Everything You Always Wanted to Know About Parsing

Part V: LR Parsing

Giorgio Satta
University of Padua, Italy

ESSLLI, August 2013
**Introduction**

*Parsing strategies* classified by the time the associated PDA commits to a production in the *analysis*:

\[ A \rightarrow B \beta \]

- **Top-Down**
- **Bottom-Up**
- **Left-Corner**

Giorgio Satta  
Everything About Parsing
Introduction

**LR parsing** discovered in 1965 in an article by Donald Knuth, California Institute of Technology, Pasadena, California

‘L’ stands for **left-to-right parsing** and ‘R’ stands for **reversed rightmost derivation**

Works in **linear time** for **deterministic context-free languages**

The algorithm consists of a shift-reduce, deterministic PDA called **LR PDA**, realizing a **bottom-up** parsing strategy

Widely used for parsing of **programming languages**, where LR PDAs are **mechanically generated** from a formal grammar
**Introduction**

Generalized LR parsing (GLR for short) extends the LR method to handle **nondeterministic and ambiguous** CFGs.


Very popular in **computational linguistics** in the late eighties.
Introduction

The GLR algorithm consists of

- the **LR PDA** as in Knuth original proposal
- the definition of a technique for **simulating** the LR PDA, based on a data structure called **graph structured stack**

We **disregard** the graph structured stack, and apply our **tabulation** algorithm to simulate the LR PDA

The outcome is an **optimized version** of the original GLR parsing algorithm
Let $P$ be a set of CFG productions; a stack symbol of the LR PDA is a non-empty subset of

$$I_P = \{ A \rightarrow \alpha_1 \bullet \alpha_2 \mid (A \rightarrow \alpha_1 \alpha_2) \in P \}$$

satisfying some additional property, to be introduced later.

Informally, a stack symbol represents a disjunction of dotted rules; in this way several alternative parsing analyses can be carried out simultaneously.
**Example**: Stack symbol \( q = \{ A \rightarrow B \ C \cdot \gamma, \ A' \rightarrow C \cdot \gamma' \} \) represents two *alternative analyses*; observe how the strings preceding the dots are *suffixes of a common string*.
We need a function \textit{closure} to precompute the \textit{closure of the prediction step} from the Earley algorithm.

Let \( q \subseteq I_P \); \( \text{closure}(q) \) is the \textit{smallest set} such that:

- \( q \subseteq \text{closure}(q) \); \textbf{and}

- if \((A \rightarrow \alpha \bullet B \alpha') \in \text{closure}(q)\) and \((B \rightarrow \beta) \in P\) then \((B \rightarrow \bullet \beta) \in \text{closure}(q)\)
**Example:**

If \( P \) consists of the **productions**:

\[
S \rightarrow S + S,  \\
S \rightarrow a
\]

we have

\[
\text{closure}(\{S \rightarrow S + \bullet S\}) =  \\
\{S \rightarrow S + \bullet S, \ S \rightarrow \bullet S + S, \ S \rightarrow \bullet a\}
\]
Stack Symbols

We need a function \texttt{goto} to precompute the \texttt{scanner} and the \texttt{completer} steps from the Earley algorithm, followed by closure() .

Let $q \subseteq \mathcal{I}_P$ and $X \in N \cup \Sigma$; we have

$$\text{goto}(q, X) = \text{closure}(\{ A \rightarrow \alpha X \cdot \alpha' \mid (A \rightarrow \alpha \cdot X \alpha') \in q \})$$
Example (cont’d):

If \( P \) consists of the productions:

\[
\begin{align*}
S & \rightarrow S + S , \\
S & \rightarrow a \\
\end{align*}
\]

we have

\[
\text{goto}(\{S \rightarrow S + \bullet S, \ S \rightarrow \bullet S + S, \ S \rightarrow \bullet a\}, \ a) = \{S \rightarrow a \bullet\}
\]

\[
\text{goto}(\{S \rightarrow S + \bullet S, \ S \rightarrow \bullet S + S, \ S \rightarrow \bullet a\}, \ S) = \{S \rightarrow S + S \bullet, \ S \rightarrow S \bullet + S\}
\]
The **initial stack symbol** of the LR PDA is

\[ q_{in} = \text{closure}(\{ S \rightarrow \bullet \alpha \mid (S \rightarrow \alpha) \in P \}) \]

The **final stack symbol** of the LR PDA is a **special symbol** \( q_{fin} \);
The **stack alphabet** $Q_{LR}$ of the LR PDA is the **smallest set** satisfying:

- $q_{in} \in Q_{LR}$;
- $q_{fin} \in Q_{LR}$; and
- if $q \in Q_{LR}$ and $\text{goto}(q, X) \neq \emptyset$ for some $X \in N \cup \Sigma$, then $q' \in Q_{LR}$.
Stack Symbols

Example (cont’d):

\( P \) consists of the productions \( S \rightarrow S + S \) and \( S \rightarrow a \)

The goto function, also called the LR table, excluding \( q_{\text{fin}} \):

By construction, in each state the strings preceding the dots are suffixes of a common string
The **LR PDA** $M_{LR}$ is constructed from $G$ as follows

The **input alphabet** of $M_{LR}$ is the terminal alphabet of $G$.

The **stack alphabet** of $M_{LR}$ is $Q_{LR}$, as previously defined.
Set $\Delta_{LR}$ contains transitions having one of the following forms:

- **shift**

  $q_1 \xrightarrow{a} q_1 q_2$

  for each $q_1, q_2$ and $a$ with $q_2 = \text{goto}(q_1, a)$

- **reduce**

  $q_0 q_1 \cdots q_m \xrightarrow{\varepsilon} q_0 q'$

  for each $q_0, q_1, \cdots, q_m, q'$ and $A \rightarrow \alpha$ with $(A \rightarrow \alpha \bullet) \in q_m$, $|\alpha| = m$ and $q' = \text{goto}(q_0, A)$

- **reduce** (special case)

  $q_{\text{in}} q_1 \cdots q_m \xrightarrow{\varepsilon} q_{\text{fin}}$

  for each $q_1, \cdots, q_m$ and $S \rightarrow \alpha$ with $(S \rightarrow \alpha \bullet) \in q_m$ and $|\alpha| = m$
Each stack symbol of $M_{LR}$ records the set of rules that are **compatible** with the sequence of nonterminals recognized so far.

Thus **alternative analyses** within a stack symbol match suffixes of the sequence of nonterminals recognized so far; we call this the **suffix property**.

The **implemented strategy** is **bottom-up**: the **commitment** to a production occurs **as late as possible**, after all of the production’s right-hand side symbols have been recognized.
A production $A \rightarrow \alpha$ has **exactly** $|A\alpha|$ dotted items; therefore

$$|\mathcal{I}_P| = |G|$$

It is not difficult to construct a **worst case** set $P$ such that

$$|Q_{LR}| = \mathcal{O}(\exp(|G|))$$

$$|M_{LR}| = \mathcal{O}(\exp(|G|))$$
Example (cont’d) : input string $w = a + a + a$
Example (cont’d) : input string $w = a + a + a$, **first** computation

- $q_{in} \xrightarrow{a} q_1$
- $q_{in} \xrightarrow{\epsilon} q_{in}$
- $q_{in} \xrightarrow{+} q_{in}$

Reducing $S \rightarrow a$

- $q_{in}$
- $q_1$
- $q_{in}$
- $q_{in}$
- $q_3$

States: 0, 1, 2
Example (cont’d) : input string \( w = a + a + a \), **first** computation

\[
\begin{align*}
\text{reduce } S & \rightarrow a \\
\text{reduce } S & \rightarrow S + S
\end{align*}
\]

\[
\begin{array}{c}
\begin{array}{c}
q_1 \\
q_3 \\
q_2 \\
q_{in}
\end{array} & \rightarrow_a & \begin{array}{c}
q_4 \\
q_3 \\
q_2 \\
q_{in}
\end{array} \\
\qquad & \xrightarrow{\varepsilon} & \qquad \\
\begin{array}{c}
q_{in} \\
q_{in} \\
q_{in} \\
q_{in}
\end{array} & \xrightarrow{+} & \begin{array}{c}
q_3 \\
q_2 \\
q_2 \\
q_{in}
\end{array}
\end{array}
\]

\[
\begin{array}{cccc}
3 & 3 & 3 & 4
\end{array}
\]
Example (cont’d) : input string \( w = a + a + a \), \textcolor{red}{\textbf{first}} computation

\texttt{reduce} \( S \to a \) \hspace{1cm} \texttt{reduce} \( S \to S + S \)

\begin{align*}
\text{q1} & \quad \text{q4} \\
\text{q3} & \quad \text{q3} \\
\text{q2} & \quad \text{q2} \\
\text{q_{in}} & \quad \text{q_{in}} \quad \quad \text{q_{fin}}
\end{align*}

\begin{align*}
\rightarrow a & \quad \rightarrow \varepsilon \quad \varepsilon \quad \varepsilon
\end{align*}

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
</table>

Giorgio Satta | Everything About Parsing
Example (cont’d): input string $w = a + a + a$, second computation

- **$M_{LR}$ Computation**

---

- **Example (cont’d): input string $w = a + a + a$, second computation**

  - **Reduce $S \rightarrow a$**
    - **Initial state:** $q_{in}$
    - **Input symbol:** $a$
    - **Next state:** $q_1$
    - **Reduce state:** $q_2$
    - **Input symbol:** $\varepsilon$
    - **Next state:** $q_3$
    - **Input symbol:** $+$
    - **Next state:** $q_{in}$

---

- **States and Transitions**
  - **State $q_{in}$**
  - **Transition $\rightarrow_a$**
  - **State $q_1$**
  - **Transition $\rightarrow_\varepsilon$**
  - **State $q_2$**
  - **Transition $\rightarrow_+$**
  - **State $q_3$**

---

- **Timeline:**
  - **Time $0$**
  - **Time $1$**
  - **Time $2$**
  - **Time $3$**

---

- **Giorgio Satta**
- **Everything About Parsing**
M_{LR} Computation

\[ M_{LR} \text{ Computation} \]

\[ \text{reduce } S \rightarrow a \]

\[ q_1 \rightarrow_a q_{in} \]
\[ q_3 \rightarrow \varepsilon q_{in} \]
\[ q_2 \rightarrow + q_{in} \]
\[ q_1 \rightarrow a q_{in} \]
\[ q_3 \rightarrow a q_{in} \]
\[ q_4 \rightarrow a q_{in} \]
\[ q_4 \rightarrow a q_{in} \]
\[ q_3 \rightarrow a q_{in} \]
\[ q_4 \rightarrow a q_{in} \]

3 3 4 5
\( M_{LR} \) Computation

\[\text{reduce } S \rightarrow a\]

\[\begin{align*}
q_4 \\
q_3 \\
q_2 \\
q_{in}
\end{align*}\]

\[\text{reduce } S \rightarrow S + S\]

\[\begin{align*}
q_4 & \quad \text{reduce } S \rightarrow S + S \\
q_4 \\
q_3 \\
q_2 \\
q_{in} & \quad \text{reduce } S \rightarrow S + S \\
q_{fin}
\end{align*}\]
**Example** : Earley PDA on sample rule $A_0 \rightarrow A_1 A_2 A_3$ (simplified notation)
$M_E$ and $M_{LR}$

**Example (cont’d)**: LR PDAs on sample rule $A_0 \rightarrow A_1A_2A_3$
(simplified notation)

\[
\begin{array}{cccc}
\{\bullet A_1A_2A_3\} & \{ A_1 \bullet A_2A_3 \} & \{ A_1A_2 \bullet A_3 \} & \{ A_1A_2A_3\bullet \} \\
\vdots & \vdots & \vdots & \vdots \\

i_1 & i_2 & i_3 & i_4
\end{array}
\]
Example (cont’d) : \( M_E \) tabulation on sample rule \( A_0 \rightarrow A_1 A_2 A_3 \)

\[
\begin{array}{c}
A_1 \\
i_1 \\
\end{array} \leftarrow \begin{array}{c}
A_2 \\
i_2 \\
\end{array} \leftarrow \begin{array}{c}
A_3 \\
i_3 \\
\end{array} \leftarrow \begin{array}{c}
A_1 A_2 A_3 \\
i_4 \\
\end{array}
\]

No need to reduce
Example (cont’d) : $M_{LR}$ tabulation on sample rule $A_0 \rightarrow A_1 A_2 A_3$

Single step reduction should be avoided; this issue will be addressed later.
$M_{LR}$ is **deterministic** only for so-called **deterministic context-free languages**

For general CFGs, $M_{LR}$ is **strictly nondeterministic**, with an **exponential** proliferation of computations in the length of the input string.

But this needs not concern us:

- the only purpose of $M_{LR}$ is to specify a **parsing strategy**
- nondeterminism can be **simulated** using (an extension of) our deterministic tabulation algorithm
We derive the **GLR algorithm** by **tabulation** of the LR PDA

**Items** in the GLR algorithm have the form

$$(q, j, q', i)$$

In this case we **can not drop** the first item component as we did for the CKY and Earley algorithm
Deduction Rules

We overview the GLR tabular algorithm using inference rules.

$M_{LR}$ starts with $q_{in}$ in the stack. This corresponds to the init step of the tabular algorithm.

\[
(\bot, 0, q_{in}, 0)
\]

\[
(\bot, q_{in})
\]
Deduction Rules

The **shift** transition \( q_1 \xrightarrow{a} q_1 q_2 \) can be implemented as

\[
\frac{(q', j, q_1, i - 1)}{(q_1, i - 1, q_2, i)} \quad \{ a = a_i \}
\]
Deduction Rules

The **reduce** transition

\[(q_0q_1\cdots q_{m-1}q_m \xrightarrow{\epsilon} q_0q')\]

is in a form **more general** than the reduce transition available for our tabulation algorithm.

This reduction can be **implemented** as

\[(q_0, j_0, q_1, j_1)
(q_1, j_1, q_2, j_2)
\vdots
(q_{m-1}, j_{m-1}, q_m, j_m)
\underline{(q_0, j_0, q', j_m)}\]
The deduction rule for the **reduce** transition

$$(q_0q_1\cdots q_{m-1}q_m \xrightarrow{\varepsilon} q_0q')$$

can be **graphically represented** as

![Diagram showing the deduction rule for the reduce transition](image-url)
Algorithm 5 GLR Algorithm (from Tabulation of LR PDA)

1: $\mathcal{T} \leftarrow \{(\bot, 0, q_{in}, 0)\}$
2: for $i \leftarrow 1, \ldots, n$ do ▶ edges with right end at $i$
3: $\mathcal{D} \leftarrow \emptyset$
4: for each $(q', j, q_1, i-1) \in \mathcal{T}$ do
5:   for each $(q_1 \xrightarrow{a_i} q_1 q_2) \in \Delta$ do ▶ shift step
6:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{(q_1, i-1, q_2, i)\}$
7:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(q_1, i-1, q_2, i)\}$
8: end for
9: end for
Pseudocode

10: while $D \neq \emptyset$ do
11:     pop $(q_{m-1}, j_{m-1}, q_m, j_m)$ from $D$
12:     for each $(q_0 q_1 \cdots q_m \xrightarrow{\varepsilon} q_0 q') \in \Delta$ do ▷ reduce step
13:         for each $(q_0, j_0, q_1, j_1), (q_1, j_1, q_2, j_2), \ldots,$
14:             $(q_{m-2}, j_{m-2}, q_{m-1}, j_{m-1}) \in T$ do
15:                 if $(q_0, j_0, q', j_m) \notin T$ then
16:                     $T \leftarrow T \cup \{(q_0, j_0, q', j_m)\}$
17:                 end if
18:             end for
19:         end for
20:     end for
21: end while
Pseudocode

20: for each \((q_inq_1 \cdots q_m \stackrel{\varepsilon}{\longrightarrow} q_{fin}) \in \Delta\) do  \(\triangleright\) reduce step
21: for each \((\perp, 0, q_in, 0), (q_in, 0, q_1, j_1), \ldots, (q_{m-2}, j_{m-2}, q_{m-1}, j_{m-1}) \in T\) do
22: if \((\perp, 0, q_{fin}, j_m) \notin T\) then
23: \(T \leftarrow T \cup \{(\perp, 0, q_{fin}, j_m)\}\)
24: \(D \leftarrow D \cup \{(\perp, 0, q_{fin}, j_m)\}\)
25: end if
26: end for
27: end for
28: end while
29: end for
30: accept iff \((\perp, 0, q_{fin}, n) \in T\)
There are two different sources of complexity in the GLR algorithm.

The first is related to the worst case exponential size of the set of stack symbols $Q_{LR}$; see our previous analysis.

This effect can be alleviated by using optimization techniques that simplify item representation for certain rules.
The second source of complexity is related to the length of the input string; let $m$ be the length of the longest rule in the grammar.

The reduce step involves $m + 1$ indices and can be instantiated a number of times $O(|w|^{m+1})$, resulting in exponential time complexity in the grammar size.

Such an exponential behavior can be avoided by factorizing each reduce transition into a sequence of pop transitions with only two stack symbols in the left-hand side.