Feature structures for parsing

L545
Spring 2013
(With thanks to Detmar Meurers)

The issue

▶ So far: parsing strategies discussed with atomic categories.
  ▶ Example: \( S \to NP \ VP \)
  ▶ How about the compound terms used as categories?
  ▶ Example: \( S \to NP(\text{Per,Num}) \ VP(\text{Per,Num}) \)

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

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:
- \( S \to NP(\text{Per,Num}) \ VP(\text{Per,Num}) \)
- \( S \to NP(\text{1.sg}) \ VP(\text{1.sg}) \)
- \( S \to NP(\text{2.sg}) \ VP(\text{2.sg}) \)
- \( S \to NP(\text{3.sg}) \ VP(\text{3.sg}) \)
- \( S \to NP(\text{1.pl}) \ VP(\text{1.pl}) \)
- \( S \to NP(\text{2.pl}) \ VP(\text{2.pl}) \)
- \( S \to NP(\text{3.pl}) \ VP(\text{3.pl}) \)

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
Feature structures for parsing

Ideas

Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

Exploring Unification

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

- Feature Structures
- Unification
- Unification-Based Grammars
- Chart Parsing with Unification-Based Grammars
  (next time)

Constraints

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

- S → 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 → NP VP only if number of NP = number of VP
  - Constraint 1: S → 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):

```
  NUMBER   sg
  PERSON   3
```

Or they can be complex, allowing for feature paths:

```
  CAT      NP
  AGREEMENT   NUMBER   sg
  PERSON   3
```

The value of the path \texttt{AGREEMENT\{NUMBER\}} is \texttt{sg}

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

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

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

is really the graph:

```
Feature structures
Ideas
Feature structures

\[ S \]

\[ \text{HEAD} \quad \text{AGR} \quad \text{NUM sg} \]

\[ \text{PER} \quad 3 \]

\[ \text{SUBJ} \quad \text{AGR} \quad \text{NUM sg} \]
```

What structure-sharing is not

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

```
\[ \text{HEAD} \quad \text{AGR} \quad \text{NUM sg} \]
```

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

```
\[ \text{HEAD} \quad \text{AGR} \quad \text{NUM sg} \]
```

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 \[ \text{[ ]} \]

```
\[ \text{CAT} \quad \text{S} \]
```

- The agreement features of both the matrix sentence & embedded subject are identical (same object)
  - This is referred to as reentrancy

Unification

We’ll often want to merge feature structures

- **Unification** (\( \cup \)) = 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:
  - \( \text{NUMBER sg} \cup \text{NUMBER sg} = \text{NUMBER sg} \)

- Conflicting FSs:
  - \( \text{NUMBER sg} \cup \text{NUMBER pl} = \text{Fail} \)

- Merging with an unspecified FS:
  - \( \text{NUMBER sg} \cup \text{[ ]} = \text{NUMBER sg} \)

Unification (cont.)

- Merging FSs with different features specified:
  - \( \text{NUMBER sg} \cup \text{PERSON 3} = \text{NUMBER sg} \)

- More examples:
  - \( \text{CAT} \cup \text{NP} \)
  - \( \text{AGR} \cup \text{NUM sg} \)
  - \( \text{SUBJ} \cup \text{AGR} \)

- When unification takes place, shared values are copied over:
  - \( \text{AGR} \cup \text{PER 3} \)
  - \( \text{SUBJ} \cup \text{AGR} \)

Unification with Reentrancies

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

```
\[ \text{AGR} \quad \text{NUM sg} \]
```

```
\[ \text{SUBJ} \quad \text{AGR} \quad \text{PER 3} \]
```

```
\[ \text{AGR} \quad \text{PER 3} \]
```
Unification with Reentrancies (cont.)

- And remember that having similar values is not the same as structure-sharing:
  \[
  \begin{align*}
  & \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \quad \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \\
  & \text{SUBJ} \begin{cases} \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \end{cases} \quad \text{SUBJ} \begin{cases} \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \end{cases}
  \end{align*}
  \]
  With structure-sharing, the values must be compatible everywhere it is specified:
  \[
  \begin{align*}
  & \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \quad \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \\
  & \text{SUBJ} \begin{cases} \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \end{cases} \quad \text{SUBJ} \begin{cases} \text{AGR} \begin{cases} \text{NUM} & \text{sg} \end{cases} \end{cases}
  \end{align*}
  \]
  \[
  = \text{Fail}
  \]

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.
  \[
  [F \ a] \text{ unifies with } [F] \text{ and } [F \ a]
  \]

- 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

Subsumption

- A more general feature structure (less values specified) **subsumes** a more specific feature structure:
  \[
  \begin{align*}
  & (1) \begin{cases} \text{NUM} & \text{sg} \end{cases} \\
  & (2) \begin{cases} \text{PER} & \text{3} \end{cases} \\
  & (3) \begin{cases} \text{NUM} & \text{sg} \end{cases}
  \end{align*}
  \]

The following subsumption relations hold:

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

Handling Linguistic Phenomena

- 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
1) Agreement in Feature-based Grammars

One way to capture agreement rules:

\[ S \rightarrow \text{NP VP} \]
\[ (\text{S head}) = (\text{VP head}) \]
\[ (\text{NP head agr}) = (\text{VP head agr}) \]
\[ \text{VP} \rightarrow \text{V NP} \]
\[ (\text{VP head}) = (\text{V head}) \]
\[ \text{NP} \rightarrow \text{D Nom(inal)} \]
\[ (\text{NP head}) = (\text{Nom head}) \]
\[ (\text{Det head agr}) = (\text{Nom head agr}) \]
\[ \text{Nom} \rightarrow \text{Noun} \]
\[ (\text{Nom head}) = (\text{Noun head}) \]
\[ \text{Noun} \rightarrow \text{flights} \]
\[ (\text{Noun head agr num}) = \text{pl} \]

Head features in the grammar

- Important concept from the previous rules: heads of grammar rules share properties with their mothers
  - \[ \text{VP} \rightarrow \text{V NP} \]
    - (\text{VP head}) = (\text{V 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:

\[ \text{VP} \rightarrow \text{V NP} \]
\[ (\text{V subcat}) = \text{intrans} \]
\[ \text{VP} \rightarrow \text{V NP} \]
\[ (\text{V subcat}) = \text{trans} \]
\[ \text{VP} \rightarrow \text{V NP} \]
\[ (\text{V subcat}) = \text{dtrans} \]

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

- To make \text{subcat} better match the rules, we can make its value a list of a verb's arguments, e.g. \text{<NP,PP>}
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 → V NP, we would have been left with subcat <NP,PP>
> The rule VP → V NP PP PP would have specified something missing from the subcat list

Handling long-distance dependencies

TOP:
(fill gap) S → wh-word be-cop NP (NP GAP) = (wh-word HEAD)
MIDDLE:
(pass gap) NP → D Nom (Nom GAP) = (Nom GAP)
Nom → Nom RelCl (Nom GAP) = (RelCl GAP)
RelCl → RelPro NP VP (RelCl GAP) = (VP GAP)
BOTTOM:
(identify gap) VP → V (VP GAP) = (V subcat second)

(Actually, we want a more general principle to introduce GAP features, but this will do for now ...)

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

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

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