The Minimalist MOVE operation in a deductive perspective

Willemijn Vermaat

Universiteit Utrecht
OTS, Trans 10, Utrecht
Willemijn Vermaat@let.uu.nl

Abstract

In this paper we analyze the Computational System as described in the Minimalist Program in a multimodal Categorial Grammar (=MMCG) framework. We can define the basic minimalist operations, such as feature checking, MERGE *and* MOVE *using the internal logical and structural reasoning facilities of the categorial system. However, we will show how the analysis of the minimalist* MOVE *operation in a multimodal categorial setting results in a decomposition of* MOVE *into a logical and a structural part. On the logical side, the concept of hypothetical reasoning provides a principled account of* MOVE *as an abstraction operation. On the structural side, structural postulates capture the phenomenon of displacement.*

1 Introduction

In this paper[1] the minimalist theory of Chomsky [1] is studied in the perspective of Multimodal Categorial Grammar [5, MMCG]. In the Minimalist Program, the basic operations of the Computational System are the structure building operations: MERGE and MOVE. MOVE defines the displacement of a phrase in a sentence, which is driven by the need to check uninterpretable features on functional categories. 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.

As indicated by many researchers the minimalist framework as worked out in the Minimalist Program [1] and resource conscious logics such as Multimodal Categorial Grammar [5] show many similarities. Lecomte [4] and Cornell [2] give different approaches to interpret the minimalist mechanisms in MMCG.

[0]Presented at the 11^{th} *European Summer School in Logic, Language and Information, ESSLLI'99*, Utrecht, August 99 as part of the workshop on *Resource Logics and Minimalist Grammars (C.Retoré & E. Stabler, organizers)*

[1]This paper is largely based on my master's thesis [9]. It contains an overview of the two theories that are at the center: *Multimodal Categorial Grammar* and *the Minimalist Program*, which can be read as background for this paper.

In MMCG, MERGE is given straightforwardly by modus ponens as defined in the base logic. The MMCG extensions $\Diamond$ and $\Box$ gives us different possibilities to control the MOVE operation. I consider two approaches: A first approach is to regard movement as a structural operation that captures the phenomenon of displacement. A better approach is to regard 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.

As the Computational System is not formalized in the Minimalist Program. I use the MG framework of Stabler [7] to link the Minimalist Program to multimodal Categorial Grammar. Stabler [7] gives an algebraic framework to capture the main components of the Minimalist Program. Section 2 starts with an explanation of Stabler's algebraic translation of the minimalist framework: *Minimalist Grammar*. Section 2.1 describes in which order the different features occur in the feature specifications of words. Section 2.2 explains how the feature specifications are stored in the lexicon. On the basis of these feature specifications the structure building rules are defined in section 2.3. After every section, a correspondence is made from Stabler's Minimalist Grammar to MMCG which results in a deductive approach of the different minimalist operations in section 2.4. A deductive analysis of MOVE on the basis of its derivational meaning in section 2.5 leads to the right treatment of MOVE as as a complex operation.

2 A computational model of Minimalism

2.1 Features

A central idea in the Minimalist Program [1] is that derivations are feature driven. In the Minimalist Grammar formalism Stabler [7] explains how features trigger structure building operations. He defines the basic operations and the basic objects of these operations: features. With this formalism, we can formulate a minimalist grammar for in principle any kind of language phenomenon.

Features 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 $\mathcal{F}$. Features represent the lexical properties $\mathcal{N}$ of words like its category and other properties such as case, gender and number. Like Stabler [7], we focus on the syntactic features, and abbreviate phonological and semantic feature information. Possible syntactic features are given in the following specification.

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

The syntactic features are divided in 2 groups: *category* features ($\mathcal{N}, =\mathcal{N}$) and *control* features ($+\mathcal{N}, -\mathcal{N}$). The category features state the role of a word in a sentence. Every word gets assigned a category $\mathcal{N}$, 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, $=\mathcal{N}$. The selector feature indicates with what kind

of category a word can be combined. The category features come in pairs: in a derivation, a word with a selector feature is always associated with and accompanied by a word with a matching category feature.

Apart from the category features, the control features play an important role in controlling the order of words and the movement of phrases within structures. The licensee features $-\mathcal{N}$ state certain properties of words, such as $[-\texttt{case}]$, $[-\texttt{wh}]$ and $[-\texttt{tense}]$, while the licensor features $+\mathcal{N}$ indicate the need for such properties. Control features also come in pairs: $[+,-]$. A licensor feature, marked with $[+]$, attracts an identical licensee feature, marked with $[-]$.

The phonological and semantic non-syntactic features are carried along in the derivation (pied-piping), 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 a derivation.

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 [5]. The following formula language is used to build the syntactic type formulas $\mathcal{F}$ of the grammar.

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

Type logical formulas $\mathcal{F}$ are built up from binary and unary connectives and atomic types. $\mathcal{A}$ is the set of atomic formulas, for example the basic types: $\{s, n, np\}$. The logical framework assigns multiple modes of composition; every binary connective is decorated with an index i. In this paper we will distinguish two modes i: $<$ for left-headed composition and $>$ for right-headed composition.

The unary connectives act as control features. For this reason they are refined with features *f* which play an important role in the application of feature checking and movement postulates. The behavior of the unary and binary connectives is given by the residuation laws:

$$\Diamond_f A \vdash B \iff A \vdash \Box_f B$$
$$A \vdash C/_i B \iff A \bullet_i B \vdash C \iff B \vdash A \backslash_i C$$

The behavior of the control features is presented in natural deduction by the introduction and elimination rules of $\Diamond_f$ and $\Box_f$. We will give the natural deduction rules of the binary connectives when we discuss the operations MERGE and MOVE.

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

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

As has been said, 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 MMCG, expressions are given as $\Gamma \vdash A$. The

antecedent Γ is the input, the structured assumptions, which has a certain type A. The type formula A 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 formula. We distinguish an input polarity $^\bullet$ and an output polarity $^\circ$. On the basis of the polarity of the whole type-logical formula, the polarities of the subformulas can be derived with the following rules (and similarly for the $\backslash_>$) :

$$(A/_< B)^\bullet \rightsquigarrow A^\bullet/_< B^\circ \quad \text{and} \quad (A/_< B)^\circ \rightsquigarrow A^\circ/_< B^\bullet$$

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	$c^\bullet$
Selector features	=c	c°: for example $(c\backslash_> -)^\bullet$, $(-/_< c)^\bullet$
Licensee features	[−f]	$\Box_f$ on a formula with polarity $^\bullet$
Licensor features	[+f]	$\Box_f$ on a formula with polarity $^\circ$

Figure 1: Feature correspondence

We take the same basic categories in MMCG for the categorial types as the categories in MG. The logical connectives left $\backslash_>$ 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, licensor and licensee, act as each other's counterpart, indicated by the polarities $[+]$ and $[-]$. Following the polarities of the type-logical formulas, the licensee feature corresponds to the unary connective $\Box_f$ on a formula with input polarity $^\bullet$; in practice the head of a formula will be decorated. The licensor feature, with an opposite polarity, corresponds to the unary connective on a formula with output polarity $^\circ$, which generally means on a subformula with input polarity $^\bullet$ 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 (for more background reading, see Heylen [3]). The licensor feature interacts with the licensee feature. In MMCG the licensor feature is defined as $\Box_f$ on a formula with polarity $^\circ$, which interacts with the structural brackets $\langle . \rangle^f$ on the structural side with polarity $^\bullet$. 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 given in the rule: $\Diamond\Box A \longrightarrow A$

2.2 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 its feature specification which is built up with a syntactic

feature and possibly selector and licensee features. In the MP, *licensor* features are assigned to functional categories; they serve as triggers for the movement of phrases with matching *licensee* features.

Stabler [7] presents the lexical specification as a list of feature occurrences respecting certain constraints. The sequence of features in the list determines the order of application of operations and thus the way 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 (phonological features `/f/` and semantic features $\underline{\mathtt{f}}$) in the case of the lexical categories; functional categories have no phonological feature information. With this feature information, we can build lexical entries.

Lexical correspondence In MMCG, a lexicon exists of lexical entries with a structural label (the headword) and the syntactic formula. Formulas are built up with the grammar rules given in the formula language (see page 3). On the basis of the feature correspondence in Fig. 1, we can compute the lexical correspondence between the syntactic feature specifications in MG and the type assignments in MMCG.

Using the regular expression we can build an algorithm that translates MG syntactic feature structures $\mathcal{F}$ into MMCG type formulas. The different successions of the regular expression correspond with the rules on the left side of the algorithm. The three states: α, β, γ map parts of the feature specification to categorial formulas.

$$\begin{array}{ll}
(\mathtt{f}\ \mathcal{F})^\alpha & = (\mathcal{F})^\gamma\ f \\
(=\mathtt{f}\ \mathcal{F})^\alpha & = (\mathcal{F})^\beta /_< f \\
(=\mathtt{f}\ \mathcal{F})^\beta & = f \backslash_> (\mathcal{F})^\beta \\
(\mathtt{f}\ \mathcal{F})^\beta & = (\mathcal{F})^\gamma\ f \\
(+\mathtt{f}\ \mathtt{g}\ \mathcal{F})^\beta & = (\mathcal{F})^\gamma\ g \text{ with } \Box_f\ g \text{ on a formula with polarity } \circ \\
(-\mathtt{f}\ \mathcal{F})^\gamma & = (\mathcal{F})^\gamma\ \Box_f \\
(\ )^\gamma & = -
\end{array}$$

The phonological and semantic feature information of the lexical categories is incorporated into the label of the logical formula. MG also deals with functional categories that have no non-syntactic feature information; in MMCG we label functional categories with their category feature to indicate their function and position in a structure.

With the algorithm we can compute an MMCG formula type from an MG feature specification. For example:

$$\mathtt{=n\ d\ -wh\ what} \rightsquigarrow \mathtt{what} \vdash \Box_{wh} d /_< n$$

2.3 Structure building operations

Structures $\mathcal{S}$ are built by concatenating lexical and phrasal structures or moving 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 [7] 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.

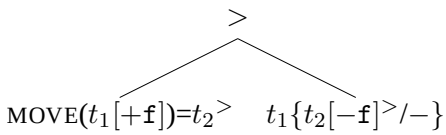
$$\mathcal{S} ::= \mathcal{F} \mid \mathcal{S} < \mathcal{S} \mid \mathcal{S} > \mathcal{S}$$

Two operations are involved with building labeled structures:
MERGE: $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ and MOVE: $\mathcal{S} \rightarrow \mathcal{S}$.

MERGE MERGE combines two trees t_1 and t_2 to form a new tree. Tree $t_1[=\mathtt{c}]$[2], with first feature $= c$, combines with tree $t_2[\mathtt{c}]$ which carries category feature c. The operation MERGE causes the cancellation of the feature $[=\mathtt{A}]$ against $[\mathtt{A}]$. Technically, MERGE can be partitioned into two functions: one that combines with a tree on the right side, $\text{MERGE}_<$: $(<, t_1, t_2)$, and one that combines with a tree on the left, $\text{MERGE}_>$: $(>, t_2, t_1)$. 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 [7] formalizes MERGE as shown in the following tree diagrams.

$$\text{MERGE}(t_1[=\mathtt{c}],t_2[\mathtt{c}]) = \begin{cases} \begin{array}{c} < \\ t_1 \quad t_2 \end{array} & \text{if } t_1 \in \mathit{Lex} \\ \begin{array}{c} > \\ t_2 \quad t_1 \end{array} & \text{otherwise} \end{cases}$$

MOVE 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 feature. The $[-\mathtt{f}]$ 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'.



The tree diagram defines the structure building operation MOVE. MOVE is applied to the *maximal projection*[3] of the subtree carrying the licensee feature $[-\mathtt{f}]$. 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.

[2] $t[\mathtt{F}]$ indicates that F is the prefixed feature of the feature structure of the head of tree t
[3] The maximal projection of subtree $t[-\mathtt{f}]^>$ is the largest subtree with $[-\mathtt{f}]$ as its head.

The definition of MOVE assumes some general constraints on movement as given by Stabler [7]: “all movement is overt, phrasal, leftward”.

2.4 Merge and Move as rules of inferences

Merge as modus ponens Compare the operation MERGE as described in section 2.3, with the modus ponens rules in MMCG. These rules are defined in the natural deduction proof system by the elimination rules. The elimination rules of the binary connectives $\{/_< \text{ and } \backslash_>\}$ capture both partitions of the MERGE operation. Fig. 2 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: leftheaded $\text{MERGE}_<$ and rightheaded $\text{MERGE}_>$, the same holds for the logical operations in MMCG. In minimalism one prefers to reason about one MERGE operation, which is the union of both separated operations:

$$\text{MERGE} = \text{MERGE}_< \cup \text{MERGE}_>$$

Leftheaded MERGE:

$$\text{MERGE}_<(t_1[=\texttt{A}],t_2[\texttt{A}]) \Rightarrow \overset{<}{\underset{t_1 \quad t_2}{\wedge}}$$

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

Rightheaded MERGE:

$$\overset{>}{\underset{t_1 \quad t_2}{\wedge}} \Leftarrow \text{MERGE}_>(t_2[=\texttt{A}],t_1[\texttt{A}])$$

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

Figure 2: MERGE as modus ponens

In the Minimalist Grammar direction arrows: $<$ and $>$ indicate the head of the tree. In MMCG indication of the head is by 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 A first approach is to directly translate Stablers MOVE as Structural Reasoning. To capture the right behavior of this 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 $\Box_f$ on a formula with output polarity $^\circ$, enforces the interaction with the licensee feature via the control diamond, $\langle . \rangle^f$ 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 following postulates can be regarded as general postulates that define movement.

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

The postulates account for the constraints on MOVE as defined by Stabler [7]. The first postulate $[P1]$ captures the constraint that movement is leftward. It checks if a phrase carrying a certain feature f has been moved to the first position. The second and third postulates consider the overt movement of feature decorated phrasal structures. $[P2]$ 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 a sentence where wh-movement occurs. For this case, the undefined feature f in the above postulates is specified as a wh-feature. We derive the structure "What tortillas Maria making" in a *natural deduction* presentation. 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.

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{c} \circ_{<} (\langle \mathsf{what} \circ_{<} \mathsf{tortillas} \rangle^{wh} \circ_{>} (\mathsf{Maria} \circ_{>} \mathsf{making})) \vdash c}{\langle \mathsf{what} \circ_{<} \mathsf{tortillas} \rangle^{wh} \circ_{>} (\mathsf{c} \circ_{<} (\mathsf{Maria} \circ_{>} \mathsf{making})) \vdash c}\,[P3]}{\langle (\mathsf{what} \circ_{<} \mathsf{tortillas}) \circ_{>} (\mathsf{c} \circ_{<} (\mathsf{Maria} \circ_{>} \mathsf{making})) \rangle^{wh} \vdash c}\,[P1]}{(\mathsf{what} \circ_{<} \mathsf{tortillas}) \circ_{>} (\mathsf{c} \circ_{<} (\mathsf{Maria} \circ_{>} \mathsf{making})) \vdash \Box_{wh} c}\,[\Box_{wh}I]}\,[P2]$$

Figure 3: ND derivation with MOVE as structural reasoning

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 implicitly done 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.

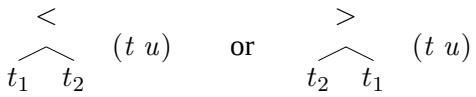
2.5 Derivational meaning of MG operations

To reason about the use of operations in a derivation, one should look at the derivational meaning of such an operation. The derivational meaning of a sentence is accomplished by decorating derivations with terms. A proof term is the 'blueprint' of the logical

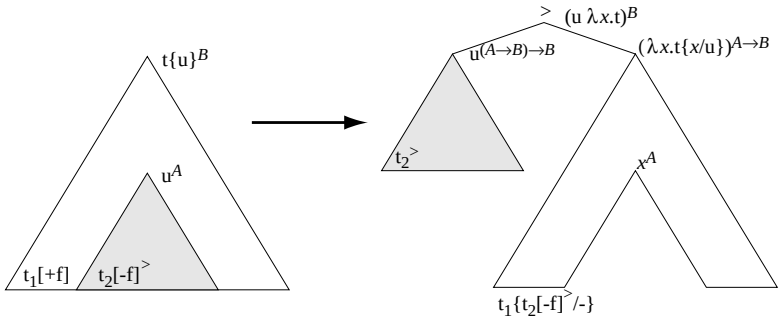
derivation: all logical steps can be read from the proof term. In the same way as the logical rules in MMCG can be decorated, the structure building operations in MG can be labeled with proof terms. The result gives us a comparison between the MMCG proof system and the derivational formalism of Stabler [7].

In his notes, Stabler [8] 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*. The abstraction part needs to be reflected in the proof term.

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



Move as abstraction Stabler [8] indicates that the derivational meaning of the operation MOVE should reflect *abstraction*. To obtain this meaning in Lambda calculus, we decorate the trees of the MG definition of MOVE with terms. The MG definition of MOVE as given on page 6 is repeated more schematically.



The figure shows how the MOVE operation is decorated with terms. First, the two main tree structures are labeled with semantic terms: $t_1[+\mathtt{f}]$ is the whole tree labeled with term t and $t_2[-\mathtt{f}]^{>}$ is the maximal projection of $t_2[-\mathtt{f}]$ labeled with u. Tree $t_2[-\mathtt{f}]^{>}$ is a subtree of tree $t_1[+\mathtt{f}]$. Therefore, the term of the whole tree can be written as $t\{u\}$, 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: $(u\ \lambda x.t\{x/u\})$. This term captures the meaning of MOVE as abstraction.

Every term belongs to a certain type. 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 $A \rightarrow B$. 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 $(A \rightarrow B) \rightarrow B$. But then there is a *type-clash* between the type of term u in the first structure A and the type in the second structure $(A \rightarrow B) \rightarrow B$.

It is not obvious how this problem can be solved in MG. However, the reasoning facilities in MMCG offer us a way to 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 hypothetical phrase. In this way one prevents the type-clash between the two occurrences of the subtree in a derivation.

2.6 Decomposing move

Section 2.4 shows how MOVE is defined in terms of structural control. Using structural reasoning one captures the actual movement of words in a sentence. 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. 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. Hypothetical reasoning is defined by the introduction rules.

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

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 by a higher order type. In our fragment, the lexical entry 'what' 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 captured by integrating the MG lexical feature specifications of the functional category c and the lexical entry what.

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

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 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. In this translation, the feature correspondence given in Fig. 1 still holds. The $\Box_{wh}$ on the subformula d with polarity $^\bullet$ indicates the licensee feature which allows the determiner phrase to

be moved. The licensor feature corresponds to the $\Diamond_{wh}$ on the subformula $(d)^\bullet$, which is the counterpart of a $\Box_{wh}$ on a formula with output polarity $^\circ$.

Apart from the lexical entry 'what' and the functional category c, all other entries stay the same. 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 type-logical formula on the right-hand side.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{what} \vdash (c/_>(\Diamond_{wh}\Box_{wh}d\backslash_>v))/_<n} \quad \dfrac{}{\mathsf{tortillas} \vdash n}\,[/_<E]}{\mathsf{what} \circ_< \mathsf{tortillas} \vdash c/_>(\Diamond_{wh}\Box_{wh}d\backslash_>v)}}{(\mathsf{what} \circ_< \mathsf{tortillas}) \circ_> (\mathsf{maria} \circ_> \mathsf{making}) \vdash c}\,[/_>E]}{\mathsf{maria} \circ_> \mathsf{making} \vdash \Diamond_{wh}\Box_{wh}d\backslash_>v}\,[\backslash_>I]^1}{\mathsf{r}_0 \circ_> (\mathsf{maria} \circ_> \mathsf{making}) \vdash v}\,[\Diamond E]^2}{\langle \mathsf{p}_1\rangle^{wh} \circ_> (\mathsf{maria} \circ_> \mathsf{making}) \vdash v}\,[P2]}{\mathsf{maria} \circ_> (\mathsf{making} \circ_< \langle \mathsf{p}_1\rangle^{wh}) \vdash v}\,[\backslash_>E]}{\mathsf{making} \vdash (d\backslash_>v)/_<d \quad \langle \mathsf{p}_1\rangle^{wh} \vdash d}\,[/_<E]}{[\mathsf{p}_1 \vdash \Box_{wh}d]^2}\,[\Box E]$$

Figure 4: ND derivation with decomposed MOVE

The proof term, ((**what tortillas**) $\lambda x.$((**making** x) **Maria**)) belonging to this derivation, exactly captures the abstraction of the object phrase out of the fully built phrase ((**making** x) **Maria**). The movement of the hypothesized object is still accomplished by means of structural reasoning. The two postulates that were needed in the structural fragment in Fig. 3 for moving a phrase out of its specifier or its complement position are kept. Postulate $[P1]$, responsible for the determination of the position of the abstracted element, is no longer needed. Postulate $[P2]$ accounts for the actual movement in this derivation, in order to retract the hypothesized object.

3 Conclusion

As the previous section shows, it is possible 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.

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. In this case the feature on the goal formula directs the derivation.

Looking at the derivational meaning, MOVE should be regarded as an operation involving abstraction. As seen in section 2.5 the accompanying proof term reflects the meaning of MOVE as an abstraction operation and thus the translation of MOVE with structural reasoning alone does not give the right derivational meaning. Appealing to

hypothetical reasoning, we get the meaning of MOVE as abstraction. This leads to the decomposition of MOVE into structural control and hypothetical reasoning. The control mechanism for this decomposed operation relies solely on the features of the higher order type. There is no longer need for features on the goal formula. The *licensor* and the *licensee* feature [7] are both present on the element which is subject to displacement.

The use of higher order types as triggers for the construction of different structural hierarchies makes the use of functional categories superfluous. The control over the movement of internal phrases and features are defined on the elements involved. Doing so, functional categories are not needed and therefore can be eliminated. The trigger of MOVE becomes 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 results. When fewer lexical elements take part in the derivation, the structural complexity reduces as well.

The determination of MOVE as a complex operation with both structural and logical aspects is the 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, $/$ and $\backslash$. The complexity of a derivation depends on the interaction between the logical and structural aspects of a language.

References

[1] N. Chomsky. *The Minimalist Program*. The MIT Press, 1995.

[2] T. Cornell. A type-logical perspective on minimalist derivations. In G. van Kruijf and R. Oehrle, editors, *Formal Grammar'97*, Aix-en-Provence, 1997. ESSLLI'97.

[3] D. Heylen. Types and sorts: resource logics for feature checking. PhD Thesis, Utrecht University, 1999.

[4] A. Lecomte. Categorial minimalism. In Moortgat [6].

[5] M. Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2, pages 93–178. Elsevier, 1997.

[6] M. Moortgat, editor. *Logical Aspects of Computational Linguistics, LACL'99*. LNCS/LNAI, Springer (to appear).

[7] E. Stabler. Remnant movement and structural complexity. In G. Bouma, H. Hinrichs, G.J. Kruijff, and R. Oehrle, editors, *Constraints and Resources in Natural Language*, Studies in Logic, Language and Information. CSLI Publications, Stanford.

[8] E. Stabler Eliminating covert movement. Slides with notes for lectures at OTS. Forthcoming report.

[9] W. Vermaat. *Controlling Movement: Minimalism in a deductive perspective*. Doctorandus thesis, University of Utrecht, 1999.