International Journal of Innovative Computing, Information and Control
Volume **10**, Number **6**, December **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 1999–2009

ET-BASED BIDIRECTIONAL SEARCH FOR PROVING FORMULAS IN THE CLASS $\mathcal{ES}$

Katsunori Miura[1] and Kiyoshi Akama[2]

[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

Received December 2013; revised April 2014

Abstract. *In this paper, we propose a new method for proving the correctness of Logical Equivalences (LEs) in a newly defined class called $\mathcal{ES}$. Proving the correctness of LEs is important for guaranteeing the correctness of Equivalent Transformation (ET) rules made from LEs. ET rules are useful for constructing correct sequential and parallel programs. The proposed method proves the equivalence of the declarative meaning of two definite clause sets using a bidirectional search that starts from two different points simultaneously. In the proposed method, the two definite clause sets at the starting points are transformed by ET rules. The method can generate ET rules that are essential for advancing the proof along with the general unfolding rules that are given before computing the proof. Class $\mathcal{ES}$ is a superclass of the classes proposed in past studies and therefore covers the LEs they contain.*

Keywords: Logical equivalence, Bidirectional search, Correctness, ET rule, Induction

1. Introduction.

In Equivalent Transformation (ET) programming [2], a program is a set of ET rules, each of which replaces one definite clause set with another while preserving the declarative meaning of the original clause set. Methodologies for automatic construction of sequential and parallel programs by ET programming have already been proposed [1, 3], and an e-learning system for programming education has been developed using ET rules [10].

ET rules can be made from logical formulas, and each of which is called a Logical Equivalence (LE) [15] and describes an equivalence relationship between two conjunctions of existentially quantified atoms under some specified preconditions. Miura et al. [13] showed that various ET rules can be made based on LEs, while researchers such as Chazarain and Muller [4], Flener [5], and Yoshida et al. [17] have proposed methods for constructing programs from logical formulas and have demonstrated the efficacy of logical formulas.

Methods that can prove the correctness of LEs are essential for guaranteeing the correctness of ET rules. Consequently, Miura et al. [12, 14] have proposed methods for proving the correctness of LEs in specified classes. In this paper, we present an expanded class that covers the provable range presented in past studies, and propose a new method of proving the correctness of LEs in this new class. The class treated in this paper is called $\mathcal{ES}$.

The new method proposed in this paper guarantees the correctness of LEs by proving equivalence of the declarative meaning between two definite clause sets using a bidirectional search. The bidirectional search starts from two different points simultaneously and terminates with success if a meeting point is found. In the new method, the starting points are two definite clause sets determined from an LE. To find the meeting point, the two definite clause sets are successively transformed by ET rules. The ET rules perform clause transformations until both clause sets are the same.

The proposed method gives general unfolding rules that are basic ET rules, before computing the proof. If ET rules other than the given unfolding rules are needed for the proof, new ET rules can be generated and used for transformation. To generate new ET rules, the correctness of other LEs is proven by recursively calling the proposed method, and new ET rules are made from the correct LEs thus proven.

There are complex LEs in which the same clause set cannot be found by simple clause transformations by ET rules. We prove the correctness of several complex LEs by introducing induction into the proof method. Proof of correctness is computed by dividing the proof into a base case and an inductive step. In the inductive step, a special ET rule, called an induction rule, is used for the clause transformations.

The remainder of this paper is organized as follows. Section 2 formulates the new class $\mathcal{ES}$ of LEs and defines the correctness of the LEs in $\mathcal{ES}$. Examples of LEs in this class are also presented. Section 3 defines ET rules and presents examples of ET rules. Section 4 outlines the new method of proving the correctness of LEs in $\mathcal{ES}$. In Section 5, we give a concrete example of LEs, and prove the correctness of a given LE using the proposed method. Section 6 explains the nondeterministic processing executed in the proposed method. Section 7 discusses the usefulness of the proposed method and compares it with other approaches.

2. Logical Equivalences and Correctness.

2.1. Logical equivalences.

The LEs considered in this paper are of the form

$$\forall_{\overline{v}}(\exists_{\overline{v}_1}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v}_2}\mathcal{E}_2),$$

where $\mathcal{E}_1$ and $\mathcal{E}_2$ are conjunctions of atoms, $\overline{v}_1$ and $\overline{v}_2$ are, respectively, sets of variables in $\mathcal{E}_1$ and $\mathcal{E}_2$, and $\overline{v}$ is the set of all variables in $\mathcal{E}_1$ and $\mathcal{E}_2$ that does not include variables in $\overline{v}_1$ and $\overline{v}_2$. This class of LEs is called $\mathcal{ES}$. When no confusion is possible, $\overline{v}$ is often omitted. We assume that $\mathcal{E}_1$ and $\mathcal{E}_2$ are written as sets of atoms.

For example,

(1) $\forall_{\{X\}}(\exists_{\{Y\}}\{rev(X,Y),\ rev(Y,X)\} \leftrightarrow \{list(X)\})$

(2) $\forall_{\{Y,Z,V,W\}}(\exists_{\{X\}}\{app(X,[Y],Z),\ app(V,[W],X)\} \leftrightarrow \{app(V,[W,Y],Z)\})$

(3) $\forall_{\{X,Y,W\}}(\exists_{\{Z\}}\{app(X,Y,Z),\ rev(Z,W)\} \leftrightarrow \exists_{\{P,Q\}}\{rev(X,P),\ rev(Y,Q),\ app(Q,P,W)\})$

are LEs in $\mathcal{ES}$, where $rev(x,\ y)$ means that lists x and y are in reverse order, $app(x,\ y,\ z)$ means that the concatenation of lists x and y is a list z, and $list(x)$ means that x is a list.

2.2. Correctness of LEs.

2.2.1. *Declarative meaning.*

Given a set $\mathcal{D}$ of definite clauses, the declarative meaning of $\mathcal{D}$, denoted by $\mathcal{M}(\mathcal{D})$, is defined as follows.

Definition 2.1. *Let $\mathbb{S}$ be the set of all substitutions. Let $\mathcal{G}$ be all ground atoms. Given a set $\mathcal{D}$ of definite clauses, $T_{\mathcal{D}}$ is defined by*

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

for any subset $\mathbb{G}$ *of* $\mathcal{G}$. $\mathcal{M}(\mathcal{D})$ *is defined by*

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

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

2.2.2. *Correctness of LEs.* The correctness of LEs in $\mathcal{ES}$ is dependent on background knowledge $\mathbb{D}$, which is a set of definite clauses. An LE $\forall_{\overline{v}}(\exists_{\overline{v}_1}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v}_2}\mathcal{E}_2)$ is correct with respect to $\mathbb{D}$ iff $\mathcal{M}(\mathbb{D})$ is a model of $\forall_{\overline{v}}(\exists_{\overline{v}_1}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v}_2}\mathcal{E}_2)$.

For example, let $\mathbb{D}$ be $\{cl_1,\ cl_2,\ cl_3,\ cl_4,\ cl_5,\ cl_6\}$, where

$$\begin{aligned}
&cl_1 : app([\ ],\ X,\ X) \leftarrow \\
&cl_2 : app([A \mid X],\ Y,\ [A \mid Z]) \leftarrow app(X,\ Y,\ Z) \\
&cl_3 : rev([\ ],\ [\ ]) \leftarrow \\
&cl_4 : rev([A \mid X],\ Z) \leftarrow app(Y,\ [A],\ Z),\ rev(X,\ Y) \\
&cl_5 : list([\ ]) \leftarrow \\
&cl_6 : list([A \mid B]) \leftarrow list(B).
\end{aligned}$$

Then, an LE

$$\forall(\exists_{\{Y\}}\{rev(X,\ Y),\ rev(Y,\ X)\} \leftrightarrow \{list(X)\})$$

is correct with respect to $\mathbb{D}$ since, for any ground term X, $\exists_{\{Y\}}\{rev(X,\ Y),\ rev(Y,\ X)\}$, and $\{list(X)\}$ have the same truth value.

3. **ET Rules.** An ET rule replaces a definite clause set with another while preserving its declarative meaning with respect to background knowledge $\mathbb{D}$, where the predicate appearing in the head of the definite clause set exists neither in $\mathbb{D}$ nor in the body of the clause set. 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 definite clause sets $cls1$ and $cls2$ such that $cls1$ is transformed to $cls2$ by r.

For example,

$$\begin{aligned}
&r_1 : app(A,\ B,\ C) \Rightarrow \{A = [\ ],\ B = C\}; \\
&\qquad\qquad\qquad\quad \Rightarrow \{A = [D \mid E],\ C = [D \mid F]\},\ app(E,\ B,\ F). \\
&r_2 : rev(A,\ B) \Rightarrow \{A = [\ ],\ B = [\ ]\}; \\
&\qquad\qquad\quad \Rightarrow \{A = [C \mid D]\},\ rev(D,\ E),\ app(E,\ [C],\ B). \\
&r_3 : list(A) \Rightarrow \{A = [\ ]\}; \\
&\qquad\qquad \Rightarrow \{A = [B \mid C]\},\ list(C).
\end{aligned}$$

are general unfolding rules for the *app*, *rev*, and *list* predicates. Rule r_1 is applicable to a clause set

$$cl_A : \{ans(X) \leftarrow app([1,\ 2,\ 3],\ [4],\ X)\}.$$

By applying r_1 to cl_A, cl_A is transformed into

$$cl_B : \{ans([1 \mid Y]) \leftarrow app([2,\ 3],\ [4],\ Y)\}.$$

4. **Proving LEs in Class $\mathcal{ES}$.** This paper proposes a new method for proving the correctness of LEs in Class $\mathcal{ES}$. The method has the following features:

1. Computation by ET rules.
2. Bidirectional search (Section 4.1)
3. Use of induction (Section 4.2)
4. Generation of ET rules (Section 4.3)

4.1. **Bidirectional search.** A bidirectional search starts from two different points simultaneously and terminates with success when a meeting point is found. In the proposed method, the starting points are two definite clause sets, which are subsequently transformed by ET rules. Given $\forall_{\overline{v}}(\exists_{\overline{v}_1}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v}_2}\mathcal{E}_2)$, the proposed method of proving it is outlined as follows:

1. Make a pair $(\mathbb{S}_L,\ \mathbb{S}_R)$ of two singleton sets of a definite clause from the LE:
   $\mathbb{S}_L = \{\mathcal{P} \leftarrow \mathcal{E}_1\}$
   $\mathbb{S}_R = \{\mathcal{P} \leftarrow \mathcal{E}_2\}$,
   where $\mathcal{P}$ is an atom with a new predicate (*ans* is used in this paper) with variables in $\overline{v}$ as arguments.
2. Make a new pair $(\mathbb{S}_L,\ \mathbb{S}_R)$ from the current pair by transforming one of $\mathbb{S}_L$ and $\mathbb{S}_R$ by applying an ET rule.
3. If $\mathbb{S}_L = \mathbb{S}_R$, terminate with success, otherwise go to Step 2.

In Step 2, there are two nondeterministic selections for ① one clause from $\{\mathbb{S}_L,\ \mathbb{S}_R\}$, and ② one ET rule.

4.2. **Induction and induction rules.** Use of induction is often required to prove the correctness of difficult LEs, where the proof of LEs is computed by dividing the proof into a base case and an inductive step. In this paper, recursive variables are selected from among variables on $\mathcal{P}$ in $\mathbb{S}_L$ and $\mathbb{S}_R$. Recursive variables are selected nondeterministically.

An induction rule is made from an LE, and is used in the inductive step. Induction rules are applicable to atoms that contain recursive variables.

For example, an LE $\forall(\exists_{\{Y\}}\{rev(X,\ Y),\ rev(Y,\ Y)\} \leftrightarrow \{list(X)\})$ is proven using induction, where variable X is selected as a recursive variable. A substitution $\{X/[V \mid P^{\sim k}]\}$ is applied to the LE. Tag k shows that variable X is a recursive variable. An induction rule

$$rev(P^{\sim k},\ Y),\ rev(Y,\ P^{\sim k}),\ \{isolated(Y)\} \Rightarrow list(P^{\sim k})$$

is made from the LE, where an atom with the predicate *isolated* is made of an existentially quantified variable Y in the left-hand side of the LE.

4.3. **Generation of ET rules.** It is difficult to prepare all ET rules that are necessary for the proof of LEs prior to the start of proof computation. The proposed method allows ET rules to be generated while the LEs are being proven. We believe that generation of ET rules is useful for bringing $\mathbb{S}_L$ and $\mathbb{S}_R$ into closer form.

For example, consider the transformation of the following set:

$$\mathbb{S}_L = \{ans(A,\ B,\ C) \leftarrow list(C),\ app(A,\ [B],\ C)\}$$

The following rule is useful because it removes the *list* atom from the body:

$$list(C),\ app(A,\ [B],\ C),\ \{isolated(C)\} \Rightarrow app(A,\ [B],\ C).$$

It is preferable to generate this rule in the course of the proof because it is difficult to predict when $app(A,\ B,\ C)$ and $list(C)$ will appear in the body, before transforming $\mathbb{S}_L$.

Table 1. Clause transformations for the proof of *rev-last* problem

Transformations of $\mathbb{S}_{rl1}^{B1}$	Transformations of $\mathbb{S}_{rl2}^{B1}$
s_1 : $\{ans([\,], A) \leftarrow \underline{rev([\,], [A \mid B])}\}$	s_3 : $\{ans([\,], A) \leftarrow \underline{last([\,], A)}\}$
$\Downarrow$ Apply r_1	$\Downarrow$ Apply r_8
s_2 : $\{\}$	s_4 : $\{\}$

Transformations of $\mathbb{S}_{rl1}^{B2}$	Transformations of $\mathbb{S}_{rl2}^{B2}$
s_5 : $\{ans([A], B) \leftarrow \underline{rev([A], [B \mid C])}\}$	s_9 : $\{ans([A], B) \leftarrow \underline{last([A], B)}\}$
$\Downarrow$ Apply r_1	$\Downarrow$ Apply r_8
s_6 : $\{ans([A], B) \leftarrow \underline{rev([\,], D)}, app(D, [A], [B \mid C])\}$	s_{10} : $\{ans([A], A) \leftarrow\}$
$\Downarrow$ Apply r_1	
s_7 : $\{ans([A], B) \leftarrow \underline{app([\,], [A], [B \mid C])}\}$	
$\Downarrow$ Apply r_2	
s_8 : $\{ans([A], A) \leftarrow\}$	

Transformations of $\mathbb{S}_{rl1}^{R}$	Transformations of $\mathbb{S}_{rl2}^{R}$
s_{11} : $\{ans([A, B \mid C^{\sim k}], D) \leftarrow$	s_{18} : $\{ans([A, B \mid C^{\sim k}], D) \leftarrow$
$\underline{rev([A, B \mid C^{\sim k}], [D \mid E])}\}$	$\underline{last([A, B \mid C^{\sim k}], D)}\}$
$\Downarrow$ Apply $r_1 \rightarrow r_1 \rightarrow r_2 \rightarrow r_2 \rightarrow r_2 \rightarrow r_1 \rightarrow r_2 \rightarrow r_2$	$\Downarrow$ Apply r_8
s_{12} : $\{ans([A, B], B) \leftarrow$	s_{19} : $\{ans([A, B \mid C^{\sim k}], D) \leftarrow \underline{last([B \mid C^{\sim k}], D)}\}$
$ans([C, D \mid E^{\sim k}], F)$	$\Downarrow$ Apply r_8
$\leftarrow \underline{app(H, [C], I)}, app(G, [D], H), rev(E^{\sim k}, [F \mid G])\}$	s_{20} : $\{ans([A, B], B) \leftarrow$
$\Downarrow$ Apply r_3	$ans([C, D \mid E^{\sim k}], F) \leftarrow last(E^{\sim k}, F)\}$
s_{13} : $\{ans([A, B], B) \leftarrow$	
$ans([C, D \mid E^{\sim k}], F)$	
$\leftarrow \underline{list(H), app(G, [D], H)}, rev(E^{\sim k}, [F \mid G])\}$	
$\Downarrow$ Apply r_4	
s_{14} : $\{ans([A, B], B) \leftarrow$	
$ans([C, D \mid E^{\sim k}], F) \leftarrow rev(E^{\sim k}, [F \mid G]), \underline{app(G, [D], H)}\}$	
$\Downarrow$ Apply r_3	
s_{15} : $\{ans([A, B], B) \leftarrow$	
$ans([C, D \mid E^{\sim k}], F) \leftarrow \underline{rev(E^{\sim k}, [F \mid G])}, list(G)\}$	
$\Downarrow$ Apply $r_7 \rightarrow r_1 \rightarrow r_7 \rightarrow r_4 \rightarrow r_7 \rightarrow r_6 \rightarrow r_7$	
s_{16} : $\{ans([A, B], B) \leftarrow$	
$ans([C, D \mid E^{\sim k}], F) \leftarrow \underline{rev(E^{\sim k}, [F \mid H])}\}$	
$\Downarrow$ Apply $\mathcal{R}_i$	
s_{17} : $\{ans([A, B], B) \leftarrow$	
$ans([C, D \mid E^{\sim k}], F) \leftarrow last(E^{\sim k}, F)\}$	

5. **Example of Proof of LEs Using the Proposed Method.**

5.1. **Replacement of problem settings for the proof.**

In this section, we prove the correctness of the following LE using the proposed method:

$$\mathbb{E}_{rl} : \forall(\exists_{\{Z\}}\{rev(X,\ [Y \mid Z])\} \leftrightarrow \{last(X,\ Y)\})$$

In this paper, this proof problem is called the *rev-last* problem. Definite clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$ are made from $\mathbb{E}_{rl}$. The body of set $\mathbb{S}_{rl1}$ is composed of atoms in the left-hand side in $\mathbb{E}_{rl}$, while set $\mathbb{S}_{rl2}$ is composed of atoms in the right-hand side. The variables in the *ans* atom comprise universally quantified variables in $\mathbb{E}_{rl}$.

$$\mathbb{S}_{rl1} = \{ans(X,\ Y) \leftarrow rev(X,\ [Y \mid Z])\}$$
$$\mathbb{S}_{rl2} = \{ans(X,\ Y) \leftarrow last(X,\ Y)\}$$

The proposed method uses induction to prove equivalence of the declarative meaning of two clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$. To prove by induction, two tasks are executed in preparation:

1. Select some variables in the *ans* atom.
2. Determine the substitutions to apply to the variables.

Some clause sets are obtained by applying the substitutions to clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$. Clause sets $\mathbb{S}_{rl1}^{B1}$ and $\mathbb{S}_{rl2}^{B1}$ are obtained by applying an empty list [ ] to variable X. Clause sets $\mathbb{S}_{rl1}^{B2}$ and $\mathbb{S}_{rl2}^{B2}$ are obtained by applying singleton list $[A]$ to variable X. Clause sets $\mathbb{S}_{rl1}^{R}$ and $\mathbb{S}_{rl2}^{R}$ are obtained by applying a list $[A,\ B \mid C^{\sim k}]$ to variable X. The proof for $\mathbb{S}_{rl1} = \mathbb{S}_{rl2}$ is replaced with the three proofs for $\mathbb{S}_{rl1}^{B1} = \mathbb{S}_{rl2}^{B1}$, $\mathbb{S}_{rl1}^{B2} = \mathbb{S}_{rl2}^{B2}$, and $\mathbb{S}_{rr1}^{R} = \mathbb{S}_{rr2}^{R}$. In Section 6.1, we present methods for selecting variables in the *ans* atom and determine substitutions for induction.

$$\mathbb{S}_{rl1}^{B1} = \{ans([\,],\ Y) \leftarrow rev([\,],\ [Y \mid Z])\}$$
$$\mathbb{S}_{rl2}^{B1} = \{ans([\,],\ Y) \leftarrow last([\,],\ Y)\}$$

$$\mathbb{S}_{rl1}^{B2} = \{ans([A],\ Y) \leftarrow rev([A],\ [Y \mid Z])\}$$
$$\mathbb{S}_{rl2}^{B2} = \{ans([A],\ Y) \leftarrow last([A],\ Y)\}$$

$$\mathbb{S}_{rl1}^{R} = \{ans([A,\ B \mid C^{\sim k}],\ Y) \leftarrow rev([A,\ B \mid C^{\sim k}],\ [Y \mid Z])\}$$
$$\mathbb{S}_{rl2}^{R} = \{ans([A,\ B \mid C^{\sim k}],\ Y) \leftarrow last([A,\ B \mid C^{\sim k}],\ Y)\}$$

5.2. **Transformations of clause sets by ET rules.** This section presents the proof of three problems: ① the equivalence of clause sets $\mathbb{S}_{rl1}^{B1}$ and $\mathbb{S}_{rl2}^{B1}$, ② the equivalence of clause sets $\mathbb{S}_{rl1}^{B2}$ and $\mathbb{S}_{rl2}^{B2}$, and ③ the equivalence of clause sets $\mathbb{S}_{rl1}^{R}$ and $\mathbb{S}_{rl2}^{R}$. The proposed method first proves problems ① and ②, then problem ③. Problems ① and ② are for the base case, while problem ③ is for the inductive step. The proof of problem ③ uses induction; i.e., problem ③ uses an induction rule. As indicated by the shaded area in Table 1, the same clause set is obtained from the clause sets of each problem, respectively. The *rev-last* problem uses the set $\mathbb{R}_{rl}$ of ET rules and an induction rule $\mathcal{R}i$:

$$\mathbb{R}_{rl} = \begin{cases} r_1 : rev(X,\ Y) \Rightarrow \{X = [\,],\ Y = [\,]\}; \\ \qquad\qquad\qquad \Rightarrow \{X = [A \mid B]\},\ rev(B,\ C),\ app(C,\ [A],\ Y). \\ r_2 : app(X,\ Y,\ Z) \Rightarrow \{X = [\,],\ Y = Z\}; \\ \qquad\qquad\qquad \Rightarrow \{X = [A \mid B],\ Z = [A \mid C]\},\ app(B,\ Y,\ C). \\ r_3 : app(X,\ [Y],\ Z),\ \{isolated(Z)\} \Rightarrow list(A). \\ r_4 : app(X,\ [Y],\ Z),\ list(Z) \Rightarrow app(X,\ [Y],\ Z). \\ r_5 : rev(X,\ Y),\ list(Y) \Rightarrow rev(X,\ Y). \\ r_6 : app(V,\ [X],\ Z),\ rev(Y,\ V),\ \{isolated(V)\} \Rightarrow rev([X \mid Y],\ Z). \\ r_7 : rev(X,\ Y) \Rightarrow rev(Y,\ X). \\ r_8 : last(X,\ Y) \Rightarrow \{X = [A],\ Y = A\}; \\ \qquad\qquad\qquad \Rightarrow \{X = [A,\ B \mid C]\},\ last([B \mid C],\ Y). \end{cases}$$

The *isolated* atom in rule r_3 means that variable Z applies to a variable only and the variable appears only in the third argument of the *app* atom, which is a body atom of a clause set to which is applied rule r_3. The *isolated* atom in rule r_6 means that variable V applies to a variable only and the variable appears only in the arguments written to be V with the head atoms of rule r_6 of the *app* and *list* atoms to which the rule is applied. An induction rule $\mathcal{R}i$ is needed in the transformations of clause set $\mathbb{S}_{rl1}^{R}$, and so rule $\mathcal{R}i$ is made from the LE $\mathbb{E}_{rl}$. The number of elements in list $C^{\sim k}$ is smaller than that in list $[A,\ B \mid C^{\sim k}]$ by two elements. Rule $\mathcal{R}i$ is added to a set $\mathbb{R}_{rl}$ of rules:

$$\mathcal{R}i : rev(C^{\sim k}, [Y \mid Z]),\ \{isolated(Z)\} \Rightarrow last(C^{\sim k},\ Y).$$

Table 2. Definite clause sets for base case and inductive step

Clause set $\mathbb{S}_{rl1}$	Substitution	Clause sets made by applying Substitution
$\{ans(X,Y) \leftarrow rev(X,[Y \mid Z])\}$	$\times$ $\mathbb{T}^B_{rl} = \{\{X/[\,]\}\}$	$\Rightarrow$ $\{ans([\,],Y) \leftarrow rev([\,],[Y \mid Z])\}$
$\{ans(X,Y) \leftarrow rev(X,[Y \mid Z])\}$	$\times$ $\mathbb{T}^R_{rl} = \{X/[A \mid B^{\sim k}]\}$	$\Rightarrow$ $\{ans([A \mid B^{\sim k}],Y) \leftarrow rev([A \mid B^{\sim k}],[Y \mid Z])\}$
$\{ans(X,Y) \leftarrow rev(X,[Y \mid Z])\}$	$\times$ $\mathbb{T}^B_{rl} = \{\{X/[\,]\},\{X/[A]\}\}$	$\Rightarrow$ $\{ans([\,],Y) \leftarrow rev([\,],[Y \mid Z])\}$ $\{ans([A],Y) \leftarrow rev([A],[Y \mid Z])\}$
$\{ans(X,Y) \leftarrow rev(X,[Y \mid Z])\}$	$\times$ $\mathbb{T}^R_{rl} = \{X/[A,B \mid C^{\sim k}]\}$	$\Rightarrow$ $\{ans([A,B \mid C^{\sim k}],Y)\}$ $\leftarrow rev([A,B \mid C^{\sim k}],[Y \mid Z])\}\}$

Clause set $\mathbb{S}_{rl2}$	Substitution	Clause sets made by applying Substitution
$\{ans(X,Y) \leftarrow last(X,Y)\}$	$\times$ $\mathbb{T}^B_{rl} = \{\{X/[\,]\}\}$	$\Rightarrow$ $\{ans([\,],Y) \leftarrow last([\,],Y)\}$
$\{ans(X,Y) \leftarrow last(X,Y)\}$	$\times$ $\mathbb{T}^R_{rl} = \{X/[A \mid B^{\sim k}]\}$	$\Rightarrow$ $\{ans([A \mid B^{\sim k}],Y) \leftarrow last([A \mid B^{\sim k}],Y)\}$
$\{ans(X,Y) \leftarrow last(X,Y)\}$	$\times$ $\mathbb{T}^B_{rl} = \{\{X/[\,]\},\{X/[A]\}\}$	$\Rightarrow$ $\{ans([\,],Y) \leftarrow last([\,],Y)\}$ $\{ans([A],Y) \leftarrow last([A],Y)\}\}$
$\{ans(X,Y) \leftarrow last(X,Y)\}$	$\times$ $\mathbb{T}^R_{rl} = \{X/[A,B \mid C^{\sim k}]\}$	$\Rightarrow$ $\{ans([A,B \mid C^{\sim k}],Y) \leftarrow last([A,B \mid C^{\sim k}],Y)\}$

The *isolated* atom in rule $\mathcal{R}i$ means that variable Z applies to a variable only and the variable appears only in the arguments written to be Z with the head atoms of rule $\mathcal{R}i$ of the *rev* atom to which the rule is applied.

In the computation by ET rules, if two or more applicable rules exist in set $\mathbb{R}_{rl}$, then a clause set can be transformed by any rule selected from among the applicable rules. The sequence of transformations of clause sets shown in Table 1 is one of various sequences for outputting answers. In Section 6.2, we discuss methods for determining atoms that apply to a rule and selecting a rule from among applicable rules. If ET rules for transforming clause sets do not exist in set $\mathbb{R}_{rl}$, then new ET rules have to be made. In Section 6.3, we explain how new ET rules that are applicable to atoms are made.

6. Nondeterministic Processing to be Executed in the Proof.

6.1. Determination of base case and inductive step.

From the meaning of the LE $\mathbb{E}_{rl}$ (see Section 5.1) and the definition of the *rev* predicate, it is clear that variable X applies to any list with zero or more elements. In this paper, the number of elements in a list is viewed as the length of that list. Even when lists of any length are substituted with variable X of definite clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$, a meeting point is always found from the two definite clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$. Since lists can have infinite lengths, it is difficult to prove the length of all lists one by one. This paper uses induction to prove that the same clause set is obtained from definite clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$ with respect to lists of any length. To use induction, it is necessary that the base case and inductive step are determined. The base case and inductive step are determined by applying a list to variable X of definite clause sets $\mathbb{S}_{rl1}$ and $\mathbb{S}_{rl2}$. The list for the base case is a fixed-length list (such as $[\,]$, $[A]$, and $[A,\ B]$), while the list for the inductive step is a list without a fixed length (such as $[A \mid B]$ and $[A,\ B \mid C]$). The minimal length of the list in the inductive step is longer than that of the base case by one element. For example, if the list of the base case is an empty list only, then the list of the inductive step is a list

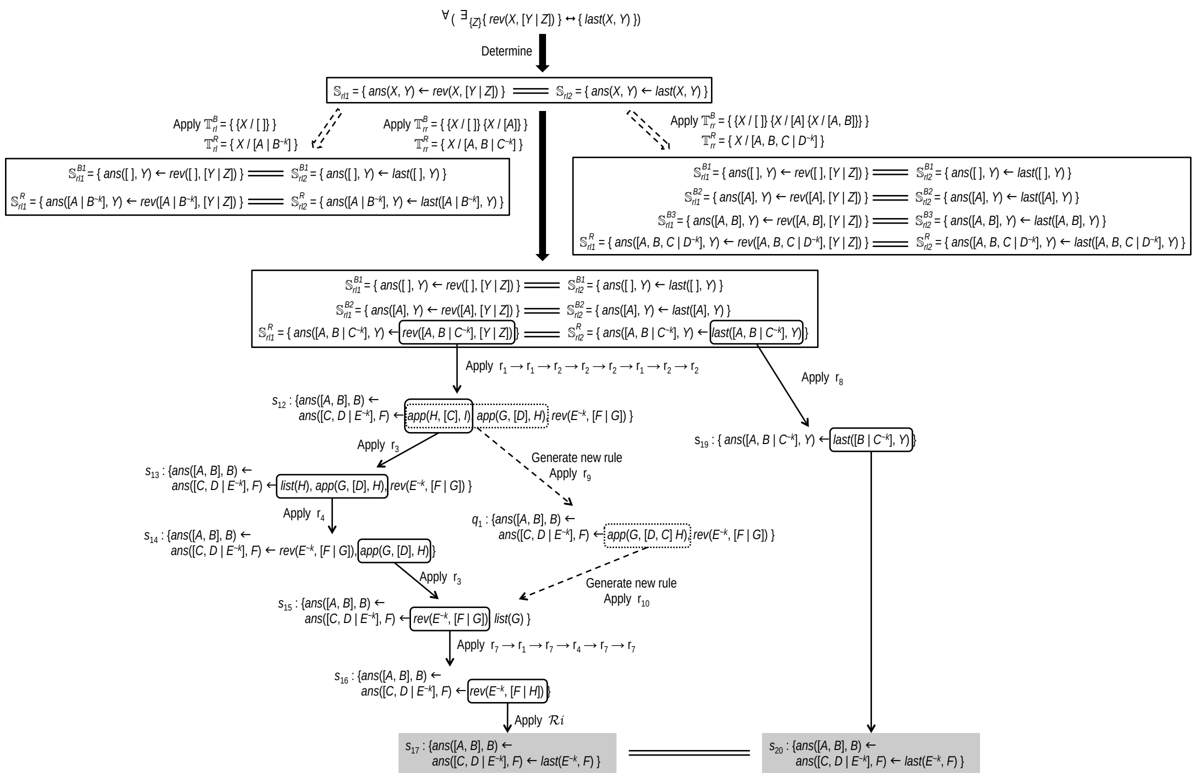
∀(∃{Z}{ rev(X, [Y | Z]) } ↔ { last(X, Y) })
Determine
S_rl1 = { ans(X, Y) ← rev(X, [Y | Z]) }
S_rl2 = { ans(X, Y) ← last(X, Y) }
Apply T^B_rl = { {X / [ ]} } T^R_rl = { X / [A | B^~k] }
Apply T^B_rl = { {X / [ ]} {X / [A]} } T^R_rl = { X / [A, B | C^~k] }
Apply T^B_rl = { {X / [ ]} {X / [A]} {X / [A, B]} } T^R_rl = { X / [A, B, C | D^~k] }
Apply r1 → r1 → r2 → r2 → r2 → r1 → r2 → r2
Apply r8
Apply r3
Generate new rule Apply r9
Apply r4
Apply r3
Generate new rule Apply r10
Apply r7 → r1 → r7 → r4 → r7 → r7
Apply Ri
s17 : {ans([A, B], B) ← ans([C, D | E^~k], F) ← last(E^~k, F) }
s20 : {ans([A, B], B) ← ans([C, D | E^~k], F) ← last(E^~k, F) }

FIGURE 1. Transformations of definite clause sets $\mathbb{S}^R_{rl1}$ and $\mathbb{S}^R_{rl2}$

comprising one or more elements. The proposed method gives substitutions for applying lists to variables of definite clause sets. Let $\mathbb{T}^B$ be a substitution for the base case, and let $\mathbb{T}^R$ be a substitution for the inductive step. Substitutions $\mathbb{T}^B_{rl}$ and $\mathbb{T}^R_{rl}$, shown by the shaded area in Table 2, signify that the base case is an empty list and a list comprising one element, while the inductive step is a list comprising two or more elements. The processing for induction is carried out as follows:

1. Determine a set $\mathbb{V}_{rl}$ of variables selected from among the variables of an *ans* atom.
2. Determine sets $\mathbb{T}^B_{rl}$ and $\mathbb{T}^R_{rl}$ of substitutions for applying lists to variables in $\mathbb{V}_{rl}$.

If two or more variables appear in an *ans* atom, then there are various patterns for selection of variables. In this example, a set $\mathbb{V}_{rl}$ is selected from among the following patterns:

$$\{X\}, \quad \{Y\}, \quad \{X,\ Y\}$$

It is preferable to at first select from among patterns with few variables. There are patterns of various combinations of sets $\mathbb{T}^B_{rl}$ and $\mathbb{T}^R_{rl}$, as exemplified below:

$$\begin{array}{lll}
\text{Pattern 1:} & \mathbb{T}^B_{rl} = \{\{X/[\ ]\}\}, & \mathbb{T}^R_{rl} = \{X/[A \mid B^{\sim k}]\} \\
\text{Pattern 2:} & \mathbb{T}^B_{rl} = \{\{X/[\ ]\},\ \{X/[A]\}\}, & \mathbb{T}^R_{rl} = \{X/[A,\ B \mid C^{\sim k}]\} \\
\text{Pattern 3:} & \mathbb{T}^B_{rl} = \{\{X/[\ ]\},\ \{X/[A]\},\ \{X/[A,\ B]\}\}, & \mathbb{T}^R_{rl} = \{X/[A,\ B,\ C \mid D^{\sim k}]\} \\
 & \vdots & \vdots \\
\text{Pattern N:} & \mathbb{T}^B_{rl} = \{\{X/[\ ]\},\ \{X/[A]\},\ \{X/[A,\ B]\},\cdots\}, & \mathbb{T}^R_{rl} = \{X/[A,\ B,\ C,\ \cdots \mid Z^{\sim k}]\}
\end{array}$$

If the number of elements in $\mathbb{T}^B_{rl}$ increases, the number of relationships of definite clause sets that need the proof correspondingly increases (refer to Figure 1).

6.2. **Selection of atom set and application of ET rules.** The sequence of transformations of definite clause set $\mathbb{S}^R_{rl1}$ shown in Table 1 is one of various sequences (refer to Figure 1). The correctness of the proof is guaranteed no matter the path followed in transformation of the definite clause sets. Various sequences exist because nondeterministic processing is included in the following two procedures for transforming definite clause sets:

1. Select atoms from among the body atoms of a definite clause.
2. Select an ET rule applicable to the atoms.

The following state exists in between states s_{11} and s_{12} in Table 1.

$$\begin{aligned} q_2 : \{&ans([A,\ B \mid C^{\sim k}],\ B) \leftarrow \underline{rev(C^{\sim k},\ [\,])},\ app([\,],\ [A],\ D) \\ &ans([E,\ F, \mid G^{\sim k}],\ H) \leftarrow \overline{rev(G^{\sim k},\ [H \mid I])},\ app(J,\ [E],\ K),\ app(I,\ [F],\ J)\} \end{aligned}$$

In the transformations in Table 1, rule r_1 in set $\mathbb{R}_{rl}$ is applied to the atom $rev(C^{\sim k},\ [\,])$. As a result, the following clause set is obtained from the above set by the application of the rule.

$$\begin{aligned} \{&ans([A,\ B],\ B) \leftarrow app([\,],\ [A],\ C) \\ &ans([D,\ E,\ H \mid I],\ E) \leftarrow app([\,],\ [D],\ G),\ rev(I,\ J),\ app(J,\ [H],\ [\,]) \\ &ans([K,\ L \mid M^{\sim k}],\ N) \leftarrow rev(M^{\sim k},\ [N \mid O]),\ app(P,\ [K],\ Q),\ app(O,\ [L],\ P)\} \end{aligned}$$

In state q_2, $rev(C^{\sim k},\ [\,])$ is selected, but it is possible to select $app([\,],\ [A],\ D)$, $rev(G^{\sim k},\ [H \mid I])$, or $app(J,\ [E],\ K)$. The atom $app([\,],\ [A],\ D)$ applies a rule r_2, $rev(G^{\sim k},\ [H \mid I])$ applies a rule r_7, and $app(J,\ [E],\ K)$ applies a rule r_3. If two or more ET rules applicable to atoms of a definite clause set exist, then the definite clause set can be transformed by any rule nondeterministically selected from among them.

6.3. **Generation of ET rules applicable to definite clause set.** We can make new ET rules that are useful for bringing two definite clause sets into a closer form. For example, state s_{12} shown in Table 1 applies rule r_3. We can attempt to make rules like rule r_3 or the following rule:

$$r_9 : app(X,\ [Y],\ Z),\ app(P,\ [V],\ X),\ \{isolated(X)\} \Rightarrow app(P,\ [V,\ Y],\ Z).$$

Rule r_9 describes a procedure for folding app atoms without using variable X, and it can be made from the following LE:

$$\forall(\exists_{\{X\}}\{app(X,\ [Y],\ Z),\ app(P,\ [V],\ X)\} \leftrightarrow \{app(P,\ [V,\ Y],\ Z)\})$$

The proposed method can prove the correctness of the above LE; thus, rule r_9 can be made by a generator that incorporates the proposed method.

Let us now look at the transformation of a definite clause set with state s_{12} by rule r_9.

$$\begin{aligned} s_{12} : \{&ans([A,\ B],\ B) \leftarrow \\ &ans([C,\ D \mid E^{\sim k}],\ F) \leftarrow \underline{app(H,\ [C],\ I),\ app(G,\ [D],\ H)},\ rev(E^{\sim k},\ [F \mid G])\} \end{aligned}$$

Rule r_9 is applied to two app atoms in the clause set; as a result, the clause set with state s_{12} is transformed into the following set:

$$\begin{aligned} q_1 : \{&ans([A,\ B],\ B) \leftarrow \\ &ans([C,\ D \mid E^{\sim k}],\ F) \leftarrow app(G,\ [D,\ C],\ H),\ rev(E^{\sim k},\ [F \mid G])\} \end{aligned}$$

By applying the following rule to state q_1, state s_{15} is obtained from q_1.

$$r_{10} : app(A,\ [B,\ C],\ D),\ \{isolated(D)\} \Rightarrow list(A).$$

7. **Concluding Remarks.** This paper defined a new class $\mathcal{ES}$ for LEs and proposed a method of proving the correctness of the LEs therein, thereby, expanding the provable range of the correctness of LEs. The proposed method can prove correctness that cannot be proven by the methods proposed in past studies [12, 14], and so we believe that this paper has expanded the provable correctness range. Important features of the proposed method are having introduced ① a bidirectional search and ② the generation of ET rules in a method for proof of correctness. The method uses feature ① to prove correctness based on equivalence of the declarative meaning of two definite clause sets, and uses feature ② to generate new ET rules that can be used in the proof.

7.1. **Application to program generation.** Methods for generating programs are an important area of research for program synthesis. Researchers such as Harada et al. [6], Ikeda et al. [9], and Lu et al. [11] have proposed methods for generating C, C++, and Java programs from specifications. C programs can be generated from ET rules, and we have already developed a system for generating C programs. If ET rules can be automatically generated from specifications, then C programs can be automatically generated (indirectly) from specifications via ET rules. Therefore, methods for making ET rules, such as the LE-based method [13], are very important. The method proposed in this paper is closely connected to the methods for making ET rules. In particular, the method is very useful for guaranteeing the correctness of ET rules made via the LE-based method.

Methods for proving the correctness of logical formulas have been proposed before [7, 8, 16]. However, these proposals were not focused on its application to program generation, so the methods are not closely connected to program generation. As a result, the proposals cannot be applied to problem settings such as those dealt with in this paper.

7.2. **Comparing with current methods.** The method proposed in this paper can prove the correctness of more LEs beyond those that are provable by current methods. Current methods guarantee the correctness of LEs by proving unsatisfiability of atom sets [12]. The unsatisfiability of an atom such as $\mathcal{E}_1 \wedge \neg\mathcal{A}$ is proven for guaranteeing the correctness of an LE $\forall_{\overline{v}}(\exists_{\overline{v}_1}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v}_2}\mathcal{E}_2)$ using these methods. An atom set $\mathcal{A}$ is made from a set $\mathcal{E}_2$ of atoms. To prove unsatisfiability, $\neg\mathcal{A}$ must be computed using ET rules. In conventional studies, provable LEs are confined to LEs in the forms $\forall_{\overline{v}}(\mathcal{E} \leftrightarrow \{eq(x, y)\} \cup \mathcal{E})$ and $\forall_{\overline{v}}(\mathcal{E} \leftrightarrow \{false\})$, in which $\neg\mathcal{A}$ does not become complex. The LE treated in this paper has an existential quantifier on both sides. In addition, the right-hand side of the LE is composed of a set of arbitrary atoms.

In the LE-based method, one ET rule is made from one LE. We surveyed four hundred and eight (408) important ET rules written by students in programming education to solve constraint satisfaction problems. The results of the survey indicated that the correctness of LEs that link to one hundred and seventy-nine (179) rules, representing forty-four percent (44%) of the total rules, can be proven by the proposed method. In contrast, conventional methods can only prove the correctness of LEs that link to eighty-four (84) rules. Thus, this paper has expanded the range of provable LEs and demonstrated that many important ET rules can be made from LEs that can be proven by the proposed method.

Acknowledgment. This study is a joint research project conducted in collaboration with the Information Initiative Center, Hokkaido University, Japan.

REFERENCES

[1] 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*, vol.1125, pp.44-58, 2000.

[2] 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.

[3] K. Akama, E. Nantajeewarawat and H. Koike, Constructing parallel programs based on rule generators, *Proc. of the 1st International Conference on Advanced Communications and Computation*, Barcelona, Spain, pp.173-178, 2011.

[4] J. Chazarain and S. Muller, Automated synthesis of recursive programs from a $\forall\exists$ logical specification, *Journal of Automated Reasoning*, vol.21, no.2, pp.233-275, 1998.

[5] P. Flener, *Logic Program Synthesis from Incomplete Information*, Kluwer Academic Publishers, 1994.

[6] M. Harada, T. Mizuno and S. Hamada, Executable C++ program generation form the structured object-oriented design diagrams, *Transactions of Information Processing Society of Japan*, vol.40, no.7, pp.2988-3000, 1999 (in Japanese).

[7] J. Hsiang and M. Srivas, Automatic inductive theorem proving using Prolog, *Theoretical Computer Science*, vol.54, no.1, pp.3-28, 1987.

[8] G. Huet and J. M. Hullot, Proofs by induction in equational theories with constructors, *Proc. of the 21st Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, pp.96-107, 1980.

[9] M. Ikeda, T. Nakamura, Y. Takata and H. Seki, Algebraic specification of user interface and its automatic implementation, *The Special Interest Group Notes of IPSJ. SE*, vol.2001, no.114, pp.9-16, 2001 (in Japanese).

[10] 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*, New York, USA, pp.147-150, 2005.

[11] Y. Lu, H. Awaya, H. Seki, M. Fujii and K. Ninomiya, On a translation from algebraic specifications of abstract sequential machines into programs, *The IEICE Transactions on Information and Systems*, vol.J73-D-I, no.2, pp.201-213, 1990 (in Japanese).

[12] K. Miura, K. Akama, H. Koike and H. Mabuchi, Proof of unsatisfiability of atom sets based on computation by equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.9, no.11, pp.4419-4430, 2013.

[13] 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.

[14] 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.

[15] 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.

[16] A. Sakurai and H. Motoda, Proving definite clauses without explicit use of inductions, *Lecture Notes in Computer Science*, vol.383, pp.11-26, 1989.

[17] H. Yoshida, H. Kato and H. Sugimoto, Retrieval of software module functions using first-order predicate logical formulae, *Logic Programming'85, Lecture Notes in Computer Science*, vol.221, pp.117-127, 1986.