Controlling Movement:
Minimalism in a deductive perspective

Willemijn Vermaat

April 1999

Thesis submitted to obtain the degree of *doctorandus* in
Cognitieve Kunstmatige Intelligentie at Utrecht University

Supervisor: Michael Moortgat

Acknowledgments

When I started writing my thesis last year, I didn't foresee that writing a thesis takes so much time, effort, but also pleasure. For this last reason I want to thank a number of people: first of all my supervisor Michael Moortgat for his constant encouragement, help and enthusiasm through the whole process of research and writing; Tom Cornell, Alain Lecomte and Christian Retoré, because their articles and ideas on 'Minimalism in a deductive perspective' gave me confidence to study this topic further; my second and third supervisor, Vincent van Oostrom and Ed Stabler for their careful reading of my thesis; Iris Mulders for helping me out with the chapter on the Minimalist Program; Brenda Kennelly for checking my English grammar and style; all my friends at the Uil-OTS: Anne-Marie, Dirk, Esther, Paola, Patrick, Richard and Tamara, for the conversations during lunch, coffee and tea breaks; but especially Raffaella for all the laughs during good times and comfort during times of depression.

Then last, but not least, I want to thank my parents, Leentje and Pim, for stimulating me to continue studying; Gert Jan for supporting me and always being there when I needed him; my brother, Joep, for his emails on cultural events showing me that there's more to life than studying.

And reaching the end of my education, I also want to thank the teachers, the bureau and my fellow students of CKI for establishing a good environment for study, discussion, pleasure and the freedom to develop your own ideas on science and education.

Contents

Chapter 1

Introduction

1.1 Topic of thesis

In this thesis the minimalist theory of Chomsky (1995) is studied from the perspective of Multimodal Categorial Grammar (Moortgat, 1996, MMCG). In particular we want to formulate an answer to the following question.

> How are certain aspects of movement as described by the Minimalist Program captured in the MMCG framework?

To tackle this problem, we first need to describe the basic components of the two frameworks. Therefore the thesis starts with an overview of the two theories that are at the center: *Multimodal Categorial Grammar* and *the Minimalist Program*.

The main goal is to give a deductive account of the operation MOVE. As the Minimalist Program leaves a formalization of the MOVE operation, we use another framework that gives a formal description of the minimalist MOVE operation. The formalism of Minimalist Grammar (Stabler, 1999, MG) captures the basic components of the minimalist framework. Using Stabler's formalism we can relate the multimodal framework to the basic operations and principles of minimalism.

1.2 Hypothesis

In the Minimalist Program, the basic operations in the Computation System are: MERGE and MOVE. The MERGE operation is a structure building operation and is given by the rules of application as defined in the base logic of MMCG. MOVE defines the displacement of a phrase in a sentence. The first step in the multimodal approach is to translate movement as a structural operation to capture the phenomenon of displacement. But then one neglects the derivational meaning of MOVE as an abstraction operation. The second approach regards MOVE as a complex operation, which can be decomposed in a logical and a structural part. On the logical side, the concept of hypothetical reasoning provides a principled account of the abstraction of a phrase. On the structural side, the structural postulates capture the actual movement of features and phrases in a structure.

Movement in the Minimalist Program is led by the need to check uninterpretable features on functional categories. The phrase carrying the matching interpretable feature moves to the checking domain of the functional category to check the uninterpretable feature. MMCG specifies different control features, which are lexically anchored on the lexical items that need to be displaced. The control features trigger the application of the structural postulates. The assignment of higher order types to displaced words invokes hypothetical reasoning, the logical component of MOVE as an abstraction operation.

1.3 Background

1.3.1 Relation with Cognitive AI

This research is done as part of the *Computational Linguistics and Logic* component of the Cognitive Artificial Intelligence program. Computational linguistics can be studied from a cognitive or a technological perspective. The two linguistic theories that are compared in this thesis make technological and cognitive claims. On the one hand MMCG has attractive proof theoretical properties allowing us to use a theorem prover Moot (1996, Grail) to quickly process analysis. On the other hand Minimalism tries to find an explanation for the *Computational System* of human language, thus contributing to our understanding of a central cognitive ability.

1.3.2 Scientific account

The two theories are the product of two distinct scientific traditions, but they both aim to give an explanation for grammatical knowledge and the use of that knowledge. The two theories show similarities in their explanation of linguistic phenomena. For that reason an integration of the two theories should be researched. With my thesis I want to start to make such an integration by showing some similarities between their analyses of a 'Computational System of Human Language' (Chomsky, 1995).

Chomsky (1995) concentrates on the description of the basic components of the Minimalist Program. The two main components are the *Lexicon*, which contains all the necessary feature information of words, and the *Computational System*, which operates on the lexical elements to form phrasal structures. As the Minimalist Program lacks a formalization, Stabler (1999) presents a formal framework to capture the main components of the Minimalist Program. Stabler's Minimalist Grammar (= MG) defines lexical feature specifications and tree structures to formulate grammars for natural language. Such a grammar contains a lexicon, which serves as storage for features, and basic structure building operations such as MERGE and MOVE.

Opposite to the minimalist framework, the logical deductive theory of Multimodal Categorial Grammar (= MMCG) lays down a formal theory of natural language. MMCG is a proof system consisting of a logical, and a structural part. A multimodal grammar lays down a lexicon, which contains logical formulas assigned to words, and operations to construct structures. The operations split up into logical inferences defined by the base logic and structural inferences captured by structural postulates.

The differences between the two theories are the result of a difference in bias. One theory, Minimalism, tries to define language with universal principles (*top down*), the other, MMCG, tries to find a logical foundation on which to build the rules and principles used in a language (*bottom up*). I want to show that in spite of this difference in bias, both theories try to capture the same Computational System.

An integration would be an improvement of both theories. The proof-theoretical approach of Moortgat (1996), also described in Morrill (1994), receives more linguistic relevance; a more linguistic approach results in a broader environment to test the theory. The minimalist approach of Chomsky (1995) receives a logical foundation where the operations can be stated in terms of logical and structural rules.

1.4 Thesis Overview

The thesis is laid out as follows. In this introduction the scientific and AI-related position of the thesis are presented. Chapter 2 describes the framework of Multimodal Categorial Grammar. After a small historic overview in section 2.1, section 2.2 starts with an elaboration of the base logic of Lambek calculus, followed by section 2.3 on the structural aspects of a derivation. The multimodal framework is further developed in section 2.4 with an explanation of structural control. Section 2.5 shows how we can capture the derivational meaning by decorating the logical rules with terms.

The Minimalist Program is described in chapter 3. As the theory is still under development the basic components of the Minimalist Program are sketched. In section 3.2 the lexicon is elaborated, giving the basic ideas of its content. The Computational System within the Minimalist Program takes a central position in this chapter. Section 3.3 introduces the basic operations of the Computational System and continues with a comprehensive explanation of the operation MOVE and of the mechanism that controls the operation. In section 3.5 we present a sample derivation within the Minimalist framework.

The main chapter is chapter 4, where we focus on *movement* from a deductive perspective. As the Minimalist Program as such is not a formalization, a comparison is hard to make. Section 4.1 starts with an explanation of Stabler's algebraic translation of the minimalist framework: *Minimalist Grammar*. A mapping from Stabler's Minimalist Grammar to MMCG in section 4.2 results in a deductive approach of the different minimalist operations. A deductive analysis of MOVE on the basis of its derivational meaning leads to the right translation of MOVE as as an abstraction operation. As an illustration, a Hungarian phenomenon of verbal complexes is analyzed in the multimodal framework in section 4.3. This leads to a discussion of the possibilities MMCG has to offer for the research of a Universal Grammar.

In the conclusion I will present an overview of the different aspects that this study of the minimalist framework from a deductive perspective brings up. I formulate an answer to the question: How are certain aspects of minimalist movement described in the multimodal framework? Furthermore I make some suggestions how to continue the research for a minimal theory of natural language.

Chapter 2

Multimodal Categorial Grammar

This chapter describes the multimodal framework starting with a brief history in section 2.1. MMCG consists of a logical and a structural part. Section 2.2 describes the logical part by introducing the base logic of Lambek calculus and is followed by section 2.3 explaining the structural part of a derivation. This section presents the concepts that form the structural landscape. Section 2.4 explains the multimodal part of the categorial framework, the need for multiple composition operations and modalities to structurally control them. Section 2.5 elaborates the meaning of derivations in terms of term decorations. The last section ends with a summary of the concepts described in this chapter.

2.1 History

The logical framework of Categorial Grammar goes back to the work of Bar-Hillel (1964) and Lambek (1958) with their logical analysis of linguistics. Both logicians based their work on the studies by other philosophers and logicians who tried to capture the type structure of natural language. After the initial analysis, the framework was further developed by the work of linguists, logicians, philosophers and mathematicians. One of these directions extends the basic Lambek Calculus to Multimodal Categorial Grammar.

2.2 Base Logic

2.2.1 Grammatical composition

In theories of grammar one concept always appears in analyzing sentences: *composition* of words, phrases or sentences. Words merge into phrases that combine with other phrases to form sentences. Defining a theory of grammar, one needs to address this concept of composition. Categorial Grammar analyzes words within their larger context, starting from structures that are taken to be "complete". By decomposing these "complete" phrases into their parts, one can view words as dependent on other words. These contextual dependencies are defined within a grammar.

Take an example: a noun phrase is decomposed into a determiner and a noun. Reversely the determiner depends on the noun to form a noun phrase. Lambek stated that this dependency or grammatical incompleteness can be captured with an 'implication' connective ($A \rightarrow B$: if A, then B). Moreover, the determiner needs to combine with a noun on its right side to account for directionality. Categorial Grammar uses two connectives $\{/, \backslash\}$ to indicate directional dependencies between words. The connective $\bullet$ defines the assembly of two words. The behavior of the logical connectives is given by the residuation laws. These laws show how the incompleteness of $/, \backslash$ relate to the composition of $\bullet$.

$$A \vdash C/B \iff A \bullet B \vdash C \iff B \vdash A \backslash C$$

On the basis of these binary connectives, type formulas are formed. The types are assigned to words to indicate their role within sentences. Some words and phrases play an essential role within sentences; these words are taken to be "complete". The formulas assigned to "complete" words or phrases are basic types: s for sentences, n for common nouns and np for noun phrases. In the case of more complex sentences the set of basic types is extended.

The following grammar defines the logical language of formulas $\mathcal{F}$, where $\mathcal{A}$ is the set of atomic formulas, for example the basic types: $\{s, n, np\}$.

$$\mathcal{F} ::= \mathcal{A} \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F} \bullet \mathcal{F} \mid \mathcal{F} \backslash \mathcal{F}$$

As an example of the way formulas are built on the basis of this formula language, we define the lexical entries in Fig. 2.1.

$$\begin{array}{l} \mathsf{Maria} \vdash np \\ \mathsf{makes} \vdash (np \backslash s)/np \\ \mathsf{the} \vdash np/n \\ \mathsf{tortillas} \vdash n \end{array}$$

Figure 2.1: Example of a lexicon

Maria and tortillas are assigned basic categories, while the determiner the requires a common noun such as tortillas to form a noun phrase. The transitive verb makes shows the use of directionality; first it needs to combine with a noun phrase on the right side, then it combines with a noun phrase on the left side to form a sentence.

2.2.2 Rules of application

The residual nature of the binary connectives is captured by logical rules. Different proof systems are used to define these rules. Two of these proof systems are regularly used within the categorial tradition: *Gentzen calculus* for a better proof search and *Natural Deduction* for a nicer proof display. The latter proof system will be used throughout this thesis because our main goal is to use the logical framework in linguistic applications.

Derivations in natural deduction are presented in a Prawitz style. We reason about expressions of the form: $\Gamma \vdash A$, asserting that a certain *structure* Γ has type A.

Structures $\mathcal{S}$ are built from formulas $\mathcal{F}$ with the following grammar rules:

$$\mathcal{S} ::= \mathcal{F} \mid (\mathcal{S} \circ \mathcal{S})$$

The structural operator $\circ$ combines the type-logical formulas, presenting the structure as a binary tree with $\circ$ as the nodes and the formulas as the leaves. The structure of the string is built during the application of the logical rules that define grammatical composition.

Lambek (1958) shows that the basic laws of grammatical composition are given in the form of Modus Ponens inferences: $A/B \circ B \vdash A$ and $B \circ B\backslash A \vdash A$. The construction of an expression A/B (or B) and an expression B (or $B\backslash A$) results in a composite structure of the two expressions typed by A. In the natural deduction proof style these Modus Ponens rules extend to the elimination rules of the connectives $\{/, \backslash\}$, as given in Fig. 2.2.

$$\frac{\Gamma \vdash A/B \quad \Delta \vdash B}{\Gamma \circ \Delta \vdash A}\ [/E] \qquad \frac{\Gamma \vdash B \quad \Delta \vdash B\backslash A}{\Gamma \circ \Delta \vdash A}\ [\backslash E]$$

Figure 2.2: Elimination rules for $/, \backslash$

As an example of a well-formed expression we derive the sentence "Maria makes the tortillas" of category s using the elimination rules in Fig. 2.2 and the lexical entries in Fig. 2.1. For clarity, we write the labels of the lexical expressions instead of the formulas to represent the structural side of the derivation in Fig. 2.3.

$$\dfrac{\mathsf{Maria} \vdash np \qquad \dfrac{\mathsf{makes} \vdash (np\backslash s)/np \qquad \dfrac{\mathsf{the} \vdash np/n \quad \mathsf{tortillas} \vdash n}{\mathsf{the} \circ \mathsf{tortillas} \vdash np}\ [/E]}{\mathsf{makes} \circ (\mathsf{the} \circ \mathsf{tortillas}) \vdash np\backslash s}\ [/E]}{\mathsf{Maria} \circ (\mathsf{makes} \circ (\mathsf{the} \circ \mathsf{tortillas})) \vdash s}\ [\backslash E]$$

Figure 2.3: Natural deduction derivation of "Maria makes the tortillas"

2.2.3 Rules of abstraction

In the last section we considered the composition of grammatical expressions by deconstructing formulas. The base logic also addresses the possibility of constructing formulas by abstracting expressions from a composite structure. The logical rules for abstraction are opposite to the Modus Ponens inferences, defined by the introduction rules of the binary connectives (Fig. 2.4).

The introduction rules allow us to use *hypothetical reasoning*: the possibility to reason on the basis of a conditional assumption. At a certain point in the derivation the assumption has to be withdrawn by means of $/, \backslash$-introduction steps. An example of a theorem which uses hypothetical reasoning is *lifting*.

$$\text{Lifting:} \quad A \longrightarrow B/(A\backslash B),\ A \longrightarrow (B/A)\backslash B$$

$$\frac{\Gamma \circ B \vdash A}{\Gamma \vdash A/B}\ [/I] \qquad \frac{B \circ \Gamma \vdash A}{\Gamma \vdash B\backslash A}\ [\backslash I]$$

Figure 2.4: Introduction rules for $/, \backslash$

Using this theorem, formulas can be lifted to a higher order formula (e.g. $np \rightarrow s/(np\backslash s)$). One assigns higher order types (types with a nested implication) to invoke the process of hypothetical reasoning.

Examples of linguistic expressions that use higher order type assignments are non-constituents (Morrill, 1994) and generalized quantifier expressions. Relative pronouns, such as which are another example. In the noun phrase "the tortillas which Maria makes", which combines with a relative sentence that misses an object phrase. The verb makes with the type assignment $(np\backslash s)/np$ (see Fig. 2.1) needs to combine with a hypothesized object first to be able to combine with the subject Maria. To invoke hypothetical reasoning on the object phrase, which gets assigned the higher order type: $(n\backslash n)/(s/np)$. Fig. 2.5 shows only the logical part of the derivation; for the structural part we need ways to restructure.

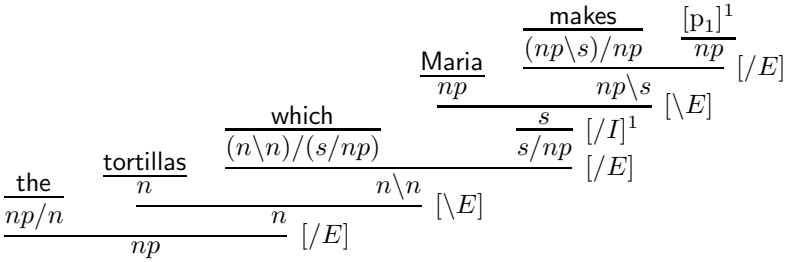


Figure 2.5: Derivation of "the tortillas which Maria makes" using hypothetical reasoning

2.3 Structural Reasoning

The composition of words produces larger phrasal structures. Within Lambek's sequent calculus, the resources are structured in two dimensions: *linear order* on the horizontal dimension and *dominance* on the vertical dimension. These structural dimensions are invoked by the lexical formulas: linear order is stated by the precedence relations defined by the directionality of $\{/, \backslash\}$ and dominance comes from the hierarchical grouping (for example $(np\backslash s)/np$ versus $np\backslash(s/np)$). As linguistic examples show, the order and hierarchical grouping sometimes has to be more flexible than defined in the base logic; one needs ways to reorder and restructure sentences.

2.3.1 Reordering/restructuring

A global option

We present derivations in Prawitz style, but structural rules are presented in an axiomatic style. This style only gives the structural part of the derivation where the structural operator is the $\bullet$. The antecedent of a structural postulate represents the structure of the upper part of a derivation step, the succedent shows the structure of the lower part:

$$\frac{\Delta \vdash A}{\Gamma \vdash A}\ [SR] \quad \Longleftrightarrow \quad \Gamma \longrightarrow \Delta \quad [SR]$$

An extension of the base logic, the non-associative Lambek calculus (**NL**) with two postulates give us the possibility to relax the strict relation of precedence and dominance between the resources. We use *Commutativity* to alter the linear order of structures and *Associativity* to change the immediate dominance relations.

$$\begin{array}{lrcl} \text{Associativity:} & (A \bullet B) \bullet C & \longleftrightarrow & A \bullet (B \bullet C) \\ \text{Commutativity:} & A \bullet B & \longrightarrow & B \bullet A \end{array}$$

Figure 2.6: Structural rules

The structural rules in combination with the logical rules of application and abstraction allow us to derive more structures than was feasible in the base logic NL. From the base logic different type logical systems can be obtained by extending the system with one or both structural postulates. The basic non-associative Lambek calculus **NL** enriched with associativity gives the type-logical system **L**, with commutativity **NLP**, and with both **LP**. Every type logical level shows some deficiency: without associativity or commutativity one loses possibilities to restructure or reorder sentences. But with one or both structural postulates there is the danger of overgeneration: every reordering and restructuring in a precedence relation is possible. In natural language such a system is much too flexible. With no structural rules the system is much to strict, with only logical theorems and rules to build structures. Ways are needed to regulate the use of the four systems.

Controlled resource management

Moortgat and Oehrle (1994) point out that the structural rules given in Fig. 2.6 are a global option to reorder or restructure sentences. Natural language requires a more local treatment to account for the subtleties in strictness of linguistic composition. They suggest to treat the differences between grammars with structural options constrained by a package of sorted logical modalities. Instead of global structural choices, one needs lexical control over resource management. In this way we account for a grammatical fine-structure from a logical perspective. The multimodal framework of Categorial Grammar elaborates the possibilities for restricted use of the structural rules.

2.4 The Multimodal Framework

As outlined in section 2.3 the four structural settings that form the categorial landscape have advantages and disadvantages. In the past decade many extensions to refine the categorial landscape have been proposed. One extension leads to Multimodal Categorial Grammar. The different resource management properties and the different directions of refining were the topic of Kurtonina and Moortgat (1996). The goal is to combine different levels of structure and order sensitivity in one framework.

This section gives an outline of extending the basic Lambek calculus to a multimodal framework. The first step is to introduce multiple composition modes for the binary connectives. The next step is to add unary modal operators to the logical setting. These connectives can be further refined with different sorts. The extensions yield a system which is much more fine grained and discriminating than the non-associative and non-commutative Lambek calculus.

2.4.1 Multiple modes of composition

The logical framework is extended by adding multiple modes of composition; every binary connective is decorated with an index. An index indicates a certain composition mode, which follow the same logical rules, but differ on the structural side. The logical binary connectives have a structural component, which interacts with the structural postulates. By the assignment of the right type-logical formulas to the lexical entries, the accompanying structural framework is invoked. The modes on the binary connectives specify which structural rules can be applied.

The formula language extended with these operators becomes:

$$\mathcal{F} \ ::= \ \mathcal{A} \mid \mathcal{F}/_i\mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F}\backslash_i\mathcal{F}$$

With the structural part of the framework:

$$\mathcal{S} \ ::= \ \mathcal{F} \mid (\mathcal{S} \circ_i \mathcal{S})$$

Instead of global options of a total associative or commutative regime, we can now define structural postulates that are keyed to certain composition modes. For example, mode a indicates the possibility to reason with associativity, c to reason with commutativity, n to have no access to any of the structural postulates.

$$\begin{array}{lrcl} \text{Associativity:} & (A \bullet_a B) \bullet_a C & \longleftrightarrow & A \bullet_a (B \bullet_a C) \\ \text{Commutativity:} & A \bullet_c B & \longrightarrow & B \bullet_c A \end{array}$$

Now we need communication between different structural regimes to have access to the combined inferential capacities of the different logics (Kurtonina and Moortgat, 1996). The theory offers two ways to communicate between the different composition modes: *inclusion* and *interaction* postulates. Fig. 2.7 shows an example of possible postulates. The inclusion postulate serves as a bridge between the modes; they specify an order on the different modes (less or more informative). For example one could decide that the structure that is composed with the less informative product has a greater freedom of resource management than the other. The interaction postulates mix the different modes

in order to distribute the resources. The first example is called Mixed Commutativity (MC), because it alters the ordering of the structure. The second example is called Mixed Associativity (MA), because it restructures the dominance relation. An important distinction in this thesis is the *dependency* relation between

$$\begin{array}{lrcll} \text{Inclusion:} & A \bullet_i B & \longrightarrow & A \bullet_j B & \\ \text{Interaction:} & (A \bullet_i B) \bullet_j C & \longrightarrow & (A \bullet_j C) \bullet_i B & \text{(MC)} \\ & (A \bullet_i B) \bullet_j C & \longrightarrow & A \bullet_i (B \bullet_j C) & \text{(MA)} \\ & \multicolumn{4}{c}{i \text{ is more informative than } j} \end{array}$$

Figure 2.7: Inclusion and Interaction postulates

two expressions (specifier-head, head-complement or modifier-head). I will capture this distinction in terms of two binary modes: $(<, >)$; the 'arrows' indicate the position of the head towards its dependency. The interaction postulates account for the communication between the dependency relations among the resources.

2.4.2 Control features

The next step is to extend the Lambek calculus by adding two unary operators to the base logic $(\Diamond, \Box)$. In this section only the logical and structural properties of the two modals will be examined. Moortgat (1996) thoroughly describes what the motivation and implications are for extending Categorial Grammar with modal operators.

The formula language extended with these operators becomes:

$$\mathcal{F} \ ::= \ \mathcal{A} \mid \Box\mathcal{F} \mid \Diamond\mathcal{F} \mid \mathcal{F}/_i\mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F}\backslash_i\mathcal{F}$$

With the structural part of the framework:

$$\mathcal{S} \ ::= \ \mathcal{F} \mid (\mathcal{S} \circ_i \mathcal{S}) \mid \langle\mathcal{S}\rangle$$

The residuation laws show the behavior of the unary modal operators.

$$\Diamond A \vdash B \iff A \vdash \Box B$$

The behavior of the unary connectives is also described by the following theorems. The box and diamond serve as 'key' and 'lock', since they cancel each other.

$$\begin{array}{llll} \text{from} & \Box A \rightarrow \Box A & \text{derive (using} \Rightarrow \text{Residuation)} & \Diamond\Box A \rightarrow A \\ \text{from} & \Diamond A \rightarrow \Diamond A & \text{derive (using} \Leftarrow \text{Residuation)} & A \rightarrow \Box\Diamond A \end{array}$$

The residual behavior of the two unary connectives as inverted duals, is captured by the logical rules, where the logical connective on the right-hand side interacts with the structural connectives on the left-hand side. The interaction plays an important role in the structural control of the lexical resources. The rules in natural deduction style in Fig. 2.8 show the interaction between the structural and logical components of the unary connective.

The structural operator $\langle . \rangle$ serves as a control feature for the structural part of the derivation. As a logical consequence of modal theory this system follows

$$\frac{\Gamma \vdash \Box A}{\langle \Gamma \rangle \vdash A}\ [\Box E] \qquad \frac{\Gamma \vdash A}{\langle \Gamma \rangle \vdash \Box A}\ [\Box I]$$

$$\frac{\Gamma \vdash A}{\langle \Gamma \rangle \vdash \Diamond A}\ [\Diamond I] \qquad \frac{\Delta \vdash \Diamond A \quad \Gamma[\langle A \rangle] \vdash B}{\Gamma[\Delta] \vdash B}\ [\Diamond E]$$

Figure 2.8: Control features

certain laws, which can be stated as structural postulates. The postulates constrain the use of the connectives. Two well-known postulates in modal theory, but less applicable to linguistics, are the theories: 4 (Transitivity) and T (reflexivity) (see Versmissen (1996)). Postulates that can be applied to linguistics are the distributivity postulates: $[K, K1, K2]$. The distributivity postulates are useful in the analysis of movement within minimalist grammars. $[K]$ postulates strong distributivity by splitting the structural operator to both sides of the composition. The weak distributivity postulates, $[K1]$ and $[K2]$, carry the structural operator over to one side of the composition relation.

$$\begin{array}{ll} \Diamond(A \bullet B) \longrightarrow \Diamond A \bullet \Diamond B & [K] \\ \Diamond(A \bullet B) \longrightarrow \Diamond A \bullet B & [K1] \\ \Diamond(A \bullet B) \longrightarrow A \bullet \Diamond B & [K2] \end{array}$$

The unary connectives are called *control* features, because they regulate the use of the structural postulates by structurally decorating expressions with the unary connective $\Diamond$. Fig. 2.9 shows how the structural connective controls reordering and restructuring postulates.

$$\begin{array}{lrcl} \text{Commutativity}_{\Diamond}: & \Diamond A \bullet_i B & \longrightarrow & B \bullet_i \Diamond A \\ \text{Associativity}_{\Diamond}: & (A \bullet_i B) \bullet_i C & \longleftrightarrow & A \bullet_i (B \bullet_i C) \end{array}$$

Where one of $\{A, B, C\}$ is decorated with $\Diamond$

Figure 2.9: Structural postulates under control of $\Diamond$

2.4.3 Multiple modes of control

The control mechanism is further refined by adding modes to the unary connectives.

The formula language becomes:

$$\mathcal{F} \ ::= \ \mathcal{A} \mid \Box_f \mathcal{F} \mid \Diamond_f \mathcal{F} \mid \mathcal{F} /_i \mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F}$$

The structural part of the framework:

$$\mathcal{S} \ ::= \ \mathcal{F} \mid (\mathcal{S} \circ_i \mathcal{S}) \mid \langle \mathcal{S} \rangle^f$$

There are two uses for the introduction of multiple sorts of unary connectives. One use is to store morphosyntactic information as explored by Heylen (1999).

He uses the different modes of the unary connectives as feature information packages. Features such as case, gender, person, number, tense and agreement can be added as modes to the unary connectives to the lexical structures. Additional feature information helps to derive the right phrasal structures.

The decorated lexical formulas form the feature specification of words. Instead of stating that a verb requires a noun phrase as subject in the specifier position ($np\backslash_{>}s$), one can now specify the necessary feature information. For example, loves is a singular, third person, finite verb. Therefore it needs to combine with a noun phrase with exactly those properties; we can decorate the type of loves as ($\Box_{sg}\Box_{3rd}np\backslash_{>}s$). The number of unary modalities is equal to the number of features one would like to specify.

Features are given in the lexicon as $\Box$ decorations on the formula or subformulas of a word. In this way one could specify very precisely what kind of categories a transitive verb, such as love wants to be merged with.

$$\mathsf{loves} \vdash \Box_{pres}\Box_{fin}((\Box_{nom}\Box_{sg}\Box_{3rd}np\backslash s)/\Box_{acc}\Box_{number}np)$$

Some lexical entries leave the value of some features unspecified. For example the number feature for the argument of loves is unspecified (*number*). To be able to combine loves with an object, which is specified for the number feature, the relation between unspecified and specified features needs to be determined. The inclusion postulates regulate the specification relations between features. For more explanation on the feature landscape see Heylen (1999).

In this thesis I will concentrate on another use of differentiating sorts of unary connectives: the refinement of structural control. With the use of sorted unary connectives $\Box_f, \Diamond_f$ we can make more restrictive use of structural postulates. With the possibility to decorate the necessary postulates with structural unary and binary operators, one is able to lexically control the use of these postulates. Different sorts of unary and binary operators help us to refine the system, which makes it better applicable to linguistic applications.

2.5 Term Decoration

In this section we consider the possible interpretation of derivations. For the semantics of derivations Lambda-calculus is used. The term language is defined as follows, where $\mathcal{V}^A$ is the set of term variables of type A.

$$\begin{array}{rcl}
\mathcal{T}^A & ::= & \mathcal{V}^A \mid (\mathcal{T}^{B\backslash_i A}\ \mathcal{T}^B) \mid (\mathcal{T}^{A/_i B}\ \mathcal{T}^B) \mid {}^{\vee}\mathcal{T}^{\Box_f A} \mid {}^{\cup}\mathcal{T}^{\Diamond_f A} \\
\mathcal{T}^{A/_i B} & ::= & \lambda\mathcal{V}^B.\mathcal{T}^A \\
\mathcal{T}^{B\backslash_i A} & ::= & \lambda\mathcal{V}^B.\mathcal{T}^A \\
\mathcal{T}^{\Box_f A} & ::= & {}^{\wedge}\mathcal{T}^A \\
\mathcal{T}^{\Diamond_f A} & ::= & {}^{\cap}\mathcal{T}^A
\end{array}$$

Now we can label the natural deduction rules for the unary and binary connectives, as presented in Fig. 2.10 and Fig. 2.11.

2.6 Summary

The non-associative Lambek system introduced in section 2.2 is enriched with multimodal binary connectives and feature decorated unary operators. One

$$x : A \vdash x : A$$

$$\frac{\Gamma \circ_i x : B \vdash t : A}{\Gamma \vdash \lambda x.t : A/_i B}\ [/I] \qquad \frac{\Gamma \vdash t : A/_i B \quad \Delta \vdash u : B}{\Gamma \circ_i \Delta \vdash (t\ u) : A}\ [/E]$$

$$\frac{x : B \circ_i \Gamma \vdash t : A}{\Gamma \vdash \lambda x.t : B\backslash_i A}\ [\backslash I] \qquad \frac{\Gamma \vdash u : B \quad \Delta \vdash t : B\backslash_i A}{\Gamma \circ_i \Delta \vdash (t\ u) : A}\ [\backslash E]$$

where $i \in \{<, >\}$

Figure 2.10: Term decorated ND-rules for the binary connectives

$$\frac{\langle \Gamma \rangle^f \vdash t : A}{\Gamma \vdash {}^\wedge t : \Box_f A}\ [\Box_f I] \qquad \frac{\Gamma \vdash t : \Box A}{\langle \Gamma \rangle^f \vdash {}^\cup t : A}\ [\Box_f E]$$

$$\frac{\Gamma \vdash t : A}{\langle \Gamma \rangle^f \vdash {}^\cap t : \Diamond_f A}\ [\Diamond_f I] \qquad \frac{\Delta \vdash u : \Diamond_f A \quad \Gamma[\langle x : A \rangle^f] \vdash t : B}{\Gamma[\Delta] \vdash t[{}^\cup/x] : B}\ [\Diamond_f E]$$

Figure 2.11: Term decorated ND-rules for the unary connectives

can derive dominance and direction sensitive ordered structures which show the dependency relations among the structural resources.

The logical and structural landscape of the multimododal framework described in section 2.2 and 2.3 is a rich system, which can define various linguistic phenomena. The communication between the various type systems is regulated by the use of multiple composition modes and is controlled by different sorts of unary connectives.

Section 2.5 introduces semantic labeling that shows the history and the interpretation of the derivation. The semantics of the derivations give possibilities to check for the interpretation of well-typed expressions.

Chapter 3

The Minimalist Program

The Minimalist Program (Chomsky, 1995, Ch4) is a model of Universal Grammar which attempts to capture the processes involved in understanding human language. Section 3.1 starts with a schematic overview of the different components of the Minimalist Program. The following sections describe the different components in more detail. Section 3.2 addresses the lexicon with respect to the content and the way information is stored. Section 3.3 gives the basic operations of the computational system that uses the content of the lexicon to trigger the right processes to generate PF and LF output. In the second part of this section we explicate the operation MOVE and the mechanisms that play a role in extraction phenomena. Section 3.4 shows the extra conditions that Chomsky (1995) states on the Computational System and on the derivation of LF and PF structures. As an illustration in section 3.5 we derive an example of an extraction phenomenon.

3.1 Schematic Overview

As shown in Fig. 3.1, the two major components of the Minimalist Program are the *lexicon* and the *Computational system of Human Language* (C_{HL}). The lexicon serves as storage for the lexical objects of a language, which are described in terms of feature bundles. The Computational System is the generator of grammatical output which serve the cognitive abilities such as hearing, speech and understanding. Chomsky (1995) distinguishes two forms of grammatical output: a Phonological Form (PF) and a Logical Form (LF). The phonological form serves the sensorimotor system (speech, hearing, grammar); the logical form serves as the instructions to the meaning system (understanding, learning, thought). The two levels are generated by the application of basic operations under the influence of a control mechanism defined within the Computational System. If one wants to capture a grammar of any specific language, one needs to establish a linguistic landscape which contains at least a lexicon and mechanisms such as described by the computational system.

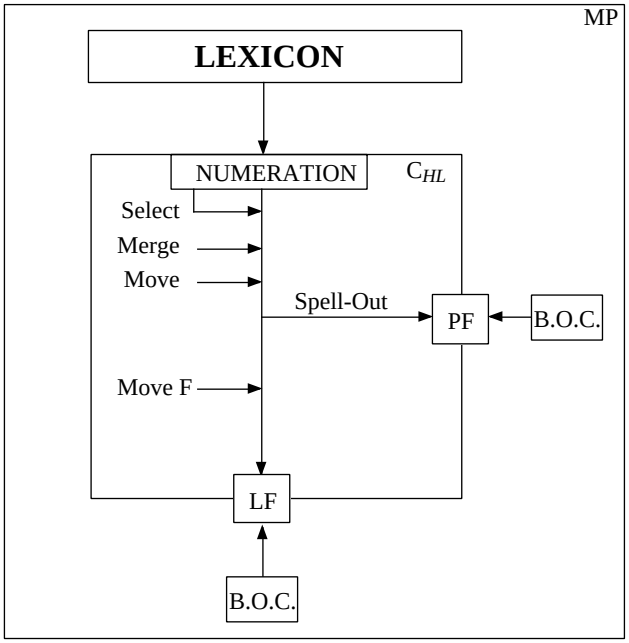


Figure 3.1: The Minimalist Program

3.2 The Lexicon

The Computational System has access to lexical items on which it can operate. The lexical items are collected in a *lexicon*, a storage for information on form, category, phonology and syntax. Each feature has a specific role in the Computational System. The C_{HL} uses phonological features to generate PF forms, other features are necessary to generate the right LF forms. Certain 'formal' features influence the application and the performance of operations in the generation of both forms of output (see section 3.3.2).

Lexicologists discuss how information is stored in the lexicon, for example inflectional information. Zwart (1997) describes the distinction between weak and strong lexicalist views. A weak lexicalist view holds a distinction between derivational and inflectional morphology. Derivational inflection is the manipulation of inflected forms by syntactic rules, whereas inflectional morphology states that inflection is already present in the lexicon. Strong lexicalism applies to inflectional morphology, where words enter the syntactic machinery already fully inflected. Chomsky (1995) holds a strong lexicalist view.

To keep things simple the lexicon of the Minimalist Program is presented as an unstructured data bank with entries which contain feature information and semantic information of words. Several theories on lexicon architecture, such as DATR (Gazdar, 1985), offer ways to structure lexical specifications.

3.3 The Computational System

This section sketches the basic components, which a minimalist grammar needs to address. The computational system, C_{HL} in Fig. 3.1 shows the different components, operations and conditions of the system. The computation starts with the numeration, a selection of lexical items from the lexicon. A small set of basic operations works on the numeration to derive both LF and PF. The derivation begins by selecting lexical items from the numeration. The derivation continues with a certain number of MERGE steps that combine lexical items into larger structures. The merging of structures alternates with MOVE steps, which are initiated by the need for feature checking. The operation MOVE and the influence of certain features on this operation is explored more thoroughly in section 3.3.2. At Spell-Out, the phonological part of the derivation is sent to PF. The formal and semantic feature information is sent to LF where under influence of more MOVE steps the derivation reaches the LF structure. The derivation converges if the numeration is empty and all the conditions (described in section 3.4) hold.

3.3.1 Basic operations

The Computational System uses three basic operations to generate PF and LF forms. The presentation of a derivation, as tree structures where the nodes are labeled by the head category, is just informal. (The discussion about representation and labeling is left as background reading.)

A derivation starts with a selection of the lexicon, the numeration. The numeration stores the lexical items needed for the derivation to converge as a multiset of lexical resources. The lexical items in the numeration contain the necessary feature information. All extra information imposed from the "outside" (language dependent) is added to the feature information before the numeration is formed. An example of such a numeration is:

$$\textit{Numeration} \;=\; \{\texttt{C, drinks,the,girl,coffee}\}$$

The order of the items in the numeration is trivial. Apart from the functional category `C`, all lexical resources contain semantic, phonological and formal feature information. The different groups of features are explained in section 3.3.2.

Select

SELECT is the first operation in the derivation. SELECT brings lexical items from the numeration into the derivation. For example, SELECT removes `coffee` from the numeration to form the Syntactic Object {`coffee`}, as shown in Fig. 3.2. For clarity, the category of the word is written to indicate the use of this word within a structure.

N
`coffee`

Figure 3.2: Syntactic Object

The numeration reduces to:

$$\textit{Numeration} \;=\; \{\texttt{C, drinks,the,girl}\}$$

Merge

MERGE is the next operation in the derivation. MERGE takes two Syntactic Objects and merges them together into a new Syntactic Object. One of the syntactic objects is determined as the head. Chomsky (1995) suggests that information about the head of two lexical items is stated in the lexicon where the selection properties of a lexical head are specified. The argument structure of a possible head licenses a certain number of arguments. The head selects its arguments on the basis of its argument structure. The role of the head is to determine the category of the newly formed structure, which settles the relation between the head and its subsequent complements or specifiers. The Computational System uses this lexical information to build the structure of a derivation.

Fig. 3.3 illustrates the merging of `the` and `girl` into the noun phrase `the girl`. The determiner, `the`, with category *D* (determiner) projects over its argument, the noun `girl`. The derivation yields the following syntactic structure, labeled with the projected category of the head, *D*.

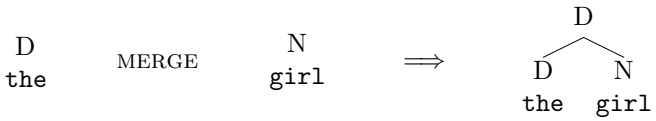


Figure 3.3: Sample of Merge

With the two basic operations, SELECT and MERGE, simple structures can be derived, such as given in Fig. 3.4. More complex structures are constructed with the operation MOVE.

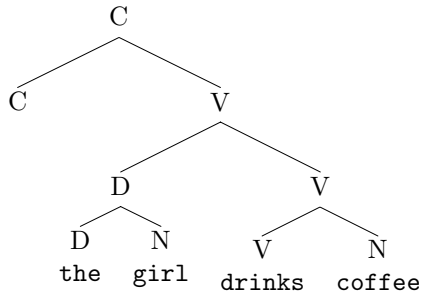


Figure 3.4: Simple phrasal structure

Move

MOVE is the mechanism which describes phenomena of "displacement". Chomsky (1998, page 17) describes the "displacement" property as follows:

> "Phrases are interpreted in positions other than those where they are heard, though in analogous expressions these positions are occupied and interpreted under natural conditions of locality."

The following examples illustrate the displacement property. Linguistic research shows that displacement occurs in many natural languages.

(3.1) What tortillas does Maria make?
∗ Maria makes what tortillas?

(3.2) Haza fogok akarni menni
Home will[1sg] want[inf] go[inf]
'I will want to go home'

∗ fogok akarni haza menni
will[1sg] want[inf] home go[inf]

(3.3) dat Alice de koning wil kunnen plagen
that Alice the king want[3sg] be able[inf] tease[inf]
'that Alice wants to be able to tease the king'

∗ dat Alice de koning plagen kunnen wil
that alice the king tease[inf] be able[inf] want[3sg]

Example (3.1) (Stabler, 1996) shows wh-raising of an object phrase, where the object phrase is obliged to move to precede the finite verb. Example (3.2) shows an instance of *verb modifier climbing* in Hungarian (Koopman and Szabolcsi, 1998). In so-called neutral sentences the verb modifier `haza` has to climb, to precede the finite verb. Section 4.3 explains more about the Hungarian verbal complex phenomenon. Example (3.3) is an example of Dutch verb raising (Moortgat and Oehrle, 1994) and shows how the order of the verbs can change in a SOV-language. Modal auxiliaries, such as `wil` (= 'want'), subcategorize for bare infinitival complements on the left, such as `kunnen` (= 'be able'). In the standard order the finite verb appears on the last position, but in forming a verb cluster the order between the auxiliary and the infinitival complement changes.

The Computational System has to account for the concept of displacement. The question arises: What controls the movement of phrases within sentences? The answer is linked to another linguistic notion: *interpretability*. Interpretability of sentences is an important property within structures. (Un-)interpretability on LF and PF structures is determined by the Bare Output Conditions (more on BOC in section 3.4.1). One Bare Output Condition, *Full Interpretation*, states that as long as uninterpretable features are present in the derivation, the derivation cannot converge. The condition prescribes that all uninterpretable features must have been removed.

3.3.2 Controlling Move

The Computational System has to give an explanation for both linguistic notions: *displacement* and *interpretability*. Displacement as described by the examples (3.1)-(3.3) is a concept that occurs in almost all languages. According to Chomsky (1995), uninterpretable features are responsible for displacement. In the Minimalist Program he links these two notions to build a theory on the operations MOVE/ATTRACT and the procedure *Feature Checking*.

The procedure of feature checking depends on the kind of features involved. Every kind of feature plays a certain role within a grammar. The following inventory[1] describes the role of the different features in natural language. After explaining the basic mechanisms of feature checking, the operations MOVE and ATTRACT are defined. MOVE/ATTRACT transfer feature information to the position where feature checking can take place.

An inventory of Features

As mentioned in section 3.2, the lexicon is a storage of lexical resources. Every lexical item is described by its feature structure. The behavior of a word depends on the features that make up its feature structure. Chomsky (1995) distinguishes different groups of features, which fulfill a defined role within the Computational System. The features can be divided into three groups of semantic, phonological and formal features. The formal features can be further divided into interpretable or uninterpretable features at PF or LF.

Semantic features Semantic features are needed for the interpretation of words at LF. The semantic features capture the meaning of words and the relations between words. For example, a verb like `eat` needs an agent, an 'eater' with the semantic value: 'animate'. The object of the verb `eat` could carry the semantic value: 'eatable'. The development of semantic theories is a direction in linguistics that will not be pursued here.

Phonological features The Phonological Form is the interpretational level that interprets the phonological features. The phonological features carry information, which at this level, are used to transfer morphological information to the articulatory-perceptual system. At a certain point in the derivation, at Spell-Out (see Fig. 3.1), the phonological features are sent to PF. The derivation continues to generate an LF structure and will crash if some phonological features are still present at LF.

Formal features The formal features are localized as a bundle of features on the lexical item. These features influence the working of the Computational System. Formal features such as ϕ-features (person, number, gender), category (noun, verb, preposition), case (nominative, accusative, dative) have their own function within the Computational System. They play an important role in the derivation of sentences, as is explicated below in the paragraph on *Feature Checking*.

Functional categories, such as determiner, auxiliary and complementizer form a special class among the formal features with an essential grammatical function in the generation of expressions. Some functional categories have no phonological or semantic feature information (complementizer, tense); they only serve the computational system to derive the right word order. The formal features assigned to functional categories control the application of the operation MOVE. The features that are involved with controlling movement have a special status, which is expressed by the interpretability at LF or PF.

[1] For a complete list of features with their properties, see appendix A

Interpretability at LF Some features are *interpretable* at LF, which means that the LF interface needs these features for interpretation. In contrast, the *uninterpretable* features cannot be interpreted at the LF interface and therefore need to be deleted. *Full Interpretation* prescribes that a derivation will only converge if there are no uninterpretable features left in the derivation. One needs to have an operation such as *feature checking* to remove the uninterpretable features from the lexical structures.

Two examples of formal features that are uninterpretable at LF are the `case` feature of nouns and the ϕ-features of verbs. All the semantic features are interpretable, but all the phonological features are uninterpretable at LF.

Interpretability at PF The phonological features are the only interpretable features at PF. At Spell-Out these features are removed from the derivation and sent to the PF interface. Some uninterpretable features at both PF and LF are distinguished from the other interpretable and uninterpretable features by assigning them the *Strength* feature. Strength is a feature of a feature, which is only assigned to formal features of functional categories. The strength feature needs to be removed before spell-out. In order to check a feature with the strength feature assigned to it, the uninterpretable strength needs to be deleted first.

Feature Checking

The distinction between interpretable and uninterpretable invokes two types of checking relations: symmetric checking and asymmetric checking. Symmetric checking is done when both features are interpretable or uninterpretable. Asymmetric checking is done in all other cases where an uninterpretable feature is in a checking relation with an interpretable feature.

On the basis of the different sorts of interpretable and uninterpretable features the following checking relations can occur. A strong feature, which is uninterpretable at PF, has to check against a corresponding feature, then the strength of the feature is deleted. After the checking of the strength the feature itself can be checked. An uninterpretable feature at LF in a checking relation with a corresponding uninterpretable feature (for example [`case`]) results in the deletion of both uninterpretable features. An asymmetric checking relation between an uninterpretable and an interpretable feature results in deletion of the uninterpretable feature.

Along the derivation, interpretable features can get into symmetric checking relations when they move along with an attracted (un-)interpretable feature. Symmetric checking of two interpretable features leaves deletion, they only 'communicate' their shared information; if the information differs, the derivation will crash. For example, two interpretable [`person`] features where one is instantiated as third person and the other first person cannot check, so the derivation fails.

Feature checking supports Full Interpretation, which prescribes that uninterpretable features must be deleted in order for the derivation to converge. Feature checking describes how features are deleted. Now it is necessary to describe when and how features get into a checking relation. The operations ATTRACT and MOVE lay the basis of feature checking.

Attract/move

Uninterpretable features have to be checked in a checking relation with features with corresponding feature values. Under influence of the operation ATTRACT the feature that needs to check an uninterpretable feature, is attracted to the uninterpretable feature. The operation is defined by Chomsky in terms of two former economy conditions: the *Minimal Link Condition* and *Last Resort*. The Minimal Link Condition states that the distance of the link between a moved phrase and its original position should be minimal. Last Resort states that all transformations are driven by the need to check a feature.

Attract Syntactic element K attracts a feature F if F is the closest feature that can enter into a checking relation with a sublabel of K (Chomsky, 1995, p.297)

Mainly functional categories carry uninterpretable features, they act as attractors of feature bundles. A functional category with an uninterpretable feature attracts a corresponding interpretable or uninterpretable feature. The attracted feature carries along other feature information, which can take part in the feature checking operation. The operation MOVE transfers the feature bundle to the domain of the functional category. The two features will now form a checking configuration.

A syntactic element with an uninterpretable feature attracts just enough material for the derivation to converge. For uninterpretable features at LF, the formal features are the only material needed for convergence. If the attractor carries an uninterpretable *strong* feature, which is uninterpretable at PF, the phonological and semantic feature information is also moved to the checking domain.

3.4 Conditions

Not all information in a grammar can be stated as feature information in the lexicon or as procedures in C_{HL}. Chomsky (1995) posits Bare Output Conditions and economy conditions to capture this extra information.

3.4.1 Bare Output Conditions

Some conditions are dictated "from the outside" at both interface levels. These Bare Output Conditions restrict the possible output of the computational system. Only a subset of the set of derivations produced by C_{HL} converges under the influence of the Bare Output Conditions. With regard to the conditions at PF one could think of unpronounceable sequences of 'words'. At LF one could think of uninterpretable sentences, like Chomsky's famous "Colorless green ideas sleep furiously", which is a grammatical expression but without meaning.

3.4.2 Economy Conditions

MOVE/ATTRACT are further restricted by economy conditions; in the evolution of the Minimalist Program these conditions are incorporated in the definition of the basic operations. In earlier proposals global economy conditions were stated to determine admissible derivations, such as Shortest Link and Last Resort.

These economy conditions state restrictions on how and when a certain feature is transferred. In the Minimalist Program only local economy conditions are stated in the Computational System; most of them are incorporated in MOVE/ATTRACT operations. The only economy condition left is *Procrastinate* to disfavor the movement of phonological and semantic features; feature movement at LF is favored over MOVE operations before Spell-Out.

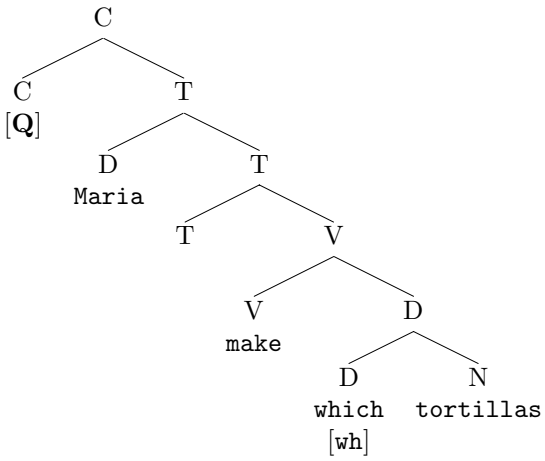
3.5 A sample derivation

All the components for a minimalist derivation have been introduced in the previous sections. As an illustration the sentence "`Which tortillas did Maria make`", an example of wh-movement, is derived below. The sentence carries information on tense, but as we focus on the features that are involved with the movement of `which tortillas`, we will abstract away from do-support. So the Tense position stays empty.

The derivation starts with the numeration with all the necessary lexical resources to build PF and LF structures. In our simplified example the numeration contains the following elements: {`C, T, Maria, tortillas, make, which`}. All lexical elements, apart from the functional categories `C` and `T`, carry formal, semantic and phonological feature information.

One by one the operation SELECT removes all lexical items and introduces them as Syntactic Objects into the derivation. The basic structure of the sentence is led by the operation MERGE, which joins Syntactic Objects into phrases. The features on the Syntactic Objects lead the right application of operations. After the first MERGE steps, the derivation arrives at the structure presented in example (3.4). The structure shows the dominance and precedence relations between the syntactic objects. The dominance relations between the syntactic objects are determined by the category labels; for example the syntactic object `which` dominates `tortillas`.

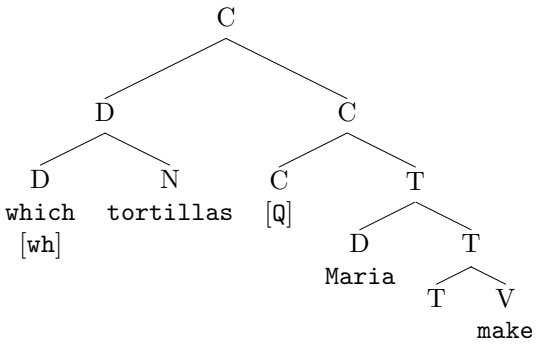
(3.4)



Under the influence of uninterpretable (strong) features on the functional categories, the operation MOVE transfers subtrees of the structure to the checking domain of the functional category. The [**Q**] feature on the functional category C is strong and needs to be checked. The closest feature which can enter

into a checking relation with [**Q**], is the [wh] feature `which`. The operation MOVE/ATTRACT transfer the formal features of `which` to the checking domain of C. Because the uninterpretable feature is strong, the semantic and phonological feature information of `which` and `tortillas` (because `which` dominates `tortillas`) pied-pipes with the formal features to the checking domain. The result is the structure presented in example (3.5).

(3.5)



Feature checking removes the uninterpretable strength of the interpretable feature [Q]. At this moment the derivation reaches spell-out, because all strong features are deleted and the numeration is empty. At spell-out the phonological features are removed from the structure to form PF. The remains of the structure, the semantic and formal features, continue the derivation.

The uninterpretable features have to be deleted in order for the derivation to converge. When all uninterpretable formal features are deleted, LF is reached, where all formal and semantic features are interpreted. The [wh] feature of `which tortillas` and the [Q] on the functional category C account for the interpretation of the sentence as a wh-question.

3.6 Summary

With the Minimalist Program, Chomsky models a theory on Universal Grammar. We concentrate on the basic principles, which need to be present in grammars of language. The basic elements: the lexicon, the operations and the output levels, are presented rather informally. The theory lacks an effective algorithm to compute derivational structures. The next chapter presents the work of Stabler (1996), further elaborated in Stabler (1999), that will serve as an algorithm capturing the basic minimalist mechanisms.

Chapter 4

Computational models of Minimalism

This chapter presents Stabler's computational account of the Minimalist Program. Section 4.1 explains the different components of Stabler's Minimalist Grammar formalism (Stabler, 1999). In section 4.2 we use his formalism to relate the Minimalist Program to Multimodal Categorial Grammar. Section 4.3 illustrates this comparison further with an elaboration of Hungarian verb movement (Koopman and Szabolcsi, 1998).

4.1 A Minimalist Grammar

Stabler's framework captures the main components of the Minimalist Program.

4.1.1 Features

A central idea in the Minimalist Program (Chomsky, 1995) is that derivations are feature driven. In his formalism Stabler explains how features trigger structure building operations, setting aside many details of recent proposals. He defines the elementary operations and the basic objects of these operations: features. With the formalism we can formulate a minimalist grammar for any language phenomenon.

Features ($\mathcal{F}$) are part of a lexical specification. We distinguish phonological, semantic and syntactic features. The syntactic features play an active role in controlling derivations. Every structure building operation is triggered by a certain syntactic feature. Stabler focuses on the syntactic features, and abbreviates phonological and semantic feature information. Possible features are given in the following BNF-specification. Fig. 4.1 gives a summary of these features and the possible feature values.

$$\mathcal{F} ::= \mathcal{N} \mid =\mathcal{N} \mid +\mathcal{N} \mid -\mathcal{N} \mid /\mathcal{N}/ \mid \underline{\mathcal{N}}$$

The phonological and semantic non-syntactic features are carried along in the derivation, but have no effect on the structure building operations. For simplicity we only write the headword to indicate that non-syntactic material is present in the derivation.

- phonological features: /marie/
- semantic features: marie
- syntactic features:
  - category features:
    - basic categories: $c, t, v, d, n, \ldots$
    - selector features: $=c, =t, =v, =d, =n, \ldots$
  - control features:
    - licensees: $-case, -wh, \ldots$
    - licensors: $+case, +wh, \ldots$

Figure 4.1: Features of MG

The syntactic features are divided in 2 groups: *category* features and *control* features. The category features state the role of a word in a sentence. Every word gets assigned a category, for example: *complementizer*, *tense*, *verb*, *determiner* or *noun*. Some of these categories show an extended functionality. Categories such as complementizer, determiner and tense are so-called functional categories, because they play a special role in derivations. The role of a word is further determined by the selector feature. The selector feature indicates with what kind of category a word can be combined. The two category features come in pairs; in a derivation a word with a selector feature is always accompanied by a word with a matching category feature.

Apart from the category features, control features play an important role in controlling the ordering of words and the movement of phrases within structures. The licensee features state certain properties of words, such as [−`case`], [−`wh`] and [−`tense`], while the licensor features indicate the need for such properties. In a derivation, control features always come in pairs: [+, −]. A licensor feature, marked with [+], attracts an identical licensee feature, marked with [−].

4.1.2 Lexicon

A lexicon serves as a storage for features. Every lexical item has its own lexical specification, which is solely made up of features. All lexical items, apart from the functional categories, have *semantic* and *phonological* features. The use and the properties of words are defined by the syntactic category, selector and licensee features. *Licensor* features are mainly assigned to functional categories; they serve as triggers for the movement of phrases with matching *licensee* features.

Stabler (1999) presents the lexical specification as a list of feature occurrences. The sequence of features in the list determines the order in which the tree structure is built. Not every order of features is possible, the ordering depends on the application of the structure building operations. Admissible

orderings are determined by the following regular expression:

$$(=\mathtt{f}\ (=\mathtt{f})^\star\ (+\mathtt{f}))\ \mathtt{f}\ (-\mathtt{f})^\star\ /\mathtt{f}/\ \underline{\mathtt{f}}$$

Parentheses indicate an option of 0 or 1 occurrences, or more if decorated with a star. A category feature can stand by itself, it can be preceded by a certain number of selector features or it can be followed by licensee features. Only one licensor feature can appear before the category feature. A feature specification ends with the non-syntactic features in the case of the lexical categories; functional categories have no phonological feature information. With this feature information, we can built lexical entries. Fig. 4.2 shows a small sample of a lexicon.

```
n maria
n tortillas
=d =d v making
=n d the
=v +wh c
=n d -wh what
```

Figure 4.2: Minimalist lexicon: sample entries

4.1.3 Structure building operations

Structures $\mathcal{S}$ are built by concatenating, moving or abstracting lexical material within structures. A phrasal structure is represented with labeled binary trees. Instead of labeling the trees with the category of the head of the tree, as is done in the syntactic tradition, Stabler (1999) labels a tree with a direction arrow $\{<, >\}$ pointing towards the head. The leaves of the tree are the lexical feature structures $\mathcal{F}$, built up with features as described above.

$$\mathcal{S} \ ::= \ \mathcal{F} \mid \mathcal{S} \ < \ \mathcal{S} \mid \mathcal{S} \ > \ \mathcal{S}$$

Two operations are involved with building labeled structures:

1. MERGE: $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$
2. MOVE: $\mathcal{S} \rightarrow \mathcal{S}$.

MERGE combines two trees t_1 and t_2 to form a new tree. Tree $t_1[=\mathtt{c}]$[1], with first feature $=c$, combines with tree $t_2[\mathtt{c}]$ which carries feature category c. Technically, MERGE can be partitioned into two functions: one that combines with a tree on the right side: $(<, t_1, t_2)$, and one that combines with a tree on the left: $(>, t_2, t_1)$. Tree t_2 combines with t_1 on the right side if t_1 is a lexical item. Tree t_2 combines with t_1 as a specifier on the left side if t_1 is already a tree structure. Both selector and category feature are deleted after merging. Stabler (1999) formalizes MERGE as shown in the tree diagrams in Fig. 4.3

MOVE operates on the substructures of a tree. A licensor feature $[+\mathtt{f}]$ on the head of tree $t_1[+\mathtt{f}]$, attracts a subtree $t_2[-\mathtt{f}]^>$ with a corresponding licensee

[1] $t[\mathtt{F}]$ indicates that $\mathtt{F}$ is the prefixed feature of the feature structure of the head of tree t

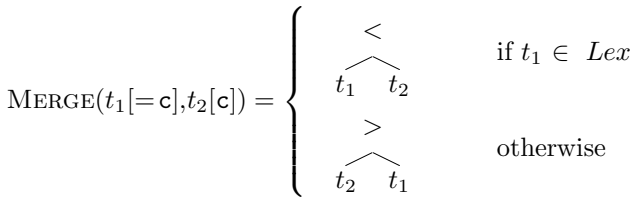


Figure 4.3: Definition of MERGE as a tree diagram

feature. The [–f] feature is found at the complement position $comp^+$ or in the specifier position $spec, comp^+$ of the head of the tree. $comp^+$ is the transitive closure on the binary relation *comp*, 'is a complement of', as given in the definition of *transitive closure*.

Transitive closure: Given any binary relation R (such as A *comp* B: "A is a complement of B"), the transitive closure R^+ is the smallest binary relation R^+ such that $R \subseteq R^+$ and xR^+y & $yR^+z \Rightarrow xR^+z$

$comp^+$ the transitive closure of the complement relation *comp*

$spec, comp^+$ the specifiers of trees in the transitive closure of the complement relation

Maximal projection: The maximal projection of subtree $t[-\mathtt{f}]^>$ is the largest subtree with [−f] as its head

The tree diagram in Fig. 4.4 defines the structure building operation MOVE. MOVE is applied to the *maximal projection* of the subtree carrying the licensee feature [–f]. After extracting the subtree from the main tree, the subtree is merged as a specifier to the head of the tree. Both control features are canceled and removed from the tree. The original occurrence is replaced by an empty tree, a single node without features.

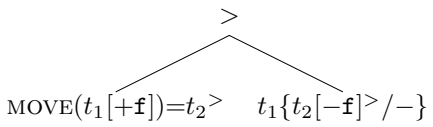


Figure 4.4: Definition of MOVE as a tree diagram

The definition of MOVE assumes some general constraints on movement, as given by Stabler (1999).

1. The whole feature structure moves, syntactic features as well as semantic and phonological feature information.

2. Movement can only apply if there is one outstanding [−f] feature.

3. Not more or less information can be moved than the maximal projection, $t[-\texttt{f}]^{>}$.

4. The moved tree must be a comp^{+} or the specifier of a comp^{+}

5. Both licensor and licensee features are canceled after movement.

The first constraint contrasts with an older version of Stabler's formalism that addresses the distinction between strong and weak control features. Under influence of the strong features the whole feature structure moves, while under influence of the weak features only syntactic features move. The second constraint is a strong version of the "shortest movement" condition (Chomsky, 1995), where movement cannot apply when two outstanding licensee features compete for the same position. The third and fourth constraints are proposals by Koopman and Szabolcsi (1998). They claim that all movement has to be XP-movement, where the XP is nothing more or less than the maximal projection of the lexical object carrying the attracted feature. The last constraint contrasts the distinction between interpretable and uninterpretable features. In the Minimalist Program only uninterpretable features are canceled. In Stabler's formalism both control features involved with the operation MOVE are canceled.

4.1.4 A derivation

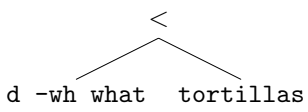
The structure building operations enable us to derive sentences on the basis of lexical specifications. As an illustration we derive the sentence "`What tortillas Maria making`" using the lexical specifications in Fig. 4.2. Because formal features such as tense and case, are not taken into account, the sentence is ill-formed with regard to the inflection of the verb.

Besides the lexical introduction steps, which introduce the lexical entries in the derivation, the structure is built with the four MERGE steps in (3), (5), (7) and (9). Step (10) is a crucial step in the derivation: the licensor feature [+`wh`] on the functional category [`c`] requires a licensee feature [−`wh`]. The lexical specification of `what` carries the required feature [−`wh`]. The maximal projection of t[−`wh`]$^{>}$ is the entire phrase `what tortillas`. Under the influence of the operation MOVE, the subtree is merged to the main tree as the specifier of the functional category [`c`]. The derivation ends here, since there are no outstanding licensor features, and the only syntactic feature is the category feature [`c`] on the head of the tree.

1. LEX: `=n d -wh what`

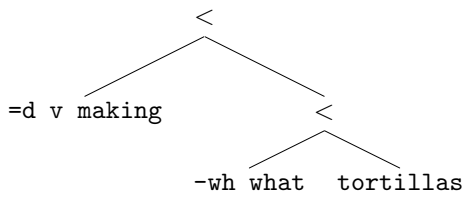
2. LEX: `n tortillas`

3. MERGE(1,2):



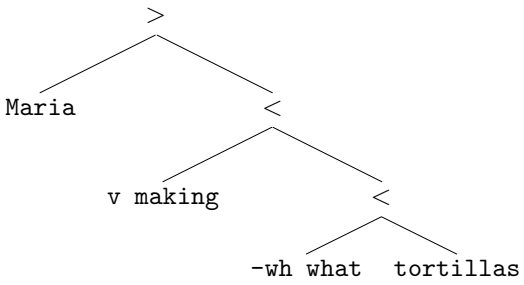
4. LEX: lex=d =d v making

5. MERGE(3,4):



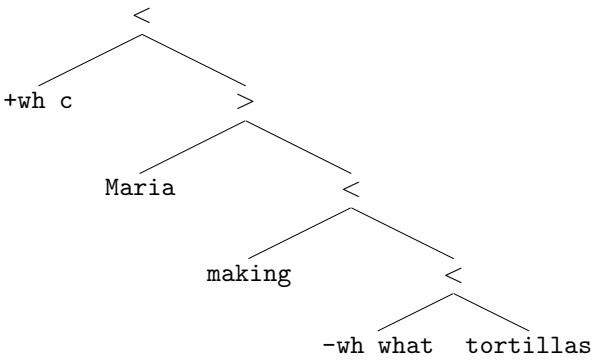
6. LEX: `d Maria`

7. MERGE(5,6):

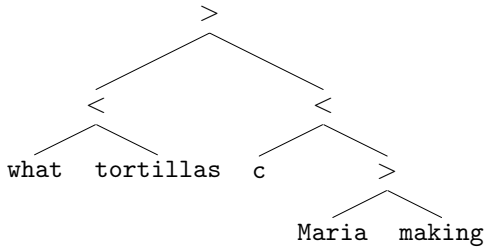


8. LEX: `=v +wh c`

9. MERGE(7,8):



10. MOVE(9):
[+wh] attracts [−wh]



4.1.5 Summary

Stabler's formalism shows that the interaction between syntactic features and structure building operations is straightforward; the feature specifications of the lexical items predict the phrasal structure of a sentence. Every step in a derivation is determined by the feature information on the head of the tree. A derivation fails if the head of the tree has no category, selector or licensor feature.

4.2 A mapping: MG and MMCG

In this section we relate Stabler's formalism to MMCG by mapping the features and operations of both systems.

4.2.1 Feature correspondence

In MMCG the semantic and syntactic features are described within the lexicon. The semantic part, which is not taken into account for now, is articulated in terms of Lambda calculus. The syntactic part is represented by the type-logical grammar. A lexicon exists of lexical entries with a structural label (the head-word) and the syntactic formula. As explained in chapter 2, the formulas are built up within a grammar of basic categories and modally decorated binary and unary connectives.

As we have seen in section 4.1.1 syntactic features in MG come in pairs: category-selector and licensee-licensor features. To give a correspondence between the feature specification in MG and type-logical formulas in MMCG, we need to be able to reason about different parts of a type-logical formula. In section 2.2 we have shown that expressions are given as $\Gamma \vdash A$. The antecedent Γ is the input, the assumptions, which has a certain type A. The type represents the output, also called the goal formula. We adopt a way of speaking about the different parts of type-logical formulas: the *polarity* of a (sub-)formula. The input receives a *positive* polarity and the output a *negative* polarity. On the basis of the polarity of the whole type-logical formula, the polarities of the subformulas can be derived with the following rule:

$$(B\backslash A)^p,\ (A/B)^p \rightsquigarrow A^p, B^{-p}$$

The following correspondence is made on the basis of the feature specifications of MG and the type-logical formulas of MMCG.

Kind of feature	MG	MMCG
Basic categories	c	c on the positive head of the formula
Selector features	=c	c on the negative subformula: $c\backslash_{>}$-, -/$_{<}c$
Licensee features	[−f]	$\Box_f$ on positive (sub-)formula
Licensor features	[+f]	$\Box_f$ on negative (sub-)formula

Figure 4.5: Feature correspondence

We take the same basic categories in MMCG for the categorial types as the categories in MG. The logical connectives, $\{/_{<}, \backslash_{>}\}$, left and right division have the same function as the selector feature, namely a request for a certain category. The right division is used to fill the complement position, the left division fills the specifier position.

In MG the control features, the licensors and the licensees, act as each other counterpart indicated by the polarities [+] and [−]. Following the polarities of the type-logical formulas, the licensee feature corresponds to the unary connective $\{\Box\}$ on a positive (sub-)formula; in practice the head of the formula will be decorated. The licensor feature, with an opposite polarity, corresponds to the unary connective on the negative formula, which generally means on a subformula or on the goal formula.

The licensor and licensee features enforce movement of features and therefore reordering of the structure. The function of the licensor feature as trigger for the rearrangement of features and lexical resources corresponds to the function of the unary connectives within MMCG. The licensor feature interacts with the licensee feature. In MMCG the licensor feature is defined as $\Box_f$ on the positive formula of a lexical entry, which interacts with the structural brackets $\langle . \rangle^f$ on the structural side. In both systems the licensor and licensee feature cancel each other. In MMCG the unary connectives play the role of 'key' and 'lock', where the diamond serves as 'key' and the box as 'lock' as shown in the following derivation rule:

$$\Diamond\Box A \longrightarrow A$$

4.2.2 Lexical correspondence

On the basis of the feature correspondence we can compute the lexical correspondence between the syntactic feature specifications in MG and the type assignments in MMCG. As described in section 4.1.2 the MG syntactic feature specification is described by the regular expression:

$$(=\mathtt{f}\ (=\mathtt{f})^\star\ (+\mathtt{f}))\ \mathtt{f}\ (-\mathtt{f})^\star$$

This regular expression can be transposed to the finite state automaton in Fig. 4.6. This automaton computes all the possible strings on the basis of the syntactic features (where $[\mathtt{f}, \mathtt{g}]$ represent the possible category and control features).

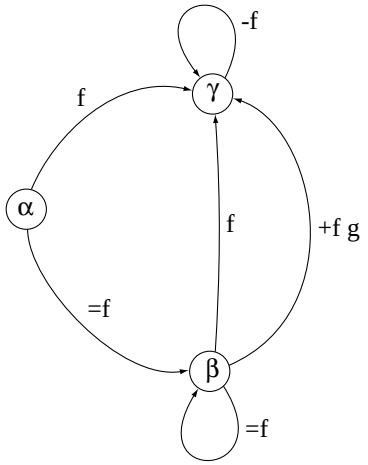


Figure 4.6: Finite state automaton to compute MG feature structures

On the basis of this automaton we build an algorithm that translates MG syntactic feature structures $\mathcal{F}$ into MMCG type formulas. The automaton in Fig. 4.6 corresponds with the rules on the left side of the algorithm in Fig. 4.7. In the algorithm, the three states: α, β, γ map parts of the feature specification to categorial formulas.

The phonological and semantic feature information of the lexical categories is incorporated into the label of the logical formula. MG also deals with functional

$$
\begin{array}{ll}
(\texttt{f}\ \mathcal{F})^{\alpha} & = (\mathcal{F}\ f)^{\gamma} \\
(=\texttt{f}\ \mathcal{F})^{\alpha} & = (\mathcal{F})^{\beta}/_{<} f \\
(=\texttt{f}\ \mathcal{F})^{\beta} & = f\backslash_{>}(\mathcal{F})^{\beta} \\
(\texttt{f}\ \mathcal{F})^{\beta} & = (\mathcal{F})^{\gamma}\ f \\
(+\texttt{f}\ \texttt{g}\ \mathcal{F})^{\beta} & = (\mathcal{F})^{\gamma}\ g \text{ with } \square_f\ g \text{ as goal formula or subformula} \\
(-\texttt{f}\ \mathcal{F})^{\gamma} & = (\mathcal{F})^{\gamma}\ \square_f \\
(\ )^{\gamma} & = -
\end{array}
$$

Figure 4.7: Algorithm to calculate categorial formulas from MG feature specifications

categories, which have no non-syntactic feature information. MMCG labels functional categories with their category feature to indicate their function and position in a structure.

Fig. 4.8 shows an example of the way the algorithm computes the categorial formula from the MG feature specification of the transitive verb `made`.

$$
\begin{array}{l}
(=\texttt{d}\ =\texttt{d v}\ -\texttt{past made})^{\alpha} \\
\textsf{made} \vdash (=\texttt{d}\ =\texttt{d v}\ -\texttt{past})^{\alpha} \\
\textsf{made} \vdash (=\texttt{d v}\ -\texttt{past})^{\beta}/_{<} d \\
\textsf{made} \vdash (d\backslash_{>}(\texttt{v}\ -\texttt{past})^{\beta})/_{<} d \\
\textsf{made} \vdash (d\backslash_{>}(-\texttt{past})^{\gamma}\ v)/_{<} d \\
\textsf{made} \vdash (d\backslash_{>}(\ )^{\gamma}\square_{past} v)/_{<} d \\
\textsf{made} \vdash (d\backslash_{>}\square_{past} v)/d_{<}
\end{array}
$$

Figure 4.8: Feature translation on the basis of the vertical function

Using the algorithm in Fig. 4.7 we can translate the MG lexicon in Fig. 4.2 into the categorial lexicon in Fig. 4.9.

MG	CG
`n tortillas`	$\textsf{tortillas} \vdash n$
`d Maria`	$\textsf{Maria} \vdash d$
`=n d the`	$\textsf{the} \vdash d/_{<} n$
`=d =d v making`	$\textsf{making} \vdash (d\backslash_{>} v)/_{<} d$
`=n d -wh what`	$\textsf{what} \vdash \square_{wh} d/_{<} n$
`=v +wh c`	$c/_{<} v$ with $\square_{wh} c$ as goal-formula

Figure 4.9: Lexical correspondence

4.2.3 Mapping of operations

Merge as application

Compare the operation MERGE as described in section 4.1.3, with the application rules in MMCG. The rules of application are defined in the natural deduction proof system by the elimination rules $\{/_<, \backslash_>\}$. MG defines MERGE as a structure building operation, which combines two tree structures t_1 which head carries a selector feature: $[= \mathtt{A}]$ and t_2 which head carries a corresponding category feature: $[\mathtt{A}]$. The operation MERGE causes the cancellation of the feature $[= \mathtt{A}]$ against $[\mathtt{A}]$. As tree t_1 can select both to the right and to the left, MERGE can be split into two operations: $\textsc{Merge}_<$ and $\textsc{Merge}_>$.

In MMCG MERGE is captured by the elimination rules of the binary connectives: $\{/_<, \backslash_>\}$. Fig. 4.10 shows both structure building rules and the matching elimination rules: MERGE on the right as complement $[/_<E]$ and MERGE on the left as specifier $[\backslash_>E]$. As the minimalist operation MERGE can be split into two operations: $\textsc{merge}_<$ and $\textsc{merge}_>$, the same holds for the logical operations in MMCG. Fig. 4.10 shows both directions of the elimination rules: left-headed MERGE and right-headed MERGE. In minimalism one prefers to reason about one MERGE operation, which is the union of both separated operations:

$$\textsc{Merge} = \textsc{Merge}_< \cup \textsc{Merge}_>$$

Leftheaded MERGE:

$$\textsc{Merge}_<(t_1[=\mathtt{A}],t_2[\mathtt{A}]) \Rightarrow \overset{<}{\underset{t_1 \quad t_2}{\wedge}}$$

$$\frac{t_1 \vdash B/_<A \quad t_2 \vdash A}{t_1 \circ_< t_2 \vdash B}\ [/_<E]$$

Rightheaded MERGE:

$$\overset{>}{\underset{t_1 \quad t_2}{\wedge}} \Leftarrow \textsc{Merge}_>(t_2[=\mathtt{A}],t_1[\mathtt{A}])$$

$$\frac{t_1 \vdash A \quad t_2 \vdash A\backslash_>B}{t_1 \circ_> t_2 \vdash B}\ [\backslash_>E]$$

Figure 4.10: MERGE as application

In the Minimalist Grammar direction arrows: $<$ and $>$ indicate the head of the tree. In MMCG indication of the head is implicitly done with a functional projection; a lexical item with category A/B is applied to a lexical item with category B to form a phrase of category A. The first lexical item is the function which takes an argument. In this way the function serves as a head, while the argument serves as a complement or a specifier. By explicitly marking the binary connectives, the 'arrow' indicates the head of two combined entries defining the dependency relation between words. For both directions the arrows $<$ and $>$ are added as modes to the structural combinator $\circ$. The most common direction is towards the head of the formula.

Move as structural reasoning

To capture the right behavior of the movement operation, the attraction of the licensee feature by the licensor feature needs to be accounted for. The translation of the licensor feature as a feature decorated $\Box$ on a negative formula, enforces the interaction with the licensee feature via the control diamond, $\langle . \rangle$ on the structural side. The structural diamond influences the use of structural

postulates that capture the possible movements of a phrase from a certain position in the structure to another. The postulates in Fig. 4.11 can be regarded as general postulates that define movement.

$$\begin{array}{rl} \Diamond_f(A \bullet_> B) \rightarrow \Diamond_f A \bullet_> B & [P1] \\ \Diamond_f A \bullet_> (B \bullet_i C) \rightarrow B \bullet_i (C \bullet_< \Diamond_f A) & [P2] \\ \Diamond_f A \bullet_> (B \bullet_i C) \rightarrow B \bullet_i (\Diamond_f A \bullet_> C) & [P3] \end{array}$$
Where $i \in \{<, >\}$

Figure 4.11: Movement postulates

The first postulate $[P1]$ checks if there is a phrase carrying a certain feature f at the first position. In languages where a word is located at another position, this postulate could be deleted or changed in such a way that it distributes to the expected place in the sentence. The second and third postulates percolate a feature decorated phrase through a structure. $[P2]$ moves a phrase A from a complement position to the specifier position of that same phrase headed by B or C. $[P3]$ moves a phrase A from the specifier position to a higher specifier position of a phrase headed by B or C.

As an example we show how these postulates are used in sentences where wh-movement occurs. For this case, the undefined feature f in the above postulates is specified as a *wh*-feature and therefore we rename the postulates to $[Pwh1, Pwh2, Pwh3]$:

$$\begin{array}{rl} \Diamond_{wh}(A \bullet_> B) \rightarrow \Diamond_{wh} A \bullet_> B & [Pwh1] \\ \Diamond_{wh} A \bullet_> (B \bullet_> C) \rightarrow B \bullet_> (C \bullet_< \Diamond_{wh} A) & [Pwh2] \\ \Diamond_{w} hA \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (\Diamond_{wh} A \bullet_> C) & [Pwh3] \end{array}$$

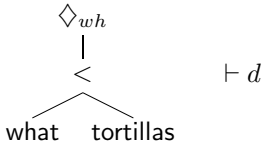
With the use of the lexical formulas in Fig. 4.9 we derive the structure "What tortillas Maria making". The structural part of this derivation is represented with binary trees. In *natural deduction* these trees are given in a flat structure, but the structural part can also be presented as a full binary tree. Both presentations are given.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{what} \vdash \Box_{wh} d/_< n \quad \mathsf{tortillas} \vdash n}{\mathsf{what} \ \circ_< \ \mathsf{tortillas} \vdash \Box_{wh} d}[/_< E]}{\langle \mathsf{what} \ \circ_< \ \mathsf{tortillas} \rangle^{wh} \vdash d}[\Box_{wh} E]}{\mathsf{making} \vdash (d \backslash_> v)/_< d \quad \langle \mathsf{what} \ \circ_< \ \mathsf{tortillas} \rangle^{wh} \vdash d}[/_< E]}{\mathsf{Maria} \vdash d \quad \mathsf{making} \ \circ_< \ \langle \mathsf{what} \ \circ_< \ \mathsf{tortillas} \rangle^{wh} \vdash d \backslash_> v}[\backslash_> E]}{\mathsf{c} \vdash c/_< v \quad \mathsf{Maria} \circ_> (\mathsf{making} \circ_< \langle \mathsf{what} \circ_< \mathsf{tortillas} \rangle^{wh}) \vdash v}[/_< E]}{\mathsf{c} \circ_< (\mathsf{Maria} \circ_> (\mathsf{making} \circ_< \langle \mathsf{what} \circ_< \mathsf{tortillas} \rangle^{wh})) \vdash c}[Pwh2]}{\mathsf{c} \circ_< (\langle \mathsf{what} \circ_< \mathsf{tortillas} \rangle^{wh} \circ_> (\mathsf{Maria} \circ_> \mathsf{making}) \vdash c}[Pwh3]}{\langle \mathsf{what} \circ_< \mathsf{tortillas} \rangle^{wh} \circ_> (\mathsf{c} \circ_< (\mathsf{Maria} \circ_> \mathsf{making})) \vdash c}[Pwh1]}{\langle (\mathsf{what} \circ_< \mathsf{tortillas}) \circ_> (\mathsf{c} \circ_< (\mathsf{Maria} \circ_> \mathsf{making})) \rangle^{wh} \vdash c}[\Box_{wh} I]}{(\mathsf{what} \ \circ_< \mathsf{tortillas}) \circ_> (\mathsf{c} \circ_< (\mathsf{Maria} \circ_> \mathsf{making})) \vdash \Box_{wh} c}$$

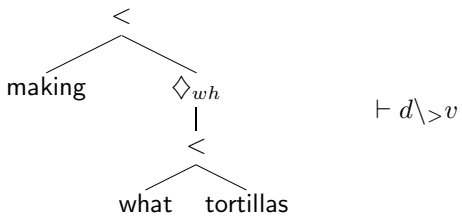
Figure 4.12: Natural deduction derivation of What tortillas Maria making with flat structures

1. what $\vdash \Box_{wh} d/_{<} n$
2. tortillas $\vdash n$
3. $[/_{<}E]$, MERGE(1,2):

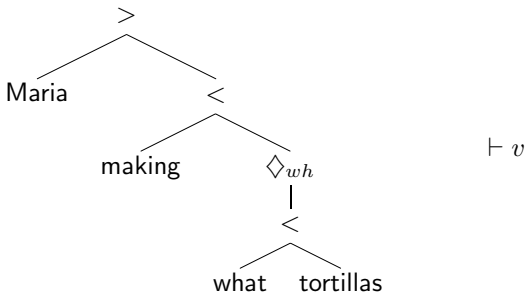
4. $[\Box_{wh} E]$



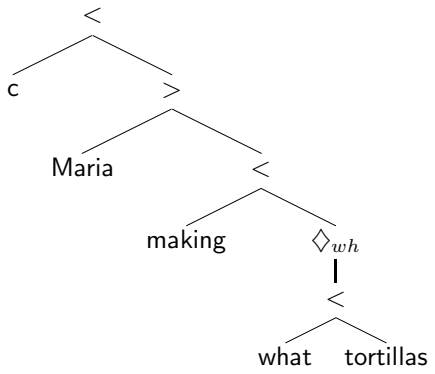
5. making $\vdash (d\backslash_{>} v)/_{<} d$
6. $[/_{<}E]$, MERGE(5,4):



7. Maria $\vdash d$
8. $[\backslash_{>} E]$, MERGE(6,7):

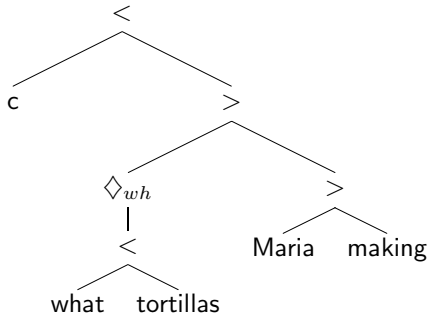


9. $c \vdash c/_{<} v$
10. MERGE(9,8):



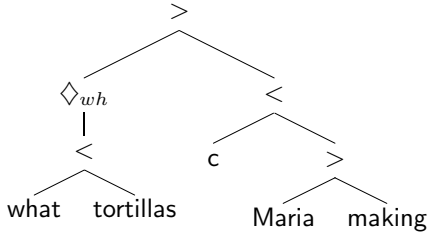
$\vdash c$

11. MOVE(10) by Pwh2:



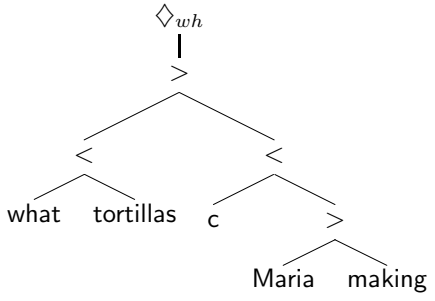
$\vdash c$

12. MOVE(11) by Pwh3



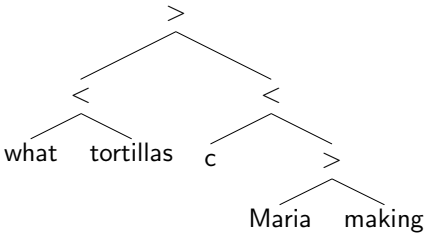
$\vdash c$

13. MOVE(12) by Pwh1



$\vdash c$

14. $\Box_{wh}$I:



$\vdash \Box_{wh}c$

If you compare the tree structure representation with the MG derivation in section 4.1.4, you can see a few resemblances and some differences. The representation is different in the way the structural information is separated from the logical part. Whereas in Stabler's formalism syntactic, semantic and phonological information is used within one representation, the MMCG tree representation separates these components. The left-hand side of the turnstile shows the hierarchical order of the structure while the right-hand side gives information on the category and the features of the whole phrase. The structural brackets, $\langle . \rangle$, projected from the logical side, are the structural domains of phrases where the postulates can work on.

The order of application of the structure building operations is the same in both systems. First a number of MERGE steps, followed by the movement of "what tortillas". In MG the operation MOVE is triggered from the licensor feature on the functional category `c`. In MMCG the feature information of the functional category is split up: the category features are assigned to the lexical entry `c` and the licensor feature is part of the goal formula. To check the 'licensee' feature on the lexical entry `what` against the 'licensor' feature on the goal formula, the control features trigger the use of structural postulates.

Using structural reasoning we can capture the movement operation in MMCG. With structural postulates we explicitly define the steps that a phrase has to make to arrive at a certain point in the structure. In MG, movement is done implicitly by abstracting a phrase from its former position, which is determined by the maximal projection, and by moving the phrase up to the specifier position.

Constraints on Move

As explained in section 4.1, the definition of minimalist movement assumes some constraints. The constraints on movement as given in section 4.1.3, are repeated here. Are they respected by the translation of MOVE as structural reasoning?

1. The whole feature structure moves, syntactic features as well as semantic and phonetic feature information.
2. Movement can only apply if there is one outstanding $[-\mathtt{f}]$ feature.
3. Not more or less information can be moved than the maximal projection of $t[-\mathtt{f}]$.
4. The moved tree must be a comp$^+$ or the specifier of a comp$^+$
5. Both licensor and licensee features are deleted after movement.

The first constraint, that the whole feature specification has to move, is met with regard to formal and phonological features. The postulates that force movement, $[Pwh2]$ and $[Pwh3]$, move the lexical structures as a whole. The other postulate, $[Pwh1]$, must not be mistaken as feature movement, although it searches through the structure to reach the placeholder of the syntactic feature. The semantic part of the derivation is done with lambda calculus. This part is not sensible to structural changes; it only captures the logical part of the derivation. Section 4.2.4 discusses the role of derivational semantics.

The second constraint, that movement can only apply if there is one outstanding licensee feature, is met under one extra condition. No structural feature domain is allowed at the end of the derivation. The structural postulates projects the licensor feature ($\Box$ on the goal formula) on the structural part of the derivation, such that it occurs as feature domain over the selected structure. The $\Box$ feature on the goal formula can cancel only one occurrence of $\Box$ coming from the lexical domain. A structure with two outstanding structural feature domains (which is the result of two $\Box$ features in the lexical domain) would be rejected because only one of these feature domains could be checked by the logical connective on the goal formula. So movement can apply when there are two outstanding licensee features, but the derivation is rejected.

For the third constraint, that the maximal projection of the licensee feature moves, maximal projection needs to be defined within MMCG. Two options are available, the first is already given with the definition of the lexical structures. As a result of defining what as $\Box_{wh} d/_{<} n$ the domain of the structural brackets will extend over the noun tortillas. The domain of structural brackets is exactly the maximal projection of the head of the formula.

Another option is to put the feature information as a box on the whole positive formula: $\Box_{wh}(d/n)$. In this case you need to define the maximal projection of the feature information. This can be done with the following two postulates which distribute the structural brackets over the specifier of the head or over the complement.

$$\Diamond_{wh}(A \bullet_{<} B) \rightarrow \Diamond_{wh} A \bullet_{<} B \quad [K1wh]$$
$$\Diamond_{wh}(A \bullet_{>} B) \rightarrow A \bullet_{>} \Diamond_{wh} B \quad [K2wh]$$

The first option seems an obvious choice, because no extra postulates are needed. But in case more licensor features are used, there is the possibility that one licensor feature blocks the other one by enclosing the other feature in its domain. The feature domain blocks the inner phrase from moving and checking its own licensee feature. There are cases where this would even be useful in order to prevent phrases from moving.

The last constraint, which prescribes the deletion of both control features, follows the interaction between the logical and the structural part of a derivation. The introduction and elimination rules of the unary connective $\Box_f$ in Fig. 4.13 give us a way of reasoning that corresponds with the behavior of the licensor and licensee features. The elimination step can be seen as the projection of the licensee feature over the required phrase. The introduction of $\Box_f$ can be read as the checking of a licensee feature against a licensor feature.

$$\frac{\Gamma \vdash \Box_f A}{\langle \Gamma \rangle^f \vdash A}\ [\Box_f E] \qquad \frac{\langle \Gamma \rangle^f \vdash A}{\Gamma \vdash \Box_f A}\ [\Box_f I]$$

Figure 4.13: Control features: ND-rules

4.2.4 Move as abstraction operation

The previous section showed how MOVE is defined in terms of structural control. Using structural reasoning one captures the actual movement of words in a sentence. As a mechanism which describes phenomena of "displacement", MOVE should be regarded as a complex operation, which has both a structural and a logical component. With the use of hypothetical reasoning as the logical component, one is able to account for the meaning of MOVE as an abstraction operation.

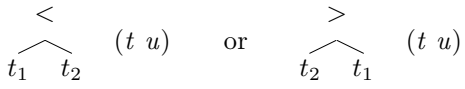
This section explores the meaning of the MOVE operation in both MG and MMCG. In MMCG proof terms indicate the meaning of derivations (Moortgat, 1996) by means of the Curry-Howard morphism. This notion can be extended to MG and the resulting proof term shows that the meaning MOVE as abstraction operation is captured.

Derivational Meaning

As explained in section 2.5 derivations can be decorated with terms. A proof term is the 'blueprint' of the logical derivation: all logical steps can be read from the proof term. In the same way as the logical rules in MMCG can be decorated, the structure building operations in MG can be labeled with proof terms. The result gives us a comparison between the MMCG proof system and the derivational formalism of Stabler (1999).

In his notes Stabler (1998) suggests what semantic values should be assigned to the operations MERGE and MOVE. A first idea is to interpret MERGE as *application* and MOVE as *abstraction*.

Merge as application To capture the meaning of MERGE as application, the accompanying term should be defined as follows:
MERGE(t_1[=c],t_2[c]), where t_1 is labeled with t and t_2 with u.



Move as abstraction To capture the meaning of MOVE is more complicated. MOVE consists of two operations: *abstraction* and *application*, which need to be addressed in the proof term. The definition as stated in Fig. 4.4 is repeated in Fig. 4.14.

The two main tree structures are labeled with semantic terms:

- t_1[+f] is the whole tree labeled with term t

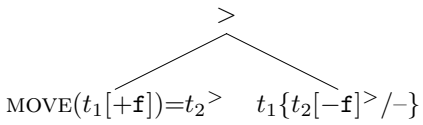


Figure 4.14: Move in MG

- $t_2[-\mathtt{f}]^{>}$ is the maximal projection of $t_2[-\mathtt{f}]$ labeled with u

Fig. 4.15 shows how the MOVE operation is decorated with terms. Tree $t_2[-\mathtt{f}]^{>}$, labeled with term u, is a subtree of tree $t_1[+\mathtt{f}]$. The whole tree is labeled with $t\{u\}$, where u is a subterm of term t. During movement, $t_2[-\mathtt{f}]^{>}$ is abstracted from $t_1[+\mathtt{f}]$ yielding the proof term: $\lambda x.t\{x/u\}$. The variable x replaces term u indicating the trace of the extracted subtree t_2. At the same time, the extracted tree t_2 is merged to tree $t_1\{t_2[-\mathtt{f}]^{>}/-\}$ yielding the tree (>, t_2, t_1) with proof term: (u $\lambda x.t\{x/u\}$).

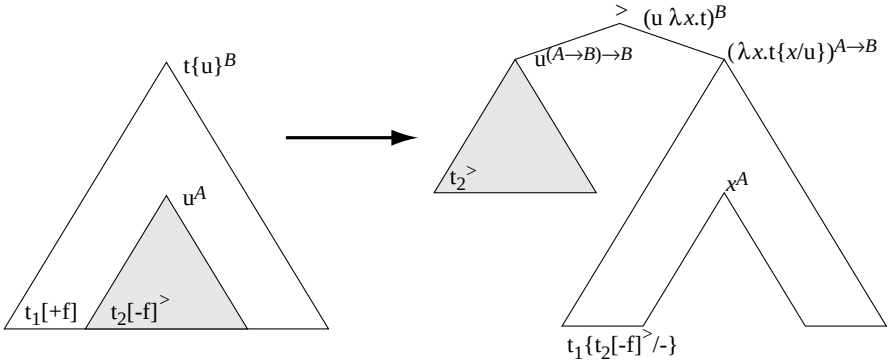


Figure 4.15: Schematic representation of term decorated MOVE

Every term belongs to a certain type. In section 2.5 we have seen how terms are built on the basis of their types. With the use of types, we can check whether a structure is well-typed. To check if the two structures of the definition of MOVE are well-typed, we need to look at the types that belong to the different terms. The types are given as exponents of the terms.

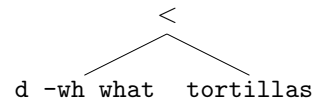
The whole tree t_1 with term t is of type B and the subtree t_2 with term u is of type A. After abstraction the whole tree (>, t_2, t_1) still has to be of type B. Tree t_1, where the subtree of type A has been abstracted from, gets assigned the type $A \to B$. Then the subtree is merged to tree t_1 (t_2 is applied to t_1). To yield a tree of type B, the type of tree t_2 has to be $(A \to B) \to B$. But then there is a *type-clash* between the type of term u in the first structure and the type in the second structure.

We can overcome this type-clash by reasoning hypothetically over the abstracted subtree. One uses term variable x to indicate the position of the abstracted tree structure. The higher order type assigned to tree t_2 accomplishes the abstraction of this subtree. In this way one prevents the type-clash between

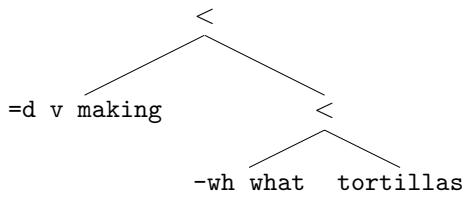
the two occurrences of the subtree. After an example of decorating a minimalist derivation, I will show how MMCG captures the abstraction component of MOVE with hypothetical reasoning.

Minimalist derivation, term decorated As an example, we derive the proof term for the sentence "`What tortillas Maria making`" as given in section 4.1.4. Step by step the derivation is decorated with the derivational meaning. For simplicity the semantic terms assigned to the lexical items correspond to the headword of the feature structure.

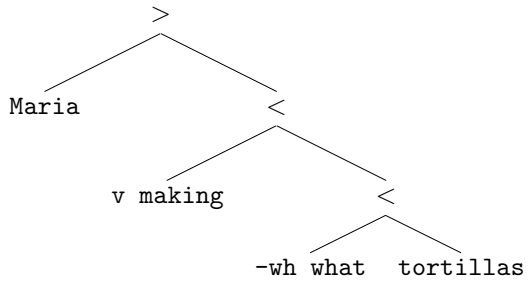
1. `=n d -wh what` ⇒ **what**
2. `n tortillas` ⇒ **tortillas**
3. MERGE(1,2):



   ⇒ (**what tortillas**)
4. `=d =d v making` ⇒ **making**
5. MERGE(4,3):

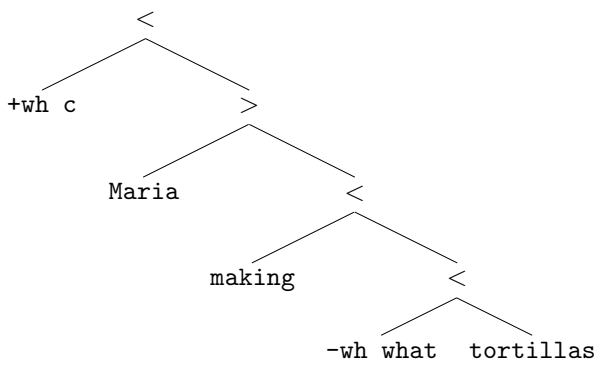


   ⇒ (**making** (**what tortillas**))
6. `d Maria` ⇒ **Maria**
7. MERGE(5,6):



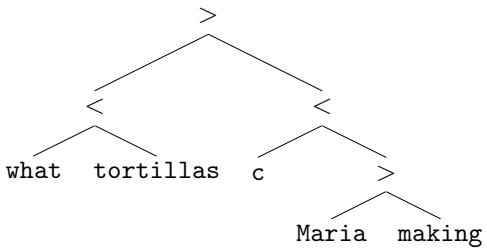
   ⇒ ((**making** (**what tortillas**)) **Maria**)
8. `=v +wh c`

9. MERGE(8,7):



⇒ ((**making** (**what tortillas**)) **Maria**)

10. MOVE(9):



⇒ ((**what tortillas**) $\lambda x.$((**making** x) **Maria**)

The result of decorating the derivation with a proof term in this way, is that the *abstraction* can be read from the derivational meaning. The proof term, ((**what tortillas**) $\lambda x.$((**making** x) **Maria**), can now be compared with the proof term from the derivation in MMCG with structural reasoning (section 4.2.3). The structural rules are not captured within the proof term, so the elimination rules are the only logical steps that are shown in the proof term. The derivational meaning of the derivation using structural reasoning becomes: ((**making** (**what tortillas**)) **Maria**). This proof term does not show any trace of abstraction as a result of the operation MOVE. Moreover the proof term deviates largely from the proof term that results after decorating the Minimalist Derivation.

In section 4.2.3 movement is translated solely by using structural postulates. But looking at the derivational meaning, it becomes clear that this cannot be the right translation between Stabler's formalism and MMCG with regard to the MOVE operation.

MMCG: Move as abstraction

The other option to define movement within MMCG comes from the possibility of reasoning hypothetically about MERGE. In Stabler's definition of MOVE abstraction occurs: a subtree is abstracted from the whole structure. With the use of *hypothetical reasoning* this phenomenon can be translated in MMCG. As explained in chapter 2, hypothetical reasoning is defined by the introduction rules.

The proof term, built during the application of the introduction rules, captures the abstraction of a phrase out of a fully built phrase. Hypothetical reasoning is triggered from the lexicon. In our fragment the lexical entry what

$$\frac{\Gamma \circ x : B \vdash t : A}{\Gamma \vdash \lambda x.t : A/_{<}B}\ [/_{<}I] \qquad \frac{x : B \circ \Gamma \vdash t : A}{\Gamma \vdash \lambda x.t : B\backslash_{>}A}\ [\backslash_{>}I]$$

Figure 4.16: $/_{<}, \backslash_{>}$ Introduction rules

projects a hypothetical determiner phrase. In MMCG the information on the functional category and the feature information on the lexical item are combined in the type-logical formula of the lexical item. The formula assigned to what incorporates the function of the functional category c as a trigger of MOVE and its own lexical function as a determiner of nouns. The lexical translation for what is given in Fig. 4.17.

$$\left.\begin{array}{l}\texttt{=v +wh c}\\ \texttt{=n -wh d what}\end{array}\right\} (c/_{>}(\Diamond_{wh}\Box_{wh}d\backslash_{>}v))/_{<}n$$

Figure 4.17: Lexical type assignment for what from MG feature structures

This formula can be read as: after combining with a noun phrase, the 'higher order formula' indicating a determiner phrase looks for a verb phrase, which is missing an object phrase. Then the determiner phrase merges with the incomplete verb phrase into a complementizer phrase. The formula is constructed in such a way that the phrase what tortillas is still regarded as a specifier of the verb phrase making tortillas. The connective $\{/_{>}\}$ carries a direction arrow pointing towards the argument, to indicate that the argument will be considered as the head of the structure.

In the translation of what as $(c/_{>}(\Diamond_{wh}\Box_{wh}d\backslash_{>}v))/_{<}n$ the wh-feature on the $\Diamond_{wh}$ acts as the licensor, the trigger of the movement steps. The $\Box_{wh}$ on the positive subformula d indicates the licensee feature which allows the determiner phrase to be moved. In this translation, the feature correspondence given in Fig. 4.5 still holds. The licensee feature corresponds to the $\Box_{wh}$ assigned to a positive subformula of the logical formula. The licensor feature corresponds to the $\Diamond_{wh}$ on the positive subformula d, which is the counterpart of a $\Box_{wh}$ on a negative subformula.

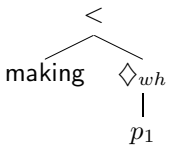
Apart from the lexical entry what and functional category c, all other entries stay the same as given in Fig. 4.9. The categorial feature [c] and the selection properties of the functional category are captured within the logical formula of what. Therefore the functional category as a lexical entry is no longer needed.

As an example we derive the sentence "what tortillas Maria making" with the accompanying proof term. The structural part of the derivation on the left-hand side is presented as a tree representation, followed by the term decoration and the type-logical formula on the right-hand side.

1. $p_1 \ \vdash y : \Box_{wh} d$ *(hypothesis)*
2. $[\Box_{wh} E]$:

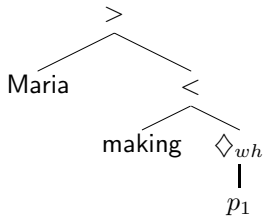
$\vdash {}^{\vee} y : d$

3. making $\vdash$ **making** : $(d\backslash_{>} v)/_{<} d$
4. $[/_{<} E]$, MERGE(3,2):



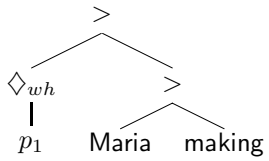
$\vdash$ (**making** ${}^{\vee} y$) : $d\backslash_{>} v$

5. Maria $\vdash$ **Maria** : d
6. $[\backslash_{>} E]$, MERGE(4,5):



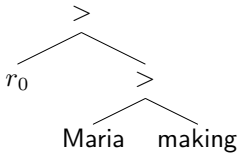
$\vdash$ ((**making** ${}^{\vee} y$) **Maria**) : v

7. MOVE(6) by Pwh2:



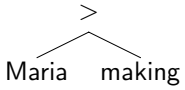
$\vdash$ ((**making** ${}^{\vee} y$) **Maria**) : v

8. $[\Diamond E]$:



$\vdash$ ((**making** ${}^{\vee\cup} x$) **Maria**) : v

9. $[\backslash_{>} I]$:

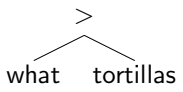


$\vdash \lambda x.((\textbf{making}\ {}^{\vee\cup} x))\ \textbf{Maria}) : \Diamond_{wh} \Box_{wh} d\backslash_{>} v$

10. what $\vdash$ **what** : $(c/_{>}(\Diamond_{wh} \Box_{wh} d\backslash_{>} v))/_{<} n$

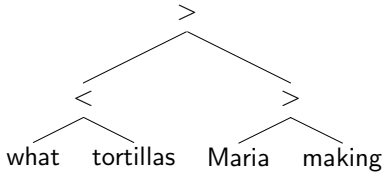
11. tortillas $\vdash$ **tortillas** : n

12. $[/_{<}E]$, MERGE(10,11):



$\vdash$ (**what tortillas**) : $c/_{>}(\Diamond_{wh}\Box_{wh}d\backslash_{>}v)$

13. $[/_{>}E]$, MERGE(12,9):



$\vdash$ ((**what tortillas**) $\lambda x.$((**making** $^{\vee\cup}x$) **Maria**)) : c

The movement of the hypothesized object is still accomplished by means of structural reasoning. The two postulates that were needed in the structural fragment for moving a phrase out of its specifier or its complement position are still the same. Postulate $[Pwh1]$, responsible for the search of the moved element and the feature checking, is no longer needed. Postulate $[Pwh2]$ accounts for the actual movement in this derivation, in order to retract the hypothesized object.

4.2.5 Lexical meaning

The question now is: Is the proof term belonging to this derivation the equivalent of the proof term from the Minimalist derivation in section 4.2.4? The difference between the two proof terms is accomplished by the use of the unary connectives. Otherwise both proof terms show the movement of what tortillas by leaving a 'trace' at the object position of the verb making.

To level both proof terms *lexical semantics* needs to be taken into account. Every logical rule has a corresponding operation on the proof term side; the logical rules labeled with terms are given in Fig. 2.10 for the binary connectives and Fig. 2.11 for the unary connectives. Every lexical item is decorated with a term, a semantic label. The semantic label gives the meaning of the logical information stated in the formula. Every logical connective has a semantic counterpart, which is stated in the label of the lexical item. As an example of a lexical item such as making, the semantic term is built up as follows:

$$\text{making} \vdash \lambda x.\lambda y.((\textbf{making}\ x)\ y) : (d\backslash_{>}v)/_{<}d$$

In this way every lexical item is decorated with a semantic label. Doing so it is possible to decorate what in such a way that the result of the lexical labeling leads to an identical proof term. The semantic label follows exactly the logical formula assigned to what. The semantic label of what becomes:

$$\text{what} \vdash \lambda x.\lambda y.((\textbf{what}\ x)\ \lambda z.(y\ ^{\cap\wedge}z)) : (c/_{>}(\Diamond_{wh}\Box_{wh}d\backslash_{>}v))/_{<}n$$

The lexical semantics is applied to the derivational semantics. Using β-reduction the proof term can be reduced:

$$\begin{array}{rcl} (\lambda x.t)\ u & \rightsquigarrow & t\{u/x\} \\ {}^{\vee\wedge}t & \rightsquigarrow & t \\ {}^{\cup\cap}t & \rightsquigarrow & t \end{array}$$

The resulting proof term is the same as the proof term deduced from the minimalist derivation in section 4.2.4. For convenience the variable x of the proof term is renamed to x_1.

$((\lambda x.\lambda y.((\textbf{what}\ x)\ \lambda z.(y\ {}^{\cap\wedge}z))\ \textbf{tortillas})\ \lambda x_1.((\textbf{making}\ {}^{\vee\cup}x_1)\ \textbf{Maria}))$
$\rightsquigarrow_\beta \quad (\lambda y.((\textbf{what tortillas})\ \lambda z.(y\ {}^{\cap\wedge}z))\ \lambda x_1.((\textbf{making}\ {}^{\vee\cup}x_1)\ \textbf{Maria}))$
$\rightsquigarrow_\beta \quad ((\textbf{what tortillas})\ \lambda z.((\textbf{making}\ {}^{\vee\cup\cap\wedge}z)\ \textbf{Maria}))$
$\rightsquigarrow_\beta \quad ((\textbf{what tortillas})\ \lambda z.((\textbf{making}\ {}^{\vee\wedge}z)\ \textbf{Maria}))$
$\rightsquigarrow_\beta \quad ((\textbf{what tortillas})\ \lambda z.((\textbf{making}\ z)\ \textbf{Maria}))$

4.3 Hungarian verb movement

This section explores another linguistic phenomenon to illustrate the correspondence between Multimodal Categorial Grammar and Minimalist Grammar.

4.3.1 Verbal complexes in Hungarian

Koopman and Szabolcsi (1998) explore the phenomenon of Hungarian verbal complexes. *Verbal complexes* appear in so-called neutral and non-neutral sentences. Neutral sentences are sentences without focused or negated phrases in it. The verb complexes are clusters of verbs, that are formed by two distinct *verb complex formation* processes. Before describing the different verbal complex formation procedures, the different components involved (verb modifiers and auxiliaries) are explored.

Verb modifiers (VM) Verb modifiers modify certain verbs by forming a verbal complex with them. Koopman and Szabolcsi (1998) distinguish three classes of verb modifiers.

1. prefixes (bare modifiers):
   - `haza` ('home') as in `haza menni` ('to go home'),
   - `be` ('in') as in `be menni` ('to go in'),
   - `el` ('away') as in `el menni` ('to go away')
2. bigger modifiers:
   - `a szobá-ba` ('into the room') as in `a szobá-ba menni` ('to go into the room')
3. infinitives:
   - `úsz-ni` ('swim') as in `úsz-ni akarni` ('to want to swim')

Prefixes are words that are placed in front of another word, in this case a verb. In contrast with bigger modifiers, prefixes are simple, single affixes. The bigger modifiers show a similar behavior as the prefixes attached to certain verbs; the behavior depends on the kind of verb that selects them. The last kind of modifier are infinitives, which optionally climb to front their selecting verb.

Auxiliaries Koopman and Szabolcsi (1998) define auxiliaries as verbs that select infinitival verbs as complements. The finite auxiliaries never carry the main accent in a sentence. Examples of Hungarian auxiliaries are: `akar` ('want'), `fog` ('will') and `kezd` ('begin').

4.3.2 Verbal complex formation

Koopman and Szabolcsi (1998) distinguish two verbal complex formation processes: *verb modifier climbing* (VM-climbing) and *verbal inversion*. The first process only occurs in neutral sentences and describes the climbing of the verb

modifier to precede the finite verb. The second process describes the recursive inversion of infinitival verbs in non-neutral sentences with focus or negative phrases. The application of a formation process depends on the kind of verbs and modifiers that form the context of the sentence.

In this section the two formation processes are elaborated. First one needs to consider what triggers this formation. The triggers have to be anchored in the lexicon. The components of a verbal complex, the verb modifier or the selecting verb, have to carry a lexical requirement. The verb modifiers of the first class, the prefixes, will always form a verbal complex, so it should be a lexical requirement on these prefixes. For the other two classes of verbal modifiers, the complex formation depends on the verb with which they form a verbal complex. In these cases there should be a lexical requirement on the selecting verb, which selects a verb modifier. Some verbs, on the contrary, block the formation of a verbal complex.

Verb modifier climbing

Verb modifier climbing is the process whereby a verb modifier procliticizes to the finite verb (either a selecting verb or an auxiliary). This type of complex formation process only appears within neutral sentences and under the influence of a certain kind of verb modifier. For the verbal modifiers of the third class, the infinitives, VM-climbing is optional. For the verb modifiers of the second class it is again obligatory.

Example (4.1) shows how modifier climbing works for a modifier of the first class, the prefixes. In the following sentences, `haza` ('home') is the verb modifier and `menni/ment` ('to go/went') is the selecting verb:

(4.1) Haza men-t-em
Home go[`past`, `1sg`]
'I went home'

* Men-t-em haza
go[`past`, `1sg`] home

[Mari] haza ment
[Mari] home go[`past`, `3sg`]
'Mary went home'

* [Mari] ment haza
[Mari] go[`past`, `3sg`] home

In neutral sentences, where the verbal complex is a sequence of infinitival verbs combined with a finite auxiliary, the verbal modifier of the lowest infinitival complement precedes the finite auxiliary. As you can see, in example (4.2) `haza` precedes the finite verb `akarok`, in example (4.3) `haza` precedes the auxiliary `fogok`. The prefix is obliged to climb to the first position. Again this climbing is optional for infinitival verb modifiers.

(4.2) haza akarok menni
home want[`1sg`] go[`inf`]
'I want to go home'

∗ akarok menni haza (as neutral sentence)
want[1sg] go[inf] home

∗ akarok haza menni (as neutral sentence)
want[1sg] home go[inf]

(4.3) haza fogok akarni menni
home will[1sg] want[inf] go[inf]
'I will want to go home'

∗ fogok haza akarni menni
will[1sg] home want[inf] go[inf]

∗ fogok akarni menni haza
will[1sg] want[inf] go[inf] home

(4.4)∗ haza fogok menni akarni
home will[1sg] go[inf] want[inf]

The sentence in example (4.4) shows that a unique order of the infinitives in neutral sentences, also called the 'English order' is a necessity. It is not possible to deviate from this order, except in non-neutral sentences under the influence of *verbal inversion*.

Verbal Inversion

Verbal inversion, the other verbal complex formation process, occurs in non-neutral sentences. Non-neutral sentences are sentences which bear *Negative* or *Focus* phrases.

Negative phrases A non-neutral sentence is built up with an auxiliary which takes a sequence of infinitives and verb modifiers as its complement. These infinitives form verbal complexes under the influence of one of the verb modifiers. Optionally these verbal complexes can undergo inversion. The finite auxiliary is the only verb which cannot invert with the verbal complex.

(4.5) [nem] fogok be menni
[not] will[1sg] in go[inf]
'I will [not] go in'

∗ [nem] be fogok menni
[not] in will[1sg] go[inf]

(4.6) [nem] fogok akarni be menni
[not] will[1sg] want in go[inf]
'I will [not] want to go in'

[nem] fogok be menni akarni
[not] will[1sg] in go[inf] want
'I will [not] want to go in'

(4.7) [nem] fogok kezdeni akarni be menni
[not] will[1sg] begin[inf] want[inf] in go[inf]
'I will not begin to want to go in'

[nem] fogok kezdeni be menni akarni
[not] will[1sg] begin[inf] in go[inf] want[inf]
'I will not begin to want to go in'

[nem] fogok be menni akarni kezdeni
[not] will[1sg] in go[inf] want[inf] begin[inf]
'I will not begin to want to go in'

∗ [nem] kezdeni be menni akarni fogok
[not] begin[inf] in go[inf] want[inf] will[1sg]

Example (4.5) shows that the modifier has to precede its selecting verb. Additionally the negated `fogok` cannot be preceded by anything else but the negation `nem`. The following example (4.6) shows the recursive inversion, where first of all, the modifier precedes its selecting verb with which it forms a verbal complex. Then the verbal complex `be menni` can invert with the selecting verb `akarni`, which results in the verbal complex `be menni akarni`. Example (4.7) shows that with any number of infinitival verbs the verbal complex can optionally invert with its preceding selecting verb up to the finite auxiliary `fogok`, which stays at its initial position.

As mentioned, inversion also depends on the kind of verb modifier and on the kind of verb that takes part in the formation of verbal complexes. The verb modifiers of the first class, the prefixes, act as described above. For the second class of bigger modifiers, VM-climbing is obligatory, while inversion is disallowed. The behavior of the third class of verb modifiers, the infinitives, is given in example (4.8). The possible structures with the verb modifier `úszni` ('to swim') show that both VM-climbing and inversion are optional.

(4.8) nem fogok kezdeni akarni úszni
not will[1sg] begin[inf] want[inf] swim[inf]
'I will not begin to want to swim'

nem fogok kezdeni úszni akarni
not will[1sg] begin[inf] swim[inf] want[inf]

nem fogok úszni akarni kezdeni
not will[1sg] swim[inf] want[inf] begin[inf]

Focus phrases Non-neutral sentences with Focus phrases take a different treatment. The focused phrase is inverted and put in front of the sentence. A subphrase which is combined with the rest of the phrase shows the same features as the inverted phrase. Koopman and Szabolcsi (1998) give example 4.9 in their manuscript, where the underlined words are focused.

(4.9) be fogok kezdeni akarni menni
in will begin want go
'I will begin to want to go in (and not out/away)'

be menni fogok kezdeni akarni
in go will begin want
'It is to go in that I will begin to want (I will not begin to want to look/cry)'

be menni akarni fogok kezdeni
in go want will begin
'It is to want to go in that I will begin (I will not begin to sit/cry)'

? be menni akarni kezdeni fogok
in go want begin will
'It is to begin to want to go in that I will (I will not cry/sit down)'

∗ akarni be menni fogok kezdeni
want in go will begin

The examples in (4.9) show that "all inverted sequences can be contrastively focused". Any inverted form is used as part of the focus phrase, which indicates that the forming of these verbal complexes is a form of recursive inversion. The last sentence is rejected because inverted strings that occur in 'English order' cannot be focused.

4.3.3 The formalization of Koopman & Szabolsci's approach

In their manuscript Koopman and Szabolcsi (1998) explain how and why verb complexes are composed. Based on their explanation Stabler (1999) presents two grammar fragments: one to show the basic ideas of inversion and another which follows the precise derivational steps of the formation of verbal complexes under influence of the features on the functional categories.

Koopman and Szabolsci's approach

An important notion Koopman and Szabolcsi (1998) want to address is "a unified account of neutral and non-neutral sentences". Within such an account one should be able to give a unified feature checking analysis of the three patterns described:

1. VM-climbing in neutral sentences
2. 'English order' in non-neutral sentences (before inversion takes place)
3. 'Inverted order' in non-neutral sentences, within the focus and negative phrases

In their opinion, a good proposal of the derivation of alternative orders should not rely on optionality, covert (feature) movement or economy conditions. In their proposal for a grammar they want to tackle some relevant properties of movement: *licensing*, *projection activation*, *pied-piping* and *extraction*.

The first property, *licensing*, is the behavior of a certain word or phrase to allow, or even require, other words or phrases to move to a certain position in their domain (for example, the 'landing position' of a verb modifier is at

Build verbal complex	:	`menni haza`
Inversion of `haza` and `menni`	:	[$_w$ `haza menni`]
Build complementizer phrase (CP)	:	[$_c$ [$_w$`menni haza`]]
Merge `akarni`	:	`akarni` [$_c$ [$_w$`haza menni`]]
Inversion of `haza menni` with `akarni`	:	[$_w$ [$_w$ `haza menni`] `akarni` [$_c$]]
Move CP to licensor position	:	[$_c$][$_w$ [$_w$ `haza menni`] `akarni`]
Inversion of `haza menni akarni` with CP	:	[$_w$ [$_w$ `haza menni`] `akarni`] [$_c$]

Figure 4.18: Inference steps for the derivation of `haza menni akarni`

the specifier position of its selecting verb). The different licensor relations that account for the derivation of the different structures show that various XP's have their own licensing positions.

The second property describes the *activation of projections*, which is "a driving force behind movement" (Koopman and Szabolcsi, 1998). Projection is a mechanism that causes lexical elements to enter into a local relationship. An example of this is the local relationship between a verb modifier and its selecting verb, a lexical requirement on both words. Thus a projection is activated by lexical material or some feature. In the absence of such material, something has to move in order to be activated. According to Koopman and Szabolcsi (1998) the *Principle of Projection Activation* plays a role in the "shuffling" of lexical material, where movement of a word does not necessarily involve feature checking. They give a description of *Mellowed PPA*: "a projection is interpretable iff it has lexical material at some stage in the derivation, CP and LP(cp) are interpretable iff they have lexical material, or the trace of lexical material".

Pied-piping captures an essential feature of linguistic grammars, which prescribes that extra material within the domain of a certain word or phrase be moved along during climbing or inversion. A requirement of the phenomenon of Hungarian verbal complexes is that the order of the pied-piped material is inverted as well.

Extraction is the main property of the inversion of words during the process of verbal complex formation. The main part of this feature is the specification of the kind of phrases that can be extracted: "Only specifiers and full complements ... can extract". The grammar explains movement and extraction with XP-movement because head-movement shows many deficiencies.

The explanation of Koopman and Szabolcsi (1998) for the phenomena described in section 4.3.2 still needs a formalization. As an example, we give the analysis of the phrase "`haza menni akarni`" as a verbal complex in a non-neutral sentence. The inverted order is achieved with the inference steps in Fig. 4.18.

Stabler's formalization

Stabler (1999) focuses on verbal inversion in non-neutral sentences. He wrote two grammars to account for this phenomenon: MG1 and MG2[2]. The first grammar gives the basic ideas of verbal inversion. He distinguishes two licensor features, [+m] and [+v]. These features can be seen as triggers for *a*) VM-climbing of modifiers decorated with a licensee feature [−m] and *b*) recursive inversion for structures decorated with [−v]. The grammar defines nine lexical entries which derive the three possible structures. Schematically the lexical entries are V1 (the auxiliary), V2, V3, V4 (infinitival verbs) and M (the verbal modifier).

Structure 1	Structure 2	Structure 3
=v v V1		
=v v V2		=v +v v V2
=v v V3	=v +v v V3	=v +v v -v V3
=m +m v V4	=m +m v -v V4	
m -m M		

Figure 4.19: MG1, minimalist grammar fragment to derive verbal inversion

With the lexical entries of this grammar, Stabler explains how the three possible structures are derived. To derive one structure from another, some of the lexical feature structures need to be changed. The entries in the first column are used to derive the first structure, for the second structure one needs to change the entries in the second column and for the last structure one needs the changes in the third column. Grammar MG1 derives the following three structures:

1. **V1 V2 V3 M V4**
2. **V1 V2 M V4 V3**
3. **V1 M V4 V3 V2**

The second grammar, MG2, formalizes the inference steps in Fig. 4.18 for the derivation of the inverted form of verbal complexes. I simplified his original grammar by stripping the functional category c and its licensor features. With only the use of the functional category w, the grammar in Fig. 4.20 derives the same inferences as in Fig. 4.18.

```
m -i haza
=m v menni
=v +i w -i
=w v akarni
```

Figure 4.20: MG2, minimalist grammar fragment to derive verbal inversion

[2]For an overview look at Appendix B

4.3.4 MMCG approach

In this section we explore the phenomenon of verbal complexes from a deductive perspective. The two approaches for translation the movement operation are contrasted. The phenomenon is explored in the same order as presented in section 4.3.2. For both approaches I implemented fragments in Grail (see Appendix C).

VM climbing

To account for *verb modifier climbing* I used MG1, where the features on the moving elements act as licensees for the movement. In the translation to MMCG, I focus on the idea that the trigger for movement is a lexical requirement on the words involved. If the modifier is a prefix, the prefix is responsible for the movement.

In Stabler's grammar MG1, the selecting verb also plays a role in triggering movement. The modifier is marked with a licensee feature, but the selecting verb carries the trigger: the corresponding licensor feature. I propose a different approach; in section 4.2 we have seen that MOVE has a structural and a logical component. To account for verbal complex formation we use *structural reasoning* to project 'verbal complex' domains and *hypothetical reasoning* to implement abstraction.

We set aside extra information about tense, aspect and other formal features. The features that are required for the analysis of the different phenomena are the focus of this section. The following basic categories are involved: v_{inf} for infinitival verbs, v_{fin} for finite verbs and m for modifiers.

The sentences in 4.10 and 4.11 demonstrate the VM-climbing of prefixes. For the glosses and the translation of these sentences look at example (4.2) and (4.3) respectively.

(4.10) haza akarok menni

akarok menni haza (as neutral sentence)

∗ akarok haza menni (as neutral sentence)

(4.11) haza fogok akarni menni

∗ fogok haza akarni menni

∗ fogok akarni haza menni

Move as structural reasoning The different words involved have a certain categorial status within phrasal structures. The categories of the distinct lexical entries are determined by the role they play within sentences. The auxiliaries (akarok/akarni, fogok) are finite or non-finite verbs, which select infinitival verbs as a complement. The infinitival verb menni selects a prefix as complement. The prefixes be and haza are modifiers, which climb to precede the finite verb. To account for the modifier feature as trigger for VM-climbing, a unary connective decorated with a 'modifier feature': $\Box_m$ is introduced.

All the features and characteristics of words need to be stated in the lexicon. The auxiliary fogok (= will) is a finite verb which takes an infinitival verb as

complement. The verbs akarni (= want) and menni (= go) are both infinitival verbs, which take another infinitival verb or a modifier as a complement. The licensor feature on the selecting verb corresponds to the $\Box_m$ feature on the negative goal formula. The modifier haza (= home) carries the special modifier feature.

MG1	MMCG	
`=v v akarni`	akarni $\vdash v_{inf}/_< v_{inf}$	('want[`inf`]')
`=v v akarok`	akarok $\vdash v_{fin}/_< v_{inf}$	('want[`1sg`]')
`=v v fogok`	fogok $\vdash v_{fin}/_< v_{inf}$	('will[`1sg`]')
`=m +m v menni`	menni $\vdash v_{inf}/_< m$	('go[`inf`]')
`m -m M`	haza $\vdash \Box_m m$	('home')

Figure 4.21: Lexicon for VM-climbing with structural reasoning

How does the modifier trigger VM-climbing? The modifier carries a licensee feature: $\Box_m$. The structural counterpart of the unary connective: $\langle . \rangle^m$ triggers 'movement'. The movement postulates as given in Fig. 4.11 are transposed for the modifier feature in Fig. 4.22. The 'modifier' feature triggers the application of these movement postulates.

$$\begin{array}{rll} \Diamond_m(A \bullet_> B) \rightarrow \Diamond_m A \bullet_> B & & [K1m] \\ \Diamond_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (C \bullet_< \Diamond_m A) & & [Mm1] \\ \Diamond_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (\Diamond_m A \bullet_> C) & & [Mm2] \end{array}$$

Figure 4.22: Structural postulates for VM climbing

To derive a sentence such as haza akarok menni, the basic "English order" is built first: akarok menni haza (see Fig. 4.23). The modifier has to move higher in the structure under influence of its modifier feature. Following postulate $[Mm1]$ the modifier moves up to precede the finite verb akarok.

$$\dfrac{\dfrac{\text{akarok} \vdash v_{fin}/_< v_{inf} \qquad \dfrac{\text{menni} \vdash v_{inf}/_< m \qquad \dfrac{\text{haza} \vdash \Box_m m}{\langle \text{haza} \rangle^m \vdash m}\,[\Box E]}{\text{menni} \circ_< \langle \text{haza} \rangle^m \vdash v_{inf}}\,[/_< E]}{\text{akarok} \circ_< (\text{menni} \circ_< \langle \text{haza} \rangle^m) \vdash v_{fin}}\,[Mm1]}{\langle \text{haza} \rangle^m \circ_> (\text{akarok} \circ_< \text{menni}) \vdash v_{fin}}\,[K1m]}{\langle \text{haza} \circ_> (\text{akarok} \circ_< \text{menni}) \rangle^m \vdash v_{fin}}\,[\Box I]}{\text{haza} \circ_> (\text{akarok} \circ_< \text{menni}) \vdash \Box_m v_{fin}}$$

Figure 4.23: VM-climbing with structural reasoning

At the end of the formation process, the whole verbal complex has a certain categorial status, namely a verb phrase which has undergone VM-climbing: a special kind of verbal complex. The category of the verbal complex phrase is marked with the same feature as the modifier. The derivation of the neutral sentence "haza akarok menni", where verb modifier climbing has taken place, gives us a sentence of category $\Box_m v_{fin}$.

Move as abstraction As explained in section 4.2, movement can be captured within the structural domain by defining 'movement' postulates. But this does not capture the meaning of movement as involving abstraction. With the use of hypothetical reasoning as defined by the introduction rules of the base logic, one is able to account for the abstraction of a hypothetical phrase out of a fully build structure:

$$\frac{\Gamma \circ_< x : B \vdash t : A}{\Gamma \vdash \lambda x.t : A/_< B}\ [/_< I] \qquad \frac{x : B \circ_> \Gamma \vdash t : A}{\Gamma \vdash \lambda x.t : B\backslash_> A}\ [\backslash_> I]$$

To bring about abstraction, the lexical formula of the word which undergoes VM-climbing, the prefix, needs to be changed. Haza is lifted to the higher order type:

$$\mathsf{haza} \vdash v_{fin}/_>(\Diamond_m \Box_m m\backslash_> v_{fin})$$

Haza has to combine with a finite verb phrase where a modifier has been abstracted from. The logical formula assigned to haza contains both licensor $\Diamond_m$ and licensee $\Box_m$ feature. Doing so, haza is the word that triggers VM-climbing; the modifier itself is responsible for the abstraction. Again the structural counterpart of this 'lexical' licensor feature: $\langle . \rangle^m$ triggers the movement postulates. Fig. 4.24 shows that the same postulate that was needed for the VM-climbing in the fragment with structural reasoning is needed here. The postulate moves the hypothesized modifier out of its former position to the specifier position, where it is subject to abstraction.

Merge over Move Another option, which makes the derivation more economical in a minimalistic sense, is assign haza the type:

$$\mathsf{haza} \vdash \Box_m(v_{inf}/_>(v_{inf}/_< m))$$

This translation also accounts for movement as abstraction operation. But it also captures the economy principle of 'MERGE over MOVE' where function application is favored over structural movement.

The derivation in Fig. 4.25 slightly deviates from the derivation in Fig. 4.23 on two points. First, the order of the applications of words differs: Haza selects menni to form the substructure haza menni where menni is still regarded as the head of the structure. the second different is the use of postulate $[Mm2]$, which moves phrases from a specifier position, instead of postulate $[Mm1]$, which moves complements.

Verbal inversion

As outlined in section 4.3.2 inversion occurs in non-neutral sentences. The overall characteristic of the phenomenon is the recursive inversion of verbal complexes, where the basic surface order, the 'English order', contrasts with the 'inverted order' of the verbal complex. Some general principles underlie inversion: a verbal complex, which is inverted with another verb, has to be in inverted form itself; the uninverted part of the structure stays in 'English order'.

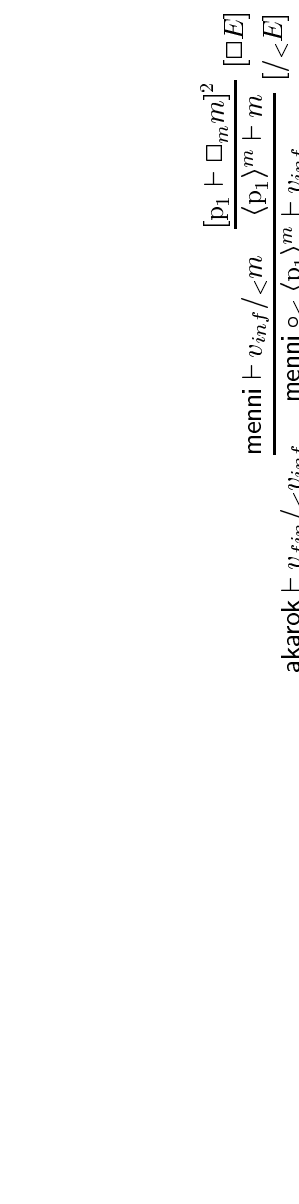


Figure 4.24: VM-climbing with hypothetical reasoning

$$\dfrac{\dfrac{\dfrac{\mathsf{haza} \vdash \Box_m(v_{inf}/_>(v_{inf}/_<m))}{\langle \mathsf{haza}\rangle^m \vdash v_{inf}/_>(v_{inf}/_<m)}\,[\Box E] \qquad \mathsf{menni} \vdash v_{inf}/_<m}{\mathsf{akarok} \vdash v_{fin}/_<v_{inf} \qquad \langle \mathsf{haza}\rangle^m \circ_> \mathsf{menni} \vdash v_{inf}}\,[/_>E]}{\mathsf{akarok} \circ_< (\langle \mathsf{haza}\rangle^m \circ_> \mathsf{menni}) \vdash v_{fin}}\,[/_<E]$$
$$\dfrac{\langle \mathsf{haza}\rangle^m \circ_> (\mathsf{akarok} \circ_< \mathsf{menni}) \vdash v_{fin}}{\langle \mathsf{haza} \circ_> (\mathsf{akarok} \circ_< \mathsf{menni})\rangle^m \vdash v_{fin}}\,[Mm2]$$
$$\dfrac{}{\mathsf{haza} \circ_> (\mathsf{akarok} \circ_< \mathsf{menni}) \vdash \Box_m v_{fin}}\,[\Box I]$$

Figure 4.25: VM-climbing with hypothetical reasoning to account for 'MERGE over MOVE'

Recursive inversion in Negative phrases The sentences of example (4.7) and repeated in (4.12) are analyzed by Stabler (1999). He defines a grammar to formalize the inversion steps. The translation of his formalism into MMCG gives us a similar analysis for inversion in MMCG.

(4.12) [nem] fogok kezdeni akarni be menni

[nem] fogok kezdeni be menni akarni

[nem] fogok be menni akarni kezdeni

* [nem] kezdeni be menni akarni fogok

Move as structural reasoning The categorial grammars that account for VM-climbing are inspired on grammar MG1 (Fig. 4.19). In this section a mapping of Stabler's second grammar MG2 (Fig. 4.20) is taken into account which formalizes verbal inversion more precisely.

As described in Koopman and Szabolcsi (1998), the inverted order is achieved when the inference steps build the succeeding phrases as outlined in figure 4.18. Stabler implements this recursive inversion of phrases using lexical feature specifications. Functional categories trigger the different inversion steps. The lexical feature structures are translated into MMCG formulas following the rules in figure 4.7. The only change in the MG grammar from Stabler's original grammar is that the feature information of [`casus`] and the corresponding functional categories are left out.

The first three lexical structures are the words that appear phonologically. They carry feature information that will be used by the functional categories. The other two entries are functional categories, which carry licensor features to trigger the movement operations. As explained in section 4.2.1 in MMCG the licensor feature corresponds to a $\Box_f$ on a negative subformula or on the goal formula. For example, in the MG grammar the functional category `w` carries a licensor feature [+`i`]. In MMCG it appears as $\Box_i$ on the negative subformula of akarni that in turn selects the category feature w: $\mathsf{akarni} \vdash v/_<\Box_i w$.

MG	MMCG
`m -i haza`	$\mathsf{haza} \vdash \Box_i m$
`=m v menni`	$\mathsf{menni} \vdash v/_< m$
`=w v akarni`	$\mathsf{akarni} \vdash v/_< \Box_i w$
`=v +i w -i`	$w \vdash (\Box_i w/_< v)$
	with $\Box_i w$ as subformula of `akarni`
`=v +i w`	$w \vdash (w/_< v)$
	with $\Box_i w$ as goal formula

Figure 4.26: Lexical correspondence for verbal inversion

To derive the inverted expression `haza menni akarni` the *movement* postulates in Fig. 4.27 are needed. With the two postulates the derivation in Fig. 4.28. It is easy to extend the lexicon to derive the inverted expressions of example (4.12).

$$\Diamond_i C \bullet_> (A \bullet_< B) \rightarrow A \bullet_< (B \bullet_< \Diamond_i C) \quad [Mi1]$$
$$\Diamond_i (A \bullet_> B) \rightarrow \Diamond_i A \bullet_> B \quad [K1i]$$

Figure 4.27: Structural postulates for recursive inversion in negative phrases

The derivation starts with all the possible steps, until there are no further logical steps possible. Then the structural postulates are applied in the same order in which the inference steps in the minimalist derivation are applied.

$$\dfrac{\dfrac{w \vdash w/_< v \qquad \dfrac{\mathsf{akarni} \vdash v/_< \Box_i w \qquad \dfrac{w \vdash \Box_i w/_< v \qquad \dfrac{\mathsf{menni} \vdash v/_< m \qquad \dfrac{\mathsf{haza} \vdash \Box_i m}{\langle \mathsf{haza} \rangle^i \vdash m}\,[\Box E]}{\mathsf{menni} \circ_< \langle \mathsf{haza} \rangle^i \vdash v}\,[/_< E]}{w \circ_< (\mathsf{menni} \circ_< \langle \mathsf{haza} \rangle^i) \vdash \Box_i w}\,[/_< E]}{\mathsf{akarni} \circ_< (w \circ_< (\mathsf{menni} \circ_< \langle \mathsf{haza} \rangle^i)) \vdash v}\,[/_< E]}{w \circ_< (\mathsf{akarni} \circ_< (w \circ_< (\mathsf{menni} \circ_< \langle \mathsf{haza} \rangle^i))) \vdash w}\,[Mi1]$$
$$\dfrac{w \circ_< (\mathsf{akarni} \circ_< (\langle \mathsf{haza} \rangle^i \circ_> (w \circ_< \mathsf{menni}))) \vdash w}{w \circ_< (\mathsf{akarni} \circ_< \langle \mathsf{haza} \circ_> (w \circ_< \mathsf{menni}) \rangle^i) \vdash w}\,[K1i]$$
$$\dfrac{\langle \mathsf{haza} \circ_> (w \circ_< \mathsf{menni}) \rangle^i \circ_> (w \circ_< \mathsf{akarni}) \vdash w}{\langle (\mathsf{haza} \circ_> (w \circ_< \mathsf{menni})) \circ_> (w \circ_< \mathsf{akarni}) \rangle^i \vdash w}\,[Mi1]$$
$$\dfrac{\langle (\mathsf{haza} \circ_> (w \circ_< \mathsf{menni})) \circ_> (w \circ_< \mathsf{akarni}) \rangle^i \vdash w}{(\mathsf{haza} \circ_> (w \circ_< \mathsf{menni})) \circ_> (w \circ_< \mathsf{akarni}) \vdash \Box_i w}\,[K1i]$$
$$[\Box I]$$

Figure 4.28: Structural derivation of haza menni akarok with functional categories

Move as abstraction In this section I present another translation of Stabler's grammar fragment of verbal inversion. This MMCG fragment uses higher

order type assignments to invoke verbal inversion by reasoning hypothetically about MERGE. Although this results in more complex lexical entries this, in my opinion, gives exactly the right result.

The lexicon from the MG grammar, given in figure 4.18, can be split up into two: one to derive the simple structure **haza menni** and another to derive the structure where recursive inversion on the auxiliary `akarni` has been applied: **haza menni akarni**. Fig. 4.29 shows the lexical correspondence between MG and MMCG for the inversion of the verbal modifier over the selecting verb.

MG	MMCG
`=v +i w`	-
`=m v menni`	$\mathsf{menni} \vdash w/_< m$
`m -i haza`	$\mathsf{haza} \vdash w/_>(w/_<\Diamond_i\Box_i m)$

Figure 4.29: Lexical correspondence for verbal inversion with hypothetical reasoning

The feature information captured by the functional categories is transferred to the lexical entries of both haza and menni. Whereas menni captures all the selecting information (`=v,=m`), haza inherits all the control features defined on the functional category `w`. The licensor and licensee feature are needed for the climbing of the modifier; they are translated with $\Diamond_i\Box_i$ in the higher order type of haza. The way the logical formulas are constructed 'predicts' the order of the structure.

On the basis of these lexical assignments we can derive the inversion of "haza menni" in Fig. 4.30 with the accompanying proof term: $(\mathbf{haza}\ \lambda z.(\mathbf{menni}\ z)) : w$. The proof term shows that abstraction takes place out of the verbal complex and furthermore that the abstracted modifier haza is applied to the verbal complex. The structure can be completed by applying other auxiliaries such as ($\mathsf{fogok} \vdash w/_< w$) and negation nem to it, yielding the structure: nem fogok haza menni.

$$\dfrac{\mathsf{haza} \vdash w/_>(w/_<\Diamond_i\Box_i m) \qquad \dfrac{\dfrac{[\mathrm{r}_0 \vdash \Diamond_i\Box_i m]^1 \qquad \dfrac{\mathsf{menni} \vdash w/_< m \qquad \dfrac{\dfrac{[\mathrm{p}_1 \vdash \Box_i m]^2}{\langle \mathrm{p}_1\rangle^i \vdash m}\,[\Box E]}{\mathsf{menni} \circ_< \langle \mathrm{p}_1\rangle^i \vdash w}\,[/_< E]}{\mathsf{menni} \circ_< \mathrm{r}_0 \vdash w}\,[\Diamond E]^2}{\mathsf{menni} \vdash w/_<\Diamond_i\Box_i m}\,[/_< I]^1}{\mathsf{haza} \circ_> \mathsf{menni} \vdash (\mathbf{haza}\ \lambda z.(\mathbf{menni}\ z)) : w}\,[/_< E]$$

Figure 4.30: MMCG derivation: haza menni

The succeeding structure is the result of the recursive inversion of the previous structure with akarni: **haza menni akarni**. In this phrase also menni has the property to invert. This is accomplished by decorating the head formula with $\Diamond_i\Box_i$. Consequently, the type assigned to the verb modifier also has to be lifted. This yields the lexicon in Fig. 4.31 on the basis of the MG grammar.

The complex derivation in Fig. 4.32 shows the recursive inversion solely on the basis of logical inferences. There is no need for structural postulates, the

MG	MMCG
`=v +i w -i`	-
`=v +i w`	-
`=w v akarni`	akarni $\vdash w/_< w$
`=m v menni`	menni $\vdash (w/_>(w/_<\Diamond_i\Box_i w))/_< m$
`m -i haza`	haza $\vdash w^*/_>(w^*/_<\Diamond_i\Box_i m)$
	$w^* \vdash w/_>(w/_<\Diamond_i\Box_i w)$

Figure 4.31: Lexical correspondence after lifting

abstraction of phrases and the trigger for this abstraction lies solely on the lexical entries involved in the derivation.

By lifting one more time one can derive the succeeding phrase **haza menni akarni kezdeni**. In the MG grammar another functional category w would be added. All occurrences of w are lifted to $(w/_>(w/_<\Diamond_i\Box_i w))$ (w^*). Kezdeni gets assigned the type $w/_< w$. The derivation is too complex to present here, but the basic principle is the same as the derivation in Fig. 4.32. The extended lexicon is presented in Appendix C.

4.4 Discussion

The goal of this chapter is to give a deductive analysis of the different operations and components of the Minimalist Program. The mapping between MG and MMCG brings out some crucial issues with respect to the MOVE operation.

At the end of section 4.2 the importance of the derivational meaning of MOVE becomes apparent. MOVE is regarded as an abstraction operation; the abstraction should be reflected by the proof term of a derivation. The translation of MOVE with structural reasoning alone does not give this derivational meaning. But by appealing to hypothetical reasoning the derivation shows exactly the abstraction of a phrase out of a bigger structure. Higher order types are used to trigger the hypothetical reasoning.

The *elimination of functional projections* stems from the use of higher order type assignments. With the possibility to incorporate feature information in the type-logical formulas assigned to the moving elements, there is no need for lexical elements such as the functional categories. The trigger of MOVE is lexically anchored. Whereas the lexicon carries all the necessary feature information, the cooperation between the structural and logical part of the derivation realizes the right order and dependency of the words involved.

Consequently the elimination of functional projections is a simplification for both the lexical and the derivational complexity. One needs fewer lexical entries and fewer logical application rules (= MERGE) to get the same result. When fewer lexical elements take part in the derivation, the structural complexity reduces as well.

A theory of Universal Grammar has to concentrate on the three different parts of a derivation and the interaction between these parts: structure, logic and derivational meaning.

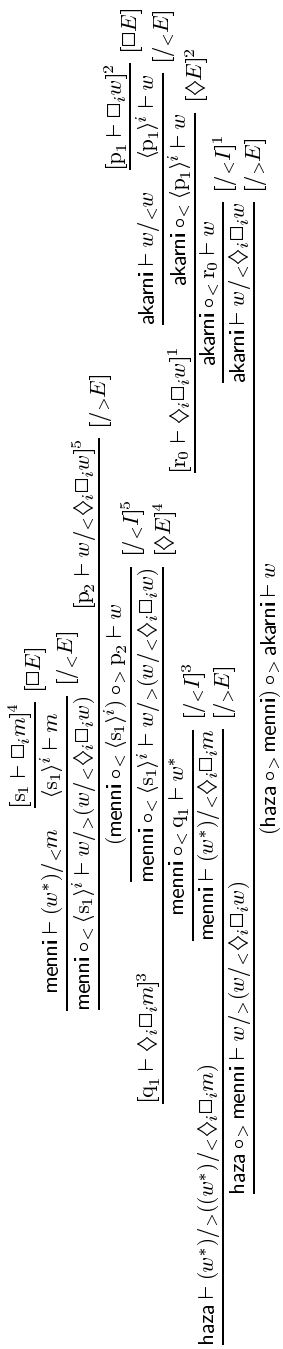


Figure 4.32: MMCG derivation of haza menni akarni using higher order reasoning

Chapter 5

Conclusion

As seen in the previous chapter, the basic ideas of a minimalist theory of natural language find their logical justification in the framework of Multimodal Categorial Grammar. This chapter concludes with a discussion on the results to find this justification. Section 5.1 gives an overview of some of the similarities and differences between the two theories explored in this thesis. Section 5.2 formulates an answer to the question raised in chapter 1. Moreover, in section 5.3 we summarize our findings on the main theme in this thesis: *Controlling Movement*. Section 5.4 gives some directions for future research.

5.1 Minimalism in a deductive framework

The Minimalist Program intends to define a *minimal* theory explaining natural language. Chomsky (1995) strives to capture minimalism in the two main components of the Minimalist Program: the lexicon and the Computational System. The minimality of a theory can be captured by two complexity measures: the *complexity* of basic operations and *economy of derivations*.

Although Chomsky (1995) does not state precisely what he means by economy or complexity, one could think of the number of steps in a derivation as a potential measure for the complexity of a derivation. To reduce the number of steps in a derivation Chomsky (1995) states *economy conditions*. On the other hand he also states that a minimal theory should reduce the use of those conditions.

As derivations are driven by the need to check features, the influence of features on the execution of operations seems a more efficient way to control the economy of derivations. Therefore the lexicon as a database system for all kinds of feature information plays an important role in the measure of the complexity of a minimal theory.

As seen in section 4.2 it is possible to translate basic components of Minimalist Grammar in a logical and structural framework. The mapping of the Minimalist Program via Stabler's formalism into MMCG offers ways to relate the minimality of the two theories. The minimality in MMCG concentrates on the basic operations, the derivational complexity and the content of the lexicon.

5.1.1 Basic operations

The Computational System of the Minimalist Program contains operations which have direct influence on the construction of phrasal structures. Likewise the base logic in MMCG consists solely of operations that build structures on the basis of the type assignments of the words involved. The set of possible operations in MMCG based on the base logic consists solely of elimination and introduction rules of the available unary and binary connectives. The operations are defined and controlled within the proof system. MERGE is given by the elimination of the binary connectives $\{/, \backslash\}$. MOVE can be captured by the structural rules as an extension of the proof system.

5.1.2 Derivational complexity

A way to measure the complexity of a derivation is is by the amount of movement steps taken. Then to consider MOVE as a decomposed operation of abstraction and application reduces the complexity of a derivation. MMCG defines MOVE as abstraction by reasoning hypothetically about MERGE. Because hypothetical reasoning depends on the higher order types assigned to lexical elements, the displacement of words can be defined as a derivational property of actual lexical items. The comparison between the two analysis of VM-climbing in Fig. 4.23 and Fig. 4.24 shows that MOVE on the basis of hypothetical reasoning uses fewer structural rules to derive the same structural configuration of words. Furthermore the analysis in Fig. 4.25 with higher order reasoning supports the economy condition of 'MERGE over MOVE', stating that the operation MERGE is favored over the operation MOVE.

Apart from the complexity measure by the number of movement steps, other complexity measures could play a role. This should be researched further and may result in the construction of a more efficient proof system with directions how to compute the complexity of a derivation within the Computational System.

5.1.3 The lexicon

Lexical Ambiguity

A lexicon consists of many lexical entries which bear information about the meaning, the use and other properties of words (section 3.2). Some words may appear in the lexicon with multiple types while their use or meaning is the same. For the lexicon to be an efficient database, this kind of ambiguity should be avoided. An efficient lexicon should consist of lexical entries with only one type assigned for one kind of derivational use.

As mentioned in section 4.3.3 Stabler (1999) needs several feature structures for one lexical entry to derive structures based on the same principle. To overcome the problem of '*ambiguity overload*' MMCG uses the internal derivational power of the proof system. A type assignment can be lifted to a higher order type; without having to add new lexical entries. In this way lexical entries can be used dynamically. As seen in section 4.3.4 on verbal inversion words such as `menni`(= go) are used to build three different structures by recursively lifting its category to a higher order type.

Parametric differences

A Universal Grammar has to capture the parametric differences between languages. The Minimalist Program uses the feature specification in the lexicon to capture these differences. Within MMCG three parameters influence the differences between languages and particular linguistic phenomena.

First, MMCG proposes *strong lexicalism* (section 3.2): the lexicon captures all the feature information needed to derive and interpret all the possible structures. Semantic labels (as seen in section 4.2.5) are added to the lexical entries in such a way that the derivational meaning of different languages correspond.

Second, similar to the way interpretability of features in the Minimalist Grammar can be differentiated, the control features can be defined differently in every lexicon. Every control feature that triggers certain structural postulates, has a different impact on their use. The 'strength' and therefore the use of these features differs in all languages.

Control features influence the third parameter: the structural postulates. They record the language specific principles and rules that account for certain phenomena. In addition the structural postulates change the order and hierarchical grouping of the structure and cause the displacement of feature information and lexical phrases.

As seen in chapter 3, the Minimalist Program leaves a distinction between the structural and the logical part of a derivation. However, the functional categories show structural and logical behavior; the categorial and formal feature information on the functional categories direct the derivation to the right structural order. Therefore the use of functional categories corresponds with the use of structural postulates which are both under control of features.

5.2 Minimal movement

In this and the following section I formulate a more concrete answer to the question raised in the introduction of my thesis.

> How are certain aspects of movement as described by the Minimalist Program captured in the MMCG framework?

After a brief summary of the aspects of minimalist movement, we address the main parts of a multimodal approach towards movement. The answer is split up into a section on MOVE as a structural operation and a section on the observation of MOVE as a complex operation.

5.2.1 Movement in the Minimalist Program

As described in section 3.3.2 the Minimalist Program defines the operation MOVE/ATTRACT as the attraction of formal, phonological and semantic features in order to check an uninterpretable feature on a functional category. The Minimalist Program concentrates on the interpretability of formal features on the PF and LF interfaces. As long as uninterpretable features are present, the derivation continues with operations until all uninterpretable features are checked. The need for feature checking drives the derivation to its phonological and logical form.

5.2.2 Movement in MMCG

Two ways of translating movement in MMCG have been worked out in this thesis. One way is the translation of MOVE as a structural operation; structural feature domains move through the structure in order to check against its logical residual. The second translation focuses on the internal meaning of MOVE as an abstraction operation.

Move as structural reasoning

The translation of MOVE as structural reasoning captures the actual movement of a phrase through a structure under the control of certain formal features. The control features are defined as unary connectives attached to the lexical formulas of the responsible words.

The distinction in MMCG between the logical and structural parts of a derivation points to the possibility to interact between the two parts. Features stated on the lexical category of a word, the logical part, correspond (using the $[\Box_f E]$ rule) with the structural part of the proof. Fig. 5.1 shows how a word carrying a control feature has direct influence on the construction of the structure. A number of MERGE and MOVE steps that are defined within the proof system by logical rules and structural postulates, drive the derivation. Feature checking takes place when the $[\Box_f I]$ rule transfers the structural feature domain over to the goal formula.

$$\dfrac{\dfrac{\dfrac{S_1 \vdash \Box_f A}{\langle S_1 \rangle^f \vdash A}\,[\Box_f E]}{\vdots\ \text{MERGE \& MOVE}}}{\dfrac{\langle S_2 \rangle^f \vdash B}{S_2 \vdash \Box_f B}\,[\Box_f I]}$$

Figure 5.1: Schematic view of a derivation

To capture the structural behavior of a linguistic phenomenon the right movement postulates have to be formulated. As seen in section 4.3.2 on verb modifier climbing and in section 4.3.2 on verbal inversion in Hungarian, the three postulates in Fig. 5.2 capture the possible movement of phrases and features:

$$\begin{array}{rl} \Diamond_f (A \bullet_> B) \rightarrow \Diamond_f A \bullet_> B & [P1] \\ \Diamond_f A \bullet_> (B \bullet_i C) \rightarrow B \bullet_i (C \bullet_< \Diamond_f A) & [P2] \\ \Diamond_f A \bullet_> (B \bullet_i C) \rightarrow B \bullet_i (\Diamond_f A \bullet_> C) & [P3] \end{array}$$
$$\text{Where } i \in \{<, >\}$$

Figure 5.2: Movement postulates

Move as abstraction

Section 4.2.4 shows MOVE is not a primitive operation. MOVE can be decomposed into two operations: *abstraction* and *application*. The derivational

meaning of MOVE cannot be captured exclusively by structural control. Using the derivational strength of the base logic, movement can be translated as the structural control over a hypothesized object by reasoning hypothetically about MERGE.

The determination of MOVE as a complex operation with both structural and logical aspects is another result of the interaction between structure and logic. This interaction is further advanced by the communication between the higher order type on the logical side and the hypothesized object on the structural side via the elimination and introduction rules of the binary connectives, $\{/, \backslash\}$.

The use of higher order types as triggers for the construction of different structural hierarchies makes the use of functional categories as triggers of the movement of internal phrases and features superfluous. The control of the movement of a modifier during verbal complex formation or the displacement of a wh-object phrase can be defined on the modifier or the wh-word alone.

5.3 Controlling Move

As the title of my thesis indicates I tried to show what mechanism controls the operation MOVE in MMCG. In this section once more, I contrast the control mechanism of the Minimalist Program with the mechanisms that MMCG offers.

5.3.1 Controlling Move in the Minimalist Program

As described in section 3.2, a minimalist lexicon contains the feature information of the lexical items on which the Computational System operates. Functional categories play an essential role in the attraction and movement of features within derivational structures. The uninterpretable features on the functional categories communicate with corresponding features on substructures. The interplay between interpretable and uninterpretable features gives rise to the movement of phrases within structures. The control over movement lies in the interaction between the uninterpretable features assigned to the functional categories and the necessity to check all uninterpretable features (*Last Resort*).

5.3.2 Controlling Move in MMCG

In the translation of MOVE as structural reasoning alone, the features on the goal formula trigger the application of structural rules to accomplish movement within the structure. The interaction between structure and logic influences the way the derivation goes; the features on the goal formula correspond with the features of the displaced elements in the structure. As long as control features are still present at the structural side of the derivation and not checked against the logical features, the derivation has to continue. The control lies in the interaction and the connection between the features on the words involved and the features on the goal category of the derivation; the feature on the goal formula directs the derivation.

In the translation of MOVE as structural control using hypothetical reasoning, control is solely captured by the features of the higher order type which is assigned to the moved element. There is no need for features on the goal formula. So to say, the *licensor* and the *licensee* feature (Stabler, 1999) are both present

on the element which is subject to displacement. The control over movement is solely defined on the feature specification of the words involved. The analysis of verbal inversion in section 4.3.4 shows that functional categories are not necessarily needed and therefore can be eliminated. The role of functional categories as triggers of movement is taken over by the feature information on the higher order type which interact with the right determined structural postulates.

5.4 Future research

A lot more needs to be researched before an unified theory on minimalism and multimodal Categorial Grammar can be formulated. A few directions for further exploration follow from the discussion of my thesis such as the economy of derivations, the role of interpretable and uninterpretable features in MMCG, the possible elimination of functional categories and more worked out fragments of linguistic phenomena.

As one can see, even for a balanced comparison between the Minimalist Program and MMCG, we need to explicate a lot of these topics. I hope this thesis contributes to the integration of two research areas that try to capture the Computational System of Human Language.

Appendix A

Index of Features

1. (a) Semantic: *+human, -human*
   (b) Phonological: ± *back,* ± *dental*
   (c) Formal:
      - ϕ-features: *person, number, gender*
      - category: *noun, verb, adjective*
      - functional categories: *V, T, C, D, agreement*
      - case: *nom, acc, dat*
      - EPP
      - Q
      - tense: *present, past, finite, infinite*
2. Strength: Language specific
3. (a) Uninterpretable for the whole Computational System:
      - the strength of feature
   (b) Uninterpretable at LF:
      - case of NP, T, V
      - ϕ-features of V and T
      - *-V, EPP* of T
      - *-V* of $\underline{v}$
      - phonological features
   (c) Interpretable at LF: all other formal features
      - all categorial features (except expletives)
      - ϕ-features of NP
      - wh, Q
      - semantic features
4. (a) Optional:
      - *number, case* of N
      - *tense, person, number, case* of Tense
      - Q of C

(b) Intrinsic: All other formal features such as

- *person, category* of N
- *-V, EPP* of Tense
- V of $\underline{v}$

Appendix B

Minimalist Grammars

A Minimalist grammar **MG** consists of a lexicon *Lex* and a set of generating functions $\mathcal{G}$.

- *Lex* is a finite set of finite sequences of features $\mathcal{F}$
- $\mathcal{F} ::= \mathcal{N} \mid =\mathcal{N} \mid +\mathcal{N} \mid -\mathcal{N} \mid /\mathcal{N}/ \mid \underline{\mathcal{N}}$
- $\mathcal{G} = \{\textsc{Merge},\textsc{Move}\}$

B.1 MG1

Grammar **MG1** grasps the basic ideas of verbal inversion. Two licensor features, [+m] and [+v] trigger the inversion of the modifier with feature [−m] and the recursive inversion of verbal complexes decorated with [−v]. The grammar defines nine lexical entries which derive the three possible structures.

Lex 1	*Lex* 2	*Lex* 3
`=v v fogok`		
`=v v akarni`		`=v +v v akarni`
`=v v kezdeni`	`=v +v v kezdeni`	`=v +v v -v kezdeni`
`=m +m v menni`	`=m +m v -v menni`	
`m -m haza`		

Figure B.1: Lexicon of **MG1**

The three sets of lexical entries derive:

1. *Lex* 1: fogok kezdeni akarni haza menni
2. *Lex* 2: fogok kezdeni haza menni akarni
3. *Lex* 3: fogok haza menni akarni kezdeni

B.2 MG2

The second grammar, **MG2**, formalizes the description which Koopman and Szabolcsi (1998) give for the derivation of the inverted form of verbal sentences in Hungarian.

Build verbal complex	:	`menni haza`
Inversion of `haza` and `menni`	:	$[_w$ `haza menni` $]$
Build complementizer phrase (CP)	:	$[_c\ [_w$`menni haza` $]\ ]$
Merge `akarni`	:	`akarni` $[_c\ [_w$`haza menni` $]]$
Inversion of `haza menni` with `akarni`	:	$[_w\ [_w$ `haza menni` $]$ `akarni` $[_c\ ]\ ]$
Move CP to licensor position	:	$[_c\ ][_w\ [_w$ `haza menni`$]$ `akarni` $]$
Inversion of `haza menni akarni` with CP	:	$[_w\ [_w$ `haza menni` $]$ `akarni` $]\ [_c\ ]$

The grammar in Fig. B.2 is the original Minimalist Grammar he presents in (Stabler, 1999). With the use of the functional categories `w,c,lc`, the grammar derives the inferences as given above.

```
=lc +m c
=w +c lc
=c v akarni
=w c -c
=v +m w -m
=m v menni
m -m haza
```

Figure B.2: Lexicon of **MG2**

Appendix C

Grail fragments

C.1 Introduction to Grail

Grail is an automated deduction prover for multimodal Categorial Grammars. It is developed as a PhD project at UiL-OTS by Richard Moot. A Tcl-Tk based interface interacts with the theorem prover. A user manual is available and describes the different functions of the window system (Moot, 1998). Within the program one can easily describe a grammar fragment to test and derive grammatical expressions.

A grammar fragment consists of a lexicon and a set of structural postulates. The lexicon is a list of words with type formulas assigned to them and optionally decorated with lexical semantics. The base logic is extended by defining structural postulates, which are used by the theorem prover to derive different structural analysis. On the basis of the grammar fragment one can test and derive grammatical expressions. One can choose between different sorts of output such as the Natural Deduction Prawitz style as used throughout this thesis.

C.2 Fragments in Grail

The grammar fragments developed in chapter 4 are implemented in Grail. This appendix displays the lexicon and the structural postulates that we use to derive the different linguistic expressions in this thesis. Every grammar fragment derives a certain set of grammatical and rejects some ungrammatical expressions.

The different linguistic phenomena that are captured within a grammar fragment are *wh-movement*, *verb modifier climbing* and *verbal inversion* in negative phrases. The search for the right translation of MOVE developed different fragments of the same linguistic phenomena. All are presented here: MOVE *as Structural Reasoning*, MOVE *as abstraction* and the more economical MERGE *over* MOVE.

C.2.1 Wh-movement

The two fragments, `tortillas1.pl` and `tortillas2.pl`, are the result of the mapping between MG and MMCG in section 4.2. Both fragments derive the

following two expressions with the given type assignments:

SENTENCES:

	`tortillas1.pl`	`tortillas2.pl`
Maria making the tortillas	c	c
What tortillas Maria making	$\Box_{wh}c$	c

Move as Structural Reasoning: `tortillas1.pl`

LEXICON:

$\mathsf{c} : c/_{<}v$
$\mathsf{making} : (d\backslash_{>}v)/_{<}d$
$\mathsf{maria} : d$
$\mathsf{the} : d/_{<}n$
$\mathsf{tortillas} : n$
$\mathsf{what} : \Box_{wh}d/_{<}n$

POSTULATES:

$$\Diamond_{wh}(A \bullet_{>} B) \rightarrow \Diamond A \bullet_{>} B \quad [Pwh1]$$
$$\Diamond_{wh}A \bullet_{>} (B \bullet_{>} C) \rightarrow B \bullet_{>} (C \bullet_{<} \Diamond_{wh}A) \quad [Pwh2]$$
$$\Diamond_{wh}A \bullet_{>} (B \bullet_{<} C) \rightarrow B \bullet_{<} (\Diamond_{wh}A \bullet_{>} C) \quad [Pwh3]$$

Move as abstraction: `tortillas2.pl`

LEXICON:

$\mathsf{making} : (d\backslash_{>}v)/_{<}d$
$\mathsf{maria} : d$
$\mathsf{the} : d/_{<}n$
$\mathsf{tortillas} : n$
$\mathsf{what} : (c/_{>}(\Diamond_{wh}\Box_{wh}d\backslash_{>}v))/_{<}n$

POSTULATES:

$$\Diamond_{wh}A \bullet_{>} (B \bullet_{>} C) \rightarrow B \bullet_{>} (C \bullet_{<} \Diamond_{wh}A) \quad [Pwh2]$$

C.2.2 VM-climbing

Section 4.3.4 on page 55 explored three ways to account for VM-climbing. All three grammar fragments are built in Grail. The goal of the three fragments was to derive or withhold the following sentences.

SENTENCES:

	`vm_climb1`	`vm_climb2`	`vm_climb3`
haza akarok menni	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$
$*$ akarok haza menni	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$
$*$ akarok menni haza	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$
haza fogok akarni menni	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$
$*$ fogok haza akarni menni	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$
$*$ fogok akarni haza menni	$\Box_m v_{fin}$	v_{fin}	$\Box_m v_{fin}$

Structural Reasoning: `vm_climb1.pl`

LEXICON:

akarni : $v\_inf/_< v\_inf$
akarok : $v\_fin/_< v\_inf$
fogok : $v\_fin/_< v\_inf$
haza : $\square_m m$
menni : $v\_inf/_< m$

POSTULATES:

$$\diamondsuit_m(A \bullet_> B) \rightarrow \diamondsuit_m A \bullet_> B \quad [K1m]$$
$$\diamondsuit_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (C \bullet_< \diamondsuit_m A) \quad [Mm1]$$
$$\diamondsuit_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (\diamondsuit_m A \bullet_> C) \quad [Mm2]$$

Move as abstraction: `vm_climb2.pl`

LEXICON:

akarni : $v\_inf/_< v\_inf$
akarok : $v\_fin/_< v\_inf$
fogok : $v\_fin/_< v\_inf$
haza : $v\_fin/_>(\diamondsuit_m \square_m m \backslash_> v\_fin)$
menni : $v\_inf/_< m$

POSTULATES:

$$\diamondsuit_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (C \bullet_< \diamondsuit_m A) \quad [Mm1]$$

Merge over Move: `vm_climb3.pl`

LEXICON:

akarni : $v\_inf/_< v\_inf$
akarok : $v\_fin/_< v\_inf$
fogok : $v\_fin/_< v\_inf$
haza : $\square_m(v\_inf/_>(v\_inf/_< m))$
menni : $v\_inf/_< m$

POSTULATES:

$$\diamondsuit_m(A \bullet_> B) \rightarrow \diamondsuit_m A \bullet_> B \quad [K1m]$$
$$\diamondsuit_m A \bullet_> (B \bullet_< C) \rightarrow B \bullet_< (\diamondsuit_m A \bullet_> C) \quad [Mm2]$$

C.2.3 Verbal inversion in negative phrases

The fragments that implement verbal inversion are mappings of the minimalist grammar **MG2**, as explained in section 4.3.4 on page 57. The fragments capture recursive inverted verbal complexes, such as the following expressions:

SENTENCES:

	`neg_inv1`	`neg_inv2`
haza menni akarni	$\square_i w$	w
haza2 menni2 akarni2 kezdeni2	-	w

Structural Reasoning: `neg_inv1.pl`

LEXICON:

akarni : $v/_< c$
$c : c/_< \Box_i w$
haza : $\Box_i m$
menni : $v/_< m$
$w : \Box_i w/_< v$
akarni : $v/_< \Box_i w$
$w1 : w/_< v$

POSTULATES:

$$\Diamond_i(A \bullet_> B) \rightarrow \Diamond_i A \bullet_> B \qquad [K1i]$$
$$\Diamond_i B \bullet_> (A \bullet_< C) \rightarrow A \bullet_< (B \bullet_< \Diamond_i C) \qquad [Mi1]$$

Move as abstraction: `neg_inv2.pl`

LEXICON:

$w^* : w/_>(w/_<\Diamond_i\Box_i w)$
akarni : $w/_< w$
akarni2 : $w^*/_< w$
haza : $w^*/_>(w^*/_<\Diamond_i\Box_i m)$
haza2 : $(w^*/_>(w^*/_<\Diamond_i\Box_i w))/_>((w^*/_>(w^*/_<\Diamond_i\Box_i w))/_<\Diamond_i\Box_i m)$
kezdeni2 : $w/_< w$
menni : $w^*/_< m$
menni2 : $(w^*/_>(w^*/_<\Diamond_i\Box_i w))/_< m$

POSTULATES: -

Bibliography

Bar-Hillel, Y. (1964). *Language and information*, Addison-Wesley, New York.

Chomsky, N. (1995). *The Minimalist Program*, MIT Press, Cambridge, MA. Chapter 4.

Chomsky, N. (1998). Minimalist inquiries: the framework, Draft.

Gazdar, G. (1985). *Generalized phrase structure grammar*, Blackwell.

Heylen, D. (1999). *Types and Sorts. Resource Logic for Feature Checking*, PhD thesis, Universiteit Utrecht.

Koopman, H. and Szabolcsi, A. (1998). Verbal complexes, Draft.

Kurtonina, N. and Moortgat, M. (1996). Structural control, *in* P. Blackburn and M. de Rijke (eds), *Specifying Syntactic Structures*, CSLI Publications, Stanford.

Lambek, J. (1958). The mathematics of sentence structure, *American Mathematical Monthly* **65**: 154–170.

Moortgat, M. (1996). Categorial type logics, *in* J. van Benthem and A. ter Meulen (eds), *Handbook of Logic and Language*, Elsevier and MIT Press, Amsterdam and Cambridge MA, chapter 2, pp. 93–177.

Moortgat, M. and Oehrle, R. (1994). Adjecency, dependency and order, *in* P. D. en Maarten Stokhof (ed.), *Proceedings Ninth Amsterdam Colloquium*, ILLC, Amsterdam, pp. 447–466.

Moot, R. (1996). *Proof nets and labeling for categorial grammar logics*, Master's thesis, University Utrecht.

Moot, R. (1998). User's guide to grail 2.0, ftp://ftp.let.uu.nl/pub/users/moot/.

Morrill, G. (1994). *Type Logical Grammar. Categorial Logic of Signs*, Kluwer, Dordrecht.

Stabler, E. (1996). Derivational minimalism, *in* C. Retoré (ed.), *Logical Aspects of Computational Linguistics*, LNAI Lecture Notes, Springer, Nancy, pp. 68–95. http://128.97.8.34.

Stabler, E. (1998). Eliminating covert movement, Slides with notes for lectures at OTS. Forthcoming report.

Stabler, E. (1999). Remnant movement and structural complexity, *in* Bouma, Hinrichs, Kruijff and Oehrle (eds), *Constraints and Resources in Natural Language*, Syntax and Semantics. Forthcoming.

Versmissen, K. (1996). *Grammatical composition: modes, models, modalitites*, PhD thesis, OTS, University Utrecht.

Zwart, J.-W. (1997). *Morphosyntax of Verb Movement. A Minimalist Approach to the Syntax of Dutch*, Kluwer, Dordrecht.