# Parsing Minimalist Languages with Interpreted Regular Tree Grammars

**Meaghan Fowlie**
Saarland University
mfowlie@coli.uni-saarland.de

**Alexander Koller**
Saarland University
koller@coli.uni-saarland.de

## 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, $\mathcal{O}(n^{4k+4})$ 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 $\mathcal{O}(n^{2k+3} \cdot 2^k)$.

### 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 $f : A \rightsquigarrow B$, the domain of $f$, $Dom(f) = \{a \in A \mid f(a) \text{ is defined}\}$. The set of all such functions is $[A \rightsquigarrow B]$.

For partial functions $f, g : A \rightsquigarrow B$, let $f \oplus g = f \cup g$ if $Dom(f) \cap Dom(g) = \emptyset$, 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 Section 3 should be familiar. Minimalist Grammars are a family of formal grammars in which parts of sentences are put together with operations *merge* and *move*. MGs are feature-driven, which means that lexical items come with a stack of features which determine whether operations apply.

Features encode properties of lexical items, such as syntactic categories (noun, verb, etc), categories of arguments of the word (such as a verb that selects a noun), as well as agreement or displacement, such as person, case, quantifier raising, or wh-movement. For example, lexical item $\langle \text{Loki}, \texttt{D}\rangle$ has string part *Loki* and feature stack `D`, meaning it has category `D`, while $\langle \text{laughed}, \texttt{=DV}\rangle$ has features `=D` and `V`, meaning that it requires something of category `D` and is itself of category `V`.

These requirements of the lexical items are fulfilled by applications of Merge and Move operations. For example, *laughed*'s requirement of a `D` is fulfilled by Merging it with *Loki* or *who*, for example:

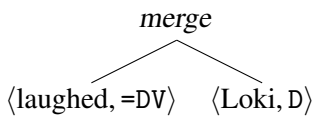



Figure 1: Derivation tree of $\langle \text{laughed Loki}, \texttt{V}\rangle$

The diagram in 1 above illustrates a ***derivation tree***, a term over $\langle merge^{(2)}, move^{(1)}, Lex^{(0)}\rangle$ which describes the application of *merge* and *move* to expressions.

An operation's applicability is determined by the features on the top of the feature stacks – the ***head features*** – and the application deletes those features; here the `=D` and `D` features are deleted, leaving us with `V`. Notice that deleting `D` left *Loki* without features; when features remain, something

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



*Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13)*, pages 11–20,
Umeå, Sweden, September 4–6, 2017. 

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, ϵ⟩ is stored as the image of feature wh, as follows:

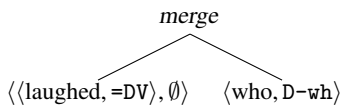



Figure 2: Derivation of ⟨⟨laughed, V⟩, {wh↦⟨who, ϵ⟩ }⟩

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.

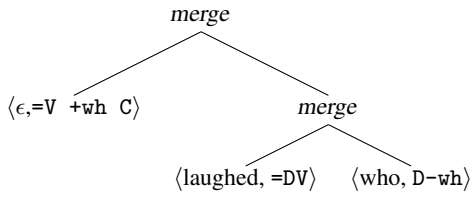



Figure 3: Derivation of ⟨⟨laughed, +wh C⟩, {wh↦⟨who,ϵ⟩ }⟩

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

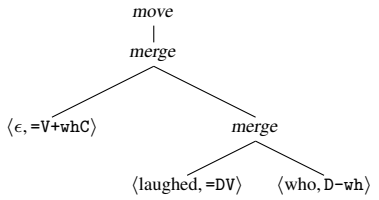



Figure 4: Derivation of ⟨⟨who laughed, C⟩,∅⟩

The item ⟨who, ϵ⟩ had only its -wh feature left, as represented by its place in storage and the ϵ 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 wh. 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.

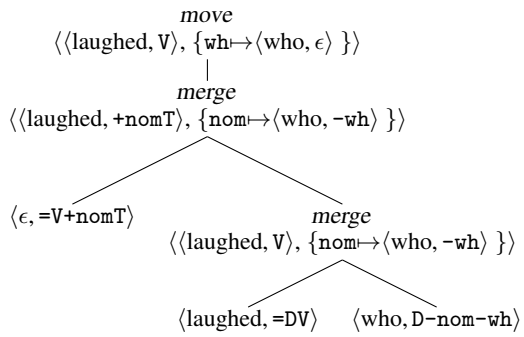



Figure 5: Annotated derivation tree of non-final Move: ⟨who, -wh⟩ has a feature remaining, so after Move applies it goes back into storage.

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 *wh*-word, it will move up to the front of the sentence.

### 2.1 Formal defintion

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∈ *lic*, X ∈ *sel* }. Let $\Sigma$ be a finite alphabet. The *lexicon* *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 Expr = $(\Sigma^* \times F^*) \times [lic \rightsquigarrow \Sigma^* \times F^*]$

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, $merge_1, merge_2, move_1$, and $move_2$, as follows.[2] They have $\oplus$ as subfunctions, and are only defined if their subfunctions are defined.

**Merge** $merge_1, merge_2$ : Expr × Expr $\rightsquigarrow$ Expr

Let $\alpha, \beta \in F^*$, let X∈ *sel*, and let f∈ *lic*.

$$merge_1(\langle(s, \texttt{=X}\alpha), \mathsf{S}\rangle, \langle(t, \texttt{X}), \mathsf{T}\rangle) = \langle\langle st, \alpha\rangle, \mathsf{S} \oplus \mathsf{T}\rangle$$

$$merge_2(\langle\langle s\texttt{=X}\alpha\rangle, \mathsf{S}_s\rangle, \langle\langle t, \texttt{X-f}\beta\rangle, \mathsf{S}_t\rangle) = \langle\langle s, \alpha\rangle, \{\texttt{f} \mapsto \langle t, \beta\rangle\} \oplus \mathsf{S}_s \oplus \mathsf{S}_t\rangle$$

**Move** $move_1, move_2$ : Expr $\rightsquigarrow$ Expr

Let $\alpha, \beta, \gamma \in F^*$, let f,g ∈ *lic*, and suppose S(f) = ⟨t, $\beta$⟩.

$$move_1\langle\langle s, \texttt{+f}\alpha\rangle, \mathsf{S}\rangle = \langle\langle ts, \alpha\rangle, \mathsf{S} - \texttt{f}\rangle \quad \text{if } \beta = \epsilon$$

$move_2$(⟨⟨s, +f$\alpha$⟩, S⟩)=
⟨⟨s, $\alpha$⟩, {g↦⟨t, $\gamma$⟩ }⊕(S−f)⟩ if $\beta$=-g$\gamma$

For example, in the derivation in Figure 5, the lowest *merge* node is an instance of $merge_2$. *merge* applies because the head feature of *laughed* is =X and that of *who* is D. It is $merge_2$ specifically because, in the language of the definition above, $\beta$ = -nom-wh. The next *merge* node is $merge_1$ because the feature stack of *laughed* is just V. The *move* node is an instance of $move_2$ since $\beta$ = -wh ≠ $\epsilon$.

An MG is a 6-tuple

$$g = \langle \Sigma, sel, lic, M, Lex, S \rangle$$

where $\Sigma$ is a finite alphabet,

$M = \{merge_1, merge_2, move_1, move_2\}$,
*Lex* $\subset \Sigma^* \times$ {+f,-f,=X,X | f∈*lic*,X∈*sel*}$^*$, and $S \subseteq$ *sel* is a designated set of start categories. From our two examples above, we can define an MG $g$ where $\Sigma$ = {Loki, laughed, who}, *sel* ={D,V,T,C}, *lic* ={nom,wh}, *Lex*={⟨who, D-wh⟩, ⟨who, D-nom-wh⟩, ⟨Loki, D⟩, ⟨laughed, =D V⟩, ⟨$\epsilon$, =V +wh C⟩, ⟨$\epsilon$, =V +nom T⟩, ⟨$\epsilon$, =T +wh C⟩ }, and $S$={T,C}.

Let Expr(⟨s, fs⟩) = ⟨⟨s, fs⟩, $\emptyset$⟩ make an expression with empty storage from a lexical item. $CL(Lex, M)$, the ***closure*** of the lexicon under the operations $M$, is the closure of Expr(*Lex*) under the operations. Then for a Minimalist Grammar $g$ with lexicon *Lex* and start categories $S \subseteq$ *sel*, the ***language*** generated by $g$ is $L(g) = \{s \mid \langle\langle s, \mathsf{S}\rangle, \emptyset\rangle \in CL(Lex, M)$ and $\mathsf{S} \in S\}$; that is, for an expression with empty storage and exactly one feature, where that feature is a start category, the string part of that expression is in the language of $g$. In our example, $L(g)$ = {who laughed, Loki laughed}.

If *merge* applies because the head feature of the first expression is =X, we say that application of *merge* is ***triggered by*** X. Similarly, if *move* applies because the head feature of the expression is +f, we say the application of *move* is triggered by f. $merge_2$ and $move_2$ always add a mover to storage; if that mover is the image of feature f,then we say it is an ***f-storing operation***.

### 2.2 Expressive capacity

MGs are mildly context sensitive; in particular they are are weakly equivalent to multiple context free grammars (MCFGs) and Linear Context-Free Rewrite Systems (LCFRSs) (Michaelis, 1998), (Michaelis, 2001), and multicomponent tree-adjoining grammars (MC-TAGs), which are more expressive than TAGs. Every MG with $k$ licensing features has a strongly equivalent ($k$+1)-MCFG(2) – an MCFG with at most binary rules and at most $k+1$-tuples –, where the category names are the features of an expression and the strings behave very much like the string tuple algebra in section 2.3 below.[3] An MG can be exponentially more succinct than its equivalent MCFG (Stabler, 2013); similarly the IRTGs defined here can be ex-

[2] The domains of Merge 1 and 2, and those of Move 1 and 2 being disjoint, the operations can alternatively be defined as just Merge and Move with 2 cases each. We choose this variant for parallelism with the minimalist string algebra defined in section 2.3 below.

[3] More preceisely, 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.

ponentially larger than the MGs they describe.

### 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. $\Sigma^* \times [lic \rightsquigarrow \Sigma^*]$, which we call *minimalist string tuples*. We define $|lic| + 1$ Merge operations and $|lic|^2 + |lic| + 1$ Move operations as follows, $\forall \mathtt{f},\mathtt{g} \in lic$. $merge_1$ and $move_1$ are for final merge/move, so the string of the merging or moving element concatenates ($\cdot$) with the main string, on the right for Merge and on the left for Move. $merge_{2\mathtt{f}}$ is for $\mathtt{f}$-storing Merge, and $move_{2\mathtt{g\text{-}f}}$ is for $\mathtt{f}$-storing Move triggered by $\mathtt{g}$.

$$
\begin{aligned}
merge_1(\langle s, \mathsf{S}\rangle, \langle t, \mathsf{T}\rangle) &= \langle s \cdot t, \mathsf{S} \oplus \mathsf{T}\rangle \\
merge_{2\mathtt{f}}(\langle s, \mathsf{S}\rangle, \langle t, \mathsf{T}\rangle) &= \langle s, \mathsf{S} \oplus \mathsf{T} \oplus \{\mathtt{f} \mapsto t\}\rangle \\
move_{1\mathtt{f}}(\langle s, \mathsf{S}\rangle) &= \langle \mathsf{S}(\mathtt{f}) \cdot s, \mathsf{S} - \mathtt{f}\rangle \\
move_{2\mathtt{f\text{-}g}}(\langle s, \mathsf{S}\rangle) &= \langle s, (\mathsf{S} - \mathtt{f}) \\
&\qquad \oplus \{\mathtt{g} \mapsto \mathsf{S}(\mathtt{f})\}\rangle
\end{aligned}
$$

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

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 (sidewards, 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 ***interpreted regular tree grammar (IRTG)*** (Koller and Kuhlmann, 2011) $\mathbb{G} = \langle \mathcal{G}, (h_1, \mathcal{A}_1), \ldots, (h_n, \mathcal{A}_n)\rangle$ derives $n$-tuples of objects, such as strings or trees from *derivation trees* in $\mathcal{G}$. A given $t \in \mathcal{G}$ is *interpreted* into the $n$ algebras $\mathcal{A}_1, \ldots \mathcal{A}_n$ 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 $\mathcal{A}_i$, which is in turn ***evaluated*** ($[\![\cdot]\!]_{\mathcal{A}_i}$) into an object of the algebra. For example, suppose we have a minimalist string algebra $\mathcal{A}$ 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 $\{mv_1 \mapsto \mathsf{merge}_{1\mathtt{nom}}, mg_{13} \mapsto \mathsf{merge}_1, lex_{11} \mapsto \epsilon, mg_2 \mapsto \mathsf{merge}_{2\mathtt{nom}}, lex_9 \mapsto$ laughed, $lex_3 \mapsto$ Loki$\}$ yields the second tree in Table 3, call it $u$. Then we write $h(t) = u$ and $[\![u]\!]_{\mathcal{A}} = \langle\langle \text{Loki laughed}, \mathtt{T}\rangle, \emptyset\rangle$. The language of the grammar $L(\mathbb{G})$ is the set of tuples $\{\langle [\![h_1(t)]\!]_{\mathcal{A}_1}, \ldots, [\![h_n(t)]\!]_{\mathcal{A}_n}\rangle \mid t \in L(\mathcal{G})\}$.

An IRTG is *regular* in that $\mathcal{G}$ is a ***regular tree language***, meaning it is a set of trees that can be generated by a finite set of production rules of the form $NT_0 \to t$ or $NT_0 \to t(NT_1, \ldots, NT_n)$ 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 $f : [lic \rightsquigarrow \Sigma^* \times F^*] \rightarrow [lic \rightsquigarrow F^*]$ strip away the string parts of a storage function, leaving only the features. Then the nonterminals of RTG($g$) are $\{\langle \mathtt{fs}, f(\mathsf{S})\rangle | \langle \langle \mathrm{s}, \mathtt{fs}\rangle, \mathsf{S}\rangle \in CL(\mathsf{Expr}(Lex))\}$. 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 $\langle \mathrm{s}, \mathtt{fs}\rangle$ has associated with it a rule with left hand side $\langle \mathtt{fs}, \emptyset\rangle$. 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 $\{\langle \mathsf{S}, \emptyset\rangle \mid \mathsf{S} \in S\}$.

**Example 3.1.**

Let *sel* = {T,V,D,C}, *lic* = {nom,wh}. Let $S = \{\mathtt{T}, \mathtt{C}\}$ 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 interprtation 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 $\mathsf{merge}_1$. e.g.:
$h(mg_i(t_1, t_2)) = \mathsf{merge}_1(h(t_1), h(t_2))$ for $i \in \{1, 3, 5, 7, 8, 12, 13, 14, 15, 16, 17\}$

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

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

**Move 2** g-storing Move triggered by f is interpreted as $\mathsf{move}_{\mathtt{2f\text{-}g}}$. e.g.:
$h(mv_3(t)) = \mathsf{move}_{\mathtt{2nom\text{-}wh}}(h(t))$

**Lex** for a production rule $\langle \mathtt{fs}, \emptyset\rangle \rightarrow lex_i$, each $h(lex_i) = s$ for some $\langle \mathrm{s}, \mathtt{fs}\rangle \in Lex$. e.g.:
$h(lex_1) = h(lex_3) =$ 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 $\langle$*Loki laughed*, $\emptyset\rangle$.

## 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 minimalist 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

| MG rule application | RTG production rule |
|---|---|
| $merge_{1\|2}(\langle\langle s, \texttt{fs}\rangle, \mathsf{S}\rangle, \langle\langle t, \texttt{ft}\rangle, \mathsf{T}\rangle) = \langle\langle s', \texttt{fs}'\rangle, \mathsf{S}'\rangle$ | $\langle \texttt{fs}', f(\mathsf{S}')\rangle \rightarrow mg_x(\langle \texttt{fs}, f(\mathsf{S})\rangle, \langle \texttt{ft}, f(\mathsf{T})\rangle)$ |
| $move_{1\|2}(\langle\langle s, \texttt{fs}\rangle, \mathsf{S}\rangle) = \langle\langle s', \texttt{fs}'\rangle, \mathsf{S}'\rangle$ | $\langle \texttt{fs}', f(\mathsf{S}')\rangle \rightarrow mg_x(\langle \texttt{fs}, f(\mathsf{S})\rangle)$ |
| $\langle \mathrm{s}, \texttt{fs}\rangle \in Lex$ | $\langle \texttt{fs}, \emptyset\rangle \rightarrow lex_x$ |

Table 1: RTG(g) rule template

| Types | strings | feature stacks |
|---|---|---|
| Nominals | Loki, Thor | `D-nom \| D` |
| wh-words | who | `D-nom-wh \| D-wh` |
| Intransitive verbs | laughed, cried | `=D V` |
| Transitive verbs | slew, tricked | `=D =D V` |
| Tense | $\epsilon$ | `=V +nom T` |
| Complementiser | $\epsilon$ | `=T +wh C` |

Table 2: Sample lexicon

runtime. 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 $\mathbb{G} = (\mathcal{G}, (h, \mathcal{A}))$, a compact representation of the language $\mathsf{parses}(w) = \{t \in L(\mathcal{G}) \mid [\![h(t)]\!] = w\}$ – 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 $C$ such that $L(C) = L(\mathcal{G}) \cap h^{-1}(L(D_w))$ –, 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 $L(C) = \mathsf{parses}(w)$.

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.

Let $w \in \Sigma^*$ be a string that we want to parse with an IRTG $\mathbb{G}$ over the minimalist string algebra. The decomposition grammar $D_w$ will describe a language of terms over this algebra, such as the term in the lower left of Table 3. Let $\mathsf{Sp}$ be the set of all *spans* in $w$, i.e. of all pairs $(i, j)$ of string positions with $1 \leq i \leq j \leq n+1$. Then the nonterminals of $D_w$ will be pairs $[s, S]$ where $s \in \mathsf{Sp}$ and $S : lic \rightsquigarrow \mathsf{Sp}$ is a partial function that assigns spans to licensing features. We assume that $s$ and the spans for all features are pairwise disjoint. The start symbol is $[(1, n+1), \emptyset]$.

Now consider first the constants $c$ of the minimalist string algebra. These derive a span of length one or, if $c = \epsilon$, of length zero. Thus we obtain the following rules for $D_w$:

$$\begin{array}{rcll} [(i, i+1), \emptyset] & \rightarrow & c & \text{if } w_i = c \in \Sigma \\ [(i, i), \emptyset] & \rightarrow & \epsilon & \text{for all } 1 \leq i \leq n+1 \end{array}$$

Furthermore, terms can be combined into larger terms using the merge and move operations. The grammar $D_w$ contains rules which essentially evaluate these operations as defined in Section 2.3, in terms of the spans represented by each substring. Concretely, the rules look as in figure 6.

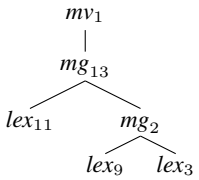
Rules in which $\oplus$ would yield undefined results (because a feature would appear twice in a partial function) do not exist in the grammar.

### 4.1 Parsing Complexity

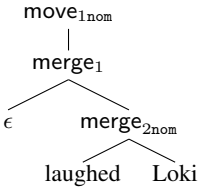
Asymptotic parsing complexity is determined by the time it takes to compute the rules of $D_w$; the rest of the IRTG parsing algorithm is linear in the size of $D_w$. The most costly rule of $D_w$, in terms of parsing complexity, is that for $merge_1$. In this rule there are $O(n^3)$ values for the string positions $i, j, p$. Within $\mathsf{S} \oplus \mathsf{T}$ 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.

| Lexical | | |
|---|---|---|
| $\langle$D, $\emptyset\rangle$ | $\to$ | $lex_1 \mid lex_2$ |
| $\langle$D-nom, $\emptyset\rangle$ | $\to$ | $lex_3 \mid lex_4$ |
| $\langle$D-nom-wh, $\emptyset\rangle$ | $\to$ | $lex_5$ |
| $\langle$D-wh, $\emptyset\rangle$ | $\to$ | $lex_6$ |
| $\langle$=D=DV, $\emptyset\rangle$ | $\to$ | $lex_7 \mid lex_8$ |
| $\langle$=DV, $\emptyset\rangle$ | $\to$ | $lex_9 \mid lex_{10}$ |
| $\langle$=V+nomT, $\emptyset\rangle$ | $\to$ | $lex_{11}$ |
| $\langle$=T+whC, $\emptyset\rangle$ | $\to$ | $lex_{12}$ |

Derivation tree




$\to_h$ Term of minimalist algorithm




$\to_{[\![\cdot]\!]}$ $\langle$Loki laughed, $\emptyset\rangle$

| Phrasal | | |
|---|---|---|
| | | ***merge* of subject** |
| $\langle$V, $\emptyset\rangle$ | $\to$ | $mg_1(\langle$=DV, $\emptyset\rangle, \langle$D, $\emptyset\rangle)$ |
| $\langle$V, {nom$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_2(\langle$=DV, $\emptyset\rangle, \langle$D-nom, $\emptyset\rangle) \mid$ |
| | | $mg_3(\langle$=DV, {nom$\mapsto\emptyset$}$\rangle, \langle$D, $\emptyset\rangle)$ |
| $\langle$V, {wh$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_4(\langle$=DV, $\emptyset\rangle, \langle$D-wh, $\emptyset\rangle) \mid$ |
| | | $mg_5(\langle$=DV, {wh$\mapsto\emptyset$}$\rangle, \langle$D, $\emptyset\rangle)$ |
| $\langle$V, {nom$\mapsto$-wh}$\rangle$ | $\to$ | $mg_6(\langle$=DV, $\emptyset\rangle, \langle$D-nom-wh, $\emptyset\rangle) \mid$ |
| | | $mg_7(\langle$=DV, {nom$\mapsto$-wh}$\rangle, \langle$D, $\emptyset\rangle)$ |
| | | ***merge* of object** |
| $\langle$=DV, $\emptyset\rangle$ | $\to$ | $mg_8(\langle$=D=DV, $\emptyset\rangle, \langle$D, $\emptyset\rangle)$ |
| $\langle$=DV, {nom$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_9(\langle$=D=DV, $\emptyset\rangle, \langle$D-nom, $\emptyset\rangle)$ |
| $\langle$=DV, {wh$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_{10}(\langle$=D=DV, $\emptyset\rangle, \langle$D-wh, $\emptyset\rangle)$ |
| $\langle$=DV, {nom$\mapsto$-wh}$\rangle$ | $\to$ | $mg_{11}(\langle$=D=DV, $\emptyset\rangle, \langle$D-nom-wh, $\emptyset\rangle)$ |
| | | **Tense selects VP** |
| $\langle$+nomT, $\emptyset\rangle$ | $\to$ | $mg_{12}(\langle$=V+nomT, $\emptyset\rangle, \langle$V, $\emptyset\rangle)$ |
| $\langle$+nomT, {nom$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_{13}(\langle$=V+nomT, $\emptyset\rangle, \langle$=DV, {nom$\mapsto\emptyset$}$\rangle)$ |
| $\langle$+nomT, {wh$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_{14}(\langle$=V+nomT, $\emptyset\rangle, \langle$V, {wh$\mapsto\emptyset$}$\rangle)$ |
| $\langle$+nomT, {nom$\mapsto$-wh}$\rangle$ | $\to$ | $mg_{15}(\langle$=V+nomT, $\emptyset\rangle, \langle$=DV, {nom$\mapsto$-wh}$\rangle)$ |
| | | **Subject moves to spec-TP** |
| $\langle$T, $\emptyset\rangle$! | $\to$ | $mv_1(\langle$+nom T, {nom$\mapsto\emptyset$}$\rangle)$ |
| $\langle$T, {wh$\mapsto\emptyset$}$\rangle$ | $\to$ | $mv_2(\langle$+nom T, {nom$\mapsto$-wh}$\rangle)$ |
| | | **C selects TP** |
| $\langle$+wh C, $\emptyset\rangle$ | $\to$ | $mg_{16}(\langle$=T +wh C, $\emptyset\rangle$ $\langle$T, $\emptyset\rangle)$ |
| $\langle$+wh C, {wh$\mapsto\emptyset$}$\rangle$ | $\to$ | $mg_{17}(\langle$=T +wh C, $\emptyset\rangle$ $\langle$T, {wh$\mapsto\emptyset$}$\rangle)$ |
| | | **wh-word moves to spec-CP** |
| $\langle$C, $\emptyset\rangle$! | $\to$ | $mv_3(\langle$+wh C, {wh$\mapsto\emptyset$}$\rangle)$ |

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

$$
\begin{array}{rcll}
[(i,p), \mathsf{S} \oplus \mathsf{T}] & \to & \mathit{merge}_1([(i,j),\mathsf{S}],[(j,p),\mathsf{T}]) & \\
[(i,j), \mathsf{S} \oplus \mathsf{T} \oplus \{\mathtt{f} \mapsto (p,l)\}] & \to & \mathit{merge}_2\mathtt{f}([(i,j),\mathsf{S}],[(p,l),\mathsf{T}]) & \\
[(i,p), \mathsf{S} - \mathtt{f}] & \to & \mathit{move}_{1\mathtt{f}}([(j,p),\mathsf{S}] & \text{if } \mathsf{S}(\mathtt{f}) = (i,j) \\
[(i,j), (\mathsf{S} - \mathtt{f}) \oplus \{\mathtt{g} \mapsto \mathsf{S}(\mathtt{f})\}] & \to & \mathit{move}_{2\mathtt{f}\text{-}\mathtt{g}}([(i,j),\mathsf{S}]), &
\end{array}
$$

Figure 6: Decomposition rules

## 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 Rewrite 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 $\mathcal{O}(n^{2k+3} \cdot 2^k)$ expressed in terms of fanout $f = k+1$ is therefore $\mathcal{O}(n^{2f+1} \cdot 2^{f-1})$, which is less than the parsing complexity for an arbitrary binary MCFG with fanout $f$: $\mathcal{O}(n^{3f})$.

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

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 $\text{MCFG}_{\text{MG}}$.

Another subclass of MCFGs with lowered parsing complexity is *well-nested* (Kuhlmann, 2007) MCFGs ($\text{MCFG}_{\text{wn}}$) in which no rule involves the interleaving of elements from two daughters (no $abab$ rules). The parsing complexity of a binary $\text{MCFG}_{\text{wn}}$ with fanout $f$ is $\mathcal{O}(n^{2f+2})$, 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 $\mathcal{O}(n^{2f+1})$ and $\mathcal{O}(n^{2f+2})$ 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 $\textit{merge}_1$ is closely related to that for $\text{MCFG}_{\text{wn}}$. The well-nested concatenation rules have the same number of indices as $\textit{merge}_1$. Therefore the complexity of $\textit{merge}_1$ ($\mathcal{O}(n^{2f+1} \cdot 2^{f-1})$) and concatenation rules for parsing a $\text{MCFG}_{\text{wn}}$ ($\mathcal{O}(n^{2f+1})$) have the same polynomial degree, $2f + 1$. This is perhaps counter-intuitive, since well-nested MCFLs 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 ($\text{MCFG}_{\text{mb}}$), 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 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 $\mathcal{O}(n^{2k+3})$. 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 $\mathcal{O}(n^{2k+3})$. 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 $\mathcal{O}(n^{4k+4})$ was in fact too conservative, and MGs can be parsed in the much smaller polynomial time of $\mathcal{O}(n^{2k+3} \cdot 2^k)$. MGs with the specifier island constraint have a parsing complexity of $\mathcal{O}(n^{2k+3})$.

[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, $\text{MCFL}_{\text{mb}} \subsetneq \text{MCFL}_{\text{wn}} \subsetneq \text{MCFL}$. 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.

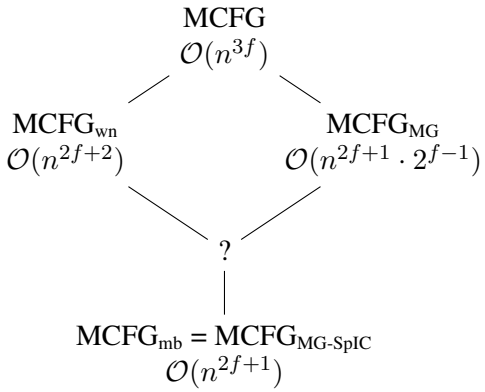



Figure 7: Subclasses of binary MCFGs with fanout $f$ and their parsing complexities

## References


Marisa Ferrara Boston, John Hale, and Marco Kuhlmann. 2010. Dependency structures derived from minimalist grammars. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language. 10th and 11th Biennial Conference, MOL 10, Revised Selected Papers*. Springer, volume 6149 of *Lecture Notes in Computer Science*, pages 1–12.

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

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2010)*. Association for Computational Linguistics, pages 276–284. http://www.aclweb.org/anthology/N10-1035.

Thomas Graf. 2012. Movement-generalized minimalist grammars. In Denis Béchet and Alexander J. Dikovsky, editors, *LACL 2012*. volume 7351 of *Lecture Notes in Computer Science*, pages 58–73.

Henk Harkema. 2001. *Parsing minimalist languages*. Ph.D. thesis, University of California Los Angeles.

Aravind Joshi. 1985. How much context-sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational and Psychological Perspectives*, Cambridge University Press, New York, pages 206–250.

Makoto Kanazawa, Jens Michaelis, Sylvain Salvati, and Ryo Yoshinaka. 2011. Well-nestedness properly subsumes strict derivational minimalism. In *International Conference on Logical Aspects of Computational Linguistics*. Springer, pages 112–128.

Greg Kobele. 2006. *Generating copies*. Ph.D. thesis, UCLA.

Gregory M. Kobele, Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In J. Rogers and S. Kepser, editors, *Model Theoretic Syntax at ESSLLI '07*. ESSLLI.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*. Dublin.

Marco Kuhlmann. 2007. *Dependency Structures and Lexicalized Grammars*. Doctoral Dissertation, Saarland University, Saarbrücken, Germany.

Jens Michaelis. 1998. Derivational minimalism is mildly context-sensitive. In *LACL*. Springer, volume 98, pages 179–198.

Jens Michaelis. 2001. Transforming linear context-free rewriting systems into minimalist grammars. In Philippe de Groote, Glyn Morrill, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics: 4th International Conference, LACL 2001 Le Croisic, France, June 27–29, 2001 Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 228–244. https://doi.org/10.1007/3-540-48199-0_14.

Jens Michaelis, Uwe Mönnich, and Frank Morawietz. 2000. Derivational minimalism in two regular and logical steps. In *Proceedings of the 5th international workshop on tree adjoining grammars and related formalisms (tag+ 5)*.

Uwe Mönnich. 2006. Grammar morphisms. *Ms. University of Tübingen* .

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

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On Multiple Context-Free Grammars. *Theoretical Computer Science* 88(2):191–229.

Stuart Shieber. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7)*.

Stuart Shieber. 2006. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*. Association for Computational Linguistics.

Stuart M Shieber. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence* 10(4):371–385.

Edward Stabler. 1997. Derivational minimalism. *Logical Aspects of Computational Linguistics* pages 68–95.

Edward P Stabler. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science* 5(3):611–633.

Edward P Stabler and Edward L Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science* 293(2):345–363.

Miloš Stanojević. 2016. Minimalist grammar transition-based parsing. In *Logical Aspects of Computational Linguistics. Celebrating 20 Years of LACL (1996–2016) 9th International Conference, LACL 2016, Nancy, France, December 5-7, 2016, Proceedings 9*. Springer, pages 273–290.

Krishnamurti Vijay-Shanker, David J Weir, and Aravind K Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 104–111.