A dependency parse

Dependencies are (labeled) asymmetrical binary relations between two lexical items (words).

CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 12:
Dependency Parsing;
Expressive Grammars

Julia Hockenmaier
juliahr@illinois.edu
3324 Siebel Center

Parsing algorithms for DG

‘Transition-based’ parsers:
learn a sequence of actions to parse sentences
Models:
State = stack of partially processed items
+ queue/buffer of remaining tokens
+ set of dependency arcs that have been found already
Transitions (actions) = add dependency arcs; stack/queue operations

‘Graph-based’ parsers:
learn a model over dependency graphs
Models:
a function (typically sum) of local attachment scores
For dependency trees, you can use a minimum spanning tree algorithm

Dependency Parsing
Transition-based parsing: assumptions

This algorithm works for projective dependency trees.

**Dependency tree:**
Each word has a single parent
(Each word is a dependent of [is attached to] one other word)

**Projective dependencies:**
There are no crossing dependencies.
For any i, j, k with i < k < j: if there is a dependency between \( w_i \) and \( w_j \), the parent of \( w_k \) is a word \( w_l \) between (possibly including) \( i \) and \( j \) while any child \( w_m \) of \( w_k \) has to occur between (excluding) \( i \) and \( j \): \( i < m < j \)

Parser configurations (\( \sigma, \beta, A \))

The stack \( \sigma \) is a list of partially processed words
We push and pop words onto/off of \( \sigma \).
\( \sigma \mid w \) : \( w \) is on top of the stack.
Words on the stack are not (yet) attached to any other words.
Once we attach \( w \), \( w \) can’t be put back onto the stack again.

The buffer \( \beta \) is the remaining input words
We read words from \( \beta \) (left-to-right) and push them onto \( \sigma \)
\( w \mid \beta \) : \( w \) is on top of the buffer.

The set of arcs \( A \) defines the current tree.
We can add new arcs to \( A \) by attaching the word on top of the stack to the word on top of the buffer, or vice versa.
Parser configurations \((\sigma, \beta, A)\)

We start in the initial configuration \([w_0], [w_1, \ldots, w_n], \{\}\) \[(\text{Root token, Input Sentence, Empty tree)}\]

We can attach the first word \((w_1)\) to the root token \(w_0\), or we can push \(w_1\) onto the stack. \((w_0\) is the only token that can’t get attached to any other word)\]

We want to end in the terminal configuration \([\{}\], [\{}\], A)\]

\[(\text{Empty stack, Empty buffer, Complete tree)}\]

Success!
We have read all of the input words (empty buffer) and have attached all input words to some other word (empty stack)

Parser actions

\((\sigma, \beta, A)\): Parser configuration with stack \(\sigma\), buffer \(\beta\), set of arcs \(A\) \((w, r, w')\): Dependency with head \(w\), relation \(r\) and dependent \(w'\)

**SHIFT:** Push the next input word \(w_i\) from the buffer \(\beta\) onto the stack \(\sigma\)

\((\sigma, \beta, A) \Rightarrow (\sigma|w_i, \beta, A)\)

**LEFT-ARC:** \(\ldots w_i \ldots\) \(\ldots w_j \ldots\) (dependent precedes the head)

Attach dependent \(w_i\) (top of stack \(\sigma\)) to head \(w_j\) (top of buffer \(\beta\)) with relation \(r\) from \(w_i\) to \(w_j\). Pop \(w_i\) off the stack.

\((\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_j, r, w_i)\})\)

**RIGHT-ARC:** \(\ldots w_i \ldots w_j \ldots\) (dependent follows the head)

Attach dependent \(w_j\) (top of buffer \(\beta\)) to head \(w_i\) (top of stack \(\sigma\)) with relation \(r\) from \(w_i\) to \(w_j\). Move \(w_i\) back to the buffer.

\((\sigma|w_j, w_i|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})\)

Transition-based parsing

We process the sentence \(S = w_0w_1\ldots w_n\) from left to right (“incremental parsing”)

In the parser configuration \((\sigma|w_i, w_j|\beta, A)\):

- \(w_j\) is on top of the stack. \(w_i\) may have some children
- \(w_j\) is on top of the buffer. \(w_j\) may have some children
- \(w_i\) precedes \(w_j\) \((i < j)\)

We have to either attach \(w_i\) to \(w_j\), attach \(w_j\) to \(w_i\), or decide that there is no dependency between \(w_i\) and \(w_j\)

If we reach \((\sigma|w_i, w_j|\beta, A)\), all words \(w_k\) with \(i < k < j\) have already been attached to a parent \(w_m\) with \(i \leq m \leq j\)

An example sentence & parse

- Economic news had little effect on financial markets.
Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>([root], [Economic,...], Ø)</td>
</tr>
</tbody>
</table>

Shift

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>([root], [Economic,...], Ø)</td>
</tr>
</tbody>
</table>

Shift
Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>(root), [Economic, ...], ()</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>(root), [news, ...], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>(root), [had, ...], A2 = A1 ∪ {had, SBJ, news})</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>(root), [Economic, ...], ()</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>(root), [news, ...], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>(root), [had, ...], A2 = A1 ∪ {had, SBJ, news})</td>
</tr>
</tbody>
</table>
Economic news had **little effect** on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇒ ([root], [root], Economic), [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [had, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [had, had, little], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [little, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [little, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [on, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [on, . . . ], θ)</td>
</tr>
</tbody>
</table>

Economic news had **little effect** on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇒ ([root], [root], Economic), [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [had, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [had, had, little], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [little, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [little, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [on, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [on, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], . . . on), [financial, markets, . . . ], θ)</td>
</tr>
</tbody>
</table>

Economic news had **little effect** on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇒ ([root], [root], Economic), [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [news, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [Economic, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [had, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [had, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [had, had, little], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], news), [little, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [little, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, little), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [effect, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{SB} ⇒ ([root], [root], [effect, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], had, effect), [on, . . . ], θ)</td>
</tr>
<tr>
<td>LA_{ATT} ⇒ ([root], [root], [on, . . . ], θ)</td>
</tr>
<tr>
<td>SH ⇒ ([root], [root], . . . on), [financial, markets, . . . ], θ)</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root),</td>
</tr>
<tr>
<td>[Economic, , .], 0</td>
</tr>
<tr>
<td>SH ⇒ (root, Economic),</td>
</tr>
<tr>
<td>[news, , .], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root),</td>
</tr>
<tr>
<td>[news, , .], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>SH ⇒ (root, news),</td>
</tr>
<tr>
<td>[had, , .], A2 = A1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root),</td>
</tr>
<tr>
<td>[had, , .], A2 = A1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH ⇒ (root, had),</td>
</tr>
<tr>
<td>[little, , .], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, had, little),</td>
</tr>
<tr>
<td>[effect, , .], A2</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root, had),</td>
</tr>
<tr>
<td>[effect, , .], A3 = A2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH ⇒ (root, had, effect),</td>
</tr>
<tr>
<td>[on, , .], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, . . on),</td>
</tr>
<tr>
<td>[financial, markets, ], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, . . financial),</td>
</tr>
<tr>
<td>[markets, ], A3</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root, . . on),</td>
</tr>
<tr>
<td>[markets, ], A3</td>
</tr>
<tr>
<td>RA_VEC ⇒ (root, had, effect),</td>
</tr>
<tr>
<td>[on, ], A5 = A3 ∪ (on, PC, markets))</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

<table>
<thead>
<tr>
<th>Transition Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root),</td>
</tr>
<tr>
<td>[Economic, , .], 0</td>
</tr>
<tr>
<td>SH ⇒ (root, Economic),</td>
</tr>
<tr>
<td>[news, , .], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root),</td>
</tr>
<tr>
<td>[news, , .], A1 = {news, ATT, Economic})</td>
</tr>
<tr>
<td>SH ⇒ (root, news),</td>
</tr>
<tr>
<td>[had, , .], A2 = A1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root),</td>
</tr>
<tr>
<td>[had, , .], A2 = A1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH ⇒ (root, had),</td>
</tr>
<tr>
<td>[little, , .], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, had, little),</td>
</tr>
<tr>
<td>[effect, , .], A2</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root, had),</td>
</tr>
<tr>
<td>[effect, , .], A3 = A2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH ⇒ (root, had, effect),</td>
</tr>
<tr>
<td>[on, , .], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, . . on),</td>
</tr>
<tr>
<td>[financial, markets, ], A3</td>
</tr>
<tr>
<td>SH ⇒ (root, . . financial),</td>
</tr>
<tr>
<td>[markets, ], A3</td>
</tr>
<tr>
<td>LA_ATT ⇒ (root, . . on),</td>
</tr>
<tr>
<td>[markets, ], A3</td>
</tr>
<tr>
<td>RA_VEC ⇒ (root, had, effect),</td>
</tr>
<tr>
<td>[on, ], A5 = A3 ∪ (on, PC, markets))</td>
</tr>
<tr>
<td>RA_ATT ⇒ (root, had),</td>
</tr>
<tr>
<td>[effect, ], A6 = A5 ∪ (effect, ATT, on))</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>([root], [Economic, ...], θ)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root], [news, ...], A_1 = (news, ATT, Economic))</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>([root], [had, ...], A_2 = A_1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had], [little, ...], A_3)</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, little], [effect, ...], A_4)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root, had], [effect, ...], A_5 = A_2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, effect], [on, ...], A_6 = A_3 ∪ (on, PC, markets))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... on], [financial, markets, ], A_7 = A_5 ∪ (financial, ATT, on))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... financial], [markets, ], A_8 = A_6 ∪ (had, OBJ, effect))</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>([root], [Economic, ...], θ)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root], [news, ...], A_1 = (news, ATT, Economic))</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>([root], [had, ...], A_2 = A_1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had], [little, ...], A_3)</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, little], [effect, ...], A_4)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root, had], [effect, ...], A_5 = A_2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, effect], [on, ...], A_6 = A_3 ∪ (on, PC, markets))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... on], [financial, markets, ], A_7 = A_5 ∪ (financial, ATT, on))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... financial], [markets, ], A_8 = A_6 ∪ (had, OBJ, effect))</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>([root], [Economic, ...], θ)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root], [news, ...], A_1 = (news, ATT, Economic))</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>([root], [had, ...], A_2 = A_1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had], [little, ...], A_3)</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, little], [effect, ...], A_4)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root, had], [effect, ...], A_5 = A_2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, effect], [on, ...], A_6 = A_3 ∪ (on, PC, markets))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... on], [financial, markets, ], A_7 = A_5 ∪ (financial, ATT, on))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... financial], [markets, ], A_8 = A_6 ∪ (had, OBJ, effect))</td>
</tr>
</tbody>
</table>

Economic news had little effect on financial markets.

Transition Configuration

<table>
<thead>
<tr>
<th>Transition</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH</td>
<td>([root], [Economic, ...], θ)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root], [news, ...], A_1 = (news, ATT, Economic))</td>
</tr>
<tr>
<td>LA_SBJ</td>
<td>([root], [had, ...], A_2 = A_1 ∪ (had, SBJ, news))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had], [little, ...], A_3)</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, little], [effect, ...], A_4)</td>
</tr>
<tr>
<td>LA_ATT</td>
<td>([root, had], [effect, ...], A_5 = A_2 ∪ (effect, ATT, little))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, had, effect], [on, ...], A_6 = A_3 ∪ (on, PC, markets))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... on], [financial, markets, ], A_7 = A_5 ∪ (financial, ATT, on))</td>
</tr>
<tr>
<td>SH</td>
<td>([root, ... financial], [markets, ], A_8 = A_6 ∪ (had, OBJ, effect))</td>
</tr>
<tr>
<td>SH</td>
<td>([root], [had], [had], A_9 = A_7 ∪ (had, OBJ, effect))</td>
</tr>
<tr>
<td>RA_PRED</td>
<td>([root], [root], [had], A_9 = A_8 ∪ (root, PRED, had))</td>
</tr>
</tbody>
</table>
Economic news had little effect on financial markets.

### Transition Configuration

<table>
<thead>
<tr>
<th>Transition Configuration</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH ⇒ ([root, Economic], [news, . . .], )</td>
<td>A1 = {([news, ATT, Economic])}</td>
</tr>
<tr>
<td>LA_ATT ⇒ ([root], [news, . . .], )</td>
<td>A2 = A1 ∪ {((had, SBJ, news))}</td>
</tr>
<tr>
<td>SH ⇒ ([root, had], [had, . . .], A1)</td>
<td>A3 = A2 ∪ {((effect, ATT, little))}</td>
</tr>
<tr>
<td>LA_ATT ⇒ ([root, had], [effect, . . .], A2)</td>
<td>A4 = A3 ∪ {((markets, ATT, financial))}</td>
</tr>
<tr>
<td>SH ⇒ ([root, . . .], [markets, . . .], A3)</td>
<td>A5 = A4 ∪ {([on, PC, markets])}</td>
</tr>
<tr>
<td>LA_ATT ⇒ ([root, . . .], [on, . . .], A4)</td>
<td>A6 = A5 ∪ {((effect, ATT, on))}</td>
</tr>
<tr>
<td>RA_ATT ⇒ ([root, had], [had, . . .], A5)</td>
<td>A7 = A6 ∪ {((had, OBJ, effect))}</td>
</tr>
<tr>
<td>SH ⇒ ([root, had], [.], A6)</td>
<td>A8 = A7 ∪ {((had, PU, .))}</td>
</tr>
<tr>
<td>RA_ATT ⇒ ([root, had], [.], A7)</td>
<td>A9 = A8 ∪ {((root, PRED, had))}</td>
</tr>
<tr>
<td>RA_ATT ⇒ ([root, had], [.], A8)</td>
<td>SH ⇒ ([root], [ ], A9)</td>
</tr>
</tbody>
</table>

### Why grammar?

- **Surface string:** Mary saw John
- **Logical form:** saw(Mary, John)
- **Pred-arg structure:**
  - AGENT Mary
  - PATIENT John
- **Dependency graph:**
  - saw
  - Mary
  - John

---

**Transition-based parsing in practice**

Which action should the parser take under the current configuration?

We also need a parsing model that assigns a score to each possible action given a current configuration.

- Possible features of the current configuration:
  - Possible actions:
    - SHIFT, and for any relation r: LEFT-ARC_r, or RIGHT-ARC_r
  - Possible actions:
    - Possible features of the current configuration:
      - The top {1,2,3} words on the buffer and on the stack, their POS tags, distances between the words, etc.

We can learn this model from a dependency treebank.
Grammar formalisms

Formalisms provide a language in which linguistic theories can be expressed and implemented.

Formalisms define elementary objects (trees, strings, feature structures) and recursive operations which generate complex objects from simple objects.

Formalisms may impose constraints (e.g. on the kinds of dependencies they can capture).

How do grammar formalisms differ?

Formalisms define different representations:
- Tree-adjoining Grammar (TAG):
  - Fragments of phrase-structure trees
- Lexical-functional Grammar (LFG):
  - Annotated phrase-structure trees (c-structure) linked to feature structures (f-structure)
- Combinatory Categorial Grammar (CCG):
  - Syntactic categories paired with meaning representations
- Head-Driven Phrase Structure Grammar (HPSG):
  - Complex feature structures (Attribute-value matrices)

The dependencies so far:

Arguments:
- Verbs take arguments: subject, object, complements, ...
- Heads subcategorize for their arguments

Adjuncts/Modifiers:
- Adjectives modify nouns, adverbs modify VPs or adjectives, PPs modify NPs or VPs
- Modifiers subcategorize for the head

Typically, these are local dependencies: they can be expressed within individual CFG rules

Context-free grammars

CFGs capture only nested dependencies
- The dependency graph is a tree
- The dependencies do not cross
Beyond CFGs: Nonprojective dependencies

Dependencies form a tree with crossing branches

Non-projective dependencies

(Non-local) scrambling: In a sentence with multiple verbs, the argument of a verb appears in a different clause from that which contains the verb (arises in languages with freer word order than English)

Die Pizza hat Klaus versprochen zu bringen
The pizza has Klaus promised to bring
Klaus has promised to bring the pizza

Extraposition: Here, a modifier of the subject NP is moved to the end of the sentence

The guy is coming who is wearing a hat
Compare with the non-extraposed variant
The [guy [who is wearing a hat]] is coming

Topicalization: Here, the argument of the embedded verb is moved to the front of the sentence.

Cheeseburgers, I [thought [he likes]]
Non-local dependencies

Unbounded nonlocal dependencies

Wh-questions and relative clauses contain **unbounded nonlocal dependencies**, where the missing NP may be arbitrarily deeply embedded:

‘the sushi that [you told me [John saw [Mary eat]]]’

‘what [did you tell me [John saw [Mary eat]]]?’

Linguists call this phenomenon **wh-extraction** (wh-movement).
The trace analysis of *wh*-extraction

Because only one element can be extracted, we can use slash categories. This is still a CFG: the set of nonterminals is finite.

German: center embedding

...daß ich [Hans schwimmen] sah
...that I     Hans saw swim
...that I saw [Hans swim]

...daß ich [Maria [Hans schwimmen] helfen] sah
...that I Maria Hans swim help saw
...that I saw [Mary help [Hans swim]]

Dutch: cross-serial dependencies

...dat ik Hans zag zwemmen
...that I Hans saw swim
...that I saw [Hans swim]

...dat ik Maria Hans zag helpen zwemmen
...that I Maria Hans saw help swim
...that I saw [Mary help [Hans swim]]

...dat ik Anna Maria Hans zag laten helpen zwemmen
...that I Anna Maria Hans saw let help swim
...that I saw [Anna let [Mary help [Hans swim]]]

Such cross-serial dependencies require *mildly context-sensitive grammars*
Two mildly context-sensitive formalisms: TAG and CCG

Mildly context-sensitive grammars

Contain all context-free grammars/languages

Can be parsed in polynomial time (TAG/CCG: \(O(n^6)\))

(Strong generative capacity) capture certain kinds of dependencies: nested (like CFGs) and cross-serial (like the Dutch example), but not the MIX language:

MIX: the set of strings \(w \in \{a, b, c\}^*\) that contain equal numbers of \(a\), \(b\), and \(c\)

Have the constant growth property:
The length of strings grows in a linear way
The power-of-2 language \(\{a^{2^n}\}\) does not have the constant growth property.

The Chomsky Hierarchy

Recursively enumerable
Context-sensitive
Mildly context-sensitive
Context-free
Regular

TAG and CCG are lexicalized formalisms

The lexicon:
- pairs words with elementary objects
- specifies all language-specific information (e.g. subcategorization information)

The grammatical operations:
- are universal
- define (and impose constraints on) recursion.
A (C)CG derivation

\[
\begin{array}{c}
\text{John} & \text{eats} & \text{tapas} \\
\text{NP} & (S\backslash\text{NP})/\text{NP} & \text{NP} \\
\end{array}
\]

S\backslash\text{NP}

S

CCG categories are defined recursively:
- Categories are atomic (S, NP) or complex (S\backslash NP, (S\backslash NP)/NP)
- Complex categories (X/Y or X\backslash Y) are functions:
  X/Y combines with an adjacent argument to its right of category Y to return a result of category X.

Function categories can be composed, giving more expressive power than CFGs
More on CCG in one of our Semantics lectures!

(Lexicalized) Tree-Adjoining Grammar

TAG is a tree-rewriting formalism:
- TAG defines operations (substitution, adjunction) on trees.
- The elementary objects in TAG are trees (not strings)

TAG is lexicalized:
- Each elementary tree is anchored to a lexical item (word)
  “Extended domain of locality”:
  The elementary tree contains all arguments of the anchor.
  TAG requires a linguistic theory which specifies the shape of these elementary trees.

TAG is mildly context-sensitive:
- can capture Dutch cross-serial dependencies
  but is still efficiently parseable

We want to capture all arguments of a word in a single elementary object.

We also want to retain certain syntactic structures (e.g. VPs).

Our elementary objects are tree fragments:
TAG substitution (arguments)

\[
\begin{array}{c}
\alpha_1: \quad X \\
\alpha_2: \quad X \\
\alpha_3: \quad Y \\
\end{array}
\]

Derived tree:

\[
\begin{array}{c}
X \\
Y \\
\end{array}
\]

Substitute

\[
\begin{array}{c}
\alpha_1: \quad X \\
\alpha_2: \quad Y \\
\end{array}
\]

Derivation tree:

\[
\alpha_1 \quad \alpha_2 \quad \alpha_3
\]

TAG adjunction

Auxiliary tree

Foot node

\[
\begin{array}{c}
\alpha_1: \quad X \\
\end{array}
\]

\[
\begin{array}{c}
\beta_1: \quad X \\
\end{array}
\]

\[
\begin{array}{c}
\alpha_1 \\
\alpha_2 \quad \alpha_3 \\
\end{array}
\]

Derived tree:

\[
\begin{array}{c}
X \\
X^* \\
\end{array}
\]

ADJOIN

The effect of adjunction

TIG: sister adjunction

No adjunction: TSG (Tree substitution grammar)
TSG is context-free

Sister adjunction: TIG (Tree insertion grammar)
TIG is also context-free, but has a linguistically more adequate treatment of modifiers

Wrapping adjunction: TAG (Tree-adjoining grammar)
TAG is mildly context-sensitive

A small TAG lexicon

\[
\begin{array}{c}
\alpha_1: \quad S \\
\alpha_2: \quad NP \\
\alpha_3: \quad NP \\
\beta_1: \quad VP \\
\end{array}
\]

\[
\begin{array}{c}
NP \\
John \\
NP \\
always \\
\end{array}
\]

\[
\begin{array}{c}
VP* \\
eats \\
\end{array}
\]

\[
\begin{array}{c}
NP \\
VBZ \\
NP \\
\end{array}
\]
A TAG derivation

$\alpha_1$:  
$\alpha_2$:  
$\alpha_3$:  

NP  
John  
always  
tapas  

NP  
NP  
NP  
NP  

S  
NP  
VP  
NP  
VP  
VP  
VP*  

NP  
NP  
NP  
NP  

NP  
NP  
NP  
NP  

$\beta_1$:  

always  
NP  
NP  
NP  
NP  

NP  
NP  
NP  
NP  

NP  
NP  
NP  
NP  

$a^n b^n$: Cross-serial dependencies

Elementary trees:

Deriving $aabb$
Feature Structure Grammars

Simple grammars overgenerate

\[
S \rightarrow NP \ VP \\
VP \rightarrow Verb \ NP \\
NP \rightarrow Det \ Noun \\
Det \rightarrow the | a | these \\
Verb \rightarrow eat | eats \\
Noun \rightarrow cake | cakes | student | students
\]

This generates ungrammatical sentences like “these student eats a cakes”

We need to capture (number/person) agreement

Refining the nonterminals

\[
S \rightarrow NP_{sg} \ VP_{sg} \\
S \rightarrow NP_{pl} \ VP_{pl} \\
VP_{sg} \rightarrow Verb_{sg} \ NP \\
VP_{pl} \rightarrow Verb_{pl} \ NP \\
NP_{sg} \rightarrow Det_{sg} \ Noun_{sg} \\
Det_{sg} \rightarrow the | a \\
\]

This yields very large grammars.

What about person, case, ...?

Difficult to capture generalizations.

Subject and verb have to have number agreement

\(NP_{sg}, NP_{pl} \) and \(NP\) are three distinct nonterminals

Feature structures

Replace atomic categories with feature structures:

\[
\begin{bmatrix}
CAT & NP \\
NUM & SG \\
PERS & 3 \\
CASE & NOM
\end{bmatrix}
\]

A feature structure is a list of features (= attributes), e.g. CASE, and values (eg NOM).

We often represent feature structures as attribute value matrices (AVM)

Usually, values are typed (to avoid CASE:SG)
Feature structures as directed graphs

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
= \begin{array}{c}
\text{NP} \\
\text{CAT} \\
\text{PERS} \\
\text{NUM}
\end{array}
\]

Complex feature structures

We distinguish between atomic and complex feature values.
A complex value is a feature structure itself.

This allows us to capture better generalizations.

Only atomic values:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\]

Complex values:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{AGR} & \text{CASE} \\
\text{NOM} & \text{NOM}
\end{bmatrix}
\]

Feature paths

A feature path allows us to identify particular values in a feature structure:

\[\langle \text{NP CAT} \rangle = \text{NP}\]
\[\langle \text{NP AGR CASE} \rangle = \text{NOM}\]

Unification

Two feature structures A and B unify \((A \sqcup B)\) if they can be merged into one consistent feature structure C:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\sqcup
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG}
\end{bmatrix} =
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG}
\end{bmatrix}
\]

Otherwise, unification fails:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\sqcup
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{PL}
\end{bmatrix} = \emptyset
\]
PATR-II style feature structures

CFG rules are augmented with constraints:

\[ A_0 \rightarrow A_1 \ldots A_n \]

\{set of constraints\}

There are two kinds of constraints:

**Unification constraints:**

\[ \langle A_i \text{ feature-path} \rangle = \langle A_j \text{ feature-path} \rangle \]

**Value constraints:**

\[ \langle A_i \text{ feature-path} \rangle = \text{atomic value} \]

A grammar with feature structures

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S \rightarrow \text{NP VP})</td>
<td>(\langle \text{NP } \text{NUM} \rangle = \langle \text{VP } \text{NUM} \rangle)</td>
</tr>
<tr>
<td>(\text{NP} \rightarrow \text{DT NOUN})</td>
<td>(\langle \text{NP } \text{NUM} \rangle = \langle \text{NOUN } \text{NUM} \rangle)</td>
</tr>
<tr>
<td></td>
<td>(\langle \text{NP } \text{CASE} \rangle = \langle \text{NOUN } \text{CASE} \rangle)</td>
</tr>
<tr>
<td>(\text{NOUN} \rightarrow \text{cake})</td>
<td>(\langle \text{NOUN } \text{NUM} \rangle = \text{sg})</td>
</tr>
</tbody>
</table>

With complex feature structures

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S \rightarrow \text{NP VP})</td>
<td>(\langle \text{NP } \text{AGR} \rangle = \langle \text{VP } \text{AGR} \rangle)</td>
</tr>
<tr>
<td>(\text{NP} \rightarrow \text{DT NOUN})</td>
<td>(\langle \text{NP } \text{AGR} \rangle = \langle \text{NOUN } \text{AGR} \rangle)</td>
</tr>
<tr>
<td>(\text{NOUN} \rightarrow \text{cake})</td>
<td>(\langle \text{NOUN } \text{AGR NUM} \rangle = \text{sg})</td>
</tr>
</tbody>
</table>

Attribute-Value Grammars and CFGs

If every feature can only have a finite set of values, any attribute-value grammar can be compiled out into a (possibly huge) context-free grammar.

Complex feature structures capture better generalizations (and hence require fewer constraints) — cf. the previous slide.
Going beyond CFGs

The power-of-2 language: \( L_2 = \{w \mid i \text{ is a power of } 2 \} \)

\( L_2 \) is a (fully) context-sensitive language.

(Mildly context-sensitive languages have the constant growth property (the length of words always increases by a constant factor \( c \)).

Here is a feature grammar which generates \( L_2 \):

\[
\begin{align*}
A & \rightarrow a \\
\langle A \ F \rangle & = 1 \\
A & \rightarrow A_1 \ A_2 \\
\langle A \ F \rangle & = \langle A_1 \rangle \\
\langle A \ F \rangle & = \langle A_2 \rangle
\end{align*}
\]

Today’s key concepts

Transition-based dependency parsing
for projective dependency trees

Going beyond projective dependencies:
- non-projective dependencies
- non-local dependencies

Expressive Grammars
- TAG
- CCG
- Feature-Structure Grammars