An Incremental Earley Parser for Simple Range Concatenation Grammar

Laura Kallmeyer and **Wolfgang Maier**
Collaborative Research Center 833
University of Tübingen
Tübingen, Germany
{lk,wmaier}@sfs.uni-tuebingen.de

Abstract

We present an Earley-style parser for simple range concatenation grammar, a formalism strongly equivalent to linear context-free rewriting systems. Furthermore, we present different filters which reduce the number of items in the parsing chart. An implementation shows that parses can be obtained in a reasonable time.

1 Introduction

Linear context-free rewriting systems (LCFRS) (Vijay-Shanker et al., 1987), the equivalent multiple context-free grammars (MCFG) (Seki et al., 1991) and simple range concatenation grammars (sRCG) (Boullier, 1998) have recently attracted an increasing interest in the context of natural language processing. For example, Maier and Søgaard (2008) propose to extract simple RCGs from constituency treebanks with crossing branches while Kuhlmann and Satta (2009) propose to extract LCFRS from non-projective dependency treebanks. Another application area of this class of formalisms is biological computing (Kato et al., 2006).

This paper addresses the symbolic parsing of sRCG/LCFRS. Starting from the parsing algorithms presented in Burden and Ljunglöf (2005) and Villemonte de la Clergerie (2002), we propose an incremental Earley algorithm for simple RCG. The strategy is roughly like the one pursued in Villemonte de la Clergerie (2002). However, instead of the automaton-based formalization in Villemonte de la Clergerie's work, we give a general formulation of an incremental Earley algorithm, using the framework of parsing as deduction. In order to reduce the search space, we introduce different types of filters on our items. We have implemented this algorithm and tested it on simple RCGs extracted from the German treebanks Negra and Tiger.

In the following section, we introduce simple RCG and in section 3, we present an algorithm for symbolic parsing of simple RCG. Section 4 then presents different filtering techniques to reduce the number of items. We close discussing future work.

2 Grammar Formalism

A **range concatenation grammar (RCG)** is a 5-tuple $G = (N, T, V, P, S)$. N is a finite set of non-terminals (predicate names) with an arity function dim: $N \to \mathbb{N}^+$, T and V are disjoint finite sets of terminals and variables. P is a finite set of clauses of the form $\psi_0 \to \psi_1 \dots \psi_m$, where $m \geq 0$ and each of the $\psi_i, 0 \leq i \leq m$, is a predicate of the form $A_i(\alpha_1^i, \dots, \alpha_{dim(A)}^i)$. Each $\alpha_j^i \in (T \cup V)^*$, $1 \leq j \leq dim(A)$ and $0 \leq i \leq k$, is an argument. As a shorthand notation for $A_i(\alpha_1, \dots, \alpha_{dim(A)})$, we use $A_i(\vec{\alpha})$. $S \in N$ is the start predicate name with $dim(S) = 1$.

Note that the order of right-hand side (RHS) predicates in a clause is of no importance. Subclasses of RCGs are introduced for further reference: An RCG $G = (N, T, V, P, S)$ is **simple** if for all $c \in P$, it holds that every variable X occurring in c occurs exactly once in the left-hand side (LHS) and exactly once in the RHS, and each argument in the RHS of c contains exactly one variable. A simple RCG is **ordered** if for all $\psi_0 \to \psi_1 \cdots \psi_m \in P$, it holds that if a variable X_1 precedes a variable X_2 in a ψ_i, $1 \leq i \leq m$, then X_1 also precedes X_2 in ψ_0. The ordering requirement does not change the expressive power, i.e., ordered simple RCG is equivalent to simple RCG (Villemonte de la Clergerie, 2002). An RCG is ε-**free** if it either contains no ε-rules or there is exactly one rule $S(\varepsilon) \to \varepsilon$ and S does not appear in any of the righthand sides of the rules in the grammar. A rule is an ε-rule if one of the arguments of the lefthand side is the empty string ε. (Boullier, 1998) shows that for every simple RCG, one can construct an equivalent ε-free simple RCG. An RCG $G = (N, T, V, P, S)$ is a k-**RCG** if for all $A \in N, dim(A) \leq k$.

The language of RCGs is based on the notion of **range**. For a string $w_1 \cdots w_n$ a range is a pair of indices $\langle i, j \rangle$ with $0 \leq i \leq j \leq n$, i.e., a string span, which denotes a substring $w_{i+1} \cdots w_j$ in the source string or a substring $v_{i+1} \cdots v_j$ in the target string. Only consecutive ranges can be concatenated into new ranges. Terminals, variables and arguments in a clause are bound to ranges by a substitution mechanism. An **instantiated** clause is a clause in which variables and arguments are consistently replaced by ranges; its components are **instantiated predicates**. For example $A(\langle g \cdots h \rangle) \rightarrow B(\langle g+1 \cdots h \rangle)$ is an instantiation of the clause $A(aX_1) \rightarrow B(X_1)$ if the target string is such that $w_{g+1} = a$. A **derive** relation $\Rightarrow$ is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause. The language of an RCG $G = (N, T, V, P, S)$ is the set $L(G) = \{w_1 \cdots w_n \mid S(\langle 0, n \rangle) \overset{*}{\Rightarrow} \varepsilon\}$, i.e., an input string $w_1 \cdots w_n$ is recognized if and only if the empty string can be derived from $S(\langle 0, n \rangle)$. In this paper, we are dealing only with ordered simple RCGs. The ordering requirement does not change the expressive power (Villemonte de la Clergerie, 2002). Furthermore, without loss of generality, we assume that for every clause, there is a $k \geq 0$ such that the variables occurring in the clause are exactly $X_1, \ldots, X_k$.

We define derivation trees for simple RCGs as unordered trees whose internal nodes are labelled with predicate names and whose leaves are labelled with ranges such that all internal nodes are licensed by RCG clause instantiations: given a simple RCG G and a string w, a tree $D = \langle V, E, r \rangle$ is a **derivation tree** of $w = a_1 \ldots a_n$ iff 1. there are exactly n leaves in D labelled $\langle 0, 1 \rangle, \ldots, \langle n-1, n \rangle$ and 2. for all $v_0 \in V$ with $v_1, \ldots, v_n \in V$, $n \geq 1$ being all vertices with $\langle v_0, v_i \rangle \in E$ ($1 \leq i \leq n$) such that the leftmost range dominated by v_i precedes the leftmost range dominated by v_{i+1} ($1 \leq i < n$): there is a clause instantiation $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_n(\vec{\rho_n})$ such that a) $l(v_i) = A_i$ for $0 \leq i \leq n$ and b) the yield of the leaves dominates by v_i is $\vec{\rho_i}$.

3 Parsing

Our parsing algorithm is a modification of the "incremental algorithm" of Burden and Ljunglöf (2005) with a strategy very similar to the strategy adopted by *Thread Automata* (Villemonte de la Clergerie, 2002). It assumes the grammar to be ordered and ε-free. We refrain from supporting non-ε-free grammars since the treebank grammars used with our implementation are all ε-free. However, note that only minor modifications would be necessary in order to support non-ε-free grammars (see below).

We process the arguments of LHS of clauses incrementally, starting from an S-clause. Whenever we reach a variable, we move into the clause of the corresponding RHS predicate (**predict** or **resume**). Whenever we reach the end of an argument, we **suspend** this clause and move into the parent clause that has called the current one. In addition, we treat the case where we reach the end of the last argument and move into the parent as a special case. Here, we first **convert** the item into a passive one and then **complete** the parent item with this passive item. This allows for some additional factorization.

The item form for passive items is $[A, \vec{\rho}]$ where A a predicate of some arity k, $\vec{\rho}$ is a range vector of arity k. The item form for active items: $[A(\vec{\phi}) \rightarrow A_1(\vec{\phi_1}) \ldots A_m(\vec{\phi_m}), pos, \langle i, j \rangle, \vec{\rho}]$ where $A(\vec{\phi}) \rightarrow A_1(\vec{\phi_1}) \ldots A_m(\vec{\phi_m}) \in P$; $pos \in \{0, \ldots, n\}$ is the position up to which we have processed the input; $\langle i, j \rangle \in \mathbb{N}^2$ marks the position of our dot in the arguments of the predicate A: $\langle i, j \rangle$ indicates that we have processed the arguments up to the jth element of the ith argument; $\vec{\rho}$ is an range vector containing the bindings of the variables and terminals occurring in the lefthand side of the clause ($\vec{\rho}(i)$ is the range the ith element is bound to). When first predicting a clause, it is initialized with a vector containing only symbols "?" for "unknown". We call such a vector (of appropriate arity) $\vec{\rho}_{init}$. We introduce an additional piece of notation. We write $\vec{\rho}(X)$ for the range bound to the variable X in $\vec{\rho}$. Furthermore, we write $\vec{\rho}(\langle i, j \rangle)$ for the range bound to the jth element in the ith argument of the clause lefthand side.

Applying a range vector $\vec{\rho}$ containing variable bindings for a given clause c to the argument vector of the lefthand side of c means mapping the ith element in the arguments to $\vec{\rho}(i)$ and concatenating adjacent ranges. The result is defined iff every argument is thereby mapped to a range.

We start by **predicting** the S-predicate:

$$\frac{}{[S(\vec{\phi}) \to \bullet\vec{\Phi}, 0, \langle 1, 0\rangle, \vec{\rho}_{init}]}\ S(\vec{\phi}) \to \vec{\Phi} \in P$$

Scan: Whenever the next symbol after the dot is the next terminal in the input, we can scan it:

$$\frac{[A(\vec{\phi}) \to \vec{\Phi}, pos, \langle i, j\rangle, \vec{\rho}]}{[A(\vec{\phi}) \to \vec{\Phi}, pos+1, \langle i, j+1\rangle, \vec{\rho}']}\ \vec{\phi}(i, j{+}1) = w_{pos+1}$$

where $\vec{\rho}'$ is $\vec{\rho}$ updated with $\vec{\rho}(i, j+1) = \langle pos, pos+1\rangle$.

In order to support ε-free grammars, one would need to store the pair of indices a ε is mapped to in the range vector, along with the mappings of terminals and variables. The indices could be obtained through a **Scan**-ε operation, parallel to the **Scan** operation.

Predict: Whenever our dot is left of a variable that is the first argument of some RHS predicate B, we predict new B-clauses:

$$\frac{[A(\vec{\phi}) \to \dots B(X, \dots) \dots, pos, \langle i, j\rangle, \vec{\rho}_A]}{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle 1, 0\rangle, \vec{\rho}_{init}]}$$

with the side condition $\vec{\phi}(i, j+1) = X, B(\vec{\psi}) \to \vec{\Psi} \in P$.

Suspend: Whenever we arrive at the end of an argument that is not the last argument, we suspend the processing of this clause and we go back to the item that was used to predict it.

$$\frac{\begin{array}{c}[B(\vec{\psi}) \to \vec{\Psi}, pos', \langle i, j\rangle, \vec{\rho}_B],\\ [A(\vec{\phi}) \to \dots B(\vec{\xi}) \dots, pos, \langle k, l\rangle, \vec{\rho}_A]\end{array}}{[A(\vec{\phi}) \to \dots B(\vec{\xi}) \dots, pos', \langle k, l+1\rangle, \vec{\rho}]}$$

where the dot in the antecedent A-item precedes the variable $\vec{\xi}(i)$, $|\vec{\psi}(i)| = j$ (the ith argument has length j and has therefore been completely processed), $|\vec{\psi}| < i$ (the ith argument is not the last argument of B), $\vec{\rho}_B(\vec{\psi}(i)) = \langle pos, pos'\rangle$ and for all $1 \leq m < i$: $\vec{\rho}_B(\vec{\psi}(m)) = \vec{\rho}_A(\vec{\xi}(m))$. $\vec{\rho}$ is $\vec{\rho}_A$ updated with $\vec{\rho}_A(\vec{\xi}(i)) = \langle pos, pos'\rangle$.

Convert: Whenever we arrive at the end of the last argument, we convert the item into a passive one:

$$\frac{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle i, j\rangle, \vec{\rho}_B]}{[B, \rho]}\ \begin{array}{l}|\vec{\psi}(i)| = j, |\vec{\psi}| = i,\\ \vec{\rho}_B(\vec{\psi}) = \rho\end{array}$$

Complete: Whenever we have a passive B item we can use it to move the dot over the variable of the last argument of B in a parent A-clause that was used to predict it.

$$\frac{[B, \vec{\rho}_B], [A(\vec{\phi}) \to \dots B(\vec{\xi}) \dots, pos, \langle k, l\rangle, \vec{\rho}_A]}{[A(\vec{\phi}) \to \dots B(\vec{\xi}) \dots, pos', \langle k, l+1\rangle, \vec{\rho}]}$$

where the dot in the antecedent A-item precedes the variable $\vec{\xi}(|\vec{\rho}_B|)$, the last range in $\vec{\rho}_B$ is $\langle pos, pos'\rangle$, and for all $1 \leq m < |\vec{\rho}_B|$: $\vec{\rho}_B(m) = \vec{\rho}_A(\vec{\xi}(m))$. $\vec{\rho}$ is $\vec{\rho}_A$ updated with $\vec{\rho}_A(\vec{\xi}(|\vec{\rho}_B|)) = \langle pos, pos'\rangle$.

Resume: Whenever we are left of a variable that is not the first argument of one of the RHS predicates, we resume the clause of the RHS predicate.

$$\frac{\begin{array}{c}[A(\vec{\phi}) \to \dots B(\vec{\xi}) \dots, pos, \langle i, j\rangle, \vec{\rho}_A],\\ [B(\vec{\psi}) \to \vec{\Psi}, pos', \langle k-1, l\rangle, \vec{\rho}_B]\end{array}}{[B(\vec{\psi}) \to \vec{\Psi}, pos, \langle k, 0\rangle, \vec{\rho}_B]}$$

where $\vec{\phi}(i)(j+1) = \vec{\xi}(k), k > 1$ (the next element is a variable that is the kth element in $\vec{\xi}$, i.e., the kth argument of B), $|\vec{\psi}(k-1)| = l$, and $\vec{\rho}_A(\vec{\xi}(m)) = \vec{\rho}_B(\vec{\psi})(m)$ for all $1 \leq m \leq k-1$.

The **goal item** has the form $[S, \langle 0, n\rangle]$.

Note that, in contrast to a purely bottom-up CYK algorithm, the Earley algorithm presented here is prefix valid, provided that the grammar does not contain useless symbols.

4 Filters

During parsing, various optimizations known from (P)CFG parsing can be applied. More concretely, because of the particular form of our simple RCGs, we can use several filters to reject items very early that cannot lead to a valid parse tree for a given input $w = w_1 \dots w_n$.

Since our grammars are ε-free, we know that each variable or occurrence of a terminal in the clause must cover at least one terminal in the input. Furthermore, since separations between arguments are generated only in cases where between two terminals belonging to the yield of a non-terminal, there is at least one other terminals that is not part of the yield, we know that between different arguments of a predicate, there must be at least one terminal in the input. Consequently, we obtain as a filtering condition on the validity of an active item that the length of the remaining input must be greater or equal to the number of variables and terminal occurrences plus the number of argument separations to the right of the dot in the left-hand side of the clause. More formally, an active item $[A(\vec{\phi}) \to A_1(\vec{\phi_1}) \dots A_m(\vec{\phi_m}), pos, \langle i, j\rangle, \vec{\rho}]$ satisfies the **length filter** iff

$$\begin{array}{l}(n - pos)\\ \geq (|\vec{\phi}(i)| - j) + \Sigma_{k=i+1}^{dim(A)} |\vec{\phi}(k)| + (dim(A) - i)\end{array}$$

The length filter is applied to results of **predict**, **resume**, **suspend** and **complete**.

A second filter, first proposed in Klein and Manning (2003), checks for the presence of required preterminals. In our case, we assume the preterminals to be treated as terminals, so this filter amounts to checking for the presence of all terminals in the predicted part of a clause (the part to the right of the dot) in the remaining input. Furthermore, we check that the terminals appear in the predicted order and that the distance between two of them is at least the number of variables/terminals and argument separations in between. In other words, an active item $[A(\vec{\phi}) \rightarrow A_1(\vec{\phi_1}) \dots A_m(\vec{\phi_m}), pos, \langle i, j\rangle, \vec{\rho}]$ satisfies the **terminal filter** iff we can find an injective mapping $f_T : \mathit{Term} = \{\langle k, l\rangle \mid \vec{\phi}(k)(l) \in T$ and either $k > i$ or ($k = i$ and $l > j$)$\} \rightarrow \{pos + 1, \dots, n\}$ such that

1. $w_{f_T(\langle k,l\rangle)} = \vec{\phi}(k)(l)$ for all $\langle k, l\rangle \in \mathit{Term}$;
2. for all $\langle k_1, l_1\rangle, \langle k_2, l_2\rangle \in \mathit{Term}$ with $k_1 = k_2$ and $l_1 < l_2$: $f_T(\langle k_2, l_2\rangle) \geq f_T(\langle k_1, l_1\rangle) + (l_2 - l_1)$;
3. for all $\langle k_1, l_1\rangle, \langle k_2, l_2\rangle \in \mathit{Term}$ with $k_1 < k_2$: $f_T(\langle k_2, l_2\rangle) \geq f_T(\langle k_1, l_1\rangle) + (|\vec{\phi}(k_1)| - l_1) + \Sigma_{k=k_1+1}^{k_2-1} |\vec{\phi}(k)| + l_2 + (k_2 - k_1)$.

Checking this filter amounts to a linear traversal of the part of the lefthand side of the clause that is to the right of the dot. We start with index $i = pos + 1$, for every variable or gap we increment i by 1. For every terminal a, we search the next a in the input, starting at position i. If it occurs at position j, then we set $i = j$ and continue our traversal of the remaining parts of the lefthand side of the clause.

The preterminal filter is applied to results of the **predict** and **resume** operations.

We have implemented the incremental Earley parser with the filtering conditions on items. In order to test it, we have extracted simple RCGs from the first 1000 sentences of Negra and Tiger (with removed punctuation) using the algorithm described in Maier and Søgaard (2008) and parsed the sentences 1001-1100 with it. The grammars contained 2474 clauses (Negra) and 2554 clauses (Tiger). The following table contains the total number of sentences for different length and resp. the number of sentences for which a parse was found, along with the average parsing times of those that had a parse:

	Negra		Tiger	
	parse/tot	av. t.	parse/tot	av. t.
$\lvert w\rvert \leq 20$	73/84	0.40 sec.	50/79	0.32
$20 \leq \lvert w\rvert \leq 35$	14/16	2.14 sec.	10/19	2.16

5 Conclusion and Future Work

We have presented an Earley-style algorithm for simple range concatenation grammar, formulated as deduction system. Furthermore, we have presented a set of filters on the chart reducing the number of items. An implementation and a test with grammars extracted from treebanks showed that reasonable parsing times can be achieved.

We are currently working on a probabilistic k-best extension of our parser which resumes comparable work for PCFG (Huang and Chiang, 2005). Unfortunately, experiments with the Earley algorithm have shown that with grammars of a reasonable size for data-driven parsing ($> 15,000$ clauses), an exhaustive parsing is no longer efficient, due to the highly ambiguous grammars. Algorithms using only passive items seem more promising in this context since they facilitate the application of A* parsing techniques.

References

Pierre Boullier. 1998. Proposal for a natural language processing syntactic backbone. Rapport de Recherche RR-3342, INRIA.

Håkan Burden and Peter Ljunglöf. 2005. Parsing linear context-free rewriting systems. In *Proceedings of IWPT 2005*.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of IWPT 2005*.

Yuki Kato, Hiroyuki Seki, and Tadao Kasami. 2006. Stochastic multiple context-free grammar for RNA pseudoknot modeling. In *Proceedings of TAG+8*.

Dan Klein and Christopher D. Manning. 2003. A* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of HLT-NAACL*.

Marco Kuhlmann and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of EACL*.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of Formal Grammar 2008*.

Hiroyuki Seki, Takahashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*.

K. Vijay-Shanker, David Weir, and Aravind Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of ACL*.

Eric Villemonte de la Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *Proceedings of COLING*.