Parsing Minimalist Languages à la Earley

Henk Harkema
Département de Linguistique
Université de Québec à Montréal
Montréal, Québec, Canada, H3C 3P8

Paper ID: P0117

Keywords: Parsing, Earley's Algorithm, Mildly Context-Sensitive Grammar Formalisms, Minimalism

Contact Author: Henk Harkema

Under consideration for other conferences (specify)? none

Abstract

This paper introduces an Earley-style parsing method for languages generated by Minimalist Grammars (Stabler, 1997). Minimalist Grammars are an algebraic formalization of the kind of grammars proposed in the linguistic framework of Chomsky's Minimalist Program (Chomsky, 1995). As Minimalist Grammars can model dependencies within sentences that are not found in Context-Free Languages, the Earley-style parsing method presented here is a generalization of Earley's original algorithm for Context-Free Grammars (Earley, 1970).

Parsing Minimalist Languages à la Earley

Paper ID: P0117

Abstract

This paper introduces an Earley-style parsing method for languages generated by Minimalist Grammars (Stabler, 1997). Minimalist Grammars are an algebraic formalization of the kind of grammars proposed in the linguistic framework of Chomsky's Minimalist Program (Chomsky, 1995). As Minimalist Grammars can model dependencies within sentences that are not found in Context-Free Languages, the Earley-style parsing method presented here is a generalization of Earley's original algorithm for Context-Free Grammars (Earley, 1970).

1 Introduction

Minimalist Grammars, henceforth MGs, are introduced in (Stabler, 1997) as a rigorous formalization of the kind of grammars proposed in the linguistic framework of Chomsky's Minimalist Program (Chomsky, 1995). In (Harkema, 2000) a chart-based bottom-up recognizer for MGs is presented. Since a bottom-up parser has no predictive power, the chart will contain numerous items that are not involved in the derivation of the sentence to be parsed. In particular, in order to be complete, a bottom-up parser has to assume that any phonetically empty category can intervene between any two adjacent words in a sentence. A top-down parser, however, has the ability to predict the presence of phonetically empty categories in a sentence based on the structure built so far. Since phonetically empty functional projections are ubiquitous in current transformational theories of natural language, this is an important motivation for the formulation of a top-down recognizer for MGs.

The bottom-up recognizer described in (Harkema, 2000) can be turned into a top-down recognizer by reversing the rules of inference and interchanging the axioms and goal items. However, as a pure top-down recognizer for Context-Free Grammars will loop forever on a sentence if the grammar is left-recursive, so the top-down recognizer for MGs obtained in this way will fail to halt for left-recursive MGs.[1] In this paper, we will present an Earley-style recognizer for MGs which will terminate for all grammar and sentence pairs.

For Context-Free Grammars, henceforth CFGs, if in a rule some non-terminal symbol A precedes another non-terminal symbol B, then in any sentence whose generation involves this rule, the terminal symbols derived from A must precede the terminal symbols derived from B. Similar claims hold for a terminal symbol and a non-terminal symbol in a rule, and two terminal symbols in a rule. This 'precedence property' of CFGs allows an Earley parser to use items with dotted rules, in which the dot moves from left to right through the right-hand side of a predicted rule as the input sentence is being parsed from left to right (Earley, 1970). As will become clear in section 2, MGs do not possess the precedence property. Consequently, an Earley-style parser for MGs cannot employ dotted rules in its items; the dot will have to be replaced by a more general mechanism for keeping track of the progress through a rule.

2 Minimalist Grammars

In the original formulation given in (Stabler, 1997), a Minimalist Grammar defines a set of trees. These trees are derived by closing the lexicon, which is a set of trees itself, under two structure building functions, *merge* and *move*. The function *merge* combines two trees into one, and the function *move* moves a subtree within a tree. In (Michaelis, 1998) it was shown how

[1] See footnote 3 for a definition of left-recursive MGs.

derivations in MGs correspond to derivations in Multiple Context-Free Grammars (Seki et al., 1991). This insight lead to a succinct reformulation of MGs in which trees are replaced with simpler expressions. Following the spirit of (Stabler and Keenan, 2000), we define an MG to be a sextuple $G = (\Sigma, F, \mathtt{c}, T, Lex, \mathcal{F})$, where Σ is a finite alphabet; F is a finite set of syntactic features; c is a distinguished feature in F; T is a finite set of types; Lex is a finite set of expressions built from Σ, F, and T; and $\mathcal{F}$ is a finite set of structure building functions.

The set of syntactic features is defined to be $F = \mathit{base} \cup \{\mathtt{=f} \mid \mathtt{f} \in \mathit{base}\} \cup \{\mathtt{+f} \mid \mathtt{f} \in \mathit{base}\} \cup \{\mathtt{-f} \mid \mathtt{f} \in \mathit{base}\}$, with $\mathit{base} \neq \emptyset$. By definition, $T = \{s, c\}$. Let a chain be an element from the set $C = \Sigma^* \times F^*$, and let an expression be an element from the set $E = C^+ \times T$. Expressions in $C^+ \times \{s\}$ are simple expressions, and expressions in $C^+ \times \{c\}$ are complex expressions. An expression $\langle\langle\sigma_1, \phi_1\rangle, \ldots, \langle\sigma_n, \phi_n\rangle, t\rangle \in E$ will be written as $[\sigma_1{:}\phi_1, \ldots, \sigma_n{:}\phi_n]_t$. Lex is defined to be a finite subset of $C \times \{s\} \subseteq E$.

The set of structure building functions consists of the functions *merge-1*, *merge-2*, and *merge-3*, all from $E \times E$ into E, and the functions *move-1* and *move-2*, both from E into E. Below, the definitions of these functions are given in the form of deductive rules, resembling the rules of inference of the bottom-up recognizer for MGs in (Harkema, 2000). For all functions, $t \in T$; $\gamma \in F^*$; $\delta \in F^+$; $\alpha_i \in C$, $0 \leq k$, $1 \leq i \leq k$; $\iota_i \in C$, $0 \leq l$, $1 \leq i \leq l$; and $\sigma\tau$ denotes the concatenation of any pair of strings $\sigma, \tau \in \Sigma^*$.

merge-1:

$$\frac{[\sigma{:}\,\mathtt{=x}\gamma]_s \qquad [\tau{:}\,\mathtt{x}, \alpha_1, \ldots, \alpha_k]_t}{[\sigma\tau{:}\,\gamma, \alpha_1, \ldots, \alpha_k]_c}$$

merge-2:

$$\frac{[\sigma{:}\,\mathtt{=x}\gamma, \alpha_1, \ldots, \alpha_k]_c \qquad [\tau{:}\,\mathtt{x}, \iota_1, \ldots, \iota_l]_t}{[\tau\sigma{:}\,\gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l]_c}$$

merge-3:

$$\frac{[\sigma{:}\,\mathtt{=x}\gamma, \alpha_1, \ldots, \alpha_k]_t \qquad [\tau{:}\,\mathtt{x}\delta, \iota_1, \ldots, \iota_l]_t}{[\sigma{:}\,\gamma, \alpha_1, \ldots, \alpha_k, \tau{:}\,\delta, \iota_1, \ldots, \iota_l]_c}$$

move-1:

$$\frac{[\sigma{:}\,\mathtt{+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, \tau{:}\,\mathtt{-f}, \alpha_{i+1}, \ldots, \alpha_k]_c}{[\tau\sigma{:}\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k]_c}$$

move-2:

$$\frac{[\sigma{:}\,\mathtt{+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, \tau{:}\,\mathtt{-f}\delta, \alpha_{i+1}, \ldots, \alpha_k]_c}{[\sigma{:}\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, \tau{:}\,\delta, \alpha_{i+1}, \ldots, \alpha_k]_c}$$

The domains of the functions *move-1* and *move-2* are restricted by the Shortest Movement Constraint: none of the sequences of syntactic features in the chains $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ can have -f as its left-most feature.

It will be convenient to write applications of the functions *merge-1* through *move-2* in a rewrite rule format. Hence, we will write $e \rightarrow e_1\ e_2$ if and only if $e = \mathit{merge\text{-}1}(e_1, e_2)$ or $e = \mathit{merge\text{-}2}(e_1, e_2)$ or $e = \mathit{merge\text{-}3}(e_1, e_2)$, and we will write $e \rightarrow e_1$ if and only if $e = \mathit{move\text{-}1}(e_1)$ or $e = \mathit{move\text{-}2}(e_1)$.

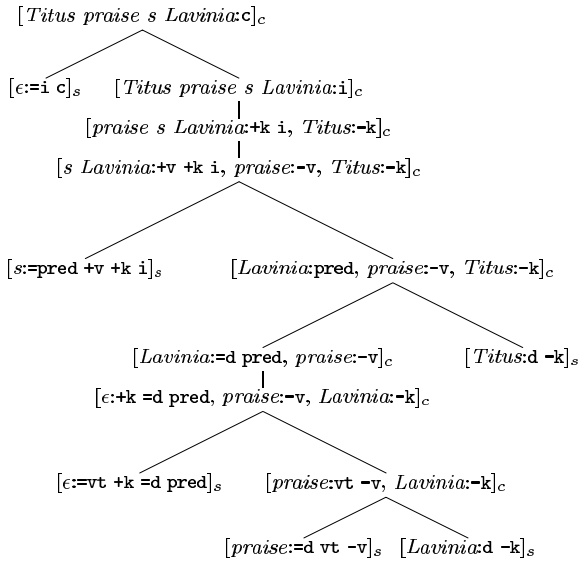
Let $G = (\Sigma, F, \mathtt{c}, T, Lex, \mathcal{F})$ be an MG, and E the set of expressions as defined above. For e an expression in E, and Γ, Δ sequences of expressions in E, we define the derivation relation $\Rightarrow$ as follows: $\Rightarrow = \{(\Gamma e\Delta, \Gamma e_1 e_2\Delta) \mid e \rightarrow e_1\ e_2\} \cup \{(\Gamma e\Delta, \Gamma e_1\Delta) \mid e \rightarrow e_1\}$. Let $\Rightarrow^*$ denote the reflexive transitive closure of $\Rightarrow$. Suppose $\Gamma \Rightarrow^* \Delta$. Then we say that Γ derives (the sequence) Δ. Also, when the context prevents any confusion, we will say that Γ derives (the expression) δ if Δ includes an expression δ, and that Γ derives (the word) w if Δ includes an expression $[w{:}\gamma]_s$. Setting $CL(G) = \{e \in E \mid \exists\, e_1, \ldots, e_n \in Lex{:}\, e \Rightarrow^* e_1 \ldots e_n\}$,[2] we define the language generated by G to be $L(G) = \{\sigma \mid [\sigma{:}\mathtt{c}]_t \in CL(G), t \in T\}$.[3] With regard to their generative capacity, MGs have been shown to be mildly context-sensitive (Harkema, 2001), (Michaelis, 2001).

By way of example, consider the MG G whose lexicon contains the following six expressions, where ϵ denotes the empty string: [*Lavinia*:d

[2] From a bottom-up perspective, CL(G) is the closure of the lexicon under the structure building functions.

[3] An MG G is left-recursive if $[\sigma{:}\mathtt{c}]_c \Rightarrow^* \Gamma\ [\sigma_1{:}\phi_1, \ldots, \sigma_n{:}\phi_n]_c\ \Delta$ and $[\sigma_1{:}\phi_1, \ldots, \sigma_n{:}\phi_n]_c \Rightarrow^+ \Gamma'\ [\sigma'_1{:}\phi_1, \ldots, \sigma'_n{:}\phi_n]_c\ \Delta'$, such that the sequence of strings $\sigma'_1, \ldots, \sigma'_n$ contains a string that in the sentence σ is to the left of all strings included in the expressions in Γ' and Δ'.

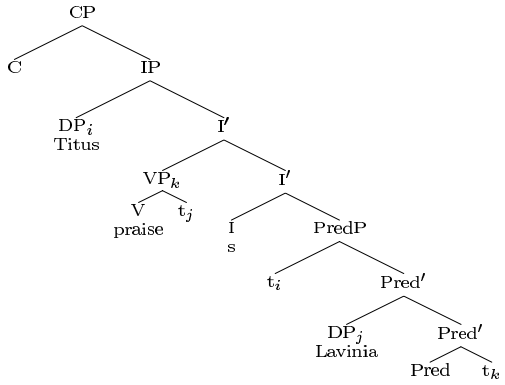
-k]$_s$, [*Titus*:d -k]$_s$, [*praise*:=d vt -v]$_s$, [*s*:=pred +v +k i]$_s$, [ϵ:=i c]$_s$, and [ϵ:=vt +k =d pred]$_s$. The derivation tree given below shows how these lexical expressions participate in the generation of the sentence *Titus praise s Lavinia*.[4]



For grammar G, for example, [*praise s Lavinia*:+k i, *Titus*: -k]$_c$ = *move-1*([*s Lavinia*: +v +k i, *praise*:-v, *Titus*: -k]$_c$), and [*praise*:vt -v, *Lavinia*: -k]$_c$ $\rightarrow$ [*praise*:=d vt -v]$_s$ [*Lavinia*:d -k]$_s$, and [ϵ:=i c]$_s$ [*Titus praise s Lavinia*:i]$_c$ $\Rightarrow^*$ [ϵ:=i c]$_s$ [*s*:=pred +v +k i]$_s$ [*Lavinia*:pred, *praise*:-v, *Titus*:-k]$_c$, and expression [*Lavinia*:=d pred, *praise*:-v]$_c$ derives the words *Lavinia*, *praise*, and an empty word at position 4 in the sentence *Titus praise s Lavinia*.

Grammar G illustrates the fact that MGs in general do not possess the precedence property mentioned in the introduction. Consider for example the rule $e \rightarrow e_1\ e_2$, where e = [*s Lavinia*:+v +k i, *praise*:-v, *Titus*:-k]$_c$, e_1 = [*s*:=pred +v +k i]$_s$, and e_2 = [*Lavinia*:pred, *praise*:-v, *Titus*:-k]$_c$, that is used in the generation of the example sentence. Expression e_1 is a lexical expression, and expression e_2 derives the four lexical expressions [ϵ:=vt +k =d pred]$_s$, [*praise*:=d vt -v]$_s$, [*Lavinia*:d -k]$_s$, and [*Titus*:d -k]$_s$. Rule $e \rightarrow e_1\ e_2$ violates the precedence property in an interesting way: even though expression e_1 precedes expression e_2 in the rule, in the sentence *Titus praise s Lavinia* the word associated with lexical expression e_1 precedes only some of the words derived from expression e_2 and follows the others. In particular, *s* precedes *Lavinia* and the empty word of lexical expression [ϵ:=vt +k =d pred]$_s$, and, as the result of movement, *s* follows both *Titus* and *praise*. In general, a depth-first, left-to-right traversal of the derivation tree given above does not visit the lexical expressions at the leaves of the tree in the order in which their words appear in the sentence *Titus praise s Lavinia*.

[4] A derivation tree contains all the information needed to construct a surface tree for a sentence; see (Stabler, 2001) for a discussion of the connection between the two kinds of tree. The surface tree for the sentence *Titus praise s Lavinia* according to grammar G follows below.



3 Earley Parsing

The canonical Earley algorithm for Context-Free Languages (Earley, 1970) avoids the non-termination problem of pure top-down parsers by predicting in a top-down manner what rules could be used to rewrite a non-terminal symbol, but only pursuing a predicted rule as long as there is bottom-up support from the input sentence for using that particular rule. The Earley-style recognizer for Minimalist Languages presented in this paper follows a similar strategy.

3.1 Items

The items of the Earley recognizer for MGs will contain rewrite rules which are instantiations of the structure building functions *merge-1* through *move-2*, with position vectors replacing the string parts of the expressions, e.g. [(2, 4):+v +k i, (1, 2):-v, (0, 1):-k]$_c$ $\rightarrow$ [(2, 3):=pred +v +k i]$_s$ [(3, 4):pred, (1, 2):-v, (0, 1):-k]$_c$ for the rule $e \rightarrow e_1\ e_2$ discussed in the example at the end of the previous section. The posi-

tion vectors indicate what portions of the input sentence are covered by the rule. As usual, the left-most position in a sentence is 0 and the right-most position is n, where n is the length of the input sentence. Expressions with position vectors rather than strings will be referred to as situated expressions. Expressions and situated expressions are related in the following way: given a sentence $w_1 \ldots w_n$, a situated expression $\sigma = [(x_0, y_0):\delta_0, \ldots, (x_k, y_k):\delta_k]_t$ corresponds to the expression $e_\sigma = [w_{x_0+1} \ldots w_{y_0}:\delta_0, \ldots, w_{x_k+1} \ldots w_{y_k}:\delta_k]_t$. Since situated expressions map straightforwardly onto expressions, we will informally use the derivation relation defined for expressions for situated expressions as well.

Like the items in the context-free version of the Earley algorithm, the items of the Earley recognizer for MGs will include an index pointing to the position in the sentence of the next (non-empty) word to be scanned.[5] Checking their position vectors the parser can identify those situated expressions in the right-hand side of a rule that are predicted to derive a word at the position in the sentence indicated by the index, and using the structure building functions *merge-1* through *move-2* in reverse the parser will be able to predict all possible expansions for these situated expressions.

Because MGs lack the precedence property, it is not possible to use a dot in a rewrite rule to record the progress of the parser through the right-hand side of a rule. For example, for the sentence *Titus praise s Lavinia* and the rule [(2, 4):`+v +k i`, (1, 2):`-v`, (0, 1):`-k`]$_c$ $\rightarrow$ [(2, 3):`=pred +v +k i`]$_s$ [(3, 4):`pred`, (1, 2):`-v`, (0, 1):`-k`]$_c$ introduced above, after having scanned the word s at position 2 for the situated expression [(2, 3):`=pred +v +k i`]$_s$, the recognizer would somehow have to put the dot in the middle of the situated expression [(3, 4):`pred`, (1, 2):`-v`, (0, 1):`-k`]$_c$, for this situated expression derives both words to the left and to the right of position 3.

The impossibility to dot rules has consequences for the way in which the MG recognizer must handle predicted empty words in a sentence. In a sentence of length n, any empty word at position x comes before the non-empty word at position x, $x \leq n$. Thus, in order for the parser to maintain the correct-prefix property, e.g. (Sippu and Soisalon-Soininen, 1988), it is necessary that the parser scan any predicted empty word at position x before the non-empty word at position x.[6] In the Earley parser for CFGs, the use of dots in rewrite rules ensures that the correct order is the only order in which the empty and non-empty words of a sentence can possibly be scanned. However, for MGs, with no dotted rules, the correct order has to be enforced explicitly. One way to achieve this is to use an index or pointer π with two parts. If $\pi = (x, \epsilon)$, the recognizer is pursuing expansions for situated expressions that possibly derive empty words at position x in the sentence; if $\pi = (x, \not\epsilon)$, the recognizer is pursuing expansions for situated expressions that possibly derive the one non-empty word at position x. The rules of inference of the recognizer are such that π will be updated from (x, ϵ) to $(x, \not\epsilon)$ only when all predicted empty words at position x have been scanned successfully.

The proper treatment of empty words requires the addition of another element to an item. Consider a CFG with a rule $E \rightarrow E_1\ E_2$ such that both non-terminal symbols E_1 and E_2 derive the empty string. Processing a sentence whose derivation involves this rule, the Earley parser will generate the items $[i, E \rightarrow \bullet\ E_1\ E_2, i]$, $[i, E \rightarrow E_1 \bullet E_2, i]$, and $[i, E \rightarrow E_1\ E_2 \bullet, i]$ for some position i in the sentence. Now consider the comparable MG rule $e \rightarrow e_1\ e_2$ such that both expressions e_1 and e_2 derive an empty word position i in the sentence. Since dotted rules are not available, we have to find another way to distinguish between the situations in which neither empty word has been scanned, one of them has been scanned,[7] and

[5] Following the notation of (Shieber et al., 1995), the index in a 'context-free' item $[i, \mathrm{A} \rightarrow \alpha \bullet \beta, j]$ is j.

[6] For a predicted situated expression $[(x, x):\gamma]_s$ the MG parser will always be able to find an empty word at position x, but scanning also involves checking that there is an expression $[\epsilon:\gamma]_s \in$ Lex, which may not be the case; cf. the rule of inference Up discussed in section 3.5.

[7] Empty words at the same position in a sentence are unordered, so they can be scanned in any order.

both words have been scanned. To this end, an item will include a set $\mathcal{C}$, containing all situated expressions of the right-hand side of the rule mentioned in the item that potentially derive a word at the position and of the kind indicated by π. Under certain conditions to be discussed in section 3.5, situated expressions will be removed from $\mathcal{C}$. For the example rule $e \rightarrow e_1\ e_2$, the algorithm will generate items with $\mathcal{C} = \{\sigma_{e_1}, \sigma_{e_2}\}$, $\mathcal{C} = \{\sigma_{e_2}\}$, $\mathcal{C} = \{\sigma_{e_1}\}$, and $\mathcal{C} = \emptyset$, provided that the generation of the input sentence actually involves the rule $e \rightarrow e_1\ e_2$, where σ_e stands for the situated expression corresponding to any expression e. In all four items, $\pi = (i, \epsilon)$. For purposes of uniformity and to simplify the correctness proofs, the use of set $\mathcal{C}$ is extended to the scanning of non-empty words, even though there can be one and only one non-empty word at any position in the input sentence.

In conclusion, the items of the Earley recognizer for MGs are triples of the form $\langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r, \mathcal{C}, \pi \rangle$, where $\mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r$ is a rule, A a situated expression or the special start symbol S′, and $\mathrm{B}_1, \ldots, \mathrm{B}_r$ a sequence of situated expressions;[8] $\mathcal{C}$ is a set containing expressions selected from $\mathrm{B}_1, \ldots, \mathrm{B}_r$;[9] and π is a position in the input sentence, annotated with either ϵ or $\not\epsilon$.

3.2 Positions and Position Vectors

Before discussing the invariant and the rules of inference of the Earley algorithm for MGs, we need to introduce some definitions regarding positions in a sentence, position vectors in a rule, and the connections between them.

For a sentence of length n, pointer π will assume the values $(0, \epsilon)$, $(0, \not\epsilon)$, $(1, \epsilon)$, $(1, \not\epsilon)$, …, $(n-1, \not\epsilon)$, (n, ϵ). The functions *next*, *prev* from the set of positions into the set of positions are defined as follows: $next(x, \epsilon) = (x, \not\epsilon)$, $next(x, \not\epsilon) = (x+1, \epsilon)$, and $prev(x, \epsilon) = (x-1, \not\epsilon)$, $prev(x, \not\epsilon) = (x, \epsilon)$. 'The word at position π' or 'the word pointed to by π' refers to an empty word at position x in the input sentence if $\pi = (x, \epsilon)$, and to the non-empty word at that position if $\pi = (x, \not\epsilon)$.

Whenever the parser generates an item $\langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r, \mathcal{C}, \pi \rangle$ it has to put in $\mathcal{C}$ all situated expressions B_i, $1 \leq i \leq r$, that possibly derive a word at position π. For this purpose, we will define a containment relation between position vectors and positions. A position vector (q, q) in a simple situated expression contains the following position: (q, ϵ). Position vector (q, r), $q < r$, in a simple situated expression contains the following positions: $(q, \not\epsilon), \ldots, (r-1, \not\epsilon)$. Position vector (q, r), $q \leq r$, in a complex situated expression contains the following positions: $(q, \epsilon), \ldots, (r, \epsilon)$.

For example, for grammar G defined in section 2, the position vector (3, 4) of the complex situated expression $\sigma = [(3, 4)$:`=d pred`, $(1, 2)$:`-v`$]_c$ contains, among others, the positions $(1, \epsilon)$ and $(3, \not\epsilon)$, indicating that the corresponding expression $e_\sigma = [$*Lavinia*:`=d pred`, *praise*:`-v`$]_c$ possibly derives an empty word at position 1 and the non-empty word at position 3 of the input sentence. The positions $(4, \not\epsilon)$ and $(0, \epsilon)$ are not contained in any of the position vectors of σ, meaning that e_σ cannot possibly derive an empty word at position 0 nor the non-empty word at position 4. The position vector of the simple situated expression $\tau = [(0, 1)$:`d -k`$]_s$ only contains position $(1, \not\epsilon)$, meaning that the string part of the corresponding expression e_τ covers the first non-empty word of the sentence and no other empty or non-empty words.

Let π_l be the left-most position and π_r be the right-most position contained in the position vectors of a situated expression σ. Then σ is to the right of position π_l and all positions preceding π_l, and to the left of position $next(\pi_r)$ and all positions following $next(\pi_r)$. Furthermore, σ crosses all positions between π_l and $next(\pi_r)$ (excluding π_l and $next(\pi_r)$).

Returning to our example, the situated expression $\sigma = [(3, 4)$:`=d pred`, $(1, 2)$:`-v`$]_c$ crosses position $(2, \not\epsilon)$; the corresponding expression e_σ derives words to the right and to the left of position $(2, \not\epsilon)$. Note that σ also crosses position $(1, \not\epsilon)$, even though e_σ does not derive a word to the left of position $(1, \not\epsilon)$. However, judging solely from the position vectors of σ, it is pos-

[8] $r = 1$ or $r = 2$.

[9] $\mathcal{C}$ is a set of occurrences, as $\mathrm{B}_1, \ldots, \mathrm{B}_r$ will always be a sequence of distinct situated expressions.

sible that e_σ derives an empty word at position (1, ϵ), which is to the left of (1, $\not\epsilon$).

3.3 Invariant

Given an MG G = (Σ, F, c, T, Lex, $\mathcal{F}$) and an input sentence $w = w_1 \ldots w_n$, the item $\langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r,\ \mathcal{C},\ \pi \rangle$ expresses the following claims about the grammaticality of the input sentence.[10] There is a sequence of situated expressions $\mathrm{D}_1, \ldots, \mathrm{D}_m$ such that:

a) $[w{:}\mathsf{c}]_t \Rightarrow^* e_{D_1} \ldots e_{D_i}\ e_A\ e_{D_{i+1}} \ldots e_{D_m}$, for some i, $1 \leq i \leq m$, $t \in \{s, c\}$.

b) No D_j, $1 \leq j \leq m$, crosses π.

c) For any D_j to the left of π, $1 \leq j \leq m$, $e_{D_j} \in$ Lex.

Furthermore, for every situated expression $\mathrm{B}_i \in \mathcal{C}$, $1 \leq i \leq r$, there is a sequence of situated expressions $\mathrm{E}_{i1}, \ldots, \mathrm{E}_{ip_i}$ such that:

d) $e_{B_i} \Rightarrow^* e_{E_{i1}} \ldots e_{E_{ip_i}}$.

e) No E_{ij}, $1 \leq j \leq p_i$, crosses π.

f) For any E_{ij} to the left of π, $1 \leq j \leq p_i$, $e_{E_{ij}} \in$ Lex.

[10] Compare these claims with similar claims following from a context-free item $[i, \mathrm{A} \rightarrow \gamma \bullet \delta, j]$ together with the precedence property of CFGs. There are sequences α, β such that:

a) $\mathrm{S} \Rightarrow^* \alpha\ \mathrm{A}\ \beta$.

b) α must derive a string that is to the left of j, β must derive a string that is to the right j.

c) $\alpha \Rightarrow^* w_1 \ldots w_i$.

Furthermore, there is a sequence μ such that:

d) $\delta \Rightarrow^* \mu$.

e) μ must derive a string that is to the right of j.

f) (Not applicable since μ is to the right of j.)

Finally, there is a sequence ν such that:

g) $\gamma \Rightarrow^* \nu$.

h) ν must derive a string that is to the left of j.

i) $\nu \Rightarrow^* w_{i+1} \ldots w_j$.

The correspondence between MGs and CFGs for claims g), h), and i) is not perfect, because in a context-free item $[i, \mathrm{A} \rightarrow \gamma \bullet \delta, j]$ the dot cannot move without the value of j being updated in the same step, whereas in an MG item $\langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r, \mathcal{C}, \pi \rangle$ a situated expression B_i will be removed from $\mathcal{C}$ by one rule of inference and the value of π will be updated by another, cf. section 3.5.

Finally, for every situated expression $\mathrm{B}_i \notin \mathcal{C}$, $1 \leq i \leq r$, there is a sequence of situated expressions $\mathrm{E}_{i1}, \ldots \mathrm{E}_{ip_i}$, such that:

g) $e_{B_i} \Rightarrow^* e_{E_{i1}} \ldots e_{E_{ip_i}}$.

h) No E_{ij}, $1 \leq j \leq p_i$, crosses *next*(π).

i) For any E_{ij} to the left of *next*(π), $1 \leq j \leq p_i$, $e_{E_{ij}} \in$ Lex.

3.4 Axioms and Goal Items

Given an MG G = (Σ, F, c, T, Lex, $\mathcal{F}$) and an input sentence $w = w_1 \ldots w_n$, an expression $[\sigma{:}\mathsf{c}]_s \in$ Lex licenses the axiom $\langle \mathrm{S}' \rightarrow [(0, n){:}\mathsf{c}]_s,\ \mathcal{C},\ (0, \epsilon) \rangle$, where $[(0, n){:}\mathsf{c}]_s \in \mathcal{C}$ if and only if position $(0, \epsilon)$ is contained in position vector $(0, n)$, i.e. $n = 0$. An expression $[\sigma{:}\gamma\mathsf{c}]_s \in$ Lex, $\gamma \in \mathrm{F}^+$, licenses the axiom $\langle \mathrm{S}' \rightarrow [(0, n){:}\mathsf{c}]_c,\ \{[(0, n){:}\mathsf{c}]_c\},\ (0, \epsilon) \rangle$.[11]

Any item of the form $\langle \mathrm{S}' \rightarrow [(0, n){:}\mathsf{c}]_t,\ \emptyset,\ (n, \epsilon) \rangle$, $t \in \{s, c\}$, is a goal item. The reader should verify that the axioms are sound, and that a goal item amounts to the claim that the sentence to be parsed is a sentence in the language generated by G.[12]

3.5 Rules of Inference

The operation of the Earley recognizer for MGs can be summarized as follows. The generation of an item $\langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r,\ \mathcal{C},\ \pi \rangle$ means that the recognizer proposes to use the rule $e_A \rightarrow e_{B_1} \ldots e_{B_r}$ in the generation of the input sentence and has already established that the use of this rule is justified by the words of the input sentence to the left of the word pointed to by π. The recognizer will now seek to confirm that the next word in the input sentence, i.e. the word pointed to by π, also justifies the use of this rule. Set $\mathcal{C}$ will contain all situated categories B_i, $1 \leq i \leq r$, that potentially derive a word at position π. Thus, for each $\mathrm{B}_i \in \mathcal{C}$ the recognizer must verify that the word at position π derived from B_i matches the actual word found at position π in the sentence, or that B_i does not derive a word at position π after all. The latter situation

[11] Note that for a complex situated expression position $(0, \epsilon)$ is always contained in position vector $(0, n)$.

[12] Since S′ is a special symbol, claims a), b), and c) of the invariant do not apply to the axioms and goal items.

may arise if π is pointing to an empty word. In either case, B_i will be removed from $\mathcal{C}$. Hence, if $\mathcal{C} = \emptyset$, the desired confirmation has been found.

The recognizer has four rules of inference: Next, Predict, Down, and Up. The function of the rule Next is to increment the value of the pointer π. According to the invariant, an item $\iota = \langle S' \rightarrow [(0, n):\mathtt{c}]_t, \emptyset, \pi\rangle$ claims that there is a sequence of situated expressions $E_{11}, \ldots, E_{1p_1}$ such that $[w:\mathtt{c}]_t \Rightarrow^* e_{E_{11}} \ldots e_{E_{1p_1}}$, where E_{1j}, $1 \leq j \leq p_1$, is harmonious with the words of the input sentence up to and including the word at position π, that is, no E_{1j}, $1 \leq j \leq p_1$, crosses $next(\pi)$ and for any E_{1j} to the left of $next(\pi)$, $1 \leq j \leq p_1$, $e_{E_{1j}} \in$ Lex. Next, the recognizer should check whether this derivation can be extended to include a lexical item for the next word of the input sentence. Given item ι, rule Next will return the item $\iota' = \langle S' \rightarrow [(0, n):\mathtt{c}]_t, \mathcal{C}, next(\pi)\rangle$, provided that $\pi < (n, \epsilon)$,[13] where $[(0, n):\mathtt{c}]_t \in \mathcal{C}$ if and only if position $next(\pi)$ is contained in position vector $(0, n)$.

Given an item $\iota = \langle A \rightarrow B_1 \ldots B_r, \mathcal{C}, \pi\rangle$, the Predict rules will apply to expand any situated expression $B_i \in \mathcal{C}$, $1 \leq i \leq r$, in accordance with the structure building functions defined in section 2. For example, if $B_i = [(p, q):\gamma, S]_c \in \mathcal{C}$, then the particular Predict rule Unmerge-1 will produce all items $\langle [(p, q):\gamma, S]_c \rightarrow [(p, v):{=}\mathtt{x}\gamma]_s\ [(v, q):\mathtt{x}, S]_t, \mathcal{C}', \pi\rangle$ for any $\mathtt{x} \in$ base for which there are expressions in Lex with syntactic features ${=}\mathtt{x}\gamma$ and $\beta\mathtt{x}$, and such that $p \leq v \leq q$; $t = s$ if $\beta = \emptyset$, $t = c$ if $\beta \neq \emptyset$, and $[(p, v):{=}\mathtt{x}\gamma]_s \in \mathcal{C}'$ if and only if position π is contained in position vector (p, v) and $[(v, q):\mathtt{x}, S]_t \in \mathcal{C}'$ if and only if position π is contained in one of the position vectors of $[(v, q):\mathtt{x}, S]_t$. Note that for any particular item $\iota' = \langle [(p, q):\gamma, S]_c \rightarrow [(p, v):{=}\mathtt{x}\gamma]_s\ [(v, q):\mathtt{x}, S]_t, \mathcal{C}', \pi)\rangle$ delivered by Unmerge-1, $[w_{p+1} \ldots w_q:\gamma, S']_c = merge\text{-}1([w_{p+1} \ldots w_v:{=}\mathtt{x}\gamma]_s, [w_{v+1} \ldots w_q:\mathtt{x}, S']_t)$, where S′ is like S but with strings instead of position vectors. Similarly defined Predict rules are formulated for the other structure building functions. All the Predict rules share an important restriction on the value of π: the rules should only apply to an item ι if π is the left-most position contained in the position vectors of $B_i \in \mathcal{C}$. If π is not, the recognizer cannot soundly predict item ι' from item ι. In our example, item ι' claims that there is a sequence of situated expressions Γ whose elements obey claims d), e), f) or g), h), i) of the invariant, such that $[w_{p+1} \ldots w_q:\gamma, S']_c \Rightarrow [w_{p+1} \ldots w_v:{=}\mathtt{x}\gamma]_s\ [w_{v+1} \ldots w_q:\mathtt{x}, S']_t \Rightarrow^* \Gamma$. If π is left-most in B_i, such a Γ does indeed exist: $\Gamma = [w_{p+1} \ldots w_v:{=}\mathtt{x}\gamma]_s\ [w_{v+1} \ldots w_q:\mathtt{x}, S']_t$. Note that if π is left-most in B_i, then the left-most positions in $[(p, v):{=}\mathtt{x}\gamma]_s$ and $[(v, q):\mathtt{x}, S]_t$ must be π or a position to the right of π, wherefore neither $[(p, v):{=}\mathtt{x}\gamma]_s$ nor $[(v, q):\mathtt{x}, S]_t$ crosses π (or $next(\pi)$) or is to the left of π (or $next(\pi)$). This does not work if π is not left-most in B_i. Since $B_i \in \mathcal{C}$, it follows from ι that there is a sequence of situated expressions Δ, all the elements of which are harmonious up to π, such that $[w_{p+1} \ldots w_q:\gamma, S']_c \Rightarrow^* \Delta$, but the recognizer cannot assume that $[w_{p+1} \ldots w_q:\gamma, S']_c \Rightarrow [w_{p+1} \ldots w_v:{=}\mathtt{x}\gamma]_s\ [w_{v+1} \ldots w_q:\mathtt{x}, S']_t \Rightarrow^* \Delta$.

The rule of inference Down applies to an item $\iota = \langle A \rightarrow B_1 \ldots B_r, \mathcal{C}, \pi\rangle$ with $B_i \in \mathcal{C}$, $1 \leq i \leq r$, when π is not the left-most position contained in the position vectors of B_i. Contingent on the existence of another item $\tau = \langle B_i \rightarrow D_1 \ldots D_m, \emptyset, \pi'\rangle$, where π' is the right-most position to the left of π contained in the position vectors of B_i, Down will posit an item $\iota' = \langle B_i \rightarrow D_1 \ldots D_m, \mathcal{C}', \pi\rangle$, where $D_i \in \mathcal{C}'$ if and only if position π is contained in the position vectors of D_i, $1 \leq i \leq m$. Note that item τ is derived from an item concerning the expansion of B_i that was predicted for a position π' that comes before position π. Informally, the Predict rules produce "new" predictions about expansions of some situated expression B_i, whereas the rule Down is used to retrieve and update "old" ones. This arrangement differs significantly from the Earley algorithm for CFGs. Since CFGs have the precedence property, there is never a need to temporarily abandon a predicted rule and resume working on it at some later stage.

Finally, the rule Up takes care of removing

[13] This condition ensures that the empty word at position n will be the last word considered by the recognizer; no sentence of length n will contain any words beyond the empty word at position (n, ϵ).

situated expressions from set $\mathcal{C}$. For an item $\iota = \langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_r, \mathcal{C}, \pi \rangle$, $\mathrm{B}_i \in \mathcal{C}$, $1 \leq i \leq r$, such that B_i is a simple situated expression, the result of rule Up will be an item $\iota' = \langle \mathrm{A} \rightarrow \mathrm{B}_1 \ldots \mathrm{B}_n, \mathcal{C}', \pi \rangle$, where $\mathcal{C}' = \mathcal{C}$ - $\{\mathrm{B}_i\}$, if e_{B_i} is an expression $[\sigma{:}\gamma]_s \in$ Lex and σ matches the word at position π in the input sentence. If B_i is a complex situated expression, Up will remove B_i from $\mathcal{C}$ if there is an item $\tau = \langle \mathrm{B}_i \rightarrow \mathrm{D}_1 \ldots \mathrm{D}_m, \emptyset, \pi \rangle$. In both cases the derivation from e_A implied to exist by item ι is harmonious with the words in the input sentence to the left of $next(\pi)$ as far as expression e_{B_i} is concerned.

4 Conclusion

In this paper we have specified an Earley-style recognizer for languages generated by MGs. Full details of the specification, including proofs of soundness and completeness, and a complexity analysis can be found in (X., 2001).[14,15] The recognizer can be turned into a parser by extending items with a field for recording their immediate ancestors, and using this field to retrieve trees or forests from the chart of items produced by the algorithm.

We hope that the insights provided by the recognizer described in this paper will also be helpful for the formulation of Earley-style recognizers for other grammar formalisms that model the same kind of discontinuous dependencies as MGs do, such as Multiple Context-Free Grammars (Seki et al., 1991), Simple Positive Range Concatenation Grammars (Boullier, 1998), and Multi-Component Tree-Adjoining Grammars (Weir, 1988).

References

P. Boullier. 1998. *Proposal for a Natural Language Processing Syntactic Backbone.* Research Report N° 3342, INRIA-Rocquencourt, France.

N. Chomsky. 1995. *The Minimalist Program.* MIT Press.

J.C. Earley. 1970. An Efficient Context-Free Parsing Algorithm. *Communications of the Association for Computing Machinery*, 13(2).

H. Harkema. 2000. A Recognizer for Minimalist Grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies.* Trento, Italy.

H. Harkema. 2001. A Characterization of Minimalist Grammars. In P. de Groote, G.F. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, Lecture Notes in A.I. 2099. Springer.

J. Michaelis. 1998. Derivational Minimalism is Mildly Context-Sensitive. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics*, Lecture Notes in A.I. 2014. Springer.

J. Michaelis. 2001. Transforming Linear Context-Free Rewriting Systems into Minimalist Grammars. In P. de Groote, G.F. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, Lecture Notes in A.I. 2099. Springer.

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

S.M. Shieber, Y. Schabes, and F.C.N. Pereira. 1995. Principles and Implementation of Deductive Parsing. *Journal of Logic Programming*, 24.

S. Sippu and E. Soisalon-Soininen. 1988. *Parsing Theory, Vol. I: Languages and Parsing.* Springer.

E.P. Stabler and E.L. Keenan. 2000. Structural Similarity. In A. Nijholt, G. Scollo, and D. Heylen, editors, *Algebraic Methods in Language Processing.* University of Iowa.

E.P. Stabler. 1997. Derivational Minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, Lecture Notes in A.I. 1328. Springer.

E.P. Stabler. 2001. Minimalist Grammars and Recognition. In C. Rohrer, A. Rossdeutscher, and H. Kamp, editors, *Linguistic Form and Its Computation.* CSLI Publications.

D.J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. thesis, University of Pennsylvania.

X. 2001. *Parsing Minimalist Languages.* Ph.D. thesis, UCLA.

[14] I cannot give the complete reference without compromising my anonymity, but if the reviewers want to consult the dissertation, I would be happy to send it to them via the area chair.

[15] Both the space and the time complexity of the algorithm are polynomial in the length of the input sentence.