Feature structures for parsing

L445 / L545

Spring 2017

(With thanks to Detmar Meurers)
The issue

- So far: parsing strategies discussed with atomic categories.
  - Example: $S \rightarrow NP \ VP$

- How about the compound terms used as categories?
  - Example: $S \rightarrow NP(\text{Per},\text{Num}) \ VP(\text{Per},\text{Num})$
Ideas for parsing with non-atomic categories

Three options for parsing with grammars using non-atomic categories:

1. Expand the grammar into a CFG with atomic categories
2. Parse using an atomic CFG backbone with reduced information
3. Incorporate special mechanisms into the parser
Idea 1
Transform into CFG with atomic categories

If only compound terms without variables are used, the rules correspond to rules with atomic categories

Example:
- \( S \rightarrow \text{NP}(1, \text{sg}) \ \text{VP}(1, \text{sg}) \)
- \( S \rightarrow \text{NP}_{1\text{sg}} \ \text{VP}_{1\text{sg}} \)
More on Idea 1

If there are a finite set of possible values for the variables occurring in the compound terms, one can replace a rule with the instances for all possible instantiations of variables.

Example:

- $S \rightarrow \text{NP(Per,Num) VP(Per,Num)}$
- $S \rightarrow \text{NP(1,sg) VP(1,sg)}$
  - $S \rightarrow \text{NP(2,sg) VP(2,sg)}$
  - $S \rightarrow \text{NP(3,sg) VP(3,sg)}$
  - $S \rightarrow \text{NP(1,pl) VP(1,pl)}$
  - $S \rightarrow \text{NP(2,pl) VP(2,pl)}$
  - $S \rightarrow \text{NP(3,pl) VP(3,pl)}$
Evaluation of Idea 1

- Leads to a potentially huge set of rules
  - number of categories grows exponentially w.r.t. the number of features
  - grammar size relevant for time & space efficiency of parsing
- Doesn’t allow for variables, i.e., misses generalizations
Idea 2
Parse using atomic CFG backbone (reduced info)

- Idea:
  - parse using a property defined for all categories
  - use other properties to filter solutions from set of parses

- Downside:
  - parsing with partial information can significantly enlarge the search space
Idea 3
Incorporate special mechanism into parser

▶ How two categories are combined has to be replaced by unification.
▶ Every active and inactive edge in a chart may be used for different uses.
  ▶ So, for each time an edge is used, a new copy needs to be made.
▶ Two effectiveness issues:
  ▶ Use subsumption test to ensure general enough predictions
  ▶ Use restriction to prevent prediction loops
▶ Two efficiency issues (not dealt with here):
  ▶ intelligent indexing of edges in chart
  ▶ packing of similar edges in chart (cf., Tomita parser)
Exploring Unification

Taking idea 3, here’s where we’re going:

- Feature Structures
- Unification
- Unification-Based Grammars
- Chart Parsing with Unification-Based Grammars (next slide set)
Feature structures

- To address the problem of adding agreement to CFGs, we need features, e.g., a way to say:

\[
\begin{bmatrix}
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\]

- A structure like this allows us to state properties, e.g., about a noun phrase

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\]

- Each **feature** (e.g., NUMBER) is paired with a value (e.g., sg)
  - A bundle of feature-value pairs can be put into an attribute-value matrix (AVM)
Constraints

Idea: each rule of the grammar is a complex bundle of constraints

- $S \rightarrow NP \ VP$ means that an $S$ object is constrained to be composed of an $NP$ followed by a $VP$

Features allow one to add more constraints

- $S \rightarrow NP \ VP$ only if number of $NP = $ number of $VP$
  - Constraint 1: $S \rightarrow NP \ VP$
  - Constraint 2: $NP\ num = VP\ num$

Often referred to as constraint-based processing
Feature paths

Values can be atomic (e.g. *sg* or *NP* or 3):

\[
\begin{bmatrix}
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\]

Or they can be complex, allowing for **feature paths**:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{AGREEMENT} & \begin{bmatrix}
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\end{bmatrix}
\]

The value of the path \([\text{AGREEMENT}|\text{NUMBER}]\) is *sg*

- Complex values allow for more expressivity than a CFG, i.e., can represent more linguistic phenomena
Feature structures as graphs

- Feature structures are directed acyclic graphs (DAGs)
- The feature structure represented by the attribute-value matrix (AVM):

\[
\begin{bmatrix}
\text{CAT} & NP \\
\text{AGR} & \begin{bmatrix}
\text{NUM} & \text{sg} \\
\text{PER} & 3
\end{bmatrix}
\end{bmatrix}
\]

is really the graph:
Reentrancy (structure sharing)

Feature structures embedded in feature structures can share the same values

- Two features share precisely the same object as their value
  - We’ll indicate this with a tag like $1$

- The agreement features of both the matrix sentence & embedded subject are identical (same object)
  - This is referred to as **reentrancy**
What structure-sharing is not

- This is structure-sharing (changing value in one place changes both):

```
[HEAD
  [AGR 1]
  [SUBJ AGR 1]]
```

- This is not (changing one value doesn’t change other):

```
[HEAD
  [AGR]
  [SUBJ AGR]
  [NUM sg]
  [PER 3]]
```

[4/4]
Unification

We’ll often want to merge feature structures

- **Unification** (⊔) = a basic operation to merge two feature structures into a resultant feature structure (FS)

The two feature structures must be compatible, i.e., have no values that conflict

- Identical FSs:
  \[
  \left[ \text{NUMBER} \ sg \right] \sqcup \left[ \text{NUMBER} \ sg \right] = \left[ \text{NUMBER} \ sg \right]
  \]

- Conflicting FSs:
  \[
  \left[ \text{NUMBER} \ sg \right] \sqcup \left[ \text{NUMBER} \ pl \right] = \text{Fail}
  \]

- Merging with an unspecified FS:
  \[
  \left[ \text{NUMBER} \ sg \right] \sqcup \left[ \right] = \left[ \text{NUMBER} \ sg \right]
  \]
Merging FSs with different features specified:

\[
\begin{bmatrix}
\text{NUMBER} & \text{sg}
\end{bmatrix} \sqcup \begin{bmatrix}
\text{PERSON} & 3
\end{bmatrix} = \begin{bmatrix}
\text{NUMBER} & \text{sg}
\end{bmatrix}
\]

More examples:

\[
\begin{bmatrix}
\text{cat} & \text{NP}
\end{bmatrix} \sqcup \begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix} = \begin{bmatrix}
\text{cat} & \text{NP}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix} \sqcup \begin{bmatrix}
\text{subj} & \begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix} = \begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix} \sqcup \begin{bmatrix}
\text{subj} & \begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix} = \begin{bmatrix}
\text{agr} & \begin{bmatrix}
\text{num} & \text{sg}
\end{bmatrix}
\end{bmatrix}
\]
Unification with Reentrancies

- Remember that structure-sharing means they are the same object:

  $\begin{array}{c}
  \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array} \square \begin{array}{c}
  \text{SUBJ } \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array} = \begin{array}{c}
  \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array} \square \begin{array}{c}
  \text{SUBJ } \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array}$

- When unification takes place, shared values are copied over:

  $\begin{array}{c}
  \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array} = \begin{array}{c}
  \text{SUBJ } \text{AGR } 1 \\
  \text{PER } 3 \\
  \text{NUM } sg
  \end{array}$
Unification with Reentrancies (cont.)

- And remember that having similar values is not the same as structure-sharing:
  \[
  \begin{array}{c}
  \text{AGR } \left[ \begin{array}{c} \text{NUM} \\ \text{sg} \end{array} \right] \\
  \text{SUBJ } \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{NUM} \\ \text{sg} \end{array} \right] \end{array} \right] \cup \left[ \begin{array}{c} \text{SUBJ } \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{PER} \\ 3 \end{array} \right] \end{array} \right] \end{array} \right] = \\
  \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{NUM} \\ \text{sg} \end{array} \right] \end{array} \right] \\
  \text{SUBJ } \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{PER} \\ 3 \end{array} \right] \end{array} \right] \\
  \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{NUM} \\ \text{sg} \end{array} \right] \end{array} \right]
  \end{array}
  \right]
  \]

- With structure-sharing, the values must be compatible everywhere it is specified
  \[
  \begin{array}{c}
  \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{NUM} \\ \text{sg} \end{array} \right] \end{array} \right] \\
  \text{SUBJ } \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{num} \\ \text{sg} \end{array} \right] \end{array} \right] \cup \left[ \begin{array}{c} \text{SUBJ } \left[ \begin{array}{c} \text{AGR } \left[ \begin{array}{c} \text{PER} \\ 3 \end{array} \right] \end{array} \right] \end{array} \right] = \text{Fail}
  \end{array}
  \]

Subsumption

A more general feature structure (less values specified) subsumes a more specific feature structure

(1) \[
\begin{array}{c}
\text{NUM} \\
\text{sg}
\end{array}
\]

(2) \[
\begin{array}{c}
\text{PER} \\
3
\end{array}
\]

(3) \[
\begin{array}{c}
\text{NUM} \\
\text{sg} \\
\text{PER} \\
3
\end{array}
\]

The following subsumption relations hold:

- (1) subsumes (3)
- (2) subsumes (3)
- (1) does not subsume (2), and (2) does not subsume (1)
Implementing Unification

How do we implement a check on unification?

- **Goal:** given feature structures $F_1$ and $F_2$, return $F$, the unification of $F_1$ and $F_2$

Unification is a recursive operation:

- If a feature has an atomic value, see if the other FS has that feature with the same value
  - $[F \ a]$ unifies with $[], [F]$, and $[F \ a]$

- If a feature has a complex value, follow the paths to see if they’re compatible & have the same values at bottom
  - To see whether $[F \ G_1]$ unifies with $[F \ G_2]$, inspect $G_1$ and $G_2$

- To avoid cycles, do an **occur check** to see if we’ve seen a FS before or not
The need for unification

Assume:

- a verb selecting for a 3rd person singular noun subject
- a subject which is 2nd person singular

What the verb specifies for the subject has to be able to unify with what the subject is

- In this case, unification will fail: person doesn’t unify
Unification-based grammars
Grammars with feature structures

One way to encode features is to augment a CFG skeleton with feature structure path equations

- CFG skeleton
  \[ S \rightarrow NP \ VP \]
- Path equations
  \[ (NP \text{ agreement}) = (VP \text{ agreement}) \]

Conditions:

1. There can be zero or more path equations for each rule skeleton \( \rightarrow \) no longer atomic
2. When a path equation references constituents, they can only be constituents from the CFG rule
We’ll look at 3 different phenomena that feature-based, or unification-based, grammars capture fairly succinctly:

1. Agreement
2. Subcategorization
3. Long-distance dependencies

You can find our more details by exploring:

- Lexical-Functional Grammar (LFG)
- Head-driven Phrase Structure Grammar (HPSG)

(Both are taught in Alternative Syntactic Theories (L614))
1) Agreement in Feature-based Grammars

One way to capture agreement rules:

\[
\begin{align*}
S & \rightarrow \quad \text{NP VP} \\
& \quad (S\ head) = (VP\ head) \\
& \quad (\text{NP head agr}) = (VP\ head\ agr) \\
VP & \rightarrow \quad \text{V NP} \\
& \quad (VP\ head) = (V\ head) \\
NP & \rightarrow \quad \text{D Nom(inal)} \\
& \quad (NP\ head) = (Nom\ head) \\
& \quad ( Det\ head\ agr) = (Nom\ head\ agr) \\
Nom & \rightarrow \quad \text{Noun} \\
& \quad (Nom\ head) = (Noun\ head) \\
Noun & \rightarrow \quad \text{flights} \\
& \quad (Noun\ head\ agr\ num) = pl
\end{align*}
\]
Percolating Agreement Features

```
S
  [HEAD 4]
  ___/
  NP   VP
  [HEAD 3 [AGR 1]]  [HEAD 4 [AGR 1]]
     ___/
    Det   Nom   V   NP
    [HEAD AGR 1]  [HEAD 3 [AGR 1]]  [HEAD 4]  [___ ___]
       ___/
      Noun   V
      [HEAD 3 [AGR 1 [NUM pl]]]
           ___/
          flights
```
Head features in the grammar

- Important concept from the previous rules: heads of grammar rules share properties with their mothers
  \[ VP \rightarrow V \ NP \]
  \[ (VP \text{ HEAD}) = (V \text{ HEAD}) \]
  - Knowing the head will tell you about the whole phrase
    - This is important for many parsing techniques
2) Subcategorization

We could specify subcategorization like so:

\[
\begin{align*}
    \text{VP} & \rightarrow V \\
    (V_{\text{SUBCAT}}) &= \text{intrans} \\
    \text{VP} & \rightarrow V \text{ NP} \\
    (V_{\text{SUBCAT}}) &= \text{trans} \\
    \text{VP} & \rightarrow V \text{ NP NP} \\
    (V_{\text{SUBCAT}}) &= \text{ditrans}
\end{align*}
\]

But values like \textit{intrans} do not correspond to anything that the rules actually look like.

- To make \textit{subcat} better match the rules, we can make its value a list of a verb’s arguments, e.g. \textlangle\textit{NP},\textit{PP}\textrangle
Subcategorization rules

\[
\begin{align*}
\text{VP} & \rightarrow \ V \ NP \ PP \\
(\text{VP HEAD}) & = (V \text{ HEAD}) \\
(V \text{ SUBCAT}) & = <\text{NP}, \text{NP}, \text{PP}>
\end{align*}
\]

\[
\begin{align*}
\text{V} & \rightarrow \text{ leaves} \\
(V \text{ HEAD AGR NUM}) & = \text{sg} \\
(V \text{ SUBCAT}) & = <\text{NP}, \text{NP}, \text{PP}>
\end{align*}
\]

More formal way to specify lists:

\[
<\text{NP,PP}> \text{ is equivalent to: } \begin{bmatrix}
\text{FIRST} & \text{NP} \\
\text{REST} & \begin{bmatrix}
\text{FIRST} & \text{PP} \\
\text{REST} & \langle \rangle
\end{bmatrix}
\end{bmatrix}
\]
Subcategorization Example

Feature structures

Ideas

Feature structures

Unification

Unification-based grammars

Agreement

Subcategorization

Long-distance dependencies
Handling Subcategorization

How do we ensure that an object’s subcategorization list corresponds to what we see in the actual tree?

▶ We need a subcategorization principle

As a tree is built, items are checked off of the subcat list

▶ The subcat list must be empty at the top of a tree
▶ If we had used the rule VP $\rightarrow$ V NP, we would have been left with subcat $<$NP,PP$>$
▶ The rule VP $\rightarrow$ V NP PP PP would have specified something missing from the subcat list
3) Long-distance dependencies

Long-distance dependencies are often also called “movement” phenomena

- Topicalization: *John she likes ___ .*
- *Wh*-questions: *Who does she like ___ ?*

To capture this without movement, one can instead pass features along the tree

- Bottom: introduce a ‘trace’
- Middle: pass the trace
- Top: Unify the features of the trace with some real word (e.g., *John, Who*)

We’ll use a *gap* feature for this
Handling long-distance dependencies

TOP:
(fill gap)  S →  \textit{wh-word be-cop NP}  
\((\text{NP GAP}) = (\textit{wh-word HEAD})\)

MIDDLE:
(pass gap)  NP →  D Nom  
\((\text{NP GAP}) = (\text{Nom GAP})\)  
Nom →  Nom RelCl  
\((\text{Nom GAP}) = (\text{RelCl GAP})\)  
RelCl →  RelPro NP VP  
\((\text{RelCl GAP}) = (\text{VP GAP})\)

BOTTOM:
(identify gap)  VP →  V  
\((\text{VP GAP}) \in (\text{V SUBCAT})\)

(Actually, we want a more general principle to introduce GAP features, but this will do for now ...
Handling long-distance dependencies

S

wh-word be-cop NP

[HEAD 1] [GAP 1]

What is D Nom

a Nom RelCl

flight RelPro NP VP

[HEAD 1] [GAP 1]

that you V

[SUBCAT ⟨NP, 1⟩]

have
What’s going on

- Traces, or gaps, are allowed as items from subcat lists
- When a trace is introduced, make sure it gets checked off subcat, so the subcat principle is satisfied
- Alternate way: the gap value of a mother of a rule is the union of the daughter’s gap values
  - So, we wouldn’t have to write separate rules for RelClause, Nom, NP, etc.
  - When a subcat list is empty & an item matches something in the gap set, remove it from gap