A PROLOG IMPLEMENTATION OF SUHG

HARADA Yasunari

0. INTRODUCTION

SUHG is a simplified version of one of HG-based language description formalisms designed to facilitate understanding of technically naive readers.[1] In a way, it could be considered of as a higher level syntax specification language to be employed in unification environments for use by linguists. In order to give a procedural definition of the meta-syntax and semantics of this grammar description formalism, a prolog implementation of a small fragment of English syntax is given, along with a tiny sample lexicon.

1. A SAMPLE PROLOG DESCRIPTION OF ENGLISH SYNTAX

What follows is a source listing of a syntax description of English written in standard DEC-10 prolog.

```
%       SAMPLE DESCRIPTION OF GRAMMAR

member(ELEMENT,[ELEMENT|ANY]).
member(ELEMENT,[ANY|LIST]):-member(ELEMENT,LIST).

append([],L,L).
append([ANY|L1],L2,[ANY|L3]):-append(L1,L2,L3).

of(category(SLASH,SUBCAT,PS,AGR,CASE),slash,SLASH).
of(category(SLASH,SUBCAT,PS,AGR,CASE),subcat,SUBCAT).
of(category(SLASH,SUBCAT,PS,AGR,CASE),ps,PS).
of(category(SLASH,SUBCAT,PS,AGR,CASE),agr,AGR).
of(category(SLASH,SUBCAT,PS,AGR,CASE),case,CASE).

compose(M,D1,D2):-label(LABEL),psr(M,D1,D2,LABEL).

label(left_complementation).
```

```
label(right_complementation).
label(left_adjunction).
label(right_adjunction).

category_sequence([]).
category_sequence([C]):-
   well_formed_category(C).
category_sequence([C|CS]):-
   well_formed_category(C),category_sequence(CS).

well_formed_category(CATEGORY):-
   of(CATEGORY,slash,SLASH),category_sequence(SLASH),
   of(CATEGORY,subcat,SUBCAT),category_sequence(SUBCAT),
   of(CATEGORY,ps,PS),member(PS,[v,n,a,p,det]),
   of(CATEGORY,agr,AGR),member(AGR,[nil,sng,plu]),
   of(CATEGORY,case,CASE),
      member(CASE,[nil,acc,nom,loc,to,for,on,in,at,of]).

psr(MOTHER,LEFT,RIGHT,LABEL) :-
   ffp(MOTHER,LEFT,RIGHT),
   sfp(MOTHER,LEFT,RIGHT,LABEL),
   hfp(MOTHER,LEFT,RIGHT,LABEL).

ffp(MOTHER,LEFT,RIGHT):-
   of(MOTHER,slash,MSLASH),
   of(LEFT,slash,LSLASH),
   of(RIGHT,slash,RSLASH),
   append(LSLASH,RSLASH,MSLASH).

sfp(MOTHER,COMPLEMENT,HEAD,left_complementation) :-
   left(HEAD),
   of(MOTHER,subcat,MSUBCAT),
   of(HEAD,subcat,HSUBCAT),
   [COMPLEMENT|MSUBCAT]=HSUBCAT.

sfp(MOTHER,HEAD,COMPLEMENT,right_complementation) :-
   right(HEAD),
   of(MOTHER,subcat,MSUBCAT),
   of(HEAD,subcat,HSUBCAT),
   [COMPLEMENT|MSUBCAT]=HSUBCAT.

sfp(MOTHER,ADJUNCT,HEAD,left_adjunction) :-
   left_modify(HEAD,ADJUNCT),
   of(MOTHER,subcat,MSUBCAT),
   of(HEAD,subcat,HSUBCAT),
   MSUBCAT=HSUBCAT.

sfp(MOTHER,HEAD,ADJUNCT,right_adjunction) :-
   right_modify(HEAD,ADJUNCT),
   of(MOTHER,subcat,MSUBCAT),
```

```
   of(HEAD,subcat,HSUBCAT),
   MSUBCAT=HSUBCAT.

hfp(MOTHER,COMPLEMENT,HEAD,left_complementation) :-
   of(MOTHER,ps,MPS),of(HEAD,ps,HPS),MPS=HPS,
   of(MOTHER,agr,MAGR),of(HEAD,agr,HAGR),MAGR=HAGR,
   of(MOTHER,case,MCASE),of(HEAD,case,HCASE),MCASE=HCASE.

hfp(MOTHER,HEAD,COMPLEMENT,right_complementation) :-
   of(MOTHER,ps,MPS),of(HEAD,ps,HPS),MPS=HPS,
   of(MOTHER,agr,MAGR),of(HEAD,agr,HAGR),MAGR=HAGR,
   of(MOTHER,case,MCASE),of(HEAD,case,HCASE),MCASE=HCASE.

hfp(MOTHER,ADJUNCT,HEAD,left_adjunction) :-
   of(MOTHER,ps,MPS),of(HEAD,ps,HPS),MPS=HPS,
   of(MOTHER,agr,MAGR),of(HEAD,agr,HAGR),MAGR=HAGR,
   of(MOTHER,case,MCASE),of(HEAD,case,HCASE),MCASE=HCASE.

hfp(MOTHER,HEAD,ADJUNCT,right_adjunction) :-
   of(MOTHER,ps,MPS),of(HEAD,ps,HPS),MPS=HPS,
   of(MOTHER,agr,MAGR),of(HEAD,agr,HAGR),MAGR=HAGR,
   of(MOTHER,case,MCASE),of(HEAD,case,HCASE),MCASE=HCASE.

left(category([],[category([],[],n,AGR,nom)],v,nil,nil)).
left(category([],[category([],[],det,AGR,nil)],n,AGR,CASE)).

right(C):-well_formed_category(C),¥+left(C).

left_modify(category([],[category([],[],det,AGR,nil)],n,AGR,CASE),
   category([],[],a,nil,nil)).

right_modify(category([],[category([],[],n,AGR,CASE)],v,nil,nil),
   category([],[],p,nil,loc)).

lexical(SPELL,category([],[],a,nil,nil)):-
   lexical(SPELL,category([],[category([],[],p,nil,of)],a,nil,nil)).
```

This syntax description, simple and abstract as it is, works fine when embedded into a bottom-up parsing algorithm. A slight modification of an earlier and more restricted version of this syntax description worked also with a top-down parsing algorithm.

As is usual, our system is undergoing a drastically rapid series of revisions, so strictly speaking, there are some minor

and unimportant discrepancies between the grammar description formalism adopted here and the one employed in Harada 1986. However, a close comparison of the two will facilitate the understanding of both.

2. A SAMPLE LEXICON OF ENGLISH

What follows is a very tiny and restricted set of lexical entries written in DEC-10 prolog to be used along with the syntax description listed above. Here, lexical redundancies are not treated in any way; they should be conceived of as "virtual lexical entries" in the terminology in Harada 1986.[2]

```
%       SAMPLE LEXICON FOR HG VERSION 2

lexical(a,category([],[],det,sng,nil)).
lexical(the,category([],[],det,AGR,nil)).
lexical(this,category([],[],det,sng,nil)).
lexical(that,category([],[],det,sng,nil)).
lexical(these,category([],[],det,plu,nil)).
lexical(those,category([],[],det,plu,nil)).
lexical(you,category([],[],n,plu,CASE)).
lexical(it,category([],[],n,sng,CASE)).
lexical(he,category([],[],n,sng,nom)).
lexical(she,category([],[],n,sng,nom)).
lexical(they,category([],[],n,plu,nom)).
lexical(him,category([],[],n,sng,acc)).
lexical(her,category([],[],n,sng,acc)).
lexical(them,category([],[],n,plu,acc)).
lexical(pretty,category([],[],a,nil,nil)).
lexical(to,category([],[category([],[],n,AGR1,acc)],p,nil,to)).
lexical(for,category([],[category([],[],n,AGR1,acc)],p,nil,for)).
lexical(on,category([],[category([],[],n,AGR1,acc)],p,nil,on)).
lexical(in,category([],[category([],[],n,AGR1,acc)],p,nil,in)).
lexical(at,category([],[category([],[],n,AGR1,acc)],p,nil,at)).
lexical(on,category([],[category([],[],n,AGR1,acc)],p,nil,loc)).
lexical(in,category([],[category([],[],n,AGR1,acc)],p,nil,loc)).
lexical(at,category([],[category([],[],n,AGR1,acc)],p,nil,loc)).
lexical(of,category([],[category([],[],n,AGR1,acc)],p,nil,of)).
lexical(run,category([],[category([],[],n,plu,nom)],v,nil,nil)).
lexical(runs,category([],[category([],[],n,sng,nom)],v,nil,nil)).
```

```
lexical(ran,category([],[category([],[],n,AGR1,nom)],v,nil,nil)).
lexical(pen,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(pens,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(book,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(books,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(student,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(students,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(teacher,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(teachers,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(museum,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(museums,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(park,category([],[category([],[],det,sng,nil)],n,sng,CASE)).
lexical(parks,category([],[category([],[],det,plu,nil)],n,plu,CASE)).
lexical(proud,category([],[category([],[],p,nil,of)],a,nil,nil)).
lexical(afraid,category([],[category([],[],p,nil,of)],a,nil,nil)).
lexical(tired,category([],[category([],[],p,nil,of)],a,nil,nil)).
lexical(are,category([],[category([],[],a,nil,nil),
   category([],[],n,plu,nom)],v,nil,nil)).
lexical(is,category([],[category([],[],a,nil,nil),
   category([],[],n,sng,nom)],v,nil,nil)).
lexical(were,category([],[category([],[],a,nil,nil),
   category([],[],n,plu,nom)],v,nil,nil)).
lexical(was,category([],[category([],[],a,nil,nil),
   category([],[],n,sng,nom)],v,nil,nil)).
lexical(love,category([],[category([],[],n,AGR1,acc),
   category([],[],n,plu,nom)],v,nil,nil)).
lexical(loves,category([],[category([],[],n,AGR1,acc),
   category([],[],n,sng,nom)],v,nil,nil)).
lexical(loved,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,nom)],v,nil,nil)).
lexical(go,category([],[category([],[],p,nil,to),
   category([],[],n,plu,nom)],v,nil,nil)).
lexical(goes,category([],[category([],[],p,nil,to),
   category([],[],n,sng,nom)],v,nil,nil)).
lexical(went,category([],[category([],[],p,nil,to),
   category([],[],n,AGR2,nom)],v,nil,nil)).
lexical(picture,category([],[category([],[],p,nil,of),
   category([],[],det,sng,nil)],n,sng,CASE)).
lexical(donate,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,plu,nom)],v,nil,nil)).
lexical(donates,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,sng,nom)],v,nil,nil)).
lexical(donated,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,AGR3,nom)],v,nil,nil)).
lexical(give,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,plu,nom)],v,nil,nil)).
lexical(gives,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,sng,nom)],v,nil,nil)).
lexical(gave,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,to),category([],[],n,AGR3,nom)],v,nil,nil)).
```

```
lexical(give,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,plu,nom)],v,nil,nil)).
lexical(gives,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,sng,nom)],v,nil,nil)).
lexical(gave,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,AGR3,nom)],v,nil,nil)).
lexical(buy,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,for),category([],[],n,plu,nom)],v,nil,nil)).
lexical(buys,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,for),category([],[],n,sng,nom)],v,nil,nil)).
lexical(bought,category([],[category([],[],n,AGR1,acc),
   category([],[],p,nil,for),category([],[],n,AGR3,nom)],v,nil,nil)).
lexical(buy,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,plu,nom)],v,nil,nil)).
lexical(buys,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,sng,nom)],v,nil,nil)).
lexical(bought,category([],[category([],[],n,AGR1,acc),
   category([],[],n,AGR2,acc),category([],[],n,AGR3,nom)],v,nil,nil)).
```

3. HOW TO UNDERSTAND THESE PROLOG SOURCE LISTINGS

Although it is not my intention to give any account of prolog here, I will give you a brief explanation of how to understand this description in a quick and informal manner.[3]

First, a prolog statement of the form "predicate_name(argument_1, ... ,argument_n). " means that the relation designated by the predicate name "predicate_name" hold of things designated by "argument_1", ... , "argument_n". Thus, "lexical(this,category([],[],det,sng,nil))." asserts that a relation "lexical" hold of "this" and "category([],[],det,sng,nil))". Informally put, this is intended to mean that "this" is a lexical item with its category specified as a singular determiner. Similarly for all lexical items in the SAMPLE LEXICON. Here, "category" is a dummy functor introduced to work just this way.

Second, a prolog statement of the form "term_0:-term_1, ..., term_n." means that term_1 through term_n must be satisfied in order for term_0 to be successfully satisfied. For instance,

```
psr(MOTHER,LEFT,RIGHT,LABEL) :-
   ffp(MOTHER,LEFT,RIGHT),
   sfp(MOTHER,LEFT,RIGHT,LABEL),
   hfp(MOTHER,LEFT,RIGHT,LABEL).
```

means that MOTHER, LEFT, RIGHT, and LABEL stand in psr relation only when MOTHER, LEFT and RIGHT stand in ffp relation and MOTHER, LEFT, RIGHT and LABEL stand in sfp relation and MOTHER, LEFT, RIGHT and LABEL stand in hfp relation; or, intuitively speaking, in order for three categories to compose a local tree Foot Feature Principle, Subcat Feature Principle and Head Feature Principle must be satisfied among the three categories involved.

In PROLOG names beginning with an upper-case letter designates a variable and constants must begin with a lower-case letter; thus things like "slash" means specific feature name while things like "SLASH" is a variable intended to range over possible SLASH values.

Thus "category" is a dummy functor intended to designate categorial specifications of syntactic categories in the grammar. Each argument place of this predicate is intended to contain information regarding SLASH values, SUBCAT values, PS (or part of speech) values, AGR (or agreement) values, and CASE values.

Introduction of a tertiary relation "of" holding of a syntactic category, a feature name and the value of the syntactic

category in question with respect to the feature named somewhat lengthens our syntax description, but it greatly increases its readability.

There are minor technical problems regarding the definitions of "category_sequence" and "well_formed_category", the details of which I would not go into here.

Definitions of "ffp" (mnemonic for Foot Feature Principle), "sfp" (mnemonic for Subcat Feature Principle), and "hfp" (mnemonic for Head Feature Principle) are separately stated four times, each for a particular "label" (or rule type). Linguists preoccupied with the desire to capture generalizations and opposed to any loss of generalizations might argue that such redundant descriptions are not any part of a successful description of natural language syntax. In fact, this seeming redundancy is not any part of the grammar. There are more concise but less readable ways of expressing the same definitions using control structures. I settled for a redundant but readable statement for ease of exposition.

4. SOME SIMPLE DEMONSTRATIONS

As evidence that I am not talking through my hat, I give a few exemplary results of running this humble syntax description. Shirai Hidetoshi kindly fixed the bugs in my version of bottom-up parser, which is a notational variant of prolog bottom-up parser by Miyoshi, Hirakawa, Yasukawa, Mukai and Hurukawa 1983. Shirai

Hidetoshi also provided me with a prolog program to draw diagrams of resulting parse trees.

First of all, let's see if our system can handle a very simple sentence like "He runs." To facilitate our experiment, I had to restrict my attention to the very basics of parsing, thus no preprocessing for lexical analysis is incorporated here, which might be best suited for a freshman exercise of computer majors. Thus, a sentence has to be fed to our system as a list of lexical items, with "," ever present as a delimiter instead of " ". When asked to parse "[he,runs]", our system will answer as follows. The lines beginning with "| ?-" shows our requests to the system, with the system's answers following them.

```
| ?-parse([he,runs]).
category([],[],v,nil,nil)

          v
          !
     n---------v
     !         !
    he       runs

yes
```

In the tree diagram all information except the value of PS of the category in question is suppressed to keep matters manageable. In the internal representation, however, all feature values are stored for all syntactic categories involved. Thus if you could supply suitable utility programs, everything could be displayed on the screen at your request. The last line of the

answer, namely "yes", means that the parsing procedure ended successfully, which is not always the case. For instance, "He goes." cannot be successfully parsed in our system as our lexicon stipulates that "go" require a prepositional phrase with "to" as its head.

```
| ?-parse([he,goes]).
no

| ?-parse([he,goes,to,the,park]).
category([],[],v,nil,nil)
           v
           !
 n----------v
 !          !
he   v---------p
     !         !
    goes  p------n
          !      !
          to  det---n
              !    !
             the park

yes
```

A prepositional phrase with "in" as its head can optionally modify any verb phrase, or penultimate projection of verbs.

```
| ?-parse([he,runs,in,the,park]).
category([],[],v,nil,nil)

           v
           !
 n----------v
 !          !
he   v---------p
     !         !
    runs  p------n
          !      !
          in  det---n
              !    !
```

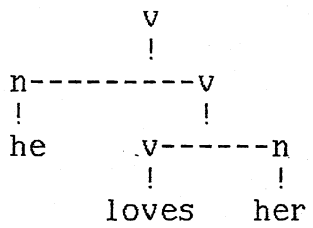
```
                the park

yes
```

Parsing may end unsuccessfully because of some unification failure, or disagreement of particular feature values. For example, "teacher" a singular noun is specified to require singular determiners, whereas "those" is specified as a plural determiner. Thus, "those teacher" is not a well-formed syntactic category of English, while "those teachers" is a plural saturated projection of a noun.

```
| ?-parse([those,teacher]).
no

| ?-parse([those,teachers]).
category([],[],n,plu,CASE_11)

          n
          !
    det---------n
     !          !
  those    teachers

yes
```

Similarly, "He gave her." is not a grammatical sentence of English in our fragment because "give" is specified to be a ditransitive verb, while "He loves her." is quite all right.

```
| ?-parse([he,gave,her]).
no

| ?-parse([he,loves,her]).
category([],[],v,nil,nil)
```



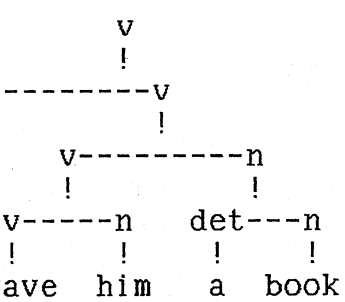
yes

Case disagreement also rules out "He loves she." because object of verbs must be accusative whereas she is specified as nominative.

```
| ?-parse([he,loves,she]).
no
```

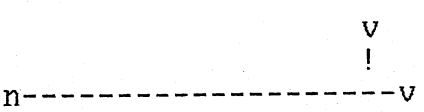
Our syntax sanctions both "She gave him a book." and "She gave him a book in the museum." as the following diagrams show.

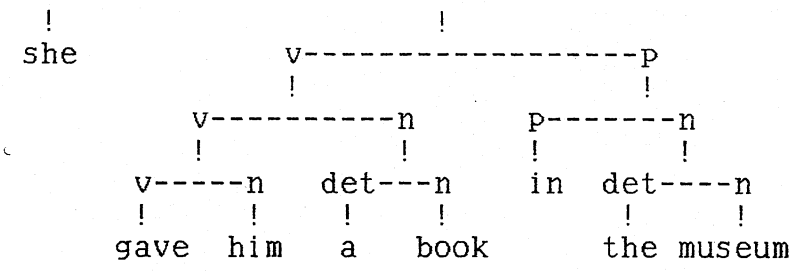
```
| ?-parse([she,gave,him,a,book]).
category([],[],v,nil,nil)
```



yes

```
| ?-parse([she,gave,him,a,book,in,the,museum]).
category([],[],v,nil,nil)
```

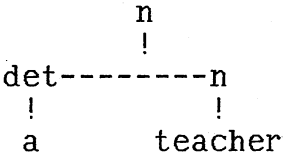




yes

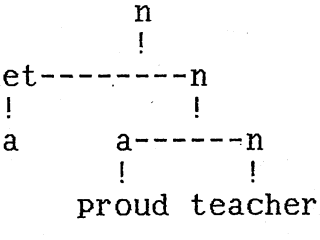
A prepositional phrase modifies a verb phrase from the right, but an adjective modifies a noun from the left, as the following two parse trees show.

```
| ?-parse([a,teacher]).
category([],[],n,sng,CASE_11)
```



yes

```
| ?-parse([a,proud,teacher]).
category([],[],n,sng,CASE_11)
```



yes

In this test run, no long distance agreement was dealt with, mainly for memory limitations of the hardware I had access to at

the time of this writing.

NOTES

1. A brief but somewhat more detailed account of SUHG is given in Harada 1986.

2. The last line in our sample syntax states that prepositional complements to adjectives are all optional. This could be considered of as a prolog counterpart of Complement Omission Metarule of Gazdar, Klein, Pullum and Sag (1985: 124). Metarules in GPSG in general, however, are replaced in HG with lexical rules. Thus, this statement might better be considered of as a kind of lexical rule. All these remarks should be construed as suggestive of further directions of research regarding the grammar description formalisms and no part of this is to be construed as a serious proposal concerning the description of English syntax per se.

3. For a more detailed introduction to the programming language prolog see Clocksin and Mellish 1981.

REFERENCES

Clocksin, W. F. and C. S. Mellish 1981. Programming in Prolog. Berlin: Springer-Verlag.

Gazdar, G., E. Klein, G. Pullum and I. Sag 1985. Generalized Phrase Structure Grammar. Oxford: Basil Blackwell.

Harada, Yasunari 1986. Complementation and Adjunction in HG, MS.

Miyoshi, H., H. Hirakawa, H. Yasukawa, K. Mukai and K. Hurukawa

1983. BUP System. ICOT Technical Report, TR-038.