1 Introduction

Minimalist Grammars (MGs) (Stabler, 1997) are a formalisation of Chomsky’s minimalist program (Chomsky, 1995), which currently dominates much of mainstream syntax. MGs are simple and intuitive to work with, and are mildly context sensitive (Michaelis, 1998), putting them in the right general class for human language (Joshi, 1985).¹ Minimalist Grammars are known to be more succinct than their Multiple Context-Free equivalents (Stabler, 2013), to have regular derivation tree languages (Kobele et al., 2007), and to be recognisable in polynomial time (Harkema, 2001) with a bottom-up CKY-like parser. However, the polynomial is large, \( O(n^{4k+4}) \) where \( k \) is a grammar constant. By approaching minimalist grammars from the perspective of Interpreted Regular Tree Grammars, we show that standard chart-based parsing is substantially computationally cheaper than previously thought at \( O(n^{2k+3} \cdot 2^k) \).

1.1 Notation

We treat functions as sets of pairs. For \( \langle a, b \rangle \in f \) we write \( a \mapsto b \). For a partial function \( f : A \leadsto B \), the domain of \( f \), \( \text{Dom}(f) = \{ a \in A \mid f(a) \text{ is defined} \} \). The set of all such functions is \([A \leadsto B]\).

For partial functions \( f, g : A \leadsto B \), let \( f \oplus g = f \cup g \) if \( \text{Dom}(f) \cap \text{Dom}(g) = \emptyset \), and undefined otherwise. For \( a \in A \), let \( f - a = f - \{(a, f(a))\} \).

2 Minimalist Grammars

We begin with a brief overview of Minimalist Grammars. Readers familiar with MGs should note that we encode movers with a partial function from licensing features to movers, otherwise

---

¹Or slightly below, if unbounded phrasal copying is required: see for example (Kobele, 2006) on Yoruba clefting.
different happens.

The lexical item ⟨who, D−wh⟩ has a D like Loki, but also has a ¬wh feature which can ultimately cause it to be pronounced in a different place in the string than it would have if it had had only feature D; this is movement. For instance, if instead we applied Merge to laughed and who, deleting the head features leaves ¬wh on who. This means that although who is selected by laughed as an argument, its final position in the string will be determined by something else: the operation licensed by its ¬wh feature. Because the final position is at this point unknown, instead of trying to add it to the string laughed, we store it for later insertion.

Our storage mechanism is a partial function from features to moving items. When laughed selects who and the D features are deleted, ⟨who, ✈⟩ is stored as the image of feature ✈, as follows:

\[
\text{merge} \\
⟨\langle \text{laughed, +DV} \rangle, \emptyset⟩ \langle \text{who, D−wh} ⟩
\]

![Figure 2: Derivation of \(\langle \langle \text{laughed, V} \rangle, \{\text{wh} \mapsto \langle \text{who}, \epsilon⟩\} \rangle\)](image)

We call the partial function the storage and the ⟨string, feature stack⟩ pair the workspace. Together they form an expression. Moving items, or movers, are taken out of storage when their head feature matches the head feature of the workspace. For example, suppose our expression ⟨⟨laughed, V⟩, {wh→⟨who,ε⟩}⟩ is selected by a silent complementiser that triggers wh-movement, ⟨ε, =V +wh C⟩. The string parts will remain unchanged, and the storage untouched, but the head feature in the workspace becomes +wh.

\[
\text{merge} \\
⟨\langle \text{laughed, +DV} \rangle, \emptyset⟩ \langle \text{who, D−wh} ⟩
\]

![Figure 3: Derivation of \(\langle \langle \text{laughed, +wh C} \rangle, \{\text{wh}→\langle \text{who},\epsilon⟩\} \rangle\)](image)

The +wh feature triggers Move. We look in storage for wh, find ⟨who, ε⟩, delete the +wh feature, and concatenate who with laughed:

\[
\text{move} \\
\text{merge} \\
⟨ε, =V +wh C⟩ \langle\langle \text{laughed, +DV} \rangle, \emptyset⟩ \langle\langle \text{who, D−wh} \rangle \rangle
\]

![Figure 4: Derivation of \(\langle \langle \text{who laughed, C} \rangle, \emptyset⟩\)](image)

The item ⟨who, ✈⟩ had only its ¬wh feature left, as represented by its place in storage and the ✈ where its features had been. If it had features left, it would go back into storage after Move, as the image of its new head feature. For example, if who also had nominative case – ⟨who, −nom−wh⟩ –, after moving for case it would go back into storage under ✈. Such a derivation would also require a locus of case assignment; we add in a silent Tense head, ⟨ε, =V+nomT⟩. We illustrate this derivation in Figure 5 with a derivation tree annotated by the expression generated at each step.

\[
\text{move} \\
\text{merge} \\
⟨ε, =V+nomT⟩ \langle\langle \text{laughed, V} \rangle, \{\text{nom}→\langle \text{who}, \epsilon⟩\} \rangle \langle\langle \text{who, D−wh} \rangle \rangle
\]

![Figure 5: Annotated derivation tree of non-final Move: \(\langle \text{who, D−wh} \rangle\) has a feature remaining, so after Move applies it goes back into storage.](image)

Intuitively, it is as though who started out beside laughed because it is an argument of the verb. But because it needed Case, it moved up to beside the Tense. Next, because it is a ✈-word, it will move up to the front of the sentence.

### 2.1 Formal definition

Formally, following Stabler and Keenan (2003), we define a (string-generating) Minimalist Grammar over expressions, with two finite, disjoint sets of bare features. Selectional features sel, drive Merge, and licensing features, lic drive Move. Each of sel and lic has a positive and negative polarity. A feature pairs a polarity with a bare feature. Merge and Move apply when head features of two items have the same bare feature,
but with opposite polarities. The features are $F=\{+f,-f,=X,X | f \in \text{lic}, X \in \text{sel}\}$. Let $\Sigma$ be a finite alphabet. The lexicon $\text{Lex} \subset \Sigma^* \times F^*$ is a finite set of string-feature stack pairs.

An expression is a string-feature stack pair, paired with a partial function from licensing features to string-feature stack pairs; that is, expressions are $\text{Expr} = (\Sigma^* \times F^*) \times [\text{lic} \rightarrow \Sigma^* \times F^*]$. MGs have one constraint: for a given negative feature $f$, only one pair whose head feature is $f$ may be in storage. This is the shortest move constraint (SMC), and we implement it by defining storage as a partial function from lic to (string, feature stack) pairs, and by defining storage parts of merge and move with $\oplus$ as defined in section 1.1 above.

We define four partial functions, $\text{merge}_1, \text{merge}_2, \text{move}_1$, and $\text{move}_2$, as follows.\(^2\) They have $\oplus$ as subfunctions, and are only defined if their subfunctions are defined.

**Merge** $\text{merge}_1, \text{merge}_2 : \text{Expr} \times \text{Expr} \leadsto \text{Expr}$

Let $\alpha, \beta \in F^*$, let $x \in \text{sel}$, and let $f \in \text{lic}$. $\text{merge}_1(\langle \langle s,=X\alpha \rangle, S \rangle, \langle \langle t, X \rangle, T \rangle) = \langle \langle s,t,\alpha \rangle, S \oplus T \rangle$

$\text{merge}_2(\langle \langle s,=X\alpha \rangle, S \rangle, \langle \langle t, X-f \beta \rangle, S \rangle) = \langle \langle s,\alpha \rangle, \{ t, f \rightarrow \langle t, \beta \rangle \} \oplus S \oplus S \rangle$

**Move** $\text{move}_1, \text{move}_2 : \text{Expr} \leadsto \text{Expr}$

Let $\alpha, \beta, \gamma \in F^*$, let $f, g \in \text{lic}$, and suppose $S(f) = \langle t, \beta \rangle$.

$\text{move}_1(\langle \langle s, +f\alpha \rangle, S \rangle = \langle \langle t, s, \alpha \rangle, S - f \rangle$ if $\beta = \epsilon$

$\text{move}_2(\langle \langle s, +f\alpha \rangle, S \rangle = \langle \langle s, \alpha \rangle, \{ g \rightarrow \langle t, \gamma \rangle \} \oplus (S - f) \rangle$ if $\beta = -g^\gamma$

For example, in the derivation in Figure 5, the lowest merge node is an instance of $\text{merge}_2$. merge applies because the head feature of laughed is $=X$ and that of who is $D$. It is $\text{merge}_2$ specifically because, in the language of the definition above, $\beta = -\text{nom}-\text{wh}$. The next merge node is $\text{merge}_1$, because the feature stack of laughed is just $V$. The move node is an instance of $\text{move}_2$ since $\beta = -\text{wh} \neq \epsilon$.

An MG is a 6-tuple

$$g = (\Sigma, \text{sel}, \text{lic}, M, \text{Lex}, S)$$

where $\Sigma$ is a finite alphabet.

\(^2\)The domains of $\text{Merge}$ and $\text{Move}$ and those of $\text{Move}_1$ and $\text{Move}_2$ being disjoint, the operations can alternatively be defined as just $\text{Merge}$ and $\text{Move}$ with 2 cases each. We choose this variant for parallelism with the minimalist string algebra defined in section 2.3 below.

\(^3\)More precisely, the licensing features are given an order, and the MCFG category names, rather than having a partial function from licensing features to feature stacks just has the feature stacks in the right order, and similarly for the order of elements in the tuples.
2.3 Minimalist String Algebra

Kobele et al. (2007) define an algebra of tree tuples, which handles how the Minimalist Grammar builds trees. We define a similar algebra which builds strings, and convert the algebra into our notation of a partial function from licensing features to strings. These functions are just the string parts of the MG operations, separated out from the feature calculus.

The values of the algebra are strings paired with a partial function from lic to strings, i.e.
\[ \Sigma^* \times [\text{lic} \rightsquigarrow \Sigma^*], \]
which we call \textit{minimalist string tuples}. We define \( |\text{lic}| + 1 \) Merge operations and \( |\text{lic}|^2 + |\text{lic}| + 1 \) Move operations as follows.

\[ \forall \epsilon, g \in \text{lic}. \merge_1 \text{ and } \move_1 \text{ are for final merge/move, so the string of the merging or moving element concatenates (') with the main string, on the right for Merge and on the left for Move.} \]

\[ \merge_{2\epsilon} \text{ is for } \epsilon\text{-storing Merge, and } \move_{2\epsilon+} \text{ is for } \epsilon\text{-storing Move triggered by } g. \]

For an MG \( \langle \Sigma, \text{sel } \cup \text{lic}, M, \text{Lex} \rangle \), the signature of a tuple-feature algebra includes each element \( s^{(0)} \) of the alphabet \( \Sigma \) (evaluates to \( \langle s, \emptyset \rangle \)), \( \merge^{(2)}_1 \) (evaluates to \( \merge_1 \)), \( \merge^{(2)}_\epsilon \) for each \( \epsilon \in \text{lic} \) (evaluates to \( \merge_{2\epsilon} \)), \( \move^{(1)}_f \) for each \( \epsilon \in \text{lic} \) (evaluates to \( \move_{1\epsilon} \)), and \( \move^{(1)}_{2\epsilon g} \) for each pair \( \epsilon, g \in \text{lic} \) (evaluates to \( \move_{2\epsilon g} \)).

If \( t = m(d_0, \ldots, d_n) \) is a term of the signature of the algebra, \( t \) evaluates to the function \( m \) evaluates to, applied to what \( d_0, \ldots, d_n \) evaluate to. We write \( [t] = [m][[d_0], \ldots, [d_n]]. \)

3 Interpreted Regular Tree Grammar

Minimalist Grammars lend themselves readily to so-called “two-step” approaches in which the feature calculus is separated from the algebra of the derived forms (strings, trees, etc). For instance, Kobele et al. (2007) show that for a given MG, the language of valid derivation trees is regular, and that a derived tree can be generated by a multi-bottom up transduction from the derivation tree. Graf (2012) adds MSO-definable constraints on the transduction to constrain movement and define different movement types (sideways, lowering, covert, etc).

Michaelis et al. (2000), Morawietz (2003), and Mönnich (2006), etc. take a related approach, generating derived trees by Monadic Second-Order (MSO)-definable transduction not from the derivation tree but rather from the equivalent MCGF, translated into a regular tree grammar. (Kobele et al. 2007 note that this second approach can generate transductions that theirs cannot.)

In this tradition, we define an interpreted regular tree grammar for Minimalist Grammars. IRTGs are a generalisation of, among other things, the synchronous grammars of Shieber (1994), 2004, 2006 that form the basis for the tree homomorphisms of Kobele et al. (2007).

3.1 IRTGs

An \textit{interpreted regular tree grammar (IRTG)} (Koller and Kuhlmann, 2011)

\[ G = \langle G, (h_1, A_1), \ldots, (h_n, A_n) \rangle \]

derives \( n \)-tuples of objects, such as strings or trees from \textit{derivation trees} in \( G \). A given \( t \in G \) is interpreted into the \( n \) algebras \( A_1, \ldots, A_n \) by means of the \( n \) tree homomorphisms \( h_1, \ldots, h_n \). For a given \( i \leq n, h_i(t) \) is a term of the signature of algebra \( A_i \), which is in turn \textit{evaluated} (\( \langle \cdot \rangle_{A_i} \)) into an object of the algebra. For example, suppose we have a minimalist string algebra \( A \) as defined in Section 2.3, and suppose we have a derivation tree as in the first tree in Table 3, call it \( t \). A tree homomorphism \( h \) that includes the rules \( \{mv_1 \mapsto \merge_{\text{inom}}, mg_{13} \mapsto \merge_1, \text{lex}_{11} \mapsto \epsilon, mg_2 \mapsto \merge_{\text{inom}}, \text{lex}_9 \mapsto \text{laughed}, \text{lex}_3 \mapsto \text{Loki} \} \) yields the second tree in Table 3, call it \( u \). Then we write \( h(t) = u \) and \( \langle u \rangle_{A_i} = \langle \langle \text{Loki laughed}, T \rangle, \emptyset \rangle \).

The language of the grammar \( L(G) \) is the set of tuples \( \{ \langle h_1(t) \rangle_{A_1}, \ldots, [h_n(t)]_{A_n} \mid t \in L(G) \} \).

An IRTG is \textit{regular} in that \( G \) is a \textit{regular tree language}, meaning it is a set of trees that can be generated by a finite set of production rules of the form \( NT_0 \rightarrow t \) or \( NT_0 \rightarrow t(NT_1, \ldots, NT_n) \) for nonterminals \( NT_i \) and terminals \( t \). The terminals are elements of the signature of the tree language. An example is given in Table 3.

3.2 IRTG for Minimalist Grammars

We use the regular tree language of derivation trees defined in Kobele et al. (2007) and define a homomorphism from the derivation trees to terms of the minimalist string algebra (with notation
modified to match ours), explicitly defining it as an IRTG. Finally, we calculate the parsing complexity.

For a Minimalist Grammar $g$ with lexicon $Lex$, the production rules of its regular tree grammar, RTG($g$), have as their nonterminal symbols the featural configurations of expressions defined by the grammar. Let $f : \{lic \rightarrow \Sigma^* \times F^*\} \rightarrow \{lic \rightarrow F^*\}$ strip away the string parts of a storage function, leaving only the features. Then the nonterminals of RTG($g$) are $\{\langle fs, f(S)\rangle | \langle s, fs, S \rangle \in CL(Expr(Lex))\}$. Since lexical items have finite feature stacks, the SMC limits the size of the storage, and each application of $merge$ or $move$ deletes features, there are a finite number of nonterminals for a given finite lexicon. Therefore each possible application of $merge$ or $move$ to expressions of $g$ belong to a finite set of instances; these are the non-lexical rules of the RTG. Each lexical item $\langle s, fs \rangle$ has associated with it a rule with left hand side $\langle fs, \emptyset \rangle$. We give each rule a name from $\{mg_i, mv_i, lex_i \mid i \in \mathbb{N}\}$ by choosing a new $i \in \mathbb{N}$ for each rule: in an IRTG, each rule has its own name. The rules are named according to Table 1. The start categories are $\{\langle S, \emptyset \rangle \mid S \in S\}$.

Example 3.1.

Let $sel = \{T, V, D, C\}$, $lic = \{nom, wh\}$. Let $S = \{T, C\}$ be the start categories. Let Lex be defined according to Table 2; for example, $\langle Thor, D-nom\rangle \in Lex$.

Table 3 lists the RTG production rules and contains an example tree and its interpretation in the minimalist string algebra defined in section 2.3 above. Rules that are greyed out are rules that can never be used in a complete derivation; the RTG could also be defined to leave them out.

3.3 Interpretation

The derivation trees – the terms over $\{mg_i^{(2)}, mv_i^{(1)}, lex_i^{(0)} \mid i \in \mathbb{N}\}$ – are interpreted in algebras, meaning for each algebra we want to interpret into, we define a tree homomorphism from derivation trees to terms of the algebra. In our case, we want to interpret into the minimalist string algebra as follows. The examples are from the grammar in Table 3.

Merge 1 Merge of a non-mover is interpreted as $merge_1$, e.g.: $h(mg_i(t_1, t_2)) = merge_1(h(t_1), h(t_2))$ for $i \in \{1, 3, 5, 7, 8, 12, 13, 14, 15, 16, 17\}$.

Move 2 $f$-storing Merge is interpreted as $merge_{2f}$, e.g.: $h(mg_i(t_1, t_2)) = merge_{2nom}(h(t_1), h(t_2))$ for $i \in \{2, 6, 9, 11\}$ or $merge_{2nom}(h(t_1), h(t_2))$ for $i \in \{4, 10\}$.

Move 1 Final move triggered by $f$ is interpreted as $move_{2f}$, e.g.: $h(mv_i(t)) = move_{1nom}(h(t))$ for $i \in \{1, 2\}$.

Move 2 $g$-storing Move triggered by $f$ is interpreted as $move_{2g}$, e.g.: $h(mv_i(t)) = move_{2nom-wh}(h(t))$.

Lex for a production rule $\langle fs, \emptyset \rangle \rightarrow lex_i$, each $h(lex_i) = s$ for some $\langle s, fs \rangle \in Lex$. e.g.: $h(lex_i) = h(lex_3) = Loki$.

For example, the derivation tree in Table 3 is interpreted by the homomorphism $h$ as a term of the string-feature tuple algebra, which evaluates to the minimalist string tuple $(\text{Loki laughed}, \emptyset)$.

4 IRTG-based parsing for minimalist grammars

Given a minimalist grammar $g$, we can ask whether a given string $w$ is grammatical according to $g$, i.e. if $w \in L(g)$. This parsing problem has been addressed in a substantial amount of literature (Harkema, 2001), (Stabler, 2013), (Stanojević, 2016). The best known upper bound for a complete parser from this literature is $O(n^{4k+4})$ (Harkema, 2001). This is based on a relatively coarse estimation, by which there are $O(n^{2k+2})$ different parse items, and binary rules such as those for Merge could combine these arbitrarily. Alternatively, by encoding $g$ into an $(k+1)$-MCFG, we can apply standard parsing algorithms for MCFGs, which yields a parsing complexity of $O(n^{3k+3})$ (Seki et al., 1991). The more efficient MCFG parsing algorithm for well-nested MCFGs of Gómez-Rodríguez et al. (2010), which would yield a parsing complexity of $O(n^{2k+4})$, is not applicable because the MCFGs that are produced by the MG-to-MCFG encoding are not well-nested (Boston et al., 2010).

Here we present a parsing algorithm for minimalistic grammars that is based on the MG-to-IRTG encoding. This algorithm has a runtime of $O(n^{2k+3})$, a substantial improvement over previously published upper bounds. It is worth noting that we achieve this improved upper bound not through a particularly clever parsing algorithm – indeed, the basic idea of the algorithm presented here is the same as in Harkema (2001) –, but through a more careful analysis of the algorithm’s...
The primary advantage we obtain from using the standard IRTG parsing algorithm is that it separates the parts that depend on the string length very cleanly from those that depend on the grammar, which makes it a bit easier to see the exact runtime complexity.

We will make a Java implementation of our parsing algorithm available open-source upon publication.

We first sketch the general approach to parsing with IRTGs (Koller and Kuhlmann, 2011). The objective of IRTG parsing is to compute, given an input object $w$ and an IRTG grammar $\mathcal{G} = (G, (h, A))$, a compact representation of the language $\text{pares}(w) = \{ t \in L(G) \mid [h(t)] = w \}$ – i.e., of those derivation trees that are both grammatically correct and that are interpreted to $w$. This is done by first computing a decomposition grammar $D_w$, that is, an RTG such that $L(D_w)$ consists of all terms that evaluate to $w$ in the algebra. Then we can exploit closure properties of regular tree languages to compute a parse chart – that is, an RTG $\mathcal{C}$ such that $L(\mathcal{C}) = L(\mathcal{G}) \cap h^{-1}(L(D_w))$, by intersecting $\mathcal{G}$ with an RTG that generates all trees which $h$ maps to a term in $L(D_w)$. By construction, we have that $L(\mathcal{C}) = \text{pares}(w)$.

Most pieces of this parsing algorithm are completely generic, and do not depend on the algebra that is being used. Thus, when one applies IRTGs to a new algebra, all that is required to obtain a complete parser is to specify how decomposition grammars $D_w$ are computed for arbitrary elements $w$ of the algebra. We now explain how to obtain decomposition grammars for the minimalist string algebra.

### Table 1: RTG(g) rule template

<table>
<thead>
<tr>
<th>Types</th>
<th>strings</th>
<th>feature stacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominals</td>
<td>Loki, Thor</td>
<td>D-nom</td>
</tr>
<tr>
<td>wh-words</td>
<td>who</td>
<td>D-nom-wh</td>
</tr>
<tr>
<td>Intransitive verbs</td>
<td>laughed, cried</td>
<td>=D =V</td>
</tr>
<tr>
<td>Transitive verbs</td>
<td>slew, triked</td>
<td>=D =D V</td>
</tr>
<tr>
<td>Tense</td>
<td>$\epsilon$</td>
<td>=V +nom T</td>
</tr>
<tr>
<td>Complementer</td>
<td>$\epsilon$</td>
<td>=T +wh C</td>
</tr>
</tbody>
</table>

---

#### 4.1 Parsing Complexity

Asymptotic parsing complexity is determined by the time it takes to compute the rules of $D_w$: the rest of the IRTG parsing algorithm is linear in the size of $D_w$. The most costly rule of $D_w$, in terms of parsing complexity, is that for merge. In this rule there are $O(n^4)$ values for the string positions $i, j, p$. Within $S \oplus T$ there are spans for at most $k$ spans, each of which has $O(n^2)$ possible values. These spans are distributed over the two child nonterminals. This can be done in $2^k$ different ways. Thus, in total, there are are $O(n^{2k+3} \cdot 2^k)$ instances of this rule, which can be enumerated asymptotically in that time.

---

Table 2: Sample lexicon

<table>
<thead>
<tr>
<th>Types</th>
<th>strings</th>
<th>feature stacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intransitive verbs</td>
<td>laugh, cried</td>
<td>=D =V</td>
</tr>
<tr>
<td>Transitive verbs</td>
<td>slew, triked</td>
<td>=D =D V</td>
</tr>
<tr>
<td>Tense</td>
<td>$\epsilon$</td>
<td>=V +nom T</td>
</tr>
<tr>
<td>Complementer</td>
<td>$\epsilon$</td>
<td>=T +wh C</td>
</tr>
</tbody>
</table>
ber of movers via the SMC. The maximum number of pieces being manipulated by the movers is frequently come with constants that limit the number of items that can exist at any time. In Minimalist Grammars it is the number of pieces being manipulated by the workspace.

Transforming an MG into an MCFG yields a grammar. In Multiple Context-Free Grammars (MCFGs) (Seki et al., 1991) and Linear Context-Free Rewriting Systems (LCFRSs) (Vijay-Shanker et al., 1987) these are the rank – the maximum number of daughters/arguments a rule can have, and the fanout – the maximum number of elements in a tuple. In Minimalist Grammars it is the number of licensing features k, which limits the number of movers via the SMC. The maximum number of elements in a minimalist item is therefore k + 1 all possible movers plus the workspace.

Transforming an MG into an MCFG yields a grammar with rank 2 and fanout k + 1 (Michaelis, 1998). Our $O(n^2k^3 \cdot 2^k)$ expressed in terms of fanout $f = k + 1$ is therefore $O(n^{2f+1} \cdot 2^{f-1})$, which is less than the parsing complexity for an arbitrary binary MCFG with fanout $f$: $O(n^3f)$.

It is difficult to compare parsing complexities across grammars, as moving from one grammar to another can change the fanout. While MGs, MCFGs, and LCFRSs with finite fanout generate the same languages, an arbitrary binary MCFG of fanout $f$ may not have a weakly equivalent MG with $f - 1$ licensing features; indeed Michaelis (2001) shows that an LCFRS with fanout $f$ has a weakly equivalent MG with $3f$ licensing features.

In terms of the string algebra, the difference between an MCFG and an MG is that an MCFG rule is unrestricted in how it concatenates strings; in an MG, only the workspace can be made by concatenation; the movers are simply pooled into one function, never concatenated with one another. In this sense, MG equivalents of MCFGs are a subclass of general MCFGs of the same fanout, one

### Table 3: Example IRTG rules and an example derivation of *Loki laughed*

<table>
<thead>
<tr>
<th>Lexical</th>
<th>Phrasal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\text{D}, \emptyset]$</td>
<td>merge of subject</td>
</tr>
<tr>
<td>$[\text{D-nom}, \emptyset]$</td>
<td>$m_{\text{g}_1}([\text{D-nom}, \emptyset], [\text{D}, \emptyset])$</td>
</tr>
<tr>
<td>$[\text{D-nom-wh}, \emptyset]$</td>
<td>$m_{\text{g}_2}([\text{D-nom-wh}, \emptyset], [\text{D}, \emptyset])$</td>
</tr>
<tr>
<td>$[\text{D-wh}, \emptyset]$</td>
<td>$m_{\text{g}_3}([\text{D-wh}, \emptyset], [\text{D}, \emptyset])$</td>
</tr>
<tr>
<td>$[\text{D-nom-wh}, \emptyset]$</td>
<td>$m_{\text{g}_4}([\text{D-nom-wh}, \emptyset], [\text{D}, \emptyset])$</td>
</tr>
</tbody>
</table>

### Figure 6: Decomposition rules

$[(i, p), S + T] \rightarrow \text{merge}_1([(i, j), S], [(j, p), T])$

$[(i, j), S + T + \{x \rightarrow (p, l)\}] \rightarrow \text{merge}_2(x, [(i, j), S], [(p, l), T])$

$[(i, j), S - x] \rightarrow \text{move}_1([(i, j), S])$

$[(i, j), (S - x) + \{g \rightarrow S(x)\}] \rightarrow \text{move}_2([(i, j), S])$

### 5 Comparison with other Mildly Context Sensitive grammars

Mildly context sensitive grammars (Joshi, 1985) frequently come with constants that limit the number of pieces being manipulated by the grammar. In Multiple Context-Free Grammars (MCFGs) (Seki et al., 1991) and Linear Context-Free Rewriting Systems (LCFRSs) (Vijay-Shanker et al., 1987) these are the rank – the maximum number of daughters/arguments a rule can have, and the fanout – the maximum number of elements in a tuple. In Minimalist Grammars it is the number of licensing features k, which limits the number of movers via the SMC. The maximum number of elements in a minimalist item is therefore k + 1 all possible movers plus the workspace. Transforming an MG into an MCFG yields a grammar with rank 2 and fanout k + 1 (Michaelis, 1998). Our $O(n^2k^3 \cdot 2^k)$ expressed in terms of fanout $f = k + 1$ is therefore $O(n^{2f+1} \cdot 2^{f-1})$, which is less than the parsing complexity for an arbitrary binary MCFG with fanout $f$: $O(n^3f)$.
which has a lower parsing complexity. To transform an MG into an MCFG we take as categories the RTG categories, choose an (arbitrary) order on the licensing features, and interpret the mover storage partial function as tuples in the chosen order. We call the class of MCFGs with string concatenation rules restricted to the rules of the Minimalist String Algebra MCFG\textsubscript{\textit{m}}.

Another subclass of MCFGs with lowered parsing complexity is well-nested (Kuhlmann, 2007) MCFGs (MCFG\textsubscript{\textit{wn}}) in which no rule involves the interleaving of elements from two daughters (no \textit{abab} rules). The parsing complexity of a binary MCFG\textsubscript{\textit{wn}} with fanout \( f \) is \( O(n^{2f+2}) \), due to the fact that there is a normal form in which all deduction rules are either concatenation rules or wrapping rules, which have complexity \( O(n^{2f+1}) \) and \( O(n^{2f+2}) \) respectively (Gómez-Rodríguez et al., 2010). In a concatenation rule, we take one element of each tuple and concatenate them, and the rest are kept as they are; in a well-nested MCFG the last element of the first daughter is concatenated with the first element of the second daughter, which maintains the well-nestedness.

Interestingly, although the MCFG equivalent of MGs is not well-nested, the argument for the parsing complexity of \textit{merge} is closely related to that for MCFG\textsubscript{\textit{wn}}. The well-nested concatenation rules have the same number of indices as \textit{merge}. Therefore the complexity of \textit{merge} \( O(n^{2f+1} \cdot 2^f) \) and concatenation rules for parsing a MCFG\textsubscript{\textit{wn}} \( O(n^{2f+1}) \) have the same polynomial degree, \( 2f+1 \). This is perhaps counter-intuitive, since well-nested MCFGs are a proper subset of MCFLs/MLs (Gómez-Rodríguez et al., 2010). However, as noted above, transforming between grammars will often change the fanout.

A proper subclass of well-nested MCFGs is monadic-branching MCFGs (MCFG\textsubscript{\textit{mb}}), which are binary MCFGs in which only the right daughter may have fanout greater than 1. MGs with the specifier island condition (SpIC), in which nothing can move out of a specifier, are weakly equivalent to monadic-branching MCFGs (Kanazawa et al., 2011). These grammars have three kinds of Merge rules: \textit{merge} \( 1 \), which merges a lexical item with its complement; \textit{merge} \( 2 \), which merges a non-lexical item with its specifier, and \textit{merge} \( 3 \), which merges a mover. Move is restricted to prevent a certain kind of movement from within a mover, and \textit{merge} is restricted to prevent movement from within a specifier. The result is grammar that never has to combine mover lists. \textit{merge} \( 1 \) can’t have movers in the selector, since lexical items never carry movers, and \textit{merge} \( 2 \) is constrained by the SpIC not to have movers in the selected item. Our string-tuple analysis of minimalist parsing makes it clear that SpIC-MGs have a parsing complexity of \( O(n^{2k+3}) \). The most complex rules are \textit{merge} \( 1 \) and \textit{merge} \( 2 \), which still have at most 3 indices for the workspace and 2 for each mover. The only difference is that in the standard MG case, the movers could have come from either daughter, but for a SpIC-MG they could only have come from one daughter. For SpIC-MGs the parsing complexity is therefore reduced to \( O(n^{2k+3}) \). For our parser the difference is not necessarily huge since \( 2^k \) is a constant, but for some, like Stabler (2013)’s top-down beam parser, the SpIC can greatly reduce the search space.

Figure 7 shows the grammars described above.\(^4\) We don’t have a linguistic characterization of the “\( ? \)”-node, which stands for the intersection between the two higher nodes. These would be well-nested MCFGs that only have concatenation in the first element of the tuple. We speculate that this is a linguistically uninteresting class, as the non-well-nestedness of the rules is a reflection of the arbitrarily-chosen order on the licensing features, and has no special linguistic significance.\(^5\)

6 Conclusion

Approaching Minimalist Grammars as interpreted regular tree grammars makes clear the parsing complexity of traditional chart-based parsing, and the options available for interpretation of a derivation as a string. We found that the commonly-cited upper bound of \( O(n^{4k+1}) \) was in fact too conservative, and MGs can be parsed in the much smaller polynomial time of \( O(n^{2k+3} \cdot 2^k) \). MGs with the specifier island constraint have a parsing complexity of \( O(n^{2k+3}) \).

\(^4\) Note that the inclusion refers to the string algebra restrictions in the grammars themselves, and not necessarily to the languages they generate. The left side of the diagram in fact is reflected in the languages – for a given fanout and rank, MCFL\textsubscript{\textit{mb}} \( \subseteq \) MCFL\textsubscript{\textit{wn}} \( \subseteq \) MCFL. We don’t make any claims about the weak generative capacity on the right side.

\(^5\) Also missing from the lattice is the class of MGs with a looser SpIC where only Move is restricted by the SpIC. This restriction leaves the asymptotic parsing complexity unchanged as Merge is still the most complex rule and is unchanged.
Figure 7: Subclasses of binary MCFGs with fanout $f$ and their parsing complexities

References


Frank Morawietz. 2003. Two-Step Approaches to Natural Language Formalism, volume 64. Walter de Gruyter.


