Function-variable Elimination and Its Limitations

Kiyoshi Akama[1] and Ekawit Nantajeewarawat[2]
[1]*Information Initiative Center, Hokkaido University, Hokkaido, Japan*
[2]*Computer Science, Sirindhorn International Institute of Technology, Thammasat University, Pathumthani, Thailand*

Keywords: Query-answering Problem, Equivalent Transformation, Meaning-preserving Skolemization, Problem Solving.

Abstract: The famous proof method by the conventional Skolemization and resolution has a serious limitation. It does not guarantee the correctness of proving theorems in the presence of built-in constraints. In order to understand this difficulty, we use meaning-preserving Skolemization (MPS) and equivalent transformation (ET), which together provide a general framework for solving query-answering (QA) problems on first-order logic. We introduce a rule for function variable elimination (FVE), by which we regard the conventional Skolemization as a kind of the composition of MPS and FVE. We prove that the FVE rule preserves the answers to a class of QA problems consisting of only user-defined atoms, while we cannot prove it in the presence of built-in constraints. By avoiding the application of the FVE rule in MPS & ET computation, we obtain a more general solution for proof problems, which guarantees the correctness of computation even in the presence of built-in constraints.

1 INTRODUCTION

One of the most important methods for proving theorems is based on Skolemization and resolution. This method, however, has a serious limitation in that it may give an incorrect result in the presence of built-in predicates. Consider, for example, a simple proof problem below, which is a modification of the tax-cut problem given in (Motik et al., 2005). Assume that

- *noteq* is a predicate for built-in constraint atoms and for any ground terms t_1 and t_2, $noteq(t_1,t_2)$ is true iff $t_1 \neq t_2$, and
- F_1, F_2, and F_3 are the first-order formulas given by:

F_1: $\forall x, \forall y, \forall z :$
$[(hasChild(x,y) \wedge hasChild(x,z) \wedge noteq(y,z))$
$\rightarrow TaxCut(x)]$

F_2: $hasChild(Peter, Paul)$

F_3: $\exists x : hasChild(Peter, x)$

The problem is to determine whether E_1 logically entails E_2, where $E_1 = F_1 \wedge F_2 \wedge F_3$ and $E_2 = TaxCut(Peter)$. Since F_3 is already implied by F_2, we know only one person who is a child of *Peter*. Hence $E_1 \not\models E_2$, i.e., E_1 does not logically entail $TaxCut(Peter)$. The correct answer to this proof problem is thus "no".

If we take conventional proof method, however, this problem is solved incorrectly. By applying Skolemization to $E_1 \wedge \neg E_2$, a 0-ary function symbol, say f_h, is introduced and $E_1 \wedge \neg E_2$ is converted into the following four clauses:

$TaxCut(x) \leftarrow hasChild(x,y), hasChild(x,z),$
$noteq(y,z)$
$hasChild(Peter, Paul) \leftarrow$
$hasChild(Peter, f_h) \leftarrow$
$\leftarrow TaxCut(Peter)$

Application of the resolution rule three times yields a negative clause ($\leftarrow noteq(Paul, f_h)$. Since *Paul* and f_h are not equal, we derive an empty clause ($\leftarrow$). Thus a proof is obtained by Skolemization and resolution, and the answer is "yes", which contradicts the intuitive and correct answer explained earlier.

From this example, the following questions naturally arise:

1. Where does this inconvenience come from?
2. How to develop a theory to deeply understand the use of Skolemization and resolution and its limitation?
3. Can we invent a new solution method to resolve this difficulty?

We give an answer to each of these questions by replacing the conventional Skolemization and resolu-

Akama, K. and Nantajeewarawat, E..
Function-variable Elimination and Its Limitations.
In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - Volume 2: KEOD*, pages 212-222
ISBN: 978-989-758-158-8

$$\mathcal{L}_1 \xrightarrow{\text{CS}} \mathcal{CL}_1 \xrightarrow{rf} \mathcal{CL}_1 \xrightarrow{rf} \mathcal{CL}_1 \xrightarrow{rf} \cdots$$

Figure 1: A conventional proof diagram.

$$\mathcal{L}_1 \xrightarrow{\text{MPS}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \cdots$$

Figure 2: An ET-based proof diagram.

tion (Robinson, 1965) with the meaning-preserving Skolemization (Akama and Nantajeewarawat, 2011a) and equivalent transformation (ET).

Let $\mathcal{L}_1$ be the set of all first-order formulas. Let CS be the conventional algorithm for transforming a first-order formula into its clausal form using the conventional Skolemization. Let $\mathcal{CL}_1$ be the powerset of the set of all usual clauses. Let *rf* denote resolution and factoring. Then a typical computation path by the conventional proof method can be depicted by Fig. 1. A first-order formula is converted by CS into a set of usual clauses possibly with new function symbols. For instance, an existentially quantified formula $\exists x : p(x)$ is transformed into a clause set $\{(p(h) \leftarrow)\}$, where h is a 0-ary function symbol.

In conventional clauses, all variables are universally quantified and existential quantification cannot be expressed. Conventional clauses are therefore not sufficiently expressive for representing first-order formulas. To extend clauses with the expressive power of existential quantification, variables of a new type, called *function variables*, were introduced (Akama and Nantajeewarawat, 2011a). A function variable may appear in an atom of a special kind, called *func*-atom, which is generally of the form $\mathit{func}(f, t_1, \ldots, t_n, t_{n+1})$, where f is an n-ary function variable or an n-ary function constant, and $t_1, \ldots, t_n, t_{n+1}$ are usual terms.

To understand the computation path in Fig. 1, we consider a new path given in Fig. 2, where MPS is the algorithm for transforming a first-order formula into its extended clausal form using meaning-preserving Skolemization (Akama and Nantajeewarawat, 2011a). For instance, an existentially quantified formula $\exists x : p(x)$ is transformed by MPS into a clause set $\{(p(x) \leftarrow \mathit{func}(f, x))\}$, where $\mathit{func}(f, x)$ is a *func*-atom and f is a 0-ary function variable, which is not included in $\mathcal{L}_1$. An existentially quantified formula cannot be equivalently transformed into a clausal form in the usual first-order formula space $\mathcal{L}_1$. We extended $\mathcal{L}_1$ into a new space, which includes function variables. In Fig. 2, $\mathcal{CL}_2$ is the powerset of the set of all extended clauses, which may possibly include function variables. MPS and extended clauses will be formally defined in Section 3, where the set of all extended clauses is referred to as ECLS_F.

$$\begin{array}{l} \xrightarrow{\text{CS}} \mathcal{CL}_1 \xrightarrow{rf} \mathcal{CL}_1 \xrightarrow{rf} \mathcal{CL}_1 \xrightarrow{rf} \cdots \\ \quad\uparrow \text{FVE} \\ \mathcal{L}_1 \xrightarrow{\text{MPS}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \mathcal{CL}_2 \xrightarrow{\text{ET}} \cdots \end{array}$$

Figure 3: Connecting the conventional proof diagram and the ET-based proof diagram using FVE.

To connect the diagrams in Fig. 1 and Fig. 2, we introduce in this paper a partial mapping FVE from $\mathcal{CL}_2$ to $\mathcal{CL}_1$ as outlined in Fig. 3. For instance, the clause set $\{(p(x) \leftarrow \mathit{func}(f, x))\}$ is mapped by FVE to $\{(p(h) \leftarrow)\}$. The conventional Skolemization, CS, is identified as the composition of MPS and FVE in the sense that

$$\{\text{CS}(L) \mid L \in \mathcal{L}_1\} = \{\text{FVE}(\text{MPS}(L)) \mid L \in \mathcal{L}_1\}.$$

We prove in this paper that FVE preserves the answers to proof problems in a certain restricted class. Since resolution and factoring are ET rules in the space of $\mathcal{CL}_1$, the conventional solution can also be regarded as ET computation, i.e., the conventional diagram supports a restricted form of computation compared with the ET-based proof diagram.

The theory in this paper enables us to compare the conventional solution and the ET-based solution for proof problems in the common MPS & ET framework. The limitation of FVE can be precisely investigated. The difficulty shown by the example at the beginning of this section can be overcome in the MPS & ET framework by using ET computation paths that do not include application of the FVE rule.

The MPS & ET theory has been developed mainly for solving query-answering (QA) problems. A QA problem is a pair $\langle K, a \rangle$, where K is a first-order formula and a is a user-defined atom, and the answer to this problem is the set of all ground instances of a that are logically entailed by K.

While the answer to a proof problem is either "yes" or "no", which does not contain any (first-order) term, the answer to a QA problem is a set of ground atoms that may contain terms. MPS is necessary for solving QA problems. Since a new term introduced by the conventional Skolemization may affect ground atoms in a model of a given first-order formula, the conventional Skolemization is inappropriate for developing a solution for QA problems. So we take MPS over CS. Since ET includes resolution and factoring, we take the MPS & ET framework over the CS & *rf* framework.

It was shown in (Akama and Nantajeewarawat, 2013) that proof problems constitute a specific subclass of QA problems. So it is natural to apply the MPS & ET framework to solve proof problems. The theory presented in this paper is developed as a the-

ory for solving QA problems based on the MPS & ET framework.

The rest of the paper is organized as follows: Section 2 formalizes QA problems and proof problems. Section 3 describes a procedure for converting first-order formulas using MPS into equivalent formulas in extended existentially quantified conjunctive normal forms, and formulates QA problems in clausal forms. Section 4 defines a target mapping, called $\mathbb{MM}$, which provides a basis for ET. Section 5 presents the main theoretical results of this work and the FVE rule. Section 6 explains the limitations of the FVE rule. Section 7 answers the three questions identified in the third paragraph of Section 1. Section 8 provides conclusions. The proofs of all theorems presented in this paper can be found in (Akama and Nantajeewarawat, 2015).

The following notation is used. Given a set A, $pow(A)$ denotes the power set of A and $fpow(A)$ denotes the set of all finite subsets of A.

2 QA PROBLEMS AND PROOF PROBLEMS

2.1 Interpretations and Models

In this paper, an atom occurring in a first-order formula can be either a user-defined atom or a constraint atom. The semantics of first-order formulas based on a logical structure given in (Akama and Nantajeewarawat, 2012) is used. The set of all ground user-defined atoms, denoted by $\mathcal{G}$, is taken as the interpretation domain. An *interpretation* is a subset of $\mathcal{G}$. A ground user-defined atom g is true with respect to an interpretation I iff g belongs to I. Unlike ground user-defined atoms, the truth values of ground constraint atoms are predetermined independently of interpretations. A *model* of a first-order formula E is an interpretation that satisfies E. The set of all models of a first-order formula E is denoted by $Models(E)$. Given first-order formulas E_1 and E_2, E_2 is a *logical consequence* of E_1, denoted by $E_1 \models E_2$, iff every model of E_1 is a model of E_2.

2.2 QA Problems

A *query-answering problem* (*QA problem*) is a pair $\langle K, a\rangle$, where K is a first-order formula, representing background knowledge, and a is a user-defined atom, representing a query. The answer to a QA problem $\langle K, a\rangle$, denoted by $ans_{\mathrm{qa}}(K, a)$, is defined as the set of all ground instances of a that are logical consequences of K. Using $Models(K)$, the answer to a QA problem $\langle K, a\rangle$ can be equivalently defined as

$$ans_{\mathrm{qa}}(K, a) = (\bigcap Models(K)) \cap rep(a),$$

where $rep(a)$ denotes the set of all ground instances of a (Akama and Nantajeewarawat, 2013).

2.3 Proof Problems

A *proof problem* is a pair $\langle E_1, E_2\rangle$, where E_1 and E_2 are first-order formulas, and the answer to this problem, denoted by $ans_{\mathrm{pr}}(E_1, E_2)$, is defined by

$$ans_{\mathrm{pr}}(E_1, E_2) = \begin{cases} \text{“yes”} & \text{if } E_1 \models E_2, \\ \text{“no”} & \text{otherwise.} \end{cases}$$

It is well known that a proof problem $\langle E_1, E_2\rangle$ can be converted into the problem of determining whether $E_1 \wedge \neg E_2$ is unsatisfiable (Chang and Lee, 1973), i.e., whether $E_1 \wedge \neg E_2$ has no model. As a result, $ans_{\mathrm{pr}}(E_1, E_2)$ can be equivalently defined by

$$ans_{\mathrm{pr}}(E_1, E_2) = \begin{cases} \text{“yes”} & \text{if } Models(E_1 \wedge \neg E_2) = \varnothing, \\ \text{“no”} & \text{otherwise.} \end{cases}$$

3 MEANING-PRESERVING TRANSFORMATION ON AN EXTENDED FORMULA SPACE

After defining an extended formula space and an existentially quantified conjunctive normal form (ECNF), a procedure for converting a first-order formula into an ECNF using meaning-preserving Skolemization (MPS) is recalled. The notions of an extended clause space, a plain clause, and a QA problem in a clausal form are then introduced.

3.1 An Extended Formula Space

We consider an extended formula space that contains three kinds of atoms, i.e., user-defined atoms, constraint atoms, and *func*-atoms. A *func*-atom is an expression of the form $func(f, t_1, \ldots, t_n, t_{n+1})$, where f is either an n-ary function constant or an n-ary function variable, and the t_i are usual terms. There are two types of variables: usual variables and function variables. A function variable is instantiated into a function constant or a function variable, but not into a usual term. Each n-ary function constant is associated with a mapping from $\mathcal{G}_{\mathrm{t}}^n$ to $\mathcal{G}_{\mathrm{t}}$, where $\mathcal{G}_{\mathrm{t}}$ denotes the set of all ground terms. The extended space contains both universal quantifiers and existential quantifiers.

An *extended disjunctive form* is a formula of the form

$$\forall v_1,\ldots,\forall v_m : (a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_p \vee \neg \mathbf{f}_1 \vee \cdots \vee \neg \mathbf{f}_q),$$

where $v_1,\ldots,v_m$ are usual variables, each of $a_1,\ldots,a_n,b_1,\ldots,b_p$ is a user-defined atom or a constraint atom, and $\mathbf{f}_1,\ldots,\mathbf{f}_q$ are *func*-atoms.

An *existentially quantified conjunctive normal form* (*ECNF*) is a formula of the form

$$\exists v_{h1},\ldots,\exists v_{hm} : (C_1 \wedge \cdots \wedge C_n),$$

where $v_{h1},\ldots,v_{hm}$ are function variables and $C_1,\ldots,C_n$ are extended disjunctive forms.

3.2 Conversion Algorithm

Assume that

- the initial space INI is the set of all first-order formulas, and
- the target space FIN is the set of all ECNFs.

Let a formula α in INI be given as input. It is transformed into a formula in FIN as follows:

1. Convert $\rightarrow$ and $\leftrightarrow$ equivalently into $\neg$, $\wedge$, and $\vee$, using the following logical equivalences:
$$\begin{array}{rcl} \beta \rightarrow \gamma & \equiv & \neg\beta \vee \gamma \\ \beta \leftrightarrow \gamma & \equiv & (\neg\beta \vee \gamma) \wedge (\neg\gamma \vee \beta) \end{array}$$
2. Move $\neg$ inwards equivalently until each occurrence of $\neg$ immediately precedes an atom, using the following logical equivalences:
$$\begin{array}{rcl} \neg(\neg\beta) & \equiv & \beta \\ \neg(\beta \wedge \gamma) & \equiv & \neg\beta \vee \neg\gamma \\ \neg(\beta \vee \gamma) & \equiv & \neg\beta \wedge \neg\gamma \\ \neg\forall x : \alpha & \equiv & \exists x : \neg\alpha \\ \neg\exists x : \alpha & \equiv & \forall x : \neg\alpha \end{array}$$
3. Repeatedly move $\vee$ in the current formula through $\exists$, $\forall$, and $\wedge$ as far as possible using the following logical equivalences in the left-to-right direction:
$$\begin{array}{rcl} (\exists x : \beta) \vee \gamma & \equiv & \exists x : (\beta \vee \gamma) \\ (\forall x : \beta) \vee \gamma & \equiv & \forall x : (\beta \vee \gamma) \\ (\beta \wedge \gamma) \vee \delta & \equiv & (\beta \vee \delta) \wedge (\gamma \vee \delta) \end{array}$$
Each time the first and the second equivalences above are used, if γ includes x as a free variable, then rename the quantified variable x in $(\exists x : \beta)$ and $(\forall x : \beta)$, respectively, by using a new variable name.
4. Repeatedly move $\wedge$ in the current formula through $\forall$ as far as possible using the following logical equivalence in the left-to-right direction:
$$\forall x : (\beta \wedge \gamma) \quad \equiv \quad (\forall x : \beta) \wedge (\forall x : \gamma)$$
5. If the current formula includes a subformula of one of the two forms
   - $\forall x_1,\ldots,\forall x_{n-1},\forall x_n : \beta$
   - $\forall x_1,\ldots,\forall x_{n-1},\exists x_n : \beta$

   such that $x_1,\ldots,x_{n-1},x_n$ are not mutually distinct, then rename these quantified variables by using new variable names so that different quantifications in this subformula refer to different variables.
6. If the current formula includes $\exists$, then:
   (a) *Skolemization:* From the current formula, select a subformula
   $$\forall x_1,\ldots,\forall x_n,\exists y : \beta,$$
   where $n \geq 0$, such that there is no further universal quantification over this subformula. Transform this subformula into
   $$\exists h,\forall x_1,\ldots,\forall x_n,\forall y : (\beta \vee \neg \mathit{func}(h,x_1,\ldots,x_n,y)),$$
   where h is a new n-ary function variable that has not been used so far.
   (b) Repeatedly move the new $\exists$-subformula (introduced at Step 6a) through $\wedge$ as far as possible using the following logical equivalence in the left-to-right direction:
   $$(\exists h : \beta) \wedge \gamma \quad \equiv \quad \exists h : (\beta \wedge \gamma)$$
   (c) Go to Step 3.
7. Stop with the current formula as the output formula.

It was shown in (Akama and Nantajeewarawat, 2011b) that this algorithm always terminates and yields an output ECNF in FIN that has the same logical meaning as the input first-order formula.

3.3 An Extended Clause Space

An *extended clause* C is a formula of the form

$$a_1,\ldots,a_n \leftarrow b_1,\ldots,b_p,\mathbf{f}_1,\ldots,\mathbf{f}_q,$$

where each of $a_1,\ldots,a_n,b_1,\ldots,b_p$ is a user-defined atom or a constraint atom, and $\mathbf{f}_1,\ldots,\mathbf{f}_q$ are *func*-atoms. All usual variables occurring in C are implicitly universally quantified and their scope is restricted to the extended clause C itself. The sets $\{a_1,\ldots,a_n\}$ and $\{b_1,\ldots,b_p,\mathbf{f}_1,\ldots,\mathbf{f}_q\}$ are called the *left-hand side* and the *right-hand side*, respectively, of the extended clause C, and are denoted by $\mathit{lhs}(C)$ and $\mathit{rhs}(C)$, respectively. When $n = 0$, C is called a *negative extended clause*. When $n = 1$, C is called an *extended definite clause*, the only atom in $\mathit{lhs}(C)$ is called the

head of C, denoted by $head(C)$, and the set $rhs(C)$ is also called the *body* of C, denoted by $body(C)$. When $n > 1$, C is called a *multi-head extended clause*.

When no confusion is caused, an extended clause, a negative extended clause, an extended definite clause, and a multi-head extended clause are also called a *clause*, a *negative clause*, a *definite clause*, and a *multi-head clause*, respectively.

The set of all extended clauses is denoted by $\mathrm{ECLS_F}$. The *extended clause space* in this paper is the powerset of $\mathrm{ECLS_F}$.

Semantically, an extended clause corresponds to an extended disjunctive form, and a set of extended clauses corresponds to an ECNF. Let Cs be a set of extended clauses. Implicit existential quantifications of function variables and implicit clause conjunction are assumed in Cs. Function variables in Cs are all existentially quantified and their scope covers all clauses in Cs. With occurrences of function variables, clauses in Cs are connected through shared function variables. After instantiating all function variables in Cs into function constants, clauses in the instantiated set are totally separated.

When no confusion is caused, a clause C is also written as

- $H \leftarrow B$, provided that $H = lhs(C)$ and $B = rhs(C)$,
- $H \leftarrow B, b_1, \ldots, b_n$, provided that $H = lhs(C)$ and $B \cup \{b_1, \ldots, b_n\} = rhs(C)$, where $n \geq 1$, and
- $h \leftarrow B$, provided that $\{h\} = lhs(C)$ and $B = rhs(C)$.

3.4 Plain Clauses

A plain clause and a plain clause set are defined below.

Definition 1. A clause C is *plain* iff the following conditions are satisfied:

1. If a *func*-atom $func(f, t_1, \ldots, t_n, t_{n+1})$ occurs in C, then $t_1, \ldots, t_n, t_{n+1}$ are usual variables and are all disctinct.
2. If *func*-atoms $func(f, t_1, \ldots, t_n, t_{n+1})$ and $func(f', t'_1, \ldots, t'_m, t'_{m+1})$ both occur in C, then t_{n+1} and t'_{m+1} are different.

A set of clauses is *plain* iff it consists of plain clauses solely. □

Theorem 1. *If a first-order formula is converted by the algorithm in Section 3.2 into a set Cs of clauses in $ECLS_F$, then Cs is a plain clause set.* □

3.5 QA Problems in Clausal Forms

Let $\langle K, a \rangle$ be a QA problem, where K is a first-order formula. Suppose that K is converted into a set Cs of extended clauses by the procedure in Section 3.2. Then the QA problem $\langle K, a \rangle$ is transformed into the QA problem $\langle Cs, a \rangle$, which is said to be in a *clausal form*.

Assume that a proof problem $\langle E_1, E_2 \rangle$ is given, where E_1 and E_2 are first-order formulas. This proof problem can be solved by first constructing a QA problem $\langle E, yes \rangle$, where

- $E = E_1 \wedge \neg E_2$, and
- *yes* is a ground atom that does not appear in E.

Since $\langle E, yes \rangle$ is a QA problem, a solution method for QA problems can be used for solving it. That is, the QA problem $\langle E, yes \rangle$ is transformed into a QA problem in a clausal form $\langle Cs, yes \rangle$, where Cs is a set of extended clauses obtained from E by the conversion procedure in Section 3.2. The answer to the QA problem $\langle Cs, yes \rangle$, i.e., $ans_{\mathrm{qa}}(Cs, yes)$, is either the singleton $\{yes\}$ or the empty set. As shown in (Akama and Nantajeewarawat, 2013), the answer to the proof problem $\langle E_1, E_2 \rangle$, i.e., $ans_{\mathrm{pr}}(E_1, E_2)$, can be obtained through $ans_{\mathrm{qa}}(Cs, yes)$ as follows:

$$ans_{\mathrm{pr}}(E_1, E_2) = \begin{cases} \text{“yes”} & \text{if } ans_{\mathrm{qa}}(Cs, yes) = \{yes\}, \\ \text{“no”} & \text{if } ans_{\mathrm{qa}}(Cs, yes) = \varnothing. \end{cases}$$

4 A TARGET MAPPING $\mathbb{MM}$

A target mapping is a key concept for generating solutions for QA problems. We first prove in this section that a mapping $\mathbb{MM}$ on $pow(\mathrm{ECLS_F})$, which is defined in Section 4.2, is a target mapping, i.e., it satisfies Theorems 2 and 4. Transformations that preserve a target mapping always preserve the answers to QA problems. For elimination of function variables (Theorem 8), Theorem 5 will play an important role in Section 5.

4.1 Preliminary Notation for Defining $\mathbb{MM}$

The notation below is used in Section 4.2 for defining $\mathbb{MM}$.

1. Assumed that (i) for any constraint atom c, $not(c)$ is a constraint atom, (ii) for any constraint atom c and any substitution θ, $not(c)\theta = not(c\theta)$, and (iii) for any ground constraint atom c, c is true iff

$not(c)$ is not true. Given a set Cs of clauses in $\mathrm{ECLS_F}$, let $mvRhs(Cs)$ be defined by

$$mvRhs(Cs) \;=\; \{mvRhs(C) \mid C \in Cs\},$$

where for any clause C, $mvRhs(C)$ is the clause obtained from C as follows: For each constraint atom c in $lhs(C)$, remove c from $lhs(C)$ and add $not(c)$ to $rhs(C)$.

2. Assume that (i) $FVar$ is the set of all function variables and $FCon$ the set of all function constants, and (ii) $Map(FVar, FCon)$ is the set of all mappings from $FVar$ to $FCon$. Given $\sigma \in Map(FVar, FCon)$ and a set Cs of clauses in $\mathrm{ECLS_F}$, let $Cs\sigma \;=\; \{C\sigma \mid C \in Cs\}$, i.e., $Cs\sigma$ is the clause set obtained from Cs by instantiating all function variables appearing in it into function constants using σ.
3. Let $\mathrm{ECLS_{NFV}}$ denote the subset of $\mathrm{ECLS_F}$ that contains all (extended) clauses with no occurrence of any function variable.[1] Let GCL be the set of all clauses that consist only of ground user-defined atoms. Given a set Cs of clauses in $\mathrm{ECLS_{NFV}}$, let $ginst(Cs)$ be defined as a subset of GCL as follows:
   (a) Let Cs_1 be a ground clause set obtained from $mvRhs(Cs)$ by

$$Cs_1 \;=\; \{C\theta \mid (C \in mvRhs(Cs)) \;\&\; (\theta \text{ is a ground instantiation for all usual variables occurring in } C)\}.$$

   (b) Let Cs_2 be a ground clause set obtained from Cs_1 by removing each clause whose right-hand side contains at least one false constraint atom or at least one false *func*-atoms.
   (c) Then let $ginst(Cs)$ be a ground clause set obtained from Cs_2 by removing all true constraint atoms and all true *func*-atoms from the right-hand side of each clause in Cs_2.
4. Let $\mathcal{G}$ denote the set of all ground user-defined atoms. Assume that $\bot$ is a ground atom outside $\mathcal{G}$. Let SEL be defined as the set of all mappings from $fpow(\mathcal{G})$ to $\mathcal{G} \cup \{\bot\}$ such that for any $sel \in \mathrm{SEL}$ and any $X \in fpow(\mathcal{G})$, the following conditions are satisfied:
   - If $X = \varnothing$, then $sel(X) = \bot$.
   - If $X \neq \varnothing$, then $sel(X) \in X$.

   Let GDC be the set consisting of every definite clause whose body contains only ground user-defined atoms and whose head is either a ground user-defined atom or $\bot$. Given a mapping $sel \in \mathrm{SEL}$ and a subset Cs of GCL, let $dc(sel, Cs)$ be defined as a subset of GDC by

$$dc(sel, Cs) \;=\; \{dc(sel, C) \mid C \in Cs\},$$

   where for each clause $C \in Cs$, $dc(sel, C)$ is the definite clause obtained from C as follows:
   - $head(dc(sel, C)) = sel(lhs(C))$
   - $body(dc(sel, C)) = rhs(C)$
5. For any definite-clause set $D \subseteq$ GDC, let the meaning of D, denoted by $\mathcal{M}(D)$, be defined as follows:
   (a) Let a mapping T_D on $pow(\mathcal{G} \cup \{\bot\})$ be defined by: for any $G \subseteq \mathcal{G} \cup \{\bot\}$,

$$T_D(G) \;=\; \{head(C) \mid (C \in D) \;\&\; (body(C) \subseteq G)\}.$$

   (b) $\mathcal{M}(D)$ is then defined as the set $\bigcup_{n=1}^{\infty} T_D^n(\varnothing)$, where $T_D^1(\varnothing) = T_D(\varnothing)$ and for each $n > 1$, $T_D^n(\varnothing) = T_D(T_D^{n-1}(\varnothing))$.

[1] Function constants may occur in clauses in $\mathrm{ECLS_{NFV}}$.

4.2 A Target Mapping $\mathbb{MM}$

Using the notation provided by Section 4.1, we define a mapping $\mathbb{MM}$ (Definition 2) and prove in Theorem 4 that $\mathbb{MM}$ is a target mapping, i.e., for any clause set Cs, $\bigcap Models(Cs) = \bigcap \mathbb{MM}(Cs)$.

Definition 2. Let Cs be a set of clauses in $\mathrm{ECLS_F}$. A collection $\mathbb{MM}(Cs)$ of ground-atom sets is defined as the set

$$\{\mathcal{M}(D) \mid (\sigma \in Map(FVar, FCon)) \;\&\; (sel \in \mathrm{SEL}) \;\&\; (D = dc(sel, ginst(Cs\sigma))) \;\&\; (\bot \notin \mathcal{M}(D))\}. \quad \square$$

In Theorems 2–5 below, assume that Cs is a set of clauses in $\mathrm{ECLS_F}$.

Theorem 2. $\mathbb{MM}(Cs) \subseteq Models(Cs)$. $\square$

Theorem 3. *For any m in $Models(Cs)$, there exists $m' \in \mathbb{MM}(Cs)$ such that $m' \subseteq m$.* $\square$

Theorem 4. $\bigcap Models(Cs) = \bigcap \mathbb{MM}(Cs)$. $\square$

Theorem 5 below is used for proving the correctness of elimination of function variables in Theorem 8.

Theorem 5. $Models(Cs) = \varnothing$ *iff* $\mathbb{MM}(Cs) = \varnothing$. $\square$

5 ELIMINATION OF FUNCTION VARIABLES

This section investigates the FVE partial mapping, defined in Section 5.1, which maps a plain set of clauses to a set of usual clauses. By the application of FVE, all function variables in a given plain clause set are instantiated. Given an arbitrary first-order formula L, the formula $\mathrm{MPS}(L)$ is a plain clause set and it can be further converted into a set of usual clauses (without function variables), which can also be obtained by applying the conversion algorithm CS (cf. Fig. 1 in Section 1) to L. The main objective of this section is to prove Theorem 9, which states that the answer to a proof problem is preserved by FVE transformation.

5.1 Satisfiability Preservation

Theorem 6 provides a foundation for satisfiability-preserving transformation.

Theorem 6. *Assume that Cs is a plain set of clauses with no occurrence of any constraint atom. Then the following conditions are equivalent:*

- $(\forall sel \in \mathrm{SEL}) : \bot \in \mathcal{M}(dc(sel, ginst(Cs\sigma_M)))$
- $(\forall \sigma \in Map(FVar, FCon))(\forall sel \in \mathrm{SEL}) : \bot \in \mathcal{M}(dc(sel, ginst(Cs\sigma)))$. □

Now we consider transformation of Cs into Cs', where

- Cs is a plain set of clauses with no occurrence of any constraint atom, and
- Cs' is obtained from Cs as follows: For any $C \in Cs$ and any *func*-atom $func(h, v_1, \ldots, v_n, u)$, denoted by **f**, occurring in C,
  1. remove the *func*-atom **f** from C, and
  2. replace every occurrence of u in C with $f_h(v_1, \ldots, v_n)$, where $f_h = funcSym_M(\sigma_M(h))$.

This transformation defines a partial mapping, which is called FVE.

Theorem 7. $\mathbb{MM}(Cs\sigma_M) = \mathbb{MM}(Cs')$. □

Theorem 8. *$Models(Cs) = \varnothing$ iff $Models(Cs') = \varnothing$.* □

5.2 An ET Rule for Elimination of Function Variables

Referring to the transformation of Cs into Cs' described in Section 5.1, Theorem 9 yields an ET rule for elimination of function variables, which is henceforth called the *FVE rule*.

Theorem 9. *Assume that yes is a ground atom that does not appear in Cs. Then $ans_{\mathrm{qa}}(Cs, yes) = ans_{\mathrm{qa}}(Cs', yes)$.* □

5.3 A Solution Method for Proof Problems

5.3.1 A Procedure for Solving Proof Problems

A procedure for solving proof problems is given below. Let $\langle E_1, E_2 \rangle$ be an input proof problem.

1. Construct a QA problem $\langle E, yes \rangle$, where
   - $E = E_1 \wedge \neg E_2$, and
   - *yes* is a ground atom that does not appear in E.
2. Convert the QA problem $\langle E, yes \rangle$ by MPS, using the conversion procedure in Section 3.2, into a QA problem $\langle Cs, yes \rangle$ in a clausal form.
3. Convert $\langle Cs, yes \rangle$ by the FVE rule into a QA problem $\langle Cs', yes \rangle$.
4. Using ET rules, transform $\langle Cs', yes \rangle$ into a QA problem $\langle Cs'', yes \rangle$
5. Determine the answer to the proof problem $\langle E_1, E_2 \rangle$ by

$$ans_{\mathrm{pr}}(E_1, E_2) = \begin{cases} \text{“yes”} & \text{if } ans_{\mathrm{qa}}(Cs'', yes) = \{yes\}, \\ \text{“no”} & \text{if } ans_{\mathrm{qa}}(Cs'', yes) = \varnothing. \end{cases}$$

If the clause set Cs'' contains an empty clause ($\leftarrow$), then $Models(Cs'') = \varnothing$ and thus $\bigcap Models(Cs'') = \mathcal{G}$. In this case, $ans_{\mathrm{qa}}(Cs'', yes) = \{yes\}$. If the clause set Cs'' has a model, i.e., $Models(Cs'') \neq \varnothing$, then $(\bigcap Models(Cs'')) \cap \{yes\} = \varnothing$. Hence, in this case, $ans_{\mathrm{qa}}(Cs'', yes) = \varnothing$.

5.3.2 Example

To illustrate application of the above procedure, let first-order formulas F_1, F_2, and F_3 be given by:

$$F_1\colon\ \forall x : [barber(x) \rightarrow (\forall y : ((person(y) \wedge \neg shave(y,y)) \rightarrow shave(x,y)))]$$

$$F_2\colon\ \forall x : [barber(x) \rightarrow (\forall y : ((person(y) \wedge shave(y,y)) \rightarrow \neg shave(x,y)))]$$

$$F_3\colon\ \forall x : (barber(x) \rightarrow person(x))$$

In plain words, they represent the following knowledge:

F_1: Every barber shaves every person who does not shave himself.

F_2: Every barber does not shave a person who shaves himself.

F_3: A barber is a person.

Let us consider a proof that this knowledge entails the nonexistence of any barber, i.e., consider the proof problem $\langle E_1, E_2 \rangle$, where $E_1 = F_1 \wedge F_2 \wedge F_3$ and E_2 is the first-order formula $\neg(\exists x : barber(x))$, which intuitively means no barber exists.

Conversion of $(E_1 \wedge \neg E_2)$, which is $((F_1 \wedge F_2 \wedge F_3) \wedge \neg E_2)$, by MPS yields a set Cs consisting of the following four extended clauses, where f is a 0-ary function variable:

C_1: $shave(x,x), shave(y,x) \leftarrow barber(y), person(x)$
C_2: $\leftarrow barber(x), person(y), shave(y,y), shave(x,y)$
C_3: $person(x) \leftarrow barber(x)$
C_4: $barber(x) \leftarrow func(f,x)$

By the application of the FVE rule, Cs is transformed into $Cs' = (Cs - \{C_4\}) \cup \{C_4'\}$, where C_4' is given by:

C_4': $barber(f) \leftarrow$

The resulting clause set Cs' can then be transformed equivalently using ET rules as follows:

- By successively unfolding at *barber*-atoms three times, removal of the definite clause defining the *barber* predicate, unfolding at *person*-atoms two times, and removal of the definite clause defining the *person* predicate, Cs' is transformed into Cs_1' consisting of the following two clauses:

  $shave(f,f), shave(f,f) \leftarrow$
  $\leftarrow shave(f,f), shave(f,f)$

- By removing duplicate atoms, Cs_1' is transformed into Cs_2' consisting of the two clauses:

  $shave(f,f) \leftarrow$
  $\leftarrow shave(f,f)$

- By unfolding at the *shave*-atom in the second clause above, Cs_2' is transformed into Cs_3' consisting of the two clauses:

  $shave(f,f) \leftarrow$
  $(\leftarrow)$

Finally, a clause set $Cs'' = Cs_3'$ is obtained. Since Cs'' contains an empty clause, $ans_{\mathrm{qa}}(Cs'', yes) = \{yes\}$. Hence $ans_{\mathrm{pr}}(E_1, E_2) =$ "yes".

6 LIMITATIONS BY EXAMPLES

The FVE rule, introduced in Section 5.2, has the following two limitations:

1. Its applicability does not cover all QA problems. It is applicable to a specific class of QA problems that corresponds to a class of proof problems.
2. It is not applicable to QA problems with built-in constraint atoms.

These limitations are demonstrated by means of examples below.

6.1 Incorrect Results when Applied to QA Problems

We show that application of the FVE rule to a QA problem may result in an incorrect result. In particular, after illustrating a QA problem on first-order logic (Section 6.1.1) and its equivalent QA problem in a clausal form (Section 6.1.2), we show that the corresponding QA problem obtained by applying the FVE rule gives an undesirable result (Section 6.1.3).

6.1.1 A QA Problem on First-order Logic

Assuming that A and B are constant symbols, consider a QA problem $\langle K, a \rangle$ on first-order logic, where K is the first-order formula

$$(\exists x : kill(x,A)) \wedge (kill(A,A) \vee kill(B,A))$$

and a is the atom $kill(x,A)$. Since $(\exists x : kill(x,A))$ follows logically from $(kill(A,A) \vee kill(B,A))$, this QA problem is equivalent to the QA problem $\langle K', a \rangle$, where $K' = (kill(A,A) \vee kill(B,A))$. Among others, $Models(K')$ contains the models

- $\{kill(A,A)\}$,
- $\{kill(B,A)\}$,
- $\{kill(A,A), kill(B,A)\}$.

Hence $\bigcap Models(K') = \varnothing$. Therefore $ans_{\mathrm{qa}}(K', a)$ is the empty set. So is $ans_{\mathrm{qa}}(K, a)$.

6.1.2 A QA Problem on Clauses with Function Variables

By MPS, K is converted into a clause set Cs_1 consisting of the two clauses, where f is a 0-ary function variable:

$kill(x,A) \leftarrow func(f,x)$
$kill(A,A), kill(B,A) \leftarrow$

The conversion yields the QA problem $\langle Cs_1, a \rangle$ in a clausal form. The set of all models of Cs_1, i.e., $Models(Cs_1)$, contains the models

- $\{kill(A,A)\}$,
- $\{kill(B,A)\}$,
- $\{kill(A,A), kill(B,A)\}$.

So $\bigcap Models(Cs_1) = \varnothing$. Hence $ans_{\mathrm{qa}}(Cs_1, a)$ is the empty set, which is the same as $ans_{\mathrm{qa}}(K, a)$ in Section 6.1.1.

6.1.3 A QA Problem Obtained by Applying the FVE Rule

By application of the FVE rule to Cs_1, with a function symbol f_h corresponding to the function variable f, we obtain a QA problem $\langle Cs_2, a\rangle$, where Cs_2 consists of the following two clauses:

$kill(f_h, A) \leftarrow$
$kill(A, A), kill(B, A) \leftarrow$

$Models(Cs_2)$ contains the models

- $\{kill(f_h, A), kill(A, A)\}$,
- $\{kill(f_h, A), kill(B, A)\}$,
- $\{kill(f_h, A), kill(A, A), kill(B, A)\}$,

along with other models each of which contains $kill(f_h, A)$. Then $\bigcap Models(Cs_2) = \{kill(f_h, A)\}$. As a result, $ans_{\mathrm{qa}}(Cs_2, a) = \{kill(f_h, A)\}$. This answer differs from $ans_{\mathrm{qa}}(K, a)$ in Section 6.1.1 and $ans_{\mathrm{qa}}(Cs_1, a)$ in Section 6.1.2, and is incorrect due to inappropriate use of the FVE rule.

6.2 Incorrect Results in the Presence of Built-in Constraint Atoms

Next, we show that in the presence of a built-in constraint atom, application of the FVE rule even to a proof problem may yield an incorrect result. For this purpose, we use the tax-cut proof problem with a constraint atom on first-order logic given in Section 1 and its equivalent proof problem on extended clauses (Section 6.2.1). Then we show that, compared with the (correct) answers to these proof problems, the corresponding proof problem obtained by applying the FVE rule gives an incorrect result (Section 6.2.2).

6.2.1 A Proof Problem on Clauses with Function Variables

Consider the proof problem $\langle E_1, E_2\rangle$ in Section 1, where $E_1 = F_1 \wedge F_2 \wedge F_3$ and $E_2 = TaxCut(Peter)$. By applying MPS to the conjunction $E_1 \wedge \neg E_2$, a 0-ary function variable, say f, is introduced and $E_1 \wedge \neg E_2$ is converted into a clause set Cs_3 consisting of the following clauses:

C_1: $TaxCut(x) \leftarrow hasChild(x, y), hasChild(x, z), noteq(y, z)$
C_2: $hasChild(Peter, Paul) \leftarrow$
C_3: $hasChild(Peter, x) \leftarrow func(f, x)$
C_4: $\leftarrow TaxCut(Peter)$

Semantically, Cs_3 is equivalent to $\{C_1, C_2, C_4\}$, which does not yield a contradiction. The expected result "no" shown in Section 1 is thus well supported.

If we simplify Cs_3 by equivalent transformation without using the FVE rule, Cs_3 is transformed as follows:

- By unfolding of *hasChild*, Cs_3 is transformed into a clause set $\{C_4, C_5, C_6, C_7, C_8\}$, where C_5–C_8 are given by:

  C_5: $TaxCut(Peter) \leftarrow noteq(Paul, Paul)$
  C_6: $TaxCut(Peter) \leftarrow noteq(Paul, x), func(f, x)$
  C_7: $TaxCut(Peter) \leftarrow noteq(x, Paul), func(f, x)$
  C_8: $TaxCut(Peter) \leftarrow noteq(x, y), func(f, x), func(f, y)$

- Since $noteq(Paul, Paul)$ is false, C_5 is removed.
- Since $func(f, x)$ and $func(f, y)$ give $x = y$ and $noteq(x, x)$ is false, C_8 is removed.

$\{C_4, C_6, C_7\}$ is obtained, which includes the function variable f. When letting $f() = Paul$, C_6 and C_7 are removed and the resulting clause set is $\{C_4\}$, which is satisfiable. So $\{C_4, C_6, C_7\}$ is satisfiable, and we cannot obtain a contradiction. This transformation result shows clearly that the answer to this proof problem is "no".

6.2.2 A Proof Problem Obtained by Applying the FVE Rule

Assume that f_h is a function symbol that corresponds to the function variable f. By application of the FVE rule to Cs_3, we have a clause set $Cs_4 = \{C_1, C_2, C_3', C_4\}$, where C_3' is given by:

C_3': $hasChild(Peter, f_h) \leftarrow$

It can be shown that Cs_4 yields a contradiction by repeatedly applying equivalent transformation as follows:

- After application of unfolding three times at *hasChild*-atoms, Cs_4 is transformed into a clause set $\{C_2, C_3', C_4, C_5', C_6'\}$, where C_5' and C_6' are given by:

  C_5': $TaxCut(Peter) \leftarrow noteq(Paul, f_h)$
  C_6': $TaxCut(Peter) \leftarrow noteq(f_h, Paul)$

- Since the constraint atoms in their bodies, i.e., $noteq(Paul, f_h)$ and $noteq(f_h, Paul)$, are true, C_5' and C_6' are transformed into:

  C_5'': $TaxCut(Peter) \leftarrow$
  C_6'': $TaxCut(Peter) \leftarrow$

From the final clause set, which contains C_4, C_5'', and C_6'', a contradiction is obtained. This contradiction

misleads us that the answer to this proof problem is "yes", which is incorrect compared to the expected result given in Section 1 and the result obtained in Section 6.2.1. Such inappropriate use of the FVE rule yields a wrong answer.

7 DISCUSSION

7.1 Incorrect Results Arising from Built-in Atoms

Incorrect results caused by the conventional Skolemization and resolution stems basically from the limitation of Skolemization. In particular:

1. If there is no built-in atom in an input first-order formula, the composition of MPS and FVE gives a set of usual clauses, which is also obtained by the conventional Skolemization.
2. If there is a built-in atom in an input first-order formula, MPS preserves its logical meaning but FVE cannot guarantee the preservation of its logical meaning, which means that the conventional Skolemization may produce an incorrect result.

The tax-cut example shown earlier (cf. Section 1 and Section 6.2) contains a built-in atom $noteq(y,z)$, and the fact that $noteq(Paul, f)$ is true for any new function symbol f adopted by the conversion algorithm CS (cf. Fig. 1) produces an incorrect result.

More generally, the maximality of the instantiation of function variables according to the usual Skolemization is broken if a ground atom set contains arbitrary built-in atoms. Hence Theorem 8 cannot be obtained.

7.2 Understanding the Conventional Skolemization and Resolution

The theory developed in the previous sections provides a deep understanding of the conventional Skolemization and resolution. We have stated (in Section 1) that the conventional Skolemization, CS, is identified as the composition of MPS and FVE in the sense that

$$\{\mathrm{CS}(L) \mid L \in \mathcal{L}_1\} = \{\mathrm{FVE}(\mathrm{MPS}(L)) \mid L \in \mathcal{L}_1\}.$$

This is a simplified explanation for the purpose of readability. More precisely, we need to consider that (i) by MPS, some function variables may be newly introduced, and (ii) by FVE, some function symbols may be associated with function variables. The conventional Skolemization may also introduce some new function symbols. Then we have the following proposition based on the theory in the previous sections: (i) each set of clauses obtained by sequentially applying MPS and FVE to a first-order formula, with any selection of function variables and a mapping to associate function variables with function symbols, can also be obtained by application of the conventional Skolemization using some function symbols, and (ii) vice versa.

Limitations of the conventional Skolemization are thus identified mainly by the limitations of FVE transformation. FVE preserves the answers to proof problems (Theorem 9); however, its applicability is rather limited:

1. FVE does not admit inclusion of built-in atoms in input extended clauses in ECLS_F, and
2. FVE can be applied to only a restricted class of QA problems.

This is a sharp contrast to most of important ET transformations, such as unfolding, resolution, factoring, subsumption, side changing, and definite-clause removal transformation, which have been invented on the space of extended clauses (Akama and Nantajeewarawat, 2014). These ET transformations preserve the answers to QA problems and can be applied to any QA problem possibly with built-in constraint atoms.

One limitation of the resolution method is that it uses only resolution and factoring. Obviously, from the viewpoint of ET, we can use other ET rules and we should use them for more efficient computation.

7.3 Inventing a New Proof Method

Since proof problems are formalized as QA problems, the MPS & ET method can be applied to proof problems. A new procedure for solving proof problems is given below.

Let $\langle E_1, E_2\rangle$ be an input proof problem.

1. Construct a QA problem $\langle E, yes\rangle$, where
   - $E = E_1 \wedge \neg E_2$, and
   - yes is a ground atom that does not appear in E.
2. Convert the QA problem $\langle E, yes\rangle$ by MPS, using the conversion procedure in Section 3.2, into a QA problem $\langle Cs, yes\rangle$ in a clausal form.
3. Using ET rules, transform $\langle Cs, yes\rangle$ into a QA problem $\langle Cs', yes\rangle$.
4. Determine the answer to the proof problem $\langle E_1, E_2\rangle$ by

$$ans_{\mathrm{pr}}(E_1, E_2) = \begin{cases} \text{"yes"} & \text{if } ans_{\mathrm{qa}}(Cs', yes) = \{yes\}, \\ \text{"no"} & \text{if } ans_{\mathrm{qa}}(Cs', yes) = \varnothing. \end{cases}$$

By ET transformation at Step 3, we may basically try to simplify *Cs* using ET rules. When we reach a set of extended clauses that contains an empty clause ($\leftarrow$), we can stop with the answer "yes". When we reach a set of positive extended clauses, we can stop with the answer "no". All ET rules, which preserve the answers to QA problems, can be used at Step 3. The reader may refer to examples given in (Akama and Nantajeewarawat, 2012; Akama and Nantajeewarawat, 2013; Akama and Nantajeewarawat, 2014).

8 CONCLUSIONS

The MPS & ET theory takes MPS in place of the conventional Skolemization and ET in place of inference rules. The work developed in this paper enables us to consider the conventional Skolemization and the conventional solution for proof problems in the MPS & ET theory.

In this paper, the FVE rule has been proposed and its correctness has been proved (i.e., the FVE rule preserves the answers to a class QA problems). The conventional Skolemization is identified as application of MPS transformation followed by equivalent transformation using the FVE rule. Since the resolution and factoring inference rules are ET rules and proof problems are a subclass of QA problems, the conventional solution for proof problems is a special case of the MPS & ET solution for QA problems.

This paper has also investigated the limitations of the FVE rule, which are also limitations of the conventional Skolemization and the conventional solution for proof problems. They are:

1. The conventional Skolemization may fail to preserve satisfiability of a given formula in the presence of built-in constraints.
2. The conventional solution for proof problems based on Skolemization and resolution cannot guarantee the correctness of an obtained answer when built-in constraints are included in a given problem representation.

By removing the FVE rule, which is less general than other important ET rules, we have a new proof method with correctness of computation being guaranteed even in the presence of built-in constraints.

ACKNOWLEDGEMENTS

This research was partially supported by JSPS KAKENHI Grant Numbers 25280078 and 26540110.

REFERENCES

Akama, K. and Nantajeewarawat, E. (2011a). Meaning-Preserving Skolemization. In *Proceedings of the 3rd International Conference on Knowledge Engineering and Ontology Development*, pages 322–327, Paris, France.

Akama, K. and Nantajeewarawat, E. (2011b). Meaning-Preserving Skolemization. Technical report, Hokkaido University, Sapporo, Japan.

Akama, K. and Nantajeewarawat, E. (2012). Proving Theorems Based on Equivalent Transformation Using Resolution and Factoring. In *Proceedings of the Second World Congress on Information and Communication Technologies*, WICT 2012, pages 7–12, Trivandrum, India.

Akama, K. and Nantajeewarawat, E. (2013). Embedding Proof Problems into Query-Answering Problems and Problem Solving by Equivalent Transformation. In *Proceedings of the 5th International Conference on Knowledge Engineering and Ontology Development*, pages 253–260, Vilamoura, Portugal.

Akama, K. and Nantajeewarawat, E. (2014). Equivalent Transformation in an Extended Space for Solving Query-Answering Problems. In *Proceedings of the 6th Asian Conference on Intelligent Information and Database Systems*, LNAI 8397, pages 232–241, Bangkok, Thailand.

Akama, K. and Nantajeewarawat, E. (2015). Function-Variable Elimination and Its Limitations. Technical report, Hokkaido University, Sapporo, Japan.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Motik, B., Sattler, U., and Studer, R. (2005). Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, 3(1):41–60.

Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23–41.