Chapter 1

The Minimalist Program

In this chapter I will give a brief introduction to the Minimalist Program [Cho93]. The two implementations described in Chapter 2 are based on this version of the Minimalist Program. The formalization presented in the Chapters 4 through 9 is also based on this version of the Minimalist Program as well as on a version of the Minimalist Program that was developed by Zwart [Zwa97]. The latter version is a combination of Chomsky’s 1993 version of the Minimalist Program (cf. [Cho93]), Chomsky’s 1995 version of the Minimalist Program (cf. [Cho95]), and some new ideas of Zwart that enable the proper description of verb movement in Germanic languages such as Dutch, which show an asymmetry between main and subordinate clauses with respect to the position of the finite verb.

An introduction to Zwart’s version of the Minimalist Program is given in Chapter 3. More detailed descriptions of certain aspects of this version are given in the Chapters 4 through 9. In this chapter I will restrict myself to a global description of Chomsky’s 1993 version of the Minimalist Program [Cho93].

1.1 Introduction

The Minimalist Program [Cho93, Cho95] is a linguistic theory which originated from an almost forty year long tradition of generative linguistics. This tradition was initiated by Noam Chomsky [Cho57, Cho65].

Two of the main objectives of the generative tradition, and of many other linguistic theories nowadays, are to reach descriptive and explanatory adequacy. Descriptive adequacy is reached when the knowledge of the speakers of each natural language is explicitly described. Explanatory adequacy is reached when it can be explained how it is possible that people acquire
CHAPTER 1. THE MINIMALIST PROGRAM

language. We operationalize these by requiring a definition of ‘well-formed phrase’ (for descriptive adequacy) and a characterization of ‘possible human language’ (for explanatory adequacy).

The linguists from before the generative tradition, studying language from the point of view of traditional grammar, did not formulate their questions in such a way that descriptive adequacy could be reached according to Chomsky [Cho65, Page 5]. Only very superficial descriptions of for example, the way questions are formed were produced. Chomsky’s recognition of the complexity of language lead to the requirement that a linguistic theory must be explanatorily adequate, as in principle all people manage to learn languages in a relatively short time when they are children. Note that we are dealing with first language acquisition here.

The idea of universal grammar (UG) arose: people are born with innate knowledge about language, and they can learn any natural language they are exposed to, and possibly more than one, in the first years of their lives. This implies that languages must have a great deal in common. Since there undeniably are differences between languages we can assume that the acquisition of a language consists of the fixing of a limited number of options provided by UG.

However, descriptive interest pushed explanatory adequacy to the background. Many of the described phenomena appeared to be more intricate than one could imagine in advance, and a whole range of construction-specific and language-specific rule systems was developed.

In the Principles and Parameter approach [Cho81] the idea of explanatory adequacy was restored. It was assumed that the theory of UG consists of principles and parameters. Principles are language-independent and construction-independent laws that apply to every natural language. The differences between languages are captured by the parameters. The number of parameters and the number of possible values per parameter are finite.

The latest development in the Principles and Parameters approach is the Minimalist Program [Cho93, Cho95]. This research program is ‘minimal’ in the sense that explanatory adequacy is aimed for by ‘minimization’ of the theoretical apparatus. Every detail of the theory is reconsidered in a critical way and eliminated if it does not seem to have a legitimate reason of existence. In other words, the scientific rule of keeping a theory as simple as possible is applied.

An example of minimization of the theory is the elimination of two of the four levels of representation from the preceding versions of generative linguistic theory. The two levels of representation that are maintained in the Minimalist Program are:

- Phonetic Form (PF): an (abstract) representation of sound
1.1. INTRODUCTION

- Logical Form (LF): an (abstract) representation of meaning

The two levels of representation that are abandoned in the Minimalist Program are:

- D(EEP)-Structure: the level at which lexical insertion operates, which is defined by predicate/argument relations, and where the words of a sentence are represented in a basic (construction-independent) order.

- S(urface)-Structure: a representation of a sentence ‘as it is pronounced’, i.e. with the words ordered for instance as an active declarative clause (*He saw her*), as a question (*Who saw her?*) or as a passive construction (*She was seen*). S-Structure is defined by grammatical relations such as case. It is assumed that S-Structures can be constructed from D-Structures by moving zero or more constituents. Slightly simplifying we could say that the above example sentences were all based on the same D-Structure. For the active declarative clause no movement is involved, i.e. S-Structure equals D-Structure. In the passive construction the object *her* moves to the subject position and appears there as *she*. In the Wh-question the subject *he* moves to a special position for Wh-words and appears there as *who*.

D-Structure is eliminated in the Minimalist Program by collapsing rewrite rules, lexical insertion and movements. In the following sections we will learn more about the treatment of rewrite rules, lexical insertion and movements in the Minimalist Program. In Section 1.6 we will see how rewrite rules, lexical insertion and movements are collapsed.

S-Structure is also abandoned as an independent level of representation. It is replaced by ‘Spell-Out’ about which we will learn more later in this chapter.

What distinguishes D-Structure and S-Structure from LF and PF is that the former two are internal levels of representation while the latter two are external levels of representation. LF and PF feed into systems external to the syntactic component and therefore it is not possible to eliminate them. Since D-Structure and S-Structure do not feed into external systems, they can be eliminated.

---

1A Wh-question is a question containing an interrogative word starting with ‘wh’ in English, for example *who*. 
1.2 Word order, phrase structure and movement

Within the Minimalist Program, word order variation is derived by movement. Constituents such as verbs are moved from one position to another in the syntactic representation of a sentence.

In earlier versions of the Chomskyan theory, part of the word order differences between languages were taken care of by the theory of phrase structure: the relative order of the constituents in the right-hand side of rewrite (or phrase structure) rules was not fixed universally. The right-hand side is the part to the right of the arrow (see Example 1.1). Rewrite rules are rules that express the relative hierarchical and linear order of constituents. Illustrations of rewrite rules can be found in Example 1.1. The rewrite rule in (a) expresses that a sentence (S) consists of a noun phrase (NP), e.g. *the girl*, followed by a verb phrase (VP), e.g. *drives the car*. The verb phrase in its turn consists of a verb (e.g. *drives, sleeps*) and possibly an NP (see (b)). Note that a transitive verb such as *to drive* may occur with a noun phrase, while an intransitive verb like *to sleep* may not. This optionality of the object is expressed by the brackets around the NP in the VP-rule in (b). See Example 1.2 for a tree that can be built using the rewrite rules given in Example 1.1. As was stated above, rewrite rules, and of course the trees that are based on them, express the relative hierarchical and linear order of constituents. Note that the tree in Example 1.2 expresses, among other things, the relative hierarchical differences between S and VP, as S is located higher up in the tree than VP, and the relative linear differences between the subject NP and the object NP, as the subject NP is located to the left of the object NP.

Example 1.1

(a) \[ S \rightarrow NP \quad VP \]
(b) \[ VP \rightarrow V \quad (NP) \]

Example 1.2

\[
\begin{tikzpicture}
  \node {S} at (0,0) [circle, draw] ;
  \node {NP} at (-1,1) [circle, draw] {the girl} ;
  \node {VP} at (1,1) [circle, draw] ;
  \node {drives} at (1,0) ;
  \node {NP} at (1,-1) [circle, draw] {the car} ;
  \path[->] (S) edge (NP) ;
  \path[->] (NP) edge (VP) ;
  \path[->] (VP) edge (drives) ;
  \path[->] (VP) edge (NP) ;
\end{tikzpicture}
\]

---

2 Noun phrases nowadays often are called determiner phrases, as the determiner instead of the noun is considered to be the head of the phrase (cf. [Abn87]).
X-Theory was developed because there appeared to be many recurring patterns in phrase structure rules that asked for a generalization. X-Theory was based on proposals by Chomsky [Cho70]. Lexical properties were removed from phrase structure rules in order to obtain a simple set of universal, category-independent phrase structure rules. For instance, the specific phrase structure rules in Example 1.3 are replaced by the more general rule in Example 1.4, where X and Y are used as variables for lexical categories such as noun (N), verb (V) and preposition (P).

Example 1.3

(a) \( V \rightarrow V, NP \)
(b) \( P \rightarrow P, NP \)
(c) \( N \rightarrow N, PP \)

Example 1.4

\( X \rightarrow X, YP \)

With the introduction of X-Theory the relative linear order in the right-hand side of the rewrite rules is still not fixed universally. The relative order is subject to parametric variation between languages. For instance, in some languages the verb appears after the subject and the object (SOV-languages) and in other language the verb appears between the subject and the object (SVO-languages). As we will see in Section 1.3, Kayne [Kay94] assumes that the theory of grammar requires a version of X-Theory where the relative linear order of the constituents in the right-hand side of the rewrite rules is universal.

Within the minimalist framework, sentences in all languages have the same phrase structure consisting of a lexical domain (VP) and a functional domain. The most generally accepted functional projections are CP (Complementizer Phrase), AgrSP (Agreement Phrase for the Subject), TP (Tense Phrase) and AgrOP (Agreement Phrase for the Object) (see Example 1.5). The lexical domain is the locus of insertion of the verb and its arguments. These are inserted in fully inflected form (stem plus inflectional affixes). The functional projections are occupied by features associated with inflectional morphology. As we will see in Section 1.5, the lexical elements move to the functional domain to ‘check’ their features. Note that the indexes \( i, j, k \) and \( l \) in Example 1.5 indicate the trajectories the different lexical constituents followed. In the Sections 1.4 through 1.6 I will give a more elaborate description of minimalist trees such as the tree in Example 1.5.

\(^3\)For applications and revisions of X-Theory: cf. [Jac77, Sto81, Muy82, Stu83].
In Section 1.7 I will show how a minimalist tree is built. In that section many of the notions introduced in the Sections 1.4 through 1.6 are applied.

**Example 1.5**

In Section 1.6 we will see that rewrite rules, or rather X-rules, are applied as part of the structure-building operations *Merge* and *Move* within the minimalist framework.

### 1.3 Directionality

As was mentioned in the previous section, Kayne [Kay94] assumes in his Linear Correspondence Axiom (LCA) that the theory of grammar requires a version of X-Theory where the relative linear order of the constituents in the right-hand side of the rewrite rules is universally fixed. Example 1.6 shows the fixed relative order of the constituents within a phrase (XP) that Kayne proposes.

Kayne assumes that the left daughter of XP is always the specifier and that the head is always located to the left of the complement. Furthermore, Kayne assumes that adjuncts always appear to the left of the node they
1.3. DIRECTIONALITY

adjoin to, as we see in Example 1.7. In earlier versions of generative linguistic theory, a parameter was supposed to determine the relative linear order of specifiers, heads, and complements per language. For instance, a specifier could either appear as the right daughter or as the left daughter of XP. In the minimalist framework, the above parameter would be redundant because movement takes care of word order differences between languages, as we saw in the previous section. We do not need a second mechanism to derive word order differences since all possible orders can be derived with the help of movement. From the minimalist point of view, redundancy is a valid reason to reconsider legitimacy of a mechanism of the theory.

Example 1.6

Example 1.7

Summarizing, we can say that Kayne draws the conclusion that the parameter determining the relative linear order of specifiers, adjuncts, heads and complements is superfluous. Therefore Kayne looks for reasons for a specific fixed order.

Kayne deduces the relative linear ordering of the specifier, adjunct, head and complement of an XP from the assumption that the linear ordering of the terminals of a tree is determined by the hierarchy of a tree. Before we will have a closer look at this assumption, I need to explain two important notions: `dominate' and `\(c\)-command'.

A definition of the notion `dominate' is given in Definition 1.1. On the basis of this definition we can conclude that in Example 1.8 XP dominates YP, but also ZP.
Definition 1.1
A node $\alpha$ dominates a node $\beta$ if and only if $\alpha$ is the mother of $\beta$ or if there exists a node $\gamma$ of which $\alpha$ is the mother and that dominates $\beta$.

The notion ‘c-command’ is defined in Definition 1.2 (see also [Rei81] and [Cho86b, Page 8]). On the basis of this definition we can conclude that in Example 1.8 XP c-commands ZP, but not that ZP c-commands XP.

Definition 1.2
A node $\alpha$ c-commands a node $\beta$ if and only if $\alpha$ does not dominate $\beta$, and every node $\gamma$ that dominates $\alpha$ also dominates $\beta$.

Having discussed the notions ‘dominate’ and ‘c-command’, we can return to Kayne’s claim that the the linear ordering of the terminals of a tree is determined by the hierarchy of a tree. Kayne supposes that if a node $\alpha$ of a tree asymmetrically c-commands another node $\beta$ of that same tree, than the terminal (lexical) elements dominated by $\alpha$ linearly precede the terminal (lexical) elements dominated by $\beta$. By ‘asymmetrical c-command’ is meant that two nodes may not c-command each other.

Example 1.8

Kayne needs a more restrictive definition of c-command than Definition 1.2, since according to this definition in Example 1.8 XP c-commands X, while at the same time XP c-commands Y. It is crucial that the c-command relation between two nodes $\alpha$ and $\beta$ is asymmetrical, as it is impossible that $\alpha$ precedes $\beta$ while at the same time $\beta$ precedes $\alpha$. However, with the current definition of c-command this is not the case. Therefore Kayne applies a modified definition of c-command, as we will see below.

In Kayne’s new definition $\alpha$ c-commands $\beta$ if and only if:

- $\alpha$ and $\beta$ are not segments (see below),
- $\alpha$ excludes $\beta$ (see below), and

---

4Note that Kayne restricts himself to binary branching trees [Kay94, Page 4].

5Note that in Example 1.8 the intermediate bar-level (e.g., $\overline{X}$) does not occur. In Chapter 7 we will argue why we do not follow this approach to $X$-Theory.
1.4. LEXICAL INSERTION, SPELL-OUT, AND LOGICAL FORM

- every \( \gamma \) that dominates \( \alpha \) also dominates \( \beta \).

A node \( \alpha \) is a segment if and only if it is not a head or a direct projection of the head. For instance, in Example 1.8 \( X \) is the head, the lower XP is the direct projection of the head, and the higher XP is a segment. A node \( \alpha \) excludes a node \( \beta \) if and only if no segment of \( \alpha \) dominates \( \beta \).

In Example 1.8 the higher XP does not c-command \( Y \) since XP is a segment. The lower XP does not c-command \( Y \) either, since the lower XP does not exclude \( Y \).

\( Y \) does c-command \( X \) in Example 1.8 since it is not a segment, it excludes \( X \) and every \( \gamma \) that dominates \( Y \) also dominates \( Y \) also dominates \( X \).

Now, we can conclude that the structure in Example 1.9, where the complement (\( ZP \)) precedes the head (\( X \)) and the specifier (\( YP \)), is incorrect. For instance, \( YP \) c-commands \( X \), but \( Y \) does not linearly precede \( X \).

Example 1.9

\[
\begin{array}{c}
XP \\
/\beta \phan{\text{YP}} \\
\text{XP} \\
ZP X Y
\end{array}
\]

Hence, from the assumptions given by Kayne, we deduce that in a phrase the head is preceded by the specifier and followed by the complement. In the next section we will see how this influences the direction of movements in the Minimalist Program.

1.4 Lexical insertion, Spell-Out, and Logical Form

Within the minimalist approach not only phrase structure, but also possible movements are universal. Hence it follows that a given 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 low in the tree, to its Logical Form (LF) position higher up (see Figure 1.1 on Page 13). For instance, in Example 1.5 (repeated here as 1.10) we see that the subject \( she \) (with index \( i \)) is inserted in the lexical domain (VP), and moves to the functional projection AgrSP via TP. The LF-position is the highest position a constituent reaches, and therefore the LF-position of \( she \) is within AgrSP. The verb \( hates \) (with index \( k \)) moves considerably more often than
the subject. It moves from VP to AgrOP to TP to AgrSP, which also is the LF-position of the verb. As we will see in Section 1.5, a lexical element may only move to a given functional projection if it needs to check one or more features there.

**Example 1.10**

Somewhere in between the position of lexical insertion and the LF-position of a constituent, we find the position where the constituent is 'pronounced'. In Example 1.10 the ‘pronunciation position’ is marked because this is the only position in the chain of movements where the constituent is actually written down. The other positions are just marked by the letter ‘e’ (for empty) with an index. Note that the pronunciation position may very well coincide with the position of lexical insertion or the LF-position, or even with both as it is the case with the complementizer that in Example 1.10, which does not move at all.

The pronunciation position is determined by the optional rule of Spell-Out. Therefore we will replace the term pronunciation position with the term ‘Spell-Out position’. Since languages may differ in word order, they may differ as to the point in the derivation where Spell-Out applies. In
Section 1.5 we will see how the Spell-Out position of a given constituent in a given language is determined.

In Figure 1.1 we see that the movements that take place before Spell-Out are called overt movements, while the movements after Spell-Out are called covert movements. This terminology is used because only the movements that take place before Spell-Out influence the sentence as we perceive it. In Section 1.5 I will go more deeply into the reasons for assuming covert movement.

![Diagram](image-url)

**Figure 1.1:** The derivation of a sentence

Summarizing, we can say that the lexicon is consulted in the lowest position of a chain, the Spell-Out position is higher than and to the left of the position of lexical insertion, and the LF-position is higher than and to the left of the Spell-Out position. Note that it might be the case that two or more of these three positions coincide.

The fact that movement is always directed towards the left is deduced from Kayne’s ideas as described in Section 1.3. We assumed there that specifiers and heads are located to the left of complements. Since movement is only possible when it is aimed towards head and specifier positions, movement is always leftward, as we will see in Section 2.1.

The fact that movement is always upward can be deduced from the fact that derivation trees are built up in a bottom up way, as we will see in Section 1.6. Constituents move from the lexical domain (VP) at the bottom of the derivation tree to the functional domain (AgrOP, TP etc.) higher up in the derivation tree. Since moving constituents must have features that can be checked against features in the functional domain (see earlier in this section and Section 1.5), it is not possible to move nodes without lexical
Therefore, the position of lexical insertion is by definition lower than (or equal to) the Spell-Out position and the LF-position.

1.5 Movement and feature checking

In the minimalist framework all movements are caused by feature checking requirements. The functional projections (AgrSP, TP etc.) as well as the lexical projections in a derivation tree, contain features such as case and agreement. The features of lexical constituents consist of three different types: formal (also syntactic or inflectional) features, semantic features, and phonological features [Cho95, Page 229ff]. Semantic and phonological features are considered to be relevant at respectively the interfaces LF and PF. Formal features are the features that cause the movements within the derivation. Movement enables the features of the moved lexical constituent to be compared with those of the landing site in a functional projection. Such a comparison is called feature checking. For instance, as we see in Example 1.10, the subject NP moves to AgrSP to check its agreement features and the verb moves to TP to check its tense features.

After features of a lexical constituent have been checked against those of a functional head, they are deleted. Below we will see why the features are deleted from the functional projections. Furthermore, we will see the difference between overt and covert movement.

Both the deletion of features and the difference between overt and covert movement are connected with the principle of Full Interpretation. The principle of Full Interpretation requires that the interface representations of a sentence consist entirely of legitimate objects (Economy of Representation). Which objects are legitimate at the interface levels LF and PF will become clear later on in this section.

In Figure 1.1 we see that Spell-Out determines the point in the derivation (from lexical insertion to LF) where instructions are given to the interface level PF. This follows directly from the definition of Spell-Out because the Spell-Out position is the position in a derivation where a constituent is as we perceive a sentence. Of course we want the constituents of a sentence to enter the interface level PF in the order in which they are pronounced so that we will perceive them in the correct order. The instructions that are given to PF consist of the features of the constituents in Spell-Out positions. The PF-representation of a sentence is the sentence as we hear it. This representation is based on the phonological features of the constituents in the Spell-Out positions (cf. [Cho95, Page 230]). Since heads enter the

---

6Note that the division in three types of features is first introduced in Chomsky's 1995 framework. Although this chapter is an introduction to Chomsky's 1993 framework I will introduce the division here for the sake of clarity.
1.5. MOVEMENT AND FEATURE CHECKING

derivation in fully inflected form (cf. [Cho93, Page 27ff]), PF does not play a role in determining the morphology of words.

The features of the functional projections can either be strong or weak according to the Minimalist Program. Weak features are ‘invisible’ and strong features would be ‘visible’ at the interface level PF. But then features, being purely syntactic/formal features, cannot be interpreted at PF. Therefore, all visible features have to be deleted before Spell-Out, since syntactic/formal features are illegitimate objects at PF. The principle of Full Interpretation namely causes a derivation of a sentence to fail or ‘crash’ when one of the interface representations contains illegitimate objects, and deletion of checked formal features can prevent the derivation from crashing. Since weak features are invisible at PF only strong features have to be eliminated after they are checked in overt syntax.

Languages that differ in word order differ only in having different strong and weak features. The strong/weak parameter is even supposed to be the only parameter in the Minimalist Program. This would make the Minimalist Program an explanatorily adequate theory, since it explains why children can acquire language in a relatively short time.

The fact that the strong/weak parameter is the only parameter, in combination with the fact that all movements are universal in the Minimalist Program, leads to the conclusion that all lexical items must contain the same features in all languages, because it is not possible that functional heads contain different features in different languages (since this would imply another parameter). Of course, it is possible that very similar lexical items have different values for the same feature in different languages. For instance, a certain noun can be masculine in one language and feminine in the other.

There is a chance that a given constituent lands at more than one position with different strong features as it travels from its position of lexical insertion to its LF-position. In such cases the highest position with strong features is the Spell-Out position, since Full Interpretation requires that all strong features are removed before PF.

At the interface level LF not only strong but also weak formal features are visible. As the derivation crashes if any visible formal features are left at LF, the constituents of a sentence have to keep on moving until all inflectional features are deleted. Hence, the LF-position of a given constituent in a given construction is universal. The LF-order of the constituents represents the meaning of the sentence. Legitimate objects at LF are predicates, arguments, modifiers and operator-variable constructions (cf. [Cho93, Page 27]) but also semantic features such as [artifact] (cf. [Cho95, Page230]). Note that movement of a lexical constituent to a functional projection is only possible if the features of the moving element and its landing site
match. If one or more features of a given lexical constituent do not match with the features of any of the landing sites (i.e. node within the functional domain) of the relevant phrase structure, the derivation of that sentence crashes.

Summarizing, we could say that a tree is an LF-representation if all lexical constituents in the tree have checked all their formal features. Hence, covert movement is needed because generally by the time of Spell-Out not all formal features are actually checked.

Movement from the position of lexical insertion to the Spell-Out position of a constituent occurs as follows. The (lexical) constituent moves to a landing site (in the functional domain). It checks its features against the features of the landing site. Movement is called overt movement until the highest position with strong features is reached. This highest position with strong features is the PF-position of the constituent. The movements that take place between Spell-Out and LF are covert, as was mentioned in Section 1.4. This does not imply that no feature checking takes place after Spell-Out, only that no checking of strong features remains. The ending point of the path is the (universal) LF-position of the constituent.

A derivation of a sentence converges only when it does not crash at PF, nor at LF. To avoid having the derivation crash, formal features have to be deleted by movement of lexical constituents in the course of the derivation. We could say that the point in the derivation where Spell-Out applies is deducible from a requirement imposed on PF and LF. This requirement is called Full Interpretation and it implies that, as we saw above, no uninterpretable object may emerge at the interface levels PF and LF. Since Spell-Out is deducible from PF and LF, it can replace the independent interface level S-Structure from earlier versions of generative linguistic theory, as I already mentioned in Section 1.1. The difference between overt and covert movement that I mentioned at the beginning of this section is caused by the fact that PF and LF differ as to which objects are interpretable.

1.6 The operations Merge and Move

The central operations of the Minimalist Program are Merge and Move.7

Merge is a structure-building operation that builds trees in a bottom-up way as is illustrated in Example 1.11. Two trees (V and NP) are combined into one. One of these two is called the target (V). A projection of the target (\(\bar{V}\)) is added to the target. The projection of the target has two

---

7Note that Move and Merge are called respectively the singular and the binary version of Generalized Transformation in [Ch69]. The notions Move and Merge are introduced by Chomsky [Ch65]. I use the notions Move and Merge instead of the old terms, because the former are the more popular at the moment.
daughters: the target itself and an empty position. The empty position is
substituted for by the second tree (NP). This second tree is itself built up
in other applications of Merge and/or Move. The target in Example 1.11 is
obtained directly from the lexicon. It is not a constituent that is the result
of earlier applications of the operation Merge and/or Move.

\begin{example}
\begin{center}
\begin{tikzpicture}
  \node[const] {\text{see}}
    child {node[const] {\text{her}}};
  \node[const] {\text{see}};
  \node[const] {\text{her}};
\end{tikzpicture}
\end{center}
\end{example}

Move is an operation that moves a tree within a tree. The operation
combines a target with a moved tree. It is assumed that movement is
always leftward and that heads and specifiers, which are the only positions
to move to, are always to the left in the tree (see Section 1.4). These two
assumptions in combination with the fact that Merge and Move are bottom-
up operations, effect that the moved tree has to be contained in the tree
that was built so far. Chomsky [Cho93, Page 23] argues that the moved
tree must be contained in the target.\footnote{See also Subsection 2.1.3.}
Illustrations of the operation Move
can be found in Example 1.5, repeated here as Example 1.12. There is a NP
in the specifier position of AgrSP. To move this NP to the specifier position
of AgrSP, AgrS is taken as a target tree (the projection above AgrS does
not exist yet at that moment). AgrSP is added as a projection of AgrS and
an empty position is created as the sister of AgrS. This empty position is
substituted for by the NP from the specifier position of the VP.
The tree in Example 1.12 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, T and AgrS. The verb adjoins
to those heads to check its features against the features that are present
there.\footnote{See [Cho93, Page 11].}

The 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 AgrSP.

Now I will return to the elimination of D-Structure, which I mentioned
earlier in Section 1.1. I claimed there that in the Minimalist Program, D-
Structure can be eliminated by collapsing rewrite rules, lexical insertion and
movements. Here we will see that the structure-building operations Move
and Merge open up possibilities to eliminate D-Structure. In earlier versions
of generative grammar, D-Structure was a representation that was based on
rewrite rules and lexical insertion. A D-Structure representation of a sentence was built by deducing a structure from the rewrite rules and filling it with items from the lexicon. By applying movements to D-Structure, S-Structure was derived. In the Minimalist Program movements, rewrite rules and lexical insertion are combined in the structure-building operations Move and Merge. Hence, in earlier versions of generative grammar one first applied all rewrite rules and then all transformations, while in the Minimalist Program rewrite rules intermingle in application with the transformational (Move) rules.

Example 1.12

Firstly, that movement is now a part of the structure-building operations is self-evident. Secondly, rewrite rules, or rather X-rules are consulted by the operations Merge and Move in the Minimalist Program. The structures that are built must be correct according to X-Theory. Thirdly, lexical items are not introduced ‘all at once’, to speak in Chomsky’s terms, in the Minimalist Program, as was done at D-Structure in earlier versions of generative linguistics. In the Minimalist Program, derivation trees of sentences are built in a bottom-up way by applying structure-building operations. Lexical items first are introduced when the position in the tree where they
belong is created. The ‘all-at-once’ representation of lexical material was the reason of existence of D-Structure.

The fact that movements, rewrite rules and lexical insertion are combined in the structure-building operations Merge and Move blurs the difference between D- and S-Structure and makes it plausible that D-Structure can be eliminated.\textsuperscript{10}

1.7 Summary

I will summarize the above by describing the derivation of the correct structure for the sentence \textit{She hates cats} more or less from start to finish. The only part I will skip is the derivation of the internal structure of the NPs \textit{she} and \textit{cats}.

The derivation starts with the introduction of the verb \textit{hates} from the lexicon (lexical insertion). This verb serves as the target at this point. The V \textit{hates} projects to V and selects a complement \textit{cats} (see Example 1.13).\textsuperscript{11} The fact that V projects to V and the fact that the complement occurs left of the head is deduced from \textit{X-Theory}. Hence, in the step described here the V \textit{hates} and the NP \textit{cats} are merged into a V.\textsuperscript{12}

Example 1.13

\begin{center}
\begin{tikzpicture}
  \node (vp) {V
    \node (v) {hates
      \node (c) {cats
        \node (n) {she}
}}
  \node (np) {NP
    \node (n) {she
      \node (n) {cats}}}
  \edge from (v) to (vp)
  \edge from (v) to (c)
  \end{tikzpicture}
\end{center}

In the next step \textit{\textbf{v}} is the target. \textit{\textbf{v}} is merged with the NP \textit{she} by projecting it to VP and selecting the NP \textit{she} as its specifier sister (see Example 1.14). Again, \textit{X-Theory} is consulted to determine that VP is the projection of the target and to determine the relative order of the specifier and the target.\textsuperscript{13}

\begin{footnotesize}
\begin{itemize}
  \item The all-at-once nature of D-Structure has often been considered problematic in recent research. See for instance: [Kl85, Lu88, Le88, Bro93].
  \item Normally, as a first step, \textit{hates} receives an empty sister, which in a second step is filled with the complement \textit{cats}. This intermediate step is left out for the sake of simplicity.
  \item The complement \textit{cats} itself is built up by applications of lexical insertion, \textit{X-Theory}, Merge and Move that are not described here.
  \item The specifier \textit{she} is built up by applications of lexical insertion, \textit{X-Theory}, Merge and Move that are not described here.
\end{itemize}
\end{footnotesize}
Example 1.14

The next step, which actually consists of two separate steps, is a Move operation. The V *hates* which is contained in the VP moves to AgrO by adjoining to it (see Example 1.15).\(^\text{14}\) AgrO is the target of the operation and merges with VP. To enable movement of V to AgrO, AgrO is disconnected. Next it is linked with the rest of the structure again via its projection (AgrO). Then it receives V as its sister by movement from within VP.\(^\text{15}\)

As we saw earlier in this chapter, movement must be associated with feature checking. In this case the object agreement and the object case features of V are checked against the agreement and case features of AgrO.\(^\text{16}\) Of course, also in this step and in all the following steps X-Theory is consulted.

Example 1.15

\(^{14}\) See Subsection 2.1.3 for a definition of the domain where a moving tree must come from.

\(^{15}\) Cf. [Cho95, Page 200].

\(^{16}\) In Chapter 5 the features that are applied in the Minimalist Program are described in detail.
In the next step AgrO is the target. It projects to AgrOP and selects a specifier (see Example 1.16). Since this step is a Move operation the specifier must be a subtree contained in the tree built so far. The specifier that is selected is a ‘copy’ of the complement of V, i.e. the NP *cats*. Note that the specifier of AgrOP contains an empty copy. This is caused by the fact that the features of the NP *cats* are not checked against strong features and hence the Spell-Out position of the NP *cats* is in situ. The case and agreement features of the NP are checked against the case and agreement features of AgrO.

**Example 1.16**

Subsequently, verb movement takes place again. This time by adjoining AgrO, including V, to T (see Example 1.17). The tense feature of V is checked against the tense feature of T.

Next, I assume that the specifier of VP moves to the specifier of TP to check its case feature against the case feature of T (see Example 1.18). Chomsky assumes that there is no movement to the specifier of TP. Instead he assumes that the case feature of the subject (*she* in our example) is checked against the case feature of T within AgrSP. This is possible since T Adjoins to AgrS, as we will see in the next step of this derivation. However, for reasons that are explained in Section 8.2, I will assume that the subject moves to the specifier of TP.

---

17The reason why not only V moves, but rather AgrO including V lies in the fact that sometimes lexical constituents check their features against a functional projection α by moving to another functional projection β, to which the head of α is adjoined. For instance, according to Chomsky [Cho95, Page 174] the case feature of the subject NP is checked against T in the specifier position of AgrSP.
Example 1.17

The last two steps of the derivation are summarized in Example 1.19. V reaches its Spell-Out position in AgrS. It checks its subject agreement feature against the strong agreement feature of AgrS. Likewise, the subject NP *she* reaches its Spell-Out position by checking its agreement feature against the strong agreement feature of AgrS.
Example 1.19