



## CMPT-413: Computational Linguistics

Anoop Sarkar

<http://www.cs.sfu.ca/~anoop>

March 26, 2013

# Writing a grammar for natural language: Grammar Development

- ▶ **Grammar development** is the process of writing a grammar for a particular language
- ▶ This can be either for a particular application or concentrating on a particular phenomena in the language under consideration
- ▶ Check against text corpora to check the **coverage** of your grammar – to do this you need a parser
- ▶ Also consider generalizations provided by a linguistic analysis

# Real Grammars get Messy

- ▶ Task: Capture all the morphological details which affect the syntax of a language.
- ▶ The CFG ends up with rules like:

$S \rightarrow 3sgAux\ 3sgNP\ VP$

$S \rightarrow Non3sgAux\ Non3sgNP\ VP$

$3sgAux \rightarrow does\ |\ has\ |\ can\ |\ \dots$

$Non3sgAux \rightarrow do\ |\ have\ |\ can\ |\ \dots$

# Real Grammars get Messy

- ▶ This is to deal with sentences like:
  1. Do I get dinner on this flight ? (1sg = 1st person singular)
  2. Do you have a flight from Boston to Fort Worth ? (2sg = 2nd person singular)
  3. Does he visit Toronto ? (3sg = 3rd person singular)
  4. Does Delta fly from Atlanta to San Diego ? (3sg = 3rd person singular)
  5. Do they visit Toronto ? (3pl = 3rd person plural)

# Real Grammars get Messy

- ▶ Not just grammatical features but also subcategorization (what kind of arguments does a verb expect?):

*VP* → *Verb-with-NP-complement NP* “prefer a morning flight”

*VP* → *Verb-with-S-complement S* “said there were two flights”

*VP* → *Verb-with-Inf-VP-complement VP<sub>inf</sub>* “try to book a flight”

*VP* → *Verb-with-no-complement* “disappear”

# Solution to non-terminal and rule blowup: Feature Structures

- ▶ **Feature structures** provide a natural way to provide complex information with each non-terminal. In some formalisms, the non-terminal is replaced with feature structures, resulting in a potentially infinite set of non-terminals.
- ▶ Feature structures are also known as f-structures, feature bundles, feature matrices, functional structures, terms (as in Prolog), or dags (directed acyclic graphs)

# Feature Structures

- ▶ A *feature structure* is defined as a partial function from features to their values.
- ▶ For instance, we can define a function mapping the feature *number* onto the value *singular* and mapping *person* to *third*. The common notation for this function is:

$$\begin{bmatrix} \text{number: singular} \\ \text{person: 3} \end{bmatrix}$$

# Feature Structures

- ▶ Feature values can themselves be feature structures:

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \end{array} \right]$$



# Feature Structures

- ▶ Consider features  $f$  and  $g$  with two distinct feature structure values of the same type:

$$\left[ \begin{array}{l} f: \left[ h: a \right] \\ g: \left[ h: a \right] \end{array} \right]$$

# Feature Structures

- ▶ Feature structures can also share values. For instance,  $g$  shares the same value as  $f$  in:

$$\left[ \begin{array}{l} f: \boxed{1} \left[ h: a \right] \\ g: \boxed{1} \end{array} \right]$$

- ▶ The shared value is written using a co-indexation – indicating that the value is stored only once, with the index acting as a pointer.

# Feature Path Notation

- ▶ The feature structure:

$$\left[ \begin{array}{ll} \text{agreement:} & \boxed{1} \left[ \begin{array}{l} \text{number: sg} \\ \text{person: 3} \end{array} \right] \\ \text{subject:} & \left[ \text{agreement: } \boxed{1} \right] \end{array} \right]$$

is represented as:

`<agreement number>=sg`

`<agreement person>=3`

`<subject agreement>=<agreement>`

or:

`[ agreement = (1) [ number = 'sg', person = 3 ],`  
`subject = [ agreement->(1) ] ]`

or:

`[ agreement = ?n [ number = 'sg', person = 3 ],`  
`subject = [ agreement = ?n ] ]`

# Subsumption

- ▶ Feature structures have different amounts of information. Can we find an ordering on feature structures that corresponds to the compatibility and relative specificity of the information contained in them.
- ▶ **Subsumption** is a precise method of defining such an ordering over feature structures.

# Subsumption

- ▶ Consider the feature structure:

$$D_{np} = \left[ \text{cat: NP} \right]$$

- ▶ Compare with the feature structure:

$$D_{np3sg} = \left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \end{array} \right]$$

# Subsumption

- ▶  $D_{np}$  makes the claim that a phrase is a noun phrase, but leaves open the question of what the agreement properties of this noun phrase are.
- ▶  $D_{np3sg}$  also contains information about a noun phrase, but makes the agreement properties specific.
- ▶ The feature structure  $D_{np}$  is said to carry *less information* than, or to be *more general* than, or to *subsume* the feature structure  $D_{np3sg}$

# Subsumption

►  $D_{var} = []$

►  $D_{np} = [\text{cat: NP}]$

►  $D_{npsg} =$   

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } [\text{number: singular}] \end{array} \right]$$

►  $D_{np3sg} =$   

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \end{array} \right]$$

►  $D_{np3sgSbj} =$   

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \\ \text{subject: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \end{array} \right]$$

►  $D'_{np3sgSbj} =$   

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \boxed{1} \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \\ \text{subject: } \boxed{1} \end{array} \right]$$

- The following subsumption relations hold:

$$D_{var} \sqsubseteq D_{np} \sqsubseteq D_{npsg} \sqsubseteq D_{np3sg} \sqsubseteq D_{np3sgSbj} \sqsubseteq D'_{np3sgSbj}$$

# Unification

- ▶ Two feature structures might have different and incompatible information:

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \text{number: singular} \right] \end{array} \right]$$
$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \text{number: plural} \right] \end{array} \right]$$

- ▶ In this case, there is no feature structure that is subsumed by both feature structures



# Unification

- ▶ Subsumption is only a partial order – that is, not every two feature structures are in a subsumption relation with each other.
- ▶ Two feature structures might have different but compatible information:

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \text{number: singular} \right] \end{array} \right]$$
$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \text{person: 3} \right] \end{array} \right]$$

# Unification

- ▶ If two feature structures have different but compatible information then there always exists a more specific feature structure that is subsumed by both feature structures:

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \end{array} \right] \end{array} \right]$$

# Unification

- ▶ But there are many feature structures subsumed by both of the original feature structures:

$$\left[ \begin{array}{l} \text{cat: NP} \\ \text{agreement: } \left[ \begin{array}{l} \text{number: singular} \\ \text{person: 3} \\ \text{gender: masculine} \end{array} \right] \end{array} \right]$$

- ▶ So instead of considering all such feature structures we only consider the most general FS that is subsumed by the two original FSs
- ▶ This definition provides a feature structure that contains information from both input FSs but no additional information.

# Unification

- ▶ Now we can define **unification**
- ▶ The *unification* of two feature structures  $D'$  and  $D''$  is defined as the most general feature structure  $D$  such that  $D' \sqsubseteq D$  and  $D'' \sqsubseteq D$ .
- ▶ This operation of unification is denoted as  $D = D' \sqcup D''$

# Unification

$$\square \sqcup [\text{cat: NP}] = [\text{cat: NP}]$$

# Unification

$$\begin{bmatrix} \text{person: sg} \end{bmatrix} \sqcup \begin{bmatrix} \text{number: 3} \end{bmatrix} = \begin{bmatrix} \text{person: sg} \\ \text{number: 3} \end{bmatrix}$$

# Unification

$$\left[ \begin{array}{l} \text{agreement:} \left[ \text{number: sg} \right] \\ \text{subject:} \left[ \text{agreement:} \left[ \text{number: sg} \right] \right] \end{array} \right] \sqcup$$
$$\left[ \text{subject:} \left[ \text{agreement:} \left[ \text{person: 3} \right] \right] \right] =$$
$$\left[ \begin{array}{l} \text{agreement:} \left[ \text{number: sg} \right] \\ \text{subject:} \left[ \text{agreement:} \left[ \begin{array}{l} \text{number: sg} \\ \text{person: 3} \end{array} \right] \right] \end{array} \right]$$

# Unification

$$\left[ \begin{array}{l} \text{agreement: } \boxed{1} \left[ \text{number: sg} \right] \\ \text{subject: } \left[ \text{agreement: } \boxed{1} \right] \end{array} \right] \sqcup \left[ \text{subject: } \left[ \text{agreement: } \left[ \text{person: 3} \right] \right] \right] =$$
$$\left[ \begin{array}{l} \text{agreement: } \boxed{1} \left[ \begin{array}{l} \text{number: sg} \\ \text{person: 3} \end{array} \right] \\ \text{subject: } \left[ \text{agreement: } \boxed{1} \right] \end{array} \right]$$



# Algorithms for Unification

- ▶ Represent input feature structure as a directed acyclic graph (dag). Unification is equivalent to the **union-find** algorithm.
- ▶ Unification is more efficient if it can be destructive: it destroys the input feature structures to create the result of unification.
- ▶ The (destructive) unification algorithm in J&M (page 423) does it in two steps: represent feature structures as dags, and then perform graph matching (and merging)
- ▶ Note that this algorithm can produce as output a dag (i.e. a feature structure) containing cycles.

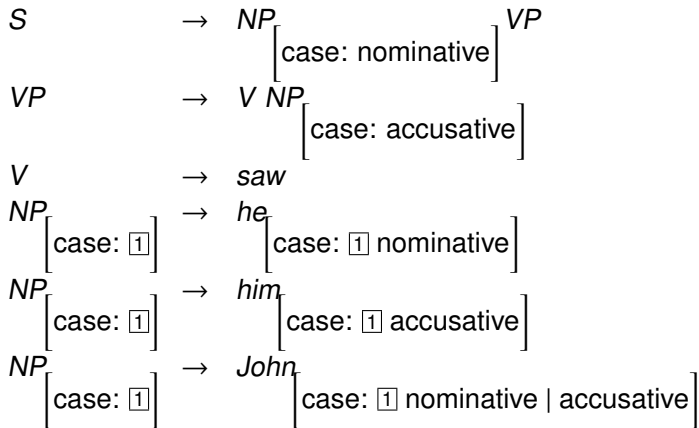
A feature structure can have part of itself as a subpart:

$$\left[ f: \boxed{1} \left[ g: \left[ h: \boxed{1} \right] \right] \right]$$

- ▶ This can be avoided with an explicit check for each call to the `unify` algorithm called the **occur check**.
- ▶ Computationally expensive since we have to traverse the whole dag at each step

# Feature Structures in CFGs

- ▶ Feature Structures impose constraints on CFG derivations:



- ▶ This CFG derives: *he saw him* but not: *\*him saw he*
- ▶ Also derives: *John saw him, he saw John.*
- ▶ Co-indexing in each FS is local to each CFG rule.

# Feature Structures in CFGs

- ▶ A more complex example for encoding subcategorization as feature structures:

$$\begin{array}{lcl}
 S & \rightarrow & NP \ VP \left[ \begin{array}{l} \text{subcat: } \boxed{1} \left[ \begin{array}{l} \text{first: } \boxed{\phantom{0}} \\ \text{rest: end} \end{array} \right] \end{array} \right] \\
 \\ 
 VP \left[ \begin{array}{l} \text{subcat: } \boxed{1} \end{array} \right] & \rightarrow & Verb \left[ \begin{array}{l} \text{subcat: } \boxed{1} \end{array} \right] \\
 \\ 
 VP \left[ \begin{array}{l} \text{subcat: } \boxed{1} \end{array} \right] & \rightarrow & VP \left[ \begin{array}{l} \text{subcat: } \left[ \begin{array}{l} \text{first: } \boxed{2} \\ \text{rest: } \boxed{1} \end{array} \right] \end{array} \right] X \left[ \begin{array}{l} \text{cat: } \boxed{2} \text{ NP} \end{array} \right]
 \end{array}$$

# Feature Structures in CFGs

- ▶ In the above example, the CFG can generate an arbitrary number of NPs in the subcat feature structure for the verb.
- ▶ In effect, the above steps of unification in a CFG derivation creates a list containing the subcat elements. The subcat feature structure uses **first** and **rest** to construct the list in the recursive rule  $VP \rightarrow VP X$ .
- ▶ The lexical terminal *Verb* can impose a constraint on which subcat frame is required.
- ▶ Other categories can be added simply by adding a new *cat* attribute for *X*: e.g.  $\left[ \text{cat: } S \right]$  for verbs that can have a subcat of *NP S*.

# Unification Algorithm

```
function unify(f1, f2):  
    returns f-structure or failure  
  
    if f1.content == null: f1.pointer = f2  
    if f2.content == null: f2.pointer = f1  
    if f1.content == f2.content: f1.pointer = f2  
    if f1.content and f2.content are complex f-structures:  
        f2.pointer = f1  
        for each f in f2.content:  
            other-feature = find or create feature  
                           corresponding to f in f1.content  
            if unify(f, other-feature) == failure:  
                return failure  
    return f1
```

# Unification in Earley Parsing

- ▶ predictor: if  $(A \rightarrow \alpha \bullet B \beta, [i, j], \text{dag}_{A_1})$  then  $\forall (B \rightarrow \gamma, \text{dag}_{B_1})$   
enqueue( $(B \rightarrow \bullet \gamma, [j, j], \text{dag}_{B_1}), \text{chart}[j]$ )
- ▶ scanner: if  $(A \rightarrow \alpha \bullet a \beta, [i, j], \text{dag}_{A_1})$  and  $a = \text{tokens}[j]$  then  
enqueue( $(A \rightarrow \alpha a \bullet \beta, [i, j + 1], \text{dag}_{A_1}), \text{chart}[j + 1]$ )
- ▶ completer: if  $(B \rightarrow \gamma \bullet, [j, k], \text{dag}_{B_1})$ , for each  
 $(A \rightarrow \alpha \bullet B \beta, [i, j], \text{dag}_{A_1})$   
enqueue( $(A \rightarrow \alpha B \bullet \gamma, [i, k], \text{copy-and-unify}(\text{dag}_{A_1}, \text{dag}_{B_1})), \text{chart}[k]$ )  
unless  $\text{copy-and-unify}(\text{dag}_{A_1}, \text{dag}_{B_1})$  fails
- ▶ copy-and-unify means that we make copies of the dags before unification because we are using a destructive unification algorithm
- ▶ copy-and-unify ensures that  $\text{dag } A_1$  in state  $(A \rightarrow \alpha \bullet B \beta, [i, j], \text{dag}_{A_1})$  is not destroyed since it can be used in the completer with other states and unify with them.

# Unification in Earley Parsing

- ▶ Consider two different enqueue requests:  
enqueue( $(A \rightarrow \alpha B \bullet \gamma, [i, k], \text{dag}_{A_1})$ , chart[k])  
enqueue( $(A \rightarrow \alpha B \bullet \gamma, [i, k], \text{dag}_{A_2})$ , chart[k])
- ▶ Consider the case where:  
 $\text{dag}_{A_1} = [\text{tense: past} \mid \text{plural}]$  and  
 $\text{dag}_{A_2} = [\text{tense: past}]$   
Clearly,  $\text{dag}_{A_1} \sqsubseteq \text{dag}_{A_2}$

# Unification in Earley Parsing

- ▶ Which feature structure should be selected after the two enqueue commands above?

Three options:  $\text{dag}_{A_1}$ ,  $\text{dag}_{A_2}$ ,  $\text{dag}_{A_1} \sqcup \text{dag}_{A_2}$

- ▶ In general, the feature inserted should subsume both  $\text{dag}_{A_1}$  and  $\text{dag}_{A_2}$
- ▶ In practice exactly one of the following conditions is always true:
  - ▶ If  $\text{dag}_{A_1} \sqsubseteq \text{dag}_{A_2}$  then enqueue picks  $\text{dag}_{A_1}$ ,
  - ▶ If  $\text{dag}_{A_2} \sqsubseteq \text{dag}_{A_1}$  then enqueue picks  $\text{dag}_{A_2}$ .
  - ▶ If  $\text{dag}_{A_1} \not\sqsubseteq \text{dag}_{A_2}$  and  $\text{dag}_{A_2} \not\sqsubseteq \text{dag}_{A_1}$  then enqueue picks  $\text{dag}_{A_1} \sqcup \text{dag}_{A_2}$



# Unification in Earley Parsing

- ▶ During the enqueue of a state, we always pick the most general feature structure possible.
- ▶ To see why consider an example:
  - ▶ Consider a chart which contains the state:  
 $S_1 = (NP \rightarrow \bullet DT NP, [i, i], dag_{S_1} = [])$
  - ▶ The parser then tries to enqueue a new state:  
 $S_2 = (NP \rightarrow \bullet DT NP, [i, i], dag_{S_2} = [DT.num = sing])$
  - ▶ Consider two possible situations:
    1. a singular DT is scanned, then either  $dag_{S_1}$  or  $dag_{S_2}$  would unify and parsing would continue.
    2. a plural DT is scanned, then if we picked  $dag_{S_2}$  we have a unification failure; on the other hand picking the more general feature structure  $dag_{S_1}$  allows parsing to continue.
- ▶ So, if there are two possible ways to derive a span, then the most general feature structure is the one we must choose.

# Summary

- ▶ Feature structures generalize the notion of non-terminals in a grammar.
- ▶ Complex morphological details can be encoded into a feature structure.
- ▶ Feature structures can have shared or co-referential parts.
- ▶ Feature structures can implement arbitrary lists (the notation is very computationally powerful).
- ▶ Unification provides a means to combine the information in two feature structures.
- ▶ Feature structures can be used in a context-free grammar, and
- ▶ Unification is done while parsing to ensure that the constraints specified in the features are not violated.