A Verification Technique Using Term Rewriting Systems and Abstract Interpretation *

Toshinori Takai

Japan Science and Technology Corporation, CREST
t-takai@aist.go.jp

Abstract. Verifying the safety property of a transition system given by a term rewriting system is an undecidable problem. In this paper, we give an abstraction for the problem which is automatically generated from a given TRS by using abstract interpretation. Then we show that there are some cases in which the problem can be decided. Also we show a new decidable subclass of term rewriting systems which effectively preserves recognizability.

1 Introduction

Term rewriting systems (TRSs for short) can be used for modeling infinite states transition systems, e.g. cryptographic protocols, and some verification techniques for TRSs are proposed. Because of the universal computational power of TRSs, even the safety property for a TRS is undecidable. There are two approaches:

1. to find a decidable sufficient condition [9, 10] and
2. to give an abstraction [5, 6].

If a given TRS satisfies some condition of the first approach, then we can automatically verify, for example, safety property. However, some simple TRSs does not satisfy any of the conditions. On the other hands, to give an abstraction, one have to analyze an instance of the problem and give a special abstraction. In this paper, we give an abstraction for the problem which is automatically generated by using abstract interpretation[4, 7]. We also propose a new decidable sufficient condition for a verification problem to be decidable.

We use the usual notions of *a (ground, linear) term, a position, a context, a TRS, a rewrite relation, a (recognizable) tree language, a tree automaton, etc.* See [1, 3] for details. In the following, we only deal with linear TRSs. For a signature Σ and a set of variables $\mathcal{V}$, $\mathcal{T}(\Sigma, \mathcal{V})$ denotes the set of all terms constructed from Σ and $\mathcal{V}$ and $\mathcal{T}(\Sigma)$ denotes the set of all ground terms constructed from Σ. In this paper, we use $f, g, h, ...$ as function symbols, $x, y, z, ...$ as variables and $a, b, c, ...$ as constant symbols. For a term t, let $\mathcal{P}os(t)$ be the set of all positions of t and $\mathcal{P}os_{\mathcal{V}}(t) = \{p \in \mathcal{P}os(t) \mid t/p \in \mathcal{V}\}$ where t/p is the subterm

* This work is done as a part of a CREST project of Japan Science and Technology Corporation.

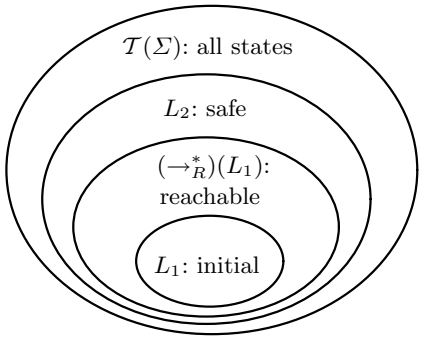

Fig. 1. A verification problem

of t at p. A TRS constructed from terms in $\mathcal{T}(\Sigma, \mathcal{V})$ will be said as *a TRS on* $\mathcal{T}(\Sigma, \mathcal{V})$. For a TRS $\mathcal{R}$, $\rightarrow_{\mathcal{R}}$ denotes the rewrite relation induced from $\mathcal{R}$ and its reflexive and transitive closure is denoted by $\rightarrow_{\mathcal{R}}^*$. For a tree language L_1 and a TRS $\mathcal{R}$, the image of L_1 by the relation $\rightarrow_{\mathcal{R}}^*$ is denoted by $(\rightarrow_{\mathcal{R}}^*)(L_1)$, i.e. $(\rightarrow_{\mathcal{R}}^*)(L_1) = \{t \mid s \rightarrow_{\mathcal{R}}^* t,\ s \in L_1\}$. A tree automaton is a 4-tuple $(\Sigma, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ where Σ is a signature, $\mathcal{Q}$ is a set of states, $\mathcal{Q}_f$ is a set of final states and Δ is a set of transition rule. Transition rules are given as rewrite rules and the transition relation is defined as the rewrite relation of transition rules as in [3]. For a TA $\mathcal{A}$, let $\mathcal{L}(\mathcal{A})$ denote the accepting language of $\mathcal{A}$.

2 Verification Problem

In this section, we define the verification problem which we deal with throughout this paper. The problem is defined as follows.

Problem 1. For given a signature Σ, a set $\mathcal{V}$ of variables, a TRS $\mathcal{R}$ on $\mathcal{T}(\Sigma, \mathcal{V})$ and two recognizable tree languages $L_1, L_2 \subseteq \mathcal{T}(\Sigma)$, decide whether $(\rightarrow_{\mathcal{R}}^*)(L_1) \subseteq L_2$ or not. □

Usually each recognizable tree language is given by a tree automaton. Problem 1 can be regarded as a verification problem as follows. For a TRS $\mathcal{R}$ on $\mathcal{T}(\Sigma, \mathcal{V})$, We can consider a transition system whose states is $\mathcal{T}(\Sigma)$ and the transition relation is defined by the rewrite relation by $\mathcal{R}$. When we regard terms (states) in L_1 as the initial states, $(\rightarrow_{\mathcal{R}}^*)(L_1)$ means the set of all reachable states of the transition system. Once a safety property or an invariant is given by the set L_2, we can see that the transition system is safe if and only if $(\rightarrow_{\mathcal{R}}^*)(L_1) \subseteq L_2$. The situation mentioned above is shown in Fig. 1.

Problem 1 is undecidable; this fact can easily be seen from the fact that the reachability problem of term rewriting systems is undecidable. One sufficient condition to be decidable for the problem is that *a TRS* $\mathcal{R}$ *effectively preserves recognizability*, i.e. for any tree automaton $\mathcal{A}$, we can compute $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = (\rightarrow_{\mathcal{R}}^*)(\mathcal{L}(TA))$. In Problem 1, if $(\rightarrow_{\mathcal{R}}^*)(L_1)$ is recognizable, we can

decide $(\to_{\mathcal{R}}^*)(L_1) \subseteq L_2$ according to the property of recognizable tree languages. Since it is undecidable whether a given TRS effectively preserves recognizability or not, some decidable subclasses have been proposed [9, 10].

3 Abstract Interpretation

Abstract interpretation[4, 7] is a typical method to give abstraction for program domains and can deal with the following problem.

Problem 2. For given an ordered set $(C, \leq)$ and elements $c_1, c_2 \in C$, decide $c_1 \leq c_2$ or not. □

When deciding $c_1 \leq c_2$ in the domain C is difficult or undecidable, abstract interpretation can be used as follows.

1. Find another ordered set $(A, \leq_A)$ and mappings $\alpha\colon C \to A$ and $\gamma\colon A \to C$ satisfying the following condition.

$$\forall c \in C \forall a \in A.\ c \leq \gamma(a) \text{ iff } \alpha(c) \leq_A a. \tag{1}$$

   The mappings α and γ satisfying the condition above is called *Galois connection*. In this situation, $(C, \leq)$ is often called the *concrete domain* and $(A, \leq_A)$ the *abstract domain*.
2. Find $a_1 \in A$ satisfying the following condition.

$$\alpha(c_1) \leq a_1 \text{ and } \gamma(a_1) \leq c_2. \tag{2}$$

   From the condition (1), if there is such a_1, we can conclude that $c_1 \leq c_2$.

Since the original problem Problem 2 has been divided into two problems as (2), one may think that the problem becomes more difficult. But in some special cases as mentioned below, we can easily find a_1 satisfying (2). For functions f and g, let $\mathit{lfp}(\mathsf{f})$ denote the least fixed point of f and $\mathsf{f} \circ \mathsf{g}$ denote the composition of f and g. If $c_1 = \mathit{lfp}(\mathsf{f})$ for a monotone function $\mathsf{f}\colon C \to C$, let $\mathsf{g}\colon A \to A$ be a function defined as $\mathsf{g} = \alpha \circ \mathsf{f} \circ \gamma$. If $\mathit{lfp}(\mathsf{g}) \in A$ exists, then we can see $\alpha(\mathit{lfp}(\mathsf{f})) \leq_A \mathit{lfp}(\mathsf{g})$ from the condition (2). Thus in this case we can take $a_1 = \mathit{lfp}(\mathsf{g})$.

4 Abstraction for Tree Languages

In this section we define abstraction for verification problems by adapting Problem 1 to Problem 2. First, let $(C, \leq)$ be $(2^{\mathcal{T}(\Sigma)}, \subseteq)$, the set of all tree languages and the inclusion order, c_1 and c_2 be $(\to_{\mathcal{R}}^*)(L_1)$ and L_2, respectively. Then Problem 1 can be seen as an instance of Problem 2. In the following, we assume that Σ, $\mathcal{V}$, $\mathcal{R}$ and L_1, L_2 as in Problem 1 are given. Let $\mathsf{f}_{\mathcal{R}}\colon 2^{\mathcal{T}(\Sigma)} \to 2^{\mathcal{T}(\Sigma)}$ be a function defined as $\mathsf{f}_{\mathcal{R}}(L) = \{t \mid s \to_{\mathcal{R}} t, s \in L\} \cup L_1$, then we obtain that $\mathit{lfp}(\mathsf{f})$ is the greatest lower bound of the sequence $(\mathsf{f}_{\mathcal{R}}^n(\emptyset))_n$ with $n \geq 0$ and thus

$lfp(\mathsf{f}_{\mathcal{R}}) = (\rightarrow_{\mathcal{R}}^*)(L_1)$. We define an abstract domain and mappings α and γ to satisfy that

$$lfp(\mathsf{g}_{\mathcal{R}}) \text{ is recognizable} \tag{3}$$

if $lfp(\mathsf{g}_{\mathcal{R}})$ is obtained where $\mathsf{g}_{\mathcal{R}} = \alpha \circ \mathsf{f}_{\mathcal{R}} \circ \gamma$. If an abstraction satisfies the condition (3), then we can check

$$lfp(\mathsf{g}_{\mathcal{R}}) \subseteq L_2 \tag{4}$$

or not and if (4) holds, then we can see that $(\rightarrow_{\mathcal{R}}^*)(L_1) \subseteq L_2$.

The abstraction used in this paper is defined by an equational theory.

Definition 1. *For equations E, let A_E be $A_E = 2^{\mathcal{T}(\Sigma)/E}$ and $\eta\colon \mathcal{T}(\Sigma) \to \mathcal{T}(\Sigma)/E$ be $t \mapsto [t]_{\approx_E}$ for $t \in \mathcal{T}(\Sigma)$ where $\mathcal{T}(\Sigma)/E$ is the equivalence classes of the equational theory induced by E and $\approx_E$ denotes the equational theory induced by E. We call A_E the* equation-based abstraction by E *with the mappings $\alpha_E\colon 2^{\mathcal{T}(\Sigma)} \to 2^{\mathcal{T}(\Sigma)/E}$ and $\gamma_E\colon 2^{\mathcal{T}(\Sigma)/E} \to 2^{\mathcal{T}(\Sigma)}$ which are induced from η, i.e. $\alpha_E(L) = \{\eta(t) \mid t \in L\}$ and $\gamma_E(L) = \{t \mid \eta(t) \in L\}$.*

Example 1. Let $\mathcal{R}_1$ be $\{f(x,y) \to f(g(x),h(y))\}$, E be a set of equations consisting of $\{g(g(x)) = g(x), h(h(x)) = h(x)\}$ and L_1 be a recognizable tree language $\{f(a,b)\}$. Then, we have

$$(\rightarrow_{\mathcal{R}_1}^*)(L_1) = \{f(g^n(a), g^n(b)) \mid n \geq 0\} \text{ and} \tag{5}$$

$$\gamma_E \circ \alpha_E((\rightarrow_{\mathcal{R}_1}^*)(L_1)) = \{f(g^n(a), g^m(b)) \mid n, m \geq 1\} \cup \{f(a,b)\}. \tag{6}$$

In fact, (6) is recognizable whereas (5) is not and (6) includes (5). □

In the next section, we present a technique to obtain a set of equations which can be used for abstraction as in Example 1.

5 Verification Procedure

For the function $\mathsf{f}_{\mathcal{R}}$ in the previous section ($\mathsf{f}_{\mathcal{R}}(L) = \{t \mid s \to_{\mathcal{R}} t, s \in L\} \cup L_1$), we adopt the following procedures. In the following, a state of tree automata may have a tree structure. A state having a tree structure t may be written as $\langle t \rangle$ to emphasize that t is a state. We call a tree structured state $\langle t \rangle$ of the form $t = f(t_1, \ldots, t_n)$ a *term state*, otherwise a *constant state*.

Procedure 1. (modify) This procedure takes a tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$ and a linear TRS $\mathcal{R}$ on $\mathcal{T}(\Sigma, \mathcal{V})$ as inputs and outputs tree automaton $\mathcal{A}'$ defined as follows: For any rewrite rule $l \to r \in \mathcal{R}$, any substitution $\sigma\colon \mathcal{V} \to \mathcal{Q}$ and any state $q \in \mathcal{Q}$, if

$$l\sigma \to_{\mathcal{A}}^* q \text{ and } r\sigma \not\to_{\mathcal{A}}^* q, \tag{7}$$

then construct $\mathcal{A}'$ by adding $\langle r\sigma \rangle \to q$ to $\mathcal{A}$ and execute addtrans($r\sigma$) where $\langle r\sigma \rangle$ is a state which may have a tree structure. That is, construct new tree automaton by adding new transition rules and states to satisfy that $r\sigma \to^* q$ holds. A transition move caused by $\langle r\sigma \rangle \to q$ is called a *rewriting move*. □

Procedure 2. (addtrans) This procedure takes a term t on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$. If t has already been defined as a state, then the procedure defines no transitions. Otherwise the procedure defines new states and transition rules of $\mathcal{A}$ as follows: (i) If $t = c$ with c a constant, then define $c \to \langle c \rangle$ as a transition rule. (ii) If $t = f(t_1, \ldots, t_n)$ with f a function symbol of arity n, then define $f(\langle t_1 \rangle, \ldots, \langle t_n \rangle) \to \langle t \rangle$ and execute $\mathsf{addtrans}(t_i)$ for $1 \leq i \leq n$. □

We assume that any state not defined in addtrans is a constant state. Let *Rec* be the class of recognizable tree languages. In the following, we regard a tree automaton $\mathcal{A}$ as a tree language $\mathcal{L}(\mathcal{A})$ and for a TRS $\mathcal{R}$ the procedure modify as a function $\mathsf{modify}_{\mathcal{R}} : \mathit{Rec} \to \mathit{Rec}$ defined by $\mathcal{A} \mapsto \mathsf{modify}(\mathcal{A}, \mathcal{R})$. Although the function $\mathsf{modify}_{\mathcal{R}}$ is not a strict implementation of $\mathsf{f}_{\mathcal{R}}$, it is proved in [8, 9] that $\mathit{lfp}(\mathsf{f}_{\mathcal{R}}) = \mathit{lfp}(\mathsf{modify}_{\mathcal{R}})$.

In order to consider an abstraction for a TRS, first we define a class of TRSs and show that for a TRS in the class we do not need any abstraction. That is, there always exists an integer k such that $\mathsf{modify}_{\mathcal{R}}^{k+1}(\mathcal{A}) = \mathsf{modify}_{\mathcal{R}}^{k}(\mathcal{A})$ for any tree automaton $\mathcal{A}$ where $\mathsf{modify}_{\mathcal{R}}^{k}(\mathcal{A})$ means applying $\mathsf{modify}_{\mathcal{R}}$ to $\mathcal{A}$ for k times. Before defining the class of TRSs, we need some notions on terms and a TRS.

Definition 2. *Let λ denote the root position.* A term s sticks-out of a term t at (p_1, p_2) *with $p_1 \in \mathcal{P}os_{\mathcal{V}}(t)$ and $p_2 \in \mathcal{P}os_{\mathcal{V}}(s)$ if for any position p with $\lambda \preceq p \prec p_1$ we have $p \in \mathcal{P}os(s)$ and the function symbol of s at o and the function symbol of t at o are the same and $p_1 \preceq p_2$. If s sticks-out of t at (p_1, p_2) and $p_1 \prec p_2$, then we say that s* properly sticks-out of *t.*

Definition 3. *The* generalized sticking-out graph *of a TRS $\mathcal{R}$ is a directed graph $G = (V, E)$ where $V = \biguplus_{l \to r \in \mathcal{R}} \mathcal{P}os_{\mathcal{V}}(l) \uplus \mathcal{P}os_{\mathcal{V}}(r)$ and $\uplus$ means disjoint union (i.e. the vertices are all variable positions of all rewrite rules in $\mathcal{R}$) and each edge in E, defined as follows, has a weight. In the following, an element p in V such that p is a position of a variable x in left- (resp. right-) hand side of a rewrite rule $l \to r$ is written as $(l \to r, L(\mathit{resp.}\ R), p)$ or $(l \to r, L(\mathit{resp.}\ R), x)$. Let $l_1 \to r_1$ and $l_2 \to r_2$ be (possibly identical) rewrite rules in $\mathcal{R}$.*

1. *If r_2 properly sticks-out of a subterm l_1/p' of l_1 with $p' \in \mathcal{P}os(l_1)$ at (p_1, p_2), then E contains an edge from $(l_2 \to r_2, R, p_2)$ to $(l_1 \to r_1, L, p' \cdot p_1)$ with weight one.*
2. *If a subterm of r_2 at $p' \in \mathcal{P}os(r_2)$ sticks-out of l_1 at (p_1, p_2), E contains an edge from $(l_2 \to r_2, R, p' \cdot p_2)$ to $(l_1 \to r_1, L, p_1)$ with weight zero.*
3. *If a subterm of l_1 at $p' \in \mathcal{P}os(l_1)$ properly sticks-out of r_2 at (p_1, p_2), then E contains an edge from $(l_2 \to r_2, R, p_1)$ to $(l_1 \to r_1, L, p' \cdot p_2)$ with weight one.*
4. *If l_1 sticks-out of a subterm r_2/p' of r_2 with $p' \in \mathcal{P}os(r_2)$ at (p_1, p_2), then E contains an edge from $(l_2 \to r_2, R, p' \cdot p_1)$ to $(l_1 \to r_1, L, p_2)$ with weight zero. Situations from 1 to 4 are shown in Fig. 2.*
5. *For a rewrite rule $l \to r$ and a variable $x \in \mathcal{V}ar(l) \cap \mathcal{V}ar(r)$, E contains an edge from $(l \to r, L, x)$ to $(l \to r, R, x)$ with weight zero.*

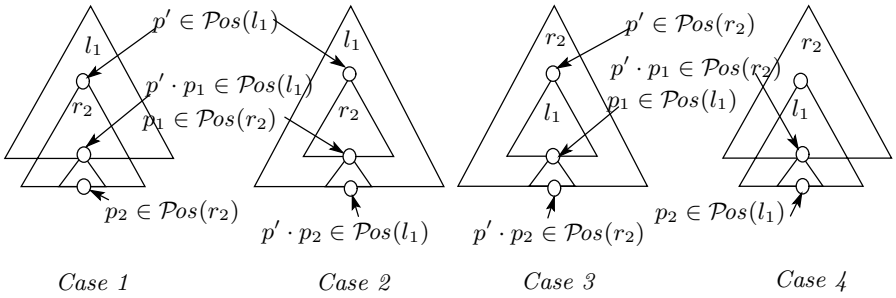

Fig. 2. Conditions for four cases

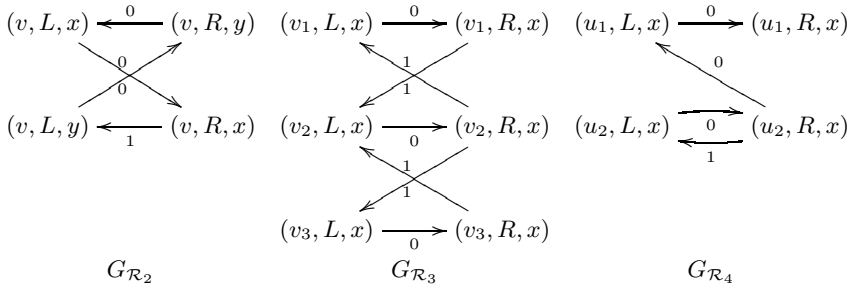

Fig. 3. Generalized sticking-out graphs

Example 2. Let $\mathcal{R}_2, \mathcal{R}_3$ and $\mathcal{R}_4$ be TRSs defined as follows.

$$\mathcal{R}_2 \left\{ v\colon f_2(x, h(g(y))) \to f_2(g(k(y)), h(x)) \right.$$

$$\mathcal{R}_3 \begin{cases} v_1\colon g(x) \to k(l(x)) \\ v_2\colon k(x) \to f(g(h(x))) \\ v_3\colon f(x) \to k(l'(x)) \end{cases}$$

$$\mathcal{R}_4 \begin{cases} u_1\colon g(f(k(k(x)))) \to g(f(k(h(x)))) \\ u_2\colon h(f(x)) \to f(k(x)) \end{cases}$$

Then the generalized sticking out graphs $G_{\mathcal{R}_2}, G_{\mathcal{R}_3}$ and $G_{\mathcal{R}_4}$ of $\mathcal{R}_2, \mathcal{R}_3$ and $\mathcal{R}_2$ respectively are shown in Fig. 3. □

Definition 4. *A TRS $\mathcal{R}$ is a* generalized finite path-overlapping term rewriting system (GFPO-TRS) *if the generarized sticking-out graph of $\mathcal{R}$ has no cycle of weight one or more.*

We write a linear GFPO-TRS as L-GFPO-TRS. To show that any L-GFPO-TRS effectively preserves recognizability, we need some notions on states and verticies

of a generalized sticking-out graph. The proof goes along the way of the proof of that RL-FPO-TRS effectively preserves recognizaiblity [8, 9]. For a state $q \in \mathcal{Q}$ which may have a tree structure (i.e. the state q may be defined in $\mathsf{addtrans}$), define the *number of layers of* q, denoted $\mathsf{layer}(q)$ as follows: (i) if q is a constant state, then $\mathsf{layer}(q) = 0$ and (ii) if $q = \langle r\sigma/o\rangle$ with $l \to r \in \mathcal{R}$, $o \in \mathcal{P}os(r)$, r/o is not a variable, $\mathcal{V}ar(r/o) = \{x_1, \ldots, x_n\}$ and $\sigma = \{x_i \mapsto q_i \mid 1 \le i \le n\}$, then $\mathsf{layer}(q) = 1 + \max\{\mathsf{layer}(q_i) \mid 1 \le i \le n\}$. For an L-GFPO-TRS $\mathcal{R}$, the *rank of a vertex* v of the generalized sticking-out graph $G_{\mathcal{R}}$ of $\mathcal{R}$ is the maximum weight of a path to v from any vertex of $G_{\mathcal{R}}$. The rank of a vertex $v = (l \to r, d, x)$ is written as $\mathsf{rank}(v)$ or $\mathsf{rank}(l \to r, d, x)$.

In the following, let $(\mathcal{F}, \mathcal{Q}_k, \mathcal{Q}_f, \Delta_k)$ be $\mathsf{modify}^k_{\mathcal{R}}(\mathcal{A})$ for a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$. If there is an integer m such that $\mathsf{modify}^{m+1}(\mathcal{A}) = \mathsf{modify}^m(\mathcal{A})$, then we have $\mathit{lfp}(\mathsf{f}_{\mathcal{R}}) = \mathsf{modify}^m(\mathcal{A})$. This implies that a TRS in the class effectively preserves recognizability. In the following, we will show that for an L-GFPO-TRS there exists an integer m such that $\mathsf{modify}^{m+1}(\mathcal{A}) = \mathsf{modify}^m(\mathcal{A})$ for any tree automaton $\mathcal{A}$.

Lemma 1. *Assume $(\mathcal{F}, \mathcal{Q}_k, \mathcal{Q}_f, \Delta_k)$ are defined as above for $k \ge 0$. Let $l \to r$ be a rewrite rule in $\mathcal{R}$, σ be a substitution and q be a state which are used at (7) of* modify*. For any variable $x \in \mathcal{V}ar(l)$ if $\mathsf{rank}(l \to r, L, x) \le j$, then $\mathsf{layer}(x\sigma) \le j$.*

Proof. The proof is shown by induction on the number of k. The base case holds since for any constant state q', $\mathsf{layer}(q') = 0$. Assume the lemma holds for $k = n - 1$ and consider the case when $k = n$. The incductive part is shown by contradiction. Let o_x be the position of x in l and p_x be a position such that $x\sigma = p_x$ and assume that $\mathsf{layer}(p_x) \ge j + 1$. In the sequence $l\sigma \to^* q$, consider how the number of layers changes from o_x to the root. There are four cases:

1. A rewriting move is caused at a certain position. Let o be the inner most position among them. There are two different subcases:
   (a) The number of layers does not increase at any o' with $o \prec o' \prec o_x$.
   (b) There is a position o' with $o \prec o' \prec o_x$ such that the number of layers increases at o'.
2. There are no rewriting moves in the sequence. There are two subcases:
   (a) The number of layers does not increase at any o' with $o \prec o' \prec o_x$.
   (b) There is a position o' with $\lambda \prec o' \prec o_x$ such that the number of layers increases at o'.

Assume case 1(a). In this case we can drive a contradiction as follows. Let $l' \to r'$ be the rewrite rule which is used for defining the rewrite move at o. Then the state just before the rewriting move at o can be written as $\langle r'\sigma'\rangle$. Because $\mathsf{layer}(\langle r'\sigma'/o''\rangle) = \mathsf{layer}(p_x)$, p_x can be written as $p_x = \langle r'\sigma'/o''\rangle$ for some σ' where $o \cdot o'' = o_x$. Since $\mathsf{layer}(p_x) \ge k + 1$, there is a variable $y \in \mathcal{V}ar(r'/o'')$ such that $\mathsf{layer}(y\sigma') \ge k$. By inductive hypothesis for the state $y\sigma'$, we have $\mathsf{rank}(l' \to r', R, y) \ge k$. On the other hand, the facts that there is no rewriting move from o_x to o and the number of layers does not increace from o_x to o impley that all moves from o_x to o are defined by $\mathsf{addtrans}$. By the construction

of addtrans, the function symbol of l at $o \cdot o'$ is the same as the function symbol of r' at o for every o' such that $o \cdot o' \prec o_x$; this means that r' properly sticks-out of l/o at (o_x, o_y) where $r'/o_y = y$ and thus there is an edge from $(l' \to r', R, y)$ to $(l \to r, L, x)$ with weight one. We have observed that $\mathsf{rank}(l' \to r', R, y) \geq k$ and thus $\mathsf{rank}(l \to r, L, x) \geq k + 1$, a contradiction.

Assume case 1(b). Let $l' \to r'$ be the rewrite rule which are used for defining the rewrite move at o. Since $\mathsf{layer}(p_x) \geq j + 1$ and since the number of layers was firstly increased at o' on the path from o_x to o for some position o', we have that $r'/o \cdot o'$ is a variable (say y). The state just before the rewriting move can be written as $\langle r'\sigma' \rangle$ for some substitution σ'. From the fact that there is no rewriting move from o' to o, we can see that there is an edge from $(l' \to r', R, y)$ to $(l \to r, L, x)$ with weight zero. On the other hand, by the inductive hypothesis, $\mathsf{layer}(r'\sigma'/o') \geq j + 1$ implies $\mathsf{rank}(l' \to r', R, y) \geq j + 1$; this leads us to a contradiction.

Assume case 2(a). Since $\mathsf{layer}(p_x) \geq j + 1$, we can write p_x as $\langle r'\sigma'/o' \rangle$ for some $l' \to r'$, σ' and o' such that r'/o' is a non-variable term having a variable. Moreover, there is a variable $y \in \mathcal{V}ar(r'/o')$ such that $\mathsf{layer}(y\sigma') \geq j$. By inductive hypothesis, we have $\mathsf{rank}(l' \to r', R, y) \geq j$. On the other hand, from the fact that there is no rewriting move and no position where the number of layers increases, q can be written as $q = \langle r'\sigma/o'' \rangle$ for some $o'' \prec o'$. Since all moves from o_x to the root in the sequence $l\sigma \to^* q$ are defined by addtrans, r'/o'' properly sticks-out of l at (o_x, o_y) where $r'/o_y = y$ wtih the fact that r'/o' is a non-variable term having a variable. Since $\mathsf{rank}(l' \to r', R, y) \geq j$, $\mathsf{rank}(l \to r, L, x)$ must be $\mathsf{rank}(l \to r, L, x) > j$, a contradiction.

Assume case 2(b). Let o' be the position in which the number of layers increases. Since the number of layers increases at o' and there is no rewriting move from o' to the root, there is a rewrite rule $l' \to r'$ such that q can be written as $q = \langle r'\sigma'/o'' \rangle$ for some σ' and o''. Also we have that $r'/o'' \cdot o'$ is a variable (say y) and thus l sticks-out of r'/o'' at (o', o_x). On the other hand, $\mathsf{layer}(y\sigma') \geq k + 1$, by the inductive hypothesis, we have $\mathsf{rank}(l' \to r') \geq k + 1$. Since l sticks-out of r'/o'' at $(o'' \cdot o', o_x)$ and thus the generalized sticking-out graph has an edge from $(l' \to r', R, y)$ to $(l \to r, L, x)$ with weight zero, which leads a contradiction. □

Theorem 1. *An L-GFPO-TRS effectively preserves recognizability.*

Proof. By definition of GFPO-TRS (Definition 4) the rank of any vertex of the generalized sticking-out graph of an L-GFPO-TRS is bounded. By Lemma 1, the numbers of layers of states are bounded by the ranks for any $\mathcal{A}_m$ with $m \geq 0$. If the numbers of layers are bounded, so is the number of states; this implies that there is an integer k such that $\mathsf{modify}_{\mathcal{R}}^{k+1}(\mathcal{A}) = \mathsf{modify}_{\mathcal{R}}^{k}(\mathcal{A})$. □

Example 3. Let $\mathcal{R} = \{v\colon f_2(x, y) \to f_2(a, g(x))\}$ be a TRS. The generalized sticking-out graph $G_{\mathcal{R}}$ of $\mathcal{R}$ has three verteces $(v, L, x), (v, L, y), (v, R, x)$ and two edges $(v, L, x) \to (v, R, x)$ and $(v, R, x) \to (v, L, y)$ with weights 0 and 1 respectively. Since $G_{\mathcal{R}}$ has no cycle, $\mathcal{R}$ belongs to GFPO-TRS. Remark that

$\mathcal{R}$ does not belong to RL-FPO-TRS [9], which is the known widest decidable subclass of TRSs effectively preserving recognizability as far as we know. □

According to Example 3, we obtain a new result on a decidable subclass of TRSs which effectively preserves recognizability.

Proposition 1. *There are term rewriting systems which belong to L-GFPO-TRS and not to RL-FPO-TRS.* □

To deal with a TRS whose generalized sticking-out graphs has loops of weight one or more, we give a technique to find an appropriate set of equations for equation-based abstraction (Definition 1). For defining $\mathsf{g}_{\mathcal{R}}$ in the previous section, we consider a function $\bigtriangledown\colon \mathit{Rec} \to \mathit{Rec}$, called an *abstraction operator* defined as $\bigtriangledown = \gamma_E \circ \alpha_E$ where γ_E and α_E are defined by a set E of equations as in Definition 1. Once we obtain an abstraction operator $\bigtriangledown$, by using $\mathsf{modify}_{\mathcal{R}}$ and $\bigtriangledown$, we can define the sequence of tree automata $(\mathcal{A}^n_{\mathcal{R},\bigtriangledown})_{n\geq 0}$ for a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ as $\mathcal{A}^n_{\mathcal{R},\bigtriangledown} = (\bigtriangledown \circ \mathsf{modify}_{\mathcal{R}})^n(\mathcal{A})$. If there is m such that $\mathcal{A}^{m+1}_{\mathcal{R},\bigtriangledown} = \mathcal{A}^m_{\mathcal{R},\bigtriangledown}$, we can take $\mathcal{A}^m_{\mathcal{R},\bigtriangledown}$ as an abstraction of $\mathit{lfp}(\mathsf{f}_{\mathcal{R}})$.

In our technique described below, the equations for abstraction are dynamically obtained during the iteration of procedure modify. In other words, instead of finding a set of equations statically from a given TRS and a tree automaton, we use a sequence $(E_n)_{n\geq 0}$ of sets of equations with $E_n \subseteq E_{n+1}$ for $n \geq 0$. Let $\bigtriangledown_n = \gamma_{E_n} \circ \alpha_{E_n}$ be an abstraction operator defined from E_n for $n \geq 0$. Each E_n for $n \geq 0$ is obtaind during the computation of $(\mathcal{A}^n_{\mathcal{R},\bigtriangledown_n})_{n\geq 0}$ where $\mathcal{A}^n_{\mathcal{R},\bigtriangledown_n}$ is defined as $\mathcal{A}^n_{\mathcal{R},\bigtriangledown_n} = (\bigtriangledown_n \circ \mathsf{modify}_{\mathcal{R}})^n(\mathcal{A})$. In this case, we still have that if there is m such that $\mathcal{A}^{m+1}_{\mathcal{R},\bigtriangledown_{m+1}} = \mathcal{A}^m_{\mathcal{R},\bigtriangledown_m}$, then we can take $\mathcal{A}^m_{\mathcal{R},\bigtriangledown_m}$ as an abstraction of $\mathit{lfp}(\mathsf{f}_{\mathcal{R}})$.

Procedure 3. ($\mathsf{e\text{-}modify}$) This procedure takes a tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$, a linear TRS $\mathcal{R}$ on $\mathcal{T}(\Sigma, \mathcal{V})$ and a S and S_e of tuples $(\langle t\rangle, o, H)$ of a state $\langle t\rangle \in \mathcal{Q}$, a position o of t and a sequence H of vertices of the generalized sticking-out graph $G_{\mathcal{R}}$ of $\mathcal{R}$ as inputs and outputs tree automaton $\mathcal{A}'$ and sets S' and S'_e of the same things of S and S_e which are defined by adding something to $\mathcal{A}$ and sets S and S_e respectively as follows. For any rewrite rule $l \to r \in \mathcal{R}$, any substitution $\sigma\colon \mathcal{V} \to \mathcal{Q}$ and any state $q \in \mathcal{Q}$, if $l\sigma \to^*_{\mathcal{A}} q$ and $r\sigma \not\to^*_{\mathcal{A}} q$, then construct $\mathcal{A}'$ by adding $\langle r\sigma\rangle \to q$ to $\mathcal{A}$ and execute $\mathsf{addtrans}(r\sigma)$. For any variable $x \in \mathcal{V}ar(l)$, let o_x, o'_x be the positions of x in l and r respectively and p_x be the state associated with x, i.e. $l/o_x = x$, $r/o'_x = x$ and $x\sigma = p_x$ and do the following.

1. For any position o' with $\lambda \prec o' \preceq o'_x$ add $(\langle r\sigma/o'\rangle, o'', (l \to r, R, x))$ where o'' is a position such that $o' \cdot o'' = o'_x$.
2. If there is an element $(p_x, o_h, H) \in S$ for some o_h and H, then add $(p_x, o_h, H \cdot (l \to r, R, x))$ to S_e and for any position o' with $\lambda \prec o' \preceq o'_x \cdot o_h$ add $(\langle r\sigma/o'\rangle, o'', H \cdot (l \to r, R, x))$ to S where o'' is a position such that $o' \cdot o'' = o'_x \cdot o_h$. □

Procedure 4. ($\bigtriangledown_E$) The abstraction operator $\bigtriangledown_E$ takes a tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta)$, a linear TRS $\mathcal{R}$ on $\mathcal{T}(\Sigma, \mathcal{V})$ and sets S and S_e of the same things as in e-modify as inputs and outputs tree automaton $\mathcal{A}'$ which is defined from $\mathcal{A}$ as follows. If there is an element $(p_x, o_h, v \cdot H \cdot v)$ in S_e for some p_x, o_h, v a vertex of $G_{\mathcal{R}}$ and H, then (1) add an ε-transition rule $\langle p_x\{o_h \leftarrow p_x\}\rangle \to p_x$ to $\mathcal{A}$ and (2) execute $\mathsf{addtrans}(\langle p_x\{o_h \leftarrow p_x\}\rangle)$ for defining $\mathcal{A}'$. Intuitively, construct new tree automaton by adding new transition rules and states to satisfy that the recognizable language is abstracted by the equation $C[C[x]] = C[x]$ where $C[\Box] = p_x\{o_h \leftarrow \Box\}$. The move caused by the transition $\langle p_x\{o_h \leftarrow p_x\}\rangle \to p_x$ is called an *abstraction move.* $\Box$

We first explain how abstraction moves implement equation-based abstraction and then we describe how the sets S and S' are used for finding equations. Each equation used for abstraction has the form $C[C[x]] = C[x]$ where C is a context with one hole and x a variable. Intuitively, C in the equations is a pattern of a context which may repeatedly appear in terms in $lfp(\mathsf{f}_{\mathcal{R}})$. In other words, $C[C[x]] = C[x]$ means that once a term t containing C, i.e. $t = C'[C[t']]$ where C' is a context and t' a term, appears in $lfp(\mathsf{f}_{\mathcal{R}})$, we approximate $lfp(\mathsf{f}_{\mathcal{R}})$ by a tree language including terms of the forms $C'[C^n[t']]$ for $n \geq 1$. To mention about the equation $C[C[x]] = C[x]$ more precisely, we remark that if a state p_x is a term state, p_x can be written as $p_x = C_{p_x}[q_1, \ldots, q_n]$ where q_i is a constatnt state ($1 \leq i \leq n$). Also remark that a tree automaton $\mathcal{A}$ can be seen as an order-sorted signature [2] by regarding each state as a sort and the set of terms of sort q (q is a state of $\mathcal{A}$) is defined by $\{t \in \mathcal{T}(\mathcal{F}) \mid t \to^*_{\mathcal{A}} q\}$. We assume $C_{p_x}[q_1, \ldots, q_n]/o_h = q_m$ for some m and write x_p for a variable of sort p. The corresponding equation for the transition $\langle p_x\{o_h \leftarrow p_x\}\rangle \to p_x$ is $C_{p_x}[x'_{q_1}, \ldots, x'_{q_{m-1}}, C_{p_x}[x_{q_1}, \ldots, x_{q_n}], x'_{q_{m+1}}, \ldots, x_{q_n}] = C_{p_x}[x_{q_1}, \ldots, x_{q_n}]$.

Next, we describe how the arguments S and S_e of e-modify (and $\bigtriangledown_E$) works for finding a set of equations for abstraction. Intuitively saying, an elemtn (q, o, H) of S or S_e means that a state q has a history H and the history started from q/o. Note that a state may have a tree structure. The history means that if H includes $(l' \to r', R, y)$, then q has the form $\langle r'\sigma'/o'\rangle$ for some substitution σ' where $\lambda \prec o' \preceq o$. For a rewrite rule $l \to r$, a substitution σ and a state p which are used of the sequence $l\sigma \to^* q$ in e-modify, consider the situation that for some $x \in \mathcal{V}ar(l)$ the state $p_x = x\sigma$ has a history including $(l \to r, R, x)$. In this case, the process of the iteration of generating new states may be infinitely executed. To break the iteration, Procedure e-modify first adds a new element to S_e and operator $\bigtriangledown_E$ introduce a transition rule which causes abstraction moves.

If we define $\mathsf{e\text{-}modify}_{\mathcal{R}}$ as a map $(\mathcal{A}, S, S_e) \mapsto (\mathcal{A}', S', S'_e)$ where $(\mathcal{A}', S', S'_e) = \mathsf{e\text{-}modify}(\mathcal{A}, \mathcal{R}, S, S_e)$ and $\bigtriangledown^{\mathcal{R}}_E$ as a map $(\mathcal{A}, S, S_e) \mapsto (\mathcal{A}', S, S_e)$ where $\mathcal{A}' = \bigtriangledown_E(\mathcal{A}, \mathcal{R}, S, S_e)$, we can consider a sequence $(\mathcal{A}'^n_{\mathcal{R},\bigtriangledown})_{n\geq 0}$ of tree automata as that $\mathcal{A}'^n_{\mathcal{R},\bigtriangledown}$ is the first component of $(\bigtriangledown^{\mathcal{R}}_E \circ \mathsf{e\text{-}modify}_{\mathcal{R}})^n(\mathcal{A}, \emptyset, \emptyset)$. In this case, we still have that if there is m such that $\mathcal{A}'^{m+1}_{\mathcal{R},\bigtriangledown} = \mathcal{A}'^m_{\mathcal{R},\bigtriangledown}$, then we can take the first component of $\mathcal{A}'^m_{\mathcal{R},\bigtriangledown}$ as an abstraction of $lfp(\mathsf{f}_{\mathcal{R}})$.

Example 4. Let $\mathcal{A}$ be a tree automaton consisting of

$$a \to q_a \qquad b \to q_b \qquad f(q_a, q_b) \to q_f$$

with a final state q_f, i.e. $\mathcal{L}(\mathcal{A}) = \{f(a,b)\}$ and $\mathcal{R}_1$ be the TRS $\{v\colon f(x,y) \to f(g(x), h(y))\}$ in Example 1. The generalized sticking-out graph $G_{\mathcal{R}_1}$ of $\mathcal{R}_1$ has cycles of weight one.

$$(v,L,x) \underset{1}{\overset{0}{\rightleftarrows}} (v,R,x) \qquad (v,L,y) \underset{1}{\overset{0}{\rightleftarrows}} (v,R,y)$$

In fact, $lfp(\mathsf{f}_{\mathcal{R}_1})$ is not recognizable, i.e.

$$lfp(\mathsf{f}_{\mathcal{R}_1}) = \{f(g^n(a), g^n(b)) \mid n \geq 0\}. \tag{8}$$

On the other hand, since $f(q_a, q_b) \to^*_{\mathcal{A}} q_f$ and $f(g(q_a), h(q_b)) \not\to^*_{\mathcal{A}} qf$, we have the following new transition rules

$$\begin{array}{ll} g(q_a) \to \langle g(q_a)\rangle & h(q_b) \to \langle h(q_b)\rangle \\ f(\langle g(q_a)\rangle, \langle h(q_b)\rangle) \to \langle f(g(q_a), h(q_b))\rangle & \langle f(g(q_a), h(q_b))\rangle \to q_f \end{array}$$

and the set S'

$$\begin{array}{ll} (q_a, \lambda, (v,R,x)) & (q_b, \lambda, (v,R,y)) \\ (\langle g(q_a)\rangle, 1, (v,R,x)) & (\langle h(q_b)\rangle, \lambda, (v,R,y)) \end{array}$$

by construction e-modify. Since there is no element (p, o, H) in S such that H has the form $v \cdot H' \cdot v$ for some v and H', the abstraction operator $\bigtriangledown_E$ does nothing. For the second step, since $f(\langle g(q_a)\rangle, \langle h(q_b)\rangle) \to^*_{\mathcal{A}} \langle f(g(q_a), h(q_b))\rangle$ and $f(g(\langle g(q_a)\rangle), h(\langle h(q_b)\rangle)) \not\to^*_{\mathcal{A}} \langle f(g(q_a), h(q_b))\rangle$, the procedure e-modify generates new transtion rules and new element for the set S' as follows.

$$\begin{array}{l} g(\langle g(q_a)\rangle) \to \langle g(g(q_a))\rangle \qquad h(\langle h(q_b)\rangle) \to \langle h(h(q_b))\rangle \\ f(\langle g(g(q_a))\rangle, \langle h(h(q_b))\rangle) \to \langle f(g(g(q_a)), h(h(q_b)))\rangle \\ \langle f(g(g(q_a)), h(h(q_b)))\rangle \to \langle f(g(q_a), h(q_b))\rangle \end{array}$$

$$\begin{array}{ll} (\langle g(q_a)\rangle, \lambda, (v,R,x)) & (\langle h(q_b)\rangle, \lambda, (v,R,y)) \\ (\langle g(g(q_a))\rangle, 1, (v,R,x)) & (\langle h(h(q_b))\rangle, \lambda, (v,R,y)) \\ (\langle g(q_a)\rangle, 1, (v,R,x) \cdot (v,R,x)) & (\langle h(q_b)\rangle, 1, (v,R,y) \cdot (v,R,y)) \\ (\langle g(g(q_a))\rangle, 1 \cdot 1, (v,R,x) \cdot (v,R,x)) & (\langle h(h(q_b))\rangle, 1 \cdot 1, (v,R,y) \cdot (v,R,y)) \end{array}$$

Here we also obtain S'_e as follows.

$$(\langle g(q_a)\rangle, 1, (v,R,x) \cdot (v,R,x)) \quad (\langle h(q_b)\rangle, 1, (v,R,y) \cdot (v,R,y))$$

By $\bigtriangledown_E$ of Procedure 4 and the set S'_e above, we obtain the following transitions.

$$\langle g(g(q_a))\rangle \to \langle g(q_a)\rangle \quad \langle h(h(q_b))\rangle \to \langle h(q_b)\rangle \tag{9}$$

Due to the abstraction moves by (9), there are no transition rules and no states to add. Finally, we have a tree automaton whose accepting tree language is

$$\{f(g^n(a), g^m(b)) \mid n, m \geq 1\} \cup \{f(a, b)\}. \tag{10}$$

In fact, (10) is recognizable whereas (8) is not and (10) includes (8). □

One may think that for a given TRS the equation used for abstraction can be obtained statically. For example, for the TRS in Example 4, it is easily imagined that an appropriate equation theory is $\{g(g(x)) = g(x), h(h(x)) = h(x)\}$ and in fact we obtained the corresponding abstraction transition rules (9) to those equations. The next example shows a non-trivial case.

Example 5. Consider the TRS $\mathcal{R}_2 = \{v\colon f_2(x, h(g(y))) \to f_2(g(k(y)), h(x))\}$ in Example 2 and tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \{f_2(g(a), h(g(b)))\}$. Then we have

$$\begin{aligned} \mathit{lfp}(\mathsf{f}_{\mathcal{R}_2}) = {} & \{f_2(g(k^n(a)), h(g(k^n(b)))) \mid n \geq 0\} \cup \\ & \{f_2(g(k^{n+1}(b)), k(g(k^n(a)))) \mid n \geq 0\} \text{ and} \end{aligned} \tag{11}$$

$$\begin{aligned} {\mathcal{A}'}^k_{\mathcal{R}_2,\bigtriangledown} = {} & \{f_2(g(k^n(a)), h(g(k^m(b)))) \mid n, m \geq 1\} \cup \\ & \{f_2(g(k^{n+1}(b)), k(g(k^m(a)))) \mid n, m \geq 1\} \cup \\ & \{f_2(g(a), h(g(b))), f_2(g(k(b)), h(g(a)))\} \end{aligned} \tag{12}$$

for some k such that ${\mathcal{A}'}^{k+1}_{\mathcal{R}_2,\bigtriangledown} = {\mathcal{A}'}^k_{\mathcal{R}_2,\bigtriangledown}$. We can see that (12) is recognizable whereas (11) is not and (12) includes (11). □

6 Discussions

In this paper we first give a new decidable subclass L-GFPO-TRS of TRSs which effectively preserves recognizability and then propose a technique to obtain abstraction for TRSs which are not L-GFPO-TRS.

The teqchnique does not work for any (even if linear) TRSs. For example, the abstraction moves defined in $\bigtriangledown_E$ correspond to equations of the form $C[C[x]] = C[x]$ for a context C. On the other hand, for the TRS $\mathcal{R}_3$ in order to obtain abstraction by equation-based abstraction, we may need equations of the form $C_1[C_2[x]] = C_2[C_1[x]]$ where C_1 and C_2 are contexts. In general a TRS whose generalized sticking-out graph has two cycles of weight one or more which share the same vertex cannot be abstracted by out technique. To overcome such cases, a new abstraction operation $\bigtriangledown'_E$ instead of $\bigtriangledown_E$ may be used where $\bigtriangledown'_E$ is defined by replacing (1) and (2) of $\bigtriangledown_E$ by (1') adding an ε-transition rule $\langle p_x \rangle \to p_x/o_h$ to $\mathcal{A}$. This move correspondes to an equation of the form $C[x] = x$ where we can regard x as a variable of sort p_x/o_h as mentioned in Sect. 5.

Acknowledgments. The author thanks to the members of the laboratory for verification and semantics of AIST for helpful discussions.

References

1. F. Baader and T. Nipkow: *Term Rewriting and All That*, Cambridge University Press, 1998.
2. H. Comon: "Equational Formulas in Order-Sorted Algebras," *Proc. of the Fourth Intl. Conf. on RTA*, vol. 443, LNCS, pp. 674–688, Springer-Verlag, 1990.
3. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi: *Tree Automata Techniques and Applications*, draft, 1999.
4. P. Cousot and R. Cousot: "Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," *Proc. of 4th POPL*, pp. 238–252, ACM Press, 1977.
5. T. Genet and F. Klay: "Rewriting for Cryptographic Protocol Verification," *Proc. of 17th CADE*, vol. 1831, LNAI, pp. 271–290, Springer-Verlag, 2000.
6. D. Monniaux: "Abstracting cryptographic protocols with tree automata," *Proc. of 6th SAS*, vol. 1694, LNCS, pp. 149–163. Springer Verlag, 1999.
7. F. Nielson, H. R. Nielson and C. Hankin: "Abstract Interpretation," In *Principles of Program Analysis*, Chapter 4, Springer-Verlang, 1999.
8. T. Takai, Y. Kaji and H. Seki: "Right-linear finite path overlapping term rewriting systems effectively preserve recognizability," Proc. of RTA2000, Norwich, U.K., vol. 1833, LNCS, pp.246–260, 2000.
9. T. Takai, Y. Kaji and H. Seki: "Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability," Scienticae Mathematicae Japonicae (to appear).
10. 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, Vol. E86-D, No. 2, pp. 285–295, February 2003.