Practical Parsing of Parallel Multiple Context-Free Grammars

Peter Ljunglöf
Department of Computer Science and Engineering
University of Gothenburg and Chalmers University of Technology
Gothenburg, Sweden
peter.ljunglof@gu.se

Abstract

We discuss four previously published parsing algorithms for parallell multiple context-free grammar (PMCFG), and argue that they are similar to each other, and implement an Earley-style top-down algorithm. Starting from one of these algorithms, we derive three modifications – one bottom-up and two variants using a left corner filter. An evaluation shows that substantial improvements can be made by using the algorithm that performs best on a given grammar. The algorithms are implemented in Python and released under an open-source licence.

We start by introducing the necessary concepts. Then we discuss four previously published PMCFG algorithms, and argue that they are similar. We take Angelov (2009) as a starting point for introducing three new parsing strategies. Finally we discuss various optimizations of the parsing strategies and give a small evaluation.

1 Background

1.1 PMCFG

Let Σ and N be sets of terminal and nonterminal symobls, respectively. A *parallel multiple context-free grammar* (PMCFG) (Seki et al., 1991) consists of a set of context-free production rules $A \to f(\vec{B})$, where $A \in N$ and $\vec{B} = B_1, \ldots, B_n \in N$ are nonterminals. f is a *linearization* function:

$$f \quad : \quad (\Sigma^*)^{\delta(B_1)} \times \cdots \times (\Sigma^*)^{\delta(B_n)} \to (\Sigma^*)^{\delta(A)}$$

where $\delta(X)$ is the *fan-out*, or *dimension*, of the non-terminal X. The linearization function f is normally written like this:

$$f\left(\left\langle x_{1,1} \ldots x_{1,\delta(B_1)}\right\rangle, \ldots, \left\langle x_{n,1} \ldots x_{n,\delta(B_n)}\right\rangle\right) = \left\langle \alpha_1, \ldots, \alpha_{\delta(A)}\right\rangle$$

where each α_i is sequence of terminal symbols and bound variables. However, in this paper we write the linearizations in variable-free form, where each bound variable $x_{d,r}$ is written as a pair of the form $\langle d.r\rangle$. We use $f.s$ to denote the sth constituent of the linearization, i.e. α_s.

Figure 1 contains an example grammar recognizing the language $a^n b^n c^n d^n$, and its variable-free form is shown in figure 2. The fanouts of this grammar are $\delta(S) = 1$ and $\delta(A) = 2$.

$S \to f(A)$	$f(\langle x, y\rangle) = \langle x\, y\rangle$
$A \to g(A)$	$g(\langle x, y\rangle) = \langle a\, x\, b,\ c\, y\, d\rangle$
$A \to h()$	$h() = \langle a\, b,\ c\, d\rangle$

Figure 1: A grammar that recognizes the langauge $\{a^n b^n c^n d^n \mid n > 0\}$.

$S \to f(A)$	$f.1 = \langle 1.1\rangle\, \langle 1.2\rangle$	
$A \to g(A)$	$g.1 = a\, \langle 1.1\rangle\, b$	$g.2 = c\, \langle 1.2\rangle\, d$
$A \to h()$	$h.1 = a\, b$	$h.2 = c\, d$

Figure 2: The same grammar in variable-free form.

1.2 MCFG and LCFRS

A linearization function is *linear* if no argument constituent $\langle d.r\rangle$ occurs more than once in the right-hand side. It is *non-erasing* if all possible argument constituents occurs in the right-hand side.

A *multiple context-free grammar* (MCFG) is a PMCFG where all linearization functions are linear. A *linear context-free rewriting system* (LCFRS) (Vijay-Shanker et al., 1987) is a linear and non-erasing PMCFG. Erasingness does not add to the expressive power, and therefore LCFRS and MCFG are weakly equivalent. However, reduplication does give extra expressive power, so PMCFG is a proper extension of MCFG/LCFRS (Seki et al., 1991).

1.3 Emptiness and left corners

We define the *context-free approximation* of a PMCFG rule $A \to f(\vec{B})$ to be $\delta(A)$ context-free rules $A.r \to \beta_r$, where β_r is the linearization $f.r$ with every occurrence of $\langle d.s \rangle$ replaced by $B_d.s$. The nonterminals of the context-free approximation are of the form $A.r$.

We define $(\Rightarrow)$ as the standard reflexive and transitive rewriting relation on the context-free approximation. We say that a constituent $A.r$ is *empty* if $A.r \Rightarrow \epsilon$. We define the *left corner* relation as: $A.r \rhd x$ iff $A.r \Rightarrow x\,\beta$ for some β, where x is either a terminal w or a constituent $B.s$.

1.4 Non-empty grammars

Our algorithms can handle all kinds of PMCFG grammars, but the two bottom-up variants work much better if the grammar has no empty linearizations, i.e., if $A.r \not\Rightarrow \epsilon$ for all constituents $A.r$. This is discussed further in section 4.3.

All grammars with empty linearizations can be transformed to nonempty grammars (Seki et al., 1991). We use an adaptation of an algorithm for context-free grammars. The transformation does not lose any important information, and it is straightforward to translate parse trees back into the original grammar efficiently.

2 Existing parsing algorithms

Most existing parsing algorithms require that the grammar is a LCFRS or a subclass thereof (Burden and Ljunglöf, 2005; de la Clergerie, 2002; Gómez-Rodríguez et al., 2008; Kallmeyer, 2010; Kallmeyer and Maier, 2009; Kanazawa, 2008), often in some kind of normal form such as a *binarized* or an *ordered* LCFRS. There are some algorithms that can handle general PMCFG grammars (Angelov, 2009; Boullier, 2004; Ljunglöf, 2004), including the algorithms presented in this paper. This is important when parsing Grammatical Framework (GF) grammars (Ranta, 2011), since the algorithm for converting a GF grammar results in an erasing PMCFG (Ljunglöf, 2004).

In this section we give an informal introduction to the algorithm by Angelov (2009), and discuss its similarities with three other algorithms (Kallmeyer and Maier, 2009; Kanazawa, 2008; Ljunglöf, 2004). We argue that all four algorithms implement the same basic Earley-style top-down parsing algorithm.

$$\boxed{\begin{array}{llll}
S \to A_1\,A_2 & S \to A_1'\,A_2' & S \to A_1''\,A_2'' & \ldots \\
A_1 \to a\,b & A_1' \to a\,A_1\,b & A_1'' \to a\,A_1'\,b & \ldots \\
A_2 \to c\,d & A_2' \to c\,A_2\,d & A_2'' \to c\,A_2'\,d & \ldots
\end{array}}$$

Figure 3: Infinite context-free equivalent of the example PMCFG in figure 1.

2.1 Angelov's top-down algorithm

Angelov (2009) views a PMCFG as a CFG with a possibly infinite number of nonterminals and rules. Note that the term "infinite CFG" is an oxymoron, since such a grammar can recognize non-context-free languages. E.g., the infinite CFG shown in figure 3 is equivalent to the grammar in figure 1, which recognizes a non-context-free language.

Since this CFG is infinite, it cannot be calculated from the PMCFG beforehand. But given a certain input string, there are only a finite number of nonterminals and rules that are used in the final parse trees. Angelov's idea is to dynamically create nonterminals and rules during parsing: Whenever the parser has recognized a new constituent r of a nonterminal A, between input positions $i{-}j$, it creates a new nonterminal $A' = A(i, j, r)$, and new grammar rules for A'.

2.2 Similarity with other approaches

Angelov's dynamic CFG is not conceptually different from the parsing algorithms described by other authors. The new nonterminals are all of the form $A' = A(i_1, j_1, r_1) \ldots (i_n, j_n, r_n)$, which is equivalent to the PMCFG nonterminal A, coupled with a sequence of the constituents that have been found during parsing. This is similar to how other algorithms store their found constituents:

- Angelov (2009) uses an ordered sequence where the constituents are stored in the order they are found.

- Ljunglöf (2004, section 4.6) separates the nonterminal A from the "range record" Γ, which is a set containing the found constituents.

Angelov	Ljunglöf	K&M	Kanazawa
$A(i_1, j_1, r_1)$	$\{r_1 : i_1\text{-}j_1\}$	$\langle i_1\text{-}j_1, ?, \ldots, ?\rangle$	$A_1(i_1, j_1)$
$A(i_1, j_1, r_1) \ldots (i_n, j_n, r_n)$	$\{r_1 : i_1\text{-}j_1, \ldots, r_n : i_n\text{-}j_n\}$	$\langle i_1\text{-}j_1, \ldots, i_n\text{-}j_n\rangle$	$A_n(i_1, j_1, \ldots, i_n, j_n)$

Table 1: How different algorithms denote similar derived facts.

- Kallmeyer and Maier (2009) also separates the nonterminal A from the "range vector" Φ, which is a ordered tuple of ranges. In a range vector, the constituents that are not yet found are uninstantiated.

- Kanazawa (2008) derives successively increasing Datalog facts of arity (2 $\times$ the number of found constituents).

In table 1 we can see how the different authors denote similar derived facts. The first row corresponds to when the first constituent r_1 of a nonterminal A has been found spanning the positions $i_1 - j_1$. The second row corresponds to when the parser has found all n constituents of A.

All the discussed algorithms use the same general top-down parsing strategy: First they predict the toplevel nonterminal, followed by its children, and further down until they reach the lexical rules. Then they try to match the lexical rules with the next input token, and so on. This strategy is a PMCFG version of Earley's context-free parsing algorithm (Earley, 1970).

2.3 Bottom-up parsing

However, there are alternative context-free parsing strategies which have not been adopted for PMCFG parsing (Sikkel and Nijholt, 1997; Moore, 2004). Ljunglöf (2004, section 4.6.1) makes an attempt at bottom-up prediction, but it is not very efficient. The problem with a pure bottom-up approach is that the algorithm will recognize all constituents independently, and then it is very costly to combine the constituents further up in the parse tree.

The solution we adopt in the next section is to always use a top-down parsing strategy to recognize additional constituents. This means that the different prediction strategies in sections 3.4–3.6 only apply when recognizing the first constituent of a grammar rule.

3 Three new algorithms

We describe our parsing algorithms as deductive parsing systems (Shieber et al., 1995; Sikkel, 1998), where we infer a set of parse items called a *chart*. Furthermore, we assume that the input is given as n input tokens $w_1 w_2 \ldots w_n$.

3.1 Parse items

We use four different kinds of parse items, and we divide the chart into four indexed sets, $\mathbf{A}_{j,k}$, $\mathbf{F}_{j,k}$, $\mathbf{P}_k$ and $\mathbf{R}_k$, where $j \leq k$ are input positions. The fundamental item is the *active item*:

$$[\, r : \alpha \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{j,k}$$

which says that the parser is trying to find the constituent $A.r$ using the linearization $f.r = \alpha\beta$. It has already found α between the positions j and k, but is still looking for β.

The other parse items are strictly not necessary, but they simplify our presentation of the algorithms. The following *predict item* says that the parser is looking for a constituent $A.r$ starting in position k:

$$[\, ?A.r\,] \in \mathbf{P}_k$$

When the parser finds the constituent $A.r$ between j and k, it creates a new nonterminal $A' = A(j, k, r)$, and infers both a *dynamic grammar rule* and a *passive item*:

$$[\, A' \to f(\vec{B})\,] \in \mathbf{R}_k \qquad [\, A.r : A'\,] \in \mathbf{F}_{j,k}$$

The dynamic rule is used whenever a parent searches for a new constituent of a partly recognized A child. The passive item says that $A.r$ has been found between j and k, and is used when the parser combines the recognized constituent with an active item looking for $A.r$.

We have reformulated Angelov's algorithm slightly so that it fits better with our alternative parsing strategies. Angelov does not use predict items, but we have added them since they simplify the filtered bottom-up parsing strategy (Moore, 2004). Another difference is that we separate the original (static) grammar rules $A \to f(\vec{B})$ from the dynamic rules $[\, A' \to f(\vec{B})\,]$.

predict-item $$\frac{[\,r : \alpha \bullet \langle d.s\rangle\ \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{j,k}}{[\,?B_d.s\,] \in \mathbf{P}_k}$$

predict-next $$\frac{[\,A \to f(\vec{B})\,] \in \mathbf{R}_j \quad [\,?A.r\,] \in \mathbf{P}_k}{[\,r :\ \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k,k}}\ f.r = \beta, j \leq k$$

scan $$\frac{[\,r : \alpha \bullet w_k\ \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{j,k-1}}{[\,r : \alpha\ w_k \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{j,k}}$$

complete $$\frac{[\,r : \alpha \bullet\ \mid A \to f(\vec{B})\,] \in \mathbf{A}_{j,k}}{[\,A' \to f(\vec{B})\,] \in \mathbf{R}_k \quad [\,A.r : A'\,] \in \mathbf{F}_{j,k}}\ A' = A(j,k,r)$$

combine $$\frac{[\,r : \alpha \bullet \langle d.s\rangle\ \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{i,j} \quad [\,B_d.s : B'_d\,] \in \mathbf{F}_{j,k}}{[\,r : \alpha\ \langle d.s\rangle \bullet \beta \mid A \to f(\vec{B}[d := B'_d])\,] \in \mathbf{A}_{i,k}}$$

Figure 4: General inference rules

3.2 General inference rules

Most of the inference rules will be reused by the alternative algorithms, so we split the inference rules into general rules (which are used by all algorithms), and algorithm-specific rules. The general inference rules are shown in figure 4. Since we want to be able to use different prediction strategies for the first constituent, and still predict additional constituents top-down, we have split Angelov's top-down prediction into two separate inference rules. The rule that predicts additional constituents is included here as *predict-next*.

The inference rules *predict-item*, *scan* and *complete* are mutually exclusive, since their antecedent is an active item which either looks for a nonterminal, a terminal, or nothing at all. The rule *predict-item* infers a predict item for $B_d.s$ from an active item looking for $\langle d.s\rangle$. The *scan* rule moves the dot forward if the active item is looking for the next input token w_k. The *complete* rule applies when the dot is at the end of the linearization, and it derives a dynamic rule and a passive item, together with a fresh dynamic nonterminal $A' = A(j,k,r)$.

The fundamental inference rule is *combine*, which takes one active item looking for a nonterminal $B_d.s$ starting in position j, and one passive item that has found $B_d.s$ between j and k. The *combine* rule moves the dot of the active item forward, but it also updates the nonterminal child B_d to $B'_d = B_d(j,k,s)$.

Now, if the active item that is inferred by *combine*, in a later parsing stage wants to find another constituent of B'_d (say $B'_d.u$ starting in position $p \geq k$), the item $[\,?B'_d.u\,] \in \mathbf{P}_p$ is infered by *predict-item*. This in turn triggers *predict-next* to look for a dynamic rule $[\,B'_d \to g(\ldots)\,]$. But since that rule was inferred at the same time as the passive item $[\,B_d.s : B'_d\,] \in \mathbf{F}_{j,k}$, *predict-next* will start recognizing the constituent $B'_d.u$.

3.3 Top-down prediction

The only infenrence rules that are specific to a parsing strategy are the prediction rules. Angelov's (2009) unfiltered top-down strategy consists of the two rules shown in figure 5. Whenever there is a predict item looking for $A.r$ (where A is a grammar nonterminal, not a dynamic one), *predict-topdown* finds all A rules in the grammar and adds them as active items.

The parsing process is initiated by *init-topdown*, which predicts the starting nonterminal S at the beginning of the string.

3.4 Bottom-up prediction

In bottom-up parsing we predict a nonterminal constituent only when its first symbol has been found (Ljunglöf and Wirén, 2010, section 4.4.4). There are three possibilities, depending on whether the first symbol is a terminal, a nonterminal, or if the constitutent is empty. They constitute the inference rules *predict-bottomup*, *scan-bottomup* and *scan-empty*, respectively.

init-topdown $$\frac{}{[\,?S.r\,] \in \mathbf{P}_0}\ \mathrm{start}(S)$$

predict-topdown $$\frac{[\,?A.r\,] \in \mathbf{P}_k}{[\,r:\ \bullet\beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k,k}}\ A \to f(\vec{B}), f.r = \beta$$

Figure 5: Top-down prediction

predict-bottomup $$\frac{[\,B_d.s : B'_d\,] \in \mathbf{F}_{j,k}}{[\,r : \langle d.s\rangle \bullet \beta \mid A \to f(\vec{B}[d := B'_d])\,] \in \mathbf{A}_{j,k}}\ A \to f(\vec{B}), f.r = \langle d.s\rangle\ \beta$$

scan-bottomup $$\frac{}{[\,r : w_k \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k-1,k}}\ A \to f(\vec{B}), f.r = w_k\ \beta$$

scan-empty $$\frac{}{[\,r:\ \bullet\ \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k,k}}\ A \to f(\vec{B}), f.r = \epsilon$$

Figure 6: Bottom-up prediction

The rule *predict-bottomup* is triggered when we have found a passive item covering the first constituent $\langle d.r\rangle$ of $f.r$. Since $B_d.r$ is found, we can directly move the dot past $\langle d.r\rangle$, but then we have to update the nonterminal B_d to the dynamic nonterminal B'_d, in the same way as *combine* does.

One problem with this algorithm is that *scan-empty* adds an item for every empty constituent $A.r$ and every position k in the input. Depending on the grammar, this can lead to a very large chart. There are several ways to solve this: One is to let *scan-bottomup* and *predict-bottomup* skip over initial empty constituents, similar to the context-free GHR algorithm (Graham et al., 1980). Another possibility is to only infer an empty constituent $A.r$ if it can be followed by the input token starting in position k. Our solution is to use the left corner relation as a filter, see section 3.6.

3.5 Filtered top-down prediction

The problem with the top-down algorithm is that it predicts lots of useless items that cannot possibly be inferred from the input tokens. So, we augment *predict-topdown* with a filter, shown in figure 7. Using this filter the parser can only predict a new $A.r$ item in position k if the next input token w_{k+1} is a left corner ($A.r \rhd w_{k+1}$), or if the constituent is empty ($A.r \Rightarrow \epsilon$).

This filter is not as strict as it can be, since it doesn't test empty constituents against the input. One way of making it stricter would be to only predict empty constituents that can be followed by the next input token w_{k+1}.

3.6 Filtered bottom-up prediction

A problem with bottom-up prediction is that it infers lots of items that cannot be used in a final parse tree. We adapt a context-free left corner strategy (Moore, 2004) to our bottom-up algorithm. The modified inference rules are shown in figure 8.

Each inference rule now requires a predict item $[\,?D.u\,]$, such that $D.u \rhd A.r$. In other words, the parser will only predict a constituent $A.r$ if it is the left corner of another constituent $D.u$ that the parser is already looking for. To initialize this left corner filter, we borrow the *init-topdown* rule from the top-down strategy.

3.7 Other possible filters

Kallmeyer and Maier (2009) discuss two additional filters. The *length filter* prohibits parse items that are too long to fit in the sentence. The *terminal filter* checks that all terminals in a linearization occurs among the input tokens, and in the same order.

We have not incorporated their filters in our parser implementations, but we see no reason why this could not be done.

$$\textit{init-topdown} \qquad \frac{}{[\,?S.r\,] \in \mathbf{P}_0} \;\; \mathrm{start}(S)$$

$$\textit{predict-topdown} \qquad \frac{[\,?A.r\,] \in \mathbf{P}_k}{[\,r : \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k,k}} \;\; \begin{cases} A \to f(\vec{B}), f.r = \beta \\ A.r \Rightarrow \epsilon \;\lor\; A.r \triangleright w_{k+1} \end{cases}$$

Figure 7: Top-down prediction with bottom-up filtering

$$\textit{init-topdown} \qquad \frac{}{[\,?S.r\,] \in \mathbf{P}_0} \;\; \mathrm{start}(S)$$

$$\textit{predict-bottomup} \qquad \frac{[\,?D.u\,] \in \mathbf{P}_j \quad [\,B_d.s : B'_d\,] \in \mathbf{F}_{j,k}}{[\,r : \langle d.s \rangle \bullet \beta \mid A \to f(\vec{B}[d := B'_d])\,] \in \mathbf{A}_{j,k}} \;\; \begin{cases} A \to f(\vec{B}), f.r = \langle d.s \rangle \; \beta \\ D.u \triangleright A.r \end{cases}$$

$$\textit{scan-bottomup} \qquad \frac{[\,?D.u\,] \in \mathbf{P}_{k-1}}{[\,r : w_k \bullet \beta \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k-1,k}} \;\; \begin{cases} A \to f(\vec{B}), f.r = w_k \; \beta \\ D.u \triangleright A.r \end{cases}$$

$$\textit{scan-empty} \qquad \frac{[\,?D.u\,] \in \mathbf{P}_k}{[\,r : \bullet \mid A \to f(\vec{B})\,] \in \mathbf{A}_{k,k}} \;\; \begin{cases} A \to f(\vec{B}), f.r = \epsilon \\ D.u \triangleright A.r \end{cases}$$

Figure 8: Bottom-up prediction with left corner filtering

4 Incrementality and optimizations

Let us define *stage* k to be all sets $\mathbf{A}_{i,k}$, $\mathbf{F}_{i,k}$, $\mathbf{P}_k$ and $\mathbf{R}_k$ that end in position k. Then we can say that a parser is *incremental* if all sets in stage k are computed before it starts computing the sets in stage $k+1$. All our inference rules are straightforward to implement incrementally, since no antecedent belongs to a later stage than the state belonging to the consequent item.

If we assume that the implementation is incremental, we can make some optimizations, some more obvious than others. One immediate consequence is that *predict-next* never needs to check if $j \leq k$ since it is trivially satisfied. But there are more things that can be optimized.

4.1 Dynamic rules

As the rule is stated in figure 4, if *predict-next* is triggered by the predict item, it will have to search through all sets $\mathbf{R}_0, \ldots, \mathbf{R}_k$ to find a matching dynamic rule. However, if parsing is performed incrementally, we do not have to separate the rules into different sets, but we can instead add all dynamic rules to one big set $\mathbf{R}$.

4.2 Optimizing previous stages

The passive sets $\mathbf{F}_{j,k}$ are only used in stage k. This means that when the parser starts building the $k+1$ sets, it can discard all sets $\mathbf{F}_{j,k}$ from stage k. The same holds for the predict items $\mathbf{P}_k$, except in the filtered bottom-up algorithm.

Furthermore, since all dynamic nonterminals $A' = A(j, k, r)$ are created in stage k, they become static in later stages. This means that when stage k is completed, we can replace all stage k nonterminals with atomic values, such as fresh integers. It is more efficient to compare atomic values than to compare sequences of the form $A(i_1, j_1, r_1) \ldots (i_n, j_n, r_n)$.

Angelov (2009) implements both these optimizations, and also the previous one merging the dynamic rule sets into one big set $\mathbf{R}$.

4.3 Filtered bottom-up and empty rules

If the grammar contains empty constituents, the filtered bottom-up strategy in figure 8 could be very slow. This is because every time a new predict item $[\,?D.u\,] \in \mathbf{P}_k$ is inferred, *predict-bottomup* tries to find some passive item $[\,B_d.s : B'_d\,] \in \mathbf{F}_{k,k}$ that is a left corner of $D.u$. Most of the time this fails, or the active item that is

	English Resource	**English FraCaS**	**Swedish FraCaS**
Nr. terminals (w)	1,549	1,549	208
Nr. nonterminals (A)	189	194	274
Nr. constituents ($A.r$)	4,663	4,728	3,178
Nr. grammar rules ($A \to f(\vec{B})$)	43,910	2,992	1,967
Nr. linearizations ($f.r = \alpha$)	256,855	74,709	35,365
Nr. left corner pairs ($D.u \triangleright A.r$)	323,471	256,865	915,650

Table 2: The grammars used for testing

	English Resource	**English FraCaS**	**Swedish FraCaS**
Nr. terminals (w)	1,549	1,549	208
Nr. nonterminals (A)	211	231	468
Nr. constituents ($A.r$)	20,669	22,422	8,017
Nr. grammar rules ($A \to f(\vec{B})$)	45,919	135,121	103,334
Nr. linearizations ($f.r = \alpha$)	567,818	1,318,915	3,701,923
Nr. left corner pairs ($D.u \triangleright A.r$)	400,657	424,709	2,288,044

Table 3: The test grammars with empty constituents removed

the consequence will already be in the chart. In the end, lot of useless work will be performed by *predict-bottomup*.

This problem completely disappears if the grammar does not have any empty constituents. In that case all passive items will span at least one input token, i.e., $j < k$, and the predict items will always be from an earlier parsing state.

4.4 Building the sets in stages

Especially the bottom-up strategies benefit from a grammar without empty constituents. In that case, the bottom-up strategies have no use *scan-empty*, and the side condition in *predict-topdown* can be simplified. Furthermore, *combine* can only be triggered by the passive item, since the active item will be from an earlier parsing state.

If the grammar is non-empty, the inference rules also say something about in which order the sets can be built. As an example, if the grammar is non-empty, the set $\mathbf{A}_{k,k}$ can only be created by predict-next from $\mathbf{P}_k$ which on the other hand is built by predict-item from $\mathbf{A}_{jk}$ ($j \leq k$). This means that $\mathbf{A}_{k,k}$ and $\mathbf{P}_k$ depend on each other.

By analyzing all inference rules, we come to the following build order, where $j < k$:

$$\mathbf{A}_{j,j}, \mathbf{P}_j \Rightarrow \mathbf{A}_{j,k}, \mathbf{F}_{j,k} \Rightarrow \mathbf{R}_k \Rightarrow \mathbf{A}_{k,k}, \mathbf{P}_k$$

This ordering suggests the following pseudo-code for parsing n tokens using a non-empty grammar:

1. Build the sets $\mathbf{P}_0$ and $\mathbf{A}_{0,0}$
   [*init-topdown, pred.-topdown, pred.-item*]

2. For each k between 1 and n:

   (a) Build $\mathbf{A}_{j,k}$ and $\mathbf{F}_{j,k}$ (for all $j < k$)
   [*complete, combine, scan, scan-bottomup, pred.-bottomup*]
   As a side-effect, this will also add new rules to $\mathbf{R}$

   (b) Build $\mathbf{P}_k$ and $\mathbf{A}_{k,k}$
   [*pred.-topdown, pred.-item, pred.-next*]

5 Evaluation

We performed a small evaluation of our four parsing strategies. We tested two English grammars and one Swedish grammar written in GF (Ranta, 2011) on 100 randomly selected sentences from the FraCaS textual inference problem set (Cooper et al., 1996). One of the English grammars is the GF English Resource grammar (Ranta, 2009) with the FraCaS lexicon added. The other two grammars and the Swedish translations of the sentences are taken from the FraCaS GF Treebank (Ljunglöf and Siverbo, 2011).

	English Resource			English FraCaS			Swedish FraCaS		
	chart	time	/item	chart	time	/item	chart	time	/item
Top-down	721,000	9,2 s	13 μs	96,000	1,7 s	18 μs	96,000	1,7 s	18 μs
Bottom-up	144,000	2,5 s	17 μs	51,000	1,1 s	21 μs	167,000	3,8 s	23 μs
Filtered top-down	239,000	3,2 s	13 μs	38,000	0,8 s	21 μs	27,000	0,5 s	20 μs
Filtered bottom-up	27,000	0,5 s	17 μs	8,000	0,2 s	20 μs	17,000	0,6 s	34 μs

Table 4: Average chart size per sentence, parse time per sentence and parse time per chart item.

5.1 Non-empty bottom-up grammars

As explained in section 4.3, the bottom-up strategies perform especially poorly if the grammars contain empty constituents. So we also created non-empty versions of the grammars. Table 2 contains some statistics about the grammars and table 3 about their non-empty versions. The tables show that the size of the grammar can explode quite dramatically when removing empty constituents, but it depends on the grammar. E.g., the number of rules in the FraCaS grammars increase by 50 times, while the Resource grammar is almost unaffected.

Despite the dramatic increase of the grammar size, the non-empty grammars always outperform the empty grammars when doing bottom-up parsing. On the other hand, the top-down strategies perform much worse on the empty grammars. Therefore we tested the bottom-up strategies on the empty grammars, and the top-down strategies on the original grammars.

5.2 Test results

We tested each of the four parsing strategies on each of the three grammars and their test sentences. The results are given in table 4, which contains the average size of the chart after parsing each test sentence, together with the average parsing time and the average parsing time per chart item. The tests were run on a 2GHz Intel Core2Duo processor with 4GB RAM.

It is clear from the table that the filtered algorithms outperform the unfiltered ones, and that the filtered bottom-up algorithm is the fastet most of the time. However, we have only tested on three quite similar grammars, so the results could very well be different when testing on other grammars.

Note the increase in parsing time per item for the filtered bottom-up algorithm on the Swedish grammar. This is most certainly caused by the extreme size of the non-empty Swedish grammar, forcing the Python interpreter to perform garbage collection much more often than usual.

6 Final remarks

We compared four previously published PMCFG/LCFRS parsing algorithms, and argued that they all implement the same top-down Earley style algorithm without bottom-up filtering. From Angelov's (2009) algorithm we derived three new PMCFG parsing algorithms, one pure bottom-up variant and two variants using a left corner filter. An initial evaluation suggested that these new algorithms can increase PMCFG parsing performance dramatically, at least for some grammars.

6.1 The correct-prefix property

An algorithm satisfies the correct-prefix property (CPP) if it aborts and reports a failure as soon as it reads a prefix that is not a prefix of any correct string in the grammar (Nederhof, 1999).

Angelov's (2009) original top-down algorithm already satisfies CPP, and since the filtered top-down algorithm does not add any additional parse items it also satisfies CPP.

The filtered bottom-up algorithm would also be prefix-correct if the left corner relation was perfect. But we extract the left corners from a context-free approximation, which means that the relation is over-generating. Therefore some non-CPP parse items can be pass through the left corner filter, which means that neither of the bottom-up algorithms satisfy CPP.

6.2 Implementation

We have implemented all four algorithms as a library in the programming language Python. The library is released under an open-source licence and can be downloaded or forked from the following URL:

http://github.com/heatherleaf/MCFParser.py

Acknowledgments

I would like to thank Krasimir Angelov and three anonymous reviewers for insightful and valuable comments and suggestions.

References

Krasimir Angelov. 2009. Incremental parsing with parallel multiple context-free grammars. In *EACL'09, 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 69–76, Athens, Greece.

Pierre Boullier. 2004. Range concatenation grammars. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New developments in parsing technology*, pages 269–289. Kluwer Academic Publishers.

Håkan Burden and Peter Ljunglöf. 2005. Parsing linear context-free rewriting systems. In *IWPT'05, 9th International Workshop on Parsing Technologies*, Vancouver, Canada.

Robin Cooper, Dick Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jaspars Jan, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman, Ted Briscoe, Holger Maier, and Karsten Konrad. 1996. Using the framework. Deliverable D16, FraCaS Project.

Éric Villemonte de la Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *COLING'02, 19th International Conference on Computational Linguistics*.

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

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2008. A deductive approach to dependency parsing. In *ACL'08: HLT, 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, Ohio.

Susan Graham, Michael Harrison, and Walter Ruzzo. 1980. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462.

Laura Kallmeyer and Wolfgang Maier. 2009. An incremental Earley parser for simple range concatenation grammar. In *IWPT'09, 11th International Workshop on Parsing Technologies*, Paris, France.

Laura Kallmeyer. 2010. *Parsing Beyong Context-Free Grammars*. Springer.

Makoto Kanazawa. 2008. A prefix-correct Earley recognizer for multiple context-free grammars. In *TAG+9, 9th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Tübingen, Germany.

Peter Ljunglöf and Magdalena Siverbo. 2011. A bilingual treebank for the FraCaS test suite. CLT project report, University of Gothenburg.

Peter Ljunglöf and Mats Wirén. 2010. Syntactic parsing. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing, 2nd edition*, chapter 4. CRC Press, Taylor and Francis. ISBN 978-1420085921.

Peter Ljunglöf. 2004. *Expressivity and Complexity of the Grammatical Framework*. Ph.D. thesis, University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden.

Robert C. Moore. 2004. Improved left-corner chart parsing for large context-free grammars. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, pages 185–201. Kluwer Academic Publishers.

Mark-Jan Nederhof. 1999. The computational complexity of the correct-prefix property for TAGs. *Computational Linguistics*, 25(3):345–360.

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2.

Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford.

Hiroyuki Seki, Takashi Matsumara, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Klaas Sikkel and Anton Nijholt. 1997. Parsing of context-free languages. In G. Rozenberg and A. Salomaa, editors, *The Handbook of Formal Languages*, volume II, pages 61–100. Springer-Verlag.

Klaas Sikkel. 1998. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199:87–103.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *ACL'87, 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, California.