International Journal of Innovative
Computing, Information and Control
Volume **19**, Number **6**, December **2023**

ICIC International ©2023 ISSN 1349-4198
pp. **1907–1918**

SOLVING MODEL-INTERSECTION PROBLEMS WITH CONSTRAINTS ATTACHED TO FUNCTION VARIABLES

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 February 2023; revised May 2023

Abstract. *The set of all model-intersection problems (MI problems) constitutes a very large class of logical problems. Formalization of logical problems as MI problems and computation by equivalent transformation (ET) are of fundamental importance for establishing a general solution method for solving deductive problems on first-order formulas, overcoming the limitations of the conventional methods that are based on inference within the first-order formula space. In this paper, we extend the space of clauses by introducing attachment of constraints to function variables and invent three ET rules for simplifying problem descriptions by using interaction between clauses and constraints. This extension provides a general solution for a larger class of MI problems.*
Keywords: Model-intersection problem, Equivalent transformation rule, Logical problem solving framework, Function variable, Constraint for function variable

1. Introduction.

1.1. Reconstruction of the structure of logic.

Logical structure is of fundamental and central importance for human intelligence. The conventional theory of logic is proof-centered and inference-based one on the first-order formula space [1, 2, 3, 4, 5]. It is, however, unsatisfactory for solvability of logical problems. The class of problems solved by conventional inference-based methods is not large enough, i.e., the problem solvability based on SLD resolution (selective linear definite clause resolution) [3] is rather low. Many logical problems cannot be solved by conventional methods correctly. SLD resolution is incomplete, both for the class of all proof problems and for the class of all QA problems that are defined by using first-order formulas [6]. The conventional computation theory based on SLD resolution has such theoretical limitations.

The theory of logic should be reconstructed extensively, seeking appropriate structure to capture human intelligence and to maximize representational and computational power. To break the representational and computational limitations of first-order formulas, LPSF (logical problem solving framework) was invented [7]. LPSF is an axiomatic logic and can be applied to conventional logics. Using LPSF, we can also generate a new logic by extending problem classes, computation methods, and the underlying formula space. In place of proof problems, we use *model-intersection problems* (*MI problems*), which are a

DOI: 10.24507/ijicic.19.06.1907

superclass of proof problems, and also include the class of query-answering (QA) problems. Computation in LPSF is successive application of an unlimited number of *equivalent transformation* (*ET*) rules. Computation by ET rules truly extends the solvability of logical problems. We solve MI problems by using ET rules on some formula space.

Let FOL be the set of all first-order formulas without built-in constraints. Let CLS be the set of all (usual) clauses without built-in constraints, corresponding to FOL. A typical formula space is FOL or CLS. We take a general approach to dealing with MI problems, which is based on the ET solution method, i.e., a given MI problem is transformed equivalently into simpler forms until its answer set can be readily obtained. The correctness of this new solution method is guaranteed based on the ET principle.

1.2. **Correct and efficient computation based on space extension.** Built-in constraint atoms play a crucial role in practical knowledge representation [8]. We need to consider first-order formulas that possibly contain built-in constraint atoms. The set of all such formulas is denoted by $\mathrm{FOL_c}$. For practical applications, we need space extension from FOL to $\mathrm{FOL_c}$. Formulas in $\mathrm{FOL_c}$ are transformed using Skolemization into clausal forms in $\mathrm{CLS_c}$, where an element of $\mathrm{CLS_c}$ is a clause possibly with built-in constraint atoms.

However, some formula in $\mathrm{FOL_c}$ cannot be transformed into a set of clauses in $\mathrm{CLS_c}$ preserving models. For transformation of QA problems on $\mathrm{FOL_c}$ into equivalent clausal forms, we have developed meaning-preserving Skolemization [9] together with a new extended space, called the $\mathrm{ECLS_F}$ space. This extended space includes function variables, which are variables ranging over function constants. The underlying clause space extension is illustrated as follows: $\mathrm{CLS} \subset \mathrm{CLS_c} \subset \mathrm{ECLS_F}$.

The set of all MI problems on extended clauses in $\mathrm{ECLS_F}$ constitutes a very large class of problems and is of great importance [6]. The class of MI problems on $\mathrm{ECLS_F}$ enables structural embedding of the full class of proof problems on $\mathrm{FOL_c}$ and the full class of QA problems on $\mathrm{FOL_c}$. All proof problems and all QA problems on $\mathrm{FOL_c}$ can be mapped, preserving their answers, into MI problems on $\mathrm{ECLS_F}$. By solving MI problems on $\mathrm{ECLS_F}$, we can solve proof problems and QA problems on $\mathrm{FOL_c}$.

1.3. **A new space and new ET rules.** Seeking efficient computation, we wish to apply less-splitting ET rules at each computation state. When we reach a computation state to which no small-splitting ET rule is applied, extension of the underlying formula space and application of a new ET rule in the new space may be useful.

One example is the Agatha puzzle (the "Dreadsbury Mansion Mystery" problem), which will be introduced in Section 2. Assume that we are in the space of $\mathrm{ECLS_F}$, and computation by ET starting from the initial Agatha problem terminates unsuccessfully without plausible further reduction by any ET rule developed so far. Then we try to proceed towards a final goal by

- introduction of the concept of constraint attached to a function variable, which results in a new clause space called $\mathrm{ECLS_{FC}}$ (Section 4), and
- invention of new ET rules (for positive function-variable restriction, negative function-variable restriction, and *func*-atom evaluation) for dealing with *func*-atoms and constraints (Section 5).

The underlying space extension explained so far is shown as follows:

$$\mathrm{CLS} \subset \mathrm{CLS_c} \subset \mathrm{ECLS_F} \subset \mathrm{ECLS_{FC}}.$$

The extended space $\mathrm{ECLS_{FC}}$ together with addition of the ET rules mentioned above makes it possible to solve a larger class of MI (and thus proof and QA) problems.

1.4. **Organization.** The rest of the paper is organized as follows. Section 2 formalizes the Agatha puzzle, and shows unsuccessfully terminated computation for the puzzle. Section 3 gives successful computation for the Agatha puzzle, and states the main objective of the paper. Section 4 introduces an extended clause space, where body atoms of each clause may contain function variables with possibly non-empty constraints being attached to them. Section 5 introduces three ET rules for computation in the extended clause space, and proves the correctness of the rules. Section 6 gives experiments for Agatha proof problems and Agatha QA problems, and compares them to the unsuccessful results produced by the conventional method based on Skolemization and resolution. Section 7 provides conclusions.

2. An Introductory Example.

We formalize the Agatha puzzle, convert it to a clausal form, and illustrate an initial part of an ET computation sequence for the puzzle.

2.1. **Dreadsbury Mansion mystery.** Consider the "Dreadsbury Mansion Mystery" problem (the Agatha puzzle), which is described as follows: Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadsbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Agatha hates. No one hates everyone. The problem is to find who is the killer.

Assume that eq and neq are predefined binary constraint predicates and for any ground usual terms t_1 and t_2, (i) $eq(t_1, t_2)$ is true iff $t_1 = t_2$, and (ii) $neq(t_1, t_2)$ is true iff $t_1 \neq t_2$. The background knowledge of this mystery is formalized as the conjunction of the first-order formulas in Figure 1, where (i) the constants A, B, C, and D denote "Agatha", "the butler", "Charles", and "Dreadsbury Mansion", respectively, and (ii) for any terms t_1 and t_2, the atoms $live(t_1, t_2)$, $kill(t_1, t_2)$, $hate(t_1, t_2)$, and $richer(t_1, t_2)$ are intended to mean "t_1 lives in t_2", "t_1 killed t_2", "t_1 hates t_2", and "t_1 is richer than t_2", respectively.

$$
\begin{array}{l}
\exists x : (live(x, D) \wedge kill(x, A)) \\
\forall x : (live(x, D) \leftrightarrow (eq(x, A) \vee eq(x, B) \vee eq(x, C))) \\
\forall x \forall y : (kill(x, y) \rightarrow hate(x, y)) \\
\forall x \forall y : (kill(x, y) \rightarrow \neg richer(x, y)) \\
\neg(\exists x : (hate(C, x) \wedge hate(A, x) \wedge live(x, D))) \\
\forall x : ((neq(x, B) \wedge live(x, D)) \rightarrow hate(A, x)) \\
\forall x : ((\neg richer(x, A) \wedge live(x, D)) \rightarrow hate(B, x)) \\
\forall x : ((hate(A, x) \wedge live(x, D)) \rightarrow hate(B, x)) \\
\neg(\exists x : (live(x, D) \wedge (\forall y : (live(y, D) \rightarrow hate(x, y))))) \\
\forall x : (kill(x, A) \rightarrow killer(x))
\end{array}
$$

FIGURE 1. Background knowledge represented by first-order formulas

The sentence "No one hates everyone" in the Agatha puzzle can be translated into the sentence "There is no one who hates everyone", which is represented by the first-order formula $\neg(\exists x : (\forall y : hate(x, y)))$. When "No one" is interpreted as "No one who lives in Dreadsbury Mansion", the subexpression "$live(x, D)\wedge$" is added before $(\forall y : hate(x, y))$. Similarly, when "everyone" is interpreted as "everyone who lives in Dreadsbury Mansion", the subexpression "$live(y, D) \rightarrow$" is added before $hate(x, y)$. Since the sentence "No one hates everyone" is translated in the context of this puzzle into the sentence "There is no one who lives in Dreadsbury Mansion and hates everyone who lives in Dreadsbury

Mansion", Figure 1 contains the first-order formula $\neg(\exists x : (live(x, D) \wedge (\forall y : (live(y, D) \rightarrow hate(x, y)))))$.

2.2. **Formalization of the puzzle.** We understand the Agatha puzzle ("who killed aunt Agatha?") as finding all persons who killed Agatha. Formalizing this puzzle is to specify the answer (the set of all killers) in terms of the background knowledge of the puzzle.

Let K be the conjunction of the first-order formulas in Figure 1 and let $q = killer(x)$. The Agatha puzzle is formalized as a QA problem $\langle K, q\rangle$. The answer to this QA problem, denoted by $answer(K, q)$, is the set of all ground instances of q that follows logically from K. K is converted into the set Cs consisting of the fifteen extended clauses C_1-C_{15} in Figure 2 by applying meaning-preserving Skolemization [9]. Let $\mathcal{G}_{\mathrm{u}}$ be the set of all ground user-defined atoms. The QA problem $\langle K, q\rangle$ is then reformulated as a model-intersection (MI) problem $\langle Cs, \varphi\rangle$, where φ is a mapping from the power set of $\mathcal{G}_{\mathrm{u}}$ to the power set of $\{A, B, C\}$, defined by $\varphi(G) = \{t | killer(t) \in G\}$ for any $G \subseteq \mathcal{G}_{\mathrm{u}}$. In other words, we have

$$answer(K, q) = \varphi\left(\bigcap Models(Cs)\right),$$

where $Models(Cs)$ denotes the set of all models of Cs. Our plan for solving the Agatha puzzle is to simplify $\varphi(\bigcap Models(Cs))$ mainly by transforming Cs preserving $Models(Cs)$ or $\bigcap Models(Cs)$.

C_1: $live(x, D) \leftarrow func(f_0, x)$
C_2: $kill(x, A) \leftarrow func(f_0, x)$
C_3: $\leftarrow live(x, D), neq(x, A), neq(x, B), neq(x, C)$
C_4: $live(A, D) \leftarrow$
C_5: $live(B, D) \leftarrow$
C_6: $live(C, D) \leftarrow$
C_7: $hate(x, y) \leftarrow kill(x, y)$
C_8: $\leftarrow kill(x, y), richer(x, y)$
C_9: $\leftarrow hate(A, x), hate(C, x), live(x, D)$
C_{10}: $hate(A, x) \leftarrow neq(x, B), live(x, D)$
C_{11}: $richer(x, A), hate(B, x) \leftarrow live(x, D)$
C_{12}: $hate(B, x) \leftarrow hate(A, x), live(x, D)$
C_{13}: $\leftarrow hate(x, y), func(f_1, x, y), live(x, D)$
C_{14}: $live(y, D) \leftarrow live(x, D), func(f_1, x, y)$
C_{15}: $killer(x) \leftarrow kill(x, A)$

FIGURE 2. The initial state with extended clauses

2.3. **Computation for solving the Agatha puzzle.** In order to simplify the set of the clauses in Figure 2, we use many ET rules, including unfolding, definite-clause removal, and side-change transformation, given in our previous papers [8, 10, 11, 12]. The transformation process is shown below together with important final steps in Section 3.2.

1) By unfolding using the definition of *kill* (C_2), all body atoms with the predicate *kill* are removed. By definite-clause removal, the definition of *kill* (C_2) is removed. By unfolding, three body atoms with the patterns $hate(A, x)$ or $hate(C, x)$ are removed.
2) By side-change transformation for *richer*, the *richer*-atoms in some clauses are removed and *not_richer*-atoms are added to the other sides of these clauses.
3) By unfolding, *not_richer*-atoms and *hate*-atoms in clause bodies are removed. The definitions of *not_richer* and *hate* are removed.

4) Unfolding with respect to *live*-atoms has been suspended since one of the definite clauses defining *live* introduces a new *live*-atom with a pure variable as its first argument. To remedy the situation, the atom $live(x, D)$ in the body of a clause is specialized into three clauses, enabling further application of unfolding.
5) By unfolding *live*-atoms and definite-clause removal, all *live*-atoms are removed and the clauses in Figure 3 are obtained.

At this stage, unfolding and definite-clause removal were applied 17 times. Other rules that were also applied and their application counts are given as follows: elimination of subsumed clauses 22 times, elimination of true body atoms 10 times, elimination of duplicate atoms 5 times, constraint solving for *neq* 4 times, side-change transformation 1 time, and atom specialization 1 time. Fifteen clauses (C_1-C_{15}) consisting of 35 atoms are reduced to twelve clauses (C_{44}-C_{55}) with 23 atoms.

f_0: { }
f_1: { }
C_{44}: $\leftarrow neq(x, B), func(f_1, B, x)$
C_{45}: $\leftarrow func(f_1, B, A)$
C_{46}: $\leftarrow func(f_1, B, C)$
C_{47}: $\leftarrow func(f_0, C)$
C_{48}: $killer(x) \leftarrow func(f_0, x)$
C_{49}: $\leftarrow func(f_1, B, x), func(f_0, x)$
C_{50}: $\leftarrow func(f_1, x, A), func(f_0, x)$
C_{51}: $\leftarrow neq(x, A), neq(x, B), neq(x, C), func(f_0, x)$
C_{52}: $\leftarrow neq(x, A), neq(x, B), neq(x, C), func(f_1, C, x)$
C_{53}: $\leftarrow func(f_1, A, A)$
C_{54}: $\leftarrow func(f_1, A, C)$
C_{55}: $\leftarrow neq(x, B), func(f_1, A, x)$

FIGURE 3. Clauses without further reduction by the unfolding-based rules

3. Extension by a New Space and New ET Rules.

Further transformation of the clauses in Figure 3 by unfolding-based rules is impossible. We overcome the difficulty by inventing a method of reduction of *func*-atoms, by which we obtain a solution for the Agatha puzzle.

3.1. The main objective of the paper.

Except for the *killer*-atom in C_{48}, the clauses in Figure 3 contain only two kinds of atoms, i.e., *neq*-atoms and *func*-atoms. We cannot apply unfolding to the clauses in Figure 3 since there is no *killer*-atom in the bodies of these clauses. We cannot apply the definite-clause-removal rule to remove the definition of *killer* (C_{48}) since the answer to the problem is concerned with *killer*-atoms. We cannot transform these clauses further by the ET rules that have been developed so far.

The main objective of this paper is to extend the existing theory by

1) introduction of constraints attached to function variables, and
2) invention of new ET rules for dealing with *func*-atoms and constraints.

This extension makes it possible to solve a larger class of problems, including the Agatha puzzle.

3.2. Transformation with function variables and constraints.

We will introduce constraints attached to function variables in Section 4 and invent three ET rules (called CFV rules), i.e., (i) positive function-variable restriction, (ii) negative function-variable

restriction, and (iii) *func*-atom evaluation. After the completion of this theory extension, we can devise a complete solution to the Agatha puzzle.

The function variables occurring in Figure 3 can be freely instantiated without any constraint, which is represented explicitly in the first two lines of the figure (i.e., f_0: $\{\}$ and f_1: $\{\}$). Further transformation is done with the interaction between clauses and constraints as follows.

1) Refer to Figure 3. By positive function-variable restriction, C_{51} changes f_0 to f_0: $\{\langle[\,], \{A, B, C\}\rangle\}$, which means $f_0 \in \{A, B, C\}$. By positive function-variable restriction, f_1 is changed sequentially as follows:
   - f_1: $\{\langle[A], \{B\}\rangle\}$ by C_{55},
   - f_1: $\{\langle[A], \{B\}\rangle, \langle[B], \{B\}\rangle\}$ by C_{44}, and
   - f_1: $\{\langle[A], \{B\}\rangle, \langle[B], \{B\}\rangle, \langle[C], \{A, B, C\}\rangle\}$ by C_{52}.

   The constraint on f_1 in the last line above means $f_1(A) \in \{B\}$ and $f_1(B) \in \{B\}$ and $f_1(C) \in \{A, B, C\}$. The clauses C_{44}, C_{51}, C_{52}, and C_{55} are then removed. The clauses in Figure 4 are obtained.

$$
\begin{array}{ll}
f_0\text{: } \{\langle[\,], \{A, B, C\}\rangle\} & \\
f_1\text{: } \{\langle[B], \{B\}\rangle, \langle[C], \{A, B, C\}\rangle, \langle[A], \{B\}\rangle\} & \\
C_{56}\text{:} & \leftarrow \mathit{func}(f_1, B, A) \\
C_{57}\text{:} & \leftarrow \mathit{func}(f_1, B, C) \\
C_{58}\text{:} & \leftarrow \mathit{func}(f_0, C) \\
C_{59}\text{:} & \mathit{killer}(x) \leftarrow \mathit{func}(f_0, x) \\
C_{60}\text{:} & \leftarrow \mathit{func}(f_1, B, x), \mathit{func}(f_0, x) \\
C_{61}\text{:} & \leftarrow \mathit{func}(f_1, x, A), \mathit{func}(f_0, x) \\
C_{62}\text{:} & \leftarrow \mathit{func}(f_1, A, A) \\
C_{63}\text{:} & \leftarrow \mathit{func}(f_1, A, C)
\end{array}
$$

FIGURE 4. Clauses obtained by positive-function-variable restriction

2) C_{56}, C_{57}, C_{62}, and C_{63} are removed by *func*-atom evaluation.
3) The clause C_{58} changes f_0 into f_0: $\{\langle[\,], \{A, B\}\rangle\}$ by negative function-variable restriction, and it is removed. At this stage, the remaining clauses are C_{59}, C_{60}, and C_{61}.
4) Since $\mathit{func}(f_1, B, B)$ is true, the clause C_{60} is transformed into
   C_{64}: $\leftarrow \mathit{func}(f_0, B)$
5) The clause C_{64} changes f_0 further into f_0: $\{\langle[\,], \{A\}\rangle\}$ by negative function-variable restriction, and the clause is removed.
6) Since $\mathit{func}(f_0, A)$ is true and $\mathit{func}(f_1, A, A)$ is false, C_{61} is removed by *func*-atom evaluation.
7) Since $\mathit{func}(f_0, A)$ is true, C_{59} is transformed into
   C_{65}: $\mathit{killer}(A) \leftarrow$

The only remaining clause is C_{65}, from which the answer can be readily obtained, i.e., Agatha is the only killer.

4. **Constraints Attached to Function Variables.** In order to provide a theoretical foundation for representation and computation based on constraints attached to function variables and the CFV rules, we introduce an extended clause space with function variables, each of which may have non-empty constraints.

4.1. **User-defined atoms, built-in 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 *built-in atom* or a *constraint 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 $\mathcal{A}_{\mathrm{u}}$ be the set of all user-defined atoms and $\mathcal{A}_{\mathrm{c}}$ the set of all built-in atoms.

A *func-atom* [9] 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.

4.2. **Constraints attached to function variables.** To enrich the expressive power of clauses and support more flexible transformation, we introduce attachment of constraints to function variables.

Let *FVar* be the set of all function variables and *FCon* the set of all function constants. Let $\mathcal{G}_{\mathrm{t}}$ denote the set of all ground usual terms. Each n-ary function constant is associated with a mapping from $\mathcal{G}_{\mathrm{t}}^n$ to $\mathcal{G}_{\mathrm{t}}$. There are two types of variables: usual variables and function variables. A function variable can be instantiated into any function constant, but not into a usual term.

A *constraint set* is a set of pairs each of which takes the form $\langle seq, G\rangle$, where *seq* is a sequence of terms in $\mathcal{G}_{\mathrm{t}}$ and G is a non-empty set of terms in $\mathcal{G}_{\mathrm{t}}$. A constraint set S is attached to each occurrence of a function variable $f \in FVar$ and such an occurrence is denoted by the pair $\langle f, S\rangle$. When the attached constraint set S is empty, f is called a *pure* function variable and is often denoted simply by f itself. The set S in $\langle f, S\rangle$ specifies constraints for instantiation of f, requiring that if $\langle [t_1, \ldots, t_n], G\rangle \in S$ and σ is a substitution for function variables such that $f\sigma$ is a function constant f_c, then $f_c(t_1, \ldots, t_n) \in G$.

4.3. **Extended clauses.** An *extended clause* C on $\mathcal{A}_{\mathrm{u}} \cup \mathcal{A}_{\mathrm{c}}$ 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 in $\mathcal{A}_{\mathrm{u}}$ or a built-in atom in $\mathcal{A}_{\mathrm{c}}$, 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. C is said to be *pure* iff each function variable occurring in C is pure.

Let $\mathrm{ECLS}_{\mathrm{FC}}$ be the set of all extended clauses, possibly containing function variables with constraints. Recall that $\mathrm{ECLS}_{\mathrm{F}}$ is the set of all extended clauses with only pure function variables. Then $\mathrm{ECLS}_{\mathrm{FC}}$ is a superset of $\mathrm{ECLS}_{\mathrm{F}}$, i.e.,

$$\mathrm{ECLS}_{\mathrm{FC}} \supset \mathrm{ECLS}_{\mathrm{F}}.$$

Let $Cs \subseteq \mathrm{ECLS}_{\mathrm{FC}}$. Cs is said to be *pure* iff all clauses in Cs are pure. Cs is said to be *normal* iff for any function variable $f \in FVar$, if $\langle f, S_1\rangle$ and $\langle f, S_2\rangle$ are occurrences of f in Cs, then $S_1 = S_2$.

5. **ET Rules for Processing *func*-atoms.** We define three CFV rules, i.e., ET rules for dealing with *func*-atoms in the space with constraints being attached to function variables, and prove the correctness of these rules.

5.1. **Meanings of constraints attached to function variables.** Assume that Cs is a set of clauses in $\mathrm{ECLS}_{\mathrm{FC}}$. All function variables in Cs are existentially quantified

at the top of Cs. Suppose that there is a ground *func*-atom $func(f, s_1, \ldots, s_m, t)$ in the right-hand side of a clause C in Cs, where $s_1, \ldots, s_m$ and t are ground terms in $\mathcal{G}_{\mathrm{t}}$, and the constraint set S attached to f contains a pair $\langle [s_1, \ldots, s_m], G\rangle$. The set G imposes a constraint on possible values of $f(s_1, \ldots, s_m)$, i.e., there is a substitution σ for function variables such that $(f\sigma)(s_1, \ldots, s_m) \in G$.

If G is the empty set, the condition $\exists\sigma$: $(f\sigma)(s_1, \ldots, s_m) \in G$ gives a contradiction. It follows that Cs has no model. Hence an MI problem $\langle Cs, \varphi\rangle$ can be equivalently transformed into $\langle\{\leftarrow\}, \varphi\rangle$, where $\{\leftarrow\}$ is a singleton set of the empty clause $(\leftarrow)$, which is false.

5.2. **Positive function-variable restriction.** Let $\langle Cs, \varphi\rangle$ be an MI problem on EC-$\mathrm{LS}_{\mathrm{FC}}$. Assume that Cs is normal and contains an extended clause C such that

$$C = (eq(x, t_1), \ldots, eq(x, t_n) \leftarrow func(f, s_1, \ldots, s_m, x)),$$

where x is a usual variable, $t_1, \ldots, t_n, s_1, \ldots, s_m$ are ground terms, and f is an m-ary function variable. The clause C means that if $f(s_1, \ldots, s_m) = x$, then x is one of $t_1, \ldots, t_n$. Hence, C gives a positive restriction for f. More precisely, $\langle Cs, \varphi\rangle$ can be equivalently transformed into $\langle Cs', \varphi\rangle$, where Cs' is obtained from Cs as follows: Assume that S is the constraint set attached to f.

1) If there is no pair of the form $\langle [s_1, \ldots, s_m], G\rangle$ in S, then change S into
$$S \cup \{\langle [s_1, \ldots, s_m], \{t_1, \ldots, t_n\}\rangle\},$$
and let $Cs' = Cs - \{C\}$.
2) If $\langle [s_1, \ldots, s_m], G\rangle \in S$ and $G \cap \{t_1, \ldots, t_n\} \neq \emptyset$, then change S into
$$(S - \{\langle [s_1, \ldots, s_m], G\rangle\}) \cup \{\langle [s_1, \ldots, s_m], G \cap \{t_1, \ldots, t_n\}\rangle\},$$
and let $Cs' = Cs - \{C\}$.
3) If $\langle [s_1, \ldots, s_m], G\rangle \in S$ and $G \cap \{t_1, \ldots, t_n\} = \emptyset$, then let $Cs' = \{\leftarrow\}$.

The correctness of this transformation is shown as follows: The constraint imposed by the clause C means that $f(s_1, \ldots, s_m) \in \{t_1, \ldots, t_n\}$. Hence, in Case 1, non-existence of $\langle [s_1, \ldots, s_m], G\rangle$ means that $f(s_1, \ldots, s_m)$ may take any arbitrary value. Therefore, the pair $\langle [s_1, \ldots, s_m], \{t_1, \ldots, t_n\}\rangle$ should be added to S. In Case 2, elements outside $\{t_1, \ldots, t_n\}$ are removed from G. In Case 3, the constraint for f is inconsistent. From the explanation in Section 5.1, Cs' can thus be $\{\leftarrow\}$.

5.3. **Negative function-variable restriction.** Let $\langle Cs, \varphi\rangle$ be an MI problem on EC-$\mathrm{LS}_{\mathrm{FC}}$. Assume that Cs is normal and contains an extended clause C such that

$$C = (\leftarrow func(f, s_1, \ldots, s_m, s)),$$

where $s_1, \ldots, s_m$ and s are ground terms and f is a function variable to which a constraint set containing $\langle [s_1, \ldots, s_m], G\rangle$, for some $G \subseteq \mathcal{G}_{\mathrm{t}}$, is attached. The clause C means that $f(s_1, \ldots, s_m) \neq s$. Hence, C gives a negative restriction for f. More precisely, $\langle Cs, \varphi\rangle$ can be equivalently transformed into $\langle Cs', \varphi\rangle$, where Cs' is obtained from Cs as follows: Assume that S is the constraint set attached to f.

1) If $\langle [s_1, \ldots, s_m], G\rangle \in S$ and $G = \{s\}$, then let $Cs' = \{\leftarrow\}$.
2) If $\langle [s_1, \ldots, s_m], G\rangle \in S$ and $G \neq \{s\}$, then change S into
$$(S - \{\langle [s_1, \ldots, s_m], G\rangle\}) \cup \{\langle [s_1, \ldots, s_m], G - \{s\}\rangle\},$$
and let $Cs' = Cs - \{C\}$.

The correctness of this transformation is shown as follows: Assume that $\langle [s_1, \ldots, s_m], G\rangle \in S$. The negative clause C imposes a constraint $f(s_1, \ldots, s_m) \neq s$. If $G = \{s\}$, the constraint means that $\mathit{func}(f, s_1, \ldots, s_m) = s$ is true, which contradicts the meaning of the clause C. Hence, there is no model of Cs, which results in $Cs' = \{\leftarrow\}$. If $G \neq \{s\}$, the possibility of s is denied by C. Hence, G becomes $G - \{s\}$ and $Cs' = Cs - \{C\}$.

5.4. ***func*-atom evaluation.** Let $\langle Cs, \varphi\rangle$ be an MI problem on $\mathrm{ECLS_{FC}}$. Assume that Cs is normal and contains an extended clause C such that C contains a *func*-atom $\mathit{func}(f, s_1, \ldots, s_m, t)$ in its right-hand side. Assume that S is the constraint set attached to f. When we know the logical value of $\mathit{func}(f, s_1, \ldots, s_m, t)$ from the constraint attached to f, we apply *func*-atom evaluation rules, which produce the following equivalent transformation:

1) If t is a ground term and the constraint set S contains $\langle [s_1, \ldots, s_m], \{t\}\rangle$, then $\mathit{func}(f, s_1, \ldots, s_m, t)$ can be removed from C.
2) If t is a usual variable v and the constraint set S contains $\langle [s_1, \ldots, s_m], \{s\}\rangle$, then C can be specialized by instantiating v into s and the true *func*-atom $\mathit{func}(f, s_1, \ldots, s_m, s)$ in the resulting clause can be removed.
3) If t is a ground term and the constraint set S contains $\langle [s_1, \ldots, s_m], G\rangle$ such that $t \notin G$, then C can be removed from Cs.

The correctness of this transformation is shown as follows: Assume that the constraint set S attached to f contains $\langle [s_1, \ldots, s_m], G\rangle$. The above transformation is correct since $\mathit{func}(f, s_1, \ldots, s_m, t)$ is true in the first two cases, and it is false in the third case.

6. **Experiments and Comparison.** We make experiments to solve Agatha proof problems and Agatha QA problems, and compare them to the results obtained by resolution with *conventional Skolemization-based decomposition* (CSD) [13].

6.1. **Experiments.** We have constructed an experimental MI-problem solver by directly implementing the theory, and have tried to apply it to solving many proof and QA problems, many of which are not large but impossible to be solved correctly if we use existing theories and techniques developed in the logic programming and the semantic-web communities.

The Agatha puzzle, which is formalized as a QA problem (cf. Section 2.2), was solved by 102 applications of ET rules, consisting of unfolding and definite-clause removal, constraint solving for *neq*, the ET rules developed in Section 5, and other ET rules devised so far in our previous works [8, 10, 11, 12].

The following three Agatha proof problems were also successfully solved by our MI-problem solver: (i) "did Agatha kill herself?", (ii) "didn't the butler kill Agatha?", and (iii) "didn't Charles kill Agatha?" These three proof problems are represented as the conjunctions of the first-order formulas in Figure 1 and $\neg \mathit{killer}(A)$, $\mathit{killer}(B)$, and $\mathit{killer}(C)$, respectively. The empty clause ($\leftarrow$) was produced from each of them, thereby proving that Agatha is the killer, not the other ones.

These solutions are summarized as

$$\mathrm{FOL_c} \longrightarrow \mathrm{ECLS_F} \longrightarrow \mathrm{ECLS_{FC}} \longrightarrow \langle \text{correct solution}\rangle,$$

where the first-order formula representing a problem in $\mathrm{FOL_c}$ is transformed by the *meaning-preserving decomposition* (MPD) algorithm [13] into a set of extended clauses in $\mathrm{ECLS_F}$.

6.2. **Comparison to the resolution-based methods.** The conventional resolution-based proof method is constructed in the spaces FOL and CLS, where a first-order formula K in FOL is transformed by the CSD algorithm [13] into a set of clauses in CLS, which is subsequently transformed by resolution for finding a solution, as outlined by

$$\text{FOL} \longrightarrow \text{CLS} \longrightarrow \langle \text{solution} \rangle.$$

Basically, we need the spaces FOL_c and CLS_c for solving the Agatha puzzle, which is represented by using the built-in predicates *eq* and *neq*. The conventional solution method may be used for solving the Agatha proof problem, which is outlined by

$$\text{FOL}_\text{c} \longrightarrow \text{CLS}_\text{c} \longrightarrow \langle \text{solution} \rangle,$$

where the CSD algorithm and resolution are applied. However, the Agatha puzzle is incorrectly transformed by the CSD algorithm, and therefore further application of the resolution rule becomes useless from the viewpoint of correctness.

We explain the above incompleteness of the conventional logic by using the Agatha puzzle. When we transform the background knowledge K represented as the conjunction of the first-order formulas in Figure 1 by using CSD, we obtain the clause set $Cs' = \{C'_1, C'_2, C_3, \ldots, C_{12}, C'_{13}, C'_{14}, C_{15}\}$, where C_3-C_{12} and C_{15} are given in Figure 2 and C'_1, C'_2, C'_{13}, and C'_{14} are given in Figure 5. This clause set is inconsistent while the original background knowledge K is consistent. Since each of $K \wedge \neg killer(A)$, $K \wedge killer(B)$, and $K \wedge killer(C)$ includes K, we also have similar contradictions from each of them. It follows that the application of the CSD algorithm to the Agatha puzzle does not preserve satisfiability. Hence, none of the three Agatha proof problems mentioned above can be solved correctly by the conventional methods.

C'_1: $live(f_0, D) \leftarrow$ $\qquad$ C'_{13}: $\leftarrow hate(x, f_1(x)), live(x, D)$
C'_2: $kill(f_0, A) \leftarrow$ $\qquad$ C'_{14}: $live(f_1(x), D) \leftarrow live(x, D)$

FIGURE 5. Clauses obtained by CSD

The difficulty was avoided by using ECLS_F and MPD in place of CLS_c and CSD:

$$\text{FOL}_\text{c} \longrightarrow \text{ECLS}_\text{F} \longrightarrow \text{ECLS}_\text{FC} \longrightarrow \langle \text{correct solution} \rangle.$$

The Agatha puzzle was successfully solved in the ET-based theory as shown in this paper using the spaces ECLS_F and ECLS_FC, where the ET rules dealing with $func$-atoms and constraints attached to function variables play important roles in place of resolution and normal clauses.

7. **Conclusions.** MI problems constitute one of the largest classes of logical problems and are of fundamental importance. We solve MI problems by using ET rules on some formula space. A typical formula space is the set of all first-order formulas, as well as that of all usual clauses. Under the LPSF theory, we can increase the solvability of MI problems by extending the formula space and devising additional ET rules.

The CSD algorithm gives a method of formula transformation from FOL_c into CLS_c, with no extension of the underlying computation space. The employment of CSD, however, is a major hindrance to development of a general solution for proof/QA problems on FOL_c since it does not generally preserve satisfiability nor logical meanings of logical formulas, and thus does not preserve the answers to proof/QA problems. To overcome the limitation, the CLS_c space is extended into the ECLS_F space and each formula in FOL_c is equivalently converted using the MPD algorithm into a clause set in ECLS_F.

This paper has further extended the representation and computation power of MI problems by introduction of constraints attached to function variables. A new space extension

is from ECLS_{F} to ECLS_{FC}. This extension has strengthened the power of solving a large class of MI (and thus proof and QA) problems. For dealing with constraints attached to function variables, the following new ET rules have been invented:

- positive function-variable restriction,
- negative function-variable restriction, and
- *func*-atom evaluation.

So far, even in the ECLS_{F} space, the Agatha puzzle could not be solved correctly from the full first-order formalization. The space extension approach in this paper has successfully solved the Agatha puzzle fully for the first time. This is one of the most important methods towards strengthening the power of representation and computation in the logical computation theory that has been developed based on the ET approach.

Acknowledgments. 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.

REFERENCES

[1] H. Aït-Kaci and R. Nasr, LOGIN: A logic programming language with built-in inheritance, *Journal of Logic Programming*, vol.3, pp.185-215, 1986.
[2] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.
[3] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[4] D. W. Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland Publishing, 1978.
[5] 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.
[6] K. Akama and E. Nantajeewarawat, Formalization of logical problems as model-intersection problems on an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.17, no.4, pp.1103-1117, 2021.
[7] 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.
[8] K. Akama and E. Nantajeewarawat, Unfolding existentially quantified sets of extended clauses, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Porto, Portugal, vol.2, pp.96-103, 2016.
[9] 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.
[10] 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.
[11] 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*, Porto, Portugal, vol.2, pp.52-63, 2016.
[12] K. Akama and E. Nantajeewarawat, Unfold/unblock computation control with side-change transformation, *International Journal of Innovative Computing, Information and Control*, vol.19, no.5, pp.1531-1542, 2023.
[13] K. Akama and E. Nantajeewarawat, Skolemization that preserves logical meanings, *International Journal of Innovative Computing, Information and Control*, vol.17, no.1, pp.1-13, 2021.

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.