Equational Approximations for Tree Automata Completion

Thomas Genet

IRISA/Université de Rennes 1, Campus Beaulieu, F-35042 Rennes Cedex

Vlad Rusu

IRISA/INRIA, Campus Beaulieu, F-35042 Rennes Cedex

Abstract

In this paper we deal with the verification of safety properties of infinite-state systems modeled by term-rewriting systems. An over-approximation of the set of reachable terms of a term-rewriting system $\mathcal{R}$ is obtained by automatically constructing a finite tree automaton. The construction is parameterized by a set E of equations on terms, and we also show that the approximating automata recognize at most the set of $\mathcal{R}/E$-reachable terms. Finally, we present some experiments carried out with the implementation of our algorithm. In particular, we show how some approximations from the literature can be defined using equational approximations.

Key words: Verification, Term rewriting systems, Reachability, Tree automata, Rewriting modulo equations

1. Introduction, motivation, and related work

Designing verification techniques that are able to handle infinite-state systems is a major challenge. In particular, techniques based on tree automata have been proposed. In this framework, the system under verification is modeled by a term-rewriting system (hereafter abbreviated as TRS) or a tree transducer, and verification can be performed using reachability analysis. Some techniques are based on *tree automata completion* (Feuillade et al., 2004; Takai, 2004; Gallagher and Rosendahl, 2008); other techniques are based on regular tree model checking (Bouajjani et al., 2006a). These techniques have

[1] This research was partially supported by ANR grant number ANR-06-SETI-14

Email addresses: genet@irisa.fr (Thomas Genet), rusu@irisa.fr (Vlad Rusu).

URLs: www.irisa.fr/celtique/genet (Thomas Genet), www.irisa.fr/vertecs/Equipe/Rusu (Vlad Rusu).

been used for the verification of various kinds of programs, at various level of abstraction: abstract models of distributed systems (Bouajjani and Touili, 2005), communication protocols (Bouajjani et al., 2006a), cryptographic protocols (Genet and Klay, 2000; Boichut et al., 2004), low-level models of C programs (Bouajjani et al., 2006b) and of Java bytecode programs (Boichut et al., 2007).

This paper continues a series of works on tree automata completion, proposed earlier in (Feuillade et al., 2004; Genet, 1998; Genet and Viet Triem Tong, 2001a). We briefly describe that approach and motivate the work in the present paper. Given a term language defined by a tree automaton $\mathcal{A}$, and a term rewriting system $\mathcal{R}$, the tree automata completion algorithm produces another tree automaton $\mathcal{A}^*_{\mathcal{R}}$. For some specific classes of TRS, $\mathcal{A}^*_{\mathcal{R}}$ recognizes exactly the language of terms that are reachable by rewriting with $\mathcal{R}$ the terms recognized by $\mathcal{A}$. Otherwise, the completed tree automaton $\mathcal{A}^*_{\mathcal{R}}$ recognizes an over-approximation of those reachable terms. In (Genet, 1998) approximations are constructed automatically. In (Feuillade et al., 2004; Genet and Viet Triem Tong, 2001a), the approximations are defined by the user under the form of *normalization rules*.

Experiments on non-trivial case studies (Genet et al., 2003; Boichut et al., 2007) have been carried out with an implementation of the completion algorithm in the Timbuk tool (Genet and Viet Triem Tong, 2001b). They have shown that normalization rules are a useful approach for defining approximations. In particular, the rules can be adapted to the program and the property being verified, which make them more flexible than the automatic approximations of (Genet, 1998). However, defining a set of normalization rules for a particular approximation remains a complex task. The main reason is that normalization rules directly refer to the structure of tree automata. Hence, to define such rules, the user needs to be highly familiar with the tree automata formalism.

In this paper, we replace normalization rules by *equations* as the main approximation technique. The idea is inspired from *equational abstractions in rewriting logic* (Meseguer et al., 2003) and from the *rewriting modulo equations* framework (Baader and Nipkow, 1998). We illustrate it on a simple example. Consider the TRS $\mathcal{R} = \{f(x) \to f(s(s(x)))\}$, and assume that we want to prove that $f(a) \not\to_{\mathcal{R}}{}^* f(s(a))$. This cannot be done simply be enumerating all reachable terms, since there are infinitely many such terms. However, a finite-state abstraction of the set of reachable terms can be simply obtained by *not distinguishing* between the terms of the form $s(s(x))$ and the term x, i.e., by using the equation $E = \{s(s(x) = x\}$. Then, all the terms reachable from $f(a)$: $f(s(s(a))), f(s(s(s(s(a))))), \ldots$ are *equivalent modulo E* and form a finite set (here, a singleton). To prove $f(a) \not\to_{\mathcal{R}}{}^* f(s(a))$ we just have to check that $f(s(a))$ is not in this set.

1.1. Related work

Regular tree languages have been used to analyze programs for a long time. As noted by J. Gallagher & M. Rosendahl (Gallagher and Rosendahl, 2008), our tree automata completion technique is very close to the flow analysis technique for functional programs (Reynolds, 1969; Jones and Andersen, 2007). These techniques are based on the same principle: completion of a regular language. However, like in many static analysis techniques, the approximation used in (Reynolds, 1969; Jones and Andersen, 2007) is built into the algorithm. By contrast, by using equations, we define approximations independently of the algorithm. By changing the set of equations, it is possible to perform different analyses, tailored for verifying different properties. In Section 7 we show that the analysis of (Jones and Andersen, 2007) can be prototyped using tree automata completion and a definition of their approximation using equations.

Another very efficient completion procedure, proposed in (Gallagher and Rosendahl, 2008), is based on an encoding of both tree automata and term-rewriting systems into Horn clauses. This encoding allows the authors to use state-of-the-art static analysis tools for logic programs to perform approximations.

The rewriting-modulo framework implemented in the Maude tool (Clavel et al., 2007) leads quite naturally to *equational abstractions* (Meseguer et al., 2003): new equations are added to the set of equations defining the system's state in order to make the system *finite*-state. However, in order to be over-approximations as required by verification, equational abstractions have to fit in the general rewriting-modulo framework of Maude. That is, the new set of equations, which consists of equations defining the state of the system, and of the approximating equations, must be *ground confluent and terminating* and must be *ground coherent* with respect to the TRS modeling the program under verification (coherence is condition similar to confluence). Such properties are undecidable, and the user must check them manually or by interacting with specialized tools. By contrast, we only require the syntactical condition of *left-linearity* on our term-rewriting systems, and impose no condition on equations. On the other hand, equational abstractions are able to deal with temporal-logic properties, which are not considered yet in our approach.

The use of equations for approximating with tree automata was already experimented in (Takai, 2004) but with strong syntactical restrictions on their form.

Several works are considering the transformation of tree automata using tree transducers rather than term rewriting systems. Those works are commonly known as *Regular Tree Model-Checking*. In this setting, model checking consists in building the tree automaton representing any number of applications of the tree transducer. This automaton can be constructed either by iterating the tree transducer like in (Bouajjani and Touili, 2002) or directly by computing the closure of the tree transducer (Abdulla et al., 2006).

Finally, the present work deals only with fixed-rank symbols. Reachability analysis can be performed on variadic terms using hedge automata, in an exact or approximated way (d'Orso and Touili, 2006; Genest et al., 2008; Jacquemard and Rusinowitch, 2008).

1.2. Contributions

The first contribution is to define equational approximations of tree automata. We propose an algorithm that, given an automaton $\mathcal{A}$, a term rewriting system $\mathcal{R}$ and a set of equations E, computes a tree automaton denoted by $\mathcal{A}^*_{\mathcal{R},E}$. The second contribution consists in proving lower and upper bounds for the language $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$:

- We prove that a lower bound for $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ is $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$, i.e., *the set of terms reachable by rewriting using* $\mathcal{R}$ *the terms recognized by* $\mathcal{A}$. This lower bound enables us to verify *safety properties*: assume the automaton $\mathcal{A}$ represents the set of initial states of a system whose dynamics is defined by a TRS $\mathcal{R}$, and the safety property to be verified is that some "unsafe" states, recognized by a tree automaton $\mathcal{A}'$, are unreachable. We verify the property by checking that the intersection $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \cap \mathcal{L}(\mathcal{A}')$ is empty. If this is the case, then, due to the inclusion $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ established here, we obtain that the intersection $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \cap \mathcal{L}(\mathcal{A}')$ is also empty, i.e., the property is satisfied.
- Assuming that $\mathcal{A}$ satisfies a technical condition called *$\mathcal{R}/E$-coherence*, we prove that an upper bound for $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ is $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$, i.e. *the set of terms reachable by rewriting with* $\mathcal{R}$ *modulo* E *the language of* $\mathcal{A}$. This upper bound demonstrates the *precision* of our approximation: it shows that the computed approximation stays within the "expected" approximation induced by the equations E. Note that our objective is *not* to compute the set of terms $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$, but only a subset of it containing $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ - the smaller the subset, the more precise the approximation. For two terms s and t such that $s =_E t$, if s is reachable, but t is not, our over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ may contain s but not contain t. This is more precise than $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A})$, which contains both terms since $s \rightarrow_{\mathcal{R}/E} t$.

The third contribution is an implementation of the completion algorithm in the Timbuk tool and some experiments. The experiments show that approximations defined using our former formalism (Genet and Viet Triem Tong, 2001a), can be described in a more concise way using equations. We also show that some theoretical static analyzes of the literature, based on regular languages, can easily be implemented using equations.

1.3. Outline

Section 2 covers prerequisites for term-rewriting systems and tree automata. Section 3 defines *simplification*, which corresponds to the application of a set of equations to a tree automaton. Section 4 defines *$\mathcal{R}/E$-coherence*, which is a key property for proving our precision result. We prove that the simplification operation preserves the $\mathcal{R}/E$-coherence property. In Section 5 we define our "basic" tree automata completion algorithm (without simplification). We show that the completion operation preserves the $\mathcal{R}/E$-coherence property as well. Then, in Section 6 we present our main algorithm, which alternates simplification steps and completion steps and produces the automaton $\mathcal{A}^*_{\mathcal{R},E}$. We prove the inclusions $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ and $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. Finally, in Section 7 we present some experiments with an implementation of our main algorithm in Timbuk.

2. Preliminaries

Comprehensive surveys can be found in (Dershowitz and Jouannaud, 1990; Baader and Nipkow, 1998) for term-rewriting systems, and in (Comon et al., 2008; Gilleron and Tison, 1995) for tree automata and tree language theory.

Let $\mathcal{F}$ be a finite set of symbols, each associated with an arity, 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 $\mathcal{T}(\mathcal{F},\mathcal{X})$ also denoted by σ. A term rewriting system $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A *set of equations* E is a set of pairs of the form $l = r$ where $l, r \in \mathcal{T}(\mathcal{F},\mathcal{X})$. We assume that the application of a substitution on a term, and the rewriting of a term with a rule, are known to the reader. A rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if each variable of l (resp. r) occurs only once in l (resp. r). A TRS $\mathcal{R}$ is left-linear if every rewrite rule $l \to r$ of $\mathcal{R}$ is left-linear. A set of equations is *linear* if all members of all equations are linear. The TRS $\mathcal{R}$ induces a rewriting relation $\to_{\mathcal{R}}$ on terms whose reflexive-transitive closure is denoted by $\to^*_{\mathcal{R}}$.

Definition 1 (E-equivalence)**.** For two ground terms $t, t' \in \mathcal{T}(\mathcal{F})$ and an equation $e : l = r$, we say that $t =_e t'$ if there exists a substitution $\tau : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $l\tau = t$ and $r\tau = t'$. The equivalence relation $=_E \subseteq \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F})$ is the smallest congruence containing the relation $\{(t,t') \in \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F}) | \exists e \in E.\, t =_e t'\}$. $\diamond$

Definition 2 ($\mathcal{R}$-descendants)**.** The set of $\mathcal{R}$-descendants of a language $\mathcal{L} \subseteq \mathcal{T}(\mathcal{F})$ is $\mathcal{R}^*(\mathcal{L}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in \mathcal{L}.\ s \to^*_{\mathcal{R}} t\}$. $\diamond$

Note that $\mathcal{R}^*(\mathcal{L})$ is possibly infinite: $\mathcal{R}$ may not terminate and/or $\mathcal{L}$ may be infinite, and the set $\mathcal{R}^*(\mathcal{L})$ is not regular in general (Gilleron and Tison, 1995). The $\mathcal{R}$-descendants of a language can be approximated by its set of $\mathcal{R}/E$-descendants, defined as follows:

Definition 3 ($\mathcal{R}/E$-descendants)**.** Given a TRS $\mathcal{R}$ and a set of equations E, the relation $\to_{\mathcal{R}/E} \subseteq \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F})$ is defined by $s \to_{\mathcal{R}/E} t$ *if there exist* $s', t' \in \mathcal{T}(\mathcal{F})$ *such that* $s =_E s' \to_{\mathcal{R}} t' =_E t$. The relation $\to^*_{\mathcal{R}/E}$ is the reflexive-transitive closure of $\to_{\mathcal{R}/E}$. The set of $\mathcal{R}/E$-descendants of a language is $\mathcal{R}^*_E(\mathcal{L}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in \mathcal{L}$ s.t. $s \to^*_{\mathcal{R}/E} t\}$. $\diamond$

We now define tree automata. Let $\mathcal{Q}$ be a finite set of symbols with arity 0, called *states*, such that $\mathcal{Q} \cap \mathcal{F} = \emptyset$. $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is called the set of *configurations*.

Definition 4 (Transition, normalized transition, and ϵ-transition)**.** A *transition* is a rewrite rule $c \to q$, where c is a configuration i.e. $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. A *normalized transition* is a transition $c \to q$ where $c = f(q_1, \ldots, q_n)$, for syme symbol $f \in \mathcal{F}$ whose arity is n, and $q_1, \ldots, q_n \in \mathcal{Q}$. An *$\epsilon$-transition* $c \to q$ is such that $c \in \mathcal{Q}$. $\diamond$

Definition 5 (Tree automaton)**.** A (bottom-up, non-deterministic, finite) tree automaton, simply called a tree automaton or an automaton in the sequel, is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f \subseteq \mathcal{Q}$ and Δ is a set of normalized transitions and of ϵ-transitions. $\diamond$

The *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is that induced by the transitions Δ of $\mathcal{A}$ and is denoted by is $\rightarrow_\Delta$ or simply by $\rightarrow_\mathcal{A}$. Similarly, we often write $c \rightarrow q \in \mathcal{A}$ instead of $c \rightarrow q \in \Delta$, and $q \in \mathcal{A}$ instead of $q \in \mathcal{Q}$, where $\mathcal{Q}$ is the set of states of $\mathcal{A}$.

Definition 6 (Recognized language)**.** The tree language recognized by $\mathcal{A}$ in a state q is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow_\mathcal{A}^* q\}$. The language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q)$. $\diamond$

We also define ϵ- and ϵ-free derivations, which are necessary both for the new completion algorithm and the application of equations to a tree automaton.

Definition 7 (ϵ- and ϵ-free derivations)**.** We denote by $\xrightarrow{\epsilon}_\mathcal{A}$ one-step rewritings performed by using an ϵ-transition of $\mathcal{A}$, and by $\rightarrow_\mathcal{A}^{\not\epsilon}$ one-step rewritings performed by using a transition of $\mathcal{A}$ other than an ϵ-transition. The relation $\xrightarrow{\epsilon}_\mathcal{A}^*$ is the reflexive transitive-closure of $\xrightarrow{\epsilon}$, and the relation $\rightarrow_\mathcal{A}^{\not\epsilon *}$ is the reflexive transitive-closure of $\rightarrow_\mathcal{A}^{\not\epsilon}$. $\diamond$

The completion and simplification operations, defined later in the paper, rely on specific substitutions that map variables to states.

Definition 8 (Sets $\Sigma(\mathcal{Q}, \mathcal{X})$ and $\Sigma(\mathcal{T}(\mathcal{F}), \mathcal{Q})$)**.** The set $\Sigma(\mathcal{Q}, \mathcal{X})$ (resp. $\Sigma(\mathcal{T}(\mathcal{F}), \mathcal{Q})$) contains all substitutions mapping a variable of $\mathcal{X}$ to a state of $\mathcal{Q}$ (resp. a state of $\mathcal{Q}$ to a ground term of $\mathcal{T}(\mathcal{F})$). $\diamond$

Example 9 (Tree automaton, recognized language, and substitutions)**.** Let $\mathcal{F} = \{f, a, b\}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q} = \{q_1, q_2\}$, $\mathcal{Q}_f = \{q_1\}$, and $\Delta = \{f(q_1) \rightarrow q_1, a \rightarrow q_1, b \rightarrow q_2, q_2 \rightarrow q_1\}$. The languages recognized by q_1 and q_2 are the following: $\mathcal{L}(\mathcal{A}, q_1)$ is the set of terms built on $\{f, a, b\}$, i.e. $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{f, a, b\})$, and $\mathcal{L}(\mathcal{A}, q_2) = \{b\}$. We can also note that $f(b) \rightarrow_\mathcal{A}{}^* q_1$, $b \rightarrow_\mathcal{A}^{\not\epsilon *} q_2$ and $f(b) \not\rightarrow_\mathcal{A}^{\not\epsilon *} q_1$. The substitution $\sigma = \{x \mapsto q_1, y \mapsto q_2\}$ belongs to $\Sigma(\mathcal{Q}, \mathcal{X})$ and $f(x, y)\sigma = f(q_1, q_2)$.

3. Simplification of Tree Automata by Equations

In this section, we define the *simplification* operation of a tree automaton $\mathcal{A}$ with respect to a set of equations E. We prove that the simplification of an automaton has the effect of over-approximating the recognized language of the automaton. The simplification operation is based on renaming states.

Definition 10 (Renaming states in tree automata)**.** Let $\mathcal{Q}, \mathcal{Q}'$ be sets 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. $\diamond$

When $\alpha = \{q_a \mapsto q_b\}$, $\mathcal{A}' = \mathcal{A}\alpha$ is the automaton where every occurrence of q_a has been replaced by q_b.

Example 11. Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ with $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \rightarrow q_0, f(q_0) \rightarrow q_0, a \rightarrow q_1\}$. $\mathcal{A}\{q_0 \mapsto q_2\} = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}'_f, \Delta' \rangle$ with $\mathcal{Q}' = \{q_1, q_2\}$, $\mathcal{Q}'_f = \{q_2\}$ and $\Delta = \{f(q_1) \rightarrow q_2, f(q_2) \rightarrow q_2, a \rightarrow q_1\}$.

The following lemma shows that every term recognized in $\mathcal{A}$ is also recognized in $\mathcal{A}'$.

Lemma 12. *Let $\mathcal{A}, \mathcal{A}'$ be tree automata and q, q_a, q_b states of $\mathcal{A}$ such that $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. For all terms $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$:*

- *if $t \rightarrow_{\mathcal{A}}^{\not\epsilon *} q$ then $t\{q_a \mapsto q_b\} \rightarrow_{\mathcal{A}'}^{\not\epsilon *} q\{q_a \mapsto q_b\}$*
- *if $t \rightarrow_{\mathcal{A}}{}^{*} q$ then $t\{q_a \mapsto q_b\} \rightarrow_{\mathcal{A}'}^{*} q\{q_a \mapsto q_b\}$*

Proof. We sketch the proof for $t\{q_a \mapsto q_b\} \rightarrow_{\mathcal{A}}{}^{*} q \implies t \rightarrow_{\mathcal{A}'}^{*} q\{q_a \mapsto q_b\}$. The proof for the other case is simpler. We proceed by structural induction on the term t.

(1) In the base case, t is either a constant or a state.

  (a) if t is a state then the derivation $t \rightarrow_{\mathcal{A}}{}^{*} q$ has the form $t = q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_n = q$. We prove by induction on the number of occurrences of q_a in that derivation that $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$. The base case is trivial because, then, all transitions of $\mathcal{A}$ used in the derivation $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_n$ are also transitions of $\mathcal{A}'$. For the induction step, let $k \in \{1, \ldots, n\}$ be the index of the "next" occurrence of q_a in our sequence. We need to distinguish several cases, depending on whether ($k = 1$ or $k > 1$) and ($k < n$ or $k = n$). We show the proof for the case when $k > 1$ and $k < n$. On the one hand, there exist derivations $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_{k-1}$ and $q_{k+1} \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_n$ with fewer instances of q_a, and by induction hypothesis, $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_{k-1}$ and $q_{k+1} \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$ On the other hand, by definition of renaming, the transitions $q_{k-1} \rightarrow q_a$ and $q_a \rightarrow q_{k+1}$ of $\mathcal{A}$ are replaced in $\mathcal{A}'$ by $q_{k-1} \rightarrow q_b$ and $q_b \rightarrow q_{k+1}$. Hence, $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_{k-1} \rightarrow_{\mathcal{A}'} q_b \rightarrow_{\mathcal{A}'} q_{k+1} \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$, i.e., $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$, which proves the result in this case. The remaining cases are similar.

  (b) if t is a constant then $t \rightarrow_{\mathcal{A}}{}^{*} q$ has the form $t \rightarrow_{\mathcal{A}} q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_n = q$. Regarding the first step, $t \rightarrow_{\mathcal{A}} q_1$ implies that there exists a transition $t \rightarrow q_1 \in \Delta$. If $q_1 \neq q_a$ then $t \rightarrow q_1 \in \Delta'$ and $t \rightarrow_{\mathcal{A}'} q_1$. For the suffix $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q_n$ we obtain like in case 1(a) that $q_1 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$, and the result follows. If $q_1 = q_a$ then $t \rightarrow q_b \in \Delta'$, and we obtain again like in case 1(a) that $q_b \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q_n$ and the result follows as well.

(2) if $t = f(t_1, \ldots, t_n)$, from $t \rightarrow_{\mathcal{A}}{}^{*} q$ and the definition of tree automata derivation, there exists a rule $f(q_1, \ldots, q_n) \rightarrow q_0 \in \Delta$ and a derivation using ϵ-transitions only: $q_0 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q$ such that $f(t_1, \ldots, t_n) \rightarrow_{\mathcal{A}}{}^{*} q_0 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q$. From $q_0 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}}^{*} q$ we obtain like in case 1(a) that $q_0 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q$. From the derivation $f(t_1, \ldots, t_n) \rightarrow_{\mathcal{A}}{}^{*} q_0$ we obtain that there exist states $q_1, \ldots, q_n$ such that $t_i \rightarrow_{\mathcal{A}}{}^{*} q_i$ for $i = 1, \ldots, n$, and by induction hypothesis, $t_i\{q_a \mapsto q_b\} \rightarrow_{\mathcal{A}'}^{*} q_i$ for $i = 1, \ldots, n$. By definition of renaming $f(q_1\{q_a \mapsto q_b\}, \ldots, q_n\{q_a \mapsto q_b\}) \rightarrow q_0\{q_a \mapsto q_b\} \in \Delta'$. Hence, $t\{q_a \mapsto q_b\} = f(t_1\{q_a \mapsto q_b\}, \ldots, t_n\{q_a \mapsto q_b\}) \rightarrow_{\mathcal{A}'}^{*} q_0 \overset{\epsilon}{\rightarrow}{}_{\mathcal{A}'}^{*} q$, which completes the proof.

□

Now we define the *simplification* operation, which merges states in a tree automaton according to an equation and a substitution.

Definition 13 (Simplification)**.** Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and E be a set of equations. The simplification operation, denoted by $\rightsquigarrow_E$, is defined as follows: $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ if there exist distinct states $q_1, q_2 \in \mathcal{Q}$, an equation $s = t \in E$, and a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$, such that $s\sigma \rightarrow_{\mathcal{A}}^{\not\epsilon *} q_1$, $t\sigma \rightarrow_{\mathcal{A}}^{\not\epsilon *} q_2$, and $\mathcal{A}' = A\{q_1 \mapsto q_2\}$. We identify the operation $\rightsquigarrow_E$ with a relation on tree automata in the obvious way. ⋄

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

Example 14. Let $\mathcal{A}$ be a tree automaton such that $\Delta = \{f(q_1, q_2) \to q_0, a \to q_1, a \to q_2, g(q_1) \to q_3\}$.

- If $E = \{g(x) = a\}$, we have $\sigma = \{x \mapsto q_1\}$ and

$$\begin{tikzcd}
g(q_1) \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & a \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_3 & q_2
\end{tikzcd}$$

  Hence, $\mathcal{A} \leadsto_E \mathcal{A}'$ where $\mathcal{A}' = \mathcal{A}\{q_3 \mapsto q_2\}$.
- If $E = \{f(x, x) = g(x)\}$ then there is no substitution σ such that $f(x, x)\sigma = f(q_1, q_2)$: the automaton is unchanged, and a is still recognized in two distinct states (q_1 and q_2).

The last example shows that simplification does not always fully reduce the automaton (here, one would expect the states q_1 and q_2 to be merged by any set of equations, but this is not the case). This limitation has no incidence on the results in this paper; in particular, we do not require the equations in the set E to be linear.

Definition 15. An automaton $\mathcal{A}$ is in normal form with respect to $\leadsto_E$ if for all automata $\mathcal{A}''$: $\mathcal{A}' \not\leadsto_E \mathcal{A}''$. We denote by $\leadsto_E^*$ the reflexive-transitive closure of the relation $\leadsto_E^*$. For tree automata $\mathcal{A}, \mathcal{A}'$, we write $\mathcal{A} \leadsto_E^! \mathcal{A}'$ when $\mathcal{A} \leadsto_E^* \mathcal{A}'$ and $\mathcal{A}'$ is in normal form. $\diamond$

Lemma 16. *For all tree automata $\mathcal{A}, \mathcal{A}'$, all set of equations E and state mappings α, such that $\mathcal{A} \leadsto_E^* \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\alpha$: $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{L}(\mathcal{A}\alpha, q\alpha)$.*

Proof. By induction on the length of the sequence $\leadsto^* E$. The base case is trivial, and the inductive step uses Lemma 12. $\square$

To build approximations we repeatedly apply the simplification operation until a normal form is obtained. The termination of the procedure is based on the following lemma.

Lemma 17. *The simplification relation $\leadsto_E$ is well founded.*

Proof. Each step of simplification $\mathcal{A} \leadsto_E \mathcal{A}'$ reduces the number of states by one . $\square$

Although it is not essential for our main algorithm, we can also prove that repeated simplifications lead to a *unique normal form up to isomorphism*, where two automata $\mathcal{A}, \mathcal{A}'$ are *isomorphic* if there exists a bijection α such that $\mathcal{A}' = \mathcal{A}\alpha$. For this, we prove the local confluence up to isomorphism of the $\leadsto_E$ relation. Together with termination, this implies confluence up to isomorphism. Details of the proof can be found in the report (Genet and Rusu, 2009).

Lemma 18. *The relation $\leadsto_E$ is locally confluent modulo isomorphism.*

4. $\mathcal{R}/E$-coherent Tree Automata

We define in this section the notion of $\mathcal{R}/E$-coherence of tree automata. The main result in this section is that simplification preserves $\mathcal{R}/E$ coherence. This result, together with a similar result for the *completion* operation proved in the next section, is crucial for proving the precision of our completion algorithm with equational approximations.

Definition 19 ($\mathcal{R}/E$-coherent automaton)**.** Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be 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 for all states $q \in \mathcal{Q}$, there exists a term $s \in \mathcal{T}(\mathcal{F})$ such that $s \to_{\mathcal{A}}^{\not\epsilon *} q$, and for all $t \in \mathcal{T}(\mathcal{F})$:

$$t \to_{\mathcal{A}}^{\not\epsilon *} q \implies s =_E t \text{ and}$$
$$t \to_{\mathcal{A}}{}^{*} q \implies s \to_{\mathcal{R}/E}^{*} t.$$

◇

In the following, any term s such that $s \to_{\mathcal{A}}^{\not\epsilon *} q$ is called a *representative* of q in $\mathcal{A}$. The first implication in Definition 19 says that $\mathcal{R}/E$-coherent automata have the property that *all states have at least one representative, and all representatives of a state are equal modulo E*. For automata having this property, we denote by $rep_{\mathcal{A}}(q)$ an arbitrary representative of a state q in the automaton $\mathcal{A}$. We now illustrate the notion of $\mathcal{R}/E$-coherence.

Example 20. Any automaton without ϵ-transitions, such that each term in its language is recognized in a different state, is $\mathcal{R}/E$-coherent, for any TRS $\mathcal{R}$ and equation system E. For example, the automaton whose states are $\mathcal{Q} = \mathcal{Q}_f = \{q_1, q_2\}$ and transitions are $a \to q_1, b \to q_2$, is so. This observation is important, because it implies that our precision result, which depend of $\mathcal{R}/E$-coherence preservation, holds for systems having finitely many initial states (encoded by an automaton reconizing a finite language). On the other hand, the same automaton as above, enriched with either (i) the additional transition $b \to q_1$, or (ii) an additional state q_3, is not $\mathcal{R}/E$-coherent (for $\mathcal{R}$ and E being the empty TRS and equation set, respectively): in the case (i) because a, b are both recognized in q_1 and are not equal *modulo* E, and in the case (ii) because q_3 does not recognize any term. The automaton (i) becomes $\mathcal{R}/E$-coherent with $E = \{a = b\}$. Finally, to illustrate the second implication in Definition 19 of $\mathcal{R}/E$-coherence, consider the automaton with states $\mathcal{Q} = \{q_1, q_2\}$, final states $\mathcal{Q}_f = \{q_1\}$, and transitions $a \to q_1, b \to q_2, q_2 \to q_1$. For any system of equations E, this automaton is not $\mathcal{R}/E$-coherent if $\mathcal{R} = \emptyset$. On the other hand, the automaton is $\mathcal{R}/E$-coherent when $\mathcal{R} = \{a \to b\}$, for any equation system E.

The next lemma is a simple consequence of Definition 19 and further illustrates it.

Lemma 21. *If an automaton $\mathcal{A}$ is $\mathcal{R}/E$-coherent then, for all states q of $\mathcal{A}$ and all representatives $rep_{\mathcal{A}}(q)$ of the state q, the inclusion $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}_E^*(rep_{\mathcal{A}}(q))$ holds.*

Lemma 21 will be used in a later section for proving our precision result. The rest of this section is mainly dedicated to showing that the simplification preserves $\mathcal{R}/E$-coherence. The next lemma says that, for $\mathcal{R}/E$-coherent automata, simplification renames states into states having the same representatives.

Lemma 22. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$, and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Then, for all representatives $rep_{\mathcal{A}}(q_a)$ of q_a and $rep_{\mathcal{A}}(q_b)$ of q_b in $\mathcal{A}$, $rep_{\mathcal{A}}(q_a) =_E rep_{\mathcal{A}}(q_b)$.*

Proof. By Definition 13, $\mathcal{A} \leadsto_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$ means that there exist an equation $s = t \in E$ and a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $s\sigma = t\sigma$, $s\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_a$, and $t\sigma \to_{\mathcal{A}}^{\not\epsilon *} q_b$. Let $q_1, \ldots, q_n$ be the states occuring in the term $s\sigma$, and $q'_1, \ldots, q'_m$ be the states occuring in the term $t\sigma$. We consider a substitution ρ that maps each state q_i and q'_i to a representative of it in $\mathcal{A}$. Then, $s\sigma\rho = t\sigma\rho$, $s\sigma\rho$ is a representative of q_a in $\mathcal{A}$, and $t\sigma\rho$ is a representative for q_b, in $\mathcal{A}$. The conclusion follows from the fact that all representatives of a state are equal *modulo* E in $\mathcal{R}/E$-coherent automata. □

The next lemma focuses on the representatives of states occurring in transitions.

Lemma 23. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Then,*

(1) for all transitions of the form $c \to q' \in \mathcal{A}'$, where c is a constant, there exists a transition $c \to q \in \mathcal{A}$, such that $rep_{\mathcal{A}}(q) =_E rep_{\mathcal{A}}(q')$;

(2) for all transitions of the form $f(q'_1, \ldots q'_n) \to q'_0 \in \mathcal{A}'$, there exists a transition $f(q_1, \ldots q_n) \to q_0 \in \mathcal{A}$ such that $rep_{\mathcal{A}}(q_i) =_E rep_{\mathcal{A}}(q'_i)$ for all $i = 0, \ldots, n$.

Proof. We prove the second statement; the proof of the first one is even simpler. By definition of renaming, the transition $f(q'_1, \ldots q'_n) \to q'_0$ of $\mathcal{A}'$ has been obtained from some transition $f(q_1, \ldots q_n) \to q_0$ of $\mathcal{A}$, by renaming all the states $q_i = q_a$ into $q'_i = q_b$. We partition the states q_i ($i = 0, \ldots, n$) of $\mathcal{A}$ into

- states q_i such that $q_i = q_a$: then, $q'_i = q_b$, and by Lemma 22, $rep_{\mathcal{A}}(q_a) =_E rep_{\mathcal{A}}(q_b)$, hence, $rep_{\mathcal{A}}(q_i) =_E rep_{\mathcal{A}}(q'_i)$;
- states q_i such that $q_i \neq q_a$. In those states, the renaming $\{q_a \mapsto q_b\}$ has no effect, hence, $q_i = q'_i$ and *a fortiori* $rep_{\mathcal{A}}(q_i) =_E rep_{\mathcal{A}}(q'_i)$.

$\square$

Proving that the simplification preserves $\mathcal{R}/E$-coherence amounts to proving that it preserves both implications in Definition 19. The first "half" of the proof deals with the first implication, and the second "half" of the proof deals with the second implication. Both "halves" of the proof use structural induction as well as the two following lemmas.

Lemma 24. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$, and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Let also $c \to q' \in \mathcal{A}'$ where c is a constant. Then, for all representatives $rep_{\mathcal{A}}(q')$: $c =_E rep_{\mathcal{A}}(q')$.*

Proof. By Lemma 23 (1st item), the corresponding transition $c \to q \in \mathcal{A}$ is such that $rep_{\mathcal{A}}(q) =_E rep_{\mathcal{A}}(q')$. Now, $c \to q \in \mathcal{A}$ implies that c is a representative of q in $\mathcal{A}$, and all representatives of a state in the $\mathcal{R}/E$-coherent automaton $\mathcal{A}$ are equal *modulo* E, hence, $c =_E rep_{\mathcal{A}}(q)$, and $c =_E rep_{\mathcal{A}}(q')$ follows by transitivity of $=_E$. $\square$

Lemma 25. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$, and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Let also $f(q'_1, \ldots q'_n) \to q'_0 \in \mathcal{A}'$ be a transition of the automaton $\mathcal{A}'$. Then, for all representatives $rep_{\mathcal{A}}(q'_0)$, ..., $rep_{\mathcal{A}}(q'_n)$: $f(rep_{\mathcal{A}}(q'_1), \ldots rep_{\mathcal{A}}(q'_n)) =_E rep_{\mathcal{A}}(q'_0)$.*

Proof. Note first that the representatives $rep_{\mathcal{A}}(q'_i)$ exist since $\mathcal{A}$ is $\mathcal{R}/E$-coherent and the states q'_i of $\mathcal{A}'(= \mathcal{A}\{q_a \mapsto q_b\})$ are also states of $\mathcal{A}$. By Lemma 23 (2nd item), the statement to prove in our lemma amounts to proving $f(rep_{\mathcal{A}}(q_1), \ldots rep_{\mathcal{A}}(q_n)) =_E rep_{\mathcal{A}}(q_0)$. By definition of representatives, $rep_{\mathcal{A}}(q_i) \to_{\mathcal{A}}^{\not\epsilon *} q_i$ for $i = 0, \ldots, n$, and since $f(q_1, \ldots q_n) \to q_0$ is a transition of $\mathcal{A}$, we obtain $f(rep_{\mathcal{A}}(q_1) \ldots rep_{\mathcal{A}}(q_n)) \to_{\mathcal{A}}^{\not\epsilon *} q_0$, meaning that the term $f(rep_{\mathcal{A}}(q_1), \ldots rep_{\mathcal{A}}(q_n))$ is a representative of q_0 in $\mathcal{A}$. The conclusion follows by the property that all representatives of state q_0 in the automaton $\mathcal{A}$ are equal *modulo* E. $\square$

The next lemma establishes the first "half" of the preservation of $\mathcal{R}/E$-coherence. We actually prove that the representatives of a state after simplification are equal (*modulo* the equations E) to the representatives of the corresponding state before simplification.

Lemma 26. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, and such that $\mathcal{A} \leadsto_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Then, for all states $q' \in \mathcal{Q}'$ and representatives $rep_{\mathcal{A}}(q')$ of q' in $\mathcal{A}$, $rep_{\mathcal{A}}(q') \to_{\mathcal{A}'}^{\not\epsilon *} q'$ holds, and for all terms $t \in \mathcal{T}(\mathcal{F})$, $t \to_{\mathcal{A}'}^{\not\epsilon *} q'$ implies $t =_E rep_{\mathcal{A}}(q')$.*

Proof. The first part of the statement: $rep_{\mathcal{A}}(q') \to^{\not\epsilon *}_{\mathcal{A}'} q'$ holds because, as any representative of q' in $\mathcal{A}$, $rep_{\mathcal{A}}(q') \to^{\not\epsilon *}_{\mathcal{A}} q'$, and Lemma 16 implies that every term recognized by the automaton $\mathcal{A}$ in a state $q' \in \mathcal{Q}'$ is recognized by the automaton $\mathcal{A}'$ in the same state.

For the second part of the statement we proceed by induction on the term t.

- if t is a constant, then $t \to^{\not\epsilon *}_{\mathcal{A}'} q'$ means that there exists a transition $t \to q' \in \mathcal{A}'$, and $t =_E rep_{\mathcal{A}}(q')$ follows by Lemma 24.
- if $t = f(t_1, \dots, t_n)$, then $t \to^{\not\epsilon *}_{\mathcal{A}} q'$ means there exists $f(q'_1, \dots, q'_n) \to q' \in \mathcal{A}'$ such that $t_i \to^{\not\epsilon *}_{\mathcal{A}'} q'_i$ for $i = 1, \dots n$. By induction hypothesis, $t_i =_E rep_{\mathcal{A}}(q'_i)$ for $i = 1, \dots n$. Hence, $t =_E f(rep_{\mathcal{A}}(q'_1), \dots, rep_{\mathcal{A}}(q'_n))$ by congruence of $=_E$. By Lemma 25, $f(rep_{\mathcal{A}}(q'_1), \dots, rep_{\mathcal{A}}(q'_n)) =_E rep_{\mathcal{A}}(q')$, and $t =_E rep_{\mathcal{A}}(q')$ follows by transitivity.
□

As a consequence of Lemma 26, all representatives of a state q' in $\mathcal{Q}'$ are equal *modulo* E, and we can legitimately use the notation $rep_{\mathcal{A}'}(q')$ to denote an arbitrarily chosen representative of q' in $\mathcal{A}'$. *Note also that Lemma 26 implies* $rep_{\mathcal{A}'}(q') =_E rep_{\mathcal{A}}(q')$.

We proceed to the second "half" of proving the preservation of $\mathcal{R}/E$-coherence. The next two lemmas deal with derivations by ϵ-transitions that occur in the definition of $\mathcal{R}/E$-coherence: first, by using only one transition, and then by using several transitions.

Lemma 27. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$, and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Let $q'_0 \to q'_1 \in \mathcal{A}'$ be an ϵ-transition of $\mathcal{A}'$. Then, $rep_{\mathcal{A}'}(q'_1) \to^*_{\mathcal{R}/E} rep_{\mathcal{A}'}(q'_0)$.*

Proof. The transition $q'_0 \to q'_1 \in \mathcal{A}'$ is obtained from some transition $q_0 \to q_1 \in \mathcal{A}$ by renaming its states. Using Lemma 22, $rep_{\mathcal{A}}(q'_0) =_E rep_{\mathcal{A}}(q_0)$ and $rep_{\mathcal{A}}(q'_1) =_E rep_{\mathcal{A}}(q_1)$. Using Lemma 26, $rep_{\mathcal{A}'}(q'_0) =_E rep_{\mathcal{A}}(q'_0)$ and $rep_{\mathcal{A}'}(q'_1) =_E rep_{\mathcal{A}}(q'_1)$, and by transitivity of $=_E$, $rep_{\mathcal{A}'}(q'_0) =_E rep_{\mathcal{A}}(q_0)$ and $rep_{\mathcal{A}'}(q'_1) =_E rep_{\mathcal{A}}(q_1)$. Next, by definition of representatives, $rep_{\mathcal{A}}(q_0) \to^{\not\epsilon *}_{\mathcal{A}} q_0$. Then, $rep_{\mathcal{A}}(q_0)$ is also recognized by $\mathcal{A}$ in q_1 using the "additional" ϵ-transition $q_0 \to q_1$, that is, $rep_{\mathcal{A}}(q_0) \to_{\mathcal{A}}{}^* q_1$. By $\mathcal{R}/E$-coherence of $\mathcal{A}$, $rep_{\mathcal{A}}(q_1) \to^*_{\mathcal{R}/E} rep_{\mathcal{A}}(q_0)$. The result follows by Definition 3 of $\mathcal{R}/E$ rewriting. □

Lemma 28. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, $\mathcal{A} \leadsto_E \mathcal{A}'$, and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. For all derivations of the form $q'_0 \xrightarrow{\epsilon}{}^*_{\mathcal{A}'} q'_n$, $rep_{\mathcal{A}'}(q'_n) \to^*_{\mathcal{R}/E} rep_{\mathcal{A}'}(q'_0)$.*

Proof. By induction on the derivation $q_0 \xrightarrow{\epsilon}{}^*_{\mathcal{A}'} q'_n$. The base case holds because, by Lemma 26, all representatives of a state in $\mathcal{Q}'$ are equal *modulo E*. For the inductive step, we decompose the derivation into $q'_0 \xrightarrow{\epsilon}_{\mathcal{A}'} q'_1 \xrightarrow{\epsilon}{}^*_{\mathcal{A}'} q'_n$, i.e., the first step is performed using an ϵ-transition $q'_0 \to q'_1 \in \mathcal{A}'$. By Lemma 27, $rep_{\mathcal{A}'}(q'_1) \to^*_{\mathcal{R}/E} rep_{\mathcal{A}'}(q'_0)$. By induction hypothesis, $rep_{\mathcal{A}'}(q'_n) \to^*_{\mathcal{R}/E} rep_{\mathcal{A}'}(q'_1)$. The result follows by transitivity of $\to^*_{\mathcal{R}/E}$. □

We are now ready to prove the second "half" in the preservation of $\mathcal{R}/E$-coherence.

Lemma 29. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent, and such that $\mathcal{A} \leadsto_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. Then, for all states $q' \in \mathcal{Q}'$ and representatives $rep_{\mathcal{A}'}(q')$ of q' in $\mathcal{A}'$, and for all terms $t \in \mathcal{T}(\mathcal{F})$, $t \to^*_{\mathcal{A}'} q'$ implies $rep_{\mathcal{A}'}(q') \to^*_{\mathcal{R}/E} t$.*

Proof. By induction on the term t.

- if t is a constant, the derivation $t \to^*_{\mathcal{A}'} q'$ can be decomposed into $t \to_{\mathcal{A}'} \tilde{q}' \xrightarrow{\epsilon}{}^*_{\mathcal{A}'} q'$, where $t \to \tilde{q}' \in \mathcal{A}'$. By Lemma 28, $rep_{\mathcal{A}'}(q') \to^*_{\mathcal{R}/E} rep_{\mathcal{A}'}(\tilde{q}')$. Using Lemma 26, $rep_{\mathcal{A}'}(\tilde{q}') =_E rep_{\mathcal{A}}(\tilde{q}')$. By Lemma 24, $rep_{\mathcal{A}}(\tilde{q}') =_E t$. The result $rep_{\mathcal{A}'}(q') \to^*_{\mathcal{R}/E} t$ follows by transitivity of $=_E$ and definition of $\to^*_{\mathcal{R}/E}$.

- if $t = f(t_1, \ldots, t_n)$, then $t \rightarrow^*_{\mathcal{A}'} q'$ means that there exists $f(q'_1, \ldots, q'_n) \rightarrow q' \in \mathcal{A}'$ such that $t_i \rightarrow^*_{\mathcal{A}'} q'_i$ for $i = 1, \ldots n$. By Lemma 26 and induction hypothesis, $rep_{\mathcal{A}}(q'_i) =_E rep_{\mathcal{A}'}(q'_i) \rightarrow^*_{\mathcal{R}/E} t_i$ for $i = 1, \ldots n$, then, $f(rep_{\mathcal{A}}(q'_1), \ldots, rep_{\mathcal{A}}(q'_n)) \rightarrow^*_{\mathcal{R}/E} f(t_1, \ldots, t_n) = t$. By Lemmas 25 and 26, $f(rep_{\mathcal{A}}(q'_1), \ldots, rep_{\mathcal{A}}(q'_n)) =_E rep_{\mathcal{A}}(q') = rep_{\mathcal{A}'}(q')$. The result follows by symmetry/transitivity of $=_E$ and definition of $\rightarrow^*_{\mathcal{R}/E}$. □

As a consequence of the definition of simplification and of Lemmas 26 and 29 we obtain

Theorem 30. *Let $\mathcal{A}, \mathcal{A}'$ be tree automata, $\mathcal{R}$ a TRS, E a set of equations and q_a, q_b states of $\mathcal{A}$ such that $\mathcal{A} \leadsto_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. If $\mathcal{A}$ is $\mathcal{R}/E$-coherent then $\mathcal{A}'$ is $\mathcal{R}/E$-coherent as well. Moreover, all states q' of $\mathcal{A}'$ are also states of $\mathcal{A}$, and all representatives of q' in $\mathcal{A}'$ are also representatives for q' in $\mathcal{A}$.*

As a corollary to Theorem 30 we obtain that the simplified automaton $\mathcal{A}'$, obtained by applying simplification steps to an automaton $\mathcal{A}$, preserves the representatives of states.

Corollary 31. *Let $\mathcal{A}, \mathcal{A}'$ be automata such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent and such that $\mathcal{A} \leadsto^*_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{A}\alpha$, where α is the composition of the state renamings occurring in $\mathcal{A} \leadsto^*_E \mathcal{A}'$. Then, for all states q of $\mathcal{A}$, $q\alpha$ is a state of $\mathcal{A}'$, and $rep_{\mathcal{A}'}(q\alpha) =_E rep_{\mathcal{A}}(q)$.*

Proof. It is enough to prove the result for one step of simplification $\mathcal{A} \leadsto_E \mathcal{A}'$ - a simple inductive argument does the rest. Then, $\mathcal{A}' = \mathcal{A}\{q_a \mapsto q_b\}$. From Theorem 30 we know that $\mathcal{A}'$ is $\mathcal{R}/E$-coherent. If $q \neq q_a$ then $q\{q_a \mapsto q_b\} = q$, which is a state of $\mathcal{A}'$ since it was not renamed. Again by Theorem 30, we know that $rep_{\mathcal{A}'}(q)$ is a representative of q in $\mathcal{A}$, which concludes the proof of the case $q \neq q_a$. If $q = q_a$ then $q\{q_a \mapsto q_b\} = q_b$, which is a state of $\mathcal{A}'$ by Definition 10 of renaming. By Lemma 26, $rep_{\mathcal{A}'}(q_b) =_E rep_{\mathcal{A}}(q_b)$ and by Lemma 22, $rep_{\mathcal{A}}(q_b) =_E rep_{\mathcal{A}}(q_a)$, and the transitivity of $=_E$ concludes the proof. □

5. Tree Automata Completion Algorithm

In this section we define a tree automaton completion algorithm, which is a variant of the algorithm defined in (Genet, 1998; Feuillade et al., 2004). Like for the simplification operation, we prove that completion preserves $\mathcal{R}/E$-coherence and the representatives of states. These properties, together with the similar properties for the simplification operation from the previous section, are used for proving the precision result for our main algorithm (that combines simplification and completion) in the next section.

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, the tree automata completion algorithm, proposed in (Genet, 1998; Feuillade et al., 2004), computes a sequence $\mathcal{A}^0_{\mathcal{R}}.\mathcal{A}^1_{\mathcal{R}} \ldots \mathcal{A}^k_{\mathcal{R}}, \ldots$ of automata such that if $s \in \mathcal{L}(\mathcal{A}^i_{\mathcal{R}})$ and $s \rightarrow_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}^{i+1}_{\mathcal{R}})$. If a fixpoint, i.e., an automaton $\mathcal{A}^k_{\mathcal{R}}$ such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}^k_{\mathcal{R}})) = \mathcal{L}(\mathcal{A}^k_{\mathcal{R}})$, is found, then $\mathcal{L}(\mathcal{A}^k_{\mathcal{R}}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}^0_{\mathcal{R}}))$, and $\mathcal{L}(\mathcal{A}^k_{\mathcal{R}}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ if $\mathcal{R}$ is in one of the classes defined in (Feuillade et al., 2004).

To build $\mathcal{A}^{i+1}_{\mathcal{R}}$ from $\mathcal{A}^i_{\mathcal{R}}$, a *completion step*, which consists of finding *critical pairs* between $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{A}^i_{\mathcal{R}}}$, is performed. For a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ and a rule $l \rightarrow r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \rightarrow^*_{\mathcal{A}^i_{\mathcal{R}}} q$, $l\sigma \rightarrow_{\mathcal{R}} r\sigma$, and $r\sigma \not\rightarrow^*_{\mathcal{A}^i_{\mathcal{R}}} q$. Since $\mathcal{R}$, $\mathcal{A}^i_{\mathcal{R}}$ and the set $\mathcal{Q}_i$ of states of $\mathcal{A}^i_{\mathcal{R}}$ are finite, there is only a finite number of critical pairs. For every critical pair detected between $\mathcal{R}$ and $\mathcal{A}^i_{\mathcal{R}}$, the tree automaton $\mathcal{A}^{i+1}_{\mathcal{R}}$ is built by adding new transitions. In (Genet, 1998), new transitions are added such as to enable the ϵ-free derivation $r\sigma \rightarrow^{\not\epsilon *}_{\mathcal{A}^{i+1}_{\mathcal{R}}} q$ (cf. Figure 1).

In the new version of the algorithm, we add a new state q', a ϵ-transition $q' \rightarrow q$, and transitions that enable the ϵ-free derivation $r\sigma \rightarrow^{\not\epsilon *}_{\mathcal{A}^{i+1}_{\mathcal{R}}} q'$ instead (cf. Figure 2).

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "*", "\mathcal{A}^i_{\mathcal{R}}, \not\epsilon"'] & r\sigma \arrow[dl, bend left, "*"', "\mathcal{A}^{i+1}_{\mathcal{R}}, \not\epsilon"] \\
q &
\end{tikzcd}$$

Figure 1. Old completion algorithm: new terms are recognized (ϵ-free) in old states.

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "*", "\mathcal{A}^i_{\mathcal{R}}, \not\epsilon"'] & r\sigma \arrow[d, "*"', "\mathcal{A}^{i+1}_{\mathcal{R}}, \not\epsilon"] \\
q & q' \arrow[l, "\mathcal{A}^{i+1}_{\mathcal{R}}"]
\end{tikzcd}$$

Figure 2. New completion algorithm: new terms are recognized (ϵ-free) in new states.

The new completion operation is required because our precision result requires us to prove that completion preserves $\mathcal{R}/E$-coherence. But the automata generated by the completion algorithm of (Genet, 1998) are not, in general, $\mathcal{R}/E$-coherent, because they recognize ϵ-free (i.e., without taking ϵ-transitions), in the same state q, terms $l\sigma$ and $r\sigma$ that are not *a priori* equal *modulo E* (cf. Figure 1). However, $\mathcal{R}/E$-coherence (Def. 19) imposes that terms recognized ϵ-free in a state be equal *modulo E*. The new algorithm solves this problem by ϵ-free recognizing new terms $r\sigma$ in new states q' (cf. Figure 2).

We proceed with the formal definition of the new completion algorithm and with proving that it preserves the $\mathcal{R}/E$-coherence property and the representatives of states. The *normalization* operation generates a set of normalized transitions for the completed automaton in Figure 2 to enable the derivation $r\sigma \rightarrow^{\not\epsilon *}_{\mathcal{A}^{i+1}_{\mathcal{R}}} q'$. Even though q' is a new state for $\mathcal{A}^i_{\mathcal{R}}$, our normalization operation re-uses old states of $\mathcal{A}^i_{\mathcal{R}}$ to recognize the subterms of $r\sigma$ whenever possible, in order to reduce the resulting automaton as much as possible.

Definition 32 (Normalization)**.** Let $\mathcal{Q}$ be a countably infinite set of states. Let Δ be a finite set of transitions whose states are in $\mathcal{Q}$. A *new state for* Δ is a state $q' \in \mathcal{Q}$ not occurring in any transition in Δ [1]. The *normalization operation* takes a transition $t \rightarrow q$ such that $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$, and q is new for Δ, and inductively generates a set of normalized transitions, by applying the following rules:

(1) $Norm_\Delta(t \rightarrow q) = \{t \rightarrow q\}$ if $t \rightarrow q$ is a normalized transition;

(2) $Norm_\Delta(f(t_1, \ldots, t_i, \ldots, t_n) \rightarrow q) = Norm_\Delta(f(t_1, \ldots, q_i, \ldots, t_n) \rightarrow q)$ if a state $q_i \in \mathcal{Q}$ can be chosen such that $t_i \rightarrow^{\not\epsilon *}_{\Delta} q_i$;

(3) $Norm_\Delta(f(t_1, \ldots, t_i, \ldots, t_n) \rightarrow q) = Norm_{\Delta \cup \{t_i \rightarrow q'_i\}}(f(t_1, \ldots, q'_i, \ldots, t_n) \rightarrow q) \cup Norm_{\Delta \cup \{t_i \rightarrow q'_i\}}(t_i \rightarrow q'_i)$ where q'_i is new for Δ, if for all $q_i \in \mathcal{Q}$, $t_i \not\rightarrow^{\not\epsilon *}_{\Delta} q_i$. $\diamond$

The normalization operation terminates since both recursive rules (2) and (3) decrease the number of symbols in $\mathcal{F}$ in their right-hand sides. A simple proof by induction establishes that for all terms $t \in (\mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q})$ and for all states $q \in \mathcal{Q}$ new for Δ, $t \rightarrow^{\not\epsilon *}_{\Delta \cup Norm_\Delta(t \rightarrow q)} q$ holds. Moreover, since q is new for Δ, adding $Norm_\Delta(t \rightarrow q)$ to Δ does not change the languages recognized by the states occurring in Δ. This will be useful in the proof of Theorem 36. These points are illustrated by the following example.

Example 33. Let $\Delta = \{b \rightarrow q_0\}$. We illustrate the above definition on the normalization of the transition $f(g(a), b, g(a)) \rightarrow q$. The states changed by each step of normalization are boldfaced. Using the third rule of the definition, $Norm_\Delta(f(g(a), b, g(a)) \rightarrow q)$ equals

$$Norm_{\Delta \cup \{g(a) \rightarrow \mathbf{q_1}\}}(f(\mathbf{q_1}, b, g(a)) \rightarrow q) \;\cup\; Norm_{\Delta \cup \{g(a) \rightarrow \mathbf{q_1}\}}(g(a) \rightarrow \mathbf{q_1})$$

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

Then, by using the second rule of the definition, and we obtain:

$Norm_{\Delta\cup\{g(a)\to q_1\}}(f(q_1,\mathbf{q_0},g(a))\to q)\ \cup\ Norm_{\Delta\cup\{g(a)\to q_1\}}(g(a)\to q_1)$

Using again the second rule of the definition:

$Norm_{\Delta\cup\{g(a)\to q_1\}}(f(q_1,q_0,\mathbf{q_1})\to q)\ \cup\ Norm_{\Delta\cup\{g(a)\to q_1\}}(g(a)\to q_1)$

Using the first rule of the definition, we can simplify it into:

$\{f(q_1,q_0,q_1)\to q\}\ \cup\ Norm_{\Delta\cup\{g(a)\to q_1\}}(g(a)\to q_1)$

Using the third rule, we get:

$\{f(q_1,q_0,q_1)\to q\}\ \cup\ Norm_{\Delta\cup\{g(a)\to q_1,a\to\mathbf{q_2}\}}(g(\mathbf{q_2})\to q_1)$
$\cup\ Norm_{\Delta\cup\{g(a)\to q_1,a\to\mathbf{q_2}\}}(a\to\mathbf{q_2})$

Which simplifies using the first rule into:

$\{f(q_1,q_0,q_1)\to q, g(q_2)\to q_1\}\ \cup\ Norm_{\Delta\cup\{g(a)\to q_1,a\to q_2\}}(a\to q_2)$

And finally, using the first rule again, we obtain:

$\{f(q_1,q_0,q_1)\to q, g(q_2)\to q_1, a\to q_2\}$.

The normalization operation is used in the completion operation. We first define the completion of an automaton for one critical pair and illustrate it on an example.

Definition 34 (Automaton completion for one critical pair)**.** Consider a tree automaton $\mathcal{A}=\langle\mathcal{F},\mathcal{Q},\mathcal{Q}_f,\Delta\rangle$, a left-linear TRS $\mathcal{R}$, and a critical pair $l\sigma\to^*_{\mathcal{A}} q$, $l\sigma\to^*_{\mathcal{R}} r\sigma$ having the property $r\sigma\not\to^*_{\mathcal{A}} q$, where $l\to r\in\mathcal{R}$, $q\in\mathcal{Q}$, and $\sigma\in\Sigma(\mathcal{Q},\mathcal{X})$. The completion of $\mathcal{A}$ for the given critical pair is the automaton $\mathcal{A}'=\langle\mathcal{F},\mathcal{Q}',\mathcal{Q}_f,\Delta'\rangle$, where

$$\Delta'=\begin{cases}\Delta\cup\{q'\to q\} & \text{if } q'\in\mathcal{Q} \text{ can be chosen such that } r\sigma\to^{\not\epsilon *}_{\Delta} q'\\ \Delta\cup Norm_{\Delta}(r\sigma\to q')\cup\{q'\to q\} & \text{otherwise, where } q' \text{ is a new state for } \Delta\end{cases}$$

and $\mathcal{Q}'$ is the union of $\mathcal{Q}$ with the set of states occurring in Δ'. ◇

Note the distinction between the case where $r\sigma$ is already recognized into a state q' of $\mathcal{A}$ and the more general case. Here is an example illustrating this definition.

Example 35. Let $\mathcal{A}$ be a tree automaton with $\Delta=\{f(q_1)\to q_0, a\to q_1, g(q_1)\to q_2\}$.

- If $\mathcal{R}=\{f(x)\to x\}$ then there is a critical pair $f(x)\sigma\to_{\mathcal{A}}{}^* q_0$ and $f(x)\sigma\to_{\mathcal{R}} x\sigma$ with $\sigma=\{x\mapsto q_1\}$. Since $q_1\to^{\not\epsilon *}_{\mathcal{A}} q_1$, by the first case of our definition, we obtain $\Delta'=\Delta\cup\{q_1\to q_0\}$;
- If $\mathcal{R}=\{f(a)\to g(a)\}$ then there is a critical pair $f(a)\sigma\to_{\mathcal{A}}{}^* q_0$ and $f(a)\sigma\to_{\mathcal{R}} g(a)\sigma$ with $\sigma=\emptyset$. Since $g(a)\to^{\not\epsilon *}_{\mathcal{A}} q_2$, by the first case of our definition, we obtain $\Delta'=\Delta\cup\{q_2\to q_0\}$;
- If $\mathcal{R}=\{f(x)\to f(g(x))\}$ then the critical pair is $f(x)\sigma\to_{\mathcal{A}}{}^* q_0$ and $f(x)\sigma\to_{\mathcal{R}} f(g(x))\sigma$ with $\sigma=\{x\mapsto q_1\}$. Since there is no state $q\in\{q_0,q_1,q_2\}$ such that $f(g(x))\sigma\to^{\not\epsilon *}_{\mathcal{A}} q$, we have to use the second case of our definition and obtain $\Delta'=\Delta\cup Norm_{\Delta}(f(g(q_1))\to q')\cup\{q'\to q_0\}$. After simplification of $Norm_{\Delta}$ we obtain $\Delta'=\Delta\cup\{f(q_2)\to q', q'\to q_0\}$.

The completion of an automaton $\mathcal{A}$ by a TRS $\mathcal{R}$ consists in repeating, for all the critical pairs between $\mathcal{A}$ and $\mathcal{R}$, the one-pair completion operation defined above. For an automaton $\mathcal{A}$ and a TRS $\mathcal{R}$. The completion of $\mathcal{A}$ by $\mathcal{R}$ is hereafter denoted by $\mathcal{C}_{\mathcal{R}}(\mathcal{A})$.

The following theorem says that the completion operation preserves $\mathcal{R}/E$-coherence and the representatives of states. In its proof, we sometimes say that *a term t is recognized ϵ-free in a state q of an automaton $\mathcal{A}$* if $t\to^{\not\epsilon *}_{\mathcal{A}'} q$, and we call the *$\epsilon$-free language of q in $\mathcal{A}$* the set of terms $t\in\mathcal{T}(\mathcal{F})$ that are recognized ϵ-free by q in $\mathcal{A}$.

Theorem 36. *For all automata $\mathcal{A}$, TRS $\mathcal{R}$, and equation system E, such that $\mathcal{A}$ is $\mathcal{R}/E$-coherent and $\mathcal{R}$ is left-linear, $\mathcal{C}_{\mathcal{R}}(\mathcal{A})$ is $\mathcal{R}/E$-coherent. Moreover, each state $\tilde{q}$ of $\mathcal{A}$, $\tilde{q}$ is also a state of $\mathcal{A}' = \mathcal{C}_{\mathcal{R}}(\mathcal{A})$ and for all terms $s \in \mathcal{T}(\mathcal{F})$, $s \rightarrow^{\not\epsilon *}_{\mathcal{A}'} \tilde{q}$ implies $s \rightarrow^{\not\epsilon *}_{\mathcal{A}} \tilde{q}$.*

Proof. We sketch the proof for the completion of one critical pair between $\mathcal{A}$ and $\mathcal{R}$ - a simple induction does the generalization. Let then our critical pair be defined by $l\sigma \rightarrow^*_{\mathcal{A}} q$ and $l\sigma \rightarrow^*_{\mathcal{R}} r\sigma$. Moreover, we focus on the more interesting case, where *the right-hand side $r\sigma$ of the critical pair is not recognized by any state of $\mathcal{A}$* - hence, $r\sigma$ is not a state, and $r\sigma \not\rightarrow^{\not\epsilon *}_{\Delta} \tilde{q}$. The case where $r\sigma$ is already recognized by a state of $\mathcal{A}$ is simpler.

Let then $\mathcal{A}'$ be the completed automaton according to the second case of Definition 34. The statement *each state $\tilde{q}$ of $\mathcal{A}$ is also a state of $\mathcal{A}' = \mathcal{C}_{\mathcal{R}}(\mathcal{A})$* of our theorem is a direct consequence of Definition 34. The next statement *for all terms $s \in \mathcal{T}(\mathcal{F})$, $s \rightarrow^{\not\epsilon *}_{\mathcal{A}'} \tilde{q}$ implies $s \rightarrow^{\not\epsilon *}_{\mathcal{A}} \tilde{q}$*, follows from the observation that *the only state of $\mathcal{A}$, whose language may change in $\mathcal{A}'$ by completion, is q, due to the* ϵ-transition $q' \rightarrow q$. This is because Definition 32 ensures that the set of transitions $Norm_{\Delta}(r\sigma \rightarrow q')$ does not modify the ϵ-free language of the states of $\mathcal{A}$. Since any term recognized in q by $\mathcal{A}'$, but not by $\mathcal{A}$, must "take" the ϵ-transition $q' \rightarrow q$, the *ϵ-free* languages of q in $\mathcal{A}'$ and in $\mathcal{A}$ are the same. Hence, for all states $\tilde{q}$ of $\mathcal{A}$, the ϵ-free languages of $\tilde{q}$ in $\mathcal{A}$ and in $\mathcal{A}'$ are the same.

Another simple observation is that we only need to prove $\mathcal{R}/E$-coherence for the subset of states $\tilde{q} \in \{q\} \cup (\mathcal{Q}' \setminus \mathcal{Q})$ of $\mathcal{A}'$. Indeed, as noted above, the languages and ϵ-free languages of all the other states of $\mathcal{A}'$ (i.e., $\mathcal{Q} \setminus \{q\}$) are left unchanged by completion, and the $\mathcal{R}/E$ coherence of $\mathcal{A}'$ in those states follows from the $\mathcal{R}/E$ coherence of $\mathcal{A}$ in the same states. Hence, we we only need to prove $\mathcal{R}/E$-coherence for the states $\tilde{q} \in \{q\} \cup (\mathcal{Q}' \setminus \mathcal{Q})$. By definition of completion, $\tilde{q}$ is either q, or a new state introduced by completion.

(A) We show the first requirement of $\mathcal{R}/E$-coherence: there exists $s \in \mathcal{T}(\mathcal{F})$ such that $s \rightarrow^{\not\epsilon *}_{\mathcal{A}'} \tilde{q}$. The case $\tilde{q} = q$ is trivial because, as noted above, the terms recognized ϵ-free by $\mathcal{A}'$ in q are exactly those recognized ϵ-free by $\mathcal{A}$ in q - and there is at least such a term by $\mathcal{R}/E$-coherence of $\mathcal{A}$. For the case $\tilde{q} \neq q$: let us first consider the case $\tilde{q} = q'$. By hypothesis, $r\sigma \rightarrow^{\not\epsilon *}_{\mathcal{A}'} q'$, and $r\sigma$ is a term in $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. Let $r = C^r[x_1, \ldots, x_n]$, where C^r is a context [2] and $x_1, \ldots, x_n$ are all the variables occurring in r. Then, σ maps each of the variables x_i to some state $q_i \in \mathcal{Q}$, i.e., $r\sigma = C^r[q_1, \ldots, q_n]$. If $rep_{\mathcal{A}}(q_i)$ is a representative for q_i ($i = 1, \ldots, n$) in $\mathcal{A}$ then $rep_{\mathcal{A}}(q_i) \rightarrow^{\not\epsilon *}_{\mathcal{A}} q_i$, and, since $r\sigma = C^r[q_1, \ldots, q_n] \rightarrow^{\not\epsilon *}_{\mathcal{A}'} q'$, we obtain $C^r[rep_{\mathcal{A}}(q_1), \ldots, rep_{\mathcal{A}}(q_n)] \rightarrow^{\not\epsilon *}_{\mathcal{A}'} q'$ by definition of derivation in tree automata. This concludes the case $\tilde{q} = q'$. If $\tilde{q} \neq q'$ is another state generated by completion, then a *subterm* $\tilde{t}$ of $r\sigma$ is recognized ϵ-free in $\tilde{q}$. We prove this case by applying the same reasoning as for the case $\tilde{q} = q'$ to the subterm $\tilde{t}$ of $r\sigma$, rather than to the term $r\sigma$ itself.

(B) We prove the second requirement of $\mathcal{R}/E$-coherence: for all $t \in \mathcal{T}(\mathcal{F})$, if $t \rightarrow^{\not\epsilon *}_{\mathcal{A}'} \tilde{q}$, then $s =_E t$, where $s \in \mathcal{T}(\mathcal{F})$ is the term satisfying $s \rightarrow^{\not\epsilon *}_{\mathcal{A}'} \tilde{q}$ as established in (A) above. We settle the case $\tilde{q} = q$ by noting again that the terms recognized *ϵ-free* by $\mathcal{A}'$ in q are exactly those recognized ϵ-free by $\mathcal{A}$ in q, and all those terms are equal *modulo E* by the $\mathcal{R}/E$-coherence of $\mathcal{A}$. We continue with the case $\tilde{q} = q'$. We write again $r = C^r[x_1, \ldots, x_n]$ and $r\sigma = C^r[q_1, \ldots, q_n]$ with $q_1, \ldots, q_n \in \mathcal{Q}$. The normalization operation ensures that all terms $t' \in \mathcal{T}(\mathcal{F})$ such that $t' \rightarrow^{\not\epsilon *}_{\mathcal{A}'} q'$ have the form $C^r[t'_1, \ldots, t'_n]$, for terms $t'_1, \ldots, t'_n \in \mathcal{T}(\mathcal{F})$ such that $t'_i \rightarrow^{\not\epsilon *}_{\mathcal{A}} q_i$ for $i = 1, \ldots, n$. In particular, for the terms s and t in our hypothesis, $s = C^r[s_1, \ldots, s_n]$ for terms $s_i \rightarrow^{\not\epsilon *}_{\mathcal{A}} q_i$ for $i = 1, \ldots, n$. and $t = C^r[t_1, \ldots, t_n]$ for terms $t_i \rightarrow^{\not\epsilon *}_{\mathcal{A}} q_i$ for $i = 1, \ldots, n$. By $\mathcal{R}/E$-coherence of $\mathcal{A}$, $t_i =_E s_i$ for $i = 1, \ldots, n$, and then $s =_E t$ follows by congruence. This concludes the case $\tilde{q} = q'$. The last case, when $\tilde{q} \neq q'$ is another state generated by completion, is settled by analogy with the corresponding case in (A): there exists a subterm $\tilde{t}$ of $r\sigma$ that is recognized ϵ-free in $\tilde{q}$, and we apply exactly the same reasoning as for the case $\tilde{q} = q'$ to the subterm $\tilde{t}$.

[2] Note that the context $C^r[\ldots]$ exists based on use our assumption that r is not a variable.

(C) we prove the third requirement of $\mathcal{R}/E$-coherence: for all $t \in \mathcal{T}(\mathcal{F})$, if $t \rightarrow^*_{\mathcal{A}'} \tilde{q}$, then $s \rightarrow^*_{\mathcal{R}/E} t$. Unlike the proofs for (A) and (B), the case $\tilde{q} = q$ is nontrivial here.

Consider then the set of terms that are recognized in q by $\mathcal{A}'$, i.e., $\mathcal{L}(\mathcal{A}', q)$. We partition this set into $\mathcal{L}(\mathcal{A}, q)$ and $\mathcal{L}(\mathcal{A}', q) \setminus \mathcal{L}(\mathcal{A}, q)$. For all terms $t \in \mathcal{L}(\mathcal{A}, q)$, our conclusion $s \rightarrow^*_{\mathcal{R}/E} t$ holds by $\mathcal{R}/E$-coherence of $\mathcal{A}$. To complete the proof, we only need to establish $s \rightarrow^*_{\mathcal{R}/E} t$ for all terms $t \in \mathcal{L}(\mathcal{A}', q) \setminus \mathcal{L}(\mathcal{A}, q)$. By construction of $\mathcal{A}'$, the derivation $t \rightarrow^*_{\mathcal{A}'} q$ in our hypothesis can be decomposed into $t \rightarrow^*_{\mathcal{A}'} q' \rightarrow_{\mathcal{A}'} q$, i.e., the last step takes the ϵ-transition $q' \rightarrow q$, and the prefix $t \rightarrow^*_{\mathcal{A}'} q'$ means that $t \in \mathcal{L}(\mathcal{A}', q')$. Let again $r = C^r[x_1, \ldots, x_n]$ and $r\sigma = C^r[q_1, \ldots, q_n]$ with $q_1, \ldots, q_n \in \mathcal{Q}$. The normalization operation ensures that the term $t \in \mathcal{L}(\mathcal{A}', q')$ has the form $t = C^r[t_1, \ldots, t_n]$, where $t_i \in \mathcal{L}(\mathcal{A}, q_i)$ for $i = 1 \ldots, n$. On the other hand, the left-hand side l of the rule $l \rightarrow r$ in our critical pair has the form $l = C^l[x_1, \ldots, x_m]$ with $m \geq n$, since $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$ [3] , and then $l\sigma = C^l[q_1, \ldots q_n, q_{n+1}, \ldots q_m]$. Consider then a substitution μ such that $\mu(q_i) = t_i$, where the terms $t_i \in \mathcal{L}(\mathcal{A}, q_i)$ for $i = 1, \ldots n$ are those occurring in $t = C^r[t_1, \ldots, t_n]$, and $\mu(x_j)$ are arbitrary terms in $\mathcal{L}(q_j)$ for $j = n+1, \ldots, m$. Then, $r\sigma\mu = C^r[t_1, \ldots, t_n] = t$. By definition of term recognition, from $l\sigma \rightarrow^*_{\mathcal{A}} q$ in our critical pair, we get $l\sigma\mu \rightarrow^*_{\mathcal{A}} q$, which, by $\mathcal{R}/E$-coherence of $\mathcal{A}$, gives $s \rightarrow^*_{\mathcal{R}/E} l\sigma\mu$. And from $l\sigma \rightarrow_R l\sigma$ we obtain $l\sigma\mu \rightarrow_{\mathcal{R}} r\sigma\mu = C^r[t_1, \ldots, t_n] = t$. The case $\tilde{q} = q$ is concluded by the transitivity of the $\rightarrow^*_{\mathcal{R}/E}$ relation. Also, note that the case $\tilde{q} = q$ has actually covered the case $\tilde{q} = q'$, because we have decomposed the derivation $t \rightarrow^*_{\mathcal{A}'} q$ into $t \rightarrow^*_{\mathcal{A}'} q' \rightarrow_{\mathcal{A}'} q$. Finally, the case where $\tilde{q} \neq q'$ is another state generated by completion, is settled by analogy with the case $\tilde{q} = q'$, like in the previous parts (A) and (B) of this proof. □

Theorem 36 implies the following corollary - note the analogy with Corollary 31:

Corollary 37. *Let $\mathcal{R}$ be a left-linear TRS, E be a set of equations, and $\mathcal{A}$ be a $\mathcal{R}/E$-coherent automaton, Then, for each state q of $\mathcal{A}$, q is also a state of $\mathcal{C}_{\mathcal{R}}(\mathcal{A})$, and $rep_{\mathcal{C}_{\mathcal{R}}(\mathcal{A})}(q) =_E rep_{\mathcal{A}}(q)$.*

The last technical lemma in this section says that the completion of $\mathcal{A}$ by a left-linear TRS $\mathcal{R}$ includes all the terms reachable in one step from the language $\mathcal{L}(\mathcal{A}, q)$.

Lemma 38. *If $\mathcal{R}$ is left-linear, then $\mathcal{R}(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{L}(\mathcal{C}_{\mathcal{R}}(\mathcal{A}), q)$.*

Proof. (sketch) By induction on the number of rules in $\mathcal{R}$. The inductive step of this first induction requires a second induction, on the number of critical pairs between $\mathcal{A}$ and $\mathcal{R}$. The core of the proof lies within the inductive step of the second induction. It amounts to proving our lemma for the TRS R consisting of exactly one rule, and for exactly one critical pair between $\mathcal{A}$ and $\mathcal{R}$ - the induction hypotheses deal with the rest.

Let then $R = \{l \rightarrow r\}$ and our critical pair be defined by $l\sigma \rightarrow_{\mathcal{A}}{}^* q$ and $l\sigma \rightarrow_{\mathcal{R}} r\sigma$. On the one hand, $l = C^l[x_1, \ldots, x_n]$ where C^l is a context in which the variables $x_1, \ldots x_n$ occur exactly once, and $l\sigma$ has the form $C^l[q_1, \ldots, q_n]$, where $q_i = x_i\sigma$ for $i = 1, \ldots, n$. Hence, $\mathcal{L}(\mathcal{A}, q) = \{l\sigma\mu \mid \mu \in \Sigma(\mathcal{T}(\mathcal{F}), \mathcal{Q}),\ q_i\mu \in \mathcal{L}(\mathcal{A}, q_i)\ \textit{for}\ i = 1, \ldots, n\}$. On the other hand, by definition, $\mathcal{R}(\mathcal{L}(\mathcal{A}, q))$ is obtained by applying the rule $l \rightarrow r$ to $\mathcal{L}(\mathcal{A}, q)$, and, since $\mathcal{R}$ is left-linear, this amounts to applying the instance $l\sigma \rightarrow r\sigma$ of our rule, i.e., $\mathcal{R}(\mathcal{L}(\mathcal{A}, q)) = \{r\sigma\mu \mid \mu \in \Sigma(\mathcal{T}(\mathcal{F}), \mathcal{Q}),\ q_i\mu \in \mathcal{L}(\mathcal{A}, q_i)\ \textit{for}\ i = 1, \ldots, n\}$. But, by construction, the completed automaton $\mathcal{C}_{\mathcal{R}}(\mathcal{A})$ recognizes the terms $r\sigma$ in q, and then the terms $r\sigma\mu$ in $\mathcal{R}(\mathcal{L}(\mathcal{A}, q))$ are also recognized by $\mathcal{C}_{\mathcal{R}}(\mathcal{A})$ in q, which proves the result. □

[3] This context exists because of the general assumption that for rewrite rules $l \rightarrow r$, l is is not a variable.

6. Combining Completion and Simplification

We now define the full completion algorithm, which combines simplification steps defined in Section 3 with completion steps defined in Section 5. Then, using the results established for simplification and for completion, together with a few more simple lemmas, we prove that the full completion algorithm, when it terminates on an input automaton $\mathcal{A}$, produces a tree automaton $\mathcal{A}^*_{\mathcal{R},E}$ that recognizes an over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ and an under-approximation of $\mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$.

Definition 39 (Completion with Simplification)**.** Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ a TRS and E a set of equations. We define the sequence of automata $(\mathcal{A}^n_{\mathcal{R},E})_{n\geq 0}$ as follows
- $\mathcal{A}^0_{\mathcal{R},E} = \mathcal{A}$,
- for all $n \in \mathbb{N}$, $\mathcal{A}^{n+1}_{\mathcal{R},E} = \mathcal{A}'$ where $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E}) \leadsto^!_E \mathcal{A}'$ and $\mathcal{A}' = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})\alpha_{n+1}$.

If there exists $k \in \mathbb{N}$ such that $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^k_{\mathcal{R},E}) = \mathcal{A}^k_{\mathcal{R},E} = \mathcal{A}'$ then we define $\mathcal{A}^*_{\mathcal{R},E} = \mathcal{A}^k_{\mathcal{R},E}$. ⋄

Note that the definition of $\mathcal{A}^*_{\mathcal{R},E}$ does not depend on the value of k, since if $\mathcal{A}^{k+1}_{\mathcal{R},E} = \mathcal{A}^k_{\mathcal{R},E}$ then $\mathcal{A}^n_{\mathcal{R},E} = \mathcal{A}^k_{\mathcal{R},E}$ for all $n \geq k$, i.e., completion and simplification do not change the automaton any more. In practice, if our algorithm terminates, then it does so at the smallest $k \in \mathbb{N}$ with the above property - we say that our algorithm converges in k steps. Note also that $\mathcal{A}^*_{\mathcal{R},E}$ does not exist in general, but it can be computed in many interesting cases, provided that the set of equations E ensures termination of the completion. A simple example follows. More substantial examples will be given in Section 7.

Example 40. 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 steps. Completion steps are summed up in the following table. To simplify the presentation, we do not repeat the common transitions, i.e. $\mathcal{A}^i_{\mathcal{R},E}$ is supposed to contain all transitions of $\mathcal{A}^0, \ldots, \mathcal{A}^{i-1}_{\mathcal{R},E}$.

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

The automaton $\mathcal{A}^1_{\mathcal{R},E}$ is exactly $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^0)$ since equations do not apply. Then $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})$ contains all the transitions of $\mathcal{A}^1_{\mathcal{R},E}$ plus those obtained by the resolution of the critical pair $f(q_1, q_2) \to_{\mathcal{A}}{}^* q_3$ and $f(q_1, q_2) \to_{\mathcal{R}} f(s(q_1), s(q_2))$. Solving this critical pair according to Definition 34 adds the transitions shown in above table. However, on this last automaton, simplification can be applied as follows:

$$\begin{tikzcd}
s(s(q_a)) \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & s(q_a) \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_4 & q_1
\end{tikzcd}
\qquad
\begin{tikzcd}
s(s(q_b)) \arrow[r, equal, "E"'] \arrow[d, "\mathcal{A},\not\epsilon"', "*"] & s(q_b) \arrow[d, "*"', "\mathcal{A},\not\epsilon"] \\
q_5 & q_2
\end{tikzcd}$$

Hence, $\mathcal{A}^2_{\mathcal{R},E}$ is obtained from $\mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})$ by renaming q_4 by q_1 and q_5 by q_2, i.e. $\mathcal{A}^2_{\mathcal{R},E} = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^1_{\mathcal{R},E})\{q_4 \mapsto q_1, q_5 \mapsto q_2\}$.

6.1. Proving the Lower Bound: $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$

We need a few lemmas. The first one says that the languages of the successive automata produced by the algorithm in Definition 39 form a monotonously increasing sequence. The renamings occurring in the algorithm, and their composition, are taken into account. Hereafter, we denote by $\Pi_{i=1}^n \alpha_i = \alpha_1 \circ \cdots \circ \alpha_n$ the composition of the renaming functions occurring in Definition 39. By convention, for $n = 0$, $\Pi_{i=1}^n \alpha_i$ is the identity function.

Lemma 41. *For all $n \in \mathbb{N}$, let $\beta_n = \Pi_{i=1}^n \alpha_i$. Then, $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{L}(\mathcal{A}^{n+1}_{\mathcal{R},E}, q\beta_{n+1})$.*

Proof. We have $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{L}(\mathcal{C}_R(\mathcal{A}^n_{\mathcal{R},E}), q\beta_n)$ because completion is an over-approximation, and $\mathcal{L}(\mathcal{C}_R(\mathcal{A}^n_{\mathcal{R},E}), q\beta_n) \subseteq \mathcal{L}(\mathcal{C}_R(\mathcal{A}^n_{\mathcal{R},E})\alpha_{n+1}, q\beta_n\alpha_{n+1})$ by Lemma 16. Since by definition $\mathcal{A}^{n+1}_{\mathcal{R},E} = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})\alpha_{n+1}$ and $\beta_{n+1} = \beta_n\alpha_{n+1}$ we obtain the result. $\square$

Therefore, all the languages of the automata produced by our algorithm include $\mathcal{L}(\mathcal{A}, q)$:

Lemma 42. *For all $n \in \mathbb{N}$, $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n)$ where $\beta_n = \Pi_{i=1}^n \alpha_i$.*

Proof. By induction on n. The base case uses the convention that $\Pi_{i=1}^0 \alpha_i$ is the identity. For the inductive step, $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{L}(\mathcal{A}^{n+1}_{\mathcal{R},E}, q\beta_{n+1})$ by Lemma 41. $\square$

Next, the languages of automata closed under completion are closed under rewriting.

Lemma 43. *If $\mathcal{R}$ is left-linear and $\mathcal{A} = \mathcal{C}_{\mathcal{R}}(\mathcal{A})$ then $\mathcal{L}(\mathcal{A}, q) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}, q))$.*

Proof. The $\subseteq$ inclusion is trivial. For the $\supseteq$ inclusion, we replace $\mathcal{L}(\mathcal{C}_{\mathcal{R}}(\mathcal{A}), q)$ by $\mathcal{L}(\mathcal{A}, q)$ in Lemma 38 since they are equal by our hypothesis, and obtain $\mathcal{R}(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{L}(\mathcal{A}, q)$. Hence, $\mathcal{R}^n(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{L}(\mathcal{A}, q)$ for all $n \in \mathbb{N}$. The conclusion follows by taking $n \to \infty$. $\square$

The next lemma is "almost" the lower-bound result. The renamings of the states occurring in the algorithm, and the composition of renamings, are taken into account.

Lemma 44. *Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of equations. If the algorithm in Definition 39 converges for $\mathcal{A}$ and $\mathcal{R}$ in n steps, then, $\mathcal{R}^*(\mathcal{L}(\mathcal{A}), q) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q\beta_n)$ where $\beta_n = \Pi_{i=1}^n \alpha_i$.*

Proof. Since our algorithm converges in n steps, $\mathcal{A}^*_{\mathcal{R},E} = \mathcal{A}^n_{\mathcal{R},E}$. Then, using Lemma 42, $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q\beta_n)$ and then (†) $\quad \mathcal{R}^*(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q\beta_n))$. The condition of termination of our algorithm in Definition 39 implies $\mathcal{A}^*_{\mathcal{R},E} = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^*_{\mathcal{R},E})$. Then, by Lemma 43, $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q\beta_n) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q\beta_n))$, and the conclusion follows from (†). $\square$

The lower-bound result is obtained from Lemma 44 and a few identities. Recall that in Definition 10, by renaming using a function α an automaton $\mathcal{A}$ whose set of accepting states is $\mathcal{Q}_f$, the set of accepting states of $\mathcal{A}\alpha$ is $\mathcal{Q}_f\alpha = \{q_f\alpha \mid q_f \in \mathcal{Q}_f\}$.

Theorem 45. *Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of equations such that algorithm in Definition 39 converges. Then $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$.*

Proof. Assume that the algorithm converges in n steps. We have the relations $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) = \mathcal{R}^*(\bigcup_{q_f \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q_f)) \subseteq \mathcal{R}^*(\bigcup_{q_f \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q_f\beta_n))$ by Lemma 44, and the last expression equals $\bigcup_{q_f \in \mathcal{Q}_f} \mathcal{R}^*(\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q_f\beta_n)) = \bigcup_{q'_f \in \mathcal{Q}_f\beta_n} \mathcal{R}^*(\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q'_f)) = \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$. $\square$

6.2. *Proving the Upper Bound: $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$*

For proving our upper-bound (or precision) result we also need a few lemmas. The first one says that the automata produced by our algorithm are $\mathcal{R}/E$-coherent - provided, of course, that the algorithm is given an $\mathcal{R}/E$-coherent automaton as input.

Lemma 46. *Let $\mathcal{R}$ be a left-linear TRS, E a set of equations and $\mathcal{A}$ a $\mathcal{R}/E$-coherent tree automaton. For all $n \in \mathbb{N}$, $\mathcal{A}^n_{\mathcal{R},E}$ is $\mathcal{R}/E$-coherent.*

Proof. $\mathcal{A}^n_{\mathcal{R},E}$ is obtained from the $\mathcal{R}/E$-coherent automaton $\mathcal{A}$ by iterating completion and simplification steps, which both preserve $\mathcal{R}/E$-coherence by Theorems 36 and 30. □

The second lemma shows that our algorithm "preserves" the representatives of states. The renamings occurring in the algorithm, and their composition, are taken into account.

Lemma 47. *Let $\mathcal{R}$ be a left-linear TRS, E a set of equations and $\mathcal{A}$ a $\mathcal{R}/E$-coherent tree automaton. For all $n \in \mathbb{N}$, $rep_{\mathcal{A}^n_{\mathcal{R},E}}(q\beta_n) =_E rep_{\mathcal{A}}(q)$.*

Proof. By induction on n. The base case is trivial. For the inductive step, we know that $\mathcal{A}^{n+1}_{\mathcal{R},E} = \mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})\alpha_{n+1}$. Then, $rep_{\mathcal{A}^{n+1}_{\mathcal{R},E}}(q\beta_{n+1}) = rep_{\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})\alpha_{n+1}}((q\beta_n)\alpha_{n+1}) =_E rep_{\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})}(q\beta_n)$ by Corollary 31. Now, $q\beta_n$ is a state of $\mathcal{A}^n_{\mathcal{R},E}$, and by Corollary 37, $rep_{\mathcal{C}_{\mathcal{R}}(\mathcal{A}^n_{\mathcal{R},E})}(q\beta_n) =_E rep_{\mathcal{A}^n_{\mathcal{R},E}}(q\beta_n)$, and the result follows by induction hypothesis. □

The next lemma is "almost" the upper-bound result. The renamings of the states occurring in the algorithm, and the compositions of renamings, are taken into account.

Lemma 48. *Let $\mathcal{R}$ be a left-linear TRS, E a set of equations and $\mathcal{A}$ a $\mathcal{R}/E$-coherent tree automaton. For all $n \in \mathbb{N}$, $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}, q))$.*

Proof. $\mathcal{A}^n_{\mathcal{R},E}$ is $\mathcal{R}/E$-coherent by Lemma 46. Using Lemma 21 we obtain $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{R}^*_E(rep_{\mathcal{A}^n_{\mathcal{R},E}}(q\beta_n))$, and $rep_{\mathcal{A}^n_{\mathcal{R},E}}(q\beta_n) =_E rep_{\mathcal{A}}(q)$ by Lemma 47. Hence, $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{R}^*_E(rep_{\mathcal{A}})(q)$, and $\mathcal{L}(\mathcal{A}^n_{\mathcal{R},E}, q\beta_n) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}, q))$ follows since $rep_{\mathcal{A}}(q) \in \mathcal{L}(\mathcal{A}, q)$. □

The upper-bound result is obtained from Lemma 48 and a few trivial identities:

Theorem 49. *Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of equations such that algorithm in Definition 39 converges. Then, $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$.*

Proof. Assume that the algorithm converges in n steps. We have the relations $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) = \bigcup_{q'_f \in \mathcal{Q}_f\beta_n} \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q'_f) = \bigcup_{q_f \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}, q_f\beta_n) \subseteq \bigcup_{q_f \in \mathcal{Q}_f} \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}, q_f))$ by Lemma 48, and then $\bigcup_{q_f \in \mathcal{Q}_f} \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}, q_f)) = \mathcal{R}^*_E(\bigcup_{q_f \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q_f)) = \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. □

The last result in this section shows that, for linear equations having the same set of variables in their left and right-hand sides, a certain instance of our completion algorithm produces (if it terminates) an automaton that recognizes exactly the set of $\mathcal{R}/E$-reachable terms. This may be interesting when E are the usual equations for associativity, commutativity, and identity: the corollary shows that we can perform rewriting *modulo* associativity, commutativity, and identity, or *modulo* a subset of these properties.

Corollary 50. *Let $\mathcal{R}$ be a left-linear TRS, E be a set of linear equations such that $\mathcal{V}ar(l) = \mathcal{V}ar(r)$ for all $l = r \in E$, and $\mathcal{A}$ be a $\mathcal{R}/E$-coherent tree automaton. Let $\overleftrightarrow{E} = \{l \to r, r \to l \mid l = r \in E\}$. If our completion algorithm with equations E and TRS $\mathcal{R} \cup \overleftrightarrow{E}$ terminates, then the automaton $\mathcal{A}^*_{\mathcal{R}\cup\overleftrightarrow{E},E}$ satisfies $\mathcal{L}(\mathcal{A}^*_{\mathcal{R}\cup\overleftrightarrow{E},E}) = \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$.*

Proof. Since E is linear and $\mathcal{V}ar(l) = \mathcal{V}ar(r)$ for all $l = r \in E$, $\mathcal{R} \cup \overleftrightarrow{E}$ is a left-linear TRS. Using Theorem 45 we obtain $\mathcal{L}(\mathcal{A}^*_{\mathcal{R}\cup\overleftrightarrow{E},E}) \supseteq (\mathcal{R} \cup \overleftrightarrow{E})^*(\mathcal{L}(\mathcal{A}))$. We clearly have $(\mathcal{R} \cup \overleftrightarrow{E})^*(\mathcal{L}(\mathcal{A})) = \mathcal{R}^*_E(\mathcal{L}(\mathcal{A})$, and then $\mathcal{L}(\mathcal{A}^*_{\mathcal{R}\cup\overleftrightarrow{E},E}) \supseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. To obtain the opposite inclusion, we use Theorem 49 and obtain $\mathcal{L}(\mathcal{A}^*_{\mathcal{R}\cup\overleftrightarrow{E},E}) \subseteq (\mathcal{R} \cup \overleftrightarrow{E})^*_E(\mathcal{L}(\mathcal{A}))$. The conclusion follows from $(\mathcal{R} \cup \overleftrightarrow{E})^*_E(\mathcal{L}(\mathcal{A})) = (\mathcal{R} \cup \overleftrightarrow{E})^*(\mathcal{L}(\mathcal{A})) = \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$. □

7. Experiments

In this section we show some verification examples that have been be carried out with an implementation of our main algorithm, presented in the previous section. We focus on verifying safety properties and thus essentially rely on Theorem 45. We have developed a new version of the Timbuk tool to support our new algorithm: Timbuk 3.0. The experiments of this section have been performed using this prototype. Here, our aim is not to compare execution times of our tool with recent implementations of completion procedures (Balland et al., 2008; Gallagher and Rosendahl, 2008). We just note that, for the examples in this section, our algorithm terminates in less than one second.

The convention followed hereafter is that the variable's names start with capital letters.

7.1. An introductory example

We borrow from (Clavel et al., 2007) the example of a readers-writers system. System states are represented by terms of the form `state(R,W)` where `R` indicates the number of readers and `W` indicates the number of writers accessing a file. Readers and writers can leave the file at any time, writers can gain access to the file if nobody else is using it, and readers can gain access to the file only if there are no writers. The initial state is `state(o,o)`.The two properties to prove are *mutual exclusion between readers and writers* and *mutual exclusion between writers*. This is defined by the following Timbuk specification.

```
Ops state:2 o:0 s:1             % arities of function symbols
Vars R W
TRS R1
  state(o,o) -> state(o,s(o))   % Writer can start if alone
  state(R,o) -> state(s(R),o)   % Reader can start if no writer
  state(R,s(W)) -> state(R,W)   % Readers and writers can stop at any time
  state(s(R),W) -> state(R,W)   %
Set A1
  state(o,o)
Patterns
  state(s(_),s(_))              % at least one writer and one reader
  state(_,s(s(_)))              % at least two writers
Equations Abs
Rules
  s(s(_))=s(s(o))
```

In Timbuk 3.0 specifications, the `TRS` section is followed by either an `Automaton` or a `Set` section defining the initial set of terms, a `Patterns` section defining the patterns of forbidden terms and an `Equations` section defining the approximation equations. Patterns are terms with variables (possibly anonymous, the '_' symbol used here) that are matched on the completed tree automaton. If a match is found then the term matched by the pattern may be reachable. Searching for a pattern t is similar to searching for a critical pair, i.e. finding a substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ and a state q such that $t\sigma \rightarrow_{\mathcal{A}}{}^{*} q$. This is efficiently implemented using tree automata intersections (Feuillade et al., 2004).

Finally, the `Equations` here consists of the single equation `s(s(_))=s(s(o))`, which identifies all natural numbers $x \geq 2$ with 2. On this example, Timbuk immediately finds a fixpoint tree automaton where no occurrence of the forbidden patterns is found. This means that both mutual-exclusion properties hold.

Timbuk also allows for *contextual equations*, which are of three different forms:
(1) $[s] \Rightarrow [s_1 = t_1 \ \ldots \ s_n = t_n]$
(2) $[s, t] \Rightarrow [s_1 = t_1 \ \ldots \ s_n = t_n]$
(3) $[s = t] \Rightarrow [s_1 = t_1 \ \ldots \ s_n = t_n]$
For a tree automaton $\mathcal{A}$, applying contextual equations on $\mathcal{A}$ is done as follows. The right-hand side of $\Rightarrow$ is a list of equations to be applied on $\mathcal{A}$ provided that the left-hand side can be matched on $\mathcal{A}$. A left-hand side of the form $[s]$ means that we look only for a matching for s on $\mathcal{A}$; if the left-hand side is $[s, t]$ then we look for matches for both s and t independently, and if the left-hand side is $[s = t]$ we look for solutions such that matches for s and t belong to the same equivalence class, i.e. are recognized by the same state of $\mathcal{A}$. In our specification, we can replace the equation `s(s(_))=s(s(o))` by the following contextual equation:
`[state(s(s(R)), o)] => [s(s(R))=s(s(o))]`
meaning that every natural number $x \geq 2$ is identified with 2 provided that it appears on the reader position. This leads to a more precise approximation, comparable to the one used in (Clavel et al., 2007). The bakery algorithm, also verified in (Meseguer et al., 2003), can also be handled using approximation equations. For proving the basic safety property, saying that the two processes cannot access the critical section at the same time, we only use a single equation. In our setting we do not need any additional proof of ground confluence, termination, and coherence, to prove safety properties of the system. On the other hand, the approach of Meseguer et al. deals with more general linear-temporal logic properties, which are not currently handled by our approach.

7.2. Comparing with normalization rules

In this section, we consider a distributed infinite-state system borrowed from (Genet and Viet Triem Tong, 2001a) and prove that it is deadlock free. We show that equational approximations are more concise than the normalization rules of (Genet and Viet Triem Tong, 2001a).

This example consists of two processes that count '+'and '−' symbols in a list. One process, let us call it P_+, counts the '+' symbols and the other one, P_-, counts the '−' symbols. Initially, the list is divided into two parts, and each part is given to one process. Each process also has an incoming message queue. The process P_+ counts the '+' symbols in its list, sends a message to the queue of P_- each time it finds a '−' symbol, and reads messages in its message queue to take into account the '+' symbols sent by process P_-. The behavior of process P_- is symmetrical to that of P_+.

This system can be described by the following TRS, where a term of the form `state(p1, p2, s1, s2)` represents a state of the system where process P_+ is in a configuration described by the term `p1`, P_- is in a configuration described by the term `p2`, and the message queues for processes P_+ and P_- are respectively `s1` and `s2`. A term of the form `proc(l, c)` is a process configuration where the current list of symbols is `l` and the local counter is `c`. Queues are represented by lists, in which the `add` symbol adds a message at the end of the list. In (Genet and Viet Triem Tong, 2001a), it is shown how completion can be used to find the right algorithm for processes to stop without deadlock. Here, we directly start from the TRS encoding the correct solution: each process adds an `end` symbol to the queue of the other process when it no longer has symbols to count. A process stops when it has no symbols to count and has read the `end` symbol sent by the other process.

```
TRS R1
add(X, nil) -> cons(X, nil)
add(X, cons(Y, Z)) -> cons(Y, add(X, Z))
state(proc(cons(plus, Y), C), Z, M, N) -> state(proc(Y, s(C)), Z, M, N)
state(proc(cons(minus, Y), C), U, M, N) -> state(proc(Y, C), U, M, add(minus, N))
state(X, proc(cons(minus, Y), C), M, N) -> state(X, proc(Y, s(C)), M, N)
state(X, proc(cons(plus, Y), C), M, N) -> state(X, proc(Y, C), add(plus, M), N)
state(proc(X, C), Z, cons(plus,M), N) -> state(proc(X, s(C)), Z, M, N)
state(X, proc(Z, C), M, cons(minus,N)) -> state(X, proc(Z, s(C)), M ,N)
state(proc(nil, C), Z, M, N) -> state(proc(nil, C), Z, M, add(end,N))
state(X, proc(nil, C), M, N) -> state(X, proc(nil, C), add(end,M), N)
state(proc(nil, C), Z, cons(end,M), N)  -> state(stop(C), Z, M, N)
state(X, proc(nil, C), M, cons(end, N)) -> state(X, stop(C), M, N)
```

The initial configuration of the system is described by the following tree automaton, recognizing every configuration where the two processes have a counter equal to zero, a nonempty list of symbols to count from, and an empty message queue.

```
Automaton A1
States q0 qinit qzero qnil qlist qsymb
Final States q0
Transitions
  o -> qzero      nil -> qnil      plus -> qsymb     minus -> qsymb
  cons(qsymb, qnil) ->qlist
  cons(qsymb, qlist) -> qlist
  proc(qlist, qzero) -> qinit
  state(qinit, qinit, qnil, qnil) -> q0
```

A forbidden configuration is any state where a process has terminated but still has symbols to count in its queue. This is given to the Timbuk tool as the following patterns:

```
  state(stop(_),_,cons(plus,_),_)
  state(_,stop(_),_,cons(minus,_))
```

Any term matching one of these patterns is a deadlock situation. To prove the absence of deadlock, (Genet and Viet Triem Tong, 2001a) requires 20 normalization rules closely related to the initial automaton's structure. On the same example, we can do better with equations. To define the approximation it is enough to note that what makes the system infinite-state are the two unbounded lists of '+' and '-' read by the two processes. Each process eithers increases its counter, or adds a symbol with the '*add*' operation to the queue of the other process. Thus, terms of the form $s(s(\ldots))$ and $add(add(\ldots))$ grow without bound. Hence, a natural choice for approximation equations could be:

```
  s(X)=X
  add(X,Y)=Y
```

Using those equations, completion terminates but the approximation is too coarse: it contains terms matching the forbidden patterns. The reason is that the equation `add(X,Y)=Y` identifies, e.g., the terms $s = add(end, add(plus, nil))$ and $t = add(plus, nil)$. In other words, it is possible to replace any queue t where the '*end*' symbol does not appear by a queue s where it appears. This corresponds to the fact that the other process has signaled its termination although it has not terminated, which is a deadlock situation. To avoid this problem, we identify "*add* chains" only if their first parameter is the same symbol:

```
  s(X)=X
  add(plus,add(plus,Z))=add(plus,Z)
  add(minus,add(minus,Z))=add(minus,Z)
  add(end, add(end, Z))=add(end, Z)
```

Using this small set of equations, completion terminates and proves the property. The completed automaton contains 14 states and 123 transitions.

In the next sections, we are concerned with the flow analysis of functional programs. First, we consider an example borrowed from (Jones and Andersen, 2007). Their analysis produces a tree grammar encoding a flow analysis of the program. In (Jones and Andersen, 2007) the contributions are the taking into account of of higher-order functions and of lazy evaluation. We here focus on the lazy evaluation part because the grammar for the example they propose, is fully detailed in their paper, and comparisons can be made. The functional program is directly given in its TRS form:

```
  g(N) -> first(N, sequence(nil))
  first(nil, Xs) -> nil
  first(cons(one,M), cons(X,Xs)) -> cons(X,first(M,Xs))
  sequence(Y) -> cons(Y,sequence(cons(one,Y)))
```

For any list `N` composed of n instances of the symbol `one`, the function g builds a list of the n first elements from the infinite list $[nil, [one], [one, one], \ldots]$. This program needs a lazy or outermost evaluation strategy to terminate, because of the `sequence` function that does not terminate. The initial set of terms is defined by the following automaton:

```
Automaton A0
States q0 ql qa q1 qnil
Final States q0
Transitions
  g(ql) -> q0
  cons(qa,ql) -> ql
  cons(q1,ql) -> ql
  cons(q1,qnil) -> ql
  cons(qa,qnil) -> ql
  nil -> qnil
  atom -> qa
  one -> q1
```

that recognizes all terms of the form `g(l)` where `l` is any list of atoms that can be `one` or another atom, as in (Jones and Andersen, 2007). The set of atoms is potentially infinite and grammars or automata can only be finite, hence, is necessary to finitely approximate this set. This is achieved using two constants: `one`, and `atom` for the atoms distinct from `one`.

In (Jones and Andersen, 2007) the objective is to infer the term structure of possible values for parameters and results of every function f without *a priori* knowledge on the inputs of the function. Since completion covers *all* reachable terms, it covers also those that can be reached by a lazy evaluation. In fact, we can achieve exactly the same flow analysis and obtain the same result as (Jones and Andersen, 2007) using *contextual equations*. The intuition behind the approximation used by (Jones and Andersen, 2007) is simply to identify all possible call values for f. Hence, for the function `first` which has two parameters, such an approximation can be defined using the single contextual equation `[first(X,Y), first(Z,U)] => [X=Z  Y=U]`. Similarly, for the function `sequence` the equation will be: `[sequence(X), sequence(Y)] => [X=Y]`. Using those equations, we obtain a completed automaton having 11 states and 18 transitions. Among the transitions, the following subset recognizes the set of results of calls to the function `g`:

```
  nil -> q13
  cons(q10,q13) -> q13
  nil -> q10
  cons(q3,q10) -> q10
  one -> q3
```

i.e., any list whose elements are lists of composed of the symbol `one`.

7.4. Adapting the approximation to the property to prove

In order to illustrate the impact of equations on the precision of the approximation we prove a certain property of the *reverse* function. This function is defined by:

```
  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))
```

Assume that we want to know what can be the result of $rev(l)$ when l is an arbitrary list composed of symbols a, b, c and d (in that order) and such that l contains at least one occurrence of each symbol. The language $rev(l)$ is recognized by the following automaton:

```
Automaton A0
States q0 qla qlb qlc qld qnil qf qa qb qc qd
Final States q0
Transitions
  rev(qla) -> q0
  cons(qa, qla) -> qla
  cons(qa, qlb) -> qla
  cons(qb, qlb) -> qlb
  cons(qb, qlc) -> qlb
  cons(qc, qlc) -> qlc
  cons(qc, qld) -> qlc
  cons(qd, qld) -> qld
  cons(qd, qnil) -> qld
  nil -> qnil
  a -> qa
  b -> qb
  c -> qc
  d -> qd
```

The expected result is, of course, the language of lists whose symbols are in the opposite order and occur at least once. If we use the following equations:

```
  [append(X,Y), append(Z, U)] => [X=Z Y=U]
  [rev(X), rev(Y)] => [X=Y]
```

then, Timbuk produces a tree automaton where state q29 recognizes the result of $rev(l)$:

```
  nil -> q29
  cons(q7,q29) -> q29
  cons(q8,q29) -> q29
  cons(q9,q29) -> q29
  cons(q10,q29) -> q29
  d -> q10
  c -> q9
  b -> q8
  a -> q7
```

Specifically, these transitions recognize into q29 the language of lists possibly containing symbols a, b, c and d in any order. We can improve the approximation by taking the calling context of the *append* function into account. The same idea is used to transform a *0-CFA analysis* into a *1-CFA analysis*: take the direct calling context into account:

```
  [cons(append(X,Y),_), cons(append(Z,U),_)] => [X=Z Y=U]
  [cons(_,append(X,Y)), cons(_,append(Z,U))] => [X=Z Y=U]
  [append(append(X,Y),_), append(append(Z,U),_)] => [X=Z Y=U]
  [append(_,append(X,Y)), append(_,append(Z,U))] => [X=Z Y=U]
```

where we merge call values of *append* only if the calling context at depth 1 is the same. Even though the resulting approximation is more precise, the resulting automaton still does not preserve the order of symbols in the list. Actually, even by distinguishing calling context up to a bounded depth $k \in \mathbb{N}$ (like in a *k-CFA analysis*), the approximation would not be precise enough to obtain the result we expect. However, we can construct a different approximation using the single equation:

```
append(append(X,Y),Z)=append(X,Z)
```

Using this equation, we obtain an approximation preserving the order of symbols: the resulting language contains any list of d, c, b and a in that order. However, the approximation is still too coarse since there is no guarantee on the occurrence of every symbol in the list. This is due to the fact that, using the previous equation, we have in particular the following equality: $append(append(cons(b, nil), cons(a, nil)), nil) = append(cons(b, nil), nil)$ meaning that every occurrence of the first term is equivalent to the second one. That is, our equation preserves the order, but not the occurrences of symbols in the list. Finally, it is possible to use the following equations:

```
cons(a, cons(a, X))=cons(a,X)
cons(b, cons(b, X))=cons(b,X)
cons(c, cons(c, X))=cons(c,X)
cons(d, cons(d, X))=cons(d,X)
```

expressing more precisely where contractions of unbounded lists have to be performed. With these equations, the completed tree automaton recognizes the expected language. The automaton has 19 states and 59 transitions.

8. Conclusion

In this paper we propose a new tree automata completion algorithm and a new approximation mechanism based on equations. The main contribution with respect to the closest related works (Meseguer et al., 2003; Takai, 2004) is that no restriction is imposed on the equations. This makes it easy to adapt the set of equations to the particular objectives of an analysis. On the other hand, our term-rewriting systems have to be left linear. However, this restriction did not prevent us from handling practically interesting case studies. For Java bytecode verification, the TRS encoding the semantics of the program is left-linear (Boichut et al., 2007). For cryptographic protocols, non left-linear rules can be encoded using conditional rules, and a simple extension of completion for conditional rules based on (Feuillade et al., 2004) is possible.

We have also obtained some results on the precision of our equation-based approximations. For a given left-linear TRS $\mathcal{R}$, a $\mathcal{R}/E$-coherent initial tree automaton and a set of equations E, we have shown that our algorithm produces an automaton that recognizes at most $\mathcal{R}/E$-reachable terms. In other words, the computed approximation is within the bounds of the expected approximation defined by the equations E.

The first application of tree automata completion was to prove security properties on cryptographic protocols. Although not presented in this paper, we have verified that approximations similar to those of (Genet and Viet Triem Tong, 2001a) can be obtained using equations. On cryptographic protocols, we came up with sets of equations far simpler than original sets of normalization rules. Experiments are also currently under way on Java bytecode in order to define 0-CFA and 1-CFA analyses, as in (Boichut et al., 2007), using equations.

The price to pay for the conciseness and expresiveness of equational approximation is in the complexity of the algorithms. Finding instances of equation members, and merging corresponding states is more complex that applying a normalization rule. However, fine-tuned data structures in our prototype made it possible to have a better overall efficiency than the Timbuk 2.2 implementation based on normalization rules. Note also that soundess of approximations is not jeopardized by implementation optimizations, since we use the certified Tree Automata Completion Checker of (Boyer et al., 2008) to check the correctness of the results.

Finally, proving safety properties is interesting but not enough. Proving temporal properties on the rewriting graph, like (Meseguer et al., 2003), is also of great interest. In the completion algorithm proposed in this paper, the graph of epsilon transition represents an abstraction of the rewriting graph. In (Boyer and Genet, 2009), from the completed automaton, we build a Kripke structure on which LTL properties can be proved. The proposed construction is limited to finite rewriting graphs. Further research consists of building finite over-approximations of infinite rewriting graphs using equations.

References

Abdulla, P. A., Legay, A., d'Orso, J., Rezine, A., 2006. Tree regular model checking: A simulation-based approach. J. Log. Algebr. Program. 69 (1-2), 93–121.

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

Balland, E., Boichut, Y., Genet, T., Moreau, P.-E., 2008. Towards an Efficient Implementation of Tree Automata Completion. In: AMAST'08. Vol. 5140 of LNCS. Springer.

Boichut, Y., Genet, T., Jensen, T., Leroux, L., 2007. Rewriting Approximations for Fast Prototyping of Static Analyzers. In: RTA. Vol. 4533 of LNCS. Springer, pp. 48–62.

Boichut, Y., Héam, P.-C., Kouchnarenko, O., 2004. Automatic Approximation for the Verification of Cryptographic Protocols. In: Proc. AVIS'2004, joint to ETAPS'04, Barcelona (Spain).

Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T., 2006a. Abstract Regular Tree Model Checking. In: Infinity'05. Vol. 149(1) of ENTCS. pp. 37–48.

Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T., 2006b. Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In: SAS'06. Vol. 4134 of LNCS. Springer, pp. 52–70.

Bouajjani, A., Touili, T., 2002. Extrapolating tree transformations. In: CAV. Vol. 2404 of LNCS. Springer.

Bouajjani, A., Touili, T., 2005. On computing reachability sets of process rewrite systems. In: RTA. Vol. 3467 of LNCS. Springer, pp. 484–499.

Boyer, B., Genet, T., 2009. Verifying Temporal Regular properties of Abstractions of Term Rewriting Systems. In: Proc. of RULE'09.

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

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

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

Dershowitz, N., Jouannaud, J.-P., 1990. Handbook of Theoretical Computer Science. Vol. B. Elsevier Science Publishers B. V. (North-Holland), Ch. 6: Rewrite Systems, pp. 244–320, also as: Research report 478, LRI.

d'Orso, J., Touili, T., 2006. Regular hedge model checking. In: IFIP TCS. Springer, pp. 213–230.

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

Gallagher, J., Rosendahl, M., 2008. Approximating term rewriting systems: a horn clause specification and its implementation. In: LPAR'08. Vol. 5330. Springer.

Genest, B., Muscholl, A., Serre, O., Zeitoun, M., 2008. Tree pattern rewriting systems. In: ATVA'08. Vol. 5311 of LNCS. Springer, pp. 332–346.

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

Genet, T., Klay, F., 2000. Rewriting for Cryptographic Protocol Verification. In: Proc. 17th CADE Conf., Pittsburgh (Pen., USA). Vol. 1831 of LNAI. Springer-Verlag.

Genet, T., Rusu, R., 2009. Equational approximations for tree automata completion. Tech. rep., INRIA, `http://hal.archives-ouvertes.fr/hal-00370166/fr/`.

Genet, T., Tang-Talpin, Y.-M., Viet Triem Tong, V., 2003. Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems. In: WITS'2003.

Genet, T., Viet Triem Tong, V., 2001a. Reachability Analysis of Term Rewriting Systems with *timbuk*. In: Proc. 8th LPAR Conf., Havana (Cuba). Vol. 2250 of LNAI. Springer-Verlag, pp. 691–702.
URL `ftp://ftp.irisa.fr/local/lande/tg-vvtt-lpar01.ps.gz`

Genet, T., Viet Triem Tong, V., 2001b. Timbuk – a Tree Automata Library. IRISA / Université de Rennes 1, `http://www.irisa.fr/celtique/genet/timbuk/`.

Gilleron, R., Tison, S., 1995. Regular tree languages and rewrite systems. Fundamenta Informaticae 24, 157–175.

Jacquemard, F., Rusinowitch, M., 2008. Closure of hedge-automata languages by hedge rewriting. In: RTA'08. Vol. 5117 of LNCS. Springer, pp. 157–171.

Jones, N. D., Andersen, N., 2007. Flow analysis of lazy higher-order functional programs. TCS 375 (1-3), 120–136.

Meseguer, J., Palomino, M., Mart-Oliet, N., 2003. Equational Abstractions. In: Proc. 19th CADE Conf., Miami Beach (Fl., USA). Vol. 2741 of LNCS. Springer, pp. 2–16.

Reynolds, J., 1969. Automatic computation of data set definitions. Information Processing 68, 456–461.

Takai, T., 2004. A Verification Technique Using Term Rewriting Systems and Abstract Interpretation. In: Proc. 15th RTA Conf., Aachen (Germany). Vol. 3091 of LNCS. Springer, pp. 119–133.