Feature structures for parsing

L545

Spring 2013

(With thanks to Detmar Meurers)

Feature structures for parsing
Ideas
Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

The issue

- So far: parsing strategies discussed with atomic categories.
  - Example: S → NP VP
- How about the compound terms used as categories?
  - Example: S → NP(Per,Num) VP(Per,Num)

Feature structures for parsing
Ideas
Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

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

Feature structures for parsing
Ideas
Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

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 → NP(1,sg) VP(1,sg)
- S → NP_{1sg} VP_{1sg}

Feature structures for parsing
Ideas
Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

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 → NP(Per,Num) VP(Per,Num)
- S → NP(1,sg) VP(1,sg)
  S → NP(2,sg) VP(2,sg)
  S → NP(3,sg) VP(3,sg)
  S → NP(1,pl) VP(1,pl)
  S → NP(2,pl) VP(2,pl)
  S → NP(3,pl) VP(3,pl)

Feature structures for parsing
Ideas
Feature structures
Unification
Unification-based grammars
Agreement
Subcategorization
Long-distance dependencies

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

Feature structures for parsing
Ideas | Feature structures | Unification | Unification-based grammars | Agreement | Subcategorization | Long-distance dependencies

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)

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)

Feature structures for parsing
Ideas | Feature structures | Unification | Unification-based grammars | Agreement | Subcategorization | Long-distance dependencies

Feature structures

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

$$\begin{bmatrix} \textsc{number} & \textit{sg} \\ \textsc{person} & \textit{3} \end{bmatrix}$$

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

$$\begin{bmatrix} \textsc{cat} & \textit{NP} \\ \textsc{number} & \textit{sg} \\ \textsc{person} & \textit{3} \end{bmatrix}$$

- 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)

Feature structures for parsing
Ideas | Feature structures | Unification | Unification-based grammars | Agreement | Subcategorization | Long-distance dependencies

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 for parsing
Ideas | Feature structures | Unification | Unification-based grammars | Agreement | Subcategorization | Long-distance dependencies

Feature paths

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

$$\begin{bmatrix} \textsc{number} & \textit{sg} \\ \textsc{person} & \textit{3} \end{bmatrix}$$

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

$$\begin{bmatrix} \textsc{cat} & \textit{NP} \\ \textsc{agreement} & \begin{bmatrix} \textsc{number} & \textit{sg} \\ \textsc{person} & \textit{3} \end{bmatrix} \end{bmatrix}$$

The value of the path $[\textsc{agreement}|\textsc{number}]$ is *sg*

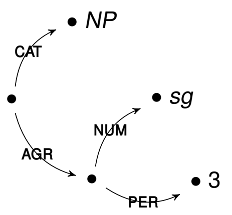
- Complex values allow for more expressivity than a CFG, i.e., can represent more linguistic phenomena

Feature structures as graphs

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

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

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 $\boxed{1}$

$$\begin{bmatrix} \text{CAT} & S \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- 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):

$$\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

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

$$\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Unification

We'll often want to merge feature structures

- **Unification** (**$\sqcup$**) = 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:
  $\begin{bmatrix} \text{NUMBER} & sg \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER} & sg \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & sg \end{bmatrix}$
- Conflicting FSs:
  $\begin{bmatrix} \text{NUMBER} & sg \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER} & pl \end{bmatrix} = Fail$
- Merging with an unspecified FS:
  $\begin{bmatrix} \text{NUMBER} & sg \end{bmatrix} \sqcup [\,] = \begin{bmatrix} \text{NUMBER} & sg \end{bmatrix}$

Unification (cont.)

- Merging FSs with different features specified:
  $\begin{bmatrix} \text{NUMBER} & sg \end{bmatrix} \sqcup \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3 \end{bmatrix}$
- More examples:
  $\begin{bmatrix} \text{CAT} & NP \end{bmatrix} \sqcup \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{CAT} & NP \\ \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix}$

  $\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix} =$

  $\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix}$

Unification with Reentrancies

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

$$\begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PER} & 3 \\ \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

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

$$\begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PER} & 3 \\ \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix} =$$

$$\begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{PER} & 3 \\ \text{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Unification with Reentrancies (cont.)

- And remember that having similar values is not the same as structure-sharing:

$$\begin{bmatrix} \text{AGR} & \begin{bmatrix}\text{NUM} & sg\end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix}\text{AGR} & \begin{bmatrix}\text{NUM} & sg\end{bmatrix}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix}\text{SUBJ} & \begin{bmatrix}\text{AGR} & \begin{bmatrix}\text{PER} & 3 \\ \text{NUM} & sg\end{bmatrix}\end{bmatrix}\end{bmatrix} =$$

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

- With structure-sharing, the values must be compatible everywhere it is specified

$$\begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix}\text{NUM} & sg \\ \text{PER} & 3\end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix}\text{AGR} & \boxed{1}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix} \text{AGR} & \begin{bmatrix}\text{NUM} & sg \\ \text{PER} & 3\end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix}\text{AGR} & \begin{bmatrix}\text{NUM} & pl \\ \text{PER} & 3\end{bmatrix}\end{bmatrix}\end{bmatrix} = \textit{Fail}$$

Subsumption

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

(1) $\begin{bmatrix}\text{NUM} & sg\end{bmatrix}$

(2) $\begin{bmatrix}\text{PER} & 3\end{bmatrix}$

(3) $\begin{bmatrix}\text{NUM} & sg \\ \text{PER} & 3\end{bmatrix}$

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 $F1$ and $F2$, return F, the unification of $F1$ and $F2$

Unification is a recursive operation:

- If a feature has an atomic value, see if the other FS has that feature with the same value
  - $\begin{bmatrix}\text{F} & a\end{bmatrix}$ unifies with $[\,]$, $\begin{bmatrix}\text{F}\end{bmatrix}$, and $\begin{bmatrix}\text{F} & a\end{bmatrix}$
- 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 $\begin{bmatrix}\text{F} & \text{G1}\end{bmatrix}$ unifies with $\begin{bmatrix}\text{F} & \text{G2}\end{bmatrix}$, inspect $G1$ and $G2$
- 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 → NP VP
- Path equations

  (NP AGREEMENT) = (VP AGREEMENT)

Conditions:

1. There can be zero or more path equations for each rule skeleton → no longer atomic
2. When a path equation references constituents, they can only be constituents from the CFG rule

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 → NP VP
(S HEAD) = (VP HEAD)
(NP HEAD AGR) = (VP HEAD AGR)

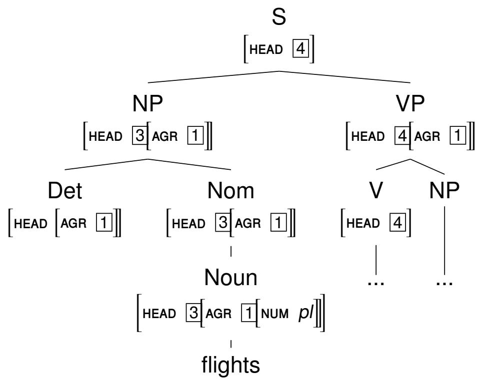
VP → V NP
(VP HEAD) = (V HEAD)

NP → D Nom(inal)
(NP HEAD) = (Nom HEAD)
(Det HEAD AGR) = (Nom HEAD AGR)

Nom → Noun
(Nom HEAD) = (Noun HEAD)

Noun → *flights*
(Noun HEAD AGR NUM) = *pl*

Percolating Agreement Features



Head features in the grammar

- Important concept from the previous rules: heads of grammar rules share properties with their mothers

  VP → V NP
  (VP HEAD) = (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:

VP → V
(V SUBCAT) = *intrans*

VP → V NP
(V SUBCAT) = *trans*

VP → V NP
(V SUBCAT) = *ditrans*

But values like *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. <NP,PP>

Subcategorization rules

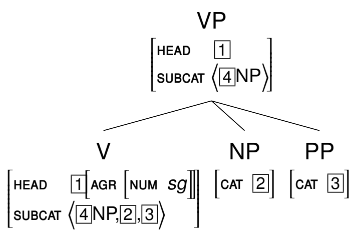
VP → V NP PP
(VP HEAD) = (V HEAD)
(V SUBCAT) = <NP, NP, PP>

V → *leaves*
(V HEAD AGR NUM) = *sg*
(V SUBCAT) = <NP, NP, PP>

More formal way to specify lists:

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

Subcategorization Example



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

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*)

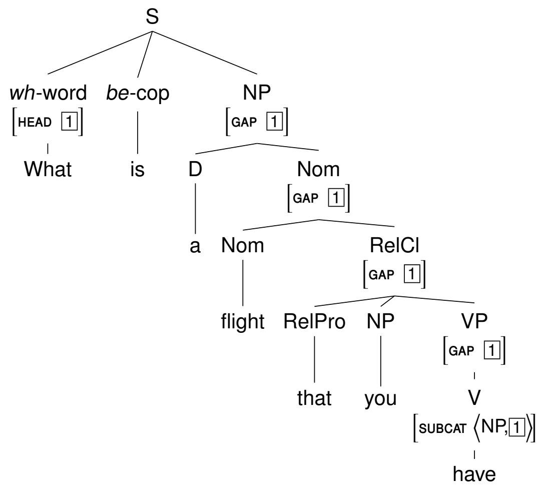
We'll use a GAP feature for this

Handling long-distance dependencies

TOP:		
(fill gap)	S →	*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 SECOND)

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

Handling long-distance dependencies



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