Automated Proof and Discovery of Inductive Theorems with Rewriting Induction over Multi-Context Reasoning Systems: State-of-the-Art Technologies and Perspectives

Masahito Kurihara, Haruhiko Sato, and ChengCheng Ji

***Abstract*— A system for efficiently running mutually related virtual processes as a single process is called a multi-context reasoning system. We focus on its application to automated proof and discovery of inductive theorems in the framework of rewriting induction and briefly survey past and current state-of-the-art technologies of such systems. Then to deal with lemma discovery in this framework, we present an extension of MRIt, the multi-context rewriting induction system with termination tools developed by Sato and Kurihara, and discuss its effectiveness.**

***Index Terms*— inductive theorem proving, rewriting induction, multi-context reasoning system, lemma generation and discovery**

I. Introduction

ALGEBRAIC structures often appear in various theories in computer science as a means of representing systems and reasoning about their properties such as their correctness. Related technologies include term rewriting, equational reasoning, and their application to formal methods in software engineering.

Executable implementation of these technologies often involves *nondeterministic* computation. In nondeterministic computational processes, users and/or algorithms make a series of choices (or decisions) at the beginning and subsequent temporal points (or choice points). We refer to such a series of choices as a *context* for such a process. Naturally, one hopes that the context will lead to *success* defined for such a computation, but it is generally not easy to make a right decision leading to success. Surprisingly, however, some researchers manage to make their computation lead to success by setting parameters and strategies beforehand to 'appropriately' control the nondeterminism. This often involves many handmade trial-and-errors and/or tricks to suppress the nondeterminism in their experiments. However, we believe that most researchers would admit that there was a fair amount of inappropriate settings and failures before their 'successes'. Thus, we should develop a highly universal technology to solve this problem for actual successful nondeterministic computation.

In a simple computational system, what is necessary is just *backtracking*, going back to the previous choice point when there was a failure. In a complex system with an unlimited search space, however, backtracking is often impossible because the system may be run indefinitely without success or failure. Therefore, a concurrent computation (or a sequential computation simulating concurrency) is necessary. However, a naïve implementation of such a concurrency often learns the hard way, facing the reality in which the number of processes exponentially grows too large to be practical.

To solve this problem, we have been developing a computational system in which a large amount of computation and reasoning can be done efficiently in a single process. This idea is based on empirical knowledge that processes with 'similar' contexts often have many mutually related computational tasks that can be carried out simultaneously since they involve common computation. Though the average computational complexity may still be exponential even in such a system, one can handle larger problems in practice by suppressing the base of the exponential function with a special mechanism implementing the idea described above.

Such a system for efficiently running mutually related virtual processes as a single process is called a *multi-context reasoning system* [1]. Focusing our attention on its application to automated proof and discovery of *inductive theorems* in the framework of *rewriting induction* [2], we briefly survey past and current state-of-the-art technologies of such systems. Then to deal with *lemma discovery* in this framework, we present an extension of MRIt, the multi-context rewriting induction system with termination tools developed by Sato and Kurihara [1], and discuss its effectiveness.

In Section 2, we describe the general idea of a multi-context reasoning system and its past development. In Section 3, we discuss three types of lemma-discovery methods necessary for powerful inductive theorem proving. In Section 4, we discuss the new ideas being developed on multi-context lemma discovery. In Section 5, we provide discussion and give concluding remarks.

Manuscript received June XX, 2018; revised July XX, 2018. This work was supported in part by JSPS KAKENHI Grant Numbers 16K00090 and 16K16032.

Masahito Kurihara is with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo, 060-0814 Japan (e-mail: kurihara@ist.hokudai.ac.jp).

Haruhiko Sato is with the Department of Engineering, Hokkai-Gakuen University, Sapporo, 064-0926 Japan (e-mail: h-sato@hgu.jp).

ChengCheng Ji is with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo, 060-0814 Japan (e-mail: kisyousei@complex.ist.hokudai.ac.jp).

II. MULTI-CONTEXT REASONING SYSTEMS FOR ALGEBRAIC STRUCTURES

A. Multi-completion

The purpose of *multi-context reasoning systems* for algebraic structures is to effectively and efficiently handle *equations* in the form $s = t$ and *rewrite rules* in the form $s \to t$, where the latter can be obtained by directing the former from left to right (or from right to left). The equations and rewrite rules consist of two terms s (the left-hand side) and t (the right-hand side). A *term* is either a *variable* or *function symbol* followed by n terms as arguments, where the nonnegative integer n denotes the *arity* of this function symbol. A set of rewrite rules is called a *term rewriting system*.

With the common ideas and technologies of multi-context reasoning, one can develop various special-purpose reasoning systems, such as equational theorem provers, termination verifiers, and equational completion systems. For example, *multi-completion* (multi-context completion), which extends the *Knuth-Bendix procedure* (a semi-algorithm) to transform the given set of equations into a complete (i.e., terminating and confluent) term rewriting system, associates each context of its computation with a partial order over the set of terms to use this structure for directing equations such that their left-hand sides are greater than the corresponding right-hand sides in that order.

In fact, various multi-context reasoning systems stem from the study by Kurihara and Kondo [3] on multi-completion in 1999. The basic idea is to introduce a data structure (called a *node*) represented by $\langle s:t, R_1, R_2, E\rangle$ where $s:t$ is an ordered pair of terms, and R_1, R_2, and E are sets of contexts (called *labels*).

Each context implicitly specifies partial orders on the set of terms and is associated with a virtual computational process running a standard (Knuth-Bendix) completion procedure [4]. The multi-completion system simulates those procedures running in parallel, maintaining a set of nodes.

In such a system, the node structure is interpreted as follows. For each context in R_1 (R_2), the associated Knuth-Bendix process contains a rewrite rule $s \to t$ ($t \to s$) in its database because s is greater than t (t is greater than s) in the associated partial order; and for each context in E, the associated process contains an equation $s = t$. Based on this semantics, the node is identified with $\langle t:s, R_2, R_1, E\rangle$.

For example, suppose that we have the following two nodes:

$$\langle b: f(a), \{C_0, C_2\}, \{C_1\}, \{C_3, C_5\}\rangle,$$
$$\langle a: c, \{C_0, C_1, C_2, C_3\}, \{C_4, C_5\}, \phi\rangle.$$

We can see that for contexts C_0 and C_2, the associated processes have rewrite rules $b \to f(a)$ and $a \to c$; thus, they can create a new rewrite rule $b \to f(c)$.

Similarly, for context C_1, we see two rules $f(a) \to b$ and $a \to c$. In this case, the completion procedure is designed to create the equation $f(c) = b$.

Finally, for context C_3, we see $b = f(a)$ and $a \to c$ to create $b = f(c)$. Combining these three inferences, the multi-completion system applies an inference rule (called 'REWRITE-2') to create a new node

$$\langle b: f(c), \{C_0, C_2\}, \phi, \{C_1, C_3\}\rangle$$

clearly consistent with our interpretation. Note that in any case, $b: f(a)$ is rewritten to $b: f(c)$ and no longer exists in contexts C_0, C_1, C_2, and C_3. Therefore, the system modifies the first node to

$$\langle b: f(a), \phi, \phi, \{C_5\}\rangle,$$

meaning that $b = f(a)$ can only exist in the process associated with C_5.

Based on this idea, the multi-completion procedure is formally defined as a *meta*-inference system (on the set of nodes) in which the *base-level* inference (on the set of equations and rewrite rules) is combined with set operations, such as union, intersection, and difference, over the family of labels.

Implementation of multi-completion was improved in 2004 and 2006, where each label (a set of contexts, i.e., a set of partial orders) was represented as a Boolean function combining atomic propositions such as $f \succ g$ [5]. Such a function was compactly represented as a type of direct acyclic graph called a *binary decision diagram*. An empirical study showed that even if the number of contexts grew exponentially by 1,000%, the execution time grew only by 1% [6].

In 2009, multi-completion was extended in a new framework, in which contexts could be *dynamically* generated and updated (rather than statically specified and maintained) so that an arbitrary *termination checker* could be dynamically invoked to determine the right context defined by the direction of equations. This idea greatly improved the functionality and performance of the system [7], [8].

In 2012, the implementation scheme of termination checkers used in multi-completion was improved [9]. Before this study, the termination checker had involved a performance bottleneck that prevented multi-context reasoning from working efficiently when trying to prove the termination of multiple term rewriting systems. However, that study proposed the implementation of termination checkers over multi-core CPUs and the resultant parallel execution resulted in a tenfold improvement in performance.

In 2013, Winkler et al. [10] refined the multi-completion procedure with 'critical pair criteria' and isomorphisms, giving the full proof of its correctness. They implemented their approach in the tool mkbTT. Another improvement in implementation was seen in 2015 in the study by Ji et al. [11], where they exploited 'lazy evaluation' schemes to improve efficiency without increasing the structural complexity of the program reimplemented in Scala, a modern object-oriented functional programming language.

B. Multi-context inductive theorem proving

An *inductive theorem* is a proposition that holds on an algebraic model consisting of a (possibly infinite) set of elements (such as natural numbers, lists, and trees) and a set of associated operations. Inductive theorems are often proved by mathematical induction and its extension. Induction is a

critical notion in software science and engineering, directly relating to program structures such as recursion and iteration. Its application includes a formal verification of program properties. For example, suppose we have two functional programs: $f(x)$, a readable but inefficient program, and $g(x)$, an efficient but hard-to-read one. Then we may want to verify the equivalence of the two by proving the inductive theorem $f(x) = g(x)$. When proved, the theorem ensures the correctness of the hard-to-read but efficient program $g(x)$, if $f(x)$ is clearly correct from its readability.

In 2010, Sato and Kurihara [1] presented an application of multi-context reasoning to inductive theorem proving. They reported that within practical computation time using their implementation, they solved 35 out of 69 standard benchmark problems without giving any information suitable for directing equations. Such information had been practically a 'must' in previous implementations.

In the following years, the 34 unsolved problems were analyzed, and it was found that the failures were caused by the system's inability to generate lemmas. A *lemma* (or an auxiliary theorem) is an auxiliary proposition necessary for proving a main theorem. Lemmas are normally created by humans or machines heuristically as hypotheses then proved and used in the proof of the main theorem. In the following section, we discuss the classification of automated procedures for lemma generation proposed in the literature.

III. Automated lemma discovery

As discussed in the previous section, lemmas are hypotheses which, if once proved as theorems, can be used to prove the main theorem. For example, some common subexpressions (representing some object) contained in the main theorem may be replaced with a variable (representing an arbitrary object) to *generalize* the main theorem to a more general hypothesis.

Basically, lemma-discovery processes in an inductive theorem prover involve two phases: *generation* and *test*. In the generation phase, hypotheses are generated as candidate lemmas according to certain heuristic algorithms. Then in the test phase, the validity of the candidates is tested by the inductive theorem prover (recursively). Those hypotheses that have been proved successfully become lemmas, which may be used for proving the main theorem. In this case, we may say that those lemmas have been *discovered*. In this section, we discuss basic automated methods for lemma discovery, classifying them into top-down and bottom-up ones.

A. Top-down lemma discovery

Given a main theorem to be proved, *top-down* lemma-discovery methods attempt to transform the theorem into inductive hypotheses, which may be useful as lemmas for proving the main theorem. The task involves various processes such as syntactical extension and logical analysis of the main theorem. Such methods are *goal oriented* in the sense that their basis is placed on the main theorem to be proved. These methods are classified into the following two categories based on their soundness.

Sound lemma-discovery methods generate only 'correct' hypotheses in the sense that they are inductive theorems if the main theorem is an inductive theorem. Since such methods tend to create only a small number of hypotheses, their ability to discover useful lemmas is not high, but the burden of proving the resultant hypotheses can be light. An example of a top-down sound method is presented by Pascal and Emmanuel [12].

By contrast, *unsound* lemma-discovery methods are used to attempt to generate useful hypotheses, even though most may be incorrect. Since incorrect hypotheses can never be proved, they put a heavy burden on the system. However, combined with the 'filtering' technique for removing the hypotheses a random instance of which has turned out to be incorrect, these methods can be improved in terms of efficiency in practice. In that case, such methods can be very useful because their ability of discovering useful lemmas is generally high. An example of a top-down unsound method is presented by Walsh [13].

B. Bottom-up lemma generation

Given a *theory* (or background knowledge) represented by a set of axioms, a *bottom-up* lemma-discovery method is used to attempt to generate various inductive hypotheses that are true on the theory, *irrespective of the main theorem*. Hypotheses that have turned out to be incorrect in the filtering process (described in the previous section) are removed. The hypotheses that have been proved true are stored as established theorems in the database. When given a main theorem to be proved, the system can use some established theorems as lemmas.

In some traditional logical systems (such as first-order predicate logic), one can generate, in principle, all the theorems by systematically applying a 'complete' set of inference rules to the initial set of axioms. However, this is not the case in inductive theorem proving, where application of heuristic methods is essential. Bottom-up methods are often referred to as automated discovery of inductive *theorems* (rather than lemmas). Examples of bottom-up lemma discovery method are given by Johansson et al. [14], McCasland et al. [15], and Sato et al. [16].

IV. Multi-context lemma discovery

In the previous section, we discussed *sound top-down*, *unsound top-down*, and *bottom-up* methods for automated lemma discovery. Then the natural question is: which is the best? However, the natural answer is neither. We believe that a sophisticated combination of these methods is the best. In this section, we discuss how multi-context reasoning systems can be useful for providing such a combination.

A. Rewriting induction and lemma postulation

Before discussing multi-context lemma discovery, we need to review the underlying *rewriting induction with termination tools*. The theoretical basis was developed based on the work of *term rewriting induction* by Reddy [17]. Later, this work was slightly modified and augmented in a simpler setting of *rewriting induction* (RI) by Aoto [2] and further extended to *rewriting induction with termination tools* (RIt) by Aoto [18].

In RIt, the system is initially given a set R of axioms (as a

term rewriting system) and a set E_0 of conjectures (in the form of equations), which represent inductive theorems to be proved.

The system starts its computation from state $\langle E_0, H_0 \rangle$ (where H_0 is the empty set of inductive hypotheses) and generates a sequence of states derived by the inference rules, where the new state derived from the previous state $\langle E_i, H_i \rangle$ is denoted by $\langle E_{i+1}, H_{i+1} \rangle$.

The inference rules generally work on a pair of conjectures E (in the form of equations) and inductive hypotheses H (in the form of rewrite rules), i.e., $\langle E, H \rangle$. There are four inference rules. Their formal description is given by Sato et al. [1] and Aoto [18]. We give only an informal account below.

1) Suppose E contains a trivial equation (with identical left- and right-hand sides). Then the DELETE rule removes such an equation to generate a new state.

2) Suppose E contains an equation $s = t$ such that s can be rewritten to another term s' by using a rewrite rule of $R \cup H$. Then in the new state, the SIMPLIFY rule replaces $s = t$ with new equation $s' = t$.

Before discussing the third inference rule, we need to define some notions. The *root* symbol of a term $f(\cdots)$ is the function symbol f. The function symbols appearing as the root symbol of the left-hand side of at least one of the rewrite rules of R are *defined* symbols. The function symbols other than defined symbols are *constructors*. A term is *basic* if its root symbol is a defined symbol and its arguments contain no defined symbols. The rewriting induction assumes that R is a so-called *constructor system* so that the left-hand side of every rewrite rule of R is basic (More precisely, R must be 'ground-reducible' and 'convergent,' but this condition is not important in our discussion of this paper). A *substitution* σ is a function that maps a variable to a term, and $\sigma(t)$ represents the term obtained by replacing every variable x in t with $\sigma(x)$. Now we can proceed to the third inference rule.

3) Suppose E contains $s = t$ such that s contains a basic subterm u unifiable with the left-hand side l of a rewrite rule $l \to r$ of R. Then we can obtain an equation by first replacing u with r then applying the most general unifier σ of u and l to both sides. Thus, the resultant equation can be formally represented as

$$\sigma(s[u \leftarrow r]) = \sigma(t),$$

where $s[u \leftarrow r]$ represents s in which u is replaced with r. If there are more than one $l \to r$ with l unifiable with u, we may obtain more than one such resultant equation. Given s, t, and u, let $\mathrm{Expd}_u(s,t)$ denote the set of all such resultant equations. Then in the new state, the EXPAND rule replaces $s = t$ with those equations of $\mathrm{Expd}_u(s,t)$ and adds a new rewrite rule $s \to t$ to H (as an inductive hypothesis) only if the term rewriting system $R \cup H \cup \{s \to t\}$ is *terminating*, i.e., there are no infinite rewrite sequences using those rules. The property of termination may be checked using automated termination tools (termination checkers) for term rewriting systems.

4) Finally, the POSTULATE rule allows us to add arbitrary equations to E as hypotheses (candidate lemmas) in the new state. Note that this inference rule is not incorporated in the systems proposed by Sato and Kurihara [1] and Aoto [18], because lemma discovery is out of the scope of their discussions.

Starting from the initial $\langle E_0, H_0 \rangle$, the system derives a sequence of states $\langle E_i, H_i \rangle$ $(i = 0,1,2,\cdots)$ and stops its derivation when we obtain $E_n = \phi$ for some n. In this case, the derivation is *successful*, meaning that it is ensured that the equations given in E_0 is inductive theorems of R. Note that such a derivation can be infinite, meaning that the computation will continue indefinitely, never stopping. In such a case, we say that the system is *diverging*.

B. Multi-context rewriting induction and lemma discovery

Let us see how the rewriting induction with termination tools can be incorporated into the framework of multi-context reasoning. Inference rules other than POSTULATE are already incorporated into the MRIt system [1]. We briefly and only informally describe this system below.

Basically, MRIt consists of five inference rules working on a set of nodes of the form

$$\langle s : t, H_1, H_2, E \rangle,$$

where $s : t$ is an ordered pair of terms, and H_1, H_2, and E are sets of *process indices* called labels. Intuitively, H_1 (H_2) represents a set of processes that contain the inductive hypothesis $s \to t$ ($t \to s$) in the H part of the current state $\langle E, H \rangle$, whereas E represents a set of processes that contain the conjecture $s = t$ in the E part.

Given a set of nodes N, the set of conjectures (equations) contained in the E part of the state of the process p is denoted by $\mathrm{E}[N, p]$. Similarly, the set of inductive hypotheses (rewrite rules) contained in the H part of the state of p is denoted by $\mathrm{H}[N, p]$. These two notions provide the theoretical tool for formally discussing the correspondence between MRIt and its underlying RIt processes [1].

The first inference rule of MRIt is DELETE, which removes a node $\langle s : t, H_1, H_2, E \rangle$ in which s and t are identical. It simulates the DELETE rule of RIt, removing the trivial equation $s = s$ from appropriate processes.

The second rule EXPAND simulates the counterpart of RIt in a slightly complicated manner. Suppose we have a node

$$\langle s : t, H_1, H_2, E \cup E' \rangle$$

in which the last element can be split into two disjoint sets E and E' such that the EXPAND rule of RIt can be applied to the conjecture $s = t$ contained in the processes of E' $(\neq \phi)$, generating a new inductive hypothesis $s \to t$ and new conjectures $\mathrm{Expd}_u(s,t)$ for some fixed u, whereas in the processes of E, the rule fails to be applied. Then the EXPAND rule of MRIt removes this node from N, adding the node

$$\langle s : t, H_1 \cup E', H_2, E \rangle$$

and nodes

$$\langle s' : t', \phi, \phi, E' \rangle$$

for all $(s' = t') \in \mathrm{Expd}_u(s,t)$.

The third rule SIMPLIFY-R simulates the SIMPLIFY rule of RIt when the simplification is carried out using a rewrite rule of R. Suppose we have a node $\langle s:t, H_1, H_2, E\rangle$ such that $E \neq \phi$ and s can be simplified into s' by a rewrite rule of R. Then the SIMPLIFY-R rule removes this node from N, adding two nodes

$$\langle s:t, H_1, H_2, \phi\rangle$$

and

$$\langle s':t, \phi, \phi, E\rangle.$$

The fourth rule SIMPLIFY-H is almost the same as SIMPLIFY-R except that it is used when the simplification is carried out using an inductive hypothesis. Suppose we have two nodes

$$\langle s:t, H_1, H_2, E\rangle$$

and

$$\langle l:r, H, \ldots, \ldots\rangle$$

such that $E \cap H \neq \phi$ and s can be simplified into s' by the inductive hypothesis $l \to r$ contained in the processes of H. Then the SIMPLIFY-H rule removes the former node from N, adding two nodes

$$\langle s:t, H_1, H_2, E \setminus H\rangle$$

and

$$\langle s':t, \phi, \phi, E \cap H\rangle,$$

where $E \setminus H$ represents subtraction of the set H from E.

The role of the fifth rule FORK is to fork a process into several processes. Suppose several choices should be made in a process p. Then this rule replaces the process index of p contained in all the labels of the current node set N with k process indices $p_1, \cdots, p_k$, where k is the number of those choices.

We add a new inference to MRIt for incorporating the POSTULATE rule of RIt, i.e., the functionality of lemma discovery, as follows. Suppose that we have now a set of processes P in which the POSTULATE rule can generate k conjectures $l_i = r_i$ $(i = 1, \ldots, k)$ that can be added to the E part of the $\langle E, H\rangle$ of each process $p \in P$. Then we have each p fork into k+1 processes $p.i$ (which represents the process index of the i-th process generated by the fork, where $0 \le i \le k$), letting the E part of $p.i$ contain the i-th conjecture $l_i = r_i$ for $1 \le i \le k$ and letting the process $p.0$ continue its computation without any of those k conjectures. This means that we have a strategy that *accepts at most one conjecture at each opportunity*. Note that $p.0$, which accepts no conjectures, is important because no such conjectures may be true. Based on this idea, our new inference rule for MRIt, MULTI-CONTEXT POSTULATE, will first modify all current nodes so that every p of P will be forked into (k+1) processes $p.i$ $(0 \le i \le k)$ then add new k nodes

$$\langle l_i : r_i, \phi, \phi, P_i\rangle,\ 1 \le i \le k,$$

to the current set of nodes, where P_i denotes the set $\{p.i \mid p \in P\}$ consisting of $|P|$ process indices.

Given E_0, an initial set of conjectures, and R, a set of axioms in the form of 'ground-reducible' and 'convergent' rewrite rules, MRIt starts its reasoning with the initial set of nodes

$$N_0 = \{\langle s:t, \phi, \phi, \{\varepsilon\}\rangle \mid (s=t) \in E_0\},$$

where ε denotes the 'root' process. Then the inference rules of MRIt will be applied to the set of nodes to generate a sequence $N_0, N_1, N_2, \cdots$ of node sets. If the system has reached some N_c containing in its labels a process p such that the set $\mathrm{E}[N_c, p]$ of conjectures held in the E part of p is empty, then the system can conclude that the conjectures initially given in E_0 are inductive theorems of R.

V. DISCUSSION AND CONCLUSION

We presented an extension of MRIt, the multi-context rewriting induction system with termination tools, incorporating the lemma postulation inference rules of RIt. We believe that this framework is general enough to incorporate various lemma-discovery methods. However, more important is what concrete methods we should incorporate. Several methods have been proposed in the literature [12] – [15], but they are not necessarily appropriate for the induction framework of RIt [2], [17]. To partially solve this problem, an idea to adapt a known method for RIt is presented [19], where a procedure called 'modification' for generating a conjecture is given and formally presented as an inference rule called POSTULATE BY JOINING [20].

Another method for generating conjectures has recently been developed [20], where a new inference rule called POSTULATE BY PERIPHERAL SCULPTURE was developed based on the idea of splitting an 'annotated' term into 'peripheral' and 'calm' parts, both defined in terms of well-known notions of 'term difference' and 'annotation' [21], [22].

Those two inference rules have been incorporated into the multi-context inductive reasoning system and their experimental results indicated its effectiveness [20]. However, we should say that this is the first step to a more effective method of combining the underlying induction systems with multi-context reasoning systems. Extension of this step is one of the most general future research topics in this field.

Other research topics include problems in discovering useful but syntactically complex lemmas [23], improvement of implementation technique exploiting lazy evaluation in a functional programming language [24], [25], and extensive analysis of the relation between inductive theorem proving and tree automata [26].

REFERENCES

[1] H. Sato and M. Kurihara, "Multi-context rewriting induction with termination checkers," *IEICE Trans. Information and Systems*, vol. E93-D, no. 5, 2010, pp. 942–952.

[2] T. Aoto, "Dealing with non-orientable equations in rewriting induction," in *Lecture Notes in Computer Science 4098: Proc. 17th Int. Conf. Rewriting Techniques and Applications, 2006*, pp. 242–256.

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

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

[5] I. Yokoyama and M. Kurihara, "Completion for multiple path orderings based on precedence and binary decision diagrams," *Trans. the Japanese Society for Artificial Intelligence*, vol. 19, no. 6, 2004, pp. 472–482. (in Japanese)

[6] H. Sato and M. Kurihara, "Implementation and performance evaluation of multi-completion procedures for term rewriting systems

with recursive path orderings with status," *IEICE Trans. Information and Systems*, vol. J89-D, no. 4, 2006, pp. 624–631. (in Japanese)
[7] H. Sato, S. Winkler, M. Kurihara, and A. Middeldorp, "Multi-completion with termination tools (system description)," in *Lecture Notes in Artificial Intelligence 5195: Proc. 4th Int. Joint Conf. Automated Reasoning*, 2008, pp. 306–312.
[8] H. Sato, M. Kurihara, S. Winkler, and A. Middeldorp, "Constraint-based multi-completion procedures for term rewriting systems," *IEICE Trans. Information and Systems*, vol. E92-D, no. 2, 2009, pp. 220–234.
[9] R. Ding, H. Sato, and M. Kurihara, "Parallelization of termination checkers for algebraic software," *Trans. Machine Learning and Artificial Intelligence*, vol. 2, no. 4, 2014, pp. 102–114.
[10] S. Winkler, H. Sato, A. Middeldorp, and M. Kurihara, "Multi-completion with termination tools," *Journal of Automated Reasoning*, vol. 50, no. 3, 2013, pp. 317–354.
[11] C. Ji, H. Sato, and M. Kurihara, "Lazy evaluation schemes for efficient implementation of multi-context algebraic completion system," *IAENG International Journal of Computer Science*, vol. 42, no. 3, 2015, pp. 282–287.
[12] U. Pascal and K. Emmanuel, "Sound generalizations in mathematical induction," *Theoretical Computer Science* 323, 2004, pp. 443–471.
[13] T. Walsh, "A divergence critic for inductive proof," *Journal of Artificial Intelligence Research*, vol. 4, 1996, pp. 209–235.
[14] M. Johansson, L. Dixon, and A. Bundy, "Conjecture synthesis for inductive theories," *Journal of Automated Reasoning*, vol. 47, no. 3, 2011, pp. 251–289.
[15] R. McCasland, A. Bundy, and S. Autexier, "Automated discovery of inductive theorems," *Studies in Logic, Grammar and Rhetoric*, vol. 10, no. 23, 2007, pp. 135–149.
[16] H. Sato and M. Kurihara, "Discovering inductive theorems using rewriting induction," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics, 2016*, pp. 989–993.
[17] U. Reddy, "Term rewriting induction," in *Lecture Notes in Computer Science 449: Proc. 10th Int. Conf. Automated Deduction*, 1990, pp. 162–177.
[18] T. Aoto, "Rewriting induction using termination checker," in *Proc. JSSST 24th Annual Conference*, 3C-3, 2007. (in Japanese)
[19] S. Shimazu, T. Aoto, and Y. Toyama, "Automated lemma generation for rewriting induction with disproof," *Computer Software*, vol. 26, no. 2, 2009, pp. 41–55. (in Japanese)
[20] C. Ji, M. Kurihara, and H. Sato, "Multi-context automated lemma generation for term rewriting induction with divergence detection," submitted for publication.
[21] D. Basin and T. Walsh, "Difference matching," in *Proc. 13th Int. Joint Conf. of Artificial Intelligence*, 1993, pp. 116–122.
[22] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Amaill, "Rippling: a heuristic for guiding inductive proofs," *Artificial Intelligence*, vol. 62, 1993, pp. 185–253.
[23] H. Sato and M. Kurihara, "On usefulness of syntactically complex lemmas in theory exploration for inductive theorems," in *Lecture Notes in Engineering and Computer Science: Proc. Int. MultiConference of Engineers and Computer Scientists, 2018*, pp. 489–492.
[24] C. Ji, H. Sato, and M. Kurihara, "An efficient implementation of multi-context algebraic reasoning systems with lazy evaluation," in *Lecture Notes in Engineering and Computer Science: Proc. Int. MultiConference of Engineers and Computer Scientists, 2015*, pp. 201–205.
[25] C. Ji, H. Sato, and M. Kurihara, "A new implementation of multi-context algebraic inductive theorem prover," in *Lecture Notes in Engineering and Computer Science: Proc. World Congress on Engineering and Computer Scientists, 2015*, pp. 109–114.
[26] H. Sato and M. Kurihara, "Recognition of normal forms with tree automata for inductive theorem proving," in *Proc. Science and Information Conference*, 2013, pp. 524–528.