A NEW DESIGN OF PROLOG-BASED BOTTOM-UP PARSING SYSTEM WITH GOVERNMENT-BINDING THEORY

Hsin-Hsi Chen*,**, I-Peng Lin* and Chien-Ping Wu**

* Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

** Graduate Institute of Electrical Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

abstract

This paper addresses the problems of movement transformation in Prolog-based bottom-up parsing system. Three principles of Government-Binding theory are employed to deal with these problems. They are Empty Category Principle, C-command Principle, and Subjacency Principle. A formalism based upon them is proposed. Translation algorithms are given to add these linguistic principles to the general grammar rules, the leftward movement grammar rules, and the rightward movement grammar rules respectively. This approach has the following specific features: the uniform treatments of leftward and rightward movements, the arbitrary number of movement non-terminals in the rule body, and automatic detection of grammar errors before parsing. An example in Chinese demonstrates all the concepts.

1. Introduction

The movement transformation is one of the major problems encountered in natural language processing. It has relation to the empty constituents (traces) that exist at various levels of representation in natural language statements. Consider the following example in Chinese:

那本書， 我 看 過 了。 (That book, I read.)

The word "看" (read) is a transitive verb, which should take a direct object. However, the object "那本書" (that book) is topicalized to the first position of the sentence. For the treatment of this phenomenon, we cannot just write down the rules:

sentence --> noun-phrase,verb-phrase.
verb-phrase --> transitive-verb,noun-phrase.
verb-phrase --> transitive-verb.
verb-phrase --> intransitive-verb.

This is because many ungrammatical sentences will be accepted. Thus, we must provide some mechanisms in the grammars in order to capture them. It is still a hard work to do. Several difficulties are listed as follows:

(1) The determination of movement is difficult. That is, an element may be in a topicalization position, but it is not moved from some other place in the sentence. For example,

水果， 我 喜歡。
(Fruit, I like.)
水果， 我 喜歡 香蕉。
(As for fruit, I like banana.)

the first can be considered to be a movement phenomenon, but the second cannot.

(2) The empty constituent may exist at many possible positions. For example, given an n-word sentence such as

$e_1\ w_1\ e_2\ w_2\ e_3\ \cdots\ e_{(n-1)}\ w_{(n-1)}\ e_{(n)}\ w_{(n)}\ e_{(n+1)}$

where w_i is the i-th word and e_i is an empty constituent, there are (n+1) possible positions from which the moved constituent may originate. That is, for a moved constituent (if there is any), there are so many possible empty constituents to co-index.

(3) Since the gap in between the moved constituent and its corresponding trace is arbitrary, it is implausible to list all the possible movements exhaustively, and specify each movement constraint explicitly in the grammars.

The Government-Binding (GB) theory [1] provides universal principles to explain the movements. Some of them are shown as follows:

(1) Empty Category Principle [7] -
A trace must be properly governed.
(2) C-command Principle [7] -
α c-commands β iff every branching node dominating α dominates β.
(3) Subjacency Principle [7] -
Any application of move - α may not cross more than one bounding node.

Summing up the above principles, we have to find a moved constituent to c-command a trace. The constituent can neither relate to a trace out of its c-command domain, nor match a trace when more than one bounding node is crossed. Such principles narrow down the searching space to some extent. For example,

(E1) 那 個 人 看見 t_i 的 學生 來 了。
(The student that the man saw t_i came.)
(E2) * t_m 看見 t_n 的 學生 來 了 的 那 個 人。
(* The man that the student who t_m saw t_n came.)

There is a trace t_i in the example (E1). Two NPs, i.e. "學生" (the student) and "那個人" (the man), may co-index with it, but only the former is acceptable. The reason is specified as below:

(E1') $[_{n''}[_S$那個人 (the man)看見 (saw)t_i]的 (de)學生$_i$ (the student)]來 (came)了 (asp)
(O)

(E1'') 那個人$_j$(the man) $[_{n''}[_S t_j$看見 (saw)t_i]的 (de)學生$_i$ (the student)]來 (came)了 (asp)
(X) (O)

The (E1") interpretation violates the subjacency principle (assuming that *s* and *n"* are two bounding nodes). Two traces exist in the example (E2). The traces t_m and t_n may co-index with the two NPs "學生" (the student) and "那個人" (the man), and vice versa. However, both are wrong because of the subjacency principle:

(E2') $[_S[_{n'}[_S t_m$看見 (saw)t_n]的 (de)學生$_n$(the student)]來 (came)了 (asp)]的 (de)那個人$_m$(the man)
(O) (X)

(E2'') $[_S[_{n'}[_S t_m$看見 (saw)t_n]的 (de)學生$_m$(the student)]來 (came)了 (asp)]的 (de)那個人$_n$(the man)
(O) (X)

2. A Government-Binding based Logic Grammar Formalism

2.1 The specifications of grammar formalism

The Government-Binding based Logic Grammar (GBLG) formalism is specified informally as follows:

(1) the general grammar rules -
(a) $c(Arg) \rightarrow c_1(Arg_1), c_2(Arg_2), \ldots, c_n(Arg_n)$.
where c(Arg) is a phrasal non-terminal, and may be also a bounding non-terminal,
$c_j(Arg_j)$ $(1 \le j \le n)$ is a lexical terminal or a phrasal non-terminal.

(b) c(Arg) --> $c_1(Arg_1)$,$c_2(Arg_2)$,...,$c_i(Arg_i)$,
trace(TraceArg),$c_{(i+1)}(Arg_{(i+1)})$,...,$c_n(Arg_n)$.
where the definitions of c(Arg) and
$c_j(Arg_j)$ ($1 \le j \le n$) are the same as above,
trace(TraceArg) is a virtual non-terminal.

The special case *i=0* is common. For example, a noun phrase is topicalized from a subject position. It is represented as *s --> trace,np.*

(2) the leftward movement grammar rules -
c(Arg) --> $c_1(Arg_1)$,$c_2(Arg_2)$,...,$c_i(Arg_i)$,
$m(Arg_m)$<<<trace(TraceArg),
$c_{(i+1)}(Arg_{(i+1)})$,...,$c_n(Arg_n)$.
where the definitions of c(Arg) and
$c_j(Arg_j)$ ($1 \le j \le n$) are the same as 1(a),
$m(Arg_m)$<<<trace(TraceArg) is a movement non-terminal.

When i=0, the movement non-terminal is the first element in the rule body.

(3) the rightward movement grammar rules -
c(Arg) --> $c_1(Arg_1)$,$c_2(Arg_2)$,...,$c_i(Arg_i)$,
trace(TraceArg)>>>$m(Arg_m)$,
$c_{(i+1)}(Arg_{(i+1)})$,...,$c_n(Arg_n)$.

Except that the operator '>>>' is used, the other definitions are the same as those in the leftward movement rules. It is apparent because of the uniform treatments of the leftward and the rightward movements.

2.2 A sample grammar

A sample grammar GBLG1 for Chinese shown below introduces the uses of the formalism:

```
( r1 ) s1bar(s1bar(Topic,S)) -->
        topic(Topic) <<< traceT(Topic),s(S).
( r2 ) s1bar(s1bar(S)) --> s(S).
( r3 ) s(s(N2bar,V2bar,Part)) -->
        n2bar(N2bar),v2bar(V2bar),* part(Part).
( r4 ) s(s(N2bar,V2bar)) --> n2bar(N2bar),v2bar(V2bar).
( r5 ) s(s(traceR(Trace),V2bar)) -->
        traceR(Trace),v2bar(V2bar).
( r6 ) topic(topic(N2bar)) --> n2bar(N2bar).
( r7 ) n2bar(n2bar(Det,CL,N1bar)) -->
        * det(Det),* cl(CL),n1bar(N1bar).
( r8 ) n2bar(n2bar(N1bar)) --> n1bar(N1bar).
( r9 ) n1bar(n1bar(Rel,N2bar)) -->
        rel(Rel),traceR(N2bar) >>> n2bar(N2bar).
(r10) n1bar(n1bar(N)) --> * n(N).
(r11) rel(rel(S,De)) --> s(S),* de(De).
(r12) v2bar(v2bar(Adv,V1bar)) --> * adv(Adv),v1bar(V1bar).
(r13) v2bar(v2bar(V1bar)) --> v1bar(V1bar).
(r14) v1bar(v1bar(TV,N2bar)) --> * tv(TV),n2bar(N2bar).
(r15) v1bar(v1bar(TV,traceT(Trace))) -->
        * tv(TV),traceT(Trace).
(r16) v1bar(v1bar(TV,traceR(Trace))) -->
        * tv(TV),traceR(Trace).
(r17) v1bar(v1bar(IV)) --> * iv(IV).
```

Among these grammar rules, (r1) deals with the leftward movement (topicalization), (r9) treats the rightward movement (relativization), and the others are normal grammar rules. The heads of the grammar rules (r3), (r4), (r5), (r7), and (r8) are bounding nodes. The virtual non-terminals *traceT(Trace)* and *traceR(Trace)* appear in the rules (r5), (r15), and (r16).

2.3 Transitive relation of c-command theory

For a phrasal non-terminal X, a virtual non-terminal Y and a transitive relation TR, X TR Y if

(1) X is the rule head of a grammar rule, and Y is an element in its rule body, or

(2) X is the rule head of a grammar rule, a phrasal non-terminal I in its rule body, and I TR Y, or

(3) there exists a sequence of phrasal non-terminals I_1, I_2, ..., I_n, such that X TR I_1 TR I_2 TR ... TR I_n.

The transitive relation TR is also a dominate relation.

The c-command theory is embedded implicitly in the GBLGs if every grammar rules satisfy the following property:

for a rule X_0 --> $X_1,X_2,...,X_m$ where X_i is a terminal or a non-terminal, $1 \le i \le m$, if X_i=(A<<<B) then there must exist some X_j ($i< j \le m$), such that X_j dominates the virtual non-terminal B in other grammar rule. That is, X_j TR B. The phrasal non-terminal X_0 is the first branching node that dominates A and X_j, and thus also dominates B. Therefore, A c-commands B. X_i=(B>>>A) has the similar behavior. Rules (r1) and (r9) in grammar GBLG1 show these <<< and >>> relations respectively.

2.4 Comparison with other logic programming approaches

Compared with other logic programming approaches, especially the RLGs [8,9], the GBLGs have the following features:

(1) the uniform treatments of leftward movement and the rightward movement -

The direction of movement is expressed in terms of movement operators <<< or >>>. The interpretation of movement non-terminals A <<< B or B >>> A is

If A is a left moved constituent (or a right moved constituent), then the corresponding trace denoted by B should be found after (or before) A <<< B (or B >>> A). It is illustrated in the Fig. 1. The two trees are symmetric and the corresponding rules are similar. However, the rules are not similar in RLGs. That is, the two types of movements are not treated in the same way. For the rightward movement, a concept of adjunct node is introduced. It says that the right moved constituent is found if the rule hung on the adjunct node is satisfied. The operation semantics is enforced on the writing of the logic grammars. It destroys the declarative semantics of logic grammars to some extent.

(2) the arbitrary number of movement non-terminals in the rule body -

In our logic grammars, the number of movement non-terminals in a rule is not restrictive if the rule satisfies the property specified in the last section. The RLGs allow at most one movement non-terminal in their rules. The position of movement non-terminal is declared in the rule head. It is difficult for a translator to tell out the position if different types

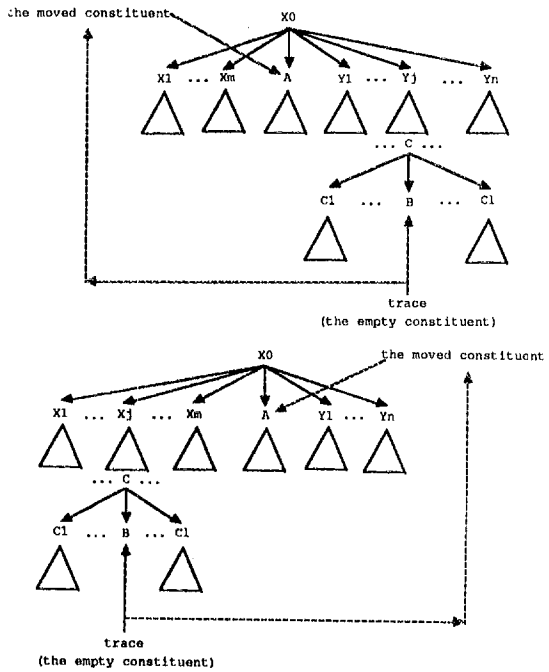

Fig. 1 Symmetric tree for leftward and rightward movement

of elements are interleaved in the rule body. Thus, our formalism is more clear and flexible than RLGs'.

(3) automatic detection of grammar errors before parsing

For significant grammar rules, a transitive relation TR must be satisfied. The violation of the transitive relation can be found beforehand during rule translation. Thus, this feature can help grammar writers detect the grammar errors before parsing.

3. A Bottom-up Parser in Prolog

3.1 Problem specifications

The Bottom-Up Parsing system (BUP) [2,3,4] uses the left-corner bottom-up algorithm to implement Definite Clause Grammars (DCGs) [5]. It overcomes the problems of top-down parsing, e.g. the left-recursive invocation, and provides an efficient way as Earley's and Pratt's algorithms [3]. However, it does not deal with the important syntactic problem - movement transformation. Extraposition Grammars (XGs) [6] propose extraposition lists (x-lists) to attack the movement problem, but when to extract traces from x-lists becomes a new obstacle [8,9]. Restricted Logic Grammars (RLGs) [8,9] based upon GB try to tackle the unrestricted extraction from the x-list. They emphasize the importance of the c-command and the subjacency principles during parsing. The extraction must obey these two principles. The parsing strategies of XGs and RLGs are all depth-first and left-to-right, thus they have the same drawbacks as DCGs do [4]. If the parsing strategy is left-corner bottom-up, the following issues have to be considered in the translation of GBLGs:

(1) the empty constituent problem -

The first element in the rule body, which acts as a left-corner, should not be empty in left-corner bottom-up algorithm. However, the type 1(b) of rules is common.

(2) the transfer of trace information -

From Fig. 1, we know that the positions of empty constituents are usually lower than those of moved constituents. Because the parsing style is bottom-up, the trace information must be transferred up from the bottom. The conventional different list cannot be applied here. Fig. 2 and Fig. 3 illustrates the differences of data flow between top-down parsing and bottom-up parsing.

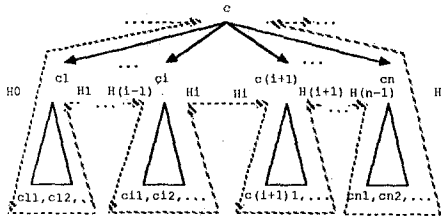

Fig. 2 the data flow in the top-down parsing

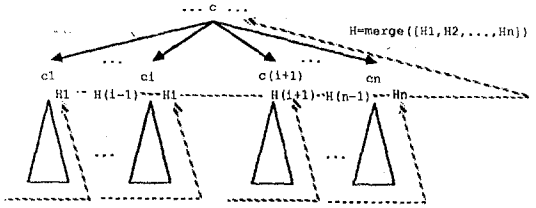

Fig. 3 the data flow in the bottom-up parsing

3.2 Data structure

The transfer of trace information is through a list called *extraposition list* (*x-list*) and denoted by a symbol *H*. The transformation of x-list is bottom-up. Fig. 3 sketches the concept. A special data structure shown below is proposed to carry the information:

[[a sequence of trace information|X],X]

Note a tail variable *X* is introduced. Based upon this notation, an empty list is represented as *[Z,Z]*. An algorithm that merges arbitrary number of lists in linear time is designed:

```
merge(X,Y) :- merge(X,Y,[Z,Z]).
merge([],L,L) :- !.
merge([[B,X]|T],Y,[A,B]) :- merge(T,Y,[A,X]).
```

In the conventional list structure such as

[a sequence of trace information]

even though the difference list concept is adopted, the computation time is still in proportion to $m_1+m_2+...+m_n$, where m_i ($1 \le i \le n$) is the number of elements in the i-th list.

Although our merge algorithm is the fastest, it is still a burden on the parsing. In most cases, the predicate merges empty lists. That is nonsense. To enhance the parsing speed, the *merge* predicate is added in which place it is needed. Observing the merge operation, we can find that it is needed only when the number of lists to be merged is greater than one. The following method can decrease the number of x-lists during rule translation, and thus delete most of the unnecessary *merges*:

Partition the basic elements in the logic grammars into two mutually exclusive sets: carry-H set and non-carry-H set. The elements in the carry-H set may contribute trace information during parsing, and those in the non-carry-H set do not introduce trace information absolutely. The transitive relation TR defined in the section 2.3 tells us which phrasal non-terminals constitute the carry-H set.

3.3 The translation of grammar rules

The translation of basic elements in the GBLGs are similar to BUPs. Only one difference is that an extra argument that carries trace information may be added to phrasal non-terminal if it belongs to carry-H set. Appendix lists the translated results of the grammar GBLG1.

3.3.1 The general grammar rules

The general grammar rules are divided into two types according as a virtual non-terminal disappears or appears in the rule body:

(a) $c(Arg) \dashrightarrow c_1(Arg_1),c_2(Arg_2),...,c_n(Arg_n)$.

When c is not a bounding node, e.g. rule (r2), the translation is the same as that in BUP [2,3,4], except that an extra argument *H* (if necessary) for x-list and a built-in predicate *merge* are added in the new translation algorithm. This predicate is used to merge all the x-lists on the same level. The transformation of x-lists is bottom-up (only one direction) as shown in Fig. 3. Thus, the rule (a) is translated into

```
c1(G,Arg1,H1,X1,X) :-
    goal(c2,Arg2,H2,X1,X2),
    ...,
    goal(cn,Argn,Hn,X(n-1),Xn),
    merge([H1,H2,...,Hn],H),
    c(G,Arg,H,Xn,X).
```

When *c* is a bounding node, e.g. rule (r4), the information is used to check the x-list transferred up. Thus, an extra predicate *bound* is tagged to this type of rules:

```
c1(G,Arg1,H1,X1,X) :-
    goal(c2,Arg2,H2,X1,X2),
    ...,
    goal(cn,Argn,Hn,X(n-1),Xn),
    merge([H1,H2,...,Hn],H),
    bound(c,H),
    c(G,Arg,H,Xn,X).
```

The predicate *bound* implements the subjacency principle. Its definition is:

```
bound(C,[X,Y]) :- (var(X),!;boundaux(C,X)).
boundaux(C,X) :- var(X),!.
boundaux(C,[x(Trace,Bound,Direction)|Xs]) :-
    (var(Bound),!,Bound=C,boundaux(C,Xs);
     Bound=s,C=s,!,boundaux(C,Xs);
     fail).
```

A variable *Bound* which records the cross information is reserved for each element in the x-list. When a bounding node is crossed, this variable is checked to avoid the illegal operation.

(b) $c(Arg) \dashrightarrow c_1(Arg_1),c_2(Arg_2),...,c_i(Arg_i)$, trace(TraceArg),$c_{(i+1)}(Arg_{(i+1)}),...,c_n(Arg_n)$.

where i >= 0. The rules (r5) and (r15) are two examples.

If the left-corner bottom-up parsing algorithm is used, the grammar rules should free of empty constituents. When i=0, the grammar rule considers a trace (an empty constituent) to be the first element in the rule body. It overrides the principle of the algorithm, but we can always select the first element $c_{(i+1)}$ that satisfies the following criterion:

(1) a lexical terminal, or

(2) a phrasal non-terminal, or

(3) a phrasal non-terminal in a movement non-terminal,

to be the left-corner and put the trace information into an x-list before this non-terminal. Thus, the translation is generalized as follows (assume that *c1* is a left-corner).

```
c1(G,Arg1,H1,X1,X) :-
    goal(c2,Arg2,H2,X1,X2),
    ...,
    goal(ci,Argi,Hi,X(i-1),Xi),
    goal(c(i+1),Arg(i+1),H(i+1),Xi,X(i+1)),
    ...,
    goal(cn,Argn,Hn,X(n-1),Xn),
    merge([H1,H2,...,Hi,
           [[x(trace(TraceArg),Bound,D)|Z],Z],
          H(i+1),...,Hn],H),
    c(G,Arg,H,Xn,X).
```

Here, the trace information is placed between Hi and H(i+1). Summing up, the virtual non-terminal is represented as a fixed format:

x(trace(TraceArg),Bound,Direction)

and placed into x-list via merge operation. The position in x-list is reflected from the original rule.

3.3.2 The leftward movement grammar rules

The leftward movement grammar rules can be generalized as below:

$c(Arg) \dashrightarrow c_1(Arg_1),c_2(Arg_2),...,$
$c_i(Arg_i)<<<trace(TraceArg),c_{(i+1)}(Arg_{(i+1)}),...,$
$c_n(Arg_n).$

The rule (r1) is an example. Its translation is shown as follows:

```
c1(G,Arg1,H1,X1,X) :-
    goal(c2,Arg2,H2,X1,X2),
    ...,
    goal(ci,Argi,Hi,X(i-1),Xi),
    goal(c(i+1),Arg(i+1),H(i+1),Xi,X(i+1)),
    ...,
    goal(cn,Argn,Hn,X(n-1),Xn),
    merge([H(i+1),...,Hn],T1),
    cut_trace(x(trace(TraceArg),Bound,left),
              T1,T2),
    merge([H1,H2,...Hi,T2],H),
    c(G,Arg,H,Xn,X).
```

Comparing this translation with that of general grammar rules, we can find a new predicate *cut_trace* is added. The *cut_trace* implements the c-command principle, and its definition is:

```
cut_trace(Trace,[Y,X],[Y1,X]) :-
    (var(Y),!,
     (Trace=x(TraceInfo,Bound,left),!;fail);
     cut_traceaux(Trace,Y,Y1)).
cut_traceaux(Trace,[Trace|Xs],Xs) :- !.
cut_traceaux(Trace,[H|X],[H|Y]) :-
    (var(X),!,
     (Trace=x(TraceInfo,Bound,left),!;fail);
     cut_traceaux(Trace,X,Y)).
```

The *cut_trace* tries to retract a trace from x-list if a movement exists. Mandarin Chinese has many specific features that other languages do not have. For example, topic-comment structure does not always involve movement transformation. The first *cut_traceaux* clause matches the trace information with the x-list transferred from the bottom on its right part. The second *cut_traceaux* tells us that if the expected leftward trace cannot match one of the elements in the x-list, then it will be drop out. The x-list is not changed and transferred up. The concept is demonstrated in Fig. 4. It also explains why we can detect grammar errors before parsing. In summary, each movement non-terminal is decomposed into a phrasal non-terminal and a virtual non-terminal. The phrasal non-terminal is translated the same as before. The virtual non-terminal is represented as

x(trace(TraceArg),Bound,left)

In this case, however, *cut_trace* is involved instead of *merge*.

3.3.3 The rightward movement grammar rules

Because we treat the leftward and the rightward movement grammar rules in a uniform way, the translation algorithm of both are similar. The rightward movement grammar rules are with the following format:

$c(Arg) \dashrightarrow c_1(Arg_1),c_2(Arg_2),...,$
$trace(TraceArg)>>> c_i(Arg_i),$
$c_{(i+1)}(Arg_{(i+1)}),...,c_n(Arg_n).$

The rule (r9) is an example. The corrsponding translated result is:

```
c1(G,[Arg1],H1,X1,X) :-
    goal(c2,[Arg2],H2,X1,X2),
    ...,
```

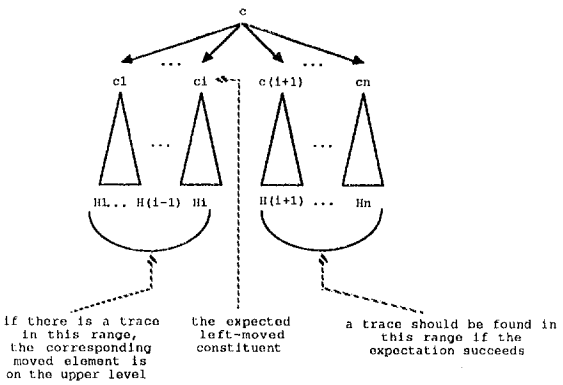

Fig. 4 the sketch of the translation of the leftward production rules

```
    goal(c(i-1),[Arg(i-1)],H(i-1),X(i-2),X(i-1)),
    merge([H1,H2,...,H(i-1)],T1),
    cut_trace(x(trace(TraceArg,Bound,right)),
              T1,T2),
    goal(ci,[Argi],Hi,X(i-1),Xi),
    ...,
    goal(cn,[Argn],Hn,X(n-1),Xn),
    merge([T2,Hi,...,Hn],H),
    c(G,[Arg],H,Xn,X).
```

The translation is very apparent for the symmetric property of the leftward and the rightward grammar rules illustrated in Fig. 4 and Fig. 5. A slight difference appears in the definition of the

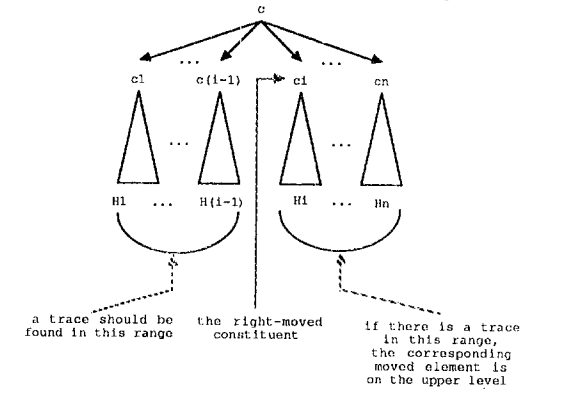

Fig. 5 the sketch of the translation of the rightward production rules

predicate *cut_trace*. It shows an important linguistic phenomenon in Mandarin Chinese: '*Relativization is always a movement transformation.*' Thus, if we expect a trace and cannot find a corresponding one, failure is issued. The direction information in *x(trace(TraceArg),Bound, right)*, i.e. right, tells out the difference between the leftward and the rightward movements. In general, we allow both leftward movement and rightward movement to appear in the same rule. A new predicate *intersection* is introduced to couple these two translations.

3.4 Invocation of the parsing system

The parsing system is triggered in the following way:

goal(*a start non-terminal*,[*a sequence of arguments*], *an empty x-list*,[*a sequence of input string*],[]).

In GBLG1, the invocation is shown as follows:

goal(s1bar,[ParseTree],[Z,Z],[input sentence],[]), var(Z).

Because an empty x-list is represented as [Z,Z] (Z: a variable) in our special data structure shown in Section 3.2, var(Z) verifies its correctness. For example, to parse the Chinese sentence "那 個 人 看見 的 學生 來 了" (the student that that man saw came), we trigger the parser by calling:

```
?- goal(s1bar,[S1bar],[Z,Z],['那','個','人','看見',
    '的','學生','來',' 了'],[]),var(Z).
/*  ?- goal(s1bar,[S1bar],[Z,Z],['that','man','saw',
    'de','student','came','aspect'],[]),var(Z).   */
```

4. Conclusion

This paper addresses the problems of movement transformation in Prolog-based bottom-up parser. Three principles of Government-Binding theory are considered to deal with these problems. They are Empty Category Principle, C-command Principle and Subjacency Principle. A sequence of translation rules is given to add these linguistic principles to the general grammar rules, the leftward movement grammar rules, and the rightward movements grammar rules respectively. The empty constituent problem is solved in this paper to allow the trace to be the first element in the grammar rule body. A special data structure for extraposition list is proposed to transfer the movement information from the bottom to the top. Based upon this structure, the fastest merge algorithm is designed. Those unnecessary *merge* predicates can be eliminated with the help of transitive relation. Thus, the new design not only extends the original bottom-up parsing system with the movement facility, but also preserves the parsing efficiency.

References

[1] N. Chomsky, *Lectures on Government and Binding*, Foris Publication, Dordrecht, Holland, 1981.

[2] Yuji Matsumoto, Hozumi Tanaka, et al., "BUP: A Bottom-Up Parser Embedded in Prolog," *New Generation Computing*, Vol. 1, No. 2, 1983, pp. 145-158.

[3] Yuji Matsumoto, Masaki Kiyono, and Hozumi Tanaka, "Facilities of the BUP Parsing System," in Dahl, V. and P. Saint-Dizier, *Natural Language Understanding and Logic Programming*, 1985, pp. 97-106.

[4] Yuji Matsumoto, Hozumi Tanaka, and Masaki Kiyono, "BUP: A Bottom-Up Parsing System for Natural Languages," in Warren, D.H.D. and M. Canegham (eds.), *Logic Programming and Its Applications*, 1986, pp. 262-275.

[5] F. Pereira and D.H.D. Warren, "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks," *Artificial Intelligence*, Vol. 13, 1980, pp. 231-278.

[6] F. Pereira, "Extraposition Grammars," *American Journal of Computation Linguistics*, Vol. 7, No. 4, 1981, pp. 243-256.

[7] P. Sells, *Lectures on Contemporary Syntactic Theories*, Center for the Study of Language and Information, 1985.

[8] E.P. Stabler, Jr., "Restricting Logic Grammars," *Proc. of the AAAI Conference*, 1986, pp. 1048-1052.

[9] E.P. Stabler, Jr., "Restricting Logic Grammars with Government-Binding Theory," *Computational Linguistics*, Vol. 13, No. 1-2, January-June, 1987, pp. 1-10.

Appendix

Based upon the translation algorithms specified in Section 3, the logic grammar GBLG1 is translated as below. The clause (ti) is the relevant translated result of the grammar rule (ri). Note the codes have been optimized. Those unnecessary *merge* operations are deleted from the translated results.

```
(t1) topic(G,[Topic],H1,X1,X) :-
        goal(s,[S],H2,X1,X2),
        cut_trace(x(traceT(Topic),Bound,left),H2,T1),
        merge([H1,T1],H),
        s1bar(G,[s1bar(Topic,S)],H,X2,X).
(t2) s(G,[S],H,X1,X) :- s1bar(G,[s1bar(S)],H,X1,X).
(t3) n2bar(G,[N2bar],H1,X1,X) :-
        goal(v2bar,[V2bar],H2,X1,X2),
        lookup(part,[Part],X2,X3),
        merge([H1,H2],H),
        bound(s,H),
        s(G,[s(N2bar,V2bar,Part)],H,X3,X).
(t4) n2bar(G,[N2bar],H1,X1,X) :-
        goal(v2bar,[V2bar],H2,X1,X2),
        merge([H1,H2],H),
        bound(s,H),
        s(G,[s(N2bar,V2bar)],H,X2,X).
(t5) v2bar(G,[V2bar],H1,X1,X) :-
        merge([[[x(traceR(Trace),Bound,right)|Z],Z],
              H1],H),
        bound(s,H),
        s(G,[s(traceR(Trace),V2bar)],H,X1,X).
(t6) n2bar(G,[N2bar],H,X1,X) :-
        topic(G,[topic(N2bar)],H,X1,X).
(t7) det(G,[Det],X1,X) :-
        lookup(cl,[CL],X1,X2),
        goal(n1bar,[N1bar],H,X2,X3),
        bound(n2bar,H),
        n2bar(G,[n2bar(Det,CL,N1bar)],H,X3,X).
(t8) n1bar(G,[N1bar],H,X1,X) :-
        bound(n2bar,H),
        n2bar(G,[n2bar(N1bar)],H,X1,X).
(t9) rel(G,[Rel],H1,X1,X) :-
        cut_trace(x(traceR(N2bar),Bound,right),H1,T1),
        goal(n2bar,[N2bar],H2,X1,X2),
        merge([T1,H2],H),
        n1bar(G,[n1bar(Rel,N2bar)],H,X2,X).
(t10) n(G,[N],X1,X) :- n1bar(G,[n1bar(N)],[Z,Z],X1,X).
(t11) s(G,[S],H,X1,X) :-
        lookup(de,[De],X1,X2),
        rel(G,[rel(S,De)],H,X2,X).
(t12) adv(G,[Adv],X1,X) :-
        goal(v1bar,[V1bar],H,X1,X2),
        v2bar(G,[v2bar(Adv,V1bar)],H,X2,X).
(t13) v1bar(G,[V1bar],H,X1,X) :-
        v2bar(G,[v2bar(V1bar)],H,X1,X).
(t14) tv(G,[TV],X1,X) :-
        goal(n2bar,[N2bar],H,X1,X2),
        v1bar(G,[v1bar(TV,N2bar)],H,X2,X).
(t15) tv(G,[TV],X1,X) :-
        v1bar(G,[v1bar(TV,traceT(Trace))],
            [[x(traceT(Trace),Bound,left)|Z],Z],H,X1,X).
(t16) tv(G,[TV],X1,X) :-
        v1bar(G,[v1bar(TV,traceR(Trace))],
            [[x(traceR(Trace),Bound,right)|Z],Z],H,X1,X).
(t17) iv(G,[IV],X1,X) :- v1bar(G,[v1bar(IV)],[Z,Z],X1,X).
```