PAPER *Special Section on Formal Approach*

Multi-Context Rewriting Induction with Termination Checkers

Haruhiko SATO†a) ***and*** **Masahito KURIHARA**†b)**,** ***Members***

SUMMARY 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.

key words: *equational theorem proving, term rewriting systems, mathematical induction, rewriting induction, multi-completion*

1. Introduction

An inductive theorem is an equation over terms which holds on recursively-defined data structures, such as natural numbers and lists. 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 have been proposed based on the theory of term rewriting systems [10]–[12]. Among them is a method called the *rewriting induction* (RI) proposed by Reddy [17], 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) which reduction order should be employed, (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, then 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. [21], 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). We should say that RIt, which solves the strategic issue shown above as (1), has another issue instead: (1') in which direction hypothetical equations should be oriented. 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 [15], [18], [19]. Our procedure efficiently simulates execution of 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

Manuscript received July 17, 2009.
Manuscript revised November 10, 2009.

†The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.
a) E-mail: haru@complex.eng.hokudai.ac.jp
b) E-mail: kurihara@ist.hokudai.ac.jp

DOI: 10.1587/transinf.E93.D.942

chosen correct or else auxiliary lemmas were discovered and supplied.

The paper is organized as follows. We review the rewriting induction and the multi-completion in Sect. 2. In Sect. 3, we discuss some strategic issues in RIt and present the new procedure MRIt. In Sect. 4, we report the results of the experiments and discuss the effectiveness of the new procedure. Section 5 contains the conclusion and possible future work.

2. Preliminaries

2.1 Term Rewriting Systems

We briefly review basic notions for term rewriting systems [4], [6], [13], [20]. A *signature* Σ is a set of function symbols, where each $f \in \Sigma$ is associated with a non-negative integer n, the arity of f. Let V be a set of variables such that $\Sigma \cap V = \emptyset$. The set $T(\Sigma, V)$ of all Σ-terms over V is inductively defined as follows: $V \subseteq T(\Sigma, V)$ and if $t_1, \ldots, t_n \in T(\Sigma, V)$ and $f \in \Sigma$, then $f(t_1, \ldots, t_n) \in T(\Sigma, V)$, where n is the arity of f. We write $s \equiv t$ when the terms s and t are identical. A term s is a *subterm* of t, if either $s \equiv t$ or $t \equiv f(t_1, \ldots, t_n)$ and s is a subterm of some t_i. We denote the set of all variables contained in a term s by $V(s)$, and $V(s) \cup V(t)$ is denoted by $V(s,t)$. A *substitution* is a function $\sigma : V \to T(\Sigma, V)$ such that $\sigma(x) \neq x$ for only finitely many xs. Any substitution σ can be extended to a mapping $\sigma : T(\Sigma, V) \to T(\Sigma, V)$ by defining $\sigma(f(s_1, \ldots, s_n)) = f(\sigma(s_1), \ldots, \sigma(s_n))$. Application $\sigma(s)$ of σ to s is also written as $s\sigma$. A term t is an *instance* of a term s if there exists a substitution σ such that $s\sigma \equiv t$. Two terms s and t are *variants* of each other and written as $s \doteq t$, if s is an instance of t and vice versa: i.e., s and t are syntactically the same up to renaming variables. An *encompassment order* $\sqsupset$ on a set of terms is defined by $s \sqsupset l$ iff some subterm of s is an instance of l and $s \not\doteq l$. A term is a *ground term* if it contains no variables. A term t is a *ground instance* of s if t is a ground term and is an instance of s. The *composition* $\sigma\tau$ of two substitutions σ and τ is defined as $s(\sigma\tau) = (s\sigma)\tau$. A substitution σ is *more general* than a substitution σ' if there is a substitution δ such that $\sigma' = \sigma\delta$. For two terms s and t, if there is a substitution σ such that $s\sigma \equiv t\sigma$, σ is a *unifier* of s and t. We denote the most general unifier of s and t by $mgu(s,t)$. Let $\square$ be a new symbol which does not occur in $\Sigma \cup V$. A *context*, denoted by C, is a term $t \in T(\Sigma, V \cup \{\square\})$ with exactly one occurrence of $\square$. $C[s]$ denotes the term obtained by replacing $\square$ in C with s. A rewrite rule $l \to r$ is an ordered pair of terms such that l is not a variable and every variable contained in r is also in l. A *term rewriting system (TRS)*, denoted by $\mathcal{R}$, is a set of rewrite rules. The *reduction relation* $\to_{\mathcal{R}} \subseteq T(\Sigma, V) \times T(\Sigma, V)$ is defined by $s \to_{\mathcal{R}} t$ iff there exists a rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s \equiv C[l\sigma]$ and $C[r\sigma] \equiv t$. A term s is *reducible* if $s \to_{\mathcal{R}} t$ for some t; otherwise, s is a *normal form*. A TRS $\mathcal{R}$ *terminates* if there is no infinite rewrite sequence $s_0 \to_{\mathcal{R}} s_1 \to_{\mathcal{R}} \cdots$. A relation R on $T(\Sigma, V)$ is *closed under substitution* if $s\ R\ t$ implies $s\sigma\ R\ t\sigma$ for any substitution σ. A relation R on $T(\Sigma, V)$ is *closed under context* if $s\ R\ t$ implies $C[s]\ R\ C[t]$ for any context C. A *reduction order* $\succ$ is a well-founded strict partial order on $T(\Sigma, V)$ that is closed under substitution and context. The *root symbol* of a term $s \equiv f(s_1, \ldots, s_n)$ is f and denoted by *root*(s). The set of all *defined symbols* of $\mathcal{R}$ is defined by $D_{\mathcal{R}} = \{root(l) \mid l \to r \in \mathcal{R}\}$. The set of all *constructor symbols* of $\mathcal{R}$ is defined by $C_{\mathcal{R}} = \Sigma \setminus D_{\mathcal{R}}$. A term consisting of only constructor symbols and variables is a *constructor term*. The relation $\leftrightarrow^*_{\mathcal{R}}$ is the reflexive, symmetric, transitive closure of the rewrite relation $\to_{\mathcal{R}}$.

2.2 Rewriting Induction

The rewriting induction (RI), proposed by Reddy [17], 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 the equational axioms $\mathcal{R}$ (regarded as a set of equations), i.e., $s\sigma \leftrightarrow^*_{\mathcal{R}} t\sigma$.

Given a set $\mathcal{R}$ of rewrite rules representing equational axioms and a reduction order $\succ$ containing $\mathcal{R}$, 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 $\mathcal{E}$. Figure 1 shows the inference rules of RI proposed in [1].

In Fig. 1, Expd denotes the function defined as follows:

$$\mathrm{Expd}_u(s,t) = \{C[r]\sigma = t\sigma \mid s \equiv C[u], l \to r \in \mathcal{R}, \\ \sigma = mgu(u,l), l : \text{basic}\}$$

where, if necessary, the variables used in $l \to r$ should be renamed in a one-to-one manner so that $V(l,r) \cap V(s,t) = \emptyset$. Let $s = t$ be an equation such that it can be oriented from s to t to form a rewrite rule $s \to t$. Given such an equation $s = t$ and a basic subterm u of s, $\mathrm{Expd}_u(s,t)$ carries out a computation similar to the computation of critical pairs by overlapping u with the basic left-hand sides l of rewrite rules $l \to r$ of $\mathcal{R}$. The resultant equations are collected in a set and returned by Expd. Those equations will be used as new

DELETE	$\langle \mathcal{E} \uplus \{s = s\}, \mathcal{H}\rangle \vdash \langle \mathcal{E}, \mathcal{H}\rangle$
SIMPLIFY	$\langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash \langle \mathcal{E} \cup \{s' = t\}, \mathcal{H}\rangle$ if $s \to_{\mathcal{R}\cup\mathcal{H}} s'$
EXPAND	$\langle \mathcal{E} \uplus \{s = t\}, \mathcal{H}\rangle \vdash$ $\langle \mathcal{E} \cup \mathrm{Expd}_u(s,t), \mathcal{H} \cup \{s \to t\}\rangle$ if $u \in \mathcal{B}(s)$ and $s \succ t$

Fig. 1 Inference rules of RI.

conjectures in the EXPAND inference rule for a case analysis to cover the original conjecture $s = t$, if $\mathcal{R}$ is quasi-reducible. In the succeeding inference steps, the rewrite rule $s \to t$ can be used as an inductive hypothesis.

The DELETE rule removes the trivial equation. The SIMPLIFY rule reduces an equation using a rule of $\mathcal{R}$ and $\mathcal{H}$.

We write $\langle \mathcal{E}, \mathcal{H} \rangle \vdash_{RI} \langle \mathcal{E}', \mathcal{H}' \rangle$ if the latter may be obtained from the former by one application of a rule of RI. Given a set of equations $\mathcal{E}_0$, a quasi-reducible terminating TRS $\mathcal{R}$, and a reduction order $>$ 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 [16] and there is a simply exponential algorithm for it [5]. The possible derivation depends on the choice of the reduction order $>$. It means that its choice is important for the success of inductive theorem proving with the rewriting induction.

Example 2.1: Let us consider the following TRS.

$$\mathcal{R} = \left\{ \begin{array}{lcl} 0 + y & \to & y \\ \mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \end{array} \right.$$

Let $>$ be the lexicographic path order induced by the precedence $+ > \mathsf{s}$. We can prove the associativity of the addition

$$(x + y) + z = x + (y + z)$$

by the following derivation:

$$\begin{array}{ll}
 & \langle \{(x + y) + z = x + (y + z)\}, \{\} \rangle \\
\vdash_{RI} & \left\langle \begin{array}{l} \{y_1 + z = 0 + (y_1 + z), \\ \quad \mathsf{s}(x_1 + y_1) + z = \mathsf{s}(x_1) + (y_1 + z)\}, \\ \{(x + y) + z \to x + (y + z)\} \end{array} \right\rangle \\
 & \quad \text{(by EXPAND) where } u \equiv x + y \\
\vdash^*_{RI} & \left\langle \begin{array}{l} \{y_1 + z = y_1 + z, \\ \quad \mathsf{s}(x_1 + (y_1 + z)) = \mathsf{s}(x_1 + (y_1 + z))\}, \\ \{(x + y) + z \to x + (y + z)\} \end{array} \right\rangle \\
 & \quad \text{(by 4 steps of SIMPLIFY)} \\
\vdash^*_{RI} & \langle \{\}, \{(x + y) + z \to x + (y + z)\} \rangle \\
 & \quad \text{(by 2 steps of DELETE)}
\end{array}$$

where $\vdash^*_{RI}$ denote the reflexive and transitive closure of $\vdash_{RI}$.

Note that in the first inference step, the Expd function is used as follows. Let $u \equiv x+y$, $s \equiv (x+y)+z$, $t \equiv x+(y+z)$, and $C = \square + z$. With the rule $l \equiv 0 + y_1 \to y_1 \equiv r$, obtained by renaming the first rule of $\mathcal{R}$, we have $\sigma = mgu(u, l) = \{x \mapsto 0, y \mapsto y_1\}$, and the conjecture $y_1 + z = 0 + (y_1 + z)$ is generated. Similarly, with the rule $\mathsf{s}(x_1) + y_1 \to \mathsf{s}(x_1 + y_1)$, the conjecture $\mathsf{s}(x_1 + y_1) + z = \mathsf{s}(x_1) + (y_1 + z)$ is generated.

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 arises, because we can often orient an equation in both directions.

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

Fig. 2 Expand rule of RIt.

$$\begin{array}{ll}
\text{DELETE:} & (\mathcal{E} \cup \{s = s\}, \mathcal{R}) \vdash (\mathcal{E}, \mathcal{R}) \\
\text{ORIENT:} & (\mathcal{E} \cup \{s = t\}, \mathcal{R}) \vdash (\mathcal{E}, \mathcal{R} \cup \{s \to t\}) \\
 & \text{if } s \succ t \\
\text{SIMPLIFY:} & (\mathcal{E} \cup \{s = t\}, \mathcal{R}) \vdash (\mathcal{E} \cup \{s = u\}, \mathcal{R}) \\
 & \text{if } t \to_{\mathcal{R}} u \\
\text{COMPOSE:} & (\mathcal{E}, \mathcal{R} \cup \{s \to t\}) \vdash (\mathcal{E}, \mathcal{R} \cup \{s \to u\}) \\
 & \text{if } t \to_{\mathcal{R}} u \\
\text{COLLAPSE:} & (\mathcal{E}, \mathcal{R} \cup \{s \to t\}) \vdash (\mathcal{E} \cup \{u = t\}, \mathcal{R}) \\
 & \text{if } l \to r \in \mathcal{R}, s \to_{\{l \to r\}} u, \text{ and } s \sqsupset l \\
\text{DEDUCE:} & (\mathcal{E}, \mathcal{R}) \vdash (\mathcal{E} \cup \{s = t\}, \mathcal{R}) \\
 & \text{if } u \to_{\mathcal{R}} s \text{ and } u \to_{\mathcal{R}} t
\end{array}$$

Fig. 3 Inference rules of KB.

2.3 Multi-Completion

Given a set $\mathcal{E}_0$ of equations and a reduction order $>$, the standard Knuth-Bendix completion procedure (KB) [14] tries to compute a convergent set $\mathcal{R}_\omega$ of rewrite rules that is contained in $>$ and that induces the same equational theory as $\mathcal{E}_0$. Starting from the initial state $(\mathcal{E}_0, \mathcal{R}_0)$, where $\mathcal{R}_0 = \emptyset$, the procedure obeys the inference system defined in Fig. 3 to generate a sequence $(\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash \cdots$ of deduction. When the set of persistent equations $\mathcal{E}_\omega = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$ is empty, the procedure halts in success, and the resultant system of persistent rewrite rules $\mathcal{R}_\omega = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$ is confluent, terminating (with every rule $l \to r$ satisfying $l > r$), and satisfies $\leftrightarrow^*_{\mathcal{R}_\omega} = \leftrightarrow^*_{\mathcal{E}_0}$.

The *multi-completion* procedure MKB developed in [15] accepts as input a finite *set* $O = \{>_1, \ldots, >_m\}$ of reduction orders as well as a set $\mathcal{E}_0$ of equations. The mission of the procedure is basically the same as KB: it tries to compute a convergent set $\mathcal{R}_\omega$ of rewrite rules that is contained in *some* $>_i$ and that induces the same equational theory as $\mathcal{E}_0$. To achieve this mission, MKB simulates the execution of m parallel processes $P = \{P_1, \ldots, P_m\}$, with P_i executing KB for the reduction order $>_i$ and the common input $\mathcal{E}_0$.

In order to efficiently simulate a lot of closely-related inferences made in different processes, MKB exploits the data structure called nodes and represents the state of the procedure by a set of nodes. Let $I = \{1, 2, \ldots, m\}$ be the set of indexes for orders in O (also for processes in P). A *node* is a tuple $\langle s : t, R_0, R_1, E \rangle$, where $s : t$ (called a *datum*) is an ordered pair of terms, and R_0, R_1, and E (called *labels*) are subsets of I satisfying the following conditions (called *label conditions*):

- $R_0 \cap R_1 = R_1 \cap E = E \cap R_0 = \emptyset$ and
- $i \in R_0$ implies $s >_i t$, and $i \in R_1$ implies $t >_i s$.

Intuitively, R_0 (resp. R_1) denotes the set of indexes of processes in which the current set of rules contains a rule

$s \to t$ (resp. $t \to s$). Similarly, E denotes the set of indexes of processes in which the current set of equations contains an equation $s = t$. The node $\langle s : t, R_0, R_1, E\rangle$ is considered to be identical with the node $\langle t : s, R_1, R_0, E\rangle$.

In the semantics of MKB, the following definition of *projections* relates the information on nodes to the states of processes. Let $n = \langle s : t, R_0, R_1, E\rangle$ be a node and $i \in I$ be an index. The *$\mathcal{E}$-projection* $\mathcal{E}[n,i]$ of n onto i is a (singleton or empty) set of equations defined by

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

Similarly, the *$\mathcal{R}$-projection* $\mathcal{R}[n,i]$ of n onto i is a set of rules defined by

$$\mathcal{R}[n,i] = \begin{cases} \{s \to t\}, & \text{if } i \in R_0, \\ \{t \to s\}, & \text{if } i \in R_1, \\ \emptyset, & \text{otherwise.} \end{cases}$$

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

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

Intuitively, $\mathcal{E}[N,i]$ (resp. $\mathcal{R}[N,i]$) denotes the set of equations (resp. rewrite rules) held in the simulated process P_i, which is the ith process executing KB for the reduction order $\succ_i$, when the state of MKB is N.

$$
\begin{array}{ll}
\textsc{Delete}: & N \cup \{\langle s : s, \emptyset, \emptyset, E\rangle\} \vdash N \text{ if } E \neq \emptyset \\[1ex]
\textsc{Orient}: & N \cup \{\langle s : t, R_0, R_1, E \cup E'\rangle\} \vdash \\
& N \cup \{\langle s : t, R_0 \cup E', R_1, E\rangle\} \\
& \text{if } E' \neq \emptyset, E \cap E' = \emptyset, \\
& \quad \text{and } s \succ_i t \text{ for all } i \in E' \\[1ex]
\textsc{Rewrite-1}: & N \cup \{\langle s : t, R_0, R_1, E\rangle\} \vdash \\
& N \cup \left\{ \begin{array}{l} \langle s : t, R_0 \setminus R, R_1, E \setminus R\rangle \\ \langle s : u, R_0 \cap R, \emptyset, E \cap R\rangle \end{array} \right\} \\
& \text{if } \langle l : r, R, \ldots, \ldots\rangle \in N, t \to_{\{l \to r\}} u, \\
& \quad t \doteq l, \text{ and } (R_0 \cup E) \cap R \neq \emptyset \\[1ex]
\textsc{Rewrite-2}: & N \cup \{\langle s : t, R_0, R_1, E\rangle\} \vdash N \cup \\
& \left\{ \begin{array}{l} \langle s : t, R_0 \setminus R, R_1 \setminus R, E \setminus R\rangle \\ \langle s : u, R_0 \cap R, \emptyset, (R_1 \cup E) \cap R\rangle \end{array} \right\} \\
& \text{if } \langle l : r, R, \ldots, \ldots\rangle \in N, t \to_{\{l \to r\}} u, \\
& \quad t \sqsupset l, \text{ and } (R_0 \cup R_1 \cup E) \cap R \neq \emptyset \\[1ex]
\textsc{Deduce}: & N \vdash N \cup \{\langle s : t, \emptyset, \emptyset, R \cap R'\rangle\} \\
& \text{if } \langle l : r, R, \ldots, \ldots\rangle \in N, \\
& \quad \langle l' : r', R', \ldots, \ldots\rangle \in N, R \cap R' \neq \emptyset, \\
& \quad u \to_{\{l \to r\}} s, \text{ and } u \to_{\{l' \to r'\}} t \\[1ex]
\textsc{Gc}: & N \cup \{\langle s : t, \emptyset, \emptyset, \emptyset\rangle\} \vdash N \\[1ex]
\textsc{Subsume}: & N \cup \left\{ \begin{array}{l} \langle s : t, R_0, R_1, E\rangle \\ \langle s' : t', R'_0, R'_1, E'\rangle \end{array} \right\} \vdash \\
& N \cup \{\langle s : t, R_0 \cup R'_0, R_1 \cup R'_1, E''\rangle\} \\
& \text{if } s : t \text{ and } s' : t' \text{ are variants and} \\
& \quad E'' = (E \setminus (R'_0 \cup R'_1)) \cup (E' \setminus (R_0 \cup R_1))
\end{array}
$$

Fig. 4 Inference rules of MKB.

The MKB procedure is defined by the inference system working on a set N of nodes, as given in Fig. 4. The DELETE, ORIENT, and DEDUCE rules simulate the counterparts of KB, respectively. The SIMPLIFY and COMPOSE rules of KB are simulated by REWRITE-1 and REWRITE-2, while REWRITE-2 additionally simulates the COLLAPSE rule of KB. Gc and SUBSUME (called optional rules) do not necessarily simulate KB, but can affect the efficiency of MKB.

Starting from the initial set of nodes,

$$N_0 = \{\langle s : t, \emptyset, \emptyset, I\rangle \mid s = t \in \mathcal{E}_0\},$$

the procedure generates a sequence $N_0 \vdash N_1 \vdash \cdots$. If, for some N and i, $\mathcal{E}[N,i]$ is empty and all critical pairs of $\mathcal{R}[N,i]$ have been created, MKB returns $\mathcal{R}[N,i]$ as the final result, as the semantics of MKB shows that it is the convergent system obtained from the successful KB sequence computed by the process P_i. Let $\vdash^=$ be $= \cup \vdash$. The following proposition states the soundness of MKB.

Proposition 2.2: If $N \vdash N'$, then for all $i \in I$, $(\mathcal{E}[N,i], \mathcal{R}[N,i]) \vdash^= (\mathcal{E}[N',i], \mathcal{R}[N',i])$.

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 subsection, we show examples in which the choice of appropriate contexts is important.

Example 3.1: Let us consider the following TRS [17].

$$\mathcal{R} = \begin{cases} \mathsf{f}(\mathsf{0}) \to \mathsf{0} \\ \mathsf{f}(\mathsf{s}(\mathsf{0})) \to \mathsf{s}(\mathsf{0}) \\ \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{f}(\mathsf{s}(x)) + \mathsf{f}(x) \\ \mathsf{g}(\mathsf{0}) \to \langle \mathsf{s}(\mathsf{0}), \mathsf{0}\rangle \\ \mathsf{g}(\mathsf{s}(x)) \to \mathsf{np}(\mathsf{g}(x)) \\ \mathsf{np}(\langle x, y\rangle) \to \langle x+y, x\rangle \end{cases}$$

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 $\mathsf{g} > \mathsf{f} > \mathsf{np} > \langle\rangle > + > \mathsf{s} > \mathsf{0}$.

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

$$\langle \mathsf{s}(0), 0\rangle = \langle \mathsf{f}(\mathsf{s}(0)), \mathsf{f}(0)\rangle$$
$$\mathsf{np}(\mathsf{g}(x)) = \langle \mathsf{f}(\mathsf{s}(\mathsf{s}(x))), \mathsf{f}(\mathsf{s}(x))\rangle,$$

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 example 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 [10] where we need to choose appropriate orientation of conjectures.

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

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 four conjectures are lemmas needed for proving them.

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

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

Example 3.3: We show an example [10] where the reduction strategy plays an important role.

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

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 $\mathcal{R}$.

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

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

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

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 is better to select the rule from the inductive hypotheses (in $\mathcal{H}$) than the axiomatic rules (in $\mathcal{R}$). In our example, however, the strategy failed. Other people might think that, for example, the outermost reduction strategy may be effective. Anyway, $\mathcal{R} \cup \mathcal{H}$ is not confluent in general, and no one knows a correct strategy of simplification. 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 $\langle \mathcal{E}_1, \mathcal{H}_1\rangle, \ldots, \langle \mathcal{E}_n, \mathcal{H}_n\rangle$ 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 with 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 $a_1 a_2 \ldots a_k$, which can be interpreted as a position in a tree. If the process with the index $p = a_1 a_2 \ldots a_k$ have n possible choices of contexts, we have it fork into n processes: $a_1 a_2 \ldots a_k 1, a_1 a_2 \ldots a_k 2, \ldots, a_1 a_2 \ldots a_k n$. Based on the label representation, we can simulate the fork operation by replacing the label p in the labels of all nodes with the set of n identifiers $p1, \ldots, pn$.

For the purpose of formal treatment, we introduce the *fork function*. Let us define the set I of all indexes as the prefix-closed subset of $\mathbb{N}^*$ such that $pn \in I$ implies $pj \in I$ for all j in $\{1, \ldots, n-1\}$. We do not distinguish between a process and its index.

Definition 3.4: *Fork function* $\psi : I \to \mathbb{N}$ 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 if 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 (resp. H_2) represents all processes containing $s \to t$ (resp. $t \to s$) 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 p be an index. The $\mathcal{E}$- and $\mathcal{H}$-projections of n onto p are defined as follows:

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

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

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

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

$\mathcal{E}[N, p]$ 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. 5. In the inference rules, $I(N)$ denotes the set of all processes that appear in a label of a node in N and $sub(N, L) = \{\langle s : t, H_1 \setminus L, H_2 \setminus L, E \setminus L\rangle \mid \langle s : t, H_1, H_2, E\rangle \in N\}$.

Let us elaborate on the EXPAND and SIMPLIFY-R rules, which are characteristic of MRIt. The EXPAND rule focuses

DELETE:	$N \cup \{\langle s : s, H_1, H_2, E\rangle\} \vdash N$
EXPAND:	$N \cup \{\langle s : t, H_1, H_2, E \uplus E'\rangle\} \vdash$ $N \cup \{\langle s : t, H_1 \cup E', H_2, E\rangle\} \cup$ $\{\langle s' : t', \emptyset, \emptyset, E'\rangle \mid s' = t' \in \mathrm{Expd}_u(s,t)\}$ if $E' \neq \emptyset, u \in \mathcal{B}(s)$ and $\mathcal{H}[N,i] \cup \mathcal{R} \cup$ $\{s \to t\}$ terminates for all $i \in E'$
SIMPLIFY-R:	$N \cup \{\langle s : t, H_1, H_2, E\rangle\} \vdash$ $N \cup \left\{ \begin{array}{l} \langle s : t, H_1, H_2, \emptyset\rangle \\ \langle s' : t, \emptyset, \emptyset, E\rangle \end{array} \right\}$ if $E \neq \emptyset$ and $s \to_{\mathcal{R}} s'$
SIMPLIFY-H:	$N \cup \{\langle s : t, H_1, H_2, E\rangle\} \vdash$ $N \cup \left\{ \begin{array}{l} \langle s : t, H_1, H_2, E \setminus H\rangle \\ \langle s' : t, \emptyset, \emptyset, E \cap H\rangle \end{array} \right\}$ if $E \cap H \neq \emptyset, \langle l : r, H, \ldots, \ldots\rangle \in N$, and $s \to_{\{l \to r\}} s'$,
FORK:	$N \vdash \psi_P(N)$ for some fork function ψ and a set P of processes in N
GC:	$N \cup \{\langle s : t, \emptyset, \emptyset, \emptyset\rangle\} \vdash N$
SUBSUME:	$N \cup \left\{ \begin{array}{l} \langle s : t, H_0, H_1, E\rangle \\ \langle s' : t', H_0', H_1', E'\rangle \end{array} \right\} \vdash$ $N \cup \{\langle s : t, H_0 \cup H_0', H_1 \cup H_1', E''\rangle\}$ if $s : t$ and $s' : t'$ are variants and $E'' = (E \setminus (H_0' \cup H_1')) \cup (E' \setminus (H_0 \cup H_1))$
SUBSUME-P:	$N \vdash sub(N, L)$ if $\forall p \in L, \exists p' \in I(N) \setminus L :$ $(\mathcal{E}[N,p], \mathcal{H}[N,p]) = (\mathcal{E}[N,p'], \mathcal{H}[N,p'])$

Fig. 5 Inference rules of MRIt.

on a node $n = \langle s : t, H_1, H_2, E \uplus E'\rangle$, and applies the EXPAND rule of RI in all processes of E' that can orient the equation $s = t$ from left to right. The set E' is moved from the third label to the first in n since in each process in E' the conjecture $s = t$ is removed and the new hypothesis $s \to t$ is added after the expansion. In addition, for each new conjecture $s' = t'$ in $\mathrm{Expd}_u(s,t)$, a new node $\langle s' : t', \emptyset, \emptyset, E'\rangle$ is created in order to store the conjecture in the processes of E'.

Example 3.6: If we continue the reasoning of Example 3.3, we will reach a state $N \uplus \{n\}$, where

$$\begin{aligned} n &= \langle s : t, \emptyset, \emptyset, E'\rangle, \\ s &\equiv \mathsf{b}(\mathsf{b2}(x, xs))@[\mathsf{b1}(x, xs)], \\ t &\equiv x : xs, \\ E' &= \{1, 2\}. \end{aligned}$$

By applying the EXPAND rule with

$$u = \mathsf{b1}(x, xs)$$

and the rules

$$\begin{aligned} &\mathsf{b1}(y, []) \to y \\ &\mathsf{b1}(y, z : zs) \to \mathsf{b1}(z, zs) \end{aligned}$$

we obtain the following set of nodes:

$$N \cup \left\{ \begin{array}{ccccc} \langle & s : t, & E', & \emptyset, & \emptyset & \rangle \\ \langle & s' : t', & \emptyset, & \emptyset, & E' & \rangle \\ \langle & s'' : t'', & \emptyset, & \emptyset, & E' & \rangle \end{array} \right\}$$

where

$$\begin{aligned} s' &\equiv \mathsf{b}(\mathsf{b2}(y, []))@[y], \\ t' &\equiv [y], \\ s'' &\equiv \mathsf{b}(\mathsf{b2}(y, z : zs))@[\mathsf{b1}(z, zs)] \\ t'' &\equiv y : (z : zs). \end{aligned}$$

The SIMPLIFY-R rule applies the SIMPLIFY rule of RI using a rewrite rule in the equational axiom $\mathcal{R}$, which is common to all processes. E is the set of all processes that have $s = t$ as a conjecture. Since this equation is transformed to an equation $s' = t$, the set E is removed from the original node, and a new node $\langle s' : t, \emptyset, \emptyset, E\rangle$ is created.

Example 3.7: We consider the state after the inference in Example 3.6. Let the current set of nodes be $N \uplus \{n\}$, where

$$\begin{aligned} n &= \langle s : t, \emptyset, \emptyset, E\rangle, \\ s &\equiv \mathsf{b}(\mathsf{b2}(y, []))@[y], \\ t &\equiv [y], \\ E &= \{1, 2\}. \end{aligned}$$

By applying the REWRITE-R rule with the rule

$$\mathsf{b2}(x, []) \to []$$

we obtain the following set of nodes:

$$N \cup \left\{ \begin{array}{ccccc} \langle & s : t, & \emptyset, & \emptyset, & \emptyset & \rangle \\ \langle & s' : t', & \emptyset, & \emptyset, & E & \rangle \end{array} \right\}$$

where

$$\begin{aligned} s' &\equiv \mathsf{b}([])@[y], \\ t' &\equiv [y]. \end{aligned}$$

The role of each of the remaining inference rules 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. The SIMPLIFY-H rule is almost the same as SIMPLIFY-R. The difference is that SIMPLIFY-R applies a rule of $\mathcal{R}$, which is common to all processes, while SIMPLIFY-H applies an inductive hypothesis of $\mathcal{H}$, which may exist only in some distinguished processes. This makes the third labels of the original node and the new node $E \setminus H$ and $E \cap H$, respectively. FORK, newly introduced in this system, enables us to produce new copies of existing processes to make nondeterministic choices in parallel. In the next subsection, 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 3.8: 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 $\langle \mathcal{E}[N,p], \mathcal{H}[N,p] \rangle = \langle \mathcal{E}[N',q], \mathcal{H}[N',q] \rangle$.

Let $\vdash_{RI}^{=}$ be $= \cup \vdash_{RI}$. The following proposition states that other inference rules either simulate RI rules or have no effect for RI processes.

Proposition 3.9: 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 $p \in I$.

3.4 Strategies Based on Process Fork

In MRIt, we introduced a new inference rule FORK, which enables us to try multiple contexts in parallel. However, the rule only generates copies of existing processes. In order to fully exploit the rule, we need to combine it with other inference rules which involve nondeterminism; that is, we need to identify nondeterministic choices, set the fork function ψ accordingly, and commit each process generated by FORK to one of the choices. As shown in Sect. 3.1, important nondeterminism is involved in the expansion and simplification operations. In this section, we present a strategy for combining FORK with those two operations.

3.4.1 Expansion

We present a way of making all nondeterministic choices in the expansion. It involves (1) the direction of orientation and (2) the choice of the basic subterm to be expanded. Let us assume that the current set of nodes is $N \uplus \{n = \langle s : t, H_1, H_2, E \rangle\}$ and we choose the node n as the target of expansion. Let $L_{lr}, L_{rl} \subseteq E$ be two sets such that

- $\mathcal{R} \cup \mathcal{H}[N,p] \cup \{s \to t\}$ terminates for all $p \in L_{lr}$
- $\mathcal{R} \cup \mathcal{H}[N,p] \cup \{t \to s\}$ terminates for all $p \in L_{rl}$

Since each process p in $L = L_{lr} \cap L_{rl}$ has two nondeterministic choices in the direction of orientation, we have a fork function with

$$\psi(p) = 2 \quad \text{for all } p \in L.$$

Let $L'_{lr} = L_{lr} \setminus L_{rl} \cup \{p.1 \mid p \in L\}$ and $L'_{rl} = L_{rl} \setminus L_{lr} \cup \{p.2 \mid p \in L\}$. If $\mathcal{B}(s) = \{s_1, \ldots, s_j\}$ and $\mathcal{B}(t) = \{t_1, \ldots, t_k\}$, all processes $p' \in L'_{lr}$ have j choices and $q' \in L'_{rl}$ have k choices with respect to which subterm to expand. Thus we have a fork function satisfy

$$\begin{aligned} \psi'(p) &= j \quad \text{for all } p \in L'_{lr}, \\ \psi'(p) &= k \quad \text{for all } p \in L'_{rl}. \end{aligned}$$

Let $L' = L'_{lr} \cup L'_{rl}$, $L^i_{lr} = \{p.i \mid p \in L'_{lr}\}$ for $1 \le i \le j$, $L^i_{rl} = \{p.i \mid p \in L'_{rl}\}$ for $1 \le i \le k$, $H'_1 = H_1 \cup \bigcup_{1 \le i \le j} L^i_{lr}$, and $H'_2 = H_2 \cup \bigcup_{1 \le i \le k} L^i_{rl}$. Intuitively, L'_{lr} (resp. L'_{rl}) is the set of all processes which orient the equation $s = t$ from left to right (resp. right to left). L^i_{lr} (resp. L^i_{rl}) is the set of processes which expand the ith basic term of the left-hand (resp. right-hand) side of the equation.

Example 3.10: Let $\langle s : t, \emptyset, \emptyset, \{1,2\} \rangle$ be the target of expansion, and let $\mathcal{B}(s) = \{s_1, s_2\}, \mathcal{B}(t) = \{t_1, t_2, t_3\}, L_{lr} = \{1,2\}$, and $L_{rl} = \{2\}$. From the above definitions, we can calculate the labels as follows:

$$\begin{aligned} &L = \{2\}, L'_{lr} = \{1, 2.1\}, L'_{rl} = \{2.2\}, \\ &L' = \{1, 2.1, 2.2\}, \\ &L^1_{lr} = \{1.1, 2.1.1\}, L^2_{lr} = \{1.2, 2.1.2\}, \\ &L^1_{rl} = \{2.2.1\}, L^2_{rl} = \{2.2.2\}, L^3_{rl} = \{2.2.3\}, \\ &H'_1 = \{1.1, 1.2, 2.1.1, 2.1.2\}, \text{ and} \\ &H'_2 = \{2.2.1, 2.2.2, 2.2.3\}. \end{aligned}$$

Let

$$\begin{aligned} N' = {} & \psi'_{L'}(\psi_L(N)) \\ & \cup \{\langle s : t, H'_1, H'_2, E \setminus (L_{lr} \cup L_{rl}) \rangle\} \\ & \cup \{\langle s' : t', \emptyset, \emptyset, L^i_{lr} \rangle \mid 1 \le i \le j, \\ & \quad s' = t' \in \mathrm{Expd}_{s_i}(s,t)\} \\ & \cup \{\langle t' : s', \emptyset, \emptyset, L^i_{rl} \rangle \mid 1 \le i \le k, \\ & \quad t' = s' \in \mathrm{Expd}_{t_i}(t,s)\}. \end{aligned}$$

Then it is not hard to verify the following proposition.

Proposition 3.11: $N \uplus \{\langle s : t, H_1, H_2, E \rangle\} \vdash^* N'$.

3.4.2 Simplification

Let us now present a way of making nondeterministic choices in the simplification for obtaining a normal form of the target term. We denote by $NF_{\mathcal{R}}(s)$ the set of normal forms of a term s with respect to a terminating TRS $\mathcal{R}$. We call a pair $\langle s, L \rangle$ of a term s and a label L, which is a set of processes, a *labeled term* and a pair $\langle l \to r, L \rangle$ of a rule and a label a *labeled rule*. Intuitively, L represents a set of processes which have the term s, or the rule $l \to r$. We denote a labeled term by $\widehat{s} = \langle s, L \rangle$, its term and label fields by $s = term(\widehat{s})$ and $L = label(\widehat{s})$ respectively. The *term set* projection $terms(\widehat{T}, p)$ of a set $\widehat{T}$ of labeled terms onto a process p is defined by $terms(\widehat{T}, p) = \{term(\widehat{s}) \mid \widehat{s} \in \widehat{T}, p \in label(\widehat{s})\}$. The *label* of $\widehat{T}$, denoted by $label(\widehat{T})$, is defined by $label(\widehat{T}) = \bigcup_{\widehat{s} \in \widehat{T}} label(\widehat{s})$. The *rule set* projection $rules(\widehat{\mathcal{R}}, p)$ of a set $\widehat{\mathcal{R}}$ of labeled rules onto p is defined by $rules(\widehat{\mathcal{R}}, p) = \{l \to r \mid \langle l \to r, L \rangle \in \widehat{\mathcal{R}}, p \in L\}$. We introduce the rewrite relation on labeled terms defined by the labeled rules $\widehat{\mathcal{R}}$ as follows:

$$\begin{aligned} &\langle s, L \rangle \to_{\widehat{\mathcal{R}}} \langle s', L' \rangle \\ &\quad \text{if } \langle l \to r, L'' \rangle \in \widehat{\mathcal{R}}, s \to_{\{l \to r\}} s' \text{ and } L' = L \cap L'' \neq \emptyset \end{aligned}$$

The set of *labeled normal forms* $NF_{\widehat{\mathcal{R}}}(\widehat{s})$ of a labeled term $\widehat{s}$ is defined by

$$NF_{\widehat{\mathcal{R}}}(\widehat{s}\,) = \bigcup_{\widehat{s}\,' \in SR_{\widehat{\mathcal{R}}}(\widehat{s}\,)} NF_{\widehat{\mathcal{R}}}(\widehat{s}\,') \cup \{rest(\widehat{s}, SR_{\widehat{\mathcal{R}}}(\widehat{s}\,))\}$$

where

$$SR_{\widehat{\mathcal{R}}}(\widehat{s}\,) = \{\widehat{s}\,' \mid \widehat{s} \to_{\widehat{\mathcal{R}}} \widehat{s}\,'\} \text{ and}$$
$$rest(\langle t, L\rangle, \widehat{T}) = \langle t, L \setminus label(\widehat{T})\rangle.$$

We can interpret this definition as follows. Suppose $\widehat{s} = \langle s, L\rangle \to_{\widehat{\mathcal{R}}} \langle s', L'\rangle = \widehat{s}\,'$, meaning that s is reducible to s' in the processes of $L' \subseteq L$. We find all such pairs $\langle s', L'\rangle$ in all possible ways, using the labeled rules of $\widehat{\mathcal{R}}$. Let L^* be the union of all such L's. Then s cannot be reduced in any processes of $L \setminus L^*$. Therefore, s is a normal form in the processes of $L \setminus L^*$, meaning that $\langle s, L \setminus L^*\rangle$ is a labeled normal form of $\widehat{s} = \langle s, L\rangle$. Additional labeled normal forms of $\widehat{s}$ can be obtained by applying this definition recursively to each $\widehat{s}\,' = \langle s', L'\rangle$ found above.

We can verify the following proposition.

Proposition 3.12: For any labeled term $\widehat{s}$ and $p \in label(\widehat{s}\,)$, $terms(NF_{\widehat{\mathcal{R}}}(\widehat{s}\,), p) = NF_{rules(\widehat{\mathcal{R}},p)}(term(\widehat{s}\,))$.

We can define the projection $NF_{\widehat{\mathcal{R}}}(\widehat{s}, p)$ of normal forms of $\widehat{s}$ onto a process p as follows:

$$NF_{\widehat{\mathcal{R}}}(\widehat{s}, p) = \{\widehat{s}\,' \mid \widehat{s}\,' \in NF_{\widehat{\mathcal{R}}}(\widehat{s}\,), p \in label(\widehat{s}\,')\}.$$

Now we construct the set of nodes N' obtained by all possible ways of simplification for a node n at the current state $N \uplus \{n\}$. Let us assume that the current set of nodes is $N \uplus \{n = \langle s : t, H_1, H_2, E\rangle\}$ and we choose the node n as the target of simplification.

Let

$$\begin{aligned}
&\widehat{s} = \langle s, E\rangle,\\
&\widehat{\mathcal{R}} = \{\langle l \to r, E\rangle \mid l \to r \in \mathcal{R}\} \cup \{\langle l \to r, H_1'\rangle, \langle r \to l, H_2'\rangle \mid\\
&\quad \langle l : r, H_1', H_2', E'\rangle \in N\},\\
&NF_{\widehat{\mathcal{R}}}(\widehat{s}\,) = \{\widehat{s}_1, \ldots, \widehat{s}_n\}, \text{ and}\\
&\psi(p) = |NF_{\widehat{\mathcal{R}}}(\widehat{s}, p)| \text{ for all } p \in E.
\end{aligned}$$

Let

$$\begin{aligned}
N' = {}& \psi_E(N) \cup \{\langle s : t, H_1, H_2, \emptyset\rangle\}\\
&\cup \{\langle term(\widehat{s}_i) : t, \emptyset, \emptyset, E_i\rangle \mid 1 \le i \le n\}
\end{aligned}$$

where

$$\begin{aligned}
E_i = \{&p.j \mid p \in label(\widehat{s}_i), NF_{\widehat{\mathcal{R}}}(\widehat{s}, p) = \{\widehat{s}_{k_1}, \ldots, \widehat{s}_{k_{\psi(p)}}\},\\
&k_1 < \cdots < k_{\psi(p)}, \widehat{s}_i = \widehat{s}_{k_j}\}.
\end{aligned}$$

When the ith element $\widehat{s}_i$ of the set of labeled normal forms $NF_{\widehat{\mathcal{R}}}(\widehat{s}\,)$ appears as the jth element of the set of labeled normal forms $NF_{\widehat{\mathcal{R}}}(\widehat{s}, p)$ of a process p, we have the process $p.j$ take care of the normal form $term(\widehat{s}_i)$. E_i is the set of all processes which take care of the ith normal form $term(\widehat{s}_i)$.

Example 3.13: Let $\widehat{s} = \langle s, \{1,2,3\}\rangle$ and $NF_{\widehat{\mathcal{R}}}(\widehat{s}\,) = \{\widehat{s}_1, \widehat{s}_2, \widehat{s}_3\}$ where

$$\begin{aligned}
\widehat{s}_1 &= \langle s_1, \{1,3\}\rangle,\\
\widehat{s}_2 &= \langle s_2, \{1\}\rangle,\\
\widehat{s}_3 &= \langle s_3, \{2,3\}\rangle.
\end{aligned}$$

Then $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 1) = \{\widehat{s}_1, \widehat{s}_2\}$, because the index 1 in the second argument belongs to the labels of $\widehat{s}_1$ and $\widehat{s}_2$, meaning that s_1 and s_2 are normal forms in the process 1. Similarly, we have

$$NF_{\widehat{\mathcal{R}}}(\widehat{s}, 2) = \{\widehat{s}_3\}$$

and

$$NF_{\widehat{\mathcal{R}}}(\widehat{s}, 3) = \{\widehat{s}_1, \widehat{s}_3\}.$$

Now let us calculate E_i for $i = 1$: which processes will take care of s_1? Since $label(\widehat{s}_1) = \{1,3\}$, we have to consider the cases $p = 1$ and $p = 3$. For $p = 1$, $\widehat{s}_1$ appears as the first ($j = 1$) element of $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 1) = \{\widehat{s}_1, \widehat{s}_2\}$. For $p = 3$, $\widehat{s}_1$ appears as the first ($j = 1$) element again in $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 3) = \{\widehat{s}_1, \widehat{s}_3\}$. Therefore, $E_1 = \{1.1, 3.1\}$. Similarly, $E_2 = \{1.2\}$, because $p = 1$ is the only element of $label(\widehat{s}_2)$ and $\widehat{s}_2$ is the second ($j = 2$) element of $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 1)$. For E_3, we have to consider $p = 2$ and $p = 3$; for $p = 2$, we have $j = 1$ as $\widehat{s}_3$ is the first element of $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 2)$; for $p = 3$, we have $j = 2$ as $\widehat{s}_3$ is the second element of $NF_{\widehat{\mathcal{R}}}(\widehat{s}, 3)$; Therefore, $E_3 = \{2.1, 3.2\}$.

In summary, the process 1 is forked to 1.1 and 1.2 to take care of s_1 and s_2, respectively; the process 3 is forked to 3.1 and 3.2 to take care of s_1 and s_3, respectively.

We can verify the following proposition.

Proposition 3.14: $N \uplus \{\langle s : t, H_1, H_2, E\rangle\} \vdash^* N'$.

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], [8], [9]. Moreover, in order to find reduction orders for ensuring termination, we used the combination of polynomial interpretation and SAT solving, as proposed in [7]. All experiments were performed on a workstation equipped with Intel Xeon 2.13 GHz CPU and 1 GB system memory.

In the implementation, we used the following strategy for applying inference rules of MRIt.

(1) First, choose a node n with the smallest size, where the size of a node $\langle s : t, \ldots\rangle$ is defined as the sum of the sizes of s and t, and the size of a term is defined as the number of symbols constituting the term. Then apply EXPAND rule (combining FORK rule in the way of Sect. 3.4.1)

(2) Next, normalize all nodes by applying SIMPLIFY-R and SIMPLIFY-H rules (combining FORK rule in the way of Sect. 3.4.2)

(3) Finally, apply DELETE, GC, SUBSUME, and SUBSUME-P rules as much as possible, and (if no process succeeds) go back to the step (1)

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 (3, 3)
Ex. 3.2	0.357	0.255	0.079	20 (20, 20)
Ex. 3.3	0.233	0.129	0.092	10 (16, 19)

Table 2 Computation time of Dream Corpus examples.

Problem	Total	Term	Simplify	# of proc.
109	0.347	0.281	0.043	6 (7, 11)
301	0.305	0.244	0.044	6 (7, 7)
115	0.042	0.026	0.010	1 (1, 1)
1018	0.028	0.014	0.009	3 (3, 3)
216	0.014	0.003	0.010	2 (2, 2)
total	0.876	0.627	0.169	

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. The numbers in the parenthesis are the maximum and the total number of processes during the computation.

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 the Dream Corpus examples[†], 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 0.01 sec computation time. The bottommost row indicates the total time of all the 35 problems solved, including those with less than 0.01 sec CPU time.

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.

Table 3 Comparison of simplification strategies using Ex. 3.2.

Strategy	Total	Simp	# of Simp	# of proc.
(a)	0.357	0.079	17681	20 (20, 20)
(b)	0.395	0.122	15493	20 (20, 20)

Table 4 Comparison of simplification strategies using Ex. 3.3.

Strategy	Total	Simp	# of Simp	# of proc.
(a)	0.233	0.092	27434	10 (16, 19)
(b)	0.197	0.076	10001	4 (5, 7)

Next, in order to know the efficiency of MRIt, we compare the following two strategies in normalization of the nodes:

(a) SIMPLIFY with FORK: calculates the set of all normal forms, and forks the processes accordingly in the way of Sect. 3.4.1.

(b) SIMPLIFY without FORK: calculates only one of the normal forms using specific reduction strategy (leftmost outermost) for SIMPLIFY operation.

The results of experimentation with the Example 3.2 and Example 3.3 are shown in Table 3 and 4, respectively. The "# of Simp" column shows the number of simplification. We can see that the number of simplification was reduced in both examples by using the leftmost outermost strategy. Moreover, all 35 examples in Dream Corpus, solved without fixing a reduction strategy, were also solved with the leftmost outermost strategy. Although the time for simplification is not proportionally reduced, it is because our implementation of MRIt was not optimized for outermost strategy. From these results, we can expect that the leftmost outermost strategy is useful in various cases. However, of course, the strategy does not always work successfully in general. Moreover, the result of the experiments shows that the cost for trying all possibilities in normalization without fixing a reduction strategy is not extremely expensive. Therefore, we can say that it makes sense to use the SIMPLIFY with FORK strategy as an initial selection.

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 and automated generation of lemmas.

[†]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).

Acknowledgements

This work was partially supported by JSPS Grant-in-Aid for Scientific Research (C), No.19500020. We are very grateful to Takahito Aoto for helpful comments and providing organized Dream Corpus datasets.

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.

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

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

[5] H. Comon and J. Florent, "Ground reducibility is EXPTIME-complete," J. Information and Computation, vol.187, no.1, pp.123–153, 2003.

[6] N. Dershowitz and J.-P. Jouannaud, "Rewrite systems," in Handbook of Theoretical Computer Science, ed. J. van Leeuwen, vol.B, pp.243–320, MIT Press, 1990.

[7] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl, "SAT solving for termination analysis with polynomial interpretations," Proc. 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007), vol.4501 of Lecture Notes in Computer Science, pp.340–354, 2007.

[8] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke, "Mechanizing and improving dependency pairs," J. Automated Reasoning, vol.37, no.3, pp.155–203, 2006.

[9] N. Hirokawa and A. Middeldorp, "Tyrolean termination tool: Techniques and features," Inf. Comput., vol.205, no.4, pp.474–511, 2007.

[10] G. Huet and J.-M. Hullot, "Proofs by induction in equational theories with constructor," J. Comput. Syst. Sci., vol.25, no.2, pp.239–266, 1982.

[11] J.-P. Jouannaud and E. Kounalis, "Automatic proofs by induction in theories without constructors," Inf. Comput., vol.82, no.1, pp.1–33, 1989.

[12] D. Kapur, P. Narendran, and H. Zhang, "Automating inductionless induction using test sets," J. Symbolic Computation, vol.11, no.1, pp.81–111, 1991.

[13] J.W. Klop, "Term rewriting systems," in Handbook of Logic in Computer Science, ed. S. Abramsky et al., pp.1–116, Oxford University Press, 1992.

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

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

[16] D. Plaisted, "Semantic confluence tests and completion methods," J. Information and Control, vol.65, pp.182–215, 1985.

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

[18] 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.

[19] H. Sato, M. Kurihara, S. Winkler, and A. Middeldorp, "Constraint-based multi-completion procedures for term rewriting systems," IEICE Trans. Inf. & Syst., vol.E92-D, no.2, pp.220–234, Feb. 2009.

[20] Terese, Term Rewriting Systems, Cambridge University Press, 2003.

[21] 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.

Haruhiko Sato received a BS, a MS, and a Ph.D. in information engineering from Hokkaido University in 2005, 2007, and 2008 respectively. He is currently a assistant professor of information science at Hokkaido University. His research interests include term rewriting systems, automated theorem proving, and software engineering. He is a member of IPSJ, JSSST.

Masahito Kurihara received a BS in electrical engineering, and a MS and a Ph.D. in information engineering from Hokkaido University, in 1978, 1980 and 1986 respectively. He is currently a professor of information science at Hokkaido University. His research interests include mathematical logic and automated reasoning in computer science and artificial intelligence.