6.863J Natural Language Processing
Lecture 10: Feature-based grammars

Robert C. Berwick

The Menu Bar

- Administrivia:
  - Schedule alert: Lab 3 out; due next Weds. (after that: Lab4 on semantics, 2 ways)
  - Lab time today, tomorrow
  - Please read notes3.pdf!! englishgrammar.pdf (on web)
- *Agenda:*
- Limits of context-free grammars: the trouble with tribbles
- Foundation for the laboratory: empty categories
- Feature-based grammars/parsing

CFG Solution to complexity of language

- Encode constraints into the non-terminals
  - Noun/verb agreement

    S→ SgS

    S → PlS

    SgS → SgNP SgVP

    SgNP → SgDet SgNom
  - Verb subcategories:

    IntransVP → IntransV

    TransVP → TransV NP

Problems with this – how much info?

- Even verb subcategories not obvious

  John gave Mary the book → NP NP

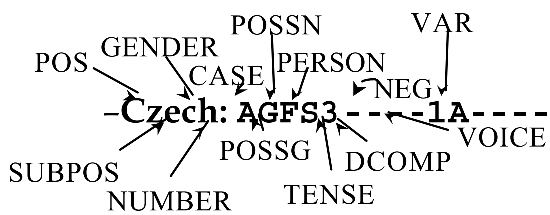
  John gave the book to Mary → NP PP

But:

John donated the book to the library

'Alternation' pattern – semantic? NO!

Agreement gets complex…



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 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

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

[how could you tell the difference?]

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

e kissed the president
Sally said e kissed the president

Gaps

- All gaps are really the same – a missing XP:
- 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
e kissed the president
Sally said e kissed the president

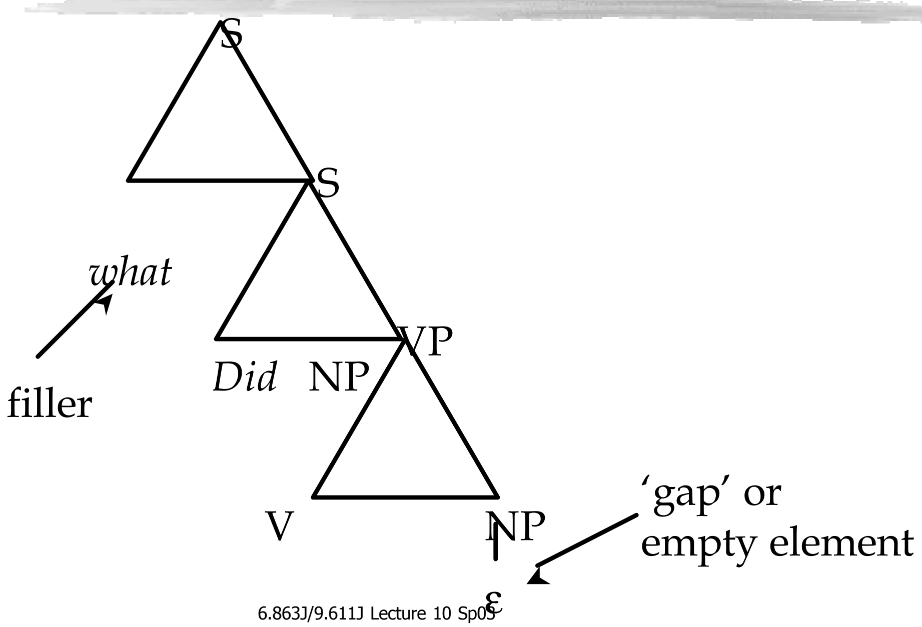
Phrases with missing NP:
X[missing=NP]
or just X/NP for short

Representation & computation questions again

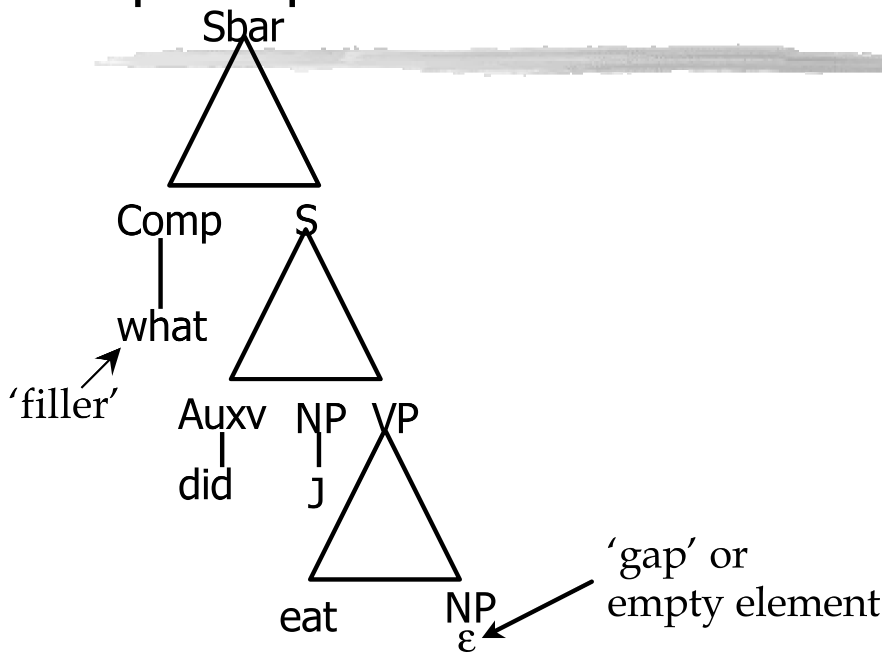
- How do we *represent* this displacement? (difference between underlying & surface forms)
- How do we *compute* it? (I.e., parse sentences that exhibit it)
- We want to recover the *underlying* structural relationship because this tells us what the predicate-argument relations are – *Who did what to 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:



Crisper representation:



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$\rightarrow\varepsilon$
- 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$\rightarrow\varepsilon$
- 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$_i$ that the president ate e$_i$

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$\rightarrow\varepsilon$
- 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

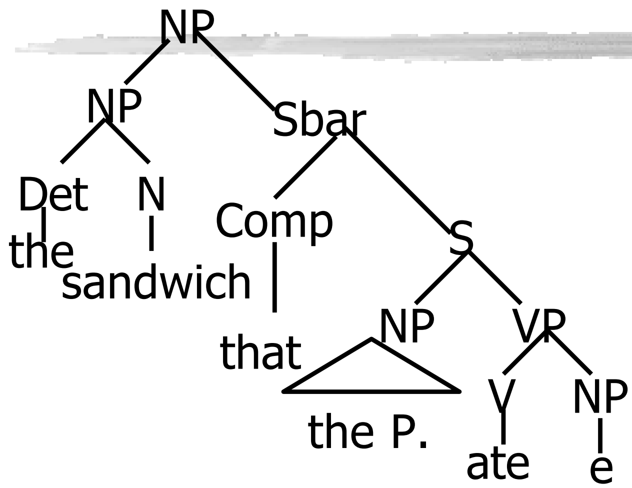
the sandwich

that the president ate e

What's this? We've seen it before…

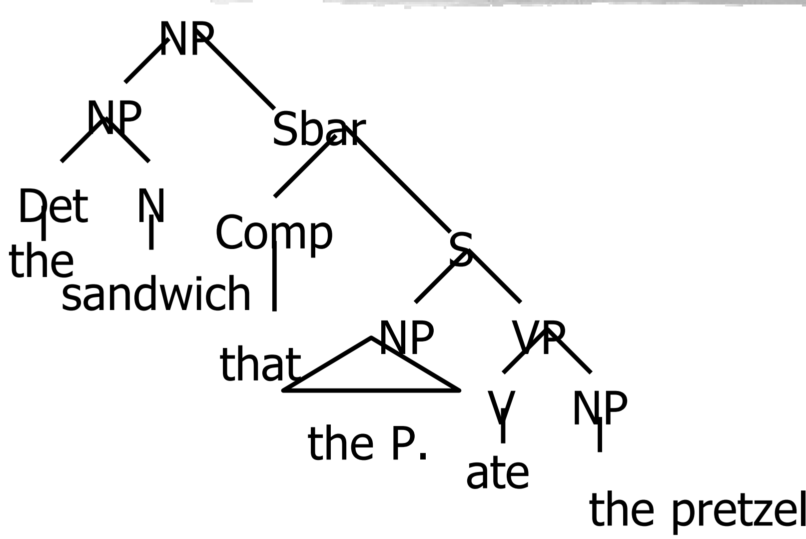
It's an Sbar = Comp+S

Parse structure for relative clause



But how to generate this and block this:

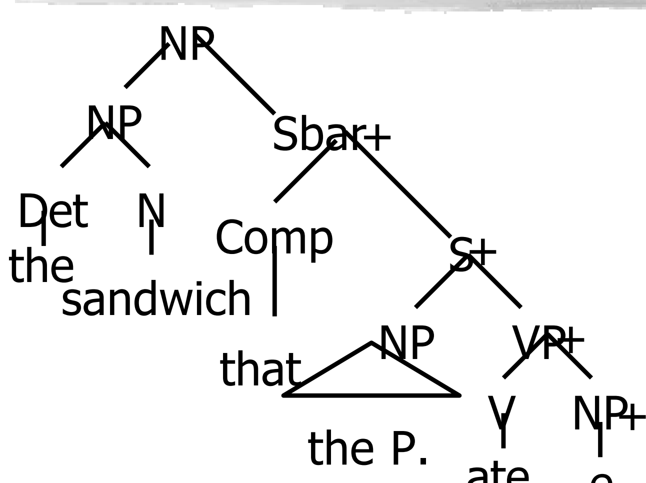
Not OK!



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…



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 <u>blocking</u> ‘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 <u>configuation</u> between filler and gap (what is this?)

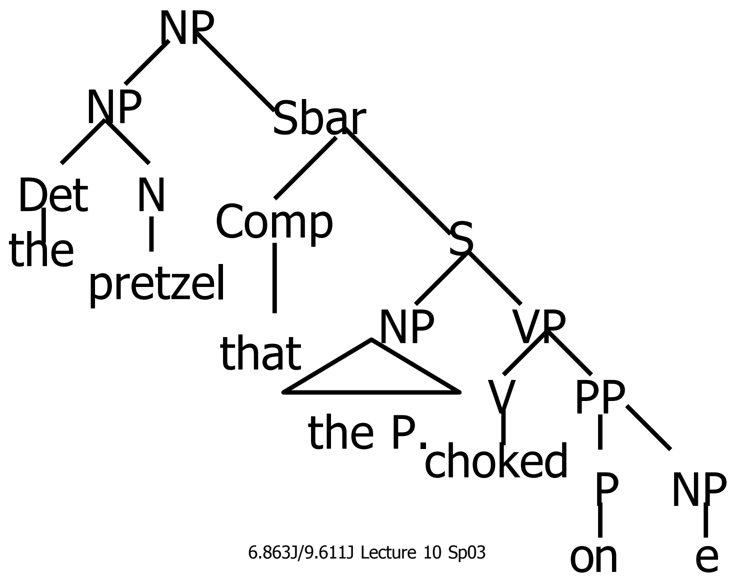
Actual 'marks' in the literature

- 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)

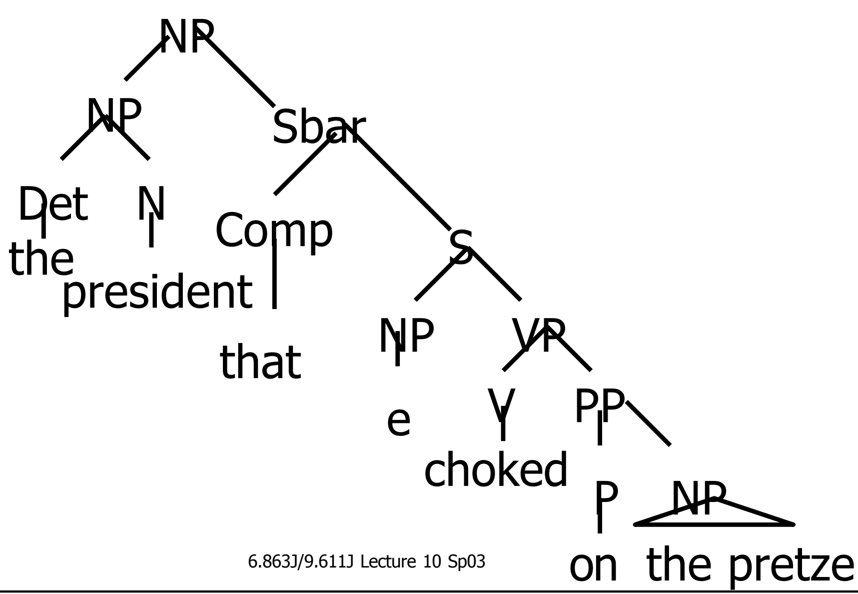
As for slash rules…

- We need slash category introduction rule, e.g., Sbar $\rightarrow$ Comp S/NP
- We need 'elimination' rule NP/NP$\rightarrow$e
- These are paired (why?)
- We'll need other slash categories, e.g.,

Need PP/NP…

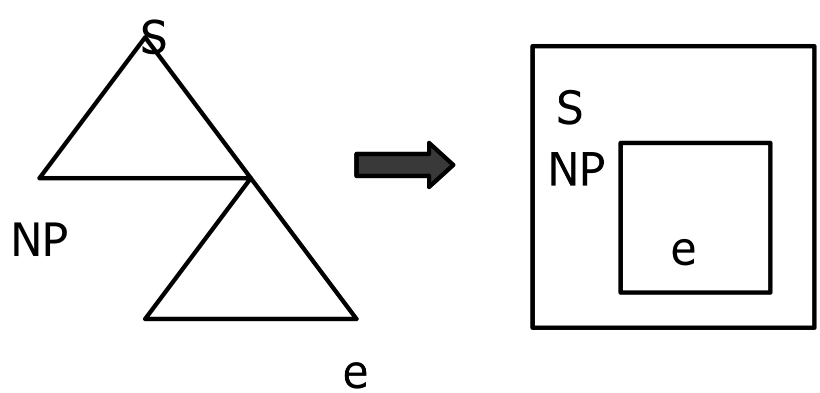


Also have 'subject' gaps



How would we write this?

Filler-gap configuration

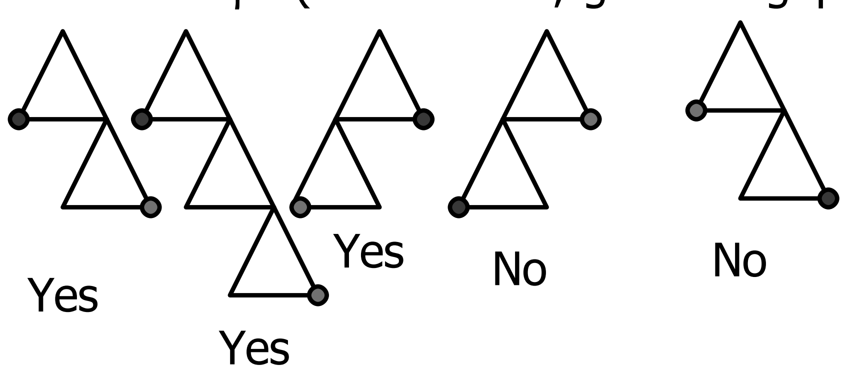


Filler-gap configuration

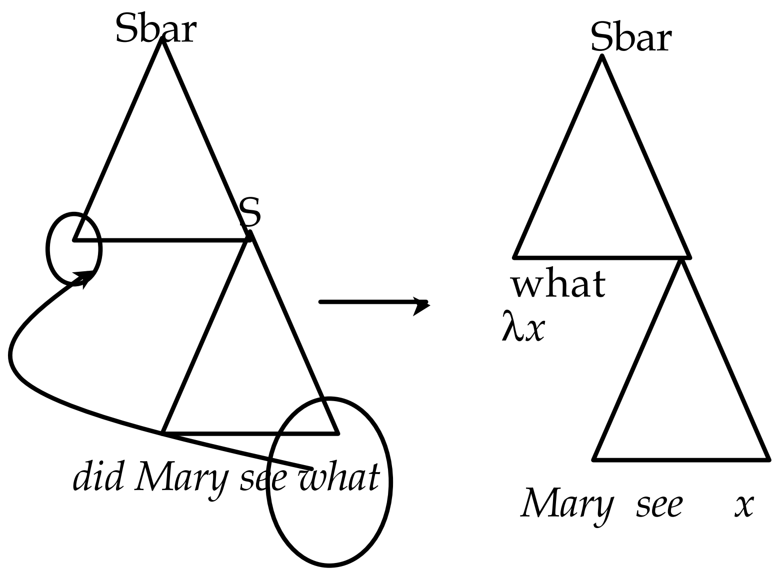
- Equivalent to notion of ‘scope’ for natural languages (scope of variables) ≈ Environment frame in Scheme/binding environment for ‘variables’ that are empty categories
- Formally: Fillers c-command gaps (constituent command)
- Definition of c-command:

C-command

- A phrase α c-commands a phrase β iff the first branching node that dominates α also dominates β (blue = filler, green = gap)



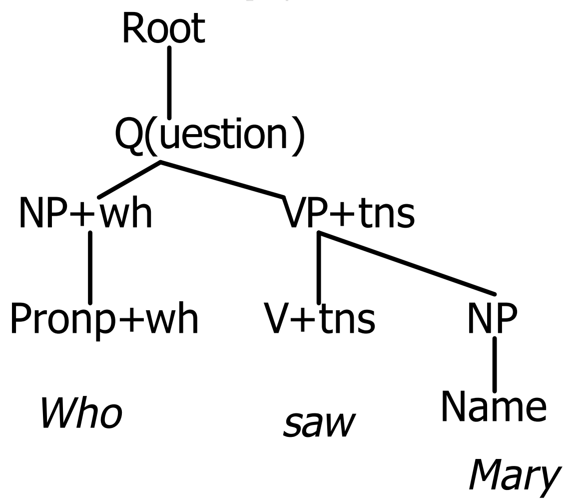
Natural for λ abstraction



Puzzle:

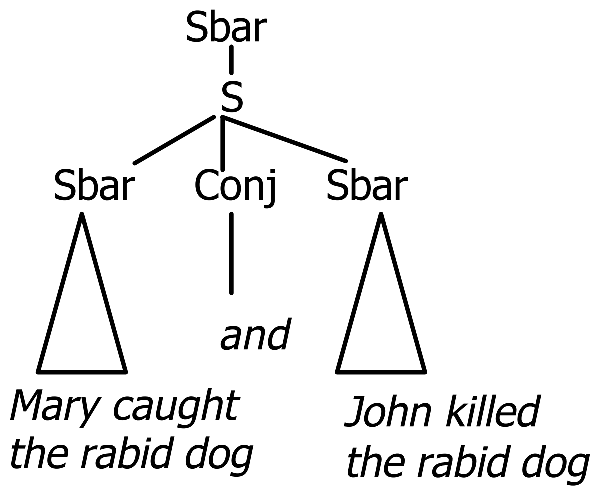
- Who saw Mary?

Idea 1: WYSIG syntax

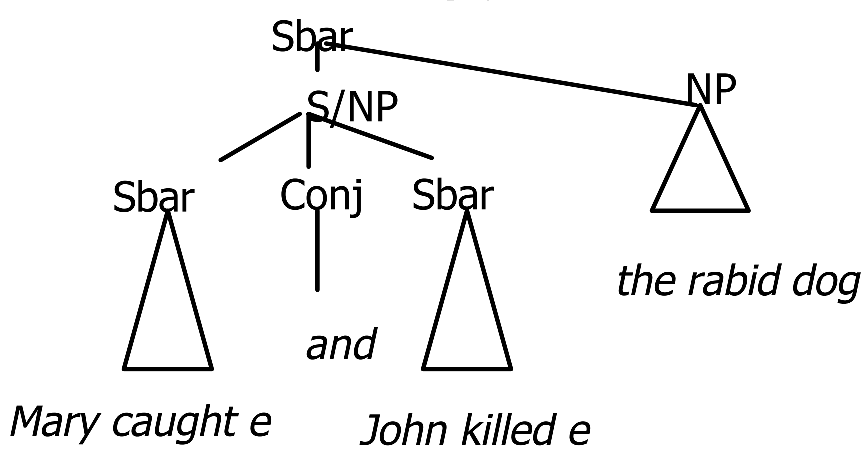


Is this right?

Another example



What if we move the object?



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 these sonatas 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

Limits of CFGs

- Agreement (A cat sleeps. Cats sleep.)

S → NP VP

NP → Det Nom

But these rules overgenerate, allowing, e.g., *A cat sleep…

- Subcategorization (Cats dream. Cats eat cantaloupe.)

VP → V

VP → V NP

But these also allow

*Cats dream cantaloupe.

- We need to constrain the grammar rules to enforce e.g. number agreement and subcategorization differences
- We'll do this with feature structures and the constraint-based unification formalism

CFG Solution

- Encode constraints into the non-terminals
  - Noun/verb agreement

    S→ SgS

    S → PlS

    SgS → SgNP SgVP

    SgNP → SgDet SgNom
  - Verb subcat:

    IntransVP → IntransV

    TransVP → TransV NP

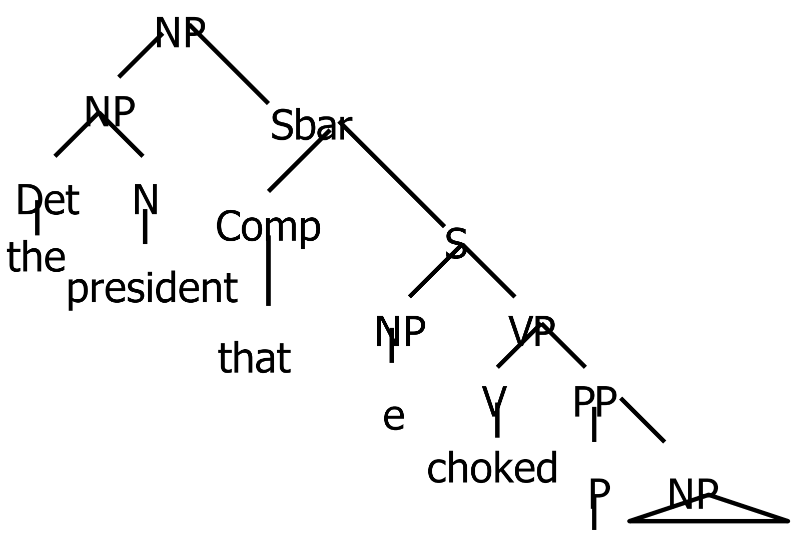
- But this means huge proliferation of rules…
- An alternative:
  - View terminals and non-terminals as complex objects with associated features, which take on different values
  - Write grammar rules whose application is constrained by tests on these features, e.g.

    S → NP VP (only if the NP and VP agree in number)

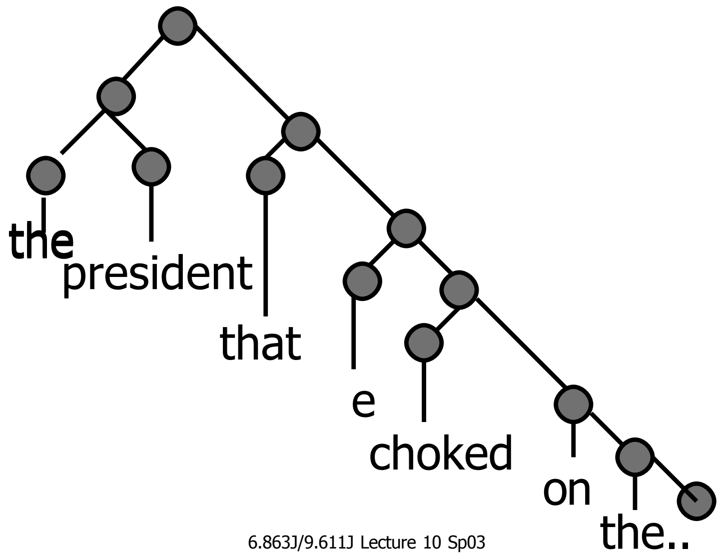
Design advantage

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

From this…



To this



Feature Structures

- Sets of feature-value pairs where:
  - Features are atomic symbols
  - Values are atomic symbols or feature structures
  - Illustrated by attribute-value matrix

$$\begin{bmatrix} Feature_1 & Value_1 \\ Feature_2 & Value_2 \\ \ldots & \ldots. \\ Feature_n & Value_n \end{bmatrix}$$

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*= {CAT, PLU, PER}
- *p:*

  p(CAT)={V, N, ADJ}

  p(PER)={1, 2, 3}

  p(PLU)={+, -}

sleep = {[CAT V], [PLU -], [PER 1]}

sleep = {[CAT V], [PLU +], [PER 1]}

sleeps= {[CAT V], [PLU -], [PER 3]}

Checking whether features are compatible is relatively simple here

Important question

- Do features have to be <u>*more*</u> complicated than this?
- More: hierarchically structured (<u>*feature structures*</u>) (directed acyclic graphs, DAGs, or even beyond)
- Then *checking* for feature compatibility amounts to *unification*
- Example

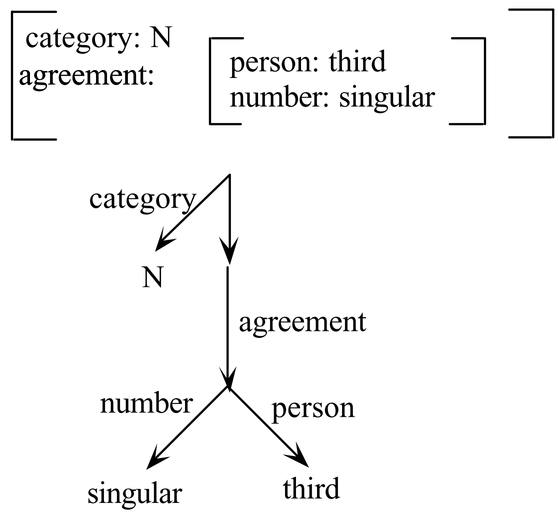
- How do we define 3plNP?
- How does this improve over the <u>CFG solution</u>?

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

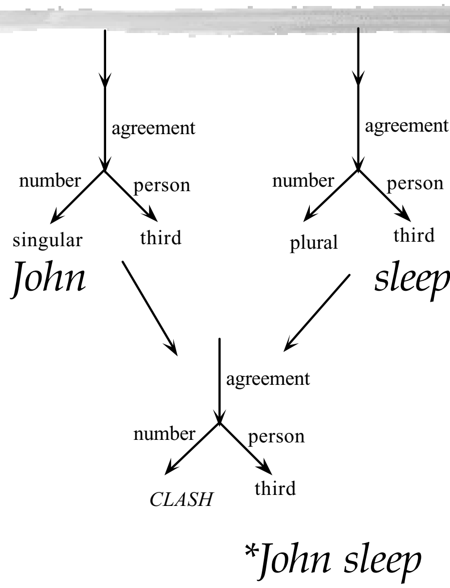
$$\begin{bmatrix} Cat & NP \\ Agr & \begin{bmatrix} Num & SG \\ Pers & 3 \end{bmatrix} \end{bmatrix}$$

  - Feature path: path through structures to value (e.g.
    Agr → Num → SG

Features and grammars



Feature checking by unification



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^i, *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

Constant growth property

- Take a language & order its sentences int terms of increasing length in terms of # of words (what's shortest sentence in English?)
- Claim: ∃Bound 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)

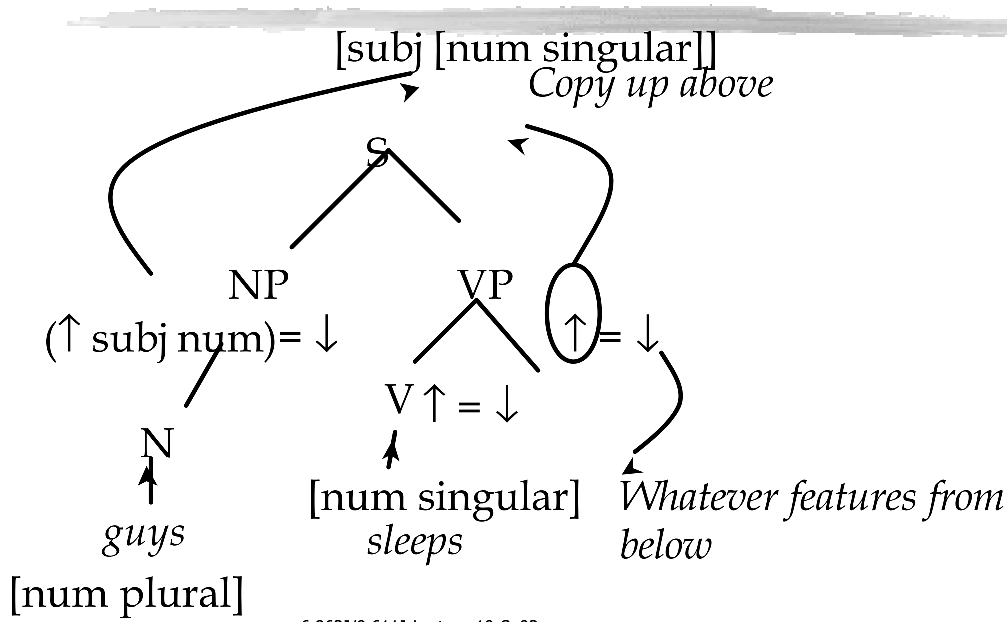
Example LFG

- Basic CF rule:
  $S \rightarrow NP\ VP$
- Add corresponding 'feature checking'

$$S \rightarrow \underset{(\uparrow \text{ subj num}) = \downarrow}{NP} \qquad \underset{\uparrow = \downarrow}{VP}$$

- What is the interpretation of this?

Applying feature checking in LFG

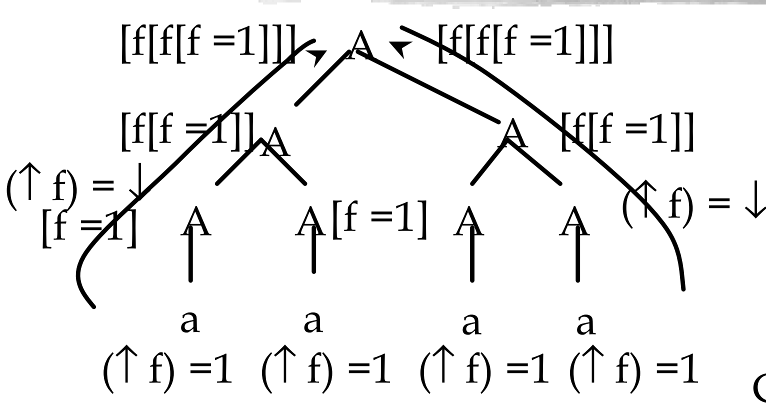


Alas, this allows non-constant growth, unnatural languages

- Can use LFG to generate power of 2 language
- Very simple to do
- A$\rightarrow$ A A
  $(\uparrow f) = \downarrow$ $(\uparrow f) = \downarrow$
  A $\rightarrow$ a
  $(\uparrow f) = 1$

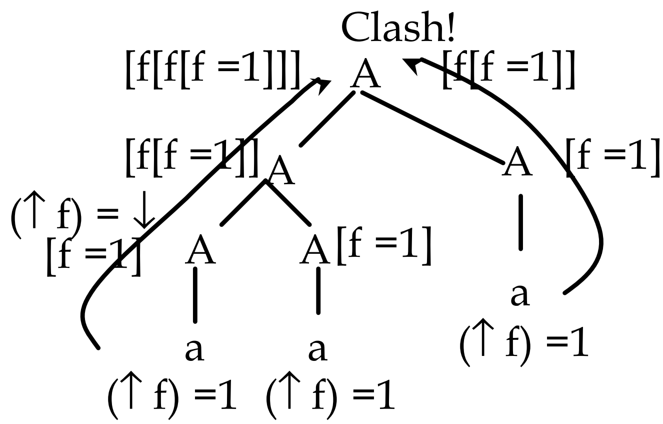
Lets us `count' the number of embeddings on the right & the left – make sure a power of 2

Example



Checks ok

If mismatch anywhere, get a feature clash…



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