Parsing
Cocke Younger Kasami (CYK)

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2017/18
Introduction

The CYK parser is

- a **bottom-up** parser: we start with the terminals in the input string and subsequently compute recognized parse trees by going from already recognized rhs of productions to the non-terminal on the lefthand side.

- a **non-directional** parser: the checking for recognized components of a rhs in order to complete a lhs is not ordered; in particular, we cannot complete only parts of a rhs, everything in the rhs must be recognized in order to complete the lefthand category.

Independently proposed by Cocke, Kasami and Younger in the 60s.

Cocke and Schwartz (1970); Grune and Jacobs (2008); Hopcroft and Ullman (1979, 1994); Kasami (1965); Younger (1967)
We store the results in a \((n + 1) \times (n + 1)\) chart (table) \(C\) such that \(A \in C_{i,l}\) iff \(A \Rightarrow^* w_i \ldots w_{i+l-1}\).

In other words,

- \(i\) is the index of the first terminal in the relevant substring of \(w\); \(i\) ranges from 1 to \(n + 1\) (the latter for an empty word following \(w_n\))
- \(l\) is the length of the substring; \(l\) ranges from 0 to \(n\).

All fields in the chart are initialized with \(\emptyset\).
The general recognizer (2)

General CYK recognizer (for arbitrary CFGs):

\[
C_{i,1} := \{w_i\} \quad \text{for all } 1 \leq i \leq n \quad \text{scan}
\]

\[
C_{i,0} := \{A \mid A \rightarrow \epsilon \in P\} \quad \text{for all } i \in [1..n+1] \quad \epsilon\text{-productions}
\]

for all \( i \in [0..n] : \)

\[
\text{for all } i_1 \in [1..n+1]: \]

repeat until chart does not change any more:

\[
\text{for every } A \rightarrow A_1 \ldots A_k : \]

if there are \( l_1, \ldots, l_k \in [0..n] \) such that

\[
l_1 + \ldots + l_k = l \quad \text{and}
\]

\[
A_j \in C_{i,j,l} \quad \text{with } i_j = i_1 + l_1 \ldots + l_{j-1},
\]

then \( C_{i_1,l} := C_{i_1,l} \cup \{A\} \quad \text{complete} \)
The general recognizer (3)

Example

\[ S \rightarrow ABC, \ A \rightarrow aA \mid \epsilon, \ B \rightarrow bB \mid \epsilon, \ C \rightarrow c. \]
\[ w = aabbbc. \]

<table>
<thead>
<tr>
<th>l</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>A,B</td>
<td>A,B</td>
<td>A,B</td>
<td>A,B</td>
<td>A,B</td>
<td>A,B</td>
<td>A,B</td>
<td>i</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The general recognizer (4)

Parsing schema for CYK:
Items have three elements:

- $X \in N \cup T$: the nonterminal/terminal that spans a substring $w_i, \ldots, w_j$ of $w$;
- the index $i$ of the first terminal in the subsequence;
- the length $l = j - i$ of the subsequence.

Item form: $[X, i, l]$ with $X \in N \cup T$, $i \in [1..n + 1]$, $l \in [0..n]$. 
The general recognizer (5)

Goal item: \([S, 1, n]\).

Deduction rules:

Scan: \(\text{[a, i, 1]} \quad w_i = a\)

\(\epsilon\)-productions: \(\text{[A, i, 0]} \quad A \rightarrow \epsilon \in P, \ i \in [1..n + 1]\)

Complete: \(\text{[A_1, i_1, l_1]}, \ldots, \text{[A_k, i_k, l_k]} \quad A \rightarrow A_1 \ldots A_k \in P, \ l = l_1 + \cdots + l_k, \ i_j = i_1 + l_1 \cdots + l_{j-1}\)
Tabulation avoids problems with loops: nothing needs to be computed more than once.

In each complete step, we have to check for all $l_1, \ldots, l_k$. This is costly.

Note, however, that we create a new chart entry (new item) only for combinations of already recognized parse trees. (No blind prediction as in Unger’s parser.)

With unary rules and $\epsilon$-productions, an entry in field $C_{i,l}$ can be reused to compute a new entry in $C_{i,l}$. This is why the repeat until chart does not change any more loop is necessary.
A CFG is in Chomsky Normal Form iff all productions are either of the form $A \rightarrow a$ or $A \rightarrow B C$. If the grammar has this form,

- we need to check only $l_1, l_2$ in a complete step, and
- we can be sure that to compute an entry in field $C_{i,l}$, we do not use another entry from field $C_{i,l}$. Consequently, we do not need the `repeat until chart does not change any more` loop.
The chart $C$ is now an $n \times n$-chart.

$$C_{i,1} := \{A | A \to w_i \in P\}$$

**scan**

for all $l \in [1..n]$:

for all $i \in [1..n]$:

for every $A \to B \in C$:

if there is a $l_1 \in [1..l-1]$ such that

$B \in C_{i,l_1}$ and $C \in C_{i+l_1,l-l_1}$,

then $C_{i,l} := C_{i,l} \cup \{A\}$

**complete**
The CNF recognizer (3)

Parsing schema for CNF CYK:

Goal item: \([S, 1, n]\)

Deduction rules:

Scan: \([A, i, 1] \rightarrow w_i \in P\)

Complete: \([B, i, l_1], [C, i + l_1, l_2] \rightarrow A \rightarrow BC \in P\)
The CNF recognizer (4)

**Example**

\[
S \rightarrow C_a C_b \mid C_a S_B, S_B \rightarrow SC_b, C_a \rightarrow a, C_b \rightarrow b. \text{ (From } S \rightarrow aSb \mid ab \text{ with transformation into CNF.)}
\]

\[w = aaabbb.\]

<table>
<thead>
<tr>
<th>[l]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>S_B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>S_B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>C_a</td>
<td>C_a</td>
<td>C_a</td>
<td>C_b</td>
<td>C_b</td>
<td>C_b</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>i</td>
</tr>
</tbody>
</table>


The CNF recognizer (5)

Time complexity: How many different instances of scan and complete are possible?

Scan: \[
[A, i, 1] \quad A \rightarrow w_i \in P \quad c_1 n
\]

Complete: \[
\frac{[B, i, l_1], [C, i + l_1, l_2]}{[A, i, l_1 + l_2]} \quad A \rightarrow BC \in P \quad c_2 n^3
\]

Consequently, the time complexity of CYK for CNF is \(O(n^3)\).

The space complexity of CYK is \(O(n^2)\).
Control structures: there are two possible orders in which the chart can be filled:

1. **off-line order**: fill first row 1, then row 2 etc.:
   - for all \( l \in [1..n] \): (length)
     - for all \( i \in [1..n-l+1] \): (start position)
       - compute \( C_{i,l} \)

2. **on-line order**: fill one diagonal after the other, starting with 1, 1 and proceeding from \( k, 1 \) to \( 1, k \):
   - for all \( k \in [1..n] \): (end position)
     - for all \( l \in [1..k] \): (length)
       - compute \( C_{k-l+1,l} \)
Soundness of the algorithm: If $[A, i, l]$, then $A \Rightarrow^* w_i \ldots w_{i+l-1}$.
Proof via induction over deduction rules.

Completeness of the algorithm: If $A \Rightarrow^* w_i \ldots w_{i+l-1}$, then $[A, i, l]$.
Proof via induction over $l$. 
We know that for every CFG $G$ with $\epsilon \notin L(G)$ we can

- eliminate $\epsilon$-productions,
- eliminate unary productions,
- eliminate useless symbols,
- transform into CNF,

and the resulting CFG $G'$ is such that $L(G) = L(G')$. Therefore, for every CFG, we can use the CNF recognizer after transformation.

How can we obtain a parser?
We need to do two things:

- turn the CNF recognizer into a parser, and
- if the original grammar was not in CNF, retrieve the original syntax from the CNF syntax.
To turn the CNF recognizer into a parser, we record not only non-terminal categories but whole productions with the positions and lengths of the rhs symbols in the chart (i.e., with backpointers):

\[
C_{i,1} := \{A \rightarrow w_i | A \rightarrow w_i \in P\}
\]

for all \( l \in [1..n] \):

for all \( i \in [1..n] \):

for every \( A \rightarrow B C \):

if there is a \( l_1 \in [1..l-1] \) such that

\( B \in C_{i,l_1} \) and \( C \in C_{i+l_1,l-l_1} \),

then \( C_{i,l} := C_{i,l} \cup \{A \rightarrow [B, i, l_1][C, i + l_1, l - l_1]\} \)

We can then obtain a parse tree by traversing the productions from left to right, starting with every \( S \)-production in \( C_{1,n} \).
Example

\[ S \rightarrow C_a C_b \mid C_a S_B, \quad S_B \rightarrow SC_b, \quad C_a \rightarrow a, \quad C_b \rightarrow b, \quad w = aaabbb. \]  
(We write \( A_{i,l} \) for \( [A, i, l] \).)

<table>
<thead>
<tr>
<th>( S \rightarrow )</th>
<th>( C_{a1,1} S_{B2,5} )</th>
<th>( S_B \rightarrow )</th>
<th>( S_{2,4} C_{b6,1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow )</td>
<td>( C_{a2,1} S_{B3,3} )</td>
<td>( S_B \rightarrow )</td>
<td>( S_{3,2} C_{b5,1} )</td>
</tr>
<tr>
<td>( S \rightarrow )</td>
<td>( C_{a3,1} C_{b4,1} )</td>
<td>( C_a \rightarrow a )</td>
<td>( C_a \rightarrow a )</td>
</tr>
</tbody>
</table>

\[ C_a \rightarrow a \]  
\[ C_a \rightarrow a \]  
\[ C_a \rightarrow a \]  
\[ C_b \rightarrow b \]  
\[ C_b \rightarrow b \]  
\[ C_b \rightarrow b \]
From the CNF parse tree to the original parse tree:
First, we undo the CNF transformation:

- replace every $C_a \rightarrow a$ in the chart with $a$ and replace every occurrence of $C_a$ in a production with $a$.

- For all $l, i \in [1..n]$: If $A \rightarrow \alpha D_{iD,lD} \in C_{i,l}$ such that $D$ is a new symbol introduced in the CNF transformation and $D \rightarrow \beta \in C_{iD,lD}$, then replace $A \rightarrow \alpha D_{iD,lD}$ with $A \rightarrow \alpha\beta$ in $C_{i,l}$.

- Finally remove all $D \rightarrow \gamma$ with $D$ being a new symbol introduced in the CNF transformation from the chart.
Example

$S \rightarrow C_a C_b \mid C_a S_B$, $S_B \rightarrow S C_b$, $C_a \rightarrow a$, $C_b \rightarrow b$, $w = aaabbb$. New symbols: $C_a$, $C_b$, $S_B$. Elimination of $C_a$, $C_b$:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$S \rightarrow a S_{B2,5}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$S_B \rightarrow S_{2,4} b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow a S_{B3,3}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$S_B \rightarrow S_{3,2} b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow a b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Example

\[ S \rightarrow C_a C_b \mid C_a S_B, \quad S_B \rightarrow S C_b, \quad C_a \rightarrow a, \quad C_b \rightarrow b, \quad w = aaabbb. \]

Replacing of \( S_B \) in rhs:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 a a a b b b

1 2 3 4 5 6
### Example

Given the grammar:

\[
S \rightarrow C_a C_b \mid C_a S_B, \quad S_B \rightarrow S C_b, \quad C_a \rightarrow a, \quad C_b \rightarrow b, \quad w = aaabbb
\]

**Elimination of** $S_B$:

<table>
<thead>
<tr>
<th>6</th>
<th>$S \rightarrow aS_{2,4}b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow aS_{3,2}b$</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow ab$</td>
</tr>
<tr>
<td>1</td>
<td>$a$</td>
</tr>
<tr>
<td>1</td>
<td>$a$</td>
</tr>
</tbody>
</table>

---

```plaintext
1 2 3 4 5 6
```
Undo the elimination of unary productions:

- For every $A \rightarrow \beta$ in $C_{i,l}$ that has been added in removing of the unary productions to replace $B \rightarrow \beta'$ ($\beta'$ is $\beta$ without chart indices): replace $A$ with $B$ in this entry in $C_{i,l}$.

- For every unary production $A \rightarrow B$ in the original grammar and for every $B \rightarrow \beta \in C_{i,l}$: add $A \rightarrow B_{i,l}$ to $C_{i,l}$. Repeat this until chart does not change any more.
Undo the elimination of $\epsilon$-productions:

- Add a row with $l = 0$ and a column with $i = n + 1$ to the chart.
- Fill row 0 as in the general case using the original CFG grammar (tabulating productions).
- For every $A \rightarrow \beta$ in $C_{i,l}$ that has been added in removing the $\epsilon$-productions: add the deleted nonterminals to $\beta$ with the position of the preceding non-terminal as starting position (or $i$ if it is the first in the rhs) and with length 0.
Terminal Filter: Observation: Information on the obligatory presence of terminals might get lost in the CNF transformation:

\[ S \rightarrow aSb \] (requires an \( a \), an \( S \) and a \( b \)) \( \sim \) \[ S \rightarrow C_aS_B \] (requires an \( a \) and an \( S \) and a \( b \)) and \[ S_B \rightarrow SC_b \] (requires an \( S \) and a \( b \))

Consider an input \( babb \):

- In a CYK parser with the original grammar, we would derive \( [S, 2, 2] \) and \( [b, 4, 1] \) but we could not apply \( S \rightarrow aSb \).
- In the CNF grammar, we would have \( [S, 2, 2] \) and \( [C_b, 4, 1] \) and then we could apply \( S_B \rightarrow SC_b \) and obtain \( [S_B, 2, 3] \) even though the only way to continue with \( S_B \) in a rhs is with \( S \rightarrow C_aS_B \) which is not possible since the first terminal is not an \( a \).
Solution: add an additional check:

- Every new non-terminal $D$ introduced for binarization stands (deterministically) for some substring $\beta$ of a rhs $\alpha\beta$. Ex: $S_B$ in our sample grammar stands for $Sb$.

- Every terminal in this rhs to the left of $\beta$, i.e., evey terminal in $\alpha$ must necessarily be present to the left for a deduction of a $D$ that leads to a goal item. Ex: $S_B$ can only lead to a goal item if to its left we have an $a$.

- **Terminal Filter**: During CNF transformation, for every non-terminal $D$ introduced for binarization, record the sets of terminals in the rhs to the left of the part covered by $D$. During parsing, check for the presence of these terminals to the left of the deduced $D$ item.
CNF leads to a binarization: In each completion, only two items are combined. Such a binarization can be obtained by using dotted productions:

- We process the rhs of a production from left to right.
- In each complete step, a production $A \rightarrow \alpha \bullet X \beta$ is combined with an $X$ whose span starts at the end of the $\alpha$-span. The result is a production $A \rightarrow \alpha X \bullet \beta$.
- We start with the completed terminals and their spans.

Note that this version of CYK is directional.
CYK with dotted productions (2)

Parsing schema for the general version (allowing also for ε-productions):

Goal items: \([S \rightarrow \alpha\bullet, 0, n]\) for all S-productions \(S \rightarrow \alpha\).

Deduction rules:

Predict (axioms): \[
\frac{[A \rightarrow \bullet\alpha, i, i]}{A \rightarrow \alpha \in P, i \in [0..n]}
\]

Scan: \[
\frac{[A \rightarrow \alpha \bullet a\beta, i, j]}{[A \rightarrow \alpha a \bullet \beta, i, j + 1]}
\]

\(w_{j+1} = a\)

Complete: \[
\frac{[A \rightarrow \alpha \bullet B\beta, i, j][B \rightarrow \gamma\bullet, j, k]}{[A \rightarrow \alpha B \bullet \beta, i, k]}
\]
CYK with dotted productions (3)

Parsing schema including passive items (just a non-terminal or terminal, no dotted production) and assuming an \( \varepsilon \)-free CFG:

Goal item: \([S, 0, n]\)

Deduction rules:

Scan (axioms): \( [a, i, i + 1] \text{ } w_{i+1} = a \)

Left-corner predict: \( \begin{array}{c}
\text{Left-corner predict: } \\
[A \rightarrow X \bullet \alpha, i, j] \quad A \rightarrow X\alpha \in P, X \in N \cup T
\end{array} \)

Complete: \( \begin{array}{c}
\text{Complete: } \\
[A \rightarrow \alpha \bullet X\beta, i, j][X, j, k] \\
[A \rightarrow \alpha X \bullet \beta, i, k]
\end{array} \)

Publish: \( \begin{array}{c}
\text{Publish: } \\
[A \rightarrow \alpha \bullet, i, j] \\
[A, i, j]
\end{array} \)

(This is actually a deduction-based version of left-corner parsing.)
Example (without $\epsilon$-productions, left-corner parsing): $S \rightarrow ab \mid aSb$

$w = aabb$

<table>
<thead>
<tr>
<th></th>
<th>$S \rightarrow aSb \bullet$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$S$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow aS \bullet b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow ab \bullet$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow a \bullet b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow a \bullet Sb$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 2 3 4
What about time complexity?

The most complex operation, \textit{complete}, involves only three indices \(i, j, k\) ranging from 0 to \(n\):

\[
\text{Complete: } \frac{[A \to \alpha \bullet X\beta, i, j][X, j, k]}{[A \to \alpha X \bullet \beta, i, k]}
\]

Consequently, the time complexity is \(O(n^3)\), as in the CNF case.

But: the data structure required for representing a parse item with a dotted production is slightly more complex than what is needed for simple passive items.
Conclusion

- CYK is a non-directional bottom-up parser.
- If used with CNF, it is very efficient. Time complexity is $O(n^3)$.
- The transformation into CNF can be undone after parsing, i.e., we still have a parser for arbitrary CFGs (as long as $\epsilon$ is not in the language).
- Instead of explicitly binarizing, we can use dotted productions and move through the righthand sides of productions step by step from left to right, which also leads to $O(n^3)$. 
Bibliography


