Certifying a Tree Automata Completion Checker

Benoît Boyer, Thomas Genet, Thomas Jensen

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Certifying a Tree Automata Completion Checker

Benoît Boyer — Thomas Genet — Thomas Jensen

N° 6462

21 février 2008

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6462--FR+ENG

Certifying a Tree Automata Completion Checker

Benoît Boyer , Thomas Genet , Thomas Jensen

Thème SYM — Systèmes symboliques
Projet Lande

Rapport de recherche n° 6462 — 21 février 2008 — 26 pages

Abstract: Tree automata completion is a technique for the verification of infinite state systems. It has already been used for the verification of cryptographic protocols and the prototyping of Java static analysers. However, as for many other verification techniques, the correctness of the associated tool becomes more and more difficult to guarantee. It is due to the size of the implementation that constantly grows and due to low level optimizations which are necessary to scale up the efficiency of the tool to verify real-size systems. In this paper, we define and develop a checker for tree automata produced by completion. The checker is defined using Coq and its implementation is automatically extracted from its formal specification. Using extraction gives a checker that can be run independently of the Coq environment. A specific algorithm for tree automata inclusion checking have been defined so as to avoid the exponential blow up. The obtained checker is certified in Coq, independant of the implementation of completion, usable with any approximation performed during completion, small and fast. Some benchmarks are given to illustrate the efficiency of the tool.

Key-words: Term Rewriting Systems,Tree Automata, Tree Automata Completion, Certification, Coq, Reachability

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Certification d'un vérificateur de complétion d'automates

Résumé : La complétion d'automates d'arbres est une technique de vérification pour les systèmes infinis. Celle-ci a déjà été utilisée pour la vérification des protocoles cryptographiques et le prototypage d'analyseurs statiques. Cependant, comme dans le cas de beaucoup d'autres techniques de vérification, la correction des outils associés devient de plus en plus difficile à garantir. Ceci est dû, d'une part à la constante augmentation de la taille de ces logiciels et, d'autre part, aux optimisations de bas niveau nécessaires pour améliorer leurs performances. Dans cet article, nous définissons un vérificateur pour les automates d'arbres produits par complétion. Le vérificateur est défini en Coq et son implantation est automatiquement extraite de sa spécification formelle. L'extraction permet d'obtenir un vérificateur qui peut être exécuté indépendamment de Coq. Nous avons défini un algorithme d'inclusion spécifique qui permet de tester l'inclusion de langages régulier. Cet algorithme n'est pas complet en général mais l'est dans notre cas. Le vérificateur complet est certifié en Coq, indépendant du type d'implantation de la complétion, utilisable avec n'importe quelle approximation appliquée pendant la complétion, peut gourmand en mémoire et rapide. Ces résultats sont illustrés sur des jeux d'essai.

Mots-clés : Systèmes de réécriture, Automate d'arbre, complétion d'automates d'arbres, certification, Coq, atteignabilité

1 Introduction

In the field of infinite system verification, three of the most successful techniques are: assisted proof, abstract interpretation and symbolic/abstract model-checking. In all those techniques, the verification relies on specific softwares that may, themselves, contain bugs. On the one hand, a proof assistant like Coq [BC04] avoids this problem since any proof, on any exotic domain, is finally checked using a small unique certified kernel. On the other hand, proof assistants like Coq offer poor automation when compared with fully automatic tools like static analyzers or model-checkers. However, the efforts achieved on the two last ones, to obtain a better automation and efficiency, have a great impact on their reliability. Static analyzers and model-checkers are usually huge, drastically optimized programs whose safety is not proved but essentially "demonstrated" by an extensive use.

Static program analysis is one of the cornerstones of software verification and is increasingly used to protect computing devices from malicious or malfunctioning code. However, program verifiers are themselves complex programs and a single error may jeopardize the entire trust chain of which they form part. Efforts have been made to certify static analyzers [KN03, BD04, CJPR05] or to certify the results obtained by static analyzers [LT00, BJP06] in Coq in order to increase confidence in the analyzers. In this paper, we instantiate the general framework used in [BJP06] to the particular case of analyzing term rewriting systems by tree automata completion [Gen98, FGVTT04]. Given a term rewriting system, the tree automata completion is a technique for over-approximating the set of terms reachable by rewriting in order to prove the unreachability of certain "bad" states that violates a given security property. This technique has already been used to prove security properties on cryptographic protocols [GK00], [GTTVTT03, BHK04, ABB$^+$05, ZD06] and, more recently, to prototype static analyzers on Java byte code [BGJL07].

In this paper, we show how to mechanize the proof, within the proof assistant Coq, that the tree automaton produced by completion recognizes an over-approximation of all reachable terms. Coq is based on constructive logic (Calculus of Inductive Constructions) and it is possible to extract an Ocaml or Haskell function implementing exactly the algorithm whose specification has been expressed in Coq. The extracted code is thus a *certified* implementation of the specification given in the Coq formalism. Extracted programs are standalone and do not require the Coq environment to be executed. For details about the extraction mechanisms, readers can refer to [BC04].

A specific challenge in the work reported here has been how to marry constructive logic and efficiency. Previous case studies with tree automata completion, on cryptographic protocols [GTTVTT03] and on Java bytecode [BGJL07] show that we need an efficient completion algorithm to verify properties of real models. For instance, the current implementation of completion (called Timbuk [GVTT00]) is based on imperative data structures like hash tables whereas Coq allows only pure functional structures. A second problem is the termination of completion. Since Coq can only deal with total functions, functions must be

proved terminating for any computation. In general, such a property cannot be guaranteed on completion because it mainly depends on term rewriting system and approximation equations given initially.

For these two reasons, there is little hope to specify and certify an efficient and purely functional version of the completion algorithm. Instead, we have adopted a solution based on a result-checking approach. It consists of building a smaller program (called the *checker*) - certified in Coq - that checks if the tree automaton computed by Timbuk is sound. In this paper, we restrict to the case of left-linear term rewriting systems which revealed to be sufficient for verifying Java programs [BGJL07]. However, a checker dealing with general term rewriting systems like completion does in [FGVTT04] is under development.

The closest work to ours is the one done by X. Rival and J. Goubault-Larrecq [RGL01]. They have designed a library to manipulate tree automata in Coq and proposed some optimized formal data structures that we reuse. However, we aim at dealing with larger tree automata than those used in their benchmarks. Moreover, we need some other tools which are not provided by the library as for example a specific algorithm to check inclusion. Inclusion checking may be done using closure operators (i.e. intersection and complementation) and emptiness checking but, as shown in Section 7, this way of performing inclusion checking has a bad complexity.

This paper is organized as follows. Rewriting and tree automata are reviewed in Section 2 and tree automata completion in Section 3. Section 4 states the main functions to define, inclusion and closure test, and the corresponding theorems to prove. Section 5 and Section 6 give the Coq formalization of rewriting and of tree automata, respectively. The core of the checker consists of two algorithms: an optimized automata inclusion test, defined in Section 7, and a procedure for checking that an automaton is *closed* under rewriting w.r.t. a given term rewriting system, defined in Section 8. An important feature of the inclusion checker is that, while it is not complete for all tree automata, we can prove that it is complete for the class of tree automata generated by the completion algorithm. Section 9 gives some details about the performances of the checker in practice. Finally, we conclude and list some on-going research on this subject.

2 Preliminaries

Comprehensive surveys can be found in [BN98] for rewriting, and in [CDG+02, GT95] for tree automata and tree language theory.

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 extended uniquely 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 ϵ denotes the

top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by:

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

If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. A term rewriting system $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* 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 whose reflexive transitive closure is denoted by $\to^\star_{\mathcal{R}}$. The set of $\mathcal{R}$-descendants of a set of ground terms E is $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to^\star_{\mathcal{R}} t\}$.

The *verification technique* defined in [Gen98, FGVTT04] is based on $\mathcal{R}^*(E)$. Note that $\mathcal{R}^*(E)$ is possibly infinite: $\mathcal{R}$ may not terminate and/or E may be infinite. The set $\mathcal{R}^*(E)$ is generally not computable [GT95]. However, it is possible to over-approximate it [Gen98, FGVTT04, Tak04] using tree automata, i.e. a finite representation of infinite (regular) sets of terms. In this verification setting, the TRS $\mathcal{R}$ represents the system to verify, sets of terms E and Bad represent respectively the set of initial configurations and the set of "bad" configurations that should not be reached. Then, using tree automata completion, we construct a tree automaton $\mathcal{B}$ whose language $\mathcal{L}(\mathcal{B})$ is such that $\mathcal{L}(\mathcal{B}) \supseteq \mathcal{R}^*(E)$. Then if $\mathcal{L}(\mathcal{B}) \cap Bad = \emptyset$ then this proves that $\mathcal{R}^*(E) \cap Bad = \emptyset$, and thus that none of the "bad" configurations is reachable. We now define tree automata.

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

Definition 1 (Transition and normalized transition) *A* transition *is a rewrite rule $c \to q$, where c is a configuration i.e. $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. A* normalized transition *is a transition $c \to q$ where $c = f(q_1, \ldots, q_n)$, $f \in \mathcal{F}$ whose arity is n, and $q_1, \ldots, q_n \in \mathcal{Q}$.*

Definition 2 (Bottom-up nondeterministic finite tree automaton) *A bottom-up nondeterministic finite tree automaton (tree automaton for short) is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \Delta \rangle$, where $\mathcal{Q}_F \subseteq \mathcal{Q}$ and Δ is a set of normalized transitions.*

The *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ induced by the transitions of $\mathcal{A}$ (the set Δ) is denoted by $\to_\Delta$. When Δ is clear from the context, $\to_\Delta$ will also be denoted by $\to_{\mathcal{A}}$.

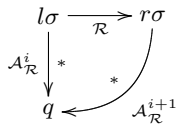
Definition 3 (Recognized language) *The tree language recognized by $\mathcal{A}$ in a state q is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to^\star_{\mathcal{A}} q\}$. The language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_F}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

Example 1 *Let $\mathcal{A}$ be the tree automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \Delta \rangle$ such that $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_F = \{q_0\}$ and $\Delta = \{f(q_0) \to q_0, g(q_1) \to q_0, a \to q_1\}$. In Δ transitions are* normalized. *A transition of the form $f(g(q_1)) \to q_0$ is not normalized. The term $g(a)$ is a term of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ (and of $\mathcal{T}(\mathcal{F})$) and can be rewritten by Δ in the following way: $g(a) \to_\Delta g(q_1) \to_\Delta q_0$. Note that $\mathcal{L}(\mathcal{A}, q_1) = \{a\}$ and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_0) = \{g(a), f(g(a)), f(f(g(a))), \ldots\} = \{f^\star(g(a))\}$.*

3 Tree Automata Completion

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, the tree automata completion algorithm, proposed in [Gen98, FGVTT04], computes a tree automaton $\mathcal{A}^*_\mathcal{R}$ such that $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ when it is possible (for some of the classes of TRSs where an exact computation is possible, see [FGVTT04]) and such that $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ otherwise.

The tree automata completion works as follows. From $\mathcal{A} = \mathcal{A}^0_\mathcal{R}$ completion builds a sequence $\mathcal{A}^0_\mathcal{R}.\mathcal{A}^1_\mathcal{R} \ldots \mathcal{A}^k_\mathcal{R}$ of automata such that if $s \in \mathcal{L}(\mathcal{A}^i_\mathcal{R})$ and $s \to_\mathcal{R} t$ then $t \in \mathcal{L}(\mathcal{A}^{i+1}_\mathcal{R})$. If we find a fixpoint automaton $\mathcal{A}^k_\mathcal{R}$ such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}^k_\mathcal{R})) = \mathcal{L}(\mathcal{A}^k_\mathcal{R})$, then we note $\mathcal{A}^*_\mathcal{R} = \mathcal{A}^k_\mathcal{R}$ and we have $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}^0_\mathcal{R}))$, or $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ if $\mathcal{R}$ is not in one class of [FGVTT04]. To build $\mathcal{A}^{i+1}_\mathcal{R}$ from $\mathcal{A}^i_\mathcal{R}$, we achieve a *completion step* which consists of finding *critical pairs* between $\to_\mathcal{R}$ and $\to_{\mathcal{A}^i_\mathcal{R}}$. To define the notion of critical pair, we extend the definition of substitutions to terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to^*_{\mathcal{A}^i_\mathcal{R}} q$ and $l\sigma \to_\mathcal{R} r\sigma$. Note that since $\mathcal{R}$, $\mathcal{A}^i_\mathcal{R}$ and the set $\mathcal{Q}$ of states of $\mathcal{A}^i_\mathcal{R}$ are finite, there is only a finite number of critical pairs. For every critical pair detected between $\mathcal{R}$ and $\mathcal{A}^i_\mathcal{R}$ such that $r\sigma \not\to^*_{\mathcal{A}^i_\mathcal{R}} q$, the tree automaton $\mathcal{A}^{i+1}_\mathcal{R}$ is constructed by adding a new transition $r\sigma \to q$ to $\mathcal{A}^i_\mathcal{R}$ such that $\mathcal{A}^{i+1}_\mathcal{R}$ recognizes $r\sigma$ in q, i.e. $r\sigma \to^*_{\mathcal{A}^{i+1}_\mathcal{R}} q$.



However, the transition $r\sigma \to q$ is not necessarily a normalized transition of the form $f(q_1, \ldots, q_n) \to q$ and so it has to be normalized first. Thus, instead of adding $r\sigma \to q$ we add $Norm(r\sigma \to q)$ to transitions of $\mathcal{A}^i_\mathcal{R}$. Here is the $Norm$ function used to normalize transition. Note that, in this function, transitions are normalized using either new states of $\mathcal{Q}_{new}$ or states of $\mathcal{Q}$, states of the automaton being completed. As mentioned in Lemma 1, this has no effect on the safety of the normalization but only on its precision.

Definition 4 ($Norm$) *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $\mathcal{Q}_{new}$ a set of* new *states such that $\mathcal{Q} \cap \mathcal{Q}_{new} = \emptyset$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. The function $Norm$ is inductively defined by:*

- $Norm(t \to q) = \emptyset$ *is* $t = q$,
- $Norm(t \to q) = \{c \to q \mid c \to t \in \Delta\}$ *if* $t \in \mathcal{Q}$,
- $Norm(f(t_1, \ldots, t_n) \to q) = \bigcup_{i=1\ldots n} Norm(t_i \to q_i) \cup \{f(q_1, \ldots, q_n) \to q\}$ *where* $\forall i = 1 \ldots n : (t_i \in \mathcal{Q} \Rightarrow q_i = t_i) \wedge (t_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q} \Rightarrow q_i \in \mathcal{Q} \cup \mathcal{Q}_{new})$.

When using only new states to normalize all the new transitions occuring in all the completion steps, completion is as precise as possible. However, doing so, completion is likely not to terminate (because of general undecidability results [GT95]). Enforcing termination of completion can be easily done by bounding the set of new states to be used with $Norm$ during the whole completion. We then obtain a finite tree automaton over-approximating the set of reachable states. The fact that normalizing with any set of states (new or not) is *safe* is guaranteed by the following simple lemma. For the general safety theorem of completion see [FGVTT04].

Lemma 1 *For all tree automaton* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \setminus \mathcal{Q}$ *and* $q \in \mathcal{Q}$, *if* $\Pi = Norm(t \to q)$ *whatever the states chosen in* $Norm(t \to q)$ *we have* $t \to_\Pi^* q$.

Proof 1 *This can be done by a simple induction on transitions to normalize, see [FGVTT04].*

To let the user of completion guide the approximation, we use two different tools: a set N of *normalization rules* (see [FGVTT04]) and a set $\mathcal{E}$ of *approximation equations*. An approximation equation is of the form $u = v$ where $u, v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Let $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ be a substitution such that $u\sigma \to_{\mathcal{A}_\mathcal{R}^{i+1}} q$, $v\sigma \to_{\mathcal{A}_\mathcal{R}^{i+1}} q'$ and $q \neq q'$. We have

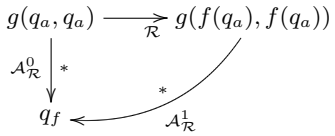
$$\begin{array}{ccc} u\sigma & \underset{\mathcal{E}}{=\!=} & v\sigma \\ {\scriptstyle \mathcal{A}_\mathcal{R}^{i+1}} \downarrow {\scriptstyle *} & & {\scriptstyle *} \downarrow {\scriptstyle \mathcal{A}_\mathcal{R}^{i+1}} \\ q & & q' \end{array}$$

and, thus, there exists some terms recognized by q and some recognized by q' which are equivalent modulo $\mathcal{E}$. A correct over-approximation of $\mathcal{A}_\mathcal{R}^{i+1}$ consits in applying the $Merge$ function to it, i.e. replace $\mathcal{A}_\mathcal{R}^{i+1}$ by $Merge(\mathcal{A}_\mathcal{R}^{i+1}, q, q')$, as long as an approximation equation of $\mathcal{E}$ applies. The $Merge$ function is defined below and an example of its application is given in Example 2.

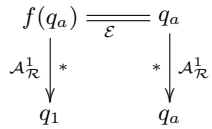
Definition 5 ($Merge$) *Let* $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \Delta \rangle$ *be a tree automaton and* q_1, q_2 *be two states of* $\mathcal{A}$. *We denote by* $Merge(\mathcal{A}, q_1, q_2)$ *the tree automaton where every occurence of* q_2 *is replaced by* q_1 *in* $\mathcal{Q}$, $\mathcal{Q}_F$ *and in every left-hand side and right-hand side of every transition of* Δ.

The following examples illustrate completion and how to carry out an approximation, using equations, when the language $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ is not regular.

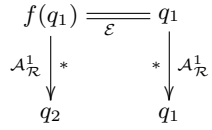
Example 2 *Let $\mathcal{R} = \{g(x,y) \to g(f(x), f(y))\}$ and let $\mathcal{A}$ be the tree automaton such that $\mathcal{Q}_F = \{q_f\}$ and $\Delta = \{a \to q_a, g(q_a, q_a) \to q_f\}$. Hence $\mathcal{L}(\mathcal{A}) = \{g(a,a)\}$ and $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) = \{g(f^n(a), f^n(a)) \mid n \geq 0\}$. Let $\mathcal{E} = \{f(x) = x\}$ be the set of approximation equations. During the first completion step on $\mathcal{A}^0_\mathcal{R} = \mathcal{A}$, we find $\sigma = \{x \mapsto q_a\}$ and the following critical pair*



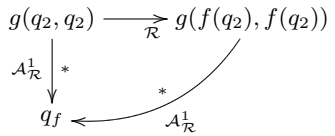
Hence, we have to add the transition $g(f(q_a), f(q_a)) \to q_f$ to $\mathcal{A}^0_\mathcal{R}$ to obtain $\mathcal{A}^1_\mathcal{R}$. This transitions can be normalized in the following way: $Norm(g(f(q_a), f(q_a)) \to q_f) = \{g(q_1, q_2) \to q_f, f(q_a) \to q_1, f(q_a) \to q_2\}$ where q_1 and q_2 are new states. Those new states and transitions are added to $\mathcal{A}^0_\mathcal{R}$ to obtain $\mathcal{A}^1_\mathcal{R}$. On this tree automaton, we can apply the equation $f(x) = x$ of $\mathcal{E}$ with the substitution $\sigma = \{x \mapsto q_a\}$:



Hence, we can replace $\mathcal{A}_\mathcal{R}$ by $Merge(\mathcal{A}^1_\mathcal{R}, q_1, q_a)$ where Δ is $\{a \to q_1, g(q_1, q_1) \to q_f, g(q_1, q_2) \to q_f, f(q_1) \to q_1, f(q_1) \to q_2\}$. Similarly, in this last tree automaton, we have



and we can thus apply $Merge(\mathcal{A}^1_\mathcal{R}, q_2, q_1)$. Finally, the value of Δ for $\mathcal{A}^1_\mathcal{R}$ approximated by $\mathcal{E}$ is $\{a \to q_2, g(q_2, q_2) \to q_f, f(q_2) \to q_2\}$. Now, the only critical pair that can be found on $\mathcal{A}^1_\mathcal{R}$ is joinable:



Hence, we have $\mathcal{A}^*_\mathcal{R} = \mathcal{A}^1_\mathcal{R}$ and $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) = \{g(f^n(a), f^m(a)) \mid n, m \geq 0\}$ which is an over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

The tree automata completion algorithm and the approximation mechanism are implemented in the Timbuk [GVTT00] tool. On the previous example, once the fixpoint automaton $\mathcal{A}^*_\mathcal{R}$ has been computed, it is possible to check whether

some terms are reachable, i.e. recognized by $\mathcal{A}^*_{\mathcal{R}}$ or not. This can be done using tree automata intersections [FGVTT04].

4 A result checker for tree automata completion

By moving the certification problem from the completion algorithm to the checker, the certification problem consists in proving the following Coq theorem:

```
Theorem sound_checker :
      ∀ A A' R, checker A R A' = true → ApproxReachable A R A'.
```

where `ApproxReachable` is a Coq predicate that describes the Soundness Property: $\mathcal{L}(A')$ *contains all terms reachable by rewriting terms of* $\mathcal{L}(A)$ *with* $\mathcal{R}$, i.e. $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$. To state formally this predicate in Coq, we need to give a Coq axiomatization of Term Rewriting Systems and of Tree Automata. It is given in Section 5. Given two automata $\mathcal{A}$, $\mathcal{A}'$ and a TRS $\mathcal{R}$ the checker verifies that $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ or (`ApproxReachable A R A'`) in Coq. To perform this, we need to check the two following properties:

- `Included`: inclusion of initial set in the fixpoint: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$.
- `IsClosed`: $\mathcal{A}'$ is closed by rewriting with $\mathcal{R}$: For all $l \to r \in \mathcal{R}$ and all $t \in \mathcal{L}(\mathcal{A}')$, if t is rewritten in t' by the rule $l \to r$ then $t' \in \mathcal{L}(\mathcal{A}')$.

For each item, we provide a Coq function and its correctness theorem: function `inclusion` is dedicated to inclusion checking and function `closure` checks if a tree automaton is closed by rewriting. We also give the theorem used to deduce `ApproxReachable A R A'` from `Included A A'` and `IsClosed R A'`:

```
Theorem inclusion_sound:
      ∀ A A', inclusion A A' = true → Included A A'.

Theorem closure_sound:
      ∀ R A', closure R A' = true → IsClosed R A'.

Theorem Included_IsClosed_ApproxReachable:
      ∀ A A' R, Included A A' → IsClosed R A' → ApproxReachable A R A'.
```

Note that, in this paper we focus on the proof of $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$. However, to prove the unreachability property, the emptiness of the intersection between $\mathcal{L}(\mathcal{A}')$ and the bad term set has also to be verified. Since the formalization in Coq of the intersection and emptiness decision are close to their standard definition [CDG+02], and since they have already been covered by [RGL01], they are not be detailed in this paper.

5 Formalization of Term Rewriting Systems

The aim of this part is to formalize in Coq: terms, term rewriting systems, reachable terms and the reachability problem itself. Firstly we use the positive integers provided by the Coq's standard library to define symbol sets like variables ($\mathcal{X}$) or function symbols ($\mathcal{F}$). We rename `positive` into `ident` to be more explicit. Then, we define term set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ using inductive types:

```
Definition ident := positive.

Inductive term : Set :=
| Fun : ident → list term → term
| Var : ident → term.
```

Now, the term $f(x, a)$ will be written `Fun 0 (Var 0::(Fun 1 nil)::nil)` assuming that we have the corresponding mapping between between symbols, variables and positive integers $f \mapsto 0$, $a \mapsto 1$ and $x \mapsto 0$ for example. Note that it is possible to attach the value 0 to f and x, since the `term`'s constructors `Fun` and `Var` allows to differentiate between variable and function symbols.

Remarks:

- Since equality is decidable, we can easily define term equality as a decidable relation. Afterward, this is very useful to define functions where term comparison is required.
- A bad point for Coq is the induction principle `term_rect` automatically generated which is too weak. To prove properties in the following, we need a more efficient theorem named `term_rect'`. This is required to be able to prove most of the theorems on terms.

A rewrite rule $l \to r$ is represented by a pair of terms with a well-definition proof, i.e. a Coq proof that the set of variables of r is a subset of the set of variables of l. The function `Fv : term →list ident` builds the set of variables for a term.

```
Inductive rule : Set :=
| Rule (l r : term)(H : subseteq (Fv r) (Fv l)) : rule.
```

In the following, `list rule` type represents a TRS. In Coq we use `(t @ sigma)` to denote the term resulting of the application of a substitution `sigma` to each variable that occurs in a term `t`.

```
Definition substitution := ident → option term.
```

In Coq, the rewriting relation *"u is rewritten in v by $l \to r$"*, commonly defined by $\exists \sigma \ s.t. \ u|_p = l\sigma \ \wedge \ v = u[r\sigma]_p$, is split into two predicates:

- The first one defines the rewriting of a term at the topmost position. In fact, the set of term pairs (t, t') which are rewritten at the top most by the rule can be seen as the set of term pairs $(l\sigma, r\sigma)$ for any substitution σ.

- The second one just defines inductively the rewriting relation at any position of a term t by a rule $l \to r$, by the topmost rewriting of any subterm of t by $l \to r$.

```
(* Topmost rewriting : *)
Inductive TRew (x : rule) : term → term → Prop :=
| R_Rew :
  ∀ s l r (H : subseteq (Fv r) (Fv l)),
    x = Rule l r H → TRew x (l @ s) (r @ s).

(* Rewrite at any position of term *)
Inductive Rew (r : rule) : term → term → Prop :=
| Rew1 : ∀ t t',
    TRew r t t' → Rew r t t'
| Rew2 : ∀ f l l',
    Rargs r l l' → Rew r (Fun f l) (Fun f l')

with Rargs (r : rule):list term→list term→Prop:=
| Ra1 : ∀ t t' l,
    Rew r t t' → Rargs r (t::l) (t'::l)
| Ra2 : ∀ t l l',
    Rargs r l l' → Rargs r (t::l) (t::l').
```

Then we have to define $\to_{\mathcal{R}}^*$. In Coq, we prefer to see it as the predicate `Reachable R u` that characterizes the set of reachable terms from u by $\to_{\mathcal{R}}^*$.

```
Inductive Reachable(R : list rule)(t : term) : term → Prop:=
| R_refl : Reachable R t t
| R_trans : ∀ u v r, Reachable R t u → In r R → Rew r u v →
    Reachable R t v.
```

6 Formalization of Tree Automata

The fact that the checker, to be executed, is directly extracted from the Coq formalization has an important consequence on the tree automata formalization. Since the data structures used in the formalization are those that are really used for the execution, they need to be formal *and* efficient. For tree automata, instead of a naive representation, it is thus necessary to use optimized formal data structures borrowed from [RGL01].

In Section 5, we have represented variables $\mathcal{X}$ and function symbols $\mathcal{F}$ by the type `ident`. We do the same for $\mathcal{Q}$. We define a tree automaton as a pair $(\mathcal{Q}_F, \Delta)$, where $\mathcal{Q}_F$ is the finite set of final states, and Δ the finite set of normalized transitions like $f(q_1, \ldots, q_n) \to q$. In Coq, $\mathcal{Q}_F$ is a simple `list ident` but Δ is represented using the `FMapPositive` Coq library of functional mappings, where data are indexed by `positive` numbers. In the `FMapPositive` structure, every transition $f(q_1, \ldots, q_n) \to q$ is encoded by a list of states $(q_1, \ldots, q_n)$ indexed by f in a map which is also indexed by q in a second map. This representation is a good solution to deal efficiently, in Coq, with transition sets.

```
Module Delta : DELTA.
   (* Transition sets : *)
   Definition config := list state.

   Definition t :=
        FMap.t (FMap.t (list config)).
   (* ............... *)
```

Now we can define a predicate to characterize the recognized language of a tree automaton. In fact, we are defining the set of ground terms that are reduced to a state q by a transition set Δ. This set, which corresponds to $\mathcal{L}(\mathcal{A}, q)$ if Δ is the set of transitions of $\mathcal{A}$, can be constructed inductively in Coq using the single deduction rule:

$$\frac{t_1 \in \mathcal{L}(\Delta, q_1) \qquad \ldots\ldots \qquad t_n \in \mathcal{L}(\Delta, q_n)}{f(t_1, \ldots, t_n) \in \mathcal{L}(\Delta, q)} \text{ If } f(q_1, \ldots, q_n) \to q \in \Delta$$

In Coq, we express this statement using the inductive predicate `IsRec`. A term t is recognized by a tree automaton $(\mathcal{Q}_F, \Delta)$, if the predicate `IsRec` Δ q t is valid for $q \in \mathcal{Q}_F$.

```
Inductive IsRec (D: Delta.t) : state → term → Prop :=
  Rec_Term : ∀ f lt q,
      IsRec' D (Delta.get q f D) lt → IsRec D q (Fun f lt)

with IsRec' (D: Delta.t) : list config → list term → Prop :=
| Rec_SubTerm : ∀ lt c lc, IsRec'' D c lt → IsRec' D (c::lc) lt
| Rec_SubTerm' : ∀ lt c lc, IsRec' D lc lt → IsRec' D (c::lc) lt

with IsRec'' (D: Delta.t) : config → list term → Prop :=
| Rec_Nil : IsRec'' D nil nil
| Rec_Cons : ∀ t q lt lq, IsRec D q t → IsRec'' D lq lt →
       IsRec'' D (q::lq) (t::lt).
```

7 An optimized inclusion checker

In this part, we give the formal definition of the `Included` property and of the `inclusion` Coq function used to effectively check the tree automata inclusion. From the previous formal definitions on tree automata, we can state the `Included` predicate in the following way:

```
  Definition Included (a b : t_aut) : Prop :=
    ∀ t q, In q a.qf → IsRec a.delta q t →
      ∃ q', In q' b.qf ∧ IsRec b.delta q' t.
```

Now let us focus on the function `inclusion` itself. The usual algorithm for proving inclusion of regular languages recognized by nondeterministic bottom-up tree automata, for instance for proving $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, consists in proving that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\overline{\mathcal{B}}) = \emptyset$, where $\overline{\mathcal{B}}$ is the complement automaton for $\mathcal{B}$. However,

the algorithm for building $\overline{\mathcal{B}}$ from $\mathcal{B}$ is EXPTIME-complete [CDG+02]. This is the reason why we here define a criterion with a better practical complexity. It is is based on a simple syntactic comparison of transition sets, i.e. we check the inclusion of transition sets modulo the renamings performed by the $Merge$ function. This increases a lot the efficiency of our checker, especially by saving memory. This is crucial to check inclusion of big tree automata (see Section 9). This algorithm is correct but, of course, it is not complete in general, i.e. not always able to prove that $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$. However, we show in the following that, under certain conditions on $\mathcal{A}$ and $\mathcal{B}$ which are satisfied if $\mathcal{B}$ is obtained by completion of $\mathcal{A}$, this algorithm is also complete and thus becomes a decision procedure. First, we introduce the following notation:

Γ	:	induction hypothesis set
Δ_i	:	transition set of the tree automaton $\mathcal{A}_i$
$\{c \mid c \to q \in \Delta\}$	:	configurations of Δ that are rewritten in q
$\{c_i\}_n^m$	:	configuration set from c_n to c_m

We formulate our inclusion problem by formulas of the form: $\Gamma \vdash_{A,B} q \Subset q'$. Such a statement stands for: under the assumption Γ, it is possible to prove that $\mathcal{L}(A, q) \subseteq \mathcal{L}(B, q')$. The algorithm consists in building proof trees for those statements using the following set of deduction rules.

$$\text{(Induction)}\ \frac{\Gamma \cup \{q \Subset q'\} \vdash_{A,B} \{c \mid c \to_{\Delta_A} q\} \Subset \{c \mid c \to_{\Delta_B} q'\}}{\Gamma \vdash_{A,B} q \Subset q'}\ \text{if } (q \Subset q') \notin \Gamma$$

$$\text{(Axiom)}\ \frac{}{\Gamma \cup \{q \Subset q'\} \vdash_{A,B} q \Subset q'} \qquad \text{(Empty)}\ \frac{}{\Gamma \vdash_{A,B} \emptyset \Subset \{c'_j\}_1^m}$$

$$\text{(Split-l)}\ \frac{\Gamma \vdash_{A,B} c_1 \Subset \{c'_j\}_1^m \quad \ldots\ldots \quad \Gamma \vdash_{A,B} c_n \Subset \{c'_j\}_1^m}{\Gamma \vdash_{A,B} \{c_i\}_1^n \Subset \{c'_j\}_1^m}$$

$$\text{(Weak-r)}\ \frac{\Gamma \vdash_{A,B} c \Subset c'_k}{\Gamma \vdash_{A,B} c \Subset \{c'_i\}_1^n}\ \text{if } (1 \leq k \leq n) \qquad \text{(Const.)}\ \frac{}{\Gamma \vdash_{A,B} a() \Subset a()}$$

$$\text{(Config)}\ \frac{\Gamma \vdash_{A,B} q_1 \Subset q'_1 \quad \ldots\ldots \quad \Gamma \vdash_{A,B} q_n \Subset q'_n}{\Gamma \vdash_{A,B} f(q_1, \ldots, q_n) \Subset f(q'_1, \ldots, q'_n)}$$

Given $\mathcal{Q}_{F_\mathcal{A}}$ and $\mathcal{Q}_{F_\mathcal{B}}$ the sets of final states of $\mathcal{A}$ and $\mathcal{B}$, $\#()$ a symbol of arity 1 not occurring in $\mathcal{F}$, to prove $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, we start our deduction by the statement: $\emptyset \vdash_{\mathcal{A},\mathcal{B}} \{\#(q) \mid q \in \mathcal{Q}_{F_\mathcal{A}}\} \Subset \{\#(q) \mid q \in \mathcal{Q}_{F_\mathcal{B}}\}$

Example 3 *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata s.t.:*

$$\mathcal{A} = \left\{ \begin{array}{rcl} a & \to & q_1 \\ b & \to & q_2 \\ f(q_1, q_2) & \to & \mathbf{q} \end{array} \right\} \textit{ with } \mathcal{Q}_{F_\mathcal{A}} = \{\mathbf{q}\} \textit{ and } \mathcal{B} = \left\{ \begin{array}{rcl} a & \to & \mathbf{q}' \\ b & \to & \mathbf{q}' \\ f(q', q') & \to & \mathbf{q}' \end{array} \right\} \textit{ with } \mathcal{Q}_{F_\mathcal{B}} = \{\mathbf{q}'\}$$

Here we have $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ and we can derive $\emptyset \vdash_{\mathcal{A},\mathcal{B}} \#(q) \Subset \#(q')$ with the deduction rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{q \Subset q', q_1 \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} a() \Subset a()}(Const.)}{\{q \Subset q',\ q_1 \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} a() \Subset \{a(), b(), f(q', q')\}}(Weark\text{-}r)}{\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_1 \Subset q'}(Induction) \quad \dfrac{\dfrac{\dfrac{}{\{q \Subset q', q_2 \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} b() \Subset b()}(Const.)}{\{q \Subset q',\ q_2 \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} b() \Subset \{a(), b(), f(q', q')\}}(Weark\text{-}r)}{\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_2 \Subset q'}(Induction)}{\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} f(q_1, q_2) \Subset f(q', q')}(Config)$$

$$\dfrac{\dfrac{\dfrac{\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} f(q_1, q_2) \Subset f(q', q')}{\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} f(q_1, q_2) \Subset \{a(), b(), f(q', q')\}}(Weark\text{-}r)}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'}(Induction)}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} \#(q) \Subset \#(q')}(Config)$$

The main property we want demonstrate in Coq is that this syntactic criterion implies the semantic inclusion for the considered languages in 6.

```
Theorem inclusion_sound :
  ∀ A B, inclusion A B = true → Included A B.
```

Before proving this in Coq, we need to define more formally the function `inclusion`. This function cannot be defined as a simple structural recursion. Thus Coq needs a termination proof for this algorithm. Thanks to the Coq feature `Function`, it is possible to define the algorithm using a measure function and provide a proof that its value decreases at each recursive call to ensure the termination.

7.1 Termination

Termination of deduction rules can be proved by defining a measure function μ on statements of he form $\Gamma \vdash_{\mathcal{A},\mathcal{B}} \alpha \Subset \beta$. The Γ relation can be seen as a subset of $\mathcal{Q}_A \times \mathcal{Q}_B$ which is a finite set. All tree automata have a finite number of states. Then the statement measure $\mu(\Gamma \vdash_{A,B} \alpha \trianglelefteq \beta)$ is defined as tuple $(\mu_1(\Gamma), \mu_2(\alpha) + \mu_2(\beta))$ where:

$$\left[\begin{array}{lcl} \mu_1(\Gamma) & = & |\mathcal{Q}_A \times \mathcal{Q}_B| - |\Gamma| \\ \mu_2(x) & = & \begin{cases} (m+1-n) \text{ if } x = \{c_i\}_n^m \\ 1 \text{ if } x = f(q_1, \ldots, q_n), \\ 0 \text{ otherwise} \end{cases} \end{array}\right.$$

Then we define the ordering $\ll$ by the lexicographic combination of the usual order $<$ on natural numbers for μ_1 and μ_2.

$$(x, y) \ll (x', y') \Longleftrightarrow \bigvee \begin{cases} x < x' \\ x = x' \wedge y < y' \end{cases}$$

Since $<$ is well founded, the lexicographic combination $\ll$ is also well founded.

Theorem 1 *(Termination) At each deduction step, the measure decreases strictly:*

$$\frac{\Gamma \vdash_{A,B} \alpha \Subset \beta}{\Gamma' \vdash_{A,B} \alpha' \Subset \beta'} \Longrightarrow \mu(\Gamma \vdash_{A,B} \alpha \Subset \beta) \ll \mu(\Gamma' \vdash_{A,B} \alpha' \Subset \beta')$$

Proof 2 *The following array summarizes for each derivation rule what component of the tuple proves that μ decreases between conclusion and premises of the rule:*

	μ_1	μ_2
Induction	$\mu_1(\Gamma) < \mu_1(\Gamma')$	$-$
Split-l	$\mu_1(\Gamma) = \mu_1(\Gamma')$	$\mu_2(c_i) = 1 < \mu_2(\{c_i\}_1^n)$
Weak-r	$\mu_1(\Gamma) = \mu_1(\Gamma')$	$\mu_2(c_k) < \mu_2(\{c_i\}_1^n)$
Config	$\mu_1(\Gamma) = \mu_1(\Gamma')$	$\mu_2(f(\dots, q_i, \dots)) = \mu_2(f(\dots, q_i', \dots)) = 1$ $\mu_2(q_i) = \mu_2(q_i') = 0$ *thus* $2 > 0$

For the `Split-l` *(resp.* `Weak-r`*) rule, we consider $n > 1$ to have a set $\alpha = \{c_i\}_1^n$ (resp. β) with at least two elements. If ($n = 1$) then this rule does not apply on the current statement $\Gamma \vdash \alpha \Subset \beta$.*

Theorem 2 *When $\mu(\Gamma \vdash_{A,B} \alpha \Subset \beta) = (0, 0)$, we have a statement as Axiom or Nil: the current proof derivation is completed.*

Proof 3 *From $\mu(\Gamma \vdash_{A,B} \alpha \Subset \beta) = (0, 0)$ we deduce immediately:*

1. *$\mu_1(\Gamma) = 0$ implies $\Gamma = \mathcal{Q}_A \times \mathcal{Q}_B$*
2. *$\mu_2(\alpha) = \mu_2(\beta) = 0$ implies α and β are both either a state or empty set.*

Thus the statement may be:

- *$\Gamma \vdash_{A,B} \emptyset \Subset \emptyset$ is the case of Empty rule : proof derivation is ended.*
- *$\Gamma \vdash_{A,B} q \Subset q'$: we can use the fact $\Gamma = \mathcal{Q}_A \times \mathcal{Q}_B$, thus $(q, q') \in \mathcal{Q}_A \times \mathcal{Q}_B \implies (q \Subset q') \in \Gamma$. This case matches with Axiom rule that terminates the proof derivation.*

7.2 Soundness

Before proving it, we have to define and prove the following theorem.

Theorem 3 *(Cut in $\Subset$-proof trees) For all tree automata $\mathcal{A}$ and $\mathcal{B}$, if there exists $\prod$ a proof tree of $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$, and a proof tree of $\Gamma \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ then exists also a proof tree of $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$.*

Proof 4 *We proceed by induction on $\mu(\Gamma)$.*
If $\mu(\Gamma) = 0$, we have immediately $\mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B} = \Gamma$. Hence, since $q_a \Subset q_b \in \Gamma$, we can prove $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ using the Axiom rule.

Now, as induction hypothesis, let us assume that $\forall \Gamma$ s.t. $\mu(\Gamma) = n$, $\forall q\ q'$, if there exists a proof tree $\prod$ of $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$ and if for all q_a, q_b there exists a proof tree of $\Gamma \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ then we have also a proof tree of $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$. Now, we aim at proving that this property is true for Γ such that $\mu(\Gamma) = n + 1$.

Let us consider the proof tree of the second hypothesis $\Gamma \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$. *Firstly, if the proof tree is built using the Axiom rule we have* $(q_a \Subset q_b) \in \Gamma \cup \{(q \Subset q')\}$. *Two cases are possible:*

- *either* $(q_a \Subset q_b) \in \Gamma$, *and then we build the proof of* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ *using the Axiom rule.*
- *or* $q = q_a$ *and* $q' = q_b$, *and then the goal* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ *is equivalent to* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$ *whose proof tree is* $\prod$.

Secondly, if the proof tree of $\Gamma \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$ *is built using the Induction rule, then we have:*

$$\dfrac{\dfrac{\dfrac{\prod_{c_1}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} c_1 \Subset c'_{k_1}}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} c_1 \Subset \{c'_k | c'_k \to_{\mathcal{B}} q_b\}_1^m}\,(\textit{Weark-r}) \quad \cdots\cdots \quad \dfrac{\dfrac{\prod_{c_n}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} c_n \Subset c'_{k_n}}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} c_n \Subset \{c'_k | c'_k \to_{\mathcal{B}} q_b\}_1^m}\,(\textit{Weark-r})}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} \{c_i | c_i \to_{\mathcal{A}} q_a\}_1^n \Subset \{c'_i | c'_i \to_{\mathcal{B}} q_b\}_1^m}\,(\textit{Split-l})}{\Gamma \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b}\,(\textit{Induction})$$

Where each $\prod_{c_i}$ *has the following form (assuming that* $c_i = f(q_{i_1}, \ldots, q_{i_n})$ *and* $c'_{k_i} = f(q'_{i_1}, \ldots, q'_{i_n})$ *):*

$$\dfrac{\dfrac{\prod_{i_1}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_1} \Subset q'_{i_1}} \quad \cdots\cdots \quad \dfrac{\prod_{i_n}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_n} \Subset q'_{i_n}}}{\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} f(q_{i_1}, \ldots, q_{i_n}) \Subset f(q'_{i_1}, \ldots, q'_{i_n})}\,(\textit{Config})$$

If we try to build the proof tree of our goal $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$, *it necessarily begins in the same way except that* $\{q \Subset q'\}$ *will not appear in the left-hand side of statements. Each branch of this tree will end by a statement of the form* $\Gamma \cup \{q_a \Subset q_b\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_j} \Subset q'_{i_j}$. *Now to conclude the proof, we have to find proof trees* $\prod'_{i_j}$ *for all those statements. We know that there exists proof trees* $\prod_{i_j}$ *for all statements* $\Gamma \cup \{q \Subset q'\} \cup \{q_a \Subset q_b\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_j} \Subset q'_{i_j}$. *We can use the induction hypothesis on* $\prod_{i_j}$ *to obtain* $\prod'_{i_j}$ *as follows:*

- *Since* $\mu(\Gamma) = n + 1$, *then* $\mu(\Gamma \cup \{q_a \Subset q_b\}) = n$
- *Since* $\prod$ *is a proof of* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$, *it is also a proof of* $\Gamma \cup \{q_a \Subset q_b\} \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$.
- *Each* $\prod_{i_j}$ *is a proof of* $\Gamma \cup \{q_a \Subset q_b\} \cup \{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_j} \Subset q'_{i_j}$

Using induction, we deduce that for all i, j *there exist proof trees* $\prod'_{i_j}$ *of* $\Gamma \cup \{q_a \Subset q_b\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_j} \Subset q'_{i_j}$. *This ends the proof tree of our goal* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q_a \Subset q_b$.

Theorem 4 *(Soundness) For all tree automata* $\mathcal{A}$ *and* $\mathcal{B}$, *if there exists* $\prod$ *a proof tree of* $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$ *then we have* $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{L}(\mathcal{B}, q')$

Proof 5 *We prove that* $\forall t,\ t \in \mathcal{L}(\mathcal{A}, q) \implies t \in \mathcal{L}(\mathcal{B}, q')$ *by induction on* t. *Let* $t = f(t_1, \ldots, t_n)$. *We assume that the property is true for each subterm* t_i, *i.e. for all* q_i, q_i' *s.t. if there exists a proof tree* $\prod_i$ *of* $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$ *then* $t_i \to_{\mathcal{A}}^* q \implies t \to_{\mathcal{B}}^* q_i'$. *Since* $t = f(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{A}, q)$, *then for each subterm* t_i, *we know that there exists* $q_1, \ldots, q_n$ *such that* $t_i \in \mathcal{L}(\mathcal{A}, q_i)$ *and* $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$. *Besides this, by unfolding* $\prod$ *the proof tree of* $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$, *we can deduce that for each transition like* $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$, *there exists* $f(q_1', \ldots, q_n') \to q' \in \mathcal{B}$ *s.t. we have a proof tree* $\prod_i$ *of* $\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$. *Since* $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$, *we obtain that* $f(q_1', \ldots, q_n') \to q' \in \mathcal{B}$ *and a proof* $\prod_i$ *for* $\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$. *To conclude that* $f(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{B}, q')$ *we just have to prove that* $t_i \in \mathcal{L}(\mathcal{B}, q_i')$. *Note that we have a proof tree* $\prod_i$ *for* $\{q \Subset q'\} \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$ *and that to apply the induction hypothesis we need a proof tree for* $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$. *Using the Theorem 3 on* $\prod$ *and* $\prod_i$, *we can deduce the existence of* $\prod_i'$ *the proof tree of* $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset q_i'$. *Then using induction hypothesis on* t_i', q_i, q_i' *and* $\prod_i'$, *we obtain that for each* $t_i \in \mathcal{L}(\mathcal{A}, q_i)$, *we also have* $t_i \in \mathcal{L}(\mathcal{B}, q_i')$. *Finally, since* $f(q_1', \ldots, q_n') \to q' \in \mathcal{B}$, *we obtain that* $t = f(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{B}, q')$.

7.3 Completeness

As said above, the described algorithm is not complete in general. However, we show that it is complete for tree automata produced by completion. In particular if $\mathcal{A}_{\mathcal{R}}^k$ is obtained after k completion step from $\mathcal{A}^0$ then we can build a proof $\prod$ for the statement $\emptyset \vdash_{\mathcal{A}^0,\mathcal{A}_{\mathcal{R}}^k} \{\#(q) \mid q \in \mathcal{Q}_{F_0}\} \Subset \{\#(q') \mid q' \in \mathcal{Q}_{F_k}\}$. Recall that the tree automaton produced by the k^{th} step of completion is noted $\mathcal{A}_k = \langle \mathcal{F}, \mathcal{Q}_k, \mathcal{Q}_{F_k}, \Delta_k \rangle$. First, let us introduce some new ordered relation on tree automata:

Definition 6 *Given* $\mathcal{A}$, $\mathcal{B}$ *two tree automata,* $\sqsubseteq$ *is the reflexive and transitive relation defined as follows:* $\mathcal{A} \sqsubseteq \mathcal{B}$ *if there exists a function* ϱ *that renames states of* $\mathcal{A}$ *into states of* $\mathcal{B}$ *and such that all renamed rules* $\Delta_{\mathcal{A}}$ *are contained in* $\Delta_{\mathcal{B}}$*:*

$$\mathcal{A} \sqsubseteq \mathcal{B} \iff \exists \varrho : \mathcal{Q}_{\mathcal{A}} \to \mathcal{Q}_{\mathcal{B}},\ \varrho(\Delta_{\mathcal{A}}) \subseteq \Delta_{\mathcal{B}} \ \wedge\ \varrho(\mathcal{Q}_{F_{\mathcal{A}}}) \subseteq \mathcal{Q}_{F_{\mathcal{B}}} \tag{1}$$

We need to extend the renaming ϱ to the structures and sets used in the following:

- $\varrho(\{q_i\}_1^n)$ stands for $\{\varrho(q_i)\}_1^n$
- $\varrho(c) = \begin{cases} f(\varrho(c_1), \ldots, \varrho(c_n)) & \text{if } c = f(c_1, \ldots, c_n) \\ c & \text{if } c \in \mathcal{F}_0 \\ \varrho(q) & \text{if } c = q \in \mathcal{Q} \end{cases}$
- $\varrho(c \to q)$ stands for $\varrho(c) \to \varrho(q)$
- $\varrho(\Delta)$ stands for $\{\varrho(c \to q) \mid c \to q \in \Delta\}$.

In tree automata completion, two main operations can be performed at each steps of calculus: *normalization* and *state merging*. The following lemma shows that those two operations ensure $\sqsubseteq$.

Lemma 2 *Given a tree automaton $\mathcal{A}$,*

$$\begin{array}{lllll} 1. & \text{if } \mathcal{A}' = \mathcal{A} \cup Norm(r\sigma \to q) & \text{then} & \mathcal{A} \sqsubseteq \mathcal{A}' \\ 2. & \text{if } \mathcal{A}' = Merge(\mathcal{A}, q_1, q_2) & \text{then} & \mathcal{A} \sqsubseteq \mathcal{A}' \end{array}$$

Proof 6

1. *This is easy to show since we trivially have $\Delta_{A'} \supseteq \Delta_A$ whatever $r\sigma$ or q may be. Then by choosing $\varrho = id$, we have immediately the conclusion $\mathcal{A} \sqsubseteq \mathcal{A}'$.*

2. *Let $\Delta_\mathcal{A}$ be the transition set of $\mathcal{A}$. Let q_i and q_j be the two states to merge. We can apply to $\Delta_\mathcal{A}$ a renaming function ϱ which has the same behavior than state merging with regard to $q_1 = q_2$:*

$$\varrho(q) = \begin{cases} \text{if } (q = q_2) \\ \quad q_1 \\ \text{else} \\ \quad q \end{cases}$$

So state merging builds $\Delta_{\mathcal{A}'} = \varrho(\Delta_\mathcal{A})$ and by Definition 6 we have trivially $\Delta_A \sqsubseteq \Delta_{\mathcal{A}'}$.

Theorem 5 *Given a tree automaton $\mathcal{A}^0$, a TRS $\mathcal{R}$ and an equation set $\mathcal{E}$, after k completion steps we obtain $\mathcal{A}^k_\mathcal{R}$ such that $\mathcal{A}^0 \sqsubseteq \mathcal{A}^k_\mathcal{R}$.*

Proof 7 *By induction on k:*

- *Since $\sqsubseteq$ is reflexive, we have trivially $\mathcal{A}^0 \sqsubseteq \mathcal{A}^0$.*
- *Let $\mathcal{A}_k$ be a tree automaton obtained after k completion steps such that $\mathcal{A}^0 \sqsubseteq \mathcal{A}^k_\mathcal{R}$. By definition of completion $\mathcal{A}^{k+1}_\mathcal{R}$ is built from $\mathcal{A}^k_\mathcal{R}$ by applying successively normalization and merge. We thus have $\mathcal{A}^k_\mathcal{R} \sqsubseteq \mathcal{A}^{k+1}_\mathcal{R}$. By transitivity of $\sqsubseteq$, from $\mathcal{A}^0 \sqsubseteq \mathcal{A}^k_\mathcal{R}$ and $\mathcal{A}^k_\mathcal{R} \sqsubseteq \mathcal{A}^{k+1}_\mathcal{R}$ we deduce immediately that $\mathcal{A}^0 \sqsubseteq \mathcal{A}^{k+1}_\mathcal{R}$.*

Theorem 6 *(Completeness) Given two tree automata $\mathcal{A}$ and $\mathcal{B}$ if $\mathcal{A} \sqsubseteq \mathcal{B}$ then there exists $\prod$ a proof of statement $\emptyset \vdash_{\mathcal{A},\mathcal{B}} \{\#(q_f) \mid q_f \in \mathcal{Q}_{F_A}\} \Subset \{\#(q'_f) \mid q'_f \in \mathcal{Q}_{F_B}\}$.*

Proof 8 *By definition of $\mathcal{A} \sqsubseteq \mathcal{B}$, we can deduce that there exists a renaming ϱ. First we prove by induction on the proof tree we have for all Γ and q, $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset \varrho(q)$:*
The hypothesis induction is $\forall \Gamma,\ q,\ q_i$,

$$\frac{\prod_i}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} q_i \Subset \varrho(q_i)}$$

We want to construct a proof tree for $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset \varrho(q)$.
Two cases are possible:

- *if* $q \Subset \varrho(q) \in \Gamma$ *then we can conclude immediately:*

$$\textit{(Axiom)}\ \frac{}{\Gamma' \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} q \Subset \varrho(q)}$$

- *Otherwise, we need to apply Induction rule to obtain the following tree:*

$$\textit{(Split-l)}\ \frac{\dfrac{\prod_{c_1}}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} c_1 \Subset \{c'_k | c'_k \to \varrho(q)\}_1^m} \quad \ldots\ldots \quad \dfrac{\prod_{c_n}}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} c_n \Subset \{c'_k | c'_k \to \varrho(q)\}_1^m}}{\textit{(Induction)}\ \dfrac{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} \{c_i | c_i \to q\}_1^n \Subset \{c'_k | c'_k \to \varrho(q)\}_1^m}{\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset \varrho(q)}}$$

From hypothesis $\varrho(\Delta_A) \subseteq \Delta_B$ *for each rule* $c \to q$ *of* Δ_A, *we have* $\varrho(c \to q) \in \Delta_{\mathcal{B}}$.

Thus for all $(c \to q) \in \Delta_{\mathcal{A}}$, *we have* $\varrho(c) \in \{c'_k | c'_k \to \varrho(q)\}_1^m$

For each $c_i = f_i(q_{i_1}, \ldots, q_{i_n})$ *we can construct the corresponding tree* $\prod_{c_i}$ *which his each branch is concluded by* $\prod_{i_j}$ *an instance of induction hypothesis for the corresponding state* q_{i_j}*:*

$$\textit{(Config)}\ \frac{\dfrac{\prod_{i_1}}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_1} \Subset \varrho(q_{i_1})} \quad \ldots\ldots \quad \dfrac{\prod_{i_n}}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} q_{i_n} \Subset \varrho(q_{i_n})}}{\textit{(Weak-r)}\ \dfrac{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} c_i \Subset \varrho(c_i)}{\Gamma \cup \{q \Subset \varrho(q)\} \vdash_{\mathcal{A},\mathcal{B}} c_i \Subset \{c'_k | c'_k \to \varrho(q)\}_1^m}}$$

Now, we have for all Γ *and* $q \in \mathcal{Q}_{\mathcal{A}}$ *there exists a proof tree* $\prod_q$ *for all statement* $\Gamma \vdash_{\mathcal{A},\mathcal{B}} q \Subset \varrho(q)$.
In particular, this is true for $\Gamma = \emptyset$ *all* q *of* $\mathcal{Q}_{F_{\mathcal{A}}}$. *Since we have* $\mathcal{A} \sqsubseteq \mathcal{B} \implies \varrho(\mathcal{Q}_{F_{\mathcal{A}}}) \subseteq \mathcal{Q}_{F_{\mathcal{B}}}$, *we can build a proof tree as:*

$$\textit{(Split-l)}\ \frac{\textit{(Weak-r)}\ \dfrac{\textit{(Config)}\ \dfrac{\prod_{q_{f_1}}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_{f_1} \Subset \varrho(q_{f_1})}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} \#(q_{f_1}) \Subset \#(\varrho(q_{f_1}))}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_{f_1} \Subset \mathcal{Q}_{F_{\mathcal{B}}}} \quad \ldots\ldots \quad \frac{\textit{(Weak-r)}\ \dfrac{\textit{(Config)}\ \dfrac{\prod_{q_{f_n}}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} q_{f_1} \Subset \varrho(q_{f_1})}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} \#(q_{f_n}) \Subset \#(\varrho(q_{f_n}))}}{\emptyset \vdash_{\mathcal{A},\mathcal{B}} \#(q_{f_n}) \Subset \{\#(q) \mid \mathcal{Q}_{F_{\mathcal{B}}}\}}$$
$$\emptyset \vdash_{\mathcal{A},\mathcal{B}} \{\#(q) \mid q \in \mathcal{Q}_{F_{\mathcal{A}}}\} \Subset \{\#(q) \mid q \in \mathcal{Q}_{F_{\mathcal{B}}}\}$$

Thus, we can ensure that for an automaton $\mathcal{A}^k_{\mathcal{R}}$ obtained by k completion steps from $\mathcal{A}^0$, there exists a proof $\prod$ of the statement $\emptyset \vdash_{\mathcal{A}^0,\mathcal{A}^k_{\mathcal{R}}} \{\#(q) \mid q \in \mathcal{Q}_{F_0} \Subset \{\#(q') \mid q' \in \mathcal{Q}_{F_k}\}$. This can be obtained by a simple combination of the two previous theorems.

Remark about restrictions: To obtain a decision procedure for all tree automata (and not only those obtained by completion), the "Weak-r" rule would have to be modified. This version is too weak; it does not take all possible cases for union construction into account.

Example 4 *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata s.t.:*

$$\mathcal{A} = \left\{ \begin{array}{rcl} a & \to & q_1 \\ b & \to & q_2 \\ c & \to & q_2 \\ f(q_1,q_2) & \to & \mathbf{q} \end{array} \right\} \text{ and } \mathcal{B} = \left\{ \begin{array}{rcl} a & \to & q'_1 \\ b & \to & q'_2 \\ c & \to & q'_3 \\ f(q'_1,q'_2) & \to & \mathbf{q}' \\ f(q'_1,q'_3) & \to & \mathbf{q}' \end{array} \right\}$$

Here we have $\mathcal{L}(\mathcal{A},q) = \mathcal{L}(\mathcal{B},q')$ but $\emptyset \vdash_{\mathcal{B},\mathcal{A}} q' \Subset q$ is clearly derivable whereas $\emptyset \vdash_{\mathcal{A},\mathcal{B}} q \Subset q'$ is not.

7.4 Complexity

As said in section 4, the standard algorithm for checking the inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is based on computing the complement automaton $\overline{\mathcal{B}}$. However, for non deterministic tree automata the size [CDG+02] of $\overline{\mathcal{B}}$ can be exponentially greater than the size of $\mathcal{B}$. The algorithm that we have proposed above has not this drawback and use only a memory size that is polynomial w.r.t. to the automata sizes.

Let $|\mathcal{Q}|$ be the maximum number of states in tree automata $\mathcal{A}$ and $\mathcal{B}$. The proof trees built using the deduction rules we gave are of height at most $|\mathcal{Q}|^2$. This is due to the rules 'Induction' and 'Axiom' ensuring that every inclusion problem $q \Subset q'$ will be analyzed only once per branch. Since $q \in \mathcal{Q}_\mathcal{A}$ and $q' \in \mathcal{Q}_\mathcal{B}$ and we know that the cardinal of $\mathcal{Q}_\mathcal{A}$ and $\mathcal{Q}_\mathcal{B}$ is bounded by $|\mathcal{Q}|$, the length of branch is at most bounded by $|\mathcal{Q}|^2$. Since, the `inclusion` function only constructs one branch of the proof tree at a time, the memory usage is thus bounded by $|\mathcal{Q}|^2$ and thus polynomial.

However, the time complexity of a straightforward implementation of this algorithm is exponential. Indeed, even if each couple $q \Subset q'$ is considered only once on each branch, the number of branches is exponential w.r.t. $\mathcal{Q}$ and the same couple $q \in q'$ may be analyzed once per branch. A very simple optimization of this algorithm is to table the result of the analysis of each $q \Subset q'$ and make it available to cut similar proof branches. Using this optimization, every couple is considered *only once* for the whole proof tree and, thus, lead to an overall polynomial time complexity, close to $|\mathcal{Q}|^2$. Nevertheless, the current Coq implementation is based on the non tabled version for two reasons. First, the proof of theorem `inclusion_sound` is more difficult on a tabled version of the Coq `inclusion` function. Second, on test cases, it appears that avoiding the exponential blow-up of memory was critical but that practical performances of the, potentially, exponential time algorithm are sufficient.

8 Formalization of closure by rewriting

In this part we aim at defining formally the `IsClosed` predicate, the function `closure` and prove the soundness of this function w.r.t. `IsClosed`. Recall that to check if a tree automaton $\mathcal{A} = \langle \mathcal{Q}_F,\ \Delta \rangle$ is closed w.r.t. a TRS $\mathcal{R}$, it is enough to prove that for all $t \in \mathcal{L}(\mathcal{A})$, if t' is reachable from t by $\rightarrow^*_{\mathcal{R}}$ then $t' \in \mathcal{L}(\mathcal{A})$. Now that we have defined in Coq rewriting and tree automata, we can define more formally the `IsClosed` predicate and recall the `closure_sound` theorem to prove:

```
Definition IsClosed (R : list rule) (A : t_aut) : Prop :=
   ∀ q t t', IsRec A.delta q t → Reachable R t t' → IsRec A.delta q t'.

Theorem closure_sound:
      ∀ R A', closure R A' = true → IsClosed R A'.
```

The algorithm to check closure of $\mathcal{A}$ by $\mathcal{R}$ computes for each rule $l \rightarrow r \in \mathcal{R}$ the full set of the substitutions σ s.t. $l\sigma \rightarrow^*_\Delta q$ and then, checks that $r\sigma \rightarrow^*_\Delta q$. Then, the correctness proof consists in showing that if `closure` answers true, then $\mathcal{L}(\mathcal{A})$ closed by $\rightarrow_{\mathcal{R}}$.

We now give some hints to define the `closure` function. First, for all rule $l \rightarrow r$ of $\mathcal{R}$, this function has to find all the substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and all the states $q \in \mathcal{Q}$ such that $l\sigma \rightarrow^*_\Delta q$. This is what we call the *matching-problem*. Second, this function has to check that for all the q and σ found, we have $r\sigma \rightarrow^*_\Delta q$. Third, in the correctness theorem, we have to show that all the substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ cover the set of substitutions on terms, i.e. of the form $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$, and hence cover all reachable terms.

We note $l \trianglelefteq q$ the matching problem consisting in finding all the substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and all the states $q \in \mathcal{Q}$ such that $l\sigma \rightarrow^*_\Delta q$. An algorithm solving this kind of problems was defined in [Gen97]. It consists in normalizing the formula $l \trianglelefteq q$ with the following deduction rules:

$$\text{(Unfold)}\ \frac{f(s_1,\ldots,s_n) \trianglelefteq f(q_1,\ldots,q_n)}{s_1 \trianglelefteq q_1 \wedge \cdots \wedge s_n \trianglelefteq q_n} \qquad \text{(Clash)}\ \frac{f(s_1,\ldots,s_n) \trianglelefteq g(q'_1,\ldots q'_m)}{\bot}$$

$$\text{(Config)}\ \frac{s \trianglelefteq q}{s \trianglelefteq c_1 \vee \cdots \vee s \trianglelefteq c_k \vee \bot} \text{if } s \notin \mathcal{X}, \text{ and } \{c_i \mid c_i \rightarrow q \in \Delta\}_1^k.$$

Moreover, after each application of one of this rules, the result is also rewritten into disjunctive normal form. using:

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

When normalization of the initial problem is terminated, we obtain a formula like $\bigvee_{i=1}^n \phi_i$ where $\phi_i = \bigwedge_{j=1}^m x_j^i \trianglelefteq q_j^i$ such that $x_j^i \in \mathcal{X}$ and $q_j^i \in \mathcal{Q}$. Each ϕ_i can be seen as a substitution $\sigma_i = \{x_j^i \mapsto q_j^i\}$.

Theorem 7 *[Gen97] Substitutions σ_i obtained by the matching algorithm are the only substitutions s.t. $s\sigma_i \rightarrow^*_\Delta q$.*

The implementation of the `matching` function in Coq is very close to this algorithm. We implement disjunction as lists where $\bot$ is mapped to `nil`. The Coq signature of the matching function is `Delta.t →state →term →list substitution`. For this algorithm, the termination is bound by a syntactic argument. We can define it easily in Coq by a simple structural recursion over the `term` which has to be matched. Moreover, the soundness and completeness properties, corresponding to Theorem 7, can be defined in Coq as follows:

```
Theorem matching_sound :
   ∀ D q l s, In s (matching D q l) → IsRec D q (l @ s).

Theorem matching_complete :
   ∀ D q l s, IsRec D q (l @ s) → In s (matching D q l).
```

The second part of the `closure` function consists in verifying that for each substitution σ s.t. $l\sigma \rightarrow_\Delta^* q$, we also have $r\sigma \rightarrow_\Delta^* q$. This job is performed using the `all_red` function, we define, whose purpose is to check that this property is true for all the found substitutions. Then, we only need to prove the soundness of this function using the following Coq theorem:

```
Theorem all_red_sound :
  ∀ D q r sigmas,
     all_red D q r sigmas = true → ∀ s, In s sigmas → IsRed D q (r@s).
```

By combining the `matching` and the `all_red` functions, we obtain the algorithm for checking up all critical pairs found at state q and for the rule $l \rightarrow r$. We define the combination as:

```
   stlisting

\begin{lstlisting}
Theorem closure_at_state_sound :
   ∀ D q l r, closure_at_state D q l r = true →
      (∀ s, IsRed D q (l @ s) → IsRed D q (r @ s)).
```

Given a rule $l \rightarrow r$ and a state q, this algorithm answers `true` if for all substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ s.t. $l\sigma \rightarrow_\Delta^* q$ then $r\sigma \rightarrow_\Delta^* q$. Now that we have proved this result for substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$, we have to prove that it implies the same property for substitutions $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$, this is Lemma 3. On the opposite, to prove that every reachable term of $\mathcal{T}(\mathcal{F})$ will be covered by a configuration of $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ in Δ, we have to prove that if there exists a substitution $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$, then we can construct a corresponding substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$, this is Lemma 4.

Lemma 3 *Given a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ a substitution s.t. $u\sigma \rightarrow_\Delta^* q$, if we have a substitution $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ s.t. $\forall x \in \mathcal{D}om(\sigma) : \sigma' x \in \mathcal{L}(\Delta, \sigma x)$, then we have $u\sigma' \rightarrow_\Delta^* q$ and thus $u\sigma' \in \mathcal{L}(\Delta, q)$.*

Roughly, if the left or right-hand side u of a rewriting rule matches a configuration $u\sigma \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $u\sigma \rightarrow_\Delta^* q$ then, all terms $u\sigma' \in \mathcal{T}(\mathcal{F})$, matched by u, are also reducible into q, i.e. $u\sigma' \rightarrow_\Delta^* q$ and $u\sigma' \in \mathcal{L}(\Delta, q)$.

Lemma 4 *Given a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, if there exists a substitution $\sigma' : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $u\sigma' \rightarrow_\Delta^* q$, then there exists a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ s.t. $\sigma' x \in \mathcal{L}(\Delta, \sigma x)$ and $u\sigma \rightarrow_\Delta^* q$.*

In the same way, for all term $u\sigma' \in \mathcal{L}(\Delta, q)$ that can be matched by u, there exists a configuration $u\sigma$ s.t. $u\sigma' \rightarrow_\Delta^* u\sigma$ and $u\sigma \rightarrow_\Delta^* q$. Using those two lemmas, we can conclude that for all term $t \in \mathcal{L}(\Delta, q)$ rewritten in t' at the topmost position by $l \rightarrow r$, then $t' \in \mathcal{L}(\Delta, q)$. This property is easily lifted as a property of the `closure` function for all states of $\mathcal{Q}$ and using all rules of $\mathcal{R}$ at topmost position.

```
Theorem closure_sound_0 :
   ∀ D R, closure D R = true →
      ∀ q lr, In q (states D) → In lr R →
         ∀ t t', IsRec D q t → TRew lr t t' → IsRec D q t'.
```

The next step consists in showing the same property but using rewriting at any position, hence proving the same theorem with `Rew` (general rewriting) instead of `TRew` (topmost rewriting) between t and t'.

```
Theorem closure_sound_1 :
   ∀ D R, closure D R = true →
      ∀ q lr, In q (states D) → In lr R →
         ∀ t t', IsRec D q t → Rew lr t t' → IsRec D q t'.
```

Finally, the `closure_sound` general theorem is shown by using a reflexive and transitive application of this `closure_sound_1` in order to deal with any term t' that can be reached using the `Reachable` predicate.

9 Benchmarks

From the Coq formal specification, we have extracted an Ocaml checker implementation. In the following table, we have collected several benchmarks. For each test, we give the size of the two tree automata (initial $\mathcal{A}^0$ and completed $\mathcal{A}^*_\mathcal{R}$) as number of transitions/number of states. For each TRS $\mathcal{R}$ we give the number of rules. The 'CS' column gives the number of completion steps necessary to complete $\mathcal{A}^0$ into $\mathcal{A}^*_\mathcal{R}$ and 'CT' gives the completion time. The 'CKT' column gives the time for the checker to certify the $\mathcal{A}^*_\mathcal{R}$ and the 'CKM' gives the memory usage. The important thing to observe here is that, the completion time is very long (sometimes more than 24 hours), the *checking* of the corresponding automaton is always fast (a matter of seconds).

The four tests are Java programs translated into term rewriting systems using the technique detailed in [BGJL07]. All of them are completed using Timbuk except the example `List2.java` which has been completed using a completion tool under development by Yohan Boichut and Emilie Balland. This shows that the completed automaton produced by a new and fully optimzed tool is also accepted by our checker. The `List1.java` and `List2.java` corresponds to the same Java program but with slightly different encoding into TRS and approximations. The `Ex_poly.java` is the example given in [BGJL07] and the

`Bad_Fixp` is the same problem as `Ex_poly.java` except that the completed automaton $\mathcal{A}^*_{\mathcal{R}}$ has been intentionally corrupted. Thus, this is thus not a valid fixpoint and rejected by the checker.

Name	$\mathcal{A}^0$	$\mathcal{A}^*_{\mathcal{R}}$	$\mathcal{R}$	CS	CT	CKT	CKM
`List1.java`	118/82	422/219	228	180	$\approx$ 3 days	0,9s	2,3 Mo
`List2.java`	1/1	954/364	308	473	1h30	2,2s	3,1 Mo
`Ex_poly.java`	88/45	951/352	264	161	$\approx$ 1 day	2,5s	3,3 Mo
`Bad_Fixp`	88/45	949/352	264	161	$\approx$ 1 day	1,6s	3,2 Mo

10 Conclusion and further research

In this paper we have defined a Coq checker for tree automata completion. The first characteristic of the work presented here is that the checker does not validate a specific implementation of completion but, instead, the result. As a consequence, the checker remains valid even if the implementation of the completion algorithm changes or is optimized. A second salient feature is that the code of the checker is directly generated from the correctness proof of its verified Coq specification through the Coq extraction mechanism. Third, we have payed particular attention to the formalization of the checker in order not to lose efficiency to obtain the certification. We have defined a specific inclusion algorithm in order to avoid the usual exponential blow-up obtained with the standard inclusion algorithm. We have defined the Coq formal specification so that it was possible to extract an independent OCaml checker. Finally, we made an extensive use of efficient formal data structures leading to more complex proof but also to faster extracted checker. An extension for non left-linear TRS, which are sometimes necessary for specifying cryptographic protocols, is under development. Since many different kind of analysis can be expressed as reachability problems over tree automata, and since verification of completed automata revealed to be very efficient, we aim at using this technique in a PCC architecture where tree automata are viewed as program certificates. At last, note that even if this checker is external to Coq, we can use the correction proof of the checker and the Coq reflexivity mechanism to lift-up the external verification into a proof in the Coq system. This can be necessary to perform efficient unreachability proofs on rewriting systems in Coq using an external completion tool.

References

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security

protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[BC04] 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.

[BD04] G. Barthe and G. Dufay. A tool-assisted framework for certified bytecode verification. In *FASE'04*, volume 2984 of *LNCS*, pages 99–113. Springer, 2004.

[BGJL07] Y. Boichut, T. Genet, T. Jensen, and L. Leroux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *RTA*, volume 4533 of *LNCS*, pages 48–62. Springer Verlag, 2007.

[BHK04] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic Approximation for the Verification of Cryptographic Protocols. In *Proc. AVIS'2004, joint to ETAPS'04, Barcelona (Spain)*, 2004.

[BJP06] F. Besson, T. Jensen, and D. Pichardie. Proof-carrying code from certified abstract interpretation and fixpoint compression. *Theor. Comput. Sci*, 364(3):273–291, 2006.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[CDG+02] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata/`, 2002.

[CJPR05] D. Cachera, T. Jensen, P. Pichardie, and V. Rusu. Extracting a data flow analyser in constructive logic. *Theor. Comput. Sci.*, 342(1):56–78, 2005.

[FGVTT04] G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *JAR*, 33 (3-4):341–383, 2004.

[Gen97] Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms (extended version). Technical Report RR-3325, INRIA, 1997.

[Gen98] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[GK00] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.

[GTTVTT03] T. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems. In *In Proceedings of Workshop on Issues in the Theory of Security*, 2003.

[GVTT00] T. Genet and V. Viet Triem Tong. Timbuk 2.0 – a Tree Automata Library. IRISA / Université de Rennes 1, 2000. `http://www.irisa.fr/lande/genet/timbuk/`.

[KN03] G. Klein and T. Nipkow. Verified bytecode verifiers. *TCS*, 298, 2003.

[LT00] P. Letouzey and L. Théry. Formalizing stalmarck's algorithm in coq. In *Proc. of TPHOL'00*, volume 1869 of *LNCS*. Springer, 2000.

[RGL01] X. Rival and Jean Goubault-Larrecq. Experiments with finite tree automata in coq. In *Proc. of TPHOL'01*, LNCS. Springer, 2001.

[Tak04] T. Takai. A Verification Technique Using Term Rewriting Systems and Abstract Interpretation. In *Proc. 15th RTA Conf., Aachen (Germany)*, volume 3091 of *LNCS*, pages 119–133. Springer, 2004.

[ZD06] R. Zunino and P. Degano. Handling exp, $\times$ (and timestamps) in protocol analysis. In *Proc. of FOSSACS'06*, volume 3921 of *LNCS*, pages 413–427. Springer, 2006.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399