

# Feature Structures and How to Represent Multiple Phenomena Simultaneously

Adam Meyers  
New York University



# Outline

- Definitions and Examples
- Parsing with Feature Structures
- The Earley Algorithm
- Other Issues
- GLARF: a Feature Structure Project at NYU



# Why Feature Structures?

- A Feature Structure is a good data structure for representing complex objects
  - Can include many linguistic features in one structure: Tense, Agreement, Semantics, Parsed Structure, Coreference, ...
- Represents objects in terms of features value pairs, where the values of features can be complex
- The mathematics of Feature Structures were worked out in great detail in the 1980s and 1990s
- Several linguistic theories are formalized in terms of Feature Structures and operations thereon



# Defining Feature Structures

- A Feature Structure is either atomic or a set of feature value pairs
  - $FS \rightarrow NIL$
  - $FS \rightarrow Atom$
  - $FS \rightarrow \{FV_1, FV_2, \dots FV_N\}$
  - $FV \rightarrow Feature = FS$ 
    - **A values of a feature must be a FS**
- Each Feature and Value Represents a Piece of Information
- More information defines more specific objects



# A Simple Example

- $FS_1 = [\text{Color} = \text{Green}]$ 
  - Describes a green thing
- $FS_2 = [\text{Height} = \text{Tall}]$ 
  - Describes a tall thing
- $FS_3 = [\text{Color} = \text{Green}, \text{Height} = \text{Tall}]$ 
  - Describes a tall green thing
- More feature value pairs describe a more specific thing



# Typed Feature Structures

- Typed feature structures:
  - Every feature structure has a type
    - The type limits what are the possible features that can be included in it
  - Every feature has a type
    - The type limits its possible values
- Examples
  - A Feature Structure of type **Lego** allows features: color, height, width, depth and material.
  - The value of the feature **Color** allows atomic TFS as values from the set {**red**, **yellow**, **blue**, **green**, ...}



# Subsumption

- The operator  $\sqsubseteq$  represents “subsumes”
- $FS_1 \sqsubseteq FS_2$ , if  $FS_1$  describes the same or larger set of possible entities than  $FS_2$  does.
  - For example, if  $FS_1$  represents something green and  $FS_2$  represents a tall green thing, then  $FS_1 \sqsubseteq FS_2$
  - **[Color = Green]  $\sqsubseteq$  [Color = Green, Height = Tall]**
- Notice that if  $FS_1 \sqsubseteq FS_2$ , then  $FS_2$  includes all of the Feature Value pairs in  $FS_1$ , but the reverse may not be true.
- For typed feature structures, one must add information about type subsumption and this is essentially based on the definitions of types (similar to type inheritance in OOP)
  - I will leave out some of the details about types, but can talk more about them if there are questions.



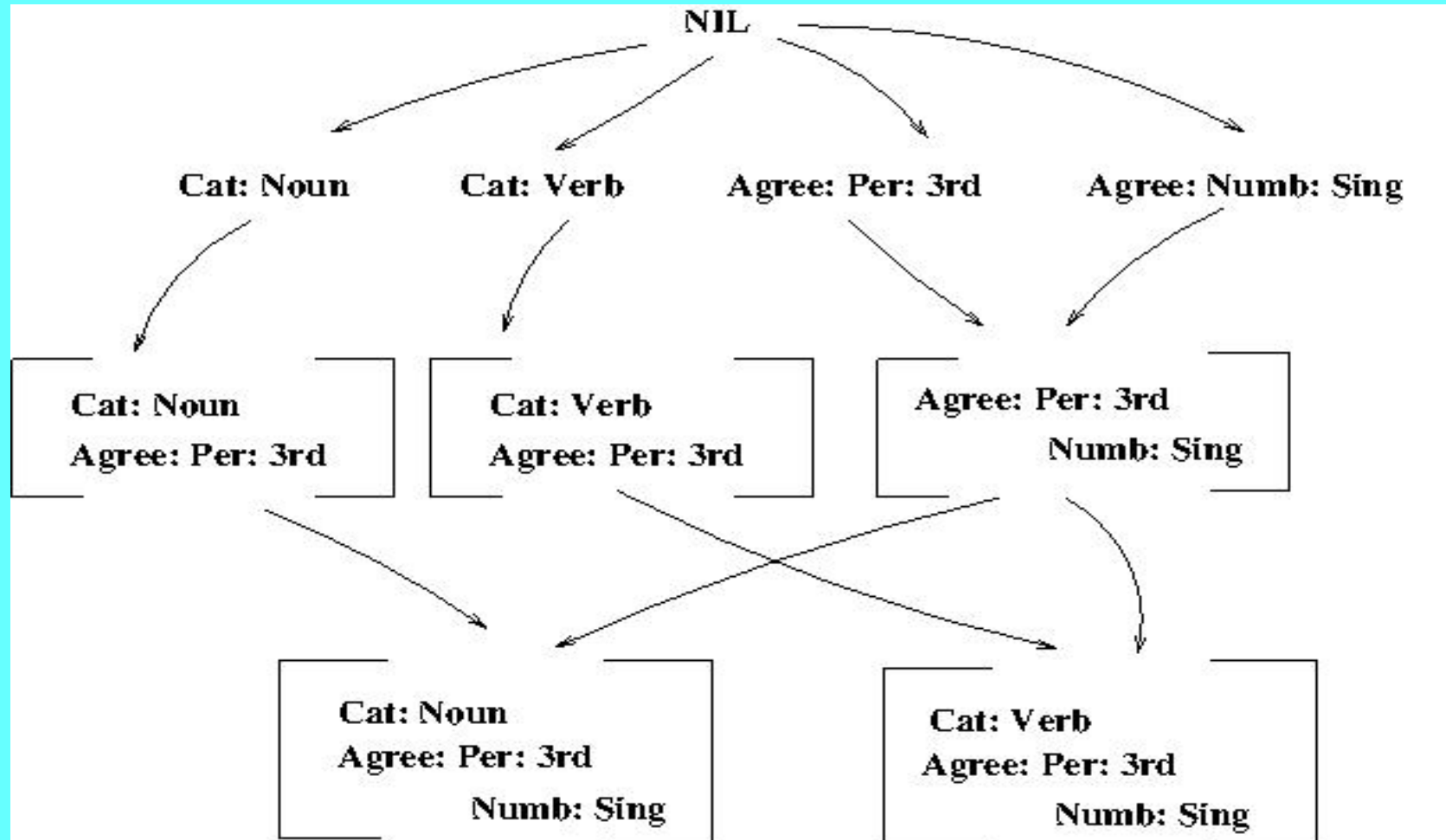
# Properties of Subsumption

- NIL is the most general feature structure
  - Subsumes every other feature structure
    - The set of zero feature value pairs
    - Also subsumes atomic feature structure
    - Possible value for all features (for typed feature structures)
- Subsumption is transitive
- If  $FS_1 \sqsubseteq FS_2$  and  $FS_2 \sqsubseteq FS_3$ , then  $FS_1 \sqsubseteq FS_3$
- Subsumption partially orders the set of all FS
  - NIL is the root of a DAG which includes all FSs
  - Edges in paths from the root represent subsumption





# Part of the Subsumption Graph for a FS-based Grammar of English



# Unification

- Unifying (operator =  $\sqcup$ ) two FSs combines the information in both feature structures to produce a FS that instantiates the intersection of entities that the two input FSs instantiate
- $FS_1 \sqcup FS_2 = FS_3$  iff  $FS_3$  is the most general Feature structure (the one with the fewest Feature Value pairs) such that:
  - $FS_1 \sqsubseteq FS_3$  and  $FS_2 \sqsubseteq FS_3$
- Properties:
  - Unification is Commutative
    - $FS_1 \sqcup FS_2 = FS_2 \sqcup FS_1$
  - Unification is Associative
    - $(FS_1 \sqcup FS_2) \sqcup FS_3 = FS_1 \sqcup (FS_2 \sqcup FS_3)$



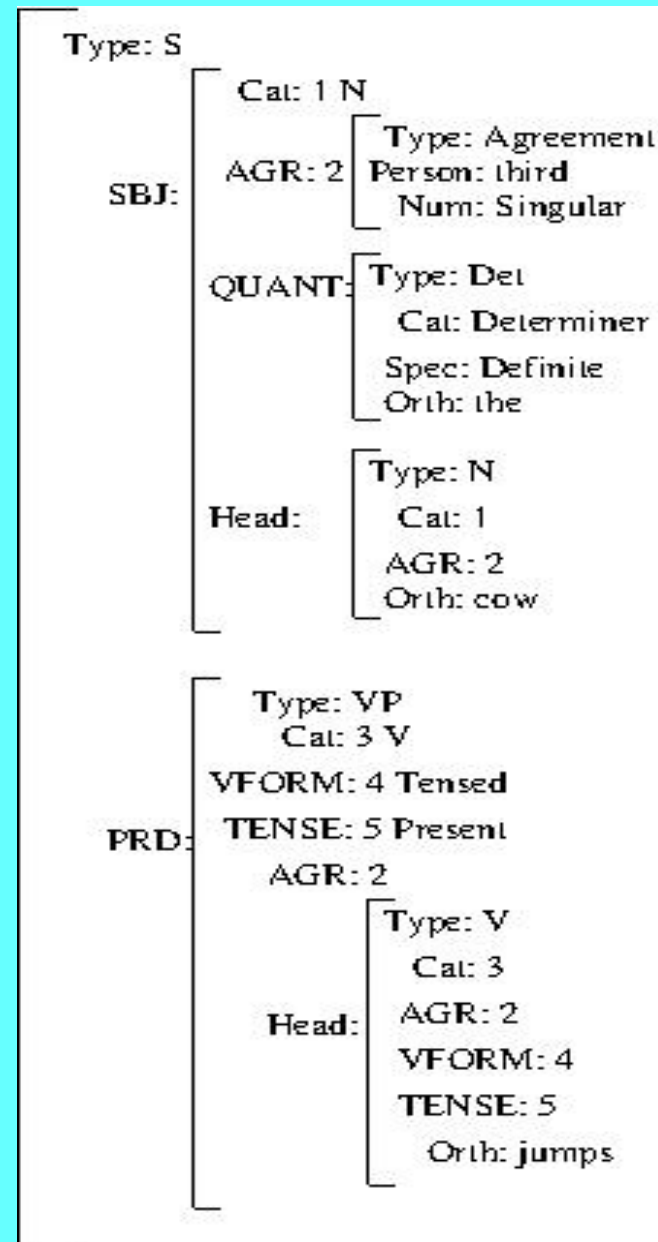
# How to Unify (not worrying about efficiency)

- $FS_x \sqcup NIL \rightarrow FS_x$
- $NIL \sqcup FS_x \rightarrow FS_x$
- $Atom_1 \sqcup Atom_2$  Fails if ***Atom<sub>1</sub> ≠ Atom<sub>2</sub>***
- To Unify Complex FSs  $FS_1$  and  $FS_2$ , producing  $FS_3$ , start with an empty  $FS_3$  and add FVs as follows:
  - For each Feature Value Pair  $FV_1$  in  $FS_1$ , try to find a matching  $FV_2$  in  $FS_2$  such that Feature  $F_1$  in  $FV_1$  is the same as  $F_2$  in  $FV_2$ 
    - If no matching feature exists, then add  $FV_1$  into  $FS_3$
    - Otherwise, try to unify  $V_1$  in  $FV_1$  with  $V_2$  in  $FV_2$ 
      - If Fail, then unification fails
      - Otherwise, add  $F$  with a value of  $V_1 \sqcup V_2$  to  $FS_3$
  - For each  $FV_x$  in  $FS_2$  that did not match any Feature in  $FV_1$ :
    - Add  $FV_x$  to  $FS_3$



# FS in Bracket Notation representing *The cow jumps*

- Indices represent shared structure
- The first feature taking a shared structure as a value is followed by a numbered index and the structure
- Other features sharing that structure are followed by that index

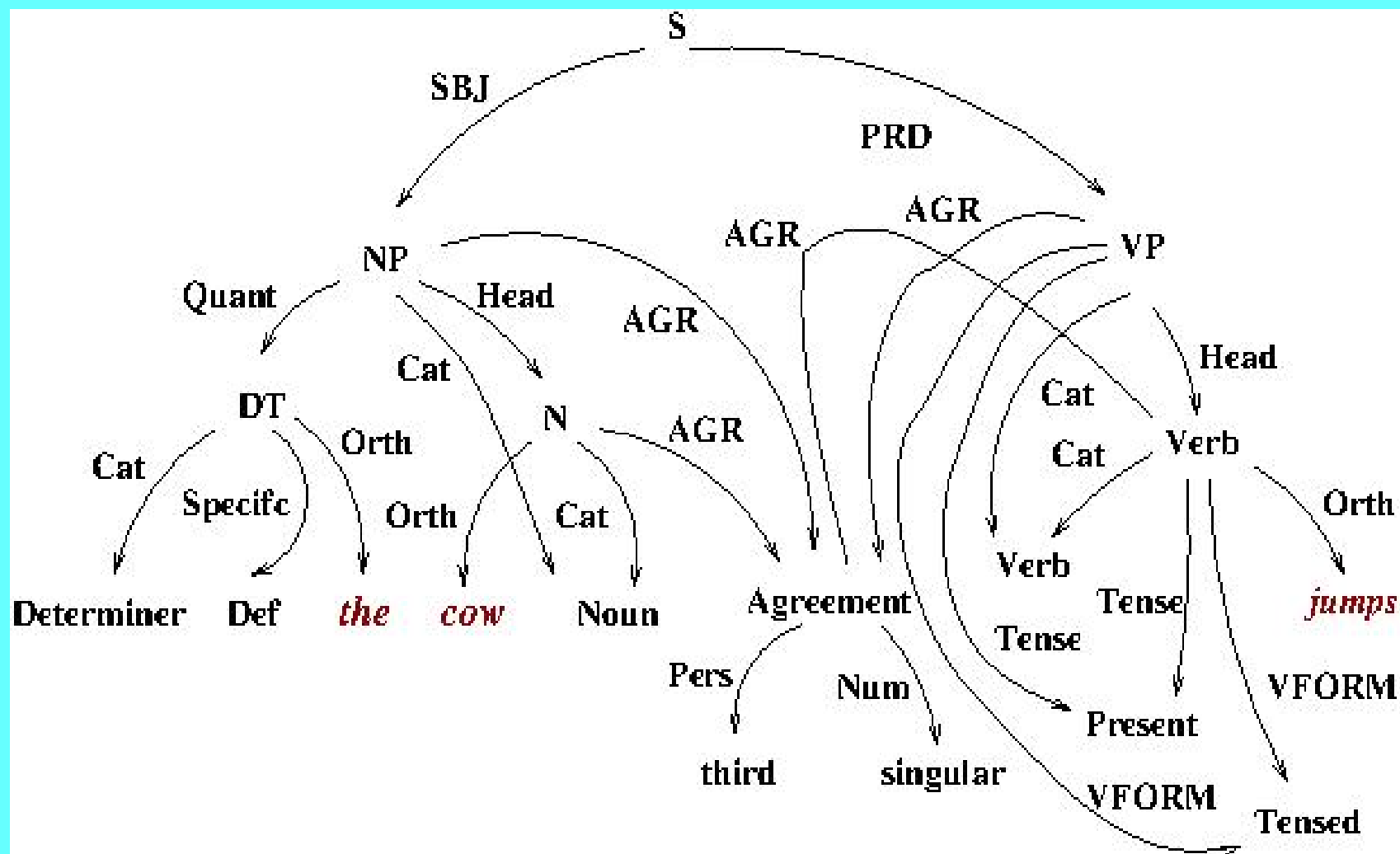


# Feature Structures as Edge-Labeled DAGs

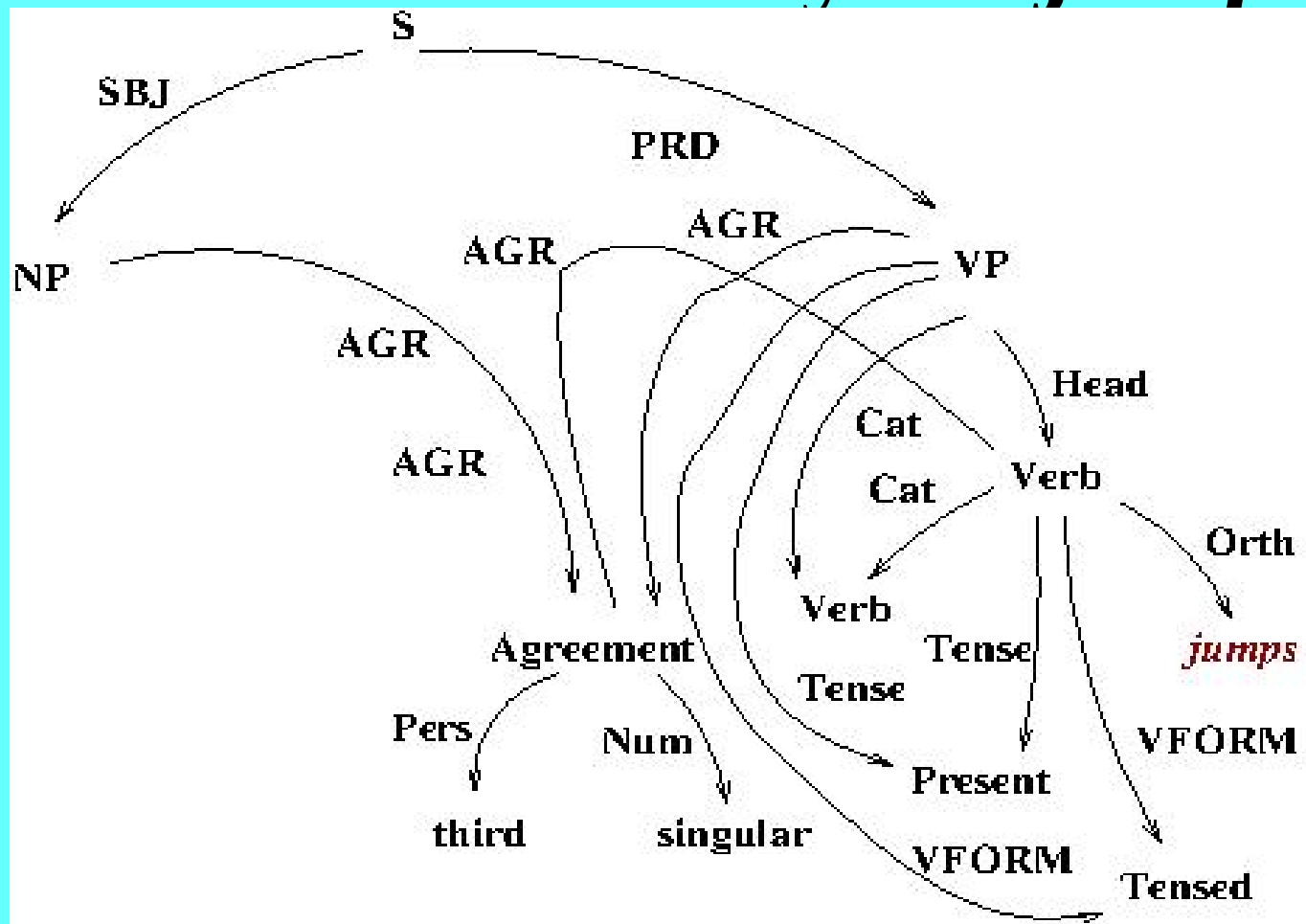
- Types = Internal Nodes = Non Terminals = Phrasal Categories and Parts of Speech
- Atomic FSs = leaves
- Features = Edge Labels
- Shared Structure is determined by grammar
  - It means that some features values are exactly the same
  - Common Instances
    - Shared between a phrase and its head
    - Agreement between a subject and a verb



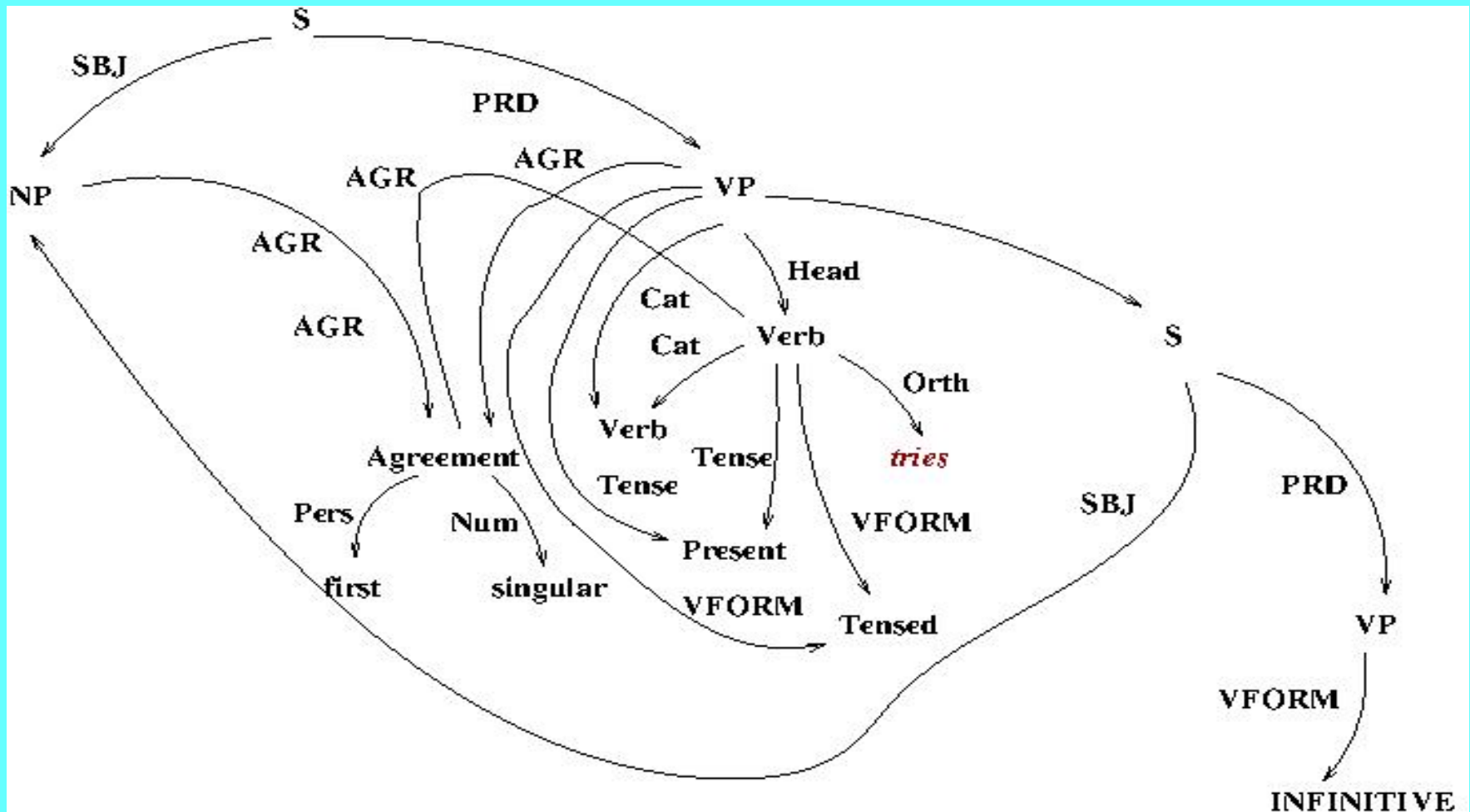
# DAG representing *The cow jumps*



# FS for lexical entry for *jumps*

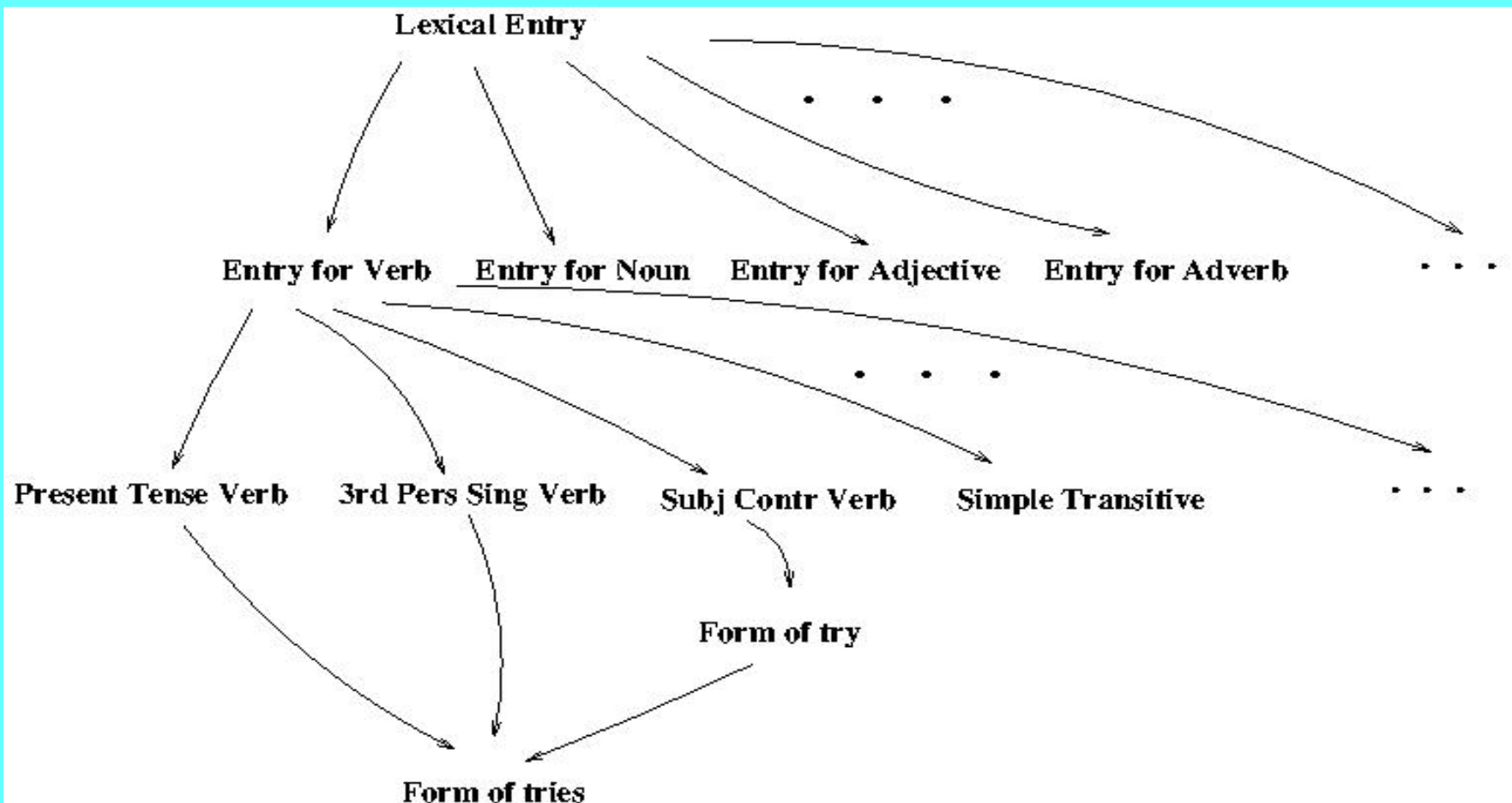


# FS Lexical Entry for the Verb *tries*





# Lexicon Can Be Arranged Hierarchically, based on Subsumption



# How Can We Use FSs for Parsing?

- For each word, we look up all its feature structure entries (instead of looking up its possible parts of speech)
  - These FSs or generalizations of these feature structures can correspond to either:
    - Initial Terminal Symbols, e.g., FS representing a noun
    - Initial NonTerminal Symbols, e.g., FS representing an S licensed by a verb
- Do we Need Context Free Grammars?
  - Using the second type of entries, it is possible to (in a way) fold the entire grammar into the lexicon
  - Alternatively, a context free grammar can be used to guide the combination of FSs, as in standard parsing
    - FSs constrain possible combinations



# The Earley Algorithm

- Shortcoming of Top Down Parsing
  - Left Recursive rules like  $\mathbf{NP} \rightarrow \mathbf{NP} \ \mathbf{PP}$
  - If  $\mathbf{NP}$  is recognized, productions starting with  $\mathbf{NP}$  are added to chart including this rule which starts with  $\mathbf{NP}$  (hence infinite recursion)
- The Earley Algorithm solves this problem:
  - it avoids adding duplicate productions to the chart
- Productions  $\mathbf{XP} \rightarrow \mathbf{X}_1 \cdot \mathbf{X}_2 \mathbf{X}_3[i,j]$  in the chart include:
  - A phrase structure rule ( $\mathbf{XP} \rightarrow \mathbf{X}_1 \cdot \mathbf{X}_2 \mathbf{X}_3$ )
  - A dot (between  $\mathbf{X}_1$  and  $\mathbf{X}_2$ ) such that complete constituents to the left of the dot have been matched
  - The span of text that this rule applies to between  $i$  and  $j$
- The Earley algorithm would not add  $\mathbf{NP} \rightarrow \mathbf{NP} \cdot \mathbf{PP} [0,1]$ 
  - If there was already an instance in the chart



# FS version of the Earley Algorithm

- We assume the model in which phrase structure rules guide combination of FSs
  - A parsing step combines 1 complete and 1 incomplete states
    - A state is complete if the dot is all the way to the right
      - $XP \rightarrow X_1 X_2 X_3 \cdot$
    - An incomplete state has the dot somewhere else
      - $YP \rightarrow W_1 \cdot XP Z_3$
  - The result combines the two by matching the complete state with the symbol following the dot and then advancing the dot
    - $YP \rightarrow W_1 XP \cdot Z_3$
- For the FS version, matching is based on subsumption
  - Matching for purposes of a parsing step (above)
  - When checking if a production is already in the chart (previous slide)



# Efficiency Issues for FS Parsing

- Efficient unification changes input FSs
  - Combining them destructively keeping parts of each
- For chart parsing, original FSs are needed
  - So FS parsing involves lots of copying (this can be inefficient)
- Solutions
  - Use general FSs in productions that subsume “real ones”
    - Generate final FS after final parse is found
  - Lazy copying (Godden 1990)
    - Use instruction like “copy FS<sub>1</sub>” to delay copying
    - Then copy only when FS is actually needed



# Linguistic Theories Using Feature Structures as Models

- Generalized Phrase Structure Grammar
  - <http://www.amazon.com/Generalized-Phrase-Structure-Grammar-Gerald/dp/0674344561>
- Head Driven Phrase Structure Grammar
  - <http://www.ling.ohio-state.edu/research/hpsg/>
- Lexical Function Grammar
  - <http://www2.parc.com/isl/groups/nltt/papers/kb82-95.pdf>
- Categorical Unification Grammar
  - <http://acl-arc.comp.nus.edu.sg/archives/acl-arc-090501d3/data/pdf/anthology-PDF/C/C86/C86-1045.pdf>



# Other Books about Feature Structures and Related Issues

- The Logic of Typed Feature Structures (B. Carpenter)
  - <http://www.amazon.com/The-Logic-Typed-Feature-Structures/dp/0521022541>
- Mathematical Methods in Linguistics (Partee, Meulen and Wall)
  - [http://books.google.com/books/about/Mathematical\\_Methods\\_in\\_Linguistics.html?id=qV7TUuaYcUIC](http://books.google.com/books/about/Mathematical_Methods_in_Linguistics.html?id=qV7TUuaYcUIC)



# GLARF

- See CUNY talk





# Readings

- J & M Chapters 13.4.2 and 15

