PAPER

# User-Defined On-Demand Matching*

**Masaki NAKAMURA**[†a)], **Kazuhiro OGATA**[††], ***Members***, ***and*** **Kokichi FUTATSUGI**[††], ***Nonmember***

**SUMMARY** We propose a user-defined on-demand matching strategy, called O-matching, in which users can control the order of matching arguments of each operation symbol. In ordinary matching schemes it is not important to set the order of matching, however, in on-demand matching schemes, it is very important since an input term may be changed while doing the on-demand matching process. O-matching is suitable to combine with the E-strategy, which is a user-defined reduction strategy in which users can control the order of reducing arguments. We show a sufficient condition under which the E-strategy with O-matching is correct for head normal forms, that is, any reduced term is a head normal form.

***key words:*** *the evaluation strategy, lazy evaluation, on-demand matching, term rewriting*

## 1. Introduction

For a given term rewriting system (TRS), a term may have more than one redex, which is an instance of the left-hand side of a rewrite rule, and may be reduced into more than one term, in general. A reduction strategy is a procedure to choose one reduced term from the several reduced terms of a given term. Many reduction strategies have been proposed and discussed [18]–[20]. The evaluation strategy (the E-strategy) is a user-defined reduction strategy [3]–[6], in which we can give a different strategy not only for each TRS but also for each operation symbol in the TRS. The E-strategy takes a term and returns a reduced term in accordance with local strategies assigned to operation symbols.

Matching is another fundamental procedure for implementing TRSs. When a subterm is chosen as a candidate of a redex according to a reduction strategy, we will check whether the subterm can be a redex, that is, an instance of the left-hand side of a rewrite rule in the TRS. While the usual matching process is static, that is, it does not change the subterm, on-demand matching is a more active matching. On-demand matching may reduce a subterm if the subterm is an obstacle in the matching process. On-demand matching is adopted by the functional strategy [18], [20], the E-strategy with on-demand flags [15], [17] and so on.

We show a typical example for on-demand matching. Consider the TRS $R$ whose rewrite rules are

$$\begin{aligned} 2nd(x; y; z) &\to y \\ from(x) &\to x; from(s(x)) \end{aligned}$$

where $x; y; z; \cdots$ is an abbreviation of the term $cons(x, cons(y, cons(z, \ldots)))$. $cons$ represents the list constructor. $2nd$ represents a function which takes a list and returns the second component of the list. For example, when $2nd$ takes $[0, 1, 2, 3]$, it returns 1. From the second rewrite rule, the term $from(0)$ represents the infinite list $[0, 1, 2, \ldots, n, n + 1, \ldots]$. Consider the term $2nd(from(0))$. It is expected to return $s(0)$ which is the second component of the infinite list. However, if the TRS is implemented naively, reduction may not terminate because of an infinite rewrite sequence

$$\begin{aligned} &2nd(from(0)) \\ &\to_R 2nd(0; from(s(0))) \\ &\to_R 2nd(0; s(0); from(s(s(0)))) \\ &\to_R 2nd(0; s(0); s(s(0)); from(s(s(s(0))))) \\ &\to_R \cdots. \end{aligned}$$

On-demand matching can solve the problem. When on-demand matching tries to match the target term $2nd(from(0))$ with the left-hand side $2nd(x; y; z)$, the subterm $from(0)$ is expanded until the matching succeeds. The expansion of $from(0)$ terminates with $0; s(0); from(s(s(0)))$ which is an instance of $x; y; z$. Then, it is rewritten into $s(0)$ with the first rewrite rule.

In usual matching schemes, the order of arguments to be matched does not influence the result of matching. However, the order of matching becomes very important for on-demand matching since it may change a target term while doing the matching process. The motivation of our study is to give a framework by which we can choose a suitable order of matching for each operation symbol in each TRS. In this paper, we propose user-defined on-demand matching. In our on-demand matching framework, we assign a matching list to each operation symbol, and the matching process is done according to the matching lists, like the reduction process of the E-strategy. We generalize the notion of the left-normality, called the $\varphi'$-normality, and by using the $\varphi'$-normality we give a sufficient condition under which any result term of O-matching is either a redex or a head normal form. We also propose a combination of the E-strategy and O-matching, and give a sufficient condition under which the E-strategy with O-matching is correct for head normal forms, that is, any reduced term is a head normal form.

Manuscript received December 5, 2008.
Manuscript revised January 29, 2009.

[†]The author is with the Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa-shi, 920–1192 Japan.

[††]The authors are with the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomishi, 923–1292 Japan.

*A preliminary version of parts of this article appeared in [16].

a) E-mail: masaki-n@is.t.kanazawa-u.ac.jp

DOI: 10.1587/transinf.E92.D.1401

## 2. Preliminaries

We introduce the fundamentals of the TRS and the E-strategy according to the literatures [3]–[6], [11], [12], [19].

### 2.1 Term Rewriting System

A *signature* $\Sigma$ is a non-empty set of *operation symbols* each of which has its arity: $ar(f) \in \mathcal{N}$ for each $f \in \Sigma$, where $\mathcal{N}$ is the set of all natural numbers. Let $V$ be a countable set of *variables* which is disjoint to the signature ($\Sigma \cap V = \emptyset$). The set $\mathcal{T}(\Sigma, V)$ of the *terms* constructed from $\Sigma$ and $V$ is defined as the smallest set satisfying the following: (1) $V \subseteq \mathcal{T}(\Sigma, V)$, and (2) $f(\vec{t}) \in \mathcal{T}(\Sigma, V)$ if $f \in \Sigma, ar(f) = n, \vec{t} \in \mathcal{T}(\Sigma, V)$. Here, $\vec{t}$ stands for $t_1, \ldots, t_n$. Hereafter we may use similar notations. We may use $\mathcal{T}$ instead of $\mathcal{T}(\Sigma, V)$. A *position* of a given term is referred by a sequence $p \in \mathcal{N}_+^*$ of positive integers, where $\mathcal{N}_+$ is the set of all positive integers and $A^*$ is the set of all sequences of $A$'s elements, where $\varepsilon$ stands for the empty sequence. The set $O(t)$ of the positions of a term $t$ is defined recursively as follows: $O(x) = \{\varepsilon\}$ and $O(f(\vec{t})) = \{\varepsilon\} \cup \{i.p \mid 1 \leq i \leq ar(f), p \in O(t_i)\}$ where $x \in V$, $f \in \Sigma$ and $\vec{t} \in \mathcal{T}$. The partial order $>$ on $O(t)$ is defined as follows: $p > q$ iff $p = q.q'$ for some non-empty sequence $q' \neq \varepsilon$. A variable or an operation symbol of a term $t \in \mathcal{T}$ at a position $p \in O(t)$ is denoted by $t(p)$. A *subterm* at a position $p \in O(t)$ is denoted by $t|_p$. The result of replacing the subterm $t|_p$ with $s$ is denoted by $t[s]_p$. We call $t(\varepsilon)$ the *root symbol* of a term $t$. For a term $t$, the sets of the positions of all variables and all operation symbols in $t$ are denoted by $O_V(t)$ and $O_\Sigma(t)$ respectively. The sets of all variables and all operation symbols in $t$ are denoted by $V(t)$ and $\Sigma(t)$ respectively. A map $\theta : V \to \mathcal{T}$ is called a *substitution*. The result term of replacing all variables $x$ in a term $t$ with $\theta(x)$ is denoted by $t\theta$, and called an *instance* of $t$. A *rewrite rule*, denoted by $l \to r$, is a pair of terms $l$ and $r$ which satisfies $V(r) \subseteq V(l)$ and $l \notin V$. The pair $(\Sigma, R)$ of a signature $\Sigma$ and a set $R \subseteq T(\Sigma, V) \times T(\Sigma, V)$ of rewrite rules is called a *term rewriting system* (*TRS*). We assume $R$ is non-empty. We often omit $\Sigma$ and say TRS $R$. The sets of the *left-hand sides* of the rewrite rules, the *defined symbols*, the *constructors* and the *redexes* of a TRS $R$ are defined as follows: $\mathcal{L}(R) = \{l \in \mathcal{T} \mid l \to r \in R\}$, $\mathcal{D}(R) = \{l(\varepsilon) \in \Sigma \mid l \to r \in R\}$, $\mathcal{C}(R) = \Sigma \setminus \mathcal{D}(R)$ and $Redex(R) = \{l\theta \in \mathcal{T} \mid \theta \in \mathcal{T}^V, l \to r \in R\}$, respectively. $B^A$ denotes the set of all maps from $A$ to $B$. A term $t$ is called a *normal form* if it cannot be rewritten, i.e. $\forall p \in O(t).(t|_p \notin Redex(R))$. The set of normal forms is denoted by $NF(R)$. When $R$ is obvious, we omit it from the above sets, like $\mathcal{L}, \mathcal{D}, \mathcal{C}, Redex$ and $NF$. A term is *linear* if it has no duplicated variables, i.e. $t(p) = t(q) \in V$ implies $p = q$. A TRS is *left-linear* if each $l \in \mathcal{L}$ is linear. A term $t \in \mathcal{T}(\mathcal{C}, V)$ is called a *constructor term*. A TRS is called a *constructor TRS* iff each $l \in \mathcal{L}$ forms $f(\vec{t})$ for some constructor terms $\vec{t} \in T(\mathcal{C}, V)$, i.e. $l(p) \in \mathcal{D}$ implies $p = \varepsilon$. The *rewrite relation* $\to_R \subseteq \mathcal{T} \times \mathcal{T}$ of a TRS $R$ is the smallest binary relation satisfying the following.

$$s \to_R t \overset{\text{def}}{\iff} \exists p \in O(s).(s \to_{R,p} t).$$
$$s \to_{R,p} t \overset{\text{def}}{\iff} \exists l \to r \in R, \theta \in \mathcal{T}^V.(s|_p = l\theta, t = s[r\theta]_p)$$

We often use $\to_p$ instead of $\to_{R,p}$. The reflexive and transitive closure and the transitive closure of a binary relation $\to$ are denoted by $\to^*$ and $\to^+$ respectively.

### 2.2 The E-Strategy

In *the E-strategy*, a term is reduced according to local strategies. A *local strategy* is a natural numbers list given to an operation symbol. Each positive integer $i > 0$ in the list corresponds to the $i$-th argument of the operation symbol $f$, and 0 stands for the whole term $f(\ldots)$. Let $L = [2, 1, 0, 4]$ be a local strategy for $f \in \Sigma$. Let $t_i$ be terms and let $t_i'$ be terms reduced from $t_i$ ($i = 1, 2, 3, 4$). Then, $f(t_1, t_2, t_3, t_4)$ is reduced as follows: (1) The second argument $t_2$ is reduced since the top of the list is 2. The result term is $f(t_1, t_2', t_3, t_4)$. (2) $t_1$ is reduced since the next element of the list is 1. The result is $f(t_1', t_2', t_3, t_4)$. (3) [**matching**] Since the next element is 0, it is checked whether $f(t_1', t_2', t_3, t_4)$ is a redex or not. (4a) If so, a possible rewrite rule is applied. The rewritten term is reduced according to the new local strategy of its root symbol. (4b) Otherwise, 0 is ignored and then $t_4$ is reduced. When all elements of the list are consumed, the result term $f(t_1', t_2', t_3, t_4')$ is returned as the reduced term. Note that if $i \notin L$ then the $i$-th argument should not be reduced [†]. Local strategies are formalized by a map $\varphi : \Sigma \to List(\mathcal{N})$ [††].

**Definition 2.1:** ([4]) A map $\varphi : \Sigma \to List(\mathcal{N})$ is called an *E-strategy map* (*E-map*) iff $\varphi(f) \subseteq \{0, 1, \ldots, ar(f)\}$ for each $f \in \Sigma$. $\varphi(f)$ is called a local strategy of $f$. An E-map $\varphi$ is extended to the map from $V \cup \Sigma$ in which $\varphi(x) = nil$ for each $x \in V$. □

The $i$-th element of a list $L$ is denoted by $L(i)$. Hereafter, for simplicity, we assume that (a) $0 \in \varphi(f)$ implies $f \in \mathcal{D}$ and (b) $\varphi(f)(i) = \varphi(f)(j) > 0$ implies $i = j$. The former is because any term whose root symbol is not a defined symbol cannot be a redex. The latter is because it is known that $j \in \varphi(f)$ has no effect on reduction if $i \in \varphi(f)$ and $i < j$ under some natural assumption [4].

Reduction under $\varphi$ is given as a function $red : \mathcal{T} \to \mathcal{T}$ and an auxiliary function $eval : \mathcal{T} \times List(\mathcal{N}) \to \mathcal{T}$ [4]. We formalize the reduction under $\varphi$ as a rewrite relation on $\mathcal{T}'$, where $\mathcal{T}' = T(\Sigma', V)$, and $\Sigma'$ is obtained by adding special operation symbols *red*, *eval* and list constructors *nil* and ; to a given $\Sigma$.

**Definition 2.2:** ([4]) Let $(\Sigma, R)$ be a TRS and $\varphi$ be an E-map. Then, $\Rightarrow^\varphi \subseteq \mathcal{T}' \times \mathcal{T}'$ [†††] is defined as the smallest set satisfying the following: For any $i \in \mathcal{N}_+$, $is \in List(\mathcal{N})$, $t$,

[†]We may regard a list $[\overrightarrow{i_n}]$ as the set $\{\overrightarrow{i_n}\}$ and use the set notations, e.g. $\in$, $\subseteq$, and so on.

[††]$List(A)$ is the set of lists whose elements are in $A$. Besides the list notation like $[a, b, c]$, we use the list constructors "*nil*" (the empty list) and "; ". E.g. $a; b; nil = [a, b]$.

[†††]We may write $\Rightarrow$ instead of $\Rightarrow^\varphi$

$t' \in \mathcal{T}$,

$$\begin{array}{rll} red(t) \Rightarrow & eval(t, \varphi(t(\varepsilon))) & \\ eval(t, nil) \Rightarrow & t & \\ eval(t, i; is) \Rightarrow & eval(t[t']_i, is) & \textbf{if } red(t|_i) \Rightarrow^* t' \\ eval(t, 0; is) \Rightarrow & \left\{ \begin{array}{ll} red(t') & \textbf{if } t \to_\varepsilon t' \\ eval(t, is) & \textbf{if } t \notin Redex. \end{array} \right. & \end{array}$$

□

Notice that $t$ and $t'$ are in $\mathcal{T}$, that is, they do not contain special operation symbols *red*, *eval*, and so on. When $red(t) \Rightarrow t'$ $(t, t' \in \mathcal{T})$, we say the E-strategy reduces $t$ into $t'$ under $\varphi$. The reduced term under $\varphi$ is not always unique since more than one rewrite rule may be applied to a given term $t$, i.e. there may be plural terms $t'$ such that $t \to_\varepsilon t'$.

**Example 2.3:** We give a typical example which shows the usefulness of the E-strategy: a lazy evaluation for lists.

$$R_{\text{inf}} = \left\{ \begin{array}{rcl} hd(x; y) & \to & x \\ tl(x; y) & \to & y \\ 0s & \to & 0; 0s \end{array} \right.$$

Operation symbols *hd* and *tl* represent functions which take a list and return the first element and the remaining list respectively. A constant $0s$ represents an infinitely long list $0; 0; 0; \cdots$. Let $\varphi$ be a local strategy satisfying $\varphi(hd) = [1, 0]$, $\varphi(tl) = [1, 0]$, $\varphi(cons) = nil$, $\varphi(0) = nil$ and $\varphi(0s) = [0]$. Note that the local strategy of *cons* is empty, i.e. any reduction of its arguments is restricted. Then, we obtain the rewrite sequences $red(0s) \Rightarrow eval(0s, [0]) \Rightarrow red(0; 0s) \Rightarrow eval(0; 0s, nil) \Rightarrow 0; 0s$, and $red(tl(0s)) \Rightarrow eval(tl(0s), [1, 0]) \Rightarrow eval(tl(0; 0s), [0]) \Rightarrow red(0s) \Rightarrow^* 0; 0s$, where $0; 0s$ cannot be rewritten any more since $\varphi(cons) = nil$. We expect $hd(tl(0s))$ to be reduced into 0, which is the second component of the infinite list $0; 0; \cdots$ represented by $0s$. Reduction under $\varphi$ works it well as follows:

$$\begin{array}{ll} red(hd(tl(0s))) & \Rightarrow eval(hd(tl(0s)), [1, 0]) \\ & \Rightarrow eval(hd(0; 0s), [0]) \\ & \Rightarrow red(0) \Rightarrow eval(0, nil) \Rightarrow 0 \end{array}$$

Note that the E-strategy avoids an infinite loop of $0s \to 0; 0s \to 0; 0; 0s \to \cdots$. □

## 3. User-Defined On-Demand Matching

Consider TRS $R_{2nd} = R_{\text{inf}} \cup \{2nd(x; y; z) \to y\}$. How to give a suitable $\varphi$ to obtain the following rewrite sequence: $2nd(0s) \to_1 2nd(0; 0s) \to_{1.2} 2nd(0; 0; 0s) \to_\varepsilon 0$? If $2 \in \varphi(cons)$, reducing $0s$ causes an infinite rewrite sequence. If $2 \notin \varphi(cons)$, the second rewrite step $2nd(0; 0s)) \to_{1.2} 2nd(0; 0; 0s)$ cannot be done. Thus, we cannot give a suitable E-map for $R_{2nd}$. Let us analyze the above rewrite sequence. We can see that the above rewrite sequence expands the argument $0s$ to make the matching of $2nd(0s)$ with $2nd(x; y; z)$ success. Such a matching procedure is called on-demand matching. On-demand matching may reduce subterms for the success of matching. In this section, like the E-strategy, we formalize a user-defined on-demand matching mechanism, called O-matching. First we formalize a matching list of each operation symbol, like a local strategy.

**Definition 3.1:** A map $\varphi' : \Sigma \to List(\mathcal{N}_+)$ is called an *O-matching map* (*O-map*) iff $\varphi'(f) \subseteq \{1, 2, \ldots, ar(f)\}$ for each $f \in \Sigma$. $\varphi'(f)$ is called a matching list of $f$. An O-map $\varphi$ is extended to the map from $V \cup \Sigma$ in which $\varphi(x) = nil$ for each $x \in V$. We call $\varphi'$ *complete* iff for any matching list $\varphi'(f) = [i_0, \ldots, i_n]$, (a) $\{i_0, \ldots, i_n\} = \{1, \ldots, ar(f)\}$ and (b) $i_k = i_{k'}$ implies $k = k'$. □

The completeness means that all arguments should be contained and that there are no duplicated elements. The difference from the E-map is that matching lists do not have zero. In O-matching, a term is matched with the left-hand sides of all rewrite rules according to matching lists in the top-to-bottom order. A given term $t$ to be matched with a rewrite rule $l \to r$ is compared as follows: First the root symbols of $t$ and $l$ are compared. If they are different, then $t$ is reduced. Let $t'$ be the reduced term. Again the root symbols of $t'$ and $l$ are compared. If they are still different, O-matching fails. When the root symbols of $t$ (or $t'$) and $l$ are same, their arguments are compared next. Let $t$ (or $t'$) and $l$ be $f(t_1, t_2)$ and $f(l_1, l_2)$, and let $\varphi'(f) = [2, 1]$. O-matching compares $t_2$ with $l_2$ first, and then compares $t_1$ with $l_1$. Note that for an ordinary (not on-demand) matching, the matching order does not have any effect on the result since it does not change a given term. For on-demand matching, the order is important since a given term may be changed. For example, if the above matching of $t_2$ with $l_2$ fails, O-matching fails without any change of $t_1$. On the other hand, if $\varphi'(f) = [1, 2]$, then $t_1$ may be changed.

We formalize O-matching as a rewrite relation like the E-strategy in Definition 2.2. Like *red* and *eval* in Definition 2.2, we define O-matching with the following operation symbols:

$$\begin{array}{rcl} match_p & : & \mathcal{T} \times \mathcal{P}(\mathcal{T}) \to \mathcal{T} \times \mathcal{P}(\mathcal{T}) \\ eval_p & : & \mathcal{T} \times \mathcal{P}(\mathcal{T}) \times List(\mathcal{N}) \to \mathcal{T} \times \mathcal{P}(\mathcal{T}) \end{array}$$

where $\mathcal{P}(A)$ is the set of all subsets of a set $A$ and $p \in \mathcal{N}_+^*$. We formalize the reduction under $\varphi'$ as a rewrite relation on $\mathcal{T}'$, where $\mathcal{T}' = T(\Sigma', V)$, and $\Sigma'$ is obtained by adding special operation symbols $match_p$, $eval_p$ and list and set constructors to a given $\Sigma$. Assume a reduction function $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ is given where $rd(t)$ is defined for all terms $t$. When $rd(t)$ is not explicitly specified for a term $t$, we define $rd(t)$ as $\{t\}$. Intuitively $rd(t)$ is the set of all terms reduced from $t$. We use a filter function $filter_p^f : \mathcal{P}(\mathcal{T}) \to \mathcal{P}(\mathcal{T})$ defined as $filter_p^f(T) = \{t \in T \mid (p \in O_\Sigma(t) \wedge f = t(p)) \vee \exists p' \le p.(t(p') \in V)\}$ for each $f \in \Sigma \cup V$ and $p \in \mathcal{N}_+^*$. Intuitively, $filter_p^f(T)$ is the set of terms whose symbol at position $p$ is $f$ or which has a variable at position $p' \le p$.

**Definition 3.2:** Let $(\Sigma, R)$ be a TRS, $\varphi'$ an O-map. $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function. Then, $\Rightarrow_{rd}^{\varphi'} \subseteq \mathcal{T}' \times \mathcal{T}'$ [†] is

[†] We may omit *rd* and $\varphi'$.

defined as the smallest set satisfying the following: For any $i \in \mathcal{N}_+$, $p \in \mathcal{N}^*$, $is \in List(\mathcal{N}_+)$, $t, t' \in \mathcal{T}$, $T, T' \in \mathcal{P}(\mathcal{T})$ where $T' \neq \emptyset$,

$$match_p(t,T) \Rightarrow \begin{cases} eval_p(t,T',\varphi'(t(\varepsilon))) & \textbf{if } filter_p^{t(\varepsilon)}(T) = T' \\ eval_p(t',T',\varphi'(t'(\varepsilon))) & \textbf{if } \begin{cases} filter_p^{t(\varepsilon)}(T) = \emptyset \\ \wedge t' \in rd(t) \\ \wedge filter_p^{t'(\varepsilon)}(T) = T' \end{cases} \\ (t',\emptyset) & \textbf{if } \begin{cases} filter_p^{t(\varepsilon)}(T) = \emptyset \\ \wedge t' \in rd(t) \\ \wedge filter_p^{t'(\varepsilon)}(T) = \emptyset \end{cases} \end{cases}$$

$$eval_p(t,T,nil) \Rightarrow (t,T)$$

$$eval_p(t,T,i;is) \Rightarrow \begin{cases} eval_p(t[t']_i,T',is) & \textbf{if } match_{p.i}(t|_i,T) \Rightarrow^* (t',T') \\ (t[t']_i,\emptyset) & \textbf{if } match_{p.i}(t|_i,T) \Rightarrow^* (t',\emptyset) \end{cases}$$

□

Notice that there may be plural pairs $(t', T')$ such that $match_p(t,T) \Rightarrow (t',T')$ since $red(t)$ may not be a singleton. Unlike the function $red$ of the E-strategy, $match_p$ takes and returns a pair of a term and a set of terms. A relation $match_p(t,T) \Rightarrow^* (t',T')$ means that on-demand matching of $t$ with $T|_p$ has been performed[†], and $t$ has been reduced into $t'$ whose subterms are reduced in order to make the matching a success. $T'$ has been obtained by removing terms whose subterms at $p$ have no chance to be matched with $t$. O-matching first checks whether the root symbol $t(\varepsilon)$ of $t$ should be reduced or not to make the matching a success. If there exists a term $s \in T$ such that $s(p) = t(\varepsilon)$ or $s(p')$ is a variable for some $p' \leq p$, i.e., $filter_p^{t(\varepsilon)}(T) \neq \emptyset$, the root symbol $t(\varepsilon)$ is not needed to be reduced. Then, the term and the filtered set are handed to $eval_p$ (the first condition of $match_p(t,T)$ of Definition 3.2). If there are no such terms, an on-demand reduction will be done. For a reduced term $t' \in rd(t)$, the root symbol $t'(\varepsilon)$ is checked again. If $filter_p^{t'(\varepsilon)}(T) \neq \emptyset$, then $eval_p$ is called (the second condition of $match_p(t,T)$). If $filter_p^{t'(\varepsilon)}(T)$ is still empty, then $(t',\emptyset)$ is returned as a matching failure (the last condition of $match_p(t,T)$). $eval_p(f(\vec{t}),T,L)$ checks whether arguments $t_i$ can be matched with $T$ at position $p.i$ for each $i \in L$ (the first condition of $eval_p(t,T,i;is)$). When all elements of the list $L$ are consumed, the term $f(\vec{t'})$ and the set $T'$ of remained terms are returned as a matching success (the case of $eval_p(t,T,nil)$). When a matching fails for some $t_i$, then the whole on-demand matching also fails, and the term whose subterms reduced partway and the empty set are returned (the last condition of $eval_p(t,T,i;is)$). Note that if the reduction function $rd$ is computable, the computation of $match_p(t,T)$ is terminating since the size of the elements of $T$ is finite.

**Example 3.3:** Consider TRS $R_{2nd}$ given at the beginning of this section. Let $\varphi'(0s) = nil$, $\varphi'(2nd) = [1]$, $\varphi'(cons) = [2]$, $\varphi'(0) = nil$. Let $0;0s \in rd(0s)$. We show how to obtain the rewrite sequence: $match_\varepsilon(2nd(0s),\mathcal{L}) \Rightarrow^* (2nd(0;0;0s),\{2nd(x;y;z)\})$ from bottom up.

1. Consider $match_{1.2.2}(0s,\{2nd(x;y;z)\})$. The symbol of $2nd(x;y;z)$ at position 1.2.2 is a variable $z$. Thus, $filter^{0s}_{1.2.2}(\{2nd(x;y;z)\}) = \{2nd(x;y;z)\}$. Since $\varphi'(0s) = nil$,

   $match_{1.2.2}(0s,\{2nd(x;y;z)\})$
   $\Rightarrow eval_{1.2.2}(0s,\{2nd(x;y;z)\},nil)$
   $\Rightarrow (0s,\{2nd(x;y;z)\})$.

2. Consider $match_{1.2}(0s,\{2nd(x;y;z)\})$. Since the symbol $cons$ (= ;) of $2nd(x;\underline{y;z})$ at position 1.2 differs from $0s$, the term $0s$ is reduced into $0;0s \in rd(0s)$. Then, the root symbol of the target term $0;0s$ becomes $cons$, and they are handed to $eval$.

   $match_{1.2}(0s,\{2nd(x;y;z)\})$
   $\Rightarrow eval_{1.2}(0;0s,\{2nd(x;y;z)\},[2])$.

   Then, the second argument of $0;0s$ is checked because of the list [2]. We have already shown that $match_{1.2.2}(0s,\{2nd(x;y;z)\}) \Rightarrow (0s,\{2nd(x;y;z)\})$. Thus,

   $match_{1.2}(0s,\{2nd(x;y;z)\})$
   $\Rightarrow eval_{1.2}(0;\underline{0s},\{2nd(x;y;z)\},[2])$
   $\Rightarrow eval_{1.2}(0;\underline{0s},\{2nd(x;y;z)\},nil)$
   $\Rightarrow (0;0s,\{2nd(x;y;z)\})$

3. Consider $match_1(0s,\{2nd(x;y;z)\})$. Similarly,

   $match_1(0s,\{2nd(x;y;z)\})$
   $\Rightarrow eval_1(0;\underline{0s},\{2nd(x;y;z)\},[2])$
   $\Rightarrow eval_1(0;\underline{0;0s},\{2nd(x;y;z)\},nil)$
   $\Rightarrow (0;0;0s,\{2nd(x;y;z)\})$

4. Finally, we obtain

   $match_\varepsilon(2nd(0s),\mathcal{L})$
   $\Rightarrow eval_\varepsilon(2nd(\underline{0s}),\{2nd(x;y;z)\},[1])$
   $\Rightarrow eval_\varepsilon(2nd(\underline{0;0;0s}),\{2nd(x;y;z)\},nil)$
   $\Rightarrow (2nd(0;0;0s),\{2nd(x;y;z)\})$

We can see that the reduction of $0s$ to $0;0s$ is applied repeatedly until the target term $2nd(0s)$ becomes a redex $2nd(0;0;0s)$. □

## 4. Normality

On-demand matching is expected to have the property that (1) it returns a redex when the matching succeeded, and (2) it returns a term which cannot to be any redex when the matching failed. A term which cannot to be reduced into any redex is called a head normal form (or a root stable form). Let $R$ be a TR. The set of all terms reduced from $t$ is defined as $Reduce(t) = \{t' \in \mathcal{T} \mid t \rightarrow_R^* t'\}$. The set of all *head normal forms* is defined as $HNF(R) = \{t \in \mathcal{T} \mid Reduce(t) \cap Redex(R) = \emptyset\}$. We use $HNF$ instead of $HNF(R)$ if no confusion occurs. The following property holds.

[†] $T|_p = \{t|_p \mid t \in T\}$.

**Lemma 4.1:** ([10]) If $t \in HNF$ and $t \to^* s$ then $s \in HNF$ and $s(\varepsilon) = t(\varepsilon)$.

Even if a reduction function $rd$ returns normal forms and an O-map $\varphi'$ is complete, O-matching may return a term which is neither in $Redex$ nor in $HNF$.

**Counter Example 4.2:** Let TRS

$$R_{b_1} = \begin{cases} \wedge(x,0) & \to & 0 \\ \wedge(0,1) & \to & 0 \\ \wedge(1,1) & \to & 1 \end{cases}$$

$\varphi'(\wedge) = [1,2], \varphi'(1) = \varphi'(0) = nil$ and $rd(\wedge(1,1)) = rd(1) = \{1\}$. O-matching of $\wedge(\wedge(1,1),1)$ with $\mathcal{L}$ does not reduce a subterm $\wedge(1,1)$ even though the reduction makes the matching a success.

$$\begin{aligned} & match_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\}) \\ \Rightarrow\ & eval_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\}, [1,2]) \\ \Rightarrow\ & eval_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(x,0)\}, [2]) \\ \Rightarrow\ & (\wedge(\wedge(1,1),1), \emptyset) \end{aligned}$$

Since all left-hand sides have the same root symbol as the input term $\wedge(\wedge(1,1),1)$, the input term and the set of the left-hand sides are handed to $eval_\varepsilon$ as it is. The second rewrite step removes left-hand sides $\wedge(0,1)$ and $\wedge(1,1)$, since their symbols at position 1 differ from the root symbol $\wedge$ of the subterm $\wedge(1,1)$ of the target term at position 1. In the third rewrite step, the remaining left-hand side $\wedge(x,0)$ is also removed, since 0 differs from 1 at position 2. Then, all left-hand sides are removed and O-matching returns $\wedge(\wedge(1,1),1)$ as the matching failure. However, since $\wedge(\wedge(1,1),1) \to_1 \wedge(1,1) \in Redex$, the returned term is not a head normal form. The reason why the subterm $\wedge(1,1)$ was not reduced is because the left-hand side $\wedge(x,0)$ had been remained as a candidate of applicable rewrite rules in the second rewrite step even though it will be removed in the third rewrite step. When $\varphi'(\wedge) = [2,1]$, we obtain the following rewrite sequences:

$$\begin{aligned} & match_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\}) \\ \Rightarrow\ & eval_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(x,0), \wedge(0,1), \wedge(1,1)\}, [2,1]) \\ \Rightarrow\ & eval_\varepsilon(\wedge(\wedge(1,1),1), \{\wedge(0,1), \wedge(1,1)\}, [1]) \\ \Rightarrow\ & eval_\varepsilon(\wedge(1,1), \{\wedge(1,1)\}, nil) \\ \Rightarrow\ & (\wedge(1,1), \{\wedge(1,1)\}) \end{aligned}$$

O-matching removes $\wedge(x,0)$ in the second rewrite step since the second argument 1 of $\wedge(\wedge(1,1),1)$ differs from 0 of $\wedge(x,0)$. Then, the first argument $\wedge(1,1)$ of $\wedge(\wedge(1,1),1)$ is reduced in the third rewrite step since $filter^{\wedge}_1(\{\wedge(1,1), \wedge(1,0)\})$ is empty. □

As above, the order of matching influences the result of the on-demnad matching. To give a guideline for defining a suitable O-matching list, we give a notion of the $\varphi'$-normality, which is a generalization of the left-normality [19] †.

**Definition 4.3:** Let $\varphi'$ be an O-map. *$\varphi'$-normal* terms are defined recursively as follows: $x \in V$ is $\varphi'$-normal, and $f(\vec{t})$ is $\varphi'$-normal if $\exists i \in \{1,\ldots,n\}.[\varphi'(f) = js@[i]@ks \wedge t_i$ is $\varphi'$-normal $\wedge\ \forall j \in js.(t_j \in T(\Sigma)) \wedge \forall k \in ks.(t_k \in V)]$, where @ is the list concatenation. A TRS $R$ is $\varphi'$-normal iff $l$ is $\varphi'$-normal for each $l \to r \in R$. □

Note that when $\varphi'(f) = [1,2,\ldots,ar(f)]$ for each $f \in \Sigma$, $t$ is $\varphi'$-normal iff $t$ is left-normal.

**Example 4.4:** Let $\varphi'(2nd) = [1]$ and $\varphi'(cons) = [2,1]$. A term $l_1 = 2nd(cons(x, cons(y,z))$ is $\varphi'$-normal, and $R_{2nd}$ is a $\varphi'$-normal TRS. □

The rest of this section, we give a sufficient condition under which O-matching always returns a redex or a head normal form. We show three lemmata in order to prove the main theorems (Theorem 4.8 and 4.9). Proofs of Lemma 4.5 and 4.6 can be found in Appendix A.

**Lemma 4.5:** Let $t, s \in \mathcal{T}$, $T, S \in \mathcal{P}(\mathcal{T})$, $p \in \mathcal{N}^*_+$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function. If $match_p(t,T) \Rightarrow^* (s,S)$, then

**(1)** $\forall l \in S.\forall q \in O_\Sigma(l|_p).(s(q) = l|_p(q))$,
**(2)** $\forall l \in T \setminus S.[p \notin O(l) \vee (\exists q \in O_\Sigma(l|_p).$
$(s(q) \neq l|_p(q) \wedge \forall q' < q.(s(q') = l|_p(q')))]$.

**Lemma 4.6:** Let $t, s \in \mathcal{T}$, $R$ a $\varphi'$-normal and constructor TRS, $T \subseteq \mathcal{L}$, $p \in \mathcal{N}^*_+$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function satisfying $rd(t) \subseteq Reduce(t) \cap HNF$. If $match_p(t,T) \Rightarrow^* (s,\emptyset)$, then for each $l \in T$, either of the following holds:

**(1)** $p \notin O(l)$,
**(2)** $p = \varepsilon$ and $s(\varepsilon) \neq l(\varepsilon)$, or
**(3)** $\exists q \in O_\Sigma(l|_p).(s|_q \in HNF \wedge s(q) \neq l|_p(q) \wedge \forall q' < q.(s(q') = l|_p(q')))$.

**Lemma 4.7:** Let $R$ be a constructor TRS, $t$ a term. If for each $l \in \mathcal{L}$, (1) $t(\varepsilon) \neq l(\varepsilon)$ or (2) there exists $p \in O_\Sigma(l)$ such that $t|_p \in HNF$, $t(p) \neq l(p)$ and $\forall q < p.(t(q) = l(q))$, then $t \in HNF$.

**Proof.** Assume $t$ is not a head normal form. There exists a left-hand side $l \in \mathcal{L}$ and a substitution $\theta \in \mathcal{T}^V$ such that $t \to^*_R l\theta$. Assume $t \to^*_R l\theta$ is minimal, that is, there is no $l' \in \mathcal{L}$ and $\theta' \in \mathcal{T}^V$ such that $t \to^*_R l'\theta' \to^+_R l\theta$. From the minimality, the condition (1) does not holds. From the condition (2), there exists $p \in O_\Sigma(l)$ such that $t|_p \in HNF$, $t(p) \neq l(p)$ and $\forall q < p.(t(q) = l(q))$. From the assumption of a constructor TRS, $\forall q \in O_\Sigma(l) \setminus \{\varepsilon\}.(l(q) \in C)$. Thus, $t|_q \in HNF$ for any $q$ satisfying $\varepsilon < q \leq p$. From the minimality, for any rewrite step $t' \to_{p'} t''$ in $t \to^*_R l\theta$, $p'$ is greater than or parallel to $p$. Therefore, $t|_p \to^*_R l\theta|_p$ holds. From Lemma 4.1, $l\theta|_p \in HNF$ and $t(p) = l\theta(p)$. Since $p \in O_\Sigma(l)$, $l\theta(p) = l(p)$ and it is inconsistent with $t(p) \neq l(p)$. □

**Theorem 4.8:** Let $t, s \in \mathcal{T}$, $S \in \mathcal{P}(\mathcal{T})$, $R$ a left-linear TRS, $p \in \mathcal{N}^*_+$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a

† A term is *left-normal* iff no variable occurs before (or in the left of) any operation symbol in the string expression of the term. $f(g(0,x),y)$ is left-normal, but $f(x,s(y))$ is not.

reduction function. If $match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S)$ and $S \neq \emptyset$, then $s$ is a redex.

**Proof.** From Lemma 4.5 (1) and $S \neq \emptyset$, there exists $l \in S \subseteq \mathcal{L}$ such that $\forall q \in O_\Sigma(l).(s(q) = l(q))$. Since $R$ is left-linear, $l$ is linear. Therefore, $l\theta = s$ for the substitution $\theta$ defined as $\theta(l(p)) = s(p)$ for each $p \in O_V(l)$ and $\theta(x) = x$ for each $x \in V \setminus V(l)$. □

**Theorem 4.9:** Let $t, s \in \mathcal{T}$, $S \in \mathcal{P}(\mathcal{T})$, $R$ a $\varphi'$-normal and constructor TRS, $p \in \mathcal{N}_+^*$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function satisfying $rd(t) \subseteq Reduce(t) \cap HNF$. If $match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, \emptyset)$, then $s$ is a head normal form.

**Proof.** From Lemma 4.6 and 4.7. □

**Corollary 4.10:** Let $t, s \in \mathcal{T}$, $S \in \mathcal{P}(\mathcal{T})$, $R$ a $\varphi'$-normal, left-linear and constructor TRS, $p \in \mathcal{N}_+^*$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function satisfying $rd(t) \subseteq Reduce(t) \cap HNF$. If $match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S)$, then $s$ is a redex or a head normal form. □

The O-map $\varphi'$ in Example 4.4 for TRS $R_{2nd}$ satisfies the assumptions of Theorem 4.8 and 4.9. Thus, for each term $t \in \mathcal{T}$, O-matching returns a redex or a head normal form. Each of the assumptions of the $\varphi'$-normality, the left-linearity and the constructor TRS in Theorem 4.8 and 4.9 is a necessary condition. Counter Example 4.2 shows that the $\varphi'$-normality is needed. We show the necessity of the left-linearity and the constructor TRS.

**Counter Example 4.11:** Let $R_{b_2} = \{eq(x, x) \to 1\}$, $\varphi'(eq) = [1, 2]$, $\varphi'(1) = nil$. Note that $R_{b_2}$ is not left-linear. Although $match_\varepsilon(eq(eq(1, 1), 1), \{eq(x, x)\}) \Rightarrow^* (eq(eq(1, 1), 1), \{eq(x, x)\})$, the returned $eq(eq(1, 1), 1)$ is not a redex. Since the position of the subterm $eq(1, 1)$ corresponds to a variable position of the left-hand side $eq(x, x)$, the subterm has not been reduced in O-matching. Next, consider the following TRS and $\varphi'$:

$$R_{b_3} = \left\{ \begin{array}{rcl} \neg(\neg(1)) & \to & 1 \\ \neg(0) & \to & 1 \\ \neg(1) & \to & 0 \end{array} \right.$$

$\varphi'(\neg) = [1], \varphi'(1) = \varphi'(0) = nil$. Note that $R_{b_3}$ is not a constructor TRS. Let $rd(0) = \{0\}$ ( $\subseteq HNF(R_{b_3})$ ). Although $match(\neg(\neg(0)), \mathcal{L}(R_{b_3})) \Rightarrow^* (\neg(\neg(0)), \emptyset)$, $\neg(\neg(0))$ is not a head normal form: $\neg(\neg(0)) \to_R \neg(1) \in Redex(R_{b_3})$. The reason why the subterm $\neg(0)$ has not been reduced is because the left-hand side $\neg(\neg(1))$ and the term $\neg(\neg(0))$ have a same symbol at position 1. □

Unfortunately, there exists a TRS which is not $\varphi'$-normal for any complete O-map $\varphi'$.

**Counter Example 4.12:** Consider the TRS

$$R_{b_4} = \left\{ \begin{array}{rcl} \wedge(x, 0) & \to & 0 \\ \wedge(0, x) & \to & 0 \\ \wedge(1, 1) & \to & 1 \end{array} \right.$$

For each complete O-map $\varphi'$, either $\varphi'(\wedge) = [1, 2]$ or $\varphi'(\wedge) = [2, 1]$ holds. The left-hand side $\wedge(x, 0)$ is not $\varphi'$-normal if $\varphi'(\wedge) = [1, 2]$. The left-hand side $\wedge(0, x)$ is not $\varphi'$-normal if $\varphi'(\wedge) = [2, 1]$. □

## 5. The E-Strategy with O-Matching

We combine the E-strategy introduced in Sect. 2 and O-matching proposed in Sect. 3, in which O-matching $match_\varepsilon(\_, \mathcal{L})$ is added before the matching part of the original E-strategy, and the reduction function of O-matching is replaced with $red(\_)$. We call it the E-strategy with O-matching. Notice that, unlike Definition 2.2, we use *reduce* instead of *eval* in the reduction part of the following definition since *eval* is confused with $eval_p$ in the definition of the matching part.

**Definition 5.1:** Let $(\Sigma, R)$ be a TRS, $\varphi$ be an E-map and $\varphi'$ an O-map. Then, $\Rightarrow_{(\varphi,\varphi')} \subseteq \mathcal{T}' \times \mathcal{T}'$ [†] is defined as the smallest set satisfying the following: For any $i \in \mathcal{N}_+$, $is \in List(\mathcal{N})$, $p \in \mathcal{N}^*$, $t, t' \in \mathcal{T}$, $T, T' \in \mathcal{P}(\mathcal{T})$ where $T' \neq \emptyset$,

$$red(t) \Rightarrow reduce(t, \varphi(t(\varepsilon)))$$
$$reduce(t, nil) \Rightarrow t$$
$$reduce(t, i; is) \Rightarrow reduce(t[t']_i, is) \text{ if } red(t|_i) \Rightarrow^* t'$$
$$reduce(t, 0; is) \Rightarrow \left\{ \begin{array}{ll} red(t') & \text{if } match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S) \\ & \wedge s \to_\varepsilon t' \\ reduce(s, is) & \text{if } match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S) \\ & \wedge s \notin Redex. \end{array} \right.$$

$$match_p(t, T) \Rightarrow \left\{ \begin{array}{ll} eval_p(t, T', \varphi(t(\varepsilon))) & \text{if } filter_p^{t(\varepsilon)}(T) = T' \\ eval_p(t', T', \varphi(t'(\varepsilon))) & \text{if } \left\{ \begin{array}{l} filter_p^{t(\varepsilon)}(T) = \emptyset \\ \wedge red(t) \Rightarrow^* t' \\ \wedge filter_p^{t'(\varepsilon)}(T) = T' \end{array} \right. \\ (t', \emptyset) & \text{if } \left\{ \begin{array}{l} filter_p^{t(\varepsilon)}(T) = \emptyset \\ \wedge red(t) \Rightarrow^* t' \\ \wedge filter_p^{t'(\varepsilon)}(T) = \emptyset \end{array} \right. \end{array} \right.$$
$$eval_p(t, T, nil) \Rightarrow (t, T)$$
$$eval_p(t, T, i; is) \Rightarrow \left\{ \begin{array}{ll} eval_p(t[t']_i, T', is) & \text{if } match_{p.i}(t|_i, T) \Rightarrow^* (t', T') \\ (t[t']_i, \emptyset) & \text{if } match_{p.i}(t|_i, T) \Rightarrow^* (t', \emptyset) \end{array} \right.$$

□

When $\varphi'(f) = nil$ for each $f \in \mathcal{D}$, the E-strategy with O-matching $\Rightarrow_{(\varphi,\varphi')}$ coincides with the original E-strategy $\Rightarrow^\varphi$ (Definition 2.2) because $s = t$ for $match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S)$ if $t(\varepsilon) \in \mathcal{D}$. If $\varphi(f) = 0; l$ for some $f \in \mathcal{C}$, when reducing $red(t)$ where $t(\varepsilon) = f$, first $red(t) \Rightarrow reduce(t, 0; l)$ and then $match_\varepsilon(t, \mathcal{L})$ is reduced in order to check the condition part. Since $filter_\varepsilon^f(\mathcal{L}) = \emptyset$ for any $f \in \mathcal{C}$, $red(t)$ should be reduced again. To avoid such a loop, we have assumed $\forall f \in \mathcal{C}.(0 \notin \varphi(f))$ in Sect. 2.

**Example 5.2:** Consider Example 3.3 again.

[†] We may write $\Rightarrow$ instead of $\Rightarrow_{(\varphi,\varphi')}$.

$$R_{2nd} = \left\{ \begin{array}{rcl} hd(x;y) & \to & x \\ tl(x;y) & \to & y \\ 0s & \to & 0;0s \\ 2nd(x;y;z) & \to & y \end{array} \right.$$

$\varphi'(0s) = \varphi'(0) = nil$, $\varphi'(2nd) = [1]$, $\varphi'(cons) = [2]$, $\varphi(0s) = \varphi(2nd) = [0]$, $\varphi(cons) = \varphi(0) = nil$. We show $red(2nd(0s)) \Rightarrow_{(\varphi,\varphi')} 0$. The following rewrite sequence is obtained straightforwardly from the definition: $match_\varepsilon(0s, \mathcal{L}) \Rightarrow reduce_\varepsilon(0s, \{0s\}, nil) \Rightarrow (0s, \{0s\})$. From this rewrite sequence, $red(0s) \Rightarrow reduce(0s, [0]) \Rightarrow red(0;0s) \Rightarrow reduce(0;0s, nil) \Rightarrow 0;0s$ holds, where the second rewrite step comes from $0s \to_\varepsilon 0;0s$. Next, we show $red(2nd(0s)) \Rightarrow reduce(2nd(0s), [0]) \Rightarrow red(0) \Rightarrow reduce(0, nil) \Rightarrow 0$. All rewrite steps except the second one are trivial from the definition. The second rewrite step holds if $match_\varepsilon(2nd(0s), \mathcal{L}) \Rightarrow (2nd(0;0;0s), T)$ and $2nd(0;0;0s) \to_\varepsilon 0$. It has been already shown in Example 3.3. □

### 5.1 Correctness

A function $f : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ is called *correct* if $t \to_R^* s$ and $s$ is a normal form whenever $s \in f(t)$. Ordinary reduction strategies (inner-most, outer-most, etc) are correct. On the other hand, the E-strategy is not always correct. For example, for TRS $R = \{a \to b\}$ and $\varphi(a) = nil$, $red(a) \Rightarrow^* a$. One of the most important features of the E-strategy is that rewriting of some arguments can be restricted to avoid an infinite loop of rewriting like $0s \to 0;0s \to 0;0;0s \to \cdots$. Thus, the correctness is too strong for the E-strategy to satisfy since if we restrict an argument then any redex under the restricted argument cannot be rewritten. For a function $S : \mathcal{T} \to \mathcal{P}(\mathcal{T})$, the correctness of $f$ with respect to $S$ is defined as follows: a function $f$ is *$S$-correct* iff $f(t) \subseteq S(t)$ for each $t \in \mathcal{T}$ [2]. We define $NF, HNF : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ as $NF(t) = \{s \mid t \to_R^* s \wedge s \in NF\}$ and $HNF(t) = \{s \mid t \to_R^* s \wedge s \in HNF\}$. The correctness coincides with the $NF$-correctness. Sufficient conditions for the $HNF$-correctness of the E-strategy have been proposed [2], [14], [15]. In this section we give a sufficient condition of the $HNF$-correctness for the E-strategy with O-matching.

**Theorem 5.3:** Let $t, s \in \mathcal{T}$, $p \in \mathcal{N}_+^*$, $\varphi$ an E-map satisfying that $0 \in \varphi(f)$ for each $f \in \mathcal{D}$, $\varphi'$ a complete O-map and $R$ a $\varphi'$-normal, left-linear and constructor TRS. If $red(t) \Rightarrow_{(\varphi,\varphi')}^* s$, then $t \to_R^* s$ and $s$ is a head normal form.

**Proof.** We prove the following properties by induction on the structure of the definition of $\Rightarrow_{(\varphi,\varphi')}$:

(1) If $match_\varepsilon(t, \mathcal{L}) \Rightarrow^* (s, S)$ for some $S \in \mathcal{P}(\mathcal{T})$, then $s \in Reduce(t) \cap (Redex \cup HNF(t))$.
(2) If $red(t) \Rightarrow^* s$ then $s \in Reduce(t) \cap HNF(t)$.

For both (1) and (2), it is trivial that $s \in Reduce(t)$ from the definition. If the reduction function $rd$ is defined as $rd(t) = \{t' \in \mathcal{T} \mid red(t) \Rightarrow^* t'\}$, then from I.H. and Theorem 4.8 and 4.9, (1) holds. The rewrite sequence $red(t) \Rightarrow^* s$ can be decomposed as follows:

$$\begin{array}{rcl} red(t) \Rightarrow^* red(u_0) & \Rightarrow & reduce(u_0, l_0) \\ & \Rightarrow & \cdots \\ & \Rightarrow & reduce(u_{m+1}, l_{m+1}) \\ & \Rightarrow & s \end{array}$$

where $u_{m+1} = s$ and $l_{m+1} = nil$. If $u_0 \in V$, then $l_0 = nil$ and $s = u_0$. Any variable is a head normal form. If $u_0(\varepsilon) \in C$, $u_0 \in HNF$ and $s \in HNF$. If $u_0(\varepsilon) \in \mathcal{D}$, $0 \in l_0$. There exists $k \in \{0, \ldots, m\}$ such that $l_k = 0; l_{k+1}$. The rewrite step $reduce(u_k, l_k) \Rightarrow reduce(u_{k+1}, l_{k+1})$ is obtained by applying the forth rule with the conditions $match_\varepsilon(u_k, \mathcal{L}) \Rightarrow^* (u_{k+1}, U)$ and $u_{k+1} \notin Redex$. From (1), $u_{k+1} \in Redex \cup HNF$. Thus, $u_{k+1} \in HNF$. It is trivial that $s \in Reduce(u_{k+1})$ from the definition. Therefore, $s = u_{m+1} \in HNF$ from Lemma 4.1. □

**Example 5.4:** (1) Consider the TRS $R_{2nd}$ again. Let $\varphi(2nd) = [0]$, $\varphi(0s) = [0]$, $\varphi(cons) = nil$, $\varphi'(2nd) = [1]$, $\varphi'(0s) = nil$, $\varphi'(cons) = [2, 1]$. Then, the TRS $R_{2nd}$, the E-map $\varphi$ and the O-map $\varphi'$ satisfy the assumptions of Theorem 5.3. Thus, it is $HNF$-correct, that is, if $red(t) \Rightarrow_{(\varphi,\varphi')}^* s$, then $s \in HNF(t)$.
(2) Consider TRS $R_{b_1}$ again in Example 4.2. Let $\varphi(\wedge) = [0]$ and $\varphi'(\wedge) = [2, 1]$. They satisfy the assumptions of Theorem 5.3, and it is $HNF$-correct. □

For both of the above examples, $cons(0, 0s)$ and $\wedge(\wedge(1, 1), x)$ may be evaluated terms, which are not normal forms but head normal forms.

## 6. Related Work

For a given function $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ satisfying $rd(t) \subseteq HNF(t)$ for each $t$, we can obtain a normal form of a term $t$ by repeating the following procedure: Apply $rd$ to $t$ and let $t'$ be the rewritten term. Apply $rd$ to the all arguments of $t'$ again and again. Like $red, match_p$, such a function, denoted by $Red_{rd}$, is also formalized as the following rewrite relation:

$$Red_{rd}(t) \Rightarrow \left\{ \begin{array}{lll} x & \texttt{if} & x \in rd(t) \cap V \\ f(\vec{t'}) & \texttt{if} & \left\{ \begin{array}{l} f(\vec{t}) \in rd(t) \wedge Red_{rd}(t_i) \Rightarrow^* t'_i \\ (i = 1, 2, \ldots) \end{array} \right. \end{array} \right.$$

where $n = ar(f)$. Let $\varphi$ be an E-map satisfying $\varphi(f) = [0]$ for each $f \in \mathcal{D}$ and $\varphi(g) = nil$ for each $g \in C$. Let $\varphi'$ be an O-map satisfying $\varphi(f) = [1, 2, \ldots, ar(f)]$ for each $f \in \Sigma$. Let $rd$ be a function $rd(t) = \{s \mid t \Rightarrow_{(\varphi,\varphi')}^* s\}$. Then, $Red_r$ almost coincides with the functional strategy [18], [20]. The functional strategy evaluates terms in top-to-bottom left-to-right lazy pattern matching, which is adopted by several lazy functional languages: Clean, Haskell, Miranda, Lazy ML, etc. We show differences between the functional strategy and the E-strategy with O-matching. Let $\varphi$ and $\varphi'$ be the above maps. The functional strategy assumes the priority of rewrite rules, and the matching is done with each

rewrite rule in the priority order. The E-strategy with O-matching matches a term with all rewrite rules simultaneously. The trigger of on-demand reduction is also different. In the functional strategy, each term whose root symbol is a defined symbol should be reduced in the matching process. In the E-strategy with O-matching, subterms are not reduced until there are no candidates of rewrite rules to be applied even if the root symbol is a defined symbol. Note that both conditions coincide for a constructor TRS. From the above points, the E-strategy with O-matching can be said to be lazier than the functional strategy. For example, when reducing $f(a)$ by $R = \{f(a) \to b, a \to a\}$, the functional strategy reduces the subterm $a$ but the E-strategy with on-demand matching does not so, and thus $f(a) \to_R b$ can be obtained. Moreover, the E-strategy with O-matching has an advantage in the flexibility of the matching and reduction order. When it turned out that an argument $i$ of an operation symbol $f$ can be reduced first by the strictness analysis [14], [18], we may obtain more efficient reduction strategies by setting $\varphi(f) = [i, 0]$.

In the current E-strategy, there is a way to set the reduction of an argument to be lazy with negative integers. For example, $\varphi(cons) = [-1, -2]$ means that "*both arguments are not evaluated until so forced, evaluation is forced, when the argument is involved in matching*" [12]. The difference from the E-strategy with O-matching is that both instructions for reduction and matching are in a single list for each operation symbol, e.g. $\varphi(f) = [-1, 2, 0]$. Since those positive and negative integers have different meanings, we propose an on-demand matching mechanism in which users give a list for matching independent with a list for reduction. The separation of lists makes the mechanism of the strategy more understandable. The understandability is very important for user-defined strategies. There are several operational semantics for the E-strategy with negative integers [1], [15], [17]. An advantage of the E-strategy with O-matching over those operational semantics is that O-matching allows us to control the order of matching while the above existing operation semantics do not.

## 7. Conclusion

We proposed an on-demand matching mechanism, called O-matching, and combined it with the E-strategy in order to obtain a user-defined lazy evaluation strategy. O-matching was designedwto give an operational semantics of algebraic specification languages like CafeOBJ. Let $\varphi$ and $\varphi'$ be an E-map and an O-map. For the $i$-th argument of an operation symbol $f$, we can select the following combination for each purpose: (1) the argument must not be reduced when $i \notin \varphi(f)$ and $i \notin \varphi'(f)$, (2) the argument is reduced on demand when $i \notin \varphi(f)$ and $i \in \varphi'(f)$, and (3) the argument is reduced eagerly when $i \in \varphi(f)$ such that $i$ occurs before 0. Users can choose a suitable one for each operation symbol of each specification (or TRS). Since we divided the existing E-strategy into the reduction part and the matching part, we can combine another reduction function $rd'$ other than the E-strategy with our on-demand matching with keeping Theorem 4.8 and 4.9.

One of the future tasks is to study on termination of the rewrite relation $\Rightarrow_{(\varphi,\varphi')}$. Context-sensitive rewriting is one of the approaches to analyzing the termination of the (positive) E-strategy [7], [8]. Context-sensitive rewriting is a kind of term rewriting in which reduction is restricted to some arguments. Termination problem of the E-strategy is reduced into termination problem of the corresponding context-sensitive rewriting. For the E-strategy with negative integers, on-demand rewriting has been proposed [9], [13] and some termination proving method has been proposed [9]. From those existing studies, it is expected to obtain a termination proving method for the E-strategy with the on-demnad map. Termination of the E-strategy is a part of a sufficient condition for head normalization of the E-strategy. If the E-strategy is terminating and $HNF$-correct, then it is head normalizing, that is, for any term which can be reduced into a head normal form, the E-strategy evaluates it into a head normal form.

## Acknowledgement

This work was supported by KAKENHI 18300008 and 18700024.

### References

[1] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas, "Improving on-demand strategy annotations," Proc. 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02), pp.1–18, Tbilisi, Georgia, LNAI2514, 2003.

[2] M. Alpuente, S. Escobar, and S. Lucas, "Correct and complete (positive) strategy annotations for OBJ," Proc. 4th International Workshop on Rewriting Logic and its Applications, WRLA, Electronic Notes in Theoretical Computer Science, vol.71, pp.70–89, 2004.

[3] R. Diaconescu and K. Futatsugi, "CafeOBJ report," World Scientific, 1998.

[4] S. Eker, "Term rewriting with operation symbol evaluation strategies," Proc. 2nd International Workshop on Rewriting Logic and its Applications, WRLA, Electronic Notes in Theoretical Computer Science, vol.15, pp.1–20, 1998.

[5] K. Futatsugi, J.A. Goguen, J.-P. Jouannaud, and J. Meseguer, "Principles of OBJ2," Proc. 12th ACM Symposium on Principles of Programming Languages, pp.52–66, 1985.

[6] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud, "Introducing OBJ," Software Engineering with OBJ: algebraic specification in action, pp.3–167, Kluwer Academic Publishers, 2000.

[7] S. Lucas, "Context-sensitive computations in functional and functional logic programs," J. Functional and Logic Programming, vol.1998, no.1, pp.1–61, 1998.

[8] S. Lucas, "Context-sensitive rewriting strategies," Technical Report DSIC-II/7/00, 56 pages, Departamento de Sistemas Informaticosy Computacion, UPV, 2000.

[9] S. Lucas, "Termination of on-demand rewriting and termination of OBJ programs," Proc. 3rd International Conference on Principles and Practice of Declarative Programming, PPDP, pp.82–93, 2001.

[10] A. Middeldorp, "Call by need computations to root-stable form," Proc. 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, Paris, pp.94–105, 1997.

[11] T. Nagaya, Reduction strategy for term rewriting system, Ph.D. thesis, Japan Advanced Institute of Science and Technology, 1999.

[12] A. Nakagawa, T. Sawada, and K. Futatsugi, CafeOBJ user's manual –ver.1.4.–, http://www.ldl.jaist.ac.jp/cafeobj/, 1998.
[13] M. Nakamura and K. Futatsugi, "The on-demand context-sensitive rewriting," IEICE Technical Report, SS2000-33, 2001.
[14] M. Nakamura and K. Futatsugi, "Completeness and strictness analysis for the evaluation strategy," Proc. International Workshop on Rewriting in Proof and Computation, RPC, pp.80–89, 2001.
[15] M. Nakamura and K. Ogata, "The evaluation strategy for head normal form with and without on-demand flags," Proc. 3rd International Workshop on Rewriting Logic and its Applications, WRLA, Electronic Notes in Theoretical Computer Science, vol.36, pp.211–227, 2000.
[16] M. Nakamura, Evaluation strategies for term rewriting systems, Ph.D. thesis, School of Information Science, Japan Advanced Institute of Science and Technology, 2002.
[17] T. Ogata and K. Futatsugi, "Operational semantics of rewriting with the on-demand evaluation strategy," Proc. 15th ACM Symposium on Applied Computing, pp.756–763, 2000.
[18] R. Plasmeijer and M. Eekelen, "Functional programming and parallel Graph rewriting," ADDISON-WESLEY, 1993.
[19] Terese, Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol.55, Cambridge University Press, 2003.
[20] Y. Toyama, S. Smetsers, M.C.J.D. van Eekelen, and R. Plasmeijer, "The functional strategy and transitive term rewriting systems," in Term Graph Rewriting — Theory and Practice, pp.117–129, John Wiley & sons, 1993.

## Appendix: Proofs

**Lemma 4.5:** Let $t, s \in \mathcal{T}$, $T, S \in \mathcal{P}(\mathcal{T})$, $p \in \mathcal{N}_+^*$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function. If $match_p(t, T) \Rightarrow^* (s, S)$, then

**(1)** $\forall l \in S.\forall q \in O_\Sigma(l|_p).(s(q) = l|_p(q))$,
**(2)** $\forall l \in T \setminus S.[p \notin O(l) \vee (\exists q \in O_\Sigma(l|_p).$
$(s(q) \neq l|_p(q) \wedge \forall q' < q.(s(q') = l|_p(q')))]$.

**Proof.** **(1)** We prove the claim by induction on the structure of the definition of $\Rightarrow$, which means that we assume that for any application of the rewrite rules $eval_p(t, T, i; is) \Rightarrow eval_p(t[t']_i, T', is)$ **if** $match_{p.i}(t|_i, T) \Rightarrow^* (t', T')$ (Definition 3.2), the claim (1) holds for the instance of the condition part $match_{p.i}(t|_i, T) \Rightarrow^* (t', T')$. If $S = \emptyset$, the claim (1) holds trivially. Assume $S \neq \emptyset$. If $t = x \in V$, $match_p(x, T) \Rightarrow eval_p(x, S, nil) \Rightarrow (x, S)$ and $filter_p^x(T) = S$. Since $x$ is a variable, $\exists p' \leq p.(l(p') \in V)$ from the definition of $filter_p^f$. Thus, $\Sigma(l|_p) = \emptyset$ and (1) holds. Assume $t \notin V$ and $\varphi'(t(\varepsilon)) = [i_1, \ldots, i_n]$. From Definition 3.2 and $S \neq \emptyset$, the rewrite sequence $match_p(t, T) \Rightarrow^* (s, S)$ can be decomposed as

$$\begin{array}{lll} match_p(t,T) & \Rightarrow & eval_p(u_0, U_0, [i_0, i_1, \ldots, i_n]) \\ & \Rightarrow & eval_p(u_1, U_1, [i_1, \ldots, i_n]) \\ & \vdots & \\ & \Rightarrow & eval_p(u_n, U_n, [i_n]) \\ & \Rightarrow & eval_p(u_{n+1}, U_{n+1}, nil) \\ & \Rightarrow & (s, S) \end{array}$$

where $s = u_{n+1}$, $S = U_{n+1}$, $match_{p.i_k}(u_k|_{i_k}, U_k) \Rightarrow^* (u'_k, U_{k+1})$ and $u_{k+1} = u_k[u'_k]_{i_k}$ for each $k \in \{0, \ldots, n\}$. Let $l \in S$ and $q \in O_\Sigma(l|_p)$.

**(a)** Consider the case of $q = \varepsilon$. Since $l \in S \subseteq U_0 = filter_p^{u_0(\varepsilon)}(t, T)$, it holds that $u_0(\varepsilon) = l(p)$ or $\exists p' \leq p.(l(p') \in V)$. Since $\varepsilon \in O_\Sigma(l|_p)$, the latter case does not hold, and $u_0(\varepsilon) = l(p)$. From Definition 3.2, any rewrite step whose left-hand side is $eval_p(t, \ldots)$ does not change the root symbol of $t$. Thus, $s(\varepsilon) = u_0(\varepsilon)$, and therefore $s(\varepsilon) = l(p) = l|_p(\varepsilon)$.
**(b)** Consider the case of $q = i.q'$. Since $\varphi'$ is complete, $q = i_k.q'$ for some $k$. From the I.H. of $match_{p.i_k}(u_k|_{i_k}, U_k) \Rightarrow^* (u'_k, U_{k+1})$, for any $l_k \in U_{k+1}$ and $q_k \in O_\Sigma(l_k|_{p.i_k})$, $u'_k(q_k) = l_k|_{p.i_k}(q_k)$. Since $l \in S \subseteq U_{k+1}$, for any $q_k \in O_\Sigma(l|_{p.i_k})$, $u'_k(q_k) = l|_{p.i_k}(q_k)$. Since $i_k.q' = q \in O_\Sigma(l|_p)$, $q' \in O_\Sigma(l|_{p.i_k})$. Thus, $u'_k(q') = l|_{p.i_k}(q')$. From Lemma A.1, $u'_k$ should not be changed in the remaining rewrite sequence, i.e. $s|_{i_k} = u'_k$. Thus, $s(i_k.q') = s|_{i_k}(q') = u'_k(q') = l|_{p.i_k}(q') = l|_p(i_k.q')$.

**(2)** We also prove the claim by induction on the structure of the definition of $\Rightarrow$. If $T \setminus S = \emptyset$, the claim (2) holds trivially. Assume $T \setminus S \neq \emptyset$ and let $l \in T \setminus S$. If $p \notin O(l)$, the claim (2) holds trivially. Assume $p \in O(l)$. $l|_p$ is not a variable since if $l|_p \in V$ then $l \in S$ from the definition of $filter_p^f$. Thus, $p \in O_\Sigma(l)$. Assume the rewrite sequence $match_p(t, T) \Rightarrow^* (s, S)$ consists of a single rewrite step, i.e. $match_p(t, T) \Rightarrow (s, S)$. From Definition 3.2, $filter_p^{s(\varepsilon)}(T) = S = \emptyset$ and thus $s(\varepsilon) \neq l(p)$. There is no $q' < \varepsilon$. Therefore, (2) holds since $s(\varepsilon) \neq l|_p(\varepsilon) \wedge \forall q' < \varepsilon.(s(q') = l|_p(q'))$. Assume the rewrite sequence $match_p(t, T) \Rightarrow^* (s, S)$ consists of more than one rewrite steps, i.e. for some $m \geq 0$,

$$\begin{array}{lll} match_p(t,T) & \Rightarrow & eval_p(u_0, U_0, l_0) \\ & \Rightarrow & eval_p(u_1, U_1, l_1) \\ & \vdots & \\ & \Rightarrow & eval_p(u_m, U_m, l_m) \\ & \Rightarrow & (s, S). \end{array}$$

Note that $S$ may be empty. There are three cases as follows: **(a)** $l \in T \setminus U_0$, **(b)** $l \in U_k \setminus U_{k+1}$ for some $k \in \{0, \ldots, m-1\}$, or **(c)** $l \in U_m \setminus S$.

**(a)** Let $l \in T \setminus U_0$. The proof is the same with the above case of $match_p(t, T) \Rightarrow (s, S)$.
**(b)** Let $l \in U_k \setminus U_{k+1}$ for some $k \in \{0, \ldots, m-1\}$. Assume $\varphi'(t(\varepsilon)) = [i_0, \ldots, i_n]$ and $l_k = [i_k, \ldots, i_n]$ for each $k \in \{0, \ldots, m\}$. Since $eval_p(u_k, U_k, l_k) \Rightarrow eval_p(u_{k+1}, U_{k+1}, l_{k+1})$, it holds that $match_{p.i_k}(u_k|_{i_k}, U_k) \Rightarrow^* (u'_k, U_{k+1})$, $U_{k+1} \neq \emptyset$, and $u_{k+1} = u_k[u'_k]_{i_k}$. Note that $p.i_k \in O(l)$ since $p \in O_\Sigma(l)$. From the I.H. and $p.i_k \in O(l)$, there exists $q \in O_\Sigma(l|_{p.i_k})$ such that $u'_k(q) \neq l|_{p.i_k}(q) \wedge \forall q' < q.(u'_k(q') = l|_{p.i_k}(q'))$. $q \in O_\Sigma(l|_{p.i_k})$ implies $i_k.q \in O_\Sigma(l|_p)$. Then, there exists $i_k.q \in O_\Sigma(l|_p)$ such that $s(i_k.q) = s|_{i_k}(q) = u'_k(q) \neq l|_{p.i_k}(q) = l|_p(i_k.q)$. Let $q' < i_k.q$. If $q' = \varepsilon$, $s(\varepsilon) = l(p)$ since $l \in U_k \subseteq U_0$. Assume $q' = i_k.q''$. Since $q'' < q$, $u'_k(q'') = l|_{p.i_k}(q'')$. From Lemma A.1, $s|_{i_k} = u'_k$. Thus, $s(i_k.q'') = s|_{i_k}(q'') = u'_k(q'') = l|_{p.i_k}(q'') = l|_p(i_k.q'')$. Therefore, $s(i_k.q) \neq l|_p(i_k.q) \wedge \forall q' < i_k.q.(s(q') = l|_p(q'))$ holds.
**(c)** Let $l \in U_m \setminus S$. Then $U_m \not\subseteq S$, i.e., $S \neq U_m$. Since

$S \neq U_m$, the last rewrite step of the above decomposed rewrite sequence can be written as $eval_p(u_m, U_m, l_m) \Rightarrow (u_m[u'_m]_i, \emptyset)$ where $match_{p.i_m}(u_m|_{i_m}, U_m) \Rightarrow^* (u'_m, \emptyset)$ and $s = u_m[u'_m]_{i_m}$. The proof is the same with (b). □

**Lemma 4.6:** Let $t, s \in \mathcal{T}$, $R$ a $\varphi'$-normal and constructor TRS, $T \subseteq \mathcal{L}$, $p \in \mathcal{N}_+^*$, $\varphi'$ a complete O-map, and $rd : \mathcal{T} \to \mathcal{P}(\mathcal{T})$ a reduction function satisfying $rd(t) \subseteq Reduce(t) \cap HNF$. If $match_p(t, T) \Rightarrow^* (s, \emptyset)$, then for each $l \in T$, either of the following holds:

**(1)** $p \notin O(l)$,
**(2)** $p = \varepsilon$ and $s(\varepsilon) \neq l(\varepsilon)$, or
**(3)** $\exists q \in O_\Sigma(l|_p).(s|_q \in HNF \wedge s(q) \neq l|_p(q) \wedge \forall q' < q.(s(q') = l|_p(q')))$.

**Proof.** We prove the claim by induction on the structure of the definition of $\Rightarrow$. If $T = \emptyset$, the claim holds trivially. Assume $T \neq \emptyset$. Let $l \in T$. If $p \notin O(l)$, the claim (1) holds. Assume $p \in O(l)$. If $p \in O_V(l)$, $l$ should not be removed in any rewrite step. It contradicts the assumption $match_p(t, T) \Rightarrow^* (s, \emptyset)$. Assume $p \in O_\Sigma(l)$. Consider the case of $match_p(t, T) \Rightarrow (s, \emptyset)$ which consists of a single rewrite step. Then, $s \in rd(t) \subseteq HNF$. $s(\varepsilon) \neq l(p)$ holds from the definition of $filter_p^f$ and $p \in O_\Sigma(l)$. Thus, the claim (3) holds with $q = \varepsilon$, i.e., $s|_\varepsilon \in HNF \wedge s(\varepsilon) \neq l|_p(\varepsilon) \wedge \forall q' < \varepsilon.(s(q') = l|_p(q'))$. Consider the case of $match_p(t, T) \Rightarrow^+ (s, \emptyset)$. Assume

$$\begin{aligned} match_p(t, T) &\Rightarrow eval_p(u_0, U_0, l_0) \\ &\Rightarrow eval_p(u_1, U_1, l_1) \\ &\vdots \\ &\Rightarrow eval_p(u_m, U_m, l_m) \\ &\Rightarrow (s, \emptyset). \end{aligned}$$

where $U_k \neq \emptyset$ for each $k \in \{0, \ldots, m\}$. There are three cases as follows: **(a)** $l \in T \setminus U_0$, **(b)** $l \in U_k \setminus U_{k+1}$ for some $k \in \{0, \ldots, m-1\}$, or **(c)** $l \in U_m$.

**(a)** Let $l \in T \setminus U_0$. From the definition of $filter_p^f$, $s(\varepsilon) = u_0(\varepsilon) \neq l(p)$ holds. If $p = \varepsilon$, (2) holds. Assume $p \neq \varepsilon$. There exists $l' \in U_0$ such that $s(\varepsilon) = l'(p) \in \Sigma$ or $\exists p' \leq p.(l'(p') \in V)$. If $\exists p' \leq p.(l'(p') \in V)$, $l'$ should not be removed and it contradicts the assumption. Thus, $s(\varepsilon) = l'(p)$. Since $R$ is a constructor TRS, $T \subseteq \mathcal{L}$ and $p \neq \varepsilon$, $s(\varepsilon) \in C$ and thus $s \in HNF$. Therefore, (3) holds with $q = \varepsilon$.

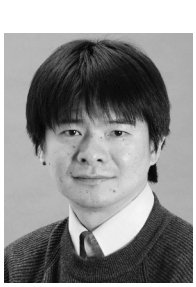
**(b)** Let $l \in U_k \setminus U_{k+1}$ for some $k \in \{0, \ldots, m-1\}$. Assume $\varphi'(t(\varepsilon)) = [i_0, \ldots, i_n]$ and $l_k = [i_k, \ldots, i_n]$. Since $eval_p(u_k, U_k, l_k) \Rightarrow eval_p(u_{k+1}, U_{k+1}, l_{k+1})$, it holds that $match_{p.i_k}(u_k|_{i_k}, U_k) \Rightarrow^* (u'_k, U_{k+1})$ and $u_{k+1} = u_j[u'_k]_{i_k}$. From Lemma 4.5 (2), there exists $q \in O_\Sigma(l|_{p.i_k})$ such that $u'_k(q) \neq l|_{p.i_k}(q) \wedge \forall q' < q.(u'_k(q') = l|_{p.i_k}(q'))$. From Lemma 4.5 (1) and $U_{k+1} \neq \emptyset$, there exists $l' \in U_{k+1}$ such that $\forall q' \in O_\Sigma(l'|_{p.i_k}).(u'_k(q') = l'_{p.i_k}(q'))$. Assume $q \notin O_\Sigma(l'|_{p.i_k})$. Then $l'(p.i_k.q) \in V$. From the $\varphi'$-normality of $l'$, $l'(p.i_{k'}) \in V$ for each $k' > k$. Thus, $l' \in U_{k+1}$ should not be filtered. It contradicts $match_p(t, T) \Rightarrow^* (s, \emptyset)$. Thus, $q \in O_\Sigma(l'|_{p.i_k})$, and $u'_k(q) = l'_{p.i_k}(q)$. Since $l'$ is a constructor term, $u'_k(q) \in C$. Thus, $u'_k|_q \in HNF$. From Lemma A.1, $s|_{i_k} = u'_k$, and thus, $s|_{i_k.q} = u'_k|_q \in HNF$. Therefore, $s|_{i_k.q} = u'_k|_q \in HNF \wedge s(i_k.q) \neq l|_p(i_k.q) \wedge \forall q' < i_k.q.(s(q') = l|_p(q'))$ holds. The proof of $\forall q' < i_k.q.(s(q') = l|_p(q'))$ is similar with the last part of the proof (2) (b) of Lemma 4.5.

**(c)** Let $l \in U_m$. The last rewrite step $eval_p(u_m, U_m, l_m) \Rightarrow (s, \emptyset)$ is obtained by the application of the rewrite rule $eval_p(t, T, i; is) \Rightarrow (t[t']_i, \emptyset)$ **if** $match_{p.i}(t|_i, T) \Rightarrow^* (t', \emptyset)$ of Definition 3.2. Thus, $match_{p.i_m}(u_m|_{i_m}, U_m) \Rightarrow^* (u'_m, \emptyset)$ and $s = u_m[u'_m]_{i_m}$. From the I.H., (1), (2) or (3) holds for $l \in U_m$, $p.i_m \in \mathcal{N}_+^*$ and $u'_m \in \mathcal{T}$. From $p \in O_\Sigma(l)$, the case I.H.(1), $p.i_m \notin O(l)$, does not hold. The case I.H.(2) is also inconsistent with $p.i_m \neq \varepsilon$. Assume I.H.(3) $\exists q \in O_\Sigma(l|_{p.i_m}).(u'_m|_q \in HNF \wedge u'_m(q) \neq l|_{p.i_m}(q) \wedge \forall q' < q.(u'_m(q') = l|_{p.i_m}(q')))$. Then, (3) holds with $i_m.q \in O_\Sigma(l|_p)$. $s|_{i_m.q} = u_m[u'_m]_{i_m}|_{i_m.q} = u'_m|_q \in HNF$ and $s(i_m.q) = u_m[u'_m]_{i_m}(i_m.q) = u'_m(q) \neq l|_{p.i_m}(q) = l|_p(i_m.q)$ hold. $\forall q'' < i_m.q.(s(q'') = l|_p(q''))$ holds as follows: if $q'' = \varepsilon$, then $s(\varepsilon) = l(p) = l|_p(\varepsilon)$. If $q'' = i_m.q'$, then $q' < q$ and $s(i_m.q') = u_m[u'_m]_{i_m}(i_m.q') = u'_m(q') = l|_{p.i_m}(q') = l|_p(i_m.q')$. □

**Lemma A.1:** Let $t, s, u, u' \in \mathcal{T}$, $T \subseteq \mathcal{T}$, and $\varphi'$ a complete O-map. If $match_p(t, T) \Rightarrow^* eval_p(u, U, i; is) \Rightarrow eval_p(u', U', is) \Rightarrow^* (s, S)$, then $u'|_i = s|_i$.

**Proof.** From Definition 3.1, $i_k = i_{k'}$ implies $k = k'$ for any matching list $\varphi'(f) = [i_0, \ldots, i_n]$. Thus, $is$ does not include $i$. From Definition 3.2, $u'|_i$ should not be changed in the rewrite sequence $eval_p(u', U', is) \Rightarrow^* (s, S)$. Thus, $u'|_i = s|_i$. □

**Masaki Nakamura** is an assistant professor at Graduate School of Natural Science and Technology, Kanazawa University. He received his PhD in information science from JAIST (Japan Advanced Institute of Science and Technology) in 2002. His research interest includes software engineering, formal methods, algebraic specification and term rewriting.



**Kazuhiro Ogata** is a research associate professor at Graduate School of Information Science, JAIST (Japan Advanced Institute of Science and Technology). He received his PhD in engineering from Graduate School of Science and Technology, Keio University in 1995. He was a research associate at JAIST from 1995 to 2001, a researcher at SRA Key Technology Laboratory, Inc. from 2001 to 2002, and a research expert at NEC Software Hokuriku, Ltd. from 2002 to 2006. Among his research interests are software engineering, formal methods and formal verification.

**Kokichi Futatsugi** is a professor at Graduate School of Information Science, JAIST (Japan Advanced Institute of Science and Technology). Before getting a full professorship at JAIST in 1993, he was working for ETL (Electrotechnical Lab.) of Japanese Government and was assigned to be Chief Senior Researcher of ETL in 1992. His research interests include formal methods, system verifications, software requirements/specifications, language design, concurrent and cooperative computing. His primary research goal is to design and develop new languages which can open up new application areas, and/or improve the current software technology. His current approach for this goal is CafeOBJ formal specification language. CafeOBJ is multi-paradigm formal specification language which is a modern successor of the most noted algebraic specification language OBJ.