Dependency Parsing
And Other Grammar Formalisms

Yejin Choi - University of Washington
Dependency Grammar

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
For each word, find one parent.

Child         Parent

A child is \textit{dependent} on the parent.
- A child is an \textit{argument} of the parent.
- A child \textit{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.
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)
poss(pajamas-7, my-6)
pobj(in-5, pajamas-7)
Naïve CKY Parsing

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

$O(n^5)$ combinations

It takes two to tango.

$O(\frac{n^5}{N^3})$ if $N$ nonterminals.

Slides from Eisner & Smith.
Eisner Algorithm  (Eisner & Satta, 1999)

This happens only once as the very final step

Without adding a dependency arc

When adding a dependency arc (head is higher)
Eisner Algorithm (Eisner & Satta, 1999)

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

One trapezoid per dependency.

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

$O(n)$ combinations

$O(n^3)$ combinations

$O(n^3)$ combinations

$O(n^3)$ dependency grammar parsing

slides from Eisner & Smith
Eisner Algorithm

- **Base case:**
  \[ \forall t \in \{\sqsubseteq, \sqsupset, \triangleleft, \triangleright\}, \quad \pi(i, i, t) = 0 \]

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

- **Final case:**
  \[
  \pi(1, n, \triangleleft\triangleright) = \max_{1 \leq k < n} \left( \pi(1, k, \triangleleft) + \pi(k + 1, n, \triangleright) \right)
  \]
CFG vs Dependency Parse I

- 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 II

- 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.
 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
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></td>
</tr>
<tr>
<td></td>
<td>Mildly Context Sensitive</td>
<td></td>
<td>TAG, CCG</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)
Mildly Context-Sensitive Grammar Formalisms
I. Tree Adjoining Grammar (TAG)
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)
Example: TAG Lexicon

\[ \alpha_2: \]
\[ \text{NP} \]
\[ \text{John} \]

\[ \alpha_3: \]
\[ \text{NP} \]
\[ \text{NP} \]
\[ \text{tapas} \]

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

\[ \beta_1: \]
\[ \text{VP} \]
\[ RB \]
\[ VP^* \]
\[ \text{always} \]
Example: TAG Derivation

\[
\begin{align*}
\alpha_1 & : S \\
\alpha_2 : & NP \\
\alpha_3 : & NP \\
\alpha_1 : & VP \\
\beta_1 : & RB \\
\end{align*}
\]

John \textit{eats} always \textit{tapas}
Example: TAG Derivation

\[
\begin{align*}
\alpha_1 & \quad \alpha_2 & \quad \beta_1 & \quad \alpha_3 \\
\alpha_1 & \quad \beta_1 & \quad \alpha_3 \\
S & \quad NP & \quad VP \\
& \quad John & \quad VBZ & \quad NP \\
& \quad eats & \quad NP & \quad tapas \\
& \quad RB & \quad VP^* & \quad always
\end{align*}
\]
Example: TAG Derivation

```
S
  NP
  |  VP
  |   RB
  |    VP*
  |     eats
  |      NP
   John  always  tapas
```
TAG rule 1: Substitution

Derived tree:

Substitute

Derivation tree:

α2

α3

α1

α1:

α2:

X

Y

α3:

X

Y
TAG rule 2: Adjunction

Auxiliary tree

Foot node

Derived tree:

ADJOIN

Derivation tree:
(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?
$a^nb^n$: Cross-serial dependencies

Elementary trees:

Deriving $aabb$
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: $x/y \ y \rightarrow x$
- backward application: $y \ x\backslash y \rightarrow x$

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

Coordination $<&>$
- $x \ conj \ x \rightarrow x$
Combinatory Rules 1: Application

- Forward application “>”
  - X/Y  Y → X
  - (S\NP)/NP  NP → S\NP

- Backward application “<”
  - Y  X\Y → X
  - NP  S\NP → S
Function

- \( \text{likes} := (S \setminus \text{NP}) / \text{NP} \)
  - 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: \((S \setminus \text{NP}) / \text{NP}\)
- Intransitive verb: \(S \setminus \text{NP}\)
- Adverb: \((S \setminus \text{NP}) \setminus (S \setminus \text{NP})\)
- Preposition: \((\text{NP} \setminus \text{NP}) / \text{NP}\)
- Preposition: \(((S \setminus \text{NP}) \setminus (S \setminus \text{NP})) / \text{NP}\)
CCG Derivation:

\[
\begin{align*}
\text{Mary} & \quad \text{likes} \quad \text{musicals} \\
NP & \quad \frac{\text{(S}\backslash NP)/NP}{NP} & \quad NP \\
& \quad \frac{\text{S}\backslash NP}{NP} \\
\text{S} & \quad < \\
\end{align*}
\]

CFG Derivation:

```
  S
 /   \   \
VP  V   VP
 |  |  |   |
NP NP NP
   |   |
   VN
   VN
   VN
```

Mary  likes  musicals
NP    V      NP
VP    V
S
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 <&>**
- $x \ \text{conj} \ x \rightarrow x$
Combinatory Rules 4 : Coordination

- X conj X \( \rightarrow \) X

- Alternatively, we can express coordination by defining conjunctions as functions as follows:
  
  - and := \( (X\backslash X) / X \)
Coordination with CCG

I loathe and detest opera

Example from Prof. Mark Steedman
Coordination with CCG

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

**Application**
- forward application: x/y  y → x
- backward application: y  x\y → x
Coordination with CCG

Marcel NP

(S\NP)/NP

called NP

and

(X\X)/X

proven NP

(S\NP)/NP

((S\NP)/NP)/(S\NP)/NP

completeness NP

((S\NP)/NP)/(S\NP)/NP

(S\NP)/NP

(S\NP)/NP

S\NP

S

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

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

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

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

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

| Marcel | conjectured \( (S\backslash NP)/NP \) and \( (X\backslash X)/X \) might \( (S\backslash NP)/((S\backslash NP)) \) prove \( (S\backslash NP)/NP \) completeness \( NP \) |
|---|---|---|---|---|

- **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**: \( x\backslash y \ y/z \rightarrow x/z \)
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: $x \Rightarrow y / (y/x)$
  - 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

<table>
<thead>
<tr>
<th>I</th>
<th>dislike</th>
<th>and</th>
<th>Mary</th>
<th>likes</th>
<th>musicals</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S\NP)/NP</td>
<td>CONJ</td>
<td>NP</td>
<td>(S\NP)/NP</td>
<td>NP</td>
</tr>
</tbody>
</table>

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

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

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

\[
\begin{align*}
I & \quad \text{dislike} \quad \text{and} \quad \text{Mary} \quad \text{likes} \quad \text{musicals} \\
NP & \quad (S\backslash NP)/NP \quad CONJ \quad NP & \quad (S\backslash NP)/NP \\
S/(S\backslash NP) & \quad \rightarrow^T \quad S/NP & \quad \rightarrow^T \\
& \quad S/NP & \quad S/NP & \quad \rightarrow^B \\
& \quad \rightarrow^B \\
& \quad \rightarrow^B \\
& \quad S/NP & \quad \rightarrow^B \\
& \quad \rightarrow \\
& \quad S
\end{align*}
\]
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!