DCKR - Knowledge Representation in Prolog and its Applications

by Hozumi Tanaka
Tokyo Institute of Technology

1. Introduction

Relationships between knowledge representation in predicate logic formulas and knowledge representation in Frames or Structured Objects are clarified by [Hayes 80], [Nilsson 80], [Goebel 85], [Bowen 85], et al. but their methods require separately an interpreter for their representation. [Nilsson 80] transforms knowledge representation in predicate formulas form into Object form which is finally transformed into a semantic network. An inference on a network is reduced as the operations which traverse it.

The authors have developed a knowledge representation called DCKR (Definite Clause Knowledge Representation) [Koyama 86]. In DCKR, the slots consisting of a Structured Object (hereinafter called an object) is represented by a Horn clause (a Prolog statement with the "sem" predicate (to be explained later)) 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 suggest a method to do efficient inference in DCKR to natural language processing, semantic processing and semantic matching algorithm. Section 4 is about the level of knowledge representation in machine language, and it indicates that a high level knowledge representation language is needed. SRL/O will be proposed as a high level knowledge representation language that is translated into DCKR forms. Finally we explain how temporal knowledge is described by DCKR.

2. Knowledge Representation in DCKR

2. 1 Object representation

The following are simple examples in DCKR.

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

1.1) sem(clyde#1,age:6,_).
1.2) sem(clyde#1,P,S) :-
          isa(elephant,P,[elephant#1|S]).
2.1) sem(elephant#1,birthYear:1980,_).
2.2) sem(elephant#1,P,S) :-
```

```
            isa(elephant,P,[ elephant#1|S ]).

  3.1) sem(elephant,color:gray,_).
  3.2) sem(elephant,P,S) :-
                 isa(mammal,P,[ elephant|S ]).

  4.1) sem(mammal,bloodTemp:warm,_).
```

The meanings of the predicates isa and hasa will be presented below.

```
  a.1) isa(Upper,P,S) :-
                 P = isa:Upper;
                 sem(Upper,P,S).
```

The first arguments in the sem predicate is the Object name. For the examples 1.1) and 1.2), 2.1) and 2.2), 3.1) and 3.2), and 4.1) the object descriptions are respectively clyde#1, elephant#1, elephant, and mammal. 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 in 1.1) and 1.2) is an individual name, while elephant in 3.1) and 3.2) is a prototype name. 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) 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 of 1.1) indicates the fact that the age of the individual clyde#1 is 6. Age is the slot name and 6 is the slot value. A pair composed of a slot name and slot value is hereinafter called an SV pair.

2.2 Inheritance of knowledge and inference

The description in 1.2) is to be read as showing that clyde#1 is an instance of the prototype elephant. Here, note that 1.2) is a direct description of inheritance of knowledge from prototypes at a higher level. From the prespective of the inheritance of knowledge, 1.2) is read that clyde#1 has property P if elephant has the property P through isa predicate. For example, in the description of 1.2) the individual clyde#1 becomes the instance of the prototype elephant. The inheritance of knowledge is automatically performed by the unification built in Prolog. When the following statement ?-sem(elephant#1,P,_). is carried out, the following statements which are related to elephant#1 are produced one by one

```
      P = birthyear:1980;
```

```
P = isa:elephant;
P = color:gray;
P = isa:mammal;
P = bloodTemp:warm;
```

Note that all the information is from the higher levels of elephant#1. When

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

is executed, the system outputs all the pieces of information in the form of X and Y pairs.

If

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

is executed by utilizing the features of prolog, the following information on the prototype and the individual located below mammal is produced.

```
X = clyde#1;
X = elephant#1;
X = elephant;
```

The individual and the prototype which are located in a position lower than mammal can be known.

In the description in 1.2), the prototype environment is called from the individual one. According to 1.2) once one enters the prototype environment, one can access all knowledge which exists there. On the one hand, because the individual is a dynamically made object, the individual environment can be understood beforehand. There is a predicate instance which is used for finding out information about the individual environments from the prototype.

```
b.1) instance([Instance|Y],Instance) :-
                 (var(Y);atomic(Y)),!,nonvar(Instance).
b.2) instance([X|Y],Instance) :-
                 instance(Y,Instance).
```

Clarifying the descriptions of 1.2),2.2),3.2) etc., the third argument of the predicate sem stacks the route followed in tracing the isa hierarchy. This is given to the first argument of instance and informs what is the individual (instance) that called the prototype environment. It can expose the knowledge which is held by the individual. For example, there is a description for mammal in 4.1) and also for 4.2).

```
4.1) sem(mammal, bloodtemp : warm, _).
4.2) sem(mammal,age : X,S) :-
                instance(S,Instance),
             sem(Instance,birthyear:Y,_),
                  X is 1986 - Y.
```

The body in 4.2) uses the predicate instance and the Instance of mammal is known, and the from birthyear in Instance, the age is calculated. Therefore, when

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

is executed,

```
      X = 6
```

is produced.

The body in 4.2) can be regarded as showing the if-needed method used to determine age. For the part-whole relation, there is a transition rule. The predicate hasa in DCKR is, for this purpose, defined below.

```
 c.1) hasa(Part,X:Y,S) :-
                 X == hasa,
                 (Y = Part;
                  sem(Part,hasa:Y,S)).
```

The following examples show how hasa works.

```
 5.1) sem(america,capital:newYork,_).
 5.2) sem(america,climate:temperate,-).
 5.3) sem(america,P,S) :-
      T = [america|S],
      isa(country,P,T);
      hasa(california,P,T);
   .....................

 6.1) sem(california,P,_) :-
      T = [california|S],
      isa(state,P,T);
      hasa(stanford,P,T);
   .....................
```

After the above definitions are given and this statement is executed,

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

the following information is produced relating to america.

```
      X = california;
      X = stanford;
      .........
```

Finally, an exercise from [Sugawara 85] will be described. That is, "If the climate of a certain place is not known, the higher level of the place is found and the climate there is used." Note how, for example, definitions are made in 7.1),

```
 7.1) sem(state,climate:X,S) :-
                instance(S,Instance),
                sem(Superpart,hasa:Instance,_),
                sem(Superpart,climate:X_).
```

When

```
?- sem(california,climate:X,_).
```

is executed, state located at the higher level of california is reached and from the description in 7.1), the value of climate at superpart of california, america is taken and the following response is produced.

```
X = temperate
```

Lastly, to expand the definition of a.1), the scope of the category (the scope of the inheritance of knowledge) can be restrained by using the third argument.

2.3 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. An example from [Nilsson 80] is easily expressed in DCKR form.

```
8.1) sem(X#J,worksin:Y#K,_) :-
                sem(Y#K,isa:department,_),
                sem(Y#k,manager:X#J,_).
```

8.1) shows that Y#K is a department, X#J is its manager, and that X#J worksin Y#K. Examine the following

```
 9.1) sem(joeSmith#1, worksIn:pd#1,_).
10.1) sem(pd#1,manager:joeJones#1,_).
10.2) sem(pd#1,P,S) :-
                isa(department,P,[pd#1|S]).
```

Here the following goal corresponds to the question "Who work in pd#1?"

```
?-sem(A#B,worksIn:pd#1,_).
```

and the following is produced.

```
A = joeSmith
B = 1;
A = joeJones
B = 1
```

Using 10.1), because the manager of pd#1 is joeJones#1, the fact "joeJones#1 worksin pd#1" can be output.

Finally, in order to confirm the validity of DCKR, please compare the production rule of section 9.4.6 in [Nilsson 80] and the description in 8.1). From this, at the same time that the description of DCKR is simple to understand, the ability of substituting DCKR inference for builtin Prolog functions will be easily understood.

3. DCKR Applications to semantic processing of natural language

3.1 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 refer'ed 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 roles to be played by fillers in sentences. The latters 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 another slot to select, select 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 machanism 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 85].

```
11.1) sem(open, subj:Filler^In^Out,_) :-
          sem(Filler, isa:human,_),
          addProp(agent:Filler^In^Out);
          (sem(Filler, isa:eventOpen,_); sem(Filler, isa:thingOpen,_))
        addProp(object:Filler^In^Out);
        sem(Filler, isa:instrument,_),
          addProp(instrument:Filler^In^Out);
          sem(filler, isa:wind,_),
          addProp(reason:Filler^In^Out).
11.2) sem(open, obj:Filler^In^Out,_) :-
          (sem(Filler, isa:eventOpen,_);sem(Filler, isa:thingOpen,_)),
          addProp(object:Filler^In^Out).
11.3) sem(open, with:Filler^In^Out,_) :-
          sem(Filler, isa:instrument,_),
          addProp(instrument:Filler^In^Out).
11.4) sem(open, P, S) :-
          T = [ open|S ],
          isa(action, P, T);
          isa(event, P, T).
```

11.1), 11.2), and 11.3) 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. In other words, subj, obj, and with show that the Filler must play the roles of the subject, object, and with-headed prepostitional 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 11.1) 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.

The second SA pair:

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

shows that if the Filler is an event 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.

From the foregoing explanation, the meaning of the slots in 11.2) and 11.3) will be evident. In addition to "with", there are many slots corresponding to prepositional phrases, but they are omitted to simplify the explanation.

11.4) shows that if the Filler cannot saitsfy the slots in 11.1), 11.2), and 11.3), the slots in the prototype action or event is accessed automatically by backtracking. It is an example of multiple inheritance.

The descriptions of 11.1)-11.4) can be completely compiled, thus ensuring the higher speed of processing. This makes a good constrast with most conventional systems which cannot compile a description of lexical items because it is reprepresented as a large data structure.

3.2 Description of grammar rules

The DCG notation [Pereira 80] is used to describe grammar rules. Semantic processing is performed by augmented 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 surrounded by { } is an augmentation part. 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 Sem Subj can satisfy the subj slot of the main verb frame in Sem Vp and returns the results of semantic processing to SemSdec. There, we can see that there is little need to prepare a program for semantic processing.

3.3 Test results

Some comments will be made on the results of semantic actual processing based on the concept explained in 3.1 and 3.2. The sentence used in the semantic processing is "He opens the door with a key."

```
Input Sentences
I: 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(key#7,P,S) :- isa(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. When the above semantic structures are represented in LFG [Bresnan 81] form, the following results.

```
 _                                                     _
| open#5                                                |
|     prototype = open                                  |
|                 _                _                    |
|     agent =    | he#4             |                   |
|                |_   prototype = he _|                 |
|                     _                    _            |
|     instrument = |key#7                   |           |
|                  |    prototype = key     |           |
|                  |_   det = a            _|           |
|                    _                _                 |
|     object =      |door#6            |                |
|                   |    prototype =door|               |
|                   |_   det = the    _|                |
|_                                                     _|
```

Note that the results of semantic processing are also in DCKR form. By obtaining semantic structures 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 obtain the answer by means of builtin functions of Prolog.

```
J = 5
X = key#7.
```

3.4 DCKR and natural language understanding system

Now the relationship between DCKR and a natural language understanding system will be touched on. From what has so far been discussed, we can envision a natural language understanding system architecture as illustrated in Fig. 1.

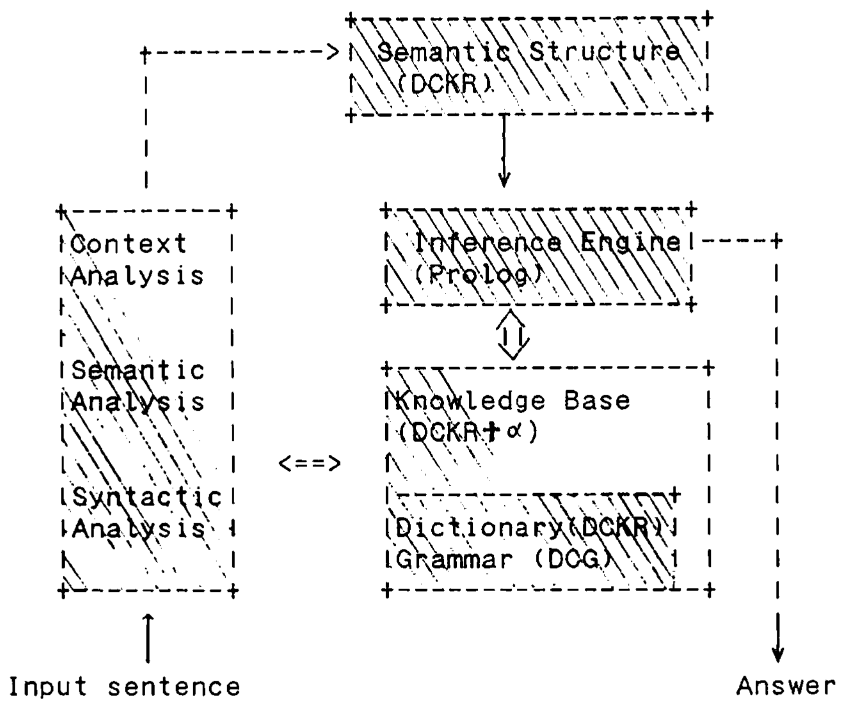

Fig. 1 DCKR and Natural Langugage Understanding System

The shaded parts in Fig. 1 are those that 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 builtin Prolog. As for syntactic processing, the grammar rules described in DCG [Pereira 80] are 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.

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.

4. High level knowledge representation in natural language SRL/0

It is thought that the knowledge description of DCKR, explained in Section 2, was not too difficult. However, the description of lexical items in Section 3 was not easy to grasp either. It suggests that DCKR-based representations are on the level of machine language representations. Here it was important for the authors to again develop a high level knowledge representation language. In this section for this purpose we propose the language SRL/0 [Okumura 86]. The DCKR representation and its equivalent information is given below with a SRL/0 description.

```
   clyde#1 ::
          [ age:6 ]
          [ isa:elephant ].
   elephant#1 ::
          [ birthYear:1980 ]
          [ isa:elephant ].
       elephant ::
          [ color:gray ]
          [ isa:mammal ].
   mammal ::
          [ bloodTemp:warm ]
          [ age:X
              where ( X = 1986 - birthYear!instance ) ].
```

Note here that where ... is the slot corresponding to the constraint, and that birthYear!instance is the value of the birthYear of the instance of mammal [Mukai 85].

```
   america ::
          [ climate:temperate ]
          [ isa:country ]
          [ hasa:california ]
          ..............

   california ::
          [ isa:state ]
          [ hasa:stanford ]
          .............

   state ::
          [ climate:X
              where
                  Superpart hasa:instance,
                  Superpart climate:X ].
```

And a general description of knowledge:

```
   X#J   ::
        [ worksIn:Y#K
            where Y#K isa:department,
                  Y#K manager:X#J ].
```

Below is a description of lexical items corresponding to 11. 1)-11. 4) but it is a relatively more precise description. The description of 11. 1)-11. 4), for instance "The conference opens with a key." would semantically present a problem for processing.

The description by SRL/0 below (Fig. 2), the condition for co-occurence of other slots is expressed as when object!instance isa:thingOpen. The character $ which comes immediately after the slot name expressed an unsatisfied slot. Under $ is the CA pair. A slot surrounded by [,] describes an neccesary deep case. A slot surrounded by (,) expresses one with an optional deep case. Please refer to [Okamura 86] to find out how these correspond to the DCKR representation and how it works.

```
     open ::
            [subj $ isa:human => agent]
            [obj $ isa:eventOpen;
                   isa:thingOpen => object]
            (with $ isa:instrument
                    when object!instance isa:thingOpen
                    => instrument)
            (with $ isa:animal => coagent)
          ::
            [subj $ isa:eventOpen;isa:thingOpen => object]
          ::
            [subj $ isa:instrument => instrument;
                    isa:wind => reason]
            [obj $ isa:thingOpen => object]
          ::
            ((at;in) $ isa:place => location).
```

Fig. 2 SRL/O description of an lexical item open

Here, "when object!instance isa:thingOpen", the value of the object slot of the instance open must be thingOpen. For the description that used when, when there is no value for the object slot of instance, demon is made and waited for until it is satisfied. After this, the demon is activated when the value is satisfied and after the semantic constraint is checked, failure or success is determined.

5. Temporal knowledge representation.

Temporal knowledge represented in DCKR will be discussed here. First we will consider how the state of the environment changes through time intervals. The time zone from Time1 to Time2 is expressed as Time1¯Time2. A point of time, Time1 is expressed as Time1¯Time1. The fact that holds Time zone Time1¯Time2 is expressed in the following manner.

```
     sem(<objname>,<property>,T1¯T2,TT1¯TT2) :-
                                  time(Time1¯Time2,T1¯T2,TT1¯TT2).
```

The predicate time takes the first two variables, Time1¯Time2 and T1¯T2, and produces the result in the third variable, TT1¯TT2 that is overlapping time zone of Time1¯Time2 and T1¯T2. If there is no overlapping, failure results. However, if T1 or Time1 remains as a variable, it is assumed to be infinite. Below is a simple example.

```
     sem(clyde#1,location:india,T,TT) :- time(1_3,T,TT).
     sem(clyde#1,location:japan,T,TT) :- time(4¯10,T,TT).
     sem(clude#1,location:india,T,TT) :- time(11¯12,T,TT).
```

The following is executed with the information given above.

```
?-sem(clyde#1, location:Location, 3~6, TT).
```

which can be expressed as "Where was clyde#1 during the time zone 3~6?" The results are

```
Location = india
T = 3~3 ;
Location = japan
T = 4~6
```

And for the following question,

```
?-sem(clyde#1, location:india, T, TT).
TT = 1~3;
TT = 11~12
```

can be output as the results.

In DCKR, "sometimes", "always", "next", and "until" are dealt as operators. In DCKR, as shown below, the definition of sometimes is given but for a more detailed time-related definition refer to the explanation in [Koyama 86].

```
sometimes sem(X, Y, T~T, TT) :- sem(X, Y, T~T1, TT).
```

Note that the present time T is given when a goal is executed, but the variable T1 of the body can be left in that state. The above difinition says that sometimes sem(...) holds if sem(X, Y, T~T1, TT) of the body is once hold after the present time T.

6. Conclusion.

As an advantage to knowledge represenatation of Frames, chunking of knowledge, could have been given. For chunking, to access only one frame, all the related information (slot) can at one time be gotten; it can be thought as having good circumstances.

In addition, it is asserted as being an appropriate psychological model. However, for the information representation given for DCKR up until this point, based on frames, there are not frames which differentiated between those slots which are surrounded and those which are not. All the slots which exist in the environment are considered equivalent. This might be seen as contradicting the one-glimpse frame idea. However, for the hushing of the predicate sem of the Horn clauses, the related knowledge can be easily taken in as can simulated of frame ideas. A fortunate aspect of Prolog is that all the related information can be listed by the predicates setof and bagof.

In Section 3, the applicated of DCKR to natural language processing was discussed. There is difficulty here specifically in regards to semantic processing. 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. It is necessary to develop a higher-level knowledge representation language regarding DCKR as a machine language. In the section 4, the authers propose a high level knowledge representation language called SRL/O.

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.

[Acknowledgements]

The 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 encouragement and valuable comments. Mr. Haruo Koyama, Mr. Manabu Okumura, Mr. Teruo Ikeda, Mr. Tadashi Kamiwaki, who are students of the Tanaka Lab at the Tokyo Institute of Technology, helped us to implement some application programs based on DCKR. Mr. Hideyo Imazu and Mrs. Sachie Saito helped in preparing this manuscript.

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).

[Bresnan 82] Bresnan, J. ed. : The Mental Representation of Grammatical Relations, The MIT Press(1982).

[Colmeraure 78] Colmeraure, A. : Metamorphosis Grammar, 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, in Metzing, D. ed. : Frame Conceptions and Text Understanding, Walter de Gruyer, Berlin, 46-61(1980).

[Koyama 86] Koyama, H. : Prolog niyoru Structured Object no Hyougenkeisiki to Suiron, Master's Thesis, Tokyo Institute of Technology(1986) in Japanese.

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

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

[Okumura 86] Okumura, M. : Imi Kizyutu-you Gengo SRL/O no Sekkei to DCKR. Jouhousyori Gakkai Jouhougaku Kiso Kenkyuukai, 1-2(1986) in Japanese.

[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).

[Sugahara 85] Sugahara, T. : Frame System niokeru Keisyou Kinou no Kakutyou, Tisiki Zyouhousyori Symposium Ronbunsyuu, Monbusyou Tagenn Tisiki Zyouhou Soukatuhan, 97-106(1985) in Japanese.

[Tanaka 85] Tanaka, H. et. al. : Definite Clause Dictionary--Prolog niyoru Zisyokoumoku Kizyutu to Imisyori, Proc. of LPC'85, ICOT, 317-328(1985) in Japanese.