PARTIALLY ORDERED MULTISET CONTEXT-FREE GRAMMARS AND FREE-WORD-ORDER PARSING

Mark-Jan Nederhof
Faculty of Arts
Univ. of Groningen
P.O. Box 716
9700 AS Groningen
The Netherlands
markjan@let.rug.nl

Giorgio Satta
Dept. of Inf. Eng'g.
University of Padua
via Gradenigo, 6/A
I-35131 Padova
Italy
satta@dei.unipd.it

Stuart Shieber
Div. of Eng'g. and Appl. Sci.
Harvard University
33 Oxford Street
Cambridge, MA 02138
USA
shieber@deas.harvard.edu

Abstract

We present a new formalism, partially ordered multiset context-free grammars (poms-CFG), along with an Earley-style parsing algorithm. The formalism, which can be thought of as a generalization of context-free grammars with partially ordered right-hand sides, is of interest in its own right, and also as infrastructure for obtaining tighter complexity bounds for more expressive context-free formalisms intended to express free or multiple word-order, such as ID/LP grammars. We reduce ID/LP grammars to poms-grammars, thereby getting finer-grained bounds on the parsing complexity of ID/LP grammars. We argue that in practice, the width of attested ID/LP grammars is small, yielding effectively polynomial time complexity for ID/LP grammar parsing.

1 Introduction

In this paper, we present a new formalism, partially ordered multiset context-free grammars (poms-CFG), along with an Earley-style parsing algorithm. The formalism shares with regular-right-part grammars (Lalonde, 1977) the idea of augmenting the operators on the right-hand side of productions with operators other than concatenation, in particular, an interleaving operator that can be thought of as generalizing context-free grammars with partially ordered right-hand sides. As such, it is of interest in its own right, and also as infrastructure for obtaining tighter complexity bounds for more expressive context-free formalisms intended to express free or multiple word-order, such as ID/LP grammars (Shieber, 1984). For the purpose of motivation, we turn to this latter application first.

Shieber (1984) presented a parsing algorithm for ID/LP grammars, along with an erroneous claim that parsing complexity was polynomial both in grammar size and string length. Barton (1985) corrected this claim, showing that the parsing algorithm was exponential in the grammar size and that this was intrinsic, as the problem of ID/LP parsing is NP-complete. (See also the chapter by Barton et al. (1987).) By recasting ID/LP grammars with poms-CFG, we can generate refined bounds on ID/LP parsing complexity; we argue that these lead to polynomial complexity in practice.

The development of poms-CFG grew out of Nederhof and Satta's work on IDL-expressions (Nederhof and Satta, 2002), a formalism for compactly representing finite languages that allow interleaving of elements. Their application was to the filtering stage of a two-level generation algorithm. Such algorithms first generate a finite but huge set of candidate sentences using shallow generation methods, and then filter them based on finer-grained statistico-grammatical grounds. This second stage requires parsing all of the sentences in the set generated by the first stage. Nederhof and Satta presented a formalism, IDL-expressions, that allow exponentially compact representation of finite languages, and an algorithm for parsing an IDL-expression relative to a context-free grammar. Notably, the complexity of parsing an IDL-expression π is $O(|G| \left(\frac{2|\pi|}{k}\right)^{3k})$ where $|\pi|$ is a measure of the size of the IDL-expression (analogous to the length n of a string) and k is the *width* of the expression, described further below.

Since in the worst case, $k = O(|\pi|)$, the algorithm is effectively exponential in the length of the string. But in concrete cases, where k is small, the complexity is polynomial, and when the compaction power of IDL-expressions is unused and $k = 1$, the algorithm performs as a standard n^3 algorithm.

In the sequel, we show an analogous result for the dual problem of parsing strings (rather than IDL-expressions) with poms-CFGs (rather than context-free grammars). Here, the extra expressivity is in the grammar, not the string, and the complexity increase goes to the grammar size complexity, rather than the string length. The class of poms-CF grammars allows compact representation of languages with free word order, and can be thought of as a generalization of ID/LP grammars. We reduce ID/LP grammars to poms-grammars, thereby getting finer-grained bounds on the parsing complexity of ID/LP grammars. We argue that in practice, the effective width of attested ID/LP grammars is small, yielding effectively polynomial time complexity for ID/LP grammar parsing.

The paper is structured as follows: After introducing some notational preliminaries (Section 2), we describe poms-expressions and their use in poms-CFGs (Section 3), and present an automata-theoretic equivalent to poms-expressions useful for parsing (Section 4). We then provide a parsing algorithm for the poms-CFG formalism (Section 5), investigate its implementation and complexity (Section 6), and discuss its application to ID/LP grammars (Section 7).

2 Notational Preliminaries

For a set Δ, $|\Delta|$ denotes the number of elements in Δ; for a string w, $|w|$ denotes the length of w. If $w = a_1 a_2 \cdots a_n$, with each a_i a symbol from the underlying alphabet, we write $w_{i,j}$, $0 \leq i \leq j \leq n$, to represent substring $a_{i+1} \cdots a_j$. We notate the empty string with ε.

We follow standard formal language notation as used by Hopcroft and Ullman (1979). A context-free grammar (CFG) is represented as a tuple $G = (V_N, V_T, S, P)$, where V_N and V_T are finite sets of nonterminal and terminal symbols, respectively, $S \in V_N$ is the start symbol and P is a finite set of productions of the form $A \rightarrow \alpha$ with $A \in V_N$ and $\alpha \in V^*$, $V = V_N \cup V_T$. We typically use the symbols A, B, C, for elements of V_N, a, b, c, for elements of V_T, and X, Y, Z for elements of V.

We also use the standard derive relation $\Rightarrow_G$ and its reflexive and transitive closure $\Rightarrow_G^*$. The language generated by grammar G is denoted $L(G)$. The size of G is notated $|G|$ and defined as

$$|G| = \sum_{(A \rightarrow \alpha) \in P} |A\alpha| \qquad .$$

The shuffling of strings will be an essential construct in this work. For $\alpha, \beta \in V^*$ we let

$$\mathsf{shuffle}(\alpha, \beta) \quad = \quad \{\alpha_1\beta_1\alpha_2\beta_2 \cdots \alpha_n\beta_n \mid \alpha_1\alpha_2 \cdots \alpha_n = \alpha,\ \beta_1\beta_2 \cdots \beta_n = \beta,$$

$$\alpha_i, \beta_i \in V^*,\ 1 \le i \le n,\ n \ge 1\}. \tag{1}$$

For languages $L_1, L_2 \subseteq V^*$, we define $\mathsf{shuffle}(L_1, L_2) = \cup_{\alpha \in L_1, \beta \in L_2} \mathsf{shuffle}(\alpha, \beta)$. More generally, for languages $L_1, L_2, \ldots, L_d \subseteq V^*$, $d \ge 2$, we define the shuffle of the languages

$$\mathsf{shuffle}_{i=1}^{d} L_i = \begin{cases} \mathsf{shuffle}(L_1, L_2) & \text{if } d = 2 \\ \mathsf{shuffle}(\mathsf{shuffle}_{i=1}^{d-1} L_i, L_d) & \text{if } d > 2 \end{cases} .$$

Finally, for languages L_1, L_2 as above we define their concatenation as $L_1 \cdot L_2 = \{\alpha \cdot \beta \mid \alpha \in L_1,\ \beta \in L_2\}$.

3 Partially Ordered Multi-Set CFGs

In this section we define partially ordered multi-set context-free grammars (poms-CFG). To represent partially ordered multi-sets, or poms for short, we define poms-expressions, which are a syntactic variant of the pomsets of Gischer (1988), inspired by the standard regular expression notation.

The main idea of poms-expressions is to use the concatenation and disjunction operators from standard regular expressions, written "·" and "+", respectively, along with the novel operator "$\|$", called **interleave**. The interleave operator interleaves strings resulting from its argument expressions. As an introductory example, the poms-expression

$$A \cdot a \cdot A \parallel (B \cdot b + C \cdot c) \tag{2}$$

denotes the finite set of strings obtained by "interleaving" in all possible ways AaA with Bb or with Cc, that is the set $BbAaA$, $BAbaA$, $BAabA$, $BAaAb$, $ABbaA$, $ABabA, \ldots$, $AaABb$, $CcAaA$, $CAcaA$, $CAacA, \ldots$, $AaACc$.

Definition 1 *Let V be a finite alphabet and let $\mathcal{E}$ be a symbol not in V. A* **poms-expression** *over V is a string π satisfying one of the following conditions:*

(i) *$\pi = X$, with $X \in V \cup \{\mathcal{E}\}$;*

(ii) *$\pi = +(\pi_1, \pi_2, \ldots, \pi_n)$, with $n \ge 2$ and π_i poms-expressions for each i, $1 \le i \le n$;*

(iii) *$\pi = \|(\pi_1, \pi_2, \ldots, \pi_n)$, with $n \ge 2$ and π_i poms-expressions for each i, $1 \le i \le n$;*

(iv) *$\pi = \pi_1 \cdot \pi_2$, with π_1 and π_2 poms-expressions.*

We will write binary uses of $+$ and $\|$ with infix notation under a precedence ordering (from highest to lowest) of $\cdot$, $\|$, $+$, as in the example (2). We take the infix operators to be right associative, though in all of the definitions in this paper, disambiguation of associativity is not relevant and can be taken arbitrarily.

We can define the language of a poms-expression by induction on its structure.

Definition 2 *Let π be a poms-expression over V. The* **language of** *π, a set $\sigma(\pi) \subseteq V^*$ is defined as follows:*

(i) *$\sigma(X) = \{X\}$ and $\sigma(\mathcal{E}) = \{\varepsilon\}$*

(ii) *$\sigma(+(\pi_1, \pi_2, \ldots, \pi_n)) = \cup_{i=1}^{n} \sigma(\pi_i)$*

(iii) *$\sigma(\|(\pi_1, \pi_2, \ldots, \pi_n)) = \mathsf{shuffle}_{i=1}^{n} \sigma(\pi_i)$*

(iv) $\sigma(\pi \cdot \pi') = \sigma(\pi) \cdot \sigma(\pi')$

Note that $\sigma(\pi)$ is always a finite set.

We are now ready to introduce the central notion of this paper, the partially ordered multi-set context-free grammar.

Definition 3 *A* **partially ordered multi-set context-free grammar** *(poms-CFG) is a tuple $G = (V_N, V_T, S, P)$, where V_N, V_T, and $S \in V_N$ are defined as for standard CFGs, and P is a finite set of productions of the form $A \to \pi$, with $A \in V_N$ and π a poms-expression over $V = V_N \cup V_T$.*

Each poms-CFG G can be naturally associated with an equivalent CFG $\sigma(G) = (V_N, V_T, S, \sigma(P))$, where $\sigma(P) = \{A \to \alpha \mid (A \to \pi) \in P,\ \alpha \in \sigma(\pi)\}$. That is, $\sigma(G)$ includes all and only those context-free productions that can be obtained by replacing the poms-expression in the right-hand side of a production of G with one of the strings denoted by that poms-expression. In this way we can define a derive relation for G by letting $\alpha \Rightarrow_G \beta$ whenever $\alpha \Rightarrow_{\sigma(G)} \beta$, $\alpha, \beta \in V^*$, and obtain $L(G) = L(\sigma(G))$.

We could parse an input string $w \in V_T^*$ with a poms-CFG G by first replacing G with its equivalent CFG $\sigma(G)$ and then applying a standard CFG parsing algorithm on $\sigma(G)$ and w. This might unnecessarily expand the space requirements for the grammar by adding productions that might not be relevant for the specific input string at hand. The alternative approach followed here is to parse w using grammar G directly, which in turn requires that we be able to process poms-expressions somehow, a topic to which we now turn.

4 Poms-Automata

Although poms-expressions may be easily constructed by linguists, they do not allow a direct algorithmic interpretation for efficient recognition of strings. We therefore define a lower level automata-theoretic representation, which we call poms-automata.

Definition 4 *A* **poms-automaton** *is a tuple (Q, V, δ, q_s, q_e), where Q is a finite set of states, V is a finite alphabet, $q_s, q_e \in Q$ are special states called* **start** *and* **end** *states, respectively, and δ is a transition relation containing triples of the form (q, X, q') with $q, q' \in Q$ and $X \in V \cup \{\varepsilon, \vdash, \dashv\}$.*

Symbol ε indicates that a transition does not consume any of the input symbols. Symbols $\vdash$ and $\dashv$ have the same meaning, but they additionally encode that the transition starts or ends, respectively, sub-automata encoding poms-expressions headed by an interleave operator.

For any poms-expression, we can construct a poms-automaton that expresses the same language. Below, we give a function μ mapping poms-expressions to poms-automata, providing a semantics for these automata afterwards and showing equivalence.

Definition 5 *Let π be a poms-expression over V. The* **corresponding poms-automaton** *$\mu(\pi) = (Q, V, \delta, q_s, q_e)$ is specified as follows:*

(i) *If $\pi = X$, $X \in V \cup \{\mathcal{E}\}$, we have*

  (a) *$Q = \{q_s, q_e\}$, q_s, q_e new states,*

  (b) *$\delta = \{(q_s, X, q_e)\}$ if $X \in V$ and $\delta = \{(q_s, \varepsilon, q_e)\}$ if $X = \mathcal{E}$;*

(ii) *if $\pi = +(\pi_1, \pi_2, \ldots, \pi_n)$ with $\mu(\pi_i) = (Q_i, V, \delta_i, q_{i,s}, q_{i,e})$, $1 \leq i \leq n$, we have*

  (a) *$Q = \cup_{i=1}^{n} Q_i \cup \{q_s, q_e\}$, q_s, q_e new states,*

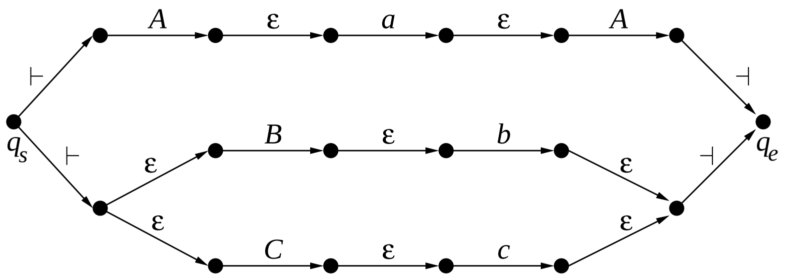

Figure 1: The corresponding poms-automaton for the poms-expression in (2).

(b) $\delta = \cup_{i=1}^{n} \delta_i \cup \{(q_s, \varepsilon, q_{i,s}) \mid 1 \leq i \leq n\} \cup \{(q_{i,e}, \varepsilon, q_e) \mid 1 \leq i \leq n\}$;

(iii) *if* $\pi = \|(\pi_1, \pi_2, \ldots, \pi_n)$ *with* $\mu(\pi_i) = (Q_i, V, \delta_i, q_{i,s}, q_{i,e})$, $1 \leq i \leq n$, *we have*

(a) $Q = \cup_{i=1}^{n} Q_i \cup \{q_s, q_e\}$, q_s, q_e *new states,*

(b) $\delta = \cup_{i=1}^{n} \delta_i \cup \{(q_s, \vdash, q_{i,s}) \mid 1 \leq i \leq n\} \cup \{(q_{i,e}, \dashv, q_e) \mid 1 \leq i \leq n\}$;

(iv) *if* $\pi = \pi_1 \cdot \pi_2$ *with* $\mu(\pi_i) = (Q_i, V, \delta_i, q_{i,s}, q_{i,e})$, $i \in \{1, 2\}$, *we have*

(a) $Q = Q_1 \cup Q_2$, *with* $q_s = q_{1,s}$ *and* $q_e = q_{2,e}$,

(b) $\delta = \delta_1 \cup \delta_2 \cup \{(q_{1,e}, \varepsilon, q_{2,s})\}$.

Figure 1 presents the corresponding poms-automaton for the poms-expression from our running example (2).

In order to provide a formal definition of the language specified by a poms-automaton, we require a way of encapsulating the current state of traversal of the automaton. For deterministic finite-state automata, the automaton state itself can serve this purpose. For nondeterministic automata, however, a traversal may leave the automaton nondeterministically in one of a set of states. (For this reason, the subset construction for converting nondeterministic to deterministic finite-state automata, for example, uses sets of states in the nondeterministic automaton to capture the state of its traversal.) The corresponding notion for poms-automata is the notion of a *cut* through a poms-automaton. This notion will play a central role in the development of our parsing algorithm in the next section.

We start by fixing some poms-expression π and let $\mu = \mu(\pi) = (Q, V, \delta, q_s, q_e)$ be the corresponding poms-automaton. Intuitively speaking, a cut through μ is a set of vertices that we might reach when traversing the graph from the initial vertex toward the end vertex, in an attempt to produce a string in $\sigma(\pi)$, following the different branches as prescribed by the encoded disjunction and interleave operators.

Some additional notation will be useful. Given three subsets of Q, namely, c, $\{q_1, \ldots, q_m\} \subseteq c$, and $\{q'_1, \ldots, q'_n\}$, we write $c[q_1, \ldots, q_m / q'_1, \ldots, q'_n]$ to denote the set $(c - \{q_1, \ldots, q_m\}) \cup \{q'_1, \ldots, q'_n\}$. Informally speaking, this is the set obtained by replacing in c states from before the slash with states from after.

A relation $\mathsf{goto}_\mu \subseteq (2^Q \times (V \cup \{\varepsilon\}) \times 2^Q)$ is defined by the following conditions:

- $(c, X, c[q/q']) \in \mathsf{goto}_\mu$ if $q \in c$ and $(q, X, q') \in \delta$ for $X \in V \cup \{\varepsilon\}$;
- $(c, \varepsilon, c[q/q'_1, \ldots, q'_n]) \in \mathsf{goto}_\mu$ if $q \in c$ and $(q, \vdash, q'_i)$, $1 \leq i \leq n$, are all the transitions in δ with q as first component;

- $(c, \varepsilon, c[q_1, \ldots, q_n/q]) \in \mathsf{goto}_\mu$ if $\{q_1, \ldots, q_n\} \subseteq c$ and $(q_i, \dashv, q)$, $1 \leq i \leq n$, are all the transitions in δ with q as last component.

The goto_μ relation simulates nondeterministic one-step moves over μ. The first item above refers to moves that follow a single transition in the automaton, labeled by a symbol from the alphabet or by the empty string. This move is exploited, for example, when visiting a state at the start of a sub-automaton that encodes a poms-expression headed by the disjunction operator. In this case there are several possible transitions, but at most one may be chosen. The second item above refers to moves that simultaneously follow all transitions emanating from the state at hand. This is used when visiting a state at the start of a sub-automaton that encodes a poms-expression headed by the interleave operator. In this case, all possible subexpressions of that operator must be evaluated in parallel. Finally, the third item refers to a move that can be read as the complement of the previous type of move. Here we complete the visit of a sub-automaton that encodes a poms-expression headed by the interleave operator. This can be done only if the evaluations of the subexpressions of that operator have all come to an end.

We are now ready to define the notion of cut.

Definition 6 *Let π be a poms-expression over V, and let $\mu = \mu(\pi) = (Q, V, \delta, q_s, q_e)$. The set of all* **cuts** *of μ, written $\mathsf{cut}(\mu)$, is the smallest subset of 2^Q satisfying the following conditions:*

(i) $\{q_s\} \in \mathsf{cut}(\mu)$, *and*

(ii) $c \in \mathsf{cut}(\mu)$ *if* $c' \in \mathsf{cut}(\mu)$ *and* $(c', X, c) \in \mathsf{goto}_\mu$ *for some* $X \in V \cup \{\varepsilon\}$.

We can informally interpret a cut $c = \{q_1, \ldots, q_k\} \in \mathsf{cut}(\mu)$ as follows. In the attempt to generate a string in $L(\pi)$, we traverse several paths in the poms-automaton μ. This corresponds to the "parallel" evaluation of some of the sub-expressions of π, and each $q_i \in c$ refers to one specific such subexpression. Thus, k provides the number of evaluations that we are carrying out in parallel at the point of the computation represented by the cut. Note however that, when drawing a straight line across a planar representation of a poms-automaton, separating the start state from the end state, the set of states that we can identify is not necessarily a cut.[1] In fact, as we have already explained when discussing function goto_μ, only one path is followed when processing a state encoding a disjunction operator.

With the notion of cut defined, we can define the language of a poms-automaton.

Definition 7 *Let $c \in \mathsf{cut}(\mu)$ and $\alpha \in V^*$. We define the* **language of a cut** *$L(c)$ as follows: $\alpha \in L(c)$ if and only if there exists $k \geq |\alpha|$, $X_i \in V \cup \{\varepsilon\}$, $1 \leq i \leq k$, and $c_i \in \mathsf{cut}(\mu)$, $0 \leq i \leq k$, such that $X_1 \cdot \cdots \cdot X_k = \alpha$, $c_0 = \{q_s\}$, $c_k = c$ and $(c_{i-1}, X_i, c_i) \in \mathsf{goto}_\mu$ for $1 \leq i \leq k$.*

The **language of the poms-automaton** *μ, written $L(\mu)$, is $L(\{q_e\})$.*

We have that $L(\mu(\pi)) = L(\pi)$, that is, the μ function constructs an automaton generating the same language. The proof is rather long and does not add much to the already intuitive ideas underlying the definitions in this section; therefore we will omit it.

Henceforth, we will abuse notation by using π for $\mu(\pi)$ where no confusion results, for example, writing $\mathsf{cut}(\pi)$ for $\mathsf{cut}(\mu(\pi))$ or goto_π for $\mathsf{goto}_{\mu(\pi)}$.

[1] The pictorial representation mentioned above comes close to a different definition of cut that is standard in the literature on graph theory and operating research. The reader should be aware that the standard graph-theoretic notion of cut is different from the one introduced in this paper.

$$\frac{}{[\{q_s\},0,0]} \left\{ \begin{array}{l} (S \rightarrow \pi) \in P, \\ q_s \text{ start state of } \mu(\pi) \end{array} \right. \tag{3}$$

$$\frac{[c,i,j]}{[\{q_s\},j,j]} \left\{ \begin{array}{l} (c,B,c') \in \mathsf{goto} \text{ for some } c', \\ (B \rightarrow \pi) \in P, \\ q_s \text{ start state of } \mu(\pi) \end{array} \right. \tag{4}$$

$$\frac{[c,i,j]}{[c',i,j+1]} \left\{ \; (c,a_{j+1},c') \in \mathsf{goto} \right. \tag{5}$$

$$\frac{[c,i,j]}{[c',i,j]} \left\{ \; (c,\varepsilon,c') \in \mathsf{goto} \right. \tag{6}$$

$$\frac{[\{q_f\},i,j]}{[A,i,j]} \left\{ \begin{array}{l} (A \rightarrow \pi) \in P, \\ q_f \text{ final state of } \mu(\pi) \end{array} \right. \tag{7}$$

$$\frac{[c,i,k] \; [B,k,j]}{[c',i,j]} \left\{ \; (c,B,c') \in \mathsf{goto} \right. \tag{8}$$

Figure 2: An abstract specification of the parsing algorithm for poms-CFG; we assume universal quantification on all variables, unless otherwise stated.

5 Parsing of poms-CFGs

In this section we develop a tabular algorithm to parse an input string $w = a_1 \cdots a_n$, $a_i \in V_T$, according to a given poms-CFG $G = (V_N, V_T, P, S)$. The algorithm is an adaptation of the well-known Earley algorithm for CFG parsing (Earley, 1970); we also use some of the optimizations presented by Graham et al. (1980). The algorithm is presented as a set of inference rules in the style of Shieber et al. (1995).

Partial results obtained in the parsing process are recorded through items of the following two forms:

- $[A,i,j]$, $A \in V_N$ and $0 \leq i \leq j \leq n$, related to derivations $A \Rightarrow_G^* w_{i,j}$;
- $[c,i,j]$, $c \in \mathsf{cut}(\pi)$ for some $(A \rightarrow \pi) \in P$ and $0 \leq i \leq j \leq n$, related to derivations $\alpha \Rightarrow_G^* w_{i,j}$ for some $\alpha \in L(c)$.

As in the original Earley algorithm, our algorithm constructs the items above only in case the related derivations can be embedded in larger derivations in G starting from S and deriving strings with prefix $w_{0,i}$.

Let $\mathsf{goto} = \cup_{(A \rightarrow \pi) \in P} \, \mathsf{goto}_\pi$. (The states in the various $\mu(\pi)$ will be assumed disjoint so that the union is disjoint as well.) The specification of our algorithm makes use of this goto relation, but this does not necessarily mean that the relation must be fully computed before invoking the algorithm. We can instead compute elements of goto "on the fly" when we visit a cut for the first time, and cache these elements for possible later use. This and other implementation issues will be addressed in the next section. Figure 2 presents an abstract specification of the algorithm.

Steps (3), (4), and (5) closely resemble the initialization, predictor, and scanner steps, respectively, from the original Earley algorithm. In addition, Step (6) is used for scanning transitions of poms-automata where no symbol from V is consumed. Finally, the completer step from the original Earley algorithm has been broken up into Steps (7) and (8) for efficiency reasons. The algorithm accepts w if and only if item $[S,0,n]$ can be deduced.

As a final technical remark, we observe that a poms-CFG can be cast into a normal form where there is only one production with a given nonterminal in its left-hand side. This is easily done using the disjunction operator as defined for poms-expressions. When such a normal form is adopted, then Steps (7) and (8) in Figure 2 can be collapsed in a single step in the obvious way.

6 Implementation and complexity

We develop a complexity analysis of our parsing algorithm for poms-CFGs.

In order to provide a more articulated worst-case bound on the complexity of parsing, we introduce the **width** of a poms-expression π, the size of the largest cut in the corresponding automaton.

$$\mathsf{width}(\pi) = \max_{c \in \mathsf{cut}(\pi)} |c|$$

As observed in Section 4, this is the maximum number of parallel evaluations that we need to carry out when processing poms-automaton $\mu(\pi)$. The quantity $\mathsf{width}(\pi)$ can be easily computed as follows:

$$\begin{array}{rcl} \mathsf{width}(X) & = & 1 \quad \text{for } X \in V \cup \{\mathcal{E}\} \\ \mathsf{width}(+(\pi_1, \ldots, \pi_n)) & = & \max_j \ \mathsf{width}(\pi_j) \\ \mathsf{width}(\|(\pi_1, \ldots, \pi_n)) & = & \sum_j \ \mathsf{width}(\pi_j) \\ \mathsf{width}(\pi_1 \cdot \pi_2) & = & \max\{\mathsf{width}(\pi_1), \, \mathsf{width}(\pi_2)\} \end{array}$$

As suggested in Section 5, we do not need to compute the goto relation before processing the input string. We instead adopt a lazy approach and compute cuts and elements of each goto_π on demand, during the execution of the parsing algorithm.

Let π be a poms-expression from G and let Q be the state set of the corresponding poms-automaton $\mu(\pi)$. We can represent cuts in $\mathsf{cut}(\pi)$ as binary strings (bit vectors) of length $|Q|$, or alternatively as strings of length $\mathsf{width}(\pi)$ over alphabet Q, assuming some canonical ordering of Q. The first solution might be convenient when set Q is not too large. The second solution should be adopted when Q is large or when $\mathsf{width}(\pi)$ is much smaller than $|Q|$, to avoid sparseness. In both solutions, cuts can be stored in and retrieved from a trie data structure C (Gusfield, 1997). Elements $(c, X, c') \in \mathsf{goto}_\pi$ can then be encoded using pointers to the leaves of C that represent c and c'.

On-the-fly construction of relations goto_π can be carried out in the following way. Whenever, for some c and X, elements $(c, X, c') \in \mathsf{goto}_\pi$ need to be used but have not been computed before, we apply the definition of goto_π for c and X, and cache the corresponding elements (c, X, c') for possible later use. The cuts c' obtained in this way that were never computed before are also stored in C.

Items $[A, i, j]$ and $[c, i, j]$ can be stored in an $(n+1) \times (n+1)$ square matrix T, as in the case of the standard Earley algorithm. Each entry of T contains nonterminals and cuts, the latter encoded as pointers to some leaf in C.

We now turn to the worst case time complexity for our algorithm. Let $G = (V_N, V_T, P, S)$ be the input poms-CFG and let $w = a_1 \cdots a_n$ be the input string. For $(A \rightarrow \pi) \in P$, let also $\mu(\pi) = (Q_\pi, V, \delta_\pi, q_{\pi,s}, q_{\pi,e})$. It is not difficult to show that $|\delta_\pi| = \mathcal{O}(|Q_\pi|)$. Since the number of occurrences in π of symbols from $V \cup \{\mathcal{E}\} \cup \{+, \|, \cdot\}$ is also proportional to $|Q_\pi|$, in what follows we will take $\mathcal{O}(|Q_\pi|)$ as a bound on the size of any reasonable encoding of poms-expression π.

We can also show that

$$|\mathsf{goto}_\pi| \quad \leq \quad |\mathsf{cut}(\pi)| \cdot |\delta_\pi| \,, \tag{9}$$

since for each $c \in \mathsf{cut}(\pi)$ we have $|\{(c, X, c') \mid (c, X, c') \in \mathsf{goto}_\pi\}| \leq |\delta_\pi|$.

The quantity $|\mathsf{cut}(\pi)|$ is obviously bounded from above by $|Q_\pi|^{\mathsf{width}(\pi)}$. The following lemma, whose proof is reported in Appendix A, states a tighter upper bound on $|\mathsf{cut}(\pi)|$, which will be used in our complexity analysis below.

Lemma 1 *Let π be a poms-expression over V and let Q_π be the state set of poms-automaton $\mu(\pi)$. We have* $|\mathsf{cut}(\pi)| \leq \left(\frac{|Q_\pi|}{\mathsf{width}(\pi)}\right)^{\mathsf{width}(\pi)}$.

Now consider Step 8 in the algorithm of Figure 2. For each production $(A \to \pi) \in P$, the number of possible executions of this step is bounded from above by $|\mathsf{goto}_\pi| \cdot n^3$, since there are $|\mathsf{goto}_\pi|$ relevant elements in goto and no more than n^3 choices for i, k, and j. We can access and store each cut in time $\mathcal{O}(\mathsf{width}(\pi))$, if cuts are encoded as strings of length $\mathsf{width}(\pi)$. Then the total amount of time taken by the executions of Step 8 for production $A \to \pi$ is $\mathcal{O}(\mathsf{width}(\pi) \cdot |\mathsf{goto}_\pi| \cdot n^3)$, or $\mathcal{O}(\mathsf{width}(\pi) \cdot |\mathsf{cut}(\pi)| \cdot |\delta_\pi| \cdot n^3)$ using (9).

Recall that $|\delta_\pi| = \mathcal{O}(|Q_\pi|)$, and let $q = \max_{(A\to\pi)\in P} |Q_\pi|$ and $k = \max_{(A\to\pi)\in P} \mathsf{width}(\pi)$ be respectively the maximal size and width of the poms-automata in the grammar. Then the time bound can be simplified to $\mathcal{O}(k \cdot |\mathsf{cut}(\pi)| \cdot q \cdot n^3)$. All that remains is bounding $|\mathsf{cut}(\pi)|$. Lemma 1 provides a bound on the number of cuts for a single automaton. We require a bound over all π, so let

$$g = \max_{(A\to\pi)\in P} \left(\frac{|Q_\pi|}{\mathsf{width}(\pi)}\right)^{\mathsf{width}(\pi)} \qquad .$$

Given this bound, the total time taken in the worst case by Step 8 is

$$\mathcal{O}(|P| \cdot k \cdot g \cdot q \cdot n^3) \qquad .$$

It is not difficult to show that this upper bound also holds for the execution of all other steps of the algorithm, including the on-the-fly construction of all relations goto_π.

We would like to characterize g in terms not of the sizes and widths of all of the π, but over some bounds thereon. We observe that the function $(n/x)^x$ is monotonically increasing for increasing real values of x in the interval $(0, n/e]$.[2] Now let q and k be defined as above. By construction, $k < q/2$. Thus, by taking the numerator in Lemma 1 to be not the maximum state size, but $\frac{e}{2} \cdot q$, the lemma still holds, and using all of the above observations we can write

$$\max_{(A\to\pi)\in P} |\mathsf{cut}(\pi)| \leq \max_{(A\to\pi)\in P} \left(\frac{q}{\mathsf{width}(\pi)}\right)^{\mathsf{width}(\pi)} < \max_{(A\to\pi)\in P} \left(\frac{\frac{e}{2}q}{\mathsf{width}(\pi)}\right)^{\mathsf{width}(\pi)} \leq \left(\frac{\frac{e}{2}q}{k}\right)^k \qquad .$$

Then the total time taken in the worst case by Step 8 is

$$\mathcal{O}(|P| \cdot k \cdot \left(\frac{\frac{e}{2}q}{k}\right)^k \cdot q \cdot n^3) \qquad ,$$

which is polynomial in grammar size for bounded k.

We can compare the above result with the time complexity of the Earley algorithm, reported

[2] This can easily be seen from its first derivative in x, $(n/x)^x \cdot (\ln(n/x) - 1)$. The factor $(n/x)^x$ is positive for all positive n and x; the factor $(\ln(n/x) - 1)$ is non-negative in the interval $(0, n/e]$, with a zero for $x = n/e$.

as $\mathcal{O}(|G| \cdot n^3)$ by Graham et al. (1980). Observe that the factor $|P| \cdot q$ in our bound above can be taken to represent the size of the input poms-CFG. Thus $\mathcal{O}(|G| \cdot n^3)$ in the Earley bound is comparable with $\mathcal{O}(|P| \cdot q \cdot n^3)$ in the present bound. The additional term $k \cdot g$ or $k \cdot \left(\frac{\frac{e}{2}q}{k}\right)^k$ accounts for the structural complexity of the worst case poms-expression in the poms-CFG. When some poms-expression π does not have any linear precedence constraint nor any disjunction, we can have a worst case with a pure multi-set encoded by a single interleave operator with $k = \frac{q}{2} - 1$ arguments from V. Then $\mathcal{O}(\left(\frac{\frac{e}{2}q}{k}\right)^k)$ can be written as $\mathcal{O}(c^q)$ for some constant c, giving rise to an exponential upper bound in the size of the longest poms-expression in the grammar. This comes as no surprise, since the recognition problem for pure multi-set CFGs is NP-complete, as already discussed. However, in practical natural-language applications, parameter k should be bounded by a quite small constant. In this case our algorithm runs effectively in polynomial time, with an asymptotical behavior much closer to the Earley algorithm.

7 Discussion

Applying the results above to the particular case of bounds on ID/LP grammars is straightforward. ID/LP format (Gazdar et al., 1985) is essentially a context-free formalism in which the multiset of right-hand side elements (provided by the immediate dominance (ID) rules) are ordered by an explicitly provided partial order (stated with linear precedence (LP) rules). Thus, any ID/LP grammar is trivially stated as a poms-CFG. As an example, we consider the grammar fragment of verb phrases in the free-word-order Makua language (Gazdar et al., 1985, page 48).

$$
\begin{array}{lll@{\qquad\qquad}lll}
VP & \rightarrow & V & V & \prec & S \\
VP & \rightarrow & V, NP & & & \\
VP & \rightarrow & V, S & & & \\
VP & \rightarrow & V, NP, NP & & & \\
VP & \rightarrow & V, NP, PP & & & \\
VP & \rightarrow & V, NP, S & & &
\end{array}
$$

The six immediate dominance rules are constrained by the single linear precedence rule. These rules, stated in the poms-CFG form, would be

			Width	*State size*
VP	$\rightarrow$	V	1	2
VP	$\rightarrow$	$V \parallel NP$	2	6
VP	$\rightarrow$	$V \cdot S$	1	4
VP	$\rightarrow$	$V \parallel NP \parallel NP$	3	8
VP	$\rightarrow$	$V \parallel NP \parallel PP$	3	8
VP	$\rightarrow$	$V \cdot S \parallel NP$	2	8

(For reference, each rule is followed by its width and state size.) The maximum value g is obtained for rules of width 3 and state size 8. We can therefore bound parsing time by an extra constant factor (beyond $\mathcal{O}(|G| \cdot n^3)$) of $3 \cdot (8/3)^3 \approx 57$. (The coarser-grained analysis in terms just of q and k gives us a bound of $3 \cdot (\frac{e}{2} \cdot 8/3)^3 \approx 143$.) The general point is clear: Typical grammars in ID/LP format — even those for languages making heavy use of the free-word order aspects of the formalism, and therefore exhibiting large widths, relatively speaking — can be analyzed by this method and seen to have small widths in absolute terms; they are therefore readily parsable by the direct parsing method we present here.

The formalism that we have presented was inspired, as noted in the introduction, by IDL-expressions (Nederhof and Satta, 2002), and could be generalized to allow full IDL-expressions on the right-hand side of productions. Such grammars, which we might dub *IDL-grammars*, would have yet more expressive power, though remaining weakly context-free equivalent. We did not do so because the extra expressivity of IDL-expressions, namely, the lock operator, was not needed for our purposes in explicating bounds on ID/LP grammars. Indeed, the ability to embed subphrases that context-free productions provide can serve the same purpose as the lock operator. Thus the difference in expressivity between poms-CFGs and IDL grammars is only in the tree languages that they specify. The same type of analysis based on the width of corresponding IDL automata could provide fine-grained bounds on the parsing of grammars expressed in that formalism as well.

References

Barton, G. Edward, Jr. 1985. The computational difficulty of ID/LP parsing. In *Proceedings of the 23rd annual meeting of the association for computational linguistics*, 76–81. Chicago, IL.

Barton, G. Edward, Jr., Robert Berwick, and Eric Sven Ristad. 1987. The complexity of ID/LP parsing. In *Computational complexity and natural language*, chap. 7, 187–213. Cambridge, MA: The MIT Press.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 13(2): 94–102.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag. 1985. *Generalized phrase structure grammar*. Oxford, England: Basil Blackwell.

Gischer, J.L. 1988. The equational theory of pomsets. *Theoretical Computer Science* 61:199–224.

Graham, S.L., M.A. Harrison, and W.L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems* 2(3):415–462.

Gusfield, D. 1997. *Algorithms on strings, trees and sequences*. Cambridge, UK: Cambridge University Press.

Hopcroft, J.E., and J.D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Lalonde, Wilf R. 1977. Regular right part grammars and their parsers. *Communications of the Association for Computing Machinery* 20(10):731–741.

Nederhof, Mark-Jan, and Giorgio Satta. 2002. IDL-expressions: A compact representation for finite languages in generation systems. In *Proceedings of the 7th conference on formal grammar*, 125–136. Trento, Italy.

Shieber, S.M., Y. Schabes, and F.C.N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming* 24:3–36.

Shieber, Stuart M. 1984. Direct parsing of ID/LP grammars. *Linguistics and Philosophy* 7(2): 135–154.

A Upper bound on $\mathsf{cut}(\pi)$

For space reasons, we only provide here an outline of the proof of Lemma 1. The result is a particularization of a more general result proved by Nederhof and Satta (2002) for a representation of finite languages that embeds poms-expressions.

Lemma 1 *Let π be a poms-expression over V and let Q_π be the state set of poms-automaton $\mu(\pi)$. We have* $|\mathsf{cut}(\pi)| \leq \left(\frac{|Q_\pi|}{\mathsf{width}(\pi)}\right)^{\mathsf{width}(\pi)}$.

Outline of the proof. We use below the following inequality. For any real values $x_i > 0$, $1 \leq i \leq h$, $h \geq 2$, we have $\Pi_{i=1}^{h} x_i \leq \left(\frac{\sum_{i=1}^{h} x_i}{h}\right)^h$. This amounts to say that the geometric mean is never larger than the arithmetic mean.

Let $k = \mathsf{width}(\pi)$. We need to prove the following claim. We can partition Q_π into subsets $Q_\pi[j]$, $1 \leq j \leq k$, such that for every $c \in \mathsf{cut}(\pi)$ and for every $q_1, q_2 \in c$, q_1 and q_2 do not belong to the same subset $Q_\pi[j]$. We use induction on $\#_p(\pi)$, the number of operator occurrences in π.

Base: $\#_p(\pi) = 0$. Then $\pi = X$, with $X \in V \cup \{\mathcal{E}\}$. We have $k = 1$ and we can set $Q_\pi[1] = Q_\pi$, since $\mathsf{cut}(\pi) = \{\{q_s\}, \{q_e\}\}$.

Induction: $\#_p(\pi) > 0$. We distinguish three cases.

Case 1: $\pi = +(\pi_1, \pi_2, \ldots, \pi_n)$. We have $\mathsf{cut}(\pi) = (\cup_{i=1}^{n} \mathsf{cut}(\pi_i)) \cup \{\{q_s\}, \{q_e\}\}$. We also have $\mathsf{width}(\pi) = \max_{i=1}^{n} \mathsf{width}(\pi_i)$ (see Section 6). If we define $Q_{\pi_i}[j] = \emptyset$ for $j > \mathsf{width}(\pi_i)$, we can set $Q_\pi[1] = (\cup_{i=1}^{n} Q_{\pi_i}[1]) \cup \{\{q_s\}, \{q_e\}\}$, and $Q_\pi[j] = \cup_{i=1}^{n} Q_{\pi_i}[j]$ for $2 \leq j \leq \mathsf{width}(\pi)$.

Case 2: $\pi = \pi_1 \cdot \pi_2$. The proof is almost identical to that of Case 1, with $n = 2$.

Case 3: $\pi = \|(\pi_1, \pi_2, \ldots, \pi_n)$. We have $\mathsf{cut}(\pi) = \{\cup_{i=1}^{n} c_i \mid c_i \in \mathsf{cut}(\pi_i),\ 1 \leq i \leq n\} \cup \{\{q_s\}, \{q_e\}\}$. We also have $\mathsf{width}(\pi) = \sum_{i=1}^{n} \mathsf{width}(\pi_i)$. We then set $Q_\pi[1] = Q_{\pi_1}[1] \cup \{\{q_s\}, \{q_e\}\}$, $Q_\pi[j] = Q_{\pi_1}[j]$ for $2 \leq j \leq \mathsf{width}(\pi_1)$, $Q_\pi[j + \mathsf{width}(\pi_1)] = Q_{\pi_2}[j]$ for $1 \leq j \leq \mathsf{width}(\pi_2)$, and so forth.

With all of the above relations we can then write

$$|\mathsf{cut}(\pi)| \leq \Pi_{j=1}^{k} |Q_\pi[j]| \leq \left(\frac{\sum_{j=1}^{k} |Q_\pi[j]|}{k}\right)^k = \left(\frac{|Q_\pi|}{k}\right)^k .$$