International Journal of Innovative Computing, Information and Control
Volume **9**, Number **6**, June **2013**

ICIC International ©2013 ISSN 1349-4198
pp. **2635–2650**

THEORETICAL BASIS FOR MAKING EQUIVALENT TRANSFORMATION RULES FROM LOGICAL EQUIVALENCES FOR PROGRAM SYNTHESIS

Katsunori Miura[1], Kiyoshi Akama[2], Hiroshi Mabuchi[3] and Hidekatsu Koike[4]

[1]Information Processing Center
Kitami Institute of Technology
Kitami, Hokkaido 090-8507, Japan
k-miura@mail.kitami-it.ac.jp

[2]Information Initiative Center
Hokkaido University
Sapporo, Hokkaido 060-0811, Japan
akama@iic.hokudai.ac.jp

[3]Faculty of Software and Information Science
Iwate Prefectural University
Takizawa, Iwate 020-0193, Japan
mabu@iwate-pu.ac.jp

[4]Faculty of Social Information
Sapporo Gakuin University
Ebetsu, Hokkaido 069-8555, Japan
koike@sgu.ac.jp

Received April 2012; revised August 2012

Abstract. *To propose methods for making Equivalent Transformation (ET) rules is important for generating correct and sufficiently efficient programs from a specification which is a set of logical formulas. An ET rule is a procedure for replacing a clause set with another one while preserving declarative meaning. This paper proposes a new method for making ET rules via a Logical Equivalence (LE) from a specification. An LE describes an equivalence relationship between two logical formulas under some specified preconditions. We newly formulate an LE and define the correctness of LEs with respect to a specification. It is guaranteed by the method of this paper that an ET rule can be made from a correct LE. The method is useful for the generation of various programs. Many ET rules included in programs which solve constraint satisfaction problems, can be made by the method.*

Keywords: Equivalent transformation (ET) rule, Logical equivalence (LE), Equivalence relationship of clause sets, Rule generation mapping (RGM), LE in Class $\mathcal{S}$

1. Introduction.

There are many studies [5, 8, 14] on program synthesis, each of which discusses methods for automatically generating correct programs from a specification. An Equivalent Transformation (ET) rule [3] is important for generating correct programs from a specification which is a set of logical formulas. An *ET rule* replaces a clause set with another one while preserving declarative meaning. Correct programs are generated by making ET rules from a given specification and accumulating them one by one. Sufficiently efficient programs could be generated by consciously making useful ET rules. For generating correct and sufficiently efficient programs, methods for making ET rules are important.

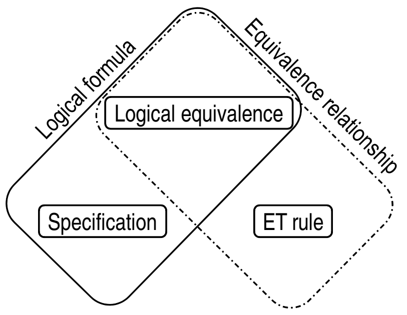

FIGURE 1. An LE is an important bridge between an ET rule and a specification

This paper proposes a new method for making ET rules via a logical formula called a *Logical Equivalence* (LE) from a given specification. In this paper, we newly formulate an LE which describes an equivalence relationship of logical formulas $\mathcal{F}1$ and $\mathcal{F}2$ under some specified preconditions $\mathcal{Z}$, where $\mathcal{F}1$ and $\mathcal{F}2$ are existentially quantified atom sets and $\mathcal{Z}$ is a set of literals. The form of each LE is $\forall(\mathcal{Z} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$, where $\forall$ is a universal closure and variables other than existentially quantified variables are quantified by $\forall$.

An LE is an important bridge between an ET rule and a specification. An ET rule is a procedure, while a specification is a logical formula. Since an ET rule is a different concept from a specification, it is not easy to guarantee that a rule made from a specification is an ET rule. An LE is a logical formula like a specification and describes an equivalence relationship like an ET rule, as shown in Figure 1. In the proposed method of this paper, it is guaranteed that an ET rule can be made via a correct LE from a given specification.

For proposing the new method using LEs, the paper carries out the following tasks.

1. Formulate an LE as being different from an ET rule.
2. Define a mapping from an LE to a rewriting rule.
3. Give a sufficient condition for the correctness of the mapping.
4. Define a class of LEs.
5. Prove that a rule made from an LE in the class by the mapping is an ET rule.

In this paper, Section 2 explains specifications, programs and computations, and defines the correctness of programs, and gives a sufficient condition for correct programs. It also discusses the importance of program synthesis using ET rules. Section 3 describes the syntax of LEs, and defines the correctness of LEs, and discusses a method for making ET rules from LEs. It also defines a mapping from an LE to a rewriting rule, and gives a sufficient condition for creating a correct mapping. Section 4 defines a class of LEs and a rule obtained by the mapping. Section 5 explains a procedure of clause transformations by rules, and defines a mapping from a rule to a relationship of clause sets. Section 6 proves that a rule obtained by a mapping shown in Section 4 is an ET rule. Section 7 shows the effectiveness of the proposed method of this paper as compared with other methods, and discusses future possibilities of the proposed method.

[Notations and definitions]

The following notations and definitions will be used. Let $\mathbb{V}$ be the set of all variables. $\mathbb{V}d$ and $\mathbb{V}r$ are defined by $\mathbb{V}d \subseteq \mathbb{V}$, $\mathbb{V}r \subseteq \mathbb{V}$ and $\mathbb{V}d \cap \mathbb{V}r = \emptyset$. Let $\mathbb{P}$ be the set of all predicates. $\mathbb{P}1$ and $\mathbb{P}2$ are defined by $\mathbb{P}1 \subseteq \mathbb{P}$, $\mathbb{P}2 \subseteq \mathbb{P}$ and $\mathbb{P}1 \cap \mathbb{P}2 = \emptyset$. $AS(\mathcal{P},\ \mathcal{V})$ is the set of atoms which are composed of a predicate set $\mathcal{P}$ and a variable set $\mathcal{V}$. $CS(\mathcal{P},\ \mathcal{P}',\ \mathcal{V})$ is the set of clauses from $AS(\mathcal{P},\ \mathcal{V})$ to $AS(\mathcal{P}',\ \mathcal{V})$. $Var(\mathcal{A})$ is the set of variables which have appeared in an atom set $\mathcal{A}$. $Term(\mathcal{V})$ is the set of terms made from variables on $\mathcal{V}$, where $\mathcal{V}$ is a variable set. Also assume that $\mathcal{V}1$ and $\mathcal{V}2$ are a variable set, a *substitution* to $\mathcal{V}2$ from $\mathcal{V}1$ is a set $\{x_1/t_1,\ x_2/t_2,\ \cdots,\ x_n/t_n\}$, and the following three conditions are satisfied:

1. $x_i \in \mathcal{V}1$ $(i = 1,\ \cdots,\ n)$,
2. $t_i \in Term(\mathcal{V}2)$ $(i = 1,\ \cdots,\ n)$,
3. $i \neq j \Rightarrow x_i \neq x_j$.

subst$(\mathcal{V}1,\ \mathcal{V}2)$ is the set of all *substitution*s from $\mathcal{V}1$ to $\mathcal{V}2$. A *renaming* to $\mathcal{V}2$ from $\mathcal{V}1$ is a substitution which replaces an element of $\mathcal{V}1$ with an element of $\mathcal{V}2$, where different variables on $\mathcal{V}1$ are replaced by different variables on $\mathcal{V}2$. *rename*$(\mathcal{V}1,\ \mathcal{V}2)$ is the set of all *renaming*s from $\mathcal{V}1$ to $\mathcal{V}2$.

2. **Program Synthesis and ET Rules.**

2.1. **Specifications, programs, and computations.**

2.1.1. *Specifications.* A *problem* is a pair $\langle \mathbb{D},\ q\rangle$, where $\mathbb{D}$ is a set of predicate definitions, representing background knowledge, and q represents a query. The aim is to generate a program which solves multiple problems, so a *specification* is a set of problems. Thus, a specification is a pair $\langle \mathbb{D},\ \mathbb{Q}\rangle$, where $\mathbb{Q}$ represents a set of queries. A specification is defined by definite clauses. For any definite clause cl_q in $\mathbb{Q}$, the predicate appearing in the head atom of cl_q exists neither in $\mathbb{D}$ nor in the body atoms of cl_q. An unlimited variety of predicates can be defined in $\mathbb{D}$. A diverse number of queries can be made by combining various atoms composed of predicates on $\mathbb{D}$. Therefore, specifications with respect to various problems can be given.

[Correct answers]

A correct answer is obtained by transforming a clause set to another one while preserving the declarative meaning. Given a set $\mathcal{D}$ of definite clauses, the declarative meaning of $\mathcal{D}$, denoted by $\mathcal{M}(\mathcal{D})$, is given by Definition 2.1.

Definition 2.1. *Declarative meaning*

Let $\mathbb{S}$ be the set of all substitutions. Let $\mathcal{G}$ be all ground atoms, where $\mathbb{G} \subseteq \mathcal{G}$. Given a set $\mathcal{A}$ of atoms, pow$(\mathcal{A})$ denotes the power set of $\mathcal{A}$. Given a definite clause cl, a mapping T_{cl} from pow$(\mathcal{G})$ to pow$(\mathcal{G})$ is defined by

$$T_{cl}(\mathbb{G}) = \{g \mid (cl = (H \leftarrow B))\ \&\ (\theta \in \mathbb{S})\ \&\ (B\theta \subseteq \mathbb{G})\ \&\ (g = H\theta \in \mathcal{G})\}.$$

Given a set $\mathcal{D}$ of definite clauses, $T_{\mathcal{D}}$ is defined by

$$T_{\mathcal{D}}(\mathbb{G}) = \{\bigcup T_{cl}(\mathbb{G}) \mid (cl \in \mathcal{D})\}.$$

$\mathcal{M}(\mathcal{D})$ is defined by

$$\mathcal{M}(\mathcal{D}) = \textstyle\bigcup_{n=1}^{\infty}[T_{\mathcal{D}}]^n(\emptyset),$$

where n is a nonnegative integer, and $[T_{\mathcal{D}}]^1(\emptyset) = T_{\mathcal{D}}(\emptyset)$, $[T_{\mathcal{D}}]^n(\emptyset) = T_{\mathcal{D}}([T_{\mathcal{D}}]^{n-1}(\emptyset))$.

Definition 2.2. *Correct answers*

An answer set $\mathbb{A}$ with respect to any query $q(\in \mathbb{Q})$ and $\mathbb{D}$ is correct iff

$$\mathbb{A} = \mathcal{M}(\mathbb{D} \cup q) - \mathcal{M}(\mathbb{D}).$$

2.1.2. *Programs.* A *program* is a set of rewriting rules, where a rewriting rule replaces a clause with one or more clauses, so a clause set is transformed into another clause set. Each rule is composed of a head part $\mathcal{X}$, a condition part $\mathcal{C}$, an execution part $\mathcal{E}_n$ and a replacement part $\mathcal{Y}_n$, where n is a nonnegative integer. $\mathcal{X}$ consists of one or more atoms while all other parts consist of zero or more atoms. Using $\mathcal{X}$, $\mathcal{C}$, $\mathcal{E}_n$ and $\mathcal{Y}_n$, a rewriting rule is described by the following syntax.

$$\begin{aligned}\mathcal{X},\ \{\mathcal{C}\} &\Rightarrow \{\mathcal{E}_1\},\ \mathcal{Y}_1;\\ &\Rightarrow \{\mathcal{E}_2\},\ \mathcal{Y}_2;\\ &\ \ \vdots\\ &\Rightarrow \{\mathcal{E}_n\},\ \mathcal{Y}_n.\end{aligned}$$

A rewriting rule applicable to a clause set is determined by a head part $\mathcal{X}$ and a condition part $\mathcal{C}$. A rule is applicable to that clause set if there exists a substitution δ that satisfies $\mathcal{X}\delta = As$, and $exec(\mathcal{C}\delta) = true$, where As is a part of body atoms appearing in a clause. $\mathcal{X}\delta$ is transformed into $\mathcal{Y}_n(\delta\theta_n \cup \rho_n)$ by applying a rewriting rule, where $exec(\mathcal{E}_n\delta) = \theta_n$, ρ_n is a *renaming* for variables that appear only in $\mathcal{Y}_n$. For example, a rewriting rule

$$\begin{aligned}r : divide([A \mid B],\ C,\ D),\ \{int(A)\} &\Rightarrow \{odd(A),\ C = [A \mid E]\},\ divide(B,\ E,\ D);\\ &\Rightarrow \{even(A),\ D = [A \mid E]\},\ divide(B,\ C,\ E).\end{aligned}$$

is applicable to

$$cl_A : \{ans(X,\ Y) \leftarrow divide([1,\ 2,\ 3],\ X,\ Y)\}.$$

By applying r to cl_A, cl_A is transformed into

$$cl_B : \{ans([1 \mid Z],\ Y) \leftarrow divide([2,\ 3],\ Z,\ Y)\}.$$

A computation is executed by repeating the transformation of a clause set.

[Correct programs]

A program $\mathcal{R}$ is correct with respect to a specification $\langle \mathbb{D},\ \mathbb{Q}\rangle$ iff for any query $q \in \mathbb{Q}$ an answer, obtained from any computation of $\mathcal{R}$ on q, is correct.

[Program synthesis]

Program synthesis in this paper is formulated as follows: given a specification $\langle \mathbb{D},\ \mathbb{Q}\rangle$, generate a program $\mathcal{R}$ such that $\mathcal{R}$ is correct with respect to $\langle \mathbb{D},\ \mathbb{Q}\rangle$ and $\mathcal{R}$ is sufficiently efficient.

2.1.3. *Computations.* A *computation* of a program $\mathcal{R}$ on a query q is to repeatedly transform a clause set, so that computation is a sequence $seq = [s_0(= q),\ s_1,\ s_2,\ \cdots]$, where s_i is a clause set, and the following conditions are satisfied:

1. For any two successive elements s_n and s_{n+1} in *seq*, s_n is transformed into s_{n+1} by a one-time application of a rule in $\mathcal{R}$,
2. If *seq* is finite, then any rule in $\mathcal{R}$ is not applicable to a last element in *seq*.

If *seq* is a finite sequence and a last state in *seq* is a set of unit clauses, then the answer set with respect to a query q is the set of all ground atoms obtained from the last state. Otherwise, the answer set cannot be obtained.

2.2. Definition of ET rules and sufficient condition for correct programs.

2.2.1. *The definition of ET rules.* An ET rule replaces a clause set with another one while preserving the declarative meaning with respect to background knowledge.

Definition 2.3. *Definition of ET rules*
A rewriting rule r is an ET rule with respect to background knowledge $\mathbb{D}$ iff the formula

$$\mathcal{M}(\mathbb{D} \cup cls1) = \mathcal{M}(\mathbb{D} \cup cls2)$$

is true for any clause sets $cls1$ and $cls2$ such that $cls1$ is transformed to $cls2$ by r.

2.2.2. *Sufficient condition for correct programs.* In program synthesis using ET rules, a sufficient condition for correct programs is given.

Proposition 2.1. *Sufficient condition for correct programs*
A program $\mathcal{R}$ is correct with respect to a specification $\langle\mathbb{D}, \mathbb{Q}\rangle$ iff the following conditions are all satisfied:

1. *A program $\mathcal{R}$ is a set of ET rules with respect to $\mathbb{D}$.*
2. *If some non-unit clause appears in q' then a rule applicable to that non-unit clause exists in $\mathcal{R}$ for any query $q(\in \mathbb{Q})$ and q' such that q' is reached from q by applying $\mathcal{R}$.*
3. *For any query $q(\in \mathbb{Q})$, every computation of $\mathcal{R}$ on q is finite.*

Conditions 1, 2, and 3 denote "partial correctness", "applicability of a program", and "termination of computation", respectively.

Proof: The correctness of Proposition 2.1 is proven in [3]. □

To satisfy a condition 1, all rewriting rules in a program must be made with ET rules. To satisfy a condition 2, if some non-unit clause appears in a clause set, then a programmer must make a rewriting rule applicable to that clause set. To satisfy a condition 3, a programmer must take care not to make a related rule which causes an infinite loop at least. Thus, a correct program is generated by making ET rules until a set of unit-clauses is obtained, while taking care not to make a related rule which causes an infinite loop.

2.3. **Importance of program synthesis using ET rules.** The following features of the ET rule could be effective for generating correct and efficient programs. ① A program composed of a set of ET rules guarantees the partial correctness with respect to a specification; ② each ET rule is completely independent and individually correct. Additionally, the previous paper [4] reports that an ET rule can describe various procedures. Therefore, a correct and sufficiently-efficient program can be generated by successively accumulating useful ET rules.

The *squeeze method* [4] had been proposed for effectively utilizing program synthesis by ET rules. The aim of the programming based on the squeeze method is not to randomly make ET rules, but to make only necessary ET rules to solve all queries. The squeeze method can be efficiently utilized if each program component can describe various correct procedures and has complete independence. Since each ET rule has the above feature, the squeeze method can be efficiently used by ET rules. As a result, various software systems [6, 12, 19], such as an e-learning system for programming education and knowledge processing systems in various data, can be developed by using ET rules.

3. Logical Equivalences and Rule Generation Mapping.

3.1. **Logical equivalences.** An LE describes an equivalence relationship of logical formulas $\mathcal{F}1$ and $\mathcal{F}2$ under some specified preconditions $\mathcal{Z}$, where $\mathcal{F}1$ and $\mathcal{F}2$ are existentially quantified atom sets and $\mathcal{Z}$ is a set of literals. The form of each LE is $\forall(\mathcal{Z} \to (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$, where $\forall$ is a universal closure and variables other than existentially quantified variables are quantified by $\forall$. A specification is a pair $\langle\mathbb{D}, \mathbb{Q}\rangle$, where $\mathbb{D}$ represents background knowledge and $\mathbb{Q}$ represents a set of queries (see Section 2.1). It is important that an LE is correct with respect to $\mathbb{D}$ when an ET rule is made by using an LE.

Definition 3.1. *Correctness of LEs*
An LE is correct with respect to $\mathbb{D}$ iff the formula

$$\mathbb{D} \models \forall(\mathcal{Z} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$$

is true.

3.2. **Rule generation mapping and its correctness.** In this section we establish a rule generation mapping (RGM) from an LE to an ET rule and propose to make ET rules by the following two steps.

1. An LE is made from a specification.
2. An ET rule is obtained from the LE by an RGM.

Let F be a subset of all LEs and let $Rule$ be a subset of all rewriting rules. An RGM is a mapping f from F to $Rule$ as shown in Figure 2. It is assumed that the output of an RGM is obtained without dependence on $\mathbb{D}$. The correctness of ET rules and LEs is dependent on $\mathbb{D}$. A correct LE with respect to $\mathbb{D}$ is given in Definition 3.1. An RGM is used with respect to various $\mathbb{D}$s, so it is desirable that the correctness of an RGM is discussed in accordance with the following condition.

An RGM is correct, iff for any $\mathbb{D}$ and any $e \in F$, if e is correct with respect to $\mathbb{D}$, then $f(e)$ is correct with respect to $\mathbb{D}$.

Therefore, strictly speaking, an ET rule is made by the following two steps.

1. A correct LE with respect to $\mathbb{D}$ is made from among F.
2. An ET rule is obtained from the LE by an RGM.

A rule made by the above step is correct with respect to $\mathbb{D}$. It is desirable that an RGM is created to output ET rules at a low cost.

3.3. **Sufficient condition for the correctness of RGM.** This section gives a sufficient condition for the correctness of an RGM in order to create a correct RGM given in Section 3.2. An ET rule is read as a set of equivalence pairs of two clause sets. Let h be a mapping from a rewriting rule to a relationship of clause sets. On the other hand, it is believed that there exists a relationship of clause sets as determined by an LE. If an equivalence relationship of clause sets can be determined by an LE, then an ET rule could be made by using a subset of the obtained relationship.

It is assumed that there exists a mapping g from an LE to a relationship of clause sets. A relationship $g(e)$ of clause sets can be determined by $e \in F$. If $g(e)$ is an equivalence relationship and a relationship $h(f(e))$ as determined by $f(e)$ satisfies $g(e) \supseteq h(f(e))$, then $f(e)$ is correct with respect to $\mathbb{D}$ (see Figure 3). $f(e)$ made from a correct e with respect to $\mathbb{D}$ is correct with respect to $\mathbb{D}$. Therefore, if an RGM satisfies the following condition, then it is guaranteed that an RGM is correct.

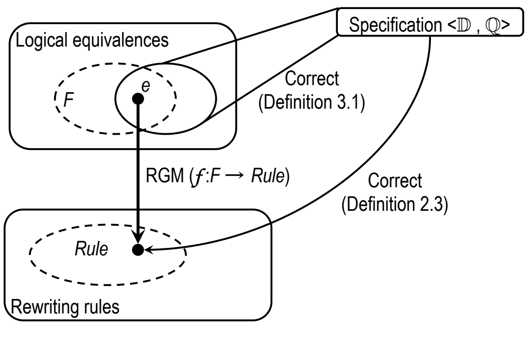

FIGURE 2. An ET rule is made from a correct LE with respect to $\mathbb{D}$

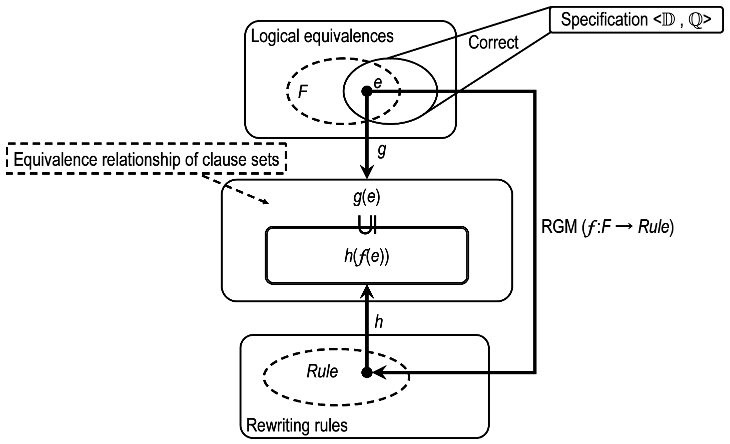

FIGURE 3. A sufficient condition for the correctness of RGM

There exists g such that

1. for any $\mathbb{D}$ and any $e \in F$, if e is correct with respect to $\mathbb{D}$ and $(\mathbb{D} \cup P_1, \mathbb{D} \cup P_2) \in g(e)$, then $\mathcal{M}(\mathbb{D} \cup P_1) = \mathcal{M}(\mathbb{D} \cup P_2)$,
2. for any $e \in F$, $g(e) \supseteq h(f(e))$.

It is proven that a rewriting rule made from a correct LE with respect to $\mathbb{D}$ by an RGM is an ET rule.

Theorem 3.1. *An RGM is correct iff f satisfies the above condition.*

Proof: From an assumption, there exists g such that

1. for any $\mathbb{D}$ and any $e \in F$, if e is correct with respect to $\mathbb{D}$ and $(\mathbb{D} \cup P_1, \mathbb{D} \cup P_2) \in g(e)$, then $\mathcal{M}(\mathbb{D} \cup P_1) = \mathcal{M}(\mathbb{D} \cup P_2)$,
2. for any $e \in F$, $g(e) \supseteq h(f(e))$.

Let $\mathbb{D}$ be any background knowledge and let e be any element in F. Assume that e is correct with respect to $\mathbb{D}$. Let $(\mathbb{D} \cup P_1,\ \mathbb{D} \cup P_2)$ be any element in $h(f(e))$.

From a condition 2, since $g(e) \supseteq h(f(e))$, $(\mathbb{D} \cup P_1,\ \mathbb{D} \cup P_2) \in g(e)$. From a condition 1, $\mathcal{M}(\mathbb{D} \cup P_1) = \mathcal{M}(\mathbb{D} \cup P_2)$. From Definition 2.3, $f(e)$ is correct with respect to $\mathbb{D}$.

□

4. **Rule Generation Using LEs in Class $\mathcal{S}$.**

4.1. **LEs in class $\mathcal{S}$ and its examples.**

An LE that can be represented by the form $\forall(\mathcal{Z} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$ exists variously. LEs treated with this paper are LEs on $D(\subseteq CS(\mathbb{P}1,\ \mathbb{P}1,\ \mathbb{V}r))$, $\mathbb{P}1$, and $\mathbb{V}r$, the form of which is

$$\forall(\mathcal{X} \leftrightarrow \exists_{\overline{y}}\mathcal{Y}),$$

where $\mathcal{X}$ and $\mathcal{Y}$ are subsets of $AS(\mathbb{P}1,\ \mathbb{V}r)$, and $\overline{y} = Var(\mathcal{Y}) - Var(\mathcal{X})$. This form is a special form of $\forall(\mathcal{Z} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$, and is a subclass. $\mathcal{Z}$ can be an empty set. In this case the form of an LE is $\forall(\{\ \} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$ which is equivalent to $\forall(\mathcal{F}1 \leftrightarrow \mathcal{F}2)$. The form of $\mathcal{F}1$ and $\mathcal{F}2$ is $\exists_{a_1}\exists_{a_2}\cdots\exists_{a_n}\mathcal{A}s$, where $\mathcal{A}s$ is an atom set and n is a nonnegative integer. $\exists_{a_1}\exists_{a_2}\cdots\exists_{a_n}$ of this form can be abbreviated to $\exists_{\{a_1,\ a_2,\cdots,a_n\}}$ using the form of a variable set. Thus, $\exists_{a_1}\exists_{a_2}\cdots\exists_{a_n}\mathcal{A}s$ can be written as $\exists_{\{a_1,\ a_2,\cdots,a_n\}}\mathcal{A}s$. If the number of existentially quantified variables on $\mathcal{F}1$ is zero and the number of existentially quantified variables on $\mathcal{F}2$ is zero or more, then the form of an LE is $\forall(\exists_{\{\ \}}\mathcal{X} \leftrightarrow \exists_{\{y_1,\ y_2,\ \cdots,\ y_n\}}\mathcal{Y})$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of atoms. $\exists_{\{\ \}}\mathcal{X}$ can be abbreviated to $\mathcal{X}$. Letting $\overline{y} = \{y_1,\ y_2,\ \cdots,\ y_n\}$, the form of an LE in this class is $\forall(\mathcal{X} \leftrightarrow \exists_{\overline{y}}\mathcal{Y})$. In this paper, LEs of this form is called Class $\mathcal{S}$. Since $\overline{y}$ is determined by $\mathcal{X}$ and $\mathcal{Y}$, an LE in this class is denoted by $LE(\mathcal{X},\ \mathcal{Y})$.

We present four relevant examples with respect to LEs in Class $\mathcal{S}$. $app(A1,\ A2,\ A3)$ means that the concatenation of lists $A1$ and $A2$ is a list $A3$. $rev(A1,\ A2)$ means that a list $A2$ is elements of a list $A1$ in reverse order. $eq(A1,\ A2)$ means that $A1$ is equal to $A2$. $false$ means that $\mathcal{X}$ of LE includes atoms which cannot satisfy the constraint of the predicate.

Example 4.1. *LEs in Class $\mathcal{S}$*

$$le_1 : \forall(\{app([X \mid Y],\ Z,\ V)\} \leftrightarrow \exists_{\{W\}}\{eq(V,\ [X \mid W]),\ app(Y,\ Z,\ W)\})$$
$$le_2 : \forall(\{rev([X \mid Y],\ Z)\} \leftrightarrow \exists_{\{W\}}\{rev(Y,\ W),\ app(W,\ [X],\ Z)\})$$
$$le_3 : \forall(\{rev(X,\ Y),\ rev(X,\ Z)\} \leftrightarrow \exists_{\{\ \}}\{eq(Y,\ Z),\ rev(X,\ Y)\})$$
$$le_4 : \forall(\{app(X,\ [Y \mid Z],\ [Z])\} \leftrightarrow \exists_{\{\ \}}\{false\})$$

The first LE le_1 describes a relationship in which $\{app([X \mid Y],\ Z,\ V)\}$ is equivalent to $\exists_{\{W\}}\{eq(V,\ [X \mid W]),\ app(Y,\ Z, W)\}$. le_2 describes a relationship in which $\{rev([X \mid Y],\ Z)\}$ is equivalent to $\exists_{\{W\}}\{rev(Y,\ W),\ app(W,\ [X],\ Z)\}$. le_3 describes a relationship in which $\{rev(X,\ Y),\ rev(X,\ Z)\}$ is equivalent to $\exists_{\{\ \}}\{eq(Y,\ Z),\ rev(X,\ Y)\}$. le_4 describes a relationship in which $\{app(X,\ [Y \mid Z],\ Z)\}$ is equivalent to $\exists_{\{\ \}}\{false\}$.

A set F as an input of an RGM is defined. It is desirable that F is given to make an ET rule that is useful for generating an efficient program. F can be arbitrarily given from among a set of all LEs represented by the form $\forall(\mathcal{Z} \rightarrow (\mathcal{F}1 \leftrightarrow \mathcal{F}2))$. In this paper, F is a set of all LEs in Class $\mathcal{S}$. Any element in F is $LE(\mathcal{X},\ \mathcal{Y})$. LEs in Class $\mathcal{S}$ has a simple form, but this class is very useful and many ET rules can be made from LEs in this class. As this paper focuses on LEs in Class $\mathcal{S}$, from here on when say an LE, we are referring to this class.

4.2. **Rules made by using LEs in Class $\mathcal{S}$.** A rewriting rule is made from $LE(\mathcal{X},\ \mathcal{Y})$ by an RGM. A set *Rule* as an output of an RGM is defined. *Rule* is a set of rewriting rules on $D(\subseteq CS(\mathbb{P}1,\ \mathbb{P}1,\ \mathbb{V}r))$, $\mathbb{P}1$, and $\mathbb{V}r$, the form of rules is

$$\mathcal{X} \Rightarrow \mathcal{Y},$$

where $\mathcal{X}$ and $\mathcal{Y}$ are subsets of $AS(\mathbb{P}1,\ \mathbb{V}r)$. In this paper, a rule of this form is denoted by $rule(\mathcal{X},\ \mathcal{Y})$. An RGM relates an element $LE(\mathcal{X},\ \mathcal{Y})$ in F to an element $rule(\mathcal{X},\ \mathcal{Y})$ in *Rule*.

5. Generation of Relationship of Clause Sets Based on Rewriting Rules.

5.1. **Clause transformation.** Since the body part of $rule(\mathcal{X},\ \mathcal{Y})$ is a single body, a clause cl_1 to which $rule(\mathcal{X},\ \mathcal{Y})$ is applied is transformed to a clause cl_2. A clause cl_1 is

$$H \leftarrow As,\ B,$$

where H is an atom in $AS(\mathbb{P}2,\ \mathbb{V}d)$, As and B are subsets of $AS(\mathbb{P}1,\ \mathbb{V}d))$. If the following conditions are satisfied, then $rule(\mathcal{X},\ \mathcal{Y})$ is applicable to As of cl_1.

$$\textbf{Condition 1}: \delta \in subst(Var(\mathcal{X}), \mathbb{V}d)$$
$$\textbf{Condition 2}: \mathcal{X}\delta = As$$

$rule(\mathcal{X},\ \mathcal{Y})$ is applied to As of cl_1 based on a substitution δ, consequently cl_1 is transformed to

$$cl_2 = (H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B),$$

where ρ is an arbitrary element in $rename((Var(\mathcal{Y}) - Var(\mathcal{X})),\ (\mathbb{V}d - Var(H \cup B \cup \mathcal{X}\delta)))$.

5.2. **Definition of mapping h.** Given an element $rule(\mathcal{X},\ \mathcal{Y})$ in $Rule$, a mapping h determines $set(\mathcal{X},\ \mathcal{Y})$. A relationship $set(\mathcal{X},\ \mathcal{Y})$ of clause sets is defined as:

$$\begin{aligned} set(\mathcal{X},\ \mathcal{Y}) = \{(&\{H \leftarrow \mathcal{X}\delta,\ B\} \cup Q \cup D,\ \{H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B\} \cup Q \cup D) \mid \\ &(① \ H \in AS(\mathbb{P}2,\ \mathbb{V}d)) \ \&\ (② \ B \subseteq AS(\mathbb{P}1,\ \mathbb{V}d)) \ \& \\ &(③ \ Q \subseteq CS(\mathbb{P}1,\ \mathbb{P}2,\ \mathbb{V}d)) \ \&\ (④ \ D \subseteq CS(\mathbb{P}1,\ \mathbb{P}1,\ \mathbb{V}d)) \ \& \\ &(⑤ \ \rho \in rename((Var(\mathcal{Y}) - Var(\mathcal{X})),\ (\mathbb{V}d - Var(H \cup B \cup \mathcal{X}\delta)))) \ \& \\ &(⑥ \ \delta(\in subst(\mathbb{V}r,\ \mathbb{V}d)) \text{ on } Var(\mathcal{X}))\}. \end{aligned}$$

A clause $H \leftarrow \mathcal{X}\delta,\ B$ is a clause cl_1, a clause $H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B$ is a clause cl_2, so $\{H \leftarrow \mathcal{X}\delta,\ B\} \cup Q$ is a query set in which $rule(\mathcal{X},\ \mathcal{Y})$ applied, $\{H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B\} \cup Q$ is a query set obtained by applying $rule(\mathcal{X},\ \mathcal{Y})$. D is background knowledge.

6. Correctness of Rule Generation Using LEs in Class $\mathcal{S}$.

6.1. **Overview.** In this section, it is proven that an RGM that relates $LE(\mathcal{X},\ \mathcal{Y})$ in F to $rule(\mathcal{X},\ \mathcal{Y})$ in $Rule$ is correct. If an RGM satisfies a sufficient condition given in Section 3.3, then an RGM is correct. The correctness of an RGM is proven by the following procedure.

1. Define a mapping g. (Section 6.2)
   A mapping g determines $R(\mathcal{X},\ \mathcal{Y})$ for an element $LE(\mathcal{X},\ \mathcal{Y})$ in F.
2. Prove that if $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D and $(D \cup P_1, D \cup P_2) \in R(\mathcal{X},\ \mathcal{Y})$, then $\mathcal{M}(D \cup P_1) = \mathcal{M}(D \cup P_2)$. (Section 6.3)
3. Prove that $R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$. (Section 6.4)
4. Prove that if $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D, then an RGM is correct. (Section 6.5)

6.2. **Definition of mapping g.** Given an element $LE(\mathcal{X},\ \mathcal{Y})$ in F, a mapping g determines $R(\mathcal{X},\ \mathcal{Y})$. A relationship $R(\mathcal{X},\ \mathcal{Y})$ of clause sets is defined as:

$$\begin{aligned} R(\mathcal{X},\ \mathcal{Y}) = \{(&\{H \leftarrow \mathcal{X}\delta,\ B\} \cup Q \cup D,\ \{H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B\} \cup Q \cup D) \mid \\ &(① \ H \in AS(\mathbb{P}2,\ \mathbb{V}d)) \ \&\ (② \ B \subseteq AS(\mathbb{P}1,\ \mathbb{V}d)) \ \& \\ &(③ \ Q \subseteq CS(\mathbb{P}1,\ \mathbb{P}2,\ \mathbb{V}d)) \ \&\ (④ \ D \subseteq CS(\mathbb{P}1,\ \mathbb{P}1,\ \mathbb{V}d)) \ \& \\ &(⑤ \ \rho \in rename((Var(\mathcal{Y}) - Var(\mathcal{X})),\ (\mathbb{V}d - Var(H \cup B \cup \mathcal{X}\delta)))) \ \& \\ &(⑥ \ \delta(\in subst(\mathbb{V}r,\ \mathbb{V}d)) \text{ on } Var(\mathcal{X}))\}. \end{aligned}$$

Clause sets $\{H \leftarrow \mathcal{X}\delta,\ B\}$, $\{H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B\}$, and Q are subsets of $CS(\mathbb{P}1,\ \mathbb{P}2,\ \mathbb{V}d)$ as with a query set. A clause set D is a subset of $CS(\mathbb{P}1,\ \mathbb{P}1,\ \mathbb{V}d)$ as with background knowledge.

6.3. **Proof with respect to $\mathcal{M}(D \cup P_1) = \mathcal{M}(D \cup P_2)$.** If a set $\{H \leftarrow \mathcal{X}\delta, B\}$ in $R(\mathcal{X},\ \mathcal{Y})$ is equivalent to a set $\{H \leftarrow \mathcal{Y}(\delta \cup \rho), B\}$, then $R(\mathcal{X},\ \mathcal{Y})$ is an equivalence relationship of clause sets. In this section, it is proven that if $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D, then $R(\mathcal{X},\ \mathcal{Y})$ is an equivalence relationship of clause sets.

Proposition 6.1. *If $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D and $(D \cup P_1,\ D \cup P_2) \in R(\mathcal{X},\ \mathcal{Y})$, then $\mathcal{M}(D \cup P_1) = \mathcal{M}(D \cup P_2)$.*

Proof: Assume that $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D and $(D \cup P_1,\ D \cup P_2) \in R(\mathcal{X},\ \mathcal{Y})$. There exist

$$\begin{aligned} &P_1 = \{cl_1\} \cup Q, \\ &P_2 = \{cl_2\} \cup Q, \\ &cl_1 = (H \leftarrow \mathcal{X}\delta,\ B), \\ &cl_2 = (H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B), \end{aligned}$$

an atom H, an atom set B, definite clause sets Q and D, substitutions ρ and δ, and definite clauses cl_1 and cl_2, where all conditions of $R(\mathcal{X},\ \mathcal{Y})$ are satisfied. To prove that $T_{cl_1}(\mathcal{M}(D)) = T_{cl_2}(\mathcal{M}(D))$, it is proven that $T_{cl_1}(\mathcal{M}(D)) \subseteq T_{cl_2}(\mathcal{M}(D))$ and $T_{cl_1}(\mathcal{M}(D)) \supseteq T_{cl_2}(\mathcal{M}(D))$ hold.

Ⓐ : $T_{cl_1}(\mathcal{M}(D)) \subseteq T_{cl_2}(\mathcal{M}(D))$

Assume $h \in T_{cl_1}(\mathcal{M}(D))$. Since definite clause $cl_1 = (H \leftarrow \mathcal{X}\delta,\ B)$, there exists a ground substitution γ on $Var(H \cup \mathcal{X}\delta \cup B)$ that satisfies $H\gamma = h$, $\mathcal{X}\delta\gamma \subseteq \mathcal{M}(D)$ and $B\gamma \subseteq \mathcal{M}(D)$, where $H\gamma$ is a ground atom.

A renaming ρ satisfies a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, a substitution δ satisfies a condition ⑥ of $R(\mathcal{X},\ \mathcal{Y})$, and γ is a substitution on $Var(H \cup \mathcal{X}\delta \cup B)$. From Proposition A.2, $\mathcal{X}\delta\gamma \subseteq \mathcal{M}(D)$ holds and there exists a substitution ω such that $\mathcal{Y}(\delta \cup \rho)\gamma\omega \subseteq \mathcal{M}(D)$ holds.

A definite clause $cl_2{}' = (H(\gamma\omega) \leftarrow \mathcal{Y}(\delta \cup \rho)(\gamma\omega),\ B(\gamma\omega))$ is obtained by applying $\gamma\omega$ to $cl_2 = (H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B)$. Since $H\gamma$ is a ground atom and $B\gamma$ is a set of ground atoms, $H(\gamma\omega) = H\gamma$ and $B(\gamma\omega) = B\gamma$, respectively. Additionally, since $H\gamma = h$, $cl_2{}' = (h \leftarrow \mathcal{Y}(\delta \cup \rho)\gamma\omega,\ B\gamma)$. Since $\mathcal{Y}(\delta \cup \rho)\gamma\omega \subseteq \mathcal{M}(D)$ and $B\gamma \subseteq \mathcal{M}(D)$, $h \in T_{cl_2}(\mathcal{M}(D))$. Consequently, $T_{cl_1}(\mathcal{M}(D)) \subseteq T_{cl_2}(\mathcal{M}(D))$.

Ⓑ : $T_{cl_1}(\mathcal{M}(D)) \supseteq T_{cl_2}(\mathcal{M}(D))$

Assume $h \in T_{cl_2}(\mathcal{M}(D))$. Since definite clause $cl_2 = (H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B)$, there exists a ground substitution γ on $Var(H \cup \mathcal{Y}(\delta \cup \rho) \cup B)$ that satisfies $H\gamma = h$, $\mathcal{Y}(\delta \cup \rho)\gamma \subseteq \mathcal{M}(D)$ and $B\gamma \subseteq \mathcal{M}(D)$, where $H\gamma$ is a ground atom. A substitution γ is the composition of $\gamma 1$ and $\gamma 2$, where $\gamma 1$ applies to variables on $Var((H \cup B \cup \mathcal{Y}(\delta \cup \rho)) - \overline{y}\rho)$ and $\gamma 2$ applies to variables on $\overline{y}\rho$.

A renaming ρ satisfies a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, a substitution δ satisfies a condition ⑥ of $R(\mathcal{X},\ \mathcal{Y})$, $\gamma 1$ is a substitution on $Var((H \cup B \cup \mathcal{Y}(\delta \cup \rho)) - \overline{y}\rho)$, and $\gamma 2$ is a substitution on $\overline{y}\rho$. From Proposition A.3, $\mathcal{Y}(\delta \cup \rho)\gamma 1\gamma 2 \subseteq \mathcal{M}(D)$ holds and for any ground substitution ω, $\mathcal{X}\delta\gamma 1\omega \subseteq \mathcal{M}(D)$ holds.

A definite clause $cl_1{}' = (H(\gamma 1\omega) \leftarrow \mathcal{X}\delta(\gamma 1\omega),\ B(\gamma 1\omega))$ is obtained by applying $\gamma 1\omega$ to $cl_1 = (H \leftarrow \mathcal{X}\delta,\ B)$. $\gamma 1$ is a substitution which applies to variables on $(Var(H \cup B \cup \mathcal{Y}(\delta \cup \rho)) - \overline{y}\rho)$. Since $H\gamma$ is a ground atom, $H\gamma 1$ also is a ground atom, so $H\gamma 1 = H\gamma$. Thus, $H(\gamma 1\omega) = H\gamma$. Since $B\gamma$ is a set of ground atoms, $B\gamma 1$ also is a set of ground atoms, so $B\gamma 1 = B\gamma$. Thus, $B(\gamma 1\omega) = B\gamma$. Additionally, since $H\gamma = h$, $cl_1{}' = (h \leftarrow \mathcal{X}\delta\gamma 1\omega,\ B\gamma)$. Since $\mathcal{X}\delta\gamma 1\omega \subseteq \mathcal{M}(D)$ and $B\gamma \subseteq \mathcal{M}(D)$, $h \in T_{cl_1}(\mathcal{M}(D))$. Consequently, $T_{cl_1}(\mathcal{M}(D)) \supseteq T_{cl_2}(\mathcal{M}(D))$.

From Ⓐ and Ⓑ, it is proven that $T_{cl_1}(\mathcal{M}(D)) = T_{cl_2}(\mathcal{M}(D)) \cdots$ Ⓒ. If predicates appearing in D do not appear in the head atom of Q, then the following formula is true.

$$\mathcal{M}(D \cup Q) = \mathcal{M}(D) \cup \{\bigcup T_{cl}(\mathcal{M}(D)) \mid (cl \in Q)\}$$

In this paper, the proof of the correctness of the above formula is omitted. Therefore, the following formula is true.

$$\begin{aligned}
\mathcal{M}(D \cup P_1) &= \mathcal{M}(D) \cup \left\{\bigcup T_{cl}(\mathcal{M}(D)) \mid cl \in (\{cl_1\} \cup Q)\right\} \\
&= \mathcal{M}(D) \cup \left\{\bigcup T_{cl}(\mathcal{M}(D)) \mid cl \in Q\right\} \cup T_{cl_1}(\mathcal{M}(D)) \\
&= \{\text{From Ⓒ}\} \\
&= \mathcal{M}(D) \cup \left\{\bigcup T_{cl}(\mathcal{M}(D)) \mid cl \in Q\right\} \cup T_{cl_2}(\mathcal{M}(D)) \\
&= \mathcal{M}(D) \cup \left\{\bigcup T_{cl}(\mathcal{M}(D)) \mid cl \in (\{cl_2\} \cup Q)\right\} = \mathcal{M}(D \cup P_2)
\end{aligned}$$

Consequently, $\mathcal{M}(D \cup P_1) = \mathcal{M}(D \cup P_2)$ is proven. □

6.4. **Proof with respect to $R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$.** If any element pa in $set(\mathcal{X},\ \mathcal{Y})$ satisfies the condition, from ① to ⑥, of $R(\mathcal{X},\ \mathcal{Y})$, then pa is an element in $R(\mathcal{X},\ \mathcal{Y})$. In this section, it is proven that $R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$.

Proposition 6.2. *$R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$.*

Proof: Any element pa in a relationship $set(\mathcal{X},\ \mathcal{Y})$ is represented as a pair $(D \cup P_1,\ D \cup P_2)$ of clause set. There exist

$$\begin{aligned} P_1 &= \{cl_1\} \cup Q, \\ P_2 &= \{cl_2\} \cup Q, \\ cl_1 &= (H \leftarrow \mathcal{X}\delta,\ B), \\ cl_2 &= (H \leftarrow \mathcal{Y}(\delta \cup \rho),\ B), \end{aligned}$$

an atom H, an atom set B, definite clause sets Q and D, substitutions ρ and δ, and definite clauses cl_1 and cl_2, where all conditions of $set(\mathcal{X},\ \mathcal{Y})$ are satisfied. A pa satisfies the condition, from ① to ⑥, of $R(\mathcal{X},\ \mathcal{Y})$, so that $pa \in R(\mathcal{X},\ \mathcal{Y})$. Consequently, $R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$. □

6.5. **Proof with respect to the correctness of RGM.** In this section, it is proven that if an element $LE(\mathcal{X},\ \mathcal{Y})$ in F is correct with respect to D, then an RGM from F to *Rule* is correct. It is guaranteed by applying Theorem 6.1 that $rule(\mathcal{X},\ \mathcal{Y})$ made from a correct $LE(\mathcal{X},\ \mathcal{Y})$ with respect to D is an ET rule.

Theorem 6.1. *If $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D, then an RGM is correct.*

Proof: Assume that $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D. Assume that $rule(\mathcal{X},\ \mathcal{Y})$ is obtained from $LE(\mathcal{X},\ \mathcal{Y})$ by an RGM, $set(\mathcal{X},\ \mathcal{Y})$ is obtained from $rule(\mathcal{X},\ \mathcal{Y})$ by a mapping h, and $R(\mathcal{X},\ \mathcal{Y})$ is obtained from $LE(\mathcal{X},\ \mathcal{Y})$ by a mapping g. Let $(D \cup P_1,\ D \cup P_2)$ be an element in $R(\mathcal{X},\ \mathcal{Y})$. From Proposition 6.1, $\mathcal{M}(D \cup P_1) = \mathcal{M}(D \cup P_2)$. From Proposition 6.2, $R(\mathcal{X},\ \mathcal{Y}) \supseteq set(\mathcal{X},\ \mathcal{Y})$. Since two conditions of Theorem 3.1 is satisfied, an RGM is correct. □

7. Discussion.

7.1. **Comparison with other theories.** To generate efficient programs, it is important that knowledge as a problem description and a procedure for solving problems are individually defined. In the program synthesis using ET rules, knowledge and a procedure are strictly defined by a logical formula and an ET rule, respectively. Thus, we can discuss how to make a useful ET rule from a logical formula. On the other hand, some formulas called a *rule* had been proposed by previous researches. The programming in CHR [1, 9, 16], LP [7, 13, 18] and CLP [10, 17] is to define declarative knowledge using a logical formula, a procedure is obtained by the procedural interpretation of the logical formula. Thus, a rule is the logical formula. The programming in the production system [11] is to make the formula written in the "if α then β" format, so a rule is this formula, where knowledge is not strictly defined. In these frameworks, since logical formulas and rules are treated as one, the concept of rules is not independent of that of logical formulas, so problem settings seeking better rules from logical formulas do not exist. Therefore, these frameworks cannot define a relationship of clause sets shown in this paper and propose a method for making a rule from a logical formula.

7.2. **Comparison with meta-computation-based method.** A meta-computation-based method [2] makes ET rules by replacing *meta-clauses* using *meta-rules*. The range of ET rules possibly made by the meta-computation-based method differs from the range of ET rules by the proposed method of this paper. By a combination of their methods, many useful ET rules would be made.

The proposed method can make the following rule, while the meta-computation-based method cannot.

$$r_1 : rev(X,\ Y),\ rev(X,\ Z) \Rightarrow \{Y = Z\},\ rev(X,\ Y).$$

The definition of the *rev* atom in r_1 is explained in Section 4.1. The rule r_1 means that if two *rev* atoms exist in a body part of a clause and the first arguments in the *rev* atoms are same variables, then the second arguments in the *rev* atoms are equalized based on the functionality of the *rev* atom. The rule r_1 is a rule in the class of *Speq rules* [15]. Rules in the class are important for solving constraint satisfaction problems [4].

In the proposed method of this paper, the rule r_1 is made by the following three steps.

1. Prove that a formula $\forall(rev(X\ Y),\ rev(X,\ Z) \to eq(Y,\ X))$ holds.
2. Make an LE $\forall(\{rev(X,\ Y),\ rev(X,\ Z)\} \leftrightarrow \exists_{\{\ \}}\{eq(Y,\ Z),\ rev(X,\ Y)\})$ from the formula.
3. Make the rule r_1 from the LE.

A reason why the rule r_1 cannot be made by the meta-computation-based method is that the variable X in r_1 presents lists of any length. If the first argument in the *rev* atom of r_1 is a variable, then a mechanism of an induction is necessary. The mechanism is incorporated in the proposed method, while it is not in the meta-computation-based method.

7.3. **Generation of useful rules by the proposed method.** An ET rule of the form $(\mathcal{X} \Rightarrow \mathcal{Y})$ can be made from an LE in Class $\mathcal{S}$ by the proposed method. The proposed method can make often-used ET rules in actual programming. They include a *Speq* rule [15] which is useful for an efficient program generation. To seek new class of LEs is important for making more useful ET rules. For example, the following form with a precondition $\mathcal{Z}$ can be made based on the form of Class $\mathcal{S}$.

$$\forall(\mathcal{Z} \to (\mathcal{X} \leftrightarrow \exists_{\overline{y}}\mathcal{Y}))$$

This LE means that if $\mathcal{Z}$ is true then $\mathcal{X}$ is equivalent to $\exists_{\overline{y}}\mathcal{Y}$. The part of ET rules with a condition part could be made by using this LE. For a program which uses ET rules to be considered efficient, it is essential that answer sets are obtained by carrying out the least possible number of clause transformations. An useful ET rule can be efficiently used by taking control of a rule application by a condition part of rules. Since this paper gives a foundation method for making ET rules using an LE, the concept of this method can be applied when an ET rule is made from an LE of above form.

7.4. **Automatic generation of ET rules based on the proposed method.** A method for automatically generating an LE from a specification had been proposed by the research paper [15]. The method seeks an LE based on the following form le_{sp} and automatically proves the correctness of the obtained LE. An LE generated by the method is a correct LE with respect to background knowledge $\mathbb{D}$.

$$le_{sp} : \forall(\mathcal{X} \leftrightarrow \{eq(P,\ Q)\} \cup \mathcal{X}),$$

where $\mathcal{X}$ is the set of atoms, $P \in Var(\mathcal{X})$ and $Q \in Var(\mathcal{X})$ satisfy $P \neq Q = P\tau$, and τ is a substitution on $Var(\mathcal{X})$. An LE of this form is a special form of Class $\mathcal{S}$. Since $\overline{y}$ of this form is an empty set, $\exists_{\overline{y}}$ is omitted from these forms. It is guaranteed by Theorem 6.1

shown in Section 6.5 that a rewriting rule made from a correct LE with respect to $\mathbb{D}$ is an ET rule. Therefore, ET rules of the following form could be generated by a combination of results of study [15] and this paper.

$$\mathcal{X} \Rightarrow eq(P,\ Q),\ \mathcal{X}.$$

7.5. **Effectiveness of the proposed method in program construction.** This section shows how many programs we were actually able to construct based on the proposed method. We targeted programs constructed by students in practicing-programming education. This practice presented thirty-three (33) problems which are constraint satisfaction problems, such as Numberlink and Sudoku. Programs consist of four hundred and eight (408) rules. Of them, the proposed method can make rules of the form ($\mathcal{X} \Rightarrow \mathcal{Y}$), such as:

$r_1 : numList([X \mid Y]) \Rightarrow num(X),\ numList(Y).$
$r_2 : notMember(X,\ [Y \mid Z]) \Rightarrow notMember(X,\ Y),\ notMember(X,\ Z).$
$r_3 : neq(A,\ X),\ member(X,\ [Y,\ A \mid Z]) \Rightarrow neq(A,\ X),\ member(X,\ [Y, \mid Z]).$

A rule r_1 replaces one atom with two atoms and defines that a number list $[X \mid Y]$ consists of one digit X and a number list Y composed of zero or more elements. A rule r_2 replaces one atom with two atoms and defines that an element X does not appear in elements of lists Y and Z. A rule r_3 replaces two atoms with two atoms and defines that since A is not equal to X, an element A of a list $[Y,\ A \mid Z]$ can be removed.

From this survey, we saw that one hundred and seventy-nine (179) rules, representing forty-four percent (44%) of the total rules, could be made based on the proposed method. Additionally, we saw that a rule of the form ($\mathcal{X} \Rightarrow \mathcal{Y}$) is included in all programs of all problems and sixty percent (60%) rules of a certain program are this form. By extending the proposed method, it is believed that rules of the form ($\mathcal{X}, \{\mathcal{C}\} \Rightarrow \mathcal{Y}$) can be made. If the extending method is obtained, ET rules which can be made by the extending method are increased eleven percent (11%). We understood that many useful ET rules can be made by the proposed method and two hundred and twenty-three (223) rules (equivalent to fifty-five percent (55%) of the total rules) could be made by the extending method. Consequently, the proposed method is efficient in the program construction and the method can be fully developed in the future.

8. Conclusions.

This paper has proposed a new method which makes ET rules from an LE by an RGM. We have given a sufficient condition for the correctness of an RGM and proven that if an RGM satisfies a sufficient condition, then a rewriting rule made by an RGM is an ET rule. The proposed method makes it possible to make useful ET rules for program synthesis. Furthermore, the proposed method is a foundation for making ET rules from a more difficult and complex LE. In future work we will try to discover other important classes of LEs for making useful ET rules and also propose methods for making ET rules from these new classes of LEs.

REFERENCES

[1] D. Aguilar-Solis, Learning semantic parsers: A constraint handling rule approach, *Lecture Notes in Computer Science*, pp.447-448, 2006.

[2] 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.

[3] K. Akama and E. Nantajeewarawat, Formalization of the equivalent transformation computation model, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol.10, no.3, pp.245-259, 2006.

[4] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, *Proc. of PSI2006, LNCS*, vol.4378, pp.41-54, 2007.
[5] D. Batory, Program refactoring, program synthesis, and model-driven development, *Proc. of the 16th International Conference on Compiler Construction*, pp.156-171, 2007.
[6] Z. Cheng, K. Akama and T. Tsuchida, Solving "all-solution" problems by et-based generation of programs, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(A), pp.4583-4595, 2009.
[7] Y. Deville and K. Lau, Logic program synthesis, *The Journal of Logic Programming*, vol.19, no.20, pp.321-350, 1994.
[8] P. Flener, *Logic Program Synthesis from Incomplete Information*, Kluwer Academic Publishers, 1994.
[9] T. Frühwirth, Theory and practice of constraint handling rules, *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, vol.37, no.1-3, pp.95-138, 1998.
[10] J. Jaffar and J. L. Lassez, Constraint logic programming, *Technical Report*, Department of Computer Science, Monash University, 1986.
[11] D. Klahr, P. Langley and R. Neches, *Production System Models of Learning and Development*, MIT Press, 1987.
[12] H. Koike, T. Ishikawa, K. Akama, M. Chiba and K. Miura, Developing an e-learning system which enhances students' academic motivation, *Proc. of the 33rd Annual ACM SIGUCCS Conference on User Services*, pp.147-150, New York, USA, 2005.
[13] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[14] Z. Manna and R. J. Waldinger, Toward automatic program synthesis, *Communications of the ACM*, vol.14, no.3, pp.151-165, 1971.
[15] K. Miura, K. Akama and H. Mabuchi, Generating Speq rules based on automatic proof of logical equivalence, *International Journal of Computer Science*, vol.3, no.3, pp.190-198, 2008.
[16] J. Sneyers, T. Schrijvers and B. Demoen, The computational power and complexity of constraint handling rules, *Journal of ACM Transactions on Programming Languages and Systems*, vol.31, no.2, pp.8:1-8:42, 2009.
[17] P. van Hentenryck, Constraint logic programming, *The Knowledge Engineering Review*, vol.6, no.3, pp.151-194, 1991.
[18] D. W. Loveland and G. Nadathur, Proof procedures for logic programming, in *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger and J. A. Robinson (eds.), Oxford University Press, 1998.
[19] H. Yoshikawa, K. Akama and H. Mabuchi, Et-based distributed cooperative system, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(A), pp.4655-4666, 2009.

Appendix.

Proposition A.1. *θ_1 is a substitution on $Var(H)$ in $LE(H,\ B)$, θ_2 is a renaming for $\overline{b}(= Var(B) - Var(H))$. If $Var(H) \cap \overline{b} = \emptyset$, $Var(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$, $Var(H)\theta_1 \cap \overline{b} = \emptyset$, and $LE(H,\ B)$ hold, then $LE(H\theta_1,\ B(\theta_1 \cup \theta_2))$ holds.*

Proof: Assume that $Var(H) \cap \overline{b} = \emptyset$, $Var(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$, and $Var(H)\theta_1 \cap \overline{b} = \emptyset$ hold. Assume that $LE(H,\ B)$ is

$$\forall(H \leftrightarrow \exists_{\overline{b}}(B)),$$

where $\overline{b} = Var(B) - Var(H)$. Since a substitution θ_1 changes only variables on $Var(H)$, from $LE(H,\ B)$,

$$\forall(H\theta_1 \leftrightarrow \exists_{\overline{b}}(B\theta_1))$$

holds. For any π in which $Var(H)\theta_1\pi$ is a ground term, it follows that

$$H\theta_1\pi \leftrightarrow \exists_{\overline{b}}(B\theta_1\pi).$$

Since a renaming θ_2 changes only variables on $\overline{b}$ on one to one, hence

$$H\theta_1\pi \leftrightarrow \exists_{\overline{b'}}(B\theta_1\pi\theta_2),$$

where $\overline{b'} = \overline{b}\theta_2$. From $\exists_{\overline{b'}}$, for any π, there exists a ground substitution σ of variables on $\overline{b'}$, and $B\theta_1\pi\theta_2\sigma$ is a ground term, thus

$$(H\theta_1)\pi \leftrightarrow (B\theta_1)\pi\theta_2\sigma.$$

$Var(H) \cap \overline{b} = \emptyset$, $Var(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$, and $Var(H)\theta_1 \cap \overline{b} = \emptyset$ hold. Since $(B\theta_1)\pi\theta_2\sigma = (B(\theta_1 \cup \theta_2))\pi\sigma$,

$$(H\theta_1)\pi \leftrightarrow (B(\theta_1 \cup \theta_2))\pi\sigma$$

holds, and for any π, there exists σ. Therefore, it follows that

$$\forall(H\theta_1 \leftrightarrow \exists_{\overline{b'}}(B(\theta_1 \cup \theta_2))).$$

□

Proposition A.2. *A renaming ρ satisfies a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, a substitution δ satisfies a condition ⑥ of $R(\mathcal{X},\ \mathcal{Y})$, and γ is a substitution on $Var(H \cup \mathcal{X}\delta \cup B)$. If $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D, then $\mathcal{X}\delta\gamma \subseteq \mathcal{M}(D)$ holds and there exists a substitution ω such that $\mathcal{Y}(\delta \cup \rho)\gamma\omega \subseteq \mathcal{M}(D)$ holds.*

Proof: Assume that $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D.

$LE(\mathcal{X},\ \mathcal{Y})$ {Premise : Letting $\theta_1 = \delta$, $\theta_2 = \rho$, $\overline{b} = Var(\mathcal{Y}) - Var(\mathcal{X})$, and $H = \mathcal{X}$.
From the definition of $LE(\mathcal{X},\ \mathcal{Y})$, $Var(H) \cap \overline{b} = \emptyset$.
Since $Var(H) \subseteq \mathbb{V}r$ and $\overline{b}\theta_2 \subseteq \mathbb{V}d$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$.
Since $\overline{b} \subseteq \mathbb{V}r$ and $Var(H)\theta_1 \subseteq \mathbb{V}d$, $Var(H)\theta_1 \cap \overline{b} = \emptyset$.
From a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, $Var(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$.
From Proposition A.1, $LE(\mathcal{X}\delta,\ \mathcal{Y}(\delta \cup \rho))$ holds.}

$\rightarrow LE(Xs\delta,\ \mathcal{Y}(\delta \cup \rho))$
{Premise : Letting $\theta_1 = \gamma$, $\theta_2 = \{\ \}$, $\overline{b} = (Var(\mathcal{Y}) - Var(\mathcal{X}))\rho$, and $H = \mathcal{X}\delta$.
From $LE(\mathcal{X}\delta,\ \mathcal{Y}(\delta \cup \rho))$, $Var(H) \cap \overline{b} = \emptyset$.
Since $Var(H)\theta_1$ is a ground term, $Var(H)\theta_1 \cap \overline{b} = \emptyset$.
Since $\theta_2 = \{\ \}$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$ and $Var(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$.
From Proposition A.1, $LE(\mathcal{X}\delta\gamma,\ \mathcal{Y}(\delta \cup \rho)\gamma)$ holds.}

$\rightarrow LE(\mathcal{X}\delta\gamma,\ \mathcal{Y}(\delta \cup \rho)\gamma)$
{From $\mathcal{X}\delta\gamma \subseteq \mathcal{M}(D)$, $\mathcal{X}\delta\gamma \leftrightarrow true$ holds.}

$\leftrightarrow LE(true,\ \mathcal{Y}(\delta \cup \rho)\gamma)$
{Letting $\overline{y'} = (Var(\mathcal{Y}) - Var(\mathcal{X}))\rho$.}

$\leftrightarrow \exists_{\overline{y'}}\mathcal{Y}(\delta \cup \rho)\gamma$

Consequently, there exists a ground substitution ω of variables on $\overline{y'}$, and $\mathcal{Y}(\delta \cup \rho)\gamma\omega \subseteq \mathcal{M}(D)$ holds.

□

Proposition A.3. *A renaming ρ satisfies a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, a substitution δ satisfies a condition ⑥ of $R(\mathcal{X},\ \mathcal{Y})$, $\gamma 1$ is a substitution on $Var((H \cup B \cup \mathcal{Y}(\delta \cup \rho)) - \overline{y}\rho)$, and $\gamma 2$ is a substitution on $\overline{y}\rho$. If $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D, then $\mathcal{Y}(\delta \cup \rho)\gamma 1\gamma 2 \subseteq \mathcal{M}(D)$ holds and for any ground substitution ω, $\mathcal{X}\delta\gamma 1\omega \subseteq \mathcal{M}(D)$ holds.*

Proof: Assume that $LE(\mathcal{X},\ \mathcal{Y})$ is correct with respect to D.

$LE(\mathcal{X},\ \mathcal{Y})$ {Premise : Letting $\theta_1 = \delta$, $\theta_2 = \rho$, $\overline{b} = Var(\mathcal{Y}) - Var(\mathcal{X})$, and $H = \mathcal{X}$.
From the definition of $LE(\mathcal{X},\ \mathcal{Y})$, $Var(H) \cap \overline{b} = \emptyset$.
Since $Var(H) \subseteq \mathbb{V}r$ and $\overline{b}\theta_2 \subseteq \mathbb{V}d$, $Var(H) \cap \overline{b}\theta_2 = \emptyset$.
Since $\overline{b} \subseteq \mathbb{V}r$ and $Var(H)\theta_1 \subseteq \mathbb{V}d$, $Var(H)\theta_1 \cap \overline{b} = \emptyset$.

From a condition ⑤ of $R(\mathcal{X},\ \mathcal{Y})$, $\mathit{Var}(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$.
From Proposition A.1, $LE(\mathcal{X}\delta,\ \mathcal{Y}(\delta \cup \rho))$ holds.$\}$

$\rightarrow LE(\mathcal{X}\delta,\ \mathcal{Y}(\delta \cup \rho))$

$\{$Premise : Letting $\theta_1 = \gamma 1$, $\theta_2 = \{\ \}$, $\overline{b} = (\mathit{Var}(\mathcal{Y}) - \mathit{Var}(\mathcal{X}))\rho$, $H = \mathcal{X}\delta$, and $B = \mathcal{Y}(\delta \cup \rho)$.
From $LE(\mathcal{X}\delta,\ \mathcal{Y}(\delta \cup \rho))$, $\mathit{Var}(H) \cap \overline{b} = \emptyset$.
Since $\mathit{Var}(H)$ in $B\theta_1$ is a ground term and θ_1 changes only variables on $\mathit{Var}(H)$ in B, $\mathit{Var}(H)\theta_1 \cap \overline{b} = \emptyset$.
Since $\theta_2 = \{\ \}$, $\mathit{Var}(H) \cap \overline{b}\theta_2 = \emptyset$ and $\mathit{Var}(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$.
From Proposition A.1, $LE(\mathcal{X}\delta\gamma 1,\ \mathcal{Y}(\delta \cup \rho)\gamma 1)$ holds.$\}$

$\rightarrow LE(\mathcal{X}\delta\gamma 1,\ \mathcal{Y}(\delta \cup \rho)\gamma 1)$

$\{$Premise : Let ω be any substitution in which $\mathit{Var}(\mathcal{X})\delta\gamma 1\omega$ is a ground term.
Letting $\theta_1 = \omega$, $\theta_2 = \{\ \}$, $\overline{b} = \overline{y}\rho$, $H = \mathcal{X}\delta\gamma 1$, and $B = \mathcal{Y}(\delta \cup \rho)\gamma 1$.
From $LE(\mathcal{X}\delta\gamma 1,\ \mathcal{Y}(\delta \cup \rho)\gamma 1)$, $\mathit{Var}(H) \cap \overline{b} = \emptyset$.
Since $\mathit{Var}(H)\theta_1$ is a ground term, $\mathit{Var}(H)\theta_1 \cap \overline{b} = \emptyset$.
Since $\theta_2 = \{\ \}$, $\mathit{Var}(H) \cap \overline{b}\theta_2 = \emptyset$ and $\mathit{Var}(H)\theta_1 \cap \overline{b}\theta_2 = \emptyset$.
From Proposition A.1, $LE(\mathcal{X}\delta\gamma 1\omega,\ \mathcal{Y}(\delta \cup \rho)\gamma 1\omega)$ holds.
Since $\mathit{Var}(H)$ in B is a ground term, $B\theta_1 = B$.$\}$

$\rightarrow LE(\mathcal{X}\delta\gamma 1\omega,\ \mathcal{Y}(\delta \cup \rho)\gamma 1)$

$\{$There exists a ground substitution $\gamma 2$ of variables on $\overline{y}\rho$.$\}$

$\rightarrow LE(\mathcal{X}\delta\gamma 1\omega,\ \mathcal{Y}(\delta \cup \rho)\gamma 1\gamma 2)$

Consequently, since $\mathcal{Y}(\delta \cup \rho)\gamma 1\gamma 2 \subseteq \mathcal{M}(D)$ holds, $\mathcal{X}\delta\gamma 1\omega \subseteq \mathcal{M}(D)$ holds.

□