Dependency Parsing
And Other Grammar Formalisms

Yejin Choi - University of Washington
Dependency Grammar

For each word, find one parent.

<table>
<thead>
<tr>
<th>Child</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>shot</td>
</tr>
<tr>
<td>shot</td>
<td>an</td>
</tr>
<tr>
<td>an</td>
<td>elephant</td>
</tr>
</tbody>
</table>

A child is dependent on the parent.
- A child is an argument of the parent.
- A child modifies the parent.
For each word, find one parent.

Child    Parent

A child is dependent on the parent.
- A child is an argument of the parent.
- A child modifies the parent.
For each word, find one parent.

Child Parent

A child is dependent on the parent.
- A child is an argument of the parent.
- A child modifies the parent.

I shot an elephant in my pajamas yesterday
I shot an elephant in my pajamas yesterday.
Typed Dependencies

nsubj(shot-2, i-1)
root(ROOT-0, shot-2)
det(elephant-4, an-3)
dobj(shot-2, elephant-4)

prep(shot-2, in-5)
pobj(in-5, pajamas-7)
poss(pajamas-7, my-6)

1 shot 2 an 3 elephant 4 in 5 my 6 pajamas 7
Both are context-free.
Both are used frequently today, but dependency parsers are more recently popular.

CKY Parsing algorithm:
- $O(N^3)$ using CKY & unlexicalized grammar
- $O(N^5)$ using CKY & lexicalized grammar ($O(N^4)$ also possible)

Dependency parsing algorithm:
- $O(N^5)$ using naïve CKY
- $O(N^3)$ using Eisner algorithm
- $O(N^2)$ based on minimum directed spanning tree algorithm (arborescence algorithm, aka, Edmond-Chu-Liu algorithm – see edmond.pdf)

Linear-time $O(N)$ Incremental parsing (shift-reduce parsing) possible for both grammar formalisms
CFG vs Dependency Parse II

- CFG focuses on “constituency” (i.e., phrasal/clausal structure)
- Dependency focuses on “head” relations.

- CFG includes non-terminals. CFG edges are not typed.
- No non-terminals for dependency trees. Instead, dependency trees provide “dependency types” on edges.

- Dependency types encode “grammatical roles” like
  - nsubj -- nominal subject
  - dobj – direct object
  - pobj – prepositional object
  - nsubjpass – nominal subject in a passive voice
CFG vs Dependency Parse III

- Can we get “heads” from CFG trees?
  - Yes. In fact, modern statistical parsers based on CFGs use hand-written “head rules” to assign “heads” to all nodes.

- Can we get constituents from dependency trees?
  - Yes, with some efforts.

- Can we transform CFG trees to dependency parse trees?
  - Yes, and transformation software exists. (stanford toolkit based on [de Marneffe et al. LREC 2006])

- Can we transform dependency trees to CFG trees?
  - Mostly yes, but (1) dependency parse can capture *non-projective* dependencies, while CFG cannot, and (2) people rarely do this in practice
Non Projective Dependencies

- Mr. Tomash will remain as a director emeritus.

- A hearing is scheduled on the issue today.
Non Projective Dependencies

- Projective dependencies: when the tree edges are drawn directly on a sentence, it forms a tree (without a cycle), and there is no crossing edge.

- Projective Dependency:

- Eg:

  Mr. Tomash will remain as a director emeritus.
Non Projective Dependencies

- Projective dependencies: when the tree edges are drawn directly on a sentence, it forms a tree (without a cycle), and there is no crossing edge.
- Non-projective dependency:

  Eg:

  A hearing is scheduled on the issue today.
Non Projective Dependencies

- which word does “on the issue” modify?
  - We scheduled a meeting on the issue today.
  - A meeting is scheduled on the issue today.

- CFGs capture only projective dependencies (why?)
Coordination across Constituents

- **Right-node raising:**
  - [[She bought] and [he ate]] bananas.

- **Argument-cluster coordination:**
  - I give [[you an apple] and [him a pear]].

- **Gapping:**
  - She likes sushi, and he sashimi

⇒ CFGs don’t capture coordination across constituents:
Coordination across Constituents

- She bought **and** he ate bananas.
- I give you an apple **and** him a pear.

Compare above to:

- She bought **and** ate bananas.
- She bought bananas **and** apples.
- She bought bananas **and** he ate apples.
The Chomsky Hierarchy

- Regular (or Right Linear) Languages
- Context-Free Languages
- Mildly Context-Sensitive Languages
- Context-Sensitive Languages
- Recursively Enumerable Languages
The Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Type</th>
<th>Common Name</th>
<th>Rule Skeleton</th>
<th>Linguistic Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turing Equivalent</td>
<td>$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$</td>
<td>HPSG, LFG, Minimalism</td>
</tr>
<tr>
<td>1</td>
<td>Context Sensitive</td>
<td>$\alpha A\beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$</td>
<td>TAG, CCG</td>
</tr>
<tr>
<td></td>
<td>Mildly Context Sensitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Context Free</td>
<td>$A \rightarrow \gamma$</td>
<td>Phrase-Structure Grammars</td>
</tr>
<tr>
<td>3</td>
<td>Regular</td>
<td>$A \rightarrow xB$ or $A \rightarrow x$</td>
<td>Finite-State Automata</td>
</tr>
</tbody>
</table>

- Lexical Functional Grammar (LFG) (Bresnan, 1982)
- Minimalist Grammar (Stabler, 1997)
- Tree-Adjoining Grammars (TAG) (Joshi, 1969)
- Combinatory Categorial Grammars (CCG) (Steedman, 1986)
Advanced Topics
- Eisner’s Algorithm -
Naïve CKY Parsing

$O(n^5 N^3)$ if $N$ nonterminals

$O(n^5)$ combinations

$p_{i,j,k}$

$0 \rightarrow n$

$\text{goal}$

$\text{takes}$

$\text{lt}$

$\text{lt}$

$\text{takes}$

$\text{two}$

$\text{to}$

$\text{tango}$

slides from Eisner
Eisner Algorithm  *(Eisner & Satta, 1999)*

- Without adding a dependency arc:
  - This happens only once as the very final step.

- When adding a dependency arc (head is higher):
  - This happens only once as the very final step.
Eisner Algorithm  (Eisner & Satta, 1999)

A triangle is a head with some left (or right) subtrees.

One trapezoid per dependency.
Eisner Algorithm (Eisner & Satta, 1999)

$O(n^3)$ combinations

Gives $O(n^3)$ dependency grammar parsing
Eisner Algorithm

- **Base case:**

  \[ \forall t \in \{\sqsubseteq, \sqsupseteq, \triangleleft, \triangleright\}, \quad \pi(i, i, t) = 0 \]

- **Recursion:**

  \[
  \begin{align*}
  \pi(i, j, \sqsubseteq) &= \max_{i \leq k < j} \left( \pi(i, k, \triangleright) + \pi(k + 1, j, \sqsubseteq) + \phi(w_j, w_i) \right) \\
  \pi(i, j, \sqsupseteq) &= \max_{i \leq k < j} \left( \pi(i, k, \triangleright) + \pi(k + 1, j, \sqsubseteq) + \phi(w_i, w_j) \right) \\
  \pi(i, j, \triangleleft) &= \max_{i \leq k < j} \left( \pi(i, k, \triangleleft) + \pi(k + 1, j, \sqsubseteq) \right) \\
  \pi(i, j, \triangleright) &= \max_{i \leq k < j} \left( \pi(i, k, \triangleright) + \pi(k + 1, j, \triangleright) \right)
  \end{align*}
  \]

- **Final case:**

  \[ \pi(1, n, \triangleleft\triangleright) = \max_{1 \leq k < n} \left( \pi(1, k, \triangleleft) + \pi(k + 1, n, \triangleright) \right) \]
Advanced Topics:
Mildly Context-Sensitive Grammar Formalisms
I. Tree Adjoining Grammar (TAG)

Some slides adapted from Julia Hockenmaier’s
**TAG Lexicon (Supertags)**

- Tree-Adjoining Grammars (TAG) (Joshi, 1969)
- “… super parts of speech (supertags): almost parsing” (Joshi and Srinivas 1994)
- POS tags enriched with syntactic structure
- Also used in other grammar formalisms (e.g., CCG)
likes bananas with
always with

TAG Lexicon (Supertags)
TAG rule 1: Substitution

\[ \alpha_1: \]

\[ \alpha_2: \]

\[ \alpha_3: \]

\[ \text{Derived tree:} \]

\[ \text{Substitute} \]

\[ \text{Derivation tree:} \]

\[ \alpha_1 \]

\[ \alpha_2 \]

\[ \alpha_3 \]
TAG rule 2: Adjunction

**Derived tree:**

**Auxiliary tree**

**Foot node**

**ADJOIN**

**Derivation tree:**
Example: TAG Lexicon

\[ \alpha_2: \]
NP  
John

\[ \alpha_3: \]
NP  
tapas

\[ \alpha_1: \]
S
NP  VP
VBZ  NP
  eats

\[ \beta_1: \]
VP  VP*
  RB  always
Example: TAG Derivation

```
S
 /\ 
VP NP
   /\ 
  VP NP
     /\ 
    RB VP
      /\ 
     VP NP
       /\ 
      John eats tapas
```

α1:

β1:

α2:

α3:
Example: TAG Derivation

α1

α2 β1 α3

S

NP VP

John VBZ NP tapas

eats

β1 VP

RB VP*

always
Example: TAG Derivation

```
S
  NP
  | 
  | John
  NP
  | 
  | always
  VP
  | 
  | RB
  | 
  | eats
  VP*
  | 
  | NP
  | 
  | tapas
```
(1) Can handle long distance dependencies
(2) Cross-serial Dependencies

dat Jan Piet Marie de kinderen zag helpen laten zwemmen

- Dutch and Swiss-German
- Can this be generated from context-free grammar?
\(a^n b^n: \text{Cross-serial dependencies}\)

**Elementary trees:**

- \(a S S b\)
- \(a S S S b\)
- \(a S S S^* b\)

**Deriving \(aabb\):**

1. \(a S S b\)
2. \(a S S\)
3. \(a S S\)
4. \(a S S S^* b\)
Tree Adjoining Grammar (TAG)

- TAG: Aravind Joshi in 1969
- Supertagging for TAG: Joshi and Srinivas 1994

- Pushing grammar down to lexicon.
- With just two rules: substitution & adjunction

- Parsing Complexity:
  - $O(N^7)$

- Xtag Project (TAG Penntree) (http://www.cis.upenn.edu/~xtag/)

- Local expert!
  - Fei Xia @ Linguistics (https://faculty.washington.edu/fxia/)
II. Combinatory Categorial Grammar (CCG)

Some slides adapted from Julia Hockenmaier’s
Categories

- **Categories = types**
  - **Primitive categories**
    - N, NP, S, etc
  - **Functions**
    - a combination of primitive categories
    - S/NP, (S/NP) / (S/NP), etc
    - V, VP, Adverb, PP, etc
Combinatory Rules

**Application**
- forward application: \( \frac{x}{y} \ y \rightarrow x \)
- backward application: \( y \ \frac{x}{y} \rightarrow x \)

**Composition**
- forward composition: \( \frac{x}{y} \ y/z \rightarrow x/z \)
- backward composition: \( y/z \ x/y \rightarrow x/z \)
- (forward crossing composition: \( \frac{x}{y} \ y/z \rightarrow x/z \))
- (backward crossing composition: \( x/y \ y/z \rightarrow x/z \))

**Type-raising**
- forward type-raising: \( x \rightarrow y / (y \backslash x) \)
- backward type-raising: \( x \rightarrow y \backslash (y/x) \)

**Coordination \( <&> \)**
- \( x \ \text{conj} \ x \rightarrow x \)
Combinatory Rules 1 : Application

- **Forward application “>”**
  - X/Y Y $\Rightarrow$ X
  - (S\NP)/NP NP $\Rightarrow$ S\NP

- **Backward application “<“**
  - Y X\Y $\Rightarrow$ X
  - NP S\NP $\Rightarrow$ S
Function

- \( \text{likes} := (S\backslash NP) / NP \)
  - A transitive verb is a function from NPs into predicate \( S \). That is, it accepts two NPs as arguments and results in \( S \).

- Transitive verb: \((S\backslash NP) / NP\)
- Intransitive verb: \( S\backslash NP \)
- Adverb: \((S\backslash NP) \backslash (S\backslash NP)\)
- Preposition: \((NP\backslash NP) / NP\)
- Preposition: \(((S\backslash NP) \backslash (S\backslash NP)) / NP\)
CCG Derivation:

Mary \hspace{1cm} \text{likes} \hspace{1cm} \text{musicals}

\[
NP \quad (S \backslash NP)/NP \quad NP \quad \rightarrow \quad S \backslash NP \quad \leftarrow S
\]

CFG Derivation:

```
NP \hspace{1cm} V \hspace{1cm} NP
```

```
S \quad VP \quad NP
```

Mary \hspace{1cm} likes \hspace{1cm} musicals
Combinatory Rules

- **Application**
  - forward application: \( x/y \ y \rightarrow x \)
  - backward application: \( y \ x\backslash y \rightarrow x \)

- **Composition**
  - forward composition: \( x/y \ y/z \rightarrow x/z \)
  - backward composition: \( y\backslash z \ x\backslash y \rightarrow x\backslash z \)
  - forward crossing composition: \( x/y \ y\backslash z \rightarrow x\backslash z \)
  - backward crossing composition: \( x\backslash y \ y/z \rightarrow x/z \)

- **Type-raising**
  - forward type-raising: \( x \rightarrow y \ (y\backslash x) \)
  - backward type-raising: \( x \rightarrow y \ (y/x) \)

**Coordination**<\&>
- \( x \text{ conj } x \rightarrow x \)
Combinatory Rules 4 : Coordination

- $X \text{ conj } X \rightarrow X$

- Alternatively, we can express coordination by defining conjunctions as functions as follows:

  - $\text{and} := (X \backslash X) / X$
Coordination with CCG

Examples from Prof. Mark Steedman
Coordination with CCG

- Marcel NP
- conjectured \( (S\backslash NP)/NP \)
- and \( (X\backslash X)/X \)
- proved \( (S\backslash NP)/NP \)
- completeness NP

**Application**
- **forward application:** \( x/y \ y \rightarrow x \)
- **backward application:** \( y \ x\backslash y \rightarrow x \)
Coordination with CCG

```
Marcel (S\NP)/NP
NP

\[
\frac{\text{conjuncted}}{(S\ NP)/NP} \quad \frac{\text{and}}{(X\ X)/X} \quad \frac{\text{proved}}{(S\ NP)/NP} \quad \frac{\text{completeness}}{NP} \\
\frac{\frac{(S\ NP)/NP}{((S\ NP)/NP)/((S\ NP)/NP)}}{>}
\frac{(S\ NP)/NP}{<}
\frac{S\ NP}{>}
\frac{S}{<}
\]
```

- Application
  - forward application:  $x/y\ y \Rightarrow x$
  - backward application:  $y\ x/y \Rightarrow x$
Combinatory Rules

- **Application**
  - forward application: \( x/y \ y \rightarrow x \)
  - backward application: \( y \ x\y \rightarrow x \)

- **Composition**
  - forward composition: \( x/y \ y/z \rightarrow x/z \)
  - backward composition: \( y\z \ x\y \rightarrow x\z \)
  - forward crossing composition: \( x/y \ y\z \rightarrow x\z \)
  - backward crossing composition: \( x\y \ y/z \rightarrow x/z \)

- **Type-raising**
  - forward type-raising: \( x \rightarrow y/ (y\x) \)
  - backward type-raising: \( x \rightarrow y\ \(y/x) \)

- **Coordination <&>**
  - \( x \ \text{conj} \ x \rightarrow x \)
Coordination with CCG

<table>
<thead>
<tr>
<th>Marcel</th>
<th>conjectured</th>
<th>and</th>
<th>might</th>
<th>prove</th>
<th>completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S\NP)/NP</td>
<td>(X/X)/X</td>
<td>(S\NP)/(((S\NP)))</td>
<td>(S\NP)/NP</td>
<td>NP</td>
</tr>
</tbody>
</table>

- **Application**
  - forward application: \( x/y \ y \Rightarrow x \)
  - backward application: \( y \ x\{y \Rightarrow x \)

- **Composition**
  - forward composition: \( x/y \ y/z \Rightarrow x/z \)
  - backward composition: \( y\z \ x\y \Rightarrow x\z \)
  - forward crossing composition: \( x/y \ y\z \Rightarrow x\z \)
  - backward crossing composition: \( x\y \ y/z \Rightarrow x/z \)
Coordination with CCG

- Application
  - forward application:  $x/y \ y \rightarrow x$
  - backward application:  $y \ x/y \rightarrow x$

- Composition
  - forward composition:  $x/y \ y/z \rightarrow x/z$
  - backward composition:  $y/z \ x/y \rightarrow x/z$
  - forward crossing composition:  $x/y \ y/z \rightarrow x/z$
  - backward crossing composition:  $x/y \ y/z \rightarrow x/z$
Combinatory Rules

- **Application**
  - forward application: \( x/y \ y \rightarrow x \)
  - backward application: \( y \ x/y \rightarrow x \)

- **Composition**
  - forward composition: \( x/y \ y/z \rightarrow x/z \)
  - backward composition: \( y/z \ x/y \rightarrow x/z \)
  - forward crossing composition: \( x/y \ y/z \rightarrow x/z \)
  - backward crossing composition: \( x/y \ y/z \rightarrow x/z \)

- **Type-raising**
  - forward type-raising: \( x \rightarrow y / (y\times) \)
  - backward type-raising: \( x \rightarrow y \ (y/x) \)

- **Coordination </&>**
  - \( x \ \text{conj} \ x \rightarrow x \)
Combinatory Rules 3 : Type-Raising

- Turns an argument into a function

- Forward type-raising: \( X \rightarrow T / (T\setminus X) \)
- Backward type-raising: \( X \rightarrow T \setminus (T/X) \)

For instance...

- Subject type-raising: \( NP \rightarrow S / (S \setminus NP) \)
- Object type-raising: \( NP \rightarrow (S\setminus NP) \setminus ((S\setminus NP) / NP) \)
Combinatory Rules 3: Type-Raising

- **Application**
  - forward application: $x/y \quad y \rightarrow x$
  - backward application: $y \quad x\backslash y \rightarrow x$

- **Type-raising**
  - forward type-raising: $x \rightarrow y \quad y / (y\backslash x)$
  - backward type-raising: $x \rightarrow y \quad y \backslash (y\backslash x)$
  - Subject type-raising: $NP \rightarrow S \quad (S \backslash NP)$
  - Object type-raising: $NP \rightarrow (S\backslash NP) \quad ((S\backslash NP) / NP)$

- **Coordination <&>**
  - $x \quad \text{conj} \quad x \rightarrow x$
Combinatory Rules 3: Type-Raising

\[
\begin{align*}
I & \quad (S \backslash NP) / NP & \quad \text{dislike} & \quad \text{and} & \quad \text{Mary} & \quad \text{likes} & \quad \text{musicals} \\
NP & \quad (S \backslash NP) / NP & \quad \text{CONJ} & \quad NP & \quad (S \backslash NP) / NP & \quad NP \\
S / (S \backslash NP) & \quad S / (S \backslash NP) & \quad S / NP & \quad S / NP & \quad S / NP & \quad S / NP \\
& \quad >^T & \quad >^T & \quad >^T & \quad >^T & \quad >^T \\
& \quad S / NP & \quad S / NP & \quad S / NP & \quad S / NP & \quad S / NP \\
& \quad >^B & \quad >^B & \quad >^B & \quad < & \quad > \\
& \quad S / NP & \quad S / NP & \quad S / NP & \quad S / NP & \quad S \\
\end{align*}
\]
Combinatory Categorial Grammar (CCG)

- CCG: Steedman in 1986
- Pushing grammar down to lexicon.
- With just a few rules: application, composition, type-raising
- We’ve looked at only syntactic part of CCG
- A lot more in the semantic part of CCG (using lambda calculus)
- Parsing Complexity:
  - $O(N^6)$
- Local expert!