Lecture 19: Feature structures and unification
Today’s lecture

Feature structures form the basis for many grammar formalisms used in computational linguistics.

Feature structure grammars (aka attribute-value grammars, or unification grammars) can be used as
- a more compact way of representing rich CFGs
- a way to represent more expressive grammars
Simple grammars overgenerate

\[
S \rightarrow NP \ VP \\
VP \rightarrow \text{Verb} \ NP \\
NP \rightarrow \text{Det} \ Noun \\
\text{Det} \rightarrow \text{the} | \text{a} | \text{these} \\
\text{Verb} \rightarrow \text{eat} | \text{eats} \\
\text{Noun} \rightarrow \text{cake} | \text{cakes} | \text{student} | \text{students}
\]

This generates ungrammatical sentences like
“these student eats a cakes”

We need to capture (number/person) agreement
Refining the nonterminals

\[ S \rightarrow NP_{sg} \ VP_{sg} \]
\[ S \rightarrow NP_{pl} \ VP_{pl} \]
\[ VP_{sg} \rightarrow \text{Verb}_{Sg} \ NP \]
\[ VP_{pl} \rightarrow \text{Verb}_{Pl} \ NP \]
\[ NP_{sg} \rightarrow \text{Det}_{Sg} \ Noun_{Sg} \]
\[ Det_{Sg} \rightarrow \text{the} \mid \text{a} \]

... ... ...

This yields very large grammars.

What about person, case, ...?

Difficult to capture generalizations.

Subject and verb have to have number agreement

\( NP_{sg}, NP_{pl} \) and \( NP \) are three distinct nonterminals
Feature structures

Replace atomic categories with feature structures:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\quad \begin{bmatrix}
\text{CAT} & \text{VP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{VFORM} & \text{FINITE}
\end{bmatrix}
\]

A feature structure is a list of features (= attributes), e.g. CASE, and values (eg NOM).

We often represent feature structures as attribute value matrices (AVM)
Usually, values are typed (to avoid CASE:SG)
Feature Structures: The Basics
Feature structures as directed graphs

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\quad =
\quad \text{NP}
\]

\[
\text{NP}
\quad \text{CASE}
\quad \text{CAT}
\quad \text{NUM}
\quad \text{PERS}
\quad \text{Nom}
\quad \text{Sg}
\quad 3
\]
Complex feature structures

We distinguish between atomic and complex feature values.
A complex value is a feature structure itself.

This allows us to capture better generalizations.

Only atomic values:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\]

Complex values:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{AGR} & \text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\]
Feature paths

A feature path allows us to identify particular values in a feature structure:

\[
\langle \text{NP CAT} \rangle = \text{NP} \\
\langle \text{NP AGR CASE} \rangle = \text{NOM}
\]
Unification

Two feature structures A and B unify (A ⊔ B) if they can be merged into one consistent feature structure C:

\[
\begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{NUM} & \text{SG} \\
    \text{CASE} & \text{NOM}
\end{bmatrix} \sqcup \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{PERS} & 3
\end{bmatrix} = \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{NUM} & \text{SG} \\
    \text{PERS} & 3 \\
    \text{CASE} & \text{NOM}
\end{bmatrix}
\]

Otherwise, unification fails:

\[
\begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{NUM} & \text{SG} \\
    \text{CASE} & \text{NOM}
\end{bmatrix} \sqcup \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{NUM} & \text{PL}
\end{bmatrix} = \emptyset
\]
Unification as graph-matching
Unification as graph-matching

Unification failure!
Feature Structure Grammars
PATR-II style
feature structures

CFG rules are augmented with constraints:

\[ A_0 \rightarrow A_1 \ldots A_n \]
{set of constraints}

There are two kinds of constraints:

**Unification constraints:**
\[ \langle A_i \text{ feature-path} \rangle = \langle A_j \text{ feature-path} \rangle \]

**Value constraints:**
\[ \langle A_i \text{ feature-path} \rangle = \text{atomic value} \]
A grammar with feature structures

<table>
<thead>
<tr>
<th>S</th>
<th>→</th>
<th>NP VP</th>
<th>Grammar rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨NP NUM⟩</td>
<td>=</td>
<td>⟨VP NUM⟩</td>
<td>Constraints</td>
</tr>
<tr>
<td>⟨NP CASE⟩</td>
<td>=</td>
<td>nom</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>→</th>
<th>DT NOUN</th>
<th>Grammar rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨NP NUM⟩</td>
<td>=</td>
<td>⟨NOUN NUM⟩</td>
<td>Constraints</td>
</tr>
<tr>
<td>⟨NP CASE⟩</td>
<td>=</td>
<td>⟨NOUN CASE⟩</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOUN</th>
<th>→</th>
<th>cake</th>
<th>Lexical entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨NOUN NUM⟩</td>
<td>=</td>
<td>sg</td>
<td>Constraints</td>
</tr>
</tbody>
</table>
With complex feature structures

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td></td>
</tr>
<tr>
<td>⟨NP AGR⟩ = ⟨VP AGR⟩</td>
<td></td>
</tr>
<tr>
<td>⟨NP CASE⟩ = nom</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → DT NOUN</td>
<td></td>
</tr>
<tr>
<td>⟨NP AGR⟩ = ⟨NOUN AGR⟩</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lexical entry</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOUN → cake</td>
<td></td>
</tr>
<tr>
<td>⟨NOUN AGR NUM⟩ = sg</td>
<td></td>
</tr>
</tbody>
</table>

Complex feature structures capture better generalizations (and hence require fewer constraints) — cf. the previous slide.
The head feature

Instead of implicitly specifying heads for each rewrite rule, let us define a head feature.

The head of a VP has the same agreement feature as the VP itself:
Re-entrancies

What we \textit{really} want to say is that the agreement feature of the head is \textit{identical} to that of the VP itself.

This corresponds to a \textit{re-entrancy} in the FS (indicated via coindexation $[1]$ )
Re-entrancies - not like this:
Re-entrancies - but like this:
Extensions of feature structures

Disjunction:
\[\text{eats}: [\text{PERS} \ 3], \quad \text{eat}: [\text{PERS}: 1 \lor 2]\]

Negation:
\[\text{eats}: [\text{PERS} \ 3], \quad \text{eat}: [\text{PERS}: \neg 3]\]

List-valued features:
English *give* takes an NP and a to-PP as arguments, and they have to appear in a specific order:

\[\text{“give the book to you”} \quad \text{not: *”give to you the book”}\]

\[\begin{align*}
\text{give:} & \quad [\text{CAT: VP}, \quad \text{SUCAT: <NP, PPto>}]
\end{align*}\]

Set-valued features:
German *geben* takes three NPs, which can appear in any order:
\[\begin{align*}
\text{ich gebe dir das Buch} & \quad | \quad \text{das Buch gebe ich dir} & \quad | \quad \text{dir gebe ich das Buch},...
\end{align*}\]

\[\begin{align*}
\text{geben:} & \quad [\text{CAT: S}, \quad \text{SUCAT: \{NPnom NPacc, NPdat\}}]
\end{align*}\]
The Expressive Power of Feature Structure Grammars
If every feature can only have a finite set of values, any attribute-value grammar can be compiled out into a (possibly huge) context-free grammar.
The power-of-2 language: \( L_2 = \{ w^i \mid i \text{ is a power of 2} \} \)

\( L_2 \) is a (fully) context-sensitive language.

(Mildly context-sensitive languages have the **constant growth property** (the length of words always increases by a constant factor \( c \)))

Here is a feature grammar which generates \( L_2 \):

\[
A \rightarrow a \\
\langle A \ F \rangle = 1 \\
A \rightarrow A_1 \ A_2 \\
\langle A \ F \rangle = \langle A_1 \rangle \\
\langle A \ F \rangle = \langle A_2 \rangle
\]
What do feature structures represent?
Using feature structures (I)

We have just seen how to use feature structures to refine/extend context-free grammars.

CFGs provide a *procedural* way to define a language:

- The *grammar* provides a set of *rewrite rules*.

- The *language* consists of the set of terminal strings (the subset of $\Sigma^*$, the set of all strings over the vocabulary $\Sigma$) that can be obtained via a sequence of rewrite rules from the start symbol $S$:

  Rewrite $S$ as NP VP, rewrite NP as DT Noun, rewrite VP as…
Using feature structures (II)

We can also view feature structures as a *declarative* way to specify a language:

- Assume the ‘universe’ of linguistic objects is $\Sigma^*$ (the set of all strings over the vocabulary $\Sigma$)
- The *grammar* specifies a set of *feature structures*.
- Each *feature structure* specifies a set of constraints over linguistic objects.
  Hence, each feature structures defines a set of terminal strings (a subset of $\Sigma^*$) that obeys these constraints.
- The *language* consists of the set of terminal strings that are allowed by at least one feature structure.
Features as constraints

Features impose **constraints** on linguistic objects.

If A and B unify, but B contains more features than A, B is more specific than A:

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\quad A
\]

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\quad B
\]

We also say that **A subsumes** the more specific B. Subsumption defines a *partial ordering* over feature structures.
Typed feature structures

In a **typed feature structure** system,
- each feature structure has a **type**
- each **type specifies which features** its structures can contain
- the **values of each feature are typed**
- types are arranged in a **multiple inheritance hierarchy**:  
  \(\top\) (‘top’) is the root, \(\text{pers}\) is a subtype of \(\text{agr}\), \(\text{3rd-pl-fem}\) is a subtype of \(\text{3rd-pl}\) and \(\text{fem}\) (and of \(\text{3rd, gend, pl, pers,\ldots, agr}\))
Another view of unification failure

- $\top$ (‘top’) is the root of every type hierarchy
  (= the most general supertype)
- $\bot$ (‘bottom’) is a subtype of every type.
Features as constraints

Type Hierarchy

Universe of ‘linguistic objects’ (=strings)
Feature structure grammars

There are a number of grammar formalisms (the most widely used is Head-Driven Phrase Structure Grammar [HPSG, Pollard & Sag 1994]) that are based on this constraint-based view of feature structures.

(See next slide for an example)
HPSG signs (feature structures)

- **PHON** string
- **SYNSEM**
  - **LOCAL**
    - **CAT**
    - **VAL**
    - **CONT**
  - **NONLOCAL**
    - **DTRS**

**SYNSEM**

**DTRS**

**phonological form**

**syntactic/semantic constraints**

**local constraints**

**syntactic category**

**syntactic head**

**modifying constraints**

**subcategorization frames**

**semantic representations**

**non-local dependencies**

**daughter structures**

CS447 Natural Language Processing
Feature structures: Summary

- We can use feature structures to refine or extend CFGs.
- Feature structures define constraints over linguistic objects (e.g. constituents)
- Feature structures may subsume each other.
- Feature structures can be simple or complex.
- A feature structure can be viewed as a directed graph.
- Feature structures can be combined via unification.
- Unification can be viewed as graph matching.
- Unification may fail.
- Feature structures may contain reentrancies (cycles).
- Feature structures can be typed.
- Types can be arrange in a type hierarchy.