Features and Augmented Grammars

Prof. Ahmed Rafea
Feature Systems and Augmented Grammar

• Number agreement
• Subject-verb agreement
• Gender agreement for pronouns
• To handle such phenomena, the grammatical formalism is extended to allow constituent to have features
  – E.g. NP \(\rightarrow\) ART N only when NUMBER1 agrees with NUMBER2
• This one rule is equivalent to two CFG rules:
  – NP-SING \(\rightarrow\) ART-SING N-SING
  – NP-PLURAL \(\rightarrow\) ART-PLURAL N-PLURAL
• Using these two rules, all grammar rules that have NP on its RHS would need to be duplicated to include a rule for NP-SING, and a rule for NP-PLURAL
• Using other features would double the size of the grammar again and again
Feature Systems and Augmented Grammar

• To add features to constituents, the constituent is defined as feature structure
  – E.g. a feature structure for the a constituent ART1 that represents a particular use of the word a might be written as follows:
    – ART1: (ART ROOT a NUMBER s)
• Feature structures can be used to represent larger constituent as well
  – E.g. NP1: (NP NUMBER s 1(ART ROOT a NUMBER s) 2 (N ROOT fish NUMBER s))
• Note that this can also be viewed as a representation of a parse tree
• The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations
  – E.g. (NP NUMBER ?n) → (ART NUMBER ?n) (N NUMBER ?n)
• Variables are also useful in specifying ambiguity in a constituent
  – E.g. (N ROOT fish NUMBER ?n) as fish could be singular and plural
• Constraints variables can also be used to specify a range of values
  – E.g. (N ROOT fish NUMBER ?n {s,p}) or simply (N ROOT fish NUMBER {s,p})
Some Basic Feature Systems for English

• It is convenient to combine Person and Number features into a single feature, AGR, that has six possible values, 1s, 2s, 3s, 1p, 2p, and 3p
  – E.g. “is” can agree with 3s and “are” can agree with {2s 1p 2p 3p}
• Verb-Form Features VFORM, base, present, past, finite {pres past}, ing, pastprt (past participle), and infinitive.
• To handle the interaction between words and their complements, an additional feature Verb Sub categorization, SUBCAT is used, _none, _np, _np_np, _vp:inf……..(see figures 4.2 and 4.4 in the James Allen Book)
  – (VP) → (V SUBCAT _np_vp:inf) (NP)(VP VFORM inf) e.g. Jack told the man to go
• Many verbs have complement structures that require a prepositional phrase with a particular preposition, therefore PFORM feature is introduced, TO, LOCation (in, on,..), MOTion (from, along..)
  – (VP) → (V SUBCAT _np_pp:loc) (NP)(PP PFORM LOC) e.g. Jack put the box in the corner
• Binary Features such as INV that indicates weather or not an S structure has an inverted subject
  – E.g. The S structure for the sentence Jack laughed will have INV – and Did jack laugh? Will have INV +
• It will be useful on many occasions to allow default value for features
Morphological Analysis and the Lexicon

• Store the base form of the verb in the lexicon and use context-free rules to combine verbs with suffixes
  - E.g. \((V \text{ ROOT } ?r \text{ SUBCAT } ?s \text{ VFORM } \text{ pres } \text{ AGR } 3s)\)
    \[\to (V \text{ ROOT } ?r \text{ SUBCAT } ?s \text{ VFORM } \text{ base}) (+S)\]
    Where +S is a new lexical category contains only the suffix morpheme –s

• This rule coupled with the lexical entry
  want: \((V \text{ ROOT want SUBCAT } _{np\_vp:inf} \text{ VFORM } \text{ base})\)
  would produce the following constituent given the input string want –s
  want: \((V \text{ ROOT want SUBCAT } _{np\_vp:inf} \text{ VFORM } \text{ pres } \text{ AGR } 3s)\)

• Another rule would generate the constituent for the present tense form not in third person singular, which for most verbs is identical to the root form:
  - \((V \text{ ROOT } ?r \text{ SUBCAT } ?s \text{ VFORM } \text{ pres } \text{ AGR } \{1s 2s 1p 2p 3p\})\)
    \[\to (V \text{ ROOT } ?r \text{ SUBCAT } ?s \text{ VFORM } \text{ base})\]
    This does not work for irregular verbs
A Simple Grammar Using Features

1. S[-inv] → (NP AGR ?a)(VP [{pres past}]AGR ?a)
2. NP → (ART AGR ?a)(N AGR ?a)
3. NP → PRO
4. VP → V[_none]
5. VP → V[_np] NP
7. VP → V[_np_vp:inf]NP VP[inf]
8. VP → V[_adjp] ADJP
9. VP[inf] → TO VP[base]
10. ADJP → ADJ
11. ADJP → ADJ[_vp:inf] VP[inf]

Head Features for S, VP: VFORM, AGR
Head Features for NP: AGR
Sample Parse Tree with Feature values

S[3s]
  /   \\nNP[3s] VP[pres,3s]
   / \\     \\     \\    \\
PRO[3s] V[pres,3s,_vp:inf] VP[inf]
      / \\     \\   \\  \\
He  wants TO  to  V[base,_adjp]
       / \\   / \\
      be  to  happy ADJP
Parsing with Features

Given an arc A, where the constituent following the dot is called next, and a new constituent X, which is being used to extend the arc,

a. Find an instantiations of the variables such that all the features specified in NEXT are found in X.

b. Create a new arc A’, which is a copy of A except for the instantiations of the variables determined in step (a).

c. Update A’ as usual in a chart parser.

For instance let A be the following arc

(NP AGR ?a) → @ (ART AGR ?a)(N AGR ?a)

and X be

(ART ROOT a AGR 3s)

and NEXT be

(ART AGR ?a)

In step a NEXT is matched against X, and you find that ?a must be instantiated to 3s . In step b a new copy of A is made (ART AGR 3s) and in step c the arc is updated to produce a new arc (NP AGR 3s) → (ART AGR 3s) @(N AGR 3s)
Introduction to PROLOG

• Computation is a proof: using the set of facts and deduction rules to verify whether a goal predicate is true
• A goal is true if there is an instantiation of the variables by which it can be deduced from the database
• Search for a proof using DFS is built into the PROLOG interpreter
• Example: The “program”:
  sibling(X,Y):- parent(Z,X),parent(Z,Y).
  parent(tom,bob).
  parent(tom,jane).

The goal: \(\text{?- sibling(bob,jane)}\) will return “true”
The goal: \(\text{?- sibling(tom,jane)}\) will return “false”
The goal: \(\text{?- sibling(X,Y)}\) will return sibling(bob,jane)
Logic Grammar

• Simple Top-down Parser is trivial to implement
• Create a database of the grammar rules, for example:
  
  \[\begin{align*}
  s(I_1, I_2) & :\text{ } np(I_1, I_3), vp(I_3, I_2). \\
  np(I_1, I_2) & :\text{ } det(I_1, I_3), n(I_3, I_2). \\
  vp(I_1, I_2) & :\text{ } v(I_1, I_2). \\
  det(I_1, I_2) & :\text{ } \text{word}(X, I_1, I_2), \text{isdet}(X). \\
  n(I_1, I_2) & :\text{ } \text{word}(X, I_1, I_2), \text{isnoun}(X). \\
  v(I_1, I_2) & :\text{ } \text{word}(X, I_1, I_2), \text{isverb}(X). \\
  \text{isdet}(a). \\
  \text{isdet}(the). \\
  \text{isnoun}(man). \\
  \text{isverb}(cried). \\
  \end{align*}\]

/* The sentence “a man cried” can be introduced as the following facts:
   word(a, 1, 2).
   word(man, 2, 3).
   word(cried, 3, 4).

The goal: ?- s(1, N) will return N=4 in the above example

• Parsing is done via the PROLOG DFS search for a proof for the goal!
Logic Grammar Using Difference Lists

- Difference list is an explicit argument pair
- Efficient for list operations
- \([1,2,3]\) is the difference between
  - \([1,2,3,4,5]\) and \([4,5]\)
  - \([1,2,3,8]\) and \([8]\)
  - \([1,2,3]\) and []
  - \([1,2,3|\text{AnyTail}]\) and \text{AnyTail}
- \(s(L_1,L) :- \text{np}(L_1,L_2), \text{vp}(L_2,L).\)
  - The difference between \(L_1\) and \(L\) is a sentence if the difference between \(L_1\) and \(L_2\) is a NP and the difference between \(L_2\) and \(L\) is a VP.
Example

s(L1,L) :- np(L1,L2), vp(L2,L).
np(L1,L) :- d(L1,L2), n(L2,L).
vp(L1,L) :- v(L1,L2), np(L2,L).
d(L1,L) :- connects(L1,the,L).
d(L1,L) :- connects(L1,a,L).
n(L1,L) :- connects(L1,dog,L).
n(L1,L) :- connects(L1,cat,L).
n(L1,L) :- connects(L1,gardener,L).
n(L1,L) :- connects(L1,policeman,L).
n(L1,L) :- connects(L1,butler,L).
v(L1,L) :- connects(L1,chased,L).
v(L1,L) :- connects(L1,saw,L).
connects([X|Y],X,Y).

?-s([the,gardener,saw,a,policeman],[]).

• Notice the way terminals of the grammar are treated
• An auxiliary predicate \texttt{connects(A,B,C)} is used to check if the difference of \texttt{A} and \texttt{C} is equal to \texttt{B}.
Adding Feature Structures

Add the features as arguments to the predicates:
Example:
S(I1, I2, Agr) :- NP(I1, I3, Agr), VP(I3, I2, Agr)
NP(I1, I2, Agr) :- det(I1, I3, Agr), n(I3, I2, Agr)
det(I1, I2, Agr) :- word(X, I1, I2), isdet(X, Agr)
isdet(a, 3s)
isdet(the, 3s)
isdet(the, 3p)
The input: word(a, 1, 2) word(man, 2, 3) ...
The goal: ?- S(1, N, Agr)
The feature structures can be represented in the form of a list:
Example: [[agr A] [vform V]...]
Grammar rules can then look like:
S(I1, I2, FS) :- NP(I1, I3, FS), VP(I3, I2, FS)
Adding Parse Tree Construction

Add an extra argument variable to the predicates for the parse tree:

Example:

\[ S(I_1, I_2, FS, s(Np, Vp)) :\] \[ NP(I_1, I_3, FS, Np), VP(I_3, I_2, FS, Vp) \]
\[ NP(I_1, I_2, FS, np(D, N)) :\] \[ det(I_1, I_3, FS, D), n(I_3, I_2, FS, N) \]
\[ det(I_1, I_2, FS, d(X)) :\] \[ word(X, I_1, I_2), isdet(X, FS) \]
\[ isdet(a, 3s) \]
\[ isdet(the, 3s) \]
\[ isdet(the, 3p) \]

The input: \[ word(a, 1, 2) \] \[ word(man, 2, 3) \] ... 

The goal: \[ ?- S(1, K, FS, PT) \]

Produced parse tree will look something like:

\[ s(np(d(a), n(man)), vp(v(cried))) \]
Notational Conventions for Writing DCGs

- Terminals are enclosed by list-brackets;
- Nonterminals are written as ordinary compound terms or constants
- The functor ‘,’/2 (comma) separates terminals and nonterminals in the right-hand side of rules;
- The functor '-->'/2 separates the left- and right-hand sides of a production rule;
- The empty string is denoted by the empty list.
- The functor ‘;'/2 (semicolon) separates terminals and nonterminals in the right-hand side of rules and means ‘or’.
- A plain Prolog goal is enclosed in braces, such as {write(‘Found NP’)}
Expansion to Prolog

- The translator automatically turns DCG rules into Prolog clauses by adding two extra arguments to every symbol that is not in braces.
- The arguments are automatically arranged for parsing using difference lists; e.g.
  \[ s \rightarrow np, \text{vp.} \]
  \[ s(L_1,L) \leftarrow np(L_1,L_2), \text{vp}(L_2,L) . \]
  \[ n \rightarrow [\text{dog}] \]
  \[ n([\text{dog} | L],L) . \]
- The built-in predicate `expand_term` is responsible for expanding DCG rules, among others, to Prolog rules.
- Try `?- expand_term((s --> np, vp),X).`
Example: Case Marking

• A Pronoun vary in case. It has a different form before the verb (nominative) than after it (accusative).
  – Nominative: he, she, they
  – Accusative: him, her, them

• Examples:
  – He sees him.
  – She sees her.
  – They see them.

• This case marking constraints can also be handled in DCG by adding an argument to the grammar rules:
  – S --> NP VP % introduces a nominative
  – VP --> V NP % introduces an accusative

• NP need to be changed to account for case making of pronouns.

Chapter 4  NLP Course
A Parser with case marking & agreement

n(singular) --> [dog];[cat];[mouse].
n(plural) --> [dogs];[cats];[mice].
v(singular) --> [chases];[sees].
v(plural) --> [chase];[see].
pronoun(singular,nominative) --> [he];[she].
pronoun(singular,accusative) --> [him];[her].
pronoun(plural,nominative) --> [they].
pronoun(plural,accusative) --> [them].
d --> [the].
np(Number,Case) --> pronoun(Number,Case).
np(Number,_), --> d, n(Number).
vp(Number) --> v(Number), np(_,accusative).
s --> np(Number,nominative), vp(Number).

Goal:
?- s([[the,dog,chases,him]],[]). (succeeds)
?- s([[the,dog,chases,he]],[]). (fails)
?- s([[the,dog,chases,Whom]],[]). % Whom = him; her; them;
Example: Subcategorization

- The structure of English VP depends on the verb form.
- Different verbs require different complements:

<table>
<thead>
<tr>
<th>Verb</th>
<th>Complement</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>none</td>
<td>the cat slept.</td>
</tr>
<tr>
<td>Chase</td>
<td>NP</td>
<td>the dog chased the cat</td>
</tr>
<tr>
<td>Sell</td>
<td>NP_NP</td>
<td>Max sold Bill his car.</td>
</tr>
<tr>
<td>Claim</td>
<td>S</td>
<td>Max claimed the cat barked.</td>
</tr>
</tbody>
</table>

- Instead of “verb” form, we will use 4 (or even more) subcategories that are grouped under the category verb:
- In DCG, structure of the VP is determined by adding an argument to the verb—dividing it into subcategories.
Parser that enforces Subcategorization

s  -->  np,  vp.
np  -->  d,  n.
d  -->  [the];[a].
n  -->  [dog];[cat];[gardener];[policeman];[butler].
vp  -->  v(1).
vp  -->  v(2),  np.
vp  -->  v(3),  np,  np.
vp  -->  v(4),  s.
v(1)  -->  [barked];[slept].
v(2)  -->  [chased];[saw].
v(3)  -->  [gave];[sold].
v(4)  -->  [said];[thought].

Goals:
?-  s([the,gardener,thought,the,dog,chased,a,cat],[]).  (succeeds)
?-  s([the,gardener,slept,the,dog,chased,a,cat],[]).  (fails)
Separating Lexicon from PS rules

• Using DCG rules for each type of words (noun, verb, etc.) makes grammar large.
• Alternative is to represent the lexicon as facts:
  – Searching is faster because of the indexing mechanism.
  – Words can be defined by rules:
    
    \[ X \text{ is a noun:} \]
    
    \[ \text{if it is in the lexicon as a noun, or} \]
    
    \[ \text{if it ends in "ness", and its stem is in the lexicon as an adjective (e.g. blueness, flatness, etc.)} \]
Implementation of a lexical rule

n --> [X], { noun(X) }.
noun(dog).
noun(cat).
noun(gardener).
noun(N) :-
    name(N,Nchars),
    append(Achars,"ness",Nchars),
    name(A,Achars),
    adjective(A).
adj --> [X], { adjective(X) }.
adjective(flat).
adjective(green).
adjective(blue).

Goals:
?- n([dog],[]).
?- n([flatness],[]).
Unification of Feature Structures

• Unification Grammars (such as LFG) establish a complete linguistic theory for a language via a set of relationships between feature structures of constituents.

• Key concept - extension relationship between two FSs:
  – F2 extends F1 if every feature-value pair in F1 is also in F2.

• Two FSs: F1 and F2 unify if there exists a FS F that is an extension of both of them.

• The Most General Unifier is the minimal FS F that extends both.

• The Unification operation allows easy expression of grammatical relationships among constituent feature structures.
Unification of Feature Structures

- **F 2 extends F1:**
  - F1 = ((cat *v))
  - F2 = ((cat *v) (root *cry))

- **F3 is MGU of F1 and F2:**
  - F1 = ((cat *v) (root *cry))
  - F2 = ((cat *v) (vform *pres))
  - F3 = ((cat *v) (root *cry) (vform *pres))

- **F1 and F2 do not unify:**
  - F1 = ((cat *v) (agr *3s))
  - F2 = ((cat *v) (agr *3p))
Unification of Feature Structures

• Feature Structures are commonly represented as DAGs
• Unification between FSs can be described as an operation that constructs a new DAG representing the unified structure
CFG Parsing with Feature Unification

• *Back-bone* CFG is augmented with a functional description that describes unification constraints between grammar constituents
• The FS corresponding to the “root” of the grammar is constructed compositionally during parsing
• The Algorithm for building the FS of a LHS constituent of a grammar rule: For a rule \( X_0 \rightarrow X_1X_2 \)
  1. Create a new FS labeled \( X_0 \)
  2. For each equation of the form \( x_n = x_m \)
     (i) Follow the DAG links of \( x_n \) and \( x_m \) to obtain the two values
     (ii) Unify the two values
     (iii) Modify the two original FSs to point to the new unified FS
A Unification Grammar Example

1. $S \rightarrow NP\ VP$  \hspace{1cm} AGR=AGR1=AGR2  
   \hspace{1cm} VFORM=VFORM2
2. $NP \rightarrow ART\ N$  \hspace{1cm} AGR=AGR1=AGR2
3. $VP \rightarrow V\ ADJP$  \hspace{1cm} SUBCAT1= _adj  
   \hspace{1cm} VFORM=VFORM1  
   \hspace{1cm} AGR=AGR1
4. $ADJP \rightarrow ADJ$
Example of a DAG of a NP

The graph for the NP The fish