Computation Control by Prioritized ET Rules

Kiyoshi Akama[1], Ekawit Nantajeewarawat[2] and Taketo Akama[3]
[1]*Information Initiative Center, Hokkaido University, Sapporo, Japan*
[2]*Computer Science, Sirindhorn International Institute of Technology, Thammasat University, Pathumthani, Thailand*
[3]*Modeleet Labs, Sapporo, Japan*

Keywords: Model-Intersection Problem, Equivalent Transformation, Function Variable, Rule Priority, Computation Control, Proof Problem, Query-Answering Problem.

Abstract: Model-intersection problems have been invented as one of the largest classes of logical problems. By solving MI problems, we can solve proof problems and query-answering problems on first-order logic. For solving MI problems on extended clauses, we propose in this paper prioritized equivalent transformation (ET) rules. A set R of ET rules with priority ordering is employed, and at each computation step an applicable ET rule with best priority in R is selected and applied. This method can be used to decrease a search space by introducing new rules and adjusting rule priority, and is useful to solve a large class of logical problems with the guarantee of strict correctness of computation results.

1 INTRODUCTION

A proof problem is a "yes/no" problem; it is concerned with checking whether or not one given logical formula entails another given logical formula. A query-answering (QA) problem is an "all-answers finding" problem, i.e., finding all ground instances of a given query atom that are logical consequences of a given formula. Much research work in the logic programming and semantic web communities has addressed subclasses of proof problems or QA problems.

Model-intersection (MI) problems have been invented as one of the largest classes of logical problems (Akama and Nantajeewarawat, 2016a). All proof problems and all QA problems on first-order formulas can be mapped, with the answers to them being preserved, into MI problems on an extended clause space. By solving MI problems, we can solve proof problems and QA problems on first-order logic.

The resolution principle is a methodology for solving proof problems. Starting from a set of clauses, computation by resolution is an inference process that adds a new clause as a resolvent of two existing clauses at each inference step. A resolvent is obtained from two clauses and two atoms occurring in them. At each computation step, selection of two clauses and two atoms determines a new resolvent. Clause and atom selection constitutes computation control, which typically affects computation efficiency. One computation control may give the shortest (and finite) path to the end of computation, while another computation control may result in non-termination, i.e., never-ending computation with no answer being obtained.

A general method for solving MI problems on extended clauses is equivalent transformation (ET), where problems are solved by repeated problem simplification using ET rules. Efficiency of computation is basically determined by (i) a set R of ET rules used for the computation, and (ii) a selection of an ET rule in R at each computation step. Depending to such computation control, an ET sequence may reach a final problem description in finite steps or may produce an infinite sequence without giving any answer to the original problem.

This paper proposes computation control to efficiently find an ET sequence from an initial problem description to a final problem description. We use prioritized ET rules to find better selection of ET rules. We also compare computation control in the conventional resolution method with our prioritized ET rules.

The rest of the paper is organized as follows: Section 2 discusses insufficiency of the conventional theory. It also explains important new concepts of an extended clause space, meaning-preserving Skolemization, and ET rules. Section 3 begins our theory by introducing our extended clause space, interpretations, and models. Section 4 defines MI pro-

Akama, K., Nantajeewarawat, E. and Akama, T.
Computation Control by Prioritized ET Rules.
DOI: 10.5220/0006934600840095
In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 2: KEOD*, pages 84-95
ISBN: 978-989-758-330-8

blems and presents a solution method based on ET. Section 5 gives a simple example for showing problem formalization and a solution, and illustrating the effects of computation control. Section 6 proposes prioritized ET rules for computation control. Correctness of computation with prioritized ET rules is also proved. Section 7 describes a method for computation control with prioritized ET rules. Section 8 explains ET rules used in this paper mainly by examples. Section 9 provides conclusions.

The notation that follows holds thereafter. Given a set A, $pow(A)$ denotes the power set of A. Given two sets A and B, $Map(A,B)$ denotes the set of all mappings from A to B, and for any partial mapping f from A to B, $dom(f)$ denotes the domain of f, i.e., $dom(f) = \{a \mid (a \in A)\ \&\ (f(a)\ \text{is defined})\}$.

2 INSUFFICIENCY OF THE CONVENTIONAL THEORY

2.1 Incompleteness of the Usual Clause Space

Let CLS be the set of all clauses consisting only of user-defined atoms, and CLS_c the set of all clauses consisting of user-defined atoms and built-in atoms. Corresponding to these, let FOL be the set of all first-order formulas consisting only of user-defined atoms, and FOL_c the set of all first-order formulas consisting of user-defined atoms and built-in atoms.

Let SKO be a mapping such that each first-order formula in FOL_c is transformed into a set of clauses in CLS_c by SKO using conventional Skolemization and other ET rules. It is well-known that SKO transforms each first-order formula in FOL into a set of clauses in CLS preserving satisfiability. This enables conventional resolution-based theorem proving, which motivates us to consider SKO and CLS as a foundation for logical problem solving.

However, we would like to stress that SKO and CLS have serious limitations:

- SKO does not generally preserve the logical meanings of formulas in FOL and those in FOL_c.
- Existential quantification cannot be represented by clauses in CLS nor those in CLS_c.
- SKO does not generally preserve satisfiability for FOL_c.

Thus CLS and CLS_c are not appropriate for entirely solving all proof problems, QA problems, and MI problems on FOL and FOL_c.

2.2 A New Extended Clause Space

Conventional clauses are not sufficiently expressive for equivalently representing first-order formulas since all variables in a clause are universally quantified and no existential quantification is allowed. Instead of the usual clause space, we use an extended clause space, called the ECLS_F space, in which a clause may contain three kinds of atoms: built-in constraint atoms, user-defined atoms, and *func*-atoms. Variables of a new type, called *function variables*, appear in *func*-atoms in the positions of their first arguments, and are existentially quantified at the top level of a clause set under consideration. Existential quantification of usual variables in first-order logic is alternatively represented by existential quantification of function variables in ECLS_F.

2.3 Model-Intersection Problems

A proof problem is concerned with checking whether one given logical formula entails another given logical formula. The proof problems solved by conventional Skolemization and resolution are on FOL, not FOL_c. However, the class of proof problems on FOL is not sufficient for practical use, since it cannot deal with most of useful built-in constraint atoms.

A query-answering (QA) problem is concerned with finding all ground instances of a given query atom that are logical consequences of a given formula. The logic programming community and the semantic web community consider QA problems together with proof problems. However, they deal with only subclasses of QA problems. No general theory of QA problems on FOL_c has been constructed.

The class of model-intersection problems (MI problems) has been invented for constructing a general theory of logical problem solving (Akama and Nantajeewarawat, 2016a). MI problems enable us to construct a unified theory of logical problem solving. A MI problem on extended clauses is a pair of a set of extended clauses and an extraction mapping. The answer to a MI problem is the value obtained by applying its extraction mapping to the intersection of all the models of the conjunction of its extended clauses. MI problems constitute a very large class of logical problems, and include both proof problems and QA problems (see Section 2.4 and Fig. 1).

2.4 Meaning-Preserving Skolemization

The usual clause space taken by conventional logic programming is too small to consider all proof problems on FOL_c and all QA problems on FOL_c. These

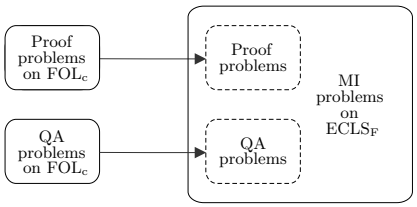

Figure 1: Embedding logical problems into MI problems.

difficulties are overcome by meaning-preserving Skolemization (MPS) (Akama and Nantajeewarawat, 2011) and the extended clause space $ECLS_F$. In particular:

- MPS preserves the logical meanings of formulas in FOL and those in FOL_c.
- Existential quantification can be represented by clauses in $ECLS_F$.
- MPS preserves satisfiability of formulas in FOL and those in FOL_c.

As depicted by Fig. 1, all proof problems and all QA problems on FOL_c are mapped, preserving their answers, into MI problems on $ECLS_F$ (Akama and Nantajeewarawat, 2016a). By solving MI problems on $ECLS_F$, we can solve proof problems and QA problems on FOL_c. The $ECLS_F$ has sufficient knowledge representation power for dealing with these problems. This is the fundamental reason why we should take the $ECLS_F$ space in place of the usual clause space.

2.5 Equivalent Transformation

A method for solving MI problems on $ECLS_F$ by equivalent transformation (ET) has been proposed (Akama and Nantajeewarawat, 2016a), where problems are solved by repeated problem simplification using ET rules. ET in our theory is more general than inference in first-order logic. Computation by inference rules is an instance of ET computation, since inference rules (e.g., the resolution and factoring inference rules) are special kinds of ET rules. This means that a resolution method for proving logical formulas is an instance of an ET solution method.

In our theory, ET rules are first-class citizens. By contrast, in logic programming, clauses are regarded as rules. For example, definite clauses are regarded as rules in Prolog, and CHR rules are regarded as formulas in the CHR theory. For understanding and generation of various algorithms and procedures, it is essential to consider ET rules as first-class citizens in the theory of logical problem solving. Note that "rules = logical formulas" in conventional logic programming is categorical mismatching since rules are procedural whereas logical formulas are declarative. The slogan "rules = logical formulas" should not be theoretically sound. In order to develop a general foundation, we need to avoid such categorical mismatching.

2.6 Insufficiency of Turing Completeness

Most of logic programming research uses subspaces of $\mathrm{CLS_c}$, i.e., conventional logic programs are sets of normal clauses and provide no representation power of existential quantification. A general framework of solving logical problems on FOL_c is thus not provided. This limitation does not contradict the Turing completeness of a logic programming language, e.g., Prolog. Turing completeness of a programming language does not mean that everything can be done using that language. Most programming languages are Turing complete.

A programming language is said to be Turing complete if it can be used to simulate any computable function. Our problem in this paper, however, is not to simulate known procedures, but to invent procedures for giving correct solutions to MI problems. Such invention is not an easy task. Once a procedure is invented, however, a simulation of it is rather an easy task.

3 AN EXTENDED CLAUSE SPACE

3.1 Built-in Constraint Atoms and User-Defined Atoms

We consider an extended formula space that contains three kinds of atoms, i.e., built-in constraint atoms, user-defined atoms, and *func*-atoms.

A *built-in constraint atom*, also simply called a *constraint atom* or a *built-in atom*, takes the form $c(t_1,\ldots,t_n)$, where c is a predefined constraint predicate and the t_i are usual terms. It is a ground built-in constraint atom if the t_i are all ground (variable free). Built-in atoms are essential for representation of knowledge using first-order formulas. Most practical problems cannot be represented without built-in constraint atoms such as equality, inequality, and arithmetic constraints. The meanings of built-in atoms are defined by specifying the set of all true ground built-in atoms. For example:

- $(s = t)$ is true iff s and t are the same ground terms.
- $(s \neq t)$ is true iff s and t are not the same ground terms.

- $(s := t_1 - t_2)$ is true iff s, t_1, and t_2 are numbers and s is equal to $t_1 - t_2$.

A *user-defined atom* takes the form $p(t_1,\ldots,t_n)$, where p is a user-defined predicate and the t_i are usual terms. It is a ground user-defined atom if the t_i are all ground (variable free). The meanings of ground user-defined atoms are determined by an interpretation.

Let $\mathcal{A}_\mathrm{u}$ be the set of all user-defined atoms, $\mathcal{G}_\mathrm{u}$ the set of all ground user-defined atoms, $\mathcal{A}_\mathrm{c}$ the set of all constraint atoms, and $\mathcal{G}_\mathrm{c}$ the set of all ground constraint atoms.

3.2 Variables and *func*-Atoms

There are two types of variables: usual variables and function variables. A function variable may only appear as the first argument of a *func*-atom. A function variable is instantiated into a function constant or a function variable, but not into a usual term.

A *func-atom* (Akama and Nantajeewarawat, 2011) is an expression of the form $\mathit{func}(f,t_1,\ldots,t_n,t_{n+1})$, where f is either an n-ary function constant or an n-ary function variable, and the t_i are usual terms. It is a *ground func*-atom if f is a function constant and the t_i are ground usual terms.

3.3 Extended Clauses

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

$$a_1,\ldots,a_m \leftarrow b_1,\ldots,b_n,\mathbf{f}_1,\ldots,\mathbf{f}_p,$$

where each of $a_1,\ldots,a_m,b_1,\ldots,b_n$ is a user-defined atom or a built-in constraint atom, and $\mathbf{f}_1,\ldots,\mathbf{f}_p$ 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 $\{a_1,\ldots,a_m\}$ and $\{b_1,\ldots,b_n,\mathbf{f}_1,\ldots,\mathbf{f}_p\}$ 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. Let $\mathit{userLhs}(C)$ denote the number of user-defined atoms in the left-hand side of C. When $\mathit{userLhs}(C) = 0$, C is called a *negative extended clause*. When $\mathit{userLhs}(C) = 1$, C is called an *extended definite clause*. When $\mathit{userLhs}(C) > 1$, C is called a *multi-head extended clause*. Given an extended definite clause C, the user-defined atom in $\mathit{lhs}(C)$ is called the *head* of C, denoted by $\mathit{head}(C)$, and the set $\mathit{rhs}(C)$ is called the *body* of C, denoted by $\mathit{body}(C)$.

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.

3.4 An Extended Clause Space

A conjunction of a finite or infinite number of extended clauses is used for knowledge representation and also for computation. As usual, such a conjunction is usually dealt with by regarding it as a set of extended clauses. The set of all extended clauses is denoted by $\mathrm{ECLS_F}$. The *extended clause space* in this paper is the powerset of $\mathrm{ECLS_F}$.

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 occurring in Cs into function constants, clauses in the instantiated set are totally separated.

3.5 Interpretations and Models

An *interpretation* is a subset of $\mathcal{G}_\mathrm{u}$. A ground user-defined atom g is true under 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. Let TCON denote the set of all true ground constraint atoms, i.e., a ground constraint atom g is true iff $g \in \mathrm{TCON}$. A ground *func*-atom $\mathit{func}(f,t_1,\ldots,t_n,t_{n+1})$ is true iff $f(t_1,\ldots,t_n) = t_{n+1}$.

A ground clause $C = (a_1,\ldots,a_m \leftarrow b_1,\ldots,b_n, \mathbf{f}_1,\ldots,\mathbf{f}_p) \in \mathrm{ECLS_F}$, where $\{a_1,\ldots,a_m,b_1,\ldots,b_n\} \subseteq \mathcal{G}_\mathrm{u} \cup \mathcal{G}_\mathrm{c}$ and $\mathbf{f}_1,\ldots,\mathbf{f}_p$ are ground *func*-atoms, is true under an interpretation I (in other words, I *satisfies* C) iff at least one of the following conditions is satisfied:

1. There exists $i \in \{1,\ldots,m\}$ such that $a_i \in I \cup \mathrm{TCON}$.
2. There exists $i \in \{1,\ldots,n\}$ such that $b_i \notin I \cup \mathrm{TCON}$.
3. There exists $i \in \{1,\ldots,p\}$ such that $\mathbf{f}_i$ is false.

Given $Cs \subseteq \mathrm{ECLS_F}$ and a substitution σ for function variables, 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 using σ.

An interpretation I is a *model* of a clause set $Cs \subseteq \mathrm{ECLS_F}$ iff there exists a substitution σ for function variables that satisfies the following conditions:

1. All function variables occurring in Cs are instantiated by σ into function constants.
2. For any clause $C \in Cs$ and any substitution θ for usual variables, if $C\sigma\theta$ is a ground clause, then $C\sigma\theta$ is true under I.

Let *Models* be a mapping that associates with each clause set the set of all of its models, i.e., $Models(Cs)$ is the set of all models of Cs for any $Cs \subseteq \mathrm{ECLS_F}$.

4 SOLVING MI PROBLEMS BY EQUIVALENT TRANSFORMATION (ET)

4.1 MI Problems on $\mathrm{ECLS_F}$

A *model-intersection problem* (for short, *MI problem*) on $\mathrm{ECLS_F}$ is a pair $\langle Cs, \varphi \rangle$, where $Cs \subseteq \mathrm{ECLS_F}$ and φ is a mapping from $pow(\mathcal{G}_\mathrm{u})$ to some set W. The mapping φ is called an *extraction mapping*. The answer to this problem, denoted by $ans_\mathrm{MI}(Cs, \varphi)$, is defined by

$$ans_\mathrm{MI}(Cs, \varphi) \;=\; \varphi(\bigcap Models(Cs)),$$

where $\bigcap Models(Cs)$ is the intersection of all models of Cs. Note that when $Models(Cs)$ is the empty set, $\bigcap Models(Cs) = \mathcal{G}_\mathrm{u}$.

Example 1. Assume that Cs consists of the following four clauses:

$pat(oe) \leftarrow$
$prob(io), pat(po) \leftarrow$
$prob(io) \leftarrow pat(po)$
$prob(oe) \leftarrow pat(po)$

Consider a MI problem $\langle Cs, \varphi \rangle$, where for any $G \subseteq \mathcal{G}_\mathrm{u}$, $\varphi(G) = \{x \mid prob(x) \in G\}$. Obviously,

- $M_1 = \{pat(po), prob(io), prob(oe), pat(oe)\}$ is a model of Cs, and
- $M_2 = \{prob(io), pat(oe)\}$ is also a model of Cs.

Assume that M is a model of Cs. Two cases are considered:

- *Case 1:* $pat(po) \in M$. By the last two clauses in Cs, $prob(io)$ and $prob(oe)$ are true. By the first clause in Cs, $pat(oe)$ is true. Hence $M \supseteq M_1$.
- *Case 2:* $pat(po) \notin M$. By the second clause in Cs, $prob(io)$ is true. By the first clause in Cs, $pat(oe)$ is true. Hence $M \supseteq M_2$.

So $\bigcap Models(Cs) = \{prob(io), pat(oe)\}$, and thus $ans_\mathrm{MI}(Cs, \varphi) = \{io\}$. □

4.2 Answer Mappings

An answer mapping is a partial mapping that gives the answer to an MI problem whenever it is applicable to that problem. When a problem description reaches the domain of an answer mapping, we compute the answer by applying the answer mapping to the final problem description.

Definition 1. Let W be a set. A partial mapping A from

$$pow(\mathrm{ECLS_F}) \times Map(pow(\mathcal{G}_\mathrm{u}), W)$$

to W is an *answer mapping* iff for any $\langle Cs, \varphi \rangle \in dom(A)$, $A(Cs, \varphi) = ans_\mathrm{MI}(Cs, \varphi)$. □

For example, suppose that a proof problem is transformed into an MI problem $\langle Cs, \varphi \rangle$, where φ is an extraction mapping such that for any $G \subseteq \mathcal{G}_\mathrm{u}$, $\varphi(G) =$ "yes" if $G = \mathcal{G}_\mathrm{u}$, and $\varphi(G) =$ "no" otherwise. Then we can use the answer mapping A defined by: for any $Cs' \subseteq \mathrm{ECLS_F}$ and any $\varphi' \in Map(pow(\mathcal{G}_\mathrm{u}), \{\text{"yes"}, \text{"no"}\})$,

$$A(Cs', \varphi') = \begin{cases} \text{"yes"} & \text{if } Cs' \text{ contains the empty clause } (\leftarrow) \text{ and } \varphi' = \varphi, \\ \text{"no"} & \text{if } Cs' \text{ contains no negative clause and } \varphi' = \varphi, \end{cases}$$

and it is undefined otherwise. When $A(Cs', \varphi')$ is defined, it is equal to $ans_\mathrm{MI}(Cs', \varphi')$ since

- if Cs' contains the empty clause, then $Models(Cs') = \varnothing$ and, thus, $\bigcap Models(Cs') = \mathcal{G}_\mathrm{u}$, and
- if Cs' contains no negative clause, then there is a model M of Cs' such that M is a proper subset of $\mathcal{G}_\mathrm{u}$ and, thus, $\bigcap Models(Cs')$ is a proper subset of $\mathcal{G}_\mathrm{u}$.

4.3 ET Steps and ET Rules

Next, a method for solving MI problems based on ET, preserving their answers, is formulated.

Let STATE be the set of all MI problems. Elements of STATE are called *states*.

Definition 2. Let $\langle S, S' \rangle \in \mathrm{STATE} \times \mathrm{STATE}$. $\langle S, S' \rangle$ is an *ET step* iff if $S = \langle Cs, \varphi \rangle$ and $S' = \langle Cs', \varphi' \rangle$, then $ans_\mathrm{MI}(Cs, \varphi) = ans_\mathrm{MI}(Cs', \varphi')$. □

Definition 3. A sequence $[S_0, S_1, \ldots, S_n]$ of elements of STATE is an *ET sequence* iff for any $i \in \{0, 1, \ldots, n-1\}$, $\langle S_i, S_{i+1} \rangle$ is an ET step. □

The role of ET computation constructing an ET sequence $[S_0, S_1, \ldots, S_n]$ is to start with S_0 and to reach S_n from which the answer to the given problem can be easily computed.

The concept of ET rule on STATE is defined by:

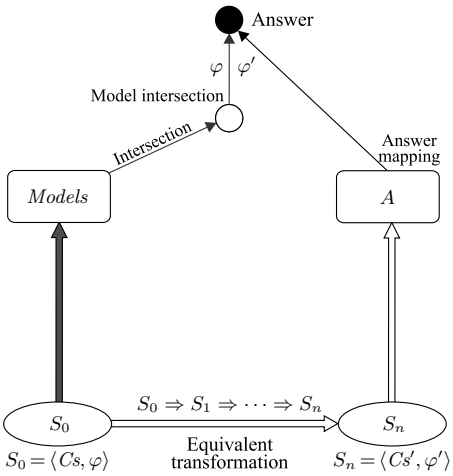

Figure 2: Computation paths are constructed by a combination of an ET sequence and application of an answer mapping.

Definition 4. An *ET rule r* on STATE is a partial mapping from STATE to STATE such that for any $S \in dom(r)$, $\langle S, r(S) \rangle$ is an ET step. □

4.4 A Correct Solution Method based on ET Rules

A MI problem $\langle Cs, \varphi \rangle$, where $Cs \subseteq \text{ECLS}_\text{F}$ and φ is an extraction mapping, can be solved as follows:

1. Let A be an answer mapping.
2. Prepare a set R of ET rules on STATE.
3. Take S_0 such that $S_0 = \langle Cs, \varphi \rangle$ to start computation from S_0.
4. Construct an ET sequence $[S_0, \ldots, S_n]$ by applying ET rules in R, i.e., for each $i \in \{0, 1, \ldots, n-1\}$, S_{i+1} is obtained from S_i by selecting and applying $r_i \in R$ such that $S_i \in dom(r_i)$ and $r_i(S_i) = S_{i+1}$.
5. Assume that $S_n = \langle Cs_n, \varphi_n \rangle$. If the computation reaches the domain of A, i.e., $\langle Cs_n, \varphi_n \rangle \in dom(A)$, then compute the answer by using the answer mapping A, i.e., output $A(Cs_n, \varphi_n)$.

Given a set Cs of clauses and an extraction mapping φ, the answer to the MI problem $\langle Cs, \varphi \rangle$ is $\varphi(\bigcap Models(Cs))$, which is called the definition path and is represented by the left path in Fig. 2. The definition path is usually not suitable for computing the answer since it may take huge cost. Instead of taking this definition path, we take a computation path consisting of (i) the lowest path (from Cs to Cs') for ET computation and (ii) the right path (from Cs' through the answer mapping A upwards to the answer) in Fig. 2.

F_1: $\exists x : W(x)$
F_2: $\exists x : F(x)$
F_3: $\exists x : B(x)$
F_4: $\exists x : C(x)$
F_5: $\exists x : S(x)$
F_6: $\forall x : (W(x) \to A(x))$
F_7: $\forall x : (F(x) \to A(x))$
F_8: $\forall x : (B(x) \to A(x))$
F_9: $\forall x : (C(x) \to A(x))$
F_{10}: $\forall x : (S(x) \to A(x))$
F_{11}: $\exists x : G(x)$
F_{12}: $\forall x : (G(x) \to P(x))$
F_{13}: $\forall x : [A(x) \to [[\forall y : (P(y) \to E(x,y))] \vee [\forall y : (A(y) \wedge M(y,x) \wedge (\exists z : (E(y,z) \wedge P(z))) \to E(x,y))]]]$
F_{14}: $\forall x : (C(x) \to (\forall y : (B(y) \to M(x,y))))$
F_{15}: $\forall x : (S(x) \to (\forall y : (B(y) \to M(x,y))))$
F_{16}: $\forall x : (B(x) \to (\forall y : (F(y) \to M(x,y))))$
F_{17}: $\forall x : (F(x) \to (\forall y : (W(y) \to M(x,y))))$
F_{18}: $\forall x : (W(x) \to (\forall y : ((F(y) \vee G(y)) \to \neg E(x,y))))$
F_{19}: $\forall x : (B(x) \to (\forall y : (C(y) \to E(x,y))))$
F_{20}: $\forall x : (B(x) \to (\forall y : (S(y) \to \neg E(x,y))))$
F_{21}: $\forall x : (C(x) \to (\exists y : (P(y) \wedge E(x,y))))$
F_{22}: $\forall x : (S(x) \to (\exists y : (P(y) \wedge E(x,y))))$
F_{23}: $\exists x : [A(x) \wedge [\exists y : (A(y) \wedge [\exists z : (G(z) \wedge E(y,z))] \wedge E(x,y))]]$

Figure 3: Background knowledge represented by first-order formulas.

The selection of r_i in R at Step 4 is nondeterministic and there may be many possible ET sequences for each MI problem. Every output computed by using any arbitrary ET sequence is correct.

Theorem 1. *Let A be an answer mapping. When an ET sequence starting from $S_0 = \langle Cs, \varphi \rangle$ reaches S_n in $dom(A)$, the above procedure gives the correct answer to $\langle Cs, \varphi \rangle$.* □

5 A SIMPLE EXAMPLE

5.1 Schubert's Steamroller Puzzle

The Steamroller puzzle was presented by Lenhart Schubert in 1978 as a challenge to automated-deduction systems. It was considered to be too hard for existing theorem provers at that time due to its big search space. Much work has been done related to the Steamroller puzzle to improve efficiency of computation (Manthey and Bry, 1988; Pelletier, 1986; Stickel, 1986; Walther, 1985; Wang and Bledsoe, 1987). This is the reason why we take this puzzle in this paper. This example will be used in Section 7 to explain

computation control with prioritized ET rules and to illustrate the effect of computation control.

This puzzle is a proof problem. Formalization of the Steamroller puzzle as a proof problem based on the conventional theory is given in Section 5.3. Since proof problems can be transformed into MI problems according to our theory, we give a formulation of the Steamroller puzzle as a MI problem in Section 5.4.

5.2 Formalization

The problem description of the Steamroller puzzle is as follows: Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Therefore there is an animal that likes to eat a grain-eating animal.

Fig. 3 shows the first-order formulas representing this problem (Stickel, 1986). The last sentence of the description ("Therefore there is an animal that likes to eat a grain-eating animal") is regarded as a conclusion to be proved and is represented by the first-order formula F_{23} in Fig. 3. All sentences except the last one form the background knowledge and are represented by F_1–F_{22} in Fig. 3. This formalization uses the following predicates as abbreviation:

$A(x)$	– x is an animal	$W(x)$	– x is a wolf
$F(x)$	– x is a fox	$B(x)$	– x is a bird
$C(x)$	– x is a caterpillar	$S(x)$	– x is a snail
$G(x)$	– x is a grain	$P(x)$	– x is a plant

$M(x,y)$ – x is much smaller than y
$E(x,y)$ – x likes to eat y

5.3 Clausal Form and A Solution

Referring to F_1–F_{23} in Fig. 3, let $F = F_1 \wedge F_2 \wedge \cdots \wedge F_{22}$. According to the conventional theory, the Steamroller puzzle is first formalized as a proof problem $F \models F_{23}$, which is proved by showing that $F \wedge \neg F_{23}$ has no model, i.e., $Models(F \wedge \neg F_{23}) = \varnothing$. By using conventional Skolemization, $F \wedge \neg F_{23}$ is converted into a clause set Cs consisting of the clauses in Fig. 4, where w, f, b, c, s, and g are Skolem constants, and f_1 and f_2 are Skolem functions. Since $Models(F \wedge \neg F_{23}) = Models(Cs)$, we need to show that $Models(Cs) = \varnothing$. For this purpose the resolution and factoring inference rules can be used.

C_1: $W(w) \leftarrow$
C_2: $F(f) \leftarrow$
C_3: $B(b) \leftarrow$
C_4: $C(c) \leftarrow$
C_5: $S(s) \leftarrow$
C_6: $G(g) \leftarrow$
C_7: $A(x) \leftarrow W(x)$
C_8: $A(x) \leftarrow F(x)$
C_9: $A(x) \leftarrow B(x)$
C_{10}: $A(x) \leftarrow C(x)$
C_{11}: $A(x) \leftarrow S(x)$
C_{12}: $P(x) \leftarrow G(x)$
C_{13}: $E(x,y), E(x,z) \leftarrow A(x), P(y), A(z), M(z,x), P(u), E(z,u)$
C_{14}: $M(x,y) \leftarrow C(x), B(y)$
C_{15}: $M(x,y) \leftarrow S(x), B(y)$
C_{16}: $M(x,y) \leftarrow B(x), F(y)$
C_{17}: $M(x,y) \leftarrow F(x), W(y)$
C_{18}: $\leftarrow W(x), F(y), E(x,y)$
C_{19}: $\leftarrow W(x), G(y), E(x,y)$
C_{20}: $E(x,y) \leftarrow B(x), C(y)$
C_{21}: $\leftarrow B(x), S(y), E(x,y)$
C_{22}: $P(f_1(x)) \leftarrow C(x)$
C_{23}: $E(x, f_1(x)) \leftarrow C(x)$
C_{24}: $P(f_2(x)) \leftarrow S(x)$
C_{25}: $E(x, f_2(x)) \leftarrow S(x)$
C_{26}: $\leftarrow A(x), A(y), G(z), E(y,z), E(x,y)$

Figure 4: Clausal form.

5.4 Solving the Puzzle as a MI Problem

This solution is explained in our theory as follows: The Steamroller puzzle is first formalized as a proof problem $\langle F, F_{23} \rangle$, which asks whether $F \models F_{23}$ or not. This is reformalized as a MI problem $\langle F \wedge \neg F_{23}, \varphi \rangle$, where φ is a mapping from $pow(\mathcal{G})$ to $\{yes, no\}$ such that for any $G \subseteq \mathcal{G}$, $\varphi(G) =$ "yes" if $G = \mathcal{G}$, otherwise $\varphi(G) =$ "no". Since $Models(F \wedge \neg F_{23}) = Models(Cs)$, the MI problem $\langle F \wedge \neg F_{23}, \varphi \rangle$ is transformed equivalently to a MI problem $\langle Cs, \varphi \rangle$. Many ET rules are used for solving this MI problem.

6 COMPUTATION WITH PRIORITIZED ET RULES

6.1 Finite and Infinite Computation with Prioritized ET Rules

Let R_p be a sequence of ET rules such that $R_p = [r_1, r_2, \ldots, r_m]$. Let S be a state. Since the rule order in $[r_1, r_2, \ldots, r_m]$ is used for specifying the priority of rule application to a state S, R_p is regarded as a set of

prioritized ET rules.

R_p is applicable to S with r_j if

1. none of $r_1, r_2, \ldots, r_{j-1}$ is applicable to S, and
2. r_j is applicable to S.

Note that j is determined uniquely by S and R_p. If R_p is applicable to S with r_j, the result of the application of R_p to S, denoted by $R_p(S)$, is $r_j(S)$.

Given a state S_0, R_p determines a finite or infinite sequence of states as follows:

1. R_p determines a finite sequence $[S_0, S_1, \ldots, S_n]$ if
   (a) R_p is applicable to S_i for each $i \in \{0, 1, \ldots, n-1\}$,
   (b) $S_{i+1} = R_p(S_i)$ for each $i \in \{0, 1, \ldots, n-1\}$, and
   (c) R_p is not applicable to S_n.
2. R_p determines an infinite sequence $[S_0, S_1, \ldots]$ if
   (a) R_p is applicable to S_i for each $i \in \{0, 1, \ldots\}$, and
   (b) $S_{i+1} = R_p(S_i)$ for each $i \in \{0, 1, \ldots\}$.

Let $R_p = [r_1, r_2, \ldots, r_m]$ and let A be an answer mapping. Assume that for any $i \in \{1, 2, \ldots, m\}$ and any state S, if $S \in dom(A)$, then r_i is not applicable to S. Then, if R_p produces an infinite computation $[S_0, S_1, \ldots]$, none of $S_0, S_1, \ldots$ is in $dom(A)$. We have the following three cases:

1. If R_p determines a finite sequence $[S_0, S_1, \ldots, S_n]$, $S_n \in dom(A)$, and $S_n = \langle Cs_n, \varphi_n \rangle$, then the computed answer is $A(Cs_n, \varphi_n)$.
2. If R_p determines a finite sequence $[S_0, S_1, \ldots, S_n]$ and $S_n \notin dom(A)$, then no answer is obtained.
3. If R_p determines an infinite sequence $[S_0, S_1, \ldots]$, then no answer is obtained.

6.2 Correctness of Computation with Prioritized ET Rules

Given a set Cs of extended clauses and an extraction mapping φ, the MI problem $\langle Cs, \varphi \rangle$ can be solved as follows:

1. Let A be an answer mapping.
2. Prepare a sequence R_p of ET rules on STATE.
3. Take S_0 such that $S_0 = \langle Cs, \varphi \rangle$ to start computation from S_0.
4. Assume that R_p determines a finite sequence $[S_0, S_1, \ldots, S_n]$ and $S_n = \langle Cs_n, \varphi_n \rangle$.
5. If the computation reaches the domain of A, i.e., $\langle Cs_n, \varphi_n \rangle \in dom(A)$, then compute the answer by using the answer mapping A, i.e., output $A(Cs_n, \varphi_n)$.

The selection of R_p is arbitrary and there are many possible computation paths depending on the selection of ET rules in R_p and the order of them in R_p. Every output computed by using any arbitrary computation path is correct.

Theorem 2. *When an ET sequence starting from $S_0 = \langle Cs, \varphi \rangle$ reaches S_n in $dom(A)$, the above procedure gives the correct answer to $\langle Cs, \varphi \rangle$.*

Proof: Since $[S_0, S_1, \ldots, S_n]$ is an ET sequence, $ans_{\mathrm{MI}}(Cs, \varphi) = ans_{\mathrm{MI}}(Cs_n, \varphi_n)$. Since A is an answer mapping, $ans_{\mathrm{MI}}(Cs_n, \varphi_n) = A(Cs_n, \varphi_n)$. Hence $ans_{\mathrm{MI}}(Cs, \varphi) = A(Cs_n, \varphi_n)$. □

7 COMPUTATION CONTROL WITH PRIORITIZED ET RULES

7.1 Resource Minimization by Prioritized ET Rules

In practical problem solving, resource for computation is limited. Typically, we minimize execution time to reach a conclusion under space constraints. For each state $S = \langle Cs, \varphi \rangle$,

- the number of clauses in Cs should not exceed a certain limit, and
- the number of all atoms in Cs should also not exceed a certain limit.

Assuming that smaller amount of space consumption tends to decrease total execution time, we will design a set of ET rules and their priority.

7.2 Restricted Resolution and Restricted Factoring

Resolution adds a resolvent of two clauses and always increases the number of clauses by one. Let (res i) be defined as an ET rule for making a resolution step with a resolvent containing not more than i atoms. Since a smaller number atoms are better for saving space and for finding simpler clauses, the priority order of (res 0), (res 1), (res 2), ... is given by:

$$(\text{res } 0) > (\text{res } 1) > (\text{res } 2) > \cdots$$

Factoring also adds a clause; it always increases the number of clauses by one. Let (fac i) be defined as a factoring rule with a new clause containing not more than i atoms. Again, for finding simpler clauses and saving space, the priority order of (fac 1), (fac 2), (fac 3), ... is specified by:

$$(\text{fac } 1) > (\text{fac } 2) > (\text{fac } 3) > \cdots$$

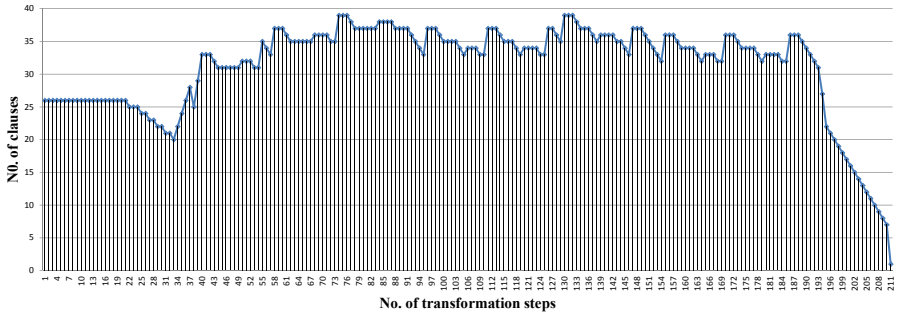

Figure 5: Changes of the number of clauses.

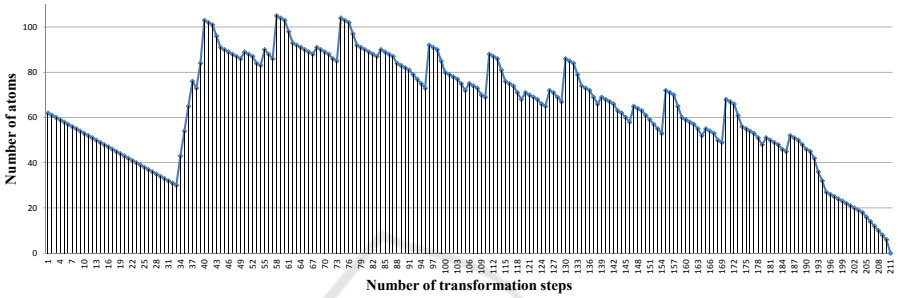

Figure 6: Changes of the number of atoms.

7.3 Restricted Unfolding

Unfolding decreases the number of clauses by one when the number of resolvents is zero, does not change the number of clauses when it produces only one resolvent, and increases the number of clauses otherwise.

Unfolding with not more than i resolvents is a transformation rule that satisfies the following conditions:

1. Letting D be a set of definite clauses used for unfolding, this rule is applicable to a body atom b in Cs with respect to Cs and D when

$$|\{C \mid (C \in D)\ \&\ (b \text{ and } head(C) \text{ is unifiable})\}| \leq i.$$

2. The result of the transformation is the same as that of usual unfolding.

Let (udi i) be defined as a transformation rule such that

- definite-clause removal (see Section 8.2) is applied if it is applicable,
- otherwise unfolding with not more than i resolvents is applied if it is applicable.

The priority order of (udi 1),(udi 2),(udi 3),... is given by:

$$\text{(udi 1)} > \text{(udi 2)} > \text{(udi 3)} > \cdots$$

7.4 A Solution to the Steamroller Problem

Let (erase) be an ET rule for erasing independent satisfiable atoms (see Section 8.5), (subsumed) an ET rule for elimination of subsumed clauses (see Section 8.6), and (fwd) an ET rule for forwarding transformation. When we take the rule priority

$$\text{(udi 1)} > \text{(erase)} > \text{(subsumed)} > \text{(fwd)} > \text{(udi 2)} > \text{(udi 3)} > \text{(udi 5)},$$

the Steamroller puzzle is solved by 211 rule applications. Changes of the number of clauses and those of the number of atoms are shown by Fig. 5 and Fig. 6, respectively, where (udi 1) is applied 116 times, (erase) 15 times, (subsumed) 38 times, (fwd) 17 times, (udi 2) 8 times, (udi 3) 4 times, and (udi 5) 13 times.

7.5 Comparison

By deleting (udi 2) and (udi 3) from the above priority, we have

$$\text{(udi 1)} > \text{(erase)} > \text{(subsumed)} > \text{(fwd)} > \text{(udi 5)},$$

which gives only 90 steps to obtain the same solution. However, when we remove the rule (udi 1), i.e., when we take

$$\text{(erase)} > \text{(subsumed)} > \text{(fwd)} > \text{(udi 5)},$$

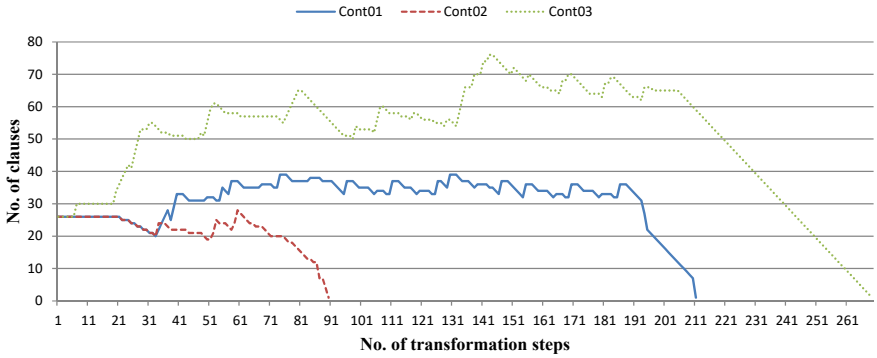

Figure 7: Computation control comparison.

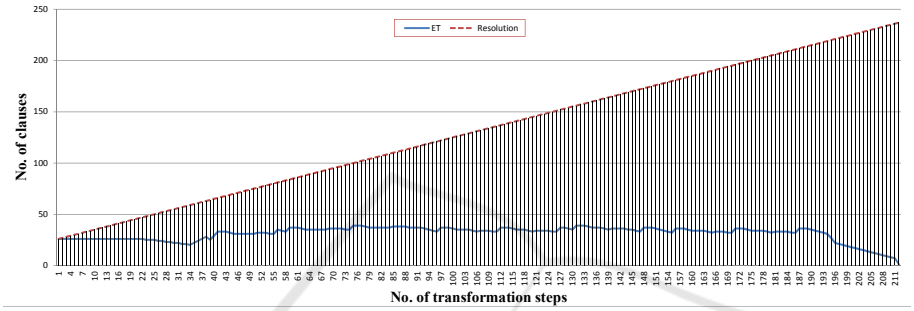

Figure 8: Comparing computation by ET with that by resolution and factoring.

we need 269 steps to reach the final singleton of the empty clause, showing that prioritized application of (udi 1) is important for efficient computation.

So far we have introduced three priority controls, which are referred to as "Cont01", "Cont02", and "Cont03". Changes of the number of clauses resulting from these priority controls when solving the Steamroller puzzle are shown in Fig 7.

Since resolution and factoring are ET rules, the conventional resolution proof method is covered by our framework. For example, we can take prioritized ET rules (res 99) $>$ (fac 99) to solve proof problems by the resolution and factoring ET rules.

Fig. 8 compares ET computation using the priority control "Cont01" with computation by resolution and factoring. Since each resolution step adds one resolvent of two clauses, each step increases the number of clauses by one. In the proof with resolution and factoring, computation goes on the upward straight line in Fig. 8, which may exceed the space limitation before the computation terminates.

7.6 Efficiency Improvement

Our method has a chance to improve efficiency compared to the conventional resolution-based methods. All strategies taken in the resolution-based methods can also be used in our theory. Adoption of new ET rules and adjustment of rule priority provide a powerful mechanism for computation control, which cannot be utilized in the conventional methods.

It is expected that the method of prioritized ET rules overcomes the limitation of the conventional methods, since we can plan to search goals with unlimited variety of ET rules. So far, this has been experimentally shown by several small proof problems and QA problems, such as the Agatha proof problem (with built-in atoms) and the Agatha QA problem (Akama and Nantajeewarawat, 2018b).

8 ET RULES BY EXAMPLES

In this section we explain the ET rules used in this paper by examples. Their strict mathematical definitions and correctness proofs can be found elsewhere (Akama and Nantajeewarawat, 2016b; Akama and Nantajeewarawat, 2018a).

8.1 Unfolding

Unfolding with respect to an atom b for extended clauses on $\mathrm{ECLS_F}$ is the same as unfolding for usual clauses except the possible avoidance of application such as existence of multi-head clauses containing b in their head parts.

For instance, suppose that Cs contains the four clauses:

C_1: $p_1 \leftarrow p_2, p_3$
C_2: $p_1 \leftarrow p_4$
C_3: $p_1, p_2 \leftarrow p_5$
C_4: $h_1, h_2 \leftarrow p_1, p_6$

Then C_4 cannot be unfolded at p_1 since p_1 belongs to the head part of the multi-head clause C_3.

8.2 Definite-Clause Removal

Useless definite clauses with respect to unfolding are removed by this rule. For instance, consider a MI problem $\langle Cs, \varphi \rangle$. Assume that φ does not depend on p_1 and Cs consists of the following clauses:

C_1: $p_1 \leftarrow p_2, p_3$
C_2: $p_1 \leftarrow p_4$
C_3: $p_8 \leftarrow p_2$
C_4: $h_1, h_2 \leftarrow p_6, p_8$

Then C_1 and C_2 can be removed from Cs since p_1 does not appear in the right-hand side of any other clause in Cs.

8.3 Resolution

Resolution for extended clauses on $\mathrm{ECLS_F}$ is the same as resolution for usual clauses except the possible existence of *func*-atoms. Only usual variables in *func*-atoms are changed by the most general unifier in use; function variables are not changed.

For instance, suppose that Cs contains the two clauses:

C_1: $p(x) \leftarrow q(x), r(x,4), s(x), \mathit{func}(h,x)$
C_2: $r(1,y), t(y,z) \leftarrow u(y), v(z)$

By applying the resolution rule to C_1 and C_2, a new clause

C_3: $p(1), t(4,z) \leftarrow q(1), s(1), u(4), v(z), \mathit{func}(h,1)$

is added to Cs as the resolvent.

8.4 Factoring

Two atoms in the same side in a clause are unified to give a new clause. For instance, suppose that Cs contains the clause:

C_1: $p(x) \leftarrow q(x), r(x,4), r(3,y), \mathit{func}(h,y)$

Then a new clause

C_2: $p(3) \leftarrow q(3), r(3,4), \mathit{func}(h,4)$

is added to Cs. Suppose that Cs contains the clause:

C_3: $p(x), r(x,4), r(3,y) \leftarrow \mathit{func}(h,y)$

A new clause

C_4: $p(3), r(3,4) \leftarrow \mathit{func}(h,4)$

is added.

8.5 Erasing Independent Satisfiable Atoms

Let C be a clause and B a set of atoms. Let $C \ominus B$ be defined as the clause obtained from C by removing all atoms in B from its right-hand side. That is, $C \ominus B$ is defined by $\mathit{lhs}(C \ominus B) = \mathit{lhs}(C)$ and $\mathit{rhs}(C \ominus B) = \mathit{rhs}(C) - B$. This rule, referred to as (erase) in Section 7, changes C into $C \ominus B$ if (i) B and $(\mathit{lhs}(C) \cup \mathit{rhs}(C)) - B$ have no common variable and (ii) B can be instantiated to be true under the condition of $Cs - \{C\}$.

For instance, suppose that Cs contains the two clauses:

C_1: $p(f(2,6)) \leftarrow$
C_2: $r(y) \leftarrow p(f(x,6)), q(y)$

Then $p(f(x,6))$ can be removed from C_2.

8.6 Elimination of Subsumed Clauses

This rule, referred to as (subsumed) in Section 7, removes a clause C from a clause set Cs if C is subsumed by some clause in Cs. For instance, suppose that Cs contains the two clauses:

C_1: $h_1, h_2 \leftarrow b_1, b_2$
C_2: $h_1 \leftarrow b_2$

Then C_1 can be removed from Cs.

9 CONCLUSIONS

Model-intersection (MI) problems constitute one of the largest classes of logical problems. Proof problems and QA problems on first-order logic can be solved by transforming them into MI-problems on extended clauses and by searching paths to target sets of extended clauses. We take extended clauses in $\mathrm{ECLS_F}$, which overcomes the serious limitation of expressive power of conventional clauses, where existential quantification is never represented.

An ET rule is a partial mapping on the powerset of $\mathrm{ECLS_F}$ that preserves the answer to a given MI problem. Most ET rules transform a clause set Cs preserving $\mathit{Models}(Cs)$ and/or $\bigcap \mathit{Models}(Cs)$. The possibility of using ET rules unlimitedly has fundamentally changed the concept of computation. The conventional concept of computation in logical problem solving is based on procedural reading of logical formulas, while computation in our theory is successive application of an unlimited number of ET rules, which are not logical formulas. Correctness of computation by ET rules has been strictly guaranteed.

On the basis of such fundamental changes of the computation framework, a new concept of computation control has been introduced in this paper. Application priority among ET rules works as computation control. Priority ordering in a set of ET rules limits computation variety. Appropriate selection of priority ordering is useful for finding efficient computation paths.[1] Many conventional methods for logical computation use restricted sets of ET rules and restricted control within the ET rules, and can be regarded as special forms of our method. It is expected that prioritized ET rules will produce more efficient solutions for logical problem solving.

ACKNOWLEDGEMENTS

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

REFERENCES

Akama, K. and Nantajeewarawat, E. (2011). 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. (2016a). Model-Intersection Problems with Existentially Quantified Function Variables: Formalization and a Solution Schema. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016), Volume 2: KEOD*, pages 52–63, Porto, Portugal.

Akama, K. and Nantajeewarawat, E. (2016b). Unfolding Existentially Quantified Sets of Extended Clauses. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016), Volume 2: KEOD*, pages 96–103, Porto, Portugal.

Akama, K. and Nantajeewarawat, E. (2018a). Computation Control by Prioritized ET Rules. Technical report, Information Initiative Center, Hokkaido University.

Akama, K. and Nantajeewarawat, E. (2018b). Solving Query-Answering Problems with Constraints for Function Variables. In *Proceedings of the 10th Asian Conference on Intelligent Information and Database Systems*, LNAI 10751, pages 36–47, Dong Hoi City, Vietnam.

Manthey, R. and Bry, F. (1988). SATCHMO: A Theorem Prover Implemented in Prolog. In *Proceedings of the 9th International Conference on Automated Deduction*, LNCS 310, pages 415–434, Argonne, IL.

Pelletier, F. J. (1986). Seventy-Five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2(2):191–216.

Stickel, M. (1986). Schubert's Steamroller Problem: Formulations and Solution. *Journal of Automated Reasoning*, 2(2):89–104.

Walther, C. (1985). A Mechanical Solution of Schubert's Steamroller by Many-Sorted Resolution. *Artificial Intelligence*, 26(2):217–224.

Wang, T.-C. and Bledsoe, W. W. (1987). Hierarchical Deduction. *Journal of Automated Deduction*, 3(1):35–77.

[1] The current version of our MI solver receives a problem description and control priority as input and produces an ET sequence and the final answer (when the computation reaches it), where finding good computation control is essential for the success of solving problems in practical time.