Unification

Subsumption

- A less specific (more abstract) feature $F$ subsumes (written $\subseteq$) another feature $G$ iff
  - For every feature $x$ in $F$, $F(x) \subseteq G(x)$
  - For all paths $p$ and $q$ in $F$ such that $F(p) = F(q)$, it is also the case that $G(p) = G(q)$
- Can add features or fill in more details, but can't change constraints when go to bigger one. *More information. Semilattice*
- Define $F \cup G$ to be smallest $H$ subsumed by both $F$ and $G$

Unification

- Algorithm most easily expressed using DAG's w/ Content & Pointer fields.
- Ignore Content if Pointer ≠ NULL
- Unify($f_1, f_2$) results in $f = f_1 \cup f_2$ where both $f_1$ and $f_2$ have been modified so they represent the same (via sharing) feature structure.

*Bird uses intersection instead of union!*

Feature Structure as DAG

Number: SG
Person: 3

Don't need Content e3 Pointer in Python e3 Java – sharing!

To Unify
Add Person to f1 & Ptr to Person of f2 so share content

function UNIFY(f1, f2) returns fstructure or failure
  \( f1\text{-real} \leftarrow \text{Real contents of } f1 \)
  \( f2\text{-real} \leftarrow \text{Real contents of } f2 \)
  if \( f1\text{-real} \) is null then
    \( f1\text{.pointer} \leftarrow f2 \)
    return f2
  else if \( f2\text{-real} \) is null then
    \( f2\text{.pointer} \leftarrow f1 \)
    return f1
  else if \( f1\text{-real} \) and \( f2\text{-real} \) are identical then
    \( f1\text{.pointer} \leftarrow f2 \)
    return f2
  else if both \( f1\text{-real} \) and \( f2\text{-real} \) are complex feature structures then
    \( f2\text{.pointer} \leftarrow f1 \)
    for each feature in \( f2\text{-real} \) do
      other-feature \leftarrow \text{Find or create a feature corresponding to feature in } f1\text{-real}
      if UNIFY(feature.value, other-feature.value) returns failure then
        return failure
    return f1
  else return failure

Unification Example

Agreement: ①  [Number: SG]
Subject:   [Agreement: ① ]
          [Subject: [Agreement: [Person: 3]]]
          = ???

Example: Unify

Recursive

- Go down Subject: Agreement path
- Need to unify [Agreement ①] with [Agreement: [Person: 3]]
- Need to unify [Number: SG] with [Person:3]
- Add Person to [Number: SG] w/ content NULL.
Recursively Go Up

- Nothing else to set -- just recursively go up tree making all nodes merge together

Summary

- Result of unification does not just make inputs equal, it makes them identical (one points to the other)
- Later unifications affect both!

Parsing w/Features

- Add features and unification to any parsing technique.
- Add constraints to rules.
- Can unify after build parse tree, but eliminate erroneous earlier if unify as build.
Adding Constraints to the Grammar

Rather than writing:

\[ S \rightarrow NP \ VP \]

\[ <NP \ Head \ Agreement> = <VP \ Head \ Agreement> \]

\[ <S \ Head> = <VP \ Head> \]

Instead write as unification structure

\[ \text{Dag} = \begin{bmatrix}
S: [\text{Head: } 1] \\
NP: [\text{Head: } [\text{Agreement: } 2]] \\
VP: [\text{Head: } 1 [\text{Agreement: } 2]]
\end{bmatrix} \]

Adding Earley

Originally entries in chart[j] were of form:

\[ A \rightarrow uv, i \] representing trying to satisfy \[ A \rightarrow uv \]
where have matched \[ u \] portion so far, and \[ u \]
portion represents word[i,j]

Actually needed one more piece to allow to
calculate actually trees -- indicate rules
that allowed to parse last step

We’ll still leave that out here in examples.

Table Entries

| Chart[2] | S23: Det → that | [1,2] | Scanner |
|          | S29: Nominal → Noun | [3,3] | (S28) |
|          | S30: NP → Det Nominal | [1,3] | (S23,S29) |
|          | S33: VP → Verb NP | [0,3] | (S12,S30) |
|          | S36: S → VP | [0,3] | (S33) |

Chart Entries

Add the DAG associated w/rule applied

At beginning of previous rule write in

Chart[0]:

\[ S \rightarrow NP \ VP, 0, [], \text{Dag} \]

Recall 3 actions in Earley:

Scan (recognize terminal)

Predict (guess production for next non-
terminal)

Attach (finished production, use to move dot
forward in others).

Earley with Unification

Add ROOT \rightarrow S, 0, dag_{prev}, to column 0.

For each j from 0 to n:

For each state (dotted rule) in column j,

(associated with production as we go)

look at what’s after the dot:

• If it's a pre-terminal B, SCAN(state).

• If it's a non-terminal B, PREDICT(state)

• If there's nothing after the dot, COMPLETER(state)

Return true if last column has ROOT \rightarrow S.

Adding to Charts

SCAN(A \rightarrow \alpha . B\beta, [i,j], dag_{\alpha}):  
If \[ B \rightarrow w_i \] is a production,

Enqueue((B \rightarrow w_j, j, dag_{\alpha}), chart[i+1])

PREDICT(A \rightarrow \alpha . B\beta, [i,j], dag_{\alpha}):

For each rule of the form \[ B \rightarrow \gamma \]

Enqueue((B \rightarrow \gamma, j, dag_{\alpha}), chart[i])

COMPLETER(B \rightarrow \gamma, [i,j], dag_{\alpha}):

For each state of the form \[ (A \rightarrow \alpha . B\beta, k, dag_{\alpha}) \] in chart[i]

if new-dag \leftarrow Unify-States(dag_{\alpha}, dag_{\beta}, B) \# fails

Enqueue((A \rightarrow \alpha \beta, k, new-dag_{i}), chart[i])
Unify-States

Unify-States(dag1, dag2, cat)

dag1-cpy ← CopyDag(dag1)
dag2-cpy ← CopyDag(dag2)

Unify(Follow-Path(cat, dag1-cpy),
      Follow-Path(cat, dag2-cpy))

- Unifies appropriate entries!
- Why make copies?
  - Unification might fail
  - Reuse entries, even if succeeds!

Enqueue

Enqueue(state, chart-entry)

if state is not subsumed by a state in chart-entry then
  add state to chart-entry

- What if state subsumed by state already in chart?
- More general state always better as can unify with more things.

NLTK & Features

>>> nltk.data.show_cfg('grammars/feat0.fcfg')
% start S
# ############################
# Grammar Rules
# ############################
# S expansion rules
S → NP[NUM=?n] VP[NUM=?n]
# NP expansion rules
NP[NUM=?n] → N[NUM=?n]
NP[NUM=?n] → PropN[NUM=?n]
# VP expansion rules

Earley Parsing

>>> tokens = 'Kim likes children'.split()
>>> from nltk.parse import load_earley
>>> cp = load_earley('grammars/feat0.fcfg', trace=2)
>>> trees = cp.nbest_parse(tokens)

Sample grammar in standard distribution

trace=1 provides less information
Unified Tree

```python
>>> for tree in trees: print tree
...
(S[
  (NP[NUM='sg'] (PropN[NUM='sg'] Kim))
  (VP[NUM='sg', TENSE='pres']
    (TV[NUM='sg', TENSE='pres'] likes)
    (NP[NUM='pl'] (N[NUM='pl'] children)))
)
```

Creating Feature Structures

```python
>>> nltk.FeatStruct("[POS='N', AGR=[PER=3, NUM='pl', GND='fem']]")
[AGR=[GND='fem', NUM='pl', PER=3], POS='N']

With sharing:

```python
>>> nltk.FeatStruct("**
...
  [S = [Head = (1)],
  ...  [NP=[Head = [ Agreement = (2) ]],
  ...  [VP=[Head = (1) [ Agreement = (3) ]]]]
**")

[NP=[Head=[Agreement=(2)]], S=[Head=(1)],
VP=[Head=[Agreement=(3)]]]
```

Unifying Feature Structures

```python
fs1.unify(fs2) -- unifies feature structures
```

Any Questions?