Feature Structures and Unification Grammars

11-711 Algorithms for NLP
1 November 2018 – Part II
Linguistic features

• (Linguistic “features” vs. ML “features”.)
• Human languages usually include agreement constraints; in English, e.g., subject/verb
  – I often swim
  – He often swims
  – They often swim
• Could have a separate category for each minor type: N1s, N1p, ..., N3s, N3p, ...
  – Each with its own set of grammar rules!
A day without features...

- NP1s → Det-s N1s
- NP1p → Det-p N1p
- NP3s → Det-s N3s
- NP3p → Det-p N3p
- S1s → NP1s VP1s
- S1p → NP1p VP1p
- S3s → NP3s VP3s
- S3p → NP3p VP3p
Linguistic features

• *Could* have a separate category for each minor type: N1s, N1p, ... , N3s, N3p, ...
  – *Each* with its own set of grammar rules!

• Much better: represent these regularities using independent *features*: number, gender, person, ...

• Features are typically introduced by lexicon; checked and propagated by constraint equations attached to grammar rules
Feature Structures (FSs)

Having multiple orthogonal features with values leads naturally to Feature Structures:

[Det
  [root: a]
  [number: sg ]]

A feature structure’s values can in turn be FSs:

[NP
  [agreement: [[number: sg]
    [person: 3rd]]]]

Feature Path: <NP agreement person>
Adding constraints to CFG rules

• $S \rightarrow NP \ VP$
  $<NP \ number> = <VP \ number>$

• $NP \rightarrow \ Det \ Nominal$
  $<NP \ head> = <Nominal \ head>$
  $<Det \ head \ agree> = <Nominal \ head \ agree>$
FSs from lexicon, constrs. from rules

Lexicon entry:

[Det
  [root: a]
  [number: sg]]

Rule with constraints:

NP \rightarrow \text{Det Nominal}

\langle \text{NP number} \rangle = \langle \text{Det number} \rangle

\langle \text{NP number} \rangle = \langle \text{Nominal number} \rangle

• Combine to get result:

[NP [Det
  [root: a]
  [number: sg]]

[\text{Nominal} [number: sg] ...]

[number: sg]]
Similar issue with VP types

Another place where grammar rules could explode:

Jack laughed

\[
\text{VP} \rightarrow \text{Verb} \quad \text{for many specific verbs}
\]

Jack found a key

\[
\text{VP} \rightarrow \text{Verb} \ \text{NP} \quad \text{for many specific verbs}
\]

Jack gave Sue the paper

\[
\text{VP} \rightarrow \text{Verb} \ \text{NP} \ \text{NP} \quad \text{for many specific verbs}
\]
Verb Subcategorization

Verbs have sets of allowed args. Could have many sets of VP rules. Instead, have a SUBCAT feature, marking sets of allowed arguments:

- +none -- Jack laughed
- +np -- Jack found a key
- +np+np -- Jack gave Sue the paper
- +vp:inf -- Jack wants to fly
- +np+vp:inf -- Jack told the man to go
- +vp:ing -- Jack keeps hoping for the best
- +np+vp:ing -- Jack caught Sam looking at his desk
- +np+vp:base -- Jack watched Sam look at his desk
- +np+pp:to -- Jack gave the key to the man
- +pp:loc -- Jack is at the store
- +np+pp:loc -- Jack put the box in the corner
- +pp:mot -- Jack went to the store
- +np+pp:mot -- Jack took the hat to the party
- +adjp -- Jack is happy
- +np+adjp -- Jack kept the dinner hot
- +sfor -- Jack believed that the world was flat
- +sfor -- Jack hoped for the man to win a prize

50-100 possible frames for English; a single verb can have several. *(Notation from James Allen “Natural Language Understanding”)*
## Frames for “ask”
*(in J+M notation)*

<table>
<thead>
<tr>
<th>Subcat</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Quo</em></td>
<td>asked [{\textit{Quo}} “What was it like?”]</td>
</tr>
<tr>
<td><em>NP</em></td>
<td>asking [{\textit{NP}} a question]</td>
</tr>
<tr>
<td><em>Swh</em></td>
<td>asked [{\textit{Swh}} what trades you’re interested in]</td>
</tr>
<tr>
<td><em>Sto</em></td>
<td>ask [{\textit{Sto}} him to tell you]</td>
</tr>
<tr>
<td><em>PP</em></td>
<td>that means asking [{\textit{PP}} at home]</td>
</tr>
<tr>
<td><em>Vio</em></td>
<td>asked [{\textit{Vio}} to see a girl called Evelyn]</td>
</tr>
<tr>
<td><em>NP Sif</em></td>
<td>asked [{\textit{NP}} him] [{\textit{Sif}} whether he could make]</td>
</tr>
<tr>
<td><em>NP NP</em></td>
<td>asked [{\textit{NP}} myself] [{\textit{NP}} a question]</td>
</tr>
<tr>
<td><em>NP Swh</em></td>
<td>asked [{\textit{NP}} him] [{\textit{Swh}} why he took time off]</td>
</tr>
</tbody>
</table>
Adding transitivity constraint

• S → NP VP
  <NP number> = <VP number>

• NP → Det Nominal
  <NP head> = <Nominal head>
  <Det head agree> = <Nominal head agree>

• VP → Verb NP
  <VP head> = <Verb head>
  <VP head subcat> = +np  (which means transitive)
Applying a verb subcat feature

Lexicon entry:

Verb
[root: found]
[head: find]
[subcat: +np ]]

Rule with constraints:

VP \rightarrow \text{Verb} \quad \text{NP} \\
<\text{VP head}> = <\text{Verb head}> \\
<\text{VP head subcat}> = +\text{np}

• Combine to get result:

VP [Verb
[root: found]
[head: find]
[subcat: +np ]]

NP ...
[head: find [subcat: +np]]]
Relation to LFG constraint notation

- VP $\rightarrow$ Verb $\quad$ NP
  $<$VP head$>$ $=$ $<$Verb head$>$
  $<$VP head subcat$>$ $=$ $+$np

(from JM book is the same as the LFG expression)

- VP $\rightarrow$ Verb $\quad$ NP
  $(\uparrow$ head$) = (\downarrow$ head$)$
  $(\uparrow$ head subcat$) = +$np
Unification

• Merging FSs (and failing if not possible) is called *Unification*

• Simple FS examples:

  [number sg]⊔[number sg] = [number sg]
  [number sg]⊔[number pl]  FAILS
  [number sg]⊔[number []] = [number sg]
  [number sg]⊔[person 3rd] = [number sg, person 3rd]
New kind of “=” sign

• Already had two meanings in programming:
  – “:=“ means “make the left be equal to the right”
  – “==” means “the left and right happen to be equal”

• Now, a third meaning:
  – \( \sqcup \) “=” means “make the left and the right be the same thing (from now on)”
Recap: applying constraints

Lexicon entry:

[Det
  [root: a]
  [number: sg]]

Rule with constraints:

NP → Det Nominal

<NP number> = <Det number>
<NP number> = <Nominal number>

• Combine to get result:

[NP [Det
  [root: a]
  [number: sg]]
  [Nominal [number: sg] ...]
  [number: sg]]
Turning constraint eqns. into FS

Lexicon entry:

[Det
 [root: a]
 [number: sg]]

• Combine to get result:

[NP [Det
 [root: a]
 [number: sg]]
 [Nominal [number: sg]]
 [number: sg]]

Rule with constraints:

NP → Det Nominal

\(<\text{NP number}>=<\text{Det number}>\)
\(<\text{NP number}>=<\text{Nominal number}>\)

becomes:

[NP [Det [number: (1)]]
 [Nominal
  [number: (1)]
  ...
  [number: (1)]]]
Another example

This (oversimplified) rule:

\[ S \to NP \ VP \]
\[ <S \ subject> = NP \]
\[ <S \ agreement> = <S \ subject \ agreement> \]

turns into this DAG:

\[ [S \ [subject \ (1) \]
    \[ \ [agreement \ (2) \ ]\]
    \[ \ [agreement \ (2) \ ]\]
    \[ [NP \ (1) \ ]\]
    \[ [VP \ ]\] \]
Unification example without “EQ“

\[
[\text{agreement} [\text{number sg}], \\
 \text{subject} [\text{agreement} [\text{number sg}]]) \\
\uplus [\text{subject} [\text{agreement} [\text{person 3rd}, \\
 \hspace{1cm} \text{number sg}]]) \\
= [\text{agreement} [\text{number sg}], \\
 \text{subject} [\text{agreement} [\text{person 3rd}, \\
 \hspace{1cm} \text{number sg}]])
\]

- \text{<agreement>} is (initially) equal to \text{<subject agreement>}, but \textbf{not} EQ
- So not equal anymore \textit{after} operation
Unification example with “EQ”

[agreement (1), subject [agreement (1)]]
⊔ [subject [agreement [person 3rd, number sg]]
= [agreement (1),
  subject [agreement (1) [person 3rd, number sg]]]

• <agreement> is <subject agreement> (EQ), so they are equal
• and stay equal, always, in the future
Representing FSs as DAGs

• Taking feature paths seriously
• May be easier to think about than numbered cross-references in text
• [cat NP, agreement [number sg, person 3rd]]
Re-entrant FS as DAGs

- $[\text{cat } S, \text{ head } [\text{agreement (1) [number sg, person 3rd]}, \text{ subject } [\text{agreement (1)}] ] ]$
Seems tricky. Why bother?

• Unification allows the systems that use it to handle many complex phenomena in “simple” elegant ways:
  – There seems to be a dog in the yard.
  – There seem to be dogs in the yard

• Unification makes this work smoothly.
  – Make the Subjects of the clauses EQ:
    <VP subj> = <VP COMP subj>
    [VP [subj: (1)] [COMP [subj: (1)]]]
  – (Ask Lori Levin for LFG details.)
Real Unification-Based Parsing

• \( X_0 \rightarrow X_1 X_2 \)
  \(<X_0 \text{ cat}> = S, <X_1 \text{ cat}> = \text{NP}, <X_2 \text{ cat}> = \text{VP} \)
  \(<X_1 \text{ head agree}> = <X_2 \text{ head agree}> \)
  \(<X_0 \text{ head}> = <X_2 \text{ head}> \)

• \( X_0 \rightarrow X_1 \text{ and } X_2 \)
  \(<X_1 \text{ cat}> = <X_2 \text{ cat}>, <X_0 \text{ cat}> = <X_1 \text{ cat}> \)

• \( X_0 \rightarrow X_1 X_2 \)
  \(<X_1 \text{ orth}> = \text{how}, <X_2 \text{ sem}> = <\text{SCALAR}> \)
Complexity

• Earley modification: “search the chart for states whose DAGs unify with the DAG of the completed state”. Plus a lot of copying.

• Unification parsing is “quite expensive”.
  – NP-Complete in some versions.
  – Early AWB paper on Turing Equivalence(!)

• So maybe too powerful?
  (like GoTo or Call-by-Name?)
  – Add restrictions to make it tractable:
    • Tomita’s Pseudo-unification (Tomabechi too)
    • Gerald Penn work on tractable HPSG: ALE
Formalities: subsumption

• Less specific FS1 \textit{subsumes} more specific FS2
  \( \text{FS1} \sqsubseteq \text{FS2} \) (Inverse is FS2 \textit{extends} FS1)
• Subsumption relation forms a \textit{semilattice},
  at the top: \[
  
  \begin{array}{c}
  [\text{number sg}] \ [\text{person 3}] \ [\text{number pl}] \\
  [\text{number sg, person 3}]
  \end{array}
  
  \]
• Unification defined \textit{wrt} semilattice:
  \( \text{F} \sqcup \text{G} = \text{H} \) s.t. \( \text{F} \sqsubseteq \text{H} \) and \( \text{G} \sqsubseteq \text{H} \)
  
  \( \text{H} \) is the \textit{Most General Unifier} (MGU)
Hierarchical Types

Hierarchical types allow values to unify too (or not):

[Diagram of hierarchical types with nodes labeled 'agr', '1st', '3rd', 'sg', 'pl', '1st-sg', '3rd-sg', '1st-pl', '3rd-pl', '3sg-masc', '3sg-fem', '3sg-neut']
Hierarchical subcat frames

Many verbs share subcat frames, some with more arguments specified than others: