University of California
Los Angeles

Parsing Minimalist Languages

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Linguistics

by

Hendrik Harkema

2001

The dissertation of Hendrik Harkema is approved.

T.K. Au

E.L. Keenan

A. Nijholt

C.T. Schütze

E.P. Stabler, Committee Chair

University of California, Los Angeles

2001

TABLE OF CONTENTS

Acknowledgments

I would like to express my sincere thanks to Dr. Ed Stabler for supervising this dissertation. His guidance and enthusiasm have been invaluable to me. I also would like to thank the other members of my committee, Dr. Terry Au, Dr. Ed Keenan, Dr. Anton Nijholt, and Dr. Carson Schütze, for their interest in my work.

I would like to thank the faculty and my fellow graduate students in UCLA's Linguistics Department for their contributions to what have been five enjoyable and rewarding years in Los Angeles, both academically and otherwise.

Finally, I wish to thank my family. I am very grateful for their continued support and encouragement to pursue my dreams.

Vita

1993	M.S., Computer Science University of Twente, Enschede, the Netherlands
1998	M.A., Linguistics, University of California, Los Angeles, California

Abstract of the Dissertation

Parsing Minimalist Languages

by

Hendrik Harkema

Doctor of Philosophy in Linguistics

University of California, Los Angeles, 2001

Professor E.P. Stabler, Chair

It seems to be a general feature of natural language that the elements of a sentence can be pronounced in one position, while at the same time serving a function in another part of the structure of a sentence. Linguistic theories in the Transformational Generative Grammar tradition have tried to provide a unified explanation for this observation by proposing analyses that involve movement of constituents.

The subject of this dissertation is parsing sentences using a grammar formalism that allows for movement of constituents. The recognizers presented in this dissertation will recognize languages generated by Minimalist Grammars as defined in [Sta97]. Minimalist Grammars are simple formal grammars based on the Minimalist approach to linguistic theory ([Cho95]). Minimalist Grammars are derivational and feature-driven: phrases are derived by applying transformational operations to lexical items and intermediate structures, and the applicability of the transformational operations is determined by the syntactic features of the structures involved.

We will show that that the set of languages generated by Multiple Context-Free Grammars ([SHF91]) is included in the set of languages generated by Minimalist Grammars. Together with the reverse result in [Mic98], this completes the proof that Multiple Context-Free Grammars and Minimalist Grammars are weakly equivalent.

Minimalist Grammars can move material from positions arbitrarily deep inside a sentence. Michaelis' equivalence result ([Mic98]) permits a representation of Minimalist Grammars in which the operations of the grammar can be characterized in such a way that they are strictly local. We will use this representation to derive several polynomial time recognition algorithms for Minimalist Languages, including an Earley-style recognizer ([Ear68]). Unlike known recognizers for Multiple Context-Free Languages, the recognizers in this dissertation can handle grammars with empty strings which are not immediately derived from the start symbol of the grammar. Moreover, the recognizers will parse a sentence from left to right.

The research reported in this dissertation contributes to a better understanding of Minimalist Grammars and related formalisms. Also, because Minimalist Grammars lends themselves very well to expressing current proposals about the kind of structures found in natural language, the recognizers described in this dissertation can be used to rigorously explore the computational consequences of psycholinguistic theories of human sentence processing.

CHAPTER 1

Introduction

It seems to be a general feature of natural language that the elements of a sentence can be pronounced in one position, while at the same time serving a function in another part of the structure of a sentence. For example, in sentence 1 the word *who* appears at the beginning of the sentence, but it is also the object of the verb *love*. Sentence 2 shows that, in simple declarative English sentences, the object of a verb occurs to the right of the verb.

1. Who does Mary love?

2. Mary loves John.

Linguistic theories in the Transformational Generative Grammar tradition have tried to provide a unified explanation for sentences such as 1 and 2 by proposing analyses that involve movement of constituents. At an abstract level of representation, all objects are assumed to be to the right of their verbs. Then, if necessary, an object will move to its surface position in the sentence. According to this analysis, the sentence in 1 is derived from an abstract structure in which *who* occupies the same position as *John* in sentence 2.

The subject of this dissertation is parsing sentences using a grammar formalism that allows for movement of constituents. The particular grammars that we will use in this dissertation are inspired by Chomsky's Minimalist Program [Cho95]. Parsing is

the process of assigning a structure to a sentence, given a grammar of some language that this sentence does or does not belong to. Of course, if the sentence does not belong to the language, no (complete) grammatical structure can be assigned to it – the sentence is ungrammatical.

The present chapter serves as an introduction to the work reported in this dissertation. I will first provide the motivation for this work and describe the contributions of this dissertation. Next, I will outline some earlier approaches to parsing with transformational grammars and introduce some grammar formalisms for which parsing algorithms have been specified and that fall in the same general class of grammars as the Minimalist Grammars that will be used in this dissertation. Finally, I will briefly sketch the main challenge of parsing with transformational grammars and the way in which this challenge will be met in this dissertation.

1.1 Motivation and Contribution

Psycholinguistic research has provided many interesting insights into the way humans parse natural languages. Most psycholinguistic work is empirical in nature. The psycholinguist selects a set of sentences with grammatical constructions that are predicted to engage the human sentence processor in a way that will reveal its inner workings, designs an experiment with these sentences and runs it with a number of human subjects, interprets the results and proposes certain properties that the human sentence processor should have in order to account for the experimental results. The results of this approach greatly expand our knowledge about the human sentence processor, but it sometimes fails to satisfy linguists of a more formal bent, because a mere description of its properties does not constitute a complete specification of a parser.

In most psycholinguistic proposals, the operations of the parser are only specified

insofar as they bear on the natural language constructions being explored. Moreover, this specification is usually very informal. While the parser works for the set of examples provided, absence of a description of the overall architecture of the parser leaves open the question whether a parser with the desired properties covering the entire language actually exists. Another drawback of some of the work in psycholinguistics is that it is sometimes based on simplistic and outdated conceptions of syntactic structure.

For example, Weinberg [Wei99] argues that the operation of the human sentence processor is based on principles that follow directly from the grammar, rather than from extra-linguistic considerations such as storage capacity or from frequency of occurrence of linguistic structures. She then shows how to interpret the principles embodied in Minimalist Grammars as a parsing algorithm ([Wei99], p. 290):

> *A derivation proceeds left to right. At each point in the derivation, merge using the fewest operations needed to check a feature on the category about to be attached. If merge is not possible, try to insert a trace bound to some element within the current command path. If neither merger nor movement is licensed, spell out the command path. Repeat until all terminals are incorporated into the derivation.*

Since Merge, Move ("insert a trace bound to some element within the current command path") and Spell-Out are just the three grammatical operations available in Weinberg's conception of Minimalist Grammars, this is not a very informative set of instructions.

In this dissertation we will pursue a formal approach to parsing natural languages. We will start from the Minimalist Grammars introduced in [Sta97], which are a rigorous formalization of the kind of grammars proposed in the framework of the Minimalist Program [Cho95], and then develop several parsing algorithms for these grammars.

The advantage of this formal approach is that the parsers actually work – they are provably sound and complete: they will parse all the sentences of a language and no others.[1] Also, the formal specification of the parsing algorithms allows for a rigorous study of their computational properties as they relate to the properties of the human sentence processor.

In chapter 3, I will show that the set of languages generated by Multiple Context-Free Grammars [SHF91] is included in the set of languages generated by Minimalist Grammars. Together with Michaelis' reverse result [Mic98], this completes the proof that Multiple Context-Free Grammars and Minimalist Grammars are weakly equivalent. Then it follows immediately from earlier results that Minimalist Grammars are also weakly equivalent to Linear Context-Free Rewriting Systems [VWJ87], Multi-Component Tree-Adjoining Grammars [Wei88], and Simple Positive Range Concatenation Grammars [Bou98]. This is a significant formal result, as it follows that the properties of the languages generated by the latter class of grammars are now known to be properties of Minimalist Languages as well, and vice versa. The result presented in chapter 3 also implies that removing head movement and covert phrasal movement from Minimalist Grammars does not affect the formalism's weak generative capacity.

Minimalist Grammars can move material from positions arbitrarily deep inside a sentence. Michaelis' equivalence result ([Mic98]) permits a representation of Minimalist Grammars in which the operations of the grammar can be characterized in such a way that they are strictly local. In chapters 4, 5, and 6, I will use this representation to derive several polynomial time recognition algorithms for Minimalist Languages. Chapter 4 contains a description of a bottom-up recognizer akin to the CKY algorithm for Context-Free Languages ([You67]). In chapter 5, a top-down recognizer is developed, with a mechanism for look-ahead. This recognizer has the correct-prefix

[1] Formalization entails the abstraction that a language is a set of sentences. A sentence is either in the language or it is not; there are no degrees of grammaticality.

property, meaning that it will stop working on an ungrammatical sentence as soon as it encounters the left-most word responsible for the ungrammaticality. The recognizers in chapters 4 and 5 lead up to the first Earley-style recognizer ever to be formulated for Minimalist Grammars, which will be revealed in chapter 6. Like its canonical Context-Free counterpart ([Ear68]), the recognizer follows a top-down strategy, but it is applicable to a wider range of Minimalist Grammars than the top-down recognizer in chapter 5.

Besides providing a solid basis for modeling psycholinguistic proposals, the parsers presented in this dissertation will also be relevant for formal approaches to learning theory, where a learner has to parse new data in order to check whether his current grammar is complete [Sta98], and for computational models of Optimality Theory, which use Multiple Context-Free Grammars to describe reduplication [Alb00].

1.2 Transformational Parsing

Minimalist Grammars are not the first kind of grammar to emerge from the Transformational Generative approach to natural language. The succession of linguistic theories developed within this tradition has been accompanied by a succession of proposals about how to parse languages according to these theories. This section provides a brief survey of these proposals. The section's brevity entails some simplifications in the presentation; for more details the reader is referred to the original works cited in the text.

1.2.1 Early Work

According to the Standard Theory proposed in [Cho65], the syntactic part of a transformational grammar consists of a base component and a transformational component.

The base component contains a set of rewrite rules, usually context-free, which generate a set of base trees. The transformational rules included in the transformational component are mappings from trees to trees. Applying the transformational rules in a prescribed order to a base tree coming out of the base component will produce a surface tree. The yield of this surface tree is a sentence in the language. A transformational grammar thus defines a relation between sentences and base trees.

Parsing a sentence using the kind of transformational grammar described above generally involves the following four steps, cf. [Kin83], [Gri86], [BF95]. First, the parser will generate a set of possible surface trees for the sentence, using a so-called covering grammar. Next, the parser will reconstruct a set of possible base trees from the set of possible surface trees by applying the transformational rules of the grammar in reverse. After this step, the parser will discard those base trees that cannot actually be generated by the base component of the grammar. Finally, the parser will feed the remaining base trees into the transformational component to establish which base trees can in fact produce the sentence to be parsed. The set of base trees surviving this last test will be returned by the parser as the result of the parsing process.

Unfortunately, parsers based on this architecture ran into serious computational problems, because the early versions of transformational grammar were not sufficiently restricted. In fact, Peters and Ritchie [PR73] showed that transformational grammars are capable of generating recursively enumerable sets. King [Kin83] provides an overview of the ways in which these computational problems manifested themselves in early transformational parsing systems. One pervasive problem was caused by transformational rules that delete material from trees. Applying these rules in reverse can be very problematic, because the tree to which the reverse transformational rule is to apply may not always contain enough information to reconstruct the content and position of the deleted material. Because of these computational problems,

early transformational parsers were not considered very successful.

1.2.2 Marcus Parser

In the late 70s and early 80s, Marcus promoted the claim that natural languages, or at least English, can be parsed in a deterministic fashion. The parser introduced in [Mar80] illustrates this point. The parser is deterministic in that the structures it builds will never be undone and will all be part of the structure that is eventually assigned to the input sentence. Also, at each point in the parsing process, the current state of the parser uniquely determines the next action of the parser. The state of the parser is represented by the contents of the active node stack and a buffer, which are the two data structures maintained by the parser. The active node stack holds incomplete structures that the parser is presently working on, and the buffer contains a limited number of look-ahead elements that will be used to resolve local ambiguities. Crucially, the buffer can contain previously recognized constituents, rather than just the next few words of the sentence.

According to Marcus, the determinism hypothesis offers an explanation for certain constraints that are generally obeyed by natural language grammars. An example of such a constraint is the complex NP constraint, which, simply put, prevents movement out of a clause within an NP ([Ros67]). This constraint accounts for the ungrammaticality of the sentence in 4, for example, which is related to the sentence in 3.

3. Mary believes [$_{NP}$ the rumor [$_{CP}$ that John kissed Sue]].

4. $*$Who$_i$ does Mary believe [$_{NP}$ the rumor [$_{CP}$ that John kissed t_i]]?

Marcus shows that sentence 4 and similar sentences cannot be parsed deterministically by his parser.

The determinism hypothesis also has psycholinguistic relevance: it explains the occurrence of garden-path sentences. Garden path sentences are sentences which pose conscious difficulties for the human sentence processor. The classic example of a garden path sentence is the sentence in 5, from [Bev70], where the human sentence processor wrongly assumes that *raced* is the main verb of a verb phrase, rather than a participle in a reduced relative clause.

5. The horse raced past the barn fell.

Marcus' parser will break down on some garden path sentences, because the limited look-ahead causes the parser to make a decision about the structure of the sentence which later will turn out to be wrong, but its deterministic nature does not allow the parser to revoke decisions that were made earlier.

One disadvantage of the parser proposed by Marcus is that it suffers from a lack of formal rigor, which makes it hard to verify the claims made about the parser. For example, the rules of the grammar are encoded in the operations of the parser. Consequently, it is not obvious to what extent the behavior of the parser is influenced by the particular grammatical analyses adopted for certain constructions in the language and what aspects of its behavior follow from the architecture of the parser – see for example the discussion in [Sam83]. Since the grammar is not separated from the parser, it is also not clear what class of languages can be dealt with by the parser. The latter issue was resolved by Nozohoor-Farshi [Noz86], but the grammars he uses in his formalization – context-free grammars augmented with attributes – are rather far removed from the grammatical descriptions that linguists use.

1.2.3 Principle-Based Parsing

In the 80s the Standard Theory and its revised and extended versions evolved into the theory of Government and Binding ([Cho81], [Cho86]). The construction-specific and often language-specific rules that were part of earlier versions of Transformational Generative Grammar were replaced by a single transformational rule – Move α: move anything anywhere – and a set of abstract principles. The principles are universal statements about the well-formedness of particular aspects of the structures generated by the grammar. They play an important role in reigning in the power of Move α.

Government and Binding theory gave rise to the construction of so-called principle-based parsers (e.g. [BAT91], [Ber91]). The theory of Government and Binding is much more restrictive than earlier transformational grammars, which helped principle-based parsers to avoid some of the computational problems facing the first transformational parsers. The organization of the grammar into one transformational rule and a set of principles, however, did not make the task of the parser any easier: rather than undoing specific transformations, the parser now has to concern itself with unraveling complex interactions between heterogeneous principles of the grammar. This generally makes principle-based parsers slow and inefficient. Efficiency can be improved by precompilation of the grammar, or "multiplying out" the interactions of the principles, but this will make the parser less faithful to the original grammar. A principled approach to precompilation is explored in [Mer95].

1.3 Beyond Context-Free Languages

It is generally assumed that Context-Free Grammars are inadequate for describing the syntax of natural languages. Shieber has shown that Swiss German has a construction involving cross-serial dependencies which go beyond the weak expressive power of

context-free grammars [Shi85]. No context-free grammar can generate all and only the right sequences of words for this construction. Dutch has a similar construction. It falls within the weak generative capacity of context-free grammars, but it is out of the reach of context-free grammars in the strong sense: while a context-free grammar can generate the right sequences of words, it cannot assign the linguistically appropriate structural description to this construction [BKP82].

Even if it appears that most natural languages do not have non-context-free constructions, linguistic practice shows that non-context-free descriptions are able to capture linguistic generalizations more succinctly than context-free ones. This is not just a matter of elegance; the cumbersomeness of context-free descriptions of natural language reduces their psycholinguistic appeal [Sta92].

Interest in the construction of compilers for context-free programming languages has generated an extensive body of work concerning parsing methods for context-free grammars, summarized in for example [AU72] and [ASU86]. In comparison, the field of non-context-free parsing is much less developed. Most of the investigations into non-context-free parsing have concentrated on languages generated by mildly context-sensitive grammar formalisms. These formalisms were first described in [Jos85] and studied later in more detail in [Wei88] and [JVW91].

Mildly context-sensitive grammars are characterized by the following three properties: 1) they are slightly more expressive than context-free grammars, weakly as well as strongly, 2) the languages they generate are parsable in polynomial time, and 3) they exhibit the constant growth property. The latter property rules out grammars that generate languages with linguistically unattested, non-linear patterns such as a^{2^n}, $n \geq 1$. These three properties of mildly context-sensitive grammar formalisms make them suitable candidates for the description of natural language. Following below is an overview of two mildly context-free grammar formalisms for which parsing meth-

ods have been developed: Linear Indexed Grammars and Tree Adjoining Grammars. The formalisms are introduced rather informally; formal definitions can be found in the referenced materials. The Minimalist Grammars to be introduced in chapter 2 of this dissertation have more expressive power than Linear Indexed Grammars and Tree Adjoining Grammars: the former kind of grammar generates languages that cannot be generated by either of the latter kind of grammar, cf. [Sta97] and chapter 3 of this dissertation.

1.3.1 Linear Indexed Grammars

Linear Indexed Grammars are a specialization of Indexed Grammars [Aho68]. Indexed Grammars are like Context-Free Grammars, except that the non-terminal symbols of the grammar can be associated with stacks of indices. The unbounded nature of these stacks gives Indexed Grammars more power than Context-Free Grammars.

Linear Indexed Grammars are Indexed Grammars whose rules obey a restriction introduced in [Gaz88]: the stack associated with the non-terminal symbol of the left-hand side of any rule is passed on to just one non-terminal symbol in the right-hand side of the rule, rather than copied onto all non-terminal symbols of the right-hand side, as in a Indexed Grammar.

The rules of a Linear Indexed Grammar can be grouped into "copy", "pop and copy", and "push and copy" rules. In the first kind of rule, the stack of indices associated with the non-terminal symbol of the left-hand side of the rule is copied intact onto the designated non-terminal symbol of the right-hand side of the rule, as in rule 6 below, for example.[2] The second kind of rule copies the stack of the left side of the

[2] Some notational conventions: non-terminal symbols are written in uppercase and terminal symbols are written in lowercase. Stacks are represented as sequences of lower case letters enclosed in square brackets, the top of the stack on the left. [ ] denotes the empty stack, and a stack with arbitrary content is written as [. . .].

rule onto the designated non-terminal symbol on the right, but only after its top-most symbol has been popped off; see rule 7, for example. The last kind of rule pushes a new index onto the stack provided by the left-hand symbol, and copies the result onto the designated non-terminal symbol on the right; rule 8 is an example.

6. A[i, j, k] $\rightarrow$ a B[i, j, k] c D[ ] e.

7. A[i, j, k] $\rightarrow$ B[ ] C[j, k].

8. A[j, k] $\rightarrow$ a B[i, j, k] c D[ ].

If the right-hand side of a rule does not contain any non-terminal symbols, the stack of the left-hand side of the rule is lost, e.g., A[i, j, k] $\rightarrow$ a b c. All derivations start from the distinguished start symbol S associated with an empty stack.

The grammar with the rules given in 9 through 12 is a Linear Indexed Grammar generating the non-context-free language L = $\{a^n b^n c^n \mid n \geq 0\}$.

9. S[. . .] $\rightarrow$ a S[i, . . .] c.

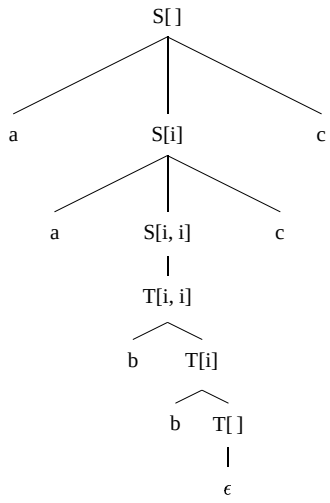
10. S[. . .] $\rightarrow$ T[. . .].

11. T[i, . . .] $\rightarrow$ b T[. . .].

12. T[ ] $\rightarrow \epsilon$.

The tree of the sentence *aabbcc* as it is generated by the grammar given above is given in 13.

13.



Parsers for Linear Indexed Grammars can be found in for example [VW91], [VW93], [Bou96], and [AGV00]. The formulation of the various parsers depends on the context-freeness property of Linear Indexed Grammars. This property allows for an efficient representation of the unbounded stacks of indices for the purposes of parsing [VW93].

1.3.2 Tree Adjoining Grammars

Tree Adjoining Grammars were introduced in [JLT75], and are elaborated and discussed in [Jos85], [JVW91], [JS97], and elsewhere. Tree Adjoining Grammars handle trees rather than strings, and they are more powerful than Context-Free Grammars. The formal properties of Tree Adjoining Grammars gave rise to the characterization of the class of mildly context-sensitive grammars in [Jos85].

A Tree Adjoining Grammar is specified by a set of initial trees and a set of auxiliary trees. An initial tree is a tree whose root node is labeled with the distinguished start symbol S and all its leaves are labeled with terminal symbols. An auxiliary tree is a tree all of whose leaves are labeled with terminal symbols, except for one – this leaf, called the foot node, is labeled with a non-terminal symbol which coincides with the label of the root of the tree. New trees are derived from these two sets of trees by

recursively adjoining auxiliary trees to initial trees. The operation of adjunction is defined as follows. Let γ be a tree with a node n labeled X, let β be an auxiliary tree whose root and foot node are labeled X, and let τ represent the subtree of γ dominated by n. Then the adjunction of β to γ at n is a tree γ' which is obtained in the following way:

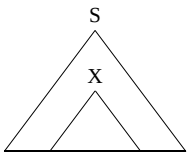
14. Remove subtree τ from γ, but leave a copy of n.

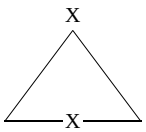
15. Insert auxiliary tree β at node n in what is left of γ.

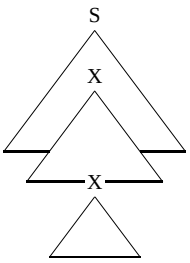
16. Attach subtree τ to the foot node of β.

The trees in 17 illustrate these three steps.

The application of adjunction can be constrained by specifying for each node of a tree what auxiliary trees can be adjoined to that node, if any. A node can also be marked for obligatory adjunction, which requires that an auxiliary tree from some specified set must adjoin to that node [Jos87].

17. γ: 

β: 

γ': 

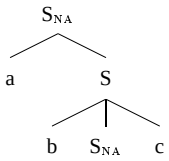
The language generated by a Tree Adjoining Grammar is the set of yields of the trees that can be derived from the set of initial trees via adjunction.[3] Thus, the Tree Adjoining Grammar with the single initial tree given in 18 and the single auxiliary

[3] Current versions of Tree Adjoining Grammars also have a substitution operation, which replaces a leaf of a tree by another tree, but adding this operation does not change the formal properties of Tree Adjoining Grammars [JS97].

tree given in 19 generates the language L = $\{a^n b^n c^n \mid n \geq 0\}$. In the trees, a node annotated with NA is subject to the adjunction constraint preventing any auxiliary tree from adjoining to that node. (Language L cannot be generated by a Tree Adjoining Grammar without adjunction constraints.)

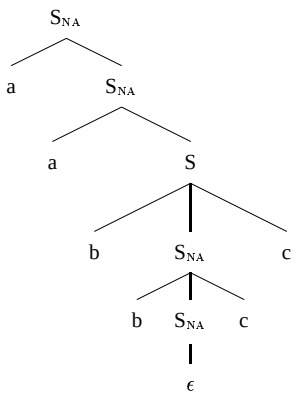
18.

19.



The tree in 20 for the sentence *aabbcc* is produced by adjoining the auxiliary tree in 19 to the only node that is available for adjunction in the initial tree in 18, and adjoining another copy of the auxiliary tree in 19 at the unconstrained S node in the tree resulting from the first adjunction.

20.



Tree Adjoining Grammars are weakly equivalent to Linear Indexed Grammars [Vij87]. Consequently, some parsers for Tree Adjoining Grammars first transform the Tree Adjoining Grammar into an equivalent Linear Indexed Grammar and then use

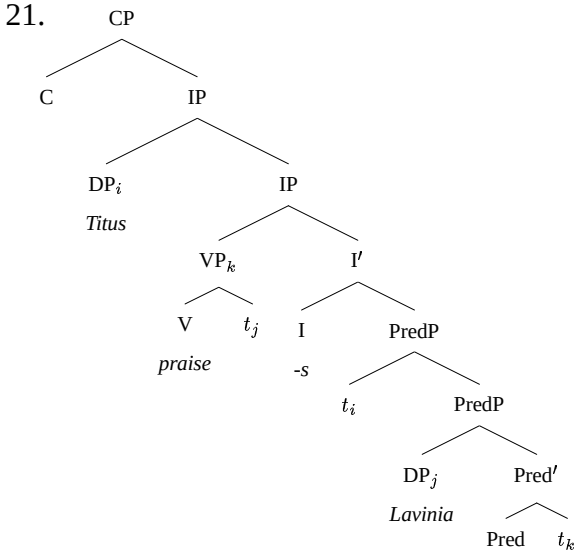
the latter grammar to parse the input sentence. This approach underlies the Tree Adjoining parsers presented in [VW91], [VW93], and [SS94], for example. The parsers formulated in [JS97] and [Ned99] use Tree Adjoining Grammars directly.

1.4 Parsing Minimalist Languages

The main challenge of parsing Minimalist Languages is to keep track of the antecedent-trace relations induced by constituents that move. A very straightforward book-keeping mechanism is the use of special registers for storing moved elements, as implemented in Augmented Transition Networks [Woo70]. For example, when the parser encounters a wh-phrase in a sentence, it will load it into a register, and unload it when it reaches a position in the sentence where the wh-phrase could have moved from, thus establishing the relation between a trace and its antecedent.

Obviously, the approach with registers is only adequate for antecedents that precede their traces. In Minimalist Grammars, antecedents may follow their traces in a sentence, because these grammars allow remnant movement. Remnant movement is movement of a constituent that remains after some other constituent has moved out of it. In [Mah00], the basic subject-verb-object order of simple English sentences is derived by moving the verb phrase to a specifier of the inflectional head I. For the object to end up to the right of I in a sentence, it must leave the verb phrase before this phrase will move. Thus, when the verb phrase moves, it is no longer a complete constituent. This approach is illustrated in the tree in 21, for the sentence *Titus praises Lavinia*.

21.



As a result of remnant movement, the trace t_j precedes its antecedent DP$_j$. When the parser reaches the word *Lavinia*, it can posit a displaced determiner phrase and store it in a register, but the register will never be emptied, because the determiner phrase's trace has already passed.

Monostratal grammar formalisms such as Generalized Phrase Structure Grammar [GPS85] use a similar mechanism to deal with displaced elements. Displacements are encoded in so-called slash categories. A slash category denotes a category from which another category is missing. For example, using WH as the category of wh-phrases, IP/WH stands for a constituent of category IP from which a wh-phrase has moved. The slash categories in the phrase structure rules 22, 23, 24, and 25 describe the displacement of a wh-phrase from the object position of a transitive verb to the beginning of a sentence.[4] Rules 22, 23, and 24 are derived from the independently existing rules S $\rightarrow$ IP, IP $\rightarrow$ DP VP, and VP$_t$ $\rightarrow$ V$_t$ DP; rule 25 is a new rule. Rule 22 states that a sentence consists of a wh-phrase followed by an IP from which a wh-phrase is missing. Rules 23 and 24 pass the missing wh-phrase on to the object of the

[4]These simple rules ignore 'do'-support and inversion in wh-questions and the subtleties of the analysis represented in tree 21.

transitive verb phrase. The slash is terminated in rule 25: a DP lacking a DP amounts to the empty string.

22. S $\rightarrow$ WH IP/WH.

23. IP/WH $\rightarrow$ DP VP/WH.

24. $\text{VP}_t \rightarrow \text{V}_t$ DP/DP.

25. $\text{DP/DP}_t \rightarrow \epsilon$.

Using the rules in 22 through 25, a parser will be able to recognize a sentence with a displaced wh-phrase in a bottom-up or a top-down manner, or in any other well-defined way.

GPSG does not support remnant movement. The grammar formalism does not provide the means to refer to complex categories such as PredP/(VP/DP) – this would be a category missing a category which itself lacks another category – nor the means to derive the rules needed to rewrite these categories. For the particular tree in 21 it may be possible to extend the formalism to generate the required categories and rules, but this will not be possible in general because of the limited power of the slash category mechanism. Implementing slash categories with stacks will give us the power of Linear Indexed Grammars, but Minimalist Grammars are more expressive than these grammars ([Sta97]).

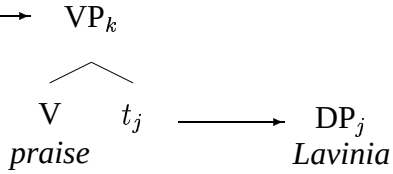
Rather than using special registers or trying to extend the employment of slash categories, Michaelis' equivalence result ([Mic98]) suggests we adopt a chain-based perspective on Minimalist Languages as explored in [Cor99] and [Sta01a] to tackle the parsing problem for Minimalist Grammars. A constituent that moves plays a role at more than one place in the structure of a sentence. A chain represents this distribution of grammatical import. Consider, for example, the chain (DP_j, t_j) in tree 21 which is

associated with the lexical item *Lavinia*. The element t_j reflects the fact that *Lavinia* is a DP because it can occur as the complement of the verb *praise*, which selects for determiner phrases. The element DP_j of the chain (DP_j, t_j) occurs in a different position because *Lavinia* is an item which needs case. Assuming that case is assigned in the specifier position of a Predicate Phrase, this element must occur in that particular position. Note that the chain (DP_j, t_j) is neutral as to the order of the trace and the antecedent. The lexical item *Lavinia* has two grammatical properties, both of which have to be reckoned with in a sentence.

Chains of the kind described above can be thought of as the primitive objects of Minimalist Grammars. According to this perspective, deriving a sentence amounts to assembling chains into a tree by connecting their elements. For example, the second element of the chain (DP_j, t_j) will be attached as the sister of the verb *praise* to form a VP, which itself is part of the chain (VP_k, t_k). The resulting structure, given in 26, is a non-concatenated or discontinuous constituent.[5]

The dangling DP_j *Lavinia* will eventually be attached in the specifier position of the predicate phrase, which is to the right of the inflectional head I. The verb phrase VP_k dominating verb V and trace t_j will end up in a specifier position to the left of I. Hence, in the sentence the inflectional element -s will intervene between *praise* and *Lavinia*.

26.



The operations that combine chains turn out to be context-free, since the applicability of an operation only depends on the particular elements of the chains which are

[5] In fact, the strings *praise* and *Lavinia* are the 'multiple' yields of the verb phrase in Michaelis' equivalent Multiple Context-Free Grammar.

involved in the operation and no other elements, and every grammar makes available only a finite number of different types of chains. This fact allows us to adapt well-know techniques for context-free parsing and apply them to parsing Minimalist Languages. These adaptations, however, are not trivial, because unlike Context-Free Grammars, Minimalist Grammars can generate discontinuous constituents.

1.5 Overview of the Dissertation

In this chapter we have motivated and sketched the context of the work reported in this dissertation. In the next chapter we will introduce the formal definitions for Minimalist Grammars. Chapter 3 will fix the position of Minimalist Grammars in the hierarchy of formal languages. The next three chapters are devoted to the descriptions of bottom-up, top-down, and Earley-style parsers for Minimalist Grammars. The dissertation will close with some conclusions and directions for future investigations.

CHAPTER 2

Minimalist Grammars

The recognizers presented in this dissertation will recognize languages generated by Minimalist Grammars as defined in [Sta97]. In this chapter, in section 2.2, we will introduce these grammars. Minimalist Grammars are a rigorous formalization of the kind of grammars proposed in the framework of Chomsky's Minimalist Program [Cho95], which will be outlined very briefly in section 2.1. Section 2.5 contains an overview of some other recent formal approaches to the Minimalist Program and Transformational Grammar in general.

2.1 Minimalist Program

In the theory of Government and Binding, the transformational rule Move α applies freely to d-structures assembled according to the X′ rewrite rules. Many of the resulting representations are not well-formed; they are ruled out by other principles of the grammar. In the transition from the theory of Government and Binding to Minimalism [Cho95], attention shifted away from the formulation of conditions acting as filters on derivations and representations to an explicit specification of the generative procedure itself, which embodies some of the conditions on well-formedness. In minimalist grammars, Move α and the X′ rewrite rules have been replaced by the structure building operations Merge and Move. Merge is an operation combining two trees into one, and Move is for displacing elements within a tree.

The grammatical module is assumed to interface with two other cognitive modules: the Articulatory-Perceptual module and the Conceptual-Intentional module. The principle of Full Interpretation requires that the representations delivered to these interfaces by the grammatical module, Phonetic Form and Logical Form respectively, must only contain symbols interpretable by these external modules. Hence, uninterpretable grammatical features in a representation must be deleted during a derivation. A feature is deleted when it enters into a checking configuration with a head which has a matching feature. These checking configurations are created by the operation Move.

To account for differing word orders between languages, a distinction is made between weak features and strong features. Deleting strong features will cause an entire head or phrase to be moved into a checking configuration, including its phonetic features. This movement is overt, because it is reflected in the Phonetic Form. In the case of weak features, only the syntactic features will be moved; the phonetic features are left behind. This is referred to as covert movement. Features that are strong in one language, may be weak in another, and vice versa.

For a more comprehensive overview of the Minimalist Program and recent developments, the reader may consult [Cho95], [Cho00], [Cho01], [AET96], and [EH99], for example.

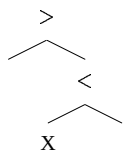
2.2 Derivational Minimalism

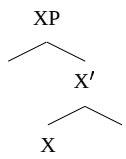
The Minimalist Grammars presented in this section, originally defined in [Sta97] are simple formal grammars based on the Minimalist approach to linguistic theory. Minimalist Grammars are derivational and feature-driven: phrases are derived by applying transformational operations to lexical items and intermediate structures, and the applicability of the transformational operations is determined by the syntactic features

of the structures involved. The description of Minimalist Grammars in this chapter is based on [Sta97]. Formal definitions can be found there, and also in section 3.1 in the next chapter of this dissertation. Other explorations of Minimalist Grammars include [Cor99], [LR99], [Sta99], [SK00], [Har01], and [Mic01a].

2.2.1 Minimalist Grammars

Minimalist Grammars generate trees. In accordance with the minimalist imperative, these trees are bare structures, as the tree in 1, for example. Rather than labeling intermediate projections with a copy of the label of the head and a bar level, cf. the traditional notation of trees in 2, intermediate projections are identified by a pointer pointing to the head of the projection. Thus, the structure in 1 is a projection of X, because the pointers lead to a head labeled X.

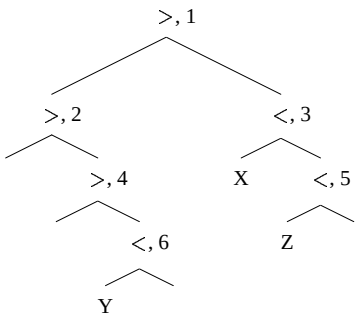
1. 

2. 

In a tree, a projection of a head X is maximal if the node immediately dominating the root of the projection is a projection of a head other than X, or if there is no node immediately dominating the root of the projection. For example, in the tree given in 3, the subtree whose root is labeled 2 is a maximal projection of head Y, because the node labeled 1 immediately dominating node 2 is a projection of head X. Similarly, the subtree with the root node labeled 5 is a maximal projection of head Z. The node labeled 4 is not the root of any maximal projection, because node 2 is a projection of

head Y, as is node 4.[1]

3.



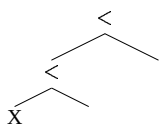
A maximal projection is the complement of a head X if the node immediately dominating the root of the maximal projection is a projection of head X and the maximal projection follows head X. In tree 3, for example, the maximal projection of head Z is a complement of head X.[2]

A maximal projection is a specifier of a head X if the node immediately dominating the root of the maximal projection is a projection of head X and the maximal projection precedes head X. In tree 3, the maximal projection of head Y is a specifier of head X, for example. A head can have any number of specifiers.[3]

[1]The nodes in the tree are numbered for illustrational purposes only; these number have no grammatical significance.

[2]The tree in 4 has two complement positions, but left-branching trees of this kind cannot be built by the structure building operations Merge and Move, as will follow from the definition of these functions in section 2.2.1.3.

4.



[3]It can be shown, however, that the language generated by a Minimalist Grammar allowing multiple specifiers per head can also be generated by a Minimalist Grammar with at most one specifier per head ([Sta99], [KK01], [Mic01b]).

2.2.1.1 Features

A Minimalist Grammar is given by a set of features, a lexicon, and two structure building functions. There are three kinds of features: syntactic features, phonetic features, and semantic features. The syntactic features determine how the structure building functions will apply to the lexical items and the trees derived from these. The syntactic features can be subdivided into category features, selection features, licensor features, and licensee features. Category features are used to identify the syntactic category of heads, e.g. `c` for complementizer, `i` for inflection, `d` for determiner, `n` for noun, etc. For each category feature there is a corresponding selection feature, written as '`=`' followed by a category feature. Verbs, for example, may have a selection feature `=d`, indicating that they combine with a projection of a head of category `d`, i.e., a determiner phrase. The licensor and licensee features play a role in movement. For example, a determiner phrase that needs to be assigned abstract case will have a licensee feature `-case` and a case-assigning head will have the corresponding licensor feature `+case`. Another linguistic example of a pair of licensor and licensee features is `+wh`, `-wh`. The heads of wh-phrases have the feature `-wh`. Assuming that wh-phrases move to the specifier position of a complementizer phrase, the heads of these phrases will have the feature `+wh`.

Semantic features are relevant for the semantic interpretation of phrases. The pronounceable aspect of a phrase is represented by a sequence of phonetic features. Since this dissertation is about syntactic parsing, we will not discuss the use of semantic features here. Also, for our purposes, the phonetic features can simply be thought of as strings over some alphabet, which roughly correspond to words.

2.2.1.2 Lexicon

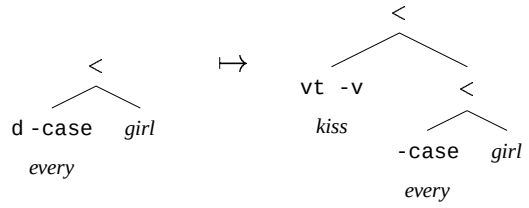
The lexicon consists of a finite set of lexical items, where each lexical item is a sequence of features. For example, the lexical item for the determiner *every* may look like `=n d -case` /εvri/ EVERY. The first three features are syntactic features, /εvri/ is the sequence of phonetic features for the word *every*, and EVERY stands for the semantic interpretation of the determiner *every*. It follows from the syntactic features of this lexical item that the determiner *every* selects a projection of a head of category `n`, that is, a noun phrase, that the head of the resulting phrase is of category `d`, that is, the resulting phrase is a determiner phrase, and that this phrase needs to be assigned abstract case. Throughout this dissertation, except in section 2.4, we will ignore the semantic features of a lexical item and use the orthographic representation of a word to stand for its phonetic features. So, we write `=n d -case` *every* instead of `=n d -case` /εvri/ EVERY. As another example, a complementizer whose projection hosts a wh-phrase may be represented by the lexical item `=t +wh c` ϵ. This lexical item selects a tensed phrase, attracts a wh-phrase, thus projects a phrase of category `c`, and it is phonologically empty, i.e., it corresponds to the empty word ϵ.

2.2.1.3 Structure Building Functions

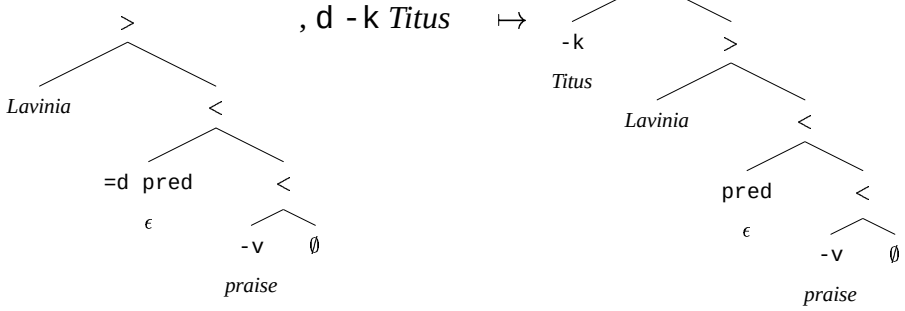
The two structure building functions available in a Minimalist Grammar are *merge* and *move*. New trees are constructed from other trees by either merging two trees into one, or by moving a subtree within a tree. Applications of the functions *merge* and *move* are triggered only by the syntactic features occurring in a tree.

Merge can apply in two ways, illustrated in 5 and 6 below. In the first situation, depicted in 5, the left-most feature of the verb *kiss* is the selection feature `=d`, indicating that *kiss* wants to select a determiner phrase. The left-most feature of the tree for

every girl is the category feature d, meaning that this structure is a determiner phrase. Since the selection feature =d and the category feature d match, the structure building function *merge* will apply to these two trees. It will produce a new tree in which the tree for *every girl* is attached as a complement to the verb *kiss*. The features triggering *merge*, =d and d, are deleted.

5. =d vt -v *kiss*, 

In the second situation, exemplified in 6, the left-most feature of the tree on the left is the selection feature =d, indicating that this tree wants to select a determiner phrase. The left-most feature of the determiner phrase *Titus* is the category feature d, implying that this simple tree is indeed a determiner phrase. The selection feature =d and the category feature d match and the structure building function *merge* will apply to these two trees. Since the complement position of the tree on the left is already occupied by a verb phrase, *Titus* will be attached as a specifier to this tree, which already has one specifier.[4] The features triggering *merge*, =d and d, are deleted.

6. 

[4] Formally, the notions 'specifier' and 'complement' are defined with regard to the head of a tree. We will use 'specifier of a tree' and 'complement of a tree' to mean 'specifier of the head of a tree' and 'complement of the head of a tree'.

Whether *merge* will attach the selected tree as a complement or as a specifier of the selecting tree depends on whether the latter tree is simple or complex. A tree that consists of just one node is a simple tree. Its complement position is available and will be filled by the tree selected for. A tree that has more than one node is a complex tree. Its complement position is already filled, so the selected tree will be attached as a specifier. Note that it follows from this specification of the structure building function *merge* that any derived tree can have at most one complement, but any number of specifiers. It does not matter for *merge* whether the tree that is selected is simple or complex.

The structure building function *merge* is formally defined as follows. A pair of trees τ_1 and τ_2 is in the domain of *merge* if the left-most feature of the head of τ_1 is `=x` and the left-most feature of the head of τ_2 is `x`. Then,

merge(τ_1, τ_2) = $<$ if τ_1 is simple, and

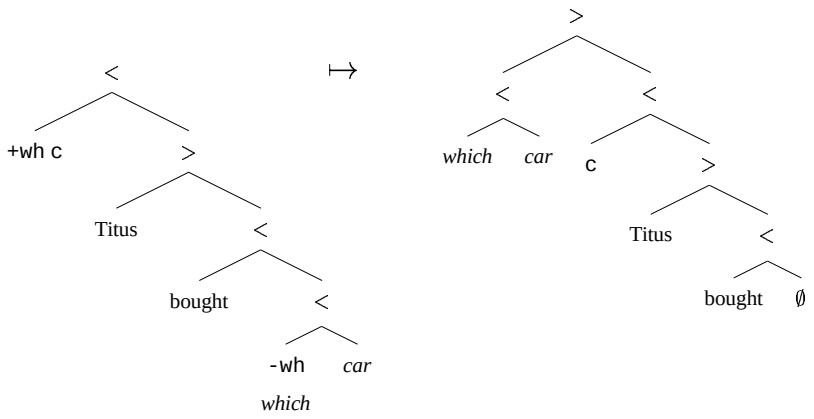
merge(τ_1, τ_2) = $>$ if τ_1 is complex,

where τ_1' is like τ_1 except that feature `=x` is deleted, and τ_2' is like τ_2 except that feature `x` is deleted.

If the left-most feature of the head of a tree is a licensor feature, and the tree contains another head whose left-most feature is a matching licensee feature, then the structure building function *move* will apply to this tree. An example is provided in 7. The licensee feature `-wh` of the head of the wh-phrase *which car* matches the licensor feature `+wh` on the head of the entire tree. The wh-phrase moves to a specifier position

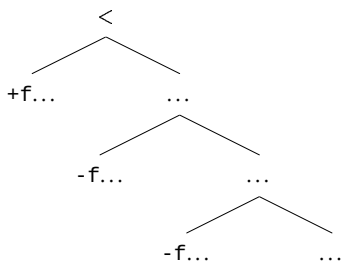
of the head possessing the licensor feature, leaving behind an empty node ∅. The two features triggering the application of *move*, `+wh` and `-wh`, are deleted.

7.



Movement is subject to the Shortest Movement Constraint: the structure building function *move* will not apply to a tree which has more than one head whose left-most feature is a licensee feature matching the licensor feature of the head of the tree, as schematically represented in 8.

8.



In the situation depicted in 8, the subtrees with the `-f` feature want to move to the same position, but moving any one subtree to that position will deprive the other subtree of its shortest move, as it will now have to move to the specifier of some higher head which has the licensee feature `+f`.[5]

[5] The Shortest Movement Constraint formulated in [Sta97] is a rather strong condition. If the head of the tree in 8 has two `+f` features, both subtrees could move to become specifiers of the same head, which would make both moves equally short from a linguistic point of view. However, under the current version of the Shortest Movement Constraint, the tree given in 8 is not in the domain of the function *move*, so no movements will take place.

Formally, a tree τ is in the domain of the structure building function *move* if the left-most feature of the head of τ is `+y` and τ has exactly one maximal subtree τ_0 the left-most feature of the head of which is `-y`. Then,

$move(\tau) =$ 

where τ_0' is like τ_0 except that feature `-y` is deleted, and τ' is like τ except that feature `+y` is deleted and subtree τ_0 is replaced by a single node without features.

2.2.1.4 Language

The set of trees CL(G) generated by a Minimalist Grammar G is the closure of the lexicon under the structure building functions *merge* and *move*, i.e., CL(G) is formed by applying the structure building functions *merge* and *move* in all possible ways to the items in the lexicon and the intermediate trees derived from these, as in the general framework of [KS96].

Assuming that a sentence is a projection of a head of category `c`, a complete tree is defined to be a tree without any syntactic features, except for the feature `c`, which must be labeling the head of the tree.

Now the language L(G) derivable by a Minimalist Grammar G consists of the yields of the complete trees in the set CL(G), where the yield of a tree is the concatenation of the phonetic features appearing at the leaves of the tree, ordered from left to right as in the tree. Hence, specifiers will precede their heads, which in turn will precede their complement, as in [Kay94]. However, in Minimalist Grammars this is not a matter of asymmetric c-command.

Note that the Minimalist Grammars introduced above only allow for overt phrasal

movement – all licensee features are strong features. Heads do not move and there is no covert phrasal movement.[6] The extensions required to model the latter two kinds of movement will be covered in section 2.4.

2.3 Examples

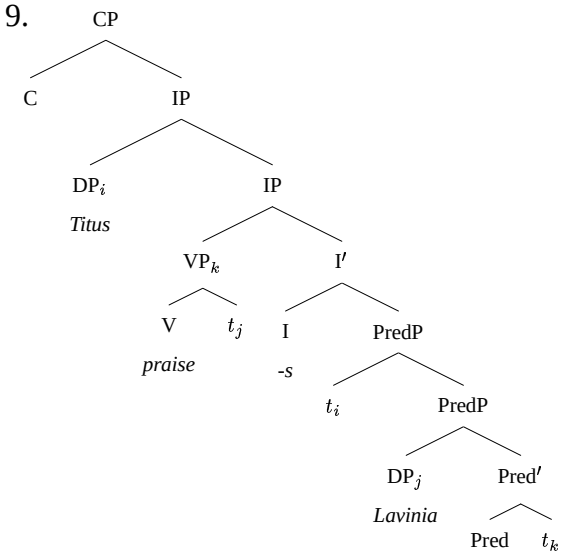
This section contains two examples of Minimalist Grammars and the languages they generate.

2.3.1 Linguistic Example

The tree in 9, repeated from section 1.4 in the previous chapter, illustrates a proposal by Mahajan [Mah00] regarding the derivation of sentences exhibiting an SVO order without resorting to (covert) head movement. Rather than moving V to I, the VP will move (overtly) to a specifier of I, but only after the object DP has left the VP.

[6] Head movement generally refers to overt movement of a head. In the *merge* operation defined above, the head of a selected phrase may be seen to move covertly to the head of the selecting phrase: the category feature of the head of the selected phrase will move and cancel, but its phonetic features do not move. Thus, a selection feature =x can be thought of as a weak head feature.

9.



This analysis can be modeled with a Minimalist Grammar whose lexicon contains the following six lexical items:

10. `d -k` *Lavinia*

11. `d -k` *Titus*

12. `=d vt -v` *praise*

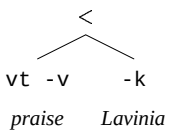
13. `=pred +v +k i` *s*

14. `=i c` ϵ

15. `=vt +k =d pred` ϵ

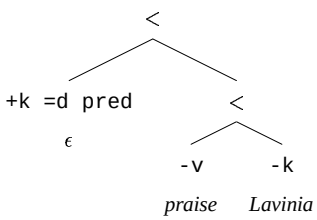
These lexical items participate in the following derivation. First, the lexical item `=d vt -v` *praise* will merge with the lexical item `d -k` *Lavinia* to form the verb phrase given in 16.

16.



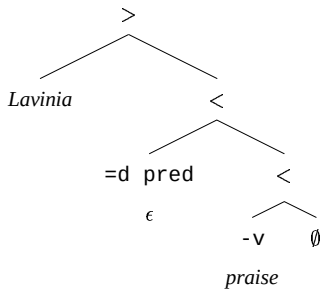
This verb phrase is then selected by the lexical item `=vt +k =d pred`. The result will be the following tree:

17.



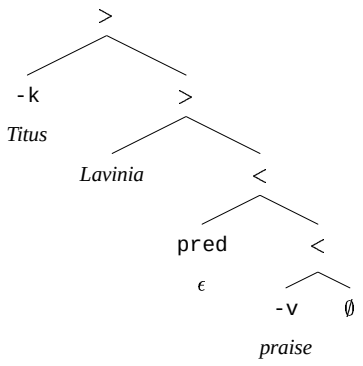
In the next step, the object *Lavinia* will move out of the verb phrase, creating a remnant verb phrase which will move in a later step of the derivation.

18.



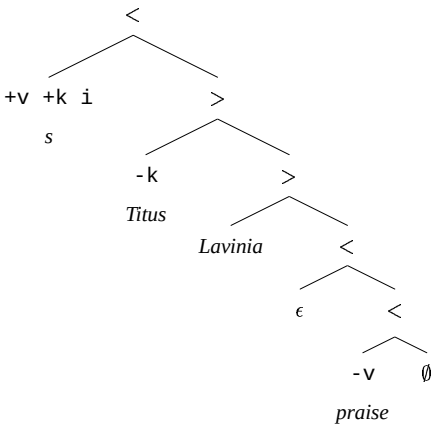
The tree in 18 will select the lexical item `d -k` *Titus*. This will produce the tree given in 19.

19.



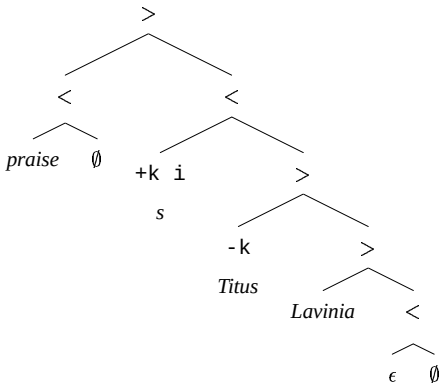
The tree in 19 will be merged with the lexical item `=pred +v +k i` *s*, resulting in the tree in 20.

20.



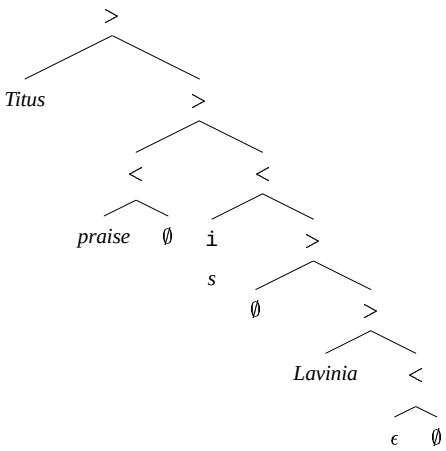
Next, the remnant verb phrase will move to a specifier of the inflectional head.

21.



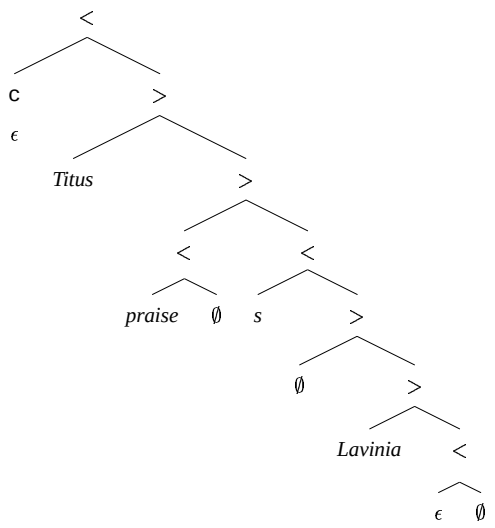
The pair of features +k and -k in the tree in 21 will trigger another application of *move*, which will yield the tree in 22.

22.



Finally, the tree in 22 will be selected by the lexical item `=i c` ϵ. The result of this final *merge* is the tree in 23.

23.



The tree in 23 contains only one unchecked feature, namely `c`, and this feature labels the head of the tree. Hence, the tree in 23 is a complete tree. We have thus established that the yield of this tree, ϵ *Titus praise s Lavinia* ϵ, i.e., *Titus praises Lavinia*, is a sentence in the language generated by the grammar given by the lexical items in 10 through 15 above.

Note that the total number of features occurring in the lexical items involved in the derivation of the sentence *Titus praises Lavinia* is 17. Therefore, the derivation of the tree in 23 takes exactly 8 steps, for each application of *merge* and *move* removes 2 features and the tree in 23 has one single feature left.

Note also that the lexical items of the grammar are concise representations of, possibly one-membered, chains, as mentioned in section 1.4 in the previous chapter. The foot of the chain is formed by the category feature of the lexical item, and every licensee feature adds another link to the chain.

2.3.2 Abstract Example

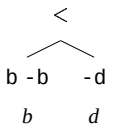
A Minimalist Grammar with the following eight lexical items generates the abstract language L = $\{a^n b^n d^n \mid n \geq 0\}$. This language is not context-free.

24. c ϵ
25. =a +d +b +a c ϵ
26. =b a -a *a*
27. =d b -b *b*
28. d -d *d*
29. =b +a a -a *a*
30. =d +b b -b *b*
31. =a +d d -d *d*

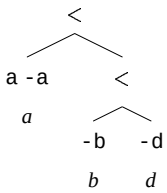
Below are some steps of the derivation of the sentence $a^2b^2d^2$. The grammar will "roll up" structures: a head of category a, which contributes one *a* to a sentence, will select a structure with n contiguous *b*'s, n contiguous *d*'s, and $n-1$ contiguous *a*'s, and move the tail of the structure containing the $n-1$ contiguous *a*'s to its specifier, thus creating a structure with n contiguous *a*'s, n contiguous *b*'s, and n contiguous *d*'s. The heads of category b and d have a similar effect.

Thus, after combining lexical items 27 and 28 into tree 32 and combining lexical item 26 and tree 32 into tree 33, tree 33 is selected by the lexical item =a +d d -d *d* which contributes one *d* and will move the lower *d* to its specifier. The tree resulting from this *merge* and *move* is given in 35.

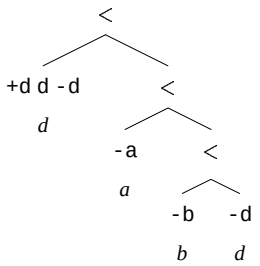
32.



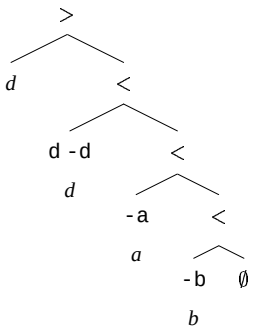
33.



34.

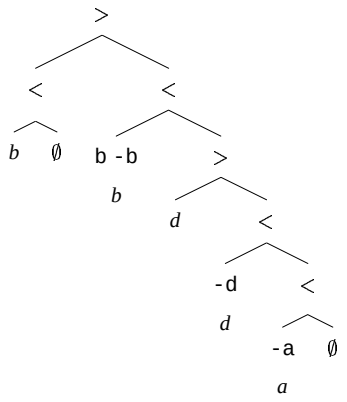


35.



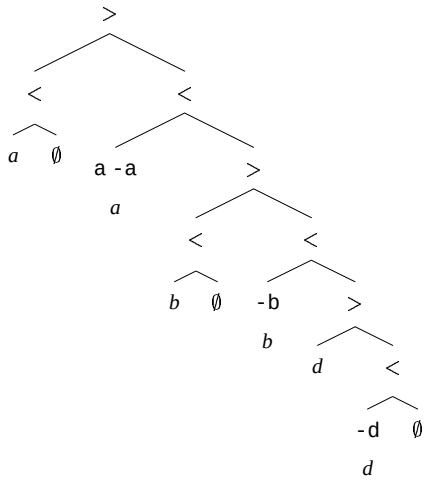
Selection of the tree in 35 by the lexical item `=d +b b -b` *b* will add one *b* and pull up the other *b* from below – see the tree in 36. Note that it is essential for the roll-up approach that the *d*'s move out of the projection of the lower head `-b` *b* before this projection will move itself. The latter movement is an instance of remnant movement, because the structure that is moving contains an empty node left behind by an earlier move operation.

36.



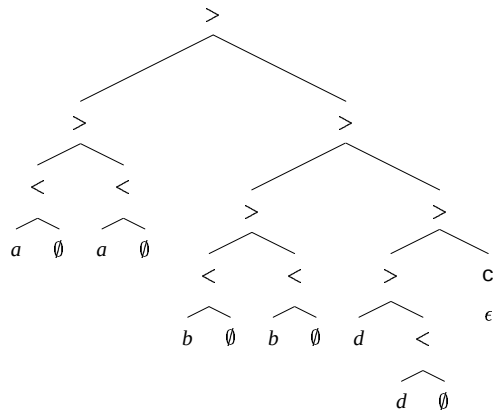
Lexical item 30 will roll up the tree in 36, producing the tree in 37.

37.



The tree in 37 is selected by the lexical item `=a +d +b +a c` ϵ. The move operations triggered by the sequence of licensor features `+d +b +a` and their corresponding licensee features in the tree in 37 will lead to the tree in 38. This is a complete tree with yield $a^2b^2d^2$.

38.



It is easy to see how this grammar for the language with tree counting dependencies can be extended to cover any number of counting dependencies. This shows that Minimalist Grammars are more expressive than Tree Adjoining Grammars and equivalent grammars, because those grammars can handle at most four counting dependencies [VW94]. It appears that Minimalist Grammars derive their greater expressive power from the mechanism of remnant movement [Sta99].

2.4 Head Movement and Covert Movement

The Minimalist Grammars defined in section 2.2.1 do not include provisions for head movement or covert phrasal movement. In the following two sections we will show how these two additional kinds of movement can be incorporated into Minimalist Grammars, following the original definition in [Sta97].

2.4.1 Head Movement

The reputation of head movement in linguistic theory is not entirely undisputed, e.g. [Cho01], [Mah00], and [KS00]. It is clearly unappealing to have a theory with two kinds of movement – phrasal movement and head movement. Head movement would be the first to go, because this kind of movement does not satisfy the Extension Principle, by

which Merge and Move operations must add another projection level to the projection which is targeted by these operations. Moving a phrase to a specifier position adds another projection level, but adjoining a head to another head does not, as observed in [GO96] and elsewhere. For example, moving a verb phrase to a specifier of I, as in 39, adds an IP level to the projection of I. However, adjoining a verbal head to an agreement head does not extend the projection of the latter head, as shown in 40.

39. Movement of VP to [Spec, IP]:

$$[_{\text{I}'}\ \text{I}\ [\ \ldots[_{\text{VP}}\ \ldots]\,]\,] \rightsquigarrow [_{\text{IP}}\ [_{\text{VP}}\ \ldots]_k\ [_{\text{I}'}\ \text{I}\ [\ \ldots t_k\ ]\,]\,]$$

40. Adjunction of V to Agr:

$$[_{\text{AGR}'}\ \text{Agr}\ [_{\text{VP}}\ \ldots \text{V}\ \ldots]\,] \rightsquigarrow [_{\text{AGR}'}\ [_{\text{AGR}}\ \text{V}_k\ \text{Agr}]\ [_{\text{VP}}\ \ldots t_k\ \ldots]\,]$$

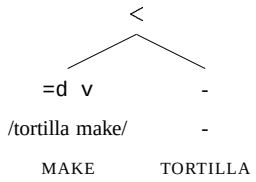
Proposals that abandon head movement or greatly reduce its role can be found in [KS00] and [Mah00], for example.

Besides the theoretical problem with the Extension Principle, there is also the formal result in chapter 3 of this dissertation, which implies that adding head movement to the Minimalist Grammars defined in section 2.2.1 does not increase the weak generative power of the formalism. This means that all sentences that can be derived with head movement can also be derived without head movement, albeit with slightly different structures.

To accommodate head movement in Minimalist Grammars, two additional selection features are introduced, =X and X=, where x is a category feature. When a head with a feature =X or X= selects a tree of category x, the phonetic features of the head of the latter tree will move to the head that selected the tree. Only the phonetic features of the head of the selected tree will move; the semantic features and any remaining

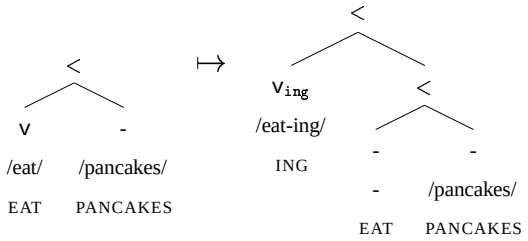
syntactic features will be left behind.[7] Thus, the selection features =X and X= can be thought of as strong head features.[8] Features =X and X= differ with regard to whether the phonetic features of the head that moves are prefixed or suffixed to the selecting head. A simple example of head movement with left adjunction is given in 41, where the verb *make* strongly selects the determiner phrase *tortilla*. If the lexical item for *make* were =D =d v *make*, the resulting structure would be the same, except that the phonetic form of the head would be /make tortilla/ rather than /tortilla make/.[9]

41. D= =d v /make/ MAKE, d /tortilla/ TORTILLA ↦



Another example of head movement is given in 42.

42. V= v_{ing} /-ing/ ING, ↦



To include head movement, the formal definition of the structure building function *merge* given in section 2.2.1.3 is extended in the following way. A pair of trees τ_1 and τ_2 is in the domain of *merge* if the left-most feature of the head of τ_1 is =x, =X or X=, and the left-most feature of the head of τ_2 is x. If the left-most feature of the head of τ_1 is =X or X=, τ_1 must be a head, i.e. a simple tree.[10]

[7] For that reason, we will write phonetic and semantic features separately in the examples to follow. To simplify matters, the phonetic features of a phonologically non-empty word or phrase are given by the orthographic form of the element enclosed in slashes.

[8] Compare footnote 6.

[9] The dashes in the trees indicate absence of features of a particular kind and only serve to increase readability.

[10] Head movement can be defined in such a way that it also applies to complex trees that have a selection feature =X or X=, but this does not seem a linguistically sensible operation.

If τ_1 is a simple tree and the left-most feature of its head is =X, then

$$merge(\tau_1, \tau_2) = \quad <\ \begin{array}{cc} \tau_1' & \tau_2' \end{array}$$

where τ_1' is like τ_1 except that feature =X is deleted and the phonetic features of the head of τ_2 are suffixed to the phonetic features of τ_1, and τ_2' is like τ_2 except that feature x and its phonetic features are deleted.

If τ_1 is a simple tree and the left-most feature of its head is X=, then

$$merge(\tau_1, \tau_2) = \quad <\ \begin{array}{cc} \tau_1' & \tau_2' \end{array}$$

where τ_1' is like τ_1 except that feature X= is deleted and the phonetic features of the head of τ_2 are prefixed to the phonetic features of τ_1, and τ_2' is like τ_2 except that feature x and its phonetic features are deleted.

The case where the left-most feature of the head of τ_1 is =x remains unchanged from the definition in section 2.2.1.3. Further discussion of head movement in Minimalist Grammars can be found in [Sta01b].

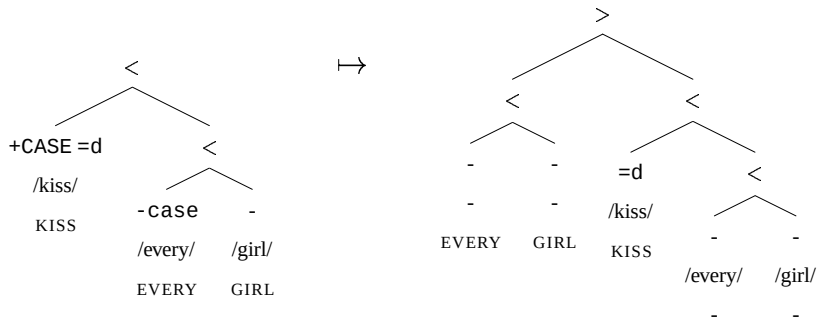
2.4.2 Covert Movement

In order to model covert phrasal movement in Minimalist Grammars, weak features are added to the set of licensee features. Weak licensee features will be written in uppercase, e.g. +CASE, to distinguish them from strong licensee features.[11] A weak

[11] This reverses the notation used in [Sta97]. While in our notation it no longer holds that all uppercase features are strong features and all lowercase features are weak features, this notation allows us to use all lowercase features in a Minimalist Grammar without head movement and covert phrasal movement.

feature causes the syntactic and semantic features of a phrase to move, but its phonetic features will stay put. An example of covert phrasal movement is given in 43. In the example, the determiner phrase *every girl* moves covertly to the specifier of the verb *kiss*.

43.



The formal definition of the structure building function *move* given in section 2.2.1.3 is adapted as follows. A tree τ is in the domain of the structure building function *move* if the left-most feature of the head of τ is +y or +Y, and τ has exactly one maximal subtree τ_0 the left-most feature of the head of which is -y. If the left-most feature of the head of τ is +Y, then

$$move(\tau) = \begin{array}{c} > \\ \tau_0' \quad \tau' \end{array}$$

where τ_0' is like τ_0 except that feature -y and all phonetic features are deleted, and τ' is like τ except that feature +Y is deleted and all syntactic and semantic features of τ_0 are deleted.

If the left-most feature of the head of τ is +y, the original clause in section 2.2.1.3 applies.

Covert phrasal movement is usually invoked to account for apparent mismatches between the order of words in a sentence and their scopal properties. For example, the

sentence in 44 has two meanings: either there is one thief who will rob all banks, or every bank will be robbed by one thief or another.

44. Some thief will rob every bank.

For the second meaning to obtain, the object determiner phrase *every bank* has to be in a position that is structurally higher than and to the left of the subject determiner phrase *some thief*. However, in the pronounced sentence, the object appears to the right of the subject. To explain this discrepancy, it is assumed that the object raises covertly to a position to the left of the subject. Kayne ([Kay98]) proposes to dispense with covert movement altogether and replace it with a combination of overt movements. In this respect, it is interesting to note that the formal equivalence result in chapter 3 entails that for Minimalist Grammars any sentence derived with covert movement can also be assigned a structure without covert movement.

2.5 Other Formalizations

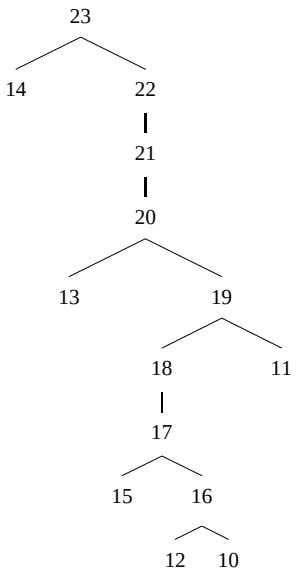
There are various other formal approaches to Transformational Grammars to be found in the literature, of which we mention [Rog98] and [Kra01].

Rogers [Rog98] introduces a logical language to describe the kinds of constraints on trees that are used in the theory of Government and Binding. The formalization of Government and Binding in terms of this logical language is then used to prove some results about the generative capacity of the theory. For the purpose of connecting the theory of Government and Binding to the hierarchy of formal languages, it is necessary to use an intermediate logical language, because Government and Binding is a constraint-based formalism, whereas the hierarchy of formal languages is based on grammars that are specified by rewriting rules. Direct proofs are hard to formulate, as it is not obvious how to cast constraints on representations as rewriting rules that

derive representations. As mentioned in section 2.1, unlike Government and Binding theory, the Minimalist Program posits two specific structure building operations for deriving representations, Merge and Move. Consequently, complexity results for Minimalist Grammars can be obtained in a direct way, by encoding the effects of Merge and Move in rewriting rules and vice versa, cf. [Mic98], [Mic01b], and chapter 3 of this dissertation.

Kracht [Kra01] offers a formalization of chains in linguistic theory. One way to describe chains is to use multi-dominance structures. A multi-dominance structure is a 'tree' in which a node can have more than one mother. Multi-dominance structures turn out to be closely related to derivation trees in Minimalist Grammars.

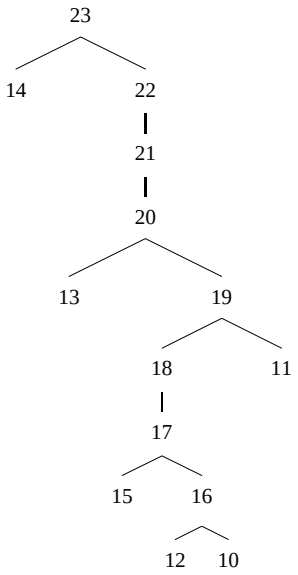
45.



Derivation trees are concise representations of derivations. The leaves of a derivation tree are lexical items and the intermediate nodes are trees derived from these via the functions *merge* and *move*. For example, the derivation tree of the sentence *Titus praises Lavinia* is given in 45. The numbered nodes stand for the trees given in the example in section 2.3.1. Derivation trees are important, because these are the structures that will be returned by the recognizers defined in chapters 4, 5, and 6.

The derivation tree in 45 is turned into a multi-dominance structure by adding dominance relations between a node representing a tree τ that is the result of the application of *move* and the node representing the head of the subtree τ_0 of τ that moves. For our example, adding these dominance relations leads to the multi-dominance structure in 46.[12]

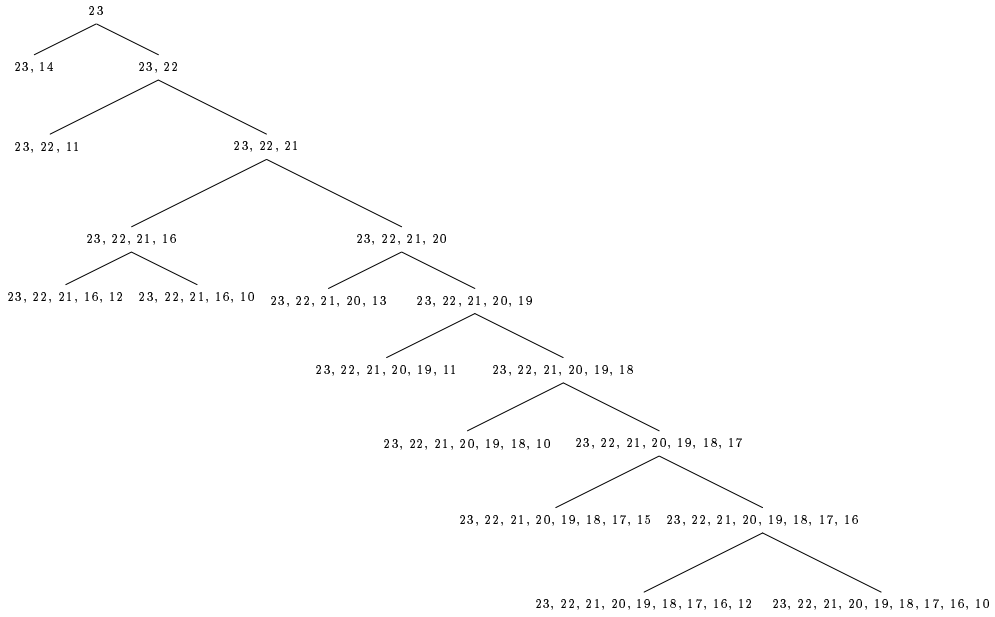
46.



Kracht [Kra01] provides a procedure for turning multi-dominance structures into trees. Applying this procedure to the structure in 46 and using the fact that specifiers precede their heads and heads precede their complements, the result will be the binary branching tree in 47.

[12] Add arcs from node 22 to node 11, from node 21 to node 16, and from 18 to 10.

47.



The tree in 47 is constructed from the the multi-dominance structure in 46 as follows. For every path from the root in the multi-dominance structure given in 46 there is a node in tree 47 labeled with the nodes of that path. The dominance relations between the nodes in tree 47 follow from the length of the paths: a node labeled with a path of length n immediately dominates any node labeled with a path of length $n + 1$ that is obtained from the path of length n by extending it with one step. The precedence relations follow from the relative order of heads and their specifiers and complements. For example, according to the derivation tree in 45, tree 14 merges with tree 22 to form tree 23. By the definition of *merge*, tree 22 will be attached as a complement, which follows the head. Therefore, in tree 47, the node labeled (23) will dominate the nodes (23, 14) and (23, 22) in that order. Similarly, the node labeled (23, 22, 21, 16) will precede the node (23, 22, 21, 20), because in the corresponding *move* operation, tree 16 will move to the specifier of tree 20 (resulting in the tree 21).

Now note that the tree in 47 corresponds to the tree which is the root of the multi-dominance structure, i.e. the surface tree in 23, except for the labels of the tree and

the fact that moved elements do not leave a trace, but rather a copy of themselves. However, the derivation tree contains enough information to assign a label '$<$' or '$>$' to the internal nodes of tree 47. For example, the label (23, 22, 21, 20, 19) can be replaced by '$>$', because tree 19 is the result of complex tree 18 selecting lexical item 11. Since tree 18 is complex, lexical item 11 will occupy a specifier position, which is to the left of tree 18, whose head is the head of tree 19. Furthermore, for a Minimalist Grammar without covert movement and head movement, all copies of moved elements can be deleted, except for the one dominated by the 'highest' immediate mother. For example, the node labeled (23, 22, 21, 16, 10) in tree 47, which is a copy of the determiner phrase *Lavinia*, can be replaced by $\emptyset$, because in tree 46, the node labeled 10 is an immediate daughter of the node labeled 16 and the node labeled 18, but node 18 is closer to the root than node 16. Consequently, in tree 47, *Lavinia* will be a daughter of node (23, 22, 21, 20, 19, 18). (Note that the third node corresponding to *Lavinia*, (23, 22, 21, 20, 19, 18, 17, 16, 10), is removed because the subtree rooted in node (23, 22, 21, 20, 19, 18, 17, 16) will be replaced by $\emptyset$ – this is the trace of the verb phrase).

Multi-dominance structures so provide an interesting way of mapping a derivation tree onto a surface tree without having to construct all the intermediate trees.

2.6 Concluding Remarks

In this chapter we have defined the Minimalist Grammars that the recognizers in chapter 4, 5, and 6 will use to recognize sentences. In fact, the recognizers will be defined for Minimalist Grammars without head movement and covert movement, because for these grammars Michaelis' equivalence result ([Mic98]) suggests an elegant reformulation in terms of chains. This is not a serious restriction, however, because Michaelis' result also holds for Minimalist Grammars with head movement and covert movement,

so the reformulation could be extended to cover these two kinds of movement as well.

CHAPTER 3

Minimalist Languages

In this chapter[1] we will fix the position of Minimalist Grammars as defined in [Sta97] in the hierarchy of formal languages. Michaelis ([Mic98]) has shown that the set of languages generated by Minimalist Grammars falls within the set of languages generated by Multiple-Context Free Grammars ([SHF91]). In this chapter we will prove the reverse by showing how to construct a Minimalist Grammar for an arbitrary Multiple Context-Free Grammar such that both grammars define the same language. A priori, it is rather remarkable that it is possible to convert a Multiple Context-Free Grammar into an equivalent Minimalist Grammar, because the structures produced by Minimalist Grammars are predominantly right-branching, whereas Multiple Context-Free Grammars have no preference for either direction of branching.

The Minimalist Grammars that we will use in this chapter do not allow for head movement or covert phrasal movement, and so are simpler than the ones defined in [Sta97] and used in Michaelis ([Mic98]). Let ML be the set of languages definable by 'full' Minimalist Grammars, ML^- the set of languages definable by Minimalist Grammars without head movement and covert phrasal movement, and MCFL the set of languages definable by Multiple Context-Free Grammars. Michaelis ([Mic98]) has shown that ML $\subseteq$ MCFL. Obviously, $ML^- \subseteq$ ML. The result presented in this chapter implies that MCFL $\subseteq ML^-$. Hence, we conclude that MCFL = ML.[2] Then it

[1] An earlier version of this chapter appeared as [Har01].

[2] Michaelis ([Mic01b]) independently established the equivalent claim that Minimalist Grammars and Linear Context-Free Rewriting Systems ([VWJ87]) define the same class of string languages.

follows immediately from earlier results that Minimalist Grammars are weakly equivalent to Linear Context-Free Rewriting Systems ([VWJ87]), Multi-Component Tree-Adjoining Grammars ([Wei88]), and Simple Positive Range Concatenation Grammars ([Bou98]). The result reported in this chapter also implies that $ML^{-} = ML$, that is, head movement and covert phrasal movement do not increase the generative power of Minimalist Grammars.[3]

This chapter is structured as follows. We will first provide the definitions for Minimalist Grammars and Multiple Context-Free Grammars, then describe and illustrate the encoding of a given Multiple Context-Free Grammar into a Minimalist Grammar, and finally prove that the Minimalist Grammar thus obtained defines the same language as the original Multiple Context-Free Grammar.

3.1 Minimalist Grammars

In this section we will give a definition of Minimalist Grammars as introduced by [Sta97], but without head movement and covert phrasal movement.

Definition 1. *A tree over a feature set F is a quintuple* $\tau = (N_\tau, \lhd_\tau, \prec_\tau, <_\tau, Label_\tau)$ *which meets the following three conditions:*

[3]The linguistic consequence of this result is that one cannot use word order to justify the need for head movement and covert phrasal movement, because whatever order is derivable using a Minimalist Grammar with head movement and covert phrasal movement can also be derived by a Minimalist Grammar without these mechanisms. This argument may bear upon certain cases of head movement, but in this form it does not apply in any meaningful way to covert phrasal movement, because, by its very nature, covert movement has no effect on word order. Covert movement does affect the structure assigned to a sentence, and so its meaning. In the case of quantifier raising, for example, a quantified expression will move covertly to a position which c-commands other quantified expressions that in the sentence appear to the left of the quantified expression, thus allowing the expression to be interpreted in a position in which it can take scope over these other quantified expressions. When covert phrasal movement is removed from a Minimalist Grammar by transforming it into a weakly equivalent Multiple Context-Free Grammar and transforming this grammar into a weakly equivalent Minimalist Grammar without covert phrasal movement, the relation between a phrase that moved covertly and the position it moved to in the original grammar survives, but it does not obtain as the result of covert phrasal movement. In this sense, covert phrasal movement is linguistically unnecessary.

1. *Triple (N_τ, $\lhd_\tau$, $\prec_\tau$) is a finite, binary ordered tree: N_τ is a non-empty set of nodes, $\lhd_\tau \subseteq N_\tau \times N_\tau$ denotes the relation of immediate dominance; $\prec_\tau \subseteq N_\tau \times N_\tau$ denotes the relation of immediate precedence.*

2. *$<_\tau \subseteq N_\tau \times N_\tau$ denotes the relation of immediate projection: for any two nodes ν_1, $\nu_2 \in N_\tau$ which are sisters in τ, either ν_1 projects over ν_2, $\nu_1 <_\tau \nu_2$, or ν_2 projects over ν_1, $\nu_2 <_\tau \nu_1$.*

3. *$Label_\tau$ is a function from N_τ to F*, assigning to each leaf of τ a finite sequence of features from F.*

Let τ = (N_τ, $\lhd_\tau$, $\prec_\tau$, $<_\tau$, $Label_\tau$) be tree over a feature set F. Tree τ is a simple tree if it consists of just one node, otherwise it is a complex tree. If τ is a complex tree, then there are proper subtrees τ_0 and τ_1 of τ such that τ = $[_<\tau_0, \tau_1]$ or τ = $[_>\tau_0, \tau_1]$, where $[_<\tau_0, \tau_1]$ denotes a tree whose root immediately dominates subtrees τ_0 and τ_1 and in which the root of τ_0 immediately projects over and precedes the root of τ_1, and $[_>\tau_0, \tau_1]$ denotes a tree whose root immediately dominates subtrees τ_0 and τ_1 and in which the root of τ_0 immediately precedes the root of τ_1 and the root of τ_1 immediately projects over τ_0.

If τ is a simple tree, then its head is the single node making up τ. If τ is a complex tree $[_<\tau_0, \tau_1]$, then the head of τ is the head of τ_0; if τ is a complex tree $[_>\tau_0, \tau_1]$, then the head of τ is the head of τ_1. Tree τ is a projection of node $\nu \in N_\tau$, if, and only if, τ is a tree whose head is ν.

A subtree τ_0 of τ is maximal if it is τ or if the smallest subtree of τ properly including τ_0 has a head other than the head of τ_0. A proper subtree τ_0 of τ is a specifier of the head of τ if τ_0 is a maximal subtree, and the smallest subtree of τ properly including τ_0 is a projection of the head of τ, and the head of τ_0 precedes the head of τ. A proper subtree τ_0 of τ is a complement of the head of τ if τ_0 is a maximal subtree,

and the smallest subtree of τ properly including τ_0 is a projection of the head of τ, and the head of τ precedes the head of τ_0.

Tree τ is said to have a feature $\mathtt{f} \in$ F if the first feature of the sequence that labels the head of τ is f.

Definition 2. *A Minimalist Grammar G is a quadruple G = (V, Cat, Lex, $\mathcal{F}$), which satisfies the following four conditions.*

1. *V = P ∪ I is a finite set of non-syntactic features, consisting of a set of phonetic features P and a set of semantic features I.*

2. *Cat = $base \cup select \cup licensors \cup licensees$ is a finite set of syntactic features, such that for each feature* x *∈ $base$ there is a feature* =x *∈ $select$, and for each feature* +y *∈ $licensors$ there is a feature* -y *∈ $licensees$. The set $base$ minimally contains the distinguished category feature* c.

3. *Lex is a finite set of trees over V ∪ Cat such that for each τ = (N_τ, $\lhd_\tau$, $\prec_\tau$, $<_\tau$, $Label_\tau$) ∈ Lex, the function $Label_\tau$ assigns a string from Cat*P*I* to each leaf of τ.*

4. *The set $\mathcal{F}$ consists of the structure building functions $merge$ and $move$, which are defined as follows:*

   (a) *A pair of trees (τ, υ) is in the domain of $merge$ if τ has feature* =x *∈ $select$ and υ has feature* x *∈ $base$. Then,*

   $merge(\tau, \upsilon) = [_< \tau', \upsilon']$ *if τ is simple, and*

   $merge(\tau, \upsilon) = [_> \upsilon', \tau']$ *if τ is complex,*

   where τ' is like τ except that feature =x *is deleted, and υ' is like υ except that feature* x *is deleted.*

(b) A tree τ is in the domain of $move$ if τ has feature +y $\in licensors$ *and τ has exactly one maximal subtree τ_0 that has feature* -y $\in licensees$*. Then,*

$move(\tau) = [_{>}\tau_0', \tau']$,

where τ_0' is like τ_0 except that feature -y *is deleted, and τ' is like τ except that feature* +y *is deleted and subtree τ_0 is replaced by a single node without any features.*

Let G = (V, Cat, Lex, $\mathcal{F}$) be a Minimalist Grammar. Then CL(G) = $\cup_{k\in\mathbb{N}}$ CL^k(G) is the closure of the lexicon under the structure building functions in $\mathcal{F}$, where CL^k(G), $k \in \mathbb{N}$ are inductively defined by:

1. CL^0(G) = Lex
2. CL^{k+1}(G) = CL^k(G) $\cup$ $\{merge(\tau, \upsilon) \mid (\tau, \upsilon) \in$ Dom($merge$) $\cap$ (CL^k(G) $\times$ CL^k(G))$\}$ $\cup$ $\{move(\tau) \mid \tau \in$ Dom($move$) $\cap$ CL^k(G)$\}$,

where Dom($merge$) and Dom($move$) are the domains of the functions $merge$, $move$ $\in \mathcal{F}$. Let τ be a tree in CL(G). Tree τ is a tree of category a if the head of τ does not contain any syntactic features except for the feature a $\in base$. Tree τ is a complete tree if it is a tree of category c and no node other than the head of τ contains any syntactic features. The yield Y(τ) of τ is the concatenation of the phonetic features in the labels of the leaves of τ, ordered by precedence. The language derivable by G consists of the yields of the complete trees in CL(G): L(G)=$\{$Y(τ) $\mid$ $\tau \in$ CL(G), τ is complete$\}$.

3.2 Multiple Context-Free Grammars

In this section we will define the class of Multiple Context-Free Grammars. A Multiple Context-Free Grammar is a Generalized Context-Free Grammar which satisfies

some additional conditions. In section 3.2.1 we will provide a definition of Generalized Context-Free Grammars. The conditions on Multiple Context-Free Grammars are discussed in section 3.2.2. In section 3.2.3, we will describe a normal form for Multiple Context-Free Grammars that will be used in this chapter.

3.2.1 Generalized Context-Free Grammars

Generalized Context-Free Grammars are introduced in [Pol84]. We will follow the definition found in [Mic98].

Definition 3. *A Generalized Context-Free Grammar is a quintuple G = (N, O, F, R, S) for which the following five conditions hold:*

1. *N is a finite, non-empty set of non-terminal symbols.*
2. *O is a set of finite tuples of finite strings over a finite, non-empty set of terminal symbols* Σ, $\Sigma \cap N = \emptyset$: $O \subseteq \bigcup_{n \in \mathbb{N}^+} (\Sigma^*)^n$.
3. *F is a finite set of partial functions from finite products of O to O:* $F \subseteq \bigcup_{n \in \mathbb{N}} F_n$, *where* F_n *is the set of partial functions from* O^n *to O.*
4. *R is a finite set of rewriting rules:* $R \subseteq \bigcup_{n \in \mathbb{N}} ((F_n \cap F) \times N^{n+1})$.
5. $S \in N$ *is the distinguished start symbol.*

Let G = (N, O, F, R, S) be a Generalized Context-Free Grammar. A rule r = (g, A, B_1, ..., B_n) $\in$ ($F_n \cap$ F) $\times$ N^{n+1} of this grammar is usually written as A $\rightarrow$ g[B_1, ..., B_n] if $n > 0$, and A $\rightarrow$ g if n = 0. A rule for which $n > 0$ will be called a non-terminating rule; a rule for which n = 0 will be called a terminating rule. Note that F_0 is the set of constants in O. Hence, for a terminating rule A $\rightarrow$ g, $g \in$ O.

For any non-terminal symbol A $\in$ N and $k \in \mathbb{N}$, the set L_G^k(A) $\subseteq$ O is defined recursively as follows:

1. If R contains a terminating rule A $\rightarrow g$, then $g \in \mathrm{L}_G^0$(A).

2. If $\theta \in \mathrm{L}_G^k$(A), then $\theta \in \mathrm{L}_G^{k+1}$(A).

3. If R contains a non-terminating rule A $\rightarrow g$[B_1, …, B_n] and $\theta_i \in \mathrm{L}_G^k(\mathrm{B}_i)$, $1 \leq i \leq n$, and $g(\theta_1, \ldots, \theta_n)$ is defined, then $g(\theta_1, \ldots, \theta_n) \in \mathrm{L}_G^{k+1}$(A).

If $\theta \in \mathrm{L}_G^k$(A) for some $k \in \mathbb{N}$, then θ is called an A-phrase in G. The language derivable from A in G is the set L_G(A) of all A-phrases in G: $\mathrm{L}_G(\mathrm{A}) = \bigcup_{k \in \mathbb{N}} \mathrm{L}_G^k(\mathrm{A})$. The language L(G) generated by G is the set of all S-phrases in G: L(G) = L_G(S).

3.2.2 m-Multiple Context-Free Grammars

The following definition of an m-Multiple Context-Free Grammar is based on definitions provided in [SHF91] and [Mic98].

Definition 4. *For any $m \in \mathbb{N}^+$, an m-Multiple Context-Free Grammar is a Generalized Context-Free Grammar G = (N, O, F, R, S) which satisfies the following four conditions:*

1. *The dimension of the tuples in O is bounded: O = $\bigcup_{i=1}^{m} (\Sigma^*)^i$.*

2. *For function $g \in F$, let n(g) $\in \mathbb{N}$ be the number of arguments of g, i.e., $g \in F_{n(g)}$. For any function $g \in F$, the following 3 conditions hold:*

   (a) *The dimension of the result of g and the dimensions of the arguments of g are fixed, that is, there are numbers r(g) $\in \mathbb{N}$, $d_i(g) \in \mathbb{N}$, $1 \leq i \leq$ n(g), such that g is a function from $(\Sigma^*)^{d_1(g)} \times \ldots \times (\Sigma^*)^{d_{n(g)}(g)}$ to $(\Sigma^*)^{r(g)}$.*

   (b) *The effect of g is fixed in the following sense. Let X = $\{x_{ij} \mid 1 \leq i \leq$ n(g), $1 \leq j \leq d_i(g)\}$ be a set of pairwise distinct variables, and define x_i = (x_{i1}, …, $x_{id_i(g)}$), $1 \leq i \leq$ n(g). Let g^h be the h^{th} component of g, $1 \leq h \leq$ r(g),*

that is, $g(\theta) = (g^1(\theta), \ldots, g^{r(g)}(\theta))$ for any $\theta = (\theta_1, \ldots, \theta_{n(g)}) \in (\Sigma^*)^{d_1(g)} \times \ldots \times (\Sigma^*)^{d_{n(g)}(g)}$. Then for each component g^h, there is a fixed number $l_h(g) \in \mathbb{N}$ such that g^h is represented by the following concatenation of constant strings in Σ^* and variables in X:

$$g^h(x_1, \ldots, x_{n(g)}) = \alpha_{h0} z_{h1} \alpha_{h1} z_{h2} \ldots z_{hl_h(g)} \alpha_{hl_h(g)},$$

where $\alpha_{hl} \in \Sigma^*$, $0 \leq l \leq l_h(g)$, and $z_{hl} \in X$, $1 \leq l \leq l_h(g)$.

(c) For each pair (i, j), $1 \leq i \leq n(g)$, $1 \leq j \leq d_i(g)$, there is at most one h, $1 \leq h \leq r(g)$, and at most one l, $1 \leq l \leq l_h(g)$, such that variable z_{hl} in the representation of component g^h of g is the variable $x_{ij} \in X$.

3. *The dimension of any non-terminal symbol $A \in N$ is fixed, that is, for every non-terminal symbol $A \in N$ there is a number $d(A) \in \mathbb{N}$ such that for any rule $A \rightarrow g[B_1, \ldots, B_{n(g)}]$ in R, $r(g) = d(A)$ and $d_i(g) = d(B_i)$, $1 \leq i \leq n(g)$.*

4. *The dimension of the distinguished start symbol is 1: $d(S) = 1$.*

Condition 2.(c) is referred to as the anti-copying condition, because it prevents any variable $x_{ij} \in$ X from occurring more than once in the whole of the representations of the components that define a function g.

Example 1. The 2-Multiple Context-Free Grammar G = ({H, S}, $\bigcup_{i=1}^{2}$ ({1, 2, 3, 4}*)^i, {g, f}, {H $\rightarrow$ (ϵ, ϵ), H $\rightarrow$ g[H], S $\rightarrow$ f[H]}, S), where $g[(x_{11}, x_{12})]$ = ($1x_{11}2$, $3x_{12}4$) and $f[(x_{11}, x_{12})]$ = ($x_{11}x_{12}$), generates the language $\{1^n2^n3^n4^n \mid n \geq 0\}$.

3.2.3 Normal Form for Multiple Context-Free Grammars

The Multiple Context-Free Grammars used in the proof of equivalence are assumed to be in a particular form. This section defines this normal form and shows that any Multiple Context-Free Grammar can be given in this form. The components of any

function $g \in$ F of the m-Multiple Context-Free Grammar G in lemma 1 are assumed to be represented as in definition 4, part 2.(b).

Lemma 1. *For any m-Multiple Context-Free Grammar G = (N, O, F, R, S) there is an m-Multiple Context-Free Grammar G′ = (N′, O′, F′, R′, S′) which is weakly equivalent to G and which satisfies the following four conditions:*

(a) For any non-terminal symbol A ∈ N′ which appears in the left-hand side of some terminating rule in R′, d(A) = 1.

(b) Any $g \in F'$, n(g) > 0, satisfies the non-erasure condition: for each pair (i, j), $1 \leq i \leq$ n(g), $1 \leq j \leq d_i(g)$, there is at least one h, $1 \leq h \leq$ r(g), and at least one l, $1 \leq l \leq l_h(g)$, such that variable z_{hl} in the representation of component g^h of g is the variable $x_{ij} \in X$.

(c) Any $g \in F'$, n(g) > 0, is free of syncategorematically introduced symbols: for all pairs (h, l), $1 \leq h \leq$ r(g), $0 \leq l \leq l_h(g)$, the constant string $\alpha_{hl} \in \Sigma^*$ in the representation of component g^h of g is the empty string.

(d) Grammar G′ is doublet-free: there is no non-terminating rule $r \in R'$ such that there is a non-terminal symbol T ∈ N′ which occurs more than once among the non-terminal symbols in the right-hand side of r.

Proof. The proof of lemma 2.2 in ([SHF91]) shows how to construct, for any m-Multiple Context-Free Grammar G = (N, O, F, R, S), another m-Multiple Context-Free Grammar G′ = (N′, O′, F′, R′, S′) satisfying conditions (a), (b), and (c), and such that L(G) = L(G′).

Possible doublets in G′ are removed by replacing a non-terminating rule $r \in$ R′ containing a doublet, i.e., r = A $\rightarrow g$[B_1, …, $B_{n(g)}$] such that $B_i = B_j$ = T ∈ N′, $1 \leq i < j \leq$ n(g), with a rule r' = A $\rightarrow g$[B_1, …, B_{j-1}, B'_j, B_{j+1}, …, $B_{n(g)}$], where

B'_j is a fresh non-terminal symbol, and adding the rule r'' = $B'_j \rightarrow id_{d(B_j)}[B_j]$, where $id_{d(B_j)}$ is the identity function with one argument of dimension d(B_j). Let H be the resulting grammar: H = (N′ ∪ $\{B'_j\}$, O′, F′ ∪ $\{id_{d(B_j)}\}$, (R′ ∪ $\{r', r''\}$) - $\{r\}$, S′). H contains one doublet less than G′ and does not violate conditions (a), (b), or (c). For every derivation of a U-phrase θ in G′ involving rule r, U ∈ N′, there is a derivation of a U-phrase θ in H using rules r' and r'', and vice versa. Derivations not depending on rule r or rules r' and r'' are valid derivations in both G′ and H. Hence, $L_{G'}$(U) = L_H(U), U ∈ N′. Then, in particular, L(G′) = $L_{G'}$(S′) = L_H(S′) = L(H). Repeated replacements and additions of rules will eventually produce an m-Multiple Context-Free Grammar G″ = (N″, O′, F″, R″, S′) for which L(G″) = L(G′) = L(G), and which satisfies conditions (a), (b), (c), and (d). □

Example 2. The 2-Multiple Context-Free Grammar G = ($\{$A, B, C, D, E, F, H, S$\}$, $\bigcup_{i=1}^{2}$ ($\{$a, b, c, d$\}^*$)^i, $\{g, f, h\}$, {A → 1, B → 2, C → 3, D → 4, E → ϵ, F → ϵ, H → h[E, F], H → g[A, B, C, D, H], S → f[H]}, S), where $h[(x_{11}), (x_{21})] = (x_{11}, x_{21})$, $g[(x_{11}), (x_{21}), (x_{31}), (x_{41}), (x_{51}, x_{52})] = (x_{11}x_{51}x_{21}, x_{31}x_{52}x_{41})$, and $f[(x_{11}, x_{12})] = (x_{11}x_{12})$, is a grammar in normal form which generates the language $\{1^n2^n3^n4^n \mid n \geq 0\}$.

3.3 Construction of Minimalist Grammar

This section shows how to construct a Minimalist Grammar G′ = (V, Cat, Lex, $\mathcal{F}$) for a given m-Multiple Context-Free Grammar G = (N, O, F, R, S) in normal form such that L(G) = L(G′). The general idea behind the construction of G′ is that for any non-terminal symbol A ∈ N and A-phrase (p_1, …, $p_{d(A)}$), the items in Lex allow for a derivation of a tree that has d(A) specifiers, the yields of which are the components p_i, $1 \leq i \leq$ d(A), and a head labeled with feature $\mathsf{a} \in base$ identifying non-terminal symbol A. Each specifier has a feature $\mathsf{-a}_t \in licensees$, $1 \leq t \leq$ d(A), so that the

specifiers can be moved to form strings that correspond to the values produced by the functions in F.

3.3.1 Non-syntactic Features

There is no need for G′ to have semantic features: I = ∅. The set of phonetic features will correspond to the set of terminal symbols of G: P = Σ for O ⊆ $\bigcup_{i=1}^{m} (\Sigma^*)^i$. Hence, regarding the non-syntactic features of G′, V = P ∪ I = Σ.

3.3.2 Syntactic Features

The set of syntactic features Cat is the union of the sets *base*, *select*, *licensees*, and *licensors*, the contents of which are defined as follows. For every non-terminal symbol A ∈ N, there is a feature $\mathsf{a} \in base$, a feature $\mathsf{=a} \in select$, and features $\mathsf{+a}_t \in licensors$, and features $\mathsf{-a}_t \in licensees$, $1 \leq t \leq$ d(A). These features are said to correspond to non-terminal symbol A. The features corresponding to non-terminal symbol S are $\mathsf{c} \in base$, $\mathsf{=c} \in select$, $\mathsf{+c}_1 \in licensors$, and $\mathsf{-c}_1 \in licensees$.[4]

Let $R_e = \langle r_1, \ldots, r_{|R|} \rangle$ be an enumeration of the rules in R. For every non-terminating rule r_s = A → *g*[B_1, …, $B_{n(g)}$] in R_e, there are features $\mathsf{a}_t^s \in$ *base*, $\mathsf{=a}_t^s \in$ *select*, $1 \leq t \leq$ d(A)+1 = r(*g*)+1, and features $\mathsf{+a}_t^s \in$ *licensors*, $\mathsf{-a}_t^s \in$ *licensees*, $1 \leq t \leq$ d(A) = r(*g*). These features are said to correspond to rule r_s. The features corresponding to a non-terminating rule r_s = S → *g*[B_1, …, $B_{n(g)}$] are c_1^s, $\mathsf{c}_2^s \in$ *base*, $\mathsf{=c}_1^s$, $\mathsf{=c}_2^s \in$ *select*, and features $\mathsf{+c}_1^s \in$ *licensors*, $\mathsf{-c}_1^s \in$ *licensees*.

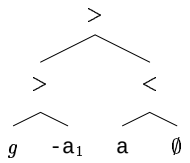
[4] By definition 4, d(S) = 1.

3.3.3 Structure Building Functions

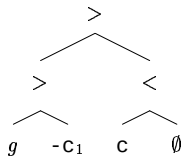
The structure building functions in $\mathcal{F}$ are prescribed by definition 2: $\mathcal{F}$ = {*merge*, *move*}.

3.3.4 Lexicon

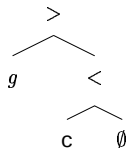
The contents of Lex are specified as follows. Let $R_e = \langle r_1, \ldots, r_{|R|} \rangle$ be an enumeration of the rules in R. For a terminating rule r_s = A $\rightarrow$ g in R_e, A $\neq$ S, the following lexical item is associated with r_s:[5]



For a terminating rule r_s = S $\rightarrow$ g in R_e, the following two lexical items will be associated with rule r:



and



[5] In a tree, ∅ represents the empty label. Each internal node of a tree will be labeled '<' or '>', indicating which of the two immediate daughters projects over the other.

Next, consider a non-terminating rule r_s = A $\rightarrow$ g[B_1, . . . , $B_{n(g)}$] in R_e,[6] and assume that the components of g are represented by $g^h(x_1, \ldots, x_{n(g)}) = z_{h1} \ldots z_{hl_h(g)}$, $z_{hl} \in X = \{x_{ij} \mid 1 \leq i \leq n(g), 1 \leq j \leq d_i(g)\}$, $x_i = (x_{i1}, \ldots, x_{id_i(g)})$, $1 \leq i \leq n(g)$, $1 \leq h \leq r(g)$, $1 \leq l \leq l_h(g)$, according to definition 4 and lemma 1. Define the set $F^+_{r_s} = \{+b_{ij} \mid 1 \leq i \leq n(g), 1 \leq j \leq d(B_i)\}$.[7] If A $\neq$ S, then the complex of lexical items associated with rule r_s consists of the following r(g)+2 lexical items:

$$=a^s_1 \;\; +a^s_{r(g)} \ldots +a^s_1 \;\; a,$$

and

$$=a^s_{h+1} \;\; f_{hl_h(g)} \ldots f_{h1} \;\; a^s_h \;\; -a^s_h \;\; -a_h,$$

$1 \leq h \leq r(g)$, where feature $f_{hl} \in F^+_{r_s}$ is feature $+b_{ij}$ if, and only if, variable $z_{hl} \in X$ is variable x_{ij}, $1 \leq l \leq l_h(g)$, and

$$=b_1 \ldots =b_{n(g)} \;\; a^s_{r(g)+1}.$$

If A = S, then non-terminating rule r_s = S $\rightarrow$ g[B_1, . . . , $B_{n(g)}$] in R_e is associated with the following complex of 2·r(g)+2 lexical items:[8]

$$=c^s_1 \;\; +c^s_1 \;\; c,$$

and

$$=c^s_2 \;\; f_{1l_1(g)} \ldots f_{11} \;\; c^s_1 \;\; -c^s_1 \;\; -c_1,$$

[6] The feature in *base* corresponding to non-terminal symbol B_i is written b_i, $1 \leq i \leq n(g)$. Thus the licensor features and licensee features corresponding to B_i will have two subscripts: $+b_{it}$, $-b_{it}$, $1 \leq t \leq d(B_i)$.

[7] By definition 4, $d_i(g) = d(B_i)$, $1 \leq i \leq n(g)$.

[8] By definition 4, if A = S, then r(g) = d(S) = 1.

$=\mathsf{c}_2^s \;\; f_{1l_1(g)} \ldots f_{11} \;\; \mathsf{c}_1^s \;\; -\mathsf{c}_1^s,$

where feature $f_{1l} \in \mathrm{F}_{r_s}^+$ is feature $+\mathsf{b}_{ij}$ if, and only if, variable $z_{1l} \in$ X is variable x_{ij}, $1 \leq l \leq \mathrm{l}_1(g)$, and

$=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)} \;\; \mathsf{c}_2^s.$

Note that in the lexical items for a non-terminating rule all features f_{hl} are defined for $1 \leq h \leq \mathrm{r}(g)$, $1 \leq l \leq \mathrm{l}_h(g)$: for any particular choice of h there will be a component g^h, whose definition will contain a z_{hl} for any legitimate choice of l.

For every rule r_s in R_e, Lex will contain the lexical items associated with it. Since the number of rules in R is finite and r(g) is bounded for any $g \in$ F, the number of items in Lex will be finite. Furthermore, since $\mathrm{l}_h(g)$ and n(g) are also bounded for any $g \in$ F, all items in Lex will consist of finite sequences of syntactic features.

3.4 Correspondence

Given an m-Multiple Context-Free Grammar G = (N, O, F, R, S) and a Minimalist Grammar G′ = (V, Cat, Lex, $\mathcal{F}$) constructed as per the instructions in section 3.3, we will distinguish a set of non-terminal trees in CL(G′) and define a relationship between these trees and pairs (A, p) $\in$ N $\times$ O where p is an A-phrase.

Definition 5. *A tree $\tau \in$ CL(G′) is a non-terminal tree if, and only if, the following three conditions are met:*

1. *The head of τ is labeled with a single syntactic feature* $\mathsf{a} \in base$ *corresponding to some non-terminal symbol A $\in$ N, and does not have any phonetic features.*

2. *Tree τ has k specifiers, $k > 0$. For each specifier τ_i^σ of τ, $1 \leq i \leq k$, the following holds:*

(a) *The head of τ_i^σ is labeled with the single syntactic feature* -$\mathsf{a}_i \in licensees$ *corresponding to the same A ∈ N as the syntactic feature labeling the head of τ, and does not have any phonetic features.*

(b) *In τ_i^σ, no node other than the head is labeled with any syntactic features.*

(c) *Specifier τ_i^σ precedes specifiers τ_j^σ, $1 < i < j \leq k$.*

3. *The complement τ^c of τ does not contain any nodes labeled with syntactic features or phonetic features.*

Definition 6. *A non-terminal tree $\tau \in$ CL(G′) corresponds to a pair (A, p) ∈ N × O for which p= (p_1, …, $p_{d(A)}$) ∈ L_G(A), if, and only if, the following conditions are met:*

1. *The head of τ is labeled with the feature* $\mathsf{a} \in base$ *corresponding to A.*

2. *Tree τ has d(A) specifiers. For each specifier τ_i^σ of τ, $1 \leq i \leq$ d(A), the following holds: the yield of τ_i^σ is p_i.*

3.5 Illustration of Proof of Equivalence

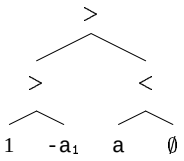
The proof that an m-Multiple Context-Free Grammar G = (N, O, F, R, S) and a Minimalist Grammar G′ = (V, Cat, Lex, $\mathcal{F}$) constructed according to the instructions section 3.3 are weakly equivalent involves showing that for every pair (A, p) ∈ N × O such that p is an A-phrase, there is a non-terminal tree $\tau \in$ CL(G′) corresponding to (A, p), and that for every non-terminal tree $\tau \in$ CL(G′), there is a pair (A, p) ∈ N × O, p an A-phrase, to which τ corresponds (propositions 1 and 2 in section 3.6.2). In this section we will illustrate these claims by means of a concrete example.

Example 3. Let G = ({A, B, C, D, E, F, H, S}, $\bigcup_{i=1}^{2}$ ({a, b, c, d}*)^i, {g, f, h}, {A → 1, B → 2, C → 3, D → 4, E → ϵ, F → ϵ, H → h[E, F], H → g[A, B, C, D, H], S

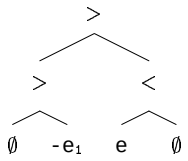
$\rightarrow f$[H]}, S), where $h[(x_{11}), (x_{21})] = (x_{11}, x_{21})$, $g[(x_{11}), (x_{21}), (x_{31}), (x_{41}), (x_{51}, x_{52})] = (x_{11}x_{51}x_{21}, x_{31}x_{52}x_{41})$, and $f[(x_{11}, x_{12})] = (x_{11}x_{12})$, be the grammar from example 2, which generates the language $\{1^n2^n3^n4^n \mid n \geq 0\}$, and let R_e an enumeration of the rules following the order given in the definition of G. Obviously, (1) $\in L_G$(A), (2) $\in L_G$(B), (3) $\in L_G$(C), (4) $\in L_G$(D), and (12, 34) $\in L_G$(H). Hence, by rule r_7 = H $\rightarrow$ g[A, B, C, D, H], g[(1), (2), (3), (4), (12, 34)] = (1122, 3344) $\in L_G$(H).

We will now specify Minimalist Grammar G′ = (Cat, V, Lex, $\mathcal{F}$) for G by giving the contents of Lex. For terminating rule r_0, Lex will contain the tree given in 1. Lex will include similar trees for rules r_1, r_2, and r_3. For terminating rule r_4, Lex will contain the tree given in 2. There is a similar tree in Lex for rule r_5.

1.



2.



For non-terminating rule r_6, Lex will contain the following lexical items:

3. =h_1^6 +h_2^6 +h_1^6 h

4. =h_2^6 +e$_1$ h_1^6 -h_1^6 -h$_1$

5. =h_3^6 +f$_1$ h_2^6 -h_2^6 -h$_2$

6. =e =f h_3^6

For non-terminating rule r_7, Lex will contain the following lexical items:

7. =h_1^7 +h_2^7 +h_1^7 h

8. =h_2^7 +b$_1$ +h$_1$ +a$_1$ h_1^7 -h_1^7 -h$_1$

9. =h_3^7 +d$_1$ +h$_2$ +c$_1$ h_2^7 -h_2^7 -h$_2$

10. =a =b =c =d =h h_3^7

For non-terminating rule r_8, Lex will contain the following lexical items:

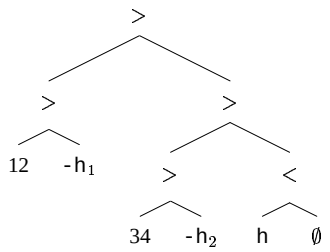
11. =c_1^8 +c_1^8 c

12. =c_2^8 +h$_2$ +h$_1$ c_1^8 -c_1^8 -c$_1$

13. $=c_2^8\ +h_2\ +h_1\ c_1^8\ -c_1^8$ 14. $=h\ c_3^8$

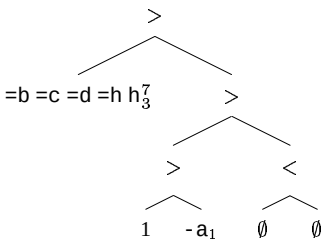
It follows from proposition 1 that CL(G′) will contain non-terminal trees corresponding to the pairs (A, (1)), (B, (2)), (C, (3)), (D, (4)), and (H, (12, 34)). The first four trees are elements of Lex; the tree corresponding to (H, (12, 34)) is the derived tree in 15.[9]

15.



We will now show how the non-terminal tree corresponding to the pair (H, (1122, 3344)) is derived from these smaller non-terminal trees and the lexical items in 7 through 10. First, item $=a\ =b\ =c\ =d\ =h\ h_3^7$ will merge with tree 1, because the former has a feature $=a \in select$ and the latter has a feature $a \in base$. The result is the tree in 16.

16.

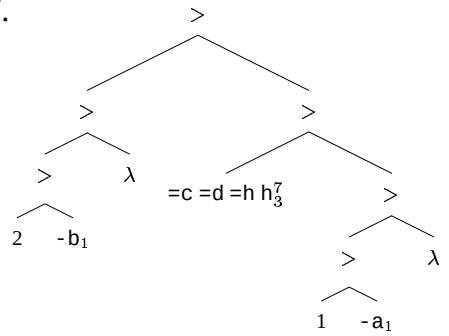


The derivation will continue with merging tree 16 and the lexical item with head b corresponding to the pair (B, (2)). Since tree 16 is a complex tree, the item it merges with will be attached as a specifier, yielding the tree in 17.[10]

[9] The specifiers of the specifiers of this tree contain nodes with empty labels that are irrelevant and hence not shown.

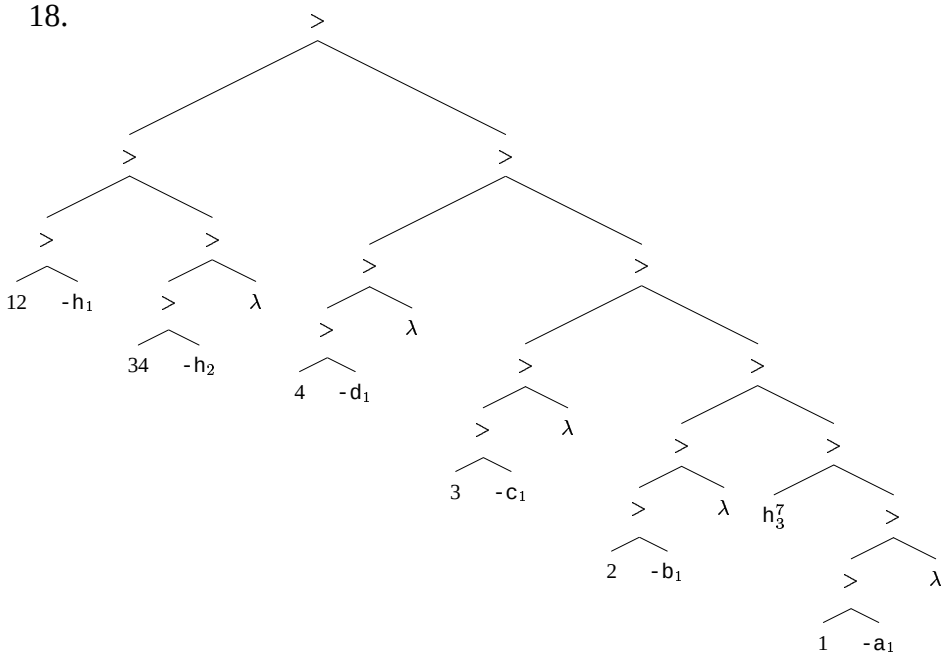
[10] In order to keep the trees manageable size-wise, λ is used to abbreviate a complex subtree whose leaves are all labeled $\emptyset$.

17.



The applications of $merge$ triggered by the features =c, =d, and =h occurring in the head of tree 17 will eventually produce the tree in 18. This tree contains the arguments (1), (2), (3), (4), (12, 34) to which g will be applied to obtain the value (1122, 3344) $\in$ L_G(H).

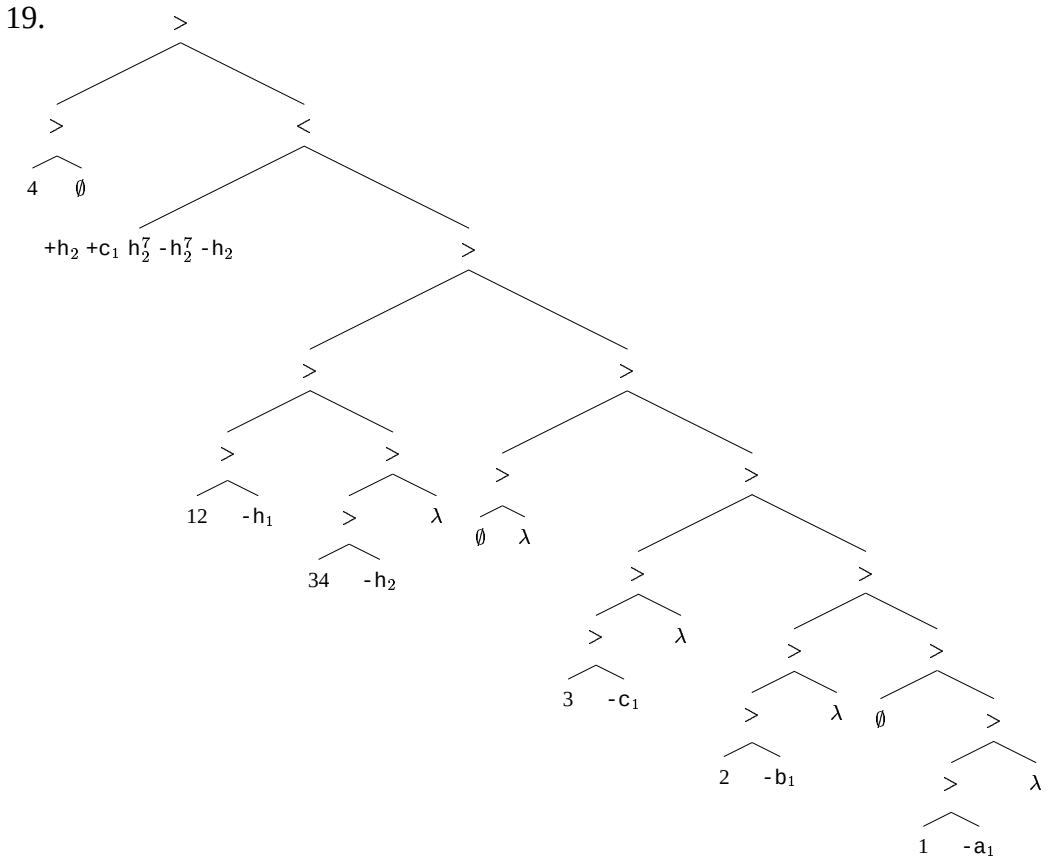
18.



Next, item =h_3^7 +d_1 +h_2 +c_1 h_2^7 -h_2^7 -h_2 will select tree 18. The resulting tree is [$_<$+d_1 +h_2 +c_1 h_2^7 -h_2^7 -h_2, τ], where τ is like tree 18, except that the feature h_3^7 of the head is deleted. The left-most feature of the head of [$_<$+d_1 +h_2 +c_1 h_2^7 -h_2^7 -h_2, τ] is +d_1 $\in$ $licensors$. Subtree τ contains one, and only one node whose left-most

feature is $\texttt{-d}_1 \in licensees$. This pair of features triggers an application of $move$. The tree in 19 is the result.

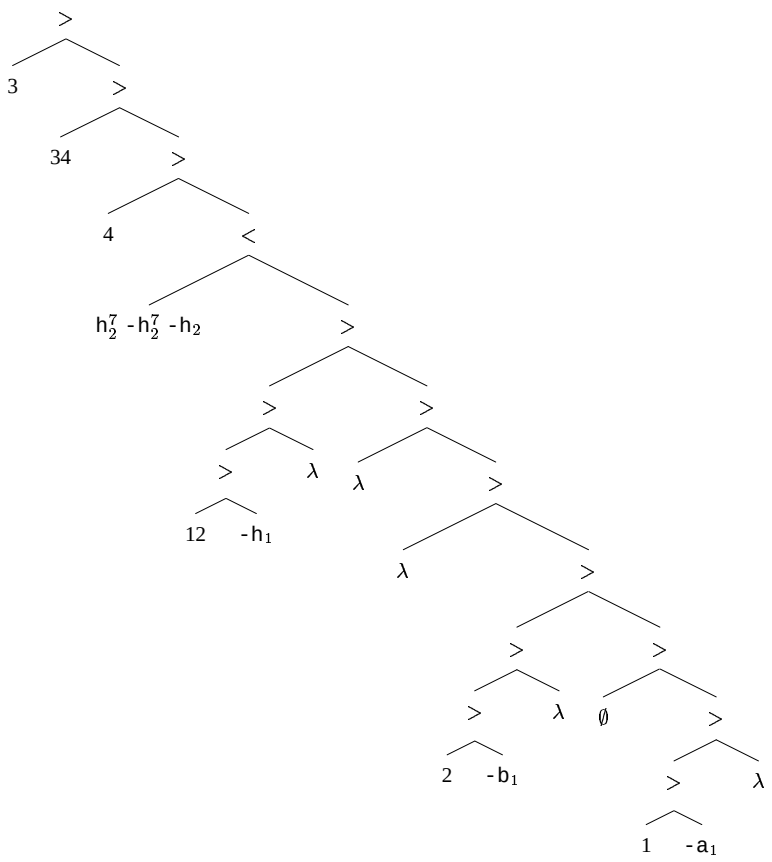
19.



After two more applications of $move$ involving the features $\texttt{+h}_2$ and $\texttt{+c}_1$, the tree in 20 is obtained.[11] The three $move$ operations have created a sequence of specifiers whose yields correspond to the value of the second component of g[(1), (2), (3), (4), (12, 34)] = (1122, 3344).

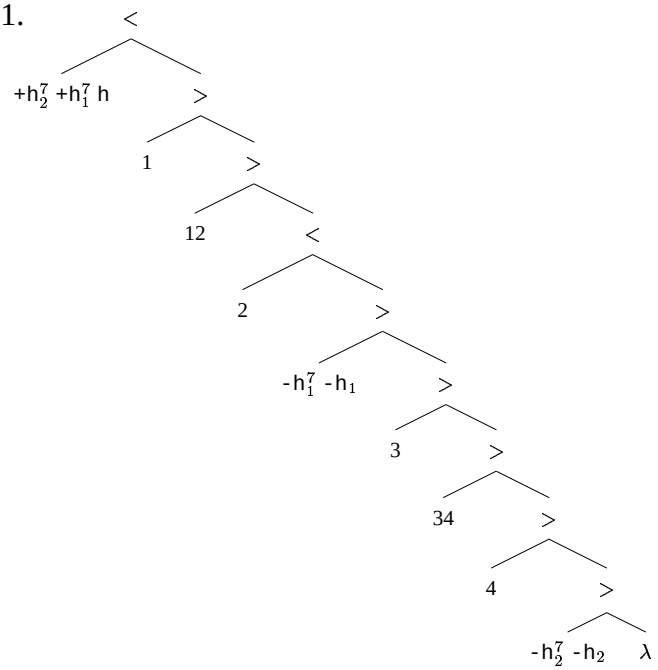
[11] To save space, two-node subtrees of the form $[_{<} x, \emptyset]$ and $[_{>} x, \emptyset]$, $x \in \text{Cat}^*\text{P}^*\text{I}^*$, are simply written as x.

20.



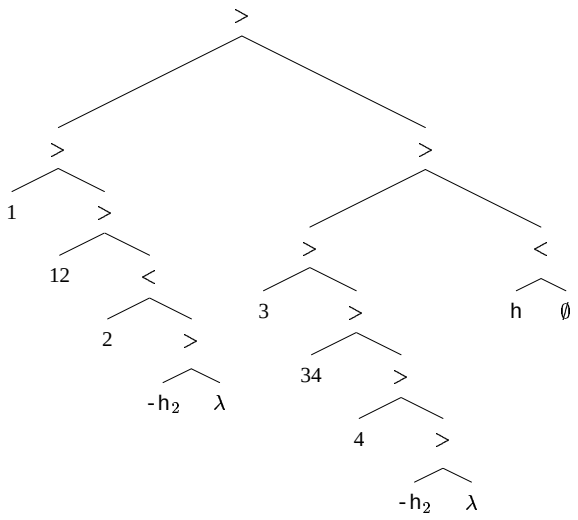
The tree in 20 will be selected by item $=h_2^7\ +b_1\ +h_1\ +a_1\ h_1^7\ -h_1^7\ -h_1$, which will take care of assembling the string corresponding to the first component of g[(1), (2), (3), (4), (12, 34)]. The tree resulting from these $merge$ and $move$ actions will then be selected by item $=h_1^7\ +h_2^7\ +h_1^7\ \ h$, as in 21.

21.



Finally, two applications of $move$ triggered by the features +h$_2^7$, -h$_2^7$ and +h$_1^7$, -h$_1^7$ in tree 21 will produce the non-terminal tree in 22. This tree is a non-terminal tree corresponding to the pair (H, (1122, 3344)).

22.



3.6 Proof of Equivalence

Let G = (N, O, F, R, S) be an m-Multiple Context-Free Grammar in normal form, R_e an enumeration of R, and G′ = (V, Cat, Lex, $\mathcal{F}$) a Minimalist Grammar constructed as specified in section 3.3. In section 3.6.2 we will show that L(G) = L(G′). First, we will prove several properties of G′.

3.6.1 Properties of Constructed Minimalist Grammar

Consider a non-terminating rule r_s = A $\rightarrow g[B_1, \ldots, B_{n(g)}]$ in R_e and the lexical items associated with rule r_s. Assume that the components of g are represented as in section 3.3.4, and let X = $\{x_{ij} \mid 1 \leq i \leq n(g), 1 \leq j \leq d_i(g)\}$ and $F^+_{r_s}$ = $\{+b_{ij} \mid 1 \leq i \leq n(g), 1 \leq j \leq d(B_i)\}$ as in section 3.3.4.

Theorem 1. *For every feature* $+b_{ij} \in F^+_{r_s}$*, there is at least one pair* (h, l)*,* $1 \leq h \leq r(g)$*,* $1 \leq l \leq l_h(g)$*, such that feature* f_{hl} *is* $+b_{ij}$.

Proof. Pick an arbitrary feature $+b_{ij} \in F^+_{r_s}$. Because of the existence of rule r_s in R_e, $d_k(g) = d(B_k)$, $1 \leq k \leq n(g)$. Therefore, there is a variable $x_{ij} \in X$. It follows from the non-erasure condition (lemma 1, part (b)) that there must be a variable z_{hl}, $1 \leq h \leq r(g)$, $1 \leq l \leq l_h(g)$, which is identical to the variable x_{ij}. Then, by the definition of the lexical items associated with rule r_s, feature f_{hl} will be the feature $+b_{ij}$. □

Theorem 2. *For every feature* $+b_{ij} \in F^+_{r_s}$*, there is at most one pair* (h, l)*,* $1 \leq h \leq r(g)$*,* $1 \leq l \leq l_h(g)$*, such that feature* f_{hl} *is* $+b_{ij}$.

Proof. Suppose that there are pairs (h, l) and (p, q), $(h, l) \neq (p, q)$, such that features f_{hl} and f_{pq} are the same feature $+b_{ij} \in F^+_{r_s}$, $1 \leq h \leq r(g)$, $1 \leq l \leq l_h(g)$, $1 \leq p \leq r(g)$, $1 \leq q \leq l_p(g)$. Then, by the way the items in Lex are defined, there are variables z_{hl} and z_{pq} in the representation of the components of g, such that both z_{hl} and z_{pq} are the same variable $x_{ij} \in X$. This, however, is ruled out by the anti-copying condition

(definition 4, part 2.(c)). □

Theorem 3. *If $\beta_1, \ldots, \beta_{n(g)}$ are non-terminal trees in CL(G′) that have features* $\mathsf{b}_1, \ldots, \mathsf{b}_{n(g)}$ *respectively, then the projection of lexical item* $\mathsf{=a}^s_1\ \mathsf{+a}^s_{r(g)} \ldots \mathsf{+a}^s_1\ \mathsf{a}$, *using lexical items* $\mathsf{=a}^s_{h+1}\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}^s_h\ \mathsf{-a}^s_h\ \mathsf{-a}_h$, $1 \leq h \leq r(g)$, $\mathsf{=b}_1 \ldots \mathsf{=b}_{n(g)}\ \mathsf{a}^s_{r(g)+1}$, *and trees $\beta_1, \ldots, \beta_{n(g)}$, is a non-terminal tree of category* a.

Proof. Item $\mathsf{=b}_1 \ldots \mathsf{=b}_{n(g)}\ \mathsf{a}^s_{r(g)+1}$ will select the trees $\beta_1, \ldots, \beta_{n(g)}$. Because the trees β_i, $1 \leq i \leq$ n(g), are non-terminal trees, the resulting tree ψ will have $\sum_{i=1}^{n(g)}$ d(B_i) licensee features. Thanks to lemma 1, part (d), these features are all different. Consequently, the derivation will not crash because ψ will have more than one maximal subtree that has some feature $\mathsf{-y} \in licensees$, cf. definition 2, part 4.(b). Let $\mathrm{F}^-_{r_s} = \{\mathsf{-b}_{ij} \mid 1 \leq i \leq \mathrm{n}(g), 1 \leq j \leq \mathrm{d}(\mathrm{B}_k)\}$ be the set of all licensee features occurring in the trees β_i, $1 \leq i \leq$ n(g).

For $1 \leq h \leq$ r(g), each item $\mathsf{=a}^s_{h+1}\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}^s_h\ \mathsf{-a}^s_h\ \mathsf{-a}_h$ will select the tree projected by its successor, $\mathsf{=a}^s_{h+2}\ f_{h+1l_{h+1}(g)} \ldots f_{h+11}\ \mathsf{a}^s_{h+1}\ \mathsf{-a}^s_{h+1}\ \mathsf{-a}_{h+1}$. The last item selected in this chain will select ψ. For these *merge* operations to succeed, all features $f_{hl_h(g)} \ldots f_{h1}$ in all items $\mathsf{=a}^s_{h+1}\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}^s_h\ \mathsf{-a}^s_h\ \mathsf{-a}_h$, $1 \leq h \leq$ r(g), should be deleted by an application of $move$. Hence, for each licensor feature f_{hl}, $1 \leq h \leq$ r(g), $1 \leq l <$ $\mathrm{l}_h(g)$, there has to be a matching licensee feature in $\mathrm{F}^-_{r_s}$. Pick an arbitrary pair (h, l), $1 \leq h \leq$ r(g), $1 \leq l <$ $\mathrm{l}_h(g)$ and assume f_{hl} is the feature $\mathsf{+b}_{ij} \in \mathrm{F}^+_{r_s}$. It follows immediately from the definition of the set $\mathrm{F}^-_{r_s}$ that there is a matching licensee feature $\mathsf{-b}_{ij} \in \mathrm{F}^-_{r_s}$. By theorem 2, there is no other feature f_{pq}, $1 \leq p \leq$ r(g), $1 \leq q \leq \mathrm{l}_p$(g), ($h$, l) $\neq$ (p, q), such that f_{pq} also is feature $\mathsf{+b}_{ij}$. Therefore, feature $\mathsf{-b}_{ij} \in \mathrm{F}^-_{r_s}$ will be available to check and delete feature f_{hl}.

Let ϕ be the tree projected from item $\mathsf{=a}^s_2\ f_{1l_1(g)} \ldots f_{11}\ \mathsf{a}^s_1\ \mathsf{-a}^s_1\ \mathsf{-a}_1$ through the $merge$ and $move$ operations described above. The head of ϕ is labeled $\mathsf{a}^s_1\ \mathsf{-a}^s_1\ \mathsf{-a}_1$. The only other syntactic features present in ϕ will be sequences $\mathsf{-a}^s_h\ \mathsf{-a}_h$, $2 \leq h \leq$

r(g), from projecting the items $=\mathsf{a}^s_{h+1}\ f_{hl_h(g)}\ldots f_{h1}\ \mathsf{a}^s_h\ \text{-}\mathsf{a}^s_h\ \text{-}\mathsf{a}_h$, $2 \leq h \leq$ r(g). All other features of these items have been deleted during the derivation of ϕ, and so have all the features of item $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{a}^s_{r(g)+1}$, and all the features contributed by the trees β_i, $1 \leq i \leq$ n(g), selected by this item: pick an arbitrary licensee feature $\text{-}\mathsf{b}_{ij} \in \mathrm{F}^-_{r_s}$. It follows immediately from the definition of the set $\mathrm{F}^+_{r_s}$ that there is a matching licensor feature $+\mathsf{b}_{ij} \in \mathrm{F}^+_{r_s}$. Then, by theorem 1, there will be a pair (h, l), $1 \leq h \leq$ r(g), $1 \leq l \leq$ l$_h$(g), such that feature f_{hl} is $+\mathsf{b}_{ij}$. Since feature $\text{-}\mathsf{b}_{ij}$ does not occur more than once in the set of trees β_i, $1 \leq i \leq$ n(g), feature $f_{hl} \in \mathrm{F}^+_{r_s}$ will be available to check and delete feature $\text{-}\mathsf{b}_{ij}$.

Item $=\mathsf{a}^s_1\ +\mathsf{a}^i_{r(g)} \ldots\ +\mathsf{a}^s_1\ \mathsf{a}$ will select ϕ. The head of the resulting tree υ is labeled $+\mathsf{a}^i_{r(g)} \ldots\ +\mathsf{a}^s_1\ \mathsf{a}$. The complement of υ will contain sequences of syntactic features $\text{-}\mathsf{a}^i_h\ \text{-}\mathsf{a}_h$, $1 \leq h \leq$ r(g), and no other syntactic features. Hence, $move$ will apply r(g) times to υ to derive another tree τ. It is easy to check that τ meets the conditions on non-terminal trees given in definition 5. □

Theorem 4. *If $\beta_1, \ldots, \beta_{n(g)}$ are non-terminal trees in CL(G′) that have features* $\mathsf{b}_1, \ldots, \mathsf{b}_{n(g)}$ *respectively, then the projection of lexical item* $=\mathsf{c}^s_1\ +\mathsf{c}^s_1\ \mathsf{c}$, *using lexical items* $=\mathsf{c}^s_2\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}^s_1\ \text{-}\mathsf{c}^s_1$, $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{c}^s_2$, *and trees $\beta_1, \ldots, \beta_{n(g)}$, is a complete tree of category* c *in CL(G′).*

Proof. The proof is similar to the proof of theorem 3 for $\mathsf{a} = \mathsf{c}$; it uses lexical item $=\mathsf{c}^s_2\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}^s_1\ \text{-}\mathsf{c}^s_1$ rather than lexical item $=\mathsf{c}^s_2\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}^s_1\ \text{-}\mathsf{c}^s_1\ \text{-}\mathsf{c}_1$. □

Theorem 5. *If $\beta_1, \ldots, \beta_{n(g)}$ are non-terminal trees and complete trees in CL(G′) that have features* $\mathsf{b}_1, \ldots, \mathsf{b}_{n(g)}$ *respectively, and at least one of these trees is a complete tree, then no tree of category* a *can be projected from lexical item* $=\mathsf{a}^i_1\ +\mathsf{a}^i_{r(g)} \ldots +\mathsf{a}^i_1\ \mathsf{a}$, *using lexical items* $=\mathsf{a}^i_{h+1}\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}^i_h\ \text{-}\mathsf{a}^i_h\ \text{-}\mathsf{a}_h$, $1 \leq h \leq$ *r(g)*, $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{a}^i_{r(g)+1}$, *and trees $\beta_1, \ldots, \beta_{n(g)}$.*

Proof. Assume tree β_k is a complete tree, $1 \leq$ k $\leq$ n(g). By theorem 1, there is a

licensor feature f_{hl}, $1 \leq h \leq$ r(g), $1 \leq$ l $\leq l_h(g)$, such that f_{hl} is feature $+\mathsf{b}_{kj}$, $1 \leq j \leq$ d(B_k). However, tree β_k, being a complete tree, does not contain a matching licensee feature $-\mathsf{b}_{kj}$. None of the other trees $\beta_1, \ldots, \beta_{n(g)}$ contains a matching feature either, because they are complete trees, or, if they are non-terminal trees, they will have different sets of licensee features as a result of lemma 1, part (d). Hence feature f_{hl} cannot be checked and deleted and the derivation cannot be completed. □

Theorem 6. *If $\beta_1, \ldots, \beta_{n(g)}$ are trees in CL(G′) that have features $\mathsf{b}_1, \ldots, \mathsf{b}_{n(g)}$ respectively, and tree β_i is a complete tree, $1 \leq i \leq$ n(g), and the other trees are either non-terminal trees or complete trees, then no tree of category* c *can be projected from lexical item $=\mathsf{c}_1^s\ +\mathsf{c}_1^s\ \mathsf{c}$, using lexical items $=\mathsf{c}_2^s\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}_1^s\ -\mathsf{c}_1^s$, $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{c}_2^s$, and trees $\beta_1, \ldots, \beta_{n(g)}$.*

Proof. The proof is analogous to the proof of theorem 5. □

Theorem 7. *If a tree $\tau \in$ CL(G′) has a feature $\mathsf{a} \in base$, then τ is a non-terminal tree or a complete tree of category* a.

Proof. The proof is by induction over k in CL^k(G′). For $\tau \in \mathrm{CL}^0$(G′) = Lex, the claim follows immediately from the definition of Lex. Let τ be an arbitrary tree in CL^k(G′), $k > 0$. If $\tau \in \mathrm{CL}^{k-1}$(G′), the claim follows from the induction hypothesis. If $\tau \notin \mathrm{CL}^{k-1}$(G′), then τ is a projection of an item $=\mathsf{a}_1^s\ +\mathsf{a}_{r(g)}^s \ldots \mathsf{a}_1^s\ \mathsf{a}$ in Lex, $1 \leq s \leq |\mathrm{R}|$, involving lexical items $=\mathsf{a}_{h+1}^s\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}_h^s\ -\mathsf{a}_h^s\ -\mathsf{a}_h$, $1 \leq h \leq$ r(g), and $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{a}_{r(g)+1}^s$, or involving items $=\mathsf{c}_2^s\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}_1^s\ -\mathsf{c}_1^s$, and $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{c}_2^s$. In the first case, assume that $\beta_1, \ldots, \beta_{n(g)}$ are the trees selected by item $=\mathsf{b}_1 \ldots =\mathsf{b}_{n(g)}\ \mathsf{a}_{r(g)+1}^s$. These trees will have features $\mathsf{b}_1 \ldots \mathsf{b}_{n(g)}$ respectively. Since the trees $\beta_1, \ldots, \beta_{n(g)}$ are closer to the lexicon than τ, the induction hypothesis applies: each tree β_i, $1 \leq i \leq$ n(g), is either a non-terminal tree, or a complete tree. However, by theorem 5, none of the trees $\beta_1, \ldots, \beta_{n(g)}$ can be a complete tree. Hence, they must be non-terminal trees, whence, by theorem 3, tree τ is a non-terminal tree, too. If the projection involves item

$=\mathsf{c}_2^s\ f_{1l_h(g)} \ldots f_{11}\ \mathsf{c}_1^s\ \text{-}\mathsf{c}_1^s$, tree τ is a complete tree. The argument is similar to the first case, and depends on theorems 6 and 4. □

Theorem 8. *There is a non-terminal tree $\tau \in CL^k(G')$ whose head is labeled with feature $\mathsf{c} \in base$ if, and only if, there is a complete tree $\tau' \in CL^k(G')$ which is like τ except for the feature $\text{-}\mathsf{c}_1$.*

Proof. This follows from the co-existence in Lex of the items $[_>[_> g,\ \text{-}\mathsf{c}_1], [_<\mathsf{c}, \emptyset]]$ and $[_> g, [_<\mathsf{c}, \emptyset]]$ for any terminating rule S $\rightarrow g$ in R, and the items $=\mathsf{c}_2^i\ f_{1l_1(g)} \ldots f_{11}\ \mathsf{c}_1^i\ \text{-}\mathsf{c}_1^i\ \ \text{-}\mathsf{c}_1$ and $=\mathsf{c}_2^i\ f_{1l_1(g)} \ldots f_{11}\ \mathsf{c}_1^i\ \text{-}\mathsf{c}_1^i$ for any non-terminating rule r_s = S $\rightarrow g$[B_1, …, $\mathrm{B}_{n(g)}$] in R. □

3.6.2 Equivalence

Proposition 1. *For every pair (A, p) $\in N \times O$ such that $p \in L_G$(A), there is a non-terminal tree $\tau \in CL(G')$ which corresponds to (A, p).*

Proof. We will show by induction on $k \in \mathbb{N}$ that the following claim holds for all A $\in$ N: if $p \in \mathrm{L}_G^k$(A), then there is a non-terminal tree $\tau \in$ CL(G′) such that τ corresponds to the pair (A, p). If k = 0, then $p \in \mathrm{L}_G^0$(A). Hence, R contains a terminating rule A $\rightarrow p$. Then, by the construction of Lex, there is an item $[_>[_> p,\ \text{-}\mathsf{a}_1], [_<\mathsf{a}, \emptyset]]$ in Lex. This is a non-terminal tree which corresponds to the pair (A, p).

For the induction step, pick an arbitrary pair (A, p) such that $p \in \mathrm{L}_G^k$(A), $k > 0$. If $p \in \mathrm{L}_G^{k-1}$(A), the claim follows from the induction hypothesis. If $p \notin \mathrm{L}_G^{k-1}$(A), then there is a non-terminating rule r_s = A $\rightarrow g$[B_1, …, $\mathrm{B}_{n(g)}$] in R_e, such that θ_i = (θ_{i1}, …, $\theta_{id(B_i)}$) $\in \mathrm{L}_G^{k-1}(\mathrm{B}_i)$, $1 \leq i \leq$ n(g), and $p = g(\theta_1, \ldots, \theta_{n(g)})$. Since $\theta_i \in \mathrm{L}_G^{k-1}(\mathrm{B}_i)$, the induction hypothesis applies: for every pair (B_i, θ_i) there is a non-terminal tree β_i $\in$ CL(G′) such that β_i corresponds to (B_i, θ_i), $1 \leq i \leq$ n(g).

Because R includes rule r_s, Lex contains the items $=\mathsf{a}_1^s\ +\mathsf{a}_{r(g)}^s \ldots +\mathsf{a}_1^s\ \ \mathsf{a}$, $=\mathsf{a}_{h+1}^s$

$f_{hl_h(g)} \ldots f_{h1}$ a^s_h $\mathsf{-a}^s_h$ $\mathsf{-a}_h$, $1 \leq h \leq$ r(g), and $\mathsf{=b}_1 \ldots \mathsf{=b}_{n(g)}$ $\mathsf{a}^s_{r(g)+1}$. It follows from theorem 3 that the projection of item $\mathsf{=a}^s_1$ $\mathsf{+a}^s_{r(g)} \ldots \mathsf{+a}^s_1$ a, using the other items and the trees β_i, $1 \leq i \leq$ n(g), is a non-terminal tree. Let τ be this tree.

The head of τ is labeled a, which corresponds to A. Since R contains the rule r_s, r(g) = d(A). Hence, τ, being a projection of $\mathsf{=a}^s_1$ $\mathsf{+a}^s_{r(g)} \ldots \mathsf{+a}^s_1$ a, has d(A) specifiers. Consider an arbitrary specifier τ^σ_h of τ, $1 \leq h \leq$ d(A). This specifier is the result of an application of *move* triggered by the licensor feature $\mathsf{+a}^s_h$ and the matching licensee feature $\mathsf{-a}^s_h$ in item $\mathsf{=a}^s_{h+1}$ $f_{hl_h(g)} \ldots f_{h1}$ a^s_h $\mathsf{-a}^s_h$ $\mathsf{-a}_h$. Thus, specifier τ^σ_h is a projection of this item. Specifier τ^σ_h does not contain the complement selected by the feature $\mathsf{=a}^s_{h+1}$, because this complement has moved to become specifier τ^σ_{h+1} before specifier τ^σ_h was created. Deleting features $f_{hl_h(g)} \ldots f_{h1}$ has equipped specifier τ^σ_h with $l_h(g)$ specifiers of its own.

Let τ^σ_{hl} be the specifier of τ^σ_h created by the application of $move$ triggered by the licensor feature f_{hl}, $1 \leq l \leq l_h(g)$. Assume that in the representation of the h^{th} component of g, variable z_{hl} is the variable $x_{ij} \in$ X. Then, by the way the contents of Lex are defined, feature f_{hl} is the licensor feature $\mathsf{+b}_{ij}$. The matching licensee feature $\mathsf{-b}_{ij}$ is found in tree β_i, whose head is labeled b_i. Thus, the yield of specifier τ^σ_{hl} is the yield of the maximal projection in tree β_i of the node labeled $\mathsf{-b}_{ij}$, that is, specifier β^σ_{ij} of β_i. By the induction hypothesis, Y(β_i) = $\theta_{i1} \ldots \theta_{id(B_i)}$, whence Y($\tau^\sigma_{hl}$) = θ_{ij}.

To determine the value of the function g applied to the arguments (θ_1, …, $\theta_{n(g)}$), any occurrence of a variable x_{pq} in the representation of the components of g will be replaced by the value θ_{pq}, $1 \leq p \leq$ n(g), $1 \leq q \leq$ $d_i(g)$ = d(B_i). Let ρ be this set of replacements.[12] Since by assumption the particular variable z_{hl} is the variable x_{ij}, it follows that $z_{hl}/\theta_{ij} \in \rho$. Hence, Y($\tau^\sigma_{hl}$) = θ_{ij} = $z_{hl}[\rho]$. Because l is arbitrary and Y(τ^σ_h) = Y(τ^σ_{h1})… Y($\tau^\sigma_{hl_h(g)}$), it follows that Y(τ^σ_h) = ($z_{h1} \ldots z_{hl_h(g)}$)[$\rho$]. Now ($z_{h1} \ldots z_{hl_h(g)}$)[$\rho$]

[12] A replacement will be notated as a pair x/y; $z[\{x/y\}] = y$ if $z = x$, $z[\{x/y\}] = z$ otherwise.

is the value of g^h applied to the arguments ($\theta_1, \ldots, \theta_{n(g)}$). Hence, Y($\tau_h^\sigma$) = $g^h(\theta_1, \ldots, \theta_{n(g)})$. Because h is arbitrary and $p = g(\theta_1, \ldots, \theta_{n(g)}) = (g^1(\theta_1, \ldots, \theta_{n(g)}), \ldots, g^{r(g)}(\theta_1, \ldots, \theta_{n(g)}))$, it follows that Y($\tau_i^\sigma$) = p_i, $1 \leq i \leq$ d(A) = r(g). Thus τ corresponds to (A, p). □

Proposition 2. *For every non-terminal tree $\tau \in CL(G')$, there is a pair (A, p), $p \in L_G(A)$, such that τ corresponds to (A, p).*

Proof. We will show by induction on $k \in \mathbb{N}$ that the following claim holds: if $\tau \in$ CL^k(G′), τ a non-terminal tree, then there is a pair (A, p), $p \in L_G$(A), such that τ corresponds to (A, p). For k = 0, CL^k(G′) = Lex. It follows from the construction of Lex that any non-terminal tree $\tau \in$ Lex is of the form $[_>[_>p,$ -a$_1]$, $[_<$a, $\emptyset]]$. The presence of τ in Lex implies that R contains the terminating rule A $\rightarrow p$. Hence, $p \in L_G$(A), and τ corresponds to the pair (A, p).

For the induction step, pick an arbitrary non-terminal tree $\tau \in$ CL^k(G′), $k > 0$, with a head labeled a and yield $s = s_1 \ldots s_m$, s_i being the yield of specifier τ_i^σ of τ, $1 \leq i \leq m$. If $\tau \in$ CL^{k-1}(G′), the claim follows from the induction hypothesis. If $\tau \notin$ CL^{k-1}(G′), then τ is a projection of an item =a$_1^s$ +a$_{r(g)}^s \ldots$ a$_1^s$ a in Lex, $1 \leq s \leq |$R$|$. Hence, R contains a non-terminating rule r_s = A $\rightarrow g$[B$_1$, …, B$_{n(g)}$]. The derivation of τ also involves the items =a$_{h+1}^s$ $f_{hl_h(g)} \ldots f_{h1}$ a$_h^s$ -a$_h^s$ -a$_h$, $1 \leq h \leq$ r(g), and =b$_1 \ldots$ =b$_{n(g)}$ a$_{r(g)+1}^s$. Assume $\beta_1, \ldots, \beta_{n(g)}$ are the trees selected by =b$_1 \ldots$ =b$_{n(g)}$ a$_{r(g)+1}^s$. By theorems 3, 5 and 7, these trees are non-terminal trees. Since $\beta_1, \ldots, \beta_{n(g)}$ are closer to the lexicon than τ, the induction hypothesis applies: for each tree β_i there is a pair (B$_i$, p_i), $p_i = (p_{i1}, \ldots, p_{id(B_i)}) \in L_G$(B$_i$) such that β_i corresponds to (B$_i$, p_i), $1 \leq i \leq$ n(g). Hence, p_i = (Y(β_{i1}^σ), …, Y($\beta_{id(B_i)}^\sigma$)), $1 \leq i \leq$ n(g). Because of rule r_s, $g(p_1, \ldots, p_{n(g)}) \in L_G$(A).

Since τ is a projection of item =a$_1^s$ +a$_{r(g)}^s \ldots$ +a$_1^s$ a and R contains rule r_s, m = r(g) = d(A), that is, τ has d(A) specifiers. An arbitrary specifier τ_h^σ of τ, $1 \leq h \leq$ r(g), is the

result of an application of *move* triggered by the licensor feature $+\mathsf{a}^s_h$ and the matching licensee feature $-\mathsf{a}^s_h$ in item $=\mathsf{a}^s_{h+1}\ f_{hl_h(g)} \ldots f_{h1}\ \mathsf{a}^s_h\ -\mathsf{a}^s_h\ -\mathsf{a}_h$. Thus, specifier τ^σ_h is a projection of this item. Specifier τ^σ_h does not contain the complement selected by the feature $=\mathsf{a}^s_{h+1}$, because this complement has moved to become specifier τ^σ_{h+1} before specifier τ^σ_h was created. Deleting features $f_{hl_h(g)} \ldots f_{h1}$ has equipped specifier τ^σ_h with $\mathrm{l}_h(g)$ specifiers of its own. Let τ^σ_{hl} denote the specifier of τ^σ_h arising from the *move* operation triggered by the licensee feature f_{hl}, $1 \leq l \leq \mathrm{l}_h(g)$. By assumption, $\mathrm{Y}(\tau^\sigma_h) = s_h$. Since no phonetic features are found outside specifiers and τ^σ_h has an empty complement, $\mathrm{Y}(\tau^\sigma_h) = s_h = s_{h1} \ldots s_{hl_h(g)}$, where $\mathrm{Y}(\tau^\sigma_{hl}) = \mathrm{s}_{hl}$, $1 \leq l \leq \mathrm{l}_h(g)$. Assume that the licensor feature f_{hl} involved in the creation of specifier τ^σ_{hl} is the feature $+\mathsf{b}_{ij} \in \mathrm{F}^+_{r_s}$. Hence, the matching licensee feature $-\mathsf{b}_{ij}$ is found in tree β_i, the head of which is labeled b_i. Then the yield of the maximal projection in β_i of the node labeled $-\mathsf{b}_{ij}$ is identical to the yield of specifier τ^σ_{hl}. By assumption, $\mathrm{Y}(\tau^\sigma_{hl}) = s_{hl}$, whence $\mathrm{Y}(\beta^\sigma_{ij}) = s_{hl}$.

To determine the value of the function g applied to the arguments $(p_1, \ldots, p_{n(g)})$, any occurrence of a variable x_{pq} in the representation of the components of g will be replaced by the value $p_{pq} = \mathrm{Y}(\beta^\sigma_{pq})$, $1 \leq p \leq \mathrm{n}(g)$, $1 \leq q \leq \mathrm{d}_i(g) = \mathrm{d}(\mathrm{B}_i)$. Let ρ be this set of replacements. Since by assumption the particular feature f_{hl} is the feature $+\mathsf{b}_{ij}$, it follows from the specification of Lex that the variable z_{hl} is the variable x_{ij}. Hence, ρ contains the replacement $z_{hl}/\mathrm{Y}(\beta^\sigma_{ij})$. Since $\mathrm{Y}(\beta^\sigma_{ij}) = s_{hl}$, this replacement is identical to the replacement z_{hl}/s_{hl}. Since l is arbitrary, $g^h(p_1, \ldots, p_{n(g)}) = z_{h1} \ldots z_{hl_h(g)}[\rho] = s_{h1} \ldots s_{hl_h(g)}$. Since h is arbitrary, $g(p_1, \ldots, p_{n(g)}) = (g^1(p_1, \ldots, p_{n(g)}), \ldots, g^{r(g)}(p_1, \ldots, p_{n(g)})) = (s_{11} \ldots s_{1l_1(g)}, \ldots, s_{r(g)1} \ldots s_{r(g)l_{r(g)}(g)}) = (\mathrm{Y}(\tau^\sigma_1), \ldots, \mathrm{Y}(\tau^\sigma_{r(g)})) \in \mathrm{L}_G(\mathrm{A})$. Hence, τ corresponds to $(\mathrm{A}, (\mathrm{Y}(\tau^\sigma_1), \ldots, \mathrm{Y}(\tau^\sigma_{r(g)}))$. □

Corollary 1. *L(G) = L(G′).*

Proof. Assume $p \in \mathrm{L}(\mathrm{G}')$. Then there is a complete tree $\tau \in \mathrm{CL}(\mathrm{G}')$ with yield p. Since

τ is a complete tree, its head is labeled c. By theorem 8, there is a non-terminal tree $\tau' \in$ CL(G′) which is like τ, but has a licensee feature -c_1. It follows from proposition 2 that there is a pair (A, p) $\in$ N $\times$ O such that $p \in L_G$(A) and τ' corresponds to (A, p). Since the head of τ' is labeled c, A = S. Hence, $p \in$ L(G).

Assume $p \in$ L(G). Then there is a pair (S, p) $\in$ N $\times$ O and $p \in L_G$(S). It follows from proposition 1 that there is a non-terminal tree $\tau \in$ CL(G′) which corresponds to the pair (S, p). Hence, the head of τ is labeled c and Y(τ) = p.[13] By theorem 8, there is a complete tree $\tau' \in$ CL(G′) which is like τ, but lacks the licensee feature -c_1. Since τ' is a complete tree, Y(τ') = Y(τ) = $p \in$ L(G′). □

3.7 Conclusion

In this chapter we have shown how to construct a Minimalist Grammar G′ for an arbitrary Multiple-Context Free Grammar G such that the language generated by G′ is identical to the language generated by G. It follows that the set of languages generated by Multiple-Context Free Grammars is a subset of the set of languages generated by Minimalist Grammars. Michaelis ([Mic98]) has shown that the reverse inclusion also holds. We therefore conclude that Minimalist Grammars and Multiple-Context Free Grammars are weakly equivalent. It follows that the class of languages generated by Minimalist Grammars is a substitution-closed full AFL, since the class of languages generated by Multiple-Context Free Grammars is ([SHF91]; see for example [MS97] for AFL-theory). The result presented in this chapter also implies that head movement and covert phrasal movement do not increase the generative power of Minimalist Grammars.

[13] By definition 4, d(S) = 1, so τ has only one specifier.

CHAPTER 4

Parsing Minimalist Languages Bottom-Up

In this chapter[1] we will formally specify a bottom-up recognizer for languages generated by Minimalist Grammars without head movement and covert phrasal movement. The recognizer is similar to the CKY algorithm for Context-Free Languages ([You67]). Following the 'parsing as deduction' paradigm, the recognizer is a deductive system in which items representing trees generated by Minimalist Grammars are closed under a set of rules of inference. If the set thus generated contains a distinguished goal item representing a complete tree, the input sentence is grammatical, otherwise it is not. The specification of the items follows from Michaelis' proof that Minimalist Grammars are Multiple Context-Free Grammars ([Mic98]). The proof entails that the geometry of the trees generated by Minimalist Grammars is largely irrelevant. Consequently, the items of the recognizer are expressions that are much simpler than the trees they represent. It turns out that for any sentence w in the language generated by some Minimalist Grammar G the derivation of a distinguished goal item from the initial set of items using the rules of inference of the deductive system is so closely connected to the derivation of a complete tree for sentence w using the structure building functions $move$ and $merge$ in grammar G, that the deductive system in fact provides a succinct reformulation of Minimalist Grammars that does not use trees. This reformulation of Minimalist Grammars will be introduced informally in sections 4.1 and 4.2. The next sections contain the specification of the bottom-up recognizer, some examples, a proof of correctness,

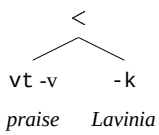
[1] This chapter is an elaboration of [Har00].

and an analysis of the time and space complexity of the recognizer. In the section before the conclusions, we will explore how the recognizer can be extended to include head movement and covert movement.

4.1 Reformulated Minimalist Grammars

In a Minimalist Grammar, new trees are derived from other trees by applications of the structure building functions *merge* and *move*. Michaelis has shown that for every Minimalist Grammar there is a Multiple Context-Free Grammar which generates the same language [Mic98]. It follows from the particular construction used in the proof of this inclusion result that the behavior with regard to the structure building functions *merge* and *move* of any tree generated by a Minimalist Grammar without head movement and covert movement is completely determined by the syntactic features appearing in the tree and whether the tree is simple or complex. The geometry of the tree is a derivational artifact of no relevance, except for the fact that the syntactic features of the head of the tree should be distinguished from syntactic features appearing at other nodes of the tree. Hence, for syntactic purposes, a tree may be collapsed into a sequence of sequences of features, associated with a tuple of strings representing the yield of the tree. For example, the tree in 1, repeated from section 2.3.1 in chapter 2, can be represented as in 2, which, for convenience, will be written as in 3. The subscript 'c' in the expressions in 2 and 3 indicates that the corresponding tree is complex. Expressions for simple trees will be marked with the subscript 's'.

1.



2. [vt -v, -k]$_c$
   (*praise*, *Lavinia*)
3. [*praise*:vt -v, *Lavinia*:-k]$_c$

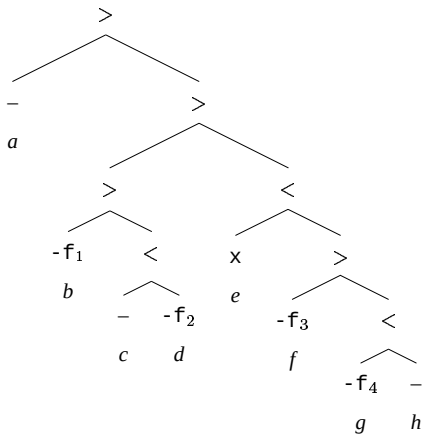
In general, following the terminology introduced in [Sta01a], a 'collapsed tree' consists of a sequence of chains. Each chain has two parts: a string and a sequence of syntactic features. The expression in 3 has two chains: *praise*:vt -v and *Lavinia*:-k, corresponding to the chains ([*praise* t_j]$_k$, t_k) and (*Lavinia*$_j$, t_j) in the sentence *Titus*$_k$ [*praise* t_i]$_j$ *s* t_k *Lavinia*$_i$ t_j eventually derived from this expression according to the Minimalist Grammar given in section 2.3.1. In fact, the tree in 26 in chapter 1 and the tree in 1 above and the expression in 3 above all represent the same structure. They are verb phrases consisting of the transitive verb *praise* and its object determiner phrase *Lavinia*. The object is part of the verb phrase, but since it will move, its phonetic yield should not be considered included in the phonetic yield of the verb phrase, as noticed originally in [Pol84]. Hence, the strings *praise* and *Lavinia* are not concatenated.

4.1.1 Collapsing Trees

Let τ be a tree generated by a Minimalist Grammar. Then the first chain of the expression e for tree τ represents the head of the tree: its features are the syntactic features labeling the head of τ, and its string consists of the phonetic material of the head of τ and any phonetic material associated with subtrees of τ that will not move out of τ. For example, the first chain of the expression corresponding to the tree in 4 is *ace*:x.[2] Note that *h* is not included in the string *ace*, because it will move as a complement of the tree whose head is labeled with the licensee feature -f$_4$.

[2] In this tree, '–' indicates absence of syntactic features. Semantic features are ignored.

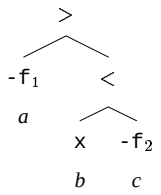
4.

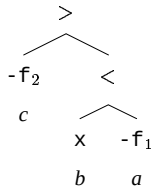


The other chains of the expression represent syntactically active subtrees of τ, that is, proper subtrees of τ whose heads have unchecked licensee features which will cause these subtrees to move.[3] The features connected with these chains are the syntactic features labeling the heads of the subtrees. The strings consist of the phonetic material that will move when the corresponding subtrees will move, minus any phonetic material of moving subtrees within these subtrees, because these will be represented by their own chain. For example, besides the chain *ace*:x the expression for the tree in 4 also contains the chains *b*:-f$_1$, *d*:-f$_2$, *f*:-f$_3$, and *gh*:-f$_4$. Note that although the node labeled with the syntactic feature -f$_3$ falls within the maximal projection of the node labeled -f$_4$, it has its own chain. Thus the complete expression for the tree in 4 is [*ace*:x, *b*:-f$_1$, *d*:-f$_2$, *f*:-f$_3$, *gh*:-f$_4$]$_c$. In fact, the order of the chains beyond the first one is immaterial, so the expression above is equivalent to the expression [*ace*:x, *gh*:-f$_4$, *d*:-f$_2$, *b*:-f$_1$, *f*:-f$_3$]$_c$, for example.

The trees in 5 and 6 both collapse into the same expression given in 7, reflecting the fact that in Minimalist Grammars there is no asymmetry between movement from a complement position and movement from a specifier position.

[3] Note that a tree containing a proper subtree whose head has a syntactic feature that is not a licensee feature is a useless tree, because this feature cannot be checked by either the structure building function *merge* or the structure building function *move*. Hence such a tree cannot participate in the derivation of a complete tree.

5. 

6. 

7. $[b{:}\mathtt{x}, a{:}\mathtt{-f}_1, c{:}\mathtt{-f}_2]_c$

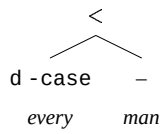
4.1.2 Reformulated Structure Building Functions

The transformation of a Minimalist Grammar into a Multiple Context-Free Grammar given in [Mic98] also suggests a succinct reformulation of the structure building functions *merge* and *move* in terms of the kind of expression given in 3. The reformulated structure building functions will combine sequences of chains rather than trees, delete pairs of matching syntactic features, and concatenate strings of chains whose features have been exhausted. A chain whose last syntactic feature has been canceled represents a syntactically inactive subtree: it has reached its final landing position and will move no further.

The original structure building function *merge* applies in four different situations – the tree whose head has the selection feature =x, henceforth the selecting tree, can be simple or complex, and the tree whose head has the category feature x, henceforth the selected tree, can move along or stay put – with varying effects. If the selecting tree is simple, the selected tree will be attached as a complement to the head of the selecting tree. If, furthermore, the selected tree is syntactically inactive after having been selected – its head does not have any syntactic features anymore – then anything

that will not move out of the selected tree will also not move out of the tree that results from merging the selecting and selected tree. In terms of collapsed expressions, this means that the string of the first chain of the expression for the selecting tree and the string of the first chain of the expression for the selected tree can be concatenated. Of course, anything that will move out of the selected tree will necessarily move out of the resulting tree. A simple example of this situation in terms of trees is given in 8. The translation into collapsed expressions is given in 9. The function Merge-1 given in a deductive format in 10 describes the general case ($t \in \{c, s\}$).

8. =n d -case *every* n *man* $\mapsto$

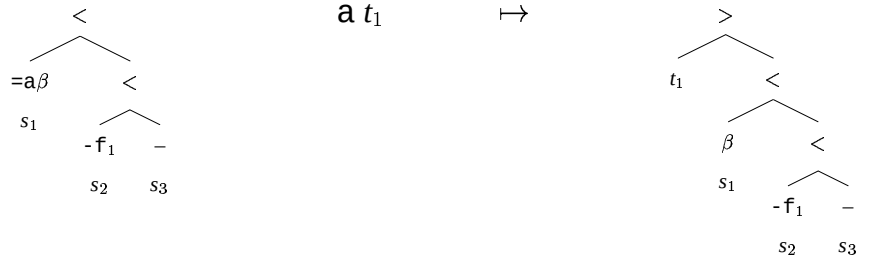


9. [*every*:=n d -case]$_s$ [*man*:n]$_s$ $\mapsto$ [*every* ⌢ *man*:d -case]$_c$

10. $$\frac{[s{:}{=}x\gamma]_s \qquad [t{:}x, \alpha_1, \ldots, \alpha_k]_t}{[st{:}\gamma, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-1

Since the noun phrase *man*, lacking any licensee features, will not move any further once it has been selected by the determiner *every* and since it is attached as a complement to the determiner *every* and complements follow their heads, the string *man* is concatenated to the right of the string *every*. In 8, the head of the resulting tree has a licensee feature -case. This will cause the entire determiner phrase *every man* to move at some later point in the derivation, but the complement *man* will never move independently from the head *every*.

The situation depicted in 11 is similar to 8, except that now the selecting tree is complex and the selected tree will therefore be attached as a specifier. Since specifiers precede their heads, string t_1 is concatenated to the left of string s_1 in 12. The general function Merge-2 describing this situation is given in 13 ($t \in \{c, s\}$).

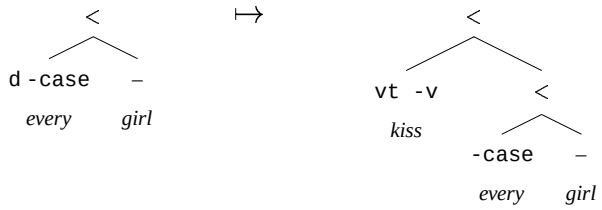
11. 

12. $[s_1$:=aβ, $s_2 s_3$:-f$_1]_c$ $[t_1$:a$]_s$ $\mapsto$ $[t_1 s_1$:β, $s_2 s_3$:-f$_1]_c$

13. $$\frac{[s\text{:=x}\gamma, \alpha_1, \ldots, \alpha_k]_c \qquad [t\text{:x}, \iota_1, \ldots, \iota_l]_t}{[ts\text{:}\gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l]_c}$$ Merge-2

Note that in the expressions 12, the string $s_2 s_3$ must be kept apart, because the subtree of which this string is the yield will eventually move.

Function Merge-3 given in 16 covers the situation in which the selected tree will move on in a later step of the derivation (in 16 δ is a non-empty sequence of licensee features, $t \in \{s, c\}$). An example is provided in 14. Because the determiner phrase *every girl* will have to move in order to check the licensee feature -case of its head, the string *every girl* should not be concatenated with the head *kiss* of the verb phrase in 15. In this situation it does not matter whether the selecting tree is simple or complex.

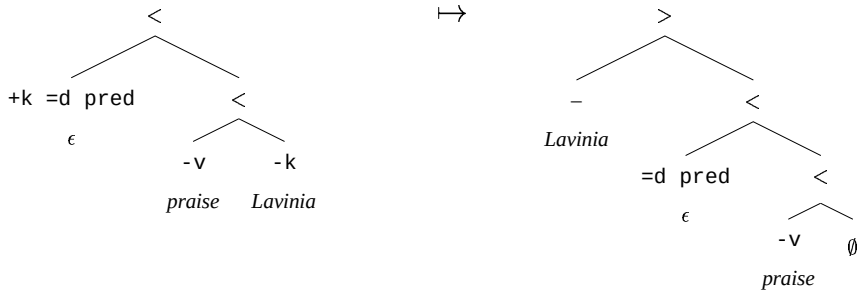
14. =d vt -v *kiss* 

15. [*kiss*:=d vt -v]$_s$ [*every girl*:d -case]$_s$ $\mapsto$ [*kiss*:vt -v, *every girl*:-case]$_c$

16. $$\frac{[s\text{:=x}\gamma, \alpha_1, \ldots, \alpha_k]_t \qquad [t\text{:x}\delta, \iota_1, \ldots, \iota_l]_c}{[s\text{:}\gamma, \alpha_1, \ldots, \alpha_k, t\text{:}\delta, \iota_1, \ldots, \iota_l]_c}$$ Merge-3

The original structure building function *move* is reformulated into two new functions Move-1 and Move-2. The function Move-1 in 19 describes the situation in which

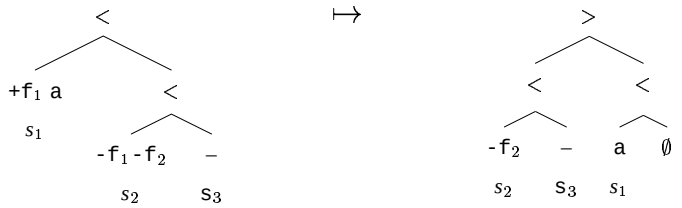
the moved element will not move any further. Since movement is to a specifier position and specifiers occur to the left of heads, concatenation will be to the left, as illustrated by the collapsed expressions in 18, which correspond to the trees in 17.

17. 

18. $[\epsilon$:+k =d pred, *praise*:-v, *Lavinia*:-k$]_c \mapsto$ [*Lavinia* ϵ:=d pred, *praise*:-v$]_c$

19. $$\frac{[s\text{:+f}\gamma,\ \alpha_1, \ldots, \alpha_{i-1},\ t\text{:-f},\ \alpha_{i+1}, \ldots, \alpha_k]_c}{[ts\text{:}\gamma,\ \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k]_c}$$ Move-1

Function Move-2 given in 22 applies when the moved element will move along in a later step of the derivation, as for example in 20 and 21 (in 22 δ is a non-empty sequence of licensee features).

20. 

21. $[s_1$:+f$_1$ a, s_2s_3:-f$_1$-f$_2]_c \mapsto [s_1$:a, s_2s_3:-f$_2]_c$

22. $$\frac{[s\text{:+f}\gamma,\ \alpha_1, \ldots, \alpha_{i-1},\ t\text{:-f}\delta,\ \alpha_{i+1}, \ldots, \alpha_k]_c}{[s\text{:}\gamma,\ \alpha_1, \ldots, \alpha_{i-1},\ t\text{:}\delta,\ \alpha_{i+1}, \ldots, \alpha_k]_c}$$ Move-2

4.1.3 Equivalence

Given a Minimalist Grammar G without head movement and covert phrasal movement as defined in sections 2.2.1 and 3.1, a reformulated Minimalist Grammar G′ is obtained by rewriting the lexical items of grammar G in chain-based format and replacing the structure building functions $merge$ and $move$ with the reformulated functions Merge-1, Merge-2, Merge-2, Move-1, and Move-2. The language generated by grammar G′ consists of all the strings s for which an expression $[s{:}\mathtt{c}]_s$ or $[s{:}\mathtt{c}]_c$ can be derived from the lexical items of the grammar by repeatedly applying the functions Merge-1, Merge-2, Merge-2, Move-1, and Move-2. It follows as a consequence from the soundness and completeness proofs presented in sections 4.5 and 4.6 of this chapter that the original Minimalist Grammar G 2.2.1 and the reformulated Minimalist Grammar G′ are equivalent: for every complete tree with yield s generated by grammar G, grammar G′ will generate an expression $[s{:}\mathtt{c}]_s$ or $[s{:}\mathtt{c}]_c$, and vice versa. Also, it follows from Michaelis' proof [Mic98] that only a finite number of instantiations of the functions Merge-1, Merge-2, Merge-3, Move-1, and Move-2 is needed to derive all the expressions of the form $[s{:}\mathtt{c}]_s$ and $[s{:}\mathtt{c}]_c$. This is a direct consequence of the Shortest Movement Constraint discussed in section 2.2.1.3 in chapter 2. For any expression e = $[s_0{:}\gamma_0, s_0{:}\gamma_1, \ldots, s_0{:}\gamma_m]_t$, $t \in \{s, c\}$ to be useful, i.e., it is one of the expressions $[s{:}\mathtt{c}]_s$ or $[s{:}\mathtt{c}]_c$, or it is used in the derivation of the expression $[s{:}\mathtt{c}]_c$, it must be the case that γ_1 through γ_m are sequences of licensee features. If some sequence γ_i is not, $1 \leq i \leq m$, then clearly expression e is not the expression $[s{:}\mathtt{c}]_s$ or the expression $[s{:}\mathtt{c}]_c$. Furthermore, expression e cannot be part a of derivation of the expression $[s{:}\mathtt{c}]_c$ because none of the structure building functions will apply and delete the one or more non-licensee features from sequence γ_i. Hence, expression e is useless and as such it can be discarded without affecting the language being generated by the grammar. Now suppose that an arbitrary expression e = $[s_0{:}\gamma_0, s_0{:}\gamma_1, \ldots, s_0{:}\gamma_m]_t$ is such that the left-

most features of both γ_i and γ_j are the same licensee feature `-f`, $1 \leq i < j \leq m$. Such an expression is also not useful, because the Shortest Movement Constraint prevents the structure building function $move$ from applying to e, so neither feature `-f` will be removed. This means that in any useful expression e = [s_0:γ_0, s_0:γ_1, ..., s_0:γ_m]$_t$, all sequences γ_i, $1 \leq i \leq m$, start with a different licensee feature. Since in each grammar there is only a finite number of licensee features available, the number of chains in e is bounded. Moreover, the sequences of features themselves are finite in length, because they are postfixes of sequences of syntactic features appearing in lexical items, and those sequences are defined to be finite. There being a finite number of useful expressions, only a finite number of instantiations of the structure building functions Merge-1, Merge-2, Merge-3, Move-1, and Move-2 is needed to derive all the expressions of the form [s:`c`]$_s$ and [s:`c`]$_c$. These instantiations are in fact the context-free rewriting rules on which the parsers in this chapter and the next are based.

4.2 Examples of Reformulated Minimalist Grammars

This section provides reformulated Minimalist Grammars for the example grammars given in section 2.3 of chapter 2.

4.2.1 Linguistic Example

The lexicon of the reformulated grammar for the Minimalist Grammar given in section 2.3.1 contains the following six items:

23. [*Lavinia*:`d -k`]$_s$
24. [*Titus*:`d -k`]$_s$
25. [*praise*:`=d vt -v`]$_s$

26. [*s*:=pred +v +k i]$_s$

27. [ϵ:=i c]$_s$

28. [ϵ:=vt +k =d pred]$_s$

The sentence *Titus praises Lavinia* is derived in eight steps, which are given below.

29. Merge-3(25, 23): [*praise*:vt -v, *Lavinia*:-k]$_c$

30. Merge-3(28, 29): [ϵ:+k =d pred, *praise*:-v, *Lavinia*:-k]$_c$

31. Move-1(30): [*Lavinia*:=d pred, *praise*:-v]$_c$

32. Merge-3(31, 24): [*Lavinia*:pred, *Titus*:-k, *praise*:-v]$_c$

33. Merge-1(26, 32): [*s Lavinia*:+v +k i *Titus*:-k, *praise*:-v]$_c$

34. Move-1(33): [*praise s Lavinia*:+k i, *Titus*:-k]$_c$

35. Move-1(34): [*Titus praise s Lavinia*:i]$_c$

36. Merge-1(27, 35): [*Titus praise s Lavinia*:c]$_c$

4.2.2 Abstract Example

Following below are the steps involved in the derivation of the sentence $a^2b^2d^2$ using the reformulated grammar corresponding to the Minimalist Grammar in section 2.3.2. The lexical items of the grammar are listed in 37 through 44; 45 through 56 are the steps of the derivation.

37. [ϵ:c]$_s$

38. [ϵ:=a +d +b +a c]$_s$

39. $[a{:}{=}\mathsf{b\ a\ -a}]_s$

40. $[b{:}{=}\mathsf{d\ b\ -b}]_s$

41. $[d{:}\mathsf{d\ -d}]_s$

42. $[a{:}{=}\mathsf{b\ +a\ a\ -a}]_s$

43. $[b{:}{=}\mathsf{d\ +b\ b\ -b}]_s$

44. $[d{:}{=}\mathsf{a\ +d\ d\ -d}]_s$

45. Merge-3(40, 41): $[b{:}\mathsf{b\ -b}, d{:}\mathsf{-d}]_c$

46. Merge-3(39, 45): $[a{:}\mathsf{a\ -a}, b{:}\mathsf{-b}, d{:}\mathsf{-d}]_c$

47. Merge-3(44, 46): $[d{:}\mathsf{+d\ d\ -d}, a{:}\mathsf{-a}, b{:}\mathsf{-b}, d{:}\mathsf{-d}]_c$

48. Move-1(47): $[dd{:}\mathsf{d\ -d}, a{:}\mathsf{-a}, b{:}\mathsf{-b}]_c$

49. Merge-3(43, 48): $[b{:}\mathsf{+b\ b\ -b}, dd{:}\mathsf{-d}, a{:}\mathsf{-a}, b{:}\mathsf{-b}]_c$

50. Move-1(49): $[bb{:}\mathsf{b\ -b}, dd{:}\mathsf{-d}, a{:}\mathsf{-a}]_c$

51. Merge-3(42, 50): $[a{:}\mathsf{+a\ a\ -a}, bb{:}\mathsf{-b}, dd{:}\mathsf{-d}, a{:}\mathsf{-a}]_c$

52. Move-1(51): $[aa{:}\mathsf{a\ -a}, bb{:}\mathsf{-b}, dd{:}\mathsf{-d}]_c$

53. Merge-3(38, 51): $[\epsilon{:}\mathsf{+d\ +b\ +a\ c}, aa{:}\mathsf{-a}, bb{:}\mathsf{-b}, dd{:}\mathsf{-d}]_c$

54. Move-1(43): $[dd{:}\mathsf{+b\ +a\ c}, aa{:}\mathsf{-a}, bb{:}\mathsf{-b}]_c$

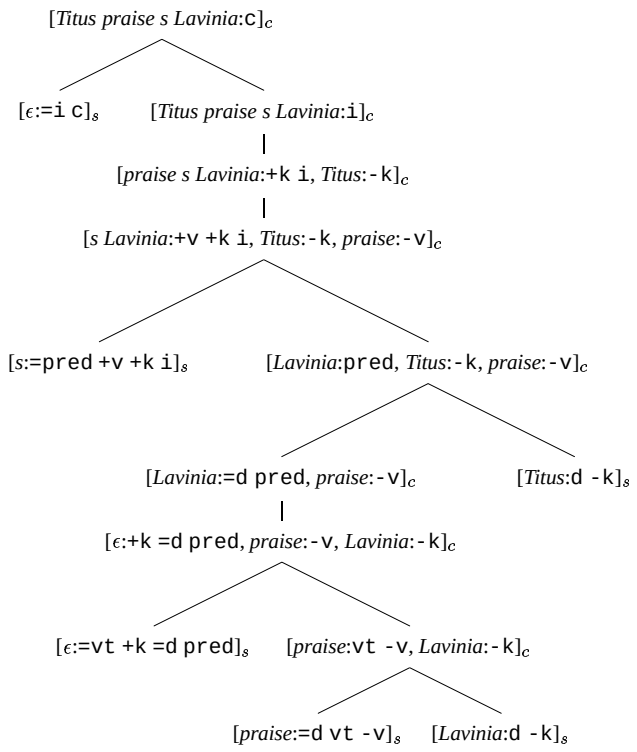
55. Move-1(54): $[bbdd{:}\mathsf{+a\ c}, aa{:}\mathsf{-a}]_c$

56. Move-1(55): $[aabbdd{:}\mathsf{c}]_c$

4.3 Parsing Minimalist Languages

The derivations shown in the previous section can be concisely represented in derivation trees. For example, the derivation tree for the sentence *Titus praises Lavinia* according the the grammar given in section 4.2.1 is given in 57 below. A derivation tree indicates how a sentence is derived from lexical items by the structure building operations and as such is very different from the 'surface' trees shown in section 2.3, for example. Tree 23 in that section, for example, exhibits the structure of the sentence *Titus praises Lavinia*, but it does not tell us what structure building operations applied to derive this sentence or in what order.

57.



The leaves of a derivation tree are lexical items. Each internal node is derived from its immediate daughter or daughters by an application of one of the functions Merge-1, Merge-2, Merge-3, Move-1, or Move-2. As is clear from the derivation tree

in 57, the leaves of the tree do not form a sentence. Rather, the sentence appears as part of the expression at the root of the tree. In fact, the root of the tree in 57 is the collapsed expression for the tree in 23 in section 2.3. Similarly, every internal node of the derivation tree 57 stands for one of the trees in section 2.3. Thus, the derivation tree in 57 is actually a tree of trees. It would be an unmanageable object, if it were not for the collapsed tree notation.

The bottom-up, top-down, and Earley-style parsers that will be presented in the remainder of this dissertation are defined on derivation trees of the sort given in 57. A bottom-up parser will start from the lexical items at the leaves of the tree, which are provided by the words of the sentence to be parsed, and try to derive the expression at the root of the tree by repeatedly applying the functions Merge-1 through Move-2. A top-down parser will start from the expression at the root of the derivation tree and, by repeatedly applying the functions Merge-1 through Move-2 in reverse, will try to break it down into lexical items at the leaves matching the words of the sentence to be parsed. The Earley-style parser essentially is a top-down parser with some additional book-keeping machinery which makes it applicable to a wider range of grammars than a basic top-down algorithm.

The parsing strategies for languages generated by Minimalist Grammars mentioned above are no different from well-known parsing strategies for languages generated by Context-Free Grammars. However, there is a fundamental difference between Context-Free Grammars and Minimalist Grammars that makes designing parsers for Minimalist Languages a non-trivial matter. If one cuts a tree licensed by some Context-Free Grammar in half and considers a non-terminal symbol A in this cut,[4] then in the sentence specified by the tree, the terminal symbols dominated by non-terminal symbol A will

[4] A sequence C of nodes of a tree T is a cut of tree T if none of the nodes in C dominates another node in C and if every node in T and not in C either dominates or is dominated by a node in C and the nodes in sequence C are ordered as in tree T.

be to the right/left of any terminal symbol that is to the right/left of A in the cut, and the terminal symbols dominated by non-terminal symbol A will be to the right/left of the terminal-symbols dominated by any non-terminal symbol that is to the right/left of A in the cut. Consequently, a parser for Context-Free Languages parsing a sentence from left to right, as the human sentence processor does, can work its way through the tree from left to right. Obviously, derivations of Minimalist Grammars do not have this 'cut property': if e_1 is a complex expression in the cut of a derivation tree of a Minimalist Grammar such that e_1 is to the right/left of another expression e_2 in the cut, then in the sentence whose derivation is given by the tree, the phonetic material of a lexical item dominated by e_1 is not necessarily to the right/left of the phonetic material of the lexical items dominated by e_2 if e_2 is a complex expression, and, if e_2 is a lexical expression, the phonetic material of a lexical item dominated by e_1 is not necessarily to the right/left of the phonetic material of expression e_2; cf. the cut – [*s*:`=pred +v +k i`]$_s$ – [*Lavinia*:`pred`, *Titus*:`-k`, *praise*:`-v`]$_c$ – in the tree 57, for example. This means that a left-to-right parser for Minimalist Languages cannot rely on the geometry of a tree; the left-to-right order will have to be enforced by some other means.

Of course, since Minimalist Grammars are weakly equivalent to Multiple Context-free Grammars ([Mic98]), the derivation trees for sentences generated by the latter formalism also lack the cut property. However, known parsing algorithms for Multiple Context-free Grammars ([SHF91], [NTS00]) use a normal form for their grammars that disallows empty elements (unless immediately generated from the start symbol). Such a normal form is not attractive for our purposes, because the linguistic theories that we want to model with Minimalist Grammars contain many phonetically empty elements. Moreover, the parsing algorithms for Multiple Context-free Grammars are not concerned with parsing a sentence from left to right.

4.4 Specification of the Bottom-Up Recognizer

This section contains a formal definition of an agenda-driven, chart-based, bottom-up recognizer for languages generated by the Minimalist Grammars without head movement and covert phrasal movement informally described in section 2.2.1 and formally defined in section 3.1. Taking a logical perspective on parsing as presented in [PW83], [SSP95] and [Sik97], the definition of the recognizer includes a specification of a grammatical deductive system and a specification of a deduction procedure.[5] The formulae of the deductive system, which are commonly called items, express claims about grammatical properties of strings. Under a given interpretation, these claims are either true or false. For a given grammar and input string, there is a set of items that, without proof, are taken to represent true grammatical claims. These are the axioms of the deductive system. Goal items represent the claim that the input string is in the language defined by the grammar. Since our objective is to recognize a string, the truth of the goal items is of particular interest. The deductive system is completed with a set of inference rules, for deriving new items from ones derived earlier. The other component of the definition of the recognizer is the specification of a deduction procedure. This is a procedure for finding all items that are true for a given grammar and input string.

4.4.1 Deduction Procedure

The deduction procedure used in the recognizer presented in this paper is taken from [SSP95]. It uses a chart holding unique items in order to avoid applying a rule of inference to items to which the rule of inference has already applied before. Furthermore, there is an agenda for temporarily keeping items whose consequences under the inference rules have not been generated yet. The deductive procedure is defined as

[5] For a constraint programming perspective on the recognizer see [Mor00].

follows:

1. Initialize the chart to the empty set of items and the agenda to the axioms of the deduction system.

2. Repeat the following steps until the agenda is exhausted:

   a) Select an item from the agenda, called the trigger item, and remove it from the agenda.

   b) Add the trigger item to the chart, if the item is not already in the chart.

   c) If the trigger item was added to the chart, generate all items that can be derived from the trigger item and any items in the chart by one application of a rule of inference,[6] and add these generated items to the agenda.

3. If a goal item is in the chart, the goal is proved, i.e., the string is recognized, otherwise it is not.

Shieber et al. ([SSP95]) prove that the deductive procedure is sound – it generates only items that are derivable from the axioms – and complete – it generates all the items that are derivable from the axioms.

4.4.2 Deductive System

Given an input string $w = w_1 \ldots w_n$ and Minimalist Grammar G = (V, Cat, Lex, F) as defined in section 3.1, the items of the deductive system will be of the form $[\alpha_0, \alpha_1, \ldots, \alpha_m]_t$, where $m \leq |\textit{licensees}|$, $t \in \{s, c\}$. For $0 \leq i \leq m$, α_i is of the form (x_i,

[6] Note that successful applications of the rules of inference Move-1 and Move-2 as defined in section 4.4.2.2 do not involve any items from the chart.

y_i):γ_i, where $0 \leq x_i \leq y_i \leq n$, and $\gamma_i \in Cat^*$. The collapsed trees introduced in section 4.1.1 are items whose position vectors (x_i, y_i) have been replaced with substrings $w_{x_i+1} \ldots w_{y_i}$ of the input string w, $0 \leq i \leq m$.

The proper interpretation of the items requires the notion of narrow yield of a tree. The narrow yield $Y_n(\phi)$ of a minimalist tree ϕ is defined in the following way. If ϕ is a complex tree, then either $\phi = [_> \tau, \upsilon]$, or $\phi = [_< \tau, \upsilon]$.[7] If $\phi = [_> \tau, \upsilon]$, then:

$Y_n(\phi) = Y_n(\tau) \cdot Y_n(\upsilon)$ if τ does not have a feature `-f` $\in$ *licensees*[8,9]

$Y_n(\phi) = Y_n(\upsilon)$ otherwise.

If $\phi = [_< \tau, \upsilon]$, then:

$Y_n(\phi) = Y_n(\tau) \cdot Y_n(\upsilon)$ if υ does not have a feature `-f` $\in$ *licensees*

$Y_n(\phi) = Y_n(\tau)$ otherwise.

If ϕ is not a complex tree, it must be a simple tree. In that case:

$Y_n(\phi) = Y(\phi)$

where $Y(\phi)$ denotes the concatenation of the phonetic features appearing at the leaves of tree ϕ, ordered according to the precedence relation on the nodes of ϕ. Informally, the narrow yield of a tree is that part of its phonetic yield that will not move out of the tree by some application of the structure building function *move*, as discussed in sections 4.1.1 and 4.1.2.

Now, an item $[(x_0, y_0):\gamma_0, (x_1, y_1):\gamma_1, \ldots, (x_m, y_m):\gamma_m]_t$ is understood to assert the existence of a tree $\tau \in CL(G)$ which has the following properties:

[7] For this particular notation of trees, see section 3.1.
[8] A tree τ has a feature `-f` if the left-most feature of the head of τ is the feature `-f`.
[9] $s \cdot t$ is the concatenation of strings s and t.

1. If t = s, τ is a simple tree; if t = c, τ is a complex tree.

2. The head of τ is labeled $\gamma_0\pi\iota$, $\pi\iota \in$ P*I*.

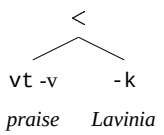
3. For every (x_i, y_i):γ_i, $1 \leq i \leq m$, there is a leaf in τ labeled $\gamma_i\pi\iota$, $\pi\iota \in$ P*I*.

4. Besides the nodes labeled by $\gamma_i\pi\iota$, $\pi\iota \in$ P*I*, $0 \leq i \leq m$, there are no other nodes with syntactic features in τ.

5. The narrow yield of the subtree whose root is the maximal projection of the node labeled by $\gamma_i\pi\iota$, $\pi\iota \in$ P*I*, is $w_{x_i+1} \ldots w_{y_i}$, $0 \leq i \leq m$.

For example, for the input sentence *Titus praise s Lavinia* and a Minimalist Grammar G such that the tree in 58 is in the closure of G, the recognizer, if complete, will generate the item in 59.

58.



59. [(1, 2):`vt -v`, (3, 4):`-k`]$_c$

4.4.2.1 Axioms and Goals

The set of axioms of the deductive system is specified in the following way. For each lexical item in Lex with syntactic features $\gamma \in$ Cat* and whose phonetic features cover $w_{i+1} \ldots w_j$ of the input string, there will be an axiom [(i, j):γ]$_s$ in the deductive system.[10]

[10] We assume that every lexical item of the grammar has one and only one category feature and that all syntactic features to the right of the category feature are licensee features. Lexical items that do not meet these conditions are useless in the sense that they are not complete trees by themselves nor can they participate in the derivation of a complete tree, because no combination of applications of the structure building functions $merge$ and $move$ will be able to remove all the syntactic features of these items.

There will be two goal items: $[(0, n):\mathsf{c}]_s$ and $[(0, n):\mathsf{c}]_c$, $\mathsf{c} \in$ *base* being the distinguished category feature. These are appropriate goal items for a recognizer, since their truth under the interpretation provided above requires the existence of a complete tree $\tau \in$ CL(G), either simple or complex, with narrow yield $Y_n(\tau) = w_1 \ldots w_n = w$. Since τ is complete, $Y_n(\tau) = Y(\tau)$. Therefore, $w \in L(G) = \{Y(\tau) | \tau \in CL(G)$ and τ is complete$\}$.

4.4.2.2 Rules of Inference

The deductive system has five rules of inference, grouped into Merge rules and Move rules:

Merge-1:

$$\frac{[(p, q):=\mathsf{x}\gamma]_s \qquad [(q, v):\mathsf{x}, \alpha_1, \ldots, \alpha_k]_t}{[(p, v):\gamma, \alpha_1, \ldots, \alpha_k]_c} \text{ Merge-1}$$

Merge-2:

$$\frac{[(p, q):=\mathsf{x}\gamma, \alpha_1, \ldots, \alpha_k]_c \qquad [(v, p):\mathsf{x}, \iota_1, \ldots, \iota_l]_t}{[(v, q):\gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l]_c} \text{ Merge-2}$$

Merge-3: ($\delta \neq \emptyset$)

$$\frac{[(p, q):=\mathsf{x}\gamma, \alpha_1, \ldots, \alpha_k]_{t_1} \qquad [(v, w):\mathsf{x}\delta, \iota_1, \ldots, \iota_l]_{t_2}}{[(p, q):\gamma, \alpha_1, \ldots, \alpha_k, (v, w):\delta, \iota_1, \ldots, \iota_l]_c} \text{ Merge-3}$$

Move-1:

$$\frac{[(p, q):+\mathsf{y}\gamma, \alpha_1, \ldots, \alpha_{i-1}, (v, p):\text{-}\mathsf{y}, \alpha_{i+1}, \ldots, \alpha_k]_c}{[(v, q):\gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k]_c} \text{ Move-1}$$

Move-2: ($\delta \neq \emptyset$)

$$\frac{[(p, q):+y\gamma, \alpha_1, \ldots, \alpha_{i-1}, (v, w):-y\delta, \alpha_{i+1}, \ldots, \alpha_k]_c}{[(p, q):\gamma, \alpha_1, \ldots, \alpha_{i-1}, (v, w):\delta, \alpha_{i+1}, \ldots, \alpha_k]_c} \text{ Move-2}$$

For all rules the following holds: $0 \leq p, q, v, w \leq n$; $0 \leq i, k, l \leq m$; and $t, t_1, t_2 \in \{s, c\}$; =x $\in$ *select*; x $\in$ *base*, +y $\in$ *licensors*; and -y $\in$ *licensees*. The rules Move-1 and Move-2 come with an additional condition on their application: if (x_j, y_j):γ_j is one of the $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$, then the first feature of γ_j is not -y, reflecting the Shortest Movement Constraint discussed in section 2.2.1.3. It is possible that an application of rule Merge-2 produces an item $[\alpha_0, \alpha_1, \ldots, \alpha_m]_c$ such that for some α_i = (x_i, y_i):γ_i and α_j = (x_j, y_j):γ_j the first feature of both γ_i and γ_j is -y $\in licensees$, $1 \leq i < j \leq m$. Clearly, such an item is not a goal item, nor can a goal item be derived from it, because none of the rules of inference can delete the features -y. Hence, the rule of inference Merge-3 is restricted so that it will not add items of this kind to the agenda. In this way we ensure that for every item $[\alpha_0, \alpha_1, \ldots, \alpha_m]_c$ produced by the recognizer $m \leq |\textit{licensees}|$.

Note that the reformulated structure building functions introduced in section 4.1.2 are the rules of inference of the recognizer with strings rather than position vectors.

Since Shieber et al. ([SSP95]) proved that the deductive procedure is sound and complete, establishing the correctness of the recognizer entails showing that the deductive system defined above is sound and complete relative to the intended interpretation of the items. This will be done in the next two sections.

4.5 Proof of Soundness

The following two lemmas will be helpful for establishing soundness of the deductive system.

Lemma 1a. *If τ, τ' and υ, υ' are trees such that* merge*(τ, υ) = [$_<\tau'$, υ'] or* merge*(τ, υ) = [$_>\upsilon'$, τ'], then $Y_n(\tau') = Y_n(\tau)$ and $Y_n(\upsilon') = Y_n(\upsilon)$.*

Proof. Inspection of the definition of *merge* shows that in both cases τ' and τ are identical except for the labels of their heads. According to the definition of narrow yield the label of the head of a tree is of no relevance for determining the narrow yield of that tree. Analogously, $Y_n(\upsilon') = Y_n(\upsilon)$. □

Lemma 1b. *If τ, τ' and τ_0, τ_0' are trees such that* move*(τ) = [$_>\tau_0'$, τ'], then $Y_n(\tau') = Y_n(\tau)$ and $Y_n(\tau_0') = Y_n(\tau_0)$.*

Proof. According to the definition of *move*, τ' will be like τ except that feature `+y` is deleted and a subtree τ_0 has been replaced by a single node with no features. As is the case for *merge*, the different head labels of τ and τ' are irrelevant as narrow yields are concerned. Furthermore, the yield of τ_0 is excluded from $Y_n(\tau)$ because τ_0 has a feature `-f` $\in$ *licensees*. The yield of τ_0 is also excluded from $Y_n(\tau')$ because τ_0 is not a subtree of τ'. Since trees τ and τ' are otherwise the same, $Y_n(\tau') = Y_n(\tau)$. Concerning τ_0' and τ_0, these trees differ only by their head label. Hence, $Y_n(\tau_0') = Y_n(\tau_0)$. □

Proving soundness of the recognizer amounts to showing that the axioms and the rules of inference of the deductive system are sound. Then it will follow that every derivable item in the deductive system will represent a true grammatical statement under the intended interpretation.

4.5.1 Soundness of the Axioms

According to the interpretation given in section 4.4.2, an axiom $[(i, j):\gamma]_s$ asserts the existence of a tree $\tau \in$ CL(G) with the following properties:

1. Tree τ is a simple tree.

2. The head of τ is labeled by $\gamma\pi\iota$, for some $\pi\iota \in P^*I^*$.

3. Besides the head of τ there are no other nodes with syntactic features in τ.

4. The narrow yield of τ is $w_{i+1} \ldots w_j$.

It is easy to see that the lexical item in Lex which occasioned the axiom [(i, j):γ]$_s$ has exactly these properties. By definition this lexical item is in CL^0(G) $\subseteq$ CL(G).

4.5.2 Soundness of the Rules of Inference

There are five rules of inference. Their soundness will be established below.[11]

Merge-1: the items [(p, q):=xγ]$_s$ and [(q, v):x, α_1, …, α_k]$_t$ in the antecedent of the rule Merge-1 assert the existence of trees τ and $\upsilon \in$ CL(G), whose heads are labeled by =xγ and x, respectively. Hence, τ and υ are in the domain of the function *merge*. This function will apply to τ and υ and produce a complex tree ϕ = [$_<\tau'$, υ'] $\in$ CL(G), since τ is a simple tree. The head of ϕ is labeled by γ. Furthermore, $Y_n(\phi) = Y_n(\tau') \cdot Y_n(\upsilon')$, since the head of υ' does not have a feature -f $\in$ *licensees*. By lemma 1a, $Y_n(\tau') \cdot Y_n(\upsilon') = Y_n(\tau) \cdot Y_n(\upsilon) = w_{p+1} \ldots w_q \cdot w_{q+1} \ldots w_v = w_{p+1} \ldots w_v$. Also, all maximal subtrees properly contained in υ will be included in ϕ with their head labels and narrow yields unchanged, since *merge* does not touch any proper subtrees of υ. As is easy to check, the item [(p, v):γ, α_1, …, α_k]$_c$ in the consequent of the rule Merge-1 claims the existence of a tree in CL(G) with the properties of ϕ. Thus Merge-1 is sound.

Merge-2: the argument unfolds in a fashion similar to Merge-1, except that now ϕ = [$_>\upsilon'$, τ'], because τ is complex. Consequently, $Y_n(\phi) = Y_n(\upsilon) \cdot Y_n(\tau) = w_{v+1} \ldots w_p \cdot w_{p+1} \ldots w_q$ = $w_v \ldots w_q$. Furthermore, ϕ inherits the narrow yields and head labels of the maximal

[11] Since applicability of the structure building functions and the rules of inference does not depend on non-syntactic features, their presence in trees will be ignored in the discussion to follow. Thus, the statement 'the head of tree ϕ is labeled γ', for example, means: 'the head of tree ϕ is labeled $\gamma\pi\iota$, for some $\pi\iota \in$ P*I*'.

subtrees in both τ and υ.

Merge-3: application of Merge-3 presupposes the existence of trees τ, $\upsilon \in$ CL(G) which will produce another tree ϕ = *merge*(τ, υ) = [$_{<}\tau'$, υ'] $\in$ CL(G). According to the definition of the function *Label*, the first feature of δ, which is the label of the head of υ', must be a feature `-f` $\in$ *licensees*. Therefore, by the definition of narrow yield and lemma 1a, $Y_n(\phi) = Y_n(\tau') = Y_n(\tau) = w_{p+1} \ldots w_q$. Obviously, υ', whose head is labeled δ, will be a maximal subtree of ϕ. By lemma 1a, $Y_n(\upsilon') = Y_n(\upsilon) = w_{v+1} \ldots w_w$. Now it is easy to see that the grammatical claim made by the item in the consequent of the rule Merge-3 is justified by $\phi \in$ CL(G).

Move-1: rule Move-1 will apply provided there is a complex tree $\tau \in$ CL(G) whose head is labeled by `+y`γ, which contains one leaf labeled by a feature `-y` and no other leaves whose first feature is `-y`. Let τ_0 be the maximal subtree in τ projected by the node labeled with the single feature `-y`.[12] Then τ is in the domain of the function *move*. The result of applying *move* to τ will be a complex tree ϕ = [$_{>}\tau_0'$, τ'] $\in$ CL(G). The head of ϕ is labeled γ. Moreover, $Y_n(\phi) = Y_n(\tau_0') \cdot Y_n(\tau') = Y_n(\tau_0) \cdot Y_n(\tau)$ by lemma 1b and because the head of τ_0' has an empty label. Since $Y_n(\tau_0) = w_{v+1} \ldots w_p$ and $Y_n(\tau) = w_{p+1} \ldots w_q$, $Y_n(\phi) = w_{v+1} \ldots w_q$. The function *move* does not affect the labels or narrow yields of any of the maximal subtrees properly contained in τ, except for τ_0 (cf. the third case in the proof of lemma 2 in section 4.6). Now it is easy to see that the item in the consequent of the inference rule Move-1 actually describes $\phi \in$ CL(G).

Move-2: application of rule Move-2 requires the existence in CL(G) of a complex tree τ to which the function *move* will apply to produce a complex tree ϕ = [$_{>}\tau_0'$, τ'] $\in$ CL(G), whose head is labeled γ, τ_0' and τ' as in the definition of *move*. $Y_n(\phi) = Y_n(\tau')$, since τ_0', whose head is labeled by δ, has a feature `-f` $\in$ *licensees*, cf. the similar

[12] $\tau \neq \tau_0$ because their head labels are different.

situation Merge-3. As in Move-1, the narrow yields and labels of maximal subtrees properly contained in τ are left intact, except for τ_0. τ_0 is not a subtree of ϕ, but τ_0' is. The label of the head of τ_0' is δ and τ_0''s narrow yield is $Y_n(\tau_0') = Y_n(\tau_0)$, by lemma 1b. It is easily checked that $\phi \in$ CL(G) has the same properties as the tree claimed to exist by the item in the consequent of Move-2. Hence, Move-2 is sound.

4.6 Proof of Completeness

This section presents a completeness proof for the recognizer. A recognizer is complete if for every string that is in the language defined by the grammar, there is a derivation of a goal item from the axioms.

First, the following two useful lemmas will be proved.

Lemma 2. *If the derivation of tree ϕ in a Minimalist Grammar G immediately includes tree τ, then for every maximal subtree σ contained in τ, there is a maximal subtree σ' in ϕ such that $Y_n(\sigma)$ is a substring of $Y_n(\sigma')$.*

Proof. Three distinct cases have to be considered: there is an $\upsilon \in$ CL(G) such that ϕ = *merge*(τ, υ), there is an $\upsilon \in$ CL(G) such that ϕ = *merge*(υ, τ), and ϕ = *move*(τ).

In the first case, either $\phi = [_< \tau', \upsilon']$ or $\phi = [_> \upsilon', \tau']$, depending on whether τ is a simple or a complex tree. Furthermore, either σ is a proper subtree of τ, or $\sigma = \tau$. If σ is a proper subtree of τ, then τ' will properly contain σ, as τ' is like τ except that =x is deleted from the label of the head. For the same reason, σ is maximal in τ'. Since ϕ contains τ', σ will be a subtree of ϕ and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\sigma)$. If $\sigma = \tau$, then $Y_n(\sigma) = Y_n(\tau) = Y_n(\tau')$ by lemma 1a. Also, $Y_n(\tau')$ is a substring of $Y_n(\phi)$ by the definition of narrow yield. Trivially, ϕ is a maximal subtree of ϕ.

In the second case, either $\phi = [_< \upsilon', \tau']$ or $\phi = [_> \tau', \upsilon']$, depending on whether υ is a simple or a complex tree. Again, either σ is a proper subtree of τ, or $\sigma = \tau$. If σ is

a proper subtree of τ, then σ will be a proper, maximal subtree of ϕ, as argued for the similar situation in the case above. If $\sigma = \tau$, then $Y_n(\sigma) = Y_n(\tau) = Y_n(\tau')$ by lemma 1a. Tree τ' is a maximal subtree contained in ϕ, and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\tau')$.

In the third case, $\phi = [_{>}\tau_0', \tau']$, with τ_0 specified as in the definition of *move*. Either σ is a proper subtree of τ_0, or $\sigma = \tau_0$, or σ is a proper subtree of τ and τ_0 is a proper subtree of σ, or $\sigma = \tau$. If σ is a proper subtree of τ_0, then σ will be a proper, maximal subtree of ϕ, as in the similar situations above. If $\sigma = \tau_0$, then $Y_n(\sigma) = Y_n(\tau_0) = Y_n(\tau_0')$ by lemma 1b. Now, τ_0' is a maximal subtree contained in ϕ, and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\tau_0')$. If σ is a proper subtree of τ and τ_0 is a proper subtree of σ, then, by the definition of *move*, τ' will contain a tree σ' which is like σ, except that subtree τ_0 is replaced by a single node without features. $Y_n(\sigma) = Y_n(\sigma')$, since the yield of τ_0 is excluded from $Y_n(\sigma)$ because of its `-f` feature, and the yield of τ_0 is excluded from $Y_n(\sigma')$ because τ_0 is not a subtree of σ'. Hence, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\sigma')$. Moreover, σ' is a maximal subtree of τ' since σ is a proper, maximal subtree of τ. Finally, if $\sigma = \tau$, then $Y_n(\sigma) = Y_n(\tau) = Y_n(\tau')$ by lemma 1b. Furthermore, $Y_n(\tau')$ is a substring of $Y_n(\phi)$ by the definition of narrow yield. Trivially, ϕ is a maximal subtree contained in ϕ. □

Lemma 3. *If for a Minimalist Grammar G, the derivation of a complete tree γ includes τ, then $Y_n(\tau)$ is a substring of $Y(\gamma)$.*

Proof. By repeated application of lemma 2, γ will contain a maximal subtree σ' such that $Y_n(\tau)$ is a substring of $Y_n(\sigma')$. Since γ is a complete tree, it does not have any labels with a feature `-f` $\in$ *licensees*. Hence, by the definition of narrow yield, $Y_n(\sigma')$ is a substring of $Y_n(\gamma) = Y(\gamma)$, and, consequently, $Y_n(\tau)$ is a substring of $Y(\gamma)$. □

Completeness of the recognizer will follow as a corollary of lemma 4 below. If w

$\in$ L(G), for some Minimalist Grammar G, then there is a complete tree $\phi \in$ CL(G) such that Y(ϕ) = w. Since $\phi \in$ CL(G), there must be a $k \in \mathbb{N}$ such that $\phi \in$ CL^k(G). Since ϕ is a complete tree, lemma 4 guarantees that an item corresponding to ϕ will be generated. Obviously, the item will be a goal item, since ϕ is complete and Y(ϕ) = w.

Lemma 4. *For a given Minimalist Grammar G, if $\phi \in$ CL^k(G), $k \in \mathbb{N}$, and ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, then an item $[\alpha_0, \alpha_1 \ldots, \alpha_m]_t$ corresponding to ϕ will be generated, $[\alpha_0, \alpha_1, \ldots, \alpha_m]_t$ as defined in section 4.4.2.*

Proof.[13] It will be shown by induction over k that for arbitrary $k \in \mathbb{N}$ and $\phi \in$ CL^k(G) such that ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, an item corresponding to ϕ will be generated.

1. k = 0. Then $\phi \in$ CL^0(G) = Lex, and ϕ is covered by one of the axioms.

2. Assume all items corresponding to $\phi \in$ CL^k(G), ϕ included in the derivation of a complete tree or a complete tree itself, are generated for a particular $k \in \mathbb{N}$. To show: for any $\phi \in$ CL^{k+1}(G), ϕ included in the derivation of a complete tree or a complete tree itself, a corresponding item will be generated.

Pick an arbitrary $\phi \in$ CL^{k+1}(G). By definition, CL^{k+1}(G) = CL^k(G) $\cup$ $\{$*merge*(τ, υ)$|(\tau, \upsilon) \in$ Dom(*merge*) $\cap$ CL^k(G)$\times$CL^k(G)$\}$ $\cup$ $\{$*move*(τ)$|\tau \in$ Dom(*move*) $\cap$ CL^k(G)$\}$. Hence, three cases can be distinguished: $\phi \in$ CL^k(G), $\phi \in$ $\{$*merge*(τ, υ)$|(\tau, \upsilon) \in$ Dom(*merge*) $\cap$ CL^k(G)$\times$CL^k(G)$\}$, and $\phi \in$ $\{$*move*(τ)$|\tau \in$ Dom(*move*) $\cap$ CL^k(G)$\}$.

In the first case, $\phi \in$ CL^k(G). Then, by the induction hypothesis, an item corresponding to ϕ will be generated.

In the second case, $\phi \in$ $\{$*merge*(τ, υ)$|(\tau, \upsilon) \in$ Dom(*merge*) $\cap$ CL^k(G)$\times$CL^k(G)$\}$. Then there are trees τ, $\upsilon \in$ CL^k(G) such that ϕ = *merge*(τ, υ). Since ϕ is included in

[13] As in the discussion of the soundness proof, the presence of non-syntactic features in trees will be ignored.

the derivation of a complete tree or ϕ is a complete tree itself, τ and υ are included in the derivation of a complete tree.[14] Hence, by the induction hypothesis, there are items corresponding to τ and υ. Since (τ, υ) $\in$ Dom(*merge*), the heads of τ and υ are respectively labeled =xγ and xδ, for =x $\in$ *licensees*, x $\in$ *base* and γ, $\delta \in$ Cat*. Tree τ is either a simple tree or a complex tree. With regard to υ, $\delta = \emptyset$ or $\delta \neq \emptyset$. Hence, there are four cases to be dealt with. If τ is a simple tree and $\delta = \emptyset$, i.e., the head of υ consists of the single feature x, then $\phi = [_{<}\tau', \upsilon']$, and $Y_n(\phi) = Y_n(\tau')\cdot Y_n(\upsilon')$. By lemma 1a, $Y_n(\phi) = Y_n(\tau)\cdot Y_n(\upsilon)$. Suppose ϕ participates in a successful derivation of a complete tree whose yield is the string $w_1 \ldots w_m$.[15] Then, by lemma 3, $Y_n(\tau)\cdot Y_n(\upsilon)$ is a substring of $w_1 \ldots w_m$, that is, $Y_n(\tau) = w_p \ldots w_q$ and $Y_n(\upsilon) = w_{q+1} \ldots w_v$, $1 \leq p \leq q \leq v \leq m$. Hence, the items corresponding to τ and υ will match the antecedents of rule Merge-1. Alternatively, ϕ is a complete tree itself. Assume $Y_n(\phi) = Y(\phi) = w_1 \ldots w_m$. Then it follows immediately that $Y_n(\tau) = w_1 \ldots w_q$ and $Y_n(\upsilon) = w_{q+1} \ldots w_m$, $1 \leq q \leq m$. Again, the items corresponding to τ and υ will match the antecedents of rule Merge-1. In both cases, application of Merge-1 will generate the item corresponding to ϕ. In a similar way it is established that the trees τ and υ for the other three cases will correspond to items that match the antecedents of the rules Merge-2 (τ is complex, $\delta \neq \emptyset$) and Merge-3 (τ is simple or complex, $\delta \neq \emptyset$). Application of these rules will produce an item corresponding to tree ϕ.

In the third case, $\phi \in \{$*move*$(\tau) | \tau \in$ Dom(*move*) $\cap$ CL^k(G)$\}$. Then there is a $\tau \in$ CL^k(G) such that $\phi =$ *move*(τ). Since ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, τ is included in the derivation of a complete tree as well. Hence, by the induction hypothesis, there is an item corresponding to τ. Since $\tau \in$ Dom(*move*), τ has a feature +y and τ has exactly one maximal subtree that has the feature -y, +y $\in$ *licensors*, -y $\in$ *licensees*. Let τ_0 be the maximal subtree of τ

[14] Since τ is the first argument of *merge*, τ cannot be a complete tree.

[15] Reference to the yield of a complete tree derived from ϕ precludes a general proof of completeness, i.e. a proof that for all $\phi \in$ CL^k(G), $k \in \mathbb{N}$, a corresponding item $[\alpha_0, \alpha_1 \ldots, \alpha_m]_t$ will be generated.

that has feature -y and let the label of its head be -yδ. Then there are two cases to be considered: $\delta = \emptyset$ and $\delta \neq \emptyset$. If $\delta = \emptyset$, then $\phi = [_{>}\tau_0', \tau']$, and $Y_n(\phi) = Y_n(\tau_0')\cdot Y_n(\tau') = Y_n(\tau_0)\cdot Y_n(\tau)$ by the definition of narrow yield and lemma 1b. As in the case of rules Merge-1 and Merge-2, $Y_n(\tau_0)\cdot Y_n(\tau)$ will be a substring of $w_1 \ldots w_m$, where $w_1 \ldots w_m$ is the yield of a complete tree derived from ϕ or the yield of ϕ itself if ϕ is a complete tree. Therefore, $Y_n(\tau_0) = w_v \ldots w_p$ and $Y_n(\tau) = w_{p+1} \ldots w_q$, $1 \leq v \leq p \leq q \leq m$. Hence, the item corresponding to τ will match the antecedent of rule Move-1. Application of this rule will generate the item corresponding to ϕ. If $\delta \neq \emptyset$, then the item generated for τ will match the antecedent of rule Move-2, application of which will generate an item corresponding to ϕ. □

4.7 Complexity Results

For a given Minimalist Grammar G = (V, Lex, Cat, $\mathcal{F}$) and input string of length n, the number of items is polynomially bounded by the length of the input string. All items are of the form $[(x_0, y_0):\gamma_0, (x_1, y_1):\gamma_1, \ldots, (x_m, y_m):\gamma_m]_t$, as defined in section 4.4.2. Each part $(x_i, y_i):\gamma_i$ of an item, $0 \leq i \leq m$, has $O(n^2)$ possible instantiations, as both x_i and y_i range between 0 and n, 0 and n included. The possible choices of γ_i do not depend on the length of the input string. The number of choices is bounded, however, because the labels occurring in any tree in CL(G) are substrings of the labels of the expressions in Lex. This follows immediately from the the definition of *merge* and *move*. Moreover, Lex is a finite set and the labels assigned by *Label* are finite sequences. Thus, the set of possible labels is bounded by the grammar, and, since the recognizer is sound, the number of possible γ_i's is bounded by the grammar, too. Since an item has at most $k + 1$ parts $(x_i, y_i):\gamma_i$, where $k = |\textit{licensees}|$, the number of items in the chart is bounded by $O(n^{2k+2})$.

As regards the time complexity of the recognizer, step 2.b) of the deduction proce-

dure specified in section 4.4.1 requires every item on the agenda to be compared with the items already in the chart. Since the number of items in the chart is $O(n^{2k+2})$, this step will take $O(n^{2k+2})$ per item on the agenda. For any item on the agenda but not already in the chart, step 2.c) of the deduction procedure checks whether any rule of inference will apply. Checking applicability of the Merge rules involves looking through all the items in the chart, since all Merge rules have two antecedents. Given an item on the agenda and an item from the chart, actually verifying whether any of the Merge rules applies to these items takes constant time. Thus, the time cost is $O(n^{2k+2})$ per item on the agenda. In order to determine whether one of the two Move rules will apply, the label γ_0 in an item has to be inspected and compared to the other labels γ_i, $1 \leq i \leq m$ in the same item. Since m is bounded by k = $|$*licensees*$|$, there is no dependency on the length of the input string. Since steps 2.b) and 2.c) are performed in sequence, the time cost of both steps is bounded by $O(n^{2k+2})$ per item on the agenda, ignoring without loss of generality the fact that step 2.c) is not performed for all items on the agenda. Steps 1.and 3.of the deduction procedure do not exceed this bound. The number of items that will be put on the agenda while recognizing a string is $O(n^{2k+2})$. This is the upper bound on the number of possible items. There will be duplicates in the agenda, but their number is finite and does not depend on n, essentially because the number of axioms and the number of inference rules is finite and all items in the chart are unique. Thus, the overall time complexity of the recognizer is $O(n^{4k+4})$.

4.8 Recognition Example

The deductive system for the Minimalist Grammar given in section 2.3.1 and the input sentence *Titus praise s Lavinia* has 14 axioms. These are given in 60 through 73 below.

60. $[(0, 0):\texttt{=i c}]_s$

61. $[(0, 0):\texttt{=vt +k =d pred}]_s$

62. $[(0, 1):\texttt{d -k}]_s$

63. $[(1, 1):\texttt{=i c}]_s$

64. $[(1, 1):\texttt{=vt +k =d pred}]_s$

65. $[(1, 2):\texttt{=d vt -v}]_s$

66. $[(2, 2):\texttt{=i c}]_s$

67. $[(2, 2):\texttt{=vt +k =d pred}]_s$

68. $[(2, 3):\texttt{=pred +v +k i}]_s$

69. $[(3, 3):\texttt{=i c}]_s$

70. $[(3, 3):\texttt{=vt +k =d pred}]_s$

71. $[(3, 4):\texttt{d -k}]_s$

72. $[(4, 4):\texttt{=i c}]_s$

73. $[(4, 4):\texttt{=vt +k =d pred}]_s$

From these axioms, the recognizer will derive a goal item $[(0, 4):\texttt{c}]_c$. The items involved in the derivation of the goal item are given below. Note the close connection between this derivation and the derivation of the sentence *Titus praise s Lavinia* in section 2.3.1. To simplify the presentation we will not mention the agenda of the recognizer explicitly.

74. Merge-3(65, 71): $[(1, 2):\texttt{vt -v}, (3, 4):\texttt{-k}]_c$

75. Merge-3(73, 74): $[(4, 4):\texttt{+k =d pred}, (1, 2):\texttt{-v}, (3, 4):\texttt{-k}]_c$

76. Move-1(75): $[(3, 4):\texttt{=d pred}, (1, 2):\texttt{-v}]_c$

77. Merge-3(76, 62): $[(3, 4):\texttt{pred}, (0, 1):\texttt{-k}, (1, 2):\texttt{-v}]_c$

78. Merge-1(68, 77): $[(2, 4):\texttt{+v +k i}\ (0, 1):\texttt{-k}, (1, 2):\texttt{-v}]_c$

79. Move-1(78): $[(1, 4):\texttt{+k i}, (0, 1):\texttt{-k}]_c$

80. Move-1(79): $[(0, 4):\texttt{i}]_c$

81. Merge-1(60, 80): $[(0, 4):\texttt{c}]_c$

4.9 Extensions

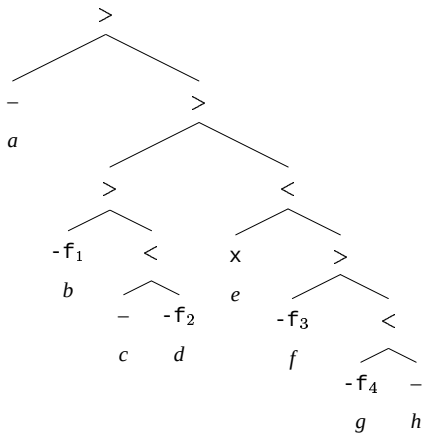
The reformulated Minimalist Grammars discussed in section 4.1 of this chapter do not allow head movement, nor covert phrasal movement. In this section we will briefly discuss the extensions necessary to include these two kinds of movement in our reformulated grammars.

4.9.1 Head Movement

The general idea behind the 'collapsed tree' notation is to split a tree and its phonetic yield into parts that will move independently from one another. If τ is a tree and e its corresponding expression, then the first chain of e represents the head of τ. According to the conventions introduced in section 4.1.1, the string associated with the first chain of an expression consists of the phonetic material of the head of the tree and phonetic material elsewhere in the tree that will not move. In order to capture head movement, the phonetic material of the head must be kept separate from the the phonetic material elsewhere in the tree that does not move. Thus, following [Sta01b], rather than writing $t{:}\gamma$ for the first chain of an expression, we will now write s, h, $c{:}\gamma$, where t is the

concatenation of s, h, and c, and h is the phonetic material of the head. Head movement can not move phonetic material other than that of the head of the tree, so there is no need to split the string parts of the other chains in an expression. An abstract example of the new notation is provided in 82 and 83 below.

82.



83. [*ac*, *e*, ϵ:x, *b*:-f$_1$ *d*:-f$_2$ *f*:-f$_3$ *gh*:-f$_4$]$_c$

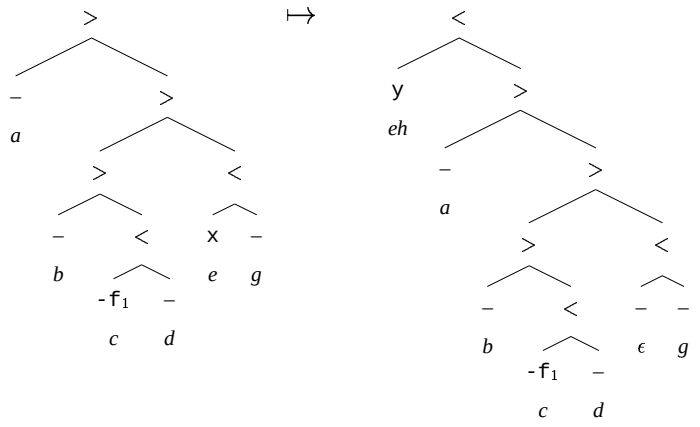
The reformulated structure building functions for head movement with left-adjunction, taken from [Sta01b], are given in 84 and 85.

84. $$\frac{[\epsilon, s, \epsilon{:}\mathsf{X}{=}\gamma]_s \qquad [t_s, t_h, t_c{:}\mathsf{x}, \alpha_1 \ldots, \alpha_k]_t}{[\epsilon, t_h s, t_s t_c{:}\gamma, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-la-1

85. $$\frac{[\epsilon, s, \epsilon{:}\mathsf{X}{=}\gamma]_s \qquad [t_s, t_h, t_c{:}\mathsf{x}\delta, \alpha_1 \ldots, \alpha_k]_t}{[\epsilon, t_h s, \epsilon{:}\gamma, t_s t_c{:}\delta, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-la-3

In the case of head movement with left adjunction, the string associated with the head of the selected expression must be prefixed to the string associated with the head of the selecting expression. The selected tree will form the complement of the resulting tree. Hence, if the selected tree will not move along, its phonetic material – except for the phonetic material of the head – will go in the complement position of the

resulting tree. This situation is covered by the function Merge-la-1 in 84. An example of an application of this function is given in 86 in terms of trees and in 87 in terms of collapsed trees.

86. x= y *h*, 

87. $[\epsilon, h, \epsilon$:X=y$]_s$, [*a, e, g*:x, *bcd*:-f$_1$]$_c$ $\mapsto$ $[\epsilon, eh, ag$:y, *bcd*:-f$_1$]$_c$

The function Merge-la-3 in 85 describes the case in which the selected tree will move along in a later derivation step. In this case the phonetic material in the specifier and complement position of the selected tree must be kept in a separate chain. By definition head movement can only occur if the selecting tree is simple. Hence the strings associated with the specifier and the complement of the selecting expressions in 84 and 85 are empty.

The functions Merge-ra-1 and Merge-ra-3 for left-adjunction, which are given in 88 and 89, are similar to the functions Merge-la-1 and Merge-la-3.

88. $$\frac{[\epsilon, s, \epsilon\text{:=X}\gamma]_s \qquad [t_s, t_h, t_c\text{:x}, \alpha_1, \ldots, \alpha_k]_t}{[\epsilon, st_h, t_st_c\text{:}\gamma, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-ra-1

89. $$\frac{[\epsilon, s, \epsilon\text{:=X}\gamma]_s \qquad [t_s, t_h, t_c\text{:x}\delta, \alpha_1 \ldots, \alpha_k]_t}{[\epsilon, st_h, \epsilon\text{:}\gamma, t_st_c\text{:}\delta, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-ra-3

Associating the first chain of an expression with three strings rather than one also requires minor changes in the functions Merge-1, Merge-2, Merge-3, Move-1, and Move-2. The revised versions of these functions follow below.

90. $$\frac{[\epsilon, s, \epsilon\text{:=x}\gamma]_s \qquad [t_s, t_h, t_c\text{:x}, \alpha_1, \ldots, \alpha_k]_t}{[\epsilon, s, t_s t_h t_c\text{:}\gamma, \alpha_1, \ldots, \alpha_k]_c}$$ Merge-1′

91. $$\frac{[s_s, s_h, s_c\text{:=x}\gamma, \alpha_1, \ldots, \alpha_k]_c \qquad [t_s, t_h, t_c\text{:x}, \iota_1, \ldots, \iota_l]_t}{[t_s t_h t_c s_s, s_h, s_c\text{:}\gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l]_c}$$ Merge-2′

92. $$\frac{[s_s, s_h, s_c\text{:=x}\gamma, \alpha_1, \ldots, \alpha_k]_t \qquad [t_s, t_h, t_c\text{:x}\delta, \iota_1, \ldots, \iota_l]_t}{[s_s, s_h, s_c\text{:}\gamma, \alpha_1, \ldots, \alpha_k, t_s t_h t_c\text{:}\delta, \iota_1, \ldots, \iota_l]_c}$$ Merge-3′

93. $$\frac{[s_s, s_h, s_c\text{:+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, t\text{:-f}, \alpha_{i+1}, \ldots, \alpha_k]_c}{[ts_s, s_h, s_c\text{:}\gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k]_c}$$ Move-1′

94. $$\frac{[s_s, s_h, s_c\text{:+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, t\text{:-f}\delta, \alpha_{i+1}, \ldots, \alpha_k]_c}{[s_s, s_h, s_c\text{:}\gamma, \alpha_1, \ldots, \alpha_{i-1}, t\text{:}\delta, \alpha_{i+1}, \ldots, \alpha_k]_c}$$ Move-2′

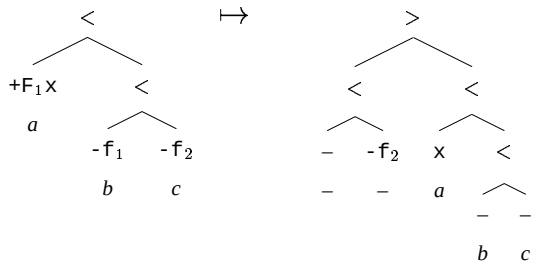
The new functions in this section are straightforwardly turned into rules of inference for the recognizer by replacing the strings with position vectors. It follows from the antecedents which position vectors are adjacent and which are not necessarily adjacent.

4.9.2 Covert Phrasal Movement

The application of the structure building function *move* in 96 shows that the collapsed tree notation does not contain enough information to describe covert movements. In order to determine the phonetic material that will be left behind by the covert move operation triggered by the pair of features $\texttt{+F}_1$, $\texttt{-f}_1$, it is necessary to know that the maximal projection of the node labeled $\texttt{-f}_1$ contains the maximal projection of the node labeled $\texttt{-f}_2$. Moreover, in order to establish that in the phonetic material that is

left behind *b* occurs to the left of *c*, it is necessary to know that the maximal projection of the node labeled -f$_2$ is a complement of the node labeled -f$_1$. Finally, in order to establish that the string *bc* is left behind in a position which is to the right of the head of the tree, it is necessary to know that the tree that moves covertly moves so from a complement position.[16]

96.



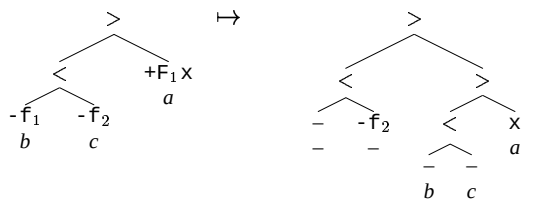
The collapsed tree notation introduced above suggests that the trees in 96 be represented by the expressions in 97.[17] However, it is clear from the preceding discussion that in 97 the expression on the left does not contain enough information to derive the expression on the right.

97. [*a*:+F$_1$x, *b*:-f$_1$, *c*:-f$_2$]$_c$ ↦ [*abc*:x, ϵ:-f$_2$]$_c$

In order to accommodate covert movement, the expressions in 97 have to be extended with a relation C, for immediate containment, and P, for immediate precedence. Given a tree τ and its corresponding expression e, a chain s:γ in e immediately contains another chain t:δ in e if in tree τ the maximal projection of the node with syntactic

[16] Compare the configuration in 96 with the following, linguistically rather bizarre situation:

95.

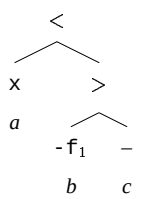


[17] To ignore unnecessary complications in the notation, head movement will be ignored in this section.

features δ is a complement or specifier of the node labeled with syntactic features γ. Thus, the relation C for the expression corresponding to the left-most tree in 96 is given by C = { $\langle a$:+$\mathtt{F}_1$x, b:-$\mathtt{f}_1\rangle$, $\langle b$:-$\mathtt{f}_1$, c:-$\mathtt{f}_2\rangle$ }. Let C* denote the transitive closure of C. For our simple case, C* { $\langle a$:+$\mathtt{F}_1$x, b:-$\mathtt{f}_1\rangle$, $\langle a$:+$\mathtt{F}_1$x, c:-$\mathtt{f}_2\rangle$ $\langle b$:-$\mathtt{f}_1$, c:-$\mathtt{f}_2\rangle$ }. A chain s:γ in e immediately precedes another chain t:δ in e if in the yield of tree τ the node labeled γ and the node labeled δ are not separated by another node with syntactic features and either 1) the maximal projection of the node labeled γ and the maximal projection of the node labeled δ are specifiers of the same head, and the node labeled γ precedes the node labeled δ in τ, or 2) the node labeled γ is the head of a specifier of the node labeled δ, or 3) the node labeled δ is the head of the complement of the node labeled γ. Relation P reflects the order specifier-head-complement; multiple specifiers of a projection are ordered from left to right as in the tree. For the left-most tree in 96, P = C. Now, since $\langle b$:-$\mathtt{f}_1$, c:-$\mathtt{f}_2\rangle \in$ C*, both *b* and *c* will be left behind by the covert movement operation in the example above. Since $\langle b$:-$\mathtt{f}_1$, c:-$\mathtt{f}_2\rangle \in$ P, *b* must be to the left of *c*. Since $\langle a$:+$\mathtt{F}_1$x, b:-$\mathtt{f}_1\rangle \in$ P, *a* must be to the left of *b*. Thus for the expression [a:+$\mathtt{F}_1$x, b:-$\mathtt{f}_1$, c:-$\mathtt{f}_2$]$_c$, C, P corresponding to the tree on the left in 96 we are able to derive an expression [*abc*:x, ϵ:-$\mathtt{f}_2$]$_c$, C′, P′ for the tree on the right in 96. The contents of C′ and P′ follow immediately from the contents of C and P definition of the structure building function $move$.

There is one further complication that needs to be addressed, concerning the manipulation of the string parts of the expressions involved in covert movement. Consider the tree in 98 below.

98.



Under current conventions, the expression for the tree in 98 will be [*ac*:x, *b*:-$\mathtt{f}_1$]$_c$. However, this does not accommodate the case in which the node -$\mathtt{f}_1$*b* will move covertly. In that case, *b* should go inbetween *a* and *c*, which is impossible if *a* and *c* have already been concatenated.

There are at least two possible approaches to resolving this issue. In the approach taken by Michaelis ([Mic98]), all movement operations are anticipated, that is to say, every licensee feature is marked with an index indicating whether the feature will be used to trigger an overt move or a covert move.[18] The functions for overt and covert movement are restricted to apply only to expressions with features that are marked with the appropriate index. So, for example, if the licensee feature -$\mathtt{f}_1$ will trigger a covert movement, the expression corresponding to the tree in 98 will be as in 99. The expression in 100 is for the situation in which -$\mathtt{f}_1$ will trigger an overt move.

99. [*abc*:x, ϵ:-$\mathtt{f}_1$(covert)]$_c$, C, P

100. [*ac*:x, *b*:-$\mathtt{f}_1$(overt)]$_c$, C′, P′

This approach is worked out in complete detail in [Mic98].

Another approach would be to postpone concatenation of two strings until all intervening phonetic material has actually either moved overtly or been left behind by covert movement. The order of concatenation will be recorded in the relations C and P. For example, in this approach the tree in 98 will be represented by the expression in 101, where C = { ⟨*a*:x, *c*:–⟩, ⟨*c*:–, *b*:-$\mathtt{f}_1$⟩ } and P = { ⟨*a*:x, *c*:–⟩, ⟨*b*:-$\mathtt{f}_1$, *c*:–⟩ }.

101. [*a*:x, *b*:-$\mathtt{f}_1$, *c*:–]$_c$, C, P

[18] Michaelis use a similar scheme to distinguish merge operations with head movement from merge operations without head movement.

Now assume that the tree in 98 is selected by the simple tree $=\mathtt{x}+\mathtt{F}_1\mathtt{y}$. Then node $-\mathtt{f}_1 b$ will move covertly, leaving behind *b*. In terms of collapsed trees, since in 101 chain *a*:x precedes and contains chain *c*:–, and chain *c*:– contains and is preceded by chain $-\mathtt{f}_1 b$, in the expression resulting from the covert movement operation *a* will precede *b*, which will precede c. Since these strings are not separated by any node with an outstanding licensee feature, they can be concatenated. Hence, the resulting expression will be $[abc{:}\mathtt{y}]_c$, C′, P′. If, however, the tree in 98 is selected by a tree $=\mathtt{x}+\mathtt{f}_1\mathtt{y}$, the node $-\mathtt{f}_1 b$ will move overtly, taking *b* with it. In that case, since movement is to a specifier position, the resulting string will be *bac*.

For both approaches it is possible to convert the functions into rules of inference for the parser. However, the functions provided in [Mic98] are very complex. The second approach sketched above appears to be less complicated, but it has not been completely worked out. Therefore we cannot be certain at this point that it is sound and complete with regard to the original definition of Minimalist Grammars.

4.10 Conclusion

In this chapter we have provided a formal specification of an agenda-driven, chart-based, bottom-up recognizer for languages generated by Minimalist Grammars without head movement and covert phrasal movement. The recognizer uses a succinct reformulation of Minimalist Grammars that does not refer to full trees. As evident from the example in section 4.2, the recognizer generates many axioms which potentially combine into items that will never lead to a goal item.[19] Thus the bottom-up recognizer cannot have the correct-prefix property. In the next chapter we will turn our

[19] In Context-Free Grammars, only adjacent constituents can combine into larger constituents. Because of their discontinuous character, there is no such restriction for Minimalist Grammars, making the problem of spurious items even more acute.

attention to a top-down recognizer for Minimalist Languages, which does not have this problem.

When the recognizer described here is confronted with a choice regarding what axiom to use for a word in the input sentence or what rule of inference to apply to an item, it will simply pursue all possibilities and store the results in the chart. Some of these choices will produce items that are not involved in the derivation of a goal item. These mistakes may be avoided by looking at the words that follow in a sentence. For example, after having heard *after Mary left the store* there is a choice between treating the determiner phrase *the store* as the object of the verb *to leave* or treating *the store* as the subject of a matrix clause that follows. If the sentence continues with *she went home*, the first choice is the right one; if the sentence continues with *closed*, the second.[20] It is still an open question whether there is a useful way to characterize Minimalist Grammars that generate languages such that k words of look-ahead suffice to make bottom-up recognition deterministic, i.e. for any input sentence the number of choices facing the recognizer at each choice point is at most one.

[20] This is not to say that this is the way the human sentence processor deals with sentences of this kind, nor that local ambiguities in English can generally be resolved by a look-ahead mechanism.

CHAPTER 5

Parsing Minimalist Languages Top-Down

In this chapter we will describe a method for top-down recognition of languages defined by Minimalist Grammars [Sta97]. Minimalist Grammars are a rigorous formalization of the kind of grammars proposed in the linguistic framework of Chomsky's Minimalist Program [Cho95]. The grammar formalism will be briefly reviewed in section 5.1. Even though Minimalist Grammars are more powerful than Context-Free Grammars, derivations in this formalism are context-free. This property of Minimalist Grammars will be explained in section 5.2

In the previous chapter, a chart-based bottom-up recognizer for Minimalist Grammars was presented. A bottom-up parser has no predictive power. Hence, the chart will contain numerous items that are not involved in the derivation of the sentence to be parsed. In particular, in order to be complete, a bottom-up parser has to assume that any phonetically empty category can intervene between any two adjacent words in a sentence. A top-down parser, however, has the ability to predict, based on the structure built so far, where in the structure of a sentence it is necessary to assume the presence of phonetically empty categories. Phonetically empty functional projections are ubiquitous in present-day transformational theories of language, so this issue is an important motivation for the formulation of a top-down recognizer for Minimalist Grammars. The design of the top-down recognizer will be discussed in section 5.3. The accuracy of the predictions can be improved by adding look-ahead. This will be addressed in section 5.4.

Another goal of the current chapter is to design a parser for Minimalist Grammars which has the correct-prefix property. A parser has the correct-prefix property if it goes through the input sentence from left to right and, in case of an ungrammatical input sentence, will halt at the first word of the sentence that does not fit into a grammatical structure, i.e., the parser will not parse a prefix that cannot be extended to a grammatical sentence in the language, e.g. [SS88], [Ned99]. Because of its lack of predictive power, the bottom-up recognizer described in the previous chapter does not have the correct-prefix property – the top-down recognizer presented here does. The additions necessary to obtain the correct prefix property are discussed in section 5.5. A parser with the correct-prefix property is computationally advantageous, because it will not spend any effort on trying to complete a parse for a sentence once it is known that the sentence is ungrammatical. Also, the parser of the human sentence processor seems to have the correct-prefix property: the ungrammaticality of a sentence is detected at the first word that makes the sentence ungrammatical, and for garden path sentences the human parser will generally hesitate at the first word that does not fit into the structure that is hypothesized for the sentence.

This chapter closes with proofs of correctness for the top-down recognizer and some concluding remarks.

5.1 Minimalist Grammars

In this section we will present the Minimalist Grammar formalism, following [Sta97]. A Minimalist Grammar defines a set of trees. These trees are derived by closing the lexicon, which is a set of trees itself, under two structure building functions, *merge* and *move*. Consider for example Minimalist Grammar G_1 with a lexicon containing the following 6 elements:

1. d -k *Lavinia*

2. d -k *Titus*

3. =d vt -v *praise*

4. =pred +v +k i *s*

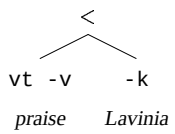
5. =i c ϵ

6. =vt +k =d pred ϵ

Each lexical item is a 1-node tree labeled with syntactic features of various kinds and a phonetic form. The last two lexical items are phonetically empty.

The features determine how the structure building functions will apply to the lexical items and the trees derived from these. Two trees, one of which has a feature =x and the other of which has a feature x, will trigger an application of the structure building function *merge*. For example, the lexical items =d vt -v *praise* and d -k *Lavinia* will merge to form the tree below:

7.

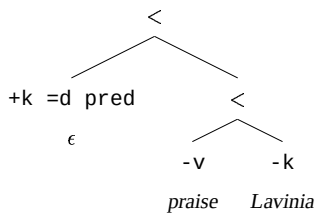


The resulting tree includes the original expressions =d vt -v *praise* and d -k *Lavinia*, minus the pair of features =d and d that triggered the function *merge*. These features have been checked and deleted. The '<' points to the head of the tree. The features of the head of the tree determine what other structure building functions will apply. The order of the features matters: the application of the structure building functions is triggered by the left-most features of the sequences of features of the expressions involved. Thus, *merge* does not apply to the lexical items =vt +k =d

pred ϵ and =d vt -v *praise*, because the feature vt is not left-most in the lexical item =d vt -v *praise*.

A tree that consists of more than one node, such as the tree in 7, is called a complex tree. Lexical items are simple trees, as they consist of just one node. We will use the following notation for complex trees: $[_{<}\tau, \upsilon]$ denotes a complex tree with immediate subtrees τ and υ, τ preceding υ, whose head is the head of τ; similarly, $[_{>}\tau, \upsilon]$ denotes a complex tree with immediate subtrees τ and υ, τ preceding υ, whose head is the head of υ. (For example, in this notation, the tree in 7 would be written as $[_{<}$vt -v *praise*, -k *Lavinia*].) The head of a simple tree is the single node making up the simple tree. The structure building function *merge* is defined in the following way. A pair of trees τ, υ is in the domain of *merge* if the left-most feature of the head of τ is =x and the left-most feature of the head of υ is x. Then *merge*(τ, υ) = $[_{<}\tau', \upsilon']$ if τ is simple, and *merge*(τ, υ) = $[_{>}\upsilon', \tau']$ if τ is complex, where τ' is like τ except that feature =x is deleted, and υ' is like υ except that feature x is deleted. So simple trees take sisters to their right, and complex trees take sisters to their left.

8.

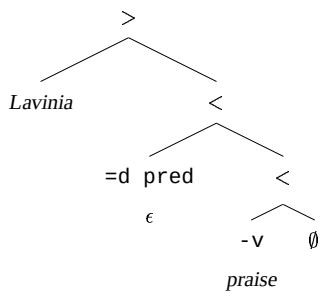


The derivation will continue with merging the lexical item =vt +k =d pred ϵ and the tree in 7. The result is given in 8.

The left-most feature of the head of the tree in 8 is +k. Furthermore, the tree contains a node whose left-most feature is -k. In this situation, the structure building function *move* will apply. It will move the maximal subtree whose head has the feature -k to the specifier position of the original tree, as in 9 below. A subtree ϕ is maximal

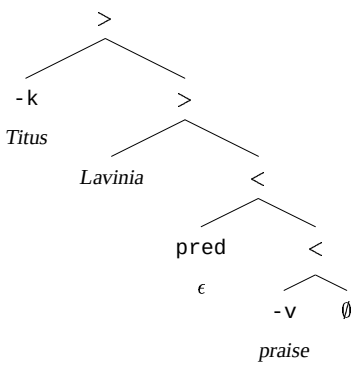
if any subtree τ properly containing ϕ has a head other than ϕ. As in the case of *merge*, the features triggering the application of *move* are checked and deleted.

9.



Formally, the structure building function *move* is defined as follows. A tree τ is in the domain of *move* if the left-most feature of the head of τ is `+y` and τ has exactly one maximal subtree τ_0 the left-most feature of the head of which is `-y`. Then *move*(τ) = [$_{>}\tau_0'$, τ'], where τ_0' is like τ_0 except that feature `-y` is deleted, and τ' is like τ except that feature `+y` is deleted and subtree τ_0 is replaced by a single node without features.

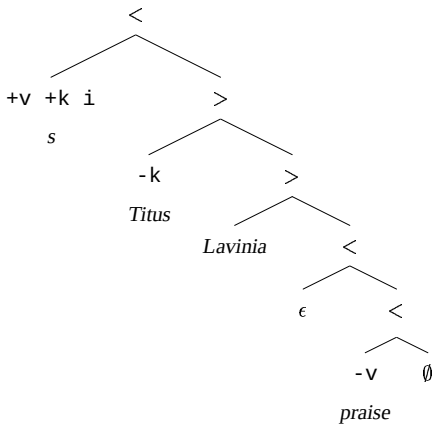
10.



The left-most feature of its head being `=d`, the tree in 9 will merge with the lexical item `d -k` *Titus.* Because the tree in 9 is a complex tree, the second clause of the definition of *merge* will apply. The result is the tree in 10.

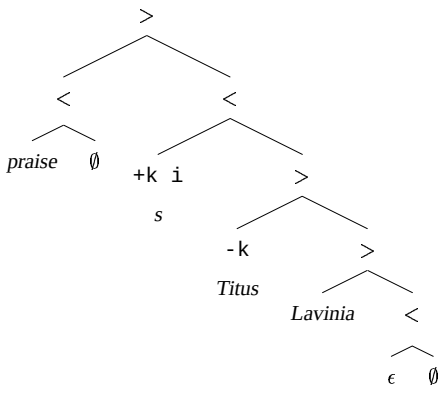
The tree in 10 will be merged with the lexical item `=pred +v +k i` *s.* This will produce the tree in 11.

11.



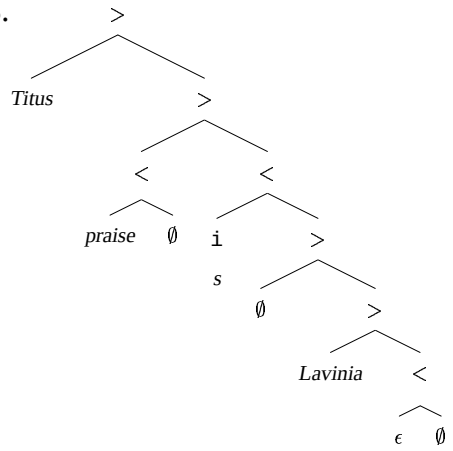
The next step of the derivation is another *move*, triggered by the feature `+v` on the head of the tree in 11 and the feature `-v` on the head of one of its subtrees. The tree in 12 is the result.

12.



The pair of features `+k` and `-k` in the tree in 12 will trigger another application of *move*, which will yield the tree in 13.

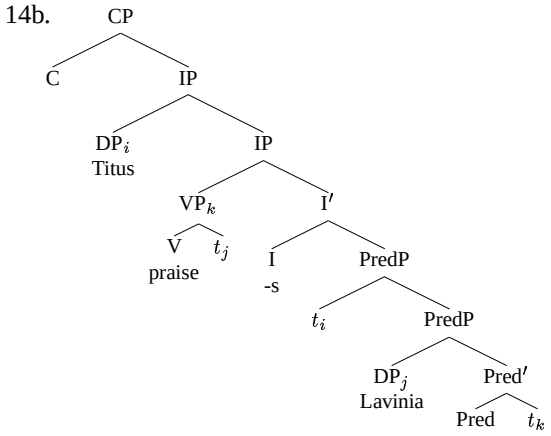
13.



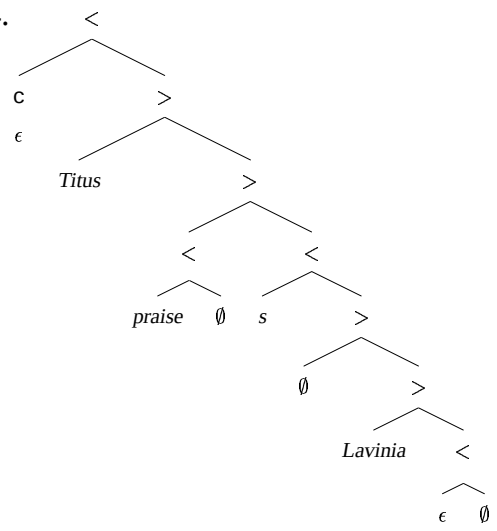
Finally, the tree in 13, whose head has the feature `i`, will be selected by the lexical item `=i c` ϵ. The result of this final *merge* is the tree in 14. This tree contains only one unchecked feature, namely `c`, and this feature labels the head of the tree. We have thus established that the yield of the tree in 14, ϵ *Titus praise s Lavinia* ϵ, i.e., *Titus praises Lavinia*, is a string of category `c`, using an analysis along the lines of [Mah00], cf. [SK00].[1]

[1] In traditional notation, the tree in 14 would be presented as in 14b.

14b.



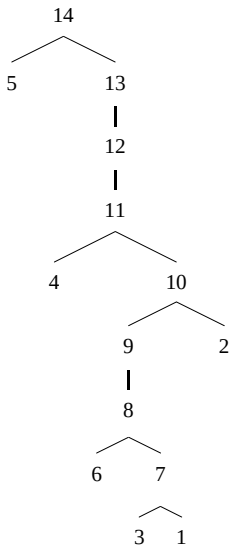
14.



The total number of features occurring in the lexical items involved in the derivation of the sentence *Titus praises Lavinia* is 17. Therefore, the derivation of the tree in 14 takes exactly 8 steps, for each application of *merge* and *move* removes 2 features and the tree in 14 has one single feature left.

The derivation can be summarized in the derivation tree given in 15. The numbered nodes stand for the trees given above, including the lexical items.

15.



Formally, the language derivable by a Minimalist Grammar G consists of the yields of the complete trees in the closure of the lexicon under the structure building functions *merge* and *move*, where a complete tree is a tree without syntactic features, except for the distinguished feature `c`, which has to label the head of the tree, and the yield of a tree is the concatenation of the phonetic forms appearing at the leaves of the tree, ordered as in the tree.

All derivations in a Minimalist Grammar are subject to the Shortest Movement Constraint, which is built in into the definition of the domain of the structure building functions *move*: this function does not apply to a tree if the left-most feature of the head of the tree is `+f` and the tree contains more than one subtree whose head begins with the feature `-f`. In this case all subtrees want to move to the same position, but moving any one subtree will deprive the other subtrees of their "shortest move", as they will now have to move to the specifier of some higher head which has the feature `+f`.

5.2 Context-Free Derivations

Michaelis ([Mic98]) has shown that the class of languages defined by Minimalist Grammars is included in the class of languages defined by Multiple Context-Free Grammars ([SHF91]) by demonstrating how to construct a Multiple Context-Free Grammar G′ which defines the same language as a given Minimalist Grammar G. For every tree τ generated by grammar G, there will be a non-terminal symbol A in grammar G′ which encodes the purely syntactic properties of τ, that is, its behavior with regard to the structure building functions *merge* and *move*. These properties are completely determined by the syntactic features appearing in the tree and whether the tree is simple or complex. The phonetic forms appearing in a tree are not relevant for the applicability of *merge* and *move*, nor does the geometry of the tree matter, except for

the fact that the syntactic features of the head of the tree should be distinguished from syntactic features appearing at other nodes in the tree. Hence, for syntactic purposes, a tree may be collapsed into a sequence of sequences of syntactic features. These are the non-terminal symbols of grammar G′, which will be referred to as categories.

Formally, a category is a sequence of the form $[\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t$, with γ_j, $\delta_j \in$ Cat*, $0 \leq j \leq n$, t $\in \{c, s, l\}$, and Cat the set of syntactic features of G.[2] A category A represents a set of trees T_c(A). A tree τ is in $T_c([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t)$ if, and only if, the following four conditions are met:

1. If $t = s$, τ is a simple tree; if $t = c$, τ is a complex tree; and if $t = l$, τ is a lexical tree.[3]

2. For every i, $0 \leq i \leq n$, there is a leaf l_i in τ such that:

   (a) The syntactic features of l_i are δ_i.

   (b) The maximal subtree headed by l_i is a projection of a lexical item with features $\gamma_i \delta_i$.[4]

3. Leaf l_0 is the head of τ.

4. Besides the leaves $l_0, \ldots, l_n$, there are no other leaves with syntactic features in τ.

[2] Michaelis ([Mic98]) and Harkema ([Har00]), see the previous chapter of this dissertation, do not consider categories with dots. The use of dots allows for a finer distinction between categories, which will be advantageous when look-ahead is added to the top-down recognizer.

[3] All lexical trees are assumed to be simple trees, but not all simple trees are lexical trees. For example, the tree d - k ϵ is a simple tree, but it is not a lexical tree in grammar G_1 discussed in section 5.1.

[4] Any lexical item ℓ is a projection of itself. If τ is a projection of ℓ and τ, υ are in the domain of *merge*, then ϕ = *merge*(τ, υ) is a projection of ℓ. If τ is a projection of ℓ and τ is in the domain of *move*, then ϕ = *move*(τ) is a projection of ℓ.

One can think of a category as an abbreviation of a possibly infinite set of trees. We will only be interested in relevant categories, that is, categories A for which $T_c(A)$ includes a tree generated by grammar G which does not violate the Shortest Movement Constraint. Any tree τ that violates the Shortest Movement Constraint is useless in that it is not a complete tree by itself and cannot participate in the derivation of a complete tree. The structure building function *move* does not apply to such a tree τ, wherefore its outstanding `-f` features cannot be deleted. The crucial observation in [Mic98] is that the set of relevant categories for grammar G is finite. This is essential for grammar G′ to be definable at all, since a Multiple Context-Free Grammar cannot have an infinite number of non-terminal symbols or categories.

With regard to the phonetic forms of the trees generated by G, any category A in G′ will be associated with a tuple of strings, rather than with just one string, as in a plain Context-Free Grammar. For a category A corresponding to a tree τ generated by G, this tuple of strings includes all and only the phonetic forms appearing in τ. Phonetic forms that will move independently from one another will be kept separate. In order to formalize this, let the narrow yield Y_n of a tree be defined as follows. The narrow yield of a simple tree ϕ equals its yield as defined above. If ϕ is a complex tree, then, in case $\phi = [_{>}\tau, \upsilon]$, $Y_n(\phi) = Y_n(\tau)^\frown Y_n(\upsilon)$ if the left-most feature of the head of τ is not of the kind `-f`, and $Y_n(\phi) = Y_n(\upsilon)$ otherwise.[5] If $\phi = [_{<}\tau, \upsilon]$, then $Y_n(\phi) = Y_n(\tau)^\frown Y_n(\upsilon)$ if the left-most feature of the head of υ is not of the kind `-f`, and $Y_n(\phi) = Y_n(\tau)$ otherwise. Then, a category $A = [\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t$ as defined above will be associated with a tuple of strings $(s_0, \ldots, s_n)$ if, and only if, there is a tree $\tau \in T_c(A)$ for which the narrow yield of the maximal projection of leaf l_i in τ labeled with syntactic features δ_i is s_i, $0 \leq i \leq n$.

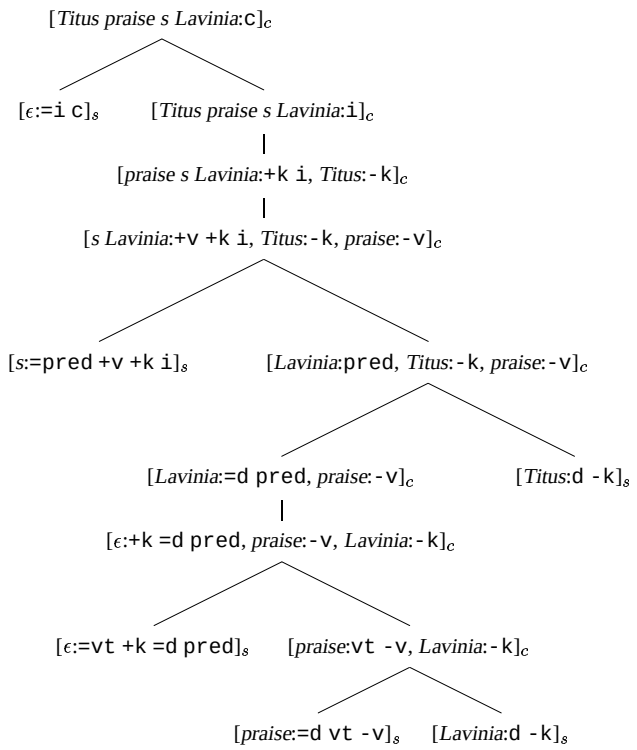
In order to derive the categories and their associated tuples of strings, grammar

[5] For two strings p and q, $p^\frown q$ denotes their concatenation in the obvious order.

G′ contains a set of context-free rewrite rules which mirror the effects of the structure building operations *merge* and *move* in grammar G. Since the set of relevant categories is finite, a finite set of rewrite rules suffices to describe all derivations.

For example, for G_1 the grammar discussed in the previous section, grammar G′ will allow us to derive a category $[\texttt{=vt +k =d·pred}, \texttt{d·-k}, \texttt{=d vt·-v}]_c$, which will be associated with the triple (*Lavinia* ϵ, *Titus*, *praise*) = (*Lavinia*, *Titus*, *praise*). The tree given in 10 is an element of $T_c([\texttt{=vt +k =d·pred}, \texttt{d·-k}, \texttt{=d vt·-v}]_c)$. We can also derive a category $[\texttt{·=pred +v +k i}]_s$, associated with the 1-tuple (*s*). The tree in 4 is an element of $T_c([\texttt{·=pred +v +k i}]_s)$. We can conveniently combine a category and its associated tuple of phonetic forms and write [*Lavinia*:=vt +k =d·pred, *Titus*:d·-k, *praise*:=d vt·-v]$_c$ and [*s*:·=pred +v +k i]$_s$ for the two examples given above. The context-free rewrite rule [p⌢q:=pred·+v +k i, r:d·-k, t:=d vt·-v] → [p:·=pred +v +k i] [q:=vt +k =d·pred, r:d·-k, t:=d vt·-v] mirrors the step in the derivation in which the tree in 11 is derived from the tree in 4 and the tree in 10. The rule will derive the category with phonetic forms [*s Lavinia*:=pred·+v +k i, *Titus*:d·-k, *praise*:=d vt·-v]$_c$ from [*s*:·=pred +v +k i]$_s$ and [*Lavinia*:=vt +k =d·pred, *Titus*:d·-k, *praise*:=d vt·-v]$_c$. The tree in 11 is an element of $T_c([\texttt{=pred·+v +k i}, \texttt{d·-k}, \texttt{=d vt·-v}]_c)$. Using categories to abbreviate trees, the derivation tree in 15, whose nodes are trees generated by grammar G_1, can be represented as the tree in 16.

16.



The translation of a Minimalist Grammar G into an equivalent Multiple Context-Free Grammar G′ reveals that the derivation trees of Minimalist Grammars are context-free, even though the languages defined by Minimalist Grammars are beyond the power of Context-Free Grammars. The next section will show how this insight can be used to obtain a transparent top-down recognizer for languages generated by Minimalist Grammars.

5.3 Top-Down Recognition

The top-down recognizer for Minimalist Grammars that will be presented in this section, like its bottom-up counterpart in the previous chapter, is based on the principle of parsing as deduction as described in [SSP95]. The recognizer uses a chart to store items, which embody predictions about the syntactic structure of the sentence to be

recognized. Initially, the chart will be filled with a set of axioms. These axioms are then closed under a set of rules of inference. If the closure contains a distinguished goal item, the sentence is in the language defined by the grammar, otherwise it is not, assuming that the deductive system is sound and complete.

5.3.1 Items and Invariant

An item is a sequence Δ_1 + ...+ Δ_m, where each subitem Δ_i, $1 \leq i \leq m$, is a prediction regarding the syntactic features and narrow yields of a tree involved in the derivation of the input sentence. A subitem is a category with position vectors and is of the general form [(x_0, y_0):$\gamma_0 \cdot \delta_0$, ..., (x_n, y_n):$\gamma_n \cdot \delta_n$]$_t$, with γ_j, δ_j, and t as above, and x_j, $y_j \in \mathbb{N}$, $0 \leq j \leq n$. A subitem Δ abbreviates a set of trees $T_s(\Delta)$. For an input string $w = w_1 \ldots w_k$ and a subitem Δ = [(x_0, y_0):$\gamma_0 \cdot \delta_0$, ..., (x_n, y_n):$\gamma_n \cdot \delta_n$]$_t$, tree a τ is in $T_s(\Delta)$ if, and only if, the following holds:

1. $\tau \in T_c([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t)$.
2. The narrow yield of the maximal projection of leaf l_i in τ labeled δ_i is $w_{x_i+1} \ldots w_{y_i}$, $0 \leq i \leq n$.

On the assumption that sentences are strings of the distinguished category C, the items generated by the recognizer will be interpreted under the following invariant: given an item Δ_1 + ...+ Δ_m, the sequence of trees τ_1, ..., τ_m, for arbitrary $\tau_i \in T_s(\Delta_i)$, $0 \leq i \leq m$, is a cut of a partial derivation tree whose root is a complete tree ρ of category C with yield w.[6,7] Recall that the nodes of a derivation tree are labeled with trees generated by a Minimalist Grammar (cf. the derivation tree in 15).

[6] A partial derivation tree is a derivation tree whose root is a complete tree, but the leaves of a partial derivation tree are not necessarily lexical items.

[7] A set C of nodes of a tree T is a cut of tree T if none of the nodes in C dominates another node in C and if every node in T and not in C either dominates or is dominated by a node in C.

5.3.2 Axioms

If c is the distinguished feature of grammar G and k is the length of the input string w, then for any lexical item ℓ with the single syntactic feature c there will be an axiom [(0, k):·c]$_s$, and for any lexical item ℓ with syntactic features γc, there will be an axiom [(0, k):γ·c]$_c$, $\gamma \in \text{Cat}^+$.

5.3.3 Rules of Inference

The rules of inference encode the context-free rewrite rules of the Multiple Context-Free Grammar G′ equivalent to Minimalist Grammar G. Grammar G′ contains a rewrite rule for every single application of the structure building functions *merge* and *move*, but the definitions of these functions allow us to condense these rewrite rules into five rules of inference: three Unmerge rules and two Unmove rules. Given a particular subitem Δ_i appearing in some item Δ_1 + … + Δ_m, the rules of inference will predict how the trees in T$_s$(Δ_i) can be 'unmerged' and 'unmoved' into smaller trees by specifying the particular subitems that these smaller trees belong to. In addition to the Unmerge and Unmove rules, there is also a Scan rule for reading the input string.

Unmerge-1: given an item Δ_1 + … + Δ_m such that Δ_i = [(p, q):=x·γ, S]$_c$, $1 \leq i \leq m$, then, for any lexical item with syntactic features βx, add the items Δ_1 + … + Δ_{i-1} + [(p, v):·=xγ]$_s$ + [(v, q):β·x, S]$_t$ + Δ_{i+1} + … + Δ_m to the chart, for all possible values of v such that $p \leq v \leq q$; for all possible values of t such that t = s if $\beta = \emptyset$, t = c if $\beta \neq \emptyset$, and if t = s then S = $\emptyset$.

Unmerge-2: given an item Δ_1 + … + Δ_m such that Δ_i = [(p, q):α=x·γ, S]$_c$, $1 \leq i \leq m$, $\alpha \neq \emptyset$, then, for any lexical item with syntactic features βx, add the items Δ_1 + … + Δ_{i-1} + [(v, q):α·=xγ, U]$_c$ + [(p, v):β·x, V]$_t$ + Δ_{i+1} + … + Δ_m to the chart, for all possible values of v such that $p \leq v \leq q$; for all possible values of U, V such that U

$\cup$ V = S; for all possible values of t such that t = *s* if $\beta = \emptyset$, t = *c* if $\beta \neq \emptyset$, and if t = *s* then V = $\emptyset$.

Unmerge-3: given an item Δ_1 + ... + Δ_m, such that Δ_i = [(p, q):α=x·γ, S, (v, w):βx·δ, T]$_c$, $1 \leq i \leq m$, then add the items Δ_1 + ... + Δ_{i-1} + [(p, q):α·=xγ, U]$_{t_1}$ + [(v, w):β·xδ, V]$_{t_2}$ + Δ_{i+1} + ... + Δ_m to the chart, for all possible values of U, V such that U $\cup$ V = S $\cup$ T; for all possible values of t_1 such that t_1 = *s* if $\alpha = \emptyset$, t_1 = *c* if $\alpha \neq \emptyset$, and if t_1 = *s* then U = $\emptyset$; for all possible values of t_2 such that t_2 = *s* if $\beta = \emptyset$, t_2 = *c* if $\beta \neq \emptyset$, and if t_2 = *s* then V = $\emptyset$.

Unmove-1: given an item Δ_1 + ... + Δ_m, such that Δ_i = [(p, q):α+y·γ, S]$_c$, $1 \leq i \leq n$, $\alpha \neq \emptyset$, then, for any lexical item with syntactic features β-y, $\beta \neq \emptyset$, add the items Δ_1 + ... + Δ_{i-1} + [(v, q):α·+yγ, (p, v):β·-y, S]$_c$ + Δ_{i+1} + ... + Δ_m to the chart, for all possible values of v such that $p \leq v \leq q$.

Unmove-2: given an item Δ_1 + ... + Δ_m, such that Δ_i = [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$, $1 \leq i \leq m$, $\alpha \neq \emptyset$, $\beta \neq \emptyset$, add the items Δ_1 + ... + Δ_{i-1} + [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$ + Δ_{i+1} + ... + Δ_m to the chart.

Scan: given an item Δ_1 + ... + Δ_m, such that Δ_i = [(p, q):·γ]$_s$, $1 \leq i \leq m$, then, if there is a lexical item ℓ with syntactic features γ and phonetic features covering $w_{p+1} \ldots w_q$ of the input string, i.e. $\ell \in$ T$_s$(Δ_i), add the following item to the chart: Δ_1 + ... + Δ_{i-1} + [(p, q):·γ]$_l$ + Δ_{i+1} + ... + Δ_m.

The rules of inference Unmove-1 and Unmove-2 come with the restriction that no items violating the Shortest Movement Constraint should be added to the chart. An item violates the Shortest Movement Constraint if it contains a subitem [(x_0, y_0):γ_0·δ_0, ..., (x_n, y_n):γ_n·δ_n]$_t$ such that there are δ_i, δ_j whose left-most features are the same feature -y, $1 \leq i < j \leq n$.

5.3.4 Goal Items

Any item of the form [(x_0, y_0):·γ_0]$_l$ + … + [(x_m, y_m):·γ_m]$_l$ is a goal item.

It follows from the definition of the Scan rule that there are lexical items $\ell_i \in$ T_s([(x_i, y_i):·γ_i]$_l$), $0 \leq i \leq m$. According to the invariant, the sequence ℓ_1,…, ℓ_m is a cut of a partial derivation tree whose root is a complete tree ρ of category `c` with yield w. Since all ℓ_i, $0 \leq i \leq m$, are lexical items, the derivation tree is in fact a complete derivation tree, which means that w is in the language defined by G.

5.3.5 Example

The recognition of the example sentence w = $_0$*Titus*$_1$*praise*$_2$*s*$_3$*Lavinia*$_4$ according to grammar G_1 given in section 5.1 will start with the assertion of the single axiom [(0, 4):`=i·c`]$_c$. This item represents a cut through the root of the derivation tree in 15. Of course, for each cut of the tree in 15, there is a corresponding cut of the derivation tree in 16.

Rule Unmerge-1 will apply to the axiom, as there is a lexical item with features `=pred +v +k i`. One of the items produced is [(0, 0):·`=i c`]$_s$ + [(0, 4):`=pred +v +k·i`]$_c$. This item corresponds to a cut through the nodes labeled 5 and 13 in derivation tree in 15. Rule Unmerge-1 also produces four other items: [(0, v):·`=i c`]$_s$ + [(v, 4):`=pred +v +k·i`]$_c$, $1 \leq v \leq 4$. These four additional items also obey the invariant defined in section 5.3.1, but they do not correspond to a cut of the particular derivation tree in 15.

The item [(0, 0):·`=i c`]$_s$ + [(0, 4):`=pred +v +k·i`]$_c$ can be rewritten in two ways. Either the Scan rule will apply to produce the item [(0, 0):·`=i c`]$_l$ + [(0, 4):`=pred +v +k·i`]$_c$, or rule Unmove-1 will apply to produce the item [(0, 0):·`=i c`]$_s$ + [(1, 4):`=pred +v·+k i`, (0, 1):`d·-k`]$_c$ (plus another four items [(0, 0):·`=i c`]$_s$ + [(v′,

4):=pred +v·+k i, (0, v′):d·-k]$_c$, v' = 0 or 2 $\leq v' \leq$ 4). Item [(0, 0):·=i c]$_s$ + [(0, 4):=pred +v·+k i (0, 0):d·-k]$_c$ corresponds to a cut through the nodes labeled 5 and 12 of the tree in 15. Rule Unmove-1 will also apply to the other four items produced in the previous step ($1 \leq v \leq 4$), but Scan will not.

Eventually, the item [(0, 0):·=i c]$_l$ + [(2, 3):·=pred +v +k i]$_l$ + [(4, 4):·=vt +k =d pred]$_l$ + [(1, 2):·=d vt -v]$_l$ + [(3, 4):·d -k]$_l$ + [(0, 1):·d -k]$_l$ will be deduced. This is a goal item, corresponding to a cut through the leaves of the derivation tree in 15.

5.3.6 Correctness and Complexity

The recognizer presented above is sound and complete: every deducible item respects the invariant given in section 5.3.1, and for every string that is in the language defined by the grammar, there is a deduction of a goal item from the axioms. The correctness proofs can be found in section 5.6. The time complexity of the recognizer is polynomial in the length of the input sentence (considering the shortest deduction of a goal item for a sentence).

Unfortunately, since items produced by the recognizer correspond to cuts of derivation trees, the recognizer will fail to halt if the grammar allows infinitely ambiguous sentences, which, given the many abstract and phonetically empty elements posited by linguists, are very likely to exist in natural language. An abstract example is the grammar with the two lexical items c ϵ and =c c ϵ. For any input string, the recognizer for this grammar will produce an infinite number of items, because the rule of inference Unmerge-1 will apply to its own output.[8] Top-down parsers are also generally stumped by left-recursive grammars – see the discussion in section 6.1 of the next chapter.

[8] The recognizer specified above will also not stop when parsing any sentence according to the grammar with the two lexical items c *a* and =c c *b* which does not generate infinitely ambiguous sentences, but this problem can be solved by adding look-ahead to the recognizer.

5.4 Look-Ahead

The example in section 5.3.5 illustrates that the recognizer will produce items which do not represent cuts of a derivation tree for the sentence to be recognized. These superfluous items occur because the rules of inference will split the position vectors of subitems in every possible way. Another situation in which spurious items are deduced is when a category can be derived in more than one way. Consider for example grammar G_2, which is grammar G_1 of section 5.1 with two additional lexical items: `=i +wh c` ϵ and `d -k -wh` *who*. The language defined by G_2 will not only contain the declarative sentences defined by G_1, but also interrogative sentences such as *who Titus praise s*.[9] For grammar G_2 and any given sentence, the recognizer presented in the previous section will deduce items for either type of sentence, even though one can tell from looking at the first word of the sentence whether it is declarative or interrogative. In the remainder of this section we will formally develop a method of look-ahead for Minimalist Grammars, which will restrict the number of items produced by the inference rules without compromising the completeness of the recognizer.

5.4.1 First$_k$

Given a Minimalist Grammar G and a relevant category A = $[\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t$, the set First$_k$(A) is defined as follows: $(s_0, \ldots, s_n) \in$ First$_k$(A) if, and only if, there is a tree $\tau \in$ T$_c$(A) and for each leaf l_i in τ labeled with syntactic features δ_i, the first k symbols of the narrow yield of the maximal projection of l_i is the string s_i, $0 \leq i \leq n$.

We can compute the sets First$_k$ using a slightly adapted version of the bottom-up recognizer from the previous chapter. The items of the bottom-up recognizer are very similar to the subitems in the top-down method defined above: they are sequences of

[9] Grammar G_2 ignores *do*-support. See [Sta01b] for a more complete Minimalist Grammar covering question formation in English.

the form [(x_0, y_0):$\gamma_0 \cdot \delta_0$, ..., (x_n, y_n):$\gamma_n \cdot \delta_n$]$_t$. To compute First$_k$, the position vectors (x_i, y_i) will be replaced by strings s_i. A revised bottom-up item [$\gamma_0 \cdot \delta_0 / s_0$, ..., $\gamma_n \cdot \delta_n / s_n$]$_t$ is understood to assert the existence of a tree τ generated by G such that:

1. $\tau \in T_c([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t)$.

2. For each leaf l_i in τ labeled with syntactic features δ_i, the first k symbols of the narrow yield of the maximal projection of l_i is the string s_i, $0 \leq i \leq n$.

Thus, an item [$\gamma_0 \cdot \delta_0 / s_0$, ..., $\gamma_n \cdot \delta_n / s_n$]$_t$ claims that (s_0, ..., s_n) $\in$ First$_k$([$\gamma_0 \cdot \delta_0$, ..., $\gamma_n \cdot \delta_n$]$_t$).

The revised bottom-up recognizer will start with the following set of axioms: for each lexical item ℓ of G labeled with syntactic features $\gamma \in \text{Cat}^+$ and phonetic form p, there will be an axiom [$\cdot \gamma$/k:p]$_s$, where $k : s$ denotes a prefix of length k of string s; if $|s| < k$, $k : s = s$. The axioms will be closed under the following five rules of inference:

- Merge-1: given two items [$\cdot$=xγ/p]$_s$ and [$\beta \cdot$x/q, S]$_t$, add the item [=x$\cdot \gamma$/ k:(p$\frown$q), S]$_c$ to the chart.

- Merge-2: given two items [$\beta \cdot$=xγ/p, S]$_c$ and [$\mu \cdot$x/q, T]$_t$, add the item [β=x$\cdot \gamma$/k:(q$\frown$p), S, T]$_c$ to the chart.

- Merge-3: given two items [$\beta \cdot$=xγ/p, S]$_t$ and [$\mu \cdot$xδ/q, T]$_t$ ($\delta \neq \emptyset$), add the item [β=x$\cdot \gamma$/p, S, μx$\cdot \delta$/q, T]$_c$ to the chart.

- Move-1: given an item [$\beta \cdot$+yγ/p, S, $\mu \cdot$-y/q, T]$_c$, add the item [β+y$\cdot \gamma$/k:(q$\frown$p), S, T]$_c$ to the chart.

- Move-2: given an item [$\beta \cdot$+yγ/p, S, $\mu \cdot$-yδ/q, T]$_c$ ($\delta \neq \emptyset$), add the item [β+y$\cdot \gamma$/p, S, μ-y$\cdot \delta$/q, T]$_c$ to the chart.

The rules Move-1 and Move-2 are constrained so as to not apply to an item that violates the Shortest Movement Constraint.

The proof of soundness provided in the previous chapter is easily converted into a proof of soundness for the method for computing First_k proposed above. Thus, every deducible item $[\gamma_0 \cdot \delta_0 / s_0, \ldots, \gamma_n \cdot \delta_n / s_n]_t$ truthfully claims that $(s_0, \ldots, s_n) \in \text{First}_k([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t)$. Furthermore, one can prove that the method is complete up to permutation of the last n elements of an item, that is, for every true claim $(s_0, \ldots, s_n) \in \text{First}_k([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t)$, an item $[\gamma_0 \cdot \delta_0 / s_0, \gamma_{j_1} \cdot \delta_{j_1} / s_{j_1}, \ldots, \gamma_{j_n} \cdot \delta_{j_n} / s_{j_n}]_t$ will be deduced, where the sequence of indices $j_1, \ldots, j_n$ is a permutation of the sequence $1, \ldots, n$. Consequently, for any category $\text{A} = [\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t$ such that $\text{T}_c(\text{A})$ contains some tree τ generated by G, the contents of set First_k can be established in the following way:

> $\text{First}_k([\gamma_0 \cdot \delta_0, \ldots, \gamma_n \cdot \delta_n]_t) = \{ (s_0, \ldots, s_n) \mid$ item $[\gamma_0 \cdot \delta_0 / s_0, \gamma_{j_1} \cdot \delta_{j_1} / s_{j_1}, \ldots, \gamma_{j_n} \cdot \delta_{j_n} / s_{j_n}]_t$ has been generated, and $j_1, \ldots, j_n$ is a permutation of $1, \ldots, n\}$.

Suppose tree ϕ participates in the derivation of a complete tree. Then it can be shown that the sequences of syntactic features labeling nodes in ϕ other than the head must be sequences of features of the kind `-f`. This implies that we can restrict our attention to relevant categories of the general form $[\gamma_0 \cdot \delta_0, \gamma_1 \cdot \texttt{-f}_1 \delta'_1, \ldots, \gamma_m \cdot \texttt{-f}_n \delta'_n]$. The Shortest Movement Constraint implies that $\texttt{-f}_i \neq \texttt{-f}_j$, for $1 \leq i, j \leq n$ and $i \neq j$. Then, for a category $\text{A} = [\gamma_0 \cdot \delta_0, \gamma_1 \cdot \texttt{-f}_1 \delta'_1, \ldots, \gamma_m \cdot \texttt{-f}_n \delta'_n]$, define $\text{First}_k^0(\text{A}) = \{\text{z}_0 \mid (\text{z}_0, \ldots, \text{z}_n) \in \text{First}_k(\text{A})\}$ and $\text{First}_k^{-\texttt{f}_i}(\text{A}) = \{\text{z}_i \mid (\text{z}_0, \ldots, \text{z}_i, \ldots, \text{z}_n) \in \text{First}_k(\text{A})\}$. Thus, $\text{First}_k^0(\text{A})$ picks out the set containing the first k symbols of the narrow yield of the maximal projection of the head of any tree $\tau \in \text{T}_c(\text{A})$,[10] and $\text{First}_k^{-\texttt{f}_i}(\text{A})$ picks out the set containing the first k symbols of the narrow yield of the maximal projection of the node whose left-most syntactic feature is $\texttt{-f}_i$ for any tree $\tau \in \text{T}_c(\text{A})$.

[10] Note that the maximal projection of the head of τ is τ itself.

For the categories in grammar G_2, we thus find, for example, First_1([=pred·+v +k i, d·-k, =d vt·-v]$_c$) = {(*s*, *Titus*, *praise*), (*s*, *Lavinia*, *praise*)} and $\text{First}_1^{-\text{k}}$([=pred·+v +k i, d·-k, =d vt·-v]$_c$) = {*Titus*, *Lavinia*}. Also, First_1^0([=i·c]$_c$) = {*Titus*, *Lavinia*} and First_1^0([=i +wh·c]$_c$) = {*who*}.

5.4.2 Recognition with Look-Ahead

The recognizer with look-ahead will check, for every new item that is about to be added to the chart according to the rules of inference given in section 5.3.3, whether it is harmonious with the input by consulting the various sets First_k^0 and $\text{First}_k^{-\text{y}}$.

For a Minimalist Grammar G with distinguished category c, input string $w = w_1 \ldots w_n$, and look-ahead k, the item [(0, n):·c]$_s$ will only be an axiom if $k{:}w \in \text{First}_k^0([\cdot \text{c}]_s)$, and the item [(0, n):$\gamma$·c]$_c$ will only be an axiom if $k{:}w \in \text{First}_k^0([\gamma \cdot \text{c}]_c)$. The goal items are defined as before.

As concerns the rules of inference, Unmerge-1 will add an item Δ_1 + … + Δ_{i-1} + [(p, v):·=xγ]$_s$ + [(v, q):β·x, S]$_t$ + Δ_{i+1} + … + Δ_m to the chart provided that $k{:}(w_{p+1} \ldots w_v) \in \text{First}_k^0([\cdot \text{=x}\gamma]_s)$ and $k{:}(w_{v+1} \ldots w_q) \in \text{First}_k^0([\beta \cdot \text{x, S}]_t)$.

Unmerge-2 will add an item Δ_1 + … + Δ_{i-1} + [(v, q):α·=xγ, U]$_c$ + [(p, v):β·x, V]$_t$ + Δ_{i+1} + … + Δ_m to the chart provided that $k{:}(w_{v+1} \ldots w_q) \in \text{First}_k^0([\alpha \cdot \text{=x}\gamma, \text{U}]_c)$ and $k{:}(w_{p+1} \ldots w_v) \in \text{First}_k^0([\beta \cdot \text{x, V}]_t)$.

Unmerge-3 will add an item Δ_1 + … + Δ_{i-1} + [(p, q):α·=xγ, U]$_{t_1}$ + [(v, w):β·xδ, V]$_{t_2}$ + Δ_{i+1} + … + Δ_m to the chart provided that $k{:}(w_{p+1} \ldots w_q) \in \text{First}_k^0([\alpha \cdot \text{=x}\gamma, \text{U}]_{t_1})$ and $k{:}(w_{v+1} \ldots w_w) \in \text{First}_k^0([\beta \cdot \text{x}\delta, \text{V}]_{t_2})$.[11]

[11] Note that the application of rule Unmerge-3 must be preceded by an application of rule Unmove-1, because in a bottom-up derivation, any application of *move* must be preceded by an application of *merge* which will contribute the tree that will move. Therefore, the condition $k{:}(w_{v+1} \ldots w_w) \in \text{First}_k^0([\beta \cdot \text{x}\delta, \text{V}]_c)$ for Unmerge-3 will be evaluated after the condition $k{:}(w_{v+1} \ldots w_w) \in \text{First}_k^{-\text{y}}([\alpha \cdot \text{+y}\gamma, \beta \text{x}\delta' \text{-y}, \text{S}]_c)$, $\delta = \delta'$-y, for Unmove-1 has been found true. A similar point can be made for the condition involving $\text{First}_k^{-\text{y}}$ in rule Unmove-2. Hence, if in general $\text{First}_k^0([\beta \cdot \text{x}\delta, \text{V}]_c) = \text{First}_k^{-\text{y}}([\alpha \cdot \text{+y}\gamma, \beta \text{x}\delta' \cdot$-

Unmove-1 will add an item Δ_1 + ...+ Δ_{i-1} + [(v, q):α·+yγ, (p, v):β·-y, S]$_c$ + Δ_{i+1} + ...+ Δ_m to the chart provided that k:($w_{v+1} \ldots w_q$) ∈ First$_k^0$([α·+yγ, β·-y, S]$_c$) and k:($w_{p+1} \ldots w_v$) ∈ First$_k^{-\mathtt{y}}$([α·+yγ, β·-y, S]$_c$).

Unmove-2 will add an item Δ_1 + ...+ Δ_{i-1} + [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$ + Δ_{i+1} + ...+ Δ_m to the chart provided that k:($w_{p+1} \ldots w_q$) ∈ First$_k^0$([α·+yγ, S, β·-yδ, T]$_c$) and k:($w_{v+1} \ldots w_w$) ∈ First$_k^{-\mathtt{y}}$([α·+yγ, S, β·-yδ, T]$_c$).

The Scan rule remains unchanged.

The reduction in the number of items brought about by the use of First$_k$ depends on particularities of the grammar and the input sentence. When the grammar describes a natural language, beneficial effects are expected, because in natural languages most words serve as the first word of only a limited number of categories. For grammar G_2, the savings are significant. Without look-ahead, i.e., k = 0, recognition of the sentence $_0$*Titus*$_1$*praise*$_2$*s*$_3$*Lavinia*$_4$ involves the deduction of 340 items. For k = 1, it only takes 15 items.[12] The recognizer with look-ahead does not consider the item [(0, 4):=i +wh·c]$_c$ an axiom, because w_1 = *Titus* ∉ First$_1^0$([=i +wh·c]$_c$). Hence, all items deducible from this item are suppressed. This example also illustrates the advantage of using dots in items. Without dots, categories [=i +wh·c]$_c$ and [=i ·c]$_c$ would be collapsed into one category [c]$_c$ for which First$_1^0$([c]$_c$) = {*Titus*, *Lavinia*, *who*}, whence the recognizer would not be able to immediately dismiss either the declarative or the interrogative analysis. Furthermore, the recognizer with look-ahead will split all position vectors correctly for G_2. For example, unmerging the axiom [(0, 4):=i·c]$_c$, the recognizer will not posit the item [(0, 1):·=i c]$_s$ + [(1, 4):=pred +v +k·i]$_c$, since w_1 = *Titus* ∉ First$_1^0$([·=i c]$_s$) = ∅ and w_2 = *praise* ∉ First$_1^0$([=pred +v +k·i]$_c$) =

y, S]$_c$), one of the conditions on either Unmerge-3 or Unmove-1 and Unmove-2 can be dropped. In that case an arrangement is possible in which all conditions involve checking the contents of First$_k^0$ rather than First$_k^{-\mathtt{y}}$.

[12] This is the least number of items possible: 1 axiom, 8 items corresponding to the 8 intermediate trees in the derivation of the sentence, plus 6 scanned items.

$\{$*Lavinia*, *Titus*$\}$.

5.5 Correct-Prefix Property

For the recognizer to possess the correct prefix property, it must read the words of the input sentence from left to right, and it must not predict items with subitems whose categories represent trees that cannot be derived from lexical trees in the grammar. These two restrictions will be discussed in the following two sections.

5.5.1 Left to Right

Obviously, the recognizer described in the preceding sections does not scan the input sentence from left to right, because the rules of inference can rewrite any subitem of an item as long as it matches the antecedent of the rule. To ensure a left to right sweep through the sentence, the items are extended to include a pointer π, which points to the next word of the sentence to be read. The rules of inference are restricted to rewrite only subitems whose trees are predicted to dominate the word the pointer π is pointing to. The pointer will be incremented when all the words at a particular position in the sentence have been scanned successfully.

For a proper implementation of the pointer mechanism it is important to realize that phonetically empty words at position x in a sentence come before a phonetically non-empty word at position x. If the recognizer does not scan the words at position x in this order, a violation of the correct-prefix property may result, as illustrated by the following example. Consider a grammar G_3 which is like grammar G_1 of section 5.1, except that the phonetically empty lexical item `=i c` ϵ has been replaced with the phonetically non-empty lexical item `=i c` *that*. Thus, all the sentences generated by grammar G_3 will start with the overt complementizer *that*. In the process of trying

to recognize the ungrammatical sentence *Titus praise s Lavinia*, a recognizer without look-ahead will derive the item [(0, 0):·=i c]$_s$ + [(2, 3):·=pred +v +k i]$_s$ + [(3, 4):=vt +k·=d pred, (1, 2):=d vt·-v]$_c$ + [(0, 1):·d -k]$_s$ (for grammar G_1 this item would correspond to the cut 5-4-9-2 in the derivation tree in 15). The first subitem of this item amounts to the prediction of a phonologically empty complementizer at position 0, i.e., at the left edge of the sentence. The last subitem predicts the word *Titus* to occur at the left edge of the sentence. Scanning the non-empty word *Titus* before the predicted empty complementizer leads to a violation of the correct-prefix property, because no sentence generated by G_3 starts with the word *Titus*. The recognizer should halt after failing to scan the empty complementizer and before scanning anything else – a phonetically empty complementizer is a prefix of no sentence in G_3.

The example above shows that we have to distinguish between the situation in which the recognizer is looking for a phonetically empty word at position x ana the situation in which the recognizer is looking for a phonetically non-empty word at position x. Therefore, pointer π will have two parts. If π = (x, ϵ), the recognizer finds itself in the former situation, if π = (x, $\not\epsilon$), in the latter. The values that π will run through are ordered in such a way that (x, ϵ) comes immediately before (x, $\not\epsilon$), and (x, $\not\epsilon$) comes immediately before ($x + 1$, ϵ). The first value of π is (0, ϵ); the last value is (n, ϵ), where n is the length of the input sentence.

In order to determine what subitems should be rewritten given some value of π, we define a containment relation between position vectors of items and values of π. The containment relation is used to determine what positions in a sentence are potentially covered by trees for subitems with particular position vectors. Position vector (q, q) in a simple subitem contains the following position: (q, ϵ). The yield of any tree $\tau \in T_s([(q, q):\gamma\cdot\delta]_s)$ can only be a phonetically empty word at position q. Position vector (q, r), $q < r$, in a simple subitem contains the following positions: (q, $\not\epsilon$),

$\ldots$, ($r - 1$, $\not\epsilon$). Positions (q, ϵ) and (r, ϵ) are not included, because the yield of any simple tree τ in T_s([(q, r):$\gamma \cdot \delta$]$_s$) is the string $s = w_{q+1} \ldots w_r$. Since $q < r$, s is not a phonologically empty word at position q, nor a phonologically empty word at position r. However, if the position vector (q, r), $q < r$, appears in a complex subitem, these positions are included in the set of positions contained in position vector (q, r), because the derivation of a tree whose yield includes the string $s = w_{q+1} \ldots w_r$ may very well involve trees whose yields are phonologically empty words at positions q or r. Hence, position vector (q, r), $q \leq r$, in a complex subitem includes the following positions: (q, ϵ), $\ldots$, (r, ϵ). A rule of inference can only apply to some item if its triggering subitem Δ_i contains the current value of π. For example, the rule of inference $Unmerge$-1 now reads:

Unmerge-1: given an item ($\Delta_1 + \ldots + \Delta_m$, π) such that Δ_i = [(p, q):=x$\cdot\gamma$, S]$_c$, $1 \leq i \leq m$, and one of the position vectors of Δ_i contains π, then, for any lexical item with syntactic features βx, add the items ($\Delta_1 + \ldots + \Delta_{i-1}$ + [(p, v):$\cdot$=xγ]$_s$ + [(v, q):$\beta \cdot$x, S]$_t$ + $\Delta_{i+1} + \ldots + \Delta_m$, π) to the chart, for all possible values of v such that $p \leq v \leq q$; for all possible values of t such that $t = s$ if $\beta = \emptyset$, $t = c$ if $\beta \neq \emptyset$, and if $t = s$ then S = $\emptyset$.

The other rules of inference are updated in the same way, except for Scan, which will be discussed shortly.

Returning to our example grammar G_3, when recognition of the ungrammatical sentence *Titus praise s Lavinia* has reached the item ([(0, 0):$\cdot$=i c]$_s$ + [(2, 3):$\cdot$=pred +v +k i]$_s$ + [(3, 4):=vt +k$\cdot$=d pred, (1, 2):=d vt$\cdot$-v]$_c$ + [(0, 1):$\cdot$d -k]$_s$, (0, ϵ)), only subitem [(0, 0):$\cdot$=i c]$_s$ can be scanned, because its position vector contains (0, ϵ), whereas the position vector of subitem [(0, 1):$\cdot$d -k]$_s$ does not. Thus, the correct-prefix property is safe.

The position vector (0, n) of any axiom is split in such a way that every item generated by the recognizer has exactly one position vector that contains the value (x,

$\not\epsilon$), $0 \leq x \leq n-1$. This, of course, reflects the obvious truth that there is exactly one phonetically non-empty word at any position x in a sentence of length n. Therefore, after having scanned the one and only non-empty word at position x, the recognizer can increment the value of the pointer π from (x, $\not\epsilon$) to ($x+1$, ϵ), assuming that the phonetic content of a phonetically non-empty lexical item amounts to one word. In contrast, there can be more than one phonetically empty word at some position x in a sentence. Consider for example grammar G_4, which has empty pronouns. The lexicon of G_4 includes the lexicon of grammar G_1 and the lexical item d -k ϵ. One of the sentences generated by this grammar is ϵ ϵ *praise s Lavinia*, in which one of the empty elements is an empty complementizer and the other is an empty determiner phrase.[13] In order to recognize this sentence both empty elements have to be scanned. Thus, after having scanned one of them, the recognizer can not increment the value of the pointer π from (0, ϵ) to (0, $\not\epsilon$), because then it would miss the other empty element. If the current value of the pointer is (x, ϵ), it can only be incremented if the item under consideration does not have any subitems left whose position vectors contain (x, ϵ), except for subitems with subscript l, that is, when all predicted empty elements at position x have been scanned successfully. So the rule of inference Scan is split into two rules, covering 'empty' and 'non-empty' scans, and a rule of inference for updating π without scanning any word is added. This last rule will apply when the recognizer is supposed to be looking for an empty word at some position x, but none is predicted to occur at that position or all have been scanned successfully.

Scan-ϵ: given an item (Δ_1 + … + Δ_m, (p, ϵ)) such that Δ_i = [(p, p):$\cdot\gamma$]$_s$, $1 \leq i \leq m$, then, if there is a lexical item with syntactic features γ and phonetic features ϵ, add the following item to the chart: (Δ_1 + … + Δ_{i-1} + [(p, p):$\cdot\gamma$]$_l$ + Δ_{i+1} + … + Δ_m, (p, ϵ)).

[13] According to the derivation tree of the sentence, the empty complement precedes the empty determiner phrase, but for the recognizer the order between multiple empty elements at the same position in a sentence is immaterial.

Scan-$\not\epsilon$: given an item (Δ_1 + ... + Δ_m, (p, $\not\epsilon$)) such that Δ_i = [(p, q):·γ]$_s$, $p < q$, 1 $\leq i \leq m$, then, if there is a lexical item with syntactic features γ and phonetic features covering $w_{p+1} \ldots w_q$ of the input string, add the following item to the chart: (Δ_1 + ... + Δ_{i-1} + [(p, q):·γ]$_l$ + Δ_{i+1} + ... + Δ_m, (q, ϵ)).

Update-π: given an item (Δ_1 + ... + Δ_m, (p, ϵ)) such that none of the Δ_i, $1 \leq i \leq m$, includes a simple or complex subitem whose position vectors contain (p, ϵ), then add the following item to the chart: (Δ_1 + ... + Δ_m, (p, $\not\epsilon$)).

For grammar G_4 and the grammatical sentence *praise s Lavinia*, the recognizer will generate the item ([(0, 0):·`=i c`]$_s$ + [(1, 2):·`=pred +v +k i`]$_s$ + [(2, 3):`=vt +k`·`=d pred`, (0, 1):`=d vt`·`-v`]$_c$ + [(0, 0):·`d -k`]$_s$, (0, ϵ)). At this point, the rule Scan-ϵ can apply twice, but after these applications, rule Update-π cannot apply, because (0, ϵ) is contained in the position vector (0, 1) occurring in a complex subitem – a tree represented by this subitem may be derived from a tree with an empty element at position 0, which should be scanned before π is incremented. We will only know for sure that there are no further empty elements at position 0 other than the empty complementizer and determiner phrase when the recognizer has generated the item ([(0, 0):·`=i c`]$_l$ + [(1, 2):·`=pred +v +k i`]$_s$ + [(3, 3):·`=vt +k =d pred`]$_s$ + [(0, 1):·`=d vt -v`]$_s$ + [(0, 0):·`d -k`]$_l$, (0, ϵ)). (For grammar G_1, this item represents the cut 5-4-6-3-1-2 of the derivation tree in 15.) The rule Update-π will apply to this item, incrementing the value of π to (0, $\not\epsilon$). Next, three applications of Scan-$\not\epsilon$, each followed by an application of Update-π, will produce a goal item, which now are items of the form ([(x_0, y_0):·γ_0]$_l$ + ... + [(x_m, y_m):·γ_m]$_l$, (n, $\not\epsilon$)), where n is the length of the input sentence.

In conclusion, the inclusion of the pointer π in an item ensures that the partial derivation tree that the item stands for according to the invariant of section 5.3.1 is developed in a depth-first manner such that the recognizer will go after the leaves

whose yields are predicted to start at position π first. When each of these leaves is reached, the applicable Scan rule checks whether the phonetic form of the simple tree corresponding to the leaf matches the phonologically empty or non-empty word π is pointing to. If so, π will be updated appropriately. It follows that the recognizer will read the words of the input sentence from left to right.[14, 15]

5.5.2 Useless Categories

To illustrate the second aspect of the correct-prefix property, that the recognizer must not generate items with subitems whose categories represent trees that cannot be derived from lexical trees in the grammar, consider grammar G_5, which is grammar G_1 with two additional lexical items: `=p pred` ϵ and `p -k` *Rufus*. Assume the input sentence is *Rufus praise s Lavinia*, which is not a sentence of the language specified by G_5, since neither lexical item `=p pred` ϵ nor lexical item `p -k` *Rufus* can be part of a derivation of a complete tree. At a certain point, the recognizer without look-ahead will have produced the item ([(0, 0):·`=i c`]$_s$ + [(2, 3):·`=pred +v +k i`]$_s$ + [(3, 4):`=p`·`pred`, (0, 1):`p`·`-k`, (1, 2):`=d vt`·`-v`]$_c$, (0, ϵ)) (for grammar G_1, this would be the cut 5-4-10 in the derivation tree in 15), from which the item ([(0, 0):·`=i c`]$_s$ + [(2, 3):·`=pred +v +k i`]$_s$ + [(3, 4):·`=p pred`, (1, 2):`=d vt`·`-v`]$_c$, + [(0, 1):·`p -k`]$_s$, (0, ϵ)) will be deduced by an application of the rule Unmerge-1. Next, the empty complementizer will be scanned, after which π will be updated, producing the item ([(0,

[14] While the words of the sentence are read from left to right by the recognizer, checking whether the predicted items are harmonious with the input may involve look-ahead strings that are not to the immediate right of the position π is pointing to. For example, for grammar G_1 and input sentence *Titus praise s Lavinia*, using rule of inference Unmerge-1 to posit the item ([(0, 0):·`=i c`]$_l$ + [(2, 3):·`=pred +v +k i`]$_s$ + [(3, 4):`=vt +k =d`·`pred`, (0, 1):`d`·`-k`, (1, 2):`=d vt`·`-v`]$_c$, (0, $\not\epsilon$)) for the cut 5-4-10 of the derivation tree in 15 involves looking at the strings k:w_3 and k:w_4, whereas the next word to be read is w_1.

[15] Note that the axioms of the top-down recognizer require the length of the sentence to be known before it has been recognized. This problem can be solved by using position vectors whose coordinates are variables.

0):·=i c]$_l$ + [(2, 3):·=pred +v +k i]$_s$ + [(3, 4):·=p pred, (1, 2):=d vt·-v]$_c$, + [(0, 1):·p -k]$_s$, (0, $\not\epsilon$)). In the next step, the first non-empty word of the input sentence will be scanned, in violation of the correct prefix property, because *Rufus* is not a prefix of any sentence in the language generated by G_5.

This example shows that the predictions of the recognizer have to be limited to items that do not contain any useless subitems. A subitem Δ = [(x_0, y_0):$\gamma_0 \cdot \delta_0$, …, (x_n, y_n):$\gamma_n \cdot \delta_n$]$_t$ is useless if its category A = [$\gamma_0 \cdot \delta_0$, …, $\gamma_n \cdot \delta_n$]$_t$ is useless, that is, if there are no trees in T_c(A) that participate in the derivation of a complete tree, cf. the notion 'useless non-terminal symbol' for Context-Free Grammars, e.g. [SS88]. It follows from the invariant in section 5.3.1 that for a subitem Δ occurring in some item, any tree $\tau \in T_c$(A), where A is the category corresponding to Δ, is part of a partial derivation of a complete tree. Hence, we still have to make sure that category A is such that set T_c(A) consists of trees that are derivable from lexical items of the grammar. We can compute the set of categories with this property by closing the set of categories for the lexical items of the grammar under the functions Merge-1, …, Move-2 that were used in section 5.4.1 to compute the sets $First_k$, ignoring the tuples of strings associated with each category. Thus, the computation will start with the following set of axioms: for each lexical item of the grammar labeled with syntactic features $\gamma \in Cat^+$, there will be an axiom [$\cdot\gamma$]$_s$, The axioms will be closed under the following five rules of inference:

Merge-1: [·=xγ]$_s$, [β·x, S]$_t$ → [=x·γ, S]$_c$[16]

Merge-2: [β·=xγ, S]$_c$, [μ·x, T]$_t$ → [β=x·γ, S, T]$_c$

Merge-3: [β·=xγ, S]$_t$, [μ·xδ, T]$_t$ → [β=x·γ, S, μx·δ, T]$_c$ ($\delta \neq \emptyset$)

Move-1: [β·+yγ, S, μ·-y, T]$_c$ → [β+y·γ, S, T]$_c$

[16] That is: if the chart contains the items [·=xγ]$_s$ and [β·x, S]$_t$, then add the item [=x·γ, S]$_c$.

Move-2: $[\beta\cdot\texttt{+y}\gamma, \text{S}, \mu\cdot\texttt{-y}\delta, \text{T}]_c \rightarrow [\beta\texttt{+y}\cdot\gamma, \text{S}, \mu\texttt{-y}\cdot\delta, \text{T}]_c, (\delta \neq \emptyset)$

In all rules of inference, $t \in \{s, c\}$. The rules Move-1 and Move-2 are constrained so as not to apply to categories of the form $[\gamma_0 \cdot \delta_0, \ldots, \gamma_i\cdot\texttt{-f}\delta_i, \ldots, \gamma_j\cdot\texttt{-f}\delta_j, \ldots, \gamma_n \cdot \delta_n]_c$, which violate the Shortest Movement Constraint.

Let $\text{U}^{\uparrow}$ be the set of axioms and all items derived from the axioms in this way. Then, whenever the recognizer is about to add a new item to the chart according to the rules of inference in section 5.3.3, it will check whether all its subitems are in the set $\text{U}^{\uparrow}$, and only add the item if this is so. Together with reading the sentence from left to right, this will ensure that the recognizer has the correct prefix property.

For G_5, category $[\texttt{=p}\cdot\texttt{pred}, \texttt{p}\cdot\texttt{-k}, \texttt{=d vt}\cdot\texttt{-v}]_c$ is not an element of $\text{U}^{\uparrow}$, whence the prediction that led to the violation of the correct prefix property will no longer be generated.

It is possible to compute the set of categories such that set $\text{T}_c(\text{A})$ consists of trees that participate in a partial derivation of a complete tree in advance, too, by computing all the categories that are deducible from axiom $[\cdot\texttt{c}]$, using the functions Merge-1, ..., Move-2 in reverse, again ignoring the tuples of strings associated with the categories. Call this set $\text{U}^{\downarrow}$. Then the set of useful categories U, which is the complement of the set of useless categories, is the set $\text{U}^{\uparrow} \cap \text{U}^{\downarrow}$. This set – or rather: $\text{U}^{\downarrow}$, because all lexical items are in $\text{U}^{\uparrow}$ – can be used to prune from a grammar G those lexical items which are not involved in any derivation of a complete tree. A recognizer using a grammar reduced in this way is expected to have the correct prefix property, although this claim still awaits confirmation by formal proof. The critical issue is that a top-down recognizer using a reduced grammar does not necessarily generate items containing subitems whose categories are in $\text{U}^{\uparrow}$. For example, not all the ways in which the contents of set S in the antecedent of the rule of inference Unmerge-2 in section 5.3.3 can be divided over the sets U and V in the consequent of that rule will correspond

to subitems whose categories are in $U^{\uparrow}$. However, at this point it is not clear whether items containing these spurious subitems will ever give rise to predictions of lexical items that prompt violations of the correct-prefix property.

5.6 Proofs of Correctness

In this section we will provide the soundness and completeness proofs for the basic recognizer specified in section 5.3 of this chapter. Given an arbitrary grammar and input sentence, the recognizer is sound if every derivable item represents a true grammatical claim under the invariant, and the recognizer is complete if for every string that is in the language defined by the grammar there is a derivation of a goal item from the axioms. Adding a pointer π to obtain the correct-prefix property merely changes the order in which the rules of inference apply, and so will compromise neither soundness nor completeness of the recognizer. Adding look-ahead will actually prevent some rules of inference from applying in certain situations, but it is easy to see that, were these rules of inference to apply, no goal items would be derived. Therefore, the recognizer with look-ahead is still sound and complete.

Although the top-down recognition algorithm in this chapter is introduced in terms of trees generated by Minimalist Grammars, the proofs in the previous chapter allow us to avoid all references to trees altogether. This will greatly simplify the correctness proofs that follow below.

5.6.1 Fundamental Definitions

We will start with a brief review of some fundamental definitions.

5.6.1.1 Minimalist Grammars

A Minimalist Grammar is defined to be a quadruple (V, Cat, Lex, $\mathcal{F}$), where V is a finite set of non-syntactic features, Cat is a finite set of syntactic features, Lex is a finite set of lexical items built from Cat and V, and $\mathcal{F}$ = {Merge-1, Merge-2, Merge-3, Move-1, Move-2} is a set of structure building functions, each of which is defined as in section 4.1.2 of the previous chapter.

Expressions in a Minimalist Grammar are of the form $[\alpha_0, \alpha_1, \ldots, \alpha_m]_t$, with $t \in \{c, s\}$. Each chain α_i, $0 \leq i \leq m$ is of the form $s{:}\gamma$, where s is a string over the alphabet $\Sigma \subseteq$ V, and $\gamma \in$ Cat*. In a Minimalist Grammar G = (V, Cat, Lex, $\mathcal{F}$), the elements of Lex are assumed to be simple expressions, i.e., they are all of the form $[\alpha_0]_s$

As before, for a Minimalist Grammar G, CL(G) denotes the closure of the lexicon Lex under the structure building functions in $\mathcal{F}$. The language derivable by G is L(G) = $\{s \mid [s{:}\mathsf{C}]_t \in$ CL(G), $t \in \{c, s\}\}$, C being the distinguished category feature in Cat.

A derivation tree in a Minimalist Grammar G is defined as follows. Any single expression generated by G is a derivation tree. Furthermore, if T_1 and T_2 are derivation trees in G with respective roots e_1 and e_2, and e = Merge-1(e_1, e_2), or e = Merge-2(e_1, e_2), or e = Merge-3(e_1, e_2), then the tree whose root is labeled e and which has two immediate daughters, T_1 and T_2, in that order, is also a derivation tree in G. If T_1 is a derivation tree with root e_1, and e = Move-1(e_1), or e = Move-2(e_1), then the tree whose root is labeled e and which has one immediate daughter, T_1, is also a derivation tree in G.

A partial derivation tree for a sentence w is a derivation tree whose root is labeled $[w{:}\mathsf{C}]_t$, $t \in \{c, s\}$. A derivation tree is a complete derivation tree for a sentence w if it is a partial derivation tree for sentence w and all its leaves are lexical items in G.

Obviously, all nodes of a complete derivation tree are expressions in CL(G). This is not necessarily true for a partial derivation tree.

5.6.1.2 Top-Down Recognizer

Given an input string $w = w_1 \ldots w_n$ and a Minimalist Grammar G = (V, Cat, Lex, F), the items of the recognizer will be sequences of subitems, where each subitem is of the form $[\alpha_0, \alpha_1, \ldots, \alpha_m]_t$, $t \in \{s, c, l\}$. For $0 \leq i \leq m$, α_i will be of the form (x_i, y_i):$\gamma_i \cdot \delta_i$, where $0 \leq x_i \leq y_i \leq n$, and $\gamma_i, \delta_i \in$ Cat*.

The definitions of the axioms, goal items, and rules of inference of the recognizer are as in section 5.3.

The items of the recognizer and the expressions generated by G are related in the following way: a subitem Δ = [(x_0, y_0):$\gamma_0 \cdot \delta_0$, (x_1, y_1):$\gamma_1 \cdot \delta_1$, …, (x_m, y_m):$\gamma_m \cdot \delta_m$]$_t$ corresponds to the expression e_Δ = [$w_{x_0+1} \ldots w_{y_0}$:δ_0, $w_{x_1+1} \ldots w_{y_1}$:δ_1, …, $w_{x_m+1} \ldots w_{y_m}$:δ_m]$_t$. If subitem Δ is represented as [(x_0, y_0):$\gamma_0 \cdot \delta_0$, S]$_t$, we will write [$w_{x_0+1} \ldots w_{y_0}$:δ_0, S_Δ]$_t$ for its corresponding expression.

The recognizer operates under the following invariant: given an item $\Delta_1 + \Delta_2 + \ldots + \Delta_m$, the sequence of corresponding expressions $e_{\Delta_1}, e_{\Delta_2}, \ldots, e_{\Delta_m}$ is the fringe of a partial derivation tree in G. The fringe of a derivation tree is a sequence of its leaves, their order determined by the precedence relation on the nodes of the derivation tree.

5.6.2 Proof of Soundness

In order to establish soundness, it has to be shown that every derivable item represents a true grammatical claim under the invariant.

5.6.2.1 Soundness of the Axioms

Obviously, the expression $[w_1 \ldots w_n$:c$]_s$ for the axiom [(0, n):$\cdot$c$]_s$ is the fringe of a partial derivation tree – the partial derivation tree consisting of the one node $[w_1 \ldots w_n$:c$]_s$; ditto for any axiom [(0, n):$\gamma\cdot$c$]_c$.

5.6.2.2 Soundness of the Rules of Inference

We will show that the rules of inference Unmerge-1 and Unmove-2 are sound. The proofs for the other rules of inference follow the same pattern.

Unmerge-1 Given an item Δ_1 + ...+ Δ_m such that Δ_i = [(p, q):=x$\cdot\gamma$, S]$_c$ for some i, $1 \leq i \leq m$, assume, in accordance with the invariant, that the corresponding sequence of expressions e_{Δ_1} + ...+ e_{Δ_m} is the fringe of a partial derivation tree T. Any pair of expressions e_1 = $[w_{p+1} \ldots w_v$:=x$\gamma]_s$, e_2 = $[w_{v+1} \ldots w_q$:x, S$_\Delta]_t$, $t \in \{s, c\}$, $p \leq v \leq q$, will be in the domain of the function Merge-1. Applying this function to the expressions e_1 and e_2 will produce the expression e_{Δ_i} = $[w_{p+1} \ldots w_q$:γ, S$_\Delta]_c$. Hence, the sequence of expressions e_{Δ_1} + ...+ $e_{\Delta_{i-1}}$ + e_1 + e_2 + $e_{\Delta_{i+1}}$ + ...+ e_{Δ_m} constitutes the fringe of another partial derivation tree T′. Applying rule Unmerge-1 to the item Δ_1 + ...+ Δ_m will yield a set of new items Δ_1 + ...+ Δ_{i-1} + [(p, v):$\cdot$=x$\gamma]_s$ + [(v, q):$\beta\cdot$x, S]$_t$ + Δ_{i+1} + ...+ Δ_m, $p \leq v \leq q$, provided there is a lexical item with syntactic features βx. These items are justified by the existence of the fringes e_{Δ_1} + ...+ $e_{\Delta_{i-1}}$ + e_1 + e_2 + $e_{\Delta_{i+1}}$ + ...+ e_{Δ_m}.

Unmove-2 Given an item Δ_1 + ...+ Δ_m, such that Δ_i = [(p, q):α+y$\cdot\gamma$, S, (v, w):β-y$\cdot\delta$, T]$_c$ for some i, $1 \leq i \leq m$, assume, in accordance with the invariant, that the corresponding sequence e_{Δ_1} + ...+ e_{Δ_m} is the fringe of a partial derivation tree T. Any expression e_1 = $[w_{p+1} \ldots w_q$:+yγ, S$_\Delta$, $w_{v+1} \ldots w_w$:-yδ, T$_\Delta]_c$, $\delta \neq \emptyset$, which satisfies the Shortest Move Constraint, will be in the domain of Move-2. Applying

this function to expression e_1 will produce the expression e_{Δ_i} = [$w_{p+1} \ldots w_q$:γ, S_Δ, $w_{v+1} \ldots w_w$:δ, T_Δ]$_c$. Hence, the sequence of expressions e_{Δ_1} + ... + $e_{\Delta_{i-1}}$ + e_1 + $e_{\Delta_{i+1}}$ + ... + e_{Δ_m} is the fringe of another derivation tree T′. Applying the rule Unmove-2 to Δ_1 + ... + Δ_m will produce another item Δ_1 + ... + Δ_{i-1} + [(p, q):$\alpha \cdot$+yγ, S, (v, w):$\beta \cdot$-yδ, T]$_c$ + Δ_{i+1} + ... + Δ_m, which is justified by the fringe e_{Δ_1} + ... + $e_{\Delta_{i-1}}$ + e_1 + $e_{\Delta_{i+1}}$ + ... + e_{Δ_m}.

5.6.3 Proof of Completeness

In order to establish completeness, it has to be shown that for every string that is in the language defined by the grammar there is a derivation of a goal item from the axioms.

The proof of lemma 1, from which completeness will follow, requires some auxiliary definitions. Let T be a derivation tree in G with root ρ. For two expressions x_1, x_2 in T, let d(x_1, x_2) denote the length of the path between x_1 and x_2, provided such a path exists. Now consider a cut Φ = e_1, ..., e_m of T and define the depth of cut Φ as follows: d(Φ) = $\sum_{i=1}^{m}$ d(ρ, e_i).

For a simple expression e = [$w_{x_0+1} \ldots w_{y_0}$:δ_0]$_s$ occurring in a derivation tree T of w, define the singleton set of subitems $\Delta(e)$ = {[(x_0, y_0):$\cdot\delta_0$]$_s$}. For a complex expression e = [$w_{x_0+1} \ldots w_{y_0}$:δ_0, $w_{x_1+1} \ldots w_{y_1}$:δ_1, ..., $w_{x_m+1} \ldots w_{y_m}$:δ_m]$_c$ occurring in a derivation tree T of w, define the set of subitems $\Delta(e)$ = {[(x_0, y_0):$\gamma_0 \cdot \delta_0$, (x_1, y_1):$\gamma_1 \cdot \delta_1$, ..., (x_m, y_m):$\gamma_m \cdot \delta_m$]$_c$ | $\gamma_i \delta_i$ are the syntactic features of some lexical item $\ell \in$ Lex, $\gamma_i \neq \emptyset$, $0 \leq i \leq m$}.

Lemma 1 If Φ = e_1, ..., e_m is a cut of a complete derivation tree T of sentence w, then the recognizer will produce all items Δ_1 + ... + Δ_m such that $\Delta_i \in \Delta(e_i)$, $0 \leq i \leq m$.

Proof. The proof will use induction on the depth of the cuts of T. Assume first that

d(Φ) = 0. This implies that Φ is a cut through the root of the derivation tree, i.e., Φ consists of the single expression $[s{:}\mathsf{C}]_t$, $t \in \{s, c\}$. For both $t = s$ and $t = c$, the items in $\Delta([s{:}\mathsf{C}]_t)$ are introduced as axioms.

Next, consider a cut Φ = $e_1, \ldots, e_m$ of T such that d(Φ) = k, $k > 0$. Pick an arbitrary expression e_i from Φ, $0 \leq i \leq m$. Since $k > 0$, e_i cannot be the root of T. Hence, one of the following five situations obtains: there is a e in T such that either e = Merge-1(e_i, e_{i+1}), or e = Merge-2(e_i, e_{i+1}), or e = Merge-3(e_i, e_{i+1}),[17] or e = Move-1(e_i), or e = Move-2(e_i).

Assume e = Merge-1(e_i, e_{i+1}). By the definition of Merge-1, e_i is a simple expression $[s{:}{=}\mathsf{x}\gamma]_s$, and e_{i+1} is an expression $[t{:}\mathsf{x}, \mathrm{S}_\Delta]_t$, $t \in \{c, s\}$; assume $t = c$. Then expression e is the complex expression $[st{:}\gamma, \mathrm{S}_\Delta]_c$. Since expression e appears in a complete derivation tree of a sentence w, there are p and q such that $st = w_{p+1} \ldots w_q$, whence $s = w_{p+1} \ldots w_r$ and $t = w_{r+1} \ldots w_q$, for some r, $0 \leq p \leq r \leq q \leq n$, n the length of sentence w. Clearly, for every $[(\mathrm{p}, \mathrm{r}){:}\cdot{=}\mathsf{x}\gamma]_s \in \Delta(e_i) = \{[(\mathrm{p}, \mathrm{r}){:}\cdot{=}\mathsf{x}\gamma]_s\}$ and $[(\mathrm{r}, \mathrm{q}){:}\beta\cdot\mathsf{x}, \mathrm{S}]_c \in \Delta(e_{i+1})$, $[(\mathrm{p}, \mathrm{q}){:}{=}\mathsf{x}\cdot\gamma, \mathrm{S}]_c \in \Delta(e)$.

Since e_{i+1} is a complex expression in a complete derivation tree, there is a lexical item with syntactic features $\beta\mathsf{x}$, $\beta \neq \emptyset$, from which expression e_{i+1} has been projected, i.e., $\Delta(e_{i+1}) \neq \emptyset$.

Because Φ = $e_1, \ldots, e_m$ is a cut of T, Φ′ = $e_1, \ldots, e_{i-1}, e, e_{i+2}, \ldots, e_m$ is also a cut of T. Clearly, d(Φ′) < d(Φ). Therefore the induction hypothesis applies: there will be items $\Delta_1 + \ldots + \Delta_{i-1} + \Delta + \Delta_{i+2} + \ldots + \Delta_m$, for all $\Delta \in \Delta(e)$ and all $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$, $j \neq i + 1$.

Pick an arbitrary expression $x = [(\mathrm{p}, \mathrm{r}){:}\cdot{=}\mathsf{x}\gamma]_s$ from $\Delta(e_i)$ and an arbitrary expres-

[17] For the purpose of this proof, the case in which the merge operation applies to a pair of expressions e_{i-1}, e_i is identical to the cases in which the merge operation applies to the pair of expressions e_i, e_{i+1}. For example, if e_i is such that e = Merge-1(e_{i-1}, e_i), then e_{i-1} is such that e = Merge-1(e_{i-1}, e_i). The reverse also holds.

sion y = [(r, q):β·x, S]$_c$ from $\Delta(e_{i+1})$. Then [(p, q):=x·γ, S]$_c \in \Delta(e)$. Because $y \in \Delta(e_{i+1})$, there must be a lexical item with syntactic features βx. Applying rule of inference Unmerge-1 to any item Δ_1 + … + Δ_{i-1} + [(p, q):=x·γ, S]$_c$ + Δ_{i+2} + … + Δ_m, $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$, $j \neq i+1$, will produce a new set of items, including the items Δ_1 + … + Δ_{i-1} + x + y + Δ_{i+2} + … + Δ_m, for all $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$, $j \neq i+1$. Since x is arbitrary in $\Delta(e_i)$ and y is arbitrary in $\Delta(e_{i+1})$, Unmerge-1 will produce the items Δ_1 + … + Δ_m, for all $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$.

Assume now that e = Move-2(e_i). By the definition of Move-2, e_i is a complex expression [s:+yγ, S$_\Delta$, t:-yδ, T$_\Delta$]$_c$, $\delta \neq \emptyset$, and e is the complex expression [s:γ, S$_\Delta$, t:δ, T$_\Delta$]$_c$. Since expressions e and e_i appear in a complete derivation tree of a sentence w, there are positions p, q and v, w such that $s = w_{p+1} \ldots w_q$ and $t = w_{v+1} \ldots w_w$, $0 \leq p \leq q \leq n$, $0 \leq v \leq w \leq n$, n the length of sentence w. Clearly, for every α, β such that [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c \in \Delta$(e$_i$), [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c \in \Delta$(e).

Since e_i is a complex expression in a complete derivation tree, there is a lexical item with syntactic features α+yγ, $\alpha \neq \emptyset$, from which expression e_i has been projected, and a lexical item with syntactic features β-yδ, $\beta \neq \emptyset$, the maximal projection of which is contained in e_i, i.e., Δ(e$_i$) $\neq \emptyset$.

Because $\Phi = e_1, \ldots, e_m$ is a cut of T, $\Phi' = e_1, \ldots, e_{i-1}, e, e_{i+1}, \ldots, e_m$ is also a cut of T. Clearly, d(Φ') < d(Φ). Therefore the induction hypothesis applies: there will be items Δ_1 + … + Δ_{i-1} + Δ + Δ_{i+1} + … + Δ_m, for all $\Delta \in \Delta(e)$ and all $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$.

Pick an arbitrary expression x = [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$ from Δ(e$_i$). Then [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c \in \Delta$(e). Applying the rule of inference Unmove-2 to any item Δ_1 + … + Δ_{i-1} + [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$ + Δ_{i+1} + … + Δ_m, $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$, will yield the item Δ_1 + … + Δ_{i-1} + x

+ Δ_{i+1} + ...+ Δ_m, $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$, $j \neq i$. Since x was arbitrary in $\Delta(e_i)$, Unmove-2 will produce the items Δ_1 + ...+ Δ_m for all $\Delta_j \in \Delta(e_j)$, $0 \leq j \leq m$.

The cases for e = Merge-2(e_i, e_{i+1}), e = Merge-3(e_i, e_{i+1}), and e = Move-1(e_i) follow the models of e = Merge-1(e_i, e_{i+1}) and e = Move-2(e_i). □

Completeness of the recognizer now follows as a corollary from lemma 1 above. If $w \in$ L(G), there is a complete derivation tree T for w in G. Since T is a complete derivation tree, there is a cut $e_1, \ldots, e_m$ such that all $e_j \in$ Lex, $1 \leq j \leq n$. According to lemma 1, the recognizer will generate an item Δ_1 + ...+ Δ_m such that $\Delta_j \in \Delta(e_j)$, $1 \leq j \leq n$. Since all expressions e_j are simple, each $\Delta(e_j)$ is a singleton set containing a subitem of the form $[(x_j, y_j):\cdot\gamma_j]_s$. Since each e_j is a lexical item, the scan rule will apply m times to item Δ_1 + ...+ Δ_m to produce a goal item.

5.7 Conclusion

In this chapter, we have motivated the usefulness of a top-down approach to recognizing languages defined by Minimalist Grammars. We described the design of a basic top-down recognizer, proved its correctness, extended it with look-ahead, and added a pointer to obtain a recognizer with the correct-prefix property. Because of its complexity and potential non-termination, the recognizer *per se* is of limited practical value. However, the top-down perspective on Minimalist Grammars that is explored in this chapter will prove to be very useful for the recognizer that will be developed in the next chapter.

Many interesting questions of a formal nature remain open, e.g. how to characterize Minimalist Grammars whose languages can be parsed deterministically in a top-down manner with at most k symbols of look-ahead, where, different from section 5.4, the look-ahead string is a single contiguous substring of the input sentence, starting at the

immediate right of the last overt word that was scanned.

CHAPTER 6

Parsing Minimalist Languages à la Earley

In this chapter we will describe a recognizer for Minimalist Languages which is based on Earley's parsing algorithm for languages generated by Context-Free Grammars ([Ear68], [Ear70]). The recognizer presented in this chapter essentially is a top-down recognizer, but unlike the top-down recognizer developed in the previous chapter, it is guaranteed to terminate for all sentence and grammar pairs. In section 6.1, we will illustrate the kind of grammar for which the top-down recognizer will fail to halt. Section 6.2 contains an informal introduction to the Earley-style recognizer, explaining how it avoids the termination problem of the top-down recognizer. The formal specification of the recognizer is given in section 6.3. Section 6.4 illustrates the operation of the recognizer with an abstract example. In section 6.5 we provide proofs of soundness and completeness and an analysis of the time and space complexity of the recognizer.

6.1 Left-Recursion

Top-down recognizers are generally unable to handle left-recursive grammars. Consider for example a simple Context-Free Grammar G with start symbol A and the following three rules (uppercase letters are non-terminal symbols, lowercase letters are terminal symbols):

1. A $\rightarrow$ A B.

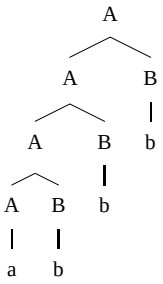
2. A $\rightarrow$ a.

3. B $\rightarrow$ b.

This grammar is left-recursive, because from non-terminal symbol A we can derive a sentential form whose left-most symbol is A.[1] For this particular grammar, a simple derivation showing that G is left-recursive takes just one step: using rule 1, non-terminal symbol A can be rewritten into the sentential form AB.

The language generated by grammar G is ab^*. The derivation of the sentence $abbb$ is summarized in the tree in 4.

4.



A basic left-to-right, top-down recognizer for a Context-Free Grammar will predict sequences of terminal and non-terminal symbols, starting from an initial sequence which consists of the start symbol of the grammar. If the left-most symbol of a sequence σ is a non-terminal symbol, the recognizer will predict all sequences that can be derived from σ by rewriting the left-most symbol of σ according to the rules of the grammar. If the left-most symbol of a sequence σ is a terminal symbol, the recognizer will try to scan a word of the input sentence: if the terminal symbol matches the word in the sentence that follows the last word that was successfully scanned, the terminal symbol will be removed from the sequence. (For a complete description of top-down recognizers for Context-Free Languages, see for example [AU72] and [SS88].) For

[1] A sentential form is a sequence of terminal and non-terminal symbols that is derivable from the start symbol of the grammar by applying the rules of the grammar.

example, for grammar G and input sentence $abbb$, the recognizer will produce the succession of sequences A, AB, ABB, ABBB, aBBB, BBB, bBB, BB, bB, B, b, ϵ, among many others. Since the last sequence is empty and all the words of the input sentence have been scanned, the succession of sequences mentioned above shows that the sentence $abbb$ is indeed in the language generated by G. The recognizer will also produce the series A, AB, ABB, aBB, BB, bB, B, b, ϵ. This attempt to recognize the sentence $abbb$ fails, because the last b of the input sentence has not been scanned.

The problem with a left-recursive grammar is that it will lead the top-down recognizer to pursue an infinite progression of sequences, preventing the recognizer from ever terminating and returning a result. For example, for grammar G, the recognizer will start to produce the succession A, AB, ABB, ABBB, ABBBB, ABBBBB, …, …, …, always using rule 1 to rewrite the left-most terminal symbol of a sequence, and this will never stop. Note that the number of bs in the sentence $abbb$ determines exactly how many times the rule A $\rightarrow$ A B is to be used in the successful recognition of this sentence. This information, however, is unavailable to the top-down parser, because it will go through the sentence from left to right and reach the bs only after it has scanned the initial a. However, if the recognizer always chooses to rewrite the left-most terminal symbol using rule 1, it will never be in a position to scan a.[2]

Left-recursive Minimalist Grammars are similarly problematic for top-down recognizers for Minimalist Languages. Consider for example the Minimalist Grammar H, whose lexicon is given below.[3]

[2]Compare grammar G to grammar G′, whose rules are given below.

1. A $\rightarrow$ B A.
2. A $\rightarrow$ a.
3. B $\rightarrow$ b.

Because terminal symbol a will follow any bs, grammar G′ is no problem for the top-down recognizer.

[3]Grammar H is left-recursive in the following, formal sense. Let G be a Minimalist Grammar as defined in section 5.6 of chapter 5 and discussed in section 4.1 of chapter 4. Let α and β denote

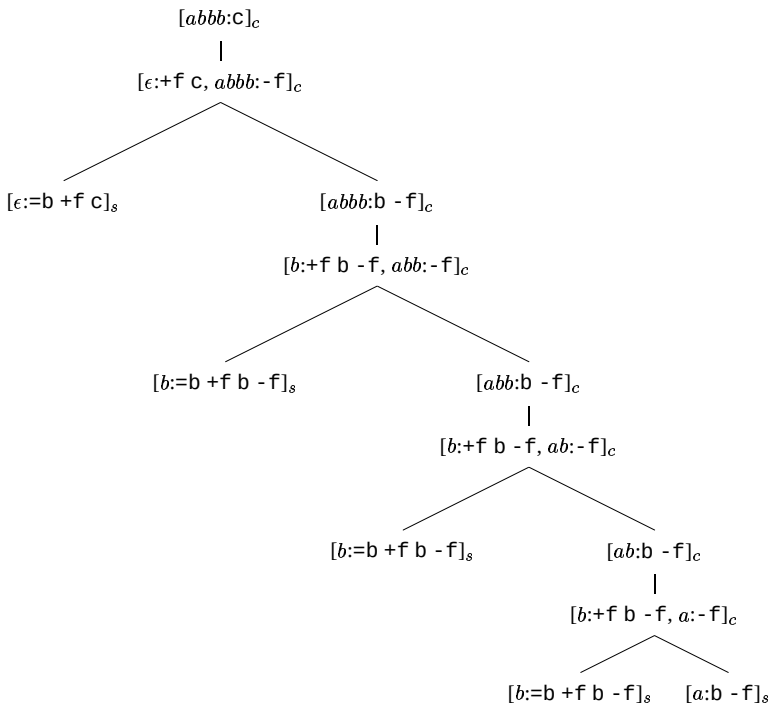
5. [a:b -f]$_s$

6. [b:=b +f b -f]$_s$

7. [ϵ:=b +f c]$_s$

The language generated by Minimalist Grammar H is identical to the language generated by Context-Free Grammar G: ab^*. The derivation tree for the sentence $abbb$ according to grammar H is given in 8.

8.



sequences of expressions. Then the derivation relation $\Rightarrow$ between sequences expressions is defined as follows:

$\alpha e \beta \Rightarrow \alpha e_1 e_2 \beta$ if e = Merge-1(e_1, e_2), or e = Merge-2(e_1, e_2), or e = Merge-3(e_1, e_2)

$\alpha e \beta \Rightarrow \alpha e_1 \beta$ if e = Move-1(e_1), or e = Move-2(e_1)

For convenience, an expression $e = [s_0{:}\gamma_0, s_1{:}\gamma_1, \ldots, s_n{:}\gamma_n]_t$, $t \in \{s, c\}$, will be written as $c/_t s$, where $c = \gamma_0, \gamma_1, \ldots, \gamma_n$, and $s = s_0, s_1, \ldots, s_n$. Let α, β, γ, and δ denote sequences of expressions. Now Minimalist Grammar G is left-recursive if $[w{:}c]_c \Rightarrow^* \alpha\ c/_c s\ \beta$ and $c/_c s \Rightarrow^+ \gamma\ c/_c t\ \delta$, such that the sequence of strings t contains a string which in w is to the left of all strings included in the expressions in γ and δ.

Obviously, this grammar is recursive, because, as the derivation tree shows, any derivation of category A = [b -f]$_c$ can be properly embedded in a larger derivation of the same category, that is, the sequence [b -f]$_c$ can be rewritten into the sequence of categories [=b +f b -f]$_s$ [b -f]$_c$, which contains category A. Since in any sentence generated by H the phonetic material of category A = [b -f]$_c$ in the sequence [=b +f b -f]$_s$ [b -f]$_c$ comes before the phonetic material of category B = [=b +f b -f]$_s$, the top-down recognizer will have to rewrite category A before scanning category B in order to be able to cover the input sentence from left to right. This strategy will lead to the same problem that a top-down recognizer for Context-Free Languages has with the left-recursive grammar G. Writing X for category [c]$_c$, Y for category [+f c, -f]$_c$, Z for category [=b +f c]$_c$, and C for category [+f c -f, -f]$_c$ the top-down recognizer for grammar H will lose itself in the following unbounded expansion: X, Y, ZA, ZC, ZBA, ZBC, ZBBA, ZBBC, ZBBBA, ... As before, the number of bs in the sentence $abbb$ indicates how many times category A should be rewritten into category C and how many times category C should be rewritten into categories B and A in order to recognize the sentence, but the basic top-down recognizer does not exploit this information.[4]

In the next section we will see how left-recursion is not a problem for Earley-style recognizers.

[4] Interestingly enough, no Minimalist Grammar gives rise to left-recursive derivation trees, that is, a derivation tree containing an expression $c/_c s$ whose left-most daughter, not necessarily immediate, is an expression $c/_c t$ – see footnote 3 for this particular notation of expressions. It is easy to see why there are no left-recursive derivation trees. Take an arbitrary Minimalist Grammar G as defined in section 5.1 of the previous chapter and consider an arbitrary expression $e = c/_c s$ Any rewrite rule applicable to e is either of the form $e \rightarrow e_1\ e_2$, where e_1 and e_2 are instantiations of the antecedents of one of the structure building functions Merge-1, Merge-2, or Merge-3, or the rule is of the form $e \rightarrow e_1$, where e_1 is an instantiation of the antecedent of one of the structure building functions Move-1 or Move-2. In all cases, the sequence of features of the first chain in e_1, =xγ_0 or -fγ_0, is longer than the sequence of features of the first chain in e_1, γ_0. So it follows immediately that the category of e_1 must be different from the category of e. A simple induction on the number of rewritings will show that the category of the left-most expression in any sequence of expressions derived from e is different from the category of e.

6.2 Earley's Algorithm

Earley's parsing algorithm for Context-Free Languages avoids the non-termination problem that plagues top-down parsers by predicting what rules could potentially be used to rewrite a non-terminal symbol, but actually using a predicted rule only when there is bottom-up support from the sentence for using that particular rule. For example, for grammar G and input sentence abb, the Earley parser will predict that the start symbol A can be rewritten by the rule A $\rightarrow$ A B and by the rule A $\rightarrow a$. Next, using the rule A $\rightarrow a$, the first word of the sentence will be scanned. The parser has thus recognized word a as a constituent of category A. This constituent can take the place of category A occurring in the right-hand side of the predicted rule A $\rightarrow$ A B. To confirm the remaining part of this rule, category B will be expanded using the rule B $\rightarrow b$. The predicted terminal symbol b matches the next word of the sentence, so the parser has established that word b is a constituent of category B. Together with word a of category A, word b of category B matches the right-hand side of the predicted rule A $\rightarrow$ A B. Thus, using this rule, the recognizer concludes that the string of words ab is another constituent of category A. So far, the rule A $\rightarrow$ A B has been used once, its use justified by the first a and b of the input sentence abb. The string ab of category A can serve as the category A in the right-hand side of the rule A $\rightarrow$ A B in another application of this rule. Again, category B will be expanded using the rule B $\rightarrow b$ and now the second and last b of the sentence will be scanned. String ab of category A and string b of category B thus justify using the rule A $\rightarrow$ A B a second time. It follows from this rule that string abb is of category A. Next, the recognizer will try to use the string abb of category A in yet another application of the rule A $\rightarrow$ A B. However, this attempt will fail, because there is no word b left in the sentence to provide a constituent of category B. Thus the recognizer will halt at this point, returning the result that the sentence abb is a string of category A. Full details of Earley's parsing algorithm for

Context-Free Languages can be found in [Ear68], of course, and also in [SN97] and [JM00], for example.

The Earley-style recognizer for Minimalist Languages presented in the next section will follow a similar strategy. However, the recognizer requires some non-trivial extensions of Earley's original algorithm, having to do with the fact that for Minimalist Grammars the order of the leaves of a derivation tree does not correspond to the order of the words of the sentence whose derivation is given by the tree. For Context-Free Grammars, these two orders coincide. This allows an Earley parser for Context-Free Languages to move from left to right through the right-hand side of a predicted rule. For example, a parser seeking bottom-up confirmation for a predicted rule $A \rightarrow A_1\ A_2$ will first predict rules for rewriting non-terminal symbol A_1 and only if one or more of these rules is confirmed bottom-up will the recognizer predict rules for rewriting non-terminal symbol A_2 and try to find bottom-up support for these.

Expressions in a Minimalist Grammar are associated with tuples of strings which may not end up adjacent to one another in a sentence derived from these expressions. Therefore an Earley recognizer for Minimalist Languages cannot move from left to right through the right-hand side of a predicted rule. The recognizer requires a more elaborate book-keeping mechanism for predicting and confirming rules. Consider for example the sentence *Titus praise s Lavinia* and the Minimalist Grammar given in section 4.2 of chapter 4 generating this sentence. In the process of recognizing this sentence as an expression of category `c`, the recognizer will posit the item in 9.

9. $\langle$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, $\{$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\}$, (1, $\not\epsilon$) $\rangle$

The item in 9 is a triple, consisting of a rewrite rule [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred,

(0, 1):d⋅-k, (1, 2):=d vt⋅-v]$_c$, a set containing the situated expression [(3, 4):=vt +k =d⋅pred, (0, 1):d⋅-k, (1, 2):=d vt⋅-v]$_c$ and a pointer (1, ℓ). The rewrite rule corresponds to an instantiation of the rule of inference Unmerge-1 of the top-down recognizer from section 5.3.3 in chapter 5. The set included in an item contains those situated expressions from the right-hand side of the rewrite rule mentioned in the item, whose position vectors contain the value of the pointer occurring in the item, where containment is defined as in section 5.5.1 of chapter 5.

The generation of the particular item in 9 means that the recognizer proposes to use the structure building function Merge-1, deriving expression [*s Lavinia*:+v +k i, *Titus*:-k, *praise*:-v]$_c$ from expressions [*s*:=pred +v +k i]$_s$ and [*Lavinia*:pred, *Titus*:-k, *praise*:-v]$_c$, and seeks to confirm that the overt word at position 1 in the sentence, *praise*, justifies the use of this structure building function in the derivation of the sentence *Titus praise s Lavinia*, that is, the recognizer will check whether there is a set of expressions including lexical items for the words of the sentence up to and including *praise*, from which the expression [*Lavinia*:pred, *Titus*:-k, *praise*:-v]$_c$ can be derived. If so, then, by virtue of the rewrite rule of the item there is a set of expressions including lexical items for the words of the sentence up to and including *praise*, from which the expression [*s Lavinia*:+v +k i, *Titus*:-k, *praise*:-v]$_c$ can be derived. When the recognizer produces the item in 9, it will already have found a derivation of the expression [*Lavinia*:pred, *Titus*:-k, *praise*:-v]$_c$ from the complex expression [*Lavinia*:=d pred, *praise*:-v]$_c$ and the lexical expression [*Titus*:d -k]$_s$, when it was checking whether the application of Merge-1 mentioned above was consistent with the first overt word of the sentence. The recognizer will now retrieve this derivation and see whether there is a derivation of the expression [*Lavinia*:=d pred, *praise*:-v]$_c$ from a lexical item for the word *praise* and other expressions. According to the derivation tree of the sentence *Titus praise s Lavinia* given in 16 in chapter 5 there is, so the recognizer will eventually derive the item in 10.

10. $\langle$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, $\emptyset$, (1, $\not\epsilon$) $\rangle$

The item in 10 is like the item in 9, except that its second element is the empty set, indicating that the confirmation that was sought has been found.

The recognizer has four different kinds of rules of inference. There is a rule Next, for incrementing the value of the pointer. Returning to our example sentence and grammar, given the item in 11, for example, the rule Next will return the item in 12.

11. $\langle$ S′ $\rightarrow$ [(0, 4):=i·c]$_c$, $\emptyset$, (1, $\not\epsilon$) $\rangle$

12. $\langle$ S′ $\rightarrow$ [(0, 4):=i·c]$_c$, { [(0, 4):=i·c]$_c$ }, (2, ϵ) $\rangle$

The rewrite rule S′ $\rightarrow$ [(0, 4):=i·c]$_c$ is a special rule, corresponding to the top of a derivation tree. The empty set in the antecedent in 11 indicates that there is a set of expressions including lexical items for the words of the sentence up to and including *praise* from which the expression [(0, 4):=i·c]$_c$ can be derived. Hence, next the recognizer will have to check whether any of these derivations can be extended to include a lexical item for the next word of the sentence.[5]

The recognizer also has a set of Predict rules. For the item in 13, one of the Predict rules will generate the items predicting in what way the situated expression [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v] can be rewritten. The item in 14 is among the items generated.

13. $\langle$ [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$ $\rightarrow$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ { [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ }, (0, ϵ) $\rangle$

[5] It so happens that the derivation of the sentence *Titus praise s Lavinia* does not have an empty word at position 2, but the recognizer must check the derivation built so far to find out.

14. $\langle$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, { [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ } (0, ϵ) $\rangle$

The Predict rules predict derivations that have not been explored before, in contrast to the rule of inference Down. For example, for the pair of items in 15 and 16, the rule of inference Down will add the item in 17.

15. $\langle$ [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$ $\rightarrow$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ { [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ }, (3, $\not\epsilon$) $\rangle$

16. $\langle$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, $\emptyset$, (3, ϵ) $\rangle$

17. $\langle$ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ $\rightarrow$ [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, { [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ }, (3, $\not\epsilon$) $\rangle$

If the recognizer wants to check whether there is a set of expressions including lexical items for the words in the sentence up to and including *Lavinia* from which the expression [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$ can be derived, and it is already known that there is a set of expressions including lexical items for all the words preceding *Lavinia* from which expression [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ can be derived, then, by virtue of the rewrite rules mentioned in the items in 15 and 16, all that is left to check is whether any of the latter derivations can be extended to include a lexical item for *Lavinia*. Notice that applying one of the Predict rules in this situation would not be sound. For example, applying the appropriate Predict rule to the item in 15 would produce the item in 18, among many others.

18. ⟨ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ → [(2, 2):·=pred +v +k i]$_s$ [(2, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, { [(2, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ } (3, ∉) ⟩

The item in 18 implies that there is a derivation of the complex expression [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ from a set of expressions including a lexical item covering an empty word at position 2. It is obvious from the grammar that no phonologically empty lexical item with syntactic features =pred +v +k i exists in the lexicon.

The fourth and final rule of inference, Up, will apply to the items in 19 and 20 to produce the item in 21, for example.

19. ⟨ [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$ → [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ { [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ }, (3, ∉) ⟩

20. ⟨ [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ → [(2, 3):·=pred +v +k i]$_s$ [(3, 4):=vt +k =d·pred, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, ∅, (3, ∉) ⟩

21. ⟨ [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$ → [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$, ∅, (3, ∉) ⟩

If the recognizer has found a derivation of the expression [(2, 4):=pred·+v +k i, (0, 1):d·-k, (1, 2):=d vt·-v]$_c$ from a set of expressions including lexical items for the words of the sentence up to and including *Lavinia*, then, because of the rewrite rule mentioned in the item on 19, this applies that a derivation with this property has been found for the expression ⟨ [(1, 4):=pred +v·+k i, (0, 1):d·-k]$_c$. A complete definition of the four kinds of rules of inference can be found in section 6.3.5.

For our example sentence *Titus praise s Lavinia*, the recognizer will start with the axiom given in 22, and it is done when the goal item in 23 has been derived.

22. $\langle$ S′ $\rightarrow$ [(0, 4):=i·c]$_c$, { [(0, 4):=i·c]$_c$ }, (0, ϵ) $\rangle$

23. $\langle$ S′ $\rightarrow$ [(0, 4):=i·c]$_c$, $\emptyset$, (4, ϵ) $\rangle$

6.3 Specification of the Recognizer

As before, the specification of the recognizer involves a specification of the items and their interpretation, a specification of the axioms and goal items, and a specification of the rules of inference under which the axioms will be closed.

First, however, some general definitions are in order. Given an input string w = $w_1 \ldots w_n$ and a Minimalist Grammar G = (V, Cat, Lex, $\mathcal{F}$) as defined in section 5.1 in chapter 5, situated expressions are of the form [α_0, α_1, …, α_m]$_t$, $t \in \{s, c, l\}$. For $0 \leq i \leq m$, α_i will be of the form (x_i, y_i):$\gamma_i \cdot \delta_i$, where $0 \leq x_i \leq y_i \leq n$, and γ_i, $\delta_i \in$ Cat*. Situated expressions generated by the recognizer and expressions generated by grammar G are related in the following way: a situated expression Δ = [(x_0, y_0):$\gamma_0 \cdot \delta_0$, (x_1, y_1):$\gamma_1 \cdot \delta_1$, …, (x_m, y_m):$\gamma_m \cdot \delta_m$]$_t$ corresponds to the expression e_Δ = [$w_{x_0+1} \ldots w_{y_0}$:δ_0, $w_{x_1+1} \ldots w_{y_1}$:δ_1, …, $w_{x_m+1} \ldots w_{y_m}$:δ_m]$_t$. If situated expression Δ is represented as [(x_0, y_0):$\gamma_0 \cdot \delta_0$, S]$_t$, we will write [$w_{x_0+1} \ldots w_{y_0}$:δ_0, S_Δ]$_t$ for its corresponding expression.

For a sentence of length n, pointer π will assume the values (0, ϵ), (0, $\not\epsilon$), (1, ϵ), (1, $\not\epsilon$), …, ($n-1$, $\not\epsilon$), (n, ϵ). If π = (x, ϵ), π is pointing to a phonologically empty word at position x in the sentence; if π = (x, $\not\epsilon$), it is pointing to a phonologically non-empty word at position x, cf. section 5.5.1.

We define functions *next*, *prev* from the set of positions into the set of positions as

follows:

$$\begin{aligned} \textit{next}(x, \epsilon) &= (x, \not\epsilon) \\ \textit{next}(x, \not\epsilon) &= (x + 1, \epsilon) \end{aligned}$$

$$\begin{aligned} \textit{prev}(x, \epsilon) &= (x - 1, \not\epsilon) \\ \textit{prev}(x, \not\epsilon) &= (x, \epsilon) \end{aligned}$$

Following the definitions in section 5.5.1, a position vector (q, q) in a simple situated expression contains the following position: (q, ϵ). Position vector (q, r), $q < r$, in a simple situated expression contains the following positions: $(q, \not\epsilon), \ldots, (r - 1, \not\epsilon)$.[6] Position vector (q, r), $q \leq r$, in a complex situated expression includes the following positions: $(q, \epsilon), \ldots, (r, \epsilon)$.

Let π_l be the left-most position and π_r be the right-most position contained in the position vectors of a situated expression Δ. Then situated expression Δ is said to be to the right of position π_l and all positions preceding π_l, and to the left of position $next(\pi_r)$ and all positions following $next(\pi_r)$. Furthermore, Δ is said to cross all positions between π_l and $next(\pi_r)$ (excluding π_l and $next(\pi_r)$).

6.3.1 Items

An item is a triple of the following form:

$$\langle\ \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_n,\ \mathrm{C},\ \pi\ \rangle$$

where A is a situated expression or the special symbol S′, $\mathrm{B}_1, \ldots, \mathrm{B}_n$ are situated expressions, C is a set containing expressions selected from $\mathrm{B}_1, \ldots,$ and B_n,[7] and π is

[6] Conceptually, positions (p, ϵ) between $(q, \not\epsilon)$ and $(r - 1, \not\epsilon)$ should not be included in the positions contained in position vector (q, r), but technically, this will not make a difference for the operation of the recognizer.

[7] For any Minimalist Grammar, $\mathrm{B}_1 \ldots \mathrm{B}_n$ is always a sequence of distinct situated expressions, so C is actually a set of occurrences of situated expressions.

a position in the input sentence.

6.3.2 Invariant

An item $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, C, π $\rangle$ embodies the following claim:

1. There is a sequence of situated expressions $E_1, \ldots, E_m$ such that:
   (a) Situated expression A is one of the situated expressions $E_1, \ldots, E_m$.
   (b) None of the situated expressions $E_i \neq A$, $1 \leq i \leq m$, crosses π.
   (c) Expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w.
   (d) Any expression e_{E_i}, $1 \leq i \leq m$, such that situated expression $E_i \neq A$ is to the left of π, is a lexical expression.
2. For every situated expression $B_j \in C$, $1 \leq j \leq n$, there is a sequence of situated expressions $F_1, \ldots, F_p$ such that:
   (a) None of the situated expressions F_i, $1 \leq i \leq p$, crosses π.
   (b) Expressions $e_{F_1}, \ldots, e_{F_p}$ are the leaves of a partial derivation tree with root e_{B_j}.
   (c) Any expression e_{F_i}, $1 \leq i \leq p$, such that situated expression F_i is to the left of π, is a lexical expression.
3. For every situated expression $B_j \notin C$, $1 \leq j \leq n$, there is a sequence of situated expressions $G_1, \ldots, G_q$ such that:
   (a) None of the situated expressions G_i, $1 \leq i \leq q$, crosses $next(\pi)$.
   (b) Expressions $e_{G_1}, \ldots, e_{G_q}$ are the leaves of a partial derivation tree with root e_{B_j}.

(c) Any expression e_{G_i}, $1 \leq i \leq p$, such that situated expression G_i is to the left of $next(\pi)$ is a lexical expression.

6.3.3 Axioms

For n the length of the sentence to be parsed, a lexical item $[s{:}\mathtt{c}]_s$ licenses the following axiom:

$$\langle\ S' \rightarrow [(0,\ n){:}\cdot\mathtt{c}]_s,\ C,\ (0, \epsilon)\ \rangle$$

where the content of C is determined as follows: $[(0,\ n){:}\cdot\mathtt{c}]_s \in C$ if, and only if, position $(0, \epsilon)$ is contained in the position vector (0, n), i.e., n = 0.

A lexical item $[s{:}\gamma\mathtt{c}]_s$, $\gamma \in \text{Cat}^+$, licenses the following axiom:

$$\langle\ S' \rightarrow [(0,\ n){:}\gamma\cdot\mathtt{c}]_c,\ \{\ [(0,\ n){:}\gamma\cdot\mathtt{c}]_c\ \},\ (0,\ \epsilon)\ \rangle.^{8}$$

6.3.4 Goal Items

Any item of the following form is a goal item, $\gamma \in \text{Cat}^*$, $t \in \{s, c\}$, n the length of the input sentence:

$$\langle\ S' \rightarrow [(0,\ n){:}\gamma\cdot\mathtt{c}]_t,\ \emptyset,\ (n,\ \epsilon)\ \rangle$$

6.3.5 Rules of Inference

Assume the input sentence is $w = w_1 \ldots w_n$. Position $(x, \cdot)$ stands for position (x, ϵ) or position $(x, \not\epsilon)$, $t \in \{s, c\}$.

[8] $[(0,\ n){:}\gamma\cdot\mathtt{c}]_c \in C$, because position $(0, \epsilon)$ is always included in position vector $(0, n)$.

6.3.5.1 Manipulating π

Next

Antecedent:

$$\langle\ \mathrm{S}' \rightarrow [(0,\ n){:}\gamma\cdot\mathsf{c}]_t,\ \emptyset,\ (x, \cdot)\ \rangle$$

such that:

$$(x, \cdot) < (n,\ \epsilon).^{9}$$

Consequent:

$$\langle\ \mathrm{S}' \rightarrow [(0,\ n){:}\gamma\cdot\mathsf{c}]_t,\ \mathrm{C},\ next(x, \cdot)\ \rangle$$

where:

1. $[(0,\ n){:}\gamma\cdot\mathsf{c}]_t \in \mathrm{C}$ if, and only if, position $next(x, \cdot)$ is contained in position vector $(0,\ n)$.

6.3.5.2 Predict Rules

Unmerge-1

Antecedent:

$$\langle\ \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_n,\ \mathrm{C} \cup \{\ [(\mathrm{p},\ \mathrm{q}){:=}\mathsf{x}\cdot\gamma,\ \mathrm{S}]_c\ \},\ (x,\ \epsilon)\ \rangle$$

such that:

[9] This ensures that the position mentioned in the consequent of the rule is at most $next(n-1,\ \not\epsilon) = (n,\ \epsilon)$, which means that the empty word at position n will be the last element sought for in the sentence. This is correct, since no sentence of length n will contain any words beyond the empty word at position $(n,\ \epsilon)$.

1. The left-most position contained in the position vectors of [(p, q):=x·γ, S]$_c$ is (x, ϵ).

2. There is a lexical item with syntactic features βx.

Consequent:

$\langle$ [(p, q):=x·γ, S]$_c$ $\rightarrow$ [(p, v):·=xγ]$_s$ [(v, q):β·x, S]$_t$, C′, (x, ϵ) $\rangle$

where:

1. $p \leq v \leq q$; t = s if $\beta = \emptyset$, t = c if $\beta \neq \emptyset$, and if t = s then S = $\emptyset$.

2. [(p, v):·=xγ]$_s$ $\in$ C′ if, and only if, position (x, ϵ) is contained in position vector (p, v).

3. [(v, q):β·x, S]$_t$ $\in$ C′ if, and only if, position (x, ϵ) is contained in one of the position vectors of [(v, q):β·x, S]$_t$.

Unmerge-2

Antecedent:

$\langle$ A $\rightarrow$ B$_1$...B$_n$, C $\cup$ { [(p, q):α=x·γ, S]$_c$ }, (x, ϵ) $\rangle$

such that:

1. The left-most position contained in the position vectors of [(p, q):α=x·γ, S]$_c$ is (x, ϵ).

2. There is a lexical item with syntactic features βx.

3. $\alpha \neq \epsilon$.

Consequent:

$\langle$ [(p, q):α=x·γ, S]$_c$ $\rightarrow$ [(v, q):α·=xγ, U]$_c$ [(p, v):β·x, V]$_t$, C′, (x, ϵ) $\rangle$

where:

1. $p \leq v \leq q$; U $\cup$ V = S; t = *s* if $\beta = \emptyset$, t = *c* if $\beta \neq \emptyset$, and if t = *s* then V = $\emptyset$.
2. [(v, q):α·=xγ, U]$_c$ $\in$ C′ if, and only if, position (x, ϵ) is contained in one of the position vectors of [(v, q):α·=xγ, U]$_c$.
3. [(p, v):β·x, V]$_t$ $\in$ C′ if, and only if, position (x, ϵ) is contained in one of the position vectors of [(p, v):β·x, V]$_t$.

Unmerge-3

Antecedent:

$\langle$ A $\rightarrow$ B$_1$ ... B$_n$, C $\cup$ { [(p, q):α=x·γ, S, (v, w):βx·δ, T]$_c$ }, (x, ϵ) $\rangle$

such that:

1. The left-most position contained in the position vectors of [(p, q):α=x·γ, S, (v, w):βx·δ, T]$_c$ is (x, ϵ).

Consequent:

$\langle$ [(p, q):α=x·γ, S, (v, w):βx·δ, T]$_c$ $\rightarrow$ [(p, q):α·=xγ, U]$_{t_1}$ [(v, w):β·xδ, V]$_{t_2}$, C′, (x, ϵ) $\rangle$

where:

1. U ∪ V = S ∪ T; t_1 = *s* if $\alpha = \emptyset$, t_1 = *c* if $\alpha \neq \emptyset$, and if t_1 = *s* then U = ∅; t_2 = *s* if $\beta = \emptyset$, t_2 = *c* if $\beta \neq \emptyset$, and if t_2 = *s* then V = ∅

2. [(p, q):$\alpha\cdot$=xγ, U]$_{t_1} \in C'$ if, and only if, position (x, ϵ) is contained in one of the position vectors of [(p, q):$\alpha\cdot$=xγ, U]$_{t_1}$.

3. [(v, w):$\beta\cdot$xδ, V]$_{t_2} \in C'$ if, and only if, position (x, ϵ) is contained in one of the position vectors of [(v, w):$\beta\cdot$xδ, V]$_{t_2}$.

Unmove-1

Antecedent:

$\langle$ A $\rightarrow$ B$_1$ ... B$_n$, C $\cup$ { [(p, q):α+y$\cdot\gamma$, S]$_c$ }, (x, ϵ) $\rangle$

such that:

1. The left-most position contained in the position vectors of [(p, q):α+y$\cdot\gamma$, S]$_c$ is (x, ϵ).

2. There is a lexical item with syntactic features β-f.

3. $\alpha \neq \epsilon$.

Consequent:

$\langle$ [(p, q):α+y$\cdot\gamma$, S]$_c$ $\rightarrow$ [(v, q):$\alpha\cdot$+yγ, (p, v):$\beta\cdot$-y, S]$_c$, C′, (x, ϵ) $\rangle$

where:

1. $p \leq v \leq q$.

2. C′ = { [(v, q):α·+yγ, (p, v):β·-y, S]$_c$ }.[10]

Unmove-2

Antecedent:

$\langle$ A $\rightarrow$ B$_1$ ... B$_n$, C $\cup$ { [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$ }, (x, ϵ) $\rangle$

such that:

1. The left-most position contained in the position vectors of [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$ is (x, ϵ).
2. $\alpha \neq \epsilon$, $\beta \neq \epsilon$.

Consequent:

$\langle$ [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$ $\rightarrow$ [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$, C′, (x, ϵ) $\rangle$

where:

1. C′ = { [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$ }.[11]

The two unmove rules should not add any items violating the Shortest Movement Constraint to the chart, cf. section 5.3.3 of the previous chapter.

[10] If position (x, ϵ) is contained in one of the position vectors of [(p, q):α+y·γ, S]$_c$, it is also contained in one of the position vectors of [(v, q):α·+yγ, (p, v):β·-y, S]$_c$.

[11] If position (x, ϵ) is contained in one of the position vectors of [(p, q):α+y·γ, S, (v, w):β-y·δ, T]$_c$, it is also contained in one of the position vectors of [(p, q):α·+yγ, S, (v, w):β·-yδ, T]$_c$.

6.3.5.3 Complete and Scan Rule

Given a set of situated expressions C, let C_s be the subset of C containing all simple situated expressions in C, and let C_c be the subset of C containing all complex situated expressions in C.

Up

Antecedent:

$$\langle\ A \rightarrow B_1 \ldots B_n,\ C,\ (x, \cdot)\ \rangle$$

$$\langle\ D_1 \rightarrow E_{11} \ldots E_{1n_1},\ \emptyset,\ (x, \cdot)\ \rangle$$

$$\ldots$$

$$\langle\ D_m \rightarrow E_{m1} \ldots E_{mn_m},\ \emptyset,\ (x, \cdot)\ \rangle$$

such that:

1. $\bigcup_{i=1}^{m} D_i = C_c$.
2. For every $B_i \in C_s$, there is an expression $e_{B_i} \in$ Lex.

Consequent:

$$\langle\ A \rightarrow B_1 \ldots B_n,\ \emptyset,\ (x, \cdot)\ \rangle$$

6.3.5.4 Updating π

Down

Antecedent:

$$\langle\ A \rightarrow B_1 \ldots B_n,\ C \cup \{\ B_i\ \},\ (x, \cdot)\ \rangle$$

$\langle\ \mathrm{B}_i \rightarrow \mathrm{D}_1 \ldots \mathrm{D}_m,\ \emptyset,\ (q,\ \cdot)\ \rangle$

such that:

1. The right-most position to the left of $(x,\ \cdot)$ contained in the position vectors of B_i is $(q,\ \cdot)$.

Consequent:

$\langle\ \mathrm{B}_i \rightarrow \mathrm{D}_1 \ldots \mathrm{D}_m,\ \mathrm{C}',\ (x,\ \cdot)\ \rangle$

where:

1. $\mathrm{D}_i \in \mathrm{C}'$ if, and only if the position $(x,\ \cdot)$ is contained in the position vectors of D_i, $1 \leq i \leq m$.

6.4 Example

The items given below are the items produced by the recognizer that lead to a goal item for the sentence ab according to grammar H as given in section 6.1 of this chapter. For each item, the underlined situated expressions form the contents of set C for that item. The items are grouped into five 'columns', one for each value of π. Note that for this particular grammar, most items appear in all columns.

I. π = (0, ϵ)

1a. $\mathrm{S}' \rightarrow \underline{[(0,\ 2)\text{:=}\texttt{b +f}\cdot\texttt{c}]_c}$ axiom

2a. $[(0,\ 2)\text{:=}\texttt{b +f}\cdot\texttt{c}]_c \rightarrow \underline{[(2,\ 2)\text{:=}\texttt{b}\cdot\texttt{+f c},\ (0,\ 2)\text{:=}\texttt{b +f b}\cdot\texttt{-f}]_c}$ Unmove-1(1a)

3a. $[(2,\ 2)\text{:=}\texttt{b}\cdot\texttt{+f c},\ (0,\ 2)\text{:=}\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2,\ 2)\text{:}\cdot\text{=}\texttt{b +f c}]_s\ \underline{[(0,\ 2)\text{:=}\texttt{b +f}\cdot\texttt{b -f}]_c}$

Unmerge-3(2a)

4a. $[(0, 2):=\texttt{b}\cdot\texttt{+f}\cdot\texttt{b -f}]_c \rightarrow \underline{[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c}$ Unmove-1(3a)

5a. $[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s\ [(0, 1):\cdot\texttt{b -f}]_s$

Unmerge-3(4a)

6a. $[(0, 2):=\texttt{b}\cdot\texttt{+f}\cdot\texttt{b -f}]_c \rightarrow [(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c$ Up(4a, 5a)

7a. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s\ [(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c$

Up(3a, 6a)

8a. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow [(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c$ Up(2a, 7a)

9a. $S' \rightarrow [(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c$ Up(1a, 8a)

II. $\pi = (0, \not\epsilon)$

1b. $S' \rightarrow \underline{[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c}$ Next(9a)

2b. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow \underline{[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c}$ Down(1b, 8a)

3b. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s\ \underline{[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c}$

Down(2b, 7a)

4b. $[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c \rightarrow \underline{[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c}$ Down(3b, 6a)

5b. $[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s\ \underline{[(0, 1):\cdot\texttt{b -f}]_s}$

Down(4b, 5a)

5b′. $[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s\ [(0, 1):\cdot\texttt{b -f}]_s$ Up(5b)

6b. $[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c \rightarrow [(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c$ Up(4b, 5b′)

7b. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s\ [(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c$

Up(3b, 6b)

8b. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow [(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c$ Up(2b, 7b)

9b. $S' \rightarrow [(0, 2):=\texttt{b} + \texttt{f} \cdot \texttt{c}]_c$ Up(1b, 8b)

III. $\pi = (1, \epsilon)$

1c. $S' \rightarrow \underline{[(0, 2):=\texttt{b +f·c}]_c}$ Next(9b)

2c. $[(0, 2):=\texttt{b +f·c}]_c \rightarrow \underline{[(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c}$ Down(1c, 8b)

3c. $[(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s \underline{[(0, 2):=\texttt{b +f·b -f}]_c}$ Down(2c, 7b)

4c. $[(0, 2):=\texttt{b +f·b -f}]_c \rightarrow \underline{[(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c}$ Down(3c, 6b)

5c. $[(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s [(0, 1):\cdot\texttt{b -f}]_s$ Down(4c, 5b)

6c. $[(0, 2):=\texttt{b +f·b -f}]_c \rightarrow [(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c$ Up(4c, 5c)

7c. $[(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s [(0, 2):=\texttt{b +f·b -f}]_c$ Up(3c, 6c)

8c. $[(0, 2):=\texttt{b +f·c}]_c \rightarrow [(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c$ Up(2c, 7c)

9c. $S' \rightarrow [(0, 2):=\texttt{b +f·c}]_c$ Up(1c, 8c)

IV. $\pi = (1, \not\epsilon)$

1d. $S' \rightarrow \underline{[(0, 2):=\texttt{b +f·c}]_c}$ Next(9c)

2d. $[(0, 2):=\texttt{b +f·c}]_c \rightarrow \underline{[(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c}$ Down(1d, 8c)

3d. $[(2, 2):=\texttt{b·+f c}, (0, 2):=\texttt{b +f b·-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s \underline{[(0, 2):=\texttt{b +f·b -f}]_c}$ Down(2d, 7c)

4d. $[(0, 2):=\texttt{b +f·b -f}]_c \rightarrow \underline{[(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c}$ Down(3d, 6c)

5d. $[(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c \rightarrow \underline{[(1, 2):\cdot=\texttt{b +f b -f}]_s} [(0, 1):\cdot\texttt{b -f}]_s$ Down(4d, 5c)

5d′. $[(1, 2):=\texttt{b·+f b -f}, (0, 1):\texttt{b·-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s [(0, 1):\cdot\texttt{b -f}]_s$ Up(5d)

6d. $[(0, 2):=\texttt{b}\cdot\texttt{+f}\cdot\texttt{b -f}]_c \rightarrow [(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c$ Up(4d, 5d′)

7d. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s\ [(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c$ Up(3d, 6d)

8d. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow [(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c$ Up(2d, 7d)

9d. $S' \rightarrow [(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c$ Up(1d, 8d)

V. $\pi = (2, \epsilon)$

1e. $S' \rightarrow \underline{[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c}$ Next(9d)

2e. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow \underline{[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c}$ Down(1e, 8d)

3e. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow \underline{[(2, 2):\cdot=\texttt{b +f c}]_s}\ \underline{[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c}$ Down(2e, 7d)

4e. $[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c \rightarrow \underline{[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c}$ Down(3e, 6d)

5e. $[(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c \rightarrow [(1, 2):\cdot=\texttt{b +f b -f}]_s\ [(0, 1):\cdot\texttt{b -f}]_s$ Down(4e, 5d)

6e. $[(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c \rightarrow [(1, 2):=\texttt{b}\cdot\texttt{+f b -f}, (0, 1):\texttt{b}\cdot\texttt{-f}]_c$ Up(4e, 5d)

7e. $[(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c \rightarrow [(2, 2):\cdot=\texttt{b +f c}]_s\ [(0, 2):=\texttt{b +f}\cdot\texttt{b -f}]_c$ Up(3e, 6e)

8e. $[(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c \rightarrow [(2, 2):=\texttt{b}\cdot\texttt{+f c}, (0, 2):=\texttt{b +f b}\cdot\texttt{-f}]_c$ Up(2e, 7e)

9e. $S' \rightarrow [(0, 2):=\texttt{b +f}\cdot\texttt{c}]_c$ Up(1e, 8e)

Item 9e is a goal item.

6.5 Proofs of Correctness and Complexity

6.5.1 Proof of Soundness

Recall that the recognizer is sound if every axiom and derivable item makes a true claim when interpreted as in section 6.3.2. We will first show that the axioms meet this condition. Next we will show that if the items serving as the antecedents of the rules of inference represent true claims, then so do their consequents.

Because the items of the recognizer are very informative, some of the proofs in this section are quite extended. However, each proof draws upon a limited number of rather simple deductive steps. To avoid any confusion between the situated expressions in an item and the expressions generated by a Minimalist Grammar, in the following situated expressions will be referred to as subitems.

6.5.1.1 Axioms

Regarding any axiom $\langle$ S′ $\rightarrow$ [(0, n):$\gamma\cdot$c]$_t$, C, $(0, \epsilon)$ $\rangle$, $t \in \{s, c\}$, such that subitem [(0, n):$\cdot$c]$_t \in$ C, consider the sequence consisting of the one subitem F$_1$ = [(0, n):$\cdot$c]$_t$. By definition, F$_1$ is to the right of position $(0, \epsilon)$. Hence, F$_1$ does not cross $(0, \epsilon)$ (2a).[12] Expression e_{F_1} is the leaf of the trivial, partial derivation tree with root e_{F_1}, namely, the tree consisting of the single node e_{F_1} (2b). Since F$_1$ is to the right of position $(0, \epsilon)$, whether F$_1$ is lexical or not is not an issue (2c).[13]

For an axiom $\langle$ S′ $\rightarrow$ [(0, n):$\gamma\cdot$c]$_s$, C, $(0, \epsilon)$ $\rangle$ such that subitem [(0, n):$\cdot$c]$_s \notin$ C, it must be the case that position $(0, \epsilon)$ is not contained in the position vector (0, n), i.e. $n > 0$. Consider the sequence consisting of the one subitem F$_1$ = [(0, n):$\gamma\cdot$c]$_s$. By definition, $(0, \not\epsilon)$ is the left-most position contained in the position vector of F$_1$, so F$_1$

[12] The parenthesized number letter sequences refer to corresponding parts of the invariant.

[13] For the axioms we can ignore the first part of the invariant, because S′ is a special symbol.

does not cross $(0, \not\epsilon)$ (3a). Trivially, expression e_{F_1} is the leaf of the partial derivation tree having no nodes but the root e_{F_1} (2b). F_1 is to the right of position $(0, \not\epsilon)$, whence it is not required to be lexical (3c).

6.5.1.2 Rules of Inference

Next: consider first the case in which the antecedent of the rule is the item $\langle$ S′ $\rightarrow$ [(0, n):$\gamma\cdot$C]$_t$, $\emptyset$, $(x, \not\epsilon)$ $\rangle$ and the consequent is the item $\langle$ S′ $\rightarrow$ [(0, n):$\gamma\cdot$C]$_t$, C, (x+1, ϵ) $\rangle$ where B = [(0, n):$\gamma\cdot$C]$_t$ $\in$ C.

According to the invariant, it follows from the antecedent that there is a sequence of subitems $F_1, \ldots, F_q$ such that (3a) none of the subitems F_i, $1 \leq i \leq q$, crosses (x+1, ϵ); (3b) expressions $e_{F_1}, \ldots, e_{F_q}$ are the leaves of a partial derivation tree with root e_B; (3c) any expression e_{F_i}, $1 \leq i \leq p$, which appears to the left of (x+1, ϵ) is a lexical expression.[14]

Since B $\in$ C, it follows immediately that the sequence of subitems $F_1, \ldots, F_q$ satisfies the invariant for the consequent (3a, 3b, 3c).

Next, assume B = [(0, n):$\gamma\cdot$C]$_t$ $\notin$ C and $t = c$. Since B $\notin$ C, position vector (0, n) does not contain position $(x, \not\epsilon)$. This happens only when $x \geq n$, but then Next will not apply, since $(x, \not\epsilon) \not< (n, \epsilon)$ for $x \geq n$. Hence, this case does not arise.

Finally, assume B = [(0, n):$\gamma\cdot$C]$_t$ $\notin$ C and $t = s$.[15] Since B is a simple subitem, the sequence of subitems $F_1, \ldots, F_q$ provided by the antecedent reduces to the one-element sequence F_1, and B = F_1. Since F_1 does not cross (x+1, ϵ), it is either to the left of this position or to the right of this position. If F_1 is to the left of position (x+1, ϵ), then F_1 cannot cross position (x+1, $\not\epsilon$), which is to the right of (x+1, ϵ) (3a). Trivially, e_B is the

[14] Because S′ is a special symbol, the first part of the invariant will be ignored for both the antecedent and the consequent.

[15] The various cases for $\pi = (x, \epsilon)$ are left as an exercise for the reader.

leaf of the partial derivation tree consisting of the one node, hence also root, e_B (3b). If, as assumed, F_1 is to the left of position (x+1, ϵ), in which case it will also be to the left of (x+1, $\not\epsilon$), then it follows from the antecedent that expression e_B, is lexical (3c). If F_1 is to the right of position (x+1, ϵ), then it must be to the right of position (x+1, $\not\epsilon$) as well, because position vector (0, n) does not contain position (x+1, ϵ), since B $\notin$ C. Hence, F_1 does not cross position (x+1, $\not\epsilon$) (3a). Trivially, e_B is the leaf of the partial derivation tree consisting of the one node, hence also root, e_B (3b). F_1 is to the right of (x+1, ϵ), so the issue regarding lexicality is moot.

For the Predict rules, we will show that the rule of inference Unmerge-2 is sound. Soundness of the other rules can be shown in a similar fashion.

Unmerge-2: regarding the antecedent of the rule Unmerge-2, $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, C, (x, ϵ) $\rangle$, assume that n = 2, B_1 is a complex subitem, $B_1 \notin$ C, B_2 = [(p, q):α=x$\cdot\gamma$, S]$_c \in$ C, A $\neq$ S′; regarding the consequent of the rule, $\langle$ [(p, q):α=x$\cdot\gamma$, S]$_c \rightarrow$ [(v, q):$\alpha\cdot$=xγ, U]$_c$ [(p, v):$\beta\cdot$x, V]$_t$, C′, (x, ϵ) $\rangle$, assume that [(v, q):$\alpha\cdot$=xγ, U]$_c \notin$ C′, [(p, v):$\beta\cdot$x, V]$_t$ $\in$ C′.[16]

Applying the invariant to the antecedent, it follows that there is a sequence of subitems E_1, …, E_m such that: (1a) subitem A is one of the subitems E_1, …, E_m – assume A = E_j; (1b) none of the subitems $E_i \neq$ A, $1 \leq i \leq m$, crosses (x, ϵ); (1c) expressions e_{E_1}, …, e_{E_m} are the leaves of a partial derivation tree of w; (1d) any expression e_{E_i}, $1 \leq i \leq m$, such that subitem $E_i \neq$ A is to the left of (x, ϵ), is a lexical expression.

Furthermore, since subitem B_2 = [(p, q):α=x$\cdot\gamma$, S]$_c \in$ C, there is a sequence of subitems F_1, …, F_p such that: (2a) none of the subitems F_i, $1 \leq i \leq p$, crosses (x, ϵ); (2b) expressions e_{F_1}, …, e_{F_p} are the leaves of a partial derivation tree with root e_{B_2}; (2c) any expression e_{F_i}, $1 \leq i \leq p$, such that subitem F_i is to the left of (x, ϵ), is a

[16] Some of these assumptions cannot be made without loss of generality. The reader is invited to provide the proofs for the cases which do not fall under the current assumptions.

lexical expression.

Moreover, since $B_1 \notin C$, there is a sequence of subitems $G_1, \ldots, G_q$ such that: (3a) none of the subitems G_i, $1 \leq i \leq q$, crosses $(x, \not\epsilon)$; (3b) expressions $e_{G_1}, \ldots, e_{G_q}$ are the leaves of a partial derivation tree with root e_{B_1}; (3c) any expression e_{G_i}, $1 \leq i \leq p$, such that subitem G_i is to the left of $(x, \not\epsilon)$, is a lexical expression.

Now consider the sequence of subitems $E_1, \ldots, E_{j-1}, G_1, \ldots, G_q, B_2, E_{j+1}, \ldots, E_m$. Obviously, subitem B_2 is one of the subitems of this sequence (1a). It follows from the antecedent that none of the subitems E_i in this sequence crosses (x, ϵ), $1 \leq i \leq m$, $i \neq j$. It also follows from the antecedent that none of the subitems G_i, $1 \leq i \leq q$, crosses $(x, \not\epsilon)$. Thus, for a subitem G_i, $1 \leq i \leq q$, to cross (x, ϵ), it has to be to the immediate left of $(x, \not\epsilon)$. According to the invariant, for any subitem G_i, $1 \leq i \leq q$, to the left of $(x, \not\epsilon)$, its expression e_{G_i} is lexical. Since lexical expressions cover exactly one (empty or non-empty) word, subitem G_i does not cross any position. Hence, none of the subitems in the sequence $E_1, \ldots, E_{i-1}, G_1, \ldots, G_q, B_2, E_{i+1}, \ldots, E_m$, excepting B_2, crosses (x, ϵ) (1b).

The antecedent implies that expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w. Inspection of the rules of inference that introduce items containing rewrites of the form $A \rightarrow B_1\ B_2$ shows that either e_A = Merge-1(e_{B_1}, e_{B_2}), or e_A = Merge-2(e_{B_1}, e_{B_2}), or e_A = Merge-3(e_{B_1}, e_{B_2}). It follows from the antecedent that expressions $e_{G_1}, \ldots, e_{G_q}$ are the leaves of a partial derivation tree with root e_{B_1}. Assembling the various parts, we see that the expressions $e_{E_1}, \ldots, e_{E_{i-1}}, e_{G_1}, \ldots, e_{G_q}$, $e_{B_2}, e_{E_{i+1}}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w (1c).

According to the antecedent any expression e_{E_i}, $1 \leq i \leq m$, such that subitem $E_i \neq A$ is to the left of (x, ϵ), is a lexical expression. Also according to the antecedent, any expression e_{G_i}, $1 \leq i \leq p$, such that subitem G_i is to the left of $(x, \not\epsilon)$, is a lexical expression. Obviously, if some subitem G_i, $1 \leq i \leq p$, is to the left of (x, ϵ), it is also

to the left of $(x, \not\epsilon)$, so expression e_{G_i} is lexical. Hence, any of the expressions $e_{E_1}, \ldots, e_{E_{i-1}}, e_{G_1}, \ldots, e_{G_q}, e_{E_{i+1}}, \ldots, e_{E_m}$ whose subitem is to the left of (x, ϵ), is a lexical expression (1d).

It follows from the conditions on the applicability on the rule Unmerge-2 that (x, ϵ) is the left-most position contained in the position vectors of subitem [(p, q):α=x$\cdot\gamma$, S]$_c$, i.e., [(p, q):α=x$\cdot\gamma$, S]$_c$ is to the right of position (x, ϵ). Then, by the way the position vectors are split both subitem [(p, v):$\beta\cdot$x, V]$_t$ and subitem [(v, q):$\alpha\cdot$=xγ, U]$_c$ are also to the right of position (x, ϵ).

Since subitem C_1 =[(p, v):$\beta\cdot$x, V]$_t$ *in* C′ is to the right of position (x, ϵ), it does not cross position (x, ϵ) (2a). Obviously, the expression e_{C_1} is a leaf of the partial derivation tree consisting of the single node, hence also root, e_{C_1} (2b). Since subitem C_1 is to the right of position (x, ϵ), it does not matter whether expression e_{C_1} is lexical or not (2c).

Since subitem C_2 = [(v, q):$\alpha\cdot$=xγ, U]$_c \notin$ C′, its position vectors do not contain position (x, ϵ). Since subitem C_2 is to the right of (x, ϵ), it must be to the right of $(x, \not\epsilon)$. Consequently, subitem C_2 does not cross position $(x, \not\epsilon)$ (3a). The expression e_{C_2} is a leaf and the root of the partial derivation tree consisting of the single node e_{C_2} (3b). Since subitem C_2 is to the right of position $(x, \not\epsilon)$, it is irrelevant whether expression e_{C_1} is lexical (3c).

Up: assume with regard to the first item of the antecedent of the rule Up, $\langle$ A $\rightarrow$ B$_1 \ldots$ B$_n$, C, $(x, \cdot)$ $\rangle$, that n = 2, B$_1 \in$ C$_c$, B$_2 \in$ C$_s$, and $(x, \cdot) = (x, \not\epsilon)$. Then for the rule to apply, there must be an another item $\langle$ B$_1 \rightarrow$ D$_1 \ldots$ D$_m$, C′, $(x, \not\epsilon)$ $\rangle$. For this item, C′ = $\emptyset$, and assume that m = 2, D$_1$ complex and D$_2$ simple. The consequent of the rule is $\langle$ A $\rightarrow$ B$_1 \ldots$ B$_n$, C″, $(x, \not\epsilon)$ $\rangle$, C″ = $\emptyset$.

Application of the invariant to item $\langle$ A $\rightarrow$ B$_1 \ldots$ B$_2$, C, $(x, \not\epsilon)$ $\rangle$ tells us that there is a sequence of subitems E$_1, \ldots,$ E$_m$ such that: (1a) subitem A is one of the subitems

$E_1, \ldots, E_m$; assume A = E_j; (1b) none of the subitems $E_i \neq A$, $1 \leq i \leq m$, crosses (x, $\not\epsilon$); (1c) expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w; (1d) any expression e_{E_i}, $1 \leq i \leq m$, such that subitem $E_i \neq A$ is to the left of (x, $\not\epsilon$), is a lexical expression.

Furthermore, since subitem $B_1 \in C$, there is a sequence of subitems $F_1, \ldots, F_p$ such that: (2a) none of the subitems F_i, $1 \leq i \leq p$, crosses (x, $\not\epsilon$); (2b) expressions e_{F_1}, $\ldots$, e_{F_p} are the leaves of a partial derivation tree with root e_{B_1}; (2c) any expression e_{F_i}, $1 \leq i \leq p$, such that subitem F_i is to the left of (x, $\not\epsilon$), is a lexical expression.

Moreover, since $B_2 \in C$, there is a sequence of subitems $G_1, \ldots, G_q$ such that: (3a) none of the subitems G_i, $1 \leq i \leq q$, crosses (x, $\not\epsilon$). (3b) expressions $e_{G_1}, \ldots, e_{G_q}$ are the leaves of a partial derivation tree with root e_{B_2}. (3c) any expression e_{G_i}, $1 \leq i \leq p$, such that subitem G_i is to the left of (x, $\not\epsilon$), is a lexical expression. As B_2 is a simple subitem, it follows that q = 1 and $G_1 = B_2$.

Similarly, for item $\langle\, B_1 \rightarrow D_1 \ldots D_2, \emptyset, (x, \not\epsilon)\, \rangle$, there is a sequence of subitems H_1, $\ldots$, H_m such that: (1a) subitem B_1 is one of the subitems $H_1, \ldots, H_m$; assume $B_1 = H_j$; (1b) none of the subitems $H_i \neq B_1$, $1 \leq i \leq m$, crosses (x, $\not\epsilon$); (1c) expressions $e_{H_1}, \ldots, e_{H_m}$ are the leaves of a partial derivation tree of w; (1d) any expression e_{H_i}, $1 \leq i \leq m$, such that subitem $H_i \neq B_1$ is to the left of (x, $\not\epsilon$), is a lexical expression.

Moreover, since $D_1 \notin C'$, there is a sequence of subitems $L_1, \ldots, L_q$ such that: (3a) none of the subitems L_i, $1 \leq i \leq q$, crosses (x+1, ϵ). (3b) expressions $e_{L_1}, \ldots, e_{L_q}$ are the leaves of a partial derivation tree with root e_{D_1}. (3c) any expression e_{L_i}, $1 \leq i \leq p$, such that subitem L_i is to the left of (x+1, ϵ), is a lexical expression.

Also, since $D_2 \notin C'$, there is a sequence of subitems $M_1, \ldots, M_r$ such that: (3a) none of the subitems G_i, $1 \leq i \leq r$, crosses ($x + 1$, ϵ). (3b) expressions $e_{M_1}, \ldots, e_{M_r}$ are the leaves of a partial derivation tree with root e_{D_2}. (3c) any expression e_{G_i}, $1 \leq i \leq r$, such that subitem M_i is to the left of ($x + 1$, ϵ), is a lexical expression. Since D_2

is a simple subitem, r = 1 and $M_1 = D_2$.

Now, meditate on the sequence of subitems $E_1, \ldots, E_m$. It follows immediately from the first item of the antecedent that this sequence is such that subitem A is one of the subitems $E_1, \ldots, E_m$: assume $A = E_j$ (1a); none of the subitems $E_i \neq A$, $1 \leq i \leq m$, crosses $(x, \not\epsilon)$ (1b); expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w (1c); any expression e_{E_i}, $1 \leq i \leq m$, such that subitem $E_i \neq A$ is to the left of $(x, \not\epsilon)$, is a lexical expression (1d).

Next, consider subitem $B_1 \notin C''$. It follows from the second item of the antecedent that none of the subitems in the sequence $L_1, \ldots, L_q$, M_1 crosses $(x + 1, \epsilon)$ (3a). Since, according to the second item expressions $e_{L_1}, \ldots, e_{L_q}$ are the leaves of a partial derivation tree with root e_{D_1}, expression e_{M_1} is the leaf of a partial derivation tree with root e_{D_2}, and either e_{B_1} = Merge-1(e_{D_1}, e_{D_2}), or e_{B_1} = Merge-2(e_{D_1}, e_{D_2}), or e_{B_1} = Merge-3(e_{D_1}, e_{D_2}), we have that the sequence of expressions $e_{L_1}, \ldots, e_{L_q}, e_{M_1}$ are the leaves of a partial derivation tree with root e_{B_1} (3b). Also, any expression e_{L_i}, $1 \leq i \leq p$, such that subitem L_i is to the left of (x+1, ϵ), is a lexical expression, and expression e_{M_1} is a lexical expression if subitem M_1 is to the left of (x+1, ϵ) (3c). This is an immediate consequence of applying the invariant to the second item of the antecedent.

Finally, consider the subitem $B_2 \notin C''$. According to the conditions on the applicability of rule Up, the expression e_{B_2} is lexical. Since lexical expressions cover exactly one empty or non-empty word, their corresponding subitems do not cross any positions. Thus, subitem B_2 does not cross (x+1, ϵ) (3a). Expression e_{B_2} is the leaf of the trivial partial derivation tree with root e_{B_2} (3b). Since expression e_{B_2} is lexical anyway, it does not matter whether subitem B_2 is to the left or to the right of position (x+1, ϵ) (3c).[17]

[17] In fact, since $B_2 \in C$, the single position vector of B_2 contains the position $(x, \not\epsilon)$. Therefore, since e_{B_2} is lexical, the position vector of B_2 is (x, x+1). Thus subitem e_{B_2} actually is to the immediate left of position (x+1, ϵ).

The case where $(x, \cdot) = (x{+}1, \epsilon)$ is dealt with similarly.

Down: assume that the first item of the antecedent is $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, C $\}$, (x, ϵ) $\rangle$, where n = 2, $B_1 \in$ C, $B_2 \in$ C; assume that the first item of the antecedent is $\langle$ $B_2 \rightarrow$ $D_1 \ldots D_m$, C′, (q, ϵ) $\rangle$, where m = 1, $D_1 \in$ C′, and $q = x - 2$. The item in the consequent is$\langle$ $B_2 \rightarrow D_1$, C″, (x, ϵ) $\rangle$

Applying the invariant to the first item of the antecedent, it follows that there is a sequence of subitems $E_1, \ldots, E_m$ such that: (1a) subitem A is one of the subitems E_1, $\ldots$, E_m; assume A = E_j; (1b) none of the subitems $E_i \neq$ A, $1 \leq i \leq m$, crosses (x, ϵ); (1c) expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w; (1d) any expression e_{E_i}, $1 \leq i \leq m$, such that subitem $E_i \neq$ A is to the left of (x, ϵ), is a lexical expression.

Furthermore, since subitem $B_1 \in$ C, there is a sequence of subitems $F_1, \ldots, F_p$ such that: (2a) none of the subitems F_i, $1 \leq i \leq p$, crosses (x, ϵ); (2b) expressions e_{F_1}, $\ldots$, e_{F_p} are the leaves of a partial derivation tree with root e_{B_1}; (2c) any expression e_{F_i}, $1 \leq i \leq p$, such that subitem F_i is to the left of (x, ϵ), is a lexical expression.

Also, since subitem $B_2 \in$ C, there is a sequence of subitems $G_1, \ldots, G_p$ such that: (2a) none of the subitems G_i, $1 \leq i \leq p$, crosses (x, ϵ); (2b) expressions $e_{G_1}, \ldots, e_{G_p}$ are the leaves of a partial derivation tree with root e_{G_2}; (2c) any expression e_{G_i}, $1 \leq i \leq p$, such that subitem G_i is to the left of (x, ϵ), is a lexical expression.

Applying the invariant to the second item of the antecedent gives a sequence of subitems $H_1, \ldots, H_m$ such that: (1a) subitem B_2 is one of the subitems $H_1, \ldots, H_m$; assume $B_2 = H_j$; (1b) none of the subitems $H_i \neq$ A, $1 \leq i \leq m$, crosses $(x - 2, \epsilon)$; (1c) expressions $e_{H_1}, \ldots, e_{H_m}$ are the leaves of a partial derivation tree of w; (1d) any expression e_{H_i}, $1 \leq i \leq m$, such that subitem $H_i \neq B_2$ is to the left of $(x - 2, \epsilon)$, is a lexical expression.

Furthermore, since $D_1 \notin$ C, there is a sequence of subitems $K_1, \ldots, K_q$ such that:

(3a) none of the subitems K_i, $1 \leq i \leq q$, crosses $(x - 2, \not\epsilon)$. (3b) expressions $e_{K_1}, \ldots, e_{K_q}$ are the leaves of a partial derivation tree with root e_{D_1}. (3c) any expression e_{K_i}, $1 \leq i \leq p$, such that subitem K_i is to the left of $(x - 2, \not\epsilon)$, is a lexical expression.

Consider the sequence of subitems $E_1, \ldots, E_{i-1}, F_1, \ldots, F_p, B_2\ E_{i+1}, \ldots, E_m$. Obviously, subitem B_2 is one of the subitems of this sequence (1a). It follows from the two items of the antecedent that none of the subitems $E_1, \ldots, E_{j-1}, F_1, \ldots, F_p, E_{j+2}, \ldots, E_m$ crosses (x, ϵ) (1b).

The antecedent implies that expressions $e_{E_1}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w. Inspection of the rules of inference that introduce items containing rewrites of the form $A \rightarrow B_1\ B_2$ shows that either e_A = Merge-1(e_{B_1}, e_{B_2}), or e_A = Merge-2(e_{B_1}, e_{B_2}), or e_A = Merge-3(e_{B_1}, e_{B_2}). It follows from the antecedent that expressions $e_{F_1}, \ldots, e_{F_p}$ are the leaves of a partial derivation tree with root e_{B_1}. Assembling the various parts, we see that the expressions $e_{E_1}, \ldots, e_{E_{i-1}}, e_{F_1}, \ldots, e_{F_p}, e_{B_2}, e_{E_{i+1}}, \ldots, e_{E_m}$ are the leaves of a partial derivation tree of w (1c). According to the two items of the antecedent, any expression e_{E_j}, $1 \leq j \leq m$, $j \neq i$, such that E_j is to the left of (x, ϵ), and any expression e_{F_j}, $1 \leq j \leq p$, such that E_j is to the left of (x, ϵ), is a lexical expression (1d).

Consider the sequence of subitems $K_1, \ldots, K_q$. It follows from the second item of the antecedent that none of these subitems crosses position $(x - 2, \not\epsilon)$. Hence, any subitem K_i, $1 \leq i \leq q$, is either to the left of $(x - 2, \not\epsilon)$ or to the right of $(x - 2, \not\epsilon)$. If subitem K_i is to the left of $(x - 2, \not\epsilon)$, it is also to the left of (x, ϵ), whence it does not cross (x, ϵ). If subitem K_i is to the right of $(x - 2, \not\epsilon)$, it also does not cross (x, ϵ): for K_i to cross (x, ϵ), it would have to contain a position between $(x - 2, \not\epsilon)$ and (x, ϵ) (and position (x, ϵ) or a position to the right of (x, ϵ)), but then position $(x - 2, \epsilon)$ would not be the right-most position to the left of (x, ϵ) contained in K_i, and consequently, position $(x - 2, \epsilon)$ would not be the right-most position to the left of (x, ϵ) contained in

B_2, violating the condition on the applicability of rule Down. Hence, no subitem K_i, $1 \leq i \leq q$, crosses position (x, ϵ) (2a).

The second item of the antecedent gives us immediately that expressions $e_{K_1}, \ldots, e_{K_q}$ are the leaves of a partial derivation tree with root e_{D_1} (2b).

As mentioned above, any subitem K_i, $1 \leq i \leq q$, crosses neither position $(x-2, \not\epsilon)$, nor position (x, ϵ). So any subitem K_i to the left of (x, ϵ) is either to the left of $(x-2, \not\epsilon)$, or between positions $(x-2, \not\epsilon)$ and (x, ϵ). However, it is impossible for K_i to be between $(x-2, \not\epsilon)$ and (x, ϵ), because in that case, K_i, and hence B_2, would contain a position between $(x-2, \not\epsilon)$ and (x, ϵ), wherefore position $(x-2, \epsilon)$ would no longer be the right-most position to the left of (x, ϵ) contained in B_2, violating the condition on the applicability of rule Down. If K_i is to the left of $(x-2, \not\epsilon)$, then expression e_{K_i} is lexical, according to the invariant applied to the second item of the antecedent (2c).

6.5.2 Proof of Completeness

The Earley-style recognizer presented in this chapter is complete for a Minimalist Grammar G if it generates a goal item for every sentence $w = w_1 \ldots w_n \in$ L(G). Consider any pair (w, T) such that $w \in$ L(G) and T is derivation tree of w in G without recursion, i.e., T does not contain two occurrences of the same expression such that one dominates the other.[18] Assume first that T has just one node. Under the assumption that a lexical item covers exactly one word, this means that sentence w consists of one word, which may be empty, and that the lexicon of G includes the item $[w{:}\mathtt{c}]_s$.

[18] For every $w \in$ L(G) a derivation tree with this property exists. Let T′ be a derivation tree with two occurrences, a_1 and a_2, of the same expression a, such that a_1 dominates a_2. Because of the non-erasure property of Minimalist Grammars the phonetic material of expression a_1 must contain the phonetic material of expression a_2 and the phonetic material of all the leaves in T′ dominated by a_1 but not by a_2. Now expression a_1 can only be identical to expression a_2 if all the leaves in T′ dominated by a_1 but not by a_2 are phonetically empty. But then the expressions between a_1 and a_2 can be removed and the two occurrences fused into one to obtain another derivation tree for w. The recursion thus removed will reappear when the derivation trees are retrieved from the chart of items produced by the recognizer.

Sentence w will be recognized in either two or four steps, depending on whether $w = \epsilon$. The derivation of a goal item for the case in which $w = \epsilon$ is given below.

1. $\langle$ S′ → [(0, 0):·C]$_s$, { [(0, 0):·C]$_s$ }, (0, ϵ) $\rangle$ Axiom

2. $\langle$ S′ → [(0, 0):·C]$_s$, $\emptyset$, (0, ϵ) $\rangle$ Up(1)

Item 2 is a goal item.

For $w \neq \epsilon$, a goal item is derived by the following 4 steps.

1. $\langle$ S′ → [(0, 1):·C]$_s$, $\emptyset$, (0, ϵ) $\rangle$ Axiom

2. $\langle$ S′ → [(0, 1):·C]$_s$, { [(0, 1):·C]$_s$ }, (0, $\not\epsilon$) $\rangle$ Next(1)

3. $\langle$ S′ → [(0, 1):·C]$_s$, $\emptyset$, (0, $\not\epsilon$) $\rangle$ Up(2)

4. $\langle$ S′ → [(0, 1):·C]$_s$, $\emptyset$, (1, ϵ) $\rangle$ Next(3)

Item 4 is a goal item.

To deal with a derivation tree T that consist of more than one node, we need to define the following notions. For any simple expression e = [$w_{x_0+1} \ldots w_{y_0}$:δ_0]$_s$ in T, define the singleton set of subitems $\Delta(e)$ = {[(x_0, y_0):·δ_0]$_s$}. For any complex expression e = [$w_{x_0+1} \ldots w_{y_0}$:δ_0, $w_{x_1+1} \ldots w_{y_1}$:δ_1, …, $w_{x_m+1} \ldots w_{y_m}$:δ_m]$_c$ in T, define the set of subitems $\Delta(e)$ = {[(x_0, y_0):γ_0·δ_0, (x_1, y_1):γ_1·δ_1, …, (x_m, y_m):γ_m·δ_m]$_c$ | $\gamma_i\delta_i$ are the syntactic features of some lexical item $\ell \in$ Lex, $\gamma_i \neq \emptyset$, $0 \leq i \leq m$}.

Let T′ be an extended derivation tree obtained from derivation tree T by adding a node labeled S′, immediately dominating the root of T. Note that the root of T is the complex expression [w:C]$_c$, since T is assumed to have more than one node.

Given a position π in sentence w, define the set of nodes N_π = {a | a is a complex expression in T′; A ∈ $\Delta(a)$ contains position π; if a immediately dominates a complex

expression b in T′, then B $\in \Delta(b)$ does not contain position $\pi\}$. Informally, set N_π consists of those expressions in T′ whose corresponding subitems contain position π and that are furthest removed from the root S′ of the extended derivation tree T′. Let the π-depth d_π of tree T′ be $\text{Max}_{l \in N_\pi}$ d(S′, l), that is, the length of the longest path from the root of T′ to an element of N_π.

For an extended derivation tree T′, input sentence $w = w_1 \ldots w_n$, and a value for pointer π, (0, ϵ) $\leq \pi \leq$ (n, ϵ), we inductively define the sets of items $I^{\downarrow}_{k,\pi}$ and $I^{\uparrow}_{k,\pi}$ for 0 $\leq k \leq d_\pi$:

1. $I^{\downarrow}_{0,\pi}$ = { ⟨ S′ → B_1, { B_1 }, π ⟩ | $B_1 \in \Delta([w{:}\mathtt{c}]_c)$ }.
2. $I^{\downarrow}_{k+1,\pi} = I^{\downarrow}_{k,\pi} \cup$ { ⟨ A → $B_1 \ldots B_n$ C, π ⟩ | there are expressions a, b_i in T′ such that A $\in \Delta(a)$, $B_i \in \Delta(b_i)$, $1 \leq i \leq n$: d(S′, a) = $k+1$; a immediately dominates $b_1 \ldots b_n$; A contains π; $B_i \in$ C iff B_i contains π, $1 \leq i \leq n$ }

and:

1. $I^{\uparrow}_{0,\pi}$ = { ⟨ A → $B_1 \ldots B_n$, ∅, π ⟩ | there are expressions a, b_i in T′ such that A $\in \Delta(a)$, $B_i \in \Delta(b_i)$, $1 \leq i \leq n$: $a \in N_\pi$; a immediately dominates $b_1 \ldots b_n$; A contains π }
2. $I^{\uparrow}_{k+1,\pi} = I^{\uparrow}_{k,\pi} \cup$ { ⟨ A → $B_1 \ldots B_n$, ∅, π ⟩ | there are expressions a, b_i in T′ such that A $\in \Delta(a)$, $B_i \in \Delta(b_i)$, $1 \leq i \leq n$: $\text{Max}_{l \in N_\pi}$ d(a, l) = $k+1$; a immediately dominates $b_1 \ldots b_n$; A contains π } ∪ { ⟨ S′ → B_1, ∅, π ⟩ | $B_1 \in \Delta([w{:}\mathtt{c}]_c)$ }.

Theorem 1 *For all π, (0, ϵ) $\leq \pi \leq$ (n, ϵ), the recognizer will generate all items in the sets $I^{\downarrow}_{d_\pi,\pi}$ and $I^{\uparrow}_{d_\pi,\pi}$.*

Proof. The proof is done by induction on π. (A) For the base case, π = (0, ϵ), we have to show that all the elements of set $I^{\downarrow}_{d_{(0,\epsilon)},(0,\epsilon)}$ are generated, and that all the elements of set $I^{\uparrow}_{d_{(0,\epsilon)},(0,\epsilon)}$ are generated.

First, we will show, by induction on k, that the contents of all sets $\mathrm{I}^{\downarrow}_{k,(0,\epsilon)}$ will be generated, $0 \leq k \leq d_{(0,\epsilon)}$. (1a) In the base case, the elements of $\mathrm{I}^{\downarrow}_{0,(0,\epsilon)}$ = $\langle$ S′ $\rightarrow$ B_1, $\{$ B_1 $\}$, (0, ϵ) $\rangle$ | $\mathrm{B}_1 \in \Delta([w{:}\mathtt{c}]_c)$ $\}$. are provided by the axioms of the deductive system.

(1b) For the induction step, $k > 0$, assume that the contents of the set $\mathrm{I}^{\downarrow}_{k-1,(0,\epsilon)}$ have been generated. We have to show that all elements of the set $\mathrm{I}^{\downarrow}_{k,(0,\epsilon)}$ = $\mathrm{I}^{\downarrow}_{k-1,(0,\epsilon)}$ $\cap$ $\{$ $\langle$ A $\rightarrow$ $\mathrm{B}_1 \ldots \mathrm{B}_n$ C, (0, ϵ) $\rangle$ | there are expressions a, b_i in T such that A $\in \Delta(a)$, $\mathrm{B}_i \in \Delta(b_i)$, $1 \leq i \leq n$: d(S′, a) = k; a immediately dominates $b_1 \ldots b_n$; A contains (0, ϵ); $\mathrm{B}_i \in$ C iff (0, ϵ) contained in B_i, $1 \leq i \leq n$ $\}$ will be generated. Pick an arbitrary item i = $\langle$ A $\rightarrow$ $\mathrm{B}_1 \ldots \mathrm{B}_n$, C, (0, ϵ) $\rangle$ from the set $\mathrm{I}^{\downarrow}_{k,(0,\epsilon)}$ and assume, to obtain the interesting case, i $\notin \mathrm{I}^{\downarrow}_{k-1,(0,\epsilon)}$. According to the definition of set $\mathrm{I}^{\downarrow}_{k,(0,\epsilon)}$, there is an expression a in T such that A $\in \Delta(a)$, for which d(ρ(T), a) = k. Also, subitem A contains (0, ϵ). Since d(S′, a) = $k > 0$, there must be a node x in T′ which immediately dominates expression a. Either x is another expression a', or x is the root node S′. In either case, d(S′, x) = $k - 1$. Assume the immediate daughters of x in T′ are $b'_1 \ldots b'_m$, and suppose that a = b'_j, $1 \leq j \leq m$. In case x = a', let A′, B'_1 be arbitrary subitems such that A′ $\in \Delta(a')$, $\mathrm{B}'_i \in \Delta(b'_i)$, $1 \leq i \leq m$, $i \neq j$, and B'_j = A. Since expression a' immediately dominates expression a in a complete derivation tree and subitem A $\in \Delta(a)$ contains position (0, ϵ), subitem A′ $\in \Delta(a')$ must also contain position (0, ϵ). Then, by the induction hypothesis, the item i' = $\langle$ A′ $\rightarrow$ $\mathrm{B}'_1 \ldots \mathrm{B}'_m$, C′, (0, ϵ) $\rangle$ $\in \mathrm{I}^{\downarrow}_{k-1,(0,\epsilon)}$ will have been generated. Since B'_j = A and A contains position (0, ϵ), B'_j = A $\in$ C′. Since there are no positions to the left of (0, ϵ), position (0, ϵ) will be left-most in B'_j = A. Hence, one of the Predict rules – Unmerge-1, Unmerge-2, Unmerge-3, Unmove-1, or Unmove-2 – will apply to i', producing, among other items, item i. This claim is verified by the proof of completeness for the top-down recognizer that was provided in section 5.6.3 in chapter 5, because subitem A and the sequence of subitems $\mathrm{B}_1, \ldots \mathrm{B}_n$ in a rewrite rule A $\rightarrow$ $\mathrm{B}_1, \ldots \mathrm{B}_n$ are in fact the antecedents and the consequents of the rules of

inference of the top-down recognizer. For the case that x = S′, a similar argument can be constructed, involving item i' = ⟨ S′ → A, { A }, (0, ϵ) ⟩ ∈ $\mathrm{I}^{\downarrow}_{0,(0,\epsilon)}$. Since i was arbitrary and the rules of inference of the top-down recognizer are complete, all items in the set $\mathrm{I}^{\downarrow}_{k,(0,\epsilon)}$ will be generated.

Now we will show, again by induction on k, that the contents of all sets $\mathrm{I}^{\uparrow}_{k,(0,\epsilon)}$ will be generated, $0 \leq k \leq d_{(0,\epsilon)}$. (2a) The base case concerns the generation of the items in the set $\mathrm{I}^{\uparrow}_{0,(0,\epsilon)}$ = { ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, ∅, (0, ϵ) ⟩ | there are expressions a, b_i in T such that A ∈ Δ(a), B_i ∈ Δ(b_i), $1 \leq i \leq n$: $a \in N_{(0,\epsilon)}$; a immediately dominates $b_1 \ldots b_n$; A contains π }. Pick an arbitrary item i = ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, ∅, (0, ϵ) ⟩ from this set. Expression $a \neq$ S′, for S′ $\notin N_{(0,\epsilon)}$. Since $a \in N_{(0,\epsilon)}$, d(S′, a) $\leq d_{(0,\epsilon)}$. Hence, as shown in (1a) and (1b) above, the item i' = ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, C, (0, ϵ) ⟩ ∈ $\mathrm{I}^{\downarrow}_{d_{(0,\epsilon)},(0,\epsilon)}$, has been generated, B_i ∈ C if, and only if, position (0, ϵ) is contained in B_i, $1 \leq i \leq n$. Since expression a is an element of $N_{(0,\epsilon)}$, any expression b_i either is a simple expression, or subitem B_i does not contain position (0, ϵ), $1 \leq i \leq n$. Hence, arbitrary B_j ∈ C, 1 $\leq j \leq n$, is a simple subitem containing (0, ϵ). Since T is a complete derivation tree, expression b_j is lexical. Since B_j is arbitrary, there is a corresponding lexical item for any simple subitem in C. Consequently, the rule of inference Up will apply and the result will be item i. Since i is arbitrary, all elements of $\mathrm{I}^{\uparrow}_{0,(0,\epsilon)}$ will be generated.

(2b) The inductive step involves showing that all elements of set $\mathrm{I}^{\uparrow}_{k,(0,\epsilon)}$ will be generated, $k > 0$, assuming that the contents of the set $\mathrm{I}^{\uparrow}_{k-1,(0,\epsilon)}$ have been generated. Pick an arbitrary item i = ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, ∅, (0, ϵ) ⟩ from the set $\mathrm{I}^{\uparrow}_{k,(0,\epsilon)}$ = $^{\uparrow}_{k-1,(0,\epsilon)}$ ∪ { ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, ∅, (0, ϵ) ⟩ | there are expressions a, b_i in T′ such that A ∈ Δ(a), B_i ∈ Δ(b_i), $1 \leq i \leq n$: $\mathrm{Max}_{l \in N_{(0,\epsilon)}}$ d(a, l) = k; a immediately dominates $b_1 \ldots b_n$; A contains (0, ϵ) } ∪ { ⟨ S′ → B_1, ∅, (0, ϵ) ⟩ | B_1 ∈ Δ([w:C]$_c$) } which is not an element of set $\mathrm{I}^{\uparrow}_{k-1,(0,\epsilon)}$. Assume first A ≠ S′. Since $\mathrm{Max}_{l \in N_{(0,\epsilon)}}$ d(a, l) = $k > 0$, d(S′, a) < $d_{(0,\epsilon)}$. Hence, by (1a) and (1b), there is an item i' = ⟨ A → $\mathrm{B}_1 \ldots \mathrm{B}_n$, C, (0, ϵ) ⟩ ∈ $\mathrm{I}^{\downarrow}_{d_{(0,\epsilon)},(0,\epsilon)}$,

where $B_i \in C$ if, and only if, (0, ϵ) contained in B_i, $1 \leq i \leq n$. Assume, without loss of generality, that n = 2, and that B_1 is a simple subitem in C and B_2 a complex subitem in C. Then expression b_1 is simple. Since it occurs in a complete derivation tree, it must be a lexical expression. For expression b_2, being immediately dominated by expression a, $\text{Max}_{l \in N_{(0,\epsilon)}}$ d(b, l) = $k - 1$. But then, by the induction hypothesis, an item $\langle\ B_1 \rightarrow D_1 \ldots D_m,\ \emptyset,\ (0,\ \epsilon)\ \rangle \in I^{\uparrow}_{k-1,(0,\epsilon)}$ has been generated. Thus all conditions for the rule Up to apply to item i' are met. The result will be item i. For A = S′, the argument unfolds analogously. Since i is arbitrary in set $I^{\uparrow}_{k,(0,\epsilon)}$, all items of this set will be generated. This concludes the base step of the proof of theorem 1.

(B) For the induction step, assume that the elements of sets $I^{\downarrow}_{d_p,p}$ and $I^{\uparrow}_{d_p,p}$ have been generated, (0, ϵ) $\leq p \leq prev(k) \leq$ (n, ϵ). We have to show that the elements of sets $I^{\downarrow}_{d_k,k}$ and $I^{\uparrow}_{d_k,k}$ will be generated.

We will first show, by induction on j, that the contents of all sets $I^{\downarrow}_{j,k}$ will be generated, $0 \leq j \leq d_k$. (3a) To satisfy the base case, the recognizer has to generate the elements of set $I^{\downarrow}_{0,k}$ = $\{\ \langle\ S' \rightarrow B_1,\ \{\ B_1\ \},\ k\ \rangle \mid B_1 \in \Delta([w{:}c]_c)\ \}$. Let i = $\langle\ S' \rightarrow B_1$, $\{\ B_1\ \}$, $k\ \rangle$ be an arbitrary element drawn from this set. From the induction hypothesis we know the item i' = $\langle\ S' \rightarrow B_1,\ \emptyset,\ prev(k)\ \rangle \in I^{\uparrow}_{d_{prev(k)},prev(k)}$ has been generated. Since $prev(k) \leq$ (n, ϵ), the rule Next will apply to item i' to produce item i. Since i was picked arbitrarily from set $I^{\downarrow}_{0,k}$, all elements of this set will be generated.

(3b) For the induction step, assume that the elements of set $I^{\downarrow}_{j-1,k}$ have been generated for arbitrary j, $1 \leq j \leq d_k$. To show: all elements of set $I^{\downarrow}_{j,k}$ = $I^{\downarrow}_{j-1,k} \cup \{\ \langle\ A \rightarrow B_1 \ldots B_n\ C,\ k\ \rangle$ | there are expressions a, b_i in T such that A $\in \Delta(a)$, $B_i \in \Delta(b_i)$, $1 \leq i \leq n$: d(S′, a) = j; a immediately dominates $b_1 \ldots b_n$; A contains k; $B_i \in C$ iff k contained in B_i, $1 \leq i \leq n$ $\}$ will be generated. Let item i = $\langle$ A $\rightarrow B_1 \ldots B_n$ C, $k\ \rangle$ be an arbitrary item in this set such that $i \notin I^{\downarrow}_{j-1,k}$. Since d(S′, a) = $j > 0$, there must be a node x in T′ which immediately dominates expression a. Either x is another

expression a', or x is the root node S′. In either case, d(A′, x) = $j - 1$. Assume the immediate daughters of x in T′ are $b'_1 \ldots b'_m$, and suppose that $a = b'_l$, $1 \leq l \leq m$. In case $x = a'$, let A′, B'_i be arbitrary subitems such that A′ $\in \Delta(a')$, $B'_i \in \Delta(b'_i)$, $1 \leq i \leq m$, $i \neq j$, and B'_l = A. Since expression a' immediately dominates expression a in a complete derivation tree and subitem A $\in \Delta(a)$ contains position k, subitem A′ $\Delta(a')$ must also contain position k. Then, by the induction hypothesis, the item i' = $\langle$ A′ $\rightarrow$ $B'_1 \ldots B'_m$, C′, k, $\rangle \in I^{\downarrow}_{j-1,k}$ will have been generated. Since B'_l = A and A contains position k, B'_l = A $\in$ C′. If position k is left-most in B'_l, then one of the Predict rules will apply. Among the results will be item i, as was shown in (1b). If k is not the left-most position in B'_l, then let (q, ·) be the right-most position to the left of k contained in the position vectors of B'_l. Obviously, (q, ·) $< k$. Hence, by the induction hypothesis, the item i'' = $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, $\emptyset$, (q, ·) $\rangle \in I^{\uparrow}_{d_{(q,\cdot)},(q,\cdot)}$ has been generated. Rule Down will apply to items i' and i'' and produce item i. For the case that expression a is immediately dominated by node S′, a similar argument can be constructed, involving items i = $\langle$ S′ $\rightarrow$ A, { A }, k $\rangle \in I^{\downarrow}_{0,k}$ and i'' = $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, $\emptyset$, (q, ·) $\rangle \in I^{\uparrow}_{d_{(q,\cdot)},(q,\cdot)}$.[19] Item i was picked at random; hence all items of set $I^{\downarrow}_{j,k}$ will be generated.

Using induction on j, we will now show that all elements of the sets $I^{\uparrow}_{j,k}$ will be generated, $0 \leq j \leq d_k$. (4a) In he base case, the elements to be generated are the elements of set $I^{\uparrow}_{0,k}$ = { $\langle$ A $\rightarrow$ $B_1 \ldots B_n$, $\emptyset$, k $\rangle$ | there are expressions a, b_i in T′ such that A $\in \Delta(a)$, $B_i \in \Delta(b_i)$, $1 \leq i \leq n$: $a \in N_k$; a immediately dominates $b_1 \ldots b_n$; A contains k }. This case is analogous to (2a).

(4b) For the induction step, we have to show that all items of set $I^{\uparrow}_{j,k}$ will be generated, given the elements of set $I^{\uparrow}_{j-1,k}$, for arbitrary j, $0 \leq j \leq d_k$. The argument is analogous to the argument in (2b). This concludes the inductive step of the proof of theorem 1 and so completes the proof of this theorem. □

[19] If expression a is immediately dominated by S′, then $a = [w{:}\mathtt{C}]_c$. Since (0, ϵ) $< k$, k will never be left-most in any subitem A $\in \Delta(a)$.

Completeness now follows as a corollary from theorem 1, because, by definition of $I^{\uparrow}_{d_{(n,\epsilon)},(n,\epsilon)}$, if $w \in$ L(G), $|w| = n$, then a goal item $\langle$ S′ $\rightarrow$ [(0, n):$\gamma \cdot$c], $\emptyset$, (n, ϵ) $\rangle$ is an element of set $I^{\uparrow}_{d_{(n,\epsilon)},(n,\epsilon)}$.

6.5.3 Complexity

The subitems of the Earley-style recognizer are in fact the items of the bottom-up recognizer presented in chapter 4 of this dissertation. Hence, according to the argument in section 4.7, for a given Minimalist Grammar G = (V, Lex, Cat, $\mathcal{F}$) and input string of length n, the number of different left-hand sides that can appear in a rewrite rule A $\rightarrow$ $B_1 \ldots B_m$ of an item $\langle$ A $\rightarrow$ $B_1 \ldots B_m$, C, π $\rangle$ is bounded by $O(n^{2k+2})$, where k = $|$*licensees*$|$. For any rewrite rule, the sequence $B_1 \ldots B_m$ contains only one element whose value depends on n, namely the value of v when the position vector (p, q) of subitem A is split by the rules of inference Unmerge-1, Unmerge-3, and Unmove-1, cf. section 6.3.5. Therefore, the number of different rewrite rules falls within the bound $O(n^{2k+3})$. Because of the inclusion of the pointer π in an item, whose value depends on n, the number of possible items is $O(n^{2k+4})$. Since for an item $\langle$ A $\rightarrow$ $B_1 \ldots B_m$, C, π $\rangle$, the contents of C are a subset of $B_1 \ldots B_m$, set C does not add to the order of the bound on the possible number of items.

Regarding the time complexity of the recognizer, the costliest step is applying the rule of inference Down, since this involves searching through the entire chart. This will amount to a time cost of $O(n^{2k+4})$ per item on the agenda. Following the argument in section 4.7, since the number of items that will be put on the agenda while recognizing the input string is $O(n^{2k+4})$, the overall time complexity of the recognizer is $O(n^{4k+8})$.

6.6 Conclusions

In this chapter we have defined an Earley-style recognizer for languages generated by Minimalist Grammars which is applicable to all Minimalist Grammars, including the class of recursive Minimalist Grammars for which the top-down recognizer defined in the previous chapter will fail to halt. The soundness and completeness proofs provided in this chapter not only prove the correctness of the Earley-style recognizer, but also expand our understanding of the formalism of Minimalist Grammars and bring us closer to the definition of an automaton that will simulate top-down derivations of a Minimalist Grammar. The number of items generated by the recognizer can be brought down by adding look-ahead to the Predict rules of inference, following the scheme of section 5.4. Complexity can be further improved if the conceptual simplifications of Earley's original algorithm for Context-Free Grammars proposed by [GHR80] can be incorporated in the recognizer presented in this chapter.

CHAPTER 7

Conclusions

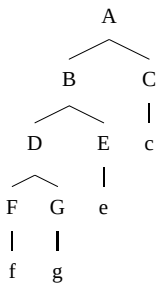
The ultimate goal of my dissertation research is to further the understanding of the syntactic aspects of the operation of the human sentence processor, by means of specifying formal parsing models for natural language. In the psycholinguistic literature many properties are ascribed to the human sentence processor. However, since psycholinguistic proposals about human sentence processing are often presented in informal terms, it is not obvious that parsers with these properties are reasonable from a computational point of view. Having read the preceding chapters, we have arrived at an understanding of some formal parsing models for Minimalist Grammars, a formalism which lends itself very well to expressing current proposals about the kind of structures found in natural language. We thus find ourselves in an excellent position to rigorously explore the computational consequences of psycholinguistic theories of human sentence processing. In this section, we will briefly indicate some directions for these explorations, as well as touch upon some issues of a more formal nature that remain for future research.

7.1 Bottom-Up vs. Top-Down

Pure bottom-up and top-down parsers are generally rejected as adequate models for human sentence processing because they require unbounded memory for right-branching and left-branching structures respectively, while the human sentence processor, which

is equipped with finite memory, seems to have no particular problems with structures of this kind (e.g. [Cro99]). For example, while recognizing the left-branching structure in 1 using a grammar that generates this tree, the top-down recognizer informally introduced in section 6.1 of chapter 6 will produce the sequence FGEC. It is easy to see that if there is no bound on the depth of left-branching trees – and there is no principled bound – then there is no bound on the length of the sequences produced by the top-down recognizer.

1.



A bottom-up recognizer for Context-Free Languages, like a top-down one, also manipulates sequences of terminal and non-terminal symbols, but these sequences are not predictions about the structure of a sentence, but rather representations of partial structures discovered so far. A basic bottom-up recognizer has two operations: shift, which appends a word from the sentence to the right of the sequence, and reduce, which replaces the terminal and non-terminal symbols matching the right-hand side of some grammar rule by the symbol of its left-hand side. The sequence of terminal and non-terminal symbols that is replaced must be a postfix of the entire sequence. The recognizer starts with the empty sequence, and a sentence has been recognized if all the words of the sentence have been shifted and the sequence has been reduced to the start symbol of the grammar (see for example [ASU86] and [SS88] for formal descriptions of bottom-up recognizers for Context-Free Languages). For example, for the Context-Free Grammar with start symbol A and the rules given below, recognition of the sentence *cegf* is represented by the sequences ϵ, c, C, Ce, CE, CEg, CEG, CEGf,

CEGF, CED, CB, A.

2. A → CB

3. B → ED

4. D → GF

5. C → c

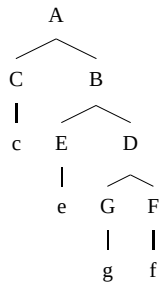
6. E → e

7. F → f

8. G → g

The right-branching tree of the sentence *cegf* is given in 9.

9.



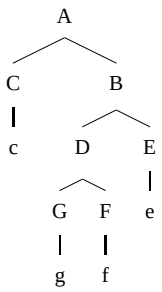
Since the sequence CEGF is among the sequences produced for the sentence *cegf* and there is no general bound on the depth of right-branching trees, it follows from the example above that there is no principled bound on the length of the sequences produced by a bottom-up recognizer.

Bottom-up recognizers have the further problem that they produce unconnected structures, as each of the symbols in a sequence in fact stands for a tree rooted in that symbol. This seems to go against the incremental nature of the human sentence

processor, which is assumed to maintain one or more structures, each of which includes all the words of the sentence heard so far (e.g. [Sta94], [PCC00]). Top-down recognizers are incremental in this sense, because the symbols of a sequence posited by a top-down recognizer are in fact the leaves of one, connected, but incomplete tree rooted in the start symbol. However, a disadvantage of the top-down strategy is that the recognizer starts predicting structures without first listening to the actual words of the sentence. This property of top-down recognizers leads to problems such as non-terminating computations for left-recursive grammars as explained in section 6.1 in chapter 6.

Because of these various drawbacks of the bottom-up and top-down mechanisms, the human sentence processor is assumed to employ a mixed strategy, using top-down predictions and bottom-up confirmations, as in a left-corner parser, for example (e.g. [Cro99]). A left-corner parser does not require unbounded memory for right-branching and left-branching structures, but it does for center-embedded structures, correctly predicting that these structure are problematic for the human sentence processor. A center-embedded structure essentially is a structure that is alternately right-branching and left-branching, like the structure in 10, for example.

10.



The differences between the top-down, bottom-up and left-corner strategies are succinctly illustrated by the following description. Given a context-free rule A $\rightarrow$ BC, a top-down recognizer, having predicted a constituent of category A, will predict that this constituent consists of a constituent of category B followed by a constituent of

category C. A bottom-up recognizer, having found a constituent of category B followed by a constituent of category C, will conclude that it has found a constituent of category A. A left-corner recognizer, having found a constituent of category B, will predict a constituent of category C and conclude that it has found a constituent of category A once the predicted constituent of category C has been found. Category B is the left-corner of the rule A $\rightarrow$ BC.

In the sequences produced by a left-corner parser, we will write A for a constituent of category that has been found, and -A for a constituent of category A that has been predicted. The parser has a shift rule identical to the shift rule of the bottom-up parser. A sequence containing B is rewritten by replacing B with A-C, provided the grammar contains a rule A $\rightarrow$ BC. As in the bottom-up parser, only postfixes of a sequence can be rewritten, the initial sequence is empty, and a sentence has been recognized successfully when all words of the sentence have been shifted and the sequence has been reduced to the start symbol of the grammar. When a rewrite operation results in sequence in which a predicted category -A is followed by a discovered category A, these symbols can cancel one another.

For example, assume that the tree in 10 is licensed by a Context-Free Grammar with the following rules:

11. A $\rightarrow$ CB
12. B $\rightarrow$ DE
13. D $\rightarrow$ GF
14. C $\rightarrow$ c
15. E $\rightarrow$ e
16. F $\rightarrow$ f

17. G $\rightarrow$ g

Then the sentence *cgfe* is parsed as follows: ϵ, c, C, A-B, A-Bg, A-BG, A-BD-F, A-BD-Ff, A-BD, A-E, A-Ee, A. This succession contains the sequence A-BD-F, which corresponds to the spine of the tree in 10. Since there is no principled bound on the depth of center-bedded trees, it follows that there is no bound on the length of the sequences produced by a left-corner recognizer for a center-embedded structure.[1, 2] (More thorough descriptions of left-corner parsing can be found in [RL70] and [Ned83], for example; [Sta94] contains a psycholinguistically oriented discussion of left-corner parsing.)

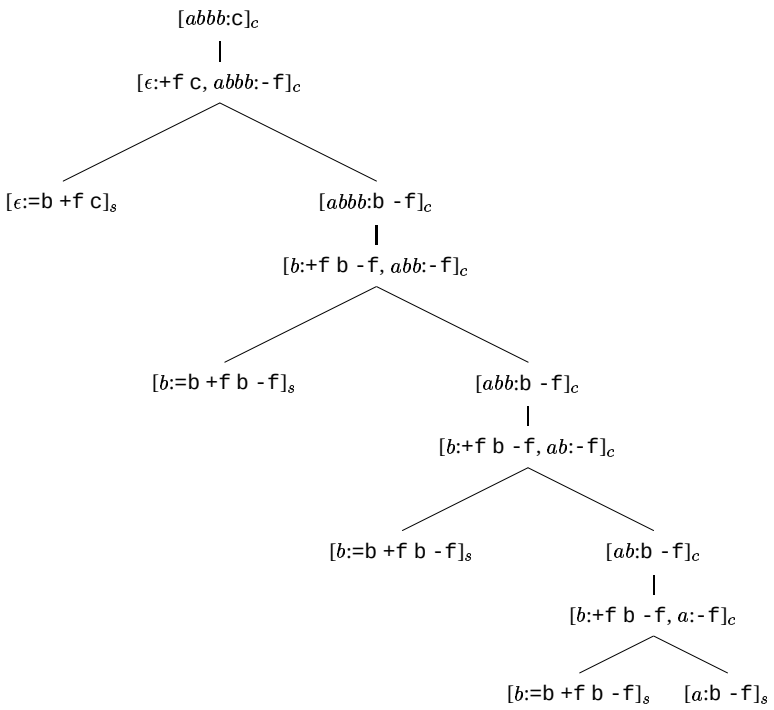
The positive and negative properties of the parsing strategies outlined above for Context-Free Grammars carry over to recognizers for Minimalist Grammars on the understanding that the notions left-branching, right-branching, center-embedding and left-corner apply to derivation trees rather than to the surface trees generated by Minimalist Grammars.[3] Since the leaves of minimalist derivation trees do not necessarily correspond to the order of the words in a sentence, the geometry of a derivation tree *per se* does not always coincide with the geometry of a derivation tree for the purposes of recognition. For example, the derivation tree given in 18 below, repeated from in section 6.1 in chapter 6, is a left-branching tree for the purposes of left-to-right recognition, even though geometrically it is a right-branching tree.

[1] Note that the argument for the unboundedness for top-down parsers and bottom-up parsers involves the non-branching nodes of the trees in 1 and 9, whereas the argument for left-corner parsers involves the branching nodes in the tree in 10.

[2] Note also that the left-corner parser does not build connected structures.

[3] The bottom-up, top-down, and left-corner recognizers for Context-Free Languages discussed above build only one derivation at a time for a sentence, whereas the recognizers for Minimalist Languages introduced in the previous chapters use a chart which allows them to pursue all derivations of a sentence in parallel, but for the purposes of the current discussion this difference is just a technicality, because there is a very close correspondence between the rules of inference of the recognizers and the structure building functions of Minimalist Grammars.

18.



For the recognizer, the tree in 18 is left-branching because the lexical items for the left-most words of the sentence *abbbb* are deepest in the tree. For the sisters [b:=b +f b -f]$_s$ and [ab:b -f]$_c$ in the tree in 18, for example, the recognizer considers expression [ab:b -f]$_c$ to be to the left of expression [b:=b +f b -f]$_s$ because the string ab of the former expression covers the non-empty words at positions 0 and 1 in the sentence $abbb$ and string b of the latter expression covers the non-empty word at position 2 in this sentence.[4]

Working through a few examples, it appears that the geometry of a complete tree generated by some Minimalist Grammar coincides with the geometry of its derivation tree for the purposes of recognition. At this point it is unclear to me whether, if true, this is a surprising fact or not. Perhaps Kracht's method for connecting derivations

[4] For the particular derivation tree in 18, it is possible to reorder sister nodes in such a way that the order of the lexical items at the leaves corresponds to the order of the words in the sentence *abbb*. However, it is not generally possible to rearrange a derivation tree in this way, cf. the derivation tree given in 16 in section 5.2 of chapter 5.

trees and complete surface trees discussed in section 2.5 in chapter 2 will provide some deeper understanding of this matter.

7.2 Parallel vs. Serial

The recognizers for Minimalist Grammars described in the preceding chapters use charts to store intermediate results. When for some grammar G and a well-formed sentence w the recognizer has computed the closure of the axioms under the rules of inference, the chart will contain representations of all possible derivations of sentence w according to grammar G. In fact, the chart is a data structure whose finite contents potentially encode an infinite number of derivation trees. This seems too powerful a feature from a psycholinguistical point of view, because it is generally believed that the human sentence processor cannot handle an infinite number of trees. Moreover, a dedicated tree collecting procedure is needed to retrieve derivation trees from the chart. This arrangement is also less appealing to psycholinguists, because the incremental nature of the human sentence processor suggests that partial analyses of a sentence are available immediately for further processing (e.g. [PCC00]). The chart-based recognizers can be made to deliver trees immediately by interleaving the tree collecting procedure with the computation of the chart, effectively replacing the items in the chart with the fragments of derivation trees they represent under the invariant.[5] This will turn the recognizers into parsers. Maintaining actual trees rather than items representing a possibly infinite number of trees, the parsers will need additional mechanisms for determining how many, and, if not all, which analyses for a sentence to pursue in parallel.

Full parallelism – keeping track of all possible analyses – is psycholinguistically implausible, because due to the highly ambiguous nature of natural language, most

[5] Doing this for the Earley-style recognizer would be a very interesting exercise!

sentences have very many derivations, and some are temporarily infinitely ambiguous. Moreover, for certain grammatical sentences, the human sentence processor has noticeable difficulties arriving at an analysis, or it may even fail to deliver an analysis at all. Such garden path effects are unexpected if all analyses of a sentence were equally available.

While full parallelism is not practical, pursuing a bounded number of analyses in parallel is a possible strategy. Such a strategy has been proposed in [Gib91] and [Gib98], for example. In this approach, the degree of parallelism of the human sentence processor is determined by the complexity of possible analyses for the sentence being processed. Analyses whose complexity exceeds the complexity of the simplest analysis for the sentence so far by a certain threshold will be removed from consideration. According to Gibson's syntactic prediction locality theory, one of the factors contributing to the complexity of a sentence is memory cost. While parsing the sentence from left to right, the human sentence processor will predict what syntactic categories are needed to complete the sentence in a grammatical way. Keeping these predictions in memory is costly. Furthermore, the longer a prediction has to be kept in memory, the more expensive it is. Gibson empirically supports his theory by applying it to a wide range of sentences, but he does not provide a description of a parser that is capable of making the syntactic predictions that determine the complexity of a sentence. It will be very interesting to see to what extent the findings in [Gib91] and [Gib98] can be replicated by the top-down and Earley recognizers presented in the preceding chapters.

Frazier (e.g. [Fra78], [FC96]) has proposed that the human sentence processor pursues only one analysis at a time. When the processor is confronted with a choice concerning how to incorporate the next word of the sentence into the grammatical structure built so far, it will use the following principle to make a decision:

a. Minimal Attachment: attach incoming material into the phrase marker being constructed using the fewest nodes consistent with the well-formedness rules of the language.

If the principle of Minimal Attachment does not resolve the ambiguity, then the principle of Late Closure is invoked to break the tie:

b. Late Closure: when possible, attach incoming material into the clause or phrase currently being parsed.

Obviously, empirical verification of these principles depends greatly on assumptions about the proper syntactic structure of sentences and on assumptions regarding the expanse of the structures that have already been built when new material is to be attached.

The formalism of Minimalist Grammars will allow us to test the consequences of the two principles mentioned above using current proposals about the syntactic structure of natural language. Furthermore, the intermediate trees produced by the parser will tell us exactly how much structure exists at each point in the parsing process.

7.3 Probabilistic Minimalist Grammars

One way to make the recognizers presented in the previous chapters computationally tractable is to use Probabilistic Minimalist Grammars. With a probabilistic grammar, a recognizer can keep track of the probabilities of the various partial analyses built at each point in the sentence and continue to work on the most promising ones and discard the rest – see for example [Sto95] for an Earley parser for Context-Free Languages computing probabilities of partial analyses and [Hal01] for an interesting psycholinguistic interpretation of these probabilities.

In a Probabilistic Context-Free Grammar (see for example [MS99]), each rewrite rule of the grammar is associated with a probability. By the context-freeness of the rewrite rules, each use of a rule in a derivation is an independent event. Hence, the probability of a particular derivation of a sentence is the product of the probabilities of the rules used in the derivation. Schabes ([Sch92]) and Resnik ([Res92]) present probabilistic versions of Tree Adjoining Grammar. For these grammars, the independent events in a derivation are the adjunction operations.

A Probabilistic Minimalist Grammar can be obtained by assigning probabilities to instantiations of the structure building functions Merge-1, Merge-2, Merge-3, Move-1, and Move-2 given in section 4.1 in chapter 4. Since the expressions of a Minimalist Grammar are composed of various parts, there is some freedom in deciding what parts of an expression the probability of an instantiation should be dependent on, or, to put the same issue differently, what the independent events are that are denoted by the instantiations of the structure building functions. This is an empirical issue. For example, as reported in [MS99] (p. 418), the verb *take* is more likely to merge with an object determiner phrase than the verb *want* in the Penn Tree Bank, suggesting that the probabilities should be predicated on (parts of) the strings associated with the expressions involved in the instantiations of the structure building functions. However, it is not clear to what extent interesting statistical generalizations about natural language are captured by making the probabilities dependent on other information contained in an expression, e.g. the identity of the chains in an expression other than the chain representing the head of the expression.

7.4 Matrix Multiplication

Valiant ([Val75]) has shown that recognition of Context-Free Languages can be carried out at least as fast matrix multiplication. There are algorithms for matrix multiplication

that run in sub-cubic time, which is better than the cubic time complexity of the CKY and Earley methods for recognizing Context-Free Languages.

Nakanishi et al. ([NTS00]) follow a strategy inspired by Valiant's to improve upon the time complexity of the recognition problem for Multiple-Context Free Grammars. Tree Adjoining Grammars are another grammar formalism more powerful than Context-Free Grammars for which a connection with matrix multiplication has been made, e.g. in [Sat94] and [RY95]. Satta's approach provides a way to construct a solution for the matrix multiplication problem given a solution for the recognition problem for Tree Adjoining Languages. Taking into account the attention that has been devoted to efficient solutions to the matrix multiplication problem, the complexity of the best solution to this problem can be regarded as a lower bound for the best solution to the recognition problem for Tree Adjoining Languages: it will be very hard to improve the run-time complexity of a recognizer for Tree Adjoining Languages if the complexity of its corresponding solution to the matrix multiplication problem is close to the best solution.

In order to assess how easy it will be to improve upon the run-time complexity of the recognizers for Minimalist Languages presented in this dissertation, it would be very interesting to reduce the recognition problem for Minimalist Languages to the problem of matrix multiplication.

REFERENCES

[AET96] W. Abraham, S.D. Epstein, H. Thráinsson, and C.J.-W. Zwart. *Minimalist Ideas: Syntactic Studies in the Minimalist Framework*. John Benjamins Publishing Company, 1996.

[AGV00] M.A. Alonso, J. Graña, M. Vilares, and E. de la Clergerie. "New Tabular Algorithms for LIG Parsing." In *Proceedings of the 6th International Workshop on Parsing Technologies, IWPT 2000*, 2000. Trento, Italy.

[Aho68] A.V. Aho. "Indexed Grammars." *Journal of the Association for Computing Machinery*, **15**, 1968.

[Alb00] D.M. Albro. "Taking Primitive Optimality Theory Beyond The Finite State." In J. Eisner, L. Karttunen, and A. Thériault, editors, *Finite State Phonology: Proceedings of Fifth Workshop of the ACL Special Interest Group in Computational Phonology*, 2000.

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[AU72] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. 1*. Prentice Hall, 1972.

[BAT91] R.C. Berwick, S.P. Abney, and C. Tenny. *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer Academic Publishers, 1991.

[Ber91] R.C. Berwick. "Principle-Based Parsing." In P. Sells, S.M. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*. Kluwer Academic Publishers, 1991.

[Bev70] T.G. Bever. "The Cognitive Basis of Linguistic Structure." In J. Morton, editor, *Cognitive Development of Language*. John Wiley, New York, 1970.

[BF95] R.C. Berwick and S. Fong. "A Quarter Century of Computation with Transformational Grammar." In J. Cole, G.M. Green, and J.L. Morgan, editors, *Linguistics and Computation*. Kluwer Academic Publishers, 1995.

[BKP82] J.W. Bresnan, R.M. Kaplan, P.S. Peters, and A. Zaenen. "Cross-Serial Dependencies in Dutch." *Linguistic Inquiry*, **13**, 1982.

[Bou96] P. Boullier. "Another Facet of LIG Parsing." In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[Bou98] P. Boullier. “Proposal for a Natural Language Processing Syntactic Backbone.” Research Report N° 3342, INRIA-Rocquencourt, France, 1998.

[Cho65] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts, 1965.

[Cho81] N. Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, the Netherlands, 1981.

[Cho86] N. Chomsky. *Knowledge of Language: Its Nature, Origin, and Use*. Praeger, 1986.

[Cho95] N. Chomsky. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts, 1995.

[Cho00] N. Chomsky. “Minimalist Inquiries: the Framework.” In R. Martin, D. Michaels, and J. Uriagereka, editors, *Step by Step: Essays on Minimalist Syntax in honor of Howard Lasnik*. MIT Press, Cambridge, Massachusetts, 2000.

[Cho01] N. Chomsky. “Derivation by Phase.” In M. Kenstowicz, editor, *Ken Hale: A Life in Linguistics*. MIT Press, Cambridge, Massachusetts, 2001.

[Cor99] T.L. Cornell. “Derivational and Representational Views of Minimalist Transformational Grammars.” In A. Lecomte, F. Lamache, and G. Perrier, editors, *Logical Aspects of Computational Linguistics (LACL ’97)*, volume 1582 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 1999.

[Cro99] M.W. Crocker. “Mechanisms for Sentence Processing.” In S. Garrod and M. Pickering, editors, *Language Processing*. Taylor and Francis, 1999.

[Ear68] J.C. Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1968.

[Ear70] J.C. Earley. “An Efficient Context-Free Parsing Algorithm.” *Communications of the Association for Computing Machinery*, **13**(2), 1970.

[EH99] S.D. Epstein and N. Hornstein. *Working Minimalism*. MIT Press, Cambridge, Massachusetts, 1999.

[FC96] L. Frazier and C. Clifton, Jr. *Construal*. MIT Press, Cambridge, Massachusetts, 1996.

[Fra78] L. Frazier. *On Comprehending Sentences: Syntactic Parsing Strategies*. PhD thesis, University of Connecticut, Storrs, Connecticut, 1978.

[Gaz88] G. Gazdar. “Applicability of Indexed Grammars to Natural Languages.” In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*. Reidel Publishing Company, 1988.

[GHR80] S.L. Graham, M.A. Harrison, and W.L. Ruzzo. “An Improved Context-Free Recognizer.” *ACM Transactions on Programming Languages and Systems*, **2**(3), 1980.

[Gib91] E. Gibson. *A Computational Theory of Human Linguistic Processing: Memory Limitations and Processing Breakdown*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991.

[Gib98] E. Gibson. “Linguistic Complexity: Locality of Syntactic Dependencies.” *Cognition*, **1**(68), 1998.

[GO96] E. Groat and J. O’Neil. “Spell-Out at the LF Interface.” In W. Abraham, S.D. Epstein, H. Thráinsson, and C.J.-W. Zwart, editors, *Minimalist Ideas: Syntactic Studies in the Minimalist Framework*. John Benjamins Publishing Company, 1996.

[GPS85] G. Gazdar, E. Klein G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, 1985.

[Gri86] R. Grishman. *Computational Linguistics: an Introduction*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, United Kingdom, 1986.

[Hal01] J. Hale. “A Probabilistic Earley Parser as a Psycholinguistic Model.” In *Proceedings of the 2nd Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-2001*, 2001. Carnegie Mellon University, Pittsburgh, Pennsylvania.

[Har00] H. Harkema. “A Recognizer for Minimalist Grammars.” In *Proceedings of the 6th International Workshop on Parsing Technologies, IWPT 2000*, 2000. Trento, Italy.

[Har01] H. Harkema. “A Characterization of Minimalist Grammars.” In P. de Groote, G.F. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics (LACL 2001)*, volume 2099 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 2001.

[JLT75] A.K. Joshi, L.S. Levy, and M. Takahashi. “Tree Adjunct Grammars.” *Journal of Computer and System Sciences*, **10**(1), 1975.

[JM00] D. Jurafski and J.H. Martin. *Speech and Language Processing: an Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[Jos85] A.K. Joshi. "Tree Adjoining Grammars: How Much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions?" In D.R. Dowty, L. Karttunen, and A.M. Zwicky, editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*. Cambridge University Press, 1985.

[Jos87] A.K. Joshi. "An Introduction to Tree Adjoining Grammars." In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, 1987.

[JS97] A.K. Joshi and Y. Schabes. "Tree-Adjoining Grammars." In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 3: Beyond Words*. Springer Verlag, Berlin, Heidelberg, Germany, 1997.

[JVW91] A.K. Joshi, K. Vijay-Shanker, and D. Weir. "The Convergence of Mildly Context-Sensitive Grammar Formalisms." In P. Sells, S.M. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*. Kluwer Academic Publishers, 1991.

[Kay94] R. Kayne. *The Antisymmetry of Syntax*. MIT Press, Cambridge, Massachusetts, 1994.

[Kay98] R. Kayne. "Overt vs. Covert Movement." *Syntax*, **1**(2), 1998.

[Kin83] M. King. "Transformational Parsing." In M. King, editor, *Parsing Natural Language*. Academic Press, 1983.

[KK01] G. Kobele and J. Kandybowicz. "A New Normal Form Theorem for Minimalist Grammars." Unpublished manuscript. Department of Linguistics, UCLA, Los Angeles, California, 2001.

[Kra01] M. Kracht. "Syntax in Chains." *Linguistics and Philosophy*, **24**, 2001.

[KS96] E.L. Keenan and E.P. Stabler. "Abstract Syntax." In A.M. DiSciullo, editor, *Configurations: Essays on Structure and Interpretation*. Cascadilla Press, 1996.

[KS00] H. Koopman and A. Szabolcsi. *Verbal Complexes*. MIT Press, Cambridge, Massachusetts, 2000.

[LR99] A. Lecomte and C. Retoré. "Towards a Minimal Logic for Minimalist Grammars." In *Proceedings of Formal Grammar 1999*, 1999. Utrecht, the Netherlands.

[Mah00] A. Mahajan. "Word Order and Remnant Movement: Eliminating Head Movement." In *23^{rd} Generative Linguistics in the Old World Colloquium, GLOW 2000*, 2000.

[Mar80] M.P. Marcus. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, Massachusetts, 1980.

[Mer95] P. Merlo. "Modularity and Information Content Classes in Principle-Based Parsing." *Computational Linguistics*, **21**(4), 1995.

[Mic98] J. Michaelis. "Derivational Minimalism is Mildly Context-Sensitive." In M. Moortgat, editor, *Logical Aspects of Computational Linguistics, (LACL '98)*, volume 2014 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 1998.

[Mic01a] J. Michaelis. *On Formal Properties of Minimalist Grammars*. PhD thesis, Potsdam University, Germany, 2001.

[Mic01b] J. Michaelis. "Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars." In P. de Groote, G.F. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics (LACL 2001)*, volume 2099 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 2001.

[Mor00] F. Morawietz. "Chart Parsing and Constraint Programming." In *Proceedings of COLING-2000*, 2000. Saarbrücken, Germany.

[MS97] A. Mateescu and A. Salomaa. "Aspects of Classical Language Theory." In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer Verlag, Berlin, Heidelberg, Germany, 1997.

[MS99] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999.

[Ned83] M.-J. Nederhof. "Generalized Left-Corner Parsing." In *Proceedings of the 6th Meeting of the European Association for Computational Linguistics*, 1983. Utrecht, the Netherlands.

[Ned99] M.-J. Nederhof. "The Complexity of the Correct-Prefix Property for TAGs." *Computational Linguistics*, **25**(3), 1999.

[Noz86] R. Nozohoor-Farshi. *LRRL(k) Grammars: A left to Right Parsing Technique with Reduced Lookaheads*. PhD thesis, University of Alberta, Canada, 1986.

[NTS00] R. Nakanishi, K. Takada, and H. Seki. "An Efficient Recognition Algorithm for Multiple Context-Free Languages." *Proceedings of the 5th Meeting on the Mathematics of Language,* 2000. Saarbrücken, Germany.

[PCC00] M.J. Pickering, C. Clifton, Jr., and M.W. Crocker. "Architectures and Mechanisms in Sentence Comprehension." In M.W. Crocker, M.J. Pickering, and C. Clifton, Jr., editors, *Architectures and Mechanisms for Language Processing*. Cambridge University Press, 2000.

[Pol84] C.J. Pollard. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford University, Stanford, California, 1984.

[PR73] P.S. Peters and R.W. Ritchie. "On the Generative Power of Transformational Grammars." *Information Sciences*, **10**, 1973.

[PW83] F.C.N. Pereira and D.H.D. Warren. "Parsing as Deduction." In *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, 1983.

[Res92] P. Resnik. "Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing." In *Proceedings of COLING 14*, 1992. Nantes, France.

[RL70] D.J. Rosenkrantz and P. M. Lewis. "Deterministic Left-Corner Parsing." In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, 1970.

[Rog98] J. Rogers. *A Descriptive Approach to Language-Theoretic Complexity*. CSLI Publications, Stanford, California, 1998.

[Ros67] J.R. Ross. *Constraints on Variables in Syntax*. PhD thesis, MIT, Cambridge, Massachusetts, 1967.

[RY95] S. Rajasekaran and S. Yooseph. "TAL Recognition in $O(M(n^2))$ Time." In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995. Cambridge, Massachusetts.

[Sam83] G.R. Sampson. "Deterministic Parsing." In M. King, editor, *Parsing Natural Language*. Academic Press, 1983.

[Sat94] G. Satta. "Tree-Adjoining Grammar Parsing and Matrix Multiplication." *Computational Linguistics*, **20**(2), 1994.

[Sch92] Y. Schabes. "Stochastic Lexicalized Tree-Adjoining Grammars." In *Proceedings of COLING 14*, 1992. Nantes, France.

[SHF91] H. Seki, T. Matsumura H., M. Fujii, and T. Kasami. "On Multiple Context-Free Grammars." *Theoretical Computer Science*, **88**, 1991.

[Shi85] S.M. Shieber. "Evidence against the Context-Freeness of Natural Language." *Linguistics and Philosophy*, **8**, 1985.

[Sik97] K. Sikkel. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Springer Verlag, Berlin, Heidelberg, Germany, 1997.

[SK00] E.P. Stabler and E.L. Keenan. "Structural Similarity." In A. Nijholt, G. Scollo, and D. Heylen, editors, *Algebraic Methods in Language Processing, AMiLP 2000*. University of Iowa, 2000.

[SN97] K. Sikkel and A. Nijholt. "Parsing of Context-Free Languages." In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 2: Linear Modeling*. Springer Verlag, Berlin, Heidelberg, Germany, 1997.

[SS88] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Vol. I: Languages and Parsing*. Springer Verlag, Berlin, Heidelberg, Germany, 1988.

[SS94] Y. Schabes and S.M. Shieber. "Another Concept of Tree-Adjoining Derivation." *Computational Linguistics*, **20**(1), 1994.

[SSP95] S.M. Shieber, Y. Schabes, and F.C.N. Pereira. "Principles and Implementation of Deductive Parsing." *Journal of Logic Programming*, **24**, 1995.

[Sta92] E.P. Stabler. *The Logical Approach to Syntax: Foundations, Specifications, and Implementation of Theories of Government and Binding*. MIT Press, Cambridge, Massachusetts, 1992.

[Sta94] E.P. Stabler. "The Finite Connectivity of Linguistic Structure." In C. Clifton, Jr., L. Frazier, and K. Rayner, editors, *Perspectives on Sentence Processing*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1994.

[Sta97] E.P. Stabler. "Derivational Minimalism." In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 1997.

[Sta98] E.P. Stabler. “Acquiring Languages with Movement.” *Syntax*, **1**(1), 1998.

[Sta99] E.P. Stabler. “Remnant Movement and Complexity.” In G. Bouma, E. Hinrichs, G.-J. Kruijff, and D. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*. CSLI Publications, Stanford, California, 1999.

[Sta01a] E.P. Stabler. “Minimalist Grammars and Recognition.” In Christian Rohrer, Antje Rossdeutscher, and Hans Kamp, editors, *Linguistic Form and Its Computation*. CSLI Publications, Stanford, California, 2001. (Presented at the SFB340 workshop at Bad Teinach, 1999).

[Sta01b] E.P. Stabler. “Recognizing Head Movement.” In P. de Groote, G.F. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence (LACL 2001)*. Springer Verlag, Berlin, Heidelberg, Germany, 2001.

[Sto95] A. Stolcke. “An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities.” *Computational Linguistics*, **21**(2), 1995.

[Val75] L.G. Valiant. “General Context-Free Recognition in Less Than Cubic Time.” *Journal of Computer and System Sciences*, **19**(4), 1975.

[Vij87] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1987.

[VW91] K. Vijay-Shanker and D.J. Weir. “Polynomial Parsing of Extensions of Context-Free Grammars.” In M. Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Publishers, 1991.

[VW93] K. Vijay-Shanker and D.J. Weir. “Parsing Some Constrained Grammar Formalisms.” *Computational Linguistics*, **19**(4), 1993.

[VW94] K. Vijay-Shanker and D.J. Weir. “The Equivalence of Four Extensions of Context Free Grammar Formalisms.” *Mathematical Systems Theory*, **27**, 1994.

[VWJ87] K. Vijay-Shanker, D.J. Weir, and A.K. Joshi. “Characterizing Descriptions Produced by Various Grammatical Formalisms.” In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987.

[Wei88] D.J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1988.

[Wei99] A. Weinberg. "A Minimalist Theory of Human Sentence Processing." In S.D. Epstein and N. Hornstein, editors, *Working Minimalism*. MIT Press, Cambridge, Massachusetts, 1999.

[Woo70] W. Woods. "Transition Network Grammars for Natural Language Analysis." *Communications of the Association for Computing Machinery*, **13**, 1970.

[You67] D.H. Younger. "Recognition and Parsing of Context-Free Languages in n^3." *Information and Control*, **10**(2), 1967.