Solving a Unification Problem under Constrained Substitutions Using Tree Automata

Yuichi Kaji† Toru Fujiwara‡ Tadao Kasami†

kaji@is.aist-nara.ac.jp kasami@is.aist-nara.ac.jp

fujiwara@ics.es.osaka-u.ac.jp

† Graduate School of Information Science,
Nara Institute of Science and Technology,
8916-5 Takayama, Ikoma, Nara 630-01, Japan

‡ Department of Information and Computer Sciences,
Faculty of Engineering Science,
Osaka University,
1-3 Machikaneyama, Toyonaka , Osaka 560, Japan

Abstract

A generalization of a unification problem for term rewriting systems, named a *unification problem under constrained substitutions* is investigated. Since the problem is undecidable, we present a sufficient condition under which the problem is decidable, and propose a procedure to solve the problem for the class of term rewriting systems which satisfies the sufficient condition. This class properly includes a known decidable class introduced in [4]. The usual unification problem is a special case of the unification problem under constrained substitutions discussed in this paper, and hence our procedure can solve the usual unification problem too. Comparing to other unification procedures such as narrowing, our strategy is quite a new approach to unification problems, and there are a number of rewriting systems for which unification problem can be solved by our procedure but cannot be solved by narrowing.

keywords:

term rewriting systems, unification problems, constrained substitutions, tree automata

Solving a Unification Problem under Constrained Substitutions Using Tree Automata

Yuichi Kaji† Toru Fujiwara‡ Tadao Kasami†

{kaji,kasami}@is.aist-nara.ac.jp fujiwara@ics.es.osaka-u.ac.jp

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-01, Japan

‡ Department of Information and Computer Sciences, Osaka University
1-3 Machikaneyama, Toyonaka , Osaka 560, Japan

Abstract

A generalization of a unification problem for term rewriting systems, named a *unification problem under constrained substitutions* is investigated. Since the problem is undecidable, we present a sufficient condition under which the problem is decidable, and propose a procedure to solve the problem for the class of term rewriting systems which satisfies the sufficient condition. This class properly includes a known decidable class introduced in [4]. The usual unification problem is a special case of the unification problem under constrained substitutions discussed in this paper, and hence our procedure can solve the usual unification problem too. Comparing to other unification procedures such as narrowing, our strategy is quite a new approach to unification problems, and there are a number of rewriting systems for which unification problem can be solved by our procedure but cannot be solved by narrowing.

keywords: term rewriting systems, unification problems, constrained substitutions, tree automata

1 Introduction

Unification problems for term rewriting systems are widely studied because of its fundamental and practical significance. In this paper, a generalization of a unification problem for term rewriting systems, named a *unification problem under constrained substitutions* is proposed, and a procedure that solves the problem for a class of term rewriting systems is presented.

Two terms s and t are *R-unifiable* for a rewriting system R if there are a substitution σ and a term u such that $s\sigma \Rightarrow_R^* u$ and $t\sigma \Rightarrow_R^* u$. An instance of a *unification problem* for a class $\mathcal{R}$ of rewriting systems consists of a term rewriting system $R \in \mathcal{R}$ and two goal terms s and t[1] , and its answer is *true* (or *unifiable*) if s and t are R-unifiable, and *false* (or *not unifiable*) otherwise. It is well-known that the problem is semi-decidable in general. Moreover, it was shown to be semi-decidable even if

[1] Strictly saying, the input contains a set F of function symbols. But for the simplicity, we assume that F is already understood throughout this paper.

$\mathcal{R}$ is restricted to the class of canonical rewriting systems[6]. A number of studies have been devoted to find classes of rewriting systems for which the problem becomes decidable, and efficient unification strategies for those classes have been investigated. For instance, narrowing[3, 7], decomposition[8], and conditional rewriting[2] are widely studied strategies for this purpose.

A unification problem is often used to solve actual problems. For example, it is used in inference mechanism of deductive systems. In these cases, a function symbol represents an operation in the actual problem, a constant represents an elementary object in the problem, and a rewrite rule describes a relation among operations. In the usual unification problem described above, a variable in a goal term can be substituted with an arbitrary term. This means that, in the actual problem, we can execute any operations on any objects to achieve the goal described by two terms. But this is not a very common case. It seems more natural and practically important to consider the case that there is a constraint on operations and objects that are available to achieve a goal. For example, in cryptographic protocols, one cannot execute supervised operations, cannot use secret data etc. This idea leads to a *unification problem under constrained substitutions*. Actually, the concept of a unification problem under constrained substitutions was originally introduced in [4] as a formal verification method of a security of cryptographic protocols in which operations (or objects) that a user can execute (or access) are restricted.

In a unification problem under constrained substitutions, a finite set A of terms is given, and a set of *A-terms* is defined to be a set of terms which is recursively constructed by terms in A and variables (see Section 3 for a formal definition). A variable in goal terms can be substituted with an A-term only. Remark that, for a set F of function symbols, if A is defined to be

$$A = \{f(x_1, \ldots, x_n) \mid f \in F, n \text{ is an arity of } f\},$$

then the set of A-terms coincides with the set of all terms over F. This implies that a procedure that solves the unification problem under constrained substitutions can be applied to the usual unification problem. We note that the reverse side is not trivial since some fundamental properties that are used in usual unification procedures do not hold under constrained substitutions.

In this paper, we introduce a semi-decision procedure that solves the unification problem under constrained substitutions using tree automata[5]. A tree automaton

is an automaton of which input is extended to trees. Since the unification problem under constrained substitutions is a generalization of the usual unification problem, our method can be applied to the usual unification problem. We also present a subclass of rewriting systems for which the problem is solvable by our procedure. This class properly includes the class of rewriting systems for which the problem is solved by the procedure proposed in [4].

2 Preliminaries

2.1 Term Rewriting Systems

Readers are assumed to be familiar with term rewriting systems (TRS) (for example see [1]). In this section, notations used in this paper are briefly introduced. The *arity* of a *function symbol* f is denoted by $a(f)$. If $a(f) = 0$ then f is called a *constant*. The set of *terms* over a set F of function symbols and a set X of *variables* is denoted by $\mathcal{T}_F[X]$. The *size* of a term $t \in \mathcal{T}_F[X]$, denoted $|t|$, is the number of function symbols and variables occurring in t. For a term $t \in \mathcal{T}_F[X]$, the set of variables occurring in t is denoted by $\mathcal{V}(t)$. If $\mathcal{V}(t) = \phi$, then t is called a *ground* term. A term is *linear* if no variable occurs more than once in the term. An *occurrence* of a term is defined in the usual way. The root occurrence is denoted by λ, and the set of occurrences of a term t is denoted by $\mathcal{O}(t)$. If an occurrence o_1 is an ancestor of o_2, then we write $o_1 \preceq o_2$. If $o_1 \preceq o_2$ and $o_1 \neq o_2$, then we write $o_1 \prec o_2$. If neither $o_1 \preceq o_2$ nor $o_2 \preceq o_1$, then o_1 and o_2 are said to be *disjoint*. A subterm of t at occurrence o is denoted by $t|_o$. If a term t is obtained from a term t' by replacing a subterm $t'|_o$ with s, then we write $t = t'[o \leftarrow s]$.

A *substitution* σ is a mapping from variables in X to terms in $\mathcal{T}_F[X]$. If the number of variables substituted by σ is finite, say $x_1, \ldots, x_m$, then σ is denoted as $\sigma = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$, where t_i is a term in $\mathcal{T}_F[X]$ which corresponds to x_i for $1 \leq i \leq m$. If a term t is obtained from a term s by replacing variables $x_1, \ldots, x_m$ in s by terms $t_1, \ldots, t_m$, respectively, then we write $t = s\sigma$ and say t is an *instance* of s. A term s is *literal similar* to a term t if s can be obtained from t by renaming variables in t.

A *rewrite rule* is an ordered pair of terms, written as $l \rightarrow r$ where l and r are terms such that $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. For terms s, t and a set R of rewrite rules, $s \Rightarrow_R t$ if t can be obtained from s by one application of a rewrite rule in R, that is, there are an occurrence $o \in \mathcal{O}(s)$ and a substitution σ such that $s|_o = l\sigma$ and $t = s[o \leftarrow r\sigma]$.

Define $\Rightarrow_R^*$ to be the reflexive and transitive closure of $\Rightarrow_R$. If R can be understood from the context, then the subscript R is omitted.

Two terms s and t are *R-unifiable* if there are a substitution σ and a term u such that $s\sigma \Rightarrow_R^* u$ and $t\sigma \Rightarrow_R^* u$. For TRS R, an *R-unification problem* is the one to decide whether two given terms are R-unifiable or not. A *unification problem* for a class $\mathcal{R}$ of TRS is the one to decide, for a given TRS $R \in \mathcal{R}$ and terms s and t, whether s and t are R-unifiable or not. It can be easily shown that the unification problem for the class of all TRS is semi-decidable, and it is also shown[6] that the problem is semi-decidable even for the class of canonical TRS.

2.2 Tree Automata

A *tree automaton* [2] M is defined by a 6-tuple $M = (Q, F, X, \delta, q_0, Q_F)$ where

- Q is a finite set of *states*,
- F is a finite set of *function symbols*,
- X is a finite set of *variables*,
- δ is a *transition function* which maps $Q \times \{F \cup \{\varepsilon\}\}$ to $\bigcup_{n=1}^{amax} 2^{Q^n}$ where 2^{Q^n} is the power set of n-tuples of Q and *amax* is the maximum arity of function symbols in F. For $q \in Q$ and $f \in F$, δ is defined as in the following forms:

$$(q_1, \ldots, q_n) \in \delta(q, f) \ (n = \max\{1, a(f)\}, q_1, \ldots, q_n \in Q), \text{ or}$$
$$(q_1) \in \delta(q, \varepsilon) \ (q_1 \in Q),$$

- $q_0 \in Q$ is an *initial state*, and
- $Q_F \subseteq Q$ is a set of *final states*.

A tree automaton M takes a term $t \in \mathcal{T}_F[X]$ as an input, and starts its move by setting its state to the initial state q_0 and by attaching its head to the root of t. When M scans a symbol f with a state q (Figure 1), and if $(q_1, \ldots, q_n) \in \delta(q, f)$, then M "splits" the head and the state into n pieces, attaching each head and the state q_i to the root of the ith child of f. M accepts t if every head successfully reaches a leaf of t, scans the leaf, and the state attached to the head gets into a state in Q_F.

To describe the moves of a tree automaton in a formal way, ID's (instantaneous descriptions) of M are defined first. Let \$ be a special end-marker symbol. For a term t, define $\mathcal{O}_E(t) = \{o\$ \mid o \text{ is an occurrence of a non-variable leaf of } t\}$. Define a

[2] The tree automaton used in this paper is a *root-to-frontier tree recognizer* in [5]. Though notations and terminologies are different between them, they are essentially the same formalism.

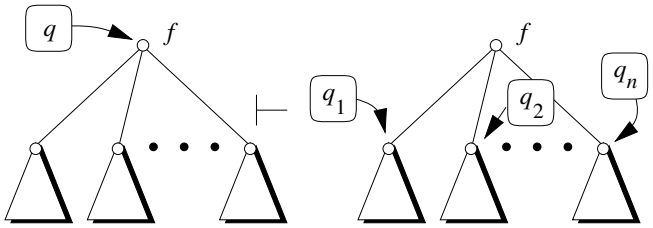


Figure 1: A move of the tree automaton M.

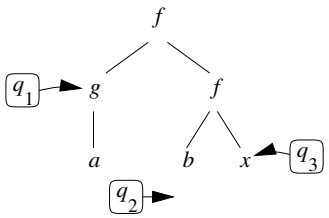


Figure 2: A tree automaton in an ID $(\{\langle q_1, 1\rangle, \langle q_2, 2 \cdot 1\$\rangle, \langle q_3, 2 \cdot 2\rangle\}, t)$.

state-occurrence pair (*so-pair* for short) of M for t to be a pair $\langle q, p\rangle$ where $q \in Q$ and $p \in \mathcal{O}(t) \cup \mathcal{O}_E(t)$. We say the *occurrence* of $\langle q, p\rangle$ is o for $p = o \in \mathcal{O}(t)$, or $p = o\$ \in \mathcal{O}_E(t)$ (ignore trailing \$). An *ID* of M for t is defined to be a 2-tuple (H, t) where H is a set of so-pairs such that, for every leaf of t, H contains exactly one so-pair whose occurrence is an ancestor of the leaf.

As for a set of so-pairs, we define the following notations for simplicity. For an occurrence $o \in \mathcal{O}(t)$ and a set H of so-pairs for $t|_o$, define $o \cdot H = \{\langle q, o \cdot p\rangle \mid \langle q, p\rangle \in H\}$. When a set H of so-pairs is written as $H = H_1 \cup o \cdot H_2$, or $H = H_1 \cup \{\langle q, o\rangle\}$, we assume that every so-pair in H_1 has an occurrence disjoint with o. For example, let $t = f(g(a), f(b, x))$ then $(\{\langle q_1, 1\rangle, \langle q_2, 2 \cdot 1\$\rangle, \langle q_3, 2 \cdot 2\rangle\}, t)$ is an ID for t. Intuitively, this ID corresponds to the situation illustrated in Figure 2. The second head from the left has read the leaf b at occurrence $2 \cdot 1$, and has no function symbol to read.

A *move* of M for a term t is represented by a binary relation "$\vdash$" on ID's of M for t, which is defined as follows: For an ID $(H \cup \{\langle q, o\rangle\}, t)$ $(o \in \mathcal{O}(t))$,

- if $(q_1) \in \delta(q, \varepsilon)$, then $(H \cup \{\langle q, o\rangle\}, t) \vdash (H \cup \{\langle q_1, o\rangle\}, t)$,
- if $(q_1, \ldots, q_{a(f)}) \in \delta(q, f)$, $a(f) \geq 1$ and the function symbol of t at the occurrence o is f, then $(H \cup \{\langle q, o\rangle\}, t) \vdash (H \cup \{\langle q_1, o \cdot 1\rangle, \ldots, \langle q_n, o \cdot n\rangle\}, t)$, and
- if $(q_1) \in \delta(q, c)$ and $t|_o$ is a constant c, then $(H \cup \{\langle q, o\rangle\}, t) \vdash (H \cup \{\langle q_1, o\$\rangle\}, t)$.

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$. If $I_1 \vdash^* I_2$, then I_2 is *reachable* from I_1.

An so-pair $\langle q, p\rangle$ is called an *accepting so-pair* if $q \in Q_F$ and $p \in O_E(t)$. An so-pair $\langle q, p\rangle$ is called a *variable scanning so-pair* if $t|_p$ is a variable (the state q does not have to belong to Q_F). An ID (H, t) is called a *partial accepting ID* if every element in H is either an accepting so-pair or a variable scanning so-pair. A partial accepting ID (H, t) is said to be *minimal* if there is no other partial accepting ID (H', t) such that $(H', t) \vdash^* (H, t)$. If any partial accepting ID of t is reachable from $(\{\langle q_0, \lambda\rangle\}, t)$, the *initial ID* of M for t, then we say M *partially accepts* the term t. If t is a ground term and (H, t) is a partial accepting ID of t, then every element in H is an accepting so-pair (since t has no variable). In this case, (H, t) is called an *accepting ID*, and if an accepting ID for t is reachable from the initial ID of M then we say M *accepts* t.

The *tree language accepted by* M, denoted $L(M)$, is defined to be a set of ground terms which are accepted by M. A set T of terms is a *regular tree language*[5] if there is a tree automaton M such that $T = L(M)$. It is known[5] that the class of regular tree languages is closed under intersection, and the emptiness problem for a regular tree language is decidable.

Example 2.1 [5]**:** Let $M = (\{q_0, q_1, q_2, q_f\}, \{+, 0, 1, 2\}, \phi, \delta, q_0, \{q_f\})$ where $a(+) = 2$ and $a(0) = a(1) = a(2) = 0$. δ is defined as follows:

$$\begin{aligned}
\delta(q_0, +) &= \{(q_0, q_0), (q_1, q_2), (q_2, q_1)\} \\
\delta(q_1, +) &= \{(q_0, q_1), (q_1, q_0), (q_2, q_2)\} \\
\delta(q_2, +) &= \{(q_0, q_2), (q_1, q_1), (q_2, q_0)\} \\
\delta(q_0, 0) &= \delta(q_1, 1) = \delta(q_2, 2) = \{(q_f)\}.
\end{aligned}$$

M accepts an input $t = +(+(2, 0), +(0, 1))$ since

$$\begin{aligned}
&(\{\langle q_0, \lambda\rangle\}, t) \vdash (\{\langle q_2, 1\rangle, \langle q_1, 2\rangle\}, t) \\
&\vdash^* (\{\langle q_2, 1 \cdot 1\rangle, \langle q_0, 1 \cdot 2\rangle, \langle q_0, 2 \cdot 1\rangle, \langle q_1, 2 \cdot 2\rangle\}, t) \\
&\vdash^* (\{\langle q_f, 1 \cdot 1\$\rangle, \langle q_f, 1 \cdot 2\$\rangle, \langle q_f, 2 \cdot 1\$\rangle, \langle q_f, 2 \cdot 2\$\rangle\}, t)
\end{aligned}$$

□

As for moves of tree automata, the following lemmas hold.

Lemma 2.1: If $(H \cup \{\langle q, o\rangle\}, t) \vdash^* (H' \cup o \cdot H'', t)$, then $(\{\langle q, \lambda\rangle\}, t|_o) \vdash^* (H'', t|_o)$. Furthermore, if $(H' \cup o \cdot H'', t)$ is a partial accepting ID of t, then $(H'', t|_o)$ is a partial accepting ID of $t|_o$. □

Lemma 2.2: If $(H,t)\vdash^*(H' \cup \{\langle q,o\rangle\},t)$ for $o \in \mathcal{O}(t)$, then $(H,t[o \leftarrow s])\vdash^*(H' \cup \{\langle q,o\rangle\},t[o \leftarrow s])$ for any $s \in \mathcal{T}_F[X]$. □

Lemma 2.3: If $(H,t)\vdash^*(H' \cup \{\langle q,o\rangle\},t)$ and $(\{\langle q,\lambda\rangle\},\ s)\vdash^*(H'',s)$, then $(H,t[o \leftarrow s])\vdash^*(H' \cup o \cdot H'',t[o \leftarrow s])$. □

Lemma 2.4: Let t be a term which has m variables $x_1,\ldots,x_m$ at occurrences $o_1,\ldots,o_m$, respectively, and let $\sigma = \{x_i \mapsto t_i \mid 1 \le i \le m\}$. If $(H,t)\vdash^*(H' \cup \{\langle q_i,o_i\rangle \mid 1 \le i \le m\},t)$ and $(\{\langle q_i,\lambda\rangle\},t_i)\vdash^*(H_i,t_i)$ for $1 \le i \le m$, then $(H,t\sigma)\vdash^*(H' \cup \bigcup_{i=1}^n o_i \cdot H_i, t\sigma)$. Furthermore, if $(H' \cup \{\langle q_i,o_i\rangle \mid 1 \le i \le m\},t)$ and (H_i,t_i) $(1 \le i \le m)$ are partial accepting ID's of t and t_i $(1 \le i \le m)$, respectively, then $(H' \cup \bigcup_{i=1}^n o_i \cdot H_i, t\sigma)$ is a partial accepting ID of $t\sigma$. □

The *tree language accepted by* M, denoted $L(M)$, is defined to be a set of ground terms which are accepted by M. A set T of terms is a *regular tree language*[5] if there is a tree automaton M such that $T = L(M)$. It is known[5] that the class of regular tree languages is closed under intersection, and the emptiness problem for a regular tree language is decidable.

3 Unification Problem under Constrained Substitutions

A generalization of unification problem, named a *unification problem under constrained substitutions* is introduced in this section.

For a finite set A of terms and a set X of variables, define a set $\mathcal{T}_A[X]$ of terms to be the minimal set satisfying the followings.

- $\mathcal{T}_A[X]$ contains every variable $x \in X$.
- For a term $t \in A$ with $\mathcal{V}(t) = \{x_1,\ldots,x_m\}$ and terms $t_1,\ldots,t_m \in \mathcal{T}_A[X]$, the term $t\sigma$ is in $\mathcal{T}_A[X]$ where $\sigma = \{x_i \mapsto t_i \mid 1 \le i \le m\}$.

A term in A is called an *available term*, and a term in $\mathcal{T}_A[X]$ is called an *A-term*.

A substitution $\sigma = \{x_i \mapsto t_i \mid 1 \le i \le m\}$ is called an *A-constrained substitution* (*A-substitution* for short) if $t_i \in \mathcal{T}_A[X]$ for $1 \le i \le m$. A term t is said to be an *A-constrained instance of s* (*A-instance of s*) if there is an A-substitution σ such that $t = s\sigma$.

We say two terms s and t are *R-unifiable under A-substitutions* if there are an A-substitution σ and a term u such that $s\sigma \Rightarrow_R^* u$ and $t\sigma \Rightarrow_R^* u$. We also define an *$R$-unification problem under A-substitutions* to be the one to decide whether two given

terms are R-unifiable under A-substitutions. A *unification problem for a class* $\mathcal{R}$ *under constrained substitutions* $\mathcal{A}$ is the one to decide, for a given TRS $R \in \mathcal{R}$, a set of available terms $A \in \mathcal{A}$, and a pair of terms s and t, whether s and t are R-unifiable under A-substitutions.

Example 3.1: Let $F = \{add, succ, zero\}$, $a(add) = 2$, $a(succ) = 1$, $a(zero) = 0$, $R = \{add(x, succ(y)) \to succ(add(x, y)), add(x, zero) \to x\}$ and $A = \{succ(succ(x)), zero\}$. The set $\mathcal{T}_A[X]$ of A-terms includes terms of the form $\underbrace{succ(\cdots succ(}_{2m} zero)\cdots)$ where m is a nonnegative integer. Two terms $add(x, succ(succ(zero)))$ and y is R-unifiable under A-substitutions. For instance, they R-unify to $succ(succ(zero))$ by an A-substitution $\sigma = \{x \mapsto zero, y \mapsto succ(succ(zero))\}$. On the other hand, two terms $add(x, succ(zero))$ and y are not R-unifiable under A-substitutions, though they are R-unifiable in the usual unification problem. □

For a set F of function symbols, if A is defined to be $A = \{f(x_1, \ldots, x_{a(f)}) \mid f \in F\}$, then $\mathcal{T}_A[X]$ coincides with $\mathcal{T}_F[X]$. That is, the usual unification problem is a special case of the unification problem under constrained substitutions, and thereby, a procedure that solves the unification problem under constrained substitutions can solve the usual unification problem too.

4 The Unification Procedure

In this section, a procedure to solve the unification problem under constrained substitutions is investigated. For a set A of available terms and a goal $G = \{g_1, g_2\}$, define $A_0 = \{t \mid t$ is a subterm of a term in $A \cup G\}$ [3]. For a term $t \in A_0$, let $S_0(t)$ be the set of ground A-instances of t, and $S_*(t)$ be the set of terms which are obtained by rewriting terms in $S_0(t)$ by using rewrite rules in R. In Subsection 4.1, for each $t \in A_0$, a tree automaton $M_0(t)$ such that $L(M_0(t)) = S_0(t)$ is constructed . In Subsection 4.2, these tree automata are modified into new automata $M_*(t)$ such that $L(M_*(t)) = S_*(t)$. Hence, two terms g_1 and g_2 are R-unifiable under constrained substitutions if and only if the set $L(M_*(g_1)) \cap L(M_*(g_2))$ is not empty. Unfortunately, our strategy cannot be applied to the class of all rewrite systems because of the inherent semi-decidability of the unification problem and of the fact that tree automata

[3] literal similar terms are treated as identical.

do not have strong computational power. In Subsection 4.1, a sufficient condition is proposed for our procedure to be a semi-decision procedure for the problem. That is, under the condition, if goal terms are unifiable then the procedure terminates and answers "unifiable", and if they are not unifiable then it can happen that the procedure does not terminate. In Subsection 4.3, a sufficient condition is proposed for our procedure to be a decision procedure, that is, our procedure always terminates under the condition, and outputs correct answer.

4.1 A-Instances and Tree Automata

Let R and A be TRS and a set of available terms, respectively, which satisfy the following Conditions 4.1 through 4.4.

Condition 4.1: Goal terms g_1, g_2, and terms in A are linear, and $\mathcal{V}(g_1) \cap \mathcal{V}(g_2) = \phi$. □

Condition 4.2: TRS R satisfies Church-Rosser property[1]. □

Condition 4.3: Every rule in R is right-linear. □

Condition 4.4: Every variable occurring in the right-hand side of a rule occurs in the left-hand side only once. For example, a rule $f(x, x, y) \to x$ does not satisfy this condition, while $f(x, x, y) \to y$ does. □

The authors conjecture that the unification problem under constrained substitutions will be undecidable under these conditions only. A condition for the problem to be decidable is discussed in Subsection 4.3.

In this section, for each $t \in A_0$, a tree automaton $M_0(t)$ which accepts ground A-instances of t is constructed under Condition 4.1. In the following discussion, we are interested in occurrences of variables in a term. Names of variables are not important since no variable occurs in a term more than once by Condition 4.1, and hence every variable in a term can be replaced independently to each other. For a term $t \in \mathcal{T}_F[X]$, define $\tilde{t}$ be a term which is obtained from t by replacing all variables of t with one variable x. For example, if $t = f(x_1, g(x_2, x_2))$ then $\tilde{t} = f(x, g(x, x))$. Let $\tilde{A} = \{\tilde{t} \mid t \in A\}$ and $\tilde{A}_0 = \{\tilde{t} \mid t \in A_0\}$. Note that there is a one-to-one correspondence between terms in A_0 and those in $\tilde{A}_0$ since terms in A_0 are linear.

Components of tree automata are defined as follows:

- $Q_0 = \{[t] \mid t \in \tilde{A}_0\} \cup \{q_{fin}\}$ and $Q_F = \{q_{fin}\}$. That is, corresponding to each term $t \in \tilde{A}_0$, there is a state denoted by $[t]$.
- F is identical to that of R, and $X = \{x\}$.
- A transition function δ_0 is given as follows:
  - For any constant $c \in \tilde{A}_0$, $(q_{fin}) \in \delta_0([c], c)$.
  - For a term $t = f(t_1, \ldots, t_n) \in \tilde{A}_0$, $([t_1], \ldots, [t_n]) \in \delta_0([t], f)$.
  - For the variable x, $([t]) \in \delta_0([x], \varepsilon)$ for every $t \in \tilde{A}$.

For $t \in A_0$, a tree automaton $M_0(t)$ is defined to be $M_0(t) = (Q_0, F, X, \delta_0, [\tilde{t}\,], Q_F)$.

As for tree automata defined above, the following theorem holds. Its proof is omitted since it is easily understood by the construction.

Theorem 4.1: $L(M_0(t)) = S_0(t)$. That is, $s \in S_0(t)$ *iff* $M_0(t)$ accepts s. □

4.2 Rewrite Rules and Modification of the Transition Function

In this section, a procedure to construct a tree automaton which accepts terms in $S_*(t)$ for each $t \in A_0$ is introduced under Conditions 4.1 through 4.4. First, the tree automata $M_0(t)$ defined in the previous section are constructed, then they are modified to accept terms in $S_*(t)$.

The idea behind this modification can be easily understood if one consider a string-to-string substitution on a regular (string) language. Let $M = (Q, T, q_0, \delta, Q_F)$ be a string automaton, α and β be strings in T^*. For every $q_1, q_2 \in Q$ such that $q_2 \in \delta(q_1, \alpha)$, adding new transition to δ to satisfy $q_2 \in \delta(q_1, \beta)$ will results another automaton M' such that

$$L(M') = \{w \mid \text{there is } w_1\alpha\cdots w_{n-1}\alpha w_n \in L(M),\ w_i \in T^*\ (1 \le i \le n),\\ \text{and } w = w_1\beta\cdots w_{n-1}\beta w_n\}.$$

The modification of tree automata in the following Procedure 4.1 is basically based on this idea except the following differences:

1. We must consider tree-to-tree substitutions which correspond to rewrite rules. In the case of string-to-string substitutions, we deal with only one state, while in the tree-to-tree case we must take care of many states that correspond to leaves of trees.
2. Handling with tree-to-tree substitutions that correspond to left-nonlinear rules should be "restricted" according to the non-linearity. We construct sets UT^j ($j \ge 1$) of j-tuples of unifiable terms, and control the applications of these substitutions.

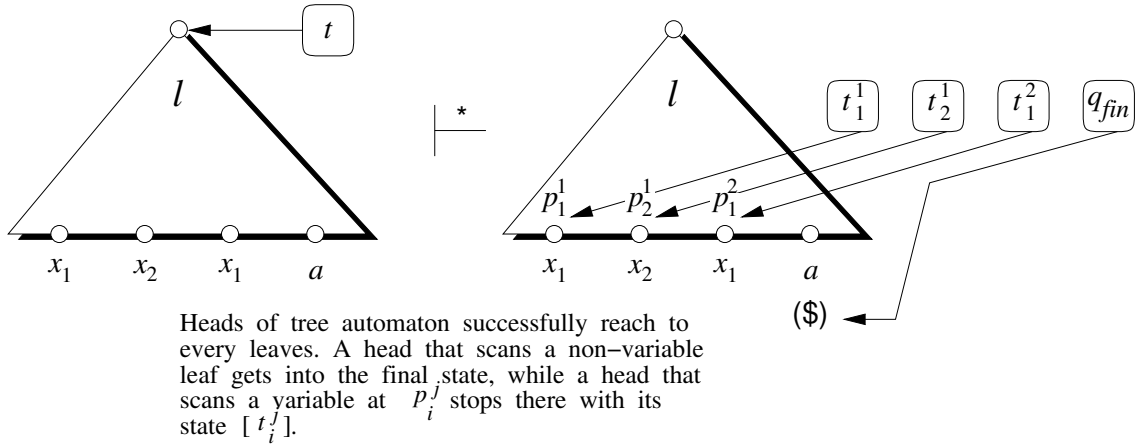


Figure 3: $M_k(t)$ partially accepts l

A formal description of the procedure is given as follows.

Procedure 4.1:

Input: TRS R, a set A of available terms, and a goal $G = \{g_1, g_2\}$ which satisfy Conditions 4.1 through 4.4. Let γ be the non-negative integer such that, for every rule $l \to r \in R$, no variable has more than γ occurrences in l.

Output: "*unifiable*" or "*not unifiable*".

Step 1. Let $k := 0$ and construct tree automata $M_0(t) = (Q_0, F, \{x\}, \delta_0, [\tilde{t}\,], \{q_{fin}\})$ for $t \in A_0$ as in the previous section. Furthermore, for each j $(1 \leq j \leq \gamma)$, let

$$\mathrm{UT}_0^j := \{(t_1, \ldots, t_j) \mid t_1, \ldots, t_j \in A_0, L(M_0(t_1)) \cap \cdots \cap L(M_0(t_j)) \neq \phi\}.$$

Step 2. Let $A_{k+1} := A_k$, $\tilde{A}_{k+1} := \tilde{A}_k$, $Q_{k+1} := Q_k$, $\delta_{k+1} := \delta_k$, and $k := k + 1$.

Step 3. Let $l \to r$ be a rewrite rule in R such that

- l has $m(\geq 0)$ variables $x_1, \ldots, x_m$, and
- the variable x_i $(1 \leq i \leq m)$ occurs in l at occurrences $p_i^1, \ldots, p_i^{\gamma_i} \in \mathcal{O}(l)$ $(1 \leq \gamma_i < \gamma)$.

If there are a set H of accepting so-pairs of l, terms t and t_i^j in A_k $(1 \leq i \leq m, 1 \leq j \leq \gamma_i)$ such that $(t_i^1, \ldots, t_i^{\gamma_i}) \in \mathrm{UT}_k^{\gamma_i}$ $(1 \leq i \leq m)$,

$$(\{\langle [\tilde{t}\,], \lambda \rangle\}, l) \vdash^*_{M_k(t)} (H \cup \{\langle [\tilde{t}_i^j], p_i^j \rangle | 1 \leq i \leq m, 1 \leq j \leq \gamma_i\}, l), \qquad (1)$$

(Figure 3) and the right-hand side of (1) is the minimal partial accepting ID of l, then let $\rho = \{x_i \mapsto t_i^1 \mid x_i$ occurs in r (and hence $\gamma_i = 1$ by Condition 4.4)$\}$ and do the following 3a through 3b:

(a) For every subterm s of $r\rho$, let $A_{k+1} := A_{k+1} \cup \{s\}$ [4], $\tilde{A}_{k+1} := \tilde{A}_{k+1} \cup \{\tilde{s}\}$, and $Q_{k+1} := Q_{k+1} \cup \{[\tilde{s}]\}$.

(b) Add an ε-transition $([\tilde{r\rho}]) \in \delta_{k+1}([\tilde{t}], \varepsilon)$. If a move between ID's is caused by this function value, then the move is called a *rewriting move*.

(c) Execute **ADDTRANS**($r\rho$,k+1), which defines transitions for newly introduced states that correspond to subterms of $r\rho$. See below for the formal description of **ADDTRANS**.

Step 4. For $t \in A_{k+1}$, define $M_{k+1}(t) = (Q_{k+1}, F, \{x\}, \delta_{k+1}, [\tilde{t}], \{q_{fin}\})$.

Step 5. For each j $(1 \leq j \leq \gamma)$, let

$$\mathrm{UT}^j_{k+1} := \{(t_1, \ldots, t_j) \mid t_1, \ldots, t_j \in A_{k+1}, L(M_{k+1}(t_1)) \cap \cdots \cap L(M_{k+1}(t_j)) \neq \phi\}.$$

Step 6. If $A_{k+1} \neq A_k$, or $\delta_{k+1} \neq \delta_k$, or $\mathrm{UT}^j_{k+1} \neq \mathrm{UT}^j_k$ for some j $(1 \leq j \leq \gamma)$, then go to Step 2.

Step 7. Let $M_*(t) := M_k(t)$ and let $\mathrm{UT}^j := \mathrm{UT}^j_k$ for $1 \leq j \leq \gamma$. Output "*unifiable*" if $(g_1, g_2) \in \mathrm{UT}^2$, "*not unifiable*" otherwise.

The subroutine **ADDTRANS** takes a term t and an integer k as its input, and modifies δ_k as follows:

Case 1. If t is a constant, then let $(q_{fin}) \in \delta_k([t], t)$.

Case 2. If t is the variable x, then do nothing (since transitions for the variable are already defined).

Case 3. If $t = f(t_1, \ldots, t_m)$, then let $([\tilde{t_1}], \ldots, [\tilde{t_m}]) \in \delta_k([\tilde{t}], f)$ and recursively execute **ADDTRANS**(t_i, k) for $1 \leq i \leq m$. □

Correctness of the Procedure

From the construction of the tree automata in Procedure 4.1, it is easy to see that inclusion hierarchies $L(M_0(t)) \subseteq L(M_1(t)) \subseteq \cdots$ holds for $t \in A_0(t)$. Furthermore, by Theorem 4.1 and by the construction of the tree automata, the following lemma holds.

Lemma 4.2: If $M_k(t)$ $(k \geq 0)$ accepts a term s without rewriting move, then $s \in S_0(t)$, and vice versa. □

[4] If A_{k+1} has contained a term which is literal similar to s, then do nothing. Since literal similar terms have the same set of R-instances by Condition 4.1, and thereby they do not have to be distinguished.

The following lemma implies the soundness of Procedure 4.1 in the sense that if $M_k(t)$ accepts a term, then the term can be obtained by rewriting an A-instance of t.

Lemma 4.3: For a non-negative integer k, $L(M_k(t)) \subseteq S_*(t)$ holds.

Proof. We show by induction that if $M_k(t)$ accepts a ground term s, then $s \in S_*(t)$. For the basis $k = 0$, $M_0(t)$ accepts terms in $S_0(t)$ by Theorem 4.1. Since $S_0(t) \subseteq S_*(t)$ the lemma holds.

Assume that the lemma holds for $k-1$ or less, and consider that $M_k(t)$ accepts s, i.e. there is a sequence α of moves from the initial ID of $M_k(t)$ to an accepting ID for s. This inductive part is shown by induction on the number of rewriting moves in this sequence α. For the basis, there is no rewriting move in α. By Lemma 4.2, $s \in S_0(t)$ and the lemma holds.

Assume that the lemma holds for every sequence with $n-1$ or less rewriting moves, and consider a sequence of moves which contains n rewriting moves. Such a sequence can be written as

$$(\{\langle[\tilde{t}], \lambda\rangle\}, s) \vdash^*_{M_k(t)} (H \cup \{\langle[\tilde{t_l}], o\rangle\}, s) \qquad (t_l \in A_k, o \in \mathcal{O}(s)) \tag{2}$$

$$\vdash_{M_k(t)} (H \cup \{\langle[\tilde{t_r}], o\rangle\}, s) \qquad (t_r \in A_k) \tag{3}$$

$$\vdash^*_{M_k(t)} (H_0 \cup o \cdot H', s) \tag{4}$$

where the move from (2) to (3) is the last rewriting move in α, and hence there is no rewriting move between (3) to (4).

Corresponding to the rewriting move between (2) to (3), there is the rewrite rule $l \to r$ and the substitution ρ which are used to define the function value used for this rewriting move. Without loss of generality, assume that

- l has m variables $x_1, \ldots, x_m$,
- the variable x_i $(1 \leq i \leq m)$ occurs in l at occurrences $p_i^1, \ldots, p_i^{\gamma_i} \in \mathcal{O}(l)$, and
- if the variable x_i $(1 \leq i \leq m)$ occurs in r, then it occurs at $o_i \in \mathcal{O}(r)$.

By the last assumption, the ID (4) can be written as

$$(H_0 \cup o \cdot H', s) = (H'_0 \cup \{o \cdot o_i \cdot H_i \mid x_i \text{ occurs in } r\}, s). \tag{5}$$

Furthermore, it is easily understood from the construction of transition function in ADDTRANS that, when a scanning head of the tree automata $M_k(t)$ passes the occurrence $o \cdot o_i$, the state attached to the head is $[\tilde{t}_i^1]$. That is, the sequence of moves

from (3) to (4) contains an ID of the form

$$(H_i'' \cup \{\langle[\tilde{t}_i^1], o \cdot o_i\rangle\}, s) \quad (x_i \text{ occurs in } r), \tag{6}$$

and this ID is reachable to the accepting ID (5). Remark that by Lemma 2.1, the following sequence of moves is possible.

$$(\{\langle[\tilde{t}_i^1], \lambda\rangle\}, s|_{o \cdot o_i}) \vdash^*_{M_k(t)} (H_i, s|_{o \cdot o_i}) \quad (x_i \text{ occurs in } r). \tag{7}$$

By the construction in Step 3, the followings hold.

- There are terms $t_i^j \in A_{k-1}$ $(1 \leq i \leq m, 1 \leq j \leq \gamma_i)$, and the following sequence of moves is possible:

$$(\{\langle[\tilde{t}_l], \lambda\rangle\}, l) \vdash^*_{M_{k-1}(t_l)} (H_0' \cup \{\langle[\tilde{t}_i^j], p_i^j\rangle \mid 1 \leq i \leq m, 1 \leq j \leq \gamma_i\}, l) \tag{8}$$

  where the right-hand side is a partial accepting ID of l.
- For $1 \leq i \leq m$, $(t_i^1, \ldots, t_i^{\gamma_i}) \in \mathrm{UT}_{k-1}^{\gamma_i}$.
- The substitution ρ can be written as $\rho = \{x_i \mapsto t_i^1 \mid x_i \text{ occurs in } r\}$, and $t_r = r\rho$.

We show that there is a term s' such that $M_k(t)$ accepts s' with $n-1$ rewriting moves (and hence $s' \in S_*(t)$ by the inductive hypothesis), and $s' \Rightarrow^* s$ by using the rule $l \to r$ at the occurrence o. Let $l' = l[p_i^j \leftarrow s_i^j \mid 1 \leq i \leq m, 1 \leq j \leq \gamma_i]$ where s_i^j $(1 \leq i \leq m, 1 \leq j \leq \gamma_i)$ are terms defined as follows (see also Figure 4):

- For x_i which does not occur in r; since $(t_i^1, \ldots, t_i^{\gamma_i}) \in \mathrm{UT}_{k-1}^{\gamma_i}$, there is a term s_i such that all of $M_{k-1}(t_i^j)$ $(1 \leq j \leq \gamma_i)$ accepts s_i. Define s_i^j to be a ground A-instance of t_i^j such that $s_i^j \Rightarrow^* s_i$. Since s_i^j is a ground A-instance of t_i^j, there is an accepting ID (H_i^j, s_i^j) of s_i^j such that

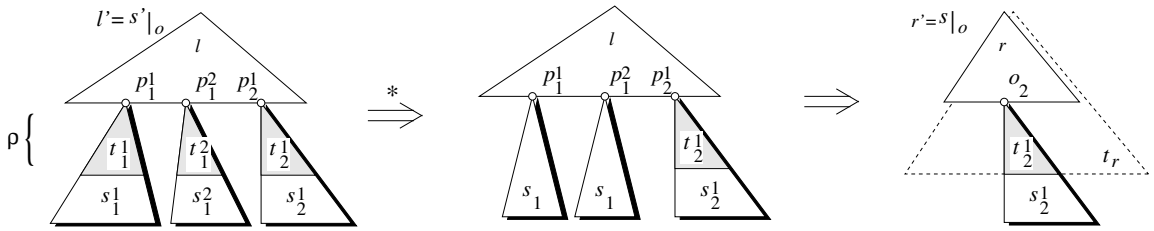
$$(\{\langle[\tilde{t}_i^j], \lambda\rangle\}, s_i^j) \vdash^*_{M_k(t)} (H_i^j, s_i^j) \qquad (1 \leq j \leq \gamma_i) \tag{9}$$

  and this sequence does not contain rewriting moves.
- For x_i which occurs in r; in this case $\gamma_i = 1$ by Condition 4.4. Define $s_i^1 = s|_{o \cdot o_i}$. For the simplicity of the discussion, we also let $s_i = s_i^1$ and $H_i^1 = H_i$. Thereby, (7) can be written as

$$(\{\langle[\tilde{t}_i^1], \lambda\rangle\}, s_i^1) \vdash^*_{M_k(t)} (H_i^1, s_i^1) \tag{10}$$

By (8), (9), (10) and Lemma 2.3,

$$\begin{aligned}(\{\langle[\tilde{t}_l], \lambda\rangle\}, l') &\vdash^*_{M_k(t)} (H_0' \cup \{\langle[\tilde{t}_i^j], p_i^j\rangle \mid 1 \leq i \leq m, 1 \leq j \leq \gamma_i\}, l') \\ &\vdash^*_{M_k(t)} (H_0' \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{\gamma_i} p_i^j \cdot H_i^j, l')\end{aligned}$$



$m = 2$, $\gamma_1 = 2$, $\gamma_2 = 1$, and only x_2 occurs in r.

Figure 4: Rewrite rule and subterms.

where the last ID is the accepting ID of l', and this sequence does not contain rewriting move. Define $s' = s[o \leftarrow l']$. Then

$$(\{\langle[\tilde{t}], \lambda\rangle\}, s') \vdash^*_{M_k(t)} (H_0 \cup \{\langle[\tilde{t}_l], o\rangle\}, s')$$
$$\vdash^*_{M_k(t)} (H_0 \cup o \cdot H'_0 \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{\gamma_i} o \cdot p_i^j \cdot H_i^j, s').$$

The last ID is the accepting ID of s', and this sequence contains $n-1$ or less rewriting moves. By the inductive hypothesis, $s' \in S_*(t)$.

Next, we show that $s' \Rightarrow^* s$. By the fact that there is no rewriting moves in the sequence from (3) to (4), and by Lemmas 2.1 and 4.2, it follows that $s|_o$ is a ground A-instance of t_r. Since $t_r = r\rho$, $s|_o$ is also a ground A-instance of r. If we define $r' = r[o_i \leftarrow s_i \mid x_i \text{ occurs in } r]$, then $s|_o = r'$ holds by the definition of s_i^1. Since $s_i^j \Rightarrow^* s_i$ for $1 \le i \le m$ and $1 \le j \le \gamma_i$,

$$\begin{aligned} l' &= l[p_i^j \leftarrow s_i^j \mid 1 \le i \le m, 1 \le j \le \gamma_i] \\ &\Rightarrow^* l[p_i^j \leftarrow s_i \mid 1 \le i \le m, 1 \le j \le \gamma_i] \\ &\Rightarrow r[o_i \leftarrow s_i \mid x_i \text{ occurs in } r] = r'. \end{aligned}$$

Since $s' = s[o \leftarrow l']$ and $s|_o = r'$, $s' \Rightarrow s$ and hence $s \in S_*(t)$. □

The following lemma implies the completeness of Procedure 4.1 in the sense that if a term can be obtained by rewriting an A-instance of t, then the term is accepted by $M_k(t)$ for some k

Lemma 4.4: For $s \in S_*(t)$, there is an integer k such that $s \in L(M_k(t))$ (and hence $s \in L(M_{k'}(t))$ for $k' \ge k$).

Proof. By the definition of $S_*(t)$, there is an A-instance s_0 of t, and s is obtained from s_0 by n (≥ 0) rewritings. The lemma is shown by induction on the number n

of rewritings. For the basis $n = 0$, the lemma holds clearly since $s = s_0 \in S_0(t)$ and thereby $s \in L(M_0(t))$.

Assume that the lemma holds for every term which is obtained by $n-1$ or less rewritings of an A-instance of any term, and consider the case that s is obtained by n rewritings of $s_0 \in S_0(t)$. In this case, there are a term s'' such that $s_0 \Rightarrow^* s'' \Rightarrow s$, a rewrite rule $l \to r$, a substitution σ, and an occurrence o of s'' such that $s''|_o = l\sigma$ and $s = s''[o \leftarrow r\sigma]$. Without loss of generality, assume that

- l has m variables $x_1, \ldots, x_m$,
- the variable x_i $(1 \le i \le m)$ occurs in l at occurrences $p_i^1, \ldots, p_i^{\gamma_i} \in \mathcal{O}(l)$, and
- if the variable x_i $(1 \le i \le m)$ occurs in r, then it is at an occurrence $o_i \in \mathcal{O}(r)$.

Since $s''|_o = l\sigma$ and $s|_o = r\sigma$, the followings hold.

- For i $(1 \le i \le m)$, there is a term s_i such that $s_i = s''|_{o \cdot p_i^1} = \cdots = s''|_{o \cdot p_i^{\gamma_i}}$.
- The substitution σ can be written as

$$\sigma = \{x_i \mapsto s_i \mid 1 \le i \le m\}. \tag{11}$$

- If x_i occurs in r then $s|_{o \cdot o_i} = s_i$.

By the inductive hypothesis, there is an integer k'' such that $M_{k''}(t)$ accepts s''. The accepting ID of s'' can be written as $(H_0 \cup o \cdot H_0' \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{\gamma_i} o \cdot p_i^j \cdot H_i, s'')$, and there exist terms t_l and t_i^j $(1 \le i \le m, 1 \le j \le \gamma_i)$ in A_* such that

$$\begin{aligned}
&(\{\langle [\tilde{t}], \lambda \rangle\}, s'') \vdash^*_{M_{k''}(t)} (H_0 \cup \{\langle [\tilde{t}_l], o \rangle\}, s'') \\
&\vdash^*_{M_{k''}(t)} (H_0 \cup o \cdot H_0' \cup \{\langle [\tilde{t}_i^j], o \cdot p_i^j \rangle \mid 1 \le i \le m, 1 \le j \le \gamma_i\}, s'') \\
&\vdash^*_{M_{k''}(t)} (H_0 \cup o \cdot H_0' \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{\gamma_i} o \cdot p_i^j \cdot H_i, s'').
\end{aligned} \tag{12}$$

By Lemma 2.1,

$$(\{\langle [\tilde{t}_l], \lambda \rangle\}, s''|_o) \vdash^*_{M_{k''}(t)} (H_0' \cup \{\langle [\tilde{t}_i^j], p_i^j \rangle \mid 1 \le i \le m, 1 \le j \le \gamma_i\}, s''|_o), \tag{13}$$

and for $1 \le i \le m, 1 \le j \le \gamma_i$, $(\{\langle [\tilde{t}_i^j], \lambda \rangle\}, s''|_{o \cdot p_i^j}) \vdash^* (H_i, s''|_{o \cdot p_i^j})$. Since $s_i = s''|_{o \cdot p_i^1} = \cdots = s''|_{o \cdot p_i^{\gamma_i}}$, this move can be written as

$$(\{\langle [\tilde{t}_i^j], \lambda \rangle\}, s_i) \vdash^*_{M_{k''}(t)} (H_i, s_i) \qquad (1 \le i \le m, 1 \le j \le \gamma_i). \tag{14}$$

This implies that all of $M_{k''}(t_i^j)$ $(1 \le j \le \gamma_i)$ accepts s_i. Hence,

$$(t_i^1, \ldots t_i^{\gamma_i}) \in \mathrm{UT}_{k''}^{\gamma_i} \qquad (1 \le i \le m). \tag{15}$$

Remark that $s''|_o = l\sigma$ and hence $l = (s''|_o)[p_i^j \leftarrow x_i \,|1 \le i \le m, 1 \le j \le \gamma_i]$. By (13) and Lemma 2.2,

$$(\{\langle[\tilde{t}_l], \lambda\rangle\}, l) \vdash^*_{M_{k''}(t)} (H_0' \cup \{\langle[\tilde{t}_i^j], p_i^j\rangle \,|1 \le i \le m, 1 \le j \le \gamma_i\}, l). \tag{16}$$

By (16), (15), and Step 3 of Procedure 4.1, defining $\rho = \{x_i \mapsto t_i^1 \,| x_i \text{ occurs in } r\}$,

$$([\tilde{r\rho}]) \in \delta_{k''+1}([\tilde{t}_l], \varepsilon). \tag{17}$$

By the construction in **ADDTRANS**, the following sequence of moves of $M_{k''+1}$ is possible:

$$(\{\langle[\tilde{r\rho}], \lambda\rangle\}, r) \vdash^*_{M_{k''+1}} (H_0'' \cup \{\langle[\tilde{t}_i^1], o_i\rangle \,| x_i \text{ occurs in } r\}, r) \tag{18}$$

where each element in H_0'' is an accepting so-pair. By the fact that $r|_{o_i} = s_i$, equations (18), (11), (14) and Lemma 2.4,

$$(\{\langle[\tilde{r\rho}], \lambda\rangle\}, r\sigma) \vdash^*_{M_{k''+1}} (H_0'' \cup \{o_i \cdot H_i \,| x_i \text{ occurs in } r\}, r\sigma). \tag{19}$$

Note that the last ID is the accepting ID of $r\sigma$.

Since $s = s''[o \leftarrow r\sigma]$, and by (12), (17) and (19),

$$\begin{aligned}
(\{\langle[\tilde{t}], \lambda\rangle\}, s) &\vdash^*_{M_{k''+1}} (H_0 \cup \{\langle[\tilde{t}_l], o\rangle\}, s) \\
&\vdash_{M_{k''+1}} (H_0 \cup \{\langle[\tilde{r\sigma}], o\rangle\}, s) \\
&\vdash^*_{M_{k''+1}} (H_0 \cup o \cdot H_0'' \cup \{o \cdot o_i \cdot H_i \,| x_i \text{ occurs in } r\}, s)
\end{aligned}$$

Note that the last ID is the accepting ID of s, and hence $M_{k''+1}(t)$ accepts s. □

Theorem 4.5: Terms g_1 and g_2 are R-unifiable under A-substitution *iff* there is an integer k such that $(g_1, g_2) \in \mathrm{UT}_k^2$.

Proof. If g_1 and g_2 are R-unifiable under A-substitution, then there are a substitution σ and a term g such that $g_1\sigma \Rightarrow_R^* g$ and $g_2\sigma \Rightarrow_R^* g$, and hence $g \in S_*(g_1)$ and $g \in S_*(g_2)$. By Lemma 4.4, there are integers k_1 and k_2 such that $M_{k_1}(g_1)$ and $M_{k_2}(g_2)$ accept g. Let $k = \max\{k_1, k_2\}$ then both of $M_k(g_1)$ and $M_k(g_2)$ accept g and hence $(g_1, g_2) \in \mathrm{UT}_k^2$. The if part can be easily shown by using Lemma 4.3. □

If Procedure 4.1 halts, then there is an integer k such that $L(M_0(t)) \subseteq L(M_1(t)) \subseteq \cdots \subseteq L(M_k(t)) = L(M_{k+1}(t)) = \cdots$ and $M_*(t)$ is defined to be $M_k(t)$. In this case, the following corollary holds.

Corollary 4.6: If Procedure 4.1 halts, then $S_*(t) = L(M_*(t))$, where $M_*(t)$ is obtained at Step 7 in Procedure 4.1. Thereby, goals g_1 and g_2 are R-unifiable under A-substitutions *iff* Procedure 4.1 outputs "*unifiable*". □

Furthermore, if Procedure 4.1 is modified to the following Procedure 4.2, then Corollary 4.7 holds.

Procedure 4.2: Almost the same as Procedure 4.1 except that the following step is inserted between Steps 2 and 3 of Procedure 4.1.

Step 3.5 If $(g_1, g_2) \in \mathrm{UT}_k^2$ then output "*unifiable*" and halt. □

Corollary 4.7: Goal terms g_1 and g_2 are R-unifiable under A-substitutions *iff* Procedure 4.2 halts and outputs "*unifiable*". □

Note that if g_1 and g_2 are not R-unifiable, then it may happen that the procedure does not halt.

4.3 Conditions for Termination of the Procedure

In this section, a sufficient condition for our procedure to be a decision procedure of the problem is investigated. We have already introduced Conditions 4.1 through 4.4, but they are not sufficient for termination of Procedure 4.1. Actually, the number of terms in A_k ($k \geq 1$) may increase monotonically as k increases without termination. Note that Procedure 4.1 halts if and only if there is a constant upper-bound on the number of terms in A_k, because once the set of terms are fixed (cannot be changed), then Procedure 4.1 only adds ε-transitions to δ_k, which saturates obviously. Also observe that if there is a constant upper-bound on the size of terms in A_k, then the number of terms in A_k is also upper-bounded. That is, a sufficient condition for the termination of Procedure 4.1 is that there is a constant upper-bound on the size of terms in A_k. In the following, we investigate such a sufficient condition.

First, the idea behind the sufficient condition is explained. Note that a newly introduced term at Step 3 of Procedure 4.1 is always an instance of a right-hand side of some rule. Hence the term can be written as $r\rho$ for a rule $l \to r \in R$ and a substitution $\rho = \{x_i \mapsto t_i^1 \mid t_i^1 \in A_k\}$. Note that t_i^1 is either a term in A_0, or again an instance of a subterm of a right-hand side of a rule in R. Inductively, there are layers in the term $r\rho$ (Figure 5), each layer is a non-ground subterm of a right-hand side of a rule in R, or a ground subterm of a right-hand side of a rule in R, or a term in A_0.

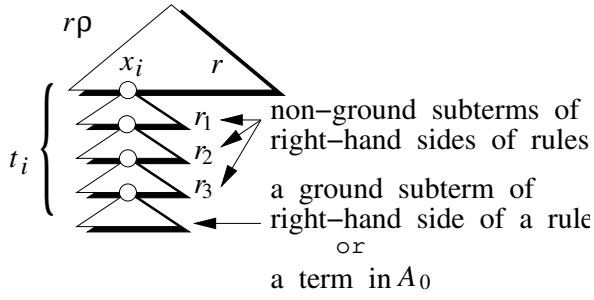


Figure 5: Layers in a newly introduced term

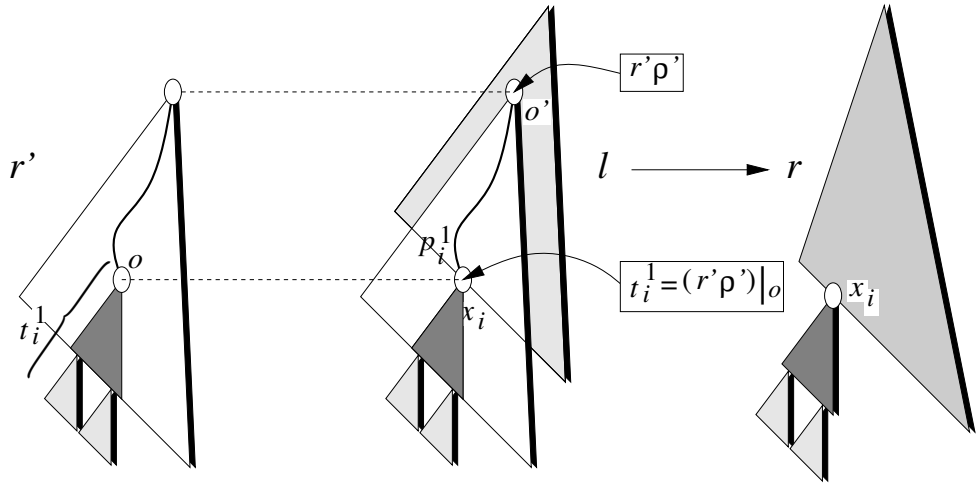


Figure 6: r' "*overlaps*" with l

We analyze the reason for an increase of these layers, and investigate a sufficient condition under which there is a constant upper-bound on the number of layers in new terms. If all of t_i^1 in the co-domain of ρ are terms in A_0, then the number of layers in $r\rho$ is two, a constant. Thereby, if the number of layers in $r\rho$ is more than two, then there is a term t_i^1 that is an instance of a subterm of a right-hand side of a rule in R, that is, there are a rule $l' \to r'$, an occurrence $o \in \mathcal{O}(r')$ and a substitution ρ' such that $t_i^1 = (r'\rho')|_o$. Note that $x_i \mapsto t_i^1 \in \rho$ means that, at the ID on the right-hand side of (1), one of heads of $M_k(t)$ scans the variable x_i at the occurrence p_i^1 with the state $[t_i^1] = [(r'\rho')|_o]$ (Figure 6). To get to this ID, the sequence (1) must contain an ID that has an so-pair $\langle [r'\rho'], o' \rangle$ where o' is an occurrence in $\mathcal{O}(l)$ such that $p_i^1 = o' \cdot o$. Furthermore, for every occurrence o'' such that $\lambda \preceq o'' \preceq o$, the function symbol of l at the occurrence $o' \cdot o''$ must be the same to that of r' at o' (Otherwise, $\langle [t_i^1], p_i^1 \rangle$ cannot appear in the right-hand side of (1)). Conversely, if there is no such "*overlapping*"

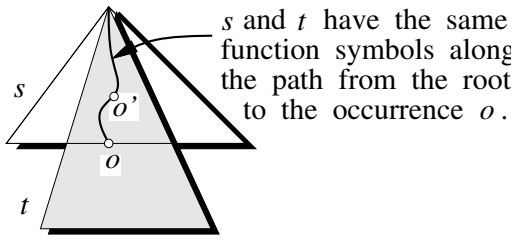


Figure 7: A term s path-matches to a term t via $o \in \mathcal{O}(s)$

of rules, then terms with three or more layers will not be included in A_k, and hence the size of terms in A_k is upper-bounded. But this condition seems too restricted for practical use.

In the following Condition 4.5, we introduce a condition under which there is no "loop" of such overlapping relations of rules. To define the condition formally, a concept of *path-matching* is defined as follows. We say a term s *path-matches* to a term t via a variable occurrence $o \in \mathcal{O}(s)$ if, for every $o'' \in \mathcal{O}(s)$ such that $\lambda \preceq o'' \prec o$, t has the occurrence o'' and the function symbol of t at o'' equals to that of s at o'' (Figure 7). We say s *strictly* path-matches to t via o if s path-matches to t via o, and $t|_o$ is neither a ground term nor a variable. The following is a formal description of the condition.

Condition 4.5: The following Procedure 4.3 halts and outputs "O.K." for R. □

Procedure 4.3:

In this procedure, an integer called a *rank* is associated to each rule.

Input: TRS R which has n rules.

Output: "O.K." or "N.G.".

Step 1. *Rank* of every rule in R is set to 1.

Step 2. For a rule $l_1 \to r_1$ with rank i_1, if there is a rule $l_2 \to r_2$ (which may be the same rule $l_1 \to r_1$) with rank $i_2 (\geq i_1)$, and either

(i) a subterm of l_1 strictly path-matches to r_2, or

(ii) l_1 strictly path-matches to a subterm of r_2,

holds (see Figure 8 (i), (ii)), then the rank of the rule $l_1 \to r_1$ is set to $i_2 + 1$. If the rank exceeds n, then output "N.G." and halt, since it means that there is a "loop" of path-matching relation, and terms with arbitrary many layers are possibly produced.

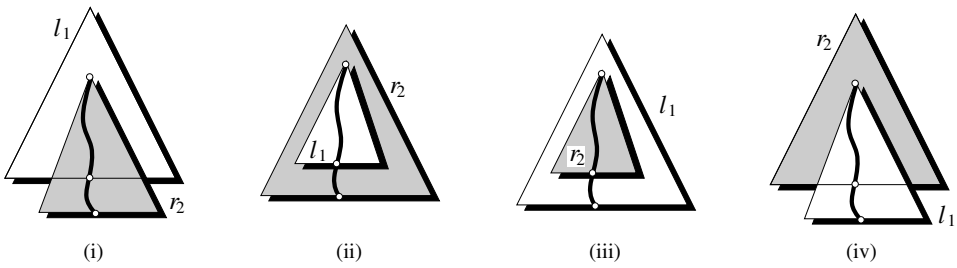


Figure 8: Rewrite rules and path-matching

Step 3. For a rule $l_1 \to r_1$ with rank i_1, if there is a rule $l_2 \to r_2$ with rank $i_2 (> i_1)$, and either

(iii) r_2 path-matches to a subterm of l_1, or

(iv) a subterm of r_2 path-matches to l_1,

holds (see Figure 8(iii),(iv)), then the rank of the rule $l_1 \to r_1$ is set to i_2.

Step 4. Repeat Steps 2 and 3 until the rank for any rule is not changed.

Step 5. Output "O.K." and halt. □

Procedure 4.3 halts clearly since the rank of a rule increases monotonically, and is upper-bounded by n. If the procedure outputs "O.K.", then each rule in R has a rank between 1 to n. Next, using ranks of rewrite rules, an integer also called a *rank* is defined for terms in A_k $(k \geq 0)$ as follows. First, define the rank of each term in A_0 to be 0. When a new term s is added into A_{k+1} at Step 3(a) of Procedure 4.1, the rank of s is defined according to the following rule. Let j be the rank of the rewrite rule $l \to r$ used in Step 3.

- If s (and its literal similar term) is not in A_{k+1}, then the rank of s is set to j.
- If s (or its literal similar term) has been in A_{k+1} as a rank $j' (> j)$ term, then the rank of s is reduced to j.
- Otherwise, (i.e. s has been already in A_{k+1} as a rank $j' (\leq j)$ term), then the rank of s is not be changed.

Remark that every element in A_k $(k \geq 0)$ has a rank between 0 to n, and the following lemma holds.

Lemma 4.8: Let $s, t \in A_k$ $(k \geq 0)$. If t is a subterm of s and the rank of t is j, then the rank of s is j or more. □

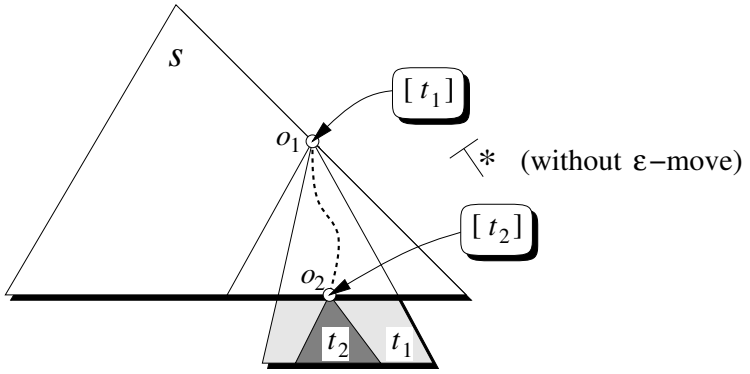


Figure 9: a relation between a state of tree automaton and path-matching.

Termination of the Procedure

To make discussion easier, some notions are defined. Let s be a term and let α be a sequence of moves from an initial ID of s to a partial accepting ID. An occurrence $o \in \mathcal{O}(s)$ is an *ε-move occurrence in α* if α contains an ε-move $(H \cup \{\langle q, o\rangle\}, s) \vdash (H \cup \{\langle q', o\rangle\}, s)$. For occurrences $o_1, o_2 \in \mathcal{O}(t)$ such that $o_1 \prec o_2$, the sequence α contains ε-move *along the path from o_1 to o_2* if there is an ε-move occurrence o such that $o_1 \preceq o \preceq o_2$. An occurrence o is the *innermost* ε-move occurrence in α along the path from o_1 to o_2 if $o_1 \preceq o \preceq o_2$, o is an ε-move occurrence and there is no other ε-move occurrence o' such that $o \prec o' \preceq o_2$.

Lemma 4.9: Let s be a term and let $o_1 \cdot o_2$ be a variable occurrence of s. If there are sets H, H' of so-pairs, terms $t_1, t_2 \in A_k$, and the following sequence of moves of $M_k(t)$ $(t \in A_k, k \geq 0)$

$$(H \cup \{\langle [\tilde{t}_1], o_1\rangle\}, s) \vdash^*_{M_k(t)} (H' \cup \{\langle [\tilde{t}_2], o_1 \cdot o_2\rangle\}, s)$$

which does not contain ε-move along the path from o_1 to $o_1 \cdot o_2$, then $s|_{o_1}$ path-matches to t_1 via o_2, and $t_2 = t_1|_{o_2}$ (see Figure 9).

Proof. Easily shown by induction on the length of o_2. □

By the definition of ranks of terms, for a rank $j(\geq 1)$ term t such that $t \in A_k - A_{k-1} (k \geq 1)$, there are a term $t_0 \in A_{k-1}$, a rewrite rule $l \to r$ with rank j, and a substitution ρ which are used in Step 3 of Procedure 4.1, and t can be written as $t = (r|_o)\rho$ for $o \in \mathcal{O}(r)$. That is, t is an instance of a (subterm of) right-hand side of some rule.

Lemma 4.10: If $r|_o$ has a variable occurrence $o' \in \mathcal{O}(r|_o)$, then $t|_{o'}$ is either a ground term with size $r_{\max}$ or less, or a term with rank $j-1$ or less.

Proof. Since $t = (r|_o)\rho$, it suffices to show that every term in co-domain of ρ is either a ground term with size $r_{\max}$ or less, or a term with rank $j-1$ or less. It is shown by induction on k. For $k=1$, co-domain of ρ consists of terms in A_0, which rank is 0 and the lemma holds. Assume that the lemma holds for $k-1$ or less. The inductive part is shown by contradiction. Assume that $\{x \mapsto s\} \in \rho$, the rank of s is greater than $j-1$, and s is either a ground term with size greater than $r_{\max}$, or a non-ground term. By Condition 4.4, there is an occurrence $o_l \in \mathcal{O}(l)$ such that the variable x occurs in l at o_l, and by Step 3 of Procedure 4.1, there is the following sequence α of moves of $M_{k-1}(t_0)$:

$$\alpha : (\{\langle[\tilde{t_0}], \lambda\rangle\}, l) \vdash^*_{M_{k-1}(t_0)} (H \cup \{\langle[\tilde{s}], o_l\rangle\}, l)$$

where the right-hand side is a minimal partial accepting ID of l.

- If α contains ε-move along the path from the root to o_l, then there is the innermost ε-move occurrence $o_1 \in \mathcal{O}(l)$, and o_l can be written as $o_l = o_1 \cdot o_2$ for some o_2. Furthermore, there are terms $t_1, t_2 \in A_{k-1}$ and α can be written as

$$(\{\langle[\tilde{t_0}], \lambda\rangle\}, l) \vdash^*_{M_{k-1}(t_0)} (H_0 \cup \{\langle[\tilde{t_1}], o_1\rangle\}, l) \tag{20}$$

$$\vdash^*_{M_{k-1}(t_0)} (H_0 \cup \{\langle[\tilde{t_2}], o_1\rangle\}, l) \tag{21}$$

$$\vdash^*_{M_{k-1}(t_0)} (H_1 \cup o_1 \cdot H_2 \cup \{\langle[\tilde{s}], o_1 \cdot o_2\rangle\}, l) \tag{22}$$

  Since o_1 is the innermost ε-move occurrence, the sequence of moves from (21) to (22) does not contain ε-move along the path from o_1 to $o_1 \cdot o_2$. By Lemma 4.9, l_{o_1} path-matches to t_2 and s can be written as $s = t_2|_{o_2}$. By Lemma 4.8, it follows that the rank of t_2 is greater than $j-1$ since the rank of s is assumed to be greater than $j-1$. Thereby, there is a rank j or more rewrite rule $l' \to r'$ such that t_2 is an instance of r'. As for variable occurrences of r', there are three cases.

  - r' has a variable at occurrence o_3 such that $o_3 \preceq o_2$: In this case, r' path-matches to $l|_{o_1}$ via o_3 (see Figure 10), and the rank of $l' \to r'$ is j (if it is greater than j, then the rank of $l \to r$ must be greater than j. See (iii) of Procedure 4.3). This implies that the rank of t_2 is j. Since $t_2 \in A_{k-1}$, by the inductive hypothesis, $t_2|_{o_3}$ is either a ground term with size $r_{\max}$ or less, or a term with rank $j-1$ or less. Remark that $s = t_2|_{o_2}$ is a subterm of $t_2|_{o_3}$, and hence s is either a ground term with size $r_{\max}$ or less, or a term with rank $j-1$ or less, a contradiction.

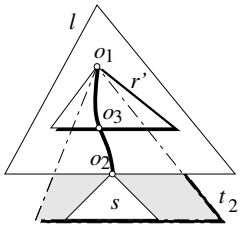


Figure 10: r' path matches to l_{o_1} and t_2.

- r' has a variable at occurrence o_3 such that $o_2 \prec o_3$: In this case, $l|_{o_1}$ strictly path-matches to r', and the rank of $l \to r$ must be greater than the rank of $l' \to r'$ (see (i) of Procedure 4.3). That is, the rank of $l \to r$ cannot be j, a contradiction.
- all variable occurrences of r' are disjoint to o_2: In this case, $r'|_{o_2} = t_2|_{o_2} = s$ is a ground term, and the size cannot be greater than $r_{\max}$, a contradiction.

Summarizing, it cannot occur that α contains ε-move along the path from the root to o_l.

- If α does not contain ε-moves along the path from the root to o_l, then by Lemma 4.9, l path matches to t_0 via o_l and s is a subterm of t_0 at occurrence o_l. Since the rank of s is greater than $j-1$, the rank of t_0 is also greater than $j-1$. Thereby, there is a rank j or more rewrite rule $l' \to r'$ and an occurrence $o_1 \in \mathcal{O}(r')$ such that s is an instance of $r'|_{o_1}$.
  As for variable occurrence of $r'|_{o_1}$, there are three cases, but all of them result a contradiction as in the previous case. Thereby, it can be concluded that s is either a ground term with size $r_{\max}$ or more, or a term with rank $j-1$ or less.

□

By using Lemma 4.10, the following lemma holds.

Lemma 4.11: The size of rank $j(>0)$ term cannot exceed $r_{\max}(1 + a_{\max} + \cdots + {a_{\max}}^{j-1}) + {t_{\max}}^{j}$. □

Since the rank of a term in A_k is at most n, the following corollary holds.

Corollary 4.12: Under Condition 4.5, the size of the term $r\rho$ at Step 3 of Procedure 4.1 is at most $r_{\max}(1 + a_{\max} + \cdots + {a_{\max}}^{n-1}) + {t_{\max}}^{n}$. □

This corollary implies that there is an integer k_1 such that $A_0 \subseteq \cdots \subseteq A_{k_1} = A_{k_1+1} = \cdots$. That is, if TRS and goal terms satisfy Conditions 4.1 through 4.4 and 4.5, then

Procedure 4.1 terminates with correct answer.

5 Comparison to Other Method

In this section, we compare our procedure to other ones.

In [4], the concept of the unification problem under constrained substitutions has been introduced as a formal verification method of a security of cryptographic protocols. In the verification method, rewriting rules in TRS describe mathematical properties of functions and operations used in the protocol. From the practical viewpoint, the class of TRS that can be dealt with the polynomial time procedure introduced in [4] was rather restricted. For example, a commutativity rule $f(x, y) \to f(y, x)$ is not allowed in the procedure. Our procedure is "upper-compatible" to the procedure in [4]. That is, our procedure can deal with properly larger class than the procedure in [4], including the commutativity rule above. This is a remarkable improvement for verifying of cryptographic protocols.

As we remarked in Section 3, our procedure can be used to solve the usual unification problem. For example, consider the following example:

Example 5.1: Let $F = \{c, f, g, h\}$ $(a(c) = 0, a(f) = a(g) = a(h) = 1)$, $R = \{f(h(x)) \to f(x), g(h(x)) \to g(x)\}$, $G = \{f(x), g(y)\}$, and $A = \{f(x), g(x), h(x), a\}$. Observe that $\mathcal{T}_A[X] = \mathcal{T}_F[X]$. □

The readers can easily verify that our procedure halts for this example and answers "*unifiable*". Basic narrowing[7] which is a widely studied strategy for the usual unification problem will not terminate for this example. On the other hand, our procedure is not available for TRS and goal terms that do not satisfy Conditions 4.1 through 4.4 and 4.5.

6 Conclusion

In this paper, a procedure that solves the unification problem under constrained substitutions using tree automata is studied. We also investigated on some conditions for our procedure (Table 1). Our method is an approach different from step-by-step rewriting methods such as narrowing, and there are a number of rewrite systems for which unification problem can be solved by our procedure but cannot be solved by a narrowing procedure.

Future work includes to relax these sufficient conditions so that our procedure can deal with a larger class of TRS. For example, the current construction of tree automata

Table 1: Conditions and halting property of Procedure 4.2

	goals are unifiable	goals are not unifiable
Conditions 4.1 through 4.4	halts with correct answer	halts with correct answer, or does not halt
Conditions 4.1 through 4.4 and 4.5	halts with correct answer	halts with correct answer

in Procedure 4.1 is relatively naive. It may be improved by using substantial works in formal language theory. We also remark that Condition 4.4 will be unnecessary if, at Step 3 of Procedure 4.1, one constructs a tree automaton that accepts $L(M_k(t_i^1)) \cap \cdots \cap L(M_k(t_i^{\gamma_i}))$ and uses the initial state of this automaton instead of $[t_i^1]$. It will be also an interesting work to define a class of TRS for which the unification problem under constrained substitutions is solvable by our procedure in deterministic polynomial time.

References

[1] Dershowitz N. and Jouannaud J.P.: "Rewrite Systems", Handbook of Theoretical Computer Science, Vol. B, North-Holland (1990).

[2] Dershowitz N. and Sivakumar G.: "Solving Goals in Equational Languages", Proc. First Workshop of CTRS, Lecture Notes in Computer Science **308**, pp.45–55 (1987).

[3] Fay M.: "First-order Unification in an Equational Theory", Proc. Fourth Workshop on Automated Deduction, Austin, Texas, pp.161–167 (Feb. 1979).

[4] Fujiwara T., Taniguchi K. and Kasami T.: "Decision Problem of the Security for Cryptographic Protocols", Trans. IEICE, **J69-D**, 6, pp.984–992 (June 1986, in Japanese); also in Perspectives in Computing, Academic Press, pp.263–286 (1987).

[5] Gécseq F. and Steinby M.: "Tree Automata", Akadémiai Kiadó, Budapest (1984).

[6] Heilbrunner S. and Hölldobler S.: "The Undecidability of the Unification and Matching Problem for Canonical Theories", Acta Informatica, **24**, pp.157–171 (1987).

[7] Hullot J.M.: "Cannonincal Forms and Unification", Proc. Fifth International Conference on Automated Deduction, Les Arcs, France, Lecture Notes in Computer Science **87**, pp.318–334 (1980).

[8] Martelli A., Moiso C. and Rossi G.F.: "An algorithm for Unification in Equational Theories", Proc. Third IEEE Symposium on Logic Programming, Salt Lake City, Utah, pp.180–186 (Sep. 1986).

[9] Robinson J.A.: "Computational Logic: The Unification Computation", Machine Intelligence, **6**, Edinburgh University Press, pp.63–72 (1971).

[10] Siekmann J.H.: "Unification Theory", J. Symbolic Computation, **7**, pp.207–274 (1989).