A Minimalist Head-corner Parser

Mettina Veenstra
University of Groningen, Dept. of Computational Linguistics
P.O. Box 716, 9700 AS Groningen
The Netherlands
e-mail: `mettina@let.rug.nl`

Abstract

In this paper a head-corner parser for a fragment of the Minimalist Program (Chomsky 1992) is described. It will be argued that because of the nature of Generalized Transformations (GT) and Move-α head-corner parsing is a suitable parsing technique for the Minimalist Program.
Furthermore a proposal is made to treat functional and lexical heads differently. Functional heads are not treated as head-corners of their mothers by the minimalist head-corner parser that is described here.

1 The Minimalist Program

1.1 Word Order and Movement

Within the Minimalist Program, Move-α ('move something') is the central mechanism for treating word order. In earlier versions of the Chomskyan theory phrase structure took care of word order differences. $\overline{\text{X}}$-Theory (Chomsky 1970; Jackendoff 1977; Stuurman 1985; etc.) was developed to constrain the possible phrase structure rules; it describes a uniform structure where every XP has at least a head, and possibly a specifier and/or a complement. The relative order of those three elements of the XP is not fixed, but is subject to parametric variation.

Within the minimalist framework all languages have the same phrase structure consisting of a lexical domain (VP) and a functional domain. The most general functional projections are CP (Complementizer Phrase), AgrSP (Agreement Phrase for the Subject) and AgrOP (Agreement Phrase for the Object) (see figure 1). The lexical domain is the locus of insertion of verbs and their arguments. These are inserted in fully inflected form (stem plus inflectional affixes). The functional projections are occupied by features associated with inflectional morphology. The phrase structure is built up in a bottom-up way by GT and Move-α (see section 2).

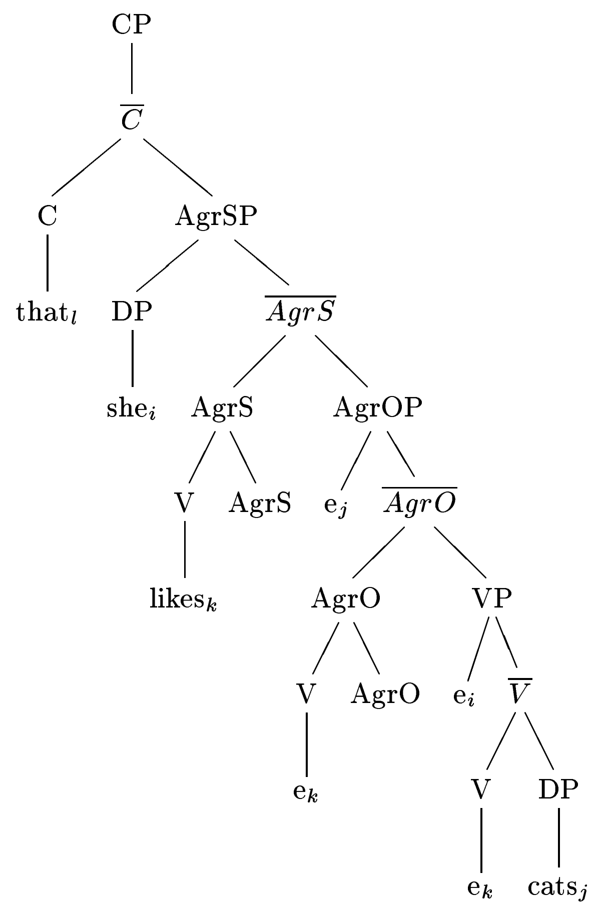

Figure 1: A simplified tree for a transitive subordinate clause in English

A parameter for the relative order of the head, specifier and complement is no longer necessary within the Minimalist Program. All languages have the same $\overline{\text{X}}$-structure with the same order, namely specifier, head, complement (see figure 1).

1.2 Spell Out and Phonetic Form

Within the minimalist approach not only phrase structure, but also possible movements are universal. Hence it follows that a certain constituent (e.g. the subject) has to cover the same path through the tree in all languages. A constituent always travels from its position of lexical insertion below in the tree, to its logical form (LF) position higher up (see figure 2).

Constituents move from point to point along a path in the tree. Phonetic form (PF) is a snapshot (using Spell Out) of the tree at one point of the derivation process (see figure 2). Since languages may differ in word order, they may differ about when Spell Out (a moment in the derivation in which instructions are given to the articulatory-perceptual system — PF) occurs.

Thus only the movements that take place before Spell Out influence PF. For this reason the part of the derivation before Spell Out is called the overt syntax. The movements between Spell Out and LF are called covert syntax. The moment of Spell Out is the only mechanism that accounts for word order differences between languages and within a single language (for different kinds of sentences such as questions an topicalized sentences). The mechanism is triggered by features (see Section 1.3).

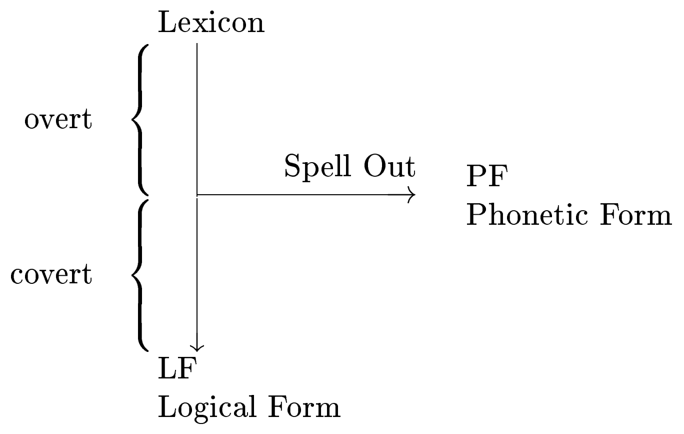

Figure 2: The derivation of a sentence

1.3 Movement and Feature Checking

All movements are caused by the necessity of feature checking. A tree contains features such as case and agreement. Movement enables the features of the moved constituent to be compared with those of the landing site. Such a comparison is called ‘checking’; after features are checked they are deleted. Features can either be strong or weak. Weak features are ‘invisible’ and strong features are ‘visible’ at PF. At PF all ‘visible’ features of the landing sites have to be deleted. That is why only strong features have to be eliminated by checking in overt syntax. Languages that differ in word order differ only in having different positions for the strong features in the tree.

Movement from the position of lexical insertion to the PF position of a constituent occurs as follows. The constituent moves to its landing site. It checks its features against the features of the landing site in the functional domain and decides whether these features are weak or strong. Movement is obligatory until a position with strong features is reached. The position with strong features is the PF position of the constituent. Every movement leaves a trace. The movements that take place between Spell Out and LF are invisible, as was mentioned in Section 1.2. This does not imply that no feature checking

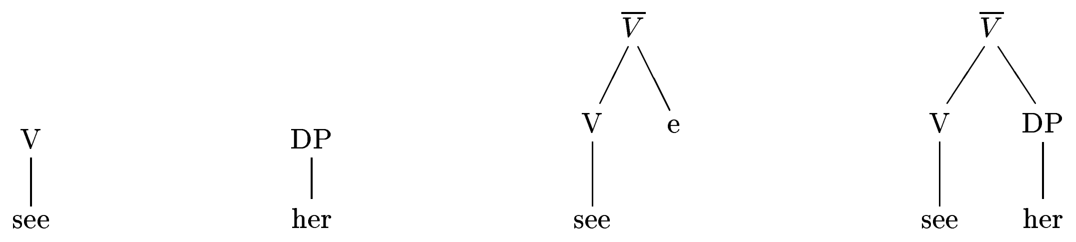

Figure 3: Generalized Transformations applied to the phrase markers V and DP yielding $\overline{\text{V}}$

takes place after Spell Out. The ending point of the path is the (universal) LF position of the constituent.

2 Generalized Transformation and Move-α

The central operations of the Minimalist Program are GT and Move-α. GT is a structure-building operation that builds trees in a bottom-up way as is illustrated in figure 3. Two phrase markers (V and DP) are combined into one. One of these two is called the target (V). A projection of the target ($\overline{\text{V}}$) is added to the target. The projection of the target has two daughters: the target itself and an empty position. The empty position is substituted for by the second phrase marker (DP). This second phrase marker is itself built up in other applications of GT and/or Move-α.

Move-α is a special kind of GT. It is an operation that combines a target with a moved phrase marker. It is assumed that movement is always leftward (Kayne 1994) and that in the universal trees of the Minimalist Program heads and specifiers, which are the only positions to move to, are always to the left of the projection line. These two assumptions in combination with the fact that GT and Move-α are bottom-up operations, effect that the moved phrase marker has to be contained in the tree that was built so far [1]. Examples of Move-α can be found in figure 1. There is a DP in the specifier position of AgrSP. To move this DP to the specifier position of AgrSP, $\overline{\text{AgrS}}$ is taken as a target phrase marker (the projection above $\overline{\text{AgrS}}$ does not exist yet at that moment). AgrSP is added as a projection of $\overline{\text{AgrS}}$ and an empty position is created as the sister of $\overline{\text{AgrS}}$. This empty position is substituted for by the phrase marker in the specifier position of the VP.

The tree in figure 1 illustrates different kinds of movement. The chain with the index *k* illustrates head movement. The verb moves from its base position in the VP to AgrO and AgrS. The verb adjoins to those heads to check its object and subject features against the features that are present there [2]. The

[1]See (Veenstra 1994) for further details.
[2]See (Chomsky 1992, page 11)

chains i and j show movement to specifier positions. The subject and the object move to the functional domain to check their features. For example, subject-verb agreement is checked in AgrSP by moving both the subject and the verb to this projection.

The lowest position of each chain is the position where lexical insertion takes place. The highest position is the LF (Logical Form) position of the element in the chain. The PF position of a chain is the position that contains a visible element instead of a trace. The PF position can be the same as the LF position or the position were the lexical insertion takes place. For example in chain i the visible element (PF) is situated in the highest position of the chain (LF).

3 Head-corner Parsing

The main idea behind head-driven parsing (Kay, 1989) is that the lexical entries functioning as heads contain valuable information for the parsing process. For example, if a verb is intransitive it will not require a complement, if it is transitive it will require a complement. Therefore the head is parsed before its sisters in a head-driven parser. A head-corner parser (Kay 1989; Bouma and Van Noord 1993) is a special type of head-driven parser. Its main characteristic is that it does not work from left to right but instead works bidirectionally. That is, first a potential head of a phrase is located and next the sisters of the head are parsed. The head can be in any position in the string and its sisters can either be to the right or to the left.

A head-corner parser starts the parsing process with a prediction step. This step is completed when a lexical head is found that is the head-corner of the goal (i.e. the type of constituent that is parsed). The head-corner relation is the reflexive and transitive closure of the head relation. A is the head of B if there is a rule with B as left hand side and A as the head daughter on the right hand side. When a (lexical) head-corner is found an $\overline{\mathrm{X}}$ rule is selected in which the (lexical) head is on the right hand side. The sisters of the head are parsed recursively. The left hand side of the rule contains the mother of the head. If this mother is a head-corner of the goal, and the mother and the goal are not equal the whole process is repeated by selecting a rule with the new head-corner (i.e. the mother of the first head-corner) on its right hand side.

In section 2 it is assumed that movement is invariably leftward and that GT and Move-α are bottom-up mechanisms. GT builds the VP before other projections. Constituents of VP are moved to higher projections by Move-α, which is a special kind of GT. Suppose that the parser should consider AgrS as the head-corner of AgrSP, which accords with $\overline{\mathrm{X}}$-Theory. Then the head (AgrS) that should be filled with an adjoined verb by movement from AgrO (in a transitive sentence) or V (in an intransitive sentence) is created before AgrO and V. To avoid moving constituents from a part of the tree that has not been built yet, the head-corner table for the minimalist head-corner parser is not constructed completely according to $\overline{\text{X}}$-Theory (see (1)).

(1)
```
hc(A̅g̅r̅S̅,AgrSP).   hc(V̅,VP).
hc(AgrOP,A̅g̅r̅S̅).   hc(V,V̅).
hc(A̅g̅r̅O̅,AgrOP).   hc(N̅,NP).
hc(VP,A̅g̅r̅O̅).      hc(N,N̅).
```

For example, instead of AgrO, VP is the head-corner of $\overline{\text{AgrO}}$. This solution is compatible with the Minimalist Program in the sense that in this way the tree is built up in an absolute bottom-up way (i.e. starting from V) so that a position that should be filled by movement is always created after the position from which the moved element comes. The head-corner table in (1) illustrates that functional heads like AgrO and AgrS are not processed as heads. Lexical projections like VP and NP are treated according to $\overline{\text{X}}$-Theory. If we follow (1) in combination with the tree in figure 1 we establish the fact that the parser searches its way down to the verb as soon as possible. The top-down prediction step moves from the goal AgrSP to $\overline{\text{AgrS}}$ to AgrOP to $\overline{\text{AgrO}}$ to VP to $\overline{\text{V}}$ and finally to the lexical head-corner V where the bottom-up process starts as the Minimalist Program requires.

4 Comparison of the Structure-building Operations of the Minimalist Program and Head-corner Parsing

If we have a closer look at the definitions of GT and Move-α we see that they resemble the strategy of head-corner parsing a lot. In both cases we start with a head. In the case of GT and Move-α this is called the target phrase marker, in the case of head-corner parsing it is called the (lexical) head-corner.

In both cases we use an $\overline{\text{X}}$ rule to obtain more information about the mother and the sister of the head. In the definitions of GT and Move-α the sister also has to be built up by GT and/or Move-α or it is an XP that is moved from a position lower in the tree to this position. In the parser that is discussed here a sister is parsed or the sister is linked to a lower position (see section 5).

The next step for GT and Move-α is to consider the new phrase marker that is built as the new target phrase marker and apply GT or Move-α to this phrase marker. The next step in the parsing process is to check if the mother of the head-corner is a head-corner of the goal. If this holds the whole process starts again with the mother as a head-corner.

5 The Algorithm

The parser that is discussed here is based on the head-corner parser in (Bouma and Van Noord 1993). Our minimalist head-corner parser mainly consists of the following 3 predicates: *parse*, *head_corner* and *predict*. The parsing process starts with calling *parse*. As we see below *parse* calls among other things the predicates *predict* and the *head_corner*.

```
% parse(Cat/CatTree,P0,P,E0,E) if there is a Cat from P0 to P,
% within the range E0, E
parse(Goal/GoalT,P0,P,E0,E) :-
  predict(Goal,Lex/LexT,Q0,Q),
  between(Q0,Q,E0,E), % Q0 and Q are between E0 and E
  head_corner(Lex/LexT,Goal/GoalT,Q0,Q,P0,P,E0,E).
```

The predicate *predict* locates a lexical head-corner. The relation *hc* implements the head-corner table.

```
% predict(Goal,Lex/LexTree,Q0,Q)
% if Lex from position Q0 to Q may be head-corner of Goal
predict(Goal,Lex/t(Lex,w(Word)),Q0,Q) :-
  hc(Lex,Goal),
  chart(Word,Q0,Q),
  lex(Word,Lex).
```

Because phrase markers in the Minimalist Program are at most binary branching, there will never be both left and right daughters in the same rule. Furthermore there is always at most one right or left daughter in each rule. It is impossible to parse both the left and the right daughters within the same *head_corner* clause. We need separate clauses to parse left and right daughters (see respectively the third and the second *head_corner* clause given below). For example, the second *head_corner* clause parses a sister of the head-corner which is to the right of the head-corner. The position of the head-corner is from Q0 to Q1. The position of the sister is from Q1 to Q, with Q <= E and Q0 <= Q1 <= Q (see parse(Dtr/DtrT,Q1,Q,Q1,E)). If Q0 = Q1 the head is empty (e.g. because it moved to another position in the tree).

The predicate *rule* implements $\overline{\text{X}}$ rules. The first argument indicates at which side of the head-corner (*Small*) the other daughter of *Mid* (*Dtr*) can be found. The third argument is the left hand side of the $\overline{\text{X}}$ rule. The second and the fourth argument represent the right hand side of the $\overline{\text{X}}$ rule.

```
head_corner(Small,Small,P0,P,P0,P,_,_).

head_corner(Small/SmallT,Goal/GoalT,Q0,Q1,P0,P,E0,E) :-
  rule(right,Small,Mid,Dtr),
  parse(Dtr/DtrT,Q1,Q,Q1,E),
  hc(Mid,Goal),
  head_corner(Mid/t(Mid,[SmallT,DtrT]),Goal/GoalT,Q0,Q,P0,P,E0,E).

head_corner(Small/Small,Goal/GoalT,Q1,Q,P0,P,E0,E) :-
  rule(left,Small,Mid,Dtr),
  parse(Dtr/DtrT,Q0,Q1,E0,Q1),
  hc(Mid,Goal),
  head_corner(Mid/t(Mid,[DtrT,SmallT]),Goal/GoalT,Q0,Q,P0,P,E0,E).
```

Furthermore we need separate clauses for GT and Move-α. As we concluded in section 4 GT has a lot in common with head-corner parsing. Therefore the plain *head_corner* clauses as given above represent GT. To account for Move-α we added movement predicates after the call to *parse* in a *head_corner* clause. The example given below is the clause that describes movement to specifier positions.

```
head_corner(Small/SmallT,Goal/GoalT,Q1,Q,P0,P,E0,E) :-
  rule(left,Small,Mid,Dtr),
  parse(Dtr/DtrT,Q0,Q1,E0,Q1),
  to_specifier_movement(MidT,_SubT,DtrT)
  hc(Mid,Goal),
  head_corner(Mid/MidT,Goal/GoalT,Q0,Q,P0,P,E0,E).
```

DtrT is the constituent that is moved to a specifier position. The root of *MidT* takes the moved constituent as its left daughter. *SubT* contains the position where the moved constituent comes from. To check if it is possible to move the constituent from *SubT* to *MidT* a simple solution is chosen. Possible movements are not determined by Economy like the Minimalist Program prescribes (Chomsky 1992; Zwart 1993). Instead a table with possible movements is consulted: move(specifier:X, specifier:Y) or move(complement:X, specifier:Y). *X* and *Y* represent respectively the category of the root of *SubT* and the category of the root of *MidT*. If a possible movement from *SubT* to *MidT* exists, the features and the chain indexes of the starting and final position of the moved constituent are unified. Head movement is treated in a way similar to movement to specifier positions.

The fact that functional heads are not head-corners causes the necessity of an unusual *head_corner* clause. The following clause is needed to be able to consider the linguistic head as a daughter and the complement as a head-corner (compare the arguments 2 to 4 of *rule* in the following clause with the same arguments within the *head_corner* clauses above). The following clause uses a rightward rule to find a daughter to the left of the head-corner. The type of rule that is used is comparable with the *head_corner* clause above that is used to parse right daughters, but the indexes for the sentence positions in this clause are the same as the indexes in the *head_corner* clause above that is used to parse left daughters. The predicate *functional* ensures that *Dtr* is a functional head. In the *head_corner* clause for movement to specifier positions, *Mid* should also be functional. Here is is not necessary to add the predicate *functional* because all possible movements are movements to functional projections. Therefore it would be redundant to prove that *Mid* is functional.

```
head_corner(Small/SmallT,Goal/GoalT,Q1,Q,P0,P,E0,E) :-
  rule(right,Dtr,Mid,Small),
  functional(Dtr),
  parse(Dtr/DtrT,Q0,Q,E0,Q1),
  hc(Mid,Goal),
  head_corner(Mid/t(Mid,[DtrT,SmallT]),Goal/GoalT,Q0,Q,P0,P,E0,E).
```

An example of the application of the *head_corner* clause that is given above is the case in which *Small* is a VP. A rule that could apply in this case is the rule where *Dtr* is AgrO and *Mid* is $\overline{\text{AgrO}}$. AgrO is the regular head of the rule, but VP is the head-corner in our parser. Therefore the rigtward rule can be applied to find a daughter that is to the left of the head-corner.

6 Parsing vs. Generation

At the end of section 3 we chose not to consider functional heads as head-corners of their mothers. This choice was made because GT starts with constructing a VP before the projections to which constituents from VP are moved are constructed. Another motivation to start with VP is that V contains information that is useful for the rest of the structure building process. For example, if the verb is intransitive we know that V does not require a complement sister, and we know that we do not need an AgrOP on top of VP. The fact that V contains lexical information and functional heads like AgrO and AgrS do not, could be used as a justification for the fact that the latter are no head-corners. The main idea of head-driven parsing is, as was stated before, that heads contain relevant information for the parsing process, and that they therefore should be parsed before their sisters. In the Minimalist Program not all heads are lexical. Functional projections do not have a lexical head. They obtain their contents via movement of elements from positions lower in the tree. This special status of functional heads makes them less useful.

The Minimalist Program is a generation-oriented framework. Because we are dealing with parsing (as opposed to generation) in this paper there are certain discrepancies between the parser and the framework it is based on. In the minimalist framework, lexical information belonging to a chain is available from the moment that the first position of the chain is created, because that is the moment when the lexicon is consulted. Lexical elements enter the tree at the bottom of their chain and the lexical information that they bring can be used during the whole length of the tree-building process. For example, a verb enters the chain in the head position of VP. Therefore the lexical information belonging to the verb can be utilized to determine whether the verb needs a complement or not. Later on in the process, the lexical information can guide the decision whether an AgrOP should be build. An AgrOP is namely only needed in a transitive sentence.

When parsing a sentence the lexicon is not by definition consulted at the beginning of the chain. Figure 1 shows a tree that contains traces and visible constituents. The position containing a visible constituent is the Spell Out position of that chain. The parser consults the lexicon at the moment in which the Spell Out position of a chain is reached. Consequently, when a trace is created before Spell Out, the features belonging to that trace are unknown. Because all positions in a chain are linked, the features of all traces of a chain are known as soon as the Spell Out position is reached. For example, in the example sentence in figure 1 we assumed that the PF position of the verb is where is is adjoined to AgrS. In that case there is a trace in V and at the moment in which it should be determined whether the verb needs a complement or not, the features of V are still unknown. Therefore the parser will have to backtrack to try different possibilities.

It can be concluded that the absolute bottom-up approach for the building of trees is more useful for generation than for parsing. When generating, lexical information can be used as soon as a position that is the beginning of a chain is created. When parsing we will have to wait until the Spell Out position is reached. In spite of this, we chose not to consider functional heads as heads in order to accomplish an absolute bottom-up process. This bottom-up approach is preferred because in this way a position to which a certain constituent is moved, is created after the position from which the constituent is moved. If we do not choose this approach, sometimes positions will be created which need a moved element from a subtree that does not exist yet. This could be inefficient and it is not a direct implementation of the ideas of the minimalist framework.

7 Conclusions and Future Work

It appeared to be possible to implement the ideas that are described here in a head-corner parser. A distinction is made between lexical and functional heads. Functional heads are not possible head-corners, while lexical heads are. The parser can judge the grammaticality of simple declarative transitive and intransitive sentences possibly with subordinate clauses. We will extend this parser in such a way that it will cover more advanced linguistic phenomena like anaphora and wh-questions.

Furthermore we will build a regular head-corner parser to be able to determine if this 'lexical'-head-corner parser is indeed more efficient than a regular head-corner parser with respect to the Minimalist Program.

8 Acknowledgements

I would like to thank Gosse Bouma, John Nerbonne, Gertjan van Noord and Jan-Wouter Zwart for their helpful comments on earlier versions of this paper.

References

Bouma, G., and Van Noord, G. (1993). Head-driven parsing for lexicalist grammars: Experimental results. In *6th Meeting of the European chapter of the Association for Computational Linguistics, Utrecht.*

Chomsky, N. (1970). Remarks on nominalization. In R. Jacobs et al, editors, *English Transformational Grammar.*

Chomsky, N. (1992). A minimalist program for linguistic theory. MIT Occasional Papers in Linguistics.

Jackendoff, R. S. (1977). *X'-Syntax: A Study of Phrase Structure.* MIT Press, Cambridge.

Kay, M. (1989). Head driven parsing. In *Proceedings of Workshop on Parsing Technologies, Pittsburg.*

Kayne, R. S. (1994). *The Antisymmetry of Syntax.* MIT Press, Cambridge.

Stuurman, F. J. (1985). *X-bar and X-plain: A Study of X-bar Theories of the Phrase Structure Component.* Foris, Dordrecht.

Veenstra, M. J. A. (1994). Towards a formalization of generalized transformation. In A. de Boer, H. d. H., and de Swart, H., editors, *Language and Cognition 4, Groningen.*

Zwart, C. J.-W. (1993). *Dutch Syntax. A Minimalist Approach.* Groningen Dissertations in Linguistics 10, University of Groningen.