Constructing Knowledge Representation Systems with First-Order Formulas as Atoms

Kiyoshi Akama
Information Initiative Center, Hokkaido University, Hokkaido, Japan
Email: akama@iic.hokudai.ac.jp

Ekawit Nantajeewarawat
Computer Science, Sirindhorn International Institute of Technology, Thammasat University, Thailand
Email: ekawit@siit.tu.ac.th

Tadayuki Yoshida
Faculty of Computer Science, Hokkaido University, Hokkaido, Japan
Email: tadayuki@sh.rim.or.jp

***Abstract*—This paper proposes a knowledge representation system by extension of the concept of atom. Not only conventional simple atoms, but also atoms representing first-order formulas, which are called formula atoms, are used. By this extension, predicates, logical connectives, and quantifiers may occur in atoms, and can be regarded as objects in the same class, allowing more natural translation from natural language sentences into extended formulas and more flexible computation for solving logical problems.**

***Index Terms*—formula atom, constraint, knowledge representation system, declarative description, query-answering problem**

I. Introduction

A query-answering problem (QA problem) is a pair $\langle q, K\rangle$, where q is a query atom and K is a logical formula describing background knowledge. The problem is concerned with finding all ground instances of the query atom q that are logical consequences of the background knowledge K. Using definite clauses to represent background knowledge, QA problems are investigated extensively in logic programming [1]. So far, many subclasses of QA problems have been discussed. Recently wide attention has been given to QA problems whose background knowledge is a combination of description logic axioms/assertions and clauses [2]-[4]. However, these are rather small subclasses of QA problems. QA problems on full first-order logic with built-in constraints (for short, QA problems on FOL_B), which constitute a far larger class of QA problems, have not been investigated. One of our long term objectives is to develop a general method for solving QA problems in this general class.

In [5], we showed that proof problems can be regarded as a subclass of QA problems. As they include proof problems, QA problems on FOL_B form one of the most basic and fundamental classes of problems for the research of human intelligent behaviors. In the conventional proof theory, a first-order formula is converted into a clause set using the conventional Skolemization [6], and new clauses are inferred from existing clauses using the resolution rule. This method, however, does not work well with a larger class of QA problems. For example, as illustrated in [7], the conventional Skolemization and resolution may give incorrect results for solving QA problem on FOL_B.

To solve QA problems on FOL_B, we have extended first-order logic with function variables in [8]. This extension enables us to equivalently convert first-order formulas into extended clauses. The conversion process is called meaning-preserving Skolemization.

To extend first-order logic, we need a general theory of logical structures [9], [10], which allows us to invent a new logic systematically. The concept of model in the conventional logics is too specific and restricted. It depends on concepts of predicates, terms, and variables. In the theory of logical structures [9], [10], a model is a subset of some predetermined set $\mathcal{G}$, which is independent of predicates, terms, and variables. A declarative description determines a set of models in a logical structure.

Based on the equivalent transformation (ET) principle, we invented the solution method for QA problems [11], which can solve a far larger class of QA problems compared to definite-clause-based QA problems and description-logic-based QA Problems. Moreover, the ET-based method provides more flexible solution paths than resolution-based methods and tableau-based methods for description logics.

In this paper, we aim to extend the concept of atom itself by allowing it to have other atoms as arguments. For example, we introduce an atom such as $(say\ john\ (say\ mary\ hello))$, which has as its arguments the atom $(say\ mary\ hello)$. This cannot be represented as an atomic first-order formula. The predicate *say* above

Manuscript received July 1, 2014; revised November 1, 2014.

doi: 10.12720/jiii.3.3.238-245

relates the term *john* and the atom $(say\,mary\,hello)$. We also introduce atoms such as $(and\ (boy\ john)\ (girl\ mary))$ and $(\forall x\ (imply\ (dog\ x)\ (animal\ x)))$ to represent first-order formulas. The symbol '*and*' represents logical conjunction ($\wedge$) and the symbol '$\forall$' represents the universal quantifier ($\forall$). They appear in the predicate position in an atom, and they take atoms as arguments.

By the introduction of atoms in argument positions and that of formulas in atom positions, logical connectives and quantifiers inside formula atoms become more similar to predicates, since formula atoms may contain them at predicate positions. If we can extend the conventional theory to allow such enriched formulas, then more natural translation from a natural language into formulas is possible. Moreover, the concept of computation, which is regarded as transformation of formulas, is also extended and becomes more flexible.

The rest of the paper is organized as follows: Section II formulates S-expressions, based on which formula atoms are defined in Section III. Section IV introduces simple constraints, referential constraints, and *func*-constraints. After defining extended formulas in Section V, Section VI formulates declarative descriptions and Section VII establishes their semantics. Section VIII introduces two basic classes of formulas, i.e., clauses and if-and-only-if formulas. Section IX presents the construction of a knowledge representation system. Section X demonstrates transformation rules and computation using them. Section XI concludes the paper.

The following notation is used: For any set *A*, *pow*(*A*) denotes the power set of *A*. *Bool* denotes the set $\{true, false\}$.

II. S-Expressions

Let Σ be a set of symbols such that $nil \in \Sigma$. An $S-expression$ ($symbolic\,expression$) on Σ is defined inductively as follows:

1) An element of Σ is an S-expression on Σ.
2) If a and a' are S-expressions on Σ, then $(a \mid a')$ is an S-expression on Σ.

An S-expression $(a_1 \mid (a_2 \mid (\cdots \mid (a_n \mid nil)\cdots)))$ is often written as $(a_1\, a_2 \cdots a_n)$. The S-expression *nil* is often written as ( ). The set of all S-expressions on Σ is denoted by $\boldsymbol{S}(\Sigma)$.

An alphabet $\Delta = \langle \mathbf{K}, \mathbf{V}, \mathbf{FV}, \mathbf{L} \rangle$ is assumed, where (i) **K**, **V**, **FV**, and **L** are countably infinite sets of constants, variables, function variables, and labels, respectively, (ii) these four sets are mutually disjoint, and (iii) $nil \in K$. Each variable in **V** is called an *AT-variable* (*atom/term-variable*), and each function variable in **FV** is called an *F-variable*. In the rest of this paper, the term "variable" alone means an AT-variable or an F-variable.

Let $FC(n)$ be the set of all mappings from $S(K)n$ to $S(K)$. Let $FC = \cup\{FC(i) \mid i\ is\ a\ nonnegative\ integer\}$. An element of FC is called a *function constant*.

III. Formula Atoms

A *formula atom* is defined inductively as follows:

1) If $p \in K$ and $t_1, \ldots, t_n$ are S-expressions in $S(K \cup V \cup FC \cup FV)$, then the S-expression $(p\, t_1 \cdots t_n)$ is a formula atom.
2) If *a* is a formula atom, then the S-expression (*not a*) is a formula atom.
3) If *a* and *b* are formula atoms, then the S-expressions (*and a b*), (*or a b*), and (*imply a b*) are formula atoms.
4) If *x* is an AT-variable in **V** and *a* is a formula atom, then the S-expressions (*A x a*) and (*E x a*) are formula atoms.
5) If *h* is an F-variable in **FV** and *a* is a formula atom, then the S-expressions $(A_f\, h\, a)$ and $(E_f\, h\, a)$ are formula atoms.

The symbols '*A*', '*E*', 'A_f', and 'E_f' in the last two conditions are called *inside-atom* quantifiers.

Let $\mathcal{A}$ denote the set of all formula atoms. A substitution on **V** as well as a substitution on **FV** determines a total mapping on $\mathcal{A}$.

For any formula atom $a \in A$, let $freeV(a)$ denote the set of all free AT-variables occurring in *a*, and $freeFV(a)$ denote the set of all free F-variables occurring in *a*. A formula atom containing no free variable is called a *ground* formula atom, i.e., a formula atom *a* is ground iff $freeV(a) = \varnothing\ and\ freeFV(a) = \varnothing$. Let $\mathcal{G}$ denote the set of all ground formula atoms.

Example 1: Assume that *isChild*, *say*, *append*, and *eq* are predicate symbols in **K**, *john*, *mary*, and *hello* are constants in **K**, and *w*, *x*, *y*, *z*, *X*, and *Z* are AT-variables in **V**. Then the following four S-expressions are formula atoms (1):

$$\begin{array}{l}(isChild\ john\ mary)\\ (say\ john\ (say\ mary\ hello))\\ (append\ x\ y\ z)\\ (E\,w\ (E\,X\ (E\,Z\ (and\ (eq\ x\ (w \mid X))\\ \qquad\qquad (and\ (eq\ x\ (w \mid Z))\\ \qquad\qquad\quad (append\ X\ y\ Z))))))\end{array} \tag{1}$$

IV. Constraints

Let $G_F = FC$, and let $\mathbf{CM}(m)$ denote the set of all partial mappings from

$$(S(K) \cup G \cup pow(S(K)) \cup pow(G) \cup G_F)^m$$

to *Bool*. Constraints, simple constraints, referential constraints, and *func*-constraints are defined below.

1) Assume that
   - $\Phi \in CM(m)$, where $m \geq 1$, and
   - Each of $t_1, \ldots, t_m$ is an S-expression in $S(K \cup V)$ or a label in **L**.

Then $\langle \phi, t_1, \ldots, t_m \rangle$ is a *constraint*, which is called either a simple constraint or a referential constraint as follows:

- If none of $t_1, \ldots, t_m$ is a label in **L**, then it is called a *simple constraint*.
- If at least one of $t_1, \ldots, t_m$ is a label in **L**, then it is called a *referential constraint*.

2) Assume that
- h is an n-ary F-variable in **FV** or a function constant in **FC**, where $n \geq 0$,
- $t_1, \ldots, t_n, t_{n+1}$ are S-expressions in $\boldsymbol{S}(\mathbf{K} \cup \mathbf{V})$, and
- ϕ_{func} is a mapping such that if h is a function constant in **FC** and $t_1, \ldots, t_n, t_{n+1}$ are S-expressions in $\boldsymbol{S}(\mathbf{K})$, then $\phi_{func}(h, t_1, \ldots, t_n, t_{n+1}) = true$ iff $h(t_1, \ldots, t_n) = t_{n+1}$.

  Then $\langle \phi_{func}, h, t_1, \ldots, t_n, t_{n+1} \rangle$ is a *constraint*, which is called a *func-constraint*.

A constraint is called a ground constraint if it contains no free variable in $V \cup FV$.

Example 2: Let $\Phi eq \in CM(2)$ such that for any S-expressions $t_1, t_2 \in \boldsymbol{S}(\mathbf{K})$, $\phi_{eq}(t_1, t_2) = true$ iff $t_1 = t_2$. Then $\langle \phi_{eq}, x, y \rangle$ and $\langle \Phi eq, 1, 2 \rangle$ are simple constraints. The truth value of a simple constraint is determined when it is ground. For example, $\langle \Phi eq, 1, 2 \rangle$ is false and $\langle \Phi eq, 4, 4 \rangle$ is true.

Example 3: Let $\Phi not \in \mathbf{CM}(2)$ such that for any formula atom $g \in \mathcal{G}$ and any subset G of $\mathcal{G}$, $\phi_{not}(g, G) = true$ iff $g \notin G$. Then $\langle \phi_{not}, (p\ 2), l_0 \rangle$ is a referential constraint. The truth value of a referential constraint is determined if it is ground and a model corresponding to each label appearing in the constraint is given. For instance, $\langle \phi_{not}, (p\ 2), l_0 \rangle$ is true if G is a model corresponding to the label l_0 and $(p\ 2) \notin G$.

Example 4: For meaning-preserving Skolemization devised in [8], *func*-constraints are used. For example, the first-order formula

$$\exists x\text{: } (hasChild(Peter, x) \wedge (\exists y\text{: } motherOf(x, y)))$$

is converted by meaning-preserving Skolemization into the clause set $\{C_1, C_2\}$ given by:

C_1: $(hasChild\ Peter\ {*x}) \leftarrow \langle \phi_{func}, {*h_1}, {*x} \rangle$

C_2: $(motherOf\ {*x}\ {*y}) \leftarrow \langle \phi_{func}, {*h_1}, {*x} \rangle, \langle \phi_{func}, {*h_2}, {*y} \rangle$

As shown above, variables with the prefix '*' are often used in a clause. By contrast, variables without the prefix '*' are often used inside a formula atom. Variables with the prefix '*' and those without it both belong to **V**.

V. Extended Formulas

An *extended formula* (for short, *formula*), is defined inductively as follows:

1) Each formula atom is a formula.
2) Each constraint is a formula.
3) If α is a formula, then so is $\neg \alpha$.
4) If α and β are formulas, then so are $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, and $\alpha \leftrightarrow \beta$.
5) If x is an AT-variable in **V** and α is a formula, then $\forall x$: α and $\exists x$: α are formulas.
6) If h is an F-variable in **FV** and α is a formula, then $\forall_{\mathbf{f}} h$: α and $\exists_{\mathbf{f}} h$: α are formulas.

The quantifiers $\forall$, $\exists$, $\forall_f$, and $\exists_f$ in the last two conditions are called *formula-level* quantifiers.

Given a formula α, the following notation is introduced:

- $V(\alpha)$ is the set of all AT-variables occurring in α.
- $FV(\alpha)$ is the set of all F-variables occurring in α.
- $BV_1(\alpha)$ is the set of all AT-variables that are bound by inside-atom quantifiers in α.
- $BV_2(\alpha)$ is the set of all AT-variables that are bound by formula-level quantifiers in α.
- $BFV_1(\alpha)$ is the set of all F-variables that are bound by inside-atom quantifiers in α.
- $BFV_2(\alpha)$ is the set of all F-variables that are bound by formula-level quantifiers in α.
- $freeV(\alpha) = V(\alpha) - (BV_1(\alpha) \cup BV_2(\alpha))$.
- $freeFV(\alpha) = FV(\alpha) - (BFV_1(\alpha) \cup BFV_2(\alpha))$.

A *ground* formula is a formula that contains no free variable, i.e., a formula is ground iff $freeV(\alpha) = \varnothing$ *and* $freeFV(\alpha) = \varnothing$. For example, the formula

$$\forall x\text{: } (\langle \phi_{func}, h, x \rangle \rightarrow (hasChild\ Peter\ x))$$

is not ground (since h occurs in it as a free variable), while the formula

$$\exists_{\mathbf{f}} h\text{: } (\forall x\text{: } (\langle \phi_{func}, h, x \rangle \rightarrow (hasChild\ Peter\ x)))$$

is ground.

VI. Declarative Descriptions

A declarative description D contains sets of formulas, each of which may contain constraints. A constraint may in turn contain labels, which refer to some other worlds in the description D. More precisely, a *declarative description* D is a set of pairs of labels and formulas, i.e.(2),

$$D = \{(l_0\text{: } Ds_0), (l_1\text{: } Ds_1), \ldots, (l_n\text{: } Ds_n)\}, \qquad (2)$$

where each of the l_i is a label and each of the D_{s_i} is a formula. For any $i \in \{0, 1, \ldots, n\}$, l_i is called a *world identifier* and D_{Si} a *world description*.

The semantics of a declarative description D, which will be given in Section VII, is outlined below. Let D be a declarative description $\{(l_0\text{: } Ds_0), (l_1\text{: } Ds_1), \ldots, (l_n\text{: } Ds_n)\}$. Assume that for each $i \in \{0, 1, \ldots, n\}$, G_i is the intended model, which is a set of ground formula atoms, of D_{Si}. Then the meaning of each referential constraint in D_{Si} is determined with reference to D_{Si}. Accordingly, D_{Si} which contains labels, determines a set M_{Si} of possible models. Then G_i should satisfy the condition $G_i \in M_{Si}$. Consequently, G_i should satisfy the system of constraints

given in Section VII-C, where a mapping $model_i$ is defined by: $Ms_i = model_i(G_0, G_1, \ldots, G_n)$. By the obtained system of constraints, all the n+1-tuples of $[G_0, G_1, \ldots, G_n]$ are determined, and the first components of all these tuples collectively constitute the set of all models of the declarative description D.

Formulas of two specific forms, i.e., clauses and if-and-only-if-formulas, will be introduced in Section VIII. They are important for rich representation and efficient computation for a declarative description. Consider, for example, a declarative description $D = \{(l_0 : D_{S0})\}$. D_{S0} may be a closed formula of the form $\forall_F h_1 \cdots \forall_F h_N : E$, where E is divided into a set of iff-formulas E_1 and another set E_2 of formulas, and E_2 is converted by meaning-preserving transformation into a clause set C_S, possibly containing in clause bodies referential constraints with a label l_0, which refers to the meaning of D_{S0}. Computation using iff-formulas and clauses will be shown in Section X.

VII. Semantics

A. Interpretations

In the following, let D be a declarative description $\{(l_0: Ds_0), (l_1: Ds_1), \ldots, (l_n: Ds_n)\}$. An *interpretation* is a subset of $\mathcal{G}$. A *model* of D is an interpretation that "satisfies" D. The objective of this section is to determine the set of all models of D, by making clear what "satisfies" above means.

B. Truth Values of Ground Formulas

Assume that for each $i \in \{0, 1, \ldots, n\}$, a set G_i of ground formula atoms that corresponds to a label l_i is given. Then, for any interpretation I, the truth value of a ground formula under I is defined as follows:

1) A ground formula atom g is true under I iff $g \in I$.
2) A ground constraint $\langle \phi, d_1, \ldots, d_m \rangle$ is true under I iff $\phi(d'_1, \ldots, d'_m) = true$, where for each i $\{1, \ldots, m\}$,
   - if $d_i \in L$, then $d'_i = G_i$;
   - if $d_i \notin L$, then $d'_i = d_i$.
3) A ground *func*-constraint $\langle \phi_{func}, h, t_1, \ldots, t_m, t_{m+1} \rangle$ is true under I iff $\phi_{func}(h, t_1, \ldots, t_m, t_{m+1}) = true$.
4) For any ground formula α, $\neg\alpha$ is true under I iff α is false under I.
5) For any ground formulas α and β, $\alpha \wedge \beta$ (respectively, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, and $\alpha \leftrightarrow \beta$) is true under I iff α and β are true (respectively, at least one of α and β is true, at least one of $\neg\alpha$ and β is true, and α and β have the same truth value) under I.
6) A ground formula $\forall x: E$, where x is an AT-variable in **V**, is true under I iff for any S-expression $t \in S(K)$, $E\{x/t\}$ is true under I.
7) A ground formula $\exists x: E$, where x is an AT-variable in **V**, is true under I iff there exists at least one S-expression $t \in S(K)$ such that $E\{x/t\}$ is true under I.
8) A ground formula $\forall_{\mathbf{f}} h: E$, where h is an F-variable in **FV**, is true under I iff for any function constant f in **FC**, $E\{h/f\}$ is true under I.
9) A ground formula $\exists_{\mathbf{f}} h: E$, where h is an F-variable in **FV**, is true under I iff there exists at least one function constant f in **FC** such that $E\{h/f\}$ is true under I.

An interpretation I is a *model* of a closed formula α iff α is true under I.

C. A System of Membership Constraints and Models of a Declarative Description

Consider $i \in \{0, 1, \ldots, n\}$. If no label occurs in it, D_{Si} determines the set of all models, which is denoted by $model(l_i)$. However, in general, labels may occur in D_{Si}. In the presence of labels, when models $G_0, G_1, \ldots, G_n$ of $Ds_0, Ds_1, \ldots, Ds_n$, respectively, are known, the set of all models of D_{Si} can be determined uniquely and this set is denoted by $model(l_i, G_0, G_1, \ldots, G_n)$.

By its definition, G_i is an element of $model(l_i, G_0, G_1, \ldots, G_n)$ for each $i \in \{0, 1, \ldots, n\}$. Accordingly, we have a system of membership constraints, which is denoted by SMC(D), as (3)

$$\begin{aligned} x_0 &\in model(l_0, x_0, x_1, \ldots, x_n) \\ x_1 &\in model(l_1, x_0, x_1, \ldots, x_n) \\ x_2 &\in model(l_2, x_0, x_1, \ldots, x_n) \\ &\cdots \\ x_n &\in model(l_n, x_0, x_1, \ldots, x_n) \end{aligned} \tag{3}$$

Let D be a declarative description. $[G_0, G_1, \ldots, G_n]$ is called an *extended model* of D iff the set of equations $\{(x_0 = G_0), (x_1 = G_1), \ldots, (x_n = G_n)\}$ satisfies SMC (D). G_0 is a *model* of D iff there exist $G_1, \ldots, G_n$ such that $[G_0, G_1, \ldots, G_n]$ is an *extended model* of D. The set of all models of D is denoted by *Models*(D).

VIII. Clauses and If-and-Only-If Formulas

In general, any formula can be used for defining D_{Si}. However, some specific classes of formulas are commonly known to be useful. Two classes of formulas, i.e., clauses and if-and-only-if formulas, are introduced in this section.

A. Clauses ∈

A *clause* C is an expression of the form (4)

$$a_1, \ldots, a_k \leftarrow b_1, \ldots, b_n \tag{4}$$

where (i) k, $n \geq 0$, (ii) each of the a_i is a formula atom, and (iii) each of the b_j is a formula atom or a constraint. The meaning of the clause C is given by the formula (5)

$$\forall((b_1 \wedge \cdots \wedge b_n) \rightarrow (a_1 \vee \cdots \vee a_k)), \tag{5}$$

where $\forall$ denotes universal quantifications for all free AT-variables.

Example 5: The clauses C_1 and C_2 in Example 4 together represent the knowledge that Peter has a child who is someone's mother.

B. If-and-Only-If Formulas

An *if-and-only-if formula* (for short, *iff-formula*) I is a formula of the form (6)

$$a \leftrightarrow (conj_1 \vee \cdots \vee conj_n) \tag{6}$$

where (i) $n \geq 0$, (ii) a is a formula atom, and (iii) each of the $conj_i$ is a finite subset of formula atoms and/or constraints. When $n \leq 1$, the pair of braces on the right-hand side is often omitted. For each $i \in \{1, \ldots, n\}$, $conj_i$ corresponds to the existentially quantified atom conjunction FOL($conj_i$, a) given by (7)

$$\text{FOL}(conj_i, a) = \exists y_1 \ldots \exists y_k \colon \wedge\{b \mid b \in conj_i\}, \tag{7}$$

where $y_1, \ldots, y_k$ are all the variables that occur in $conj_i$ but do not occur in a. The iff-formula I corresponds to the universally quantified formula(8)

$$\forall(a \leftrightarrow (\text{FOL}(conj_1, a) \vee \ldots \vee \text{FOL}(conj_n, a))), \tag{8}$$

which is denoted by FOL(I).

Example 6: What it means for a binary relation to be symmetric can be defined by the iff-formula(9) below.

$$(sym\ {*r}) \leftrightarrow \{(A\ x\ (A\ y\ (imply\ ({*r}\ x\ y)\ ({*r}\ y\ x))))\} \tag{9}$$

The formula atom on its right-hand side specifies a necessary and sufficient condition for a relation $*r$ to be symmetric, i.e., whenever $*r$ contains a pair $\langle x, y\rangle$, it must also contain the pair $\langle y, x\rangle$.

C. A Formula Attached to a Label

A formula D_{Si} attached to a label I_i is often a closed conjunction E of clauses and iff-formulas. Such a conjunction E can be represented as a set of clauses and iff-formulas. These clauses and iff-formulas may include function variables $h_1, \ldots, h_n$, which make the clause set appear to be an open formula. Semantically, these function variables are globally existentially quantified. As such, D_{Si} is a closed formula $\exists_{\mathbf{f}} h_1 \ldots {}_{\mathbf{f}} h_n \colon E$, where all function variables occurring in E are existentially quantified by

IX. Constructing a Knowledge Representation System

We show how to define built-in constraints in Section IX-A. The meanings of formula atoms are defined in Section IX-B. Knowledge representation for QA problems based on built-in constraints and formula atoms is described in Sections IX-C–IX-E.

A. The Meanings of Basic Built-In Atoms

An *eq*-atom is defined by the iff-formula (10)

$$(eq\ {*X_1}\ {*X_2}) \leftrightarrow \langle \phi_{eq}, {*X_1}, {*X_2}\rangle \tag{10}$$

where $\langle \phi_{eq}, *X_1, *X_2\rangle$ is a constraint defined in Example 2 (Section IV). This is a typical form of the definition of a built-in atom. Other examples of built-in atom definitions are (11):

$$\begin{aligned}(< {*X_1}\ {*X_2}) &\leftrightarrow \langle \phi_{lt}, {*X_1}, {*X_2}\rangle \\ (+ {*X_1}\ {*X_2}\ {*X_3}) &\leftrightarrow \langle \phi_{sum}, {*X_1}, {*X_2}, {*X_3}\rangle,\end{aligned} \tag{11}$$

where (i) $\langle \phi_{lt}, t_1, t_2 = true$ iff t_1 and t_2 are numbers such that $t_1 < t_2$, and (ii) $\langle \phi_{sum}, t_1, t_2, t_3\rangle = true$ iff t_1, t_2, and t_3 are numbers such that $t_1 + t_2 = t_3$.

B. The Meanings of Formula Atoms

Assume that only one world with the label l_0 is considered. To introduce the logical connective *not*, we define it by (12)

$$(not * X) \leftrightarrow (\phi_{not} * X\, l_0) \tag{12}$$

where $*X$ is a $(not * X) \leftrightarrow (\phi_{not} * X\, l_0)$ variable in **V** and for any $g \in \mathcal{G}$ and any $G \subseteq \mathcal{G}$, $\phi_{not}(g, G)$ is true iff $g \notin G$.

Similarly, we define the meanings of *and*, *or*, and *imply* by

- $(and\ {*X}\ {*Y}) \leftrightarrow \langle \phi_{and}, {*X}, {*Y}, l_0\rangle$,
- $(or\ {*X}\ {*Y}) \leftrightarrow \langle \phi_{or}, {*X}, {*Y}, l_0\rangle$,
- $(imply\ {*X}\ {*Y}) \leftrightarrow \langle \phi_{imply}, {*X}, {*Y}, l_0\rangle$,

where $*X$ and $*Y$ are variables in **V** and for any $g, g' \in \mathcal{G}$ and any $G \subseteq \mathcal{G}$,

- $\phi_{and}(g, g', G)$ is true iff $g \in G$ and $g' \in G$,
- $\phi_{or}(g, g', G)$ is true iff $g \in G$ or $g' \in G$,
- $\phi_{imply}(g, g', G)$ is true iff $g \notin G$ or $g' \in G$.

The meanings of A and E are defined as follows: For any variable x in **V** and any formula atom X' that contains no free variable other than x and contains no free function variable,

- $(AxX') \leftrightarrow \langle \phi_{any}, (AxX')l_0\rangle$,
- $(ExX') \leftrightarrow \langle \phi_{exists}, (ExX')l_0\rangle$,

where for any $g \in \mathcal{G}$ and any $G \subseteq \mathcal{G}$,

- $\phi_{any}(g, (A\ x\ X'), G)$ is true iff
$$\{X'\theta \mid (t \in \mathcal{G})\ \&\ (\theta = \{x/t\})\} \subseteq G,$$
- $\phi_{exists}(g, (E\ x\ X'), G)$ is true iff
$$\{X'\theta \mid (t \in \mathcal{G})\ \&\ (\theta = \{x/t\})\} \cap G \neq \varnothing.$$

The meanings of $A_{\mathbf{f}}$ and $E_{\mathbf{f}}$ are defined below. For any function variable h and any formula atom X'' that contains no free function variable other than h and contains no free variable,

- $(A_{\mathbf{f}}\ h\ X'') \leftrightarrow \langle \phi_{any\text{-}\mathbf{f}}, (A_{\mathbf{f}}\ h\ X''), l_0\rangle$
- $(E_{\mathbf{f}}\ h\ X'') \leftrightarrow \langle \phi_{exists\text{-}\mathbf{f}}, (E_{\mathbf{f}}\ h\ X''), l_0\rangle$

where for any $g \in \mathcal{G}$ and any $G \subseteq \mathcal{G}$,

- $\phi_{any\text{-}\mathbf{f}}(g, (A_{\mathbf{f}}\ h\ X''), G)$ is true iff
$$\{X''\theta \mid (t \in \mathcal{G}_{\mathrm{F}})\ \&\ (\theta = \{h/t\})\} \subseteq G,$$
- $\phi_{exists\text{-}\mathbf{f}}(g, (E_{\mathbf{f}}\ h\ X''), G)$ is true iff
$$\{X''\theta \mid (t \in \mathcal{G}_{\mathrm{F}})\ \&\ (\theta = \{h/t\})\} \cap G \neq \varnothing.$$

C. Modeling Using a Declarative Description

We consider a declarative description $D = \{l_0: Ds_0\}$, where $D_{S0} = DEF_{B\cup}DEF_F \cup DEF_U$ such that

- DEF_B consists of iff-formulas defining basic built-in constraints,
- DEF_F consists of iff-formulas defining formula atoms, and
- DEF_U consists of iff-formulas and clauses representing background knowledge.

A user can construct a declarative description by defining DEF_B and DEF_F as in Sections IX-A and IX-B, and writing iff-formulas and clauses based on constraints in DEF_B and formula atoms defined by DEF_F. For example, to define the inclusion relation, DEF_U may contain the iff-formulas below: (14), (15)

$$(subset{*}X\,{*}Y) \leftrightarrow (Ae\ (imply\ (eleme{*}X)\ (eleme{*}Y))) \quad (14)$$

$$(elem{*}e\ (*a\,|\,{*}X)) \leftrightarrow (or\ (eq{*}e{*}a)\ (elem{*}e{*}X)) \quad (15)$$

where the meaning of *eq* is defined by DEF_B, and the meanings of *A*, *imply*, and *or* are defined by DEF_F.

D. Query-Answering Problems

A *query-answering problem* (for short, *QA problem*) on a declarative description D is a pair $\langle q, D\rangle$, where q is a formula atom. The answer of this problem is the set $rep(q) \cap (\cap Models(D))$, where $rep(q)$ is the set of all ground instances of q and $Models(D)$ is the set of all models of D. For example, if q = (*subset* (2 3) (4 2 1 3)), $D = \{(l_0 : D_{S0})\}$, $D_{S0} = DEF_{B\cup}DEF_F \cup DEF_U$, and DEF_U consists of the iff-formulas defining *subset* and *elem* given in Section IX-C, then $\langle q, D\rangle$ is a QA problem with its answer being {(*subset* (2 3) (4 2 1 3))}.

E. Construction of Knowledge Representation Systems

In this paper, a knowledge representation system on $\mathcal{G}$ is a mapping from some set X to $pow(pow(\mathcal{G}))$. Each time DEF_B and DEF_F are determined, a mapping (16)

$$\mathbf{m}: X \rightarrow pow(pow(\mathcal{G})) \quad (16)$$

where X is the set of all possible DEF_U, is obtained such that for any $x \in X$, $\mathbf{m}(x) = Models(D)$, where $D = \{(l_0 : D_{S0})\}$ and $D_{S0} = DEF_B \cup DEF_F \cup x$. This means that we can obtain a knowledge representation system on $\mathcal{G}$ by defining the following components: (i) a set $\mathcal{G}$, (ii) constraints as given in Section IV, (iii) basic built-in atoms given by DEF_B, and (iv) formula atoms given by DEF_F.

X. Transformation Rules and Computation

In the proposed knowledge representation system, QA problems are solved by using many equivalent transformation rules (ET rules). Some ET rules are explained in Section X-A, and an example of computation using ET rules is shown in Section X-B.

A. ET Rules

Some ET rules are general rules, such as unfolding, forwarding, resolution, side-change transformation, etc. Specialized ET rules are also often used. An important class of specialized ET rules is designed to transform a set of atoms in the right-hand side of a clause. Examples of rules in this class are:

r_1: $(and\ {*}X\ {*}Y) \Rightarrow {*}X, {*}Y$

r_2: $(E\ {*}x\ {*}F) \Rightarrow \{(construct\ {*}x\ {*}F\ {*}F')\}, {*}F'$

where the built-in atom $(construct{*}x{*}F\,{*}F')$ constructs ${*}F'$ from ${*}x$ and ${*}F$ by replacing each occurrence of ${*}x$ in ${*}F$ with a new variable.

B. Computation by Using ET Rules

Consider the Oedipus problem described in [12]. 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. The problem is to find a person who has a patricide child who has a non-patricide child. Assume that "*oe*", "*io*", "*po*", and "*th*" stand, respectively, for Oedipus, Iokaste, Polyneikes, and Thersandros.

This problem is represented as a QA problem with the query atom $(prob{*}x)$ and the background knowledge consisting of the following clauses:

C_1: $(prob{*}x) \leftarrow (E\ y\ (and\ (isChild\ y{*}x)\ (and\ (paty)\ (E\ z\ (and\ (isChild\ z\ y)\ (not\ (pat\ z)))))))$

C_2: $(isChild\ oe\ io) \leftarrow$ C_3: $(isChild\ po\ io) \leftarrow$

C_4: $(isChild\ po\ oe) \leftarrow$ C_5: $(isChild\ th\ po) \leftarrow$

C_6: $(pat\ oe) \leftarrow$ C_7: $\leftarrow (pat\ th)$

Refer to the rules r_1 and r_2 in Section X-A. By applying the rule r_2 to C_1, a new variable, say $*y$, is introduced and C_1 is transformed into:

C_8: $(prob\ *x) \leftarrow (and\ (isChild\ *y\ *x)\ (and\ (pat\ *y)\ (E\ z\ (and\ (isChild\ z\ *y)\ (not\ (pat\ z))))))$

The body of C_8 consists only of one formula atom, with *and* appearing in the predicate position and with two arguments, i.e., an *isChild*-atom and another *and*-atom. By the application of the rule r_1, this body formula atom is split into two formula atoms, resulting in the clause:

C_9: $(prob\ *x) \leftarrow (isChild\ *y\ *x),\ (and\ (pat{*}y)\ (E\ z\ (and\ (isChild\ z{*}y)\ (not\ (pat\ z)))))$

Again the *and*-atom in the body of C_9 is selected. By applying the rule r_1 to it, C_9 is transformed into the clause C_{10} below, with three formula atoms in its body.

C_{10}: $(prob\ *x) \leftarrow (isChild\ *y\ *x),\ (pat{*}y),\ (E\ z\ (and\ (isChild\ z{*}y)\ (not\ (pat\ z))))$

Next, the rule r_2 is applied. The application introduces a new variable *z and transforms C_{10} into:

$$C_{11}: \quad (prob * x) \leftarrow (isChild * y * x), (pat * y), (and\ (isChild * z * y)\ (not\ (pat * z)))$$

Further application of the rule r_1 to C_{11} yields:

$$C_{12}: \quad (prob * x) \leftarrow (isChild * y * x), (pat\ * y), (isChild\ * z\ * y), (not\ (pat\ * z))$$

By side change transformation [11] of the a *not*-atom, C_{12} is transformed into the multi-head clause below.

$$C_{13}: \quad (pat\ * z), (prob\ * x) \leftarrow (isChild\ * y\ * x), (pat * y), (isChild * z * y).$$

C_{13} is a conventional clause consisting of only simple atoms. By unfolding with respect to *isChild* (two times), C_{13} is transformed into the following three clauses:

C_{14}: $(pat\ po), (prob\ io) \leftarrow (pat\ oe)$

C_{15}: $(pat\ th), (prob\ io) \leftarrow (pat\ po)$

C_{16}: $(pat\ th), (prob\ oe) \leftarrow (pat\ po)$

By forwarding transformation [13], C_{15} and C_{16} are transformed into:

C_{17}: $(prob\ io) \leftarrow (pat\ po)$

C_{18}: $(prob\ oe) \leftarrow (pat\ po)$

By erasing independent satisfiable atoms [11], C_{14} is transformed into:

C_{19}: $(pat\ \ po), (prob\ io) \leftarrow$

Applying resolution to C_{17} and C_{19} yield the resolvent clause:

C_{20}: $(prob\ io) \leftarrow$

By elimination of subsumed clauses [11], C_{17} and C_{19} are removed. C_{18} is no longer useful for forwarding [13] and is removed. As a result, C_1 is transformed into C_{20}, from which the answer to the problem is readily obtained.

XI. Conclusions

We have successfully extended the concept of atom. The concepts of predicates, logical connectives, and quantifiers are combined. Such extension is not allowed by the conventional semantics, where the concepts of interpretations and models depend on the meanings of atoms. The general theory of logical structures [9], [10] and the new definition of a model based on referential constraints are essential for the construction of knowledge representation systems in this paper.

References

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

[2] I. Horrocks, P. F. Patel-schneider, S. Bechhofer, and D. Tsarkov, "OWL rules: A proposal and prototype implementation," *Journal of Web Semantics*, vol. 3, no. 1, pp. 23-40, 2005.

[3] 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.

[4] B. Motik and R. Rosati, "Reconciling description logics and rules," *Journal of the ACM*, vol. 57, no. 30, pp. 1-62, 2010.

[5] K. Akama and E. Nantajeewarawat, "Embedding proof problems into query-answering problems and problem solving by equivalent transformation," in *Proc. 5th International Conference on Knowledge Engineering and Ontology Development*, Vilamoura, Portugal, 2013, pp. 253-260.

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

[7] K. Akama and E. Nantajeewarawat, "Proving theorems based on equivalent transformation using resolution and factoring," in *Proc. 2nd World Congress on Information and Communication Technologies*, Trivandrum, India, 2012, pp. 7-12.

[8] K. Akama and E. Nantajeewarawat, "Meaning-preserving skolemization," in *Proc. International Conference on Knowledge Engineering and Ontology Development*, Paris, France, 2011, pp. 322-327.

[9] K. Akama and E. Nantajeewarawat, "Logical structures on specialization systems: Formalization and satisfiability-preserving transformation," in *Proc. 7th International Conference on Intelligent Technologies*, Taipei, Taiwan, 2006, pp. 100-109.

[10] K. Akama and E. Nantajeewarawat, "Construction of logical structures on specialization systems," in Proc. *World Congress on Information and Communication Technologies*, Mumbai, India, 2011, pp. 1030-1035.

[11] K. Akama and E. Nantajeewarawat, "Equivalent transformation in an extended space for solving query-answering problems," in *Proc. 6th Asian Conference on Intelligent Information and Database Systems*, LNAI 8397, Bangkok, Thailand, 2014, pp. 232-241.

[12] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook*, 2nd ed., Cambridge University Press, 2007.

[13] K. Akama and E. Nantajeewarawat, "Conjunction-based clauses for equivalent transformation of query-answering problems," *International Journal of Future Computer and Communication*, vol. 1, no. 1, pp. 5-8, 2012.

Kiyoshi Akama received his DEng from Tokyo Institute of Technology. He is a Professor at Division of Large-Scale Computational Systems, Information Initiative Center, Hokkaido University. His research interests include program generation and computation based on the equivalent transformation model, programming paradigms, knowledge representation, semantic web, and e-learning. He is a member of the Japanese Society for Artificial Intelligence and the Information Processing Society of Japan.

Ekawit Nantajeewarawat received his DEng in Computer Science from the Asian Institute of Technology. He is an Associate Professor at Computer Science Program, Sirindhorn International Institute of Technology, Thammasat University. His research interests include knowledge representation and automated reasoning. He is a member of the Association for Computing Machinery and the Institute of Electronics, Information and Communication Engineers.

Tadayuki Yoshida is a PhD candidate in Faculty of Computer Science, Hokkaido University. His research interests include program generation and computation based on equivalent transformation.