Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 124 (2005) 97–111
www.elsevier.com/locate/entcs

ACTAS: A System Design for Associative and Commutative Tree Automata Theory

Hitoshi Ohsaki

National Institute of Advanced Industrial Science and Technology (AIST)
&
PRESTO – Japan Science and Technology Agency (JST)
ohsaki@ni.aist.go.jp

Toshinori Takai

CREST – Japan Science and Technology Agency (JST)
takai@ni.aist.go.jp

Abstract

ACTAS is an integrated system for manipulating associative and commutative tree automata (AC-tree automata for short), that has various functions such as for Boolean operations of AC-tree automata, computing rewrite descendants, and solving emptiness and membership problems. In order to deal with high-complexity problems in reasonable time, over- and under-approximation algorithms are also equipped. Such functionality enables us automated verification of safety property in infinite state models, that is helpful in the domain of, e.g. network security, in particular, for security problems of cryptographic protocols allowing an equational property. In runtime of model construction, a tool support for analysis of state space expansion is provided. The intermediate status of the computation is displayed in numerical data table, and also the line graphs are generated. Besides, a graphical user interface of the system provides us a user-friendly environment for handy use.

Key words: term rewriting, AC-tree automata, verification, cryptographic protocols

1 Introduction

Tree automata are the counterpart of finite automata for strings, in the sense that they inherit most of the properties holding for finite automata. It is known that *tree languages* recognized by tree automata are closed under Boolean operations and most of the decidability results are positive [4].

doi:10.1016/j.entcs.2004.07.017

The tree automata framework is useful in dealing with trees (i.e. terms), and several verification techniques based on the tree automata theory have been studied [5,7]. For instance, Kaji *et al.* pointed out in [8] that several important cryptographic protocols can be modeled by *term rewriting systems* (TRS for short, [2]) and tree automata, and moreover, the positive decidability results and closure properties of tree automata allow us to design an automated deduction technique for reasoning about the security problems.

In fact, verification tools for security protocols have been developed by using tree automata framework [1,3]. Genet and Viet Triem Tong provided a tree automata library, called Timbuk [5,6], in which associative and commutative properties of functions symbols are treated by using approximation. Rule-based approaches allowing associativity and commutativity have been also investigated, e.g. in [9].

Let us briefly explain below how to handle the model checking problem for infinite state transition systems in practice by using term rewriting and tree automata. We suppose that a TRS $\mathcal{R}$ over the signature $\mathcal{F}$ specifies the transition relation of a transition system $\mathcal{M}$. We say $\mathcal{M}$ admits the transition step $s \to_{\mathcal{M}} t$ under an initial state space L if (1) $s = C[l\sigma]$ and $t = C[r\sigma]$ for some rewrite rule $l \to r$ in $\mathcal{R}$, context C and substitution σ, and (2) there is a state s_0 in L such that $s_0 \to^*_{\mathcal{M}} s$, where $\to^*_{\mathcal{M}}$ is the reflexive and transitive closure of $\to_{\mathcal{M}}$. One should notice that $\to_{\mathcal{M}} \subseteq \to_{\mathcal{R}}$. Namely, the domain of the system $\mathcal{M}$ is the set of all ground terms over $\mathcal{F}$, and the state space of $\mathcal{M}$ to be verified is the reachable states from L by $\mathcal{R}$. We suppose that the initial state space L can be represented by some tree automaton $\mathcal{A}$, in such a way that $t \in L$ if and only if t is accepted by $\mathcal{A}$. So, in this setting, given a rewrite system and a tree automaton specifying each of $\mathcal{M}$ and L, the reachable state space of a transition system is considered to be defined. In the paper, we denote by $\mathcal{L}(\mathcal{A})$ the set of elements accepted by a tree automata $\mathcal{A}$, and by $[\to^*_{\mathcal{R}}](\mathcal{L}(\mathcal{A}))$ the set of reachable states.

Let P be some subset of the domain of $\mathcal{M}$, that consists of states to which we do not allow $\mathcal{M}$ to admit the transition step from any initial state in L. For instance in the network protocols, P is the set of private information, L is the initial knowledge of the intruder, and $\mathcal{R}$ is the intruder's possible operations. So the information obtainable by the intruder can be represented by $[\to^*_{\mathcal{R}}](L)$, and thus, the intersection of P and $[\to^*_{\mathcal{R}}](L)$ contains a private information that is reachable somehow by the intruder. In other words, the non-emptiness of the intersection indicates that the protocol is *not* secure.

However, the set $[\to^*_{\mathcal{R}}](L)$ of reachable states is not a regular tree language even if L is a regular tree language. Even worse, it is not computable in general. A tree language L is called *regular* if there exists a tree automata $\mathcal{A}$ such that

L is recognized by $\mathcal{A}$. To overcome the above problem there have been several studies, such as: (1) to find a subclass, i.e. sufficient conditions, of $\mathcal{R}$ in which regularity is preserved, and (2) to extend the tree automata framework so that a wider class of tree languages can be handled. Decidable subclasses of such TRS that effectively preserve regularity have been investigated in [16,17].

Regarding the second approach, it is known that regularity is not AC-closed. Precisely, the AC-closure of a regular tree language is no longer regular. A binary function symbol f in a signature $\mathcal{F}$ is *associative and commutative* if the following axioms are assumed:

$$f(x, f(y, z)) = f(f(x, y), z) \qquad f(x, y) = f(y, x)$$

The AC-closure of a tree language L is, given a subset $\mathcal{F}_{\mathsf{AC}}$ of the binary function symbols in $\mathcal{F}$, a set $\{t \mid \exists s \in L.\ s =_{\mathsf{AC}} t\}$. Here $=_{\mathsf{AC}}$ denotes the equivalence relation induced by AC-axioms of all function symbols in $\mathcal{F}_{\mathsf{AC}}$. The above negative observation reveals that for modeling a cryptographic protocol allowing equational property like Diffie-Hellman key exchange protocol, the reachable state space can not be handled by the standard tree automata.

In this research we take the second approach. We proposed in [14] an extension of tree automata, called *equational tree automata*. We also showed in [11,12] that under certain useful equational axioms, e.g. associativity and/or commutativity, tree languages accepted by the equational tree automata are closed under Boolean operations. To this extent, the previous Diffie-Hellman key exchange protocol can be handled, and even the verification process is automatable [13].

The AC-tree automata simulator (ACTAS) is a tool for the computation of tree automata allowing that some of the binary function symbols are associative and commutative. A screen shot of this system is presented in Fig. 1.

The class of AC-tree automata is effectively closed under union and intersection, and the membership and emptiness problems are decidable. In regular case, the emptiness test is solvable in linear time. The decidability result of emptiness problem for *non*-regular case is also positive, however, it is not manageable in the sense of real computation, that can be observed by the fact that the reachability of a Petri-net instance is known to be EXPSPACE-hard and non-regular AC-tree automata are in some sense a generalization of Petri-nets. Therefore we designed in ACTAS over- and under-approximation algorithms, for efficiently computing rewrite descendants $[\rightarrow^*_{\mathcal{R}/\mathsf{AC}}](\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ of given AC-tree automaton $\mathcal{A}/\mathsf{AC}$ and AC-TRS $\mathcal{R}/\mathsf{AC}$.

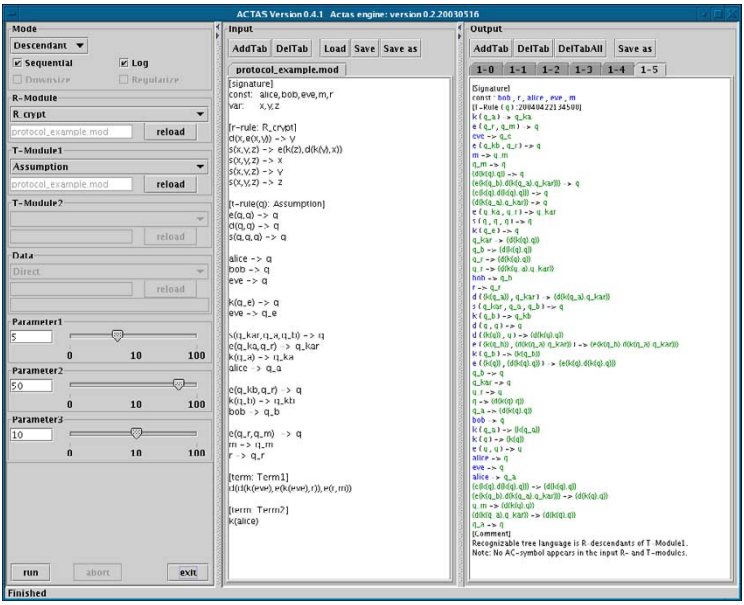

Fig. 1. Control panel of ACTAS

2 AC-Tree Automata

We begin this section by introducing AC-tree automata. We then explain how to operate ACTAS as a tool for manipulating AC-tree automata and even as a tool for supporting automated verification.

A *tree automaton* (TA for short) $\mathcal{A}$ is a 4-tuple $(\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{fin}, \Delta)$, whose components are the *signature* $\mathcal{F}$, i.e. a finite set of function symbols with fixed arities, a finite set $\mathcal{Q}$ of special constant symbols, called *states*, with $\mathcal{F} \cap \mathcal{Q} = \varnothing$, a subset $\mathcal{Q}_{fin}$ of $\mathcal{Q}$ whose elements are called *final states*, and a finite set Δ of *transition rules* in one of the following forms:

$$
\begin{array}{rll}
f(p_1, \ldots, p_n) & \to q_1 & (\textsc{Type}\,1) \\
f(p_1, \ldots, p_n) & \to f(q_1, \ldots, q_n) & (\textsc{Type}\,2) \\
p_1 & \to q_1 & (\textsc{Type}\,3)
\end{array}
$$

such that $f \in \mathcal{F}$ with $\mathsf{arity}(f) = n$ and $p_1, \ldots, p_n, q_1, \ldots, q_n \in \mathcal{Q}$. In Type 2 the root function symbols of the left- and right-hand sides must be the same. Transition rules in Type 3 are called *ϵ-rules* ("epsilon-rules"). A TA $\mathcal{A}$ is called *regular* if Δ consists only of rules in Type 1. Rules of Type 2 are not treated in [4]. Under consideration of equational properties, however, Type 2 is essential in the sense that, e.g. recognizable tree languages of our definition have a bijective correspondence to the word language hierarchy [11]. An efficient

algorithm for the intersection of such AC-tree automata, presented in [14], is also one of the advantages.

A transition move $\rightarrow_{\mathcal{A}}$ is the rewrite relation $\rightarrow_{\Delta}$ by taking Δ as a TRS Δ over the signature $\mathcal{F} \cup \mathcal{Q}$. A ground term t over $\mathcal{F}$ is *accepted* if $t \rightarrow_{\mathcal{A}}^{*} q_f$ for some $q_f \in \mathcal{Q}_{fin}$. The set of terms accepted by a tree automaton $\mathcal{A}$ is denoted as $\mathcal{L}(\mathcal{A})$. A tree language L, that is a subset of all ground terms, is *recognizable* if there is a tree automata $\mathcal{A}$ such that $L = \mathcal{L}(\mathcal{A})$.

An equational tree automaton is a pair of a tree automaton $\mathcal{A}$ and an equational theory $\mathcal{E}$, denoted as $\mathcal{A}/\mathcal{E}$. The transition move is defined by the relation $\rightarrow_{\mathcal{A}}$ modulo $\mathcal{E}$. An AC-tree automaton is an equational tree automaton whose equational theory is the associativity and commutativity axioms for some of the binary function symbols in $\mathcal{F}$. The basic properties of AC-tree automata are stated below:

Theorem 2.1 *[12,14] (1) The class of tree languages recognizable with AC-tree automata are closed under union and intersection. (2) The class of tree languages recognizable with* regular *AC-tree automata are closed under Boolean operations. (3) The membership problem and the emptiness problem for AC-tree automata are decidable.* □

We consider the tree automaton $\mathcal{A}$ with the following transition rules

$$\mathsf{a} \rightarrow q_{\mathsf{a}} \quad \mathsf{b} \rightarrow q_{\mathsf{b}} \quad \mathsf{f}(q_{\mathsf{a}}, q_{\mathsf{b}}) \rightarrow q \quad \mathsf{f}(q, q) \rightarrow q$$

with the final state q. Suppose f is associative and commutative, then the AC-tree automaton $\mathcal{A}/\mathsf{AC}$ accepts such trees t that

$$|t|_{\mathsf{a}} = |t|_{\mathsf{b}}$$

i.e. the number of occurrences of a is the same as the number of occurrences of b in the same tree t. One should notice that the above language is not recognizable with any tree automata.

In ACTAS the above AC-tree automaton $\mathcal{A}/\mathsf{AC}$ is specified as shown below:

```
[signature]
AC: f
const: a,b

[T-rule(q): A]
a -> q_a
b -> q_b
f(q_a,q_b) -> q
f(q,q) -> q
```

The signature of the term model is specified by declaring AC-symbols (e.g. `AC: f`) and also declaring constant function symbols (e.g. `const: a,b`). According to the syntax, the other constant symbols are recognized as state symbols. The tree automaton $\mathcal{A}$ is specified in the second module, named `A`,

```
1: [signature]
2: AC: sum
3: const: 0
4:
5: [R-rule: R]
6: 0 -> s(s(0))
```

```
7: [T-rule(q): A1]
8: 0 -> q_0
9: q_0 -> q
10: s(q_0) -> q_1
11: sum(q_1,q_1) -> q
12: sum(q,q) -> q
```

```
13: [T-rule(q): A2]
14: 0 -> q
15: s(q) -> q_1
16: s(q_1) -> q
17: sum(q_1,q_1) -> q
18: sum(q,q) -> q
```

Fig. 2. AC-rewrite descendant computation in ACTAS

by listing the transition rules. The argument value `q` of `T-rule` is the final state of the tree automaton $\mathcal{A}$.

At the current implementation, ACTAS is equipped with the following functions for (i)–(ii) Boolean operations and (iii) rewrite descendants computation, and the two solvers for (iv) membership problem and (v) emptiness problem:

(i) Given AC-tree automata $\mathcal{A}/\mathsf{AC}$ and $\mathcal{B}/\mathsf{AC}$, construct an AC-tree automaton $\mathcal{C}/\mathsf{AC}$ such that $\mathcal{L}(\mathcal{C}/\mathsf{AC}) = \mathcal{L}(\mathcal{A}/\mathsf{AC}) \cup \mathcal{L}(\mathcal{B}/\mathsf{AC})$.

(ii) Given AC-tree automata $\mathcal{A}/\mathsf{AC}$ and $\mathcal{B}/\mathsf{AC}$, construct an AC-tree automaton $\mathcal{C}/\mathsf{AC}$ such that $\mathcal{L}(\mathcal{C}/\mathsf{AC}) = \mathcal{L}(\mathcal{A}/\mathsf{AC}) \cap \mathcal{L}(\mathcal{B}/\mathsf{AC})$.

(iii) Given an AC-tree automaton $\mathcal{A}/\mathsf{AC}$ and an AC-TRS $\mathcal{R}/\mathsf{AC}$ whose rewrite rules do not contain AC-function symbols, construct AC-tree automaton $\mathcal{C}/\mathsf{AC}$ such that $\mathcal{L}(\mathcal{C}/\mathsf{AC}) = [\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](\mathcal{L}(\mathcal{A}/\mathsf{AC}))$.

(iv) Given an AC-tree automaton $\mathcal{A}/\mathsf{AC}$ and a term t, determine whether $t \in \mathcal{L}(\mathcal{A}/\mathsf{AC})$.

(v) Given an AC-tree automaton $\mathcal{A}/\mathsf{AC}$, determine whether $\mathcal{L}(\mathcal{A}/\mathsf{AC}) = \varnothing$.

The computation results obtained by the operations (i)–(iii) can be re-used for new inputs. For automated verification, the above function (iii) is useful in order to construct models.

We consider the example in Fig. 2. The rewrite system $\mathcal{R}$ consists of the single rewrite rule $\mathtt{0} \rightarrow \mathtt{s}(\mathtt{s}(\mathtt{0}))$. The tree automaton $\mathcal{A}_1$ accepts a tree t if one of the following three conditions is satisfied: (1) $t = \mathtt{0}$, (2) $t = \mathtt{sum}(\mathtt{s}(\mathtt{0}), \mathtt{s}(\mathtt{0}))$, or (3) $t = \mathtt{sum}(t_1, t_2)$ such that t_1, t_2 are accepted by $\mathcal{A}_1/\mathsf{AC}$. Due to the declaration `AC: sum`, the binary symbol `sum` is associative and commutative. Thus, the above language coincides with $L = \{t \mid |t|_{\mathtt{s}(\mathtt{0})} \text{ is even }\}$. This means t consists of the three components `sum`, `0`, $\mathtt{s}(\mathtt{0})$ and the number of occurrences of $\mathtt{s}(\mathtt{0})$ in t is even.

We take $\mathcal{L}(\mathcal{A}_1/\mathsf{AC})\,(= L)$ as an initial state space, then the transition system induced by $\mathcal{R}/\mathsf{AC}$ satisfies that: s is an element in $[\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](\,L\,)$ if and only if $s = \mathtt{sum}\{\,s_1, \ldots, s_n\,\}$ such that $\sum_{i=1}^{n} [\![s_i]\!]$ is even, where $[\![\mathtt{0}]\!] = 0$ and

$[\![\mathtt{s}(t)]\!] = [\![t]\!]+1$. Namely, s is a term generated by $\mathcal{R}/\mathsf{AC}$ with $\mathcal{A}_1/\mathsf{AC}$ whenever the sum of natural numbers occurring in s is even. On the other hand, the tree language $[\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](L)$ can be represented by an AC-tree automaton, for instance, $\mathcal{A}_2/\mathsf{AC}$ defined in Fig. 2. But the advantage of using ACTAS is that we can construct an AC-tree automaton as a result of descendant computation.

One can observe that in constructing AC-tree automata by the fixpoint computation, the bounded computation is often required, because given an AC-TRS $\mathcal{R}/\mathsf{AC}$ and an AC-tree automaton $\mathcal{A}/\mathsf{AC}$, (a) $[\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ is *not* computable in general, and (b) even if $[\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ is computable under a certain condition, it may not be recognizable with AC-tree automata. These two cases correspond to non-terminating computation.

From the above observation, in ACTAS, three parameters are arranged in the control panel. (See the lower left-hand corner in Fig. 1). Parameter 1 restricts the number of the execution of the outermost-loop in the algorithm: By setting Parameter 1 to be n ($\geqslant 1$) in function (iii), we can execute the rewrite descendant computation only of n-loops. In case that Parameter 1 is 0, the program checks whether a given ACTAS code is syntactically correct as no substantial computation occurs.

For computing over- or under-approximated results, we select appropriate positive integers for Parameters 2 and 3. If non-left-linear rewrite rules like $f(x,x) \rightarrow x$ are included in the rewrite system, we need to check, in the algorithm of function (iii), whether tree automata $\mathcal{A}_{p_1} = (\mathcal{F}, \mathcal{Q}, \{p_1\}, \Delta)$ and $\mathcal{A}_{p_2} = (\mathcal{F}, \mathcal{Q}, \{p_2\}, \Delta)$ satisfy

$$\mathcal{L}(\mathcal{A}_{p_1}/\mathsf{AC}) \cap \mathcal{L}(\mathcal{A}_{p_2}/\mathsf{AC}) \neq \varnothing$$

for some p_1, p_2 in $\mathcal{Q}$ with $p_1 \neq p_2$. The values of Parameters 2 and 3 restrict the search depth and width of the decision procedure of the above question. But if Parameters 2 and 3 are the maximum 100, upper-bound limitation is ignored. That in turn results in the exact solution if the computation terminates. Hence, by selecting appropriate positive integers for Parameters 1–3, one can obtain under-approximated results of $[\,\rightarrow^*_{\mathcal{R}/\mathsf{AC}}\,](\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ in reasonable time. On the other hand, by letting Parameters 2 and 3 be 0, it turns out over-approximated results.

3 Cryptographic Protocol Verification

We explain below how to verify network protocols by using ACTAS. In the protocol illustrated in Fig. 3, we write $E(x,y)$ for a message y encrypted by some key x, and $K(x)$ for a principal x's secret key. The goal of this protocol is to send a secret message m from alice to bob without losing the secrecy.

Hence m is encrypted with another secret key (nonce) r, and thus r is also encrypted and transfered to bob. In this network communication, alice first sends a triple of $E(K(\mathsf{alice}), \mathsf{r})$, alice the sender's ID, and bob the receiver's ID. Then server reacts to this request by sending back $E(K(\mathsf{bob}), \mathsf{r})$ to alice. At the final step alice sends the pair of $E(K(\mathsf{bob}), \mathsf{r})$ and $E(\mathsf{r}, \mathsf{m})$, to bob. The latter component is generated by encrypting m by r. The receiver bob can retrieve the clear-text m by decrypting $E(K(\mathsf{bob}), \mathsf{r})$ first.

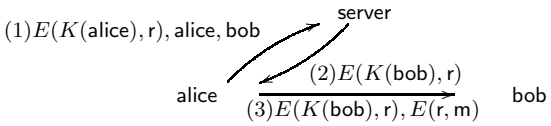

Fig. 3. A cryptographic protocol

Now we assume that an intruder eve has the following 4 abilities:

1. If eve knows x and $E(x, y)$, then eve also knows y,
2. eve knows how to apply encryption and decryption functions E and D, i.e. if eve knows x and y, eve can construct $E(x, y)$ and $D(x, y)$,
3. eve knows its own secret key $K(\mathsf{eve})$ and names of all the other principals, e.g. alice and bob,
4. eve can wiretap the network channels, i.e. eve knows all information flowing in the network of Fig. 3.

To detect the security flaw (otherwise, to ensure the secrecy of the protocol), we verify the protocol by using TRS and tree automata. We first model by a tree automaton the initial knowledge of the intruder eve. We then generate the set of states reachable from the initial knowledge by the following TRS:

$$\mathcal{R}_{\mathsf{crypt}} = \{\, D(x, E(x, y)) \to y \,\}$$

The TRS $\mathcal{R}_{\mathsf{crypt}}$ corresponds to the above intruder's ability 1. The other assumptions 2–4, which are the intruder's initial knowledge and available operations, can be represented by the tree automaton $\mathcal{A}_{\mathsf{initial}}$, that is shown in Fig. 4.

The tree automaton $\mathcal{A}_{\mathsf{initial}}$ accepts a term t if and only if t is obtainable by the intruder without using the encryption-decryption axiom $\mathcal{R}_{\mathsf{crypt}}$. The set $[\,\to^*_{\mathcal{R}_{\mathsf{crypt}}}](\mathcal{L}(\mathcal{A}_{\mathsf{initial}}))$ of reachable states corresponds to the fixpoint of the intruder's knowledge. By computing rewrite descendants (function (iii)), we have a tree automaton $\mathcal{A}_{\mathsf{fixpoint}}$ that satisfies $\mathcal{L}(\mathcal{A}_{\mathsf{fixpoint}}) = [\,\to^*_{\mathcal{R}_{\mathsf{crypt}}}](\mathcal{L}(\mathcal{A}_{\mathsf{initial}}))$ if there exists. Therefore, by solving membership constraint (function (iv))

```
 1:  [signature]
 2:  const:  alice,bob,eve,m,r
 3:  var:    x,y
 4:
 5:  [R-rule: R_crypt]
 6:  d(x,e(x,y)) -> y
 7:
 8:  [T-rule(q): A_initial]
 9:  d(q,q) -> q
10:  e(q,q) -> q
11:
12:  alice -> q
13:  bob -> q
14:  eve -> q
15:  k(q_e) -> q
16:  eve -> q_e
17:
18:  e(q_ka,q_r) -> q
19:  k(q_a) -> q_ka
20:  alice -> q_a
21:
22:  e(q_kb,q_r) -> q
23:  k(q_b) -> q_kb
24:  bob -> q_b
25:
26:  e(q_r,q_m)  -> q
27:  m -> q_m
28:  r -> q_r
```

Fig. 4. Specification code of cryptographic protocol assuming wiretapping only

```
 1:  [signature]
 2:  const:  alice,bob,eve,m,r
 3:  var:    x,y,z
 4:
 5:  [R-rule: R_crypt2]
 6:  d(x,e(x,y)) -> y
 7:  s(x,y,z) -> e(k(z),d(k(y),x))
 8:  s(x,y,z) -> x
 9:  s(x,y,z) -> y
10:  s(x,y,z) -> z
11:
12:  [T-rule(q): A_initial2]
13:  d(q,q) -> q
14:  e(q,q) -> q
15:  s(q,q,q) -> q
16:
17:  alice -> q
18:  bob -> q
19:  eve -> q
20:  k(q_e) -> q
21:  eve -> q_e
22:
23:  s(q_kar,q_a,q_b) -> q
24:  e(q_ka,q_r) -> q_kar
25:  k(q_a) -> q_ka
26:  alice -> q_a
27:
28:  e(q_kb,q_r) -> q
29:  k(q_b) -> q_kb
30:  bob -> q_b
31:
32:  e(q_r,q_m)  -> q
33:  m -> q_m
34:  r -> q_r
```

Fig. 5. Specification code of cryptographic protocol assuming active attack

$\mathsf{m} \in \mathcal{L}(\mathcal{A}_{\mathsf{fixpoint}})$?, it can be determined whether or not the protocol is secure against wiretapping.

Along the similar construction scheme, we can detect that the same protocol is *not* secure against impersonation. The associated tree automaton and TRS is illustrated in Fig. 5, that represents the previous protocol example in which intruder's active attack is assumed. By allowing that every principal (including the intruder eve) sends a request to server, we add the rewrite rule

$$s(x, y, z) \to E(k(z), D(k(y), x))$$

loop	♯(T-rule)	♯(state)	time (sec)
0	18	9	—
1	30	16	3
2	41	16	12
3	44	16	22
4	44	16	32

Fig. 6. The numbers of transition rules and state symbols (loop 0–4)

and the transition rule $s(q,q,q) \to q$ to the previous code. In the left-hand side $s(x,y,z)$ of the rewrite rule, the variables x,y,z are respectively (intended to) assigned to encrypted data, sender's ID and receiver's ID. The result of this rewriting step is a server's reply, and that is hoped to be received by a sender. More precisely, the sender receives an encrypted message $E(K(s_1), D(K(s_2), s_3))$, that is once decrypted with a sender's key at server site and the result is encrypted with a receiver's key. We assume also that eve can decompose any data of the form $s(t_1,t_2,t_3)$, and this situation is represented by the other three rewrite rules.

In the experiment, by using function (iii) of ACTAS with PARAMETER 1 to be 4 or the greater (and the others to be arbitrary positive integers), we obtain an under-approximated result. The numbers of transition rules and state symbols at each loop are shown in Fig. 6. This table together with the line graphs is generated automatically. We can save the displayed data as HTML format files (Fig. 7).

The resulting tree automaton accepts the secret message m, and thus, we know that the protocol is not secure. Actually, the protocol allows the following security flaw: The intruder eve first sends the tuple of $E(K(\mathsf{alice}), \mathsf{r})$, alice, eve to server. These elements are included in eve's initial knowledge due to the assumptions 3 and 4. Then eve obtains $E(K(\mathsf{eve}), \mathsf{r})$ as a response from server. By the assumptions 2 and 3, eve creates $D(D(K(\mathsf{eve}), E(K(\mathsf{eve}), \mathsf{r})), E(\mathsf{r}, \mathsf{m}))$ that gives rise to m, because of $\mathcal{R}_{\mathsf{crypt}}$ that is the assumption 1.

4 Diffie-Hellman Key-Exchange Protocol

The protocol illustrated below is called Diffie-Hellman key-exchange algorithm (e.g. Section 22.1, [15]):

In the figure $H(x,y)$ stands for the composition of data x and y, that is an integer $y^x \pmod p$ where p is given) in the real situation. To simplify the property of H, we assume in this model that H is implemented as $H(x,y) = p^{x+y} \pmod q$ where p,q are given). A secret key of a principal x is denoted by $K(x)$. The goal of this protocol is without losing the secrecy to share a session key by exchanging some data between the initiator (alice in the

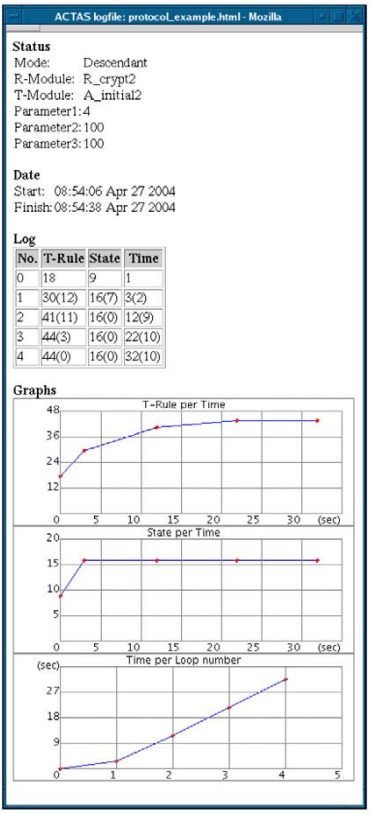

Fig. 7. Tool support for state space analysis

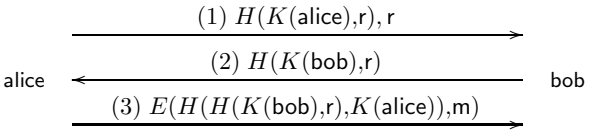

Fig. 8. Diffie-Hellman key-exchange algorithm

example) and the receiver (bob), and then to send from the initiator to receiver a message encrypted by the session key. The protocol consists of the three steps: alice first chooses a number r and sends it to bob together with an integer $H(K(\mathsf{alice}),\mathsf{r})$. We suppose that no one else can retrieve $K(\mathsf{alice})$ only from $H(K(\mathsf{alice}),\mathsf{r})$. At the second step bob returns $H(K(\mathsf{bob}),\mathsf{r})$ to alice. Because of the exponentiation in the implementation of H, one can assume that H is associative and commutative:

$$H(x,H(y,z)) = H(H(x,y),z) \qquad H(x,y) = H(y,x)$$

```
 1: [signature]
 2: AC: h
 3: const: alice,bob,eve,m,r
 4: var: x,y
 5:
 6: [R-rule: R_crypt]
 7: d(x,e(x,y)) -> y
 8:
 9: [T-rule(q): Assumption]
10: d(q,q) -> q
11: e(q,q) -> q
12: h(q,q) -> q
13:
14: alice -> q
15: bob -> q
16: eve -> q
17:
18: k(q_e) -> q
19: eve -> q_e
20: h(q_ka,q_r) -> q
21: k(q_a) -> q_ka
22: alice -> q_a
23: r -> q_r
24:
25: r -> q
26:
27: h(q_kb,q_r) -> q
28: k(q_b) -> q_kb
29: bob -> q_b
30:
31: e(q_kabr,q_m) -> q
32: h(q_kbr,q_ka) -> q_kabr
33: h(q_kb,q_r) -> q_kbr
34: m -> q_m
```

Fig. 9. Diffie-Hellman key-exchange protocol (assuming wiretapping only)

Due to the first two steps, alice can generate $H(H(K(\mathsf{bob}), \mathsf{r}), K(\mathsf{alice}))$ by combining $H(K(\mathsf{bob}), \mathsf{r})$ and $K(\mathsf{alice})$. The latter component is her secret key. Similarly, bob also generates $H(H(K(\mathsf{alice}), \mathsf{r}), K(\mathsf{bob}))$ such that:

$$H(H(K(\mathsf{alice}), \mathsf{r}), K(\mathsf{bob})) =_{\mathsf{AC}} H(H(K(\mathsf{bob}), \mathsf{r}), K(\mathsf{alice})).$$

Hence bob obtain the message m as follows.

$$D(H(H(K(\mathsf{alice}), \mathsf{r}), K(\mathsf{bob})), E(H(H(K(\mathsf{bob}), \mathsf{r}), K(\mathsf{alice})), \mathsf{m})) =_{\mathsf{AC}}$$
$$D(H(H(K(\mathsf{bob}), \mathsf{r}), K(\mathsf{alice})), E(H(H(K(\mathsf{bob}), \mathsf{r}), K(\mathsf{alice})), \mathsf{m})) \to_{\mathcal{R}_{\mathsf{crypt}}} \mathsf{m}$$

The assumption of the protocol can be specified as the ACTAS code like in Fig. 9. In this setting we suppose that (a) the intruder eve can wiretap the network channels, but (b) eve does not actively attack to the protocol. Even if the binary function symbol H is associative and commutative, the AC-rewrite descendants can be computed by using the same algorithm of [16], because H does not appear in rewrite rules. But since the left-hand side of $\mathcal{R}_{\mathsf{crypt}}$ has multiple occurrences of the variable x, the intersection-emptiness problem for AC-tree automata has to be dealt with in the algorithm. The intersection of AC-tree automata can be computed in ACTAS efficiently. However, the resulting AC-tree automata are no longer AC-regular in this efficient construction, and to solve the emptiness problem for non-regular AC-tree automata is EXPSPACE-hard [12].

In fact, when computing the exact solution fully automatically, it is nearly

non-terminating computation. Then there are two choices for this example: th over-approximation algorithm (by setting PARAMETER 2 to be 0) and the under-approximation (by choosing positive integers for PARAMETERS 2 and 3).

Over-approximation. In this case, the result, that is an AC-tree automaton, is some superset of eve's obtainable knowledge. Namely, if the secret message m is not accepted, the secrecy of the protocol is guaranteed. By taking the above input (Fig. 9), we obtain an AC-tree automaton as the output that accepts m. But then, it does not imply security flaw of the protocol. At the current implementation there is no option to refine the over-approximated result.

Under-approximation. In the full automation mode, the security flaw of the protocol is not detected either. This meas that by taking several pairs of positive integers for PARAMETERS 2 and 3, the under-approximated result is obtained, but none of the AC-tree automata that represents some subset of eve's obtainable knowledge accepts the secret message m. Despite of this fact, we have another possibility to handle this protocol example in the AC-tree automata framework. The underlying idea of computing rewrite descendants is, given a tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{fin}, \Delta)$, to find a rewrite rule $l \to r \in \mathcal{R}$ and an assignment $\rho = \{x_i \mapsto q_i \mid 1 \leq i \leq n\}$ with $q, q_i \in \mathcal{Q}$ and $1 \leq i \leq n$ such that $l\rho \to^*_{\mathcal{A}/\mathsf{AC}} q$ but $r\rho \not\to^*_{\mathcal{A}/\mathsf{AC}} q$. Then we add new states and transition rules to $\mathcal{A}$, so that the newly obtained system $\mathcal{A}'/\mathsf{AC}$ satisfies $r\rho \to^*_{\mathcal{A}'/\mathsf{AC}} q$. This process corresponds to one-step rewrite descendant computation for the case that $t \to_{\mathcal{R}} t'$ for some $t \in \mathcal{L}(\mathcal{A}/\mathsf{AC})$ and $t' \notin \mathcal{L}(\mathcal{A}/\mathsf{AC})$ such that $t/o = l\sigma$, $t' = s[r\sigma]_o$, $\rho = \sigma \downarrow_{\mathcal{A}/\mathsf{AC}}$ (i.e. ρ is a normalized substitution of σ with respect to $\mathcal{A}/\mathsf{AC}$). Furthermore, as an exceptional case in the above computation, non-left-linear rewrite rules are treated as follows. For instance, we consider the rule $f(x, x) \to x$ and take two (different) state symbols p_1 and p_2, then we check whether

$$\mathcal{L}(\mathcal{A}(p_1)/\mathsf{AC}) \cap \mathcal{L}(\mathcal{A}(p_2)/\mathsf{AC}) \neq \varnothing.$$

The above $\mathcal{L}(\mathcal{A}(p_i)/\mathsf{AC})$ is a tree language accepted by $\mathcal{A}/\mathsf{AC}$ whose final state $\mathcal{Q}_{fin}$ is replaced by $\{p_i\}$. If the above question is positively solved, we take the different assignment $x \mapsto p_1$ and $x \mapsto p_2$ for the same variable x. Formally, non-linear variable x is replaced by x_1 and x_2, and then define the substitution ρ containing the assignments $x_1 \mapsto p_1$ and $x_2 \mapsto p_2$.

Suppose $\mathcal{A}_0/\mathsf{AC}$ is the AC-tree automaton initially provided. Then the following statement is stated.

Lemma 4.1 *For each state symbols p and q of $\mathcal{A}_0/\mathsf{AC}$, the intersection-*

emptiness of $\mathcal{L}(\mathcal{A}_0(p)/\mathsf{AC})$ *and* $\mathcal{L}(\mathcal{A}_0(q)/\mathsf{AC})$ *is tested by solving finitely many membership problems.*

Proof. We suppose $p \neq \mathsf{q}$, where q is the final state symbol of the AC-tree automaton in Fig. 9. By construction, $\mathcal{L}(\mathcal{A}_0(p)/\mathsf{AC})$ accepts only finitely many trees $t_1, \ldots, t_n$, and $t_1, \ldots, t_n$ are effectively generated, due to König's Lemma e.g. in [10]. So, in this case, the intersection-emptiness is solvable by membership tests. The remaining case is obvious. □

Membership function in the system assists us to perform the above test. In theory, membership problem for regular AC-tree automata is NP-complete (Corollary 4, [12]), but in this example, the size of trees is at most 9. This implies that all the possible combinations of p and q are computable in reasonable time. By using this result, we can examine the following property is correct.

Lemma 4.2 *Let* p_1, p_2, p_3, p_4 *be state symbols of* $\mathcal{A}_0/\mathsf{AC}$ *such that*

$$\mathcal{L}(\mathcal{A}_0(p_1)/\mathsf{AC}) \cap \mathcal{L}(\mathcal{A}_0(p_2)/\mathsf{AC}) \neq \varnothing,$$

then $D(p_1, E(p_2, p_3)) \rightarrow^*_{\mathcal{A}_0/\mathsf{AC}} p_4$ *if and only if* $p_1 = p_2 = p_3 = p_4 = \mathsf{q}$. □

Theorem 4.3 *The tree language* $[\rightarrow^*_{\mathcal{R}_{\mathsf{crypt}}/\mathsf{AC}}](\mathcal{L}(\mathcal{A}_0/\mathsf{AC}))$ *is recognizable with* $\mathcal{A}_0/\mathsf{AC}$. □

Therefore $\mathcal{A}_0/\mathsf{AC}$ is already the fixpoint, namely, the protocol is secure against wiretapping, because m is not in eve's initial knowledge $\mathcal{L}(\mathcal{A}_0/\mathsf{AC})$.

Regarding the active attack by assuming impersonation, the security flaw of the protocol is noted in [15]. For instance, this protocol allows the following attack:

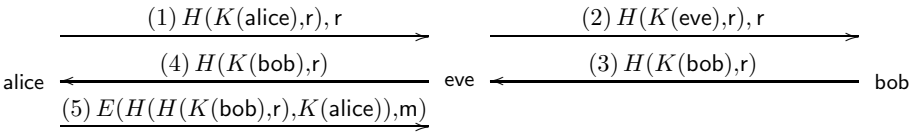

Fig. 10. Man-in-the-middle attack in Diffie-Hellman key exchange protocol

Acknowledgement

The authors thank the three anonymous referees for their numerous comments and suggestions to improve the early version of the paper.

References

[1] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò and L. Vigneron: *The AVISS Security Protocol Analysis Tool*, Proc. of 14th CAV, Copenhagen (Denmark), LNCS 2404, pp. 349–353, Springer-Verlag, 2002.

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

[3] Y. Chevalier and L. Vigneron: *Automated Unbounded Verification of Security Protocols*, Proc. of 14th CAV, Copenhagen (Denmark), LNCS 2404, pp. 324–337, Springer-Verlag, 2002.

[4] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi: *Tree Automata Techniques and Applications*, 2002. Draft available from http://l3ux02.univ-lille3.fr/tata/.

[5] T. Genet and F. Klay: *Rewriting for Cryptographic Protocol Verification*, Proc. of 17th CADE, Pittsburgh (PA), LNCS 1831, pp. 271–290, Springer-Verlag, 2000.

[6] T. Genet and V. Viet Triem Tong: *Reachability Analysis of Term Rewriting Systems with Timbuk*, Proc. of 8th LPAR, Havana (Cuba), LNAI 2250, pp. 691–702, Springer-Verlag, 2001.

[7] H. Hosoya, J. Vouillon and B.C. Pierce: *Regular Expression Types for XML*, Proc. of 5th ICFP, Montreal (Canada), SIGPLAN Notices 35(9), pp. 11–22, ACM Press, 2000.

[8] Y. Kaji, T. Fujiwara and T. Kasami: *Solving a Unification Problem Under Constrained Substitutions Using Tree Automata*, Journal of Symbolic Computation 23, pp. 79–117, 1997.

[9] José Meseguer: *Software Specification and Verification in Rewriting Logic*, Proc. of NATO Advanced Study Institute on Models, Algebras and Logic of Engineering Software, Computer and Systems Sciences 191, pp. 133–193, IOS Press, 2003

[10] Y.N. Moschovakis: *Notes on Set Theory*, Undergraduate Texts in Mathematics, Springer-Verlag, 1994.

[11] H. Ohsaki, H. Seki and T. Takai: *Recognizing Boolean Closed A-Tree Languages with Membership Conditional Rewriting Mechanism*, Proc. of 14th RTA, Valencia (Spain), LNCS 2706, pp. 483–498, Springer-Verlag, 2003.

[12] H. Ohsaki and T. Takai: *Decidability and Closure Properties of Equational Tree Languages*, Proc. of 13th RTA, Copenhagen (Denmark), LNCS 2378, pp. 114–128, Springer-Verlag, 2002.

[13] H. Ohsaki and T. Takai: *A Tree Automata Theory for Unification Modulo Equational Rewriting*, Proc. of 16th UNIF, Copenhagen (Denmark), 2002.
Draft available from http://staff.aist.go.jp/hitoshi.ohsaki/.

[14] H. Ohsaki: Beyond Regularity: *Equational Tree Automata for Associative and Commutative Theories*, Proc. of 15th CSL, Paris (France), LNCS 2142, pp. 539–553, Springer-Verlag, 2001.

[15] B. Schneier: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition, John Wiley & Sons, 1996.

[16] T. Takai, Y. Kaji and H. Seki: *Right-Linear Finitely Path overlapping Term Rewriting Systems Effectively Preserve Recognizability*, Proc. of 11th RTA, Norwich (UK), LNCS 1833, pp. 246–260, 2000.

[17] T. Takai, H. Seki, Y. Fujinaka and Y. Kaji: *Layered Transducing Term Rewriting System and Its Recognizability Preserving Property*, IEICE Transactions on Information and Systems E86-D(2), pp. 285–295, 2003, Information about IEICE Transactions is found at http://www.ieice.org/.