A RECOGNIZER FOR MINIMALIST GRAMMARS

Henk Harkema
Department of Linguistics
University of California
Los Angeles, CA 90025, USA
harkema@humnet.ucla.edu

Abstract

Minimalist Grammars are a rigorous formalization of the sort of grammars proposed in the linguistic framework of Chomsky's Minimalist Program. One notable property of Minimalist Grammars is that they allow constituents to move during the derivation of a sentence, thus creating discontinuous constituents. In this paper we will present a bottom-up parsing method for Minimalist Grammars, prove its correctness, and discuss its complexity.

1 Introduction

It seems to be a general feature of natural language that the elements of a sentence are pronounced in one position, while at the same time serving a function in another part of the structure of the sentence. Linguistic theories in the transformational tradition have tried to capture this fact by proposing analyses that involve movement of constituents. Stabler ([8]) presents a formalism for defining minimalist grammars that allow for movement of constituents. This formalism is based on Chomsky's Minimalist Program ([2]).

Michaelis ([5]) provides an argument showing that minimalist grammars as defined in ([8]) are weakly equivalent to multiple context-free grammars as described in Seki et al. ([6]).[1] Multiple context-free grammars are non-concatenative in the sense that a non-terminal symbol in this grammar can dominate a sequence of strings of terminal symbols, rather than just one string, as in the case of ordinary context-free grammars. Each of the strings dominated by a non-terminal symbol in a multiple context-free grammar will be a substring of a sentence whose derivation includes this non-terminal, but in the sentence these strings are not necessarily adjacent. The main insight contained in [5] is that minimalist grammars are non-concatenative in a similar way. In minimalist grammars, non-concatenativity arises as the result of movement. Thus, in a minimalist grammar a constituent can dominate non-adjacent substrings of a sentence. Seki et al. ([6]) also describe an algorithm for recognizing multiple context-free grammars. Stabler ([10]) sketches how this algorithm may be extended to minimalist grammars.

This paper contains a formal specification of a recognizer for minimalist grammars. Furthermore, it is shown that the recognizer is sound and complete, and that its time complexity is polynomial in the length of the input string. Besides this introduction, this paper has six sections. Section

[1] Hence, minimalist grammars are also weakly equivalent to linear context-free rewriting systems ([11]), multi-component tree-adjoining grammars ([12]), and simple positive range concatenation grammar ([1]), as these formalisms are equivalent to multiple context-free grammars.

2 introduces the minimalist grammars as defined in [8]. Section 3 contains the specification of the recognizer. Sections 4 and 5 present the proofs of soundness and completeness. Section 6 describes some complexity results. The paper concludes with some directions for future work.

2 Minimalist Grammars

Minimalist grammars manipulate minimalist trees. Minimalist trees are finite, binary ordered trees whose leaves are labeled with sequences of syntactic and non-syntactic features. New trees are built by either merging two trees into one, or by moving a subtree in a tree. The application of merge and move operations is driven by the syntactic features labeling the leaves of the trees. The language defined by a minimalist grammar consists of the yields of a particular subset of the trees generated by the grammar.

Formally, following [8], a minimalist grammar G is defined to be quadruple (V, Cat, Lex, 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 expressions built from V and Cat, and F is a finite set of structure building functions.

The set of non-syntactic features V is made up of a set of phonetic features P and a set of semantic features I: V=P $\cup$ I. The set Cat comprises four kinds of syntactic features: Cat=*base* $\cup$ *select* $\cup$ *licensors* $\cup$ *licensees*. The elements of *base* represent basic syntactic categories. The set *base* minimally contains the distinguished category feature `c`. For each category feature `x` $\in$ *base*, there will be a selection feature `=x` $\in$ *select*, although *select* does not necessarily contain a feature `=c`. The features in *base* and *select* play a role in merge operations. The features in *licensors* and *licensees* regulate movement operations. For each feature `-y` $\in$ *licensees*, there will be a feature `+y` $\in$ *licensors*.

A minimalist tree τ is given by a non-empty set of nodes N_τ, a function $Label_\tau$, and three relations on N_τ: dominance, precedence and projection. Immediate dominance and immediate precedence and their reflexive and transitive closures are defined as for trees of the usual kind. For any two sister nodes x, y in a minimalist tree τ, either x projects over y, or y projects over x, notated x $<$ y and y $<$ x, respectively, or x $>$ y and y $>$ x. The function $Label_\tau$ assigns to each leaf of τ a string from (V $\cup$ Cat)*, in particular, a string from $select^*\, licensors^*\, select^*\, base\; licensees^*\, \mathrm{P}^*\mathrm{I}^*$. The other nodes of τ are not labeled. The elements of Lex are are assumed to be trees consisting of one node.

This paragraph will introduce some arboreal notions that will be used in the remainder of the paper. For nodes x, y in a tree τ, x is the head of y if and only if y is a leaf and x=y, or there is a node z in τ such that y immediately dominates z, z projects over its sister and x is the head of z. The head of a tree τ is the head of the root of τ. A node y in a tree τ is a maximal projection of a head x in τ if and only if x is the head of y and y's sister projects over y. A tree τ is maximal if and only if its root is the maximal projection of some head. A tree τ is complex if and only if it has more than one node, otherwise, τ is simple. A tree τ is said to have feature `f` $\in$ V $\cup$ Cat if the first element of the sequence that labels the head of τ is `f`. For τ and υ minimalist trees, $[_<\tau, \upsilon]$ denotes a tree with immediate subtrees τ and υ where the root of τ projects over and precedes the root of υ, and $[_>\tau, \upsilon]$ denotes a tree with immediate subtrees τ and υ where the root of τ precedes the root of υ and the root of υ projects over the root of τ. A tree τ is complete if its head does not contain any syntactic features except for the distinguished category feature `c`, and no node in τ other than the head has any syntactic features. The yield Y(τ) of a tree τ is the concatenation of the phonetic features in the labels of the leaves of τ, ordered by precedence.

There are two structure building functions: F={*merge*, *move*}. A pair of trees τ, υ is in the domain of *merge* if τ has feature $\texttt{=x}$ and υ has feature $\texttt{x}$ for some $\texttt{x} \in$ *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 $\texttt{=x}$ is deleted, and υ' is like υ except that $\texttt{x}$ is deleted. A tree τ is in the domain of *move* if τ has feature $\texttt{+y} \in$ *licensors* and τ has exactly one maximal subtree τ_0 that has feature $\texttt{-y} \in$ *licensees*.[2] Then,

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

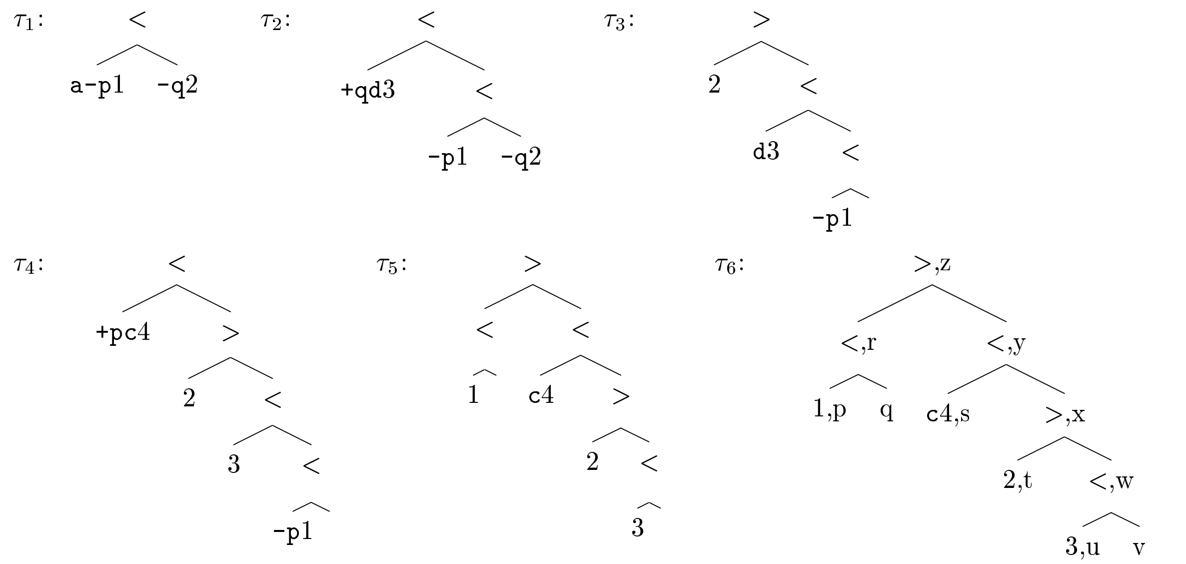
where τ_0' is like τ_0 except that $\texttt{-y}$ is deleted, and τ' is like τ except that $\texttt{+y}$ is deleted and subtree τ_0 is replaced by a single node without features.[3]

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

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

where Dom(*merge*) and Dom(*move*) denote the domains of the functions *merge* and *move*. The language derivable by G consists of the yields of the complete trees in the closure of the lexicon under the structure building functions: $\mathrm{L(G)}=\{\mathrm{Y}(\tau) | \tau \in \mathrm{CL(G)}$ and τ is complete$\}$.

Example 1: Consider the minimalist grammar G defined by the following sets: I=$\emptyset$, P={1, 2, 3, 4}, *base*={a, b, c, d}, *select*={=a, =b, =d}, *licensors*={+p, +q}, *licensees*={-p, -q}, Lex={=ba-p1, b-q2, =a+qd3, =d+pc4}. The sentence '1423' is derived as follows: *merge*(=ba-p1, b-q2)=$[_<$a-p1, -q2$]$, or tree τ_1 below; *merge*(=a+qd3, τ_1)= τ_2; *move*(τ_2)= τ_3; *merge*(=d+pc4, τ_3)= τ_4; *move*(τ_4)= τ_5.



[2] Since $\texttt{+y}\neq\texttt{-y}$, τ and τ_0 have different head labels. Hence, τ_0 is properly included in τ.

[3] The structure building functions in Stabler ([8]) also allow for head movement, which is not discussed in this paper. The recognizer described in this paper is easily adapted to deal with this kind of movement.

Tree τ_5 is a complete tree.[4] Tree τ_6 is the same as tree τ_5. The nodes in τ_6 are named for illustrational purposes only; these names have no significance in the grammar. In tree τ_6, node u is the head of nodes u, w and x. Node s is the head of nodes s, y and z. Node t is the head of node t and no other node. Nodes z, r, q, x, t and v are maximal projections; nodes s, p, q, u, t, v are their respective heads. Node s has feature c. No other node has a syntactic feature. τ_w=[$_<$ τ_u, τ_v], and τ_x=[$_>$ τ_t, τ_w], where τ_n denotes the subtree of τ_6 whose root is named n. □

3 Specification of the Recognizer

This section contains a formal definition of an agenda-driven, chart-based recognizer for minimalist languages. Taking a logical perspective on parsing as presented in Shieber et al. ([7]), the definition of the recognizer includes a specification of a grammatical deductive system and a specification of a deduction procedure. 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 old ones. 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.

3.1 Deduction Procedure

The deduction procedure used in the recognizer presented in this paper is taken from Shieber et al. ([7]). 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 procedure is defined as 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.
   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,[5] 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.

[4]Incidentally, note that the last movement of this derivation is an instance of remnant movement. Remnant movement is movement of a constituent from which material has already been extracted. In this particular case it creates a configuration in which the moved constituent with yield '2' no longer c-commands its trace. This example shows that remnant movement is easily modeled in minimalist grammars (see Stabler ([9]) for further discussion). Recently, remnant movement has gained some linguistic popularity, e.g. [4], [3].

[5]Note that successful applications of Move-1 and Move-2 as defined in section 3.2.2 do not involve any items from the chart.

Shieber et al. ([7]) 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.

3.2 Deductive System

Given an input string w=$w_1 \ldots w_n$ and minimalist grammar G=(V, Cat, Lex, F), the items of the deductive system will be of the form $[\alpha_0, \alpha_1, \ldots, \alpha_m]_t$, where m $\leq$ |*licensees*|, t $\in$ {s, c}. For 0 $\leq$ i $\leq$ m, α_i is of the form (x_i, y_i):γ_i, where 0 $\leq x_i \leq y_i \leq$ n, n=|w|, and $\gamma_i \in$ Cat*.

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 $\phi=[_<\tau, \upsilon]$. 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*[6]
$Y_n(\phi)=Y_n(\upsilon)$ 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:

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

Informally, the narrow yield of a tree is that part of its yield that will not move out of the tree by some application of the function *move*. For example, the trees from example 1 have the following narrow yields: $Y_n(\tau_1)$=1; $Y_n(\tau_2)$=3; $Y_n(\tau_3)$=23; $Y_n(\tau_4)$=423; $Y_n(\tau_5)=Y(\tau_5)$=1423.

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:

1. If t=s, τ is a simple tree; if t=c, τ is a complex tree.
2. The head of τ is labeled by $\gamma_0\pi\iota$, $\pi\iota \in$ P*I*.
3. For every $(x_i, y_i){:}\gamma_i$, 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.

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 $[(i, j){:}\gamma]_s$ in the deductive system.

There will be two goal items: $[(0, n){:}\mathtt{c}]_s$ and $[(0, n){:}\mathtt{c}]_c$, `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(τ)|τ $\in$ CL(G) and τ is complete}.

[6] Recall that 'to have a feature' is a formal notion, defined in section 2.

Rules of Inference

The deductive system has six rules of inference, grouped into Merge rules and Move rules:
Merge-1:

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}\texttt{=x}\gamma]_s,\ [(\mathrm{q},\ \mathrm{v}){:}\texttt{x},\ \alpha_1,\ \ldots,\ \alpha_k]_t}{[(\mathrm{p},\ \mathrm{v}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c}$$

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

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}\texttt{=x}\gamma]_s,\ [(\mathrm{v},\ \mathrm{w}){:}\texttt{x}\delta,\ \alpha_1,\ \ldots,\ \alpha_k]_t}{[(\mathrm{p},\ \mathrm{q}){:}\gamma,\ (\mathrm{v},\ \mathrm{w}){:}\delta,\ \alpha_1,\ \ldots,\ \alpha_k]_c}$$

Merge-3:

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}\texttt{=x}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c,\ [(\mathrm{v},\ \mathrm{p}){:}\texttt{x},\ \iota_1,\ \ldots,\ \iota_l]_t}{[(\mathrm{v},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k,\ \iota_1,\ \ldots,\ \iota_l]_c}$$

Merge-4: ($\delta \neq \emptyset$)

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}\texttt{=x}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c,\ [(\mathrm{v},\ \mathrm{w}){:}\texttt{x}\delta,\ \iota_1,\ \ldots,\ \iota_l]_t}{[(\mathrm{p},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k,\ (\mathrm{v},\ \mathrm{w}){:}\delta,\ \iota_1,\ \ldots,\ \iota_k]_c}$$

Move-1:

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

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

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

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 `-f`. For all rules the following holds: $0 \leq$ p, q, v, w $\leq$ n; $0 \leq$ i, k, l $\leq$ m; and t $\in$ {s, c}, `=x` $\in$ *select*, `x` $\in$ *base*, `+y` $\in$ *licensors*, and `-y` $\in$ *licensees*.

Example 2: The deductive system for the minimalist grammar G given in example 1 has four axioms: $[(0, 1){:}\texttt{=ba-p}]_s$, $[(2, 3){:}\texttt{b-q}]_s$, $[(3, 4){:}\texttt{=a+qd}]_s$, and $[(1, 2){:}\texttt{=d+pc}]_s$. The goal item $[(0, 4){:}\texttt{c}]_c$ is deduced in the following way (in this simple example the agenda will not be mentioned explicitly): $[(0, 1){:}\texttt{=ba-p}]_s$, $[(2, 3){:}\texttt{b-q}]_s \Rightarrow [(0, 1){:}\texttt{a-p}, (2, 3){:}\texttt{-q}]_c$ (Merge-2); $[(3, 4){:}\texttt{=a+qd}]_s$, $[(0, 1){:}\texttt{a-p}, (2, 3){:}\texttt{-q}]_c \Rightarrow [(3, 4){:}\texttt{+qd}, (0, 1){:}\texttt{-p}, (2, 3){:}\texttt{-q}]_c$ (Merge-2); $[(3, 4){:}\texttt{+qd}, (0, 1){:}\texttt{-p}, (2, 3){:}\texttt{-q}]_c \Rightarrow [(2, 4){:}\texttt{d}, (0, 1){:}\texttt{-p}]_c$ (Move-1); $[(1, 2){:}\texttt{=d+pc}]_s$, $[(2, 4){:}\texttt{d}, (0, 1){:}\texttt{-p}]_c \Rightarrow [(1, 4){:}\texttt{+pc}, (0, 1){:}\texttt{-p}]_c$ (Merge-1); $[(1, 4){:}\texttt{+pc}, (0, 1){:}\texttt{-p}]_c \Rightarrow [(0, 4){:}\texttt{c}]_c$ (Move-1). □

Since Shieber et al. ([7]) 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 Proof of Soundness

The following two lemma's will be helpful for establishing soundness of the deductive system.

Lemma 1a: if τ, τ' and υ, υ' are trees such that *merge*$(\tau, \upsilon)=[_<\tau', \upsilon']$ or *merge*$(\tau, \upsilon)=[_>\upsilon', \tau']$, then $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$ and $\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\upsilon)$.

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, $\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\upsilon)$. □

Lemma 1b: if τ, τ' and τ_0, τ_0' are trees such that *move*$(\tau)=[_>\tau_0', \tau']$, then $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$ and $\mathrm{Y}_n(\tau_0')=\mathrm{Y}_n(\tau_0)$.

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 $\mathrm{Y}_n(\tau)$ because τ_0 has a feature `-f` $\in$ *licensees*. The yield of τ_0 is also excluded from $\mathrm{Y}_n(\tau')$ because τ_0 is not a subtree of τ'. Since trees τ and τ' are otherwise the same, $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$. Concerning τ_0' and τ_0, these trees differ only by their head label. Hence, $\mathrm{Y}_n(\tau_0')=\mathrm{Y}_n(\tau_0)$. □

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.

4.1 Soundness of the Axioms

According to the interpretation given in section 3.2, an axiom $[(\mathrm{i}, \mathrm{j}){:}\gamma]_s$ asserts the existence of a tree $\tau \in \mathrm{CL(G)}$ with the following properties:

1. Tree τ is a simple tree.
2. The head of τ is labeled by $\gamma\pi\iota$, for some $\pi\iota \in \mathrm{P^*I^*}$.
3. Besides the head of τ there are no other nodes with syntactic features in τ.
4. The narrow yield of τ is $\mathrm{w}_{i+1} \ldots \mathrm{w}_j$.

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

4.2 Soundness of the Rules of Inference

There are six rules of inference. Their soundness will be established below.[7]

Merge-1: the items $[(\mathrm{p}, \mathrm{q}){:}\mathtt{=x}\gamma]_s$ and $[(\mathrm{q}, \mathrm{v}){:}\mathtt{x}, \alpha_1, \ldots, \alpha_k]_t$ in the antecedent of the rule Merge-1 assert the existence of trees τ and $\upsilon \in \mathrm{CL(G)}$, whose heads are labeled by $\mathtt{=x}\gamma$ 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 $\phi=[_<\tau', \upsilon'] \in \mathrm{CL(G)}$, since τ is a simple tree. The head of ϕ is labeled by γ. Furthermore, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau')\cdot\mathrm{Y}_n(\upsilon')$, since the head of υ' does not have a feature `-f` $\in$ *licensees*. By lemma 1a, $\mathrm{Y}_n(\tau')\cdot\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\tau)\cdot\mathrm{Y}_n(\upsilon)=\mathrm{w}_{p+1} \ldots \mathrm{w}_q\cdot\mathrm{w}_{q+1} \ldots \mathrm{w}_v=\mathrm{w}_{p+1} \ldots \mathrm{w}_v$. 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,

[7] 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 by γ', for example, means: 'the head of tree ϕ is labeled by $\gamma\pi\iota$, for some $\pi\iota \in \mathrm{P^*I^*}$'.

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: application of the Merge-2 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, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)=\mathrm{w}_{p+1} \ldots \mathrm{w}_q$. Obviously, υ', whose head is labeled δ, will be a maximal subtree of ϕ. By lemma 1a, $\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\upsilon)=\mathrm{w}_{v+1} \ldots \mathrm{w}_w$. Now it is easy to see that the grammatical claim made by the item in the consequent of the rule Merge-2 is justified by $\phi \in$ CL(G).

Merge-3: the argument unfolds in a fashion similar to Merge-1, except that now ϕ=[$_>\upsilon'$, τ'], because τ is complex. Consequently, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\upsilon)\cdot\mathrm{Y}_n(\tau)=\mathrm{w}_{v+1} \ldots \mathrm{w}_p\cdot\mathrm{w}_{p+1} \ldots \mathrm{w}_q=\mathrm{w}_v \ldots \mathrm{w}_q$. Furthermore, ϕ inherits the narrow yields and head labels of the maximal subtrees in both τ and υ.

Merge-4: this rule of inference is treated analogously to Merge-2, with the provisos mentioned under Merge-3.

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`.[8] 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, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau_0')\cdot\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau_0)\cdot\mathrm{Y}_n(\tau)$ by lemma 1b and because the head of τ_0' has an empty label. Since $\mathrm{Y}_n(\tau_0)=\mathrm{w}_{v+1} \ldots \mathrm{w}_p$ and $\mathrm{Y}_n(\tau)=\mathrm{w}_{p+1} \ldots \mathrm{w}_q$, $\mathrm{Y}_n(\phi)=\mathrm{w}_{v+1} \ldots \mathrm{w}_q$. 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 5). 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*. $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau')$, since τ_0', whose head is labeled by δ, has a feature `-f` $\in$ *licensees*, cf. similar situations in Merge-2 and Merge-4. 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 $\mathrm{Y}_n(\tau_0')=\mathrm{Y}_n(\tau_0)$, 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.

5 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 lemma's 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 $\mathrm{Y}_n(\sigma)$ is a substring of $\mathrm{Y}_n(\sigma')$.

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

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 ϕ=$[_< \tau', \upsilon']$ or ϕ=$[_> \upsilon', \tau']$, depending on whether τ is a simple or a complex tree. Furthermore, either σ is a proper subtree of τ, or σ=τ. 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 σ=τ, 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 ϕ=$[_< \upsilon', \tau']$ or ϕ=$[_> \tau', \upsilon']$, depending on whether υ is a simple or a complex tree. Again, either σ is a proper subtree of τ, or σ=τ. 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 σ=τ, 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, ϕ=$[_> \tau_0', \tau']$, with τ_0 specified as in the definition of *move*. Either σ is a proper subtree of τ_0, or σ=τ_0, or σ is a proper subtree of τ and τ_0 is a proper subtree of σ, or σ=τ. 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 σ=τ_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 σ=τ, 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 of $Y_n(\gamma)$=$Y(\gamma)$, and, consequently, $Y_n(\tau)$ is a substring of $Y(\gamma)$. □

Completeness of the parser 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 $Y(\phi)$=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 $Y(\phi)$=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 3.2.

Proof:[9] 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.

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

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

2. Assume all items corresponding to $\phi \in \mathrm{CL}^k(\mathrm{G})$, 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 $\phi \in \mathrm{CL}^{k+1}(\mathrm{G})$, ϕ included in the derivation of a complete tree or a complete tree itself, a corresponding item will be generated.

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

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

In the second case, $\phi \in \{\mathit{merge}(\tau, \upsilon) | (\tau, \upsilon) \in \mathrm{Dom}(\mathit{merge}) \cap \mathrm{CL}^k(\mathrm{G}) \times \mathrm{CL}^k(\mathrm{G})\}$. Then there are trees $\tau, \upsilon \in \mathrm{CL}^k(\mathrm{G})$ such that $\phi=\mathit{merge}(\tau, \upsilon)$. Since ϕ is included in 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.[10] Hence, by the induction hypothesis, there are items corresponding to τ and υ. Since $(\tau, \upsilon) \in \mathrm{Dom}(\mathit{merge})$, the heads of τ and υ are respectively labeled $\mathtt{=x}\gamma$ and $\mathtt{x}\delta$, for $\mathtt{=x} \in$ *licensees*, $\mathtt{x} \in$ *base* and $\gamma, \delta \in \mathrm{Cat}^*$. 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 $\mathtt{x}$, then $\phi=[_< \tau', \upsilon']$, and $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau')\cdot\mathrm{Y}_n(\upsilon')$. By lemma 1a, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau)\cdot\mathrm{Y}_n(\upsilon)$. Suppose ϕ participates in a successful derivation of a complete tree whose yield is the string $\mathrm{w}_1 \ldots \mathrm{w}_m$.[11] Then, by lemma 3, $\mathrm{Y}_n(\tau)\cdot\mathrm{Y}_n(\upsilon)$ is a substring of $\mathrm{w}_1 \ldots \mathrm{w}_m$, that is, $\mathrm{Y}_n(\tau)=\mathrm{w}_p \ldots \mathrm{w}_q$ and $\mathrm{Y}_n(\upsilon)=\mathrm{w}_{q+1} \ldots \mathrm{w}_v$, $1 \leq p \leq q \leq v \leq m$. Hence, the items corresponding to τ and υ will match the antecedents of rule Merge-1. Alternatively, ϕ is a complete tree itself. Assume $\mathrm{Y}_n(\phi)=\mathrm{Y}(\phi)=\mathrm{w}_1 \ldots \mathrm{w}_m$. Then it follows immediately that $\mathrm{Y}_n(\tau)=\mathrm{w}_1 \ldots \mathrm{w}_q$ and $\mathrm{Y}_n(\upsilon)=\mathrm{w}_{q+1} \ldots \mathrm{w}_m$, $1 \leq q \leq m$. 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 antecdents of the rules Merge-1, Merge-3, and Merge-4. Application of these rules will produce an itemn corresponding to tree ϕ.

In the third case, $\phi \in \{\mathit{move}(\tau) | \tau \in \mathrm{Dom}(\mathit{move}) \cap \mathrm{CL}^k(\mathrm{G})\}$. Then there is a $\tau \in \mathrm{CL}^k(\mathrm{G})$ such that $\phi=\mathit{move}(\tau)$. 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 $\tau \in \mathrm{Dom}(\mathit{move})$, τ has a feature $\mathtt{+y}$ and τ has exactly one maximal subtree that has the feature $\mathtt{-y}$, $\mathtt{+y} \in$ *licensors*, $\mathtt{-y} \in$ *licensees*. Let τ_0 be the maximal subtree of τ that has feature $\mathtt{-y}$ and let the label of its head be $\mathtt{-y}\delta$. Then there are two cases to be considered: $\delta=\emptyset$ and $\delta\neq\emptyset$. If $\delta=\emptyset$, then $\phi=[_> \tau_0', \tau']$, and $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau_0')\cdot\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau_0)\cdot\mathrm{Y}_n(\tau)$ by the definition of narrow yield and lemma 1b. As for Merge-1 and Merge-3 above, $\mathrm{Y}_n(\tau_0)\cdot\mathrm{Y}_n(\tau)$ will be a substring of $\mathrm{w}_1 \ldots \mathrm{w}_m$, where $\mathrm{w}_1 \ldots \mathrm{w}_m$ is the yield of a complete tree derived from ϕ or the yield of ϕ itself if ϕ is a complete tree. Therefore, $\mathrm{Y}_n(\tau_0)=\mathrm{w}_v \ldots \mathrm{w}_p$ and $\mathrm{Y}_n(\tau)=\mathrm{w}_{p+1} \ldots \mathrm{w}_q$, $1 \leq v \leq p \leq q \leq m$. 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

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

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

of rule Move-2, application of which will generate an item corresponding to ϕ. □

6 Complexity Results

For a given minimalist grammar G=(V, Lex, Cat, 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 $[(\mathrm{x}_0, \mathrm{y}_0){:}\gamma_0, (\mathrm{x}_1, \mathrm{y}_1){:}\gamma_1, \ldots, (\mathrm{x}_m, \mathrm{y}_m){:}\gamma_m]_t$, as defined in section 3.2. Each part $(\mathrm{x}_i, \mathrm{y}_i){:}\gamma_i$ of an item, $0 \leq \mathrm{i} \leq \mathrm{m}$, 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 $(\mathrm{x}_i, \mathrm{y}_i){:}\gamma_i$, where k=|*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 procedure specified in section 3.1 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, $1 \leq \mathrm{i} \leq \mathrm{m}$ 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 upperbound 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 $O(n^{4k+4})$.

7 Conclusions and Future Work

In this paper we have provided a formal definition of a recognizer for minimalist grammars, together with proofs of completeness and soundness and an analysis of its space and time complexity.

There are several issues that deserve further investigation. First of all, the recognizer has to be developed into a parser. This can be done by either extending the items with a field for recording the derivational history of an item, or by devising a method for retrieving derivation trees from the chart. Secondly, we conjecture that the efficiency of the parser can be greatly improved by imposing some order on the chart. In the current recognizer, the entire chart is searched in order to determine whether any one of the Merge rules will apply. The definitions of the Merge rules suggest that grouping the items in the chart according to the first feature of their 'head labels' will allow for a more efficient search. The current recognizer operates in a bottom-up manner: no constituent is recognized until all substrings that make up its yield have been encountered. So, thirdly, it would be interesting to investigate other recognition strategies. Finally, there are still some open questions with regard to the formal power of minimalist grammars in comparison with other grammar formalisms. Careful examination of the complexity of the algorithms for recognizing and parsing these various grammar formalisms might answer some of the questions.

Acknowledgements

Thanks to Ed Stabler and Crit Cremers for inspiring discussions.

References

[1] Boullier, P. 1998. Proposal for a Natural Language Processing Syntactic Backbone. *Research Report N° 3342*, `http://www.inria.fr/RRRT/RR-3342.html`. INRIA-Rocquencourt.

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

[3] Kayne, R. 1999. A note on Prepositions and Complementizers. In: *A Celebration.* MIT Press.

[4] Koopman, H. and A. Szabolsci. Forthcoming. *Verbal Complexes.* MIT Press.

[5] Michaelis, J. 1998. Derivational Minimalism is Mildly Context-Sensitive. In: *Proceedings, Logical Aspects of Computational Linguistics*, Grenoble.

[6] Seki, H., T. Matsumura, M. Fujii and T. Kasami. 1991. On Multiple Context-Free Grammars. In: *Theoretical Computer Science*, 88.

[7] Shieber, S.M., Y. Shabes and F.C.N. Pereira. 1995. Principles and Implementation of Deductive Parsing. In: *Journal of Logic Programming*, 24.

[8] Stabler, E.P. 1997. Derivational Minimalism. In: *Logical Aspects of Computational Linguistics.* C. Retoré (ed.). Lecture Notes in Artificial Intelligence 1328.

[9] Stabler, E.P. 1999. Remnant Movement and Complexity. In: *Constraints and Resources in Natural Language Syntax and Semantics.* G. Bouma, E. Hinrichs, G.-J. Kruijff, D. Oerhle (eds.). CSLI.

[10] Stabler, E.P. Forthcoming. Performance models for a Derivational Minimalism. In: *Linguistic Form and its Computation*, Final Workshop of SFB340.

[11] Vijay-Shanker, K., D.J. Weir and A.K. Joshi. 1987. Descriptions Produced by Various Grammatical Formalisms. In: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.*

[12] Weir, D.J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. dissertation, University of Pennsylvania.