# Feature Structures and Parsing Unification Grammars

11-711 Algorithms for NLP

19 November 2015

# Linguistic features

- (Linguistic “features” vs. ML “features”.)
- Human languages usually include *agreement* constraints; in English, e.g., subject /verb
- *Could* have a separate category for each minor type: N1sf, N1sm, N1pf, N1pm, …
  - *Each* with its own set of grammar rules!
- Much better: represent these regularities using independent ***features:*** number, gender, person, …
- Features are typically introduced by lexicon; checked and propagated by constraint equations attached to grammar rules

# Feature Structures (FSs)

Having multiple orthogonal features with values leads naturally to ***Feature Structures***:

[Det
 [root: *a*]
 [number: sg ]]

A feature structure's values can in turn be FSs:

[NP
 [agreement: [[number: sg]
 [person: 3rd]]]]

# Adding constraints to CFG rules

- S → NP VP

  <NP number> = <VP number>

- NP → Det Nominal

  <NP head> = <Nominal head>

  <Det head agree> = <Nominal head agree>

# FSs from lexicon, constrs. from rules

Lexicon entry:

[Det
[root: *a*]
[number: sg ]]

Rule with constraints:

NP → Det Nominal

<NP number> = <Det number>

<NP number> = <Nominal number>

- Combine to get result:

[NP [Det
[root: *a*]
[number: sg ]]
[Nominal [number: sg] …]
[number: sg]]

# Verb Subcategorization

Verbs have sets of allowed args. Could have many sets of VP rules. Instead, have a SUBCAT feature, marking sets of allowed arguments:

+none -- Jack laughed

+np -- Jack found a key

+np+np -- Jack gave Sue the paper

+vp:inf -- Jack wants to fly

+np+vp:inf -- Jack told the man to go

+vp:ing -- Jack keeps hoping for the best

+np+vp:ing -- Jack caught Sam looking at his desk

+np+vp:base -- Jack watched Sam look at his desk

+np+pp:to -- Jack gave the key to the man

+pp:loc -- Jack is at the store

+np+pp:loc -- Jack put the box in the corner

+pp:mot -- Jack went to the store

+np+pp:mot -- Jack took the hat to the party

+adjp -- Jack is happy

+np+adjp -- Jack kept the dinner hot

+sthat -- Jack believed that the world was flat

+sfor -- Jack hoped for the man to win a prize

50-100 possible ***frames*** for English; a single verb can have several.

*(Notation from James Allen "Natural Language Understanding")*

# Adding transitivity constraint

- S → NP VP

  <NP number> = <VP number>

- NP → Det Nominal

  <NP head> = <Nominal head>

  <Det head agree> = <Nominal head agree>

- VP → Verb NP

  <VP head> = <Verb head>

  <VP head subcat> = +np  *(which means transitive)*

# Applying a verb subcat feature

Lexicon entry:

[Verb
[root: *found*]
[head: find]
[subcat: +np ]]

Rule with constraints:

VP → Verb NP
<VP head> = <Verb head>
<VP head subcat> = +np

- Combine to get result:

[VP [Verb
[root: *found*]
[head: find]
[subcat: +np ]]
[NP …]
[head: [find [subcat: +np]]]]

# Relation to LFG constraint notation

- VP → Verb NP

  <VP head> = <Verb head>

  <VP head subcat> = +np

  *from JM book is the same as the LFG expression*

- VP → Verb NP

  (↑ head) = (↓ head)

  (↑ head subcat) = +np

# Unification

- Merging FSs (and failing if not possible) is called ***Unification***
- Simple FS examples:

[number sg] ⊔ [number sg] = [number sg]

[number sg] ⊔ [number pl] FAILS

[number sg] ⊔ [number []] = [number sg]

[number sg] ⊔ [person 3rd] = [number sg, person 3rd]

# Recap: applying constraints

Lexicon entry:

[Det
[root: *a*]
[number: sg ]]

Rule with constraints:

NP → Det Nominal

<NP number> = <Det number>

<NP number> = <Nominal number>

- Combine to get result:

[NP [Det
[root: *a*]
[number: sg ]]
[Nominal [number: sg] …]
[number: sg]]

# Turning constraint eqns. into FS

Lexicon entry:

[Det
[root: *a*]
[number: sg ]]

- Combine to get result:

[NP [Det
[root: *a*]
[number: sg ]]
[Nominal [number: sg]
…]
[number: sg]]

Rule with constraints:

NP → Det Nominal

<NP number> = <Det number>
<NP number> = <Nominal number>

*becomes:*

[NP [Det [number: (1) ]]
[Nominal
[number: (1) ]
…]
[number: (1) ]]

# Another example

This (oversimplified) rule:

S → NP VP

<S subject> = NP

<S agreement> = <S subject agreement>

turns into this DAG:

[S [agreement (1) ]

[subject [agreement (1) ]]

[subject (2) ]

[NP (2) ]

[VP ]

# Unification example with “EQ“

[agreement (1), subject [agreement (1)]]

⊔ [subject [agreement [person 3rd, number sg]

= [agreement (1),

subject [agreement (1) [person 3rd,

number sg]]]

- <agreement number> ***is*** <subject agreement number> (EQ), so they are equal

# Unification example without “EQ“

[agreement [number sg],
subject [agreement [number sg]]]

⊔ [subject [agreement [person 3rd,
number sg]]]

= [agreement [number sg],
subject [agreement [person 3rd,
number sg]]]

- <agreement number> is equal to <subject agreement number>, but ***not*** EQ

# Seems tricky. Why bother?

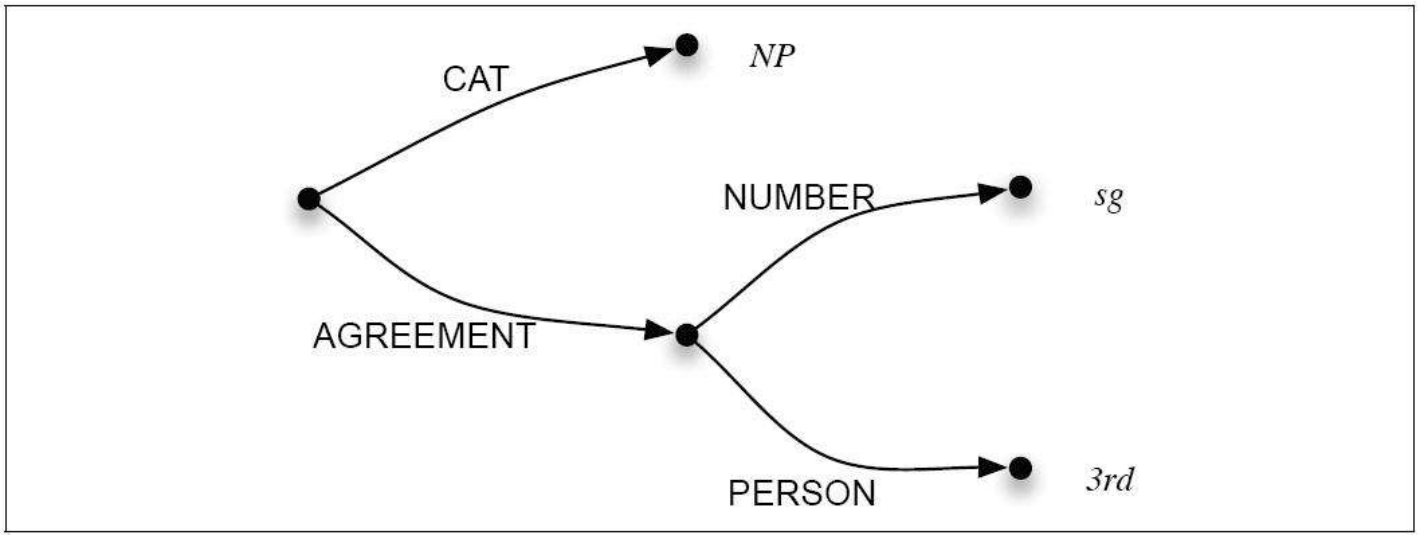
- Unification allows the systems that use it to handle many complex phenomena in "simple" elegant ways:
  - There seems to be a dog in the yard.
  - There seem to be dogs in the yard
- Unification makes this work smoothly.
  - Make the Subjects of the clauses EQ:

    <VP subj> = <VP COMP subj>

    [VP [subj: (1)] [COMP [subj: (1)]]]
  - (Ask Lori Levin for LFG details.)

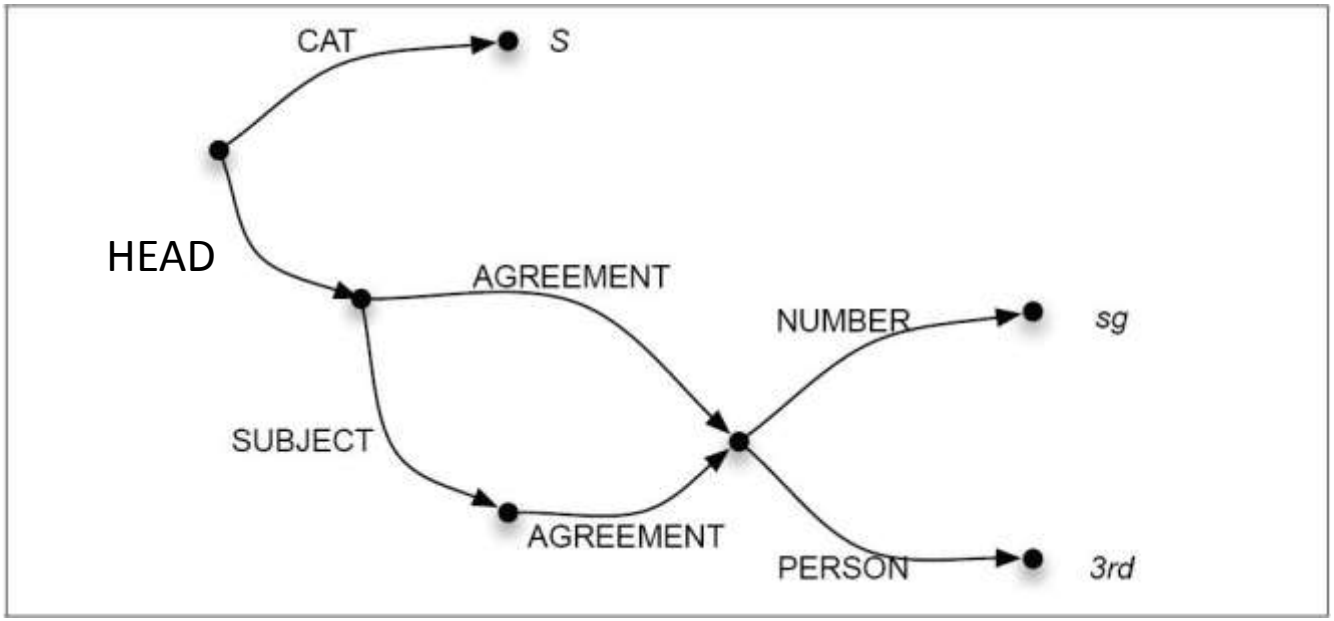
# Representing FSs as DAGs

- Taking feature paths seriously
- May be easier to think about than numbered cross-references in text
- [cat NP, agreement [number sg, person 3rd]]



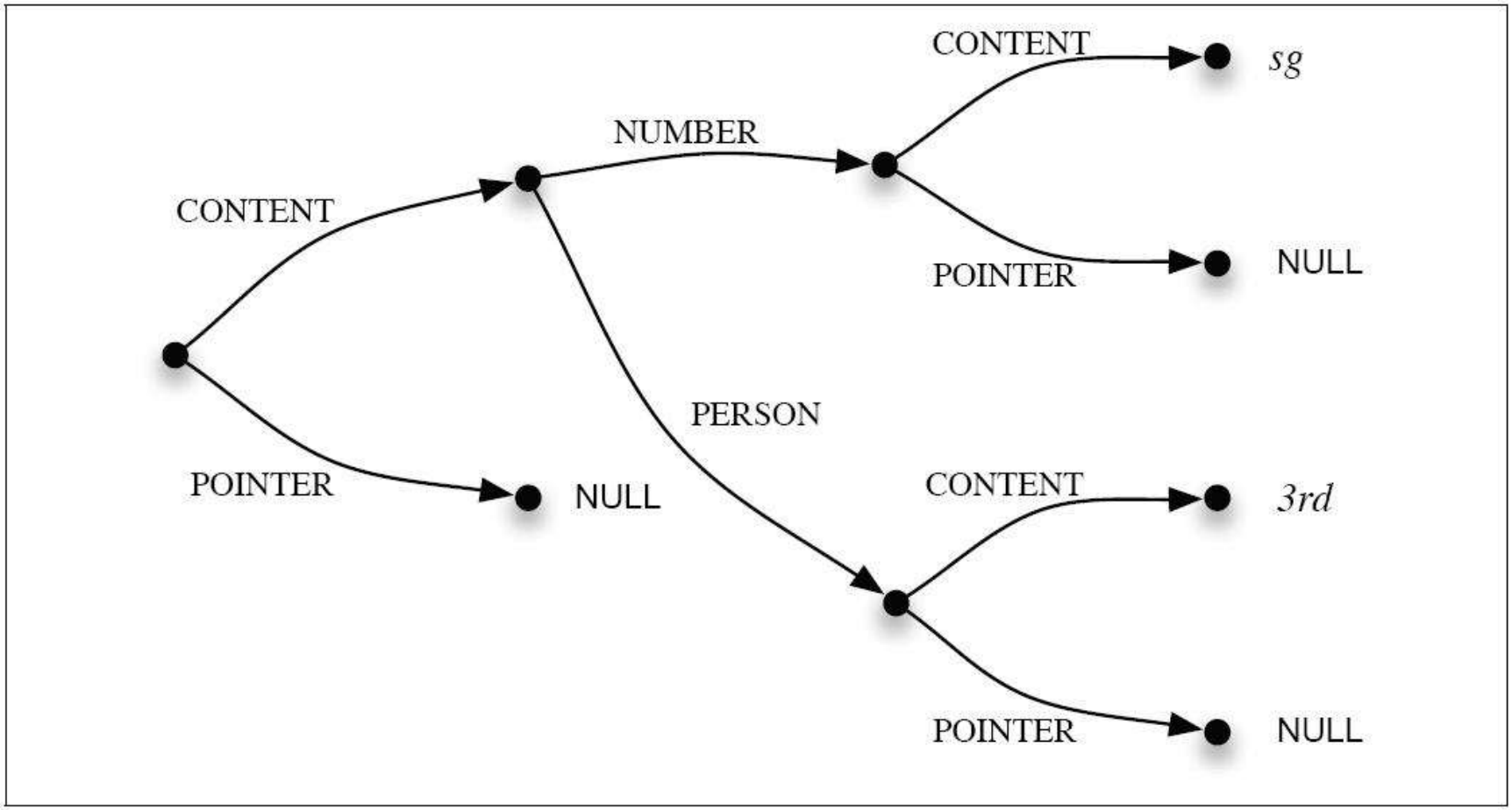
# Re-entrant FS as DAGs

- [cat S, head [agreement (1) [number sg, person 3rd], subject [agreement (1)]]]

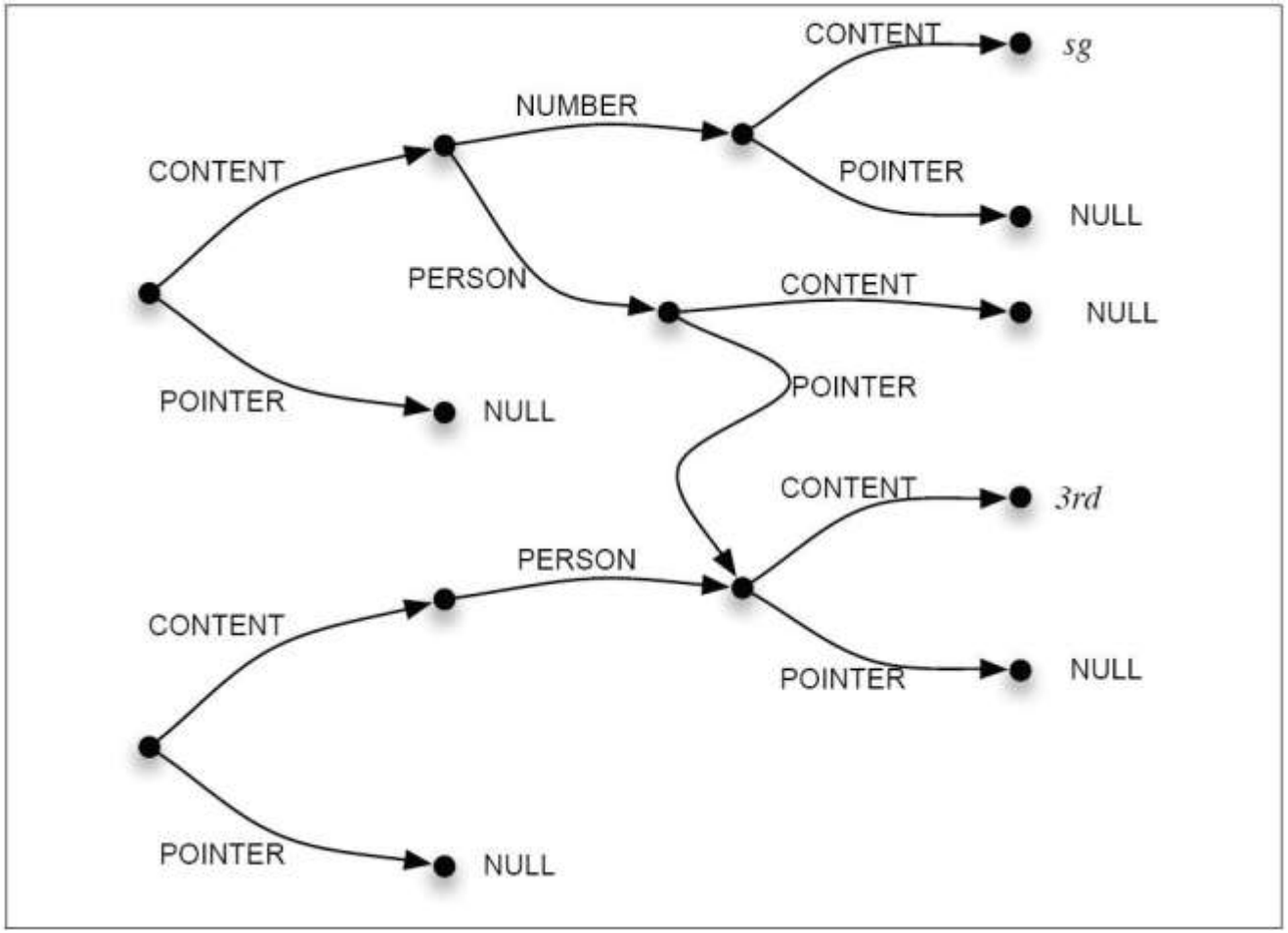


# Splitting nodes into content and pointer

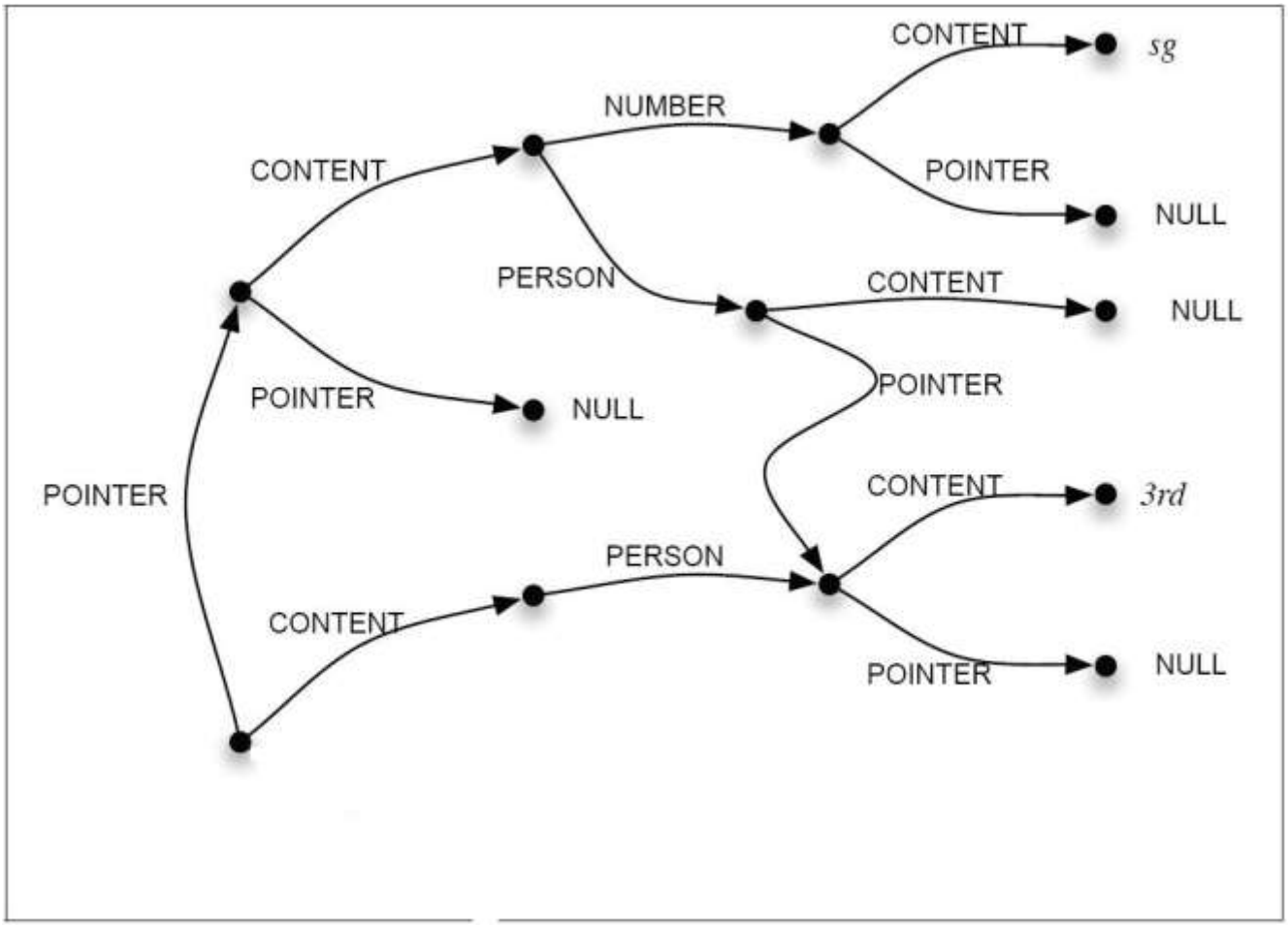
[number sg, person 3rd]



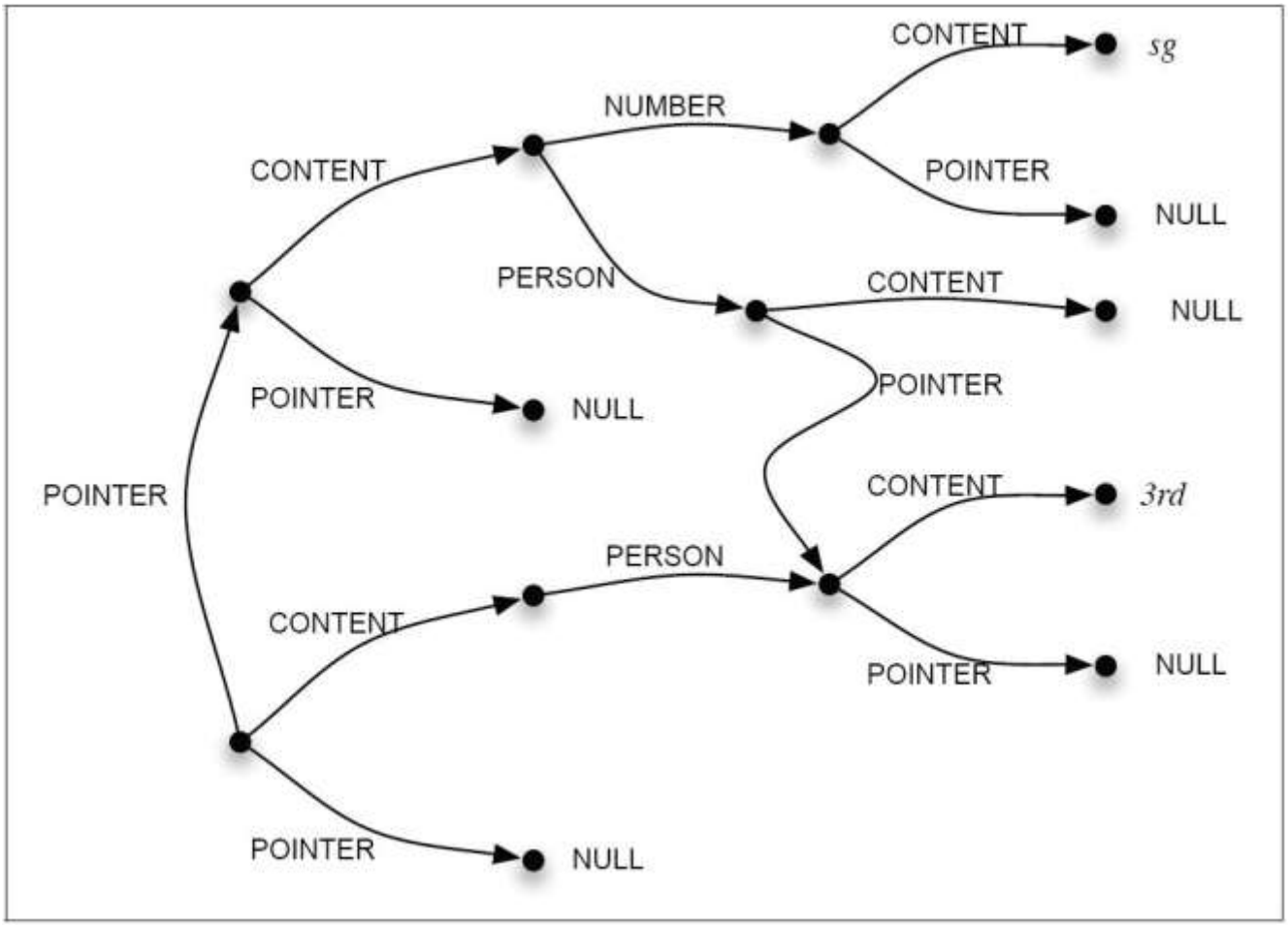
# [number sg] ⊔ [person 3rd]



# [number sg, person 3rd] (corrected)



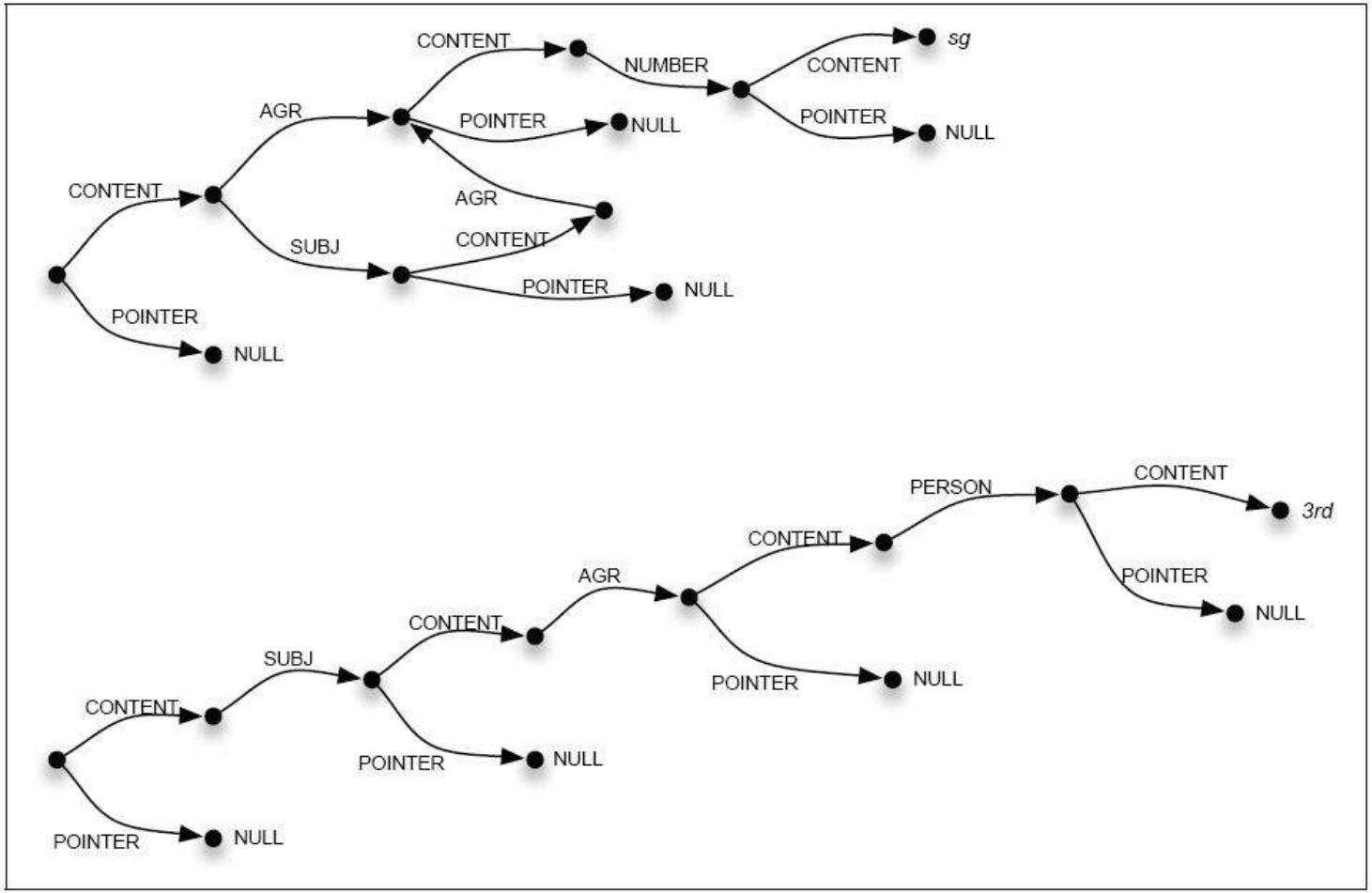
# [number sg, person 3rd] (unlikely…)



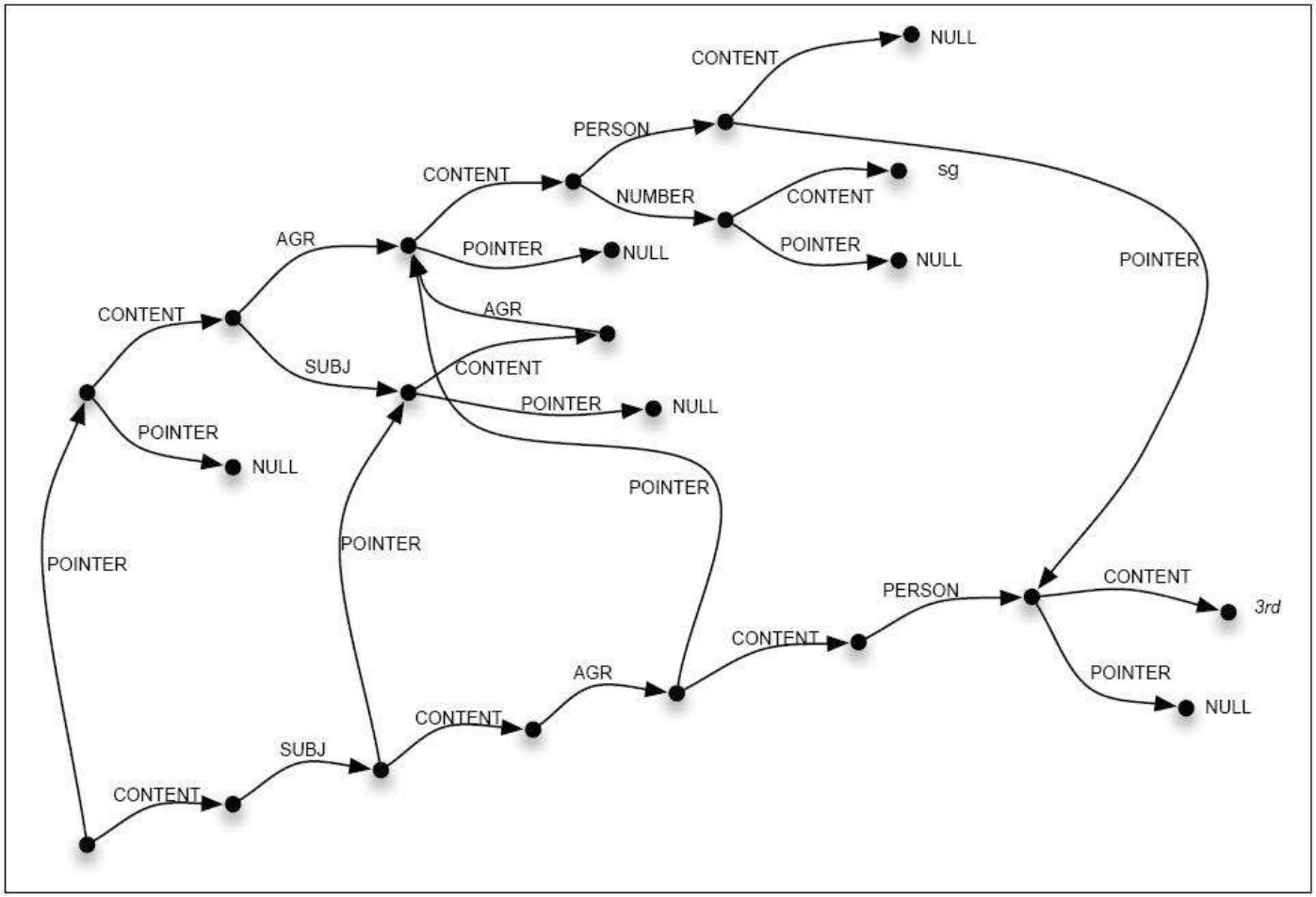
# Unification algorithm

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

 f1 ← Dereferenced contents of f1-orig
 f2 ← Dereferenced 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
```

# [agr (1) [number sg], subj [agr (1)]] ⊔ [subj [agr [person 3rd]]]



# [agr (1) [number sg, person 3rd], subj [agr (1)]]



# Adding Unification to Earley Parser

- Could just parse, then unify FSs at the end
  - But this fails to use constraints to limit bad parses.

1: Add feature structures to rules. This rule:

S → NP VP

<NP head agr> = <VP head agr>

<S head> = <VP head>

turns into this DAG:

[S [head (1)]

NP [head [agr (2)]]

VP [head (1) [agr (2)]]]

# Adding Unification to Earley Parser

2: Include a DAG field in each state in chart

3: Completer unifies the two input DAGs when producing a new state

4: AddToChart tests whether new state is ***subsumed*** by any chart state

5: Unify-States makes copies of both of its DAG args

# Earley+Unification: DAGs added

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

  ADDTOCHART((γ → • S, [0,0], dag_γ), 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)
```

# Earley+Unification: the rest

**procedure** PREDICTOR($(A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A)$)
  **for each** $(B \rightarrow \gamma)$ **in** GRAMMAR-RULES-FOR($B$, *grammar*) **do**
    ADDTOCHART($(B \rightarrow \bullet\,\gamma,\ [j,j],\ dag_B)$, *chart*[*j*])
  **end**

**procedure** SCANNER($(A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A)$)
  **if** B $\in$ PARTS-OF-SPEECH(*word*[*j*]) **then**
    ADDTOCHART($(B \rightarrow word[j]\bullet,\ [j,j+1],\ dag_B)$, *chart*[*j*+1])

**procedure** COMPLETER($(B \rightarrow \gamma \bullet,\ [j,k],\ dag_B)$)
  **for each** $(A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A)$ **in** *chart*[*j*] **do**
    if *new-dag* $\leftarrow$ UNIFY-STATES($dag_B, dag_A, B$) $\neq$ Fails!
      ADDTOCHART($(A \rightarrow \alpha\ B \bullet \beta,\ [i,k]$, *new-dag*), *chart*[*k*])
  **end**

**procedure** UNIFY-STATES(*dag1*, *dag2*, *cat*)
  *dag1-cp* $\leftarrow$ COPYDAG(*dag1*)
  *dag2-cp* $\leftarrow$ 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**

# *Real* Unification-Based Parsing

- X0 → X1 X2

  <X0 cat> = S, <X1 cat> = NP, <X2 cat> = VP

  <X1 head agree> = <X2 head agree>

  <X0 head> = <X2 head>

- X0 → X1 *and* X2

  <X1 cat> = <X2 cat>, <X0 cat> = <X1 cat>

- X0 → X1 X2

  <X1 orth> = *how*, <X2 sem> = <SCALAR>

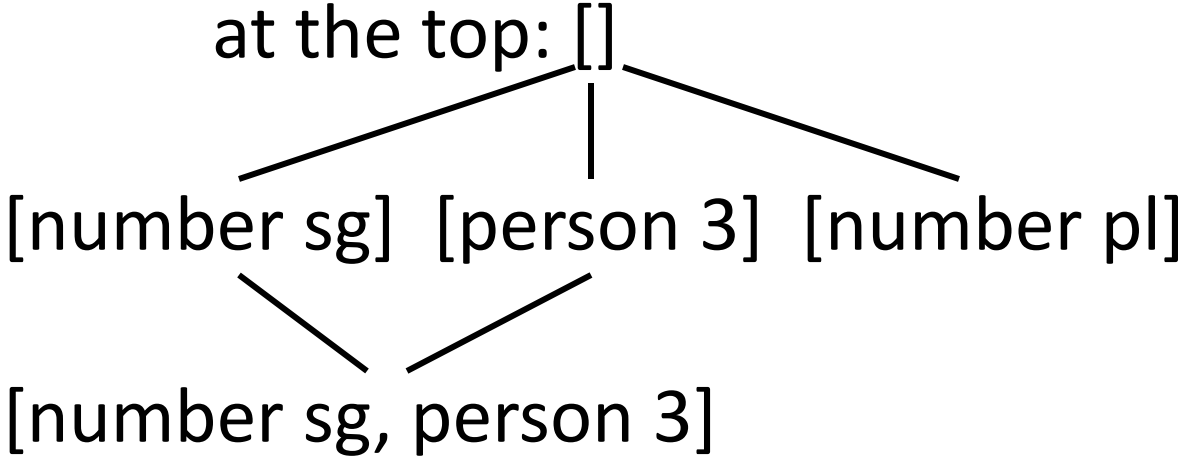
# Complexity

- Earley modification: “search the chart for states whose DAGs ***unify*** with the DAG of the completed state”. Plus a lot of copying.
- Unification parsing is “quite expensive”.
  - NP-Complete in some versions.
  - Early AWB paper on Turing Equivalence(!)
- So maybe *too* powerful?

    (like GoTo or Call-by-Name?)
  - Add restrictions to make it tractable:
    - Tomita’s Pseudo-unification (Tomabechi too)
    - Gerald Penn work on tractable HPSG: ALE

# Formalities: subsumption

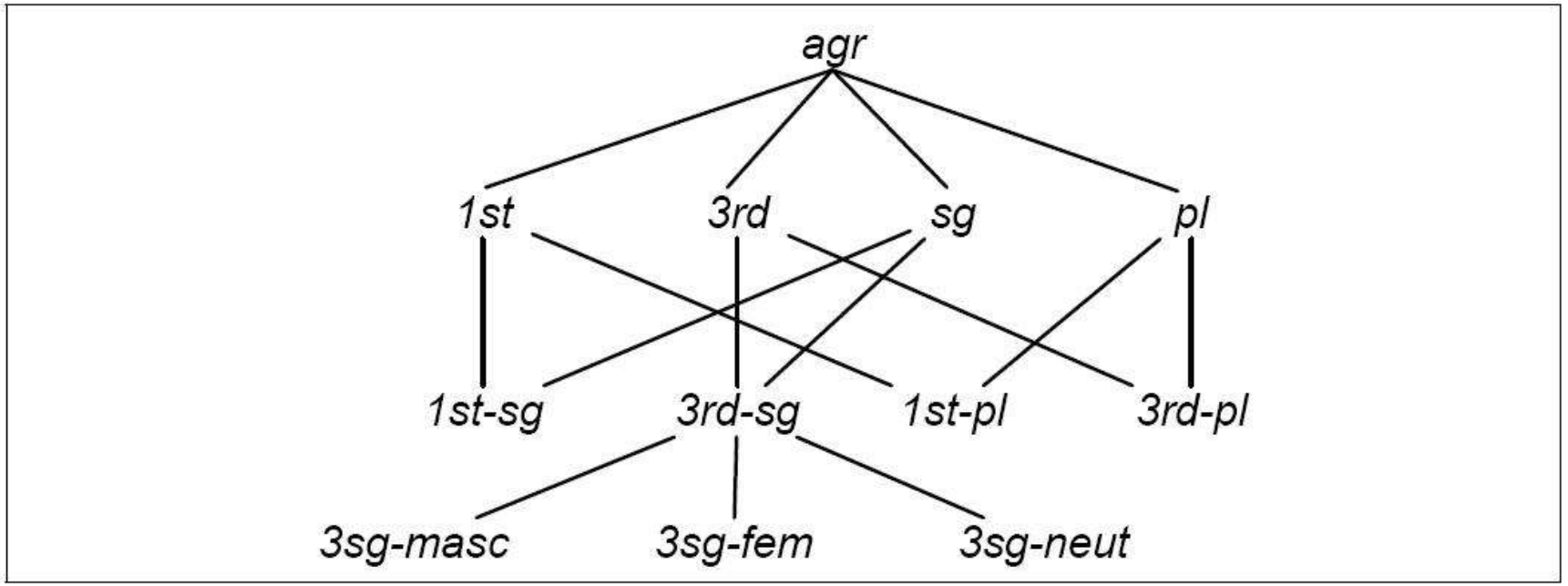
- Less specific FS1 ***subsumes*** more specific FS2
  FS1 $\sqsubseteq$ FS2  (Inverse is FS2 ***extends*** FS1)
- Subsumption relation forms a ***semilattice***,




- Unification defined wrt semilattice:
  F $\sqcup$ G = H s.t. F $\sqsubseteq$ H and G $\sqsubseteq$ H
  H is the Most General Unifier (MGU)

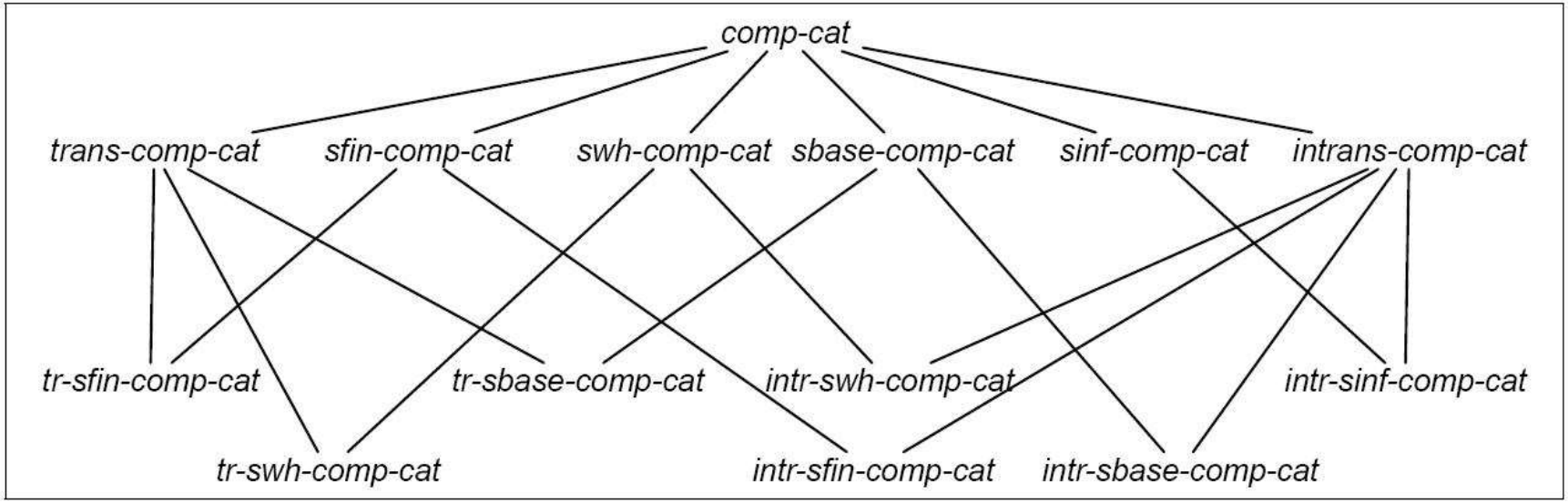
# Hierarchical Types

Hierarchical types allow *values* to unify too (or not):



# Hierarchical subcat frames

Many verbs share *subcat* frames, some with more arguments specified than others:



Questions?

# Subcategorization

| | | |
|---|---|---|
| **Noun Phrase Types** | | |
| **There** | nonreferential there | **There** *is still much to learn* |
| **It** | nonreferential it | **It** *was evident that my ideas* |
| **NP** | noun phrase | *As he was relating* **his story** |
| **Preposition Phrase Types** | | |
| **PP** | preposition phrase | *couch their message* **in terms** |
| **PPing** | gerundive PP | *censured him* **for not having intervened** |
| **PPpart** | particle | *turn it* **off** |
| **Verb Phrase Types** | | |
| **VPbrst** | bare stem VP | *she could* **discuss it** |
| **VPto** | to-marked infin. VP | *Why do you want* **to know**? |
| **VPwh** | wh-VP | *it is worth considering* **how to write** |
| **VPing** | gerundive VP | *I would consider* **using it** |
| **Complement Clause types** | | |
| **Sfin** | finite clause | *maintain* **that the situation was unsatisfactory** |
| **Swh** | wh-clause | *it tells us* **where we are** |
| **Sif** | whether/if clause | *ask* **whether Aristophanes is depicting a** |
| **Sing** | gerundive clause | *see* **some attention being given** |
| **Sto** | to-marked clause | *know* **themselves to be relatively unhealthy** |
| **Sforto** | for-to clause | *She was waiting* **for him to make some reply** |
| **Sbrst** | bare stem clause | *commanded* **that his sermons be published** |
| **Other Types** | | |
| **AjP** | adjective phrase | *thought it* **possible** |
| **Quo** | quotes | *asked* **"What was it like?"** |

# Frames for “ask”

| Subcat | Example |
|---|---|
| *Quo* | asked [$_{Quo}$ “What was it like?”] |
| *NP* | asking [$_{NP}$ a question] |
| *Swh* | asked [$_{Swh}$ what trades you’re interested in] |
| *Sto* | ask [$_{Sto}$ him to tell you] |
| *PP* | that means asking [$_{PP}$ at home] |
| *Vto* | asked [$_{Vto}$ to see a girl called Evelyn] |
| *NP Sif* | asked [$_{NP}$ him] [$_{Sif}$ whether he could make] |
| *NP NP* | asked [$_{NP}$ myself] [$_{NP}$ a question] |
| *NP Swh* | asked [$_{NP}$ him] [$_{Swh}$ why he took time off] |