数理解析研究所講究録
918巻 1995年 34-47

Completion for Multiple Reduction Orderings

Masahito Kurihara[1] Hisashi Kondo[2] Azuma Ohuchi[1]

[1] Institute of Systems and Information Engineering
Hokkaido University, Sapporo, 060 Japan
e-mail: {kurihara,ohuchi}@huie.hokudai.ac.jp

[2] Department of Systems Engineering
Ibaraki University, Hitachi, 316 Japan
e-mail: kondo@lily.dse.ibaraki.ac.jp

Abstract

We present a completion procedure (called MKB) which works with multiple reduction orderings. Given equations and a set of reduction orderings, the procedure simulates a computation performed by the parallel processes each of which executes the standard Knuth-Bendix completion procedure (KB) with one of the given orderings. To gain efficiency, however, we develop new inference rules working on objects called nodes, which are data structure consisting of a pair $s : t$ of terms associated with the information to show which processes contain the rule $s \to t$ (or $t \to s$) and which processes contain the equation $s \leftrightarrow t$. The idea is based on the observation that some of the inferences made in the processes are closely related, so we can design inference rules that simulate multiple KB inferences in several processes all in a single operation. Our experiments show that MKB is significantly more efficient than the naive simulation of parallel execution of KB procedures, when the number of reduction orderings is large enough.

1 Introduction

Given equations and a reduction ordering, the Knuth-Bendix completion procedure (KB) [9] tries to compute a complete (convergent) set of rewrite rules. As a result, it may either succeed (with a finite, convergent set of rules), or fail (because of unorientable equations), or loop (in a diverging process trying to generate an infinite set of rules). The practical interest of the completion processes is limited by the possibility of the failure and divergence. The success of the procedure heavily depends on the choice of the reduction ordering. Thus the simplest (but often effective) way of trying to hopefully avoid the failure or divergence is to change the orderings. Actually, in many existing implementations the user can interactively change (or extend) the orderings. Note, however, that this kind of implementation necessarily requires that the users have knowledge of appropriate class of reduction orderings and intuition which is hopefully correct. From the viewpoint of interface with software designers and/or AI researchers who are not familiar with termination proof techniques, automatic change of (or search for) the orderings is desired. However, the problem is that since the completion process can diverge (and we can never decide the divergence in general), it is inappropriate to search for a correct ordering by just sequentially scanning possible orderings. This means that we have to consider, more or less, parallel execution of the completion procedures each working with one of possible orderings. However, naive implementation would result in serious inefficiency.

In this paper, we present a *single* completion procedure (called MKB) which works with *multiple* reduction orderings. Basically, given equations and a set of reduction orderings, the procedure simulates a computation performed by the parallel processes each of which executes KB with one of the given orderings. To gain efficiency, however, we develop new inference rules working on objects called nodes, which are data structure consisting of a pair $s : t$ of terms associated with the information to show which processes contain the rule $s \to t$ (or $t \to s$) and which processes contain the equation $s \leftrightarrow t$. The idea is based on the observation that some of the inferences made in the processes are closely related, so we can design inference rules that simulate multiple KB inferences in several processes all in a single operation. Our experiments show that MKB is significantly more efficient than the naive simulation of parallel execution of KB procedures, when the number of reduction orderings is large enough. In Section 2 we review the standard completion very briefly. Then we present MKB as an inference system in Section 3. A possible MKB completion procedure is presented in Section 4. Section 5 summarizes our work.

2 Standard Completion

We assume that the reader is familiar with the general idea of term rewriting systems. The reader may consult the surveys by Dershowitz and Jouannaud[5], Klop[8], Huet and Oppen[6], Avenhaus and Madlener[1], and Plaisted[11]. In this section we briefly review the standard completion techniques, based on [2, 3, 4].

A set R of rewrite rules is *convergent* (or *complete*) if it is terminating and confluent. The system is *(inter)reduced* if for all $l \to r$ in R, r is irreducible with R and l is irreducible with $R - \{l \to r\}$. A convergent, reduced system is called *canonical*. Let $\succ$ be a reduction ordering (i.e., a well-founded, strict partial ordering on terms such that $s \succ t$ implies $C[s\sigma] \succ C[t\sigma]$ for all contexts $C[\,]$ and substitutions σ). Given a set E of equations and a reduction ordering $\succ$, the standard completion procedure tries to generate a convergent (canonical) set R of rewrite rules which is contained in $\succ$ and which induces the same equational theory as E (if rules of R are regarded as equations). The standard completion is defined in terms of the inference system KB that consists of the following six inference rules:

DELETE:	$(E \cup \{s \leftrightarrow s\}; R\}) \vdash (E; R)$	
COMPOSE:	$(E; R \cup \{s \to t\}) \vdash (E; R \cup \{s \to u\})$	if $t \to_R u$
SIMPLIFY:	$(E \cup \{s \leftrightarrow t\}; R) \vdash (E \cup \{s \leftrightarrow u\}; R)$	if $t \to_R u$
ORIENT:	$(E \cup \{s \leftrightarrow t\}; R) \vdash (E; R \cup \{s \to t\})$	if $s \succ t$
COLLAPSE:	$(E; R \cup \{t \to s\}) \vdash (E \cup \{u \leftrightarrow s\}; R)$	if $l \to r \in R$, $t \to_{\{l \to r\}} u$, and $t \rhd l$
DEDUCE:	$(E; R) \vdash (E \cup \{s \leftrightarrow t\}; R)$	if $s \leftarrow_R u \to_R t$

where $\rhd$ is a well-founded ordering on terms. In this paper, we use as $\rhd$ the *encompassment ordering*: $s \rhd t$ if a subterm of s is an instance of t, but not vice versa.

In practice we assume that the symbol $\cup$ used in the left-hand sides of the inference rules denotes disjoint union. We write $(E; R) \vdash_{\mathrm{KB}\succ} (E'; R')$ if the latter may be obtained from the former by one application of rule in KB. We usually leave $\succ$ implicit and write $\vdash_{\mathrm{KB}}$. Moreover, the subscript KB will be left out, if it is understood.

A *completion procedure* is any program that takes as input a finite set E_0 of equations and a reduction ordering $\succ$, and computes a sequence of inferences from $(E_0; R_0)$ with $R_0 = \emptyset$. The results of a possibly infinite completion sequence $(E_0; R_0) \vdash (E_1; R_1) \vdash \cdots$ are the sets $E_\infty =$

$\bigcup_{i\geq 0}\bigcap_{j\geq i} E_j$ and $R_\infty = \bigcup_{i\geq 0}\bigcap_{j\geq i} R_j$ of *persisting* equations and rules. For a finite sequence ending with $(E_n; R_n)$, we let $(E_\infty; R_\infty) = (E_n; R_n)$. A completion sequence is *successful* if E_∞ is empty and R_∞ is convergent.

The rules in KB are evidently *sound*, in that the class of provable theorems is unchanged by an inference step, i.e., $\leftrightarrow^*_{E\cup R} = \leftrightarrow^*_{E'\cup R'}$ whenever $(E; R) \vdash (E'; R')$. Moreover, $R' \subseteq \succ$ whenever $R \subseteq \succ$. Consequently, the result R_∞ of any successful completion sequence is terminating and presents the same equational theory as E_0.

Since the rule DEDUCE can lead to infinitely long chains of inference, fairness conditions aim to minimize applications of that rule, while ensuring that it is not completely ignored. Let $CP(R)$ denote the set of all critical pairs between rules in R. In particular, $CP(R_\infty)$ denotes the set of all *persistent critical pairs*. A completion sequence $(E_0; R_0) \vdash (E_1; R_1) \vdash \cdots$ in KB is *fair* if (1) all persistent critical pairs are generated ($CP(R_\infty) \subseteq \bigcup_{i\geq 0} E_i$) and (2) no equation persists ($E_\infty = \emptyset$). An n-step KB completion sequence *fails* at step n if no fair sequence has it as a prefix; in that case the completion procedure fails. Assuming the procedure never discriminates against any critical pair or simplifiable rule or equation, the only possible reason for failure is that all equations are unorientable. A completion procedure is *correct* if it generates only fair or failing sequences. The main result in Huet[7] and Bachmair, Dershowitz, and Hsiang[2] is that if a completion sequence is fair, then the limit rewrite system R_∞ is convergent. Therefore, if a correct completion procedure does not fail, then it generates a convergent system in the limit. Moreover, if neither COMPOSE nor COLLAPSE is applicable to $(\emptyset, R_\infty)$, then R_∞ is even canonical.

3 Completion for Multiple Reduction Orderings

3.1 Inference Rules

Let us define a completion procedure MKB for multiple reduction orderings. Let $O = \{\succ_1, \ldots, \succ_n\}$ be a finite set of reduction orderings and $I = \{1, \ldots, n\}$ its index set. (Actually, we may allow O to be a multiset in order to avoid the problem of deciding the identity of orderings.) Given O and an initial set of equations, MKB simulates a computation performed by the parallel processes $\{P_1, \ldots, P_n\}$, where the process P_i executes KB for the reduction ordering $\succ_i$. Thus we may regard I as indices of the parallel processes. However, naive implementation could result in serious inefficiency, because many inferences made by several processes are related so closely (or even essentially the same) that they could cause much waste of computation time. We can be smarter by exploiting this observation to design inference rules that simulate the related KB inferences all in a single operation. The following inference system MKB is based on this idea.

MKB is an inference system that works on objects called *nodes*. A *node* is a tuple $\langle s : t, L_1, L_2, L_3\rangle$, where $s : t$ is an ordered pair of terms s and t, and L_1, L_2, L_3 are subsets of I such that

- L_1, L_2, and L_3 are mutually disjoint,
- if $i \in L_1$ then $s \succ_i t$, and
- if $i \in L_2$ then $t \succ_i s$.

The pair $s : t$ is called a *datum* and L_1, L_2, L_3 are called *labels* of the node. This node is identified with the node $\langle t : s, L_2, L_1, L_3\rangle$. Intuitively, L_1 (L_2) denotes the set of indices of processes (executing the KB) in which the current set of rules, R, contains a rule $s \to t$ ($t \to s$);

L_3 denotes the set of indices of processes in which the current set of equations, E, contains an equation $s \leftrightarrow t$ (or $t \leftrightarrow s$). MKB consists of the following inference rules:

DELETE: $N \cup \{\langle s : s, \emptyset, \emptyset, L\rangle\} \vdash N$ if $L \neq \emptyset$

REWRITE-1: $N \cup \{\langle s : t, L_1, L_2, L_3\rangle\} \vdash N \cup \left\{ \begin{array}{l} \langle s : t, L_1 \backslash L, L_2, L_3 \backslash L\rangle, \\ \langle s : u, L \cap L_1, \emptyset, L \cap L_3\rangle \end{array} \right\}$
if $\langle l : r, L, \ldots, \ldots\rangle \in N$, $t \rightarrow_{\{l \rightarrow r\}} u$, $L \cap (L_1 \cup L_3) \neq \emptyset$, and
either $t \doteq l$ or $L \cap L_2 = \emptyset$

REWRITE-2: $N \cup \{\langle s : t, L_1, L_2, L_3\rangle\} \vdash N \cup \left\{ \begin{array}{l} \langle s : t, L_1 \backslash L, L_2 \backslash L, L_3 \backslash L\rangle, \\ \langle s : u, L \cap L_1, \emptyset, L \cap (L_2 \cup L_3)\rangle \end{array} \right\}$
if $\langle l : r, L, \ldots, \ldots\rangle \in N$, $t \rightarrow_{\{l \rightarrow r\}} u$, $t \rhd l$ and $L \cap L_2 \neq \emptyset$

ORIENT: $N \cup \{\langle s : t, L_1, L_2, L_3 \cup L\rangle\} \vdash N \cup \{\langle s : t, L_1 \cup L, L_2, L_3\rangle\}$
if $L \neq \emptyset$ and $s \succ_i t$ for all $i \in L$

DEDUCE: $N \vdash N \cup \{\langle s : t, \emptyset, \emptyset, L \cap L'\rangle\}$
if $\langle l : r, L, \ldots, \ldots\rangle \in N$, $\langle l' : r', L', \ldots, \ldots\rangle \in N$, $L \cap L' \neq \emptyset$, and
$s \leftarrow_{\{l \rightarrow r\}} u \rightarrow_{\{l' \rightarrow r'\}} t$

where N denotes a finite set of nodes. The equation $t \doteq l$ in the condition of REWRITE-1 denotes that t is an instance of l and vice versa; in other words, t and l are syntactically the same up to renaming variables. Note that with $t \rightarrow_{\{l \rightarrow r\}} u$ and $\rhd$ being the encompassment ordering, the negation of $t \rhd l \wedge L \cap L_2 \neq \emptyset$, appearing in REWRITE-2, is equivalent to $t \doteq l \vee L \cap L_2 = \emptyset$, appearing in REWRITE-1.

DELETE rule of MKB removes a node with a trivial equation $s \leftrightarrow s$. It simulates, in a single operation, applications of DELETE operation (of KB) performed in all processes P_i with $i \in L$.

REWRITE-1 and REWRITE-2 rewrite a term t to u by the rule $l \rightarrow r$. The result is the modification of the labels of the node $\langle s : t, \ldots\rangle$ and the creation of a node $\langle s : u, \ldots\rangle$. REWRITE-1 simulates COMPOSE operation in the processes P_i, $i \in L \cap L_1$, and SIMPLIFY operation in the processes P_i, $i \in L \cap L_3$. All these KB operations in the processes P_i, $i \in L \cap (L_1 \cup L_3)$, are simulated by this single MKB operation. Moreover, REWRITE-2 additionally simulates COLLAPSE operation in the processes P_i, $i \in L \cap L_2$. To see this, consider the following two nodes:

$$\langle l : r, L, \ldots, \ldots\rangle \quad (1)$$

$$\langle s : t, L_1, L_2, L_3\rangle \quad (2)$$

We assume that $t \rightarrow_{\{l \rightarrow r\}} u$. In our interpretation, the current set of rules, R, in every process P_i, $i \in L \cap L_1$, contains both $l \rightarrow r$ and $s \rightarrow t$. Therefore, COMPOSE may be applied on the two rules. As a result, the rule $s \rightarrow t$ is replaced by a new rule $s \rightarrow u$. This could be simulated by the modification of the node (2) to

$$\langle s : t, L_1 \backslash L, L_2, L_3\rangle$$

and the creation of the node

$$\langle s : u, L \cap L_1, \emptyset, \emptyset\rangle.$$

Similarly, in every process P_i, $i \in L \cap L_3$, R contains $l \to r$ and E contains $s \leftrightarrow t$. Therefore, SIMPLIFY operation would result in the replacement of the equation $s \leftrightarrow t$ by a new equation $s \leftrightarrow u$. This could be simulated by the modification of the node (2) to

$$\langle s : t, L_1, L_2, L_3 \backslash L \rangle$$

and the creation of the node

$$\langle s : u, \emptyset, \emptyset, L \cap L_3 \rangle.$$

It follows that the combination of COMPOSE and SIMPLIFY could be simulated by REWRITE-1, which modifies the node (2) to

$$\langle s : t, L_1 \backslash L, L_2, L_3 \backslash L \rangle$$

and creates the node

$$\langle s : u, L \cap L_1, \emptyset, L \cap L_3 \rangle.$$

To make this inference really effective, we naturally require that $L \cup (L_1 \cap L_3) \neq \emptyset$.

If $t \rhd l$ holds and $L \cap L_2$ is not empty, we could combine more. In this case, every process P_i, $i \in L \cap L_2$, contains $l \to r$ and $t \to s$ in R. Application of COLLAPSE would result in the removal of $t \to s$ and the creation of $u \leftrightarrow s$. This could be simulated by the modification of the node (2) to

$$\langle s : t, L_1, L_2 \backslash L, L_3 \rangle$$

and the creation of the node

$$\langle s : u, \emptyset, \emptyset, L \cap L_2 \rangle.$$

It follows that the combination of COMPOSE, SIMPLIFY, and COLLAPSE could be simulated by REWRITE-2. Note that if labels are implemented as bit vectors (in which the ith bit is 1 if and only if i belongs to the label), then their union, intersection, and difference can be computed very quickly. This scheme requires three bits for each index i in order to distinguish $i \in L_1$, $i \in L_2$, $i \in L_3$, and $i \notin L_1 \cup L_2 \cup L_3$. Since exactly one of the four cases can occur, one could also encode this information in two bits at the cost of some extra computation time.

We refer to the nodes $\langle s : t, L_1, \ldots \rangle$, $\langle s : t, L_1 \backslash L, \ldots \rangle$, and $\langle s : u, L \cap L_1, \ldots \rangle$ in these inference rules as *original*, *updated*, and *created* nodes, respectively. In actual implementation, we could directly modify the labels of an original node in order to put the updated node on the same memory location as is occupied by the original node.

ORIENT orients an equation $s \leftrightarrow t$ to $s \to t$ in processes P_i with $s \succ_i t$. This is achieved by suitable modification of labels of the node. In practice we let L to be the maximal subset of the third label of the node such that $s \succ_i t$ for all $i \in L$. The maximal label may be trivially obtained by scanning the indices in the third label one by one to see if $s \succ_i t$, but for some classes of reduction orderings this may be obtained more quickly by other means. For example, if O is a set of recursive path orderings and the equation is $f(x) \to g(x)$, then L is the intersection of the third label and the set of indices corresponding to orderings containing the precedence $f \succ g$. A practically most effective case is when O is a set of simplification orderings and the right-hand side of the rule is homeomorphically embedded in the left-hand side; then L is identical with the third label, because then the left-hand side is greater than the right-hand side in every simplification ordering.

DEDUCE creates a node for equational consequences derived from two rules. Of course, only critical pairs need to be considered. Every process P_i, $i \in L \cap L'$, contains rules $l \to r$ and $l' \to r'$ in R. Therefore, the critical pair $s \leftrightarrow t$ (if it exists) can be deduced in all these processes. The node $\langle s : t, \ldots \rangle$ created here is called a *critical* node.

Note that we have assumed that we never distinguish a node $\langle s : t, L_1, L_2, L_3\rangle$ from $\langle t : s, L_2, L_1, L_3\rangle$. This implies that some inference rules implicitly specify the symmetric cases. For example, ORIENT rule implicitly specify the following case:

$$\text{ORIENT}': \quad N \cup \{\langle s : t, L_1, L_2, L_3 \cup L\rangle\} \vdash N \cup \{\langle s : t, L_1, L_2 \cup L, L_3\rangle\}$$
$$\text{if } L \neq \emptyset \text{ and } t \succ_i s \text{ for all } i \in L.$$

We write $N \vdash_{\text{MKB}} N'$ if the latter may be obtained from the former by one application of rule in MKB. An MKB completion procedure is any program that takes as input a finite set E_0 of equations and a set O of reduction orderings, and computes a sequence of inferences from the initial set of nodes, $N_0 = \{\langle s : t, \emptyset, \emptyset, I\rangle \mid s \leftrightarrow t \in E_0\}$, where $I = \{1, \ldots, |O|\}$.

3.2 Soundness and Completeness

Let us see the relationships between MKB and KB.

DEFINITION **3.1** Let $N = \{\langle s_j : t_j, L_1^j, L_2^j, L_3^j\rangle \mid 1 \leq j \leq m\}$ be a set of nodes and $i \in I$ be an index. The *E-projection* $E[N, i]$ of N onto i is a set of equations defined by

$$E[N, i] = \{s_j \leftrightarrow t_j \mid i \in L_3^j, 1 \leq j \leq m\}.$$

Similarly, The $R-projection$ $R[N, i]$ of N onto i is a set of rules defined by

$$R[N, i] = \{s_j \rightarrow t_j \mid i \in L_1^j, 1 \leq j \leq m\} \cup \{t_j \rightarrow s_j \mid i \in L_2^j, 1 \leq j \leq m\}.$$

In the following proposition, $\vdash_{\text{KB}}^{=}$ denotes the reflexive closure of $\vdash_{\text{KB}}$. In other words, $\vdash_{\text{KB}}^{=}$ means either $\vdash_{\text{KB}}$ or $=$. Moreover, $\vdash_{\text{KB}}$ is an abbreviation for $\vdash_{\text{KB}\succ_i}$. The proposition formally states that MKB actually simulates KB.

PROPOSITION **3.2 (Soundness)** *If $N \vdash_{\text{MKB}} N'$ then for all $i \in I$,*

$$(E[N, i]; R[N, i]) \vdash_{\text{KB}}^{=} (E[N', i]; R[N', i]),$$

where the strict part, $\vdash_{\text{KB}}$, holds for at least one i.

Usefulness of MKB comes from the observation that the strict part, $\vdash_{\text{KB}}$, often holds for many i's. The following proposition shows that MKB is as powerful as KB in its ability of inference.

PROPOSITION **3.3 (Completeness)** *If $(E[N, i]; R[N, i]) \vdash_{\text{KB}} (E'; R')$ then there exists a set N' of nodes such that $E' = E[N', i]$, $R' = R[N', i]$, and $N \vdash_{\text{MKB}} N'$.*

3.3 Optional Rules

It is possible to add the following optional inference rules to MKB. They do not correspond to any inference rules of KB, but can affect efficiency of the completion procedure.

$$\text{REMOVE:} \quad N \cup \{\langle s : t, \emptyset, \emptyset, \emptyset\rangle\} \vdash N$$

$$\text{MERGE:} \quad N \cup \left\{ \begin{array}{l} \langle s : t, L_1, L_2, L_3\rangle, \\ \langle s : t, L_1', L_2', L_3'\rangle \end{array} \right\} \vdash N \cup \{\langle s : t, L_1 \cup L_1', L_2 \cup L_2', L_3 \cup L_3'\rangle\}$$

We will abuse the notation and write $N \vdash_{\mathrm{MKB}} N'$ if N' is obtained from N by one application of rule in either the original MKB rules or the two optional rules. REMOVE removes a node if its projections onto equations and rules are empty for all processes. MERGE merges two nodes into a single one if they have the same datum $s : t$. Note that the optional operations make the size of the current node set smaller, without affecting projections.

PROPOSITION **3.4** *If $N \vdash_{\mathrm{MKB}} N'$ by applying* REMOVE *or* MERGE, *then*

$$(E[N,i]; R[N,i]) = (E[N',i]; R[N',i])$$

for all $i \in I$.

This proposition implies that when the optional inference rules are included in MKB, we have to revise the previous proposition on soundness as follows.

PROPOSITION **3.5 (Soundness)** *If $N \vdash_{\mathrm{MKB}} N'$ then for all $i \in I$,*

$$(E[N,i]; R[N,i]) \vdash_{\mathrm{KB}}^{=} (E[N',i]; R[N',i]),$$

where the strict part, $\vdash_{\mathrm{KB}}$, holds for at least one i if the employed rule is not optional.

On the other hand, the completeness result need not be revised. The optional rules are helpful in saving the memory space. Moreover, MERGE operation can make the reasoning process more efficient, because a single inference on the new node can replace the corresponding two inferences on the two old nodes. This saving can be significant when the number of nodes with the same datum increases rapidly. Note, however, that naive implementation of MERGE can make the program slower, because it requires the search for the same datum in the node database.

3.4 Fairness

Given a completion sequence $N_0 \vdash_{\mathrm{MKB}} N_1 \vdash_{\mathrm{MKB}} \cdots$ in MKB, the set of persisting nodes is defined by $N_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$. For a finite sequence ending with N_n, we let $N_\infty = N_n$. The sequence is *successful* if there exists an index $i \in I$ such that $E[N_\infty, i]$ is empty and $R[N_\infty, i]$ is convergent. The sequence is *fair for $i \in I$* if all persistent critical nodes are generated ($CN[N_\infty, i] \subseteq \bigcup_{i \geq 0} N_i$) and no equation persists ($E[N_\infty, i] = \emptyset$) for process i, where $CN[N, i]$ denotes the set of all critical nodes between nodes $\langle l : r, L_1, L_2, L_3 \rangle \in N$ and $\langle l' : r', L'_1, L'_2, L'_3 \rangle \in N$ with $i \in (L_1 \cup L_2) \cap (L'_1 \cup L'_2)$. The sequence *fails for $i \in I$* if no MKB sequence which is fair for i has it as a prefix. The sequence *fails* if a prefix of it fails for every $i \in I$.

Fairness for i only ensures fair creation of (and selection from) critical pairs in process i. For MKB to be really useful, we need a stronger notion of fairness which ensures that every non-failing process is fair. Then we would have a greater possibility of success. This motivates the following definition.

DEFINITION **3.6** A completion sequence in MKB is *fair* if it satisfies the following conditions:

- It is fair for some $i \in I$.
- If it is infinite, then it is either fair or failing for every $i \in I$.

Let S be a completion sequence $N_0 \vdash_{\mathrm{MKB}} N_1 \vdash_{\mathrm{MKB}} \cdots$ in MKB. By projecting each node set onto equations and rules of process $i \in I$, we have a sequence $(E_0; R_0) \vdash_{\mathrm{KB}}^{=} (E_1; R_1) \vdash_{\mathrm{KB}}^{=} \cdots$, where $E_j = E[N_j, i]$ and $R_j = R[N_j, i]$ for $j = 0, 1, \ldots$. By removing all equivalent steps $(E_j; R_j) = (E_{j+1}; R_{j+1})$ we have a proper completion sequence $(E_{k_0}; R_{k_0}) \vdash_{\mathrm{KB}} (E_{k_1}; R_{k_1}) \vdash_{\mathrm{KB}} \cdots$ in KB, where $0 \leq k_0 < k_1 < \cdots$. We denote this sequence by $KB[S, i]$. Then we can easily verify the following relationships between completion sequences in MKB and in KB.

PROPOSITION **3.7** *Let S be a completion sequence in MKB.*

(1) *If S fails for $i \in I$, then $KB[S, i]$ fails.*

(2) *If S fails, then $KB[S, i]$ fails for all $i \in I$.*

(3) *If S is fair for $i \in I$, then $KB[S, i]$ is fair.*

(4) *If S is fair, then $KB[S, i]$ is fair for some $i \in I$. Moreover, if S is infinite, then every $KB[S, i]$, $i \in I$, is either fair or failing.*

An MKB completion procedure is *correct* if it generates only fair or failing MKB sequences. If a correct MKB completion procedure generates a non-failing MKB completion sequence S, then there exists an index $i \in I$ such that the KB completion sequence $KB[S, i]$ is fair, thus the limit rewrite system $R[N_\infty, i]$ is convergent.

4 Completion Procedure

4.1 Completion Procedure

A possible MKB completion procedure, named *mkb*, is given in Fig. 1. It accepts as input a set E of equations and a set O of reduction orderings, and return as output a convergent set of rewrite rules if it successfully halts. The procedure is based on the open/closed lists algorithm which is well-known in the literature of search techniques for artificial intelligence; the sets NE and NR of nodes play a role of the open and closed lists, respectively. Initially, NR is empty and NE consists of all the initial nodes. The union $NE \cup NR$ of the current set of nodes defines a set N_j of nodes in an MKB completion sequence. Although we have seen the inference rules as working on a set of nodes, we can naturally see them as working on a single node or two nodes as well. More precisely, let us call DELETE, ORIENT, and REMOVE the *single-node operations*, and REWRITE-1, REWRITE-2, DEDUCE, and MERGE the *double-node operations*. Then the former is applied to a single node, while the latter to a pair of nodes. We assert that in the computation of *mkb* every node in NR has been fully considered for application of single-node operations, and that every pair of nodes in NR has been fully considered for double-node operations. This implies that all we have to do is applying single-node operations to nodes in NE and double-node operations on pairs of nodes of which at least one is from NE.

The procedure *success*(NE, NR) checks if the completion process has succeeded. More precisely, it is successful if there exists an index $i \in I$ such that i is not contained in any labels of NE nodes and any L_3 labels of NR nodes. Then $E[NE \cup NR, i]$ is the empty set of equations and $R[NR, i]$ is a convergent set of rules contained in $\succ_i$. We assume that the procedure returns such an index i if it is successful, and returns **false** otherwise. Actually, we need not scan all the nodes every time *success* is invoked; this decision could be made more efficiently, if we introduce integer variables c_i $(i = 1, \ldots, |I|)$ for counting the occurrences of the index i in the labels as above, increasing and/or decreasing them every time the labels are updated.

```
procedure mkb(E, O);
begin
  NE := {⟨s : t, ∅, ∅, I⟩ | s ↔ t ∈ E} where I = {1, ..., |O|};
  NR := ∅;
  while success(NE, NR) = false do
    if NE = ∅ then return(fail)
    else
      n := choose(NE);
      NE := merge(NE\{n}, delete(rewrite({n}, NR)));
      if n ≠ ⟨..., ∅, ∅, ∅⟩ then
        n := orient(n);
        if n ≠ ⟨..., ∅, ∅, ...⟩ then
          NE := merge(NE, delete(rewrite(NR, {n})));
          NE := merge(NE, cp(n, NR))
        end;
        NR := merge(NR, {n})
      end
    end
  end;
  return(R[NR, success(NE, NR)])
end mkb.
```

Figure 1: A completion procedure for multiple reduction orderings

The procedure *choose*(NE) selects a node from NE. We assume that this selection is *fair* in that every node in NE will be eventually selected if *mkb* does not fail. It may be a heuristically good strategy to select a node with the "smallest" datum (in a sense, say, measured by its size). However, do not forget to meet the fairness requirement.

The procedure *merge*(N, N') computes the union of N and N'. If the optional MERGE operation is employed, the operation is applied to a suitable pair of nodes of which at least one is from N'.

The procedure *delete*(N) applies DELETE and optional REMOVE to N and returns the set of the remaining nodes. Note that REMOVE is also implemented implicitly by the second **if** statement.

The procedure *rewrite*(N, N') repeatedly applies REWRITE-1 and REWRITE-2 (zero or more times) to $N \cup N'$, rewriting the data of N by the rules of N', until no more rewriting is possible. It returns the set of nodes "created" in this process. We assume that this process contains mutation operations in which the labels of original nodes of N are directly modified in order to make the updated nodes in the same memory location as the original node. This means that the procedure implicitly executes the assignment such as

$$N := N \backslash \{\text{original nodes}\} \cup \{\text{updated nodes}\}.$$

The procedure *orient*(n) repeatedly applies ORIENT (zero or more times) to the node n until no more application is possible, and returns the resultant node n which is the original object but whose labels may have been modified.

The procedure $cp(n, N)$ applies DEDUCE to $\{n\} \cup N$, and returns a set of all critical nodes between n and a node from $\{n\} \cup N$.

Note that the assertion made in the beginning of this section together with the fair selection by *choose* ensures the correctness of the procedure.

PROPOSITION **4.1 (Correctness)** *Mkb is a correct MKB completion procedure.*

4.2 Example

Let us illustrate the completion procedure by a very simple problem. Consider the equational system consisting of the two equations $(x+y)+z = x+(y+z)$ and $f(x)+f(y) = f(x+y)$. Let $\succ_1$ and $\succ_2$ be the lexicographic path orderings induced by the precedence $+ \succ f$ and $f \succ +$, respectively. It is known [1] that if we use $\succ_1$ in the standard completion then the procedure will loop, generating the infinite canonical system

$$\begin{aligned} R_\infty \;= \{\;& (x+y)+z \to x+(y+z), \\ & f(x)+f(y) \to f(x+y) \;\} \\ \cup \{\;& f^n(x+y)+z \to f^n(x)+(f^n(y)+z) \mid n = 1,2,\ldots\}, \end{aligned}$$

while the use of $\succ_2$ would lead almost immediately to the finite canonical system

$$R_\infty = \{(x+y)+z \to x+(y+z),\ f(x+y) \to f(x)+f(y)\}.$$

Let us apply the *mkb* procedure to this problem with $O = \{\succ_1, \succ_2\}$ and $I = \{1,2\}$. The initial sets of nodes are

$$NE = \begin{cases} \langle (x+y)+z : x+(y+z), \emptyset, \emptyset, \{1,2\}\rangle & (1) \\ \langle f(x)+f(y) : f(x+y), \emptyset, \emptyset, \{1,2\}\rangle & (2) \end{cases}$$

and $NR = \emptyset$. We assume that the node (1) is selected by *choose*. Then by *orient* the node is modified to

$$\langle (x+y)+z : x+(y+z), \{1,2\}, \emptyset, \emptyset\rangle \qquad (1')$$

and a critical node

$$\langle (w+x)+(y+z) : (w+(x+y))+z, \emptyset, \emptyset, \{1,2\}\rangle \qquad (3)$$

between itself is created. After $(1')$ is moved from NE into NR, we have $NE = \{(2), (3)\}$ and $NR = \{(1')\}$.

Suppose the node (2) is selected then. It is oriented to

$$\langle f(x)+f(y) : f(x+y), \{1\}, \{2\}, \emptyset\rangle. \qquad (2')$$

Two critical nodes

$$\langle f(x+y)+z : f(x)+(f(y)+z), \emptyset, \emptyset, \{1\}\rangle \qquad (4)$$

$$\langle f(x+(y+z)) : f(x+y)+f(z), \emptyset, \emptyset, \{2\}\rangle \qquad (5)$$

are created between $(1')$ and $(2')$, and $(2')$ is moved into NR. At this point, we have $NE = \{(3), (4), (5)\}$ and $NR = \{(1'), (2')\}$.

By fairness the nodes (3) and (5) will be eventually selected. To make the story short, let us suppose that they are selected successively from now. Then by *rewrite* both are reduced

to a node with trivial equation and thus DELETEd. All the intermediate nodes (including the original nodes), which contain only empty labels, are REMOVEd.

Now we have $NE = \{(4)\}$ and $NR = \{(1'), (2')\}$. Since the third labels of $(1')$ and $(2')$ are empty and since the index 2 is not contained in any labels of (4), the *success* procedure reports the success of completion by returning 2, and the procedure is finished. This means that the completion has succeeded under the reduction ordering $\succ_2$, yielding the finite canonical set $R[NR, 2]$ of rules described before.

Although this simple example shows almost nothing but *mkb*'s ability of simulating parallel execution of KB, its effect on performance should be clear if we think of the extension of this example with more equations on a larger set of function symbols. For example, let O be a larger set of lexicographic path orderings and I the corresponding set of indices. Nevertheless, by *orient* we can expect to have a node

$$\langle (x + y) + z : x + (y + z), I, \emptyset, \emptyset \rangle \tag{1'}$$

at almost the same cost as before, because $(x + y) + z$ is greater than $x + (y + z)$ in every ordering in O and an appropriate implementation of *orient* could determine and exploit this fact very quickly. Similarly, we would have a node

$$\langle f(x) + f(y) : f(x + y), I_1, I_2, \emptyset \rangle \tag{2'}$$

where I_1 and I_2, both being subsets of I, correspond to the sets of all the orderings (in O) containing $+ \succ f$ and $f \succ +$, respectively. Then by *cp* we would have the following critical nodes:

$$\langle f(x + y) + z : f(x) + (f(y) + z), \emptyset, \emptyset, I_1 \rangle \tag{4}$$

$$\langle f(x + (y + z)) : f(x + y) + f(z), \emptyset, \emptyset, I_2 \rangle \tag{5}$$

This simulates the creation of critical pairs in all processes P_i, $i \in I_1 \cup I_2$, at virtually the same cost as the simple example.

4.3 Experiments

We have implemented the *mkb* procedure in Lisp and made some experiments on sample problems taken from Steinbach and Kühler [12]. *Mkb* was compared with a standard completion procedure *kb* and also with a procedure (named *pkb*) that simply simulates parallel execution of *kb*. *Kb* is implemented in a framework proposed by Lescanne[10] and *pkb* is implemented on the multitasking facility of Lucid Common Lisp. The results show that because of the overhead for node manipulation *mkb* is slightly slower than *pkb* when the number of reduction orderings, n, is relatively small, but when n is large enough, *mkb* is significantly faster than *pkb*.

To show how the efficiency depends on n, we present the results on the following problem (Example 3.14 of [12]).

$$\begin{array}{lcl} s(s(x)) & = & x \\ f(0, y) & = & y \\ f(s(x), y) & = & s(f(x, y)) \\ f(f(g(x, y), 0), 0) & = & g(x, y) \\ g(0, y) & = & y \\ g(s(x), y) & = & f(g(x, y), 0) \\ h(0) & = & s(0) \end{array}$$

The problem is to complete a system of these *equations*. (In [12] the problem is to complete a system of rewrite rules defined by orienting these equations from left to right.) The completion

Table 1: Experimental results

n	1	5	10	20	40	120
kb	1					
pkb	1.1	8.4	17	35	72	99
mkb	1.2	7.1	9.2	13	17	18

(a) Execution time

n	1	5	10	20	40	120
kb	1					
pkb	1	5.8	13.6	24	38	102
mkb	1.4	4.5	4.5	6.6	5.5	4.1

(b) The number of equations/nodes

Table 2: Experimental results for ten problems ($n = 40$)

No.	13	14	19	20	21	22	23	29	30	31
pkb	38	72	70	48	1.6	45	15	48	93	2
mkb	8.4	17	16	8.7	1.2	30	6.1	8.8	8.5	1.8

(a) Execution time

No.	13	14	19	20	21	22	23	29	30	31
pkb	38	38	52	51	8.6	49	24	47	65	24
mkb	3.8	5.5	6.5	12	3.4	62	3.3	4.4	4.7	3.8

(b) The number of equations/nodes

succeeds if we specify as a reduction ordering a recursive path ordering induced by a precedence satisfying $g \succ f \succ s, g \succ 0, h \succ s$. The result is the following canonical system of rewrite rules.

$$\begin{aligned}
s(s(x)) &\to x \\
f(0,y) &\to y \\
f(s(x),y) &\to s(f(x,y)) \\
g(0,y) &\to y \\
g(s(x),y) &\to f(g(x,y),0) \\
h(0) &\to s(0) \\
f(f(x,0),0) &\to x
\end{aligned}$$

The statistics are given in Table 1. We have considered $5! = 120$ recursive path orderings induced by all total precedences on the set $\{f, g, h, s, 0\}$. From them we have randomly selected n $(= 1, 5, 10, 20, 40, 120)$ distinct orderings except that the total ordering $g \succ f \succ h \succ s \succ 0$ has been always included to ensure success. The table (a) shows the average execution time. The table (b) is included for evaluating the consumed memory space. It shows the average number of equations (or nodes for *mkb*) required in the computation, i.e., the number of initial equations (nodes) plus the number of equations (nodes) created in the processes. (For *pkb*, the number of initial equations is n times greater than those for *kb*.) In both tables the entries for *kb* are normalized to 1. We see that *mkb* is less efficient than *pkb* when n is less than about 5, but when n is greater than 10, it is significantly more efficient.

The Table 2 shows the results for other problems. From [12] we selected ten problems which contained five or more function symbols and were solved with recursive path orderings with left-to-right status. (The problem number xx denotes Example 3.xx of [12].) The number of

reduction orderings to consider was fixed to $n = 40$. We see that in most problems (except problem 22) the number of nodes for *mkb* was less than that of equations for *pkb*, and that in all the problems *mkb* was faster than *pkb* on average.

5 Conclusion

We have presented a completion procedure MKB for multiple reduction orderings. Basically, MKB simulates a parallel execution of KB procedures. Formally, MKB is defined in an abstract framework as an inference system which works on a set of nodes consisting of a pair $s : t$ of terms associated with three labels to show which processes contain the rule $s \to t$ (or $t \to s$) and which processes contain the equation $s \leftrightarrow t$. This makes it possible to simulate multiple KB inferences in several processes all in a single operation. We have also discussed the soundness, completeness, and fairness. We have proposed a possible correct implementation and made some experiments to show that MKB is significantly more efficient than the naive simulation of parallel execution of KB procedures, when the number of the reduction orderings is large enough.

Acknowledgment

This work is partially supported by the Grants-in-Aid for Scientific Research, No.04650298, the Education Ministry of Japan; and also by the donation from Toshiba Corporation.

References

[1] Avenhaus J. and Madlener, K., Term rewriting and equational reasoning, Banerji, R.B. ed., *Formal Techniques in Artificial Intelligence: A Sourcebook*, North-Holland, 1-44, 1990.

[2] Bachmair, L., Dershowitz, N., and Hsiang, J., Orderings for equational proofs, *Proc. Symp. on Logic in Computer Science*, 346–357, 1986.

[3] Bachmair, L., *Canonical Equational Proofs*, Birkhäuser, 1991.

[4] Dershowitz, N., Completion and its applications, Aït-Kaci, H. and Nivat, M. eds., *Resolution of Equations in Algebraic Structures*, Vol.2: Rewriting Techniques, Academic Press, 31–85, 1989.

[5] Dershowitz, N. and Jouannaud, J.-P., Rewrite systems, van Leeuwen, J. ed., *Handbook of Theoretical Computer Science*, vol. B, North-Holland, 243–320, 1990.

[6] Huet, G. and Oppen, D. C., Equations and rewrite rules: a survey, Book, R. ed., *Formal Language Theory: Perspectives and Open Problems*, Academic Press, 349–405, 1980.

[7] Huet, G., A complete proof of correctness of the Knuth and Bendix completion algorithm, *J. Comput. Syst. Sci.* 23, 11–21, 1981.

[8] Klop, J.W., Term rewriting systems, Abramsky, S., et al. eds., *Handbook of Logic in Computer Science*, vol.*II*, Oxford Univ. Press, 1–116, 1992.

[9] Knuth, D.E. and Bendix, P.B., Simple word problems in universal algebras, Leech, J. ed., *Computational Problems in Abstract Algebra*, Pargamon Press, 263–297, 1970.

[10] Lescanne, P., Completion procedures as transition rules + control, *Proc. TAPSOFT (vol. 1)*, Lect. Notes in Comput. Sci. 351, 28–41, 1989.

[11] Plaisted, D. A., Equational reasoning and term rewriting systems, Gabbay, D. M. et al. eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 1, Oxford Univ. Press, 274–367, 1993.

[12] Steinbach, J. and Kühler, U., Check your ordering: termination proofs and open problems, SEKI report, SR-90-25 (SFB), 1990.