Strategies for the semantic interpretation

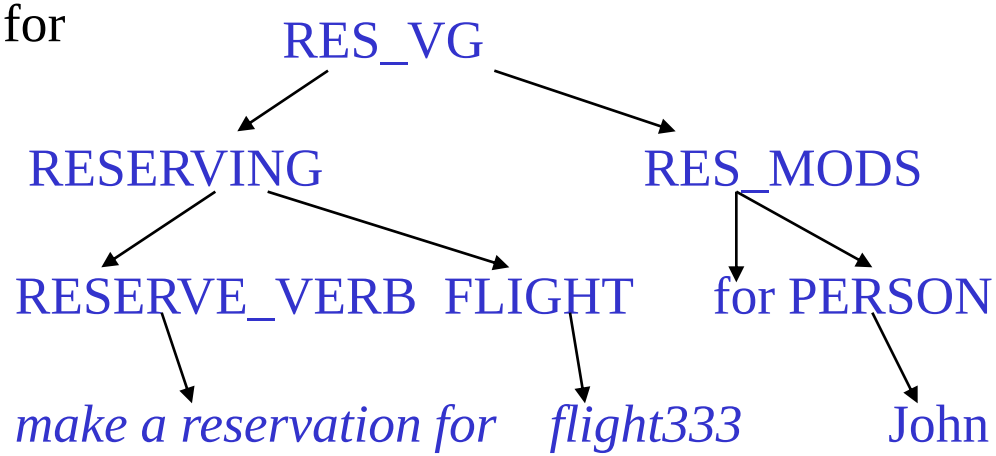
- Once syntactic processing has finished
  - Input: syntactic tree
- Syntactic/semantic processing in parallel
  - There is only one process
  - Grammatical categories include semantic information
  - Semantic information is used to validate hypothesis of syntactic analysis
  - Partial semantic interpretations and syntactic tree are built at the same time.
- Semantic-guided analyzers
  - Syntactic information (when used) is reduced to validate semantic hypothesis

Semantic Grammars$_1$

RES_VG → RESERVING, RES_MODS

RESERVING → RESERVE_VERB, FLIGHT
RES_MODS → for, PERSON
RES_MODS → []

RESERVE_VERB->make a reservation for
FLIGHT -> flight333
FLIGHT -> flight1D3
PERSON -> John
PERSON -> Mary



It can recognize the following sentences

- *make a reservation for flight333*
- *make a reservation for flight333 for John*

Semantic Grammars$_2$

DEP_VG $\rightarrow$ DEPARTING, DEP_MODS

DEPARTING $\rightarrow$ DEPART_VERB
DEPARTING$\rightarrow$DEPART_VERB,SOURCE_LOC

DEP_MODS $\rightarrow$ DEP_MOD, DEP_MODS
DEP_MODS $\rightarrow$[]
DEP_MOD $\rightarrow$ to, DEST_LOC
DEP_MOD $\rightarrow$ from, SOURCE_LOC

Semantic Grammars$_3$

- A formalism combining syntax and semantics (usually, context free grammar)
- Terminal symbols correspond to semantic categories
  - The number of rules increases
  - The number of word for each category decreases
  - Ambiguity increases
  - Adaptability decreases
- Application: Efficient systems in restricted domains
- The logical form representing semantic information is obtained directly from the grammar

Composition function = lambda evaluation$_1$

- Lexical semantics
  - "Pedro" $\rightarrow$ pedro.
  - “runs" $\rightarrow$ (lambda(x), run(x)))
- composition
  - (lambda(x), runs(x)) pedro $\rightarrow$ run(pedro).
- How? Composition functions
- When?

Composition function = lambda evaluation$_2$

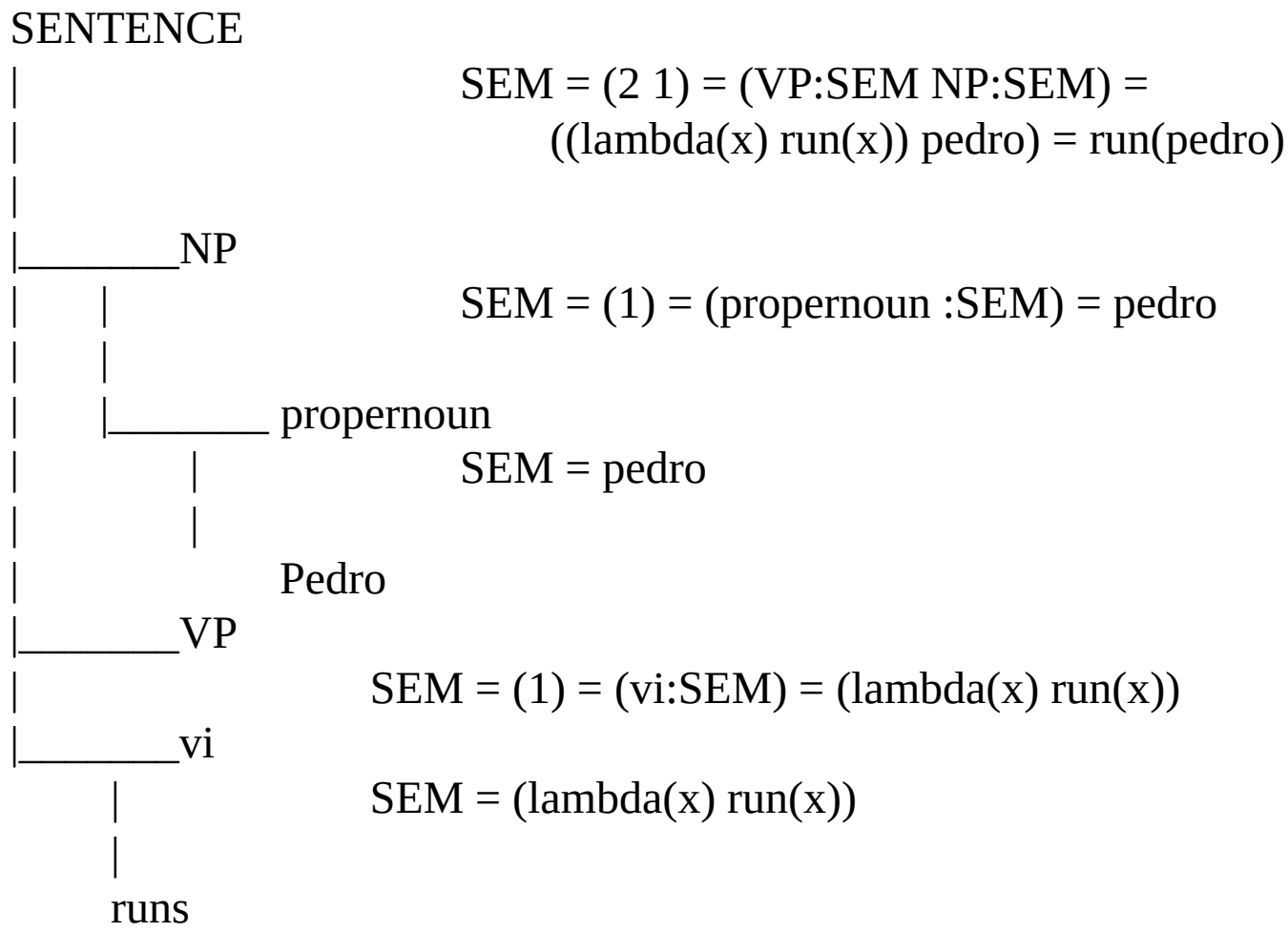
SENTENCE	→ NP VP	(2 1)	
NP	→ propernoun,		(1)
VP	→ vi,	(1)	
VP	→ vt NP	(1 2)	

Pedro	→ propernoun ,		pedro
María	→ propernoun,		maria
ríe	→ vi,	(lambda (x), run(x))	
ama a	→ vt,	((lambda (x), (lambda (y), loves(y,x))))	

Composition function = lambda evaluation$_3$

Example

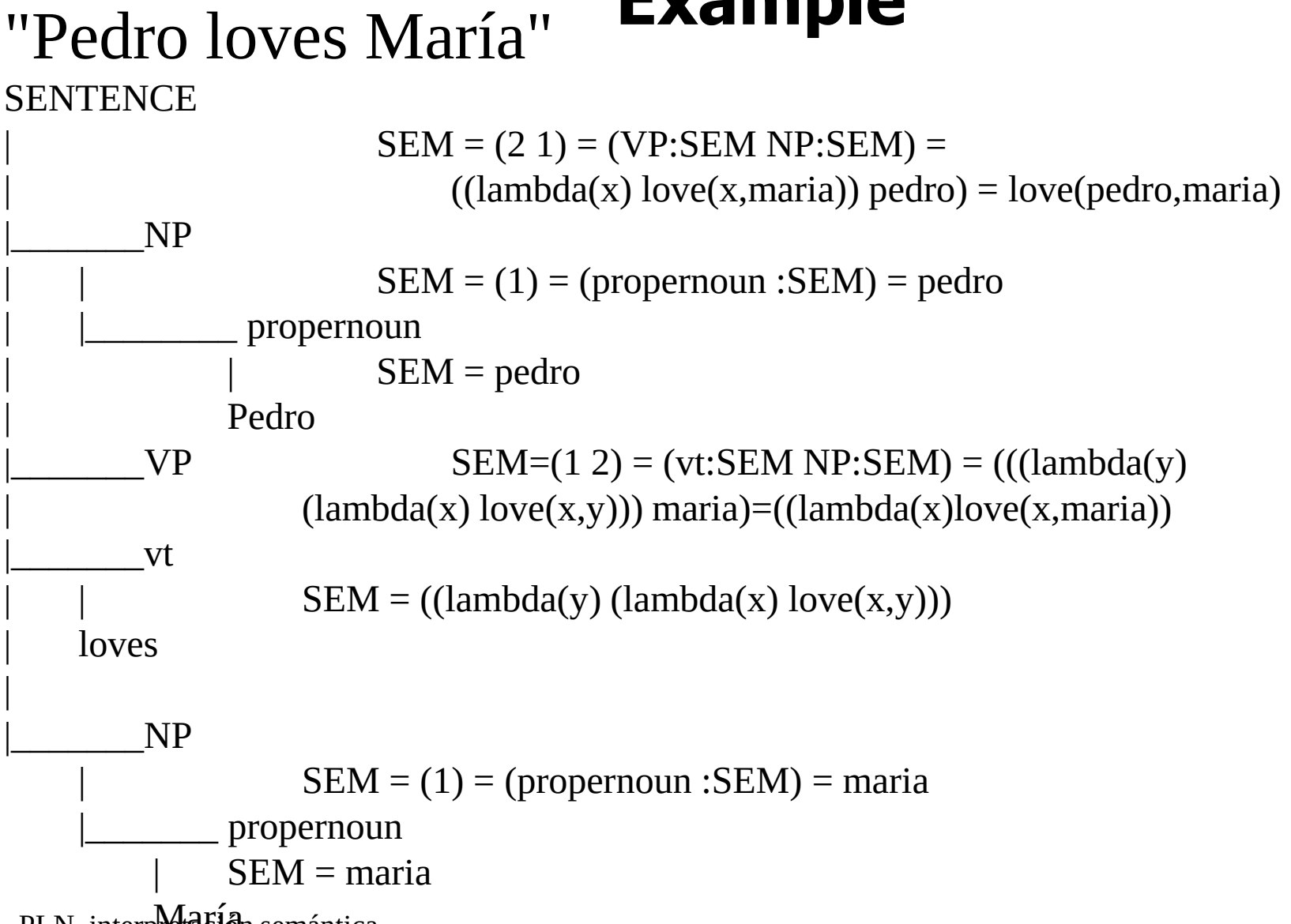
"Pedro runs"



Composition function = lambda evaluation$_4$

Example

"Pedro loves María"



Logical Grammars

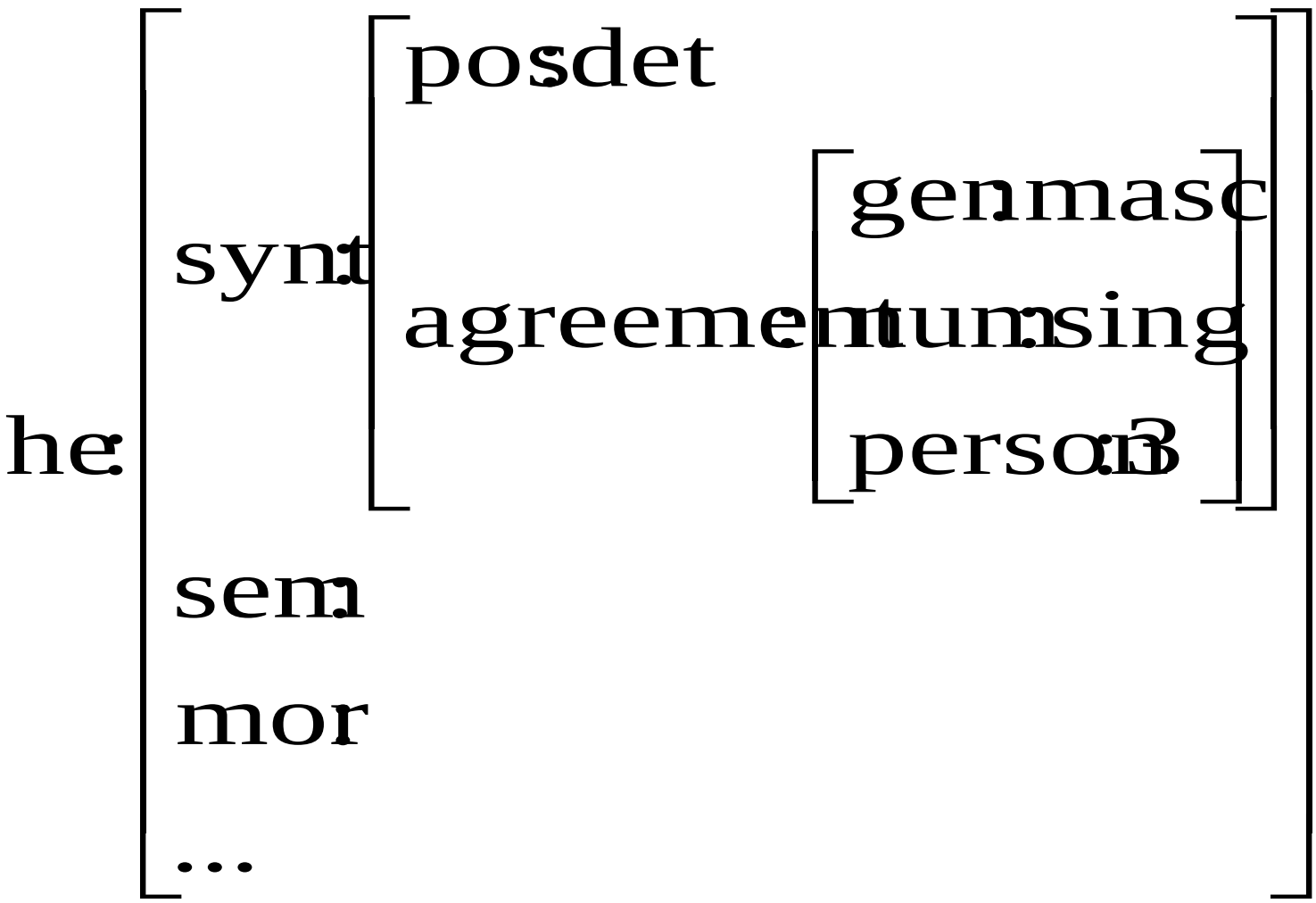
Basic idea

- In Phrase structured grammars Σ and V were finite sets (simple colections of labels, tagsets)
  - $A \rightarrow \alpha \quad \alpha \in (V \cup \Sigma)^*$
- In Logic Grammars (LG) elements of V and Σ can be complex categories, owning internal structure and possibly infinite

Feature structures1

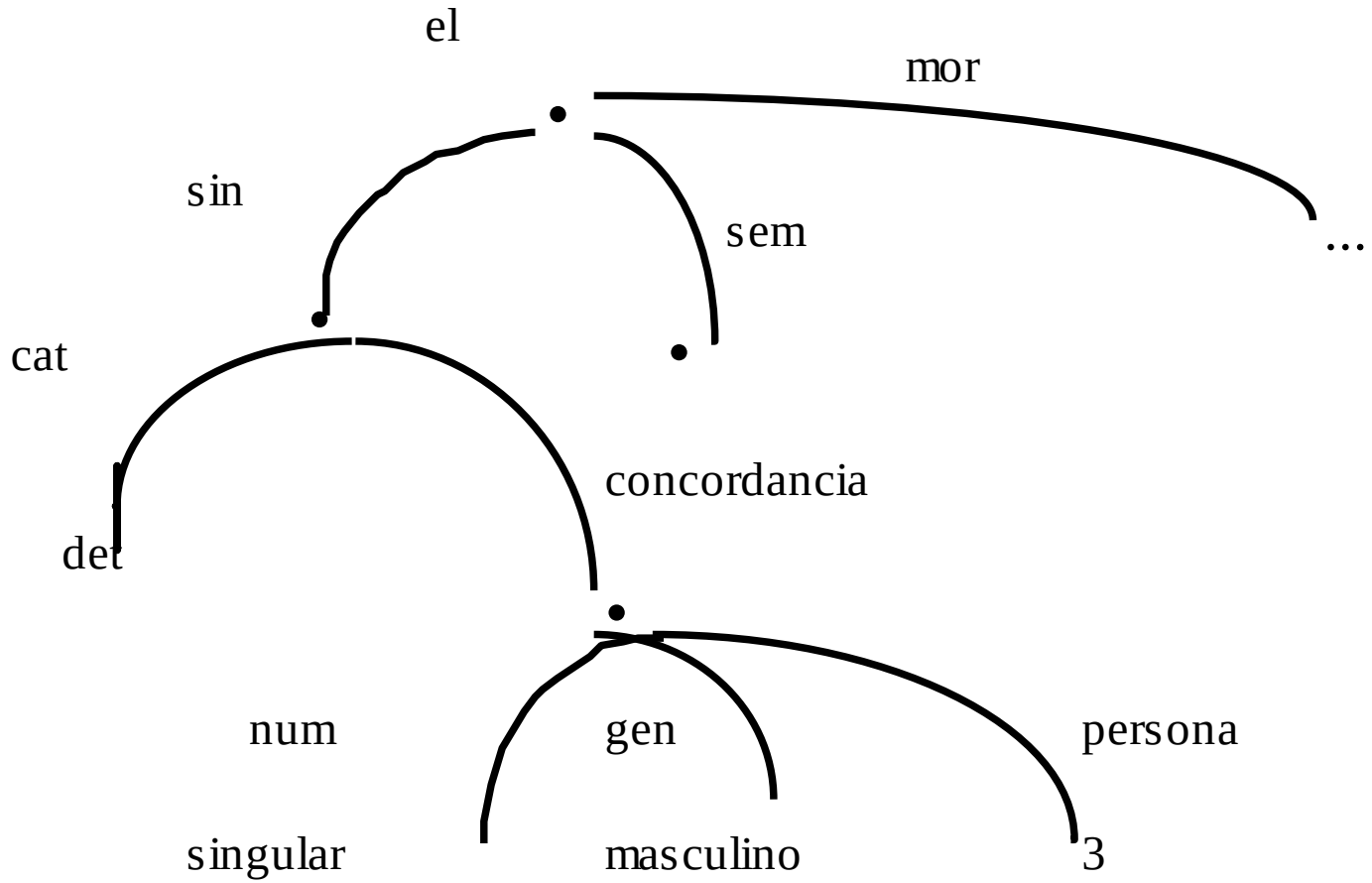
- Two families:
  - Open Feature Structures
  - Typed Feature Structures
    - every FS belongs to a type
    - The values allowed for assigning to a feature belong to a type.

Feature structures 2



Feature structures$_3$

FS as DAG (Directed Acyclic Graphs)

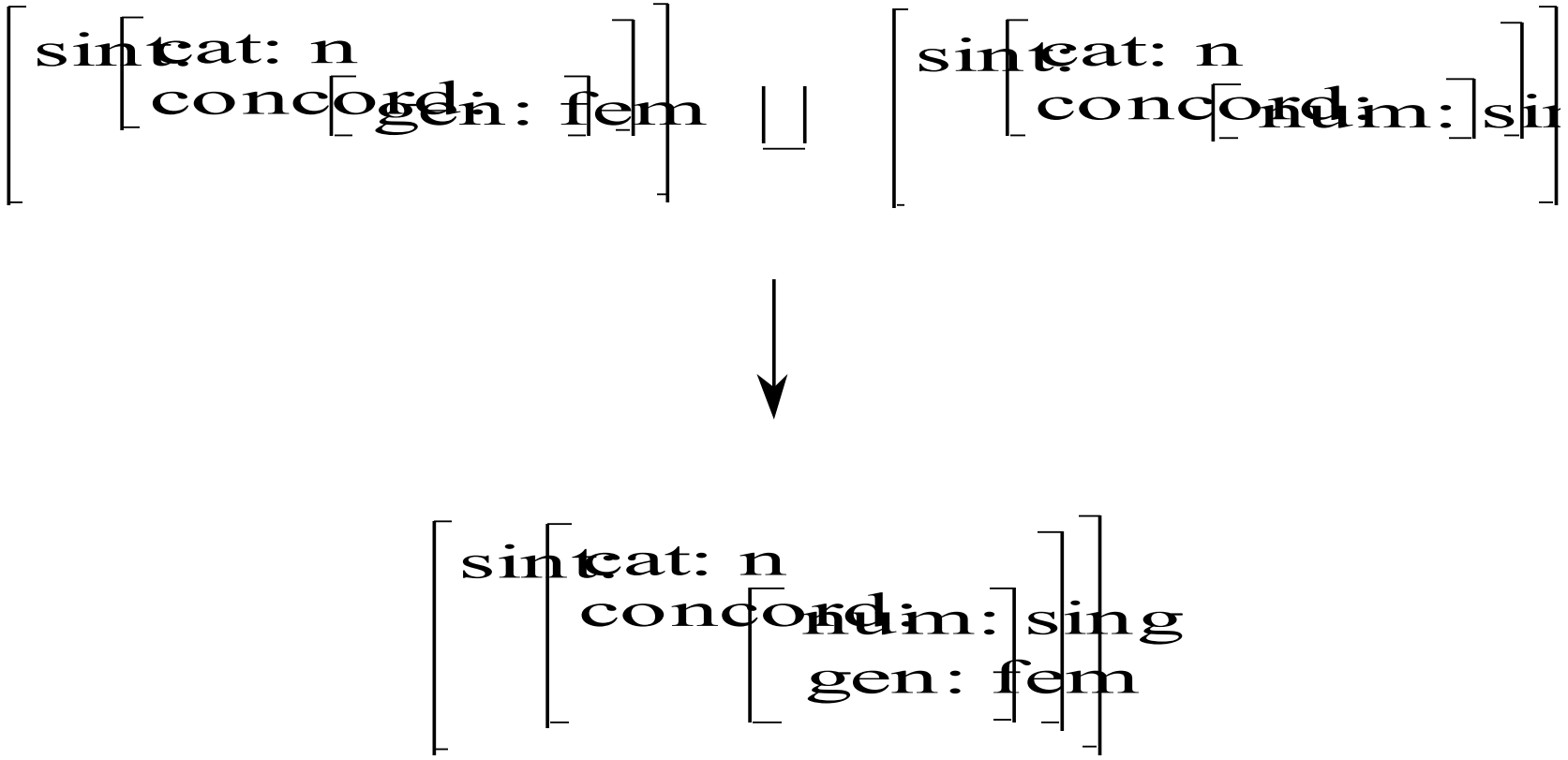


Feature structures$_4$

- 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

$$FS \sqcup F$$

Feature structures$_5$



Feature structures$_6$

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

the fish

cat(+,-,3,_,_,2)

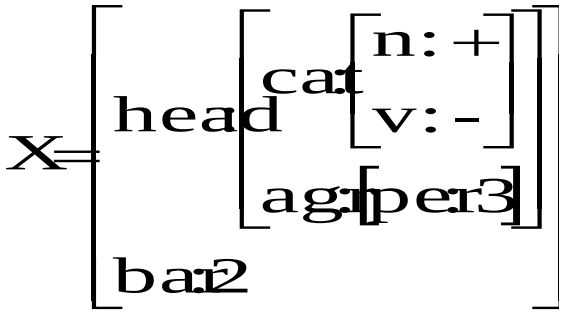
$$X = \begin{bmatrix} \text{head}: & \begin{bmatrix} \text{cat}: & \begin{bmatrix} n: + \\ v: - \end{bmatrix} \\ \text{agr}: & \begin{bmatrix} \text{per}: 3 \\ \text{num}: \text{pl} \end{bmatrix} \\ \text{case}: & \text{nom} \end{bmatrix} \\ \text{bar}: & 2 \end{bmatrix}$$

the fish are colourful

cat(+,-,3,pl,nom,2)

Feature structures 7

Parsing with open FS



the fish

cat(+,-,3,_,_,2)

X : head : cat : n <== '+'
X : head : cat : v <== '-'
X : head : agr : per <== 3
X : bar <== 2

Logical Grammars

- Most known system:
  - Definite clause grammars (DCG)

Composition of meaning from the meaning of parts

Adding **sem** feature. Example1

- S → NG loves NG
- S[sem=loves(x,y)] → NG[sem=x] loves NG[sem=y]
- NG [sem=x] -> propernoun(x)
- NG [sem=x] -> det noun(x)

  propernoun(George) -> George

  propernoun(Laura) -> Laura

  - `George loves Laura`
  - sem=loves(Laura)(George)

Composition of meaning from the meaning of parts

Adding **sem** feature. Example 2

- S[sem=vp(subj)] → NG[sem=subj] VG[sem=vp]
- NG[sem= subj] → propernoun(subj)
- VG[sem=v(obj)] → V[sem=v] NG[sem=obj]
- V[sem=loves] → loves
- propernoun(George) -> George
- propernoun(Laura) -> Laura
- `George loves Laura`
- sem=loves(Laura)(George)

- Simple semantic interpretation
  - IS bottom-up
  - Gramatic CNF. Each node two sons: 1 function & 1 argument

Feature structures 15

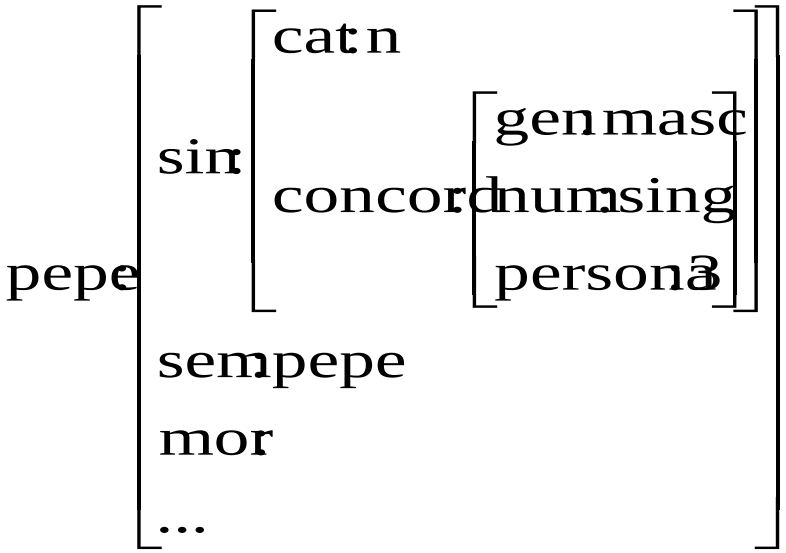
Rpresenting FS with open Prolog lists (Gazdar, Mellish)

```
[cat:      v
 arg0:     [cat:    sn
            case:    nom
            num:     sing]
]
```

```
[cat:v, arg0:[cat:sn, case:nom, num:sing|_]|_]
```

Feature structures 16

PatrII notation



word pepe:

<syn cat> = n
< syn agreement gen> = male
< syn agreement num> = sing
< syn agreement person> = 3
<sem> = pepe
...

Feature structures 17

PatrII notation

let noun-male-sing **be**:
<syn cat> = n
< syn agreement gen> = masc
< syn agreement num> = sing
< syn agreement person> = 3

word pepe: noun-male-sing
<sem> = pepe

Feature structures 18

PatrII inheritance

let verb **be**: <syn cat> = v
<syn subj cat> = ng
<syn subj case> = nominative

let vt **be**: verb
<syn obj1 cat> = ng
<syn obj1 case> = acusative

let vdat **be**: vt
<syn obj2 cat> = prepg
<syn obj2 prep> = to

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)

Feature structures 19

Syntactic rules in PatrII

```
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 >
```

Sense Desambiguation 1

- Senses
  - Different meanings of one word (word type) observable in different uses (word tokens)
- Word Sense Disambiguation (WSD)
  - Given a word (word token) in a context determine its correct sense

Sense Desambiguation 2

- Problems
  - Homonymy: meanings not related
  - Polisemy: related meanings
  - Metaphor
  - ¿Is it possible?

Sense Desambiguation 3

- A similar problem to POS tagging seambiguation
  - Similar approaches
  - A more complex problem
- Common restrictions
  - Yarowsky (1995)
    - One sense per discourse
    - One sense per collocation

Sense Desambiguation 4

A possible solution (Naive Bayes)

if **w** is the word to desambiguate

and c_k the possible senses

And $\vec{x}$ the context vector (i.e. a fix window of 100 words)

$$c' = \arg\max_{c_k} P(c_k \mid \vec{x})$$

We can apply Bayes theory, resulting:

Sense Desambiguation 5

$$c' = \arg\max_{c_k} P(c_k \mid \vec{x}) =$$

$$\arg\max_{c_k} \frac{P(\vec{x} \mid c_k)}{P(\vec{x})} \cdot P(c_k) =$$

$$\arg\max_{c_k} P(\vec{x} \mid c_k) \cdot P(c_k) =$$

$$\arg\max_{c_k} \left[\log P(\vec{x} \mid c_k) + \log P(c_k)\right] =$$

$$\arg\max_{c_k} \left[\sum_{v_j \ in \ \vec{x}} \log P(v_j \mid c_k) + \log P(c_k) \right]$$