International Journal of Innovative Computing, Information and Control
Volume **19**, Number **6**, December **2023**

ICIC International ©2023 ISSN 1349-4198
pp. **1891–1906**

GENERATION OF EQUIVALENT TRANSFORMATION RULES FROM LOGICAL EQUIVALENCES

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 January 2023; revised May 2023

Abstract. *In the "Equivalent Transformation model" (ET model), computation is regarded as Equivalent Transformation (ET) of declarative descriptions, and a program consists of correct rewriting rules (i.e., ET rules) and a control description. An ET rule can be generated by meta computation, i.e., repeated transformation of meta clauses by meta rules. If we use correct meta rules, the obtained rewriting rules are correct. Input of the generation of meta rules in this paper is a logical equivalence, which is an extension of an if-and-only-if formula. We propose three theorems that explain how to generate meta rules based on logical equivalences. Various correct meta rules including both single-head and multi-head rules, and both splitting and non-splitting rules can be generated by these methods.*

Keywords: Equivalent transformation rule, Meta rule, If-and-only-if formula, Logical equivalence, Rule generation, Meta computation

1. **Introduction.** Proof problems on first-order formulas historically constitute the most important problem class in logical problem solving [1, 2, 3]. Resolution provides us with a refutation proof procedure for logical formulas [4]. By adjusting rule application control, many logical solution methods have been invented [5, 6, 7, 8]. However, the conventional resolution-based theories have failed to estabilish a correct proof method for full first-order formulas [9]. A Query-Answering (QA) problem is an "all-answers finding" problem to satisfy a given logical consequence relation [10, 11, 12, 13]. The definition of completeness of SLD resolution is too weak for ensuring the correctness of solutions for all QA problems on definite clauses. The conventional resolution-based theories have failed to estabilish a correct QA solution method for full first-order formulas [14]. To overcome these failures in computational logic, a Logical Problem Solving Framework (LPSF) was developed [15]. Among others, LPSF introduced Equivalent Transformation (ET) rules for procedures. This is fundamentally different from considering logical formulas as programs in logic programming. For proof and QA problems, an extended logic, together with many newly invented ET rules, was shown to improve solvability, compared to conventional solvers on definite clauses with basic ET rules such as resolution and unfolding [16, 17, 18, 19].

DOI: 10.24507/ijicic.19.06.1891

At the core of the LPSF theory, we have the concept of logical structure, which determines the relation between formulas and models. A formula F is associated with the set of all models of F, denoted by $Models(F)$. A model is a set of ground user-defined atoms, and the intersection of all models of a formula F can be considered and is denoted by $\bigcap Models(F)$. Formalization of proof/QA problems as Model-Intersection (MI) problems is of fundamental importance for establishing a general solution method for solving all deductive problems on first-order formulas [20]. To solve an MI problem, ET rules are used. The ET-based theory has been successfully applied to proof problems and QA problems [18, 21, 22, 23, 24]. In the ET model, computation is regarded as equivalent transformation of declarative descriptions, and a program consists of ET rules and a control description.

Rule generation is fundamental for solving problems since it produces programs from a given problem. SLD resolution for QA problems can be regarded as a solver that uses only general unfolding rules. By generating various less general and more efficient rules other than the general unfolding rules, we can solve more problems in shorter time. Rule generation increases solvability even if a small problem is considered [25]. Greater increase of solvability is expected for larger and practical problems. A typical program synthesis method, called the squeeze method [26], has been developed.

We extend the previous rule generation theory by changing the underlying logic and generalize concepts around meta descriptions and meta rules. A meta rule is a rule for transforming one meta description into another meta description. A meta computation consists of repeated application of meta rules to a meta description, by which we obtain a sequence of meta descriptions. The first and the last meta descriptions of the obtained sequence give a pair of meta descriptions, which determines a partial mapping to transform one description into another description. A rewriting rule is obtained as a partial mapping for description transformation.

A theory of meta computation was proposed on ET-based computation of QA problems [27, 28, 29, 30, 31, 32, 33], which were formalized by sets of definite clauses. The minimal model semantics and the closed-world assumption were used. This is in sharp contrast to first-order formulas and usual clauses, which usually take all-model semantics without the closed-world assumption.

Instead of the definite-clause formalization, we take full-clause formalization in order to establish a logical computation theory that can deal with all first-order formulas and all clauses with all-model semantics [34]. We take problem formalization using if-and-only-if formulas (*iff*-formulas). Our starting point for meta-rule generation is a set of logical equivalences that are correct with respect to given problem formalization. We introduce three theorems that explain how to generate meta rules based on logical equivalences that are implied by the given formalization. We prove the correctness of the three theorems.

Section 2 introduces the concepts of if-and-only-if formula (*iff*-formula) and logical equivalence, each of which determines a class of first-order formulas. Section 3 explains a task of rule generation by formalizing a proof problem and by showing a set of ET rules for solving the problem. Section 4 describes the syntax of meta rules, and introduces meta rules to be used for rule generation. Section 5 gives three theorems that explain how to generate meta rules based on *iff*-formulas and logical equivalences, and proves their correctness. Section 6 gives examples of rule generation by meta computation using generated meta rules. Section 7 concludes the paper.

2. If-and-Only-If Formulas and Logical Equivalences.

We introduce if-and-only-if formulas and logical equivalences, which determine two classes of first-order formulas.

2.1. **If-and-only-if formulas (*iff*-formulas).** An atom is a user-defined atom or a built-in constraint atom. Given an atom a, let $var(a)$ denote the set of all variables occurring in a. Assume that A is a set of atoms. Let $var(A) = \bigcup\{var(a)|a \in A\}$. The conjunction of all elements in A is denoted by the set A itself for simplicity. If a conjunction $conj$ is denoted by a set A, let $var(conj)$ denote the set of all variables occurring in A, i.e., $var(conj) = var(A)$. The conjunction of all elements in A is simply called a conjunction of atoms, if it is not necessary to refer to A. Let $\mathcal{A}_{\rm u}$ be the set of all user-defined atoms and $\mathcal{G}_{\rm u}$ the set of all ground user-defined atoms. Let $\mathcal{A}_{\rm c}$ be the set of all built-in constraint atoms and $\mathcal{G}_{\rm c}$ the set of all ground built-in constraint atoms. An *if-and-only-if formula* (for short, *iff-formula*) I on $\mathcal{A}_{\rm u} \cup \mathcal{A}_{\rm c}$ is a formula of the form

$$a \leftrightarrow (conj_1 \vee conj_2 \vee \cdots \vee conj_n),$$

where $n \geq 1$, a is an atom in $\mathcal{A}_{\rm u}$, and each of the $conj_i$ is a conjunction of atoms in a finite subset of $\mathcal{A}_{\rm u} \cup \mathcal{A}_{\rm c}$. The atom a is called the *head* of the *iff*-formula I. It is a *ground iff*-formula iff a and $conj_1, conj_2, \ldots, conj_n$ are all ground. When emphasis is given to its head, an *iff*-formula whose head is an atom a is often referred to as *iff*(a).

For any conjunction $conj'$ of atoms and any atom a', let $\textsc{Fol}(conj', a')$ be an existentially quantified atom conjunction defined by

$$\textsc{Fol}(conj', a') = \exists y_1 \exists y_2 \cdots \exists y_k : \bigwedge \{b|b \in conj'\},$$

where $\{y_1, y_2, \ldots, y_k\} = var(conj') - var(a')$. Then, for each $i \in \{1, 2, \ldots, n\}$, $conj_i$ corresponds to $\textsc{Fol}(conj_i, a)$, which may contain variables in $var(a)$ as free variables. The *iff*-formula I corresponds to the universally quantified formula

$$\forall(a \leftrightarrow (\textsc{Fol}(conj_1, a) \vee \textsc{Fol}(conj_2, a) \vee \cdots \vee \textsc{Fol}(conj_n, a))),$$

which is denoted by $\textsc{Fol}(I)$. Variables in $var(a)$ are quantified universally at the top of the formula. The first-order formula $\textsc{Fol}(I)$ is also called an *iff*-formula if no confusion occurs.

2.2. **Logical equivalences.** Let $\mathcal{F}$ and $\mathcal{F}_i$ $(i = 1, 2, \ldots)$ be conjunctions of atoms. Let V and V_i $(i = 1, 2, \ldots)$ be sets of usual variables. A *logical equivalence LE* is a formula of the form

$$\forall(\exists_V \mathcal{F} \leftrightarrow (\exists_{V_1} \mathcal{F}_1 \vee \exists_{V_2} \mathcal{F}_2 \vee \cdots \vee \exists_{V_m} \mathcal{F}_m)),$$

where $m \geq 1$. For example, $\exists_{\{\}} pal(x) \leftrightarrow \exists_{\{\}} rev(x, x)$ and $\exists_{\{\}} init(x, y) \leftrightarrow \exists_{\{m\}} app(x, m, y)$ are logical equivalences. Since $\mathcal{F}$ as well as $\mathcal{F}_i$ can be a conjunction of atoms,

$$\forall a, \forall b, \forall c, \forall d : \\ (\exists m : (app(a, b, m) \wedge app(m, c, d)) \leftrightarrow \exists n : (app(b, c, n) \wedge app(a, n, d)))$$

is a logical equivalence. Moreover, since the part $\exists_{V_1} \mathcal{F}_1 \vee \exists_{V_2} \mathcal{F}_2 \vee \cdots \vee \exists_{V_m} \mathcal{F}_m$ can be a disjunction of more than one conjunction,

$$\forall x, \forall y : rev(x, y) \quad \leftrightarrow \\ ((eq(x, [\,]) \wedge eq(y, [\,])) \quad \vee \quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge rev(w, u) \wedge app(u, [a], y)))$$

is a logical equivalence.

For any conjunction $\mathcal{F}'$ of atoms and any set V' of variables, let $\textsc{Fol}(\mathcal{F}', V')$ be an existentially quantified atom conjunction defined by

$$\textsc{Fol}(\mathcal{F}', V') = \exists y_1 \exists y_2 \cdots \exists y_k : \bigwedge \{b|b \in \mathcal{F}'\},$$

where $\{y_1, y_2, \ldots, y_k\} = V'$. Then, for each $i \in \{1, 2, \ldots, m\}$, $\exists_{V_i} \mathcal{F}_i$ corresponds to $\textsc{Fol}(\mathcal{F}_i, V_i)$, which may contain variables in $var(\mathcal{F}_i) - V_i$ as free variables. The logical

equivalence LE corresponds to the universally quantified formula

$$\forall(\textsc{Fol}(\mathcal{F}, V) \leftrightarrow (\textsc{Fol}(\mathcal{F}_1, V_1) \vee \textsc{Fol}(\mathcal{F}_2, V_2) \vee \cdots \vee \textsc{Fol}(\mathcal{F}_m, V_m))),$$

which is denoted by $\textsc{Fol}(LE)$. Variables in $(var(\mathcal{F}) - V) \cup (var(\mathcal{F}_1) - V_1) \cup (var(\mathcal{F}_2) - V_2) \cup \cdots \cup (var(\mathcal{F}_m) - V_m)$ are quantified universally at the top of the formula. The first-order formula $\textsc{Fol}(LE)$ is also called a logical equivalence if no confusion occurs.

2.3. **Correct logical equivalences.** If a first-order formula F can be represented by an *iff*-formula, then F can also be represented by a logical equivalence, i.e.,

$$\{\textsc{Fol}(I) | I \textit{ is an iff-formula}\} \subseteq \{\textsc{Fol}(LE) | LE \textit{ is a logical equivalence}\}.$$

In this sense, an *iff*-formula is regarded as a logical equivalence.

A logical equivalence is defined to be correct with respect to a first-order formula F iff $F \models LE$. For instance, the logical equivalence

$$\forall a, \forall b, \forall c, \forall d : \\ (\exists m : (app(a, b, m) \wedge app(m, c, d)) \leftrightarrow \exists n : (app(b, c, n) \wedge app(a, n, d)))$$

is a logical consequence of, and is thus correct with respect to, the first-order formula (an *iff*-formula)

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

Given a first-order formula F that serves as background knowledge of a first-order formalization, we generate meta rules from logical equivalences that are correct with respect to F. A logical equivalence is defined to be correct with respect to a formalization if there is background knowledge F of the formalization such that the logical equivalence is correct with respect to F.

2.4. **More general problem formalization.** In our ET-based computation theory, MI problems are solved by employment of ET rules that are generated by meta computation using meta rules. The previous theories for rule generation were constructed based on solutions for QA problems on definite clauses, and the minimal model semantics of definite clauses was taken. Background knowledge in a definite-clause formalization is a set of definite clauses. The core structure of our previous theory of rule generation is as follows:

$$\textit{definite clauses} \rightarrow \textit{logical equivalences} \rightarrow \textit{meta rules} \rightarrow \textit{ET rules}$$

If we use a set D of definite clauses as background knowledge, we cannot obtain enough logical equivalences LE by implication of D, i.e., $D \models LE$. For instance, the logical equivalence illustrated in Section 2.3 is not a logical consequence of the background knowledge D_0 of the definite-clause formalization shown in Section 3.3.

To solve all MI problems on first-order formulas, we take the LPSF theory and formalization by using normal clauses together with *iff*-formulas, in place of definite-clause formalization [34]. We develop a new method of rule generation with the following core structure:

$$\textit{iff-formulas} \rightarrow \textit{logical equivalences} \rightarrow \textit{meta rules} \rightarrow \textit{ET rules}$$

MI problems are solved by ET rules that are generated by meta computation using meta rules. Meta rules are generated from logical equivalences that are logical consequences of the *iff*-formulas contained in the formalization of a given MI problem. A method for transformation of *iff*-formulas into logical equivalences will be developed in Section 5.

3. **Problem Formalization and Rule Generation.** We introduce a proof problem and two types of its formalization and show a set of ET rules for solving the problem, which forms a task of rule generation.

3.1. **The pal-pal proof problem and unsatisfiability-based formalization.** Assume as background knowledge a set consisting of the three *iff*-formulas below, where *pal*, *rev*, *eq*, and *app* stand for "palindrome", "reverse", "equal", and "append", respectively.

(1) $\forall x : pal(x) \leftrightarrow rev(x, x)$.

(2) $\forall x, \forall y : rev(x, y) \quad \leftrightarrow$
$((eq(x, [\,]) \wedge eq(y, [\,])) \quad \vee \quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge rev(w, u) \wedge app(u, [a], y)))$.

(3) $\forall x, \forall y, \forall z : app(x, y, z) \quad \leftrightarrow$
$((eq(x, [\,]) \wedge eq(y, z)) \quad \vee \quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge eq(z, [a|u]) \wedge app(w, y, u)))$.

Consider the problem "prove that there are no ground terms s and t such that two lists $[1, s|t]$ and $[2, s|t]$ are palindromes", which is called the pal-pal proof problem [9].

Let F_0 be the conjunction of the above three *iff*-formulas, and E_0 the following formula:

$$\exists A, \exists X : (pal([1, A|X]) \wedge pal([2, A|X])).$$

The pal-pal proof problem to prove $F_0 \models \neg E_0$ can be formalized as $Models(\neg(F_0 \rightarrow \neg E_0)) = \emptyset$, which is equivalent to $Models(F_0 \wedge E_0) = \emptyset$. This is called *unsatisfiability-based formalization* [9].

3.2. **Existence-finding formalization.** We refer to F_0 and E_0 in Section 3.1. Since the pal-pal proof problem negates existence of terms by saying that "prove that there are no ground terms s and t such that $[1, s|t]$ and $[2, s|t]$ are palindromes", we introduce a formula $E_1 = (E_0 \rightarrow$ "exists") and consider $\bigcap Models(F_0 \wedge E_1)$. Then the answer of the *pal-pal* problem is formalized as

$$answer_{\textit{pal-pal}} = \varphi_1 \left(\bigcap Models(F_0 \wedge E_1) \right),$$

where φ_1 is a mapping defined as follows.

Definition 3.1. φ_1 is a mapping from $pow(\mathcal{G}_{\mathrm{u}})$ to {"yes", "no"} such that for each $G \subseteq \mathcal{G}_{\mathrm{u}}$,

- $\varphi_1(G) =$ "yes" if $G - \bigcap Models(F_0) = \emptyset$, and
- $\varphi_1(G) =$ "no" if $G - \bigcap Models(F_0) =$ {"exists"}.

We take this formalization, which is called *existence-finding formalization* [9].

3.3. **Definite clauses and *iff*-formulas.** Let D_0 be a set consisting of the following definite clauses:

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

D_0 can be used for formalizing the pal-pal proof problem. This formalization is usually used in the logic programming approach with SLD resolution being the basic computation method. However, our formalization is based on *iff*-formulas, as introduced in Section 3.1. In this paper, ET rules are generated based on *iff*-formulas.

3.4. **ET rule generation.** A solution by using a set of ET rules can be found in [9]. The following ET rules can also be used to solve the pal-pal proof problem.

r_{pal}: $pal(X) \Rightarrow rev(X, X)$.

r_{rev_1}: $rev([A|X], Y) \Rightarrow rev(X, V), app(V, [A], Y)$.

r_{rev_2}: $rev(X, Y), rev(X, Z) \Rightarrow \{=(Y, Z)\}, rev(X, Y)$.

r_{app_1}: $app(X, Y, [A|Z]) \Rightarrow \{=(X, [\,]), =(Y, [A|Z])\}$;
$\Rightarrow \{=(X, [A|V])\}, app(V, Y, Z)$.

r_{rev_3}: $rev(X, [A|Y]) \Rightarrow \{=(X, [U|V])\}, rev(V, W), app(W, [U], [A|Y])$.

r_{app_2}: $app(X, [A], [B, C|Y]) \Rightarrow \{=(X, [B|V])\}, app(V, [A], [C|Y])$.

r_{app_3}: $app(X, [A], [B]) \Rightarrow \{=(X, [\,]), =(A, B)\}$.

r_{rev_4}: $rev([\,], X) \Rightarrow \{=(X, [\,])\}$.

Among the ET rules used for solving the pal-pal proof problem, the multi-head rule r_{rev_2} shown above is indispensable. Compared to other ET rules that can be generated based on unfolding, it is more difficult to generate this multi-head rule. It was shown in [9] that the pal-pal proof problem cannot be solved by SLD resolution, the reason for which is that SLD resolution cannot deal with two atoms in the body of a clause at the same time and therefore cannot realize transformation that is done by the rule r_{rev_2}. Such a multi-head rule can be obtained from a meta rule that is generated from a logical equivalence. Theorem 5.3 plays a central role in generation of multi-head rules.

4. Meta Rules.

We explain the syntax of meta rules, and introduce important meta rules to be used for rule generation in subsequent sections.

4.1. Syntax of meta rules.

Meta rules consist of meta-meta atoms that may contain $*$-variables and %-variables. A meta rule is an expression of the form

$\alpha, \{cond\}$
$\Rightarrow \{exec_1\}, \beta_1;$
$\Rightarrow \{exec_2\}, \beta_2;$
$\ldots$
$\Rightarrow \{exec_n\}, \beta_n,$

where $n \geq 1$, each of *cond* and $exec_i$ is an optional sequence of built-in meta-meta atoms, and each of $\alpha, \beta_1, \beta_2, \ldots, \beta_n$ is a sequence of meta-meta atoms. Basically, when this meta rule is applied to a target meta clause, the target meta clause is replaced with n meta clauses, which are modified by the execution of *cond* and $exec_i$. For a detailed explanation, the reader is referred to [33].

4.2. Simple meta rules.

We will explain meta rules by using the following simple examples:

(a) $final(*A, *B) \Rightarrow app(\%X, *A, *B)$.
(b) $rev(*X, *Y)$
$\Rightarrow eq(*X, [\,]), eq(*Y, [\,]);$
$\Rightarrow eq(*X, [\%A|\%V]), rev(\%V, \%M), app(\%M, [\%A], *Y)$.
(c) $app(\%X, *A, *B) \Rightarrow final(*A, *B)$.
(d) $eq(*Z, \%V) \Rightarrow \{bind(\%V, *Z)\}$.

Given a target meta clause, each of the above meta rules works as follows.

- By the meta rule (a), a target meta atom that $final(*A, *B)$ matches is rewritten into a new meta atom of the form $app(\%X, *A, *B)$. The values of $*A$ and $*B$ are determined uniquely by the matching. A new #-variable, which does not appear in the target meta clause, is introduced and substituted for $\%X$.

- By the meta rule (b), a target meta atom that $rev(*X, *Y)$ matches is rewritten, and the target meta clause is replaced with two meta clauses. To construct the first meta clause, the target meta atom is changed into a sequence of $eq(*X, [\,])$ and $eq(*Y, [\,])$. Since the values of $*X$ and $*Y$ are determined uniquely by the matching, the first meta clause is determined uniquely. To construct the second meta clause, the target meta atom is changed into three meta atoms of the form $eq(*X, [\%A|\%V])$, $rev(\%V, \%M)$, and $app(\%M, [\%A], *Y)$. Three new #-variables, which do not appear in the target meta clause, are introduced and substituted for $\%A$, $\%V$, and $\%M$.
- By the meta rule (c), a meta atom that $app(\%X, *A, *B)$ matches is rewritten into a meta atom of the form $final(*A, *B)$. If the #-variable that is matched by $\%X$ has an occurrence in some other part of the target meta clause, the matching fails and the rule cannot be applied.
- By the meta rule (d), a meta atom that $eq(*Z, \%V)$ matches is removed and the resulting meta clause is specialized by $\{\%V/*Z\}$. For example, the target meta clause $h \leftarrow eq(5, \#X), p(\#X)$ is transformed by this meta rule into $h \leftarrow p(5)$.

4.3. **Correctness of meta rules and meta computation.** The correctness of a rewriting rule and that of a meta rule are next defined. A computation consists of repeated application of rewriting rules to a description, by which we obtain a sequence of descriptions.

Definition 4.1. *A rewriting rule r is correct with respect to a first-order formula F iff for any clause C and any clause set Cs', if r transforms $\{C\}$ into Cs', then for any clause set Cs such that $Cs \models F$, $\bigcap Models(Cs \cup \{C\}) = \bigcap Models(Cs \cup Cs')$.*

A meta rule is a rule for transforming one meta description into another meta description. A meta computation consists of repeated application of meta rules to a meta description, by which we obtain a sequence of meta descriptions [33]. A meta description is a set of meta clauses. Meta descriptions are instantiated into descriptions. The operation for obtaining a declarative description from a meta description is called *instantiation*. A meta rule is correct if all rewriting steps that are obtained by instantiation of the transformation determined by the meta rule preserve model intersection. More precisely, the correctness of a meta rule is defined based on the instantiation structure as follows.

Definition 4.2. *A meta rule mr is correct with respect to a first-order formula F iff the following condition is satisfied: If a meta description d_m is transformed by mr into a meta description d'_m, θ is a specialization for meta descriptions, Cs is a set of clauses such that $Cs \models F$, d_m is instantiated by θ into a clause set $Cs \cup \{C\}$, and d'_m is instantiated by θ into a clause set $Cs \cup Cs'$, where C is a clause and Cs' is a set of clauses, then $\bigcap Models(Cs \cup \{C\}) = \bigcap Models(Cs \cup Cs')$.*

A correct meta rule makes an equivalent transformation step as follows.

Theorem 4.1. *Assume that a meta rule mr is correct with respect to a first-order formula F. Assume that a meta description d_m is transformed by mr into a meta description d'_m. If d_m is instantiated by a specialization θ into a clause set $Cs \cup \{C\}$, d'_m is instantiated by θ into a clause set $Cs \cup Cs'$, and $Cs \models F$, where Cs is a set of clauses, C is a clause, and Cs' is a set of clauses, then $\bigcap Models(Cs \cup \{C\}) = \bigcap Models(Cs \cup Cs')$.*

Proof: The result directly follows from Definition 4.2. □

4.4. **Correctness of meta rules for equality.** We also use meta rules for equality atoms (eq atoms).

(Ea) $eq([*A|*X],[*B|*Y]) \Rightarrow eq(*A,*B), eq(*X,*Y).$

(Eb) $eq(*X,*X) \Rightarrow .$

(Ec) $eq([\,],[*X|*Y]) \Rightarrow \{false\}.$

$eq([*X|*Y],[\,]) \Rightarrow \{false\}.$

(Ed) $eq(\%V,*Z) \Rightarrow \{bind(\%V,*Z)\}.$

$eq(*Z,\%V) \Rightarrow \{bind(\%V,*Z)\}.$

The correctness of these meta rules can be justified as follows: By the meta rule (Ea), a target equality meta atom is replaced with two equality meta atoms. The corresponding transformation in the level of usual clauses is decomposition of lists, which is obviously correct transformation. The correctness of the meta rules (Eb) and (Ec) can be similarly justified.

By the meta rule (Ed), equality is solved. Since the meta-meta variable $\%V$ in (Ed) matches a #-variable that does not appear in any other part of a target meta clause, the equality meta atom in (Ed) is instantiated to an equality atom with variables that do not appear in any other part of a target clause. Hence, the variables can be bound without affecting any other part of the target clause, and the variable binding obviously yields correct transformation.

5. **Making Meta Rules from Logical Equivalences.** Systematic and correct generation of procedural knowledge from declarative knowledge is a central concern in the logical computation theory. Logical equivalences are formulas representing declarative knowledge, while rewriting rules constitute a procedure representing procedural knowledge. Meta rules are procedural rules used for producing rewriting rules. We give three theorems that explain how to generate meta rules from logical equivalences, and prove their correctness.

5.1. **A mapping to obtain meta rules.** Now we consider a relation between a logical equivalence and a meta rule. The correctness of the meta rule

$$rev(*X,[*B|*Y]) \Rightarrow eq(*X,[\%A|\%W]), rev(\%W,\%U), app(\%U,[\%A],[*B|*Y])$$

will be shown from the *iff*-formula

$$\forall x,\forall y,\forall b : rev(x,[b|y]) \quad \leftrightarrow$$
$$\exists a,\exists w,\exists u : (eq(x,[a|w]) \wedge rev(w,u) \wedge app(u,[a],[b|y])),$$

which is obtained by specializing the *iff*-formula (2) in Section 3.1, and is represented by the following logical equivalence:

$$\forall \big(\exists_{\{\}}(rev(x,[b|y]) \leftrightarrow \exists_{\{a,w,u\}}(eq(x,[a|w]) \wedge rev(w,u) \wedge app(u,[a],[b|y])))\big).$$

In the above example, the atom $rev(x,[b|y])$ in the logical equivalence corresponds to the meta atom $rev(*X,[*B|*Y])$ in the head of the meta rule. Similarly, the atom conjunction $eq(x,[a|w]) \wedge rev(w,u) \wedge app(u,[a],[b|y])$ in the logical equivalence corresponds to the meta-atom conjunction $eq(*X,[\%A|\%W]), rev(\%W,\%U), app(\%U,[\%A],[*B|*Y])$ in the body of the meta rule. The variables a, w and u are mapped into the %-variables $\%A$, $\%W$ and $\%U$, respectively. Other variables occurring in the logical equivalence are mapped into $*$-variables.

Consider a general form of a logical equivalence

$$\forall(\exists_V conj \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n)).$$

Let $V_{seq} = [V, V_1, V_2, \ldots, V_n]$. Comparing the meta rule and the logical equivalence above, we can observe that

- there is a mapping, say mm, such that $mm(V_{seq}, x) = *X$, $mm(V_{seq}, b) = *B$, $mm(V_{seq}, y) = *Y$, $mm(V_{seq}, a) = \%A$, $mm(V_{seq}, w) = \%W$, and $mm(V_{seq}, u) = \%U$, where we notice that x, b and y are universally quantified variables and are mapped into $*$-variables, and a, w and u are existentially quantified variables and are mapped into %-variables.

We extend the mapping mm into a mapping for associating a meta atom conjunction with each conjunction of atoms. For instance, $mm(V_{seq}, (eq(x, [\,]) \wedge eq(y, [a|z]))) = (eq(*X, [\,]) \wedge eq(*Y, [*A| * Z]))$. We also extend the mapping mm into a mapping for associating a logical equivalence with a meta rule.

Formally, we define mm as follows.

- Let LE be a logical equivalence $\forall(\exists_V conj \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n))$, and $V_{seq} = [V, V_1, V_2, \ldots, V_n]$.
- $mm(V_{seq}, LE)$ is defined as a meta rule of the form $(\alpha \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$ that satisfies the following conditions.
  - α is the same conjunction as $conj$ except for variables occurring in them.
  - β_i $(i = 1, 2, \ldots, n)$ is the same conjunction as $conj_i$ except for variables occurring in them.
  - Existentially quantified variables, i.e., variables in the disjoint union $V \uplus V_1 \uplus V_2 \uplus \cdots \uplus V_n$, are mapped into mutually different %-variables.
  - Universally quantified variables, i.e., variables in LE except for existentially quantified variables, are mapped into mutually different $*$-variables.

5.2. **Making reverse meta-rules from logical equivalences.** From the logical equivalence $\forall(rev([a|x], y) \leftrightarrow \exists_{\{u\}}(rev(x, u) \wedge app(u, [a], y)))$, we have a meta rule

$$rev([*A| * X], *Y) \Rightarrow rev(*X, \%U), app(\%U, [*A], *Y).$$

Moreover, we have a reverse meta rule

$$rev(*X, \%U), app(\%U, [*A], *Y) \Rightarrow rev([*A| * X], *Y).$$

Let LE be a logical equivalence $\forall(a(x) \leftrightarrow \exists y : conj(x, y))$. Consider a meta rule $mr = mm(V_{seq}, LE) = (a(*X) \Rightarrow conj(*X, \%Y))$, where $V_{seq} = [\{\,\}, \{y\}]$. Let mr' be the reverse of mr, i.e., $mr' = (conj(*X, \%Y) \Rightarrow a(*X))$. By replacement of $a(x)$ with $\exists y : conj(x, y)$, (a) $\forall x$: $(h(x) \leftarrow a(x))$ is transformed into (b) $\forall x$: $(h(x) \leftarrow \exists y : conj(x, y))$ equivalently. Since (b) is equivalent to (c) $\forall x, \forall y$: $(h(x) \leftarrow conj(x, y))$, the transformation from (a) to (c) preserves models in the presence of LE. Let θ_a be an instantiation for mr such that $mr\theta_a = (a(\&X) \Rightarrow conj(\&X, \#Y))$. Then $mr\theta_a$ rewrites (a) into (c). Similarly, $mr'\theta_a$ rewrites (c) into (a). The equivalence of (a) and (c) has been already obtained above. More generally, we have the following theorem.

Theorem 5.1. *Assume that a is a user-defined atom and $conj$ is a conjunction of atoms. Let LE be a logical equivalence $\forall\left(\exists_{\{\}} a \leftrightarrow \exists_V conj\right)$. Let mr be a meta rule $\alpha \Rightarrow \beta$ that is defined by $mr = mm(V_{seq}, LE)$, where $V_{seq} = [\{\,\}, V]$. Then mr and its reverse $\beta \Rightarrow \alpha$ are meta rules that are correct with respect to LE.*

Proof: Let θ_a be an arbitrary instantiation for the meta rule mr. Let $\alpha' = \alpha\theta_a$ and $\beta' = \beta\theta_a$. By θ_a, we have a rewriting rule $\alpha' \Rightarrow \beta'$. We prove that this rewriting rule is correct with respect to LE by showing that any instantiation of it produces model-preserving transformation in the context of LE.

Let θ_b be an arbitrary instantiation for the rewriting rule $\alpha' \Rightarrow \beta'$. Given a clause C, we have a replacement of α'' with β'' in the body of C, where $\alpha'' = \alpha\theta_a\theta_b$ and $\beta'' = \beta\theta_a\theta_b$.

Since θ_b is a renaming instantiation that changes $*$-variables into &-variables, and %-variables into #-variables, there are renaming instantiations ρ on $var(a) \cup (var(conj) - V)$ and ρ' on V such that (i) $\alpha'' = a\rho$ and (ii) $\beta'' = conj(\rho \cup \rho')$.

Consider the transformation of the clause C into a clause C' by replacing α'' in the body of C with β''. This transformation is obtained from (i) the replacement transformation by the logical equivalence LE using ρ together with (ii) removal of $\exists_V$ using ρ'. Since these two types of transformation preserve models in the presence of LE, we have $Models(Cs \cup \{C\}) = Models(Cs \cup \{C'\})$ for any clause set Cs. Therefore, the rewriting rule $\alpha' \Rightarrow \beta'$ is correct with respect to LE, and the meta rule $\alpha \Rightarrow \beta$ is correct with respect to LE.

Let mr' be the meta rule $\beta \Rightarrow \alpha$. Assume that we obtain a rewriting rule $\beta' \Rightarrow \alpha'$ by instantiation of mr'. Since the reverse transformation of $\alpha' \Rightarrow \beta'$ also preserves models in the presence of LE, mr' is also correct with respect to LE. □

5.3. **Making single-head meta rules from a logical equivalence.** From the logical equivalence

$$\forall(rev(x,y) \leftrightarrow \exists_{\{\}}(eq(x,[\,]) \wedge eq(y,[\,])) \vee \exists_{\{a,w,u\}} : (eq(x,[a|w]) \wedge rev(w,u) \wedge app(u,[a],y))),$$

we have a meta rule

$$\begin{aligned} rev(*X,*Y) &\Rightarrow eq(*X,[\,]), eq(*Y,[\,]); \\ &\Rightarrow eq(*X,[\%A|\%W]), rev(\%W,\%U), app(\%U,[\%A],*Y). \end{aligned}$$

Let LE be a logical equivalence $\forall(a(x) \leftrightarrow \exists y_1 : conj_1(x,y_1) \vee \exists y_2 : conj_2(x,y_2) \vee \cdots \vee \exists y_n : conj_n(x,y_n))$. Consider a meta rule $mr = mm(V_{seq}, LE) = (a(*X) \Rightarrow conj_1(*X,\%Y_1); \Rightarrow conj_2(*X,\%Y_2); \cdots; \Rightarrow conj_n(*X,\%Y_n))$, where $V_{seq} = [\{\}, \{y_1\}, \{y_2\}, \ldots, \{y_n\}]$. By replacement of $a(x)$ with $\exists y_1 : conj_1(x,y_1) \vee \exists y_2 : conj_2(x,y_2) \vee \cdots \vee \exists y_n : conj_n(x,y_n)$, (a) $\forall x : (h(x) \leftarrow a(x))$ is transformed into (b) $\forall x : (h(x) \leftarrow \exists y_1 : conj_1(x,y_1) \vee \exists y_2 : conj_2(x,y_2) \vee \cdots \vee \exists y_n : conj_n(x,y_n))$ equivalently. Since (b) is equivalent to (c) $\forall x, \forall y_1, \forall y_2, \ldots, \forall y_n : (h(x) \leftarrow conj_1(x,y_1) \vee conj_2(x,y_2) \vee \cdots \vee conj_n(x,y_n))$, the transformation from (a) to (c) preserves models in the presence of LE. Let θ_a be an instantiation for mr such that $mr\theta_a = (a(\&X) \Rightarrow conj_1(\&X,\#Y_1); \Rightarrow conj_2(\&X,\#Y_2); \cdots; \Rightarrow conj_n(\&X,\#Y_n))$. Then $mr\theta_a$ rewrites (a) into (c). The equivalence of (a) and (c) has been already obtained above. More generally, we have the following theorem.

Theorem 5.2. *Assume that a is a user-defined atom and $conj_1, conj_2, \ldots, conj_n$ are conjunctions of atoms. Let LE be a logical equivalence $\forall\big(\exists_{\{\}} a \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n)\big)$. Let mr be a meta rule $(\alpha \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$ that is defined by $mr = mm(V_{seq}, LE)$, where $V_{seq} = [\{\}, V_1, V_2, \ldots, V_n]$. Then the meta rule mr is correct with respect to LE.*

Proof: Let θ_a be an arbitrary instantiation for the meta rule mr. Let $\alpha' = \alpha\theta_a$ and $\beta'_i = \beta_i\theta_a$ $(i = 1, 2, \ldots, n)$. By θ_a, we have a rewriting rule $r = (\alpha' \Rightarrow \beta'_1; \Rightarrow \beta'_2; \cdots; \Rightarrow \beta'_n)$. We prove that the rewriting rule r is correct with respect to LE by showing that any instantiation of r produces model-preserving transformation.

Let θ_b be an arbitrary instantiation for the rewriting rule r. Given a clause C, the replacement of α'' with β''_i $(i = 1, 2, \ldots, n)$ in the body of C produces C_i $(i = 1, 2, \ldots, n)$, where $\alpha'' = \alpha\theta_a\theta_b$ and $\beta''_i = \beta\theta_a\theta_b$ $(i = 1, 2, \ldots, n)$. Since θ_b is a renaming instantiation that changes $*$-variables into &-variables, and %-variables into #-variables, there are renaming instantiations ρ on $var(a) \cup (var(conj_1) - V_1) \cup (var(conj_2) - V_2) \cup \cdots \cup (var(conj_n) - V_n)$ and ρ'_i on V_i $(i = 1, 2, \ldots, n)$ such that (i) $\alpha'' = a\rho$ and (ii) $\beta''_i = conj_i(\rho \cup \rho'_i)$ $(i = 1, 2, \ldots, n)$.

Consider the transformation of the clause C into clauses $C'_1, C'_2, \ldots, C'_n$ by replacing α'' in the body of C with β''_i $(i = 1, 2, \ldots, n)$. This transformation is obtained from (i)

the replacement transformation by the logical equivalence LE using ρ together with (ii) removal of $\exists_{V_i}$ using ρ'_i $(i = 1, 2, \ldots, n)$. Since these two types of transformation preserve models in the presence of LE, we have $Models(Cs \cup \{C\}) = Models(Cs \cup \{C'_1, C'_2, \ldots, C'_n\})$ for any clause set Cs. Therefore, the rewriting rule r is correct with respect to LE, and the meta rule mr is correct with respect to LE. □

5.4. **Making multi-head meta rules from a logical equivalence.** From the logical equivalence

$$\forall \left(\exists_{\{m\}}(app(a, b, m) \wedge app(m, c, d)) \leftrightarrow \exists_{\{n\}}(app(b, c, n) \wedge app(a, n, d))\right),$$

we have a meta rule

$$app(*A, *B, \%M), app(\%M, *C, *D) \Rightarrow app(*B, *C, \%N), app(*A, \%N, *D).$$

The correctness of such meta-rule generation is proved by the following theorem.

Theorem 5.3. *Assume that $conj, conj_1, conj_2, \ldots, conj_n$ are conjunctions of atoms. Let LE be a logical equivalence $\forall(\exists_V conj \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n))$. Let mr be a meta rule $(\beta \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$ that is defined by $mr = mm(V_{seq}, LE)$, where $V_{seq} = [V, V_1, V_2, \ldots, V_n]$. Then the meta rule mr is correct with respect to LE.*

Proof: Let $Var = (var(conj) - V) \cup (var(conj_1) - V_1) \cup (var(conj_2) - V_2) \cup \cdots \cup (var(conj_n) - V_n)$. Assume that $Var = \{x_1, x_2, \ldots, x_m\}$. Let a be a user-defined atom $p(x_1, x_2, \ldots, x_m)$, where p is a new predicate that does not appear elsewhere. Let $LE_1 = \forall(a \leftrightarrow \exists_V conj)$, and $LE_2 = \forall(a \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n))$. By Theorems 5.1 and 5.2, we have two meta rules $(\beta \Rightarrow \alpha)$ and $(\alpha \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$, which are correct with respect to $LE \wedge LE_1 \wedge LE_2$. By making the composition of these two meta rules, we have the meta rule $mr = (\beta \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$, which is also correct with respect to $LE \wedge LE_1 \wedge LE_2$. Hence, mr is correct with respect to LE. □

5.5. **Making meta rules from a logical equivalence.** Let $var(LE)$ denote the set of all variables that appear in a logical equivalence LE. Theorem 5.2 is more general than Theorem 5.1. Theorem 5.3 is more general than Theorem 5.1 and Theorem 5.2. By Theorem 5.3, meta rules can be generated from logical equivalences as follows.

1) Input a logical equivalence LE of the form
$$\forall(\exists_V conj \leftrightarrow (\exists_{V_1} conj_1 \vee \exists_{V_2} conj_2 \vee \cdots \vee \exists_{V_n} conj_n)).$$
2) Make a sequence of variables $V_{seq} = [V, V_1, V_2, \ldots, V_n]$.
3) Rename the existentially quantified variables in the disjoint union $V \uplus V_1 \uplus V_2 \uplus \cdots \uplus V_n$ to mutually different %-variables. Let $\theta_\exists$ be the renaming substitution used for this purpose.
4) Rename the universally quantified variables in $var(LE)$ - $(V \cup V_1 \cup V_2 \cup \cdots \cup V_n)$ to $*$-variables. Let $\theta_\forall$ be the renaming substitution used for this purpose.
5) Make α by renaming $conj$ using $\theta_\forall$ and $\theta_\exists$.
6) Make β_i by renaming $conj_i$ using $\theta_\forall$ and $\theta_\exists$ for $i = 1, 2, \ldots, n$.
7) Make a meta rule of the form $(\alpha \Rightarrow \beta_1; \Rightarrow \beta_2; \cdots; \Rightarrow \beta_n)$.

6. **Rule Generation by Meta Computation.** We provide examples of rule generation by meta computation.

6.1. **Definite clauses and *iff*-formulas.** We define two predicates *initial* and *app* by three definite clauses:

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

Behind these definite clauses, there are two *iff*-formulas as follows:

$\forall x, \forall z : (initial(x, z) \leftrightarrow \exists y : append(x, y, z)).$
$\forall x, \forall y, \forall z : app(x, y, z) \leftrightarrow$
$\quad ((eq(x, [\,]) \wedge eq(y, z)) \ \vee$
$\quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge eq(z, [a|u]) \wedge app(w, y, u))).$

6.2. **Meta rules and rewriting rules.** The next three rules are simple meta rules.

(a) $initial(*A, *B) \Rightarrow app(*A, \%Y, *B).$
(b) $app(*X, *Y, *Z)$
$\quad \Rightarrow eq(*X, [\,]),\ eq(*Y, *Z);$
$\quad \Rightarrow eq(*X, [\%A|\%V]),\ eq(*Z, [\%A|\%W]),\ app(\%V, *Y, \%W).$
(c) $app(*X, \%Y, *Z) \Rightarrow initial(*X, *Z).$

The meta rules (a), (b), and (c) are obtained from the two *iff*-formulas in Section 6.1 using Theorem 5.1, Theorem 5.2, and Theorem 5.1, respectively. We will generate the following three rewriting rules in the next three subsections:

$initial([\,], \&Z) \Rightarrow.$
$initial([\&A|\&V], \&Z) \Rightarrow eq(\&Z, [\&A|\#W]), initial(\&V, \#W).$
$initial([\&A|\&V], [\&A|\&Z]) \Rightarrow initial(\&V, \&Z).$

Each of them is obtained by meta computation using meta rules (a), (b) and (c) above and the meta rules for equality atoms in Section 4.4.

6.3. **Generation of the first rule by meta computation.** We start meta computation from the following meta clause:

$h \leftarrow initial([\,], \&Z).$

By application of the meta rule (a), we have the following meta clause:

$h \leftarrow app([\,], \#Y, \&Z).$

This is split into the following two meta clauses by the meta rule (b):

$h \leftarrow eq([\,], [\,]), eq(\#Y, \&Z).$
$h \leftarrow eq([\,], [\#A|\#V]), eq(\&Z, [\#A|\#W]), app(\#V, \#Y, \#W).$

The second meta clause is removed by the meta rule (Ec). Only one meta clause is left, i.e.,

$h \leftarrow eq([\,], [\,]), eq(\#Y, \&Z).$

The first body atom $eq([\,], [\,])$ is removed by the meta rule (Eb), resulting in

$h \leftarrow eq(\#Y, \&Z).$

The meta atom $eq(\#Y, \&Z)$ is then removed by the meta rule (Ed). We thus have the following rewriting rule, which is guaranteed to be a correct ET rule:

$initial([\,], \&Z) \Rightarrow.$

6.4. **Generation of the second rule by meta computation.** We start meta computation from the following meta clause:

$h \leftarrow initial([\&A|\&V], \&Z).$

By application of the meta rule (a), we have the following meta clause:

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

This is split into the following two meta clauses by the meta rule (b):

$h \leftarrow eq([\&A|\&V], [\,]), eq(\#Y, \&Z)$.
$h \leftarrow eq([\&A|\&V], [\#A|\#V]), eq(\&Z, [\#A|\#W]), app(\#V, \#Y, \#W)$.

The first meta clause is removed by the meta rule (Ec). Only one meta clause remains, i.e.,

$h \leftarrow eq([\&A|\&V], [\#A|\#V]), eq(\&Z, [\#A|\#W]), app(\#V, \#Y, \#W)$.

The meta rule (Ea) is then applied, resulting in

$h \leftarrow eq(\&A, \#A), eq(\&V, \#V), eq(\&Z, [\#A|\#W]), app(\#V, \#Y, \#W)$.

The meta rule (Ed) is then applied twice, resulting in

$h \leftarrow eq(\&Z, [\&A|\#W]), app(\&V, \#Y, \#W)$.

It is transformed by the meta rule (c) into

$h \leftarrow eq(\&Z, [\&A|\#W]), initial(\&V, \#W)$.

Hence, we have the following rewriting rule, which is guaranteed to be a correct ET rule:

$initial([\&A|\&V], \&Z) \Rightarrow eq(\&Z, [\&A|\#W]), initial(\&V, \#W)$.

6.5. **Generation of the third rule by meta computation.** We start meta computation from the following meta clause:

$h \leftarrow initial([\&A|\&V], [\&A|\&Z])$.

By application of the meta rule (a), we have the following meta clause:

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

This is split into the following two meta clauses by the meta rule (b):

$h \leftarrow eq([\&A|\&V], [\,]), eq(\#Y, [\&A|\&Z])$.
$h \leftarrow eq([\&A|\&V], [\#A|\#V]), eq([\&A|\&Z], [\#A|\#W]), app(\#V, \#Y, \#W)$.

The first meta clause is removed by the meta rule (Ec). Only one meta clause remains, i.e.,

$h \leftarrow eq([\&A|\&V], [\#A|\#V]), eq([\&A|\&Z], [\#A|\#W]), app(\#V, \#Y, \#W)$.

The meta rules (Ea) and (Ed) are then applied, resulting in

$h \leftarrow eq(\&V, \#V), eq([\&A|\&Z], [\&A|\#W]), app(\#V, \#Y, \#W)$.

The meta rule (Ed) is next applied, yielding the meta clause

$h \leftarrow eq([\&A|\&Z], [\&A|\#W]), app(\&V, \#Y, \#W)$.

The meta rules (Ea) and (Eb) are then applied, resulting in

$h \leftarrow eq(\&Z, \#W), app(\&V, \#Y, \#W)$.

The meta rule (Ed) is next applied, yielding the meta clause

$h \leftarrow app(\&V, \#Y, \&Z)$.

It is transformed by the meta rule (c) into

$h \leftarrow initial(\&V, \&Z)$.

Hence, we have the following rewriting rule, which is guaranteed to be a correct ET rule:

$initial([\&A|\&V], [\&A|\&Z]) \Rightarrow initial(\&V, \&Z)$.

7. **Concluding Remarks.** To solve a QA problem, we first formalize it as an MI problem on first-order formulas. Many QA problems can be formulated on a class of first-order formulas that may contain *iff*-formulas. Newly generated ET rules by meta computation improve solvability of proof problems and QA problems. First-order formalization enables us to develop a general theory of rule generation that can be applied to both proof problems and QA problems.

We discussed a method of generating ET rules from logical equivalences that are correct with respect to first-order formalizations for proof problems and QA problems. This

paper introduced three theorems that explain how to generate meta rules based on logical equivalences. The correctness of these theorems was proved. A meta computation consists of repeated application of meta rules to a meta description, by which we obtain a sequence of meta descriptions. The first and the last meta descriptions of the obtained sequence give a pair of meta descriptions, which determines an ET rule.

The proposed method of rule generation consists of the following four steps: To obtain ET rules for solving a given proof/QA problem, we 1) take a first-order formalization that may include *iff*-formulas, 2) derive logical equivalences that are correct with respect to the first-order formalization, 3) generate meta rules from the logical equivalences, and 4) obtain ET rules by meta computation using the meta rules. This theory can be applied to QA problems that have been formalized by using definite clauses. To solve this class of QA problems, SLD resolution has been applied. However, SLD resolution is incomplete for this problem class due to lack of multi-head ET rules. Generation of meta rules from logical equivalences based on *iff*-formula formalization overcomes the difficulty, where useful multi-head ET rules can be sought and generated by meta computation. The method of generation of meta rules from logical equivalences can also be applied to proof problems that are formalized by using clauses and *iff*-formulas. Future research includes development of methods for deriving correct logical equivalences from a given first-order formalization.

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] K. Doets, *From Logic to Logic Programming*, The MIT Press, 1994.
[3] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.
[4] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.
[5] J. R. Slagle, Automatic theorem proving with renamable and semantic resolution, *Journal of the ACM*, vol.14, pp.687-697, 1967.
[6] R. S. Boyer, *Locking: A Restriction of Resolution*, Ph.D. Thesis, University of Texas at Austin, Texas, 1971.
[7] D. W. Loveland, A linear format for resolution, in *Symposium on Automatic Demonstration. Lecture Notes in Mathematics*, M. Laudet, D. Lacombe, L. Nolin and M. Schützenberger (eds.), Berlin, Heidelberg, Springer, 1970.
[8] D. Luckham, Refinement theorems in resolution theory, in *Symposium on Automatic Demonstration. Lecture Notes in Mathematics*, M. Laudet, D. Lacombe, L. Nolin and M. Schützenberger (eds.), Berlin, Heidelberg, Springer, 1970.
[9] K. Akama and E. Nantajeewarawat, Solving proof problems with equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.1, pp.331-344, 2022.
[10] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[11] R. A. 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] K. Akama and E. Nantajeewarawat, Solving query-answering problems based on equivalent transformation, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1547-1558, 2022.

[15] K. Akama and E. Nantajeewarawat, A foundation of logical problem solving, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1559-1570, 2022.

[16] K. Akama and E. Nantajeewarawat, Unfolding-based simplification of query-answering problems in an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.9, no.9, pp.3515-3526, 2013.

[17] K. Akama and E. Nantajeewarawat, Equivalent transformation in an extended space for solving query-answering problems, *Proc. of the 6th Asian Conference on Intelligent Information and Database Systems*, Bangkok, Thailand, pp.232-241, 2014.

[18] 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.

[19] K. Akama, E. Nantajeewarawat and T. Akama, Term rewriting that preserves models in KR-logic, *Lecture Notes in Artificial Intelligence*, vol.11431, pp.41-52, 2019.

[20] 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.

[21] 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*, Porto, Portugal, vol.2, pp.52-63, 2016.

[22] 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*, Porto, Portugal, vol.2, pp.96-103, 2016.

[23] 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*, Seville, Spain, vol.2, pp.84-95, 2018.

[24] 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.

[25] K. Akama and E. Nantajeewarawat, Computing logically with generation of equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.1, pp.315-330, 2022.

[26] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, in *Perspectives of Systems Informatics. PSI 2006. Lecture Notes in Computer Science*, I. Virbitskaite and A. Voronkov (eds.), Berlin, Heidelberg, Springer, 2007.

[27] K. Akama, H. Koike and E. Miyamoto, A theoretical foundation for generation of equivalent transformation rules, *Program Transformation, Symbolic Computation and Algebraic Manipulation*, Research Institute for Mathematical Sciences, Kyoto University, Koukyuroku, no.1125, pp.44-58, 2000.

[28] K. Akama, H. Koike and H. Mabuchi, A theoretical foundation of program synthesis by equivalent transformation, in *Perspectives of System Informatics. PSI 2001. Lecture Notes in Computer Science*, D. Bjørner, M. Broy and A. V. Zamulin (eds.), Berlin, Heidelberg, Springer, 2001.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] K. Akama and E. Nantajeewarawat, Generation of equivalent transformation rules by meta computation, *International Journal of Innovative Computing, Information and Control*, vol.18, no.2, pp.361-376, 2022.
[34] K. Akama and E. Nantajeewarawat, Representation and computation of logical problems with if-and-only-if formulas, *International Journal of Innovative Computing, Information and Control*, vol.18, no.6, pp.1799-1813, 2022.

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.