CSE 517
Natural Language Processing
Winter 2017

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

I shot an elephant in my pajamas
For each word, find one 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

\[ O(n^5 N^3) \text{ if } N \text{ nonterminals} \]

\[ O(n^5) \text{ combinations} \]

\[ \begin{align*}
\text{goal} & \\
\text{takes} & \\
\text{takes} & \\
\text{takes} & \\
\text{to} & \\
tango & \\
\end{align*} \]

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

One trapezoid per dependency.

Eisner Algorithm (Eisner & Satta, 1999)

It takes two to tango

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

$O(n)$ combinations

$O(n^3)$ combinations

$O(n^3)$ combinations

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

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

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

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

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)
TAG Lexicon (Supertags)

always
bananas
likes
the
with
Example: TAG Lexicon

\[ \alpha_2: \]
\[
\begin{array}{c}
NP \\
John
\end{array}
\]

\[ \alpha_1: \]
\[
\begin{array}{c}
S \\
NP \\
\text{eats}
\end{array}
\]

\[ \beta_1: \]
\[
\begin{array}{c}
VP \\
RB \\
\text{always}
\end{array}
\]

\[ \alpha_3: \]
\[
\begin{array}{c}
NP \\
\text{tapas}
\end{array}
\]
Example: TAG Derivation

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

\[
\begin{array}{c}
\alpha_1 \\
\alpha_2 \quad \beta_1 \quad \alpha_3
\end{array}
\]

\[
S 
\quad \text{NP} 
\quad \text{VP} \\
\quad \text{John} 
\quad \text{VBZ} 
\quad \text{eats} \\
\quad \text{NP} 
\quad \text{tapas} \\
\quad \text{VP} 
\quad \text{RB} 
\quad \text{always} \\
\quad \text{VP} 
\quad \text{VP}^*
\end{array}
\]
Example: TAG Derivation

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

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

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

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

\[
\alpha_1
\]

\[
\alpha_2
\]

\[
\alpha_3
\]
TAG rule 2: Adjunction

\[
\alpha_1: \quad \begin{array}{c}
\boxed{X} \\
\boxed{X^*}
\end{array} \\
\beta_1: \quad \begin{array}{c}
\boxed{X}
\end{array}
\]

Auxiliary tree

Derived tree:

Foot node

\[
\begin{array}{c}
\alpha_1 \\
\beta_1
\end{array}
\]

Derivation tree:

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

ADJOIN
(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?
$a^n b^n$: 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: $y x / y \rightarrow x$

**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 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

- `likes := (S\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\NP) / NP`
- Intransitive verb: `S\NP`
- Adverb: `(S\NP) \ (S\NP)`
- Preposition: `(NP\NP) / NP`
- Preposition: `((S\NP) \ (S\NP)) / NP`
CCG Derivation:

Mary \quad \text{likes} \quad \text{musicals}

\[
\begin{array}{c}
\frac{NP}{ (S \setminus NP) / NP } \\
\frac{NP}{ S \setminus NP } & \quad \text{\textgreater} \\
\frac{S}{ } & \quad \text{\langle} \\
\end{array}
\]

CFG Derivation:

\[
\begin{array}{c}
\text{Mary} \\
\text{NP} \\
\text{VP} \\
\text{S} \\
\end{array}
\]

\[
\begin{array}{c}
\text{likes} \\
\text{V} \\
\text{NP} \\
\text{S} \\
\end{array}
\]

\[
\begin{array}{c}
\text{musicals} \\
\text{NP} \\
\end{array}
\]
Combinatory Rules

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

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

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

Coordination $\lhd&\rhd$

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

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

and := (X\X) / X
Coordination with CCG

I loathe and detest opera

\[ \overline{NP} \quad (S \backslash NP)/NP \quad CONJ \quad (S \backslash NP)/NP \quad \overline{NP} \]

\[ \overline{(S \backslash NP)/NP} \]

\[ \overline{(S \backslash NP)/NP} \]

\[ \overline{S \backslash NP} \]

\[ S \]

Examples from Prof. Mark Steedman
Coordination with CCG

Marcel  
\[ \frac{\text{NP}}{\text{NP}} \]

\text{conjectured}  \quad \frac{(S \backslash \text{NP})/\text{NP}}{}  \quad \text{and}  \quad \frac{\text{X}/\text{X}}{\text{X}}  \quad \text{proved}  \quad \frac{(S \backslash \text{NP})/\text{NP}}{\text{NP}}  \quad \text{completeness}  \quad \frac{\text{NP}}{\text{NP}}

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

Marcel
NP

conjectured
(S\NP)/NP

and
(X/X)/X

proved
(S\NP)/NP

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

(S\NP)/NP

(S\NP)/NP

completeness
NP

S\NP

S

- Application
  - forward application:  \text{x/y}  y \Rightarrow x
  - backward application:  y  x\text{\slash}y \Rightarrow x
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:  \( x \rightarrow y \backslash (y/x) \)

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

- **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

I
NP

dislike
(S\NP)/NP

and
CONJ

Mary
NP

likes
(S\NP)/NP

musicals
NP

- Application
  - forward application: x/y \[\rightarrow\] x
  - backward application: x/\[\rightarrow\] x

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

- Coordination <&>
  - x conj x \[\rightarrow\] x
Combinatory Rules 3 : Type-Raising

\[
\begin{array}{c}
S/(S\backslash NP) \\
\downarrow \text{>}_T \\
S/\text{NP} \\
\downarrow \text{>}_B \\
S/\text{NP} \\
\downarrow \text{<&>} \\
S/\text{NP} \\
\downarrow \text{>} \\
S
\end{array}
\]
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!