IPSJ Transactions on Programming Vol. 2 No. 2 166–175 (Mar. 2009)

Regular Paper

Recognizability of Redexes for Higher-Order Rewrite Systems

Hideto Kasuya,[†1] Masahiko Sakai[†2] and Kiyoshi Agusa[†2]

It is known that the set of all redexes for a left-linear term rewriting system is recognizable by a tree automaton, which means that we can construct a tree automaton that accepts redexes. The present paper extends this result to Nipkow's higher-order rewrite systems, in which every left-hand side is a linear fully-extended pattern. A naive extension of the first-order method causes the automata to have infinitely many states in order to distinguish bound variables in λ-terms, even if they are closed. To avoid this problem, it is natural to adopt de Bruijn notation, in which bound variables are represented as natural numbers (possibly finite symbols, such as 0, s(0), and s(s(0))). We propose a variant of de Bruijn notation in which only bound variables are represented as natural numbers because it is not necessary to represent free variables as natural numbers.

1. Introduction

Tree automata[3] are valuable in proving various properties for term rewriting systems (TRSs) and also in constructing their automated tools. The automata construction that recognizes all redexes of a given TRS is a well-known technique, which is based on the automata construction that recognizes all closed instances of a given term. These techniques are used to, for example, characterize the set of normal-forms and, when combined with the ground tree transducer technique, solve reachability problems.

TRSs are extended to higher-order rewrite systems (HRSs)[9], introducing higher-order variables. The matching problem of λ-terms is decidable by using the notion of $\Box$-tree automata[4]. Since a $\Box$-tree automaton is constructed from a term p containing one free variable x and a term t and recognizes the set of all terms s such that $t = p[x \rightarrow s]$, it is insufficient for characterizing all normal forms and applying most automata techniques. Moreover, this method is applicable for patterns of the fourth or lower order.

The present paper clarifies the construction of automata that recognize the set of all redexes and/or the set of all terms having redexes for a given HRS. Since tree automaton is used in the characterization, we have benefits that we can use fruitful results of tree automaton technique to show various properties on HRSs. In real, we use the notion of GTT (ground tree transducer) to show that the set of terms reachable from a given regular set of terms is recognizable for a given HRS having no common free variable in both sides of rewrite rules. Although this class of HRSs is not practical itself, this reachability result is valuable because it is applicable for ordinary HRSs to determine an approximated set of reachable terms, which includes all reachable terms.

In extending the automata construction that recognizes all closed instances of a first-order term to higher-order term, the α-equivalence causes a problem. Closed terms may contain bound variables, and so we cannot bound the number of possible variables, which causes the automata to have infinitely many symbols and states to distinguish bound variables. It is natural to adopt de Bruijn notation[2),7] in order to avoid the α-equivalence problem.

We introduce a variation of de Bruijn notation. The original de Bruijn notation is modified as follows:

- We adopt an algebraic representation in which "application" is represented by a special symbol "@" with arity two.
- Natural numbers are represented as 0, $S(0)$, $S(S(0))$, ... by using special symbols S with arity one and 0 with arity zero.
- Free variables are not coded by natural numbers.

For example, a λ-term $\lambda x.xy(\lambda z.zx)$ is represented as $\lambda.12(\lambda.12)$ in de Bruijn notation and as $\lambda(@(@(S(0), y), \lambda(@(S(0), S(S(0))))))$ in the variant introduced in the present paper. The first two modifications are natural because the language definable by a tree automaton is a set of algebraic terms of a finite signature. The third modification came from technical reason. More precisely, although we need to prove that a redex $p\sigma$ in de Bruijn notation is accepted by the produced

†1 Faculty of Information Science and Technology, Aichi Prefectural University
†2 Graduate School of Information Science, Nagoya University

tree automata for a left-hand side p and substitution σ in the notation, the domain of substitution in the notation is restricted to singleton sets because it is designed for dealing β-reduction. Hence some extension would be necessary for this purpose. Moreover, the substitution is complicated and difficult since even the substitution lemma in the notation proved by Ohtsuka[11] is complicated. Thus we tried a different approach by using $\phi(P\theta)$ for $p\sigma$ in proofs where ϕ is the transformation of usual term to de Bruijn notation, and P and θ are usual representations of left-hand side and substitution, respectively. Since de Bruijn notation is not unique with respect to free variable assignment, we omit the free variables in order to obtain uniqueness of the de Bruijn notation.

The remainder of the present paper is organized as follows. We review the basic notions of HRSs and tree automata in Section 2. We introduce a variant of de Bruijn notation and describe some properties in Section 3. In Section 4, we show a construction of tree automata that recognize the set of all closed instances of a fully-extended pattern and discuss the regularity of redexes of fully-extended HRSs. Finally, in Section 5, we state an application showing that the reduction of some class of HRSs preserves regularity.

2. Preliminary

Let Σ be a signature with arity function *arity*: $F \to \mathbb{N}$. Let $\mathcal{X}$ be a set of variables. We denote the set of all terms constructed from symbols in Σ and $\mathcal{X}$ by $\mathcal{T}(\Sigma \cup \mathcal{X})$ in the usual manner. We sometimes refer to these types of terms as "algebraic" terms.

We use $Occ(t)$ for the set of all occurrences of t. We denote the *subterm* of a term s at an occurrence $p \in Occ(s)$ by $s|_p$. In addition, $s[t]_p$ denotes the term obtained from s by replacing subterm $s|_p$ by the term t. The *top symbol* of $t = f(t_1, \ldots, t_n)$ is defined as $top(t) = f$.

2.1 Higher-Order Rewrite System

Let S be a set of basic types. The set τ_s of types is generated from S by the function space constructor $\to$ as follows:

$\tau_s \supseteq S$

$\tau_s \supseteq \{\alpha \to \alpha' \mid \alpha, \alpha' \in \tau_s\}$

Let $\mathcal{X}_\alpha$ be a set of variables of type α, and let $\mathcal{F}_\alpha$ be a set of function symbols of type α. The set of all variables is denoted by $\mathcal{X} = \bigcup_{\alpha\in\tau_s} \mathcal{X}_\alpha$, and the set of all function symbols is denoted by $\mathcal{F} = \bigcup_{\alpha\in\tau_s} \mathcal{F}_\alpha$. We represent *simply typed λ-terms* by algebraic terms $\mathcal{T}(\mathcal{F} \cup \mathcal{X} \cup \{@, \lambda x \mid x \in \mathcal{X}\})$, where $arity(a) = 0$ for $a \in \mathcal{F} \cup \mathcal{X}$, $arity(\lambda x) = 1$ and $arity(@) = 2$. For example, $\lambda xy.xy$ in ordinary representation is written by $\lambda x(\lambda y(@(x, y)))$. The definition is given by the following inference rules:

$$\frac{x \in \mathcal{X}_\alpha}{x : \alpha} \quad \frac{f \in \mathcal{F}_\alpha}{f : \alpha} \quad \frac{s : \alpha \to \alpha' \quad t : \alpha}{@(s,t) : \alpha'} \quad \frac{x : \alpha \quad s : \alpha'}{\lambda x(s) : \alpha \to \alpha'}$$

If $M : \alpha$ is inferred from the rules, then M is a simply typed λ-term of type α. A simply typed λ-term is called a *higher-order term*, or simply a *term*, if no confusion would arise. We use the concepts of bound variables and free variables. The sets of bound and free variables occurring in a term M are denoted by $BV(M)$ and $FV(M)$, respectively. The set $FV(M) \cup BV(M)$ is denoted by $Var(M)$. A higher-order term without free variables is said to be *closed*. If a term N is generated by the bound variables renaming in a term M, then N and M are α-equivalent and are denoted by $N \equiv M$.

We use F, G, X, Y, and Z for free variables and x, y, and z for bound variables, unless known to be free or bound from other conditions. We also use c, d, f, g, and h for function symbols and a for a variable or a function symbol. We also use P, M, and N for λ-terms

For a term $@(\lambda x(M), N)$, *β-reduction* is the operation that returns the term obtained from M by replacing all xs by N, where $@(\lambda x(M), N)$ is called a *β-redex*. Let M be a term of type $\alpha \to \alpha'$, and let $x \notin Var(M)$ be a variable of type α. Then, *η-expansion* is the operation that replaces M in a term by $\lambda x(@(M, x))$ if it produces no new β-redex. A term is said to be *η-long*, if it is in normal form with respect to η-expansion. In addition, a term is said to be *normalized* if it is in β and η-long normal form. A normalized term of M is denoted by $M{\downarrow}$. Each higher-order term has a unique normalized term[1].

A mapping σ from variables to higher-order terms is called a *substitution* if $\sigma(X)$ is of the same type as X and the domain $Dom(\sigma) = \{X \mid X \not\equiv \sigma(X)\}$ is finite. If $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and $\sigma(X_i) \equiv M_i$, we also write the mapping as $\sigma = \{X_1 \mapsto M_1, \ldots, X_n \mapsto M_n\}$. Let W be a set of variables, and let σ be a substitution. We write $\sigma|_W$ for the substitution obtained by restricting the domain of σ to $Dom(\sigma) \cap W$ and write $\sigma|_{\overline{W}}$ for that obtained by restricting its domain to $Dom(\sigma) - W$. For a substitution σ, the set of free variables in the range of σ is defined by $VRan(\sigma) = \bigcup_{X \in Dom(\sigma)} FV(\sigma(X))$.

A substitution σ is extended to a mapping $\tilde{\sigma}$ from higher-order terms to higher-order terms as follows:

$$\tilde{\sigma}(M) = @(@(\lambda X_1(\lambda X_2(M)), M_1), M_2)$$

where, for simplicity, $Dom(\sigma) = \{X_1, X_2\}$. Generally, when we extend a substitution σ to $\tilde{\sigma}$, the condition is required whereby the domain and range of σ do not contain any bound variables in the term to which the substitution $\tilde{\sigma}$ is applied. Here, note that when we adopt the above definition of $\tilde{\sigma}$ obtained by using β-reduction, we need not mention the condition explicitly. The above condition can always be satisfied by appropriately renaming bound variables. In the following, we simply write σ for $\tilde{\sigma}$ and $M\sigma$ for $\sigma(M)$. A substitution σ is said to be normalized if $\sigma(X)$ is normalized for any $X \in Dom(\sigma)$.

For $N = @(\cdots @(@(M, N_1), N_2), \ldots, N_n)$ such that $top(M) \neq @$, we define $etop(N)$ by $top(M)$. Let N be a normalized term such that $etop(N) \in \mathcal{F}$, and let $M{\downarrow_\eta}$ denote the η-normal form of M 8). A linear normalized term N is said to be a *pattern*, if every basic typed subterm M of N, such that $etop(M)$ is a free variable, has the form $@(\cdots @(@(F, M_1), M_2), \ldots, M_n)$ and if $M_1{\downarrow_\eta}, \ldots, M_n{\downarrow_\eta}$ are different bound variables. Moreover, this linear normalized term is said to be *fully-extended* if $M_1{\downarrow_\eta} \cdots M_n{\downarrow_\eta}$ is a sequence of all bound variables. For example, $@(f, \lambda x(@(F, \lambda y(@(x, y)))))$ ⋆1 is a fully-extended pattern.

Let α be a basic type, let $l : \alpha$ be a pattern, and let $r : \alpha$ be a normalized term. Then, $l \rhd r : \alpha$ is called a higher-order rewrite rule of type α ⋆2. A *higher-order rewrite system* (HRS) is a set of higher-order rewrite rules. Let R be an HRS, let $l \rhd r$ be a rewrite rule of R, and let σ be a substitution. Then, $l\sigma{\downarrow}$ is said to be a *redex*. If M and N are normalized terms such that $M \equiv M[l\sigma\downarrow]_p$ for an occurrence p and $N \equiv M[r\sigma\downarrow]_p$, then M can be reduced to N. This is denoted by $M \rightarrow_R N$, or simply $M \rightarrow N$. Since all rewrite rules are of basic type, N is normalized if M is normalized 10).

⋆1 $f(\lambda x.F(\lambda y.xy))$ in Nipkow's algebraic notation.
⋆2 In the present paper, we do not assume $FV(l) \supseteq FV(r)$.

2.2 Tree Automaton

Definition 1 (tree automaton) A *tree automaton* is a tuple $A = \langle Q, \Sigma, Q_f, \delta \rangle$, where Q is a finite set of states, Σ is a signature such that $Q \cap \Sigma = \emptyset$, $Q_f (\subseteq Q)$ is a set of final states, and δ is a set of transition rules of the following forms:

$$f(q_1, \ldots, q_n) \rightarrow q$$

where $q_1, \ldots, q_n, q \in Q$, $f \in \Sigma$, $arity(f) = n$. □

In particular, a tree automaton is *deterministic* if there are no two rules having the same left-hand side. A *transition relation* $\rightarrow_A$ of a tree automaton A on $\mathcal{T}(Q \cup \Sigma)$ is defined as follows:

$$\begin{cases} f(q_1, \ldots, q_n) \rightarrow_A q & \text{if } f(q_1, \ldots, q_n) \rightarrow q \in \delta \\ f(\ldots, s, \ldots) \rightarrow_A f(\ldots, t, \ldots) & \text{if } s \rightarrow_A t \end{cases}$$

Reflexive transitive closure of $\rightarrow_A$ is denoted by $\rightarrow_A^*$.

A tree automaton A *accepts* a term $t \in \mathcal{T}(\Sigma)$ if $\overset{def}{\Leftrightarrow}$ $t \rightarrow_A^* q \in Q_f$. The set of all acceptable terms are called the *language* of A, which is denoted as $\mathcal{L}(A)$. A set L of terms are *regular* if there exists a tree automata A such that $L = \mathcal{L}(A)$.

3. Variant of de Bruijn Notation

In this section, we introduce *db-terms* as a variant of de Bruijn notation.

For a function symbol $\mathcal{F}$ of higher-order terms, de Bruijn notation is a term with a signature $\Sigma = \mathcal{F} \cup \mathcal{X} \cup \{\lambda, @, S, 0\}$, where $arity(a) = 0$ for $a \in \mathcal{F} \cup \mathcal{X} \cup \{0\}$, $arity(\lambda) = 1$, $arity(S) = 1$, and $arity(@) = 2$. Note that variables in $\mathcal{X}$ are used as free variables. In the following, we use p, s, and t for db-terms.

Definition 2 (db-term) Let M be a λ-term. The db-term that represents M is $\phi(M)$, where the function ϕ and $[\![\ ]\!]$, which represents the replacement of bound variables, are defined as follows:

$$\begin{array}{ll} \phi(a) = a & \text{if } a \in \mathcal{F} \cup \mathcal{X} \\ \phi(@(M, N)) = @(\phi(M), \phi(N)) & \\ \phi(\lambda x(M)) = \lambda(\phi(M)[\![x \mapsto 1]\!]) & \end{array}$$

$$a[\![y \mapsto n]\!] = \begin{cases} S^n(0) & \text{if } a = y \in \mathcal{X} \\ a & \text{otherwise} \end{cases}$$

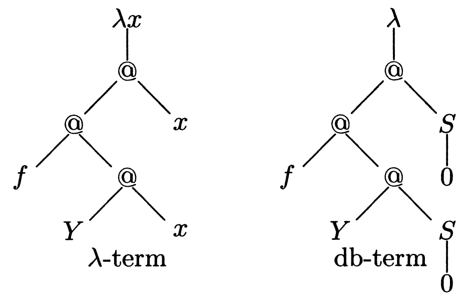

Fig. 1 Correspondence of λ-term and db-term.

$@(s,t)[\![y \mapsto n]\!] = @(s[\![y \mapsto n]\!], t[\![y \mapsto n]\!])$
$\lambda(s)[\![y \mapsto n]\!] \equiv \lambda(s[\![y \mapsto n+1]\!])$ □

The following are examples of translation from algebraic terms to db-terms.

Example 3

$$\begin{aligned}
&\phi(\lambda x(\lambda y(@(x,y))))\\
=\ &\lambda(\phi(\lambda y(@(x,y)))[\![x \mapsto 1]\!])\\
=\ &\lambda(\lambda(@(x,y)[\![y \mapsto 1]\!])[\![x \mapsto 1]\!])\\
=\ &\lambda(\lambda(@(S^2(0),S(0))))\\
&\phi(\lambda x(@(@(f,@(Y,x)),x)))\\
=\ &\lambda(@(@(f,@(Y,x)),x))[\![x \mapsto 1]\!])\\
=\ &\lambda(@(@(f,@(Y,S(0))),S(0)))
\end{aligned}$$
□

Note that $Occ(M) \subseteq Occ(\phi(M))$ holds. Moreover, the symbol at each occurrence in a λ-term corresponds to that in the db-term. For the second example, the correspondence between the terms is shown in **Fig. 1**.

It is easy to replace free variables by natural numbers, as in the original de Bruijn notation. Let $X_1, \ldots, X_k$ be free variables of a λ-term M. Then, $\phi(M)[\![X_1 \mapsto 1]\!] \cdots [\![X_k \mapsto k]\!]$ is one such term. In this sense, the original de Bruijn notation of terms having free variables is not unique.

The original de Bruijn notation is closed under subterms. A similar property also holds for the variant notation. For example, consider a subterm $t = s_{11} = @(S^2(0),S(0))$ of a db-term $s = \lambda(\lambda(@(S^2(0),S(0))))$. We have a λ-term $N = @(x,y)$ that satisfies $\phi(N)[\![y \mapsto 1]\!][\![x \mapsto 2]\!] = t$. The following lemma and proposition state this type of closedness under subterms.

Before showing the lemma, we introduce some notation. For a term M and $p \in Occ(M)$, we use $BV_p(M)$ to denote the sequence of λ-binders in the path from p to ε. For example, $BV_{11}(\lambda x(\lambda y(@(x,y)))) = y\ x$. We sometimes write $M_{x_1 \cdots x_n}[N]_p$ by presenting this information $BV_p(M) = x_1 \cdots x_n$ explicitly for replacement $M[N]_p$.

Lemma 4 Let M be an algebraic term and an occurrence $u \in Occ(M)$ such that no free variable appears at each $v(\leq u)$. For any algebraic λ-term T,

$\phi(M_{x_1 \cdots x_k}[T]_u) = \phi(M)[\phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!]]_u$. □

The proof of this lemma is found in Appendix A.1

Proposition 5 Let t be a subterm of a db-term. Then, there exists an algebraic term M and variables $x_1, \ldots, x_k$ such that

$\phi(M)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!] = t$.

Proof. Let t' be a db-term, and let T' be an algebraic term such that $t' = \phi(T') = t'[t]_u$. By renaming bound variables, we can represent T' as $T'_{x_1 \cdots x_k}[T'|_u]_u$. We take $T = T'|_u$. By Lemma 4, we have $\phi(T'_{x_1 \cdots x_k}[T]_u) = \phi(T')[\phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!]]_u$. Thus, $t = \phi(T')|_u = \phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!]$. □

4. Tree Automata Construction

We show a construction of a tree automaton that accepts instances of a pattern. First, we present an example. Consider a pattern $P \equiv \lambda x(@(@(f,@(Y,x)),x))$. The automaton we construct must accept db-term $t = \lambda(@(@(f,s),S(0)))$, where s is arbitrary. The set $Occ(P) \cap Occ(t)$ of occurrences regarding s as constant is called the *skeleton* of P, which is given as follows.

Definition 6 (Occurrences of skeleton) Let P be a pattern. $Occ^s(P)(\subseteq Occ(P))$, which denotes the set of all occurrences of the skeleton of P, is defined as follows.

$$Occ^s(M) = \begin{cases} \{\varepsilon\} \cup \{1u \mid u \in Occ^s(N)\} & \text{if } M = \lambda x(N) \\ \{\varepsilon\} & \text{if } etop(M) \in FV(P) \text{ or } M \in \mathcal{F} \cup \mathcal{X} \\ \{\varepsilon\} \cup \{iu \mid u \in Occ^s(M_i)\} & \text{otherwise, where } M = @(M_1, M_2) \end{cases}$$

Next, we present an automata construction.

Definition 7 (Tree automata construction) Let P be a fully-extended pattern, and let p be a db-term such that $p = \phi(P)$. Let k be the maximum of $S^k(0)$ that appears in p. We construct a tree automaton $A_P = \langle Q, \Sigma, Q_f, \delta\rangle$ that recognizes the set of all db-terms that represent instances $P\theta\!\downarrow$ of pattern P. $Q = \{q_\perp\} \cup \{q_u \mid u \in Occ^s(P)\} \cup \{q'_m \mid 0 \le m \le k\} \cup \{q_f \mid f \in \mathcal{F}\}$, $\Sigma = \mathcal{F} \cup \{\lambda, @, S, 0\}$, and $Q_f = \{q_\varepsilon\}$. The set δ of transition rules consists of the following rules.

(1) For each $u \in Occ^s(P)$,
  (1-1) $q_\perp \to q_u$ if $etop(P|_u)$ is a free variable,
  (1-2) otherwise
    (1-2-1) $a \to q_u$ if $P|_u = a \in \mathcal{F}$,
    (1-2-2) $q'_n \to q_u$ if $P|_u$ is a bound variable, where n is determined by $p|_u = S^n(0)$,
    (1-2-3) $\lambda(q_{u1}) \to q_u$ if $P|_u = \lambda(T)$,
    (1-2-4) $@(q_{u1}, q_{u2}) \to q_u$ if $P|_u = @(M, N)$,
(2) $0 \to q'_0$ and $S(q'_{n-1}) \to q'_n$ for each $n \in \{1, \ldots, k\}$,
(3) $q_a \to q_\perp$ for each $a \in \mathcal{F}$, $S(q'_0) \to q_\perp$, $S(q_\perp) \to q_\perp$, $\lambda(q_\perp) \to q_\perp$, and $@(q_\perp, q_\perp) \to q_\perp$.

Example 8 (tree automaton A_P) Let $P \equiv \lambda x(@(@(f, @(Y, x)), x))$. Then, $p = \lambda(@(@(f,\ @(Y, S(0))), S(0)))$ and $k = 1$. The tree automaton that accepts db-terms of instances of P is $A_P = \langle Q, \Sigma, Q_f, \delta\rangle$, where $Q_f = \{q_\varepsilon\}$, and δ is the set of transition rules shown in **Fig. 2**. By this tree automaton, db-term $\lambda(@(@(f, S(0)), S(0)))$ of $(\lambda x.f(Y(x), x))[y \mapsto \lambda y.y]\!\downarrow = \lambda x.f(x, x)$ is accepted by A_P, as shown in **Fig. 3**. □

Theorem 9 Let P be a fully-extended pattern, and let A be the automaton constructed from P. Then,

$$\mathcal{L}(A) = \{\phi(M) \mid M \in I_P\},$$

where I_P is the set of all closed instances of P. □

The proof of this theorem is presented in Appendix A.2.

Note that it is impossible to remove the "fully-extended" condition from the Theorem 9. Consider a non-fully-extended pattern $\lambda x(X)$. The following db-terms are instances of the pattern.

$$\phi(\lambda x(\lambda y_1(y_1))) = \lambda(\lambda(S(0))),$$

Rules by (1)		
$q_\perp$	$\to$	q_{112}
f	$\to$	q_{111}
q'_1	$\to$	q_{12}
$\lambda(q_1)$	$\to$	q_ε
$@(q_{11}, q_{12})$	$\to$	q_1
$@(q_{111}, q_{112})$	$\to$	q_{11}
Rules by (2)		
0	$\to$	q'_0
$S(q'_0)$	$\to$	q'_1
Rules by (3)		
f	$\to$	$q_\perp$
$S(q'_0)$	$\to$	$q_\perp$
$S(q_\perp)$	$\to$	$q_\perp$
$\lambda(q_\perp)$	$\to$	$q_\perp$
$@(q_\perp, q_\perp)$	$\to$	$q_\perp$

Fig. 2 Transition rules of A_P in Example 8.

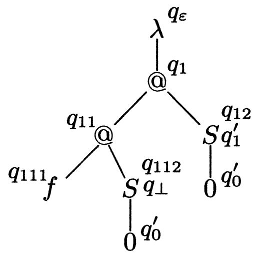

Fig. 3 Run of db-term $\lambda(@(@(f, S(0)), S(0)))$ by A_P.

$$\phi(\lambda x(\lambda y_1(\lambda y_2(y_1)))) = \lambda(\lambda(\lambda(S^2(0)))),$$
$$\vdots$$

However, the following db-terms are not instances of the pattern.

$$\phi(\lambda x(\lambda y_1(x))) = \lambda(\lambda(S^2(0))),$$
$$\phi(\lambda x(\lambda y_1(\lambda y_2(x)))) = \lambda(\lambda(\lambda(S^3(0)))),$$
$$\vdots$$

are not instances of the pattern. These examples show that we must distinguish

$S^n(0)$ and $S^{n+1}(0)$ for arbitrary n, which is impossible by an automaton, since the states are finite. Thus, there exists no automaton that recognizes instances of the pattern. We omit the lengthy and tiresome proof using pumping lemma [3], which is not difficult to prove.

Theorem 10 Let algebraic λ-term P be a fully-extended pattern. There is a tree automaton that accepts db-terms that represent closed terms containing instances of P.

Proof. Let A be the tree automaton obtained from A_P by adding the following rules.

$$\lambda(q_\varepsilon) \to q_\varepsilon$$
$$@(q_\varepsilon, q_\perp) \to q_\varepsilon$$
$$@(q_\perp, q_\varepsilon) \to q_\varepsilon$$

Then, A accepts db-terms of $C[P\theta\downarrow]$. Based on the construction of A and Theorem 9, this is trivial. □

Theorem 11 Let R be an HRS. It is possible to make a tree automaton that accepts db-term representations of reducible terms by R.

Proof. This is shown from Theorem 10 and the closure property under union of tree automata. □

5. Application to Reachability Problems

The Ground Tree Transducer (GTT) [6] is a pair of automata that represents a relation between trees. The advantage of GTT lies in its closure property for transitivity. GTT is defined as follows:

Definition 12 (GTT) GTT is a pair of tree automata working on the same alphabet. Their sets of states may have the same symbols. Let A_1 and A_2 be tree automata on Σ. A pair $(t, t') \in L \subseteq \mathcal{T}(\Sigma, \emptyset) \times \mathcal{T}(\Sigma, \emptyset)$ is recognized by a GTT $GTT(A_1, A_2)$, where L is the smallest set that satisfies the following conditions:

$$\begin{cases} \{(t,s) \mid t \to^*_{A_1} q \leftarrow^*_{A_2} s\} \subseteq L \\ (f(t_1,\ldots,t_n), f(s_1,\ldots,s_n)) \in L \ \text{ if } (t_i,s_i) \in L \end{cases}$$

□

Theorem 13 (Reference [5]) If a relation $R \subseteq \mathcal{T}(\Sigma^{\mathcal{N}})^2$ is recognized by a GTT, its reverse closure R^{-1} and transitive closure R^* are also recognized by a GTT.

For a regular set $L \subseteq \mathcal{T}(\Sigma^{\mathcal{N}})$, $R[L] = \{s \mid sRt \ \exists t \in L\}$ is also regular. □

Next, we show that reduction by a class of HRSs preserves regularity.

Theorem 14 Let R be an HRS such that every rule $l \triangleright r$ satisfies the following conditions:

- $FV(l) \cap FV(r) = \emptyset$ and
- r is fully-extended pattern.

If $L \subseteq \mathcal{T}(\Sigma^{\mathcal{N}})$ is regular set, then

$$\to^*_R [L] = \{s \mid s \to^*_R t, \ \exists t \in L\}$$

is also regular.

Proof. Let $R = \{l_1 \triangleright r_1, \ldots, l_n \triangleright r_n\}$. We have tree automata A_i (B_i), each of which accepts instances of l_i (r_i) with the only final state q_i, such that states other than final states are disjoint. This is made possible by Theorem 9. Next, we construct a GTT G on $\mathcal{T}(\Sigma^{\mathcal{N}}, \emptyset)$ as $GTT(\bigcup_i A_i, \bigcup_i B_i)$. Then, G recognizes a relation L such that $\to_R \subseteq L \subseteq \to^*_R$. Since there exists a GTT G' that recognizes L^* by Theorem 13, the GTT G' recognizes $\to^*_R$.

Therefore, the theorem follows from the latter part of Theorem 13. □

Acknowledgments We thank anonymous referees for giving useful comments. This study was supported in part by MEXT. KAKENHI #18500011, #20300010, and #20500008.

References

1) Andrews, P.B.: Resolution in Type Theory, *Journal of Symbolic Logic*, Vol.36, No.3, pp.414–432 (1971).
2) Bonelli, E., Kesner, D., and Rios, A.: A de Bruijn Notation for Higher-Order Rewriting, *Proc. RTA 2000*, *LNCS*, Vol.1833, pp.62–79 (2000).
3) Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M.: *Tree Automata Techniques and Applications*, http://www.grappa.univ-lille3.fr/tata (2007).
4) Comon, H. and Jurski, Y.: Higher-Order Matching and Tree Automata, *Proc. CSL '97*, *LNCS*, Vol.1414, pp.157–176 (1997).
5) Conquidé, J.-L. and Gilleron, R.: Proofs and Reachability Problem for Ground Rewrite Systems, *Proc. 6th International Meeting of Young Computer Scientists*, *LNCS*, Vol.464, pp.120–129 (1990).
6) Dauchet, M. and Tison, S.: Decidability of Confluence for Ground Term Rewriting Systems, *Proc. 1st Fundamentals of Computation Theory*, *LNCS*, Vol.199, pp.80–89 (1985).

7) de Bruijn, N.G.: Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, *Indagations Mathematicae*, Vol.34, pp.381–392 (1972).
8) Miller, D.: A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification, *Journal of Logic and Computation*, Vol.1, No.4, pp.497–536 (1991).
9) Nipkow, T.: Higher-Order Critical Pairs, *Proc. 6th IEEE Symposium, Logic in Computer Science*, pp.342–349, IEEE Press (1991).
10) Nipkow, T.: Orthogonal Higher-Order Rewrite Systems are Confluent, *Proc. Typed Lambda Calculi and Applications*, *LNCS*, Vol.664, pp.306–317, Springer-Verlag (1993).
11) Ohtsuka, H.: A Proof of the Substitution Lemma in de Bruijn's Notation, *Information Processing Letters*, Vol.46, Issue 2, pp.63–66 (1993).

Appendix

A.1 Proof of Lemma 4

Proof of Lemma 4. We prove the following generalized claim by induction on the length $|u|$ of u.

$$\phi(M_{x_1\cdots x_k}[T]_u)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$$
$$= \phi(M)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$$
$$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_u$$

In the case of $u = \varepsilon$, k must be 0. The claim trivially holds from $\phi(M[T]_\varepsilon) = \phi(T)$.

In the case of $u = iu'$ we have two subcases, because $M = a \in \mathcal{F} \cup \mathcal{X}$ implies $u = \varepsilon$.

(1) Consider the subcase of $M = \lambda x_k.M'$ and $i = 1$.

(left-hand side of the claim)

$=\phi((\lambda x_k(M'))_{x_1\cdots x_k}[T]_{1u'})[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$=\phi(\lambda x_k(M'_{x_1\cdots x_{k-1}}[T]_{u'}))[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$=\lambda(\phi(M'_{x_1\cdots x_{k-1}}[T]_{u'})[\![x_k \mapsto 1]\!])[\![y_1 \mapsto 1]\!]\cdots[\![y_k \mapsto l]\!]$

$=\lambda(\phi(M'_{x_1\cdots x_{k-1}}[T]_{u'})[\![x_k \mapsto 1]\!][\![y_1 \mapsto 2]\!]\cdots[\![y_l \mapsto 1+l]\!])$

$=\lambda(\phi(M')[\![x_k \mapsto 1]\!][\![y_1 \mapsto 2]\!]\cdots[\![y_l \mapsto 1+l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_{u'})$

where the last step is realized by induction.

(right-hand side of the claim)

$=\phi(\lambda x_k.M')[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_u$

$=\lambda(\phi(M')[\![x_k \mapsto 1]\!])[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_u$

$=\lambda(\phi(M')[\![x_k \mapsto 1]\!][\![y_1 \mapsto 2]\!]\cdots[\![y_l \mapsto 1+l]\!])$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_u$

$=\lambda(\phi(M')[\![x_k \mapsto 1]\!][\![y_1 \mapsto 2]\!]\cdots[\![y_l \mapsto 1+l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_{u'})$

(2) Consider the case of $M = @(M_1, M_2)$ and $i \in \{1, 2\}$. We assume $i = 2$ without loss of generality.

(left-hand side of the claim)

$=\phi(@(M_1, M_2)_{x_1\cdots x_k}[T]_{2u'})[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$=\phi(@(M_1, (M_2)_{x_1\cdots x_k}[T]_{u'}))[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$=@(\phi(M_1), \phi((M_2)_{x_1\cdots x_k}[T]_{u'}))[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$=@(\phi(M_1)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!], \phi((M_2)_{x_1\cdots x_k}[T]_{u'})[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!])$

$=@(\phi(M_1)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!], \phi(M_2)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_{u'})$

where the last step is realized by induction.

(right-hand side of the claim)

$=\phi(@(M_1, M_2))[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]\ ]_u$

$=@(\phi(M_1), \phi(M_2))[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]\ ]_u$

$=@(\phi(M_1)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!], \phi(M_2)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!])$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_u$

$=@(\phi(M_1)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!], \phi(M_2)[\![y_1 \mapsto 1]\!]\cdots[\![y_l \mapsto l]\!]$

$[\phi(T)[\![x_1 \mapsto 1]\!]\cdots[\![x_k \mapsto k]\!][\![y_1 \mapsto k+1]\!]\cdots[\![y_l \mapsto k+l]\!]]_{u'})$ □

A.2 Proof of Theorem 9

For any tree automaton A_P constructed by Definition 7, the following propositions trivially hold.

Proposition 15 For any λ-term M, $\phi(M) \xrightarrow{*}_{A_P} q_\perp$. □

Proposition 16 If $t = S^m(0)$ and m is less than or equal to k in the definition 7, then $S^m(0) \xrightarrow{*}_{A_P} q'_m$. □

Some technical lemmas are necessary in proving Theorem 9.

Lemma 17 Let P be a pattern, and let θ be a substitution such that every free variables does not appear as a bound variable in P. Let $u \in Occ^s(P)$, and let P be represented as $P_{x_1 \cdots x_k}[P|_u]_u$ for some bound variables $x_1, \ldots, x_k$. Then,

$$P\theta\!\downarrow = (P\theta\!\downarrow)_{x_1 \cdots x_k}[(P|_u)\theta\!\downarrow]_u.$$

Hence,

$$(P\theta\!\downarrow)|_u = (P|_u)\theta\!\downarrow .$$

Proof. We prove this lemma by induction on the length $|u|$ of occurrence u. In the case $u = \varepsilon$, $k = 0$, and so the lemma follows trivially. Consider the case $u = vi$. We have only the following two cases because $u \in Occ^s(P)$.

- If $P|_v = @(T_1, T_2)$ and $i \in \{1, 2\}$, we have $P = P_{x_1 \cdots x_k}[@(T_1, T_2)]_v$. Here, $etop(T_1)$ is not a free variable; otherwise $vi \notin Occ^s(P)$. We have $P\theta\!\downarrow = (P\theta\!\downarrow)_{x_1 \cdots x_k}[@(T_1, T_2)\theta\!\downarrow]_v$ by induction. Thus, $P\theta\!\downarrow = (P\theta\!\downarrow)_{x_1 \cdots x_k}[T_i\theta\!\downarrow]_u$.
- If $P|_v = \lambda y(T)$ and $i = 1$, we have $y = x_1$ and $P = P_{x_2 \cdots x_k}[\lambda x_1(T)]_v$. We have $P\theta\!\downarrow = (P\theta\!\downarrow)_{x_2 \cdots x_k}[(\lambda x_1(T))\theta\!\downarrow]_v$ by induction. Since free variables are not used as bound variables, we have $(\lambda x_1(T))\theta\!\downarrow = \lambda x_1(T\theta\!\downarrow)$. Hence, $P\theta\!\downarrow = (P\theta\!\downarrow)_{x_1 \cdots x_k}[T\theta\!\downarrow]_u$. □

Lemma 18 Let P be a pattern, $u \in Occ^s(P)$. In addition, let θ be a substitution, and let k be the maximum of $S^k(0)$ that appears in $\phi(P)$.

(a) $\phi(P\theta\!\downarrow)|_u = @(\phi(P\theta\!\downarrow)|_{u1}, \phi(P\theta\!\downarrow)|_{u2})$ if $P|_u = @(T_1, T_2)$ and $etop(P|_u) \notin FV(P)$

(b) $\phi(P\theta\!\downarrow)|_u = a$ if $P|_u = a \in \mathcal{F}$

(b') $\phi(P\theta\!\downarrow)|_u = \phi(P)|_u = S^n(0)$ for some n $(0 < n \le k)$ if $P|_u \in BV(P)$.

(c) $\phi(P\theta\!\downarrow)|_u = \lambda(\phi(P\theta\!\downarrow)|_{u1})$ if $P|_u = \lambda x(T')$

Proof. Since $P\theta\!\downarrow$ can be written as $(P\theta\!\downarrow)_{x_1 \cdots x_m}[(P\theta\!\downarrow)|_u]_u$ for some bound variables $x_1, \ldots, x_m$ $(m \le k)$, we have

$$\phi(P\theta\!\downarrow) = \phi(P\theta\!\downarrow)[\phi((P\theta\!\downarrow)|_u)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]]_u$$

by Lemma 4. Thus, we have

$$\begin{aligned}\phi(P\theta\!\downarrow)|_u &= \phi((P\theta\!\downarrow)|_u)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi((P|_u)\theta\!\downarrow)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!].\end{aligned}$$

by Lemma 17.

(a) Let $P|_u = @(T_1, T_2)$ and $etop(P|_u) \notin FV(P)$. Then,

$$\begin{aligned}\phi(P\theta\!\downarrow)|_u &= \phi((P|_u)\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi(@(T_1\theta\!\downarrow, T_2\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= @(\phi(T_1\theta\!\downarrow)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &\qquad \phi(T_2\theta\!\downarrow)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!])\end{aligned}$$

On the other hand, we also have the following by Lemma 4 and Lemma 17, because $ui \in Occ^s(P)$ for $i \in \{1, 2\}$:

$$\begin{aligned}\phi(P\theta\!\downarrow)|_{ui} &= \phi((P\theta\!\downarrow)|_{ui})[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi(T_i\theta\!\downarrow)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\end{aligned}$$

Therefore,

$$\phi(P\theta\!\downarrow)|_u = @(\phi(P\theta\!\downarrow)|_{u1}, \phi(P\theta\!\downarrow)|_{u2}).$$

(b) Let $P|_u = a \in \mathcal{F}$. Then,

$$\begin{aligned}\phi(P\theta\!\downarrow)|_u &= \phi((P|_u)\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi(a)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= a\end{aligned}$$

(b') Let $P|_u = x_n \in BV(P)$ for some n $(1 \le n \le m \le k)$. Then

$$\begin{aligned}\phi(P\theta\!\downarrow)|_u &= \phi((P|_u)\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi(x_n)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= x_n[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!])\\ &= S^n(0)\end{aligned}$$

Similarly, using Lemma 4 and Lemma 17, we can show that $\phi(P)|_u = S^n(0)$.

(c) Let $P|_u = \lambda x(T)$. Then,

$$\begin{aligned}\phi(P\theta\!\downarrow)|_u &= \phi((P|_u)\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi((\lambda x(T))\theta\!\downarrow)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \phi(\lambda x(T\theta\!\downarrow))[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \lambda x(\phi(T\theta\!\downarrow)[\![x \mapsto 1]\!])[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!]\\ &= \lambda x(\phi(T\theta\!\downarrow)[\![x \mapsto 1]\!][\![x_1 \mapsto 2]\!] \cdots [\![x_m \mapsto m+1]\!])\end{aligned}$$

On the other hand,

$$P\theta\!\downarrow = (P\theta\!\downarrow)_{xx_1 \cdots x_m}[(P\theta\!\downarrow)|_{u1}]_{u1}.$$

Thus, we have the following by Lemma 4 and Lemma 17 because $u1 \in Occ^s(P)$:

$$\begin{aligned}\phi(P\theta\!\downarrow)|_{u1} &= \phi((P\theta\!\downarrow)|_{u1})[\![x \mapsto 1]\!][\![x_1 \mapsto 2]\!] \cdots [\![x_m \mapsto m+1]\!]\\ &= \phi(T\theta\!\downarrow)[\![x \mapsto 1]\!][\![x_1 \mapsto 2]\!] \cdots [\![x_m \mapsto m+1]\!]\end{aligned}$$

Therefore, $\phi(P\theta\!\downarrow)|_u = \lambda(\phi(P\theta\!\downarrow)|_{u1})$. □

Proof of Theorem 9.

$\supseteq$-*direction*: It is sufficient to show the claim that

$$\phi(P\theta\!\downarrow)|_u \xrightarrow{*}_A q_u$$

for any substitution θ, such that $P\theta\!\downarrow$ is closed and for any position $u \in Occ^s(P)$. This claim is proved by induction on the structure of $P|_u$. We have several cases according to the definition of Occ^s.

(1) In the case of $P|_u \equiv \lambda x(N)$, we have $\phi(P\theta\downarrow)|_u = \lambda(\phi(P\theta\downarrow)|_{u1})$ from Lemma 18(c). Since $\phi(P\theta\downarrow)|_{u1} \xrightarrow{*}_A q_{u1}$ by induction hypothesis, we obtain $\phi(P\theta\!\downarrow)|_u \xrightarrow{*}_A \lambda(q_{u1}) \to_A q_u$ from Definition 7 (1-2-3).

(2-1) In the case of $etop(P|_u) = X \in FV(P)$, we have $\phi(P\theta\!\downarrow)|_u \xrightarrow{*}_A q_\perp \to_A q_u$ by Proposition 15 and Definition 7 (1-1).

(2-2) In the case of $P|_u$ is a function symbol a, we have $\phi(P\theta\downarrow)|_u = a$ from Lemma 18(b). Thus, $a \to_A q_u$ from Definition 7 (1-2-1).

(2-3) In the case of $P|_u$ is a bound variable x, we have $\phi(P\theta\downarrow)|_u = S^n(0) = \phi(P)|_u$ for some $n(>0)$ from Lemma 18(b'). Thus, $S^k(0) \xrightarrow{*}_A q'_k \to_A q_u$ from Definition 7 (2) and Proposition 16.

(3) In the case of $P|_u = @(M,N)$ and $etop(P|_u) \notin FV(P)$, we have $\phi(P\theta\!\downarrow)|_u = @(\phi(P\theta\downarrow)|_{u1}, \phi(P\theta\downarrow)|_{u2})$ from Lemma 18(a). Since $\phi(P\theta\downarrow)|_{ui} \xrightarrow{*}_A q_{ui}$ by induction for $i \in \{1,2\}$, we obtain $\phi(P\theta\!\downarrow)|_u \xrightarrow{*}_A @(q_{u1}, q_{u2}) \to_A q_u$ from Definition 7 (1-2-4).

$\subseteq$-*direction*: It is sufficient to show the claim that

$$t \xrightarrow{*}_A q_u \text{ implies } \exists\theta \text{ s.t. } \phi(P\theta\!\downarrow)|_u = t$$

for any subterm $t = s|_u$ of a closed db-term s. Note that $u \in Occ^s(P)$ for $q_u \in Q$. This claim is proved by induction on the length of $t \xrightarrow{*}_A q_u$. We have several cases according to the rules used in the last step of the sequence.

(1-1) In the case of $t \xrightarrow{*}_A q_\perp \to_A q_u$, the rule of the last step is constructed in Definition 7 (1-1), and so $etop(P|_u)$ is a free variable F. We can write P as $P_{x_1\cdots x_m}[P|_u]_u$ by displaying bound information at the occurrence u. Since P is a free-extended pattern, $P|_u$ is in the form of $@(\cdots @(F, x_{j_1}) \cdots, x_{j_m})$, where $j_1, \ldots, j_m$ is a permutation of $1, \ldots, m$.

On the other hand, there exists a term T such that $t = \phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!]$ for some k (Proposition 5).

Next, we take θ as $F\theta = \lambda x_{j_1}(\cdots \lambda x_{j_m}(T))$. Then, we have $(P\theta\!\downarrow)|_u = (P|_u)\theta\!\downarrow = T$ by Lemma 17. Hence,

$$\begin{aligned} P\theta\!\downarrow &= P\theta\!\downarrow_{x_1\cdots x_m} [(P\theta\!\downarrow)|_u]_u \\ &= P\theta\!\downarrow_{x_1\cdots x_m} [T]_u \end{aligned}$$

Therefore,

$$\begin{aligned} \phi(P\theta\!\downarrow)|_u &= \phi((P\theta\!\downarrow)_{x_1\cdots x_m}[T]_u)|_u \\ &= \phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!] \quad \text{by Lemma 4} \end{aligned}$$

Here, $FV(T) \subseteq \{x_1, \ldots, x_m\}$, and t has no x_i's because t is a subterm of a closed db-term. Thus,

$$\begin{aligned} \phi(P\theta\!\downarrow)|_u &= \phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_m \mapsto m]\!] \\ &= \phi(T)[\![x_1 \mapsto 1]\!] \cdots [\![x_k \mapsto k]\!] \\ &= t. \end{aligned}$$

(1-2-1) In the case of $t \xrightarrow{*}_A a \to_A q_u$, we have $P|_u = a \in \mathcal{F}$ from Definition 7. Thus, $\phi(P\theta\!\downarrow)|_u = a = t$.

(1-2-2) In the case of $t \xrightarrow{*}_A q'_n \to_A q_u$, $P|_u$ is a bound variable and $t = S^n(0) = \phi(P)|_u$ from Definition 7. Thus, $\phi(P\theta\!\downarrow)|_u = t$ follows from $\phi(P\theta\!\downarrow)|_u = \phi(P)|_u$.

(1-2-3) In the case of $t \xrightarrow{*}_A \lambda(q_{u1}) \to_A q_u$, $P|_u$ is in the form of $\lambda x(T)$ from Definition 7. Since $t = \lambda(t_1)$ for some t_1 and $t_1 \xrightarrow{*}_A q_{u1}$, we have $\phi(P\theta)|_{u1} = t_1$ for some θ by induction. Therefore, we have $\phi(P\theta)|_u = \lambda(\phi(P\theta)|_{u1}) = \lambda(t_1) = t$ by Lemma 18 (c).

(1-2-4) In the case of $t \xrightarrow{*}_A @(q_1, q_2) \to_A q_u$, we have $P|_u = @(M_1, M_2)$, and $etop(P)|_u$ is not a free variable from Definition 7. Since $t = @(t_1, t_2)$ for some t_1 and t_2 and $t_i \xrightarrow{*}_A q_i$ $(i \in \{1,2\})$, we have $\phi(P\theta_i)|_{ui} = t_i$ for some θ_i by induction. From the linearity of P, we can take some θ such that $\phi(P\theta)|_{ui} = t_i$ for all $i \in \{1,2\}$. Therefore, $\phi(P\theta\downarrow)|_u = @(\phi(P\theta\downarrow)|_{u1}, \phi(P\theta\downarrow)|_{u2}) = @(t_1, t_2) = t$ by Lemma 18 (a). □

(Received September 28, 2008)
(Accepted December 27, 2008)

Hideto Kasuya completed graduate course of Nagoya University in 1997. He is a Research Associate of the Faculty of Information Science and Technology, Aichi Prefectural University. He is interested in term rewriting system and rewriting strategy. He is a member of IPSJ, IEICE and JSSST.

Masahiko Sakai completed graduate course of Nagoya University in 1989 and became Assistant Professor, where he obtained a D.E. degree in 1992. From April 1993 to March 1997, he was Associate Professor in JAIST, Hokuriku. In 1996 he stayed at SUNY at Stony Brook for six months as Visiting Research Professor. From April 1997, he was Associate Professor in Nagoya University. Since December 2002, he has been Professor. He is interested in term rewriting system, verification of specification and software generation. He received the Best Paper Award from IEICE in 1992. He is a member of IEICE and JSSST.

Kiyoshi Agusa is a professor of Department of Information Systems, Graduate School of Information Science, Nagoya University. He received Ph.D. degree in computer science from Kyoto University in 1982. His research area is software engineering, especially dependable software, programming environment and software reusing. He is a member of ACM, IEEE, IPSJ, IEICE and JSSST.