IEICE TRANSACTIONS

on Information and Systems

VOL. E102-D NO. 2
FEBRUARY 2019

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Multi-Context Automated Lemma Generation for Term Rewriting Induction with Divergence Detection

Chengcheng JI[†a)], ***Nonmember***, **Masahito KURIHARA**[†b)], ***and*** **Haruhiko SATO**[††c)], ***Members***

SUMMARY We present an automated lemma generation method for equational, inductive theorem proving based on the term rewriting induction of Reddy and Aoto as well as the divergence critic framework of Walsh. The method effectively works by using the divergence-detection technique to locate differences in diverging sequences, and generates potential lemmas automatically by analyzing these differences. We have incorporated this method in the multi-context inductive theorem prover of Sato and Kurihara to overcome the strategic problems resulting from the unsoundness of the method. The experimental results show that our method is effective especially for some problems diverging with complex differences (i.e., parallel and nested differences).

key words: *term rewriting systems, term rewriting induction, multi-context induction, lemma generation*

1. Introduction

Automated inductive theorem proving based on equational logics has been studied for decades. The frameworks for inductive theorem proving are classified into two categories: the *explicit* induction which directly follows the paradigm of inductions, and the *implicit* induction in which some part of the paradigm is implicit in the sense that, typically, the induction rule and/or the well-founded induction order need not be provided explicitly by the users [33]. These two types of induction both have to find an induction pattern (a finite cyclic representation) for an infinite deductive proof and an induction order for ensuring the termination. The theorem prover Nqthm developed by Boyer&Moore [9] stands for the former one, because it requires that the induction patterns be provided as inference rules. Generally speaking, however, providing such induction patterns is difficult in practice. The *inductionless induction* proposed and extended by [15], [21] falls into the latter category, because, with no induction patterns provided, it implicitly tries to prove inductive theorems based on the principle of ground completion. However, this method was claimed to be so inefficient and practically useless, because, based on the notion of "proof by consistency", it has to search all over the search space for an inconsistency before it concludes that there is no inconsistency and thus the given formula should be certainly an inductive theorem [33]. On the other hand, the *term rewriting induction* (RI for short) proposed by Reddy [25] and enhanced by Aoto [1], [2] implicitly obtains the induction patterns by the inference procedure itself and in this sense, it is a practically promising framework for implicit induction.

In general, an execution of RI leads to one of the following three results: success, failure, or divergence. The *divergence* occurs when the procedure generates in vain an infinite sequence of conjectures which cannot be proved automatically as lemmas for establishing the target theorem. To avoid the divergence, the users often need to supply appropriate lemmas which can be proved and used to solve the overall problems, but in practice, requiring their mathematical intuition and experience, this is so difficult to general users.

Automated lemma generation, therefore, is desired. Generally, there are two categories for lemma generation methods: *bottom-up* and *target-aimed* (top-down). The bottom-up methods generate lemmas from the given equational axioms with no consideration of the target theorem [18]. These methods have outstanding ability of generating conjectures, but their computational cost is extremely high. Meanwhile, the target-aimed methods work in a different way by considering the candidates for conjecture to generate potentially appropriate lemmas [22], [31]. Even though the generating power is limited, the computational cost is comparatively low. Thus it is desirable to strengthen the power of the target-aimed method while preserving its acceptable cost.

Target-aimed methods are classified into *sound* and *unsound* ones. The sound methods [22] generate only correct conjectures in the sense that the goal is an inductive theorem if and only if the generated conjectures are inductive theorems. These methods have a very low computational cost, but the ability of generating appropriate lemmas is extremely low. On the other hand, the unsound methods [31] try to generate useful conjectures without being restricted by the soundness. This gives higher ability of generating appropriate lemmas with a modest computational cost. We focus on the latter one to keep balance between power and cost. Basically, the unrestricted use of classic generalization techniques for the unsound methods based on replacing constants and ground terms with universally-quantified variables limits the practical usefulness. The framework based on divergence-detection proposed by Walsh [31] greatly im-

Manuscript received November 10, 2017.
Manuscript revised September 20, 2018.
Manuscript publicized November 12, 2018.

[†]The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.

[††]The author is with the Faculty of Engineering, Hokkai-Gakuen University, Sapporo-shi, 064–0926 Japan.

a) E-mail: kisyousei@complex.ist.hokudai.ac.jp
b) E-mail: kurihara@ist.hokudai.ac.jp
c) E-mail: h-sato@hgu.jp

DOI: 10.1587/transinf.2017EDP7368

proves the practical usefulness by the following steps: 1) detect a potential divergence from the sequence of generated conjectures; 2) generate candidates for lemma by locating the differences between two consecutive conjectures in the diverging sequence.

In this paper, we put into the Walsh's framework a new heuristic lemma generation method, *peripheral sculpture*, to make the theorem prover more powerful without introducing significant cost increase. Since the method is unsound, there is no guarantee of the correctness of the generated candidates for lemma. If the system accepts a wrong lemma, it could cause an infinite sequence of wasteful inferences. While it is generally not easy to decide which newly generated lemma candidates to add to a currently running process, we can add all candidates by separating the choices of whether to accept each of them into parallel contexts. This can be implemented in a multi-context reasoning framework that effectively simulates related inductions. The combination of multi-context induction and lemma generation is thus an attractive research area in automated reasoning. As a first step in such research, we used our lemma generation method to extend the efficient *multi-context rewriting induction system* of Sato and Kurihara [26]. The experimental results show that, with no much redundant costs, we have succeeded in solving several lemma-required benchmark problems which encountered complex differences (i.e., parallel and nested differences) and which the original systems [26], [31] could not solve.

This paper is organized as follows. In Sect. 2, we provide a brief overview of term rewriting systems and term rewriting induction. In Sect. 3, we discuss the details of divergence detection. In Sect. 4, we introduce the peripheral sculpture method. In Sect. 5, we introduce the multi-context postulation. In Sect. 6, we discuss the experimental results, and conclude in Sect. 7.

2. Preliminaries

2.1 Term Rewriting System

Assuming that readers are familiar with the basic notions for term rewriting systems as summarized in [5], [11], [19], [24], [30], we basically review only some definitions and notations used in this paper.

A *signature* (i.e., a set of function symbols equipped with their fixed arity) is denoted by $\Sigma = \{f, g, h, \ldots\}$. The set of *variables* is denoted by $V = \{x, y, z, \ldots\}$. A *term* is either a variable or a function symbol (of arity $n \geq 0$) followed by n terms as arguments (possibly delimited by commas and enclosed by parentheses). Function symbols of arity 0 are *constants*. When the function symbol is binary (i.e., of arity 2) and its name consists of special characters (such as + and :), the term may be written in infix form (such as $0 + x$ and $x : nil$). The set of *terms* over Σ and V is denoted by $T(\Sigma, V)$. The set of variables occurring in a term t is represented by $\mathcal{V}(t)$. The symbol $\equiv$ denotes the identity for terms.

A *substitution* σ is a mapping from V to $T(\Sigma, V)$ such that $\sigma(x) \neq x$ for only finitely many xs, and is extended to a mapping from $T(\Sigma, V)$ to $T(\Sigma, V)$ by defining $\sigma(f(s_1, \ldots, s_m)) = f(\sigma(s_1), \ldots, \sigma(s_m))$, where m is the arity of f. The *domain* of σ is the set $Dom(\sigma) = \{x \in V \mid \sigma(x) \neq x\}$. We write a substitution σ as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ if $\sigma(x_i) = t_i$ for $x_i \in \{x_1, \ldots, x_n\}$ and $\sigma(x) = x$ for $x \notin \{x_1, \ldots, x_n\}$. The application $\sigma(t)$ may be also written as $t\sigma$. If $t\sigma$ is a ground term (containing no variables), it is a *ground instance* of t. We denote the *most general unifier* of terms s and t by $mgu(s, t)$.

A substitution σ that replaces distinct variables by distinct variables (i.e. σ is injective and $x\sigma$ is a variable for every x) is called a *renaming*. If $s \equiv t\sigma$ for some renaming substitution σ, then we say that s is a *variant* of t, and write $s \cong t$.

Let $\Box$ be an extra constant called a *hole*. A *context* C is a term in $T(\Sigma \cup \{\Box\}, V)$. If C is a context with n occurrences of holes and $t_1, \ldots, t_n$ are terms, then $C[t_1, \ldots, t_n]$ is the result of replacing the holes by $t_1, \ldots, t_n$ from left to right. The *empty* context consists of only a single hole.

A *term rewriting system* (TRS), denoted by $\mathcal{R}$, is a set of rewrite rules of the form $l \to r$ consisting of two terms: the left-hand side l and the right-hand side r. The *reduction relation* $\to_{\mathcal{R}}$ induced on $T(\Sigma, V)$ by $\mathcal{R}$ is defined as $s \to_{\mathcal{R}} t$ iff there exists a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s \equiv C[l\sigma]$ and $t \equiv C[r\sigma]$. A term s is *reducible* if there is a term t such that $s \to_{\mathcal{R}} t$; otherwise, it is a *normal form*. The reflexive, symmetric, transitive closure of $\to_{\mathcal{R}}$ is denoted by $\leftrightarrow^*_{\mathcal{R}}$. Two terms s, t in TRS $\mathcal{R}$ are *joinable* (notation $s \downarrow t$,), if there exists a term v such that $s \to^*_{\mathcal{R}} v$ and $t \to^*_{\mathcal{R}} v$, where $\to^*_{\mathcal{R}}$ is the reflexive transitive closure of $\to_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *confluent* iff for all terms $s, t, u \in T(\Sigma, V)$, $u \to^*_{\mathcal{R}} s$ and $u \to^*_{\mathcal{R}} t$ implies $s \downarrow t$.

A *reduction order* $\succ$ on $T(\Sigma, V)$ is a strict partial order that is well-founded and closed under substitution and context. It is well-known that a TRS $\mathcal{R}$ is *terminating* if and only if there exists a reduction order in which the left-hand side is greater than the corresponding right-hand side for all rewrite rules of $\mathcal{R}$.

2.2 Term Rewriting Induction

We fix some definitions and notations on term rewriting induction as follows.

The set of all *defined symbols* of $\mathcal{R}$ is defined as $D_{\mathcal{R}} = \{root(l) \mid l \to r \in \mathcal{R}\}$, where the *root symbol* of a term $s \equiv f(s_1, \ldots, s_n)$ is f, denoted by $root(s)$.

The function symbols other than defined symbols are *constructors*. A term consisting of only constructors and variables is a *constructor term*.

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 a term t by $\mathcal{B}(t)$.

A TRS $\mathcal{R}$ is *ground-reducible* (also called quasi-reducible) if every ground basic term is reducible in $\mathcal{R}$. Plaisted [23] proved that ground-reducibility is decidable.

An *equation* is expressed in the form $s = t$. We do not distinguish between $s = t$ and $t = s$. 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 $\mathcal{R}$ (i.e., $s\sigma \leftrightarrow^*_{\mathcal{R}} t\sigma$).

The term rewriting induction (RI) [1], [25] works on two sets $\langle \mathcal{E}, \mathcal{H} \rangle$, where $\mathcal{E}$ stands for the *conjectures* containing the equations to be proved, while $\mathcal{H}$ indicates the inductive *hypotheses* generated during the inferences. Given as input a ground-reducible and convergent (terminating and confluent) TRS $\mathcal{R}$, a reduction order $\succ$ covering $\mathcal{R}$, and a set $\mathcal{E}_0$ of target inductive theorems or related lemmas, the RI theorem prover starts from $\langle \mathcal{E}_0, \mathcal{H}_0 \rangle$, where $\mathcal{H}_0 = \emptyset$, and generates a derivation sequence $\langle \mathcal{E}_0, \mathcal{H}_0 \rangle \vdash \langle \mathcal{E}_1, \mathcal{H}_1 \rangle \vdash \cdots$ until it (hopefully) stops with success at $\langle \mathcal{E}_f, \mathcal{H}_f \rangle$ such that $\mathcal{E}_f = \emptyset$. The inference rules of RI are summarized as follows:

DELETE: $\langle \mathcal{E} \cup \{s = s\}, \mathcal{H} \rangle \vdash \langle \mathcal{E}, \mathcal{H} \rangle$.

SIMPLIFY: $\langle \mathcal{E} \cup \{s = t\}, \mathcal{H} \rangle \vdash \langle \mathcal{E} \cup \{s' = t\}, \mathcal{H} \rangle$
if $s \to_{\mathcal{R} \cup \mathcal{H}} s'$.

EXPAND: $\langle \mathcal{E} \cup \{s = t\}, \mathcal{H} \rangle \vdash$
$\langle \mathcal{E} \cup \mathrm{Expd}_u(s, t), \mathcal{H} \cup \{s \to t\} \rangle$
if $u \in \mathcal{B}(s)$ and $s \succ t$,

where:

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

POSTULATE: $\langle \mathcal{E}, \mathcal{H} \rangle \vdash \langle \mathcal{E} \cup \mathcal{E}', \mathcal{H} \rangle$.

The DELETE rule removes meaningless conjectures. The SIMPLIFY rule rewrites a conjecture in $\mathcal{E}$ by applying a rewrite rule taken from $\mathcal{R} \cup \mathcal{H}$.

The EXPAND rule generates conjectures and hypotheses from the current conjectures and $\mathcal{R}$. The new conjectures are generated by the function $\mathrm{Expd}_u(s, t)$, which overlaps a basic subterm u of s with each basic left-hand side of rewrite rules $l \to r$ in $\mathcal{R}$. Those conjectures will become the subgoals of the proof for the original conjecture $s = t$, while the original conjecture is transformed into an inductive hypothesis $s \to t$ usable in the succeeding inferences.

The POSTULATE rule adds a set of equations $\mathcal{E}'$ to $\mathcal{E}$. Logically speaking, the equations in $\mathcal{E}'$ can be any equations, but in practice, they should be the *lemmas* that will be necessary or useful for leading the theorem prover to success. Generally, such lemmas should be added manually by highly experienced users' intuitions or generated mechanically by some heuristic algorithms.

In general, it is not straightforward to provide a suitable reduction order and to 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 follows.

EXPAND: $\langle \mathcal{E} \cup \{s = t\}, \mathcal{H} \rangle \vdash$
$\langle \mathcal{E} \cup \mathrm{Expd}_u(s, t), \mathcal{H} \cup \{s \to t\} \rangle$
if $u \in \mathcal{B}(s)$ and
$\mathcal{R} \cup \mathcal{H} \cup \{s \to t\}$ terminates.

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.

3. Divergence Detection

In this section, we take simple examples for illustrating a successful proof and a divergence case in RI.

3.1 A Simple Successful Proof

Consider the following axioms given as a TRS $\mathcal{R}_1$ defining the binary function *append* (@ for short) on lists constructed from the *cons* (: for short) operator, where the constant *nil* denotes an empty list.

$$\mathcal{R}_1 = \left\{ \begin{array}{ll} nil@xs \to xs, & (1) \\ (x : ys)@zs \to x : (ys@zs) & (2) \end{array} \right\}.$$

The target inductive theorem denoted by $\mathcal{T}_1$ is the associativity of *append*.

$$\mathcal{T}_1 : (xs@ys)@zs = xs@(ys@zs). \quad (3)$$

With a reduction order $\succ$ such as the lexicographic path order based on a precedence @ > : , the derivation starts from $\langle \mathcal{E}_0, \mathcal{H}_0 \rangle$, where $\mathcal{E}_0$ consists of the target theorem (3) and $\mathcal{H}_0$ is the empty set. The first step of the derivation is conducted by the EXPAND rule applied on the basic subterm $u \equiv xs@ys$ of (3) to get $\langle \mathcal{E}_1, \mathcal{H}_1 \rangle$, where $\mathcal{E}_1$ consists of the two equations

$$xs@zs1 = nil@(xs@zs1),$$
$$(x : (ys@zs))@zs1 = (x : ys)@(zs@zs1),$$

created by the expand function, and $\mathcal{H}_1$ consists of a single rewrite rule

$$(xs@ys)@zs \to xs@(ys@zs)$$

created by orienting the Eq. (3). To be more specific, the function $\mathrm{Expd}_u(s, t)$ was invoked with $s \equiv (xs@ys)@zs$, $t \equiv xs@(ys@zs)$ and $C = \Box@zs$. The first and the second equations of $\mathcal{E}_1$ were obtained by overlapping u with the left-hand sides of (1) and (2), respectively. Then, several SIMPLIFY rules were applied to $\langle \mathcal{E}_1, \mathcal{H}_1 \rangle$ for normalization until no more applications were possible. As the result, we obtain $\langle \mathcal{E}_2, \mathcal{H}_2 \rangle$, where $\mathcal{E}_2$ consists of two equations

$$xs@zs1 = xs@zs1,$$
$$x : (ys@(zs@zs1)) = x : (ys@(zs@zs1)),$$

and $\mathcal{H}_2 = \mathcal{H}_1$. After that, the DELETE rule was invoked twice to remove the self-evident conjectures from $\mathcal{E}_2$ to get $\langle \mathcal{E}_3, \mathcal{H}_3 \rangle$, where $\mathcal{E}_3 = \{\}$, $\mathcal{H}_3 = \mathcal{H}_2$. Therefore, the proof succeeds, because $\mathcal{E}_3$ is empty.

3.2 A Divergence Case

Execution of automated RI theorem provers may *diverge* due to the accumulation of the generated "unprovable" conjectures. We illustrate such a case in the following:

Example 3.1:

$$\mathcal{R}_2 = \left\{ \begin{array}{ll} nil@xs \to xs, & \\ (x : ys)@zs \to x : (ys@zs), & \\ r(nil) \to nil, & (4) \\ r(x : xs) \to r(xs)@(x : nil) & (5) \end{array} \right\}.$$

Two rewrite rules (4), (5) were added to $\mathcal{R}_1$ in this problem, where the new function r defines the *reverse* operation on a list. The target inductive theorem is

$$\mathcal{T}_2 : r(r(xs)) = xs.$$

The derivation starts from applying the EXPAND rule to get $\langle \mathcal{E}_1, \mathcal{H}_1 \rangle$, where

$$\mathcal{E}_1 = \left\{ \begin{array}{ll} r(nil) = nil, & (6) \\ r(r(xs)@(x : nil)) = x : xs & (7) \end{array} \right\}$$

and $\mathcal{H}_1 = \{r(r(xs)) \to xs\}$. Obviously, (6) is normalized by (4) to get the equation $nil = nil$ to be removed by DELETE. However, since no rewrite rule of $\mathcal{R}_2 \cup \mathcal{H}_1$ can rewrite (7), it is expanded, and then the derivation goes to $\langle \mathcal{E}_2, \mathcal{H}_2 \rangle$, where

$$\mathcal{E}_2 = \left\{ \begin{array}{ll} r(nil@(x1 : nil)) = x1 : nil, & (8) \\ r((r(xs)@(x : nil))@(x1 : nil)) = x1 : (x : xs) & (9) \end{array} \right\},$$

$$\mathcal{H}_2 = \left\{ \begin{array}{ll} r(r(xs)) \to xs, & \\ r(r(xs)@(x : nil)) \to x : xs & (10) \end{array} \right\}.$$

Note that (7) was turned into (10), when (8) and (9) were generated by EXPAND. The good thing is that (8) will be simplified and removed after several steps of rewriting. However, the bad thing is that (9) still cannot be simplified. The derivation continues in this way for several steps before getting to $\langle \mathcal{E}_3, \mathcal{H}_3 \rangle$, where

$$\mathcal{E}_3 = \left\{ \begin{array}{l} r((nil@(x1 : nil))@(x2 : nil)) = x2 : (x1 : nil), \\ r(((r(xs)@(x : nil))@(x1 : nil))@(x2 : nil)) \\ \quad = x2 : (x1 : (x : xs)) \end{array} \right\},$$

$$\mathcal{H}_3 = \left\{ \begin{array}{ll} r(r(xs)) \to xs, & (11) \\ r(r(xs)@(x : nil)) \to x : xs, & (12) \\ r((r(xs)@(x : nil))@(x1 : nil)) & \\ \quad \to x1 : (x : xs) & \end{array} \right\}.$$

Clearly, there exists a regular pattern of growth in the accumulation of hypotheses. In fact, this process will continue indefinitely and generate an infinite set of hypotheses, meaning that this derivation is diverging.

It is known that in this case we can suppress the divergence by using the POSTULATE rule to provide the following equations as conjectures (becoming lemmas when proved):

$$r(xs@ys) = r(ys)@r(xs), \quad (13)$$

$$(xs@ys)@zs = xs@(ys@zs). \quad (14)$$

To be more specific, we can use the POSTULATE rule to put conjecture (13) into the set of equations, hoping that it may be useful for ending the divergence. However, we see that the proof of (13) itself requires a new postulation, because it will turn out that the derivation for proving (13) causes another divergence. To solve this problem, we can use the POSTULATE rule again to additionally put conjecture (14) into the set. Fortunately, conjecture (14) can be proved without any help of extra lemmas, and thus it is established as a lemma. It will turn out that lemma (14) is helpful for ending the divergence generated when trying to prove (13), and thus (13) is established as a lemma. Now lemma (13) can be used to end the divergence mentioned in the previous paragraph to finally establish the target $\mathcal{T}_2$ as a theorem. Of course, the real problem here is how we can come up with (13), (14), or any other conjectures leading our proof to success. This is the main topic of this paper.

3.3 Term Annotation and Difference Match

As discussed in [6], [10], [16], a crucial point in proving inductive theorems is to transform the induction conclusion to enable the use of the induction hypothesis. This can be often done by controlling the deduction so that it will remove (or "ripple out") the "difference" between the conclusion and the hypothesis. The difference is also called the "wave-front". In the following, we present some basic definitions and notations related to this subject. (Actually, we present them in a formal way suitable for the term rewriting community.)

In [6], [10], a *wave-front* is described as a term t with a proper subterm t' deleted. The deleted subterm may itself contain wave-fronts. This means that we can identify the innermost deleted subterms. A wave-front is often represented by an annotation which encloses t in a box and underlines the deleted subterm t'. (Note, however, that in the theory of the standard term rewriting, a "term" with a subterm deleted cannot be a term!) In this paper, we formally define the notion of term annotation and wave-fronts as follows.

Definition 3.2: Let us call the elements of $T(\Sigma, V)$ and $T(\Sigma \cup \{\Box\}, V)$ the *ordinary* terms and contexts, respectively. Let **box** and **ul** be the distinguished unary function symbols not contained in the signature Σ at hand. Then an *annotated term* is defined inductively as follows.

- If C is an ordinary context (which can be empty), $D_1, \ldots, D_n (n > 0)$ are nonempty ordinary contexts with a single hole, and $s_1, \ldots, s_n$ are either ordinary

or annotated terms, then

$$C[\mathbf{box}(D_1[\mathbf{ul}(s_1)]), \ldots, \mathbf{box}(D_n[\mathbf{ul}(s_n)])],$$

displayed with *annotations* as

$$C[\boxed{D_1[\underline{s_1}]}, \ldots, \boxed{D_n[\underline{s_n}]}],$$

is an annotated term.

Each context $D_i (i \in \{1 \ldots n\})$ is called a *wave-front*, and its hole is called a *wave-hole*.

Definition 3.3: A pair of annotated terms u and v, written $u \to v$, is an *annotated rule*.

Definition 3.4: Let w be an ordinary or annotated term. Then its *body*, $body(w)$, is defined as follows.
$body(w) =$

$$\begin{cases} w, & \text{if } w \in T(\Sigma, V), \\ C[D_1[body(s_1)], \ldots, D_n[body(s_n)]], & \text{if } w \equiv C[\boxed{D_1[\underline{s_1}]}, \ldots, \boxed{D_n[\underline{s_n}]}]. \end{cases}$$

The *skeleton* of w, $skel(w)$, is defined as follows.
$skel(w) =$

$$\begin{cases} w, & \text{if } w \in T(\Sigma, V), \\ C[skel(s_1), \ldots, skel(s_n)], & \text{if } w \equiv C[\boxed{D_1[\underline{s_1}]}, \ldots, \boxed{D_n[\underline{s_n}]}]. \end{cases}$$

Intuitively, $body(w)$ is an ordinary term obtained by canceling all annotations from w; $skel(w)$ is obtained by erasing all wave-fronts from the body. The wave-fronts are also regarded as the *difference* between the body and the skeleton. (In [6], the *body* function is called *erase* and its return value the body.)

For instance, consider an *annotated term*

$$w \equiv r(\boxed{\underline{r(xs)}@(x : nil)}).$$

Then its *body* and *skeleton* are as follows:

$$body(w) = r(r(xs)@(x : nil)),$$
$$skel(w) = r(r(xs)).$$

Note that in this example, $t \equiv skel(w)$ and $s \equiv body(w)$ are the left-hand side of (11) and (12), respectively. Intuitively, the symbols removed from s to get t constitute the *difference* between s and t. However, given two terms s and t, the difference between them is not unique in general. Formally, the *difference match* function $dm(s, t)$ in Definition 3.5 adapted from [6] computes all the *differences* $\langle w, \delta \rangle$ such that

- $s \equiv body(w)$ (w is obtained by annotating s)
- $t \equiv skel(w)\delta$ (the annotation in w defines the difference between s and t)
- δ is a variable renaming substitution with domain $\mathcal{V}(skel(w))$.

Definition 3.5: Let $x, y \in V$, $s \equiv f(s_1, \ldots, s_n)$, $s' \equiv f(s'_1, \ldots, s'_n)$ and $t \equiv g(t_1, \ldots, t_m)$ where $f \neq g$.

$$dm(x, y) = \{\langle x, \{x \mapsto y\}\rangle\}$$
$$dm(x, t) = \{\}$$
$$dm(s, s') = \{\langle f(w_1, \ldots, w_n), \bigcup_i \delta_i \rangle \mid \langle w_i, \delta_i \rangle \in dm(s_i, s'_i) \text{ for all } 1 \le i \le n \text{ and } \delta_1 \ldots \delta_n \text{ are mutually compatible}\} \cup \bigcup_i \{\langle \boxed{f(s_1, \ldots, s_{i-1}, \underline{w_i}, s_{i+1}, \ldots, s_n)}, \delta \rangle \mid \langle w_i, \delta \rangle \in dm(s_i, s')\}$$
$$dm(s, t) = \bigcup_i \{\langle \boxed{f(s_1, \ldots, s_{i-1}, \underline{w_i}, s_{i+1}, \ldots, s_n)}, \delta \rangle \mid \langle w_i, \delta \rangle \in dm(s_i, t)\}$$

where two substitutions σ_1 and σ_2 are *compatible* if $\sigma_1(x) = \sigma_2(x)$ for every $x \in Dom(\sigma_1) \cap Dom(\sigma_2)$. For two compatible substitutions σ_1 and σ_2, the union $\sigma_1 \cup \sigma_2$ of them can be uniquely defined as the substitution σ satisfying $Dom(\sigma) = Dom(\sigma_1) \cup Dom(\sigma_2)$, $\sigma(x) = \sigma_1(x)$ for $x \in Dom(\sigma_1)$, and $\sigma(x) = \sigma_2(x)$ for $x \in Dom(\sigma_2)$.

In the sequel, each element $\langle w, \delta \rangle$ returned from the dm function will be simply displayed as an annotated term $w\delta$. For example, an element $\langle \boxed{x1 : \underline{xs1}}, \{xs1 \mapsto xs\} \rangle$ returned from $dm((x1 : xs1), xs)$ will be displayed as $\boxed{x1 : \underline{xs}}$.

As commented in [8], annotated terms can be considered as decorated trees where the skeleton is represented as a tree and each wave-front as a box decorating a node. The wave-fronts (boxes) often move up or down through the skeleton tree to make different annotations during difference matching. In such cases, we follow the heuristics described in [8], [31] etc. by only considering the *maximal* difference match in which wave-fronts are as high as possible in the skeleton tree.

Example 3.6: Let

$$s \equiv x2 : (y2 : ys2),$$
$$t \equiv x1 : xs1.$$

Then

$$dm(s, t) = \left\{ \begin{array}{l} \boxed{x2 : \underline{(x1 : xs1)}}, \\ x1 : \boxed{\underline{(xs1 : ys2)}}, \\ x1 : \boxed{(y2 : \underline{xs1})} \end{array} \right\}.$$

Clearly, the first element is maximal, as the box is attached at the highest position (root) of the tree for the skeleton $x1 : xs1$.

The original definition of the difference matching algorithm in [6] is presented in a logic programming style where the predicate $dm(s, t, w, \delta)$ with inputs s and t supplies appropriate outputs w and δ satisfying the specified relationship among the four arguments, when it succeeds. Repeating this predicate call, one can collect all of such outputs. Our dm function is its functional version that returns those outputs as a set of $\langle w, \delta \rangle$ pairs. It was commented in [31] that using ground difference matching with renaming of variables seemed to be sufficient for identifying accumulating term structure. Therefore, we have slightly restricted the general definition of the original version according to its

normal usage in the inductive theorem proving context (as is implicit in [31]). More precisely, our version restricts the substitution δ to a variable renaming substitution rather than an arbitrary substitution. Though there exists a fast polynomial algorithm for difference matching using dynamic programming [7], in this paper, we introduced the concise specification based on [6].

Note that this restricted version of the *dm* function is clearly related to the homeomorphic embedding [5] used in the theory of the simplification ordering, because the set $dm(s,t)$ contains an element $\langle w, \delta\rangle$ if and only if t is homeomorphically embedded in $s\delta$. The exact information on how to embed is encoded in w, an annotation to s, so that we can get t from $s\delta$ by removing all the symbol occurrences other than those in $skel(w\delta)$.

3.4 Automated Lemma Postulation

In [10], an effective tactic named *rippling* was proposed by Bundy et al. Walsh [31] combined this technique with difference matching [6] to overcome the difficulty faced when the induction theorem provers generate diverging sequences. Based on this technique, Shimazu, Aoto and Toyama [29] formalized an automated lemma postulation procedure in the framework of the rewriting induction as follows.

We first show a simplified version. Suppose that a sequence of hypotheses (which is seemingly diverging) contains the following rewrite rules.

$$C[s] \to t, \tag{19}$$
$$C[D[s]] \to F[t]. \tag{20}$$

Note that the difference matching between them gives an annotation to (20) as follows.

$$C[\boxed{D[\underline{s}]}] \to \boxed{F[\underline{t}]}.$$

Then the technique for lemma postulation may be applied in the three steps as follows.

The first step applies the rewrite rule (19) to (20) in reverse (i.e., $t \to C[s]$) to get an equation

$$C[D[s]] = F[C[s]].$$

Though in [29], this step is referred to as a rather general name, *modification*, in this paper, it will be called *joining*, as the resultant equation implies that $C[D[s]]$ and $F[C[s]]$ are joinable by (19) and (20) at $F[t]$.

The second step replaces occurrences of s (if it is a non-variable) with a fresh variable x to get a new conjecture:

$$C[D[x]] = F[C[x]].$$

This step has been called simply *generalization* elsewhere [29]. Here we call it *non-var generalization* to distinguish it from variable-renaming generalization introduced in the next section. Formally, an equation $s = t$ is a *non-var generalization* of an equation $s\sigma = t\sigma$, if $x\sigma$ is a non-variable for all $x \in Dom(\sigma)$.

This lemma postulation method is unsound, because the generated conjectures are not necessarily inductive theorems of $\mathcal{R}$ even if the target equation is actually an inductive theorem. Hence the third step *filtering* is invoked to check if there is a real possibility that it is actually an inductive theorem. To test the equality, the system substitutes randomly-generated ground terms to the variables in the equation before normalizing its left- and right-hand sides to see their joinability. (Since $\mathcal{R}$ is convergent, this test is decidable.) The conjecture which has led to a counterexample for the equality is filtered out. On the other hand, the survived conjecture is added (by the POSTULATE inference rule) to the conjecture set C as a candidate for lemma to be proved later in the process.

Note that if the conjecture is directed from left to right, it works as a rewrite rule $C[D[x]] \to F[C[x]]$ for removing (or "rippling out") the difference (or "wave-front") D from the left-hand side of (20) to get a term $F[C[s]]$. Since $F[C[s]]$ can be further rewritten by (19) to $F[t]$, the hypothesis (20) may be reduced to a trivial equation $F[t] = F[t]$.

This procedure is formally described as the following inference rule:

POSTULATE BY JOINING:

$$\langle \mathcal{E}, \mathcal{H}\rangle \vdash \langle \mathcal{E} \cup \{C[D[x]] = F[C[x]]\}, \mathcal{H}\rangle$$
$$\text{if } \{C[s] \to t, C[D[s]] \to F[t]\} \subseteq \mathcal{H},$$

where x is a fresh variable.

Note that introducing a renaming substitution, the procedure in [29] is presented in a slightly more general form. Here, however, we adopted the presentation without explicit renaming, following the convention where variables in equations may be renamed whenever appropriate.

Example 3.7: Suppose we have the following two hypotheses annotated by difference-matching (12) with (11):

$$r(r(xs)) \to xs,$$
$$r(\boxed{\underline{r(xs)}@(x1 : nil)}) \to \boxed{x1 : \underline{xs}}.$$

Clearly, we have $C = r(\Box)$, $D = \Box@(x1 : nil)$, $F = x1 : \Box$, $s = r(xs)$, $t = xs$, and obtain

$$r(r(xs)@(x1 : nil)) = x1 : r(r(xs))$$

by joining. Then non-var generalization is applied to get a conjecture

$$r(ys@(x : nil)) = x : r(ys).$$

It turns out that this conjecture is in fact a lemma that is sufficient to lead to a proof for the target theorem, $\mathcal{T}_2$, shown in Example 3.1.

4. Peripheral Sculpture

In this section, we introduce *zipped difference* to analyze the potential divergence patterns and then present a lemma

postulation method called *peripheral sculpture* based on divergence detection. We introduce the basic definitions and inference rules.

4.1 Zipped Difference

Definition 4.1: Let $w \equiv C[\boxed{D_1[\underline{s_1}]}, \ldots, \boxed{D_n[\underline{s_n}]}]$ be an annotated term. Then its *peripheral part*, $peri(w)$, and *calm part*, $calm(w)$, are the unannotated contexts defined respectively as follows.

$$peri(w) = C,$$
$$calm(w) = C[D_1, \ldots, D_n].$$

The variables occurring in $peri(w)$ are *peripheral*.

Example 4.2: Consider an annotated term

$$w \equiv (y@\boxed{(x : \underline{y})})@\boxed{(\underline{x} : y)}.$$

Then y is a peripheral variable but x is not.

Definition 4.3: Given a (finite or infinite) sequence of terms $S = \{s_i \mid i = 0, 1, \ldots\}$, a *zipped difference* for S is a sequence $Z = \{w_i \mid i = 0, 1, \ldots\}$ of annotated terms such that

$$\forall i, \exists \delta_i : \langle w_i, \delta_i \rangle \in dm(s_{i+1}, s_i) \quad \text{and}$$
$$\forall i, j : calm(w_i) \cong calm(w_j).$$

Recall that $s \cong t$ means s is a variant of t. An element $w_i \in Z$ consisting of the smallest number of occurrences of function symbols and variables is *minimal*. Definition 4.3 was inspired by Walsh [31].

Example 4.4: Let

$$S = \left\{ \begin{array}{l} s_0 \equiv xs, \\ s_1 \equiv x1 : xs1, \\ s_2 \equiv x2 : (y2 : ys2) \end{array} \right\}.$$

Then $dm(s_1, s_0)$ contains two differences

$$\{\boxed{x1 : \underline{xs}}, \boxed{\underline{xs} : xs1}\},$$

and $dm(s_2, s_1)$ contains the maximal difference

$$\boxed{x2 : \underline{(x1 : xs1)}}$$

and two non-maximal ones. The zipped difference is

$$Z = \{\boxed{x1 : \underline{xs1}}, \boxed{x2 : \underline{(y2 : ys2)}}\},$$

where the minimal element is $\boxed{x1 : \underline{xs1}}$. However, $\boxed{\underline{xs} : xs1}$ cannot be an element of a zipped difference, and hence only Z is the correct zipped difference.

Example 4.5: We apply Definition 4.3 to a sequence of hypotheses $\mathcal{H}$ as well, by regarding $\rightarrow$ as a binary function symbol with infix notation. Consider the following sequence $\mathcal{H}_3$ from Sect. 3.2 (with variable renaming):

$$\mathcal{H}_3 = \left\{ \begin{array}{l} h_0 \equiv r(r(xs)) \rightarrow xs, \\ h_1 \equiv r(r(xs1)@(x1 : nil)) \rightarrow x1 : xs1, \\ h_2 \equiv r((r(ys2)@(y2 : nil))@(x2 : nil)) \\ \quad \rightarrow x2 : (y2 : ys2) \end{array} \right\}.$$

Then $dm(h_1, h_0)$ contains two differences

$$\left\{ \begin{array}{l} r(\boxed{\underline{r(xs)}@(x1 : nil)}) \rightarrow \boxed{x1 : \underline{xs}}, \\ r(\boxed{\underline{r(xs)}@(x1 : nil)}) \rightarrow \boxed{\underline{xs} : xs1} \end{array} \right\},$$

and $dm(h_2, h_1)$ contains the maximal difference

$$r(\boxed{\underline{(r(xs1)@(x1 : nil))}@(x2 : nil)}) \rightarrow \boxed{x2 : \underline{(x1 : xs1)}}$$

and four non-maximal differences. The following is the the zipped difference:

$$Z' = \left\{ \begin{array}{l} r(\boxed{\underline{r(xs1)}@(x1 : nil)}) \rightarrow \boxed{x1 : \underline{xs1}}, \\ r(\boxed{\underline{(r(ys2)@(y2 : nil))}@(x2 : nil)}) \\ \quad \rightarrow \boxed{x2 : \underline{(y2 : ys2)}} \end{array} \right\}.$$

Intuitively, given a potentially diverging sequence S, its zipped difference postulates a divergence pattern of S, representing a common annotation pattern for the differences between successive terms. The calm parts of its elements represent a stable context and the remaining parts in the wave-holes are considered to represent a growing pattern of the divergence.

4.2 Lemma Postulation by Peripheral Sculpture

Given a sequence of hypotheses $\mathcal{H}$, the first two rules can be represented as follows when a zipped difference Z for $\mathcal{H}$ exists (where the variables may be renamed in the second rule.):

$$C[s_1, \ldots, s_n] \rightarrow C'[t_1, \ldots, t_m], \tag{22}$$
$$C[\boxed{D_1[\underline{s_1}]}, \ldots, \boxed{D_n[\underline{s_n}]}] \rightarrow C'[\boxed{D'_1[\underline{t_1}]}, \ldots, \boxed{D'_m[\underline{t_m}]}]. \tag{23}$$

In this subsection, we present a lemma postulation method applicable in this situation.

Definition 4.6: Let $\nu_1, \ldots, \nu_k$ be fresh variables, and consider a renaming substitution

$$\delta = \{x_1 \mapsto \nu_1, \ldots, x_k \mapsto \nu_k\},$$

with the domain

$$\{x_1, \ldots, x_k\} = \mathcal{V}(C) \cap \mathcal{V}(C') \cap [(\cup_i \mathcal{V}(D_i[s_i]) \cup (\cup_i \mathcal{V}(D'_i[t_i])].$$

Then

$$C\delta[s_1, \ldots, s_n] \to C'\delta[t_1, \ldots, t_m]$$

is called a *peripheral sculpture* of Z.

Note that every variable in the domain should occur in every one of the three parts: C, C', and the remaining part. We rename the occurrences only in C and C' with the remaining part untouched.

Example 4.7: Suppose we have

$$w + (x + (x + (x + 0))) \to w + (x + (x + x)),$$
$$w + (x + (x + \boxed{s(\underline{x + 0})})) \to w + (x + (x + \boxed{s(\underline{x})})),$$

and

$$Z = \{w + (x + (x + \boxed{s(\underline{x + 0})})) \to w + (x + (x + \boxed{s(\underline{x})}))\}.$$

Then its peripheral sculpture is

$$w + (v + (v + (x + 0))) \to w + (v + (v + x)).$$

Given a sequence $\mathcal{H}$ of hypotheses, the following non-deterministic procedure tries to generate a conjecture to be proved as a lemma for the target theorem.

1. Compute a zipped difference Z for $\mathcal{H}$.
2. If Z exists, compute a peripheral sculpture $\mathcal{S}$ of $\mathcal{H}$ with respect to Z.
3. If $\mathcal{S}$ exists, send it to the filtering process to see its possibility of being a theorem.
4. If $\mathcal{S}$ has survived, return $\mathcal{S}$ itself or its non-var generalization as a conjecture.

Step 2 is the key to this method. In practice, the two rules (22) and (23) are the first (minimal) and the second (second-minimal) elements of $\mathcal{H}$ and the remaining elements are just used for checking the existence of Z. If Z does not exist, the procedure is aborted. If there are more than one Z's, every Z is considered non-deterministically.

Note that the longer sequence we have for $\mathcal{H}$, the more reliable conjecture we get, because Z for a longer $\mathcal{H}$ shows us a longer diverging pattern. On the other hand, longer $\mathcal{H}$ may lead to less efficiency caused by the delay of useful lemma generation. Based on experience, the length 3 is recommended in [31].

The procedure can be formally described as an inference rule as follows:

Postulate by Peripheral Sculpture:

$$\langle \mathcal{E}, \mathcal{H} \rangle \vdash \langle \mathcal{E} \cup \{p = q\}, \mathcal{H} \rangle$$

if

$$\text{-} \left\{ \begin{array}{l} C[s_1, \ldots, s_n] \to C'[t_1, \ldots, t_m], \\ C[\boxed{D_1[\underline{s_1}]} \ldots \boxed{D_n[\underline{s_n}]}] \to C'[\boxed{D'_1[\underline{t_1}]} \ldots \boxed{D'_m[\underline{t_m}]}] \end{array} \right\} \subseteq \mathcal{H},$$

- there exists a zipped difference Z for $\mathcal{H}$, and

- $p \to q$ is a peripheral sculpture of Z or its non-var generalization.

Note that the filtering process in Step 3 is not involved in the inference rule. This is because the filtering does not affect the exact form of the generated conjecture. We regard the filtering as a part of the control strategy of the theorem prover which uses it as a deciding factor for applying the postulation rule.

Example 4.8: Consider the following TRS $\mathcal{R}_3$.

$$\mathcal{R}_3 = \left\{ \begin{array}{c} 0 + y \to y, \\ s(x) + y \to s(x + y), \\ 0 * y \to 0, \\ s(x) * y \to y + (x * y) \end{array} \right\},$$

where $+$ and $*$ are recursively defined as the algebraic *addition* and *multiplication*. The target theorem is:

$$\mathcal{T}_3 : s(s(0)) * x = x + x.$$

Starting with $\mathcal{T}_3$, the procedure diverges by constantly expanding new conjectures which cannot be simplified to a trivial equation. The difference matching procedure annotates the diverging sequence $\mathcal{H}$ as follows.

$$x + (x + 0) \to x + x,$$
$$x + \boxed{s(\underline{x + 0})} \to x + \boxed{s(\underline{x})},$$
$$x + \boxed{s(\underline{s(x + 0)})} \to x + \boxed{s(\underline{s(x)})}.$$

In Step 1, we have a zipped difference

$$Z = \left\{ \begin{array}{l} x + \boxed{s(\underline{x + 0})} \to x + \boxed{s(\underline{x})}, \\ x + \boxed{s(\underline{s(x + 0)})} \to x + \boxed{s(\underline{s(x)})} \end{array} \right\}.$$

In Step 2, we have a peripheral sculpture $\mathcal{S}$:

$$v + (x + 0) \to v + x.$$

After Steps 3 and 4, we get a new conjecture:

$$v + (x + 0) = v + x.$$

It will turn out that this conjecture is actually a lemma to resolve the divergence to complete the proof of $\mathcal{T}_3$.

The next example demonstrates that postulate by peripheral sculpture may be applicable to more complex diverging patterns involving 'parallel' and 'nested' differences.

Example 4.9: With $\mathcal{R}_3$ in the previous example, our target theorem here is

$$\mathcal{T}_4 : s(s(s(s(0)))) * x = s(s(0)) * (s(s(0)) * x).$$

The diverging sequence is annotated as follows.

$$(x + (x + 0)) + ((x + (x + 0)) + 0)$$
$$\to x + (x + (x + (x + 0))),$$
$$(x + \boxed{s(\underline{x + 0})}) + \boxed{s((x + \boxed{s(\underline{x + 0})}) + 0)}$$
$$\to x + \boxed{s(x + \boxed{s(x + \boxed{s(\underline{x + 0})})})},$$
$$(x + \boxed{s(s(\underline{x + 0}))}) + \boxed{s(s((x + \boxed{s(s(\underline{x + 0}))}) + 0))}$$
$$\to x + \boxed{s(s(x + \boxed{s(s(x + \boxed{s(s(\underline{x + 0}))})}))})}.$$

In Step 1, we have a zipped difference $Z =$

$$\left\{\begin{array}{l} (x + \boxed{s(\underline{x + 0})}) + \boxed{s((x + \boxed{s(\underline{x + 0})}) + 0)} \\ \quad \to x + \boxed{s(x + \boxed{s(x + \boxed{s(\underline{x + 0})})})}, \\ (x + \boxed{s(s(\underline{x + 0}))}) + \boxed{s(s((x + \boxed{s(s(\underline{x + 0}))}) + 0))} \\ \to x + \boxed{s(s(x + \boxed{s(s(x + \boxed{s(s(\underline{x + 0}))})}))})} \end{array}\right\}.$$

In Step 2, we have a peripheral sculpture $\mathcal{S}$:

$$(\nu + (x + 0)) + ((x + (x + 0)) + 0)$$
$$\to \nu + (x + (x + (x + 0))).$$

After Step 3, we have three subterms

$$\{x + (x + 0), x + 0, 0\}$$

for non-var generalization procedure. By generalizing them, a conjecture (corresponding to $x + (x + 0)$) and an equivalent of $\mathcal{S}$ passed the random testing, where the former one

$$(\nu + (x + 0)) + (y + 0) = \nu + (x + y)$$

leads to a successful proof. Note that this conjecture is a consequence of the right identity and the associativity of $+$, but we are not given the corresponding axioms or lemmas.

5. Multi-Context Postulation

In this section, we introduce the multi-context reasoning framework into which our lemma postulation method is built and show how our lemma postulations can be incorporated in a multi-context reasoning system, particularly a *multi-context rewriting induction* (MRIt) system.

5.1 Multi-Context Rewriting Induction

Based on the ideas of *multi-completion* (MKB) [20], [27], [28] and *rewriting induction with termination checkers* (RIt) [2], the *multi-context rewriting induction* (MRIt) [17], [26] efficiently simulates RIt processes, each corresponding to a non-deterministic computation which has made a particular series of commitments at the choice points they encountered for various decisions.

In particular, reduction orders (in which way to orient an equation; implicitly inducing induction patterns based on Noetherian induction) can be selected dynamically by calling external, modern automated termination checkers more powerful than the classical, simply parameterized reduction orders (such as recursive path orders and polynomial orders), based on the work of [32]. Other choices include induction strategies (which variable to select for induction) and rewriting strategies (which rule to apply and which subterm to be applied to).

To distinguish processes, each process is represented by a sequence of natural numbers $a_1 a_2 \dots a_k$ (for some $k \geq 0$) called an *index*, when the i-th decision of this process was the choice No. a_i ($1 \leq i \leq k$). Thus the index can be interpreted as a position of a node at depth k in a search tree. In particular, the initial process (the root node) is represented by an empty sequence (denoted by ϵ). We do not distinguish between a process and its index.

In order to efficiently simulate a lot of closely-related inferences made in different processes, MRIt exploits the data structure called *nodes* and represent the state of the inference system by a set of nodes. The node is a tuple $\langle s : t, H_1, H_2, E\rangle$, where $s : t$ is an ordered pair of terms s and t, and H_1, H_2, E are subsets of process indices called *labels*. Intuitively, E represents all processes containing $s = t$ as a conjecture, and H_1 (resp. H_2) represents all processes containing $s \to t$ (resp. $t \to s$) as an inductive hypothesis. The set of possible indices I is infinite in MRIt. The node $\langle s : t, H_1, H_2, E\rangle$ is considered to be identical with $\langle t : s, H_2, H_1, E\rangle$.

Given the current set N of nodes, $\mathcal{E}[N, p]$ and $\mathcal{H}[N, p]$ defined below represent the current sets of conjectures (equations) and hypotheses (rules), respectively, held in the process p.

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

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

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

where $n = \langle s : t, H_1, H_2, E\rangle$. $\mathcal{E}[n, p]$ and $\mathcal{H}[n, p]$ are called $\mathcal{E}$*-projection* and $\mathcal{H}$*-projection* of n onto p, respectively.

Given the initial set of conjectures $\mathcal{E}_0$ and a ground-reducible and convergent TRS $\mathcal{R}$, MRIt starts with the initial set N_0 of nodes:

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

Note that $\langle \mathcal{E}[N_0, \epsilon], \mathcal{H}[N_0, \epsilon] \rangle = \langle \mathcal{E}_0, \emptyset \rangle$. The inference rules of MRIt are listed in Appendix A. A series of applications of those rules to the sets of nodes generates a derivation $N_0 \vdash N_1 \vdash \cdots \vdash N_c$. If $\mathcal{E}[N_c, p]$ is empty for some process p, the system concludes that all the initial conjectures $\mathcal{E}_0$ are inductive theorems of $\mathcal{R}$.

We elaborate on inference rules DELETE, FORK and EXPAND of MRIt to show how the multi-context reasoning works.

The DELETE rule of MRIt simulates its counterpart of RIt. That is, if a trivial conjecture appears in any process, it is removed.

DELETE: $N \cup \{\langle s : s, H_1, H_2, E \rangle\} \vdash N$.

Assume that process p_1 holds a trivial conjecture $s = s$ in the corresponding $\mathcal{E}_1$ and that process p_2 also holds $s = s$ in the corresponding $\mathcal{E}_2$. To remove the trivial conjectures, the DELETE of RIt is invoked in each process, p_1 and p_2. In MRIt, such manipulations are effectively done by simply removing one node $\langle s : s, H_1, H_2, \{p_1, p_2\} \rangle$ from the node set. Note that H_1, H_2 can be empty in the sense that the deletion happens in $\mathcal{E}$ in RIt.

Suppose that the process with an index $p = a_1 a_2 \ldots a_k$ have n possible choices. Then it will be forked into n different processes $p\,1, p\,2, \ldots, p\,n$, each taking care of one of the choices. In [26] this operation is simulated in the node structure by replacing the index p in every label of every node with those new n indices, and formalized as the FORK inference rule:

FORK: $N \vdash \psi_P(N)$.

To formally understand this inference rule, we need the following definitions and notations introduced in [26].

The basic *fork function over* a given set P of processes, denoted by $\psi_P : I \to \mathbb{P}(I)$, is defined as follows:

Definition 5.1:

$$\psi_P(p) = \begin{cases} \{p1, p2, \ldots, p\psi(p)\}, & \text{if } p \in P, \\ \{p\}, & \text{otherwise.} \end{cases}$$

where I is the set of all indices (processes), $\mathbb{P}(I)$ the power set of I, and $\psi(p)$ the number of processes that p is to be forked into.

The notation $\psi_P(N)$ used in FORK represents the set of nodes created from N by replacing all the processes p in P with $p\,1, p\,2, \ldots, p\,\psi(p)$.

The EXPAND rule of MRIt is the counterpart of that of RIt, playing the leading role in rewriting induction.

EXPAND: $N \cup \{\langle s : t, H_1, H_2, E \cup E' \rangle\} \vdash$
$N \cup \{\langle s : t, H_1 \cup E', H_2, E \rangle\}$
$\cup \{\langle s' : t', \emptyset, \emptyset, E' \rangle \mid s' = t' \in \mathrm{Expd}_u(s, t)\}$
if $E' \neq \emptyset, u \in \mathcal{B}(s)$ and
$\mathcal{H}[N, p] \cup \mathcal{R} \cup \{s \to t\}$ terminates for all $p \in E'$.

Focusing on a node $n = \langle s : t, H_1, H_2, E \cup E' \rangle$ and a basic term $u \in \mathcal{B}(s)$, this inference rule applies the EXPAND rule of RIt to all processes (E') that can orient $s = t$ from left to right. As a result, new nodes $\langle s' : t', \emptyset, \emptyset, E' \rangle$ are created for all conjectures $s' = t'$ generated by $\mathrm{Expd}_u(s, t)$. The labels of the original node n is modified so that the processes of E' moves from the third to the first label, meaning that in those processes the equation $s = t$ was oriented from left to right. Note that the choice of the direction of orientation and the choice of the basic subterm are two kinds of nondeterministic choices. In practice, therefore, the theorem prover should combine EXPAND operation with two types of FORK operations: One is to fork a process into two processes depending on the two possible direction of orientation, and another is to fork each of the resultant processes with k basic subterm choices into k processes. The formal treatment of this combination is presented in [26].

Let $\vdash^{=}_{RIt}$ be the reflexive closure of $\vdash_{RIt}$ (meaning $\vdash^{=}_{RIt}$ denotes either $=$ or $\vdash_{RIt}$). The following two propositions shown in [26] state the soundness of FORK and other inference rule of MRIt other than FORK.

Proposition 5.2: Let $N' = \psi_P(N)$ be the set of nodes obtained by applying FORK to N. Then $\langle \mathcal{E}[N, p], \mathcal{H}[N, p] \rangle = \langle \mathcal{E}[N', q], \mathcal{H}[N', q] \rangle$ for all $p \in I$ and $q \in \psi_P(p)$.
In other words, FORK have no effect on processes, only generating copies of some processes.

Proposition 5.3: Let N' be the set of nodes obtained by applying to N an inference rule of MRIt other than FORK. Then $\langle \mathcal{E}[N, p], \mathcal{H}[N, p] \rangle \vdash^{=}_{RIt} \langle \mathcal{E}[N', p], \mathcal{H}[N', p] \rangle$ for all $p \in I$.
In other words, the inference rules of MRIt other than FORK simulate an RIt inference in some processes and have no effect on other processes.

5.2 Postulation in Multi-Context System

In this subsection, we will extend MRIt to develop an inductive theorem prover *MRIt+* which combines 1) multi-context reasoning and 2) RIt with divergence-detection-based automated lemma postulation, where we replace the general POSTULATE rule of RIt with more specific inference rules for postulation: POSTULATE BY JOINING and POSTULATE BY PERIPHERAL SCULPTURE.

Suppose we have several processes $P = \{p_1, p_2, \ldots\}$ and a potentially diverging sequence $\{h_1 \equiv s_1 \to t_1, h_2 \equiv s_2 \to t_2, \ldots\}$ held by $p \in P$. Also suppose that there are k conjectures $\{l_i = r_i\}$ that can be added to $\langle \mathcal{E}, \mathcal{H} \rangle$ in p by applying either POSTULATE BY JOINING or POSTULATE BY PERIPHERAL SCULPTURE. To deal with these conjectures, we have process p fork into $k + 1$ processes where each process $p\,i (1 \leq i \leq k)$ holds one new conjecture $l_i = r_i$, respectively, and the remaining process $p\,(k + 1)$ continues without any of the newly generated conjectures. Such a process without postulation is necessary because divergence detection is "unsound", meaning that the postulated conjecture may be incorrect, causing a search in vain for its proof.

Note that the same inference (postulation) discussed in the previous paragraph can be made in other processes as

well containing $s_1 \to t_1$ and $s_2 \to t_2$ as hypotheses. Actually, this can be handled efficiently by the standard technique of multi-context equational reasoning as in the following inference rule.

MULTI-CONTEXT POSTULATE

$$
\begin{aligned}
&N \vdash \psi_P(N) \cup \{\langle l_i : r_i, \emptyset, \emptyset, P_i\rangle \mid 1 \le i \le k\}\\
&\text{if } \langle \mathcal{E}, \mathcal{H}\rangle \vdash_{RIt} \langle \mathcal{E} \cup \{l_i = r_i\}, \mathcal{H}\rangle\\
&\qquad \text{for some process } p \in P (1 \le i \le k)\\
&\text{where } n_1 = \langle s_1 : t_1, H_1, \ldots, \ldots\rangle \in N,\\
&\qquad\quad n_2 = \langle s_2 : t_2, H_2, \ldots, \ldots\rangle \in N,\\
&\qquad\quad P = H_1 \cap H_2,\\
&\qquad\quad P_i = \{p.i \mid p \in P\},\\
&\qquad\quad \mathcal{E} = \mathcal{E}[N, p],\\
&\qquad\quad \mathcal{H} = \mathcal{H}[N, p],\\
&\qquad\quad \psi(p) = k + 1 \text{ for all } p \in P,\\
&\text{and } \vdash_{RIt} \text{ denotes one-step derivation in RIt.}
\end{aligned}
$$

Since we have added MULTI-CONTEXT POSTULATE to MRIt, we need to augment these results with the following proposition. (The easy proof is omitted.)

Proposition 5.4: Let N' be the set of nodes obtained by applying MULTI-CONTEXT POSTULATE to N. Then $\langle \mathcal{E}[N, p], \mathcal{H}[N, p]\rangle \vdash^{=}_{RIt} \langle \mathcal{E}[N', q], \mathcal{H}[N', q]\rangle$ for all $p \in I$ and $q \in \psi_P(p)$.
In other words, MULTI-CONTEXT POSTULATE simulates an RIt inference (in fact, POSTULATE either BY JOINING or BY PERIPHERAL SCULPTURE) in some processes and have no effect on other processes.

6. Experiments and Discussion

In this section, we present some results of the experiments to see how POSTULATE BY PERIPHERAL SCULPTURE and POSTULATE BY JOINING are effective in MULTI-CONTEXT POSTULATE.

6.1 Settings for Experiments

The experiments were performed with MRIt and MRIt+ on a PC with i5 CPU and 4GB memory. We used a total of 80 test problems, most of which were modified from Dream Corpus examples[†] created by the Mathematical Reasoning Group, University of Edingurgh. In Dream Corpus, there were 69 unconditional equational problems suitable for the input to our system. In addition, we included 11 other problems which cannot be solved without lemma generation. For MRIt+, we had developed a built-in termination checker based on the dependency-pair method [4], [13], [14]. We had also implemented the combination of polynomial interpretation and SAT solving as proposed in [12]. The time limit for each problem solving was set to 15 minutes.

We used the following strategy to apply the inference rules of MRIt+.

1. Choose a node n with the smallest size (where the size of a node $\langle s : t, \ldots\rangle$ is the number of symbols constituting s and t), then apply EXPAND.
2. Normalize all nodes by applying SIMPLIFY-R and SIMPLIFY-H.
3. Apply DELETE, GC, SUBSUME and SUBSUME-P as much as possible, and if there exists a process p such that $\mathcal{E}[N, p] = \emptyset$, then stop with success.
4. Apply MULTI-CONTEXT POSTULATE, then activate the filtering process (where the maximal size of the randomly-generated ground terms is limited to 3), and go back to step 1.

To increase the capability of dealing with more potential proofs, we changed a condition in the EXPAND inference rule from $u \in \mathcal{B}(s)$ to $u \in \mathcal{QB}(s)$ based on [3] as follows. A term u is a *quasi-basic term* with respect to R, if (1) $root(u)$ is a defined symbol and (2) for all $l \to r \in R$ such that l is a basic term, l is unifiable with $cap(u)$, where $cap(f(u_1, \ldots, u_n))$ is a term $f(w_1, \ldots, w_n)$ with each w_i obtained from u_i by replacing maximal subterms with defined root symbol by fresh *constants*. The set of quasi-basic terms of s is denoted by $\mathcal{QB}(s)$.

For example, when $R = \{0+x \to x, s(x)+y \to s(x+y)\}$, the term $z + (v + 0)$ is not basic but quasi-basic, as each left-hand is unifiable with $z + c$, where c is a fresh constant.

Note that in the original definition by Aoto [3], *cap* was defined with fresh *variables* (rather than constants) prohibited from being instantiated when unifying $cap(u)$ with l. This would require a slightly special unification algorithm. Implementable with the standard unification algorithm, our definition is slightly simpler than and clearly equivalent to Aoto's (as constants are never instantiated).

6.2 Results of Experiments

Among the test problems, 36 problems were solved by MRIt (without automated lemma generation). As for the rest of the problems (which, intuitively, need lemmas), MRIt+ (with MULTI-CONTEXT POSTULATE including both BY PERIPHERAL SCULPTURE and BY JOINING) solved 18 more problems that MRIt failed to solve. We also separately tested these problems according to the combinations of the two postulation methods and summarized the results in Table 1.

In Table 1, the problem numbers with the prefix "d_" indicate some of the 69 problems selected from Dream Corpus, and those with "p_" indicate some of the 11 problems added by the authors. The first header indicates whether the problems were tested by SCULPTURE only, JOINING only or both SCULPTURE and JOINING. The "time(ms)" column shows the computation time of MRIt+ in milliseconds, where ∞ indicates that MRIt+ did not succeed within the time limit. The "SUG/GENR" column shows the rate of generated conjectures that got through the test by the conjecture filter, where

[†]The test problems borrowed from Dream Corpus are available at: http://kusshard.complex.eng.hokudai.ac.jp/~haru/mrit/. All the problems mentioned in Table 1 are listed in Appendix B.

Table 1 Some experimental results

	SCULPTURE			JOINING			SCULPTURE&JOINING		
No.	time (ms)	SUG/ GENR	# of proc.	time (ms)	SUG/ GENR	# of proc.	time (ms)	SUG/ GENR	# of proc.
p_1	1688	19/113	13	∞	-	-	1662	19/113	13
p_2	1286	10/14	11	∞	-	-	1232	10/14	11
p_21	967	13/13	15	∞	-	-	992	13/13	15
p_22	28850	200/200	286	2362	7/7	22	2793	32/32	47
p_23	6875	29/29	26	5347	6/22	11	7090	35/51	28
p_24	7853	74/74	79	1348	2/2	9	1408	23/23	24
p_25	1612	25/25	40	∞	-	-	1732	25/25	40
p_26	10475	66/66	47	∞	-	-	29508	212/262	127
p_27	2225	14/14	19	2001	2/2	15	1904	16/16	21
d_25	∞	-	-	9433	133/240	36	9088	133/240	45
d_43	∞	-	-	775	18/32	4	737	18/32	4
d_47	∞	-	-	258	8/10	4	212	8/10	4
d_60	∞	-	-	1679	26/26	10	1683	26/26	10
d_111	∞	-	-	878	4/46	12	805	4/46	12
d_116	∞	-	-	729	4/40	12	891	4/40	12
d_232	460	6/6	5	∞	-	-	656	10/12	5
d_270	68603	864/2076	54	∞	-	-	48090	472/1170	222
d_1052	3108	16/16	28	2829	15/15	12	2902	19/19	15

GENR indicates the number of generated conjectures and SUG indicates the number of suggested lemmas that passed the conjecture filter. The "# of proc." column shows the number of the generated processes in MRIt+, where the data are given only for the successful trials, as other trials are considered to have generated an indefinite number of processes.

The results are very pleasing. The problems p_1, p_2, p_21, p_25, p_26, d_232 and d_270 could not be solved by JOINING but solved by SCULPTURE. In particular, p_1 was solved automatically, although the work in [31] had suggested some additional manual works to resolve the divergence. The problems d_25, d_270 and all problems with the prefix "p_" introduced complex differences as discussed in Example 4.9 (i.e., parallel and nested differences), but our method could treat such differences appropriately at a relatively small cost. Note that, by renaming *reverse* to r, the problem d_43, $r(r(x : xs)) = x : xs$, is essentially equivalent to Example 3.1, $r(r(xs)) = xs$, because $r(r(nil)) = nil$, the base case for the latter, is trivially established by proving (6). In fact, both problems have been solved by MRIt+.

By observing the "SUG/GENR" columns, we see that the conjecture filter prevents incorrect conjectures from being suggested as lemmas for further processing. Clearly, the rate of correctly generated lemmas depends on the specific problems with specific methods. The average rate of correctly generated lemmas was 87% in SCULPTURE and 67% in JOINING.

6.3 Effectiveness of Peripheral Sculpture

The SCULPTURE method and the JOINING method can mutually strengthen the ability of generating effective lemmas to solve more problems, because one can solve some problems that the other could not. In addition, these two methods can also solve the same problems by postulating different conjectures. In our experiments, we have observed such different resolutions of different divergences in five problems: p_22, p_23, p_24, p_27 and d_1052. For example, in the experiment with p_24, we observed a process generating the following diverging sequence:

$$y + (x + y) \to (y + y) + x,$$

$$y + \underline{\boxed{s(x + \boxed{s(\underline{y})})}} \to (y + \boxed{s(\underline{y})}) + \boxed{s(\underline{x})},$$

.....

SCULPTURE successfully resolved this divergence and completed the proof after postulating the following conjecture:

$$v + (x + y) = (v + y) + x.$$

Meanwhile, another process started to generate another diverging sequence as follows:

$$y + 0 \to y,$$

$$y + \boxed{s(\underline{0})} \to \boxed{s(\underline{y})},$$

.....

JOINING resolved this divergence and completed the proof after postulating the following conjecture:

$$y + s(x) = s(y + x).$$

Since these two processes are run concurrently, MRIt+ will complete the proof successfully as soon as one of them succeeds. In our actual experiment, JOINING reached the success earlier than SCULPTURE. Note that the overhead by running SCULPTURE together with JOINING for this case (p_24) was only $1408 - 1348 = 60$ ms, i.e., 4.3%.

Intuitively, JOINING focuses on the parts inside the wave-fronts, while SCULPTURE works on the opposite way by focusing on the parts outside the wave-fronts. Such different characteristics of them are why we can expect them to resolve different divergences in different processes of MRIt+.

6.4 Effectiveness of Multi-Context Postulation

An obvious advantage of multi-context reasoning systems commented in [26] is that they allow, in parallel, various strategies to be tried and various non-deterministic choices to be made in an efficient way. Thanks to this advantage, MULTI-CONTEXT POSTULATE can increase the possibility of success by combining different postulation methods in multi-context reasoning. In particular, there is no need to care about the unsoundness of the lemma postulation methods. By observing the "SCULPTURE&JOINING" column in Table 1, we see that SCULPTURE and JOINING worked very well together in MULTI-CONTEXT POSTULATE and solved the problems that might not have been solved otherwise. Since it is hard in practice to predict which process may face what kind of divergence, we use MULTI-CONTEXT POSTULATE to automatically give every diverging process a chance to adopt different postulation methods in different combinations.

In particular, multi-context reasoning can help different postulating methods cooperate with each other for leading to easier proofs. For example, in d_270, SCULPTURE suggested a conjecture

$$(v + (x + 0)) + (y + 0) = v + (x + y)$$

which could lead to a successful proof by picking up the subterm $x + 0$ for expansion.

However, if JOINING had been enabled as well, it additionally suggested a smaller-sized conjecture

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

after $v+(x+0)$ was picked up for expansion. Simple enough to be focused on by the heuristics of MRIt+, this conjecture turned out to be very useful for a shorter proof. This is why MULTI-CONTEXT POSTULATE with both PERIPHERAL SCULPTURE and JOINING spent less time than that with SCULPTURE only, as shown in the d_270 row of Table 1.

7. Conclusion

In this paper, we have presented a target-aimed automated lemma generation method called PERIPHERAL SCULPTURE for inductive theorem proving based on the rewriting induction of Reddy and Aoto. We have combined it with the JOINING method in the framework of multi-context reasoning system MRIt+. The experiments have shown that the multi-context postulation with these two methods was effective in increasing the possibility of constituting successful proofs. Combination with other postulation methods (target-aimed or bottom-up) in multi-context reasoning systems to develop more powerful and efficient inductive theorem provers is one of our future works.

Acknowledgments

This work was supported by JSPS KAKENHI grant numbers 16K00090 and 16K16032.

References

[1] T. Aoto, "Dealing with non-orientable equations in rewriting induction," Proc. 17th International Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol.4098, pp.242–256, 2006.
[2] T. Aoto, "Rewriting induction using termination checker," JSSST 24th Annual Conf., 3C-3, 2007 (in Japanese).
[3] T. Aoto, "Designing a rewriting induction prover with an increased capability of nonorientable equations," Proc. Symbolic Computation in Software Science AustrianJapanese Workshop, RISC Technical Report, vol.08-08, pp.1–15, 2008.
[4] T. Arts and J. Giesl, "Termination of term rewriting using dependency pairs," Theor. Comput. Sci., vol.236, no.1-2, pp.133–178, 2000.
[5] F. Baader and T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.
[6] D. Basin and T. Walsh, "Difference matching," Proc. 11th International Conf. on Automated Deduction, Lecture Notes in Artificial Intelligence, vol.607, pp.295–309, 1992.
[7] D. Basin and T. Walsh, "Difference unification," Proc. 13th International Joint Conf. of Artificial Intelligence, pp.116–122, 1993.
[8] D.A. Basin and T. Walsh, "Termination orderings for rippling," Proc. 12th International Conf. on Automated Deduction, Lecture Notes in Artificial Intelligence, vol.814, pp.466–483, 1994.
[9] R.S. Boyer and J.S. Moore, A Computational Logic, Academic Press, New York, 1979.
[10] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill, "Rippling: A heuristic for guiding inductive proofs," Artificial Intelligence, vol.62, no.2, pp.185–253, 1993.
[11] 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.
[12] 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 Conf. on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science, vol.4501, pp.340–354, 2007.
[13] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke, "Mechanizing and improving dependency pairs," J. Autom. Reasoning., vol.37, no.3, pp.155–203, 2006.
[14] N. Hirokawa and A. Middeldorp, "Tyrolean termination tool: Techniques and features," Inform. Comput., vol.205, no.4, pp.474–511, 2007.
[15] G. Huet and J.-M. Hullot, "Proofs by induction in equational theories with constructors," J. Comput. Syst. Sci., vol.25, pp.239–266, 1982.
[16] D. Hutter, "Guiding induction proofs," Proc. 10th International Conf. on Automated Deduction, Lecture Notes in Artificial Intelligence, vol.449, pp.147–161, 1990.
[17] CC. Ji, H. Sato, and M. Kurihara, "A new implementation of multi-context algebraic inductive theorem prover," Lecture Notes in Engineering and Computer Science: Proc. The World Congress on Engineering and Computer Science 2015, pp.109–114, 2015.
[18] M. Johansson, L. Dixon, and A. Bundy, "Conjecture synthesis for inductive theories," J. Autom. Reasoning., vol.47, no.3, pp.251–289, 2011.
[19] 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.
[20] M. Kurihara and H. Kondo, "Completion for multiple reduction orderings," J. Autom. Reasoning., vol.23, no.1, pp.25–42, 1999.
[21] D.R. Musser, "On proving induction properties of abstract data types," Proc. 7th ACM Symposium on Principles of Programming Languages, pp.154–162, 1980.
[22] P. Urso and E. Kounalis, "Sound generalizations in mathematical

induction," Theor. Comput. Sci., vol.323, pp.443–471, 2004.
[23] D.A. Plaisted, "Semantic confluence tests and completion methods," Inform. Comput., vol.65, no.2-3, pp.182–215, 1985.
[24] D.A. Plaisted, "Equational reasoning and term rewriting systems," in Handbook of Logic in Artificial Intelligence and Logic Programming, ed. D.M. Gabbay et al., vol.1, pp.274–367, Oxford Univ. Press, 1993.
[25] U.S. Reddy, "Term rewriting induction," Proc. 10th International Conf. on Automated Deduction, Lecture Notes in Computer Science, vol.449, pp.162–177, 1990.
[26] H. Sato and M. Kurihara, "Multi-context rewriting induction with termination checkers," IEICE Trans. Inf. & Syst., vol.E93-D, no.5, pp.942–952, 2010.
[27] H. Sato, S. Winkler, M. Kurihara, and A. Middeldorp, "Multi-completion with termination tools (system description)," Proc. 4th International Joint Conf. on Automated Reasoning, Lecture Notes in Artificial Intelligence, vol.5195, pp.306–312, 2008.
[28] 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, 2009.
[29] S. Shimazu, T. Aoto, and Y. Toyama. "Automated lemma generation for rewriting induction with disproof," Computer Software, vol.26, no.2, pp.41–55, 2009 (in Japanese).
[30] Terese, Term Rewriting Systems, Cambridge University Press, 2003.
[31] T. Walsh, "A divergence critic for inductive proof," J. Artif. Intell. Res., vol.4, pp.209–235, 1996.
[32] I. Wehrman, A. Stump, and E. Westbrook, "Slothrop: Knuth-Bendix completion with a modern termination checker," Proc. 17th International Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol.4098, pp.287–296, 2006.
[33] C.-P. Wirth, "History and future of implicit and inductionless induction: beware the old jade and the zombie!," in Mechanizing Mathematical Reasoning, Lecture Notes in Artificial Intelligence, vol.2605, pp.192–203, 2005.

Appendix A: Inference Rules of MRIt

DELETE: $N \cup \{\langle s : s, H_1, H_2, E\rangle\} \vdash N$

EXPAND: $N \cup \{\langle s : t, H_1, H_2, E \cup E'\rangle\} \vdash$
$N \cup \{\langle s : t, H_1 \cup E', H_2, E\rangle\}$
$\quad \cup \{\langle s' : t', \emptyset, \emptyset, E'\rangle \mid s' = t' \in \mathrm{Expd}_u(s, t)\}$
if $E' \neq \emptyset, u \in \mathcal{B}(s)$ and
$\mathcal{H}[N, p] \cup \mathcal{R} \cup \{s \to t\}$ terminates for all $p \in E'$

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: $N \cup \{\langle s : t, H_1, H_2, E\rangle\} \vdash$
$$N \cup \left\{\begin{array}{l}\langle s : t, H_1, H_2, E \backslash H\rangle, \\ \langle s' : t, \emptyset, \emptyset, E \cap H\rangle\end{array}\right\}$$
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: $N \cup \left\{\begin{array}{l}\langle s : t, H_1, H_2, E\rangle, \\ \langle s' : t', H_1', H_2', E'\rangle\end{array}\right\} \vdash$
$N \cup \{\langle s : t, H_1 \cup H_1', H_2 \cup H_2', E''\rangle\}$
if $s : t$ and $s' : t'$ are variants and
$E'' = (E \backslash (H_1' \cup H_2')) \cup (E' \backslash (H_1 \cup H_2))$

SUBSUME-P: $N \vdash sub(N, L)$
if $\forall p \in L, \exists p' \in I(N) \backslash L :$
$\langle \mathcal{E}[N, p], \mathcal{H}[N, p]\rangle = \langle \mathcal{E}[N, p'], \mathcal{H}[N, p']\rangle$

where $I(N)$ denotes the set of all processes that appear in a label of a node in N and $sub(N, L) = \{\langle s : t, H_1 \backslash L, H_2 \backslash L, E \backslash L\rangle \mid \langle s : t, H_1, H_2, E\rangle \in N\}$.

Appendix B: Problems and Generated Lemmas

Some problems used in our experiments and lemmas automatically generated by MRIt+ with the SCULPTURE&JOINING setting are listed in this appendix.
Each problem starts with its name, followed by rewrite rules (as axioms), followed by an equation (as an inductive theorem to be proved), and ends with the generated lemmas displayed in square brackets.

- p_1
  $nil@x \to x$
  $(x : ys)@zs \to x : (ys@zs)$
  $(xs@xs)@xs = xs@(xs@xs)$
  $[(xs@ys)@zs = xs@(ys@zs)]$

- p_2
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(x + x) + x = x + (x + x)$
  $[(x + y) + y = x + (y + y)]$

- p_21
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(x + y) + x = x + (y + x)$
  $[(x + y) + z = x + (y + z)]$

- p_22
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(x + y) + x = x + (x + y)$
  $[x + s(y) = s(x + y)]$

- p_23
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(x + y) + x = y + (x + x)$
  $$\left[\begin{array}{l}(x + 0) + y = x + y \\ (x + s(y)) + z = s((x + y) + z)\end{array}\right]$$

- p_24
  $0 + y \to y$

$s(x) + y \to s(x + y)$
$(x + x) + y = x + (y + x)$
$[x + s(y) = s(x + y)]$

- p_25
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(x + x) + y = x + (x + y)$
  $[(x + y) + z = x + (y + z)]$
- p_26
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(y + x) + x = x + (x + y)$
  $\begin{bmatrix} x + (y + 0) = x + y \\ x + (y + s(z)) = s(x + (y + z)) \end{bmatrix}$
- p_27
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $(y + x) + x = x + (y + x)$
  $[x + s(y) = s(x + y)]$
- d_25
  $(x : xs)@ys \to x : (xs@ys)$
  $nil@ys \to ys$
  $reverse(x : xs) \to reverse(xs)@(x : nil)$
  $reverse(nil) \to nil$
  $reverse(xs@ys) = reverse(ys)@reverse(xs)$
  $[xs@(ys@zs) = (xs@ys)@zs]$
- d_43
  $(x : xs)@ys \to x : (xs@ys)$
  $nil@ys \to ys$
  $reverse(x : xs) \to reverse(xs)@(x : nil)$
  $reverse(nil) \to nil$
  $reverse(reverse(x : xs)) = x : xs$
  $[reverse(xs@(x : nil)) = cons(x, reverse(xs))]$
- d_47
  $(x : xs)@ys \to x : (xs@ys)$
  $nil@ys \to ys$
  $reverse(x : xs) \to reverse(xs)@(x : nil)$
  $reverse(nil) \to nil$
  $length(x : xs) \to s(length(xs))$
  $length(nil) \to 0$
  $length(reverse(xs)) = length(xs)$
  $[length(xs@(x : nil)) = s(length(xs))]$
- d_60
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $0 * y \to 0$
  $s(x) * y \to y + (x * y)$
  $double(0) \to 0$
  $double(s(x)) \to s(s(double(x)))$
  $double(x) = s(s(0)) * x$
  $[x + s(y) = s(x + y)]$
- d_111
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $difference(0, j) \to 0$
  $difference(s(i), 0) \to s(i)$
  $difference(s(i), s(j)) \to difference(i, j)$
  $difference((y + x), x) = y$
  $[x + s(y) = s(x + y)]$
- d_116
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $difference(0, j) \to 0$
  $difference(s(i), 0) \to s(i)$
  $difference(s(i), s(j)) \to difference(i, j)$
  $difference(s(y + x), x) = s(y)$
  $[x + s(y) = s(x + y)]$
- d_232
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $0 * y \to 0$
  $s(x) * y \to y + (x * y)$
  $s(s(0)) * x = x + x$
  $[x + (y + 0) = x + y]$
- d_270
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $0 * y \to 0$
  $s(x) * y \to y + (x * y)$
  $s(s(s(s(0)))) * x = s(s(0)) * (s(s(0)) * x)$
  $\begin{bmatrix} (x + (y + 0)) + (z + 0) = x + (y + z) \\ (x + y) + z = x + (y + z) \end{bmatrix}$
- d_1052
  $0 + y \to y$
  $s(x) + y \to s(x + y)$
  $0 * y \to 0$
  $s(x) * y \to y + (x * y)$
  $(x + y) * z = (x * z) + (y * z)$
  $[(x + y) + z = x + (y + z)]$

Chengcheng Ji received BS and MS degrees from Hokkaido Information University in 2012 and 2014, respectively. He is currently a Ph.D. candidate at Graduate School of Information Science and Technology, Hokkaido University. His current research interests include term rewriting systems, automated theorem proving, and software engineering. He is a member of IPSJ.

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

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