Strict Deterministic Aspects of Minimalist Grammars

John T. Hale[1] and Edward P. Stabler[2]

[1] Michigan State University,
East Lansing MI 48824-1027, USA
[2] University of California, Los Angeles,
Los Angeles CA 90095-1543, USA

Abstract. The Minimalist Grammars (MGs) proposed by Stabler(1997) have tree-shaped derivations (Harkema, 2001b; Michaelis, 2001a). As in categorial grammars, each lexical item is an association between a vocabulary element and complex of features, and so the "yields" or "fringes" of the derivation trees are sequences of these lexical items, and the string parts of these lexical items are reordered in the course of the derivation. This paper shows that while the derived string languages can be ambiguous and non-context-free, the set of yields of the derivation trees is always context-free and unambiguous. In fact, the derivation yield languages are strictly deterministic context-free languages, which implies that they are LR(0), and that the generation of derivation trees from a yield language string can be computed in linear time. This result suggests that the work of MG parsing consists essentially of guessing the lexical entries associated with words and empty categories.

1 Introduction

A derivation is a witness to the fact that a string is generated by a grammar. A derivation says *how* a string is generated, or, equally, what that string's analysis is in terms of the grammar. Derivations are, in this sense, authoritative about a grammar's view of a well-formed sentence. From the perspective of a grammar, a generated string is just one facet of the full story: its derivation.

So for any purpose where the structure of a string matters, it is desirable to work with the full story, the derivation – or some needed subset of the information it encodes. In computational linguistics, such work might involve drawing pictures of a sentence's structure, or generating other sentences, for instance, in other languages. These kinds of applications benefit from the efficient coding of derivations.

This paper shows that, for a particular grammar formalism, the Minimalist Grammars (Stabler, 1997; Stabler and Keenan, 2003) there exists an encoding of derivations that is highly restricted: they can be coded by the sequence of lexical entries as it appears along the fringe of any well-formed derivation tree. This is the same unique readability property familiar from

P. Blache et al. (Eds.): LACL 2005, LNAI 3492, pp. 162–176, 2005.

the syntax of logical languages (Enderton, 2001, 40,108), (Shoenfield, 1967, 15), (Ebbinghaus, Flum, and Thomas, 1994, 22). Not only does this lexical sequence determine the derivation, it is also structured in such a way that such determination can be carried out in linear time by a shift-reduce automaton. This kind of operation on lexical sequences is useful for drawing dependency graphs, X-bar trees annotated with traces, and other kinds of diagrams. The compact representation provided by lexical sequences could have other uses as well. If lexical entries contain "semantic" information, such lexical sequences might indeed represent meanings, constituting a "logical form" for sentences in minimalist languages. These logical forms might be combinable with other (perhaps real-world) knowledge in a natural language understanding system. Similarly, such lexical sequences might be useful in a transfer-based machine translation system.

For all these reasons, and also to gain a deeper understanding of the formalism itself, the simplicity of MG yield languages is of interest. The main result in this paper is that for every MG grammar, the language of MG lexical sequences is a strict deterministic language in the sense of Harrison and Havel (1973) This property is defined in section 2. Then in section 3, MGs are presented as an instance of the more general class of "bare grammars" which includes other formalisms. Section 4 presents a context-free grammar (CFG) for the tree-shaped MG derivations, and shows how this CFG can always be extended to be a strict deterministic grammar. Because any strict deterministic grammar is also an LR(0) grammar in the sense of Kunth (1965), this shows that the language of MG lexical sequences is an LR(0) language.

2 Definitions

To indicate the relationship of the central result about MGs to more familiar grammars, it will be helpful to fix some auxiliary definitions. These stipulations "if, and only if, by definition" are abbreviated by $\doteq$ and "equals, by definition" by $\doteq$.

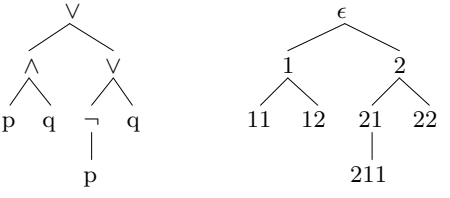
2.1 Trees

At issue most fundamentally are the derivations of linguistic expressions, which are naturally regarded as trees. A **tree** is a function $t : T \to S$ whose domain is a finite set of sequences T of the positive integers $T \subset (\mathbb{N}_+)^*$ such that, for every $n \in T$:

1. if $n = si$ for some $s \in (\mathbb{N}_+)^*, i \in \mathbb{N}$, then $s \in T$,
   'higher nodes are also in the domain' and
2. if $n = si$ for some $s \in (\mathbb{N}_+)^*, i \in \mathbb{N}$, then if $i > 1$ and $j = i - 1$, $tj \in T$
   'lower-numbered sisters are also in the domain'

The elements of a tree domain T are often called **nodes**; ϵ is the **root node**, and the values of t are often called **labels**.

Given any tree $t : T \to S$ and any $d \in T$, the **subtree** of t at d is the function t/d with domain $\{u |\ du \in dom(t)\}$, such that $t/d(u) = a$ if and only if $t(du) = a$. In later sections, trees will sometimes be represented with **expression** which are sequences of labels, S and parentheses, as follows. By the previous definitions, for any tree $t : T \to S$, $t(\epsilon) = a$ is the root label and $k = |\mathbb{N}_+ \cap T|$ is the number of daughters the root has, because intersecting the tree domain T with the positive integers retrieves just the sequences of length 1. When $k = 0$ the expression of t is a, and otherwise it is $a(t_1 \dots t_k)$, where $t_1, \dots, t_k$ are the expressions of the subtrees $t/1, \dots, t/k$. For example, the tree with the conventional depiction here has the tree domain indicated on the right:



The expression of this tree is $\vee(\wedge(pq) \vee (\neg(p)q))$. The **yield** of this tree, on the left, is $pqpq$, where the yield of a tree $t : T \to S$ is the sequence of elements of S defined as follows,

$$yield(t) = \begin{cases} a & \text{if } dom(t) = \{\epsilon\}, t(\epsilon) = a \\ yield(a/1) \dots yield(a/k) & \text{otherwise, where } k = |\mathbb{N}_+ \cap T|. \end{cases}$$

2.2 Context-Free Grammars

Following Keenan and Stabler (2003), take a context-free grammar (CFG) to be a triple $G = \langle \Sigma, N, (\to) \rangle$ where Σ, N are nonempty and disjoint, the start symbol S is an element of the nonterminal set N, and $(\to) \subseteq N \times (\Sigma \cup N)^*$ is finite. The immediate rewriting relation $(\to)$ is to be viewed as an infix operator so that $x \to y$ is a shorthand for $\langle x, y \rangle \in (\to)$. Similarly, the rewriting relation $\Rightarrow_G \doteq \{\langle x, y \rangle \in (\Sigma \cup N)^* \times (\Sigma \cup N)^* |\ \exists u, v, w \in (\Sigma \cup N)^*,\ A \in N,\ \text{such that } x = uAv,\ A \to w,\ x = uwv\}$ on sentential forms is abbreviated as $x \Rightarrow y$.

Let $\Rightarrow^*$ be the reflexive transitive closure of $\Rightarrow$. For any $A \in N$, the sequences of category A, $L_A(G) = \{x \in \Sigma^* |\ A \Rightarrow^* x\}$. For any $A \in N$, $L_A(G)$ is a context-free language.

To set the stage for the following more general framework of subsection 3.1, define the **derivation trees** $\Gamma(G)$ of a CFG G as follows, labeling internal nodes with elements of $\to$ rather than with just their left sides:

$$\Gamma(G)^0 \doteq \{a \in \Sigma^* |\ A \to a\},$$
$$\Gamma(G)^{k+1} \doteq \Gamma(G)^k \cup \{A \to a(a) |\ A \to a \text{ and } a \in (\Gamma(G)^k)^*\},$$
$$\Gamma(G) \doteq \bigcup_{k \in \mathbb{N}} \Gamma(G)^k.$$

Furthermore, for any $A \in N$,

$$\Gamma_A(G) \doteq \{t \in \Gamma(G) |\ t(\epsilon) = A \to a,\ \text{for some } a \in (\Sigma \cup N)^*\}.$$

Clearly, $yield(\Gamma_A(G)) = L_A(G)$. G **has ambiguous yields** $\doteq$ there are two distinct trees in $\Gamma(G)$ with the same yield. Typically, one just says that such a grammar is "ambiguous."

2.3 Strict Determinism

Strict determinism is a property that context-free grammars can have; it is a technical notion that figures prominently in the theory of deterministic context-free grammars, which is reviewed in (Harrison, 1978, §11). Most simply, a CFG is strict deterministic if its symbols can be divided up into blocks in a certain way. The formal definition uses Harrison's notation ${}^{(n)}\alpha$ for the prefix of α of length $\min\{n, |\alpha|\}$.

For any CFG G, a partition π of $N \cup \Sigma$ is **strict** if and only if

i. $\Sigma \in \pi$
ii. for any $A, A' \in N$, $\alpha, \beta, \beta' \in (N \cup \Sigma)^*$, if $A \to \alpha\beta$ and $A' \to \alpha\beta'$ and $A \equiv A' \pmod{\pi}$, then either
   a. both $\beta, \beta' \neq \epsilon$ and ${}^{(1)}\beta \equiv^{(1)} \beta' \pmod{\pi}$, or
   b. $\beta = \beta' = \epsilon$ and $A = A'$.

If the partition π is clear from context, $A \equiv B \pmod{\pi}$ is sometimes simplified to $A \equiv B$, and similarly the block of π where a symbol A resides is written $[A]$ rather than $[A]_\pi$ if no confusion would result.

With this definition, it is possible to say of a CFG G that it is **strict deterministic** if there exists a strict partition π of $N \cup \Sigma$. A language is strict deterministic if some strict deterministic grammar generates it.

A CFG is said to be reduced with respect to a given start symbol S, abbreviated **reduced**$_S$ just in case $\to = \emptyset$ or for every $A \in N$, $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$ for some $\alpha, \beta \in (N \cup \Sigma)^*$ and $w \in \Sigma^*$.

It is also known that

Theorem (Harrison, 11.4.1). *Any strict deterministic grammar is equivalent to a reduced strict deterministic grammar.*

Theorem (Harrison, 13.2.3). *Any reduced strict deterministic grammar is also an LR(0) grammar.*

These theorems link strict determinism to its consequences for efficient parsing. Harrison (page 347) explains that

> The motivation behind this definition is that we wish to make certain restrictions on the simultaneous occurrences of substrings in different productions. Intuitively, if $A \to \alpha\beta$ is a production in our grammar, then "partial information" about A, together with complete information about a prefix α of $\alpha\beta$, yields similar partial information about the next symbol of β when $\beta \neq \epsilon$, or the complementary information about A when $\beta = \epsilon$. In the formal definition, the intuitive notion of "partial information" is precisely represented by means of the partition π.

Loosely speaking, in an automaton processing a strict deterministic grammar, top-down knowledge of $A \equiv A'$ buys knowledge of ${}^{(1)}\beta \equiv^{(1)} \beta'$.

The family of all strict deterministic languages is a subfamily of the languages accepted by deterministic pushdown automata that accept by *empty* stack in a final state (Harrison, theorem 11.5.4). In strict deterministic grammars, right-hand sides of rewriting rules cannot be prefixes of one another.

But if this restriction is relaxed (the qualifier "strict" is dropped), by distinguishing a special set of prefix-deriving parents, the defined language family grows to be identical with the one defined by deterministic pushdown automata that accept in a final state with *any* stack configuration (Harrison, problem 4 page 392).

2.4 Examples

This section exercises the definitions just given with some CFGs for a simple language, such as might be used in sentential logic.

Example 1 (Ambiguity). Consider the context-free grammar $G1 = \langle \Sigma, N, \rightarrow \rangle$, where
$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
$N = \{S\}$, and
$\rightarrow$ has the following 6 pairs in it:

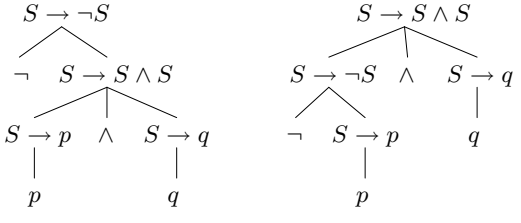
$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow S \vee S & S \rightarrow S \wedge S \end{array}$$

This grammar is (yield-)ambiguous since there are the following derivation trees for $\neg p \wedge q$:

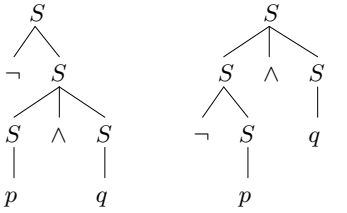
$$S \rightarrow \neg S(\neg, S \rightarrow S \wedge S(S \rightarrow p(p), S \rightarrow q(q)))$$

$$S \rightarrow S \wedge S(S \rightarrow \neg S(\neg, S \rightarrow p(p)), S \rightarrow q(q))$$

One can draw graphical presentations that are more readable, like this:



These CFG derivation trees are slightly redundant, since the right sides of the rules at each internal node can be read off the daughters. Eliminating the right side of each production gives the standard depictions:



There is only one partition of $\Sigma \cup N$ that contains Σ, namely $\pi = \{\Sigma, \{N\}\}$. Evidently, this partition is not strict, because it fails condition (ii) on page 165. When $\alpha = \epsilon$, then $G1$ has, for example $\beta = p$ and $\beta' = S$. These are both non-empty, and their first symbols are not in the same block of the partition π.

Example 2 (Unambiguous Polish Notation). Consider the context-free grammar $G2 = \langle \Sigma, N, \rightarrow \rangle$, where
$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
$N = \{S\}$, and
$\rightarrow$ has the following 6 pairs in it:

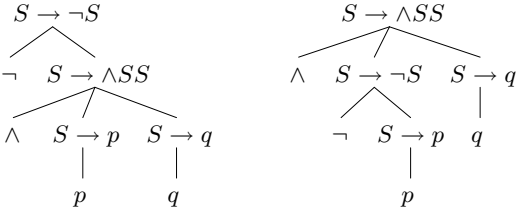
$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow \vee\ S\ S & S \rightarrow \wedge\ S\ S \end{array}$$

In $G2$, the operators $\vee$ and $\wedge$ have been pushed to the leftmost position in the right-hand side of each rule. On this grammar, there is just one derivation tree for $\wedge\neg pq$, and just one for $\neg \wedge pq$:

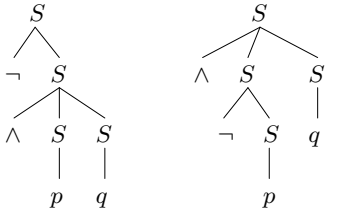
$$S \rightarrow \neg S(\neg, S \rightarrow \wedge SS(\wedge, S \rightarrow p(p), S \rightarrow q(q)))$$

$$S \rightarrow \wedge SS(\wedge, S \rightarrow \neg S(\neg, S \rightarrow p(p)), S \rightarrow q(q))$$

The same, more readable diagrams can be drawn:



The labels on these derivation trees can be shortened to just the left-hand side of the applied rule, in the same way as with $G1$:



There is only one partition of $\Sigma \cup N$ that contains Σ, namely $\pi = \{\Sigma, \{N\}\}$. This partition is strict. When $\alpha = \epsilon$, then no matter which rules we choose, ${}^{(1)}\beta \in \Sigma$ and so the conditions are satisfied. And there are no two different β, β' and two A, A' such that for some non-empty α, $A \to \alpha\beta$ and $A' \to \alpha\beta'$.

To set the stage for later developments, it is worth briefly considering two further variants of $G2$.

Example 3 (Unambiguous, But Not Strict Deterministic). Consider the context-free grammar $G2a = \langle \Sigma, N, \to \rangle$, where
$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
$N = \{S, B\}$, and
$\to$ has the following 7 pairs in it:

$$\begin{array}{lll} S \to p & S \to q & S \to r \\ S \to \neg S & S \to BSS & \\ B \to \wedge & B \to \vee & \end{array}$$

Clearly, $G2a$ generates the same strings of category S as $G2$, but $G2a$ is not strictly deterministic, since the set $\Sigma \cup N$ has no strict partition.

Example 4 (Unambiguous, and Strict Deterministic again). Consider the context-free grammar $G2b = \langle \Sigma, N, \to \rangle$, where
$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
$N = \{S, B, U, A\}$, and
$\to$ has the following 9 pairs in it:

$$\begin{array}{lll} S \to A & S \to US & S \to BSS \\ A \to p & A \to q & A \to r \\ U \to \neg & B \to \wedge & B \to \vee \end{array}$$

Clearly, $G2b$ generates the same strings of category S as $G2$ and $G2a$, but $G2b$ is strictly deterministic, since the set $\Sigma \cup N$ has the strict partition $\{\Sigma, \{A, U, B\}, \{S\}\}$.

With these examples clarifying what ambiguity and strict determinism amount to in CFGs, section 3 turns to the Minimalist Grammars.

3 Grammars

3.1 Bare Grammars

Minimalist Grammars are one of a variety of formalisms that construes a grammar $\mathcal{G}$ as a set of basic expressions Lex and a set $\mathcal{F}$ of partial functions from tuples of expressions to expressions (Keenan and Stabler, 2003). The language $L(\mathcal{G})$ is then the closure of Lex with respect to the functions in $\mathcal{F}$. Internal nodes in the **derivations** of $\mathcal{G}$, $\Gamma(\mathcal{G})$, are labeled with elements of $\mathcal{F}$ just in case $f \in \mathcal{F}$ is applicable to the children:

$$\Gamma(\mathcal{G})^0 \doteq Lex,$$
$$\Gamma(\mathcal{G})^{k+1} \doteq \Gamma(\mathcal{G})^k \cup \{f(a) |\ f \in \mathcal{F} \text{ and } a \in ((\Gamma(\mathcal{G})^k)^* \cap dom(f))\},$$
$$\Gamma(\mathcal{G}) \doteq \bigcup_{k \in \mathbb{N}} \Gamma(\mathcal{G})^k.$$

Clearly, if f labels the root of some tree in $\Gamma(\mathcal{G})$, then f is a function expression whose evaluation returns an element $e \in L(\mathcal{G})$. So in these derivation trees, each node has a **value** which is the denotation of the function expression which is its label, always an element of $L(\mathcal{G})$.

Grammar $\mathcal{G}$ **has ambiguous expressions** $\doteqdot$ some expression $e \in L(\mathcal{G})$ is the value of the roots of two distinct trees in $\Gamma(\mathcal{G})$. Grammar $\mathcal{G}$ **has ambiguous yields** $\doteqdot$ there are two distinct trees in $\Gamma(G)$ with the same yield. Notice that the yields of derivations from bare grammars are sequences from Lex^*.

3.2 Minimalist Grammars

Minimalist Grammars instantiate this general picture, with elements of Lex comprising the sequences of a quite limited inventory of "features", along with two structure-building functions that are constrained in their application. A **Minimalist Grammar** $\mathcal{G} \doteq \langle \Sigma, B, Lex, \mathcal{F} \rangle$, where

1. Σ is a non-empty set (the pronounced elements)
2. B is a non-empty set of **basic features**, which serve to specify the **features** $F \doteq B \cup S \cup M \cup N$ where $=, -, +$ are 1-1 functions with domain B such that **selectors** $S \doteq \{=f |\ f \in B\}$, **licensees** $M \doteq \{-f |\ f \in B\}$, **licensors** $N \doteq \{+f |\ f \in B\}$, and B, S, M, N are pairwise disjoint.
3. The **lexicon** $Lex \subset \Sigma^* :: F^*$ is a finite, nonempty set of lexical chains, where the **chains** $C \doteq \Sigma^* T F^*$, and the types $T \doteq \{::, :\}$ distinguish **lexical** chains from **derived chains**, respectively.
4. The generating functions $\mathcal{F} = \{merge, move\}$ are partial functions from tuples of expressions to expressions, where **expressions** $E \doteq C^+$. It will be convenient to define these functions in a deductive format, with the arguments as premises and the values as the conclusion.

   (a) $merge : (E \times E) \to E$ is the union of the following 3 functions, for $s, t \in \Sigma^*$, for $\cdot \in \{:, ::\}$, for $f \in B$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$

$$\frac{s :: =f\gamma \qquad t \cdot f, \alpha_1, \ldots, \alpha_k}{st : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1}$$

r1 applies when s is lexical; it selects its argument on the right

$$\frac{s : =f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f, \iota_1, \ldots, \iota_l}{ts : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r2}$$

r2 applies when s is phrasal and t has no more features; it selects its argument on the left

$$\frac{s \cdot {=}f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f\delta, \iota_1, \ldots, \iota_l}{s : \gamma, t : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3}$$

r3 applies when s is phrasal and t has more features; it selects its argument on the left

Here st is the concatenation of strings s, t. Note that since the domains of r1, r2, and r3 are disjoint, their union is a function.

(b) $move : E \to E$ is the union of the following 2 functions, for $s, t \in \Sigma^*$, $f \in B$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$ satisfying the following condition: (SMC) none of $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ has $-f$ as its first feature.

$$\frac{s : {+}f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : {-}f, \alpha_{i+1}, \ldots, \alpha_k}{ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k}\ \text{m1}$$

m1 when no features follow `-f`

$$\frac{s : {+}f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : {-}f\delta, \alpha_{i+1}, \ldots, \alpha_k}{s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k}\ \text{m2}$$

m2 when some features follow `-f`

Notice that the domains of m1 and m2 are disjoint, so their union is a function. The (SMC) restriction on the domain of *move* is a simple version of the "shortest move condition" (Chomsky, 1995).

Often, one is interested in a subset of $L(G)$, for instance just the derivations of complementizer phrases. These derivations are all expressions of a particular syntactic category. More generally, for any $f \in B$, the **expressions of category** f, $L_{\mathtt{f}}(\mathcal{G}) \doteq \{s \cdot \mathtt{f} \in L(\mathcal{G}) |$ for some $\cdot \in \{:, ::\}\}$; the **strings of category** f, $S_{\mathtt{f}}(\mathcal{G}) \doteq \{s |\ s \cdot \mathtt{f} \in L_f(\mathcal{G})$ for some $\cdot \in \{:, ::\}\}$; the **derivations of** f, $\Gamma_f(\mathcal{G}) \doteq \{d \in \Gamma(\mathcal{G}) |\ d(\epsilon) \in L_f(G)\}$. A derivation d is **complete** $\doteqdot$ it is in some $\Gamma_f(\mathcal{G})$ for some $f \in B$. A set $L \subseteq \Sigma^*$ is a **minimalist language** $\doteqdot$ for some MG and some $f \in B$, $S_f(\mathcal{G}) = L$.

The MG-definable languages are exactly the same as the languages definable by set-local multicomponent tree adjoining grammars, by multiple context-free grammars, and other well known grammars (Michaelis, 1998; Michaelis, 2001b; Harkema, 2001a).

Example 5 (an MG for an ambiguous language). Consider now an MG similar to the context-free grammar $G1$, $\mathcal{G}3 = \langle \Sigma, N, Lex, \mathcal{F} \rangle$, where
$\Sigma \quad = \{p, q, r, \neg, \vee, \wedge\}$,
$N \quad = \{S\}$, and
Lex has the following 6 lexical items built from Σ and N

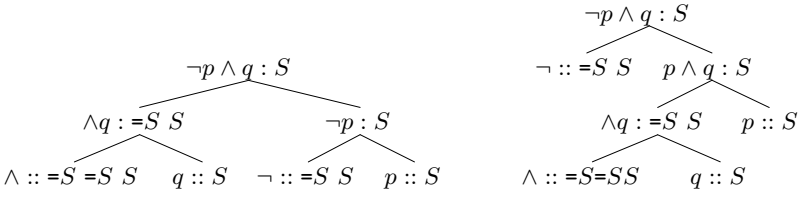
$$\begin{array}{lll} p :: S & q :: S & r :: S \\ \neg :: {=}S\ S & \vee :: {=}S\ {=}S\ S & \wedge :: {=}S\ {=}S\ S \end{array}$$

Grammar $\mathcal{G}3$ has ambiguous expressions, since we have the following two different derivations of $\neg p \wedge q$:

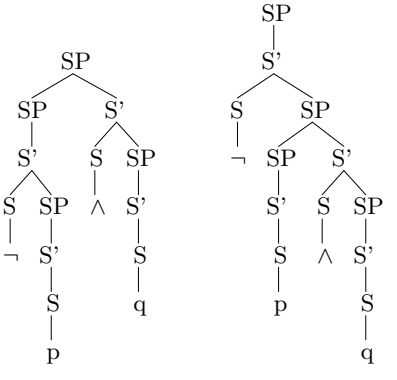
$$merge(merge(\wedge :: {=}S{=}SS, q :: S), merge(\neg :: {=}SS, p :: S))$$

$$merge(\neg :: {=}SS, merge(merge(\wedge :: {=}S{=}SS, q :: S), p :: S))$$

A graphical presentation can be provided which, instead of marking all the internal nodes with the uninformative symbol *merge*, labels them with the values of *merge*:



These derivations correspond to the X-bar structures given below:



While these examples show that $\mathcal{G}3$ has ambiguous expressions, they do not show that $\mathcal{G}3$ has ambiguous yields. Notice that the yields of the two simple derivation trees shown above (not the X-bar structures, but the derivation trees) are not the same. The two yields are, respectively,

$$\wedge :: {=}S{=}SS \quad q :: S \quad \neg :: {=}SS \quad p :: S$$

$$\neg :: {=}SS \quad \wedge :: {=}S{=}SS \quad q :: S \quad p :: S$$

In fact, $\mathcal{G}3$ derivations have unambiguous yields. That is, each sequence of lexical items is the yield of at most one derivation. However, while these sequences of lexical items determine their derivations, the corresponding multisets do not, as can be seen in this example from the fact that exchanging the positions of lexical items p and q, gives two new derivations which, respectively have the same multisets of lexical items as the two derivations shown above. These latter two derivations derive the string $\neg q \wedge p$.

The suggestion is that extra information contained in the leaves – thrown away when the structure building function *merge* is evaluated – is enough to fully determine the derivation. This suggestion is amplified into a general procedure in section 4.

4 Strict Deterministic Grammars for MG Derivations

This section first gives the "natural" CFG for MG derivation tree fringes, adapting some basic ideas from Michaelis (1998). The CFGs obtained this way (subsection 4.1) are not, in general, strict deterministic, but subsection 4.2 shows how they can always be extended so as to become so. The argument then is that, because a strict deterministic grammar for the derivation tree fringe language exists, the language is strict deterministic, hence LR(0), hence uniquely readable.

4.1 The Natural Translation

Perhaps the most natural view of MG derivations as generated by CFGs simply ignores the string-manipulation parts of the structure-building functions. Abbreviate by numerical subscripting i the set of i^{th} projections of each element in a set of n-tuples, $i \leq n$. Then for any MG $\mathcal{G} = \langle \Sigma, B, Lex, \{merge, move\}\rangle$, define

$$R(Lex) \doteq \{Fs \to S :: Fs |\ S :: Fs \in Lex\},$$
$$R(\mathcal{G}) \doteq closure(R(Lex), \{rmerge, rmove\})$$
$$h(\mathcal{G}) \doteq \langle \Sigma', Cat, \to\rangle \text{ where } \Sigma' = Lex, Cat = R(\mathcal{G})_1, \text{ and } \to = R(\mathcal{G}).$$

The functions $rmerge, rmove$ are the obvious modifications of $merge, move$. To obtain the CFG, simply eliminate the string components everywhere except at the leaves. So instead of chains C with string components, the **cchains** $CC \doteq F^*$ are just feature sequences, and the **possible rules** $R \doteq CC^+ \times (Lex \cup CC^+)$. To generate the needed context-free rules, $rmerge : (R \times R) \to R$ is the union of the following 3 functions, for $f \in B$, $\gamma \in F^*$, $\delta \in F^+$, for cchains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$, for non-lexical right sides $N \in CC^+$, and for arbitrary right sides $M, L \in (Lex \cup CC^+)$

$$\frac{{=}f\gamma \to s :: {=}f\gamma \qquad f, \alpha_1, \ldots, \alpha_k \to M}{\gamma, \alpha_1, \ldots, \alpha_k \to {=}f\gamma \quad f, \alpha_1, \ldots, \alpha_k}\ \text{rr1}$$

$$\frac{{=}f\gamma, \alpha_1, \ldots, \alpha_k \to N \qquad f, \iota_1, \ldots, \iota_l \to M}{\gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l \to {=}f\gamma, \alpha_1, \ldots, \alpha_k \quad f, \iota_1, \ldots, \iota_l}\ \text{rr2}$$

$$\frac{{=}f\gamma, \alpha_1, \ldots, \alpha_k \to M \qquad f\delta, \iota_1, \ldots, \iota_l \to L}{\gamma, t : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l \to {=}f\gamma, \alpha_1, \ldots, \alpha_k \quad f\delta, \iota_1, \ldots, \iota_l}\ \text{rr3}$$

Note that since the domains of rr1, rr2, and rr3 are disjoint, their union is a function.

Similarly, $rmove : R \to R$ is the union of the following 2 functions:

$$\frac{+f\gamma, \alpha_1, \ldots, \alpha_{i-1}, -f, \alpha_{i+1}, \ldots, \alpha_k \to N}{\gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k \to +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, -f, \alpha_{i+1}, \ldots, \alpha_k}\ \text{rm1}$$

$$\frac{+f\gamma, \alpha_1, \ldots, \alpha_{i-1}, -f\delta, \alpha_{i+1}, \ldots, \alpha_k \to N}{\gamma, \alpha_1, \ldots, \alpha_{i-1}, \delta, \alpha_{i+1}, \ldots, \alpha_k \to +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, -f\delta, \alpha_{i+1}, \ldots, \alpha_k}\ \text{rm2}$$

Notice several facts about the translation h.

Theorem (Michaelis, Harkema). *$R(\mathcal{G})$ is finite.*

Because of $R(\mathcal{G})$'s finitude, h is well-defined.

Theorem (h-Correctness). *For every MG $\mathcal{G}$, $s \in yield(\Gamma_f(\mathcal{G}))$ if and only if $s \in L_f(h(\mathcal{G}))$.*

Proof idea: this is established with an easy induction on derivation lengths, since the context-free rules rr and rm correspond to every possible application of merge and move.

Theorem 1 (Non-left-recursive). *For every MG $\mathcal{G}$, $h(\mathcal{G})$ is not left recursive.*

Proof idea: This is easy to see, since the label of any left daughter of any node in any derivation is always strictly larger than its parent. In the case of merge, it is one feature larger; in the case of move, it is two features larger.

Examples 6 and 7 show how the range of this translation is not restricted to strict deterministic CFGs. However, subsection 4.2 illustrates another translation g that is restricted in this way.

Example 6 ($h(\mathcal{G}3)$ is not strictly deterministic). Consider grammar $\mathcal{G}3$ from page 170. The CFG $h(\mathcal{G}3) = \langle \Sigma, N, \to \rangle$ where $\Sigma = Lex$, $N = \{S, {=}SS, {=}S{=}SS\}$, and $\to$ is the following 8 pairs:

$S \to {=}SS \quad S$ $\qquad$ $S \to p :: S$ $\qquad$ $S \to q :: S$ $\qquad$ $S \to r :: S$

${=}SS \to {=}S{=}SS \quad S$ $\qquad$ ${=}SS \to \neg :: {=}SS$

${=}S{=}SS \to \wedge :: {=}S{=}SS$ $\qquad$ ${=}S{=}SS \to \vee :: {=}S{=}SS$

Notice that 6 of the 8 pairs are lexical. Also, it is clear that $\Sigma \cup N$ has no strict partition. Notice, in particular, that the category ${=}SS$ can rewrite as a lexical item or as a pair of nonterminals.

Example 7 ($h(\mathcal{G}3)a \equiv h(\mathcal{G}3)$ and $h(\mathcal{G}3)a$ is strictly deterministic). Let $h(\mathcal{G}3)a \doteq \langle \Sigma, N, \to \rangle$ where $\Sigma = Lex$, $N = \{1_{{=}SS}, 2_{{=}SS}, 1_{{=}S{=}SS}, S, {=}SS, {=}S{=}SS\}$, and $\to$ is the following 11 pairs:

$S \rightarrow {=}SS \quad S$ $\qquad$ $S \rightarrow 2_{=SS}$

$2_{=SS} \rightarrow 1_{=SS}$ $\qquad$ $1_{=SS} \rightarrow p :: S$ $\qquad$ $1_{=SS} \rightarrow q :: S$ $\qquad$ $1_{=SS} \rightarrow r :: S$

${=}SS \rightarrow {=}S{=}SS \quad S$ $\qquad$ ${=}SS \rightarrow 1_{=S=SS}$

$1_{=S=SS} \rightarrow \neg :: {=}SS$

${=}S{=}SS \rightarrow \wedge :: {=}S{=}SS$ $\qquad$ ${=}S{=}SS \rightarrow \vee :: {=}S{=}SS$

Clearly $L_S(h(\mathcal{G}3)) = L_S(h(\mathcal{G}3)a)$, but now $\Sigma \cup N$ has a strict partition:

$$\{\{S\}, \{{=}SS, 2_{=SS}\}, \{{=}S{=}SS, 1_{=S=SS}, 1_{=SS}\}, \Sigma\}.$$

4.2 Extending the Natural Translation

The step from $h(\mathcal{G}3)$ to $h(\mathcal{G}3)a$ can be generalized to show that, for any MG $\mathcal{G}$ the grammar $h(\mathcal{G})$ generates an LR(0) language. This is shown using another language-preserving map g on the grammars $h(\mathcal{G})$ whose range includes only strict deterministic CFGs.

Given $h(\mathcal{G}) = \langle \Sigma, N, \rightarrow \rangle$, for any $A \in N$, sequence $(A_1 A_2 \ldots A_n) \in N^+$ is a **left branch** of A if and only if the following three conditions hold:

1. $A = A_1$
2. for all $1 \leq i < n$, either $A_i \rightarrow A_{i+1}$ or $A_i \rightarrow A_{i+1}B$ for some $B \in N$, and
3. $A_n \rightarrow a$ for some $a \in \Sigma$

When $A \rightarrow a$ for $a \in \Sigma$, the empty sequence ϵ is a left branch of A.

As observed earlier (theorem 1), for any MG $\mathcal{G}$ $h(\mathcal{G})$ is never left recursive. Since there are no infinite left-recursive branches, one can define a ranking function $rank : (N \cup Lex) \rightarrow \mathbb{N}$ so that for $a \in \Sigma$, $rank(a) = 0$ and for $A \in N$, $rank(A)$ is the length of the longest left branch of A.

Example 8. In $h(\mathcal{G}3)$, notice that $rank(S) = 3$, $rank({=}SS) = 2$, and $rank({=}S{=}SS) = 1$.

Now g can be defined directly on MGs using h. Observe that all rules $p \in h(\mathcal{G})$ are of the form $p = A \rightarrow a$ or $p = A \rightarrow BC$ or $p = A \rightarrow B$. Define a function *pad* by cases that maps each such rule to a set of pairs as follows:

$p = A \rightarrow a$**:**

If $rank(A) = 1$, then $pad(p) = \{p\}$. Else we add padded categories from $rank(A)$ down to 1 as follows:

$$pad(p) = \{A \rightarrow (rank(A){-}1)_A, (rank(A){-}1)_A \rightarrow (rank(A){-}2)_A, \ldots, 1_A \rightarrow a\}.$$

$p = A \rightarrow BC$**:**

If $rank(B) = rank(A){-}1$, then $pad(p) = \{p\}$. Else we add padded categories from $rank(A)$ down to $rank(B)$ as follows:

$$pad(p) = \{\ A \to (rank(A) - 1)_B C, (rank(A) - 1)_B \to (rank(A) - 2)_B, \\ \ldots, rank(B)_B \to B\}.$$

$p = A \to B$**:** $pad(p)$ is defined similarly.

The new rule set $R'(\mathcal{G})$ is defined to be $\bigcup_{r \in (\to)} pad(r)$, and for any MG $\mathcal{G}$, the context-free grammar $g(\mathcal{G}) \doteq \langle \Sigma, R'(\mathcal{G})_1, R'(\mathcal{G}) \rangle$.

Theorem (g-Correctness). *For every MG $\mathcal{G}$, $h(\mathcal{G}) = \langle \Sigma, N, \to \rangle$ and $g(\mathcal{G}) = \langle \Sigma, N', \to' \rangle$, and any $A \in N$, $L_A(h(\mathcal{G})) = L_A(g(\mathcal{G}))$.*

Proof: From the definition of g, we have immediately that $N \subseteq N'$ and a simple induction shows, using the definition of *pad*, that the same sets of terminal strings are derivable. □

The ranking of nonterminals by their longest left branch induces a partition. For each $x \in (N \cup \Sigma)$, $[x] \doteq \{y \in (N \cup \Sigma) |\ rank(y) = rank(x)\}$. Since there is a unique maximum left branch length for every nonterminal, the $[x]$ are disjoint and since every nonterminal has a maximum left branch length, the $[x]$ are exhaustive.

Theorem 2 (Strict Determinism of MG derivation languages). *For any MG $\mathcal{G}$, $g(\mathcal{G}) = \langle \Sigma, N, \to \rangle$, the set*

$$\pi = \{[x] |\ x \in (N \cup \Sigma)\}$$

is a strict partition of $N \cup \Sigma$.

Proof. Referring again to the definition of strict partitions (2.3) on page 165, each condition is satisfied by construction:

1. $\Sigma \in \pi$ since by the definition of *rank*, $\Sigma = [a]$ for all $a \in \Sigma$.
2. For $\alpha = \epsilon$, it follows from the definition of *pad* that rules with equivalent left sides have equivalent first symbols on their right sides.
   And there are no two different β, β' and two A, A' such that for some non-empty α, $A \to \alpha\beta$ and $A' \to \alpha\beta'$. □

5 Conclusion

In showing that the derivation tree fringes of MGs are strict deterministic, it has been important to keep in mind the difference between having (un)ambiguous *expressions* – i.e. that the string languages are unambiguous, which is certainly false for MGs – and having (un)ambiguous *yields* – i.e. that the language of lexical entry sequences is unambiguous. This latter point is the one demonstrated in this paper. Because strict determinism implies LR(0), shift-reduce automata can quickly assemble a sequence of lexical entries into a tree, dependency graph or other representation, after chart-parsing or some other method has disambiguated a grammatical string into the correct sequence of lexical entries.

References

Chomsky, Noam. 1995. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.

Ebbinghaus, Heinz-Dieter, Jörg Flum, and Wolfgang Thomas. 1994. *Mathematical Logic*. Springer-Verlag.

Enderton, Herbert B. 2001. *A Mathematical Introduction to Logic*. Harcourt.

Harkema, Henk. 2001a. A characterization of minimalist languages. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'01*, Port-aux-Rocs, Le Croisic, France.

Harkema, Henk. 2001b. *Parsing Minimalist Grammars*. Ph.D. thesis, UCLA.

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachusetts.

Harrison, Michael A. and Ivan M. Havel. 1973. Strict deterministic grammars. *Journal of Computer and System Sciences*, 7:237–277.

Keenan, Edward L. and Edward P. Stabler. 2003. *Bare Grammar: Lectures on Linguistic Invariants*. Stanford Monographs in Linguistics. CSLI Publications.

Knuth, Donald. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'98*, Grenoble.

Michaelis, Jens. 2001a. *On Formal Properties of Minimalist Grammars*. Ph.D. thesis, Potsdam University.

Michaelis, Jens. 2001b. Transforming linear context free rewriting systems into minimalist grammars. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'01*, Le Croisic, France.

Shoenfield, Joseph R. 1967. *Mathematical Logic*. Addison-Wesley.

Stabler, Edward and Edward Keenan. 2003. Structural similarity. *Theoretical Computer Science*, 293:345–363.

Stabler, Edward P. 1997. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 68–95. Springer.