# Feasible Trace Reconstruction for Rewriting Approximations

Yohan Boichut[1] and Thomas Genet[2]

[1] LIFC / Université de Franche-Comté
16, route de Gray
F-25030 Besançon cedex
boichut@lifc.univ-fcomte.fr

[2] IRISA / Université de Rennes 1
Campus de Beaulieu
F-35042 Rennes Cedex
genet@irisa.fr

**Abstract.** Term Rewriting Systems are now commonly used as a modeling language for programs or systems. On those rewriting based models, reachability analysis, i.e. proving or disproving that a given term is reachable from a set of input terms, provides an efficient verification technique. For disproving reachability (i.e. proving non reachability of a term) on non terminating and non confluent rewriting models, Knuth-Bendix completion and other usual rewriting techniques do not apply. Using the tree automaton completion technique, it has been shown that the non reachability of a term $t$ can be shown by computing an over-approximation of the set of reachable terms and prove that $t$ is not in the approximation. However, when the term $t$ is in the approximation, nothing can be said. In this paper, we refine this approach and propose a method taking advantage of the approximation to compute a rewriting path to the reachable term when it exists, i.e. produce a counter example. The algorithm has been prototyped in the Timbuk tool. We present some experiments with this prototype showing the interest of such an approach w.r.t. verification of rewriting models.

## 1 Introduction

In the rewriting theory, the reachability problem is the following: given a term rewriting system (TRS) $\mathcal{R}$ and two terms $s$ and $t$, can we decide whether $s \rightarrow_{\mathcal{R}}^{*} t$ or not? This problem, which can easily be solved on strongly terminating TRS (by rewriting $s$ into all its possible reduced forms and compare them to $t$), is undecidable on non terminating TRS. There exists several syntactic classes of TRSs for which this problem becomes decidable: some are surveyed in [FGVTT04], more recent ones are [GV98,TKS00]. In general, the decision procedures for those classes compute a finite tree automaton recognizing the possibly infinite set of terms reachable from a set $E \subseteq \mathcal{T}(\mathcal{F})$ of initial term, by $\mathcal{R}$, denoted by $\mathcal{R}^*(E)$. Then, provided that $s \in E$, those procedures check whether $t \in \mathcal{R}^*(E)$ or not. On the other hand, outside of those decidable classes, one can prove $s \not\to_{\mathcal{R}}^* t$ using over-approximations of $\mathcal{R}^*(E)$ [Jac96,Gen98,FGVTT04] and proving that $t$ does not belong to this approximation.

Recently, reachability analysis turned out to be a very efficient verification technique for proving properties on infinite systems modeled by TRS. Some of the most successful experiments, using proofs of $s \not\to_{\mathcal{R}}^* t$, were done on cryptographic protocols [Mon99,GK00,OCKS03,GTTVTT03,BHK05] where protocols and intruders are described using a TRS $\mathcal{R}$, $E$ represents the set of initial configurations of the protocol and $t$ a possible flaw. Then reachability analysis can detect the flaw (if $s \to_{\mathcal{R}}^* t$) or prove its absence (if $\forall s \in E : s \not\to_{\mathcal{R}}^* t$). However, the main drawback of those techniques based on tree automata, is that if $t \in \mathcal{R}^*(E)$ then we have the proof but not the rewriting path (also denoted by *trace* in the following). Indeed, from the tree automaton recognizing $\mathcal{R}^*(E)$ it is not possible to reconstruct the rewrite path from a possible $s$ to $t$ (i.e. the attack leading to a flaw in the context of cryptographic protocols). On the other hand, when dealing with an over-approximation $App \supseteq \mathcal{R}^*(E)$ if $t \in App$ then there is no way to check whether $t \in \mathcal{R}^*(E)$ ($t$ is really reachable from $s$) or if $t \in App \setminus \mathcal{R}^*(E)$ ($t$ is an artefact of the approximation). This problem becomes crucial when using approximations to prove security and safety properties. In that case, producing counter examples on a faulty specification makes the user more confident with the tool when it finally claims that the property is proven on a fixed version of the specification.

This paper tackles those two problems and proposes a solution that automatically gives the rewrite path when $\mathcal{R}^*(E)$ is constructed exactly and helps to discriminate between terms of $\mathcal{R}^*(E)$ and terms of the approximation when $\mathcal{R}^*(E)$ is over-approximated.

This paper is organized as follows. In section 2, we give the basic definitions for TRS and tree automata. In section 3, we recall the tree automata completion technique. In section 4, we define the trace reconstruction method we propose. Finally, in section 5, we present some experimentations done with our prototype implemented within Timbuk [GVTT00] a tree automata completion tool.

## 2 Preliminaries

Comprehensive surveys for TRSs and tree automata can be found respectively in [BN98] and in [CDG$^+$02].

Let $\mathcal{F}$ be a finite set of symbols, each one with an arity and let $\mathcal{X}$ be a countable set of variables. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the set of terms, and

$\mathcal{T}(\mathcal{F})$ denotes the set of ground terms (terms without variables). The set of variables of a term $t$ is denoted by $\mathcal{V}\mathrm{ar}(t)$. A substitution is a function $\sigma$ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F},\mathcal{X})$, which can uniquely be extended to an endomorphism of $\mathcal{T}(\mathcal{F},\mathcal{X})$. A position $p$ for a term $t$ is a word over $\mathbb{N}$. The empty sequence $\epsilon$ denotes the top-most position. The set $\mathcal{P}\mathrm{os}(t)$ of positions of a term $t$ is inductively defined by:

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

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

A TRS $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}\mathrm{ar}(l) \supseteq \mathcal{V}\mathrm{ar}(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if each variable of $l$ (resp. $r$) occurs only once in $l$ (resp. in $r$). A rule is linear if it is both left and right-linear. A TRS $\mathcal{R}$ is linear (resp. left-linear, right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is linear (resp. left-linear, right-linear). The TRS $\mathcal{R}$ 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 $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to^{\star}_{\mathcal{R}} t\}$.

Let $\mathcal{Q}$ be an infinite 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 $c = f(q_1,\ldots,q_n)$, $f \in \mathcal{F}$, $Arity(f) = n$, and $q_1,\ldots,q_n \in \mathcal{Q}$.*

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

**Definition 2 (Bottom-up non-deterministic finite tree automaton).** *A bottom-up non-deterministic finite tree automaton (tree automaton for short) is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f \subseteq \mathcal{Q}$ and $\Delta$ 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 $\Delta$) is denoted by $\to_{\Delta}$. When $\Delta$ is clear from the context, $\to_{\Delta}$

will also be denoted by $\rightarrow_{\mathcal{A}}$. Similarly, by notation abuse, we will often note $q \in \mathcal{A}$ and $t \rightarrow q \in \mathcal{A}$ respectively for $q \in \mathcal{Q}$ and $t \rightarrow q \in \Delta$.

**Definition 3 (Recognized language).** *The tree language recognized by $\mathcal{A}$ in a state $q$ is $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow^{\star}_{\mathcal{A}} q\}$. 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. A state $q$ is a* dead state *if $\mathcal{L}(\mathcal{A}, q) = \emptyset$.*

*Example 1.* Let $\mathcal{A}$ be the tree automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ such that $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_0) \rightarrow q_0, g(q_1) \rightarrow q_0, g(q_2) \rightarrow q_2, a \rightarrow q_1\}$. In $\Delta$ transitions are normalized. A transition of the form $f(g(q_2)) \rightarrow q_0$ 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 $\Delta$ in the following way: $g(a) \rightarrow_{\Delta} g(q_1) \rightarrow_{\Delta} q_0$. Note that $\mathcal{L}(\mathcal{A}, q_1) = \{a\}$ and $\mathcal{L}(\mathcal{A}, q_0) = \{f(g(a)), f(f(g(a))), \ldots\} = \{f^{\star}(g(a))\}$. Note also that $\mathcal{L}(\mathcal{A}, q_2) = \emptyset$ since no term of $\mathcal{T}(\mathcal{F})$ rewrites to $q_2$, hence $q_2$ is a dead state.

## 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 $\mathcal{A}_k$ such that $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ when it is possible (for the classes of TRSs covered by this algorithm see [FGVTT04]) and such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ otherwise.

The tree automata completion works as follows. From $\mathcal{A} = \mathcal{A}_0$ 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 $s \rightarrow_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$. If we find a fixpoint automaton $\mathcal{A}_k$ such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_k)) = \mathcal{L}(\mathcal{A}_k)$, then we have $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ (or $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ if $\mathcal{R}$ is not in one class of [FGVTT04]). To build $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, we achieve a *completion step* which consists in finding *critical pairs* between $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \rightarrow r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of $l$ such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \rightarrow^*_{\mathcal{A}_i} q$ and $l\sigma \rightarrow_{\mathcal{R}} r\sigma$. For every critical pair detected between $\mathcal{R}$ and $\mathcal{A}_i$ such that $r\sigma \not\rightarrow^*_{\mathcal{A}_i} q$, $\mathcal{A}_{i+1}$ is constructed by adding a new transition $r\sigma \rightarrow q$ to $\mathcal{A}_i$ such that $\mathcal{A}_{i+1}$ recognizes $r\sigma$ in $q$, i.e. $r\sigma \rightarrow_{\mathcal{A}_{i+1}} q$.

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "A_i"', "*"] & r\sigma \arrow[dl, dashed, bend left, "A_{i+1}", "*"'] \\
q &
\end{tikzcd}$$

However, the transition $r\sigma \rightarrow q$ is not necessarily a normalized transition of the form $f(q_1, \ldots, q_n) \rightarrow q$ and so it has to be normalized first. For example, to normalize 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 a set of normalized 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 normalized form does not make any difference. Now, assume that $q_1 = q_2$, the normalized 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 normalized transitions represents the regular set of non normalized transitions of the form $f(g^\star(a), h(q')) \rightarrow q$ which contains the transition we want to add but also many others. 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.

For every transition, there exists an equivalent set of normalized transitions. Normalization consists in decomposing a transition $s \rightarrow q$, into a set $\mathrm{Norm}(s \rightarrow q)$ of normalized transitions. The method consists in abstracting subterms $s'$ of $s$ s.t. $s' \notin \mathcal{Q}$ by states of $\mathcal{Q}$.

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

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

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

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

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

1. *if $s = q$, then $\mathrm{Norm}_\alpha(s \rightarrow q) = \emptyset$, and*
2. *if $s \in \mathcal{Q}$ and $s \neq q$, then $\mathrm{Norm}_\alpha(s \rightarrow q) = \{c \rightarrow q \mid c \rightarrow s \in \Delta\}$, and*
3. *if $s = f(t_1, \ldots, t_n)$, then $\mathrm{Norm}_\alpha(s \rightarrow q) =$*
   $\{f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \rightarrow q\} \cup \bigcup_{i=1}^n \mathrm{Norm}_\alpha(t_i \rightarrow top_\alpha(t_i))$.

*Example 2.* Let $\alpha$ an abstraction function such that: $\alpha = \{g(q_1, q_0) \mapsto q, b \mapsto q_1, a \mapsto q_2)\}$. Consequently, $top_\alpha = \{q_0 \mapsto q_0, b \mapsto q_1, a \mapsto q_2, g(b, q_0) \mapsto q\}$. The transition $f(a, g(b, q_0)) \rightarrow q$ can be normamized using $\mathrm{Norm}_\alpha$ in the follwing way :

$$\begin{aligned}\mathrm{Norm}_\alpha(f(a, g(b, q_0)) \to q) = \{&f(top_\alpha(a), top_\alpha(g(b, q_0))) \to q\} \\ &\cup \mathrm{Norm}_\alpha(a \to q_2) \cup \mathrm{Norm}_\alpha(g(b, q_0) \to q)\end{aligned}$$

By applying Definition 6 on $\mathrm{Norm}_\alpha(a \to q_2)$ and $\mathrm{Norm}_\alpha(g(b, q_0) \to q)$, we obtain that $\mathrm{Norm}_\alpha(f(a, g(b, q_0)) \to q) = \{f(q_2, q) \to q, a \to q_2, b \to q_1 g(q_1, q_0) \to q\}$.

With different abstraction function, on the same transition, one can obtain different normalizations. Hence, the precision of the fixpoint automaton $\mathcal{A}_k$ depends on the abstraction $\alpha$.

**Definition 7 (Automaton completion).** *Let $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$ be a tree automaton, $\mathcal{R}$ a TRS and $\alpha$ an abstraction function. The one step completed automaton $\mathcal{A}_{i+1}$ is a tree automaton $\langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_f, \Delta_{i+1} \rangle$ such that:*

$$\Delta_{i+1} = \Delta_i \cup \bigcup_{l \to r \in \mathcal{R},\, q \in \mathcal{Q},\, \sigma : \mathcal{X} \mapsto \mathcal{Q},\, l\sigma \to^*_{\Delta_i} q} \mathrm{Norm}_\alpha(r\sigma \to q)$$

$$\mathcal{Q}_{i+1} = \{q \mid c \to q \in \Delta_{i+1}\}$$

## 4 Reconstruction Method

To make the reading of this paper easier, we give once for all the notations used in the remainder of this paper:

- $\mathcal{R}$ is a left-linear TRS;
- $\alpha$ is a given abstraction function (following Definition 4);
- $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ is a finite sequence of automata obtained by the completion algorithm presented in Definition 7 for a given abstraction function $\alpha$;
- $\mathcal{A}_k$ is a fixpoint automaton obtained from $A_0$,$\mathcal{R}$ and $\alpha$;
- $\Delta_i$ is the set of transitions of the automaton $\mathcal{A}_i$;
- $\mathcal{Q}_f$ is the set of final states of automata $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$.

We suppose that for all $i = 0, \ldots, k$: $\Delta_i$ are a sets of normalized transitions. In particular all $\Delta_i$ do not contain epsilon transitions (see Definition 1).

**Definition 8 ($\Delta$−Unifier).** *Let $\Delta$ be a set of transitions, $t_1 \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $t_2 \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. A substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is a $\Delta$-unifier of $t_1$ and $t_2$ if and only if*

*1.* $t_1\sigma \rightarrow^*_\Delta t_2$, *and*
*2.* *for all* $x \in \mathcal{V}\mathrm{ar}(t_1)$*:*
   – *if* $\mathcal{P}\mathrm{os}_x(t_1) \cap \mathcal{P}\mathrm{os}_{\mathcal{F}}(t_2) \neq \emptyset$ *then* $\sigma(x) = t_2|_{p'}$ *for some* $p' \in \mathcal{P}\mathrm{os}_x(t_1) \cap \mathcal{P}\mathrm{os}_{\mathcal{F}}(t_2)$.
   – *otherwise,* $\sigma(x) = q$ *where* $q \in \mathcal{Q}$.

*We denote by* $\Uparrow_\Delta(t_1, t_2)$ *the set of* $\Delta-$*unifiers of* $t_1$ *and* $t_2$.

The example below illustrates that, in general, for two terms $t_1$ and $t_2$ there exist several $\Delta$-unifiers. The example also illustrates the case when $t_1$ is not linear. After the example, in Lemma 1, we show that for two terms $t_1$ and $t_2$ the set $\Uparrow_\Delta(t_1, t_2)$ is finite.

*Example 3.* Let $t_1 = f(g(x), h(y, y))$, $t_2 = f(q_1, h(g(a), g(q_2)))$ and a set of transitions $\Delta$ containing at least the following transitions $a \rightarrow q_2$, $g(q) \rightarrow q_1$, $g(q_1) \rightarrow q_1$ and $g(q_2) \rightarrow q_1$. The following two substitutions $\sigma_1 = \{x \mapsto q, y \mapsto g(a)\}$ and $\sigma_2 = \{x \mapsto g(q_1), y \mapsto g(a)\}$ are $\Delta$-unifiers of $t_1$ and $t_2$, but $\sigma_3 = \{x \mapsto q_1, y \mapsto g(q_2)\}$ is not since $g(q_2) \not\rightarrow^*_\Delta g(a)$ and thus $t_1\sigma_3 \not\rightarrow^*_\Delta t_2$.

**Lemma 1.** *For all* $\Delta$, $t_1$ *and* $t_2$, *the set* $\Uparrow_\Delta(t_1, t_2)$ *is finite.*

*Proof.* Given a term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, let us denote by $Sub(t)$ the set of all subterms of $t$, i.e. $Sub(t) = \{t|_p \mid p \in \mathcal{P}\mathrm{os}(t)\}$. By definition of $\Uparrow_\Delta(t_1, t_2)$, any substitution $\sigma$ of this set maps a variable either to a state or to a subterm of $t_1$. Hence $\Uparrow_\Delta(t_1, t_2) \subset (\mathcal{V}\mathrm{ar}(t_1) \mapsto (\mathcal{Q} \cup Sub(t_1))$. Since $\mathcal{V}\mathrm{ar}(t_1)$, $\mathcal{Q}$ and $Sub(t_1)$ are finite then so is $(\mathcal{V}\mathrm{ar}(t_1) \mapsto (\mathcal{Q} \cup Sub(t_1))$ and thus $\Uparrow_\Delta(t_1, t_2)$ is finite.

During the completion algorithm presented in Definition 7, critical pairs are detected. The role of *Occurency of a critical pair* (*OCCP*) is to store the information on every critical pair found during completion (the applied rule, the substitution, the position) so as to infer information on feasible traces afterwards.

**Definition 9 (Occurency of a Critical Pair($OCCP$)).** *Let* $\mathcal{A}_k$ *be a fixpoint tree automaton obtained from* $\mathcal{A}_0$ *using the completion algorithm of Definition 7 such that* $\mathcal{A}_k = \mathcal{A}_{k+1}$ *and* $k \geq 0$. *An OCCP is a triple* $\langle l \rightarrow r, \rho, q\rangle$ *where:*

- $l \rightarrow r$ *is a rewriting rule of* $\mathcal{R}$*;*
- $\rho$ *a substitution of variables in* $\mathcal{V}\mathrm{ar}(l)$ *by states in* $\mathcal{Q}$*;*
- $q \in \mathcal{Q}$*;*

– $l\rho \rightarrow^*_{\Delta_k} q$

*Let $OCCP_k$ be the set of all OCCP built on $\mathcal{A}_k$.*

Our method of trace reconstruction is based on data back-tracking on the automaton $\mathcal{A}_k = \langle \mathcal{F}, \mathcal{Q}_k, \mathcal{Q}_f, \Delta_k \rangle$ obtained by the completion of an automaton $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}_0, \mathcal{Q}_f, \Delta_0 \rangle$ by a TRS $\mathcal{R}$.

From $\mathcal{A}_k$, $OCCP_k$ and a term $t \in \mathcal{L}(\mathcal{A}_k)$, we need to find $t' \in \mathcal{L}(\mathcal{A}_k)$ a predecessor of $t$ i.e. such that $t' \rightarrow_{\mathcal{R}} t$. Then we search for a predecessor of $t'$ and so on until reaching a term $t_0 \in \mathcal{L}(\mathcal{A}_0)$. Before defining formally the set of predecessors of a term $t$ from $\mathcal{A}_k$ and $OCCP_k$, we define a particular substitution constructor.

**Definition 10.** $\sigma \bigsqcup \rho = \sigma \cup \{x \mapsto t \mid x \notin dom(\sigma) \ \wedge \ x \mapsto t \in \rho\}$

*Example 4.* Let $\sigma_1, \sigma_2 : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ be two substitutions such that:

– $\sigma_1 = \{x \rightarrow a, y \rightarrow q_1\}$ and
– $\sigma_2 = \{x \rightarrow q_2, y \rightarrow q_3, z \rightarrow b\}$.

Thus, $\sigma_1 \bigsqcup \sigma_2 = \{x \mapsto a, y \mapsto q_1, z \mapsto b\}$ and $\sigma_2 \bigsqcup \sigma_1 = \{x \mapsto q_2, y \mapsto q_3, z \mapsto b\} = \sigma_2$.

**Definition 11 (Pred).** *Let $cp = \langle l \rightarrow r, \rho, q \rangle$ be an OCCP such that $cp \in OCCP_k$. The set of predecessors of $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ w.r.t. $cp$ at position $p \in \mathcal{P}os(t)$ is defined by $Pred(t, cp, p) = \{t[l\sigma \bigsqcup \rho]_p \mid \sigma \in \Uparrow_{\Delta_k}(r, t|_p)$ and $r\sigma \rightarrow^*_{\Delta_k} r\rho\}$.*

*Example 5.* Let $\Delta$ be a set of transitions containing the transitions $g(q_2) \rightarrow q_3, a \rightarrow q_2, g(q_1) \rightarrow q_1, g(q_4) \rightarrow q_1$ and $l \rightarrow r = f(x, y) \rightarrow f(g(x), h(y, y))$ be a rewriting rule. Let $cp = \langle l \rightarrow r, q, \rho \rangle$ be an $OCCP$ where $\rho = \{x \mapsto q_1, y \mapsto q_3\}$ and $t = f(q_1, h(g(a), g(q_2)))$ be a term over $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. For the position $\epsilon$ and the term $t$, $Pred(t, cp, \epsilon) = \{f(q_1, g(a))\}$. Indeed, the only $\Delta-$unifier $\sigma \in \Uparrow_{\Delta}(r, t)$ respecting the condition $r\sigma \rightarrow^*_{\Delta} r\rho$ is $\sigma = \{x \mapsto g(q_1), y \mapsto g(a)\}$. Consequently, by applying the substitution $\sigma \bigsqcup \rho$ on $l$, we obtain $f(g(q_1), g(a))$.

Thus, by iterating this process on the terms obtained at each step, we are able to build some sequences of terms as shown in Definition 12.

**Definition 12 (Sequence).** *Let $t_0, \ldots, t_n \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$ such that $\forall i = 1 \ldots n : t_i \rightarrow^*_{\Delta_k} q$. Let $cp_1, \ldots, cp_n \in OCCP_k$ and $p_1, \ldots, p_n \in \mathbb{N}^*$ such that $\forall i = 1 \ldots n : cp_i = \langle l_i \rightarrow r_i, \rho_i, q_i \rangle$. If $\forall i = 1 \ldots n : t_{i-1} \in Pred(t_i, cp_i, p_i)$ then*

$$t_n \stackrel{cp_n,p_n}{\longleftarrow} t_{n-1} \ldots \stackrel{cp_1,p_1}{\longleftarrow} t_0.$$

*is a sequence from $t_n$ to $t_0$.*

The following theorem relates sequences to rewriting pathes. Roughly, if a sequence starts from $t$ and ends up on a term $t_0 \in \mathcal{L}(\mathcal{A}_0)$ then there exist a rewriting path from $t_0$ to $t$. Note that if $t_0 \notin \mathcal{T}(\mathcal{F})$ (but $t_0 \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$) then there exists a term $t_0' \in \mathcal{T}(\mathcal{F})$ and $t_0' \in \mathcal{L}(\mathcal{A}_0)$ such that $t_0' \rightarrow_{\mathcal{A}_0} t_0$ and $t_0' \rightarrow_{\mathcal{R}}^* t$.

**Theorem 1 (Correctness).** *Given $t \in \mathcal{L}(A_k)$ and a final state $q_f \in \mathcal{Q}_f$ such that $t \rightarrow_{\Delta_k}^* q_f$, if there exists a sequence $t_n \stackrel{cp_n,p_n}{\longleftarrow} t_{n-1} \ldots \stackrel{cp_1,p_1}{\longleftarrow} t_0$ such that $t = t_n$, $\forall i = 0 \ldots n : t_i \rightarrow_{\Delta}^* q_f$ and $t_0 \rightarrow_{\Delta_0}^* q_f$ then there exists a term*

$$s \in \mathcal{L}(\mathcal{A}_0),\ s \rightarrow_{\Delta_0}^* t_0 \text{ and } s \rightarrow_{\mathcal{R}}^* t.$$

*Moreover, there exists a rewrite path (or trace)*

$$s_n \stackrel{rl_n,p_n}{\longrightarrow} s_{n-1} \ldots s_1 \stackrel{rl_0,p_0}{\longrightarrow} t$$

*where $rl_i = l_i \rightarrow r_i$ is the rule of the OCCP $cp_i$, $s_n = s$, and $s_{i-1} = s_i[r_i\mu_i]_{p_i}$ and $l\mu_i = s_i|_{p_i}$.*

*Proof (Sketch).* The proof can be done by induction on the length of the sequence. See [BG06] for more details.

Not only to be correct, our method is also complete in the sense that if there exists a rewriting path between two terms then our method finds at least one path. However, because of Definition 8, only minimal pathes are constructed. Thus, between two terms $s$ and $t$ it is not possible to find all the possible traces but only minimal ones.

**Theorem 2 (Completeness).**
*Let $t, u \in \mathcal{L}(\mathcal{A}_k)$. If $u \rightarrow_{\mathcal{R}}^* t$ then there exists $t_0, \ldots, t_{n-1} \in \mathcal{L}(\mathcal{A}_k)$, $cp_1, \ldots, cp_n \in OCCP_k$, $p_1, \ldots p_n \in \mathbb{N}^*$ and a sequence $t \stackrel{cp_n,p_n}{\longleftarrow} t_{n-1} \ldots \stackrel{cp_1,p_1}{\longleftarrow} t_0$ such that $u \rightarrow_{\Delta}^* t_0$.*

*Proof (Sketch).* The proof is done by induction of the length of the rewrite derivation $u \rightarrow_{\mathcal{R}}^* t$. See [BG06] for details.

Thus, thanks to Theorem 2 and Theorem 1, we can define an algorithm that builds a valid sequence. For constructing a valid sequence, we start from a term $t$, construct the finite set of predecessors of $t$ for all positions of $t$ and all the computed $OCCP$s. Then, we repeat non deterministically the same operation on all the predecessors of $t$ until finding a term $t_0 \rightarrow^*_{\Delta_0} q_f$ where $q_f$ is a final state of $\mathcal{A}_0$. Of course, since we are dealing with infinite models, it is not always possible to conclude because trace reconstruction may diverge if it starts from a term that is not reachable.

## 5 Experimental Results

Timbuk[GVTT01,GVTT00] was used to prototype the method presented in Section 4. Timbuk is a collection of tools for achieving proofs of reachability over TRS and for manipulating tree automata (bottom-up nondeterministic finite tree automata).

In particular, Timbuk implements the tree automata completion algorithm presented in Section 3. To reconstruct a trace, the process is the following: Let $\mathcal{A}_0$, $\mathcal{R}$, $\alpha$ and $t$ be respectively a tree automaton, a TRS, an abstraction function and a term to find. Timbuk performs the completion of $\mathcal{A}_0$ by $\mathcal{R}$ using the abstraction function $\alpha$.

If the term $t$ is not recognized by the completed tree automaton, then $t$ is not reachable. Otherwise, either it is reachable or it is in the over-approximation part. To discriminate between the two solutions, we can use trace reconstruction. First, the set of $OCCP$ is computed. Then, given $t$ and a natural $N$, we search the tree of possible sequences for a predecessor of $t$ in $\mathcal{L}(\mathcal{A}_0)$, breadth-first and up to a depth $N$. Our system returns the first found trace which is one of the minimal traces.

Three different results can be obtained:

1. $t_0 \rightarrow_{\mathcal{R}} \ldots \rightarrow_{\mathcal{R}} t$: a trace is found and returned;
2. `Term of the approximation`: No trace can be provided because $t$ is in the approximation part. This can be shown when the tree of predecessors of $t$ is of depth $M$ such that $M \leq N$ and no term of the tree belongs to the initial set, i.e. $\mathcal{L}(\mathcal{A}_0)$.
3. `Cannot conclude`: The tree of predecessors of $t$ has been explored up to a depth $N$ (and depth of the tree is not bounded by $N$) without finding a term of the initial set.

Note that in practice, we also use a generalization of this: patterns of forbidden terms, $t_p \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, instead of a single term $t$. In those cases, the trace reconstruction process is very similar: we look for substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ such that $t_p\sigma$ is recognized by the over-approximation. Then, we can start reconstruction from $t_p\sigma$. If such a $\sigma$ exists then we know that there exists at least one substitution $\rho : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that $t_p\rho$ is recognized by the over-approximation (no dead states). Then we can start reconstruction from $t_p\rho$.

Now, let us present some experimentations on the verification of a simple two processes counting system. The following TRS describes the behavior of two processes each one equipped with an input list and a FIFO. Each process receives a list of symbols '+' and '−' to count, as an input. One of the processes, say $P_+$, is counting the '+' symbols and the other one, say $P_-$ is counting the '−' symbols. When $P_+$ receives a '+', it counts it and when it receives a '−', it adds the symbol to $P_-$'s FIFO. The behavior of $P_-$ is symmetric. When a process' input list and FIFO is empty then it stops and gives the value of its counter.

Here is a possible rewrite specification of this system, given in the Timbuk language, where `S(_,_,_,_)` represents a configuration with a process $P_+$, a process $P_-$, $P_+$'s FIFO and $P_-$'s FIFO. The term `Proc(_,_)` represents a process with an input list and a counter, `add(_,_)` implements adding of an element in a FIFO, and `cons`, `nil`, `s`, `o` are the usual constructors for lists and natural numbers.

```
Ops
        S:4 Proc:2 Stop:1 cons:2 nil:0 plus:0 minus:0 s:1 o:0 end:0 add:2
Vars    x y z u c m n
TRS R1
 add(x, nil) -> cons(x, nil)
 add(x, cons(y, z)) -> cons(y, add(x, z))
 S(Proc(cons(plus, y), c), z, m, n) -> S(Proc(y, s(c)), z, m, n)
 S(Proc(cons(minus, y), c), u, m, n) -> S(Proc(y, c), u, m, add(minus, n))
 S(x, Proc(cons(minus, y), c), m, n) -> S(x, Proc(y, s(c)), m, n)
 S(x, Proc(cons(plus, y), c), m, n) -> S(x, Proc(y, c), add(plus, m), n)
 S(Proc(x, c), z, cons(plus,m), n) -> S(Proc(x, s(c)), z, m, n)
 S(x, Proc(z, c), m, cons(minus,n)) -> S(x, Proc(z, s(c)), m ,n)
 S(Proc(nil, c), z, nil, n)  -> S(Stop(c), z, nil, n)
 S(x, Proc(nil, c), m, nil) -> S(x, Stop(c), m, nil)
```

On this specification, we aim at proving that, for any input lists, there is no possible deadlock. In this example, a deadlock is a configuration where a process has stopped but there are still symbols to count in its FIFO, i.e. terms of the form (pattern $t_p$): `S(Stop(x), z, cons(plus, u), c)`. The set of initial configurations of the system is described by the following tree automaton, where each process has a counter initialized to

0 and has an unbounded input list (with both '+' and '−') and with at least one symbol.

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

Let $\alpha_1$ be the (constant) abstraction function normalizing every configuration into a single state $q$, i.e. $\alpha_1 : \mathcal{T}(\mathcal{F} \cup \mathcal{Q}) \mapsto \{q\}$ and let $\alpha_2$ be the abstraction function normalizing every configuration into a new state. Roughly, $\alpha_1$ and $\alpha_2$ are respectively the worst and the better abstraction functions. Using completion, $\alpha_1$ (resp. $\alpha_2$) produces the most approximated (resp. precise) automaton. By running Timbuk on the previous specification, with $\alpha_1$, we can obtain within a few seconds a tree automaton over-approximating $\texttt{R1}^*(\mathcal{L}(\texttt{A1}))$. However, we cannot prove that the system is deadlock free. Indeed, when looking for patterns `S(Stop(x), z, cons(plus, u), c)` in the over-approximation, some solutions are found, i.e. there exists substitutions $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ such that `S(Stop(x), z, cons(plus, u), c)`$\sigma$ is recognized by the tree automaton. Without trace reconstruction, there was no mean to figure out if it was a real problem or an approximation artefact. Now, using trace reconstruction, we can find automatically a counter-example, i.e. the smallest rewriting path between a particular term of the infinite language $\mathcal{L}(\texttt{A1})$ and a term matching `S(Stop(x), z, cons(plus, u), c)`. The rewriting path obtained by our prototype is given using the following syntax: $s$ `-[|` *applied rule, position* `]->` $s'$ ...

```
Statistics:
 - Number of nodes visited: 23921
 - Computation Time: 11.39 seconds
 - Trace(s):
S(Proc(cons(plus,nil),o),Proc(cons(plus,nil),o),nil,nil)
-[|S(Proc(cons(plus,y),c),z,m,n) -> S(Proc(y,s(c)),z,m,n),epsilon|]->
   S(Proc(nil,s(o)),Proc(cons(plus,nil),o),nil,nil)
   -[|S(Proc(nil,c),z,nil,n) -> S(Stop(c),z,nil,n),epsilon|]->
      S(Stop(s(o)),Proc(cons(plus,nil),o),nil,nil)
      -[|S(x,Proc(cons(plus,y),c),m,n) ->
         S(x,Proc(y,c),add(plus,m),n),epsilon|]->
            S(Stop(s(o)),Proc(nil,o),add(plus,nil),nil)
            -[|add(x,nil) -> cons(x,nil),epsilon.3|]->
               S(Stop(s(o)),Proc(nil,o),cons(plus,nil),nil)
```

Thanks to trace reconstruction, our system has found the two necessary conditions for the problem to occur:

- the $P_-$ process has to have at least one '+' symbol in its input list (initial term), and
- the $P_+$ process needs to count all the symbols of its list and terminates before the $P_-$ process starts to store '+' symbols in the $P_+$ FIFO (rewriting sequence)

then $P_+$ is stopped with a non-empty FIFO. Note that, using $\alpha_2$ instead of $\alpha_1$, completion does not terminate but after a finite number of completion steps, a similar trace can be found by visiting fewer nodes (113 nodes in 2.86 seconds) thanks to more precise approximation.

The problem found here can be fixed by adding an additional symbol: 'end' which has to be added by process $P_+$ to $P_-$ FIFO when $P_+$ has reached the end of its list, and symmetrically for $P_-$. Then, a process can stop if and only if it has reached the end of its list and if it has read the 'end' symbol in its FIFO. On the corrected TRS, it is possible to compute an over-approximation of all reachable terms with Timbuk. In the obtained approximation, no dead-lock situation occurs, proving the property [BG06].

In some other experiments, we used approximations and trace reconstruction, to find attacks in rewrite specifications of cryptographic protocols [BG06]. On those examples with more complex search trees, the obtained results show that computing first an over-approximation of reachable terms and then searching for a particular reachable term provides a very efficient alternative to usual breath-first search in the rewriting tree. In particular, in [BG06] we give an example where trace reconstruction succeeds and Maude [CDE$^+$01] exhausts memory.

## 6 Conclusion

In this paper, we have presented a trace reconstruction method for over-approximations of sets of reachable terms. The proposed algorithm takes advantage of the completion-based approximation construction to prune the search space in the set of all possible traces. Completeness of the approach ensures that if a trace exists then it can be obtained by trace reconstruction, whatever the approximation may be. However, since we are dealing with infinite models, it is not always possible to conclude because trace reconstruction may diverge if it starts from a term that is not reachable.

With regards to other works in the domain, the first main interest of our technique is that it can find reachable terms on infinite sets (regular tree languages) of initial input terms. As shown on the counting processes example, trace reconstruction permits to find the exact rewriting path to a reachable term from an infinite set of possible input terms. When (1) the set of initial terms is infinite and (2) the term rewriting system is not confluent not terminating, this problem can hardly be tackled by usual rewriting tools such as Elan[BKK+98] or Maude[CDE+01] as well as by completion based tools like Waldmeister[GHLS03]. However, in order to compare with some existing tools, we also achieved some experiments on finite sets of initial terms in the case of cryptographic protocols [BG06]. It comes up that, for reachability analysis and in some particular cases, our prototype can compete with a cutting edge rewrite engine like Maude.

Furthermore, when every problem is corrected in the rewrite specification, the usual tree automata completion algorithm is able to prove that problem/attack/deadlock are not reachable, hence cannot happen in the system, by over-approximating the set of reachable terms. This is to be used in the TA4SP tool (based on Timbuk) which is part of the AVISPA [ABB+05] protocol verification tool so as to discriminate between reachable terms (real attacks) and terms of the approximation.

## 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.

[BG06] Y. Boichut and Th. Genet. Trace reconstruction. Research Report RR2006-02, LIFC - Laboratoire d'Informatique de l'Université de Franche Comtéand IRISA / Université de Rennes, 2006. `http://lifc.univ-fcomte.fr/publis/papers/Year/2006.html`.

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

[BKK+98] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of elan. In *Proc. 2nd WRLA*, ENTCS, Pont-à-mousson (France), 1998. Elsevier.

[BMV03] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Available at *http://www.avispa-project.org*.

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

[CDE+01] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

[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.

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

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. 2nd WRLA Workshop, Pont à Mousson (France)*, 1998.

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

[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.

[GHLS03] J.-M. Gaillourdet, Th. Hillenbrand, B. Löchner, and H. Spies. The new Waldmeister loop at work. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction*, volume 2741 of *LNAI*, pages 317–321. Springer-Verlag, 2003.

[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.

[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.

[GV98] P. Gyenizse and S. Vágvölgyi. Linear Generalized Semi-Monadic Rewrite Systems Effectively Preserve Recognizability. *TCS*, 194(1-2):87–122, 1998.

[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/`.

[GVTT01] T. Genet and Valérie Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with timbuk. In *Proc. 8th LPAR Conf., Havana (Cuba)*, volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proc. 7th RTA Conf., New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.

[Low95] G. Lowe. An Attack on the Needham-Schroder Public-Key Protocol. *IPL*, 56:131–133, 1995.

[Mon99] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Proc. 6th SAS, Venezia (Italy)*, 1999.

[NR05] M. Nesi and G. Rucci. Formalizing and Analyzing the Needham-Schroeder Symmetric-Key Protocol by Rewriting. In *In Proceedings of the 2nd Workshop on Automated Reasoning for Security Protocol Analysis*, 2005.

[NRV03] Monica Nesi, Giuseppina Rucci, and Massimo Verdesca. A rewriting strategy for protocol verification. *Electr. Notes Theor. Comput. Sci*, 86(4), 2003.

[NS78] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM*, 21(12):993–999, 1978.

[OCKS03] F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic Approximation for the Verification of Cryptographic Protocols. In *In Proceedings of FASE'03*, volume 2629 of *LNCS*, pages 34–48. Springer-Verlag, 2003.

[TKS00] T. Takai, Y. Kaji, and H. Seki. Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability. In *Proc. 11th RTA Conf., Norwich (UK)*, volume 1833 of *LNCS*. Springer-Verlag, 2000.