International Journal of Innovative Computing, Information and Control
Volume **19**, Number **5**, October **2023**

ICIC International ©2023 ISSN 1349-4198
pp. **1531–1542**

UNFOLD/UNBLOCK COMPUTATION CONTROL WITH SIDE-CHANGE TRANSFORMATION

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

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

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

Received July 2022; revised December 2022

Abstract. *Many logical problems, such as proof problems and query-answering problems, can be mapped into model-intersection (MI) problems, which constitute one of the largest and most fundamentally important classes of logical problems. To solve MI problems, many equivalent transformation rules have been employed. In this paper, we introduce unblocking transformation, and propose unfold/unblock computation control, i.e., when neither unfolding nor definite-clause removal is applicable, an attempt is made to transform a given problem using unblocking transformation so as to derive an equivalent problem to which unfolding is applicable. While a resolution-based proof method increases problem size monotonically no matter what control is taken, repeated reduction of problem size can be achieved by using the unfold/unblock control.*
Keywords: Model-intersection problem, Equivalent transformation, Rewriting rule, ET rule, Rule generation, Correctness

1. Introduction.

1.1. Limitations of resolution-based theories on first-order formulas.

The conventional logical computation theory has both representational limitations and computational limitations. Proof problems and query-answering (QA) problems are two most important and largest classes of logical problems, and many solutions to their subclasses have been intensively studied [1-15]. Proof problems on first-order formulas historically constitute the most important problem class in logical problem solving. The conventional resolution-based theory, however, failed to establish a correct proof method for full first-order formulas [16]. A QA problem on first-order formulas is an "all-answers finding" problem to satisfy a given logical consequence relation. The conventional resolution-based QA theory on definite clauses was constructed as an extension of the resolution-based proof theory [1]. However, it failed to establish a correct QA solution method for full first-order formulas [17].

The conventional concept of computation in logical problem solving is based on procedural reading of logical formulas. Only one inference rule (i.e., the resolution rule) is used in the conventional computation by SLD resolution, and control means to select an occurrence of an atom in a clause at each step of computation. Simple resolution-based

DOI: 10.24507/ijicic.19.05.1531

computation without the possibility of using many equivalent transformation (ET) rules is the major reason for the computational limitations of conventional logical reasoning and answer-finding.

1.2. **ET framework and resource minimization control.** To break the representational and computational limitations of first-order formulas, LPSF (Logical Problem Solving Framework) was invented [18]. LPSF is an axiomatic logic and can be applied to conventional logics. LPSF can also generate new logics.

LPSF solves model-intersection (MI) problems with various ET rules. MI problems constitute one of the largest and most fundamentally important classes of logical problems, including proof problems and QA problems. Computation in the ET framework is successive application of an unlimited number of ET rules. Computation by ET rules truly extends solvability of logical problems.

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. Resource minimization control is possible by assigning a higher priority to an ET rule whose application yields a smaller number of clauses. Resource minimization control is essential for solving logical problems efficiently.

1.3. **Partial-applicability of unfolding and unfold/unblock control.** One promising strategy for efficient computation is to repeatedly apply unfolding, by which a decrease of problem size can often be expected. However, unfolding may be inapplicable to an atom in the right-hand side of a clause in a problem description P when there exist multi-head clauses in P. Computation may often reach a state in which unfolding is blocked, i.e., unfolding is not applicable to the state. To get out of such an inapplicable state, we try to transform a given problem into an equivalent problem to which unfolding is further applicable. This is called unfold/unblock control.

Side-change transformation, which moves atoms from one side of a clause to the other side, is introduced as unblocking transformation in this paper. It does not change problem size, which is evaluated by the number of clauses and the number of atoms in the set of clauses representing a problem. However, side-change transformation contributes to efficient computation since it often enables further application of unfolding and repeated unfolding often leads to efficient simplification of problems.

The resolution rule is applicable to any clause in the existence of multi-head clauses. The importance of unfold/unblock control has never been recognized in the resolution-based research in logic programming. Computational limitations of the theory of logic programming, including i) incompleteness of SLD resolution for the full first-order formula space and ii) inefficient computation by inflexible control, arise from the single-rule approach using the resolution rule.

1.4. **Organization.** The rest of this paper is organized as follows. Section 2 defines the extended clause space and the semantics of a set of extended clauses. It also gives a formalization of a QA problem used as the running example in this paper. Section 3 explains inapplicability of unfolding in unfolding-based computation for solving MI problems. Section 4 introduces side-change transformation and explains its typical usage. It then proves the correctness of this transformation. Section 5 solves the running problem by unfolding-based computation with side-change transformation. Section 6 concludes the paper.

2. Formalization of a QA Problem.

We first introduce an extended clause space and give the formal definition of an MI problem on it. We then formalize a QA problem used for illustration in this paper.

2.1. **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 $p(t_1, \ldots, t_n)$, 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 $c(t_1, \ldots, t_n)$, where c is a predefined constraint predicate and the t_i are usual terms. Let $\mathcal{A}_{\mathrm{u}}$ denote 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.

A *func-atom* [19] 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. It is a *ground func*-atom if f is a function constant and the t_i are ground usual terms.

2.2. **Extended clauses and the extended clause space.** 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 $\{a_1, \ldots, a_m, b_1, \ldots, b_n\} \subseteq \mathcal{A}_{\mathrm{u}} \cup \mathcal{A}_{\mathrm{c}}$ 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 $lhs(C)$ and $rhs(C)$, respectively. When there is only one user-defined atom in $lhs(C)$, C is called an *extended definite clause*. When no confusion is caused, an extended clause is simply called a clause. 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 in *Cs* into function constants, clauses in the instantiated set are totally separated.

2.3. **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 $func(f, t_1, \ldots, t_n, t_{n+1})$ is true iff $f(t_1, \ldots, t_n) = t_{n+1}$.

Let $C = (a_1, \ldots, a_m \leftarrow b_1, \ldots, b_n, \mathbf{f}_1, \ldots, \mathbf{f}_p)$ be a ground clause, 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. C 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.

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.

For any $Cs \subseteq \mathrm{ECLS_F}$, let $Models(Cs)$ denote the set of all models of *Cs*.

2.4. **MI problems on the extended clause space.** A *model-intersection problem* (*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 as $\varphi(\bigcap Models(Cs))$, where $\bigcap Models(Cs)$ is the intersection of all models of Cs. Note that when $Models(Cs)$ is the empty set, $\bigcap Models(Cs) = \mathcal{G}_{\mathrm{u}}$.

The notion of an independent extraction mapping is next introduced; it is used for giving a sufficient condition for the correctness of side-change transformation. Given a set P of predicates, let $GAtoms(P)$ denote the set of all ground atoms in $\mathcal{G}_{\mathrm{u}}$ the predicates of which belong to P. An extraction mapping φ is said to be *independent* of a set P of predicates iff for any $G_1, G_2 \subseteq \mathcal{G}_{\mathrm{u}}$, if $(G_1 - G_2) \cup (G_2 - G_1) \subseteq GAtoms(P)$, then $\varphi(G_1) = \varphi(G_2)$.

2.5. **Problem description.** Consider the clause set Cs consisting of C_1-C_{25} in Figure 1, which provides the background knowledge on $\mathrm{ECLS_F}$ for the *mayDoThesis* problem (for short, the *mdt* problem), modified from the original problem given in [10]. All atoms appearing in Figure 1 are user-defined atoms. The clauses C_9-C_{11} together provide the conditions for a student to do his/her thesis with a professor, where for any student s and any professor p, $mdt(s, p)$ is intended to mean "s may do his/her thesis with p".

C_1: $FM(x) \leftarrow FP(x)$ $\quad$ C_2: $FP(john) \leftarrow$
C_3: $FP(mary) \leftarrow$ $\quad$ C_4: $teach(john, ai) \leftarrow$
C_5: $St(paul) \leftarrow$ $\quad$ C_6: $AC(ai) \leftarrow$
C_7: $Tp(kr) \leftarrow$ $\quad$ C_8: $Tp(lp) \leftarrow$

C_9: $curr(x, z) \leftarrow exam(x, y), subject(y, z), St(x), Co(y), Tp(z)$
C_{10}: $mdt(x, y) \leftarrow curr(x, z), expert(y, z), St(x), Tp(z), FP(y), AC(w), teach(y, w)$
C_{11}: $mdt(x, y) \leftarrow St(x), NFP(y)$

C_{12}: $exam(paul, ai) \leftarrow$ $\quad$ C_{13}: $subject(ai, kr) \leftarrow$
C_{14}: $subject(ai, lp) \leftarrow$ $\quad$ C_{15}: $expert(john, kr) \leftarrow$
C_{16}: $expert(mary, lp) \leftarrow$

C_{17}: $AC(x) \leftarrow teach(mary, x)$ $\quad$ C_{18}: $\leftarrow AC(x), BC(x)$
C_{19}: $AC(x), BC(x) \leftarrow Co(x)$ $\quad$ C_{20}: $Co(x) \leftarrow AC(x)$
C_{21}: $Co(x) \leftarrow BC(x)$ $\quad$ C_{22}: $FP(x) \leftarrow NFP(x)$

C_{23}: $\leftarrow NFP(x), teach(x, y), Co(y)$
C_{24}: $teach(x, y), NFP(x) \leftarrow FP(x), func(f_0, x, y)$
C_{25}: $Co(y), NFP(x) \leftarrow FP(x), func(f_0, x, y)$

FIGURE 1. Background knowledge for the *mdt* problem on $\mathrm{ECLS_F}$

Suppose that we want to find all professors with whom *paul* may do his thesis. This is a QA problem and is formalized as an MI problem $\langle Cs, \varphi \rangle$, where φ is a mapping used for constructing the output answer from the intersection of all models of Cs, defined by

$$\varphi(G) = \{x \mid mdt(paul, x) \in G\}$$

for any set G of ground user-defined atoms.

3. **Unfolding-Based Computation and Inapplicability of Unfolding.** In the existence of multi-head clauses, repeated application of the unfolding rule may result in an inapplicable state for unfolding, which is demonstrated by using the *mdt* problem.

3.1. **Inapplicability of unfolding.** Since unfolding is a definite-clause-based operation, the existence of multi-head clauses makes it difficult to apply unfolding.

Proposition 3.1. *Assume that Cs is a set of clauses and C is a clause in Cs. Assume that a is an atom in the right-hand side of C. If there is a multi-head clause C' in Cs such that $lhs(C')$ contains an atom b that is unifiable with a, then we cannot apply unfolding to Cs at a.*

Proof: Suppose that C' is a multi-head clause in Cs and $lhs(C')$ contains an atom b that is unifiable with a. Assume that unfolding Cs at a is applicable. Then there is a set D of definite clauses such that each clause in $Cs - D$ does not have any atom in its left-hand side that is unifiable with a. Since C' is a multi-head clause, C' is in $Cs - D$. Since b is unifiable with a, a contradiction occurs. □

3.2. **Computation by unfolding-based rules.** Given a set Cs of clauses, a body atom a in a clause C in Cs is selected to unfold Cs using a set D of definite clauses that is a subset of Cs. To unfold Cs at a, any atom in the head of a clause in $Cs - D$ should not be unifiable with a. We assume in this paper that D is the set of all definite clauses in Cs whose heads are unifiable with a. By the unfolding operation, the clause C is removed and each definite clause in D produces a clause called a resolvent. Hence the number of clauses changes from $|Cs|$ into $|Cs| - 1 + |D|$, i.e., the clause C splits into $|D|$ clauses and the number of clauses increases by $|D| - 1$. For instance, assume that $FP(y)$ is a selected body atom in C_{10}. Then D is the set $\{C_2, C_3, C_{22}\}$ and C_{10} splits into the following three clauses:

$mdt(x, john) \leftarrow curr(x, z), expert(john, z), St(x), Tp(z), AC(w), teach(john, w)$
$mdt(x, mary) \leftarrow curr(x, z), expert(mary, z), St(x), Tp(z), AC(w), teach(mary, w)$
$mdt(x, y) \leftarrow curr(x, z), expert(y, z), St(x), Tp(z), NFP(y), AC(w), teach(y, w)$

Consider C_1 in Figure 1. If there is an FM atom in the body of a clause in Cs, the singleton set $\{C_1\}$ can be a set D of definite clauses for unfolding Cs. We know that we can remove C_1 from Cs equivalently by an ET rule, called the definite-clause removal rule.

When only the unfolding rule and the definite-clause removal rule are applied to the clause set P_0 consisting of C_1-C_{25} in Figure 1 with the priority of ET rules

$$(\text{rm}) \geq (\text{ud}),$$

where

- (rm) removes definite clauses by the definite-clause removal rule, and
- (ud) unfolds a given clause set at a selected body atom in a selected clause using a set of definite clauses,

we have the following 32 rule applications:

$(rm, FM), (ud, exam, 1), (rm, exam), (ud, curr, 1), (rm, curr), (ud, St, 1),$
$(ud, FP, 3), (ud, FP, 3), (ud, expert, 2), (rm, expert), (ud, St, 1), (ud, St, 1),$
$(ud, Tp, 1), (ud, Tp, 1), (ud, FP, 2), (ud, FP, 2), (rm, FP), (ud, subject, 1),$
$(ud, subject, 1), (ud, subject, 1), (ud, subject, 1), (rm, subject), (ud, St, 1),$
$(ud, St, 1), (ud, St, 1), (ud, St, 1), (rm, St), (ud, Tp, 1), (ud, Tp, 1),$
$(ud, Tp, 1), (ud, Tp, 1), (rm, Tp),$

where $(rm, \langle p \rangle)$ means removing the definition of a predicate $\langle p \rangle$, and $(ud, \langle p \rangle, \langle n \rangle)$ means unfolding at a $\langle p \rangle$-atom with $\langle n \rangle$ splitting resolvents.

After these applications, we reach a set P_1 consisting of the clauses shown in Figure 2. Neither the unfolding rule nor the definite-clause removal rule is applicable to this state P_1. The clause set $\{C'_9, C'_{16}, C'_{17}, C'_{18}, C'_{19}\}$ defines the predicate mdt. However, we cannot remove it by the definite-clause removal rule since the predicate mdt is necessary for answering the original problem. User-defined body atoms in the right-hand side of the clauses in Figure 2 are $teach(mary, x)$, $AC(x)$, $BC(x)$, $Co(x)$, $NFP(x)$, $teach(x, y)$, $NFP(john)$,

C'_1: $teach(john, ai) \leftarrow$
C'_2: $AC(ai) \leftarrow$
C'_3: $AC(x) \leftarrow teach(mary, x)$
C'_4: $\leftarrow AC(x), BC(x)$
C'_5: $AC(x), BC(x) \leftarrow Co(x)$
C'_6: $Co(x) \leftarrow AC(x)$
C'_7: $Co(x) \leftarrow BC(x)$
C'_8: $\leftarrow NFP(x), teach(x, y), Co(y)$
C'_9: $mdt(paul, x) \leftarrow NFP(x)$
C'_{10}: $teach(john, x), NFP(john) \leftarrow func(f0, john, x)$
C'_{11}: $teach(mary, x), NFP(mary) \leftarrow func(f0, mary, x)$
C'_{12}: $teach(x, y), NFP(x) \leftarrow func(f0, x, y), NFP(x)$
C'_{13}: $Co(x), NFP(john) \leftarrow func(f0, john, x)$
C'_{14}: $Co(x), NFP(mary) \leftarrow func(f0, mary, x)$
C'_{15}: $Co(x), NFP(y) \leftarrow func(f0, y, x), NFP(y)$
C'_{16}: $mdt(paul, john) \leftarrow AC(x), teach(john, x), Co(ai)$
C'_{17}: $mdt(paul, john) \leftarrow AC(x), teach(john, x), Co(ai), NFP(john)$
C'_{18}: $mdt(paul, mary) \leftarrow AC(x), teach(mary, x), Co(ai)$
C'_{19}: $mdt(paul, mary) \leftarrow AC(x), teach(mary, x), Co(ai), NFP(mary)$

FIGURE 2. P_1: Clauses obtained only by unfolding-based rules

$teach(john, x)$, $Co(ai)$, $NFP(mary)$. However, none of these atoms can be used as a selected atom to unfold the clause set P_1. Consider the body atom $teach(john, x)$. The set D of all definite clauses whose head atoms are unifiable with $teach(john, x)$ is $\{C'_1\}$. The clause C'_{10} is a multi-head clause in P_1 and C'_{10} is in $P_1 - D$. By Proposition 3.1, we cannot unfold P_1 at $teach(john, x)$.

3.3. **Blocked states in unfolding-based computation.** For further transformation, we adopt more ET rules and take the following priority of ET rules:

$$\text{(dup)} \geq \text{(valid)} \geq \text{(subsumed)} \geq \text{(erase)} \geq \text{(func-erase)} \geq \text{(rm)} \geq \text{(ud)},$$

where

- (dup) removes duplicated atoms in the left-hand side of a clause or those in the right-hand side,
- (valid) removes a clause that contains the same atom occurring in both its left-hand side and its right-hand side,
- (subsumed) removes a subsumed clause from a clause set,
- (erase) erases independent satisfiable atoms, and
- (func-erase) erases an independent satisfiable *func*-atom.

Using this control, the clause set P_0 is equivalently transformed into the clause set $P_2 = \{C_{26}, C_{27}, \ldots, C_{40}\}$ in Figure 3, using the 27 rule applications shown in the following list:

(rm, FM), $(ud, exam, 1)$, $(rm, exam)$, $(ud, curr, 1)$, (dup), $(rm, curr)$, $(ud, St, 1)$, $(ud, FP, 3)$, $(valid)$, $(ud, FP, 3)$, $(valid)$, $(ud, expert, 2)$, $(rm, expert)$, $(ud, FP, 2)$, $(subsumed)$, $(ud, subject, 1)$, $(ud, FP, 2)$, $(subsumed)$, (rm, FP), $(ud, subject, 1)$, $(rm, subject)$, $(ud, St, 1)$, $(ud, St, 1)$, (rm, St), $(ud, Tp, 1)$, $(ud, Tp, 1)$, (rm, Tp).

None of (dup), (valid), (subsumed), (erase), (func-erase), and (ud) is applicable to P_2. Again, we cannot unfold P_2. By Proposition 3.1, the existence of multi-head clauses blocks unfolding as follows:

- Unfolding of the predicates AC and BC are blocked since C_{30} contains an AC-atom and a BC-atom in its left-hand side.
- Unfolding of the predicate Co is blocked since C_{39} and C_{40} contain Co-atoms and NFP-atoms in their left-hand sides.
- Unfolding of the predicate *teach* is blocked since C_{37} and C_{38} contain *teach*-atoms and NFP-atoms in their left-hand sides.
- Unfolding of the predicate *mdt* is blocked since the answer to the problem is influenced by *mdt*-atoms.

For further application of unfolding, moving atoms in the left-hand side of a clause to its right-hand side is useful. We will introduce side-change transformation in the next section.

C_{26}: $teach(john, ai) \leftarrow$
C_{27}: $AC(ai) \leftarrow$
C_{28}: $AC(x) \leftarrow teach(mary, x)$
C_{29}: $\leftarrow AC(x), BC(x)$
C_{30}: $AC(x), BC(x) \leftarrow Co(x)$
C_{31}: $Co(x) \leftarrow AC(x)$
C_{32}: $Co(x) \leftarrow BC(x)$
C_{33}: $\leftarrow NFP(x), teach(x, y), Co(y)$
C_{34}: $mdt(paul, mary) \leftarrow AC(x), teach(mary, x),\ Co(ai)$
C_{35}: $mdt(paul, john) \leftarrow AC(x), teach(john, x), Co(ai)$
C_{36}: $mdt(paul, x) \leftarrow NFP(x)$
C_{37}: $teach(mary, x), NFP(mary) \leftarrow func(f_0, mary, x)$
C_{38}: $teach(john, x), NFP(john) \leftarrow func(f_0, john, x)$
C_{39}: $Co(x), NFP(mary) \leftarrow func(f_0, mary, x)$
C_{40}: $Co(x), NFP(john) \leftarrow func(f_0, john, x)$

FIGURE 3. Clauses obtained by transformation of C_1-C_{25} in Figure 1

4. Side-Change Transformation and Its Usage.

We introduce side-change transformation and describe two types of its usage. We then show that side-change transformation preserves the answers to MI problems.

4.1. Side-change transformation.

Let Cs be a clause set and p a predicate occurring in Cs. A p-atom is an atom (atomic formula) that has the predicate p. The clause set Cs can be transformed by *side-change transformation* for p-atoms (also called side-change transformation for p) as follows.

1) Determine a new predicate *notp* for p.
2) Move each p-atom in each clause to its opposite side in the same clause (i.e., from the left-hand side to the right-hand side and vice versa) with its predicate being changed from p to *notp*.

For example, a singleton clause set

$$\{(p(1, x), q(x, y) \leftarrow p(4, x), p(5, x), r(y))\}$$

is transformed by side-change transformation for p-atoms into

$$\{(notp(4, x), notp(5, x), q(x, y) \leftarrow notp(1, x), r(y))\}.$$

Let the clause set obtained from Cs by side-change transformation for p-atoms into *notp*-atoms be denoted by $\textsc{SideCh}(Cs, p, notp)$.

4.2. **Sufficient conditions for enabling unfolding transformation.** Let Cs be a set of clauses. To apply unfolding or definite-clause removal to Cs, we need to determine a set D of definite clauses and a predicate p such that i) D is a subset of Cs, ii) the head of each definite clause in D is a p-atom, and iii) no p-atom appears in the left-hand side of any clause in $Cs - D$.

When unfolding is not applicable, side-change transformation is useful for deriving from Cs a new set of clauses to which unfolding or definite-clause removal can be applied. Two types of its usage that enable unfolding are described below.

4.2.1. *Type 1.* Let p be a predicate and *notp* a new predicate determined for side-change transformation for p. Unfolding with respect to *notp*-atoms can be applied after side-change transformation for p if each clause C in Cs satisfies the following two conditions.

1) There is at most one p-atom in the right-hand side of C.
2) If there is exactly one p-atom in the right-hand side of C, then all user-defined atoms in the left-hand side of C are p-atoms.

Example 4.1. *Consider the following clauses:*

C_a: $q(x), p(5, x) \leftarrow r(x)$
C_b: $\leftarrow p(x, y), r(y, w), s(y, w)$
C_c: $p(x, z) \leftarrow p(x, y), r(y, z)$
C_d: $p(x, w), p(x, 2) \leftarrow p(x, y), t(w)$

Each of these clauses satisfies the two conditions above. By side-change transformation for p, these four clauses are transformed into

C'_a: $q(x) \leftarrow r(x), notp(5, x)$
C'_b: $notp(x, y) \leftarrow r(y, w), s(y, w)$
C'_c: $notp(x, y) \leftarrow notp(x, z), r(y, z)$
C'_d: $notp(x, y) \leftarrow notp(x, w), notp(x, 2), t(w)$

We obtain the set $\{C'_b, C'_c, C'_d\}$ of definite clauses for the predicate notp, which can be used for unfolding with respect to $notp(5, x)$ in the right-hand side of the first clause C'_a.

Assume that D_{notp} is the set of all definite clauses for *notp* in Cs. If there is no *notp*-atom in the right-hand side of any clause in $Cs - D_{notp}$, then D_{notp} can be removed by definite-clause removal transformation.

4.2.2. *Type 2.* Assume that p and q are distinct predicates. Unfolding with respect to q-atoms can be applied after side-change transformation for p if each clause C in Cs satisfies the following two conditions.

1) There is at most one q-atom in the left-hand side of C.
2) If there is exactly one q-atom in the left-hand side of C, then i) all other user-defined atoms in the left-hand side of C are p-atoms and ii) no p-atom appears in the right-hand side of C.

Example 4.2. *Consider the following four clauses:*

C_e: $s(x, y) \leftarrow q(x, y)$
C_f: $q(x, y) \leftarrow r(y)$
C_g: $p(x, y), q(x, y) \leftarrow r(x), s(y)$
C_h: $p(x, y), p(x, z), q(x, y) \leftarrow t(y, z)$

Each of them satisfies the two conditions for Type 2. By side-change transformation for p, these clauses are changed into

C'_e: $s(x,y) \leftarrow q(x,y)$
C'_f: $q(x,y) \leftarrow r(y)$
C'_g: $q(x,y) \leftarrow notp(x,y), r(x), s(y)$
C'_h: $q(x,y) \leftarrow notp(x,y), notp(x,z), t(y,z)$

We obtain the set $\{C'_f, C'_g, C'_h\}$ *of definite clauses for the predicate* q*, which can be used for unfolding with respect to* $q(x,y)$ *in the right-hand side of* C'_e*.*

Assume that D_q is the set of all definite clauses for q in Cs. If there is no q-atom in the right-hand side of any clause in $Cs - D_q$, then D_q can be removed by definite-clause removal transformation.

4.3. **Correctness theorem for side-change transformation.** We show below that side-change transformation is answer-preserving.

Theorem 4.1. *Let* $\langle Cs, \varphi \rangle$ *be an MI problem on* $\mathrm{ECLS_F}$*. Let* p *be a predicate occurring in* Cs *and* $notp$ *a new predicate determined for side-change transformation for* p*. If* φ *is independent of* $\{p, notp\}$*, then* $ans_{\mathrm{MI}}(Cs, \varphi) = ans_{\mathrm{MI}}(\textsc{SideCh}(Cs, p, notp), \varphi)$.

Proof: Assume that i) φ is independent of $\{p, notp\}$, ii) $Cs' = \textsc{SideCh}(Cs, p, notp)$, and iii) π_p is a mapping that changes p-atoms into $notp$-atoms. Let m be an arbitrary model of Cs. Let a set $m' \subseteq \mathcal{G}_{\mathrm{u}}$ be constructed from m by

$$m' = (m - GAtoms(\{p\})) \cup \{\pi_p(x) \mid x \in (GAtoms(\{p\}) - m)\}.$$

Obviously, m' is a model of Cs'. This construction determines a mapping ψ from $Models(Cs)$ to $Models(Cs')$. Obviously, ψ^{-1} is a mapping from $Models(Cs')$ to $Models(Cs)$ and $\psi \circ \psi^{-1}$ is an identity mapping. Hence ψ is a bijective mapping from $Models(Cs)$ to $Models(Cs')$. As a result,

- $\bigcap Models(Cs) - \bigcap Models(Cs') \subseteq GAtoms(\{p, notp\})$, and
- $\bigcap Models(Cs') - \bigcap Models(Cs) \subseteq GAtoms(\{p, notp\})$.

Since the extraction mapping φ is independent of $\{p, notp\}$, it follows that $\varphi(\bigcap Models(Cs))$ is equal to $\varphi(\bigcap Models(Cs'))$. □

5. **Rule Application Control with Side-Change Transformation.** We solve the *mdt* problem by unfolding-based computation with side-change transformation.

Solution to the *mdt* problem. Let (chSide) denote the side-change transformation rule and (r) denote the resolution rule. We solve the *mdt* problem by using the following priority of ET rules

(dup) ≥ (valid) ≥ (subsumed) ≥ (erase) ≥ (func-erase) ≥ (rm) ≥ (ud) ≥ (chSide) ≥ (r),

which realizes unfolding-based computation with side-change transformation. Transformation starting from the clause set $P_0 = \{C_1, C_2, \ldots, C_{25}\}$ (see Figure 1) is detailed as follows.

1) Before the first application of (chSide), the clause set P_0 is transformed into the clause set $P_2 = \{C_{26}, C_{27}, \ldots, C_{40}\}$ (see Figure 3), where (dup) is applied once, (valid) 2 times, (subsumed) 2 times, (rm) 8 times, and (ud) 14 times. The definitions of the predicates *FM*, *FP*, *Tp*, *curr*, *subject*, *expert*, *St*, and *exam* are used for unfolding and removed. The predicate *NFP* disallows unfolding P_2, which is shown by the following clauses contained in P_2:

   C_{33}: $\leftarrow NFP(x), teach(x,y), Co(y)$
   C_{36}: $mdt(paul, x) \leftarrow NFP(x)$
   C_{37}: $teach(mary, x), NFP(mary) \leftarrow func(f_0, mary, x)$

C_{38}: $teach(john, x), NFP(john) \leftarrow func(f_0, john, x)$
C_{39}: $Co(x), NFP(mary) \leftarrow func(f_0, mary, x)$
C_{40}: $Co(x), NFP(john) \leftarrow func(f_0, john, x)$

2) Side-change transformation for NFP changes the clauses C_{33}, C_{36}, C_{37}, C_{38}, C_{39}, and C_{40} into the clauses C'_{33}, C'_{36}, C'_{37}, C'_{38}, C'_{39}, and C'_{40} given by

C'_{33}: $notNFP(x) \leftarrow teach(x, y), Co(y)$
C'_{36}: $mdt(paul, x), notNFP(x) \leftarrow$
C'_{37}: $teach(mary, x) \leftarrow notNFP(mary), func(f_0, mary, x)$
C'_{38}: $teach(john, x) \leftarrow notNFP(john), func(f_0, john, x)$
C'_{39}: $Co(x) \leftarrow notNFP(mary), func(f_0, mary, x)$
C'_{40}: $Co(x) \leftarrow notNFP(john), func(f_0, john, x)$

After that, (rm) is applied 2 times, (ud) 9 times, (dup) 3 times, (valid) 5 times, and (subsumed) 5 times, where the predicates $teach$ and Co are unfolded and their definitions are removed.

3) Side-change transformation for BC enables i) unfolding using the definition of AC, ii) definite-clause removal, and iii) application of basic reduction rules. The resolution rule and basic reduction rules are then applied. (rm) is applied 2 times, (ud) 5 times, (dup) 2 times, (valid) 2 times, (subsumed) 15 times, and (func-erase) once. The list of rule applications is shown below.

$(chSide, BC), (ud, AC, 3), (subsumed), (subsumed), (subsumed), (subsumed),$
$(subsumed), (ud, AC, 3), (valid), (subsumed), (subsumed), (subsumed),$
$(ud, AC, 3), (dup), (subsumed), (subsumed), (func\text{-}erase), (ud, AC, 3), (dup),$
$(subsumed), (subsumed), (subsumed), (subsumed), (subsumed),$
$(ud, AC, 3), (valid), (rm, AC), (rm, notBC).$

The following clauses are obtained.

C'_{41}: $notNFP(john) \leftarrow$
C'_{42}: $mdt(paul, john) \leftarrow$
C'_{43}: $mdt(paul, mary) \leftarrow notNFP(mary)$
C'_{44}: $mdt(paul, x), notNFP(x) \leftarrow$

4) Side-change transformation for $notNFP$ is applied, where $notNFP$ is the predicate determined by side-change transformation for NFP at Step 2. The following clauses are obtained.

C''_{41}: $\leftarrow NFP(john)$
C''_{42}: $mdt(paul, john) \leftarrow$
C''_{43}: $mdt(paul, mary), notNFP(mary) \leftarrow$
C''_{44}: $mdt(paul, x) \leftarrow notNFP(x)$

After that, computation continues. (ud) is applied 2 times, (dup) once, (subsumed) once, and (r) once. The clause C''_{41} is removed by unfolding. By resolution, C''_{43} and C''_{44} produce the clause $(mdt(paul, mary),\ mdt(paul, mary) \leftarrow)$, which is simplified into $(mdt(paul, mary) \leftarrow)$ by (dup). C''_{43} is removed by (subsumed). C''_{44} is removed by unfolding. The list of rule applications is shown below.

$(chSide, (notNFP)), (ud, NFP, 0), (r), (dup), (subsumed), (ud, NFP, 0).$

The atom $notNFP(mary)$ cannot be removed by (ud) and is removed by the resolution rule (r). Now we obtain the clauses C_{41} and C_{42} given as follows:

C_{41}: $mdt(paul, mary) \leftarrow$
C_{42}: $mdt(paul, john) \leftarrow$

As a result, the MI problem $\langle Cs, \varphi \rangle$ in Section 2.5 is transformed equivalently into the MI problem $\langle \{C_{41}, C_{42}\}, \varphi \rangle$. Hence the answer to the original MI problem $\langle Cs, \varphi \rangle$

coincides with the answer to the simpler MI problem $\langle\{C_{41}, C_{42}\}, \varphi\rangle$, which can be readily determined as the set

$$\varphi\left(\bigcap \mathit{Models}(\{C_{41}, C_{42}\})\right) = \varphi(\{\mathit{mdt}(\mathit{paul}, \mathit{mary}), \mathit{mdt}(\mathit{paul}, \mathit{john})\}) = \{\mathit{mary}, \mathit{john}\},$$

where $\mathit{Models}(\{C_{41}, C_{42}\})$ denotes the set of all models of $\{C_{41}, C_{42}\}$.

6. **Conclusions.** MI problems on the extended space $\mathrm{ECLS_F}$ constitute one of the largest classes of logical problems and are of fundamental importance. Since all proof problems on the first-order formula space with built-in constraint atoms and all QA problems on the same space can be mapped into MI problems on $\mathrm{ECLS_F}$, improvements to a solution method for MI problems on $\mathrm{ECLS_F}$ are useful for efficiently solving logical problems. We introduced side-change transformation on $\mathrm{ECLS_F}$ and proved the correctness of this transformation. We proposed side-change transformation and unfold/unblock computation control in the framework of prioritized ET rules, by which we expect a continuous reduction of problem size. Such a reduction often contributes to more efficient computation. This computation strategy is in sharp contrast to the resolution proof procedure, which monotonically increases the problem size regardless of whatever control it may take. Invention of ET rules that are useful for reduction of problem size is the most important future work in the ET-based computation paradigm, by which we expect to construct an efficient MI solver for proof and QA problems with flexible control.

Acknowledgment. This work was partially supported by i) JSPS KAKENHI Grant Numbers 25280078 and 26540110, ii) the Center of Excellence in Intelligent Informatics, Speech and Language Technology and Service Innovation (CILS), Thammasat University, and iii) the Intelligent Informatics and Service Innovation (IISI) Center, Sirindhorn International Institute of Technology (SIIT), Thammasat University. The authors also gratefully acknowledge the helpful comments and suggestions from the reviewers.

REFERENCES

[1] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[2] J. R. Slagle, Automatic theorem proving with renamable and semantic resolution, *Journal of the ACM*, vol.14, pp.687-697, 1967.
[3] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.
[4] B. Motik, U. Sattler and R. Studer, Query answering for OWL-DL with rules, *Journal of Web Semantics*, vol.3, no.1, pp.41-60, 2005.
[5] D. Luckham, Refinement theorems in resolution theory, *Proc. of the 1968 IRIA Symp. Automatic Demonstration*, Versailles, France, pp.163-190, 1970.
[6] D. W. Loveland, A linear format for resolution, *Proc. of the 1968 IRIA Symp. Automatic Demonstration*, Versailles, France, pp.147-162, 1970.
[7] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, *The Description Logic Handbook*, 2nd Edition, Cambridge University Press, 2007.
[8] F. J. Pelletier, Seventy-five problems for testing automatic theorem provers, *Journal of Automated Reasoning*, vol.2, no.2, pp.191-216, 1986.
[9] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.
[10] F. M. Donini, M. Lenzerini, D. Nardi and A. Schaerf, $\mathcal{AL}$-log: Integrating datalog and description logics, *Journal of Intelligent Information Systems*, vol.16, pp.227-252, 1998.
[11] C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
[12] R. S. Boyer, *Locking: A Restriction of Resolution*, Ph.D. Thesis, University of Texas at Austin, Texas, 1971.
[13] K. Akama and E. Nantajeewarawat, Solving query-answering problems with constraints for function variables, *Lecture Notes in Artificial Intelligence*, vol.10751, pp.36-47, 2018.

[14] K. Akama and E. Nantajeewarawat, Model-intersection problems with existentially quantified function variables: Formalization and a solution schema, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol.2, Porto, Portugal, pp.52-63, 2016.
[15] K. Akama and E. Nantajeewarawat, Equivalent transformation in an extended space for solving query-answering problems, *Proc. of the 6th Asian Conference on Intelligent Information and Database Systems*, Bangkok, Thailand, pp.232-241, 2014.
[16] K. Akama and E. Nantajeewarawat, Solving proof problems with equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.1, pp.331-344, 2022.
[17] K. Akama and E. Nantajeewarawat, Solving query-answering problems based-on equivalent transformation, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1547-1558, 2022.
[18] K. Akama and E. Nantajeewarawat, A foundation of logical problem solving, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1559-1570, 2022.
[19] K. Akama and E. Nantajeewarawat, Meaning-preserving skolemization, *Proc. of the 3rd International Conference on Knowledge Engineering and Ontology Development*, Paris, France, pp.322-327, 2011.

Author Biography

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

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

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

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