

Verification of Cryptographic Protocols

Part 3: Tree Automata and Approximation Techniques

Heiko Stamer

University of Kassel
Department of Mathematics/Computer Science
Heinrich-Plett-Straße 40, D-34132 Kassel
stamer@theory.informatik.uni-kassel.de
76F7 3011 329D 27DB 8D7C 3F97 4F58 4EB8 FB2B E14F

Seminar: Theoretische Informatik, December 2005

U N I K A S S E L
V E R S I T Ä T

1 Tree Automata

2 Approximation Techniques

3 Applications

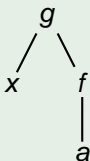
- \mathcal{F} Finite set of **symbols** (ranked alphabet), associated with an arity function $Ar : \mathcal{F} \rightarrow \mathbb{N}_0$
- \mathcal{F}_n Set of symbols with **arity n** , $\mathcal{F}_n = \{f \in \mathcal{F} \mid Ar(f) = n\}$
 \mathcal{F}_0 constants; e.g. written as a, b, c
 \mathcal{F}_n n -ary symbols; e.g. written as $f(\cdot), g(\cdot, \cdot), h(\cdot, \cdot, \cdot)$
- \mathcal{X} Countable set of **variables** where $\mathcal{F} \cap \mathcal{X} = \emptyset$
- $\mathcal{T}(\mathcal{F}, \mathcal{X})$ Set of **terms**; smallest set inductively defined by
- $\mathcal{F}_0 \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$
 - If $n \geq 1$, $f \in \mathcal{F}_n$ and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, then $f(t_1, \dots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.
- $\mathcal{T}(\mathcal{F})$ Set of **ground terms** (terms without variables), $\mathcal{X} = \emptyset$
- $Var(t)$ Set of variables of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$
- linear** A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is **linear**, if each variable from \mathcal{X} occurs at most once in t .

Example ($\mathcal{F} = \{a, f(\cdot), g(\cdot, \cdot)\}$, $\mathcal{X} = \{x, y\}$)

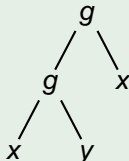
$f(f(a))$ [linear]



$g(x, f(a))$ [linear]



$g(g(x, y), x)$ [non-linear]



A **position** is a word over \mathbb{N} (The ϵ denotes the top-most position).

$Pos(t)$ Set of **positions** of a term t inductively defined by

- $Pos(t) = \{\epsilon\}$, if $t \in \mathcal{X}$ (variables) or $t \in \mathcal{F}_0$ (constants)
- $Pos(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \{ip \mid 1 \leq i \leq n \text{ and } p \in Pos(t_i)\}$

$t|_p$ Subterm of t at position $p \in Pos(t)$

$t[s]_p$ Term obtained by replacing subterm $t|_p$ at position p by term s

Definition

A **substitution** is a function $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$, which can be uniquely extended to an endomorphism $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Definition

A **term rewriting system (TRS)** is a set \mathcal{R} of rewrite rules $l \rightarrow r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, and $\text{Var}(l) \supseteq \text{Var}(r)$.

A rewrite rule is **left-linear** (resp. **right-linear**) if each variable of l (resp. r) occurs only once in l (resp. r). A rule is **linear** if it is left and right-linear. A TRS \mathcal{R} is linear (resp. left-linear, right-linear) if every rewrite rule of \mathcal{R} is linear (resp. left-linear, right-linear).

The TRS \mathcal{R} induces a **rewriting relation** $\rightarrow_{\mathcal{R}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$:

$$t_1 \rightarrow_{\mathcal{R}} t_2 \Leftrightarrow \exists l \rightarrow r \in \mathcal{R}, \exists p \in \text{Pos}(t_1), \exists \sigma \text{ such that } t_1|_p = \sigma(l) \text{ and } t_2 = t_1[\sigma(r)]_p$$

The reflexive transitive closure is denoted as usually by $\rightarrow_{\mathcal{R}}^*$.

Definition

Let $E \subseteq \mathcal{T}(\mathcal{F})$ be a set of ground terms. The **\mathcal{R} -descendants** of E are

$$\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ such that } s \rightarrow_{\mathcal{R}}^* t\}.$$

The extension to a single term $s \in \mathcal{T}(\mathcal{F})$ is given by $\mathcal{R}^*(s) = \mathcal{R}^*({s})$.

$IRR(\mathcal{R})$ Set of **irreducible** terms with respect to the TRS \mathcal{R}

$\mathcal{R}^!(E)$ Set of **\mathcal{R} -normal forms** of E , $\mathcal{R}^!(E) = \mathcal{R}^*(E) \cap IRR(\mathcal{R})$

Additional notations for (tree) automata which recognize terms:

Q Finite set of constants (arity 0) called **states**, $Q \cap \mathcal{F} = \emptyset$

$\mathcal{T}(\mathcal{F} \cup Q)$ Set of **configurations** for (tree) automata

Definition

A **transition** is a rewrite rule $c \rightarrow q$ where $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is a configuration and $q \in \mathcal{Q}$ is a state.

A **normalized transition** is a transition $c \rightarrow q$ where $c = q' \in \mathcal{Q}$ or $c = f(q_1, \dots, q_n)$, $f \in \mathcal{F}_n$, and $q_1, \dots, q_n \in \mathcal{Q}$.

An **ϵ -transition** has the form $q' \rightarrow q$ where q' and q are states.

Definition (bottom-up NFTA)

A **bottom-up non-deterministic finite tree automaton** is a quadruple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$, where

- $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and
- Δ is a set of normalized transitions.

The **move relation** of \mathcal{A} induced by Δ is denoted by $\rightarrow_{\mathcal{A}}$.

The reflexive and transitive closure of $\rightarrow_{\mathcal{A}}$ is denoted by $\rightarrow_{\mathcal{A}}^*$.

A finite tree automaton is **deterministic** (DFTA), if there are no two transitions with the same left-hand side and no ϵ -transition.

Definition

The **tree language recognized by a state q** in \mathcal{A} is

$$\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \rightarrow_{\mathcal{A}}^* q\}.$$

The **tree language recognized by \mathcal{A}** is

$$\mathcal{L}(\mathcal{A}) = \cup_{q \in Q_f} \mathcal{L}(\mathcal{A}, q).$$

A tree language is **regular**, iff it can be recognized by a NFTA.

A state q in \mathcal{A} is a so-called **dead state**, if $\mathcal{L}(\mathcal{A}, q) = \emptyset$.

Example $(\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta), \mathcal{F} = \{f(\cdot), g(\cdot), a\}, \mathcal{Q} = \{q_0, q_1, q_2\}, \mathcal{Q}_f = \{q_0\}, \Delta = \{f(q_0) \rightarrow q_0, g(q_1) \rightarrow q_0, g(q_2) \rightarrow q_2, a \rightarrow q_1\})$

- \mathcal{A} is a DFTA and all transitions in Δ are normalized.
- The term $f(f(g(a))) \in \mathcal{T}(\mathcal{F})$ can be recognized as follows:

$$f(f(g(a))) \xrightarrow{(4)}_{\mathcal{A}} f(f(g(q_1))) \xrightarrow{(2)}_{\mathcal{A}} f(f(q_0)) \xrightarrow{(1)}_{\mathcal{A}} f(q_0) \xrightarrow{(1)}_{\mathcal{A}} q_0$$

- The tree language recognized by \mathcal{A} is

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_0) = \{g(a), f(g(a)), f(f(g(a))), \dots\} = \{f^*(g(a))\}.$$

- The tree language recognized by the state q_1 is $\mathcal{L}(\mathcal{A}, q_1) = \{a\}$.
- Note that $\mathcal{L}(\mathcal{A}, q_2) = \emptyset$, thus q_2 is a dead state.

Basic facts (cf. Comon et al. TATA book)

- If L is recognized by a NFTA with ϵ -transitions, then L can be recognized by a NFTA without ϵ -transitions.
- Let L be a regular tree language. Then there exists a DFTA that accepts L . (exponential blow-up in \mathcal{Q} is possible)
- The class of regular tree languages is closed under union, under complement, and under intersection.
- For each regular tree language there exists an “unique” DFTA with a minimum number of states. (minimization algorithm)
- The uniform membership problem is decidable in linear time for DFTA and in polynomial time for NFTA.
- The emptiness (linear time), finiteness (polynomial time), and equivalence (EXPTIME-complete for NFTA) are decidable.

Reachability Problem for TRS

Instance: Given a TRS \mathcal{R} and two ground terms $s, t \in \mathcal{T}(\mathcal{F})$.

Question: Decide whether $s \rightarrow_{\mathcal{R}}^* t$ holds or not.

If \mathcal{R} is **terminating**, then the problem can be (inefficiently) solved by computing the finite set $\mathcal{R}^*(s)$ and a check whether t is a member.

If \mathcal{R} is **not terminating**, then more sophisticated techniques (e.g. tree automata) can be used to finitely represent the infinite set $\mathcal{R}^*(s)$.

Much work was devoted to characterize $\mathcal{R}^*(E)$ for regular tree languages E and somehow restricted TRS \mathcal{R} , e.g. ground TRS, right-linear monadic TRS, linear semi-monadic TRS, etc.

On the other hand, for a given regular tree language E , the set $\mathcal{R}^*(E)$ is not necessarily regular, even if \mathcal{R} is a confluent and terminating linear TRS. (Gilleron and Tison, 1995)

Approximation (Genet, 1998): Given a NFTA $\mathcal{A}_0 = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_0)$ and a somehow restricted (e.g. left-linear) TRS \mathcal{R} compute a NFTA \mathcal{A}' such that $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ (over-approximation).

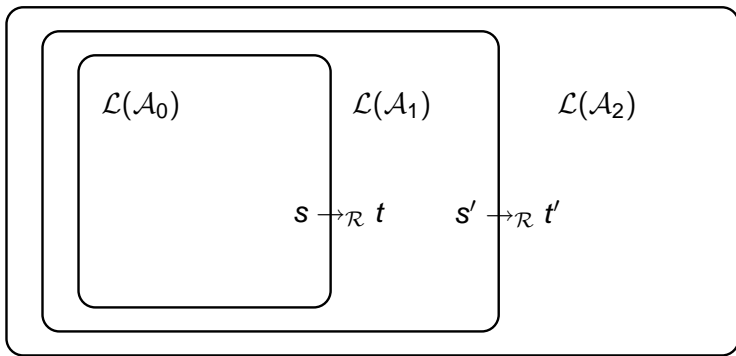
Protocol Verification (Genet and Klay, 2000): Such approximations are used to show that terms recognized by the NFTA \mathcal{A}_{bad} are **not reachable** by rewriting terms of $\mathcal{L}(\mathcal{A}_0)$ with rules from \mathcal{R} , i.e.

$$\forall s \in \mathcal{L}(\mathcal{A}_0) \forall t \in \mathcal{L}(\mathcal{A}_{\text{bad}}) : s \not\rightarrow_{\mathcal{R}}^* t.$$

Obviously, it is enough to show that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \cap \mathcal{L}(\mathcal{A}_{\text{bad}}) = \emptyset$.

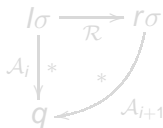
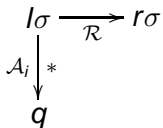
Basic Idea:

- 1 Successively compute tree automata $\mathcal{A}_1, \mathcal{A}_2, \dots$ such that
 - (a) $\forall i \geq 0 : \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$, and
 - (b) if $s \in \mathcal{L}(\mathcal{A}_i)$, $t \notin \mathcal{L}(\mathcal{A}_i)$, and $s \rightarrow_{\mathcal{R}} t$ holds, then $t \in \mathcal{L}(\mathcal{A}_{i+1})$.
- 2 Repeat the previous step until $\mathcal{L}(\mathcal{A}_k) = \mathcal{L}(\mathcal{A}_{k+1})$ for a $k \in \mathbb{N}$.



Completion Step $\mathcal{A}_i \Rightarrow \mathcal{A}_{i+1}$:

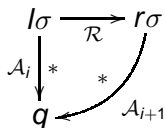
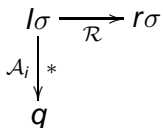
- 1 Finding **critical pairs** between $\rightarrow_{\mathcal{A}_i}$ and $\rightarrow_{\mathcal{R}}$, i.e. instances $(l\sigma, r\sigma)$ such that $l\sigma \rightarrow_{\mathcal{A}_i}^* q$ and $r\sigma \not\rightarrow_{\mathcal{A}_i}^* q$, for a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{Q}$ (match), a rule $l \rightarrow r \in \mathcal{R}$, and a state $q \in \mathcal{Q}$.
- 2 Solve the divergences by adding new transitions $r\sigma \rightarrow q$ to \mathcal{A}_{i+1} (more precisely to Δ_{i+1} , such that $\Delta_i \subset \Delta_{i+1}$ is preserved).



Problem: New transitions $r\sigma \rightarrow q$ are not necessarily normalized.

Completion Step $\mathcal{A}_i \Rightarrow \mathcal{A}_{i+1}$:

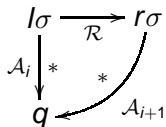
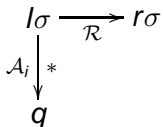
- 1 Finding **critical pairs** between $\rightarrow_{\mathcal{A}_i}$ and $\rightarrow_{\mathcal{R}}$, i.e. instances $(l\sigma, r\sigma)$ such that $l\sigma \rightarrow_{\mathcal{A}_i}^* q$ and $r\sigma \not\rightarrow_{\mathcal{A}_i}^* q$, for a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{Q}$ (match), a rule $l \rightarrow r \in \mathcal{R}$, and a state $q \in \mathcal{Q}$.
- 2 Solve the divergences by adding new transitions $r\sigma \rightarrow q$ to \mathcal{A}_{i+1} (more precisely to Δ_{i+1} , such that $\Delta_i \subset \Delta_{i+1}$ is preserved).



Problem: New transitions $r\sigma \rightarrow q$ are not necessarily normalized.

Completion Step $\mathcal{A}_i \Rightarrow \mathcal{A}_{i+1}$:

- 1 Finding **critical pairs** between $\rightarrow_{\mathcal{A}_i}$ and $\rightarrow_{\mathcal{R}}$, i.e. instances $(l\sigma, r\sigma)$ such that $l\sigma \rightarrow_{\mathcal{A}_i}^* q$ and $r\sigma \not\rightarrow_{\mathcal{A}_i}^* q$, for a substitution $\sigma : \mathcal{X} \rightarrow \mathcal{Q}$ (match), a rule $l \rightarrow r \in \mathcal{R}$, and a state $q \in \mathcal{Q}$.
- 2 Solve the divergences by adding new transitions $r\sigma \rightarrow q$ to \mathcal{A}_{i+1} (more precisely to Δ_{i+1} , such that $\Delta_i \subset \Delta_{i+1}$ is preserved).



Problem: New transitions $r\sigma \rightarrow q$ are not necessarily normalized.

Solution: Normalization by abstracting subterms with states.

Example (Normalize $s \rightarrow q$ where $s = f(g(a), h(q'))$)

- Normalization by adding the new states q_1, q_2, q_3 to \mathcal{Q} :

$$\{a \rightarrow q_1, g(q_1) \rightarrow q_2, h(q') \rightarrow q_3, f(q_2, q_3) \rightarrow q\}$$

- Normalization by merging states, e.g. $q_1 = q_2$ (approximation):

$$\{a \rightarrow q_1, g(q_1) \rightarrow q_1, h(q') \rightarrow q_3, f(q_1, q_3) \rightarrow q\}$$

Represents the regular set $f(g^*(a), h(q')) \rightarrow q$ of non-normalized transitions. Hence, this is a approximation for $s \rightarrow q$.

Definition

Let \mathcal{F} be a set of symbols, and \mathcal{Q} a set of states. An **abstraction function** α maps every normalized configuration into a state, i.e.

$$\alpha : \{f(q_1, \dots, q_n) \mid f \in \mathcal{F}_n \text{ and } q_1, \dots, q_n \in \mathcal{Q}\} \rightarrow \mathcal{Q}.$$

Definition

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

- If $t \in \mathcal{Q}$, then $top_\alpha(t) = t$,
- if $t = f(t_1, \dots, t_n)$, then $top_\alpha(t) = \alpha(f(top_\alpha(t_1), \dots, top_\alpha(t_n)))$.

Definition

Let \mathcal{F} be a set of symbols, \mathcal{Q} a set of states, $s \rightarrow q$ a transition such that $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$, and α an abstraction function. The set $Norm_\alpha(s \rightarrow q)$ of **normalized transitions** is inductively defined by:

- 1 If $s = q$, then $Norm_\alpha(s \rightarrow q) = \emptyset$, and
- 2 if $s \in \mathcal{Q}$ and $s \neq q$, then $Norm_\alpha(s \rightarrow q) = \{s \rightarrow q\}$, and
- 3 if $s = f(t_1, \dots, t_n)$, then

$$Norm_\alpha(s \rightarrow q) = \{f(top_\alpha(t_1), \dots, top_\alpha(t_n)) \rightarrow q\} \cup$$

$$\bigcup_{i=1}^n Norm_\alpha(t_i \rightarrow top_\alpha(t_i)).$$

Example $(\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta), \mathcal{F} = \{f(\cdot), g(\cdot, \cdot), a\}, \mathcal{Q}_f = \{q_0\}, \mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}, \Delta = \{f(q_1) \rightarrow q_0, g(q_1, q_1) \rightarrow q_1, a \rightarrow q_1\})$

- $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{g, a\})$, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_0) = \{f(t) \mid t \in \mathcal{L}(\mathcal{A}, q_1)\}$
- Let $s = f(g(q_1, f(a)))$, and α_1 be the abstraction function given by

$$\{a \mapsto q_4, f(q_4) \mapsto q_3, g(q_1, q_3) \mapsto q_2\}.$$

The normalization of the transition $s \rightarrow q_0$ with abstraction α_1 is

$$\{f(q_2) \rightarrow q_0, g(q_1, q_3) \rightarrow q_2, f(q_4) \rightarrow q_3, a \rightarrow q_4\}.$$

Definition

A **regular language substitution** (\mathcal{Q} -substitution) over an automaton \mathcal{A} (with states \mathcal{Q}) is a function $\sigma : \mathcal{X} \rightarrow \mathcal{Q}$. The definition is expandable to a morphism $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{Q})$ and $\Sigma(\mathcal{Q}, \mathcal{X})$ denotes the set of all regular language substitutions built over \mathcal{Q} and \mathcal{X} .

Definition

Let $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ be a NFTA, \mathcal{R} a TRS, and α an abstraction function. The **one step completed automaton** $\mathcal{C}_{\alpha, \mathcal{R}}(\mathcal{A})$ is a NFTA $(\mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta')$ such that $\mathcal{Q}' = \{q \mid c \rightarrow q \in \Delta'\}$ and

$$\Delta' = \Delta \cup \bigcup_{\substack{l \rightarrow r \in \mathcal{R}, q \in \mathcal{Q} \\ \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}), l\sigma \rightarrow_{\Delta}^* q}} \text{Norm}_{\alpha}(r\sigma \rightarrow q).$$

Definition

Let \mathcal{A} be a NFTA, \mathcal{R} a TRS, and α an abstraction function.

The **tree automata completion** procedure works as follows:

- Starting with the automaton $\mathcal{A}_{\alpha, \mathcal{R}}^0 = \mathcal{A}$,
- it computes $\mathcal{A}_{\alpha, \mathcal{R}}^{i+1} = \mathcal{C}_{\alpha, \mathcal{R}}(\mathcal{A}_{\alpha, \mathcal{R}}^i)$, for $i \in \mathbb{N}$,
- until a fixpoint $\mathcal{A}_{\alpha, \mathcal{R}}^* = \mathcal{A}_{\alpha, \mathcal{R}}^k = \mathcal{A}_{\alpha, \mathcal{R}}^{k+1}$ is reached for some $k \in \mathbb{N}$.

Unfortunately, the NFTA $\mathcal{A}_{\alpha, \mathcal{R}}^*$ does not exist in general. However, it can be computed in many interesting cases, provided that α ensures the termination of the completion procedure.

Definition

Let $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ be a NFTA and \mathcal{R} a TRS. We say that \mathcal{A} and \mathcal{R} satisfy the **left-coherence** condition, if $\forall \tau : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}), \forall l \rightarrow r \in \mathcal{R}, \forall q \in \mathcal{Q}$ the relation $l\tau \rightarrow_{\Delta}^* q$ implies $\exists \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}) : l\tau \rightarrow_{\Delta}^* l\sigma \rightarrow_{\Delta}^* q$.

Lemma

Every **left-linear** TRS \mathcal{R} and every NFTA \mathcal{A} satisfy the left-coherence.

Proof.

In a left-linear TRS, for every LHS l the occurring variables x_1, \dots, x_n are distinct and have a unique position. Since $l\tau \rightarrow_{\Delta}^* q$ holds there exist some ground terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ and some states $q_1, \dots, q_n \in \mathcal{Q}$ such that $\tau = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ and $t_i \rightarrow_{\Delta}^* q_i$ for all $1 \leq i \leq n$. Since all the variables $x_i \in \text{Var}(l)$ are distinct, $\sigma = \{x_1 \mapsto q_1, \dots, x_n \mapsto q_n\}$ is a function and thus it is a valid \mathcal{Q} -substitution such that $l\tau \rightarrow_{\Delta}^* l\sigma \rightarrow_{\Delta}^* q$.



Roughly, the problem with non left-linear rules is the following:

Example $(f(x, x) \rightarrow g(x) \in \mathcal{R}, \{f(q_1, q_1) \rightarrow q_0, f(q_2, q_3) \rightarrow q_0\} \subseteq \Delta)$

- There is only a valid \mathcal{Q} -substitution $\sigma = \{x \mapsto q_1\}$ for matching the rewrite rule on the first transition.
- The semantics of a completion between $f(x, x) \rightarrow g(x)$ and the second transition $f(q_2, q_3) \rightarrow q_0$ would be to find a common language of terms, which are recognized by q_2 and q_3 , i.e. compute a new NFTA $\mathcal{A}' = (\mathcal{F}, \mathcal{Q}', \mathcal{Q}'_f, \Delta')$, $\mathcal{Q} \cap \mathcal{Q}' = \emptyset$ such that $\exists q' \in \mathcal{Q}' : \mathcal{L}(\mathcal{A}', q) = \mathcal{L}(\mathcal{A}, q_2) \cap \mathcal{L}(\mathcal{A}, q_3)$, and add the transitions of \mathcal{A}' to \mathcal{A} including a new transition $g(q) \rightarrow q_0$.
- The non-linearity problem disappears for a DFTA $\bar{\mathcal{A}}$ since $\forall q, q' \in \bar{\mathcal{Q}}$ we have $\mathcal{L}(\bar{\mathcal{A}}, q) \cap \mathcal{L}(\bar{\mathcal{A}}, q') = \emptyset$. However, the determinization is often not practical (exponential blow-up).

Idea (Genet and Tong, 2001): Ensuring determinism only for a subset of states which are to be matched by the non-linear variables of the non-linear rules. (locally deterministic tree automata)

Let \mathcal{A} be an NFTA, $l \rightarrow r$ a rewrite rule over $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\{x_1, \dots, x_m\}$ the set of variables **non-linear** in l , and \mathcal{Y} a set of **fresh** variables distinct from \mathcal{X} . Let $Ren(l)$ be the pair (l', E) where l' denotes the term l with all non-linear variables **renamed** and E a set of **constraints**:

- 1 $Ren(l) = (l, \emptyset)$, if l is either a constant or a variable that does not appear in $\{x_1, \dots, x_m\}$.
- 2 $Ren(l) = (y, \{x = y\})$, if l is a variable $x \in \{x_1, \dots, x_m\}$ and y is a fresh variable of \mathcal{Y}
- 3 $Ren(l) = (f(t'_1, \dots, t'_n), \bigcup_{i=1}^n E_i)$, if $l = f(t_1, \dots, t_n)$, $f \in \mathcal{F}_n$, and $Ren(t_i) = (t'_i, E_i)$, for all $i = 1, \dots, n$.

Definition

A NFTA \mathcal{A} and a TRS \mathcal{R} satisfy the **simplified left-coherence** condition, if for all rules $l \rightarrow r \in \mathcal{R}$ such that $Ren(l) = (l', E)$:

$$\forall (x = y) \in E, \forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}), \forall q, q_x, q_y \in \mathcal{Q} :$$

$$\sigma(x) = q_x \neq q_y = \sigma(y) \quad l' \sigma \rightarrow_{\Delta}^* q, \quad \text{implies } \mathcal{L}(\mathcal{A}, q_x) \cap \mathcal{L}(\mathcal{A}, q_y) = \emptyset.$$

Lemma

The simplified left-coherence implies left-coherence.

Theorem (Genet et al, 2004)

Let \mathcal{A} be a NFTA, \mathcal{R} be a TRS, and α an abstraction function. If \mathcal{R} and $\mathcal{A}_{\alpha, \mathcal{R}}^$ satisfy the left-coherence condition, then $\mathcal{L}(\mathcal{A}_{\alpha, \mathcal{R}}^*) \supseteq \mathcal{R}^*(\mathcal{A})$.*

- Alternative algorithms and proofs for the **regularity of $\mathcal{R}^*(E)$** are obtained using an exact (rather than approximative) completion:
 - Ground TRS: injective abstraction with finite domain
 - Right-linear and monadic TRS: abstraction with empty domain
 - Linear and semi-monadic: injective abstraction with finite domain
 - ...
- Properties like strong non-termination has been used to prove the **deadlock-freeness** for parallel processes.
- (Non-)reachability testing by over-approximation has been used for **theorem proving** and **verification** purposes.

Application to the Needham-Schroeder Public-Key Protocol (NSPK):

Goal: Mutual authentication of two agents, an initiator A and a responder B , over an insecure network.

Protocol: (simplified version without key servers)

N : nonces (fresh random numbers),

K : already distributed keys,

$\{\cdot\}_k$: asymmetric encryption of messages

$$1 \quad A \rightarrow B : \{N_A, A\}_{K_B}$$

$$2 \quad B \rightarrow A : \{N_A, N_B, B\}_{K_A}$$

$$3 \quad A \rightarrow B : \{N_B\}_{K_B}$$

- **Authentication**, i.e. if a honest agent X believes (at the end of step 2. resp 3.) that the message was built by agent Y , then the message was effectively built by Y .
- **Confidentiality**, i.e. nonces and private keys remain confidential.

The protocol avoids the well-known parallel session attack of Lowe.

First, encoding the protocol and the intruder by a TRS \mathcal{R} :

Signature \mathcal{F} and the ground terms $\mathcal{T}(\mathcal{F})$ for representing agents, (encrypted) messages, keys, nonces, and storage (current state):

L_{agt} Set of agent labels (unique identifier for each agent)

$\text{agt}(i)$ denotes the agent whose label is $i \in L_{\text{agt}}$

$\text{mesg}(x, y, c)$ denotes a message whose header refers agent x as emitter, agent y as receiver and whose content is c

$\text{pubkey}(a)$ denotes the public key of agent a

$\text{encr}(k, a, c)$ denotes the result of the encryption of content c performed by agent a with the key k

$N(x, y)$ denotes a nonce generated by agent x for identifying a communication with y

\sqcup denotes an AC binary function used to represent sets (storage), e.g. $x \sqcup (y \sqcup z) =_{\text{AC}} (x \sqcup y) \sqcup z$ will represent the set $\{x, y, z\}$

Starting from a set of initial requests, later the aim is to compute a NFTA recognizing an over-approximation of all sent messages. This approximation also contains some terms signaling either communication requests or established communications:

$\text{goal}(x, y)$ denotes that x requests to open a communication with y .

$\text{c_init}(x, y, z)$ denotes that x believes to have initiated a communication with y , but, in reality x communicates with z .

$\text{c_resp}(y, x, z)$ denotes that y believes to have responded to a communication request from x , but z is the real initiator

Encoding the NSPK protocol into AC rewrite rules:

- LHS: Precondition on the current state (set of received messages and communication requests)
- RHS: Message to be sent and sometimes a term which signals an established communication; update of the current state

Every rule is a **cumulative rule** of the form $l \rightarrow LHS \sqcup r$.

$$\mathcal{X} = \{x, y, z, u, v, x_2, x_3, z_2\}$$

$$A \rightarrow B : \{N_A, A\}_{K_B}$$

$$\text{goal}(x, y) \rightarrow LHS \sqcup \text{mesg}(x, y, \text{encr}(\text{pubkey}(y), x, \langle N(x, y), x \rangle))$$

$\langle \dots \rangle$ denotes a list, i.e. terms constructed by $\text{cons}(\cdot, \cdot)$ and nil .

$$B \rightarrow A : \{N_A, N_B, B\}_{K_A}$$

$$\begin{aligned} \text{mesg}(x, \text{agt}(u), \text{encr}(\text{pubkey}(\text{agt}(u)), z, \langle v, \text{agt}(x_2) \rangle)) \rightarrow \\ LHS \sqcup \text{mesg}(\text{agt}(u), \text{agt}(x_2), \text{encr}(\text{pubkey}(\text{agt}(x_2)), \text{agt}(u), \\ \langle v, N(\text{agt}(u), \text{agt}(x_2)), \text{agt}(u) \rangle)) \end{aligned}$$

$$A \rightarrow B : \{N_B\}_{K_B}$$

$$\begin{aligned} \text{mesg}(x, \text{agt}(y), \text{encr}(\text{pubkey}(\text{agt}(y)), z_2, \langle N(\text{agt}(y), \text{agt}(z)), u, \text{agt}(z) \rangle)) \rightarrow \\ LHS \sqcup \text{mesg}(\text{agt}(y), \text{agt}(z), \text{encr}(\text{pubkey}(\text{agt}(z)), \text{agt}(y), \langle u \rangle)) \\ \sqcup \text{c_init}(\text{agt}(y), \text{agt}(z), z_2) \end{aligned}$$

$A \rightarrow B : \{N_B\}_{K_B}$ Final step: report the communication

$$\text{mesg}(x, \text{agt}(y), \text{encr}(\text{pubkey}(\text{agt}(y)), z_2, \langle N(\text{agt}(y), z) \rangle)) \rightarrow \\ LHS \sqcup \text{c_resp}(\text{agt}(y), z, z_2)$$

Unbounded number of agent labels: $L_{\text{agt}} = \{A, B\} \cup \{s^*(0)\}$

Initial set of terms: $E = \{\text{goal}(\text{agt}(s), \text{agt}(t)) \mid s, t \in L_{\text{agt}}\}$

$$\mathcal{A}_0 = (\mathcal{F}, \{q_A, a_B, q_{\text{int}}, q_{\text{agt}I}, q_{\text{agt}A}, q_{\text{agt}B}, q_{\text{net}}\}, \{q_{\text{net}}\}, \Delta)$$

$$\Delta = \{ 0 \rightarrow q_{\text{int}}, s(q_{\text{int}}) \rightarrow q_{\text{int}}, A \rightarrow q_A, B \rightarrow q_B, \text{agt}(q_{\text{int}}) \rightarrow q_{\text{agt}I}, \\ \text{agt}(q_A) \rightarrow q_{\text{agt}A}, \text{agt}(q_B) \rightarrow q_{\text{agt}B}, q_{\text{net}} \sqcup q_{\text{net}} \rightarrow q_{\text{net}}, \\ \text{goal}(q_{\text{agt}A}, q_{\text{agt}B}) \rightarrow q_{\text{net}}, \dots, \text{goal}(q_{\text{agt}I}, q_{\text{agt}I}) \rightarrow q_{\text{net}} \}$$

$\mathcal{L}(\mathcal{A}_0) = E$ (i.e. all possible communication requests)

Encoding the intruder possibilities into AC rewrite rules:

- Agents with identifier A and B are honest.
- Agents with identifier $s^*(0)$ collaborate with the intruder.
- The intruder is the network, i.e. he knows every message sent on the network.

Disassembling messages:

$$\text{cons}(x, y) \sqcup z \rightarrow LHS \sqcup x$$

$$\text{cons}(x, y) \sqcup z \rightarrow LHS \sqcup y$$

$$\text{mesg}(x, y, z) \sqcup u \rightarrow LHS \sqcup z$$

Decrypting messages of collaborating agents:

$$\text{encr}(\text{pubkey}(\text{agt}(0)), y, z) \sqcup u \rightarrow LHS \sqcup z$$

$$\text{encr}(\text{pubkey}(\text{agt}(s(x))), y, z) \sqcup u \rightarrow LHS \sqcup z$$

The intruders ability to build **new messages** from his knowledge is defined by some tree automaton transitions added to Δ :

Intruder knows the identities and the public keys:

$$\begin{array}{lll} \text{agt}(q_{\text{int}}) \rightarrow q_{\text{net}} & \text{agt}(q_A) \rightarrow q_{\text{net}} & \text{agt}(q_B) \rightarrow q_{\text{net}} \\ \text{pubkey}(q_{\text{agtI}}) \rightarrow q_{\text{net}} & \text{pubkey}(q_{\text{agtA}}) \rightarrow q_{\text{net}} & \text{pubkey}(q_{\text{agtB}}) \rightarrow q_{\text{net}} \end{array}$$

Collaborating agents give the intruder their nonces:

$$N(q_{\text{agtI}}, q_{\text{agtI}}) \rightarrow q_{\text{net}} \quad N(q_{\text{agtI}}, q_{\text{agtA}}) \rightarrow q_{\text{net}} \quad N(q_{\text{agtI}}, q_{\text{agtB}}) \rightarrow q_{\text{net}}$$

Compose messages and lists:

$$\begin{array}{ll} \text{cons}(q_{\text{net}}, q_{\text{net}}) \rightarrow q_{\text{net}} & \text{nil} \rightarrow q_{\text{net}} \\ \text{mesg}(q_{\text{net}}, q_{\text{net}}, q_{\text{net}}) \rightarrow q_{\text{net}} & \text{encr}(q_{\text{net}}, q_{\text{agtI}}, q_{\text{net}}) \rightarrow q_{\text{net}} \end{array}$$

After each completion step $\mathcal{A}_i \Rightarrow \mathcal{A}_{i+1}$ all new messages m obtained by rewriting have to be added as new transitions $m \rightarrow_{\mathcal{A}_{i+1}}^* \mathcal{Q}_{\text{net}}$. This will ensure the progress of the intruders knowledge.

Dealing with the AC storage

Simulate the AC behavior of \sqcup by the following left-linear rules:

$$x \sqcup y \rightarrow y \sqcup x, \quad (x \sqcup y) \sqcup z \rightarrow x \sqcup (y \sqcup z), \quad x \sqcup (y \sqcup z) \rightarrow (x \sqcup y) \sqcup z$$

Summery

The construction of Genet and Klay covers

- an **unbounded number of agents** executing
- an **unbounded number of parallel sessions**.

Approximation function α :

- Merge collaborating agents together
- Collapse together all the messages sent and received by dishonest agents

Tree Automata Completion: Genet and Klay obtained an NFTA $\mathcal{A}_{\alpha, \mathcal{R}}^*$ with about 130 states and 340 transitions.

Confidentiality property: Ensure that the intruder cannot capture a nonce of the form $N(\text{agt}(s), \text{agt}(t))$ where $s, t \in \{A, B\}$, i.e. the intersection between $\mathcal{L}(\mathcal{A}_{\alpha, \mathcal{R}}^*)$ and $\mathcal{L}(\mathcal{A}_{\text{conf}})$ is empty.

Authentication property: Check whether there is no distortion between belief and reality in the communication reports of the honest agents, e.g. $c_init(\text{agt}(A), \text{agt}(B), \text{agt}(s^*(0)))$.
Again, this can be checked easily by computing the intersection between $\mathcal{L}(\mathcal{A}_{\alpha, \mathcal{R}}^*)$ and $\mathcal{L}(\mathcal{A}_{\text{aut}})$.

Implementation: Timbuk/Taml: Completion Engine/Tree Automata Library (Objective CAML), <http://www.irisa.fr/lande/genet/timbuk/>

Related work: (Oehl and Sinclair, 2001–2002)

- Combine Paulson's inductive verification approach with the approximation technique to exploit the strengths of each method
- Implementation: tool for the combined usage with ISABELLE (theorem prover) and Timbuk

Related work: (Oehl et al., 2002)

- Approximation function that can be build automatically
- Approximation function guarantees the termination of completion
- Implementation: OCAML code for usage with Timbuk
- Application to the following protocols: NSSK, NSPK, fixed NSPK, simplified Otway-Rees, Woo-Lam Pi3

Conclusion:

- Tree Automata Completion is a powerful tool for the analysis of security protocols.
- There are many directions for further research.



Thomas Genet.

Decidable approximations of sets of descendants and sets of normal forms.
Proceedings of the 9th International Conference on Rewriting Techniques and Applications, LNCS 1379, 1998.



Thomas Genet and Francis Klay.

Rewriting for Cryptographic Protocol Verification.
Proceedings of the 17th CADE Conference, LNAI 1831, 2000.



Frédéric Oehl, Gérard Cece, Olga Kouchnarenko, and David Sinclair.

Automatic Approximation for the Verification of Cryptographic Protocols.
Proceedings of Formal Aspects of Security, LNCS 2629, 2002.



Guillaume Feuillade, Thomas Genet, and Valérie V.T. Tong.

Reachability Analysis over Term Rewriting Systems.
Journal of Automated Reasoning, 2004.