Approximation-based Tree Regular Model-Checking

Y. Boichut[(a)*] and P.-C. Héam[(b)†] and O. Kouchnarenko[(c)]
[(a)] LIFO, Université d'Orléans
Rue Léonard de Vinci,
BP 6759, F-4067 Orleans Cedex 2, FRANCE
Yohan.Boichut@univ-orleans.fr

[(b)] LSV, INRIA-CNRS, ENS de Cachan
61 avenue du Président Wilson,
94235 Cachan, FRANCE
pcheam@lsv.ens-cachan.fr

[(c)] LIFC, INRIA-CASSIS
16, route de Gray,
25030 Besançon, FRANCE
olga.kouchnarenko@lifc.univ-fcomte.fr

Abstract

This paper addresses the following general problem of tree regular model-checking: decide whether $\mathcal{R}^*(L) \cap L_p = \emptyset$ where $\mathcal{R}^*$ is the reflexive and transitive closure of a successor relation induced by a term rewriting system $\mathcal{R}$, and L and L_p are both regular tree languages. We develop an automatic approximation-based technique to handle this – undecidable in general – problem in most practical cases, extending a recent work by Feuillade, Genet and Viet Triem Tong. We also make this approach fully automatic for practical validation of security protocols.

Key-words: Verification, model-checking, regular languages, security protocols.
Computing Reviews Categories: D.2.4 and F.4.2.

1 Introduction

Automatic verification of software systems is one of the most challenging research problems in computer aided verification. In this context, regular model-checking has

*This author was at IRISA-LANDES, Rennes, France when performing the work.
†This work was done while this author was at LIFC, INRIA-CASSIS, Besançon, France

been proposed as a general framework for analysing and verifying infinite state systems. In this framework, systems are modelled using regular representations: the systems configurations are modelled by finite words or trees (of unbounded size) and the dynamic behaviour of systems is modelled either by a transducer or a (term) rewriting system. Afterwards, a system reachability-based analysis is reduced to the regular languages closure computation under (term) rewriting systems: given a regular language L, a relation $\mathcal{R}$ induced by a (term) rewriting system and a regular set L_P of *bad configurations*, the problem is to decide whether $\mathcal{R}^*(L) \cap L_p = \emptyset$ where $\mathcal{R}^*$ is the reflexive and transitive closure of $\mathcal{R}$. Since $\mathcal{R}^*(L)$ is in general neither regular nor decidable, several approaches handle restricted cases of this problem.

In this paper we address this problem for tree regular languages by automatically computing over- and under-approximations of $\mathcal{R}^*(L)$. Computing an over-approximation K_{over} of $\mathcal{R}^*(L)$ may be useful for the problem if $K_{\text{over}} \cap L_p = \emptyset$, proving that $\mathcal{R}^*(L) \cap L_p = \emptyset$. Dually, under-approximation may be suitable to prove that $\mathcal{R}^*(L) \cap L_p \neq \emptyset$. This approach is relevant if the computed approximations are not too coarse. Another important point is that in general, there are some restrictions on the rewriting systems in order to ensure the soundness of the above approach. This paper 1) generalises this approach for any kind of term rewriting systems, and 2) describes its successful application for the security protocol analysis.

1.1 Contributions

This paper extends an expert-human guided approximation technique introduced in [FGVTT04] for left-linear term-rewriting systems. The contributions of this paper are:

1. We show how to extend the over-approximation approach of [FGVTT04] to all term rewriting systems,
2. We show how the under-approximation approach of [FGVTT04] may be extended to a suitable sub-class of non-left-linear term-rewriting systems,
3. We explain how 1. can be efficiently implemented, particularly for quadratic rewriting rules that are very useful in practice.
4. We explain how to make the approach fully automatic and how to successfully exploit this approach in the context of security protocols verification.

Notice that 1. and 2. were respectively presented in [BHK06] and in [BHK07] without proofs nor explicit examples.

1.2 Related Works

Model checking is a central verification technique based on state exploration. Since for infinite state systems an exhaustive exploration is impossible, several symbolic techniques (consisting in representing infinite sets of states by a symbolic finite representation) have been developed.

1.2.1 Regular Model Checking

Regular Model Checking (RMC for short) is a symbolic approach using finite automata [ABJ98] [BW98] [BG96] [PS00] [JN00] [DLS02] (and sometimes regular expressions [BMT07]) in order to encode infinite sets of states. Most of these works deal with word automata (see [FL02] for automata with Presuburger constraints, [BFL04] for automata with counters, [FWW97] for pushdown automata, etc). These techniques have been successfully used for verifying parametrized tree networks and data-flow analysis of multithreaded programs [BT02], for lossy communicating system modelling and verification [AAB99] [CF05] or for static analysis of programs [BET03] [LJ07].

Tree data structures are more complex objects, and adapting or developing new techniques remains a deep challenge. In [BT02], given a tree transducer T, the authors explore how to use acceleration techniques to compute, under several hypotheses, the transducer T^*. In [BET05], the authors investigate how to use constraint systems in order to address the reachability problem for thread-based programs. The work [BHRV06] extends the abstract regular model-checking technique from words to trees. In this work reachability sets are over-approximated using a predicate abstraction. When computing reachability sets, involved automata cope with a combinatorial state-space blow up; there are works to reduce these automata using simulation-based state-space partitions. The work in [ALdR06] follows this approach and develops it for tree automata.

1.2.2 Term Rewriting Systems and Reachability Analysis on Regular Languages

Given a term rewriting system $\mathcal{R}$ and two ground terms s and t, deciding whether $s \rightarrow_{\mathcal{R}}^{*} t$ is a central question in automatic proof theory. This problem is shown decidable for term rewriting systems which are terminating but it is undecidable in the general case. Several syntactic classes of term rewriting systems have been pointed out to have a decidable accessibility problem, for instance by providing an algorithm to compute $\mathcal{R}^*(L)$ when L is a regular tree language [DT90] [CDGS91] [GT95] [Jac96] [RV02] [Sal88].

In [FGVTT04], authors focus on a general completion based human-guided technique. This technique has been successfully used (not automatically) to prove the security of cryptographic protocols [GK00] and recently Java Bytecode programs [BGJLR07]. This framework was extended in [OT05] to languages accepted by AC-tree automata.

1.2.3 Verification of Security Protocols

The challenge we want to take on is to automate [FGVTT04] for the security protocol verification in a very general context. Cryptographic protocols are widely used to secure information exchange over open modern networks. It is now widely accepted that formal analysis can provide the level of assurance required by protocols both the developers and the users. But, whatever the used formal model, analysing cryptographic protocols is a complex task because the set of configurations to consider is very large,

and can even be infinite. Indeed, any number of sessions (sequential or parallel executions) of protocols, sessions interleaving, any size of messages, algebraic properties of encryption or data structures give rise to infinite-state systems. In the context of protocols verification, the security problem we are dealing with consists in deciding whether a protocol preserves secrecy against an intruder, or not.

Complexity Issues. For this problem, current model-checking based verification methods can be applied whenever the number of participants and the number of sessions between the agents are bounded. In this case, the protocol security problem is co-NP-complete [RT01]. The work in [Tru05] presents new decidability results for a bounded number of sessions, restricted to the case where the initial knowledge of the intruder is a regular language and under the assumption that the keys used in protocols are atomic. When the number of sessions is unbounded, the security problem of cryptographic protocols becomes undecidable, even when the length of the messages is bounded [DLMS99] [DLMS04]. Decidability can be recovered by adding some restrictions to protocols as, for instance, in [CLC05]. Another way to circumvent the problem is to employ abstraction-based approximation methods [Mon99][GK00].

Theoretical Works and Tools. A lot of theoretical work has been done for analysing cryptographic protocols for different kinds of protocols and intruders models (wireless network [NH06], time-stamps [BEL05], combinations of theories [CR05], abelian groups [LLT07], homomorphisms [CT03], isomorphisms [Del06], xor [CLS03] [CKRT05], probabilistic encryption [DJ06], voting protocols [KR05] [NAN05], non-repudiation protocols [AG02], Diffie-Hellman like protocols [GRV05], e-mail-certification [AB05], etc), with different approaches (tree automata [OT05][KW04], SAT-solving [AC05], model-checking [BMV03]) and many tools (ProVerif [Bla01], Athena [Son99], AVISPA [ABB+05], Hermes [BLP03], Murφ [MMS97], ...) have been developed.

Close Works on Automated Protocol Analysis. An independent work close to our approach is presented in [ZD06] where authors use a process algebra based model and constraints-guided over-approximations. This approach has been successfully applied to the complex Kerberos protocol. However, [ZD06] does not presently handle under-approximations.

There already exists a number of tools in the literature that provide automated semi-decision procedures for security protocols with an unbounded number of sessions: Blanchet's ProVerif, Ernie Cohen's TAPS, Isabelle proof assistant for the Protocol Composition Logic (PCL) from Stanford.

The CL-AtSe tool (Constraint Logic based Attack Searcher) [CKRT05] now supports complete analysis of cryptographic protocols modulo the `xor`, including all the intruder deduction rules for that operator, and modulo the `exp` except for the rule $g_1 = g_2$ (i.e. exponentials are tagged). CL-AtSe analyses are performed for a bounded number of sessions.

The On-the-fly Model-Checker (OFMC) [BMV03] tool-set, is based on two symbolic techniques and now supports the specification of cryptographic operators algebraic properties, and typed or untyped protocol models, in the context of a bounded number of sessions. sessions, [Möd07] develops an abstract interpretation based approach.

The Proverif tool [Bla01] allows an unbounded number of sessions but – like in our models – abstractions are performed on fresh data and thus false attacks can be detected. Moreover, in [BAF08], the authors note that their technique does have limitations, and in particular, it does not apply to some equational theories.

The recent Scyther tool [Cre06] can verify protocols with an unbounded number of sessions and nonces. It can handle verification of complex authentication properties, handle non-atomic keys, and generate correct attacks. A performance comparison between Scyther and a number of other tools has been detailed in [CL07]. Notice that [CL07] reports on a set of protocols that excludes protocols using algebraic properties.

One of the new features of the Maude-NPA tool is that it allows to equationally reason about security when facing attempted attacks on low-level algebraic properties of the functions used in a protocol such as, for example, associativity-commutativity, Boolean theory, and some forms of modular exponentiation [EMM07]. The Maude-NPA tool follows an approach similar to that of OFMC since the authors consider depth parameters for unification problems in some equational theories. However, the Maude-NPA tool needs the help of expert users.

In the survey [CDL06], the authors emphasise the fact that the results often remain theoretical, and very few implementations automatically verify protocols with algebraic properties.

1.3 Layout of the paper

Section 2 introduces notations and the basic completion approach. Next, Section 3 presents the main theoretical contributions of the paper. We introduce the notion of $(l \to r)$-substitutions in Section 3.1. We show how it can be used to develop an over-approximation based technique for tree regular model-checking in Section 3.2. The case of under-approximations is handled in Section 3.4, while Section 3.3 is dedicated to an example. Section 4 exposes how the techniques are successfully exploited for analysing security protocol.

2 Formal Background

As for prerequisites, the reader is expected to be familiar with basic notions on term rewriting systems and tree automata. We just recall the terminology which is consistent with [CDG+02], thus making our exposition as self-contained as possible.

2.1 Notations

Given the set $\mathbb{N}$ of natural integers, $\mathbb{N}^*$ denotes the finite strings over $\mathbb{N}$. Let $\mathcal{F}$ be a finite set of symbols, associated with an arity function $ar : \mathcal{F} \to \mathbb{N}$. The set of symbols of $\mathcal{F}$ of arity i is denoted $\mathcal{F}_i$. Let $\mathcal{X}$ be a countable set of variables. We assume that $\mathcal{X} \cap \mathcal{F} = \emptyset$. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the set of terms, and $\mathcal{T}(\mathcal{F})$ denotes the set of ground terms (terms without variables).

A finite ordered tree t over a set of labels $(\mathcal{F}, \mathcal{X})$ is a function from a prefix-closed set $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ to $\mathcal{F} \cup \mathcal{X}$. A term t over $\mathcal{F} \cup \mathcal{X}$ is a labelled tree whose domain $\mathcal{P}os(t)$

satisfies the following properties: 1) $\mathcal{P}os(t)$ is non-empty and prefix closed, 2) for each $p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{F}_n$ (with $n \neq 0$), then $\{i \mid p.i \in \mathcal{P}os(t)\} = \{1, \ldots, n\}$, and 3) for each $p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{X}$ or $t(p) \in \mathcal{F}_0$, then $\{i \mid p.i \in \mathcal{P}os(t)\} = \emptyset$. The empty sequence ϵ denotes the top-most position.

Each element of $\mathcal{P}os(t)$ is called a position of t. For each subset $\mathcal{K}$ of $\mathcal{F} \cup \mathcal{X}$ and each term t $\mathcal{P}os_{\mathcal{K}}(t)$ is the subset of positions p's of t such that $t(p) \in \mathcal{K}$. Each position p of t such that $t(p) \in \mathcal{F}$, is called a functional position.

A subterm $t_{|p}$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ at position p is defined by the following: $\mathcal{P}os(t_{|p}) = \{w \in \mathbb{N}^* \mid p.w \in \mathcal{P}os(t)\}$, and for all $j \in \mathcal{P}os(t_{|p})$, $t_{|p}(j) = t(p.j)$. The term $t[s]_p$ is obtained from t by replacing the subterm $t_{|p}$ by s. $\mathcal{V}ar(t)$ is the set of variables occurring within t and is formally defined as follows: $\mathcal{V}ar(t) = \{t(p) \mid p \in \mathcal{P}os(t) \wedge t(p) \in \mathcal{X}\}$.

For all sets A and B, we denote by $\Sigma(A, B)$ the set of functions from A to B. If $\sigma \in \Sigma(\mathcal{X}, B)$, then for each term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, the term $t\sigma$ is obtained from t by replacing for each $x \in \mathcal{X}$, the variable x by $\sigma(x)$.

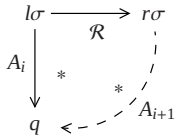
A term rewriting system (TRS for short) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a finite set of pairs (l, r) from $\mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$, written $l \to r$, such that the set of variables occurring in r is included in the set of variables of l. A TRS is left-linear if for each rule $l \to r$, every variable occurring in l occurs once at most. For each ground term t, we denote by $\mathcal{R}(\{t\})$ the set of ground terms t' such that there exist a rule $l \to r$ of $\mathcal{R}$, a function $\mu \in \Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ and a position p of t satisfying $t_{|p} = l\mu$ and $t' = t[r\mu]_p$. The relation $\{(t, t') \mid t' \in \mathcal{R}(\{t\})\}$ is classically denoted $\to_{\mathcal{R}}$. If $t \to_{\mathcal{R}} t'$ for $t, t' \in \mathcal{T}(\mathcal{F})$, then t is a *rewriting predecessor* of t' and t' is *rewriting successor* of t. For a set of ground terms B, $\mathcal{R}^*(B)$ is the set of ground terms related to an element of B modulo the reflexive-transitive closure of $\to_{\mathcal{R}}$.

A tree automaton $\mathcal{A}$ is a tuple (Q, Δ, F), where Q is the set of states, Δ the transition set, and F the set of final states. Transitions are rewriting rules of the form $f(q_1, \ldots, q_k) \to q_{k+1}$, where $f \in \mathcal{F}_k$ and the q_i's are in Q. Such transitions are so called normalised transitions. A term $t \in \mathcal{T}(\mathcal{F})$ is accepted or recognised by $\mathcal{A}$ if there exists $q \in F$ such that $t \to_{\Delta}^* q$ (we also write $t \to_{\mathcal{A}}^* q$). The set of terms accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. For each state $q \in Q$, we write $\mathcal{L}(\mathcal{A}, q)$ for the tree language $\mathcal{L}((Q, \Delta, \{q\}))$. A tree automaton is finite if its set of transitions is finite.

2.2 Completion

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, for several classes of automata and TRSs, the tree automata completion algorithm computes a tree automaton $\mathcal{A}_k$ such that $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ when it is possible and such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ otherwise [GK00][FGVTT04]. The tree automata completion works as follows. From $\mathcal{A} = \mathcal{A}_0$ the completion builds a sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ of automata such that if $s \in \mathcal{L}(\mathcal{A}_i)$ and $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$. If the automaton $\mathcal{A}_k$ is a fixpoint, i.e. if $\mathcal{A}_k = \mathcal{A}_{k+1}$, then we have $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ (or $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ for some restrictive cases of [FGVTT04]). To build $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, a *completion step* is performed which consists in finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto Q$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in Q$ satisfying $l\sigma \to_{\mathcal{A}_i}^* q$ and $r\sigma \not\to_{\mathcal{A}_i}^* q$. For every critical pair $l\sigma \to_{\mathcal{A}_i}^* q$ and $r\sigma \not\to_{\mathcal{A}_i}^* q$ detected between $\mathcal{R}$

and $\mathcal{A}_i$, $\mathcal{A}_{i+1}$ is constructed by adding new transitions to $\mathcal{A}_i$ to recognise $r\sigma$ in q, i.e. $r\sigma \rightarrow_{\mathcal{A}_{i+1}} q$.



However, the transition $r\sigma \rightarrow q$ is not necessarily a normalised transition of the form $f(q_1, \ldots, q_n) \rightarrow q'$ and so has to be normalised first. For example, to normalise a transition of the form $f(g(a), h(q')) \rightarrow q$, we need to find some states q_1, q_2, q_3 and replace the previous transition by the following set of normalised transitions: $\{a \rightarrow q_1, g(q_1) \rightarrow q_2, h(q') \rightarrow q_3, f(q_2, q_3) \rightarrow q\}$.

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

The above method does not work for all TRSs. For instance, consider a constant A and the tree automaton $\mathcal{A} = (\{q_1, q_2, q_f\}, \{A \rightarrow q_1, A \rightarrow q_2, f(q_1, q_2) \rightarrow q_f\}, \{q_f\})$ and the TRS $\mathcal{R} = \{f(x, x) \rightarrow g(x)\}$. There is no substitution σ such that $l\sigma \rightarrow^*_{\mathcal{A}} q$, for a q in $\{q_1, q_2, q_f\}$. Thus, following the procedure, there is no transition to add. But $f(A, A) \in \mathcal{L}(\mathcal{A})$. Thus $g(A) \in \mathcal{R}(\mathcal{L}(\mathcal{A}))$. Since $g(A) \notin \mathcal{L}(\mathcal{A})$, the fixpoint automaton obtained is not an over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

This constraint may prevent someone from specifying a system, in particular concerning protocols. Unfortunately, to be sound, the approximation-based analysis described in [GK00] requires using of left-linear TRSs. Nevertheless, this method can still be applied to some non left-linear TRSs, which satisfy some weaker conditions. In [FGVTT04] the authors propose new linearity conditions. However, these new conditions are not well-adapted to be automatically checked in the sense that, they are verified as soon as the computation is over. And if these conditions are not satisfied then the computation must be done again by changing some inputs of the approximation technique.

3 Main Results – Sound Completion for any Kind of TRSs

The challenge of this section is to describe an alternative way for the technique [FGVTT04] in order to make it sound for any kind of TRSs. We first introduce in Section 3.1 the notion of $(l \rightarrow r)$-substitutions and the normalisation related to it. This kind of substitution allows variables to store different values. Second, we present in Section 3.2 an extension of the completion procedure to any TRS for computing sound over-approximations. This algorithm is then detailed on an example in Section 3.3. Third,

Section 3.4 explores how the technique can be used in a restrictive case in order to provide under-approximations for non left-linear TRSs. Fourth, Section 3.5 discusses theoretical and practical aspects for applying the developed techniques to a large class of applications.

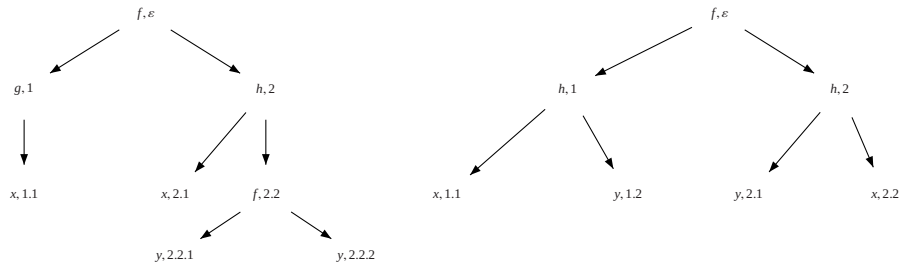
3.1 $(l \to r)$-substitutions, Normalisation

In this technical subsection, we define the notion of a $(l \to r)$-substitution suitable for the present work.

Definition 3.1 *Let $\mathcal{R}$ be a term rewriting system and $l \to r \in \mathcal{R}$. A $(l \to r)$-substitution is an application from $\mathcal{P}\text{os}_\mathcal{X}(l)$ into $\mathcal{Q}$.*

Let $l \to r \in \mathcal{R}$ and σ be a $(l \to r)$-substitution. We denote by $l\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined as follows: $\mathcal{P}\text{os}(l\sigma) = \mathcal{P}\text{os}(l)$, and for each $p \in \mathcal{P}\text{os}(l)$, if $p \in \mathcal{P}\text{os}_\mathcal{X}(l)$ then $l\sigma(p) = \sigma(l(p))$, otherwise $l\sigma(p) = l(p)$. Similarly, $r\sigma$ is the term in $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined by: $\mathcal{P}\text{os}(r\sigma) = \mathcal{P}\text{os}(r)$, and for each $p \in \mathcal{P}\text{os}(r)$, if $p \notin \mathcal{P}\text{os}_\mathcal{X}(r)$ then $r\sigma(p) = r(p)$ and $r\sigma(p) = \sigma(l(p'))$ otherwise, where $p' = \min(\mathcal{P}\text{os}_{r(p)}(l))$ (positions are lexicographically ordered). Let us note that the above choice (i.e. the minimal position for p') is arbitrary. Basically, every position in r where the considered variable occurs, may be chosen. The heuristics to chose this position have not been investigated yet.

Example 3.1 *Let us consider $l = f(g(x), h(x, f(y, y)))$ and $r = f(h(x, y), h(y, x))$ represented by the following trees (elements after the comma are the positions in the term; l is represented on the left and r on the right):*



Variable positions of l are 1.1 *and* 2.1 *for x, and* 2.2.1 *and* 2.2.2 *for y. Let $\sigma(1.1) = q_1$, $\sigma(2.1) = q_2$, $\sigma(2.2.1) = q_3$ and $\sigma(2.2.2) = q_4$; σ is a $(l \to r)$-substitution and $l\sigma = f(g(q_1), h(q_2, f(q_3, q_4)))$ is the term obtained from l by substituting the variable in position p by $\sigma(p)$. Now we explain how to compute $r\sigma$. The minimal position where x [resp. y] occurs in l is* 1.1 *[resp.* 2.2.1*]. Thus $r\sigma$ is obtained from r by substituting all x's in r by $\sigma(1.1) = q_1$ and all y's by $\sigma(2.2.1) = q_3$. Thus $r\sigma = f(h(q_1, q_3), h(q_3, q_1))$.*

As mentioned before, the completion procedure does not work for all tree automata and TRSs. That is why we introduce a notion of compatibility between finite tree-automata and $(l \to r)$-substitutions. The intuition behind the next definition is that

different occurrences of a variable may be substituted by different states if there exists a term recognised by all these states, at least. Notice that the condition required below is weaker than the conditions in [FGVTT04]. Moreover, it is more general and can be applied to a larger class of applications.

Definition 3.2 *Let $\mathcal{A}$ be a finite tree automaton. We say that a $(l \to r)$-substitution σ is $\mathcal{A}$-compatible if for each $x \in \mathcal{V}\mathrm{ar}(l)$,*

$$\bigcap_{p \in \mathcal{P}\mathrm{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)) \neq \emptyset.$$

Example 3.2 *Let $\mathcal{A}_{\mathrm{exe}} = (\{q_0, q_f\}, \Delta_{\mathrm{exe}}, \{q_f\})$ with the set of transitions $\Delta_{\mathrm{exe}} = \{A \to q_0, A \to q_f, f(q_f, q_0) \to q_f, h(q_0, q_0) \to q_0\}$. Let $\mathcal{R}_{\mathrm{exe}}$ be the TRS such that $\mathcal{R}_{\mathrm{exe}} = \{f(x, h(x, y)) \to h(A, x)\}$. The automaton $\mathcal{A}_{\mathrm{exe}}$ recognises the set of trees such that every path from the root to a leaf is of the form f^*h^*A. Let us consider the substitution σ_{exe} defined by $\sigma_{\mathrm{exe}}(1) = q_f$, $\sigma_{\mathrm{exe}}(2.1) = q_0$ and $\sigma_{\mathrm{exe}}(2.2) = q_0$. The tree $t = A$ can be reduced to q_f and belongs to $\mathcal{L}(\mathcal{A}, \sigma_{\mathrm{exe}}(1))$. Furthermore $t \to q_0$, so $t \in \mathcal{L}(\mathcal{A}, \sigma_{\mathrm{exe}}(2.2))$. Therefore σ_{exe} is $\mathcal{A}$-compatible.*

The notion of normalisation and approximation functions are close to the ones given in [FGVTT04][BHK05]. Indeed, the definitions below are simply adapted to our notion of $(l \to r)$-substitutions.

Definition 3.3 *Let $\mathcal{A}$ be a finite tree automaton. An approximation function (for $\mathcal{A}$) is a function which associates a function from $\mathcal{P}\mathrm{os}(r)$ to Q to each tuple $(l \to r, \sigma, q)$, where $l \to r \in \mathcal{R}$, σ is an $\mathcal{A}$-compatible $(l \to r)$-substitution and q a state of $\mathcal{A}$.*

Example 3.3 *Consider the automaton $\mathcal{A}_{\mathrm{exe}}$, the TRS $\mathcal{R}_{\mathrm{exe}}$ and the substitution σ_{exe} defined in Example 3.2. For σ_{exe}, an approximation function γ_{exe} may be defined by*

$$\gamma_{\mathrm{exe}}(l \to r, \sigma_{\mathrm{exe}}, q_0) : \{\varepsilon \mapsto q_1,\ 1 \mapsto q_2,\ 2 \mapsto q_f\}$$

To totally define γ_{exe}, the others (finitely many) $\mathcal{A}_{\mathrm{exe}}$-compatible substitutions should be considered too.

The notion of normalisation below is basic. The only difference comes from our notion of $(l \to r)$-substitutions.

Definition 3.4 *Let $\mathcal{A} = (Q_0, \Delta_0, F_0)$ be a finite tree automaton, γ an approximation function for $\mathcal{A}$, $l \to r \in \mathcal{R}$, σ an $\mathcal{A}$-compatible $(l \to r)$-substitution, and q a state of $\mathcal{A}$. We denote by* $\mathrm{Norm}_\gamma(l \to r, \sigma, q)$ *the following set of transitions, called* normalisation *of $(l \to r, \sigma, q)$:*

$$\begin{aligned}\{f(q_1, \ldots, q_k) \to q' \mid & p \in \mathcal{P}\mathrm{os}_{\mathcal{F}}(r),\ r(p) = f,\\ & q' = q \text{ if } p = \varepsilon \text{ otherwise } q' = \gamma(l \to r, \sigma, q)(p)\\ & q_i = \gamma(l \to r, \sigma, q)(p.i) \text{ if } p.i \notin \mathcal{P}\mathrm{os}_{\mathcal{X}}(r),\\ & q_i = \sigma(\min\{p' \in \mathcal{P}\mathrm{os}_{\mathcal{X}}(l) \mid l(p') = r(p.i)\}) \text{ otherwise}\}\end{aligned}$$

The min *is computed for the lexicographic order.*

Notice that the set $\{p' \in \mathcal{P}os_\mathcal{X}(l) \mid l(p') = r(p.i)\}$ used in the above definition is not empty as soon as $\mathcal{V}ar(l)$ is not empty. Indeed, in a TRS, variables occurring in its right hand-sides must, by definition, occur in the left-hand side too.

Example 3.4 *Following Example 3.3, ε is the unique functional position of the term $r = h(A, y)$. We set q' of the definition to be equal to q_f. Thus* $\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f)$ *is of the form $\{A \to q?, h(q?, q??) \to q_f\}$. Since for r, the position 1 is a functional position and 2 is in $\mathcal{P}os_\mathcal{X}(r)$, we use the last line of the definition to compute $q??$, and $q?$ is defined by the approximation function γ_{exe}. Finally, we obtain:*

$$\begin{aligned}\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f) = \quad & \{r(1) \to \gamma_{\mathrm{exe}}(1), r(\varepsilon)(\gamma_{\mathrm{exe}}(1), \sigma_{\mathrm{exe}}(1)) \to q_f\} \\ & = \{A \to q_0, h(q_0, q_f) \to q_f\}.\end{aligned}$$

Lemma 3.1 *Let $\mathcal{A}$ be a finite tree automaton, γ an approximation function, $l \to r \in \mathcal{R}$, σ an $\mathcal{A}$-compatible $(l \to r)$-substitution, and q a state of $\mathcal{A}$. If $l\sigma \to^*_\mathcal{A} q$ then $r\sigma \to^*_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q$.*

The proof is obvious. The transitions in Norm_γ are precisely those added to reduce $r\sigma$ to q.

3.2 Over-Approximations for TRSs without Left-Linearity Constraint

This section is dedicated to the proof of the main result and explains how to build a regular over-approximation of $\mathcal{R}^*(\mathcal{A})$. The following definition presents the construction of a tree automaton $C_\gamma(\mathcal{A}_0)$ from the tree automaton $\mathcal{A}_0$, the approximation function γ and the TRS $\mathcal{R}$. This construction is usually named a completion step. Again, the only considered substitutions are $(l \to r)$-substitutions.

Definition 3.5 *Let $\mathcal{R}$ be a TRS. Let $\mathcal{A}_0 = (Q_0, \Delta_0, F_0)$ be a finite tree automaton and γ an approximation function for $\mathcal{A}_0$. The automaton $C_\gamma(\mathcal{A}_0) = (Q_1, \Delta_1, F_1)$ is defined by:*

$$\Delta_1 = \Delta_0 \cup \bigcup \mathrm{Norm}_\gamma(l \to r, \sigma, q)$$

where the union involves all rules $l \to r \in \mathcal{R}$, all states $q \in Q_0$, all $\mathcal{A}_0$-compatible $(l \to r)$-substitutions σ such that $l\sigma \to^*_{\mathcal{A}_0} q$ and $r\sigma \not\to^*_{\mathcal{A}_0} q$,

$$F_1 = F_0 \quad \textit{and} \quad Q_1 = Q_0 \cup Q_2,$$

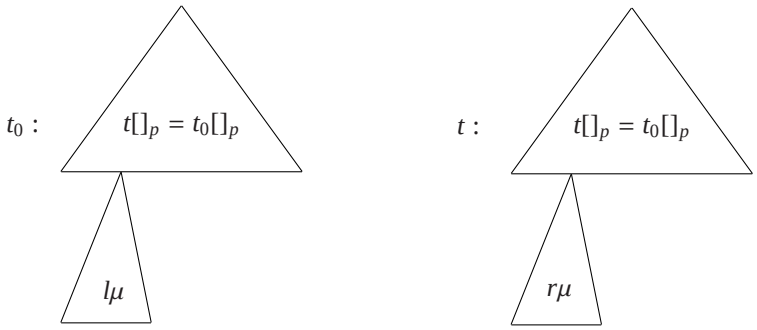
where Q_2 denotes the set of states occurring in left or right-hand sides of Δ_1 transitions.

The above lemma shows that a completion step computes an over-approximation of terms obtained by one rewriting step.

Lemma 3.2 *Let $\mathcal{A}_0 = (Q_0, \Delta_0, F_0)$ be a finite tree automaton and γ be an approximation function for $\mathcal{A}_0$. Let $\mathcal{R}$ be a TRS. One has $\mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(C_\gamma(\mathcal{A}_0))$.*

PROOF. Let $t \in \mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. By definition of $C_\gamma(\mathcal{A}_0)$ one has $\mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{L}(C_\gamma(\mathcal{A}_0))$. Consequently, if $t \in \mathcal{L}(\mathcal{A}_0)$ then one has $t \in \mathcal{L}(C_\gamma(\mathcal{A}_0))$. Thus we now assume that $t \in \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. Thus there exists a rule $l \to r \in \mathcal{R}$, a term t_0 in $\mathcal{L}(\mathcal{A}_0)$, a position p of t_0 and a substitution μ in $\Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ such that

$$t_{0|p} = l\mu \quad \text{and} \quad t = t_0[r\mu]_p. \tag{1}$$



Since $t_0 \in \mathcal{L}(\mathcal{A}_0)$, there exist a state $q \in Q_0$ and a state $q_f \in F_0$ such that

$$l\mu \to^*_{\mathcal{A}_0} q \quad \text{and} \quad t_0[q]_p \to^*_{\mathcal{A}_0} q_f. \tag{2}$$

Since $l\mu \to^*_{\mathcal{A}_0} q$ there exists an $(l \to r)$-substitution σ such that $l\mu \to^*_{\mathcal{A}_0} l\sigma$. Furthermore, for each $x \in \mathcal{V}\mathrm{ar}(l)$,

$$\mu(x) \in \bigcap_{p \in \mathcal{P}\mathrm{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)),$$

thus the $(l \to r)$-substitution σ is $\mathcal{A}_0$ compatible. Therefore, using Lemma 3.1 (by hypothesis, $l\sigma \to^*_{\mathcal{A}_0} q$), one has

$$r\sigma \to^*_{C_\gamma(\mathcal{A}_0)} q. \tag{3}$$

For each variable x occurring in l and all positions p of x in l, one has $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p)$. In particular, for each variable x occurring in l, $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p')$, where p' is the minimal position where x occurs in l. Consequently and by definition of $r\sigma$, one has

$$r\mu \to^*_{\mathcal{A}_0} r\sigma. \tag{4}$$

We are now able to conclude: using (1) one has $t = t_0[r\mu]_p$. Now, by (4) $t \to^*_{\mathcal{A}_0} t_0[r\sigma]_p$. To finish, using (3) and then (2) we obtain the following derivation: $t \to^*_{C_\gamma(\mathcal{A}_0)} t_0[q]_p \to^*_{\mathcal{A}_0} q_f$. Thus $t \in \mathcal{L}(C_\gamma(\mathcal{A}_0))$, proving the lemma. □

Let us remark that using well-chosen approximation functions may iteratively lead to a fixpoint automaton which recognises an over-approximation of $\mathcal{R}^*(\mathcal{A}_0)$. One can formally express this by the following (soundness) main theorem.

Theorem 3.1 *Let $(\mathcal{A}_n)$ and (γ_n) be respectively a sequence of finite tree automata and a sequence of approximation functions defined by: for each integer n, γ_n is an approximation function for $\mathcal{A}_n$ and $\mathcal{A}_{n+1} = \mathcal{C}_{\gamma_n}(\mathcal{A}_n)$. If there exists a positive integer N, such that for every $n \geq N$, $\mathcal{A}_n = \mathcal{A}_N$, then $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_N)$.*

The proof is by induction using Lemma 3.2.

3.3 Completion Example

In this section we explain how our approach works on an example in touch with the mathematical world.

We consider terms defined over $\mathcal{F}_0 = \{0\}$, $\mathcal{F}_1 = \{\mathrm{Opp}, s\}$, $\mathcal{F}_2 = \{+\}$ and $\mathcal{F}_{k\geq 3} = \emptyset$.

Here, the symbol s denotes the successor function. For instance, $s(s(s(0)))$ is the successor of the successor of the successor of 0 and denotes the integer 3. The operator Opp denotes the opposite value of an integer. For example, $\mathrm{Opp}(s(0))$ is the opposite value of the successor of 0 and denotes the integer -1. We use the following TRS to encode addition and subtraction over $\mathbb{Z}$. To simplify notations, we write $(x+y)$ or $x+y$ for $+(x,y)$.

$$
\begin{aligned}
\mathcal{R} = \{&\mathrm{Opp}(\mathrm{Opp}(x)) \to x && (5)\\
&x \to \mathrm{Opp}(\mathrm{Opp}(x)) && (6)\\
&x + \mathrm{Opp}(x) \to 0 && (7)\\
&x + y \to y + x && (8)\\
&x + (y + z) \to (x + y) + z && (9)\\
&x + 0 \to x && (10)\\
&x + s(0) \to s(x) && (11)\\
&s(x) \to x + s(0) && (12)\\
&\mathrm{Opp}(s(x)) \to \mathrm{Opp}(s(s(x))) + s(0)\} && (13)
\end{aligned}
$$

Notice that this TRS is not left-linear (Rule (7)). We are interested in the following problem: given three integers a, b and c, are there integers λ and μ such that $\lambda a + \mu b = c$? A basic number theory result states that the answer to the previous question is yes if and only if c is a multiple of the greatest common divisor of a and b.

For instance, it is possible for $a = 7$, $b = 3$ and $c = 15$ (since $\gcd(a,b) = 1$). We may prove it using the above TRS. Indeed, from $s^7(0)$ and $s^3(0)$ one can reach $s^{15}(0)$ using +, Opp and rewriting rules. For example, $s^3(0) \to^*_{12} s(0) + s(0)) + s(0)$. Consequently, $s^3(0) + s^3(0) \to^*_{(12),(9)} s^6(0)$. Similarly one has $(((s^7(0) + s^7(0)) + s^7(0)) \to^*_{(12),(9)} s^{21}(0)$. Moreover, $\mathrm{Opp}(s^3(0) + s^3(0)) \to^*_{(12),(9),(8)} \mathrm{Opp}(s^{21}(0)) + s^{15}(0)$. Therefore, $(((s^7(0) + s^7(0)) + s^7(0)) + \mathrm{Opp}(s^3(0) + s^3(0))$
$\to^*_{(8),(12),(9)} (s^{21}(0) + \mathrm{Opp}(s^{21}(0))) + s^{15}(0) \to^*_{(7),(10)} s^{15}(0)$.

Now we prove that the problem has no solution for $a = 2$, $b = 4$ and $c = 5$ (this is mathematically trivial, the goal is just to illustrate that it can be automatically proved using our over-approximation approach).

We consider for initial terms the language accepted by the following tree automaton $\mathcal{A}$ whose states are q_0, q_1, q_2, q_3, q_4, q_{-2} q_{-4} and q_f, whose final states are q_2,q_{-2}, q_{-4}, q_4, and q_f, and whose transitions are $0 \to q_0$, $s(q_0) \to q_1$, $s(q_1) \to q_2$, $s(q_2) \to q_3$, $s(q_3) \to q_4$ (encodes that $s^2(0)$ and $s^4(0)$ are initially known), $\mathrm{Opp}(q_4) \to q_{-4}$ (encodes that one can compute the opposite value of 4), $\mathrm{Opp}(q_2) \to q_{-2}$ (encodes that one can compute the opposite value of 2), $q_{f_1} + q_{f_2} \to q_f$ for all final states q_{f_1}, q_{f_2}, (encodes that one can do the addition of two computed integers terms). We want to prove that $s^5(0) \notin \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$. Some details on the first completion step are given below.

Rule (5) This rule provides no new transition. Indeed, there is no state q in $\mathcal{A}$ such that $\mathrm{Opp}(\mathrm{Opp}(q))$ can be reduced in $\mathcal{A}$ to a state.

Rule (6) For each state q one has to add the normalisation of the transition $\mathrm{Opp}(\mathrm{Opp}(q)) \to q$. Assume that

$$\gamma(\mathit{Rule}(6), \{\varepsilon \mapsto q_1\}, q_1)(1) = q_3.$$

Then during the completion step, the normalisation of $\mathrm{Opp}(\mathrm{Opp}(q_1)) \to q_1$ adds the transitions $\mathrm{Opp}(q_1) \to q_3$ and $\mathrm{Opp}(q_3) \to q_1$. With similar assumptions on γ, one adds during the first completion step $\mathrm{Opp}(q_0) \to q_0$, $\mathrm{Opp}(q_{-4} \to q_4)$ and $\mathrm{Opp}(q_{-2}) \to q_2$.

Rule (7) Since $q_4 + \mathrm{Opp}(q_4) \to^*_{\mathcal{A}} q_f$, one has to add the transition $0 \to q_f$ (we may easily verify this is the only compatible $l \to r$-substitution).

Rule (8-11) These rules don't provide new transitions.

Rule (12) Since $s(q_0) \to_{\mathcal{A}} q_1$ and $q_0 + s(0) \not\to^*_{\mathcal{A}} q_1$, one has to add the following transitions (with correct assumptions on γ) $0 \to q_0$, $s(q_0) \to q_1$ (these two transitions are already in $\mathcal{A}$) and $q_0 + q_1 \to q_1$. Similarly, one has to add transitions $q_0 + q_2 \to q_2$, $q_0 + q_3 \to q_3$, $q_0 + q_4 \to q_4$.

Rule (13) Since $\mathrm{Opp}(s(q_1)) \to^*_{\mathcal{A}} q_{-2}$ and $\mathrm{Opp}(s(s(q_1)) + s(0)) \not\to^*_{\mathcal{A}} q_{-2}$, one has to add the transitions (with correct assumption on γ), $s(0) \to q_1$, $s(q_1) \to q_2$, $s(q_2) \to q_3$, $\mathrm{Opp}(q_3) \to q_1$, $q_1 + q_1 \to q_2$ and $\mathrm{Opp}(q_2) \to q_{-2}$.

Similar completion steps lead to the following tree automaton $\mathcal{B}$:

- States of $\mathcal{B}$ are $q_{-4}, q_{-2}, q_1, q_2, q_3, q_4$ and q_f. Final states are q_2, q_4, q_{-2}, q_{-4} and q_f, transitions on constants are $0 \to q_0$ and $0 \to q_f$.
- Transitions with symbol s are given by the following table:

	q_0	q_1	q_2	q_3	q_4
s	q_1	q_2	q_3	q_4	q_1

  For instance, $s(q_2) \to q_3$ is a transition.

- Transitions with symbols Opp and +are given by the following tables:

	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
Opp	q_4	q_2	q_0	q_3	q_{-2}	q_1	q_{-4}	q_f

+	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
q_{-4}	q_{-4}, q_f	q_{-2}, q_f	q_{-4}	q_1	q_2, q_f	q_3	q_4, q_f q_0	q_f
q_{-2}	q_{-2}, q_f	q_0, q_f	q_{-2}, q_f	q_3	q_0, q_4, q_f	q_1	q_2, q_f	q_f
q_0	q_{-4}, q_f	q_{-2}, q_f	q_0	q_1	q_2, q_f	q_3	q_4, q_f	$\emptyset$
q_1	q_1	q_3	q_1	q_2, q_f	q_3	q_0, q_f q_4	q_f	$\emptyset$ $\emptyset$
q_2	q_2, q_f	q_4, q_f q_0	q_2, q_f	q_3	q_f, q_0 q_4	q_1	q_2, q_f	q_f q_f
q_3	q_3	q_1	q_3	q_4	q_1	q_2, q_f	q_3	$\emptyset$
				q_4				$\emptyset$
q_4	q_0, q_4 q_f	q_2, q_f	q_4, q_0 q_f	q_1	q_2, q_f	q_3	q_4, q_f q_0	q_f
q_f	q_f	q_f	$\emptyset$	$\emptyset$	q_f	$\emptyset$	q_f	q_f

The automaton $\mathcal{B}$ is stable by the C_γ completion. Consequently, it accepts an over-approximation of reachable terms of $\mathcal{A}$ by $\mathcal{R}$. Since $s^5(0) \notin \mathcal{L}(\mathcal{B})$, we may not have $\lambda.2 + \mu.4 = 5$ with $\lambda, \mu \in \mathbb{Z}$.

3.4 Under-Approximations for TRSs without Left-Linearity Constraint

The main idea (and problem) behind the under-approximations is that one wants the languages of computed tree automata to be in the set of terms reachable by rewriting. Having some conditions on the TRS allows us to prove that a term is actually reachable.

In order to obtain under-approximations, we do not want the completion procedure to introduce unreachable terms. Classically, we then work with injective approximation functions. We define here γ to be an injective approximation function from $\mathcal{R} \times (\mathbb{N}^* \mapsto Q) \times \mathbb{N}^* \times Q$ into Q. Theorem 3.2 shows that with such an approximation function, an under-approximation of the set of reachable terms is possible. Before, Lemma 3.3 presents an intermediary result useful for proving Theorem 3.2: this result reveals some features of terms recognised by $C_\gamma(\mathcal{A})$ for which there exists a rewriting predecessor recognised by $\mathcal{A}$. In the following, we introduce the notation $NLV(t)$ which for a term t of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, denotes the set of non-linear variables of t, i.e., the set of variables occurring at least twice within t.

Lemma 3.3 *Let $\mathcal{R}$ be a right-linear TRS for which $NLV(l) \cap \mathcal{V}\mathrm{ar}(r) = \emptyset$ for all $l \to r \in \mathcal{R}$. Let $\mathcal{A}$ be a tree automaton. There exists $t_0 \in \mathcal{T}(\mathcal{F})$ such that $t_0 \in \mathcal{L}(\mathcal{A}, q)$ and $t_0 \to_{\mathcal{R}} t$, if there exist a ground term t over $\mathcal{F}$, a state q of $\mathcal{A}$ and a function τ from $\mathcal{P}\mathrm{os}(t)$ to Q such that $t \in \mathcal{L}(C_\gamma(\mathcal{A}), q)$, $t \notin \mathcal{L}(\mathcal{A}, q)$ and τ satisfies the following conditions: (i) $\tau(\varepsilon) = q$; (ii) for all $p \in \mathcal{P}\mathrm{os}(t)$, $t_{|_p} \in \mathcal{L}(C_\gamma(\mathcal{A}), \tau(p))$ and, (iii) for all $p \in \mathcal{P}\mathrm{os}(t) \setminus \{\varepsilon\}$, if $\tau(p)$ is a state of $\mathcal{A}$, then $t_{|_p} \in \mathcal{L}(\mathcal{A}, \tau(p))$.*

Notice that the condition $NLV(l) \cap \mathcal{V}\mathrm{ar}(r) = \emptyset$ is a very strong condition that can be easily weakened. However, this theoretical restriction can be practically relevant too,

e.g. to model inverse operators. The example developed in Section 3.3 contains such a rule: see (7).

Proof. To simplify the notations we denote by Δ_1 the set of transitions of the automaton $C_\gamma(\mathcal{A})$, Δ_0 the set of transitions of $\mathcal{A}$, and Q_0 the set of states of $\mathcal{A}$.

The proof consists of 1) the construction of a term $s_1 \in \mathcal{T}(\mathcal{F}, Q)$ such that

$$t \to^*_{\Delta_1} s_1 \to_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q, \tag{14}$$

2) the construction, by iterating a backward process, of a term $s \in \mathcal{T}(\mathcal{F}, Q)$ such that

$$t \to^*_{\Delta_1} s \to^*_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q, \text{ and} \tag{15}$$

3) the proof that

$$t \to^*_{\Delta_0} r\sigma \to^*_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q. \tag{16}$$

First, using (ii) at the position ε gives $t_{|\varepsilon} \to^*_{\Delta_1} \tau(\varepsilon)$. Since $t = t_{|\varepsilon}$ and since $\tau(\varepsilon) = q$ (by (i)), one has $t \to^*_{\Delta_1} q$.

Since $t \in \mathcal{T}(\mathcal{F})$ one has $t \neq q$, and every derivation $t \to^*_{\Delta_1} q$ has the length one, at least. Consequently, there exists $s_1 \in \mathcal{T}(\mathcal{F}, Q)$ such that $t \to^*_{\Delta_1} s_1 \to_{\Delta_1} q$.

We now show by contradiction that the transition $s_1 \to q \notin \Delta_0$. Suppose that $s_1 \to q$ is a transition of Δ_0. Then $s_1 \in \mathcal{T}(\mathcal{F}, Q_0)$. Thus, using (*iii*), $t \to^*_{\Delta_0} s_1 \to_{\Delta_0} q$, a contradiction ($t \not\to^*_{\Delta_0} q$).

Therefore, the transition $s_1 \to q$ is in $\Delta_1 \setminus \Delta_0$. By definition of Δ_1 (see Definition 3.5), there exist q', $\sigma : \mathcal{P}\mathrm{os}_X(l)^* \mapsto Q$ and $l \to r \in \mathcal{R}$ such that $s_1 \to_{C_\gamma(\mathcal{A})} q \in \mathrm{Norm}_\gamma(l \to r, \sigma, q')$ and

$$l\sigma \to^*_{\Delta_0} q'. \tag{17}$$

Now by definitions of $\mathrm{Norm}_\gamma(l \to r, \sigma, q')$ and γ, each target state of a transition in $\mathrm{Norm}_\gamma(l \to r, \sigma, q')$ is either $Q \setminus Q_0$, or is equal to q'. Since $s_1 \to_{C_\gamma(\mathcal{A})} q \in \mathrm{Norm}_\gamma(l \to r, \sigma, q')$, either $q \in Q \setminus Q_0$, or $q = q'$. Because $q \in Q_0$, one has $q = q'$ and $t \to^*_{\Delta_1} s_1 \to_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q$.

We are done for (14). We now perform an iterative construction. If $s_1 \notin \mathcal{T}(\mathcal{F}, Q_0)$, then there exists a position p of s_1 such that $s_1(p) \in Q \setminus Q_0$. Thus $s_1(p)$ is of the form $s_1(p) = \gamma(l \to r, \sigma, q)(p)$. Since γ is injective, the only transition of Δ_1 leading to $s_1(p)$ is

$$r(p)(\gamma(l \to r, \sigma, q)(p.1), \ldots, \gamma(l \to r, \sigma, q)(p.\ell)) \to s_1(p).$$

Consequently, the derivation $t \to^*_{\Delta_1} s_1$ has to conclude by $t \to^*_{\Delta_1} s_2 \to s_1$ where

$$s_2 = s_1[r(p)(\gamma(l \to r, \sigma, q)(p.1), \ldots, \gamma(l \to r, \sigma, q)(p.\ell))]_p.$$

So, one has $t \to^*_{\Delta_1} s_2 \to_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} s_1 \to_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q$. Now, if $s_2 \notin \mathcal{T}(\mathcal{F}, Q_0)$, the same construction can be iteratively applied to s_2, and so on. Consequently, one can build a term $s \in \mathcal{T}(\mathcal{F}, Q_0)$ such that $\mathcal{P}\mathrm{os}(s) = \mathcal{P}\mathrm{os}(r)$ and

$$t \to^*_{\Delta_1} s \to^*_{\mathrm{Norm}_\gamma(l\to r,\sigma,q)} q, \tag{18}$$

and for each position p of s such that $s(p) \notin Q$,

$$s(p) = r(p). \tag{19}$$

We are done for (15) .

We can begin the last part of the proof. Let $q_1, \ldots, q_n$ be the states occurring in s while reading s from the left to the right. Let $p_1, \ldots, p_n$ be respectively the positions in s of states $q_1, \ldots, q_n$. Notice that the backward construction of s is deterministic. Indeed every derivation from t to q can be split up to

$$t \to^*_{\Delta_1} s \to^*_{\mathrm{Norm}_\gamma(l \to r,\sigma,q)} q.$$

It implies that for each q_i, with $i = 1, \ldots, n$, one has

$$q_i = \tau(p_i). \tag{20}$$

At this stage, s is of the form $r\sigma$ since γ is defined for every position of r.

Now using (20) and the hypothesis iii), one has

$$t \to_{\Delta_0} r\sigma \to^*_{\mathrm{Norm}_\gamma(l \to r,\sigma,q)} q.$$

The TRS $\mathcal{R}$ being right-linear with $NLV(l) \cap \mathcal{V}\mathrm{ar}(r) = \emptyset$ for each rule $l \to r$ of $\mathcal{R}$, one can built a substitution $\mu : \mathcal{P}\mathrm{os}_\chi(l) \mapsto \mathcal{T}(\mathcal{F})$ such that:

- For $p \in \mathcal{P}\mathrm{os}_{\mathcal{V}\mathrm{ar}(r)}(l)$, one can set $\mu(p) = t'$ and $t' = t|_{p'}$ with $p' \in \mathcal{P}\mathrm{os}_{\{l|_p\}}(r)$. Moreover, since $l|_p \notin NLV(l)$, one obtains $\mu(p) = t' \to^*_{\Delta_0} \sigma(p)$.
- For $p \in \mathcal{P}\mathrm{os}_{\mathcal{V}\mathrm{ar}(l)\setminus\mathcal{V}\mathrm{ar}(r)}(l)$, one can proceed in the following way:
  - if $l(p) \in NLV(l)$ then one can set $\mu(p'_1), \ldots, \mu(p'_1)$ to t' where $t' \in \mathcal{L}(\mathcal{A}, \sigma(p'_1)) \cap \ldots \cap \mathcal{L}(\mathcal{A}, \sigma(p'_n))$ with $\{p'_1, \ldots, p'_n\} = \mathcal{P}\mathrm{os}_{\{l(p)\}}(l)$.
  - Otherwise, one can set $\mu(p)$ to a term $t' \in \mathcal{L}(\mathcal{A}, \sigma(p))$.

By this way, there exists $t_0 = l\mu \in \mathcal{T}(\mathcal{F})$ such that $t_0 \to^*_{\mathcal{A}_0} q$ and $t_0 \to_{\mathcal{R}} t$, proving the lemma. □

The following result shows that each term of the language $C_\gamma(\mathcal{A}_0)$ is reachable by rewriting from $\mathcal{A}_0$ using $\mathcal{R}$.

Theorem 3.2 *Let $\mathcal{A}_0 = (Q_0, \Delta_0, F_0)$ be a finite tree automaton. Let $\mathcal{R}$ be a right-linear TRS. Given the approximation function γ defined at the beginning of Section 3.4, if for all $l \to r \in \mathcal{R}$, $\mathcal{V}\mathrm{ar}(r) \cap NLV(l) = \emptyset$ then $\mathcal{L}(C_\gamma(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.*

Proof. Let $\mathcal{P}_n$ be the following proposition:
For all $t \in \mathcal{L}(C_\gamma(\mathcal{A}_0))$, if there exists a function τ from $\mathcal{P}\mathrm{os}(t)$ to Q such that $\tau(\varepsilon) = q_f$ and for all $p \in \mathcal{P}\mathrm{os}(t)$,

$$t|_p \to^*_{C_\gamma(\mathcal{A}_0)} \tau(p) \quad \textit{and} \quad t[\tau(p)]_p \to^*_{C_\gamma(\mathcal{A}_0)} q_f$$

and such that $\quad |\{p \in \mathcal{P}os(t) \mid \tau(p) \in Q_0 \ \wedge \ t|_p \not\to^*_{\mathcal{A}_0} \tau(p)\}| = n,$

then $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.

We prove that $\mathcal{P}_n$ is true for all $n \geq 0$ by induction on n. To simplify notations, let

$$NR(t, \tau) = \{p \in \mathcal{P}os(t) \mid \tau(p) \in Q_0 \text{ and } t|_p \not\to^*_{\mathcal{A}_0} \tau(p)\}.$$

$\mathcal{P}_0$: Assume that t and τ satisfy the hypothesis on $\mathcal{P}_0$. We have $|NR(t, \tau)| = 0$. In particular, $\varepsilon \notin NR(t, \tau)$. So, $t = t|_\varepsilon \to_{\mathcal{A}_0} \tau(\varepsilon) = q_f$. Since $\mathcal{A}_0$ and $C_\gamma(\mathcal{A}_0)$ have the same set of final states, $t \in \mathcal{L}(\mathcal{A}_0)$.

$\mathcal{P}_n \implies \mathcal{P}_{n+1}$: Assume that $\mathcal{P}_n$ is true for $n \geq 0$ and that t and τ satisfy the hypothesis on $\mathcal{P}_{n+1}$. Since $NR(t, \tau)$ is non-empty, let p be a maximal element of $NR(t, \tau)$ (for the lexicographical order). Then, by maximality of p, one can apply Lemma 3.3 to $t_{|p}$. Thus, there exists $t_0 \in \mathcal{T}(\mathcal{F})$ such that $t_0 \to^*_{\mathcal{A}_0} \tau(p)$ and $t_0 \to_{\mathcal{R}} t_p$. Therefore, there exists a function τ_1 from $\mathcal{P}os(t_0)$ into Q_0 such that for all p', $t_0 \to^*_{\mathcal{A}_0} \tau_1(p')$, $t[\tau_1(p')]_{p'} \to^*_{C_\gamma(\mathcal{A}_0)} \tau(p)$. We define the function τ_2 from $\mathcal{P}os(t[t_0]_p)$ to Q as follows.

- If p is not a prefix of p', then $\tau_2(p') = \tau(p')$,
- Otherwise, if p' is of the form $p.u$, then $\tau_2(p') = \tau_1(u)$.

By construction, $t[t_0]_p \to_{\mathcal{R}} t$ and $|NR(t[t_0]_p, \tau_2)| = n - 1$. Thus, by induction, $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.

It follows that $\mathcal{P}_n$ is true for all $n \geq 0$, proving the theorem. □

Let $C^{(n)}_\gamma(\mathcal{A}_0)$ be the tree automaton obtained after n completion steps performed from $\mathcal{A}_0$ by using the TRS $\mathcal{R}$ and the approximation function γ. Finally, Proposition 3.1 shows that the approximation function γ provides a sound under-approximation of reachable terms.

Proposition 3.1 *If $\mathcal{R}$ is right-linear and for all $l \to r \in \mathcal{R}$, $NLV(l) \cap \mathcal{V}\text{ar}(r) = \emptyset$ then for all $n \leq 0$, $\mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$, $\mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{L}(C^{(n+1)}_\gamma(\mathcal{A}_0))$ and $\bigcup_{n\geq 0} \mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.*

Proof. By definition $C^{(n+1)}_\gamma(\mathcal{A}_0) = g_\gamma(C^{(n)}_\gamma(\mathcal{A}_0)))$. Consequently, the set of transitions of $C^{(n)}_\gamma(\mathcal{A}_0)$ is included in the transition set of $C^{(n+1)}_\gamma(\mathcal{A}_0)$. Thus $\mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{L}(C^{(n+1)}_\gamma(\mathcal{A}_0))$.

Now, using Lemma 3.2, one has for all $n \geq 1$:

$$\mathcal{R}(\mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0))) \subseteq \mathcal{L}(C^{(n+1)}_\gamma(\mathcal{A}_0)).$$

Consequently, by a direct induction, $\mathcal{R}^{\leq n}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(C^{(n+1)}_\gamma(\mathcal{A}_0))$. It implies that

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \bigcup_{n\geq 0} \mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)).$$

One can prove that for all $n \in \mathbb{N}$, $\mathcal{L}(C^{(n)}_\gamma(\mathcal{A}_0)) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ by direct induction on n using Theorem 3.2, and we are done. □

3.5 Practical Issues

3.5.1 Approximations for Ensuring Safety

Thanks to the above theoretical contributions, at this point, we have means to compute over-approximations and under-approximations of reachable terms. From a system verification point of view, by representing the set of initial configurations of a given system by a tree automaton $\mathcal{A}_0$, and by encoding its evolution by a TRS $\mathcal{R}$, $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ stands for the set of actually reachable configurations. Given a set of bad configurations encoded by a tree automaton $\mathcal{A}_{Bad}$, in order to verify a safety property, it is enough to decide whether the intersection between $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ and $\mathcal{L}(\mathcal{A}_{bad})$ is empty. So, as shown in Fig. 1, under-approximations are useful to show that there is one bad configuration reachable, at least. And computing over-approximations is useful to show that no bad configuration is reachable.

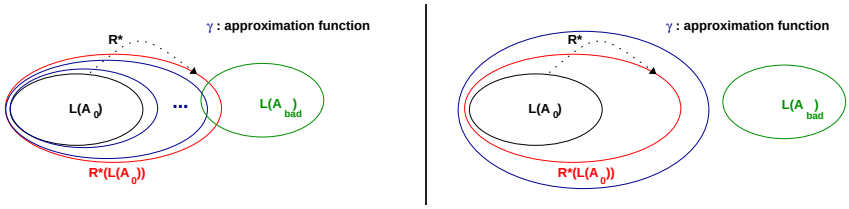


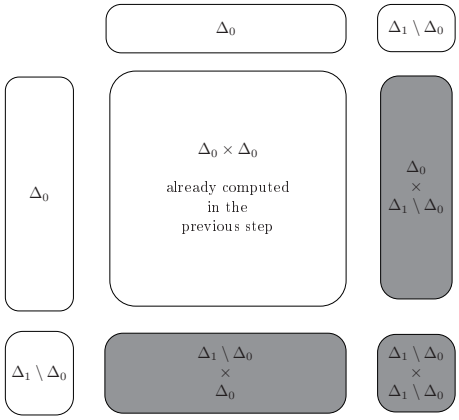
Figure 1: System verification using approximations.

3.5.2 Application to Protocols with Algebraic Properties

The completion procedure in Sect. 3.2 fits with all non left-linear TRSs. However, the user may be interested in developing algorithms to efficiently handle the completion for a particular class of non-linear rewrite rules. In the security protocol analysis framework, the problematic non left-linear rules usually concern the decoding abilities of an intruder and the algebraic properties of some cryptographic primitives. A particularity of such rules is that they are quadratic, i.e. , rules where a variable can occur at most twice within the left-hand side of the rule; let mention $x \oplus x \to 0$ for example. For this application field, this section gives an algorithm to efficiently handle the completion on TRSs with quadratic rules, called quadratic completion. This algorithm is then used for the experiments described in Section 4.1.

Recall that each completion step requires the computation of $(l \to r)$-substitutions compatible with the current tree automaton. In Example 3.2, the rule in $\mathcal{R}_{\text{exe}}$ is quadratic, and the substitution σ_{exe} is $\mathcal{A}_{\text{exe}}$-compatible because $\mathcal{L}(\mathcal{A}_{\text{exe}}, \sigma_{\text{exe}}(1)) \cap \mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(2.2)) \neq \emptyset$. This last computation can be done thanks to the square of $\mathcal{A}_{\text{exe}}$ by establishing the non emptiness of $\mathcal{L}(\mathcal{A}_{\text{exe}} \times \mathcal{A}_{\text{exe}}, \langle \sigma_{\text{exe}}(1), \sigma_{\text{exe}}(2.2) \rangle)$.

Definition 3.6 *Let $\mathcal{A} = (Q, \Delta, F)$ be a tree automaton. The square of $\mathcal{A}$, denoted $\mathcal{A}^2$,*



is the automaton $= (Q \times Q, \Delta', F \times F\rangle$ *where:*

$$\Delta' = \{f(\langle q_1,q_1'\rangle, \ldots, \langle q_n, q_n'\rangle) \to \langle q, q'\rangle \mid f(q_1, \ldots, q_n) \to q \in \Delta \wedge f(q_1', \ldots, q_n') \to q' \in \Delta\}.$$

Roughly speaking, for TRSs specifying protocols with quadratic rules, the square of a tree automaton – the product of a tree automaton with itself – can be computed by using the square of its predecessor. Computing the square of an automaton allows us to know whether there is a common datum between two states q and q' of the built automaton. The quadratic rules are then linearised, and the values taken by the linearised variables are checked on-the-fly. For example, if variable x occurs twice in a rule, one of occurrences is replaced by a fresh variable y in this rule the left-hand side. Then, this rule can be fired if the states q and q' – taken as values by resp. x and y – share at least a term.

More formally, let $\mathcal{A} = (Q, \Delta, F)$ be a tree automaton. Let $\mathcal{A}^2 = (Q_{\mathcal{A}^2}, \Delta_{\mathcal{A}^2}, F_{\mathcal{A}^2})$ be the square of the current tree automaton $\mathcal{A}$ according to Def. 3.6. The square of the tree automaton $C_\gamma(\mathcal{A})$ can be computed in the following way: $(C_\gamma(\mathcal{A}))^2 = (Q_{\mathcal{A}^2} \cup (Q \times (Q' \setminus Q)) \cup ((Q' \setminus Q) \times Q) \cup ((Q' \setminus Q) \times (Q' \setminus Q)), \Delta_{\mathcal{A}^2} \cup (\Delta \times (\Delta' \setminus \Delta)) \cup ((\Delta' \setminus \Delta) \times \Delta) \cup ((\Delta' \setminus \Delta) \times (\Delta' \setminus \Delta)), F_{\mathcal{A}^2})$. Note that the square of $C_\gamma(\mathcal{A})$ is based on the square of $\mathcal{A}$.

To decide the firing of a rule, an efficient state-of-the-art algorithm for the emptiness decision (see [CDG+02] for example) and an adapted data structure updated on-the-fly are used. Doing so, one can decide in a very efficient way whether the language of the squared automaton recognised by $\{(q, q')\}$ is not empty. In that case, the rule is fired with the computed substitution.

Thanks to these new features, a large number of protocols has been successfully validated: `NSPK-xor`, `View-only`, and also the Encrypted Key Exchange protocol (`EKE2`) using the exponential operator. All of these improvements are carried out in the next section.

4 Automatic Approximations and Applications for Verifying Cryptographic Protocols

Modelling security protocols by tree automata and term rewriting systems is basic, see[GK00] for instance. On the one hand, the initial knowledge and synthesis abilities of the intruder are encoded by a tree automaton $\mathcal{A}_0$. On the other hand, a term rewriting system $\mathcal{R}$ encodes protocol steps and intruder's analysis abilities. Our main purpose being to automate the protocol analysis in so far as possible, approximations should be generated automatically. The main ideas behind such a fully automatic generation are given in Section 4.1. We then give in Section 4.2 new experimental results obtained with the new version of the `TA4SP`tool.

4.1 Automatic Generation of Approximations

Notice that a safe and sound abstraction with only two agents [CLC03] is considered.

For each rule $l \to r$, each $(l \to r)$-substitution σ, each state q and each position p one has to define $\gamma(l \to r, \sigma, q)(p)$. Note that the approximation function is build in order to use finitely many nonces (*number used once*).

Next $\mathcal{R}$ is divided into two parts: $\mathcal{R}_1$ that encodes protocol steps and $\mathcal{R}_2$ that encodes intruder's abilities. For rules in $\mathcal{R}_2$, $\gamma(l \to r, \sigma, q)(p)$ is independent of both σ and q. Moreover, for every pair of rules $l_1 \to r_1$ and $l_2 \to r_2$ and positions of p_1 of r_1 and p_2 of r_2, one has $\gamma(l_1 \to r_1)(p_1) \neq \gamma(l_2 \to r_2)(p_2)$.

For rules in $\mathcal{R}_1$, $\gamma(l \to r, \sigma, q)(p)$ does not depend on q but only on $l \to r$, p and the value of σ on a finite set of variables $\{x_1, \ldots x_n\}$, representing agents names. The intuition is that one can ensure (by automaton properties) that if $l\sigma \to^*_{\mathcal{A}} q$ then $\sigma(x_i)$ may have a bounded number of values. Furthermore, the injectiveness like for $\mathcal{R}_2$ rules is required.

Since there are finitely many rules with finitely many positions, the number of states introduced during completion steps is bounded. Consequently, the completion procedure always stops with that approximation function and then computes an over-approximation. Concerning the under-approximations, roughly speaking new states are introduced each time it is necessary. We refer the interested reader to [BHK05] for more details.

4.2 Experimental Results

This section reports on new experimental results obtained when using the new version of the `TA4SP` tool [1].

The fully automatic `TA4SP` [BHK05] tool has been plugged into the high level protocol specification language `HLPSL`. We have thoroughly assessed the `TA4SP` tool by running it against some IETF[2] protocols of the AVISPA Library[3] and others from the

[1] `http://www.loria.fr/~boichut/ta4sp.html`.

[2] Internet Engineering Task Force

[3] Available at `http://www.avispa-project.org/`.

Protocol	Computation time(s) (seconds)	Diagnostic
NSPKL	4.12	SAFE
NSPK	10.26	RMU
NSSK	266.88	SAFE
NSPK-XOR	1803.97	RMU
Denning-Sacco sh. key	24.98	SAFE
Yahalom	874.35	SAFE
Andrew Secure RPC	212.01	SAFE
Wide Mouthed Frog	30.45	SAFE
Kaochow v1	227.30	SAFE
Kaochow v2	153.00	SAFE
TMN	109.08	RMU
AAA Mobile IP	1115.00	SAFE
UMTS-AKA	2.55	SAFE
CHAPv2	18.69	SAFE
CRAM-MD5	1.14	SAFE
DHCP-Delayed-Auth	1.05	SAFE
EKE	11.76	SAFE
EKE2	1541.43	SAFE
LPD-IMSR	12.24	SAFE
LPD-MSR	6.52	RMU
h.530-fix	54687.67	??
TSIG	1140.38	SAFE
SHARE	50.41	SAFE
View-only-untyped	18444.57	SAFE

Figure 2: Experiments on some secrecy properties using TA4SP

Clark and Jacob library [CJ97]. The experimental results are reported in Fig. 4.2, below. The diagnostic *SAFE* means that all secrecy properties have been verified for an unbounded number of executions of the initial HLPSL scenario. A contrario, the diagnostic *RMU* – Rewriting Model is Unsafe – relates that there exists an attack against one of the secrecy properties in our unbounded rewriting model. The diagnostic *??* means that no conclusion can be drawn.

Using implemented under-approximations presented in Sect. 3.4, four protocols (NSPK, NSPK-XOR, TMN and LPD-MSR) given in Fig. 4.2 have been diagnosed as flawed and the attack traces have, indeed, been built with other tools of the AVISPA platform. Nineteen protocols, specified in HLPSL, have been shown secure using over-approximations as in Sect. 4.1. Notice that we have successfully applied TA4SP not only to well-known protocols like NSPKL, SHARE, LPD-IMSR, from the Clark and Jacob library, but also to large-scale IETF protocols as DHCP-Delayed -Auth, CRAM-MD5, CHAPv2, TSIG, AAA Mobile IP, etc.

The protocols EKE2, h.530-fix and View-only-untyped use cryptographic op-

erators with algebraic properties: `exp` and `xor`. The protocols `View-only- untyped` and `EKE2` have been successfully analysed despite the computation time for the latter. Not so good computation time is better than an inconclusive result, as for `h.530-fix`. Note that the on-the-fly computation (OFC) in Sect. 3.5.2 has allowed us to check the protocol `View-only-untyped` when a naive approach was result-less as shown below.

Completion step	0	1	2	3	4	5	6	7	8
OFC Time (in s)	0.41	0.59	0.97	2.84	4.55	25	1709	7343	8956
Time (in s)	0.25	0.33	0.79	1.67	6.02	73.60	49.55	>18000	×

Completion step	9	10	Total
OFC Time (in s)	363.12	36.38	18444.57
Time (in s)	×	×	×

Before concluding this section, let us focus on the interesting but inconclusive result concerning the protocol `h.530-fix`. In [BMV03], the authors have detected an unknown attack against the protocol `h.530` using OFMC (On-the-Fly Model-Checker). They have then proposed a new version of this protocol: `h.530-fix`. OFMC has shown this protocol secure for the given scenario. So it is quite challenging to show that this protocol is indeed secure now for an unbounded number of sessions. We have tried to check the new version but, its analysis leads to an inconclusive result. However, it would be interesting to define finer approximations in order to show the safety of this protocol. We plan also to investigate in this direction – notably using a trace reconstruction technique we have developed in [BG06].

One last word about the computation times of Fig. 4.2: they are indeed not as good as we wished, but we are developing a promising new engine for the completion which gives impressive results. For example, a computation taking four days long has been reduced to forty-five minutes. This rewrite engine is still in progress, but we hope to integrate the next engine within `TA4SP` in the coming months.

5 Conclusion

This paper presents an essential improvement of [FGVTT04] showing how to extend that work to any kind of TRSs. Moreover, we also explained how to automate this approach in a suitable way. In this context we have provided a new tree regular model-checking technique. We also exposed how to use the technique for analysing security protocols, showing that the approach is not a purely domain-theoretic framework. We want to emphasise the fact that all the algorithms have been implemented, giving rise to the new version of the `TA4SP` tool.

The construction presented in this paper has several interesting features. The automatic generation of approximation functions for security protocols is quite intuitive. Indeed, it can be summarised to the following intuition: "for such a session, normalise in such a way". But, one can wonder how to generate approximations automatically for more complex objects, as Java programs [BGJLR07]. It would also be interesting to investigate similar approaches for unranked-tree automata that are useful for XML documents analysis. Moreover, a central question arising out of our work is how to combine the approximation-based techniques with existing tree automata regular techniques approaches, in order to get benefit from both approaches.

References

[AAB99] P. Abdulla, A. Annichini, and A. Bouajjani. Algorithmic verification of lossy channel systems: An application to the bounded retransmission protocol. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 208–222, 1999.

[AB05] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. *Science of Computer Programming*, 58(1-2):3–27, 2005.

[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. Moedersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The Avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, CAV'05*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, UK, July 2005. Springer-Verlag.

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Computer Aided Verification, CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–322, 1998.

[AC05] A. Armando and L. Compagna. An optimized intruder model for SAT-based model-checking of security protocols. *Electronic Notes in Theoretical Computer Science*, 125(1):91–108, 2005.

[AG02] A. Aldini and R. Gorrieri. Security analysis of a probabilistic nonrepudiation protocol. In Holger Hermanns and Roberto Segala, editors, *Process Algebra and Probabilistic Methods : Performance Modeling and Verification*, volume 2399 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, July 25–26 2002.

[ALdR06] P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree regular model checking: A simulation-based approach. *Journal of Logic and Algebraic Programming*, 69(1-2):93–121, 2006.

[BAF08] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[BEL05] L. Bozga, C. Ene, and Y. Lakhnech. A symbolic decision procedure for cryptographic protocols with time stamps. *Journal of Logic and Algebraic Programming*, 65(1):1–35, 2005.

[BET03] A. Bouajjani, X. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science*, 14:551–582, 2003.

[BET05] A. Bouajjani, J. Esparza, and T. Touili. Reachability analysis of synchronized PA systems. *Electronique Notes in Theoretical Computer Science*, 138(3):153–178, 2005.

[BFL04] S. Bardin, A. Finkel, and J. Leroux. FASTer acceleration of counter automata in practice. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for Construction and Analysis of Systems, TACAS'04*, volume 2988 of *Lecture Notes in Computer Science*, pages 576–590, Barcelona, Spain, March 2004. Springer.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Computer Aided Verification, CAV'96*, volume 1102, pages 1–12. Lecture Notes in Computer Science, 1996.

[BG06] Y. Boichut and Th. Genet. Feasible trace reconstruction for rewriting approximations. In *Rewriting Techniques and Applications, RTA'06*, volume 4098 of *Lecture Notes in Computer Science*, pages 123–135. Springer-Verlag, 2006.

[BGJLR07] Y. Boichut, Th. Genet, Th. P. Jensen, and L. Le-Roux. Rewriting approximations for fast prototyping of static analyzers. In Franz Baader, editor, *Rewriting Techniques and Applications, RTA'07, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2007.

[BHK05] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Technical Report RR-5727, INRIA, 2005.

[BHK06] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Handling algebraic properties in automatic analysis of security protocols. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *International Colloquium on Theorical Aspects of Computing, ICTAC'06*, volume 4281 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2006.

[BHK07] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Tree automata for detecting attacks on protocols with algebraic cryptographic primitives. In *Verification of Infinite-State Systems, INFINITY'07*, pages 44–53, Lisboa, Portugal, September 2007. The final version will be published in EN in Theoretical Computer Science, Elsevier.

[BHRV06] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking. *Electronic Notes in Theoretical Computer Science*, 149(1):37–48, 2006.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Computer Security Foundations Workshop, CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[BLP03] L. Bozga, Y. Lakhnech, and M. Perin. HERMES: An automatic tool for verification of secrecy in security protocols. In *Computer Aided Verification, CAV'03*, volume 2725 of *Lecture Notes in Computer Science*, pages 219–222, 2003.

[BMT07] A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. *Information and Computation*, 205(2):199–224, 2007.

[BMV03] Basin, Modersheim, and Vigano. An on-the-fly model-checker for security protocol analysis. In *European Symposium on Research in Computer Security, ESORICS'03*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Lecture Notes in Computer Science, Springer-Verlag, 2003.

[BT02] A. Bouajjani and T. Touili. Extrapolating tree transformations. In Springer-Verlag, editor, *Computer Aided Verification, CAV'02 Copenhagen (Denmark)*, volume 2404 of *Lecture Notes in Computer Science*, pages 539–554, 2002.

[BW98] B. Boigelot and P. Wolper. Verifying systems with infinite but regular state spaces. In *Computer Aided Verification, CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, June 1998.

[CDG+02] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 2002. Available at *http://www.grappa.univ-lille3.fr/tata/*.

[CDGS91] J.-L. Coquidé, M. Dauchet, R. Gilleron, and Vágvölgyi S. Bottom-up tree pushdown automata and rewrite systems. In Ronald V. Book, editor, *Rewriting Techniques and Applications, RTA'91*, volume 488 of *Lecture Notes in Computer Science*, pages 287–298. Springer-Verlag, 1991.

[CDL06] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14:1–43, 2006.

[CF05] G. Cece and A. Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202:166–190, 2005.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature : Version 1.0., November 1997.

[CKRT05] Y. Chevalier, R. Kusters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. *Theoretical Computer Science*, 338:247–274, 2005.

[CL07] C.J.F. Cremers and P. Lafourcade. Comparing state spaces in security protocol analysis. In *Automated Verification of Critical Systems, AVoCS'07*, Electronic Notes in Theoretical Computer Science, pages 49–63, 2007.

[CLC03] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *European Symposium on Programming, ESOP'03*, volume 2618 of *Lecture Notes in Computer Science 2618*, pages 99–113. Springer-Verlag, 2003.

[CLC05] H. Comon-Lundh and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, February 2005.

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Logic in Computer Science (LICS'03)*, pages 271–280, Los Alamitos, CA, June 22–25 2003. IEEE Computer Society.

[CR05] Y. Chevalier and M. Rusinowitch. Combining intruder theories. In *International Colloquium on Automata, Languages and Programming, ICAPL'05*, volume 3580 of *Lecture Notes in Computer Science*, pages 639–651, 2005.

[Cre06] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Ph.D. dissertation, Eindhoven University of Technology, 2006.

[CT03] H. Comon-Lundh and R. Treinen. Easy intruder deductions. In Nachum Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 225–242. Springer, February 2003. Invited paper.

[Del06] Stéphanie Delaune. Easy intruder deduction problems with homomorphisms. *Information Processing Letters*, 97(6):213–218, 2006.

[DJ06] S. Delaune and F. Jacquemard. Decision procedures for the security of protocols with probabilistic encryption against offline dictionary attacks. *Journal on Automated Reasoning*, 36(1-2):85–124, 2006.

[DLMS99] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Formal Methods and Security Protocols, FMSP'99*, 1999.

[DLMS04] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004. Preliminary report under the title 'Undecidability of bounded security protocols', in Proc. FloC'99.

[DLS02] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *Journal of Logic and Algebraic Programming*, 52-53:109–127, 2002.

[DT90] Dauchet and Tison. The theory of ground rewrite systems is decidable. In *Logic in Computer Science, LICS'90*, 1990.

[EMM07] S. Escobar, C. Meadows, and J. Meseguer. Equational cryptographic reasoning in the maude-NRL protocol analyzer. *Electronic Notes in Theoretical Computer Science*, 171(4):23–36, 2007.

[FGVTT04] G. Feuillade, Th. Genet, and V. Viet-Triem-Tong. Reachability analysis over term rewriting systems. *Journal on Automated Reasoning*, 33(3-4):341–383, 2004.

[FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In Manindra Agrawal and Anil Seth, editors, *Fundations of Software Technology and Theoretical Computer Science FSTTCS'02*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156, Kanpur, India, December 2002. Springer.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In Faron Moller, editor, *Verification of Infinite State Systems, INFINITY'97*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 27–39, Bologna, Italy, July 1997. Elsevier Science Publishers.

[GK00] Th. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Conference on Automated Deduction, CADE'00*, volume 1831 of *Lecture Notes in Computer Science*, pages 271–290. Springer-Verlag, 2000.

[GRV05] J. Goubault-Larrecq, M. Roger, and K.N. Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, 2005.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informatica*, 24(1/2):157–174, 1995.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In *Rewriting Techniques and Applications, RTA'96*, volume 1103 of *Lecture Notes in Computer Sciences*, pages 362–376. Springer-Verlag, 1996.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *Tools and Algorithms for Construction and Analysis of Systems, TACAS'00*, volume 1785 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.

[KR05] S. Kremer and M. Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *European Symposium on Programming, ESOP'05*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, 2005.

[KW04] R. Küsters and Th. Wilke. Automata-based analysis of recursive cryptographic protocols. In *Symposium on Theoretical Aspects of Computer Science, STACS'04*, volume 2996 of *Lecture Notes in Computer Science*, pages 382–393. Springer, 2004.

[LJ07] T. LeGall and B. Jeannet. Lattice automata: A representation for languages on infinite alphabets, and some applications to verification. In *Static Analysis Symposium, SAS'07*, volume 4634 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2007.

[LLT07] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for the equational theory of abelian groups with distributive encryption. *Information and Computation*, 205(4):581–623, 2007.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur. In *Proceedings of the 1997 Conference on Security and Privacy (S&P-97)*, pages 141–153, Los Alamitos, May 4–7 1997. IEEE Press.

[Möd07] S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD thesis, PhD Thesis, ETH Zürich, Information Security Group, Haldeneggsteig 4, CH-8092 Zürich, 2007.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Static Analysis Symposium, SAS'99*, pages 149–163, 1999.

[NAN05] C. R. Nielsen, E. H. Andersen, and H. R. Nielson. Static validation of a voting protocol. In *Automated Reasoning for Security Protocol Analysis, ARSPA 2005*, volume 135 of *Electronic Notes in Theoretical Computer Science*, pages 115–134. Elsevier, July 2005.

[NH06] S. Nanz and Ch. Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

[OT05] H. Ohsaki and T. Takai. ACTAS: A system design for associative and commutative tree automata theory. *Electronic Notes in Theoretical Computer Science*, 124(1):97–111, 2005.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Computer Aided Verification, CAV'00*, pages 328–343, 2000.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Computer Security Foundations Workshop CSFW '01*, pages 174–190. IEEE, 2001.

[RV02] P. Réty and J. Vuotto. Regular sets of descendants by some rewrite strategies. In *Rewriting Techniques and Applications, RTA'02*, volume 2378 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2002.

[Sal88] K. Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *Journal of Computer and System Sciences*, 37(3):367–394, 1988.

[Son99] Dawn Xiaodong Song. Athena: A new efficient automatic checker for security protocol analysis. In *Computer Security Fondations Workshop, CSFW'99*, pages 192–202, 1999.

[Tru05] T. Truderung. Regular protocols and attacks with regular knowledge. In *Conference on Automated Deduction, CADE'05*, volume 3632 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2005.

[ZD06] R. Zunino and P. Degano. Handling exp, $\times$ (and timestamps) in protocol analysis. In *Foundations of Software Science and Computation Structures, FoSSaCS'06*, volume 2987, pages 413–427, 2006.