sticking-out graph has no cycle of weight 1 or more.

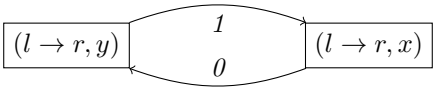
Theorem 32 (**RL-FPO** TRS preserve regularity [12])**.** *If $\mathcal{S}$ is a regular language $\mathcal{S}$ and $\mathcal{R}$ is a right-linear* **FPO** *(**RL-FPO**) TRS then $\mathcal{R}^*(\mathcal{S})$ is regular.*

When dealing with linear TRS, the criterion can be improved as shown in [10]. It is based on the notion of *Generalized sticking-out graph* we now define.

Definition 33 (Generalized sticking-out graph). The *generalized sticking-out graph* of a TRS $\mathcal{R}$ is a directed graph $G = (V, E)$. The set V of vertices is defined by $V = \{(l \to r, x) \mid l \to r \in \mathcal{R} \text{ and } x \in \mathcal{V}ar(l) \cup \mathcal{V}ar(r)\}$. The set E of edges is defined as follows. Let $l_1 \to r_1$ and $l_2 \to r_2$ be two rules of $\mathcal{R}$, possibly identical.

1. If r_2 properly sticks-out at position p of $l_1|_{p'}$ with $p' \in \mathcal{P}os(l_1)$, then for $y = l_1|_{p.p'}$ and for all variables $x \in \mathcal{V}ar(r_2|_p)$, E contains edges from $(l_2 \to r_2, x)$ to $(l_1 \to r_1, y)$ with weight one.
2. If $l_1|_{p'}$ with $p' \in \mathcal{P}os(l_1)$ sticks-out of r_2 at position p, then for $x = r_2|_p$ and for all variables $y \in \mathcal{V}ar(l_1|_{p.p'})$, E contains edges from $(l_2 \to r_2, x)$ to $(l_1 \to r_1, y)$ with weight zero.
3. If $r_2|_{p'}$ with $p' \in \mathcal{P}os(r_2)$ properly sticks-out of l_1 at position p, then for $y = l_1|_p$ and for all variables $x \in \mathcal{V}ar(r_2|_{p.p'})$, E contains edges from $(l_2 \to r_2, x)$ to $(l_1 \to r_1, y)$ with weight one.
4. If l_1 sticks-out of $r_2|_{p'}$ at position p, then for $x = r_2|_{p.p'}$ and for all variables $y \in \mathcal{V}ar(l_1|_p)$, E contains edges from $(l_2 \to r_2, x)$ to $(l_1 \to r_1, y)$ with weight zero.

Example 34. *Let $\mathcal{R} = \{h(f(x, h(g(y)))) \to f(g(k(y)), h(x))\}$, where $l = h(f(x, h(g(y))))$ and $r = f(g(k(y)), h(x))$. With $p' = 1.\lambda$, we have that $l|_{1.\lambda} = f(x, h(g(y)))$ (properly) sticks-out of $r = f(g(k(y)), h(x))$ at position $p = 2.1.\lambda$. We are in the second case of the previous definition. We thus get that there are edges between $r|_p = x$ and all variables of $l|_{p'.p}$ which is the set $\{y\}$. Hence we have an edge between $(l \to r, x)$ and $(l \to r, y)$ of weight 0. Symmetrically, again with $p' = 1.\lambda$, $r = f(g(k(y)), h(x))$ properly sticks-out of $l|_{p'} = f(x, h(g(y)))$ at position $p = 1.\lambda$. This corresponds to the first case of the previous definition. Hence, there are edges between all variables of $\mathcal{V}ar(r|_p) = \{y\}$ and $x = l|_{p'.p}$. Thus, there is an edge between $(l \to r, y)$ and $(l \to r, x)$ of weight 1. Here is the complete generalized sticking-out graph:*



Definition 35. A TRS is *Generalized Finite Path Overlapping* (GFPO) if its generalized sticking-out graph has no cycle of weight 1 or more.

Theorem 36 (**L-GFPO** TRS preserve regularity [10])**.** *If $\mathcal{S}$ is a regular language $\mathcal{S}$ and $\mathcal{R}$ is a linear* **GFPO** *(**L-GFPO**) TRS then $\mathcal{R}^*(\mathcal{S})$ is regular.*

Theorem 37 (Expressiveness of Linear GFPO w.r.t. Linear FPO [10])**.** **L-GFPO** $\supset$ **L-FPO**.

The algorithms defined in [12] and in [10] rely on the notion of *packed state* (named structured state in [10]). For linear TRS, the notion of packed state can be simplified as follows.

Definition 38 (Packed state and Packed automaton). For a set of symbols $\mathcal{F}$ and a set of states $\mathcal{Q}$, the set of packed states, denoted by $\mathcal{P}_{\mathcal{F},\mathcal{Q}}$, is defined by:

- if $q \in \mathcal{Q}$ then $\langle q \rangle \in \mathcal{P}_{\mathcal{F},\mathcal{Q}}$, and
- if $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{P}_{\mathcal{F},\mathcal{Q}})$ then $\langle t \rangle \in \mathcal{P}_{\mathcal{F},\mathcal{Q}}$.

Note that, the second case of the definition is more general than the original definition of [12] where a packed state $\langle f(p_1, \ldots, p_n) \rangle$ exists only if $f \in \mathcal{F}$ and $p_1, \ldots, p_n$ are packed states. Indeed, we think their definition is not adapted to their algorithm. In particular, to solve a critical pair $(l \to r, \sigma, q)$, their algorithm defines a packed state $\langle r\sigma \rangle$ where σ associates variables to packed states. Thus packed states of the form $\langle f(g(\langle q_2 \rangle), a) \rangle$ are likely to occur but are not covered by their definition. As matter of a fact, packed states can be seen as a simple mechanism to get a unique state name from a given term. If $\mathcal{A}$ is a tree automaton and Δ its set of transitions, the *packed automaton* $pack(\mathcal{A})$ is the tree automaton where, for all states $q \in \mathcal{A}$, all occurrences of q in $\mathcal{Q}, Q_f$ and Δ are replaced by $\langle q \rangle$. We also denote by $pack(\Delta)$ the set of packed transitions of $pack(\mathcal{A})$.

Example 39. *Let $\mathcal{F} = \{f : 1, g : 2, a : 0\}$ and $\mathcal{Q} = \{q_1, q_2\}$. Here are some possible packed states: $\langle q_1 \rangle$, $\langle a \rangle$, $\langle f(\langle q_1 \rangle) \rangle$, $\langle f(g(\langle q_2 \rangle), a) \rangle$.*

Then, the principle of the algorithm is very close to completion. Starting from $\mathcal{R}$ and $pack(\mathcal{A})$, for all rewrite rules $l \to r \in \mathcal{R}$ and all packed states q, it searches for state substitutions $\sigma : \mathcal{X} \mapsto \mathcal{P}_{\mathcal{F},\mathcal{Q}}$ such that $l\sigma \to^*_{pack(\mathcal{A})} q$ and adds necessary transitions to have $r\sigma \to^*_{pack(\mathcal{A})} q$. The only difference with the completion algorithm lays in how transitions are normalized. Their algorithm adds the epsilon transition $\langle r\sigma \rangle \to q$ and the transition $r\sigma \to \langle r\sigma \rangle$ is normalized using packed states, instead of new states, by calling the following procedure on the term $r\sigma$.

Procedure addtrans(t) This procedure takes a term t of $\mathcal{T}(\mathcal{F} \cup \mathcal{P}_{\mathcal{F},\mathcal{Q}})$ as input and adds new packed states to $\mathcal{P}_{\mathcal{F},\mathcal{Q}}$ and new transitions to $pack(\Delta)$.

1. if $p = c$ with c constant of $\mathcal{F}$, then define $c \to \langle c \rangle$ as a transition of Δ;
2. if $p = f(p_1, \ldots, p_n)$ with $f \in \mathcal{F}$, then define $f(p'_1, \ldots, p'_n) \to \langle p \rangle$ as a transition of Δ where $p'_i = p_i$ if $p_i \in \mathcal{P}_{\mathcal{F},\mathcal{Q}}$, otherwise $p'_i = \langle p_i \rangle$ for $1 \leq i \leq n$ and execute addtrans(p'_i) for $1 \leq i \leq n$.

Again, the original procedure of [12] has another case for packed states containing several states, which is not necessary here since we consider only left-linear TRSs.

Example 40. *If we call* addtrans$(f(\langle q\rangle, f(a,b)))$ *on* $\mathcal{Q} = \{\langle q\rangle\}$ *and* $\Delta = \{a \to \langle q\rangle\}$*, then* $\mathcal{Q}$ *becomes* $\{\langle q\rangle, \langle a\rangle, \langle b\rangle, \langle f(\langle a\rangle, \langle b\rangle)\rangle, \langle f(\langle q\rangle, \langle f(\langle a\rangle, \langle b\rangle)\rangle)\rangle\}$ *and* $\Delta = \{a \to \langle q\rangle, a \to \langle a\rangle, b \to \langle b\rangle, f(\langle a\rangle, \langle b\rangle) \to \langle f(\langle a\rangle, \langle b\rangle)\rangle, f(\langle q\rangle, \langle f(\langle a\rangle, \langle b\rangle)) \to \langle f(\langle q\rangle, \langle f(\langle a\rangle, \langle b\rangle)\rangle)\rangle\}$.

5.2. Completion preserves $\not\epsilon$-determinism, injectivity and $\not\epsilon$-reducibility

To prove the termination results, we first need to prove structural properties about the completed tree automata. Namely, they are $\not\epsilon$-deterministic, $\not\epsilon$-reducible and their sets of transitions are injective. Proofs of those technical lemmas can be found in Appendix A.

Lemma 41 (Completion preserves $\not\epsilon$-determinism)**.** *Let $\mathcal{R}$ be a TRS and $\mathcal{A}$ an $\not\epsilon$-deterministic automaton. For all $i \in \mathbb{N}$, $\mathcal{A}^i_{\mathcal{R},\emptyset}$ is $\not\epsilon$-deterministic.*

As we will see in the following, the completion defined in [12, 10], does not enjoy this property. Note that from any initial tree automaton $\mathcal{A}$ it is possible to obtain an equivalent deterministic automaton [15], which is trivially $\not\epsilon$-deterministic. Another strong property of sets of transitions produced by completion is the injectivity. Preservation of injectivity will be necessary to prove termination of the completion procedure for TRSs in the **L-GFPO** class.

Definition 42 (Injective sets of transitions). Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be a tree automaton, $q \in \mathcal{Q}$ and $c, c' \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$. The set Δ is *injective* if for all (normalized) transitions $c \to q$ and $c' \to q$ of Δ, we have $c = c'$. Note that injectivity does not depend on ϵ-transitions.

Lemma 43 (Completion preserves injectivity)**.** *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ be a tree automaton whose set of transitions is injective. For all $i \in \mathbb{N}$, the set of transitions of $\mathcal{A}^i_{\mathcal{R},\emptyset}$ is injective.*

Lemma 44 (Completion preserves $\not\epsilon$-reducibility)**.** *Let $\mathcal{R}$ be a TRS, E a set of equations and $\mathcal{A}$ a $\not\epsilon$-reduced tree automaton. For any $i \in \mathbb{N}$: $\mathcal{A}^n_{\mathcal{R},E}$ is $\not\epsilon$-reduced.*

5.3. Termination Proof for the **Constructor** *class*

We prove that the completion terminates on the **Constructor** class. The principle of the proof is to show that the number of critical pairs to solve is finite, and that this entails termination of completion. The proof relies on the fact that for $k \in \mathbb{N}$, $(\mathcal{A}^G)^k_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}$ is an $\not\epsilon$-deterministic tree automaton. This is guaranteed by the fact that $\mathcal{A}^G$ is trivially $\not\epsilon$-deterministic and that completion with $E = \emptyset$ preserves this property (Lemma 41).

Theorem 45 (Termination on the **Constructor** class)**.** *Let $\mathcal{R}$ be a* **Constructor** *TRS and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be a tree automaton. If R^G and $\mathcal{A}^G$ are the shifting TRS and automaton of $\mathcal{A}$ then completion of $\mathcal{A}^G$ with $\mathcal{R} \cup \mathcal{R}^G$ and $E = \emptyset$ terminates.*

Proof. For **Constructor**, recall that there is also a restriction on the initial language $\mathcal{S} = \mathcal{L}(\mathcal{A}) = \{t\mu \mid \mu : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear. Let $\mathcal{Q}_{arg}$ be the set containing states of $\mathcal{A}^G$ and the new states used to normalize all the ground subterms of the right-hand sides of the rules of $\mathcal{R} \cup \mathcal{R}^G$. Now, let us prove that during any completion step, when solving any critical pair $(l \to r, \sigma, q)$, σ always range over this (finite) set $\mathcal{Q}_{arg}$. First, note that for a state q to appear in a substitution of a critical pair, it needs to occur under a defined symbol f, *i.e.* $f(\ldots, C'[q], \ldots) \to_{\mathcal{A}}{}^* q'$. More precisely, the left-hand side of the critical pair is necessarily of the form $C[x]\sigma$ where $C[\,]$ is a non empty context of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\mathit{Root}(C[\,]) \in \mathcal{D}$, $x \in \mathcal{X}$ and σ is a substitution such that $\sigma(x) = q$. Initially, this property is trivially true: in $\mathcal{A}^G$ all defined symbols $\#q \in \mathcal{F}_{\mathcal{Q}}$ are of arity 0. The first completion step necessarily uses only rules of $\mathcal{R}^G$. Because of the form of the rules of $\mathcal{R}^G$, this first completion step will only add the transitions necessary to recognize the right-hand sides of the rules, *i.e.* terms of the form $f(\#q_1, \ldots, \#q_n)$ with $n \in \mathbb{N}$, $f \in \mathcal{F}$ and $\#q_1, \ldots, \#q_n \in \mathcal{F}_{\mathcal{Q}}$. The added transitions will thus be of the form $f(q_1, \ldots, q_n) \to q'$, $\#q_i \to q_i$ for $1 \leq i \leq n$ and $q' \to q$. As assumed above, $\mathcal{Q}_{arg}$ contains states of $\mathcal{A}^G$ and all states necessary to normalize the ground subterms of right-hand sides of the rules of $\mathcal{R}^G \cup \mathcal{R}$. Thus $q, q', q_1, \ldots, q_n \in \mathcal{Q}_{arg}$. As a result, under all $f \in \mathcal{F}$ and in particular if $f \in \mathcal{D}$, all states under f belongs to $\mathcal{Q}_{arg}$. Thus, after the first completion step, the property is true. What remains to be shown is that this property is maintained by the next completion steps. From completion step 2 to the end, rules of $\mathcal{R}^G$ do not produce any new critical pairs to solve: they are all solved. Rules of $\mathcal{R}$ are the only ones to apply. By definition of the **Constructor** class, in the right-hand sides of the rules of $\mathcal{R}$, under a defined symbol one can only find either a variable or a ground term. The subterm rooted by a defined symbol $f \in \mathcal{D}$ is thus of the form $f(t_1, \ldots, t_n)$ for $n \in \mathbb{N}$ where t_i for $1 \leq i \leq n$ is either a ground term or a variable. Since all ground subterms of right-hand sides of $\mathcal{A}$ are recognized by states in $\mathcal{Q}_{arg}$, so is t_i if it is ground. It is thus normalized by a state of $\mathcal{Q}_{arg}$. If t_i is a variable then it will be substituted by a state obtained by matching the left-hand side of the rule on the automaton of the previous completion step. However, we just have shown that such states, under defined symbols rooting left-hand sides of rules, are all in $\mathcal{Q}_{arg}$. Thus, the state used to substitute t_i in the right-hand side will be in $\mathcal{Q}_{arg}$. As a result $f(t_1, \ldots, t_n)\sigma$ will be normalized as $f(q_1, \ldots, q_n)$ with $q_1, \ldots, q_n \in \mathcal{Q}_{arg}$. To normalize $f(q_1, \ldots, q_n)$, a new state q_{new} may be necessary. However, because of the definition of the **Constructor** class, we know that there could be no other defined symbol above f. Thus, this new state will only appear under a constructor symbol g, and thus cannot appear in other critical pairs. A similar reasoning can be used for all next completion steps. As a result, for any completion step, when solving any critical pair $(l \to r, \sigma, q)$, σ always range over $\mathcal{Q}_{arg}$. Since $\mathcal{Q}_{arg}$ is finite, there are only a finite number of possible substitutions σ mapping variables of l to $\mathcal{Q}_{arg}$. Thus, there are only a finite number of substituted right-hand sides $r\sigma$ to recognize and to normalize. Since automata produced by completion are $\not\epsilon$deterministic (Lemma 41), completion can only produce a

finite set of new states to recognize the finite set of possible $r\sigma$. This entails that completion of $\mathcal{A}^G$ with $\mathcal{R} \cup \mathcal{R}^G$ terminates. □

To sum-up, completion terminates on the **Constructor** class and it exactly computes reachable terms for linear TRS (Theorem 26). Thus, completion computes reachable terms for linear TRSs of the **Constructor** class, *i.e.* TRS of the **L-Constructor** class.

5.4. Termination Proof for the **L-GFPO** *class*

Since TRSs of the classes **G**, **L-SM**, **L-G**$^{-1}$, **L-GSM** and **L-FPO** are **L-GFPO** TRSs, it is enough to prove the termination result for the **L-GFPO** class. When restricted to linear TRS, the algorithms used in [12] and in [10] are identical. We use the algorithm of [12] because it is more precisely described. Our termination proof builds upon strong similarities between the completion algorithm and the algorithm of [12]. We prove that there exists a simulation relation between automata produced by the two algorithms.

Definition 46 (epsilon free simulation $\lhd^{\not\varepsilon}$ and simulation $\lhd$). Let $\mathcal{A}$, $\mathcal{B}$ be two tree automata with respective sets of states $\mathcal{Q}_\mathcal{A}$ and $\mathcal{Q}_\mathcal{B}$. The relation $\lhd^{\not\varepsilon} \subseteq \mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B}$ is an epsilon free simulation if for all $q, q_1, \ldots, q_n \in \mathcal{Q}_\mathcal{A}$ and $p_1, \ldots, p_n \in \mathcal{Q}_\mathcal{B}$:

$$(1) \quad \left.\begin{array}{l} f(q_1, \ldots, q_n) \to q \in \mathcal{A} \\ q_i \lhd^{\not\varepsilon} p_i \text{ for } 1 \leq i \leq n \end{array}\right| \implies \exists p \in \mathcal{Q}_\mathcal{B} : f(p_1, \ldots, p_n) \to p \in \mathcal{B} \ \wedge \ q \lhd^{\not\varepsilon} p$$

$$(2) \ \exists p \in \mathcal{Q}_\mathcal{B} : f(p_1, \ldots, p_n) \to p \in \mathcal{B} \wedge q \lhd^{\not\varepsilon} p \implies \left|\begin{array}{l} f(q_1, \ldots, q_n) \to q \in \mathcal{A} \\ q_i \lhd^{\not\varepsilon} p_i \text{ for } 1 \leq i \leq n \end{array}\right.$$

The relation $\lhd \subseteq \mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B}$ is a simulation if it is an epsilon free simulation (*i.e.* $\lhd^{\not\varepsilon} \subseteq \lhd$) and for all $q \in \mathcal{Q}_\mathcal{A}$ and $p \in \mathcal{Q}_\mathcal{B}$:

$$(3) \ q \lhd p \ \wedge \ \exists q' : q \to q' \in \mathcal{A} \iff \exists p' \in \mathcal{Q}_\mathcal{B} : p \to p' \in \mathcal{B} \ \wedge \ q' \lhd p'$$

We note $\mathcal{A} \lhd^{\not\varepsilon} \mathcal{B}$ (resp. $\mathcal{A} \lhd \mathcal{B}$) if there is an epsilon free simulation (resp. a simulation) such that for all states q of $\mathcal{A}$ there exists a state p of $\mathcal{B}$ such that $q \lhd^{\not\varepsilon} p$ (resp. $q \lhd p$).

If $\mathcal{A} \lhd^{\not\varepsilon} \mathcal{B}$ (resp. $\mathcal{A} \lhd \mathcal{B}$), this definition guarantees that each state of $\mathcal{A}$ is simulated by at least one state of $\mathcal{B}$. However, the opposite is not true: there may be some states of $\mathcal{B}$ that are not related to states of $\mathcal{A}$ by the simulation. This property will be used, for instance, in the proof of Lemma 50. Besides, note that $\lhd \subseteq \lhd^{\not\varepsilon}$ since $\lhd$ is a $\lhd^{\not\varepsilon}$ with one more condition. Case (1) of the definition ensures that all transitions of $\mathcal{A}$ have a counterpart in $\mathcal{B}$. Case (3) ensures that this can be lifted to the case of epsilon transitions. Case (2) ensures that if a state p epsilon free simulates a state q ($q \lhd^{\not\varepsilon} p$) then all transitions leading to p in $\mathcal{B}$ have a counterpart in $\mathcal{A}$. Note that case (2) is not lifted to the case of

epsilon transitions (as case (1) is). For instance, a transition $p_1 \to p_2$ of $\mathcal{B}$ may not be mirrored in $\mathcal{A}$ if there is no state in $\mathcal{A}$ related with p_2 by $\lhd$. Again, this property will be necessary to achieve the proof of Lemma 50.

The simulation relation is oriented and is (generally) not a bi-simulation. In particular, tree automata produced by completion are $\not\epsilon$-deterministic (see Lemma 41). This is not the case for the algorithm of [12]. See Example 40, where using addtrans results into a set of normalized transition that is not $\not\epsilon$-deterministic: it has transitions $a \to \langle q \rangle$ and $a \to \langle a \rangle$. As a result, tree automata produced by completion can be more compact.

States which are in relation with the simulation enjoy the following properties. The first property follows from case (2) of the above definition: if $q \lhd^{\not\epsilon} p$ then all transitions of $\mathcal{B}$ leading to p have a counterpart in $\mathcal{A}$, and thus we naturally have $\mathcal{L}^{\not\epsilon}(\mathcal{A}, q) \supseteq \mathcal{L}^{\not\epsilon}(\mathcal{B}, p)$. The second property tightly relates runs in $\mathcal{A}$ and in $\mathcal{B}$. For any run in $\mathcal{A}$ where the state q' is used at any position, there is a corresponding run in $\mathcal{B}$ where a state p' is used at the same position and such that $q' \lhd p'$.

Lemma 47 (Simulations and runs)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two tree automata such that $\mathcal{A} \lhd \mathcal{B}$ and $\mathcal{A}$ is $\not\epsilon$-deterministic. For all $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $q \in \mathcal{Q}_\mathcal{A}$, $p \in \mathcal{Q}_\mathcal{B}$ such that $t \to_\mathcal{A}{}^* q$, $t \to_\mathcal{B}^* p$ and $q \lhd p$, for all positions $u \in \mathcal{P}os(t)$ and all states $q' \in \mathcal{Q}_\mathcal{A}$, if $t \to_\mathcal{A}{}^* t[q']_u \to_\mathcal{A}{}^* q$ then there exists a state $p' \in \mathcal{Q}_\mathcal{B}$ such that $t \to_\mathcal{B}^* t[p']_u \to_\mathcal{B}^* p$ and $q' \lhd p'$.*

Proof. We make a proof by induction on the height of $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. For the base case, t is a constant a and we necessarily have runs of the form $a \to_\mathcal{A} q' \to_\mathcal{A}{}^* q$ and $a \to_\mathcal{B} p' \to_\mathcal{B}^* p$. Let $p' \to_\mathcal{B} p_1 \to_\mathcal{B} \ldots \to_\mathcal{B} p_n \to_\mathcal{B} p$ be the steps used in $p' \to_\mathcal{B}^* p$. We can apply the equivalence of the case (3) of Definition 46 in the right to left direction to all steps and get there are states $q'', q_1, \ldots, q_n$ such that $q'' \to_\mathcal{A} q_1 \to_\mathcal{A} \ldots \to_\mathcal{A} q_n \to_\mathcal{A} q$ and $q'' \lhd p'$, $q_i \lhd p_i$ for $1 \leq i \leq n$. Furthermore, q'' is equal to q' because from $a \to_\mathcal{B} p'$, $q'' \lhd p'$ and case (2) of Definition 46, we get that $a \to q'' \in \mathcal{A}$. Since $a \to_\mathcal{A} q'$ and $\mathcal{A}$ is $\not\epsilon$-deterministic, $q' = q''$. For the inductive case, let us consider a term $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and assume that the property is true for all terms t_i for $1 \leq i \leq n$. By assumption, we know that $f(t_1, \ldots, t_n) \to_\mathcal{A}{}^* q$, $f(t_1, \ldots, t_n) \to_\mathcal{B}^* p$, $q \lhd p$ and that $f(t_1, \ldots, t_n)[q']_u \to_\mathcal{A}{}^* q$. By definition of runs of automata, we know that there exists states $q_1, \ldots, q_n \in \mathcal{Q}_\mathcal{A}$ and $p_1, \ldots, p_n \in \mathcal{Q}_\mathcal{B}$ such that $f(t_1, \ldots, t_n) \to_\mathcal{A}{}^* f(q_1, \ldots, q_n) \to_\mathcal{A}{}^* q$ and $f(t_1, \ldots, t_n) \to_\mathcal{B}^* f(p_1, \ldots, p_n) \to_\mathcal{B}^* p$. From $f(p_1, \ldots, p_n) \to_\mathcal{B}^* p$, $q \lhd p$ and case (2) of Definition 46, we get that $q_i \lhd p_i$ for $1 \leq i \leq n$. Note that, like in the base case, if some epsilon transitions are concerned by $f(q_1, \ldots, q_n) \to_\mathcal{A}{}^* q$ we can conclude using the case (3) of Definition 46 and the $\not\epsilon$-determinism of $\mathcal{A}$. Then, by case on the position u:

- If position u is λ, then $f(t_1, \ldots, t_n)[q']_u = q'$. Thus, we know that $f(t_1, \ldots, t_n) \to_\mathcal{A}{}^* q' \to_\mathcal{A}{}^* q$. Like in the base case, from $f(t_1, \ldots, t_n) \to_\mathcal{B}^* p$

and $q \lhd p$, we can iteratively use the case (3) of Definition 46 to find the state p' such that $f(t_1, \ldots, t_n) \to_{\mathcal{B}}^* p' \to_{\mathcal{B}}^* p$ and $q' \lhd p'$.

- Assume that position u belongs to the subterm t_k with $1 \leq k \leq n$. Thus, $u = k.v$ where $v \in \mathcal{P}os(t_k)$ and $f(\ldots, t_k, \ldots)[q']_u = f(\ldots, t_k[q']_v, \ldots)$. From above, we know that $f(\ldots, t_k[q']_v, \ldots) \to_{\mathcal{A}}{}^* f(\ldots, q_k, \ldots)$, that $f(\ldots, t_k, \ldots) \to_{\mathcal{B}}^* f(\ldots, p_k, \ldots)$ and $q_i \lhd p_i$ for $1 \leq i \leq n$. Thus, we can apply the induction hypothesis on $t_k[q']_v \to_{\mathcal{A}}{}^* q_k$, $t_k \to_{\mathcal{B}}^* p_k$ and $q_k \lhd p_k$. We get that there exists a state $p' \in \mathcal{Q}_{\mathcal{B}}$ such that $f(t_1, \ldots, t_{k-1}, t_k[p']_v, t_{k+1}, \ldots, t_n) \to_{\mathcal{B}}^* f(p_1, \ldots, p_{k-1}, p_k, p_{k+1}, \ldots, p_n) \to_{\mathcal{B}} p$ and $q' \lhd p'$, which concludes the case.

□

The next lemma is a direct consequence of the previous one: we can lift the previous property to the case of contexts.

Lemma 48 (Simulations and contexts)**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two tree automata such that $\mathcal{A} \lhd \mathcal{B}$ and $\mathcal{A}$ is $\not\epsilon$-deterministic. For all ground contexts $C[\,]$, all states $q, q_1, \ldots, q_n \in \mathcal{Q}_{\mathcal{A}}$, if $C[q_1, \ldots, q_n] \to_{\mathcal{A}}{}^* q$ then there exists states $p, p_1, \ldots, p_n \in \mathcal{Q}_{\mathcal{B}}$ such that $q \lhd p$, $q_i \lhd p_i$ for $1 \leq i \leq n$ and $C[p_1, \ldots, p_n] \to_{\mathcal{B}}^* p$.*

Proof. Let $t, t_1, \ldots, t_n$ be terms of $\mathcal{T}(\mathcal{F})$ such that $t = C[t_1, \ldots, t_n]$ and $t \to_{\mathcal{A}}{}^* C[q_1, \ldots, q_n] \to_{\mathcal{A}}{}^* q$. From the case (2) of Definition 46, we know that there exists a state $p \in \mathcal{Q}_{\mathcal{B}}$ such that $q \lhd p$ and $t \to_{\mathcal{B}}^* p$. Using iteratively Lemma 47 on all the positions of terms t_i for $1 \leq i \leq n$, we get that there exists states $p_1, \ldots, p_n$ such that $C[t_1, \ldots, t_n] \to_{\mathcal{B}}^* C[p_1, \ldots, p_n] \to_{\mathcal{B}}^* p$ and $q_i \lhd p_i$ for $1 \leq i \leq n$. □

Lemma 49 (addtrans and $Norm$ produce comparable transition sets)**.** *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}_{\mathcal{A}}, \mathcal{Q}_{\mathcal{A}}^f, \Delta_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle \mathcal{F}, \mathcal{Q}_{\mathcal{B}}, \mathcal{Q}_{\mathcal{B}}^f, \Delta_{\mathcal{B}} \rangle$ be two tree automata such that $\mathcal{A} \lhd \mathcal{B}$. Let q be a new state for $\Delta_{\mathcal{A}}$, $C[\,]$ a ground context, $q_1, \ldots, q_n \in \mathcal{Q}_{\mathcal{A}}$ and $p_1, \ldots, p_n \in \mathcal{Q}_{\mathcal{B}}$ such that $q_i \lhd^{\not\epsilon} p_i$ for $1 \leq i \leq n$. Let $\mathcal{A}'$ be the automaton with transition set $\Delta \cup Norm_{\Delta_{\mathcal{A}}}(C[q_1, \ldots, q_n] \to q)$ and $\mathcal{B}'$ the automaton obtained by running* addtrans$(C[p_1, \ldots, p_n])$ *on $\mathcal{B}$. If $\Delta_{\mathcal{A}}$ is injective then $\mathcal{A}' \lhd^{\not\epsilon} \mathcal{B}'$ and, in particular, $q \lhd^{\not\epsilon} \langle C[p_1, \ldots, p_n] \rangle$.*

Proof. We prove this property by induction on the number of symbols of $\mathcal{F}$ in $C[\,]$. If there is one symbol, then $C[q_1, \ldots, q_n] = f(q_1, \ldots, q_n)$ and by Definition 2, we have $Norm_{\Delta_{\mathcal{A}}}(f(q_1, \ldots, q_n) \to q) = \{f(q_1, \ldots, q_n) \to q\}$. In this case, the effect of addtrans is simply to add the transition $f(p_1, \ldots, p_n) \to \langle f(p_1, \ldots, p_n) \rangle$. By hypothesis, $q_i \lhd^{\not\epsilon} p_i$ for $1 \leq i \leq n$ thus, by Definition 46, $q \lhd^{\not\epsilon} \langle f(p_1, \ldots, p_n) \rangle$.

Now, assume that this property is true for all ground contexts $C[\,]$ having no more than $n \geq 1$ symbols. Assume that $C[\,]$ is a ground context having $n + 1$ symbols. Let f be one of the symbols occurring at the leaves of $C[\,]$. Thus, there exists a ground context $C'[\,]$ and $1 \leq i \leq j \leq n$ such that

$C[q_1,\ldots,q_n] = C'[q_1,\ldots,q_{i-1},f(q_i,\ldots,q_j),q_{j+1},\ldots,q_n]$ and $C[p_1,\ldots,p_n] = C'[p_1,\ldots,p_{i-1},f(p_i,\ldots,p_j),p_{j+1},\ldots,p_n]$. By case 2 of Definition 2, the set $Norm_{\Delta_\mathcal{A}}(C'[q_1,\ldots,q_{i-1},f(q_i,\ldots,q_j),q_{j+1},\ldots,q_n])$ is equal to $\{f(q_i,\ldots,q_j)\to q'\}\cup Norm_{\Delta_\mathcal{A}\cup\{f(q_1,\ldots,q_n)\to q'\}}(C'[q_1,\ldots,q_{i-1},q',q_{j+1},\ldots,q_n])$, where either the transition $f(q_1,\ldots,q_n)\to q'$ belongs to $\Delta_\mathcal{A}$ or q' is a new state. In both cases, note that $\Delta_\mathcal{A}\cup\{f(q_1,\ldots,q_n)\to q'\}$ remains injective. Recall that by hypothesis $q_i \lhd^{\not\epsilon} p_i$ for $1\leq i\leq n$. If we manage to prove that $q' \lhd^{\not\epsilon} \langle f(p_i,\ldots,p_j)\rangle$ then we will be able to use the induction hypothesis on the set $Norm_{\Delta_\mathcal{A}\cup\{f(q_1,\ldots,q_n)\to q'\}}(C'[q_1,\ldots,q_{i-1},q',q_{j+1},\ldots,q_n])$ and addtrans on $C'[p_1,\ldots,p_{i-1},\langle f(p_i,\ldots,p_j)\rangle,p_{j+1},\ldots,p_n]$ to get that $q\lhd^{\not\epsilon}\langle C[p_1,\ldots,p_n]\rangle$. Since having $q\lhd^{\not\epsilon}\langle C[p_1,\ldots,p_n]\rangle$ concludes the proof, what remains to be proved is that $q'\lhd^{\not\epsilon}\langle f(p_i,\ldots,p_j)$. As above, this is a direct consequence of the fact that, by hypothesis, $q_i\lhd^{\not\epsilon} p_i$ for $1\leq i\leq n$ and thus, by Definition 46, $q'\lhd^{\not\epsilon}\langle f(p_1,\ldots,p_n)\rangle$. □

Now we take advantage of the fact that the simulation is not a bi-simulation. The principle is to compare the automaton under completion (which may not be complete) $\mathcal{A}^i_{\mathcal{R},\emptyset}$ with the automaton $\mathcal{B}$ produced by the algorithm of [12]. Since $\mathcal{B}$ contains more states and transitions than incomplete $\mathcal{A}^i_{\mathcal{R},\emptyset}$, there exists states in $\mathcal{B}$ that are not already mirrored in $\mathcal{A}^i_{\mathcal{R},\emptyset}$.

Lemma 50 (An automaton produced by the algorithm of [12] simulates an automaton produced by completion)**.** *Let $\mathcal{R}$ be a linear TRS, $\mathcal{A}$ be an $\not\epsilon$-deterministic automaton with an injective set of transitions and $i\in\mathbb{N}$. If running the completion procedure of [12] on pack($\mathcal{A}$) and $\mathcal{R}$ stops on the automaton $\mathcal{B}$, then $\mathcal{A}^i_{\mathcal{R},\emptyset}\lhd\mathcal{B}$.*

Proof. We do a proof by induction on i. The result is true for $\mathcal{A}^0_{\mathcal{R},\emptyset}=\mathcal{A}$ since $\mathcal{B}$ contains $pack(\mathcal{A})$. Thus, for all states q of $\mathcal{A}$ we can associate the state $\langle q\rangle$ of $\mathcal{B}$. Since all the transitions of $\mathcal{A}$ are translated into transitions of $pack(\mathcal{A})$, conditions (1), (2) of $\lhd^{\not\epsilon}$ and condition (3) for $\lhd$ are trivially satisfied. This is also due to the fact that, by definition, the addtrans procedure does add any transition $c\to\langle q\rangle$ with $q\in\mathcal{Q}_\mathcal{A}$. Now, assume that the property is true for $\mathcal{A}^i_{\mathcal{R},\emptyset}=\langle\mathcal{F},\mathcal{Q}^i,\mathcal{Q}_f,\Delta_i\rangle$. We prove that it is true for $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$. Recall that $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}=\mathcal{C}_\mathcal{R}(\mathcal{A}^i_{\mathcal{R},\emptyset})$. Let q_s be a state of $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$. We prove that there exists a state p_s in $\mathcal{B}$ such that $q_s\lhd p_s$ by case on q_s: either **(a)** q_s is a new state ($q_s\notin\mathcal{Q}^i$) or **(b)** q_s already belongs to $\mathcal{A}^i_{\mathcal{R},\emptyset}$ ($q_s\in\mathcal{Q}^i$).

(a) For q_s to be a new state, it needs to be built while solving a critical pair, *i.e.* there exists a rewrite rule $l\to r\in\mathcal{R}$, a state $q\in\mathcal{A}^i_{\mathcal{R},\emptyset}$ and a substitution $\sigma:\mathcal{X}\mapsto\mathcal{Q}^i$ such that $l\sigma\to^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$ and $r\sigma\not\to^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$. Then completion adds $Norm_{\Delta^i}(r\sigma\to q')$ where q' is a new state. Thus, we need to prove the property for q' and for all the new states q'' introduced by $Norm_{\Delta^i}(r\sigma\to q')$. First, let us prove the property for q'. Let $C[]$ be a ground context and $x_1,\ldots,x_n\in\mathcal{X}$ such that $l=C[x_1,\ldots,x_n]$. Let $q_1,\ldots,q_n\in\mathcal{Q}^i$ be the states such that $\sigma=\{x_1\mapsto q_1,\ldots,x_n\mapsto q_n\}$. Thus

$l\sigma = C[q_1, \ldots, q_n]$. We can apply the induction hypothesis on $\mathcal{A}^i_{\mathcal{R},\emptyset}$ and get that $\mathcal{A}^i_{\mathcal{R},\emptyset} \lhd \mathcal{B}$. Since $C[q_1, \ldots, q_n] \rightarrow^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$, we can apply Lemma 48 ($\mathcal{A}^i_{\mathcal{R},\emptyset}$ is $\not\epsilon$deterministic by Lemma 41) and get that there exists states $p, p_1, \ldots, p_n \in \mathcal{Q}_\mathcal{B}$ such that $q_i \lhd p_i$ for $1 \leq i \leq n$ and $C[p_1, \ldots, p_n] \rightarrow^*_\mathcal{B} p$. Let $\sigma' = \{x_1 \mapsto p_1, \ldots, x_n \mapsto p_n\}$. We thus have $C[p_1, \ldots, p_n] = l\sigma' \rightarrow^*_\mathcal{B} p$ which entails that a critical pair has also been solved on $\mathcal{B}$ and the addtrans procedure has been run on $r\sigma'$. Then, since $\lhd \subseteq \lhd^{\not\epsilon}$, $q_i \lhd p_i$ implies $q_i \lhd^{\not\epsilon} p_i$ for $1 \leq i \leq n$. Then, we can use Lemma 49 on $Norm_{\Delta_i}(r\sigma \rightarrow q')$ and addtrans$(r\sigma')$ to get that $q' \lhd^{\not\epsilon} \langle r\sigma' \rangle$. Furthermore since q' is a new state added by completion of $\mathcal{A}^i_{\mathcal{R},\emptyset}$, there cannot be any epsilon transition leading to q' in $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$, thus condition (3) of Definition 46 is trivially satisfied on $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$ for q'. Thus, $q' \lhd^{\not\epsilon} \langle r\sigma \rangle$ implies $q' \lhd \langle r\sigma \rangle$. What remains to be proved is that for any new state q'' obtained by normalizing $r\sigma$, there exists a state p'' such that $q'' \lhd p''$. Since $r\sigma \rightarrow^*_{\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}} q'$, $r\sigma \rightarrow^*_\mathcal{B} \langle r\sigma \rangle$ and $q' \lhd \langle r\sigma \rangle$, we can use Lemma 47 and get that for all states q'' used to normalize any subterm of $r\sigma$ there exists a state p'' such that $q'' \lhd p''$.

(b) If $q_s \in \mathcal{Q}^i$ then we now prove that completion preserves simulation on this state. we first prove that the language recognized by q_s will not be changed by normalizations. Since the set of transitions of $\mathcal{A}$ is injective so is the set Δ_i of $\mathcal{A}^i_{\mathcal{R},\emptyset}$ by Lemma 43. Using Lemma 73 of Appendix A, we can ensure that for all terms t and all state q'' adding to Δ_i the transitions from $Norm_{\Delta_i}(t \rightarrow q)$ will not change the language recognized by q_s and thus will not affect its simulations. Now, we prove that if a transition $q' \rightarrow q_s$ is added then simulation is preserved. If such a transition is added this means that a critical pair has been solved as in case **(a)**. Thus, as above, there exists a rewrite rule $l \rightarrow r \in \mathcal{R}$ and a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}^i$ such that $l\sigma \rightarrow^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q_s$ and $r\sigma \not\rightarrow^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q_s$. Then the critical pair is solved by adding transitions $Norm_{\Delta_i}(r\sigma \rightarrow q')$ and $q' \rightarrow q_s$. Using the same reasoning as in case **(b)**, we can deduce that in $\mathcal{B}$ there exists a substitution σ', a state p and a state $\langle r\sigma' \rangle$ such that $l\sigma' \rightarrow^*_\mathcal{B} p$ and $q' \lhd \langle r\sigma' \rangle$. We also know that for the critical pair to be solved by the algorithm of [12], it has been necessary to add the transition $\langle r\sigma' \rangle \rightarrow p$. Since $q' \lhd \langle r\sigma' \rangle$, $q' \rightarrow q_s \in \mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$ and $\langle r\sigma' \rangle \rightarrow p$ the fact that $q_s \lhd p$ is preserved.

□

Now, we can take use this last lemma to state the termination theorem.

Theorem 51. *Let $\mathcal{R}$ be a* **L-GFPO** *TRS and $\mathcal{A}$ be a tree automaton. If R^G and $\mathcal{A}^G$ are the shifting TRS and automaton of $\mathcal{A}$ then completion of $\mathcal{A}^G$ with $\mathcal{R} \cup \mathcal{R}^G$ and $E = \emptyset$ terminates.*

Proof. Note that if $\mathcal{R}$ is **L-GFPO**, so is $\mathcal{R} \cup \mathcal{R}^G$. Thus the algorithm of [12] should stop on a tree automaton recognizing $(\mathcal{R} \cup \mathcal{R}^G)(\mathcal{L}(\mathcal{A}^G))$. Let us call

$\mathcal{B}$ this automaton. Assume that completion of $\mathcal{A}^G$ by $\mathcal{R} \cup \mathcal{R}^G$ diverges. This means that it produces infinitely many states $q_1, q_2, \ldots$ Let A^k be the automaton produced by completion of $\mathcal{A}^G$ by $\mathcal{R} \cup \mathcal{R}^G$ so that it has at least one more state than $\mathcal{B}$. Since $\mathcal{A}^G$ has an injective set of transitions, we can apply Lemma 50 and get that $\mathcal{A}^k \lhd \mathcal{B}$. Since $\mathcal{A}^k$ has more states than $\mathcal{B}$ we know that there exists at least two states q_1, q_2 of $\mathcal{A}^k$ such that $q_1 \neq q_2$ and a state p of $\mathcal{B}$ such that $q_1 \lhd p$ and $q_2 \lhd p$. Since $\lhd \subseteq \lhd^{\not\epsilon}$, we thus have that $q_1 \lhd^{\not\epsilon} p$ and $q_2 \lhd^{\not\epsilon} p$. Let t be a term such that $t \in \mathcal{L}^{\not\epsilon}(\mathcal{B}, p)$. Since $q_1 \lhd^{\not\epsilon} p$ and $q_2 \lhd^{\not\epsilon} p$, we know that $t \in \mathcal{L}^{\not\epsilon}(\mathcal{A}^k, q_1)$ and $t \in \mathcal{L}^{\not\epsilon}(\mathcal{A}^k, q_2)$. This is a contradiction with Lemma 41 which guarantees that $\mathcal{A}^k$ is $\not\epsilon$deterministic. □

Finally, we have the proof that on the **L-GFPO** class tree automata completion terminates. Again, combining it with Theorem 26 yields that on the **L-GFPO** class completion produces a tree automaton exactly recognizing the set of reachable terms. Besides, to get the termination result we had to prove the Lemma 50 which says that tree automata completion and the algorithm of [12] computes comparable tree automata. As explained above, the only difference between the two is that tree automata produced by completion are always $\not\epsilon$ deterministic, though automata produced by [12] may not be. However, since the packed state mechanism can only produce a finite number of states recognizing a given term, we conjecture that the algorithm of [12] also stops when completion does. This algorithm is thus likely to also cover the **L-Constructor** class. This is left for future work.

5.5. An application example

Using the fact that completion covers the **L-Constructor** class, we can use completion to compute exactly the set of reachable terms for TRS and initial tree automata in this class. To perform those computations, we use the Timbuk [13] tool with the `--exact` option which implements the exact completion and totally hides the shifting/projection operations on tree automata. Here is a typical Timbuk's specification:

```
Ops app:2 nil:0 cons:2 A:0 B:0
Vars X Y Z
TRS R1
append(nil,X)->X
append(cons(X,Y),Z)->cons(X,append(Y,Z))
Automaton A0
States q0 qla qlb qa qb qf
Final States qf
Transitions
append(qla,qlb)->qf   cons(qa,qla)->qla  nil->qla  b->qb
cons(qb, qlb)->qlb    nil->qlb           a->qa
```

The TRS `R1` defines the `append` function concatenating lists and the initial automaton `A0` recognizes terms of the form $append(l_a, l_b)$ where l_a (resp. l_b) is a list containing 0 or more occurrences of a (resp. b). On this specification (in the

L-Constructor class), Timbuk can compute the automaton recognizing the set of reachable terms. Completion terminates in 2 steps and gives the automaton:

```
States q0 q1 q2 q3 q4 q5 q6 q7 q9 q10 q11 q12 q13 Final States q7
Transitions
cons(q12,q10)->q10  nil->q10  cons(q11,q9)->q1   app(q9,q10)->q0
cons(q11,q0)->q0    nil->q0   cons(q12,q10)->q0  a->q11  b->q12
cons(q11,q9)->q9    nil->q9   cons(q11,q0)->q13  app(q9,q10)->q7
cons(q12,q10)->q7   nil->q7   cons(q12,q10)->q3  a->q5   b->q6
cons(q11,q0)->q7    nil->q4   nil->q2
```

Note that, during completion, Timbuk computes ϵ-deterministic tree automata $\mathcal{A}^i_{\mathcal{R}}$ with epsilon transitions. However, for a better readability, we here use Timbuk's outputs where epsilon transitions have been normalized [15]: any epsilon transition $q \to q'$ is replaced by the set of transitions $\{f(q_1, \ldots, q_n) \to q' \mid f(q_1, \ldots, q_n) \to q \in \mathcal{A}\}$. The result is thus an automaton with no epsilon transitions and that may not be ϵ-deterministic. By carefully looking at the outputted automaton above, we can see that the only terms, representing lists (built on *cons*, *nil*, a and b), recognized by this automaton are (possibly empty) lists of a's followed by some b's.

5.6. Going further

Completion used in the setting of Theorem 26 can also compute exactly sets of reachable terms for cases that are outside of known decidable classes. A very simple example is the TRS $\mathcal{R} = \{f(g(x)) \to g(f(x))\}$ and the initial language $\mathcal{S} = \{h^*(f(g^*(a)))\}$. First, this TRS does not preserve regularity for any set $\mathcal{S}$. For instance, if we choose the initial set $\mathcal{S}' = \{(fg)^*(a)\}$ the set $\mathcal{R}^*(\mathcal{S}')$ is not regular. This is due to the fact that $\mathcal{R}^!(\mathcal{S}') = \mathcal{R}^*(\mathcal{S}') \cap \textsc{Irr}(\mathcal{R})$ and $\textsc{Irr}(\mathcal{R})$ is regular. If $\mathcal{R}^*(\mathcal{S}')$ was regular then so would be $\mathcal{R}^!(\mathcal{S}')$. However, $\mathcal{R}^!(\mathcal{S}') = \{g^n(f^n(a)) \mid n \in \mathbb{N}\}$ is not regular and, thus, $\mathcal{R}^*(\mathcal{S}')$ is not regular. Hence, this TRS is outside of all the classes that do not impose restrictions on the initial set of terms, i.e. all of them except the class **L-Constructor**. This particular example is also outside of the **L-Constructor** class because the initial set is not of the form $\{t\sigma\}$ where t is a linear term. However, we can easily build a tree automaton $\mathcal{A}$ recognizing S and the TRS is linear. Moreover, completion of $\mathcal{A}^G$ with $\mathcal{R} \cup \mathcal{R}^G$ and $E = \emptyset$ terminates on a tree automaton $(\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset}$. Thanks to Theorem 26, we know that $\Pi_{\mathcal{F}}((\mathcal{A}^G)^*_{\mathcal{R}\cup\mathcal{R}^G,\emptyset})$ recognizes $\mathcal{R}^*(S)$. We thus have a proof of the regularity of $\mathcal{R}^*(\mathcal{S})$ for this specific $\mathcal{R}$ and $\mathcal{S}$.

Another simple example is the TRS $\mathcal{R} = \{f(x) \to f(g(f(x)))\}$ and the initial language $\mathcal{S} = \{f(a)\}$. This TRS is outside of all the regular classes we survey in Chapter 5.1. In particular, it is outside the **L-GFPO** class because its GFPO-graph has only one node and a looping edge of weight 1 on it. This is due to the fact that $f(g(f(x)))$ properly sticks out of $f(x)$. It is also outside of the **L-Constructor** class because the right hand side of the rewrite rule has nested function symbol: f. However, again, completion terminates on a tree automaton recognizing $\mathcal{R}^*(\mathcal{S}) = \{(fg)^*f(a)\}$.

Besides, coming back to our previous example (the `append` function) if we add to the TRS the rules defining the `reverse` function:

```
rev(nil)->nil
rev(cons(X,Y))->append(rev(Y),cons(X,nil))
```

then it is no longer in the **L-Constructor** class (nor in the **Constructor** class). This is due to the fact that in the right-hand side of the second rule we have two nested functional symbols: `append` and `rev`. If we try to apply completion on this TRS and an initial tree automaton recognizing terms of the form $\texttt{rev}(\texttt{append}(l_a, l_b))$, completion is not terminating. This time we can use approximation equations to enforce termination. In the next section, we define sufficient conditions on approximation equations for completion to always terminate on any left-linear TRS and any initial tree automaton.

6. Termination criteria depending on equations

When the TRS and initial automaton to consider are outside of the above classes, completion may not terminate. The remaining question is thus how to ensure termination using the set of approximation equations E. 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 of an equation, for all the equations of E. In particular, equations are not used as rewrite rules. Thus, we do not assume confluence or termination of the (oriented) equations in our setting.

6.1. General termination criterion

In this section, we give a sufficient condition on E and on the completed tree automaton $\mathcal{A}^i_{\mathcal{R},E}$ for the tree automata completion to always terminate. The intuition behind this condition is simple: if the set of equivalence classes for E is finite then so should be the set of new states used in completion. This can be seen as a straightforward application of the Myhill-Nerode Theorem for trees [15]: if the set of classes of $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite, then there exists a tree automaton recognizing them. As a result, completion of any $\mathcal{A}$ with any $\mathcal{R}$ provided that E has a finite set of E-equivalence classes should terminate. We first need to define the notion of E-compatibility for a tree automaton which ensures that a E-equivalence class is recognized (without epsilon transitions) by at most one state in the tree automaton.

Definition 52 (E-compatible tree automaton $\mathcal{A}$). Let $\mathcal{A}$ be an automaton and E a set of equations. The automaton $\mathcal{A}$ is E-compatible if for all $s, t \in \mathcal{T}(\mathcal{F})$, $q_1, q_2 \in \mathcal{A}$ such that $s =_E t$, $s \to^{\not\varepsilon *}_{\mathcal{A}} q_1$ and $t \to^{\not\varepsilon *}_{\mathcal{A}} q_2$ then $q_1 = q_2$.

Note that the notion of E-compatibility is complementary to $\mathcal{R}/E$ coherence. $\mathcal{R}/E$-coherence ensures that if two terms are recognized (with $\to^{\not\varepsilon *}_{\mathcal{A}}$) into the same state then they are equivalent modulo E. On the opposite, E-compatibility

ensures that if two terms are equivalent modulo E and if both are recognized (with $\rightarrow_{\mathcal{A}}^{\not\epsilon *}$) then they are recognized by the same state. As a result, there are $\mathcal{R}/E$-coherent tree automata that are not E-compatible and conversely. For instance, let $E = \{a = b\}$. If $\mathcal{A}$ has transitions $a \rightarrow q_1, b \rightarrow q_2$ then it is $\mathcal{R}/E$-coherent but not E-compatible. On the opposite if $\mathcal{A}$ has transitions $a \rightarrow q_1, c \rightarrow q_1$ and $E = \emptyset$ then it is $\mathcal{A}$ is E-compatible but not $\mathcal{R}/E$-coherent. The first interesting property of E-compatibility is that, whatever E may be, it entails $\not\epsilon$-determinism.

Lemma 53 (E-compatible tree automata are $\not\epsilon$-deterministic)**.** *Let E be a set of equations. If $\mathcal{A}$ is an E-compatible tree automaton then $\mathcal{A}$ is $\not\epsilon$-deterministic.*

Proof. Note that $=_E$ is reflexive. Thus, for all terms $t \in \mathcal{T}(\mathcal{F})$ and for any two states $q, q' \in \mathcal{A}$ such that $t \rightarrow_{\mathcal{A}}^{\not\epsilon *} q$ and $t \rightarrow_{\mathcal{A}}^{\not\epsilon *} q'$ since $t =_E t$ for any E and $\mathcal{A}$ is a E-compatible, we have $q = q'$. As a result, $\mathcal{A}$ is $\not\epsilon$-deterministic. □

On the opposite, it is easy to see that every tree automaton $\mathcal{A}$ that is not $\not\epsilon$-deterministic is not E-compatible whatever E may be. Using the above lemma, we can state a general termination theorem for completion of reduced automata with sets equations having a finite set of equivalence classes.

Theorem 54 (General termination criterion for completion of $\mathcal{A}$ by $\mathcal{R}$ and E)**.** *Let $\mathcal{A}$ be a $\not\epsilon$-reduced tree automaton, $\mathcal{R}$ a left-linear TRS, $j \in \mathbb{N}$, and E a set of equations such that $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite. If for all $i \geq j$, $\mathcal{A}^i_{\mathcal{R},E}$ is E-compatible then there exists an integer n such that $\mathcal{A}^n_{\mathcal{R},E}$ is a fixpoint.*

Proof. We make a proof by contradiction. Let k be the number of equivalence classes in $\mathcal{T}(\mathcal{F})/_{=_E}$. Assume that completion diverges. Thus, it infinitely produces automata $\mathcal{A}^1_{\mathcal{R},E}$, $\mathcal{A}^2_{\mathcal{R},E}$, ... This is possible only if completion produces infinitely many new states $q_1, q_2, \ldots$ Let $p \geq j$ and $\mathcal{A}^p_{\mathcal{R},E}$ the first E-compatible automaton having at least $k+1$ distinct new states. By Lemma 53, we know that $\mathcal{A}^p_{\mathcal{R},E}$ is $\not\epsilon$-deterministic. As a result, each new state $q_1, \ldots, q_{k+1}$ recognizes disjoint languages, *i.e.* for all $i, j \in \{1 \ldots k+1\}$ and $i \neq j$: $\mathcal{L}^{\not\epsilon}(\mathcal{A}^p_{\mathcal{R},E}, q_i) \cap \mathcal{L}^{\not\epsilon}(\mathcal{A}^p_{\mathcal{R},E}, q_j) = \emptyset$. Since $\mathcal{A}$ is reduced and completion preserves $\not\epsilon$-reducibility (Lemma 44), we know that $\mathcal{A}^p_{\mathcal{R},E}$ is reduced. Thus, for all $1 \leq i \leq k+1$ there exists a term $s_i \in \mathcal{T}(\mathcal{F})$ such that $s_i \in \mathcal{L}^{\not\epsilon}(\mathcal{A}^p_{\mathcal{R},E}, q_i)$. Furthermore, thanks to E-compatibility of $\mathcal{A}^p_{\mathcal{R},E}$, we know that for any $1 \leq j \leq k+1$ such that $i \neq j$, we have $s_i \neq_E s_j$. As a result, we have $k+1$ terms distinct w.r.t. $=_E$ which contradicts the fact that $\mathcal{T}(\mathcal{F})/_{=_E}$ has k equivalence classes. □

Note that A has to be $\not\epsilon$-reduced for the lemma to hold because otherwise completion can generate infinitely many new states as shown on the following example.

Example 55. *Let $\mathcal{F} = \{f, a\}$ and $E = \{f(a) = a\}$. The set $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite and only contains one equivalence class: the class which contains all terms of $\mathcal{T}(\mathcal{F})$. Let $\mathcal{A}$ be the automaton having a set of transitions $\Delta = \{f(q_1) \rightarrow q_0\}$ and let $\mathcal{R}$ be the TRS $f(x) \rightarrow f(f(x))$. The automaton $\mathcal{A}$ is not $\not\epsilon$-reduced but it is E-compatible. Completion of $\mathcal{A}$ with $\mathcal{R}$ and E diverges and generates infinitely*

many new states. To illustrate this, it is enough to see that to solve the (unique) critical pair and get $\mathcal{A}^1_{\mathcal{R},E}$ *it is enough to add transitions* $f(q_0) \to q_2, q_2 \to q_0$ *so that* $f(f(q_1)) \to^*_{\mathcal{A}^1_{\mathcal{R},E}} q_0$*. Note that* $\mathcal{A}^1_{\mathcal{R},E}$ *is still* E*-compatible since neither* a *nor* $f(a)$ *are recognized by* $\mathcal{A}^1_{\mathcal{R},E}$*. However,* $\mathcal{A}^1_{\mathcal{R},E}$ *has* 3 *states though* $\mathcal{T}(\mathcal{F})/_{=_E}$ *has only one equivalence class. Note that next completion steps will also produce new states and completion does not stop.*

6.2. Syntactic restrictions on E *for completion to terminate*

From an algorithmic point of view, the termination Theorem 54 cannot be used straightforwardly. This is due to two problems. First, determining whether $\mathcal{T}(\mathcal{F})/= E$ is finite is not decidable [32]. Second, unlike $\not$-determinism and $\not$-reducibility, E-compatibility is not preserved by completion. More precisely, $\mathcal{C}_\mathcal{R}$ can produce a non E-compatible tree automaton from an E-compatible one. Furthermore, if an automaton $\mathcal{A}$ is simplified with a set of equations E into an automaton $\mathcal{A}'$, *i.e.* $\mathcal{A} \rightsquigarrow^*_E \mathcal{A}'$ then $\mathcal{A}'$ is not necessarily E-compatible.

Example 56 (Completion does not preserve E-compatibility in general)**.** *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* $\{a \to q, c \to q', q' \to q\}$ *and* $\mathcal{C}_\mathcal{R}(\mathcal{A})$ *is no longer* E*-compatible since* $a =_E c$ *but* $a \in \mathcal{L}^{\not}(\mathcal{C}_\mathcal{R}(\mathcal{A}), q)$ *and* $c \in \mathcal{L}^{\not}(\mathcal{C}_\mathcal{R}(\mathcal{A}), q')$*. On the automaton* $\mathcal{C}_\mathcal{R}(\mathcal{A})$*, no simplification situation (as described by Definition 9), can be found because the term* b *is not recognized by* $\mathcal{C}_\mathcal{R}(\mathcal{A})$*. Hence, the simplified automaton is* $\mathcal{C}_\mathcal{R}(\mathcal{A})$*, which is not* E*-compatible.*

The above problem is clearly avoided if the automaton on which completion is applied is complete, *i.e.* if for all terms $t \in \mathcal{T}(\mathcal{F})$ there exists at least a state q such that $t \in \mathcal{L}^{\not}(\mathcal{A}, q)$. In the above example, we would have a transition $b \to q''$ and thus simplification could have been performed until having a and c recognized by the same state. However, using *complete* initial automata to compute over-approximation of reachable terms may produce very rough approximations. This is the case when the structure of the complete initial tree automaton interfere with E and completion thus add transitions recognizing unreachable terms.

Example 57. *Let* $\mathcal{F} = \{a, b, c\}$*,* $\mathcal{R} = \{a \to b\}$*,* $E = \{b = c\}$ *and* $\mathcal{A}$ *the complete tree automaton with* $\mathcal{Q}_f = \{q_0\}$ *and* $\Delta = \{a \to q_0, b \to q_1, c \to q_1\}$*. Note that* $\mathcal{L}(\mathcal{A}) = \{a\}$ *and* $\mathcal{A}$ *is* E*-compatible,* $\mathcal{R}/E$*-coherent and* $\not$*-deterministic. The first completion step yields the transition* $q_1 \to q_0$*. The transition set of the final automaton* $\mathcal{B}$ *is thus* $\{a \to q_0, q_1 \to q_0, b \to q_1, c \to q_1\}$ $\mathcal{L}(\mathcal{B}) = \{a, b, c\}$ *which is a coarse approximation of* $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$*. This result has to be compared with the result obtained when completing an equivalent initial tree automaton* $\mathcal{A}'$ *which is not complete. Let* $\Delta' = \{a \to q_0\}$ *be the set of transitions of* $\mathcal{A}'$*. Completion of* $\mathcal{A}'$ *stops on* $\mathcal{B}'$ *with transitions* $\{a \to q_0, b \to q'_1, q'_1 \to q_0\}$ *where* q'_1 *is a new state and* $\mathcal{L}(\mathcal{B}) = \{a, b\}$ *which is precisely* $\mathcal{R}^*(\mathcal{L}(\mathcal{A}'))$ *and equal to* $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$*.*

Now, we provide a simple syntactic criterion on the set E, overcoming those two problems: decidability of $\mathcal{T}(\mathcal{F})/_{=E}$ and E-compatibility of $\mathcal{A}^i_{\mathcal{R},E}$. We give a simple syntactic restriction on E to ensure that completion produce only tree automata having a finite set of states. For general TRSs, the restrictions on E are quite strong. However, we will see in Section 6.3 that such restrictions are easily met when the TRS under consideration is a "functional" TRS, *i.e.* a typed first-order functional program translated into a TRS. As a result, for "functional" TRSs, we can get a less restrictive termination criterion. Using completion with this criterion provides an efficient way to perform static analysis technique for first-order functional programs.

What Example 56 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 define a set E^c of *contracting* equations so that this property is true. What Example 56 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 58 (Set of reflexivity equations E^r). For a given set of symbols $\mathcal{F}$, $E^r = \{f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \mid f \in \mathcal{F}, \text{ and arity of } f \text{ is } n\}$, 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 any automaton into an $\not\epsilon$-deterministic automaton, as stated in the following lemma.

Lemma 59. *For all tree automata $\mathcal{A}$ and all sets of equations E, if $E \supseteq E^r$ and $\mathcal{A} \leadsto^!_E \mathcal{A}'$ then $\mathcal{A}'$ is $\not\epsilon$-deterministic.*

Proof. We prove this by induction on the height of the terms recognized by $\mathcal{A}'$. This is true for constants because otherwise there would be a constant a such that $a \to^{\not\epsilon *}_{\mathcal{A}'} q$ and $a \to^{\not\epsilon *}_{\mathcal{A}'} q'$ with $q \neq q'$. However since $a = a \in E^r$ we can simplify $\mathcal{A}'$ which contradicts the fact that $\mathcal{A}'$ is in normal form w.r.t. $\leadsto_E$. For the inductive case, assume that there exists a term $t = f(t_1, \ldots, t_n)$ such that $t \to^{\not\epsilon *}_{\mathcal{A}'} q$ and $t \to^{\not\epsilon *}_{\mathcal{A}'} q'$ with $q \neq q'$. Using the induction hypothesis, we know that for each t_i for $i = 1 \ldots n$ there exists a unique state q_i such that $t_i \to^{\not\epsilon *}_{\mathcal{A}'} q_i$. Hence, $f(t_1, \ldots, t_n) \to^{\not\epsilon *}_{\mathcal{A}'} f(q_1, \ldots, q_n)$ but $f(q_1, \ldots, q_n) \to^{\not\epsilon *}_{\mathcal{A}'} q$ and $f(q_1, \ldots, q_n) \to^{\not\epsilon *}_{\mathcal{A}'} q'$. However, this is a simplification situation for the equation $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \in E^r$ which contradicts the fact that $\mathcal{A}'$ is in normal form for $\leadsto_E$. □

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 60 (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 a linear term of $\mathcal{T}(\mathcal{K}, \mathcal{X})$, $p \neq \lambda$, 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.

Note that finiteness of the set of normal forms of $\mathcal{T}(\mathcal{K})$ w.r.t. $\overrightarrow{E^c_{\mathcal{K}}}$ is decidable. Furthermore, when contracting equations are defined on the whole set $\mathcal{F}$ then there exists an upper bound on the number of states of a simplified automaton: the number of normal forms in $\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}})$.

Lemma 61. *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 $\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}})$.*

Proof. First, we prove that all states q of $\mathcal{S}_E(\mathcal{A})$ recognize at least one normal form of $\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}})$. We make a proof by contradiction. Assume that for all states q of $\mathcal{S}_E(\mathcal{A})$, $\mathcal{L}^{\not\epsilon}(\mathcal{S}_E(\mathcal{A}), q) \cap \text{IRR}(\overrightarrow{E^c_{\mathcal{F}}}) = \emptyset$. 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. $\overrightarrow{E^c_{\mathcal{F}}}$. Let us choose a term s of minimal height. Since s is not in normal form w.r.t. $E^c_{\mathcal{F}}$, we know 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 $u\theta \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q'$. From $u\theta \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q'$, 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 $u|_p\theta \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q''$. 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 get that $q' = q''$. As mentioned above, we know that $C[q'] \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$. Hence $C[u|_p\theta] \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} C[q'] \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$. If $C[u|_p\theta]$ is not in normal form, then this contradicts the fact that $s = C[u\theta]$ which was the smaller term not in normal form recognized by q (by definition, p is different from λ). Thus, $C[u|_p\theta]$ is a normal form w.r.t. $\overrightarrow{E^c_{\mathcal{F}}}$ which contradicts the fact that q recognizes no normal form.

Besides, by definition of $E^c_{\mathcal{F}}$, we know that $\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}})$ is finite. Let $\{t_1, \ldots, t_n\}$ be the subset of $\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}})$ 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 $t_1, \ldots, t_n$ without epsilon transitions because $E \supseteq E^r$ and Lemma 59 entails that $\mathcal{S}_E(\mathcal{A})$ is $\not\epsilon$-deterministic. Now, for all terms s which is not a normal form and that is recognized by a state q of $\mathcal{S}_E(\mathcal{A})$, *i.e.* $s \to^{\not\epsilon *}_{\mathcal{S}_E(\mathcal{A})} q$, 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 $card(\text{IRR}(\overrightarrow{E^c_{\mathcal{F}}}))$ 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 62. *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 61). □

6.3. 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 [14]. 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 $E^c_{\mathcal{C}}$ (rather than of $E^c_{\mathcal{F}}$) 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 $s \to_R t$ then s and t are recognized by equivalent states. This is the role of the set of equations E_R.

Definition 63 ($E_{\mathcal{R}}$). 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 64. *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 61 . 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 61, 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. Variables can be of any sort. In terms, they have

the sort of the term they replace, *e.g.* if $f : S_1 \times \ldots \times S_k \mapsto S$ then the variable x will have the sort S_i in the term $f(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_n)$. 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. We assume that all sorts are inhabited, *i.e.* there is at least one term of each sort. 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 rules $l \to r \in R$ and all equation $u = v \in E$, $l, r, u, v \in \mathcal{T}(\mathcal{F}, \mathcal{X})^{\mathcal{S}}$, l, r, u, v are well-sorted terms, 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 $s \to_{\mathcal{R}}{}^* t$.

Definition 65 (Well-sorted tree automaton). Given a many sorted signature $\mathcal{F}$, a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ is *well-sorted* if all its states recognize well-sorted terms of at most one sort, *i.e.* for all $q \in \mathcal{Q}$, $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{T}(\mathcal{F})^{\mathcal{S}}$ and terms of $\mathcal{L}(\mathcal{A}, q)$ are of the same sort.

Now, we slightly refine the definition of contracting equations for well-sorted terms and add another kind of contracting equations defined on the unique constant symbol of a sort S. For all sorts S, if S has a unique constant symbol we note it c^S. This will be of interest for all recursive datatypes having such a symbol: *nil* for lists, $leaf$ for trees, 0 for natural numbers, etc. Besides equations of the form $u = u|_p$, the following definition permits to define equation of the form $v = c^S$. This is of interest to define equations of the form $cons(x, y) = nil$, though *nil* is not a subterm of $cons(x, y)$.

Definition 66 (Set $E^c_{\mathcal{K},\mathcal{S}}$ of contracting equations for $\mathcal{K}$ and $\mathcal{S}$). The set of well-sorted equations $E^c_{\mathcal{K},\mathcal{S}}$ is *contracting* (for $\mathcal{K} \subseteq \mathcal{F}$) if its equations are of the form (a) $u = u|_p$ with u a linear term of $\mathcal{T}(\mathcal{K}, \mathcal{X})^{\mathcal{S}}$ 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.

We first need to ensure that completion only produces well-sorted tree automata. If completion produces ill-sorted automata this may jeopardize its termination. Ill-sorted terms cannot be rewritten using $\mathcal{R}$ into constructor terms and, thus, cannot be put with $E_{\mathcal{R}}$ into any equivalence class of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=_{E^c_{\mathcal{C},\mathcal{S}}}}$. Thus, recognizing infinitely many ill-sorted terms may require infinitely many states in the tree automaton. The purpose of the following lemma is precisely to show that if the initial tree automaton, TRS and set of equations are well-sorted, so are the completed automata.

Lemma 67 (Completion produces well-sorted automata)**.** *Let $\mathcal{F}$ be a many sorted signature whose set of sorts is $\mathcal{S}$. We assume that each sort of $\mathcal{S}$ is inhabited. Let $\mathcal{A}$ be an epsilon-free tree automaton, $\mathcal{R}$ a tree automaton and E*

a set of equations. If $\mathcal{A}$ is well-sorted and $\mathcal{R}$ and E are sort-preserving, then for all $i \in \mathbb{N}$, $\mathcal{A}^i_{\mathcal{R},E}$ is well-sorted.

Proof. For the proof, the objective is to take advantage of Theorem 17 to show that completion produces tree automata $\mathcal{A}^i_{\mathcal{R},E}$ recognizing no more terms than $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$, *i.e.* for all $i \in \mathbb{N}$, $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. Since $\mathcal{A}$ is well-sorted and $\mathcal{R}$ and E are sort preserving, then $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$ and thus $\mathcal{A}^i_{\mathcal{R},E}$ is well-sorted. The remaining problem is that Theorem 17 needs $\mathcal{R}/E$-coherence of $\mathcal{A}$ to be applied. However, for any $\mathcal{A}$, $\mathcal{R}$ and E the $\mathcal{R}/E$-coherence of $\mathcal{A}$ is not guaranteed. The principle of the proof is simply to add to E an additional set of equations $E^{\mathcal{S}}$ merging together all terms of the same sort. For all sorts $S \in \mathcal{S}$, we denote by $r(S)$ the representative of the sort S, *i.e.* $r(S)$ is an arbitrarily chosen term of $\mathcal{T}(\mathcal{F})^{\mathcal{S}}$ of sort S. We know that such a term exists since all sorts are inhabited. Then the set $E^{\mathcal{S}}$ can be defined as follows: $E^{\mathcal{S}} = \{f(r(S_1), \ldots, r(S_n)) = r(S) \mid \text{for all symbols } f \in \mathcal{F} \text{ and profile } f : S_1 \times \ldots \times \mapsto S\}$. Note that the set $\mathcal{T}(\mathcal{F})^{\mathcal{S}}/_{=E^{\mathcal{S}}}$ has a finite set of equivalence classes which is $\{r(S) \mid S \in \mathcal{S}\}$. Thus for all terms s, t of sort $S \in \mathcal{S}$, $s =_{E^{\mathcal{S}}} t$. Now, let $E' = E \cup E^{\mathcal{S}}$. We now prove that $\mathcal{A}$ is necessarily $\mathcal{R}/E'$-coherent. Since $\mathcal{A}$ is epsilon-free, we only have epsilon free derivations with $\mathcal{A}$, and thus w.r.t. Definition 14, it is enough to show that for all states $q \in \mathcal{Q}$ and all terms $s, t \in \mathcal{T}(\mathcal{F})^{\mathcal{S}}$ if $s \to^{\not\varepsilon *}_{\mathcal{A}} q$ and $t \to^{\not\varepsilon *}_{\mathcal{A}} q$ then $s =_{E'} t$. Since $\mathcal{A}$ is well-sorted, we know that s and t are of the same sort. Using the property of $E^{\mathcal{S}}$ shown above, we know that $s =_{E^{\mathcal{S}}} t$. Since $E' = E \cup E^{\mathcal{S}}$, we thus have $s =_{E'} t$. Thus $\mathcal{A}$ is $\mathcal{R}/E'$-coherent. Applying the Theorem 17, we thus get that for all $i \in \mathbb{N}$, $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'}) \subseteq \mathcal{R}^*_{E'}(\mathcal{L}(\mathcal{A}))$ and $\mathcal{A}^i_{\mathcal{R},E}$ is $\mathcal{R}'/E$-coherent. Since equations of E and $E^{\mathcal{S}}$ are sort preserving, rules of $\mathcal{R}$ are sort preserving and $\mathcal{A}$ is well-sorted, $\mathcal{R}^*_{E'}(\mathcal{L}(\mathcal{A}))$ is well-sorted. Moreover, since $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'}) \subseteq \mathcal{R}^*_{E'}(\mathcal{L}(\mathcal{A}))$ then all terms recognized by $\mathcal{A}^i_{\mathcal{R},E'}$ are well-sorted. Besides, from $\mathcal{R}'/E$-coherence of $\mathcal{A}^i_{\mathcal{R},E'}$ we get that for all terms s and t recognized by the same state q of $\mathcal{A}^i_{\mathcal{R},E'}$ then there exists a term u recognized by $\mathcal{A}^i_{\mathcal{R},E'}$ (and thus well sorted) without epsilon transitions such that $u \to^*_{\mathcal{R}/E'} s$ and $u \to^*_{\mathcal{R}/E'} t$. Since $\mathcal{R}$ and E' are sort preserving then s and t necessarily have the same sort than u. This entails the well-sortedness of $\mathcal{A}^i_{\mathcal{R},E'}$. Finally, $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'})$. More precisely for all states q of $\mathcal{A}^i_{\mathcal{R},E}$ it is possible to find a state q' in $\mathcal{A}^i_{\mathcal{R},E'}$ such that $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}, q) \subseteq \mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'}, q')$. This is an easy consequence of Lemma 16 of [7]. Thus, since $\mathcal{A}^i_{\mathcal{R},E'}$ is well-sorted, so is $\mathcal{A}^i_{\mathcal{R},E}$: for all states q of $\mathcal{A}^i_{\mathcal{R},E}$ there exists a state q' in $\mathcal{A}^i_{\mathcal{R},E'}$ such that $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'})$ and we know by well-sortedness of $\mathcal{A}^i_{\mathcal{R},E'}$ that terms of $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E'})$ are well sorted and all of the same sort. □

Now we can state the termination theorem for completion of sorted TRSs, which is close to the one for the untyped case.

Theorem 68. *Let $\mathcal{A}_0$ be a well-sorted epsilon-free tree automaton, $\mathcal{R}$ 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.*

Proof. Let $\mathcal{A}$ be any tree automaton obtained by completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E. Since $\mathcal{A}_0$ is well-sorted and $\mathcal{R}$ and E are sort-preserving, using Lemma 67, we get that $\mathcal{A}$ is well-sorted. We can thus restrict the proof to well-sorted terms recognized by $\mathcal{A}$. As in Lemma 61, from finiteness of the set normal forms of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ w.r.t. $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$, we can obtain finiteness of the set of states recognizing terms of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ in the completed automaton. The only slight difference comes from rules of the form $u = c^S$. If a term $s \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}$ is not in normal form w.r.t. $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ because the rule $u \to c^S$ applies then we have: $s = C[u\sigma] \to_{\mathcal{A}}{}^* q$. Thus there exists a state q' such that $u\sigma \to_{\mathcal{A}}{}^* q'$. Since c^S is the only constant of sort S and since $u\sigma$ is of sort S, we know that c^S is necessarily a subterm of $u\sigma$ (*e.g.* nil is strictly a subterm of any ground term $cons(a, cons(\ldots))$). Thus there exists a state q'' such that $c^S \to_{\mathcal{A}}{}^* q''$ and since completed automata are simplified, $q' = q''$ and finally $C[c^S] \to_{\mathcal{A}}{}^* q$. As in the proof of Lemma 61, we can iterate the process until finding a normal form of $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$. This entails the finiteness of the set of states recognizing terms of $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ in $\mathcal{A}$. Then, as in the proof of Theorem 64 we can use the fact that $E \supseteq E_{\mathcal{R}}$ to have that terms of $\mathcal{T}(\mathcal{F})^{\mathcal{S}}$ are recognized in $\mathcal{A}$ using a finite set of states. □

6.4. 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 $\mathcal{A}_{\text{IRR}(\mathcal{R})}$. Completions are performed using Timbuk. All the $\mathcal{A}_{\text{IRR}(\mathcal{R})}$ automata and intersections were performed using Taml. All tools are freely available [13]. All completion results have been certified by Coq using the Coq-extracted completion checker [33].

We now give some examples showing how Theorem 68 can be used to perform data-flow static analysis on functional TRSs. The examples we are going to use are simple recursive functions on lists. For sake of simplicity, we chose to note `[a*]` the language of lists with 0 or more occurrences of the symbol `a`. We note `[a*,b*]` the language of lists with 0 or more `a`'s followed by 0 or more `b`'s. We note `[a+,b+]` in the case where there is at least one `a` and one `b` in the list. We note `[(a|b)*]` any list with 0 or more occurrences of `a` or `b` (in any order).

6.5. Basic example

We follow up the example of Section 5.5: the *append* and *reverse* functions. Without equations, completion of this example was not terminating. Now we complete this specification with sets of equations $E_{\mathcal{R}}$, E^r and $E^c_{\mathcal{C},\mathcal{S}}$.

```
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
%%% Ec
cons(X,cons(Y,Z))=cons(Y,Z)
%%% 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
```

The language recognized by automaton $\mathcal{A}_0$ is the set of terms of the form `rev([a+,b+])`. 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 68, 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 $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$[5] recognizing $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$ and the automaton $\mathcal{A}_{\text{IRR}(\{cons(X,cons(Y,Z))\to cons(Y,Z)\})}$. The result of this intersection is:

```
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 [34] or even Timbuk [17] itself. 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 $\text{IRR}(R)$. Since we are dealing with sufficiently complete TRSs, we know that $\text{IRR}(R) \subseteq \mathcal{T}(\mathcal{C})^{\mathcal{S}}$. Thus, we can use again $\mathcal{A}_{\mathcal{T}(\mathcal{C})^{\mathcal{S}}}$ to compute the intersection which 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`'s and `b`'s, *i.e.* `[(a|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(Y,Z)` makes, for instance, $cons(b, cons(a, nil))$ equal

[5] 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 $nil \to qlist$.

to $cons(a, nil)$. Thus, this equation puts in the same equivalence class a list with a 'b' and a list without. The effect of such an equation is to make appear (or disappear) b's (or a's) from the computed lists. However, it is possible to refine $E^c_{\mathcal{C},\mathcal{S}}$ into the following three equations: `cons(a,cons(a,X))=cons(a,X)`, `cons(b,cons(b,X))=cons(b,X)`, and `cons(a,cons(b,cons(a,X)))=cons(a,X)`. This set of equations avoids the previous problem. The two first equations contract two successive elements only if they are the same. The third one is necessary to have a finite $\text{IRR}(\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}})$. On the reverse example it will have no effect since a's and b's cannot be mixed. Again, E verifies the conditions of Theorem 68 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+,a+]`, as expected. In the following, we sum up this result by writing `reverse([a+,b+])=[b+,a+]`.

6.6. More advanced examples: How to gain precision

In this section, we give some more examples of properties we can prove on TRS (and functional programs) using this technique. We also show how to take advantage of Theorem 17 to gain precision when necessary. The next example is the `delete` function defined as follows:

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)))`

On this function, we want to prove that `delete(a,[(a|b)*])=[b*]`, *i.e.* prove that `delete` deletes all occurrences of a in the list but does not remove the other elements. We want to use the same set of contracting equations $E^c_{\mathcal{C}}$ than in the above example, *i.e.* `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)`. We can define the initial language using the automaton:

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`

Completion of `A0` with $\mathcal{R}$ and $E = E^c_{\mathcal{C}} \cup E^r \cup E_{\mathcal{R}}$ terminates but the resulting tree automaton is not precise: it recognizes lists of a's and b's. We loosed precision because the initial tree automaton is not $\mathcal{R}/E$-coherent. Without $\mathcal{R}/E$-coherence of `A0`, completion is still guaranteed to terminate using Theorem 68. However, we cannot take advantage of the precision Theorem 17 to ensure that the completed automaton recognizes no more than terms reachable by R modulo E. To have more precision, we can start completion from the tree automaton `A1`:

Automaton `A1` **States** `qf qa qb qlb qlab qnil` **Final States** `qf`
Transitions `delete(qa,qnil)->qf  delete(qa,qlab)->qf  a->qa  b->qb`
`cons(qa,qnil)->qlab  cons(qb,qnil)->qlb  cons(qb,qlb)->qlb  nil->qnil`
`cons(qa,qlb)->qlab  cons(qa,qlab)->qlab  cons(qb,qlab)->qlab`

Automaton `A1` recognizes the same language as `A0`, *i.e.* terms of the form `delete(a,[(a|b)*])` but is $\mathcal{R}/E$-coherent. Then, using $E_{\mathcal{C}}^c \cup E^r \cup E_{\mathcal{R}}$, Timbuk terminates within milliseconds and the completed automaton recognizes lists of `b`'s, *i.e.* we proved that `delete(a,[(a|b)*])=[b*]`. Defining a $\mathcal{R}/E$-coherent initial tree automaton can be difficult if the automaton badly interfere with the equivalence classes defined by E. However, the problem can be avoided by replacing transitions of the tree automaton by rewrite rules in the TRS itself. For instance, assume that we add the following rules to the TRS $\mathcal{R}$:

```
merge(nil,X)->X    merge(cons(X,Y),U)->cons(X,merge(Y,U))
merge(X,nil)->X    merge(X,cons(Z,U))->cons(Z,merge(X,U))
```

where `merge` corresponds to a function merging non-deterministically two lists. Note that `merge([a*],[b*])` results into `[(a|b)*]`. A tree automaton recognizing terms of the form `delete(a,merge([a*],[b*]))` can be:

Automaton `A0` **States** `qf qa qb qla qlb qnil qm` **Final States** `qf`
Transitions `delete(qa,qm)->qf  merge(qla,qlb)->qm  a->qa  b->qb`
`nil->qnil  cons(qa,qnil)->qla  cons(qa,qla)->qla  cons(qb,qnil)->qlb`
`cons(qb,qlb)->qlb`

This automaton is $\mathcal{R}/E$ coherent w.r.t. $E = E_{\mathcal{C}}^c \cup E^r \cup E_{\mathcal{R}}$, completion terminates and produce a tree automaton recognizing lists of `b`'s, which is the expected result. Since `merge([a*],[b*])` results into `[(a|b)*]`, we made the same proof as above without having to refine the initial tree automaton to make it $\mathcal{R}/E$-coherent. The next example concerns the (insertion) `sort` function defined as follows.

TRS `R`
```
leq(a,a)->true                ite(true,X,Y)->X
leq(a,b)->true                ite(false,X,Y)->Y
leq(b,a)->false               leq(a,b)->true
insert(X,nil)->cons(X,nil)
insert(X,cons(Y,Z))->ite(leq(X,Y),cons(X,cons(Y,Z)),cons(Y,insert(X,Z)))
sort(nil)->nil
sort(cons(X,Y))->insert(X,sort(Y))
```

We want to prove that `sort([((a,b)|(b,a)),(a|b)*])=[a+,b+]`, provided that `a`$\leq$`b` and using the same set of contracting equations as above. Note that `[((a,b)|(b,a)),(a|b)*]` represent the language of lists of `a`'s and `b`'s with at least one `a` and one `b`. Again, if we do not pay attention to $\mathcal{R}/E$-coherence of the initial tree automaton, completion produces an approximation tree automaton which is too imprecise to prove this property. To easily have a $\mathcal{R}/E$-coherent automaton, we use the same encoding as above with the

`merge` function, add the `merge` TRS to `R` and use an initial tree automaton recognizing the language `sort(merge([a+],[b+]))`. Here, `merge([a+],[b+])=[((a,b)|(b,a)),(a|b)*]`. From this initial language and within milliseconds, Timbuk produces a tree automaton with 83 states and 294 transitions. The intersection of this automaton and $\mathcal{A}_{\mathcal{T}(\mathcal{C})^S}$ gives:

States `q5 q4 q3 q2 q1 q0` **Final States** `q5` **Transitions** `nil->q0 B->q1 A->q3 cons(q1,q0)->q2 cons(q3,q2)->q4 cons(q3,q4)->q4 cons(q1,q2)->q2 cons(q3,q2)->q5 cons(q3,q4)->q5`

The language recognized by this automaton is `[a+,b+]`. We thus get an automatic proof that, for infinite lists of two values: `a` and `b`, the `sort` function always outputs a *non-empty sorted list with at least one* `a` *and one* `b`. Some more examples, including a similar proof for a merge sort function, can be found in the Timbuk 3.1 source distribution [13].

7. Related work

TRS classes preserving regularity. The first contribution of this paper is to show that tree automata completion computes exactly the set of reachable terms for some known TRS classes preserving regularity: *i.e. TRS classes such that* $\mathcal{R}^*(S)$ *is regular if* S *is.* Those TRS classes are Ground TRS [23, 24], Linear and Semi-Monadic [26], Linear and (inversely) Growing [16], Linear Generalized Semi-Monadic [30], Linear Finite-Path Overlapping [12], Linear Generalized Finite-Path Overlapping [10], Constructor Based [11]. This property has been shown in two steps. First, we showed that if completion of shifted tree automaton and TRS stops with $E = \emptyset$ then it exactly computes the set of reachable terms. In a second step we showed that completion stops on the Constructor Based and Linear Generalized Finite-Path Overlapping (that contains all the aforementioned classes) classes. On the way, we showed that completion and the algorithm of [12] (in the linear case) are very close: completion produces tree automata that can be simulated by those produced by the algorithm of [12]. This simulation also permitted to show that automata produced by completion can be more compact.

Equational abstraction. When a TRS is outside of the above classes, tree automata completion is able to over-approximate the set of reachable terms. Such approximations can be defined using *equational abstraction* [35]. In [35], the set of reachable terms is not statically built, using a tree automaton as we do, but dynamically explored on the fly using rewriting and abstractions. In completion, we only implement a fragment of the equational abstraction framework: only non conditional equations. On the other side, using equational abstraction with completion does not require to prove additional properties between the TRS and the equations like the coherence property of [35]. Furthermore, we have a termination criterion for the reachability analysis which is based on the finiteness of the set of equivalence classes of $\mathcal{T}(\mathcal{F})/_{=E}$. We believe that such a

termination criterion could also been applied on proofs made using the rewriting framework [35] but, as far as we know, this has not been done.

Tree automata completion. With regards to most papers about completion [17, 12, 10, 6, 19, 7], 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 consideration, like in [36]. 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 [10], 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. The approach followed by [37, 18] is very different. Starting from the TRS, the set of initial terms and a set of "bad" terms that should not be reachable, they tend to directly characterize a correct approximation automaton by constraint solving for the first and by finite model generation for the second. The interest w.r.t. the tree automata completion setting is that it is not necessary to provide a set of approximation equations to over-approximate the set of reachable terms, this is automatized by transforming the completion problem into a satisfiability checking problem. The termination criteria we propose here do not have a counterpart in their setting. Our first criterion permits to build *exactly* an automaton recognizing sets of reachable terms and does not relies on "bad" terms. In this case, algorithms of [37, 18] cannot be applied. Our second criterion does not either require to define "bad" terms. However, in the applications we presented, we could have such sets of terms: *e.g.* prove that `delete(a,[(a|b)*])=[b*]` means proving that from `delete(a,[(a|b)*])` we cannot reach any constructor term in the complement of the language `[b*]`. The strength of [37, 18] is that it does not require to invent approximation equations to prove it. Its weakness is that it may not terminate: there may not exists a regular over-approximation where "bad" terms are not reachable [38] and thus satisfiability checking may diverge. Moreover, when the TRS to analyze is big, the satisfiability checking problem becomes very difficult in practice [37, 18]. In the last paper, the `reverse([a+,b+])=[b+,a+])` problem is also studied and cannot be proven directly. The encoding had to be tweaked by hand for the proof to succeed. There is no such limitation when equations comes into play because a well-chosen set of approximation equations can easily overcome this complexity (which is inherent to the verification problem itself). For instance, equational completion can handle TRSs of thousand rules representing the semantics of a Java program [1, 8].

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 [39, 40, 41], rather than predicting the control flow [42]. All those papers, as well as many other

ones, deal with higher order functions using either tree grammars (for [39]) or higher-order grammar formalisms (PMRS and HORS). The mechanisms used in our work and [39] are very similar. However, using equations, we can tune the approximation more precisely than the fixed "independent attributes" approximation of [39], where all relation between parameters of a function call are lost. In particular, such an approximation is unable to precisely analyze the `reverse` examples of Section 6.6. More details can be found in [8]. Higher-order functions that are considered by [39, 40, 41] are not in the scope of the solution we propose here. However, we obtained some preliminary results [43] showing that an extension to higher order functions is possible and gives some relevant results. We use an encoding of higher-order functions into first-order TRS used by [39]. With this encoding, we obtain results comparable to [41] and better[6] than [39]. Furthermore, in [40], data structures are tagged as input or outputs. This makes it difficult to apply a function to a data structure produced by another function. We do not have such a limitation as it is shown by the combination of the `merge` and `sort` (or `delete`) functions in the example of Section 6.6: the `sort` function directly operates on `merge`'s output. Finally, the verification mechanisms of [41] use automatic abstraction refinement: when the approximation is too coarse it is automatically refined. This can be also performed in the completion setting [44] and adapted to the analysis of functional programs using completion [43].

8. Conclusion and future work

This paper investigates what tree automata completion computes, and when it stops. We surveyed a large number of known TRS classes preserving regularity and showed that tree automata completion terminates for many of them: the linear classes from **G** to **L-GFPO** as well as the **L-Constructor** class. Furthermore, we showed that completion computes exactly the set of reachable terms for those classes. Using Timbuk, which is an optimized implementation of tree automata completion, it is thus possible to efficiently and exactly compute sets of reachable terms for all those classes.

The completion algorithm presented here is not designed to handle non left-linear rules. However, some of the aforementioned completion techniques can [19, 18]. An interesting research direction would be to extend the coverage results of section 5 to completion techniques of [19, 18] with the non left-linear TRS classes preserving regularity. In the same way, completion fails with non left-linear TRSs when the tree automaton to complete is non deterministic (see Example 25). However, we showed in Lemma 41 that completed tree automata are $\not\epsilon$-deterministic. In such automata, a term t can be recognized into two distinct states q_1 and q_2 only if $t \to_{\mathcal{A}}^{\not\epsilon *} q$ (q is unique), $q \to_{\mathcal{A}}{}^{*} q_1$ and $q \to_{\mathcal{A}}{}^{*} q_2$. It should be possible to take advantage of this property to have a tree automata

[6] This is due to the fact that our approximation technique on first order TRS can be tuned to be more precise the fixed "independent attributes" they use.

completion algorithm dealing with non left-linear TRS. Besides, it would be interesting to see if completion also covers TRS classes like [21, 22] where rewriting is applied w.r.t. a given strategy. In the case of the innermost rewriting strategy, this could be a direct consequence of results obtained in [45, 43]. In particular, the lower bound Theorem 16 and the Upper bound Theorem 17 used as basic bricks to prove the main results of this paper both have a counterpart in the case of innermost tree automata completion [45].

For TRS located outside of those classes preserving regularity, termination is not guaranteed and it is necessary to over-approximate the set of reachable terms. Approximations are defined using sets of equations defining so-called equational abstractions. There were some results on the precision of completion w.r.t. $\mathcal{R}_E^*$ [7], but we still did not have any termination guarantees. Our second criterion on the set of approximation equations provides such a guarantee. This criterion restricts the sets of equations for which completion is known to terminate. On a practical point of view, the proposed restriction is strong but it can be refined into a more convenient criterion when the TRS under consideration encodes typed sufficiently-complete functional programs. On some examples, we showed that, using this second criterion, tree automata is a promising alternative to static analysis technique for functional programs. In particular, when we pay attention to $\mathcal{R}/E$-coherence of the initial tree automaton, the analysis is guaranteed to be at least as precise as the approximation defined by the set of equations E. Defining an initial $\mathcal{R}/E$-coherent tree automaton can be technical. In Section 6.6, we showed how to avoid this construction by generating the initial language using the TRS itself. It would be interesting to see if we can generalize this idea using the shifting automaton and TRS of Definition 18. The shifting automaton is $\mathcal{R}/E$-coherent by construction: each state recognize exactly one term of the form $\#_q$. However, this encoding adds new symbols to $\mathcal{F}$: the $\#_q$ terms and termination Theorem 68 cannot be straightforwardly applied nor adapted. This is left for future work.

For completion to provide a decent static analysis technique for functional programs there are still some gaps to bridge. The first one is to deal with higher-order functions. As mentioned in the previous section, one can encode higher-order functions into first-order TRS [39]. However, using this specific encoding makes sufficient-completeness of TRS more difficult to establish and thus termination Theorem 68 difficult to apply. Further research is thus to refine Theorem 68 to adapt it to the TRSs obtained by encoding higher-order functions into TRSs. An interesting open question is: does sufficient completeness of pattern matching in an OCaml [46] function entail sufficient completeness of the generated equivalent TRS? The second gap to bridge is to automatically infer the contracting equations. In Section 6.4, we needed to refine by hand the first trivial equations into a set of three equations, having a better precision. This last set of equation was sufficient to carry out all the proof that we wanted to do on `reverse`, `merge`, `delete` and `sort`. But, how to infer automatically such equations? As far as we know, T. Takai is the only one to have tackled this problem [10]. Using the sticking-out graph, he proposed an algorithm detecting simple loops in TRS application and infering equations of

the form $C[C[x]] = C[x]$, for a ground context $C[\,]$. We have to investigate this further and see if it can, in practice, infer equations precise enough to prove the examples of Section 6.6. Note that, it is also possible to refine equations automatically using Counter Example Guided Abstraction Refinement (CEGAR) completion algorithm [44]. This provides a nice alternative to equation inference. We can start completion with trivial contracting equations automatically built from the signature, *e.g.* $cons(x, y) = y$, and then refine them automatically. This has been successfully used on higher-order functions, see [43]. We also experimented this approach on the examples of Section 6.6. It managed to automatically prove the `delete` and `merge` functions. However, it did not succeed on the `sort` function, the abstraction refinement being to costly to perform on the 17 equations of $E_{\mathcal{C}}^{c} \cup E^{r} \cup E_{\mathcal{R}}$ for this specification. Finally, the two remaining gaps to bridge to use tree automata completion to perform static analysis technique of functional programs are related to the specificities of the languages itself. For instance, OCaml [46] programs use a call-by-value evaluation strategy where Haskell [47] use call-by-need. As far as we know, no static analysis technique for functional programs already takes evaluation strategies into account. Here, a completion-based static analysis is valuable because it can recognizes only innermost descendants [43], *i.e.* values obtained by a call-by-value strategy, where other analysis techniques recognize all possible values independently of their evaluation strategy. This can improve a lot the precision of the analysis. For instance, it is possible to automatically show that a program, having a result with call-by-need but being non terminating with call-by-value, has an empty set of results with call-by-value [43]. Finally, to bridge the last gap related to *real* programming language, it is necessary to take built-in types and values into account. Values manipulated by real functional programs are not always terms or trees. They can be numerals, strings, characters, or terms embedding numerals, etc. Again, for this problem, the completion framework provides another interesting perspective. It is possible to seamlessly plug into completion any abstract domain to represent built-in subterms [48]. The structural part of the term is approximated using tree automata and the built-in part is approximated using lattices and abstract interpretation. Further work thus consists in bridging those gaps and taking advantage of the completion framework to provide a static analysis tool for OCaml programs with built-ins.

Acknowledgments. The author would like to thank the anonymous referees for their comments on this document.

References

[1] Y. Boichut, T. Genet, T. Jensen, L. Leroux, Rewriting Approximations for Fast Prototyping of Static Analyzers, in: RTA'07, Vol. 4533 of LNCS, Springer, 2007, pp. 48–62.

[2] J. Kochems, L. Ong, Improved Functional Flow and Reachability Analyses Using Indexed Linear Tree Grammars, in: RTA'11, Vol. 10 of LIPIcs, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

[3] T. Genet, Y.-M. Tang-Talpin, V. Viet Triem Tong, Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems, in: WITS'2003, Warshaw (Poland), 2003.

[4] 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, L. Vigneron, The AVISPA Tool for the automated validation of internet security protocols and applications, in: CAV'2005, Vol. 3576 of LNCS, Springer, 2005, pp. 281–285.

[5] A. Geser, D. Hofbauer, J. Waldmann, H. Zantema, On tree automata that certify termination of left-linear term rewriting systems, in: RTA'05, Vol. 3467 of LNCS, Springer, 2005, pp. 353–367.

[6] G. Feuillade, T. Genet, V. Viet Triem Tong, Reachability Analysis over Term Rewriting Systems, Journal of Automated Reasonning 33 (3-4) (2004) 341–383.
URL `http://www.irisa.fr/celtique/genet/publications.html`

[7] T. Genet, R. Rusu, Equational tree automata completion, Journal of Symbolic Computation 45 (2010) 574–597.

[8] T. Genet, Reachability analysis of rewriting for software verification, Université de Rennes 1, 2009, habilitation document, `http://www.irisa.fr/celtique/genet/publications.html`.

[9] T. Genet, Towards Static Analysis of Functional Programs using Tree Automata Completion, in: WRLA'14, Vol. 8663 of LNCS, Springer, 2014.

[10] T. Takai, A Verification Technique Using Term Rewriting Systems and Abstract Interpretation, in: RTA'04, Vol. 3091 of LNCS, Springer, 2004, pp. 119–133.

[11] P. Réty, Regular Sets of Descendants for Constructor-based Rewrite Systems, in: Proc. 6th LPAR Conf., Tbilisi (Georgia), Vol. 1705 of LNAI, Springer-Verlag, 1999.

[12] T. Takai, Y. Kaji, H. Seki, Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability, in: RTA'11, Vol. 1833 of LNCS, Springer, 2000.

[13] T. Genet, Y. Boichut, B. Boyer, V. Murat, Y. Salmon, Reachability Analysis and Tree Automata Calculations, IRISA / Université de Rennes 1, `http://www.irisa.fr/celtique/genet/timbuk/`.

[14] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.

[15] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, M. Tommasi, Tree automata techniques and applications, `http://tata.gforge.inria.fr` (2008).

[16] F. Jacquemard, Decidable approximations of term rewriting systems, in: H. Ganzinger (Ed.), Proc. 7th RTA Conf., New Brunswick (New Jersey, USA), Springer-Verlag, 1996, pp. 362–376.

[17] T. Genet, Decidable approximations of sets of descendants and sets of normal forms, in: Proc. 9th RTA Conf., Tsukuba (Japan), Vol. 1379 of LNCS, Springer-Verlag, 1998, pp. 151–165.

[18] A. Lisitsa, Finite Models vs Tree Automata in Safety Verification, in: RTA'12, Vol. 15 of LIPIcs, 2012, pp. 225–239.

[19] Y. Boichut, R. Courbis, P.-C. Héam, O. Kouchnarenko, Handling non left-linear rules when completing tree automata, IJFCS 20 (5).

[20] R. Gilleron, S. Tison, Regular tree languages and rewrite systems, Fundamenta Informaticae 24 (1995) 157–175.

[21] P. Réty, J. Vuotto, Tree automata for rewrite strategies, JSC 40 (1) (2005) 749–794.

[22] A. Gascon, G. Godoy, F. Jacquemard, Closure of Tree Automata Languages under Innermost Rewriting, in: WRS'08, Vol. 237 of ENTCS, Elsevier, 2008, pp. 23–38.

[23] M. Dauchet, S. Tison, The theory of ground rewrite systems is decidable, in: Proc. 5th LICS Symp., Philadelphia (Pa., USA), 1990, pp. 242–248.

[24] W. S. Brainerd, Tree generating regular systems, Information and Control 14 (1969) 217–231.

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

[26] J. Coquidé, M. Dauchet, R. Gilleron, S. Vágvölgyi, Bottom-up tree pushdown automata and rewrite systems, in: R. V. Book (Ed.), Proc. 4th RTA Conf., Como (Italy), Vol. 488 of LNCS, Springer-Verlag, 1991, pp. 287–298.

[27] I. Durand, A. Middeldorp, Decidable call by need computations in term rewriting, in: CADE, Vol. 1249 of LNCS, Springer, 1997, pp. 4–18.

[28] T. Nagaya, Y. Toyama, Decidability for left-linear growing term rewriting systems, in: RTA, Vol. 1631 of LNCS, Springer, 1999, pp. 256–270.

[29] H. Seki, T. Takai, Y. Fujinaka, Y. Kaji, Layered transducing term rewriting system and its recognizability preserving property, in: Proc. 13th RTA Conf., Copenhagen (Denmark), Vol. 2378 of LNCS, Springer-Verlag, 2002.

[30] P. Gyenizse, S. Vágvölgyi, Linear Generalized Semi-Monadic Rewrite Systems Effectively Preserve Recognizability, TCS 194(1-2) (1998) 87–122.

[31] I. Durand, M. Sylvestre, Left-linear bounded trss are inverse recognizability preserving, in: RTA'11, Vol. 10 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[32] S. Tison, Finiteness of the set of E-equivalence classes is undecidable, private communication (2010).

[33] B. Boyer, T. Genet, T. Jensen, Certifying a Tree Automata Completion Checker, in: IJCAR'08, Vol. 5195 of LNCS, Springer, 2008.

[34] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. L. Talcott., All About Maude, A High-Performance Logical Framework, Vol. 4350 of Lecture Notes in Computer Science, Springer, 2007.

[35] J. Meseguer, M. Palomino, N. Martí-Oliet, Equational abstractions, TCS 403 (2-3) (2008) 239–264.

[36] F. Oehl, G. Cécé, O. Kouchnarenko, D. Sinclair, Automatic Approximation for the Verification of Cryptographic Protocols, in: Proc. of FASE'03, Vol. 2629 of LNCS, Springer-Verlag, 2003, pp. 34–48.

[37] Y. Boichut, T. Dao, V. Murat, Characterizing Conclusive Approximations by Logical Formulae, in: RP'11, Vol. 6945 of LNCS, Springer, 2011, pp. 72–84.

[38] Y. Boichut, P.-C. Héam, A theoretical limit for safety verification techniques with regular fix-point computations, IPL 108 (1) (2008) 1–2.

[39] N. D. Jones, N. Andersen, Flow analysis of lazy higher-order functional programs, Theoretical Computer Science 375 (1-3) (2007) 120–136.

[40] N. Kobayashi, N. Tabuchi, H. Unno, Higher-order multi-parameter tree transducers and recursion schemes for program verification, in: POPL'10, ACM, 2010, pp. 495–508.

[41] L. Ong, S. Ramsay, Verifying higher-order functional programs with pattern-matching algebraic data types, in: POPL'11, ACM, 2011.

[42] N. Kobayashi, Model Checking Higher-Order Programs, Journal of the ACM 60.3 (20).

[43] T. Genet, Y. Salmon, Tree Automata Completion for Static Analysis of Functional Programs, Technical report, INRIA, `http://hal.archives-ouvertes.fr/hal-00780124/PDF/main.pdf` (2013).

[44] Y. Boichut, B. Boyer, T. Genet, A. Legay, Equational Abstraction Refinement for Certified Tree Regular Model Checking, in: ICFEM'12, Vol. 7635 of LNCS, Springer, 2012.

[45] T. Genet, Y. Salmon, Reachability Analysis of Innermost Rewriting, in: RTA'15, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, to be published.

[46] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, J. Vouillon, The Objective Caml system release 3.12 – Documentation and user's manual, INRIA, `http://caml.inria.fr/ocaml/` (2012).

[47] The haskell programming language, `http://www.haskell.org` (2012).

[48] T. Genet, T. Le Gall, A. Legay, V. Murat, A Completion Algorithm for Lattice Tree Automata, in: CIAA'13, Vol. 7982 of LNCS, 2013, pp. 134–145.

Appendix A. Completion preserves $\not\epsilon$-determinism, injectivity and $\not\epsilon$-reducibility

Before proving those properties on the completion algorithm, we first have to prove similar properties on the normalization algorithm itself.

Appendix A.1. Properties of $Norm_\Delta$

Lemma 69 ($Norm_\Delta$ preserves transitions of Δ)**.** *Let Δ be an $\not\epsilon$-deterministic set of transitions, q_{new} a new state for Δ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ s.t. there exists no state q' such that $t \to_\Delta^{\not\epsilon *} q'$. If $f(q_1, \ldots, q_n) \to q \in \Delta$ and $f(q_1, \ldots, q_n) \to q' \in Norm_\Delta(t \to q_{new})$ then $q = q'$.*

Proof. The first thing to remark is that the "Δ parameter" of the $Norm$ function only increases and remains $\not\epsilon$deterministic, whatever the recursive calls may be, if its initial value is. This is a consequence of case 2 of the Definition 2 where we add to this parameter the transition $f(q_1, \ldots, q_n) \to q'$ only if there exists no transition $f(q_1, \ldots, q_n) \to q''$ in Δ.

Now, we assume that $f(q_1, \ldots, q_n) \to q \in \Delta$, and we prove that if there is a transition $f(q_1, \ldots, q_n) \to q' \in Norm_\Delta(t \to q_{new})$ then $q = q'$. The proof is done by induction on the number of symbols (of $\mathcal{F}$) in t. If t has one symbol then it is of the form $g(q'_1, \ldots, q'_m)$. We can only apply the case 1 of the definition. If $t \neq f(q_1, \ldots, q_n)$ then the result is $\{t \to q_{new}\}$ and the property trivially holds. The other situation where $t = f(q_1, \ldots, q_n)$ is not possible since $f(q_1, \ldots, q_n) \to q \in \Delta$ contradicts the assumption $t = f(q_1, \ldots, q_n) \not\to_\Delta^{\not\epsilon *} q$.

Now, assume that the property is true for t whose number of symbols is lesser or equal to n. For $f(q_1, \ldots, q_n) \to q'$ to belong to $Norm_\Delta(t \to q_{new})$ it is necessarily added by case 2 of the definition of $Norm$. Hence, there exists a recursive call to $Norm$ of the form $Norm_{\Delta'}(C[f(q_1, \ldots, q_n)] \to q_{new})$ where $C[\,]$ is a non empty context and $\Delta' \supseteq \Delta$. Since the transition $f(q_1, \ldots, q_n) \to q$ is in Δ then it is in Δ' and, by Definition 2, the added transition will be $f(q_1, \ldots, q_n) \to q$ (*i.e.* $q = q'$). Thus, as explained above, we know that Δ' is $\not\epsilon$deterministic and that $f(q_1, \ldots, q_n) \to q \in \Delta'$. Thus $\Delta' \cup \{f(q_1, \ldots, q_n) \to q\}$ is $\not\epsilon$deterministic and we can use the induction hypothesis on $Norm_{\Delta \cup \{f(q_1, \ldots, q_n) \to q\}}(C[q] \to q_{new})$ and obtain that if $f(q_1, \ldots, q_n) \to q'' \in Norm_{\Delta \cup \{f(q_1, \ldots, q_n) \to q\}}(C[q] \to q)$ then $q = q''$. □

Lemma 70 (Result of $Norm_\Delta$ is $\not\epsilon$deterministic)**.** *Let Δ be an $\not\epsilon$-deterministic set of transitions, q_{new} a new state for Δ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ s.t. there exists no state q' such that $t \to_\Delta^{\not\epsilon *} q'$. The set $Norm_\Delta(t \to q_{new})$ is $\not\epsilon$-deterministic.*

Proof. We show this lemma by contradiction. Assume that $Norm_\Delta(t \to q_{new})$ is not $\not\epsilon$deterministic. Thus, there exists a configuration $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$ and two states q, q' such that $q \neq q'$ and $\{c \to q, c \to q'\} \subseteq Norm_\Delta(t \to q_{new})$. Since there are (at least) two transitions in $Norm_\Delta(t \to q_{new})$, we know that (at least) one transition in $\{c \to q, c' \to q\}$ have been added by the case 2 of Definition 2. Now, let $c = f(q_1, \ldots, q_n)$. Assume that $c \to q$ is found in recursive calls of $Norm$ before $c \to q'$. The recursive call is thus of the form: $\{f(q_1, \ldots, q_n) \to$

$q\} \cup Norm_{\Delta' \cup \{f(q_1,\ldots,q_n)\to q\}}(C[q] \to q_{new})$, where $\Delta' \supseteq \Delta$. Furthermore since $c \to q'$ is not in Δ', it is in $Norm_{\Delta' \cup \{f(q_1,\ldots,q_n)\to q\}}(C[q] \to q_{new})$. However, using Lemma 69, we obtain that $q = q'$ which is a contradiction. □

Definition 71 (Injective fractions). Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $q \in \mathcal{Q}$ and $c, c' \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$. The set $\Delta' \subseteq \Delta$ is an *injective fraction* of Δ if Δ' is injective and for all transitions $c \to q \in \Delta'$ there is no transition $c' \to q \in \Delta$. Note that the injective fraction notion does not depend on ϵ-transitions.

In the following, *injective normalization* denotes a normalization performed using an injective fraction of the set of transitions. Now, we prove that normalization $Norm_\Delta$ preserves injectivity of Δ.

Lemma 72 (Normalization preserves injectivity)**.** *Let Δ be a set of transitions, q a new state for Δ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$ a configuration. If Δ is injective then so is $\Delta \cup Norm_\Delta(t \to q)$.*

Proof. We prove this by contradiction. Assume that there exist configurations $c, c' \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$ with $c \neq c'$ and $\{c \to q', c' \to q'\} \subseteq (\Delta \cup Norm_\Delta(t \to q))$.

- If $c \to q'$ and $c' \to q'$ are all exclusively in Δ then $c \neq c'$ contradicts injectivity of Δ;
- Assume that $c \to q'$ and $c' \to q'$ are all exclusively in $Norm_\Delta(t \to q)$. Since there are at least two transitions in $Norm_\Delta(t \to q_{new})$, we know that at least one transition in $\{c \to q', c' \to q'\}$ have been added by the case 2. of Definition 2. Now, assume that one of the configurations, say c, is of the form $c = f(q_1, \ldots, q_n)$ and one recursive call of $Norm_\Delta(t \to q)$ is of the form: $\{f(q_1, \ldots, q_n) \to q'\} \cup Norm_{\Delta \cup \{f(q_1,\ldots,q_n)\to q'\}}(C[q'] \to q)$. However, in all following recursive calls of $Norm$, the state q' will no longer be new for $\Delta \cup \{f(q_1, \ldots, q_n) \to q'\}$ and no other transition leading to q' can be added except the same transition $f(q_1, \ldots, q_n) \to q'$. In this case $c = c'$ which is a contradiction;
- Finally, assume that $c \to q' \in \Delta$ and $c' \to q' \in Norm_\Delta(t \to q)$. Assume that $c' \to q'$ is added to $Norm_\Delta(t \to q)$ using the first case of the definition of $Norm$. Then $q = q'$ but since $c \to q' \in \Delta$ this contradicts the fact that q is new for Δ. Assume, now, that $c' \to q'$ is added to $Norm_\Delta(t \to q)$ using the second case of the definition. As above, this means that $c' = f(q_1, \ldots, q_n)$ and one recursive call of $Norm$ is of the form: $\{f(q_1, \ldots, q_n) \to q'\} \cup Norm_{\Delta \cup \{f(q_1,\ldots,q_n)\to q'\}}(C[q'] \to q)$. However, in this case of the definition, either the transition $f(q_1, \ldots, q_n) \to q' \in \Delta$ or q' is a new state for Δ. If $f(q_1, \ldots, q_n) \to q' \in \Delta$ then since, $c' \to q' \in \Delta$ with $c' \neq c = f(q_1, \ldots, q_n)$ we have a contradiction to the injectivity of Δ. If this case of the definition applies with q' new state for Δ then this contradicts the fact that $c' \to q' \in \Delta$.

□

Lemma 73 (Normalization preserves languages of existing states)**.** *Let Δ and Δ' be two sets of transitions such that Δ' is an injective fraction of Δ, q a new state for Δ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$ a configuration. For all states q' of Δ the language recognized by q' is the same with Δ and $\Delta \cup Norm_{\Delta'}(t \to q)$.*

Proof. This property is true by construction of the $Norm$ function since q is a new state for Δ and $Norm$ only adds a transition $f(q_1, \ldots, q_n) \to q''$ if q'' is not in Δ' or if this transition is present in Δ' (which is injective). □

Appendix A.2. Properties of completion

Lemma (Completion preserves $\not\epsilon$determinism)**.** *Let $\mathcal{R}$ be a TRS and $\mathcal{A}$ an $\not\epsilon$-deterministic automaton. For all $i \in \mathbb{N}$, $\mathcal{A}^i_{\mathcal{R},\emptyset}$ is $\not\epsilon$-deterministic.*

Proof. The proof is done by induction on i. The case $i = 0$ trivially holds. Now, we assume that the property holds for $\mathcal{A}^i_{\mathcal{R},\emptyset}$ and we prove that $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$ is $\not\epsilon$deterministic. If $\mathcal{A}^i_{\mathcal{R},\emptyset}$ was a fixpoint then $\mathcal{A}^i_{\mathcal{R},\emptyset} = \mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$ and thus is $\not\epsilon$ deterministic. Otherwise, since E is empty the only possibility for completion to change $\mathcal{A}^i_{\mathcal{R},\emptyset}$ is to solve at least one critical pair and add some transitions to $\mathcal{A}^i_{\mathcal{R},\emptyset}$ to build $\mathcal{A}^{i+1}_{\mathcal{R},\emptyset}$. Now, we show that solving one critical pair preserves $\not\epsilon$determinism, this can be easily generalized to any number of critical pairs using induction. Let Δ^i be the set of transitions of $\mathcal{A}^i_{\mathcal{R},\emptyset}$. By definition 6, to solve a critical pair $l\sigma \to^*_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$ and $l\sigma \to_{\mathcal{R}} r\sigma$, if $r\sigma \to^{\not\epsilon *}_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$ then adding the transition $q' \to q$ is enough. It is easy to see that, by definition of $\not\epsilon$ determinism, if Δ_i is $\not\epsilon$deterministic then so is $\Delta^i \cup \{q' \to q\}$. If there exists no state q' such that $r\sigma \to^{\not\epsilon *}_{\mathcal{A}^i_{\mathcal{R},\emptyset}} q$, to solve the critical pair the set of transitions becomes $\Delta^i \cup Norm_{\Delta^i}(r\sigma \to q') \cup \{q' \to q\}$ where q' is a new state for Δ^i. Using Lemma 70, we get that $Norm_{\Delta^i}(r\sigma \to q')$ is $\not\epsilon$deterministic and using Lemma 69 that its union with Δ^i is also $\not\epsilon$deterministic. Finally, as above, adding $q' \to q$ does not jeopardize $\not\epsilon$determinism. □

Lemma (Completion preserves injectivity)**.** *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ be a tree automaton whose set of transition is injective. For all $i \in \mathbb{N}$, the set of transitions of $\mathcal{A}^i_{\mathcal{R},\emptyset}$ is injective.*

Proof. This can be proven by induction on i. This is trivially true for $\mathcal{A}^i_{\mathcal{R},\emptyset}$. Since the set of approximation equations E is empty, transitions are only modified and added by the completion operation. Recall that a completion step consists in adding transitions $Norm_\Delta(r\sigma \to q')$ and $q' \to q$. Adding $q' \to q$ has no effect on injectivity and by Lemma 73, we know that adding transitions $Norm_\Delta(r\sigma \to q')$ does not jeopardize injectivity provided that Δ is, which is the induction hypothesis. □

Lemma (Completion preserves $\not\epsilon$reducibility)**.** *Let $\mathcal{R}$ be a TRS, E a set of equations and $\mathcal{A}$ a $\not\epsilon$-reduced tree automaton. For any $i \in \mathbb{N}$: $\mathcal{A}^n_{\mathcal{R},E}$ is $\not\epsilon$-reduced.*

Proof. We prove this by induction on i. For $i = 0$, $\mathcal{A}^i_{\mathcal{R},E} = \mathcal{A}$ and is thus $\not\epsilon$reduced by assumption. Assume that the property is true for $\mathcal{A}^n_{\mathcal{R},E}$, we prove that it is true for $\mathcal{A}^{n+1}_{\mathcal{R},E}$. The automaton $\mathcal{A}^{n+1}_{\mathcal{R},E}$ is obtained from $\mathcal{A}^n_{\mathcal{R},E}$ by applying $\mathcal{C}_\mathcal{R}$ to $\mathcal{A}^n_{\mathcal{R},E}$ and then simplifying $\mathcal{C}_\mathcal{R}(\mathcal{A}^n_{\mathcal{R},E})$ by E. We first prove that $\mathcal{C}_\mathcal{R}(\mathcal{A}^n_{\mathcal{R},E})$ is $\not\epsilon$reduced. The operator $\mathcal{C}_\mathcal{R}$ can only add transitions to states of $\mathcal{A}^n_{\mathcal{R},E}$ or creates new states. Since states of $\mathcal{A}^n_{\mathcal{R},E}$ are $\not\epsilon$reachable and $\mathcal{C}_\mathcal{R}$ only adds transitions to those states, they remain reachable. For new states, by definition of $\mathcal{C}_\mathcal{R}$, we know that new states come from the normalization with $Norm$ of transitions of the form $r\sigma \to q$ where σ maps states $q_1, \ldots, q_n$ of $\mathcal{A}^n_{\mathcal{R},E}$ to variables $x_1, \ldots, x_n$ of r. Since states of $\mathcal{A}^n_{\mathcal{R},E}$ are $\not\epsilon$reachable, there exists terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ such that $t_i \to^{\not\epsilon *}_{\mathcal{A}^p_{\mathcal{R},E}} q_i$ for $1 \leq i \leq n$. By construction of the $Norm$ function, all new states created by this function recognize subterms of $r\sigma$. Let $r\sigma|_p$ be a subterm and q the state created by $Norm$ to recognize it. By construction of $Norm$ we thus have $r\sigma|_p \to^{\not\epsilon *}_{\mathcal{A}^p_{\mathcal{R},E}} q$. Let μ be the substitution such that $\mu(x_i) = t_i$ for $1 \leq i \leq n$. We thus have $r\mu \to^{\not\epsilon *}_{\mathcal{A}^p_{\mathcal{R},E}} r\sigma \to^{\not\epsilon *}_{\mathcal{A}^p_{\mathcal{R},E}} q$. Hence, any new state q produced by $Norm$ is $\not\epsilon$reachable and so are states already present in $\mathcal{A}^n_{\mathcal{R},E}$. Thus $\mathcal{C}_\mathcal{R}(\mathcal{A}^n_{\mathcal{R},E})$ is $\not\epsilon$reducible. Besides, it is easy to see that simplification with any set of equation E preserves $\not\epsilon$reducibility. Simplification replaces all occurrences of a state q' by a state q. If all states of the automaton are $\not\epsilon$deterministic then so is q. Replacing q' by q does not change $\not\epsilon$reachability of q itself. Moreover, it does not change the $\not\epsilon$reachability of any other state. Indeed, any transition of the form $f(q_1, \ldots, q', \ldots, q_n) \to q_f$ used to make q_f $\not\epsilon$reachable becomes $f(q_1, \ldots, q, \ldots q_n) \to q_f$ where q is $\not\epsilon$ reachable and thus q_f remains $\not\epsilon$reachable. Finally, since $\mathcal{A}^{n+1}_{\mathcal{R},E}$ is obtained from the $\not\epsilon$reachable automaton $\mathcal{C}_\mathcal{R}(\mathcal{A}^n_{\mathcal{R},E})$ by several simplification steps that preserve $\not\epsilon$reducibility, the automaton $\mathcal{A}^{n+1}_{\mathcal{R},E}$ is $\not\epsilon$reduced. □