Logical and Computational Structures for Linguistic Modeling
Part 2 – Parsing CFGs and beyond

Éric de la Clergerie
<Eric.De_La_Clergerie@inria.fr>

23 Septembre 2014
Part I

Parsing CFGs
Parsing as tree gluing

s → np vp
np → pn
np → det n
np → np pp
vp → v np
vp → vp pp
pp → prep np

INRIA
É. de la Clergerie
TAL
23/09/2014
3 / 101
Essential to clearly distinguish

- Parsing strategy
- Control strategy

A parsing strategy describes the allowed steps to be tried during parsing

- **top-down** strategies (guided by goals, starting from the axiom)
- **bottom-up** strategies (guided by answers, starting from terminals)
- hybrid strategies (including Earley strategy)
- table-driven strategies (Left Corner, Head Corner, LR, ...)

---
A control strategy specifies how to handle non-determinism, especially scheduling:

**Scheduling**  In which order perform parsing steps?
- depth first
- breadth first,
- (left-to-right) string scanning synchronization
- parallel, concurrent, . . .

**Ambiguity**  How to handle ambiguities
- disambiguation (probabilities, heuristics, lookahead),
- backtracking,
- tabulation, . . .
for a chain of $k$ PPs, exponential number of syntactic trees wrt $k$
The principles of Dynamic Programming are

1. (recursively) break a problem into smaller ones
2. compute once the (best) solution(s) to the small problems
3. reuse the solution to (recursively) solve larger problems

~ tabulation of the solutions for reuse

Mostly found for optimization problems

- shortest path in a graph
- knapsack problem
- editing distance

But also a long tradition in parsing
A long story with many algorithms:

- CKY [Cocke-Kasami-Younger]
- Earley algorithm – Chart parsing [Kay]
- Generalized LR [Tomita]
- Stack automata / dynamic programming [Lang]
Outline

1. CKY
2. Chart Parsing
3. Generalized LR
4. Shared Forests
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

*If there exists a production* \( A_0 \leftarrow A_1 \ldots A_n \) *with, for all* \( i > 0 \), \( A_i \) *present in* \( (x_i, l_i) \) *and* \( x_{i+1} = x_i + l_i \), *then tabulate the non terminal* \( A_0 \) *in the entry* \( (x_1, \sum_i l_i) \) *(unless already tabulated).*
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

*If there exists a production* $A_0 \leftarrow A_1 \ldots A_n$ *with, for all* $i > 0$, $A_i$ *present in* $(x_i, l_i)$ *and* $x_{i+1} = x_i + l_i$, *then tabulate the non terminal* $A_0$ *in the entry* $(x_1, \Sigma_i l_i)$ *unless already tabulated.*

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

<table>
<thead>
<tr>
<th>Length</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production \( A_0 \leftarrow A_1 \ldots A_n \) with, for all \( i > 0 \), \( A_i \) present in \((x_i, l_i)\) and \( x_{i+1} = x_i + l_i \), then tabulate the non terminal \( A_0 \) in the entry \((x_1, \Sigma_i l_i)\) (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \sum_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$
present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in
the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right
but actually not mandatory!
Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

<table>
<thead>
<tr>
<th>Length</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>S_{17}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>VP_{15,16}</td>
<td>NP_{14}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S_{13}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>VP_{11}</td>
<td>NP_{9}</td>
<td>PP_{12}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>VP_{11}</td>
<td>NP_{9}</td>
<td>PP_{12}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PN_{1}NP_{8}</td>
<td>V_{2}</td>
<td>Det_{3}</td>
<td>N_{4}</td>
<td>Prep_{5}</td>
<td>Det_{6}</td>
<td>NP_{10}</td>
</tr>
<tr>
<td>1</td>
<td>John</td>
<td>observes</td>
<td>a</td>
<td>man</td>
<td>with</td>
<td>a</td>
<td>telescope</td>
</tr>
</tbody>
</table>

Constituents usually built by increasing length going left-to-right but actually not mandatory!
If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, $A_i$ present in $(x_i, l_i)$ and $x_{i+1} = x_i + l_i$, then tabulate the non terminal $A_0$ in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Constituents usually built by increasing length going left-to-right but actually not mandatory!
```
<table>
<thead>
<tr>
<th>Table Initialize</th>
</tr>
</thead>
<tbody>
<tr>
<td>for all positions x and lengths l</td>
</tr>
<tr>
<td>for all productions A₀ ← A₁ ... Aᵥ</td>
</tr>
<tr>
<td>for all lengths l₁,...,lv−1 with ( \sum_{k=1}^{\cdot \cdot \cdot v-1} l_k &lt; l )</td>
</tr>
<tr>
<td>( l_v = l - \sum_{k=1}^{\cdot \cdot \cdot v-1} l_k )</td>
</tr>
<tr>
<td>( x_j = x + l_1 + \cdot \cdot \cdot + l_{j-1} )</td>
</tr>
<tr>
<td>if ( A_j \in T[x_j, l_j] ) for all ( j &gt; 1 )</td>
</tr>
<tr>
<td>then add A₀ in T[x, l] (unless present)</td>
</tr>
</tbody>
</table>
```

Worst-case time complexity provided by nested iterations on \( x, l \) and \( l_j \) (\( 1 \leq j < v \)) bounded by the input string length \( n \).

\( \Rightarrow O(n^{v+1}) \) where \( v \) is the length of the longest production.

For a recognizer, worst-case space complexity given by the number of table cells and number of constituents per cell.

\( \Rightarrow O(n^2) \)
Complexity in $O(n^{v+1})$ reduced to $O(n^3)$ using **Chomsky normal form** (binarization).

Ternary rule $\text{VP} \rightarrow \text{V, NP, NP}$ gives a $O(n^4)$ complexity but may be replaced by following binary rules

\[
\begin{align*}
\text{VP} & \rightarrow \text{V, VP_ARGS.} \\
\text{VP_ARGS} & \rightarrow \text{NP, NP.}
\end{align*}
\]

But involves grammar transformation more elegant to manipulate **dotted rules**.

Worst-case $O(n^3)$ time and $O(n^2)$ space complexities (almost) optimal for CFGs but CKY not (always) efficient!
Useless constituents

In the model, the longer \([s \text{ the word is the less frequent}]\) it is

Trace hypotheses

John who looks \([s \text{ Marie leaves }\)]
Outline

1. CKY
2. Chart Parsing
3. Generalized LR
4. Shared Forests
Historically, motivated by the wish to

- use tabulation (for computation sharing)
- preserves optimal complexity $O(n^3)$ for CFGs
- introduce (top-down) prediction

→ development of generic techniques based on charts.
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as items \( \langle i, j, Cat \rangle \).

0 John 1 observes 2 a 3 man 4 with 5 a 6 telescope 7

Time complexity in \( O(n^{v+1}) \)
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as \texttt{items } \langle i, j, \text{Cat} \rangle.

\begin{align*}
0 & \quad \text{PN} & 1 & \quad \text{observes} & 2 & \quad \text{Det} & 3 & \quad \text{N} & 4 & \quad \text{Prep} & 5 & \quad \text{Det} & 6 & \quad \text{N} & 7 \\
& & & & & & & & & & & & & \\
& & & & & & & & & & & & & \\
0 & \quad \text{John} & 1 & \quad \text{a} & 2 & \quad \text{man} & 3 & \quad \text{with} & 4 & \quad \text{a} & 5 & \quad \text{telescope} & 6 & \quad 7
\end{align*}

Time complexity in $O(n^{v+1})$
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as \textbf{items} \( [i, j, \text{Cat}] \).

\begin{itemize}
  \item \( \text{NP} \)\hspace{1cm} \( \text{VN} \)
  \item 0 \hspace{1cm} John \hspace{1cm} 1 \hspace{1cm} observes \hspace{1cm} 2 \hspace{1cm} a \hspace{1cm} 3 \hspace{1cm} man \hspace{1cm} 4 \hspace{1cm} with \hspace{1cm} 5 \hspace{1cm} a \hspace{1cm} 6 \hspace{1cm} telescope \hspace{1cm} 7
  \item \( \text{NP} \) \hspace{1cm} \( \text{NP} \)
\end{itemize}

Time complexity in \( O(n^{v+1}) \)
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as items \( \langle i, j, \text{Cat} \rangle \).

Time complexity in \( O(n^{v+1}) \)
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as items \( (i, j, \text{Cat}) \).

Time complexity in \( O(n^{v+1}) \)
CKY table entries visually represented by edges and stored as items \( \langle i, j, \text{Cat} \rangle \).

Time complexity in \( O(n^{v+1}) \)
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as items \( \langle i, j, \text{Cat} \rangle \).

Time complexity in \( O(n^{v+1}) \)
CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as items $\langle i, j, \text{Cat} \rangle$.

Time complexity in $O(n^{y+1})$
Active chart and dotted rules

An active chart not only store recognized constituents but also partial ones.

Use of

- dotted rules

- edges labeled by dotted rules (items \( \equiv \langle i, j, A \leftarrow \alpha \bullet \beta \rangle \))

- a deductive system specifying how to derive items
CKY as a deductive system

\[ \langle i, i, A \leftarrow \bullet \alpha \rangle \]

\[ \langle i, j, A \leftarrow \alpha \bullet a\beta \rangle \]

\[ \langle i, j + 1, A \leftarrow \alpha a \bullet \beta \rangle \]

\[ a = a_{j+1} \]

\[ \langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \]

\[ \langle j, k, B \leftarrow \gamma \bullet \rangle \]

\[ \langle i, k, A \leftarrow \alpha B \bullet \beta \rangle \]

(Seed)

(Scan)

(Check)

Notion of Parsing as Deduction F. Pereira & D.H.D. Warren
Each item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ satisfies the invariant: $\alpha \rightarrow^* a_{l+1} \ldots a_r$

Using dotted rules provides implicit binarization

$\Rightarrow O(n^3)$ time complexity
Possibility to use a (top-down) predictive rule $\implies$ **Earley algorithm** [1970]

\[
\begin{align*}
\langle i, j, A \leftarrow \alpha \cdot B \beta \rangle \quad & \quad \exists B \leftarrow \gamma \\
\langle j, j, B \leftarrow \gamma \rangle \\
\end{align*}
\]

+ rules (Scan) and (Complete) (but not (Seed))

\[
\begin{align*}
\langle i, j, A \leftarrow \alpha \cdot B \beta \rangle \quad & \quad \langle j, k, B \leftarrow \gamma \cdot \rangle \\
\langle i, k, A \leftarrow \alpha B \cdot \beta \rangle \\
\end{align*}
\]

\[
\begin{align*}
(A \leftarrow \alpha B \cdot \beta) \\
(B \leftarrow \gamma \cdot k) \\
\end{align*}
\]

(Pred)

(Check Complete)
Each item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ satisfies two invariants:

1. Recognition of $\alpha$ between $l$ and $r$ (as for CKY)
2. **prefix validity**: $\exists \gamma \in (\Sigma \cup \mathcal{N})^*, S \rightarrow^* a_1 \ldots a_l A \gamma$

Worst-case time complexity remains $O(n^3)$
But in practice, prediction cuts search space and reduces complexity.
A chart algorithm relies on:

- a **table** (i.e. chart) where are stored the items, **without duplicates**.
- an **agenda** where are stored items to be treated
A chart algorithm relies on:
- a **table** (i.e. chart) where are stored the items, **without duplicates**.
- an **agenda** where are stored items to be treated

One algorithm step implies:
1. **Select an item** \( I \) **in the agenda**
2. **Unless** \( I \) **already tabulated, store it** ; otherwise move to step 1
3. **Build new items by combining** \( I \) **with tabulated items**
4. **Insert the new items in the agenda**
A chart algorithm relies on:

- a **table** (i.e. chart) where are stored the items, **without duplicates**.
- an **agenda** where are stored items to be treated.

One algorithm step implies:

1. Select an item \(I\) in the agenda
2. Unless \(I\) already tabulated, store it ; otherwise move to step 1
3. Build new items by combining \(I\) with tabulated items
4. Insert the new items in the agenda

**Variant**: The items are **first** tabulated then inserted in the agenda.

Earley selection order: \(\langle i, j, A \rangle\) selected before \(\langle k, l, B \rangle\) if \(j < l\) :

\[ \Rightarrow \text{left-to-right synchronized scanning} \]

For CFGs, the selection order is not so important (finite universe): the algorithm terminates and is complete.
Variants

Possibility of better filtering out useless steps (CKY examples)

\[
\begin{align*}
\langle i, j, B \leftarrow \beta \bullet \rangle & \quad \exists A \leftarrow B\alpha \\
\langle i, i, A \leftarrow \bullet B\alpha \rangle & \quad A \leftarrow \bullet B\alpha
\end{align*}
\]

(filteredSeed)

Possibility to merge several steps (CKY example)

\[
\begin{align*}
\langle i, j, B \leftarrow \beta \bullet \rangle & \quad \exists A \leftarrow B\alpha \\
\langle i, j, A \leftarrow B \bullet \alpha \rangle & \quad A \leftarrow B \bullet \alpha
\end{align*}
\]

(greedySeed)
Description of parsing strategies in terms (of classes) of partial parse trees

[Sikkel] “These intermediate results are not necessarily partial trees, but they must be objects that denote relevant properties of those partial parses.”

A schema indicates
- the domain of items (and their form)
- the item invariants

Very close from chart algorithms
Left-Corner Strategies

\[ \langle i, j, B \leftarrow \beta \bullet \rangle \langle k, i, X \leftarrow \nu \bullet R \mu \rangle \]
\[ \langle i, j, A \leftarrow B \bullet \alpha \rangle \]
\[ \exists A \leftarrow B\alpha \text{ avec } A \angle R \]

(LCSeed)

\[ \langle k, i, A \leftarrow \alpha \bullet B \beta \rangle \]
\[ \langle i, i + 1, C \leftarrow a \bullet \gamma \rangle \]
\[ \exists C \leftarrow a\gamma \text{ avec } a_{i+1} = a \angle B \]

(LCPred)

B, C, D left corners of A, denoted by \( D \angle A \)
Bidirectional scanning: Head driven parsing

Chart parsers not restricted to left-to-right scanning

bidirectional scanning is possible, for instance with strategies driven by **syntactic heads**.

- similar to left-corner parsing strategies, except that the head is not necessarily the following word
- mixed top-down & bottom-up parsing strategies
- 2- or 4-positions items

\[
\langle l, r, A \leftarrow \alpha \bullet \beta H \gamma \bullet \delta \rangle \quad \langle l, r, A \leftarrow \alpha \bullet \beta H \gamma \bullet \delta, al, ar \rangle
\]
Chart parser limitations

Large variety of items and deductive systems  
⇒ allow coupling tabulation with many parsing strategies

but still difficult with some strategies

Need of a rule like (Complete) using the recognition of a constituent to advance

Characterize bottom-up strategies, with or without some top-down prediction  
⇒ a strictly top-down parsing strategy can’t be expressed with a chart parser (or parsing schemata)
1. CKY
2. Chart Parsing
3. Generalized LR
4. Shared Forests
Originally described by Knuth (1965) and mostly used by programming language compilers (YACC, bison) to process deterministically CFG sub-classes [Aho, Ulman, and Hopcroft 1972].

Adapted for non-deterministic CFGs as found for natural languages [GLR – Tomita 1985].

Principle:

- **L**: Left-to-right scanning  
  Scan rightward while the current prefix is a valid one

- **R**: Right-to-left reduction  
  Reduce when a production has been fully recognized
LR strategy combines left corner and prefix sharing. Based on the computation of the closure and goto relations.

**Closure** of \( A \leftarrow \alpha \bullet B \beta \) includes all dotted rules \( C \leftarrow \bullet \gamma \) with \( C \) left corner of \( B \).

**Goto** of \( A \leftarrow \alpha \bullet B \beta \) is \( A \leftarrow \alpha B \bullet \beta \).

The “**the finite state grammar automaton**” defined by:

- states are closures
- A “goto B” transition exists from \( S_1 \) to \( S_2 \) if there exists \( A \leftarrow \alpha \bullet B \beta \in S_1 \) and \( A \leftarrow \alpha B \bullet \beta \in S_2 \)
LR automaton

\[
\begin{array}{l}
S \gets \bullet NP \ VP \\
NP \gets \bullet PN \\
NP \gets \bullet Det \ N \\
NP \gets \bullet NP \ PP \\
\end{array}
\]
LR automaton

\[ S \leftarrow \bullet \text{NP VP} \]

\[ \text{NP} \leftarrow \bullet \text{PN} \]

\[ \text{NP} \leftarrow \bullet \text{Det N} \]

\[ \text{NP} \leftarrow \bullet \text{NP PP} \]

\[ \text{PN} \]

\[ \text{NP} \leftarrow \text{PN} \bullet \]

\[ \text{Det} \]

\[ \text{NP} \leftarrow \text{Det} \bullet \text{N} \]
LR automaton

\[
\begin{align*}
S & \rightarrow \text{NP} \cdot \text{VP} \\
\text{NP} & \rightarrow \text{NP} \cdot \text{PP} \\
\text{VP} & \rightarrow \text{V} \cdot \text{NP} \\
\text{VP} & \rightarrow \text{VP} \cdot \text{PP} \\
\text{PP} & \rightarrow \text{Prep} \cdot \text{NP}
\end{align*}
\]
LR automaton

S ← NP VP
VP ← VP • PP
PP ← • Prep NP

VP

S ← NP • VP
NP ← NP • PP
VP ← • V NP
VP ← • VP NP
PP ← • Prep NP

NP

S ← • NP VP
NP ← • PN
NP ← • Det N
NP ← • NP PP

PN

NP ← PN

Det

NP ← Det • N
LR tables

Automaton exploited through 2 tables:

- **action** table: shift (s<state>), reduction (r<prod>), ...
- **goto** table: g<state>

<table>
<thead>
<tr>
<th>state</th>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PN DET N V PREP end</td>
<td>NP PP VP S</td>
</tr>
<tr>
<td>1</td>
<td>s2 s3</td>
<td>g4</td>
</tr>
<tr>
<td>2</td>
<td>rNP1 rNP1 rNP1 rNP1 rNP1</td>
<td>g0</td>
</tr>
<tr>
<td>3</td>
<td>s12</td>
<td>g6 g5</td>
</tr>
<tr>
<td>4</td>
<td>s7 s8</td>
<td>g6 g5</td>
</tr>
<tr>
<td>5</td>
<td>s8/rS2</td>
<td>g9</td>
</tr>
<tr>
<td>6</td>
<td>rNP2 rNP2 rNP2 rNP2 rNP2</td>
<td>g10</td>
</tr>
<tr>
<td>7</td>
<td>s2 s3</td>
<td>g10</td>
</tr>
<tr>
<td>8</td>
<td>s2 s3</td>
<td>g11</td>
</tr>
<tr>
<td>9</td>
<td>rVP2 rVP2 rVP2 rVP2 rVP2</td>
<td>g6</td>
</tr>
<tr>
<td>10</td>
<td>rVP2 rVP2 rVP2 rVP2 s8/rVP2</td>
<td>g6</td>
</tr>
<tr>
<td>11</td>
<td>rPP2 rPP2 rPP2 rPP2 s8/rPP2</td>
<td>g6</td>
</tr>
<tr>
<td>12</td>
<td>rNP2 rNP2 rNP2 rNP2 rNP2</td>
<td>g11</td>
</tr>
</tbody>
</table>

Potential existence of conflicts *shift/reduce* or *reduce/reduce*.
Using the LR tables

The LR tables guide the actions of a Push-Down Automata:

- Stacks formed of triples (state, terminal or non-terminal, position).
- A shift action pushes a new state
- A reduce action for a production $P_u : A \leftarrow A_1 \ldots A_n$ pops $n$ states and, pushes the state given by “$goto(\sigma, A)$” where $\sigma$ topmost state

The conflicts could be handled by backtracking, but exponential time complexity and potential loops
Tabular algorithm:

- All alternatives are explored (in case of conflicts)
- Maximum sharing of sub-stacks

\[ \text{⇒ graph-structured stacks or cactus stacks.} \]
Tabular algorithm:

- All alternatives are explored (in case of conflicts)
- Maximum sharing of sub-stacks

$\Rightarrow$ graph-structured stacks or cactus stacks.
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Trace for “John observes a man with a telescope”
Ensures a time complexity in $O(n^{v+1})$, $v$ length of longest production
Modifiable to ensure $O(n^3)$ time complexity
$O(n^2)$ space complexity (for recognizer)
Variants to handle cyclic grammars

Invariant no-longer expressed in terms of partial parse trees but of derivatives of the PDA
1. CKY
2. Chart Parsing
3. Generalized LR
4. Shared Forests
Language ambiguity $\implies$ Several possible parses per sentence!

Forest $\equiv$ set of parse trees

Shared (or packed) forest $\equiv$ Compact forest representation sharing identical or similar sub-trees.
Syntactic ambiguities and sharing

We can observe **common subparts**
AND-OR graphs may also be formalized as hyper-graphs.
John observes a man with a telescope.
Shared forests and grammars

A forest is a grammar $G'$ instance of $G$ [Lang].

0 John 1 observes 2 a 3 man 4 with 5 a 6 telescope 7

s --> np vp
np --> pn
np --> det n
np --> np pp
vp --> v np
vp --> vp pp
pp --> prep np

s07 --> np01 vp17
np01 --> pn01
vp17 --> v12 np27
vp17 --> vp14 pp47
np27 --> np24 pp47
n37 --> n34 pp47
np24 --> det23 n34
pp47 --> prep45 np57
np57 --> det56 n67
vp14 --> v12 np24
pn01 --> John
v12 --> observes
det23 --> a
n34 --> man
prep45 --> with
det56 --> a
n67 --> telescope

Some non-terminals (vp17) multiply defined (ambiguities).
Some non-terminals (v12, np24, pp47) are used several times (sharing).
Actually, a shared forest is the intersection of a grammar with a regular language (generated by a Finite State Automaton [FSA]).

\[ L(G') = L(G) \cap \{ \text{"John observes a man with a telescope"} \} \]

Bar Hillel 1964

The intersection of a context free language with a regular language is again a context free language.
Intersecting with a FSA

Given a CFG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ and a FSA $A = (Q, \Sigma, \delta, I, F)$, we construct $G_\cap = (\mathcal{N} \times Q \times Q, \Sigma \times Q \times Q, \langle S, I, F \rangle, \mathcal{P}')$

For each $A_0 \leftarrow A_1 \ldots A_n \in \mathcal{P}$, add to $\mathcal{P}'$

$$\langle A_0, q_0, q_n \rangle \leftarrow \langle A_1, q_0, q_1 \rangle \ldots \langle A_n, q_{n-1}, q_n \rangle$$

with

$$\forall i \in \{1, \ldots, n-1\}, \left\{ \begin{array}{l} A_{i+1} \in \mathcal{N} \implies (q_i, q_{i+1}) \in Q \\ A_{i+1} \in \Sigma \implies (q_i, A_{i+1}, q_{i+1}) \in \delta \\ A_{i+1} = \epsilon \implies q_i = q_{i+1} \end{array} \right.$$ 

We show that

$$L(G) \cap L(A) = L(G_\cap)$$

Construction in time $O(|G|.|Q|^{n+1})$, where $n$ is length of longest clause.

Many useless productions in $\mathcal{P}'$ need **grammar reduction**: removal of non reachable clauses from axiom ∼ parsing !
For parsing, input string may be replaced by input FSAs

\[ L(G') = L(G) \cap L(FSA) \]

“[unreadable word] observes [unreadable words] with a telescope”

0 ——— 1 ——— 2 ——— 3 ——— 4 ——— 5

observes with a telescope
Parse forest for an incomplete sentence

S
  NP
    PN
    VP
      V
        NP
          Prep
            Prep
              with
                Det
                  a
                    N
                      telescope
            PP
              NP
                Det
                  N
The tree structure shows the syntactic relationships between the words: 

- S (sentence)
- NP (noun phrase)
- VP (verb phrase)
- V (verb)
- Det (determiner)
- N (noun)
- Prep (preposition)

The words 'observes' and 'telescope' are present in the parse forest, indicating that the sentence possibly starts with "observes" and ends with "telescope". The structure suggests that there could be a missing part of the sentence between these words, possibly another noun phrase or a prepositional phrase.
Grammar for an incomplete sentence

\[
\begin{align*}
s05 & \rightarrow \text{np01} \quad \text{vp15} \\
np01 & \rightarrow \text{pn01} \\
\text{vp15} & \rightarrow \text{v12} \quad \text{np25} \\
\text{vp15} & \rightarrow \text{vp12} \quad \text{pp25} \\
np25 & \rightarrow \text{np22} \quad \text{pp25} \\
\text{vp12} & \rightarrow \text{vp12} \quad \text{pp22} \\
\text{vp12} & \rightarrow \text{v12} \quad \text{np22} \\
\text{pp25} & \rightarrow \text{prep22} \quad \text{np25} \\
\text{pp25} & \rightarrow \text{prep23} \quad \text{np35} \\
np22 & \rightarrow \text{np22} \quad \text{pp22} \\
np22 & \rightarrow \text{det22} \quad \text{n22} \\
np22 & \rightarrow \text{pn22} \\
\text{pp22} & \rightarrow \text{prep22} \quad \text{np22} \\
np35 & \rightarrow \text{det34} \quad \text{n45}
\end{align*}
\]

\[
\begin{align*}
\text{pn01} & \rightarrow * \\
\text{v12} & \rightarrow \text{observes} \\
\text{pn22} & \rightarrow * \\
\text{det22} & \rightarrow * \\
\text{n22} & \rightarrow * \\
\text{prep22} & \rightarrow * \\
\text{prep23} & \rightarrow \text{with} \\
\text{det34} & \rightarrow \text{a} \\
\text{n45} & \rightarrow \text{telescope}
\end{align*}
\]
Tabular parsers easily modifiable to take as input an FSA (or a word lattice)

Parsing an FSA done with time complexity in $O(n^3)$ for CFGs where $n$ is the number of states of the FSA.

FSAs (or word lattice) useful for
- noisy or incomplete sentences (speech data)
- lexical ambiguities
- segmentation ambiguities

FSAs Possibly with probabilities or weights (weighted FSAs)

Same results extend for most grammatical formalisms (Unification grammars, TAGs, LIGs, ...)
Forest extraction

Shared forests may be built or extracted post parsing.

Extraction uses **backpointers** from tabulated object to their parents

Starting from answer \((S_{07})\), backpointers are followed to retrieve instantiated productions and identify non-terminals

Note: Space complexity increases from \(O(n^2)\) to \(O(n^3)\) for binarized grammars
Part II

Towards Unification-based grammars
CFG in practice

CFG are not really adequate for fine-grained descriptions!

s → np vp
np → pn
np → det n
np → np pp
vp → v np
vp → vp pp
pp → prep np
s → vp % imperative

How to rule out?

- *il manges les pomme
- *mangeront la pomme
Duplicate CFG rules!

\[ s \rightarrow \text{np}_p\text{1\_sing } \text{vp}_p\text{1\_sing} \]
\[ s \rightarrow \text{np}_p\text{1\_pl } \text{vp}_p\text{1\_pl} \]
\[ s \rightarrow \text{np}_p\text{2\_sing } \text{vp}_p\text{2\_sing} \]
\[ s \rightarrow \text{np}_p\text{2\_pl } \text{vp}_p\text{2\_pl} \]
\[ s \rightarrow \text{np}_p\text{3\_sing } \text{vp}_p\text{3\_sing} \]
\[ s \rightarrow \text{np}_p\text{3\_pl } \text{vp}_p\text{3\_pl} \]

but also

\[ \text{np}_p\text{3\_sing } \rightarrow \text{det}_m\text{asc\_sing } \text{n}_m\text{asc\_sing} \]
\[ \text{np}_p\text{3\_sing } \rightarrow \text{det}_f\text{em\_sing } \text{n}_f\text{em\_sing} \]
\[ \text{np}_p\text{3\_pl } \rightarrow \text{det}_m\text{asc\_pl } \text{n}_m\text{asc\_pl} \]
\[ \text{np}_p\text{3\_pl } \rightarrow \text{det}_f\text{em\_pl } \text{n}_f\text{em\_pl} \]

and

\[ s \rightarrow \text{vp\_imperative} \]
\[ \text{vp\_imperative } \rightarrow \text{v\_imperative } \text{np} \]

actually, need to combine all these bits of informations
\[ \Rightarrow \text{greatly increase the number of relatively similar productions} \]
Using *underspecified rules* with variables ranging over (finite) set of values

\[
\begin{align*}
\text{s} \rightarrow & \ np(P,G,N) \ vp((P,N,M)). \\
\text{s} \rightarrow & \ vp(P, \text{imperative}). \\
np(3,G,N) \rightarrow & \ det(G,N) \ n(G,N).
\end{align*}
\]

Alternate notations

\[
\begin{align*}
\text{s} \rightarrow & \\
np \{ \text{person} \rightarrow P, \text{number} \rightarrow N \}, \\
vp \{ \text{person} \rightarrow P, \text{number} \rightarrow N \}. \\
\text{s} \rightarrow & \ vp \{ \text{mood} \rightarrow \text{imperative} \}. \\
np \{ \text{person} \rightarrow 3, \text{gender} \rightarrow G, \text{number} \rightarrow N \} \rightarrow \\
det \{ \text{gender} \rightarrow G, \text{number} \rightarrow N\}, \\
n \{ \text{gender} \rightarrow G, \text{number} \rightarrow N\}.
\end{align*}
\]

The abstracted rules and possible instantiations may be used to generate CFG rules, but large number of CFG rules

Also, wish of richer instantiations, with no finite expansion

\[\rightarrow\text{ Better to move to Unification Grammars}\]
Unification Grammars

Decorated non-terminals
but no fundamentally different rule applications

Horn Clauses
DCG
LFG
HPG

λ-Prolog

literal complexity

CFG productions & Horn clauses are very similar

\[ S \leftarrow \text{NP VP} \leadsto S(X_0, X_2) \leftarrow \text{NP}(X_0, X_1) \text{ VP}(X_1, X_2). \]

- Allow information propagation from one point to another logical variables, reentrenncy
- Allow underspecification (partial information)
LFG and Feature Structures

Charts revisited for Unification Grammars

Push-Down Automata
**Lexical Functional Grammars [LFG]**

**Bresnam et Kaplan** (1982) *The mental representation of grammatical representation*

Théorie: Associate constituent structures (*c*-structures) & functional structures (*f*-structure):

```
S
 |   
NP  VP
 |   |
Pierre  V  NP
     |  |
  admire  NP  
       |  |
       Mary
```

```
S
 |   
NP  NP  V
 |   |
Petrus  puellam  amat
     |  |
      
```

```
[Subj  [Pred pierre]
Pred  admirer
Obj  [Pred mary]
]

[Subj  [Pred pierre]
Pred  admirer
Obj  [Pred mary]
]```
A grammar is given as CFG productions whose non-terminals are decorated by functional equations.

\[
\begin{align*}
S & \rightarrow \ NP \ V \ NP \\
\ & \rightarrow \ John \ NP \\
\ & \rightarrow \ sleep\ V \\
John & \rightarrow \ (↑Subj)=↓ \ ↑=↓ \\
(↑Num)= & \ sing \\
(↑Gender)= & \ masc \\
(↑Pred)= & \ 'John' \\
V & \rightarrow \ sleep\ \\
(↑Subj Num)= & \ sing \\
(↑Subj Pers)= & \ 3 \\
(↑Mood)= & \ indicative \\
(↑Pred)= & \ 'sleep<Subj>' \\
\end{align*}
\]
Formalism: Feature structure

FS may be seen as property-value records,
- possibly with FS as values (recursion)
- possibly with reentreny (shared FS)

Generally represented as Attribute Value Matrix

\[
\begin{pmatrix}
\text{Det} & \text{Agr} [1] & \text{Num sing} \\
\text{Cat Det} & \text{Gender masc} \\
\text{Nom} & \text{Agr} [1] \\
\text{Cat N} & 
\end{pmatrix}
\]
Graph notation

Feature structures may be formalized as acyclic directed graphs (maybe extended with cycles)

Note: leads to a notion of path for a sequence of features, in graph and AVM
ex: chemin spec.accord.gender
FS: formalization

We suppose given a signature $S = (V, F)$ where $F$ is a finite set of properties/features and $V$ a set of atomic values.

Formally, a FS $A$ over $S$ is denoted by

$$(Q_A, r_A, \delta_A, \theta_A)$$

where:

- $Q_A$ is a set of states
- $r_A \in Q_A$ is the root state
- $\delta_A : Q_A \times F \leftarrow Q_A$ is a partial function for following features such that each state in $Q_A$ is reachable from $r_A$ by reflexive-transitive closure of $\delta_A$
  
  i.e. $\forall q \in Q_A, \ q = r_A \lor \exists (q', f), \ \delta(q', f) = q$

- $\theta_A : Q_A \leftarrow V$, a partial labeling function only defined on terminal states, i.e. $q \in Q_A, \forall f \in F, \delta(q, f) \uparrow$

Path $\pi(A)$ defined as $\{p \in F^* | \delta(r_A, p) \downarrow\}$

$p_1 \neq p_2$ are 2 reentrant paths iff $\delta(r_A, p_1) = \delta(r_A, p_2) \downarrow$
FS Subsumption

FS may be seen as *specifying information* (properties of entities)

\[ A \sqsubseteq B \] if \( A \) more general than \( B \)
or alternatively \( A \) less constraint than \( B \)

\[ \Rightarrow \sqsubseteq \text{ is a partial pre-order on feature structures} \]
Sketch of an algorithm:

\[ A \sqsubseteq B \text{ iff for each path } p \text{ in } A, \text{ there exists a path } p.q \text{ in } B \]

but beware of reentrancy!
FS unification

Unification accumulates partial information:

Formally, most general instance of $A$ and $B$

$$A \sqcup B = C, \text{ such that } \forall D, \ A \sqsubseteq D \land B \sqsubseteq D \implies C \sqsubseteq D$$
Typed Feature Structures

Formalized by B. Carpenter and used in HPSG (*Head-driven Phrase Structure Grammars*).

FS are typed, with types $\tau$ in some finite multiple-inheritance hierarchy:

- $\tau$ may have several parents
- $\tau$ may introduce authorized features $f$,
  with most general appropriate type $\rho_{\tau,f}$ for values
- $\tau$ may further instanciate a feature introduced by an ancestor
Fragment of a type hierarchy
*(Semantic-Head-Driven Generation, Shieber et al, in ALE)*
LFG: richer equations

Constraints on existence

\[ \text{NP} \rightarrow (\text{Det}) \quad \text{N} \rightarrow \text{Jean} \quad \text{N} \rightarrow \text{chien} \]

\[ \uparrow = \downarrow \quad \uparrow = \downarrow \]

\[ \downarrow \quad \downarrow \]

\[ \text{Det} \rightarrow \text{le} \quad (\uparrow \text{Det})=\text{le} \]

Constraint equations

\[ \text{S'} \rightarrow \text{NP} \quad \text{S} \rightarrow (\downarrow \text{Wh})=c+ \quad (\uparrow \text{Wh})=+ \]

\[ \uparrow = \downarrow \]

Set equations

\[ \text{VP} \rightarrow \text{V} \quad (\text{NP}) \quad (\text{PP})^* \]

\[ \uparrow = \downarrow \quad \uparrow \text{Obj}=\downarrow \quad \uparrow \text{Adjunct} \exists \downarrow \quad \uparrow \text{Adjunct} \exists \downarrow \]

(Jean dort le matin. Jean mange le gateau Jean mange ce gateau avec Anne)
Grammatical functions

Possible functions: Subject, Object, Comp(letive), XComp (infinitives and participiales), Prep-Obj (prepositional complements)

Vcomp  Jean veut **partir à Rio**.
Acomp  Jean devient **fou**.
Ncomp  Ils ont élu Jean **président**
Vajout  **Partant en voyage**, Marie se prépare
Aajout  Paul est parti **content**
Prep-Obj  Paul ressemble à **Jean**
The **Pred** feature states the expected functions for a word

- **manger** \((\uparrow\text{Pred})='\text{manger}<\text{Suj,Obj}>'\)
- **donner** \((\uparrow\text{Pred})='\text{donner}<\text{Suj,Obj,A-Obj}>'\)
- **falloir** \((\uparrow\text{Pred})='\text{falloir}<\text{Obj}>\text{Suj}' \text{ et } (\uparrow\text{Suj Form})=c \text{ il}\)
- **vouloir** \((\uparrow\text{Pred})='\text{vouloir}<\text{Suj,Vcomp}>' \text{ et } (\uparrow\text{Suj})=(\uparrow\text{Vcomp Suj}) \text{ Jean veut}\)
  \text{venir}
- **proposer** \((\uparrow\text{Pred})='\text{proposer}<\text{Suj,A-Obj,Vcomp}>' \text{ et } (\uparrow\text{Vcomp Suj})=(\uparrow\text{Suj})/(\uparrow\text{A-Obj})\)
  Jean propose à Jean de venir
- **destruction** \((\uparrow\text{Pred})='\text{destruction}<\text{De-Obj,Par-Obj}>'\) **Destruction de la maison par les promoteurs**
Extractions

\[ P' \rightarrow SN \quad P \]

\[ (\downarrow Qu) = c + \quad \uparrow = \downarrow \]
\[ (\uparrow Focus) = \uparrow \quad (\downarrow Qu) = + \]
\[ (\uparrow Focus) = (\uparrow Obj) \]

demande, V: \quad (\uparrow Pred) = 'demander<Suj, Comp>'
\[ (\uparrow Comp Qu) = c + \]

quel, Det: \quad (\uparrow Det) = 'quel'
\[ (\uparrow Qu) = + \]

SN \quad P \quad SV

\[ \text{Jean} \quad \text{demande} \quad \text{quel} \quad \text{homme} \quad \text{Marie} \quad \text{regarde} \]
Arbitrary number of embeddings between an extract constituent and its associated predicates:

Jean demande [quel homme Paul pense [que Marie regarde \( \epsilon \)]]

\[
S' \quad \rightarrow \quad NP \\
\text{(\( \downarrow \text{Wh} \)) =}_c + \\
\text{(\( \uparrow \text{Focus} \)) =}_c + \\
\text{(\( \uparrow \text{Focus} \)) =}_c + (\text{Comp})^* \text{Obj}
\]
Very easy for Unification Grammars to have the power of a Turing machine!

Essentially, because of recursive feature structures

Nevertheless, interesting to explore parsing algorithms for UG

Note: actually, decorations and unification may be added to most base formalism
LFG and Feature Structures

Charts revisited for Unification Grammars

Push-Down Automata
Unification \((N & sing)\) is used to glue partial parse trees
Existence of information flow propagated thanks to substitutions \((N / sing)\)
In inference rules, **unification** used to combine items.

\[
\begin{align*}
\langle i, j, A \leftarrow \alpha \bullet B \beta \rangle & \quad \langle j, k, C \leftarrow \gamma \bullet \rangle \\
\frac{}{\langle i, k, (A \leftarrow \alpha B \bullet \beta) \sigma \rangle}
\end{align*}
\]

\[\sigma = \text{mgu}(B, C) \quad \text{(Complete)}\]

\[
\begin{align*}
\langle i, j, A \leftarrow \alpha \bullet B \beta \rangle & \quad \langle j, j, (C \leftarrow \gamma) \sigma \rangle \\
\frac{}{\exists C \leftarrow \gamma \text{ and } \sigma = \text{mgu}(B, C) \quad \text{(Pred)}}
\end{align*}
\]

\[
\begin{align*}
\langle i, j, A \leftarrow \alpha \bullet a \beta \rangle & \quad \langle j, j + 1, A \leftarrow \alpha a \bullet \beta \rangle \\
\frac{}{a = a_{j+1} \quad \text{(Scan)}}
\end{align*}
\]
Renaming of item variables before rule application

- Traditionally, productions are renamed before use (Prolog)

\[
\begin{align*}
q(X) & \leftarrow \bullet q(f(X)) \\
q(f(X)) & \leftarrow \bullet q(f(f(X)))
\end{align*}
\]

\[
\exists q(X') \leftarrow q(f(X')) \text{ et } \sigma = \{X'/f(X)\} \quad \text{(Pred)}
\]

Failure if no renaming of \(X/X'\) in production \(q(X) \leftarrow q(f(X))\)

- But require also item renaming, for instance for (Complete)
Item redundancy checking by simple identity not longer enough because of renaming \((q(X) \neq q(X'))\)

Need more powerful redundancy checking

**Variance** Items identical modulo variable renaming

\(q(X)\) variant of \(q(X')\).

**Subsumption** Logical terms are ordered by \(\preceq\)

\[
A \preceq B \iff \exists \sigma, \ B = A\sigma \begin{cases} 
A \text{ generalizes } B \\
A \text{ subsumes } B \\
B \text{ is an instance of } A
\end{cases}
\]

Examples: \(g(X, Y) \preceq g(Z, Z) \preceq g(f(a), f(a))\)

An item is not tabulated if it is an instance of an already tabulated item

**Justification**: Each item \(J'\) derivable from \(I'\) instance of \(I\) is instance of some item \(J\) derivable from \(I\).
For the program

\[
q(X) \leftarrow q(f(X)). \\
q(f(f(a))).
\]

and goal \( ? \leftarrow q(X) \), the expected answers are: \( X = f(f(a)) \), \( X = f(a) \), \( X = a \)

Loops with variance test (+ computation duplication)

\[
\bullet q(X) \leftarrow \bullet q(f(X)) \leftarrow \bullet q(f(f(X))) \leftarrow \bullet q(f^n(X)) \leftarrow \ldots
\]

\[
\bullet q(a) \leftarrow \bullet q(f(a)) \leftarrow \bullet q(f(f(a))) \leftarrow \ldots
\]
Loops: variance vs subsumption

For the program

\[ q(X) : - q(f(X)). \]
\[ q(f(f(a))). \]

and goal \(- q(X),\) the expected answers are: \( X = f(f(a)), \ X = f(a), \ X = a \)

Loops with variance test (+ computation duplication)

Loops with variance test (+ computation duplication)

\[ \bullet q(X) \rightarrow \bullet q(f(X)) \rightarrow \bullet q(f(f(X))) \rightarrow \bullet q(f^n(X)) \rightarrow \ldots \]

\[ q(a) \rightarrow q(f(a)) \rightarrow q(f(f(a))) \]

Terminates when using subsumption

\[ q(a) \rightarrow q(f(a)) \rightarrow q(f(f(a))) \]

Trade-off between simple variance and more precise and costly subsumption
Termination not always ensured, even using subsumption.

The item family with growing sub-terms \( f(a), f(f(a)), \ldots, f^n(a) \) not cut by subsumption (spiral case)

First remedy: only consider Datalog grammars (i.e. without function symbol \( f \))

But not satisfactory!

Two origins to loops

1. Loops due to answers
2. Loops due to predictions
Loops due to answers

A program or grammar produces an infinite set of answers due to a loop during answer propagation.

In generation mode, \texttt{append} produces infinitely many answers

\begin{verbatim}
append([], Y, Y).
append([A \mid X], Y, [A \mid Z]) :- append(X, Y, Z).
\end{verbatim}

\begin{align*}
\text{append}([], Y, Y) \\
\text{append}([A], Y, [A \mid Y]) \\
\sim \\
\text{append}([A,B], Y, [A,B \mid Y]) \\
\ldots
\end{align*}

Rare in Parsing, possible in Generation.

\textbf{Solution:} No real solution, except using \textit{finitely ambiguous} grammars.
Off-line parsable grammars are finitely ambiguous:

[Shieber] There exists a projection $\rho$ towards a finite domain generalizing parse trees. i.e. $\rho \tau \preceq \tau$, in such a way that no projected tree $\rho \tau$ is its own sub-tree for a given input string.

In particular, if satisfied when projecting to the CF backbone, then the grammar is off-line parsable.
Off-line parsable grammars are finitely ambiguous:

Shieber] There exists a projection $\rho$ towards a finite domain generalizing parse trees. i.e. $\rho \tau \preceq \tau$, in such a way that no projected tree $\rho \tau$ is its own sub-tree for a given input string.

In particular, if satisfied when projecting to the CF backbone, then the grammar is off-line parsable.
But existence of off-line parsable grammars whose CF backbone is cyclic.

\[
q(f(f(a)), u) \leadsto q(\_, u) \leadsto q(\_, \_) \\
q(f(a), v) \quad q(\_, v) \quad q(\_, \_) \\
q(a, w) \quad q(\_, w) \quad q(\_, \_) 
\]

In Logic Programming $\equiv$ data driven stratification
Non termination may arise from more and more precise predictions.

\[
q(f(f(a))). \quad ?- q(a).
\]

\[
q(X) :- q(f(X)).
\]
Cutting prediction loops

Prediction items may be generalized without altering neither correction nor answer completeness

Use of prediction restrictions [Shieber]

\[
\langle i, j, A \leftarrow \alpha \bullet B^\beta \rangle \\
\langle j, j, (C \leftarrow \bullet \gamma)^\sigma \rangle
\]

\[\exists C \leftarrow \gamma \text{ et } \sigma = mgu(\Phi B, C) \] (PredR)

with \(\Phi B\) generalization of \(B\) (\(\Phi B \preceq B\))

Idea: Transform spirals into loops that may be cut by subsumption.

\[\Phi q(f^{n \geq 1}(t)) = q(f(\_)) \implies q(a) \bullet \quad q(f(a)) \bullet \quad q(f(f(X))) \bullet \]
In parsing, used to cut prediction spirals:
- on constituent lists
- on trace lists (gap)

They can improve computation sharing by removing pieces of information not needed to guide computations (ex: semantic forms)

But they may also induce useless computations (over-generalization)

Note: in Logic Programming: Term depth abstraction
No tabular techniques can ensure a systematic termination

However, tabulation allows suspension and resuming of computations \( \rightarrow \) ensures computation completeness by scheduling in a fair way computation steps.

**fairness** No computation step can be forever ignored
Complexities may be exponential both in time and space

- Number of items (exponential in $n$) $\implies$ table look-up
- Term size (exponential in $n$)
- Access to variable values (constant to linear wrt derivation lengths)
- Occurrence checking (exponential wrt term size)

Furthermore, 2 costly operations: unification & subsumption.

Note: Polynomial complexity for Datalog programs and grammars
LFG and Feature Structures

Charts revisited for Unification Grammars

Push-Down Automata
Approach [Lang, De la Clergerie] relying on:

1. automata to describe the steps of a parsing strategy
   $\Rightarrow$ use of Push-Down Automata [PDA] working on “information-rich” stacks.
   Note: PDAs well-known for CFGs (equivalence)

2. Dynamic Programming principles to design tabular evaluations for these automata

Program Query
Grammar String

\[\begin{array}{c}
\text{Strategy} \\
\text{Compilation}
\end{array} \quad \text{non-deterministic automaton} \quad \begin{array}{c}
\text{Tabular} \\
\text{Evaluation}
\end{array} \quad \begin{array}{c}
\text{Answers} \\
\text{Parse trees}
\end{array}\]
PDA extension::

- Stacks of 1st order logical terms
- 3 transition kinds (PUSH, SWAP & POP).
- Unification used to apply transitions

\[ \sigma = \text{mgu}(A_1, B) \]
\[ \sigma = \text{mgu}(A_1, B) \]
\[ \sigma = \text{mgu}(A_1 A_2, BD) \]
Call of a non-terminal to recognize
Selection of a production
Publication of a recognized non-terminal
Return to the calling production
Parsing steps

- Call $C_B$
- Selection $C_C$
- Return $R_B$
- Selection of a production
- Publication $R_C$
- Publication of a recognized non-terminal
- Return to the calling production

---

\[ \text{... } \bullet B \ldots \]

\[ \text{... } \bullet B \ldots \]

\[ \text{... } \bullet B \ldots \]
Parsing steps

Call $C_B$  
Selection $C_C$  
Return $R_B$  
Publication $R_C$

Call of a non-terminal to recognize
Selection of a production
Publication of a recognized non-terminal
Return to the calling production

\[ \ldots \bullet B \ldots \]
\[ \ldots \bullet B \ldots \]
\[ \ldots \bullet B \ldots \]
\[ \ldots \bullet B \ldots \]
Parsing steps

- Call $C_B$
- Selection $C_C$
- Return $R_B$
- Publication $R_C$
- Call of a non-terminal to recognize
- Selection of a production
- Publication of a recognized non-terminal
- Return to the calling production
Call of a non-terminal to recognize
Selection of a production
Publication of a recognized non-terminal
Return to the calling production
Modulated Call/Return strategies

Approximation of each non-terminal $A$ by \[
\begin{cases}
C_A & \text{for Call & Selection steps} \\
R_A & \text{for Return & Publication steps}
\end{cases}
\]

- **[Select]**
  \[C_A \rightarrow A \leftarrow \bullet \ldots\]

- **[Publish]**
  \[A \leftarrow \ldots \bullet \rightarrow R_A\]

- **[Call]**
  \[A \leftarrow \ldots \bullet B \ldots \rightarrow C_B \rightarrow A \leftarrow \ldots \bullet B \ldots\]

- **[Return]**
  \[A \leftarrow \ldots \bullet B \ldots \rightarrow A \leftarrow \ldots B \bullet \ldots\]

**Strategy**

<table>
<thead>
<tr>
<th>Strategy</th>
<th>$C_A$</th>
<th>$R_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Down</td>
<td>$A$</td>
<td>$\perp$</td>
</tr>
<tr>
<td>Bottom-Up</td>
<td>$\perp$</td>
<td>$A$</td>
</tr>
<tr>
<td>Earley</td>
<td>$A$</td>
<td>$A'$</td>
</tr>
</tbody>
</table>

**Modulation validity:**

\["Information"(A) = "Information"(C_A) + "Information"(R_A)\]
Dynamic Programming  Recursive decomposition of a problem into simpler sub-problems that may be re-used (ex. knapsack problem).

For (L)PDAs, we try to

1. Identify elementary sub-derivations

2. Identify pertinent information in these derivations to build traces (items) as compact as possible.  
   (motivation: save space and improve computation sharing)

3. Combine these items to get a tabular evaluation sound and complete w.r.t. the standard derivations.
Context-free derivations

A \sim \text{PUSH} derivation representable:

[forgetting about instantiation] by pair (green, purple)
[taking into account instantiation] same pair + instantiation measure

transition properties \implies (green, purple) or even (purple)

Conclusion: An item is a PUSH derivation representable by a stack fragment.
PDA derivations may be retrieved by composition of items and transitions.

Composition of a POP transition with two items: \((A, B) \circ (B, C) \circ \tau = (A, D)\)
Item composition (without instantiation) III

[POP]  

\[ \text{Gluing} \]  

\[ \text{POP} \rightarrow \circ \equiv \]
Relationships with Graph-structured stacks

No instantiation (CFG case) 2-items & Graph-structured stacks are similar

With instantiation Not equivalent because of $\sigma$

Graph-structured stacks also factorize on $B$ (interesting when no instantiation)
Instead of PUSH transition, we consider $\epsilon$–PUSH that only examine a fraction $\epsilon$ of information on stack tops.

**Combination:** Similar to $S2$ but more complex combining

$S1 + \epsilon$ is sound and complete for PDDs using $\epsilon$–PUSH transitions.
Approximation of each non-terminal $A$ by

\[
\begin{cases}
C_A & \text{for each Call & Select} \\
R_A & \text{for each Return & Publish}
\end{cases}
\]

[S]elect \quad $C_{l.0}$ $\rightarrow$ $\nabla_{l.0}$

[P]ublish \quad $\nabla_{l.n}$ $\rightarrow$ $R_{l.0}$

[C]all \quad $\nabla_{k.i}$ $\rightarrow$ $C_{k.i+1}$

[R]eturn \quad $R_{k.i+1}$ $\rightarrow$ $\nabla_{k.i+1}$

PUSH Call transitions equivalent to $\epsilon$–PUSH with $\epsilon(\nabla_{k.i}) = C_{k.i+1}$.

\[
\begin{array}{c}
\nabla_{k.i} \\
C_{k.i+1} \downarrow \\
R_{k.i+1} \downarrow \\
\nabla_{k.i+1}
\end{array}
\cdots
\]
For bottom-up strategies (with or without prediction), i.e. $R_A \equiv A$, the stack topmost element holds a lot of information.

$$
\begin{array}{c}
A \\
C_A
\end{array}
$$

induces "info"$(C_A) \subset \text{"info"}(A)$

$\implies$ Possible to take the topmost stack elements as items

$S1$ interpretation similar to deductive systems

$$
\begin{array}{c}
A \\
B
\end{array} + \text{POP } \{(A, B) \to C\}
$$

equivalent

$$
\frac{A \cdot B}{C}
$$

$1 + \epsilon$-items as efficient as 1-items for a wider spectrum of strategies.