IEICE TRANS. INF. & SYST., VOL. E80–D, NO. 12 DECEMBER 1997

PAPER

Left-Incompatible Term Rewriting Systems and Functional Strategy

Masahiko SAKAI†*, *Member*

SUMMARY This paper extends left-incompatible term rewriting systems defined by Toyama et al. [17]. It is also shown that the functional strategy is normalizing in the class, where the functional strategy is the reduction strategy that finds index by some rule selection method and top-down and left-to-right lazy pattern matching method.

key words: *term rewriting system, normalizing strategy, lazy evaluation*

1. Introduction

According to lazy evaluation method widely adopted in functional programming language implementations, the evaluation of an expression is postponed until the result is actually needed [2], [5], [11], [18]. If the value of an expression is not used anywhere, it will never be evaluated. Hence, it is often observed that the computation of an expression terminates by lazy method; While it does not terminate by usual call-by-value method, called eager evaluation. For example, let rules of if be

$$if(true, x, y) \to x \text{ and}$$
$$if(false, x, y) \to y.$$

Then $if(g(a), h(b), i(c))$ will be lazily evaluated by the following steps;

(1) Select the first pattern $if(true, x, y)$;

(2) Find a position needed to evaluate in order to decide whether the pattern will be applicable in future. Thus, $g(a)$ is selected and evaluated first;

(3) If the result of evaluation of $g(a)$ is $true$, then reduce $if(g(a), h(b), i(c))$ to $h(b)$ and then evaluate $h(b)$. If not, select the second pattern $if(false, x, y)$ and repeat (2) and (3) in similar way.

Suppose the value of $g(a)$ is $true$ and the value of $h(b)$ can be computed. Then the value of target expression can lazily be computed even if the evaluation of $i(c)$ requires infinite computation steps.

The lazy evaluation is not only efficient but also have a normalizing property as illustrated above. However, it is not so clear the class of programs on which the lazy method works as a normalizing strategy, i.e., the value of an expression can be computed if it exists.

On the other hand, Huet and Lévy proposed a concept of index [6]. They defined a class of strongly sequential term rewriting systems(SS) in which each term not in normal forms has an index and in which index reduction is a normalizing strategy. Their work was extended by Toyama [16], Oyamaguchi [10], Nagaya, Sakai and Toyama [12], [13], Antoy and Middeldorp [1] and Jacquemard [7]. Huet and Lévy presented efficient index-finding algorithm using matching dag [6]. However, no one considered the way to repeat reductions efficiently. Even in strongly sequential systems, redex searching must not be continued from the occurrence where the last reduction has taken place. This is mainly because indices have no transitve property in general. For example, consider the following orthogonal term rewriting system R_1 that is in SS:

$$R_1 : \begin{cases} f(g(x, a)) \to a \\ g(b, c) \to e \\ h \to g(g(b, c), d) \\ d \to a \\ e \to e. \end{cases}$$

All Ω-occurrences in $f(\Omega)$ and $g(\Omega, \Omega)$ are indices. However, the leftmost Ω-occurrence in $f(g(\Omega, \Omega))$ is not an index. This fact leads that index reduction has no locality. The reduction $f(h) \to f(g(g(b, c), d))$ is an index reduction by the third rule of R_1. If we continue reducing indices of the contractum $g(g(b, c), d)$ of previous reduction, we have an infinite reduction $f(g(g(b, c), d)) \to f(g(e, d)) \to f(g(e, d)) \to \cdots$, which is not an index reduction, while there is an index reduction $f(g(g(b, c), d)) \to f(g(g(b, c), a)) \to a$. Thus, we must not continue finding index of t from the position where it is reduced just before.

Toyama et al. proposed transitive system (TS) [17] that is a subclass of SS. TS has the transitivity property of indices. Strandh separately defined forward-branching class (FB) [15], which is the same class as TS shown in later [4]. Durand showed that efficient construction of matching automaton given by Huet and Lévy [6] for simple systems can also be applied to FB [4]. However, the reduction algorithm presented in [15] always goes back to the root position to find an index after a reduction.

Toyama et al. proposed left-incompatible sys-

Manuscript received September 25, 1996.
Manuscript revised March 18, 1997.

†The author is with Japan Advanced Institute of Science and Technology, Ishikawa-ken, 923–12 Japan.
*Presently, with Nagoya University.

tem [17] (we call it SLI in this paper), which is proper subclass of TS. In the system, top-to-down and left-to-right lazy pattern matching method usually used in lazy evaluation can find an index. They introduced the notion of marked term to decrease useless search and proposed an index finding algorithm based on top-to-down and left-to-right lazy pattern matching method. However, they did not mention the way to find an index repeatedly with starting not from its root position.

In this paper, we define left-incompatible system (LI), which is strictly larger class than SLI and still proper subclass of TS. We show an efficient method for checking LI. Then, we show an algorithm to calculate a head normal form of a given term. The algorithm enjoys not only a normalizing property in LI but also it always starts searching an index efficiently from the position where the previous reduction has taken place. Of course, this algorithm can be applied to the class SLI and it is more efficient than that in [17].

2. Term Rewriting Systems

We mainly follow the notation of [8], [17] and assume that readers are familiar with term rewriting systems (TRS's) [3], [8]. Let $\Sigma = V \cup F$ be a signature where F and V are a set of function symbols with arity and a set of variables, respectively. T_Σ (or simply T) denotes the set of terms well constructed by symbols in Σ. The set of variables that appear in term t are denoted by $V(t)$. A rewrite rule is a pair of terms $l \to r$ such that l is not a variable and $V(l) \supseteq V(r)$. A TRS R is a finite set of rewrite rules.

Let $\Box$ be an extra constant. A term $C \in T_{\Sigma \cup \{\Box\}}$ is called a *context* denoted by $C[\ ,\ldots,\ ]$. For $C[\ ,\ldots,\ ]$ containing $n\Box$'s and for $t_1,\ldots,t_n \in T_\Sigma$, $C[t_1,\ldots,t_n]$ denotes the term obtained by replacing $\Box$'s with $t_1,\ldots,t_n$ from left to right order. In particular, $C[\,]$ denotes a context contaning precisely one $\Box$.

Let σ be a substitution. $t\sigma$ denotes an application of σ to a term t. Syntactical equality of terms s and t is denoted by $s \equiv t$. If s is a subterm of t, then we write $s \trianglelefteq t$. $l\sigma$ is called a redex for some rule $l \to r$.

We say that t *reduces* to s by using rule $l \to r$ (on context $C[\,]$), if there exist a substitution σ and a context $C[\,]$ such that $t \equiv C[l\sigma]$ and $s \equiv C[r\sigma]$. We write $t \to s$ when t reduces to s. We use $\xrightarrow{*}$ to denote reflexive and transitive closure of $\to$. $\xrightarrow{+}$ is also used to denote transitive closure of $\to$. If there is no term s such that $t \to s$, then we say t is a *normal form*. If s is a normal form such that $t \xrightarrow{*} s$, then we say s is a normal form of t. A term t is a *head-normal form* if there exists no redex s such that $t \xrightarrow{*} s$.

A TRS is called *left-linear*, if every variable of l occurs only once for every left-hand side l. A TRS is called *ambiguous*, if there exist rules $l \to r$, $l' \to r'$, a term $s \notin V$ such that $s \trianglelefteq l'$ and substitutions σ, σ' such that $l\sigma \equiv s\sigma'$ except trivial case, i.e., the rules are the same and $l \equiv s$. A left-liner and unambiguous TRS is called *orthogonal*. A TRS is *confluent*, if and only if

$$t \xrightarrow{*} s \wedge t \xrightarrow{*} s' \Rightarrow \exists w, s \xrightarrow{*} w \wedge s' \xrightarrow{*} w.$$

A TRS (or $\to$) is *strongly normalizing (or teminating)*, if and only if there is no infinite sequence $t_0 \to t_1 \to \cdots$.

Let $\to_s$ be a sub-relation of $\to$. We say $\to_s$ is a *normalizing strategy*, if $t \xrightarrow{*}_s s$ and there exists no infinite sequence $t \equiv t_0 \to_s t_1 \to_s \cdots$ for any t having a normal form s w.r.t. $\to$.

In this paper, we only deal with orthogonal TRSs.

3. Transitive Systems

In this section, we review transitive system and their properties according to Klop and Middeldorp [9], and Toyama, Smetsers, Eekelen and Plasmeijer [17]. The class of transitive system is a subclass of strongly sequential system [6].

Ω-terms are introduced for representing prefixes of terms.

Definition 3.1 (Ω-terms): Let Ω be an extra constant. Ω-terms are elements in $T_{\Sigma \cup \{\Omega\}}$ (also simply denoted by T_Ω).

- t_Ω denotes the Ω-term obtained from a term t by replacing each variable with Ω.
- The *prefix ordering* $\succeq$ on T_Ω is defined as follows:

$$\begin{array}{ll} t \succeq \Omega & \text{for any } t \in T_\Omega, \\ t \succeq t & \text{for each variable or constant,} \\ f(t_1,\ldots,t_n) \succeq f(s_1,\ldots,s_n) & \\ \qquad \text{if } t_i \succeq s_i \text{ for } i = 1,\ldots,n. & \end{array}$$

- If $u \succeq t$ and $u \succeq s$ for some u, we say t and s are *compatible*, written by $t \uparrow s$; Otherwise they are *incompatible*, denoted by $t \# s$.
- Let $S \subseteq T_\Omega$. Then $t \# S$, if and only if $t \# s$ for any $s \in S$.

Definition 3.2 (Ω-systems): Let R be a term rewriting system.

- The *set of redex schemata of* R is

$$Red = \{l_\Omega \mid l \to r \in R\}.$$

- Ω-*reduction* $\to_\Omega$ is defined on T_Ω as $C[t] \to_\Omega C[\Omega]$ where $t \uparrow s$ for some $s \in Red$ and $t \not\equiv \Omega$.
- NF_Ω is the set of all normal forms w.r.t. Ω-reduction.

Lemma 3.3 ([8]): Ω-reduction is confluent and terminating.

Definition 3.4: The *direct approximant* $\omega(t)$ of an Ω-term t is a normal form of t w.r.t. Ω-reduction.

Example 3.5: Let R_2 be a TRS with the following rules:

$$R_2 : \begin{cases} f(g(x,a)) \to g(x,x) \\ g(b,c) \to f(d). \end{cases}$$

Then, $Red = \{f(g(\Omega,a)), g(b,c)\}, g(f(g(b,c)),a) \to_\Omega g(f(\Omega),a) \to_\Omega g(\Omega,a)$, and $\omega(g(f(g(b,c)),a))) \equiv g(\Omega,a)$.

Lemma 3.6 ([17]):

(a) If $t \trianglerighteq s$ then $\omega(t) \trianglerighteq \omega(s)$.

(b) Let $C[\Omega] \in NF_\Omega$. Then, $C[t] \in NF_\Omega$ for any $t \in NF_\Omega$.

Definition 3.7: A term t is in strong head-normal form if $\omega(t) \not\equiv \Omega$.

Lemma 3.8 ([17]): A term t in strong head-normal form is a head-normal form.

Definition 3.9 (Transitive index):

- Let $C[\ ]$ be a context and let z be a fresh variable. If $z \trianglelefteq \omega(C[z])$, the Ω-occurrence displayed in $C[\Omega]$ is called an *index*, denoted by $C[\Omega_I]$.
- The *index displayed in* $C_1[\Omega_I]$ *is transitive* if the Ω occurrence displayed in $C_2[C_1[\Omega]]$ is an index for any Ω-term $C_2[\Omega_I]$. The transitive index is denoted by $C_1[\Omega_T]$.
- The reduction $t \to s$ on context $C[\ ]$ is called index reduction if $C[\Omega_I]$ and it is called transitive reduction if $C[\Omega_T]$. We write this reduction relation as $\to_I$ and $\to_T$, respectively.

Transitivity allows local search for indices. After an index has been rewritten, the search for the next index can be done from the position of the last rewritten index. Hence, transitive reduction can be performed in efficient depth-first way.

Example 3.10: Let $Red = \{f(g(\Omega,a)), g(b,c)\}$. The Ω occurrence in $g(\Omega,a)$ is an index but not transitive index, because the Ω occurrence in $f(g(\Omega,a))$ is not an index.

Lemma 3.11 ([17]): If $C[\Omega_T]$ and $C'[\Omega_T]$ then $C[C'[\Omega_T]]$.

Definition 3.12: A *term rewriting system is transitive* if each term t not in strong head-normal form has a transitive index.

Every transitive TRS is strongly sequential, i.e., each term not in normal form has a index [17]. Thus, the next lemma follows.

Lemma 3.13: Index reduction $\to_I$ is a normalizing strategy for transitive term rewriting systems.

Definition 3.14: Let R be a term rewriting system.

- $Red^* = \{p \mid p \not\equiv \Omega,\ p \trianglelefteq q \text{ for some } q \in Red\}$.
- $Red^+ = Red^* - Red$.
- $Red^\prec = \{p \mid p \not\equiv \Omega,\ p \prec q \text{ for some } q \in Red\}$.

Lemma 3.15: $p \not\equiv q \Rightarrow p \# q$ for any $p \in Red$ and $q \in Red^*$.

Proof: Trivial from unambiguity. □

Definition 3.16 (Transitive direction):

- Let $Q \subseteq T_\Omega$. The Ω occurrence displayed in $C[\Omega]$ is a *direction for* Q if $C[z] \# Q$. A direction for Q is indicated with $C[\Omega_Q]$.
- A *transitive direction* is defined as a direction for Red^*. A transitive direction is denoted with $C[\Omega_D]$.

The transitive direction gives a sufficient condition of transitivity shown as the following lemmas.

Lemma 3.17 ([17]): If $C[\Omega_D]$ and $C[z] \in NF_\Omega$ then $C[\Omega_T]$.

Lemma 3.18 ([17]): Let R be a TRS. R is transitive iff every $t \in Red^\prec$ has a transitive direction.

We next prepare some technical lemmas used in later.

Definition 3.19:

- If $t \equiv C[s]$, $C[\Omega_T]$ and $C[\ ] \not\equiv \Box$, then we write $t \rhd_T s$.
- The relation $\to_T \cup \rhd_T$ is denoted by $\trianglerighteq$.

Proposition 3.20: If $t \xrightarrow{*}_T s$ and $C[\Omega_T]$, then $C[t] \xrightarrow{*}_T C[s]$.

Proof: Let $t \equiv C'[l\sigma]$ and $s \equiv C'[r\sigma]$ for some $l \to r \in R$, σ and $C'[\Omega_T]$. We have $C[C'[\Omega_T]]$ by lemma 3.11. This indicates $C[t] \equiv C[C'[l\sigma]] \to_T C[C'[r\sigma]] \equiv C[s]$. An easy induction concludes the proof. □

Lemma 3.21: If $t \rhd_T s \to_T u$, then $t \to_T s' \rhd_T u$ for some s'.

Proof: We have $t \equiv C[s]$ and $C[\Omega_T]$ by the definition of $\rhd_T$. Then, we have $C[s] \to_T C[u] \rhd_T u$ by proposition 3.20. We can take $C[u]$ as s'. □

Lemma 3.22: Let R be a transitive TRS. If a term t has a normal form, then there exists no infinite sequences $t \equiv t_0 \trianglerighteq t_1 \trianglerighteq t_2 \trianglerighteq \cdots$.

Proof: Shown by using lemmas 3.13, 3.21 and the fact that $\to_T \subseteq \to_I$. □

4. Left-Incompatible Systems and Functional Strategy

In simple left-incompatible systems (SLI) in [17], an index of a given term is efficiently computed basically by a lazy pattern matching method. In this section, a left-incompatible system (LI) is presented without losing the advantages of SLI system. We then show $SLI \subset LI \subset TS$.

Definition 4.1 (Left-incompatibility): Let $s, t \in T_\Omega$. The *left-incompatibility* of s and t, indicated by $t\#_{\leqq}s$, is defined as follows:

- $t \not\equiv \Omega$, $s \not\equiv \Omega$, and
- $f = g \Rightarrow \exists i[\forall j < i[t_j \preceq s_j] \wedge t_i\#_{\leqq}s_i]$, where $t \equiv f(t_1, \ldots, t_n)$ and $s \equiv g(s_1, \ldots, s_m)$.

Note that the condition $t \not\equiv s$ in the original definition [17] is omitted because it is redundant.

We introduce the following abbreviations.

- $t\#_< s$ for $t\#_{\leqq}s \wedge \neg(s\#_{\leqq}t)$,
- $t\#_= s$ for $t\#_{\leqq}s \wedge s\#_{\leqq}t$, and
- $t\#_\vee s$ for $t\#_{\leqq}s \vee s\#_{\leqq}t$.

Intuitively, $t\#_{\leqq}s$ if and only if each node of t is Ω or the same as the corresponding node of s until the nodes are different constants or function symbols in depth-first order comparison.

Example 4.2:

$f(g(\Omega, a), h(\Omega, b), c)\#_{\leqq}f(g(b, a), h(\Omega, d), \Omega)$.

Lemma 4.3: If $s\#_\vee t$ and $\neg(s\#_{\leqq}t)$, then $t\#_< s$.

Proof: Trivial. □

Lemma 4.4 ([17]): Let $C[\Omega] \uparrow p$ and let $C[\Omega_{\{p\}}]$ be the leftmost direction for $\{p\}$. Then, for any q,

$$p\#_{\leqq}q \Rightarrow C[\Omega_{\{q\}}].$$

Definition 4.5 ([17]): An orthogonal TRS R is *simple left-incompatible (SLI)* if Red can be expressed as a list $[p_1, \ldots, p_n]$ satisfying the following conditions:

- $i < j \Rightarrow p_i\#_{\leqq}p_j$ for any i, j, and
- $\forall q \in Red^+[p_i\#_{\leqq}q]$ for any i.

LI is defined by relaxing essentially the second condition of SLI.

Definition 4.6 (left-incompatible system): An *orthogonal TRS R is left-incompatible (LI)* if there exists a list $SRed = [p_1, \ldots, p_n]$ that satisfies all of the following conditions:

(a) $Red \subseteqq SRed \subseteqq Red^*$,

(b) $i < j \Rightarrow p_i\#_{\leqq}p_j$ for any i, j, and

(c) $\forall q \in (Red^* - SRed)[p_i\#_{\leqq}q]$ for any i.

Example 4.7: Let $Red = \{f(g(\Omega, a)), g(b, c)\}$. The system is LI, since there is $SRed = [f(g(\Omega, a)), g(\Omega, a), g(b, c)]$ satifying the conditions. However, it is not SLI because $\neg(g(b, c)\#_{\leqq}g(\Omega, a))$.

Lemma 4.8: SLI is a proper subclass of LI.

Proof: Let $SRed = Red$. Then the definitions of SLI and LI are equivalent. Thus, $SLI \subseteqq LI$. $SLI \neq LI$ is followed from example 4.7. □

Lemma 4.9: Let R be a left-incompatible TRS with $SRed = [p_1, \ldots, p_d, \ldots, p_n]$. Let $C[\,]$ be a context such that $C[\Omega] \uparrow p_d$, $C[\Omega]\#p_i (1 \leqq i < d)$ and $C[\Omega_{\{p_d\}}]$ displays the leftmost direction for $\{p_d\}$. Then $C[\Omega_D]$, i.e., transitive direction.

Proof: Since $C[\Omega]\#p_i (1 \leqq i < d)$, we have $C[\Omega_{\{p_i\}}](1 \leqq i < d)$. We have $p_d\#_{\leqq}p_j$ for $d < j \leqq n$ and $p_d\#_{\leqq}q$ for $q \in (Red^* - SRed)$ from the definition of LI. Thus, we can show that $C[\Omega_{\{q\}}]$ for any $q \in Red^*$ by lemma 4.4. □

Lemma 4.10: LI is a proper subclass of TS.

Proof: First, let's prove that each $t \in Red^\prec$ has a transitive direction. Letting $t \in Red^\prec$, there exists some $p_d \in SRed$ such that $t\#p_i (i < d)$ and $t \uparrow p_d$. Since $t \succeq p_d$ contradicts unambiguity, we have $t \not\succeq p_d$. Thus, t must have a direction for $\{p_d\}$, from the definition of a direction. By lemma 4.9, the leftmost direction of t for $\{p_d\}$ is a transitive direction. By lemma 3.18, we have $LI \subseteqq TS$. On the other hand, $LI \neq TS$ can be shown from the system $Red = \{f(\Omega, a, a), f(a, \Omega, b)\}$, because of $\neg(f(\Omega, a, a)\#_{\leqq}f(a, \Omega, b))$ and $\neg(f(a, \Omega, b)\#_{\leqq}f(\Omega, a, a))$. □

5. An Efficient Method for Checking LI

LI is decidable because Red^* is finite. However, in order to decide it naively, we must check as $SRed$ every set X such that $Red \subseteqq X \subseteqq Red^*$. Since this method is inefficient, we improve a more efficient method to check LI and also give a construction method of $SRed$.

Lemma 5.1: Let R be orthogonal. R is left-incompatible, if and only if

(a) $\forall p \in Red, q \in Red^*[p \not\equiv q \Rightarrow p\#_\vee q]$, and

(b) $\forall q \in Red^+[\exists p \in Red[\neg(p\#_{\leqq}q)] \Rightarrow \forall q' \in Red^+[q \not\equiv q' \Rightarrow q\#_\vee q']]$.

This lemma is not trivial, because the left-incompatible relation $\#_{\leqq}$ is not a partial order since it does not have transitivity. For example, $f(\Omega, \Omega, 0, 0)\#_< f(\Omega, 1, 1, 1)$ and $f(\Omega, 1, 1, 1)\#_< f(2, 2, \Omega, 2)$, but $\neg(f(\Omega, \Omega, 0, 0)\#_{\leqq}f(2, 2, \Omega, 2))$.

Before proving lemma 5.1, several notions are needed.

Definition 5.2: The lexicographic prefix ordering $\prec_l$ on T_Ω is defined as follows:

$$\begin{aligned} &\Omega \prec_l t \qquad \text{if } t \not\equiv \Omega, \\ &f(t_1, \ldots, t_n) \prec_l f(s_1, \ldots, s_n) \\ &\qquad \text{if } [t_1, \ldots, t_n] \prec\!\prec_l [s_1, \ldots, s_n]. \end{aligned}$$

where $\prec\!\prec_l$ is lexicographic extension of $\prec_l$.

It is easily showen that $\prec_l$ is a partial order on T_Ω.

Proposition 5.3: Let t and s be in T_Ω. Then,

(a) $t \prec s \Rightarrow t \prec_l s$,

(b) $t \prec_l s \Rightarrow \neg(s\#_{\leqq}t)$, and

(c) $t\#_< s \Rightarrow t \prec_l s$.

Proof: (a) is trivial from the definitions.

For the proof for (b), we assume $t \prec_l s$. In case of $t \equiv \Omega \prec_l s \not\equiv \Omega$, we have $\neg(s\#_{\leqq}t)$ from the definition. In case of $t \equiv f(t_1, \cdots, t_n) \prec_l f(s_1, \cdots, s_n) \equiv s$ and $[t_1, \cdots, t_n] \prec\!\!\prec_l [s_1, \cdots, s_n]$, there exists k such that $t_i \equiv s_i (i < k)$ and $t_k \prec_l s_k$. From induction hypothesis, we have $\neg(s_k\#_{\leqq}t_k)$. On the other hand, since $t_k \prec_l s_k$, we have $\neg(s_k \preceq t_k)$ by (a) and transitivity of $\preceq_l$. Hence, we have $\neg(s\#_{\leqq}t)$.

Finally, let's prove (c). We have $t \not\equiv \Omega$ and $s \not\equiv \Omega$ from $t\#_{\leqq}s$. Letting $t \equiv f(t_1, \ldots, t_n)$ and $s \equiv g(s_1, \ldots, s_n)$, we also have $f = g$ from $\neg(s\#_{\leqq}t)$. From these facts, there exists k such that

$$\forall j < k[t_j \preceq s_j] \wedge t_k\#_{\leqq}s_k \tag{1}$$

and we have also

$$\forall i[\exists m < i[s_m \not\preceq t_m] \vee \neg(s_i\#_{\leqq}t_i)]. \tag{2}$$

In case of $s_k\#_{\leqq}t_k$, it follows from (2) that $s_m \not\preceq t_m$ for some $m < k$. Since $t_m \preceq s_m$ from (1), we have $t_m \prec_l s_m$ by (a). Since $\forall j < m[t_j \preceq_l s_j]$ from (1) by (a), we have $[t_1, \ldots, t_n] \prec\!\!\prec_l [s_1, \ldots, s_n]$.

In case of $\neg(s_k\#_{\leqq}t_k)$, we have $t_k\#_< s_k$ from (1). Hence, we have $t_k \prec_l s_k$ by induction hypothesis. On the other hand, we have $\forall j < k[t_j \preceq_l s_j]$ from (1) by (a). These conclude that $[t_1, \ldots, t_n] \prec\!\!\prec_l [s_1, \ldots, s_n]$. Therefore, $t \prec_l s$. □

Lemma 5.4: The relation $\#_<$ on T_Ω is acyclic.

Proof: By proposition 5.3(c). □

Proof of lemma 5.1: Since the 'only if'-part is trivial, let us show the 'if'-part. Let $X = \{q \in Red^+ \mid \forall p \in Red[p\#_{\leqq}q]\}$. Let $SRed = [p_1, \ldots, p_m]$ be a list sorted from $Red^* - X$ topologically with respected to $\#_<$ in increasing order.

Now let's prove that the list $SRed$ satisfies the conditions (a), (b) and (c) of definition 4.6. The condition (a), i.e., $Red \subseteqq SRed \subseteqq Red^*$, is trivial from construction of $SRed$. Before showing the condition (b), we first show that $p_i\#_\vee p_j$ if $i \neq j$. In case that either p_i or p_j is in Red, it is clear from the assumption (a); Otherwise, both of p_i and p_j are in Red^+. Then, $p_i \notin X$. Hence, there exists $p \in Red$ such that $\neg(p\#_{\leqq}p_i)$. By taking p_i as q in the assumption (b), it follows that $q' \in Red^+[p_i \not\equiv q' \Rightarrow p_i\#_\vee q']$. Thus, we have $p_i\#_\vee p_j$. Consider the condition (b). If we assume $\neg(p_i\#_{\leqq}p_j)$ for $i < j$, it follows that $p_j\#_{\leqq}p_i$. Then, we have $p_j\#_< p_i$, which is contradictory to the fact that $SRed$ is topologically sorted.

Let's prove the condition (c). Let $p_i \in SRed$ and $q \in (Red^* - SRed) = X$. In case of $p_i \in Red$, we have $p_i\#_{\leqq}q$ from the construction of X. Next, consider the case that $p_i \in Red^+$. Since $p_i \notin X$, there exists $p_j \in Red \subseteqq SRed$ such that $\neg(p_j\#_{\leqq}p_i)$. It follows from the condition (b) that $p_i\#_< p_j$. Hence, by proposition 5.3 (c) we have

$$p_i \prec_l p_j. \tag{3}$$

On the other hand, by taking p_i as q and taking p_j as p in the assumption (b), it follows that $q' \in Red^+[p_i \not\equiv q' \Rightarrow p_i\#_\vee q']$. Since $q \in X \subseteqq Red^+$ and $q \not\equiv p_i$, we have

$$p_i\#_\vee q. \tag{4}$$

Now we assume that $\neg(p_i\#_{\leqq}q)$. Since $q\#_{\leqq}p_i$ from (4), we have $q\#_< p_i$. Then, $q \prec_l p_i$ follows by proposition 5.3 (c). Hence, we have $q \prec_l p_j$ from (3). It follows by propsition 5.3 (c) that $\neg(p_j\#_{\leqq}q)$. On the other hand, since $q \in X$ and $p_j \in Red$, we have $p_j\#_{\leqq}q$ from the construction of X, which is a contradiction. □

From the proof of lemma 5.1, we can construct a list $SRed$ from Red of left-incompatible TRS.

Example 5.5: Let $Red = \{f(g(a, a, \Omega)), f(g(b, \Omega, a)), g(b, a, b), g(c, \Omega, \Omega)\}$. One of $SRed$ of the system is

$$[f(g(a, a, \Omega)), f(g(b, \Omega, a)), g(c, \Omega, \Omega), g(b, \Omega, a), g(b, a, b)]$$

by proof of lemma 5.1.

6. A Normalizing Reduction Procedure for LI

This section formalizes the procedure usually used in implementation of lazy evaluation. We then show the procedure has a normalizing property in LI.

First, we need to introduce the notion of well-marked terms [17]. Occurrences of subterms are marked when the subterms are known to be in strong head-normal form. Thus, once a defined symbol is marked, it can be treated as a constructor.

Definition 6.1: Let R be a TRS.

- $root(t)$ denotes outermost symbol of a term t.
- Let $D = \{root(l) \mid l \to r \in R\}$ be the set of defined symbols. The *set of marked symbols* D^* is defined as $\{f^* \mid f \in D\}$, where each f^* is a new symbol having the same arity as that of f. The *set of marked terms* is $T_{\Sigma \cup D^*}$, simply written by T^*.
- Let t be a marked term. $e(t)$ denotes the term obtained from t by erasing all marks. $\delta(t)$ denotes the Ω-term obtained from t by replacing all subterms satisfying $root(t) \in D$ with Ω. $\overline{\delta}(t)$ denotes $f(\delta(t_1), \ldots, \delta(t_n))$, where $t \equiv f(t_1, \ldots, t_n) (0 \leqq n)$.

Definition 6.2: $t \in T^*$ is *well marked* if

$$\forall s \trianglelefteq t[root(s) \in D^* \Rightarrow e(\delta(s)) \in NF_\Omega].$$

Note that $t \in T$ is well marked.

Lemma 6.3: Let $t \in T^*$ be well marked. Then, s is well marked for any $s \trianglelefteq t$.

Proof: Trivial. □

Lemma 6.4: $t \in T^*$ is well marked if and only if $e(\delta(s)) \in NF_\Omega$ for any $s \trianglelefteq t$.

Proof: Easy induction on the size of t. □

Lemma 6.5: Let $t \in T^*$ be well marked. If $root(t) \notin D$, then $e(t)$ is in strong head-normal form.

Proof: Since $root(t) \notin D$, we have $e(\delta(t)) \not\equiv \Omega$. It follows from $e(t) \succeq e(\delta(t))$ by lemmas 3.6 (a) and 6.4 that $\omega(e(t)) \succeq \omega(e(\delta(t))) \equiv e(\delta(t)) \not\equiv \Omega$. □

Reduction on marked term is defined as follows.

Definition 6.6: Let $t, s \in T^*$. $t \to s$ if there exists a context $C[\ ]$, substitution σ, $l \to r \in R$ and $l' \in T^*$ such that $e(l') \equiv l$, $t \equiv C[l'\sigma]$ and $s \equiv C[r\sigma]$.

Reduction on marked terms ignores the marked information in matching. However, the marked information in the substitution is not ignored.

Example 6.7: Consider the TRS defined in example 3.5. Then, $f(g^*(f^*(b), a)) \to g(f^*(b), f^*(b))$.

Lemma 6.8: Let t be a well-marked term. If $t \to s$, then s is well marked.

Proof: Let $t \equiv C[t']$, $s \equiv C[s']$, $t' \equiv l'\sigma$, $s' \equiv r\sigma$ and $e(l') \equiv l$ for some rule $l \to r$.

Since $t \equiv C[l'\sigma]$ is well marked, $x\sigma$ is well marked for each variable $x \in V(r) \subseteq V(l')$ by lemma 6.3. It follows from $r \in T$ that $s' \equiv r\sigma$ is well marked.

Therefore, it is enough to show that $e(\delta(C'[s'])) \in NF_\Omega$ for any $C'[\ ] \trianglelefteq C[\ ]$ such that $root(C'[s']) \in D^*$. Let $C''[\Box] \equiv e(\delta(C'[\Box]))$. Since $C'[t']$ is well marked by lemma 6.3, we have $C''[\Omega] \equiv e(\delta(C'[t'])) \in NF_\Omega$. Since s' is well marked, we also have $e(\delta(s')) \in NF_\Omega$ by lemma 6.4. Hence, $e(\delta(C'[s'])) \equiv C''[e(\delta(s'))] \in NF_\Omega$ by lemma 3.6(b). □

We now formalize the procedure that computes strong head-normal forms. Then, we show that this procedure eventually terminates and return a head-normal form of a given term if it has a normal form. The normalizing procedure to compute a normal form of a term can be easily constructed by applying this procedure repeatedly in top-to-bottom order.

Definition 6.9 ([17]): Let $p_d \in T_\Omega$ and let $t \equiv C[t_1, \ldots, t_k, \ldots, t_m] \in T^*$ and $\overline{\delta}(t) \equiv C[\Omega, \ldots, \Omega, \ldots, \Omega]$. Furthermore, let $e(C)[\Omega, \ldots, \Omega_{\{p_d\}}, \ldots, \Omega]$ display the leftmost direction for $\{p_d\}$. Then we say that t_k is the *leftmost directed subterm of t with respect to p_d*.

Definition 6.10 (Functional Strategy): Let R be a left-incompatible TRS with $SRed = [p_1, \ldots, p_n]$. The procedure $HNF(t)$ computing the strong head-normal form of t is defined as follows.

Input: A well-marked term $t \in T^*$.

Output: A well-marked term t' such that $e(t')$ is a strong head-normal form of $e(t)$.

Procedure $HNF(t)$:

(1) Find the first compatible pattern p_d to $e(\overline{\delta}(t))$ in the list $SRed$ if it exists; Otherwise go to (5).

(2) If $e(\overline{\delta}(t)) \succeq p_d$ and $p_d \in Red^+$, go to (5)

(3) If $e(\overline{\delta}(t)) \succeq p_d$ and $p_d \in Red$, rewrite t to t' by the corresponding rule to p_d, and return $HNF(t')$.

(4) Let s be the leftmost directed subterm of t w.r.t. p_d. Replace s in t by $HNF(s)$ and let s' be the resulting term. Then, return $HNF(s')$.

(5) If $root(t) \in D$ then return $mark(t)$, else return t, where $mark(f(t_1, \ldots, t_n))$ denotes $f^*(t_1, \ldots, t_n)$.

Note that the step (2) has no meaning in SLI, because $SRed \cap Red^+$ is an empty set.

Lemma 6.11: Let R be a left-incompatible TRS and let $t \in T^*$ be well marked. If there exists no infinite sequences $e(t) \equiv t_0 \rhd\!\!\rhd t_1 \rhd\!\!\rhd t_2 \rhd\!\!\rhd \cdots$, then

i) $e(t) \xrightarrow{*}_T e(HNF(t))$,

ii) $root(HNF(t)) \notin D$, and

iii) $HNF(t)$ is well marked.

Proof: We prove by induction on the lexicographic composition of $\rhd\!\!\rhd$ and the number of unmarked symbols in t denoted by $|\ t\ |_{um}$.

- In case of the execution path (1)-(5), i) and ii) are obvious. Since t is well marked, $\forall s \trianglelefteq t[root(s) \in D^* \Rightarrow e(\delta(s)) \in NF_\Omega]$. Since $HNF(t)$ may differ only in root symbol from t and $e(\delta(HNF(t))) \equiv e(\overline{\delta}(t))$, we have to show that $e(\overline{\delta}(t)) \in NF_\Omega$, which follows from condition in (1). Hence iii) is shown.

- In case of the execution path (1)-(2)-(5), the proof is almost the same as above. The difference is as follows: since $e(\overline{\delta}(t)) \succeq p_d \in Red^+$, there is no compatible $p_i \in Red$ to $e(\overline{\delta}(t))$ from unambiguity. Hence, $e(\overline{\delta}(t)) \in NF_\Omega$.

- In case of the execution path (1)-(2)-(3), the reduction $t \to t'$ in (3) is obviously a transitive index reduction of t, because it is a reduction on root position. Therefore, i) - iii) hold with respect to t' by induction hypothesis. Hence, i) - iii) clearly hold.

- In case of the execution path (1)-(2)-(3)-(4), let $t \equiv C[s]$. $C[\Omega] \uparrow p_d$ and $C[\Omega] \# p_i (1 \leq i < d)$ follow from (1), and $C[\Omega_{\{p_d\}}]$ displays the leftmost direction for $\{p_d\}$. Since $C[\Omega_D]$ by lemma 4.9 and $C[z] \in NF_\Omega$, we have $C[\Omega_T]$ by lemma 3.17. Hence $t \rhd_T s$. By induction hypothesis, i) — iii) hold with respect to s. It follows that s' is well marked by lemma 6.8.

– If $e(s) \equiv e(HNF(s))$, we have $|HNF(s)|_{um} < |s|_{um}$ because $root(s)$ is in D and $root(HNF(s))$ is not.

– Otherwise, since $s \xrightarrow{+}_T HNF(s)$, we have $t \equiv C[s] \xrightarrow{+}_T C[HNF(s)] \equiv s'$ by proposition 3.20.

In both of cases above, i) — iii) hold with respect to s'. From these facts we can easily conclude i) — iii) w.r.t. t. □

Theorem 6.12: Let R be a left-incompatible TRS and let $t \in T$ have a normal form. The procedure $HNF(t)$ eventually terminates and term $t' \equiv e(HNF(t))$ is a head normal form of t.

Proof: Direct consequence of lemma 3.8, 3.13, 3.22, 6.5 and 6.11.

7. Concluding Remarks

By extending the class SLI, we have proposed the class LI and showed that functional strategy, which works efficiently in similar way as usual lazy evaluation method, is a normalizing strategy of R in LI.

The functional strategy obeys the priority restriction between rules in priority term rewriting systems [14]. Thus, it may be interesting to examine the class of priority systems that the functional strategy is normalizing in.

It may be possible to extend LI by introducing permutations of arguments for each defined function symbol. Although the problem to find the permutations and $SRed$ is interesting, the class is still proper subclass of TS, which can be shown by following example:

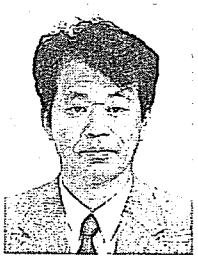
$$Red = \{ \begin{array}{ll} f(c,a,a,\Omega), & f(c,b,\Omega,a), \\ f(d,a,a,\Omega), & f(d,\Omega,b,a), \\ f(e,a,\Omega,a), & f(e,\Omega,a,b) \end{array} \}.$$

Acknowledgment

We would like to thank Referees of IEICE for useful suggestion. We also thank Prof. Yoshihito Toyama, Mr. Takashi Nagaya, Prof. Yasuyoshi Inagaki and Prof. Toshiki Sakabe for the discussions on this work. This work is partially supported by Grants from Ministry of Education, Science and Culture of Japan, #07680350.

References

[1] S. Antoy and A. Middeldorp, "A sequential reduction strategy," LNCS, vol.850, pp.168–185, 1995.
[2] A.J.T. Davie, "An introduction to functional programming systems using haskell," chap.8, Cambridge University Press, 1992.
[3] N. Dershowitz and J.-P. Jouannaud, "Rewrite systems," Handbook of Theoretical Computer Science, vol.B, pp.243–320, North-Holland, 1990.
[4] I. Durand, "Bounded, strongly sequential and forward-branching term rewriting systems," J. Symbolic Computation, vol.18, pp.319–352, 1994.
[5] P. Henderson, "Functional programming," chap.8, Prentice Hall International, 1980.
[6] G. Huet and J.-J. Lévy, "Call by need computations in non-ambiguous linear term rewriting systems," Technical Report 359, INRIA, 1979.
[7] F. Jacquemard, "Decidable approximations of term rewriting systems," LNCS, vol.1103, pp.362–376, Springer-Verlag, 1996.
[8] J.W. Klop, "Term rewriting systems," Handbook of Logic in Computer Science, vol.I, Oxford University Press, 1992.
[9] J.W. Klop and A. Middeldorp, "Sequentiality in orthogonal term rewriting systems," J. Symbolic Computation, vol.12, pp.161–195, 1991.
[10] M. Oyamaguchi, "NV-sequentiality: A decidable condition for call-by need computations in term rewriting systems," SIAM Journal on Computation, vol.22, no.1, pp.113–135, 1993.
[11] S.L.P. Jones, "The implementation of functional programming languages," chap.11, Prentice Hall International, 1987.
[12] T. Nagaya, M. Sakai, and Y. Toyama, "NVNF-Sequentiality of Left-Linear Term Rewriting Systems," RIMS Technical Report, vol.918, University of Kyoto, pp.109–117. 1995.
[13] T. Nagaya, M. Sakai, and Y. Toyama, "Index reduction of Overlapping Strong Sequential Systems," IEICE Technical Report, SS96-32, pp.39–48, 1996.
[14] M. Sakai and Y. Toyama, "Semantics and strong sequentiality of priority term rewriting systems," LNCS, vol.1103, pp.377–391, 1996.
[15] R.I. Strandh, "Classes of equational programs that compile into efficient machine code," LNCS, vol.355, pp.449–461, 1989.
[16] Y. Toyama, "Strong sequentiality of left-linear overlapping term rewriting systems," Proc. on Logic in Computer Science, pp.274–284. 1992.
[17] Y. Toyama, S. Smetsers, M. van Eekelen, and R. Plasmeijer, "The functional strategy and transitive term rewriting systems," Term Graph Rewriting: Theory and Practice, pp.61–75, John Wiley & Sons Ltd., 1993.
[18] D. Turner, "Miranda: A non-strict functional language with polimorphic types," LNCS, vol.201, pp.1–16, 1985.



Masahiko Sakai was born on June 10 1961. He 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 Researh Professor. Since April 1997, he has been Associate Professor in Nagoya University. He is interested in term rewriting system, verification of specification and software generation. He was received the Best Paper Award from IEICE in 1992. He is member of IPSJ.