NLP
Introduction to NLP

Features and Unification
Need for Feature-based Grammars

• Example (number agreement)
  – The dogs bites

• Example (count/mass nouns)
  – many water

• Example in French (number and person agreement w/subject)
  – Paul est parti, Michelle est partie, Ils sont partis, Elles sont parties

• Example in French (number and person agreement w/direct object)
  – Je l’ai vu (I saw him), Je l’ai vue (I saw her)

• Idea
  – S $\rightarrow$ NP VP
  (but only if the person of the NP is equal to the person of the VP)
Parameterized Grammars

• Parameterized rules, e.g.,
  – S -> NP[person,number,”nominative”] VP[person,number]
  – VP[person,number] -> V[person,number] NP[person,number,”accusative”]
  – NP[“first”,number,”nominative”] -> DET[number]N[number]

• Appropriate modifications are needed to the parser
Unification Grammars

• Various unification grammar formalisms
  – LFG, HPSG, FUG
• Handle agreement
  – e.g., number, gender, person
• Unification
  – Two constituents can be combined only if their features can ‘unify’
• Feature structures (FS or FD)
  – Nested structures that represent all features in an attribute-value matrix
  – Values are typed, so GENDER=PLURAL is not allowed
  – FSs can also be represented as graphs (DAG)
  – Feature paths (from root to a node in the graph)
import nltk;
from __future__ import print_function
from nltk.featstruct import FeatStruct
from nltk.sem.logic import Variable, VariableExpression, Expression
fs1 = FeatStruct(number='singular', person=3)
print (fs1)
[ number = 'singular' ]
[ person = 3          ]
fs2 = FeatStruct(type='NP', agr=fs1)
print (fs2)
[ agr  = [ number = 'singular' ] ]
[  [ person = 3       ] ]
[  ]
[ type = 'NP'         ]

http://www.nltk.org/howto/featstruct.html
Feature Unification

- Graph-matching
- Recursive definition
  - Two FSs unify if they can be merged into a consistent FS
  - Leaf nodes unify if:
    - They are the same
    - One can “subsume” the other
    - Special case: One or both are blank
Feature Unification

\[
\begin{array}{c}
\text{CAT} \quad \text{NP} \\
\text{PERSON} \quad 3 \\
\text{NUMBER} \quad \text{SINGULAR}
\end{array}
\quad \mathbf{U} \quad \begin{array}{c}
\text{CAT} \quad \text{NP} \\
\text{NUMBER} \quad \text{SINGULAR} \\
\text{PERSON} \quad 3
\end{array}
\]
Feature Unification

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{PERSON} & 3
\end{bmatrix}
\cup
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{PERSON} & 1
\end{bmatrix}
\]

\text{FAILURE}
Example in NLTK

fs2 = FeatStruct(type='NP', agr=fs1)
print (fs2)
[ agr  = [ number = 'singular' ] ]
[       [ person = 3 ] ]
[   ]
[ type = 'NP' ]

fs3 = FeatStruct(agr=FeatStruct(number=Variable('?n')), subj=FeatStruct(number=Variable('?n')))
print(fs3)
[ agr = [ number = ?n ] ]
[       ]
[ subj = [ number = ?n ] ]

print(fs2.unify(fs3))
[ agr = [ number = 'singular' ] ]
[       [ person = 3 ] ]
[   ]
[ subj = [ number = 'singular' ] ]
[   ]
[ type = 'NP' ]

http://www.nltk.org/howto/featstruct.html
Agreement with Features

• $S \rightarrow NP\ VP$
  $\{NP\ PERSON\} = \{VP\ PERSON\}$

• $S \rightarrow Aux\ NP\ VP$
  $\{Aux\ PERSON\} = \{NP\ PERSON\}$

• $Verb \rightarrow \text{bites}$
  $\{Verb\ PERSON\} = 3$

• $Verb \rightarrow \text{bite}$
  $\{Verb\ PERSON\} = 1$
Types in Semantics

- **e** – entities, **t** – facts
- \( <e,t> \) : unary predicates – maps entities to facts
- \( <e,<e,t>> \) : binary predicates
- \( <<e,t>,t> \) : type-raised entities

Examples:
- “Jorge”, “he”, A123: **e**
- “Janice likes cats”: **t**
- “likes”: \( <e,<e,t>> \)
- “likes cats”: \( <e,t> \)
- “every person”: \( <<e,t>,t> \)
Type Coercion

• Programming languages
  – How is it done in your favorite programming language?

• Examples in natural language
  – I had a coffee this morning (-> one cup of coffee)
  – I tried two wines last night (-> two types of wine)
  – I had fish for dinner (-> some fish, not “a fish”)
Subtypes and Selectional Restrictions

• Type hierarchy
  – object > edible object > fruit > banana
  – noun > count noun
  – noun > mass noun

• Selectional restrictions
  – Some verbs can only take arguments of certain types
  – Example: eat + “edible object”, believe + “idea“

• Selectional restrictions and type coercion (metonymy)
  – I have read this title (“title” -> “book”)
  – I like Shakespeare (“Shakespeare” -> “works by Shakespeare”)
Subcategorization with Features

- **VP → Verb**
  \[
  \{\text{VP SUBCAT}\} = \{\text{Verb SUBCAT}\} \\
  \{\text{VP SUBCAT}\} = \text{INTRANS}
  \]

- **VP → Verb NP**
  \[
  \{\text{VP SUBCAT}\} = \{\text{Verb SUBCAT}\} \\
  \{\text{VP SUBCAT}\} = \text{TRANS}
  \]

- **VP → Verb NP NP**
  \[
  \{\text{VP SUBCAT}\} = \{\text{Verb SUBCAT}\} \\
  \{\text{VP SUBCAT}\} = \text{DITRANS}
  \]
Representing FSs as DAGs

- FS = feature structure
- DAG = directed acyclic graph (not a tree and not an arbitrary graph)

[Example from Jurafsky and Martin]
FS Unification

[Example from Jurafsky and Martin]
Unification Procedure

```
function UNIFY(f1-orig,f2-orig) returns f-structure or failure

f1 ← Derefenced contents of f1-orig
f2 ← Derefenced contents of f2-orig

if f1 and f2 are identical then
    f1.pointer ← f2
    return f2
else if f1 is null then
    f1.pointer ← f2
    return f2
else if f2 is null then
    f2.pointer ← f1
    return f1
else if both f1 and f2 are complex feature structures then
    f2.pointer ← f1
    for each f2-feature in f2 do
        f1-feature ← Find or create a corresponding feature in f1
        if UNIFY(f1-feature.value,f2-feature.value) returns failure then
            return failure
    return f1
else return failure
```

[Example from Jurafsky and Martin]
FS Unification

[Example from Jurafsky and Martin]
Unification with the Earley Parser

• Important to use the constraints during, not after parsing

```plaintext
function EARLEY-PARSE(words, grammar) returns chart

ADDTOCHART(\( \gamma \rightarrow \bullet S, [0,0], \text{dag}_{\gamma} \), chart[0])

for i ← from 0 to LENGTH(words) do
  for each state in chart[i] do
    if INCOMPLETE?(state) and
      NEXT-CAT(state) is not a part of speech then
      PREDICTOR(state)
    elseif INCOMPLETE?(state) and
      NEXT-CAT(state) is a part of speech then
      SCANNER(state)
    else
      COMPLETER(state)
  end
end
return(chart)
```

[Example from Jurafsky and Martin]
procedure PREDICTOR((A → α • B β, [i, j], dag_A))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ADDTOCHART((B → • γ, [j, j], dag_B), chart[j])
    end

procedure SCANNER((A → α • B β, [i, j], dag_A))
    if B ∈ PARTS-OF-SPEECH(word[j]) then
        ADDTOCHART((B → word[j]•, [j, j + 1], dag_B), chart[j + 1])
    end

procedure COMPLETER((B → γ •, [j, k], dag_B))
    for each (A → α • B β, [i, j], dag_A) in chart[j] do
        if new-dag ← UNIFY-STATES(dag_B, dag_A, B) ≠ Fails!
            ADDTOCHART((A → α B • β, [i, k], new-dag), chart[k])
        end
    end

procedure UNIFY-STATES(dag1, dag2, cat)
    dag1-cp ← COPYDAG(dag1)
    dag2-cp ← COPYDAG(dag2)
    UNIFY(FOLLOW-PATH(cat, dag1-cp), FOLLOW-PATH(cat, dag2-cp))

procedure ADDTOCHART(state, chart-entry)
    if state is not subsumed by a state in chart-entry then
        PUSH-ON-END(state, chart-entry)
    end

[Example from Jurafsky and Martin]
Subsumption

- Unification of a more general concept with a more specific concept
- “undefined” is the most general concept
- “fail” is the least general concept
## Subcategorization

### Noun Phrase Types
- **There**: nonreferential
- **It**: nonreferential
- **NP**: noun phrase

### Preposition Phrase Types
- **PP**: preposition phrase
- **PPing**: gerundive PP
- **PPpart**: particle

### Verb Phrase Types
- **VPbrst**: bare stem VP
- **VPto**: to-marked infin. VP
- **VPwh**: wh-VP
- **VPing**: gerundive VP

### Complement Clause types
- **Sfin**: finite clause
- **Swh**: wh-clause
- **Sif**: whether/if clause
- **Sing**: gerundive clause
- **Sto**: to-marked clause
- **Sf Turk**: for-to clause
- **Sbrst**: bare stem clause

### Other Types
- **AjP**: adjective phrase
- **Quo**: quotes

---

*Example from Jurafsky and Martin*
# Subcategorization

<table>
<thead>
<tr>
<th>Subcat</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Quo</em></td>
<td>asked $[Quo \ &quot;What was it like?&quot;]$</td>
</tr>
<tr>
<td><em>NP</em></td>
<td>asking $[NP \ a \ question]$</td>
</tr>
<tr>
<td><em>Swh</em></td>
<td>asked $[Swh \ what \ trades \ you\’re \ interested \ in]$</td>
</tr>
<tr>
<td><em>Sto</em></td>
<td>ask $[Sto \ him \ to \ tell \ you]$</td>
</tr>
<tr>
<td><em>PP</em></td>
<td>that means asking $[PP \ at \ home]$</td>
</tr>
<tr>
<td><em>Vto</em></td>
<td>asked $[Vto \ to \ see \ a \ girl \ called \ Evelyn]$</td>
</tr>
<tr>
<td><em>NP \ Sif</em></td>
<td>asked $[NP \ him] [Sif \ whether \ he \ could \ make]$</td>
</tr>
<tr>
<td><em>NP \ NP</em></td>
<td>asked $[NP \ myself] [NP \ a \ question]$</td>
</tr>
<tr>
<td><em>NP \ Swh</em></td>
<td>asked $[NP \ him] [Swh \ why \ he \ took \ time \ off]$</td>
</tr>
</tbody>
</table>

[Example from Jurafsky and Martin]
Notes

- FSs can have a special, “head” feature
- If all features have a finite domain, attribute-value grammars can be converted into a CFG
- The power-of-2 language doesn’t have the “constant growth property”.
  - It is a CSL and cannot be recognized by a CFG.
  - It can, however, be recognized by a mildly-context-sensitive grammar.
“Power-of-2 Language”

- Can be generated by an LFG

\[
\begin{align*}
A & \rightarrow A \\
(\uparrow f) & = \downarrow \\
A & \rightarrow a \\
(\uparrow f) & = 1
\end{align*}
\]

[Example from Bob Berwick]
Summary

• Feature structures define constraints
• Subsumption
• Nested features structures
• Cycles are allowed (DAG)
• Unification
• Types
• Type subsumption
NLP