Handling Non Left-Linear Rules When Completing Tree Automata

Yohan Boichut, Roméo Courbis, Pierre-Cyrille Héam, Olga Kouchnarenko

To cite this version:

HAL Id: hal-00561373
https://hal.archives-ouvertes.fr/hal-00561373
Submitted on 1 Feb 2011

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

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Handling Non Left-Linear Rules When Completing Tree Automata*

YOHAN BOICHUT
LIFO Bat. SIA Université d’Orléans Rue Léonard de Vinci B.P. 6759 F-45067 ORLEANS

ROMEO COURBIS
INRIA-CASSIS LIFC, Université de Franche-Comté, 16 route de Gray, 25000 Besançon, FRANCE

PIERRE-CYRILLE HEAM
LSV, CNRS-INRIA, Ecole Normale Supérieure de Cachan, 61 avenue du Président Wilson, 94230 Cachan, FRANCE

OLGA KOUCHNARENKO
INRIA-CASSIS LIFC, Université de Franche-Comté, 16 route de Gray, 25000 Besançon, FRANCE

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

This paper addresses the following general problem of tree regular model-checking: decide whether $R^*(L) \cap L_p = \emptyset$ where $R^*$ is the reflexive and transitive closure of a successor relation induced by a term rewriting system $R$, 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 the case when term rewriting system rules are non left-linear.

Keywords: Rewriting techniques; tree automata; left-linearity; security.

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 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 (TRS for short). Afterwards, a system reachability-based analysis

*This work was granted by the French ANR project RAVAJ.
is reduced to the regular languages closure computation under (term) rewriting systems: given a regular language \( L \), a relation \( R \) induced by a (term) rewriting system and a regular set \( L_p \) of bad configurations, the problem is to decide whether \( R^* (L) \cap L_p = \emptyset \) where \( R^* \) is the reflexive and transitive closure of \( R \). Since \( R^* (L) \) 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 \( R^* (L) \). Computing an over-approximation \( K_{\text{over}} \) of \( R^* (L) \) may be useful for the verification if \( K_{\text{over}} \cap L_p = \emptyset \), proving that \( R^* (L) \cap L_p = \emptyset \). Dually, under-approximation may be suitable to prove that \( R^* (L) \cap L_p \neq \emptyset \) (see Fig.1). 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 follows and adapts an expert-human guided approximation technique introduced in [17] for left-linear term-rewriting systems. More precisely, the paper 1) extends this approach to term rewriting systems with non-left-linear rules, 2) illustrates its advantages on examples, and 3) points out the application domains.

Notice that a preliminary version of the present paper has been published [7]. The results in [7] are obtained for left-quadratic TRSs. To make it short, the present paper extends the model in [7] and generalises the underpinning constructions to deal with all non left-linear term rewriting systems. In addition, more examples are given to illustrate the definitions.

![Figure 1. Is the intersection between \( K_{\text{over}} \) and \( L_p \) empty?](image)

**Related Work** Given a term rewriting system \( R \) and two ground terms \( s \) and \( t \), deciding whether \( s \overset{*}{\rightarrow}_R t \) 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 general. Several syntactic classes of term rewriting systems have been pointed out to have a decidable accessibility problem, for in-
Handling Non-left Linear Rules When Completing Tree Automata

2. Preliminaries

2.1. Terms and TRSs

Comprehensive surveys can be found in [16,2] for term rewriting systems, and in [12,19] for tree automata and tree language theory.

Let \( \mathcal{F} \) be a finite set of symbols, associated with an arity function \( \alpha : \mathcal{F} \rightarrow \mathbb{N} \), 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 \( \text{Var}(t) \). A substitution is a function \( \sigma \) from \( \mathcal{X} \) into \( \mathcal{T}(\mathcal{F}, \mathcal{X}) \), which can be extended uniquely to an endomorphism of \( \mathcal{T}(\mathcal{F}, \mathcal{X}) \). A position \( p \) for a term \( t \) is a word over \( \mathbb{N} \). The empty sequence \( \epsilon \) denotes the top-most position. The set \( \mathcal{P}(t) \) of positions of a term \( t \) is inductively defined by: \( \mathcal{P}(t) = \{ \epsilon \} \) if \( t \in \mathcal{X} \) and \( \mathcal{P}(f(t_1, \ldots, t_n)) = \{ \epsilon \} \cup \{ \mathcal{P}(t_i) \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}(t_i) \} \). If \( p \in \mathcal{P}(t) \), then \( t|_p \) denotes the subterm of \( t \) at position \( p \) and \( t|_p s \) denotes the term obtained by replacement of the subterm \( t|_p \) at position \( p \) by the term \( s \). We also denote by \( t(p) \) the symbol occurring at \( t \) at position \( p \). Given a term \( t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \), we denote \( \mathcal{P}(t) \) (\( \subseteq \mathcal{P}(t) \)) the set of positions of \( t \) such that \( \mathcal{P}(t) = \{ p \in \mathcal{P}(t) \mid t(p) \in \mathcal{X} \} \).

Thus \( \mathcal{P}(t) \) is the set of functional positions of \( t \).

A term rewriting system (TRS) \( \mathcal{R} \) is a set of rewrite rules \( l \rightarrow r \), where \( l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \) and \( l \not\in \mathcal{X} \). A rewrite rule \( l \rightarrow r \) is \( h \)-left-linear (resp. \( h \)-right-linear) if each variable of \( l \) (resp. \( r \)) occurs at most \( h \) times within \( l \) (resp. \( r \)). A TRS \( \mathcal{R} \) is \( h \)-left-linear (resp. \( h \)-right-linear) if every rewrite rule \( l \rightarrow r \) of \( \mathcal{R} \) is \( h \)-left-linear (resp. \( h \)-right-linear). For above linearity definitions, if \( h = 1 \), the prefix “\( h \)” is omitted. A TRS \( \mathcal{R} \) is linear if it is right-linear and left-linear. The TRS \( \mathcal{R} \) induces a rewriting relation \( \rightarrow_\mathcal{R} \) on terms whose reflexive transitive closure is written \( \rightarrow^*_\mathcal{R} \). The set of \( \mathcal{R} \)-descendants of a set of ground terms \( E \) is \( \mathcal{R}^*(E) = \{ t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \rightarrow^*_\mathcal{R} t \} \). For every positive integer \( k \), any ground terms \( s \) and \( t \), we inductively define \( s \rightarrow^k_\mathcal{R} t \) by: \( s \rightarrow^0_\mathcal{R} t \) if and only if \( s = t \), and \( s \rightarrow^{k+1}_\mathcal{R} t \) if and only if there exists a term \( s_1 \) such that \( s_1 \rightarrow^k_\mathcal{R} t \) and \( s \rightarrow_\mathcal{R} s_1 \).
2.2. Tree Automata Completion

Note that $\mathcal{R}^*(E)$ is possibly infinite: $\mathcal{R}$ may not terminate and/or $E$ may be infinite. The set $\mathcal{R}^*(E)$ is generally not computable [19]. However, it is possible to over-approximate it [17] using tree automata, i.e. a finite representation of infinite (regular) sets of terms. We next define tree automata.

Let $Q$ be a finite set of symbols, of arity 0, called states such that $Q \cap F = \emptyset$. $T(F \cup Q)$ is the set of configurations $A$ transition is a rewrite rule $c \rightarrow q$, where $c \in T(F \cup Q)$ is a configuration and $q \in Q$. A normalised transition is a transition $c \rightarrow q$ where $c = f(q_1, \ldots, q_n)$, $f \in F$, $\text{ar}(f) = n$, and $q_1, \ldots, q_n \in Q$. A bottom-up non-deterministic finite tree automaton (tree automaton for short) is a quadruple $A = (F, Q, Q_f, \Delta)$, $Q_f \subseteq Q$ and $\Delta$ is a finite set of normalised transitions. The rewriting relation on $T(F \cup Q)$ induced by the transition set $\Delta$ of $A$ is denoted $\rightarrow_\Delta$. When $\Delta$ is clear from the context, $\rightarrow_\Delta$ is also written $\rightarrow_A$. The tree language recognised by $A$ in a state $q$ is $L(A, q) = \{ t \in T(F) \mid t \rightarrow^*_A q \}$. The language recognised by $A$ is $L(A) = \bigcup_{q \in Q_f} L(A, q)$. A tree language is regular if and only if it is recognised by a tree automaton. We denote by $C[q_1, \ldots, q_n]$ a term of $T(F \cup \{ \ldots, q_n \})$.

Let us now recall how tree automata and TRSs can be used for term reachability analysis. Given a tree automaton $A$ and a TRS $\mathcal{R}$, the tree automata completion algorithm proposed in [17] computes a tree automaton $A^{k}_{\mathcal{R}}$ such that $L(A^{k}_{\mathcal{R}}) = \mathcal{R}^*(L(A))$ when it is possible (for the classes of TRSs where an exact computation is possible, see [17]), and such that $L(A^{k}_{\mathcal{R}}) \supseteq \mathcal{R}^*(L(A))$ otherwise.

The tree automata completion works as follows. From $A = A^0_{\mathcal{R}}$ completion builds a sequence $A^0_{\mathcal{R}}, A^1_{\mathcal{R}}, \ldots A^k_{\mathcal{R}}$ of automata such that if $s \in L(A^k_{\mathcal{R}})$ and $s \rightarrow_{\mathcal{R}} t$ then $t \in L(A^{k+1}_{\mathcal{R}})$. If there is a fix-point automaton $A^k_{\mathcal{R}}$ such that $\mathcal{R}^*(L(A^k_{\mathcal{R}})) = L(A^k_{\mathcal{R}})$, then $L(A^k_{\mathcal{R}}) = \mathcal{R}^*(L(A^k_{\mathcal{R}}))$ (or $L(A^k_{\mathcal{R}}) \supseteq \mathcal{R}^*(L(A))$ if $\mathcal{R}$ is in no class of [17]). To build $A_{\mathcal{R}}^{k+1}$ from $A_{\mathcal{R}}^k$, a completion step is achieved. It consists of finding critical pairs between $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{A^k_{\mathcal{R}}}$. To define the notion of critical pair, the substitution definition is extended to terms in $T(F \cup Q)$. For a substitution $\sigma : \mathcal{X} \rightarrow Q$ and a rule $l \rightarrow r \in \mathcal{R}$ such that $\text{Var}(r) \subseteq \text{Var}(l)$, if there exists $q \in Q$ satisfying $l\sigma \rightarrow_{A^k_{\mathcal{R}}}^* q$ then $l\sigma \rightarrow_{A^k_{\mathcal{R}}} q$ and $l\sigma \rightarrow_{\mathcal{R}} r\sigma$ is a critical pair. Note that since $\mathcal{R}$ and $A^k_{\mathcal{R}}$ is finite, there is only a finite number of critical pairs. Thus, for every critical pair detected between $\mathcal{R}$ and $A^k_{\mathcal{R}}$ such that $r\sigma \not\rightarrow_{A^k_{\mathcal{R}}}^* q$, the tree automaton $A_{\mathcal{R}}^{k+1} = A_{\mathcal{R}}^{k+1} \cup \{ q \}$ is constructed by adding a new transition $r\sigma \rightarrow q$ to $A^k_{\mathcal{R}}$. Consequently, $A_{\mathcal{R}}^{k+1}$ recognises $r\sigma$ in $q$, i.e. $r\sigma \rightarrow_{A_{\mathcal{R}}^{k+1}} q$. However, the transition $r\sigma \rightarrow q$ is not necessarily normalised. Then, we use abstraction functions whose goal is to define a set of normalised transitions $\text{Norm}$ such that $r\sigma \rightarrow_{\text{Norm}}^* q$. Thus, instead of adding the transition $r\sigma \rightarrow q$ which is not normalised, the set of transitions $\text{Norm}$ is added to $\Delta$, i.e., the transition set of the current automaton $A^k_{\mathcal{R}}$.

We give below a very general definition of abstraction functions which allow to each functional position of $r\sigma$ a state of $Q$. The role of an abstraction function remains to define equivalence classes of terms where one class corresponds to one
state of \( Q \). An abstraction function \( \gamma \) is a function \( \gamma : ((R \times (X \rightarrow Q) \times Q) \rightarrow \mathbb{N}^*) \rightarrow Q \) such that \( \gamma(l \rightarrow r, \sigma, q)(\epsilon) = q \). Thus, given an abstraction function \( \gamma \), the normalisation of a transition \( r \sigma \rightarrow q \) is defined as follows. Let \( \gamma \) be an abstraction function, \( \Delta \) be a transition set, \( l \rightarrow r \in R \) with \( \text{Var}(r) \subseteq \text{Var}(l) \) and \( \sigma : X \rightarrow Q \) such that \( l \sigma \rightarrow \Delta q \). The \( \gamma \)-normalisation of the transition \( r \sigma \rightarrow q \), written \( \text{Norm}_\gamma(l \rightarrow r, \sigma, q) \), is defined by:

\[
\text{Norm}_\gamma(l \rightarrow r, \sigma, q) = \{ r(p)(\beta_{p,1}, \ldots, \beta_{p,n}) \rightarrow \beta \mid p \in \text{Pos}_R(r), \\
\beta = \begin{cases} \{ q \mid p = e \}, & \gamma(l \rightarrow r, \sigma, q)(p) \text{ otherwise}, \\
\gamma(l \rightarrow r, \sigma, q)(p) \text{ otherwise}. & \end{cases} \\
\beta_{p,i} = \begin{cases} \sigma(r(p,i)) \text{ if } r(p,i) \in X \\
\{ \sigma(r(p,i)) \mid \text{otherwise} \}. & \end{cases}
\]

Example 1. Let \( A = (F, Q, Q_f, \Delta) \) be the tree automaton such that \( F = \{ a, b, c, d, e, f, \omega \} \) with \( ar(s) = 1 \) with \( s \in \{ a, b, c, d, e, f, \omega \} \) and \( ar(\omega) = 0 \), \( Q = \{ q_0, q_f, q_\omega \} \), \( Q_f = \{ q_f \} \) and \( \Delta = \{ \omega \rightarrow q_\omega, b(q_\omega) \rightarrow q_0, a(q_0) \rightarrow q_f \} \). Thus, \( L(A) = \{ a(b(\omega)) \} \). Given the TRS \( R = \{ a(x) \rightarrow c(d(x)), b(x) \rightarrow e(f(x)) \} \), two critical pairs are computed: \( a(q_0) \rightarrow a q_f, a(q_0) \rightarrow c(d(q_0)) \) and \( b(q_\omega) \rightarrow b q_f, b(q_\omega) \rightarrow c(f(q_\omega)) \).

Let \( \gamma \) be the abstraction function such that \( \gamma(a(x) \rightarrow c(d(x)), \epsilon) = q_f, \gamma(q_0) \rightarrow q_f(1) = q_f, \gamma(b(x) \rightarrow e(f(x)), \epsilon) = q_f \) and \( \gamma(b(x) \rightarrow e(f(x)), \epsilon) = q_f, \gamma(q_0) \rightarrow q_f(1) = q_f \). So, \( \text{Norm}_\gamma(a(x) \rightarrow c(d(x)), \epsilon) = \{ q_0, q_f \} \) and \( \text{Norm}_\gamma(b(x) \rightarrow e(f(x)), \epsilon) = \{ q_0, q_f \} \).

Now we formally define what a completion step is. Let \( A = (F, Q, Q_f, \Delta) \) be a tree automaton, \( \gamma \) an abstraction function and \( R \) a left-linear TRS. We define a completion \( C_R^\gamma(A) = (F, Q, Q', Q', \Delta') \) with:

- \( \Delta' = \Delta \cup \{ q \mid r \rightarrow q \sigma \rightarrow \Delta' q \text{ with } \text{Norm}_\gamma(r \rightarrow q, \sigma, q) \} \) and
- \( Q' = \{ q \mid \epsilon \rightarrow q \in \Delta' \} \) and
- \( Q_f = Q_f \).

Example 2. Given a tree automaton \( A \) and a completion step \( \text{Step}(A) \), performing one completion step on \( A \) gives the automaton \( C_R^\gamma(A) = (F, Q, Q_f, \Delta') \) where \( \Delta' = \Delta \cup \{ q \mid r \rightarrow q \sigma \rightarrow \Delta' q \text{ with } \text{Norm}_\gamma(r \rightarrow q, \sigma, q) \} \).

Notice that \( C_R^\gamma(A) \) is \( R \)-close, and in fact an over-approximation of \( R^\gamma(L(A)) \) is computed. Indeed, the tree automaton \( C_R^\gamma(A) \) recognises the term \( a(c(e(f(\omega)))) \) when \( R^\gamma(L(A)) = \{ a(b(\omega)), a(c(e(f(\omega)))), c(d(b(\omega))), c(d(e(f(\omega)))) \} \).

Proposition 3 (Theorem 1 in [17]) Let \( A \) be a tree automaton and \( R \) be a TRS such that \( A \) is deterministic or \( R \) is left-linear, and for every \( l \rightarrow r \in R \), \( \text{Var}(r) \subseteq \text{Var}(l) \). For any abstraction function \( \gamma \), one has:

\[
L(A) \cup R(L(A)) \subseteq L(C_R^\gamma(A)).
\]
In addition, an abstraction function can be defined in such a way that only terms, actually reachable, will be computed. This class of abstraction functions is called \((\mathcal{A}, \mathcal{R})\)-exact abstraction functions in [3].

Let \(\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)\) be a tree automaton and \(\mathcal{R}\) be a TRS. Let \(\text{Im}(\gamma) = \{q | \forall l \rightarrow r \in \mathcal{R}, \forall p \in \text{Pos}(r) \text{ s.t. } \gamma(l \rightarrow r, \sigma, q)(p) = q\}\). An abstraction function \(\gamma\) is \((\mathcal{A}, \mathcal{R})\)-exact if \(\gamma\) is injective and \(\text{Im}(\gamma) \cap \mathcal{Q} = \emptyset\).

By adapting the proof of Theorem 2 in [17] to the new class of abstractions, we show that with such abstraction functions, only reachable terms are computed.

**Theorem 4 (Theorem 2 in [17])** Let \(\mathcal{A}\) be a tree automaton and \(\mathcal{R}\) be a TRS such that \(\mathcal{A}\) is deterministic or \(\mathcal{R}\) is right-linear. Let \(\alpha\) be an \((\mathcal{A}, \mathcal{R})\)-exact abstraction function. One has:

\[
\mathcal{C}^{\mathcal{R}}_{\alpha}(\mathcal{A}) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A})).
\]

We now give the general result in [17] saying that, if there exists a fix-point automaton, then its language contains all the terms actually reachable by rewriting, at least.

**Theorem 5 (Theorem 1 in [17])** Let \(\mathcal{A}, \mathcal{R}\) be respectively a tree automaton, a TRS. For any abstraction function \(\gamma\), if there exists \(N \in \mathbb{N}\) and \(N \geq 0\) such that \((\mathcal{C}^{\mathcal{R}}_{\gamma})^{(N)}(\mathcal{A}) = (\mathcal{C}^{\mathcal{R}}_{\gamma})^{(N+1)}(\mathcal{A})\), then \(\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}((\mathcal{C}^{\mathcal{R}}_{\gamma})^{(N)}(\mathcal{A}))\).

The above method does not work for all TRSs. For instance, consider a constant \(A\) and the tree automaton \(\mathcal{A} = (\{q_1, q_2, q_f\}, \{A \rightarrow q_1, A \rightarrow q_2, f(q_1, q_2) \rightarrow q_f\}, \{q_f\})\) (Fig. 2) and the TRS \(\mathcal{R} = \{f(x, x) \rightarrow g(x)\}\). There is no substitution \(\sigma\) 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 procedure stops (in fact does not begin) before providing an over-approximation of \(\mathcal{R}^*(\mathcal{L}(\mathcal{A}))\).

### 3. Contributions

This section extends an approximation-based technique introduced in [17] for left-linear term-rewriting systems, to TRSs with non left-linear rules. If the term-rewriting system is not left-linear, it is possible to apply Proposition 3 by determinizing the involved automaton. However, since the determinisation is exponential and since it has to be done at each completion step (which doesn’t preserve the determinism), the procedure is too expensive to be used in practice. In this framework,
our contribution provides a well-suited (parametrised) model and underpinning theoretical tools that avoid these exponential steps when the term rewriting system is h-left-linear, with a small h. The small values of h can be justified by the fact that most of TRS’s modelling concrete verification problems are 2-left-linear. Indeed, non-left linear variables frequently occur to encode comparisons between two elements of the specified system and, in most cases only two elements are compared.

Definition 6. Let \( A = (Q, \Delta, Q_f) \) be a finite bottom-up tree automaton. The automaton \( A^{(h)} = (Q^{(h)}, \Delta^{(h)}, Q_f^{(h)}) \), for \( h \geq 1 \), is defined by:

- \( Q^{(h)} = \{ \{q\} \mid q \in Q \} \cup \{ \{q\} \mid \text{Card}Q \leq h \} \) (states of \( Q^{(h)} \) are denoted with an \( (h) \) exponent),
- \( Q_f^{(h)} = \{ \{q\} \mid q \in Q_f \} \),
- \( \Delta^{(h)} = \{ f(q_1^{(h)}, \ldots, q_n^{(h)}) \rightarrow q^{(h)} \mid \exists q, q_1, \ldots, q_n \in Q, \forall 1 \leq i \leq n \in Q_i^{(h)} \} \) and \( f(q_1, \ldots, q_n) \rightarrow q \in \Delta \).

![Diagram of \( A^{(2)} \) on \( f(A, A) \).]

To illustrate the definition above, let’s consider the automaton \( A \) whose final state is \( q_f \) and whose transitions are \( A \rightarrow q_1, A \rightarrow q_2 \) and \( f(q_1, q_2) \rightarrow q_f \). The states of \( A^{(2)} \) are all pairs of states and singletons over \( \{q_1, q_2, q_f\} \), and the transitions are \( A \rightarrow \{q_1\}, A \rightarrow \{q_2\}, A \rightarrow \{q_1, q_2\}, f(\{q_1\}, \{q_2\}) \rightarrow \{q_f\}, f(\{q_1, q_2\}, \{q_2, q_1\}) \rightarrow \{q_f\} \) for all \( i, j \)’s in \( \{1, 2, f\} \) (Fig. 3). When considering only the accessible states, among all the transitions above we just have transitions \( f(\{q_1, q_2\}) \rightarrow \{q_1, q_2\} \rightarrow \{q_f\} \) for all \( i, j \)’s in \( \{1, 2\} \).

Proposition 7. One has \( \mathcal{L}(A) = \mathcal{L}(A^{(h)}) \).

Proof. By definition of \( A^{(h)} \), if \( f(q_1, \ldots, q_n) \rightarrow q \in \Delta, \) then \( f(\{q_1\}, \ldots, q_n) \rightarrow \{q\} \in \Delta^{(h)} \). Consequently, for every term \( t \) such that \( t \rightarrow_A q \), one also has \( t \rightarrow_{A^{(h)}} \{q\} \). Since for every \( q_f \in Q_f, \{q_f\} \in Q_f^{(h)} \), \( \mathcal{L}(A) \subseteq \mathcal{L}(A^{(h)}) \).

It remains to prove that \( \mathcal{L}(A^{(h)}) \subseteq \mathcal{L}(A) \). We will prove by induction on \( k \) that for every \( k \geq 1 \), for every term \( t, \) every state \( q^{(h)} \) of \( A^{(h)} \), if \( t \rightarrow_k A^{(h)} q^{(h)} \), then for all \( q \in q^{(h)} \), \( t \rightarrow_A q \).

- If \( t \rightarrow_A q^{(h)} q^{(h)} \), then, by definition of \( \Delta^{(h)} \), \( t \) is a constant and for all \( q \in q^{(h)} \), there exists a transition \( t \rightarrow q \) in \( A \).
Proof. We prove by induction on $i$ that there are states of $\mathcal{A}(h)$ such that $q \rightarrow^i \mathcal{A}(h) q$. Consequently, for all $q \in q(h)$, $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$, proving the induction.

\begin{lemma}
Let $k \geq 1$. If $C[q_1, \ldots, q_n] \rightarrow_A q$ and if $q_1^{(h)}, \ldots, q_n^{(h)}$ are states of $\mathcal{A}(h)$ satisfying $q_i \in q_i^{(h)}$ for all $1 \leq i \leq n$, then $C[q_1^{(h)}, \ldots, q_n^{(h)}] \rightarrow^k \mathcal{A}(h) \{q\}$. 
\end{lemma}

Proof. Assume now that the claim is true for a fixed positive integer $k$. Let $t$ be a term and $q(h) \in \mathcal{A}(h)$ such that $t \rightarrow^{k+1} q(h)$. Consequently, there exists $f \in \mathcal{T} \ x \in \mathcal{F}_n$ such that $t \rightarrow^k \mathcal{A}(h) f(q_1^{(h)}, \ldots, q_n^{(h)}) \rightarrow^k \mathcal{A}(h) q(h)$. It follows that $t = f(t_1, \ldots, t_k)$ and for all $1 \leq i \leq k$, $t_i \rightarrow^k \mathcal{A}(h) q_i^{(h)}$. Using the induction hypothesis, $t_i \rightarrow^k \mathcal{A}(h) q_i^{(h)}$, for all $q_i \in q_i^{(h)}$. Consequently, for all $q \in q(h)$, $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$, proving the induction.

So, $\mathcal{L}(\mathcal{A}(h)) \subseteq \mathcal{L}(\mathcal{A})$.

\begin{lemma}
If $C[q_1, \ldots, q_n] \rightarrow_A q$ and if $q_1^{(h)}, \ldots, q_n^{(h)}$ are states of $\mathcal{A}(h)$ satisfying $q_i \in q_i^{(h)}$ for all $1 \leq i \leq n$, then $C[q_1^{(h)}, \ldots, q_n^{(h)}] \rightarrow^k \mathcal{A}(h) \{q\}$. 
\end{lemma}

Proof. Assume now that the proposition is true for all $j \leq k$ and that $C[q_1, \ldots, q_n] \rightarrow^k \mathcal{A}(h) q$. There exist $q_1^{(h)}, \ldots, q_n^{(h)}$ states of $\mathcal{A}$ and $f \in \mathcal{F}_n$ such that $C[q_1, \ldots, q_n] \rightarrow^k \mathcal{A}(h) f(q_1', \ldots, q_n') \rightarrow^k \mathcal{A}(h) q$. Consequently, $C[q_1, \ldots, q_n]$ of the form $C[q_1', \ldots, q_n'] = f(t_1, \ldots, t_k)$ where the $t_i$’s are terms over $\mathcal{F}_n$. Moreover, for all $i$, there exists $k_i \leq k$ such that $t_i \rightarrow^{k_i} \mathcal{A}(h) q_i'$. It follows that $C[q_1', \ldots, q_n'] \rightarrow^k \mathcal{A}(h) \{q_i'\}$, proving the lemma.

\begin{lemma}
If there are states of $\mathcal{A}$, with $j \leq h$ such that $t \rightarrow^* \mathcal{A}(h)$ for every $1 \leq i \leq j$, then $t \rightarrow^* \mathcal{A}(h) \{q_i \mid 1 \leq i \leq j\}$. 
\end{lemma}

Proof. If $t \rightarrow^* \mathcal{A}(h) q_i$ for every $1 \leq i \leq j$, then there exist functions $\pi_i$’s from positions of $t$ into $\mathcal{Q}$ such that $\pi_i(\varepsilon) = q_i$ and for every position $p$ of $t$, if $t|_p \in \mathcal{F}_n$, then $t(p)(\pi_i(p.1), \ldots, \pi_i(p.n)) \rightarrow \pi_i(p)$ is a transition of $\mathcal{A}$.

Therefore, by definition of $\mathcal{A}(h)$, $t(p)(\pi_i(p.1) \mid 1 \leq i \leq j), \ldots, (\pi_i(p.n) \mid 1 \leq i \leq j)$ is in $\mathcal{A}(h)$. It follows that $t \rightarrow^* \mathcal{A}(h) \{q_i \mid 1 \leq i \leq j\}$.

\begin{proposition}
If each variable occurs at most $h$ times in left-hand sides of rules \begin{math}\end{math} of $\mathcal{R}$, then $\mathcal{R}(\mathcal{L}(\mathcal{A})) \cup \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{C}_\mathcal{R}(\mathcal{A}(h)))$. 
\end{proposition}

Proof. Since $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}(h))$ and since $\mathcal{L}(\mathcal{A}(h)) \subseteq \mathcal{L}(\mathcal{C}_\mathcal{R}(\mathcal{A}(h)))$, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{C}_\mathcal{R}(\mathcal{A}(h)))$. It remains to show that $\mathcal{R}(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{C}_\mathcal{R}(\mathcal{A}(h)))$. 

Let $t \in \mathcal{R}(\mathcal{L}(\mathcal{A}))$. By definition there exists a rule $l \rightarrow r \in \mathcal{R}$, a position $p$ of $t$ and a substitution $\mu$ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F})$ such that

$$t = t[r\mu]_p \quad \text{and} \quad t[l\mu]_p \in \mathcal{L}(\mathcal{A})$$

(1)

It follows there exist states $q, q_f$ of $\mathcal{A}$ such that $q_f$ is final, and

$$l\mu \rightarrow^*_{\mathcal{A}} q \quad \text{and} \quad t[q]_p \rightarrow^*_{\mathcal{A}} q_f.$$  

(2)

Consequently, by Definition 6, one has

$$l\mu \rightarrow^*_{\mathcal{A}(h)} \{q\} \quad \text{and} \quad t[q]_p \rightarrow^*_{\mathcal{A}(h)} \{q_f\}.$$  

(3)

If $r\mu \rightarrow^*_{\mathcal{A}(h)} \{q\}$, then (3) implies that $t[r\mu]_p \rightarrow^*_{\mathcal{A}(h)} \{q_f\}$. In this case, since $t = t[r\mu]_p$ and since $\{q_f\}$ is by construction a final state of $\mathcal{A}(h)$, $t$ is in $\mathcal{L}(\mathcal{A}(h))$, which is a subset of $\mathcal{L}(\mathcal{C}_h(\mathcal{A}(h)))$.

Now we may assume that $r\mu \not\rightarrow^*_{\mathcal{A}(h)} \{q\}$. Let $P_i$ be the set of variable positions of $l$; i.e. $P_i = \{p \mid l(p) \in \mathcal{X}\}$. Set $P_l = \{p_1, \ldots, p_l\}$. Since $l\mu \rightarrow^*_{\mathcal{A}} q$, by (2) there exist states $q_1, \ldots, q_l$ of $\mathcal{A}$ such that

$$\mu(l(p_i)) \rightarrow^*_{\mathcal{A}} q_i \quad \text{and} \quad l[q_1]_{p_1} \ldots [q_l]_{p_l} \rightarrow^*_{\mathcal{A}} q.$$  

(4)

We define the substitution $\sigma$ from variables occurring in $l$ into $2^\mathcal{Q}$ by: $\sigma(x_i) = \{q_i \mid l(p_i) = x_i\}$. Since $l$ is h-left-linear, for each $x_i$, $\sigma(x_i)$ contains at most h states. We claim that $l\sigma \rightarrow^*_{\mathcal{A}(h)} q$. Indeed by (4) and by Lemma 9 for each $x_i$ occurring in $l$, $\mu(x_i) \rightarrow^*_{\mathcal{A}(h)} \sigma(x_i)$. It follows that $l\mu \rightarrow^*_{\mathcal{A}(h)} l\sigma$. By (4) and using Lemma 8, $l\sigma \rightarrow^*_{\mathcal{A}(h)} \{q\}$, proving the claim. By construction of $\mathcal{C}_h^{\mathcal{R}}(\mathcal{A}(h)), r\sigma \rightarrow^*_{\mathcal{C}_h^{\mathcal{R}}(\mathcal{A}(h))} \{q\}$. Moreover, by definition of $\sigma$, $r\mu \rightarrow^*_{\mathcal{A}(h)} r\sigma$. It follows that

$$t = t[r\mu]_p \rightarrow^*_{\mathcal{A}(h)} t[r\sigma]_p \rightarrow^*_{\mathcal{C}_h(\mathcal{A}(h))} t[q]_p \rightarrow^*_{\mathcal{A}(h)} \{q_f\},$$

which completes the proof. 

\[\square\]

**Proposition 11.** If $\mathcal{R}$ is right-linear and if $\alpha$ is $(\mathcal{A}, \mathcal{R})$-exact, then $\mathcal{L}(\mathcal{C}_h(\mathcal{A}(h))) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

**Proof.** This is a direct consequence of Theorem 4 and Proposition 7. 

\[\square\]

Notice that if $\mathcal{A}$ is a non-deterministic $n$-state automaton, any deterministic automaton recognising $\mathcal{L}(\mathcal{A})$ may have $O(2^n)$ states. Furthermore, considering $h$ has a constant, $\mathcal{A}(h)$ has $O(n^h)$ states and $O(n^{hk})$ transitions, where $k$ is the maximal arity symbol of $\mathcal{F}$. For 2-left-linear term-rewriting systems, $\mathcal{A}(h)$ has significantly less states that a deterministic automaton accepting $\mathcal{L}(\mathcal{A})$. 

4. Example and Application Domains

4.1. Example

We have tested our approach on the following family of examples. We first consider a family of tree automata \((A_n)\) defined as follows: the set of states of \(A_n\) is \(\{q_1, \ldots, q_{2n+2}, q_f\}\), the set of final state is \(\{q_f\}\), and the set of transitions is \(\{\omega \rightarrow q_1, \omega \rightarrow q_2, a(q_1) \rightarrow q_1, a(q_2) \rightarrow q_2, b(q_1) \rightarrow q_1, b(q_2) \rightarrow q_2, a(q_1) \rightarrow q_3, a(q_2) \rightarrow q_4, a(q_1) \rightarrow q_{i+2}, b(q_i) \rightarrow q_{i+2}, f(q_{2n+1}, q_{2n+2}) \rightarrow q_f\}\), for \(i \geq 3\). The automaton \(A_n\) accepts the set of terms of the form \(f(t_1, t_2)\) where \(t_1\) and \(t_2\) are terms over \(\{a, b, \omega\}\) such that \(t_1|_{x \leftarrow i}\) and \(t_2|_{x \leftarrow i}\) exist and are in \(\{a, \{a, b\}\}^*\). Roughly speaking, when using word automata, \(a(b(\omega))\) denotes \(ab\), and each pair \((t_1, t_2)\) can be viewed as words of \(L = \{a, b\}^{n-1}, \{a\}, \{a, b\}^*\) satisfying the condition above. We second consider the term rewriting system \(\mathcal{R}\) containing the single rule \(f(x, x) \rightarrow x\), and we want to prove that \(b^{n-1}a(\omega) \in \mathcal{R}^*(L(A_n))\). Using finitely many times Theorem 4 directly on \(A_n\) may not prove the results. However, to prove the results, one can determinise \(A_n\) before using Theorem 4. But, the minimal automaton of \(L(A_n)\) has \(2^n\) states at least [21], [Exercise 3.20, p. 73]. Then, the completion should be applied to this automaton. Consequently, this automatic proof requires an exponential time step. Using our approach, one can compute \(A^{(t)}\) and apply Proposition 11, that provides the proof requiring a polynomial time step.

4.2. Left-linearity and Security Issues

4.2.1. Security Protocol Analysis

The TRSs used in the security protocol verification context are often non left-linear. Indeed, there is a lot of protocols that cannot be modeled by left-linear TRSs. Unfortunately, to be sound, the approximation-based analysis described in [18] requires the use of left-linear TRSs. Nevertheless, this method can still be applied to some non left-linear TRSs, which satisfy some weaker conditions. In [17] the authors propose new linearity conditions. However, these new conditions are not well-adapted to be automatically checked.

In our previous work [5] we explain how to define a criterion on \(\mathcal{R}\) and \(A\) to make the procedure automatically work for industrial protocols analysis. This criterion ensures the soundness of the method described in [18, 17]. However, to handle protocols the approach in [5] is based on a kind of constant typing. In [6] we go further and propose a procedure supporting a fully automatic analysis and handling – without typing – algebraic properties like \(\text{XOR}\).

Let us first remark that the criterion defined in [17] does not allow managing the \(\text{XOR}\) non left-linear rule. Second, in [5] we have restricted \(\text{XOR}\) operations to typed terms to deal with the \(\text{XOR}\) non left-linear rule. However, some protocols are known to be flawed by type confusing attacks [14, 10, 11]. Notice that our approach in [6] can be applied to any kinds of TRSs. Moreover, it can cope with exponentiation algebraic properties and this way analyse Diffie-Hellman based protocols.
4.2.2. Backward Analysis of Java Bytecode

A recent work [4], dedicated to the static analysis of Java bytecode programs using term-rewriting systems, provides an automatic procedure to translate a Java bytecode into a term rewriting system modeling the code execution on the Java Virtual Machine. In this context, generated TRSs are left-linear but non right-linear. For example the rewriting rule:

\[ x_{\text{InvokeSpecialCC}}(\text{pprotected}, \text{valtrue}, cc, ca, cam, ic) \rightarrow x_{\text{InvokeSpecialCC}}(\text{subclass}(ic, cc), \text{valtrue}, cc, ca, cam, ic) \]

where \( ca, cam, ic \) and \( cc \) are variables, is a 2-right-linear rule. This rule is a part of a Java bytecode instruction translation, corresponding to the invocation of the method. In order to compute approximation refinements as in [3] or to manage backward analyses that are – in general and in practice – more efficient that forward analyses – term rewriting systems have to be turned left-right, i.e. left- and right-hand sides of rules have to be permuted. By this permutation 2-right-linear TRSs become 2-left-linear ones.

5. Conclusion

Regular approximation techniques have been successfully used in the context of security protocol analysis. In order to apply them to other applications, this paper proposed an extension of the completion procedure for handling non left-linear rules. Our contributions allow analysing some reachability problems using polynomial steps computing \( A^{(h)} \), rather than automata determinisation steps that are exponential, even in practical cases. Notice that the approach presented only for quadratic rules can be extended to more complex TRSs. We intend to optimise this technique: polynomial is better than exponential but may also lead to huge automata in few steps. We have been implementing the techniques in an efficient rewriting tool like TOM\(^a\) in order to investigate complex systems backward analyses.

Bibliography


\(^a\)http://tom.loria.fr


[21] J. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and


