Unfolding Existentially Quantified Sets of Extended Clauses

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: Unfolding, Extended Clause, Function Variable, Model-intersection Problem, Equivalent Transformation.

Abstract: Conventional theories cannot solve many logical problems due to the limitations of the underlying clause space. In conventional clauses, all variables are universally quantified and no existential quantification is allowed. 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, have been introduced, resulting in a new space of extended clauses, called $\mathrm{ECLS_F}$. This new space is necessary to overcome the limitations of the conventional clause space. To solve problems on $\mathrm{ECLS_F}$, many equivalent transformation rules are used. We formally defined unfolding transformation on $\mathrm{ECLS_F}$, which is applicable not only to definite clauses but also to multi-head clauses. The proposed unfolding transformation preserves the answers to model-intersection problems and is useful for solving many logical problems such as proof problems and query-answering problems on first-order logic with built-in constraint atoms.

1 INTRODUCTION

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 $\mathrm{ECLS_F}$ space, in which a clause may contain three kinds of atoms: user-defined atoms, built-in constraint atoms, and *func*-atoms. Variables of a new type, called *function variables*, appear in the first argument positions of *func*-atoms, and they are existentially quantified at the top level of a clause set under consideration.

A *model-intersection problem* (*MI problem*) on $\mathrm{ECLS_F}$ is a pair $\langle Cs, \varphi\rangle$, where Cs is a set of extended clauses in $\mathrm{ECLS_F}$ and φ is a mapping, called an *exit mapping*, used for constructing the output answer from the intersection of all models of Cs. More formally, the answer to a MI problem $\langle Cs, \varphi\rangle$ is $\varphi(\bigcap Models(Cs))$, where $Models(Cs)$ is the set of all models of Cs and $\bigcap Models(Cs)$ is the intersection of all such models.

Note that we can take the intersection of all elements of $Models(Cs)$ since each interpretation (hence each model) is, in our semantics, a set of ground user-defined atoms, which is similar to a Herbrand interpretation (Chang and Lee, 1973; Fitting, 1996). The logical structure theory (Akama and Nantajeewarawat, 2006; Akama and Nantajeewarawat, 2011a) has already shown the generality and usefulness of this semantics.

MI problems on $\mathrm{ECLS_F}$ constitute a very large class of logical problems, which is of great importance. Let $\mathrm{FOL_c}$ denote the set of all first-order formulas with built-in constraint atoms. As depicted by Fig. 1, all proof problems and all query-answering (QA) problems on $\mathrm{FOL_c}$ are mapped, preserving their answers, into MI problems on $\mathrm{ECLS_F}$ (Akama and Nantajeewarawat, 2015). By solving MI problems on $\mathrm{ECLS_F}$, we can solve proof problems and QA problems on $\mathrm{FOL_c}$.

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 QA problem is an "all-answers finding" problem, i.e., finding all ground instances of a given query atom

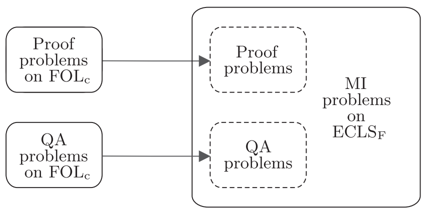

Figure 1: Embedding logical problems into MI problems.

Akama, K. and Nantajeewarawat, E.
Unfolding Existentially Quantified Sets of Extended Clauses.
DOI: 10.5220/0006051500960103
In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016) - Volume 2: KEOD*, pages 96-103
ISBN: 978-989-758-203-5

that are logical consequences of a given formula. The usual clause space taken by conventional logic programming is too small to consider all proof problems on $\mathrm{FOL_c}$ and all QA problems on $\mathrm{FOL_c}$. By contrast, the $\mathrm{ECLS_F}$ has enough knowledge representation power for dealing with all these problems. This is the fundamental reason why we should take the $\mathrm{ECLS_F}$ space in place of the usual clause space.

A general schema for solving MI problems on $\mathrm{ECLS_F}$ by equivalent transformation (ET) has been proposed (Akama and Nantajeewarawat, 2015), where problems are solved by repeated problem simplification using ET rules. The proposed solution schema for MI problems comprises the following steps: (i) formalize a given problem as a MI problem or map it into a MI problem, (ii) prepare ET rules, (iii) construct an ET sequence, and (iv) compute the answer.

This paper proposes unfolding transformation on the $\mathrm{ECLS_F}$ space, and proves its correctness. Unfolding transformation (also simply called unfolding) has been one of the most important equivalent transformation for definite clauses. In contrast to a definite clause, a clause in $\mathrm{ECLS_F}$ may contain more than one user-defined atom in its left-hand side and also function variables in its right-hand side. Unfolding in the $\mathrm{ECLS_F}$ space therefore requires a new definition and a new correctness proof.

The rest of the paper is organized as follows: Section 2 introduces the extended space with function variables and the semantics of extended clauses. Section 3 formalizes MI problems and provides a solution method for them based on equivalent transformation (ET). Section 4 defines occurrence relations and unfolding transformation. Section 5 shows a correctness theorem for unfolding. Section 6 provides conclusions. The proofs of all results presented in this paper can be found in (Akama and Nantajeewarawat, 2016).

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 AN EXTENDED CLAUSE SPACE

2.1 Built-in Atoms

Built-in atoms are essential for representation of knowledge using first-order formulas. For instance, the predicate *length* may be defined as follows:

$$\begin{aligned} length(X,Y)\ \text{ iff }\ &(not((X=[\,])\text{ or }(Y=0))\\ &\text{and }(not(length(X,Y1))\text{ or}\\ &\quad not(Y:=Y1+1)\text{ or}\\ &\quad length([A|X],Y))),\end{aligned}$$

where $(X = [\,])$, $(Y = 0)$, and $(Y := Y1+1)$ are built-in atoms. The meanings of built-in atoms are defined by specifying the set of all true ground atoms. For example:

- $(s = t)$ is true iff s and t are the same ground terms.
- $(s := t-1)$ is true iff s and t are numbers and s is equal to $t-1$.

This is different from the semantics of user-defined atoms. The truth or falsity of a ground user-defined atom is determined by an interpretation. A ground user-defined atom g is true with respect to an interpretation I iff g is an element of I.

A first-order formula may determine several models, and the truth or falsity of a ground user-defined atom depends on a model under consideration, i.e., a ground user-defined atom may be contained in one model but not in another model. The truth or falsity of a ground built-in atom is predetermined uniquely. The objective of representation by first-order formulas is to determine a set of models, where built-in atoms are useful and indispensable as shown in the *length* example above.

2.2 Incompleteness of the Usual Clause Space

Let CLS be the set of all clauses consisting only of user-defined atoms, and $\mathrm{CLS_c}$ 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 $\mathrm{FOL_c}$ the set of all first-order formulas consisting of user-defined atoms and built-in atoms.

It is well-known that there is a mapping SKO such that each first-order formula in FOL is transformed by SKO into a set of clauses in CLS preserving satisfiability. This enables resolution-based theorem proving, and motivates us to consider SKO and CLS as a foundation for logical problem solving.

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

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

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

These difficulties are overcome by meaning preserving Skolemization (MPS) and an extended clause space, called 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.
- All proof problems and all QA problems on FOL and those on FOL_c can be transformed into MI problems on ECLS_F.

2.3 Insufficiency of Conventional Logic Programming

Most of logic programming research uses subspaces of CLS_c, i.e., conventional logic programs are sets of normal clauses and provide no representation power of existential quantification. So they can never provide a general framework of solving logical problems.

Even if a logic programming language (e.g., Prolog) is Turing complete, it does not mean that everything can be done using such a language. 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 procedures, but to invent procedures for giving correct solutions to MI problems. Such invention is not an easy task, but once a procedure is invented, a simulation of it is rather an easy task. Turing completeness means not so large advantages; most practical programming languages are Turing complete.

2.4 User-defined Atoms, Constraint Atoms, and *func*-Atoms

We consider an extended formula space that contains three kinds of atoms, i.e., user-defined atoms, built-in constraint atoms, and *func*-atoms. A *user-defined atom* takes the form $p(t_1,\dots,t_n)$, where p is a user-defined predicate and the t_i are usual terms. A *built-in constraint atom*, also simply called a *constraint atom* or a *built-in atom*, takes the form $c(t_1,\dots,t_n)$, where c is a predefined constraint predicate and the t_i are usual terms. Let $\mathcal{A}_u$ be the set of all user-defined atoms, $\mathcal{G}_u$ the set of all ground user-defined atoms, $\mathcal{A}_c$ the set of all constraint atoms, and $\mathcal{G}_c$ the set of all ground constraint atoms.

A *func-atom* (Akama and Nantajeewarawat, 2011b) is an expression of the form $func(f,t_1,\dots,t_n,t_{n+1})$, 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.

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. Let *FVar* be the set of all function variables and *FCon* the set of all function constants. A substitution for function variables is a mapping from *FVar* to $\mathit{FVar}\cup\mathit{FCon}$. Each n-ary function constant is associated with a mapping from $\mathcal{G}_t^n$ to $\mathcal{G}_t$, where $\mathcal{G}_t$ denotes the set of all ground usual terms.

2.5 Extended Clauses

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

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

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

Let DCL denote the set of all extended definite clauses with no constraint atom in their left-hand sides. Given a definite clause $C\in\text{DCL}$, the user-defined atom in $lhs(C)$ is called the *head* of C, denoted by $head(C)$, and the set $rhs(C)$ is called the *body* of C, denoted by $body(C)$. Given $D\subseteq\text{DCL}$, let $head(D)=\{head(C)\mid C\in D\}$.

2.6 An Extended Clause Space

The set of all extended clauses is denoted by ECLS_F. The *extended clause space* in this paper is the powerset of 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.

2.7 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 for function variables $\sigma \in \mathit{Map}(\mathit{FVar}, \mathit{FVar} \cup \mathit{FCon})$, 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., $\mathit{Models}(Cs)$ is the set of all models of Cs for any $Cs \subseteq \mathrm{ECLS_F}$.

Note that the standard semantics is taken in this paper, i.e., all models of a formula are considered instead of specific ones, such as those considered in the minimal model semantics (Clark, 1978; Lloyd, 1987) (i.e., the semantics underlying definite logic programming) and those considered in the stable model semantics (Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1991) (i.e., the semantics underlying answer set programming).

3 SOLVING MI PROBLEMS BY EQUIVALENT TRANSFORMATION (ET)

3.1 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 $\mathit{pow}(\mathcal{G}_{\mathrm{u}})$ to some set W. The mapping φ is called an *exit mapping*. The answer to this problem, denoted by $\mathit{ans}_{\mathrm{MI}}(Cs, \varphi)$, is defined by

$$\mathit{ans}_{\mathrm{MI}}(Cs, \varphi) \;=\; \varphi(\bigcap \mathit{Models}(Cs)),$$

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

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

$\mathit{pat}(oe) \leftarrow$
$\mathit{prob}(io), \mathit{pat}(po) \leftarrow$
$\mathit{prob}(io) \leftarrow \mathit{pat}(po)$
$\mathit{prob}(oe) \leftarrow \mathit{pat}(po)$

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

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

Moreover, for any $M \subseteq \mathcal{G}_{\mathrm{u}}$, M is a model of Cs iff there exists $M_0 \subseteq \mathcal{G}_{\mathrm{u}}$ such that

1. $M = M_0 \cup M_1$ or $M = M_0 \cup M_2$, and
2. $\mathit{pat}(po) \notin M_0$.

So $\bigcap \mathit{Models}(Cs) = \{\mathit{prob}(io), \mathit{pat}(oe)\}$. Therefore $\mathit{ans}_{\mathrm{MI}}(Cs, \varphi) = \{io\}$. □

3.2 Target Mappings

Given a MI problem $\langle Cs, \varphi \rangle$, since $\mathit{ans}_{\mathrm{MI}}(Cs, \varphi) = \varphi(\bigcap \mathit{Models}(Cs))$, the answer to this MI problem is determined uniquely by $\mathit{Models}(Cs)$ and φ. As a result, we can equivalently consider a new MI problem with the same answer by switching from Cs to another clause set Cs' if $\mathit{Models}(Cs) = \mathit{Models}(Cs')$, i.e., MI problems can be transformed into simpler forms by equivalent transformation (ET) preserving the mapping *Models*.

In order to use more partial mappings for simplification of MI problems, we extend our consideration from the specific mapping *Models* to a class of partial mappings, called GSETMAP, defined below.

Definition 1. GSETMAP is the set of all partial mappings from $pow(\mathrm{ECLS_F})$ to $pow(pow(\mathcal{G}_u))$. □

As defined in Section 2.7, $Models(Cs)$ is the set of all models of Cs for any $Cs \subseteq \mathrm{ECLS_F}$. Since a model is a subset of $\mathcal{G}_u$, *Models* is regarded as a total mapping from $pow(\mathrm{ECLS_F})$ to $pow(pow(\mathcal{G}_u))$. Since a total mapping is also a partial mapping, the mapping *Models* is a partial mapping from $pow(\mathrm{ECLS_F})$ to $pow(pow(\mathcal{G}_u))$, i.e., it is an element of GSETMAP.

A partial mapping M in GSETMAP is of particular interest if $\bigcap M(Cs) = \bigcap Models(Cs)$ for any $Cs \in dom(M)$. Such a partial mapping is called a *target mapping*.

Definition 2. A partial mapping $M \in$ GSETMAP is a *target mapping* iff for any $Cs \in dom(M)$, $\bigcap M(Cs) = \bigcap Models(Cs)$. □

It is obvious that:

Theorem 1. *The mapping Models is a target mapping.* □

The next theorem provides a sufficient condition for a mapping in GSETMAP to be a target mapping.

Theorem 2. *Let $M \in$ GSETMAP. M is a target mapping if the following conditions are satisfied:*

1. *$M(Cs) \subseteq Models(Cs)$ for any $Cs \in dom(M)$.*
2. *For any $Cs \in dom(M)$ and any $m \in Models(Cs)$, there exists $m' \in M(Cs)$ such that $m' \subseteq m$.* □

3.3 Answer Mappings

A set of problems that can be solved at low cost is useful to provide a desirable final destination for ET computation. It can also be specified as a partial mapping that is preserved by ET. Such a specification is useful to invent and to justify a new ET rule. This motivates the concept of answer mapping, which is formalized below.

Definition 3. Let W be a set. A partial mapping A from

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

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

If M is a target mapping, then M can be used for constructing answer mappings.

Theorem 3. *Let M be a target mapping. Suppose that A is a partial mapping such that*

- *$dom(M) = \{x \mid \langle x, y\rangle \in dom(A)\}$, and*
- *for any $\langle Cs, \varphi\rangle \in dom(A)$,*

$$A(Cs, \varphi) = \varphi(\bigcap M(Cs)).$$

Then A is an answer mapping. □

3.4 ET Steps and ET Rules

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

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

Definition 4. Let $\langle S, S'\rangle \in$ STATE $\times$ 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 5. 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:

Definition 6. 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. □

We also define ET rules on $pow(\mathrm{ECLS_F})$ as follows:

Definition 7. An *ET rule* r with respect to a target mapping M is a partial mapping from $pow(\mathrm{ECLS_F})$ to $pow(\mathrm{ECLS_F})$ such that for any $Cs \in dom(r)$, $M(Cs) = M(r(Cs))$. □

We can construct an ET rule on STATE from an ET rule with respect to a target mapping.

Theorem 4. *Assume that M is a target mapping and r is an ET rule with respect to M. Suppose that $\overline{r}$ is a partial mapping from STATE to STATE such that*

- *$dom(r) = \{x \mid \langle x, y\rangle \in dom(\overline{r})\}$, and*
- *$\overline{r}(S) = \langle r(Cs), \varphi\rangle$ if $S = \langle Cs, \varphi\rangle \in dom(\overline{r})$.*

Then $\overline{r}$ is an ET rule on STATE. □

3.5 A Correct Solution Method based on ET Rules

A MI problem $\langle Cs, \varphi\rangle$, where $Cs \subseteq \mathrm{ECLS_F}$ and φ is an exit mapping, can be solved as follows:

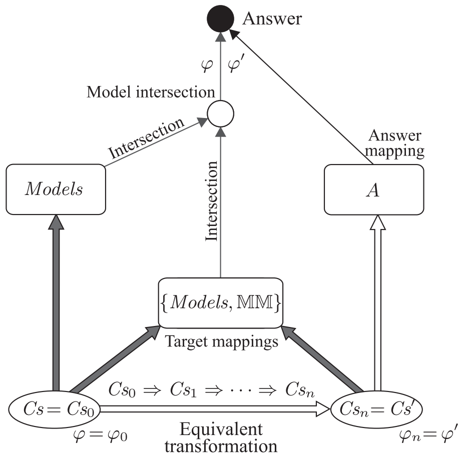

Figure 2: ET computation paths constructed by a combination of target mappings and answer mappings.

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 exit mapping φ, the answer to the MI problem $\langle Cs, \varphi \rangle$, i.e., $ans_{\mathrm{MI}}(Cs, \varphi) = \varphi(\bigcap Models(Cs))$, can be directly obtained by the computation shown in the leftmost path in Fig. 2. Instead of taking this computation path, the above solution takes a different one, i.e., the lowest path (from Cs to Cs') followed by the rightmost path (through the answer mapping A) in Fig. 2.

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 5. *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 $\langle Cs, \varphi \rangle$.* □

4 UNFOLDING ON ECLS$_{\mathrm{F}}$

4.1 Occurrence Relations

For definite-clause unfolding, a body atom in a target clause is specified for unification with each head atom in a set of definite clauses. An atom occurrence is usually used for such specification, which is generalized into an occurrence relation defined below.

Given $Cs \subseteq \mathrm{ECLS_F}$, a subset occ of $Cs \times \mathcal{A}_{\mathrm{u}}$ is said to be an *occurrence relation* on Cs iff for any $C \in Cs$, if $\langle C, b \rangle \in occ$, then $b \in rhs(C)$.

Assume that occ is a given occurrence relation on Cs. Let $dom(occ) = \{C \mid \langle C, b \rangle \in occ\}$ and $ran(occ) = \{b \mid \langle C, b \rangle \in occ\}$. Let $gran(occ)$ be defined as the set

$$\{b\theta \mid (\langle C, b \rangle \in occ)\ \&\ (\theta \text{ is a substitution for usual variables})\ \&\ (b\theta \text{ is ground})\}.$$

For any clause C, let $occ(C) = \{b \mid \langle C, b \rangle \in occ\}$.

Example 2. Assume that Cs consists of the following clauses:

C_1: $p_6, p_4 \leftarrow$
C_2: $p_5, p_4 \leftarrow$
C_3: $p_4, p_1 \leftarrow$
C_4: $\leftarrow p_1, p_2$
C_5: $p_3 \leftarrow p_6$
C_6: $\leftarrow p_4$
C_7: $p_2 \leftarrow p_5, p_3$

Let $occ = \{\langle C_4, p_2 \rangle\}$. Then occ is an occurrence relation on Cs, with $dom(occ) = \{C_4\}$. □

4.2 Unfolding Operation on ECLS$_{\mathrm{F}}$

An unfolding operation for a clause set Cs by using an arbitrary set D of definite clauses is defined below. For unfolding to preserve answers to MI problems, some additional conditions on Cs, D, and a specified occurrence relation are required. They will be given in Section 5 (Theorem 6).

Assume that

- $Cs \subseteq \mathrm{ECLS_F}$,
- D is a set of definite clauses in DCL, and
- occ is an occurrence relation on Cs.

By unfolding Cs using D at occ, Cs is transformed into $\mathrm{UNF}(Cs, D, occ)$, which is defined by

$$\mathrm{UNF}(Cs, D, occ) = (Cs - dom(occ)) \cup Reso(dom(occ), D, occ),$$

where $Reso(dom(occ), D, occ)$ is the set

$$\bigcup\{resolvent(C, C', b) \mid (C \in dom(occ)) \ \&\ (C' \in D) \ \&\ (b \in occ(C))\},$$

and for any $C \in dom(occ)$, any $C' \in D$, and any $b \in occ(C)$, $resolvent(C, C', b)$ is defined as follows, assuming that ρ is a renaming substitution for usual variables such that C and $C'\rho$ have no usual variable in common:

1. If b and $head(C'\rho)$ are not unifiable, then

$$resolvent(C, C', b) = \emptyset.$$

2. If they are unifiable, then

$$resolvent(C, C', b) = \{C''\},$$

where C'' is the clause obtained from C and $C'\rho$ as follows, assuming that θ is the most general unifier of b and $head(C'\rho)$:

(a) $lhs(C'') = lhs(C\theta)$

(b) $rhs(C'') = (rhs(C\theta) - \{b\theta\}) \cup body(C'\rho\theta)$

5 CORRECTNESS THEOREM

5.1 Correctness of Unfolding

We provide in Theorem 6 a sufficient condition for unfolding to preserve the answer to a MI problem. Given $Cs \subseteq \mathrm{ECLS_F}$, let $gleft(Cs)$ denote the set of all ground instances of user-defined atoms in the left-hand sides of extended clauses in Cs.

Theorem 6. *Assume that:*

1. $Cs \subseteq ECLS_{\mathrm{F}}$.
2. $D \subseteq Cs \cap \mathrm{DCL}$ *such that*

$$gleft(D) \cap gleft(Cs - D) = \emptyset.$$

3. *occ is an occurrence relation on Cs such that $dom(occ) \subseteq Cs - D$.*
4. $gran(occ) \cap gleft(Cs - D) = \emptyset$.
5. *φ is an exit mapping.*

Then $ans_{\mathrm{MI}}(Cs, \varphi) = ans_{\mathrm{MI}}(\mathrm{UNF}(Cs, D, occ), \varphi)$. □

Given a set Cs of extended clauses, one way to apply unfolding is as follows:

1. Select a clause C in Cs.
2. Select an atom b in the right-hand side of C.
3. Assuming that p is the predicate of the selected atom b, determine the set D consisting of all clauses in Cs that contain p-atoms in their left-hand sides.
4. If $C \notin D$ and D consists only of definite clauses, then unfold Cs with respect to b using D into Cs', i.e., make $Cs' = \mathrm{UNF}(Cs, D, occ)$, where $occ = \{\langle C, b\rangle\}$.

According to Theorem 6, this unfolding transformation is equivalent transformation.

Example 3. Consider the clause set Cs and the clauses C_1–C_7 in Example 2. Unfolding can be applied successively to this clause set as follows:

- By unfolding Cs with respect to p_2 in C_4 using $D = \{C_7\}$, we obtain $Cs_1 = (Cs - \{C_4\}) \cup \{C_4'\}$, where $C_4' = (\leftarrow p_1, p_5, p_3)$.
- By unfolding Cs_1 with respect to p_3 in C_4' using $D' = \{C_5\}$, we obtain $Cs_2 = (Cs_1 - \{C_4'\}) \cup \{C_4''\}$, where $C_4'' = (\leftarrow p_1, p_5, p_6)$.
- By unfolding Cs_2 with respect to p_3 in C_7 using $D' = \{C_5\}$, we obtain $Cs_3 = (Cs_2 - \{C_7\}) \cup \{C_7'\}$, where $C_7' = (p_2 \leftarrow p_5, p_6)$.

The resulting set Cs_3 contains the following clauses:

C_1: $p_6, p_4 \leftarrow$
C_2: $p_5, p_4 \leftarrow$
C_3: $p_4, p_1 \leftarrow$
C_4'': $\leftarrow p_1, p_5, p_6$
C_5: $p_3 \leftarrow p_6$
C_6: $\leftarrow p_4$
C_7': $p_2 \leftarrow p_5, p_6$

No further application of unfolding is possible to the clause set Cs_3. □

5.2 Target Mapping $\mathbb{MM}$

The answer preservation of a given MI problem by unfolding comes from the preservation of a target mapping, called $\mathbb{MM}$, which is given as follows: Given a set Cs of extended clauses, $\mathbb{MM}(Cs)$ is the set of all the least models of $D(\sigma, sel, Cs)$ such that σ is a possible function-variable instantiation and sel is a possible head-atom selection function, where $D(\sigma, sel, Cs)$ is the set of all ground definite clauses obtained by

1. applying the function-variable instantiation σ to clauses in Cs,
2. instantiating the resulting clauses by using all possible usual-variable instantiations,
3. simplification of the instantiated clauses, and
4. applying the head-atom selection function sel to the resulting simplified clauses.

The precise definition of $\mathbb{MM}$ can be found in (Akama and Nantajeewarawat, 2016).

To illustrate, suppose that *Cs* consists of the following three clauses:

$taxcut(x) \leftarrow hc(x,y), hc(x,z), (y \neq z)$
$hc(Peter, Paul) \leftarrow$
$hc(Peter, x) \leftarrow func(f, x)$

Then $\mathbb{MM}(Cs)$ is the union of

$$\{\{hc(Peter, Paul), hc(Peter, t), taxcut(Peter)\} \mid (t \text{ is a ground term}) \ \& \ (t \neq Paul)\}$$

and $\{\{hc(Peter, Paul)\}\}$.

6 CONCLUSIONS

The usual clause space has been extensively employed to compute the answers to proof problems and QA problems on first-order logic. However, it has not been successfully used for larger classes of proof problems and QA problems. A fundamental reason is the incompleteness of its representation power of existential quantification.

Considering the representation power of built-in constraint atoms and existential quantification, we take the $ECLS_F$ space. The $ECLS_F$ space is sufficient for representing all proof problems on FOL_c and all QA problems on FOL_c. MI problems on FOL_c constitute a large class of logical problems that can integrate all proof problems on FOL_c and all QA problems on FOL_c.

Equivalent transformation is a general principle for solving MI problems on $ECLS_F$, where many equivalent transformation rules (ET rules) are used. Many solution algorithms and procedures will be developed by inventing new ET rules. In the usual space, unfolding has been one of the most important and most often used ET rules. It is natural to try to extend unfolding rules used in the definite-clause space into unfolding on the $ECLS_F$ space.

The basic differences between the two spaces are as follows: A clause in the $ECLS_F$ space may contain (i) more than one atom in its left-hand side and (ii) function variables in its right-hand side. We proposed an unfolding operation that can be applied in the $ECLS_F$ space, which avoids the influence of non-definite clauses in a given clause set *Cs*. A set *D* of definite clauses in *Cs* is selected and used for unfolding at specified target atoms. The predicates appearing in the heads of definite clauses in the selected set *D* are required not to appear in the left-hand sides of clauses outside *D*.

In this paper, we also have reported a correctness theorem for unfolding transformation on the $ECLS_F$ space. The proof is given in (Akama and Nantajeewarawat, 2016) and is based on preservation of the target mapping $\mathbb{MM}$. The preservation of $\mathbb{MM}$ implies, with an unchanged exit mapping, the preservation of the answer to a given MI problem.

ACKNOWLEDGEMENTS

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

REFERENCES

Akama, K. and Nantajeewarawat, E. (2006). Logical Structures on Specialization Systems: Formalization and Satisfiability-Preserving Transformation. In *Proceedings of the 7th International Conference on Intelligent Technologies*, pages 100–109, Taipei, Taiwan.

Akama, K. and Nantajeewarawat, E. (2011a). Construction of Logical Structures on Specialization Systems. In *Proceedings of the 2011 World Congress on Information and Communication Technologies*, WICT 2011, pages 1030–1035, Mumbai, India.

Akama, K. and Nantajeewarawat, E. (2011b). 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. (2015). A General Schema for Solving Model-Intersection Problems on a Specialization System by Equivalent Transformation. In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015), Volume 2: KEOD*, pages 38–49, Lisbon, Portugal.

Akama, K. and Nantajeewarawat, E. (2016). Unfolding Existentially Quantified Sets of Extended Clauses. Technical report, Information Initiative Center, Hokkaido University.

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

Clark, K. L. (1978). Negation as Failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition.

Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–386.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer-Verlag, second, extended edition.