International Journal of Innovative Computing, Information and Control
Volume **9**, Number **11**, November **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 4419–4430

PROOF OF UNSATISFIABILITY OF ATOM SETS BASED ON COMPUTATION BY EQUIVALENT TRANSFORMATION RULES

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

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

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

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

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

Received December 2012; revised April 2013

Abstract. *Equivalent Transformation (ET) rules can be used to construct correct sequential and parallel programs, as they are inherently correct. One important method for making ET rules uses logical formulas. Among logical formulas, unsatisfiable conjunctions of finite atoms, each of which is represented by an atom set in this paper, are especially useful for making many ET rules that are used for constructing efficient programs. This paper proposes a method of proving unsatisfiability of an atom set based on computation by ET rules. The proposed method is recursive, in the sense that, during proving the given atom set, it may find a new atom set based on subset relation or inductive structure. Since our proposed method incorporates search based on subset and inductive relations, it can be used to prove unsatisfiability of various atom sets that cannot be proven by conventional methods.*

Keywords: Unsatisfiability, Proof, Equivalent transformation rule, Induction

1. Introduction.

Equivalent Transformation (ET) rules are useful for developing algorithms for sequential program construction [1, 9]. An ET rule is a procedure that is used to replace one clause set with another while preserving the original clause's declarative meaning. In recent years, methodologies for constructing parallel programs, which extend the methodologies used to construct sequential programs, have been proposed [3]. In the ET program construction, correct programs are constructed by accumulating ET rules made from specifications, sets of logical formulas, one by one. Methods for automatically making ET rules are therefore useful for constructing correct programs.

Miura et al. [14] recently proposed a method for making ET rules from logical formulas (each called a Logical Equivalence (LE)) made from specifications. An LE describes an equivalence relationship between two logical formulas under some specified preconditions.

Since ET rules can be made from LEs using this method, methods by which correct logical formulas can be made from specifications are therefore essential.

One objective of this paper is to propose a solution for making LEs from specifications. We achieve this by proposing a new method for proving atom sets based on computation by ET rules and constructing an algorithm based on the proposed method. An atom is a logical formula that has no deeper propositional structure. Since an atom conjunction is treated as an atom set in this paper, it is called an atom set.

In this paper we specially focus on the proof of unsatisfiable atom sets: that is, atom sets in which one or more atoms do not satisfy the constraint of predicates. An LE made from such an atom set is efficient for making *False rules*, which are useful ET rules for constructing efficient programs. A False rule removes any definite clause that contains body atoms which do not satisfy the constraints of predicates.

In proposing a method for proving the unsatisfiability of atom sets, this paper

1. Defines the concept of an unsatisfiable atom set with respect to background knowledge,
2. Gives a sufficient condition for an unsatisfiable atom set,
3. Proposes procedures for checking whether a given atom set satisfies the condition.

Further, we explicate a method for proving unsatisfiability of atom sets using induction based on computation by ET rules.

The remainder of this paper is organised as follows: Section 2 expounds on the purpose of this paper by discussing the importance of ET rules in program synthesis. Section 3 discusses background knowledge as it relates to program synthesis and gives a definition for unsatisfiable atom sets. Examples of this type of atom set are also presented. Section 4 defines declarative meaning and the concept of an ET rule, and outlines the conditions that must be satisfied for an atom set to be deemed unsatisfiable. Section 5 presents an algorithm for proving unsatisfiability of an atom set based on computation by ET rules. Section 6 outlines the process involved in proving unsatisfiability of a concrete atom set using the algorithm. Section 7 discusses the usefulness of the method proposed in this paper and compares it with other approaches. Section 8 gives conclusions.

2. Importance of ET Rules and Proof of Unsatisfiability.

2.1. Importance of ET rules in program synthesis.

Methodologies for the construction of both sequential and parallel programs based on program synthesis by ET rules have been proposed [1, 3]. In the proposed methodologies, ET rules are used to describe procedures. An ET rule guarantees partial correctness with respect to a specification; thus, if a program is constructed using ET rules then partial correctness of that program is also guaranteed. Additionally, since each ET rule is completely independent and individually correct, partial correctness of the program is guaranteed even if a new ET rule is added to the existing program.

Program synthesis using ET rules is important in our proposed method for proving atom sets. In program synthesis using ET rules, programs can be constructed simply by making ET rules and accumulating them one by one [2, 12]. In order to prove atom sets, it is necessary to construct an algorithm for the proof. As algorithms become more complex, the need for the construction of cost-effective and reliable algorithms increases. Program synthesis based on the accumulation of components can effectively construct complex algorithms. Since ET rules can be considered components of a program, they can confer considerable advantage in this respect.

2.2. **Importance of proof of unsatisfiability.** An ET program transforms a clause set into a simpler one repeatedly, while preserving its declarative meaning. ET rules can be used to describe various procedures. Efficient programs may be constructed by making efficient ET rules from a specification, and by accumulating them. Thus, methods for making ET rules are important for constructing efficient programs. The efficiency of this computation depends on the number of clause set transformations. Generally, it is preferable to decrease the number of transformations for the efficiency of the computation.

A False rule is a useful rule that speeds up computation by removing definite clauses that contain body atoms which do not satisfy the constraints of predicates. Since there is no need to obtain a set of answers for a clause that is removed by a False rule, transforming such a clause unnecessarily increases the number of clause set transformations. Thus, it is better to remove that clause as early as possible. A False rule can remove the clause, so the rule is useful for constructing efficient programs.

This paper proposes a new method for proving that a given atom set is an unsatisfiable atom set. The method is important in the making of False rules from specifications. Assume that a False rule that applies to a clause $\mathcal{C}$ is made. The False rule can be made using the following steps:

1. Select an atom set from the body of clause $\mathcal{C}$.
2. Check whether the atom set is unsatisfiable.
3. Make a False rule from the unsatisfiable atom set.

In Step 1, one or more atoms are nondeterministically selected from the body atoms. Step 3 can be accomplished using the method proposed by Miura, in which False rules are generated from LEs made from specified unsatisfiable atom sets [13]. Step 2 of the procedure has not yet been realized; therefore, it is essential that a method for Step 2 be developed. This paper proposes just such a method for proving that a given atom set is an unsatisfiable atom set; consequently, making the overall procedure complete.

3. **Definition of Unsatisfiable Atom Set with Examples.**

3.1. **Background knowledge.** The decision as to whether an atom set is unsatisfiable is dependent on background knowledge. Background knowledge is defined by predicates described by a set of clauses, denoted by $\mathbb{D}$. A predicate is generally defined by one or more clauses. For example, a predicate *app* for the concatenation of lists is defined by the following two clauses:

$$\begin{aligned} cl_1 :&\ \ app([\,],\ X,\ X) \leftarrow \\ cl_2 :&\ \ app([A \mid X],\ Y,\ [A \mid Z]) \leftarrow app(X,\ Y,\ Z) \end{aligned}$$

Clause cl_1 means that the concatenation of an empty list and a list X yields a list X. Clause cl_2 means that if list Z is the concatenation of lists X and Y, then list $[A \mid Z]$ is the concatenation of lists $[A \mid X]$ and Y. Ground atoms obtained from the two clauses are valid atoms; otherwise, they are unsatisfiable atoms. $app([\,],[1],[1])$ and $app([a,b],[c],[a,b,c])$ are valid atoms, while $app([\,],[1],[\,])$ and $app([a],[b],[c])$ are unsatisfiable atoms. A predicate *rev* is defined by the following two clauses and the definition of the *app* predicate:

$$\begin{aligned} cl_3 :&\ \ rev([\,],[\,]) \leftarrow \\ cl_4 :&\ \ rev([A \mid X],Z) \leftarrow app(Y,[A],Z),\ rev(X,Y) \end{aligned}$$

Clause cl_3 means an empty list is an empty list even if it is in reverse order. Clause cl_4 means that if a list Y is an element of a list X in reverse order and a list Z is the concatenation of Y and $[A]$, then Z is an element of $[A \mid X]$ in reverse order.

3.2. **Definition of unsatisfiable atom sets.** An unsatisfiable atom set contains atoms that do not satisfy predicate constraints on background knowledge $\mathbb{D}$. The condition of an unsatisfiable atom set is read as the equivalence relationship of the unsatisfiable atom set and $\{false\}$ with respect to $\mathbb{D}$. The $false$ atom is a special built-in atom that always equates to false.

Definition 3.1. *An atom set $\mathbb{E}$ is unsatisfiable with respect to background knowledge $\mathbb{D}$ iff the formula*

$$\mathcal{M}(\mathbb{D}) \models \forall(\mathbb{E} \leftrightarrow \{false\})$$

is true.[1]

3.3. **Examples of unsatisfiable atom sets.** Let us now look at examples of unsatisfiable atom sets using the predicates *app* and *rev*. The atom $neq(A1,\ A2)$ means that $A1$ is not equal to $A2$.

$$\begin{aligned}&\mathbb{E}_1: && \{app(A,\ [X],\ A)\}\\ &\mathbb{E}_2: && \{app(A,\ B,\ C),\ app(A,\ B,\ D),\ neq(C,\ D)\}\end{aligned}$$

Set $\mathbb{E}_1$ means that the concatenation of lists A and $[X]$ is A. From the viewpoint of the meaning of the predicate *app*, the third argument is a list that adds an element X to the last element of A. Since the atom does not satisfy the predicate constraint, $\mathbb{E}_1$ is unsatisfiable with respect to $\mathbb{D}$. The next example is a set $\mathbb{E}_2$. This set means that lists C and D result from the concatenation of lists A and B, and C is not equal to D. If the *neq* atom satisfies the predicate constraint, then the two *app* atoms are false. Whereas, if the two *app* atoms satisfy the predicate constraint, then the *neq* atom is false. Thus, $\mathbb{E}_2$ is unsatisfiable with respect to $\mathbb{D}$.

$$\mathbb{E}_3: \ \{rev(A,\ B),\ rev(A,\ C),\ neq(B,\ C)\}$$

The set $\mathbb{E}_3$ means that lists B and C are elements of the list A in reverse order, and B is not equal to C. From the viewpoint of the meaning of the predicate *rev*, B is equal to C. Thus, $\mathbb{E}_3$ is unsatisfiable with respect to $\mathbb{D}$.

4. Theorem for Unsatisfiable Atom Set.

4.1. **Declarative meaning.** In Theorem 4.1 (see Section 4.2), the unsatisfiability of an atom set is proven by using the equivalence of declarative meaning. Given a set $\mathcal{D}$ of definite clauses, the declarative meaning of $\mathcal{D}$, denoted by $\mathcal{M}(\mathcal{D})$, is given by Definition 4.1.

Definition 4.1. *Declarative meaning*

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

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

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

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

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

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

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

[1] Let $\mathcal{G}$ be a set of ground atoms and let $\mathcal{F}$ be a logical formula. $\mathcal{G} \models \mathcal{F}$ means that $\mathcal{G}$ is a model of $\mathcal{F}$. From $\mathcal{G} \models \mathcal{F}$, $\mathcal{M}(\mathbb{D}) \models \forall(\mathbb{E} \leftrightarrow \{false\})$ is equal to $\forall\theta : (E\theta \subseteq \mathcal{G} \Rightarrow (E\theta \subseteq \mathcal{M}(\mathbb{D}) \leftrightarrow false))$.

4.2. Theorem for unsatisfiable atom set.

From the definition of the declarative meaning, if E is valid with respect to background knowledge $\mathbb{D}$, then $\mathcal{M}(\{ans \leftarrow E\})$ is the set of ans atoms. On the other hand, if E is unsatisfiable, then an empty set is obtained.

Theorem 4.1. *Let E be an atom set. It is assumed that ans does not exist as predicates on $\mathbb{D}$. An atom set E is unsatisfiable with respect to $\mathbb{D}$ iff the formula*

$$\mathcal{M}(\mathbb{D} \cup \{ans \leftarrow E\}) = \mathcal{M}(\mathbb{D})$$

is true.

Proof: $\mathcal{M}(\mathbb{D})$ is a model of $\mathbb{D}$.

$$\begin{array}{l}
\mathcal{M}(\mathbb{D} \cup \{ans \leftarrow E\}) = \mathcal{M}(\mathbb{D}) \\
\qquad\qquad\Updownarrow \\
\mathcal{M}(\mathbb{D}) \cup \{ans \mid \exists\theta : E\theta \subseteq \mathcal{M}(\mathbb{D})\} = \mathcal{M}(\mathbb{D}) \\
\qquad\qquad\Updownarrow \\
\{ans \mid \exists\theta : E\theta \subseteq \mathcal{M}(\mathbb{D})\} = \emptyset \\
\qquad\qquad\Updownarrow \\
\neg(\exists\theta : E\theta \subseteq \mathcal{M}(\mathbb{D})) \\
\qquad\qquad\Updownarrow \\
\forall\theta : (E\theta \subseteq \mathcal{G} \Rightarrow E\theta \not\subseteq \mathcal{M}(\mathbb{D})) \\
\qquad\qquad\Updownarrow \\
\forall\theta : (E\theta \subseteq \mathcal{G} \Rightarrow (E\theta \subseteq \mathcal{M}(\mathbb{D}) \leftrightarrow false)) \\
\qquad\qquad\Updownarrow \\
\mathcal{M}(\mathbb{D}) \models \forall(E \leftrightarrow \{false\})
\end{array}$$

□

4.3. Definition of ET rules.

In this paper, proof that an atom set is unsatisfiable is computed using ET rules. An ET rule is a rewriting rule that replaces a clause set with another one while preserving its declarative meaning with respect to background knowledge $\mathbb{D}$.

Definition 4.2. *Definition of ET rules*
A rewriting rule r is an ET rule with respect to $\mathbb{D}$ iff the formula

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

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

5. Algorithm for Proving Unsatisfiability of Atom Set.

5.1. Algorithm $\mathcal{P}$.

In this section, we present an algorithm $\mathcal{P}$ for proving unsatisfiability of an atom set E using ET rules. From Theorem 4.1, a set E is an unsatisfiable atom set with respect to background knowledge $\mathbb{D}$ iff E satisfies the condition $\mathcal{M}(\mathbb{D} \cup \{ans \leftarrow E\}) = \mathcal{M}(\mathbb{D})$. If an empty set $\{\ \}$ is obtained from the clause set $\{ans \leftarrow E\}$ due to the application of ET rules, then set E satisfies the condition $\mathcal{M}(\mathbb{D} \cup \{ans \leftarrow E\}) = \mathcal{M}(\mathbb{D})$. The proof procedure using algorithm $\mathcal{P}$ is as follows:

[**Algorithm** $\mathcal{P} = unsat(CS, R)$]

Algorithm $\mathcal{P}$ is a function $unsat$. The inputs to the function are a clause set $CS(= \{ans \leftarrow E\})$ and an ET rule set R; while the output is $\{true,\ false\}$.

```
1. Transform $CS$ into $CS'$ by applying $R$ zero or more times.
2. If $CS' = \{\ \}$, the output is $true$.
3. Find an atom set $E'$ and a clause set $CS''$ such that
        i. $B$ is a set of atoms,
        ii. $\{ans \leftarrow B\} \cup CS'' = CS'$, and
        iii. $E' \subset B$.
    If $unsat(\{ans \leftarrow E'\} \cup CS'',\ R) = true$, then stop with success.
4. Select a clause $ans \leftarrow B$ and a clause set $CS''$ such that
        i. $B$ is a set of atoms and contains one or more variables, and
        ii. $\{ans \leftarrow B\} \cup CS'' = CS'$.
    Select a variable $V$ such that
        i. $V$ is a variable that has appeared in $B$.
    Repeat the selection of other $\{ans \leftarrow B\} \cup CS''$ and other $V$ until $true$ is obtained
    for all the outputs of procedures ①, ②, ③ and ④.
    ① Make a substitution $\theta^0$ and a clause $C^0$ such that
        i. $\theta^0 = \{V/[\ ]\}$, and
        ii. $C^0 = (ans\theta^0 \leftarrow B\theta^0)$.
      Call $unsat(\{C^0\} \cup CS'',\ R)$.
    ② Make a substitution $\theta^k$ and a clause $C^k$ such that
        i. $\theta^k = \{V/[P \mid Q^{\sim k}]\}$, and
        ii. $C^k = (ans\theta^k \leftarrow B\theta^k)$.
      Generate an inductively-used rule $iR$ such that
        i. $\theta^i = \{V/Q^{\sim k}\}$, and
        ii. $iR = gen(\forall(B\theta^i \leftrightarrow \{false\}))$.
      Call $unsat(\{C^k\} \cup CS'',\ R \cup \{iR\})$.
    ③ Make a substitution $\theta^{num}$ and a clause $C^{num}$ such that
        i. $\theta^{num} = \{V/X^{\sim num}\}$, and
        ii. $C^{num} = (ans\theta^{num} \leftarrow B\theta^{num})$.
      Call $unsat(\{C^{num}\} \cup CS'',\ R)$.
    ④ Make a substitution $\theta^{sym}$ and a clause $C^{sym}$ such that
        i. $\theta^{sym} = \{V/X^{\sim sym}\}$, and
        ii. $C^{sym} = (ans\theta^{sym} \leftarrow B\theta^{sym})$.
      Call $unsat(\{C^{sym}\} \cup CS'',\ R)$.
```

The algorithm contains many non-deterministic procedures. In Step 1 of the algorithm, if an empty set is obtained from CS, it is preferable that clause transformation is repeated until an empty set is obtained. However, an empty set is not necessarily obtained. In such a case, the number of clause transformations is controlled in order to be able to select a useful subset in Step 3 or a useful variable in Step 4. In Step 3, it is preferable that several ground instances of E' are false with respect to background knowledge $\mathbb{D}$. Since various E's which satisfy that condition will exist, the set E' is nondeterministically selected from among them. The variable V in Step 4 is nondeterministically selected from among useful variables to execute an induction. *gen* is a function for relating an unsatisfiable formula to an inductively-used rule. This function is expounded on in Section 5.2.

[Feature of induction in this paper]

The induction in this paper is executed by using the recursiveness of the list which appears in an atom set. The computation is executed using a set of ET rules and an inductively-used rule. Let $\delta 1$ be a substitution $\{\mathcal{V}/[P \mid Q^{\sim k}]\}$; let $\delta 2$ be a substitution $\{\mathcal{V}/Q^{\sim k}\}$; and let $\delta 3$ be a substitution $\{\mathcal{V}/[\ ]\}$. $\mathcal{V}$ is a variable in an atom set $\mathcal{E}$. The unsatisfiability of $\mathcal{E}\delta 1$ is proven by induction in accordance with the following steps:

1. Make an inductively-used rule iR based on $\mathcal{E}\delta 2$, where it is assumed that $\mathcal{E}\delta 2$ is unsatisfiable.
2. The unsatisfiability of $\mathcal{E}\delta 3$ is proven using rule set $\mathcal{R}$.

3. The unsatisfiability of $\mathcal{E}\delta 1$ is proven using rule set $\mathcal{R} \cup \{iR\}$.

5.2. **Generation of inductively-used rules.** An inductively-used rule is generated from an unsatisfiable formula in Step 4 of algorithm $\mathcal{P}$. An unsatisfiable formula called Fe, describes the equivalence relationship of atom sets $\mathbb{E}$ and $\{false\}$, and is defined by

$$\forall(\mathbb{E} \leftrightarrow \{false\}).$$

This formula means that the truth-value of $\mathbb{E}$ is equivalent to that of $\{false\}$ for any ground substitutions of variables in $\mathbb{E}$. An inductively-used rule, iR, is defined by

$$\mathbb{U} \Rightarrow \mathit{not}(\mathbb{B}), \mathbb{U},$$

where $\mathbb{E} = \mathbb{U} + \mathbb{B}$. If atom set $\mathbb{B}$ is an empty set $\{\ \}$, then the form of the generated rules is $\mathbb{U} \Rightarrow false$. Function *gen* is a mapping that relates an element in iR to an element in Fe. For example, it is assumed that the following formula is obtained as an element in Fe:

$$\forall(\{rev(X,Y), rev(X,Z), neq(Y,Z)\} \leftrightarrow \{false\})$$

Atom set $\mathbb{B}$ is the set of built-in atoms. Since the built-in atoms in the above formula are $neq(Y,Z)$, atom sets $\mathbb{B}$ and $\mathbb{U}$ are

$$\begin{aligned} \mathbb{B} = &\quad \{neq(Y,Z)\}, \\ \mathbb{U} = &\quad \{rev(X,Y), rev(X,Z)\}. \end{aligned}$$

Thus, the following formula is obtained:

$$\begin{aligned} & rev(X,Y), rev(X,Z) \Rightarrow \mathit{not}(neq(Y,Z)), rev(X,Y), rev(X,Z) \\ & \quad \Downarrow \ \ \mathit{not}(neq(Y,Z)) = eq(Y,\ Z). \\ & rev(X,Y), rev(X,Z) \Rightarrow eq(Y,Z), rev(X,Y), rev(X,Z) \\ & \quad \Downarrow \ \ \text{The right-hand side can be replaced with } \{eq(Y,Z), rev(X,Y)\}. \\ & rev(X,Y), rev(X,Z) \Rightarrow eq(Y,Z), rev(X,Y) \end{aligned}$$

6. **Example of Proof of Unsatisfiability.** In this section, we prove the unsatisfiability of an atom set that comprises atoms of predicates *rev* and *neq*, and which is described as

$$\mathbb{E}_{rev} = \{rev(X,A), rev(X,B), neq(A,B)\}$$

Set $\mathbb{E}_{rev}$ means that lists A and B are elements of a list X in reverse order and A is not equal to B. An initial set of rules for proving the unsatisfiability of $\mathbb{E}_{rev}$ is the following set, $\mathbb{R}_{rev}$.

$$\mathbb{R}_{rev} = \begin{cases} r_1 : rev(A,B) \Rightarrow eq(A,[\ ]), eq(B,[\ ]) \\ \qquad\qquad\qquad \Rightarrow eq(A,[A1 \mid A2]), rev(A2,\ A3), app(A3,[A1],B) \\ r_2 : rev(A,B), rev(A,B) \Rightarrow rev(A,B) \\ r_3 : neq(A,A) \Rightarrow \{false\} \\ r_4 : eq(A,B) \Rightarrow \{A = B\} \\ r_5 : eq([A \mid B],[C \mid D]) \Rightarrow eq(A,C), eq(B,D) \\ r_6 : eq(A,B), \{ivar(A), getInfo(A,num), list(B)\} \Rightarrow \{false\} \\ r_7 : eq(A,B), \{ivar(A), getInfo(A,sym), list(B)\} \Rightarrow \{false\} \end{cases}$$

The clause set CS for this example is $\{ans \leftarrow rev(X,\ A),\ rev(X,\ B),\ neq(A,\ B)\}$. In Step 1 of algorithm $\mathcal{P}$, an empty set cannot be obtained from CS by applying $\mathbb{R}_{rev}$, so let CS be CS'. In Step 3, a subset $\mathbb{E}'_{rev}$ in which *true* is obtained as an output of the function *unsat* does not exist, so we execute Step 4. In Step 4, let variable X be a variable selected from among those in the body of CS'. Additionally, the following functions are called in Step 4:

① : $unsat(\{ans \leftarrow rev([\,], A), rev([\,], B), neq(A, B)\}, \mathbb{R}_{rev})$

② : $unsat(\{ans \leftarrow rev([P \mid Q^{\sim k}], A), rev([P \mid Q^{\sim k}], B), neq(A, B)\}, \mathbb{R}_{rev} + \{iR_{rev}\})$
$iR_{rev} = (rev(Q^{\sim k}, A), rev(Q^{\sim k}, B) \Rightarrow eq(A, B), rev(Q^{\sim k}, A))$

③ : $unsat(\{ans \leftarrow rev(X^{\sim num}, A), rev(X^{\sim num}, B), neq(A, B)\}, \mathbb{R}_{rev})$

④ : $unsat(\{ans \leftarrow rev(X^{\sim sym}, A), rev(X^{\sim sym}, B), neq(A, B)\}, \mathbb{R}_{rev})$

[Executing function ①]

An empty set is obtained from $\{ans \leftarrow rev([\,],\ A),\ rev([\,],\ B),\ neq(A,\ B)\}$ by applying $\mathbb{R}_{rev}$. In Step 1, the following set is obtained from the clause set:

$$\Big\{ans \leftarrow neq(A, [\,]), \underline{eq([\,], [C \mid D])}, rev(D, E), app(E, [C], A)\Big\}$$

Since the first argument of the underlined atom is an empty set $[\,]$ and the second argument is a list, $[C \mid D]$, composed of one or more elements, the constraint of the eq predicate cannot be satisfied. An empty set is obtained because this clause is removed.

[Executing function ②]

By applying the rules in $\mathbb{R}_{rev} + \{iR_{rev}\}$ four times, the following set is obtained from $\{ans \leftarrow rev([P \mid Q^{\sim k}], A), rev([P \mid Q^{\sim k}], B), neq(A, B)\}$:

$$\Big\{ans \leftarrow \underline{neq(A, B), app(C, [D], A), app(C, [D], B)}, rev(E^{\sim k}, C)\Big\}$$

We now treat the underlined atoms as an atom set. The atom set can be read as follows: The concatenation of lists C and $[D]$ results in lists A and B, and A is not equal to B. From the viewpoint of the meaning of the app predicate, it is believed that A is equal to B. Thus, there is a possibility that the atom set is an unsatisfiable atom set. In Step 3, let E' be the atom set $\{neq(A, B), app(C, [D], A), app(C, [D], B)\}$. The details of the computation of the following function is omitted in this paper due to space constraints.

$$unsat(\{ans \leftarrow neq(A, B), app(C, [D], A), app(C, [D], B)\}, \mathbb{R}_{rev})$$

[Executing function ③]

An empty set is obtained from $\{ans \leftarrow rev(X^{\sim num}, A), rev(X^{\sim num}, B), neq(A, B)\}$ when $\mathbb{R}_{rev}$ is applied. In Step 1, the following set is obtained from the clause set:

$$\{ans \leftarrow rev(A^{\sim num}, B), neq(C, B), \underline{eq(A^{\sim num}, [D \mid E])}, rev(E, F), app(F, [D], C)\}$$

Since the first argument of the underlined atom is a digit and the second argument is a list, the constraint of the eq predicate cannot be satisfied. An empty set is obtained because this clause is removed.

[Executing function ④]

An empty set is obtained from $\{ans \leftarrow rev(X^{\sim sym}, A), rev(X^{\sim sym}, B), neq(A, B)\}$ when $\mathbb{R}_{rev}$ is applied. In step 1, the following set is obtained from the clause set:

$$\{ans \leftarrow rev(A^{\sim sym}, B), neq(C, B), \underline{eq(A^{\sim sym}, [D \mid E])}, rev(E, F), app(F, [D], C)\}$$

Since the first argument of the underlined atom is a symbol and the second argument is a list, the constraint of the eq predicate cannot be satisfied. An empty set is obtained because this clause is removed.

Consequently, since the outputs of the four functions ①, ②, ③, and ④ are $true$, $\mathbb{E}_{rev}$ is deemed unsatisfiable with respect to $\mathbb{D}$.

7. Discussion and Comparisons.

7.1. Application to program generation.

7.1. Application to program generation. Methods for generating programs from specifications are essential. Methods for generating C, C++, and Java programs have been proposed by many researchers [5, 8, 11]. We also proposed a system for generating C programs from sets of ET rules [16]. Consequently, if ET rules can be automatically generated from specifications, then C programs can be automatically generated (indirectly) from those specifications (via ET rules) by combining the system and an ET rule generator. Therefore, methods for generating ET rules, such as the LE-based method [13], are very important.

The proof method proposed in this paper is closely connected to the methods for making ET rules. In particular, the proof method is very useful when ET rules are generated via the LE-based method. In the LE-based method, ET rules are generated from specifications via LEs. An LE describes an equivalence relationship between two logical formulas under specified preconditions. It is necessary to prove the equivalence of LEs when ET rules are made by the LE-based method. The proof method proposed in this paper is useful for proving the equivalence of LEs.

Methods for proving the correctness of logical formulas have been proposed before [6, 7, 15]. However, these proposals were not focused on its application to program generation, so the methods are not closely connected to program generation. As a result, the proposals cannot be applied to problem settings such as those dealt with in this paper.

7.2. Comparison with related studies. Researchers such as Hsiang and Srivas [6] and Sakurai and Motoda [15] proposed methods for proving the correctness of logical formulas. In their proposed methods, the correctness of logical formulas is proven based on logic programming theory [10]. Sakurai and Motoda [15] presented problems [4] that cannot be proven except by induction and proposed a method for proving those problems. On the other hand, Hsiang and Srivas [6] proposed a method for proving logical formulas that includes a list data structure that is similar to our *app* predicate. Induction is also used in a proposed method by Huet and Hullot for proving sets of equations [7].

In this paper and related studies, the correctness of logical formulas is proven by replacing a logical formula with another until certain specified conditions have been satisfied. The condition used in this paper is that the result of replacements is an empty set. In related studies, a logical formula in which atoms on the right-hand side are equal to those on the left-hand side is obtained. The method proposed in this paper (*the structural method*) replaces a logical formula using a set $\mathbb{R}$ of ET rules, while the methods proposed in related studies (*the flat method*) use procedures obtained from a set $\mathbb{D}$ of definite clauses. The difference between our proposed method and those methods is that the set $\mathbb{R}$ can add new rules while the set $\mathbb{D}$ cannot. The structural method can prove logical formulas by making new rules and adding them to the set $\mathbb{R}$ if rules that are needed for the proof do not exist (refer to the left side of Figure 1). Our original contribution in this paper is the concept underlying the structural method. For example, a rule that is made based on $\{app(A, B, C), app(A, B, D), neq(C, D)\}$ is needed to prove unsatisfiability of $\{rev(A, B), rev(A, C), neq(B, C)\}$. In the structural method, the rule can be made by proving the unsatisfiability of $\{app(A, B, C), app(A, B, D), neq(C, D)\}$ in the middle of the proof of $\{rev(A, B), rev(A, C), neq(B, C)\}$ if the rule does not exist in the set $\mathbb{R}$. Additionally, the application of ET rules can be recycled for various proof problems, so a competent procedure set can be constructed by accumulating ET rules.

7.3. Enhancement of ET rule generation system. The LE-based method and a meta-computation-based method [2] have been proposed for generating ET rules from

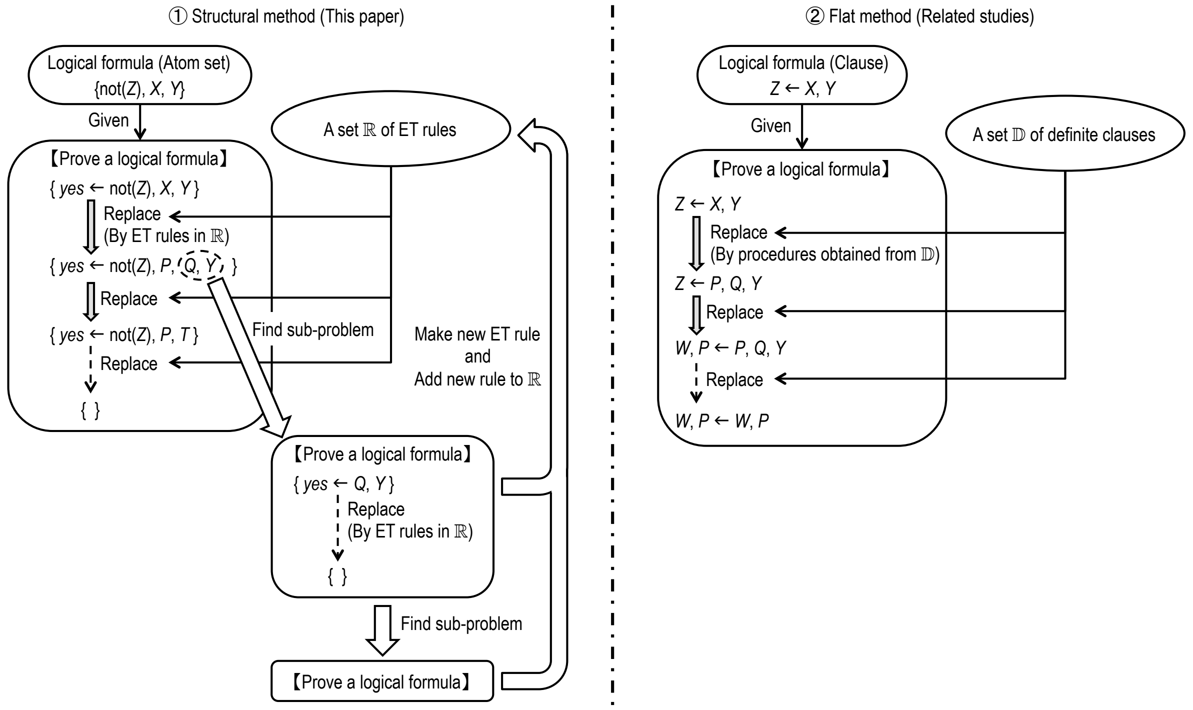

Figure 1. Difference between the proof method proposed in this paper and those from related studies

Table 1. Atom sets that can be proved using our proposed method

	Atom set
Pattern which does not need induction	$\{app(A,\ [B],\ [\ ])\}$, $\{rev([\ ],\ [A \mid B])\}$
Pattern which needs induction	$\{app(A,\ [B],\ A)\}$, $\{app(A,\ [B \mid C],\ C)\}$, $\{app(A,\ B,\ C),\ app(A,\ B,\ D),\ neq(C,\ D)\}$, $\{rev([A \mid B],\ B)\}$, $\{rev(A,\ B),\ rev(A,\ C),\ neq(B,\ C)\}$, $\{rev(B,\ A),\ rev(A,\ C),\ neq(B,\ C)\}$

specifications. By incorporating our proposed method into the LE-based method, the range of ET rules that can be made by the LE-based method is expanded. As a result, many False rules, which are important for solving constraint satisfaction problems, can be made based on the proof of unsatisfiable atom sets [2]. By proposing a method for proving unsatisfiable atom sets, the correctness of the following LE can be guaranteed. The $\mathbb{E}$ in the following formula is a set of atoms.

$$\forall(\mathbb{E} \leftrightarrow \{false\})$$

The correctness of this LE cannot be proven by conventional LE-based methods.

The following rule can be generated by the method proposed in this paper, but not by the meta-computation-based method:

$$r : rev(A,\ B),\ rev(A,\ C),\ neq(B,\ C) \Rightarrow false.$$

Rule r is made by the following three steps:

1. Prove that atom set $\{rev(A,\ B),\ rev(A,\ C),\ neq(B,\ C)\}$ is unsatisfiable.
2. Make an LE $\forall(\{rev(A,\ B),\ rev(A,\ C),\ neq(B,\ C)\} \leftrightarrow \{false\})$ from the atom set.
3. Generate rule r from the LE.

One reason why r cannot be made by the meta-computation-based method is that the variable A in r can present lists of any length. If the first argument in the *rev* atom of r is a variable, then induction is necessary. The induction mechanism is incorporated into our proposed method, whereas it is not in the meta-computation-based method.

In this paper, we proposed, for the first time, an induction method based on computation by ET rules which greatly increases the number of provable atom sets. With a focus on atom sets for the *app* and the *rev* predicates, we proved unsatisfiability of the eight (8) atom sets shown in Table 1 using our proposed method. An induction mechanism was necessary to prove the unsatisfiability of six (6) of the atom sets listed. The unsatisfiability of these atom sets has been proven for the very first time as a result of our proposed method.

8. **Conclusions.** In this paper, we outlined the importance of proving atom sets in the making of ET rules and explained the need for an induction mechanism based on computation by ET rules for proving various atom sets. Consequently, we proposed an induction mechanism based on computation by ET rules and derived a new method for proving atom sets using the mechanism. In particular, in this paper we focused on proof of unsatisfiability of atom sets. Our proposed method executes induction using the recursiveness of a list that appears in an atom set. Since the proof of unsatisfiability of atom sets is effective for making False rules, we believe that many False rules can be made from the results outlined in this paper. Future work will include further development of ET rules generation system based on the method proposed in this paper.

REFERENCES

[1] K. Akama, H. Koike and E. Miyamoto, A theoretical foundation for generation of equivalent transformation rules (program transformation, symbolic computation and algebraic manipulation), *Research Institute for Mathematical Sciences Kyoto University Koukyuroku*, vol.1125, pp.44-58, 2000.

[2] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, *Proc. of PSI2006, LNCS*, vol.4378, pp.41-54, 2007.

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

[4] R. S. Boyer and J. S. Moore, Proving theorems about lisp functions, *Journal of the ACM*, vol.22, no.1, pp.129-144, 1975.

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

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

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

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

[9] H. Koike, K. Akama and E. Boyd, Program synthesis by generating equivalent transformation rules, *Proc. of the 2nd International Conference on Intelligent Technologies*, Bangkok, Thailand, pp.250-259, 2001.

[10] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.

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

[12] H. Mabuchi, K. Akama and T. Wakatsuki, Equivalent transformation rules as components of programs, *International Journal of Innovative Computing, Information and Control*, vol.3, no.3, pp.685-696, 2007.

[13] K. Miura, K. Akama and H. Mabuchi, Creation of ET rules from logical formulas representing equivalent relations, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.263-277, 2009.

[14] K. Miura, K. Akama and H. Mabuchi, Generating Speq rules based on automatic proof of logical equivalence, *International Journal of Computer Science*, vol.3, no.3, pp.190-198, 2008.

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

[16] T. Wakatsuki, K. Akama and H. Mabuchi, A framework for synthesizing low-level imperative programs from deterministic abstract programs, *Technical Report of the Institute of Electronics, Information and Communication Engineers*, vol.107, no.392, pp.37-42, 2007 (in Japanese).