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

ICIC International ©2023 ISSN 1349-4198
pp. **1827–1845**

AUTOMATIC PROOF OF LOGICAL EQUIVALENCES BELONGING TO THE ES CLASS USING CONFLUENCE SEARCH INCORPORATING SLICING

Katsunori Miura[1] and Kiyoshi Akama[2]

[1]Department of Information and Management Science
Faculty of Commerce
Otaru University of Commerce
3-5-21, Midori, Otaru, Hokkaido 047-8501, Japan
k-miura@res.otaru-uc.ac.jp

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

Received March 2023; revised June 2023

Abstract. *This paper proposes a framework for automatically proving logical equivalences belonging to the equivalent simple class. A logical equivalence is a mathematical formula that describes the equivalence of the declarative meaning between two sets of atomic formulas, and is used to generate programs based on equivalent transformation. In the equivalent simple class, atomic formulas use universally quantified variables and existentially quantified variables in addition to constants. The proposed framework uses a confluence search to determine the correct logical equivalence. A confluence search proves an equivalence relation between two definite clauses to determine the correct logical equivalence. Two definite clauses are generated from a logical equivalence. Although confluence search is useful for proving the correctness of logical equivalences, no method or principle for automation has been proposed. To automate the proof of logical equivalences, the proposed framework incorporates a method called slicing into the confluence search. A slicing is an original method for generating subproblems from the original problem given to the confluence search. Experimental results indicate that the proposed framework can reduce the computation time required to prove one logical equivalence from minutes to seconds and even milliseconds. For example, using the proposed framework, each logical equivalence given in the experiment is proven in eleven seconds on average.*
Keywords: Equivalent transformation, Logical equivalence, Confluence search, Slicing, Automatic proof

1. **Introduction.** A *logical equivalence* (LE) [1, 2] is a mathematical formula that describes the equivalence of the declarative meaning between two sets of *atomic formulas* (atoms), and is used to generate programs that solve problems efficiently. Different classes of LEs, such as C2LE, Speq, and False, have been proposed in previous studies [3, 4, 5], which are useful for generating efficient programs. The full forms of C2LE and Speq are omitted because these classes are outside the scope of this paper. In this paper, we use LEs belonging to the *equivalent simple* (ES) class. In the ES class, atoms use universally quantified variables and existentially quantified variables in addition to constants. LEs belonging to the C2LE, Speq, and False classes are special formulas of the ES class. Miura et al. [2] showed that using correct LEs can generate correct programs; therefore, LEs are

DOI: 10.24507/ijicic.19.06.1827

very important in suggesting methods to generate correct programs. The correctness of LEs is defined in Section 2.2. We believe that generating several correct LEs will lead to the generation of a wide variety of correct programs.

Previous studies [3, 4, 6] have proposed different methods for proving the correctness of LEs. These extant methods prove LEs based on the theoretical basis of correct LEs provided in [2]. The method proposed in [3] proves the equality of two different variables based on constraints determined from given atoms, and is mainly used for LEs belonging to the Speq class. Next, the method proposed in [4] proves the unsatisfiability of atoms appearing in the targeted LE, and is mainly used for LEs belonging to the False class. Finally, the method proposed in [6] proves the equivalence of the declarative meaning between two definite clauses. Previous studies have focused on manual proofs, so automation of these extant methods has not been discussed. In this paper, we use the proof method [6] as a basis for proposing a framework for automatically proving LEs belonging to the ES class.

A *confluence search* is an original method to guarantee the correctness of LEs by proving the equivalence of the declarative meaning between two definite clauses. In this paper, a confluence search is used to determine the correct LE. If two definite clauses are equivalent, the same set of clauses is obtained from them. These two definite clauses are generated from an LE to be proven. A confluence search performs clause replacement to determine whether both definite clauses produce the same set of clauses. The initial state of clause replacement is a singleton comprising each of the two definite clauses. Given a correct LE, we obtain the same set of clauses. However, the correctness of the given LE is not guaranteed if the same set of clauses cannot be obtained; that is, it is unknown whether the given LE is correct. Therefore, an efficient method is required to find the same set of clauses.

In the confluence search, it is not easy to find the same set of clauses, because different sets of clauses are obtained depending on the order in which the procedures are applied. Therefore, although confluence search is useful for proving the correctness of LEs, no method or principle for automation has been proposed. Currently, the correctness of LEs is proven manually, so if the number of LEs to be proven is large, the computation time becomes impractical. For example, the LE generator proposed in [5] generated approximately twelve thousand five hundred LEs. To automate the proof of LEs, this paper proposes a framework that incorporates a method called *slicing* into the confluence search. A slicing is an original method for generating multiple relations between two definite clauses from the original problem given to the confluence search, and is used to simplify the original problem. Thus, the multiple relations generated by the slicing are subproblems of the original problem. A set of subproblems is used to prove the LE using induction. To determine a set of subproblems, the proposed framework uses a *problem structure vector* (PSV) that specifies the parameters and settings required for induction. Multiple PSVs are determined, because there are many different sets of subproblems for the original problem. The proposed framework repeatedly selects PSV candidates until an appropriate PSV is found.

Given an LE, the proposed framework performs the following tasks to prove the LE: (a) generate two definite clauses from the LE; (b) determine an equivalence relation between two definite clauses; (c) generate a set of subproblems based on a PSV from the equivalence relation, where a PSV is randomly selected; (d) prove that all subproblems specify equivalence relations and return true if all proofs hold. If the result is true, the LE is correct. Otherwise, repeat from task (c) with the other PSV. The slicing adds tasks (c) and (d) to the original process of the confluence search. If the proposed framework

does not contain the slicing, we directly prove that the equivalence relation determined in task (b) holds.

To enable automatic proofs of LEs, we need to implement a solver program that runs the automation framework proposed in this paper. Thus, we implement a solver program for automatic proofs of LEs and use the solver program to evaluate the proposed framework. The proposed framework includes procedures that describe complex controls to correctly check the status of definite clauses while preserving the declarative meaning of the original problem given to the confluence search. For example, there are controls for determining whether a given definite clause represents a recursive state and whether clause replacement continues. Therefore, it is important to select an appropriate framework for program synthesis. An *equivalent transformation* (ET) [7, 8, 9, 10] is a program synthesis framework that can generate completely correct programs that solve problems. The ET framework is used to develop various solver programs for query-answering problems; for example, Miura et al. [11, 12] developed a *cloud services brokerage* (CSB) system [13, 14, 15]. Programming based on the ET framework allows programmers to run imperfect programs and find imperfections that exist within programs, thus developing more efficient and stable solvers. Therefore, the ET framework helps develop larger and more complex solver programs. In the ET framework, a problem is specified using definite clauses, and a program is a set of rewriting rules called *ET rules*, each of which replaces one of the definite clauses, called a *goal clause*, with one or more clauses while preserving the declarative meaning of the union of the original clause and problem clauses. In this study, solver programs are generated using ET rules. Programmers add new ET rules and improve existing ET rules to complete solver programs. For example, the CSB system developed in [11, 12] comprises approximately four hundred and sixty ET rules.

This paper evaluates the effectiveness and efficiency of the proposed framework based on two perspectives. One perspective is whether the proposed framework can select an appropriate PSV to prove the LE, and the other perspective is whether the proposed framework can find several correct LEs within working time. Many useful LEs are generated from goal clauses. LEs used in the experiment are generated from goal clauses that appear in the actual computation of problem solving. Furthermore, we show that the proposed framework is useful for automatically generating ET rules from LEs.

The remainder of this paper is organized as follows. Section 2 defines the LE class used in this paper and presents examples of LEs that belong to that class. This section also defines the correctness of LEs. Section 3 explains the process of using the confluence search to prove LEs. Section 4 explains the process of using the slicing to generate a set of subproblems from the original problem given to the confluence search. Section 5 describes automatic proofs of LEs using a new framework that incorporates the slicing into the confluence search. This section also describes specifications and programs based on ET-based program synthesis. Section 6 discusses the effectiveness and efficiency of using the proposed framework based on experimental evaluations. Section 7 presents concluding remarks.

2. Logical Equivalence.

2.1. Definition of LEs.

An LE [2] describes the equivalence of the declarative meaning between two sets of atoms according to first-order predicate logic [16, 17]. An atom comprises one predicate and zero or more terms. For example, given a predicate p and two terms t_1 and t_2, an atom with two terms is written as $p(t_1, t_2)$. The LE is similar to *constraint handling rules* [18, 19] in that it focuses on the equivalence of atoms. Different LE classes can be determined by changing the constraints of an atom set. Various classes

of LEs such as C2LE, Speq, and False have been proposed in [3, 4, 5]. The LE class used in this paper is called the ES class. LEs belonging to the C2LE, Speq, and False classes are special formulas of the ES class. In the ES class, atoms use universally quantified variables and existentially quantified variables in addition to constants. The ES class has the following syntax:

$$\forall_{\overline{v_u}} \left(\exists_{\overline{v_{e_1}}} \mathcal{E}_1 \leftrightarrow \exists_{\overline{v_{e_2}}} \mathcal{E}_2 \right), \tag{1}$$

where $\mathcal{E}_1$ and $\mathcal{E}_2$ are atom sets, $\overline{v_{e_1}}$ is a set of variables that only appear in $\mathcal{E}_1$, $\overline{v_{e_2}}$ is a set of variables that only appear in $\mathcal{E}_2$, and $\overline{v_u}$ is a set of variables that appear in both $\mathcal{E}_1$ and $\mathcal{E}_2$.

2.2. **Correctness of LEs.** For any LE class, the correctness of LEs depends on the declarative meaning of predicates. All predicates are defined by definite clauses. Let $\mathbb{D}$ be a set of definite clauses. Given $\mathbb{D}$, $\mathcal{M}(\mathbb{D})$ is the declarative meaning of $\mathbb{D}$ and is defined as a minimal model of $\mathbb{D}$ [20]. An LE is correct with respect to $\mathbb{D}$ iff $\mathcal{M}(\mathbb{D})$ is a model of the LE.

2.3. **LE examples.** LE examples are written using the *list*, *eq*, *app*, and *rev* predicates. The meaning of each predicate is defined as follows. The *list* predicate specifies that the value type is a list. The *eq* predicate specifies that two terms are equal. The *app* predicate specifies the concatenation of two lists. The *rev* predicate specifies the reverse order of elements in a list. Furthermore, in this paper, variables appearing in atoms are written with a single capital letter. In the LE examples in this section, A, B, C, D, E, and F are variables. The following LEs are examples of LEs belonging to the ES class.

le_1: $\forall_{\{A\}} \left(\exists_{\{B\}} \{rev(A, B)\} \leftrightarrow \{list(A)\} \right)$

le_2: $\forall_{\{A,B,C\}} (\{rev(A, B), rev(A, C)\} \leftrightarrow \{eq(B, C), rev(A, B)\})$

le_3: $\forall_{\{A,B,D,E\}} \left(\exists_{\{C\}} \{app(A, B, C), app(C, [D], E)\} \leftrightarrow \exists_{\{F\}} \{app(B, [D], F), app(A, F, E)\} \right)$

In le_1, $\{list(A)\}$ is the same as $\exists_{\{\ \}}\{list(A)\}$, and le_1 omits $\exists_{\{\ \}}$. le_2 also omits $\exists_{\{\ \}}$. A correct LE specifies that the truth values of $\exists_{\overline{v_{e_1}}} \mathcal{E}_1$ and $\exists_{\overline{v_{e_2}}} \mathcal{E}_2$ are equal for all ground instances for variables on $\overline{v_u}$. These examples le_1, le_2, and le_3 are correct LEs. For example, le_3 specifies that the truth values of $\exists_{\{C\}}\{app(A, B, C), app(C, [D], E)\}$ and $\exists_{\{F\}}\{app(B, [D], F), app(A, F, E)\}$ are equal for all ground instances of A, B, D, and E.

3. Confluence Search.

3.1. **Outline of confluence search.** A confluence search guarantees the correctness of LEs by proving the equivalence of the declarative meaning between two definite clauses. Let le be an LE belonging to the ES class defined in Formula (1). Given le, the confluence search performs the following tasks to prove the correctness of le: (a) generate two definite clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ from le; (b) determine an equivalence relation $\mathcal{R}$ between $\mathcal{C}_1$ and $\mathcal{C}_2$; (c) prove the equivalence specified by $\mathcal{R}$. If $\mathcal{R}$ holds, the correctness of le is guaranteed; that is, le specifies that the truth values of $\exists_{\overline{v_{e_1}}} \mathcal{E}_1$ and $\exists_{\overline{v_{e_2}}} \mathcal{E}_2$ are equal for all ground instances for variables on $\overline{v_u}$.

3.1.1. *Generation of two definite clauses $\mathcal{C}_1$ and $\mathcal{C}_2$.* Two definite clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ are generated from le by simple formula transformation. A definite clause $\mathcal{C}_1$ is defined as follows:

$$H \leftarrow X_1, \ldots, X_n,$$

where H is $ans(v_1, \ldots, v_i)$, $\overline{v_u}$ is $\{v_1, \ldots, v_i\}$, and $\mathcal{E}_1$ is $\{X_1, \ldots, X_n\}$. An *ans* atom is a special atom that represents the answer. The body part of $\mathcal{C}_1$ comprises all atoms in $\mathcal{E}_1$.

For example, if $\mathcal{E}_1$ has three atoms, the body part of $\mathcal{C}_1$ has three atoms. A definite clause $\mathcal{C}_2$ is defined as follows:

$$H \leftarrow Y_1, \ldots, Y_m,$$

where H is $ans(v_1, \ldots, v_i)$, $\overline{v_u}$ is $\{v_1, \ldots, v_i\}$, and $\mathcal{E}_2$ is $\{Y_1, \ldots, Y_m\}$. H appearing in $\mathcal{C}_2$ is the same as that in $\mathcal{C}_1$. The body part of $\mathcal{C}_2$ comprises all atoms in $\mathcal{E}_2$.

3.1.2. *Determination of an equivalence relation* $\mathcal{R}$. An equivalence relation $\mathcal{R}$ specifies that $\mathcal{C}_1$ is equivalent to $\mathcal{C}_2$ with respect to the declarative meaning determined by the definition of predicates. Here, we assume that le is generated using predicates on $\mathbb{D}$. Formula (2) represents an equivalence relation $\mathcal{R}$ with respect to the declarative meaning of $\mathbb{D}$.

$$\mathcal{M}(\{H \leftarrow X_1, \ldots, X_n\} \cup \mathbb{D}) = \mathcal{M}(\{H \leftarrow Y_1, \ldots, Y_m\} \cup \mathbb{D}) \tag{2}$$

If Formula (2) holds, le is correct with respect to $\mathbb{D}$, because both $\mathcal{C}_1$ and $\mathcal{C}_2$ produce the same set of ground ans atoms. Therefore, the confluence search proves that Formula (2) holds.

3.1.3. *Proof of equivalence specified by* $\mathcal{R}$. The proof computation determines whether the same set $\mathcal{K}$ of clauses is obtained from both $\{\mathcal{C}_1\}$ and $\{\mathcal{C}_2\}$ by performing clause replacement. A singleton $\{\mathcal{C}_1\}$ appears on the left-hand side of Formula (2), whereas a singleton $\{\mathcal{C}_2\}$ appears on the right-hand side of Formula (2). Let seq_{cl} be a sequence of clause sets obtained by clause replacement when $\{cl\}$ is given as the initial state, where cl is a definite clause. If both $seq_{\mathcal{C}_1}$ and $seq_{\mathcal{C}_2}$ have the same set $\mathcal{K}$, Formula (2) holds, because $\{\mathcal{C}_1\}$ and $\{\mathcal{C}_2\}$ are equivalent with respect to the declarative meaning of $\mathbb{D}$. The same set $\mathcal{K}$ does not necessarily occur at the same position in $seq_{\mathcal{C}_1}$ and $seq_{\mathcal{C}_2}$. Therefore, $seq_{\mathcal{C}_1}$ and $seq_{\mathcal{C}_2}$ often have different lengths.

3.2. **Example of LE proof using confluence search.** This section specifies LEs using the predicates app, rev, and eq. The predicates app, rev, and eq have the same meaning as given in Section 2.3. Definite clauses that define app and rev are as follows:

$$cl_A\colon\ app([\,], X, X) \leftarrow$$
$$cl_B\colon\ app([A \mid X], Y, [A \mid Z]) \leftarrow app(X, Y, Z)$$
$$cl_C\colon\ rev([\,], [\,]) \leftarrow$$
$$cl_D\colon\ rev([A \mid X], Z) \leftarrow app(Y, [A], Z), rev(X, Y)$$

As explained in Section 2.3, variables are written with a single capital letter, so A, X, Y, and Z appearing in these definite clauses are variables. In this example, these definite clauses are contained in $\mathbb{D}$. This section uses the following LEs le_A and le_B to illustrate the computational flow of a confluence search. le_A is a special formula of le_B, obtained by applying $[a, b, c]$ to A in le_B. In $[a, b, c]$, a, b, and c are symbols. Whereas, le_B is the same as le_2 given in Section 2.3. Therefore, we omit the explanation of these LEs.

$$le_A\colon\ \forall_{\{A,B\}}(\{rev([a, b, c], A), rev([a, b, c], B)\} \leftrightarrow \{eq(A, B), rev([a, b, c], A)\})$$
$$le_B\colon\ \forall_{\{A,B,C\}}(\{rev(A, B), rev(A, C)\} \leftrightarrow \{eq(B, C), rev(A, B)\})$$

3.2.1. *Proof of correctness of* le_A. Given le_A, two definite clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ generated from le_A are written as follows:

$$\mathcal{C}_1\colon\ ans(A, B) \leftarrow rev([a, b, c], A), rev([a, b, c], B)$$
$$\mathcal{C}_2\colon\ ans(A, B) \leftarrow eq(A, B), rev([a, b, c], A)$$

In le_A, $\overline{v_u}$ is $\{A, B\}$, so an ans atom is $ans(A, B)$. $\mathcal{E}_1$ is $\{rev([a, b, c], A), rev([a, b, c], B)\}$ and $\mathcal{E}_2$ is $\{eq(A, B), rev([a, b, c], A)\}$, so the bodies of $\mathcal{C}_1$ and $\mathcal{C}_2$ are written as above. Subsequently, an equivalence relation $\mathcal{R}$ is determined according to Formula (2). In this example, $\mathcal{R}$ is written as follows:

$$\begin{aligned}&\mathcal{M}(\{ans(A, B) \leftarrow rev([a, b, c], A), rev([a, b, c], B)\} \cup \mathbb{D})\\ =\ &\mathcal{M}(\{ans(A, B) \leftarrow eq(A, B), rev([a, b, c], A)\} \cup \mathbb{D})\end{aligned}$$

If this $\mathcal{R}$ holds, the correctness of le_A with respect to $\mathbb{D}$ is guaranteed. This example performs clause replacement according to unfolding operations [8] determined by cl_A, cl_B, cl_C, and cl_D. As a result of performing clause replacement with $\{\mathcal{C}_1\}$ as the initial state, the following clause set appears in $seq_{\mathcal{C}_1}$. This section omits the process of clause replacement.

$$\{ans([c, b, a], [c, b, a]) \leftarrow\}$$

If this clause set appears in $seq_{\mathcal{C}_2}$, $\mathcal{R}$ is guaranteed to hold because the same set $\mathcal{K}$ exists. As a result of performing clause replacement with $\{\mathcal{C}_2\}$, this clause set appears in $seq_{\mathcal{C}_2}$. Consequently, le_A is correct with respect to $\mathbb{D}$ because $\mathcal{R}$ holds.

3.2.2. *Proof of correctness of le_B.* Given le_B, two definite clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ generated from le_B are written as follows:

$$\mathcal{C}_1\colon\ ans(A, B, C) \leftarrow rev(A, B), rev(A, C)$$

$$\mathcal{C}_2\colon\ ans(A, B, C) \leftarrow eq(B, C), rev(A, B)$$

In le_B, $\overline{v_u}$ is $\{A, B, C\}$, so an ans atom is $ans(A, B, C)$. $\mathcal{E}_1$ is $\{rev(A, B), rev(A, C)\}$ and $\mathcal{E}_2$ is $\{eq(B, C), rev(A, B)\}$, so the bodies of $\mathcal{C}_1$ and $\mathcal{C}_2$ are written as above. In this example, an equivalence relation $\mathcal{R}$ is written as follows:

$$\begin{aligned}&\mathcal{M}(\{ans(A, B, C) \leftarrow rev(A, B), rev(A, C)\} \cup \mathbb{D})\\ =\ &\mathcal{M}(\{ans(A, B, C) \leftarrow eq(B, C), rev(A, B)\} \cup \mathbb{D})\end{aligned}$$

If this $\mathcal{R}$ holds, the correctness of le_B with respect to $\mathbb{D}$ is guaranteed. This example also uses unfolding operations determined by cl_A, cl_B, cl_C, and cl_D. From the definition of predicates, variables A, B, and C are specialized to lists. The next task is to generate $seq_{\mathcal{C}_1}$ and $seq_{\mathcal{C}_2}$ by clause replacement; however, it is very difficult to find the same set $\mathcal{K}$ that appears in both $seq_{\mathcal{C}_1}$ and $seq_{\mathcal{C}_2}$. This is because it is not easy to find a definite clause specifying that B and C are always equal for all lists that apply to A. For example, as a result of performing clause replacement with $\{\mathcal{C}_1\}$, the following set Cs_1 of clauses appears in $seq_{\mathcal{C}_1}$.

$$\begin{aligned}Cs_1\colon\ \{&ans([\,], [\,]) \leftarrow\\ &ans([A], [A]) \leftarrow\\ &ans([A|B], [C|D]) \leftarrow rev(E, [A|F]), rev(E, [C|G]), app(F, [H], B), app(G, [H], D)\}\end{aligned}$$

Whereas, the following set Cs_2 of clauses appears in $seq_{\mathcal{C}_2}$.

$$\begin{aligned}Cs_2\colon\ \{&ans([\,], [\,]) \leftarrow\\ &ans([A], [A]) \leftarrow\\ &ans([A|B], [A|B]) \leftarrow rev(C, [A|D]), app(D, [E], B)\}\end{aligned}$$

The first and second clauses appearing in Cs_1 and Cs_2 are the same, respectively. Thus, we need to generate the same state from the third clause of each of Cs_1 and Cs_2. However, generating the same state is very difficult; therefore, the framework proposed in this paper incorporates the slicing into the confluence search.

4. Slicing.

4.1. **Outline of slicing.** A slicing generates a set of subproblems from an equivalence relation $\mathcal{R}$ defined in Formula (2). A set of subproblems is a set of multiple relations between two definite clauses, and is used to prove the LE using induction. Given $\mathcal{R}$, the slicing performs the following tasks to generate a set of subproblems: (a) select one or more variables from an *ans* atom, and then determine a set $\mathbb{V}$ of variables; (b) specify the number of induction steps, and then determine a set Δ of substitutions to specialize variables in $\mathbb{V}$; (c) generate a set $\mathbb{R}$ of subproblems determined from $\mathcal{R}$ and Δ. The proposed framework proves that all subproblems appearing in $\mathbb{R}$ hold. If all subproblems hold, $\mathcal{R}$ is guaranteed to hold.

4.1.1. *Determination of a set $\mathbb{V}$ of variables.* Given an atom A, let $\mathsf{var}(A)$ be a set of all variables that appear in A. Since $\overline{v_u}(= \{v_1, \ldots, v_i\})$ is a set of universally quantified variables that can be specialized to any ground instance, the proposed framework specifies variables on $\overline{v_u}$ as inductive variables. Therefore, a set $\mathbb{V}$ comprises variables that appear in an *ans* atom; that is, $\mathbb{V} \subseteq \mathsf{var}(ans(v_1, \ldots, v_i))$. One or more variables are randomly selected from $\mathsf{var}(ans(v_1, \ldots, v_i))$. This is because we cannot reveal the appropriate variables until the targeted LE is proven.

4.1.2. *Determination of a set Δ of substitutions.* Let $\mathbb{K}$ be a set of assigned values. Given $\mathbb{K}$ and $\mathbb{V}$, let $\mathsf{sub}(\mathbb{K}, \mathbb{V})$ be a set of all substitutions determined from $\mathbb{K}$ and $\mathbb{V}$. For example, a substitution $\{V/K\}$ specifies that $K(\in \mathbb{K})$ applies to $V(\in \mathbb{V})$. Given any set E, let $\mathsf{size}(E)$ be the number of elements appearing in E. The proposed framework specifies the base case and inductive step of induction by specializing variables in $\mathbb{V}$. A set Δ is used to specialize variables in $\mathbb{V}$ and is $\{\delta_1, \ldots, \delta_i\}$, where δ_i is a set of substitutions satisfying the following conditions:

1) $\delta_i = [\{V_1/K_1\}, \ldots, \{V_j/K_j\}]$,
2) $\{V_j/K_j\} \in \mathsf{sub}(\mathbb{K}, \mathbb{V})$,
3) For any V_j in δ_i, V_j occurs uniquely within δ_i,
4) For any K_j in δ_i, variables appearing in K_j are not used in other assigned values in δ_i,
5) $\mathsf{size}(\delta_i) = \mathsf{size}(\mathbb{V})$.

If $V(\in \mathbb{V})$ applies to lists, the proposed framework specializes V using finite and infinite lists as assigned values. A finite list has a fixed length, whereas an infinite list does not. However, infinite lists have a specified minimum length. For example, $[X|Y]$ means that a list has one or more elements. Variable X represents the first element and variable Y represents a list comprising the remaining elements. For one-step induction on variables that can be specialized to any list, [ ] and $[\mathrm{x}_1|\mathrm{x}_2{}^{\sim\mathsf{rs}}]$ are used as assigned values in this paper. Furthermore, for two-step induction, [ ], $[\mathrm{x}_1]$, and $[\mathrm{x}_1, \mathrm{x}_2|\mathrm{x}_3{}^{\sim\mathsf{rs}}]$ are used as assigned values. x_1, x_2, and x_3 represent any variables that are different from each other. [ ] represents an empty list with no elements. In this paper, variables with a tag rs, such as $X^{\sim\mathsf{rs}}$, represent recursive states.

4.1.3. *Generation of a set $\mathbb{R}$ of subproblems.* A set $\mathbb{R}$ contains relations determined by applying Δ to $\mathcal{R}$. A subproblem $R_i(\in \mathbb{R})$ is defined as follows:

$$\mathcal{M}(\{H\delta_i \leftarrow X_1\delta_i, \ldots, X_n\delta_i\} \cup \mathbb{D}) = \mathcal{M}(\{H\delta_i \leftarrow Y_1\delta_i, \ldots, Y_m\delta_i\} \cup \mathbb{D}), \tag{3}$$

where $\delta_i \in \Delta$. All elements in Δ are used to generate $\mathbb{R}$. For example, if Δ has four elements, four subproblems are generated, that is, $\mathbb{R} = \{R_1, R_2, R_3, R_4\}$. By using all elements in Δ, the ground instances covered by $\mathbb{R}$ are equal to $\mathcal{R}$.

4.2. **Example of subproblem generation using slicing.** This section applies the slicing to the equivalence relation $\mathcal{R}$ described in Section 3.2.2. The $\mathcal{R}$ is as follows:

$$\begin{aligned}&\mathcal{M}(\{ans(A,B,C) \leftarrow rev(A,B), rev(A,C)\} \cup \mathbb{D})\\ =\ &\mathcal{M}(\{ans(A,B,C) \leftarrow eq(B,C), rev(A,B)\} \cup \mathbb{D})\end{aligned} \tag{4}$$

Furthermore, we determine the following cases as a set $\mathbb{V}$ and generate a set $\mathbb{R}$ of subproblems. A set $\mathbb{V}$ is a subset of $\mathsf{var}(ans(A,B,C))$.

Case 1: $\mathbb{V} = \{A\}$
Case 2: $\mathbb{V} = \{B,C\}$

From the definition of predicates, variables A, B, and C are specialized to lists. Therefore, this example uses finite and infinite lists as assigned values.

4.2.1. *Generation of set* $\mathbb{R}$ *in Case 1.* In task (b) of the slicing, if one-step induction is specified as the number of induction steps, $\mathsf{sub}(\{[\,], [X|Y^{\sim\mathsf{rs}}]\}, \{A\})$ is a set of two substitutions $\{A/[\,]\}$ and $\{A/[X|Y^{\sim\mathsf{rs}}]\}$; thus, Δ is determined as follows:

$$\Delta = \{[\{A/[\,]\}], \\ [\{A/[X|Y^{\sim\mathsf{rs}}]\}]\}$$

Therefore, the subproblems generated by task (c) are as follows:

$$\begin{aligned}R_1^{\mathsf{c1o}}\colon\ &\mathcal{M}(\{ans([\,],B,C) \leftarrow rev([\,],B), rev([\,],C)\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans([\,],B,C) \leftarrow eq(B,C), rev([\,],B)\} \cup \mathbb{D})\\ R_2^{\mathsf{c1o}}\colon\ &\mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],B,C) \leftarrow rev([X|Y^{\sim\mathsf{rs}}],B), rev([X|Y^{\sim\mathsf{rs}}],C)\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],B,C) \leftarrow eq(B,C), rev([X|Y^{\sim\mathsf{rs}}],B)\} \cup \mathbb{D})\end{aligned}$$

As a result, we obtain $\mathbb{R} = \{R_1^{\mathsf{c1o}}, R_2^{\mathsf{c1o}}\}$. Label $\mathsf{c1o}$ in R_1^{c1o} is used to distinguish from others. Labels are optional. If two-step induction is specified, $\mathsf{sub}(\{[\,], [X], [X,Y|Z^{\sim\mathsf{rs}}]\}, \{A\})$ is a set of three substitutions $\{A/[\,]\}$, $\{A/[X]\}$, and $\{A/[X,Y|Z^{\sim\mathsf{rs}}]\}$; thus, Δ is determined as follows:

$$\Delta = \{[\{A/[\,]\}], \\ [\{A/[X]\}], \\ [\{A/[X,Y|Z^{\sim\mathsf{rs}}]\}]\}$$

Therefore, the subproblems are as follows:

$$\begin{aligned}R_1^{\mathsf{c1t}}\colon\ &\mathcal{M}(\{ans([\,],B,C) \leftarrow rev([\,],B), rev([\,],C)\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans([\,],B,C) \leftarrow eq(B,C), rev([\,],B)\} \cup \mathbb{D})\\ R_2^{\mathsf{c1t}}\colon\ &\mathcal{M}(\{ans([X],B,C) \leftarrow rev([X],B), rev([X],C)\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans([X],B,C) \leftarrow eq(B,C), rev([X],B)\} \cup \mathbb{D})\\ R_3^{\mathsf{c1t}}\colon\ &\mathcal{M}(\{ans([X,Y|Z^{\sim\mathsf{rs}}],B,C) \leftarrow rev([X,Y|Z^{\sim\mathsf{rs}}],B), rev([X,Y|Z^{\sim\mathsf{rs}}],C)\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans([X,Y|Z^{\sim\mathsf{rs}}],B,C) \leftarrow eq(B,C), rev([X,Y|Z^{\sim\mathsf{rs}}],B)\} \cup \mathbb{D})\end{aligned}$$

As a result, we obtain $\mathbb{R} = \{R_1^{\mathsf{c1t}}, R_2^{\mathsf{c1t}}, R_3^{\mathsf{c1t}}\}$. Increasing the number of induction steps divides the area of ground instances specified by the infinite list.

4.2.2. *Generation of set* $\mathbb{R}$ *in Case 2.* In task (b) of the slicing, if one-step induction is specified as the number of induction steps, $\mathsf{sub}(\{[\,], [X|Y^{\sim\mathsf{rs}}], [V|W^{\sim\mathsf{rs}}]\}, \{B,C\})$ is a set of six substitutions $\{B/[\,]\}$, $\{C/[\,]\}$, $\{B/[X|Y^{\sim\mathsf{rs}}]\}$, $\{C/[X|Y^{\sim\mathsf{rs}}]\}$, $\{B/[V|W^{\sim\mathsf{rs}}]\}$, and $\{C/[V|W^{\sim\mathsf{rs}}]\}$; thus, Δ is determined as follows:

$$\begin{aligned}\Delta = \{&[\{B/[\,]\},\{C/[\,]\}],\\ &[\{B/[\,]\},\{C/[V|W^{\sim\mathsf{rs}}]\}],\\ &[\{B/[X|Y^{\sim\mathsf{rs}}]\},\{C/[\,]\}],\\ &[\{B/[X|Y^{\sim\mathsf{rs}}]\},\{C/[V|W^{\sim\mathsf{rs}}]\}]\}\end{aligned}$$

Substitutions $\{B/[X|Y^{\sim\mathsf{rs}}]\}$ and $\{B/[V|W^{\sim\mathsf{rs}}]\}$ have the same result, so $\{B/[X|Y^{\sim\mathsf{rs}}]\}$ is used in this example. Furthermore, the reason for using $\{C/[V|W^{\sim\mathsf{rs}}]\}$ is the same. Therefore, the subproblems generated by task (c) are as follows:

$$\begin{aligned}R_1^{\mathsf{c2o}}\colon\ &\mathcal{M}(\{ans(A,[\,],[\,]) \leftarrow rev(A,[\,]), rev(A,[\,])\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans(A,[\,],[\,]) \leftarrow eq([\,],[\,]), rev(A,[\,])\} \cup \mathbb{D})\\ R_2^{\mathsf{c2o}}\colon\ &\mathcal{M}(\{ans(A,[\,],[V|W^{\sim\mathsf{rs}}]) \leftarrow rev(A,[\,]), rev(A,[V|W^{\sim\mathsf{rs}}])\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans(A,[\,],[V|W^{\sim\mathsf{rs}}]) \leftarrow eq([\,],[V|W^{\sim\mathsf{rs}}]), rev(A,[\,])\} \cup \mathbb{D})\\ R_3^{\mathsf{c2o}}\colon\ &\mathcal{M}(\{ans(A,[X|Y^{\sim\mathsf{rs}}],[\,]) \leftarrow rev(A,[X|Y^{\sim\mathsf{rs}}]), rev(A,[\,])\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans(A,[X|Y^{\sim\mathsf{rs}}],[\,]) \leftarrow eq([X|Y^{\sim\mathsf{rs}}],[\,]), rev(A,[X|Y^{\sim\mathsf{rs}}])\} \cup \mathbb{D})\\ R_4^{\mathsf{c2o}}\colon\ &\mathcal{M}(\{ans(A,[X|Y^{\sim\mathsf{rs}}],[V|W^{\sim\mathsf{rs}}]) \leftarrow rev(A,[X|Y^{\sim\mathsf{rs}}]), rev(A,[V|W^{\sim\mathsf{rs}}])\} \cup \mathbb{D})\\ &= \mathcal{M}(\{ans(A,[X|Y^{\sim\mathsf{rs}}],[V|W^{\sim\mathsf{rs}}]) \leftarrow eq([X|Y^{\sim\mathsf{rs}}],[V|W^{\sim\mathsf{rs}}]), rev(A,[X|Y^{\sim\mathsf{rs}}])\} \cup \mathbb{D})\end{aligned}$$

As a result, we obtain $\mathbb{R} = \left\{R_1^{\mathsf{c2o}}, R_2^{\mathsf{c2o}}, R_3^{\mathsf{c2o}}, R_4^{\mathsf{c2o}}\right\}$. If two-step induction is specified, $\mathbb{R}$ is a set of nine subproblems. In this paper, we omit the set $\mathbb{R}$ generated in the case of two-step induction.

5. Automatic Proof of LEs Using the Confluence Search Incorporating the Slicing.

5.1. **ET-based program synthesis.** In this paper, we use the ET framework [7, 8, 9, 10] to generate solver programs for proving LEs. The ET framework is used to develop various software systems such as CSB systems and learning management systems. Previous studies [2, 21] have proposed methods to generate correct solver programs with respect to given specifications.

In the ET framework, a specification is denoted as $\langle \mathbb{D}, \mathbb{Q} \rangle$, where $\mathbb{Q}$ is a set of queries and a query is a set of definite clauses. The definition of $\mathbb{D}$ is given in Section 2.2. In this paper, $\mathbb{D}$ is called *background knowledge*. Various predicates, called *user-defined predicates*, can be defined in $\mathbb{D}$, and various queries can be made by combining multiple predicates on $\mathbb{D}$. We assume that for any definite clause q appearing in $\mathbb{Q}$, any predicate appearing in the head part of q exists neither in $\mathbb{D}$ nor in the body part of q.

In the ET framework, a program is a set of ET rules, using which we obtain a goal sequence $[G_1, G_2, \ldots, G_n]$, where $G_1 = Q(\in \mathbb{Q})$. For any i in $\{1, 2, \ldots, n-1\}$, G_{i+1} is obtained from G_i using an ET rule. Any definite clause appearing in a goal sequence is called a goal clause. For any query Q in $\mathbb{Q}$, a set $\mathbb{A}$ of answers with respect to Q and $\mathbb{D}$ is obtained from G_n, and is correct if Formula (5) holds. A set $\mathbb{A}$ obtained using ET rules is a set of correct answers.

$$\mathbb{A} = \mathcal{M}(\mathbb{D} \cup Q) - \mathcal{M}(\mathbb{D}) \tag{5}$$

An ET rule applies to a goal clause $g(\in G_i)$ and replaces g with a set Cs of one or more clauses while preserving $\mathcal{M}(\mathbb{D} \cup G_i)$, that is, Formula (6) holds.

$$\mathcal{M}(\mathbb{D} \cup \{g\}) = \mathcal{M}(\mathbb{D} \cup Cs) \tag{6}$$

Therefore, if G_{i+1} is obtained from G_i using an ET rule, G_{i+1} is the union of $(G_i - \{g\})$ and Cs.

5.2. **Outline of automatic proof of LEs.** In the proposed framework, the input is a set $\mathbb{S}^{\mathsf{in}}$ of unchecked LEs belonging to the ES class, whereas the output is a set $\mathbb{S}^{\mathsf{out}}$ of correct LEs. All LEs in $\mathbb{S}^{\mathsf{in}}$ are proven using the confluence search incorporating the slicing, and consequently correct LEs appearing in $\mathbb{S}^{\mathsf{in}}$ are added to $\mathbb{S}^{\mathsf{out}}$. The proposed framework automatically determines the problem structure, such as a set $\mathbb{R}$ of subproblems and the number of clause replacements, according to a three-tuple comprising ivar, step, and repl. Here, we refer to the three-tuples as PSVs, and the description of PSVs is given in Section 5.2.1.

As shown in Figure 1, the flowchart for generating $\mathbb{S}^{\mathsf{out}}$ comprises two phases: 1) generating an equivalence relation $\mathcal{R}$ and 2) proving that $\mathcal{R}$ holds. Furthermore, Phase 1 comprises two tasks (a) and (b), whereas Phase 2 comprises five tasks (c), (d), (e), (f), and (g). These tasks are executed sequentially. Given $\mathbb{S}^{\mathsf{in}}$, the proposed framework generates $\mathbb{S}^{\mathsf{out}}$ according to the following order: (a) select le from $\mathbb{S}^{\mathsf{in}}$; (b) determine an equivalence relation $\mathcal{R}$ between $\mathcal{C}_1$ and $\mathcal{C}_2$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are generated from le; (c) select one $(= \langle \mathsf{ivar}, \mathsf{step}, \mathsf{repl} \rangle)$ of multiple PSV candidates determined from $\mathcal{R}$; (d) determine a set Δ of substitutions from ivar and step; (e) generate a set $\mathbb{R}$ of subproblems determined from $\mathcal{R}$ and Δ; (f) prove that the same set $\mathcal{K}_i$ is obtained for $R_i (\in \mathbb{R})$, where the number of clause replacements is specified by repl; (g) add le to $\mathbb{S}^{\mathsf{out}}$ if $\mathcal{K}_i$ occurs for any R_i. If there exists R_i in which $\mathcal{K}_i$ does not appear, go back to task (c) and select the other PSV. The flow of these tasks is repeated until all LEs in $\mathbb{S}^{\mathsf{in}}$ get execution results.

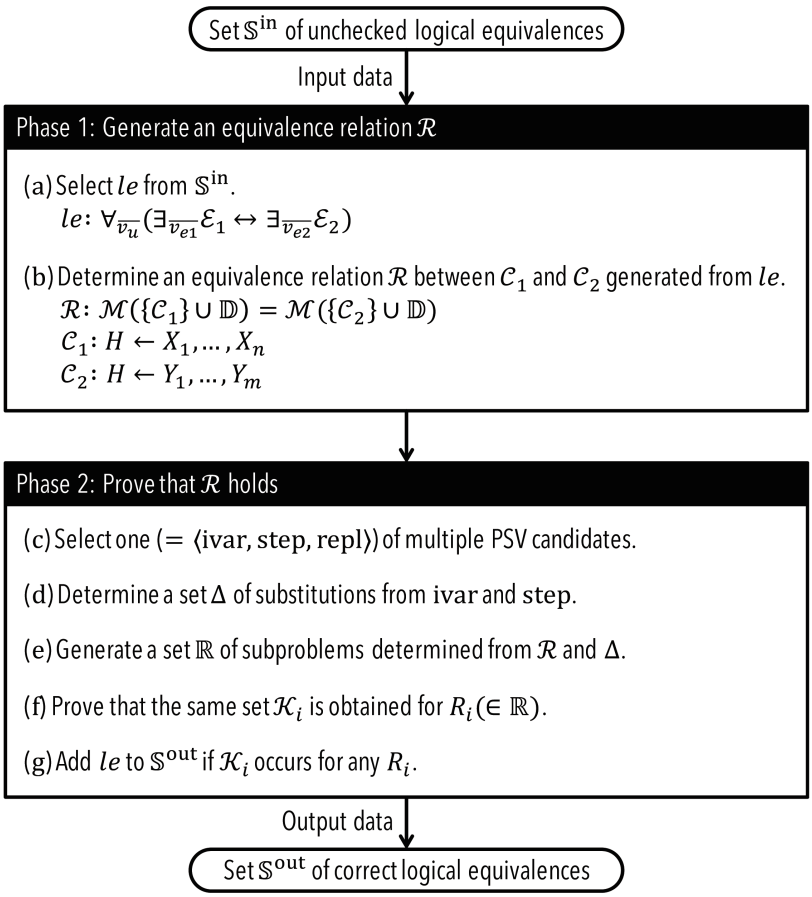

FIGURE 1. Process to generate a set $\mathbb{S}^{\mathsf{out}}$ of correct LEs using the proposed framework

5.2.1. *Problem structure vector.* A PSV is a three-tuple $\langle \mathsf{ivar}, \mathsf{step}, \mathsf{repl} \rangle$, where ivar is a subset of $\mathsf{var}(ans(v_1, \ldots, v_i))$, and step and repl are non-negative integers. ivar specifies a set $\mathbb{V}$ of variables and step specifies the number of induction steps. If step is 1, it requires performing one-step induction; if step is 2, it requires performing two-step induction. In the proposed framework, step is incremented by one, so step is an integer appearing in $\{1, 2, \ldots, n, n+1\}$. repl specifies the number of clause replacements to find $\mathcal{K}_i$ for $R_i (\in \mathbb{R})$. As shown in Section 3.1.3, $\mathcal{K}_i$ appears in both $seq_{\mathcal{C}_1\delta_i}$ and $seq_{\mathcal{C}_2\delta_i}$. $\{\mathcal{C}_1\delta_i\}$ appears on the left-hand side of R_i, whereas $\{\mathcal{C}_2\delta_i\}$ appears on the right-hand side of R_i. If repl is 10, it means to perform clause replacement ten times. The length of each of $seq_{\mathcal{C}_1\delta_i}$ and $seq_{\mathcal{C}_2\delta_i}$ depends on the number of clause replacements. Based on procedures for manual LE proofs, the proposed framework performs clause replacement at least ten times. Furthermore, in the proposed framework, repl is incremented by twenty, so repl is an integer appearing in $\{10, 30, \ldots, m, m + 20\}$.

Next, we present PSV examples. In this example, we use $ans(A, B, C)$ as $ans(v_1, \ldots, v_i)$. For $\mathsf{ivar} \subseteq \{A, B, C\}$, $\mathsf{step} \in \{1, 2, \ldots, n, n + 1\}$, and $\mathsf{repl} \in \{10, 30, \ldots, m, m + 20\}$, the following PSV is generated.

$$\langle \{A\}, 1, 10 \rangle$$
$$\langle \{B, C\}, 2, 30 \rangle$$

$\langle \{A\}, 1, 10 \rangle$ specifies to (1) determine $\{A\}$ as $\mathbb{V}$, (2) perform one-step induction, and (3) perform clause replacement ten times. $\langle \{B, C\}, 2, 30 \rangle$ specifies to (1) determine $\{B, C\}$ as $\mathbb{V}$, (2) perform two-step induction, and (3) perform clause replacement thirty times.

5.2.2. *Four classes of ET rules.* The proposed framework uses four classes of ET rules: *unfolding class*, *equality class*, *user-defined class*, and *induction class*. In task (f), $\mathcal{K}_i$ for $R_i (\in \mathbb{R})$ is found by clause replacement using ET rules belonging to these classes. If there is more than one ET rule applicable to a goal clause, one ET rule is selected according to the following priority order: induction class > equality class > user-defined class > unfolding class. The ET framework guarantees the correctness of the calculation of clause replacements regardless of the order in which ET rules are applied. ET rules belonging to unfolding, equality, or user-defined classes are used in clause replacement to generate $seq_{\mathcal{C}_1\delta_i}$ and $seq_{\mathcal{C}_2\delta_i}$. These classes are used multiple times to replace goal clauses. By contrast, ET rules belonging to the induction class are only used in clause replacement to generate $seq_{\mathcal{C}_1\delta_i}$. This class is used once to replace a goal clause that represents a recursive state.

ET rules belonging to the unfolding class are generated from definite clauses on $\mathbb{D}$. For example, cl_A and cl_B, described in Section 3.2, generate ET rules that apply to atoms of the *app* predicate, whereas cl_C and cl_D generate ET rules that apply to atoms of the *rev* predicate. ET rules belonging to the equality class are given as built-in rules to solver programs that prove LEs. The equality class is used to unify two terms. ET rules belonging to the user-defined class are generated from correct LEs. When generating this class, we use extant methods such as the LE-based method [2]. ET rules belonging to the induction class are special rules, each of which is generated from $le (\in \mathbb{S}^{\mathsf{in}})$. Given $le \left(= \forall_{\overline{v_u}} \left(\exists_{\overline{v_{e_1}}} \mathcal{E}_1 \leftrightarrow \exists_{\overline{v_{e_2}}} \mathcal{E}_2\right)\right)$, the induction class is generated according to the following tasks: (a) attach a tag rs to all variables in $\overline{v_u}$ specified by ivar; (b) determine the head and body parts from $\mathcal{E}_1$ and $\mathcal{E}_2$; (c) generate an *isol* atom from each variable in $\overline{v_{e_1}}$; (d) generate an ET rule using the parts obtained in tasks (b) and (c). An *isol* atom means that the specified variable appearing in a goal clause only appears in atoms that match the head part of an ET rule. If le does not contain $\overline{v_{e_1}}$, task (c) does not generate an *isol* atom.

Next, we present an ET rule belonging to the induction class. In this example, we use le_3 shown in Section 2.3. Furthermore, we use $\{A\}$ as ivar. The le_3 is as follows:

$$\forall_{\{A,B,D,E\}}\left(\exists_{\{C\}}\{app(A,B,C),app(C,[D],E)\}\leftrightarrow\exists_{\{F\}}\{app(B,[D],F),app(A,F,E)\}\right)$$

First, determine $A^{\sim\mathsf{rs}}$ in task (a). Next, in task (b), determine $app(A^{\sim\mathsf{rs}},B,C)$ and $app(C,[D],E)$ as head atoms and $app(B,[D],F)$ and $app(A^{\sim\mathsf{rs}},F,E)$ as body atoms. Subsequently, $isol(C)$ is generated in task (c). Finally, the following ET rule is generated in task (d).

$$app(A^{\sim\mathsf{rs}},B,C),app(C,[D],E),\{isol(C)\}\Rightarrow app(B,[D],F),app(A^{\sim\mathsf{rs}},F,E).$$

5.3. **Example of LE proof using the proposed framework.** This section describes the usage of the proposed framework to prove that le_3 shown in Section 2.3 is correct. Given le_3, the following equivalence relation $\mathcal{R}$ is determined.

$$\begin{aligned}&\mathcal{M}(\{ans(A,B,D,E)\leftarrow app(A,B,C),app(C,[D],E)\}\cup\mathbb{D})\\=\;&\mathcal{M}(\{ans(A,B,D,E)\leftarrow app(B,[D],F),app(A,F,E)\}\cup\mathbb{D})\end{aligned}$$

In this example, we use $\langle\{A\},1,10\rangle$ as the PSV. From the definition of the *app* predicate, variables are specialized to lists, so a set Δ determined from this PSV is as follows:

$$\begin{aligned}\Delta=\{&[\{A/[\,]\}],\\&[\{A/[X|Y^{\sim\mathsf{rs}}]\}]\}\end{aligned}$$

Thus, from $\mathcal{R}$ and Δ above, subproblems R_1^{ex} and R_2^{ex} are generated. That is, $\mathbb{R}=\{R_1^{\mathsf{ex}},R_2^{\mathsf{ex}}\}$.

$$\begin{aligned}R_1^{\mathsf{ex}}\colon\;&\mathcal{M}(\{ans([\,],B,D,E)\leftarrow app([\,],B,C),app(C,[D],E)\}\cup\mathbb{D})\\&=\mathcal{M}(\{ans([\,],B,D,E)\leftarrow app(B,[D],F),app([\,],F,E)\}\cup\mathbb{D})\\R_2^{\mathsf{ex}}\colon\;&\mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],B,D,E)\leftarrow app([X|Y^{\sim\mathsf{rs}}],B,C),app(C,[D],E)\}\cup\mathbb{D})\\&=\mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],B,D,E)\leftarrow app(B,[D],F),app([X|Y^{\sim\mathsf{rs}}],F,E)\}\cup\mathbb{D})\end{aligned}$$

Subsequently, clause replacement is performed for R_1^{ex} and R_2^{ex} to obtain the same sets $\mathcal{K}_1$ and $\mathcal{K}_2$ from R_1^{ex} and R_2^{ex}, respectively. ET rules used in clause replacement are as follows:

$$\begin{aligned}&etr_1\colon\; app([\,],A,B)\Rightarrow eq(B,A).\\&etr_2\colon\; app([A|B],C,D)\Rightarrow eq(D,[A|E]),app(B,C,E).\\&etr_3\colon\; eq(A,B),\{pvar(A),pvar(B)\}\Rightarrow\{A=B\}.\\&etr_4\colon\; eq(A,[B|C]),\{pvar(A),notoccur(A,[B|C])\}\Rightarrow\{A=[B|C]\}.\\&etr_5\colon\; app(A^{\sim\mathsf{rs}},B,C),app(C,[D],E),\{isol(C)\}\Rightarrow app(B,[D],F),app(A^{\sim\mathsf{rs}},F,E).\end{aligned}$$

ET rules etr_1 and etr_2 belong to the unfolding class, each of which specifies a particular procedure in the general unfolding operations of the *app* predicate. ET rules etr_3 and etr_4 belong to the equality class and are used to unify two terms. The *pvar* predicate specifies that a given variable is untagged, that is, a pure variable without rs. The *notoccur* predicate specifies that a given variable does not appear in a given list. ET rule etr_5 belongs to the induction class, and is generated from le_3.

Table 1 shows clause replacements to find $\mathcal{K}_1$. $G_1^{\mathcal{C}_1\delta_1}$ represents $\{\mathcal{C}_1\delta_1\}$ appearing to the left-hand side of R_1^{ex}, and $G_1^{\mathcal{C}_2\delta_1}$ represents $\{\mathcal{C}_2\delta_1\}$ appearing to the right-hand side of R_1^{ex}. Table 2 shows clause replacements to find $\mathcal{K}_2$. $G_1^{\mathcal{C}_1\delta_2}$ represents $\{\mathcal{C}_1\delta_2\}$ appearing to the left-hand side of R_2^{ex}, and $G_1^{\mathcal{C}_2\delta_2}$ represents $\{\mathcal{C}_2\delta_2\}$ appearing to the right-hand side of R_2^{ex}. In these tables, the underlined atoms represent atoms that apply to the ET rule.

TABLE 1. Two sequences $seq_{\mathcal{C}_1\delta_1}$ and $seq_{\mathcal{C}_2\delta_1}$ to find $\mathcal{K}_1$ from R_1^{ex}

(a) Sequence $seq_{\mathcal{C}_1\delta_1}$, and ET rules applied

Clause set appearing in $seq_{\mathcal{C}_1\delta_1}$
$G_1^{\mathcal{C}_1\delta_1}$: $\{ans([\,],B,D,E) \leftarrow \underline{app([\,],B,C)}, app(C,[D],E)\}$
$\Downarrow$ Replace using etr_1
$G_2^{\mathcal{C}_1\delta_1}$: $\{ans([\,],B,D,E) \leftarrow \underline{eq(B,C)}, app(C,[D],E)\}$
$\Downarrow$ Replace using etr_3
$G_3^{\mathcal{C}_1\delta_1}$: $\{ans([\,],B,D,E) \leftarrow app(B,[D],E)\}$

(b) Sequence $seq_{\mathcal{C}_2\delta_1}$, and ET rules applied

Clause set appearing in $seq_{\mathcal{C}_2\delta_1}$
$G_1^{\mathcal{C}_2\delta_1}$: $\{ans([\,],B,D,E) \leftarrow app(B,[D],F), \underline{app([\,],F,E)}\}$
$\Downarrow$ Replace using etr_1
$G_2^{\mathcal{C}_2\delta_1}$: $\{ans([\,],B,D,E) \leftarrow app(B,[D],F), \underline{eq(F,E)}\}$
$\Downarrow$ Replace using etr_3
$G_3^{\mathcal{C}_2\delta_1}$: $\{ans([\,],B,D,E) \leftarrow app(B,[D],E)\}$

*$\mathcal{C}_1\delta_1$ in $G_1^{\mathcal{C}_1\delta_1}$ and $\mathcal{C}_2\delta_1$ in $G_1^{\mathcal{C}_2\delta_1}$ are written for distinction.

TABLE 2. Two sequences $seq_{\mathcal{C}_1\delta_2}$ and $seq_{\mathcal{C}_2\delta_2}$ to find $\mathcal{K}_2$ from R_2^{ex}

(a) Sequence $seq_{\mathcal{C}_1\delta_2}$, and ET rules applied

Clause set appearing in $seq_{\mathcal{C}_1\delta_2}$
$G_1^{\mathcal{C}_1\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow \underline{app([X\|Y^{\sim\mathsf{rs}}],B,C)}, app(C,[D],E)\}$
$\Downarrow$ Replace using etr_2
$G_2^{\mathcal{C}_1\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow \underline{eq(C,[X\|Z])}, app(Y^{\sim\mathsf{rs}},B,Z), app(C,[D],E)\}$
$\Downarrow$ Replace using etr_4
$G_3^{\mathcal{C}_1\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow app(Y^{\sim\mathsf{rs}},B,Z), \underline{app([X\|Z],[D],E)}\}$
$\Downarrow$ Replace using etr_2
$G_4^{\mathcal{C}_1\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow \underline{app(Y^{\sim\mathsf{rs}},B,Z), app(Z,[D],V)}, eq(E,[X\|V])\}$
$\Downarrow$ Replace using etr_5
$G_5^{\mathcal{C}_1\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow app(B,[D],F), app(Y^{\sim\mathsf{rs}},F,V), eq(E,[X\|V])\}$

(b) Sequence $seq_{\mathcal{C}_2\delta_2}$, and ET rules applied

Clause set appearing in $seq_{\mathcal{C}_2\delta_2}$
$G_1^{\mathcal{C}_2\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow app(B,[D],F), \underline{app([X\|Y^{\sim\mathsf{rs}}],F,E)}\}$
$\Downarrow$ Replace using etr_2
$G_2^{\mathcal{C}_2\delta_2}$: $\{ans([X\|Y^{\sim\mathsf{rs}}],B,D,E) \leftarrow app(B,[D],F), app(Y^{\sim\mathsf{rs}},F,V), eq(E,[X\|V])\}$

*$\mathcal{C}_1\delta_2$ in $G_1^{\mathcal{C}_1\delta_2}$ and $\mathcal{C}_2\delta_2$ in $G_1^{\mathcal{C}_2\delta_2}$ are written for distinction.

First, we explain clause replacement until $\mathcal{K}_1$ is obtained. For $G_1^{\mathcal{C}_1\delta_1}$ in Table 1(a), etr_1 applies to $app([\,],B,C)$, and replaces the atom with $eq(B,C)$. The result is $G_2^{\mathcal{C}_1\delta_1}$. For $G_2^{\mathcal{C}_1\delta_1}$, B and C are pure variables, so etr_3 applies to $eq(B,C)$ and unifies B and C. The

result is $G_3^{\mathcal{C}_1\delta_1}$. This state is the same as $G_3^{\mathcal{C}_2\delta_1}$ in $seq_{\mathcal{C}_2\delta_1}$. Therefore, $\mathcal{K}_1$ is $G_3^{\mathcal{C}_1\delta_1}$ and $G_3^{\mathcal{C}_2\delta_1}$. The replacement procedures for $G_1^{\mathcal{C}_2\delta_1}$ and $G_2^{\mathcal{C}_2\delta_1}$ are the same as for $G_1^{\mathcal{C}_1\delta_1}$ and $G_2^{\mathcal{C}_1\delta_1}$, respectively.

Next, we explain clause replacement until $\mathcal{K}_2$ is obtained. For $G_1^{\mathcal{C}_1\delta_2}$ in Table 2(a), etr_2 applies to $app([X|Y^{\sim\mathsf{rs}}], B, C)$, and replaces the atom with $eq(C, [X|Z])$ and $app(Y^{\sim\mathsf{rs}}, B, Z)$. The result is $G_2^{\mathcal{C}_1\delta_2}$. For $G_2^{\mathcal{C}_1\delta_2}$, C is a pure variable that does not appear in $[X|Z]$, so etr_4 applies to $eq(C, [X|Z])$ and unifies C and $[X|Z]$. The result is $G_3^{\mathcal{C}_1\delta_2}$. The replacement procedure for $G_3^{\mathcal{C}_1\delta_2}$ is the same as for $G_1^{\mathcal{C}_1\delta_2}$. The result of applying etr_2 is $G_4^{\mathcal{C}_1\delta_2}$, which represents a recursive state. For $G_4^{\mathcal{C}_1\delta_2}$, Z does not appear in $ans([X|Y^{\sim\mathsf{rs}}], B, D, E)$ and $eq(E, [X|V])$, so etr_5 applies to a set of $app(Y^{\sim\mathsf{rs}}, B, Z)$ and $app(Z, [D], V)$. These atoms are replaced with two atoms $app(B, [D], F)$ and $app(Y^{\sim\mathsf{rs}}, F, V)$. The result is $G_5^{\mathcal{C}_1\delta_2}$. This state is the same as $G_2^{\mathcal{C}_2\delta_2}$ in $seq_{\mathcal{C}_2\delta_2}$. Therefore, $\mathcal{K}_2$ is $G_5^{\mathcal{C}_1\delta_2}$ and $G_2^{\mathcal{C}_2\delta_2}$. The replacement procedure for $G_1^{\mathcal{C}_2\delta_2}$ is the same as for $G_1^{\mathcal{C}_1\delta_2}$.

Since $\mathcal{K}_1$ and $\mathcal{K}_2$ exist, R_1^{ex} and R_2^{ex} hold. Consequently, the $\mathcal{R}$ determined from le_3 is guaranteed to hold. The correctness of le_3 is guaranteed, so le_3 is added to $\mathbb{S}^{\mathsf{out}}$.

6. Experimental Evaluation.

6.1. **Problem settings.** For experimental evaluation, we developed a proof system based on the proposed framework. In this section, we show that the proof system automatically proves several LEs during working time. LEs are often generated from goal clauses; thus, this experiment uses LEs generated from goal clauses that appear in the actual computation of problem solving. The actual goal clauses are as follows:

$$g_A\colon\ ans([D|F], G, E) \leftarrow app(A, B, C), app(C, [D], E), rev(F, B), rev(G, A)$$

$$g_B\colon\ ans([C|A], D, E) \leftarrow rev(A, B), app(B, [C], D), rev(D, E), neq([C|A], E)$$

The neq predicate appearing in g_B specifies that two terms are not equal. g_A and g_B are goal clauses that appear when proving le_X and le_Y, respectively.

$$le_X\colon\ \forall_{\{A,B,D\}}(\exists_{\{C\}}\{app(A, B, C), rev(C, D)\} \leftrightarrow \exists_{\{E,F\}}\{rev(A, E), rev(B, F), app(F, E, D)\})$$

$$le_Y\colon\ \forall_{\{A,B,C\}}(\{rev(A, B), rev(B, C)\} \leftrightarrow \{eq(A, C), rev(A, B)\})$$

The experiments use two hundred and sixteen LEs generated from g_A and six hundred and twelve LEs generated from g_B. Furthermore, let $\mathbb{S}_A^{\mathsf{in}}$ be a set of LEs generated from g_A, and $\mathbb{S}_B^{\mathsf{in}}$ be a set of LEs generated from g_B. An LE in $\mathbb{S}_A^{\mathsf{in}}$ specifies $\{app(A, B, C), app(C, [D], E)\}$ as $\mathcal{E}_1$, whereas an LE in $\mathbb{S}_B^{\mathsf{in}}$ specifies $\{app(B, [C], D), rev(D, E)\}$ as $\mathcal{E}_1$. The generator proposed in [5] is used to generate LEs from these goal clauses.

Ranges for step and repl are specified in this experiment. As shown in Section 5.2.1, step and repl are elements of the PSV. By predetermining the range of step and repl, the number of PSV candidates is finite. In this experiment, step and repl are determined from $\{1\}$ and $\{10, 30\}$, respectively. Since there is only one candidate for step, this experiment performs one-step induction on all proofs. Furthermore, ivar is selected from singletons that appear in $\mathsf{var}(ans(v_1, \ldots, v_i))$ to reduce the number of PSV candidates. For example, given $\mathsf{var}(ans(A, B, C))$, ivar is selected from $\{\{A\}, \{B\}, \{C\}\}$.

6.2. **Experimental result.** In Table 3, the values in the Size column are the numbers of LEs appearing in the input. The values in the Proven column are the numbers of LEs proven by the proposed framework. By contrast, the values in the Unproven column are the numbers of LEs for which proof has not been completed. The values in the Time

TABLE 3. Execution result of proving LEs appearing in $\mathbb{S}_A^{\mathsf{in}}$ and $\mathbb{S}_B^{\mathsf{in}}$

Input	Size	Number of LE proofs completed		Time (msec)
		Proven	Unproven	
$\mathbb{S}_A^{\mathsf{in}}$	216	4	212	2,467,458±5,390
$\mathbb{S}_B^{\mathsf{in}}$	612	36	576	6,752,132±18,711

column are the average times when executing the proposed framework on the input five times. Plus/minus values are standard variations.

In this experiment, all proofs of LEs use the unfolding, equality, and induction classes. ET rules belonging to the unfolding class specify procedures determined by the definitions of the *app* and *rev* predicates. ET rules belonging to the user-defined class are used to prove LEs appearing in $\mathbb{S}_B^{\mathsf{in}}$, and are as follows:

$$app(A, [\,], B) \Rightarrow eq(A, B), list(A).$$
$$app(A, B, C), app(C, [D], E), \{isol(C)\} \Rightarrow app(B, [D], F), app(A, F, E).$$
$$rev(A, B), list(B) \Rightarrow rev(A, B).$$

These ET rules are generated from LEs belonging to the ES class by simple formula transformation. For example, the second ET rule is generated from le_3 shown in Section 2.3.

Given $\mathbb{S}_A^{\mathsf{in}}$ as input, the proof system finished the computation in 2,467,458 milliseconds and generated a set $\mathbb{S}_A^{\mathsf{out}}$ comprising 4 correct LEs from $\mathbb{S}_A^{\mathsf{in}}$, whereas the proof of 212 LEs could not be completed. However, even with the use of all PSV candidates, these LEs cannot be proven. This is because we know that these LEs are incorrect. Therefore, the proof system was able to find all correct LEs. Next, we show that the proof system can select an appropriate one from multiple PSV candidates. Since le_3 described in Section 2.3 appears in $\mathbb{S}_A^{\mathsf{in}}$, we will use le_3 for explanation. Given le_3, the following equivalence relation $\mathcal{R}$ is determined.

$$\begin{aligned} &\mathcal{M}(\{ans(A, B, D, E) \leftarrow app(A, B, C), app(C, [D], E)\} \cup \mathbb{D}) \\ =\ &\mathcal{M}(\{ans(A, B, D, E) \leftarrow app(B, [D], F), app(A, F, E)\} \cup \mathbb{D}) \end{aligned}$$

In this example, elements of $\langle \mathsf{ivar}, \mathsf{step}, \mathsf{repl} \rangle$ are as follows:

$$\mathsf{ivar} \in \{\{A\}, \{B\}, \{D\}, \{E\}\}$$
$$\mathsf{step} \in \{1\}$$
$$\mathsf{repl} \in \{10, 30\}$$

Thus, there are eight PSV candidates. Using $\langle \{A\}, 1, 10 \rangle$, $\langle \{A\}, 1, 30 \rangle$, or $\langle \{E\}, 1, 30 \rangle$, the slicing can generate an appropriate set $\mathbb{R}$ of subproblems. In running the proof system, $\langle \{A\}, 1, 10 \rangle$ was selected and the following subproblems were generated.

$$\begin{aligned} &\mathcal{M}(\{ans([\,], B, D, E) \leftarrow app([\,], B, C), app(C, [D], E)\} \cup \mathbb{D}) \\ =\ &\mathcal{M}(\{ans([\,], B, D, E) \leftarrow app(B, [D], F), app([\,], F, E)\} \cup \mathbb{D}) \\ &\mathcal{M}(\{ans([X|Y^{\sim \mathsf{rs}}], B, D, E) \leftarrow app([X|Y^{\sim \mathsf{rs}}], B, C), app(C, [D], E)\} \cup \mathbb{D}) \\ =\ &\mathcal{M}(\{ans([X|Y^{\sim \mathsf{rs}}], B, D, E) \leftarrow app(B, [D], F), app([X|Y^{\sim \mathsf{rs}}], F, E)\} \cup \mathbb{D}) \end{aligned}$$

As a result, the above $\mathcal{R}$ is guaranteed to hold, thus proving the correctness of le_3.

Given $\mathbb{S}_B^{\mathsf{in}}$ as input, the proof system finished the computation in 6,752,132 milliseconds and generated a set $\mathbb{S}_B^{\mathsf{out}}$ comprising 36 correct LEs from $\mathbb{S}_B^{\mathsf{in}}$, whereas the proof of 576 LEs

could not be completed. We know that these LEs are incorrect. Next, we show that the proof system can select an appropriate PSV. Since le_Z appears in $\mathbb{S}^{\mathsf{in}}_B$, we will use le_Z for explanation. This LE is a special formula of le_X.

$$\begin{aligned} le_Z\colon\ & \forall_{\{B,C,E\}}(\exists_{\{D\}}\{app(B,[C],D), rev(D,E)\} \\ & \leftrightarrow \exists_{\{F,G\}}\{rev(B,F), rev([C],G), app(G,F,E)\}) \end{aligned}$$

Given le_Z, the following equivalence relation $\mathcal{R}$ is determined.

$$\begin{aligned} & \mathcal{M}(\{ans(B,C,E) \leftarrow app(B,[C],D), rev(D,E)\} \cup \mathbb{D}) \\ = \ & \mathcal{M}(\{ans(B,C,E) \leftarrow rev(B,F), rev([C],G), app(G,F,E)\} \cup \mathbb{D}) \end{aligned}$$

In this example, elements of $\langle \mathsf{ivar}, \mathsf{step}, \mathsf{repl}\rangle$ are as follows:

$$\begin{aligned} \mathsf{ivar} &\in \{\{B\},\{C\},\{E\}\} \\ \mathsf{step} &\in \{1\} \\ \mathsf{repl} &\in \{10, 30\} \end{aligned}$$

Thus, there are six PSV candidates. Using $\langle\{B\}, 1, 30\rangle$, the slicing can generate an appropriate set $\mathbb{R}$ of subproblems. When using $\langle\{B\}, 1, 10\rangle$, the number of clause replacements is not sufficient. In running the proof system, the following subproblems were generated using $\langle\{B\}, 1, 30\rangle$.

$$\begin{aligned} & \mathcal{M}(\{ans([\,],C,E) \leftarrow app([\,],[C],D), rev(D,E)\} \cup \mathbb{D}) \\ = \ & \mathcal{M}(\{ans([\,],C,E) \leftarrow rev([\,],F), rev([C],G), app(G,F,E)\} \cup \mathbb{D}) \\ & \mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],C,E) \leftarrow app([X|Y^{\sim\mathsf{rs}}],[C],D), rev(D,E)\} \cup \mathbb{D}) \\ = \ & \mathcal{M}(\{ans([X|Y^{\sim\mathsf{rs}}],C,E) \leftarrow rev([X|Y^{\sim\mathsf{rs}}],F), rev([C],G), app(G,F,E)\} \cup \mathbb{D}) \end{aligned}$$

From these subproblems, we obtain the following clause sets $\mathcal{K}_A$ and $\mathcal{K}_B$. $\mathcal{K}_A$ is obtained from the first subproblem, whereas $\mathcal{K}_B$ is obtained from the second subproblem.

$$\begin{aligned} \mathcal{K}_A\colon\ & \{ans([\,],C,[C]) \leftarrow\} \\ \mathcal{K}_B\colon\ & \{ans([X|Y^{\sim\mathsf{rs}}],C,[C|F]) \leftarrow rev(Y^{\sim\mathsf{rs}},Z), app(Z,[X],F)\} \end{aligned}$$

As a result, the above $\mathcal{R}$ is guaranteed to hold, thus proving the correctness of le_Z. This means that the proof system can select an appropriate PSV to prove that $\mathcal{R}$ holds. Consequently, the proposed framework can be used for automatic proofs of LEs.

6.3. **Effectiveness and efficiency of using the proposed framework.** One of the contributions is to reduce the computation time required to prove the correctness of the LE. Previously, given an LE, its correctness was manually proven using the confluence search. It requires ten to twenty minutes to manually prove an LE. This is because several steps are performed manually, such as generating $\mathbb{R}$, replacing clauses by ET rules, and finding $\mathcal{K}_i$ for $R_i(\in \mathbb{R})$. For example, manually proving the following LE required nine minutes and forty-five seconds to complete, even though the appropriate PSV was prespecified. This LE appears in $\mathbb{S}^{\mathsf{in}}_A$.

$$\forall_{\{A,B,D,E\}}(\exists_{\{C\}}\{app(A,B,C), app(C,[D],E)\} \leftrightarrow \exists_{\{F\}}\{app(B,[D],F), app(A,F,E)\})$$

As another example, manually proving the following LE required thirteen minutes and two seconds to complete the proof. This LE appears in $\mathbb{S}^{\mathsf{in}}_B$. The appropriate PSV was also used in this proof.

$$\forall_{\{B,C,E\}}(\exists_{\{D\}}\{app(B,[C],D), rev(D,E)\} \leftrightarrow \exists_{\{F,G\}}\{rev(B,F), rev([C],G), app(G,F,E)\})$$

By contrast, the proposed framework can prove one LE in seconds or milliseconds, as most of the steps are performed automatically. For example, the first LE was proven in

2,033 milliseconds and the second LE was proven in 13,228 milliseconds. However, the computation time depends on the number of PSV candidates. In the experimental results shown in Table 3, the average computation time for LE proofs is 11,134 milliseconds. Therefore, the proposed framework helps reduce the computation time for LE proofs. Moreover, using the proposed framework does not require advanced techniques or complex configuration.

Another contribution is to help expand methods for automatically generating ET rules from specifications. Miura et al. [2] proposed a method for generating ET rules via LEs. The method generates ET rules according to the following process: (i) generate a set of LEs from a goal clause; (ii) determine a set of correct LEs by removing incorrect LEs based on proof; (iii) generate an ET rule from a correct LE. Part of the LE belonging to the ES class can be automatically generated using extant frameworks [3, 4, 5] related to step (i). Furthermore, ET rules can be generated from LEs belonging to the ES class by simple formula transformation. Given $le\left(=\forall_{\overline{v_u}}\left(\exists_{\overline{v_{e_1}}}\mathcal{E}_1 \leftrightarrow \exists_{\overline{v_{e_2}}}\mathcal{E}_2\right)\right)$, the syntax for ET rules generated from le is as follows:

$$X_1, \ldots, X_n, \{isol(w_1), \ldots, isol(w_i)\} \Rightarrow Y_1, \ldots, Y_m,$$

where $\overline{v_{e_1}}$ is $\{w_1, \ldots, w_i\}$, $\mathcal{E}_1$ is $\{X_1, \ldots, X_n\}$, and $\mathcal{E}_2$ is $\{Y_1, \ldots, Y_m\}$. The proposed framework automates step (ii); therefore, all processes can be executed automatically. LEs belonging to the ES class are useful to generate ET rules needed for solver programs for LE proofs. Therefore, the proposed framework is important to expand the automatic generation of ET rules.

7. **Conclusions.** This paper proposed a framework for automatically proving LEs belonging to the ES class. To automate LE proofs, the proposed framework incorporates the slicing into the confluence search. The novelty of our approach can be summarized as follows: 1) using the confluence search to replace the proof problem of LEs with the equivalence problem of clause sets; 2) using the slicing to generate subproblems from the original problem given to the confluence search. Furthermore, we provided a PSV that automatically determines the problem structure such as subproblems and the number of clause replacements.

In experimental evaluation, we used LEs generated from goal clauses that appear in the actual computation of problem solving. Experimental results showed that the proposed framework reduces the computation time and work efficiency required for LE proofs and helps automatically generate ET rules from LEs.

As future work, we will expand the proposed framework to find correct LEs more efficiently. For example, we plan to incorporate a filtering function to remove incorrect LEs. Furthermore, we aim to formulate another class of LEs to help generate ET rules that are useful for efficient computation. For example, we plan to propose an LE class that contains the guard part for equivalence of $\mathcal{E}_1$ and $\mathcal{E}_2$.

REFERENCES

[1] 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.

[2] 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.

[3] K. Miura, K. Akama and H. Mabuchi, Generating functionality-based rules for program construction, *International Journal of Innovative Computing, Information and Control*, vol.5, no.9, pp.2463-2479, 2009.

[4] 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.
[5] K. Miura and K. Akama, Generation of logical equivalences belonging to the *C2LE* class applied to program synthesis based on equivalent transformation, *International Journal of Innovative Computing, Information and Control*, vol.17, no.4, pp.1119-1135, 2021.
[6] K. Miura and K. Akama, ET-based bidirectional search for proving formulas in the class *ES*, *International Journal of Innovative Computing, Information and Control*, vol.10, no.6, pp.1999-2009, 2014.
[7] K. Akama, E. Nantajeewarawat and H. Koike, Model-intersection problems with existentially quantified function variables, *The 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol.2, pp.52-63, 2016.
[8] K. Akama and E. Nantajeewarawat, Unfolding existentially quantified sets of extended clauses, *The 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol.2, pp.96-103, 2016.
[9] K. Akama and E. Nantajeewarawat, Skolemization that preserves logical meanings, *International Journal of Innovative Computing, Information and Control*, vol.17, no.1, pp.1-13, 2021.
[10] K. Akama and E. Nantajeewarawat, Formalization of logical problems as model-intersection problems on an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.17, no.4, pp.1103-1117, 2021.
[11] K. Miura and M. Munetomo, A predicate logic-defined specification method for systems deployed by intercloud brokerages, *The 2016 IEEE International Conference on Cloud Engineering Workshops (IC2EW)*, pp.172-177, 2016.
[12] K. Miura, C. Powell and M. Munetomo, Optimal and feasible cloud resource configurations generation method for genomic analytics applications, *The 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp.137-144, 2018.
[13] Gartner, Inc., *Definition of Cloud Services Brokerage (CSB) – IT Glossary*, https://www.gartner.com/en/information-technology/glossary/cloud-services-brokerage-csb, 2021.
[14] F. Micota, M. Erascu and D. Zaharie, Constraint satisfaction approaches in cloud resource selection for component based applications, *The 2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing*, 2018.
[15] S. Bayless, N. Kodirov, S. M. Iqbal, I. Beschastnikh, H. H. Hoos and A. J. Hu, Scalable constraint-based virtual data center allocation, *Artificial Intelligence*, vol.278, 2020.
[16] L. Sterling and E. Shapiro, *The Art of Prolog*, 2nd Edition, The MIT Press, 1994.
[17] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1993.
[18] T. Frühwirth, A devil's advocate against termination of direct recursion, *The 17th International Symposium on Principles and Practice of Declarative Programming*, 2015.
[19] T. Frühwirth, Constraint handling rules – What else?, *arXiv.org*, arXiv: 1701.02668, 2017.
[20] 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.
[21] H. Koike, K. Akama, H. Mabuchi and Y. Shigeta, Rule generation by meta-computation, *The 6th International Conference on Information Systems Analysis and Synthesis*, pp.429-434, 2000.

Author Biography

Katsunori Miura received the B.Sc. and M.Sc. degrees in Business Administration and Information Science from Hokkaido Information University, Japan, in 2001 and 2004, respectively; and the Ph.D. degree in Information Science and Technology from Hokkaido University, Japan, in 2009. He was a lecturer at Information Processing Center, Kitami Institute of Technology, Japan, 2012-2018.

Dr. Miura is currently an associate professor at Department of Information and Management Science, Faculty of Commerce, Otaru University of Commerce, Japan. His research interests include artificial intelligence, optimization, program generation, and cloud computing.

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

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