Programming of Future Generation Computers
K. Fuchi, M. Nivat (Editors)
Elsevier Science Publishers B.V. (North-Holland), 1988

# DCKR -- Knowledge Representation in Prolog and Its Application to Natural Language Processing

Hozumi Tanaka

Dept. of Computer Science
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku
Tokyo, Japan

ABSTRACT: Semantic processing is one of the important tasks for natural language processing. Basic to semantic processing are descriptions of lexical items. The most frequently used form of description of lexical items is probably Frames or Objects. Therefore in what form Frames or Objects are expressed is a key issue for natural language processing. A method of the Object representation in Prolog called DCKR will be introduced. It will be seen that if part of general knowledge and a dictionary are described in DCKR, part of context-processing and the greater part of semantic processing can be left to the functions built in Prolog.

## 1. Introduction

Relationships between knowledge represented in predicate logic formulas and knowledge represented in Frames or Structured objects are clarified by [ Hayes 80 ], [ Nilsson 80 ], [ Goebel 85 ],[ Bowen 85 ], et al, but their methods requires separately an interpreter for their representation. The authors have developed a knowledge representation form called DCKR (Definite Clause Knowledge Representation) [ Koyama 85 ]. In DCKR, each of the slots composing of a Structured Object (hereinafter simply called an object) is represented by a Horn clause (a Prolog statement) with the "sem" predicate (to be explained in Section 2) as its head. Therefore, an Object can be regarded as a set of Horn clauses (slots) headed by the sem predicate with the same first argument. From the foregoing it follows that almost all of a program for performing inferences relative to knowledge described in DCKR can be replaced by functions built in Prolog. That is, there is no need to prepare a special program to perform inferences.

DCKR will be described in detail in Section 2. Section 3 will discuss applications of DCKR to natural language processing, semantic processing and semantic matching algorithm. Programming efforts of semantic processing will be alleviated a lot if DCKR is used for the description of lexical items, since most of programming efforts can be left to the functions built in Prolog. In Section 4, a method is discussed to increase the execution speed of DCKR.

## 2. Knowledge Representation in DCKR

### 2.1 Object representation and inference

The following examples of knowledge representation in DCKR will be used in Section 3 and later.

```
:-op(100,yfx,'-').
```

```
        op(100,yfx,':'),
        op(90,xfy,'#').

01) sem(clyde#1,age:5,_).
02) sem(clyde#1,P,S) :-
            isa(elephant,P,[ clyde#1|S ]).
03) sem(elephant#1,birthYear:1980,_).
04) sem(elephant#1,P,S) :-
            isa(elephant,P,[ elephant#1|S ]).
05) sem(elephant#2,birthYear:1982,_).
06) sem(elephant#2,P,S) :-
            isa(elephant,P,[ elephant#2|S ]).
07) sem(mcCarthy#1,address:stanford,_).
08) sem(mcCarthy#1,nationality:american,_).
09) sem(mcCarthy#1,P,S) :-
            isa(human,P,[ mcCarthy#1|S ]).
10) sem(mister5G#1,address:japan,_).
11) sem(mister5G#1,P,S) :-
            isa(human,P,[ mister5G#1|S ]).
12) sem(misterAI#1,address:america,_).
13) sem(misterAI#1,P,S) :-
            isa(human,P,[ misterAI#1|S ]).
14) sem(human,P,S) :-
            isa(mammal,P,[ human|S ]).
15) sem(elephant,P,S) :-
            isa(mammal,P,[ elephant|S ]).
16) sem(mammal,bloodTemp:warm,_).
17) sem(mammal,P,S) :-
            isa(animal,P,[ mammal|S ]).
18) sem(animal,P,S) :-
            isa(creature,P,[ animal|S ]);
            hasa(body,P,[ animal|S ]).
19) sem(creature,age:X,_) :-
            bottomof(S,B),
            sem(B,birthYear:Y,_),
            X is 1985 - Y.
20) sem(america,P,S) :-
            isa(country,P,[ america|S ]);
            hasa(california,P,[ america|S ]).
            ...... .
21) sem(california,P,S) :-
            isa(state,P,[ california|S ]);
            hasa(stanford,P,[ california|S ]).
            ...... .
```

Now the meanings of the sem, isa and hasa predicates, which are important to descriptions in DCKR, are explained using the DCKR examples given above.

The first argument in the sem predicate is the Object name. Objects are broadly divided into two types, individuals and prototypes. Psychologists often refer to prototypes as stereotypes. An Object name with # represents an individual name and the one without #, a prototype name. For example, clyde#1 and elephant, which appears in 01) and 02), represent an individual name and a prototype name, respectively. A set of Horn clauses headed by the sem predicate with the same individual name represents an individual. A set of Horn clauses headed by the sem predicate with the same prototype name represents a prototype. Therefore, the Object representation by DCKR (in a Horn clause form) can be completely compiled. Knowledge compilation leads to high speed.

The second argument in the sem predicate is a pair composed of a slot name and a slot value. For example, the description in 01) indicates the fact that the age of the individual clyde#1 is 5. And the age is a slot name and 5 is a slot value. A pair composed of a slot name and a slot value is hereinafter called an SV pair.

The description in 02) is to be read as showing that clyde#1 is an instance of the prototype elephant. Here, note that 02) is a direct description of inheritance of knowledge from prototypes at higher level. 02) means that if a prototype called elephant has a property P, the individual clyde#1 also has the property P. 14) and 17) describe the fact that a human is a mammal and that a mammal is an animal. Also, note that inheritance of knowledge is automatically performed by the unification built in Prolog. 18) describes the fact that an animal is a creature and has a body. From the foregoing it can be seen that the isa predicate used for the inheritance of knowledge is a predicate for traversing the hierarchy of prototype Objects. The predicates, isa, hasa and bottomof are defined below.

```
22) isa(Upper,P,S) :-
        P = isa:Upper;
        sem(Upper,P,S).
23) hasa(Part,X:Y,S) :-
        X == hasa,
        (Y = Part;
         sem(Part,hasa:Y,S)).
24) bottomof([B|T],B) :-
        (var(T);atomic(T)),!,nonvar(B).
25) bottomof([H|T],B) :-
        bottomof(T,B).
```

The hasa predicate is used for the inheritance of knowledge through part-whole relations.

Let us look back at the description of 02) from a different perspective. 02) can be regarded as a description for calling the world of prototypes from the world of individuals and extract the information held by prototypes. In DCKR, once an entry is made into the world of prototypes by means of the isa predicate, it is possible to access all prototypes existing in the world of prototypes.

Since, however, individuals are dynamically produced, it is impossible to know about the world of individuals beforehand. DCKR is provided with the bottomof predicate, which is used in the body of 19) and is defined by 24) and 25), as a means for gaining knowledge of the world of individuals from the world of prototypes. By using the predicate, it is possible to know what the calling individual (the individual that called the world of prototypes) is and extract the knowledge held by that individual. This is accomplished by using the third argument in the sem predicate, since in the third argument of the sem predicate is stacked the route followed in tracing the hierarchy.

For example, 19) identifies the caller B by means of the bottomof predicate and calculates his age by using B's birthyear. Therefore, if

```
    ?-sem (elephant#1,age:X,_)
```

is executed, 19) is reached by the isa predicate in 04), 15), 17) and 18). As a result,

```
    X=5
```

is derived.

Also, if

```
    ?-sem(elephant#1,P,_)
```

is executed, a succession of pieces of knowledge about elephant#1 can be obtained as follows:

```
    P = birthYear:1980;
```

```
        P = isa:elephant;
        P = isa:mammal;
        P = bloodTemp:warm;
        P = isa:animal;
        P = isa:creature;
        P = age:5
```

Note that all knowledge (SV pair: property) at higher level prototypes than elephant#1 is obtained through the unification mechanism of Prolog. In other words, inheritance of knowledge is carried out automatically by the functions built in Prolog.

As you may notice, if

```
        ?-sem (X, Y, _)
```

is executed, the system begins calculating all knowledge it has (as X-Y pairs).

If

```
        ?-sem (X, isa:mammal, _)
```

is executed by utilizing the features of Prolog, it is possible to access an individual or prototype at the lower level from a mammal at the higher level. However, this is not always executed efficiently. For this good can be unified with all heads of Horn clauses which perform inheritance of knowledge as in 02). Since many of them end in failure, the cost of computation increases with the number of Horn clauses which perform inheritance of knowledge. This problem will be addressed again in Section 3, and a possible solution presented. Finally, to check the function of the hasa predicate, you may execute

```
        ?-sem (america, hasa:X, _).
```

From the foregoing explanation, you will understand what the descriptions of 07) and later are like and that there is no need whatever for an inference program. If only knowledge is described in DCKR, inference is automatically performed by the interpreter built in Prolog. Knowledge described in DCKR seems easy to read. This also leads to ease of describing knowledge.

## 2.2 General knowledge representation and inference

In the example of Object descriptions in DCKR given in 2.1, an Object was represented as a set of Horn clauses headed by the sem predicate (which has an Object name as the first argument). And the Object name was always a constant (representing an individual or prototype). By contrast, knowledge in which the first argument in the sem predicate is a variable representing an individual sometimes plays an important role in DCKR. Such a variable is hereinafter called an individual variable.

Generally, an individual variable is represented, for instance, as A#B. A DCKR expression headed by the sem predicate which has an individual variable as the first argument functions as an inference rule which creates new knowledge mainly from existing knowledge.

Let us take up an example and describe it in DCKR to find how it works. The DCKR description corresponding to the sentence "Everyone who lives in stanford is a professor" is as follows:

```
26) sem(X#J, profession:professor, _) :-
          sem(X#J, isa:human, _),
          sem(X#J, address:stanford, _).
```

Here X#J represents an individual variable. 07) has no description related to the profession of mcCarthy#1. Under the inference rule of 26), however, executing the following goal, which corresponds to the question "What is the profession of mcCarthy#1?".

```
      ?-sem(mcCarthy#1.profession:A,_)
```

can get the following:

```
      A = professor
```

DCKR inferences can be also carried out by functions built in Prolog.

## 3. DCKR Applications to natural language processing

This section explains a method of semantic processing of natural language and semantic pattern-matching algorithm as applications of DCKR. The effectiveness of DCKR is also discussed.

### 3.1 Semantic processing of natural language

Semantic processing is one of the important tasks for natural language processing. Basic to semantic processing are descriptions of lexical items. The most frequently used form of description of lexical items is probably Frames or Objects. A method of the Object representation in Prolog called DCKR is introduced in section 2. In this section, it will be shown that DCKR representation of lexical items enables to alleviate a lot of programming efforts of semantic processing.

#### 3.1.1 Descriptions of lexical items in DCKR

Basic to semantic processing are descriptions of lexical items. The most frequently used form of description of lexical items is probably frames (Objects). In DCKR, an Object consists of a set of slots each of which is represented by a Horn clause headed by the sem predicate. However, the first argument in the sem predicate is the Object name. The values of slots used in semantic processing are initially undecided but are determined as semantic processing progresses. This is referred to as slots being satisfied by fillers. To be the value of a slot, a filler must satisfy the constraints written in the slot.

If the filler satisfies the constraints written in a slot, action is started to extract a semantic structure or to make a more profound inference. Constraints written in slots are broadly divided into two, syntactic constraints and semantic constraints. The former represent the syntactic roles to be played by fillers in sentences. The letter are constraints on the meaning to be carried by fillers. Typical semantic processing proceeds roughly as follows:

i) If a filler satisfies the syntactic and semantic constraints on a slot selected, start action and end with success. Else, go to ii)

ii) If there is an another slot to select, select it and go to i). Else, go to iii)

iii) If there is a higher-level prototype, get its slot and go to i). Else, and on the assumption that the semantic processing is a failure.

From the semantic processing procedures in i) through iii) above, the following can be seen:

a) The semantic constraints in i) are often expressed in logical formulas. This can be easily done with DCKR as explained later.
b) The slot selection in ii) can use the backtracking mechanism built in Prolog. For in DCKR a slot is represented as a Horn clause.
c) iii) can be easily implemented by the knowledge inheritance mechanism of DCKR explained in 2.1.

Thus, if lexical items are described in DCKR, programs central to semantic processing can be replaced by the basic computation mechanism built in Prolog. This will be demonstrated by examples below. Cited first is a DCKR description of the lexical item 'open' [ Tanaka 85a ].

```
(30) sem(open,subj:Filler^In^Out,_) :-
         sem(Filler,isa:human,_),
         extractsem(agent:Filler^In^Out);
         (sem(Filler,isa:eventOpen,_);
             sem(Filler,isa:thingOpen,_)),
         extractsem(object:Filler^In^Out);
         sem(Filler,isa:instrument,_),
         extractsem(instrument:Filler^In^Out);
         sem(Filler,isa:wind,_),
         extractsem(reason:Filler^In^Out).
(31) sem(open,obj:Filler^In^Out,_) :-
         (sem(Filler,isa:eventOpen,_);
             sem(Filler,isa:thingOpen,_)),
         extractsem(object:Filler^In^Out).
(32) sem(open,with:Filler^In^Out,_) :-
         sem(Filler,isa:instrument,_),
         extractsem(instrument:Filler^In^Out).
(33) sem(open,P,S) :-
         isa(action,P,[ open|S ]);
         isa(event,P,[ open|S ]).
```

30),31) and 32) are slots named subj, obj and with, which constitute open. Variable Filler is the filler for these slots. The slot names represent the syntactic constraints to be satisfied by the Filler. Subj, obj and with show that the Filler must play the roles of the subject, object, and with-headed prepositional phrase, respectively, in sentences. The body of each of the Horn clauses corresponding to the slots describes a pair composed of semantic constraint and action (hereinafter called an CA(Constraint-Action) pair). For example, the body of 30) describes four CA pairs, each of them joined by or(';').

The first CA pair:

```
sem(Filler,isa:human,_),
extractsem(agent:Filler^In^Out);
```

shows that if the Filler is a human, extractsem(agent:Filler^In^Out), action to make the deep case of the Filler the agent case, is started to extract a deep case structure. Here, sem(Filler,isa:human,_), which checks if the Filler is a human, represents a semantic constraint on the Filler. Predicate extractsem returns the extracted deep case structure with results added to In sent to Out.

As described above, checking semantic constraints can be replaced by direct Prolog program execution. Therefore, relatively complex semantic constraints, e.g., person of blood type A or AB, can be easily described as shown below:

```
sem(Filler,isa:human,_),
   (sem(Filler,boodType:a,_);
    sem(Filler,boodType:ab,_))
```

The second SA pair:

```
  (sem(Filler, isa:eventOpen,_);
          sem(Filler, isa:thingOpen,_)),
  extractsem(object:Filler^In^Out);
```

shows that if the Filler is an even which opens (eventOpen) or a thing which opens (thingOpen), its deep case is made the object case.

The third CA pair:

```
  sem(Filler, isa:instrument,_),
  extractsem(instrument:Filler^In^Out);
```

indicates that if the Filler is an instrument, its deep case is made the instrument case.

The fourth CA pair:

```
  sem(Filler, isa:wind,_),
  extractsem(reason:Filler^In^Out).
```

shows that if the Filler is wind, its deep case is made the reason case.

Form the foregoing explanation, the meaning of the slots in 31) and 32) will be evident. In addition to 'with', there are many slots corresponding to prepositional phrases, but they are omitted to simplify the explanation.

33) shows that if the Filler cannot satisfy the slots in 30), 31) and 32), the slots in the prototype action or event is accessed automatically by backtracking. This was explained in detail as inheritance of knowledge in 2.1. and provides an example of multiple inheritance of knowledge as well.

The descriptions of 30) through 33) can be completely compiled, thus ensuring higher speed of processing. This makes a good contrast with most conventional systems which cannot compile a description of lexical items because it is represented as a large data structure.

### 3.1.2 Description of grammar rules

The DCG notation [Pereira 80] is used to describe grammar rules. Semantic processing is performed by reinforcement terms in DCG. An example of a simple grammar rule to analyze a declarative sentence is given below.

```
  sdec(SynVp, SemSdec) -->
     np(SynSubj, SemSubj),
     vp(SynVp, SemVp),
     {concord(SynSubj, SynVp),
      seminterp(SemVp, subj:SemSubj, SemSdec)}.
```

The part encircled by { } is a reinforcement term. The predicate concord is to check concord between subject and verb. The predicate seminterp, intended to call sem formally, is a small program of about five lines. In this example the grammar rule checks if the head noun in SemSubj can satisfy the subj slot of the main verb frame (e.g., open in 30) -33)) in SemVp and returns the results of semantic processing to SemSdec. Therefore, we can see that there is little need to prepare a program for semantic processing.

As semantic processing is performed by reinforcement terms added to DCG, syntactic processing and semantic processing are amalgamated. This has been held to be a psychologically reasonable language-processing model.

### 3.1.3 Test result

Some comments will be made on the results of semantic processing based on the concept explained in 3.1.1 and 3.1.2. The sentence used in the semantic processing is "He opens the door with a key."

```
Input sentences
|: He opens the door with a key.

Semantic structure is:

sem(open#5,P,S) :- isa(open,P,[ open#5|S ]).
sem(open#5,agent:he#4,_).
sem(open#5,instrument:key#7,_).
sem(open#5,object:door#6,_).
sem(he#4,P,S) :- isa(he,P,[ he#4|S ]).
sem(door#6,P,S) :- isa(door,P,[ door#6|S ]).
sem(door#6,det:the,_).
sem(key#7,P,S) :- isa(key,P,[ key#7|S ]).
sem(key#7,det:a,_).
```

Besides, results of semantic processing of "the door with a key" are obtained but their explanation is omitted.

Here it is to be noted that results of semantic processing are also in DCKR form. By obtaining semantic processing results in DCKR form, it is possible to get, for example,

sem(open#J,instrument:X,_)

from the interrogative sentence "With what does he open the door?" and get the answer

X=key#7

by merely executing that.

### 3.1.4 DCKR and natural language understanding system

Now the relationship between DCKR and a natural language understanding system will be touched on. From what has no far been discussed, we can envision a natural-language-understanding system architecture as illustrated in Fig. 1.

```
                        --------------------
     ------------------|Semantic Structure|---------------------
     |                 |     (DCKR)       |                    |
     |                 --------------------                    |
     |                          |                              |
     |        -----------------------------------------        |
----------    |             Inference                  |       |
|        | |  |             Engine                     |       |
|        | |  |  ---------------------------------     |       |
|Context | |  |  |            |                  |     |       |
|Analysis| |  |  |  ------------------------     |     |       |
|        | |  |  |  |Knowledge Base(DCKR + ?)|   |     |       |
|        | |  |  |  |                        |   |     |   ----------
|Semantic|-|  |  |  |                        |   |     |   |Sentence|
|Analysis| |  |---|  |  ------------------    |   |     |---|Genera- |
|        | |  |  |  | | Dictionary(DCKR)|    |   |     |   |tion    |
|        | |  |  |  | |                 |    |   |     |   ----------
|Syntactic|-| |  |  | |-----------------|    |   |     |       |
|Analysis| |  |  |  | | Grammar         |    |   |     |       |
|        | ----  |  | |                 |    |   ------        |
-----------      --------------------------        |           |
     |                                             |           |
     |                                             |           |
 Input Sentence(Source)                         Answer     (Target)
```

Fig. 1 DCKR and Natural-Language-Understanding System

The shaded parts in Fig. 1 are those will be achieved by the interpreter built in Prolog. From the foregoing explanation, it will be seen that if part of general knowledge and a dictionary are described in DCKR, part of context-processing and the greater part of semantic processing can be left to the functions built in Prolog. As for syntactic processing, the grammar rules described in DCG [ Pereira 80 ] automatically converted into a Prolog program, and parsing can be replaced by Prolog program execution. As shown in Fig. 1, therefore, syntactic processing can be left almost in its entirety to the Prolog interpreter. There is no need to prepare a parser [ Tanaka 84 ].

Given the foregoing facts and assuming the inference engine to be the Prolog interpreter, it may be concluded that a Prolog machine plus something else will be a natural-language-processing machine. If asked what that something will be, we might say that it will be a knowledge base machine. Anyway, this concept is in line with what the Japanese fifth-generation computer systems project is aimed at.

## 3.2 Implementation of Semantic Matcher

### 3.2.1 Semantic Matching

Various Objects must be treated in the field of natural language processing. And there often arises a need for pattern matching between Objects. For example, identifying anaphora and recognising coordinate components in a coordinate structure will need semantic pattern matching. Carried further in this direction, it was developed a language employing unifications of Objects in its basic computation mechanism [ Mukai 85 ] but it was limited to the level of syntax.

One problem in unification of Objects is that since there are no constraints on the order in which slots constituting Objects are arranged, the unification must be independent of the order in which slots are arranged. This gives rise to the problem of computation cost when an Object is represented as a big data structure (e.g., a list structure). This problem is somewhat alleviated by DCKR: Since a slot is represented as a Horn clause, the slot selection required for unification can be left to the backtracking function built in Prolog. Identifying anaphora and coordinate components require semantic pattern matching which is not limited to the level of syntax.

Meantime, the importance of judging identity between Objects and unifying Objects with regard to semantics is discussed as Forced Matching by [ Bobrow 77 ], as Semantic Matching by [ Nilsson 80 ], and as the attempt to expand the unification function of Prolog by Colmeraure. Here a DCKR-based method for unification of Objects will be discussed against the background of the relatively simple linguistic knowledge explained in Section 2. In the latter case, it will be necessary to know what the body of knowledge described in DCKR is like. Therefore, the algorithm discussed here uses the following meta knowledge.

```
34) metakb(mammal,ntype:exclusive).
35) metakb(age,ltype:exclusive).
36) metakb(address,ltype:exclusive).
```

34) shows that prototypes (e.g., human and elephant) immediately below the prototype mammal are mutually inconsistent. 35) and 36) mean that slots with different slot values for age and address, respectively, are mutually inconsistent. In other words, 34) shows that an individual that is a human cannot be unified with an individual that is an elephant, while 35) indicates that an individual aged 44 cannot be unified with an individual aged 55. Now some examples of Semantic

Matching are given. From the DCKR descriptions in Section 2, we can easily derive the following inferences i) - iv).

i) Address of mcCarthy#1 is stanford and address of misterAI#1 is america. Since we know that stanford lies within america, we can infer that it is not inconsistent to identify and unify misterAI#1 with mcCarthy#1.
ii) Likewise, we can infer that since america and japan are two different countries, it is impossible to identify mister5G#1 with mcCarthy#1.
iii) If the present year minus the birth year represents age, we can infer that clyde#1, aged five, may be unified with elephant#1 born in 1980 (assuming the present year is 1985).
iv) By similar reasoning we can conclude that clyde#1 cannot be unified with elephant #2.
v) mcCarthy#1 cannot be unified with clyde#1 because the former is a human and the latter an elephant.

The unification of two Objects by considering their meanings is called Semantic Matching (Forced Matching). And a program to perform Semantic Matching is called a Semantic Matcher.

### 3.2.2 Algorithm for Semantic Matchers

While the need for Semantic Matchers has often been discussed, there have not been many attempts made to prepare such programs. This is presumably because even a semantic match of the level illustrated above would be very complex. With DCKR, however, it is relatively easy to prepare Semantic Matchers. By using the algorithm shown in [ A ] through [ F ] below, we can prepare a Semantic Matcher with the level of unification capability illustrated in i) through v) above, though it is not a perfect program.

[ A ]: If there is a higher-level Object (Oi) common to two individuals o#1 and o#2 considered for unification, get the Object and go to [ B ]. Else , go to [ D ].
[ B ]: If metakb(Oi, ntype:exclusive) holds (Objects one level below Oi are mutually exclusive), go to [ C ]. Else, go to [ A ].
[ C ]: If there are two different Objects Oj and Ok just one level below of Oi, and Oj and Ok are positioned above o#1 and o#2, respectively, return on the assumption that the unification attempt is unsuccessful. Else, go to [ A ].
[ D ]: If o#1 has the SV pair Ax:Bx and o#2, the SV pair Ax:By, form the set S shown below and go to [ E ]. (Note that the slot name of two SV pair is the same.) Else, go to [ F ] on the assumption that o#1 and o#2 can be unified because there is no positive reason prohibiting it.

```
S = { (Ax:Bx, Ax:By) | metakb(Ax, ltype:exclusive),
                         (Bx == By ;
                          sem(Bx, isa:By,_);
                          sem(By, isa:Bx,_);
                          sem(Bx, hasa:By,_);
                          sem(By, hasa:Bx,_) )  }
```

[ E ]: If S is not an empty set, go to [ F ] on the assumption that unification is possible. If S is an empty set, return on the assumption that the unification attempt is unsuccessful.

[ F ]: If o#1 and o#2 can be unified by the algorithm given in [ A ] through [ E ], assert the following facts to unify o#1 and o#2.

```
    sem(o#1,P,S) :- isa(o#2,P,[ o#1|S ]).
    sem(o#2,P,S) :- isa(o#1,P,[ o#2|S ]).
```

In this way, o#1 and o#2 automatically inherit each other's properties and are thereby unified.

### 3.2.3 Experiments of semantic matchings

The algorithm explained in [ A ] through [ F ] is realized by about 40 lines of predicates called mkeq. Test examples are given below.

```
(a) ?-mkeq(x#1,y#1).
(b) yes
(c) ?-mkeq(x#1,misterAI#1).
(d) yes
(e) ?-sem(y#1,P,_).
(f) P = isa:x#1;
(g) P = isa:misterAI#1;
(h) P = address:america;
(i) P = isa:human;
(j) P = isa:mammal;
(k) P = bloodTemp:warm;
(l) P = isa:animal;
(m) P = isa:creature;
(n) no
(o) ?-mkeq(mcCarthy#1,mister5G#1).
(p) no
(q) ?-mkeq(mcCarthy#1,misterAI#1).
(r) yes
(s) ?-mkeq(mcCarthy#1,clyde#1).
(t) no
(u) ?-mkeq(elephant#1,clyde#1).
(v) yes
(w) ?-mkeq(elephant#2,clyde#1).
(x) no
```

(a) creates two Objects called x#1 and y#1 and makes them equal [ D ]. (c) makes x#1 and misterAI#1 equal [ D ]. Therefore, if the properties of y#1 is asked in (e), it can be seen from responses (f) through (m) that y#1 has inherited the properties of misterAI#1 [ F ]. Responses to (o),(q),(u) and (w), based on [ E ], provide examples of Semantic Matching explained in ii), i), iii) and iv), respectively. The response to (s), based on [ C ], provides an example of Semantic Matching explained in (v). Here, it is to be rated that the correct responses are shown, through (u) and (w) give no description of the age of either elephant#1 or elephant#2. The reason has already been explained in Section 2.

## 4. Speedup of Execution Time

In the above sections, we have explained DCKR is a powerful and flexible formalism to express many sorts of knowledge. Inferences on the knowledge in DCKR are directly carried out by Prolog interpreter.

However, there is a problem we did not mentioned in the preceding paragraphs.

As all pieces of knowledge is expressed by a set of Horn clauses headed by the same "sem" predicate, the order of retrieval time will be problematic. The reader can understand the situation when considering a following example: If a goal "sem(X, isa:mammal,_)" will be executed, almost all of "isa" link knowledge will be invoked once. The reason is that the first argument of "sem" is a variable. (See 02, 04 and 06 in the section 2.) Note that the above goal statement forced us to traverse "isa" links in a reverse way, namely from superordinate to subordinate. In such a case, it is much time consuming.

Degradation of retrieval speed will cause a serious problem if we are going to build a large knowledge base. However the situation will be alleviated if we can use Quintus Prolog. Instead of using a "sem" predicate, we can use a "record" predicate that creates an internal database which gives us a fast look-up method. However, naive transformation of "sem" into "record" unables to solve the problem mentioned before.

One of the easest solutions is to create two types of knowledge in our internal database, one of which expresses a normal "isa" link, and the other expresses a reversal of "isa" link record. Now, a top level predicate "sem" should be changed: If the first argument of "sem" is not a variable, then follows a normal "isa" link record, else follows a reversal of "isa" link record.

The results of our experiment suggest that the retrival time is not exponetially but linealy increased when size of internal datasbase increases.

## 5. Conclusion

To understand discourse, which consists of a chain of sentences, it is necessary to infer which of the many Objects arising as the discourse proceeds are the same as which other. A typical example is anaphora in linguistics. For instance, that "oxygen" and "gas" appearing in the passage (discourse)

"... oxygen was generated. The gas ....

are the same things will be known by a Semantic Matchers is a long-term R & D challenge. Therefore, the method discussed in 3.1 is no more than a small step toward a solution to problems of that sort.

Semantic Matchers are expected to be applicable to the problems of Analogical Reasoning and Leaning which will assume growing importance in the research field of artificial intelligence in the future.

Chunking, of knowledge was cited as an advantage of knowledge representation in frame form: Chunking was considered convenient for association since it permits obtaining all knowledge (slots) related to a frame by merely accessing the frame. It was also said to be a psychologically reasonable memory model.

By contrast, knowledge representation in DCKR regards all slots existing in the world as standing on an equal footing instead of framing related slots to differentiate then from others. On the face of it, this is inconsistent with the frame concept. Since, however, related knowledge can be quickly brought in by hushing the first argument (Object name) in the sem predicate heading a Horn clause which corresponds to a slot, the frame concept can be easily simulated.

Fortunately, Prolog is provided with a setof and bagof predicates to extract all related knowledge as a list. These predicates could be utilized for that purpose. At the end of 2.1 we touched on the ease of writing and reading knowledge in DCKR. But we should develop a higher-level knowledge representation language. For instance, the third argument in the description of 02) should be automatically added in the process of compiling such a high-level knowledge representation language. Also. the variables In and Out appearing in the descriptions of 33) through 36). Thinking this way. we can see that representation in DCKR is. as it were. representation in machine language. It is necessary to develop a higher-level knowledge representation language regarding DCKR as a machine language.

Finally. knowledge representation has a multitude of difficult problems to be solved. such as how to represent high-order knowledge. negative knowledge or mathematical concept of sets and how to achieve default reasoning. The authors wish to get down to research in natural-language-understanding systems. In the process they will probably encounter various unexpected problems. Then will come the real test of DCKR.

[ Acknowledgment ]

Authors wish to express their great gratitude to Dr.Kazuhiro Fuchi. the director of the Research Center of ICOT. and Dr. Koichi Furukawa. the chief of the Research Center of ICOT. for their encouragements and valuable comments. Mr.Haruo Koyama.Mr.Manabu Okumura. Mr.Teruo Ikeda. Mr.Tadashi Kamiwaki. who are students of Tanaka Lab. of Tokyo Institute of Technology. helped us to implement some application programs based on DCKR.

## 6. References

[ Bobrow 77 ] Bobrow.D.G. et.al. : An Overview of KRL-0. Cognitive Science. 1. 1. 3-46(1977).

[ Bowen 85 ] Bowen.K.A. : Meta-Level Programming and Knowledge Representation. Syracuse Univ.. (1985).

[ Colmeraure 78 ] Colmeraure.A. : Metamorphosis Grammer. in Bolc (ed):Natural Language Communication with Computers. Springer-Verlag 133-190(1978).

[ Goebel 85 ] Goebel. R. : Interpreting Descriptions in a Prolog-Based knowledge Representation System. Proc.of IJCAI'85. 711-716(1985).

[ Hayes 80 ] Hayes.P.J. : The Logic of Frame Conceptions an Text Understanding. Walter de Gruyer. Berlin. 46-61(1980).

[ Koyama 85 ] Koyama.H. and Tanaka.H. : Definite Clause Knowledge Representation. Proc.of LPC'85. 95-106(1985). in Japanese.

[ Matsumoto 83 ] Matsumoto.Y. et.al.:BUP--A Bottom-UP Parser Embedded in Prolog. NEW Generation Computing. 1. 2. 145-158 (1983).

[ Mukai 85 ] Mukai.K. : Unification over Complex Indeterminates in Prolog. Proc.of LPC'85. 271-278(1985).

[ Nilsson 80 ] Nilsson.N.J. : Principles of Artificial Intelligence. Tioga. (1980).

[ Pereira 80 ] Pereira.F. et.al: Definite Clause Grammar for Language Analysis --A Survey of the Formalism and a Comparison with Augmented Transition Networks. Artificial Intelligence. 13. 231-278(1980).

[ Tanaka 84 ] Tanaka.H. and Matsumoto.Y. : Natural Language Processing in Prolog. Information Processing. Society of Japan. 25. 12. 1396-1403(1984). in Japanese.

[ Tanaka 85a ] Tanaka.H. et.al: Definite Clause Dictionary and its Application to Semantic Analysis of Natural Language. Proc. of LPC'85. 317-328(1985). in Japanese.