Parsing
Unger’s Parser

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2016/17
Unger’s parser (Grune and Jacobs, 2008) is a CFG parser that is

- a **top-down** parser: we start with S and subsequently replace lefthand sides of productions with righthand sides

- a **non-directional** parser: the expanding of non-terminals (with appropriate righthand sides) is not ordered; therefore we need to guess the yields of all non-terminals in a right-hand side at once
**Introduction (2)**

\[ G = \langle N, T, P, S \rangle, \quad N = \{ S, NP, VP, PP, V, \ldots \}, \quad T = \{ Mary, man, telescope, \ldots \}, \text{ productions: } S \rightarrow NP \ VP, \ VP \rightarrow VP PP, \ VP \rightarrow V NP, \ NP \rightarrow Mary, \ldots \]  

**Input:** Mary sees the man with the telescope

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( S )</td>
<td>Mary sees the man with the telescope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>( NP )</td>
<td>Mary</td>
<td>( S \rightarrow NP VP ) (1.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>( VP )</td>
<td>sees the man with the telescope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>( NP )</td>
<td>Mary sees</td>
<td>( S \rightarrow NP VP ) (1.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>( VP )</td>
<td>the man with the telescope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>Mary</td>
<td>Mary</td>
<td>( NP \rightarrow Mary ) (2.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>( VP )</td>
<td>sees</td>
<td>( VP \rightarrow VP PP ) (3.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>( PP )</td>
<td>the man with the telescope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>( VP )</td>
<td>sees the</td>
<td>( VP \rightarrow VP PP ) (3.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>( PP )</td>
<td>man with the telescope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parsing strategy:

- The parser takes an $X \in N \cup T$ and a substring $w$ of the input.
- Initially, this is $S$ and the entire input.
- If $X$ and the remaining substring are equal, we can stop (success for $X$ and $w$).
- Otherwise, $X$ must be a non-terminal that can be further expanded. We then choose an $X$-production and partition $w$ into further substrings that are paired with the righthand side elements of the production.
- The parser continues recursively.
Assume CFG without $\epsilon$-productions and without loops $A \Rightarrow A$

```
function unger(w,X):
    out := false;
    if $w = X$, then out := true
    else for all $X \rightarrow X_1 \ldots X_k$:
        for all $x_1, \ldots, x_k \in T^+$ with $w = x_1 \ldots x_k$:
            if $\bigwedge_{i=1}^{k} unger(x_i,X_i)$
                then out := true;
    return out
```

The following holds:

$$unger(w,X) \text{ iff } X \Rightarrow^* w \text{ (for } X \in N \cup T, w \in T^*)$$
Extension to deal with ε-productions and loops:

- Add a list of preceding calls
- pass this list when calling the parser again
- if the new call is already on the list, stop and return false

Initial call: \texttt{unger}(w, S, \emptyset)
function unger\((w,X,L)\):
    out := false;
    if \(\langle X,w \rangle \in L\), return out;
    else if \(w = X\) or \(w = \epsilon\) and \(X \rightarrow \epsilon \in P\)
        then out := true
    else for all \(X \rightarrow X_1 \ldots X_k \in P\):
        for all \(x_1, \ldots, x_k \in T^*\) with \(w = x_1 \ldots x_k\):
            if \(\bigwedge_{i=1}^k\) unger\((x_i,X_i, L \cup \{\langle X,w \rangle\})\)
                then out := true;
    return out
So far, we have a recognizer, not a parser.

To turn this into a parser, every call `unger(..)` must return a (set of) parse trees.

This can be obtained from

1. the successful productions $X \rightarrow X_1 \ldots X_k$, and
2. the parse trees returned by the calls `unger(x_i, X_i)`.

Note, however, that there might be a large amount of parse trees since in each call, there might be more than one successful production.

We will come back to the compact presentation of several analyses in a parse forest.
An example (1)

- Assume a CFG without $\varepsilon$-productions
- Production $S \rightarrow NP \ VP$
- Input sentence $w$ with $|w| = 34$:

Mr. Sarkozy’s pension reform, which only affects about 500,000 public sector employees, is the opening salvo in a series of measures aimed more broadly at rolling back France’s system of labor protections.

(New York Times)
An example (2)

Partitions according to Unger’s parser:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( NP )</td>
<td>( VP )</td>
</tr>
<tr>
<td>1. Mr.</td>
<td>Sarkozy’s … protections</td>
</tr>
<tr>
<td>2. Mr. Sarkozy</td>
<td>’s … protections</td>
</tr>
<tr>
<td>3. Mr. Sarkozy’s</td>
<td>pension … protections</td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
</tr>
<tr>
<td>33. Mr. …labor</td>
<td>protections</td>
</tr>
</tbody>
</table>

\(|w| = 34\), consequently we have 33 different partitions.
Consider the following partition for $S \rightarrow NP \ VP$:

$S\ NP\ Mr.\ Sarkozy’s\ pension\ reform,\ which\ …\ employees,$

$VP\ is\ …\ protections$

- For $NP \rightarrow NP\ S$, there are 12 partitions of the $NP$ part
- The partition above is just one partition for one production!
- In the worst case, parsing is exponential in the length $n$ of the input string!
We say that an algorithm is of

- **polynomial time complexity** if there is a constant $c$ and a $k$ such that the parsing of a string of length $n$ takes an amount of time $\leq cn^k$.

  Notation: $\mathcal{O}(n^k)$

- **exponential time complexity** if there is a constant $c$ and a $k$ such that the parsing of a string of length $n$ takes an amount of time $\leq ck^n$.

  Notation: $\mathcal{O}(k^n)$
As an additional filter, we can constrain the set of partitions that we investigate:

- Check on occurrences of terminals in rhs.
- Check on minimal length of terminal string derived by a non-terminal.
- Check on obligatory terminals (pre-terminals) in strings derived by non-terminals, e.g., each NP contains an N, each VP contains a V, …
- Check on the first terminals derivable from a non-terminal.
Furthermore, we can use tabulation (dynamic programming) in order to avoid computing several times the same thing:

1. Whenever \texttt{unger} (\texttt{X}, \texttt{w}, \texttt{L}) yields a result \texttt{res}, we store \texttt{\langle X, w, res \rangle} in our table of partial parsing results.

2. In every call \texttt{unger} (\texttt{X}, \texttt{w}, \texttt{L}), we first check whether we have already computed a result \texttt{\langle X, w, res \rangle} and if so, we stop immediately and return \texttt{res}. 
Optimizations (3)

Results $\langle X, w, res \rangle$ can be stored in a three-dimensional table (chart) $C$:

- Assume $k = |N + T|$ and non-terminals $N$ and terminals $T$ to have a unique index $\leq k$. Furthermore, assume $|w| = n$ with $w = w_1 \cdots w_n$, then you can use a $k \times n \times n$ table, the chart!
  
  1. Whenever $\text{unger}(X, w_i \cdots w_j, L)$ yields a result $res$ and $m$ index of $X$, then $C(m, i, j) = res$
  
  2. In every call $\text{unger}(X, w_i \cdots w_j, L)$, we first check whether we have already a value in $C(m, i, j)$ and if so, we stop and return $C(m, i, j)$

- Advantage: access of $C(m, i, j)$ in constant time.

- Disadvantage: storing the Chart needs more memory.

- Assumption: grammar is $\varepsilon$-free – otherwise we need a $k \times (n + 1) \times (n + 1)$ chart.
Optimizations (4)

Example

- $G = \langle N, T, P, S \rangle$, \( N = \{ S, B \} \), \( T = \{ a, b, c \} \) and productions $S \rightarrow aSB \mid c \ B \rightarrow bb$

- Input word $w = acbb$.

- We assume that, when guessing the span of a rhs element, we take into account that …
  1. each terminal spans only a corresponding single terminal
  2. the span of an $S$ has to start with an $a$ or a $c$
  3. the span of a $B$ has to start with a $b$
  4. the span of each $X \in N \cup T$ contains at least one symbol (no $\varepsilon$-productions)
Example continued

Chart obtained for $w = acbb$

<table>
<thead>
<tr>
<th>j</th>
<th>$\langle S, t \rangle$</th>
<th>$\langle B, t \rangle$</th>
<th>$\langle b, t \rangle$</th>
<th>$\langle B, f \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$\langle S, f \rangle$</td>
<td>$\langle b, t \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$\langle S, t \rangle$</td>
<td></td>
<td>$\langle c, t \rangle$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$\langle S, t \rangle$</td>
<td>$\langle c, t \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\langle a, t \rangle$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Productions: $S \rightarrow aSB \mid c$ $B \rightarrow bb$)

$S \Rightarrow^{*} acbb? \rightarrow t$

$a \Rightarrow^{*} a? \rightarrow t$

$S \Rightarrow^{*} c? \rightarrow t$

$c \Rightarrow^{*} c? \rightarrow t$

$B \Rightarrow^{*} bb? \rightarrow t$

$b \Rightarrow^{*} b? \rightarrow t$

$b \Rightarrow^{*} b? \rightarrow t$

$S \Rightarrow^{*} cb \rightarrow f$

$B \Rightarrow^{*} b \rightarrow f$
In addition, we can tabulate entire productions with the spans of their different symbols. This gives us a compact presentation of the parse forest!

- In every call `unger(X, w_i \cdots w_j)`, we first check whether we have already a value in `C(m, i, j)` and if so, we stop and return `C(m, i, j)`.

- Otherwise, we compute all possible first steps of derivations `X \Rightarrow^* w`: for every production `X \rightarrow X_1 \ldots X_k` and all `w_1, \ldots, w_k` such that the recursive Unger calls yield `true`, we add `\langle X, w \rangle \rightarrow \langle X_1, w_1 \rangle \ldots \langle X_k, w_k \rangle` with the indices of the spans to the list of productions.

- If at least one such production has been found, we return `true`, otherwise `false`.

Example on handout.
Conclusion

Unger’s parser is

- a non-directional top-down parser.
- highly non-deterministic because during parsing, the yields of all non-terminals in righthand sides must be guessed.
- in general of exponential (time) complexity.
- of polynomial time complexity if tabulation is applied.