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 → NIL
  – FS → Atom
  – FS → \{FV_1, FV_2, \ldots FV_N\}
  – FV → 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, \ldots\}
Subsumption

• The operator $\sqsubseteq$ represents “subsumes”
• $\text{FS}_1 \sqsubseteq \text{FS}_2$, if $\text{FS}_1$ describes the same or larger set of possible entities than $\text{FS}_2$ does.
  
  – For example, if $\text{FS}_1$ represents something green and $\text{FS}_2$ represents a tall green thing, than $\text{FS}_1 \sqsubseteq \text{FS}_2$

  – $[\text{Color} = \text{Green}] \sqsubseteq [\text{Color} = \text{Green, Height} = \text{Tall}]$
• Notice that if $\text{FS}_1 \sqsubseteq \text{FS}_2$, than $\text{FS}_2$ includes all of the Feature Value pairs in $\text{FS}_1$, 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)
Properties of Subsumption (See next Slide)

- 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 \( FS_1 \subseteq FS_2 \) and \( FS_2 \subseteq FS_3 \), then \( FS_1 \subseteq FS_3 \)
- 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 = ⊔) 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.

• $\text{FS}_1 \sqcup \text{FS}_2 = \text{FS}_3$ iff $\text{FS}_3$ is the most general Feature structure (the one with the fewest Feature Value pairs) such that:
  - $\text{FS}_1 \subseteq \text{FS}_3$ and $\text{FS}_2 \subseteq \text{FS}_3$

• Properties:
  - Unification is Commutative
    • $\text{FS}_1 \sqcup \text{FS}_2 = \text{FS}_2 \sqcup \text{FS}_1$
  - Unification is Associative
    • $(\text{FS}_1 \sqcup \text{FS}_2) \sqcup \text{FS}_3 = \text{FS}_1 \sqcup (\text{FS}_2 \sqcup \text{FS}_3)$
How to Unify (not worrying about efficiency)

- \( FS_x \sqcup NIL \rightarrow FS_x \)
- \( NIL \sqcup FS_x \rightarrow FS_x \)
- \( \text{Atom}_1 \sqcup \text{Atom}_2 \) Fails if \( \text{Atom}_1 \neq \text{Atom}_2 \)
- 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 the recursive call to unification Fails, then the larger unification fails as well
      - 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 $NP \rightarrow NP \; PP$
  – If $NP$ is recognized, productions starting with $NP$ are added to chart including this rule which starts with $NP$ (hence infinite recursion)

• The Earley Algorithm solves this problem:
  – it avoids adding duplicate productions to the chart

• Productions $XP \rightarrow X_1 \cdot X_2 \cdot X_3[i,j]$ in the chart include:
  – A phrase structure rule ($XP \rightarrow X_1 \cdot X_2 \cdot X_3$)
  – A dot (between $X_1$ and $X_2$) 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 $NP \rightarrow NP \cdot PP[0,1]$
  – 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
      – $XP \rightarrow X_1 X_2 X_3$.
    • 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
  – Match for purposes of a parsing step (above)
  – Match to check 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\_1” to delay copying
    • Then copy only when FS is actually needed
Linguistic Theories Using Feature Structures as Models

• Generalized Phrase Structure Grammar

• 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

• Categorial Unification Grammar
Other Books about Feature Structures and Related Issues

- The Logic of Typed Feature Structures (B. Carpenter)
- Mathematical Methods in Linguistics (Partee, Meulen and Wall)
  - 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