Event, Property and Hierarchy in Order-Sorted Logic

Ken Kaneiwa, Satoshi Tojo
School of Information Science
Japan Advanced Institute of Science and Technology
Tatsunokuchi, Ishikawa 923-1292, Japan
{kaneiwa,tojo}@jaist.ac.jp

Abstract

Knowledge representation in logics, even in the order-sorted logic that includes a sort hierarchy, tends to lose the conciseness and the nuances of natural language. If we could construct a logic that includes both predicates and terms as classes in the hierarchies, it would be very useful for connecting general knowledge to specific knowledge. Although there are actually logics that are equipped with such a predicate hierarchy, they are built by logical implication and they cause the problem of predicate unification between different argument structures. In this paper, we present a logic language with a class hierarchy of predicates, where in the unification of predicates we devise a mechanism for deriving superordinate predicates in the hierarchy and for quantifying supplementary arguments. The arguments are quantified differently, depending on whether a predicate is interpreted as an occurrence of an event or a universal property. Thus, we include the distinction between events and properties in predicates and present a logic language that can flexibly relate predicates with different argument structures. We formalize the logic language both by syntax and semantics, and develop the inference system.

1. Introduction

The goal of knowledge representation is not to put statements to be executed in order into a computer system, but to describe declarative knowledge in the real world. However, the logic language that is often applied to represent knowledge is unable to express the various nuances of natural language. In order to realize correct inferences in logic language, we need devices to describe more complicated expressions.

In working on this problem, many logic languages have been investigated from the viewpoints of knowledge representation and the rational reasoning. Amongst these languages, there seem to be two approaches to knowledge representation. One is typed logic programming [5], including order-sorted logic [13, 14, 4, 11]. LOGIN [1], LIFE [2], F-logic [6] and $\mathcal{Quixote}$ [15, 16] are inference systems with such type notions. They introduce class hierarchies together with feature structures, where a term represents a set of objects. New HELIC-II [10, 9], developed as a legal reasoning system, introduces H-terms (based on Ψ-terms in LOGIN) that consist of verb-type and and noun-type symbols. The other approach is the temporal reasoning [3, 8, 12] that considers various aspects of an eventuality. Allen [3] distinguished between event, property, and process in English sentences, and so did McDermott [8] between fact and event.

We consider defining both predicates and terms as classes in the hierarchies. In

order to do this, we amalgamate the above two approaches, and attribute both predicate hierarchy and event/property distinction to the issue of proper quantification of classes.

The objective of this paper is to propose a logic language which has two extensions as follows:

- Class-hierarchy of predicates (as well as sorted terms), together with an inference mechanism which is independent of argument structures;
- Distinction between event and property in predicate interpretation with the appropriate unification/ resolution mechanism.

We develop an inference system that includes substitution, quantification and supplementation of arguments, to realize the above specifications.

The paper is arranged as follows. Section 2 contains preliminaries about order-sorted logic and Ψ-terms [1]. Section 3 discusses the problem using examples. In Section 4, we illustrate how we introduce our extensions into order-sorted logic. In Section 5, we formalize the syntax and the semantics of the proposed logic. In Section 6, we define the logic programming based on this logic. Finally in Section 7, we give our conclusions and discuss future works.

2. Preliminaries

In this section, we state the notation of order-sorted logic, especially for Ψ-terms in LOGIN. S is a set of *sorts*, and the sorts are ordered by a subsort relation $\sqsubseteq_S$ ($\subseteq S \times S$). A hierarchy of sorts is a pair $(S, \sqsubseteq_S)$ of the set S and a subsort relation $\sqsubseteq_S$, containing the greatest sort $\top$ and the least $\bot$.

We can declare that *apple* and *orange* are subsorts of *fruit* as below.

$$apple \sqsubseteq_S fruit$$
$$orange \sqsubseteq_S fruit$$

A *sorted term* t_s is a term t of sort s. A variable x of a sort s is written as

$$x\colon s$$

which is called a *sorted variable*.

Aït-Kaci [1] proposes a notation, called Ψ-terms, based on sorted terms in order-sorted logic. A Ψ-term accompanies a sorted term $x\colon s$ with a *feature structure* written by a sequent of pairs of an attribute label and its value. The Ψ terms with feature structures represent more detailed information than simple sorted terms. For instance, the Ψ-term corresponding to 'red sour apples' is as below where *color* and *taste* are attribute labels and *red* and *sour* are their values, respectively.

$$x\colon apple[color \rightarrow y\colon red, taste \rightarrow z\colon sour]$$

Since the meaning of a Ψ-term is narrowed by the succeeding feature structure, the sorted term $x\colon apple$ is a more abstract expression than the Ψ term $x\colon apple[color \rightarrow y\colon red, taste \rightarrow z\colon sour]$. Note that $x\colon apple$ shows 'apples' and $x\colon apple[color \rightarrow y\colon red, taste \rightarrow z\colon sour]$ shows 'red sour apples'.

LOGIN has been developed as a logic programming language where PROLOG's arguments are replaced by Ψ-terms. For a predicate p, attribute labels $l_{11}, ..., l_{nk}$, Ψ-terms $t_{11}, ..., t_{nk}$, and variables $x_1\colon s_1, ..., x_k\colon s_k$, a fact is written in LOGIN as:

$$p(x_1{:}\,s_1[l_{11} \rightarrow t_{11}, ..., l_{n1} \rightarrow t_{n1}], ..., x_k{:}\,s_k[l_{1k} \rightarrow t_{1k}, ..., l_{nk} \rightarrow l_{nk}]).$$

Thus, it can express the relation of the concepts as complex objects.

Although we can translate Ψ-terms into the form of first order logic without sorted terms, we cannot directly represent knowledge by them. Sorted expressions(sorted terms or Ψ-terms) do not only retain the meaning of the original knowledge, but also represent the form simply. For example, the assertion describes an expression of 'John, who is a twenty years old man, is walking' as below.

$$walk(x{:}\,John[age \rightarrow y{:}\,twenty\_years, sex \rightarrow z{:}\,male]).$$

For the sake of simplicity, we use a sort *John* as a singleton instead of a constant.

In [1], Aït-Kaci contended that the inference by unification of sorted terms was more efficient than by resolution processes. In Section 4 and Section 5, we will extend the expression based on the Ψ-term explained above.

3. Motivation

In this section, we shall discuss examples that raise the questions of knowledge representation using Ψ-terms.

Example 1: A Hierarchy of Predicates

We now consider that superordinate predicates are derived from subordinate ones in the hierarchy of predicates. That is, the abstract expression of predicate can be inferred from the concrete expression.

In the hierarchy of predicates in Fig. 1 we expect the following results of an inference. Suppose a fact `hit(x:John)` holds, then the superordinate predicate `illegal_act` can be derived from the predicate `hit` from the direction (1) in Fig. 1. However, the first query 'Did John do an illegal act against Mary?' will give the answer `no`. It is certain that John hit somebody but not that John hit Mary. Thus, the second query 'Did John do an illegal act against somebody?' will yield `yes`.

```
hit(x:John).
?-illegal_act(x:John, y:Mary).
no.
?-illegal_act(x:John, y:person).
yes.
```

This exemplifies the case of a derivation of predicate higher in the hierarchy of predicates with more arguments than the predicate representing the fact. To make the inference above, we have to supplement the arguments existing only in `illegal_act` and not existing in `hit`. In addition, we have to take account of the quantification of the supplemented arguments. When the second argument `y:person` is interpreted as all persons, the answer to second query may be `no`. However the interpretation does not fit with what we expect, so that `y:person` should be interpreted as a person(somebody).

Similarly, assuming a predicate `illegal_act` with fewer arguments than the predicate in a fact `steal(x:John, y:Mary)`, in the direction (2) in Fig. 1 the derivation of the following query results in `yes`. The answer is plausible because a fact `steal(x:John, y:Mary)` implies `illegal_act(x:John)` as more abstract information.

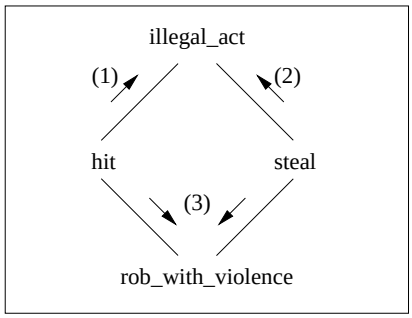

Figure 1: A hierarchy of predicates

```
steal(x:John, y:Mary).
?-illegal_act(x:John).
yes.
```

Additionally, the predicate `rob_with_violence` as the conjunction of `hit` and `steal` will be derived from the two predicates (on (3) in Fig. 1). That is, by the facts `hit(x:John, y:Mary)` and `steal(x:John, z:wallet)` in an incident John's robbing with violence holds and then the query 'Did John steal Mary's wallet using robbery with violence?' will yield `yes`.

```
hit(x:John, y:Mary).
steal(x:John, z:wallet).
?-rob_with_violence(x:John, y:Mary, z:wallet).
yes.
```

Example 2: Event and Property Interpretations

Interpreting a predicate from a natural language assertion can result in the reasoning process having a different direction, depending on the interpretation. Using a predicate in knowledge representation, there are two roles for the predicate, these being *event* and *property* [3, 8]. In the example of the sort hierarchy in Fig. 2, we consider the assertion `fly(x:bird)`. This is ambiguous because the term can be interpreted in two ways.

The fact `fly(x:bird)`, when interpreted as an event, states that a bird is flying. The fact entails that an animal is flying so that the query `?-fly(x:animal)` results in `yes`. However the fact does not state that a penguin is flying, and the answer to query `?-fly(x:penguin)` is `no` as follows.

Interpretation 1: A bird is flying.

```
fly(x:bird).
?-fly(x:animal).
yes.
?-fly(x:penguin).
no.
?-move(x:animal).
yes.
```

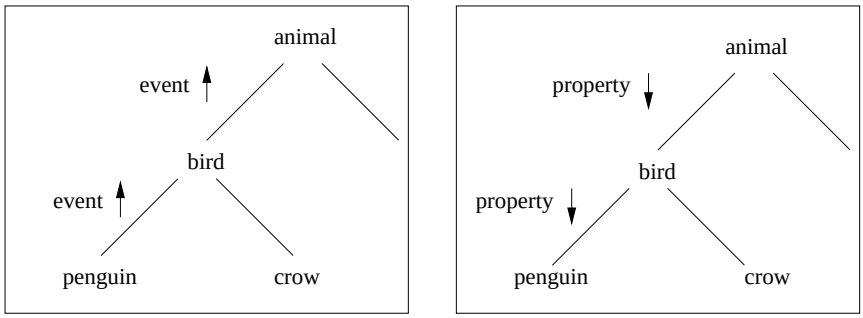

Figure 2: The inference by event and property

The assertion `fly(x:bird)` as an event can be used to deduce the superordinate terms. For example, `animal` is one of the superordinate terms of `bird` as in Fig 2. In contrast, the fact `fly(x:bird)` which is interpreted as a property, states that birds have the property of flight. If birds have this property, then penguins should have the same property. Thus, the query `?-fly(x:penguin)` will have the answer `yes`. However since the information does not imply that all animals have the property of flight, the following answer for query `?-fly(x:animal)` will yield `no`.

Interpretation 2: Birds have the property of flight.

```
fly(x:bird).
?-fly(x:animal).
no.
?-fly(x:penguin).
yes.
?-move(x:animal).
no.
```

Fig. 2 demonstrates that the subordinate terms inherit the property from superordinate terms. Namely, a conclusion `fly(x:penguin)` can be inferred from the fact `fly(x:bird)` by an inheritance to the subordinate term `x:penguin`. If an assertion does not distinguish between event and property, then we cannot deal with inferences using the two kinds of directions in the hierarchy.

The problems of the above two examples we have seen suggest the necessity of both hierarchies of sorts(terms) and predicates and the distinction of a predicate (event or property).

4. Event, Property and Hierarchy

In order to realize the inference in the previous section, we introduce the hierarchy and the event/property distinction in predicates.

4.1. A hierarchy of predicates

We would like to realize a hierarchy of predicates, independent of the hierarchy of sorts (terms). A hierarchy of predicates is built by a binary relation $\sqsubseteq_P$ of predicate

symbols that is a partial order, e.g. for any predicates p_1, p_2, $p_1 \sqsubseteq_P p_2$ means that p_2 is a superordinate predicate of p_1 and that p_1 is a subordinate predicate of p_2.

Normally, a relation in the hierarchy of predicates is expressed by the implication symbol, used in first order logic, as an ISA-relation. The following example will illustrate a rule that *move* is a superordinate predicate of *fly* using the logical implication.

$$fly(X) \rightarrow move(X)$$

The implication infers that if *fly(a)* holds, then *move(a)* does.

However, such rules do not completely represent the relation between predicates in the hierarchy. The problem is that each superordinate/subordinate predicate has its own unique argument structure, and that there may be a difference between them. For example, suppose there are predicates *explain* and *talk*. Both predicates have two arguments and are written like:

$$explain(x\colon John, y\colon book)$$
$$talk(x\colon John, z\colon Mary)$$

In this case, the first argument of both predicates is same, whereas the second arguments are different. The role of both first arguments is agent, and the role of the second argument in *explain* is object but in *talk* is coagent. Therefore, the second argument cannot be unified between *explain* and *talk* when these predicates are in the relation $\sqsubseteq_P$ in the hierarchy of predicates. If we were to stick to semantics by logical implication, we may have to write several rules to associate arguments in the premise with those in the conclusion (e.g. $p(X,Y) \rightarrow q(X,Y)$, $p(X,Y) \rightarrow q(Y,X)$, $p(X,X) \rightarrow q(X,X), ...$). In particular, they would become more complicated as the number of predicates in the hierarchy increased.

Suppose that

$$explain \sqsubseteq_P talk$$

is declared, then to fill the gap between the argument structures of predicates, we need to complement missing arguments by adding and deleting some of them. How do we obtain the information for the argument structures of predicates? We use *argument labels* to render arguments roles. Note that these differ from the attribute labels in Ψ-terms shown in Section 2. Argument labels *agt, obj* and so on, which are written like

$$explain(\underline{agt} \Rightarrow x\colon John, \underline{obj} \Rightarrow y\colon book)$$

uniquely represent the argument roles to build an argument structure. In this example *agt* is an argument label 'agent' and *obj* is an argument label 'object'. This notation would be able to treat the query

$$?\text{-}talk(agt \Rightarrow x\colon person, coagt \Rightarrow y\colon person)$$

where the fact

$$explain(agt \Rightarrow x\colon John, obj \Rightarrow y\colon book)$$

is given, in the following way.

$explain(agt \Rightarrow x{:}John, obj \Rightarrow y{:}book)$

$\downarrow(1)$ derivation of the superordinate predicate

$talk(agt \Rightarrow x{:}John, obj \Rightarrow y{:}book)$

$\downarrow(2)$ deletion of an argument

$talk(agt \Rightarrow x{:}John)$

$\downarrow(3)$ addition of an argument

$talk(agt \Rightarrow x{:}John, coagt \Rightarrow z{:}person)$

$\downarrow(4)$ substitution of the sort

$talk(agt \Rightarrow x{:}person, coagt \Rightarrow z{:}person)$

By (1), the predicate *talk* is derived from the subordinate predicate *explain*. (2) deletes the argument $obj \Rightarrow y{:}book$ that is a surplus argument in predicate *talk*. (3) adds the argument $coagt \Rightarrow z{:}person$ that is deficient in the arguments of the predicate *talk*. Finally, (4) substitutes $agt \Rightarrow x{:}person$ for the first argument $agt \Rightarrow x{:}John$.

For (2) and (3), each argument label gives a scope to the argument. Let *SCP* be a function from the set of argument labels to the set of sorts. For argument labels *agt,obj*, if we define *SCP(agt) = person* and *SCP(obj) = thing*, then the sorts *person* and *thing* indicate the scope of *agt* and *obj*. In the supplementation of arguments, the addition and deletion of arguments depend on the argument structure, and the value of the added argument is supplied from the scope of the argument indicated by the argument label.

4.2. Predicates as Event and Property

Predicate symbols can be used to represent features or states of objects. For example, $walk(x{:}John)$ means that John is walking, and $red(y{:}apple)$ means that an apple is red. However, predicates in these assertions may not have a consistent usage. That is, the predicate in $walk(x{:}John)$ implies an event and the predicate in $red(y{:}apple)$ implies a property. As a result, the inference in the hierarchy can give rise to erroneous unification for the two kinds of usage. Also the single predicate *walk* may have two kinds of interpretation as the event 'is walking' and the property 'can walk'.

We need to present reasoning for each predicate either as an event or a property with different notation and need to define the relevant unification. The distinction is whether an assertion is interpreted as an occurrence of an event, or as a property of objects. We define two aspects of predicates as follows.

Definition 4.1 (Two aspects of a predicate). *For any predicate p_i there are the predicate p_i as event and the predicate $p_i^\sharp$ as property.*

We alter terms, which are based on Ψ-terms, to distinguish between an existential notation and a universal notation for the sorted domain, e.g. *bird* denotes all birds and $x{:}bird$ denotes a bird. Thus, examples of an assertion defined as event and as a property are given below:

Event assertion: The predicate as event expresses that there is a fact or there is an occurrence of such an event. The argument of the predicate is limited to an object, so that the following assertion is interpreted as 'A bird is flying'.

$$fly(x{:}bird) \simeq \exists x\ fly\_as\_event(x{:}bird)$$

Property assertion: The predicate as property expresses an attribute of objects. The argument is universal in the set of objects within a sort, so that the following form means 'All birds have the property of flight'.

$$fly^{\sharp}(bird) \simeq \forall x\ fly\_as\_property(x{:}\,bird)$$

Moreover, the two aspects of a predicate naturally cause a difference in inferences. If the predicate is an event, then the added argument should be one occurrence of an object corresponding to one event. If the predicate is a property, then the added argument should not be a unique object but a global feature of that sort. Therefore, if we add a term $x{:}\,s$ to a predicate interpreted as event, the term will denote an object within sort s; if we add a term s to a predicate interpreted as property, it will denote all objects in sort s.

For example, the supplementation of arguments can be distinguished as follows.

Predicate as event:

$hit(agt \Rightarrow x{:}\,John)$
$\Downarrow$ addition of an argument
$hit(agt \Rightarrow x{:}\,John, coagt \Rightarrow \underline{y{:}\,person})$
(John hit a person.)

Predicate as property:

$hit^{\sharp}(agt \Rightarrow x{:}\,John)$
$\Downarrow$ addition of an argument
$hit^{\sharp}(agt \Rightarrow x{:}\,John, coagt \Rightarrow \underline{person})$
(John has the property of hitting every person.)

The scope of the argument label *coagt* can determine the sort *person* of terms added to the predicates *hit* and $hit^{\sharp}$. In this case, $y{:}\,person$ is added to the predicate as event, and *person* is added to the predicate as property.

5. An Order-Sorted Logic with Event, Property and Hierarchy

In this section, we formalize an order-sorted logic which adopts a hierarchy of predicates and a distinction between event and property.

5.1. Signature

We now revise the order-sorted logics and the Ψ-term in [7, 4, 1].

Definition 5.1 (Signature). *A signature for logic with order-sorts is* $\Sigma = (S, P, LP)$ *if*

(1) $(S, \sqsubseteq_S)$ *is a partially ordered set of sort symbols with the greatest sort* $\top$ *and the least* $\bot$.

(2) $(P, \sqsubseteq_P)$ *is a partially ordered set of predicate symbols.*

(3) LP *is a set of argument labels of predicates.*

(4) SCP *is a function from* LP *to* S.

(5) ARG *is a function from* P *to* 2^{LP}.

$SCP(l)$ denotes a sort as the scope of an argument label l. $ARG(p)$ indicates a set of argument labels as the unique argument structure of a predicate p. We use $\sharp$ to indicate that the predicate is read as a property, where the predicate $p^\sharp$ is called a *property predicate* and the predicate p without it is called an *event predicate*. The set P of predicates can be extended as follows: $P^+ = P \cup \{p^\sharp | p \in P\}$. In the latter sections, all our signatures will be the extended signature $\Sigma = (S, P^+, LP)$.

5.2. Syntax

A language L of signature Σ contains the following: the family $V = \{V_s | s \in S\}$ of variables where $V_s (= \{v_1 : s,\ v_2 : s, ...\})$ is all the variables of sort s, the set LS of attribute labels and the propositional operations$(\vee, \neg)$.

We define the expressions of L: *terms* and *formulas*.

Definition 5.2 (Terms). *Given a language L of signature Σ, the set TERM of terms is defined by:*

(1) A sort s and a sorted variable $x : s$ are atomic terms of sort s.

(2) If t_0 is an atomic term of sort s, $t_1, ..., t_n$ are terms without variable and $l_1, ..., l_n$ are attribute labels, then $t_0[l_1 \rightarrow t_1, ..., l_n \rightarrow t_n]$ is a term of sort s.

We say that a term t of sort s is a term of argument label l if s and $SCP(l)$ are in the subsort relation, namely, $(s, SCP(l)) \in \sqsubseteq_S$.

Definition 5.3 (Formulas). *Given a language L of a signature Σ, the set FORM of formulas is defined by:*

(1) If $t_1, ..., t_n$ are terms of $l_1, ..., l_n$ and p is a predicate where $ARG(p) = \{l_1, ..., l_n\}$, then $p(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n)$ is the atomic formula for an event predicate.

If $t_1, ..., t_n$ are terms of $l_1, ..., l_n$ and $p^\sharp$ is a predicate where $ARG(p) = \{l_1, ..., l_n\}$, then $p^\sharp(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n)$ is the atomic formula for a property predicate.

(2) If A and B are formulas, then $\neg A$ and $A \vee B$ are formulas.

5.3. Semantics

For a language L of signature Σ, a structure is a pair $T = (U, [\![\ ]\!])$ (a universe U and an interpretation $[\![\ ]\!]$.) For each attribute label l_i in LS, $[\![l_i]\!]$ is a function from U to U. An interpretation of sorts is defined by $[\![\ ]\!] : (S, \sqsubseteq_S) \rightarrow (U, \subseteq)$ where $[\![\top]\!] = U$ and $[\![\bot]\!] = \phi$.

Definition 5.4 (Interpretation). *Given a family $\alpha = \{\alpha_s : V_s \rightarrow [\![s]\!] | s \in S\}$ of variable assignments, an interpretation $[\![\ ]\!]$ of terms and atomic formulas is defined by:*

(1) $[\![s]\!]_\alpha = [\![s]\!]$, $[\![x : s]\!]_\alpha = \{\alpha_s(x : s)\}$.

(2) $[\![t_0[l_1 \rightarrow t_1, ..., l_n \rightarrow t_n]]\!]_\alpha = \{x \in [\![t_0]\!]_\alpha | \exists y_1 \in [\![t_1]\!]_\alpha, ..., \exists y_n \in [\![t_n]\!]_\alpha,\ [\![l_1]\!](x) = y_1, ..., [\![l_n]\!](x) = y_n\}$.

(3) $[\![\varphi]\!] \subseteq X_p$,
$[\![\varphi(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n)]\!]_\alpha = 1$ *iff* $\{f \in X_p | \forall l_i \in ARG(p), f(l_i) \in [\![t_i]\!]_\alpha\} \subseteq [\![\varphi]\!]$

where φ *is an event predicate* p *or a property predicate* $p^\sharp$ *and* $X_p = \{f \in (ARG(p) \to U) | \forall l \in ARG(p), f(l) \in [\![SCP(l)]\!]\}$.

By this definition, the ordering of arguments in a predicate does not alter the interpretation of the atomic formula. For example, formulas

$$p(l_1 \Rightarrow t_1, l_2 \Rightarrow t_2) \text{ and } p(l_2 \Rightarrow t_2, l_1 \Rightarrow t_1)$$

are regarded as semantically identical. As an argument is a term of a sort, that is, interpreted as a subset of U, a predicate has to be interpreted as the relation of the subsets.

I^σ or $I^{\sigma\sharp}$ is an interpretation of the supplementation of predicate arguments distinguished between event and property.

Definition 5.5 (Interpretation of supplementation of arguments). *For* p, $p^\sharp, q \in P^+$, *let be* $l_j \in ARG(p) \cap ARG(q)$ *and* $r_k \in ARG(q) - ARG(p)$. *An interpretation* ($I^\sigma_{q/p}$ *or* $I^{\sigma\sharp}_{q/p}$) *of supplementation of arguments is defined by:*

- $I^\sigma_{q/p}([\![p]\!]) = \{g \in (ARG(q) \to U) | \forall f \in F_i, g(l_j) = f(l_j), g(r_k) = \alpha_k(x_k{:}\, s_k)\}$
- $I^{\sigma\sharp}_{q/p}([\![p^\sharp]\!]) = \{g \in (ARG(q) \to U) | \forall f \in F_i, g(l_j) = f(l_j), g(r_k) \in [\![s_k]\!]\}$

where $s_k = SCP(r_k)$ *and* $\alpha_k{:}\, V_{s_k} \to [\![s_k]\!]$.

The subscript q/p of $I^\sigma_{q/p}$ and $I^{\sigma\sharp}_{q/p}$ means a translation from an interpretation of the predicate p into the predicate q. Accordingly, we can say $I^\sigma_{q/p}([\![p]\!]) \subseteq X_q$ where $X_q = \{f \in (ARG(q) \to U) | \forall l \in ARG(q), f(l) \in [\![SCP(l)]\!]\}$.

Next we will give an interpretation of hierarchy of predicates.

Definition 5.6 (Hierarchy of predicates). *Given the set* P *of predicates and a subpredicate relation* $\sqsubseteq_P$ *, a hierarchy of predicates is a pair* $(P, \sqsubseteq_P)$, *which is interpreted by:*

$$[\![\ ]\!] : (P, \sqsubseteq_P) \to (2^{X_P}, I^\sigma, I^{\sigma^\sharp}, \subseteq)$$

where $X_p = \{f \in (ARG(p) \to U) | \forall l \in ARG(p), f(l) \in [\![SCP(l)]\!]\}$ *and* $X_P = \bigcup_{p \in P} X_p$.

We define the one-step subordinate relation $\sqsubseteq^1_P = \{(p_i, p_j) \in \sqsubseteq_P | i \neq j, (p_i, r), (r, p_j) \notin \sqsubseteq_P\}$. *For any predicates* p_i *in the hierarchy of predicates, the interpretation* $[\![\ ]\!]$ *of subpredicate relation* $\sqsubseteq_P$ *satisfies the following conditions.*

(1) If $p_1 \sqsubseteq_P p_2$, *then* $I^\sigma_{p_2/p_1}([\![p_1]\!]) \subseteq [\![p_2]\!]$ *and* $I^{\sigma\sharp}_{p_2/p_1}([\![p_1^\sharp]\!]) \subseteq [\![p_2^\sharp]\!]$

(2) If $p_0 \sqsubseteq_P p_1, ..., p_0 \sqsubseteq_P p_n (n > 1)$,
then $\bigcap_{i=1}^n [I^\sigma_{p_0/p_i}([\![p_i]\!])] \subseteq [\![p_0]\!]$ *and* $\bigcap_{i=1}^n [I^{\sigma\sharp}_{p_0/p_i}([\![p_i^\sharp]\!])] \subseteq [\![p_0^\sharp]\!]$

where $p_1, ..., p_n$ *are all predicates such that* $p_0 \sqsubseteq^1_P p_i$.

Definition 5.7 (Satisfaction relation). *Let* T *be a structure and* F *a formula. The satisfaction relation* $T \models F$ *is defined by:*

(1) $T \models \varphi(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n) \Leftrightarrow$ *for some family* α *of variable assignments,* $[\![\varphi(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n)]\!]_\alpha = 1$

(2) $T \models A \vee B \Leftrightarrow T \models A$ *or* $T \models B$

(3) $T \models \neg A \Leftrightarrow T \not\models A$

For every formula $F \in \Gamma$, we write $T \models \Gamma$ (T is a model of Γ) if $T \models F$. We say that Γ is *satisfiable* if it has a model. Otherwise, we say that Γ is *unsatisfiable* if it has no models.

6. Logic Programming

We define a logic programming that includes a definition of our extended order-sorted language.

Definition 6.1 (Program). *Let $\neg L_1, ..., \neg L_n$ be negative literals and L be a positive literal. A program clause is $L \vee \neg L_1 \vee ... \vee \neg L_n$. The general form of program clause is written as: $R_i : L \leftarrow L_1, ..., L_n$.*

Let $p, p' \in P$ and $s, s' \in S$. HS_i is the declaration of a subsort relation and HP_j the declaration of a subpredicate relation as follows.

$$HS_i : s \sqsubseteq_S s',$$

$$HP_j : p \sqsubseteq_P p'.$$

A program $\mathcal{P}$ is a finite set of program clauses, declarations of a subsort relation and declarations of a subpredicate relation.

$$\mathcal{P} = \{R_1, ..., R_n, HS_1, ..., HS_k, HP_1, ..., HP_m\}$$

A goal clause is $\neg L_1 \vee ... \vee \neg L_n$. The form of goal clause is written as: $\leftarrow L_1, ..., L_n$. The substitution of sorted terms depends on whether the literals with the terms are positive or negative in the clause. $Atts$(called *attributes*) is a finite sequence of pairs of attribute labels and terms $l_1 \rightarrow t_1, ..., l_k \rightarrow t_k (k \geq 0)$.

Definition 6.2 (Substitution). *Given a program $\mathcal{P}$ and a goal clause G, for $x{:}\, s_i \in V_{s_i}$, $y{:}\, s_j \in V_{s_j}$, $s_i, s_j \in S$, a substitution θ is a function from TERM to TERM defined as one of the following by the rules:*

(1) $x{:}\, s_i[Atts] \stackrel{\theta}{\Longrightarrow} s_i[Atts]$,
(2) $s_i[Atts] \stackrel{\theta}{\Longrightarrow} s_j[Atts]$ *where* $(s_i \sqsubseteq_S s_j) \in \mathcal{P}$,
(3) $x{:}\, s_i[Atts] \stackrel{\theta}{\Longrightarrow} y{:}\, s_j[Atts]$ *where* $(s_j \sqsubseteq_S s_i) \in \mathcal{P}$,
(4) $x{:}\, s_i[Atts] \stackrel{\theta}{\Longrightarrow} x{:}\, s_i[Atts, l_{k+1} \rightarrow t_{k+1}]$ *where $l_{k+1} \rightarrow t_{k+1}$ does not occur in the Atts.*

Let $\varphi(l_1 \Rightarrow t_1, ..., l_m \Rightarrow t_m), L_1, ..., L_n$ be atomic formulas and θ a substitution. $\theta\varphi(l_1 \Rightarrow t_1, ..., l_m \Rightarrow t_m)$ and $\theta(L_1, ..., L_n)$ are defined by:

- $\theta\varphi(l_1 \Rightarrow t_1, ..., l_m \Rightarrow t_m) = \varphi(l_1 \Rightarrow \theta t_1, ..., l_m \Rightarrow \theta t_m)$ *where θt_i is a term of l_i,*
- $\theta(L_1, ..., L_n) = \theta L_1, ..., \theta L_n$.

Let A and B be expressions. A substitution θ is a unifier of A and B if $\theta A = B$. $Args$(called *arguments*) is a finite sequence of pairs of argument labels and terms $l_1 \Rightarrow t_1, ..., l_m \Rightarrow t_m (m > 0)$. $FORM_0 (\subset FORM)$ is the set of atomic formulas, and

$$FORM_0^* = \{\varphi(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n) | \{l_1, ..., l_n\} \in 2^{LP}, \varphi \in P^+, t_i \in TERM\}$$

is the set of atomic formulas that is expanded by all argument structures including illegal ones (e.g. $p(l_1 \Rightarrow t_1, l_1 \Rightarrow t_2)$ is illegal when $ARG(p) \neq \{l_1, l_2\}$).

Definition 6.3 (Supplementation of arguments). *Given an atomic formula A, a supplementation σ of predicate arguments, that is a function from $FORM_0^*$ to $FORM_0$, is defined by:*

$$\sigma(A) = ADD^m(DEL^n(A))$$

where ADD^m is a composition of m-functions of ADD and DEL^n a composition of n-functions of DEL and therefore m is the least number such that $ADD^m = ADD^{m+1}(m > 0)$ and n the least number such that $DEL^n = DEL^{n+1}(n > 0)$. For $l \in LP$, $\varphi \in P^+$, $s \in S, x{:}s \in V_s$ and $t \in TERM$, an addition ADD of arguments is

$$ADD(\varphi(Args)) = \begin{cases} \varphi(Args, l \Rightarrow x{:}s) & \text{if } \varphi = p \text{ and } \exists l \in A, \\ \varphi(Args, l \Rightarrow s) & \text{if } \varphi = p^\sharp \text{ and } \exists l \in A, \\ \varphi(Args) & \text{otherwise}, \end{cases}$$

where $A = \{l \in ARG(p) | l \Rightarrow t \notin Args\}$ and $SCP(l) = s$. And a deletion DEL of arguments is

$$DEL(\varphi(Args, l \Rightarrow t)) = \begin{cases} \varphi(Args) & \text{if } l \notin ARG(\varphi) \text{ and} \\ & (\ \varphi = p^\sharp, t = v{:}s) \text{ or} \\ & (\ \varphi = p, t = s), \\ \varphi(Args, l \Rightarrow t) & \text{if } B - ARG(\varphi) = \phi, \end{cases}$$

where $B = \{l_i | l_i \Rightarrow t_i \in (Args \cup \{l \Rightarrow t\})\}$ and $SCP(l) = s$.

We define the inference rules in the logic programming that are applied to derivations. LS and LS' are finite sequences of positive literals, L is a positive literal.

Definition 6.4 (Inference rules). *Let θ be a substitution and σ a supplementation of arguments. Inference rules are written as follows:*

$$\frac{\leftarrow LS', L \quad L \leftarrow LS}{\leftarrow LS, LS'}\ (RP) \qquad \frac{\leftarrow LS}{\leftarrow \theta LS}\ (Sub)$$

$$\frac{p_1 \sqsubseteq_P p_2 \quad \leftarrow LS, p_2(Args)}{\leftarrow LS, \sigma(p_1(Args))}\ (Spec1)$$

$$\frac{p_1 \sqsubseteq_P p_2 \quad \leftarrow LS, p_2^\sharp(Args)}{\leftarrow LS, \sigma(p_1^\sharp(Args))}\ (Spec2)$$

$$\frac{p_0 \sqsubseteq_P p_1 \quad ... \quad p_0 \sqsubseteq_P p_n \quad \leftarrow LS, p_0(Args)}{\leftarrow LS, \sigma(p_1(Args)), ..., \sigma(p_n(Args))}\ (Gen1)$$

$$\frac{p_0 \sqsubseteq_P p_1 \quad ... \quad p_0 \sqsubseteq_P p_n \quad \leftarrow LS, p_0^\sharp(Args)}{\leftarrow LS, \sigma(p_1^\sharp(Args)), ..., \sigma(p_n^\sharp(Args))}\ (Gen2)$$

if σ is properly applied in each of $(Spec1)$, $(Spec2)$, $(Gen1)$ and $(Gen2)$, where $p_1, ..., p_n (n > 1)$ are all predicates such that $p_0 \sqsubseteq_P^1 p_i$ in $(Gen1)$ and $(Gen2)$.

We write $\Gamma \models F$ (F is a consequence of Γ) if every model of Γ is a model of a formula F.

$\mathcal{P}_1 = \{$rob_with_violence $\sqsubseteq_{\mathrm{P}}$ hit, rob_with_violence $\sqsubseteq_{\mathrm{P}}$ steal,
hit $\sqsubseteq_{\mathrm{P}}$ illegalact, steal $\sqsubseteq_{\mathrm{P}}$ illegalact, wallet $\sqsubseteq_{\mathrm{S}}$ thing,
John $\sqsubseteq_{\mathrm{S}}$ person, Mary $\sqsubseteq_{\mathrm{S}}$ person,
hit(agt $\Rightarrow$ x:John, coagt $\Rightarrow$ y:Mary),
steal(agt $\Rightarrow$ x:John, obj $\Rightarrow$ w:wallet) $\}$

$\mathcal{P}_2 = \{$fly $\sqsubseteq_{\mathrm{P}}$ move, walk $\sqsubseteq_{\mathrm{P}}$ move,
bird $\sqsubseteq_{\mathrm{S}}$ animal, penguin $\sqsubseteq_{\mathrm{S}}$ bird, crow $\sqsubseteq_{\mathrm{S}}$ bird,
fly(sbj $\Rightarrow$ x: bird),
fly$^\sharp$(sbj $\Rightarrow$ bird) $\}$

Figure 3: The programs for examples

Theorem 6.1. *The conclusion C' of each inference rule of definition 6.4 is a consequence of its premise $\{C_1, C_2\}$ or $\{C\}$. That is, $\{C_1, C_2\} \models C'$ or $\{C\} \models C'$.*

Definition 6.5 (Derivation). *From a program $\mathcal{P}$ and a goal G ($\mathcal{P} \cup \{G\}$), a goal G' is obtained from G using one of the inference rules. We write $\mathcal{P} \cup \{G\} \vdash_{LP} G'$ to indicate the derivation.*

Given $\mathcal{P} \cup \{G\}$, we can construct a signature $\Sigma_{\mathcal{P} \cup \{G\}}$ where $s, s' \in S,\ p, p' \in P$, $(s, s') \in \sqsubseteq_S$ and $(p, p') \in \sqsubseteq_P$ for every $s \sqsubseteq_S s', p \sqsubseteq_P p' \in \mathcal{P}$.

Let T be a structure of $\Sigma_{\mathcal{P} \cup \{G\}}$,

$$T \models_{LP} \mathcal{P} \cup \{G\} \textit{ iff } T \models C_i \textit{ for every clause } C_i \in \mathcal{P} \cup \{G\}.$$

We say that T is a model of program $\mathcal{P} \cup \{G\}$. We write $\mathcal{P} \cup \{G\} \models_{LP} C$ (C is a consequence of program $\mathcal{P} \cup \{G\}$) if every model of program $\mathcal{P} \cup \{G\}$ is a model of a clause C.

A *refutation* is to derive the empty clause $\Box$ from $\mathcal{P} \cup \{G\}$.

Theorem 6.2 (Soundness of refutation). $\mathcal{P} \cup \{G\} \vdash_{LP} \Box \Rightarrow \mathcal{P} \cup \{G\} \models_{LP} \Box$

Let us look at the refutation processes from the examples we have seen in Section 3. The programs $\mathcal{P}_1$ for the example 1 and $\mathcal{P}_2$ for the example 2 are shown in Fig. 3. From a goal

$$?\text{-}rob\_with\_violence(agt \Rightarrow x\text{:}\, John)$$

and $\mathcal{P}_1$, the refutation process is described in Fig. 4.

In the same way, from a goal

$$?\text{-}move(sbj \Rightarrow y\text{:}\, animal)$$

and $\mathcal{P}_2$, the refutation succeeds as in Fig. 5.

On the other hand, another goal

$$?\text{-}move^\sharp(sbj \Rightarrow y\text{:}\, animal)$$

and $\mathcal{P}_2$ cannot derive the empty clause and the refutation fails.

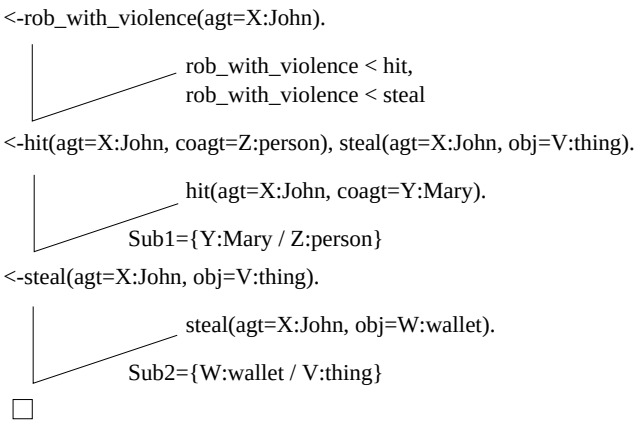

Figure 4: A refutation process 1

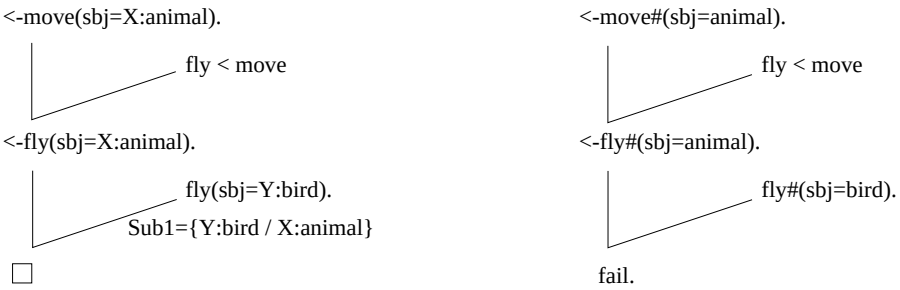

Figure 5: Refutation processes 2

7. Conclusions and Future Work

We have presented an order-sorted logic for knowledge representation, which enables us to describe the hierarchies of both predicates and terms inherent in natural language. We have developed the new logic programming language using SICStus Prolog^{TM} ver.3.0, and have shown that the language properly solves the examples shown in Section 3.

In this language, we can write database statements, disregarding the argument structures of other predicates. The most advantageous point of our system is the mechanism of argument supplementation to unify predicates with different argument structures. It allows us to write hierarchical relations between predicates flexibly and concisely. Also, we can precisely distinguish *event* from *property* as two different aspects of a predicate. In addition, our logic programming language provides unification/ resolution mechanisms that fit human reasoning.

Thus far, we have considered making order-sorted logic closer to the semantics of natural language. In addition to the event/property distinction, we are now tackling the inclusion of the proper treatment of negation in the predicate/sort hierarchy. Although this work seems to be difficult, we contend that such notions are significant in representing knowledge naturally and precisely.

References

[1] H. Ait-Kaci and R. Nasr. Login: A logic programming language with built-in inheritance. *Journal of Logic Programming*, pages 185–215, 1986.

[2] H. Ait-Kaci and A. Podelski. Towards a meaning of life. *Journal of Logic Programming*, pages 195–234, 1993.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[4] C. Beiercle, U. Hedtsuck, U. Pletat, P.H. Schmit, and J. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55:149–191, 1992.

[5] P. M. Hill and R. W. Topor. A semantics for typed logic programs. In F. Pfenning, editor, *Types in Logic Programming*. MIT Press, 1992.

[6] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.

[7] M. Manzano. Introduction to many-sorted logic. In *Many-sorted Logic and its Applications*, pages 3–86. John Wiley and Sons, 1993.

[8] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[9] K. Nitta, S. Tojo, and et al. Knowledge representation of new helic II. In *Workshop on Legal Application of Logic Programming,ICLP '94*, 1994.

[10] K. Nitta, S. Wong, and Y. Ohtake. A computational model for trial reasoning. In *Proc. 4th Int. Conf. on AI and Law, Amsterdam*, 1993.

[11] M. Schmidt-Schauss. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Springer-Verlag, 1989.

[12] Y. Shoham. *Reasoning about Change*. The MIT Press, 1988.

[13] C. Walter. A mechanical solution of schuber's steamroller by many-sorted resolution. *Artificial Intelligence*, 26(2):217–224, 1985.

[14] C. Walter. Many-sorted unification. *Journal of the Association for Computing Machinery*, 35:1, 1988.

[15] H. Yasukawa, H. Tsuda, and K. Yokota. Objects properies and modules in $\mathcal{QUIXOTE}$. In *Proc. FGCS'92*, pages 257–268, 1992.

[16] K. Yokota. *Quixote:A Constraint Based Approach to a Deductive Object-Oriented Database*. PhD thesis, 1994.