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

Hozumi Tanaka

Tokyo Institute of Technology
Dept. of Computer Science
O-okayama, 2-12-1, Megro-ku
Tokyo, Japan

ABSTRACT: Semantic processing is one of the important tasks for natural language processing. Basic to semantic processing is 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 semantic intepretations relative to lexical items described in DCKR can be replaced by functions built in Prolog. That is, most of programming efforts of semantic processing can be left to the functions built in Prolog.

DCKR will be described in detail in Section 2. Section 3 will discuss applications of DCKR to semantic processing of natural languages.

2. Knowledge Representation in DCKR

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:6,_).
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,P,S) :-
        isa(mammal,P,[elephant|S]).
06) sem(mammal,bloodTemp:warm,_).
07) sem(mammal,P,S) :-
        isa(animal,P,[mammal|S]).
08) sem(animal,P,S) :-
        isa(creature,P,[animal|S]);
        hasa(face,P,[animal|S]);
        ........
        hasa(body,P,[animal|S]).
09) sem(animal,age:X,_) :-
        bottomof(S,B),
        sem(B,birthYear:Y,_),
        X is 1986 - Y.
10) sem(face,P,S) :-
        hasa(eye,P,[face|S]);
        hasa(nose,P,[face|S]);
        ........
        hasa(mouth,P,[face|S]).
```

Now the meanings of the sem, isa and hasa predicates, which are important to descriptions in DCKR, are explained later 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 05), represent an individual name and a prototype name, respectively. A set of Horn clauses headed by the sem predicate with the same individual name or prototype name represents an individual object or a prototype object, respectively.

The second argument in the sem predicate is a pair composed of a slot name and a slot value. The pair is hereinafter called a 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 same property P. 05) and 07) describe the fact that an elephant is a mammal and that a mammal is an animal. 08) describes the fact that an animal is a creature and has a face, 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 and hasa are defined below.

```
11) isa(Upper,P,S) :-
        P = isa:Upper;
        sem(Upper,P,S).
12) hasa(Part,X:Y,S) :-
        X == hasa,
        (Y = Part;
         sem(Part,hasa:Y,S)).
```

The isa predicate and the hasa predicates are used for the inheritance of knowledge through subordinate-superordinate and part-whole relations, respectively.

DCKR is provided with the bottomof predicate, which is used in the body of 09). 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, 09) identifies the individual (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, 09) is reached by the isa predicate in 04), 05) and 07). As a result,

```
X=6
```

is derived by the Prolog interpreter.

Also, if

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

is executed, all properties 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:6
```

Note that all knowledge (SV pairs; properties) 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, it is possible to access an individual or prototype at the lower level from a mammal at the higher level:

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

Finally, if

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

is executed, you may have the following results:

```
X = face;
X = eye;
X = nose;
 ......
X = mouth;
 ......
X = body
```

From the foregoing explanation, you will understand that if only knowledge is described in DCKR, inference is automatically performed by the interpreter built in Prolog.

3. Semantic Processing of Natural Language

3.1 Dsecriptions of Lexical Items in DCKR

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.

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 represents 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 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].

```
13) 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).
14) sem(open,obj:Filler~In~Out,_) :-
          (sem(Filler,isa:eventOpen,_);
               sem(Filler,isa:thingOpen,_)),
          extractsem(object:Filler~In~Out).
15) sem(open,with:Filler~In~Out,_) :-
          sem(Filler,isa:instrument,_),
          extractsem(instrument:Filler~In~Out).
16) sem(open,P,S) :-
          isa(action,P,[open|S]);
          isa(event,P,[open|S]).
```

13),14) and 15) 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 pair). For example, the body of 13) 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 (a semantic constraint), the action extractsem(agent:Filler~In~Out) starts making the deep case of the Filler the agent case that is 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 meaning of the second, third and forth SA pair in 13) is obvious now.

Form the foregoing explanation, the meaning of the slots in 14) and 15) will be evident. In addition to "with", there are many slots corresponding to prepositional phrases, but they are omitted to simplify the explanation.

16) 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, and provides an example of multiple inheritance of knowledge as well.

The descriptions of 13) through 16) 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.

3.2 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 13) - 16)) 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.

3.3 Test result

Some comments will be made on the results of semantic 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:

  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

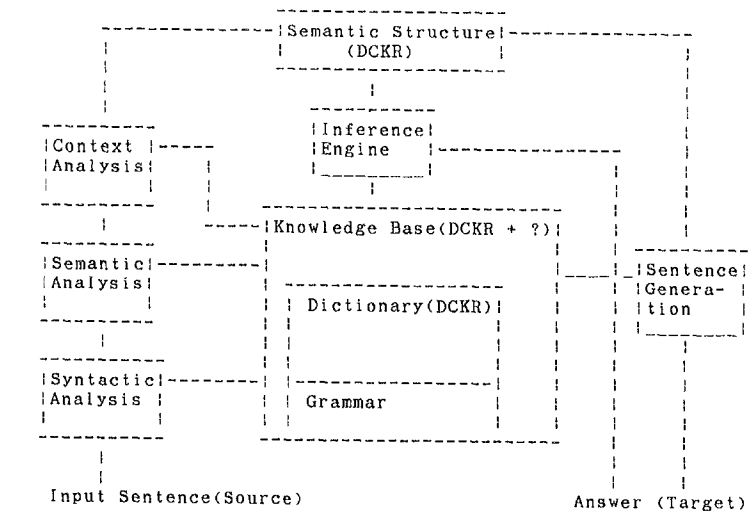

Fig. 1 DCKR and Natural-Language-Understanding System

X=key#7

by merely executing that.

4. Conclusion

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.

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.

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.

[Acknowledgment]

Authors wish to express their great gratitude to Mr.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. Mrs. Sachie Saito helped us for preparing this manuscript.

5. 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, ICOT 95-106(1986), 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, ICOT 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, ICOT, 317-328(1985), in Japanese.

[Tanaka 86] Tanaka,H.: Definite Clause Knowledge Representation and its Applications, ICOT-TR(in press).