Reachability Analysis over Term Rewriting Systems

Guillaume Feuillade *
IRISA / ENS Cachan
Campus de Beaulieu
F-35042 Rennes Cedex

Thomas Genet † and Valérie Viet Triem Tong ‡
IRISA / Université de Rennes 1
Campus de Beaulieu
F-35042 Rennes Cedex

Abstract. This paper surveys some techniques and tools for achieving reachability analysis over term rewriting systems. The core of those techniques is a generic tree automata completion algorithm used to compute in an exact or approximated way the set of descendants (or reachable terms). This algorithm has been implemented in the Timbuk tool. Furthermore, we show that many classes with regular sets of descendants of the literature corresponds to specific instances of the tree automata completion algorithm and can thus be efficiently computed by Timbuk. An extension of the completion algorithm to conditional term rewriting systems and some applications are also presented.

Introduction

Given a term rewriting system $\mathcal{R}$ and two ground terms s and t, we focus on proving automatically that $s \rightarrow_{\mathcal{R}}^* t$ or $s \not\rightarrow_{\mathcal{R}}^* t$. This problem has several applications in equational proofs used in theorem proving or in proof assistants as well as in verification where term rewriting systems can be used to model programs. The reachability problem is known to be decidable for Term Rewriting Systems (TRS for short) which are terminating. However, in automated deduction and in verification, systems considered in practice are rarely terminating and, even when they are, automatically proving their termination is difficult. On the other hand, reachability is known to be decidable on several syntactic classes of term rewriting systems (not necessarily terminating nor confluent). On those classes, the technique used to prove reachability is based on the computation of the set $\mathcal{R}^\star(E)$ of $\mathcal{R}$-descendants (or $\mathcal{R}$-reachable terms) of an initial set of terms E. For those classes, $\mathcal{R}^\star(E)$ is a regular tree language if E is also regular and can thus be represented using a *tree automaton*. Tree automata offer a finite way to represent

* gfeuilla@irisa.fr
† genet@irisa.fr
‡ viettrie@irisa.fr

© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

infinite (regular) sets of reachable terms when a non terminating term rewriting system is under concern. However the algorithms used to build $\mathcal{R}^\star(E)$ are rather different from a class to another.

In this paper, our aim is to propose a common, simple, efficient and implemented algorithm for computing set of descendant terms for known classes and to construct some approximation when it is not decidable. This algorithm is essentially a *completion* of a *tree automaton*, thus taking advantage of an algorithm similar to the Knuth-Bendix *completion* (Knuth and Bendix, 1970) in order not to restrict to a specific syntactic class of term rewriting systems and *tree automata* so as to deal efficiently with infinite sets of reachable terms produced by non-terminating term rewriting systems.

This algorithm is implemented in the Timbuk tool (Genet and Viet Triem Tong, 2000). However, as we will see in the following, our implementation does not cover *every* decidable class since this would have led to an inefficient tool. As an example, for dealing with non left-linear TRSs, one can refine the algorithm we propose by applying determinisation after each step of tree automata completion. In this way, one may obtain a more general theorem covering the non left-linear case without restriction. However, since determinisation is an exponential-time operation, this would not be realistic in practice. Thus, we choose to stick to the basic completion algorithm and give some conditions sufficient for covering the non left-linear case in many practical cases.

The paper is organized as follows. In section 1 we recall the basic notations for term rewriting systems and tree automata. Then, in section 2 we recall the known classes of TRS preserving regularity. Section 3 presents the tree automata completion algorithm and the result for over-approximation of $\mathcal{R}^\star(E)$ for any TRS $\mathcal{R}$ and any initial regular language E. In section 4 we give some sufficient conditions for the tree automata completion to compute exactly $\mathcal{R}^\star(E)$ for any TRS $\mathcal{R}$ and any initial regular language E. In this section we also show how some regular classes of the literature can be obtained using tree automata completion. In section 5, an extension of the tree automata completion algorithm for conditional term rewriting systems is given. Then, in section 6, we present the Timbuk tool and the techniques to perform exact and approximated completions. In section 7, some applications of $\mathcal{R}^\star(E)$ are presented: sufficient completeness, strong non-termination proof and reachability testing. Finally, in section 8, we present the algorithmic optimization of matching in automata in Timbuk and some indications on its efficiency.

1. Formal background

Comprehensive surveys can be found in (Dershowitz and Jouannaud, 1990; Baader and Nipkow, 1998) for term rewriting systems, in (Comon et al., 2002; 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 function ar, 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 $\mathcal{V}ar(t)$. The domain and range of a function will be denoted respectively by $\mathcal{D}om$ and $\mathcal{R}an$. A substitution is a function σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F},\mathcal{X})$, which can uniquely be extended to an endomorphism of $\mathcal{T}(\mathcal{F},\mathcal{X})$. Its domain $\mathcal{D}om(\sigma)$ is finite and such that $\{x \in X \mid x\sigma \neq x\}$. A position p for a term t is a word over $\mathbb{N}$. The empty sequence ϵ denotes the top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by:

- $\mathcal{P}os(t) = \{\epsilon\}$ if $t \in \mathcal{X}$
- $\mathcal{P}os(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\}$

If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. For any term $s \in \mathcal{T}(\mathcal{F},\mathcal{X})$, we denote by $\mathcal{P}os_{\mathcal{F}}(s)$ the set of functional positions in s, i.e. $\{p \in \mathcal{P}os(s) \mid p \neq \epsilon \text{ and } \mathcal{R}oot(s|_p) \in \mathcal{F}\}$ where $\mathcal{R}oot(t)$ denotes the symbol at position ϵ in t. Conversely, we denote by $\mathcal{P}os_{\mathcal{X}}(s)$ the set of variable positions in s, i.e. $\mathcal{P}os_{\mathcal{X}}(s) = \{p \in \mathcal{P}os(s) \mid s|_p \in \mathcal{X}\}$.

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 rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if each variable of l (resp. r) occurs only once in l (resp. in r). A rule is linear if it is both left and right-linear. A TRS $\mathcal{R}$ is linear (resp. left-linear, right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is linear (resp. left-linear, right-linear). The TRS $\mathcal{R}$ induce a rewriting relation $\to_{\mathcal{R}}$ on terms whose reflexive transitive closure is denoted by $\to^{\star}_{\mathcal{R}}$. The set of $\mathcal{R}$-descendants of a set of ground terms E is $\mathcal{R}^{\star}(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to^{\star}_{\mathcal{R}} t\}$. We extend this notation to terms in the following way: $\mathcal{R}^{\star}(s) = \mathcal{R}^{\star}(\{s\})$. We denote by $IRR(\mathcal{R})$ the set of terms irreducible by $\mathcal{R}$ and by $\mathcal{R}^{!}(E)$ the set of $\mathcal{R}$-normal forms of E, i.e. $\mathcal{R}^{!}(E) = \mathcal{R}^{\star}(E) \cap IRR(\mathcal{R})$.

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 1 (Transition and normalized 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 $c = q' \in \mathcal{Q}$ or $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$, $ar(f) = n$, and $q_1, \ldots, q_n \in \mathcal{Q}$.*

An epsilon transition is a transition of the form $q \to q'$ where q and q' are states. Any set of transition $\Delta \cup \{q \to q'\}$ can be equivalently replaced by $\Delta \cup \{c \to q' \mid c \to q \in \Delta\}$.

DEFINITION 2 (Bottom-up non-deterministic finite tree automaton). *A bottom-up non-deterministic finite tree automaton (tree automaton for short) 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.*

A tree automaton is *deterministic* if there are no two rules with the same left-hand side and no rule of the form $q' \to q$ with $q, q' \in \mathcal{Q}$. The *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ induced by the transitions of $\mathcal{A}$ (the set Δ) is denoted by $\to_\Delta$. When Δ is clear from the context, $\to_\Delta$ will also be denoted by $\to_\mathcal{A}$. Similarly, by notation abuse, we will often note $q \in \mathcal{A}$ and $t \to q \in \mathcal{A}$ respectively for $q \in \mathcal{Q}$ and $t \to q \in \Delta$.

DEFINITION 3 (Recognized language). *The tree language recognized by a state q in $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_\mathcal{A}^\star q\}$. The language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q)$. A tree language is regular if and only if it can be recognized by a tree automaton.*

A state q is a *dead state* if $\mathcal{L}(\mathcal{A}, q) = \emptyset$.

EXAMPLE 1. *Let $\mathcal{A}$ be the tree automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ such that $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_0) \to q_0, g(q_1) \to q_0, g(q_2) \to q_2, a \to q_1\}$. In Δ transitions are normalized. A transition of the form $f(g(q_2)) \to q_0$ is not normalized. The term $g(a)$ is a term of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ (and of $\mathcal{T}(\mathcal{F})$) and can be rewritten by Δ in the following way: $g(a) \to_\Delta g(q_1) \to_\Delta q_0$. Note that $\mathcal{L}(\mathcal{A}, q_1) = \{a\}$ and $\mathcal{L}(\mathcal{A}, q_0) = \{f(g(a)), f(f(g(a))), \ldots\} = \{f^\star(g(a))\}$. Note also that $\mathcal{L}(\mathcal{A}, q_2) = \emptyset$ since no term rewrites to q_2, hence q_2 is a dead state.*

2. Existing solutions

The basic reachability problem we are going to consider is the following: given a term rewriting system $\mathcal{R}$ and two terms $s, t \in \mathcal{T}(\mathcal{F})$, can we decide whether $s \to_\mathcal{R}^\star t$ or not? In this part, we focus on the existing solutions designed for particular cases.

The simplest case is when $\mathcal{R}$ is terminating. Here is a simple but inefficient procedure: to decide whether $s \to_{\mathcal{R}}^{\star} t$ or not it is enough to see if $t \in \mathcal{R}^{\star}(s)$ since $\mathcal{R}^{\star}(s)$ is finite and computable.

When $\mathcal{R}$ is not terminating, deciding reachability needs some additional formal tools. For instance, tree automata can be used to finitely represent the infinite set $\mathcal{R}^{\star}(s)$ and then check if $t \in \mathcal{R}^{\star}(s)$. Many works are devoted to the construction of $\mathcal{R}^{\star}(E)$ for a regular language E and a term rewriting system $\mathcal{R}$ fulfilling some restrictions:

- $\mathcal{R}$ is either a ground TRS (Dauchet and Tison, 1990; Brainerd, 1969).
- a right-linear and monadic TRS (Salomaa, 1988), i.e. right-hand sides of the rules are either variables or terms of the form $f(x_1, \ldots, x_n)$ where $f \in \mathcal{F}$ and $x_1, \ldots, x_n$ are variables.
- a linear and semi-monadic TRS (Coquidé et al., 1991), i.e. rules are linear and their right-hand sides are of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ and $\forall i = 1, \ldots, n$, t_i is either a variable or a ground term.
- a "decreasing" TRS (Jacquemard, 1996), where "decreasing" means that every right-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $ar(f) = n$, and $\forall i = 1, \ldots, n$, t_i is a variable, a ground term, or a term whose variables do not occur in the left-hand side.
- Linear generalized semi-monadic TRS (Gyenizse and Vágvölgyi, 1998), Right-linear finite-path overlapping TRS (Takai et al., 2000) and Layered transducing TRS (Seki et al., 2002) which are more recent decidable classes where those criteria do a more careful inspection of the syntactic structure of rewrite rules so that recursive application of rewrite rules are guaranteed to preserve regularity. Note that (Takai et al., 2000) includes all the previous classes except (Seki et al., 2002) which is orthogonal. Those classes are not covered by the completion algorithm we propose in this paper.

On the other hand, for a given regular language E, $\mathcal{R}^{\star}(E)$ is not necessarily regular, even if $\mathcal{R}$ is a confluent and terminating linear TRS (Gilleron and Tison, 1995).

Another regular class was found by P. Réty (Réty, 1999) where restrictions are weaker on the TRS and stronger on the regular language E. The alphabet $\mathcal{F}$ is separated into a set of *defined symbols* $\mathcal{D} = \{f \mid \exists l \to r \in \mathcal{R} \text{ s.t. } \mathcal{R}oot(l) = f\}$ and constructor symbols $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. The restriction on E is the following: E is the set of

ground constructor instances of a linear term t, i.e. $E = \{t\sigma\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})$. The restrictions on $\mathcal{R}$ are the following: for each rule $l \to r$

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

Finally, P. Réty and J. Vuotto have shown that $\mathcal{R}^\star(E)$ is still regular with the same restrictions on $\mathcal{R}$ and E when rules of $\mathcal{R}$ are applied under some specific strategies (innermost, outermost, ...) (Réty and Vuotto, 2002).

3. Tree Automata completion

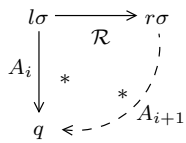
In (Genet, 1998), we proposed a *tree automaton completion* algorithm for over-approximating $\mathcal{R}^\star(E)$ for left-linear term rewriting systems and a regular language E. The completion is parametrized by an abstraction function α mapping terms to states of the automaton.

Let us first recall the tree automata completion algorithm. Starting from a tree automaton $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_0 \rangle$ and a left-linear[1] TRS $\mathcal{R}$, the aim of the approximation algorithm is to compute a tree automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}_0))$. Approximations are used to show that terms recognized by a tree automaton $\mathcal{A}_{bad}$ are not reachable by rewriting terms of $\mathcal{L}(\mathcal{A}_0)$ with $\mathcal{R}$, i.e. $\forall s \in \mathcal{L}(\mathcal{A}_0)\ \forall t \in \mathcal{L}(\mathcal{A}_{bad}) : s \not\to_{\mathcal{R}}{}^* t$. For this, it is enough to show that $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}_{bad}) = \emptyset$ i.e., compute the automaton recognizing the intersection and show that the recognized language is empty.

The technique consists in successively computing tree automata $\mathcal{A}_1$, $\mathcal{A}_2$, ...such that $\forall i \geq 0 : \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$ and if $s \in \mathcal{L}(\mathcal{A}_i)$, such that $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$, until we get an automaton $\mathcal{A}_k$ with $k \in \mathbb{N}$ such that $\mathcal{L}(\mathcal{A}_k) = \mathcal{L}(\mathcal{A}_{k+1})$. Thus, $\mathcal{A}_k$ is a fixpoint and $\mathcal{A}_k$ also verifies $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}_0))$. More precisely, to construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, we achieve a *completion step* which consists in finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to^\star_{\mathcal{A}_i} q$ and $r\sigma \not\to^\star_{\mathcal{A}_i} q$. For $r\sigma$ to be recognized as

[1] This restriction will be weakened in the following.

the same state and thus model the rewriting of $l\sigma$ into $r\sigma$, it is enough to join the critical pair:



and add the new transition $r\sigma \to q$ to $\mathcal{A}_{i+1}$. However, the transition $r\sigma \to q$ is not necessarily a normalized transition of the form $f(q_1, \ldots, q_n) \to q'$ and so has to be normalized first. For example, to normalize a transition of the form $f(g(a), h(q')) \to q$, we need to find some states q_1, q_2, q_3 and replace the previous transition by a set of normalized transitions: $\{a \to q_1, g(q_1) \to q_2, h(q') \to q_3, f(q_2, q_3) \to q\}$.

Assume that q_1, q_2, q_3 are new states, then adding the transition itself or its normalized form does not make any difference. Now, assume that $q_1 = q_2$, the normalized form becomes $\{a \to q_1, g(q_1) \to q_1, h(q') \to q_3, f(q_1, q_3) \to q\}$. This set of normalized transitions represents the regular set of non normalized transitions of the form $f(g^\star(a), h(q')) \to q$ which contains the transition we wanted to add initially but also many others. Hence, this is an approximation. We could have made an even more drastic approximation by identifying q_1, q_2, q_3 with q, for instance.

For every transition, there exists an equivalent set of normalized transitions. Normalization consists in decomposing a transition $s \to q$, into a set $Norm(s \to q)$ of normalized transitions. The method consists in abstracting subterms s' of s s.t. $s' \notin \mathcal{Q}$ by states of $\mathcal{Q}$. We first define the abstraction function as follows:

DEFINITION 4 (Abstraction function). *Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. An* abstraction *function α maps every normalized configuration into a state:*

$$\alpha : \{f(q_1, \ldots, q_n) \mid f \in \mathcal{F}, ar(f) = n \text{ and } q_1, \ldots q_n \in \mathcal{Q}\} \mapsto \mathcal{Q}$$

DEFINITION 5 (Abstraction state). *Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. For a given abstraction function α and for all configuration $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ the abstraction state of t, denoted by $top_\alpha(t)$, is defined by:*

1. *if $t \in \mathcal{Q}$, then $top_\alpha(t) = t$,*
2. *if $t = f(t_1, \ldots, t_n)$ then $top_\alpha(t) = \alpha(f(top_\alpha(t_1), \ldots, top_\alpha(t_n)))$.*

DEFINITION 6 (Normalization function). *Let $\mathcal{F}$ be a set of symbols, $\mathcal{Q}$ a set of states, $s \to q$ a transition s.t. $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$, and α an abstraction function. The set $Norm_\alpha(s \to q)$ of* normalized transitions *is inductively defined by:*

1. *if* $s = q$, *then* $Norm_\alpha(s \to q) = \emptyset$, *and*

2. *if* $s \in \mathcal{Q}$ *and* $s \neq q$, *then* $Norm_\alpha(s \to q) = \{s \to q\}$, *and*

3. *if* $s = f(t_1, \ldots, t_n)$, *then* $Norm_\alpha(s \to q) =$
$\{f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to top_\alpha(t_i))$.

EXAMPLE 2. *Let* $\mathcal{F} = \{f, g, a\}$ *and* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, *where* $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_0\}$, *and* $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.

• *The languages recognized by* q_1 *and* q_0 *are the following:* $\mathcal{L}(\mathcal{A}, q_1)$ *is the set of terms built on* $\{g, a\}$, *i.e.* $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{g, a\})$, *and* $\mathcal{L}(\mathcal{A}, q_0) = \mathcal{L}(\mathcal{A}) = \{f(x) \mid x \in \mathcal{L}(\mathcal{A}, q_1)\}$.

• *Let* $s = f(g(q_1, f(a)))$, *and* α_1 *be the abstraction function* $\{a \mapsto q_4, f(q_4) \mapsto q_3, g(q_1, q_3) \mapsto q_2\}$. *The normalization of transition* $f(g(q_1, f(a))) \to q_0$ *with abstraction* α_1 *is the following:* $Norm_{\alpha_1}(f(g(q_1, f(a))) \to q_0) = \{f(q_2) \to q_0, g(q_1, q_3) \to q_2, f(q_4) \to q_3, a \to q_4\}$.

DEFINITION 7. *A* regular language substitution *(or a* $\mathcal{Q}$*-substitution) over an automaton* $\mathcal{A}$ *with a set of states* $\mathcal{Q}$ *is a function* $\sigma : \mathcal{X} \mapsto \mathcal{Q}$. *We can extend this definition to a morphism* $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{T}(\mathcal{F}, \mathcal{Q})$. *We denote by* $\Sigma(\mathcal{Q}, \mathcal{X})$ *the set of regular language substitutions built over* $\mathcal{Q}$ *and* $\mathcal{X}$.

DEFINITION 8 (One step automaton completion). *Let* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *be a tree automaton,* $\mathcal{R}$ *a TRS and* α *an abstraction function. The one step completed automaton* $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ *is a tree automaton* $\langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$ *such that:*

$$\Delta' = \Delta \cup \bigcup_{l \to r \in \mathcal{R},\, q \in \mathcal{Q},\, \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}),\, l\sigma \to^\star_\Delta q} Norm_\alpha(r\sigma \to q)$$

$$\mathcal{Q}' = \{q \mid c \to q \in \Delta'\}$$

DEFINITION 9 (Automaton completion). *Let* $\mathcal{A}$ *be a tree automaton,* $\mathcal{R}$ *a TRS and* α *an abstraction function.*

- $\mathcal{A}^0_{\alpha,\mathcal{R}} = \mathcal{A}$
- $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}^n_{\alpha,\mathcal{R}})$ *for* $n \in \mathbb{N}$
- $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ *is a fixpoint* $\mathcal{A}^\star_{\alpha,\mathcal{R}} = \mathcal{A}^k_{\alpha,\mathcal{R}} = \mathcal{A}^{k+1}_{\alpha,\mathcal{R}}$ *with* $k \in \mathbb{N}$

Note that $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ does not exists in general, but it can be computed in many interesting cases, provided that the α ensures termination of the completion. In the following proposition, we give some sufficient conditions for building an over-approximation automaton $\mathcal{B}$ of the set of $\mathcal{R}$-descendants of a regular language recognized by $\mathcal{A}$.

DEFINITION 10. *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be an automaton and $\mathcal{R}$ a TRS, $\mathcal{R}$ and $\mathcal{A}$ satisfy the left-coherence condition if:*

$$\forall \tau : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}),\ \forall l \to r \in \mathcal{R},\ \forall q \in \mathcal{Q} :$$

$$l\tau \to^{\star}_{\Delta} q \quad \Rightarrow \quad \exists \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}) \ \ s.t.\ \ l\tau \to^{\star}_{\Delta} l\sigma \to^{\star}_{\Delta} q$$

LEMMA 1. *Every left-linear TRS $\mathcal{R}$ and every tree automaton $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{A}$ satisfy the left-coherence condition.*

Proof. In a left-linear TRS, every left-hand side l of a rewrite rule is of the form $l = C[x_1, \ldots, x_n]$ where all the variables of $\{x_1, \ldots, x_n\}$ are distinct and there is no other variable position in C. Since $t = l\tau \to^{\star}_{\Delta} q$ there exist some terms $t_1, \ldots, t_n$ and some states $q_1, \ldots, q_n$ such that $\tau = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, $t = C[t_1, \ldots, t_n]$ and $\forall i = 1 \ldots n : t_i \to^{\star}_{\Delta} q_i$. Since all the $x_1, \ldots, x_n$ are distinct, $\sigma = \{x_1 \mapsto q_1, \ldots, x_n \mapsto q_n\}$ is a function and thus is a valid substitution such that $t \to^{\star}_{\Delta} l\sigma \to^{\star}_{\Delta} q$.

On the opposite, note that if l is not linear the relation σ we build is not necessarily a function.

This left-coherence condition is, in fact, necessary for non left-linear TRS. Roughly, the problem with non left-linear rules is the following: let $f(x, x) \to g(x)$ be a rule of $\mathcal{R}$ and let $\mathcal{A}$ be a tree automaton whose set of transitions contains $f(q_1, q_1) \to q_0$ and $f(q_2, q_3) \to q_0$. Although we can construct a valid substitution $\sigma = \{x \mapsto q_1\}$ for matching the rewrite rule on the first transition, it is not the case for the second one. The semantics of a completion between rule $f(x, x) \to g(x)$ and transition $f(q_2, q_3) \to q_0$ would be to find the common language of terms recognized both by q_2 and q_3. This can be obtained by computing a new tree automaton $\mathcal{A}'$ with a set of states $\mathcal{Q}'$ such that $\mathcal{Q}'$ is disjoint from states of $\mathcal{A}$ and $\exists q \in \mathcal{Q}' : \mathcal{L}(\mathcal{A}', q) = \mathcal{L}(\mathcal{A}, q_2) \cap \mathcal{L}(\mathcal{A}, q_3)$. Then, to end the completion step it would be enough to add transitions of $\mathcal{A}'$ to $\mathcal{A}$ with the new transition $g(q) \to q_0$.

On the other hand, one can remark that the non-linearity problem would disappear with deterministic automata since for any deterministic automaton $\mathcal{A}_{det}$ and for all states q, q' of $\mathcal{A}_{det}$ we trivially have $\mathcal{L}(\mathcal{A}, q) \cap \mathcal{L}(\mathcal{A}, q') = \emptyset$. However, determinization of a tree automaton

may result in an exponential blow-up of the number of states (Comon et al., 2002).

A solution, in between the two previous ones, is to use the *left-coherence condition* defined above by ensuring determinism for a subset of states $q \in \mathcal{Q}$ which are to be matched by the non linear variables of the non linear rules [2]. For instance, on the last example, it is enough to build the first critical pair, add the transition $g(q_1) \to q_0$, and keep q_2, q_3 deterministic, i.e. such that $\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}, q_2) \cap \mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}, q_3) = \emptyset$. We now define the condition called *simple left-coherence* which implies the left-coherence condition. Let $\mathcal{A}$ be an automaton, $l \to r$ a rewrite rule over $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\{x_1, \ldots, x_k\}$ the set of variables non linear in l and $\mathcal{Y}$ a set of variables distinct from $\mathcal{X}$. Let $Ren(l)$ be the pair (l', E) where l' denotes the term l where non linear variables are renamed and E is a set of constraints.

$$
\begin{array}{lcll}
Ren(l) & = & (l, \emptyset) & \text{if } l \text{ is either a constant or a variable that does not appear in } \{x_1, \ldots, x_k\} \\
& = & (y, \{x = y\}) & \text{if } l \text{ is a variable } x \in \{x_1, \ldots, x_k\} \text{ and } y \text{ if a fresh variable of } \mathcal{Y} \\
& = & (f(t'_1, \ldots, t'_n), \bigcup_{i=1}^n E_i) & \text{if } l = f(t_1, \ldots, t_n) \text{ and } Ren(t_i) = (t'_i, E_i) \text{ for all } i = 1 \ldots n.
\end{array}
$$

DEFINITION 11 (Simple left-coherence condition). *An automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ satisfy the* simplified left-coherence condition *if for all rules $l \to r \in \mathcal{R}$ such that $Ren(l) = (l', E)$:*

$$\forall (x = y) \in E,\ \forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}),\ \forall q, q_x, q_y \in \mathcal{Q} :$$

$$l'\sigma \to^\star_\Delta q \wedge \sigma(x) = q_x \neq q_y = \sigma(y) \implies \mathcal{L}(\mathcal{A}, q_x) \cap \mathcal{L}(\mathcal{A}, q_y) = \emptyset$$

PROPOSITION 1. *Simple left-coherence implies left-coherence.*

Proof. If $\mathcal{A}$ does not verify the left-coherence condition induced by $l \to r$, there is at least a ground term t recognized by a state q of $\mathcal{A}$, a substitution $\tau : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $t = l\tau$ and $t \to^\star_\Delta q$, and there is no $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $t \to^\star_\Delta l\sigma \wedge l\sigma \to^\star_\Delta q$. However, if t is an instance of l then t is also instance of l' the renamed

[2] This is what is called locally deterministic tree automata in (Genet and Viet Triem Tong, 2001).

version of l, let $t = l'\rho$ where $\rho : \mathcal{Y} \to \mathcal{T}(\mathcal{F})$. The problem is solved by case reasoning on t: t cannot be a variable a otherwise $l = a$ and $\mathcal{A}$ respects the coherence condition, hence t is a term of depth at least 1. Let $\rho : \{y_1, \ldots, y_k\} \mapsto \mathcal{T}(\mathcal{F})$ such that $\rho(y_j) = t_j$. If $t \to_\Delta^\star q$ then all subterm of t are recognized by $\mathcal{A}$ and there are k states $q_1, \ldots, q_k$ such that $t_j \to q_j$ for $1 \leq j \leq k$. Observing that $t_j \to q_j$ and $\rho(y_j) = t_j$, we can construct a $\mathcal{Q}$-substitution $\sigma' : \{y_1, \ldots, y_k\} \to \mathcal{Q}$ defined by $\sigma'(y_j) = q_j$. We have $t \to_\Delta^\star l'\sigma'$ then $l'\sigma' \to_\Delta^\star q$. Either we have found a $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $t \to_\Delta^\star l\sigma \wedge l\sigma \to_\Delta^\star q$ which contradicts the hypothesis or there are at least two variables y_i and y_j such that

1. $\sigma'(y_i) = q_i$ and $\sigma'(y_j) = q_j$ with $q_i \neq q_j$
2. $y_i = y_j$ is necessarily a constraint of E
3. $t_i = \rho(y_i) = \rho(y_j) = t_j$

The condition 1. and 2. hold true for at least one pair of variables (y_i, y_j) otherwise we could construct a $\mathcal{Q}$-substitution σ such that $l\sigma = l'\sigma'$. The condition 3. holds true because t is an instance of l. In that case, $\mathcal{A}$ does not verify the simple left-coherence which is a contradiction.

The following proposition states the necessary conditions on two tree automata A and B so that the language recognized by B over-approximate the set of $\mathcal{R}$-descendants of terms recognized by A.

PROPOSITION 2. *Let $\mathcal{R}$ be a TRS, $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, and $B = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$ two tree automata such that $\mathcal{R}$ and $\mathcal{B}$ satisfy the left-coherence condition. We have $\mathcal{R}^\star(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$ if*

1. *$\Delta \subseteq \Delta'$, and*
2. *$\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma \in \Sigma(\mathcal{Q}', \mathcal{X})$, $l\sigma \to_{\Delta'}^\star q$ implies $r\sigma \to_{\Delta'}^\star q$.*

Proof. By definition, any term t of $\mathcal{R}^\star(\mathcal{L}(A))$ is such that $\exists s \in \mathcal{L}(A)$ s.t. $s \to_\mathcal{R}^\star t$. By induction on the length of the derivation $s \to_\mathcal{R}^\star t$, we prove that if $s \to_\mathcal{R}^\star t$ and $s \to_{\Delta'}^\star q$ with $q \in \mathcal{Q}_f$ then $t \to_{\Delta'}^\star q$, which implies that $t \in \mathcal{L}(B)$.

1. if $t = s$ then, since $s \in \mathcal{L}(A)$, we have that $\exists q \in \mathcal{Q}_f$ s.t. $t = s \to_\Delta^\star q$. Moreover, $\Delta \subseteq \Delta'$, hence $\exists q \in \mathcal{Q}_f$ s.t. $t \to_{\Delta'}^\star q$,
2. if $s \to_\mathcal{R}^+ t$, then $\exists s' \in \mathcal{T}(\mathcal{F})$ s.t. $s \to_\mathcal{R}^\star s' \to_\mathcal{R} t$. By induction hypothesis applied to $s \to_\mathcal{R}^\star s'$, we obtain that $\exists q \in \mathcal{Q}_f$ s.t. $s' \to_{\Delta'}^\star q$.

Moreover, since $s' \to_{\mathcal{R}} t$, there exists a rule $l \to r \in \mathcal{R}$, a substitution τ, and a position p in s' such that $l\tau = s'|_p$ and $t = s'[r\tau]_p$. By construction of bottom-up tree automata with normalized transitions, if $s' \to^\star_{\Delta'} q$, then any subterm of s' is reducible by Δ' into a state of $\mathcal{Q}'$. Hence, since $l\tau = s'|_p$, we get that $\exists q' \in \mathcal{Q}'$ s.t. $l\tau \to^\star_{\Delta'} q'$ and $s'[q']_p \to^\star_{\Delta'} q$. Now, let us show that $r\tau \to^\star_{\Delta'} q'$. Let $\mathcal{V}ar(l) = \{x_1, \ldots, x_k\}$ be the variables of l. Since $\mathcal{R}$ and $\mathcal{B}$ satisfy the left-coherence condition, we get that there exists $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $l\tau \to^\star_{\Delta'} l\sigma \to^\star_{\Delta'} q'$. Hence, there exist some states $q_i \in \mathcal{Q}$ such that $\sigma = \{x_i \mapsto q_i \mid i = 1 \ldots k\}$ and $x_i\tau \to^\star_{\Delta'} q_i$ for $i = 1 \ldots k$. From $x_i\tau \to^\star_{\Delta'} q_i$ we get that $r\tau \to^\star_{\Delta'} r\sigma$. Finally, since $r\sigma \to^\star_{\Delta'} q'$, we get that $r\tau \to^\star_{\Delta'} q'$ and thus $t = s'[r\tau]_p \to^\star_{\Delta'} q$.

In this first theorem, we show that completion always over-approximate the set of descendants for TRSs and tree automata satisfying the left-coherence condition.

THEOREM 1. *Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ be a TRS and α an abstraction function. If $\mathcal{R}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ satisfy the left-coherence condition then:*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

Proof. For proving $\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) \supseteq \mathcal{R}^\star\mathcal{L}(A)$, we can use Proposition 2 with $A = \mathcal{A}$ and $B = \mathcal{A}^\star_{\alpha,\mathcal{R}} = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$. Thus, it is enough to prove that the approximation automata $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ verifies Conditions 1 and 2 of Proposition 2, for all abstraction function α. By Definition 9, $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ trivially verifies Condition 1. Now, to prove that $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ also verifies Condition 2 of Proposition 2, it is enough to prove that $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^\star_{\Delta'} q$.

Let s' be any subterm of $r\sigma$ (possibly non-strict) and $q' \in \mathcal{Q}'$. By induction on the size of s', we show that $Norm_\alpha(s' \to q') \subseteq \Delta'$ implies that $s' \to^\star_{\Delta'} q'$:

- if $s' = q'$, then we trivially have $s' \to^\star_{\Delta'} q'$.
- if $s' = q'' \in \mathcal{Q}'$ s.t. $q'' \neq q'$ then, by case 2 of definition of $Norm$, we get that $Norm_\alpha(s' \to q') = \{s' \to q'\}$. Since $Norm_\alpha(s' \to q') \subseteq \Delta'$, we have $s' \to^\star_{\Delta'} q'$.
- if $s' = g(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}')$, by applying case 3 of definition of $Norm$, we get that
  (a) $\{g(top_\alpha(t_1), \ldots, top_\alpha(t_m)) \to q'\} \subseteq \Delta'$, and
  (b) $\bigcup_{i=1}^m Norm_\alpha(t_i \to top_\alpha(t_i)) \subseteq \Delta'$,

where $\forall i = 1 \dots m$, $top_\alpha(t_i) \in \mathcal{Q}'$ by definition of $\mathcal{A}^\star_{\alpha,\mathcal{R}}$. By applying the induction hypothesis to (b), we get that $\forall i = 1 \dots m$, $t_i \to^\star_{\Delta'} top_\alpha(t_i)$. On the other hand, (a) implies that $g(top_\alpha(t_1), \dots, top_\alpha(t_m)) \to_{\Delta'} q'$. As a result, $g(t_1, \dots, t_m) \to^\star_{\Delta'} g(top_\alpha(t_1), \dots, top_\alpha(t_m)) \to_{\Delta'} q'$.

Hence $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^\star_{\Delta'} q$, and Condition 2 of Proposition 2 is satisfied by $\mathcal{A}^\star_{\alpha,\mathcal{R}}$.

4. The exact case

The aim of this part is to refine the previous result and show that some of the known regular classes of descendants can be computed using the tree automata completion algorithm and some particular abstraction functions. In a first part, we give some sufficient conditions on the abstraction function α so that completion is exact w.r.t. $\mathcal{R}^\star(E)$. Then we will see how many regular classes of the literature can be expressed using abstraction functions satisfying those conditions.

During a completion step, for a rewrite rule $l \to r$, adding a transition $r\sigma \to q$ for a regular language substitution σ such that $l\sigma \to^\star_\Delta q$ is not necessarily exact and may lead to an over-approximation of $\mathcal{R}^\star(E)$. This can be the case in particular when r is not linear. This is detailed in the following example.

EXAMPLE 3. *Let $\mathcal{R} = \{f(x) \to g(x,x)\}$ be a non right-linear TRS and let $\mathcal{A}$ be the tree automaton such that $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \to q_0, a \to q_1, b \to q_1\}$. Note that $\mathcal{A}$ is deterministic and that $\mathcal{L}(\mathcal{A}) = \{f(a), f(b)\}$ is finite. However, the completed automaton $\mathcal{A}^1_{\alpha,\mathcal{R}} = \mathcal{A}^\star_{\alpha,\mathcal{R}}$ (for any abstraction function α) has a new transition $g(q_1, q_1) \to q_0$ and the recognized language becomes $\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \{f(a), f(b), g(a,a), g(a,b), g(b,a), g(b,b)\}$, a* superset *of $\mathcal{R}^\star(\mathcal{L}(\mathcal{A})) = \{f(a), f(b), g(a,a), g(b,b)\}$.*

Note that this problem trivially disappears if every state q of the automaton recognizes exactly one term. Hence, in the previous example, on an automaton $\mathcal{A}'$ s.t. the set of final states $\mathcal{Q}_f$ is $\{q_0, q'_0\}$ and the set of transitions is $\Delta = \{f(q_1) \to q_0, f(q_2) \to q'_0, a \to q_1, b \to q_2\}$, completion would produce two transitions $g(q_1, q_1) \to q_0$ and $g(q_2, q_2) \to q'_0$ leading to an exact recognition of $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}'))$. Instead of requiring that every state should recognize exactly one term, we can somewhat relax this constraint by requiring that every state q should recognize exactly one term t and any of its $\mathcal{R}$-descendants. Thus, in example 3, if $\mathcal{R} = \{f(x) \to g(x,x), a \to b\}$ the language recognized by $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ is

still $\mathcal{L}(\mathcal{A}^{\star}_{\alpha,\mathcal{R}}) = \{f(a), f(b), g(a,a), g(a,b), g(b,a), g(b,b)\}$ but this time exactly covers $\mathcal{R}^{\star}(\mathcal{L}(\mathcal{A}))$. This weaker condition will be of interest in the following since completion steps iteratively add some terms (which are $\mathcal{R}$-descendants) in the languages recognized by every states. Hence, under some assumptions, this condition is preserved by completion steps. Finally, those restrictions can trivially be left if the TRS is right-linear. The following definition formalizes all those aspects.

DEFINITION 12 (Right-coherence condition). *A TRS $\mathcal{R}$ and tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ satisfy the right-coherence condition if*

1. *$\mathcal{R}$ is right-linear, or*
2. *$\forall q \in \mathcal{Q} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^{\star}(t)$*

Note that this condition focuses only on the *initial* automaton and not on the completed one. Hence, this condition is trivially satisfied (using the second case) by any tree automaton recognizing a finite language. This will be useful in the following theorems for defining regular classes of $\mathcal{R}^{\star}(E)$ for finite sets E.

LEMMA 2. *Let $\mathcal{R}$ be a TRS, $l \to r \in \mathcal{R}$ be a rewrite rule with $r \notin \mathcal{X}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton without dead states. Let $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ be a $\mathcal{Q}$-substitution, such that $l\sigma \to^{\star}_{\mathcal{A}} q$ with $q \in \mathcal{Q}$.*

If $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-coherence condition, for all $t \in \mathcal{T}(\mathcal{F})$ s.t. $t \to^{\star}_{\mathcal{A}} r\sigma$ then there exists $\delta : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ s.t. $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \to_{\mathcal{A}^{\star}} l\sigma \to_{\mathcal{A}^{\star}} q$ and $l\delta \to^{\star}_{\mathcal{R}} r\delta \to^{\star}_{\mathcal{R}} t$.

Proof. Let $\{p_1, \ldots, p_n\} = \mathcal{P}os_{\mathcal{X}}(r)$ and $\forall i = 1 \ldots n : x_i = r|_{p_i}$. Note that if there exist p_i and p_j s.t. $r|_{p_i} = r|_{p_j}$ then $x_i = x_j$. Let $\{y_1, \ldots, y_m\} = \mathcal{V}ar(l) \setminus \mathcal{V}ar(r)$ be the set of variables of l that do not occur in r. Note that for l it is not necessary to distinguish the multiple occurrences of non linear variables. Let $q_1, \ldots, q_n, q'_1, \ldots, q'_m \in \mathcal{Q}$ be the states such that $\sigma = \{x_1 \mapsto q_1, \ldots x_n \mapsto q_n, y_1 \mapsto q'_1, \ldots, y_m \mapsto q'_m\}$ and we obviously have $q_i = q_j$ for all $i, j \in \{1, \ldots, n\}$ such that $x_i = x_j$ since σ is a function. Thus $r = r[x_1, \ldots, x_n]$ and $r\sigma = r[q_1, \ldots, q_n]$. On the other hand, by construction of tree automata $t \to^{\star}_{\mathcal{A}} r\sigma = r[q_1, \ldots, q_n]$ implies that there exist $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ such that $t = r[t_1, \ldots, t_n]$ and $t_1 \to^{\star}_{\mathcal{A}} q_1, \ldots, t_n \to^{\star}_{\mathcal{A}} q_n$.

Now our aim is to show that there exists a substitution δ such that $l\delta \to r\delta = t$. The substitution δ we build is such that $\delta = \delta_r \cup \delta_l$ where $\mathcal{D}om(\delta_r) = \{x_1, \ldots, x_m\}$ and $\mathcal{D}om(\delta_l) = \{y_1, \ldots, y_m\}$. First we show how to construct δ_r. Let δ_r be the relation $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. Note that δ_r is a substitution (i.e. a function) if and only if there is no $i, j \in \{1, \ldots, n\}$ such that $x_i = x_j$ and $t_i \neq t_j$. This is of course trivially

the case if r is linear. Otherwise, we may have $x_i = x_j$, $t_i \rightarrow_{\mathcal{A}}^{\star} q_i$, $t_j \rightarrow_{\mathcal{A}}^{\star} q_j$, $q_i = q_j$ but $t_i \neq t_j$ and thus δ_r would not be a function. However, if condition 2. of definition 12 is satisfied then we know that $\forall i = 1 \ldots n : \exists t'_i : \mathcal{L}(\mathcal{A}, q_i) \subseteq \mathcal{R}^{\star}(\{t'_i\})$. Hence, every term which is recognized into q_i is either t'_i or one of its descendants. As a result, since $q_i = q_j$ we have that $t_i, t_j \in \mathcal{R}^{\star}(t'_i)$. In this case, if we replace t_i and t_j by t'_i in δ_r (and proceed similarly for every other occurrence of a non linear variable), we obtain a valid substitution δ_r such that $r\delta_r \rightarrow_{\mathcal{R}}{}^{\star} r[t_1, \ldots, t_n]$. Thus, using case 1. or case 2. of definition 12 leads to the same property: we have built a substitution δ_r such that $r\delta_r \rightarrow_{\mathcal{R}}{}^{\star} r[t_1, \ldots, t_n]$ and $r[t_1, \ldots, t_n] \rightarrow_{\mathcal{A}}^{\star} r[q_1, \ldots, q_n] = r\sigma$.

Now, recall that $\delta = \delta_r \cup \delta_l$. For δ_l we construct a substitution mapping the variables of l not occurring in r to any term recognized by the corresponding state in σ, i.e. $\delta_l = \bigcup_{i=1\ldots m}\{y_i \mapsto u'_i \mid \exists u'_i \in \mathcal{T}(\mathcal{F})$ s.t. $u'_i \rightarrow_{\mathcal{A}}^{\star} q'_i$ and $q'_i = \sigma(y_i)\}$. Note that the existence of u'_i s.t. $u'_i \rightarrow_{\mathcal{A}}^{\star} q'_i$ is guaranteed by the fact that q'_i is a state of $\mathcal{A}$ and there is no dead state in $\mathcal{A}$. The relation δ_l is a functional substitution and so is δ_r. Furthermore, since $\mathcal{D}om(\delta_r) \cap \mathcal{D}om(\delta_l) = \emptyset$ then δ is a substitution. Finally. we have $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \rightarrow_{\mathcal{A}}^{\star} l\sigma$ and $l\delta \rightarrow_{\mathcal{R}}^{\star} r\delta \rightarrow_{\mathcal{R}}^{\star} r[t_1, \ldots, t_n] = t$.

We now introduce *coherent abstraction function* which define some subclasses of completion algorithms for which the automaton completion algorithm is exact. Informally, an abstraction function is coherent with regards to a tree automaton $\mathcal{A}$ and a term rewriting system $\mathcal{R}$ if for every configuration t and every state q such that α maps t to q, either q is not a state of $\mathcal{A}$ (it is a new state) or terms recognized by q in $\mathcal{A}$ are either a term t' recognized by t (i.e. $t' \rightarrow_{\mathcal{A}}{}^{\star} t$) or $\mathcal{R}$-descendants of t'.

DEFINITION 13 (Coherent abstraction function). *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and α be an abstraction function. The function α is said to be* coherent *with $\mathcal{R}$ and $\mathcal{A}$ if for all $t \in \mathcal{D}om(\alpha)$, for all $q \in \mathcal{Q} \cap \mathcal{R}an(\alpha)$ if $\alpha(t) = q$ then $t \rightarrow q \in \mathcal{A}$ and there exists a term $t' \in \mathcal{T}(\mathcal{F})$ called the* representative *of q s.t. $t' \rightarrow_{\mathcal{A}}{}^{\star} t$ and $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^{\star}(\{t'\})$.*

The following lemma shows some additional properties on the coherent abstraction functions.

LEMMA 3. *Let α be an abstraction function coherent with a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$. For all $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{A}$ s.t. $top_{\alpha}(t) = q$ then we have $t \rightarrow_{\mathcal{A}}^{\star} q$ and the representative t' of q is such that $t' \rightarrow_{\mathcal{A}}^{\star} t$.*

Proof. We proceed by induction on the height of t:

- if $t = a$ a constant. From $top_\alpha(a) = q$ we get that $\alpha(a) = q$ and from definition 13, we obtain that $a \to q \in \mathcal{A}$. Furthermore, since in that case the representative t' of q is a, we trivially have $t' = a \to^\star_\mathcal{A} a$.
- if $t = f(t_1, \ldots t_n)$ then $top_\alpha(f(t_1, \ldots, t_n)) = \alpha(f(top_\alpha(t_1), \ldots, top_\alpha(t_n))) = q$. Hence, there exist states $q_1, \ldots, q_n$ such that $\forall i = 1 \ldots n : top_\alpha(t_i) = q_i$ and $\alpha(f(q_1, \ldots, q_n)) = q$. By definition 13, from $\alpha(f(q_1, \ldots, q_n)) = q$, we get that $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$ and thus $q_1, \ldots, q_n \in \mathcal{A}$. Applying the induction hypothesis on $q_1, \ldots, q_n$ and $\forall i = 1 \ldots n : top_\alpha(t_i) = q_i$ we get that $\forall i = 1 \ldots n : t_i \to^\star_\mathcal{A} q_i$ and the corresponding representatives t'_i are such that $t'_i \to^\star_\mathcal{A} t_i$. From $t_i \to^\star_\mathcal{A} q_i$ and from $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$ we get that $t = f(t_1, \ldots, t_n) \to^\star_\mathcal{A} q$. From $t'_i \to^\star_\mathcal{A} t_i$ we get that $t' = f(t'_1, \ldots, t'_n) \to^\star_\mathcal{A} f(t_1, \ldots, t_n) = t$.

The next lemma gives some sufficient conditions on a completion step to ensure that terms recognized by $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ are all reachable terms. One of the condition is that α has to be *injective* i.e. there is no couple of distinct terms $t, t' \in \mathcal{D}om(\alpha)$ such that $\alpha(t) = \alpha(t')$. The fact that the conditions used in this lemma are all necessary will be shown in the following, on examples.

LEMMA 4. *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and α be an abstraction function. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-coherence condition and if α is injective and coherent with regards to $\mathcal{R}$ and $\mathcal{A}$ then:*

$$\forall t \in \mathcal{T}(\mathcal{F}), \forall q \in \mathcal{Q} : t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q) \implies t \in \mathcal{R}^\star(\mathcal{L}(A, q))$$

Proof. For terms t such that $t \in \mathcal{L}(\mathcal{A}, q)$, we trivially have that $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$. So, we can restrict the proof to terms t such that $t \notin \mathcal{L}(\mathcal{A}, q)$. Similarly, we can distinguish two other particular cases where $t \to^\star_\Delta q$ is in fact of the form $t \to^\star_\Delta q' \to q$.

- either $q' \to q \in \mathcal{A}$ and $t \notin \mathcal{L}(\mathcal{A}, q)$. Since $q' \to q$ is in $\mathcal{A}$, we have $\mathcal{L}(\mathcal{A}, q') \subseteq \mathcal{L}(\mathcal{A}, q)$ and $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q')) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q)$. Hence, to prove that $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q)$ it is enough to prove that $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q')$ from $t \to^\star_\Delta q'$. We can proceed similarly to remove every epsilon transition of the form $q_1 \to q_2$ which are already in $\mathcal{A}$ and prove that $t \to^\star_\Delta q'$ implies that $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q')$ in the general case.

– either $t \in \mathcal{L}(\mathcal{A}, q')$, $t \notin \mathcal{L}(\mathcal{A}, q)$ and $q' \to q \in \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$. In that case, the completion step producing $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ from $\mathcal{A}$ necessarily builds a critical pair of the form $l\sigma \to_{\mathcal{R}} r\sigma = q'$ and $l\sigma \to_{\mathcal{A}^\star} q$ where $l \to r \in \mathcal{R}$. In that case, we necessarily have $l = C[x]$ and $r = x$ where $x \in \mathcal{V}ar(l)$ and $\sigma = \{x \mapsto q\} \cup \sigma'$. Hence, we have $l\sigma = C[q']\sigma' \to_{\mathcal{A}^\star} q$ and since $t \in \mathcal{L}(\mathcal{A}, q')$, we have $C[t] \to_{\mathcal{A}^\star} C[q'] \to_{\mathcal{A}^\star} q$ and thus $C[t] \in \mathcal{L}(\mathcal{A}, q)$. Finally since the rule $l \to r$ is of the form $C[x] \to x$ we get that $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$.

Now for the general case, we proceed by induction over the height of t.

– If height of t is 0 then $t = a$ where a is a constant. Since $a \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$ we have $a \to q \in \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$. Since $a \notin \mathcal{L}(\mathcal{A}, q)$ then the completion step producing $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ from $\mathcal{A}$ necessarily builds a critical pair of the form $l\sigma \to_{\mathcal{R}} a$ and $l\sigma \to_{\mathcal{A}^\star} q$ where $l \to a \in \mathcal{R}$. By lemma 2, we obtain that there exists a substitution δ such that $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \to_{\mathcal{A}^\star} q$ and $l\delta \to_{\mathcal{R}^\star} a$, hence $a \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$.

– Now, we assume that the property is true for terms of height n. Let us prove that the property also holds for terms of height $n+1$. Let t be a term of height $n+1$ such that $t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$. Let $f \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ terms of height lesser or equal to n such that $t = f(t_1, \ldots, t_n)$. By construction of tree automata, $t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$ implies that there exist states $q_1, \ldots, q_n$ such that $f(t_1, \ldots, t_n) \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} f(q_1, \ldots, q_n) \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q$. Then by cases on $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$, we obtain:

  • Assume that $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$. Then $q_1, \ldots, q_n \in \mathcal{A}$ and by induction hypothesis we get that $t_i \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q_i))$ for $i = 1 \ldots n$. Hence, there exist terms t'_i such that $t'_i \to_{\mathcal{A}^\star} q_i$ and $t'_i \to_{\mathcal{R}} t_i$. Hence $f(t'_1, \ldots t'_n) \to_{\mathcal{A}^\star} f(q_1, \ldots, q_n) \to_{\mathcal{A}} q$ and $f(t'_1, \ldots t'_n) \to_{\mathcal{R}^\star} f(t_1, \ldots, t_n)$, i.e. $f(t'_1, \ldots, t'_n) \in \mathcal{L}(\mathcal{A}, q)$ and $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$.
  • Now, assume that $f(q_1, \ldots, q_n) \to q \notin \mathcal{A}$. Thus, transition $f(q_1, \ldots, q_n) \to q$ has been added to $\mathcal{A}$ by the completion step. Hence there exists terms $t''_1, \ldots, t''_n$ such that $\forall i = 1 \ldots n$: $top_\alpha(t''_i) = q_i$ and there is either a critical pair of the form **(a)** $l\sigma \to_{\mathcal{R}} C[f(t''_1, \ldots, t''_n)]$, with $top_\alpha(f(t''_1, \ldots, t''_n)) = q$ and $l\sigma \to_{\mathcal{A}^\star} q'$ or **(b)** $l\sigma \to_{\mathcal{R}} f(t''_1, \ldots, t''_n)$ and $l\sigma \to_{\mathcal{A}^\star} q$. Let us continue the proof on those two cases:

    (a) Assume that there is a critical pair of the form $l\sigma \to_{\mathcal{R}} C[f(t''_1, \ldots, t''_n)]$ with $top_\alpha(f(t''_1, \ldots, t''_n)) = q$. From the

fact that $\{f(q_1,\ldots,q_n) \to q\} \subseteq Norm_\alpha(f(t''_1,\ldots,t''_n) \to q)$, we get that $\alpha(f(q_1,\ldots,q_n)) = q$. But, $q \in \mathcal{A}$ and since α is coherent with $\mathcal{R}$ and $\mathcal{A}$ we get that $f(q_1,\ldots,q_n) \to q \in \mathcal{A}$, which contradicts the hypothesis that $f(q_1,\ldots,q_n) \to q \notin \mathcal{A}$.

(b) Assume that there is a critical pair of the form $l\sigma \to_{\mathcal{R}} r\sigma = f(t''_1,\ldots,t''_n)$ and $l\sigma \to_{\mathcal{A}^\star} q$. First, let us prove that there exist terms $s_1,\ldots,s_n$ s.t. $\forall i = 1\ldots n : s_i \to_{\mathcal{R}^\star} t_i$ and $s_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i$. Doing so permits to prove that there exists a term $f(s_1,\ldots,s_n)$ such that $f(s_1,\ldots,s_n) \to_{\mathcal{R}^\star} f(t_1,\ldots,t_n) = t$ and $f(s_1,\ldots,s_n) \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} f(t''_1,\ldots,t''_n) = r\sigma$. By cases on $q_i \in \mathcal{A}$, we obtain:

- Assume that q_i occurs in $\mathcal{A}$. Since $top_\alpha(t''_i) = q_i$ and α is injective, we know that there exists a unique term u_i such that $\alpha(u_i) = q_i$. Furthermore, since $q_i \in \mathcal{A}$ and α coherent with $\mathcal{R}$ and $\mathcal{A}$ we get from definition 13 that $\exists s_i \in \mathcal{T}(\mathcal{F})$ s.t. $s_i \to^\star_{\mathcal{A}} u_i$ and $\mathcal{L}(\mathcal{A}, q_i) \subseteq \mathcal{R}^\star(\{s_i\})$ where s_i is the representative of q_i. On the other hand, by applying the induction hypothesis to $t_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$, we get that $t_i \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q_i))$. Combining the two, we get that $t_i \in \mathcal{R}^\star(\mathcal{R}^\star(\{s_i\}))$ i.e. $t_i \in \mathcal{R}^\star(\{s_i\})$. Thus $s_i \to_{\mathcal{R}^\star} t_i$. Now, what remains to be proved is that $\forall i = 1\ldots n : s_i \to^\star_{\mathcal{A}} t''_i$. Applying lemma 3 on $q_i \in \mathcal{A}$ where $top_\alpha(t''_i) = q_i$, α is coherent with $\mathcal{R}$ and $\mathcal{A}$, and s_i is the representative of q_i, we get that $s_i \to^\star_{\mathcal{A}} t''_i \to^\star_{\mathcal{A}} q_i$. Hence, we have $s_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$ and $s_i \to_{\mathcal{R}^\star} t_i$.
- if q_i does not occur in $\mathcal{A}$ then the only rewriting from t_i to q_i is necessarily $t_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$, since the only rewriting path to q_i, which is $t''_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$, has been added by the current completion step (α is injective). Hence $t_i \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i$. In that case let $s_i = t_i$.

Thus, we have $f(s_1,\ldots,s_n) \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} f(t''_1,\ldots,t''_n) = r\sigma$, $r\sigma \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q$ and $f(s_1,\ldots,s_n) \to_{\mathcal{R}^\star} f(t_1,\ldots,t_n) = t$. By, applying lemma 2 to term $f(s_1,\ldots,s_n) \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} r\sigma$, we get that there exists δ such that $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} l\sigma \to^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(A)} q$ and $l\delta \to_{\mathcal{R}^\star} r\delta \to_{\mathcal{R}^\star} f(s_1,\ldots,s_n)$. Let $\{x_1,\ldots,x_m\}$ be the domain of σ. For all $i = 1\ldots m$

let $q'_i = x_i\sigma$ and $u_i = \delta x_i$. Since $u_i \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q'_i)$ and u_i is a subterm of t, we can apply the induction hypothesis on u_i and we obtain that $u_i \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q'_i))$. Let v_i be a term of $\mathcal{L}(\mathcal{A}, q'_i)$ such that $v_i \to_{\mathcal{R}}{}^\star u_i$. Finally let $\delta' = \{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m\}$. We have $l\delta' \to_{\mathcal{A}}{}^\star q$, $l\delta' \to_{\mathcal{R}}{}^\star l\delta \to_{\mathcal{R}}{}^\star r\delta \to_{\mathcal{R}}{}^\star t$.

The following theorem states that under some conditions, iterating completion steps builds an automaton $\mathcal{A}^n_{\alpha,\mathcal{R}}$ recognizing only reachable terms. By extension, this is of course also true for $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ if it can be built. However, this theorem is given using $\mathcal{A}^n_{\alpha,\mathcal{R}}$ for all n because it is more general. Furthermore it can be used to under-approximate $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$ by computing $\mathcal{A}^n_{\alpha,\mathcal{R}}$ for a fixed n when completion does not terminate ($\mathcal{A}^\star_{\alpha,\mathcal{R}}$ does not exist).

THEOREM 2. *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton and α an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-coherence condition then*

$$\forall n \in \mathbb{N} : \mathcal{L}(\mathcal{A}^n_{\alpha,\mathcal{R}}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

Proof. We proceed by induction on the number of completion steps: n. If $n = 0$ we have $\mathcal{A}^0_{\alpha,\mathcal{R}} = \mathcal{A}$ and thus $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$. Then, we assume that the property holds for n completion steps and we prove that it holds for $n+1$. Let us denote by $\mathcal{B}$ the tree automaton $\mathcal{A}^1_{\alpha,\mathcal{R}} = \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$. Then, the proof is done by using the induction hypothesis on $\mathcal{B}$ since we have $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{B}^n_{\alpha,\mathcal{R}}$. By lemma 4, we know that for every state $q \in \mathcal{Q}$, and for every term $t \in \mathcal{L}(\mathcal{B}, q) = \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$ we have $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$. This property is true in particular for final states, thus we have: $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$. Now, in order to use the induction hypothesis on $\mathcal{B}$, we need to prove that $\mathcal{B}$ fulfills the conditions of the theorem, i.e. that **(a)** α is coherent with $\mathcal{R}$ and $\mathcal{B}$ and that **(b)** $\mathcal{R}$ and $\mathcal{B}$ satisfy the right-coherence condition.

(a) For every left-hand side of a normalized transition $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ such that $\alpha(t) = q$,

- if $q \in \mathcal{A}$ then we had already $t \to q \in \mathcal{A}$ so $t \to q \in \mathcal{B}$. Since α is coherent with $\mathcal{R}$ and $\mathcal{A}$, we know that $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^\star(\{t' \mid t' \to_{\mathcal{A}}{}^\star t\})$. By applying the $\mathcal{R}^\star$ operator to both sides of the previous inequality, we obtain that $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{R}^\star(\mathcal{R}^\star(\{t' \mid t' \to_{\mathcal{A}}{}^\star t\}))$. On the other hand, by lemma 4,

we get that $\mathcal{L}(\mathcal{B},q) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A},q))$. Note that $\mathcal{R}^\star(\mathcal{R}^\star(E)) = \mathcal{R}^\star(E)$ for any set E, hence we have: $\mathcal{L}(\mathcal{B},q) \subseteq \mathcal{R}^\star(\{t' \mid t' \rightarrow_{\mathcal{A}}{}^\star\ t\})$. Moreover, since $\{t' \mid t' \rightarrow_{\mathcal{A}}{}^\star\ t\} \subseteq \{t' \mid t' \rightarrow_{\mathcal{B}}^\star\ t\}$ we have $\mathcal{R}^\star(\{t' \mid t' \rightarrow_{\mathcal{A}}{}^\star\ t\}) \subseteq \mathcal{R}^\star(\{t' \mid t' \rightarrow_{\mathcal{B}}^\star\ t\})$ and by transitivity of $\subseteq$, we get that $\mathcal{L}(\mathcal{B},q) \subseteq \mathcal{R}^\star(\{t' \mid t' \rightarrow_{\mathcal{B}}^\star\ t\})$.

- if $q \notin \mathcal{A}$ but $q \in \mathcal{B}$ then q is a state that has been introduced by $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ and since α is injective we know that t is the unique left-hand side of a normalized transition s.t. $t \rightarrow_{\mathcal{B}}{}^\star\ q$. Hence, $\mathcal{L}(\mathcal{B},q) = \{t' \mid t' \rightarrow_{\mathcal{B}}{}^\star\ t\} \subseteq \mathcal{R}^\star(\{t' \mid t' \rightarrow_{\mathcal{B}}{}^\star\ t\})$.

(b) We know by hypothesis that $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-coherence condition. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the condition because $\mathcal{R}$ is right-linear then it will clearly be the case for $\mathcal{R}$ and $\mathcal{B}$. Otherwise, by hypothesis, we know that every term t' recognized by q in $\mathcal{A}$ has a common ancestor t such that $t \rightarrow_{\mathcal{R}}{}^\star\ t'$, i.e. $\forall q \in \mathcal{A} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A},q) \subseteq \mathcal{R}^\star(t)$. Now, we have to prove that it is also the case for $\mathcal{B}$.

- if $q \in \mathcal{A}$ then we know that $\exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A},q) \subseteq \mathcal{R}^\star(t)$. From lemma 4 we get that every term recognized by q in $\mathcal{B}$ has an ancestor in the terms recognized by q in $\mathcal{A}$, i.e. $\forall t_{\mathcal{B}} \in \mathcal{L}(\mathcal{B},q) : \exists t_{\mathcal{A}} \in \mathcal{L}(\mathcal{A},q)$ s.t. $t_{\mathcal{A}} \rightarrow_{\mathcal{R}}{}^\star\ t_{\mathcal{B}}$. Since, by hypothesis on $\mathcal{A}$, all the terms recognized by have a common ancestor (w.r.t $\mathcal{R}$) t , it is also the case for terms recognized by q in $\mathcal{B}$ (and it is the same ancestor t), i.e. $t \rightarrow_{\mathcal{R}}{}^\star\ t_{\mathcal{A}} \rightarrow_{\mathcal{R}}{}^\star\ t_{\mathcal{B}}$. Hence, $\forall q \in \mathcal{B} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{B},q) \subseteq \mathcal{R}^\star(t)$.
- if $q \notin \mathcal{A}$ but $q \in \mathcal{B}$ then q is a state that has been introduced by $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$. Let $s \rightarrow q'$ be the new transition whose normalization has led to construction of state q, i.e. $s = C[u]$ and $top_\alpha(u) = q$. By induction on the height of u we show that $\exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{B},q) \subseteq \mathcal{R}^\star(t)$:
  - if u is a constant, since $q \notin \mathcal{A}$ and α is injective we know that $u \rightarrow q$ is the unique transition with q on the right-hand side, hence $\mathcal{L}(\mathcal{B},q) = \{u\} \subseteq \mathcal{R}^\star(u)$.
  - if $u = f(t_1,\ldots,t_n)$ then since α is injective we know that there is a unique left-hand side of a normalized transition $f(q_1,\ldots,q_n)$ such that $f(q_1,\ldots,q_n) \rightarrow_{\mathcal{B}} q$ and $t_i \rightarrow_{\mathcal{B}}{}^\star\ q_i$ for $i = 1 \ldots n$. For every state q_i, $i = 1 \ldots n$, it is possible to find a unique term t'_i such that $\mathcal{L}(\mathcal{B},q_i) \subseteq \mathcal{R}^\star(t'_i)$. If $q_i \notin \mathcal{A}$ then we use the induction hypothesis on transition $t_i \rightarrow q_i$. Otherwise, if $q_i \in \mathcal{A}$, then the proof is similar to

the first case of the proof: by hypothesis we know that $\mathcal{L}(\mathcal{A}, q_i) \subseteq \mathcal{R}^\star(t'_i)$ and from Lemma 4, we can lift this property to $\mathcal{B}$, i.e. $\mathcal{L}(\mathcal{B}, q_i) \subseteq \mathcal{R}^\star(t'_i)$. Finally $\mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(f(t'_1, \ldots, t'_n))$

Finally, applying the induction hypothesis to $\mathcal{B}$, we get that $\mathcal{L}(\mathcal{A}^{n+1}_{\alpha,\mathcal{R}}) = \mathcal{L}(\mathcal{B}^n_{\alpha,\mathcal{R}}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{B})) \subseteq \mathcal{R}^\star(\mathcal{R}^\star(\mathcal{L}(\mathcal{A}))) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$.

THEOREM 3. *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ be a tree automaton, α be an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$.*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

if $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ exists, $\mathcal{R}$ and $\mathcal{A}$ fulfill the right-coherence condition and if $\mathcal{R}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ fulfill the left-coherence condition.

Proof. Direct consequence of theorems 2 and 1.

This theorem states the general properties of $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ but it says nothing about the existence of $\mathcal{A}^\star_{\alpha,\mathcal{R}}$, i.e. of termination of the completion. In the following, we give some interesting instances of this theorem as corollaries and some conditions for completion to terminate. The two first corollaries permits one to use automata completion as a rewriting tool: for any given finite initial language, tree automata completion produces every possible reachable term. We will show in section 8.3 that using tree automata completion in this setting provide an efficient alternative to breadth-first search for a particular descendant.

COROLLARY 1. *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton such that $\forall q \in \mathcal{Q} : \mathcal{C}ard(\mathcal{L}(\mathcal{A}, q)) = 1$, α an injective abstraction such that $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$.*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

if $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ and $\mathcal{R}$ satisfy the left-coherence condition.

Proof. Consequence of Theorem 3. Since, $\mathcal{A}$ satisfies $\forall q \in \mathcal{Q} : \mathcal{C}ard(\mathcal{L}(\mathcal{A}, q)) = 1$, the right-coherence condition is trivially fulfilled. Similarly, since $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$, α is trivially coherent with $\mathcal{R}$ and $\mathcal{A}$.

A direct consequence of this corollary is that applying completion to a tree automaton recognizing one term models exactly rewriting if α is injective and $\mathcal{R}$ is left-linear. Now, let us show that if any of the above restrictions is not satisfied then the completed automaton no longer recognizes exactly the set of reachable terms.

EXAMPLE 4 (Left-coherence condition is necessary). *Let* $\mathcal{R} = \{f(x,x) \to g(x),\ a \to b\}$, *let* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *be the tree automaton with* $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_0\}$ *and a set of transitions* $\Delta = \{f(q_1,q_2) \to q_0,\ a \to q_1,\ b \to q_2\}$. *Note that* $\mathcal{A}$ *is deterministic, it recognizes a finite language* $\mathcal{L}(\mathcal{A}) = \{f(a,b)\}$ *and it satisfies* $\forall q \in \mathcal{Q} : \mathcal{C}ard(\mathcal{L}(\mathcal{A},q)) = 1$. *However, for any abstraction function* α, *the tree automata completion produces a unique new transition:* $b \to q_1$ *and the completed automaton does not recognize term* $g(b)$ *which is a descendant of* $f(a,b)$.

EXAMPLE 5 (Unicity of initial recognized language is necessary). *Let* $\mathcal{R} = \{f(x) \to g(x,x)\}$, *let* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *be the tree automaton with* $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ *and set of transitions* $\Delta = \{f(q_1) \to q_0,\ a \to q_1,\ b \to q_1\}$. *Note that* $\mathcal{A}$ *is deterministic and recognizes a finite language* $\mathcal{L}(\mathcal{A}) = \{f(a), f(b)\}$. *However, completion with any* α *produces a new transition* $g(q_1,q_1) \to q_0$ *and thus the completed automaton recognizes terms* $g(a,b)$ *and* $g(b,a)$ *which are not valid descendants of* $f(a)$ *nor of* $f(b)$.

EXAMPLE 6 (Abstraction function needs to be injective). *Let* $\mathcal{R} = \{f(c) \to g(a,b)\}$, *let* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *be the tree automaton with* $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ *and set of transitions* $\Delta = \{f(q_1) \to q_0,\ c \to q_1\}$. *Note that* $\mathcal{A}$ *is deterministic, it recognizes a finite language* $\mathcal{L}(\mathcal{A}) = \{f(c)\}$ *and it satisfies* $\forall q \in \mathcal{Q} : \mathcal{C}ard(\mathcal{L}(\mathcal{A},q)) = 1$. *However, tree automata completion produces a new transition* $g(a,b) \to q_0$ *which has to be normalized. Now assume that* α *is a non injective function mapping* a *and* b *to the same state* q_2, *i.e.* $\alpha = \{a \mapsto q_2,\ b \mapsto q_2\}$, *then* $Norm_\alpha(g(a,b) \to q_0) = \{a \to q_2,\ b \to q_2,\ g(q_2,q_2) \to q_0\}$. *Thus, the completed automaton recognizes terms* $g(a,b)$ *which is correct but also* $g(a,a)$, $g(b,b)$ *and* $g(b,a)$ *which are not valid descendants of* $f(a)$.

As we will see in section 6 with non regular sets of descendants, using non injective abstraction functions is a very convenient way to force completion to terminate and build over-approximations.

The following corollary will be used to give alternative proofs of results for ground TRSs (Dauchet and Tison, 1990; Brainerd, 1969), linear and semi-monadic (Coquidé et al., 1991), linear and "decreasing" TRSs (Jacquemard, 1996).

COROLLARY 2. *Let* $\mathcal{R}$ *be a linear TRS,* $\mathcal{A}$ *be a tree automaton,* α *an injective abstraction such that* $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$. *If* $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ *exists then*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

Proof. Left-coherence, right-coherence and coherence of Theorem 3 are trivially satisfied.

LEMMA 5 (Termination of tree automata completion). *If $\mathcal{R}an(\alpha)$ is finite then completion terminates and the tree automaton $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ exists.*

Proof. If $\mathcal{R}an(\alpha)$ is finite then the number of new states introduced by completion is finite. If the number of new states is finite then the set of possible transitions built on $\mathcal{F}$, $\mathcal{Q}$ and on the new states is finite and thus completion necessarily terminates. Hence, $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ exists.

Note that for injective abstraction functions, proving that the range is finite is equivalent to proving that the domain is, i.e. that completion produce a finite number of distinct transitions to be normalized.

Now we give alternative algorithms and proofs of regularity of $\mathcal{R}^{\star}(E)$ for some of the classes described in section 2. For a regular language E and

$\mathcal{R}$ ground (Dauchet and Tison, 1990; Brainerd, 1969): we use corollary 2 (ground TRSs are linear) and an injective abstraction α with a finite domain $\{r|_p \mid l \to r \in \mathcal{R} \text{ and } p \in \mathcal{P}os(r) \setminus \{\epsilon\}\}$. We can restrict α to this finite domain since in every new transition $f(t_1, \ldots, t_n) \to q$ added by the completion, $f(t_1, \ldots, t_n)$ is necessarily ground and is a right-hand side of a rule of $\mathcal{R}$. So it is enough to normalize $t_1, \ldots, t_n$ and all their subterms to normalize the transition. Hence, in α for every rule $l \to r$, every strict subterm of r is mapped to a new state. Since the domain is finite, so is the range and completion terminates.

$\mathcal{R}$ right-linear and monadic (Salomaa, 1988): we use theorem 3 and an abstraction function α with an empty domain which trivially satisfy the injectivity and coherence property w.r.t. $\mathcal{R}$ and $\mathcal{A}$. The domain of α is empty since every new transition produced by the completion is of the form $f(q_1, \ldots, q_n) \to q$ where $q_1, \ldots, q_n$ are states or of the form $q \to q'$. None of these forms needs to be normalized. Assume that after each completion step, we determinize the completed automaton $\mathcal{A}^{n}_{\alpha,\mathcal{R}}$. Since the domain of α is empty, completion ends on $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ which is determinized. Thanks to determinisation of the last completion step left-coherence condition is trivially satisfied and since the TRS is right linear, this is also the case for right-coherence condition.

$\mathcal{R}$ linear and semi-monadic (Coquidé et al., 1991): as in the ground case we define α as an injective function on the finite domain: $\{r|_p \mid l \to r \in \mathcal{R} \text{ and } p \in \mathcal{P}os_{\mathcal{F}}(r)\}$. Similarly, we can restrict to this finite domain since in every new transition $f(t_1, \ldots, t_n) \to q$ added by the completion, t_i is either a ground term (and can be

normalized by a single state) or is itself a state and thus does not need normalization.

$\mathcal{R}$ linear and "decreasing" (Jacquemard, 1996): Recall that "decreasing" means that every right-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $ar(f) = n$, and $\forall i = 1, \ldots, n$, t_i is a variable, a ground term, or a term whose variables do not occur in the left-hand side. For this class, the proof and abstraction function is similar to the linear and semi-monadic case except for variables occurring in the right-hand side but not in the left-hand side. For those variables, it is enough to substitute them by a specific state $q_{\mathcal{T}(\mathcal{F})}$ (which recognize $\mathcal{T}(\mathcal{F})$) and add the set of transitions $\{f(q_{\mathcal{T}(\mathcal{F})}, \ldots, q_{\mathcal{T}(\mathcal{F})}) \to q_{\mathcal{T}(\mathcal{F})} \mid f \in \mathcal{F},\ ar(f) = n\}$ to the transitions of $\mathcal{A}$. As in the linear and semi-monadic case, we define α as an injective function on the finite domain: $\{r|_p \mid l \to r \text{ and } p \in \mathcal{P}os_{\mathcal{F}}(r)\}$ where variables in r not occurring in l are substituted by $q_{\mathcal{T}(\mathcal{F})}$.

$\mathcal{R}$ constructor based (Réty, 1999): in this particular case, there is also a restriction on the initial language $E = \{t\sigma\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})$. Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be the tree automaton recognizing E. In this particular case, our aim is more to give an alternative algorithm rather than a proof of regularity. As in (Réty, 1999), we focus on the algorithm for left and right-linear TRSs since the left-coherence restriction can be discarded using determinization of tree automata (as in the right-linear and monadic case). We use theorem 3 ($\mathcal{R}$ is linear) and an injective abstraction function α such that $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$, thus left-coherence, right-coherence and coherence are satisfied. Now, let us prove that the domain of α is finite. Let $Q_{t\sigma}$ be the finite set of states necessary to normalize deterministically $t\sigma$, Q_{arg} be the set of states necessary to normalize the ground subterms of the right-hand sides of the rules and Δ_{arg} the related set of transitions. In (Réty, 1999), it is shown that for every defined symbol of t, for every substitution $\delta : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$, for every rewriting $l\delta \to_{\mathcal{R}} r\delta$, there exists a substitution $\sigma : \mathcal{X} \mapsto Q_{t\sigma} \cup Q_{arg}$ such that $l\delta \to_{\Delta \cup \Delta_{arg}} l\sigma$ and $r\delta \to_{\Delta \cup \Delta_{arg}} r\sigma$. Hence, every critical pair encountered during the completion is of the form: $l\sigma \to_{\mathcal{A}} q$ and $l\sigma \to_{\mathcal{R}} r\sigma$ with $\sigma : \mathcal{X} \mapsto Q_{t\sigma} \cup Q_{arg}$. Since the number of defined symbols of t is finite, since $Q_{t\sigma} \cup Q_{arg}$ is finite, then so is the set of every possible critical pair and so is the domain of α.

In this last case, we did not give an explicit definition of α. The good news, and this is one of the main interest of tree automata completion

algorithm in practice, is that it is useless to define α since it can be constructed automatically *during* completion. For all the above classes, since the domain of α is finite and since α is injective, we can construct an injective α function "on-the-fly" by associating a new state to every subterm occurring during the normalization of a new transition. This leads to a fully automatic and terminating completion algorithm covering all the decidable classes we summed up here.

For building α on the fly, it is enough to start a completion with an empty abstraction function α and to create a new state $q \notin \mathcal{R}an(\alpha)$ and a new association $c \mapsto q$ in α for every new configuration c to normalize. If completion terminates (and it is necessarily the case for all decidable decidable classes we saw above) then the completed automaton $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ recognizes $\mathcal{R}^\star(E)$ if $\mathcal{R}$ is linear (or if $\mathcal{R}$ is right-linear and $\mathcal{R}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ satisfy the left-coherence condition). Note that this algorithm even covers some decidable cases that are not included in the above decidable classes. A very simple example is TRS $\mathcal{R} = \{f(s(x)) \to g(s(x)),\ g(s(x)) \to h(s(x))\}$ and initial language $E = \{f(s^\star(a))\}$. The set $\mathcal{R}^\star(E)$ is clearly regular but this example is outside of the decidable classes we saw above. However, this TRS is linear and completion terminates with an injective abstraction function built on the fly so we have a proof of regularity of $\mathcal{R}^\star(E)$ and it is recognized by the completed automaton $\mathcal{A}^\star_{\alpha,\mathcal{R}}$.

5. Extensions to the conditional case

In this section, we propose an extension of the completion algorithm for dealing with conditional term rewriting systems (CTRS for short). A natural way to compute the set of reachable terms for CTRSs is to encode CTRSs into TRS and use the tree automata completion algorithm for TRS. However, as shown in (Feuillade and Genet, 2003), a completion algorithm adapted to the specific case of CTRS is likely to give some better results in practice. This algorithm specific to the conditional case is described in this section.

A conditional term rewriting system (CTRS) over a set of ground terms $\mathcal{T}(\mathcal{F})$ is a set $\mathcal{R}$ of conditional rules $t_l \to t_r$ if *cond*, where $t_l, t_r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and *cond* designates a conjunction of conditions that must be checked before rewriting. In this paper, conditions are pairs of terms denoted by $c_1 \downarrow c_2$ where $c_1, c_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X}), (Var(c_1) \cup Var(c_2)) \subseteq Var(t_l)$; these are join conditions. Such a condition is said to be true for a substitution σ if there exists a term $u \in \mathcal{T}(\mathcal{F})$ such that $c_1\sigma$ and $c_2\sigma$ can be both rewritten by the CTRS $\mathcal{R}$ into a same term u. Then, the rewriting $t_l\sigma \to t_r\sigma$ is said to be *enabled* and can be applied to the

term $t \in \mathcal{T}(\mathcal{F})$ at position p as for a TRS. $\rightarrow_{\mathcal{R}}$ also defines a rewriting relation on $\mathcal{T}(\mathcal{F})$ and thus the set of reachable terms $\mathcal{R}^{\star}(E)$ is defined as in the non-conditional case. For the following we will only consider *atomic* join conditions, and then propose an extension to conjunction of conditions in a very simple way.

For recognizing conditions in the tree automata and compute separately their value, we will need separate states as well as the property that completion builds automata where every states (not only final ones) are closed by rewriting.

We first define a rewriting relation $t \xrightarrow[\mathcal{R}]{\downarrow n} s$ meaning that to rewrite t into s, it is necessary to evaluate at most n recursive conditions (n is called the depth of the derivation in (Dershowitz et al., 1988)).

DEFINITION 14. *For a CTRS $\mathcal{R}$ with a subset $\mathcal{R}_{nc}$ of non conditional rules, we note $\xrightarrow[\mathcal{R}]{\downarrow n}$ the relation defined by:*

- $\xrightarrow[\mathcal{R}]{\downarrow 0} = \rightarrow_{\mathcal{R}_{nc}}$
- $a \xrightarrow[\mathcal{R}]{\downarrow n+1} b \Leftrightarrow a \xrightarrow[\mathcal{R}]{\downarrow 0} b$ *or* $\exists \sigma$ *substitution*, $p \in \mathcal{P}os(a)$ *and* $(l \rightarrow r$ *if* $s \downarrow t) \in \mathcal{R}$ *such that* $a|_p = l\sigma, b = a[r\sigma]_p$ *and* $\exists u \in \mathcal{T}(\mathcal{F})$ *such that* $s\sigma \xrightarrow[\mathcal{R}]{\downarrow n\ *} u$ *and* $t\sigma \xrightarrow[\mathcal{R}]{\downarrow n\ *} u$.

Note that $l \rightarrow^{*}_{\mathcal{R}} r$ *means that* $\exists n \in \mathbb{N}$ *s.t.* $l \xrightarrow[\mathcal{R}]{\downarrow n\ *} r$.

Let $\mathcal{A}_0$ be the tree automaton to be completed using the left-linear CTRS R. Let us consider the following algorithm, where we complete at the i^{th} step the automaton $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$ to an automaton $\mathcal{A}_{i+1}$. The set of state Q_i is partitioned into three set of states: $Q_0 \cup Q_{i,new} \cup Q_{i,cond}$. Q_0 is the set of states of $\mathcal{A}_0$, $Q_{i,new}$ is a set of states produced by transition normalization and indexed by naturals, $Q_{i,cond}$ is a set of conditional states indexed by terms of $\mathcal{T}(\mathcal{F}, Q_i)$. Let α be an abstraction function. We use the following algorithm :

1. from $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$, the i^{th} step of completion, we compute the automaton $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_f, \Delta_{i+1} \rangle$ with the initialization: $\mathcal{Q}_{i+1} = \mathcal{Q}_i, \Delta_{i+1} = \Delta_i$.
2. Let us consider each *critical pair* without considering the condition of the rule. A pair (q, r) of $\mathcal{Q} \times \mathcal{T}(\mathcal{F})$ is said to be critical for a rule either non conditional $l \rightarrow r$, or conditional $l \rightarrow r$ if $c_1 \downarrow c_2$ where $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ is a regular language substitution $\sigma = \{x_1 \mapsto q_{i_1}, x_2 \mapsto q_{i_2}, \ldots, x_n \mapsto q_{i_n}\}$, where $\{x_1, x_2, \ldots, x_n\} = var(l)$, if $l\sigma \rightarrow^{*}_{\Delta_i} q$ and $r\sigma \not\rightarrow^{*}_{\Delta_i} q$.

3. for all of these critical pairs, the rule is either:
   - a conditional one: $l \to r$ if $c_1 \downarrow c_2$. There are two possibilities
     - there is no state indexed by $c_1\sigma$ or $c_2\sigma$ in the conditional subset of states of $\mathcal{Q}_i$ ($q_{c_1\sigma} \notin \mathcal{Q}_{i,cond}$ or $q_{c_2\sigma} \notin \mathcal{Q}_{i,cond}$), then we create these two states (or the one missing) and we add to the automaton $\mathcal{A}_{i+1}$ the following transitions :
     $$Norm_\alpha(c_1\sigma \to q_{c_1\sigma}) \cup Norm_\alpha(c_2\sigma \to q_{c_2\sigma})$$
     - there exist two states $q_{c_1\sigma}$ and $q_{c_2\sigma}$ in $\mathcal{Q}_i$. We have to calculate $\mathcal{L}(\mathcal{A}_i, q_{c_1\sigma}) \cap \mathcal{L}(\mathcal{A}_i, q_{c_2\sigma})$. If this set is empty, the condition is, for this completion step, considered as false. If it is not empty, then the condition is true and we go on processing the critical pair as if the rule were not conditional.
   - a non conditional one (or it is conditional and the condition has been found true in the previous step), then we add to the automaton the transitions $Norm_\alpha(r\sigma \to q)$.
4. the new automaton $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_{f,i+1}, \Delta_{i+1} \rangle$ is the result of one step of completion of $\mathcal{A}_i$.

If there exists $i \in \mathbb{N}$ such that $\mathcal{A}_i = \mathcal{A}_{i+1}$, then $\mathcal{A}_i$ is the result. Each time we add a transition to the automaton, we have to normalize it with new states (indexed by naturals and added in $Q_{i,new}$) and then we have the opportunity to make an approximation in order to limit the number of new states created for the normalization. As in the non conditional case, this completion may not have a fixed point: we may produce infinitely many new states. However, approximation techniques similar to those of section 3 and section 6 apply: let Q_{cond} be the set of new states $q_{c_1\sigma}$ and $q_{c_1\sigma}$ produced by conditions, $\mathcal{Q}_{new}$ the set of new states used to normalize the transitions, one may restrict in any way the set $\mathcal{Q}_{new}$ to force completion to terminate. Note that there is no need to limit the number of states of Q_{cond}, since the number of possible conditions c_1, c_2 is finite and the number of possible σ is finite if $\mathcal{Q}_{new}$ is.

THEOREM 4. *Let A_0 be a tree automaton such that $\mathcal{L}(\mathcal{A}_0) \supseteq E$ and $\mathcal{R}$ a left linear CTRS. If $\mathcal{A}'$ is the result of the completion of $\mathcal{A}_0$ w.r.t $\mathcal{R}$, then $\mathcal{L}(\mathcal{A}')$ is closed with respect to $\mathcal{R}$ and $\mathcal{R}^*(E) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}')$*

Proof. Let $\mathcal{A}' = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$. We prove that $\forall t \in \mathcal{T}(\mathcal{F})$ s.t. $\exists q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, $\forall u \in \mathcal{T}(\mathcal{F})$ s.t. $t \to_{\mathcal{R}}^* u$, we have $u \in \mathcal{L}(\mathcal{A}', q)$. We

prove by induction on n that $\forall n \in \mathbb{N}, q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, u s.t. $t \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} u$, then $u \in \mathcal{L}(\mathcal{A}', q)$

- If $q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, and $t \xrightarrow{\downarrow 0}{}^*_{\mathcal{R}} u$ then we trivially have $u \in \mathcal{L}(\mathcal{A}', q)$. Indeed, $\xrightarrow{\downarrow 0}{}^*_{\mathcal{R}}$ means that we consider the subset of non conditional rules of $\mathcal{R}$ and then the proof follows from theorem 1 for the automaton $\mathcal{A}'$ with q as final state.

- now suppose that for a given n: $\forall k \leq n, t \xrightarrow{\downarrow k}{}^*_{\mathcal{R}} u$ and $t \rightarrow^*_{\Delta'} q \Rightarrow u \rightarrow^*_{\Delta'} q$. We want to show that:

$$t \xrightarrow{\downarrow n+1}{}^*_{\mathcal{R}} u \text{ and } t \rightarrow^*_{\Delta'} q \Rightarrow u \rightarrow^*_{\Delta'} q$$

  $t \xrightarrow{\downarrow n+1}{}^*_{\mathcal{R}} u$ means that exists $\{t_1, t_2, \ldots, t_j\} \subseteq \mathcal{T}(\mathcal{F})$ such that

$$t_0 = t \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_1 \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_2 \xrightarrow{\downarrow n+1}_{\mathcal{R}} \ldots \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{j-1} \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_j = u$$

  Now we show that for every t_i, if $t_i \rightarrow^*_{\Delta'} q$ then $t_{i+1} \rightarrow^*_{\Delta'} q$, this leads to two cases:

  - $t_i \xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$, then using the induction hypothesis, $t_{i+1} \rightarrow^*_{\Delta'} q$.
  - $t_i \not\xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$ and $t_i \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{i+1}$, so there exists a rule $(k)\, l \rightarrow r$ if $c_1 \downarrow c_2 \in \mathcal{R}$ a closed context $C[]$, and a substitution σ such that:

$$t_i = C[l\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] = t_{i+1} \text{ if } c_1\sigma \downarrow c_2\sigma$$
$$\text{and } \exists c \text{ s.t. } c_1\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c, \text{ and } c_2\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c$$

    Since no critical pair between $\mathcal{R}$ and Δ' exists because the automaton is a fixed point for the completion, we necessarily have that $\exists q_{c_1\sigma}, q_{c_2\sigma} \in \mathcal{Q}'$. Thus, we have:

$$c_1\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c \quad \text{and } c_1\sigma \rightarrow^*_{\Delta'} q_{c_1\sigma}$$
$$c_2\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c \quad \text{and } c_1\sigma \rightarrow^*_{\Delta'} q_{c_2\sigma}$$

    The induction hypothesis leads to $c \in \mathcal{L}(\mathcal{A}', q_{c_1\sigma})$ and $c \in \mathcal{L}(\mathcal{A}', q_{c_2\sigma})$. Consequently, since $t_i \rightarrow^*_{\Delta'} q$, the condition $\mathcal{L}(\mathcal{A}', q_{c_1,\sigma}) \cap \mathcal{L}(\mathcal{A}', q_{c_2,\sigma}) \neq \emptyset$ is true, and $\mathcal{A}$ is a fixed

point for the completion for automaton $\mathcal{A}$, we necessarily have $t_{i+1} \rightarrow^*_{\Delta'} q$.

We have $t = t_0 \in \mathcal{L}(\mathcal{A}', q)$, so by induction $\forall i \leq j, t_i \in \mathcal{L}(\mathcal{A}', q)$, in particular $u = t_j$.

We get the result that $t \xrightarrow[\mathcal{R}]{\downarrow n+1^*} u$ and $t \rightarrow^*_\Delta q$ implies $u \rightarrow^*_\Delta q$
So $\forall n \in \mathbb{N}, t \xrightarrow[\mathcal{R}]{\downarrow n\ ^*} u$ and $t \rightarrow^*_{\Delta'} q$ implies $u \rightarrow^*_{\Delta'} q$, then $t \rightarrow^*_{\mathcal{R}} u$ and $t \rightarrow^*_{\Delta'} q$ implies $u \rightarrow^*_{\Delta'} q$. This leads us to $\forall q \in \mathcal{Q}', \mathcal{L}(\mathcal{A}', q)$ is closed under rewriting by $\mathcal{R}$, in particular for $q \in \mathcal{Q}_f$, thus $\mathcal{L}(\mathcal{A}')$ is closed under rewriting by $\mathcal{R}$. Since completion is incremental, we have the inequalities $\Delta \subseteq \Delta'$ and thus $E \subseteq \mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{L}(\mathcal{A}')$, and finally $\mathcal{R}^*(E) \subseteq \mathcal{L}(\mathcal{A}')$.

In order to compute the completion for a set of rules including rules with conjunction of conditions, one can modify the algorithm in the following way: when such a rule is involved in a critical pair, create a pair of new states for each atomic condition. The latter times the critical pair is encountered, the rule is enabled for the considered substitution if each related atomic condition is satisfied (according to the emptiness checking of the intersection of the conditional state languages). Then we have to normalize the right-hand term as in the non conditional case.

6. Using completion and approximations in practice

6.1. The exact case

Here is a first example showing how tree automata completion on non conditional TRSs can be used within the Timbuk tool.

EXAMPLE 7. *Let us consider the following example given in* Timbuk *syntax:*

```
Ops 0:0 s:1 plus:2 even:1 odd:1 true:0 false:0
Vars x y z
TRS R
    plus(0, x) -> x
    plus(s(x), y)-> s(plus(x, y))
    even(0) -> true
    even(s(0)) -> false
    even(s(x)) -> odd(x)
    odd(0) -> false
    odd(s(0)) -> true
    odd(s(x)) -> even(x)
```

```
Automaton A
States qf qodd qeven qpo qpe
Final States qf
Transitions
        even(qpo) -> qf                 even(qpe) -> qf
        plus(qodd, qodd) -> qpo         s(qeven) -> qodd        s(qodd) -> qeven
        plus(qeven, qeven) -> qpe       0 -> qeven

Automaton Reach
States qf
Final States qf
Transitions
        false -> qf
```

Where $\mathcal{R}$ defines the 'plus' function and the 'even' and 'odd' predicates on the naturals, and the tree automaton $\mathcal{A}$ defines the language $E = \{even(plus(t_1, t_2))\}$ where t_1, t_2 are either two even or two odd naturals. This example is in Réty's class. The language $\mathcal{R}^\star(E)$ is regular and can be automatically computed by Timbuk *within some milliseconds:*

```
Automaton current
States qnew3:0 qnew2:0 qnew1:0 qnew0:0 qf:0 qodd:0 qeven:0 qpo:0 qpe:0
Final States qf

Prior
  plus(qodd,qodd) -> qnew3      plus(qeven,qeven) -> qnew2
  plus(qodd,qeven) -> qnew0     plus(qeven,qodd) -> qnew1

Transitions
  even(qpo) -> qf               even(qpe) -> qf         plus(qodd,qodd) -> qpo
  s(qeven) -> qodd              s(qodd) -> qeven        plus(qeven,qeven) -> qpe
  0 -> qeven                    s(qnew1) -> qpo         plus(qeven,qodd) -> qnew1
  s(qodd) -> qpe                0 -> qpe                s(qnew0) -> qpe
  plus(qodd,qeven) -> qnew0     s(qeven) -> qnew1       s(qnew3) -> qnew1
  plus(qodd,qodd) -> qnew3      s(qnew2) -> qnew0       plus(qeven,qeven) -> qnew2
  true -> qf                    odd(qnew1) -> qf        odd(qodd) -> qf
  odd(qnew0) -> qf              s(qnew1) -> qnew3       0 -> qnew2
  s(qodd) -> qnew2              s(qnew0) -> qnew2       even(qeven) -> qf
  even(qnew2) -> qf             even(qnew3) -> qf
```

Note that the specific subset of transitions denoted by `Prior` *represents in this case the injective abstraction function α that has been built automatically. If we compute the intersection between $\mathcal{R}^\star(E)$ and the Reach automaton recognizing the term $false$, we obtain an empty automaton:*

```
Intersection with Reach gives (the empty automaton):

States
Final States
Transitions
```

which means that $false$ is not reachable. Thus we have proved that $even(plus(t_1, t_2))$ where t_1 and t_2 are either both even or both odd numbers cannot rewrite to $false$. However, we are still not sure that it

always rewrites to true. This can be done in the following way: Timbuk *can compute the tree automaton recognizing $IRR(\mathcal{R})$ which is:*

```
Automaton Nf
States q3:0 q2:0 q1:0 q0:0
Final States q0 q1 q2 q3

Transitions
  false -> q3           true -> q3              odd(q3) -> q3
  even(q3) -> q3        plus(q3,q3) -> q3       0 -> q2
  plus(q3,q2) -> q3     s(q3) -> q1             s(q2) -> q0
  s(q0) -> q1           s(q1) -> q1             plus(q3,q0) -> q3
  plus(q3,q1) -> q3
```

and the intersection between $\mathcal{R}^\star(E)$ and $IRR(\mathcal{R})$ gives $\mathcal{R}^!(E)$:

```
Intersection with Nf gives (not empty):
States q0:0
Final States q0
Transitions
  true -> q0
```

which means that $\mathcal{R}^!(E) = \{true\}$. Hence, every term of E necessarily rewrites to true[3].

6.2. From exactness to approximation

One of the main interest of this algorithm is the ability to switch on the fly from exact to approximate computations. Using approximation may be necessary in several contexts. When the construction of $\mathcal{R}^\star(E)$ does not converge (because $\mathcal{R}$ and E are outside of the decidable classes we discussed in section 4) approximations force completion to terminate on an automaton over-approximating $\mathcal{R}^\star(E)$. When the completion is too long ($\mathcal{R}$ and E are in a decidable class but the tree automaton recognizing $\mathcal{R}^\star(E)$ is too big) approximations permit to accelerate completion. In general completion diverges because it produces an infinite set of transitions used to recognize an infinite set of new distinct reachable terms. The idea behind approximation is to explicitly merge together (or to identify) some terms in order to limit the set of new transitions necessary to recognize them.

Merging terms together is similar to defining an equivalence class on terms. In order to merge together two terms s and t in a common equivalence class, an usual way is to use an equation $s = t$. For instance, in order to build an approximation where terms 0 and $s(0)$ are equivalent, it is enough to achieve a completion with an additional *approximation equation* $0 = s(0)$[4]. Similarly, to build an approximation where every

[3] Since $\mathcal{R}$ is terminating, every term of E has a normal form

[4] This in fact will have an even stronger effect since it will collapse together all the natural numbers.

natural number is abstracted to its parity (even or odd) it is enough to perform a completion with an additional approximation equation $s(s(x)) = x$.

In Timbuk, for applying an approximation equation $l = r$ to an automaton $\mathcal{A}$, we simply search for $\mathcal{Q}$-substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and states $q \in \mathcal{Q}$ such that $l\sigma \rightarrow_{\mathcal{A}}{}^{*} q$. Then, for every state q' different from q and such that $r\sigma \rightarrow_{\mathcal{A}}{}^{*} q'$, we merge together q and q', i.e. every occurrence of state q in $\mathcal{A}$ is renamed by q' (or q' is renamed by q since the recognized language will be the same). Now, let us give a small example of what can be done using such approximations.

EXAMPLE 8. *Let us consider the following* Timbuk *specification:*

```
Ops 0:0 s:1 plus:2 times:2 square:1 true:0 false:0 even:1 odd:1
Vars x y z
TRS R
    plus(0, x) -> x
    plus(s(x), y)-> s(plus(x, y))
    times(0, x) -> 0
    times(s(x), y) -> plus(y, times(x, y))
    square(x) -> times(x, x)

    even(0) -> true
    even(s(0)) -> false
    even(s(x)) -> odd(x)
    odd(0) -> false
    odd(s(0)) -> true
    odd(s(x)) -> even(x)

    even(square(x)) -> odd(square(s(x)))
    odd(square(x)) -> even(square(s(x)))

Automaton A0
States q0 q1 q2 q3 qf1
Final States qf1
Transitions
        0 -> q0         square(q0) -> q2        even(q2) -> qf1

Automaton Reach
States q0
Final States q0
Transitions
        false -> q0
```

With this example, we intend to prove that for all natural number n, if n is even (resp. odd) then so is n^2. The method is the following: we are going to iteratively produce terms $even(square(0))$, $odd(square(s(0)))$, $even(square(s(s(0))))$, ... by rewriting and show that all of them evaluate to 'true'. The initial language $E = \mathcal{L}(A0) = \{even(square(0))\}$ is infinitely rewritten using the two last rules of the TRS $\mathcal{R}$. Then, each of these terms can be rewritten by the definition of 'square', 'even' and 'odd' till reaching the term 'true' or 'false'. If we try

to build the abstraction function α on the fly using new states, the completion diverges. After the 9-th step of completion, $\mathcal{A}^9_{\alpha,\mathcal{R}}$ has 201 transitions and completion is still not over. At this stage and in order to force completion to terminate, it is possible to add an approximation equation:

```
Type additional equations and end by a dot '.':

s(s(x))=x.
```

and this merges together 23 states of $\mathcal{A}^9_{\alpha,\mathcal{R}}$ and it now have only 104 transitions. Then completion continues and approximation equation is applied after every step until we reach step 12 where $\mathcal{A}^{12}_{\alpha,\mathcal{R}} = \mathcal{A}^{13}_{\alpha,\mathcal{R}}$ and thus $\mathcal{A}^{12}_{\alpha,\mathcal{R}} = \mathcal{A}^{\star}_{\alpha,\mathcal{R}}$. This last automaton has only 25 transitions but it is the fixpoint. Then if we compute the intersection between $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ and the automaton Reach recognizing only the term $'false'$ we obtain an empty automaton. Thus, we have proved that for all natural number n, if n is even (resp. odd) then so is n^2.

Timbuk also provide another tool to describe approximations: *approximation rules* which are more related to automata structure and thus more precise. Approximation rules are necessary when approximation equations are not expressive or precise enough to build adequate approximations. Approximation rules also offer more control on the domain and the range of the abstraction function α and thus let the users ensure termination of the completion anytime they needs to.

The general form for approximation rules is the following: $[s \to x] \to [l_1 \to x_1, \ldots, l_n \to x_n]$ where $[s \to x]$ with $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}, \mathcal{X})$ and $x \in \mathcal{X} \cup \mathcal{Q}$ is a pattern to be matched over the new transitions $t \to q'$ obtained by completion and $[l_1 \to x_1, \ldots, l_n \to x_n]$ are rules used to normalize t. The syntactical constraint for those rules is the following: $l_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}, \mathcal{X})$ and either $x_i \in \mathcal{Q}$ or $x_i \in \mathcal{V}ar(l_i) \cup \mathcal{V}ar(s) \cup \{x\}$. To normalize a transition of the form $t \to q'$, we match s on t and x on q', obtain a substitution σ from the matching and then we normalize t with the rewrite system $\{l_1\sigma \to r_1\sigma, \ldots, l_n\sigma \to r_n\sigma\}$ where $r_1\sigma, \ldots, r_n\sigma$ are necessarily states. For example, for normalizing a transition $f(h(q_1), g(q_2)) \to q_3$ with the approximation rule $[f(x, g(y)) \to z] \to [g(u) \to z]$, we will match $f(x, g(y))$ over $f(h(q_1), g(q_2))$ and z over q_3 and this will give a substitution $\sigma = \{x \mapsto h(q_1), y \mapsto q_2, z \mapsto q_3\}$. The set of rewrite rules instantiated by σ will thus be $[g(u) \to q_3]$. Finally, using those instantiated rewrite rules on the left-hand side of the transition to normalize, i.e. $f(h(q_1), g(q_2))$ the subterm $g(q_2)$ can be rewritten into q_3. Hence, the transition $f(h(q_1), g(q_2)) \to q_3$ will be normalized into a normalized transition $g(q_2) \to q_3$ and a partially normalized transition $f(h(q_1), q_3) \to q_3$. Some examples of the use of approximation rules in practice are given in section 7.3.

7. Application examples

In this section, we present three applications of the above completion procedure, namely: proving sufficient completeness, proving strong non-termination and achieve some reachability testing.

7.1. Sufficient completeness

This property has already been investigated (Comon, 1986; Kounalis, 1985; Nipkow and Weikum, 1983; Kapur et al., 1987), in the context of algebraic specifications. We give here a definition of sufficient completeness of a TRS on a subset of the set of ground terms $E \subseteq \mathcal{T}(\mathcal{F})$.

DEFINITION 15. *A TRS $\mathcal{R}$ is sufficiently complete on $E \subseteq \mathcal{T}(\mathcal{F})$ if $\forall s \in E$, $\exists t \in \mathcal{T}(\mathcal{C})$ s.t. $s \rightarrow_{\mathcal{R}}^{*} t$, where $\mathcal{C}$ is the set of constructors in $\mathcal{F}$.*

We propose, here, to check this property thanks to the set $\mathcal{R}^{!}(E)$. Usual methods for checking this property on algebraic specifications are either based on enumeration and testing techniques (Kounalis, 1985; Nipkow and Weikum, 1983; Kapur et al., 1987) or on disunification (Comon, 1986). Note that in (Kounalis, 1985; Kapur et al., 1987), it was also shown that sufficient completeness could be proven using ground reducibility whose definition is related to $\mathcal{R}^{!}(E)$.

PROPOSITION 3. *If the TRS $\mathcal{R}$ is weakly normalizing on $E \subseteq \mathcal{T}(\mathcal{F})$, and $\mathcal{R}^{!}(E) \subseteq \mathcal{T}(\mathcal{C})$, then $\mathcal{R}$ is sufficiently complete on E.*

This comes from the fact that since $\mathcal{R}$ is weakly normalizing on E, for all terms $s \in E$, $\exists t \in IRR(\mathcal{R})$ s.t. $s \rightarrow_{\mathcal{R}}^{*} t$. Moreover, $t \in \mathcal{R}^{!}(E)$. Since $\mathcal{R}^{!}(E) \subseteq \mathcal{T}(\mathcal{C})$, we have $t \in \mathcal{T}(\mathcal{C})$.

EXAMPLE 9. *Looking back to the example 7, one can remark that $\mathcal{R}^{!}(E) = \{true\} \subseteq \mathcal{T}(\mathcal{C})$ hence, the TRS $\mathcal{R}$ is sufficiently complete on E.*

On the other hand, sufficient completeness on E does not necessarily imply that $\mathcal{R}^{!}(E) \subseteq \mathcal{T}(\mathcal{C})$. For example, let $\mathcal{R} = \{f(a) \rightarrow a, f(a) \rightarrow f(b)\}$, $\mathcal{C} = \{a, b\}$ and let $E = \{f(a)\}$. Then $\mathcal{R}$ is sufficiently complete on E, since $f(a) \rightarrow_{\mathcal{R}} a$, but $\mathcal{R}^{!}(E) = \{a, f(b)\} \not\subseteq \mathcal{T}(\mathcal{C})$. Note that using tree automata permits one to give a very precise description of the domain on which a function is complete, more precise that what can be done with simple types for instance.

7.2. Strong non-termination

DEFINITION 16 (Strong non-termination). *Let E be a set of terms and $\mathcal{R}$ be a TRS. The TRS $\mathcal{R}$ is said to be* strongly non-terminating *on E if there exists no finite $\mathcal{R}$-rewrite chains from terms of E.*

As far as we know, this property has never been defined elsewhere in the literature. We now give a way of proving it on a given initial language E and a given TRS $\mathcal{R}$.

THEOREM 5. *A TRS $\mathcal{R}$ is strongly non-terminating on E if $\mathcal{R}^!(E) = \emptyset$.*

Proof. Obvious, since $\mathcal{R}^!(E) = \emptyset$ means that every term of E is reducible, and so are every terms $\mathcal{R}$-reachable from E.

When the TRS represents some parallel processes, the strong non-termination property is close to deadlock-freeness. Let us show a very simple example of this aspect.

EXAMPLE 10. *Assume that we have two processes each one having a list of elements to count. Assume that the counter is a shared variable that should not be accessed by the two processes at the same time. Each process has two possible states 'busy' if it is accessing the shared counter or 'free' otherwise. A similar flag is associated to the shared counter in order to protect it from a concurrent access. The behavior of this system is described by the following TRS $\mathcal{R}$ where x, y, z, u are variables, Proc represents a process, cons and null are used to build the lists and S represents a configuration of the system:*

$$
\begin{array}{l}
S(Proc(free, cons(x, y)), z, free, u) \to S(Proc(busy, cons(x, y)), z, busy, u) \\
S(Proc(busy, cons(x, y)), z, busy, u) \to S(Proc(free, y), z, free, s(u)) \\
S(z, Proc(free, cons(x, y)), free, u) \to S(z, Proc(busy, cons(x, y)), busy, u) \\
S(z, Proc(busy, cons(x, y)), busy, u) \to S(z, Proc(free, y), free, s(u)) \\
S(Proc(x, null), Proc(y, null), z, u) \to S(Proc(x, null), Proc(y, null), z, u)
\end{array}
$$

The initial language E is recognized by the following tree automaton $\mathcal{A}$ whose final state is q_0 and set of transitions is:

$$
\begin{array}{lll}
S(q_1, q_1, q_2, q_3) \to q_0 & free \to q_2 & 0 \to q_3 \\
Proc(q_2, q_4) \to q_1 & null \to q_4 & cons(q_3, q_4) \to q_4
\end{array}
$$

E contains terms of the form $S(Proc(free, l_1), Proc(free, l_2), free, 0)$ where l_1 and l_2 are any lists of 0 (possibly infinite). Using an exact abstraction function α the completion does not terminate. On the 7-th completion step the completed tree automaton $\mathcal{A}^7_{\alpha,\mathcal{R}}$ contains 41

transitions. In order to make the completion terminate, we choose to add an approximation equation $s(x) = x$ *which merge some states and transition of* $\mathcal{A}^7_{\alpha,\mathcal{R}}$ *together so that the tree automaton contains only* 19 *transitions. Finally* $\mathcal{A}^8_{\alpha,\mathcal{R}} = \mathcal{A}^7_{\alpha,\mathcal{R}}$, *hence* $\mathcal{A}^\star_{\alpha,\mathcal{R}} = \mathcal{A}^7_{\alpha,\mathcal{R}}$. *The automaton* $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ *over approximating* $\mathcal{R}^\star(E)$ *is the following:*

```
Ops S:4 Proc:2 cons:2 null:0 busy:0 free:0 s:1 o:0
Automaton current
States qnew10:0 qnew9:0 qnew8:0 qnew7:0 qnew6:0 qnew5:0 qnew4:0
       qnew3:0 qnew2:0 qnew1:0 qnew0:0 q0:0 q1:0 q2:0 q3:0 q4:0
Final States q0
Prior
  s(qnew10) -> qnew10           Proc(qnew5,q4) -> qnew4         free -> qnew5
  Proc(qnew1,qnew3) -> qnew0    cons(qnew10,q4) -> qnew3        busy -> qnew1

Transitions
  S(q1,q1,q2,qnew10) -> q0         free -> q2
  o -> qnew10                      Proc(q2,q4) -> q1
  null -> q4                       cons(qnew10,q4) -> q4
  busy -> qnew1                    cons(qnew10,q4) -> qnew3
  Proc(qnew1,qnew3) -> qnew0       free -> qnew5
  Proc(qnew5,q4) -> qnew4          S(qnew0,qnew4,qnew1,qnew10) -> q0
  S(qnew0,q1,qnew1,qnew10) -> q0   S(qnew4,qnew0,qnew1,qnew10) -> q0
  S(q1,qnew0,qnew1,qnew10) -> q0   S(qnew4,q1,qnew5,qnew10) -> q0
  S(q1,qnew4,qnew5,qnew10) -> q0   S(qnew4,qnew4,qnew5,qnew10) -> q0
  s(qnew10) -> qnew10
```

Now, if we compute the intersection with $IRR(\mathcal{R})$ *we obtain an automaton over-approximating* $\mathcal{R}^!(E)$. *The automaton obtained by intersection recognizes an empty language. Hence, we also have* $\mathcal{R}^!(E) = \emptyset$ *and thus* $\mathcal{R}$ *is strongly non-terminating on* E.

7.3. Reachability

In this part, we focus on the applications of non-reachability testing, i.e. using over-approximations of $\mathcal{R}^\star(E)$ to show that $s \not\to_{\mathcal{R}}{}^* t$. The positive result (i.e. the exact cases for $\mathcal{R}^\star(E)$ which permits to show properties of the form $s \to_{\mathcal{R}}{}^* t$) of section 4 is more recent but it should quickly find some applications in theorem proving on equational theories and in verification.

Several experiments have been done on non-reachability testing for proving properties over functional programs (Genet, 1998), communicating parallel processes (Genet and Viet Triem Tong, 2001), but the most significant experiment has been done on cryptographic protocols. First, Timbuk has been used to prove secrecy and authentication properties on the Needham-Schroder Public Key Protocol which is a typical case study for verification methods (Genet and Klay, 2000). More recently it was used to prove an anti-replay property on a protocol

of the SmartRight system designed by Thomson Multimedia for digital rights management (Genet et al., 2003).

In this setting, the used TRSs are highly non-terminating and user defined approximation reveals to be a very powerful and flexible way to over-approximate the set of reachable terms. In those works, using over-approximations have led to semi-automatic proof of some properties that require induction, lemmas and user expertise when they are proved in a proof assistant.

Let us show some particular aspects of the TRSs and approximation rules used for verifying cryptographic protocols. Those protocols are supposed to be secure in an hostile environment where an intruder stores every message and every key he sees, decrypts some parts, forges new messages with the parts he has and sends every possible message in its store in order to attack some agents. We can model the intruder store using a term built with an Associative Commutative (AC) symbol `store`, where for example the term `store(a, store(store(b, a), c))` represents the multiset $\{\!\{a, a, b, c\}\!\}$. The terms `pubkey(x)`, `privkey(x)`, `encr(k, c)`, and `cons(x, y)` represent respectively the public and private key of an agent `x`, the encryption of `c` using the key `k` and a message composed of two parts `x` and `y`. We can model some of the message constructions that an intruder can do on its store, like it is done for example in (Paulson, 1997):

```
(* The intruder can encrypt any stored component with any stored key *)
store(z, pubkey(x)) -> store(encr(pubkey(x), z), store(z, pubkey(x)))
store(z, privkey(x)) -> store(encr(privkey(x), z), store(z, privkey(x)))

(* The intruder can decompose or compose any component he has *)
store(cons(x,y), m) -> store(store(cons(x,y), m),store(x, y))
store(x, y) -> store(cons(x, y), store(x,y))

(* The intruder can decrypt a message if he has the related key *)
store(encr(pubkey(x), z), privkey(x)) ->
       store(encr(pubkey(x), z), store(privkey(x), z))
store(encr(privkey(x), z), pubkey(x)) ->
       store(encr(privkey(x), z), store(pubkey(x), z))
```

The rules encoding the AC behavior of the `store` symbol are also necessary:

```
store(x, y) -> store(y, x)
store(store(x, y), z) -> store(x, store(y, z))
store(x, store(y, z)) -> store(store(x, y), z)
```

There are two particular things to remark on those rules. First, they are all non terminating, we thus have to define strong approximation rules in order to restrain divergence of the completion. Second, rules for decryption are non left-linear and we must check the left-coherence condition on the TRS and the completed tree automaton.

Now, let us show some of the approximation rules we use in this particular case. When AC symbols are simply used for representing sets of objects, a quite natural approximation rule for the `store` symbol is the following: `[store(x, y) -> z] -> [x -> z  y -> z]`. This rule normalizes every new configuration of the form `store(s, t) -> q` (where `s` and `t` are not states) into configurations `s -> q`, `t -> q` and `store(q, q) -> q`. The intuition behind this rule is that every 'subset' `x` and `y` of the store `store(x, y)` should be recognized by the same state as `store(x, y)`. Similarly the approximation rule dealing with the decryption rules is of the form: `[encr(pubkey(qA), y) -> z] -> [y -> qAsecret]` where `qA` is the state recognizing the agent `A` and `qAsecret` is a state used to recognize the language of terms protected by the public key of `A` and thus that should remain secret during the protocol execution.

Those simple approximation rules permit us to restrain the divergence of rewriting into a finite (and approximated) set of reachable terms recognized by a finite tree automaton. Then, what remains to be proved is that the completed automaton and the TRS fulfills the left-coherence condition. This can easily and automatically be checked using the simple left-coherence condition (see definition 11). During completion, ensuring this property is easy since every non left-linear variables of the TRS match agent names (like in the above rules). Hence, it is enough to build an approximation such that agent names are deterministically recognized[5] in order to ensure the simple left-coherence condition.

8. Implementing tree automata completion

In this section, we briefly present the Timbuk tool (Genet and Viet Triem Tong, 2000) in which the tree automata completion is implemented. We also discuss the efficiency of the matching algorithm over tree automata used to find critical pairs during completion. Timbuk is a tree automata library providing basic primitives on non deterministic tree automata like intersection, union, complement of languages, determinisation of tree automata, construction of $IRR(\mathcal{R})$ for left-linear TRS, as well as the tree automata completion algorithm with some tools for building abstraction function by hand or automatically. The currently distributed Timbuk 1.1 library is written in Ocaml (Leroy et al., 2000), contains nearly no specific optimization, can only build over-approximations and provide only approximation rules (see section 6)

[5] For instance by fixing $\alpha(X) = q_X$ for every agent X.

to construct abstraction functions by hand. Now, we are distributing version 2.0 which includes some improvements on the matching algorithm described in the next sections, some new automatic normalization strategies (in particular an exact one corresponding to the results of section 4) and the approximation equations facility described in section 6. Timbuk 2.0 is still written in Ocaml and is available at `http://www.irisa.fr/lande/genet/timbuk`.

Let us now present the basic matching algorithm and the optimized one. Recall that the matching problem, for a given rewriting rule $l \to r$ and a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, consists in computing all the $\mathcal{Q}$-substitutions σ such that there is a state $q \in \mathcal{Q}$ and $l\sigma \to^\star_\Delta q$.

8.1. Basic matching algorithm

This algorithm proposed in (Genet, 1997) is close to a standard matching algorithm on terms. It is defined using deduction rules over specific formulas called *matching problems*. In the following, a *matching problem* is a quantifier-free first order formula built on literals $\bot$, $s \trianglelefteq c$ where $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and closed by the connectives $\vee$ and $\wedge$. The literal $s \trianglelefteq c$ should be read as s is matched over c. An empty conjunction $\bigwedge_\emptyset$ is a trivially true matching problem.

DEFINITION 17. *Let ϕ, ϕ_1, ϕ_2 be matching problems, $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a term, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton. A* solution *to the matching problem ϕ is a $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that*

- *if $\phi = s \trianglelefteq c$, then $s\sigma \to^*_\Delta c$, or*
- *if $\phi = \phi_1 \wedge \phi_2$, then σ is a solution of ϕ_1 and a solution of ϕ_2, or*
- *if $\phi = \phi_1 \vee \phi_2$, then σ is a solution of ϕ_1 or a solution of ϕ_2.*

We assume that matching is applied to automata without epsilon-transitions which can be easily produced from a tree automaton with epsilon transitions (see section 1).

DEFINITION 18. *Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $f \in \mathcal{F}$, $ar(f) = n$, $g \in \mathcal{F}$, $ar(g) = m$, $q, q_1, \ldots, q_n \in \mathcal{Q}$, $q'_1, \ldots, q'_n \in \mathcal{Q}$, $c_1, \ldots, c_d \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $s, s_1, \ldots, s_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and ϕ_1, ϕ_2, ϕ_3 be non-empty matching problems. The matching algorithm consists in normalizing any matching problem of the form $s \trianglelefteq q$ by the following set of rules.*

Decompose $\dfrac{f(s_1,\ldots,s_n) \trianglelefteq f(q_1,\ldots,q_n)}{s_1 \trianglelefteq q_1 \wedge \ldots \wedge s_n \trianglelefteq q_n}$

Clash $\dfrac{f(s_1,\ldots,s_n) \trianglelefteq g(q'_1,\ldots,q'_m)}{\bot}$

Configuration $\dfrac{s \trianglelefteq q}{s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d \vee \bot}$

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

Moreover, after each application of any of these rules, matching problems are normalized by the following set of rules ξ*:*

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

Correctness, completeness and termination of the algorithm comes from the following theorem of (Genet, 1997).

THEOREM 6. *Given* $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, *and* $q \in \mathcal{Q}$, *every matching problem* $s \trianglelefteq q$ *has a normal form such that*

- *if it is* $\bot$ *then there is no* $\mathcal{Q}$*-substitution* σ *s.t.* $s\sigma \to^*_\Delta q$,
- *if it is empty, then for all* $\mathcal{Q}$*-substitution* σ, $s\sigma \to^*_\Delta q$,
- *otherwise, the normal form is a disjunction* $\bigvee_{i=1}^{k} \phi_i$ *s.t.* $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$, *where* $x_j^i \in \mathcal{X}$ *and* $q_j^i \in \mathcal{Q}$, *and* $\sigma_1 = \{x_j^1 \mapsto q_j^1 \mid j = 1 \ldots n_1\}, \ldots, \sigma_k = \{x_j^k \mapsto q_j^k \mid j = 1 \ldots n_k\}$ *are the only* $\mathcal{Q}$*-substitutions s.t.* $s\sigma_i \to^*_\Delta q$.

Thanks to this algorithm, for a given rule $l \to r$ and a given state q, it is possible to find every $\mathcal{Q}$-substitution σ s.t. $l\sigma \to^*_\Delta q$.

EXAMPLE 11. *Let* $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, *where* $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ *and* $\Delta = \{f(q_1) \to q_0, g(q_1) \to q_1, a \to q_1\}$. *The language* $\mathcal{L}(A) = \{f(g^*(a))\}$. *Let* $\mathcal{R} = \{f(g(x)) \to g(f(x))\}$. *If we apply matching on* $f(g(x)) \trianglelefteq q_0$, *we obtain the following deductions, where the name of the applied rule is given on the right, and normalization with simplification rules are omitted:*

$f(g(x)) \trianglelefteq q_0$	*rule* **Configuration**
$f(g(x)) \trianglelefteq f(q_1)$	*rule* **Decompose**
$g(x) \trianglelefteq q_1$	*rule* **Configuration**
$g(x) \trianglelefteq g(q_1) \ \vee \ g(x) \trianglelefteq a$	*rule* **Clash**
$g(x) \trianglelefteq g(q_1)$	*rule* **Decompose**
$x \trianglelefteq q_1$	

Let σ be the $\mathcal{Q}$-substitution $\sigma = \{x \mapsto q_1\}$. Thus, we deduced that $l\sigma = f(g(q_1)) \rightarrow_{\Delta}^{*} q_0$.

8.2. An optimized algorithm

We propose here a more efficient algorithm: we represent a rewriting system $\mathcal{R}$ with a tree automaton $\mathcal{A}_{\mathcal{R}}$, which will permit us to compute all the critical pairs between $\mathcal{R}$ and a tree automaton $\mathcal{A}$ thanks to $\mathcal{A} \cap \mathcal{A}_{\mathcal{R}}$. First, for every term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we define a tree automaton whose language is exactly $\{t\}$ using abstraction and normalization functions defined in section 3.

DEFINITION 19 (Term automaton). *Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and consider $\mathcal{S}$ the set of all the subterms of t, $\mathcal{Q}_t$ a set of states, and $\alpha : \mathcal{S} \rightarrow \mathcal{Q}_t$ an injective abstraction function. The term automaton for t is defined by $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \mathcal{Q}_t, \mathcal{Q}_{tf}, \Delta_t \rangle$ where $\mathcal{Q}_{tf} = \{top_\alpha(t)\}$ and $\Delta_t = Norm_\alpha(t \rightarrow top_\alpha(t))$.*

PROPOSITION 4. *Consider $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, α an injective abstraction and $\mathcal{A}_{\alpha,t}$ its term automaton,*

$$\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$$

Proof. The proof is an inductive reasoning over the depth of t:

- If $depth(t) = 1$ then $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \{top_\alpha(t)\}, \{top_\alpha(t)\}, \{t \rightarrow top_\alpha(t)\} \rangle$, and $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$
- Let us assume that for all terms t such that $depth(t) \leq n$ then $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$. Now, if $depth(t) = n + 1$ and $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \bigcup_{t_i \in \mathcal{S}} \{top_\alpha(t_i)\}, \{top_\alpha(t)\}, Norm_\alpha(t \rightarrow top_\alpha(t)) \rangle$, necessarily

t is of the form $f(t_1, \ldots, t_n)$. Thanks to the induction hypothesis, all subterms t_i are recognized by states $top_\alpha(t_i)$ then $t = f(t_1, \ldots, t_n) \rightarrow_\Delta f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \rightarrow_\Delta top_\alpha(t)$, and $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$.

Let us now define the substitution automaton $\mathcal{A}_\cap = \mathcal{A}_{\alpha,t} \cap \mathcal{A}$ which recognizes a set of $\mathcal{Q}$-instances which corresponds to the substitutions solutions of the matching of t on $\mathcal{A}$.

DEFINITION 20 (Substitution automaton). *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be an automaton, α an injective abstraction function and $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \mathcal{Q}_t, \mathcal{Q}_{ft}, \Delta_t \rangle$ the automaton of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we define $\mathcal{A}_\cap = \langle \mathcal{F} \cup (\mathcal{X} \times \mathcal{Q}), \mathcal{Q}_\cap, \mathcal{Q}_{f,\cap}, \Delta_\cap \rangle$ the substitution automaton of t in $\mathcal{A}$ by:*

- $\mathcal{Q}_\cap = \mathcal{Q}_t \times \mathcal{Q}$
- $\mathcal{Q}_{f,\cap} = \{\alpha(t)\} \times \mathcal{Q}$
- $\Delta_\cap =$
  $\{f((q_1, q'_1), \ldots, (q_n, q'_n)) \rightarrow (q_{n+1}, q'_{n+1}) \mid q_i \in \mathcal{Q}_t,\ q'_i \in \mathcal{Q},$
  $f(q_1, \ldots, q_n) \rightarrow q_{n+1} \in \Delta_t,\ f(q'_1, \ldots, q'_n) \rightarrow q'_{n+1} \in \Delta\}$
  $\cup \{(x, q) \rightarrow (q_x, q) \mid x \rightarrow q_x \in \Delta_t,\ q \in \mathcal{Q}\}$

DEFINITION 21 ($\mathcal{Q}$-instance). *Let $\mathcal{F}$ be an alphabet, $\mathcal{Q}$ a set of states, $\mathcal{X}$ a set of variables, we inductively define the set $\mathcal{Q}$-instances:*

- *every pair (x, q) for $x \in \mathcal{X}$ and $q \in \mathcal{Q}$ is a $\mathcal{Q}$-instance*
- *$\forall f \in \mathcal{F}$ $ar(f) = n$, if $i_1, \ldots, i_n$ are some $\mathcal{Q}$-instances then $f(i_1, \ldots, i_n)$ is also a $\mathcal{Q}$-instance.*

DEFINITION 22 ($\mathcal{Q}$-instance recognized by a substitution automaton). *Let $\mathcal{A}_\cap = \langle \mathcal{F}, \mathcal{Q}_\cap, \mathcal{Q}_{\cap f}, \Delta_\cap \rangle$ be a substitution automaton. $\mathcal{A}_\cap$ defines the set of all $\mathcal{Q}$-instances i such that there exists a final state $q \in \mathcal{Q}_{\cap f}$ verifying $i \rightarrow^\star_{\Delta_\cap} q$*

DEFINITION 23 ($\mathcal{Q}$-substitutions defined by a $\mathcal{Q}$-instance). *A $\mathcal{Q}$-instance s inductively defines a set of $\mathcal{Q}$-substitutions*

- *if $s = (x, q)$, then s defines the $\mathcal{Q}$-substitution $\{x \mapsto q\}$*
- *if $s = (a, a)$ then s defines the empty substitution*
- *if $s = f(s_1, \ldots, s_n)$, then if $\{\sigma_{i,j}\}_{i \in I_j}$ are the substitutions associated to s_j, s define the set of substitutions $\tau = \sigma_{1,i_1} \circ \ldots \circ \sigma_{n,i_n}$.*

EXAMPLE 12. *Let A_1 be the automaton for term $f(x, g(y))$. Let A_2 be the automaton recognizing the language $\{f(a, g^\star(b)), f(g^\star(b), a)$. Let A_3 be the intersection automaton ($A_\cap$). This last automaton recognize a unique $\mathcal{Q}$-instance $f((qx, q1), g((qy, q3))$ which defines the $\mathcal{Q}$-substitution $\sigma = \{x \mapsto q_1, y \mapsto q_3\}$*

A_1 with
$\mathcal{Q} = \{qx, qy, qf, qg\}$
$\mathcal{Q}_f = \{qf\}$
$\Delta =$
$$\begin{aligned} y &\to qy \\ g(qy) &\to qg \\ x &\to qx \\ f(qx, qg) &\to qf \end{aligned}$$

A_2 with
$\mathcal{Q} = \{q1, q2, q3, q4\}$
$\mathcal{Q}_f = \{q4\}$
$\Delta =$
$$\begin{aligned} a &\to q1 \\ b &\to q2 \\ g(q2) &\to q3 \\ g(q3) &\to q3 \\ f(q1, q3) &\to q4 \\ f(q3, q1) &\to q4 \end{aligned}$$

A_3 with
$\mathcal{Q} = \{qx, qy, qf, qg\} \times \{q1, q2, q3, q4\}$
$\mathcal{Q}_f = \{qf\} \times \{q1, q2, q3, q4\}$
$\Delta =$ *(we omit useless transitions)*
$$\begin{aligned} (y, q2) &\to (qy, q2) \\ (y, q3) &\to (qy, q3) \\ (x, q1) &\to (qx, q1) \\ (x, q3) &\to (qx, q3) \\ g((qy, q2)) &\to (qg, q2) \\ g((qy, q3)) &\to (qg, q3) \\ f((qx, q1), (qg, q3)) &\to (qf, q4) \\ f((qx, q3), (qg, q1)) &\to (qf, q4) \end{aligned}$$

THEOREM 7. *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be an automaton, $l \to r$ a rewriting rule, $\mathcal{A}_l$ the automaton of term l and $\mathcal{A}_\cap$ the substitution automaton, if $\mathcal{A}$ respects the left-coherence condition induced by $l \to r$ then the set of $\mathcal{Q}$-substitutions defined by all the final states $(\alpha(l), q)$ of $\mathcal{A}_\cap$ is exactly the set of $\mathcal{Q}$-substitutions $\{\sigma_i\}_{i \in I}$ such that $l\sigma_i \to^\star_\Delta q$ in $\mathcal{A}$.*

Proof.

1. First assume that $\{\sigma_i\}_{i \in I}$ is a set of $\mathcal{Q}$-substitutions defined by a term s such that $s \to^\star (\alpha(l), q)$ in $\mathcal{A}_\cap$. For all σ_i, we prove that $l\sigma_i \to^\star q$ by induction on the length of the derivations $s \to^n (\alpha(l), q)$:

   - If $s \to^1_{\mathcal{A}_\cap} (\alpha(l), q)$ then $depth(s) = 1$ and
     - either $s = (a, a)$ where a is a constant, we have $(a, a) \to_{\mathcal{A}_\cap} (\alpha(a), q)$ then $l = a$ and $l \to_\Delta q$, s defines the substitution of empty domain.
     - or $s = (x, a)$ we have also $(x, a) \to_{\mathcal{A}_\cap} (\alpha(a), q)$ then $l = x$, and s defines $\sigma_i = \{x \mapsto q\}$, we have $l\sigma = a$ and $a \to q$
   - Assume now that for all set $\{\sigma_i\}_{i \in I}$ defined by a term s of $\mathcal{A}_\cap$ verifying $s \to^k (\alpha(l), q)$ for every $k \leq n$ then $l\sigma_i \to^\star q$ in $\mathcal{A}$.
     Consider now $\{\tau_i\}_{i \in I}$ defined by a term s such that $s \to^{n+1} (\alpha(l), q)$. Clearly s is a term of the form $f(s_1, \ldots, s_n)$, where

$s_j = (l_j, q_j)$ define $\{\tau_i\}_{i \in I}$ such that $\tau_i = \sigma_{1,i_1} \circ \ldots \circ \sigma_{n,i_n}$ where σ_{j,i_j} range over the set of substitutions defined by the term s_j. $f(s_1, \ldots, s_n) \rightarrow_{\mathcal{A}_\cap} (\alpha(l), q)$ then l is of the form $f(l_1, \ldots, l_n)$ and there exist some states q_j verifying $s_j \rightarrow^k_{\mathcal{A}_\cap} (\alpha(l_j), q_j)$. We use the inductive hypothesis $s_j \rightarrow^k (\alpha(l_j, q_j)$ with $k \leq n$, then $l_j\sigma_j \rightarrow^\star q_j$ in $\mathcal{A}$ and for all τ_i there exists a combination of $\sigma_{i,j}$ verifying $l\tau_i \rightarrow f(l_1\sigma_{1,i_1}, \ldots, l_n\sigma_{n,i_n}) \rightarrow f(q_1, \ldots, q_n) \rightarrow^\star_\Delta q$

2. We use now a structural induction over l to prove that if $l\sigma \rightarrow^\star_\Delta q$ then σ correspond to a term s in $\mathcal{L}(\mathcal{A}_\cap)$:

   a) if l is a constant a
   On the one hand $a \rightarrow_{\mathcal{A}_t} \alpha(a)$ and on the other hand there exists a state q in $\mathcal{Q}$ such that $a \rightarrow_{\mathcal{A}} q$ then $(a, a) \rightarrow_{\mathcal{A}_\cap} (\alpha(a), q)$ and $(\alpha(a), q)$ is a final state of $\mathcal{A}_\cap$, $\mathcal{A}_\cap$ recognized (a, a), that defines the empty substitution.

   b) If l is a variable x
   $x \rightarrow_{\mathcal{A}_t} \alpha(x)$ and all the terms recognized by $\mathcal{A}$ may be an instance of x, we have $\{(x, q) \rightarrow (\alpha(x), q) | q \in \mathcal{Q}\} \subset \Delta_\cap$ and $\mathcal{Q}_{f,\cap} = \{\alpha(x)\} \times \mathcal{Q}$ we have defined all the $\mathcal{Q}$-substitution $\{\{x \mapsto q\} | q \in \mathcal{Q}\}$.

   c) If l is of the form $f(l_1, \ldots, l_n)$
   $\mathcal{A}$ recognized $l\sigma$ then there exist some states $q, q_1, \ldots q_n$ in $\mathcal{A}$ such that $f(q_1, \ldots, q_n) \rightarrow^\star_{\mathcal{A}} q$, $l_i\sigma \rightarrow_{\mathcal{A}} q_i$. Assume that $s = f((\alpha(l_1), q_1), \ldots, (\alpha(l_n), q_n))$, there exist q_i in $\mathcal{A}$ and q_i recognized $t_i\sigma$ then thanks to induction hypothesis for each l_i there exists s_i, $\mathcal{Q}$-instance defining σ_i. If $\sigma_1 \circ \ldots \sigma_n$ is not defined then $l\sigma$ is not recognized by $\mathcal{A}$, contradiction ; else $\sigma_1 \circ \ldots \sigma_n$ exists and $\mathcal{A}_\cap$ recognized $s = f(s_1, \ldots, s_n)$ and s define σ.

8.3. Efficiency of matching in practice

During our experiments using Timbuk for verifying cryptographic protocols, it appeared that using the optimized matching algorithm divided the computation time of the *whole* completion process by a factor 6 in the worst case and more than 10 in the best case. This optimized algorithm also restricts memory usage. Thus, in some cases where the

considered TRSs and tree automata are getting bigger, like for verifying the protocol of the SmartRight system (Genet et al., 2003), using the optimized matching algorithm permitted us to conclude whereas the completion process failed with the initial algorithm because of an highest memory usage. Those experiments have been done on several dozens of Timbuk specification files including different modelisations of cryptographic protocols.

In this part, we want to give an idea of the efficiency of the completion procedure with an optimized matching algorithm. Since as far as we know, there exists no other implementation of a similar algorithm, we chose to compare with a rewriting tool, Elan. Elan (Borovanský et al., 1998) is a very fast implementation of rewriting where rewrite rules are directly compiled into C code. Given a term rewriting system $\mathcal{R}$, we chose to use Elan and Timbuk for proving $s \to_{\mathcal{R}}^{\star} t$ by breadth-first search. The breadth-first search strategy for Elan was given by P.-E. Moreau. For using Timbuk to prove reachability starting on a single term s, we use the exact case (and Corollary 1) and automatic abstraction function construction.

EXAMPLE 13. *Let $\mathcal{R}$ be the following TRS:*

$plus(0, x) \to x$
$plus(s(x), y) \to s(plus(x, y))$
$mult(0, x) \to x$
$mult(s(x), y) \to plus(y, mult(x, y))$
$fact(0) \to s(0)$
$fact(s(x)) \to mult(s(x), fact(x))$

To check if $fact(s^4(0)) \to_{\mathcal{R}}^{*} s^{24}(0)$ Elan takes only some milliseconds where Timbuk *takes more than* 6 *seconds. Similarly, to check if $fact(s^5(0)) \to_{\mathcal{R}}^{*} s^{120}(0)$ Elan takes less than a second and* Timbuk *takes more than* 11 *minutes.*

The above example and computation time are clearly not in favor of Timbuk. However, the above TRS is terminating and confluent and thus the rewriting tree is narrow. Now let us consider another example where $\mathcal{R}$ is neither terminating nor confluent and the rewriting tree is wider.

EXAMPLE 14. *Let $\mathcal{R}$ be the following TRS:*

$f(x) \to f(s(x))$
$s(s(x)) \to f(x)$
$f(s(x)) \to f(f(x))$

To check if $f(a) \to_{\mathcal{R}}^{*} f^6(a)$ Elan takes more than 18 *seconds where* Timbuk *takes only* 1 *second for the same task. Similarly, to check if*

$f(a) \rightarrow_{\mathcal{R}}^{*} f^{8}(a)$ *Elan takes more than* 10 *minutes, where* Timbuk *takes only* 2 *seconds.*

Timbuk is clearly not as fast as Elan for rewriting but Timbuk takes advantage of tree automata structure (which provides some kind of sharing). Thus, for achieving reachability testing, when finite sets of very similar terms are rewritten, and when systems are neither confluent nor terminating, Timbuk obtains some results close or even better than those of Elan. This shows that for, reachability testing over non-terminating or non confluent term rewriting systems, the data structure used to represent sets of terms and the related matching algorithm over this structure plays a central role in the efficiency of the search algorithm. Note that the above well represent the kind of TRSs that can be encountered when using reachability testing on cryptographic protocols, for instance, where rewrite rules are highly non terminating (see section 7.3).

9. Conclusion

In this paper, we have presented some tools for dealing in practice with the reachability and the unreachability problem, i.e. given two terms s and t and a term rewriting system $\mathcal{R}$ show that $s \rightarrow_{\mathcal{R}}^{\star} t$ or on the opposite show that $s \not\rightarrow_{\mathcal{R}}^{\star} t$. Recently, unreachability proofs have successfully been applied to the verification of cryptographic protocols. We have also shown some other applications in the term rewriting domain, namely the proof of strong non-termination and the proof of sufficient completeness.

The proposed algorithm, called tree automata completion, constructs a tree automaton recognizing $\mathcal{R}^{\star}(E)$ or an over-approximation: the set of terms reachable by rewriting terms of the initial regular set E with a TRS $\mathcal{R}$. The proposed algorithm is parametrized by an abstraction function and can be adapted to several purposes.

By choosing an injective abstraction function, exactness of the algorithm is guaranteed if it terminates. A first result is that we thus have an alternative proof of regularity and an alternative algorithm for many known decidable cases of the literature. In some cases, like for the constructor system case, the resulting algorithm seems to be simpler to the initial algorithm. For more recent classes (Gyenizse and Vágvölgyi, 1998; Takai et al., 2000; Seki et al., 2002), we hope to be able to give some specific normalization strategies so as to cover exactly those classes with Timbuk like it has been done for the others.

A second result is that the regular classes of (Dauchet and Tison, 1990; Brainerd, 1969; Salomaa, 1988; Coquidé et al., 1991; Jacquemard,

1996; Réty, 1999) can uniformly be implemented by a single uniform algorithm that covers them all at the same time. As far as we know this is the only implementation for those decidable classes.

Thirdly, outside of those decidable classes when completion does not terminate, using the abstraction function as an approximation tool permits to force completion to terminate on an automaton recognizing an over-approximation of $\mathcal{R}^\star(E)$.

To sum up, the same completion algorithm is able to build exactly $\mathcal{R}^\star(E)$ when it is possible and build an over-approximation otherwise. Those techniques have been implemented in the Timbuk tool which thus permits to compute $\mathcal{R}^\star(E)$ for many of the decidable classes as well as an over-approximation otherwise. Using this prototype on practical examples has shown that efficiency of the tree automata completion algorithm strongly depends on the efficiency of the matching of left-hand side of rules on tree automata. So, we proposed an optimized algorithm for matching making it possible for Timbuk to handle completion on large TRSs or large tree automata. We also showed that resulting performances makes Timbuk usable and even more relevant than usual rewriting tools to check reachability even on finite sets of terms when dealing with non-terminating TRSs. This may be of interest for proof search in theorem provers or proof assistant when the used equational theories cannot be oriented into confluent and terminating TRSs.

Note also that since approximations are only sets of first order terms, it is also possible to use approximations to perform abstract interpretation over the theories manipulated by a proof assistant and make proof more automatic. This is what is done in (Oehl and Sinclair, 2001) for proving automatically some lemmas in Isabelle/HOL (Nipkow et al., 2002) by approximation.

The construction of a tree automaton recognizing exactly or not the set of reachable terms turns out to have several practical applications: reachability testing, sufficient completeness, proofs of strong non-termination, etc. Among all those applications, reachability testing has been successfully used for cryptographic protocol verification on some real cases.

Finally, the tree automata completion algorithm can be extended to tackle the problem of approximating reachable terms for any join conditional term rewriting system. As far as we know, this is the first time that this problem is addressed. This extension is rather natural w.r.t. the existing algorithm and uses similar techniques, in particular for approximation construction. These results suggest that some syntactic classes of CTRSs having regular sets of descendants can certainly be defined by imposing the same syntactic constraints on the right-hand

side of the rules for TRS than on every right-hand side of rules and left and right-hand side of every condition for CTRS. Like in the non conditional case, those regular classes are likely to be built using the tree automata completion algorithm for the conditional case.

Acknowledgements

We would like to thank referees and Hitoshi Ohsaki for their careful reading and their very precise and helpful comments.

References

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

Borovanský, P., C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen: 1998, 'An overview of ELAN'. In: *Proc. 2nd WRLA*. Pont-à-mousson (France), Elsevier.

Brainerd, W. S.: 1969, 'Tree Generating Regular Systems'. *Information and Control* **14**, 217–231.

Comon, H.: 1986, 'Sufficient Completeness, Term Rewriting System and Anti-Unification'. In: J. Siekmann (ed.): *Proc. 8th CADE Conf., Oxford (UK)*, Vol. 230 of *LNCS*. pp. 128–140, Springer-Verlag.

Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi: 2002, 'Tree Automata Techniques and Applications'. `http://www.grappa.univ-lille3.fr/tata/`.

Coquidé, J., M. Dauchet, R. Gilleron, and S. Vágvölgyi: 1991, 'Bottom-up Tree Pushdown Automata and Rewrite Systems'. In: R. V. Book (ed.): *Proc. 4th RTA Conf., Como (Italy)*, Vol. 488 of *LNCS*. pp. 287–298, Springer-Verlag.

Dauchet, M. and S. Tison: 1990, 'The Theory of Ground Rewrite Systems is Decidable'. In: *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*. pp. 242–248.

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

Dershowitz, N., M. Okada, and G. Sivakumar: 1988, 'Canonical conditional Rewrite Systems'. In: *Proc. 9th CADE Conf., Argonne (Ill., USA)*, Vol. 310 of *LNCS*. Springer-Verlag.

Feuillade, G. and T. Genet: 2003, 'Reachability in conditional term rewriting systems'. In: *FTP'2003, International Workshop on First-Order Theorem Proving*, Vol. 86 n. 1 of *ENTCS*. Elsevier.

Genet, T.: 1997, 'Decidable Approximations of Sets of Descendants and Sets of Normal forms (extended version)'. Technical Report RR-3325, INRIA.

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*. pp. 151–165, Springer-Verlag.

Genet, T. and F. Klay: 2000, 'Rewriting for Cryptographic Protocol Verification'. In: *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, Vol. 1831 of *LNAI*. Springer-Verlag.

Genet, T., Y.-M. Tang-Talpin, and V. Viet Triem Tong: 2003, 'Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems'. In: *In Proceedings of Workshop on Issues in the Theory of Security*.

Genet, T. and V. Viet Triem Tong: 2000, 'Timbuk 2.0 – A Tree Automata Library'. IRISA / Université de Rennes 1. `http://www.irisa.fr/lande/genet/timbuk/`.

Genet, T. and V. Viet Triem Tong: 2001, 'Reachability Analysis of Term Rewriting Systems with *Timbuk*'. In: *Proc. 8th LPAR Conf., Havana (Cuba)*, Vol. 2250 of *LNAI*. pp. 691–702, Springer-Verlag.

Gilleron, R. and S. Tison: 1995, 'Regular Tree Languages and Rewrite Systems'. *Fundamenta Informaticae* **24**, 157–175.

Gyenizse, P. and S. Vágvölgyi: 1998, 'Linear Generalized Semi-Monadic Rewrite Systems Effectively Preserve Recognizability'. *TCS* **194(1-2)**, 87–122.

Jacquemard, F.: 1996, 'Decidable Approximations of Term Rewriting Systems'. In: H. Ganzinger (ed.): *Proc. 7th RTA Conf., New Brunswick (New Jersey, USA)*. pp. 362–376, Springer-Verlag.

Kapur, D., P. Narendran, and H. Zhang: 1987, 'On sufficient completeness and related properties of term rewriting systems'. *Acta Informatica* **24**, 395–415.

Knuth, D. E. and P. B. Bendix: 1970, 'Simple word problems in universal algebras'. In: J. Leech (ed.): *Computational Problems in Abstract Algebra*. Oxford: Pergamon Press, pp. 263–297.

Kounalis, E.: 1985, 'Completeness in data type specifications'. In: B. Buchberger (ed.): *Proceedings EUROCAL Conference, Linz (Austria)*, Vol. 204 of *LNCS*. pp. 348–362, Springer-Verlag.

Leroy, X., D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon: 2000, 'The Objective Caml system release 3.00 – Documentation and user's manual'. INRIA. `http://caml.inria.fr/ocaml/htmlman/`.

Nipkow, T., L. C. Paulson, and M. Wenzel: 2002, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, Vol. 2283 of *LNCS*. Springer.

Nipkow, T. and G. Weikum: 1983, 'A decidability result about sufficient completeness of axiomatically specified abstract data types'. In: *6th GI Conference*, Vol. 145 of *LNCS*. pp. 257–268, Springer-Verlag.

Oehl, F. and D. Sinclair: 2001, 'Combining two approaches for the formal verification of cryptographic protocols'. In: *Proceedings of ICLP Workshop on Specification, Analysis and Validation for Emerging technologies in computational logic*.

Paulson, L.: 1997, 'Proving Properties of Security Protocols by Induction'. In: *10th Computer Security Foundations Workshop*. IEEE Computer Society Press.

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

Réty, P. and J. Vuotto: 2002, 'Regular Sets of Descendants by some Rewrite Strategies'. In: *Proc. 13th RTA Conf., Copenhagen (Denmark)*, Vol. 2378 of *LNCS*. Springer-Verlag.

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

Seki, H., T. Takai, Y. Fujinaka, and Y. Kaji: 2002, 'Layered transducing term rewriting system and its recognizability preserving property'. In: *Proc. 13th RTA Conf., Copenhagen (Denmark)*, Vol. 2378 of *LNCS*. Springer-Verlag.

Takai, T., Y. Kaji, and H. Seki: 2000, 'Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability'. In: *Proc. 11th RTA Conf., Norwich (UK)*, Vol. 1833 of *LNCS*. Springer-Verlag.