Lund University, Dept. of Linguistics
Working Papers 38 (1991), 169-87

Prolog Implementations of English and Swedish GB Grammars

Bengt Sigurd and Mats Eeg-Olofsson

Introduction and abstract

Government and Binding theory (Chomsky 1981, Sells 1985) plays a dominant role in current linguistics and is an almost compulsory part of the linguistics curriculum at universities. The advantage of GB is its rigorous theory, allowing only certain simple trees and transformations, supplemented by certain simple principles and constraints. The GB approach makes it possible to characterize language in a simple way and to pinpoint the differences between languages as different settings of the parameters of the base structure, transformations and constraints.

In spite of its dominance in linguistics, GB has a comparatively low status in computational linguistics, as is witnessed by the proceedings of COLING and ACL (for exceptions see References). Computational linguists instead favour competing theories such as Generalized Phrase Structure Grammar, Lexical-Functional Grammar, Tree Adjoining Grammar, or eclectic variants. There is, however, a demand for computer implementations of GB for linguistic and pedagogical purposes.

This paper presents an experimental Prolog (LPA MacProlog) implementation of the basic features of GB, including categorial base rules for deep structures (d-structures) and transformations for movements of tense, *wh*-words, noun phrases, verbs and adverbs. The movements leave traces in the surface structure (s-structure) in accordance with current theory. Both the leaves (words) of the d-structure tree and the leaves of the s-structure tree can be projected as sentences, the s-structures with or without traces. Sentences can be generated from the d-structure through the transformations or parsed by finding the d-structure after running the transformations in reverse. The English and Swedish grammars differ, as the English auxiliaries are generated in the tense slot *(infl)* and *not* is a barrier in English. Furthermore, Swedish moves all finite verbs to the

second *(comp)* position, which is done only in questions in English, e.g. *Whom did Bill like?* The paper also shows how the grammars can be used for machine translation, handling differences in the d-structure by transfer rules.

Overview

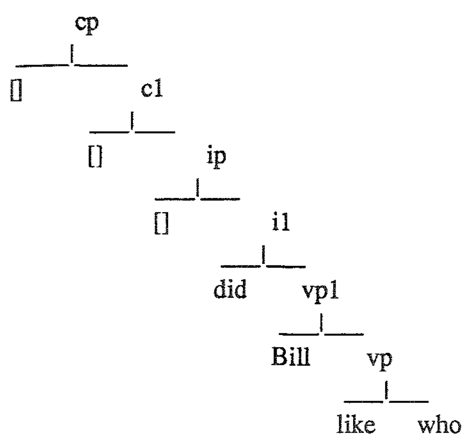
Current theory agrees on using so-called X-bar syntax, which entails that the characteristic feature (head) of a category should be reflected in all larger phrases constructed (projected) from it, and that the hierarchy of the phrases should be indicated by a (bar) number. Thus there are verb phrases of different sizes, e.g. a simple verb as in *v(like)*, this verb phrase adjoined to an object, as in *v1(like Eve)*, or this phrase adjoined to a preceding subject, as in *v2(Bill like Eve)*, and according to this theory the successively larger phrases should be marked as *v1*, *v2*, *v3*, etc. (using numbers instead of superimposed bars). Adverbs may be adjoined rather freely, allowing still larger verb phrases such as *(not Bill like Eve)* or *(Bill like Eve much)*.

The X-bar hierarchy is at least partly reflected in our experimental d-rules. The node below the top node *cp* (the *comp phrase*) is called *c1*, as is usual, and the node below *ip* (the *inflectional phrase*) is called *i1*. This usage is in accordance with the presentation of Swedish in Falk 1991. The verb phrase including the object is termed *vp*, the verb phrase including also the deep subject is termed *vp1*, the verb phrase including a sentence adverbial is, however, termed *vps*, and the verb phrase with an adjoined adverbial of any other kind is called *vpa*. These labels could of course be changed if required. Figure 1 shows a sample GB d-structure tree. The representation states that *Bill* is in the deep subject *(specifier)* position of the verb phrase labelled *vp1*, and that *who* is in the *complement* position after the verb *like*. The tense slot *(i1)* is occupied by *did* (the *d* tense affix alone would not be accepted by the later transformations). We will call the specifier position of *cp* the *fundament*, following the Danish linguist Diderichsen.

After having passed through the transformations, the representation would surface as the following s-structure tree:

```
cp(whom,c1(did,ip(bill,i1([],vp1(n1,vp(like,n2)))))).
```

This s-tree shows that *who* has been changed into the object form *whom* and moved to the initial position, *did* has been moved to the *c1* position in order to precede the subject, which is moved to the normal surface subject



```
cp([],c1([],ip([],i1(did,vp1(bill,vp(like,who))))))
```

Figure 1. d-representation (tree and parenthesis) of *Whom did Bill like?*

position *ip*. The grammatical modules and their results are shown in the following (comments within /* */ as in Prolog).

D-structure rules (sample) in Definite Clause Grammar

```
d(cp(Cp,C1))  -->  fund(Cp),c1(C1).  /*  fund  for  cp
(Diderichsen) */

fund([]) --> []. /* fund may be empty */

c1(c1(C,Ip)) --> c(C),ip(Ip).

c([]) --> []. /* c(omp) may be empty, if main clause */
c(Subj) --> subj(Subj). /* c is a subjunction in sub
clauses */

ip(ip(I,I1)) --> i(I),i1(I1).

i([]) --> [].

i1(d) --> [d]. /* tense marker d etc. - see below */
```

The rules generate structures such as

```
cp([],c1([],ip([],i1(d,vp1(who,vp(like,Eve)))))))
```

corresponding to the d-structure projection (d-sentence)

```
d,who,like,Eve.
```

Transformations (sample)

Subject *wh*-movement to the front ('fund') position, leaving the trace *n1*:

```
cp([],c1([],ip([],i1(T,vp1(who,X))))) ==>
cp(who,c1([],ip([],i1(T,vp1(n1,X))))).
```

This transformation results in structures like

```
cp(who,c1([],ip([],i1([],vps(n1,vp(liked,Eve))))))
```

whose projection (without traces) is

```
who,liked,Eve.
```

English d-structure rules (categorial base rules)

The d-structure rules used for English are the following, where the prefix *e* (as in *efund, ec1, ecomp* etc.) marks English categories to prevent them from being confused with the corresponding Swedish categories, which are labelled *fund, c1, comp,* etc. The DCG formalism used when constructing the d-structure yields a direct projection of the d-structure. The d-structure is established as an argument to the left of the arrow, and the projection of the leaves of this d-structure is seen to the right of the arrow. The projection of the leaves of the d-structure may also be called the d-structure sentence or 'base string'. It is an artificial grammatical product, characterized by the d-structure order, where the aux and tense morphemes come first, the sentence adverbial next, and finally the SVO complex, as illustrated by: [d,Bill,like,Eva], corresponding to the surface: [Bill,liked,Eva]; or [d,perhaps,bill,move], corresponding, e.g., to normal surface structure [perhaps,Bill,moved].

English d-rules

```
ed(cp(Fund,c1(Comp,Ip))) -->
     efund(Fund),ec1(c1(Comp,Ip)),
     {(Fund=[],Comp=[];Fund=[],Comp=that;
     Fund=q,Comp\=that)}.
/* A complementizer phrase (cp) may consist of a
Fundament (=Spec of cp) followed by a Comp node
```

```
(=c1); the combination of Fund and Comp must be
restricted as shown between {}. Both may be empty as
in main declarative clauses, the Comp may be a
subjunction as in a subordinated clause, or the
Fundament may be q (marking a question), but in that
case there cannot be any subjunction */

efund([]) --> []. /* the fundament may be empty
(declarative) */
efund(q) --> [?]. /* a question mark denotes a yes-no
question */

ec1(c1(Comp,Ip)) --> ecomp(Comp),eip(Ip).

ecomp([]) --> []. /* main clause */
ecomp(that) --> [that]. /* subordinate clause */

eip(ip(I,I1)) --> ei(I),ei1(I1). /* infl phrase */

ei([]) --> [].

ei1(i1(I,Vp1)) --> ein(I),evp1(Vp1). /* without
sentence adverb */
ei1(i1(I,Vps)) --> ein(I),evps(Vps). /* with sentence
adverb vp */

ein(d) --> [d]. /* general tense marker (avoiding
irregularities) */
ein([A,d]) --> eaux(A),[d]. /* tensed auxiliary, did
included */

evp1(vp1(N,Vp)) --> enp(N),evp(Vp). /* subject vp */
evp1(vp1(N,Vpa)) --> enp(N),evpa(Vpa). /* with adverb
vp */

evps(vps(S,Vp1)) --> esadv(S),evp1(Vp1).

evpa(vp(Vp,A)) --> evp(Vp),eadv(A). /* vp with final
adverb */

evp(vp(V,[])) --> evi(V). /* intransitive */
evp(vp(V,N)) --> evt(V),enp(N). /* transitive */

/* Lexicon */
evt(like) --> [like].

evi(move) --> [move]. /* allows d as general tense
marker */
```

```
eaux(coul) --> [coul]. /* allows d as general tense
marker */
eaux(di) --> [di].
/* allows d as general tense marker */

enp(bill) --> [bill].
enp(eva) --> [eva].
enp(who) --> [who].

esadv(nt) --> [nt].
/* contracted form chosen, as not is occupied in
Prolog */
esadv(perhaps) --> [perhaps].
eadv(yesterday) --> [yesterday].

everb(like).
everb(move).
everb(coul).
everb(di).

eaux(di).
eaux(coul).

enp(bill).
enp(eve).
```

English transformations

The d-structure rules presented above require a number of transformations in order to produce suitable s-structures. Some of these are presented in detail below with comments.

English tense-moving transformations

```
eiatt(X,Y) :-
    X=cp(A,c1(B,ip(C,i1(d,vps(S,vp1(N,vp(V,N2))))))),
    S\=nt,
    Y=cp(A,c1(B,ip(C,i1([],
        vps(S,vp1(N,vp([V,d],N2))))))).
/* moves and attaches tense (d) to verb stem (V), if
not (nt) does not intervene: Bill perhaps liked Eve
*/
```

All such rules of movement should preferably be summed up in a more general rule, expressing the interactions of various principles such as c-command etc. It is difficult to combine such a modular approach with reversibility, i.e. the demand that it should be possible to use the same rule

for both analysis and synthesis. One might suggest a rule of the following format with six variables, which is a specification of the famous *move(@)* rule:

```
move(Alpha,Intree,Outree,Source,Target,Barrier).
```

This rule states that a category (Alpha) in an input tree (Intree) should result in an output tree (Outree), if situated in Source and moved to Target without passing a certain Barrier (such as *not* or an NP boundary). The implementation of such a rule is not straightforward. We will not present a general rule, but we will examine some of the different cases in detail in the following.

As can be seen, our specific rule above can only move the simple tense marker *d*, not auxiliaries such as *could* or *did* which are generated in the same slot. The existence of *not* in the tree requires *do*-support, as is well-known. On our approach there is, in fact, no reason to talk about *do*-support or *do*-insertion. The only thing to note is that in English past tense may be expressed both by *d* and *did*, the last form being used when the tense marker cannot be moved down to the verb stem; *do*-forms may, of course, also be used for emphasis.

In certain cases a different form of the rule must be written for analysis. In analysis, the variables in the tree X are not instantiated when the rule is to apply. If the tense-moving rule is to apply obligatorily to all d-trees, those with auxiliaries should be able to pass vacuously, and so a rule of the following form is needed, allowing, e.g., *[coul,d]* to pass without having to be moved. The variable Rest denotes whatever follows.

```
eiatt(X,Y) :-
    X=cp(A,c1(B,ip(C,i1([Aux,d],Rest)))),
    Y=cp(A,c1(B,ip(C,i1([Aux,d],Rest)))).
/* Bill could like Eva */
```

Wh-moving transformations

```
efmove(X,Y) :-
    X=cp([],c1([],ip(C,i1(D,vp1(who,Z))))),
    Y=cp(who,c1([],ip(C,i1(D,vp1(n1,Z))))).
/* moves subj who to fund position if Fund and Comp
are empty (not in subordinate clause) */
```

This rule illustrates how a trace *n1* is posited in the place of the *who* moved to the *fund* position. No *do*-support is needed here, unlike the case where the object *who (whom)* is moved *(Whom did Bill like)*. In English there is no reason to move all subjects to the *fund* position, as is done in Swedish. In Swedish the equivalent of *who* and ordinary subjects may be moved by the same transformation. In English subjects are normally only moved to the *ip* position.

The movement of the object *who* is illustrated by the following rule, which requires a *do* form (specified as *[di,d]* in our restricted program). In that case *n2* is left as a trace. The *do*-form is also moved to the *c1* position, and the pronoun is simply given its object form *whom* in the process. One may of course handle the *do*-movement and the change of *who* into *whom* as separate processes. Case assignment is not covered separately in our model program.

```
efmove(X,Y) :-
    X=cp([],c1([],ip([],i1([di,d],
        vp1(N,vp(V,who)))))),
    Y=cp(whom,c1([di,d],ip(N,i1([],
        vp1(n1,vp(V,n2)))))).
/* moves objwho(m) to fund position and did to c1 */
```

Adverbs may also be moved to the *fund* position, and this case is handled by the following rule, where the trace *a1* is placed in the original position of the adverb in the d-structure. The adverb is represented by the variable Adv. Only one adverb is treated.

```
efmove(X,Y) :-
    X=cp([],c1([],ip(C,i1(D,vp1(N,vp(vp(E,Adv)))))),
    Y=cp(Adv,c1([],ip(C,i1(D,vp1(N,vp(vp(E,a1)))))).
/* moves Adv to fund position if Fund empty, leaving
trace a1 */
```

The word order in *yes-no* questions is illustrated by *Did Bill move?* and *Did Bill like Eva?* The *do* auxiliary is required (if there is not any other auxiliary) in all such questions in English. This may be handled by requiring that the simple tense marker should not be accepted in the tense slot in questions, and that the auxiliary (*did, could, ...*) should be moved to the *c1* position, in order to precede the subject in *c1*. Yes-no questions are marked by a *q* in the *fund* position in our simple system, but, alternatively, one might mark this in some variable outside the d-structure tree. (*Wh*-questions need not be marked by a *q* in the *fund* position in our system.)

Alternatively, one may move the auxiliary in *yes-no* questions to the *fund* position, but there are several reasons for not doing so. What is

important is that it precedes the subject (which is moved to the *ip* position by the subsequent subject movement transformation). The movement of the aux in *yes-no* questions does not leave any trace in our implementation.

```
efmove(X,Y) :-
    X=cp(q,c1(A,ip(B,i1([Aux,d],Rest)))),
    Y=cp(q,c1([Aux.,d],ip(B,i1([],Rest)))).
/* aux (do included) move to c1 in question */
```

Subject-moving transformations
The movement of an English subject to the standard position *ip* is illustrated by the following rule:

```
esmove(X,Y) :-
    X=cp(A,c1(B,ip([],i1(C,vp1(N,Vp))))),
    Y=cp(A,c1([],ip(N,i1(C,vp1(n1,Vp))))).
/* moves subj (N) to ip position, leaving trace n1 */
```

The subject may be said to move obligatorily in order to get case, which can be distributed by *ip*.

Wh-filter to delete remaining wh-words
Normally, *wh*-words that have not been fronted are to be filtered out. *Wh*-words may, however, stay in their original position, as illustrated by *Who gave whom a book?* or *When did who do what?* When there are several *wh*-words, only one can be fronted. The others must remain in situ. Some languages tolerate unmoved *wh*-words, and there is thus a *wh*-parameter to set for each language. The following rule shows how structures are accepted if they do not include *wh*-words in the subject and object positions indicated.

```
ewhfilter(X,Y) :-
    X=cp(A,c1(B,ip(C,i1(D,vp1(N,vp(V,N2)))))),
    N\=who,N2\=who,
    Y=cp(A,c1(B,ip(C,i1(D,vp1(N,vp(V,N2)))))).
/* blocks remaining subject (N) or object (N2) who */
```

Swedish d-structure (base) rules

The Swedish base d-structure rules look very much the same, except that auxiliaries are not generated under *i1* but under *vp* in Swedish. The following are the corresponding Swedish rules, with lexical items chosen strategically to enable translation between English and Swedish.

```
d(cp(Fund,c1(Comp,Ip))) -->
    fund(Fund),c1(c1(Comp,Ip)),
    {(Fund=[],Comp=[];Fund=[],Comp=att;
    Fund=q,Comp\=att)}.

fund([]) --> []. /* declarative clauses */
fund(q) --> [?]. /* questions */

c1(c1(Comp,Ip)) --> comp(Comp),ip(Ip).

comp([]) --> []. /* main clause */
comp(att) --> [att]. /* subordinate clause */

ip(ip(I,I1)) --> i(I),i1(I1).

i([]) --> [].

i1(i1(I,Vp1)) --> in(I),vp1(Vp1).
i1(i1(I,Vps)) --> in(I),vps(Vps).

in(de) --> [de]. /* the past tense marker */

vp1(vp1(N,Vp)) --> np(N),vp(Vp).
vp1(vp1(N,Vpa)) --> np(N),vpa(Vpa).

vps(vps(S,Vp1)) --> sadv(S),vp1(Vp1).

vp(vp(V,N)) --> vt(V),np(N). /* transitive */
vp(vp(V,[])) --> vi(V). /*intransitive */
vp(vp(V,Vp)) --> aux(V),vpm(Vp).
/* aux with main verb */

vpa(vp(Vp,A)) --> vp(Vp),adv(A).

vpm(vp(V,N)) --> vt(V),np(N). /* trans main verb */
vpm(vp(V,[])) --> vi(V). /* intrans main verb */

vt(gilla) --> [gilla]. /* gilla=like */
vi(flytta) --> [flytta]. /* flytta=move */

aux(kun) --> [kun]. /* kun=coul */

np(bill) --> [bill].
np(eva) --> [eva].
np(vem) --> [vem]. /* vem=who(m) */

sadv(inte) --> [inte]. /* inte=not */
sadv(kanske) --> [kanske]. /* kanske=perhaps */
```

```
adv(igår) --> [igår]. /* igår=yesterday */
verb(gilla).
verb(flytta).
verb(kun).

aux(kun).

np(bill).
np(eva).
np(vem).
```

The Swedish transformations must differ somewhat from the English ones. The Swedish subject may be focused and placed in the *cp* position, while the Swedish finite verb is always moved to position *c1*. Clearly the tense morpheme *de* must be moved down to the verb before the verb is fronted. We will not illustrate these transformations here, however.

Projection rules

The s-structure trees should have the words in the proper linear order – this is the main objective of the movement transformations. But if traces are left, the s-structure trees will include elements that we do not want to appear in the final surface sentence, e.g. *n1* (trace of subject), *n2* (trace of object), *a1* (trace of adverb). The rules which delete unwanted elements of s-structures are here called projection rules. Several types of such rules may be written. The simplest type of projection rule collects the leaves in a list according to their order in the s-structure and then deletes unwanted members of this list.

The simple projection rule mentioned works in generation, but not in analysis (parsing). The parsing problem of GB is not only the problem of finding the relations between the words and the constituents of the sentence, but also the problem of finding the places where traces are to be found (inserted) and identifying the type of trace. The difficult problem of parsing is generally not mentioned in the linguistic GB literature. Since the GB approach is generative and synthetic at heart, the projection of the s-tree deleting unwanted traces etc. is not a problem.

One may write projection rules as DCG rules as illustrated below. These work both in generation and analysis, but they add another layer of grammar rules to the whole system and take away some of the simplicity and beauty of GB.

```
/* English projection rules for surface sentence
without traces etc. */

eproj(cp([],c1([],ip(N,i1([],vp1(n1,vp(V,N2)))))))
--> enp(N),[V],enp(N2).
/* Eng transitive: bill liked bill */

eproj(cp([],c1([],ip(N,i1([],vp1(n1,vp(V,[])))))))
--> enp(N),[V]. /*intransitive: bill moved */

eproj(cp([],c1([],ip(N,i1([],
    vp1(n1,vp(vp(V,[]),A))))))) -->
        enp(N),[V],eadv(A).
/*intransitive with adv: bill moved yesterday */
```

Rules specifying well-formed GB trees

As an alternative approach one may use Node Admissibility Condition *(nac)* rules in the Prolog implementation of GB. The following node admissibility rules specify which Swedish syntactic s-trees are well-formed according to GB X-bar theory. These rules deviate somewhat from the previous rules, following Falk 1991 more closely.

```
nac(cp,cp(Fund,c1(Comp,Ip))) :- nac(ip,Ip).
nac(ip,ip(SpecIP,I1(I,Vp))) :- nac(vp,Vp).
nac(vp,vp(v1(V))). % Impersonal verb
nac(vp,vp(Np,V1)) :- nac(v1,V1).
nac(vp,vp(Sadv,Vp)) :- nac(vp,Vp).
    % Sentence adverb adjoined
nac(vp,vp(Vp,Adv)) :- nac(vp,Vp).
    % Other adverb (ADV,PP,CP)
nac(v1,v1(V)). % 1-place (intransitive) verb
nac(v1,v1(V,Arg)). % 2-place (transitive) verb
nac(v1,v1(Aux,Vp)) :- nac(vp,Vp). % Auxiliary
nac(ap,ap(a1(A))). % Impersonal adjective
nac(ap,ap(Np,A1)) :- nac(a1,A1).
nac(a1,a1(Adj)). % Adjective without complement
nac(a1,a1(A,Acomp)). % Adjective with complement (NP,PP)
nac(pp,pp(SpecPP,p1(P,Np))). % With specifier
nac(np,np(N1)) :- nac(n1,N1). % No specifier
nac(np,np(Np,N1)) :- nac(n1,N1). % Specifier
nac(n1,n1(N)). % No complement
nac(n1,n1(N,Ncomp)).
    % Complement (PP, subclause, infinitive)
nac(n1,n1(AP,N1)) :- nac(ap,AP), nac(n1,N1).
    % Adj attrib adjoined
```

Deriving GB d-structure without passing s-structure and transformations

It is clearly possible – and in fact much simpler – to arrive at a GB d-structure of a sentence without deriving first an s-structure with all the traces in the proper places and then passing through all the transformations in reverse. The problem of finding the proper s-structure does not have a straightforward solution. It is also much simpler to go directly from the d-structures to the sentences in sentence generation. This approach is mentioned in Sigurd 1990. Using the DCG formalism it is sufficient to describe which of the d-structures the combinations of categories specified to the right correspond to. The d-structure `cp(A,c1(B,ip(C...` appears as an argument within parentheses to the left of the arrow. The following Swedish rules illustrate how this is done, using the same categories and lexical items as before (for the last more compact rule some additional categories are needed).

```
/* GB d-structure direct without s-structure and
transformations,for the same types of sentences */

sent(cp([],c1([],ip([],i1(T,
    vp1(Subj,vp(V,[]))))))) -->
                                np(Subj),
                                vi(V), in(T). /* intrans */

sent(cp([],c1([],ip([],i1(T,
    vp1(Subj,vp(A,vp(V,[])))))))) -->
                                np(Subj),
                                aux(A),in(T),
                                vi(V). /* aux + intrans */

sent(cp([],c1([],ip([],i1(T,
    vp1(Subj,vp(V,Obj))))))) -->
                                np(Subj),
                                vt(V), in(T), /* trans */
                                np(Obj).

sent(cp([],c1([],ip([],i1(T,
    vp1(Subj,vp(vp(V,[]),Adv))))))) -->
                                np(Subj),
                                vi(V), in(T), /* intrans */
                                adv(Adv). /* with adv */
```

```
sent(cp([],c1([],ip([],i1(T,
    vps(Sadv,vp1(Subj,vp(V,[]))))))))) -->
                                   np(Subj),
                                   vi(V), in(T), /* intrans */
                                   sadv(Sadv). /* with sadv */

sent(cp([],c1([],ip([],i1(T,
    vps(Sadv,vp1(Subj,vp(V,Obj)))))))) -->
                                   np(Subj),
                                   vt(V), in(T), /* trans */
                                   sadv(Sadv), /* with sadv */
                                   np(Obj).

sent(cp([],c1([],ip([],i1(T,
    vp1(Subj,vp(vp(V,[]),Adv))))))) -->
                                   adv(Adv),
                                   vi(V),in(T), /* intrans */
                                   np(Subj). /* with initial adv */

sent(cp(q,c1([],ip([],i1(T,vp1(Subj,vp(V,[]))))))) -->
                                   vi(V),in(T), /* intrans */
                                   np(Subj). /* question */
```

The following is a more compact alternative rule for transitive sentences with or without adv. It is written in the GWOG style (Generalized Word Order Grammar) presented in Sigurd 1990. Some necessary extra categories (subj, obj, advl) are added after the rule.

```
sent(cp([],
c1([],ip([],i1(T,vp1(Subj,vp(vp(V,Obj),Adv)))))))) -->
                                        (np(N1);adv(A1)),
                                        vt(V),in(T),
                                        subj(N2),
                                        obj(N3),
                                        advl(A2),
    {(nonvar(N1),Subj=N1,N2=[],Obj=N3,Adv=A2;
    nonvar(N1),Obj=N1,Subj=N2,N3=[],Adv=A2;
    nonvar(A1),Adv=A1,Subj=N2,Obj=N3,A2=[])}.
    /* assignment of functional roles */

subj([]) --> [].
subj(Np) --> np(Np).

obj([]) --> [].
obj(Np) --> np(Np).

advl([]) --> [].
advl(Adv) --> adv(Adv).
```

These rules for Swedish can be used in conjunction with the previous lexical and transfer rules to translate into English. They insert the values of the different categories directly into the GB tree, which is considered merely as a way of recording some syntactic characteristics of the sentences analysed. The direct rules do not offer a level from which the word order of the sentence can be projected as shown above. Nor do they offer any set of explicit transformations that could be taken to be a characteristic of the language. The direct rules do not offer explicit points where the differences between languages can be pinpointed as nicely as in GB grammar. But the deep structure differences between languages can be kept even in direct d-derivation. The deep structure differences assumed, e.g. the generation of *aux* in the *i1* node would, however, never have been posited if there had not been a transformational component, where the surface order is arrived at through a series of cooperating movement transformations.

Transfer rules for translation

When used in automatic translation between English and Swedish, the d-structure trees of the source language must be changed. The most important change is the change of lexical items. In most cases this can be handled by simple rules, such as the following, where the English term is found first.

```
eslex(move,flytta).
eslex(like,gilla).
```

One may also handle the translation of the tense marked *d* into *de* in this way:

```
eslex(d,de).
```

However, as mentioned before, it is necessary also to allow *did* to be translated as *de*, to be used in negated sentences, where English cannot use simple *d*. Our transfer and lexical rules can also be used in reverse, and in that case the processor of English may e.g. first try to use *d* as the equivalent of Swedish *de* in a negative sentence, but then must settle for the alternative *did*, as in *Bill did not move*. The following is the general format for transfer rules, where many lexical transfer rules are called upon by *eslex*. In a more sophisticated system with complex NPs, more complex transfer rules are of course needed.

```
estransf(X,Y) :-
     X=cp(A,c1(B,ip(C,i1(D,vp1(E,vp(F,G)))))),
     eslex(A,K),eslex(B,L),eslex(C,M),
     eslex(D,N),eslex(E,O),
     eslex(F,P),eslex(G,Q),
     Y=cp(K,c1(L,ip(M,i1(N,vp1(O,vp(P,Q)))))).
```

Transfer between Swedish and English is handled by the following rule:

```
setransf(X,Y) :- estransf(Y,X).
```

The following detailed lexical transfer rules are needed in order to translate using our two simple grammars.

```
eslex(bill,bill).
eslex(eva,eva).
eslex(who,vem).
eslex(d,de).
eslex([di,d],de). /* do support if no other aux*/
eslex(coul,kun).
eslex(move,flytta).
eslex(like,gilla).
eslex([],[]).
eslex(q,q).
eslex(nt,inte).
eslex(perhaps,kanske).
eslex(yesterday,igår).
eslex(that,att).
```

Predicates and commands for interaction

The following are some useful predicates for interaction with the system. The printout of some processes shows which transformations have been applied in the analysis, the deep structure of the source, and the target sentences before and after transfer.

```
etrans(X,R) :-
     eiatt(X,Y),efmove(Y,W),esmove(W,V),ewhfilter(V,R)
.
/* Calls on individual transformations in due order
*/

eretrans(R,U) :- esmove(W,R),print(spassed),nl,
     efmove(V,W),print(fpassed),nl,
     eiatt(U,V),print(ipassed),nl.
/* Calls on transformations in reverse order and
reports when each has been passed */
```

```
/* Some predicates for analysis of English sentences
and translation between English and Swedish */

eanal(X,Z) :- eproj(Y,X,[]),eretrans(Y,Z).
/* Analyses the sentence and derives tree from
string, running the transformations backwards, eg
eanal([bill,[like,d],eva],X) */

estra(X,Y) :-
     eproj(F,X,[]),eretrans(F,F2),estransf(F2,F3),
     trans(F3,F4),proj(F4,Y,[]).
/* Translates between English and Swedish via d-
structure */

/* Some tests */
epstest :- ed(F,X,[]), print(X),nl,print(F),nl.
/* generates d-trees and prints the underlying base
sentence and the tree */

etest :- ed(F,X,[]),print(F),nl,etrans(F,Y),
print(Y). /* generates sample d-tree, transforms it
and prints the resulting s-structure*/

etest1 :-
     ed(F,X,[]),etrans(F,Y),eproj(Y,Z,[]),print(Z).
/* generates d-structure, transforms it, and projects
the s-structure */
```

Translatable English and Swedish sentences

The following are some sentences that can be analysed, generated and translated (in both directions). Note that the verb stem and ending must be included in [ ], e.g. *[coul,d]*. (This can, of course, be avoided by adding some cosmetic operations.)

English	*Swedish*
[bill,[move,d]]	[bill,[flytta,de]]
[bill,[move,d],yesterday]	[bill,[flytta,de],igår]
[yesterday,bill,[move,d]]	[igår,[flytta,de],bill]
[who,[move,d]]	[vem,[flytta,de]]
[[di,d],bill,move]	[[flytta,de],bill]
[bill,[di,d],nt,move]	[bill,[flytta,de],inte]
[bill,[like,d],eva]	[bill,[gilla,de],eva]
[bill,[like,d],eva,yesterday]	[bill,[gilla,de],eva,igår]
[yesterday,eva,[like,d],bill]	[igår,[gilla,de],eva,bill]

[eva,[di,d],nt,like,bill]	[eva,[gilla,de],inte,bill]
[eva,[coul,d],like,bill]	[eva,[kun,de],gilla,bill]
[who,[like,d],bill]	[vem,[gilla,de],bill]
[whom,[di,d],bill,like]	[vem,[gilla,de],bill]
[[di,d],eva,like,bill]	[[gilla,de],eva,bill]
[that,bill,[move,d]]	[att,bill,[flytta,de]]

Conclusion

The GB model can be implemented without too much difficulty, and such implementations can be used in order to test the generative power of the base rules, the ordering or the conditions of transformations, etc. Writing grammatical rules to be tested in a computer program requires great care, detail and regard for consequences. On the other hand, the technical and programming problems tend to dominate when the grammar rules are to fit the computer. The implementation presented here models the base structures and movement transformations suggested by GB, but some of the principles and constraints are concealed in the transformations. The program may in a sense be said to simulate GB grammar. Government, as reflected in case, and binding, as reflected in pronouns, are not implemented, but certainly can be. As the list of references shows, GB implementations have focused on different aspects of the theory.

The need in GB for a set of parsing rules to arrive at surface structures with all the traces required by the theory adds to the difficulty of using the model for parsing. We have written such rules in the DCG formalism and tested some alternatives. One could, in fact, arrive at the required GB d-structures more easily by one set of DCG rules which derive d-structures directly without calling several transformations. Such an approach, which may be called Direct GB Parsing (DIG), conceals the movements of the surface constituents to the places they are to have in the d-structure. The attraction of GB rests on the belief in the universality and psychological reality of the d-structure suggested with certain basic nodes storing mode, finiteness, subordination, tense, sentential adverbs, SVO, etc. as well as the belief in the universality of a small set of movement transformations supplemented with certain universal principles and constraints.

A copy of the implementation presented is available at the Department of Linguistics, University of Lund.

References

Abney, S. & J. Cole. 1985. 'A Government-binding parser'. *Proc. of the North Eastern Linguistic Society XVI.*

Berwick, R. & A. Weinberg. 1984. *The grammatical basis of linguistic performance.* Cambridge, Mass: MIT Press.

Chen, H. 1990. 'A logic-based government-binding parser for Mandarin Chinese'. *Proc COLING 13,* vol 2, 48-53. Helsinki.

Chomsky, N. 1981. *Lectures on government and binding.* Dordrecht:Foris.

Chomsky, N. 1986. *Barriers.* Cambridge, Mass: MIT Press.

Dooley Collberg, S. 1991. *Comparative studies in current syntactic theories.* Working papers 37. Lund: Dept of Linguistics, Lund University.

Falk, C. 1991. *Modern grammatisk teori.* Lund: Inst för nordiska språk.

Kuhns, R. J. 1986. 'A Prolog implementation of Government-Binding theory'. *Proc COLING 11,* 546-550. Bonn.

Latecki, L. 1991. 'An indexing technique for implementing command relations'. *Proc of the 5th Conf of the European chapter of ACL,* 39-44. Berlin.

Pritchett, B. & J. Reitano. 1990. 'Parsing with on-line principles: a psychologically plausible, object oriented approach'. *Proc COLING 13,* vol 3, 437-439. Helsinki.

Sells, P. 1985. *Lectures on contemporary syntactic theories.* Stanford: CSLI.

Sigurd, B. 1990. 'Implementing the generalized word order grammars of Diderichsen and Chomsky'. *Proc COLING 13,* vol 2, 336-340. Helsinki.

Stabler, E. 1987. 'Restricting logical grammars with binding theory'. *Computational linguistics,* Vol 13, 1-2.

Wehrli, E. 1988. 'Parsing with GB-grammar'. *Natural language parsing and linguistic theories,* eds. U. Reyle & C. Rohrer. Dordrecht: Reidel.

Acknowledgement

We are indebted to Barbara Gawrońska and Christer Platzack for valuable comments on this paper and the program presented.