HAL
open science

Equational Abstraction Refinement for Certified Tree Regular Model Checking

Yohan Boichut, Benoît Boyer, Thomas Genet, Axel Legay

To cite this version:

Yohan Boichut, Benoît Boyer, Thomas Genet, Axel Legay. Equational Abstraction Refinement for Certified Tree Regular Model Checking. [Technical Report] 2010. inria-00501487v2

HAL Id: inria-00501487

https://inria.hal.science/inria-00501487v2

Submitted on 11 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Equational Abstraction Refinement for Certified Tree Regular Model Checking

Y. Boichut[1], B. Boyer[4], T. Genet[2], and A. Legay[3]

[1] LIFO - Université Orléans, France
[2] IRISA - Université Rennes 1, France
[3] INRIA - Rennes, France
[4] VERIMAG - Université Joseph Fourier, France

Abstract. *Tree Regular Model Checking* (TRMC) is the name of a family of techniques for analyzing infinite-state systems in which states are represented by trees and sets of states by tree automata. The central problem is to decide whether a set of bad states belongs to the set of reachable states. An obstacle is that this set is in general neither regular nor computable in finite time.
This paper proposes a new CounterExample Guided Abstraction Refinement (CEGAR) algorithm for TRMC. Our approach relies on a new equational-abstraction based completion algorithm to compute a regular overapproximation of the set of reachable states in finite time. This set is represented by $\mathcal{R}_{/E}$-automata, a new extended tree automaton formalism whose structure can be exploited to detect and remove false positives in an efficient manner. Our approach has been implemented in TimbukCEGAR, a new toolset that is capable of analyzing Java programs by exploiting an elegant translation from the Java byte code to term rewriting systems. Experiments show that TimbukCEGAR outperforms existing CEGAR-based completion algorithms. Contrary to existing TRMC toolsets, the answers provided by TimbukCEGAR are certified by Coq, which means that they are formally proved correct.

1 Introduction

Infinite-state models are often used to avoid potentially artificial assumptions on data structures and architectures, e.g. an artificial bound on the size of a stack or on the value of an integer variable. At the heart of most of the techniques that have been proposed for exploring infinite state spaces, is a symbolic representation that can finitely represent infinite sets of states. In this paper, we rely on Tree Regular Model Checking (TRMC) [19, 31], and assume that states of the system are represented by trees and sets of states by tree automata. The transition relation of the system is represented by a set of rewriting rules. Contrary to specific approaches that are dedicated to specific applications, TRMC is generic and expressive enough to describe a broad class of communication protocols [5], various C programs [16] with complex data structures, multi-threaded programs [34], cryptographic protocols [26, 28, 6], and Java [13].

In TRMC, the central objective is to decide whether a set of states representing some state-property belongs to the set of reachable states. An obstacle is that this set is in general neither regular nor computable in a finite time. Most existing solutions rely on computing the transitive closure of the transition relation of the systems through heuristic-based semi-algorithms [31, 5], or on the computation of some regular abstraction of the set of reachable states [19, 16]. While the first approach is precise, it is acknowledged to be ineffective on complex systems. This paper focuses on the second approach.

The first abstraction-based technique for TRMC, *Abstract Tree Regular Model Checking* (ATRMC), was proposed by Bouajjani et al [17, 15, 16]. ATRMC computes sequences of automata by successive applications of the rewriting relation to the automaton representing the initial set of states. After each computation step, techniques coming from predicate abstraction are used to over-approximate the set of reachable states. If the property holds on the abstraction, then it also holds on the concrete system. Otherwise, a counter-example is detected and the algorithm has to decide if it is a false positive or not. In case of a spurious counter-example, the algorithm refines the abstraction by backward propagation of the set of rewriting rules. The approach, which may not terminate, proceeds in a CounterExample Guided Abstraction Refinement fashion by successive abstraction/refinement until a decision can be taken. The approach has been implemented in a toolset capable, in part, to analyse C programs.

Independently, Genet et al. [24] proposed *Completion* that is another technique to compute an over-approximation of the set of reachable states. Completion exploits the structure of the term rewriting system to add new transitions in the automaton and obtain a possibly overapproximation of the set of one-step successor states. Completion leads to a direct application of rewriting rules to the automaton, while other approaches rely on possibly heavy applications of sequences of transducers to represent this step. Completion alone may not be sufficient to finitely computes the set of reachable states. A first solution to this problem is to plug one of the abstraction techniques implemented in ATRMC. However, in this paper, we prefer another solution that is to apply equational abstraction [33]. There, the merging of states is induced by a set of equations that largely exploit the structure of the system under verification and its corresponding TRS, hence leading to accurate approximations. We shall see that, initially, such equations can easily be derived from the structure of the system. Latter, they are refined automatically with our procedure without manual intervention. Completion with equational abstraction has been applied to very complex case studies such as the verification of (industrial) cryptography protocols [26, 28] and Java bytecode applications [13]. CEGAR algorithms based on equational-abstraction completion exist [11, 12], but are known to be inefficient.

In this paper, we design the first efficient and certified CEGAR framework for equational-abstraction based completion algorithm. Our approach relies on $\mathcal{R}_{/E}$-automaton, that is a new tree automaton formalism for representing sets of reachable states. In $\mathcal{R}_{/E}$-automata, equational abstraction does not merge states, but rather link them with rewriting rules labeled with equations. Such

technique is made easy by exploiting the nature of the completion step. During completion steps, such equations are propagated, and the information can used to efficiently decide whether a set of terms is reachable from the set of initial states. If the procedure concludes positively, then the term is indeed reachable. Else, one has to refine the $\mathcal{R}_{/E}$-automaton and restart the process again.

Our approach has been implemented in TimbukCEGAR. (T)RMC toolsets result from the combination of several libraries, each of them being implemented with thousands of lines of code. It is thus impossible to manually prove that those tool deliver correct answers. A particularity of TimbukCEGAR is that it is certified. In order to ensure that the whole set of reachable states has been explored, any TRMC technique needs to check whether a candidate overapproximation B is indeed a fixed point, that is if $\mathcal{L}(B) \supseteq \mathcal{R}^*(\mathcal{L}(A))$. Such check has been implemented in various TRMC toolsets, but there is no guarantee that it behaves correct. In [20], a checker for tree automata completion was designed and proved correct using the Coq [10] proof assistant. Any automaton B that passes the checker can be claimed to formally satisfy the fixed point. TimbukCEGAR implements an extension of [20] for $\mathcal{R}_{/E}$-automata, which means that the tool delivers correct answers. Our TimbukCEGAR is capable, in part, of analyzing Java programs by exploiting a elegant translation from the javabyte code to term rewriting systems. Experiments show that TimbukCEGAR outperforms existing CEGAR-based completion algorithms by orders of magnitude.

Related work. Regular Model Checking (RMC) was first applied to compute the set of reachable states of systems whose configurations are represented by words [18, 14, 22]. The approach was then extended to trees and applied to very simple case studies [5, 19]. Other regular model checking works can be found in [3, 4], where an abstraction of the transition relation allows to exploit well-quasi ordering for finite termination. Such techniques may introduce false positives; a CEGAR approach exists for the case of finite word [2], but not for the one of trees. Learning techniques apply to RMC [37, 38] but trees have not yet been considered. We mention that our work extends equational abstractions [33, 36] with counter-example detection and refinement. We mention the existence of other automata-based works that can handle a specific class of system [34]. CEGAR principles have been implemented in various tools such as Arme [35] or SLAM [8]. Those specific tools are more efficient than our approach. On the other hand, RMC and rewriting rules offers a more general framework in where the abstraction and the refinements can be computed in a systematic manner.

Structure of the paper. Section 2 introduces the basic definitions and concepts used in the paper. TRMC and Completion are introduced in Section 3. $\mathcal{R}_{/E}$-automata are introduced in Section 4. A new completion procedure is then defined in Section 5. Section 6 proposes a CEGAR approach for TRMC and Completion. Section 7 presents TimbukCEGAR. Section 8 concludes the paper and discusses future research. Due to space constraints proofs are reported to appendix.

2 Background

In this section, we introduce some definitions and concepts that will be used throughout the rest of the paper (see also [7, 21, 30]). Let $\mathcal{F}$ be a finite set of symbols, each associated with an arity function, and let $\mathcal{X}$ be a countable set of *variables*. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the set of *terms* and $\mathcal{T}(\mathcal{F})$ denotes the set of *ground terms* (terms without variables). The set of variables of a term t is denoted by $\mathcal{V}ar(t)$. A *substitution* is a function σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$, which can be uniquely extended to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A *position* p for a term t is a word over $\mathbb{N}$. The empty sequence λ denotes the top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by $\mathcal{P}os(t) = \{\lambda\}$ if $t \in \mathcal{X}$ and $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\lambda\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\}$ otherwise. If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s.

A *term rewriting system* (TRS) $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. right-linear) if each variable of l (resp. r) occurs only once in l. A TRS $\mathcal{R}$ is left-linear if every rewrite rule $l \to r$ of $\mathcal{R}$ is left-linear. The TRS $\mathcal{R}$ induces a rewriting relation $\to_{\mathcal{R}}$ on terms as follows. Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $l \to r \in \mathcal{R}$, $s \to_{\mathcal{R}} t$ denotes that there exists a position $p \in \mathcal{P}os(s)$ and a substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\to_{\mathcal{R}}^*$ and $s \to_{\mathcal{R}}^! t$ denotes that $s \to_{\mathcal{R}}^* t$ and t is irreducible by $\mathcal{R}$. The set of $\mathcal{R}$-descendants of a set of ground terms I is $\mathcal{R}^*(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to_{\mathcal{R}}^* t\}$. An *equation set* E is a set of *equations* $l = r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. For all equation $l = r \in E$ and all substitution σ, we have $l\sigma =_E r\sigma$. The relation $=_E$ is the smallest congruence such that for all substitution σ we have $l\sigma = r\sigma$. Given a TRS $\mathcal{R}$ and a set of equations E, a term $s \in \mathcal{T}(\mathcal{F})$ is rewritten modulo E into $t \in \mathcal{T}(\mathcal{F})$, denoted $s \to_{\mathcal{R}/E} t$, if there exist $s' \in \mathcal{T}(\mathcal{F})$ and $t' \in \mathcal{T}(\mathcal{F})$ such that $s =_E s' \to_{\mathcal{R}} t' =_E t$. Thus, the set of $\mathcal{R}$-descendants modulo E of a set of ground terms I is ${\mathcal{R}/E}^*(I) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in I \text{ s.t. } s \to_{\mathcal{R}/E}^* t\}$.

Let Q be a finite set of symbols with arity 0, called *states*, such that $Q \cap \mathcal{F} = \emptyset$. $\mathcal{T}(\mathcal{F} \cup Q)$ is called the set of *configurations*. A *transition* is a rewrite rule $c \to q$, where c is a configuration and q is state. A transition is *normalized* when $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$ is of arity n, and $q_1, \ldots, q_n \in Q$. A *ε-transition* is a transition of the form $q \to q'$ where q and q' are states. A bottom-up nondeterministic finite tree automaton (tree automaton for short) over the alphabet $\mathcal{F}$ is a tuple $A = \langle \mathcal{F}, Q, Q_F, \Delta \rangle$, where $Q_F \subseteq Q$, Δ is a set of normalized transitions and ε-transitions. The transitive and reflexive *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup Q)$ induced by all the transitions of A is denoted by $\to_A^*$. The tree language recognized by A in a state q is $\mathcal{L}(A, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_A^* q\}$. We define $\mathcal{L}(A) = \bigcup_{q \in Q_F} \mathcal{L}(A, q)$.

3 Tree Regular Model Checking with Completion

We first introduce *Tree Regular Model Checking* (TRMC), a tree automata based framework to represent possibly infinite-state systems. In TRMC, a program is

represented by a tuple $(\mathcal{F}, A, \mathcal{R})$, where $\mathcal{F}$ is an alphabet on which a set of terms $\mathcal{T}(\mathcal{F})$ can be defined; A is the tree automaton representing a possibly infinite set of configurations I, and $\mathcal{R}$ is a set of term rewriting rules that represent a transition relation *Rel*. We consider the following problem.

Definition 1 (Reachability Problem (RP)). *Consider a program* $(\mathcal{F}, A, \mathcal{R})$ *and a set of bad terms Bad. The Reachability Problem consists in checking whether there exists a term of* $\mathcal{R}^*(\mathcal{L}(A))$ *that belongs to Bad.*

For finite-state systems, computing the set of reachable terms $(\mathcal{R}^*(\mathcal{L}(A)))$ reduces to enumerating the terms that can be reached from the initial set of configurations. For infinite-state systems, acceleration-based methods are needed to perform this possibly infinite enumeration in a finite time. In general, such accelerations are not precise and the best one can obtain is an R-closed approximation $A^*_{\mathcal{R},E}$. A tree automaton $A^*_{\mathcal{R},E}$ is $\mathcal{R}$-closed if for all terms $s, t \in \mathcal{T}(\mathcal{F})$ such that $s \rightarrow_{\mathcal{R}} t$ and s is recognized by $A^*_{\mathcal{R},E}$ into state q then so is t. It is easy to see that if $A^*_{\mathcal{R},E}$ is $\mathcal{R}$-closed and $\mathcal{L}(A^*_{\mathcal{R},E}) \supseteq \mathcal{L}(A)$, then $\mathcal{L}(A^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(A))$. A wide range of acceleration techniques have been developed, most of them have been discussed in Section 1. Here, we focus on Completion [24], whose objective is to computes successive automata $A^0_{\mathcal{R}} = A, A^1_{\mathcal{R}}, A^2_{\mathcal{R}}, \ldots$ that represent the effect of applying the set of rewriting rules to the initial automaton. To compute infinite sets in a finite time, each completion step is eventually followed by an widening operator. More precisely, each application of $\mathcal{R}$, which is called a *completion step*, consists in searching for *critical pairs* $\langle t, q\rangle$ with $s \rightarrow_{\mathcal{R}} t$, $s \rightarrow^*_A q$ and $t \not\rightarrow^*_A q$. The idea being that the algorithm solves the critical pair by building from $A^i_{\mathcal{R}}$, a new tree automaton $A^{i+1}_{\mathcal{R}}$ with the additional transitions that represent the effect of applying $\mathcal{R}$. As the language recognized by A may be infinite, it is not possible to find all the critical pairs by enumerating the terms that it recognizes. The solution that was promoted in [24] consists in applying sets of substitutions $\sigma : \mathcal{X} \mapsto Q$ mapping variables of rewrite rules to states that represent infinite sets of (recognized) terms. Given a tree automaton $A^i_{\mathcal{R}}$ and a rewrite rule $l \rightarrow r \in \mathcal{R}$, to find all the critical pairs of $l \rightarrow r$ on $A^i_{\mathcal{R}}$, completion uses a *matching algorithm* [23] that produces the set of substitutions $\sigma : \mathcal{X} \mapsto Q$ and states $q \in Q$ such that $l\sigma \rightarrow^*_{A^i_{\mathcal{R}}} q$ and $r\sigma \not\rightarrow^*_{A^i_{\mathcal{R}}} q$. Solving critical pairs thus consists in adding new transitions: $r\sigma \rightarrow q'$ and $q' \rightarrow q$. Those new transitions may have to be *normalized* in order to satisfy the definition of transitions of tree automata (see [23] for details). As it was shown in [24], this operation may add not only new transitions but also new states to the automaton. In the rest of the paper, the completion-step operation will be represented by $\mathtt{C}$, i.e., the automaton obtained by applying the completion step to $A^i_{\mathcal{R}}$ is denoted $\mathtt{C}(A^i_{\mathcal{R}})$. Observe that when considering right-linear rewriting rules, we have that $\mathtt{C}$ is precise, i.e. it does not introduce in $A^{i+1}_{\mathcal{R}}$ terms that cannot be obtain from $A^i_{\mathcal{R}}$ by applying the set of rewriting rules. Observe also that if the system is non left-linear, then completion step may not produce all the reachable terms. Non left-linear rules will not be considered in the present paper.

The problem is that, except for specific classes of systems [23, 25], the automaton representing the set of reachable terms cannot be obtained by applying

a finite number of completion steps. The computation process thus needs to be accelerated. For doing so, we apply a *widening operator* $\mathtt{W}$ that uses a set E of equations[5] to merge states and produce a $\mathcal{R}$-closed automaton that is an over-approximation of the set of reachable terms, i.e., an automaton $A^*_{\mathcal{R},E}$ such that $\mathcal{L}(A^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(A))$. An equation $u = v$ is applied to a tree automaton A as follows: for all substitution $\sigma : \mathcal{X} \mapsto Q$ and distinct states q_1 and q_2 such that $u\sigma \to^*_A q_1$ and $v\sigma \to^*_A q_2$, states q_1 and q_2 are merged. Completion and widening steps are applied, i.e., $A^{i+1}_{\mathcal{R},E} = \mathtt{W}(\mathtt{C}(A^i_{\mathcal{R},E}))$, until a $\mathcal{R}$-closed fixpoint $A^*_{\mathcal{R},E}$ is found. Our approximation framework and methodology are close to the equational abstractions of [33]. In [27], it has been shown that, under some assumptions, the widening operator may be exact, i.e., does not add terms that are not reachable.

Example 1. Let $\mathcal{R} = \{f(x) \to f(s(s(x)))\}$ be a rewriting system, $E = \{s(s(x)) = s(x)\}$ be an equation, and $A = \langle \mathcal{F}, Q, Q_F, \Delta \rangle$ be a tree automaton with $Q_F = \{q_0\}$ and $\Delta = \{a \to q_1, f(q_1) \to q_0\}$, i.e. $\mathcal{L}(A) = \{f(a)\}$. The first completion step finds the following critical pair: $f(q_1) \to^*_A q_0$ and $f(s(s(q_1))) \not\to^*_A q_0$. Hence, the completion algorithm produces $A^1_{\mathcal{R}} = \mathtt{C}(A)$ having all transitions of A plus $\{s(q_1) \to q_2, s(q_2) \to q_3, f(q_3) \to q_4, q_4 \to q_0\}$ where q_2, q_3, q_4 are new states produced by normalization of $f(s(s(q_1))) \to q_0$. Applying $\mathtt{W}$ with the equation $s(s(x)) = s(x)$ on $A^1_{\mathcal{R}}$ is equivalent to rename q_3 into q_2. The set of transitions of $A^1_{\mathcal{R},E}$ is thus $\Delta \cup \{s(q_1) \to q_2, s(q_2) \to q_2, f(q_2) \to q_4, q_4 \to q_0\}$. Completion stops on $A^1_{\mathcal{R},E}$ that is $\mathcal{R}$-closed, and thus $A^*_{\mathcal{R},E} = A^1_{\mathcal{R},E}$.

$$\begin{array}{ccc} s(s(q_1)) & = & s(q_1) \\ {\scriptstyle A^1_{\mathcal{R}}}\downarrow{\scriptstyle *} & & {\scriptstyle *}\downarrow{\scriptstyle A^1_{\mathcal{R}}} \\ q_3 & & q_2 \end{array}$$

Observe that if the intersection between $A^*_{\mathcal{R},E}$ and Bad is not empty, then it does not necessarily mean that the system does not satisfy the property. Consider a set $Bad = \{f(s(a)), f(s(s(a)))\}$, the first term of this set is not reachable from A, but the second is. There is thus the need to successively refine the $\mathcal{R}$-closed automaton. The latter can be done by using a CounterExample Guided Abstraction Refinement algorithm (CEGAR). Developing such an algorithm for completion and equational abstraction is the objective of this paper.

4 $\mathcal{R}_{/E}$-Automata

Existing CEGAR approaches [17, 15, 16, 11] check for spurious counter examples by performing a sequence of applications of the rewriting rules to $A^*_{\mathcal{R},E}$. To avoid this potentially costly step, we suggest to replace the merging of states by the addition of new rewriting rules that carry out information on the merging through equations. Formally:

Definition 2 ($\mathcal{R}_{/E}$-automaton). *Given a TRS $\mathcal{R}$ and a set E of equations, a $\mathcal{R}_{/E}$-automaton A is a tuple $\langle \mathcal{F}, Q, Q_F, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$. Δ is a set of normalized*

[5] Those equations have to be provided by the user. In many cases, they can be produced when formalizing the problem in the TRMC framework [36]. The situation is similar for the predicates used in [17, 15, 16].

transitions. ε_E *is a set of* ε*-transitions.* $\varepsilon_{\mathcal{R}}$ *is a set of* ε*-transitions labeled by* $\top$ *or conjunctions over predicates of the form* $Eq(q, q')$ *where* $q, q' \in Q$*, and* $q \to q' \in \varepsilon_E$*.*

Set $\varepsilon_{\mathcal{R}}$ is used to distinguish a term from its successors that has been obtained by applying one or several rewriting rules. Instead of merging states according to the set of equations, A links them with epsilon transitions in ε_E. During completion step, when exploiting critical pairs, the combination of transitions in ε_E generates transition in $\varepsilon_{\mathcal{R}}$ that are labeled with a conjunction of equations representing those transitions in ε_E. In what follows, we use $\to^*_{\Delta}$ to denote the transitive and reflexive closure of Δ. Given a set Δ of normalized transitions, the set of representatives of a state q is defined by $Rep(q) = \{t \in \mathcal{T}(\mathcal{F}) | t \to^*_{\Delta} q\}$.

Definition 3 (Run of a $\mathcal{R}_{/E}$-automaton A).

- $t|_p = f(q_1, \ldots, q_n)$ *and* $f(q_1, \ldots, q_n) \to q \in \Delta$ *then* $t \xrightarrow{\top}_A t[q]_p$
- $t|_p = q$ *and* $q \to q' \in \varepsilon_E$ *then* $t \xrightarrow{Eq(q,q')}_A t[q']_p$
- $t|_p = q$ *and* $q \xrightarrow{\alpha} q' \in \varepsilon_{\mathcal{R}}$ *then* $t \xrightarrow{\alpha}_A t[q']_p$
- $u \xrightarrow{\alpha}_A v$ *and* $v \xrightarrow{\alpha'}_A w$ *then* $u \xrightarrow{\alpha \wedge \alpha'}_A w$

Theorem 1. $\forall t \in \mathcal{T}(\mathcal{F} \cup Q),\ q \in Q,\ t \xrightarrow{\alpha}_A q \iff t \to^*_A q$

A run $\xrightarrow{\alpha}$ abstracts a rewriting path of $\to_{\mathcal{R}/E}$. If $t \xrightarrow{\alpha} q$, then there exists a term $s \in Rep(q)$ such that $s \to^*_{\mathcal{R}/E} t$. The formula α denotes the subset of transitions of ε_E needed to recognize t into q.

Example 2. Let $I = f(a)$ be an initial set of terms, $\mathcal{R} = \{f(c) \to g(c),\ a \to b\}$ be a set of rewriting rules, and $E = \{b = c\}$ be a set of equations. We build A an overapproximation automaton for $\mathcal{R}^*(I)$, using E. Thanks to ε-transitions, the automaton A represented in Fig. 1 contains some information about the path used to reach terms using $\mathcal{R}$ and E. Each state has a representative term from which others are obtained. The equality $b = c$ is represented by the two transitions $q_c \to q_b$ and $q_b \to q_c$ of ε_E, taking into account that b and c are the representatives terms for states q_b and q_c, respectively. Consider now State q_c, Transition $q_b \to q_c$ indicates that the term b is obtained from Term c by using the equality. Conversely, Transition $q_c \to q_b$ leads to the conclusion that Term c is obtained

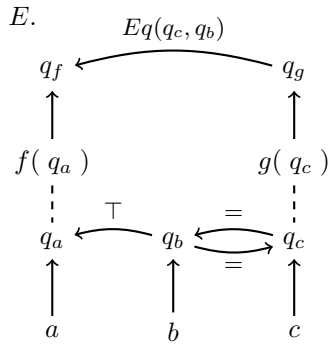


Fig. 1: Automaton A

$q_b \to q_a$ of $\varepsilon_{\mathcal{R}}$ denote rewriting steps. The transition $q_b \to q_a$ denotes that the term b is a descendant of a by rewriting. Using Definition 3, the runs $f(c) \xrightarrow{Eq(q_c,q_b)} q$ indicates that to obtain $f(c)$ from $f(a)$ – the representative term of q_f – we used the equality $b = c$, which is obtained from $q_c \to q_b$. We indeed observe $f(a) \to_{\mathcal{R}} f(b) =_E f(c)$. If we now consider the transition $q_g \to q_f$ we labeled the transition with the formula $Eq(q_c, q_b)$. To reach $g(c)$ from $f(a)$, we rewrite $f(c)$. We have seen this term is reachable thanks to the equivalence relation induced by $b = c$. By transitivity, this equivalence is also used to reach the term $g(c)$. We thus label the transition of $\varepsilon_{\mathcal{R}}$ to save this information. We obtain the run $g(c) \xrightarrow{Eq(q_c,q_b)} q_f$. We observe that the transition $q_b \to q_a$ is labeled by the formula $\top$ since b is reachable from a without any equivalence. By congruence, so is $f(b)$ from $f(a)$. The run $f(b) \xrightarrow{\top} q_f$ denotes it.

We now introduce a property that will be used in the refinement procedure to distinguish between counter-examples and false positives.

Definition 4 (A well-defined $\mathcal{R}_{/E}$-automaton). *A is a* well-defined $\mathcal{R}_{/E}$-*automaton, if :*

- *For all state q of A, and all term v such that $v \xrightarrow{\top}_A q$, there exists u a term representative of q such that $u \to^*_{\mathcal{R}} v$*
- *If $q \xrightarrow{\phi} q'$ is a transition of $\varepsilon_{\mathcal{R}}$, then there exist terms $s, t \in \mathcal{T}(\mathcal{F})$ such that $s \xrightarrow{\phi}_A q$, $t \xrightarrow{\top}_A q'$ and $t \to_{\mathcal{R}} s$.*

The first item in Definition 4 guarantees that every term recognized by using transitions labeled with the formula $\top$ is indeed reachable from the initial set. The second item is used to refine the automaton. A rewriting step of $\to_{\mathcal{R}/E}$ denoted by $q \xrightarrow{\phi} q'$ holds thanks to some transitions of ε_E that occurs in ϕ. If we remove transitions in ε_E in such a way that ϕ does not hold, then the transition $q \xrightarrow{\phi} q'$ should also be removed.

According to the above construction, a term t that is recognized by using at least a transition labeled with a formula different from $\top$ can be removed from the language of the $\mathcal{R}_{/E}$-automaton by removing some transitions in ε_E. This "pruning" operation will be detailed in Section 6.

5 Solving the Reachability Problem with $\mathcal{R}_{/E}$-automaton

In this section, we extend the completion and widening principles introduced in Section 3 to take advantage of the structure of $\mathcal{R}_{/E}$−automata. We consider an initial set I that can be represented by a tree automaton $A^0_{\mathcal{R},E} = \langle \mathcal{F}, Q^0, Q_F, \Delta^0 \rangle$, and transition relation represented by a set of linear rewriting rules $\mathcal{R}$. In the next section, we will see that the right-linearity condition may be relaxed using additionnal hypotheses. We compute successive approximations $A^i_{\mathcal{R},E} = \langle \mathcal{F}, Q^i, Q_f, \Delta^i \cup \varepsilon^i_{\mathcal{R}} \cup \varepsilon^i_E \rangle$ from $A^0_{\mathcal{R},E}$ using $A^{i+1}_{\mathcal{R},E} = \mathtt{W}(\mathtt{C}(A^i_{\mathcal{R},E}))$. Observe that $A^0_{\mathcal{R},E}$ is well-defined as the sets $\varepsilon^0_{\mathcal{R}}$ and ε^0_E are empty.

5.1 The Completion step $\mathtt{C}$

Extending completion to $\mathcal{R}_{/E}$-automaton requires to modify the concept of critical pair and so the algorithm to compute them. A critical pair for a $\mathcal{R}_{/E}$-automaton is a triple $\langle r\sigma, \alpha, q\rangle$ such that $l\sigma \to r\sigma$, $l\sigma \xrightarrow{\alpha}_{A^i_{\mathcal{R},E}} q$ and there is no formula α' such that $r\sigma \xrightarrow{\alpha'}_{A^i_{\mathcal{R},E}} q$. The resolution of such a critical pair consists of adding to $\mathtt{C}(A^i_{\mathcal{R},E})$ the transitions to obtain $r\sigma \xrightarrow{\alpha}_{\mathtt{C}(A^i_{\mathcal{R},E})} q$. This is followed by a normalization step $\mathtt{Norm}$ whose definition is similar to the one for classical tree automata (see appendix D).

Definition 5 (Resolution of a critical pair). *Given a $\mathcal{R}_{/E}$-automaton $A = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E\rangle$ and a critical pair $p = \langle r\sigma, \alpha, q\rangle$, the* resolution *of p on A is the $\mathcal{R}_{/E}$-automaton $A' = \langle \mathcal{F}, Q', Q_f, \Delta' \cup \varepsilon'_{\mathcal{R}} \cup \varepsilon_E\rangle$ where*

- $\Delta' = \Delta \cup \mathtt{Norm}(r\sigma, \Delta \setminus \Delta^0)$;
- $\varepsilon'_{\mathcal{R}} = \varepsilon_{\mathcal{R}} \cup \{q' \xrightarrow{\alpha} q\}$ *where q' is the state such that $r\sigma \to_{\Delta' \setminus \Delta_0} q'$;*
- *Q' is the union of Q with the set of states added when creating Δ'.*

Note that Δ_0, the set of transitions of $A^0_{\mathcal{R}}$, is not used in the normalization process. This is to guarantee that A' is well-defined. The $\mathcal{R}_{/E}$-automaton $\mathtt{C}(A^i_{\mathcal{R},E})$ is obtained by recursively applying the above resolution principle to all critical pairs p of the set of critical pairs between $\mathcal{R}$ and $A^i_{\mathcal{R},E}$.

The set of all critical pairs is obtained by solving the *matching problems* $l \trianglelefteq q$ for all rewrite rule $l \to r \in \mathcal{R}$ and all state $q \in A^i_{\mathcal{R},E}$. Solving $l \trianglelefteq q$ is in two steps. First, one computes S, that is the set of all couples (α, σ) such that α is a formula, σ is a substitution of $\mathcal{X} \mapsto Q^i$, and $l\sigma \xrightarrow{\alpha} q$. The formula α is a conjunction of Predicate Eq that denotes the used transitions of ε_E to rewrite $l\sigma$ in q, in accordance with Definition 3. Due to space constraints the algorithm, which always terminates, can be found in Appendix C.

Second, after having computed S for $l \trianglelefteq q$, we identify elements of the set that correspond to critical pairs. By definition of S, we know that there exists a transition $l\sigma \xrightarrow{\alpha}_{A^i_{\mathcal{R},E}} q$ for $(\alpha, \sigma) \in S$. If there exists a transition $r\sigma \xrightarrow{\alpha'}_{A^i_{\mathcal{R},E}} q$, then $r\sigma$ has already been added to $A^i_{\mathcal{R},E}$. If there does not exist a transition of the form $r\sigma \xrightarrow{\alpha'}_{A^i_{\mathcal{R},E}} q$, then $\langle r\sigma, \alpha', q\rangle$ is a critical pair to solve on $A^i_{\mathcal{R},E}$. The following theorem shows that our methodology is complete.

Theorem 2. *If $A^i_{\mathcal{R},E}$ is well-defined then so is $\mathtt{C}(A^i_{\mathcal{R},E})$, and $\forall q \in Q^i$, $\forall t \in \mathcal{L}(A^i_{\mathcal{R},E}, q)$, $\forall t' \in \mathcal{T}(\mathcal{F})$, $t \to_{\mathcal{R}} t' \implies t' \in \mathcal{L}(\mathtt{C}(A^i_{\mathcal{R},E}), q)$.*

Example 3. Let $\mathcal{R} = \{f(x) \to f(s(s(x)))\}$ be a set of rewriting rules and $A^0_{\mathcal{R},E} = \langle \mathcal{F}, Q, Q_F, \Delta^0\rangle$ be a tree automaton such that $Q_F = \{q_0\}$ and $\Delta^0 = \{a \to q_1, f(q_1) \to q_0\}$. The solution of the matching problem $f(x) \trianglelefteq q_0$ is $S = \{(\sigma, \phi)\}$, with $\sigma = \{x \to q_1\}$ and $\phi = \top$. Hence, since $f(s(s(q_1))) \overset{\top}{\not\to}_{A^0_{\mathcal{R},E}} q_0$, $\langle f(s(s(q_1))), \top, q_0\rangle$ is the only critical pair to be solved. So, we have $\mathtt{C}(A^0_{\mathcal{R},E}) = \langle \mathcal{F}, Q^1, Q_F, \Delta^1 \cup \varepsilon^1_{\mathcal{R}} \cup \varepsilon^0_E\rangle$, with:

$\Delta^1 = \mathtt{Norm}(f(s(s(q_1))), \emptyset) \cup \Delta^0 = \{s(q_1) \to q_2, s(q_2) \to q_3, f(q_3) \to q_4\} \cup \Delta^0$,
$\varepsilon^1_{\mathcal{R}} = \{q_4 \xrightarrow{\top} q_0\}$, since $f(s(s(q_1))) \to_{\Delta^1 \setminus \Delta^0} q_4$, $\varepsilon^0_E = \emptyset$ and $Q^1 = \{q_0, q_1, q_2, q_3, q_4\}$.

Observe that if $\mathtt{C}(A^i_{\mathcal{R},E}) = A^i_{\mathcal{R},E}$, then we have reached a fixpoint.

5.2 The Widening Step W

Consider a $\mathcal{R}_{/E}$-automaton $A = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$, the widening consists in computing a $\mathcal{R}_{/E}$-automaton $\mathtt{W}(A)$ that is obtained from A by using E. For each equation $l = r$ in E, we consider all pair (q, q') of distinct states of Q^i such that there exists a substitution σ to obtain the following diagram. Observe that $\xrightarrow{=}_A$, the transitive and reflexive rewriting relation induced by $\Delta \cup \varepsilon_E$, defines particular runs which exclude transitions of $\varepsilon_{\mathcal{R}}$. This allow to build a more accurate approximation. The improvment in accurary is detailed in [27].

$$\begin{tikzcd} l\sigma \arrow[r, equal, "E"'] \arrow[d, "A"', "="] & r\sigma \arrow[d, "A"', "="] \\ q & q' \end{tikzcd}$$

Intuitively, if we have $u \xrightarrow{=}_A q$, then we know that there exists a term t of $Rep(q)$ such that $t =_E u$. The automaton $\mathtt{W}(A)$ is given by the tuple $\langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon'_E \rangle$, where ε'_E is obtained by adding the transitions $q \to q'$ and $q' \to q$ to ε_E (for each pair (q, q')).

Theorem 3. *Assuming that A is well-defined, we have A syntactically included in $\mathtt{W}(A)$, and $\mathtt{W}(A)$ is well-defined.*

Example 4. Consider the $\mathcal{R}_{/E}$-automaton $\mathtt{C}(A^0_{\mathcal{R},E})$ given in Example 3. Using Equation $s(s(x)) = s(x)$, we compute $A^1_{\mathcal{R},E} = \mathtt{W}(\mathtt{C}(A^0_{\mathcal{R},E}))$. We have $\sigma = \{x \mapsto q_1\}$ and the following diagram. We then obtain $A^1_{\mathcal{R},E} = \langle \mathcal{F}, Q^1, Q_f, \Delta^1 \cup \varepsilon^1_{\mathcal{R}} \cup \varepsilon^1_E \rangle$, where $\varepsilon^1_E = \varepsilon^0_E \cup \{q_3 \to q_2,\ q_2 \to q_3\}$ and $\varepsilon^0_E = \emptyset$. Observe that $A^1_{\mathcal{R},E}$ is a fixpoint, i.e., $\mathtt{C}(A^1_{\mathcal{R},E}) = A^1_{\mathcal{R},E}$.

$$\begin{tikzcd} s(s(q_1)) \arrow[r, equal, "E"'] \arrow[d, "\mathtt{C}(A^0_{\mathcal{R},E})"', "="] & s(q_1) \arrow[d, "\mathtt{C}(A^0_{\mathcal{R},E})"', "="] \\ q_3 & q_2 \end{tikzcd}$$

6 A CEGAR procedure for $\mathcal{R}_{/E}$-automata

Let $\mathcal{R}$ be a TRS, I be a set of initial terms characterized by the $\mathcal{R}_{/E}-$automaton $A^0_{\mathcal{R},E}$ and Bad the set of forbidden terms represented by A_{Bad}. We now complete our CEGAR approach by proposing a technique that checks whether a term is indeed reachable from the initial set of terms. If the term is a spurious counter-example i.e. an counter-example of the approximation, then it has to be removed from the approximation automatically, else one can deduce that the involved term is actually reachable.

Let $A^k_{\mathcal{R},E} = \langle \mathcal{F}, Q^k, Q_f, \Delta^k \cup \varepsilon^k_{\mathcal{R}} \cup \varepsilon^k_E \rangle$ be a $\mathcal{R}_{/E}$-automaton obtained after k steps of completion and widening from $A^0_{\mathcal{R},E}$ and assume that $\mathcal{L}(A^k_{\mathcal{R},E}) \cap Bad \neq \emptyset$. Let $S_{A^k_{\mathcal{R},E} \cap A_{Bad}}$ be a set of triples $\langle q, q', \phi \rangle$ where q is a final state of $A^k_{\mathcal{R},E}$, q' is a final state of A_{Bad} and ϕ is a formula on transitions of ε^k_E and such that for each triple (q, q', ϕ), the formula ϕ holds if and only if there exists $t \in \mathcal{L}(A^k_{\mathcal{R},E}, q) \cap \mathcal{L}(A_{Bad}, q')$ and $t \xrightarrow{\phi}_{A^k_{\mathcal{R},E}} q$. Note that $S_{A^k_{\mathcal{R},E} \cap A_{Bad}}$ can be

obtained using the algorithm in Definition 10 presented in Appendix H. We consider two cases. First, as $A^k_{\mathcal{R},E}$ is well-defined, if $\phi = \top$, we deduce that t is indeed a reachable term. Otherwise, ϕ is an formula whose atoms are of the form $Eq(q_j, q'_j)$, and t is possibly a spurious counter-example, and the run $t \xrightarrow{\phi}_{A^k_{\mathcal{R},E}} q$ must be removed. Refinement consists in computing a pruned version $\mathtt{P}(A^k_{\mathcal{R},E}, S_{A^k_{\mathcal{R},E} \cap A_{Bad}})$ of $A^k_{\mathcal{R},E}$.

Definition 6. *Given a $\mathcal{R}_{/E}-$automaton $A = \langle \mathcal{F}, Q, Q_F, \Delta_0 \cup \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$ and a set of specified by the automaton A_{Bad}, the prune process is defined by*

$$\mathtt{P}(A, S_{A \cap A_{Bad}}) = \begin{cases} \mathtt{P}(A', S_{A' \cap A_{Bad}}) & \text{if } S_{A' \cap A_{Bad}} \neq \emptyset \text{ and with} \\ & A' = \mathtt{Clean}(A, S_{A \cap A_{Bad}}) \\ A & \text{if } S_{A \cap A_{Bad}} = \emptyset \text{ or there exists } t \in Bad \\ & \text{s.t. } t \xrightarrow{\top}_A q_f \text{ and } q_f \in Q_F. \end{cases}$$

where $\mathtt{Clean}(A, S_{A \cap A_{Bad}})$, consists of removing transitions of ε_E until for each $\langle q_f, q'_f, \phi \rangle \in S_{A \cap A_{Bad}}$, ϕ does not hold, i.e., $\phi = \bot$ with q_f, q'_f respectively two final states of A and A_{Bad}.

To replace Predicate $Eq(q, q')$ by $\bot$ in ϕ, we have to remove the transition $q \to q'$ from ε_E. In addition, we also have to remove all transitions $q \xrightarrow{\alpha} q' \in \varepsilon_{\mathcal{R}}$, where the conjunction α contains some atoms transitions removed from ε_E. In general, removing Transition $q \to q'$ may be too rude. Indeed, assuming that there also exists a transition $q'' \to q$ of ε_E, removing the transition $q \to q'$ also avoids the induced reduction $q'' \to q'$ from the automaton and then, unconcerned terms of q'' are also removed. To save those terms, Transition $q'' \to q'$ is added to ε_E, but only if it has never been removed by a pruning step. This point is important to refine the automaton with accuracy. The prune step is called recursively as inferred transitions may keep the intersection non-empty.

Theorem 4. *Let $t \in Bad$ be a spurious counter-example. The pruning process always terminates, and removes all the runs of the form $t \xrightarrow{\phi} q$.*

Example 5. We consider the $\mathcal{R}_{/E}$-automaton A of Example 2. It is easy to see that A recognizes the term $g(c)$. Indeed, by Definition 3, we have $g(c) \xrightarrow{Eq(q_c, q_b)} q_f$. Consider now the rewriting path $f(a) \to_{\mathcal{R}} f(b) =_E f(c) \to_{\mathcal{R}} g(c)$. If we remove the step $f(b) =_E f(c)$ denoted by the transition $q_c \to q_b$, then $g(c)$ becomes unreachable and should also be removed. The first step in pruning A consists thus in removing this transition. In a second step, we propagate the information by removing all transition of $\varepsilon_{\mathcal{R}}$ labeled by a formula that contains $Eq(q_c, q_b)$. This is done to remove all terms obtained by rewriting with the equivalence $b =_E c$. After having pruned all the transitions, we observe that the terms recognized by A are given by the set $\{f(a), f(b)\}$.

Let us now characterize the soundness and completness of our approach.

Theorem 5 (Soundness on left-linear TRS). *Consider a left-linear TRS $\mathcal{R}$, a set of terms Bad, a set of equations E and a well-defined $\mathcal{R}_{/E}-$automaton A_0. Let $A^*_{\mathcal{R},E}$ be a fixpoint $\mathcal{R}_{/E}$-automaton of $\mathtt{P}(A', S_{A' \cap A_{Bad}})$ and $A' = \mathtt{W}(\mathtt{C}(A_i))$ for $i \geq 0$. If $\mathcal{L}(A^*_{\mathcal{R},E}) \cap Bad = \emptyset$, then $Bad \cap \mathcal{R}^*(\mathcal{L}(A_0)) = \emptyset$.*

Theorem 6 (Completeness on Linear TRS). *Given a linear TRS $\mathcal{R}$, a set of terms Bad defined by automata A_{Bad}, a set of equations E and a well-defined $\mathcal{R}_{/E}-$automaton A_0. For any $i > 0$, let us consider A_i be the $\mathcal{R}_{/E}-$automaton obtained from A_{i-1} in such a way: $A_i = \mathtt{P}(A', S_{A' \cap A_{Bad}})$ and $A' = \mathtt{W}(\mathtt{C}(A_{i-1}))$. If $Bad \cap \mathcal{R}^*(\mathcal{L}(A_0)) \neq \emptyset$ then there exists $t \in Bad$ and $j > 0$ such that $t \xrightarrow{\top}_{A_j} q_f$ and q_f is a final state of A_j.*

This result also extends to left-linear TRS with a finite set of initial terms. This is sufficient to capture a large class of systems such as various java programs.

Theorem 7 (Completeness on Left-Linear TRS). *Theorem 6 extends to left-linear TRS if for any state q of A_0, the cardinality of $Rep(q)$ is 1.*

7 Implementation and Certification

Our approach has been implemented in TimbukCEGAR that is an extension of the Timbuk 3.1 toolset [29]. Timbuk is a well-acknowledged tree automata library that implements several variants of the completion approach. TimbukCEGAR is around 11000 lines of OCaml, 75% of them being common with Timbuk 3.1. TimbukCEGAR exploits a BDD based representation of equation formulas through the Buddy BDD library [32].

A particularity of TimbukCEGAR is that it is certified. At the heart of any abstraction algorithm there is the need to check whether a candidate over-approximation B is indeed a fixed point, that is if $\mathcal{L}(B) \supseteq \mathcal{R}^*(\mathcal{L}(A))$. Such check has been implemented in various TRMC toolsets, but there is no guarantee that it behaves correct, i.e., that the TRMC toolset gives a correct answer. In [20], a checker for tree automata completion was designed and proved correct using the Coq [10] proof assistant. As such, any TRMC toolset that produces an automaton B that passes the checker can be claimed to work properly. TimbukCEGAR implements an extension of [20] for $\mathcal{R}_{/E}$-automata, which means that the tool delivers correct answers.

In what follows, we describe how Java programs can be analyzed using our approach. Both Timbuk and TimbukCEGAR are available at `http://www.irisa.fr/celtique/genet/timbuk/`.

In a national initiative called RAVAJ [1], we have defined a generic certified verification chain based on TRMC. This chain is composed of three main links. The two first links rely on an encoding of the operational semantics of the programming language as a term rewriting system and a set of rewrite rules. The third link is a TRMC toolset, here TimbukCEGAR. With regards to classical static analysis, the objective is to use TRMC and particularly tree automata completion as a foundation mechanism for ensuring, by construction, safety of

static analyzers. For Java, using approximation rules instead of abstract domains makes the analysis easier to fine-tune. Moreover, our approach relies on a checker that certifies the answer to be correct.

We now give more details and report some experimental results. We used Copster [9], to compile a Java `.class` file into a TRS. The obtained TRS models exactly a subset of the semantics[6] of the Java Virtual Machine (JVM) by rewriting a term representing the state of the JVM [13]. States are of the form `IO(st,in,out)` where `st` is a program state, `in` is an input stream and `out` and output stream. A program state is a term of the form `state(f,fs,h,k)` where `f` is current frame, `fs` is the stack of calling frames, `h` a heap and `k` a static heap. A frame is a term of the form `frame(m,pc,s,l)` where `m` is a fully qualified method name, `pc` a program counter, `s` an operand stack and `t` an array of local variables. The frame stack is the call stack of the frame currently being executed: `f`. We consider the following program:

```
class List{
    List next;
    int val;
    public List(int elt, List l){
        next= l;
        if (elt<0) val= -elt;
        else val= elt;
    }
    public void printSum(){
        List l= this;
        int sum= 0;
        while (l != null){
            sum= sum+l.val;
            l= l.next;
        }
        System.out.println(sum);
    }}
```

```
class TestList{
    public static void main(String[] argv){
        List ls= null;
        int x= 0;
        while (x!=-1) {
            try {x= System.in.read();}
            catch(java.io.IOException e){};
            ls= new List(x,ls);
        }
        ls.printSum();
    }
}
```

Let us now check that the sum output by the program can never be equal to zero, for all non-empty input stream of integers.

The TRS generated by Copster has 879 rules encoding both the JVM semantics and the bytecode of the above Java program. Initial terms are of the form `IO(s,lin,nilout)` where `s` is the initial JVM state, `lin` is a non-empty unbounded list of integers and `nilout` is the empty list of outputs. Starting from this initial set of terms, completion is likely to diverge without approximations. Indeed, the program is going to allocate infinitely many objects of class `List` in the heap and, furthermore, compute an unbounded sum in the method `printSum`. In the heap, there is one separate heap for each class. Each heap consists of a list of objects. For instance, in the heap for class `List`, objects are stored using a list constructor `stackHeapList(x,y)`. Thus, to enforce termination we can approximate the heap for objects of class `List` using the following equation `stackHeapList(x,y)=y`. The effect of this equation is to collapse all the possible lists built using `stackHeapList`, hence all the possible heaps for class `List`. The other equations are `succ(x)=x` and `pred(x)=x` for approximating infinitely growing or decreasing integers.

[6] essentially basic types, arithmetic, object creation, field manipulation, virtual method invocation, as well as a subset of the String library.

By using those equations, TimbukCEGAR finds a counterexample. This is due to the fact that, amongst all considered input streams, an input stream consisting of a list of 0 results into a 0 sum. The solution is to restrict the initial language to non-empty non-zero integer streams. However, refinement of equations is needed since `succ(x)=x` and `pred(x)=x` put 0 and all the other integers in the same equivalence class. Refining those equations by hand is hard, *e.g.* using equations `succ(succ(x))=succ(x)` and `pred(pred(x))=pred(x)` is not enough to eliminate spurious counterexamples. After 334 completion steps and 4 refinement steps, TimbukCEGAR is able to complete the automaton and achieve the certified proof. The resulting automaton produce by the tool has 3688 transitions which are produced in $128s$ and certified in $17017s$. The memory usage for the whole process does not exceed $531Mb$. One of the reason for which certifying automata produced by TimbukCEGAR takes more time than for Timbuk 3.1 is that the checker has to normalize epsilon transitions of $\mathcal{R}_{/E}$-automata. This is straightforward but may cause an explosion of the size of the tree automaton to be checked. It is worth mentioning that the term rewriting rules corresponding to the above example is not right-linear. However, here completion steps do not introduce spurious counter examples.

We give another example of application in Appendix J.

8 Conclusion

We have presented a new CounterExample Guided Abstraction Refinement procedure for TRMC based on equational abstraction. Our approach has been implemented in TimbukCEGAR that is the first TRMC toolset certified correct. Our approach leads, in part, to a java program analyzer starting from code to verification, but without relying on (1) potentially heavy assumptions on datas and architectures, (2) abstraction techniques when translating the code to TRS.

We are convinced that our work open news doors in application of RMC approaches to rigorous system design. One of the remaining challenge is definitively to consider non left-linear TRS. Completion can be extended to deal with such TRS [25]. This is necessary to verify cryptographic protocols with completion [26, 6]. The theoretical challenge is to extend the CEGAR completion to non left-linear TRS. The technical challenge is to extend the Coq checker to handle non left-linear TRS and tree automata with epsilon transitions. Tackling those two goals would allow us to propose the first *certified* automatic verification tool for security protocols, a major advance in the formal verification area.

Acknowledgements Thanks to F. Besson for his help in integrating Buddy.

References

1. Ravaj: Rewriting and approximations for java applications verification. `http://www.irisa.fr/celtique/genet/RAVAJ`.
2. P. A. Abdulla, Y.-F. Chen, G. Delzanno, F. Haziza, C.-D. Hong, and A. Rezine. Constrained monotonic abstraction: A cegar for parameterized verification. In *CONCUR*, LNCS. Springer, 2010.

3. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, LNCS. Springer, 2007.
4. P. A. Abdulla, N. B. Henda, G. Delzanno, F. Haziza, and A. Rezine. Parameterized tree systems. In *FORTE*, volume 5048 of *LNCS*, pages 69–83. Springer, 2008.
5. P. A. Abdulla, A. Legay, A. Rezine, and J. d'Orso. Simulation-based iteration of tree transducers. In *TACAS*, volume 3440 of *LNCS*, pages 30–40. Springer, 2005.
6. Avispa – a tool for Automated Validation of Internet Security Protocols. `http://www.avispa-project.org`.
7. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
8. T. Ball, B. Cook, V. Levin, and S. K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In *IFM*, LNCS. Springer, 2004.
9. N. Barré, F. Besson, T. Genet, L. Hubert, and L. Le Roux. Copster homepage, 2009. `http://www.irisa.fr/celtique/genet/copster`.
10. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer Verlag, 2004.
11. Y. Boichut, R. Courbis, P.-C. Heam, and O. Kouchnarenko. Finer is better: Abstraction refinement for rewriting approximations. In *RTA*, LNCS. Springer, 2008.
12. Y. Boichut, T.-B.-H. Dao, and V. Murat. Characterizing conclusive approximations by logical formulae. In *RP*, volume 6945 of *LNCS*, pages 72–84. Springer, 2011.
13. Y. Boichut, T. Genet, T. Jensen, and L. Leroux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *RTA*, LNCS 4533, pages 48–62, 2007.
14. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *CAV*, LNCS, pages 223–235. Springer, 2003.
15. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking. *ENTCS*, 149(1):37–48, 2006.
16. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract rmc of complex dynamic data structures. In *SAS*, LNCS. Springer, 2006.
17. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV*, volume 3114 of *LNCS*, pages 372–386. Springer, 2004.
18. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, volume 1855 of *LNCS*, pages 403–418. Springer-Verlag, 2000.
19. A. Bouajjani and T. Touili. Extrapolating tree transformations. In *CAV*, volume 2404 of *LNCS*, pages 539–554. Springer, 2002.
20. B. Boyer, T. Genet, and T. Jensen. Certifying a Tree Automata Completion Checker. In *IJCAR'08*, volume 5195 of *LNCS*. Springer, 2008.
21. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. 2008.
22. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *Journal of Logic and Algebraic Programming (JLAP)*, 52-53:109–127, 2002.
23. G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasonning*, 33 (3-4):341–383, 2004.
24. T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *RTA*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
25. T. Genet. Reachability analysis of rewriting for software verification. Université de Rennes 1, 2009. Habilitation.
26. T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE*, volume 1831 of *LNAI*. Springer-Verlag, 2000.

27. T. Genet and R. Rusu. Equational tree automata completion. *JSC*, 45, 2010.
28. T. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems. In *WITS'2003*, 2003.
29. T. Genet and V. Viet Triem Tong. Timbuk 2.0 – a Tree Automata Library. IRISA / Université de Rennes 1, 2001. `http://www.irisa.fr/celtique/genet/timbuk/`.
30. R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.
31. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *CAV*, LNCS. Springer, 1997.
32. J. Lind-Nielsen. Buddy 2.4, 2002. `http://buddy.sourceforge.net`.
33. J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. *TCS*, 403:239–264, 2008.
34. G. Patin, M. Sighireanu, and T. Touili. Spade: Verification of multithreaded dynamic and recursive programs. In *CAV*, LNCS. Springer, 2007.
35. A. Podelski and A. Rybalchenko. Armc: The logical choice for software model checking with abstraction refinement. In *PADL*, LNCS, 2007.
36. T. Takai. A Verification Technique Using Term Rewriting Systems and Abstract Interpretation. In *RTA*, volume 3091 of *LNCS*, pages 119–133. Springer, 2004.
37. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using language inference to verify omega-regular properties. In *TACAS*, LNCS. Springer, 2005.
38. A. Vardhan and M. Viswanathan. Lever: A tool for learning based verification. In *CAV*, LNCS. Springer, 2006.

A Running Example

Consider the $\mathcal{R}_{/E}-$automaton $A^1_{\mathcal{R},E}$ given in Example 4 in Section 5.2. We define A_{Bad} to be a tree automaton whose final state is q'_0 and whose transitions are $a \to q'_1, s(q'_1) \to q'_2, s(q'_2) \to q'_1$ and $f(q'_2) \to q'_0$. The forbidden terms in $\mathcal{L}(A_{Bad})$ are of the form $f(s^{2k+1}(a))$. We observe that $\mathcal{L}(A^1_{\mathcal{R},E}) \cap \mathcal{L}(A_{Bad}) \neq \emptyset$. According to the intersection algorithm described in Appendix H, one can build a set $S_{A^1_{\mathcal{R},E} \cap A_Bad}$ of triples (q_0, q'_0, ϕ), where ϕ is the formula used to prune $A^1_{\mathcal{R},E}$, i.e., to remove those terms that belong to $\mathcal{L}(A^1_{\mathcal{R},E}) \cap \mathcal{L}(A_{Bad})$. Here, $S_{A^1_{\mathcal{R},E} \cap A_Bad} = \{(q_0, q'_0, Eq(q_2, q_3) \wedge Eq(q_3, q_2)), (q_0, q'_0, Eq(q_2, q_3)), (q_0, q'_0, Eq(q_3, q_2))\}$. Thus, we perform the prune step $\mathtt{P}(A^1_{\mathcal{R},E}, S_{A^1_{\mathcal{R},E} \cap A_Bad})$. Removing the transition $q_2 \to q_3$ from ε^1_E is sufficient to invalidate $Eq(q_2, q_3) \wedge Eq(q_3, q_2)$. Moreover, this action invalidates $(q_0, q'_0, Eq(q_2, q_3))$ too. It remains to prune with $(q_0, q'_0, Eq(q_3, q_2))$. This is done by removing the transition $q_3 \to q_2$ from ε^1_E. Thus, ε^1_E becomes empty. At this point, no transition of $\varepsilon^1_{\mathcal{R}}$ can be removed. Indeed, all these transitions are all labeled by $\top$. We thus define $A^2_{\mathcal{R},E} = \mathtt{P}(A^1_{\mathcal{R},E}, S_{A^1_{\mathcal{R},E} \cap A_Bad})$, with $\Delta^2 = \Delta^1$, $\varepsilon^2_{\mathcal{R}} = \varepsilon^1_{\mathcal{R}}$, and $\varepsilon^2_E = \emptyset$. We observe that $A^2_{\mathcal{R},E}$ is not $\mathcal{R}-$closed and should be completed. We thus define $A^3_{\mathcal{R},E} = \mathtt{W}(\mathtt{C}(A^2_{\mathcal{R},E}))$. We found a new critical pair for $f(x) \to f(s(s(x)))$ and we obtain $\Delta^3 = \Delta^2 \cup \{s(q_3) \to q_5, s(q_5) \to q_6, f(q_6) \to q_7$, and $\varepsilon^3_{\mathcal{R}} = \varepsilon^2_{\mathcal{R}} \cup \{q_7 \xrightarrow{\top} q_4\}$.

The interesting point is in the application of $\mathtt{W}$. Observe that the transitions of ε^3_E directly results from the application of the equation $s(x) = s(s(x))$, *i.e.* transitions $q_2 \to q_3, q_3 \to q_2, q_3 \to q_5, q_5 \to q_3, q_5 \to q_6$, and $q_6 \to q_5$.

The two transitions $q_2 \to q_3$ and $q_3 \to q_2$ are ignored since they have been deleted earlier by a prune step, but two new transitions are added: $q_2 \to q_5$ and $q_5 \to q_2$. Indeed, it should have been possible to connect q_2 and q_5 together using transitions $q_2 \to q_3$ and $q_3 \to q_5$ if $q_2 \to q_3$ had not been deleted. So, $\varepsilon^3_E = \{q_2 \to q_5, q_5 \to q_2, q_5 \to q_6, q_6 \to q_5, q_3 \to q_5, q_5 \to q_3\}$. We then check the emptiness of $\mathcal{L}(A^3_{\mathcal{R},E}) \cap \mathcal{L}(A_{Bad})$. This intersection is still not empty and we obtain the following set of triples:

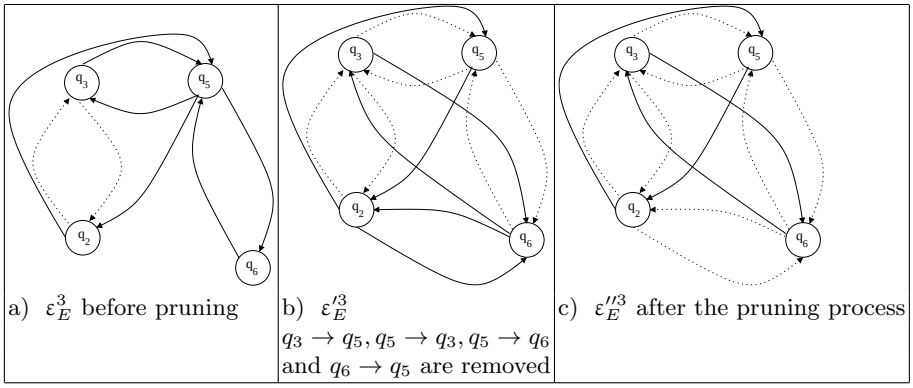
$$S_{A^3_{\mathcal{R},E} \cap A_{Bad}} = \left\{ \begin{array}{l} (q_0, q'_0, Eq(q_3, q_5)), (q_0, q'_0, Eq(q_5, q_3)), (q_0, q'_0, Eq(q_5, q_6)), \\ (q_0, q'_0, Eq(q_6, q_5)), (q_0, q'_0, Eq(q_2, q_5) \wedge Eq(q_5, q_3)), \\ (q_0, q'_0, Eq(q_2, q_5) \wedge Eq(q_5, q_6)), (q_0, q'_0, Eq(q_5, q_2) \wedge Eq(q_3, q_5)), \\ (q_0, q'_0, Eq(q_5, q_2) \wedge Eq(q_6, q_5)) \end{array} \right\}.$$

So, in order to perform $\mathtt{P}(A^3_{\mathcal{R},E}, S_{A^3_{\mathcal{R},E} \cap A_{Bad}})$, the first transitions to remove are $q_3 \to q_5$, $q_5 \to q_3$, $q_5 \to q_6$ and $q_6 \to q_5$. According to Figure 2, ε^3_E of 2a. becomes the one described in 2b. Note that four new transitions are generated: $q_3 \to q_6$, $q_6 \to q_3$, $q_2 \to q_6$ and $q_6 \to q_2$. We denote this new set of transitions by ε'^3_E. According to Definition 6, one has to test if the removing of guilty transitions has broken the non-empty intersection. And here, it is not the case. Indeed, it is still possible to recognize an odd number of s between the symbols f and a using for instance the transition $q_2 \to q_6$: $f(s(a)) \to^*_{\Delta^3} f(q_2)$ and $f(q_2) \to^*_{\varepsilon'^3_E} f(q_6) \to_{\varepsilon^3_{\mathcal{R}}} q_0$. Let $A'^3_{\mathcal{R},E}$ be the $\mathcal{R}_{/E}-$automaton such that $\Delta'^3 = \Delta^3$, ε'^3_E and $\varepsilon'^3_{\mathcal{R}} = \varepsilon^3_{\mathcal{R}}$. Computing $\mathtt{P}(A^3_{\mathcal{R},E}, S_{A^3_{\mathcal{R},E} \cap A_{Bad}})$ remains to compute $\mathtt{P}(A'^3_{\mathcal{R},E}, S_{A'^3_{\mathcal{R},E} \cap A_{Bad}})$ where $S_{A'^3_{\mathcal{R},E} \cap A_{Bad}}$ is defined as follows:

$$S_{A'^3_{\mathcal{R},E} \cap A_{Bad}} = \left\{ \begin{array}{l} (q_0, q'_0, Eq(q_2, q_6)), (q_0, q'_0, Eq(q_6, q_2) \wedge Eq(q_3, q_6)), \\ (q_0, q'_0, Eq(q_6, q_2)), (q_0, q'_0, Eq(q_2, q_6) \wedge Eq(q_6, q_3)) \end{array} \right\}.$$

Removing the transitions $q_2 \to q_6$ and $q_6 \to q_2$ makes the whole set of formula involved in $S_{A'^3_{\mathcal{R},E} \cap A_{Bad}}$ non valid. Let $A''^3_{\mathcal{R},E}$ be the $\mathcal{R}_{/E}-$automaton such that $\varepsilon''^3_{\mathcal{R}} =$

$\varepsilon'^3_{\mathcal{R}}$, $\varepsilon''^3_E = \varepsilon'^3_E \setminus \{q_2 \to q_6, q_6 \to q_2\}$ and $\Delta''^3 = \Delta'^3$. Note that as soon as these transitions are removed, the intersection between $\mathcal{L}(A''^3_{\mathcal{R},E})$ and $\mathcal{L}(A_{Bad})$ is empty. So, $\mathtt{P}(A'^3_{\mathcal{R},E}, S_{A'^3_{\mathcal{R},E} \cap A_{Bad}}) = A''^3_{\mathcal{R},E}$ and consequently, $\mathtt{P}(A^3_{\mathcal{R},E}, S_{A^3_{\mathcal{R},E} \cap A_{Bad}}) = A''^3_{\mathcal{R},E}$.



a) ε^3_E before pruning

b) ε'^3_E
$q_3 \to q_5, q_5 \to q_3, q_5 \to q_6$ and $q_6 \to q_5$ are removed

c) ε''^3_E after the pruning process

Fig. 2: Evolution of ε^3_E during the pruning process

Considering $A^4_{\mathcal{R},E} = \mathtt{P}(A^3_{\mathcal{R},E}, S_{A^3_{\mathcal{R},E} \cap A_{Bad}})$, we restart the completion process and we observe that $A^4_{\mathcal{R},E} = \mathtt{C}(A^4_{\mathcal{R},E})$. We have thus reached a fix-point. There, we observe that $\mathcal{L}(A^4_{\mathcal{R},E}) \cap \mathcal{L}(A_{Bad}) = \emptyset$ and conclude that $\mathcal{R}^*(I) \cap Bad = \emptyset$. Observe that our refinement is accurate in this case. Indeed $\mathcal{L}(A^4_{\mathcal{R},E}) = f(s^{2k}(a))$, that is the exact set of reachable states.

The above example cannot be handled with the approach of [11]. Indeed, this technique cannot handle set of bad terms whose cardinality is infinite.

B Proof of Theorem 1

We want to show that the following result holds:

$$\forall t \in \mathcal{T}(\mathcal{F} \cup Q),\ q \in Q,\ t \xrightarrow{\alpha}_A q \iff t \to^*_A q$$

Proof. The proof is easily done by induction by arguing that it is enough to forget the formulas manipulated by the definition 3 to have the equivalent step with $\to_A$.

C Matching Algorithm for $\mathcal{R}_{/E}$-automata

We assume a left-linear TRS $\mathcal{R}$ and a $\mathcal{R}_{/E}$-automaton $A^i_{\mathcal{R},E}$ such that $A^i_{\mathcal{R},E} = \langle \mathcal{F}, Q^i, Q_f, \Delta^i \cup \varepsilon^i_{\mathcal{R}} \cup \varepsilon^i_E \rangle$

Definition 7 (Matching Algorithm).
Assuming the matching problem $l \trianglelefteq q$ for a $\mathcal{R}_{/E}$-automaton $A^i_{\mathcal{R},E}$. S is the solution

of the matching problem, which is denoted $l \trianglelefteq q \vdash_{A^i_{\mathcal{R},E}} S$*, if there exists a derivation of the statement* $l \trianglelefteq q \vdash_{A^i_{\mathcal{R},E}} S$ *using the rules:*

$$\textit{(Var)}\ \frac{}{x \trianglelefteq q \vdash_A \{(\alpha_k, \{x \mapsto q_k\}) \mid q_k \xrightarrow{\alpha_k}_A q\}} (x \in \mathcal{X})$$

$$\textit{(Delta)}\ \frac{t_1 \trianglelefteq q_1 \vdash_A S_1 \quad \dots \quad t_n \trianglelefteq q_n \vdash_A S_n}{f(t_1, \dots, t_n) \lhd q \vdash_A \bigotimes_1^n S_k} \left(f(q_1, \dots, q_n) \to q \in \Delta \right)$$

$$\textit{(Epsilon)}\ \frac{t \lhd q \vdash_A S_0 \quad t \lhd q'_1 \vdash_A S_1 \quad \dots \quad t \lhd q'_n \vdash_A S_n}{t \trianglelefteq q \vdash_A S_0 \cup \bigcup_{k=1}^n \{(\phi \wedge \alpha_k, \sigma) \mid (\phi, \sigma) \in S_k\}} \begin{pmatrix} \{(q_k, \alpha_k) \mid q_k \xrightarrow{\alpha_k} q\}_1^n \\ t \notin \mathcal{X} \end{pmatrix}$$

Using $\bigotimes_1^n S_j = \{(\top, id) \oplus (\phi_1, \sigma_1) \oplus \dots \oplus (\phi_n, \sigma_n) \mid (\phi_j, \sigma_j) \in S_j\}$*, and* $(\phi, \sigma) \oplus (\phi', \sigma') = (\phi \wedge \phi', \sigma \cup \sigma')$*.*

Observe that, by definition, the matching problem considers possibly infinite runs of the form $l\sigma \xrightarrow{\alpha} q$. Indeed, transitions in $\varepsilon^i_{\mathcal{R}} \cup \varepsilon^i_E$ can introduce loops. In the matching algorithm, we exclude such runs. This is done to keep a finite set of rewriting path, which is computable in a finite amount of time. It is worth mentioning that removing loops does not influence the result. As an example, consider the automaton A of Example 2. We observe that $f(b) \xrightarrow{Eq(q_b,q_c) \wedge Eq(q_c,q_b)}_A q_f$ uses the loop $f(b) =_E f(c) =_E f(b)$. This loop can be removed as $f(a) \to^*_{\mathcal{R}} f(b)$ can be obtained by $f(b) \xrightarrow{\top}_A q_f$, a run which does not contain any loops.

We show that the matching algorithm given in Definition 7 is complete.

Lemma 1. *Let* A *be a* $\mathcal{R}_{/E}-$*automaton,* q *one of its states,* $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *the linear left member of a rewriting rule and* σ *a* Q*-substitution with a domain range-restricted to* $\mathcal{V}(l)$*. If the set* S *is solution of the matching problem* $l\sigma \trianglelefteq q$*, then we have* $\forall(\alpha, \sigma),\ l\sigma \xrightarrow{\alpha}_A q \iff (\alpha, \sigma) \in S$

Proof. Assuming $\mathcal{F}$ a set of symbols, $\mathcal{X}$ a set of variable and Q a set of states. We define $A = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$; $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $q \in Q$; $\sigma : \mathcal{V}ar(l) \to Q$ and $\alpha = \bigwedge_1^n Eq(q_k, q'_k)$ such that $l\sigma \xrightarrow{\alpha}_A q$.

The proof is done by induction on the term l.

Base case: l is a variable.

In this case, σ must be a Q-substitution of the form $\sigma = \{l \mapsto q'\}$. Using this observation and the hypothesis, we have $q' \xrightarrow{\alpha}_A q$. The matching problem $l \trianglelefteq q$ is solved using Rule (Var). This means that $S = \{(\alpha_k, \{l \mapsto q_k\}) \mid q_k \xrightarrow{\alpha_k}_A q\}$. By definition of S we see that S contains (α, σ).

Induction : Assume now l is a linear term of the form $f(t_1, \dots, t_n)$.

We are going to decompose $f(t_1, \dots, t_n)\sigma \xrightarrow{\alpha}_A q$ into sequences of transitions. First observe that, by splitting σ into $\sigma_1 \dots \sigma_n$, we have that $f(t_1, \dots, t_n)\sigma$ is equal to $f(t_1\sigma_1, \dots, t_n\sigma_n)$. Assume $\sigma = \sigma_1 \sqcup \dots \sqcup \sigma_n$ with $dom(\sigma_i) = \mathcal{V}(t_i)$ and $\forall x \in dom(\sigma_i),\ \sigma_i(x) = \sigma(x)$. Since l is linear, each variable in X occurs at most one time in l. This means that the sets $\mathcal{V}(t_i)$ are disjoints and so are the domains of the σ_i. This ensures that σ is well-defined.

We now study the decomposition of $f(t_1\sigma_1, \dots, t_n\sigma_n) \xrightarrow{\alpha}_A q$ to show that transitions of A used to recognized the term $f(t_1\sigma_1, \dots, t_n\sigma_n)$ are considered by the corresponding steps of the matching algorithm.

We observe that the term $f(t_1\sigma_1, \ldots, t_n\sigma_n)$ is recognized in State q. Indeed, we have $f(q_1, \ldots, q_n) \to q' \in \Delta$, and each subterm $t_i\sigma_i$ is recognized in state q_i such that $t_i\sigma_i \xrightarrow{\alpha_i} q_i$. Composing recognizing of each subterm, we obtain the following sequence:

$$f(t_1, \ldots, t_n) \xrightarrow{\alpha_1} f(q_1, t_2, \ldots, t_n) \xrightarrow{\bigwedge_1^2 \alpha_i} f(q_1, q_2, t_3, \ldots, t_n) \xrightarrow{\bigwedge_1^3 \alpha_i} \ldots$$
$$\ldots \xrightarrow{\bigwedge_1^n \alpha_i} f(q_1, \ldots, q_n) \xrightarrow{\bigwedge_1^n \alpha_i \wedge \top} q'$$

There

are two cases that have to be considered : (1) $q = q'$ and (2) $q \neq q'$. (1) If $q = q'$, the decomposition is complete and $f(t_1\sigma_1, \ldots, t_n\sigma_n) \xrightarrow{\alpha}_A q$ with $\alpha = \bigwedge_1^n \alpha_i$.

$$f(t_1\sigma_1, \ldots, t_n\sigma_n) \xrightarrow{\bigwedge_{i=1}^n \alpha_i} f(q_1, \ldots, q_n) \xrightarrow{\bigwedge_{i=1}^n \alpha_i} q$$

(2) $q \neq q'$: $f(t_1\sigma_1, \ldots, t_n\sigma_n) \xrightarrow{\alpha}_A q$ holds only if we have a transition $q' \xrightarrow{\alpha'} q$ such that $\alpha = \bigwedge_1^n \alpha_i \wedge \alpha'$.

$$f(t_1\sigma_1, \ldots, t_n\sigma_n) \xrightarrow{\bigwedge_{i=1}^n \alpha_i} f(q_1, \ldots, q_n) \xrightarrow{\top} q' \xrightarrow{\alpha'} q$$

By induction, we know that for each sequence $t_i\sigma_i \xrightarrow{\alpha_i} q_i$, the matching problem is solved *i.e.* $t_i \trianglelefteq q_i \vdash S_i$ with S_i contains (α_i, σ_i). Rule (Delta) is applied to all premises $t_i \trianglelefteq q_i \vdash_A S_i$ for the transition $f(q_1, \ldots, q_n) \to q' \in \Delta$. From this, we obtain a set $S' = \bigotimes_1^n S_i$. By unfolding the definition of $\bigotimes$, we have $S = \{(\top, id) \oplus (a^1, s^1) \oplus \ldots (a^n, s^n) \mid (a^i, s^i) \in S_i\}$. Since each S_i contains (α_i, σ_i), S' contains $(\top, id) \oplus (\alpha_1, \sigma_1) \oplus \ldots (\alpha_n, \sigma_n)$ which is, by definition of $\oplus$ equal to $(\bigwedge_1^n \alpha_i,\ \sigma)$. Thus, we obtain a intermediate statement $f(t_1, \ldots, t_n) \lhd q' \vdash_A S'$ such that $f(t_1, \ldots, t_n)\sigma \xrightarrow{\bigwedge_1^n \alpha_i} q'$, where $(\bigwedge_1^n \alpha_i, \sigma) \in S'$.

This statement must correspond to one of the premises of Rule (Epsilon) to produce the expected statement $f(t_1, \ldots, t_n) \trianglelefteq q \vdash_A S$. Two cases have to be considered :$q = q'$ and $q \neq q'$.

If $f(q_1, \ldots, q_n) \to q' \in \Delta$ is the last transition used to have $f(t_1, \ldots, t_n)\sigma \xrightarrow{\alpha}_A q$ then we have $\alpha = \bigwedge_1^n \alpha_i$ and we are in the case $q = q'$: this case corresponds to the premiss 0 of Rule (Epsilon) and $S' = S_0$. By definition of Rule (Epsilon), S' is included in S. This means that $(\alpha, \sigma) \in S$.

If we have $q \neq q'$, then it remains a sequence of transitions $q' \xrightarrow{\alpha'} q$ to have $f(t_1, \ldots, t_n)\sigma \xrightarrow{\alpha}_A q$. The couple (α', q') is in the set $\{(q_k, \alpha_k) \mid q_k \xrightarrow{\alpha_k} q\}$.This means that the statement $f(t_1, \ldots, t_n) \trianglelefteq q \vdash_A S'$ is one the remaining premisses. By definition of Rule (Epsilon), S contains all couple $(a \wedge \alpha', s)$ where $(a, s) \in S'$. In particular, S contains $(\bigwedge_1^n \alpha_i \wedge \alpha', \sigma)$ which concludes the proof.

D Normalization for $\mathcal{R}_{/E}$-automata

Definition 8 (Normalization). *The normalization is done in two mutually inductive steps parametrized by the configuration c to recognize, and by the set of transitions Δ to extend. Let Q_{new} be a set of new states,*

$$\begin{cases} \texttt{Norm}(c, \Delta) = \texttt{Slice}(d, \Delta) & c \to_\Delta d, \text{ and } c,\ d \in \mathcal{T}(\mathcal{F} \cup Q) \\ \texttt{Slice}(q, \Delta) = \Delta & q \in Q \\ \texttt{Slice}(f(q_1, \ldots, q_n), \Delta) = \Delta \cup \{f(q_1, \ldots, q_n) \to q\} & q_i \in Q \text{ and } q \in Q_{new} \\ \texttt{Slice}(f(t_1, \ldots, t_n), \Delta) = \texttt{Norm}(f(t_1, \ldots, t_n), \texttt{Slice}(t_i, \Delta)) & t_i \in \mathcal{T}(\mathcal{F} \cup Q) \setminus Q \end{cases}$$

E Proofs of Theorem 2

In order to prove that $\texttt{C}(A^i_{\mathcal{R},E})$ is well-defined, we need to show that, first $\texttt{C}(A^i_{\mathcal{R},E})$ is determinist if we do not consider the transition set of $A^i_{\mathcal{R},E}$ (Lemma 2) and second, solving one critical pair preserves well-definition (Lemma 3).

Lemma 2 (Existence of a representative). *Assume that $A_{\mathcal{R},E} = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$ is a $\mathcal{R}_{/E}$-automaton obtained after k steps of completion from $A^0_{\mathcal{R},E}$. Let c be a configuration. If $\Delta' = \texttt{Norm}(c, \Delta \setminus \Delta^0)$, then there exists a state q such that $c \to^!_{\Delta'} q$.*

Proof. Let $\mu : \mathcal{T}(\mathcal{F} \cup Q) \to \mathbb{N}$ be the measure that counts the number of occurences of symbols in $\mathcal{F}$ of a configuration. Example : $\mu(f(q_1, g(q_2), a)) = 3$. We define it inductively by $\mu(q) = 0$ if $q \in Q$, and $\mu(f(t_1, \ldots, t_n)) = 1 + \sum_1^n \mu(t_i)$.

Assuming $\mathcal{F}$ a set of symbols, and Q a set of states. We define $A_{\mathcal{R},E} = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E \rangle$; $c \in \mathcal{T}(\mathcal{F} \cup Q)$. Assume that $\Delta^1 = \Delta \setminus \Delta^0$ is determinist.

The first step $\texttt{Norm}(t, \Delta^1)$ consists in rewriting c by Δ^1 in its normal form d

The second step $\texttt{Slice}(d, \Delta^1)$ returns Δ^2 such that there exists a unique state q such that $d \to^!_{\Delta^2} q$.

The proof is one by induction on the decreasing of $\mu(d)$. We consider the 3 cases of $\texttt{Slice}(d, \Delta^1)$

1. $\texttt{Slice}(q, \Delta^1) = \Delta^1$. It means that d is the state q. There exists a unique state ,which is q, such that $d \to^!_{\Delta^1} q$.
2. $\texttt{Slice}(f(q_1, \ldots, q_n), \Delta^1) = \Delta^1 \cup \{f(q_1, \ldots, q_n) \to q \mid q \in Q_{new}\}$. Each q_i is a state. The configuration $f(q_1, \ldots, q_n)$ can be used as the left-member of a normalised ground transition. We build the new transition $f(q_1, \ldots, q_n) \to q$ using a new state q. Adding a such transition to Δ^1 preserves determinism. We know that it is impossible to rewrite more $d = f(q_1, \ldots, q_n)$ using transitions of Δ^1 : the new transition $f(q_1, \ldots, q_n) \to q$ is the unique way to rewrite d. We deduce that $\Delta^2 = \Delta^1 \cup \{f(q_1, \ldots, q_n) \to q \mid q \in Q_{new}\}$ is deterministic, and $d \to^!_{\Delta^2} q$.
3. $\texttt{Slice}(f(t_1, \ldots, t_n), \Delta^1) = \texttt{Norm}(f(t_1, \ldots, t_n), \texttt{Slice}(t_i, \Delta^1)\ )$, $t_i \in \mathcal{T}(\mathcal{F} \cup Q) \setminus Q$. Here, we have the direct subterm t_i of d which is not a state. We deduce $\mu(t_i) < \mu(d)$ from the definition of μ. By induction, Δ is extended by $\texttt{Slice}(t_i, \Delta^1)$ to obtain Δ^2 for which there exists a state q such that $t_i \to^!_{\Delta^2} q$. Using this new set Δ^2, we unfold $\texttt{Norm}(f(t_1, \ldots, t_n), \Delta^2)$ which consists in rewriting $f(t_1, \ldots, t_n)$ using Δ^2. We obtain a new configuration $f(t'_1, \ldots, t'_n)$ where we know at less t'_i is equal to q since the direct subterm t_i can be rewritten in q using Δ^2. Note that if some subterms of t_i are also subterms of some other t_j, it will also be rewritten by Δ^2 in t'_j until we reach the normal form. Each step of rewriting by Δ^2 necessarly

replaces a symbol of $\mathcal{F}$ by a state of Q by definition of a normalised transition. This remark allows to prove that $\mu(f(t_1,\ldots,t_n) > \mu(f(t'_1,\ldots,t'_n)$. For the direct subterm t_i, we know $\mu(t_i) > 0$ (t_i is not a state), and $\mu(t'_i) = 0$ (t'_i is the state q). For all other direct subterm t_j with $j <> i$ we deduce $\mu(t_j) \geq \mu(t'_j)$ from $t_j \rightarrow^!_{\Delta^2} t'_j$ using Δ^2. We have $\mu(f(t_1,\ldots,t_n) > \mu(f(t'_1,\ldots,t'_n)$ by definition of μ, and $f(t'_1,\ldots,t'_n)$ is rewritten as most as possible by the deterministic Δ^2. Then, we use again the induction hypothesis to deduce that $\Delta' = \texttt{Slice}(f(t'_1,\ldots,t'_n),\Delta^2)$ extends Δ^2 in order to have a unique state q such that $f(t'_1,\ldots,t'_n) \rightarrow^!_{\Delta^3} q$. By transivity, we have $d \rightarrow^!_{\Delta'} q$ using the deterministic set Δ' for d which is equal to $f(t_1,\ldots,t_n)$.

Finally, we proved that $\Delta' = \texttt{Slice}(d,\Delta^1)$ extends Δ^1 preserving its determinism such that there exists a state q for which $d \rightarrow^!_{\Delta'} q$. We also know that $c \rightarrow^!_{\Delta'} d$. We can conclude that $\Delta' = \texttt{Norm}(c,\Delta^1)$ is determinist, and there exists a state q such that $c \rightarrow^!_{\Delta'} q$.

Let us now show that solving one critical pair preserves well-definition. Let $CP = \{\langle r_1\sigma_1,\alpha_1,q_1\rangle,\ldots,\langle r_n\sigma_n,\alpha_n,q_n\rangle\}$ be the finite set of critical pairs computed from $A^i_{\mathcal{R},E}$ that have to be solved by using Definition 5. By definition, considering $A_0 = A^i_{\mathcal{R},E}$ there exists a sequence of $\mathcal{R}_{/E}-$automata $A_1,\ldots,A_n$, where A_j is obtained from A_{j-1} by solving the critical pair $\langle r_j\sigma_j,\alpha_j,q_j\rangle$. Thus, $\texttt{C}(A^i_{\mathcal{R},E}) = A_n$. For a question of readability and in order to prevent any confusion between notations, each $\mathcal{R}_{/E}-$automaton A_j is defined as follows: $A_j = \langle \mathcal{F}, Q^{n+1}, Q_f, \Delta'^j \cup \varepsilon'^j_{\mathcal{R}} \cup \varepsilon'^j_E\rangle$.

Lemma 3. *Let $\mathcal{R}$ be a linear TRS. Let A and A' be two $\mathcal{R}_{/E}-$automaton such that A' is obtained from A by solving a critical pair $\langle r\sigma,\alpha,q\rangle$ of A. If A is well-defined then so is A'.*

Proof. Assume that $A = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E\rangle$ and $A' = \langle \mathcal{F}, Q', Q_f, \Delta' \cup \varepsilon'_{\mathcal{R}} \cup \varepsilon'_E\rangle$. According to Definition 5, $\Delta' = \Delta \cup \texttt{Norm}(r\sigma, \Delta \setminus \Delta^0)$, $\varepsilon'_{\mathcal{R}} = \{q' \xrightarrow{\alpha} q\} \cup \varepsilon_{\mathcal{R}}$ and $\varepsilon'_E = \varepsilon_E$. Following Definition 4, we first show in (1) that for all state q'' of A', and all term v such that $v \xrightarrow{\top}_{A'} q''$, there exists u a term representative of q'' such that $u \rightarrow^*_{\mathcal{R}} v$. Then, in (2) we show that if $q_1 \xrightarrow{\phi} q_2$ is a transition of $\varepsilon'_{\mathcal{R}}$, then there exist terms $s,t \in \mathcal{T}(\mathcal{F})$ such that $s \xrightarrow{\phi}_{A'} q_1$, $t \xrightarrow{\top}_{A'} q_2$ and $t \rightarrow_{\mathcal{R}} s$.

1. We show the property by induction on the height of t. Let us assume that for all term t' of height less than the height of t and for all $q \in Q_{A'}$, we have $t' \xrightarrow{\top}_{A'} q \implies \exists u \in Rep(q) : u \rightarrow^*_{\mathcal{R}} t'$. Now let us prove that the result holds for t. We consider several cases.
   - If $q \in Q_A$ and $t \xrightarrow{\top}_A q$, then since A is well defined, we get the representative $u \in Rep(q)$ such that $u \rightarrow^*_{\mathcal{R}} t$ from well-definition of A.
   - Assume now that $q \in Q_A$, $t \not\xrightarrow{\top}_A q$ and $t \xrightarrow{\top}_{A'} q$. We show the property by induction on the height of t. Since t is recognized in A' and not in A, this means that the run $t' \xrightarrow{\top}_{A'} q$ needs the transitions added by the resolution of a critical pair. Hence there exists a rewrite rule $l \rightarrow r$, a substitution $\sigma : \mathcal{X} \mapsto Q_A$, a formula α and a state q_c such that $l\sigma \xrightarrow{\alpha}_A q_c$ and $\langle r\sigma,\alpha,q_c\rangle$ is the critical pair. Moreover, the resolution of this critical pair produces the following set of transitions: $\Delta_{A'} = \texttt{Norm}(r\sigma, \Delta_A \setminus \Delta_0)$ and $\varepsilon^{A'}_{\mathcal{R}} = \varepsilon^A_{\mathcal{R}} \cup \{q'_c \rightarrow q_c\}$ such that $r\sigma \rightarrow^!_{\Delta'\setminus\Delta_0} q'_c$. Recall that $t' \xrightarrow{\top}_{A'} q$ needs transitions not

occurring in A. However, all the *new* transitions produced by $\texttt{Norm}(r\sigma, \Delta_A \setminus \Delta_0)$ necessarily range on *new* states, i.e. states not occurring in Q_A. As a result, those transitions cannot be used to get $t' \xrightarrow{\top}_{A'} q$ with $q \in Q_A$. This means that the run $t \xrightarrow{\top}_{A'} q$ uses at least once $q'_c \to q_c$ and $\alpha = \top$. To sum up, we know that there exists a ground context $C[\,]$ such that $t = C[t'] \xrightarrow{\top}_{A'} C[q'_c] \xrightarrow{\top}_{A'} C[q_c] \xrightarrow{\top}_{A} q$. Note that if $q'_c \to q_c$ the same reasonning can be applied. We start to reason on the occurrence of $q'_c \to q_c$ that is the closest to q. Now, our objective is to show that there exists $u \in Rep(q'_c)$ such that $u \to^*_{\mathcal{R}} t'$. If $t' \xrightarrow{\top}_{A} q'_c$, then since A is well defined the result is a direct consequence of Definition 4. Otherwise this means that q'_c is a new state of A (i.e. $q'_c \notin Q_A$) that has been added by the resolution of the critical pair, i.e. $r\sigma \to^!_{\Delta'} q'_c$. By Lemma 2, we get that there exists a substitution $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $t' = r\sigma'$. Using the same Lemma, from $t' = r\sigma' \xrightarrow{\top}_{A'} q'_c$ and $r\sigma \xrightarrow{\top}_{A'} q'_c$, we get that for all variable x of r: $\sigma'(x) \xrightarrow{\top}_{A'} \sigma(x)$. Note that $\sigma(x) \in Q_A$ and that $\sigma'(x)$ are necessarily terms of height less to the height of t. Using the induction hypothesis, we get that for all state $\sigma(x)$ there exists a representative u_x such that $u_x \to^*_{\mathcal{R}} \sigma'(x)$. Let σ_{Rep} be the substitution mapping every variable x to u_x. We have $r\sigma_{Rep} \in Rep(q'_c)$. Moreover, $r\sigma_{Rep} \to^*_{\mathcal{R}} r\sigma' = t'$. Now, we show that $l\sigma_{Rep} \to_{\mathcal{R}} r\sigma_{Rep}$. This is not straightforward since $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. However, it is possible to extend σ_{Rep} into σ'_{Rep}, where every variable y of $\mathcal{V}ar(l)$ not occurring in σ_{Rep} is mapped to a representative of $\sigma(y)$. Hence, $l\sigma'_{Rep} \to_{\mathcal{R}} r\sigma'_{Rep} \to^*_{\mathcal{R}} t'$. From the critical pair we know that $l\sigma \xrightarrow{\alpha}_{A} q_c$ and we found that $\alpha = \top$. Hence $l\sigma'_{Rep} \xrightarrow{\top}_{A} q_c$. Since A is well-defined, we get that there is a representative $v \in Rep(q_c)$ such that $v \to^*_{\mathcal{R}} l\sigma'_{Rep}$. By transitivity of $\to_{\mathcal{R}}$, we get that $v \to^*_{\mathcal{R}} t'$. Above, we found that $t = C[t'] \xrightarrow{\top}_{A'} C[q'_c] \xrightarrow{\top}_{A'} C[q_c] \xrightarrow{\top}_{A} q$. From this and $v \in Rep(q_c)$, we get that $C[v] \xrightarrow{\top}_{A} q$. Since A is well defined, we know that there exists a representative $w \in Rep(q)$ such that $w \to^*_{\mathcal{R}} C[v]$. To conclude, we found $w \in Rep(q)$ and $w \to^*_{\mathcal{R}} C[v] \to C[t'] = t$.

- If $q \notin Q_A$ ($q \in Q'_A \setminus Q_A$), $t \not\xrightarrow{\top}_{A} q$ and $t \xrightarrow{\top}_{A'} q$. Since $q \in Q'_A \setminus Q_A$, we know that q has been added by the resolution of a critical pair. As above, we can deduce that there exists a rewrite rule $l \to r$, a substitution $\sigma : \mathcal{X} \mapsto Q_A$, a formula α and a state q_c such that $l\sigma \xrightarrow{\alpha}_{A} q_c$ and $\langle r\sigma, \alpha, q_c\rangle$ is the critical pair. Moreover, the resolution of this critical pair creates $\Delta_{A'} = \texttt{Norm}(r\sigma, \Delta_A \setminus \Delta_0)$ and $\varepsilon^{A'}_{\mathcal{R}} = \varepsilon^{A}_{\mathcal{R}} \cup \{q'_c \to q_c\}$ such that $r\sigma \to^!_{\Delta' \setminus \Delta_0} q'_c$. Since q is a new state of A' that has been used in the normalization of a subterm of $r\sigma$. More precisely, we know that there exists a term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and a context $C[\,]$ (possibly empty) such that $r\sigma = C[s]$, $C[s]\sigma \to^*_{\Delta'} q'_c$ and $s\sigma \to^*_{\Delta'} q$. Similarly, we know that there exists a substitution $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $s\sigma' = t$. We get that for every variable x of r: $\sigma'(x) \xrightarrow{\top}_{A'} \sigma(x)$. Note that $\sigma(x) \in Q_A$ and that $\sigma'(x)$ are necessarily terms of height lesser to the height of t. By induction hypothesis, we obtain that for every state $\sigma(x)$ there exists a representative u_x such that $u_x \to^*_{\mathcal{R}} \sigma'(x)$. Let σ_{Rep} be the substitution mapping every variable x to u_x. We have $s\sigma_{Rep} \in Rep(q)$ and $s\sigma_{Rep} \to^*_{\mathcal{R}} s\sigma' = t$.

2. It is easy to see that, for any transitions $q_1 \xrightarrow{\phi} q_2 \in \varepsilon_{\mathcal{R}}$, the property still holds. Let us now focus on the transition $q' \xrightarrow{\alpha} q$ resulting from the resolution of Critical pair $\langle r\sigma, \alpha, q\rangle$. By definition, $\langle r\sigma, \alpha, q\rangle$ results from the application of the matching algorithm of Definition 7 given in Appendix C. So there exists a rule $l \to r \in \mathcal{R}$ such

that $(\alpha, \sigma) \in S$, with $l \trianglelefteq q \vdash_A S$. Moreover, since the critical pair has to be solved: $l\sigma \xrightarrow{\alpha} q$ and there is no formula α' such that $r\sigma \xrightarrow{\alpha'}_A q$. Since $\mathcal{R}$ is left-linear, for each variable $x \in \mathcal{V}ar(l)$, one can define the substitution $\sigma' : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ as follows: Assuming q_s being the state of A such that $\sigma(x) = q_s$, let $\sigma'(x) = Rep(q_s)$. By definition of Rep, $Rep(q_s) \xrightarrow{\top} q_s$. So, there exists a derivation such that $l\sigma' \xrightarrow{\top} l\sigma$ and $l\sigma \xrightarrow{\alpha} q$. One can deduce that $r\sigma' \xrightarrow{\top} r\sigma$. According to Lemma 2, one can deduce that there exists a unique q' such that $r\sigma \to^*_{\mathtt{Norm}(r\sigma, \Delta \setminus \Delta^0)} q'$. If $\mathtt{Norm}(r\sigma, \Delta \setminus \Delta^0) \neq \emptyset$ then each transition composing it is of the form $f(q'_1, \ldots, q'_n) \to q'_{n+1}$. Consequently, $r\sigma \xrightarrow{\top} q'$. Considering the transition $q' \xrightarrow{\alpha} q$, one has $r\sigma' \xrightarrow{\top} r\sigma \xrightarrow{\top} q' \xrightarrow{\alpha} q$. Finally, assuming $s = l\sigma'$ and $t = r\sigma'$, there exists $s, t \in \mathcal{T}(\mathcal{F})$ such that one has $s \xrightarrow{\alpha} q$, $t \xrightarrow{\alpha} q'$ and $s \to_{\mathcal{R}} t$.

To conclude, A' is also well-defined.

Theorem 2 is in two parts. We first show that $\mathtt{C}(A^i_{\mathcal{R},E})$ is well-defined if $A^i_{\mathcal{R},E}$ is well-defined.

Proof. Let P_n be the following proposition: A_n is well-defined.

- P_0: Trivial since $A_0 = A^i_{\mathcal{R},E}$ and $A^i_{\mathcal{R},E}$ is well-defined by hypothesis.
- $P_n \Rightarrow P_{n+1}$: By hypothesis, A_{n+1} is obtained from A_n by solving the critical pair $\langle r_{n+1}\sigma_{n+1}, \alpha_{n+1}, q_{n+1}\rangle$. Applying Lemma 3, one obtains automatically that A_{n+1} is well-defined.

So, one can deduce that $\mathtt{C}(A^i_{\mathcal{R},E})$ is well-defined.

We now show that $\mathcal{L}(A^i_{\mathcal{R},E}) \subseteq \mathcal{L}(\mathtt{C}(A^i_{\mathcal{R},E}))$.

Proof. Let q be a state of $A^i_{\mathcal{R},E}$ and t be a term of $\mathcal{L}(A^i_{\mathcal{R},E}, q)$. Suppose that there exist a position $p \in \mathcal{P}os(t)$, a rule $l \to r \in \mathcal{R}$ and a substitution $\sigma' : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $t|_p = l\sigma'$. Let t' be the term such that $t' = t[r\sigma']_p$. Since $t \in \mathcal{L}(A^i_{\mathcal{R},E}, q)$, there exists a state q' of $A^i_{\mathcal{R},E}$ such that $t|_p = l\sigma' \to^*_{A^i_{\mathcal{R},E}} q'$ and $t[q']_p \to^*_{A^i_{\mathcal{R},E}} q$. Following Property 1 given in Appendix C, there exists $(\alpha, \sigma) \in S$ with $l \trianglelefteq q' \vdash_{A^i_{\mathcal{R},E}} S$ such that $l\sigma' \to^*_{A^i_{\mathcal{R},E}} l\sigma$ and $l\sigma \to^*_{A^i_{\mathcal{R},E}} q'$. If $\langle r\sigma, \alpha, q'\rangle$ is already solved then $r\sigma \to^*_{A^i_{\mathcal{R},E}} q'$. Consequently, $r\sigma'$ can also be reduced to q' in $A^i_{\mathcal{R},E}$. Since $t' = t[r\sigma']_p \to^*_{A^i_{\mathcal{R},E}} q$, $t' \in \mathcal{L}(\mathtt{C}(A^i_{\mathcal{R},E}), q)$. Suppose now that $r\sigma \not\to^*_{A^i_{\mathcal{R},E}} q'$. So, there exists $\langle r_i\sigma_i, \alpha_i, q_i\rangle \in CP$ such that $\langle r_i\sigma_i, \alpha_i, q_i\rangle = \langle r\sigma, \alpha, q'\rangle$. By construction, $r\sigma \to^*_{A_i} q'$. Consequently, $r\sigma'$ can also be reduced to q' in A_i. Since A_i is syntactically included in $\mathtt{C}(A^i_{\mathcal{R},E})$, one can deduce that $t' = t[r\sigma']_p \to^*_{C(A^i_{\mathcal{R},E})} q$. Concluding the proof.

F Proofs of Theorem 3

The theorem is in two parts. We first show that if a $\mathcal{R}_{/E}$-automaton A is well-defined, then so is $\mathtt{W}(A)$.

Proof. Assume that $A = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon_E\rangle$ is well-defined. We have $\mathtt{W}(A) = \langle \mathcal{F}, Q, Q_f, \Delta \cup \varepsilon_{\mathcal{R}} \cup \varepsilon'_E\rangle$. We also have $\varepsilon_{\mathcal{R}} \supseteq \varepsilon'_{\mathcal{R}}$. Indeed, $\mathtt{W}$ only adds transitions to the $\varepsilon_{\mathcal{R}}$. We have to prove that the two items of Definition 4 are satisfied.

– The transitions of ε'_E do not participate to runs of the form $\xrightarrow{\alpha}$, whith $\alpha = \top$ (due to the second item in Definition 3). This means that for any term t and any state q, $t \xrightarrow{\top}_{W(A)} q$ is equivalent to $t \xrightarrow{\top}_A q$. Since A is well-defined, we know that there exists $u \in Rep(q)$ such that $u \rightarrow^*_{\mathcal{R}} t$. u is also a representative of $\mathtt{W}(A)$, and we deduce that first item of Definition 4 holds for $\mathtt{W}(A)$.
– By definition, $\mathtt{W}$ only adds transitions to ε'_E and do not remove transitions of A. For all transitions $q \xrightarrow{\alpha} q' \in \varepsilon'_{\mathcal{R}}$, we have $q \xrightarrow{\alpha} q' \in Drw$. Since A is well-defined, we know that there exist terms $s, t \in \mathcal{T}(\mathcal{F})$ such that $s \xrightarrow{\phi}_A q$, $t \xrightarrow{\top}_A q'$ and $t \rightarrow_{\mathcal{R}} s$. We also have $s \xrightarrow{\phi}_{\mathtt{W}(A)} q$, $t \xrightarrow{\top}_{\mathtt{W}(A)} q'$ and $t \rightarrow_{\mathcal{R}} s$.

We now show the second part of the theorem. For all $\mathcal{R}_{/E}$-automaton A, $\mathcal{L}(\mathtt{W}(A)) \supseteq \mathcal{L}(A)$.

Proof. We observe that the widening operator can only adds transitions. As a consequence, this operator cannot restrict the language of A.

G Proofs of Theorem 4

The theorem is in two parts. We first show that the pruning process always terminates.

Proof. Let $t \in \mathcal{T}(\mathcal{F})$ be a term such that $t \xrightarrow{\phi}_{A^k_{\mathcal{R},E}} q$ and $\phi \neq \top$. By hypothesis, t is thus a spurious counter-example. Moreover, ϕ is a formula whose atoms are of the form $Eq(q_i, q_j)$, with $q_i \rightarrow q_j \in \varepsilon^k_E$. Pruning $A^k_{\mathcal{R},E}$ remains to remove transitions $q'_i \rightarrow q'_j$ from ε^k_E until ϕ does not hold anymore i.e. $\phi = \bot$. Since ε^k_E is finite, the pruning process always terminates.

We now show that the pruning process removes any spurious counter-example using etablished results in Appendix C and Apendix D.

Now, let us show that the pruning process removes a given spurious counter-example from $\mathcal{L}(A^k_{\mathcal{R},E})$.

Proof. According to Lemma 1, the matching algorithm is complete. So given a term $t \in \mathcal{T}(\mathcal{F})$, if $t \in \mathcal{L}(A^k_{\mathcal{R},E})$ then there exists $q \in Q_f$ such that $t \rightarrow^*_{A^k_{\mathcal{R},E}} q$. According to Theorem 1, there exists α such that $t \xrightarrow{\phi}_{A^k_{\mathcal{R},E}} q$. More precisely, using Definition 7 given in Appendix C, one can deduce that $(\alpha, \emptyset) \in S$ with $t \trianglelefteq q \vdash_{A^k_{\mathcal{R},E}} S$. Let ϕ be the following formula: $\phi = \bigvee_{(\alpha', \emptyset) \in S} (\alpha')$. Consequently, ϕ is a formula characterizing all possible reductions of t into q. Since t is a spurious counter-example, for all $(\alpha', \emptyset) \in S$, $\alpha' \neq \top$. Removing transitions of ε^k_E until ϕ does not hold remains to remove each possible reduction in $A^k_{\mathcal{R},E}$ of t into q. As a conclusion, when the pruning process terminates, t is not recognized anymore.

H Computing the Intersection of a Tree Automaton and a $\mathcal{R}_{/E}$-Automaton

In Definition 10, we propose a specific algorithm building the set S of reachable states for the intersection between a $\mathcal{R}_{/E}$-automaton A and automaton B where each product

state is labelled by a formula on states of A. As stated in Section 6, this is useful to characterize the possibly infinite set of terms that have to be refined in a single step. In addition, Lemma 4 proposes a methodology to decide whether the intersection is empty or not.

We first define an order $>$ on formulas.

Definition 9. *Given ϕ_1 and ϕ_2 two formulas, $\phi_1 > \phi_2$ iff $\phi_2 \models \phi_1$ and $\phi_1 \not\models \phi_2$.*

Definition 10 (Reachable states of the product of a $\mathcal{R}_{/E}$-automaton and a tree automaton). *Let $A = \langle \mathcal{F}, Q^A, Q^A_f, \Delta^A, \varepsilon_{\mathcal{R}}, \varepsilon_E \rangle$ be a $\mathcal{R}_{/E}$-automaton and $B = \langle \mathcal{F}, Q^B, Q^B_f, \Delta^B \rangle$ be an epsilon-free tree automaton. The set S of reachable states of $A \times B$ is the set of triples (q, q', ϕ) where $q \in Q^A, q' \in Q^B$ and ϕ is a formula. Starting from the set $Q^A \times Q^B \times \{\bot\}$, the value of S can be computed using the following two deduction rules :*

$$\frac{\{(q_1, q'_1, \phi_1), \ldots, (q_n, q'_n, \phi_n)\} \cup \{(q, q', \phi)\} \cup P}{\{(q_1, q'_1, \phi_1), \ldots, (q_n, q'_n, \phi_n)\} \cup \{(q, q', \phi \vee \bigwedge_{i=1}^{n} \phi_i)\} \cup P} \quad \begin{array}{c} \textit{if } f(q_1, \ldots, q_n) \to q \in \Delta^A \\ \textit{and } f(q'_1, \ldots, q'_n) \to q' \in \Delta^B \\ \textit{and } (\phi \vee \bigwedge_{i=1}^{n} \phi_i) > \phi \end{array}$$

$$\frac{\{(q_1, q, \phi_1), (q_2, q, \phi_2)\} \cup P}{\{(q_1, q, \phi_1), (q_2, q, (\phi_1 \wedge \phi) \vee \phi_2)\} \cup P} \quad \begin{array}{l} \textit{if } q_1 \xrightarrow{\phi} q_2 \in \varepsilon_{\mathcal{R}} \textit{ and } ((\phi_1 \wedge \phi) \vee \phi_2) > \phi_2 \\ \textit{or} \\ \textit{if } q_1 \to q_2 \in \varepsilon_E \textit{ and } \phi = Eq(q_1, q_2) \\ \textit{and } ((\phi_1 \wedge \phi) \vee \phi_2) > \phi_2 \end{array}$$

With regards to the reachability problem, this definition, provides a way to distinguish between real counterexamples and terms which can be rejected using abstraction refinement. Indeed, for all triple $(q, q', \phi) \in S$ with q final in A and q' final in B, if $\phi \models \top$ then some of the terms recognized by q' in B are reachable. Otherwise, ϕ is the formula to invalidate, i.e. negate some of its atom so that it becomes $\bot$.

Lemma 4 (Emptiness decision of the product of a $\mathcal{R}_{/E}$-automaton and a tree automaton). *Let A be a $\mathcal{R}_{/E}$-automaton and B a tree automaton. Let S be the set of reachable states of $A \times B$ defined according to definition 10. For all final state q of A, all final state q' of B, all formulas $\phi_S \neq \bot$, $\phi \neq \bot$ and all term $t \in \mathcal{T}(\mathcal{F})$, we have $t \xrightarrow{\phi}{}^*_A q$ and $t \to^*_B q'$ (i.e. $\mathcal{L}(A) \cap \mathcal{L}(B) \neq \emptyset$) if and only if there exists a triple $(q, q', \phi_S) \in S$ such that $\phi \models \phi_S$.*

Proof. Let $A = \langle \mathcal{F}, Q^A, Q^A_f, \Delta^A, \varepsilon_{\mathcal{R}}, \varepsilon_E \rangle$ be the $\mathcal{R}_{/E}$-automaton and $B = \langle \mathcal{F}, Q^B, Q^B_f, \Delta^B \rangle$ be the tree automaton. We prove a stronger property on all states q of A and q' of B (and not only for final states). First, we prove the 'only if' part. Let us assume that there exists a term $t \in \mathcal{T}(\mathcal{F})$ such that $t \xrightarrow{\phi}{}^*_A q$, $t \to^*_B q'$. By induction on the height of t we have:

- If t is a constant, since B is an epsilon-free tree automaton, the only way to have $t \to^*_B q'$ is to have $t \to q' \in B$. With regards to A, by definition 3, $t \xrightarrow{\phi}{}^*_A q$ means that there exists states $q_0, q_1, \ldots, q_n$ and formulas $\phi_1, \ldots, \phi_n$ such that $t \to_{\Delta_A} q_0 \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \ldots q_n$ with $q = q_n$ and $\phi = \phi_1 \wedge \ldots \wedge \phi_n$. Transitions $q_i \xrightarrow{\phi_i} q_{i+1}$ are either transitions of $\varepsilon_{\mathcal{R}}$ or transitions of ε_E with $\phi_i = \top$. Because of transitions $t \to q_0 \in \Delta_A$ and $t \to q' \in \Delta_B$, using the first case of definition 10, we get that $(q_0, q', \top) \in S$. Similarly, using the second case of the definition, we obtain

that there exists formulas ϕ'_i with $i = 1 \dots n$ such that $(q_1, q', \phi_1 \vee \phi'_1), (q_2, q', (\phi_1 \wedge \phi_2) \vee \phi'_2), \dots (q_n, q', (\phi_1 \wedge \dots \wedge \phi_n) \vee \phi'_n)$ belong to S. Finally, since $q_n = q$ and $\phi = \phi_1 \wedge \dots \wedge \phi_n$, we that $(q, q', \phi \vee \phi'_n) \in S$. Furthermore, we trivially have that $\phi_S = \phi \vee \phi'_n$ and $\phi \models \phi_S$.

- Assume that for all term of height lesser or equal to $n \in \mathbb{N}$, the property is true. Let us prove that it is also true for a term $f(t_1, \dots, t_n)$ with $t_1, \dots, t_n$ of height lesser or equal to n. Since $f(t_1, \dots, t_n) \rightarrow^*_B q'$ and B is an epsilon free tree automaton, we obtain that $\exists q'_1, \dots, q'_n \in Q^B$ such that $\forall i = 1 \dots n : t_i \rightarrow^*_B q'_i$ and $f(q'_1, \dots, q'_n) \rightarrow q' \in \Delta_B$. With regards to A, by definition 3, $f(t_1, \dots, t_n) \xrightarrow{\phi}{}^*_A q$ means that there exists states $q_0, q_1, \dots, q_m, q''_1, \dots, q''_n$ and formulas $\phi_1, \dots, \phi_m, \phi'_1, \dots, \phi'_n$ such that $\forall i = 1 \dots n : t_i \xrightarrow{\phi'_i}{}^*_A q''_i$, $f(q''_1, \dots, q''_n) \rightarrow_{\Delta_A} q_0$ and $q_0 \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \dots q_n$, $q = q_n$. Furthermore, we obtain that $\phi = \bigwedge_{i=1}^n \phi'_i \wedge \bigwedge_{i=1}^m \phi_i$.
  Since terms t_i are of height lesser or equal to n, $\forall i = 1 \dots n : t_i \rightarrow^*_B q_i$ and $\forall i = 1 \dots n : t_i \xrightarrow{\phi'_i}{}^*_A q''_i$, we can apply the induction hypothesis and obtain that $\forall i = 1 \dots n : (q_i, q''_i, \phi''_i) \in S$ with $\phi'_i \models \phi''_i$. Besides to this, using case 1 of definition 3 on $f(q_1, \dots, q_n) \rightarrow q' \in \Delta_B$, $f(q''_1, \dots, q''_n) \rightarrow q_0 \in \Delta_A$, and $\forall i = 1 \dots n : (q_i, q''_i, \phi''_i) \in S$, we obtain that there exists a formula ϕ' such that $(q_0, q', (\bigwedge_{i=1}^n \phi''_i) \vee \phi') \in S$. Then, like in the base case, since $q_0 \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \dots q_n$, $q = q_n$, we can deduce that there exists a formula ϕ'' such that $(q, q', (\bigwedge_{i=1}^n \phi''_i \wedge \bigwedge_{i=1}^m \phi_i) \vee \phi'') \in S$. Let $\phi_S = (\bigwedge_{i=1}^n \phi''_i \wedge \bigwedge_{i=1}^m \phi_i) \vee \phi''$. Since we know from above that $\phi = \bigwedge_{i=1}^n \phi'_i \wedge \bigwedge_{i=1}^m \phi_i$ and $\forall i = 1 \dots n : \phi'_i \models \phi''_i$, we obtain that $\phi \models \phi_S$.

Second, we prove the 'if' part: if $(q, q', \phi_S) \in S$ and $\phi_S \neq \bot$ then there exists a term t and a formula $\phi \neq \bot$ such that $\phi \models \phi_S$, $t \xrightarrow{\phi}{}^*_A q$ and $t \rightarrow^*_B q'$. We make a proof by induction on the number of applications of the two rules of definition 10, necessary to prove that (q, q', ϕ_S) in S.

- If the number of steps is 0 then, since the computation of S starts from the set $Q^A \times Q^B \times \bot$, then all (q, q', ϕ_S) are such that $\phi_S = \bot$, which is a contradiction.
- We assume that the property is true for any triple (q, q', ϕ_S) which can be deducted by n or less applications of the rules of definition 10. Now, we consider the case of a triple (q, q', ϕ_S) that is deduced at the $n+1$-th step of application of the deduction rules.
  - If the first rule is concerned, this means that there exists triples $(q_1, q'_1, \phi_1), \dots, (q_n, q'_n, \phi_n)$ and (q, q', ϕ) in S deduced before $n+1$-th step, as well as transitions $f(q_1, \dots, q_n) \rightarrow q \in \Delta_A$ and $f(q'_1, \dots, q'_n) \rightarrow q' \in \Delta_B$. Furthermore, we know that $\phi_S = \phi \vee \bigwedge_{i=1}^n \phi_i$. If $\phi \neq \bot$ then, since (q, q', ϕ) was shown to belong to S before $n+1$-th step, we can apply the induction hypothesis and directly obtain that there exists a term t and a formula ϕ' such that $\phi' \models \phi$, $t \xrightarrow{\phi'}{}^*_A q$ and $t \rightarrow^*_B q'$. Note that $\phi' \models \phi$ implies $\phi' \models \phi_S$. Otherwise, if $\phi = \bot$, then we can apply the induction hypothesis on triples (q_i, q'_i, ϕ_i), $i = 1 \dots n$ and obtain that $\forall i = 1 \dots n : \exists \phi'_i : \exists t_i \in \mathcal{T}(\mathcal{F}) : \phi'_i \models \phi_i$, $t_i \xrightarrow{\phi'_i}{}^*_A q_i$ and $t_i \rightarrow^*_B q'_i$. Finally, because of the two transitions $f(q_1, \dots, q_n) \rightarrow q \in \Delta_A$ and $f(q'_1, \dots, q'_n) \rightarrow q' \in \Delta_B$, we get that $f(t_1, \dots, t_n) \xrightarrow{\phi'}{}^*_A f(q_1, \dots, q_n) \rightarrow^*_A q$ with $\phi' = \bigwedge_{i=1}^n \phi'_i$ on one side and $f(t_1, \dots, t_n) \rightarrow_B f(q'_1, \dots, q'_n) \rightarrow^*_B q$ on the other side. Furthermore, since $\forall i = 1 \dots n : \phi'_i \models \phi_i$, we have $\bigwedge_{i=1}^n \phi'_i \models \bigwedge_{i=1}^n \phi_i$. Recall that $\phi' = \bigwedge_{i=1}^n \phi'_i$ and $\phi_S = \phi \vee \bigwedge_{i=1}^n \phi_i$. Hence, $\phi' \models \phi_S$.

- If the second rule is concerned, this means that there exists triples (q_1, q', ϕ_1) and (q, q', ϕ_2) in S deduced before the $n+1$-th step. Furthermore, we know that $\phi_S = (\phi_1 \wedge \phi) \vee \phi_2$. Like above, if $\phi_2 \neq \bot$ then we can apply induction hypothesis on (q, q', ϕ_2) and trivially get the result. Otherwise, if $\phi_2 = \bot$ then we can use induction hypothesis on the triple (q_1, q', ϕ_1) and obtain that there exists a formula ϕ_1' and a term t_1 such that $t_1 \xrightarrow{\phi_1'}{}^*_A q_1$, $t_1 \to^*_B q'$ and $\phi_1' \models \phi_1$. Then, by case on the epsilon transition used for the deduction on S, we prove that $t_1 \xrightarrow{\phi_1' \wedge \phi}{}^*_A q$:
  - Assume that $q_1 \xrightarrow{\phi} q \in \varepsilon_{\mathcal{R}}$. Then, by definition 3, we obtain that $t_1 \xrightarrow{\phi_1' \wedge \phi}{}^*_A q$. Furthermore, since $\phi_1' \models \phi_1$, we have that $\phi_1' \wedge \phi \models \phi_1 \wedge \phi$ and, finally, that $\phi_1' \wedge \phi \models \phi_S$.
  - Assume that $q_1 \to q \in \varepsilon_E$. By definition 3, we obtain that $t \xrightarrow{\phi_1 \vee Eq(q_1, q)}{}^*_A q$. Finally, like above, we can deduce that $\phi_1' \wedge Eq(q_1, q) \models \phi_1 \wedge Eq(q_1, q)$ and thus $\phi_1' \wedge Eq(q_1, q) \models \phi_S$.

I Soundness and Completeness

Concerning the prune step P, it preserves also the well-definition of a $\mathcal{R}_{/E}-$automaton. Indeed, given an $\mathcal{R}_{/E}-$automaton $A^i_{\mathcal{R},E}$, one can see the prune step as a removing of transitions of ε^i_E and $\varepsilon^i_{\mathcal{R}}$. So, it does not affect Definition 4.

I.1 Proof of Theorem 5

The completion stops when any critical pair is solved. We show that this automaton is $\mathcal{R}-$closed for any state of $A^*_{\mathcal{R},E}$ the fixpoint $\mathcal{R}_{/E}$-automaton. Since $\mathtt{C}(A^*_{\mathcal{R},E}) = A^*_{\mathcal{R},E}$, for any q of $A^*_{\mathcal{R},E}$, any rule $l \to r \in \mathcal{R}$ and any substitution $\sigma : \mathcal{X} \mapsto Q$, one has $l\sigma \to^*_{A^*_{\mathcal{R},E}} q \Rightarrow r\sigma \to^*_{A^*_{\mathcal{R},E}} q$. Applying Theorem 2, one has for any state q of $A^*_{\mathcal{R},E}$, any $u \in \mathcal{L}(A^*_{\mathcal{R},E}, q)$, any $v \in \mathcal{T}(\mathcal{F})$, $u \to_{\mathcal{R}} v \implies v \in \mathcal{L}(A^*_{\mathcal{R},E}, q)$. Thanks to the fixpoint property, this result extends to the reflexive and transitive closure of $\to_{\mathcal{R}}$ for all state, and especially final states of $A^*_{\mathcal{R},E}$. We then deduce $\mathcal{R}^*(\mathcal{L}(A^*_{\mathcal{R},E})) \subseteq \mathcal{L}(A^*_{\mathcal{R},E})$. Finally, all the transitions of the initial automaton $A^0_{\mathcal{R},E}$ are always preserved at each step of C and W, and never removed by P.

Similarly, the set of final states is never modified during the process. It means that we have a syntactical inclusion of all transitions of $A^0_{\mathcal{R},E}$ in $A^*_{\mathcal{R},E}$, and thus $\mathcal{L}(A_0) \subseteq \mathcal{L}(A^*_{\mathcal{R},E})$. The conjunction of this last result with $\mathcal{R}^*(\mathcal{L}(A^*_{\mathcal{R},E})) \subseteq \mathcal{L}(A^*_{\mathcal{R},E})$ is sufficient to conclude that if $\mathcal{L}(A^*_{\mathcal{R},E}) \cap \mathcal{L}(A_{Bad}) = \emptyset$ then $\mathcal{R}^*(\mathcal{L}(A_O)) \cap \mathcal{L}(A_{Bad}) = \emptyset$.

I.2 Proof of Theorem 6

Since P preserves the well-definition of a $\mathcal{R}_{/E}-$automaton, one can deduce that any $\mathcal{R}_{/E}-$automaton A_i is then well-defined for linear TRS.

Let $t \in Bad$ and $t \in \mathcal{R}^*(\mathcal{L}(A_0))$. Note that if A_i exists then necessarly, in order to get $t \xrightarrow{\top} q_f$ and q_f a final state of A_i, A_i must contain a rewriting chain $t_0 \to_{\mathcal{R}} t_1 \ldots \to_{\mathcal{R}} t_n$ such that $t_0 \in \mathcal{L}(A_0)$, $t_n = t$ and for any t_i, $t_i \xrightarrow{\top} q_f$. Let $t \in Bad$ and $t \in \mathcal{R}^*(\mathcal{L}(A_0))$. Thus, there exists a minimal rewriting chain $t_0 \to_{\mathcal{R}} t_1 \ldots \to_{\mathcal{R}} t_n$ such that $t_0 \in \mathcal{L}(A_0)$ and $t_n = t$. We are going to show that a such rewriting chain leads eventually to the wanted $\mathcal{R}_{/E}-$automaton A_j.

$n = 0$ By hypothesis, $t_0 \in \mathcal{L}(A_0) \cap Bad$. Since $\varepsilon_E^0 = \emptyset$, one can deduce that there exists $q_f \in Q_f$ such that $t_0 \xrightarrow{\top}_{A_0} q_f$. So, A_0 is the wanted $\mathcal{R}_{/E}-$automaton.

$n + 1$ Suppose that the property is true for a rewriting chain of length n. Then, there exists a $\mathcal{R}_{/E}-$automaton A_k such that for any t_i with $i = 0, \ldots, n$, $t_i \xrightarrow{\top}_{A_k} q_f$ and $t_n \notin \mathcal{L}(A_{k-1})$. Consider now a rewriting chain of length $n + 1$. So, one has $t_0 \to_{\mathcal{R}} t_1 \ldots \to_{\mathcal{R}} t_n \to_{\mathcal{R}} t_{n+1}$. By construction, t_{n+1} can not be in $\mathcal{L}(A_k)$. Indeed, the rewriting chain is the minimal one to get t_{n+1}. So, if t_{n+1} was in a previous $\mathcal{R}_{/E}-$automaton then it has been deleted because t_{n+1} would have been obtained by accelerating the computation and then considered as a spurious example. Thus, A_k is not $\mathcal{R}-$closed. So, applying Theorem 2 one gets that $t_{n+1} \in \mathcal{L}(\mathtt{C}(A_k))$. Moreoever, since there exists $q_f \in Q_f$ such that $t_n \xrightarrow{\top} q_f$, one can deduce that there exists $u, v \in \mathcal{T}(\mathcal{F})$, $p \in \mathcal{P}os(t_n)$ and $q \in Q^k$ such that $u \xrightarrow{\top}_{A_k} q$, $t_n|_p = u$, $u \to_{\mathcal{R}} v$ and $t_{n+1} = t_n[v]_p$. Consequently, there exists a rule $l \to r \in \mathcal{R}$ and a substitution $\sigma : \mathcal{X} \to Q^k$ such that $u \xrightarrow{\top}_{A_k} l\sigma$ and $v \xrightarrow{\top}_{A_k} r\sigma$. So, one can deduce that $t_{n+1} \xrightarrow{\top}_{\mathtt{C}(A_k)}$. Trivially, if $t_{n+1} \in Bad$ then there exists $j > 0$ such that $t_{n+1} \xrightarrow{\top}_{A_j}$.

I.3 Proof of Theorem 7

Since $\mathcal{R}$ is not linear, we can not applied directly Theorem 2.

We illustrate the problem of the non right-linearity of $\mathcal{R}$. Let A be the $\mathcal{R}_{/E}-$automaton whose $\Delta = \{a \to q', b \to q' f(q') \to q\}$, $\varepsilon_{\mathcal{R}} = \emptyset$ and $\varepsilon_E = \emptyset$. Let $\mathcal{R}$ be the TRS composed of a single rule $f(x) \to g(x, x)$. To compute $\mathtt{C}(A)$, we have to add the new transitions $g(q', q') \to q''$ and $q'' \xrightarrow{\top} q$. We have the new runs $g(a, a) \xrightarrow{\top} q$, $g(b, b) \xrightarrow{\top} q$, $g(a, b) \xrightarrow{\top} q$ and $g(b, a) \xrightarrow{\top} q$. but we note that terms $g(a, b)$ and $g(b, a)$ are not reachable from $f(a)$ or $f(b)$. $\mathcal{R}_{/E}$-automaton $\mathtt{C}(A)$ is not well-defined anymore. The reason is that several representative terms are in $Rep(q')$: it implies the term $g(q', q')$ denotes more than the expected terms. To avoid this approximation, it is sufficient to ensure that all q has a unique representative term the initial RE-automaton. The normalization defined in Appendix D maintains this property by producing a fresh state for each new transition created.

J Additional Experiments

J.1 Processes Counting Symbols

The example considered in [11] is linear. The example deals with a simple two processes counting system. The following TRS describes the behavior of two processes each one equipped with an input list and a FIFO. Each process receives a list of symbols '+' and '−' to count, as an input. One of the processes, say P_+, is counting the '+' symbols and the other one, say P_- is counting the '−' symbols. When P_+ receives a '+', it counts it and when it receives a '−', it adds the symbol to P_-'s FIFO. The behavior of P_- is symmetric. When a process input list and FIFO is empty then it stops and gives the value of its counter.

Here is a possible rewrite specification of this system, given in the Timbuk language, where `S(_,_,_,_)` represents a configuration with a process P_+, a process P_-, P_+'s FIFO and P_-'s FIFO. The term `Proc(_,_)` represents a process with an input list and

a counter, `add(_,_)` implements adding of an element in a FIFO, and `cons`, `nil`, `s`, `o` are the usual constructors for lists and natural numbers (peano's representation). The symbols '+' and '−' are represented respectively by the terms `plus` and `minus`. When a process has terminated its task, the value returned by the process is represented by a term of the form `stop(i)` where `i` is a peano's integer.

A first TRS representing this system is given in Section J.2.

Our objective is to show that no deadlock state can be reached. A deadlock state is a state when the process has stopped but there are still symbols to count.

The set of bad terms is also defined by a tree automaton recognizing all terms of the form `S(stop(_),_,cons(plus,_),_)` and `S(_,stop(_),_,cons(minus,_))`, i.e. any configuration where a stopped process has a non empty FIFO. What remains to be done is to provide the equations so as to have a finite model. Since each process reads symbols in an unbounded list and either counts it or adds it to the other's process FIFO, the terms which are likely to become infinite are counters and FIFOs. To have a finite completion, it is enough to add the following equations: `s(X)=X` whose effect is to place all natural numbers in the same equivalence class and `add(X,add(Y,Z))=add(X,Z)` whose effect is to place all terms built on FIFO additions in the same equivalence class. If we launch TimbukCEGAR on this example it stops in less than a second and states that there is a counterexample. This is due to the fact that after P_+ termination, P_- may add a `"+"` to P_+'s FIFO. This can be solved by a more precise termination condition for each process: when a process exhausts its list it adds an `end` symbol to the FIFO of the other process. A process terminates if it exhausts its list and if it reads `end` on its FIFO. This patched specification is given in Section J.3.

Refining the termination condition in the TRS of the Timbuk's specification, we can restart completion. Then no counterexample is found but a refinement is necessary. This is due to the fact that the equation `add(X,add(Y,Z))=add(X,Z)` may break the order of symbols in the FIFO. Thus, using this equation, `"+"` or `"-"` symbols may be occur in the FIFO after the `end` symbol, resulting in a false counterexample. To achieve the proof using Timbuk 3.1, it was necessary to guess a good set of equations, which is hard in general. On this particular example, it is necessary to figure out that the equation `add(X,add(Y,Z))=add(X,Z)` is likely to mix additions of `"+"`, `"-"` and `end` symbols. Hence, replacing this equation by the three equations `add(plus,add(plus,Z))=add(plus,Z)`, `add(minus,add(minus,Z))=add(minus,Z)` and `add(end,add(end,Z))=add(end,Z)` avoid this problem and permits to have a terminating completion and no false counterexamples. Tuning of equations by hand is hard on large examples and can be avoided here using automatic refinement. TimbukCEGAR and the initial equation proves it in 8 steps of completion and 5 steps of refinement. All the results can be certified by the checker.

The table of Figure 3 gives the number of rules of the TRS, the number of transitions of initial tree automaton, the number of refinement steps, the size of the completed automaton, execution time and memory usage for completion, and checking time. Line 1 gives the results for the automatic refinement of the initial equation. The same example is run with the three good equations (no refinement necessary) on line 2 and with Timbuk 3.1 on line 3.

The completion time with refinement (255s) has to be compared with the execution time of [11] which is superior to one hour[7].

[7] Personnal communication with the authors, not given in the paper.

Tool	$\mathcal{R}$ nb of rules	Initial TA nb of trans.	Ref. steps	Final TA nb of trans.	Comp. time	Comp. memory	Checking time
TimbukCEGAR	12	8	5	4145	255s	170Mb	12504s
TimbukCEGAR	12	8	0	1996	20s	63Mb	693s
Timbuk 3.1	12	8	0	12219	19s	25Mb	21s
TimbukCEGAR	879	14	4	3688	128s	531Mb	17017s

Fig. 3: Time and memory usage for completion with and without refinement.

J.2 Timbuk Specification leading to a Counterexample

```
Ops
  S:4 proc:2 stop:1 cons:2 nil:0 plus:0 minus:0 s:1 o:0 add:2

Vars X Y Z U C M N

TRS R1

add(X,nil) -> cons(X,nil)
add(X,cons(Y,Z)) -> cons(Y,add(X,Z))
S(proc(cons(plus,Y),C),Z,M,N) -> S(proc(Y,s(C)),Z,M,N)
S(proc(cons(minus,Y),C),U,M,N) -> S(proc(Y,C),U,M,add(minus,N))
S(X,proc(cons(minus,Y),C),M,N) -> S(X,proc(Y,s(C)),M,N)
S(X,proc(cons(plus,Y),C),M,N) -> S(X,proc(Y,C),add(plus,M),N)
S(proc(X,C),Z,cons(plus,M),N) -> S(proc(X,s(C)),Z,M,N)
S(X,proc(Z,C),M,cons(minus,N)) -> S(X,proc(Z,s(C)),M ,N)
S(proc(nil,C),Z,M,N)  -> S(stop(C),Z,M,N)
S(X,proc(nil,C),M,N) -> S(X,stop(C),M,N)


Automaton A1

States q0 qinit qzero qnil qlist qsymb

Final States q0

Transitions
  o -> qzero                          nil -> qnil
  plus -> qsymb                       minus -> qsymb
  cons(qsymb,qnil) ->qlist           cons(qsymb,qlist) -> qlist
  proc(qlist,qzero) -> qinit         S(qinit,qinit,qnil,qnil) -> q0
```

J.3 Timbuk Specification leading to a Conclusive Analysis

```
Ops
  S:4 proc:2 stop:1 cons:2 nil:0 plus:0 minus:0 s:1 o:0 end:0 add:2

Vars X Y Z U C M N
```

TRS R1

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

Automaton A1

States q0 qinit qzero qnil qlist qsymb

Final States q0

Transitions

```
  o -> qzero                          nil -> qnil
  plus -> qsymb                       minus -> qsymb
  cons(qsymb,qnil) ->qlist           cons(qsymb,qlist) -> qlist
  proc(qlist,qzero) -> qinit         S(qinit,qinit,qnil,qnil) -> q0
```