Logical Grammars

- Introduction
- Logic grammars
- Feature grammars
Logical Grammars 2

Basic idea

• In Phrase structured grammars $\Sigma$ and $V$ were finite sets (simple collections of labels, tagsets)
  • $A \rightarrow \alpha \quad \alpha \in (V \cup \Sigma)^*$

• In Logic Grammars (LG) elements of $V$ and $\Sigma$ can be complex categories, owning internal structure and possibly infinite
  • functors, Prolog terms, feature vectors and matrices, ...

NLP Logical grammars  2
Logical Grammars

• Most known system:
  • Definite clause grammars (DCG)
Feature structures

• Features and complex categories
  • FS (Feature Structures)
• Unification Formalisms (Shieber)
  • Declarative
  • Based on complex categories (FS)
  • Lexicalized (usually based on nuclear CFG).
  • Unification as basic mechanism combination of FS.
  • Constraints over FS as way of representing the well formation rules of the Language.
Feature structures

• Two families:
  • Open Feature Structures
  • Typed Feature Structures
    • every FS belongs to a type
    • The values allowed for assigning to a feature belong to a type.
Feature structures

```
he: [pos : det
  synt :
  agreement :
    gen : masc
  num : sing
  person : 3
]

sem :

mor :
...```

NLP Logical grammars
Feature structures

FS as DAG (Directed Acyclic Graphs)
Feature structures 5

- **Path**: sequence of features (a branche of the tree).

- **Constraints** can be expressed as:
  - `<path_1> = <path_2>` (reentrancy)
  - `<path_1> = <value>`
  - `<synt agreement gen> = fem`
Reentrancy

\[
\begin{align*}
[f & : [h : a]] \\
[g & : [h : a]]
\end{align*}
\]

\[
\begin{align*}
[f & : <1> [h : a]] \\
[g & : <1>]
\end{align*}
\]
Feature structures

Subsumption

- **FS₁** subsumes **FS₂** (is more general, or contains less information) iff:
  - Each feature in **FS₁** exists in **FS₂**.
  - Whatever value of a feature of **FS₁** subsumes the corresponding value of **FS₂**.
  - If two values are reentrant in **FS₁** they are also reentrant in **FS₂**.

\[ FS₁ \sqsubseteq FS₂ \]
Feature structures

\[ \text{FS}_1 \subseteq \text{FS} \]

\[
\begin{array}{c}
\left[ \text{sint: [ cat: n ]} \right] \\
\cap \\
\left[ \text{sint: [ cat: n, concord: [ gen: fem ]]} \right]
\end{array}
\]

\[
\begin{array}{c}
\left[ \text{sint: [ cat: n, concord: [ num: sing ]]} \right]
\end{array}
\]

\[
\begin{array}{c}
\left[ \text{sint: [ cat: n, concord: [ num: sing, gen: fem ]]} \right]
\end{array}
\]
The basic operation is **unification**. This operation allows combining the information of two FSs if they compatible (unificable). The unification of $FS_1$ and $FS_2$ is the more general FS subsumed by both

$$FS = FS_1 \downarrow FS_2$$
Feature structures

\[
\begin{array}{c}
\text{sint: cat: n} \\
\text{concord: gen: fem}
\end{array}
\] 

\[
\begin{array}{c}
\text{sint: cat: n} \\
\text{concord: num: sing}
\end{array}
\]
Feature structures

• Prolog terms vs (untyped) FS (DAGs)
  • Prolog terms are a restricted class of DAGs where reentrancy (subgraph sharing) is only allowed for leaves assigned to variables.
  • Both formalisms can represent partial information.
  • Both formalisms allow incremental addition of information
  • DAGs allow omitting non essential information
  • DAGs allow, at the same embedding level an order free declaration of features
• The most known system is PATR II (Shieber)
Feature structures

- Prolog terms vs FS.
  - naive FS representation
    - Gazdar, Mellish, 1989
  - Boyer, 1988
  - P-Patr, Hirsh, 1988
  - Schöter, 1993

- Unification of PT vs Unification of FS.
  - PT unification is (almost) lineal
  - DAG unification is $O(n^2)$
Feature structures

$$X = \begin{bmatrix}
\text{head:} & \begin{bmatrix}
\text{cat:} & \begin{bmatrix}
\text{n: +}
\end{bmatrix}
\text{v: -}
\end{bmatrix}
\text{agr:} & \begin{bmatrix}
\text{per: 3}
\end{bmatrix}
\text{bar: 2}
\end{bmatrix}
\text{the fish}
\end{bmatrix}$$

$$\text{cat}(+, -, 3, _, _, 2)$$

$$X = \begin{bmatrix}
\text{head:} & \begin{bmatrix}
\text{cat:} & \begin{bmatrix}
\text{n: +}
\end{bmatrix}
\text{v: -}
\end{bmatrix}
\text{agr:} & \begin{bmatrix}
\text{per: 3}
\text{num: pl}
\end{bmatrix}
\text{case: nom}
\text{bar: 2}
\end{bmatrix}
\text{the fish are colourful}
\end{bmatrix}$$

$$\text{cat}(+, -, 3, \text{pl}, \text{nom}, 2)$$
Parsing with open FS

\[
X = \left[ \begin{array}{c}
\text{head:} \\
\text{cat:} \\
\text{agr:} \\
\text{bar:} \\
\end{array} \right]\left[ \begin{array}{c}
\text{[n: +]} \\
\text{[v: -]} \\
\text{[per: 3]} \\
\text{2} \\
\end{array} \right]
\]

the fish

\text{cat(+-,3,_,_,2)}

\begin{itemize}
\item X: head : cat : n <= ' + '
\item X: head : cat : v <= ' - '
\item X: head : agr : per <= 3
\item X: bar <= 2
\end{itemize}
Representing FS with open Prolog lists (Gazdar, Mellish)

\[
\begin{aligned}
\text{cat} : v \\
\text{arg0} : \left[ \begin{array}{l}
\text{cat} : \text{sn} \\
\text{caso} : \text{nom} \\
\text{num} : \text{sing} \| \_ \| \_
\end{array} \right]
\end{aligned}
\]

\[
\begin{aligned}
\text{[cat:v, arg0:[cat:sn, caso:nom, num:sing|_|_]]}
\end{aligned}
\]
Feature structures

PatrII notation

```
pepe : [...
    sem : pepe
    mor :
    sin :
        concord : [num : sing
                     persona : 3]
        gen : masc
    cat : n
]
```

word pepe:

- `<sin cat> = n`
- `<sin concordancia gen> = masc`
- `<sin concordancia num> = sing`
- `<sin concordancia persona> = 3`
- `<sem> = pepe`
- `...`

let nombre-masc-sing be:
  <sin cat> = n
  <sin concordancia gen> = masc
  <sin concordancia num> = sing
  <sin concordancia persona> = 3.

word pepe:   nombre-masc-sing
             <sem> = pepe.
let verb be:
<sin cat> = v
<sin suj cat> = gn
<sin suj caso> = nominative.

let vt be:
verbo
<sin obj1 cat> = gn
<sin obj1 caso> = acusative.

let vdat be:
vt
<sin obj2 cat> = gprep
<sin obj2 prep> = a.

word laugh:
verb
<sem pred> = laugh
<sem arg1> = human.
(someone laughs)

word give:
vdat
<sem pred> = give
<sem arg1> = human
<sem arg2> = thing
<sem arg3> = human.
(someone gives something to someone).
X0 --> X1 X2
   <X0 cat> = NP
   <X1 cat> = det
   <X2 cat> = n
   <X1 agree> = <X2 agree>
   <X0 agree> = <X1 agree>.

NP --> det n
   <det agree> = <n agree>
   <NP agree> = <n agree>.
Typed FS

examples: ALE, CUF, TFS, ALEP

Each FS owns an associated type
Each feature has an associated type for its values
Type structure is usually prescripted

bot sub [list, atom]
  list sub [e-list ne-list]
    e-list sub []
    ne-list sub []
    intro [hd:bot,
          tl: list]
atom sub [a, b]
  a sub []
  b sub []
Feature structures

HPSG using ALE

bot sub [sign,synsem,loc,cat,head,list,nonloc,pform,bool,cont,context,para,index,psoa,con_struc,pers,num,gen,case,vform,ontologia,morfo,wh_str,marking,xbar].
sign sub [word,phrase]
intro [phon:list,synsem:synsem].
phrase sub []
intro [dtrs:con_struc].

... cat sub [cat_subst,cat_funct]
intro [head:head,subj:list,comps:list,mark:list,fill:list,adj:list,
marking:bool,xbar:xbar].

... synsem sub [synsemsubst,synsemfunct]
intro [loc:loc,nonloc:nonloc].
synsemsubst sub [synsemnoun,synsemverb,synsemadj,synsemprep]
intro [loc:locsubst].
synsemfunct sub [synsemdet,synsemmark].
synsemnoun sub [synsempropi,synsemcomu]
intro [loc:locnoun].