Features and Unification
Chapter 15

Lecture #10

October 2012

1

Context Free Grammars

- We have been introduced to the notion of a context free grammar for capturing English constructions.
  - Context Free rules, have a single non-terminal on the left hand side, and a list of terminals and/or non-terminals on the right hand side.
- We have seen a very simple example of a context free grammar for English
- We have seen that we can parse using context free grammars fairly easily.

2

English Constituent Problems for Context Free Grammars

- Agreement
- Subcategorization
- Movement (for want of a better term)

3

Agreement

Determiner/Noun Agreement	Our grammar also generates
• This dog	• *This dogs
• Those dogs	• *Those dog

Subject/Verb Agreement	Our grammar also generates
• This dog eats	• *This dog eat
• Those dogs eat	• *Those dogs eats

4

Handing Number Agreement in CFGs

To handle, would need to expand the grammar with multiple sets of rules. We must have a different word class for each kind of determiner and noun.

- NP_sg → Det_sg N_sg
- NP_pl → Det_pl N_pl
- .....
- VP_sg → V_sg NP_sg
- VP_sg → V_sg NP_pl
- VP_pl → V_pl NP_sg
- VP_pl → V_pl NP_pl

5

Subcategorization

• Sneeze:	John sneezed
	*John sneezed [the book]$_{NP}$
• Find:	Please find [a flight to NY]$_{NP}$
	*Please find
• Give:	Give [me]$_{NP}$[a cheaper fare]$_{NP}$
	*Give [with a flight]$_{PP}$
• Help:	Can you help [me]$_{NP}$[with a flight]$_{PP}$
• Prefer:	I prefer [to leave earlier]$_{TO\text{-}VP}$
	*I prefer [United has a flight]$_{S}$
• Told:	I was told [United has a flight]$_{S}$
• …	

6

Subcategorization

- Subcat expresses the constraints that a predicate (verb for now) places on the number and type of the argument it wants to take

7

So?

- So the various rules for VPs overgenerate.
  - They permit the presence of strings containing verbs and arguments that don't go together
  - For example
  - VP -> V NP therefore

    Sneezed the book is a VP since "sneeze" is a verb and "the book" is a valid NP

8

Possible CFG Solution

- VP -> V
- VP -> V NP
- VP -> V NP PP
- …

- VP -> IntransV
- VP -> TransV NP
- VP -> TransPP NP PP
- …

9

Movement

- Core example
  - My travel agent booked the flight

10

Movement

- Core example
  - [[My travel agent]$_{NP}$ [booked [the flight]$_{NP}$]$_{VP}$]$_{S}$

- I.e. "book" is a straightforward transitive verb. It expects a single NP arg within the VP as one of its arguments, and a single NP arg as the subject.

11

Movement

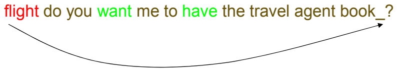
- What about?
  - Which flight do you want me to have the travel agent book_?

- The direct object argument to "book" isn't appearing in the right place. It is in fact a long way from where its supposed to appear.

12

Movement

- What about?
  - Which flight do you want me to have the travel agent book_?



- The direct object argument to “book” isn’t appearing in the right place. It is in fact a long way from where its supposed to appear.
- And note that its separated from its verb by 2 other verbs.

13

The Point

- CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.
- But there are problems
  - That can be dealt with adequately, although not elegantly, by staying within the CFG framework.
- There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power)
- We will use feature structures and the constraint-based unification formalism

14

Features

- Go back to subject verb agreement case
- An alternative is to rethink the terminal and non-terminals as complex objects with associated properties (called features) that can be manipulated.
- Features take on different values
- The application of grammar rules is constrained by testing on these features

15

Subject-Verb Agreement

- We could use features that allow us to code rules such as the following:

- S → NP VP
- *Only if the number of the NP is equal to the number of the VP (that is, the NP and VP agree in number).*

- This allows us to have the best of both worlds.

16

Features and Feature Structures

- We can encode these properties by associating what are called Feature Structures with grammatical constituents.
- Feature structures are sets of feature-value pairs where:
  - The features are atomic symbols and
  - The values are either atomic symbols or feature structures

$$\begin{bmatrix} \mathit{Feature}_1 & \mathit{Value}_1 \\ \mathit{Feature}_2 & \mathit{Value}_2 \\ \vdots & \vdots \\ \mathit{Feature}_n & \mathit{Value}_n \end{bmatrix}$$

17

Example Feature Structures

$$\begin{bmatrix} \mathit{Number} & \mathit{SG} \end{bmatrix}$$

$$\begin{bmatrix} \mathit{Number} & \mathit{SG} \\ \mathit{Person} & \mathit{3} \end{bmatrix}$$

$$\begin{bmatrix} \mathit{Cat} & \mathit{NP} \\ \mathit{Number} & \mathit{SG} \\ \mathit{Person} & \mathit{3} \end{bmatrix}$$

18

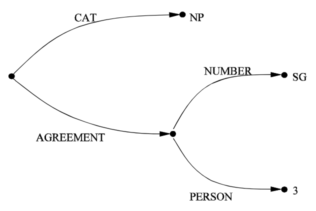
Bundles of Features

- Feature Values can be feature structures themselves.
- This is useful when certain features commonly co-occur, as number and person.

$$\begin{bmatrix} \mathit{Cat} & \mathit{NP} \\ \mathit{Agreement} & \begin{bmatrix} \mathit{Number} & \mathit{SG} \\ \mathit{Person} & 3 \end{bmatrix} \end{bmatrix}$$

19

Feature Structures as DAGs



20

Reentrant Structure

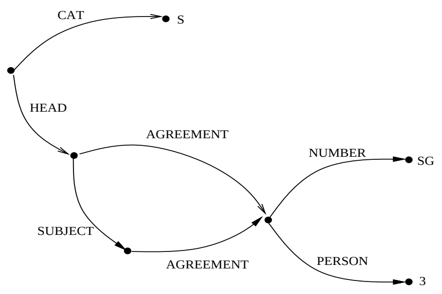
- We'll allow multiple features in a feature structure to share the same values. By this we mean that they share the same structure, not just that they have the same value.

$$\begin{bmatrix} \mathit{Cat} & S \\ \mathit{Head} & \begin{bmatrix} \mathit{Agreement} & \boxed{1} \begin{bmatrix} \mathit{Number} & \mathit{SG} \\ \mathit{Person} & 3 \end{bmatrix} \\ \mathit{Subject} & \begin{bmatrix} \mathit{Agreement} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- Numerical indices indicate the shared value.

21

Reentrant DAGs



22

Reentrant Structure

- It will also be useful to talk about paths through feature structures. As in the paths
- <HEAD AGREEMENT NUMBER>
- <HEAD SUBJECT AGREEMENT NUMBER>

$$\begin{bmatrix} \mathit{Cat} & S \\ \mathit{Head} & \begin{bmatrix} \mathit{Agreement} & \boxed{1} \begin{bmatrix} \mathit{Number} & \mathit{SG} \\ \mathit{Person} & 3 \end{bmatrix} \\ \mathit{Subject} & \begin{bmatrix} \mathit{Agreement} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

23

The Unification Operation

So what do we want to do with these things...

- check the compatibility of two structures
- merge the information in two structures

We can do both with an operation called Unification.

Merging two feature structures produces a new feature structure that is more specific (has more information) than, or is identical to, each of the input feature structures.

24

The Unification Operation

- We say two feature structures can be unified if the component features that make them up are compatible.

- [number sg] U [number sg] = [number sg]
- [number sg] U [number pl] = fails!

- Structures are compatible if they contain no features that are incompatible.
- If so, unification returns the union of all feature/value pairs.

25

The Unification Operation

- [number sg] U [number []] =

26

The Unification Operation

- [number sg] U [number []] = [number sg]

- [number sg] U [person 3] =

27

The Unification Operation

- [number sg] U [number []] = [number sg]

- [number sg] U [person 3] = $\begin{bmatrix} \textit{number} & \textit{sg} \\ \textit{person} & \textit{3} \end{bmatrix}$

28

Unification Operation

$$\begin{bmatrix} \textit{Agreement} & \textit{[Number sg]} & \\ \textit{Subject} & \textit{[Agreement} & \textit{[Number sg]]} \end{bmatrix}$$

U

[Subject [Agreement [Person 3]]]

=

$$\begin{bmatrix} \textit{Agreement} & \textit{[Number sg]} \\ \textit{Subject} & \begin{bmatrix} \textit{Agreement} & \begin{bmatrix} \textit{Number} & \textit{sg} \\ \textit{Person} & \textit{3} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

29

The Unification Operation

[Head [Subject [Agreement [Number PL]]]]

U

$$\begin{bmatrix} \textit{Cat} & \textit{S} \\ \textit{Head} & \begin{bmatrix} \textit{Agreement} & \boxed{1}\begin{bmatrix} \textit{Number SG} \\ \textit{Person 3} \end{bmatrix} \\ \textit{Subject} & \begin{bmatrix} \textit{Agreement} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

= Fail!

30

Properties of Unification

- Monotonic: if some description is true of a feature structure, it will still be true after unifying it with another feature structure.
- Order independent: given a set of feature structures to unify, we can unify them in any order and we'll get the same result.

31

Features, Unification, and Grammars

We'll incorporate all this into our grammars in two ways:

- We'll assume that constituents are objects which have feature-structures associated with them
- We'll associate sets of unification constraints with grammar rules that must be satisfied for the rule to be satisfied.

32

Unification Constraints

$\beta_0 \rightarrow \beta_1 \ldots \beta_n$

{ set of constraints }

< β_i feature path > = atomic value
< β_i feature path > = < β_k feature path >

33

Agreement

NP → Det Nominal
< Det AGREEMENT > = < Nominal AGREEMENT >
< NP AGREEMENT > = < Nominal AGREEMENT >

Noun → flight
< Noun AGREEMENT NUMBER > = SG

Noun → flights
< Noun AGREEMENT NUMBER > = PL

Nominal → Noun
< Nominal AGREEMENT > = < Noun AGREEMENT >

Det → this
< Det AGREEMENT NUMBER > = SG

34

Unification and Parsing

- OK, let's assume we've augmented our grammar with sets of path-like unification constraints.
- What changes do we need to make to a parser to make use of them?
  - Building feature structures and associating them with a subtree
  - Unifying feature structures as subtrees are created
  - Blocking ill-formed constituents

35

Unification and Earley Parsing

With respect to an Earley-style parser…

- Building feature structures (represented as DAGs) and associate them with states in the chart
- Unifying feature structures as states are advanced in the chart
- Block ill-formed states from entering the chart

36

Building Feature Structures

- Features of most grammatical categories are copied from head child to parent (e.g., from V to VP, Nom to NP, N to Nom)

VP → V NP
- < VP HEAD > = < V HEAD >

S → NP VP
- < NP HEAD AGREEMENT > = < VP HEAD AGREEMENT>
- < S HEAD > = < VP HEAD >

$$\left[\begin{array}{lll} S & [head\ \boxed{1}\ ] & \\ NP & [head & [agreement\ \boxed{2}\ ]] \\ VP & [head & \boxed{1}\ [agreement\ \boxed{2}\ ]] \end{array}\right]$$

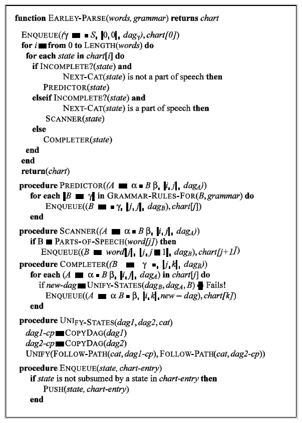
Augmenting States with DAGs

- We just add a new field to the representation of the states

S → . NP VP, [0,0], [], Dag

Unifying States and Blocking

- Keep much of the Earley Algorithm the same.
- We want to unify the DAGs of existing states as they are combined as specified by the grammatical constraints.
- Alter COMPLETER – when a new state is created, first make sure the individual DAGs unify. If so, then add the new DAG (resulting from the unification) to the new state.



```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE((γ ■ ▪ S, |0, 0|, dagγ), chart[0])
  for i ■ from 0 to LENGTH(words) do
   for each state in chart[i] do
     if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
         PREDICTOR(state)
     elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
          SCANNER(state)
     else
         COMPLETER(state)
   end
  end
  return(chart)

 procedure PREDICTOR((A ■ α ▪ B β, |i, j|, dagA))
    for each |B ■ γ| in GRAMMAR-RULES-FOR(B, grammar) do
         ENQUEUE((B ■ ▪ γ, |j, j|, dagB), chart[j])
    end

 procedure SCANNER((A ■ α ▪ B β, |i, j|, dagA))
    if B ■ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE((B ■ word|j|, |j, j ■ 1|, dagB), chart[j+1])
 procedure COMPLETER((B ■ γ ▪, |j, k|, dagB))
    for each (A ■ α ▪ B β, |i, j|, dagA) in chart[j] do
       if new-dag ■ UNIFY-STATES(dagB, dagA, B) ■ Fails!
         ENQUEUE((A ■ α B ▪ β, |i, k|, new − dag), chart[k])
    end

 procedure UNIFY-STATES(dag1, dag2, cat)
   dag1-cp ■ COPYDAG(dag1)
   dag2-cp ■ COPYDAG(dag2)
   UNIFY(FOLLOW-PATH(cat, dag1-cp), FOLLOW-PATH(cat, dag2-cp))

 procedure ENQUEUE(state, chart-entry)
    if state is not subsumed by a state in chart-entry then
        PUSH(state, chart-entry)
    end
```

Modifying Earley

Completer
- Recall: Completer adds new states to chart by finding states whose dot can be advanced (i.e., category of next constituent matches that of completed constituent)
- Now: Completer will only advance those states if their feature structures unify.

Also, new test for whether to enter a state in the chart
- Now DAGs may differ, so check must be more complex
- Don't add states that have DAGs that are more specific than states in chart; is new state subsumed by existing states?

Example

- NP → Det . Nominal [0,1], [SDet], DAG1

$$\left[\begin{array}{llll} np & [head\ \boxed{1}\ ] & & \\ det & [head & [agreement\ \boxed{2} & [number\ sg]]] \\ nominal & [head & \boxed{1}\ [agreement\ \boxed{2} &]] \end{array}\right]$$

- Nominal → Noun ., [1,2], [SNoun], Dag2

$$\left[\begin{array}{llll} nominal & [head\ \boxed{1}\ ] & & \\ noun & [head & \boxed{1}\ [agreement & [number\ sg]]] \end{array}\right]$$