It is the last one

Please copy now!!!

A Method of Semantic Processing in Prolog

Hozumi TANAKA

(Tokyo Institute of Technology)

After the proposal of metamorphosis grammar by A. Colmeraure, F.C.N. Pereira and D.H.D Warren have developed an excellent parsing method embedded in Prolog. They have not only developed a grammar description formalism called DCG (Definite Clause Grammar), but also given us a flexible way of performing syntactic processing of natural languages. According to their method, it is not necessary for us to use a special program called a parser, which is replaced by Prolog interpreter.

In this paper, we will propose a semantics representation formalism called DCD (Definite Clause Dictionary) which not only makes easy for us to describe semantics of each lexical entry, but also alleviates our programming efforts of semantic processing. Essential parts of semantic processing is directly carried out through Prolog interpreter.

A usual lexical entry has been expressend in the form of a frame or a structured object that is composed of many slots. In DCD, each slot in a frame is expressed in the form of a definite clause, and a frame, which corresponds to the description of a lexical entry, is composed of a set of definite clauses. An example of DCD is shown in Fig. 1 where a frame and a slot name appear in the head of each clause. With regard to the first clause in the Fig.1, a frame name and a slot name are "open" and "subj", respectively. Semantic constraints for the slot are described in the body of each clause, where a variable N is considered as a filler of the slot.

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

sem(open,subj:N~1~0) :- nonvar(N),
       (sem(N,human),
        addProp(agent:N~1~0)
        ;
        (sem(N,thingOpened);
         sem(N,meeting)),
        addProp(object:N~1~0)
        ;
        sem(N,instrument),
        addProp(instrument:N~1~0)
        ;
        sem(N,wind),
        addProp(reason:N~1~0)),
        !.
sem(open,obj:N~1~0) :- nonvar(N),
       (sem(N,thingOpened);
        sem(N,meeting)),
       addProp(object:N~1~0).
sem(open,with:N~1~0) :- nonvar(N),
       sem(N,instrument),
       addProp(instrument:N~1~0).
sem(open,Prop) :-
       sem(act,Prop).
                .
                .
sem(X,X).
```

Fig.1 An example of DCD descriptions for "open."

From the DCD description shown in the Fig.1, it is easy for us to understand the following facts:

1) The selection of a slot is carried out automatically through the backtracking mechanism of Prolog interpreter.
2) Checking of semantic constraints is replaced by the execution of the body of each Prolog clause.
3) DCD descriptions are completely compilable.

DCD provides a natural and a simple way of expressing the inheritance of knowledge. The following clause in the Fig.1 indicates an example:

```
sem(open,Prop) :- sem(act,Prop).
```

The above clause means that if "act", which is an upper concept of "open", has the property of Prop, "open" gets the same Prop. Through the unification mechanism of Prolog interpreter, all slots in "act" will be able to access from "open."

In order to understand the mechanism of our inheritance of knowledge, let us take another example from [Nillson 80]:

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

sem(clyde,Prop) :- sem(elephant,Prop).
sem(elephant,color:gray).
sem(elepahnt,Prop) ;- sem(mammal,Prop).
sem(mammal,bloodTemp:warm).
                .
                .
                .
sem(X,X).
```

```
?- sem(clyde,bloodTemp:X).

X = warm
yes

?-sem(clyde,Prop).

Prop = color:gray;
Prop = bloodTemp:warm;
Prop = mammal;
Prop = elephant;
Prop = clyde;
no

?- sem(X,bloodTemp:warm).

X = clyde;
X = elephant;
X = mammal;
no
```

From tracing some examples of goal executions give above, it is not difficult for us to understand how to make the inheritance of knowledge.

An example of semantic processing is shown in the Fig.2 (a) in which DCD descriptions of the Fig.1 are used. As shown in the Fig.2 (b) DCG description, only a "sem1" predicate is called for the semantic interpretation. Readers should note that as the results of semantic processing are also in the forms of DCD, queries for the results is easily answerable through the use of inference mechanism in Prolog interpreter.

```
/*semantic interpretation programs */
seml([Uname,Sem | OldSem],
      Sname:[FnamelRest],NewSem) :-
                  sem(Uname,Sname:Fname
                          -[Uname,Sem | [[FnamelRest] | OldSem]]
                          -NewSem).
-------------------------------------------------

        ..........

sdec([not_adv | VP_A],[SDEC_S | ADV1_S]) -->
   subj(SUBJ_A,SUBJ_S),
  ([],
    {ADV1_S=[] };
   adv(ADV1_A,ADV1_S) ),
   vp(VP_A,VP_S).
      { subj_v(SUBJ_A,VP_A),
        seml(VP_S,subj:SUBJ_S,SDEC_S) }.

sentence(SDEC_A,SDEC_S) -->
   sdec(SDEC_A,SDEC_S).

        ..........

-------------------------------------------------
```

Fig 2. (a)

```
Input sentences
1: 1: i open the door with a key.


open#5::
  prototype:open          % sem(open#5,Prop) :- sem(open,Prop).
  agent:i#4               % sem(open#5,agent:i#4).
  instrument:key#7        % sem(open#5,instrument:key#7).
  object:door#6           % sem(open#5,object:door#6).
i#4::
  prototype:i             % sem(i#4,Prop) :- sem(i,Prop).

door#6::
  prototype:door          % sem(door#6,Prop) :- sem(door,Prop).
  det:the                 % sem(door#6,det:the).

key#7::
  prototype:key           % sem(key#7,Prop) :- sem(key,Prop).
  det:a                   % sem(key#7,det:a).
```

```
open#5::
  prototype:open          % sem(open#5,Prop) :- sem(open,Prop).
  agent:i#4               % sem(open#5,agent:i#4).
  object:door#6           % sem(open#5,object:door#6).

i#4::
  prototype:i             % sem(i#4,Prop) :- sem(i,Prop).

door#6::
  prototype:door          % sem(door#6,Prop) :- sem(door,Prop).
  det:the                 % sem(door#6,det:the).
  with:key#7              % sem(door#6,with:key#7).

key#7::
  prototype:key           % sem(key#7,Prop) :- sem(key,Prop).
  det:a                   % sem(key#7,det:a).
```

Fig 2. (b)