CS 181: Natural Language Processing

Lecture 13: Features and Unification

Kim Bruce
Pomona College
Spring 2008

Disclaimer: Slide contents borrowed from many sources on web!

Unification

Subsumption

- A less specific (more abstract) feature F *subsumes* (written ⊆) another feature G iff
  - For every feature x in F, F(x) ⊆ 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 ∪ G to be smallest H subsumed by both F and G

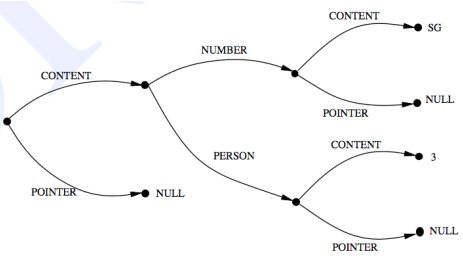
Unification

- Algorithm most easily expressed using DAG's w/ Content & Pointer fields.
  - Ignore Content if Pointer ≠ NULL
- Unify(f1,f2) results in f = f1 ∪ f2 where both f1 and f2 have been modified so they represent the same (via sharing) feature structure.

Bird uses intersection instead of union!

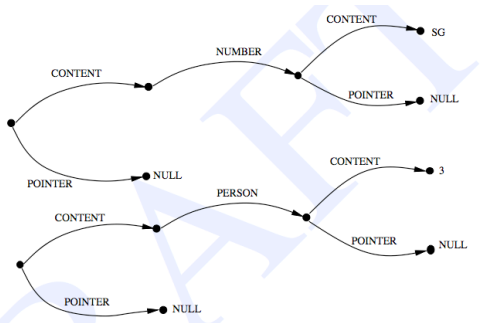
Feature Structure as DAG

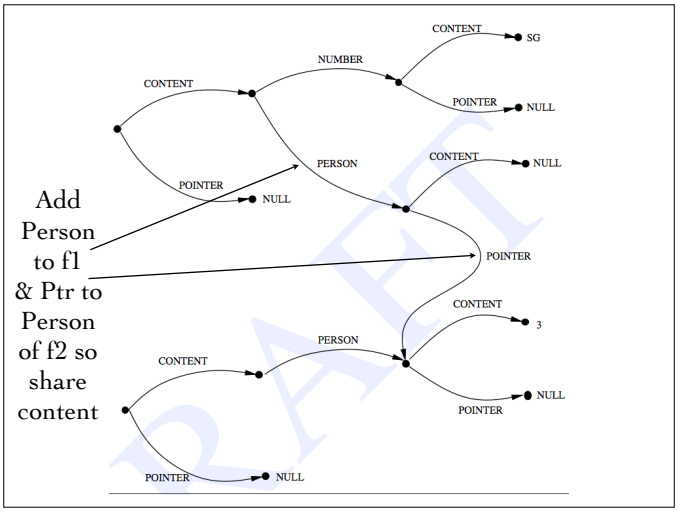
Number: SG
Person: 3



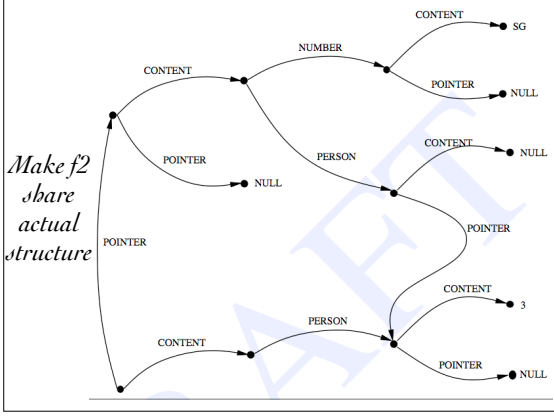
Don't need Content & Pointer in Python & Java – sharing!

To Unify





Not Enough!



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

Unification Example

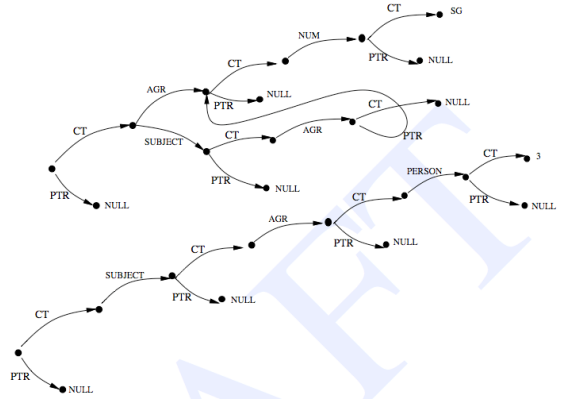
Agreement: ①	[Number: SG]
Subject:	[Agreement: ①]

∪

[Subject: [Agreement: [Person: 3}}}

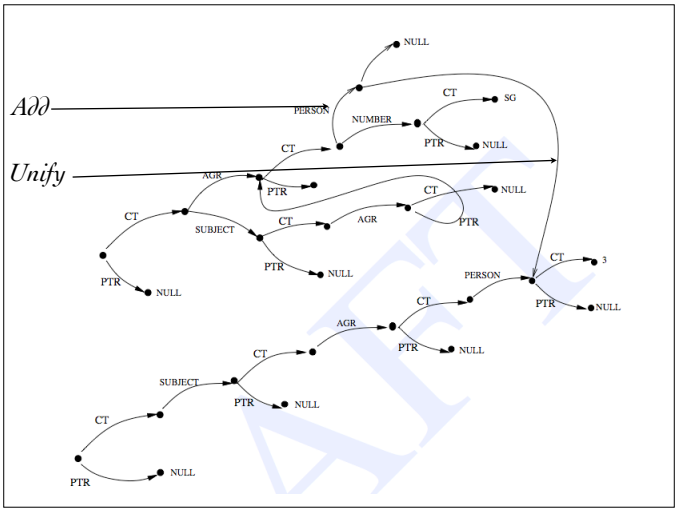
= ???

Example: Unify



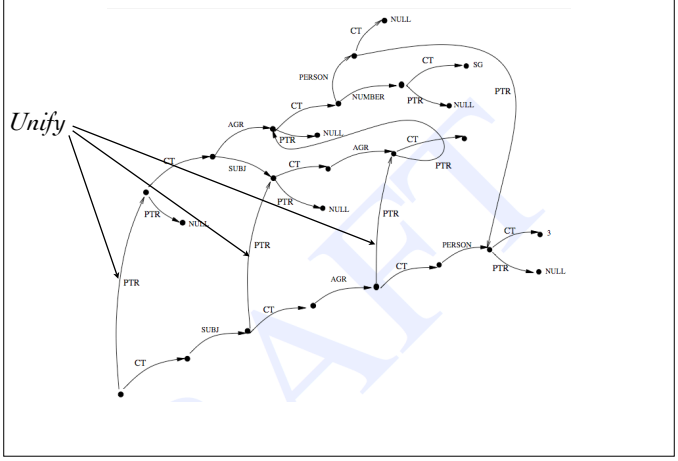
Recursive

- 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.
- Unify [Person: NULL] with [Person: 3] by sharing contents.



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!

Earley + Unification ⇒ Features

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 → NP VP
    - <NP Head Agreement> = <VP Head Agreement>
    - <S Head> = <VP Head>
- Instead write as unification structure

Dag = [S: [Head: ①]
NP: [Head: [Agreement: ②]]
VP: [Head: ① [Agreement: ②]]]

Adapting Earley

- Originally entries in chart[j] were of form:
  - A → u.v, i representing trying to satisfy A → uv where have matched u portion so far, and u portion represents word[i,j]
- Actually needed one more piece to allow to construct actually trees -- indicate rules that allowed to parse last step
- We'll still leave that out here in examples.

Table Entries

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

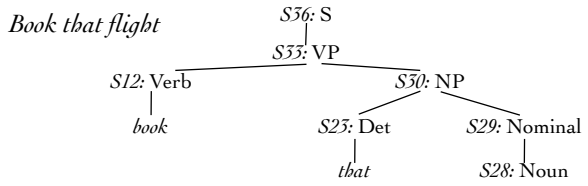


Chart Entries

- Add the DAG associated w/rule applied
- At beginning of previous rule write in Chart[0]:
  - S → . NP VP, 0, [], *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 → . S, 0, dag_{Root}, to column 0.
For each j from 0 to n:
  For each state (dotted rule) in column j,
    (including those added 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 → S .

Adding to Charts

SCAN(A → α . Bβ, [i,j], dag_A):
  If B → w_j is a production,
    Enqueue((B→w . , j, dag_B), chart[j+1])

PREDICT(A → α . Bβ, [i,j], dag_A):
  For each rule of the form B→γ
    Enqueue((B → . γ , j, dag_B), chart[j])

COMPLETER(B → γ . , [i,j], dag_B):
  For each state of the form (A → α . Bβ, k, dag_A) in chart[i]
    if new-dag ← Unify-States(dag_B,dag_A,B) ≠ fails
      Enqueue((A → αB . β , k, new-dag), chart[j])

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]
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
NP[NUM=pl] -> N[NUM=pl]
# VP expansion rules
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t, NUM=?n]
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP
```

```
# ############################
# Lexical Rules
# ############################
Det[NUM=sg] -> 'this' | 'every'
Det[NUM=pl] -> 'these' | 'all'
Det -> 'the' | 'some'
PropN[NUM=sg]-> 'Kim' | 'Jody'
N[NUM=sg] -> 'dog' | 'girl' | 'car' | 'child'
N[NUM=pl] -> 'dogs' | 'girls' | 'cars' | 'children'
IV[TENSE=pres,  NUM=sg] -> 'disappears' | 'walks'
TV[TENSE=pres, NUM=sg] -> 'sees' | 'likes'
IV[TENSE=pres,  NUM=pl] -> 'disappear' | 'walk'
TV[TENSE=pres, NUM=pl] -> 'see' | 'like'
IV[TENSE=past, NUM=?n] -> 'disappeared' | 'walked'
TV[TENSE=past, NUM=?n] -> 'saw' | 'liked'
```

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

```
    |.K.l.c.|
Processing queue 0
Predictor |> . . .| [0:0] S[] -> * NP[NUM=?n] VP[NUM=?n] {}
Predictor |> . . .| [0:0] NP[NUM=?n] -> * N[NUM=?n] {}
Predictor |> . . .| [0:0] NP[NUM=?n] -> * PropN[NUM=?n] {}
Predictor |> . . .| [0:0] NP[NUM=?n] -> * Det[NUM=?n] N[NUM=?n] {}
Predictor |> . . .| [0:0] NP[NUM='pl'] -> * N[NUM='pl'] {}
Scanner   |[-] . .| [0:1] 'Kim'
Scanner   |[-] . .| [0:1] PropN[NUM='sg'] -> 'Kim' *
Processing queue 1
Completer |[-] . .| [0:1] NP[NUM='sg'] -> PropN[NUM='sg'] *
Completer |[-> . .| [0:1] S[] -> NP[NUM=?n] * VP[NUM=?n] {?n: 'sg'}
Predictor |. > . .| [1:1] VP[NUM=?n, TENSE=?t] -> * IV[NUM=?n, TENSE=?t] {}
Predictor |. > . .| [1:1] VP[NUM=?n, TENSE=?t] -> * TV[NUM=?n, TENSE=?t]
NP[] {}
Scanner   |. [-] .| [1:2] 'likes'
Scanner   |. [-] .| [1:2] TV[NUM='sg', TENSE='pres'] -> 'likes' *
```

```
Processing queue 2
Completer |. [-> .| [1:2] VP[NUM=?n, TENSE=?t] -> TV[NUM=?n, TENSE=?t]
* NP[] {?n: 'sg', ?t: 'pres'}
Predictor |. . > .| [2:2] NP[NUM=?n] -> * N[NUM=?n] {}
Predictor |. . > .| [2:2] NP[NUM=?n] -> * PropN[NUM=?n] {}
Predictor |. . > .| [2:2] NP[NUM=?n] -> * Det[NUM=?n] N[NUM=?n] {}
Predictor |. . > .| [2:2] NP[NUM='pl'] -> * N[NUM='pl'] {}
Scanner   |. . [-]| [2:3] 'children'
Scanner   |. . [-]| [2:3] N[NUM='pl'] -> 'children' *
Processing queue 3
Completer |. . [-]| [2:3] NP[NUM='pl'] -> N[NUM='pl'] *
Completer |. [---]| [1:3] VP[NUM='sg', TENSE='pres'] -> TV[NUM='sg',
TENSE='pres'] NP[] *
Completer |[=====]| [0:3] S[] -> NP[NUM='sg'] VP[NUM='sg'] *
Completer |[=====]| [0:3] [INIT][] -> S[] *
```

Unified Tree

```
>>> 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

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

With sharing:

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

Unifying Feature Structures

fs1.unify(fs2) -- unifies feature structures

Any Questions?