CS114 Lecture 14
Dependency Grammars and Functional Unification Grammars

March 11, 2013
Professor Meteer

Slides from UPenn, Adapted from slides by Kathy McCoy, University of Delaware
### Another Earley Example

**CHART 1**

<table>
<thead>
<tr>
<th>Step</th>
<th>Rule</th>
<th>Predictor</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Spec → S</td>
<td>Predictor</td>
</tr>
<tr>
<td>S1</td>
<td>S → . NP VP</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S2</td>
<td>S → . VP</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S3</td>
<td>NP → . Det Noun</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S4</td>
<td>NP → . PrN</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S5</td>
<td>Det → . A</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S6</td>
<td>Det → . the</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S7</td>
<td>PrN → . Mark</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S8</td>
<td>VP → . V</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S9</td>
<td>VP → . V NP</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S10</td>
<td>V → . Mark</td>
<td>[0,0]</td>
</tr>
<tr>
<td>S11</td>
<td>V → . read</td>
<td>[0,0]</td>
</tr>
</tbody>
</table>

- **Spec → S**
- **S → NP VP**
- **S → VP**
- **NP → Det Noun**
- **NP → PrN**
- **VP → V**
- **VP → V NP**
- **Det → a | the**
- **N → book**
- **V → Mark | read**
- **PrN → Mark**
Chart 1: Mark read

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Predictors</th>
</tr>
</thead>
<tbody>
<tr>
<td>S12 PrN→ Mark .</td>
<td>[0,1] Scanner S7</td>
<td></td>
</tr>
<tr>
<td>S13 V → Mark .</td>
<td>[0,1] Scanner S10</td>
<td></td>
</tr>
<tr>
<td>S14 VP → V .</td>
<td>[0,1] Completer S13</td>
<td></td>
</tr>
<tr>
<td>S15 S → VP .</td>
<td>[0,1] Completer S14</td>
<td></td>
</tr>
<tr>
<td>S16 Spec → S .</td>
<td>[0,1] Completer S15</td>
<td></td>
</tr>
<tr>
<td>S17 VP → V . NP</td>
<td>[0,1] Completer S13</td>
<td></td>
</tr>
<tr>
<td>S18 NP → PrN .</td>
<td>[0,1] Completer S12</td>
<td></td>
</tr>
<tr>
<td>S19 S → NP . VP</td>
<td>[0,1] Completer S18</td>
<td></td>
</tr>
<tr>
<td>S20 NP → . Det Noun</td>
<td>[1,1] Predictor S17</td>
<td></td>
</tr>
<tr>
<td>S21 NP → . PrN</td>
<td>[1,1] Predictor S17</td>
<td></td>
</tr>
<tr>
<td>S22 Det → . a</td>
<td>[1,1] Predictor S20</td>
<td></td>
</tr>
<tr>
<td>S23 Det → . the</td>
<td>[1,1] Predictor S20</td>
<td></td>
</tr>
<tr>
<td>S24 PrN→ . Mark</td>
<td>[1,1] Predictor S21</td>
<td></td>
</tr>
<tr>
<td>S25 VP → . V</td>
<td>[1,1] Predictor S19</td>
<td></td>
</tr>
<tr>
<td>S26 VP → . V NP</td>
<td>[1,1] Predictor S19</td>
<td></td>
</tr>
<tr>
<td>S27 V→ . Mark</td>
<td>[1,1] Predictor S25,26</td>
<td></td>
</tr>
<tr>
<td>S28 V→ . read</td>
<td>[1,1] Predictor S25,26</td>
<td></td>
</tr>
</tbody>
</table>
Chart 2: Mark Read

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Completer</th>
</tr>
</thead>
<tbody>
<tr>
<td>S29 V → read</td>
<td></td>
<td>[1,2]Scanner</td>
</tr>
<tr>
<td>S30 VP → V</td>
<td></td>
<td>[1,2]Completer S29</td>
</tr>
<tr>
<td>S31 VP → V . NP</td>
<td></td>
<td>[1,2]Completer S29</td>
</tr>
<tr>
<td>S32 S → VP</td>
<td></td>
<td>[1,2]Completer S30</td>
</tr>
<tr>
<td>S33 Spec → S</td>
<td></td>
<td>[1,2]Completer S32</td>
</tr>
<tr>
<td>S34 S → NP VP</td>
<td></td>
<td>[0,2]Completer S19, S30</td>
</tr>
<tr>
<td>S35 Spec → S</td>
<td>[0,2]</td>
<td>Completer S34</td>
</tr>
</tbody>
</table>

S19 S → NP . VP      [0,1]Completer S18
S18 NP → PrN .       [0,1]Completer S12
S12 PrN→ Mark .      [0,1]Scanner S7
Dependency Grammars

• In CFG-style phrase-structure grammars the main focus is on *constituents*.  
• But it turns out you can get a lot done with just binary relations among the words in an utterance.  
• In a dependency grammar framework, a parse is a tree where
  – the nodes stand for the words in an utterance
  – The links between the words represent dependency relations between pairs of words.
    • Relations may be typed (labeled), or not.
Well-formedness

• A dependency graph is **well-formed** iff
  
  – **Single head**: Each word has only one head.
  
  – **Acyclic**: The graph should be acyclic.
  
  – **Connected**: The graph should be a single tree with all the words in the sentence.
  
  – **Projective**: If word A **depends** on word B, then all words between A and B are also **subordinate** to B (i.e. dominated by B).
Comparison

• Dependency structures explicitly represent
  – Head-dependent relations (directed arcs)
  – Functional categories (arc labels)
  – Possibly some structural categories (parts-of-speech)

• Phrase structure explicitly represent
  – Phrases (non-terminal nodes)
  – Structural categories (non-terminal labels)
  – Possibly some functional categories (grammatical functions)
# Dependency Relations

<table>
<thead>
<tr>
<th>Argument Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>nominal subject</td>
</tr>
<tr>
<td>csubj</td>
<td>clausal subject</td>
</tr>
<tr>
<td>dobj</td>
<td>direct object</td>
</tr>
<tr>
<td>iobj</td>
<td>indirect object</td>
</tr>
<tr>
<td>pobj</td>
<td>object of preposition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifier Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmod</td>
<td>temporal modifier</td>
</tr>
<tr>
<td>appos</td>
<td>appositional modifier</td>
</tr>
<tr>
<td>det</td>
<td>determiner</td>
</tr>
<tr>
<td>prep</td>
<td>prepositional modifier</td>
</tr>
</tbody>
</table>
Dependency Parse

They hid the letter on the shelf
Dependency Tree with Labels

Red figures on the screens indicated falling stocks.
Dependency Parsing

• The dependency approach has a number of advantages over full phrase-structure parsing.
  – Deals well with free word order languages where the constituent structure is quite fluid
  – Parsing is much faster than CFG-bases parsers
  – Dependency structure often captures the syntactic relations needed by later applications
    • CFG-based approaches often extract this same information from trees anyway.
There are two modern approaches to dependency parsing:

- Optimization-based approaches that search a space of trees for the tree that best matches some criteria.
- Shift-reduce approaches that greedily take actions based on the current word and state.
• Three main traditions
  – Dynamic programming
    • CYK, Eisner, McDonald
  – Constraint satisfaction
    • Maruyama, Foth et al., Duchier
  – Deterministic search
    • Covington, Yamada and Matsumuto, Nivre
Dynamic Programming

- Basic Idea: Treat dependencies as constituents.
- Use, e.g., CYK parser (with minor modifications)
Red figures on the screen indicated falling stocks.
Example

Spans:

Red figures on the screen indicated falling stocks

{Red figures} {indicated falling stocks}
Assembly of correct parse

Start by combining adjacent words to minimal spans

{Red figures} {figures on} {on the}
Combine spans which overlap in one word; this word must be governed by a word in the left or right span.
Combine spans which overlap in one word; this word must be governed by a word in the left or right span.
Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

Invalid span
Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

\[
\{\text{indicated}, \text{falling}\} + \{\text{falling}, \text{stocks}\} \rightarrow \{\text{indicated}, \text{falling}, \text{stocks}\}
\]
Features and Unification
Capturing Grammatical Features

A Simple Context Free Grammar Fragment

NP → Det N
NP → PropN

Det → a, the, this, those

N → book, dog, books, dogs
PropN → John, Mary

V → sneezed, visited, gave
eat, eats

S → NP VP

VP → V
(John sneezed)

VP → V NP
(John visited Mary)

VP → V NP NP
(John gave Mary a book)

VP → V NP PP
(John gave a book to Mary)
Agreement

Determiner/Noun Agreement

- This dog
- Those dogs

Subject/Verb Agreement

- This dog eats
- Those dogs eat

Our grammar also generates

- *This dogs
- *Those dog

- *This dog eat
- *Those dogs eats
Encoding Number Agreement in CFGs

\[
\begin{align*}
\text{NP}_{\text{sing}} & \rightarrow \text{Det}_{\text{sing}} \text{ N}_{\text{sing}} \\
\text{NP}_{\text{pl}} & \rightarrow \text{Det}_{\text{pl}} \text{ N}_{\text{pl}} \\
\text{VP}_{\text{pl}} & \rightarrow \text{V}_{\text{pl}} \text{ NP}_{\text{sing}} \\
\text{VP}_{\text{pl}} & \rightarrow \text{V}_{\text{pl}} \text{ NP}_{\text{pl}} \\
\text{VP}_{\text{sing}} & \rightarrow \text{V}_{\text{sing}} \text{ NP}_{\text{sing}} \\
\text{VP}_{\text{sing}} & \rightarrow \text{V}_{\text{sing}} \text{ NP}_{\text{pl}} \\
\text{S}_{\text{sing}} & \rightarrow \text{NP}_{\text{sing}} \text{ VP}_{\text{sing}} \\
\text{S}_{\text{pl}} & \rightarrow \text{NP}_{\text{pl}} \text{ VP}_{\text{pl}} \\
\end{align*}
\]

\[
\begin{align*}
\text{Det}_{\text{sing}} & \rightarrow \text{this} \\
\text{Det}_{\text{pl}} & \rightarrow \text{those} \\
\text{N}_{\text{sing}} & \rightarrow \text{dog} \\
\text{N}_{\text{pl}} & \rightarrow \text{dogs} \\
\text{V}_{\text{sing}} & \rightarrow \text{eats} \\
\text{V}_{\text{pl}} & \rightarrow \text{eat} \\
\end{align*}
\]
Subcategorization

• Sneeze:  John sneezed
  *John sneezed [the book]_{NP}
• Find:  Please find [a flight to NY]_{NP}
  *Please find
• Give:  Give [me]_{NP}[a cheaper fare]_{NP}
  *Give [with a flight]_{PP}
• Prefer:  I prefer [to leave earlier]_{TO-VP}
  *I prefer [United has a flight]_{S}
• …
Possible CFG Solution

REPLACE:

- $VP \rightarrow V$
- $VP \rightarrow V \ NP$
- $VP \rightarrow V \ NP \ PP$
- ...

WITH:

- $VP \rightarrow V_{\text{Intrans}}$
- $VP \rightarrow V_{\text{Trans}} \ NP$
- $VP \rightarrow V_{\text{Trans+PP}} \ NP \ PP$
- $V_{\text{Intrans}} \rightarrow \text{sneeze}$
- $V_{\text{Trans}} \rightarrow \text{find}$
- $V_{\text{Trans+PP}} \rightarrow \text{give}$
Encoding Number Agreement + Subcats...

- $VP \rightarrow V_{\text{Intrans/sing}}$
- $VP \rightarrow V_{\text{Intrans/pl}}$
- $VP \rightarrow V_{\text{Trans/sing}} \text{ NP}$
- $VP \rightarrow V_{\text{Trans/pl}} \text{ NP}$
- $VP \rightarrow V_{\text{Trans+PP/sing}} \text{ NP PP}$
- $VP \rightarrow V_{\text{Trans+PP/pl}} \text{ NP PP}$
- $V_{\text{Intrans/sing}} \rightarrow \text{sneezes}$
- $V_{\text{Intrans/pl}} \rightarrow \text{sneeze}$
- $V_{\text{Trans/sing}} \rightarrow \text{finds}$
- $V_{\text{Trans/pl}} \rightarrow \text{find}$
- $V_{\text{Trans+PP/sing}} \rightarrow \text{gives}$
- $V_{\text{Trans+PP/pl}} \rightarrow \text{give}$

But what about “I sneeze”, “you sneeze”, “he sneezes”....
Features, informally

View both words and grammar non-terminals as *complex objects*, each of which has a set of *associated property-value pairs* (called *features*) that can be manipulated.

- Det [num = sg] \(\rightarrow\) this
- Det [num = pl] \(\rightarrow\) those
- N [num = sg] \(\rightarrow\) dog
- N [num = pl] \(\rightarrow\) dogs

Then a grammar can contain:

NP \(\rightarrow\) Det N  *but only if* Det [num] = N [num]
Feature Agreement

OK:
NP → Det N  *but only if Det [num] = N [num]*

Better:
NP → Det [num = α] N [num = α]

Best:
NP [num = α] → Det [num = α] N [num = α]  
as well as
S → NP [num = α] VP [num = α]
Features and Feature Structures

• We can encode these properties by associating what are called *feature structures* with grammatical constituents.

• A feature structure is a set of *feature-value pairs* where:
  – *features* are atomic symbols
  – *values* are either atomic symbols or (recursively embedded) feature structures
Example Feature Structures

```
[Number     SG]

[Number   SG]
[Person    3]

[Cat       NP]
[Number   SG]
[Person    3]
```
Bundles of Features

- Feature Values can be feature structures themselves.
- This is useful when certain features commonly co-occur, as number and person.

```
  Cat
  Agreement  NP
  [ Number  SG
     Person  3 ]
```

Feature Structures as DAGs

- CAT
- NP
- AGREEMENT
- NUMBER
- SG
- PERSON
- 3
Reentrant Structure

- Multiple features in a feature structure can share the same value. In this case they share structure, not just have the same value.

- Numerical indices indicate the shared value.

\[
\begin{array}{c}
\text{Cat} \\
\text{Head} \\
\text{Subject}
\end{array}
\quad S
\quad \begin{array}{c}
\text{Agreement} \\
1
\end{array}
\quad \begin{array}{c}
\text{Number SG} \\
\text{Person} \\
3
\end{array}
\quad \begin{array}{c}
\text{Agreement} \\
1
\end{array}
\]
Feature Paths

• It will also be useful to talk about paths through feature structures. As in the paths
• <HEAD AGREEMENT NUMBER>
• <HEAD SUBJECT AGREEMENT NUMBER>
Key operations on feature structures
1. check the compatibility of two structures
2. merge the information in two structures

We can do both with a single operation called *Unification*.

Unifying two feature structures produces a new feature structure that is more specific (has more information) than, or is identical to, each of the input feature structures.
The Unification Operation: \( U \)

- Two feature structures can be unified if the component features that make them up are compatible.

\[
[number \ sg] U [number \ sg] = [number \ sg] \\
[number \ sg] U [number \ pl] = \text{fails!}
\]

- Structures are compatible if they contain no features that are incompatible.
- If so, unification returns the union of all feature/value pairs.
The Unification Operation

\[
[\text{Number sg}] \cup [\text{Number } [\text{ }]] = [\text{Number sg}]
\]

\[
[\text{Number sg}] \cup [\text{Person 3}] = \begin{bmatrix}
\text{Number} \\
\text{Person}
\end{bmatrix}
\begin{bmatrix}
\text{sg} \\
3
\end{bmatrix}
\]
The Unification Operation

\[
[\text{Agreement}[\text{Number sg}] \\
\text{Subject} \quad [\text{Agreement} \quad [\text{Number sg}] \\
\text{U} \\
[\text{Subject} \quad [\text{Agreement} \quad [\text{Person 3}] \\
= \\
[\text{Agreement}[\text{Number sg}] \\
\text{Subject} \quad [\text{Agreement} \quad \text{Number sg} \\
\text{Person 3}]
\]
The Unification Operation

\[ [\text{Head} \quad [\text{Subject} \quad [\text{Agreement} \quad [\text{Number pl} ] ] ] ] ] \]

\[ U \]

\[ \begin{align*}
\text{Cat} & \quad S \\
\text{Head} & \quad [\text{Agreement} \quad [\text{Number sg} \quad \text{Person} \quad 3 ] ] ] \\
\text{Subject} & \quad [\text{Agreement} \quad [1] ] ] \\
\end{align*} \]

= Fail!
Properties of Unification

- **Monotonic**: if some description is true of a feature structure, it will still be true after unifying it with another feature structure.

- **Order independent (commutative)**: Unifying a set of feature structures in any order yields the same result.
Features, Unification, and Grammars

To incorporate all this into grammars:

• Each constituent has a feature-structure associated with it

• Each grammar rule has a (potentially empty) set of unification constraints associated with it.
  
  – The set of unification constraints must be satisfied for the rule to be satisfied.
Unification Constraints

\[ x_0 \rightarrow x_1 \ldots x_n \} \text{Grammar rule} \]

\[ < X_i \text{ feature path } >= \text{ atomic value} \]

\[ < X_i \text{ feature path } >= < X_k \text{ feature path } >= \}

\text{Set of constraints}
Agreement

NP → Det Nominal
< Det AGREEMENT > = < Nominal AGREEMENT >
< NP AGREEMENT > = < Nominal AGREEMENT >

Noun → flight
< Noun AGREEMENT NUMBER > = SG

Noun → flights
< Noun AGREEMENT NUMBER > = PL

Nominal → Noun
< Nominal AGREEMENT > = < Noun AGREEMENT >

Det → this
< Det AGREEMENT NUMBER > = SG
Unification and Parsing

• Assume we’ve augmented our grammar with sets of unification constraints.

• What changes do we need to make to a parser to make use of them?
  1. Build feature structures and associate each with a subtree
  2. Unify feature structures as subtrees are created from smaller subtrees
  3. Block ill-formed constituents
Unification and Earley Parsing

With respect to an Earley-style parser...

• Build feature structures (represented as DAGs) and associate them with states in the chart

• Unify feature structures as states are advanced in the chart

• Block ill-formed states from entering the chart
Building Feature Structures

• Features of most grammatical categories are copied from head child to parent
  – (e.g., from V to VP, Nom to NP, N to Nom)

\[ VP \rightarrow V \, NP \]
\[ < \, VP \, HEAD \, > \, = \, < \, V \, HEAD \, > \]

\[ S \rightarrow NP \, VP \]
\[ < \, NP \, HEAD \, AGREEMENT \, > \, = \, < \, VP \, HEAD \, AGREEMENT \, > \]
\[ < \, S \, HEAD \, > \, = \, < \, VP \, HEAD \, > \]

\[
\begin{align*}
S & \rightarrow [ \text{head} \, 1 ] \\
NP & \rightarrow [ \text{head} \, 1 ] \\
VP & \rightarrow [ \text{head} \, 1 ] \\
\end{align*}
\]

\[
\begin{align*}
[ \text{agreement} \, 2 ] & \\
2 & \\
\end{align*}
\]
Augmenting States with DAGs

• We just add a new field to the representation of the states

\[ S \rightarrow . \text{NP VP, [0,0], Dag} \]
Example

• NP \rightarrow Det \cdot Nominal [0,1], DAG1

\[
\begin{array}{c}
\text{np} \quad [\text{head} 1] \\
\text{det} \quad [\text{head} 1, \text{agreement} 2, [\text{number sg}]] \\
\text{Nominal} \quad [\text{head} 1, \text{agreement} 2] \\
\end{array}
\]

Nominal \rightarrow Noun \cdot, [1,2], DAG2

\[
\begin{array}{c}
\text{Nominal} \quad [\text{head} 1] \\
\text{noun} \quad [\text{head} 1, \text{agreement} 2, [\text{number sg}]] \\
\end{array}
\]
Unifying States and Blocking

• Keep much of the Earley Algorithm the same.
• We want to unify the DAGs of existing states as they are combined as specified by the grammatical constraints.

• Alter COMPLETER – when a new state is created, first make sure the individual DAGs unify. If so, then add the new DAG (resulting from the unification) to the new state.
Unification for Semantics

Figure 9: Feature Structure for 'barbed wire'

Figure 12: Feature Structure for Multimodal Line Creation