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 8

Chains

8.1 Introduction

Chains in the formalization represent what is called movement in the Minimalist Program. There, movement is connected with feature checking. A constituent moves only if it is forced to do so because of the need of feature checking. A moving constituent is ‘attracted’ to the constituent it moves to when the both constituents have similar features. The moving constituent is always a lexical projection or a lexical head. The landing site is always a node within a functional projection (such as the specifier of AgrSP or the position adjoined to the head of TP). Features are divided into three classes: formal, semantic and phonological. The formal features are the only class of features that require feature checking. A lexical constituent may check its formal features in different functional projections, so that it can move more than once. In each functional projection where it lands, it must check at least one feature. The overlap in features of the moving constituent and the landing side must therefore concern at least one joint feature-value pair. The process of movement of lexical constituents to functional projections with overlapping features is called feature checking.

The feature checking requirement described above can be derived from the Economy of Derivation [Cho91]. Zwart [Zwa96, Zwa97] considers it to be a consequence of the Economy of Derivation that the derivation must involve as few steps as possible. A way to restrict the number of movements is to allow only movements which involve feature checking.

There are more constraints on movement in the minimalist theory:

- Movement is only allowed to specifier positions of functional heads and positions adjoined to heads of functional projections (i.e. the checking domain).
• The moving element must originate from a position reflexively dominated by the complement (i.e., the complement domain) of the functional head whose checking domain the element moved to.

• XPs are only allowed to move to specifier positions, since the specifier position is the position where the type of formal features (NP-features, cf. [Cho93, Page 28ff]) that XPs are associated with can be checked. Xs are only allowed to move to positions adjoined to heads, since the specifier position is the position where the type of formal features (V-features, cf. [Cho93, Page 28ff]) that Xs are associated with can be checked.\footnote{Cf. Uniformity of Attachment [Cor97, Page 18].}

Chomsky [Cho93] assumes that the Economy of Derivation not only implies that the derivation involves as few steps as possible, but also that the steps in the derivation are as short as possible. That subjects must move from the specifier of the VP to the specifier of the AgrSP skipping the specifier of AgrOP is problematic from this point of view. Therefore, Chomsky [Cho93, Page 17] introduces the notion ‘equidistance’, of which the definition will not be given here for the sake of simplicity. The notion ‘equidistance’ makes it possible to consider the movement from position P3 to position P1 in Example 8.1 to be the shortest possible movement, even though a possible candidate landing site (P2) which is closer to P3 is skipped. Namely, if two positions are equidistant according to Chomsky’s definition, a movement to either the higher or the lower position of the two can be referred to as the shortest move.

Example 8.1

\[
\begin{align*}
P_1 \\
\downarrow \\
P_2 \\
\downarrow \\
P_3
\end{align*}
\]

Zwart [Zwa96] considers the ‘shortest steps and fewest steps requirement’ to be contradictory, since shorter steps in a derivation will result in more steps and fewer steps in a derivation will result in longer steps. Zwart therefore argues that Economy of Derivation only involves the fewest steps requirement. As we saw earlier in this section, the fewest steps requirement is connected with the feature checking requirement.
The fact that there is no shortest steps requirement in Zwart’s 1997 framework makes the notion ‘equidistance’ (which turns out to be problematic in many cases, see for instance [Zwa97, Page 238ff]) superfluous. But still one needs a way to prevent the subject (i.e. the specifier of the VP) from moving to the specifier of AgrOP, and the object (i.e. the complement of the VP) moving to the specifier of TP and AgrSP as in Example 8.2. Although the PF-representation of the structure is correct, this is not the kind of structure we want to be described by the formalization (because of the confused indices). Since no satisfying solutions to this problem have been presented for Dutch, I solved this problem in a trivial way in the formalization: all the movements that are needed are given and all other movements are considered to be incorrect (see Section 8.2).²

Example 8.2

In the next section I will give the formalization which is a representational interpretation of the derivational constraints given above. Rather than describing movements that build chains, I describe connections between two nodes that build chains.

As we saw in Chapter 3, the choice of a representational approach forces us to apply copies instead of traces to indicate that an element moved from

²For a solution regarding English see Stabler [Sta06, Page 115], who applies a simple version of an idea from Ferguson and Groat [FG94] in one of his formalizations.
one position to another. It is crucial that we check that all the copies within a chain are in fact copies of the original that was introduced in the structure by lexical insertion.\(^3\) Whether two positions in a tree may be connected by a chain is given in the formalization. But we need an extra constraint which only allows chain links between exact copies. This constraint is not needed in a derivational approach since movement already implies that the linked positions are associated with the same content.

### 8.2 The formalization

In Section 8.1 I mentioned that movement is connected with feature checking in the Minimalist Program. In the formalization, which is a representational version of the Minimalist Program, movements are represented by chains, that is, sequences of connected nodes. Feature checking is represented by a constraint which concerns all the copies in a chain, that is all elements that build the chain except for the original in the lowest position of the chain where lexical insertion takes place. This constraint implies that a copy must have formal features which can be checked against the formal features of the functional projection it is part of.\(^4\) Feature checking succeeds if the lexical constituent contains at least the same formal features as the functional projection it is part of. The reason why only the copies (and not the original) in a chain are associated with feature checking lies in the fact that only the copies in a chain can be compared with moved elements in the Minimalist Program.

At this point I give a brief overview of the functions specified in the chain module of the formalization. After that I will give the formalization of the different functions in combination with a more detailed description.

To be able to check the formal features of the lexical constituent against those of the functional projection, we have to separate the formal features from the rest of the features (i.e., semantic and phonological features) in a feature structure. The formal features of lexical constituents are isolated by the function \(\text{FFLexical} \ \text{md}\) (where ‘FF’ stands for ‘formal features’); the formal features of the functional projection are isolated with the function \(\text{FFF} \ \text{Functional} \ \text{md}\).

In the Minimalist Program the formal features of the functional heads are split up in two categories: NP-features and V-features. NP-features are checked by movement to the specifier of the functional head, and V-features are checked by movement to the position adjoined to the functional head. In the formalization presented here, NP-features and V-features are not ex-

---

\(^3\)See also Brody [Bro95, Page 139].

\(^4\)As we saw in the previous section, the copy may either stand in the specifier position of the functional head or in the position adjoined to the functional head.
8.2. THE FORMALIZATION

Explicitly defined as NP-features and V-features. The function \texttt{FunctionalEnd} yields feature structures that represent what is referred to as NP-features and V-features within the Minimalist Program. The type of features that are checked in a position \texttt{nd} depends on whether \texttt{nd} is a specifier position or a position adjoined to a head.

The actual feature checking is performed by the boolean function \texttt{FeaturesChecked tr}. With this function we can determine whether every lexical constituent in the tree \texttt{tr}, which is part of a chain and which is not the lowest position of that chain, can check its formal features against those of the functional projection it is part of.

Whether or not a constituent is part of a chain, and, if so, whether or not it is the lowest element in the chain, can be determined by the function \texttt{LinkSource nd1}. This function indicates links between nodes in a tree. \texttt{LinkSource nd1 = nd2} means that there is a link from a node \texttt{nd2} (the source) to the node \texttt{nd1} (the target) as we see in Example 8.3. Hence, both nodes are the roots of an element in a chain. Since movement in the Minimalist Program is always upward and leftward, the source (\texttt{nd2}) is always lower than and to the right of the target (\texttt{nd1}). If \texttt{LinkSource nd} is undefined, we know that either \texttt{nd} is the root of the lowest element of a chain (since it has no source, see for instance \texttt{nd3}), or the constituent that has \texttt{nd} as a root is not an element of a chain at all.

Example 8.3

As we saw in Section 8.1, the requirement that feature checking must occur is not the only constraint on movements. Further constraints are, firstly, that the source of the movement is in the complement domain, while the target of the movement is in the checking domain of a given functional head, and secondly, that XPs only can move to specifiers while Xs only can move
to positions adjoined to a head. In Section 8.1 I argued that I need an additional constraint to prevent subjects from moving to the specifier of AgrO. All these constraints, plus the boolean function `FeaturesChecked` which I mentioned earlier this section, are united in the function `CorrectChains` which guarantees that all chains in the tree `tr` are built according to the Minimalist Program, as the name of the function already suggests.

Now I will proceed to the more elaborate descriptions of the functions given above.

The function `FFLexical nd` (see Definition 8.2) yields a feature structure that contains the formal features that the node `nd` inherited from the lexicon. All other features, such as semantic features, are left aside. For instance, the lexical item for the Dutch verb form `zie` (sees), given in Definition 8.1, is a feature structure containing a `Category` feature, a `Sememe` feature, a `Subject` feature (which contains the agreement features for the subject selected by the verb), an `Object` feature (which contains the agreement features for the object selected by the verb), a `Tense` feature, a `CompCat` feature (for subcategorization with respect to the complement) and a `SpecCat` feature (for subcategorization with respect to the specifier). The only features ending up in the feature structure that `FFLexical nd` yields are the formal features `Object`, `Subject`, `Tense` and `Inversion`.

**Definition 8.1**

```latex
MODULE Lexicon
IMPORT LexiconCommon
AXIOM Lexicon = .

++ ( EmptyCatStruct[V]
  Add[Sememe] "zie"
  Add[Subject] ( EmptyStruct
    Add[Agreement] ( EmptyStruct
      Add[Person] Third
      Add[Number] Singular
      Add[Gender] Masculine
    )
  )
  Add[Object] ( EmptyStruct
    Add[Agreement] ( EmptyStruct
      Add[Person] First
      Add[Number] Plural
      Add[Gender] Neuter
    )
    Add[Case] Accusative
  )
)```
8.2. THE FORMALIZATION

Add[Tense] Present
Add[Inversion] No
Add[CompCat] D
Add[SpecCat] D

END MODULE

It would have been possible to split up the features in the formalization into two different types: formal and nonformal (i.e., phonological and semantic). In that case the function FFLexical nd (see Definition 8.2) would have looked more elegant than it does now, as it would not have been necessary to sum up all features that have to be checked. Just referring to the formal features of the node nd would have been sufficient. The reason why this solution is not preferred is that the formalization in Definition 8.2 makes explicit which features have to be checked if they are present in the relevant lexical item. The formalization never literally refers to formal, semantic or phonological features. But in the description of the formalization I will always be explicit about the types of features we are dealing with in a given function.

FFLexical nd only yields a nonempty feature structure for nodes that are verbs of bar-level 0, nouns of bar-level 0, and determiners of bar-level 2, as these represent the lexical constituents that may move for feature checking. Definition 8.2 defines that only the tense, inversion, subject and object features of a verb are formal features. The first axiom in Definition 8.2 literally says that for a node nd with bar-level 0 and category V, FFLexical nd yields the features of nd of which we obtain only the tense, inversion, subject and object features. For instance, a verb may adjoin to AgrO to check its object agreement, then to T to check its tense feature, then to AgrS to check its subject agreement, and finally to C to check its inversion feature.

Determiners (Ds) of bar-level 2 must check their agreement, case and Wh-features, as we see in the second axiom. For instance, the subject DP may move to the specifier of T to check its case feature, then to the specifier of AgrS to check its agreement feature, and finally to the specifier of C to check its Wh-feature.

In the third axiom we see that nouns of bar-level 0 must check their determiner, agreement, case and Wh-features. Later on in this section I will argue why and how nouns must check these features.

Certain verbs, nouns and determiners do not have all the features that are named in the function FFLexical. For instance, intransitive verbs do not have an object feature. In that case the feature structure that FFLexical
nd yields only contains subject, inversion and tense features.

In the fourth axiom we see that if nd is an adjunction (i.e. the mother of an adjunct), FFLexical nd yields the same feature structure as its adjunct. In this way the formal features of a moving head can percolate up to the root of the adjunction structure the moving head is part of. Hence, FFLexical nd, where nd is for instance the upper AgrS in Example 8.4, yields the formal features of the V at the bottom of the structure. Since nd (AgrS) is an adjunction, FFLexical nd equals FFLexical Adjunct nd. The adjunct of AgrS is T, which in its turn is also an adjunction. Eventually, via AgrO, we find that FFLexical nd yields the formal features of V.

Example 8.4

![Diagram of a tree structure]

For all the remaining nodes nd in a tree, FFLexical nd yields an empty feature structure as we see in the fifth axiom.

Definition 8.2

**FUNC** FFLexical : NodeS → FeatureStructS

**AXIOM** BarLevel nd = 0
And Value[Category] nd = V
⇒ FFLexical nd
   = (Features nd) Keep Set(Subject, Object, Tense, Inversion)

**AXIOM** BarLevel nd = 2
And Value[Category] nd = D
⇒ FFLexical nd
   = (Features nd) Keep Set(Agreement, Case, WhWord)

**AXIOM** BarLevel nd = 0
And Value[Category] nd = N
⇒ FFLexical nd
   = (Features nd)
8.2. THE FORMALIZATION

Keep Set (Determiner, Agreement, Case, WhWord)

**AXIOM**

IsAdjunction nd

\[ \Rightarrow \text{FFlexical nd} \]

= FFlexical Adjunct nd

**AXIOM**

Not ( BarLevel nd = 0

And Value[Category] nd = V

)

And Not ( BarLevel nd = 2

And Value[Category] nd = D

)

And Not ( BarLevel nd = 0

And Value[Category] nd = N

)

And Not IsAdjunction nd

\[ \Rightarrow \text{FFlexical nd} \]

= EmptyStruct

The function **FFfunctional nd** yields a feature structure containing the features that can be checked by movement to nd. As we saw earlier, feature checking can only take place in positions within functional projections (such as AgrSP and TP). Definition 8.3 shows that only the adjuncts and specifiers of functional projections (i.e., the checking domain) are available for feature checking. Hence, the formal features of functional heads can only be checked in their checking domain.

At this point I will give a brief explanation of the axioms in Definition 8.3. Next, a more elaborate description will be given of some unusual cases.

In the specifier position of C (nd is a specifier and the category of its mother is C), the Wh-feature of the subject or object DP can be checked against the Wh-feature of the head of CP since **FFfunctional nd** yields the Wh-feature (WhWord) of the functional head C (first axiom).\(^5\) In the adjunct position of C, tense, inversion and subject agreement features of the verb can be checked (second axiom). In the specifier position of AgrSP the agreement feature of the subject DP can be checked against the subject agreement feature of the head of AgrSP (third axiom). In the adjunct position of AgrSP the subject agreement features of V can be checked against the subject agreement features of the head of AgrSP (fourth axiom). In the specifier position of TP the case feature of the subject DP can be checked against the case feature of the head of TP (fifth axiom). In the adjunct position of TP the tense feature of V can be checked against the subject case features and the tense feature of the head of TP (sixth axiom). In the specifier position of AgrOP the case and agreement features of the object DP can be checked against the object feature (including case and agreement) of

---

\(^5\)Note that the function Keep takes a feature structure and a set of feature names, and yields another feature structure which is the first feature structure of which the feature-value pairs of which the feature names are mentioned in the set are kept.
the head of AgrOP (seventh axiom). In the adjunct position of AgrOP the object feature (including case and agreement) of V can be checked against the object features of the head of AgrOP (eighth axiom). In the specifier position of DP no features are checked (ninth axiom), but it is mentioned explicitly since the specifier position of DP in principle is a checking position because D is a functional category. In the adjunct position of DP the determiner, case, agreement and Wh-features of N can be checked against the determiner, case, agreement and Wh-features of the head of DP (tenth axiom). All non-checking positions, which also yield an empty feature structure, are dealt with in the eleventh axiom (i.e. the closure of the function $\text{Functional}$).

**Definition 8.3**

\[
\begin{align*}
\text{FUNC } & \text{Functional} : \text{NodeS} \rightarrow \text{FeatureStructS} \\
\text{AXIOM IsSpecifier nd} & \quad \text{And Value[Category] Mother nd} = \text{C} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Features Head Mother nd}\right) \text{ Keep Set(WhWord)} \\
\text{AXIOM IsAdjunct nd} & \quad \text{And Value[Category] Mother nd} = \text{C} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Features Head Mother nd}\right) \text{ Keep Set(Inversion,Tense,Subject)} \\
\text{AXIOM IsSpecifier nd} & \quad \text{And Value[Category] Mother nd} = \text{Agrs} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Value[Subject] Head Mother nd}\right) \text{ Keep Set(Agreement)} \\
\text{AXIOM IsAdjunct nd} & \quad \text{And Value[Category] Mother nd} = \text{Agrs} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Value[Subject] Head Mother nd}\right) \text{ Keep Set(Agreement)} \\
\text{AXIOM IsSpecifier nd} & \quad \text{And Value[Category] Mother nd} = \text{T} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Value[Subject] Head Mother nd}\right) \text{ Keep Set(Case)} \\
\text{AXIOM IsAdjunct nd} & \quad \text{And Value[Category] Mother nd} = \text{T} \\
& \quad \Rightarrow \text{Functional nd} \\
& \quad = \left(\text{Features Head Mother nd}\right) \text{ Keep Set(Tense)}
\end{align*}
\]
AXIOM IsSpecifier nd
And Value[Category] Mother nd = Agro
==>
FFFunctional nd = (Value[Object] Head Mother nd)

AXIOM IsAdjunct nd
And Value[Category] Mother nd = Agro
==>
FFFunctional nd = (Features Head Mother nd)
Keep Set (Object)

AXIOM IsSpecifier nd
And Value[Category] Mother nd = D
==>
FFFunctional nd = EmptyStruct

AXIOM IsAdjunct nd
And Value[Category] Mother nd = D
==>
FFFunctional nd = (Features Head Mother nd)
Keep Set (Determiner, Case, Agreement, WhWord)

AXIOM (Value[Category] /= C
And Value[Category] /= Agrs
And Value[Category] /= T
And Value[Category] /= Agro
And Value[Category] /= D
)
Or (Not IsAdjunct nd
And Not IsSpecifier nd
)
==>
FFFunctional nd = EmptyStruct

Now we move on to the more elaborate description. I will start with the second axiom. Zwart [Zwa97, Page 204], following Den Besten [Bes78], assumes that the tense of T is checked against C. The reason for this type of feature checking is the fact that complementizers are connected with the tense of a clause. Some complementizers, such as *dat* (that), introduce a finite embedded clause, whereas others, such as *om* (to), introduce nonfinite embedded clauses (see Example 8.5). In the formalization I check the tense feature of C against the tense feature of V instead of the tense feature of T. Extensionally this is the same, as V and T must have identical tense values since V must check its tense feature against T. The features that have to be checked against the feature of C are yielded by the function \texttt{FFLexical}. If we apply this function to the upper T in Example 8.6, all the formal features of V are yielded. T namely is an adjunction, and, as we saw earlier section,
FFLexical applied to an adjunction yields the same feature structure as FFLexical applied to its adjunct.

Example 8.5

(a) Ik vertel haar dat jij het boek koopt
   (I tell her that you the book buy)
   I tell her that you buy the book
(b) Ik vraag haar om het boek te kopen
   (I ask her to the book to buy)
   I ask her to buy the book

Example 8.6

Agreement is also checked in C. The reason for this type of feature checking is the fact that many Germanic languages and dialects that show an asymmetry in verb movement between main and embedded clauses also show complementizer agreement. Example 8.7 gives two embedded clauses in Frisian that show complementizer agreement. The form in which the complementizer *dat* (that) appears depends on the subject by which it is followed. Therefore, the agreement features of the complementizer (C) are checked against the agreement features of the verb, which must equal the agreement features of the subject.

Example 8.7

(a) datsto (datAGR do) him sjocht
   (thatyou (thatAGR you) him see)
   that you see him
(b) dat by him sjocht
   (that he him sees)
   that he sees him
The following mainly concerns the fifth axiom of the function \textbf{FFFFunctional}. According to the Minimalist Program [Cho95, Page 174] V and T are responsible for case assignment. The case feature of the subject and the object are respectively checked against T (in the specifier position of AgrSP) and V (in the specifier position of AgrOP).\textsuperscript{6} Case feature checking takes place in AgrSP and AgrOP respectively because V moves to AgrO and T moves to AgrS.

In the formalization it is assumed that the verb moves from AgrO to T to AgrS and that case feature checking of the subject takes place in the specifier position of TP. This solution is preferred because feature checking of the subject DP against the case feature of T within AgrSP (see Example 8.8) would be an exceptional kind of feature checking. Usually, a specifier or an adjunct within a projection a checks its features against the head of a, which is AgrS in this case. Hence, in Example 8.8 V and DP check their features against AgrS, not against T or AgrO. Therefore, the checking of the case feature of the subject DP must take place within TP.

\textbf{Example 8.8}

\begin{itemize}
\item AgrSP
\item DP
\item AgrS
\item AgrS
\item TP
\item T
\item AgrS
\item AgrO
\item T
\item V
\item AgrO
\end{itemize}

Now we turn to the tenth axiom of the function \textbf{FFFFunctional}. I am aware of the fact that there have been various proposals within Chomskyan linguistics with respect to the internal structure of NPs and DPs. These proposals deal with agreement between determiners and nouns on the one hand, and adjectives and nouns on the other hand (cf. [Abn87], [Cor91], [Cor94]).

\textsuperscript{6}Note that therefore the object feature of the verb includes a case feature, while the subject feature of the verb does not include a case feature (see for instance Definition 8.1).
In the formalization only very simple DPs, which contain a noun and possibly a determiner (D), are dealt with. D, which possibly is an empty head (i.e., a head without a word value), directly takes an NP as its complement, without any intermediate functional projections, as we can see in Example 8.9. The only type of movement that takes place within the DP is head movement of N to D. The features that N checks in D are given in the ninth axiom. The determiner, case and agreement features are checked to obtain determiner noun agreement.

The determiner feature is a feature that does not occur in the Minimalist Program. It is applied here to be able to distinguish the treatment of pronouns such as zij (she) and interrogative words such as wat (what) on the one hand from nouns such as jongen (boy) on the other hand with respect to the selection of determiners. Pronouns and interrogative words can only check their features against those of an empty D (i.e., a D with no phonological content), while nouns, except for the plural forms, can only check their features against those of a D with lexical content such as de (the).

The fact that D has case features is not very surprising. For instance, in German there are different determiners for each case-gender combination. Gender and number features also may influence the choice of determiners. For example, in French there are different determiners for neuter and masculine on the one hand (le) and feminine on the other hand (la). All plural forms, neuter, feminine as well as masculine, have the determiner les in French.

The Wh-feature is a feature which the DP wants to check against the Wh-feature of CP. Therefore it is convenient that D has a Wh-feature so that we do not have to go into the NP to find the Wh-feature. The noun and the determiner must have the same value for the Wh-feature. Therefore, N adjoins to D to check its Wh-feature, and other features. For instance, in the DP who we have an empty D with a feature-value pair WhWord Yes and a noun who with the same feature-value pair.

Example 8.9

```
  DP
     ↓<empty>
      D
         ↓
          D NP
             ↓<empty>
              N
               ↓<empty>
           N
```
In the following, I will describe the function \textit{LinkSource nd} which I already mentioned above, together with its counterpart \textit{LinkTarget nd}. In derivational terms: \textit{LinkTarget nd} yields a node where the constituent in \textit{nd} moved to; \textit{LinkSource nd} yields a node where the constituent in \textit{nd} moved from. In representational terms: \textit{LinkTarget nd} = \textit{nd2} implies that the constituents in \textit{nd1} and \textit{nd2} are part of the same chain and that \textit{nd1} is a node that is located lower and to the right of \textit{nd2}; \textit{LinkSource nd} = \textit{nd2} implies that the constituents in \textit{nd1} and \textit{nd2} are part of the same chain and that \textit{nd1} is higher and to the left of \textit{nd2}. Of course, both functions are partial as not each node in a tree is the root of a constituent that is part of a chain. The formalization of both functions can be found in Definition 8.4.

The function \textit{LinkTarget} is based on the function \textit{ConnectionTarget} from the tree module. \textit{ConnectionTarget} just indicates a connection between one node and another. \textit{LinkTarget} takes into account that head movement involves a tree that grows with each movement (see Example 8.4 and 8.6). For instance, V adjoins to AgrO, the adjunction structure with AgrO as a root adjoins to T etc. Therefore the second axiom deals with adjuncts: if \textit{nd} is an adjunct, then we can say that it moved to the node its mother moved to (for instance, the V adjoined to AgrO moves to the position its mother (AgrO) moves to (i.e. the adjunct position of T)). \textit{LinkTarget nd} simply is the reverse case of \textit{LinkSource nd} (see third axiom).

\textbf{Definition 8.4}

\begin{verbatim}
F: LinkTarget : NodeS -> PARTIAL NodeS
F: LinkSource : NodeS -> PARTIAL NodeS
A: Not IsAdjunct nd
    ==> LinkTarget nd = ConnectionTarget nd
A: IsAdjunct nd
    ==> LinkTarget nd = ConnectionTarget Mother nd
A: LinkSource nd1 = nd2
    <= LinkTarget nd2 = nd1
\end{verbatim}

\footnote{Note that head movement is considered to be problematic for representational versions of generative syntactic theory (for instance, Rizzi [Riz90] and Brody [Bro95]), since chains rather than movements are central in representational approaches. Movements are subject to locality conditions, and those locality conditions are impossible to express for head chains, where the target of adjunction for a head can itself subsequently move. Considering traces to be copies is a possible solution here.}
CorrectChains tr is a boolean function that ensures that no impossible chain links occur within tr and that each link in the chain is associated with feature checking (see Definition 8.5). The possible links are specified by the functions PossibleLinkA (nd1, nd2) and PossibleLinkB (nd1, nd2). The solution given in those functions is rather trivial for reasons I specified in Section 8.1.

The functions PossibleLinkA, PossibleLinkB and FeaturesChecked will be described in more detail below. The function PossibleLink is split up into two parts (A and B) for the sake of perspicuity. PossibleLinkA specifies the general rules for movement within the Minimalist Program: always move from the complement domain of a functional head to its checking domain. PossibleLinkB specifies the category-specific rules which we need to avoid certain impossible movements which are not ruled out by the Minimalist Program.

**Definition 8.5**

\[
\text{FUNC } \text{CorrectChains} : \text{TreeS} \rightarrow \text{BoolS}
\]

\[
\text{AXIOM } \text{CorrectChains tr} \\
\iff \forall nd_1, nd_2 \\
( nd_1 \text{ NodeOf tr} \\
\text{And ConnectionTarget } nd_1 = nd_2 \\
\implies \text{PossibleLinkA (nd1, nd2)} \\
\text{And PossibleLinkB (nd1, nd2)} \\
\text{And FeaturesChecked tr})
\]

The predicate \( tr_1 \text{ InComplementDomainOf } tr_2 \) is applied within the function PossibleLinkA to define the notion ‘complement domain’. \( tr_1 \) is in the complement domain of \( tr_2 \) if the complement of \( tr_2 \) reflexively dominates \( tr_1 \) (see Definition 8.6).

**Definition 8.6**

\[
\text{FUNC } \text{InComplementDomainOf : TreeS, TreeS } \rightarrow \text{BoolS}
\]

\[
\text{AXIOM } tr_1 \text{ InComplementDomainOf } tr_2 \\
\iff (\text{Complement } tr_2) \text{ RefD} \text{omina} \text{tes } tr_1
\]

As was said above, the function PossibleLinkA (\( tr_1, tr_2 \)) specifies certain general rules for possible links. The first conjunct of the axiom in Definition 8.7 shows that links must always connect two identical trees. In the formalization, which is static because it only formalizes which trees are
built according to the Minimalist Program and not how these trees are built, movement is indicated by connections between identical trees. We need the function \texttt{Cut} because we cannot say that two different subtrees \texttt{tr}_1 and \texttt{tr}_2 of a tree are equivalent, this because they have different mothers, as we saw in Chapter 4. If the subtrees are cut out of the bigger tree we can say they are equivalent. The second conjunct says that if \texttt{tr}_1 is a head or an adjunction structure, it may not be too deep in the structure (i.e. it must be the sister of a complement). If \texttt{tr}_2 is an adjunct, it may not be to deep either (fifth conjunct). Its mother must be the sister of a complement. According to the third and the fourth conjunct, \texttt{tr}_1 must be in the complement domain of the functional projection it moves to. If \texttt{tr}_2 is a specifier (\texttt{Not IsAdjunct} \texttt{tr}_2) we look at the complement domain of the maximal projection (\texttt{XP: Mother} \texttt{tr}_2). If \texttt{tr}_2 is an adjunct we look at the intermediate projection (\texttt{X: Mother Mother} \texttt{tr}_2).

\textbf{Definition 8.7}

\begin{verbatim}
FABC PossibleLinkA : TreesS, TreesS -> BoolS
AXIOM PossibleLinkA (tr1, tr2)
  <=t Cut tr1 = Cut tr2
  And ( IsHead tr1
        Or IsAdjunction tr1
        => IsComplement Sister tr1
    )
  And ( Not IsAdjunct tr2
        => tr1 InComplementDomainOf (Mother tr2)
    )
  And ( IsAdjunct tr2
        => tr1 InComplementDomainOf (Mother Mother tr2)
    )
  And ( IsAdjunct tr2
        => IsComplement Sister Mother tr2
    )

The function \texttt{PossibleLinkB(nd1,nd2)} specifies the trivial, category-specific rules (see Definition 8.8). All possible movements are listed in the formalization: movement from the specifier of VP to the specifier of TP (for intransitive sentences), movement from the specifier of TP to the specifier of AgrSP, movement from the complement of VP to the specifier of AgrOP, movement from the head of VP to the adjunct of AgrOP, movement from the head of NP to the adjunct of DP, movement from the head of VP to the adjunct of TP, movement from the adjunction of AgrOP to the adjunct of TP, movement from the adjunction of AgrOP to the adjunct of TP (i.e. the highest adjunction as is specified in the function \texttt{PossibleLinkA}) to the adjunct of AgrSP, movement from the adjunction of AgrSP to the adjunct of C, and finally,
movement from the specifier of AgrO or AgrS to the specifier of C. Definition 8.8 contains just a selection of the possible movements I summed up above.

**Definition 8.8**

\[
\text{FUNC} \quad \text{PossibleLinkB} : \text{TreeS}, \text{TreeS} \rightarrow \text{BoolS}
\]

\[
\text{AXIOM} \quad \text{PossibleLinkB}(\text{tr1}, \text{tr2}) \leftrightarrow
\begin{array}{l}
\quad (\text{IsSpecifier tr1} \\
\quad \quad \text{And IsSpecifier tr2} \\
\quad \quad \text{And Value[Category] Mother tr1 = V} \\
\quad \quad \text{And Value[Category] Mother tr2 = T})
\end{array}
\]

\[
\cup
\begin{array}{l}
\quad (\text{IsSpecifier tr1} \\
\quad \quad \text{And IsSpecifier tr2} \\
\quad \quad \text{And Value[Category] Mother tr1 = T} \\
\quad \quad \text{And Value[Category] Mother tr2 = Agrs})
\end{array}
\]

\[
\cup
\begin{array}{l}
\quad (\text{IsComplement tr1} \\
\quad \quad \text{And IsSpecifier tr2} \\
\quad \quad \text{And Value[Category] Mother tr1 = V} \\
\quad \quad \text{And Value[Category] Mother tr2 = Agro})
\end{array}
\]

\[
\cup
\begin{array}{l}
\quad (\text{IsHead tr1} \\
\quad \quad \text{And IsAdjunct tr2} \\
\quad \quad \text{And Value[Category] Mother tr1 = V} \\
\quad \quad \text{And Value[Category] Mother tr2 = Agro})
\end{array}
\]

\[
\cup
\begin{array}{l}
\quad (\text{IsAdjunction tr1} \\
\quad \quad \text{And IsAdjunct tr2} \\
\quad \quad \text{And Value[Category] Mother tr1 = Agro} \\
\quad \quad \text{And Value[Category] Mother tr2 = T})
\end{array}
\]

The function \text{FeaturesChecked tr} describes the actual feature checking within a tree \text{tr} (see Definition 8.9). Each node \text{nd} of \text{tr} that has been the target of movement (\text{LinkSource nd} \neq \text{Undef}) may not yield an empty feature structure for the function \text{FFFunctional nd}. Furthermore it must be possible to check \text{FFFunctional nd} against \text{FFLexical nd}, that is, the feature structure that is yielded by \text{FFLexical nd} contains at least all the features yielded by \text{FFFunctional nd} and possibly more.
8.3. Summary

The main objective of this chapter was to define which chains are correct according to the Minimalist Program. An important constraint on chains is that each link in a chain must be connected with feature checking. But the feature checking requirement is not the only constraint on links. There are also constraints on the link source and the link target (i.e. ‘landing site’ in derivational terms). The link target must be in the checking domain of a functional head. The link source must be in the complement domain of that same functional head. I was also forced to formulate some trivial constraints on links to prevent subjects from checking their features in AgrOP and objects from checking theirs in AgrSP.