6.863J Natural Language Processing
Lecture 12: Featured attraction

Instructor: Robert C. Berwick
berwick@ai.mit.edu

The Menu Bar

- Administrivia:
  - 3a due Friday; Lab 3b out Weds; due after vacation

Agenda:
  - Parsing strategies: Honey, I shrank the grammar!
  - Features
Why: recover meaning from structure

John ate ice-cream → ate(John, ice-cream)

-This must be done from \textit{structure}
-Actually want something like $\lambda x \lambda y \text{ate}(x, y)$

How?
Two parts:

- Syntax: define hierarchical structure
- Semantics: interpret over hierarchical structure

- What are the constraints?

Conclusion we will head to

- If we use too powerful a formalism, it lets us write ‘unnatural’ grammars
- This puts burden on the person writing the grammar – which may be ok.
- However, child doesn’t presumably do this (they don’t get ‘late days’)
- We want to strive for automatic programming – ambitious goal
Key elements – part 1

- Establish basic phrase types: S, VP, NP, PP, ...
- Where do these come from???

What kinds of phrases are there?

- Noun phrases, verb phrases, adjectival phrases ("green with envy"), adverbial phrases ("quickly up the hill"), prepositional phrases ("off the wall"), etc.
- In general: grounded on lexical items
- Shows us the constraints on context-free rules for natural grammars
- Example:
Phrase types are constrained by **lexical** projection

Verb Phrase $\rightarrow$ Noun Phrase
  
  "is-a"
  
  Prepositional Phrase $\rightarrow$ Preposition Noun Phrase
  
  "on the table"

Adjective Phrase $\rightarrow$ Prep. Phrase
  
  "green with envy"

Etc. ... what is the pattern?

---

Function-argument relation

XP $\rightarrow$ X arguments, where X = Noun, Verb, Preposition, Adjective (all lexical categories in the language)

Like function-argument structure

(so-called "Xbar theory")

Constrains what grammar rules cannot be:

- Verb Phrase $\rightarrow$ Noun Noun Phrase
- or even
- Verb Phrase $\rightarrow$ Noun Phrase Verb Noun Phrase
English is function-argument form

Other languages are the mirror-inverse: arg-function

This is like Japanese
Key elements – part 2

- Establish *verb subcategories*
- What are these?
  - Different verbs take different # arguments
  - 0, 1, 2 arguments (‘complements’)
  - Poirot thought; Poirot thought the gun; Poirot thought the gun was the cause.
  - Some verbs take certain sentence complements:
    - *I know who John saw/*? *I think who John saw*
    - Propositional types:
      - Embedded questions: *I wonder whether...*
      - Embedded proposition: *I think that John saw Mary*

Key elements

- Subtlety to this
- Believe, know, think, wonder,...
  - ? *I believe why John likes ice-cream*
  - *I know why John likes ice-cream*
  - *I believe that John likes ice-cream*
  - *I believe (that) John likes ice-cream*

- # args, type: *Verb subcategories*
- How many subcategories are there?
- What is the structure?
Idea for phrases

- They are based on ‘projections’ of words (lexical items) – imagine features ‘percolating’ up

Heads of phrases

V +proposition

know [V +proposition]
The parse structure for ‘embedded’ sentences

\[ \text{I believe (that) John likes ice-cream} \]

New phrase type: S-bar

\[ \text{that J. likes ice-cream} \]
I believe that Comp S. J. likes ice-cream.
In fact, true for all sentences...

Comp ⊒ S
ε
S
J. likes ice-cream
S_{bar} John likes ice-cream

Why?

What rules will we need?

- (U do it..)
Verb types - continued

- What about:
  Clinton admires honesty/Honesty admires Clinton

How do we encode these in a CFG?
Should we encode them?
- Colorless green ideas sleep furiously
- Revolutionary new ideas appear infrequently

Problems with this – how much info?
Agreement gets complex...

Czech: AGFS3 ----1A----

Other sentence types

• Questions:
  • Will John eat ice-cream?
  • Did John eat ice-cream?
  • How do we encode this?
`Empty’ elements or categories

Where surface phrase is displaced from its canonical syntactic position

Examples:

- The ice-cream was eaten vs.
- John ate the ice-cream
- What did John eat?
- What did Bill say that that John thought the cat ate?
- For What x, did Bill say... the cat ate x
- Bush is too stubborn to talk to
- Bush is too stubborn [x to talk to Bush]
- Bush is too stubborn to talk to the Pope
- Bush is too stubborn [Bush to talk to the Pope]

More interesting clause types

Apparently “long distance” effects:

- “displacement” of phrases from their “base” positions

1. So-called ‘wh-movement’:
   \[ \text{What did John eat?} \]
2. Topicalization (actually the same)
   \[ \text{On this day, it snowed two feet.} \]
3. Other cases: so-called ‘passive’:
   \[ \text{The eggplant was eaten by John} \]

- How to handle this?
We can think of this as ‘fillers’ and ‘gaps’

- Filler = the displaced item
- Gap = the place where it belongs, as argument
- Fillers can be NPs, PPs, S’s
- Gaps are *invisible*- so hard to parse! (we have to guess)
- Can be complex:
  
  *Which book did you file__ without__ reading__ ?
  *Which violins are these sonatas difficult to play__ on

Problems with this – how much info?

- Even verb subcategories not obvious
  
  John gave Mary the book $\rightarrow$ NP NP
  John gave the book to Mary $\rightarrow$ NP PP

  But:
  
  John donated the book to the library

  ‘Alternation’ pattern – semantic? NO!
Agreement gets complex...

- Czech: "AGFS3" --- "1A"

More interesting clause types

- Apparently "long distance" effects: 
  - "displacement" of phrases from their "base" positions

  1. So-called 'wh-movement':
     
     *What did John eat?*

  2. Topicalization (actually the same)
     
     *On this day, it snowed two feet.*

  3. Other cases: so-called 'passive' *
     
     *The eggplant was eaten by John*

- How to handle this?
`Empty’ elements or categories

*Where surface phrase is displaced from its canonical syntactic position & *nothing* shows on the surface*

- Examples:
  - The ice-cream was eaten vs.
  - John ate the ice-cream
  - What did John eat?
  - What did Bill say that John thought the cat ate?
  - For What x, did Bill say... the cat ate x
  - Bush is too stubborn to talk to
  - Bush is too stubborn [x to talk to Bush]
  - Bush is too stubborn to talk to the Pope
  - Bush is too stubborn [Bush to talk to the Pope]

‘missing’ or empty categories

- John promised Mary ___ to leave
- John promised Mary [John to leave]
- Known as ‘control’

- John persuaded Mary [___ to leave]
- John persuaded Mary [Mary to leave]
We can think of this as ‘fillers’ and ‘gaps’

- Filler = the displaced item
- Gap = the place where it belongs, as argument
- Fillers can be NPs, PPs, S’s
- Gaps are invisible - so hard to parse! (we have to guess)
- Can be complex:
  - Which book did you file without reading?
  - Which violins are these sonatas difficult to play on?

Gaps

- Pretend “kiss” is a pure transitive verb.
- Is “the president kissed” grammatical?
  - If so, what type of phrase is it?

- the sandwich that
- I wonder what
- What else has

- the president kissed e
- Sally said the president kissed e
- Sally consumed the pickle with e
- Sally consumed e with the pickle
Gaps

- **Object gaps:**
  - the sandwich that
  - I wonder what
  - What else has

- **Subject gaps:**
  - the sandwich that
  - I wonder what
  - What else has

[how could you tell the difference?]

Phrases with missing NP:

\[ X[\text{missing=NP}] \]
or just \( X/\text{NP} \) for short
Representation & computation questions again

- How do we represent this displacement? (difference between underlying & surface forms)
- How do we compute it? (I.e., parse sentences that exhibit it)
- We want to recover the underlying structural relationship because this tells us what the predicate-argument relations are – Who did what to whom
- Example: What did John eat → For which x, x a thing, did John eat x?
- Note how the eat-x predicate-argument is established

Representations with gaps

- Let's first look at a tree with gaps:

[Diagram of a tree with gaps, labeled 'gap' or empty element]
Fillers can be arbitrarily far from gaps they match with...

- What did John say that Mary thought that the cat ate___?
Fillers and gaps

- Since ‘gap’ is NP going to empty string, we could just add rule, NP→ε
- But this will overgenerate why?
- We need a way to distinguish between
  - What did John eat
  - Did John eat
- How did this work in the FSA case?

So, what do we need?

- A rule to expand NP as the empty symbol; that’s easy enough: NP→ε
- A way to make sure that NP is expanded as empty symbol iff there is a gap (in the right place) before/after it
- A way to link the filler and the gap
- We can do all this by futzing with the nonterminal names: Generalized Phrase Structure Grammar (GPSG)
Example: relative clauses

- What are they?
- Noun phrase with a sentence embedded in it:
  - The sandwich that the president ate
- What about it? What’s the syntactic representation that will make the semantics transparent?

The sandwich that the president ate

OK, that’s the output...what are the cfg rules?

- Need to expand the object of eat as an empty string
- So, need rule NP→ε
- But more, we need to link the head noun “the sandwich” to this position
- Let’s use the fsa trick to ‘remember’ something – what is that trick???
- Remember?
Memory trick

- Use state of fsa to remember
- What is state in a CFG?
- The nonterminal names
- We need something like vowel harmony – sequence of states = nonterminals
  
  the sandwich that the president ate e

As a parse structure

NP
   
Det  N
the sandwich

that the president ate e

What’s this? We’ve seen it before...

It’s an Sbar = Comp+S
Parse structure for relative clause

But how to generate this and block this:

Not OK!
In short..

- We can expand out to \( e \) iff there is a prior NP we want to link to
- So, we need some way of ‘marking’ this in the state – I.e., the nonterminal
- Further, we have to somehow co-index \( e \) and ‘the sandwich’
- Well: let’s use a mark, say, “+”

The mark...

```
NP
| NP
| Det the sandwich
| N
| Comp
| Sbar+
| S+
| NP
| that
| V
| ate
| NP+
| the P.
| VN P

+ NP

+ +
```
But we can add + except this way:

- Add as part of atomic nonterminal name
  - Before:  
    - NP → NP Sbar
    - Sbar → Comp S
    - S → NP VP
    - VP → VP NP
  - After:  
    - NP → NP Sbar+
    - Sbar+ → Comp S+
    - S+ → NP VP+
    - VP+ → V NP+
    - NP+ → e

Why does this work?

- Has desired effect of blocking ‘the sandwich that the P. ate the pretzel’
- Has desired effect of allowing e exactly when there is no other object
- Has desired effect of ‘linking’ sandwich to the object (how?)
- Also: desired configuration between filler and gap (what is this?)
Actual ‘marks’ in the literature

- Called a ‘slash category’
- Ordinary category: Sbar, VP, NP
- Slash category: Sbar/NP, VP/NP, NP/NP
- “X/Y” is ONE atomic nonterminal
- Interpret as: Subtree X is missing a Y (expanded as e) underneath
- Example: Sbar/NP = Sbar missing NP underneath (see our example)

As for slash rules...

- We need slash category introduction rule, e.g., Sbar → Comp S/NP
  - We need ‘elimination’ rule NP/NP → e
  - These are paired (why?)
  - We’ll need other slash categories, e.g.,
Need PP/NP...

Also have ‘subject’ gaps
How would we write this?

Filler-gap configuration

[Diagram showing a triangle labeled S, NP, and e, with an arrow pointing to a box labeled S, NP, e]
Filler-gap configuration

- Equivalent to notion of ‘scope’ for natural languages (scope of variables) \( \approx \) Environment frame in Scheme/binding environment for ‘variables’ that are empty categories
- Formally: Fillers c-command gaps (constituent command)
- Definition of c-command:

C-command

- A phrase \( \alpha \) c-commands a phrase \( \beta \) iff the first branching node that dominates \( \alpha \) also dominates \( \beta \) (blue = filler, green = gap)
Natural for $\lambda$ abstraction

Puzzle:

- Who saw Mary?
Idea 1: WYSIG syntax

Root

Q(uestion)

NP+wh       VP+tns

Pronp+wh    V+tns    NP

Who        saw    Name

Mary

Is this right?
Another example

S

Sbar

Conj

Sbar

and

Mary caught the rabid dog

John killed the rabid dog

What if we move the object?

S/NP

Sbar

Conj

Sbar

and

Mary caught e

John killed e

the rabid dog
Why not read off the rules?

- Why can’t we just build a machine to do this?
- We could induce rules from the structures
- But we have to know the right representations (structures) to begin with
- Penn treebank has structures – so could use learning program for that
- This is, as noted, a construction based approach
- We have to account for various constraints, as noted

So what?

- What about multiple fillers and gaps?

- Which violins are these sonatas difficult to play _____ on which violins?
How many context-free rules?

- For every displaced phrase, what do we do to the ‘regular’ context-free rules?
- How many kinds of displaced rules are there? Which book and Which pencil did Mary buy?
  *Mary asked who and what bought
- Well, how many???
- Add in agreement...

And then..

- John saw more horses than bill saw cows or Mary talked to
- John saw more horses than bill saw cows or mary talked to cats
- The kennel which Mary made and Fido sleeps in has been stolen
- The kennel which Mary made and Fido sleeps has been stolen
CFG Solution

- Encode constraints into the non-terminals
  - Noun/verb agreement
    \[ S \rightarrow SgS \]
    \[ S \rightarrow PlS \]
    \[ SgS \rightarrow SgNP SgVP \]
    \[ SgNP \rightarrow SgDet SgNom \]
  - Verb subcategories:
    \[ IntransVP \rightarrow IntransV \]
    \[ TransVP \rightarrow TransV NP \]
  - Complex nonterminal names

---

How big can the grammar get???
But this means huge proliferation of rules...

An alternative:

- View terminals and non-terminals as complex objects with associated features, which take on different values
- Write grammar rules whose application is constrained by tests on these features, e.g.
  \[ S \to \text{NP VP} \text{ (only if the NP and VP agree in number)} \]

Design advantage

- Decouple skeleton syntactic structure from lexicon
- In fact, the syntactic structure really is a skeleton:
From this...

To this
Features are everywhere

morphology of a single word:
\[ \text{Verb}[\text{head}=\text{thrill}, \text{tense}=\text{present}, \text{num}=\text{sing}, \text{person}=3, \ldots] \rightarrow \text{thrills} \]

projection of features up to a bigger phrase
\[ \text{VP}[\text{head}=\alpha, \text{tense}=\beta, \text{num}=\gamma, \ldots] \rightarrow \text{V}[\text{head}=\alpha, \text{tense}=\beta, \text{num}=\gamma, \ldots] \text{ NP} \]
provided \( \alpha \) is in the set \( \text{TRANSITIVE-VERBS} \)

agreement between sister phrases:
\[ \text{S}[\text{head}=\alpha, \text{tense}=\beta] \rightarrow \text{NP}[\text{num}=\gamma, \ldots] \text{ VP}[\text{head}=\alpha, \text{tense}=\beta, \text{num}=\gamma, \ldots] \]
provided \( \alpha \) is in the set \( \text{TRANSITIVE-VERBS} \)

Better approach to factoring linguistic knowledge

- Use the *superposition* idea: we superimpose one set of constraints on top of another:
  1. Basic skeleton tree
  2. Plus the added feature constraints
- \[ S \rightarrow \text{NP} \text{ VP} \]
  \[ \text{[num x]} \quad \text{[num x]} \quad \text{[num x]} \]
  \[ \text{the guy} \quad \text{eats} \]
  \[ \text{[num singular]} \quad \text{[num singular]} \]
Or in tree form:

```
S [number x]
  NP [number x]  VP [number x]
    DT [number x]  N [number x]  V [number x]  NP
      the  guy  eats
      [number singular]  [number singular]  [number singular]
```

Values trickle up

```
S [number x]
  NP [number x]  VP [number x]
      the  guy  eats
      [number singular]  [number singular]  [number singular]
```
Checking features

What sort of power do we need here?

• We have \([\text{feature value}]\) combinations so far
  • This seems fairly widespread in language
    • We call these \(\textit{atomic feature-value combinations}\)
    • Other examples:
      1. In English:
         person feature (1\(^{st}\), 2\(^{nd}\), 3\(^{rd}\));
         Case feature (degenerate in English: nominative,
         object/accusative, possessive/genitive): I know \(\text{her}\) vs.
         I know \(\text{she}\);
         Number feature: plural/sing; definite/indefinite
         Degree: comparative/superlative
Other languages; formalizing features

- Two kinds:
  1. Syntactic features, purely grammatical function
     Example: Case in German (NOMinative, ACCusative, DATive case) – relative pronoun must agree w/ Case of verb with which it is construed
     Wer nicht stark ist, muss klug sein
     Who not strong is, must clever be
     NOM   NOM
     Who isn’t strong must be clever

Continuing this example

Ich nehme, wen du mir empfehlst
I take whomever you me recommend
ACC  ACC  ACC
I take whomever you recommend to me

*Ich nehme, wen du vertraust
I take whomever you trust
ACC  ACC  DAT
Other class of features

2. Syntactic features w/ meaning – example, number, def/indef., adjective degree

Hungarian

Akart egy könyvet
He-wanted a book
-DEF -DEF

ey könyv amit akart
A book which he-wanted
-DEF -DEF

Feature Structures

Sets of feature-value pairs where:

- Features are atomic symbols
- Values are atomic symbols or feature structures
- Illustrated by attribute-value matrix
How to formalize?

- Let $F$ be a finite set of feature names, let $A$ be a set of feature values.
- Let $p$ be a function from feature names to permissible feature values, that is, $p: F \rightarrow 2^A$.
- Now we can define a *word category* as a triple $<F, A, p>$.
- This is a partial function from feature names to feature values.

Example

$F = \{\text{CAT, PLU, PER}\}$

- $p$:
  - $p(\text{CAT}) = \{V, N, ADJ\}$
  - $p(\text{PER}) = \{1, 2, 3\}$
  - $p(\text{PLU}) = \{+, -\}$

$sleep = \{[\text{CAT V}], [\text{PLU }-], [\text{PER }1]\}$
$sleep = \{[\text{CAT V}], [\text{PLU }+], [\text{PER }1]\}$
$sleeps = \{[\text{CAT V}], [\text{PLU }-], [\text{PER }3]\}$

Checking whether features are compatible is relatively simple here... how bad can it get?
Operations on Feature Structures

- What will we need to do to these structures?
  - Check the compatibility of two structures
  - Merge the information in two structures
- We can do both using unification
- We say that two feature structures can be unified if the component features that make them up are compatible
  - [Num SG] U [Num SG] = [Num SG]
  - [Num SG] U [Num PL] fails!
  - [Num SG] U [Num []] = [Num SG]

- [Num SG] U [Pers 3] =

- Structures are compatible if they contain no features that are incompatible
- Unification of two feature structures:
  - Are the structures compatible?
  - If so, return the union of all feature/value pairs
  - A failed unification attempt
Features, Unification and Grammars

How do we incorporate feature structures into our grammars?

• Assume that constituents are objects which have feature-structures associated with them
• Associate sets of unification constraints with grammar rules
• Constraints must be satisfied for rule to be satisfied
• For a grammar rule $\beta_0 \rightarrow \beta_1 ... \beta_n$
  • $<\beta_i \text{ feature path}> = \text{Atomic value}$
  • $<\beta_i \text{ feature path}> = <\beta_j \text{ feature path}>$
• NB: if simple feat-val pairs, no arbitrary nesting, then no need for paths

Feature unification examples

(1) [ agreement: [ number: singular
  person: first ] ]

(2) [ agreement: [ number: singular]
  case: nominative ]

• (1) and (2) can unify, producing (3):

(3) [ agreement: [ number: singular
  person: first ]
  case: nominative ]

(try overlapping the graph structures corresponding to these two)
Feature unification examples

1) [ agreement: [ number: singular  
  person: first ] ]
(2) [ agreement: [ number: singular  
  case: nominative ] ]
(4) [ agreement: [ number: singular  
  person: third ] ]

- (2) & (4) can unify, yielding (5):
(5) [ agreement: [ number: singular  
  person: third ]  
  case: nominative ] ]

- BUT (1) and (4) cannot unify because their values conflict on <agreement person>

To enforce subject/verb number agreement

\[
S \rightarrow NP \ VP \\
<NP \ NUM> = <VP \ NUM>
\]
Head Features

- Features of most grammatical categories are copied from head child to parent (e.g. from V to VP, Nom to NP, N to Nom, ...)
- These normally written as ‘head’ features, e.g.
  - VP $\rightarrow$ V NP
  - $<\text{VP HEAD}> = <\text{V HEAD}>$
  - NP $\rightarrow$ Det Nom
  - $<\text{NP HEAD}> = <\text{Nom HEAD}>$
  - $<\text{Det HEAD AGR}> = <\text{Nom HEAD AGR}>$
  - Nom $\rightarrow$ N
  - $<\text{Nom HEAD}> = <\text{N HEAD}>$

```
6863J/9611J SP04 Lecture 12
```
The plan to swallow Wanda has been thrilling Otto.
The plan to swallow Wanda has been thrilling Otto.
Why use heads?

- Morphology (e.g., word endings)
  - N[\(h=\text{plan}, n=1\)] \(\rightarrow\) plan
  - N[\(h=\text{plan}, n=2+\)] \(\rightarrow\) plans
  - V[\(h=\text{thrill}, \text{tense}=\text{prog}\)] \(\rightarrow\) thrilling
  - V[\(h=\text{thrill}, \text{tense}=\text{past}\)] \(\rightarrow\) thrilled
  - V[\(h=\text{go}, \text{tense}=\text{past}\)] \(\rightarrow\) went

- Subcategorization (i.e., transitive vs. intransitive)
  - When is VP \(\rightarrow\) V NP ok?
    - VP[\(h=\alpha\)] \(\rightarrow\) V[\(h=\alpha\)] NP
      - restrict to \(\alpha \in \text{TRANSITIVE_VERBS}\)
  - When is N \(\rightarrow\) N VP ok?
    - N[\(h=\alpha\)] \(\rightarrow\) N[\(h=\alpha\)] VP
      - restrict to \(\alpha \in \{\text{plan, plot, hope, ...}\}\)
Why use heads?

- Selectional restrictions
  - \( VP[h=\alpha] \rightarrow V[h=\alpha] NP \)
  - I.e., \( VP[h=\alpha] \rightarrow V[h=\alpha] NP[h=\beta] \)
  - Don’t fill template in all ways:
    - \( VP[h=\text{thrill}] \rightarrow V[h=\text{thrill}] NP[h=\text{Otto}] \)
    - Leave out, or low prob

Equivalently: keep the template but make prob depend on \( \alpha, \beta \)

- Feature values can be feature structures themselves
  - Useful when certain features commonly co-occur, e.g. number and person

- Feature path: path through structures to value (e.g. \( Agr \rightarrow Num \rightarrow SG \))
Features and grammars

Feature checking by unification

*John sleep
Our feature structures

- Maria NAME[agr [person 3, plural -]]

Kimmo entry for Verb (eg, 'coge' after analysis):
- +e Suffix "[fin +, agr [tense pres, mode ind, person 3, plural -]]"

How can we parse with feature structures?

- Unification operator: takes 2 features structures and returns either a merged feature structure or fail
- Input structures represented as DAGs
  - Features are labels on edges
  - Values are atomic symbols or DAGs
- Unification algorithm goes through features in one input DAG\(_1\) trying to find corresponding features in DAG\(_2\) – if all match, success, else fail
- WE WILL USE MUCH SIMPLER kind of feature structure
Features and Earley Parsing

Goal:

- Use feature structures to provide richer representation
- Block entry into chart of ill-formed constituents

Changes needed to Earley:

- Add feature structures to grammar rules, & lexical entries
- Add field to states containing set representing feature structure corresponding to state of parse, e.g.

\[ S \rightarrow \bullet \text{ NP VP, [0,0], [], Set= [Agr [plural -]]} \]

Add new test to Completer operation

Recall: Completer adds new states to chart by finding states whose • 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

New test for whether to enter a state in the chart

- Now feature structures may differ, so check must be more complex
- Suppose feature structure is more specific than existing one tied to this state? Do we add it?
Evidence that you don’t need this much power

- Linguistic evidence: looks like you just check whether features are *nondistinct*, rather than equal or not – variable *matching*, not variable substitution
- Full unification lets you generate unnatural languages:
  \[ a', \ s.t. \ i \ a \ power \ of \ 2 \] – e.g., a, aa, aaaa, aaaaaaaa, ...
  why is this 'unnatural' – another (seeming) property of natural languages:
Natural languages seem to obey a *constant growth* property

Parsing with features – hook from kimmo to earley

- Features written in this form (in Kimmo)
  
  +as Suffix "[fin +, agr [tense pres, mode ind, person 2, plural -]]"

  In general:
  
  [feature value, feature [feature val, ..., feature val]]
Where wolf
6•863J/9•611J SP04 Lecture 12

Parse Tree

<table>
<thead>
<tr>
<th>Maria'</th>
<th>'Yoga'</th>
<th>los'</th>
<th>los'</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
<tr>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
<td>□ &amp;L &amp;L &amp;L &amp;L</td>
</tr>
</tbody>
</table>

NPs in the word 'Maria' have the following structure:

- Maria' is a noun phrase.
- 'Yoga' is a noun phrase.
- los' is a noun phrase.
- los' is a noun phrase.

In the sentence 'Maria' 'Yoga' los' los', the structure can be represented as follows:

1. Maria' (subject)
2. 'Yoga' (object)
3. los' (direct object)
4. los' (indirect object)

This structure is consistent with the parse tree diagram presented in the lecture.
Constant growth property

Claim: \( \exists \) Bound \( k \) on the ‘distance gap’ between any two consecutive sentences in this list, which can be specified in advance (fixed)

- ‘Intervals’ between valid sentences cannot get too big – cannot grow w/o bounds
- We can do this a bit more formally

Constant growth

- **Dfn.** A language \( L \) is *semilinear* if the number of occurrences of each symbol in any string of \( L \) is a linear combination of the occurrences of these symbols in some fixed, finite set of strings of \( L \).

- **Dfn.** A language \( L \) is *constant growth* if there is a constant \( c_0 \) and a finite set of constants \( C \) s.t. for all \( w \in L \), where \( |w| > c_0 \) \( \exists w' \in L \) s.t. \( |w| = |w'| + c \) some \( c \in C \)

- **Fact.** (Parikh, 1971). Context-free languages are semilinear, and constant-growth

- **Fact.** (Berwick, 1983). The power of 2 language is non constant-growth
General feature grammars – how violate these properties

- Take example from so-called “lexical-functional grammar” but this applies as well to any general unification grammar
- Lexical functional grammar (LFG): add checking rules to CF rules (also variant HPSG)

Example LFG

- Basic CF rule:
  \[ S \rightarrow NP \ VP \]
- Add corresponding ‘feature checking’
  \[ S \rightarrow \quad NP \quad VP \quad \]
  \[ (\uparrow \text{subj num}) = \downarrow \quad \uparrow = \downarrow \]
- What is the interpretation of this?
Applying feature checking in LFG

NP VP
(↑ subj num)=↓
V↑=↓

S

NP
(↑ subj num)=↓

VP

V↑=↓

Whatever features from below

[subj [num singular]]
Copy up above

N
guys

[num plural]

[num singular]
sleeps

Alas, this allows non-constant growth, unnatural languages

- Can use LFG to generate power of 2 language
- Very simple to do
- \(A \rightarrow A\) \(A\)
  \((↑ f) = ↓\) \((↑ f) = ↓\)

\(A \rightarrow a\)
\((↑ f) = 1\)

Lets us `count' the number of embeddings on the right & the left – make sure a power of 2
Example

If mismatch anywhere, get a feature clash...

Checks ok

Fails!
Conclusion then

- If we use too powerful a formalism, it lets us write ‘unnatural’ grammars
- This puts burden on the person writing the grammar – which may be ok.
- However, child doesn’t presumably do this (they don’t get ‘late days’)
- We want to strive for automatic programming – ambitious goal