Natural Language Processing
(A brief look at CKY, and then)
Dependency parsing

Sofia Serrano
sofias6@cs.washington.edu

Credit to Yulia Tsvetkov for slides
A3 is out on gitlab!
The CKY algorithm for parsing with PCFGs
Parsing

- **Parsing is search** through the space of all possible parses
  - e.g., we may want either any parse, all parses or the highest scoring parse (if PCFG):

  \[
  \arg \max_{T \in G(x)} P(T)
  \]

- **Bottom-up:**
  - One starts from words and attempt to construct the full tree

- **Top-down**
  - Start from the start symbol and attempt to expand to get the sentence
CKY algorithm (aka CYK)

- Cocke-Kasami-Younger algorithm
  - Independently discovered in late 60s / early 70s

- An efficient bottom up parsing algorithm for (P)CFGs
  - can be used both for the recognition and parsing problems
  - Very important in NLP (and beyond)

- We will start with the non-probabilistic version
Constraints on the grammar

● The basic CKY algorithm supports only rules in the **Chomsky Normal Form (CNF):**

\[
C \rightarrow x \\
C \rightarrow C_1 C_2
\]

Unary *preterminal* rules (generation of words given PoS tags)

\[
N \rightarrow \text{telescope} \\
D \rightarrow \text{the}
\]

Binary *inner* rules

\[
S \rightarrow NP VF \\
NP \rightarrow D \ N
\]
Constraints on the grammar

- The basic CKY algorithm supports only rules in the Chomsky Normal Form (CNF):
  
  \[ C \rightarrow x \]
  
  \[ C \rightarrow C_1 C_2 \]

- Any CFG can be converted to an equivalent CNF
  - Equivalent means that they define the same language
  - However (syntactic) trees will look differently
  - It is possible to address it by defining such transformations that allows for easy reverse transformation
Transformation to CNF form

- What one need to do to convert to CNF form
  - Get rid of rules that mix terminals and non-terminals
  - Get rid of unary rules: $C \rightarrow C_1$
  - Get rid of N-ary rules: $C \rightarrow C_1 C_2 \ldots C_n \ (n > 2)$

Crucial to process them, as required for efficient parsing.
Transformation to CNF form: binarization

- Consider

\[ NP \rightarrow DT \quad NNP \quad VBG \quad NN \]

- How do we get a set of binary rules which are equivalent?
Transformation to CNF form: binarization

- Consider

\[ NP \rightarrow DT \ NNP \ VBG \ NN \]

\[ NP \]

- How do we get a set of binary rules which are equivalent?

\[ NP \rightarrow DT \ X \]
\[ X \rightarrow NNP \ Y \]
\[ Y \rightarrow VBG \ NN \]
Transformation to CNF form: binarization

- Consider

  \[ NP \rightarrow DT\ NNP\ VBG\ NN \]

  ![Diagram](image)

  - How do we get a set of binary rules which are equivalent?

  \[
  NP \rightarrow DT\ X \\
  X \rightarrow NNP\ Y \\
  Y \rightarrow VBG\ NN
  \]

- A more systematic way to refer to new non-terminals

  \[
  NP \rightarrow DT\ @NP\ DT \\
  @NP\ DT \rightarrow NNP\ @NP\ DT\_NNP \\
  @NP\ DT\_NNP \rightarrow VBG\ NN
  \]
Transformation to CNF form: binarization

- Instead of binarizing tuples we can binarize trees on preprocessing:

Also known as lossless Markovization in the context of PCFGs

Can be easily reversed on postprocessing
CKY: Parsing task

- We are given
  - a grammar \( <N, T, S, R> \)
  - a sequence of words \( w = (w_1, w_2, \ldots, w_n) \)

- Our goal is to produce a parse tree for \( w \)
CKY: Parsing task

- We are given
  - a grammar \(<N, T, S, R>\)
  - a sequence of words \(w = (w_1, w_2, \ldots, w_n)\)
- Our goal is to produce a parse tree for \(w\)
- We need an easy way to refer to substrings of \(w\)

\[\text{span (i, j) refers to words between fenceposts i and j}\]
Parsing one word

\[ C \rightarrow w_i \]
Parsing one word

\[ C \rightarrow w_i \]
Parsing one word

\[ C \rightarrow w_i \]

covers all words between \( i - 1 \) and \( i \)
Parsing longer spans

$C \rightarrow C_1 \ C_2$

Check through all $C_1$, $C_2$, mid

- $C_1$: covers all words btw $min$ and $mid$
- $C_2$: covers all words btw $mid$ and $max$
Parsing longer spans

\[ C \rightarrow C_1 \quad C_2 \]

Check through all C1, C2, mid

covers all words btw min and mid

covers all words btw mid and max
Parsing longer spans

covers all words between min and max
Preterminal rules

<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

max = 1  max = 2  max = 3

min = 0

min = 1

min = 2

Chart (aka parsing triangle)

Inner rules

\[
S \rightarrow NP \ VP
\]

\[
VP \rightarrow M \ V
\]

\[
VP \rightarrow V
\]

\[
NP \rightarrow N
\]

\[
NP \rightarrow N\ NP
\]

\[
N \rightarrow can
\]

\[
N \rightarrow lead
\]

\[
N \rightarrow poison
\]

\[
M \rightarrow can
\]

\[
M \rightarrow must
\]

\[
V \rightarrow poison
\]

\[
V \rightarrow lead
\]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Preterminal rules

Outer rules

\[ S \rightarrow NP \ VP \]

\[ VP \rightarrow M \ V \]
\[ VP \rightarrow V \]

\[ NP \rightarrow N \]
\[ NP \rightarrow N \ NP \]

Inner rules

\[ N \rightarrow can \]
\[ N \rightarrow lead \]
\[ N \rightarrow poison \]

\[ M \rightarrow can \]
\[ M \rightarrow must \]

\[ V \rightarrow poison \]
\[ V \rightarrow lead \]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

![Diagram](image)

**Preterminal rules**

\[ S \rightarrow NP \ VP \]

**Inner rules**

\[ VP \rightarrow M \ V \]

\[ NP \rightarrow N \]

\[ NP \rightarrow N \ NP \]

\[ N \rightarrow can \]

\[ N \rightarrow lead \]

\[ N \rightarrow poison \]

\[ M \rightarrow can \]

\[ M \rightarrow must \]

\[ V \rightarrow poison \]

\[ V \rightarrow lead \]
\[ S \rightarrow NP\ VP \]
\[ VP \rightarrow M\ V \]
\[ NP \rightarrow N \]
\[ NP \rightarrow N\ NP \]
\[ N \rightarrow can \]
\[ N \rightarrow lead \]
\[ N \rightarrow poison \]
\[ M \rightarrow can \]
\[ M \rightarrow must \]
\[ V \rightarrow poison \]
\[ V \rightarrow lead \]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

max = 1  max = 2  max = 3

min = 0

min = 1

min = 2

\[ S \rightarrow NP \ VP \]

\[ VP \rightarrow M \ V \]

\[ VP \rightarrow V \]

\[ NP \rightarrow N \]

\[ NP \rightarrow N \ NP \]

\[ N \rightarrow can \]

\[ N \rightarrow lead \]

\[ N \rightarrow poison \]

\[ M \rightarrow can \]

\[ M \rightarrow must \]

\[ V \rightarrow poison \]

\[ V \rightarrow lead \]
Preterminal rules

<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
\text{max} = 1 & \text{max} = 2 & \text{max} = 3 \\
\text{min} = 0 & 1 & 4 & 6 \\
\text{min} = 1 & 2 & 5 & 3 \\
\text{min} = 2
\end{array}
\]

Inner rules

\[
S \rightarrow NP \ VP
\]

\[
VP \rightarrow M \ V
\]

\[
VP \rightarrow V
\]

\[
NP \rightarrow N
\]

\[
NP \rightarrow N \ NP
\]

\[
N \rightarrow \text{can}
\]

\[
N \rightarrow \text{lead}
\]

\[
N \rightarrow \text{poison}
\]

\[
M \rightarrow \text{can}
\]

\[
M \rightarrow \text{must}
\]

\[
V \rightarrow \text{poison}
\]

\[
V \rightarrow \text{lead}
\]
Preterminal rules

A → poison
M → must
N → can
S → N NP
VP → M V
V → lead
NP → N

Inner rules

V → poison
M → must
N → can
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Preterminal rules

\[
S \rightarrow NP \ VP
\]

\[
VP \rightarrow M \ V
\]

\[
VP \rightarrow V
\]

\[
NP \rightarrow N
\]

\[
NP \rightarrow N \ NP
\]

Inner rules

\[
N \rightarrow can
\]

\[
N \rightarrow lead
\]

\[
N \rightarrow poison
\]

\[
M \rightarrow can
\]

\[
M \rightarrow must
\]

\[
V \rightarrow poison
\]

\[
V \rightarrow lead
\]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|c|}
\hline
\text{max} = 1 & \text{max} = 2 & \text{max} = 3 \\
\hline
\text{min} = 0 & ? & \text{?} \\
\hline
\text{min} = 1 & 2 & ? \\
\hline
\text{min} = 2 & 3 & ? \\
\hline
\end{array}
\]

\[
S \rightarrow NP \ VP
\]

\[
VP \rightarrow M \ V \\
VP \rightarrow V
\]

\[
NP \rightarrow N \\
NP \rightarrow N \ NP
\]

\[
N \rightarrow \text{can} \\
N \rightarrow \text{lead} \\
N \rightarrow \text{poison}
\]

\[
M \rightarrow \text{can} \\
M \rightarrow \text{must}
\]

\[
V \rightarrow \text{poison} \\
V \rightarrow \text{lead}
\]
$$S \rightarrow NP \ VP$$

$$VP \rightarrow M \ V$$
$$VP \rightarrow V$$

$$NP \rightarrow N$$
$$NP \rightarrow N \ NP$$

$$N \rightarrow can$$
$$N \rightarrow lead$$
$$N \rightarrow poison$$

$$M \rightarrow can$$
$$M \rightarrow must$$

$$V \rightarrow poison$$
$$V \rightarrow lead$$
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c}
\text{min} & 0 & 1 & 2 \\
1 & N, V & NP, VP & \\
2 & N, M & NP & \\
3 & N, V & NP, VP & \\
4 & ? & \\
\end{array}
\]

**Preterminal rules**

\[
S \rightarrow NP \ VP
\]

**Inner rules**

\[
VP \rightarrow M \ V \\
VP \rightarrow V
\]

\[
NP \rightarrow N \\
NP \rightarrow N NP
\]

\[
N \rightarrow can \\
N \rightarrow lead \\
N \rightarrow poison
\]

\[
M \rightarrow can \\
M \rightarrow must
\]

\[
V \rightarrow poison \\
V \rightarrow lead
\]
Preterminal rules

<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
S & \Rightarrow NP \ VP \\
VP & \Rightarrow M \ V \\
NP & \Rightarrow N \\
NP & \Rightarrow N \ NP \\
N & \Rightarrow can \\
N & \Rightarrow lead \\
N & \Rightarrow poison \\
M & \Rightarrow can \\
M & \Rightarrow must \\
V & \Rightarrow poison \\
V & \Rightarrow lead
\end{align*}
\]

Inner rules

\[
\begin{align*}
\text{max = 1} & & \text{max = 2} & & \text{max = 3} \\
\text{min = 0} \\
\text{min = 1} \\
\text{min = 2}
\end{align*}
\]

\[
\begin{array}{ccc}
1 \quad N, V & 4 \quad ? \\
\quad NP, VP \\
2 \quad N, M & 3 \quad N, V \\
\quad NP \\
\quad NP, VP
\end{array}
\]
Preterminal rules

<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

max = 1  
\[
\begin{array}{c}
1 \quad N, V \\
1 \quad NP, VP
\end{array}
\]

max = 2  
\[
\begin{array}{c}
2 \quad N, M \\
2 \quad NP
\end{array}
\]

max = 3  
\[
\begin{array}{c}
3 \quad N, V \\
3 \quad NP, VP
\end{array}
\]

Inner rules

\[
S \to NP \ VP
\]

\[
VP \to M \ V
\]

\[
NP \to N
\]

\[
NP \to N \ NP
\]

\[
N \to can
\]

\[
N \to lead
\]

\[
N \to poison
\]

\[
M \to can
\]

\[
M \to must
\]

\[
V \to poison
\]

\[
V \to lead
\]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Preterminal rules:

\[
S \rightarrow NP \; VP \\
VP \rightarrow M \; V \\
NP \rightarrow N \\
NP \rightarrow N \; NP
\]

Inner rules:

\[
N \rightarrow can \\
N \rightarrow lead \\
N \rightarrow poison \\
M \rightarrow can \\
M \rightarrow must \\
V \rightarrow poison \\
V \rightarrow lead
\]

The table and diagram show the relationships between lead, can, and poison, with rules governing their order and interaction.
\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow M \ V \]
\[ VP \rightarrow V \]
\[ NP \rightarrow N \]
\[ NP \rightarrow N \ NP \]

Preterminal rules

Inner rules

\[ N \rightarrow can \]
\[ N \rightarrow lead \]
\[ N \rightarrow poison \]

\[ M \rightarrow can \]
\[ M \rightarrow must \]

\[ V \rightarrow poison \]
\[ V \rightarrow lead \]
$S \rightarrow NP \ VP$

$VP \rightarrow M \ V$

$VP \rightarrow V$

$NP \rightarrow N$

$NP \rightarrow N \ NP$

$N \rightarrow can$

$N \rightarrow lead$

$N \rightarrow poison$

$M \rightarrow can$

$M \rightarrow must$

$V \rightarrow poison$

$V \rightarrow lead$
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Preterminal rules**

- $S \rightarrow NP \ VP$
- $VP \rightarrow M \ V$
- $VP \rightarrow V$
- $NP \rightarrow N$
- $NP \rightarrow N \ NP$

**Inner rules**

- $N \rightarrow can$
- $N \rightarrow lead$
- $N \rightarrow poison$
- $M \rightarrow can$
- $M \rightarrow must$
- $V \rightarrow poison$
- $V \rightarrow lead$

**Grid**

- $max = 1$
- $max = 2$
- $max = 3$

- $min = 0$

- $min = 1$

- $min = 2$

**Mid=1**
Preterminal rules

<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
S \rightarrow NP\ VP
\]

\[
VP \rightarrow M\ V
\]

\[
VP \rightarrow V
\]

\[
NP \rightarrow N
\]

\[
NP \rightarrow N\ NP
\]

Inner rules

\[
N \rightarrow can
\]

\[
N \rightarrow lead
\]

\[
N \rightarrow poison
\]

\[
M \rightarrow can
\]

\[
M \rightarrow must
\]

\[
V \rightarrow poison
\]

\[
V \rightarrow lead
\]
<table>
<thead>
<tr>
<th>lead</th>
<th>can</th>
<th>poison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Preterminal rules

Inner rules

 Apparently the sentence is ambiguous for the grammar: (as the grammar overgenerates)

\[
S \rightarrow NP \ VP
\]

\[
VP \rightarrow M \ V
\]

\[
VP \rightarrow V
\]

\[
NP \rightarrow N
\]

\[
NP \rightarrow N \ NP
\]

\[
N \rightarrow can
\]

\[
N \rightarrow lead
\]

\[
N \rightarrow poison
\]

\[
M \rightarrow can
\]

\[
M \rightarrow must
\]

\[
V \rightarrow poison
\]

\[
V \rightarrow lead
\]
Ambiguity

No subject-verb agreement, and *poison* used as an intransitive verb
Dependency parsing
Dependency trees

- Nodes are **words** (along with part-of-speech tags)
- Directed arcs encode **syntactic dependencies between them**
- Labels are types of relations between the words
  - `poss` – possessive
  - `dobj` – direct object
  - `nsubj` - subject
  - `det` - determiner
Some semantic information can be (approximately) derived from syntactic information

- Subjects (nsubj) are (often) agents ("initiator / doers for an action")
- Direct objects (dobj) are (often) patients ("affected entities")
Recovering shallow semantics

- Some semantic information can be (approximately) derived from syntactic information
  - Subjects (nsubj) are (often) agents ("initiator / doers for an action")
  - Direct objects (dobj) are (often) patients ("affected entities")

- But even for agents and patients consider:
  - Mary is baking a cake in the oven
  - A cake is baking in the oven

- In general it is not trivial even for the most shallow forms of semantics
  - E.g., consider prepositions: *in* can encode direction, position, temporal information, ...
Constituent and dependency representations

- Constituent trees can (potentially) be converted to dependency trees

- Dependency trees can (potentially) be converted to constituent trees
Dependency representation

- A dependency structure can be defined as a directed graph $G$, consisting of:
  - A set $V$ of nodes – vertices, words, punctuation, morphemes
  - A set $A$ of arcs – directed edges,
  - A linear precedence order $<$ on $V$ (word order).

- Labeled graphs:
  - Nodes in $V$ are labeled with word forms (and annotation).
  - Arcs in $A$ are labeled with dependency types.
  - $L = \{l_1, \ldots, l_{|L|}\}$ is the set of permissible arc labels;
  - Every arc in $A$ is a triple $(i, j, k)$, representing a dependency from $w_i$ to $w_j$ with label $l_k$.  

I prefer the morning flight through Denver.
Conversion from constituency to dependency

- Xia and Palmer (2001)
  - mark the head child of each node in a phrase structure, using the appropriate head rules
  - make the head of each non-head child depend on the head of the head-child

![Lexicalized tree](image)
Dependency vs Constituency

- Dependency structures explicitly represent
  - head-dependent relations (directed arcs),
  - functional categories (arc labels)
  - possibly some structural categories (parts of speech)

- Phrase (aka constituent) structures explicitly represent
  - phrases (nonterminal nodes),
  - structural categories (nonterminal labels)
Dependency vs Constituency trees

I prefer the morning flight through Denver

S
  NP
    Pro
    Verb
      I
      prefer
      Det
        the
        Nom
          Nom
          Noun
          P
          NP
            Noun
            flight
            through
            Pro
            Denver
I prefer the morning flight through Denver

Я предпочитаю утренний перелет через Денвер
Languages with free word order

I prefer the morning flight through Denver

Я предпочитаю утренний перелет через Денвер

Я предпочитаю через Денвер утренний перелет

Утренний перелет я предпочитаю через Денвер

Перелет утренний я предпочитаю через Денвер

Через Денвер я предпочитаю утренний перелет

Я через Денвер предпочитаю утренний перелет

...
Dependency relations"
Types of relationships

- The clausal relations NSUBJ and DOBJ identify the **arguments**: the subject and direct object of the predicate *cancel*

- The NMOD, DET, and CASE relations denote **modifiers** of the nouns *flights* and *Houston*.
## Grammatical functions

<table>
<thead>
<tr>
<th>Clausal Argument Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>Nominal subject</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect object</td>
</tr>
<tr>
<td>CCOMP</td>
<td>Clausal complement</td>
</tr>
<tr>
<td>XCOMP</td>
<td>Open clausal complement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nominal Modifier Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMOD</td>
<td>Nominal modifier</td>
</tr>
<tr>
<td>AMOD</td>
<td>Adjectival modifier</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Numeric modifier</td>
</tr>
<tr>
<td>APPOS</td>
<td>Appositional modifier</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>CASE</td>
<td>Prepositions, postpositions and other case markers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Notable Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONJ</td>
<td>Conjunct</td>
</tr>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
</tbody>
</table>

**Figure 13.2** Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)
Dependency Constraints

- Syntactic structure is complete (**connectedness**)
  - connectedness can be enforced by adding a special root node
- Syntactic structure is hierarchical (**acyclicity**)
  - there is a unique pass from the root to each vertex
- Every word has at most one syntactic head (**single-head constraint**)
  - except root that does not have incoming arcs

This makes the dependencies a tree

I prefer the morning flight through Denver
Projectivity

- Projective parse
  - arcs don’t cross each other
  - mostly true for English
- Non-projective structures are needed to account for
  - long-distance dependencies
  - flexible word order
Projectivity

- Dependency grammars do not normally assume that all dependency-trees are projective, because some linguistic phenomena can only be achieved using non-projective trees.

- But a lot of parsers assume that the output trees are projective

- Reasons
  - conversion from constituency to dependency
  - the most widely used families of parsing algorithms impose projectivity
## Non-Projective Statistics

<table>
<thead>
<tr>
<th>Language</th>
<th>Non-Projective Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>11.2%</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>5.4%</td>
</tr>
<tr>
<td>Chinese</td>
<td>0.0%</td>
</tr>
<tr>
<td>Czech</td>
<td>23.2%</td>
</tr>
<tr>
<td>Danish</td>
<td>15.6%</td>
</tr>
<tr>
<td>Dutch</td>
<td>36.4%</td>
</tr>
<tr>
<td>German</td>
<td>27.8%</td>
</tr>
<tr>
<td>Japanese</td>
<td>5.3%</td>
</tr>
<tr>
<td>Polish</td>
<td>18.9%</td>
</tr>
<tr>
<td>Slovene</td>
<td>22.2%</td>
</tr>
<tr>
<td>Spanish</td>
<td>1.7%</td>
</tr>
<tr>
<td>Swedish</td>
<td>9.8%</td>
</tr>
<tr>
<td>Turkish</td>
<td>11.6%</td>
</tr>
<tr>
<td>English</td>
<td>0.0% (SD: 0.1%)</td>
</tr>
</tbody>
</table>

Percentage of non-projective trees for some treebanks of the CoNLL-X Shared Task and English.
The parsing problem for a dependency parser is to find the optimal dependency tree $y$ given an input sentence $x$.

This amounts to assigning a syntactic head $i$ and a label $l$ to every node $j$ corresponding to a word $x_j$ in such a way that the resulting graph is a tree rooted at the node 0.
Parsing problem

- This is equivalent to finding a spanning tree in the complete graph containing all possible arcs.
Parsing algorithms

- Transition based
  - greedy choice of local transitions guided by a good classifier
  - deterministic
  - MaltParser (Nivre et al. 2008)

- Graph based
  - Minimum Spanning Tree for a sentence
  - McDonald et al.’s (2005) MSTParser
  - Martins et al.’s (2009) Turbo Parser
Transition Based Parsing

- greedy discriminative dependency parser
- motivated by a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho & Ullman, 1972).
- Nivre 2003
Configuration

$C = (\sigma, \beta, A)$

**Figure 13.5** Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.
Configuration

Stack: partially processed words

Oracle: a classifier

\[ C_{\text{initial}} = ([\text{ROOT}], w, \emptyset) \]
Operations

At each step choose:

- **Shift**

**Stack**: partially processed words

**Buffer**: unprocessed words

**Oracle**: a classifier
At each step choose:

- Shift
- Reduce left

**Stack:** partially processed words

**Buffer:** unprocessed words

**Oracle:** a classifier
Operations

At each step choose:

- Shift
- LeftArc or Reduce left
- RightArc or Reduce right

**Stack**: partially processed words

**Buffer**: unprocessed words

**Oracle**: a classifier

\[ C_{\text{accept}} = ([\text{ROOT}], \emptyset, A) \]
Shift-Reduce Parsing

Configuration:

- Stack, Buffer, Oracle, Set of dependency relations

Operations by a classifier at each step:

- **Shift**
  - remove $w_1$ from the buffer, push it onto the stack as $s_1$

- **LeftArc or Reduce left**
  - assert a head-dependent relation between $s_1$ and $s_2$ ($s_1 \rightarrow s_2$)
  - pop $s_1$ from the stack; pop $s_2$ from the stack; then push $(s_2 \leftarrow s_1)$ onto the stack

- **RightArc or Reduce right**
  - assert a head-dependent relation between $s_2$ and $s_1$ ($s_2 \rightarrow s_1$)
  - pop $s_1$ from the stack; pop $s_2$ from the stack; then push $(s_2 \rightarrow s_1)$ onto the stack
Want to see an example of transition-based parsing in action?

Slides 30-44 of this slide deck by Noah Smith do a really nice job of walking through the full transition-based assembly of a sentence’s parse visually.