International Journal of Innovative Computing, Information and Control
Volume **18**, Number **1**, February **2022**

 ISSN 1349-4198
pp. 315–330

COMPUTING LOGICALLY WITH GENERATION OF EQUIVALENT TRANSFORMATION RULES

Kiyoshi Akama[1] and Ekawit Nantajeewarawat[2,*]

[1]Information Initiative Center
Hokkaido University
Kita 11, Nishi 5, Kita-ku, Sapporo, Hokkaido 060-0811, Japan
akama@iic.hokudai.ac.jp

[2]School of Information, Computer and Communication Technology
Sirindhorn International Institute of Technology
Thammasat University
99 Moo 18, Km. 41 on Paholyothin Highway, Khlong Luang, Pathum Thani 12120, Thailand
*Corresponding author: ekawit@siit.tu.ac.th

Received July 2021; revised November 2021

Abstract. *A class of model-intersection problems (MI problems) and computation by equivalent transformation (ET) rules have been introduced. ET computation is a new concept of logical computation and overcomes the incompleteness of conventional inference-based methods. We take a query-answering problem that cannot be solved by unfolding alone. To overcome the difficulty, we introduce a class of specialized rewriting rules, and generate several ET rules, consisting of one-head rules and a multi-head rule. We prove the correctness of these rules. The basic idea behind construction of many one-head rules is restricted unfolding, i.e., the unfolding rule is restricted by specializing the head atom to be replaced. The basic idea behind construction of the two-head rule is inductive accumulation of restricted rules. ET computation with rule generation extends the most basic concepts of conventional logic and computation. Rule generation is invoked in the ET computation of solving a given problem. ET transformation and rule generation are combined in one computation. This is an important extension of the concept of logical computation.*

Keywords: Model-intersection problem, Equivalent transformation, Rewriting rule, ET rule, Single-head rule, Multi-head rule, Rule generation, Correctness

1. Introduction.

1.1. Inference as computation.

Proof problems have long been the main target for logical problem solving. A problem in this class is a "yes/no" problem concerning with checking whether one logical formula is a logical consequence of another logical formula. A resolution-based proof procedure for logical formulas was invented [1, 2, 3, 4], based on which automated theorem proving has been extensively investigated [5, 6, 7, 8, 9]. Meanwhile, the importance of another class of problems, called query-answering problems (QA problems), has been increasingly recognized [10, 11, 12, 13]. A QA problem is an "all-answers finding" problem concerning with finding all ground instances of a query atomic formula that are logical consequences of a given logical formula. Success of the research on proof problems and QA problems has motivated the integration of logical or symbolic reasoning into neural network architectures in the deep learning community [14, 15, 16, 17, 18, 19, 20, 21, 22, 23].

DOI: 10.24507/ijicic.18.01.315

SLD resolution has been a main method for solving proof problems and QA problems. SLD resolution for proof problems was constructed on the resolution principle [2], and was proved to be sound and complete for the class of conventional clauses. Being inspired by the resolution proof method, SLD resolution for solving QA problems was invented [10, 11, 12, 13]. Soundness and completeness theorems were established [24, 25]. Logical computation has been regarded as inference in the conventional clause space. In the name of inference by resolution, SLD resolution integrates solution methods for proof problems and QA problems.

However, integration of solution methods for proof problems and QA problems by SLD resolution is not successful. Many logical problems, e.g., pal-pal proof/QA problems [26] and Agatha proof/QA problems [27], cannot be solved by SLD resolution. The completeness theorem developed for SLD resolution is useless for these problems. SLD resolution is incomplete both for the class of all proof problems that are defined by using first-order formulas, and for the class of all QA problems that are defined by using first-order formulas. The conventional computation theory based on SLD resolution has such a theoretical limitation.

1.2. **Equivalent transformation as computation.** To overcome the limitation of conventional methods based on inference within the first-order formula space, we introduced a class of model-intersection problems (MI problems) and computation by equivalent transformation rules (ET rules) [28]. The main advantages of the method are the extension of a class of logical problems and computation by many ET rules.

- Formalization: MI problems in this paper form a superset of proof problems and QA problems on first-order logic.
- Computation: Various ET rules can be used, which can solve more problems.

Formalization as MI problems is of fundamental importance to establish a general solution method to solve all deductive problems on first-order formulas. With the power of more ET rules, the ET-based theory can be successfully applied to more proof problems and QA problems [27, 29, 30, 31, 32].

1.3. **Rule generation as computation.** Assuming that we formalize problems as MI problems, the set of all problems that can be solved successfully by using a set R of ET rules with some control is called a *finitely solvable area* with respect to R, and is denoted by $FSA(R)$. Since FSA is monotonic with R, increasing R by rule generation is of fundamental importance in logical problem solving. We show in Section 3 that there is some problem $prob$ and a rule r such that $prob \notin FSA(\{unfolding\})$ and $prob \in FSA(\{unfolding\} \cup \{r\})$, which shows that (1) conventional resolution-based methods are incomplete, and (2) by rule generation, there is a chance to solve more problems.

It follows that in order to solve an MI problem $\langle Cs, \varphi \rangle$, we generate many useful rules one by one ($i = 1, 2, \ldots, n$) starting with an initial rule set R_0, and we have an increasing sequence of rule sets

$$R_0 \subset R_1 \subset R_2 \subset \cdots \subset R_n,$$

at the end of which, we may solve the given problem successfully:

$$\langle Cs, \varphi \rangle \in FSA(R_n).$$

Together with equivalent transformation, rule generation is regarded as computation.

1.4. **Extending rule generation on definite clauses.** Introduction of rewriting rules and methods of rule generation were discussed in the context of ET computation with the minimal model semantics of definite clauses [33]. The main purpose of rule generation in the research is to develop a theory of program synthesis, where a program is a set of

ET rules. This is a sharp contrast to logic programs in the context of logic programming [34, 35]. A typical program synthesis method is called the squeeze method [26], where program transformation and rule generation are combined into one computation. ET rule generation from a logical formula, called a logical equivalence, has been introduced [36, 37], and a correctness proof for a class of logical equivalences by ET-based bidirectional search was developed for rule generation [38]. The underlying semantics of these theories is the minimal models of definite clauses, and knowledge representation is restricted to definite clauses. It is important future work to bridge the gap between the minimal model semantics and the all-model semantics in this paper, and to extend the previous theory of rule generation to the one on the all-model semantics with arbitrary (non-definite) clauses.

1.5. **Organization.** The rest of the paper is organized as follows. Section 2 defines rewriting rules and a syntax of a class of rewriting rules. Section 3 takes a QA problem, formalizes it as an MI problem, shows a solution of the problem by ET rules, and introduces the concept of computation based on rule generation. Section 4 discusses a class of restricted unfolding rules, and explains rule generation of single-head rules based on restricted unfolding. Section 5 explains rule generation of a two-head rule by proving the correctness of a group of two-head rules inductively. Section 6 provides conclusions.

2. Rewriting Rules.

We define rewriting rules and a syntax of a class of rewriting rules that is useful for generation of various ET rules.

2.1. **Model-intersection problems on clauses.** A *model-intersection problem* (for short, *MI problem*) is a pair $\langle Cs, \varphi \rangle$, where Cs is a clause set and φ is a mapping from $pow(\mathcal{G}_{\rm u})$ to some set W, where $\mathcal{G}_{\rm u}$ is the set of all ground user-defined atoms [28]. The mapping φ is called an *extraction mapping*. The answer to this problem, denoted by $ans_{\rm MI}(Cs, \varphi)$, is defined by

$$ans_{\rm MI}(Cs, \varphi) = \varphi\left(\bigcap Models(Cs)\right),$$

where $\bigcap Models(Cs)$ is the intersection of all models of Cs. Intuitively, this MI problem is to find some element $w \in W$ that is determined by the intersection of all models of Cs. The set W is arbitrarily chosen as the co-domain of φ. Note that when $Models(Cs)$ is the empty set, $\bigcap Models(Cs) = \mathcal{G}_{\rm u}$.

2.2. **Rewriting rules.** A rewriting rule r is a partial mapping to transform clause sets.

Definition 2.1. *A rewriting rule r is model-preserving iff if $(Cs, Cs') \in r$, then $Models(Cs) = Models(Cs')$.*

Definition 2.2. *A rewriting rule r is model-intersection preserving iff if $(Cs, Cs') \in r$, then $\bigcap Models(Cs) = \bigcap Models(Cs')$.*

Obviously, if r is model-preserving, then r is model-intersection preserving.

Proposition 2.1. *Assume that a rule r is model-intersection preserving. If $(Cs, Cs') \in r$, then for any extraction mapping φ,*

1) $\varphi(\bigcap Models(Cs)) = \varphi(\bigcap Models(Cs'))$, and
2) $ans_{\rm MI}(Cs, \varphi) = ans_{\rm MI}(Cs', \varphi)$.

Proof: Assume that $(Cs, Cs') \in r$. Since r is model-intersection preserving, $\bigcap Models(Cs) = \bigcap Models(Cs')$. Hence $ans_{\rm MI}(Cs, \varphi) = \varphi(\bigcap Models(Cs)) = \varphi(\bigcap Models(Cs')) = ans_{\rm MI}(Cs', \varphi)$. □

2.3. **A class of rewriting rules.** The syntax of rewriting rules used in this paper will now be described. The class of rewriting rules that can be represented by this syntax is an important subclass of rewriting rules defined in Section 2.2. Examples of rewriting rules are shown in Figure 1.

$$
\begin{aligned}
r_{pal}\colon\quad & pal(X) \Rightarrow rev(X, X).\\
r_{rev_1}\colon\quad & rev([A|X], Y) \Rightarrow rev(X, V), app(V, [A], Y).\\
r_{rev_2}\colon\quad & rev(X, Y), rev(X, Z) \Rightarrow \{=(Y, Z)\}, rev(X, Y).\\
r_{app_1}\colon\quad & app(X, Y, [A|Z]) \Rightarrow \{=(X, [\,]), =(Y, [A|Z])\};\\
& \qquad\qquad\quad \Rightarrow \{=(X, [A|V])\}, app(V, Y, Z).\\
r_{rev_3}\colon\quad & rev(X, [A|Y]) \Rightarrow \{=(X, [U|V])\}, rev(V, W), app(W, [U], [A|Y]).\\
r_{app_2}\colon\quad & app(X, [A], [B, C|Y]) \Rightarrow \{=(X, [B|V])\}, app(V, [A], [C|Y]).\\
r_{app_3}\colon\quad & app(X, [A], [B]) \Rightarrow \{=(X, [\,]), =(A, B)\}.\\
r_{rev_4}\colon\quad & rev([\,], X) \Rightarrow \{=(X, [\,])\}.
\end{aligned}
$$

FIGURE 1. Examples of rewriting rules

Usual atoms are used in a rule to specify patterns of user-defined atoms that appear in definite clauses. A user-defined atom *eq* gives the equality relation. In addition, atoms of a special kind, called *executable atoms*, are used to represent built-in operations. Arbitrary built-in deterministic operations, e.g., unification, arithmetic functions, and false operation, can be used as executable atoms. An executable atom *false* simply gives failure. An executable atom $pvar(t)$ outputs true if t is a usual variable. An executable atom $=(t, t')$ denotes the operation to find the most general unifier of terms t and t'. An executable atom is evaluated by some predetermined evaluator, and if the evaluation succeeds, it yields a substitution as the result; e.g., the evaluation of $=([1|X], [Y, 2, Z])$ yields $\{X/[2, Z], Y/1\}$.

A rewriting rule r in this paper takes the form

$$
\begin{aligned}
Hs, \{Cs\} \quad &\Rightarrow \quad \{Es_1\}, Bs_1;\\
&\cdots\\
&\Rightarrow \quad \{Es_n\}, Bs_n,
\end{aligned}
$$

where $n \geq 1$, Hs is a nonempty sequence of user-defined atoms, the Cs and Es_i are sequences of executable atoms, and the Bs_i are sequences of user-defined atoms. Note that a rule with $n = 0$ is not admitted. However, to denote a rule with $n = 0$ rule, we can use the *false* atom in a rule with $n = 1$ rule as follows:

$$Hs \quad \Rightarrow \quad \{false\}.$$

For each i $(1 \leq i \leq n)$, the pair $\langle \{Es_i\}, Bs_i \rangle$ is called a *body* of r, $\{Cs\}$ is called the *condition part*, and $\{Es_i\}$ and Bs_i are called the *execution part* and the *replacement part*, respectively. The condition part and each of the execution parts are optional. When Hs contains more than one atom, r is called a *multi-head rule*. It is called a *single-head rule* otherwise.

Given a definite clause C containing atoms $b_1, \ldots, b_m$, where $m \geq 1$, in its body, the rule r is *applicable* to C at $b_1, \ldots, b_m$ iff Hs matches these atoms by a substitution θ (i.e., $Hs\theta$ is the sequence $b_1, \ldots, b_m$), and the execution of the condition part succeeds. We assume that no specialization occurs in the execution of the condition part. To apply r to the clause C, the pattern-matching substitution θ is additionally required to instantiate

all variables that occur in r but not in Hs into distinct usual variables that do not occur in C. The application then replaces C with the clauses that are obtained as follows: for each i $(1 \leq i \leq n)$, if the evaluation of $Es_i\theta$ succeeds and yields a substitution σ, then a clause obtained from $C\sigma$ by replacing $b_1\sigma, \ldots, b_m\sigma$ with $Bs_i\theta\sigma$ is constructed.

3. **Computing with Rule Generation.** A QA problem is taken and is formalized as an MI problem. We solve QA problems by using rewriting rules whose syntax is given in Section 2. We also introduce rule generation, which is included into computation together with usual computation of repeated application of ET rules.

3.1. **Pal-pal problem.** Let $\mathcal{G}_t$ be the set of all ground terms. Assume as background knowledge a set D_0 consisting of the five definite clauses in Figure 2, where *pal*, *rev*, and *app* stand for "palindrome", "reverse", and "append", respectively. Let C be a definite clause:

$$ans(X) \leftarrow pal([1|X]), pal([2|X]),$$

which is intended to mean that "X is an answer if both $[1|X]$ and $[2|X]$ satisfy the definition of *pal*". Consider the problem "find all ground terms t such that $[1|t]$ and $[2|t]$ are palindromes", which is called a pal-pal QA problem, and is given by the set $D_0 \cup \{C\}$. The pal-pal QA problem is formalized as an MI problem to find

$$\varphi_1\left(\bigcap Models(D_0 \cup \{C\})\right),$$

where φ_1 is a mapping from $pow(\mathcal{G}_{\mathrm{u}})$ to $W = \mathcal{G}_t$ that is defined by

$$\varphi_1(G) = \{t \,|\, ans(t) \in G\}.$$

1) $pal(X) \leftarrow rev(X, X)$
2) $rev([\,], [\,]) \leftarrow$
3) $rev([A|X], Y) \leftarrow rev(X, R), app(R, [A], Y)$
4) $app([\,], X, X) \leftarrow$
5) $app([A|X], Y, [A|Z]) \leftarrow app(X, Y, Z)$

FIGURE 2. Definite clauses defining the predicates *pal*, *rev*, and *app*

3.2. **Computation.** Let $Cs = D_0 \cup Q$, where D_0 is the set of definite clauses in Figure 2, and Q is a set of definite clauses with the predicate *ans* in the head. A clause in Q is called a goal clause. For example, the pal-pal problem is represented by $D_0 \cup Q$, where $Q = \{C_1\}$. By unfolding, C_1 is sequentially changed as follows:

$C_1 = (ans(X) \leftarrow pal([1|X]), pal([2|X]))$
$C_2 = (ans(X) \leftarrow rev([1|X], [1|X]), pal([2|X]))$
$C_3 = (ans(X) \leftarrow rev([1|X], [1|X]), rev([2|X], [2|X]))$
$C_4 = (ans(X) \leftarrow rev(X, R1), app(R1, [1], [1|X]), rev([2|X], [2|X]))$
$C_5 = (ans(X) \leftarrow rev(X, R1), app(R1, [1], [1|X]), rev(X, R2), app(R2, [2], [2|X]))$

3.3. **Unsuccessful result without multi-head rules.** Consider the set of all problems that can be solved successfully by formalizing them as MI problems, and by using a set R of ET rules with some control, which is denoted by $FSA(R)$ and is called a *finitely solvable area* with respect to R.

One typical improvement is obtained by creation and accumulation of ET rules. Let R be a set of ET rules. We try to maximize $FSA(R)$ by increasing R.

The goal of computation

$$D_0 \cup Q_1, D_0 \cup Q_2, \ldots, D_0 \cup Q_n$$

is to reach final states where Q_n is a set of unit clauses. Q_n can be an empty set. If Q_n contains a definite clause, its body needs to be empty.

If we use only unfolding with respect to D_0, Q never reaches a goal, i.e.,

$$\text{Pal-pal QA problem} \notin \mathit{FSA}(\{\mathit{unfolding}\}).$$

Is the resolution rule useful to overcome this limitation? The answer is negative. Even with the help of resolution, we cannot reach a goal:

$$\text{Pal-pal QA problem} \notin \mathit{FSA}(\{\mathit{unfolding}, \mathit{resolution}\}).$$

Unfolding and resolution replace only one body atom in Q_i, and they cannot detect inconsistency among atoms in the bodies in Q_i. Is the factoring rule useful to overcome this limitation? The answer is also negative. Even with the help of factoring, we cannot reach a goal:

$$\text{Pal-pal QA problem} \notin \mathit{FSA}(\{\mathit{unfolding}, \mathit{resolution}, \mathit{factoring}\}).$$

Compared with unfolding and resolution, factoring deals with two atoms in a body of a clause in Q_i, and it can solve constraints among the atoms. However, factoring never removes an input clause to be transformed, which is a common disadvantage of inference rules such as resolution and factoring.

3.4. **Successful result with a two-head rule.** To solve the pal-pal QA problem, we used one-head rules and a two-head rule. The details of the solution are explained in Table 1, Table 2, and Figure 1. Let Rules_1 be the set of all rewriting rules in Figure 1. By experiment, we found that Rules_1 can solve the pal-pal QA problem, i.e.,

$$\text{Pal-pal QA problem} \in \mathit{FSA}(\mathit{Rules}_1).$$

Since the ET rules in Rules_1 except r_{rev_2} are unfolding-based rules, we know

$$\text{Pal-pal QA problem} \in \mathit{FSA}(\{\mathit{unfolding}\} \cup \{r_{rev_2}\}).$$

We define a mapping ans_0 by

$$\mathit{ans}_0(\mathit{Cs}) = \varphi_1\left(\bigcap \mathit{Models}(\mathit{Cs})\right).$$

Then, $\mathit{ans}_0(\mathit{Cs}) = \{t \mid t = s\theta \in \mathcal{G}_t, \mathit{ans}(s) = \mathit{head}(C), C \in \mathit{Cs}, \theta \in \mathcal{S}\}$ if all ans-clauses in Cs are facts (positive unit clauses), i.e., $|\mathit{head}(C)| = 1$ and $\mathit{body}(C) = \emptyset$ for any $C \in \mathit{Cs}$; and otherwise undefined. By ans_0, the answer will be obtained by

$$\mathit{ans}_0(\mathit{Cs}_{11}) = [\,],$$

where Cs_{11} is the last set of clauses corresponding to prb_{11} in Table 1.

3.5. **Rule generation in computation.** In Table 2, ET rules are generated in the order of the list $[r_{pal}, r_{rev_1}, r_{rev_2}, r_{app_1}, r_{rev_3}, r_{app_2}, r_{app_3}, r_{rev_4}]$. Let R_0 be an empty set. For $i = 1, 2, \ldots$, let R_i be the set of all ET rules consisting of the first one to the i-th one in the list. R_i $(i = 1, 2, \ldots)$ are monotonically increasing.

More generally, we generate many useful rules one by one $(i = 1, 2, \ldots, n)$, starting with an initial rule set R_0. Then we have an increasing sequence of rule sets

$$R_0 \subset R_1 \subset R_2 \subset \cdots$$

until we can solve the given problem $\langle \mathit{Cs}, \varphi \rangle$ successfully by R_n:

$$\langle \mathit{Cs}, \varphi \rangle \in \mathit{FSA}(R_n).$$

We regard rule generation as computation. Rule generation is invoked in ET computation for solving a given problem. We try to increase the rule set to solve the problem.

TABLE 1. Transformation of problems

Problem	Problem representation	Rule applied
prb_1	$\{ans(X) \leftarrow \underline{pal([1\|X])}, pal([2\|X])\}$	r_{pal}
prb_2	$\{ans(X) \leftarrow rev([1\|X],[1\|X]), \underline{pal([2\|X])}\}$	r_{pal}
prb_3	$\{ans(X) \leftarrow \underline{rev([1\|X],[1\|X])}, rev([2\|X],[2\|X])\}$	r_{rev_1}
prb_4	$\{ans(X) \leftarrow rev(X,A1), app(A1,[1],[1\|X]), \underline{rev([2\|X],[2\|X])}\}$	r_{rev_1}
prb_5	$\{ans(X) \leftarrow \underline{rev(X,A1)}, app(A1,[1],[1\|X]), \underline{rev(X,A2)}, app(A2,[2],[2\|X])\}$	r_{rev_2}
prb_6	$\{ans(X) \leftarrow rev(X,A1), \underline{app(A1,[1],[1\|X])}, app(A1,[2],[2\|X])\}$	r_{app_1}
prb_7	$\{ans([\,]) \leftarrow rev([\,],[\,]), app([\,],[2],[2]),$ $ans(X) \leftarrow \underline{rev(X,[1\|A3])}, app(A3,[1],X), app([1\|A3],[2],[2\|X])\}$	r_{rev_3}
prb_8	$\{ans([\,]) \leftarrow rev([\,],[\,]), app([\,],[2],[2]),$ $ans([A4\|A5]) \leftarrow rev(A5,A6), app(A6,[A4],[1\|A3]),$ $app(A3,[1],[A4\|A5]), \underline{app([1\|A3],[2],[2,A4\|A5])}\}$	r_{app_2}
prb_9	$\{ans([\,]) \leftarrow rev([\,],[\,]), \underline{app([\,],[2],[2])}\}$	r_{app_3}
prb_{10}	$\{ans([\,]) \leftarrow \underline{rev([\,],[\,])}\}$	r_{rev_4}
prb_{11}	$\{ans([\,]) \leftarrow\}$	–

TABLE 2. An example of rule generation

Iteration	Problem	Atom(s) selected	Atom(s) pattern	Rule obtained	Priority assigned
1st	prb_1	$pal([1\|X])$	$pal(X)$	r_{pal}	PR-1
2nd	prb_3	$rev([1\|X],[1\|X])$	$rev([A\|X],Y)$	r_{rev_1}	PR-1
3rd	prb_5	$rev(X,A1), rev(X,A2)$	$rev(X,Y), rev(X,Z)$	r_{rev_2}	PR-1
4th	prb_6	$app(A1,[1],[1\|X])$	$app(X,Y,[A\|Z])$	r_{app_1}	PR-2
5th	prb_7	$rev(X,[1\|A3])$	$rev(X,[A\|Y])$	r_{rev_3}	PR-1
6th	prb_8	$app([1\|A3],[2],[2,A4\|A5])$	$app(X,[A],[B,C\|Y])$	r_{app_2}	PR-1
7th	prb_9	$app([\,],[2],[2])$	$app(X,[A],[B])$	r_{app_3}	PR-1
8th	prb_{10}	$rev([\,],[\,])$	$rev([\,],X)$	r_{rev_4}	PR-1
9th	prb_{11}	–	–	–	–

ET transformation and rule generation are combined in one computation. This is an important extension of the concept of logical computation.

3.6. **Rule generation and correctness of rules.**

3.6.1. *ET theory.* In the ET theory, a set of clauses is transformed by ET rules preserving models and/or model-intersection. The ET framework considers a set, $Cs = Q \cup D$, of arbitrary clauses, and a set R of ET rules. Q is called a set of query clauses and D is called a knowledge base. Rule generation is to generate a new rule r to be stored in a rule set, by which R is changed into $R \cup \{r\}$. D is a declarative knowledge base (formulas), while R is a procedural knowledge base (rules). Rule generation is basically defined as making procedural knowledge (rules) from declarative knowledge (formulas).

3.6.2. *Proof theory.* In the conventional logic (e.g., [1, 2, 4, 39]), a set of clauses is transformed by inference rules such as the resolution rule preserving satisfiability. Computation in logic is defined as repeated application of inference rules. One specific class of transformation rules, i.e., resolution, was used. Utility of inventing and using various rules has not been recognized. The main research direction is to control generation of redundant clauses, not to invent new inference rules. Semantic resolution, lock resolution, and linear resolution were proposed to prevent large numbers of useless clauses from being generated [1].

3.6.3. *SLD resolution for QA problems.* Based on the proof theory, SLD resolution for solution of a class of QA problems was introduced [10, 12]. SLD resolution for QA problems considers two sets, Q and D, of definite clauses. Q is called a set of query clauses and D is called a knowledge base. A set Q of query clauses is transformed into another set Q' of query clauses by a definite clause in D using resolution. Assuming only one inference rule (resolution), each definite clause in D is given a procedural meaning. D is regarded as a logic program. By the slogan "What = How" of the declarative programming paradigm, D is a declarative knowledge base and a procedural knowledge base at the same time. More precisely, D is a declarative knowledge base (formulas), and D is associated with a procedure (a rule). Only very small part of rules correspond to D (formulas). We need to be aware that "What = How" gives a "degenerated" view of logical computation, where utility of inventing and using various rules has not been well recognized. Since the full concept of rules is not established, logic programming fails to produce the concept of (full power of) rule generation.

3.6.4. *Rule-based systems.* Rule-based systems are also known as production systems or expert systems [40, 41, 42, 43, 44]. Instead of representing knowledge as a set of true ground atoms, a rule-based system represents knowledge in terms of a set of rules that tells what to do in different situations. Rules are written by mimicing the reasoning of a human expert in solving a knowledge intensive problem. There is no theoretical framework (correctness) to discuss rule generation formally.

3.6.5. *Model-intersection preservation.* A rewriting rule r is model-intersection preserving iff $\bigcap Models(Cs) = \bigcap Models(Cs')$ for any Cs and Cs' that satisfy $(Cs, Cs') \in r$. All model-preserving rules are also model-intersection preserving rules. In this sense, model-intersection preservation is weaker than model preservation, and more rules can be used than logically-equivalent rules in usual logic. All model-intersection problems can be equivalently transformed by rewriting rules that preserve model-intersection, without considering their extraction mappings. We will prove in Section 4 and Section 5 that five rules in Figure 1 are model-intersection preserving. Among them, the correctness of the two-head rule is inductively proved, which clearly suggests inductive construction of the rule.

3.6.6. *Rule generation for increasing solvability.* Rule generation is fundamental for solving problems since it produces programs from the problems. SLD resolution for QA problems can be regarded as a solver that uses only the general unfolding rules. By generating various rules that are less general and more efficient, we can solve more problems in a shorter time. One of the basic techniques is (1) to generate specialized ET rules, and (2) to control computation flexibly based on the applicability and priority of rules [31].

4. **Making One-Head Rewriting Rules.** We introduce a class of restricted unfolding rules, and explain rule generation of single-head rules based on restricted unfolding.

4.1. **Unfolding operation on $\mathbf{CLS_B}$.** Let $\mathrm{CLS_B}$ be the set of all clauses that possibly have built-in constraint atoms in their right-hand sides. Given a clause C, let $lhs(C)$ and $rhs(C)$ denote the set of all atoms in the left-hand side of C and the set of all atoms in the right-hand side of C, respectively. The unfolding operation on $\mathrm{CLS_B}$ is formulated below. Assume that Cs is a set of clauses in $\mathrm{CLS_B}$, D is a set of definite clauses in $\mathrm{CLS_B}$, and occ is an occurrence of an atom b in the right-hand side of a clause C in Cs. By unfolding Cs using D at occ, Cs is transformed into

$$(Cs - \{C\}) \cup \left(\bigcup\{resolvent(C, C', b) \mid C' \in D\}\right),$$

where for each $C' \in D$, $resolvent(C, C', b)$ is defined as follows, assuming that ρ is a renaming substitution for usual variables such that C and $C'\rho$ have no usual variable in common:

1) If b and $head(C'\rho)$ are not unifiable, then $resolvent(C, C', b) = \emptyset$.
2) If they are unifiable, then $resolvent(C, C', b) = \{C''\}$, where C'' is the clause obtained from C and $C'\rho$ as follows, assuming that θ is the most general unifier of b and $head(C'\rho)$:
   (a) $lhs(C'') = lhs(C\theta)$.
   (b) $rhs(C'') = (rhs(C\theta) - \{b\theta\}) \cup body(C'\rho\theta)$.

The resulting clause set is denoted by $\textsc{Resol}(Cs, D, C, occ, b)$.

$$
\begin{aligned}
rule_1 &: app([\,], Y, Z) \Rightarrow eq(Y, Z).\\
rule_2 &: app([A|X], Y, Z) \Rightarrow eq(Z, [A|M]), app(X, Y, M).\\
rule_3 &: app(X, Y, Z)\\
&\qquad \Rightarrow eq(X, [\,]), eq(Y, Z);\\
&\qquad \Rightarrow eq(X, [A|P]), eq(Z, [A|Q]), app(P, Y, Q).\\
rule_4 &: app(X, [D], Y), app(X, [D], Z) \Rightarrow eq(Y, Z), app(X, [D], Y).\\
rule_5 &: app(X, [D], Z), app(Y, [D], Z) \Rightarrow eq(X, Y), app(X, [D], Z).\\
rule_6 &: rev([\,], Y) \Rightarrow eq(Y, [\,]).\\
rule_7 &: rev([A|X], Y) \Rightarrow rev(X, M), app(M, [A], Y).\\
rule_8 &: rev(P, Q)\\
&\qquad \Rightarrow eq(P, [\,]), eq(Q, [\,]);\\
&\qquad \Rightarrow eq(P, [A|X]), rev(X, Y), app(Y, [A], Q).\\
rule_9 &: eq(P, Q), \{pvar(P)\} \Rightarrow \{=(P, Q)\}.\\
rule_{10} &: eq(P, Q), \{pvar(Q)\} \Rightarrow \{=(P, Q)\}.\\
rule_{11} &: eq([\,], [A|Y]) \Rightarrow \{false\}.\\
rule_{12} &: eq([A|Y], [\,]) \Rightarrow \{false\}.\\
rule_{13} &: eq([A|X], [B|Y]) \Rightarrow eq(A, B), eq(X, Y).
\end{aligned}
$$

FIGURE 3. Transformation rules R_{base}

4.2. **Restricted unfolding.** Let B be an atom. Restricted unfolding (or specialized unfolding) with respect to B is denoted by $\textsc{Unfold}(B)$, and is defined as the rewriting rule that is obtained from the unfolding rule by restricting each selected body atom to be an instance of B. More precisely, $\textsc{Unfold}(B)$ transforms $\langle Cs, \varphi\rangle$ into $\langle Cs', \varphi\rangle$ as follows:

1) Let $\langle Cs, \varphi\rangle$ be an MI problem on $\mathrm{CLS_B}$.
2) Take D, C, occ, and b such that
   (a) D is a subset of definite clauses in Cs,
   (b) C is a clause in $Cs - D$,
   (c) occ is an occurrence of a body atom b in C,

(d) b is an instance of B, i.e., there is a substitution θ such that $b = B\theta$.

3) $Cs' = (Cs - \{C\}) \cup \textsc{Resol}(Cs, D, C, occ, b)$.

The rule $rule_3$ in Figure 3 is a restricted rule with respect to $app(X, Y, Z)$. The rule $rule_1$ in Figure 3 is a restricted rule with respect to $app([\,], Y, Z)$. All rules in Figure 1 except the rule r_{rev_2} are restricted rules of this type.

4.3. **A rule for $pal(X)$.** Consider $\textsc{Unfold}(pal(X))$. For the predicate pal, we have one definite clause:

$$pal(X) \leftarrow rev(X, X).$$

Since an atom $pal(t_X)$ in the body of a goal clause is replaced with $rev(t_X, t_X)$, we name the rule r_{pal} and represent it as

$$pal(X) \Rightarrow rev(X, X).$$

In the following we use a simplified notation of a definite clause as follows. Let C_1 and C_2 be definite clauses. Let H and A_i be atoms. Let B be a set of atoms. Then the following description

$C_1 : H \leftarrow A_1, B,$
$C_2 : H \leftarrow A_1, A_2, B,$

denotes the following two conditions, respectively.

- C_1 is a definite clause with the head H and the body $\{A_1\} \cup B$.
- C_2 is a definite clause with the head H and the body $\{A_1, A_2\} \cup B$.

Theorem 4.1. *The rule r_{pal} is model-intersection preserving.*

Proof: Let C_1 and C_2 be

$C_1 : H \leftarrow pal(t_x), B,$
$C_2 : H \leftarrow rev(t_x, t_x), B,$

where H is an atom, B is a set of atoms, and t_x is a term. Since we know that $\forall X : (pal(X) \leftrightarrow rev(X, X))$, $pal(t_x)$ in C_1 can be replaced with $rev(t_x, t_x)$ preserving models to obtain C_2. Hence, r_{pal} preserves model-intersection, i.e., $\bigcap Models(Cs \cup \{C_1\}) = \bigcap Models(Cs \cup \{C_2\})$. □

4.4. **A rule for $rev([\,], X)$.** Consider $\textsc{Unfold}(rev([\,], X))$. The following definite clause in D_0 is used at unfolding:

$$rev([\,], [\,]) \leftarrow .$$

Assume that there is an atom $rev([\,], t_X)$ in the body of a goal clause. Then, by unfolding, if the unification of t_X and $[\,]$ is successful, $rev([\,], t_X)$ is removed from the goal (with specialization by unification being applied), and otherwise, the goal clause itself is removed. We name the rule r_{rev_4} and represent it as

$$rev([\,], X) \Rightarrow \{=(X, [\,])\}.$$

Theorem 4.2. *The rule r_{rev_4} is model-intersection preserving.*

Proof: Let C_1 and C_2 be

$C_1 : H \leftarrow rev([\,], t_X), B,$
$C_2 : H \leftarrow eq(t_X, [\,]), B.$

Since $\forall X : (rev([\,], X) \leftrightarrow eq(X, [\,]))$, the atom $rev([\,], t_X)$ in C_1 can be replaced with the atom $eq(t_X, [\,])$. Hence, $\bigcap Models(Cs \cup \{C_1\}) = \bigcap Models(Cs \cup \{C_2\})$, which implies that $rev([\,], X) \Rightarrow eq(X, [\,])$ is an ET rule. Hence we know that r_{rev_4} is model-intersection preserving, and is an ET rule. □

4.5. **A rule for $rev(X, [B|Y])$.** Consider UNFOLD($rev(X, [B|Y])$). The following definite clause in D_0 is used at unfolding:

$$rev([A|R], V) \leftarrow rev(R, W), app(W, [A], V).$$

Assume that there is an atom $rev(t_X, [t_B|t_Y])$ in the body of a goal clause. Then by unfolding, if t_X is unifiable with $[A|R]$, and a most general unifier is $\{A/t_A, R/t_R\}$, then $rev(t_X, [t_B|t_Y])$ is replaced with the conjunction of $rev(t_R, t_W)$ and $app(t_W, [t_A], [t_B|t_Y])$. We call the rule r_{rev_3} and represent it as

$$rev(X, [B|Y]) \Rightarrow \{=(X, [A|R])\}, rev(R, W), app(W, [A], [B|Y]).$$

Theorem 4.3. *The rule r_{rev_3} is model-intersection preserving, and is an ET rule.*

Proof: Since

$$\forall : (rev(X, [B|Y]) \leftrightarrow eq(X, [A|R]), rev(R, W), app(W, [A], [B|Y])),$$

the atom $rev(t_X, [t_B|t_Y])$ can be replaced with the atom set

$$\{eq(t_X, [t_A|t_R]), rev(t_R, t_W), app(t_W, [t_A], [t_B|t_Y])\}$$

in a clause body. Hence, r_{rev_3} is model-intersection preserving, and is an ET rule. □

4.6. **A two-body rule for $app(X, Y, [A|Z])$.** Consider UNFOLD($app(X, Y, [A|Z])$). The following definite clauses in D_0 are used at unfolding:

$app([\,], Y, Y) \leftarrow$.
$app([A|V], Y, [A|Z]) \leftarrow app(V, Y, Z)$.

Assume that there is an atom $app(t_X, t_Y, [t_A|t_Z])$ in the body of a goal clause. Then, by unfolding,

1) if t_X is unifiable with $[\,]$, t_Y is unifiable with $[t_A|t_Z]$, and a most general unifier is θ, then a clause is produced by removing the atom $app(t_X, t_Y, [t_A|t_Z])$ and by specializing by θ.
2) if t_X is unifiable with $[A|V]$, t_Y is unifiable with Y, $[t_A|t_Z]$ is unifiable with $[A|Z]$, and a most general unifier is $\theta = \{A/t_A, V/t_V, Y/t_Y, Z/t_Z\}$, then a clause is produced by replacing the atom $app(t_X, t_Y, [t_A|t_Z])$ with a new atom $app(t_V, t_Y, t_Z)$ and by specializing by θ.

We obtain a rule named r_{app_1}, which is represented by

$app(X, Y, [A|Z])$
$\Rightarrow \{=(X, [\,]), =(Y, [A|Z])\}$;
$\Rightarrow \{=(X, [A|V])\}, app(V, Y, Z)$.

Theorem 4.4. *The rule r_{app_1} is model-intersection preserving and is an ET rule.*

Proof: Since

$$\forall x, y, z, a : app(x, y, [a|z]) \leftrightarrow (eq(x, [\,]) \wedge eq(y, [a|z])) \vee \exists v : (eq(x, [a|v]) \wedge app(v, y, z)),$$

the rule r_{app_1} preserves model-intersection and is an ET rule. □

5. **Making Two-Head Rewriting Rules.** We explain rule generation of a two-head rule by proving the correctness of a group of two-head rules inductively.

5.1. **Prepared ET rules.** Let R_{base} be the set of all rules in Figure 3. Rules in R_{base} will be used for generation of ET rules in this section. Among the one-head rules in R_{base}, all rules with *app* or *rev* as head predicates, i.e., $rule_1$, $rule_2$, $rule_3$, $rule_6$, $rule_7$, $rule_8$, are restricted rules. The rules $rule_4$ and $rule_5$ can be constructed inductively in the same way explained in this section.

5.2. **Correctness of the two-head rule r_{rev_2}.** Recall the two-head rule r_{rev_2}:

$$rev(X,Y), rev(X,Z) \Rightarrow \{=(Y,Z)\}, rev(X,Y).$$

To prove that r_{rev_2} is model-intersection preserving, it suffices to prove that the following rule, named rr, is model-intersection preserving.

$$rev(X,Y), rev(X,Z) \Rightarrow eq(Y,Z), rev(X,Y).$$

Consider two clauses, C_a and C_b:

$C_a : H \leftarrow rev(t_x,t_y), rev(t_x,t_z), B$, and
$C_b : H \leftarrow rev(t_x,t_y), eq(t_y,t_z), B$,

where H is an *ans*-atom and B represents the rest body atoms. C_a is transformed by rr into C_b. A necessary and sufficient condition for the correctness (model-intersection preservation) of rr with respect to D_0 is

$$\bigcap Models(D_0 \cup Q \cup \{C_a\}) = \bigcap Models(D_0 \cup Q \cup \{C_b\})$$

for any set Q of *ans*-clauses. We prove this condition by finding a clause set Cs that satisfies

$$\bigcap Models(D_0 \cup Q \cup \{C_a\}) = \bigcap Models(D_0 \cup Q \cup Cs) = \bigcap Models(D_0 \cup Q \cup \{C_b\}).$$

Finding such a clause set at the confluent position is called a confluent search. Assume that each rule in R is model-intersection preserving. The result of a confluent search of C_a and C_b is denoted by $C_a \xrightarrow{R} Cs$ and $C_b \xrightarrow{R} Cs$. Consider a more general case: $C_a \xrightarrow{R} Cs$ and $C_b \xrightarrow{R} Cs'$. We also call it a confluent search if $\bigcap Models(D_0 \cup Q \cup Cs) = \bigcap Models(D_0 \cup Q \cup Cs')$.

5.3. **Stratification.** Let r_i be a rule

$$rev(X,Y), rev(X,Z), \{length(X,i)\} \Rightarrow eq(Y,Z), rev(X,Y),$$

where $length(X,i)$ is an atom stating that X is a list of length i. Let R_∞ be the set of rules such that

$$R_\infty = \{r_i \mid i = 0,1,2,\ldots\}.$$

Based on the fact that each rule in R_{base} is model-intersection preserving, we will show that each rule in R_∞ is model-intersection preserving.

5.4. **Base case ($k = 0$).** Assuming that each rule in R_{base} is model-intersection preserving, we prove that r_0 is model-intersection preserving. Let

$C_{a0} : H \leftarrow rev([\,],t_y), rev([\,],t_z), B$,
$C_{b0} : H \leftarrow rev([\,],t_y), eq(t_y,t_z), B$.

C_{a0} is transformed as follows:

$C_{a0} : H \leftarrow rev([\,],t_y), rev([\,],t_z), B$
by $rule_6$ (rev-[ ] rule) in R_{base}
$C_{c0} : H \leftarrow eq([\,],t_y), eq([\,],t_z), B$

C_{b0} is transformed as follows:

$C_{b0} : H \leftarrow rev([\,],t_y), eq(t_y,t_z), B$
by $rule_6$ (rev-[ ] rule) in R_{base}
$C_{d0} : H \leftarrow eq([\,],t_y), eq(t_y,t_z), B$

We have $C_{a0} \xrightarrow{R_{base}} C_{c0}$ and $C_{b0} \xrightarrow{R_{base}} C_{d0}$. The following two equalities hold.

$\bigcap Models(D_0 \cup Q \cup \{C_{a0}\}) = \bigcap Models(D_0 \cup Q \cup \{C_{c0}\})$.
$\bigcap Models(D_0 \cup Q \cup \{C_{b0}\}) = \bigcap Models(D_0 \cup Q \cup \{C_{d0}\})$.

Since $\bigcap Models(D_0 \cup Q \cup \{C_{c0}\}) = \bigcap Models(D_0 \cup Q \cup \{C_{d0}\})$, we have

$$\bigcap Models(D_0 \cup Q \cup \{C_{a0}\}) = \bigcap Models(D_0 \cup Q \cup \{C_{b0}\}),$$

which means that r_0 is model-intersection preserving.

5.5. **Inductive case ($k \geq 1$).** Assume that each rule in R_{base} is model-intersection preserving. Assuming that r_k is model-intersection preserving, we prove that r_{k+1} is model-intersection preserving. In the following, t_x is a list of length k. M and N are new variables that do not appear in the other part of the clauses. They satisfy $H\{M/N\} = H$, and $B\{M/N\} = B$.

We try a confluent search for C^a_{k+1} and C^b_{k+1}, where

$C^a_{k+1} : H \leftarrow rev([t_a|t_x], t_y), rev([t_a|t_x], t_z), B,$

$C^b_{k+1} : H \leftarrow rev([t_a|t_x], t_y), eq(t_y, t_z), B.$

C^a_{k+1} is transformed into C^c_{k+1} as follows:

$C^a_{k+1} : H \leftarrow rev([t_a|t_x], t_y), rev([t_a|t_x], t_z), B$
by $rule_7$ (rev-unfold rule) in R_{base}
$H \leftarrow rev(t_x, M), app(M, [t_a], t_y), rev(t_x, N), app(N, [t_a], t_z), B$
by r_k (rule for the inductive assumption)
$H \leftarrow rev(t_x, M), app(M, [t_a], t_y), eq(M, N), app(N, [t_a], t_z), B$
by $rule_9$ (equality constraint solving) in R_{base}
$H \leftarrow rev(t_x, M), app(M, [t_a], t_y), app(M, [t_a], t_z), B$
by $rule_4$ (app-eq rule) in R_{base}
$C^c_{k+1} : H \leftarrow rev(t_x, M), app(M, [t_a], t_y), eq(t_y, t_z), B$

C^b_{k+1} is transformed into C^d_{k+1} as follows:

$C^b_{k+1} : H \leftarrow rev([t_a|t_x], t_y), eq(t_y, t_z), B$
by $rule_7$ (rev-unfold rule) in R_{base}
$C^d_{k+1} : H \leftarrow rev(t_x, M), app(M, [t_a], t_y), eq(t_y, t_z), B$

Hence we have $C^a_{k+1} \xrightarrow{R'_k} C^c_{k+1}$ and $C^b_{k+1} \xrightarrow{R'_k} C^d_{k+1}$, where

$$R'_k = R_{base} \cup \{r_i \mid i = 0, 1, 2, \ldots, k\}.$$

It follows that

$$\bigcap Models(D_0 \cup Q \cup \{C^a_{k+1}\}) = \bigcap Models(D_0 \cup Q \cup \{C^c_{k+1}\}),$$
$$\bigcap Models(D_0 \cup Q \cup \{C^b_{k+1}\}) = \bigcap Models(D_0 \cup Q \cup \{C^d_{k+1}\}).$$

Since $C^c_{k+1} = C^d_{k+1}$, we have

$$\bigcap Models(D_0 \cup Q \cup \{C^a_{k+1}\}) = \bigcap Models(D_0 \cup Q \cup \{C^b_{k+1}\}),$$

which means that r_{k+1} is model-intersection preserving.

5.6. **Correctness of the rule r_{rev_2}.** Assume that each rule in R_{base} is model-intersection preserving.

Theorem 5.1. *Each rule in R_∞ is model-intersection preserving.*

Proof: We prove this theorem by induction.

- (Base case) By the result in Section 5.4, r_0 is model-intersection preserving.
- (Inductive case) Assume that each rule in r_k is model-intersection preserving. By the result in Section 5.5, the rule r_{k+1} is model-intersection preserving. □

Theorem 5.2. *The rule r_{rev_2} is model-intersection preserving and is an ET rule.*

Proof: By Theorem 5.1, each rule in R_∞ is model-intersection preserving. Hence r_{rev_2} is model-intersection preserving, and is an ET rule. □

6. **Conclusions.** ET computation is a new concept of logical computation and overcomes the incompleteness of conventional inference-based methods. One of the most important characteristics of the ET-based computation theory is the existence of the concept of rewriting rule, which is a sharp contrast to the conventional logical computation, where a program is given by procedural reading of specific logical formulas, e.g., definite clauses [10].

Since rewriting rules have rich expressive power, ET computation enables us to consider rule generation to the full extent. When new rules are generated, the power of computation increases monotonically and more problems can be solved. Rule generation is the key point of solving logical problems. Rule generation is invoked in ET computation for solving a given problem. ET transformation and rule generation are combined in one computation. This is an important extension of the concept of logical computation.

We defined the concept of rewriting rule and a syntax of a class of rewriting rules. A query-answering problem that cannot be solved by unfolding alone was taken to explain the new concept of computation. We introduced a class of specialized rewriting rules, and generated several ET rules, consisting of one-head rules and a multi-head rule. We proved the correctness of these rules, i.e., these rules preserve model intersection. All one-head rules needed for solving the problem are restricted unfolding, i.e., unfolding restricted by specializing the head atom to be replaced. The two-head rule used for solving the problem was constructed by inductive accumulation of infinite restricted rules.

Acknowledgment. This work was partially supported by (i) JSPS KAKENHI Grant Numbers 25280078 and 26540110, (ii) the Center of Excellence in Intelligent Informatics, Speech and Language Technology and Service Innovation (CILS), Thammasat University, and (iii) the Intelligent Informatics and Service Innovation (IISI) Center, Sirindhorn International Institute of Technology (SIIT), Thammasat University. The authors also gratefully acknowledge the helpful comments and suggestions from the reviewers.

REFERENCES

[1] C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
[2] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.
[3] K. Doets, *From Logic to Logic Programming*, MIT Press, 1994.
[4] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.
[5] C. Walther, A mechanical solution of Schubert's steamroller by many-sorted resolution, *Artificial Intelligence*, vol.26, no.2, pp.217-224, 1985.
[6] F. J. Pelletier, Seventy-five problems for testing automatic theorem provers, *Journal of Automated Reasoning*, vol.2, no.2, pp.191-216, 1986.
[7] M. Stickel, Schubert's steamroller problem: Formulations and solution, *Journal of Automated Reasoning*, vol.2, no.2, pp.89-104, 1986.
[8] T. C. Wang and W. W. Bledsoe, Hierarchical deduction, *Journal of Automated Deduction*, vol.3, no.1, pp.35-77, 1987.
[9] R. Manthey and F. Bry, SATCHMO: A theorem prover implemented in prolog, *Proc. of the 9th International Conference on Automated Deduction*, Argonne, Illinois, pp.415-434, 1988.
[10] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[11] R. Kowalski, Predicate logic as a programming language, *Proc. of the 6th IFIP Congress 1974*, Stockholm, Sweden, pp.569-574, 1974.
[12] R. A. Kowalski, Algorithm = logic + control, *Communications of the ACM*, vol.22, pp.424-435, 1979.

[13] J. Minker, *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers Inc., 1988.
[14] A. Garcez, T. R. Besold, L. D. Raedt, P. Földiák, P. Hitzler, T. Icard, K.-U. Kühnberger, L. Lamb, R. Miikkulainen and D. Silver, Neural-symbolic learning and reasoning: Contributions and challenges, *Proc. of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches*, Stanford, 2015.
[15] R. B. Palm, U. Paquet and O. Winther, Recurrent relational networks, *arXiv Preprint*, arXiv: 1711.08028, 2017.
[16] F. Yang, Z. Yang and W. W. Cohen, Differentiable learning of logical rules for knowledge base reasoning, *Advances in Neural Information Processing Systems*, pp.2319-2328, 2017.
[17] N. Cingillioglu and A. Russo, DeepLogic: End-to-end logical reasoning, *arXiv Preprint*, arXiv: 1805.07433, 2018.
[18] W.-Z. Dai, Q.-L. Xu, Y. Yu and Z.-H. Zhou, Tunneling neural perception and logic reasoning through abductive learning, *arXiv Preprint*, arXiv: 1802.01173, 2018.
[19] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester and L. De Raedt, DeepProbLog: Neural probabilistic logic programming, *Advances in Neural Information Processing Systems*, pp.3749-3759, 2018.
[20] G. Sourek, V. Aschenbrenner, F. Zelezny, S. Schockaert and O. Kuzelka, Lifted relational neural networks: Efficient learning of latent relational structures, *Journal of Artificial Intelligence Research*, vol.62, pp.69-100, 2018.
[21] J. Xu, Z. Zhang, T. Friedman, Y. Liang and G. V. den Broeck, A semantic loss function for deep learning with symbolic knowledge, *Proc. of the 35th International Conference on Machine Learning*, Stockholm, Sweden, pp.5502-5511, http://proceedings.mlr.press/v80/xu18h.html, 2018.
[22] Z. Hu, X. Ma, Z. Liu, E. Hovy and E. Xing, Harnessing deep neural networks with logic rules, *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, pp.2410-2420, 2016.
[23] D. Selsam, M. Lamm, B. Bunz, P. Liang, L. de Moura and D. L. Dill, Learning a sat solver from single-bit supervision, *arXiv Preprint*, arXiv: 1802.03685, 2018.
[24] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, *Proc. of International Logic Programming Conference and Symposium*, pp.1070-1080, 1988.
[25] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, *New Generation Computing*, vol.9, pp.365-386, 1991.
[26] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, *Lecture Notes in Computer Science*, vol.4378, pp.41-54, 2007.
[27] K. Akama and E. Nantajeewarawat, Solving query-answering problems with constraints for function variables, *Proc. of the 10th Asian Conference on Intelligent Information and Database Systems*, Dong Hoi City, Vietnam, pp.36-47, 2018.
[28] K. Akama and E. Nantajeewarawat, Formalization of logical problems as model-intersection problems on an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.17, no.4, pp.1103-1117, 2021.
[29] K. Akama and E. Nantajeewarawat, Model-intersection problems with existentially quantified function variables: Formalization and a solution schema, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, vol.2, Porto, Portugal, pp.52-63, 2016.
[30] K. Akama and E. Nantajeewarawat, Unfolding existentially quantified sets of extended clauses, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, vol.2, Porto, Portugal, pp.96-103, 2016.
[31] K. Akama, E. Nantajeewarawat and T. Akama, Computation control by prioritized ET rules, *Proc. of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018)*, vol.2, Seville, Spain, pp.84-95, 2018.
[32] K. Akama, E. Nantajeewarawat and T. Akama, Logical problem solving framework, *Proc. of the 11th Asian Conference on Intelligent Information and Database Systems*, Yogyakarta, Indonesia, pp.28-40, 2019.
[33] K. Akama, E. Nantajeewarawat and H. Koike, A class of rewriting rules and reverse transformation for rule-based equivalent transformation, *Electronic Notes in Theoretical Computer Science*, vol.59, no.4, pp.1-16, 2001.

[34] K. Akama, H. Koike and H. Mabuchi, A theoretical foundation of program synthesis by equivalent transformation, *Perspectives of System Informatics, Lecture Notes in Computer Science*, vol.2244, pp.131-139, 2001.
[35] K. Akama, E. Nantajeewarawat and H. Koike, Program synthesis based on the equivalent transformation computation model, *Proc. of the 12th International Workshop on Logic Based Program Development and Transformation*, Madrid, Spain, pp.285-304, 2002.
[36] K. Miura, K. Akama and H. Mabuchi, Creation of ET rules from logical formulas representing equivalent relations, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.263-277, 2009.
[37] K. Miura, K. Akama, H. Mabuchi and H. Koike, Theoretical basis for making equivalent transformation rules from logical equivalences for program synthesis, *International Journal of Innovative Computing, Information and Control*, vol.9, no.6, pp.2635-2650, 2013.
[38] K. Miura and K. Akama, ET-based bidirectional search for proving formulas in the class ES, *International Journal of Innovative Computing, Information and Control*, vol.10, no.6, pp.1999-2009, 2014.
[39] M. Newborn, *Automated Theorem Proving: Theory and Practice*, Springer-Verlag, 2000.
[40] J. Durkin, *Expert Systems: Design and Development*, Prentice Hall, New York, 1994.
[41] S. Lindsay, *Practical Applications of Expert Systems*, John Wiley & Sons Inc., Chichester, 1988.
[42] J. C. Giarratano, *Expert Systems: Principles and Programming*, Brooks Cole, Pacific Grove, 1998.
[43] A. L. Kidd, *Knowledge Acquisition for Expert Systems*, A Practical Handbook, Plenum Publishing Corporation, New York, 1987.
[44] A. Alamoudi, A. Alomari, S. Alwarthan and Atta-ur-Rahman, A rule-based information extraction approach for extracting metadata from PDF books, *ICIC Express Letters, Part B: Applications*, vol.12, no.2, pp.121-132, 2021.

Author Biography

Kiyoshi Akama received the B.Eng. and M.Eng. degrees in control engineering from Tokyo Institute Technology, Japan, in 1973 and 1975, respectively; and the D.Eng. degree in control engineering from Tokyo Institute Technology, Japan, in 1989. He was an assistant professor at Faculty of Engineering, Tokyo Institute Technology, Japan, 1979-1981; a lecturer at Faculty of Letters, Hokkaido University, Japan, 1981-1989; an associate professor at Faculty of Engineering, Hokkaido University, Japan, 1989-1999; a professor at Center for Multimedia Studies, Hokkaido University, Japan, 1999-2003; a professor at Information Initiative Center, Hokkaido University, Japan, 2003-2013; a specially-appointed professor at Information Initiative Center, Hokkaido University, 2013-2015; a professor at Graduate School of Information Science and Technology, Hokkaido University, Japan, 1999-2015.

Dr. Akama is currently an emeritus professor of Hokkaido University, Japan. His research interests include artificial intelligence, computer science, logic and computation, program generation and computation based on the equivalent transformation model, programming paradigms, and knowledge representation.

Ekawit Nantajeewarawat received the B.Eng. degree in computer engineering from Chulalongkorn University, Thailand, in 1987; and the M.Eng. and D.Eng. degrees in computer science from the Asian Institute of Technology, Thailand, in 1991 and 1997, respectively.

Dr. Nantajeewarawat is currently an associate professor of computer science at Sirindhorn International Institute of Technology, Thammasat University, Thailand. His research interests include knowledge representation, automated reasoning, rule-based equivalent transformation, program synthesis, formal ontologies, and object-oriented modelling.