Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I,
IMECS 2010, March 17 - 19, 2010, Hong Kong

Design and Implementation of Multi-Context Rewriting Induction

Haruhiko Sato and Masahito Kurihara [*]

Abstract—**Inductive theorem proving plays an important role in the field of formal verification of systems. The rewriting induction (RI) is a method for inductive theorem proving proposed by Reddy. In order to obtain successful proofs, it is very important to choose appropriate contexts (such as in which direction each equation should be oriented) when applying RI inference rules. If the choice is not appropriate, the procedure may diverge or the users have to come up with several lemmas to prove together with the main theorem. Therefore we have a good reason to consider parallel execution of several instances of the rewriting induction procedure, each in charge of a distinguished single context in search of a successful proof. In this paper, we propose a new procedure, called *multi-context rewriting induction*, which efficiently simulates parallel execution of rewriting induction procedures in a single process, based on the idea of the *multi-completion procedure*. By the experiments with a well-known problem set, we discuss the effectiveness of the proposed procedure when searching along various contexts for a successful inductive proof.**

Keywords: term rewriting systems, inductive theorem proving, rewriting induction

1 Introduction

An inductive theorem is a proposition which holds on recursively-defined data structures, such as natural numbers or lists, and can be proved using the well-founded induction. In the field of formal verification of information systems, inductive theorem proving plays an important role. In order to automatically prove inductive theorems in equational logic, various methods [5] have been proposed based on the theory of term rewriting systems [4]. Among them is a method called the *rewriting induction* (RI) proposed by Reddy [7], which is a principle generalizing and refining several procedures for proving inductive theorems based on term rewriting. The RI method relies on the termination of the given term rewriting systems representing the axioms, because if we have a terminating term rewriting system (i.e. there exists no infinite rewrite sequence), then we can use the transitive closure of the corresponding rewrite relation of the system as a well-founded order over terms for the basis of induction. However, there exist strategic issues coming from the nondeterminism in constructing proofs, and therefore for guiding this procedure to success, we need to choose appropriate proof steps. There are at least three kinds of strategic issues: (1) in which direction hypothetical equations should be oriented, (2) which (axiomatic or hypothetical) rules should be employed for rewriting, and (3) which variables should be instantiated for induction. In general, it is difficult to choose appropriate strategies leading to success and if we chose an inappropriate one, the inductive theorem prover would easily diverge. In the standard RI procedure, the strategy for (1) is fixed before starting the reasoning steps by specifying a reduction order, which is used to ensure the termination of the axiomatic rewrite system and decide the direction of hypothetical equations. The reduction order should be given by the user as an input. This means that the user needs to decide a most difficult part of the strategy beforehand and this has been making it really hard to fully automate the RI-based inductive theorem proving.

In order to solve this problem, Aoto [2] proposed a variant of RI, called the rewriting induction with termination checker (RIt), which, based on the work of Wehrman, et al. [10], uses an external automated termination checker instead of a specific reduction order. In this method, the users need to provide no reduction orders. Moreover, they can implicitly exploit modern termination proving methods more powerful than the classical, simply parameterized reduction orders (such as recursive path orders and polynomial orders). From the viewpoint of strategy, the use of termination checkers gives us more flexibility in the orientation strategy, because they increase the possibility of success in the orientation and we can decide the direction of the equations dynamically. In order to prove inductive theorems as automatic as possible, we can strengthen this flexibility by trying various strategies in parallel. However, if we physically created and ran a number of parallel processes, such naive parallelization would cause serious inefficiency.

In this paper, we present a new variant of rewriting induction procedures, called multi-context rewriting induction (MRIt), based on the idea of the multi-completion [6, 8, 9]. Our procedure efficiently simulates execution of

[*]Graduate School of Information Science and Technology, Hokkaido University; haru@complex.eng.hokudai.ac.jp, kurihara@ist.hokudai.ac.jp; This work was partially supported by JSPS Grant-in-Aid for Scientific Research (C), No. 19500020.

parallel RIt processes in a single process. By the experiments, we will see that the procedure is actually useful for trying various strategies and contexts in parallel and thus guiding some of the promising processes to success. In particular, we demonstrate that there are inductive theorems which are easily proved by MRIt but were not proved by the standard RI or RIt unless the strategies and contexts were chosen correct or else auxiliary lemmas were discovered and supplied.

2 Rewriting Induction

The rewriting induction (RI), proposed by Reddy [7], is a principle for proving inductive theorems in equational logic. Before describing RI, let us briefly review basic notions. A term is a *basic term* if its root symbol is a defined symbol and its arguments are constructor terms. We denote all basic subterms of t by $\mathcal{B}(t)$. A TRS $\mathcal{R}$ is *quasi-reducible* (also called ground-reducible) if every ground basic term is reducible in $\mathcal{R}$. An equation $s = t$ is an inductive theorem of $\mathcal{R}$ if all its ground instances $s\sigma = t\sigma$ are equational consequences of $\mathcal{R}$ (regarded as a set of equations), i.e., $s\sigma \leftrightarrow^*_{\mathcal{R}} t\sigma$.

RI is represented as an inference system working on a pair of a set of equations $\mathcal{E}$ and a set of rewrite rules $\mathcal{H}$. Intuitively, $\mathcal{E}$ represents conjectures (i.e., theorems and lemmas) to be proved and $\mathcal{H}$ represents inductive hypotheses applicable to $\mathcal{E}$. Fig.1 shows the inference rules of RI proposed in [1].

$$\begin{array}{ll}
\textsc{Delete} & \langle \mathcal{E} \uplus \{s = s\}, \mathcal{H}\rangle \vdash \langle \mathcal{E}, \mathcal{H}\rangle \\
\textsc{Simplify} & \langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash \langle \mathcal{E} \cup \{s' = t\}, \mathcal{H}\rangle \\
 & \text{if } s \rightarrow_{\mathcal{R}\cup\mathcal{H}} s' \\
\textsc{Expand} & \langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash \\
 & \langle \mathcal{E} \cup \mathrm{Expd}_u(s,t), \mathcal{H} \cup \{s \rightarrow t\}\rangle \\
 & \text{if } u \in \mathcal{B}(s) \text{ and } s \succ t
\end{array}$$

Figure 1: Inference rules of RI

In Fig.1, $\mathcal{R}$ denotes a set of rewrite rules representing the axioms, $\succ$ is a reduction order containing $\mathcal{R}$, and Expd denotes the function defined by

$$\begin{aligned}
&\mathrm{Expd}_u(s,t) = \\
&\qquad \{C[r]\sigma = t\sigma \mid s \equiv C[u], l \rightarrow r \in \mathcal{R}, \\
&\qquad\qquad \sigma = mgu(u,l), l : \text{basic}\}.
\end{aligned}$$

The DELETE rule removes the trivial equation. The SIMPLIFY rule reduces an equation using a rule of $\mathcal{R}$ and $\mathcal{H}$. The EXPAND rule is the heart of RI. It expands a conjecture into several new conjectures, storing the expanded one in $\mathcal{H}$ as an inductive hypothesis used later as a rewrite rule. Given a set of equations $\mathcal{E}_0$, a quasi-reducible terminating TRS $\mathcal{R}$, and a reduction order $\succ$ containing $\mathcal{R}$, if we have a derivation sequence $\langle \mathcal{E}_0, \mathcal{H}_0\rangle \vdash_{RI} \langle \mathcal{E}_1, \mathcal{H}_1\rangle \vdash_{RI} \cdots \vdash_{RI} \langle \mathcal{E}_n, \mathcal{H}_n\rangle$ where $\mathcal{H}_0 = \mathcal{E}_n = \emptyset$, then all equations in $\mathcal{E}_0$ are inductive theorems of $\mathcal{R}$. It is known that quasi-reducibility is decidable and there is a simply exponential algorithm for it. The possible derivation depends on the choice of the reduction order $\succ$. It means that its choice is important for the success of inductive theorem proving with the rewriting induction.

In general, it is not straightforward to provide a suitable reduction order and choose appropriate inference rules to be applied in the reasoning steps.

Aoto [2] proposed a variant of the rewriting induction, using an arbitrary termination checker instead of a reduction order. The new system, called RIt, is defined by modifying the EXPAND rule as in Fig. 2. It allows us to use more powerful termination checking techniques. However, the neccessity of approprite choice of the direction of the equation in applying the EXPAND rule still remains, because we can often orient an equation in both directions.

$$\begin{array}{ll}
\textsc{Expand}: & \langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash \\
 & \langle \mathcal{E} \cup \mathrm{Expd}_u(s,t), \mathcal{H} \cup \{s \rightarrow t\}\rangle \\
 & \text{if } u \in \mathcal{B}(s) \text{ and } \mathcal{R} \cup \mathcal{H} \cup \{s \rightarrow t\} \text{ terminates}
\end{array}$$

Figure 2: Expand rule of RIt

3 Multi-Context Rewriting Induction

In this section, we show some examples in which the results of inductive theorem proving with RI are different, depending on the ways of applying inference rules. Then we present a new MKB-like procedure which enables us to follow multiple reasoning paths in parallel.

3.1 Examples of Strategic Issues in RI

In this section, we show examples in which the choice of appropriate contexts is important.

Example 3.1 *Let us consider the following TRS [7].*

$$\mathcal{R} = \left\{ \begin{array}{rcl}
\mathsf{f}(\mathsf{0}) & \rightarrow & \mathsf{0} \\
\mathsf{f}(\mathsf{s}(\mathsf{0})) & \rightarrow & \mathsf{s}(\mathsf{0}) \\
\mathsf{f}(\mathsf{s}(\mathsf{s}(x))) & \rightarrow & \mathsf{f}(\mathsf{s}(x)) + \mathsf{f}(x) \\
\mathsf{g}(\mathsf{0}) & \rightarrow & \langle \mathsf{s}(\mathsf{0}), \mathsf{0}\rangle \\
\mathsf{g}(\mathsf{s}(x)) & \rightarrow & \mathsf{np}(\mathsf{g}(x)) \\
\mathsf{np}(\langle x, y\rangle) & \rightarrow & \langle x + y, x\rangle
\end{array} \right.$$

This example defines Fibonacci numbers in two ways: a naive definition by f and an iterative definition by g. The following equation, which represents the correctness of the iterative definition with respect to the naive definition, is an inductive theorem in R.

$$\mathsf{g}(x) = \langle \mathsf{f}(\mathsf{s}(x)), \mathsf{f}(x)\rangle$$

This conjecture can be proved in RI if we orient it from left to right by choosing as a reduction order an appropriate one such as the lexicographic path order (LPO) over the precedence $\mathsf{g} > \mathsf{f} > \mathsf{np} > \langle\rangle > + > \mathsf{s} > 0$.

Actually, expanding this equation by overlapping its left-hand side with the fourth and fifth rules of R, we get

$$\begin{array}{rcl} \langle \mathsf{s}(0), 0\rangle & = & \langle \mathsf{f}(\mathsf{s}(0)), \mathsf{f}(0)\rangle \\ \mathsf{np}(\mathsf{g}(x)) & = & \langle \mathsf{f}(\mathsf{s}(\mathsf{s}(x))), \mathsf{f}(\mathsf{s}(x))\rangle, \end{array}$$

both of which are simplified to trivial equations and deleted. However, the RI procedure will diverge, if we orient the original conjecture from right to left by choosing the LPO with the precedence $\mathsf{f} > \mathsf{g} > \mathsf{np} > \langle\rangle > + > \mathsf{s} > 0$. In general, it is not a trivial task to provide an appropriate reduction order, particularly when it should be automated. When the ordering is inappropriate, we will often have to supply additional conjectures as lemmas, such as

$$\mathsf{sum}(\mathsf{g}(x)) = \mathsf{f}(\mathsf{s}(x)) + \mathsf{f}(x)$$

in addition to some axioms such as

$$\mathsf{sum}(\langle x, y\rangle) \to x+y$$

in our case. This examples demonstrates that appropriate (and automated) choice of the direction in the orientation can sometimes reduce the burden of lemma discovery imposed on the users.

Example 3.2 *We show another example [5] where we need to choose appropriate orientation of conjectures.*

$$\mathcal{R} = \left\{ \begin{array}{rcl} @([], ys) & \to & ys \\ @(xs, []) & \to & xs \\ @(x : xs, ys) & \to & x : @(xs, ys) \\ \mathsf{iter}([], x) & \to & [] \\ \mathsf{iter}(y : ys, x) & \to & x : \mathsf{iter}(ys, x) \\ \mathsf{dcons}(x, []) & \to & [] \\ \mathsf{dcons}(x, y : ys) & \to & (x : y) : \mathsf{dcons}(x, ys) \\ \mathsf{vm}([]) & \to & [] \\ \mathsf{vm}(x : xs) & \to & (x : xs) : \\ & & \mathsf{dcons}(x, \mathsf{vm}(xs)) \\ \mathsf{itvm}([], z, ys) & \to & ys \\ \mathsf{itvm}(x : xs, z, ys) & \to & \mathsf{itvm}(xs, z, z : ys) \end{array} \right.$$

The function $\mathsf{iter}(ys, x)$ *replaces each element in ys with x. The function* $\mathsf{dcons}(x, ys)$ *replaces each element y in ys with* $x : y$*. The function* $\mathsf{vm}(xs)$ *replaces each element in xs with xs, that is, it is the same as* $\mathsf{iter}(xs, xs)$*.*

The itvm *function is an iterative definition of the* vm *function. In the following conjectures, the first two conjectures represent the correctness of the iterative definition and the last three conjectures are lemmas needed for proving them.*

$$\mathcal{E} = \left\{ \begin{array}{rcl} \mathsf{itvm}(xs, xs, []) & = & \mathsf{iter}(xs, xs) \\ \mathsf{vm}(xs) & = & \mathsf{iter}(xs, xs) \\ \mathsf{dcons}(x, \mathsf{iter}(ys, z)) & = & \mathsf{iter}(ys, x : z) \\ \mathsf{itvm}(xs, z, ys) & = & \mathsf{iter}(xs, z)@ys \\ \mathsf{iter}(xs, y)@(y : zs) & = & y : \mathsf{iter}(xs, y)@zs \end{array} \right.$$

When we want to prove these theorems, we have 20 ways of possible combinations of orientations. However, only one of them, which orients all conjectures from left to right, can lead to successful proofs.

Example 3.3 *We show an example [5] where the reduction strategy plays an important role.*

$$\mathcal{R} = \left\{ \begin{array}{rcl} []@ys & \to & ys \\ (x : xs)@ys & \to & x : (xs@ys) \\ \mathsf{r}([]) & \to & [] \\ \mathsf{r}(x : xs) & \to & \mathsf{r}(xs)@[x] \\ \mathsf{b}([]) & \to & [] \\ \mathsf{b}(x : xs) & \to & \mathsf{b1}(x, xs) : \mathsf{b2}(x, xs) \\ \mathsf{b1}(x, []) & \to & x \\ \mathsf{b1}(x, y : ys) & \to & \mathsf{b1}(y, ys) \\ \mathsf{b2}(x, []) & \to & [] \\ \mathsf{b2}(x, y : ys) & \to & \mathsf{b}(x : \mathsf{b}(\mathsf{b2}(y, ys))) \end{array} \right.$$

In this example, we denote the singleton list $x : []$ *by* $[x]$*. Since both functions* r *and* b *calculate the reverse of the given list, the following two equations are inductive theorems in* $\mathcal{R}$*.*

$$\mathcal{E} = \left\{ \begin{array}{rcl} \mathsf{r}(xs) & = & \mathsf{b}(xs) \\ \mathsf{b}(\mathsf{b}(xs)) & = & xs \end{array} \right.$$

These conjectures are proved in a mutually inductive way. By expanding both equations with left-to-right orientation and applying the simplification as much as possible, we have the following two conjectures

$$\begin{array}{l} \mathsf{b1}(x, xs) : \mathsf{b2}(x, xs) = \mathsf{b}(xs)@[x], \\ \quad \mathsf{b1}(\mathsf{b1}(x, xs), \mathsf{b2}(x, xs)) : \mathsf{b2}(\mathsf{b1}(x, xs), \mathsf{b2}(x, xs)) \\ \hfill = x : xs \end{array}$$

and two hypotheses $\{\mathsf{r}(xs) \to \mathsf{b}(xs), \mathsf{b}(\mathsf{b}(xs)) \to xs\}$*. Expansion of the first conjecture from left to right at* $u \equiv \mathsf{b1}(x, xs)$ *followed by four steps of simplification with* $\mathcal{R}$*-rules yields the equation*

$$\mathsf{b1}(y, ys) : (\mathsf{b}(\mathsf{b}(\mathsf{b2}(y, ys)))@[x]) = (\mathsf{b1}(y, ys) : \mathsf{b2}(y, ys))@[x]$$

and the third hypothesis

$$\mathsf{b1}(x, xs) : \mathsf{b2}(x, xs) \to \mathsf{b}(xs)@[x].$$

Moreover, simplification of this equation with the second hypothesis yields the following equation

$$\mathsf{b1}(y, ys) : (\mathsf{b2}(y, ys)@[x]) = (\underline{\mathsf{b1}(y, ys) : \mathsf{b2}(y, ys)})@[x].$$

At this point, if we reduce the whole term of the right-hand side with the second rule of $\mathcal{R}$, we will succeed in proving the conjecture. However, if we apply the third inductive hypothesis to the underlined part, the procedure will diverge. Some people might think that when they want to apply a rewrite rule in the simplification, it seems that an effective strategy would be to give precedence to the inductive hypotheses (in $\mathcal{H}$) over the axiomatic rules (in $\mathcal{R}$). In our example, however, the strategy failed. Note that $\mathcal{R} \cup \mathcal{H}$ is not confluent in general. Therefore, it is a non-trivial task to choose appropriate simplification rules to apply.

3.2 Branching processes

As we have seen in the previous section, it is important but difficult to choose appropriate contexts for obtaining successful results. Some contexts lead to failure, and others to divergence. Therefore, it makes sense to pursue multiple contexts in parallel. In order to do it efficiently, we adapt the idea of the multi-completion to the rewriting induction. The most basic idea is inherited without difficulty: we can reuse the node structure $\langle s : t, H_1, H_2, E\rangle$ and represent the state of n multiple RI processes $\langle E_1, H_1\rangle, \ldots, \langle E_n, H_n\rangle$ by a set of nodes. Then we define the inference rules which simulate a lot of RI-inferences made in different processes.

The difference from the standard multi-completion procedure is that we cannot decide the number of processes and strategies statically (before running the procedure), while in the multi-completion the number is decided by the size of the given set of reduction orders. In the multi-completion, we were only concerned about the way of orientation and it was simple enough to represent it by a predetermined, single object, i.e., a reduction order. Meanwhile, in the rewriting induction, we also have to deal strategies such as how we simplify a term, as shown in Example 3.3. Compared with the orientation, such strategies are not easily enumerated beforehand.

For this reason, we do not fix the number of processes in the new procedure, and allow it to dynamically change. When a process encounters n nondeterministic choices, we will have it *fork* into n different processes, with each process associated with one of the choices. Stated in terms of the tree-search algorithms, each process explores one of the possible n branches. To distinguish such processes, we represent the identifier of each process (called *index*) as a sequence of natural numbers $a_1a_2\ldots a_k$, which can be interpreted as a position in a tree. If the process with the index $p = a_1a_2\ldots a_k$ have n possible choices of contexts, we have it fork into n processes: $a_1a_2\ldots a_k1, a_1a_2\ldots a_k2, \ldots, a_1a_2\ldots a_kn$. Based on the label representation, we can simulate the fork operation by replacing p with the set of n identifiers $p1, \ldots, pn$ in all labels of all nodes.

For the purpose of formal treatment, we introduce the *fork function*. Let us define the set I of all indexes as the set N^* of sequences of natural numbers. We do not distinguish between a process and its index.

Definition 3.4 Fork function $\psi : I \to N$ *maps each process index to a natural number which represents the number of processes to be created from the given process by the fork operation. The* fork function over *a given set P of processes, denoted by $\psi_P : I \to \mathcal{P}(I)$, is defined as follows:*

$$\psi_P(p) = \begin{cases} \{p.1, p.2, \ldots, p.\psi(p)\} & \text{if } p \in P \\ \{p\} & \text{otherwise} \end{cases}$$

where $\mathcal{P}(I)$ denotes the powerset of I. This function will be used to fork all processes in P, while remaining other processes untouched. The domain of the function is lifted to labels, nodes, and sets of nodes as follows:

$$\psi_P(L) = \bigcup_{p \in L} \psi_P(p)$$

$$\psi_P(\langle s : t, H_1, H_2, E\rangle) = \langle s : t, \psi_P(H_1), \psi_P(H_2), \psi_P(E)\rangle$$

$$\psi_P(N) = \{\psi_P(n) \mid n \in N\}$$

3.3 Multi-context rewriting induction

In this section, we present a new procedure, the multi-context rewriting induction procedure with termination checkers (MRIt), which simulates execution of multiple RIt processes based on the framework of MKB. Like MKB, MRIt is represented by an inference system working on a set of nodes. A node is a 4-tuple $\langle s : t, H_1, H_2, E\rangle$ consisting of an ordered pair of terms $s : t$, three sets of indexes of processes H_1, H_2, E, where each index is a sequence of natural numbers. Note that the set of possible indexes I is infinite in MRIt, while it was finite in MKB because the number of processes was fixed beforehand, given the number of reduction orders. In MRIt, the number of running processes is not fixed: the procedure starts with one (root) process and in the course of the execution, adds new processes created by forking existing processes as necessary, when we have nondeterministic choices in applying inference rules. Intuitively, E represents all processes containing $s = t$ as a conjecture to be proved, and $H_1(H_2)$ represents all processes containing $s \to t (t \to s)$ as an inductive hypothesis.

Definition 3.5 ($\mathcal{E}$- and $\mathcal{H}$-projections) *Let $n = \langle s : t, H_1, H_2, E\rangle$ be a node, and i be an index. The $\mathcal{E}$- and $\mathcal{H}$-projections of n onto i are defined as follows:*

$$\mathcal{E}[n, i] = \begin{cases} \{s = t\}, & \text{if } i \in E, \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$\mathcal{H}[n,i] = \begin{cases} \{s \to t\}, & \textit{if } i \in H_1, \\ \{t \to s\}, & \textit{if } i \in H_2, \\ \emptyset, & \textit{otherwise.} \end{cases}$$

The definitions are extended for a set N of nodes as follows:

$$\mathcal{E}[N,i] = \bigcup_{n \in N} \mathcal{E}[n,i], \ \mathcal{H}[N,i] = \bigcup_{n \in N} \mathcal{H}[n,i]$$

$\mathcal{E}[N,p]$ is interpreted as a set of conjectures the process p holds in the state represented by N. Similarly, $\mathcal{H}[N,p]$ is interpreted as a set of inductive hypotheses held in the process p.

Based on the intended interpretation described above, we have developed inference rules of MRIt as shown in Fig. 3. In the inference rules, $sub(N,L) = \{\langle s : t, H_1 \setminus L, H_2 \setminus L, E \setminus L\rangle \mid \langle s : t, H_1, H_2, E\rangle \in N\}$.

The role of each inference rule is as follows. DELETE simulates its counterpart of RI. GC, SUBSUME, and SUBSUME-P are optional rules for efficiency and the first two play the same role as in MKB. The third optional rule stops redundant processes, which have the same state as other existing processes. SIMPLIFY-R and SIMPLIFY-H simulate the SIMPLIFY rule of RI. The difference is that SIMPLIFY-R applies a rule of $\mathcal{R}$, which is common among all processes, while SIMPLIFY-H applies an inductive hypothesis of $\mathcal{H}$, which may exist only in some distinguished processes. FORK, newly introduced in this system, enables us to produce new copies of existing processes to make nondeterministic choices in parallel. In the next section, we will show some strategies to exploit this rule in connection with other inference rules involving nondeterminism.

Let N and N' be two sets of nodes. We write $N \vdash N'$ if the latter is obtained from the former by one application of an inference rule of MRIt. Given a set $\mathcal{E}_0$ of equations and a quasi-reducible terminating TRS $\mathcal{R}$, MRIt starts from the initial set of nodes $N_0 = \{\langle s : t, \emptyset, \emptyset, \{\epsilon\}\rangle \mid s = t \in \mathcal{E}_0\}$, since we want to start with the single (root) simulated process denoted by the empty sequence ϵ. MRIt generates a sequence $N_0 \vdash N_1 \vdash \cdots$.

Finally, we state the soundness of MRIt. The following proposition claims the soundness of FORK, that is, the fork function itself has virtually no effect on the semantics of our procedure, as it only generates copies of existing processes.

Proposition 3.6 *Let N and N' be two sets of nodes and P be a set of indexes such that $N' = \psi_P(N)$. If $p \in I$ and $q \in \psi_P(p)$, then $\langle \mathcal{E}[N,p], \mathcal{H}[N,p]\rangle = \langle \mathcal{E}[N',q], \mathcal{H}[N',q]\rangle$.*

The following proposition states that other inference rules either simulate RI rules (in the strict, $\vdash_{RI}$ part) or have no effect (the reflexive, = part).

Proposition 3.7 *If N' is obtained from N by applying inference rules in MRIt other than* FORK, *then $(\mathcal{E}[N,p], \mathcal{H}[N,p]) \vdash^{=}_{RI} (\mathcal{E}[N',p], \mathcal{H}[N',p])$ for all $p \in I$.*

$$\begin{array}{ll}
\textsc{Delete}: & N \cup \{\langle s : s, H_1, H_2, E\rangle\} \vdash N \\
\textsc{Expand}: & N \cup \{\langle s : t, H_1, H_2, E \uplus E', \rangle\} \vdash \\
& N \cup \{\langle s : t, H_1 \cup E', H_2, E\rangle\} \cup \\
& \{\langle s' : t, \emptyset, \emptyset, E'\rangle \mid s' = t \in \mathrm{Expd}_u(s,t)\} \\
& \text{if } E' \neq \emptyset, u \in \mathcal{B}(s) \text{ and } \mathcal{H}[N,i] \cup \mathcal{R} \cup \\
& \ \{s \to t\} \text{ terminates for all } i \in E' \\
\textsc{Simplify-R}: & N \cup \{\langle s : t, H_1, H_2, E\rangle\} \vdash \\
& N \cup \left\{ \begin{array}{l} \langle s : t, H_1, H_2, \emptyset\rangle \\ \langle s' : t, \emptyset, \emptyset, E\rangle \end{array} \right\} \\
& \text{if } E \neq \emptyset \text{ and } s \to_{\mathcal{R}} s' \\
\textsc{Simplify-H}: & N \cup \{\langle s : t, H_1, H_2, E\rangle\} \vdash \\
& N \cup \left\{ \begin{array}{l} \langle s : t, H_1, H_2, E \setminus H\rangle \\ \langle s' : t, \emptyset, \emptyset, E \cap H\rangle \end{array} \right\} \\
& \text{if } E \cap H \neq \emptyset, \langle l : r, H, \ldots, \ldots\rangle \in N, \\
& \ \text{and } s \to_{\{l \to r\}} s', \\
\textsc{Fork}: & N \vdash \psi_P(N) \\
\textsc{Gc}: & N \cup \{\langle s : t, \emptyset, \emptyset, \emptyset\rangle\} \vdash N \\
\textsc{Subsume}: & N \cup \left\{ \begin{array}{l} \langle s : t, H_0, H_1, E\rangle, \\ \langle s' : t', H'_0, H'_1, E'\rangle \end{array} \right\} \vdash \\
& N \cup \{\langle s : t, H_0 \cup H'_0, H_1 \cup H'_1, E''\rangle\} \\
& \text{if } s : t \text{ and } s' : t' \text{ are variants and} \\
& \ E'' = (E \setminus (H'_0 \cup H'_1)) \cup (E' \setminus (H_0 \cup H_1)) \\
\textsc{Subsume-p}: & N \vdash sub(N', L) \\
& \text{if } \forall p \in L,\ \exists p' \notin L : \\
& \ (\mathcal{E}[N,p], \mathcal{H}[N,p]) = (\mathcal{E}[N,p'], \mathcal{H}[N,p'])
\end{array}$$

Figure 3: Inference rules of MRIt

4 Experiments

In this section, we report some experimental results. In the implementation of MRIt, we used a built-in termination checker (developed by ourselves) based on the dependency-pair method [3]. Moreover, in order to find reduction orders for ensuring termination, we used the combination of polynomial interpretation and SAT solving. All experiments were performed on a workstation equipped with Intel Xeon 2.13GHz CPU and 1GB system memory.

4.1 Effectiveness in trying various strategies

In order to discuss the effectiveness of MRIt in choosing appropriate inference steps for obtaining successful proofs, we experimented with examples discussed in Sect. 3.1. The results are shown in Table 1. The "Total" column shows the total time (in seconds), the "Term" column shows the time consumed by the termination checking in the EXPAND rule, and the "Simpilfy" column shows the time consumed by the SIMPLIFY-H and SIMPLIFY-R rules. The "# of proc" column shows the number of processes which existed when one of the processes succeeded in proving all conjectures.

Table 1: Computation time for examples in Sect. 3.1

Problem	Total	Term	Simplify	# of proc.
Ex. 3.1	0.009	0.002	0.005	3
Ex. 3.2	0.409	0.255	0.080	20
Ex. 3.3	0.262	0.130	0.097	10

In all examples, only one process was in a successful state when the whole system stopped. Moreover, when we continued to run the remaining processes not yet in a successful state, almost no processes (precisely no process in the case of Example 3.2) succeeded because of the divergence. In example 3.3, the number of processes kept increasing (by forking) and among them, less than 20% were those which eventually led to success, and the others seemed to be diverging. From the results, we can see that MRIt is effective in choosing appropriate contexts for obtaining successful proofs.

4.2 Efficiency

We experimented with Dream Corpus examples[1], which are standard examples for inductive theorem proving. In those examples, there were 69 unconditional equational problems suitable for the input to our system. Among them, 35 problems were successfully solved by our system. The results are shown in Table 2, where we only show the results which required more than 0.01sec computation time. The bottommost row indicates the total time of all the 35 problems solved, including those with less than 0.01sec CPU time.

Table 2: Computation time of Dream Corpus examples

Problem	Total	Term	Simplify	# of proc.
109	0.347	0.281	0.043	6
301	0.305	0.244	0.044	6
115	0.042	0.026	0.010	1
1018	0.028	0.014	0.009	3
216	0.014	0.003	0.010	2
total	0.876	0.627	0.169	

We can see that our system can solve those example problems in practical time. They required a relatively small number of contexts, but it was nice for us to be able to run the system efficiently enough without any care of reduction orders and sophisticated strategies.

[1]The examples are available at: `http://kussharo.complex.eng.hokudai.ac.jp/~haru/mrit/` (modified from the originals created by the Mathematical Reasoning Group, University of Edinburgh)

5 Conclusion

In this paper, we have presented MRIt, the multi-context rewriting induction procedure, which simulates parallel execution of rewriting induction procedures. We have reported that MRIt was effective in trying various strategies for obtaining successful proofs efficiently. As future work, we are planning to study extensions for handling non-orientable equations.

References

[1] T. Aoto, "Dealing with Non-orientable Equations in Rewriting Induction," Proc. 17th International Conference on Rewriting Techniques and Applications, vol. 4098 of Lecture Notes in Computer Science, pp.242–256, 2006.

[2] T. Aoto, "Rewriting induction using termination checker," JSSST 24th Annual Conference, 3C-3, 2007 (in Japanese).

[3] T. Arts and J. Giesl, "Termination of term rewriting using dependency pairs," Theoretical Computer Science, vol.236, pp.133–178, 2000.

[4] F. Baader and T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.

[5] G. Huet and J.-M. Hullot, "Proofs by induction in equational theories with constructor," Journal of Computer and System Sciences, vol.25, no.2, pp.239–266, 1982.

[6] M. Kurihara and H. Kondo, "Completion for multiple reduction orderings," Journal of Automated Reasoning, vol.23, no.1, pp.25–42, 1999.

[7] U. Reddy, "Term rewriting induction," 10th Int. Conf. on Automated Deduction, vol.814 of Lecture Notes in Computer Science, pp.162–177, 1990.

[8] H. Sato, S. Winkler, M. Kurihara, and A. Middeldorp, "Multi-completion with termination tools (system description)," Proc. 4th International Joint Conference on Automated Reasoning, vol.5195 of Lecture Notes in Artificial Intelligence, pp.306–312, 2008.

[9] H. Sato, M. Kurihara, S. Winkler, and A. Middeldorp, "Constraint-based multi-completion procedures for term rewriting systems," IEICE Transactions on Information and Systems, vol.E92-D, pp.220–234, 2009.

[10] I. Wehrman, A. Stump, and E. Westbrook, "SLOTHROP: Knuth-Bendix completion with a modern termination checker," Proc. 17th International Conference on Rewriting Techniques and Applications, vol.4098 of Lecture Notes in Computer Science, pp.287–296, 2006.