Everything You Always Wanted to Know About Parsing

Part I : Background

Giorgio Satta
University of Padua, Italy

ESSLLI, August 2013
Tabular Parsing

**Input**: grammar \( G \) and string \( w \)

- **recognition**: deciding whether grammar \( G \) generates string \( w \)
- **parsing**: constructing all derivations/trees assigned by grammar \( G \) to string \( w \)
- **Viterbi**: provide the highest probability derivation/tree assigned by probabilistic grammar \( G \) to string \( w \)

**Tabular algorithms** for the above problems are all **closely related**
Tabular Parsing

Tabular algorithms use **dynamic programming**:

- input instance broken into all possible **sub-instances** of the problem
- solutions of sub-problems are **stored** and **recombined** to obtain solution of the original instance

Tabular algorithms allow **representation sharing**, in contrast with approaches that explore one solution at a time and use **backtracking**
Tabular Parsing

Tabular algorithms are the most widely used methods for parsing based on phrase-structure grammars:

- **context-free** grammars (CFGs)
- **mildly context-sensitive** grammars
- **synchronous** grammars

Tabular algorithms are also used for:

- parsing of **dependency** grammars, both projective and non-projective
- induction of **probabilistic** grammars, e.g., the **inside/outside** algorithm
Most popular tabular algorithms for CFG parsing:

- Cocke-Kasami-Younger algorithm [Younger, 1967]
- Earley algorithm [Earley, 1970]
- Graham, Harrison and Ruzzo algorithm [Graham et al., 1980]
- Chart parsing [Kay, 1986]
- Generalized LR algorithm [Tomita, 1986]
- Left-corner parsing [Leiss, 1990]
Questions often asked by students:

- how do Shift-Reduce, Earley, Left-Corner and LR parsing strategies compare one another?
- what is the relation between the chart data structure and the graph-structured stack?
- can we do LR parsing with a chart data structure?
- how can we design a new parsing strategy and implement it using tabular techniques?

Relations among tabular parsing algorithms are not well documented in the existing literature.
CFG Parsing

We present a framework for the specification of CFG tabular parsers, due to [Lang, 1974], that can be used to:

- investigate deep similarities/differences between existing algorithms
- design new parsing algorithms
- derive formal properties as correctness and computational complexity

The framework is based on the notions of

- push-down automata
- tabular simulation of nondeterminism
CFG Parsing

- Shift-Reduce
- Earley PDA
- LC PDA
- LR PDA
- Tabulation
- CKY
- Earley
- Gen LC
- Gen LR
A CFG is a **declarative description** of the structure underlying sentences in a language: there is no specification of how strings in the language should be processed.

A PDA is a **procedural specification** of how the structure underlying sentences in a language should be tested.

The latter notion is called **parsing strategy**; we therefore consider PDAs as convenient ways of specifying parsing strategies for context-free languages.
In the traditional presentation of CFG parsing algorithms, two independent concepts are always mixed up:

- parsing strategy
- algorithm realization

The tabulation approach allows a clean separation of parsing strategy and algorithm realization.

This separation is an important conceptual advantage in the design, realization and investigation of parsing algorithms.
Some **basic notation** for **sets**:

- for **non-negative** integers $i$ and $j$ with $i \leq j$

  $$[i, j] = \{i, i + 1, \ldots, j\}$$

- when $i > j$ we assume $[i, j] = \emptyset$
Binary Relations

The **Cartesian product** of sets $\Delta$ and $\Delta'$ is:

$$\Delta \times \Delta' = \text{def} \quad \{(d, d') \mid d \in \Delta, d' \in \Delta'\}$$

A **binary relation** $R$ over set $\Delta$ is a set:

$$R \subseteq (\Delta \times \Delta)$$
Some **standard operators** for binary relations:

\[
\begin{align*}
R \circ R' & = \text{def} \quad \{(d, d') \mid (d, d'') \in R, (d'', d') \in R'\} \\
R^0 & = \text{def} \quad \{(d, d) \mid d \in \Delta\} \quad \text{(identity relation)} \\
R^{i+1} & = \text{def} \quad R \circ R^i, \ i \geq 0 \\
R^* & = \text{def} \quad \bigcup_{i \geq 0} R^i \\
R^+ & = \text{def} \quad \bigcup_{i \geq 1} R^i
\end{align*}
\]
**Strings**

**Alphabets and strings**:

- an *alphabet* is a **finite** set of *atomic* symbols
- a *string* is a **finite** sequence of tokens from some alphabet
Some **basic notation** for strings:

- for a string $x$, $|x|$ denotes the **length** of $x$
- symbol $\varepsilon$ denotes the **empty string**, with $|\varepsilon| = 0$
- $\Sigma^*$ denotes the set of all strings over alphabet $\Sigma$
- $x \cdot y$, also written $xy$, is the **concatenation** of strings $x$ and $y$

**String indices**:

- $j$ is a fence position, or **index**, for string $x$ if $j \in [0, |x|]$
A set of strings $L \subseteq \Sigma^*$ is called a **language**

Some **standard operators** for languages:

\[
\begin{align*}
L \cdot L' &= \text{def} \quad \{ x \cdot y \mid x \in L, \ y \in L' \} \\
L^0 &= \text{def} \quad \{ \varepsilon \} \\
L^{i+1} &= \text{def} \quad L \cdot L^i, \ i \geq 0 \\
L^* &= \text{def} \quad \bigcup_{i \geq 0} L^i \\
L^+ &= \text{def} \quad \bigcup_{i \geq 1} L^i
\end{align*}
\]
$f(n) = \mathcal{O}(g(n))$ if and only if there are $n_0$ and $c$ such that, for every $n \geq n_0$, we have $0 \leq f(n) \leq c \cdot g(n)$

$f(n) = \Theta(g(n))$ if and only if there are $n_0$, $c$ and $c'$ such that, for every $n \geq n_0$, we have $0 \leq c \cdot g(n) \leq f(n) \leq c' \cdot g(n)$
A context-free grammar, or CFG for short, is a tuple

\[ G \equiv \text{def} \ (N, \Sigma, P, S) \]

where:

- \( N \) is a finite set of nonterminal symbols
- \( \Sigma \) is a finite set of terminal symbols; we assume \( N \cap \Sigma = \emptyset \)
- \( S \in N \) is the start symbol
- \( P \) is a finite set of productions, each having the form

\[ A \rightarrow \alpha \]

with \( A \in N \) and \( \alpha \in (N \cup \Sigma)^* \)
Some conventions for CFGs:

- $V = N \cup \Sigma$
- $A, B, C, \ldots \in N$
- $a, b, c, \ldots \in \Sigma$
- $X, Y, \ldots \in \Sigma \cup \{\varepsilon\}$
- $\alpha, \beta, \gamma, \ldots \in V^*$
- $u, v, w, x, \ldots \in \Sigma^*$
Rewriting

A **rewrite step** in a CFG is the **application** of a production to a string in $V^*$. This is represented through a binary relation called **rewrite**

If $p = (A \rightarrow \alpha) \in P$ we write

$$\gamma A \delta \Rightarrow \gamma \alpha \delta$$

Sometime also written as $\gamma A \delta \xrightarrow{P} \gamma \alpha \delta$

**Leftmost rewrite** relation used when the rewritten nonterminal must be the first from the left

$$xA \delta \Rightarrow_{lm} x\alpha \delta$$
A **derivation** in a CFG is a **finite** sequence of rewrite steps. We use relations $\Rightarrow^*$ and $\Rightarrow^+$ to represent derivations.

We **specialize** the notation for derivations: If

$$\gamma_0 \xrightarrow{p_1} \gamma_1, \gamma_1 \xrightarrow{p_2} \gamma_2, \ldots, \gamma_{q-1} \xrightarrow{p_q} \gamma_q$$

we write

$$\gamma_0 \xrightarrow{p_1 \cdots p_q} \gamma_q$$

If $\pi = p_1 \cdots p_q$ we also write

$$\gamma_0 \xrightarrow{\pi} \gamma_q$$
Some additional notions for CFGs:

- if $S \xrightarrow{\pi} \alpha$, $\alpha$ is called a **sentential** form
- if $S \xrightarrow{\pi} w$, $w$ is called a **sentence**
- the **language generated** by $G$ is

\[
L(G) = \text{def} \quad \{ w \mid S \Rightarrow^* w \}
\]

- **CFL** is the class of all languages generated by CFGs
Basic Definitions

Some additional notions for CFGs:

- $A \in N$ is **nullable** if there exists some derivation of the form $A \Rightarrow^* \varepsilon$
- $G$ is **reduced** if, for every $A \in N$, there exists some derivation of the form $S \Rightarrow^* \gamma A \delta \Rightarrow^* w$
- $G$ is **cyclic** if $G$ is reduced and, for some $A \in N$, $A \Rightarrow^+ A$
Parse Trees

Derivations can be described by means of node-labeled ordered trees called parse trees.

Parse trees do not represent the order of rule application, but represent the nonterminal that is rewritten by each rule.

The map between derivations and parse trees is one to many; the map between leftmost derivations and parse trees is one to one.
Parse Trees

Example:

- \( G = (\{S\}, \{a, b\}, P, S) \)
- \( P = \{ S \rightarrow aSb, S \rightarrow ab \} \)

\[
S \Rightarrow a \ S \ b \\
\Rightarrow a \ a \ S \ b \ b \\
\Rightarrow a \ a \ a \ a \ b \ b \ b
\]
Example:

- \( G = (\{S\}, \{a\}, P, S) \)
- \( P = \{S \rightarrow SS, S \rightarrow a\} \)

\[
\begin{align*}
S & \Rightarrow SS \\
   & \Rightarrow SSS \\
   & \Rightarrow aSS \\
   & \RightarrowaaS \\
   & \Rightarrow aaS \\
   & \Rightarrow aaa
\end{align*}
\]
We can **linearize** a parse tree by projecting its internal nodes into its **yield**:

\[
S \\
\downarrow \\
S \quad S \\
\downarrow \quad \downarrow \\
a \quad S \quad S \\
\downarrow \quad \downarrow \\
a \quad a
\]

\[
[s [s a ] [s [s a ] [s a ] ] ]
\]
Parse Trees

Some **basic notation** about parse trees:

- $T(G)$ is the set of all parse trees for $G$
- $T(G, w)$ is set of all parse trees for $G$ with **yield** $w$
Ambiguity refers to the possibility of generating the same string in several ways

- **$G$ is unambiguous** if for every $w \in L(G)$ we have $|T(G, w)| = 1$
- **$G$ is ambiguous** if it is not unambiguous
- **$L \in CFL$ is ambiguous** if there is no CFG $G$ such that $L(G) = L$ and $G$ is unambiguous
**Census** function of a CFG $G$:

$$C_G(n) \overset{\text{def}}{=} \max_{|w|=n} |T(G, w)|$$

$G$ is **massively ambiguous** if there is no polynomial $p(n)$ such that $C_G(n) = O(p(n))$

$G$ is **infinitely ambiguous** if, for some $w$, $T(G, w)$ is an **infinite** set
Ambiguity

Example (cont’d):

- $G = (\{S\}, \{a\}, P, S)$
- $P = \{S \rightarrow SS, S \rightarrow a\}$
- $G$ is massively ambiguous, generating all possible binary bracketings for a string
- $C_G(n)$ is the sequence of Catalan numbers
Ambiguity

Example:

- $G = (N, \Sigma, P, S)$
- $N = \{S, VP, NP, PP, V, N, P, Det\}$
- $\Sigma = \{eat, I, chocolate, fork, strawberries, with, a\}$
Example (cont’d) :

- $P$ defined by the following productions, where operator ‘$|$’
denotes alternative right-hand sides :

\[
\begin{align*}
S & \rightarrow \text{NP } \text{VP} \\
\text{NP} & \rightarrow \text{NP } \text{PP} \mid \text{Det } \text{N} \mid \text{N} \\
\text{VP} & \rightarrow \text{VP } \text{PP} \mid \text{V } \text{NP} \\
\text{PP} & \rightarrow \text{P } \text{NP} \\
\text{N} & \rightarrow \text{chocolate } \mid \text{I } \mid \text{fork } \mid \text{strawberries} \\
\text{V} & \rightarrow \text{eat,} \\
\text{Det} & \rightarrow \text{a} \\
\text{P} & \rightarrow \text{with}
\end{align*}
\]
Amiguity

```
S
  /   \
 NP  VP
  /   /|
 N   V  NP
   /   /
  I   eat N
       /
       strawberries
```
Ambiguity

S

NP       VP

N       V       NP

I     eat       NP

strawberries with N

chocolate
Introduction
Background
Basic Definitions
String and Languages
Context-Free Grammars
Push-Down Automata

Ambiguity

```
S
  /\  
 NP  VP
   /\  /
  N  VP PP
   /\  |
  I  V NP P  NP
   /\  |
  eat N with Det N
      /\  /
     strawberries a fork
```
Ambiguity

\[
S \quad ?
\]

\[
\begin{array}{c}
NP \\
N \\
I \\
N \\
strawberries
\end{array}
\quad \begin{array}{c}
VP \\
V \\
eat
\end{array}
\quad \begin{array}{c}
NP \\
P \\
with
\end{array}
\quad \begin{array}{c}
NP \\
Det \\
a
\end{array}
\quad \begin{array}{c}
NP \\
N \\
fork
\end{array}
\]
Ambiguity

A sentence can have more than one parse tree, indicating ambiguity. For example:

```
S -> NP VP
  /   \
NP   VP
  /     \
N     NP
   /     \
I     VP
   /     \
V     NP
    /     \
eat   P
     /     \nNP     NP
  /       \
strawberries chocolate
```
The \textbf{size} of a CFG is

\[ |G| \overset{\text{def}}{=} \sum_{(A \rightarrow \alpha) \in P} |A \cdot \alpha| \]

The size of a CFG is a \textbf{reasonable} measure of the \textbf{computational space} we need to represent the grammar.
Push-Down Automata

A push-down automaton, or PDA for short, is a tuple

$$M = \text{def} \ (Q, \Sigma, q_{\text{in}}, q_{\text{fin}}, \Delta)$$

where:

- $Q$ is a finite set of stack symbols
- $\Sigma$ is a finite set of input symbols with $Q \cap \Sigma = \emptyset$
- $q_{\text{in}} \in Q$ is the initial stack symbol
- $q_{\text{fin}} \in Q$ is the final stack symbol
- $\Delta$ is a finite set of transitions, each having the form
  $$\sigma_1 \rightarrow X \sigma_2$$
  with $\sigma_1, \sigma_2 \in Q^*$ and $X \in \Sigma \cup \{\varepsilon\}$

Note that there is no internal state in $M$; internal states can always be encoded within $Q$.
Assume an input string $w = a_1a_2\cdots a_n$, $a_i \in \Sigma$.

A configuration of $M$ on $w$ is a pair $(\sigma, i)$, where :

- $\sigma \in Q^*$ is a stack, with top-most symbol in the right-most position in the string.
- $i$ is an index for $w$, indicating the number of symbols read so far.
A **move** of $M$ is the **application** of a transition to a configuration. This is represented through a binary relation on configurations

$$(\sigma, j) \triangleright (\sigma', i)$$

holding if and only if

- $$(\sigma_1 \xrightarrow{X} \sigma_2) \in \Delta$$
- $\sigma = \sigma'' \sigma_1$, $\sigma' = \sigma'' \sigma_2$, $\sigma'' \in Q^*$
- $X = a_{j+1}$ and $i = j + 1$, or else $X = \varepsilon$ and $i = j$
Computation

A **computation** of $M$ is a **finite** sequence of moves. We use relations $\triangleright^*$ and $\triangleright^+$ to represent computations.

More precisely, if

$$(\sigma_0, j_0) \triangleright (\sigma_1, j_1), (\sigma_1, j_1) \triangleright (\sigma_2, j_2), \ldots, (\sigma_{q-1}, j_{q-1}) \triangleright (\sigma_q, j_q),$$

we write

$$(\sigma_0, j_0) \triangleright^* (\sigma_q, j_q).$$
The language **recognized** by $M$ is

$$L(M) \overset{\text{def}}{=} \{ w \mid (q_{in},0) \xrightarrow{\ast} (q_{fin},|w|) \}$$

Computations having the form $(q_{in},0) \xrightarrow{\ast} (q_{fin},|w|)$, $w$ some string, are called **accepting computations** of $M$

The **class of languages** recognized by PDAs is **exactly** the class of languages generated by CFGs
Computation

Example:

- \( \Sigma = \{ a, b, c, d \} \)
- \( Q = \{ q_0, q_1, \ldots, q_9 \} \)
- \( q_{in} = q_0, \quad q_{fin} = q_9 \)
- \( \Delta \) contains the following transitions

\[
\begin{array}{c|c|c|c|c}
q_0 & \mapsto a & q_0q_1 & q_4q_5 & \mapsto \varepsilon & q_6 \\
q_0q_1 & \mapsto b & q_0q_2 & q_2q_6 & \mapsto \varepsilon & q_7 \\
q_0q_1 & \mapsto b & q_0q_3 & q_0q_7 & \mapsto \varepsilon & q_9 \\
q_2 & \mapsto c & q_2q_4 & q_3q_6 & \mapsto \varepsilon & q_8 \\
q_3 & \mapsto c & q_3q_4 & q_0q_8 & \mapsto \varepsilon & q_9 \\
q_4 & \mapsto d & q_4q_5 & & & \\
\end{array}
\]
Example (cont’d) : Input string \( w = abcd \), first computation
Example (cont’d) : Input string $w = abcd$, second computation
Nondeterminism

$M$ is **deterministic** if for each input string and each configuration there is **at most** one transition that can be applied.

$M$ is **strictly nondeterministic** if it is **not** deterministic.
If $M$ is non-deterministic, some strings can be recognized by several computations.

The set of accepting computations of $M$ for a string of length $n$ can grow with an exponential function of $n$.

For some string, the set of accepting computations of $M$ can even be infinite.
The size of a PDA is

$$|M| =_{\text{def}} \sum_{(\sigma_1 \xrightarrow{X} \sigma_2) \in \Delta} |\sigma_1 \cdot X \cdot \sigma_2|$$

The size of a PDA is a reasonable measure of the computational space we need to represent the automaton.
J. Earley.
1970.
An efficient context-free parsing algorithm.

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

M. Kay.
1986.
Algorithm schemata and data structures in syntactic processing.

B. Lang.


H. Leiss.
M. Tomita.  
1986.  
*Efficient Parsing for Natural Language.*  

D.H. Younger.  
1967.  
Recognition and parsing of context-free languages in time $n^3$.  