Formalizing the minimalist program
Veenstra, Mettina Jolanda Arnoldina

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
1998

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Chapter 9

Interfaces

9.1 Introduction

The language faculty is a set of capacities realized by the human brain. One component of the language faculty enables people to express themselves in a language. The expressions generated are called structural descriptions (SDs). Each SD contains several linguistic properties, such as semantic and phonetic properties. Universal Grammar (UG) is the theory of the initial states of all languages. SDs (or expressions) are part of the theory of a given language. UG specifies different linguistic levels for the different types of linguistic properties in SDs.

As the language faculty includes performance systems, it is possible to use linguistic expressions for actions such as inquiring, interpreting and articulating. Each SD contains information for these performance systems. There are two types of performance systems: articulatory-perceptual (A-P) and conceptual-intentional (C-I). Chomsky refers to the level A-P as Phonetic Form (PF), while the level C-I is referred to as Logical Form (LF).

A derivation consists of a sequence of applications of the operations Merge and Move, and if it converges it yields legitimate LF and PF objects. A derivation tree is legitimate at LF if all features that have to be checked are checked by applications of the operation Move; a derivation tree is legitimate at PF if all strong features are checked by applications of the operation Move.

The definitions of LF in the formalization of Chomsky’s 1993 framework and Zwart’s framework differ slightly. The difference is mainly due to the shift from lexical constituents to functional constituents as indicators for convergence at LF.
In Chomsky’s 1993 framework the formal features (such as tense and case features) of lexical heads and lexical projections must be checked against features associated with functional heads before LF. A derivation crashes if it does not yield at least one LF-object in which all formal features of lexical constituents are checked.\(^1\) Features that are checked are removed from functional heads, as they are not interpretable at LF.

In the Chomsky’s 1995 framework it is required that each functional head checks its formal features before LF. A derivation crashes if it does not yield at least one object (i.e., phrase marker) of which all functional heads that host formal features, ‘attracted’ lexical constituents for feature checking.\(^2\) Zwart adopts this approach in his 1997 framework.

The definition of PF is also different in Chomsky’s 1993 framework and Zwart’s framework. This is due to the fact that Zwart [Zwa97] partly adopts ideas from Chomsky’s 1995 framework about feature checking; isolated formal features are uninterpretable at PF and therefore movement that takes place before PF (i.e. overt movement) must include movement of phonological and semantic features.\(^3\) Zwart applies this idea in the following way to head movement: in principle only formal features of lexical heads are moved for feature checking, and movement of LC-features (i.e. semantic plus categorial features) is a Last Resort for interpretability at PF.\(^4\) For instance, in subject-initial main clauses in Dutch the verb must be spelled out in AgrS. If only the formal features of the verb move to AgrS, the PF-representation of the sentence is incorrect since isolated formal features are uninterpretable at PF. Therefore the LC-features of the verb move directly to AgrS as a Last Resort, that is, without moving to the intermediate landing sites in TP and possibly AgrOP. The word order at PF is now SVO (subject, verb, object), which is correct for Dutch main clauses. In subordinate clauses, the verb’s formal features move to C. If there is a complementizer in C, there are LC-features of the complementizer in C, and no Last Resort movement of LC-features of the verb is required. Therefore the verb is spelled out in situ (i.e. within the VP) in Dutch subordinate clauses. The word order at PF is now CSOV, which is correct for Dutch subordinate clauses (see for instance the examples in Section 3.3).

In the formalization, formal features and LC-features are not moved separately, as this is problematic for copy theory. More specifically, Last Resort movement of LC-features skips intermediate positions (i.e. positions between the lowest position of the chain and the PF-position of the chain) and consequently not all the copies within a chain are identical. As we saw

\(^1\) Cf. [Cho93, Page 29].
\(^2\) Cf. [Cho95, Page 297].
\(^3\) Cf. [Cho95, Page 262].
\(^4\) Zwart assumes that semantic and categorial features belong to the same class since categorial features are derivable from semantic features [Zwa97, Page 169].
in the previous chapter a requirement on chains is that all its copies are identical. Therefore I present a new solution, which is equivalent to Zwart's solution (see Section 9.2).

As opposed to Chomsky, Zwart proposes that phonological features are added after Spell-Out (i.e., postlexicalism [HM93]). Morphology is a post-syntactic (PF) component, and it involves a function mapping bundles of semantic and formal features onto (inflected) forms from a paradigm in the lexicon. The entries selected from the lexicon on the basis of formal and semantic features contain phonological features which represent words that form the sentence associated with the structure that is spelled out.

In the formalization of Zwart's framework, a notion that does not occur in Chomsky's 1993 framework is formalized: the numeration. The idea of the numeration is introduced by Chomsky [Cho95, Page 225]. A numeration is applied to assure that a semantic representation at LF and a phonological representation at PF are based on the same lexical items. A numeration is a set of pairs \((LI, i)\), where \(LI\) is a lexical item and \(i\) is an index indicating the number of times a lexical item is selected. When a lexical item is selected by the derivation the index is reduced by 1. At the end of the derivation all indices of the numeration must be reduced to zero. A semantic representation and a phonological representation are based on the same lexical choices if and only if they are based on the same numeration. In the formalization it was necessary to formulate constraints on numerations (see Section 9.2).

The formalization of Chomsky's 1993 framework with respect to the interfaces LF and PF differs considerably from the formalization of Zwart's Framework. Therefore I give a description of the formalization of interfaces within Chomsky's framework in Section 9.3.

### 9.2 The formalization

In the interface module, LF and PF are the key notions, and therefore \(\text{LogicalForm}_{tr}\) and \(\text{PhoneticForm}_{tr}\) are the key functions of the interface module. The function \(\text{LogicalForm}_{tr}\) yields a boolean value to indicate whether or not a tree \(tr\) is a correct representation at the interface LF. The function \(\text{PhoneticForm}_{tr}\) yields a list of words.

In Zwart's framework, LF is reached when all functional heads checked their formal features.\(^5\) The definition of the function \(\text{LogicalForm}\) can be found in Definition 9.1. As we see in the first conjunct, \(tr\) must be a phrase marker, which implies that \(tr\) must be built according to \(\mathbf{X}\)-Theory and that \(tr\) must have correct chains as we saw in the previous chapter. The

\(^5\) Cf. [Cho95, Page 297] and [Zwa97, Page 184].
second conjunct deals with numerations. I will return to this conjunct after a more elaborate description of the use of numerations in the formalization.

**Definition 9.1**

\[
\text{FUNC} \quad \text{LogicalForm} \quad : \text{TreeS} \quad \to \text{BoolS}
\]

**AXIOM LogicalForm tr**

\[
\langle \Rightarrow \rangle \quad \text{PhraseMarker tr}
\]

\[
\text{And EXISTS num}
\]

\[
( \text{num in NumerationLexicon}
\quad \text{And num Check (FuncHeadFeat tr)}
\quad )
\]

\[
\text{And FORALL nd}
\]

\[
( \quad \text{nd NodeOf tr}
\quad \text{And LocalCheck nd \neq EmptyStruct}
\quad \Rightarrow \quad \text{LinkSource nd \neq Undef}
\quad )
\]

In the formalization numerations are applied in a way different from the way Chomsky [Cho93] and Zwart [Zwa97] apply them (see Section 3.3). Zwart and Chomsky apply numerations to assure that LF and PF are based on the same lexical choices. In the formalization, as we will see below, numerations are mainly applied to make the boolean function \text{LogicalForm} work.

In Chomsky’s 1993 framework, lexical constituents are forced to check all their features before LF. In order to do so, lexical constituents require functional projections to move to. The lexical constituents determine the selection of functional heads. For instance, if a Wh-noun is selected, the structure that is built must contain a C-head to check the Wh-feature of this noun.

In Zwart’s framework, functional heads as opposed to lexical heads require feature checking before LF. If the numeration was not a compulsory component of the framework, structures without any functional heads could make correct sentences, as such a structure does not contain functional heads and hence the LF-requirement is met by definition. This is undesirable because in such a framework it is, for instance, possible to approve of sentences without subject-verb agreement. Namely, if a tree does not contain a head AgrS, it is not possible to move the verb and the subject to check subject-verb agreement. Therefore the formalization requires the numeration to meet certain requirements.

The formalization of Zwart’s framework contains an additional lexicon besides the prelexicon and the postlexicon, which contains abstract representations of possible numerations. The representations are abstract in the sense that they only indicate which functional heads with which feature-value pairs must be present in a numeration for a given type of sentence, for
instance a yes/no-question or an embedded clause. A selection of four of the eight items from the numeration lexicon is given in Definition 9.2. The eight items represent the four sentence types that are dealt with in the formalization: subject-initial main clauses, embedded clauses, Wh-questions and yes/no questions. For each type there are two items: one for the transitive and one for the intransitive version. In Definition 9.2 we find respectively: the transitive version of a subject-initial main clause, the intransitive version of an embedded clause, the intransitive version of a Wh-question and the intransitive version of a yes/no-question. The items in the numeration lexicon are built up in a way that is similar to the way items in a regular lexicon are built up.

Definition 9.2

 OBJ NumerationLexicon : SetS[Numeral Lexicon]
 Axiom NumerationLexicon
 = ( EmptyNumerations
   Add ( EmptyCatsStruct[Agro]
     Add[Subject] ( EmptyStruct
       Add[Agreement] Any[Agreement]
     )
   )
   Add ( EmptyCatsStruct[T]
     Add[Subject] ( EmptyStruct
       Add[Case] Any[Case]
     )
     Add[Tense] Any[Tense]
   )
   Add ( EmptyCatsStruct[Agr]
     Add/Object] Any[Object]
   )
   ...
   ++ ( EmptyNumerations
     Add ( EmptyCatsStruct[C]
       Add[Tense] Any[Tense]
       Add[Sememe] Any[Sememe]
       Add[Subject] ( EmptyStruct
         Add[Agreement] Any[Agreement]
       )
     )
     Add ( EmptyCatsStruct[Agr]
       Add[Object] ( EmptyStruct
         Add[Agreement] Any[Agreement]
       )
     )
   )
In the first item we see that transitive main clauses need three functional heads. For both the subject and the object we need to check case and
agreement features. The agreement features of the verb and the subject are both checked in AgrSP. The case features of the subject are checked in TP, and so is the tense feature of the verb. Both the verb and the object check their case and agreement features in AgrOP. Zwart argues that subject-initial main clauses do not need a C. Therefore C is not added to the items representing main clauses. Example 9.1 contains a phrase marker representing the main clause *De pers volgt de ster* (The press pursues the star) according to Zwart’s ideas.

**Example 9.1**

In the case of a transitive embedded clause (see Example 9.2), there is no functional head with object case and agreement features. The item does contain the functional head C, as embedded clauses need a complementizer. The tense feature of C forces verb movement to C. As we see in Example 9.2 (*dat de pers zwijgt*; that the press keeps silent), the verb is not spelled out in C in embedded clauses since C already contains LC-features of the complementizer (*Add[Sememe] any[Sememe]*)

In the third item (see Example 9.3), Wh-questions are represented. Movement of the subject or the object with Wh-features to the specifier of CP is caused by the Wh-feature of C. Verb movement to the adjunct position of C is caused by the tense feature of C. In Example 9.3 the intransitive Wh-question *Wie zwijgt?* (Who keeps silent?) is represented.
Example 9.2

In the fourth item (see Example 9.4), intransitive yes/no-questions are represented. The tense and the inversion features of C force the inverted verb to move to the adjunct position of C. Movement to the specifier position of C is not allowed as C does not contain a feature that causes this type of movement. In Example 9.4 the yes/no-question *Zwijgt de pers?* (Does the press keep silent?) is represented.
Example 9.4

Now we can return to the second conjunct of Definition 9.1. There we find that there must be a numeration \texttt{num} for \texttt{tr} that is an item in the numeration lexicon, and that it must be possible to check \texttt{num} against the set of feature structures that is built by joining the feature structures of the functional heads of \texttt{tr} (FuncHeadFeat \texttt{tr}), which is also a numeration.\footnote{Note that while Chomsky emphasizes the lexical items in the numeration, our numerations only contain functional heads, as functional heads determine the selection of lexical heads. For instance, if a numeration contains an \texttt{AgrO}, the sentence must contain at least two DPs.} Checking numerations against each other is comparable with checking feature structures against each other (see Chapter 5): the second argument must contain the same functional heads as the first argument and possibly more.

The function \texttt{FuncHeadFeat \texttt{tr}} is defined in 9.3. In this definition we see that only the features of heads that are not of the category D, V or N, are added to the numeration (see second axiom). In the last axiom we see that trees with \texttt{BarLevel 2} that were moved from a lower position are not taken into consideration. The reason for this is that it is redundant to consider all the copies of a given constituent, as copies are by definition the same as their original. Therefore only the lowest element of a chain is taken into account (see fourth axiom: LinkSource = Undef). For heads (with \texttt{BarLevel 0}) we do not have to make the distinction between original and copies, as copies are always subtrees of functional heads. As we saw above, functional heads are dealt with in the second axiom. As soon as a functional category with \texttt{BarLevel 0} is found, its features are added to...
the numeration, and the tree it is the root of is not considered any further. Therefore, the function \texttt{FuncHeadFeat} will never have to deal with copies of lexical heads.

\textbf{Definition 9.3}

\begin{definition}
\begin{align*}
\text{FUNC } & \text{ FuncHeadFeat : } \text{TreeS } \to \text{ NumerationS} \\
\text{AXIOM } & \quad \text{BarLevel tr } = \text{ Undef} \\
& \quad \implies \text{ FuncHeadFeat tr } = \text{ Nil} \\
\text{AXIOM } & \quad \text{BarLevel tr } = \text{ 0} \\
& \quad \text{And } \text{Value[Category] tr } =/= \text{ D} \\
& \quad \text{And } \text{Value[Category] tr } =/= \text{ V} \\
& \quad \text{And } \text{Value[Category] tr } =/= \text{ N} \\
& \quad \implies \text{ FuncHeadFeat tr } \\
& \quad = (\text{Features tr } : : \text{ Nil}) \\
\text{AXIOM } & \quad \text{BarLevel tr } = \text{ 1} \\
& \quad \implies \text{ FuncHeadFeat tr } \\
& \quad = (\text{FuncHeadFeat LeftDaughter tr}) \\
& \quad ++ (\text{FuncHeadFeat RightDaughter tr}) \\
\text{AXIOM } & \quad \text{BarLevel tr } = \text{ 2} \\
& \quad \text{And } \text{LinkSource tr } = \text{ Undef} \\
& \quad \implies \text{ FuncHeadFeat tr } \\
& \quad = (\text{FuncHeadFeat LeftDaughter tr}) \\
& \quad ++ (\text{FuncHeadFeat RightDaughter tr}) \\
\text{AXIOM } & \quad \text{BarLevel tr } = \text{ 2} \\
& \quad \text{And } \text{LinkSource tr } =/= \text{ Undef} \\
& \quad \implies \text{ FuncHeadFeat tr } \\
& \quad = \text{ Nil}
\end{align*}
\end{definition}

Within generative grammar and other linguistic theories abstract trees model phrases. I introduce a lexicalist approach to the modelling of phrases: different types of phrases are described by different feature structures, in a way comparable to the description of lexical items by feature structures. A similar approach to phrases is introduced in HPSG (cf. [Hud90, FK, Kat95, Sag97]).

Finally, we can return to the function \texttt{LogicalForm} in Definition 9.1. The third conjunct says that for every node \texttt{nd} of \texttt{tr} for which \texttt{Functional nd} yields a nonempty feature structure, \texttt{LinkSource nd} must be defined. That is, if a given node \texttt{nd} is a position where lexical constituents can move for feature checking, then this position \texttt{nd} must be filled by movement (of a lexical constituent). This conjunct deals with the requirement that all functional heads must attract lexical constituents.
to check their features before LF, as only certain positions within functional projections yield a nonempty feature structure for the function \textit{FFF\text{Functional}}. Note that the same requirement in reverse order can be found in the definition of the function \textit{FeaturesChecked} (see Definition 8.9 on Page 147). This function is applied within the function \textit{LogicalForm} the same way as it is applied in the function \textit{CorrectChains} which in its turn is applied within the function \textit{PhraseMarker} (see first conjunct in Definition 9.1). The requirement

\begin{align*}
\text{LinkSource}_{\text{nd}} \neq \text{Undef} \implies \text{FFF\text{Functional}} \neq \text{EmptyStruct}
\end{align*}

is completely different from the requirement made in the third conjunct of Definition 9.1. The former is a requirement on phrase markers. It implies that if a constituent moves, its landing site must contain formal features to check against. Hence, movement is impossible without feature checking.

For determining the phonetic form of a tree, the functions \textit{StrongPos} and \textit{SpellOutPos} are relevant. Furthermore, we need three additional functions (\textit{IsSpecialAdjunct}, \textit{IsSpecialLeaf} and \textit{LookUpWord}) in the formalization of Zwart’s framework because of the postlexicalism that is applied in Zwart's approach.

The words that are yielded by PF must in general be related to positions with strong features. Which features are strong differs from language to language. Thus, strength is a parameter determining word order differences between languages. In Zwart’s framework, the features of both the specifier and adjunct position of all functional projections except for TP are strong (see Definition 9.4).

\textbf{Definition 9.4}

\begin{verbatim}
FUNC StrongPos : NodeS \rightarrow BoolS
AXIOM StrongPos nd
  <=
    IsSpecifier nd
    And Value[Category] Mother nd = Agro
  Or
    IsAdjunct nd
    And Value[Category] Mother nd = Agro
  Or
    IsSpecifier nd
    And Value[Category] Mother nd = Agrs
  Or
    IsAdjunct nd
    And Value[Category] Mother nd = Agrs
  Or
    IsSpecifier nd

\end{verbatim}
Before we have a closer look at the definition of Spell-Out positions we will consider the definition of the function \texttt{ChainedTo}, since this function is applied within the definition of Spell-Out positions. The function \texttt{LinkTarget} described in the previous chapter only covers one link of a chain. In the definition of Spell-Out we need to be able to talk about bigger distances within chains. A chain is a sequence of links, and therefore the function \texttt{ChainedTo} is the transitive closure of the function \texttt{LinkTarget}. Both \texttt{ChainedTo} and \texttt{LinkTarget} are defined in the chain module. The function \texttt{LinkTarget} is described in the chapter dealing with the chain module (Chapter 8). The function \texttt{ChainedTo} is described here because the interface module is the only module where this function is applied.

In Definition 9.5 we see that a node \texttt{nd1} is not only ‘chained to’ the next highest node in its chain \texttt{nd2}; it is also ‘chained to’ all other higher nodes of the same chain.

\textbf{Definition 9.5}

\begin{verbatim}
FUNCTION ChainedTo : NodeS, NodeS \rightarrow BoolS
AXIOM nd1 ChainedTo nd2
  \iff LinkTarget nd1 = nd2
  \lor \exists nd3
    ( LinkTarget nd1 = nd3
      \land nd3 ChainedTo nd2
    )
\end{verbatim}

The positions where words can be spelled out are given by the boolean function \texttt{SpellOutPos nd} (see Definition 9.6). These positions are not always equivalent to positions that are strong. If a chain contains more than one strong position, the Spell-Out position of that chain is the highest position with strong features, as all strong features must be checked before PF. If a chain contains no positions with strong features, it is spelled out in situ, that is in the lowest position of the chain. For instance, complementizers enter the derivation by lexical insertion in C. They are not involved in movement for feature checking. Therefore complementizers are spelled out in the lowest position of their chain, which is in this case the only position of its chain.
9.2. THE FORMALIZATION

The first conjunct of the axiom in Definition 9.6 defines that if the node nd1, which is a Spell-Out position, is ‘chained to’ the node nd2, then nd2 is not a node with strong features.

The second conjunct of the axiom in Definition 9.6 defines that the node nd1, which is a Spell-Out position, is either a strong position, namely the highest strong position of the chain (see also Section 3.4), or nd1 is the lowest position of the chain.

Note that nodes nd, which are not part of a chain, are always Spell-Out positions, since for these nodes holds that LinkSource nd1 = Undef.

**Definition 9.6**

\[
\text{FUNC SpellOutPos : NodeS \to BoolS}
\]

\[
\text{AXIOM SpellOutPos nd1} \\
\quad \iff \ \forall nd2 \\
\quad \quad (\ nd1 \ ChainedTo nd2 \\
\quad \quad \Rightarrow \ \neg \ StrongPos nd2 \\
\quad ) \\
\quad \quad \text{And} (\ StrongPos nd1 \\
\quad \quad \text{Or} \ LinkSource nd1 = \text{Undef} \\
\quad )
\]

The boolean function IsSpecialAdjunct nd (see Definition 9.7) defines a node nd that is an adjunct and a Spell-Out position. The Sememe value of its sister must be defined. In this formalization D and C are the only functional heads that can have Sememe features. The Sememe feature is an LC-feature. Verbs or nouns that are adjoined to C respectively D are not spelled out in this position, even though it is a Spell-Out position (see the definition (9.8) of special leaves) as no Last Resort movement of the LC-features of the verb or the noun is required by PF, this because the Spell-Out position already contains LC-features, namely those of C or D.

**Definition 9.7**

\[
\text{FUNC IsSpecialAdjunct : NodeS \to BoolS}
\]

\[
\text{AXIOM IsSpecialAdjunct nd} \\
\quad \iff \ IsAdjunct nd \\
\quad \quad \text{And} \ SpellOutPos nd \\
\quad \quad \text{And} \ Value[Sememe] \text{ Sister nd } =/= \text{Undef}
\]

The function IsSpecialLeaf nd (see Definition 9.8) defines a head that remains in situ as its Spell-Out position is already occupied by another Sememe (LC-)feature (see the definition of special adjunctions (9.7)).
A node nd1 has to meet two requirements to be a special leaf. Firstly, nd1 must be a leaf. Secondly, nd1 must be in one chain with a special adjunct nd2. Hence, a lexical head is spelled out in situ, if its Spell-Out position is occupied by the LC-features of a functional head. Therefore, we can conclude that the Phonetic Form of a given tree does not only depend on Spell-Out positions in the formalization Zwart's framework. The presence of special leaves and special adjuncts also plays an important role, as we will see in the definition of the function PhoneticForm in Definition 9.10.

**Definition 9.8**

```
FUNC IsSpecialLeaf : NodeS -> BoolS
AXIOM
IsSpecialLeaf nd1
<=> IsLeaf nd1
And EXISTS nd2
    ( nd1 ChainedTo nd2
    And IsSpecialAdjunct nd2
    )
```

Before we take a closer look at the function PhoneticForm we will consider the function LookUpWord in Definition 9.9. The function LookUpWord takes a feature structure and yields a string, i.e., a word. This function is applied to consult the postlexicon at PF. In Zwart's framework, the postlexicon must be consulted after the derivation because the regular lexicon does not contain phonological features. Hence, during the derivation no phonological features are present in the tree. As phonological features determine the word that is spelled out at PF, the function LookUpWord is indispensable within the function PhonologicalForm.

In Definition 9.9 fstruct1, fstruct2 and fstruct3 are feature structures. str is a string, or rather a word. If we want to look up the word belonging to a feature structure fstruct1 in a given tree, there must be a feature structure fstruct2 in the postlexicon which can be checked against fstruct1, if we remove the word feature from fstruct2. This implies that fstruct1 contains the same feature-value pairs as fstruct2 and possibly more. If there is a feature structure fstruct3 in the postlexicon which also can be checked against fstruct1 (if we remove the word feature of fstruct3), then it must be possible that fstruct3 is checked against fstruct2. The word feature does not have to be removed from fstruct2 and fstruct3 in the latter case, as both feature structures contain a word feature since they both come from the postlexicon. fstruct1 comes from the regular lexicon, and therefore it does not contain a word feature.

The function LookUpWord effects that we will always select the most specific lexical item from the postlexicon. If there are two feature structures
fstruct2 and fstruct3, that can be checked against fstruct1, we will select the feature structure (fstruct2) that contains most feature-value pairs.

**Definition 9.9**

\[
\text{fstruct/2} \text{ and } \text{fstruct/3}, \text{ that can be checked against fstruct/1, we will select the feature structure (fstruct/2) that contains most feature-value pairs.}
\]

\[
\text{De/0Cnition } \text{LookUpWord} : \text{FeatureStruct} \rightarrow \text{PARTIAL String} \text{S}
\]

\[
\text{AXIOM } \text{LookUpWord fstruct/1 = str}
\]

\[
\iff \text{EXISTS fstruct/2}
\]

\[
( \text{Value[Word] fstruct/2 = str}
\]

\[
\text{And fstruct/2 In PostLexicon}
\]

\[
\text{And (fstruct/2 Remove Set(Word)) Check fstruct/1}
\]

\[
\text{And FORALL fstruct/3}
\]

\[
( \text{fstruct/3 In PostLexicon}
\]

\[
\text{And (fstruct/3 Remove Set(Word))}
\]

\[
\text{Check fstruct/1}
\]

\[
\implies fstruct/3
\]

\[
\text{Check fstruct/2}
\]

\]

The function PhoneticForm tr in Definition 9.10 yields phrases. The tree tr is searched from left to right and, in principle, all leaves that are Spell-Out positions (and have a Sememe value) are spelled out by picking the right feature structure fstruct from the postlexicon and putting its word value at the end of the phrase. Thus, Zwart’s postlexicalism is represented in the function PhoneticForm tr. The phonological features, which are not present in the derivation (i.e. in the tree), are added after Spell-Out by selecting the most specific feature structure from the postlexicon.

The third and the fifth axiom in Example 9.10 are the only two that add words to the phrase that PhoneticForm tr yields. Conditions for adding words are:

- tr must be a Spell-Out position or a special leaf (i.e. a leaf which occurs in one chain with an adjunct that has a sister with LC-features).
- tr must be a leaf (possibly a special leaf).
- If tr is a Spell-Out position, tr may not be a special adjunct and tr must have a Sememe feature.

The only way for a tree tr to yield words is by being a Spell-Out position or by being a special leaf. Hence, for trees that are not Spell-Out positions or special leaves we do not add any words to the phrase (see first axiom). But there are more reasons not to add any words to the phrase for a given tree tr. Firstly, trees which are Spell-Out positions but also special adjuncts
are not spelled out (see second axiom). Secondly, trees which are Spell-Out positions and leaves but which do not have a Sememe feature are not spelled out for the obvious reason that they cannot be connected with a word feature (see fourth axiom).

In the fifth axiom we see that trees which are Spell-Out positions, but which are not a special adjunct or a leaf, are spelled out by applying the function PhoneticForm to their left and right daughters.

**Definition 9.10**

\[
\text{FUNC} \quad \text{PhoneticForm} :: \text{TreeS} \rightarrow \text{PhraseS}
\]

\[
\text{AXIOM Not SpellOutPos tr} \\
\quad \text{And Not IsSpecialLeaf tr} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} = \text{Nil}
\]

\[
\text{AXIOM IsSpecialAdjunct tr} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} = \text{Nil}
\]

\[
\text{AXIOM IsSpecialLeaf tr} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} \\
\quad = \text{(LookUpWordFeatures tr)} :: \text{Nil}
\]

\[
\text{AXIOM SpellOutPos tr} \\
\quad \text{And IsLeaf tr} \\
\quad \text{And Value[Stem] tr} = \text{Undef} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} = \text{Nil}
\]

\[
\text{AXIOM SpellOutPos tr} \\
\quad \text{And Not IsSpecialAdjunct tr} \\
\quad \text{And IsLeaf tr} \\
\quad \text{And Value[Stem] tr} /= \text{Undef} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} \\
\quad = \text{(LookUpWordFeatures tr)} :: \text{Nil}
\]

\[
\text{AXIOM SpellOutPos tr} \\
\quad \text{And Not IsSpecialAdjunct tr} \\
\quad \text{And Not IsLeaf tr} \\
\quad \Rightarrow \quad \text{PhoneticForm tr} \\
\quad = \text{(PhoneticForm LeftDaughter tr)} ++ \text{(PhoneticForm RightDaughter tr)}
\]

### 9.3 Interfaces in Chomsky’s framework

In the interface module of the formalization of Chomsky’s 1993 framework both the definitions of LF and PF differ considerably from those in the formalization of Zwart’s framework,
There are two aspects that determine whether a tree \( tr \) is a correct LF-representation. These two aspects are represented in the formalization of LF given in Definition 9.11.

Firstly, the tree in question must be a phrase marker, i.e., it must obey X-Theory and its chains must be correct according to the Minimalist Program (see Definition 8.10). Secondly, all features that need to be checked before LF must be checked.

The second aspect needs some discussion. As we saw earlier in this chapter, an LF-representation may not contain lexical constituents with unchecked features. That is, all lexical constituents with root \( nd \) that are the highest constituent of the chain they belong to (\( \text{LinkTarget } nd = \text{Undef} \)) need to have checked all their features that had to be checked (\( \text{FFLexical } nd = \text{Checked } nd \)). This also holds for nodes that do not have to check features (\( \text{FFLexical } nd = \text{Undef} \)), as these nodes cannot move and therefore do not check features (\( \text{Checked } nd = \text{Undef} \)). The function \( \text{Checked} \) is not applied within Zwart's framework. This function is given in Definition 9.12.

**Definition 9.11**

\[
\begin{align*}
\text{FURC} & \quad \text{LogicalForm} : \text{TreeS} \rightarrow \text{BoolS} \\
\text{AXIOM} & \quad \text{LogicalForm } tr \\
& \iff \text{PhraseMarker } tr \\
& \quad \text{And } \forall nd \\
& \quad \quad (nd \ \text{NodeOf } tr \\
& \quad \quad \quad \quad \text{And } \text{LinkTarget } nd = \text{Undef} \\
& \quad \quad \quad \quad \quad \quad \quad \iff \text{FFLexical } nd = \text{Checked } nd)
\end{align*}
\]

According to Chomsky [Cho93] features are deleted when they are checked, and LF is reached when there are no formal features left. The function \( \text{Checked } nd \) which yields a structure containing the features that have been checked via movement of the constituent of which \( nd \) is the root (see Definition 9.12). In the lowest position of a chain (\( \text{LinkSource } nd = \text{Undef} \)), no features are checked (\( \text{Checked } nd = \text{EmptyStruct} \)). For every other position \( nd \) in the chain holds that checked features are the features that were checked until the preceding position in the chain (\( \text{Checked } \text{LinkSource } nd \)) unified with the features that are checked in \( nd \) (\( \text{FFFunctional } nd \)).

\[\text{7See Appendix for the definition of Unify. This definition is a part of the feature structure module, but it is only discussed here because this is the only place in the formalization where this function is applied.}\]
Definition 9.12

\[
\text{FUNC} \quad \text{Checked : NodeS} \rightarrow \text{FeatureStructS}
\]

\[
\text{AXIOM} \quad \text{LinkSource nd} = \text{Undef} \\
\implies \text{Checked nd} = \text{EmptyStruct}
\]

\[
\text{AXIOM} \quad \text{LinkSource nd} \neq \text{Undef} \\
\implies \text{Checked nd} = (\text{Checked LinkSource nd}) \text{Unify (LocalCheck nd)}
\]

The function \texttt{PhoneticForm tr} searches \texttt{tr} and yields a list of words. The positions that are strong in Dutch are indicated by the boolean function \texttt{StrongPos nd} (see Definition 9.13). In the formalization of Chomsky’s 1993 framework the features in the specifier position of AgrS and AgrO are strong. There are no adjunct positions with strong features.

In Chapter 3 we saw that Chomsky’s framework is not adequate to cover verb movement in Dutch. The parameters given in Definition 9.13 only yield a correct word order for embedded clauses in Dutch. Main clauses and Wh-questions as in Example 9.5 ((a) respectively (b)) do not get the right word order in Dutch. Yes/no-questions are not dealt with in the formalization of Chomsky’s framework.

Definition 9.13

\[
\text{FUNC} \quad \text{StrongPos : NodeS} \rightarrow \text{BoolS}
\]

\[
\text{AXIOM} \quad \text{StrongPos nd} \\
\iff \begin{cases} 
\text{IsSpecifier nd} \\
\text{And Value[Category] Mother nd} = \text{AgrO} 
\end{cases} \\
\text{Or} \begin{cases} 
\text{IsSpecifier nd} \\
\text{And Value[Category] Mother nd} = \text{AgrS} 
\end{cases}
\]

Example 9.5

(a)  *De politieagent de politicus arresteert*  
The policeman the politician arrests
(b)  *Wie de politicus arresteert*  
Who the politician arrests

The function \texttt{SpellOutPos nd} (see Definition 9.6 in the previous section) is the same as in the formalization of Zwart’s framework.
The function \texttt{PhoneticForm} \( \text{tr} \) yields a phrase \( \text{ph} \). The tree \( \text{tr} \) is searched from left to right and all leaves that are Spell-Out positions are spelled out by putting the word values of those positions (if they have any) in a list.

In the first axiom in Definition 9.14 it is defined that trees that are not Spell-Out positions do not yield a word. Spell-Out positions that are leaves without a word value, for instance the functional head \texttt{AgrS}, do not yield a word either, as we see in the second axiom. Spell-Out positions that are leaves with a word value yield a word which is concatenated to the list of words yielded so far, as we see in the third axiom. In the fourth axiom we see that, if a tree \( \text{tr} \) is a Spell-Out position but is not a leaf, the phonetic form of \( \text{tr} \) is the phonetic form of its left daughter joined with the phonetic form of its right daughter.

**Definition 9.14**

**FNC** \( \text{PhoneticForm} : \text{Trees} \rightarrow \text{Phrases} \)

**AXIOM** \( \text{Not SpellOutPos tr} \)
\[ \Rightarrow \text{PhoneticForm tr} = \text{Nil} \]

**AXIOM** \( \text{SpellOutPos tr} \)
\[ \text{And IsLeaf tr} \]
\[ \text{And Value[Word] tr} \neq \text{Undef} \]
\[ \Rightarrow \text{PhoneticForm tr} = \text{Nil} \]

**AXIOM** \( \text{SpellOutPos tr} \)
\[ \text{And IsLeaf tr} \]
\[ \text{And Value[Word] tr} \neq \text{Undef} \]
\[ \Rightarrow \text{PhoneticForm tr} = (\text{Value[Word] tr}) :: \text{Nil} \]

**AXIOM** \( \text{SpellOutPos tr} \)
\[ \text{And Not IsLeaf tr} \]
\[ \Rightarrow \text{PhoneticForm tr} = (\text{PhoneticForm LeftDaughter tr}) \]
\[ ++ (\text{PhoneticForm RightDaughter tr}) \]

**Summary**

There are two reasons why the formalization of the interface levels LF and PF are less straightforward in Zwart’s framework than the formalization of the interface levels in Chomsky’s 1993 framework.

Firstly, the switch in focus from lexical to functional heads at LF results in some extra requirements at LF. Since LF in Zwart’s framework is reached
when all functional heads in a tree checked all their features, I needed something to make sure that an LF-tree always contains the required functional projections. Otherwise an LF-tree consisting of only a VP could be approved of by the formalization. In Chomsky’s 1993 framework lexical heads needed to check all their features before LF, and to check their features they require functional heads. In Zwart’s approach I introduced an additional lexicon (besides the pre- and the postlexicon) which contains templates for all the types of sentences covered by the formalization. The templates contain the required functional heads for the given sentences, including the required features for the case in question.

Secondly, Zwart’s postlexicalism puts some extra requirements on the formalization. It is more complicated to determine the PF-position of a chain because functional heads with lexical content can cause a lexical head to remain in situ although it contains one or more strong positions. At PF, isolated formal features are assumed to be uninterpretable and if the functional projection the PF-position is part of does not contain any LC-features, then the LC-features of the lexical head in question need to move to the PF-position as a Last Resort. But if the ‘usual’ PF-position of the lexical head already contains LC-features because the functional projection it is part of contains phonological material, then the LC-features of the lexical head remain in situ, which implies that the lexical head is spelled out in situ.