ACTAS: Associative and Commutative Tree Automata Simulator

Toshinori Takai
CREST - JST
takai@ni.aist.go.jp

Hitoshi Ohsaki
AIST & PRESTO - JST
ohsaki@ni.aist.go.jp

Abstract

ACTAS is a tool set manipulating associative and commutative tree automata (AC-tree automata for short). The system is equipped with various functions for Boolean operations and rewrite descendant computation. This functionality allows to verify safety property in infinite state models, which is helpful in the domain of network security, in particular, for security problems of cryptographic protocols allowing an equational property, e.g. Diffie-Hellman key exchange protocol. Besides, a graphical user interface of the system provides us a user-friendly environment for handy use.

1 Introduction

Tree automata are the counterpart of string automata, in the sense that it inherits most of properties holding for finite automata, e.g. closure under Boolean operations and positive decidability results. The tree automata framework is useful in dealing with trees (i.e. terms), and several verification techniques based on the tree automata framework have been developed. For instance, Kaji *et. al* presented in [3] that some important cryptographic protocols are modeled by *term rewriting systems* (TRS, explained in the next section) and tree automata, and moreover, the positive decidability results and closure properties of tree automata allow to design an automated deduction technique for reasoning about the security problems.

We explain below the idea how to model infinite state transition systems in practice by combining the two concepts, tree automata and term rewriting. Given a TRS $\mathcal{R}$ over the signature $\mathcal{F}$, a (possible) transition relation of a system $M_\mathcal{R}$ is defined by a *rewrite relation* $\to_\mathcal{R}$ of $\mathcal{R}$. A transition system allows the transition step $s \to_\mathcal{R} t$ 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 $s_0 \to^*_\mathcal{R} s$, where $\to^*_\mathcal{R}$ is the reflexive and transitive closure of $\to_\mathcal{R}$. So the reachable states from L are the domain of $M_\mathcal{R}$ to be verified. We suppose the initial state space L is represented by some tree automaton $\mathcal{A}$, in such a way that L is a set of trees accepted by $\mathcal{A}$. The set of reachable states is denoted by $(\to^*_\mathcal{R})(\mathcal{L}(\mathcal{A}))$. In other words, given a rewrite system and a tree automaton, the reachable state space of a transition system in consideration can be defined. However, since the set $(\to^*_\mathcal{R})(\mathcal{L}(\mathcal{A}))$ is not regular in general, decidable subclasses of TRS which effectively preserves regularity, i.e. the set $(\to^*_\mathcal{R})(\mathcal{L}(\mathcal{A}))$ is recognized by some tree automaton, have been studied [8, 9]. Unfortunately, it is known that regular tree languages, which is the class of tree automata, are not AC-closed, that means; the AC-closure of tree language accepted by a tree automata is *not* representable with any tree automaton. This 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 in the standard tree automata framework.

To cope with this problem, we proposed in [4] an extension of tree automata, called *equational tree automata*. We also showed in [6, 7] 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. Moreover, in this extended framework the previous Diffie-Hellman key exchange protocol can be dealt with, even more the verification process is automatable [5].

A binary function symbol f in the signature $\mathcal{F}$ is *associative and commutative* if the following axioms are assumed for f:

$$\begin{aligned} f(x, f(y, z)) &= f(f(x, y), z) \\ f(x, y) &= f(y, x) \end{aligned}$$

The AC-tree automata simulator (ACTAS) is a tool for tree automata computation but allowing some function symbols are associative and commutative. See Figure 1 for a screen shot of this system. In fact, 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.

However, although the decidability result of emptiness

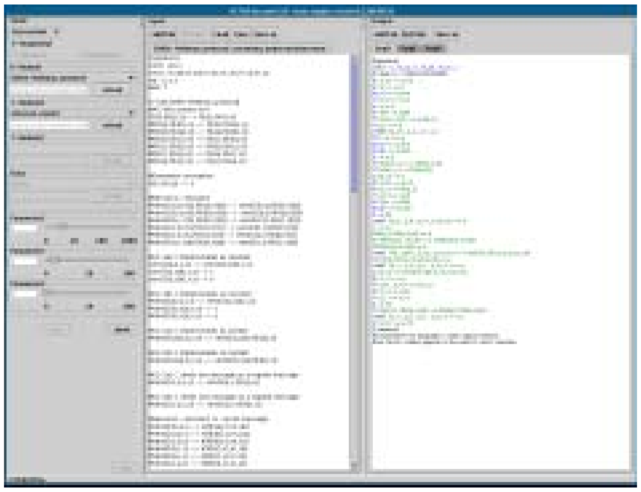

Figure 1. Control panel of ACTAS

problem for *non*-regular case is positive, it is not manageable in real computation. This observation is obtained by the fact that the reachability of a Petri-net instance is known to be EXPSPACE-hard and AC-tree automata are in some sense a generalization of Petri-nets. Hence, we designed in ACTAS under- and over-approximation algorithms, for efficiently computing rewrite descendants $(\to^*_{\mathcal{R}})(\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ of a given AC-tree automaton $\mathcal{A}/\mathsf{AC}$ and TRS $\mathcal{R}$. At the current stage, the prototyping engine equipped with the above functionality has still room to be modified and optimized.

In the following sections, we briefly introduce AC-tree automata and term rewriting, and then we explain how to operate ACTAS as a tool manipulating AC-tree automata and even as a tool supporting automated verification.

2 AC-tree automata

Given the signature $\mathcal{F}$ together with a set $\mathcal{V}$ of variables, a *term rewriting system* $\mathcal{R}$ is a finite set of pairs $l \to r$ of terms l, r over $\mathcal{F}$ and $\mathcal{V}$. Elements in $\mathcal{R}$ are called rewrite rules. A *rewrite relation* $\to_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is the smallest relation of $\mathcal{R}$ closed under substitutions and contexts. We write $\to^*_{\mathcal{R}}$ for the reflexive and transitive closure of $\to_{\mathcal{R}}$. A *tree automaton* consists mainly of 3 components: $\mathcal{Q}$ a finite set of state symbols such that $\mathcal{F} \cap \mathcal{Q} = \emptyset$, $\mathcal{Q}_f$ $(\subseteq \mathcal{Q})$ is a set of final state symbols, and Δ is a finite set of transition rules with one of the following shapes:

$$\begin{array}{rcll} f(p_1, \dots, p_n) & \to & q_1 & \qquad (\textsc{Type}1) \\ f(p_1, \dots, p_n) & \to & f(q_1, \dots, q_n) & \qquad (\textsc{Type}2) \end{array}$$

for a function symbol f and state symbols $p_1, \dots, p_n$, $q_1, \dots, q_n$. In TYPE2 the root symbols in the left- and right-hand sides must be the same. We note that in the literature [2] one can find the form $p \to q$ of transition rules instead of TYPE2. Under consideration of equational properties, our definition are more beneficial, e.g. for the tree language hierarchy [7]. Efficient algorithm for the intersection of AC-tree automata, presented in [4], are also one of the advantages.

The transition move $\to_{\mathcal{A}}$ is defined as the rewrite relation of a TRS Δ. A ground term t over $\mathcal{F}$ is *accepted* if $t \to^*_{\mathcal{A}} q_f$ for some $q_f \in \mathcal{Q}_f$. 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 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 property $\mathcal{E}$ for some function symbols, denoted as $\mathcal{A}/\mathcal{E}$, and the transition move is defined by the relation $\to_{\mathcal{A}}$ modulo $\mathcal{E}$. An *AC-tree automaton* is an equational tree automaton whose equational property is associativity and commutativity of some binary function symbols in $\mathcal{F}$. For example, consider a tree automaton $\mathcal{A}$ whose transition rules are $\{a \to q_a, b \to q_b, f(q_a, q_b) \to q, f(q, q) \to q\}$ with a final state q. Suppose f is associative and commutative, then the AC-tree automaton $\mathcal{A}/\mathsf{AC}$ accepts such trees t that

$$|t|_a = |t|_b$$

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

```
1: [signature]
2: AC: f
3: const: a,b
4:
5: [t-rule(q): A]
6: a -> q_a
7: b -> q_b
8: f(q_a,q_b) -> q
9: f(q,q) -> q
```

Figure 2. AC-tree automaton in ACTAS **syntax**

In ACTAS the above example $\mathcal{A}/\mathsf{AC}$ is specified as shown in Figure 2. The signature is specified by declaring AC-symbols and constant function symbols. The other constant symbols are recognized as state symbols. The tree automaton $\mathcal{A}$ is specified from the 5th line in the module named A, by listing the transition rules. The argument `q` of `t-rule` represents the final state.

In the current version, ACTAS is equipped with the following functions for Boolean operations (1)–(2) and rewrite descendants computation (3), together with two solvers for membership and emptiness problems (4)–(5):

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

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

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

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

(5) Given an AC-tree automaton $\mathcal{A}/\mathsf{AC}$, determine if $\mathcal{L}(\mathcal{A}/\mathsf{AC}) = \emptyset$,

In the system, one can re-use the results obtained by the operations (1) and (2), as new inputs. For computing approximated result of (3), PARAMETERS 1–3 are arranged in the control panel. This approximation is required by the fact that (a) the language $(\to^*_{\mathcal{R}})(\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ is *not* computable in general, and (b) even if $(\to^*_{\mathcal{R}})(\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ is computable in theory, it may not be (effectively) recognizable with AC-tree automata. Roughly speaking, by selecting appropriate positive integers for these PARAMETERS 1–3, one can obtain an under-approximated result of the language $(\to^*_{\mathcal{R}})(\mathcal{L}(\mathcal{A}/\mathsf{AC}))$ in reasonable time. On the other hand, PARAMETERS 2 to be 0 turns out an over-approximated result.

3 Cryptographic protocol verification

We present an example of the verification, by using ACTAS, of a network protocol. For simplicity, we do not assume AC-axioms in the example.

In the protocol illustrated in Figure. 3, we denote $E(x, y)$ to be a message y encrypted by a key x, and $K(x)$ to be a principal x's private 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 a secret key (nonce) r, and thus r is also encrypted and transfered to Bob. In this network communication, first Alice sends a tuple of $E(K(\mathsf{Alice}), \mathsf{r})$, Alice the sender's ID, and Bob the receiver's ID. Then Key Server reacts to this request, by sending $E(K(\mathsf{Bob}), \mathsf{r})$ back 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.

Now we assume the following abilities for an intruder Eve.

$$\begin{tikzcd}
& & \text{Key Server} \arrow[dll, bend left, "(2)E(K(\text{Bob}),\text{r})"] \\
\text{Alice} \arrow[urr, bend left, "(1)E(K(\text{Alice}),\text{r}),\text{Alice},\text{Bob}"] \arrow[rrr, "(3)E(K(\text{Bob}),\text{r}),E(\text{r},\text{m})"'] & & & \text{Bob}
\end{tikzcd}$$

Figure 3. A cryptographic protocol

1. Eve knows how to decrypt a message, i.e. if Eve knows x and $E(x, y)$, then Eve knows y.

2. Eve applies 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 private key $K(\mathsf{Eve})$, principles ID's Alice, Bob and Eve.

4. Eve wiretaps the network channel, i.e. Eve knows all data flowing in the network of Figure 3.

To detect the security flaw or to ensure the secrecy of the protocol, first we model by a tree automaton the initial knowledge of the intruder Eve, and then we test if the set of states reachable from the initial knowledge by means of the following TRS, which specifies the above intruder's ability 1, contains secret information.

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

For the other assumptions 2–4, which corresponds to the intruder's initial knowledge, can be represented by the following tree automaton $\mathcal{A}_{\mathit{initial}}$ with the final state q:

$$\begin{array}{l}
\left.\begin{array}{lll} E(q,q) \to q & D(q,q) \to q & \end{array}\right\} \cdots 2 \\
\left.\begin{array}{lll} \mathsf{Alice} \to q & \mathsf{Bob} \to q & \mathsf{Eve} \to q \\ K(q_{\mathsf{Eve}}) \to q & \mathsf{Eve} \to q_{\mathsf{Eve}} & \end{array}\right\} \cdots 3 \\
\left.\begin{array}{lll} E(q_r, q_{\mathsf{m}}) \to q & \mathsf{m} \to q_{\mathsf{m}} & \mathsf{r} \to q_{\mathsf{r}} \\ E(q_{K(\mathsf{Alice})}, q_{\mathsf{r}}) \to q & & \\ K(q_{\mathsf{Alice}}) \to q_{K(\mathsf{Alice})} & \mathsf{Alice} \to q_{\mathsf{Alice}} & \\ E(q_{K(\mathsf{Bob})}, q_{\mathsf{r}}) \to q & & \\ K(q_{\mathsf{Bob}}) \to q_{K(\mathsf{Bob})} & \mathsf{Bob} \to q_{\mathsf{Bob}} & \end{array}\right\} \cdots 4
\end{array}$$

The set of reachable state $(\to^*_{\mathcal{R}_{\mathit{crypt}}})(\mathcal{L}(\mathcal{A}_{\mathit{initial}}))$ corresponds to the final knowledge of the intruder. Using the function (3) in ACTAS, it can be computed as a tree automaton $\mathcal{A}_{\mathit{final}}$. Therefore, by solving membership constraint $m \in \mathcal{L}(\mathcal{A}_{\mathit{final}})$?, it can be concluded that the protocol is secure against wiretapping.

Along the similar construction scheme, we have detected that the same protocol is *not* secure against impersonation. The detailed ACTAS instruction set, a tree automaton and a TRS, is noted in the appendix.

References

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

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

[3] Y. Kaji, T. Fujiwara and T. Kasami: *Solving a Unification Problem under Constrained Substitutions Using Tree Automata*, JSC 23, pp. 79–117, 1997.

[4] H. Ohsaki: Beyond Regularity: *Equational Tree Automata for Associative and Commutative Theories*, Proc. of CSL2001, Paris (France), LNCS 2142, pp. 539-553.

[5] H. Ohsaki and T. Takai: *A Tree Automata Theory for Unification Modulo Equational Rewriting*, Proc. of UNIF2002, Copenhagen (Denmark), 2002.

[6] H. Ohsaki and T. Takai: *Decidability and Closure Properties of Equational Tree Languages*, Proc. of RTA2002, Copenhagen (Denmark), LNCS 2378, pp. 114–128.

[7] H. Ohsaki, H. Seki and T. Takai: *Recognizing Boolean Closed A-Tree Languages with Membership Conditional Rewriting Mechanism*, Proc. of RTA2003, Valencia (Spain), LNCS 2706, pp. 483–498.

[8] T. Takai, Y. Kaji and H. Seki: *Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability*, Scienticae Mathematicae Japonicae. To appear.

[9] 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. (See `http://www.ieice.org` for the details about IEICE.)

A ACTAS specification for active attack

In Figure 4 we show the specification of the cryptographic protocol (Section 3) assuming intruder's active attack. In this setting, every principal including the intruder Eve can send a request to Key Server. This assumption is represented by adding the rewrite rule `s(x,y,z) -> e(k(z),d(k(y),x))` and the transition rule `s(q,q,q) -> q` to the previous specification. In the left-hand side `s(x,y,z)` of the rewrite rule, each of variables `x`, `y`, `z` is expected to be assigned to encrypted data, sender's ID and receiver's ID, respectively. The result of this rewriting step is a server's reply, that is (assumed 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 Key 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.

```
[signature]
const:  a,b,c,m,r
var:    x,y,z

[r-rule: Rewriting_system]
d(x,e(x,y)) -> y
s(x,y,z) -> e(k(z),d(k(y),x))
s(x,y,z) -> x
s(x,y,z) -> y
s(x,y,z) -> z

[t-rule(q): Initial_knowlegde]
d(q,q) -> q
e(q,q) -> q
s(q,q,q) -> q

a -> q
b -> q
c -> q
k(q_c) -> q

a -> q_a
b -> q_b
c -> q_c
k(q_a) -> q_ka
k(q_b) -> q_kb
k(q_c) -> q_kc

m -> q_m
r -> q_r
e(q_r,q_m)  -> q
e(q_ka,q_r) -> q
e(q_kb,q_r) -> q
s(q_kar,q_a,q_b) -> q
e(q_ka,q_r) -> q_kar
```

Figure 4. Specification code of cryptographic protocol assuming active attack

In real computation, by using Function (3) of ACTAS with PARAMETER1 to be 4 or more (and the others to be arbitrary positive integers), we obtain an under-approximated result, that represents by a tree automaton a subset of reachable states. The tree automaton accepts m, and thus, it can be concluded that the protocol is not secure. Actually, the protocol allows the following security flaw: First Eve sends the tuple of $E(K(\mathsf{Alice}), \mathsf{r})$, Alice, Eve to Key Server. These three elements are included in Eve's initial knowledge due to Assumptions 3–4. Then, Eve receives $E(K(\mathsf{Eve}), \mathsf{r})$ as a response from Key Server. By Assumptions 2–3, Eve can construct $D(D(K(\mathsf{Eve}), E(K(\mathsf{Eve}), \mathsf{r})), E(\mathsf{r}, \mathsf{m}))$ that results in m because of $\mathcal{R}_{crypt}$ (Assumption 1).