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

ICIC International ©2022 ISSN 1349-4198
pp. 331–344

SOLVING PROOF PROBLEMS WITH EQUIVALENT TRANSFORMATION RULES

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

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

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

Received August 2021; revised November 2021

Abstract. *The conventional resolution method is often used to solve proof problems on first-order logic. We prove that a proof problem, called a pal-pal proof problem, cannot be solved by the resolution method, although this problem is represented by using first-order formulas. Moreover, we show that the pal-pal problem can be solved by using equivalent transformation (ET) rules. Such a transformational method is called the ET-based method, and is general enough to produce many proof methods, not only the conventional resolution method but also a new method that can solve the pal-pal problem. We conclude that the conventional proof theory is not general enough to be applied for all proof problems on first-order logic, the difficulty of which can be overcome, at least partly, by the ET-based method.*

Keywords: Logical problem solving framework, Model-intersection problem, Equivalent transformation, Proof problem, Query-answering problem, Correctness

1. Introduction.

1.1. **Proof problems and the resolution method.** Assume that A and B are first-order formulas. A proof problem with respect to A and B is a "yes/no" problem; it is concerned with checking whether $A \models B$, i.e., whether A entails B. Proof problems on first-order formulas historically constitute the most important problem class in logical problem solving. Resolution provides us with a refutation proof procedure for logical formulas [1, 2]. Based on the resolution proof method, automated theorem proving has been extensively investigated [3, 4]. Their common foundation has been the first-order formulas containing no built-in constraint atoms.

1.2. **Satisfiability preservation.** To solve a proof problem with respect to A and B, we (i) first convert $A \wedge \neg B$ into a clause set Cs and (ii) try to transform Cs into a clause set Cs' that contains an empty clause ($\leftarrow$). We call the conversion in (i) *formula decomposition*, or CSD (conventional Skolemization-based decomposition) to stress the inclusion of Skolemization steps. CSD maps a first-order formula to a set of clauses preserving satisfiability. Let F be a first-order formula and $\mathrm{CSD}(F) = Cs$. Then $\mathit{Models}(F) = \emptyset$ iff

DOI: 10.24507/ijicic.18.01.331

$Models(Cs) = \emptyset$. To show $Models(Cs) = \emptyset$, we find a sequence starting from Cs $(= Cs_1)$ and ending with Cs_n that contains an empty clause:

$$Cs_1 \rightarrow Cs_2 \rightarrow \cdots \rightarrow Cs_n.$$

Such a sequence is obtained by inference rules such as resolution that preserve satisfiability. This proof method can be seen as a resolution-based method.

1.3. **Built-in constraint atoms.** Built-in constraint atoms, such as *eq*-atoms, *neq*-atoms, and inequality-atoms, play a very important role in knowledge representation [5]. They are indispensable for practical applications. This is suggested from many built-in constraint atoms that prolog provides in spite of the underlying proof theory on first-order formulas without built-in constraint atoms [6, 7].

Let $\mathrm{FOL_p}$ be the set of all first-order formulas without built-in constraint atoms, and let $\mathrm{FOL_c}$ be the set of all first-order formulas possibly including built-in constraint atoms. $\mathrm{FOL_c}$ is a superset of $\mathrm{FOL_p}$. Theory extension from $\mathrm{FOL_p}$ to $\mathrm{FOL_c}$ is of fundamental importance to grow logic for practical knowledge science.

1.4. **Limitation of Skolemization.** Skolemization and CSD preserve satisfiability in $\mathrm{FOL_p}$, which is a requirement for correct proof methods based on resolution. However, in the space of $\mathrm{FOL_c}$, i.e., in the presence of built-in constraint atoms, Skolemization and CSD often fail to preserve satisfiability [8, 9]. For this reason, the conventional Skolemization is a major hindrance to development of a general solution for proof problems on $\mathrm{FOL_c}$. We will prove that the resolution-based method has a limitation (incompleteness) on $\mathrm{FOL_c}$ in spite of the completeness theorem on $\mathrm{FOL_p}$.

1.5. **ET-based method and objective.** The ET-based method is a solution method of repeated application of equivalent transformation rules (ET rules) [5]. We apply the ET-based method to proving theorems. We will compare the ET-based method with the resolution-based method, and show that the ET-based method has the superiority over the resolution-based method, i.e., it has more formalizations, more transformation rules, more computation paths, and more solvability than the resolution-based method.

1.6. **Rule generation and program synthesis.** This gives a strong motivation of making a new proof method by inventing new ET rules. Automatic rule generation is also important. The main purpose of rule generation is to develop a theory of program synthesis, where a program is a set of ET rules [10, 11, 12]. ET-based program synthesis has more general foundation than the theories of program synthesis in logic programming, where a program is regarded as a logical formula [13, 14].

A typical program synthesis method is called the squeeze method [15], where ET computation and rule generation are combined into one computation. ET rule generation from a logical formula, called a logical equivalence, has been introduced [16, 17], and ET-based bidirectional search was developed for proving correctness of a logical equivalence [18].

1.7. **Organization.** The rest of the paper is organized as follows. Section 2 introduces the resolution-based method and the ET-based method. It shows that the latter is the extension of the former. Section 3 introduces a proof problem, called a pal-pal proof problem, and its formalization (called the unsatisfiability-based formalization). Using the pal-pal proof problem, it explains the limitation of the conventional proof method, which consists of CSD and resolution-based inference. Section 4 defines a transformational approach to logic and logical computation and introduces a new formalization (called the existence-finding formalization) of the pal-pal proof problem. Section 5 shows that the pal-pal proof problem cannot be solved by unfolding solely. Section 6 shows that the

pal-pal proof problem can be solved by using many specialized ET rules, where a multi-head ET rule plays indispensable roles. Section 7 compares the resolution-based solution method and the ET-based solution method, and shows the superiority of the latter one over the former one. Section 8 provides conclusions.

1.8. **Notation.** The notation that follows holds thereafter. Given a set A, $pow(A)$ denotes the power set of A. Given a partial mapping f from a set A to a set B, $dom(f)$ denotes the domain of f, i.e., $dom(f) = \{a \mid (a \in A)\ \&\ (f(a) \text{ is defined})\}$. Assuming that $t_1, t_2, \ldots, t_n$, and s are terms, the following list notation is used. A list $[\,]$ represents a constant nil. $[t_1, t_2, \ldots, t_n|s]$ is a list representing $cons(t_1, cons(t_2, cons(t_3, \ldots cons(t_n, s)\ldots)))$. $[t_1, t_2, \ldots, t_n]$ is a list representing $cons(t_1, cons(t_2, cons(t_3, \ldots cons(t_n, nil)\ldots)))$.

2. Resolution-Based Method vs ET-Based Method.

We introduce the ET-based method and show that the ET-based method is the extension of the resolution-based method. Superiority of the ET-based method will be discussed in the subsequent sections.

2.1. **ET-based method.** The ET-based method is a solution method of repeated application of equivalent transformation rules (ET rules). In the following, a conjunction $K \wedge Cs$ of a first-order formula K and a clause set Cs is considered by regarding a clause set as a first-order formula. K represents background knowledge and Cs is used as a changing part. We take first-order formulas with Herbrand semantics.

A set G of ground user-defined atoms is a model of a closed formula F iff F is true with respect to G, i.e., F is true if all ground atoms in G are true and all ground atoms outside G are false. We formalize a proof problem by a pair of a first-order formula $K \wedge Cs$ and a mapping φ, where K is a first-order formula and Cs is a set of clauses, and solve the problem by calculating

$$\varphi\left(\bigcap Models(K \wedge Cs)\right).$$

Note that a model of a first-order formula in this paper is a set of ground user-defined atoms, which is a subset of the set of all ground user-defined atoms, denoted by $\mathcal{G}_{\mathrm{u}}$. Hence $Models(K \wedge Cs)$ is a subset of the power set of $\mathcal{G}_{\mathrm{u}}$, and its intersection is well-defined. The mapping φ is called an *extraction mapping*. A problem of this type, consisting of a first-order formula and an extraction mapping, is called a *model-intersection (MI) problem* [5, 19].

To calculate $\varphi\left(\bigcap Models(K \wedge Cs)\right)$, we find a sequence of clause sets starting from Cs $(= Cs_1)$:

$$Cs_1 \to Cs_2 \to \cdots \to Cs_n.$$

This sequence is obtained by equivalent transformation (ET) rules with respect to φ and K. A rewriting rule r is an ET rule with respect to φ and K iff for any clause sets Cs and Cs', if the result of the application of r to Cs is equal to Cs' (i.e., $r(Cs) = Cs'$), then $\varphi\left(\bigcap Models(K \wedge Cs)\right) = \varphi\left(\bigcap Models(K \wedge Cs')\right)$. An answer is obtained from $K \wedge Cs_n$ by application of a partial mapping, called an *answer mapping*.

2.2. **Generality of the ET-based approach.** The ET-based method can produce many solution methods. In this paper, we solve proof problems with the following two solution types. Assume that F and E are first-order formulas, and Cs_i $(i = 1, 2, \ldots)$ is a set of clauses. We start with first-order formalization $F \wedge E$ of a logical problem.

2.2.1. *Solutions of Type A.* We convert $F \wedge E$ to Cs_1 using CSD and construct a sequence

$$Cs_1 \to Cs_2 \to \cdots \to Cs_n.$$

The conventional proof method based on resolution is of this type with limited use of ET rules (inference rules). However, when $F \wedge E$ contains built-in constraint atoms, this transformation often does not preserve satisfiability and we cannot use CSD for correct computation.

2.2.2. *Solutions of Type B.* We use conversion with $F = K$ and $\mathrm{CSD}(E) = Cs_1$, and construct a sequence

$$\langle K, Cs_1\rangle \to \langle K, Cs_2\rangle \to \cdots \to \langle K, Cs_n\rangle.$$

This is more general than Type A in the sense that we have F that can be used with no application of CSD. Formal various ET rules with respect to K are useful to search for a solution path. When E does not contain built-in constraint atoms, this transformation preserves the satisfiability of E.

2.2.3. *Comparison of A with B.* Consider the formula K in solutions of Type B. If we remove K by letting K be a true atom $\top$, we have solution methods of Type A. Hence we regard that Type A is included by Type B. We will use explanation in Type A in Section 3, and explanation in Type B from Section 4 onwards. The resolution method is explained in Type A, and the ET-based method is typically discussed in Type B. By a structural consideration, it follows that the ET-based method has

- more formalizations than the resolution-based method,
- more transformation rules than the resolution-based method, and
- more computation paths than the resolution-based method.

CSD is applied to $F \wedge E$ in a solution method of Type A, while it is applied only to E in that of Type B. Such a restriction often allows a method of Type B to avoid suffering from ill-transformation by CSD. The resolution method restricts ET rules to only resolution and factoring, while the ET-based method tries to solve problems by adoption of all useful ET rules. More problems can be solved by increasing ET rules in Type B (Section 6), which will overcome the limitation of the resolution method.

2.3. **Our method.** We will show that

- the resolution-based method is incomplete to solve all proof problems on $\mathrm{FOL_c}$, and
- the ET-based method overcomes the limitation of the resolution-based method.

To reach the goal, we use a proof problem, called a pal-pal proof problem (Section 3.1).

- We show that the pal-pal proof problem cannot be solved by the resolution method (Section 3), although this problem is represented by using first-order formulas.
- We show that the pal-pal problem can be solved by the ET-based method (Section 6).
- We show that the pal-pal problem cannot be solved only by simpler rules, such as unfolding (Section 5).

3. Limitations of the Conventional Proof Method.

We introduce a proof problem and show that this problem cannot be solved by CSD and resolution method. Non-preservation of satisfiability by CSD destroys the correctness of computation. We argue that the widely-spread belief that all proof problems on the first-order formulas can be solved by resolution with some appropriate control strategy is incorrect.

3.1. **A pal-pal proof problem and its formalization.** Assume as background knowledge a set consisting of the three if-and-only-if formulas in Figure 1, where *pal*, *rev*, *eq*, and *app* stand for "palindrome", "reverse", "equal", and "append", respectively. Consider the problem "prove that there are no ground terms s and t such that two lists $[1, s|t]$ and $[2, s|t]$ are palindromes", which is called the pal-pal proof problem.

$$
\begin{array}{ll}
(1) & \forall x : pal(x) \leftrightarrow rev(x, x). \\
(2) & \forall x, \forall y : rev(x, y) \leftrightarrow \\
 & \quad (eq(x, [\,]) \wedge eq(y, [\,]) \ \vee \\
 & \quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge rev(w, u) \wedge app(u, [a], y))). \\
(3) & \forall x, \forall y, \forall z : app(x, y, z) \leftrightarrow \\
 & \quad (eq(x, [\,]) \wedge eq(y, z) \ \vee \\
 & \quad \exists a, \exists w, \exists u : (eq(x, [a|w]) \wedge eq(z, [a|u]) \wedge app(w, y, u))).
\end{array}
$$

FIGURE 1. If-and-only-if formulas defining the predicates *pal*, *rev*, and *app*

Let F_0 be the conjunction of the three if-and-only-if formulas in Figure 1, and E_0 the following formula:

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

The pal-pal proof problem to prove $F_0 \models \neg E_0$ can be formalized as $Models(\neg(F_0 \rightarrow \neg E_0)) = \emptyset$, which is equivalent to $Models(F_0 \wedge E_0) = \emptyset$. This is a commonly used formalization in the conventional proof theory, and is called the *unsatisfiability-based formalization* in this paper.

3.2. **Incompleteness of Skolemization-based proof methods.** It was reported in [8] that the conventional Skolemization often does not preserve the satisfiability of a first-order formula that includes built-in constraint atoms. Note that F_0 includes built-in constraint atoms, i.e., *eq*-atoms. The satisfiability of F_0 is not preserved by CSD. We prove below that F_0 has a model, while $\mathrm{CSD}(F_0)$ has no model, i.e.,

- $Models(F_0) \neq \emptyset$, and
- $Models(\mathrm{CSD}(F_0)) = \emptyset$.

Let D_0 be the set consisting of the definite clauses in Figure 2. The former is proved to be true since F_0 has a model as follows. Let T_{D_0} denote the immediate consequence operator associated with D_0 [2]. Since F_0 is the completion of D_0, the least fixpoint of T_{D_0} is a model of F_0. The latter is proved to be true since $\mathrm{CSD}(F_0)$ is inconsistent as follows: Since the clause set $\mathrm{CSD}(F_0)$ contains all definite clauses in D_0, $rev([1], [1])$ is true. However, $\mathrm{CSD}(F_0)$ also contains the following clause, where f_0 and f_1 are function symbols:

$$eq(A, [\,]), eq(A, [f_0(B, A)|f_1(B, A)]) \leftarrow rev(A, B).$$

When the above clause is specialized by $A = B = [1]$, the two atoms $eq([1], [\,])$ and $eq([1], [f_0([1], [1])|f_1([1], [1])])$ are obviously false. Then we have

$$\leftarrow rev([1], [1]).$$

This means $rev([1], [1])$ is false, which is a contradiction.

Since the conventional Skolemization does not preserve satisfiability, formula decomposition that includes conventional Skolemization steps is not general enough to solve all proof problems on first-order logic.

$$
\begin{array}{ll}
(1) & pal(X) \leftarrow rev(X, X) \\
(2) & rev([\,], [\,]) \leftarrow \\
(3) & rev([A|X], Y) \leftarrow rev(X, R), app(R, [A], Y) \\
(4) & app([\,], X, X) \leftarrow \\
(5) & app([A|X], Y, [A|Z]) \leftarrow app(X, Y, Z)
\end{array}
$$

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

4. **A Transformational Approach.** We introduce a new formalization of the pal-pal proof problem, which is called the existence-finding formalization. We explain a transformational approach to logic and logical computation by showing the general framework of ET-based problem solving, computation power of which is evaluated by a "finitely solvable area".

4.1. **Formalization as a model-intersection problem.** Since the pal-pal proof problem says that "prove that there are no ground terms s and t such that $[1, s|t]$ and $[2, s|t]$ are palindromes", we introduce a formula ($E_0 \rightarrow$ "exists"), where E_0 is the following formula:

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

Letting $E_1 = (E_0 \rightarrow$ "exists") and referring to F_0 in Section 3, we consider $\bigcap Models(F_0 \wedge E_1)$. This set results in the union of two sets:

$$\bigcap Models(F_0 \wedge E_1) = \bigcap Models(F_0) \cup G_{ans},$$

where

$$G_{ans} = \begin{cases} \{\text{"exists"}\} & \text{if there are ground terms } s \text{ and } t \text{ such that} \\ & [1, s|t] \text{ and } [2, s|t] \text{ are palindromes,} \\ \emptyset & \text{otherwise.} \end{cases}$$

Then the answer of the *pal-pal* problem is formalized as

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

where φ_1 is a mapping defined by the following.

Definition 4.1. *φ_1 is a mapping from $pow(\mathcal{G}_{\mathrm{u}})$ to $\{$"yes", "no"$\}$ such that for each $G \subseteq \mathcal{G}_{\mathrm{u}}$,*

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

The *pal-pal* problem is formalized as a model-intersection (MI) problem $\langle F_0 \wedge E_1, \varphi_1 \rangle$. The formalization with the extraction mapping φ_1 is called an *existence-finding formalization*.

4.2. **ET-based problem solving and its correctness.** Let a logical problem formalized by $\varphi(\bigcap Models(F \wedge E))$ be given. We solve it by successive transformation by using rules as shown in Section 2.2.

A rewriting rule is a partial mapping on the set of all first-order formulas.

Definition 4.2. *Let φ be an extraction mapping and K a first-order formula. A rewriting rule r is an equivalent transformation rule (ET rule) with respect to φ and K iff for any first-order formulas Cs and Cs', if $r(Cs) = Cs'$, then $\varphi(\bigcap Models(K \wedge Cs)) = \varphi(\bigcap Models(K \wedge Cs'))$.*

Definition 4.3. *A partial mapping ans is an answer mapping for an extraction mapping φ iff for any first-order formulas K and Cs,*

$$ans(K \wedge Cs, \varphi) = \varphi \left(\bigcap Models(K \wedge Cs) \right)$$

if it is defined.

Theorem 4.1. *Let φ be an extraction mapping and K a first-order formula. Let F and E be first-order formulas. If ans is an answer mapping for φ, R is a set of ET rules with respect to φ and K, and there is a sequence of pairs of first-order formulas*

$$\langle K, Cs_1\rangle \to \langle K, Cs_2\rangle \to \cdots \to \langle K, Cs_n\rangle,$$

such that $\varphi\left(\bigcap Models(F \wedge E)\right) = \varphi\left(\bigcap Models(K \wedge Cs_1)\right)$, $r_i \in R$ and $r_i(Cs_i) = Cs_{i+1}$ for any $i \in \{1, 2, \ldots, n-1\}$, and $\langle K \wedge Cs_n, \varphi\rangle \in dom(ans)$, then

$$\varphi\left(\bigcap Models(F \wedge E)\right) = ans(K \wedge Cs_n, \varphi).$$

Proof: The first condition is $\varphi\left(\bigcap Models(F \wedge E)\right) = \varphi\left(\bigcap Models(K \wedge Cs_1)\right)$. Since r_i is an ET rule, we have $\varphi\left(\bigcap Models(K \wedge Cs_i)\right) = \varphi\left(\bigcap Models(K \wedge Cs_{i+1})\right)$ for any $i \in \{1, 2, \ldots, n-1\}$. Since *ans* is an answer mapping for φ and $\langle K \wedge Cs_n, \varphi\rangle \in dom(ans)$, it holds that $ans(K \wedge Cs_n, \varphi) = \varphi\left(\bigcap Models(K \wedge Cs_n)\right)$. From these, we have $\varphi\left(\bigcap Models(F \wedge E)\right) = ans(K \wedge Cs_n, \varphi)$. □

4.3. **An answer mapping.** We define a partial mapping ans_1 for φ_1 as follows: Let $\mathrm{CLS_B}$ be the set of all clauses consisting of user-defined atoms and built-in constraint atoms.

Definition 4.4. *Let ans_1 be a partial mapping to assign "yes" or "no" to each problem that is defined by the following: for any first-order formula K and any clause set $Cs \subseteq \mathrm{CLS_B}$,*

$$ans_1(K \wedge Cs, \varphi_1) = \begin{cases} \text{"yes"} & \text{if } Cs \text{ is the empty set,} \\ \text{"no"} & \text{if } Cs \text{ contains a unit clause ("exists" } \leftarrow), \end{cases}$$

and it is undefined otherwise.

The partial mapping ans_1 is an answer mapping for φ_1, i.e.,

$$ans_1(K \wedge Cs, \varphi_1) = \varphi_1\left(\bigcap Models(K \wedge Cs)\right),$$

a proof of which is as follows: $ans_1(K \wedge Cs, \varphi)$ is defined only when $\varphi = \varphi_1$. Consider the following two cases.

- Case 1: $Cs = \emptyset$. Then $\bigcap Models(K \wedge Cs) - \bigcap Models(K) = \emptyset$. Hence we have $\varphi_1\left(\bigcap Models(K \wedge Cs)\right) =$ "yes" $= ans_1(K \wedge Cs, \varphi_1)$.
- Case 2: $Cs \ni$ ("exists" $\leftarrow$). It follows that $\bigcap Models(K \wedge Cs) - \bigcap Models(K) = \{$"exists"$\}$. Hence we have $\varphi_1\left(\bigcap Models(K \wedge Cs)\right) =$ "no" $= ans_1(K \wedge Cs, \varphi_1)$.

4.4. **Evaluation by a finitely solvable area.** A problem *prob* is solved when a sequence of problems starting from *prob* reaches the domain of a given answer mapping. A control selects an ET rule that is applied to a problem at each step of computation to determine a sequence of problems. For any set R of ET rules, any control *ctrl*, and any set A of answer mappings, we define $FS(R, ctrl, A)$ as the set of all problems that are successfully solved by using ET rules in R, the control *ctrl*, and answer mappings in A, where *FS* stands for "finite solvability". A solver is evaluated in the ET-method by the set of all problems that can be solved in finite steps, which is called a finitely solvable area (FSA). When we have more answer mappings and more ET rules, an FSA increases.

Consider the set of all problems that can be solved successfully by some control, using a set R of ET rules and a set A of answer mappings, which is denoted by $FSA(R, A)$ and is called a finitely solvable area with respect to R and A, i.e.,

$$FSA(R, A) = \bigcup\{FS(R, ctrl, A) \mid ctrl \in CTRL\},$$

where $CTRL$ is the set of all controls. Given a problem $prob$, we try to solve $prob$, i.e., we try to find R and A such that

$$prob \in FSA(R, A).$$

Maximization of $FSA(R, A)$ by increasing R and A is of central importance in the research of logical computation.

5. **Failure with General Unfolding.** We show that by using only the unfolding rule, the pal-pal problem is transformed infinitely and cannot reach the final state with the existence-finding formalization.

5.1. **Failure with unfolding only transformation.** Let E_1 be the formula ($E_0 \rightarrow$ "exists"), where E_0 is given in Section 4.1. A definite clause whose head is an atom "exists" is called an "exists" clause. Let C_1 be the "exists" clause ("exists" $\leftarrow pal([1, A|X])$, $pal([2, A|X])$). Let Cs_1 be the singleton set $\{C_1\}$. Then E_1 is equivalent to Cs_1, i.e., $Models(E_1) = Models(Cs_1)$.

Refer to F_0 and D_0 in Section 3. The formula F_0 is the completion of D_0. We consider the unfolding rule with respect to D_0, and let $R = \{unfolding\}$. Note that the pal-pal problem is transformed by using the unfolding rule with respect to D_0, which is determined by F_0. From Section 4 onwards, we use a solution method of Type B (cf. Section 2.2), where transformation is done by ET rules with respect to φ_1 and F_0.

In this section, we show that R cannot solve the pal-pal problem by the existence-finding formalization and $\{ans_1\}$, i.e.,

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \notin FSA(\{unfolding\}, \{ans_1\}).$$

This means that the pal-pal problem cannot reach the domain of ans_1 by using only the unfolding rule with respect to D_0.

5.2. **Infinite computation.** By unfolding, C_1 is typically transformed sequentially from C_1 to C_5 in Figure 3, where the body atoms selected for unfolding are underlined. Depending on the selection of body atoms, we have many possible sequences of clause sets. We will prove in Section 5.3 that unfolding by D_0 repeats forever and no answer is obtained.

C_1 : "exists" $\leftarrow \underline{pal([1, A|X])}, pal([2, A|X])$.
C_2 : "exists" $\leftarrow \overline{rev([1, A|X], [1, A|X])}, \underline{pal([2, A|X])}$.
C_3 : "exists" $\leftarrow \underline{rev([1, A|X], [1, A|X])}, \overline{rev([2, A|X], [2, A|X])}$.
C_4 : "exists" $\leftarrow \overline{rev([A|X], B), app(B, [1], [1, A|X])}, \underline{rev([2, A|X], [2, A|X])}$.
C_5 : "exists" $\leftarrow rev([A|X], B), app(B, [1], [1, A|X]), \overline{rev([A|X], C), app(C, [2], [2, A|X])}$.

FIGURE 3. A typical transformation sequence starting from C_1

We introduce D' as a subset of D_0 consisting of the following three clauses:
$C_p : (pal(X) \leftarrow rev(X, X))$,
$C_r : (rev([A|X], Y) \leftarrow rev(X, R), app(R, [A], Y))$,
$C_a : (app([A|X], Y, [A|Z]) \leftarrow app(X, Y, Z))$.

Referring to D', we prove the following proposition for preparation of the theorem in Section 5.3.

Proposition 5.1. *Let $C =$ ("exists" $\leftarrow pal([1, A|X]), pal([2, A|X]))$. Let Cs'_n be the result of n-times application of unfolding with respect to D' to C. Then Cs'_n is not an empty set, and all clauses in Cs'_n are "exists" clauses with non-empty bodies.*

Proof: Let C' be a clause in Cs'_n. Since unification occurring in unfolding with C_r or C_a is always successful, C' is never removed by only unfolding. Since unfolding with D' does not decrease the number of atoms in the body, C' has non-empty body. Since C is an "exists" clause, C' is an "exists" clause. Hence Cs'_n is not an empty set, and all clauses in Cs'_n are "exists" clauses with non-empty bodies. □

5.3. **Unreachability and unsolvability.** Unfolding is a major transformation rule for solving proof problems. However, we show that the pal-pal proof problem can never reach the domain of ans_1 by unfolding with respect to D_0 alone.

Proposition 5.2. *Let $C = (\text{"exists"} \leftarrow pal([1, A|X]), pal([2, A|X]))$. Let Cs_n be the result of n-times application of unfolding with respect to D_0 to C. Then Cs_n is not an empty set, and all clauses in Cs_n are "exists" clauses with non-empty bodies.*

Proof: Refer to D' in Section 5.2. There is Cs'_n such that Cs'_n is the result of n-times application of unfolding with respect to D' to C, and $Cs'_n \subseteq Cs_n$. By Proposition 5.1, Cs'_n is not an empty set, and all clauses in Cs'_n are "exists" clauses with non-empty bodies. Hence Cs_n is not an empty set. If there is a clause in Cs_n that has an empty body, there are ground terms s and t such that $[1, s|t]$ and $[2, s|t]$ are palindromes. Hence there is no clause in Cs_n that has an empty body. It follows that Cs_n is not an empty set, and all clauses in Cs_n are "exists" clauses with non-empty bodies. □

Let *finalProb* be a partial mapping that determines a final problem P_n from an initial state P_0, a rule set R, and a control *ctrl*. Unfolding with respect to D_0 is denoted by $unfolding(D_0)$.

Theorem 5.1. *Let $C = (\text{"exists"} \leftarrow pal([1, A|X]), pal([2, A|X]))$. Let $P_0 = D_0 \cup \{C\}$. For any control ctrl, there is no $P_n \in dom(ans_1)$ such that*

$$P_n = finalProb(P_0, \{unfolding(D_0)\}, ctrl).$$

Proof: Let Cs_n be the result of n-times application of unfolding with respect to D_0 to C. By Proposition 5.2, Cs_n is not an empty set, and all clauses in Cs_n are "exists" clauses with non-empty bodies. Hence $P_n = D_0 \cup Cs_n \notin dom(ans_1)$. □

From Theorem 5.1, we have

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \notin FSA(\{unfolding(D_0)\}, \{ans_1\}).$$

6. **Solutions with Specialized ET Rules.** We show that the pal-pal proof problem can be solved by using many specialized ET rules. Among these rules, a multi-head ET rule plays indispensable roles.

6.1. **Solutions with specialized/multi-head ET rules.** As shown in Section 5, the pal-pal problem cannot be solved by unfolding alone. What rule should be used together with the unfolding rule to solve the pal-pal problem? It is a two-head ET rule, called r_{rev_2}, which will be introduced in Section 6.2.

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \in FSA(\{unfolding, r_{rev_2}\}, \{ans_1\}).$$

Furthermore, we show in Section 6.4 that the pal-pal problem can be solved by using six specialized unfolding rules and the two-head rule r_{rev_2}:

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \in FSA(R_1, \{ans_1\}),$$

where R_1 is the set consisting of the rules in Figure 4.

6.2. **A solution by using a two-head ET rule.** Given an answer mapping, one typical improvement of finite solvability is obtained by creation and accumulation of ET rules. After the transformation from C_1 to C_5 in Figure 3, the clause

C_5 : "exists" $\leftarrow \underline{rev([A|X], B)}, app(B, [1], [1, A|X]), \underline{rev([A|X], C)}, app(C, [2], [2, A|X])$.

has two rev atoms, $rev([A|X], B)$ and $rev([A|X], C)$. Since the first arguments of the two rev atoms are the same, the second arguments are also the same, i.e., B and C can be unified. More precisely, two rev atoms, $rev(t, t_1)$ and $rev(t, t_2)$, are found, and the second rev atom can be removed after specialization by $t_1 = t_2$. This transformation is obtained by an ET rule r_{rev_2}, which is represented by

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

Let us observe further transformation. By application of r_{rev_2}, we have C_6

C_6 : "exists" $\leftarrow rev([A|X], B), \underline{app(B, [1], [1, A|X])}, app(B, [2], [2, A|X])$.

Unfolding by selecting $app(B, [1], [1, A|X])$ gives C_7

C_7 : "exists" $\leftarrow rev([A|X], [1|Y]), app(Y, [1], [A|X]), \underline{app([1|Y], [2], [2, A|X])}$.

Unfolding by selecting $app([1|Y], [2], [2, A|X])$ gives no clause. So the resulting clause set contains no "exists" clause, and thus, the resulting MI problem enters the domain of ans_1. Hence, the pal-pal problem can be solved by using two ET rules, unfolding and r_{rev_2}:

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \in FSA(\{unfolding, r_{rev_2}\}, \{ans_1\}).$$

r_{pal}: $pal(X) \Rightarrow rev(X, X)$.
r_{rev_0}: $rev([\,], X) \Rightarrow \{=(X, [\,])\}$.
r_{rev_1}: $rev([A|X], Y) \Rightarrow rev(X, V), app(V, [A], Y)$.
r_{app_0}: $app([\,], Y, Z) \Rightarrow \{=(Y, Z)\}$.
r_{app_1}: $app([A|X], Y, Z) \Rightarrow \{=(Z, [A|W])\}, app(W, Y, Z)$.
r_{app_2}: $app(X, Y, [A|Z]) \Rightarrow \{=(X, [\,]), =(Y, [A|Z])\}$;
$\Rightarrow \{=(X, [A|V])\}, app(V, Y, Z)$.
r_{rev_2}: $rev(X, Y), rev(X, Z) \Rightarrow \{=(Y, Z)\}, rev(X, Y)$.

FIGURE 4. Examples of rewriting rules

6.3. **Specialized unfolding.** Unfolding is the most important transformation used in the ET-based method, while resolution is mainly used in conventional proof methods. Unfolding and resolution have been often confused, since the operation of unfolding is similar to making a group of resolvents at a time. However, they are fundamentally different. From a clause C, unfolding produces child clauses, and the clause C is removed, while resolution produces one clause without removing the original clause C. When we solve QA problems, clause-increasing transformation by resolution should be basically avoided since it often causes explosion of clauses without solving the original problem.

Let $\mathcal{S}$ be the set of all substitutions. Given a user-defined atom B and a definite-clause set D in $pow(\mathrm{CLS_B})$, we define specialized unfolding with respect to B and D as follows:

$$\begin{aligned}\textsc{Unfold}(B, D) = \{\langle Cs, Cs' \rangle \mid\ & (Cs \subseteq \mathrm{CLS_B})\ \&\\ & (C \text{ is a clause in } (Cs - D))\ \&\\ & (occ \text{ is an occurrence of a body atom } b \text{ in } C)\ \&\\ & (b = B\theta)\ \&\ (\theta \in \mathcal{S})\ \&\\ & (Cs' = (Cs - \{C\}) \cup \textsc{Resol}(C, D, occ, b))\}.\end{aligned}$$

The atom B is used to restrict the range of a selected atom b by $b = B\theta$ for some substitution $\theta \in \mathcal{S}$. B is specified from the viewpoint of control [20]. D is determined as the set of all definite clauses in Cs whose head atoms are unifiable with B. $\textsc{Resol}(C, D, occ, b)$ is the set of child clauses (resolvents) each of which is produced by resolution of C and

some definite clause C' in D, using the unification of a selected atom b at the occurrence *occ* in C with the head of C'.

6.4. **Computing with specialized ET rules.** Refer to D_0 in Section 3. In Figure 4, we have one two-head rule r_{rev_2} along with six specialized ET rules, which are specialized unfolding rules as follows:

$r_{pal} = \textsc{Unfold}(pal(X), D_0)$,
$r_{rev_0} = \textsc{Unfold}(rev([\,], X), D_0)$,
$r_{rev_1} = \textsc{Unfold}(rev([A|X], Y), D_0)$,
$r_{app_0} = \textsc{Unfold}(app([\,], Y, Z), D_0)$,
$r_{app_1} = \textsc{Unfold}(app([A|X], Y, Z), D_0)$,
$r_{app_2} = \textsc{Unfold}(app(X, Y, [A|Z]), D_0)$.

These six rules can be used in place of unfolding in Section 6.2. By using these six rules together with r_{rev_2}, we have a sequence of sets of clauses:

$$\{C_1\} \to \{C_2\} \to \{C_3\} \to \{C_4\} \to \{C_5\} \overset{r_{rev_2}}{\longrightarrow} \{C_6\} \to \{C_7\} \to \{\,\}.$$

Hence, the pal-pal problem is solved by using the six specialized unfolding rules and the two-head rule r_{rev_2}:

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \in FSA(R_1, \{ans_1\}),$$

where $R_1 = \{r_{pal}, r_{rev_0}, r_{rev_1}, r_{app_0}, r_{app_1}, r_{app_2}, r_{rev_2}\}$.

7. **Superiority of the ET-Based Solution Method.** We compare the resolution-based solution method and the ET-based solution method, and show the superiority of the latter one over the former one.

7.1. **Failure of the resolution method.** Limitations of resolution method (together with CSD) have been shown in Section 3. CSD often does not preserve satisfiability if an input formula contains built-in constraint atoms. Consider the first-order formulas F_0 and E_0 in Section 3.1. The pal-pal proof problem to prove $F_0 \models \neg E_0$ can be formalized as $Models(F_0 \wedge E_0) = \emptyset$. CSD fails to transform $F_0 \wedge E_0$ into clauses correctly (Section 3). Hence the resolution method together with CSD is incomplete for the proof problems on FOL_c, which is a sharp contrast to the wide-spread belief that all proof problems on first-order formulas can be solved by using the resolution inference rule. This view comes invalidly from the completeness theorem of SLD resolution (Theorem 8.6 in [2]) due to the non-preservation of satisfiability by CSD. We need to be aware of the incompleteness of the resolution method and should invent a complete proof procedure for the proof problems on FOL_c.

7.2. **Success of the ET-based solution method.** The ET-based method has more formalizations than the resolution-based method. We introduced two MI formalizations for the pal-pal proof problem:

- unsatisfiability-based formalization, and
- existence-finding formalization.

The former corresponds to the conventional satisfiability-based solution, while the latter is different from the conventional satisfiability checking.

We took the existence-finding formalization and tried to solve the pal-pal proof problem. We had two results of failure and success using different rule sets as follows:

- Failure of only unfolding (Section 5)

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \notin FSA(\{unfolding\}, \{ans_1\}).$$

- Success by using multi-head rules (Section 6.2)

$$\text{Pal-pal proof problem} = \langle F_0 \wedge Cs_1, \varphi_1 \rangle \in FSA(\{unfolding, r_{rev_2}\}, \{ans_1\}).$$

7.3. **Superiority of the ET-based method.** The ET-based method can take any ET rules. There is no limitation to a single rule such as the resolution rule. The ET-based method can use more transformation rules than the resolution-based method. It provides more computation paths and more successful computation paths. Hence we conclude the superiority of the ET-based solution method over the resolution-based method together with CSD, i.e., the ET-based method has more formalizations, more transformation rules, more computation paths, and more solvability than the resolution-based method.

8. **Concluding Remarks.** We proposed an ET-based method for solving proof problems. We formalize a proof problem as an MI problem, using a formula and an extraction mapping. The ET-based method admits many formalizations. Unsatisfiability-based formalization and existence-finding formalization are tried for the pal-pal proof problem in this paper. The class of MI problems can be considered as a superset of the class of proof problems and that of QA problems.

In the ET-based method, an unlimited number of ET rules, including specialized unfolding-based rules, are used. We solve MI problems by using ET rules, which is a sharp contrast to the conventional proof-centered logical problem solving by using inference rules. General inference rules, such as resolution, are usually used in the conventional methods.

For each resolution-based proof, there is an ET-based proof corresponding to it. The ET-based method has more transformation rules and more computation paths than the resolution-based method. It was proved that the pal-pal problem cannot be solved by the resolution method (using unsatisfiability-based formalization), but can be solved by the ET-based method (using existence-finding formalization). It was thus shown that the ET-based method has larger finite solvability of logical problems than the conventional resolution method. We conclude the superiority of the ET-based solution method over the resolution-based solution method.

The ET-based method is useful to overcome the limitation of resolution-based proof procedures. The ET-based method can produce many solution methods that are guaranteed to be correct. By the research of various ET rules, more logical problems will be solved practically, compared to the logical computation by a limited number of inference rules in the conventional methods [19, 20, 21, 22, 23].

Acknowledgments. This work was partially supported by (i) JSPS KAKENHI Grant Numbers 25280078 and 26540110, (ii) the Center of Excellence in Intelligent Informatics, Speech and Language Technology and Service Innovation (CILS), Thammasat University, and (iii) the Intelligent Informatics and Service Innovation (IISI) Center, Sirindhorn International Institute of Technology (SIIT), Thammasat University.

REFERENCES

[1] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.
[2] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[3] C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
[4] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.

[5] 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.
[6] R. A. Kowalski, Predicate logic as a programming language, *Proc. of the 6th IFIP Congress 1974*, Stockholm, Sweden, pp.569-574, 1974.
[7] R. A. Kowalski, Algorithm = logic + control, *Communications of the ACM*, vol.22, pp.424-435, 1979.
[8] K. Akama and E. Nantajeewarawat, Meaning-preserving Skolemization, *Proc. of the 3rd International Conference on Knowledge Engineering and Ontology Development*, Paris, France, pp.322-327, 2011.
[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, H. Koike and H. Mabuchi, A theoretical foundation of program synthesis by equivalent transformation, *Perspectives of System Informatics, Lecture Notes in Computer Science*, vol.2244, Springer Verlag, Heidelberg, pp.131-139, 2001.
[11] 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.
[12] K. Akama and E. Nantajeewarawat, Computing logically with generation of equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.1, pp.315-330, 2022.
[13] Y. Shigeta, K. Akama, H. Mabuchi and H. Koike, Converting constraint handling rules into equivalent transformation rules, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol.10, no.3, pp.339-348, 2006.
[14] T. Frühwirth, Constraint handling rules – What else?, *arXiv Preprint*, arXiv: 1701.02668v1, 2017.
[15] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, *Lecture Notes in Computer Science*, vol.4378, pp.41-54, 2007.
[16] 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.
[17] 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.
[18] 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.
[19] K. Akama and E. Nantajeewarawat, Model-intersection problems with existentially quantified function variables: Formalization and a solution schema, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, vol.2, Porto, Portugal, pp.52-63, 2016.
[20] K. Akama, E. Nantajeewarawat and T. Akama, Computation control by prioritized ET rules, *Proc. of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018)*, vol.2, Seville, Spain, pp.84-95, 2018.
[21] K. Akama and E. Nantajeewarawat, Solving query-answering problems with constraints for function variables, *Proc. of the 10th Asian Conference on Intelligent Information and Database Systems*, Dong Hoi City, Vietnam, pp.36-47, 2018.
[22] K. Akama and E. Nantajeewarawat, Unfolding existentially quantified sets of extended clauses, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, vol.2, Porto, Portugal, pp.96-103, 2016.
[23] K. Akama, E. Nantajeewarawat and T. Akama, Logical problem solving framework, *Proc. of the 11th Asian Conference on Intelligent Information and Database Systems*, Yogyakarta, Indonesia, pp.28-40, 2019.

Author Biography

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

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

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

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