Feature structures for parsing

Ideas

Feature structures for parsing

Unification

Unification-based grammars

Agreement

Subcategorization

Long-distance dependencies

The issue

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

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:
  ▶ $S \rightarrow NP\text{(1,sg)} \ VP\text{(1,sg)}$
  ▶ $S \rightarrow NP\text{$_{1\text{sg}}$} \ VP\text{$_{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:
  ▶ $S \rightarrow NP\text{(Per,Num)} \ VP\text{(Per,Num)}$
  ▶ $S \rightarrow NP\text{(1,sg)} \ VP\text{(1,sg)}$
    $S \rightarrow NP\text{(2,sg)} \ VP\text{(2,sg)}$
    $S \rightarrow NP\text{(3,sg)} \ VP\text{(3,sg)}$
    $S \rightarrow NP\text{(1,pl)} \ VP\text{(1,pl)}$
    $S \rightarrow NP\text{(2,pl)} \ VP\text{(2,pl)}$
    $S \rightarrow NP\text{(3,pl)} \ VP\text{(3,pl)}$

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

(With thanks to Detmar Meurers)
Feature structures

### 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

```
   number  sg
  person 3
```

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

```
   cat
   number  sg
  person 3
```

- 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)

### 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 [AGREEMENT][NUMBER] is sg

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

### 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 structures as graphs

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

```
  CAT NP
    AGR  [NUM sg]
      PER 3
```

is really the graph:

```
  NP
  \cat
  AGR
  NUM
  3
```

What structure-sharing is not

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

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

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

```
  HEAD
    AGR  [NUM sg]
      PER 3
    SUBJ
      AGR  [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 [1]

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

- 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:
  \[\text{NUMBER} \text{sg} \cup \text{NUMBER} \text{sg} = \text{NUMBER} \text{sg}\]

- Conflicting FSs:
  \[\text{NUMBER} \text{sg} \cup \text{NUMBER} \text{pl} = \text{Fail}\]

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

Unification (cont.)

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

More examples:

```
  \cat NP \cup \text{AGR} \text{NUM sg} = \text{AGR} \text{NUM sg}
  \text{AGR} \text{NUM sg} \cup \text{SUBJ} \text{AGR} \text{NUM sg} = \text{SUBJ} \text{AGR} \text{NUM sg}
  \text{AGR} \text{NUM sg} \cup \text{SUBJ} \text{AGR} \text{NUM sg} = \text{SUBJ} \text{AGR} \text{NUM sg}
```

Unification with Reentrancies

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

```
  AGR  [NUM sg] \cup \text{SUBJ} \text{AGR} \text{PER 3} = \text{SUBJ} \text{AGR} \text{PER 3}
```

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

```
  AGR  [NUM sg] \cup \text{SUBJ} \text{AGR} \text{PER 3} = \text{SUBJ} \text{AGR} \text{PER 3}
  \text{AGR} \text{PER 3} \cup \text{NUM sg} = \text{AGR} \text{PER 3} \cup \text{NUM sg}
  \text{SUBJ} \text{PER 3} \cup \text{NUM sg} = \text{SUBJ} \text{PER 3} \cup \text{NUM sg}
```
If a feature has a complex value, follow the paths to see a verb selecting for a 3rd person singular noun subject (Lexical-Functional Grammar (LFG) and Head-driven Phrase Structure Grammar (HPSG)).

If a feature has an atomic value, see if the other FS has:

1. With structure-sharing, the values must be compatible and remember that having similar values is not the same as structure-sharing:
   \[
   \begin{align*}
   \text{AGR} & : \text{NUM SG} \\
   \text{SUBJ} & : \text{AGR} \text{ NUM SG} \\
   \text{SUBJ} & : \text{num 3} \\
   \text{AGR} & : \text{num 3} \\
   \text{SUBJ} & : \text{subj} \\
   \text{AGR} & : \text{subj} \\
   \text{PER 3} & : \text{subj} \\
   \text{num 3} & : \text{subj} \\
   \text{num 3} & : \text{subj} \\
   \end{align*}
   \]
   With structure-sharing, the values must be compatible everywhere it is specified.
   \[
   \begin{align*}
   \text{AGR} & : \text{num 3} \\
   \text{SUBJ} & : \text{AGR} \text{ num 3} \\
   \text{SUBJ} & : \text{num pl} \\
   \text{AGR} & : \text{num pl} \\
   \text{SUBJ} & : \text{subj} \\
   \text{AGR} & : \text{subj} \\
   \text{PER 3} & : \text{subj} \\
   \text{num pl} & : \text{subj} \\
   \end{align*}
   \]

2. The unification will fail: person doesn't unify.

Implementing Unification

How do we implement a check on unification?
- **Goal:** given feature structures F1 and F2, return F, the unification of F1 and F2.

1. Unification is a recursive operation:
   - If a feature has an atomic value, see if the other FS has that feature with the same value.
     \[
     \text{F a} \text{ unifies with } \text{F a}
     \]
   - 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 \( \text{F G1} \text{ unifies with } \text{F G2} \), inspect \( \text{G1} \) and \( \text{G2} \).
   - To avoid cycles, do an **occur check** to see if we’ve seen a FS before or not.

Subsumption

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

1. \( \text{NUM SG} \)
2. \( \text{PER 3} \)
3. \( \text{NUM SG} \text{ PER 3} \)

The following subsumption relations hold:
- (1) subsumes (3)
- (2) subsumes (3)
- (1) does not subsume (2), and (2) does not subsume (1)

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

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

Handing 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

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:

\[
S \rightarrow \text{NP VP} \\
(V \text{head}) = (V \text{head}) \\
(NP \text{head agr}) = (V \text{head agr})
\]

\[
VP \rightarrow V \text{NP} \\
(V \text{head}) = (V \text{head})
\]

\[
NP \rightarrow D \text{Nom(inal)} \\
(NP \text{head}) = (\text{Nom head}) \\
(Do\text{t head agr}) = (\text{Nom head agr})
\]

Nom \rightarrow Noun \\
(Nom head) = (Noun head)

Noun \rightarrow \text{flights} \\
(Noun head agr num) = pl

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:

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

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

\[
VP \rightarrow V \text{NP NP} \\
(V \text{subcat}) = \text{ditrans}
\]

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

To make subcat better match the rules, we can make its value a list of a verb's arguments, e.g. \langle\text{NP, PP}\rangle

Subcategorization rules

\[
VP \rightarrow V \text{NP PP} \\
(V \text{head}) = (V \text{head}) \\
(V \text{subcat}) = \langle\text{NP, NP, PP}\rangle
\]

\[
V \rightarrow \text{leaves} \\
(V \text{head agr num}) = \text{sg} \\
(V \text{subcat}) = \langle\text{NP, NP, PP}\rangle
\]

More formal way to specify lists:

\[
\langle\text{NP, PP}\rangle \text{ is equivalent to:} \\
\begin{array}{c}
\text{FIRST} \\
\text{NP} \\
\text{REST} \\
\text{FIRST} \\
\text{PP} \\
\text{REST} \\
\text{()}
\end{array}
\]
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
(NP 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)

(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.

TOP:
(fill gap) S

MIDDLE:
(np gap) NP

BOTTOM:
(identify gap) VP