

# Lecture Examples, 11/01/2006

## 1 Feature Structures

Feature structures (FS) map *features* into *values* (which are themselves feature structures). Below are examples of associating FS with words. We use the AVM representation (each 'raw' is a pair F:v, see discussion of AVMs in Section 2).

- (a) *lamb*  
 $\left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right]$
- (b) *lambs*  
 $\left[ \begin{array}{l} NUM : pl \\ PERS : third \end{array} \right]$
- (c) *fish*  
 $\left[ \begin{array}{l} NUM : [] \\ PERS : third \end{array} \right]$
- (d) *I*  
 $\left[ \begin{array}{l} NUM : sg \\ PERS : first \end{array} \right]$
- (e) *lambs*  
 $\left[ AGR : \left[ \begin{array}{l} NUM : pl \\ PERS : third \end{array} \right] \right]$
- (f) *you*  
 $\left[ \begin{array}{l} NUM : X([]) \\ PERS : second \end{array} \right]$
- (g) *you*  
 $\left[ \begin{array}{l} NUM : \boxed{[]} \\ PERS : second \end{array} \right]$

A special case of feature structures are *atoms* (also called atomic feature structures), which represent structureless values. For example, feature NUM has as its values the atoms *sg* or *pl* (in example (a) and (b)).

A feature can also have an *underspecified value*, represented as an empty feature structure (example (c)).

A feature can have as value a complex feature structure (e.g., AGR has as its value a complex FS in (e)).

We can assign variables to AVMs as in (f) and (g). We can either use the notation  $X$  or  $\boxed{a}$  to denote variables. Useful for value sharing (*reentrancy*): e.g.,

$$\begin{bmatrix} F : X(a) \\ G : X(a) \end{bmatrix}, \begin{bmatrix} F : \boxed{a} \\ G : \boxed{a} \end{bmatrix}.$$

## 2 Attribute-Value Matrices

AVMs are used to represent feature structures.

An AVM consists of a finite, possibly empty set of pairs, where each pair consists of a feature and a value. *Features* are drawn from a fixed (per grammar) pre-defined set FEATS; *values* can be either atoms (drawn from a fixed set ATOMS), or recursively, AVM themselves. Some AVMs are assigned variables. FEATS and ATOMS are parameters for the collection of AVMs, and are referred to as the *signature*.

If

$$A = \begin{bmatrix} F1 & A1 \\ \vdots & \\ Fn & An \end{bmatrix}$$

is a feature structure, then the *domain* of  $A$  is  $dom(A) = \{Fi | 1 \leq i \leq n\}$ . For every  $i \neq j$  we have that  $Fi \neq Fj$ . The *empty* AVM,  $[\ ]$ , has as its domain the empty set of features.

The *value* of a feature  $Fi \in A$  is  $val(A, Fi) = Ai$ .

A *path* is a (possibly empty) sequence of features that can be used to pick a value in a feature structure. Given the feature structure:

$$A = \left[ AGR : \begin{bmatrix} NUM : pl \\ PERS : third \end{bmatrix} \right]$$

the paths of  $A$  are  $\{\epsilon, \langle AGR \rangle, \langle AGR, NUM \rangle, \langle AGR, PERS \rangle\}$ , where  $\epsilon$  is the empty path. This notion is important as grammar constraints can be encoded as path equations (see discussion in J&M book about how this is done in the PATR formalism).

A value of a path,  $\pi$  is denoted by  $val(A, \pi)$ , and is a feature structure. The value for the empty path is  $val(A, \epsilon) = A$ , for every non-atomic feature structure  $A$ .

**Reentrancy.** Difference between *type-* and *token-* identity:

$$(1) \begin{bmatrix} F : a \\ G : a \end{bmatrix} \text{ and } (2) \begin{bmatrix} F : \boxed{a} \\ F : \boxed{a} \end{bmatrix}$$

Notion of reentrancy is extended to paths: two paths are said to be reentrant if they are associated with the same (token-identical) value.

$$\begin{array}{l} \left[ \begin{array}{l} F1 : \left[ G : \boxed{1} \left[ H : b \right] \right] \\ F2 : \left[ G : \boxed{1} \right] \end{array} \right] \\ \langle F1, G \rangle \text{ and } \langle F2, G \rangle \text{ are reentrant.} \end{array}$$

### 3 Subsumption

Feature structures represent information, and the amount of information stored within different feature structures can be compared, introducing a partial pre-order on the structures. This relation is called *subsumption*, and denoted by  $\sqsubseteq$ .

**Definition of subsumption:** Let  $A$  and  $B$  be two feature structures over the same signature. We say  $A$  *subsumes*  $B$  ( $A \sqsubseteq B$ ; also  $A$  is more general than  $B$ ; or  $A$  contains less information than  $B$ ) if the following conditions hold:

1. if  $A$  is an atom then  $B$  is an identical atom.
2. for every  $F \in FEATS$ , if  $F \in dom(A)$  then  $F \in dom(B)$  and  $val(A, F)$  subsumes  $val(B, F)$ ; and
3. if two paths are reentrant in  $A$  they are also reentrant in  $B$ .

**Examples.** Let us consider the following feature structures:

$$\begin{array}{l} A = [] \\ B = [NUM : sg] \\ C = [PERS : third] \\ D = \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \\ E = \left[ \begin{array}{l} CAT : vp \\ AGR : \boxed{1} \\ SUBJ : [AGR : \boxed{1}] \end{array} \right] \\ F = \left[ \begin{array}{l} CAT : vp \\ AGR : \boxed{1} \\ SUBJ : \left[ AGR : \boxed{1} \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \right] \end{array} \right] \end{array}$$

We have that:

$$\begin{array}{l} A \sqsubseteq B \sqsubseteq D \\ A \sqsubseteq C \sqsubseteq D \\ A \sqsubseteq E \sqsubseteq F \end{array}$$

$$B \not\sqsubseteq C$$

$$C \not\sqsubseteq B$$

The empty feature structure  $[]$  is the most general feature structure and subsumes all others (including atoms), as it encodes no information at all. For example  $[] \sqsubseteq [NUM : sg]$ . In the same way we can have  $[NUM : X] \sqsubseteq [NUM : sg]$  since by convention  $X$  is a shorthand for  $X[]$  (again  $X$ , or  $\square$  denotes variables). Subsumption is a partial pre-order since not every pair of feature structures is comparable:  $B \not\sqsubseteq C$ .

## 4 Unification

Unification is an information combination operation,  $\sqcup$ , and is defined over pairs of feature structures, yielding the most general feature structure that is most specific than both operands, if one exists.  $A = B \sqcup C$ , if  $A$  is the most general feature structure such that  $B \sqsubseteq A$  and  $C \sqsubseteq A$ . If such a structure exists, the unification *succeeds*, and the two arguments are said to be *unifiable (or consistent)*. Otherwise the unification *fails*, and the operands are said to be *inconsistent*. Failure is denoted by  $\top$ . (See J&M book for representation of feature structures as DAGs and the unification algorithm over DAGs).

### Examples

1. Unification combines consistent information:

$$[NUM : sg] \sqcup [PERS : third] = \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix}$$

2. Different atoms are inconsistent:

$$[NUM : sg] \sqcup [NUM : pl] = \top \text{ (i.e., unification fails)}$$

3. Atoms and non-atoms are inconsistent:

$$[NUM : sg] \sqcup sg = \top \text{ (i.e., unification fails)}$$

4. Unification is absorbing (i.e., if  $A \sqsubseteq B$ , then  $A \sqcup B = B$ ):

$$[NUM : sg] \sqcup \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix} = \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix}$$

5. Empty feature structures are identity elements:

$$[] \sqcup \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix} = \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix}$$

6. Reentrancy causes two consistent values to coincide:

$$\begin{bmatrix} F : [NUM : sg] \\ G : [PERS : third] \end{bmatrix} \sqcup \begin{bmatrix} F : \square \\ G : \square \end{bmatrix} = \begin{bmatrix} F : \square \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix} \\ G : \square \end{bmatrix}$$

7. Unification acts differently depending on whether the values are equal:

$$\begin{bmatrix} F : [NUM : sg] \\ G : [NUM : sg] \end{bmatrix} \sqcup [F : [PERS : third]] = \begin{bmatrix} F : \begin{bmatrix} NUM : sg \\ PERS : third \end{bmatrix} \\ G : [NUM : sg] \end{bmatrix}$$

... or identical:

$$\left[ \begin{array}{l} F : \boxed{1}[NUM : sg] \\ G : \boxed{1} \end{array} \right] \sqcup [F : [PERS : third]] = \left[ \begin{array}{l} F : \boxed{1} \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \\ G : \boxed{1} \end{array} \right]$$

8. Unification binds two variables together (i.e., they both share one and the same value which is the result of the unification). The *scope* of the variable must be taken into account: all other occurrences of the same variables in this scope are affected:

$$\begin{aligned} & \left[ F : \boxed{1}[NUM : sg] \right] \sqcup \left[ F : \boxed{2}[PERS : third] \right] = \\ = & \left[ F : \boxed{12} \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \right] = \left[ F : \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \right] \end{aligned}$$

(since variables  $\boxed{1}$  and  $\boxed{2}$  do not occur anywhere else we can omit them).

But if  $\boxed{1}$  and  $\boxed{2}$  occur elsewhere their values will be modified as a result of the unification:

$$\left[ \begin{array}{l} F : \boxed{1}[NUM : sg] \\ G : \boxed{1} \end{array} \right] \sqcup [F : \boxed{2}[PERS : third]] = \left[ \begin{array}{l} F : \boxed{1} \left[ \begin{array}{l} NUM : sg \\ PERS : third \end{array} \right] \\ G : \boxed{1} \end{array} \right]$$

## 5 Adding features to rules

When a feature is assigned to a non-terminal symbol C (or category) it means that this feature is appropriate for all the phrases of category C.

Generalized categories have a base category and an associated feature structure:

$$\begin{array}{c} NP \\ \left[ \begin{array}{l} NUM : \boxed{\phantom{0}} \\ PERS \boxed{\phantom{0}} \end{array} \right] \end{array} .$$

Grammar rules (productions) are augmented with structural information by assigning a feature structure to every nonterminal symbol in the CFG skeleton of the production.

## 5.1 Number agreement

$$\begin{array}{lcl}
 N & \rightarrow & lamb \\
 [NUM: X] & & [NUM: X(sg)] \\
 N & \rightarrow & lambs \\
 [NUM: X] & & [NUM: X(pl)] \\
 S & \rightarrow & NP \quad VP \\
 & & [NUM: X] [NUM: X] \\
 NP & \rightarrow & D \quad N \\
 [NUM: X] & & [NUM: X] [NUM: X]
 \end{array}$$

Using the path equation constraints as used in PATR, and given in J&M book we have:

$$\begin{array}{l}
 N \rightarrow lamb \\
 \langle N \text{ NUM} \rangle = sg \\
 S \rightarrow NP VP \\
 \langle NP \text{ NUM} \rangle = \langle VP \text{ NUM} \rangle \\
 NP \rightarrow D N \\
 \langle NP \text{ NUM} \rangle = \langle N \text{ NUM} \rangle \\
 \langle D \text{ NUM} \rangle = \langle N \text{ NUM} \rangle
 \end{array}$$

## 5.2 Subcategorization

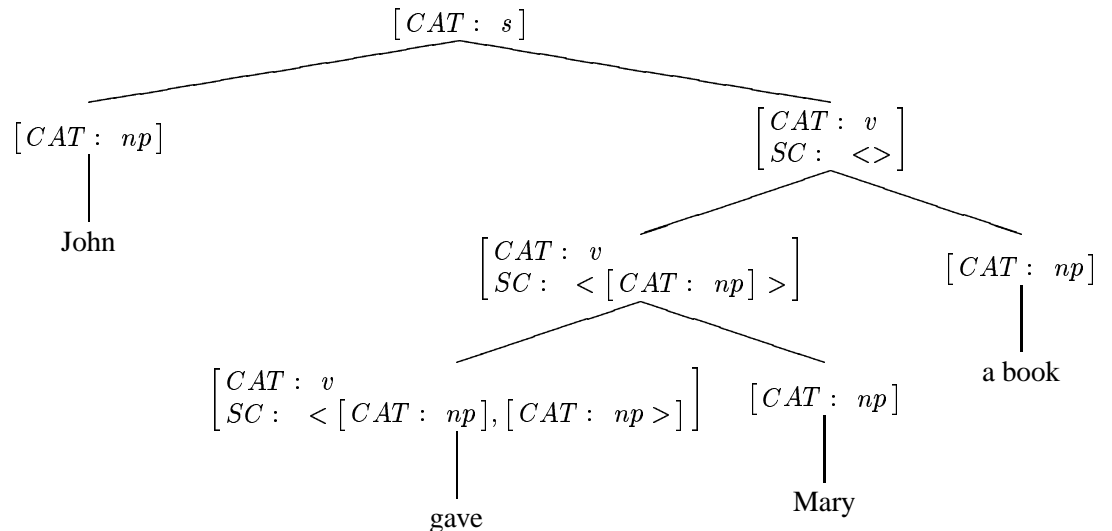
We can add a feature SUBCAT that can take the values *intrans*, *trans* etc.

$$\begin{array}{lcl}
 VP & \rightarrow & V \\
 [NUM: X] & & \left[ \begin{array}{l} NUM: X \\ SUBCAT: intrans \end{array} \right] \\
 VP & \rightarrow & V \quad N \\
 [NUM: X] & & \left[ \begin{array}{l} NUM: X \\ SUBCAT: trans \end{array} \right] [NUM: Y]
 \end{array}$$

A more elegant solution is the use of *subcategorization lists* and internalized categories (i.e., the category becomes a feature inside the feature structure). We denote by  $\langle \rangle$  the empty list.

$$\begin{aligned}
\textit{swim} &\rightarrow \begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : pl \\ \textit{SUBCAT} : \langle \rangle \end{bmatrix} \\
\textit{loves} &\rightarrow \begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : sg \\ \textit{SUBCAT} : \langle [\textit{CAT} : np] \rangle \end{bmatrix} \\
\textit{gives} &\rightarrow \begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : sg \\ \textit{SUBCAT} : \langle [\textit{CAT} : np], [\textit{CAT} : np] \rangle \end{bmatrix} \\
[\textit{CAT} : s] &\rightarrow \begin{bmatrix} \textit{CAT} : np \\ \textit{NUM} : X \end{bmatrix} \begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : X \\ \textit{SUBCAT} : \langle \rangle \end{bmatrix} \\
\begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : X \\ \textit{SUBCAT} : Y \end{bmatrix} &\rightarrow \begin{bmatrix} \textit{CAT} : v \\ \textit{NUM} : X \\ \textit{SUBCAT} : \begin{bmatrix} \textit{FIRST} : Z \\ \textit{REST} : Y \end{bmatrix} \end{bmatrix} Z(\[])
\end{aligned}$$

An example of a parse tree for the sentence *John gave Mary a book* is given below:<sup>1</sup>



### 5.3 Long distance dependencies

Example: *She wondered whom John loved \_*

<sup>1</sup>We use SC as a shorthand for SUBCAT

Implement long distance dependencies by introducing a new feature GAP (or SLASH) signaling that a category (in this case a noun phrase) is missing.

Given the rules :

$$\begin{aligned}
 [CAT : s] &\rightarrow \begin{bmatrix} CAT : np \\ NUM : X \end{bmatrix} \begin{bmatrix} CAT : v \\ NUM : X \end{bmatrix} \\
 \begin{bmatrix} CAT : v \\ NUM : X \end{bmatrix} &\rightarrow \begin{bmatrix} CAT : v \\ NUM : X \\ SUBCAT : trans \end{bmatrix} \begin{bmatrix} CAT : np \\ NUM : Y \end{bmatrix}
 \end{aligned}$$

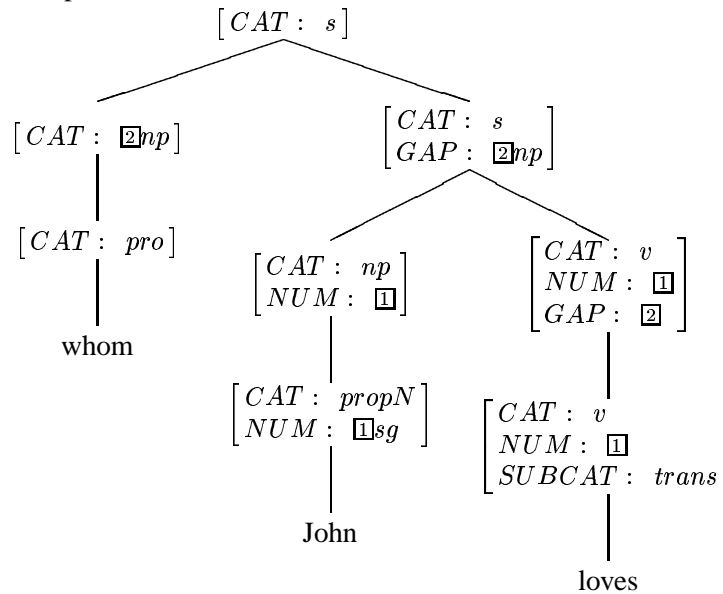
We add two additional rules that will allow us to derive partial phrases such as *John loves \_*

$$\begin{aligned}
 \begin{bmatrix} CAT : s \\ GAP : Z \end{bmatrix} &\rightarrow \begin{bmatrix} CAT : np \\ NUM : X \end{bmatrix} \begin{bmatrix} CAT : v \\ NUM : X \\ GAP : Z \end{bmatrix} \\
 \begin{bmatrix} CAT : v \\ NUM : X \\ GAP : np \end{bmatrix} &\rightarrow \begin{bmatrix} CAT : v \\ NUM : X \\ SUBCAT : trans \end{bmatrix}
 \end{aligned}$$

We then need a rule to create 'complete' sentences by combining the missing category with a 'gapped'(or, 'slashed') sentence (e.g., *whom John loves \_*).

$$[CAT : s] \rightarrow [CAT : Z] \begin{bmatrix} CAT : s \\ GAP : Z \end{bmatrix}$$

Example of parse tree for *whom John loves*:



**State (item form)**  $[A \rightarrow \alpha \bullet \beta, i, j, Dag_{A1}]$

### Inference Rules

<b>Prediction</b>	$\frac{[A \rightarrow \alpha \bullet B \beta, i, j, Dag_{A1}]}{[B \rightarrow \bullet \gamma, j, j, Dag_{B1}]} \langle B \rightarrow \gamma, Dag_{B1} \rangle$
<b>Scanning</b>	$\frac{[A \rightarrow \alpha \bullet B \beta, i, j, Dag_{A1}]}{[B \rightarrow word_j \bullet, j, j+1, Dag_{B1}]} \langle B \rightarrow word_j, Dag_{B1} \rangle$
<b>Completion</b>	$\frac{[A \rightarrow \alpha \bullet B \beta, i, j, Dag_{A1}] \quad [B \rightarrow \gamma \bullet, j, k, Dag_{B1}]}{[A \rightarrow \alpha B \bullet \beta, i, k, Dag_{A1} \sqcup_c Dag_{B1}]}$

Table 1: Earley parser for unification grammars<sup>2</sup>

## 6 Earley parser

Table 1 shows the states and inference rules of Earley parser for unification grammars. There are three main modifications to Earley parser (used for CFGs) in order to handle unification.

1. Each *state* (or item form) is extended to include the left-hand side DAG (which can get augmented as it goes along), i.e., add a feature structure (in DAG form) to each state. Thus,  $[S \rightarrow \bullet NP VP, 0, 0]$  becomes  $[S \rightarrow \bullet NP VP, 0, 0, Dag]$ . If we have the constraint rule :

$$S \rightarrow NP VP \\ \langle NP NUM \rangle = \langle VP NUM \rangle$$

then  $Dag = \left[ S : \left[ NP : \left[ NUM : \boxed{\quad} \right] \right] \right]$ . The predictor, scanner, and completer have to pass in the DAG, so all three operations have to be altered.

2. The most important change appears in the Completion rule. The completer combines two rules and *unifies* the two feature structures associated with them. We denote by  $\sqcup_c$  the unification that implements copying of the initial feature structures (i.e. it is a non-destructive unification)! Let's assume we want to parse the sentence *these fish swim* and we have in the chart the following two states, corresponding to *these fish*, and *swim*, respectively:

<sup>2</sup>See the previously assigned reading (Shieber et al., 1995) for the general definition of parsing as deduction. The part in  $\langle \rangle$  represents *side conditions* which refer to the rules of a particular grammar.

$$[S \rightarrow NP \bullet VP, 0, 2, \left[ S : \left[ NP : \left[ \begin{array}{l} NUM : \boxed{1} \\ DET : \left[ \begin{array}{l} WORD : these \\ NUM \boxed{pl} \end{array} \right] \\ N : \left[ \begin{array}{l} WORD : fish \\ NUM : \boxed{1} \end{array} \right] \end{array} \right] \right] \right] ]$$

$$[VP \rightarrow V \bullet, 2, 3, \left[ VP : \left[ \begin{array}{l} NUM : \boxed{1} \\ V : \left[ \begin{array}{l} WORD : swim \\ NUM : \boxed{pl} \end{array} \right] \end{array} \right] \right] ]$$

Since the unification between the above two DAGs succeeds, we obtain the following state using the completion rule, which is then added to the chart:

$$[S \rightarrow NP VP \bullet, 0, 3, \left[ S : \left[ \begin{array}{l} NP : \left[ \begin{array}{l} NUM : \boxed{1} \\ DET : \left[ \begin{array}{l} WORD : these \\ NUM \boxed{pl} \end{array} \right] \\ N : \left[ \begin{array}{l} WORD : fish \\ NUM : \boxed{1} \end{array} \right] \end{array} \right] \\ VP : \left[ \begin{array}{l} NUM : \boxed{1} \\ V : \left[ \begin{array}{l} WORD : swim \\ NUM : \boxed{pl} \end{array} \right] \end{array} \right] \end{array} \right] \right] ]$$

Notice that if we wanted to parse the sentence *this fish swim* the unification would have failed (since the first DAG would have had the NUM set to **sg** while the second DAG would have had the NUM set to **pl**).

3. The third modification is to impose a restriction on which states get added to the chart, i.e., a modification to the Enqueue procedure in J&M book. A subsumption test should be used: do not add a state to the chart if an equivalent or more general state is already there (i.e., if the new state is subsumed by an old one, then it should not be added). For example, if Enqueue wants to add a singular noun state at  $[i, j]$  ( $\left[ \begin{array}{l} CAT : n \\ NUM : sg \end{array} \right]$ ), and the chart already has a noun state at  $[i, j]$  unspecified for number ( $\left[ \begin{array}{l} CAT : n \\ NUM : \boxed{\phantom{sg}} \end{array} \right]$ ), then Enqueue will not add it. If we do not impose a subsumption restriction, Enqueue will add two states at  $[i, j]$ , one expecting to see a singular noun, the other just a noun. On seeing a singular noun, the parser will advance the dot on both rules, creating two edges (since singular will unify with both singular and with unspecified). As a result, we would get duplicate edges. If we impose the restriction, and we see either a single or plural noun, and we advance the dot, only one edge (singular or plural) gets created at  $[i, j]$ .