Dependency Grammars and Parsing

CMSC 473/673
UMBC
November 20th, 2017
Course Announcement 1: Assignment 3

Due Wednesday (< 2 days)

Any questions?
Course Announcement 2: Graduate Paper 2

Due Monday 11/27 (~7 days)

Any questions?
Recap from last time...
PCFG Likelihood

\[ p(w_1 \ w_2 \ w_3 \ ... \ w_N) \]

likelihood of word sequence \( w_1w_2...w_N \)

\[ p(S \rightarrow^+ w_1 \ w_2 \ w_3 \ ... \ w_N) \]

likelihood of word sequence \( w_1w_2...w_N \)
based on starting at \( S \)

\[ p(\ ) \ + \ p(\ ) \ + \ p(\ ) \ + \ + \ \cdots \]
PCFG Likelihood

\[ p(S \rightarrow^+ w_1 w_2 w_3 \ldots w_N) \]
PCFGs: Inside Algorithm

\[ \beta(X, s, t) = \sum_{Y,Z} \sum_{k:s<k<t} \beta(Y, s, k) \times \beta(Z, k, t) \times p(X \rightarrow Y Z) \]

1. That start from non-terminal X, with left index s
2. That terminate after the t\textsuperscript{th} word
3. That emit the observations from s (inclusive) to t (exclusive)
\[ \beta = \text{WeightedCell}[K][N][N+1] \]

for \( j = 1; j \leq N; ++j \) {
    \[ \beta[X][j-1][j] += [ p(X \rightarrow \text{word}_j) \text{ for non-terminal } X \text{ in } G \text{ if } X \rightarrow \text{word}_j ] \]
}

for \( \text{width} = 2; \text{width} \leq N; ++\text{width} \) {
    for \( \text{start} = 0; \text{start} < N - \text{width}; ++\text{start} \) {
        \[ \text{end} = \text{start} + \text{width} \]
        for \( \text{mid} = \text{start}+1; \text{mid} < \text{end}; ++\text{mid} \) {
            for rule \( X \rightarrow Y Z : G \) {
                \[ \beta[X][\text{start}][\text{end}] += \]
                \[ [ p(X \rightarrow Y Z) \times \beta[Y][\text{start}][\text{mid}] \times \beta[Z][\text{mid}][\text{end}] \]
                \[ \text{for rule } X \rightarrow Y Z : G \]
            }
        }
    }
}
}
# CKY Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Weights</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recognizer</strong></td>
<td>Boolean (True/False)</td>
<td>or</td>
<td>and</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td><strong>Viterbi</strong></td>
<td>[0,1]</td>
<td>max</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Inside</strong></td>
<td>[0,1]</td>
<td>+</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Adapted from Jason Eisner
PCFG Outside Score

\[ p(w_1 \, w_2 \, \ldots \, w_{l-1} \, A \, w_{m+1} \, \ldots \, w_N) \]

likelihood of non-terminal \( A \) from \( l \) to \( m \),
and surrounding words

\[ p(S \xrightarrow{+} w_1 \, w_2 \, \ldots \, w_{l-1} \, A \, w_{m+1} \, \ldots \, w_N) \]
PCFG Outside Score

\[ p(S \xrightarrow{+} w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]

Option 1: \( B \rightarrow C A \)
PCFG Outside Score

\[ p(S \rightarrow^+ w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]

Option 1: B \rightarrow C A

Option 2: B \rightarrow A C

must have already considered these words
PCFGs: Outside Algorithm

\[ \alpha(X, s, t) = \]

Option 1: \( B \rightarrow C A \)

\[ \sum_{Y,Z} \sum_{k: 0 \leq k < s} \beta(Z, k, s) \ast \alpha(Y, k, t) \ast p(Y \rightarrow Z X) + \]

Option 2: \( B \rightarrow A C \)

\[ \sum_{Y,Z} \sum_{k: t \leq k \leq N} \beta(Z, t, k) \ast \alpha(Y, s, k) \ast p(Y \rightarrow X Z) \]

\[ \alpha(X, s, t) \text{ is the total probability of all derivations:} \]
1. that include non-terminal \( X \) (but not words) from \( s \) to \( t \)
2. and all (& only) the observed words before \( s \) and after \( t \)
\[ \alpha = \text{WeightedCell}[K][N][N+1] \]

for(\text{width} = N; \text{width} \geq 1; --\text{width}) {
    for(\text{start} = 0; \text{start} \leq N - \text{width}; ++\text{start}) {
        \text{end} = \text{start} + \text{width}
        for(\text{mid} = \text{start}+1; \text{mid} < \text{end}; ++\text{mid}) {
            \alpha[Y][\text{start}][\text{mid}] +=
            \left[ \text{p}(X \rightarrow Y Z) \times \alpha[Y][\text{start}][\text{end}] \times \beta[Z][\text{mid}][\text{end}] \right]
            for rule X \rightarrow Y Z : G

            \alpha[Z][\text{mid}][\text{end}] +=
            \left[ \text{p}(X \rightarrow Y Z) \times \alpha[Z][\text{start}][\text{end}] \times \beta[Y][\text{mid}][\text{end}] \right]
            for rule X \rightarrow Y Z : G
        }
    }
}

Outside Algorithm
## CKY Algorithms

<table>
<thead>
<tr>
<th>Recognizer</th>
<th>Weights</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>$\mathbf{0}$</th>
<th>$\mathbf{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizer</td>
<td>Boolean (True/False)</td>
<td>or</td>
<td>and</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Viterbi</td>
<td>[0,1]</td>
<td>max</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Inside</td>
<td>[0,1]</td>
<td>+</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Adapted from Jason Eisner
Inside-Outside Algorithm

1. Run inside algorithm (bottom-up, small-to-large)

2. Run outside algorithm (top-down, large-to-small)

Q: Why?

A: Compute Posterior Probabilities (Expectations)
Getting Expected Counts

\[ \alpha[VP, 1, 4] = p(\text{Papa VP with a spoon}) \]

\[ \beta[VP, 1, 4] = p(\text{ate the caviar} \mid VP) \]

\[ \alpha[VP, 1, 4] \ast \beta[VP, 1, 4] \begin{array}{c} \text{\[\beta[S, 0, 7]\]} \\ \text{\[p(\text{VP}_{1\rightarrow4} \mid \text{Papa ... spoon})\]} \end{array} = \]
Getting Expected Counts

\[ \alpha[VP, 1, 4] = p(\text{Papa VP with a spoon}) \]

\[ \beta[VP, 1, 2] = p(\text{ate} | V) \]
\[ \beta[NP, 2, 4] = p(\text{the caviar} | NP) \]
\[ p(VP \rightarrow V NP) \]

\[ \frac{\alpha[VP, 1, 4] \times \beta[VP, 1, 2] \times \beta[NP, 2, 4] \times P(VP \rightarrow V NP)}{\beta[S, 0, 7]} = p(VP_{1 \rightarrow 4} \rightarrow V_1 \text{ NP}_{2 \rightarrow 4} | \text{Papa ... spoon}) \]
Expected Counts

$$\mathbb{E}[X \rightarrow a \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow a)}{L(w_1 w_2 \cdots w_N)} \sum_{0 \leq i < N: w_i = a} \alpha(X, i, i + 1)$$

$$\mathbb{E}[X \rightarrow Y Z \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow Y Z)}{L(w_1 w_2 \cdots w_N)} \sum_{0 \leq i < k < j \leq N} \alpha(X, i, j) \beta(Y, i, k) \beta(Z, k, j)$$
Expectation Maximization (EM)

0. Assume *some* value for your parameters \( p(X \rightarrow Y Z) \)

Two step, iterative algorithm

1. E-step: count under uncertainty, assuming these parameters

\[
\mathbb{E}[X \rightarrow a \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow a)}{L(w_1 w_2 \cdots w_N)} \sum_{0 \leq i < N : w_i = a} \alpha(x, i, i + 1)
\]

\[
\mathbb{E}[X \rightarrow Y Z \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow Y Z)}{L(w_1 w_2 \cdots w_N)} \sum_{0 \leq i < k < j \leq N} \alpha(X, i, j) \beta(Y, i, k) \beta(Z, k, j)
\]

\[p(z_i) \rightarrow \text{count}(z_i, w_i)\]

2. M-step: maximize log-likelihood, assuming these uncertain counts

\[p^{(t)}(z) \leftrightarrow \text{estimated counts} \]

\[p^{(t+1)}(z)\]

“Inside-outside”
Probabilistic Context Free Grammar (PCFG) Tasks

Find the most likely parse (for an observed sequence)

Calculate the (log) likelihood of an observed sequence $w_1, ..., w_N$

Learn the grammar parameters
Structure vs. Word Relations

Constituency trees/analyses: based on structure

Dependency analyses: based on word relations
Remember: (P)CFGs Help Clearly Show Ambiguity

I ate the meal with friends
I ate the meal with friends.
I ate the meal with friends.
I ate the meal with friends.
I ate the meal with friends.
Labeled Dependencies

Word-to-word labeled relations

governor (head)

dependent

Chris ate

nsubj

Chris

ate
Labeled Dependencies

Word-to-word labeled relations

governor (head)
dependent

<table>
<thead>
<tr>
<th>Core dependents of clausal predicates</th>
<th>Nominal dep</th>
<th>Predicate dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>csubj</td>
<td></td>
</tr>
<tr>
<td>nsubjpass</td>
<td>esubjpass</td>
<td></td>
</tr>
<tr>
<td>dobj</td>
<td>ecomp</td>
<td>xcomp</td>
</tr>
<tr>
<td>iobj</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-core dependents of clausal predicates</th>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Modifier word</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>csubj</td>
<td></td>
<td>amod</td>
</tr>
<tr>
<td>appos</td>
<td>nfincl</td>
<td>det</td>
<td></td>
</tr>
<tr>
<td>nmod</td>
<td>ncmod</td>
<td>neg</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special clausal dependents</th>
<th>Nominal dep</th>
<th>Auxiliary</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocative</td>
<td>aux</td>
<td>mark</td>
<td></td>
</tr>
<tr>
<td>discourse</td>
<td>auxpass</td>
<td>punct</td>
<td></td>
</tr>
<tr>
<td>expl</td>
<td>cop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Coordination | | |
|--------------|---------------||
| conj         | cc             | |

<table>
<thead>
<tr>
<th>Noun dependents</th>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Modifier word</th>
</tr>
</thead>
<tbody>
<tr>
<td>compound</td>
<td>mwe</td>
<td>goeswith</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>foreign</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compounding and unanalyzed</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>compound</td>
<td>mwe</td>
</tr>
<tr>
<td>name</td>
<td>foreign</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case-marking, prepositions, possessive case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loose joining relations</td>
</tr>
<tr>
<td>list</td>
</tr>
<tr>
<td>dislocated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence head</td>
<td>Unspecified dependency</td>
</tr>
<tr>
<td>root</td>
<td>dep</td>
</tr>
</tbody>
</table>

de Marneffe et al., 2014
http://universaldependencies.org/
(Labeled) Dependency Parse

Directed graphs

- Vertices: linguistic blobs in a sentence
- Edges: (labeled) arcs
(Labeled) Dependency Parse

Directed graphs

Vertices: linguistic blobs in a sentence
Edges: (labeled) arcs

Often directed trees
1. A single root node with no incoming arcs
2. Each vertex except root has exactly one incoming arc
3. Unique path from the root node to each vertex
Dependency Parsing as a (Renewed) Core NLP Problem

Parser

Grammar

Evaluation

Gold (correct) reference trees

Other NLP task (entity coref., MT, Q&A, ...)

score

sentence 1

sentence 2

sentence 3

sentence 4

independent operations
Projective Dependency Trees

No crossing arcs

United canceled the morning flights to Houston

✔ Projective
Projective Dependency Trees

No crossing arcs

✔ Projective

✗ Not projective
Projective Dependency Trees

No crossing arcs

✔ Projective

Not projective

- non projective parses capture
  - certain long-range dependencies
  - free word order

United canceled the morning flights to Houston

JetBlue canceled our flight this morning which was already late

SLP3: Figs 14.2, 14.3
Are CFGs for Naught?
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

SLP3: 11.4.3
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.

2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Dependency Post-Processing
(Keep Tree Structure)
Amaranthus, collectively known as amaranth, is a cosmopolitan genus of annual or short-lived perennial plants.
(Some) Dependency Parsing Algorithms

Dynamic Programming
  Eisner Algorithm (Eisner 1996)

Transition-based
  Shift-reduce, arc standard

Graph-based
  Maximum spanning tree
(Some) Dependency Parsing Algorithms

Dynamic Programming
Eisner Algorithm (Eisner 1996)

Transition-based
Shift-reduce, arc standard

Graph-based
Maximum spanning tree
Shift-Reduce Parsing

Recall from CMSC 331
Shift-Reduce Parsing

Recall from CMSC 331

Bottom-up

Tools: input words, some special root symbol ($), and a stack to hold configurations
Shift-Reduce Parsing

Recall from CMSC 331
Bottom-up

Tools: input words, some special root symbol ($) , and a stack to hold configurations

Shift:
- move tokens onto the stack
- match top two elements of the stack against the grammar (RHS)

Reduce:
- If match occurred, place LHS symbol on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:

– move tokens onto the stack
– decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:

– If there’s a valid relation, place head on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:
- move tokens onto the stack
- decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:
- If there’s a valid relation, place head on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

- **Shift:**
  - move tokens onto the stack
  - decide if top two elements of the stack form a valid (good) grammatical dependency

- **Reduce:**
  - If there’s a valid relation, place head on the stack

**decide how?**
Search problem!

**what is valid?**
Learn it!
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:

- move tokens onto the stack
- decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:

- If there’s a valid relation, place head on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:
- move tokens onto the stack
- decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:
- If there’s a valid relation, place head on the stack
## Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
</tr>
</tbody>
</table>
# Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td><strong>LEFTARC</strong></td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td><strong>RIGHTARC</strong></td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td><strong>SHIFT</strong></td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
# Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td><strong>LEFTARC</strong></td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td><strong>RIGHTARC</strong></td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td><strong>SHIFT</strong></td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
Arc Standard Parsing

\[ \text{state} \leftarrow \{ \text{[root]}, \text{[words]}, [] \} \]
Arc Standard Parsing

\[
\text{state} \leftarrow \{[\text{root}], [\text{words}], [] \}
\]

\[
\text{while state not final} \{ \\
\}
\]
Arc Standard Parsing

state $\leftarrow \{[\text{root}], [\text{words}], [] \}$

while state not final {
    t $\leftarrow$ ORACLE(state)
    state $\leftarrow$ APPLY(t, state)
}
Arc Standard Parsing

\[
\text{state} \leftarrow \{[\text{root}], [\text{words}], [] \}
\]

while \text{state} \text{ not final} {
    \[
    t \leftarrow \text{ORACLE(\text{state})}
    \]
    \[
    \text{state} \leftarrow \text{APPLY}(t, \text{state})
    \]
}\n
\[
\text{return state}
\]
Arc Standard Parsing

state ← { [root], [words], [] }

while state ≠ { [root], [], [(deps)] } {
    t ← ORACLE(state)
    state ← APPLY(t, state)
}

return state
Arc Standard Parsing

state ← \{[\text{root}], [\text{words}], [] \}

while state \neq \{[\text{root}], [], [(\text{deps})]\} \{
    t ← \text{ORACLE}(\text{state})
    \text{state} ← \text{APPLY}(t, \text{state})
}\}

return state

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
Arc Standard Parsing

\\texttt{state} \leftarrow \{[\text{root}], [\text{words}], []\}

\textbf{while} \texttt{state} \neq \{[\text{root}], [], [(\text{deps})]\} \{ \\
\texttt{t} \leftarrow \text{ORACLE(} \texttt{state} \text{)} \\
\texttt{state} \leftarrow \text{APPLY(} \texttt{t, state} \text{)}
\}

\textbf{return} \texttt{state}

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>\textbf{LEFTARC}</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>\textbf{RIGHTARC}</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>\textbf{SHIFT}</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
Arc Standard Parsing

\[\text{state} \leftarrow \{[\text{root}], [\text{words}], [] \} \]

while \( \text{state} \neq \{[\text{root}], [], [(\text{deps})]\} \) {
\[\text{t} \leftarrow \text{PREDICT}(\text{state})\]
\[\text{state} \leftarrow \text{APPLY}(\text{t}, \text{state})\]
}

return \(\text{state}\)

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>

what is valid? Learn it!
Arc Standard Parsing

Q: What is the time complexity?

while state ≠ \{[root], [], [(deps)]\} { 
    t ← ORACLE(state)
    state ← APPLY(t, state)
}

return state
Arc Standard Parsing

Q: What is the time complexity?

A: Linear

\[
\text{state} \leftarrow \text{APPLY}(t, \text{state})
\]

return \text{state}
Arc Standard Parsing

Q: What is the time complexity?
A: Linear

Q: What’s potentially problematic?
Arc Standard Parsing

Q: What is the time complexity?

A: Linear

Q: What’s potentially problematic?

A: This is a greedy algorithm
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:
- move tokens onto the stack
- decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:
- If there’s a valid relation, place head on the stack

What is valid? Learn it!

What are the possible actions? Search problem!

decide how?
Learning An Oracle (Predictor)

Training data: dependency treebank
Input: configuration
Output: \{LEFTARC, RIGHTARC, SHIFT\}

\[ t \leftarrow \text{ORACLE(state)} \]
Learning An Oracle (Predictor)

Training data: dependency treebank
Input: *configuration*
Output: \{**LEFTARC**, **RIGHTARC**, **SHIFT**\}

\[
t \leftarrow \text{ORACLE(state)}
\]

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
Learning An Oracle (Predictor)

Training data: dependency treebank

Input: configuration

Output: \{LEFTARC, RIGHTARC, SHIFT\}

\[ t \leftarrow \text{ORACLE} (\text{state}) \]

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
- Choose RIGHTARC if
  - it produces a correct head-dependent relation given the reference parse and
  - all of the dependents of the word at the top of the stack have already been assigned
Learning An Oracle (Predictor)

Training data: dependency treebank
Input: configuration
Output: \{LEFTARC, RIGHTARC, SHIFT\}

\[ t \leftarrow \text{ORACLE(state)} \]

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
- Choose RIGHTARC if
  - it produces a correct head-dependent relation given the reference parse and
  - all of the dependents of the word at the top of the stack have already been assigned
- Otherwise, choose SHIFT