LR Parsing for LCFRS

Laura Kallmeyer and **Wolfgang Maier**
Institute for Language and Information
University of Düsseldorf
Düsseldorf, Germany
{kallmeyer,maierwo}@phil.hhu.de

Abstract

LR parsing is a popular parsing strategy for variants of Context-Free Grammar (CFG). It has also been used for mildly context-sensitive formalisms, such as Tree-Adjoining Grammar. In this paper, we present the first LR-style parsing algorithm for Linear Context-Free Rewriting Systems (LCFRS), a mildly context-sensitive extension of CFG which has received considerable attention in the last years.

1 Introduction

LR parsing is an incremental shift-reduce parsing strategy in which the transitions between parser states are guided by an automaton which is compiled offline. LR parsers were first introduced for deterministic context-free languages (Knuth, 1965) and later generalized to context-free languages (Tomita, 1984) and tree-adjoining languages (Nederhof, 1998; Prolo, 2003).

Linear Context-Free Rewriting System (LCFRS) (Vijay-Shanker et al., 1987) is an immediate extension of CFG in which each non-terminal can cover more than one continuous span of the input string. LCFRS and equivalent formalisms have been used for the modeling of discontinuous constituents (Maier and Lichte, 2011) and non-projective dependencies (Kuhlmann, 2013), as well as for data-driven parsing of such structures (Maier and Kallmeyer, 2010; Kallmeyer and Maier, 2013; van Cranenburgh, 2012; Angelov and Ljunglöf, 2014). They have also been used for modeling non-concatenative morphology (Botha and Blunsom, 2013), for grammar engineering (Ranta, 2011), and for modeling alignments in machine translation (Søgaard, 2008; Kaeshammer, 2013). To our knowledge, so far, no LR strategy for LCFRS has been presented in the literature. In this paper, we present an LR-style parser for LCFRS. It is based on the incremental parsing strategy implemented by Thread Automata (Villemonte de la Clergerie, 2002).

The remainder of the article is structured as follows. In the following section, we introduce LCFRS and thread automata. Section 3 presents the algorithm along an example. In particular, section 3.2 gives the algorithms for automaton and parse table constructions, and section 3.3 presents the parsing algorithm. Section 4 concludes the article.

2 Preliminaries

2.1 LCFRS

In this paper, we restrict ourselves to string rewriting LCFRS and omit the more general definition (Weir, 1988).

In LCFRS, a single non-terminal can span $k \geq 1$ continuous blocks of a string. A CFG is simply a special case of an LCFRS in which $k = 1$. k is called the *fan-out* of the non-terminal. We notate LCFRS with the syntax of Simple Range Concatenation Grammars (SRCG) (Boullier, 1998), a formalism equivalent to LCFRS.

An LCFRS[1] (Vijay-Shanker et al., 1987; Seki et al., 1991) is a tuple $G = (N, T, V, P, S)$ where N

[1]Note that for purposes of exposition, we limit ourselves to ε-free LCFRS.

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 1250–1255,
Denver, Colorado, May 31 – June 5, 2015.

is a finite set of non-terminals with a function dim: $N \to \mathbb{N}$ determining the *fan-out* of each $A \in N$; T and V are disjoint finite sets of terminals and variables; $S \in N$ is the start symbol with $dim(S) = 1$.

P is a finite set of rewriting rules with *rank* $m \geq 0$. All $\gamma \in P$ have the form

$$A(\alpha_0, \ldots, \alpha_{dim(A)-1}) \to A_1(X_0^{(1)}, \ldots, X_{dim(A_1)-1}^{(1)}) \\ \cdots A_m(X_0^{(m)}, \ldots, X_{dim(A_m)-1}^{(m)})$$

where $A, A_1, \ldots, A_m \in N$, $X_j^{(l)} \in V$ for $1 \leq l \leq m, 0 \leq j < dim(A_i)$ and $\alpha_i \in (V \cup T)^+$ for $0 \leq i < dim(A)$. All α_i and $X_j^{(l)}$ are called *arguments* (or sometimes *components*); the elements in α_i are called *argument elements*. $\mathcal{A}_\gamma$ is the set of all argument elements of γ. Variable occurrences in the arguments of the non-terminals of γ are ordered by a strict total order $\prec$. For all $X_1, X_2 \in V$ occurring in arguments of a non-terminal of γ, it holds that $X_1 \prec X_2$ iff either X_1 precedes X_2 in an argument of the non-terminal or the argument X_1 occurs in precedes the argument X_2 occurs in.

For all $\gamma \in P$, every variable X occurring in γ occurs exactly once in the left-hand side (LHS) and exactly once in the right-hand side (RHS). Furthermore, if for two variables $X_1, X_2 \in V$, it holds that $X_1 \prec X_2$ on the RHS, then also $X_1 \prec X_2$ on the LHS. The *rank* of G is the maximal rank of any of its rules, its *fan-out* is the maximal fan-out of any of its non-terminals.

We use the following additional notation: For a rule $\gamma \in P$, $lhs(\gamma)$ gives the LHS non-terminal; $lhs(\gamma, i)$ gives the ith argument of the LHS and $lhs(\gamma, i, j)$ its jth symbol; $rhs(\gamma, k)$ gives the kth RHS non-terminal; and $rhs(\gamma, k, l)$ gives the lth component of the kth RHS element (starting with index 0 in all four cases). These function have value $\perp$ whenever there is no such element. Furthermore, in the sense of dotted productions, we define for each $\gamma \in P$ a set of symbols denoting computation points of γ, $\mathcal{C}_\gamma = \{\gamma_{i.j} \mid 0 \leq i < dim_A, 0 \leq j \leq |\alpha_i|\}$, as well as the set $\mathcal{C} = \bigcup_{\gamma \in P} \mathcal{C}_\gamma$.

A non-terminal $A \in N$ can be *instantiated* w.r.t. an input string $w_1 \cdots w_{|w|}$ and a rule $\gamma \in P$ with $lhs(\gamma) = A$. An instantiation maps all argument elements of γ to spans of w ($(i-1, j)^w$ denotes the span $w_i \cdots w_j$, $1 \leq i \leq j \leq n$). All instantiations are given by a function $\sigma : \mathcal{A}_\gamma \to \mathbb{N} \times \mathbb{N}$ where

$\alpha : S(xy) \to A(x, y) \quad \gamma : A(a, b) \to \varepsilon$
$\beta : A(ax, ya) \to A(x, y)$

Figure 1: LCFRS for $\{a^n aba^n \mid n \geq 0\}$

for all $x, y \in \mathcal{A}_\gamma$ with $x \neq y$, $\sigma(x) = (i, j)^w$ and $\sigma(y) = (k, l)^w$ it holds that $i, k \geq 0$; $j, l \leq |w|$; if x (y) is a terminal, then $j = i + 1$ ($l = k + 1$), otherwise $j > i$ ($k > l$). Iff $x \prec y$ in γ, then $j \leq k$. A derivation rewrites strings of instantiated non-terminals, i.e., given an instantiated clause, the instantiated LHS non-terminal may be replaced with the sequence of instantiated RHS terminals. The language of the grammar is the set of strings which can be reduced to the empty word, starting with S instantiated to the input string.

See figure 1 for a sample LCFRS.

2.2 Thread Automata

Thread automata (TA) (Villemonte de la Clergerie, 2002) are a generic automaton model which can be parametrized to recognize different mildly context-sensitive languages. The TA for LCFRS (LCFRS-TA) implements a prefix-valid top-down incremental parsing strategy similar to the ones of Kallmeyer and Maier (2009) and Burden and Ljunglöf (2005).

An LCFRS-TA for some LCFRS $G = (N, T, V, P, S)$ works as follows. The processing of a single rule is handled by a single *thread* which will traverse the LHS arguments of the rule. A thread is given by a pair $p : X$, where $p \in \{1, \ldots, m\}^*$ with m the rank of G is the *address*, and $X \in N \cup \{ret\} \cup \mathcal{C}$ where $ret \notin N$ is the *content* of the thread. An automaton state is given by a tuple $\langle i, p, \mathcal{T} \rangle$ where $\mathcal{T}$ is a set of threads, the *thread store*, p is the address of the active thread, and $i \geq 0$ indicates that i tokens have been recognized. We introduce a new start symbol $S' \notin N$ that expands to S and use $\langle 0, \varepsilon, \{\varepsilon : S'\} \rangle$ as start state.

The specific TA for a given LCFRS $G = (N, T, V, P, S)$ can be defined as tuple $\langle N', T, S', ret, \delta, \Theta \rangle$ with $N' = N \cup \mathcal{C} \cup \{S', ret\}$; δ is a function from $\mathcal{C}$ to $\{1, \ldots, m\} \cup \{\perp\}$ such that $\delta(\gamma_{k,i}) = j$ if there is a l such that $lhs(\gamma, k, i) = rhs(\gamma, j-1, l)$, and $\delta(\gamma_{k,i}) = \perp$ if $lhs(\gamma, k, i) \in T \cup \{\perp\}$ (intuitively, a δ value j tells us that the next symbol to process is a variable that

Call:	$S' \to [S']S$	$\alpha_{0,0} \to [\alpha_{0,0}]A$	$\beta_{0,1} \to [\beta_{0,1}]A$	
Predict:	$S \to \alpha_{0,0}$	$A \to \beta_{0,0}$	$A \to \gamma_{0,0}$	
Scan:	$\beta_{0,0} \xrightarrow{a} \beta_{0,1}$	$\beta_{1,1} \xrightarrow{a} \beta_{1,2}$	$\gamma_{0,0} \xrightarrow{a} \gamma_{0,1}$	$\gamma_{1,0} \xrightarrow{b} \gamma_{1,1}$
Publish:	$\alpha_{0,2} \to ret$	$\beta_{1,2} \to ret$	$\gamma_{1,1} \to ret$	
Suspend:	$[\alpha_{0,1}]ret \to \alpha_{0,2}$	$[\beta_{1,0}]ret \to \beta_{1,1}$		
	$[\alpha_{0,0}]\beta_{0,2} \to \alpha_{0,1}[\beta_{0,2}]$	$[\alpha_{0,0}]\gamma_{0,1} \to \alpha_{0,1}[\gamma_{0,1}]$	$[\beta_{0,1}]\beta_{0,2} \to \beta_{0,2}[\beta_{0,2}]$	$[\beta_{0,1}]\gamma_{0,1} \to \beta_{0,2}[\gamma_{0,1}]$
Resume:	$\alpha_{0,1}[\beta_{0,2}] \to [\alpha_{0,1}]\beta_{1,0}$	$\alpha_{0,1}[\gamma_{0,1}] \to [\alpha_{0,1}]\gamma_{1,0}$	$\beta_{1,0}[\beta_{0,2}] \to [\beta_{1,0}]\beta_{1,0}$	$\beta_{1,0}[\gamma_{0,1}] \to [\beta_{1,0}]\gamma_{1,0}$

Figure 2: TA transitions for the LCFRS from figure 1

is an argument of the jth RHS non-terminal); and Θ is a finite set of transitions. Every transition has the form $\alpha \xrightarrow{a} \beta$ with $a \in T \cup \{\varepsilon\}$ and they roughly indicate that in the thread store, α can be replaced with β while scanning a. Square brackets in α and β indicate parts that do not belong to the active thread. This will be made more precise below. Θ contains the following transitions (see figure 2):

- **Call** transitions start a new thread, either for the start symbol or for a daughter non-terminal. They move down in the parse tree.
  $S' \to [S']S$ (initial call), $\gamma_{k,i} \to [\gamma_{k,i}]A$ if $A = rhs(\gamma, j-1)$ and $lhs(\gamma, k, i) = rhs(\gamma, j-1, 0)$ where $j = \delta(\gamma_{k,i})$.
- **Predict** transitions predict a new rule for a non-terminal A: $A \to \gamma_{0,0}$ if $A = lhs(\gamma)$.
- **Scan** reads a LHS terminal while scanning the next input symbol:
  $\gamma_{k,i} \xrightarrow{lhs(\gamma,k,i)} \gamma_{k,i+1}$ if $lhs(\gamma, k, i) \in T$.
- **Publish** marks the completion of a production, i.e., its full recognition:
  $\gamma_{k,j} \to ret$ if $dim(lhs(\gamma)) = k+1$ and $j = |lhs(\gamma, k)|$.
- **Suspend** suspends a daughter thread and resumes the parent. i.e., moves up in the parse tree. There are two cases:
  (i) The daughter is completely recognized: $[\gamma_{k,i}]ret \to \gamma_{k,i+1}$ if $lhs(\gamma, k, i) = rhs(\gamma, \delta(\gamma_{k,i})-1, dim(rhs(\delta(\gamma_{k,i})-1))-1)$.
  (ii) The daughter is not yet completely recognized, we have only finished one of its components: $[\gamma_{k,i}]\beta_{l,j} \to \gamma_{k,i+1}[\beta_{l,j}]$ if $dim(lhs(\beta)) > l+1$, $|lhs(\beta, l)| = j$, $lhs(\gamma, k, i) = rhs(\gamma, \delta(\gamma_{k,i}) - 1, l)$ and $rhs(\gamma, \delta(\gamma_{k,i}) - 1) = lhs(\beta)$.
- **Resume** resumes an already present daughter thread, i.e., moves down into some daughter that has already been partly recognized.
  $\gamma_{k,i}[\beta_{l,j}] \to [\gamma_{k,i}]\beta_{l+1,0}$ if $lhs(\gamma, k, i) = rhs(\gamma, \delta(\gamma_{k,i}) - 1, l+1)$, $rhs(\gamma, \delta(\gamma_{k,i}) - 1) = lhs(\beta)$ and $\beta_{l,j+1} \notin \mathcal{C}$.

This is not exactly the TA for LCFRS proposed in Villemonte de la Clergerie (2002) but rather the one from Kallmeyer (2010), which is close to the Earley parser from Burden and Ljunglöf (2005).

The set of configurations for a given input $w \in T^*$ is then defined by the deduction rules in figure 3 (the use of set union $\mathcal{S}_1 \cup \mathcal{S}_2$ in these rules assumes that $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$). The accepting state of the automaton for some input w is $\langle |w|, 1, \{\varepsilon : S', 1 : ret\}\rangle$.

2.3 LR Parsing

In an LR parser, the parser actions are guided by an automaton, resp. a *parse table* which is compiled offline. Consider the context-free case. An LR parser for CFG is a guided shift-reduce parser, in which we first build the LR automaton. Its states are sets of dotted productions closed under prediction, and its transitions correspond to having recognized a part of the input, e.g., to moving the dot over a RHS element after having scanned a terminal or recognized a non-terminal. Given an automaton with n states, we build the *parse table* with n rows. Each row i, $0 \leq i < n$, describes the possible parser actions associated with the state q_i, i.e., for each state and each possible shift or reduce operation, it tells us in which state to go after the operation.

3 LR for LCFRS

3.1 Intuition

The states in the automaton are predict and resume closures of TA thread stores. In order to keep them finite, we allow the addresses to be regular expressions. A configuration of the parser consists of a

Initial configuration: $\dfrac{}{\langle 0, \varepsilon, \{\varepsilon : S'\}\rangle}$ Initial call: $\dfrac{\langle 0, \varepsilon, \{\varepsilon : S'\}\rangle}{\langle 0, 1, \{\varepsilon : S', 1 : S\}\rangle}$

Further calls: $\dfrac{\langle i, p, \mathcal{S} \cup p : \gamma_{k,i}\rangle}{\langle i, pj, \mathcal{S} \cup p : \gamma_{k,i} \cup pj : A\rangle}$ $\gamma_{k,i} \to [\gamma_{k,i}]A \in \Theta,\ A \in N, \delta(\gamma_{k,i}) = j+1$ Predict: $\dfrac{\langle i, p, \mathcal{S} \cup p : A\rangle}{\langle i, p, \mathcal{S} \cup p : \gamma_{0,0}\rangle}$ $A \in N,\ A \to \gamma_{1,0} \in \Theta$

Scan: $\dfrac{\langle j, p, \mathcal{S} \cup p : \gamma_{k,i}\rangle}{\langle j+1, p, \mathcal{S} \cup p : \gamma_{k,i+1}\rangle}$ $\gamma_{k,i} \overset{w_{j+1}}{\to} \gamma_{k,i+1} \in \Theta$ Publish: $\dfrac{\langle i, p, \mathcal{S} \cup \{p : \gamma_{k,i}\}\rangle}{\langle i, p, \mathcal{S} \cup \{p : ret\}\rangle}$ $\gamma_{k,j} \to ret \in \Theta$

Suspend 1: $\dfrac{\langle i, pj, \mathcal{S} \cup \{p : \gamma_{k,i}, pj : ret\}\rangle}{\langle i, p, \mathcal{S} \cup \{p : \gamma_{k,i+1}\}\rangle}$ $[\gamma_{k,i}]ret \to \gamma_{k,i+1} \in \Theta$

Suspend 2: $\dfrac{\langle i, pj, \mathcal{S} \cup \{p : \gamma_{k,i}, pj : \beta_{l,m}\}\rangle}{\langle i, p, \mathcal{S} \cup \{p : \gamma_{k,i+1}, pj : \beta_{l,m}\}\rangle}$ $[\gamma_{k,i}]\beta_{l,m} \to \gamma_{k,i+1}[\beta_{l,m}] \in \Theta$

Resume: $\dfrac{\langle i, p, \mathcal{S} \cup \{p : \gamma_{k,i}, p\delta(\gamma_{k,i}) : \beta_{l,j}\}\rangle}{\langle i, p\delta(\gamma_{k,i}), \mathcal{S} \cup \{p : \gamma_{k,i}, p\delta(\gamma_{k,i}) : \beta_{l+1,0}\}\rangle}$ $\gamma_{k,i}[\beta_{l,j}] \to [\gamma_{k,i}]\beta_{l+1,0} \in \Theta$

Figure 3: Deduction rules for TA configurations

stack, a set of completed components and the remaining input. The completed components are of the form $p : \gamma_i$ where p is an address and γ_i the component of a rule. The stack has the form $\Gamma_1 x_1 \Gamma_2 \dots x_{n-1} \Gamma_n$ where Γ_i is an address followed by a state and $x_i \in T \cup \{A_k \,|\, A \in N, 1 \le k \le dim(A)\}$.

Shift: Whenever we have $p : q$ on top of the stack and an edge from q to q' labeled with the next input symbol and an address p', we add the input symbol followed by $pp' : q'$ to the stack.

Suspend: Whenever the top of the stack is $p_1 : q$ such that there is a $\gamma_{i-1,k} \in q$ with $k = |lhs(\gamma, i-1)|$ and $i < dim(\gamma)$, we can suspend. If $i = 1$, we add $p_1 : \gamma_i$ to the set of completed components and we remove $|lhs(\gamma, i)|$ terminals/component nonterminals and their preceding states from the stack. If $i \ge 1$, we check whether there is a $p_2 : \gamma_{i-1}$ in the set of completed components such that the intersection $L(p_1) \cap L(p_2)$ is not empty.[2] We then remove $p_2 : \gamma_{i-1}$ from the set of complete components and we add $p : \gamma_i$ to it where p is a regular expression denoting $L(p_1) \cap L(p_2)$. Suppose the topmost state on the stack is now $p' : q'$. We then have to follow the edge leading from q' to some q'' labeled $A_i : p''$ where $A = lhs(\gamma)$. This means that we push A_i followed by $p'p'' : q''$ on the stack.

[2]Note that the corresponding finite state automata can be deterministic; in this case the intersection is quadratic in the size of the two automata.
In LCFRS without left recursion in any of the components, the intersection is trivial since the regular expressions denote only a single path each.

Reduce: Whenever there is a $\gamma_{i-1,k}$ in our current state with $k = |lhs(\gamma, i-1)|$ and $i = dim(\gamma)$, we can reduce, which is like suspend except that nothing is added to the set of completed components.

3.2 Automaton and parse table construction

The states of the LR-automaton are sets of pairs $p : X$ where p is a regular expression over $\{1, \dots, m\}$, m the rank of G, and $X \in \mathcal{C} \cup \{S'\}$. They represent predict and resume closures.The predict/resume closure $\overline{q}$ of some set q is described by the deduction rules in figure 4. This closure is not always finite.

$$\frac{\varepsilon : S'}{1 : \alpha_{0,0}} \quad lhs(\alpha) = S$$

$$\frac{p : \gamma_{i,j}}{pk : \gamma'_{l,0}} \quad \begin{array}{l} lhs(\gamma, i, j) = rhs(\gamma, k-1, l), \\ rhs(\gamma, k) = lhs(\gamma') \end{array}$$

Figure 4: Predict/resume closure

However, if it is not, we obtain a set of items that can be represented by a finite set of pairs $r : \gamma_{i,j}$ plus eventually $\varepsilon : S'$ such that r is a regular expression denoting a set of possible addresses. As an example for such a case, see q_3 in figure 5.

The reason why we can represent these closures by finite sets using regular expressions for paths is the following: There is a finite number of possible elements $\gamma_{i,j}$. For each of these, the set of possible addresses it might be combined with in a state that is the closure of $\{\varepsilon : X_1, \varepsilon : X_2, \dots, \varepsilon : X_n\}$ is generated by the CFG $\langle \mathcal{C} \cup \{S'\} \cup \{S_{new}\}, \{1, \dots, m\}, P, S_{new}\rangle$ with $S_{new} \to X_i \in P$ for all $1 \le i \le n$, $X \to Yk \in P$ for all in-

stances $\frac{p:X}{pk:Y}$ of deduction rules and $\gamma_{i,j} \to \varepsilon$. This is a regular grammar, its string language can thus be characterized by a regular expression.

The construction of the set of states starts with $q_0 = \overline{\{\varepsilon : S'\}}$. For every state q, every nonterminal A and every $1 \leq i \leq dim(A)$, we define $read(q, A_i, p) = \{\varepsilon : \gamma_{j,k+1} \,|\, p : \gamma_{j,k} \in q$ and there is some l such that $rhs(\gamma, l) = A$ and $lhs(\gamma, j, k) = rhs(\gamma, l, i{-}1)\}$ and $\overline{read}(q, A_i, p) = \overline{read(q, A_i, p)}$. Similarly, for every such q and every $a \in T$, we define $read(q, a, p) = \{\varepsilon : \gamma_{j,k+1} \,|\, p : \gamma_{j,k} \in q$ and $lhs(\gamma, j, k) = a\}$ and $\overline{read}(q, a, p) = \overline{read(q, a, p)}$. The set of states of our automaton is then the closure of $\{q_0\}$ under the application of the $\overline{read}$-functions. The edges in our automaton correspond to $\overline{read}$-transitions, where each edge is labeled with the corresponding pair A_i, p or a, p respectively. The automaton we obtain for the grammar in figure 1 is shown in figure 5. The number of possible states

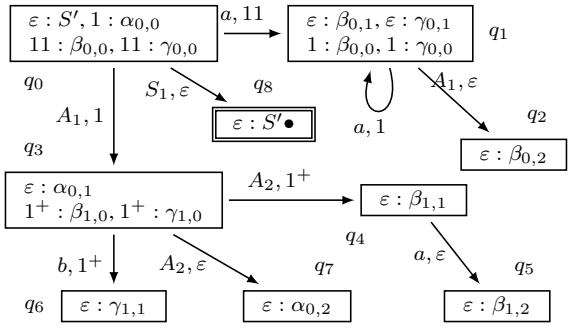

Figure 5: The automaton

is necessarily finite since each state is the closure of some set containing only items with address ε. There are only finitely many such sets.

In the parse table, our operations are $s(p, q)$ for shifting some terminal a followed by the old address concatenated with p and state q and $r(\alpha, i)$ for reducing the ith component of rule α. The two reduce operations can be distinguished by the component indices. Furthermore, the goto-part of the table tells where to go when traversing a component edge and which address to add then. The parse table can be read off the automaton as follows: $action(q, a) = s(p, q')$ iff $\overline{read}(q, a, p) = q'$; $action(q, -) = r(\gamma, i)$ iff there is some $p : \gamma_{i,k} \in q$ such that $k = |lhs(\gamma, i)|$. Concerning the goto part of the table, we have $goto(q, A_i) = \langle p, q' \rangle$ iff $\overline{read}(q, A_i, p) = q'$. Figure 6 shows the parse table

	a	b		A_1	A_2	S_1
0	$s(11,1)$			$\langle 1,3\rangle$		$\langle \varepsilon,8\rangle$
1	$s(1,1)$		$r(\gamma,1)$	$\langle \varepsilon,2\rangle$		
2			$r(\beta,1)$			
3		$s(1^+,6)$			$\langle 1^+,4\rangle$, $\langle \varepsilon,7\rangle$	
4	$s(\varepsilon,5)$					
5			$r(\beta,2)$			
6			$r(\gamma,2)$			
7			$r(\alpha,1)$			
8			acc			

Figure 6: The parse table

stack	**completed**	**input**	**operation**
ε:q_0	[ ]	$aaba$	initial state
ε:q_0 a 11:q_1	[ ]	aba	shift a,11
ε:q_0 a 11:q_1 a 111:q_1	[ ]	ba	shift a,1
ε:q_0 a 11:q_1 A_1 11:q_2	[111:γ_1]	ba	suspend $\gamma_{0,1}$
ε:q_0 A_1 1:q_3	[111:γ_1,11:β_1]	ba	suspend $\beta_{0,2}$
ε:q_0 A_1 1:q_3 b 11^+:q_6	[111:γ_1,11:β_1]	a	shift b,1^+
ε:q_0 A_1 1:q_3 A_2 11^+:q_4	[11:β_1]	a	reduce $\gamma_{1,1}$
ε:q_0 A_1 1:q_3 A_2 11^+:q_4 a 11^+:q_5	[11:β_1]	ε	shift a,ε
ε:q_0 A_1 1:q_3 A_2 1:q_4	[ ]	ε	reduce $\beta_{1,2}$
ε:q_0 S_1 ε:q_8	[ ]	ε	reduce $\alpha_{0,2}$

Figure 7: Sample run with $w = aaba$

for our example.

3.3 Parsing

We run the automaton with $\langle \varepsilon : q_0, [\,], w \rangle$ and input $w = aaba$. The trace is shown in figure 7. We start in q_0, and shift two as, which leads to q_1. We have then fully recognized the first components of γ and β: We suspend them and keep them in the set of completed components, which takes us to q_3. Shifting the b takes us to q_6, from where we can reduce, which finally takes us to q_4. From there, we can shift the remaining a (to q_5), with which we have fully recognized β. We can now reduce both β and with that, α, which takes us to the accepting state q_8.

4 Conclusion

We presented the first LR style algorithm for LCFRS parsing. It offers a convenient factorization of predict/resume operations. We are currently exploring the possibility to use it in data-driven parsing.

Acknowledgments

The work presented in this paper was partly funded by the German Research Foundation (DFG). We wish to thank three anonymous reviewers for their valuable comments.

References

Krasimir Angelov and Peter Ljunglöf. 2014. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg, Sweden.

Jan A. Botha and Phil Blunsom. 2013. Adaptor grammars for learning non-concatenative morphology. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 345–356, Seattle, WA.

Pierre Boullier. 1998. Proposal for a Natural Language Processing syntactic backbone. Research Report 3342, INRIA-Rocquencourt, Rocquencourt, France.

Håkan Burden and Peter Ljunglöf. 2005. Parsing linear context-free rewriting systems. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 11–17, Vancouver, BC.

Miriam Kaeshammer. 2013. Synchronous linear context-free rewriting systems for machine translation. In *Proceedings of the Seventh Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 68–77, Atlanta, GA.

Laura Kallmeyer and Wolfgang Maier. 2009. An incremental Earley parser for simple range concatenation grammar. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 61–64, Paris, France.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Laura Kallmeyer. 2010. *Parsing beyond Context-Free Grammar*. Springer, Heidelberg.

Donald E. Knuth. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, July.

Marco Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387.

Wolfgang Maier and Laura Kallmeyer. 2010. Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+10)*, pages 119–126, New Haven, CT.

Wolfgang Maier and Timm Lichte. 2011. Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected Papers*, volume 5591 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Berlin, Heidelberg, New York. Springer-Verlag.

Mark-Jan Nederhof. 1998. An alternative LR algorithm for TAGs. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 946–952, Montreal, QC.

Carlos A. Prolo. 2003. *LR Parsing for Tree Adjoining Grammars and its Application to Corpus-based Natural Language Parsing*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

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

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Anders Søgaard. 2008. Range concatenation grammars for translation. In *The 22nd International Conference on Computational Linguistics (COLING)*, pages 103–106, Manchester, England.

Masaru Tomita. 1984. LR parsers for natural languages. In *Proceedings of COLING 1984: The 10th International Conference on Computational Linguistics*, pages 354–357, Stanford University.

Andreas van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon, France.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA.

Éric Villemonte de la Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *Proceedings of COLING 2002: The 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylviania, Philadelphia, PA.