International Journal of Innovative Computing, Information and Control
Volume **18**, Number **4**, August **2022**

ICIC International ©2022 ISSN 1349-4198
pp. **1055–1069**

KNOWLEDGE-REPRESENTATION-LOGIC: AN EXTENSION OF FIRST-ORDER LOGIC

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 January 2022; revised April 2022

Abstract. *To develop a firm foundation for logical problem solving, and to improve solvability of logical problems, a new logic, called KR-Logic (knowledge-representation-logic), is invented as an extension of first-order logic with standard semantics and the one with Herbrand semantics. We propose a schema for developing a new logic as a logical structure by defining well-formed formulas, forms, mappings for transformation and evaluation, closed formulas, interpretations, and models. According to the schema, we construct KR-Logic together with LP-Logic. KR-Logic provides, among other things, function constants and function variables, which enable us to introduce existential quantification of function variables. If we take LP-Logic, solution methods cannot be complete due to non-preservation of models. If we take KR-Logic, models can be preserved, and we can try to solve all problems on first-order formulas by equivalent formula transformation with guarantee of computation correctness.*
Keywords: Logical structure, Equivalent transformation rule, Function variable, Proof problem, Query-answering problem

1. Introduction.

KR-Logic. To develop a firm foundation for logical problem solving, and to improve solvability of logical problems, we invent a new logic, called KR-Logic, where KR is an acronym for knowledge representation. KR-Logic is an extension of usual first-order logic, and provides, among other things, function constants and function variables. Knowledge representation in KR-Logic can have an extended clause that contains function variables and atoms of a special kind called *func*-atoms.

A logical structure is an axiomatic concept for defining a logical system. KR-Logic was defined as a logical structure (Section 2.1). Conventional first-order logic with classic interpretations is called FO-Logic in this paper, while the first-order logic with Herbrand interpretations, which is profoundly used in logic programming, is called LP-Logic. KR-Logic is considered as an extension of FO-Logic and LP-Logic (Figure 1, where "MI problems" and "QA problems" stand for "model-intersection problems" and "query-answering problems", respectively).

DOI: 10.24507/ijicic.18.04.1055

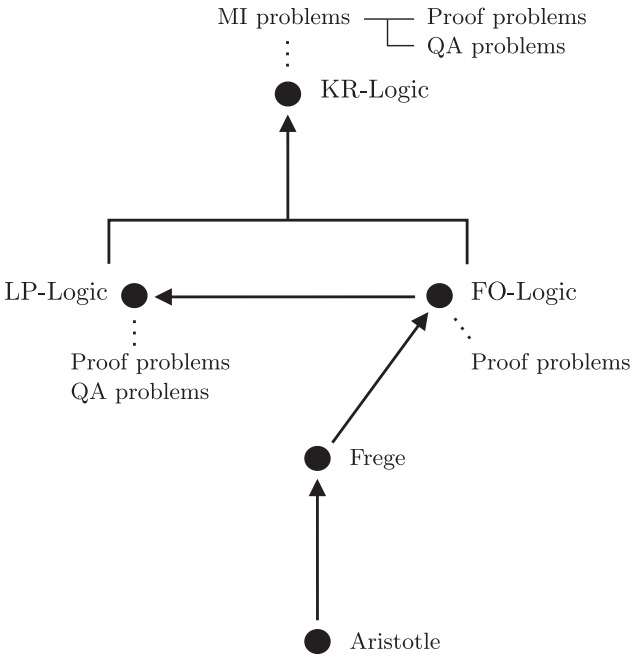

FIGURE 1. From FO-Logic and LP-Logic to KR-Logic

In the long history from Aristotle through Frege to the 20th century, first-order logic (FO-Logic) has been established as the main field of logic, in the center of which there have been two key concepts, proof problems and inference (Figure 1) [8, 9]. An interpretation in FO-Logic is rather complicated, consisting of a domain along with mappings and relations on the domain. It is considered to be designed mainly for the mathematical description in the movement of logicism, introduced by Frege [10] and developed by Whitehead and Russell [11, 12].

Canonical logical structures and MI problems. Similarly to Herbrand interpretations considered in the logic programming community, KR-Logic has a simple interpretation, i.e., an interpretation is a subset of a given set $\mathcal{G}_{\mathrm{u}}$, which is called a canonical interpretation. A logical structure that has canonical interpretations is called a canonical logical structure. If we are in the space of a canonical logical structure, we can take intersection of all models of a formula, which provides a foundation for defining model-intersection (MI) problems [13]. KR-Logic is considered as a knowledge representation system that associates each logical formula with the set of all of its models based on the concept of canonical interpretation (Section 2.3).

Increasing solvability. We have proposed a general schema for solving MI problems by equivalent transformation (ET), where problems are solved by repeated problem simplification using ET rules [14]. ET-based computation has been shown to be a fertile principle of solving logical problems. All deductive computation paths realized by logic programming (resolution) can also be produced by ET computation. Moreover, new deductive computation paths can be produced by ET computation [15]. KR-Logic increases the set of problems that can be solved successfully by formalizing them as MI problems and using various ET rules [14, 16, 17, 18, 19].

Organization. The rest of the paper is organized as follows. Section 2 explains a methodology to develop a new logic, according to which we construct a new logic in subsequent sections. Section 3 introduces the concept of well-formed formula. Section 4 defines the

concept of forms on an arbitrary set together with two mappings, *value* and *subst*. Section 5 defines the concept of interpretations. Section 6 introduces a mapping mf. Section 7 introduces the concept of closed formulas, based on which a logical structure for KR-Logic is defined. Section 8 states that KR-Logic overcomes the limitation of first-order logic. Section 9 provides concluding remarks.

Notations. The following notations will be used. 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, $parMap(A, B)$ denotes the set of all partial mappings from A to B, and for any partial mapping f from A to B, $dom(f)$ denotes the domain of f, i.e., $dom(f) = \{a \mid (a \in A)\ \&\ (f(a) \text{ is defined})\}$.

2. How to Construct a Logic.

For the preparation of constructing KR-Logic in later sections, we explain a schema for developing a new logic as a logical structure by defining well-formed formulas (wffs), forms, mappings for transformation/evaluation, closed formulas, interpretations, and models.

2.1. Logical structures and basic related concepts.

The notion of a logical structure below is a core structure of logic. Given a description (a formula) and an interpretation, a logical structure determines a truth value. Hence a logical structure also produces a mapping to determine the set of all models of each description (each formula).

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 is a mapping from $\mathcal{K}$ to $Map(\mathcal{I}, \{true, false\})$.

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

Despite its simplicity, this abstract notion 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) 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.

2.2. A method for constructing logical structures.

We propose in this section a method for constructing logical structures (Figure 2). By this method, $ATOM$, $\mathcal{A}_{\mathrm{u}}$, $\mathcal{G}_{\mathrm{u}}$, WFF, and mf together determine $\mathcal{K}$, $\mathcal{I}$ and ν, thereby giving a logical structure $\langle \mathcal{K}, \mathcal{I}, \nu \rangle$ as follows.

1) Let $ATOM$, $\mathcal{A}_{\mathrm{u}}$, and $\mathcal{G}_{\mathrm{u}}$ be arbitrary sets such that $ATOM \supseteq \mathcal{A}_{\mathrm{u}} \supseteq \mathcal{G}_{\mathrm{u}}$. The elements of $\mathcal{A}_{\mathrm{u}}$ and those of $\mathcal{G}_{\mathrm{u}}$ are called, respectively, user-defined atoms and user-defined ground atoms.
2) Let WFF be an arbitrary set such that WFF $\supseteq ATOM$. Each element of WFF is called a well-formed formula (wff).

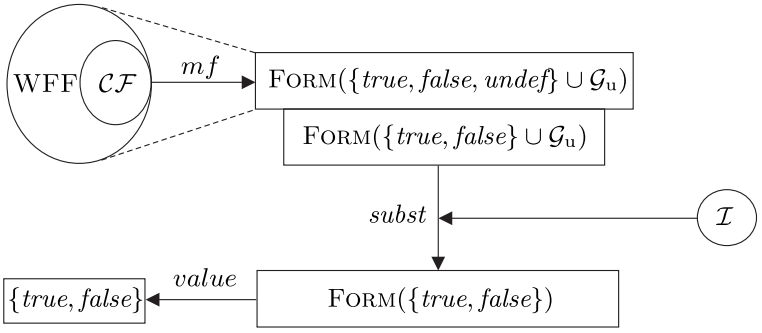

FIGURE 2. Defining a logical structure with $\nu : \mathcal{CF} \to Map(\mathcal{I}, \{true, false\})$

3) Let FORM(A) be the set of all forms on an arbitrary set A, which is defined in Section 4.
4) Let *value* be a mapping from FORM($\{true, false\}$) to $\{true, false\}$ that evaluates forms.
5) Let *mf* be a mapping from WFF to FORM($\{true, false, undef\} \cup \mathcal{G}_u$).
6) Let $\mathcal{CF}$ be defined as the set of all closed formulas, where a *closed formula* is defined as a wff E such that $mf(E)$ is a form that does not contain *undef*.
7) Let $\mathcal{I} = pow(\mathcal{G}_u)$. Each element of $\mathcal{I}$ is called an interpretation.
8) Let *val* be a mapping from $\mathcal{G}_u \times \mathcal{I}$ to $\{true, false\}$, which is defined in Section 5.2.
9) Let $\mathcal{K} = \mathcal{CF}$.
10) Let *subst* be a mapping from FORM($\{true, false\} \cup \mathcal{G}_u$) $\times \mathcal{I}$ to FORM($\{true, false\}$) such that $subst(f, I)$ is obtained from f by substituting each atom $g \in \mathcal{G}_u$ occurring in f with $val(g, I)$, which is formally defined in Section 4.3.
11) Let $\nu : \mathcal{K} \to Map(\mathcal{I}, \{true, false\})$ be defined by: for any $E \in \mathcal{K}$ and any $I \in \mathcal{I}$, $\nu(E)(I) = value(subst(mf(E), I))$.

This method has five input parameters: *ATOM*, $\mathcal{A}_u$, $\mathcal{G}_u$, WFF, and *mf*, while FORM, *value*, *val*, and *subst* have the common definitions. They produce $\mathcal{K}$, $\mathcal{I}$ and ν to determine a logical structure $\langle \mathcal{K}, \mathcal{I}, \nu \rangle$. In Section 7.2, we will make two logical structures, LP-Logic and KR-Logic that have the common interpretation domain $\mathcal{G}_u$. The four parameter-value pairs, $ATOM = ATOM_{LP}$, $\mathcal{A} = \mathcal{A}_{LP}$, WFF = WFF_{LP}, and $mf = mf_{LP}$ determine LP-Logic, while the four parameter-value pairs, $ATOM = ATOM_{KR}$, $\mathcal{A} = \mathcal{A}_{KR}$, WFF = WFF_{KR}, and $mf = mf_{KR}$ determine KR-Logic.

2.3. **Representing models.** Assume that a logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$ is given. Let E be a closed formula in $\mathcal{K}$. Each element $I \in \mathcal{I}$ such that $\nu(E)(I) = true$ is a model of E. Let $Models(E)$ be defined by

$$Models(E) = \{I \mid (I \in \mathcal{I}) \ \& \ (\nu(E)(I) = true)\},$$

i.e., $Models(E)$ is the set of all models of E.

Assume that the logical structure $\mathcal{L}$ has canonical interpretations in $\mathcal{I} = pow(\mathcal{G}_u)$. Then *Models* is regarded as a mapping from $\mathcal{CF}$ to $pow(pow(\mathcal{G}_u))$ and is called a *representation mapping*. We say that a formula E represents $Models(E)$. The main role of a logical structure is to define a representation mapping.

2.4. **Model-intersection problems.** With canonical interpretations, we define a class of problems, called *model-intersection (MI) problems*. Given a closed formula $E \in \mathcal{CF}$ and a mapping φ from $pow(\mathcal{G}_u)$ to a set W, a pair $\langle E, \varphi \rangle$ is a model-intersection problem,

the answer to which, denoted by $ans(E, \varphi)$, is defined by

$$ans(E, \varphi) = \varphi\left(\bigcap Models(E)\right).$$

For more detailed explanation and examples, the reader is referred to [13, 14].

3. **Well-Formed Formulas.** Based on the method of constructing a logical structure explained in Section 2, we will define the concepts of term and atom, on which we introduce the concept of well-formed formula. There are three kinds of atoms, i.e., user-defined atoms, built-in constraint atoms, and *func*-atoms.

3.1. **Alphabet.** An alphabet $\langle \mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV}, Pred, Pred_{\mathrm{C}} \rangle$ is assumed, where $\mathbb{F}$ is a set of usual function symbols, $\mathbb{V}$ a set of usual variables, $\mathbb{FC}$ a set of function constants, $\mathbb{FV}$ a set of function variables, $Pred$ a set of usual predicate symbols, and $Pred_{\mathrm{C}}$ a set of built-in constraint predicate symbols. There are two types of variables: usual variables and function variables. (In Figure 3, for example, x and y are usual variables, while f_0 and f_1 are function variables.)

F_1: $\forall x : (live(x, D) \vee \neg func(f_0, x))$
F_2: $\forall x : (kill(f_0, A) \vee \neg func(f_0, x))$
F_3: $\forall x : (\neg live(x, D) \vee \neg neq(x, A) \vee \neg neq(x, B) \vee \neg neq(x, C))$
F_4: $live(A, D)$
F_5: $live(B, D)$
F_6: $live(C, D)$
F_7: $\forall x : (\forall y : (hate(x, y) \vee \neg kill(x, y)))$
F_8: $\forall x : (\forall y : (\neg kill(x, y) \vee \neg richer(x, y)))$
F_9: $\forall x : (\neg hate(A, x) \vee \neg hate(C, x) \vee \neg live(x, D))$
F_{10}: $\forall x : (hate(A, x) \vee \neg neq(x, B) \vee \neg live(x, D))$
F_{11}: $\forall x : (richer(x, A) \vee hate(B, x) \vee \neg live(x, D))$
F_{12}: $\forall x : (hate(B, x) \vee \neg hate(A, x) \vee \neg live(x, D))$
F_{13}: $\forall x : (\forall y : (\neg hate(x, y) \vee \neg live(x, D) \vee \neg func(f_1, x, y)))$
F_{14}: $\forall x : (\forall y : (live(y, D) \vee \neg live(x, D) \vee \neg func(f_1, x, y)))$

FIGURE 3. A set of formulas in KR-Logic

3.2. ***ATOM*, $\mathcal{A}_{\mathbf{u}}$, and $\mathcal{G}_{\mathbf{u}}$.** Let $\mathrm{T}(\mathbb{F}, \mathbb{V})$ be the set of all terms on $\langle \mathbb{F}, \mathbb{V} \rangle$, which are inductively defined as follows.

1) If $f \in \mathbb{F}$ is a 0-ary function symbol, then f is a term on $\langle \mathbb{F}, \mathbb{V} \rangle$.
2) If $v \in \mathbb{V}$ is a variable, then v is a term on $\langle \mathbb{F}, \mathbb{V} \rangle$.
3) If $f \in \mathbb{F}$ is an n-ary function symbol, where $n \geq 1$, and $t_1, \ldots, t_n$ are terms on $\langle \mathbb{F}, \mathbb{V} \rangle$, then $f(t_1, \ldots, t_n)$ is a term on $\langle \mathbb{F}, \mathbb{V} \rangle$.

Let $\mathrm{T}(\mathbb{F})$ be the set of all terms in $\mathrm{T}(\mathbb{F}, \mathbb{V})$ that contain no variable. A term in $\mathrm{T}(\mathbb{F})$ is called a *ground term*.

We use 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 n-ary predicate and the t_i are usual terms. Supposing that *live*, *kill*, and *hate* are user-defined predicates, $live(x, D)$, $kill(x, y)$, and $hate(A, x)$ are user-defined atoms (cf. Figure 3). 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 n-ary constraint predicate and the t_i are usual terms. Typical examples of built-in constraint atoms are $eq(x, x)$ and $neq(1, 2)$, where *eq* and *neq* are predefined constraint predicates, standing for "equal" and "not equal", respectively (cf. Figure 3).

Let $\mathcal{A}_{\mathrm{u}}$ be the set of all user-defined atoms, $\mathcal{G}_{\mathrm{u}}$ the set of all ground user-defined atoms, $\mathcal{A}_{\mathrm{c}}$ the set of all constraint atoms, and $\mathcal{G}_{\mathrm{c}}$ the set of all ground constraint atoms. Let TCon denote the set of all ground built-in constraint atoms that are true.

A *func-atom* is an expression of the form $\mathit{func}(f, t_1, \ldots, t_n, t_{n+1})$, where f is either an n-ary function constant or an n-ary function variable, and the t_i are usual terms. For example, supposing that f_1 is a unary function variable, $\mathit{func}(f_1, x, y)$ is a *func*-atom (cf. Figure 3). A *func*-atom $\mathit{func}(f, t_1, \ldots, t_n, t_{n+1})$ is *ground* if f is a function constant and the t_i are ground usual terms. Let $\mathcal{F}$ be the set of all *func*-atoms. Let *ATOM* consist of all atoms in $\mathcal{A}_{\mathrm{u}}$, $\mathcal{A}_{\mathrm{c}}$, and $\mathcal{F}$, i.e., $\mathit{ATOM} = \mathcal{A}_{\mathrm{u}} \cup \mathcal{A}_{\mathrm{c}} \cup \mathcal{F}$.

A variable in $\mathbb{V}$ is specialized by instantiation into a term in $\mathrm{T}(\mathbb{F}, \mathbb{V})$. A function variable in $\mathbb{FV}$ is specialized by instantiation into a function variable in $\mathbb{FV}$ or a function constant in $\mathbb{FC}$, i.e., a substitution for function variables is a mapping from $\mathbb{FV}$ to $\mathbb{FV} \cup \mathbb{FC}$. Each n-ary function constant is associated with a mapping from $\mathrm{T}(\mathbb{F})^n$ to $\mathrm{T}(\mathbb{F})$. Let ψ be the mapping that determines such association. More precisely, ψ is a surjective mapping from $\mathbb{FC}$ to

$$\bigcup_{i \in \{0,1,2,\ldots\}} \mathit{Map}\left(\mathrm{T}(\mathbb{F})^i, \mathrm{T}(\mathbb{F})\right).$$

For each n-ary function constant f_c, $\psi(f_c)$ is a mapping from $\mathrm{T}(\mathbb{F})^n$ to $\mathrm{T}(\mathbb{F})$. For each mapping m in $\mathit{Map}(\mathrm{T}(\mathbb{F})^n, \mathrm{T}(\mathbb{F}))$, there is an n-ary function constant $f_c \in \mathbb{FC}$ such that $\psi(f_c) = m$.

3.3. **The set of all well-formed formulas.** A *well-formed formula* (for short, *wff*) is constructed inductively by finitely many applications of the following rules.

1) *Atoms:* If α is an atom in $\mathcal{A}_{\mathrm{u}} \cup \mathcal{A}_{\mathrm{c}} \cup \mathcal{F}$, then α is a wff.
2) *Negation:* If α is a wff, then $\neg\alpha$ is a wff.
3) *Binary connectives:* If α and β are wffs, then $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, and $\alpha \leftrightarrow \beta$ are wffs.
4) *Quantifiers for usual variables:* If α is a wff and v is a variable in $\mathbb{V}$, then $\forall v : \alpha$ and $\exists v : \alpha$ are wffs.
5) *Quantifiers for function variables:* If α is a wff and f_v is a variable in $\mathbb{FV}$, then $\forall_F f_v : \alpha$ and $\exists_F f_v : \alpha$ are wffs.

In Figure 3, all the formulas F_1-F_{14} are wffs in KR-Logic. The usual variables x and y are universally quantified. However, the function variables f_0 and f_1 are not quantified. For representing the background knowledge of the Agatha puzzle [6], the wff

$$\exists_F f_0 : (\exists_F f_1 : (F_1 \wedge F_2 \wedge \cdots \wedge F_{14}))$$

is used.

4. **Forms.** We define the concept of form on an arbitrary set together with two mappings: *value* for truth-value evaluation and *subst* for replacement of ground user-defined atoms with their truth values.

4.1. **Forms on an arbitrary set.** We define the notion of a form, which is a key concept in the method for constructing logical structures. Let A be an arbitrary set. We define a *form* on A inductively as follows.

1) Each element of A is a form on A.
2) If f is a form on A, then $\neg f$ is a form on A.
3) If f_1 and f_2 are forms on A, then $f_1 \wedge f_2$ and $f_1 \vee f_2$ are forms on A.
4) If S is a set of forms on A, then $\bigwedge S$ is a form on A.
5) If S is a set of forms on A, then $\bigvee S$ is a form on A.

The set of all forms on A is denoted by $\textsc{Form}(A)$.

4.2. **A mapping *value*.** A mapping *value* evaluates forms in $\textsc{Form}(\{true, false\})$ to give *true* or *false*. Formally, *value* is a mapping:

$$value : \textsc{Form}(\{true, false\}) \to \{true, false\}.$$

Assuming that f, f_1, and f_2 are forms on $\{true, false\}$ and S is a set of forms on $\{true, false\}$, the value of a form in $\textsc{Form}(\{true, false\})$ is defined inductively as follows.

1) The value of *true* is true and that of *false* is false.
2) If f is true, then $\neg f$ is false. If f is false, then $\neg f$ is true.
3) If f_1 and f_2 are true, then $f_1 \wedge f_2$ is true; otherwise it is false. If at least one of f_1 and f_2 is true, then $f_1 \vee f_2$ is true; otherwise it is false.
4) If S contains a form that is false, then $\bigwedge S$ is false; otherwise $\bigwedge S$ is true.
5) If S contains a form that is true, then $\bigvee S$ is true; otherwise $\bigvee S$ is false.

4.3. **A mapping *subst*.** A mapping *subst* replaces a ground atom in a form with a truth value with respect to an interpretation. Formally, *subst* is a mapping

$$subst : \textsc{Form}(\{true, false\} \cup \mathcal{G}_{\mathrm{u}}) \times \mathcal{I} \to \textsc{Form}(\{true, false\}),$$

such that $subst(f, I)$ is obtained from f by substituting each atom $g \in \mathcal{G}_{\mathrm{u}}$ occurring in f with the truth value determined by $val(g, I)$ as the following inductive definition shows.

1) $subst(true, I) = true$.
2) $subst(false, I) = false$.
3) $subst(g, I) = val(g, I)$ if $g \in \mathcal{G}_{\mathrm{u}}$.
4) $subst(\neg f, I) = \neg subst(f, I)$.
5) $subst(f_1 \wedge f_2, I) = subst(f_1, I) \wedge subst(f_2, I)$.
6) $subst(f_1 \vee f_2, I) = subst(f_1, I) \vee subst(f_2, I)$.
7) $subst(\bigwedge S, I) = \bigwedge\{subst(f, I) \mid f \in S\}$, if S is a set of forms.
8) $subst(\bigvee S, I) = \bigvee\{subst(f, I) \mid f \in S\}$, if S is a set of forms.

An extension of *subst* to a mapping on $\textsc{Form}(\{true, false, undef\} \cup \mathcal{G}_{\mathrm{u}})$ can be obtained by adding a condition: $subst(undef, I) = undef$. The definition of *subst* in this paper is restricted by exclusion of *undef* since we can consider only closed formulas (not containing *undef*) when defining the mapping ν in Section 7.1.

5. **Interpretations.** We define the concept of interpretation, which is a set of ground user-defined atoms and is similar to a Herbrand interpretation frequently used in logic programming.

5.1. **The set $\mathcal{I}$ of all interpretations.** An interpretation is defined as a subset of $\mathcal{G}_{\mathrm{u}}$, where $\mathcal{G}_{\mathrm{u}}$ is the set of all ground user-defined atoms and is equal to the set

$$\{p(t_1, \ldots, t_m) \mid (m \text{ is a nonnegative integer}) \,\&\, (p \text{ is an } m\text{-ary predicate in } Pred) \,\&\, (t_1, \ldots, t_m \in \mathrm{T}(\mathbb{F}))\}.$$

Let $\mathcal{I}$ denote the set of all interpretations, i.e., $\mathcal{I} = pow(\mathcal{G}_{\mathrm{u}})$. The set $\mathcal{G}_{\mathrm{u}}$ is called the interpretation domain.

A standard interpretation for a first-order formula is domain-based, i.e., for each domain considered, each constant in the formula is associated with an element in the domain, and each function symbol is associated with a mapping on the domain [20]. A Herbrand interpretation for a first-order formula is defined again in terms of constants, functions, and predicates. These definitions are specific to first-order terms and predicates, and hence not adequate for invention of a new logic.

The concept of interpretation in this paper is dependent on neither the domain structure nor specific structure (such as functions and predicates) in first-order logic. It is relying on a general concept of logical structure (Section 2.2). An interpretation in our theory is a subset of the interpretation domain $\mathcal{G}_{\mathrm{u}}$, which is common to all formulas. This is a sharp contrast to a Herbrand interpretation in first-order logic. Each Hebrand interpretation is regarded as a subset of a Herbrand base [20]. However, a Herbrand base is determined by each formula depending on specific structures (constants and functions) of a formula. The concept of Herbrand interpretation for first-order logic needs to be modified for invention of a new logic.

5.2. **A mapping *val*.** An interpretation in $\mathcal{I}$ determines truth or falsity for each ground atom in $\mathcal{G}_{\mathrm{u}}$, which is described by a mapping $val : \mathcal{G}_{\mathrm{u}} \times \mathcal{I} \to \{true, false\}$, where for any interpretation $I \in \mathcal{I}$,

$$val(g, I) = \begin{cases} true & \text{if } g \in I, \\ false & \text{otherwise.} \end{cases}$$

6. **Transformation of Well-Formed Formulas.** We introduce a mapping *mf* together with two mappings, *e* and *ev*. By calling the mapping *ev*, the mapping *mf* transforms a wff into a formula called a form that may include ground user-defined atoms and *undef*.

6.1. **A mapping *e*.** Given a substitution ρ for usual/function variables, *ev* instantiates a wff into a form using instantiation of terms by ρ. Term instantiation is necessary since quantifiers implicitly include instantiation of variables. Given a substitution for variables, *e* instantiates terms with evaluation of usual/function variables.

More formally,

$$e : \mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV}) \times \textsc{Rho} \to \mathrm{T}(\mathbb{F}) \cup \{undef\}$$

is defined, where a set Rho of bindings is given as the second argument and *undef* denotes "not full instantiation".

1) If v is a variable in $\mathbb{V}$ and $v \in dom(\rho)$, then $e(v, \rho) = \rho(v)$.
2) If v is a variable in $\mathbb{V}$ and $v \notin dom(\rho)$, then $e(v, \rho) = undef$.
3) If f_s is an n-ary function symbol in $\mathbb{F}$ and $t = f_s(t_1, \ldots, t_m)$, then
   a) $e(t, \rho) = f_s(e(t_1, \rho), \ldots, e(t_n, \rho))$ if for each $i \in \{1, \ldots, m\}$, $e(t_i, \rho) \in \mathrm{T}(\mathbb{F})$;
   b) $e(t, \rho) = undef$ otherwise.
4) If f_v is an m-ary function variable in $\mathbb{FV}$ and $t = f_v(t_1, \ldots, t_m)$, then
   a) $e(t, \rho) = \psi(\rho(f_v))(e(t_1, \rho), \ldots, e(t_m, \rho))$ if $f_v \in dom(\rho)$ and for each $i \in \{1, \ldots, m\}$, $e(t_i, \rho) \in \mathrm{T}(\mathbb{F})$;
   b) $e(t, \rho) = undef$ otherwise.
5) If f_c is an m-ary function constant in $\mathbb{FC}$ and $t = f_c(t_1, \ldots, t_m)$, then
   a) $e(t, \rho) = \psi(f_c)(e(t_1, \rho), \ldots, e(t_m, \rho))$ if for each $i \in \{1, \ldots, m\}$, $e(t_i, \rho) \in \mathrm{T}(\mathbb{F})$;
   b) $e(t, \rho) = undef$ otherwise.

6.2. **Bindings and substitutions.** Bindings and substitutions are used for specializing variables. Substitutions are formed by accumulation of bindings. A substitution determines a partial mapping.

6.2.1. *Ground bindings and ground substitutions.* There are two kinds of ground bindings; one is for usual variables, and the other is for function variables. A *ground binding* for a usual variable is a pair $\langle v, t \rangle$ such that $v \in \mathbb{V}$ and $t \in \mathrm{T}(\mathbb{F})$. A *ground binding* for a function variable is a pair $\langle f_v, f_c \rangle$ such that $f_v \in \mathbb{FV}$ and $f_c \in \mathbb{FC}$.

A *ground substitution* is a finite set of ground bindings for usual variables and/or ground bindings for function variables that contains no different bindings for the same (usual or

function) variable. The empty substitution is denoted by $\emptyset$. For example, if $x \in \mathbb{V}$, $t \in \mathrm{T}(\mathbb{F})$, $h \in \mathbb{FV}$, and $g \in \mathbb{FC}$, then $\{\langle x, t\rangle, \langle h, g\rangle\}$ is a ground substitution.

Only ground substitutions and ground bindings are considered in this paper. When no confusion occurs, they are often simply called substitutions and bindings, respectively.

6.2.2. *Associated mappings.* A ground substitution for usual variables determines a partial mapping from $\mathbb{V}$ to $\mathrm{T}(\mathbb{F})$. A ground substitution for function variables determines a partial mapping from $\mathbb{FV}$ to $\mathbb{FC}$. A ground substitution ρ is identified with the partial mapping that is determined by ρ. Letting $\textsc{Rho}_1 = \mathit{parMap}(\mathbb{V}, \mathrm{T}(\mathbb{F}))$ and $\textsc{Rho}_2 = \mathit{parMap}(\mathbb{FV}, \mathbb{FC})$, we define the set $\textsc{Rho}$ of all ground substitutions by $\textsc{Rho} = \textsc{Rho}_1 \cup \textsc{Rho}_2$.

6.2.3. *Addition of bindings to a substitution.* Let $\langle v, t\rangle$ be a ground binding for a usual variable, $\langle f_v, f_c\rangle$ a ground binding for a function variable, and ρ a ground substitution. $\langle v, t\rangle \oplus \rho$ and $\langle f_v, f_c\rangle \oplus \rho$ are the ground substitutions defined as follows:

- $\langle v, t\rangle \oplus \rho = \{\langle v, t\rangle\} \cup (\rho - (\{v\} \times \mathrm{T}(\mathbb{F})))$.
- $\langle f_v, f_c\rangle \oplus \rho = \{\langle f_v, f_c\rangle\} \cup (\rho - (\{f_v\} \times \mathbb{FC}))$.

Let $b_1, b_2, \ldots, b_n$ be ground bindings, where $n \geq 2$. Then $b_1 \oplus b_2 \oplus \cdots \oplus b_n$ is a ground substitution inductively defined by

- $b_1 \oplus b_2 \oplus \cdots \oplus b_n = b_1 \oplus (b_2 \oplus \cdots \oplus b_n)$ if $n \geq 3$, and
- $b_1 \oplus b_2 = b_1 \oplus \{b_2\}$.

Note that $\oplus$ overwrites bindings with the same variables, e.g., $\langle x, 2\rangle \oplus \langle x, 5\rangle = \{\langle x, 2\rangle\}$, and $\langle f_v, f_{c1}\rangle \oplus \langle f_v, f_{c2}\rangle = \{\langle f_v, f_{c1}\rangle\}$.

6.3. **A mapping *ev*.** Given a ground substitution ρ for variables, ev transforms a wff into a form using specialization of terms by ρ. Formally, ev is a mapping:

$$ev : \mathrm{WFF} \times \textsc{Rho} \to \textsc{Form}(\{\mathit{true}, \mathit{false}, \mathit{undef}\} \cup \mathcal{G}_{\mathrm{u}}).$$

For intuitive understanding, consider the following examples.

1) $ev(p(x), \{\langle x, 5\rangle\}) = p(5)$.
2) $ev(eq(x, 6), \{\langle x, 7\rangle\}) = \mathit{false}$.
3) $ev(\mathit{func}(f_v, x, y), \langle f_v, f_c\rangle \oplus \langle x, 2\rangle \oplus \langle y, 3\rangle) = \mathit{true}$ if $f_c(2) = 3$.
4) $ev(\mathit{func}(f_c, x, y), \langle x, 3\rangle \oplus \langle y, 4\rangle) = \mathit{false}$ if $f_c(3) = 7$.
5) $ev(\neg p(x), \{\langle x, 9\rangle\}) = \neg p(9)$.
6) $ev(p(x) \wedge q(y), \langle x, 9\rangle \oplus \langle y, 1\rangle) = p(9) \wedge q(1)$.
7) $ev(\forall x : p(x), \{\}) = p(1) \wedge p(2) \wedge \cdots$.
8) $ev(\forall x : (\forall y : p(x, y)), \{\}) = p(1, 1) \wedge p(2, 3) \wedge \cdots$.
9) $ev(\forall x : (\exists x : p(x)), \{\}) = p(1) \vee p(2) \vee \cdots$.
10) $ev(\exists x : (\forall x : p(x)), \{\}) = p(1) \wedge p(2) \wedge \cdots$.

Infinite conjunction and disjunction in the 7th to 10th examples above are formally represented by $\bigwedge S$ and $\bigvee S$ with an infinite set S.

The formal definition of the mapping ev. Based on the notation related to bindings, we give the formal definition of the mapping ev below. Let $\rho \in \textsc{Rho}$. In the following items (5th to 13th), assume that α and β are wffs.

1) If a is an atom $p(t_1, \ldots, t_m)$, where p is an m-ary predicate in *Pred*, then
   a) $ev(a, \rho) = p(e(t_1, \rho), \ldots, e(t_m, \rho))$ if for each $i \in \{1, \ldots, m\}$, $e(t_i, \rho) \in \mathrm{T}(\mathbb{F})$;
   b) $ev(a, \rho) = \mathit{undef}$ otherwise.
2) If a is an atom $p(t_1, \ldots, t_m)$, where p is an m-ary constraint predicate in $\mathit{Pred}_{\mathrm{C}}$, then
   a) $ev(a, \rho) = \mathit{true}$ if (i) for each $i \in \{1, \ldots, m\}$, $e(t_i, \rho) \in \mathrm{T}(\mathbb{F})$, and (ii) $p(e(t_1, \rho), \ldots, e(t_m, \rho)) \in \mathrm{TCon}$;

   b) $ev(a,\rho) = \mathit{false}$ if (i) for each $i \in \{1,\ldots,m\}$, $e(t_i,\rho) \in \mathrm{T}(\mathbb{F})$, and (ii) $p(e(t_1,\rho), \ldots, e(t_m,\rho)) \in \mathcal{G}_\mathrm{u} - \mathrm{TCON}$;
   c) $ev(a,\rho) = \mathit{undef}$ otherwise.
3) If a is a *func*-atom $\mathit{func}(f_v, t_1, \ldots, t_m, t)$, where $f_v \in \mathbb{FV}$, then
   a) $ev(a,\rho) = \mathit{true}$ if (i) $e(t,\rho) \in \mathrm{T}(\mathbb{F})$, (ii) for each $i \in \{1,\ldots,m\}$, $e(t_i,\rho) \in \mathrm{T}(\mathbb{F})$, (iii) $f_v \in \mathit{dom}(\rho)$, and (iv) $\psi(\rho(f_v))(e(t_1,\rho),\ldots,e(t_m,\rho)) = e(t,\rho)$;
   b) $ev(a,\rho) = \mathit{false}$ if (i) $e(t,\rho) \in \mathrm{T}(\mathbb{F})$, (ii) for each $i \in \{1,\ldots,m\}$, $e(t_i,\rho) \in \mathrm{T}(\mathbb{F})$, (iii) $f_v \in \mathit{dom}(\rho)$, and (iv) $\psi(\rho(f_v))(e(t_1,\rho),\ldots,e(t_m,\rho)) \neq e(t,\rho)$;
   c) $ev(a,\rho) = \mathit{undef}$ otherwise.
4) If a is a *func*-atom $\mathit{func}(f_c, t_1, \ldots, t_m, t)$, where $f_c \in \mathbb{FC}$, then
   a) $ev(a,\rho) = \mathit{true}$ if (i) $e(t,\rho) \in \mathrm{T}(\mathbb{F})$, (ii) for each $i \in \{1,\ldots,m\}$, $e(t_i,\rho) \in \mathrm{T}(\mathbb{F})$, and (iii) $\psi(f_c)(e(t_1,\rho),\ldots,e(t_m,\rho)) = e(t,\rho)$;
   b) $ev(a,\rho) = \mathit{false}$ if (i) $e(t,\rho) \in \mathrm{T}(\mathbb{F})$, (ii) for each $i \in \{1,\ldots,m\}$, $e(t_i,\rho) \in \mathrm{T}(\mathbb{F})$, and (iii) $\psi(f_c)(e(t_1,\rho),\ldots, e(t_m,\rho)) \neq e(t,\rho)$;
   c) $ev(a,\rho) = \mathit{undef}$ otherwise.
5) $ev(\neg\alpha,\rho) = \neg(ev(\alpha,\rho))$.
6) $ev(\alpha \wedge \beta,\rho) = ev(\alpha,\rho) \wedge ev(\beta,\rho)$.
7) $ev(\alpha \vee \beta,\rho) = ev(\alpha,\rho) \vee ev(\beta,\rho)$.
8) $ev(\alpha \to \beta,\rho) = ev(\neg\alpha \vee \beta,\rho)$.
9) $ev(\alpha \leftrightarrow \beta,\rho) = ev((\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha),\rho)$.
10) If $v \in \mathbb{V}$, then $ev(\forall v : \alpha,\rho) = \bigwedge\{ev(\alpha, \langle v,t\rangle \oplus \rho) \mid t \in \mathrm{T}(\mathbb{F})\}$.
11) If $v \in \mathbb{V}$, then $ev(\exists v : \alpha,\rho) = \bigvee\{ev(\alpha, \langle v,t\rangle \oplus \rho) \mid t \in \mathrm{T}(\mathbb{F})\}$.
12) If $f_v \in \mathbb{FV}$, then $ev(\forall_F f_v : \alpha,\rho) = \bigwedge\{ev(\alpha, \langle f_v, f_c\rangle \oplus \rho) \mid f_c \in \mathbb{FC}\}$.
13) If $f_v \in \mathbb{FV}$, then $ev(\exists_F f_v : \alpha,\rho) = \bigvee\{ev(\alpha, \langle f_v, f_c\rangle \oplus \rho) \mid f_c \in \mathbb{FC}\}$.

6.4. **Definition of a mapping *mf*.** Given a wff α, *mf* transforms α to give a form that may include ground user-defined atoms and is later evaluated to be true or false by an interpretation. Formally,

$$\mathit{mf} : \mathrm{WFF} \to \textsc{Form}(\{\mathit{true}, \mathit{false}, \mathit{undef}\} \cup \mathcal{G}_\mathrm{u})$$

is defined by simply calling the mapping ev as follows: for any wff α,

$$\mathit{mf}(\alpha) = ev(\alpha, \{\}).$$

Examples are shown as follows.

1) $\mathit{mf}(p(5)) = p(5)$.
2) $\mathit{mf}(eq(5,6)) = \mathit{false}$.
3) $\mathit{mf}(\mathit{func}(f_v, 2, 4)) = \mathit{undef}$.
4) $\mathit{mf}(\mathit{func}(f_c, 2, 4)) = \mathit{false}$ if $f_c(2) = 3$.
5) $\mathit{mf}(\neg p(8)) = \neg p(8)$.
6) $\mathit{mf}(p(9) \wedge eq(3,3)) = p(9) \wedge \mathit{true}$.
7) $\mathit{mf}(\forall x : p(x)) = p(1) \wedge p(2) \wedge \cdots$.
8) $\mathit{mf}(\forall x : (\forall y : p(x,y))) = p(1,1) \wedge p(2,3) \wedge \cdots$.
9) $\mathit{mf}(\forall x : (\exists x : p(x))) = p(1) \vee p(2) \vee \cdots$.
10) $\mathit{mf}(\exists x : (\forall x : p(x))) = p(1) \wedge p(2) \wedge \cdots$.

7. **Closed Formulas and a Logical Structure for KR-Logic.** We introduce the concept of closed formula. Based on this concept, a logical structure for KR-Logic is defined. A logical structure for LP-Logic is also defined as a sublogic of KR-Logic.

7.1. **Closed formulas and a logical structure for KR-Logic.** A formula E in WFF is a closed formula iff $mf(E)$ does not contain *undef*. The set of all closed formulas is denoted by $\mathcal{CF}$, i.e.,

$$\mathcal{CF} = \{E \mid (E \in \text{WFF}) \ \& \ (mf(E) \in \text{FORM}(\{true, false\} \cup \mathcal{G}_{\text{u}}))\}.$$

Let $\mathcal{K} = \mathcal{CF}$. Let $\nu : \mathcal{K} \to Map(\mathcal{I}, \{true, false\})$ be defined by: for any $E \in \mathcal{K}$ and any $I \in \mathcal{I}$,

$$\nu(E)(I) = value(subst(mf(E), I)).$$

A logical structure $\langle \mathcal{K}, \mathcal{I}, \nu \rangle$ is thus obtained.

7.2. **KR-Logic and LP-Logic.** In Sections 3.2-7.1, KR-Logic was defined by defining *ATOM*, $\mathcal{A}_{\text{u}}$, $\mathcal{G}_{\text{u}}$, WFF, $\mathcal{I}$, *val*, *mf*, *ev*, *e*, and RHO. LP-Logic is defined here similarly. Among these ten concepts, the three concepts, $\mathcal{G}_{\text{u}}$, $\mathcal{I}$, and *val*, are common to both KR-Logic and LP-Logic. The other seven concepts, i.e., *ATOM*, $\mathcal{A}_{\text{u}}$, WFF, *mf*, *ev*, *e*, and RHO, are different. We refer to these seven concepts for KR-Logic as $ATOM_{\text{KR}}$, $\mathcal{A}_{\text{KR}}$, WFF_{KR}, mf_{KR}, ev_{KR}, e_{KR}, and RHO_{KR}, and refer to those for LP-Logic as $ATOM_{\text{LP}}$, $\mathcal{A}_{\text{LP}}$, WFF_{LP}, mf_{LP}, ev_{LP}, e_{LP}, and RHO_{LP}.

LP-Logic is a sublogic of KR-Logic. Each of the seven concepts for LP-Logic is defined by removing, from the corresponding concept for KR-Logic, elements that contain a function variable, a function constant, or a *func*-atom. $ATOM_{\text{LP}}$ consists of user-defined atoms and built-in constraint atoms, and is a subset of $ATOM_{\text{KR}}$, i.e., *func*-atoms are not used in LP-Logic. WFF_{LP} is a subset of WFF_{KR}, i.e., *func*-atoms and quantifications of function variables are not used in LP-Logic. Accordingly, mf_{LP}, ev_{LP}, e_{LP}, and RHO_{LP} are restrictions/subsets of mf_{KR}, ev_{KR}, e_{KR}, and RHO_{KR}, respectively.

According to the method for constructing logical structures, $\mathcal{K}_{\text{KR}}$ and ν_{KR} are determined for KR-Logic and $\mathcal{K}_{\text{LP}}$ and ν_{LP} are determined for LP-Logic. As shown by Theorem 7.1 below, ν_{LP} is a restriction of ν_{KR}.

Theorem 7.1. *If E is a formula in $\mathcal{K}_{\text{LP}}$, then E is also a formula in $\mathcal{K}_{\text{KR}}$ and for any $I \in \mathcal{I}$, $\nu_{\text{LP}}(E)(I) = \nu_{\text{KR}}(E)(I)$.*

Proof: Let $E \in \mathcal{K}_{\text{LP}}$. Obviously, $E \in \mathcal{K}_{\text{KR}}$. For any $I \in \mathcal{I}$,

$$\nu_{\text{LP}}(E)(I) = value(subst(mf_{\text{LP}}(E), I)) = value(subst(mf_{\text{KR}}(E), I)) = \nu_{\text{KR}}(E)(I).$$

□

8. **Improving Solvability on KR-Logic.** On the formula space of KR-Logic, we state that KR-Logic can increase solvability of MI problems on first-order formulas.

8.1. **Concept of decomposition.** Conversion of a given formula into a conjunction of smaller formulas (such as clauses), is referred to as *formula decomposition* or, for short, *decomposition* [4]. The conventional decomposition in first-order logic, which is called the conventional Skolemization-based decomposition (CSD), involves removal of existential quantifications by Skolemization, i.e., by replacement of an existentially quantified variable with a Skolem term, which is determined by a relevant part of a formula prenex. CSD is commonly used in automated reasoning. CSD preserves satisfiability of a first-order formula, which is a necessary condition for the correctness of the conventional proof method with CSD and resolution.

8.2. **Non-preservation of satisfiability by CSD.** CSD does not generally preserve satisfiability, which can be proved by using the "tax-cut" problem in [21] (which is obtained by modification of a problem from [22]). Let $E = F_1 \wedge F_2 \wedge F_3 \wedge F_4$ and $Cs = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where

F_1: $\forall x : ((\exists y, \exists z : \mathit{hasChild}(x, y) \wedge \mathit{hasChild}(x, z) \wedge \mathit{noteq}(y, z)) \rightarrow \mathit{TaxCut}(x))$.
F_2: $\mathit{hasChild}(\mathit{Peter}, \mathit{Paul})$.
F_3: $\exists x : \mathit{hasChild}(\mathit{Peter}, x)$.
F_4: $\neg\exists x : \mathit{TaxCut}(x)$.
$C_1 = (\mathit{TaxCut}(X) \leftarrow \mathit{hasChild}(X, Y), \mathit{hasChild}(X, Z), \mathit{noteq}(Y, Z))$.
$C_2 = (\mathit{hasChild}(\mathit{Peter}, \mathit{Paul}) \leftarrow)$.
$C_3 = (\mathit{hasChild}(\mathit{Peter}, h) \leftarrow)$.
$C_4 = (\leftarrow \mathit{TaxCut}(X))$.

Then we have $\mathrm{CSD}(E) = \mathit{Cs}$. We compare $\mathit{Models}(E)$ and $\mathit{Models}(\mathit{Cs})$ as follows.

- $\mathit{Models}(E) \neq \{\}$: F_3 does not tell us that there is some person that is not equal to *Paul*. Hence $\mathit{TaxCut}(\mathit{Peter})$ cannot be inferred by F_1, and no contradiction occurs.
- $\mathit{Models}(\mathit{Cs}) = \{\}$: From C_2 and C_3, there are two children, *Paul* and h, of *Peter*. Since they are not equal, it follows that $\mathit{TaxCut}(\mathit{Peter})$ is true, which contradicts C_4.

This shows that CSD does not preserve models nor satisfiability. The existence of the *noteq* predicate causes this difficulty. Satisfiability preservation by CSD does not generally hold for formulas in $\mathrm{WFF}_{\mathrm{LP}}$ [4, 23].

8.3. **Preservation of satisfiability by MPD.** The basic idea of meaning-preserving Skolemization is to use existentially quantified function variables instead of function symbols. By meaning-preserving Skolemization or by MPD,

$$\mathrm{MPD}(\forall x, \exists y : p(x, y)) = \exists h, \forall x : p(x, h(x)),$$

where h is a function variable. Since $\exists h, \forall x : p(x, h(x))$ means that there is some mapping h such that $\forall x : p(x, h(x))$, it follows that $\forall x, \exists y : p(x, y)$ and $\exists h, \forall x : p(x, h(x))$ are equivalent. Hence, MPD preserves the meaning of $\forall x, \exists y : p(x, y)$. Since the Skolemization step is improved by meaning-preserving Skolemization, MPD generally preserves models [4].

8.4. **Failure by first-order logic.** In the following, assume that K is a formula in $\mathrm{WFF}_{\mathrm{LP}}$, and Cs and Cs' are formulas in $\mathcal{K}_{\mathrm{KR}}$. A transformation scheme for solving an MI problem $\langle K, \varphi\rangle$ typically consists of two phases.

1) K is converted by decomposition into Cs.
2) $\langle \mathit{Cs}, \varphi\rangle$ is transformed equivalently into $\langle \mathit{Cs}', \varphi\rangle$.

From $\langle \mathit{Cs}', \varphi\rangle$, the answer to the problem $\langle K, \varphi\rangle$ is determined by

$$\mathit{ans}(K, \varphi) = \varphi\left(\bigcap \mathit{Models}\,(\mathit{Cs}')\right).$$

If we take CSD in the first phase and $\mathrm{CSD}(K)$ is equal to Cs, then Cs is in $\mathcal{K}_{\mathrm{LP}}$. In the second phase, we can use usual transformations such as resolution or unfolding, and take a path within $\mathcal{K}_{\mathrm{LP}}$. However, since

$$\exists E \in \mathrm{WFF}_{\mathrm{LP}} : \mathit{Models}(\mathrm{CSD}(E)) \neq \mathit{Models}(E),$$

some problem on $\mathrm{WFF}_{\mathrm{LP}}$ cannot be solved correctly. Due to the non-preservation of models in the first phase, we know that solution methods with CSD are incomplete in LP-Logic. The conventional Skolemization does not provide a transformation process towards correct solutions to proof problems on $\mathrm{WFF}_{\mathrm{LP}}$.

8.5. **Improving solvability by extension of first-order logic.** If we take MPD in the first phase and $\mathrm{MPD}(K)$ is equal to Cs, then Cs is in $\mathcal{K}_{\mathrm{KR}}$. Since

$$\forall E \in \mathrm{WFF}_{\mathrm{LP}} : \mathit{Models}(\mathrm{MPD}(E)) = \mathit{Models}(E),$$

we can try to solve all problems on $\mathrm{WFF}_{\mathrm{LP}}$ by a search for a transformation path in the second phase. KR-Logic and MPD open the possibility of increasing solvability of MI problems on $\mathrm{WFF}_{\mathrm{LP}}$.

First-order logic has not enough formulas for making meaning-preserving decomposition possible, which gives many incomplete solutions, while KR-Logic has enough formulas for the existence of meaning-preserving decomposition.

9. **Conclusions.** We took a general concept of a logical structure. Despite its simplicity, this abstract notion provides a sufficient structure for defining the concepts of logical equivalence, models, satisfiability, and logical consequence. We proposed a schema for developing a new logic as a logical structure by defining well-formed formulas, forms, mappings for transformation/evaluation, closed formulas, interpretations, and models. According to the schema, we constructed KR-Logic together with LP-Logic.

KR-Logic is an extension of the conventional first-order logic with standard semantics and that with Herbrand semantics. KR-Logic provides, among other things, function constants and function variables, which enable us to introduce existential quantification of function variables. A transformation scheme for solving an MI problem $\langle K, \varphi\rangle$ typically consists of two phases: K is converted by decomposition into Cs. $\langle Cs, \varphi\rangle$ is transformed equivalently into $\langle Cs', \varphi\rangle$. If we take LP-Logic, solution methods with CSD are incomplete due to non-preservation of models (and satisfiability). If we take MPD in the first phase and $\mathrm{MPD}(K)$ is equal to Cs in $\mathcal{K}_{\mathrm{KR}}$, then the change form K to Cs preserves models. We can try to solve all problems on $\mathrm{WFF}_{\mathrm{LP}}$ by a search for a transformation path in the second phase. KR-Logic with MPD opens the possibility of improving solvability of MI problems on $\mathrm{WFF}_{\mathrm{LP}}$ by increasing ET rules.

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

[3] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.

[4] 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.

[5] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, in *Perspectives of Systems Informatics. PSI 2006. Lecture Notes in Computer Science*, I. Virbitskaite and A. Voronkov (eds.), Berlin, Heidelberg, Springer, 2007.

[6] K. Akama and E. Nantajeewarawat, Solving query-answering problems with constraints for function variables, *Proc. of the 10th Asian Conference on Intelligent Information and Database Systems*, Dong Hoi City, Vietnam, pp.36-47, 2018.

[7] 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.

[8] G. Frege, *On Formal Theories of Arithmetic*, Lecture at the July 17, 1885 meeting of Jena's Society for Medicine and Natural Science, in Jenaische Zeitschrift für Naturwissenschaft, 19/Supplement II, 1886.

[9] A. Tarski, The semantic conception of truth and the foundations of semantics, *Philosophy and Phenomenological Research*, vol.4, no.3, pp.341-376, 1944.
[10] G. Frege, *Basic Laws of Arithmetic*, Jena, H. Pohle, 1893.
[11] A. N. Whitehead and B. Russell, *Principia Mathematica*, vol.1, 2nd Edition, Cambridge University Press, Cambridge, 1925.
[12] A. N. Whitehead and B. Russell, *Principia Mathematica*, vols.2&3, 2nd Edition, Cambridge University Press, Cambridge, 1927.
[13] 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.
[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*, Porto, Portugal, pp.52-63, 2016.
[15] K. Akama and E. Nantajeewarawat, Solving proof problems with equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.2, pp.361-376, 2022.
[16] 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.
[17] K. Akama, E. Nantajeewarawat and T. Akama, Logical problem solving framework, *Proc. of the 11th Asian Conference on Intelligent Information and Database Systems*, Yogyakarta, Indonesia, pp.28-40, 2019.
[18] 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*, Seville, Spain, pp.237-246, 2018.
[19] 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, pp.96-103, 2016.
[20] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[21] K. Akama and E. Nantajeewarawat, Proving theorems based on equivalent transformation using resolution and factoring, *Proc. of the 2nd World Congress on Information and Communication Technologies*, Trivandrum, India, pp.7-12, 2012.
[22] B. Motik, U. Sattler and R. Studer, Query answering for OWL-DL with rules, *Journal of Web Semantics*, vol.3, pp.41-60, 2005.
[23] K. Akama and E. Nantajeewarawat, Function-variable elimination and its limitations, *Proc. of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Lisbon, Portugal, pp.212-222, 2015.

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.