International Journal of Innovative
Computing, Information and Control
Volume **18**, Number **5**, October **2022**

ICIC International ©2022 ISSN 1349-4198
pp. **1559–1570**

A FOUNDATION OF LOGICAL PROBLEM SOLVING

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 March 2022; revised July 2022

Abstract. *We propose a Logical Problem Solving Framework (LPSF), an axiomatic structure for generating methods of logical problem solving. Input parameters of LPSF consist of 1) a canonical logical structure, 2) a set of equivalent transformation rules (ET rules), 3) a control, and 4) a set of answer mappings. Given these input parameters, LPSF provides a logical problem solver, which receives an input problem, sets an initial state by using a formula on the logical structure, makes each computation step by application of an ET rule selected by the control, and if the resulting computation path reaches the domain of an answer mapping, it outputs an answer, which is guaranteed to be correct. LPSF is useful for design of knowledge representation systems and construction of logical problem solving methods.*

Keywords: Proof problem, Query-answering problem, Skolemization, Equivalent transformation rule, Logical structure, First-order logic

1. **Introduction.** A proof problem with respect to a first-order formula A and a first-order formula B is a "yes/no" problem; it is concerned with checking whether $A \models B$, i.e., whether A entails B. Proof problems on first-order formulas historically constitute the most important problem class in logical problem solving. Resolution provides us with a refutation proof procedure for logical formulas [1]. To solve a proof problem with respect to A and B, we (i) first convert $A \wedge \neg B$ into a clause set Cs and (ii) try to transform Cs into a clause set Cs' that contains an empty clause ($\leftarrow$). The correctness of the resolution proof procedure is given by the soundness and completeness theorem. This does not, however, guarantee the correctness of computation since the first-phase conversion does not preserve satisfiability in the presence of clauses containing built-in atoms [2]. The conventional resolution-based theory fails to estabilish a correct proof method for full first-order formulas [3].

A query-answering (QA) problem is an "all-answers finding" problem to satisfy a given logical consequence relation. SLD resolution has been proposed for solving QA problems on definite clauses [4]. The correctness of the QA solution procedure is given by the soundness and completeness theorem for SLD resolution. Due to the completeness theorem for SLD resolution, it is believed that SLD resolution can be successfully applied to all QA problems. However, we proved that a QA problem, called a pal-pal QA problem, cannot

DOI: 10.24507/ijicic.18.05.1559

be solved by SLD resolution [5]. The completeness condition is too weak for ensuring the correctness of solutions for all QA problems on definite clauses. The SLD resolution for QA problems fails to estabilish a correct QA solution method for all definite clauses. The conventional resolution-based theory has failed to estabilish a correct QA solution method for full first-order formulas.

To overcome these failures of solution methods in computational logic, we propose a Logical Problem Solving Framework (LPSF), which is an axiomatic structure for generating methods of logical problem solving. LPSF extends the most basic concepts of logic and computation as follows.

- LPSF utilizes a general concept of logical structure, which is a superconcept that includes many logics such as first-order logic and propositional logic as instances.
- LPSF regards model-intersection (MI) problems as the main class of logical problems, which is a superset of the class of proof problems and that of query-answering (QA) problems.
- LPSF takes the class of equivalent transformation rules, which is a superset of the class of inference rules.

A general principle for solving MI problems on formulas is equivalent transformation (ET), where problems are solved by repeated problem simplification using ET rules. Efficiency of computation is basically determined by

- a set R of ET rules used for the computation, and
- selection of an ET rule in R at each computation step.

ET rules and computation control may give an ET sequence that reaches a final or a non-final problem description in finite steps or may produce an infinite sequence without giving any answer to the original problem.

We can invent a logical problem solving method using LPSF by designing four parameters: 1) a canonical logical structure $\mathcal{L}$, 2) a set R of ET rules, 3) a control *ctrl*, and 4) a set A of answer mappings. LPSF with these input parameters works as a problem solver for MI problems as follows.

- A problem q formalized on $\mathcal{L}$ is received as input.
- The input problem q is taken as the initial state P_0.
- P_0 is successively transformed by using ET rules in R, with rule selection being determined by *ctrl*.
- Computation is a sequence of problems $P_0, P_1, P_2, \ldots$ on $\mathcal{L}$.
- If the computation $P_0, P_1, P_2, \ldots$ reaches the domain of an answer mapping ans in A at P_n, then LPSF outputs $ans(P_n)$ as an answer.

The strength of the LPSF theory comes from the fact that

- it is constructed on mathematically general and precise concepts, and
- a logical problem solving method generated by LPSF is guaranteed to be correct for any combination of parameters.

The LPSF theory proposed in this paper provides a general foundation of logical problem solving methods. The definition of a logical structure used in LPSF covers a wide range of logics, and the conventional first-order logic can be regarded as one instance of it. MI problems considered in LPSF are a large problem class, which basically covers all proof problems and QA problems considered so far. The concept of computation in LPSF is equivalent transformation, which provides a wider class of computation and covers computation by logical inference as its special case.

LPSF can be used for comparison and evaluation of conventional logical problem solving methods. A resolution-based method can be regarded as a solver instance generated

by LPSF. Basically, it uses clauses of first-order logic as a logical structure and the resolution and factoring inference rules as ET rules. By adjusting rule application control, many logical solution methods have been invented [6, 7, 8, 9]. Let FOL be the set of all first-order formulas, and $\mathrm{FOL_c}$ the set of all first-order formulas with built-in constraint atoms. Conventional resolution methods are basically constructed on FOL, with no built-in constraint atoms. The solution method for pure Prolog problems on usual clauses can be regarded as one specific logical problem solving method (a solver) that can be generated by LPSF. They take a specific logical structure (definite clauses in first-order logic), a subclass of MI problems (a subclass of proof problems and/or QA problems), and a subclass of ET rules (unfolding-based rules). Some logical problems cannot be solved correctly by conventional methods.

LPSF is useful for creation of new solution methods. Since LPSF has a parameter for logical structures, we can take larger problem classes and try to create a new logical structure for inventing solution methods that can solve them correctly. Based on the new logical structure, we can accumulate rules and adjust controls to maximize the solvability. For instance, a new logic, called KR-Logic, was created as an extension of LP-Logic. For proof and QA problems, KR-Logic together with many newly invented ET rules was shown to improve solvability, compared to conventional solvers on LP-Logic with ET rules such as resolution and unfolding [10, 11, 12, 13].

The rest of this paper is organized as follows. Section 2 defines logical structures and model-intersection problems on a logical structure, by which we can formalize a large class of logical problems. Section 3 defines computation as an ET sequence, based on the concepts of answer mappings, rewriting rules, and controls. Section 4 introduces Logical Problem Solving Framework (LPSF), by which a partial mapping to associate problems with their answers is obtained. Section 5 proves two correctness theorems, i.e., correctness of computation paths, and that of solution methods. Section 6 defines solvability of an LPSF solver, and explains the improvement of the solvability of logical problem solving. Section 7 concludes the paper.

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 $PartialMap(A, B)$ denotes the set of all partial mappings from A to B. For any partial mapping f from A to B, $dom(f)$ denotes the domain of f, i.e., $dom(f) = \{a|(a \in A)\ \&\ (f(a)\ \text{is defined})\}$.

2. Logical Structures and Formalization.

We define the concept of logical structure axiomatically. We also introduce the concept of model-intersection problems on a logical structure, which can be used to formalize a large class of logical problems.

2.1. Canonical logical structures and basic related concepts.

Logic is the primary basis of logical problem solving. Abstract structure of logic has been formalized as an axiomatic definition as follows [14].

Definition 2.1. *A logical structure $\mathcal{L}$ is a triple $\langle \mathcal{K}, \mathcal{I}, \nu \rangle$, where*

1) $\mathcal{K}$ and $\mathcal{I}$ are sets,

2) ν: $\mathcal{K} \to Map(\mathcal{I}, \{true, false\})$.

An element of $\mathcal{K}$ is called a description and that of $\mathcal{I}$ is called an interpretation.

A typical description is a logical formula in the first-order syntax. A set of clauses on first-order logic is also a typical example of a description. The sets $\mathcal{K}$ and $\mathcal{I}$ in the above definition are arbitrary sets. They are only required to satisfy the relations described in the definition. Given $k \in \mathcal{K}$ and $I \in \mathcal{I}$, the mapping ν determines truth or falsity,

which shows whether k is true or false with respect to I. Despite its simplicity, the notion of a logical structure provides a sufficient structure for defining the concepts of logical equivalence, models, satisfiability, and logical consequence, which are given below.

Definition 2.2. *Let $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$ be a logical structure. Two descriptions $k_1, k_2 \in \mathcal{K}$ are logically equivalent iff $\nu(k_1) = \nu(k_2)$. An interpretation $I \in \mathcal{I}$ is a model of a description $k \in \mathcal{K}$ iff $\nu(k)(I) = true$. The set of all models of a description $k \in \mathcal{K}$ is denoted by $Models(k)$. A description $k \in \mathcal{K}$ is satisfiable iff there exists a model of k. A description $k_1 \in \mathcal{K}$ entails a description $k_2 \in \mathcal{K}$ (i.e., k_2 is a logical consequence of k_1), denoted by $k_1 \models k_2$, iff every model of k_1 is a model of k_2.*

Each logical structure defines one logic. Two logical structures with the same set of descriptions but different sets of interpretations are considered to be different. In this sense, first-order logic with standard semantics and first-order logic with Herbrand interpretations are different logical structures.

Let $\mathcal{G}_{\mathrm{u}}$ be a set, the elements of which are regarded as ground user-defined atoms. A logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$ is a canonical logical structure iff $\mathcal{I} = pow(\mathcal{G}_{\mathrm{u}})$. We assume that all logical structures considered hereinafter are canonical logical structures. Strictly speaking, first-order logic with standard semantics and first-order logic with Herbrand interpretations are not canonical logical structures. However, by modifying their interpretations, they can be regarded as canonical logical structures that are equivalent to the original ones.

2.2. **Model-intersection problems.** Let $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$ be a canonical logical structure and W be a set. A *model-intersection problem* (for short, *MI problem*) on $\langle \mathcal{K}, W \rangle$ is a pair $\langle Cs, \varphi \rangle$, where Cs is a description in $\mathcal{K}$ and φ is a mapping from $pow(\mathcal{G}_{\mathrm{u}})$ to W. We use Cs rather than k as an element of $\mathcal{K}$ since typical descriptions in logical structures considered in this paper are sets of clausal formulas. The mapping φ is called an *extraction mapping*. The answer to this problem, denoted by $ans_{\mathrm{MI}}(Cs, \varphi)$, is defined by

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

where $\bigcap Models(Cs)$ is the intersection of all models of Cs. Intuitively, this MI problem is to find some element $w \in W$ that is determined by the intersection of all models of Cs. The set W is arbitrarily chosen as the co-domain of φ. Note that when $Models(Cs)$ is the empty set, $\bigcap Models(Cs) = \mathcal{G}_{\mathrm{u}}$. The set of all MI problems on $\langle \mathcal{K}, W \rangle$ is denoted by $MI_{PROB}(\mathcal{K}, W)$, i.e.,

$$MI_{PROB}(\mathcal{K}, W) = \mathcal{K} \times Map\,(pow(\mathcal{G}_{\mathrm{u}}), W).$$

An MI problem on $\langle \mathcal{K}, W \rangle$ for some W is called an MI problem on the logical structure $\mathcal{L}$. The set of all MI problems on $\mathcal{L}$ is denoted by $MI_{PROB}(\mathcal{L})$, i.e., $MI_{PROB}(\mathcal{L}) \bigcup \{MI_{PROB}(\mathcal{K}, W) | W \text{ is a set}\}$.

2.3. **Examples of model-intersection problems.** Many problems can be considered as MI problems on some logical structures [11, 12, 15, 16]. We explain in this section the generality of the class of MI problems by showing examples (two for Example 2.1 and two for Example 2.2).

Example 2.1. *Consider the Oedipus puzzle described in* [17]. *Oedipus killed his father, married his mother Iokaste, and had children with her, among them Polyneikes. Polyneikes also had children, among them Thersandros, who is not a patricide.*

1) *The Oedipus proof problem is to answer whether Iokaste has a patricide child who has a non-patricide child.*

2) The Oedipus QA problem is to find all persons who have a patricide child who has a non-patricide child.

Assume that (i) "oe", "io", "po" and "th" stand, respectively, for Oedipus, Iokaste, Polyneikes and Thersandros, (ii) for any terms t_1 and t_2, $isCh(t_1, t_2)$ denotes "t_1 is a child of t_2", and (iii) for any term t, $pat(t)$ denotes "t is a patricide" and $prob(t)$ denotes "t is an answer to this puzzle".

Let CLS_{B} be the set consisting of all usual clauses possibly with built-in constraint atoms. This set CLS_{B} forms a canonical logical structure.

Let $Cs_1 \subseteq CLS_{\mathrm{B}}$ consist of the following seven clauses:

$$\begin{array}{ll} isCh(oe, io) \leftarrow & isCh(po, io) \leftarrow \\ isCh(po, oe) \leftarrow & isCh(th, po) \leftarrow \\ pat(oe) \leftarrow & \leftarrow pat(th) \\ prob(x), pat(y) \leftarrow isCh(z, x), pat(z), isCh(y, z) & \end{array}$$

The Oedipus proof problem and the Oedipus QA problem are then formalized as follows.

1) Let φ_1 be defined by $\varphi_1(G) =$ "yes" if $G = \mathcal{G}_{\mathrm{u}}$; otherwise $\varphi_1(G) =$ "no", for any $G \subseteq \mathcal{G}_u$. The Oedipus proof problem is formalized as the MI problem $\langle Cs_1 \cup \{\leftarrow prob(io)\}, \varphi_1 \rangle$.
2) Let φ_2 be defined by $\varphi_2(G) = \{x | prob(x) \in G\}$ for any $G \subseteq \mathcal{G}_{\mathrm{u}}$. The Oedipus QA problem is formalized as the MI problem $\langle Cs_1, \varphi_2 \rangle$.

Example 2.2. *Consider the tax-cut problem discussed in* [18]*. The background knowledge consists of the following statements. (i) Any person who has two children or more can get discounted tax. (ii) Men and women are not the same. (iii) It is false that a person is not the same as himself/herself. (iv) A person's mother is always a woman. (v) Peter has a child, who is someone's mother. (vi) Peter has a child named Paul. (vii) Paul is a man.*

1) The tax-cut proof problem is to answer whether Peter can have discounted tax.
2) The tax-cut QA problem is to find all persons who can have discounted tax.

Let $Cs_2 \subseteq CLS_{\mathrm{B}}$ consist of the following eight clauses:

$$\begin{array}{l} TaxCut(x) \leftarrow hasChild(x, y), hasChild(x, z), notSame(y, z) \\ notSame(x, y) \leftarrow Man(x), Woman(y) \\ \leftarrow notSame(x, x) \\ Woman(x) \leftarrow motherOf(x, y) \\ hasChild(Peter, f_1) \leftarrow \\ motherOf(f_1, f_2) \leftarrow \\ hasChild(Peter, Paul) \leftarrow \\ Man(Paul) \leftarrow \end{array}$$

The fifth and the sixth clauses together represent the fifth statement (i.e., "Peter has a child, who is someone's mother"), where f_1 and f_2 are 0-ary function symbols. The tax-cut proof problem and the tax-cut QA problem are then formalized as follows.

1) Referring to φ_1 of Example 2.1, let $\varphi_3 = \varphi_1$. The tax-cut proof problem is formalized as the MI problem $\langle Cs_2 \cup \{\leftarrow TaxCut(Peter)\}, \varphi_3 \rangle$.
2) Let φ_4 be defined by $\varphi_4(G) = \{x | TaxCut(x) \in G\}$ for any $G \subseteq \mathcal{G}_{\mathrm{u}}$. The tax-cut QA problem is formalized as the MI problem $\langle Cs_2, \varphi_4 \rangle$.

2.4. **Subclasses of model-intersection problems.** A *proof problem* is a pair $\langle E_1, E_2 \rangle$, where E_1 and E_2 are closed first-order formulas in $\mathrm{FOL_c}$, and the answer to this problem,

denoted by $ans_{\mathrm{Pr}}(E_1, E_2)$, is defined by

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

It was shown in [19] that a proof problem on $\mathrm{FOL_c}$ can be converted equivalently into an MI problem on $\mathrm{CLS_B}$.

A *query-answering problem* (*QA problem*) on $\mathrm{FOL_c}$ is a pair $\langle E, a\rangle$, where E is a closed first-order formula in $\mathrm{FOL_c}$ and a is a user-defined atom. Let $\mathcal{S}$ be the set of all substitutions for usual variables. The answer to a QA problem $\langle E, a\rangle$, denoted by $ans_{\mathrm{QA}}(E, a)$, is defined by

$$ans_{\mathrm{QA}}(E, a) = \{a\theta | (\theta \in \mathcal{S}) \ \&\ (a\theta \in \mathcal{G}_{\mathrm{u}}) \ \&\ (E \models a\theta)\}.$$

It was shown in [19] that a QA problem on $\mathrm{FOL_c}$ can be converted equivalently into an MI problem on $\mathrm{CLS_B}$.

3. Computation.

We define the concepts of answer mappings, rewriting rules, and controls, by which computation is determined as an ET sequence.

3.1. Answer mappings.

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

Definition 3.1. *Let W be a set. A partial mapping ans from $MI_{PROB}(\mathcal{K}, W)$ to W is an answer mapping iff for any $\langle Cs, \varphi\rangle \in dom(ans)$, $ans(Cs, \varphi) = ans_{\mathrm{MI}}(Cs, \varphi)$.*

An answer mapping is taken so that all answers to problems in its domain can be computed at low cost. When $P = \langle Cs, \varphi\rangle$, $ans(Cs, \varphi)$ is also denoted by $ans(P)$.

3.2. Rewriting rules and ET rules.

We consider two types of rewriting rules, rewriting rules on $MI_{PROB}(\mathcal{K}, W)$ or on $\mathcal{K}$. Let X be $MI_{PROB}(\mathcal{K}, W)$ or $\mathcal{K}$. A rewriting rule r on X is a partial mapping from X to X, i.e.,

$$r \in PartialMap(X, X).$$

Definition 3.2. *A rewriting rule r on $\mathcal{K}$ is model-preserving iff if $(Cs, Cs') \in r$, then $Models(Cs) = Models(Cs')$.*

Definition 3.3. *A rewriting rule r on $\mathcal{K}$ is model-intersection preserving iff if $(Cs, Cs') \in r$, then $\bigcap Models(Cs) = \bigcap Models(Cs')$.*

Obviously, if r is model-preserving, then r is model-intersection preserving.

Definition 3.4. *A rewriting rule r on $MI_{PROB}(\mathcal{K}, W)$ is an ET rule iff*

$$\varphi\left(\bigcap Models(Cs)\right) = \varphi'\left(\bigcap Models(Cs')\right)$$

for any $(\langle Cs, \varphi\rangle, \langle Cs', \varphi'\rangle) \in r$.

Proposition 3.1. *Assume that a rule r on $\mathcal{K}$ is model-intersection preserving. If $(Cs, Cs') \in r$, then for any $\varphi \in Map(pow(\mathcal{G}_{\mathrm{u}}), W)$,*

1) $ans_{\mathrm{MI}}(Cs, \varphi) = ans_{\mathrm{MI}}(Cs', \varphi)$, and
2) $\{(\langle Cs, \varphi\rangle, \langle Cs', \varphi\rangle) \,|\, (Cs, Cs') \in r\}$ is an ET rule on $MI_{PROB}(\mathcal{K}, W)$.

Proof: Assume that $(Cs, Cs') \in r$. Since r is model-intersection preserving, $\bigcap Models(Cs) = \bigcap Models(Cs')$. Hence, $ans_{\mathrm{MI}}(Cs, \varphi) = \varphi(\bigcap Models(Cs)) = \varphi(\bigcap Models(Cs')) = ans_{\mathrm{MI}}(Cs', \varphi)$. The second result follows immediately from the first result. □

3.3. **Control and ET sequences.** Let R be a set of ET rules. When no confusion is caused, we use *Prob* for simplicity as the set of all MI problems, i.e., $Prob = MI_{PROB}(\mathcal{K}, W)$.

3.3.1. *Control.* Given a sequence $[P_0, \ldots, P_i]$ of problems in *Prob*, a control gives an ET rule in R that is applicable to P_i. More precisely, a control *ctrl* is a partial mapping from the set of all sequences of problems in *Prob* to the ET-rule set R that satisfies the following conditions.

1) If there are ET rules in R that are applicable to P_i, $ctrl([P_0, P_1, \ldots, P_i])$ is an ET rule in R that is applicable to P_i.
2) If there is no applicable rule to P_i, $ctrl([P_0, P_1, \ldots, P_i])$ is undefined.

3.3.2. *Computation.* A partial mapping *comp* is defined by

$$comp(P_0, R, ctrl) = [P_0, P_1, P_2, \ldots, P_n]$$

if the following two conditions are satisfied.

1) For each $i = 0, 1, 2, \ldots, n-1$, $ctrl([P_0, P_1, P_2, \ldots, P_i]) = r_i \in R$ and $(P_i, P_{i+1}) \in r_i$.
2) $ctrl([P_0, P_1, P_2, \ldots, P_n])$ is undefined.

Otherwise, $comp(P_0, R, ctrl)$ is undefined.

3.3.3. *ET sequence.* Given a set A of answer mappings, let $dom(A)$ be defined as the union of all domains of the answer mappings in A, i.e., $\bigcup\{dom(ans) | ans \in A\}$. Given a set A of answer mappings and a problem P_0, the role of a control is to construct an ET sequence $[P_0, P_1, \ldots, P_n]$ that starts with P_0 and reaches a final problem $P_n \in dom(A)$.

Definition 3.5. *A sequence $[P_0, P_1, \ldots, P_n]$ of problems in Prob is an ET sequence iff $ans_{\mathrm{MI}}(P_i) = ans_{\mathrm{MI}}(P_{i+1})$ for any $i \in \{0, 1, \ldots, n-1\}$.*

ET rules produce an ET sequence.

Proposition 3.2. *Assume that R is an ET-rule set. Let P_0 be an initial problem and ctrl a control. If the value of $comp(P_0, R, ctrl)$ is defined, then it is an ET sequence.*

Proof: Assume that $comp(P_0, R, ctrl) = [P_0, P_1, P_2, \ldots, P_n]$. Then, for each $i = 0, 1, 2, \ldots, n-1$, the conjunction of $ctrl([P_0, P_1, P_2, \ldots, P_i]) = r_i \in R$ and $(P_i, P_{i+1}) \in r_i$ holds. Since R is an ET-rule set, for each $i = 1, 2, \ldots, n-1$, the condition $(P_i, P_{i+1}) \in r_i \in R$ implies $ans_{\mathrm{MI}}(P_i) = ans_{\mathrm{MI}}(P_{i+1})$. Hence, $[P_0, P_1, P_2, \ldots, P_n]$ is an ET sequence. □

4. **Logical Problem Solving Framework and an LPSF Solver.** We introduce Logical Problem Solving Framework (LPSF), which determines a solver to compute an answer to an MI problem, and defines a partial mapping from MI problems to answers.

4.1. **Logical problem solving framework.** Given a set *Cs* of clauses and an extraction mapping φ, the answer to the MI problem $\langle Cs, \varphi \rangle$ is $\varphi(\bigcap Models(Cs))$. The definition $\varphi(\bigcap Models(Cs))$ is usually not suitable for computing the answer since it may take huge cost. Instead of using this definition to compute the answer, we take a path consisting of (i) ET computation $[P_0, \ldots, P_n]$ and (ii) an answer mapping *ans* in A. LPSF finds the value of $\varphi(\bigcap Models(Cs))$ by the calculation of $ans(P_n)$.

Given an MI problem $\langle Cs, \varphi \rangle$, Logical Problem Solving Framework (LPSF) is defined more precisely as follows: Input parameters of LPSF consist of 1) a canonical logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$, 2) a set R of ET rules, 3) a control *ctrl*, and 4) a set A of answer mappings, and the output of LPSF is a solver M (also called an LPSF solver), which is denoted by

$$M = \mathrm{LPSF}(\mathcal{L}, R, ctrl, A).$$

Procedurally, LPSF works as follows.

1) Let $P_0 = \langle Cs, \varphi \rangle$.
2) Construct an ET sequence $[P_0, \ldots, P_n]$ by (i) taking P_0 as an initial problem, (ii) applying ET rules in R determined by *ctrl*, and (iii) stopping at P_n.
3) Assume that $P_n = \langle Cs_n, \varphi_n \rangle$. If the computation reaches the domain of A, i.e., $\langle Cs_n, \varphi_n \rangle \in dom(A)$, then output $ans(Cs_n, \varphi_n)$ by using an answer mapping ans in A such that $P_n \in dom(ans)$.

Let $w = ans(Cs_n, \varphi_n)$. When $\varphi(\bigcap Models(Cs)) = w$, w is called a correct answer to $\langle Cs, \varphi \rangle$. Otherwise, w is called an incorrect answer.

4.2. **A partial mapping determined by LPSF.** Assume that we have an LPSF solver $M = \text{LPSF}(\mathcal{L}, R, ctrl, A)$ by taking input parameters $\mathcal{L}$, R, *ctrl*, and A. An LPSF solver M is associated with a partial mapping (which is also denoted by M) from problems to answers as follows.

From an initial problem P_0, a rule set R, and a control *ctrl*, a partial mapping *finalProb* determines a final problem P_n as follows: $finalProb(P_0, R, ctrl)$ is the last element of $comp(P_0, R, ctrl)$ if $comp(P_0, R, ctrl)$ is defined, and is undefined otherwise. Using *finalProb* and A, we define a partial mapping M from *Prob* to W by

- $M(P_0) = ans(P_n)$, if $finalProb(P_0, R, ctrl) = P_n \in Prob$, $P_n \in dom(ans)$, and $ans \in A$,
- $M(P_0)$ is undefined otherwise.

M covers a problem P_0 iff $finalProb(P_0, R, ctrl) \in dom(A)$.

5. **Correctness.** We prove two correctness theorems: correctness of each computation path based on a solution method obtained using LPSF, and correctness of each solution method obtained using LPSF.

5.1. **Correctness of each computation path on a solution method.** Assume that M is a solution method designed based on the LPSF theory with parameters $\mathcal{L}, R, ctrl$, and A, i.e.,

$$M = \text{LPSF}(\mathcal{L}, R, ctrl, A).$$

Assume that we are given an MI problem $\langle Cs, \varphi \rangle$. If M covers $\langle Cs, \varphi \rangle$, then M determines a problem in the domain of A, and, from the final problem, an answer mapping ans in A determines a correct answer.

Theorem 5.1. *Assume that $M = LPSF(\mathcal{L}, R, ctrl, A)$. If M covers $\langle Cs, \varphi \rangle$, then $M(\langle Cs, \varphi \rangle)$ is the correct answer to $\langle Cs, \varphi \rangle$.*

Proof: Assume that M covers $\langle Cs, \varphi \rangle$. Then there is an ET sequence $[P_0, P_1, \ldots, P_n]$ such that $P_0 = \langle Cs, \varphi \rangle$ and $P_n = \langle Cs_n, \varphi_n \rangle \in dom(A)$. Since this sequence consists only of ET steps, $ans_{\text{MI}}(Cs, \varphi) = ans_{\text{MI}}(Cs_n, \varphi_n)$. Since ans in A is an answer mapping, $ans_{\text{MI}}(Cs_n, \varphi_n) = ans(Cs_n, \varphi_n)$. By the definition of ans_{MI}, $ans_{\text{MI}}(Cs, \varphi) = \varphi(\bigcap Models(Cs))$. Since $M(\langle Cs, \varphi \rangle) = ans(Cs_n, \varphi_n)$, $M(\langle Cs, \varphi \rangle) = \varphi(\bigcap Models(Cs))$. □

Theorem 5.1 says that if there is a computation path from a given initial problem P_0 to a final problem P_n in the domain of an answer mapping, P_n is mapped by the answer mapping to a correct answer, i.e., a result of computation $(ans(P_n))$ based on LPSF is always correct. This resembles the soundness of SLD resolution, where every computed answer for a given problem is a correct answer for the problem. However, computation in SLD resolution is for each answer, while computation in LPSF is for each answer set. SLD resolution transforms a clause into a clause by using a definite clause, while LPSF transforms a set of clauses into a set of clauses by using ET rules. By this gap, SLD

resolution for QA problems is less suitable for managing computation for "all-answers finding" problems. This is one of the most important points for developing correct and efficient solvers for various logical problems.

5.2. **Correctness of each solution method.** Assume that $M = \mathrm{LPSF}(\mathcal{L}, R, \mathit{ctrl}, A)$. The correctness of M is defined as follows.

Definition 5.1. *M is correct iff for each problem $prb \in Prob$, if M covers prb, then M gives a correct answer to prb.*

Based on the above definition of correctness of M, we have the following theorem.

Theorem 5.2. *For any $\mathcal{L}$, R, ctrl, and A, if $M = LPSF(\mathcal{L}, R, \mathit{ctrl}, A)$, then M is correct.*

Proof: Obvious from Theorem 5.1. □

As we stated at the top of the introduction, the conventional resolution-based theory fails to estabilish a correct proof method for full first-order formulas. Moreover, the conventional resolution-based theory has failed to estabilish a correct QA solution method for full first-order formulas. These solution methods cannot be practical since they are incorrect, i.e., they sometimes output incorrect answers. There are problems that cannot be solved correctly by these conventional methods, even though these problems are very small and there is no explosion of computation. The conventional methods have fatal limitation regarding computation correctness.

LPSF can overcome these difficulties by generation of new LPSF-based solvers. Theorem 5.2 says that a solver that is generated by $\mathcal{L}$, R, *ctrl*, and A is always correct, which forms a method of systematically making solvers for MI problems. The LPSF theory is, in some sense, a "meta-level" logic that can generate various logical solvers with guarantee of correctness.

Since it is axiomatic, any form of atoms can be used, regardless of whether they are included in first-order formulas or not. KR-Logic is invented under LPSF from first-order logic and LP-Logic. Function-variables are new terms that are not used in first-order logic and LP-Logic, by which we can transform each first-order formula equivalently into a set of clauses (extended with function variables). In this way, LPSF can extend conventional solvers, or invent new solvers with strict correctness.

6. **Designing Solution Methods Based on LPSF.** We define solvability of an LPSF solver. LPSF enables us to improve solvability of logical problem solving.

6.1. **Solvability.** Assume that $M = \mathrm{LPSF}(\mathcal{L}, R, \mathit{ctrl}, A)$. A solver M determines a set $FS(\mathcal{L}, R, \mathit{ctrl}, A)$ of all problems that are successfully solved:

$$\begin{aligned} &FS(\mathcal{L}, R, \mathit{ctrl}, A) \\ &= \{P_0 | (P_0 \text{ is an MI problem on } \mathcal{L}) \,\&\, (\mathit{finalProb}(P_0, R, \mathit{ctrl}) = P_n) \,\&\, (P_n \in \mathit{dom}(A))\}, \end{aligned}$$

where *FS* is a short form of finite solvability. If we have more ET rules, then more problems can be solved.

Theorem 6.1. *Assume that R_1 and R_2 are sets of ET rules such that $R_1 \subseteq R_2$. For any $\mathcal{L}$, ctrl_1, and A, there is ctrl_2 such that*

$$FS(\mathcal{L}, R_1, \mathit{ctrl}_1, A) \subseteq FS(\mathcal{L}, R_2, \mathit{ctrl}_2, A).$$

Proof: Define ctrl_2 by $\mathit{ctrl}_2 = \mathit{ctrl}_1$. Then, $FS(\mathcal{L}, R_1, \mathit{ctrl}_1, A) = FS(\mathcal{L}, R_2, \mathit{ctrl}_1, A) \subseteq FS(\mathcal{L}, R_2, \mathit{ctrl}_2, A)$. □

6.2. **Designing solution methods for logical problems.** Given a logical problem, we try to solve it using LPSF as follows.

1) Set a logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$.
2) Formalize the given problem as an MI problem $\langle Cs, \varphi \rangle$, where $Cs \in \mathcal{K}$.
3) Prepare a set R of ET rules.
4) Determine a control *ctrl*.
5) Determine a set A of answer mappings.
6) Obtain a solution method M by $M = \mathrm{LPSF}(\mathcal{L}, R, ctrl, A)$.
7) Solve the MI problem $\langle Cs, \varphi \rangle$ using M.

Each time we take parameters $\mathcal{L}$, R, *ctrl*, A, we have a solver M, which may succeed in solving the given problem, i.e.,

$$\langle Cs, \varphi \rangle \in FS(\mathcal{L}, R, ctrl, A),$$

or may fail to solve it. We repeat this generate-and-test process until the given problem is solved.

Given a logical structure and a set of answer mappings, one typical improvement is obtained by creation and accumulation of ET rules. Assume that $\mathcal{L}_0$ is a logical structure, and A_0 is a set of answer mappings. Consider the set of all problems that can be solved successfully using a set R of ET rules by some control, which is denoted by $FSA(R, \mathcal{L}_0, A_0)$ and is called a finitely solvable area with respect to R, i.e.,

$$FSA(R, \mathcal{L}_0, A_0) = \bigcup \{ FS(\mathcal{L}_0, R, ctrl, A_0) | ctrl \in CTRL \},$$

where $CTRL$ is the set of all controls. Given a problem *prob*, we try to solve *prob*, i.e., we try to find R such that

$$prob \in FSA(R, \mathcal{L}_0, A_0).$$

Maximization of $FSA(R, \mathcal{L}_0, A_0)$ by increasing R is of central importance in the research of logical computation. $FSA(R, \mathcal{L}_0, A_0)$ is simply denoted by $FSA(R)$ if $\mathcal{L}_0$ and A_0 are obvious from the context.

7. **Conclusions.** An axiomatic theory of Logical Problem Solving Framework (LPSF) was developed. The LPSF theory provides a general foundation for logical problem solving methods. LPSF enables us to improve solvability of logical problem solving. LPSF extends the most basic concepts of logic and computation as follows.

- LPSF utilizes a general concept of logical structures.
- LPSF regards MI problems as the main class of logical problems.
- LPSF takes ET rules as units of procedures.

LPSF is a generator of logical problem solving methods. Input parameters of LPSF consist of 1) a canonical logical structure, 2) a set of ET rules, 3) a control, and 4) a set of answer mappings. Each time we take parameters $\mathcal{L}$, R, *ctrl*, A, we have a solver M, which may succeed in solving the given problem or may fail to solve it. We repeat this generate-and-test process until the given problem is solved.

In typical usage of LPSF, we take some logical structure $\mathcal{L}_0$ and a set A_0 of answer mappings, and we adjust R and *ctrl* to improve solvability. Since an increase of ET rules leads to better solvability, improvement is typically attained by creation and accumulation of ET rules. Solvability is strictly evaluated in LPSF by the set $FSA(R, \mathcal{L}_0, A_0)$. We try to maximize $FSA(R, \mathcal{L}_0, A_0)$ by increasing R while keeping $\mathcal{L}_0$ and A_0.

Sometimes there are cases when we can never solve problems in the space of $\mathcal{L}_0$ with a set A_0 of answer mappings, i.e., there is no R such that

$$\langle Cs, \varphi \rangle \in FSA(R, \mathcal{L}_0, A_0).$$

Then we need to extend $\mathcal{L}_0$ into a new logical structure $\mathcal{L}_1$. In such cases, LPSF can also be used, i.e., we can try to create a new solver

$$M = \mathrm{LPSF}(\mathcal{L}_1, R, \mathit{ctrl}, A_1)$$

by setting a set A_1 of answer mappings, and by adjusting R and *ctrl*.

Using LPSF, we can evaluate logical problem solving methods by the partial ordering determined by the inclusion relation of finitely solvable areas. When we find a logical problem that is not a member of finitely solvable areas, we can develop a new logical problem solving method by

- creation of a new logical structure, and
- accumulation of ET rules to increase finitely solvable areas.

LPSF can be a general foundation of theories of logical problem solving and logical computation.

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. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.

[2] 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.

[3] 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.

[4] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.

[5] 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.

[6] J. R. Slagle, Automatic theorem proving with renamable and semantic resolution, *Journal of the ACM*, vol.14, pp.687-697, 1967.

[7] R. S. Boyer, *Locking: A Restriction of Resolution*, Ph.D. Thesis, University of Texas at Austin, Texas, 1971.

[8] D. W. Loveland, A linear format for resolution, *Proc. of the 1968 IRIA Symp. Automatic Demonstration*, Versailles, France, pp.147-162, 1970.

[9] D. Luckham, Refinement theorems in resolution theory, *Proc. of the 1968 IRIA Symp. Automatic Demonstration*, Versailles, France, pp.163-190, 1970.

[10] K. Akama and E. Nantajeewarawat, Unfolding-based simplification of query-answering problems in an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.9, no.9, pp.3515-3526, 2013.

[11] 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.

[12] K. Akama and E. Nantajeewarawat, Solving query-answering problems with constraints for function variables, in *Intelligent Information and Database Systems. ACIIDS 2018. Lecture Notes in Computer Science*, N. Nguyen, D. Hoang, T. P. Hong, H. Pham and B. Trawiński (eds.), Springer, DOI: 10.1007/978-3-319-75417-8_4, 2018.

[13] K. Akama, E. Nantajeewarawat and T. Akama, Term rewriting that preserves models in KR-Logic, in *Intelligent Information and Database Systems. ACIIDS 2019. Lecture Notes in Computer Science*, N. Nguyen, F. Gaol, T. P. Hong and B. Trawiński (eds.), Springer, DOI: 10.1007/978-3-030-14799-0_4, 2019.
[14] K. Akama and E. Nantajeewarawat, Logical structures on specialization systems: Formalization and satisfiability-preserving transformation, *Proc. of the 7th International Conference on Intelligent Technologies*, Taipei, Taiwan, pp.100-109, 2006.
[15] K. Akama, E. Nantajeewarawat and T. Akama, Computation control by prioritized ET rules, *Proc. of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, KEOD, Seville, Spain, vol.2, pp.84-95, 2018.
[16] K. Akama, E. Nantajeewarawat and T. Akama, Side-change transformation, *Proc. of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, KEOD, Seville, Spain, vol.2, pp.237-246, 2018.
[17] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, *The Description Logic Handbook*, 2nd Edition, Cambridge University Press, 2007.
[18] 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.
[19] 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.

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.

Prof. 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.