6.863J Natural Language Processing
Lecture 14: Features to lambdas

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

The Menu Bar

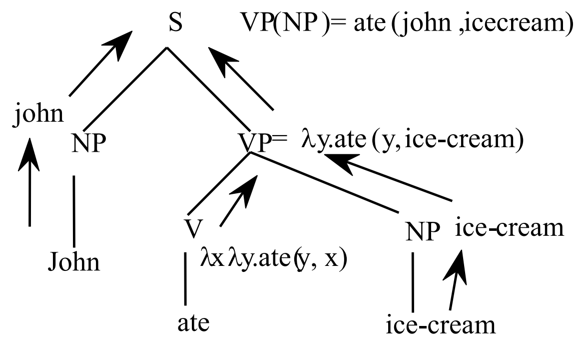
- Administrivia:
  - Lab 3b out yesterday due April 12

Agenda:

Features & feature grammars

What does all this mean?

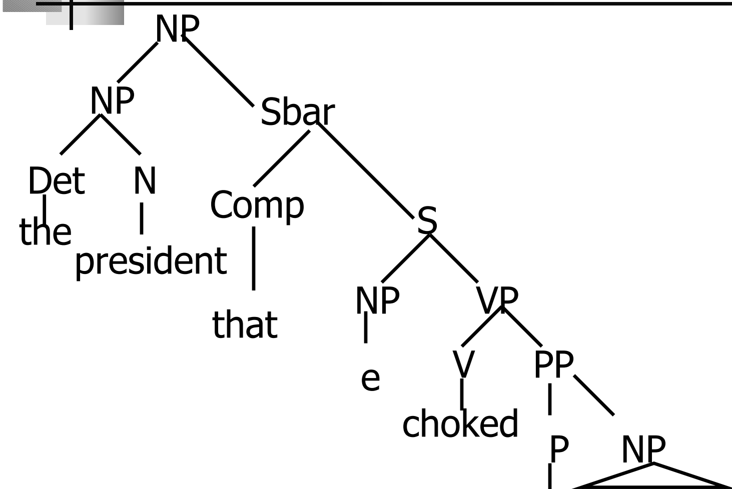
Why: recover meaning from structure



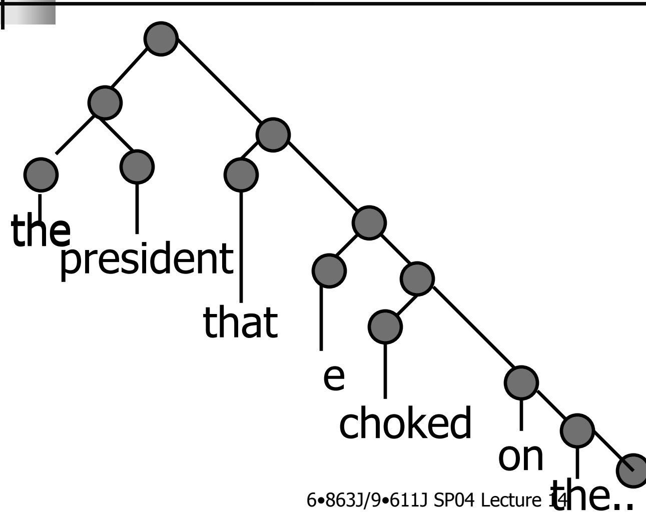
Design advantage

- 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:

Verb[head=thrill, tense=present, num=sing, person=3,…] → thrills

projection of features up to a bigger phrase

VP[head=α, tense=β, num=γ…] → V[head=α, tense=β, num=γ…] NP

provided α is in the set TRANSITIVE-VERBS

agreement between sister phrases:

S[head=α, tense=β] → NP[num=γ,…] VP[head=α, tense=β, num=γ…]

provided α is in the set TRANSITIVE-VERBS

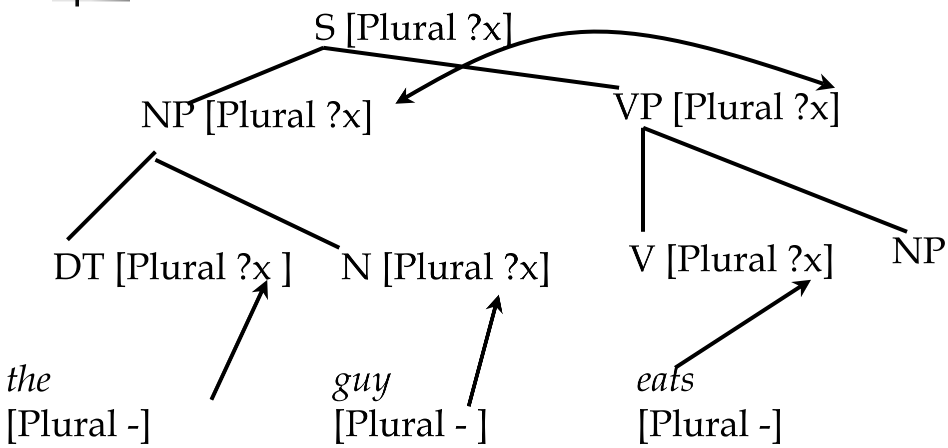
Better approach to factoring linguistic knowledge

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	→ NP	VP
[Plural ?x]	[Plural ?x]	[Plural ?x]
	the guy	*eats*
	[Plural -]	[Plural -]

Checking features



Feature Structures

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

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…how bad can it get?

What sort of power do we need here?

- We have [*feature value*] combinations so far
- This seems fairly widespread in language
- We call these *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 *her* vs. I know *she;*

Number feature: plural/sing; definite/indefinite

Degree: comparative/superlative

Operations on Feature Structures

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

Feature unification examples

(1) [agreement [Plural - ,
person first]]

(2) [agreement [Plural -] ,
case nominative]

- (1) and (2) *can* unify, producing (3):

(3) [agreement [Plural - ,
person first] ,
case nominative]

(try overlapping the graph structures corresponding to these two)

Feature unification examples

1) [agreement [Plural - ,
person first]]

(2) [agreement [Plural -] ,
case nominative]

(4) [agreement [Plural - ,
person third]]

- (2) & (4) *can* unify, yielding (5):

(5) [agreement [Plural - ,
person third],
case nominative]

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

Our feature structures

- **NP[agr ?B] -> DET[agr ?B] N[agr ?B]**
- **VP[fin ?A, agr ?B] -> V2[fin ?A, agr ?B] NP**

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

Kimmo entry for Verb (eg, 'coge' after analysis):

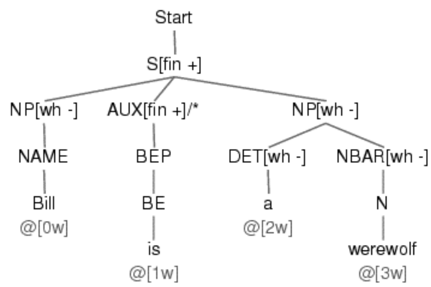
- **+e Suffix "[fin +, agr [tense pres, mode ind, person 3, plural -]]"**

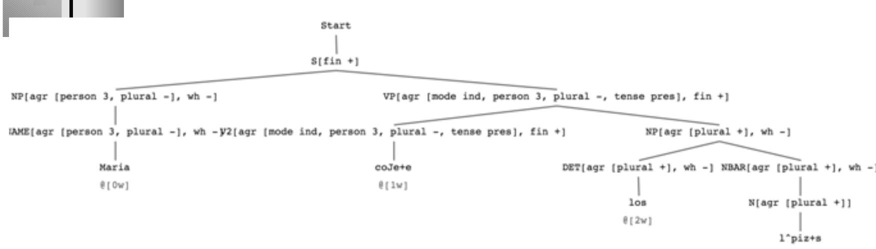
Parsing with features – hook from kimmo to earley

- 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


Start
S[fin +]
NP[wh -]
AUX[fin +]/*
NP[wh -]
NAME
BEP
DET[wh -]
NBAR[wh -]
Bill
@[0w]
BE
a
@[2w]
N
is
@[1w]
werewolf
@[3w]

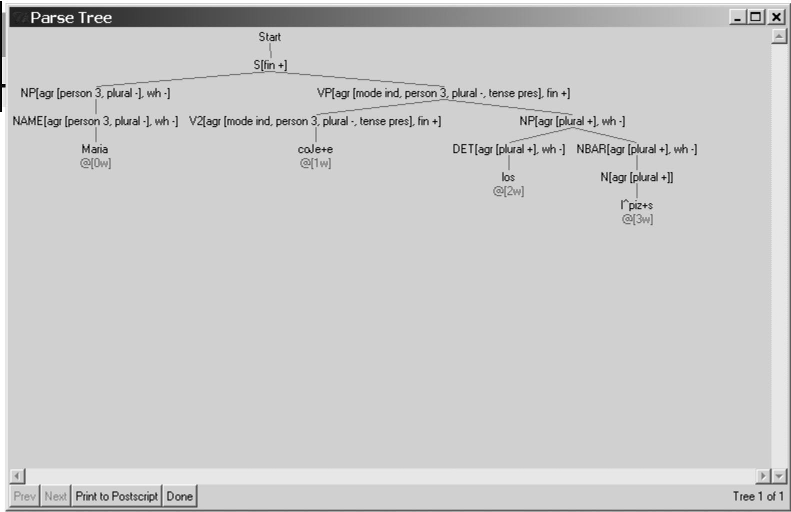

Start
S[fin +]
NP[agr [person 3, plural -], wh -]
VP[agr [mode ind, person 3, plural -, tense pres], fin +]
NAME[agr [person 3, plural -], wh -]
V2[agr [mode ind, person 3, plural -, tense pres], fin +]
NP[agr [plural +], wh -]
Maria
@[0w]
coJe+e
@[1w]
DET[agr [plural +], wh -]
NBAR[agr [plural +], wh -]
los
@[2w]
N[agr [plural +]]
l^piz+s
@[3w]

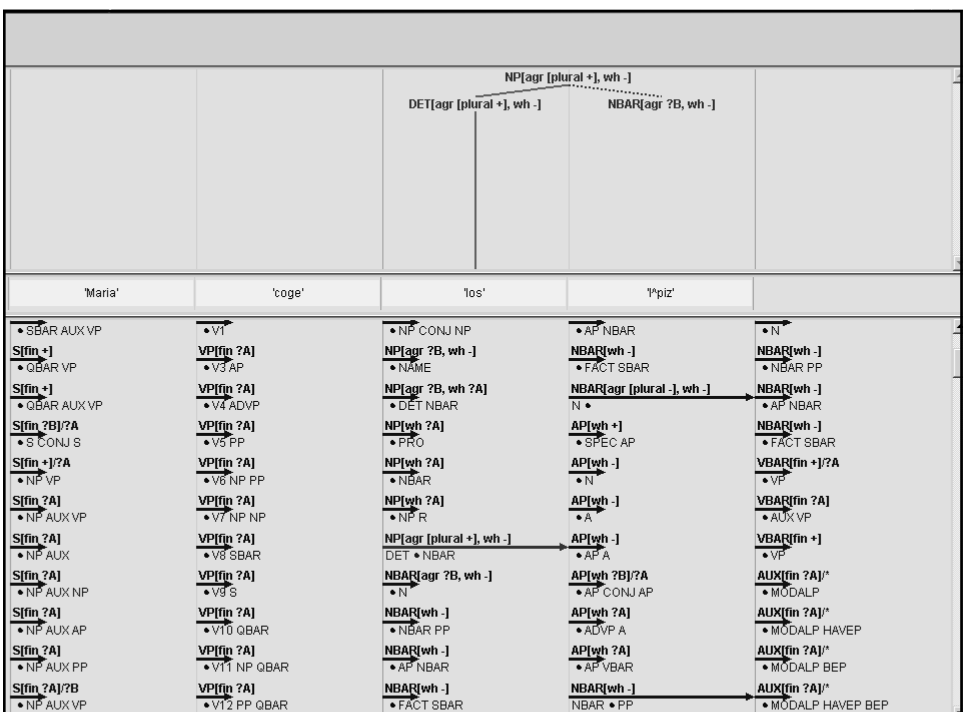
Features and Earley Parsing

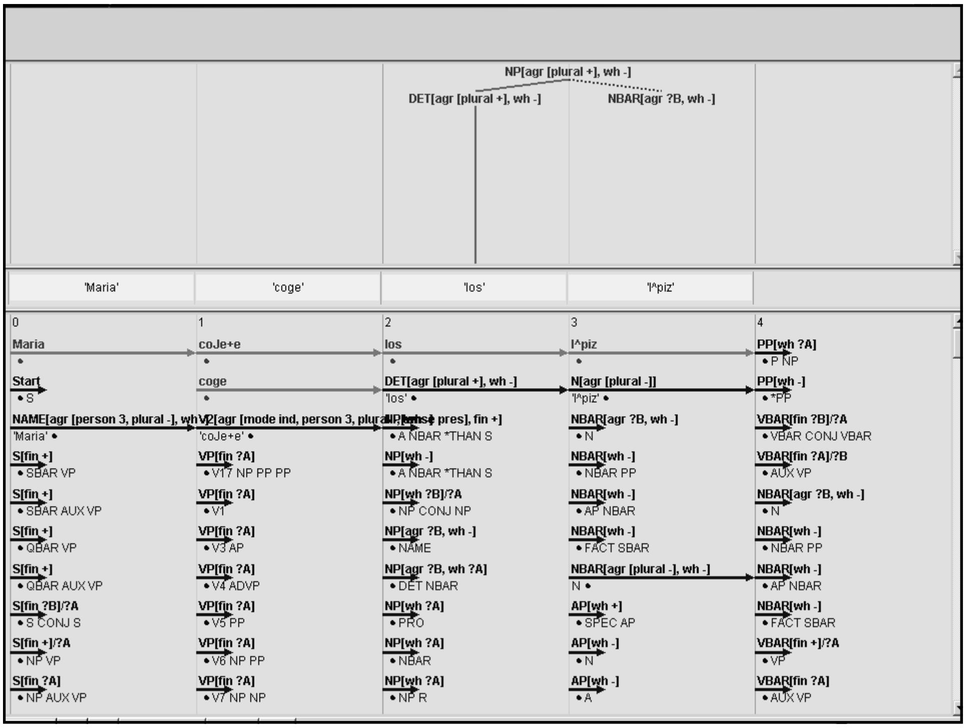
- Goal:
  - Use feature structures to provide richer representation
  - Block entry into chart of ill-formed constituents
- Changes needed to Earley
  - Add feature structures to grammar rules, & lexical entries
  - Add field to states containing set representing feature structure corresponding to state of parse, e.g.

    S → • NP VP, [0,0], [], Set= [Agr [plural -]]

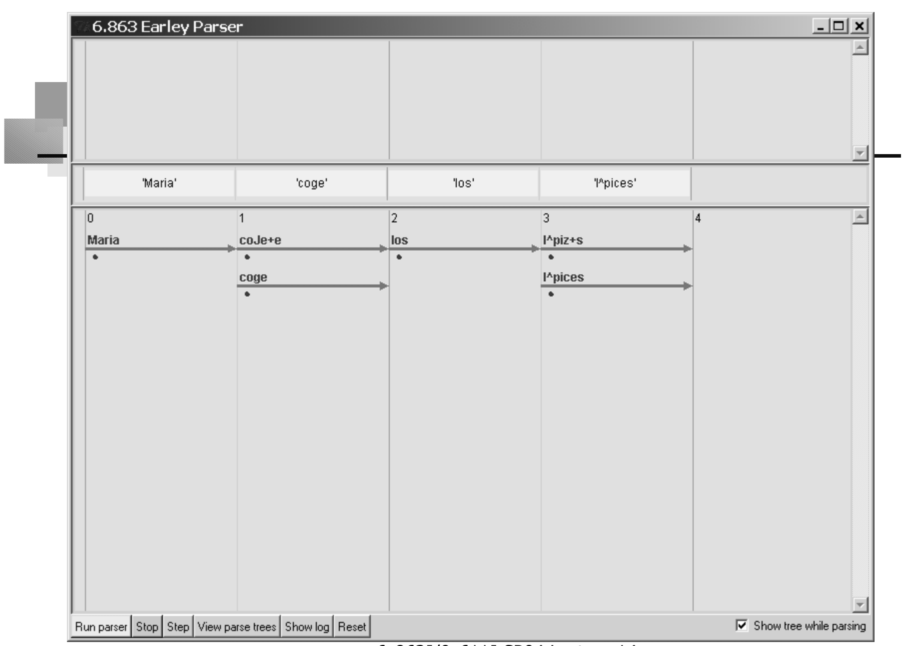
- **Add <u>new</u> test to Completer operation
- Recall: Completer adds new states to chart by finding states whose • can be advanced (i.e., category of next constituent matches that of completed constituent)
  - Now: Completer will only advance those states if their feature structures are consistent
- ** Add New test for whether to enter edge in the chart
  - Now feature structures may differ, so check must be more complex
  - Suppose feature structure is <u>more specific</u> than existing one tied to this state? Do we add it?


Parse Tree
Start
S[fin +]
NP[agr [person 3, plural -], wh -]
VP[agr [mode ind, person 3, plural -, tense pres], fin +]
NAME[agr [person 3, plural -], wh -]
V2[agr [mode ind, person 3, plural -, tense pres], fin +]
NP[agr [plural +], wh -]
Maria
@[0w]
coJe+e
@[1w]
DET[agr [plural +], wh -]
NBAR[agr [plural +], wh -]
los
@[2w]
N[agr [plural +]]
l^piz+s
@[3w]
Prev
Next
Print to Postscript
Done
Tree 1 of 1

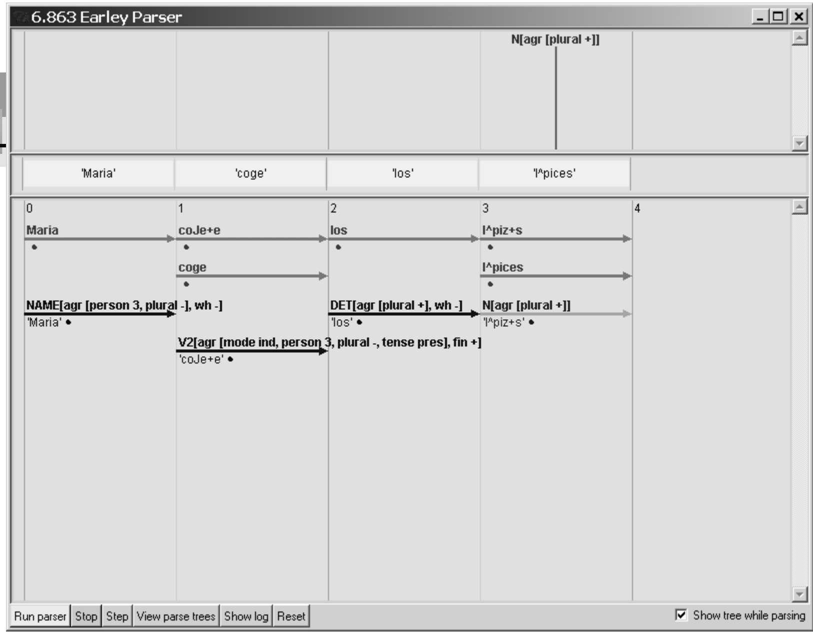

NP[agr [plural +], wh -]
DET[agr [plural +], wh -]
NBAR[agr ?B, wh -]
'Maria'
'coge'
'los'
'l^piz'
• SBAR AUX VP
S[fin +]
• QBAR VP
S[fin +]
• QBAR AUX VP
S[fin ?B]/?A
• S CONJ S
S[fin +]/?A
• NP VP
S[fin ?A]
• NP AUX VP
S[fin ?A]
• NP AUX
S[fin ?A]
• NP AUX NP
S[fin ?A]
• NP AUX AP
S[fin ?A]
• NP AUX PP
S[fin ?A]/?B
• NP AUX VP
• V1
VP[fin ?A]
• V3 AP
VP[fin ?A]
• V4 ADVP
VP[fin ?A]
• V5 PP
VP[fin ?A]
• V6 NP PP
VP[fin ?A]
• V7 NP NP
VP[fin ?A]
• V8 SBAR
VP[fin ?A]
• V9 S
VP[fin ?A]
• V10 QBAR
VP[fin ?A]
• V11 NP QBAR
VP[fin ?A]
• V12 PP QBAR
• NP CONJ NP
NP[agr ?B, wh -]
• NAME
NP[agr ?B, wh ?A]
• DET NBAR
NP[wh ?A]
• PRO
NP[wh ?A]
• NBAR
NP[wh ?A]
• NP R
NP[agr [plural +], wh -]
DET • NBAR
NBAR[agr ?B, wh -]
• N
NBAR[wh -]
• NBAR PP
NBAR[wh -]
• AP NBAR
NBAR[wh -]
• FACT SBAR
• AP NBAR
NBAR[wh -]
• FACT SBAR
NBAR[agr [plural -], wh -]
N •
AP[wh +]
• SPEC AP
AP[wh -]
• N
AP[wh -]
• A
AP[wh -]
• AP A
AP[wh ?B]/?A
• AP CONJ AP
AP[wh ?A]
• ADVP A
AP[wh ?A]
• AP VBAR
NBAR[wh -]
NBAR • PP
• N
NBAR[wh -]
• NBAR PP
NBAR[wh -]
• AP NBAR
NBAR[wh -]
• FACT SBAR
VBAR[fin +]/?A
• VP
VBAR[fin ?A]
• AUX VP
VBAR[fin +]
• VP
AUX[fin ?A]/*
• MODALP
AUX[fin ?A]/*
• MODALP HAVEP
AUX[fin ?A]/*
• MODALP BEP
AUX[fin ?A]/*
• MODALP HAVEP BEP



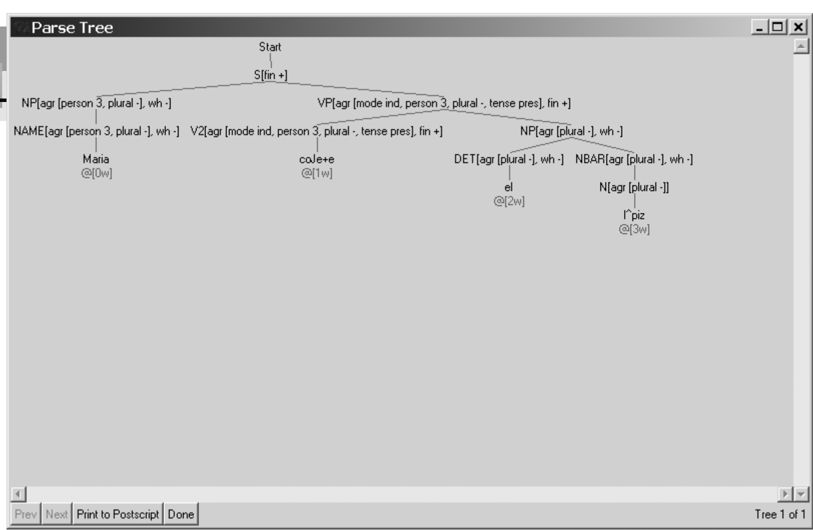
NP[agr [plural +], wh -]
DET[agr [plural +], wh -]
NBAR[agr ?B, wh -]
'Maria'
'coge'
'los'
'l^piz'
0
1
2
3
4
Maria
coJe+e
los
l^piz
PP[wh ?A]
• P NP
Start
• S
coge
DET[agr [plural +], wh -]
'los' •
N[agr [plural -]]
'l^piz' •
PP[wh -]
• *PP
'Maria' •
'coJe+e' •
• A NBAR *THAN S
NBAR[agr ?B, wh -]
• N
VBAR[fin ?B]/?A
• VBAR CONJ VBAR
S[fin +]
• SBAR VP
VP[fin ?A]
• V17 NP PP PP
NP[wh -]
• A NBAR *THAN S
NBAR[wh -]
• NBAR PP
VBAR[fin ?A]/?B
• AUX VP
S[fin +]
• SBAR AUX VP
VP[fin ?A]
• V1
NP[wh ?B]/?A
• NP CONJ NP
NBAR[wh -]
• AP NBAR
NBAR[agr ?B, wh -]
• N
S[fin +]
• QBAR VP
VP[fin ?A]
• V3 AP
NP[agr ?B, wh -]
• NAME
NBAR[wh -]
• FACT SBAR
NBAR[wh -]
• NBAR PP
S[fin +]
• QBAR AUX VP
VP[fin ?A]
• V4 ADVP
NP[agr ?B, wh ?A]
• DET NBAR
NBAR[agr [plural -], wh -]
N •
NBAR[wh -]
• AP NBAR
S[fin ?B]/?A
• S CONJ S
VP[fin ?A]
• V5 PP
NP[wh ?A]
• PRO
AP[wh +]
• SPEC AP
NBAR[wh -]
• FACT SBAR
S[fin +]/?A
• NP VP
VP[fin ?A]
• V6 NP PP
NP[wh ?A]
• NBAR
AP[wh -]
• N
VBAR[fin +]/?A
• VP
S[fin ?A]
• NP AUX VP
VP[fin ?A]
• V7 NP NP
NP[wh ?A]
• NP R
AP[wh -]
• A
VBAR[fin ?A]
• AUX VP



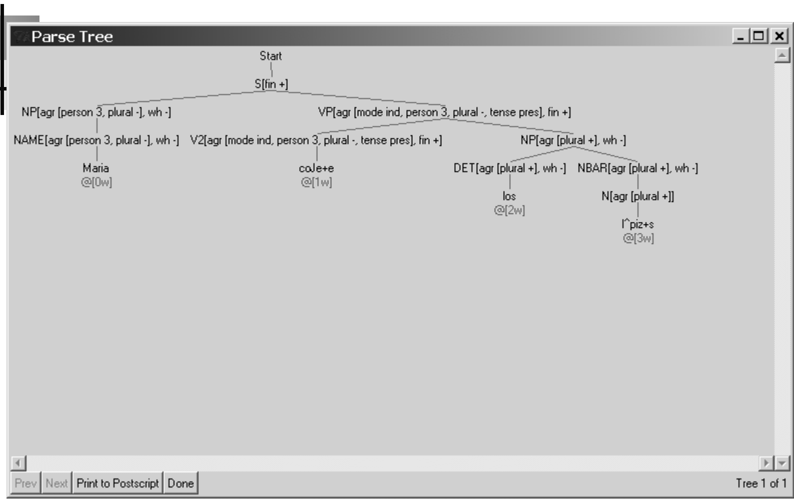
6.863 Earley Parser
'Maria'
'coge'
'los'
'l^pices'
0
1
2
3
4
Maria
coJe+e
los
l^piz+s
coge
l^pices
Run parser
Stop
Step
View parse trees
Show log
Reset
Show tree while parsing



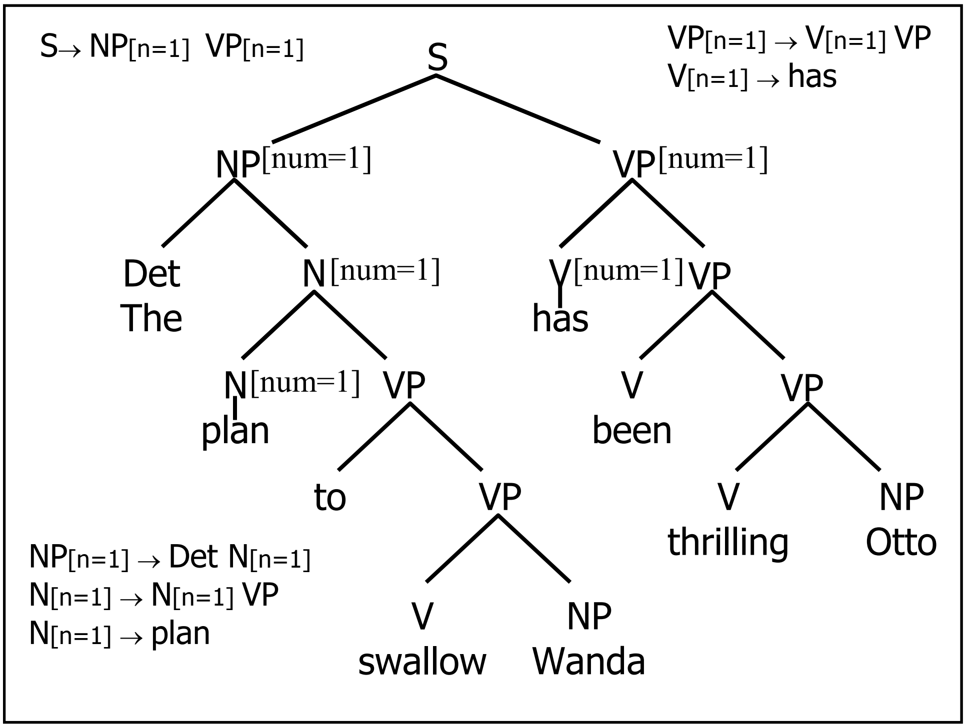
6•863J/9•611J SP04 Lecture 14

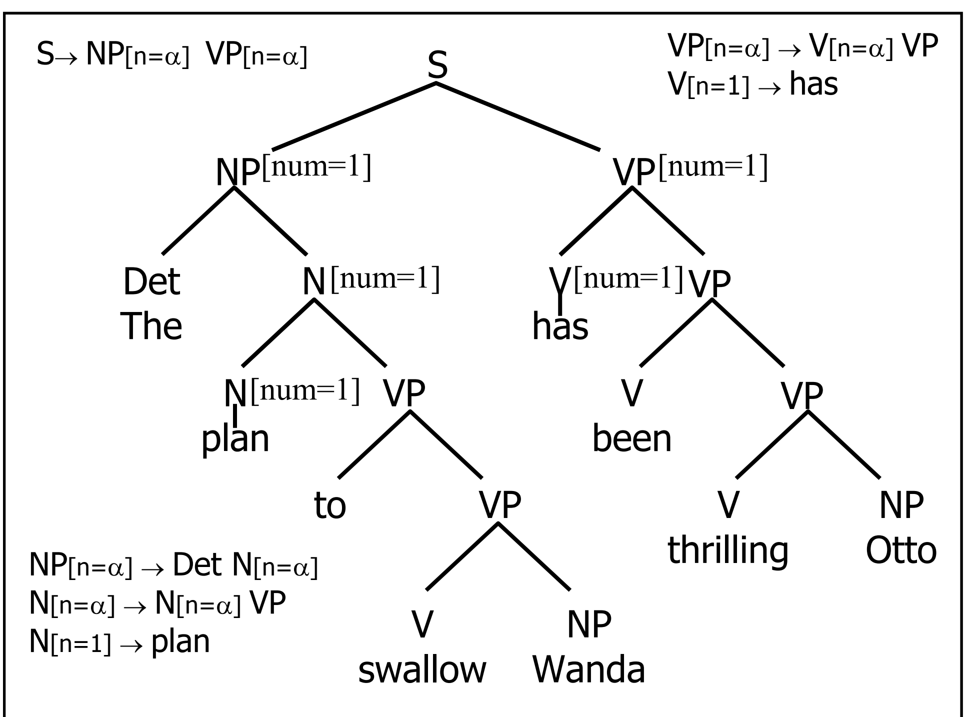


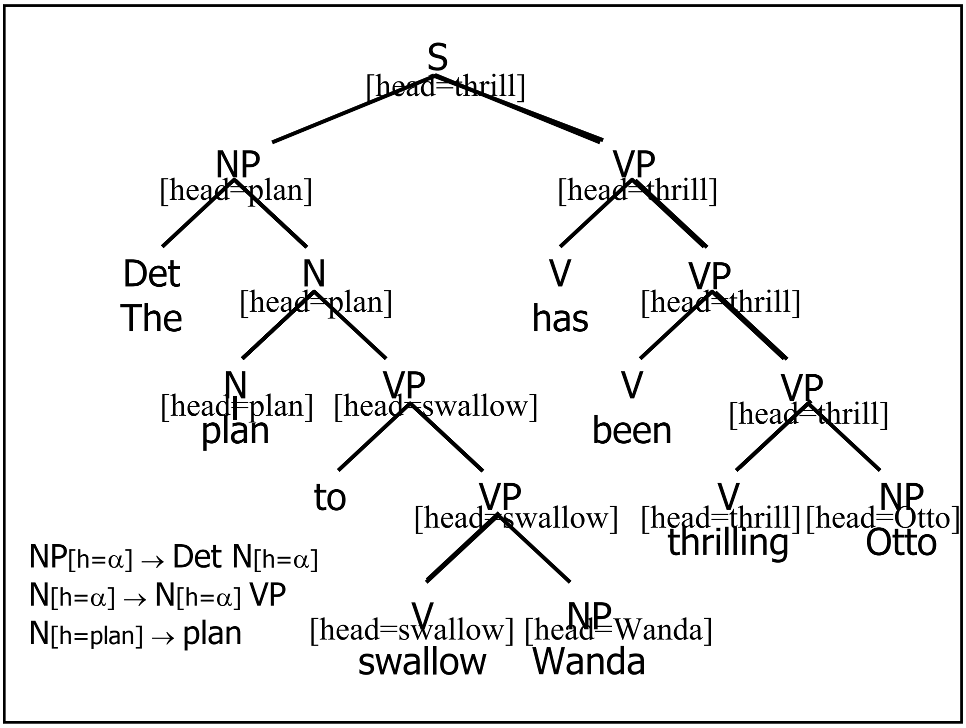
6•863J/9•611J SP04 Lecture 14

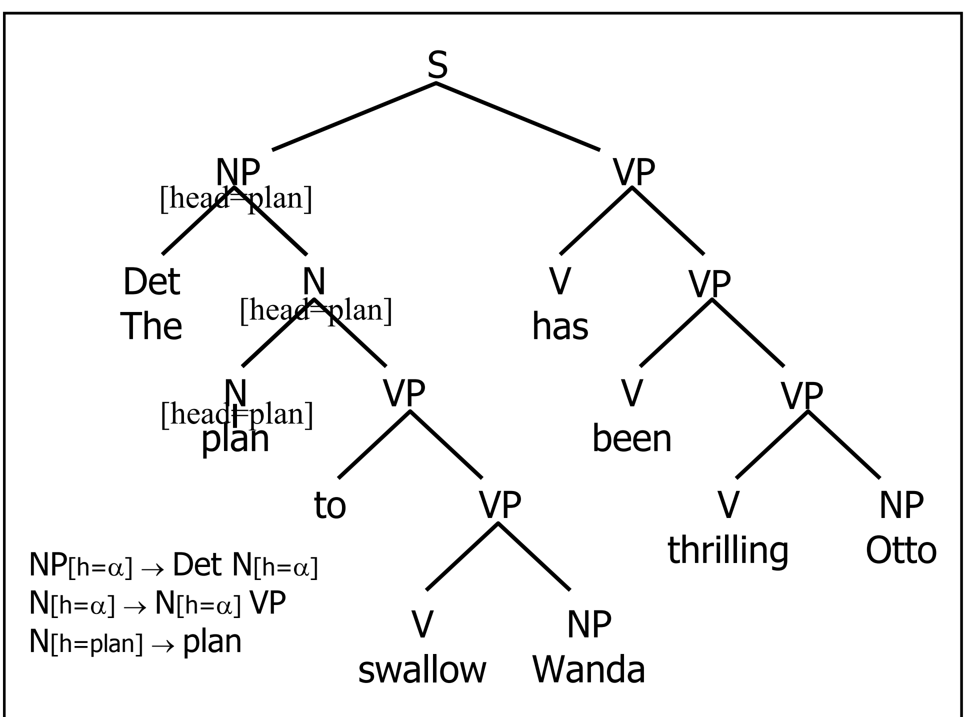


Feature extensions to other parts of syntax

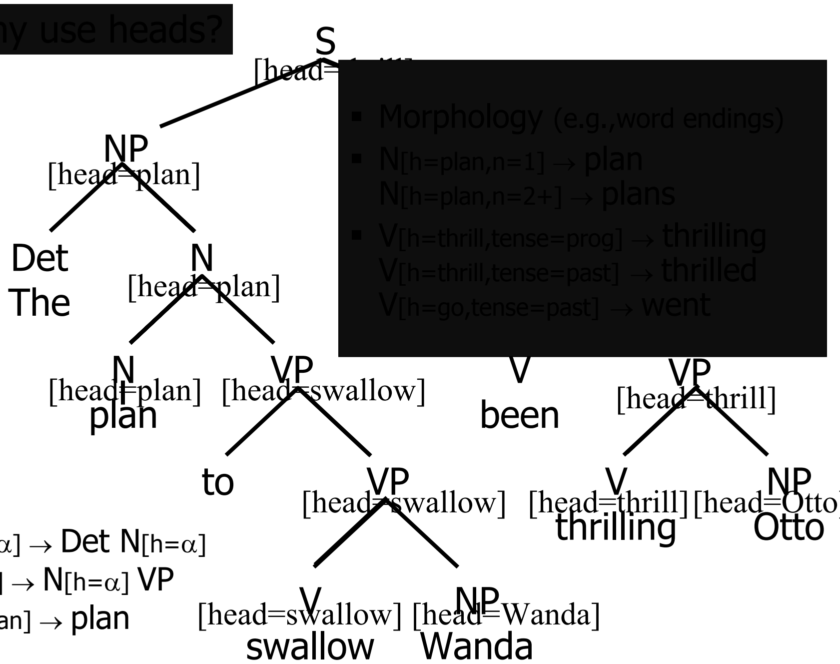

S→ NP[n=1] VP[n=1]
VP[n=1] → V[n=1] VP
V[n=1] → has
S
NP[num=1]
VP[num=1]
Det
The
N[num=1]
V[num=1]
has
VP
N[num=1]
plan
VP
V
been
VP
to
VP
V
thrilling
NP
Otto
NP[n=1] → Det N[n=1]
N[n=1] → N[n=1] VP
N[n=1] → plan
V
swallow
NP
Wanda


S→ NP[n=α] VP[n=α]
VP[n=α] → V[n=α] VP
V[n=1] → has
S
NP[num=1]
VP[num=1]
Det
The
N[num=1]
V[num=1]
has
VP
N[num=1]
plan
VP
V
been
VP
to
VP
V
thrilling
NP
Otto
NP[n=α] → Det N[n=α]
N[n=α] → N[n=α] VP
N[n=1] → plan
V
swallow
NP
Wanda


S [head=thrill]
NP [head=plan]
VP [head=thrill]
Det The
N [head=plan]
V has
VP [head=thrill]
N [head=plan] plan
VP [head=swallow]
V been
VP [head=thrill]
to
VP [head=swallow]
V [head=thrill] thrilling
NP [head=Otto] Otto
V [head=swallow] swallow
NP [head=Wanda] Wanda
NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan


S
NP [head=plan]
VP
Det The
N [head=plan]
V has
VP
N [head=plan] plan
VP
V been
VP
to
VP
V thrilling
NP Otto
V swallow
NP Wanda
NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

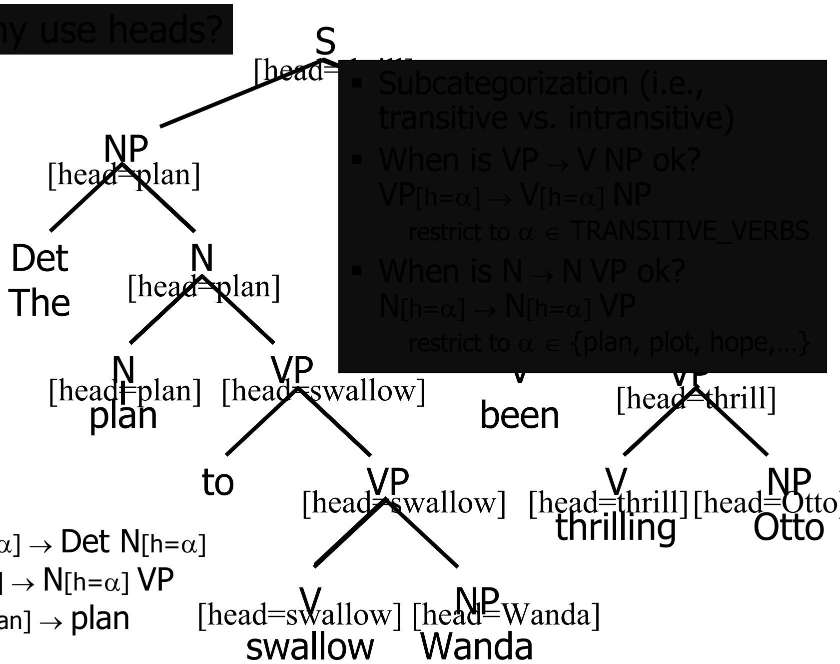
[redacted]



[redacted]

NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

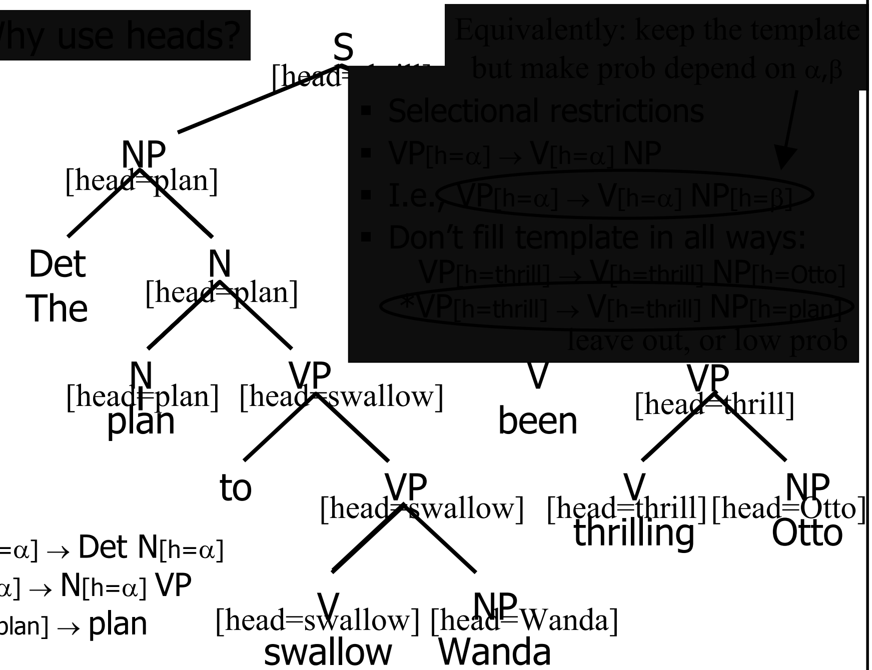
[redacted]



[redacted]

NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

[redacted]



NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

Important question

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

- [Num SG] U [Pers 3] =

- Structures are <u>compatible</u> 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

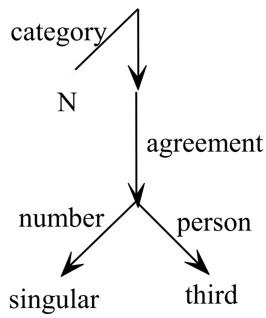
More complex feature structure in the lexical entries

```
Telescope:
[ lex:        telescope
  cat:        N
  gloss:      `telescope
  head:       [ agr:       [ 3sg: +1 ]
                number:    SG
                pos:       N
                proper:    -
               verbal:     - ]
 root_pos     N ]
```

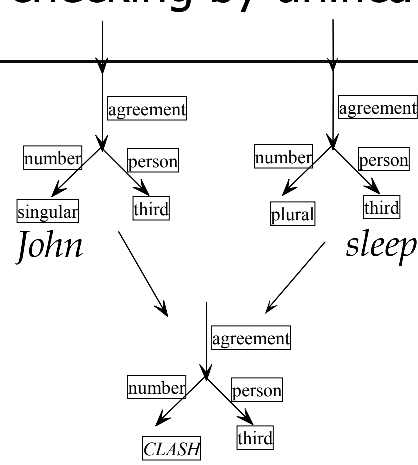
6•863J/9•611J SP04 Lecture 14

Features and grammars

category: N
agreement: [person: third
number: singular]



Feature checking by unification



**John sleep*

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

Claim: ∃ Bound <u>k</u> 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

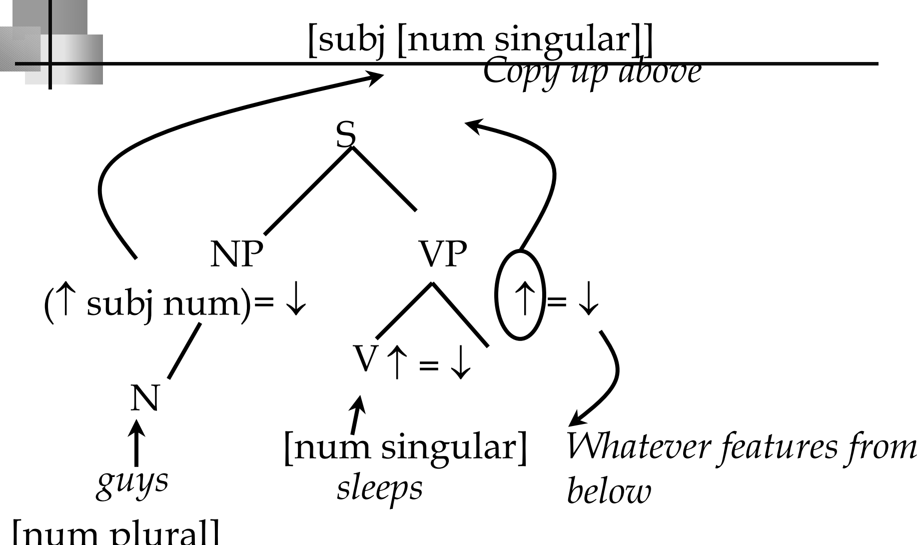
General feature grammars – how violate these properties

- 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'
  $S \rightarrow$ NP VP
  $(\uparrow \text{subj num}) = \downarrow \quad \uparrow = \downarrow$
- 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
- $A \rightarrow A \quad A$

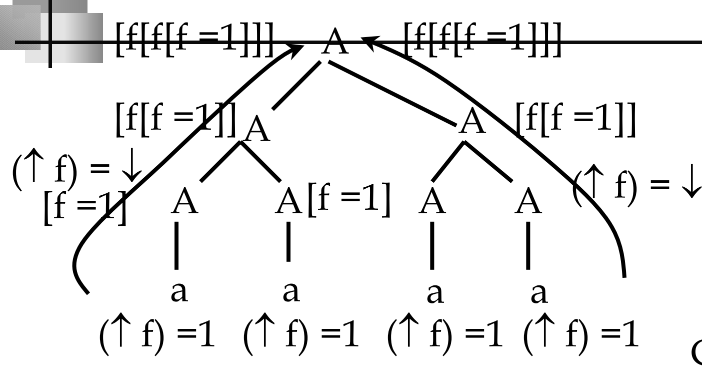
  $(\uparrow f) = \downarrow \quad (\uparrow f) = \downarrow$

  $A \rightarrow a$

  $(\uparrow f) = 1$

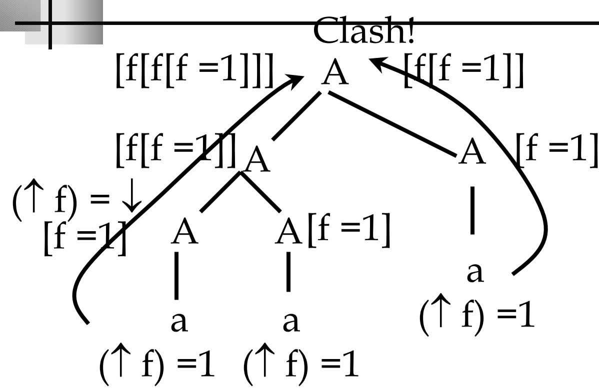
Lets us `count' the number of embeddings on the right & the left – make sure a power of 2

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

Example of what we might do: text understanding via q-answering

```
athena>(top-level)
Shall I clear the database? (y or n) y
sem-interpret>John saw Mary in the park
OK.
sem-interpret>Where did John see Mary
IN THE PARK.
sem-interpret>John gave Fido to Mary
OK.
sem-interpret>Who gave John Fido
I DON'T KNOW
sem-interpret>Who gave Mary Fido
JOHN
sem-interpret >John saw Fido
OK.
sem-interpret>Who did John see
FIDO AND MARY
```


Example of what we might do

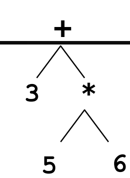

```
athena>(top-level)
Shall I clear the database? (y or n) y
sem-interpret>John saw Mary in the park
OK.
sem-interpret>Where did John see Mary
IN THE PARK.
sem-interpret>John gave Fido to Mary
OK.
sem-interpret>Who gave John Fido
I DON'T KNOW
sem-interpret>Who gave Mary Fido
JOHN
sem-interpret >John saw Fido
OK.
sem-interpret>Who did John see
FIDO AND MARY
```

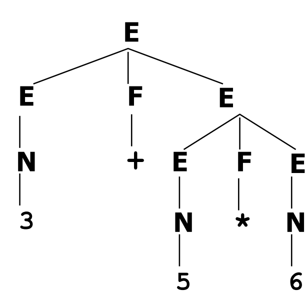
what

- The nature (representation) of meaning representations vs/ *how* these are assembled

Analogy w/ prog. language

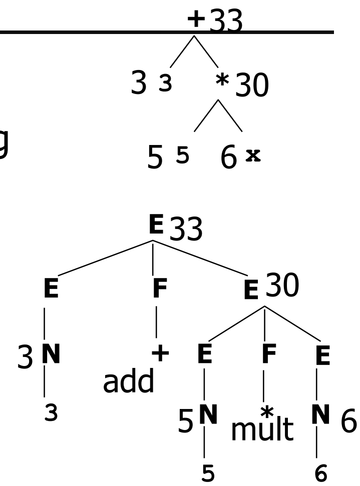
- What is meaning of `3+5*6`?
- First parse it into `3+(5*6)`





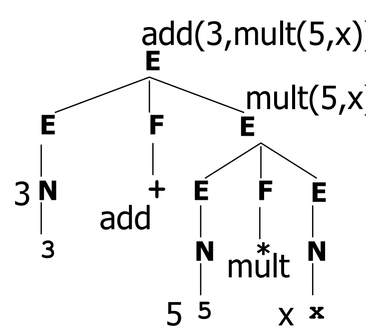
Interpreting in an Environment

- How about `3+5*x`?
- Same thing: the meaning of `x` is found from the environment (it's 6)
- Analogies in language?



Compiling

- How about `3+5*x`?
- Don't know `x` at compile time
- "Meaning" at a node is a piece of code, not a number



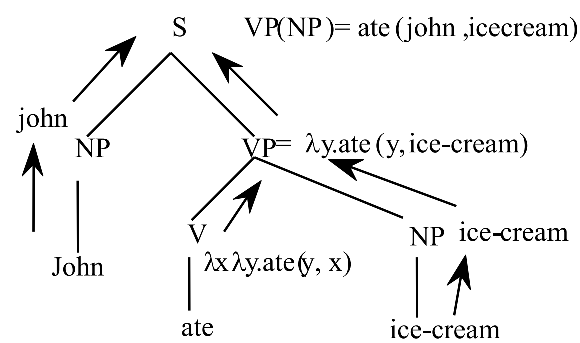
`5*(x+1)-2` is a different expression that produces *equivalent* code (can be converted to the previous code by optimization)

Analogies in language?

What

- What representation do we want for something like
  John ate ice-cream →
  ate(John, ice-cream)
- Lambda calculus
- We'll have to posit something that will do the work
- Predicate of 2 arguments:
  $\lambda x\ \lambda y\ ate(y, x)$

Why: recover meaning from structure



What's meaning? What's semantics – 2 ends of the spectrum

- Answer 1: whatever it is, it's mapping (translation) between representations
  And it depends on *all* of the text
- Answer 2: whatever it is, our answer depends on a much more focused task-specific question, viz., information extraction from texts
- Perhaps call this 'natural language engineering'

- These two ends of the spectrum have different characteristics, and difft uses
- Deep vs. Shallow?

Answer 1: translation – from 'syntactic' rep to 'semantic' rep, aka "Deep"

- Mirrors the progamming language approach
- When is it used?
- DB Q&A (but answer 2 can be used here…when and how?)
- Text understanding: when *all* the text is relevant - voice, inference, paraphrase, important
- Intentions, beliefs, desires (non-extensional= not just sets of items)

Answer 2 – 'Shallow' – information extraction

- What do we need to know to get this task done?
- Slot-and-filler semantics
- Limited parsing, limited predicate-arguments
- Let's see what we need to know about 'meaning' by looking at an example

Example – news stories/MUC

<u>Bridgestone Sports Co.</u> said Friday it has <u>set up a joint venture</u> in Taiwan <u>with a local concern</u> and <u>a Japanese trading house</u> to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and "metal wood" clubs a month.

TIE-UP-1:

Relationship:	**TIE-UP**
Entities:	**"Bridgestone Sports Co."**
	"a local concern"
	"a Japanese trading house"
Joint Venture Company:	**"Bridgestone Sports Taiwan Co."**
Activity:	**ACTIVITY-1**
Amount:	**NT$20000000**

ACTIVITY-1:

Activity:	**PRODUCTION**
Company:	**"Bridgestone Sports Taiwan Co."**
Product:	**"iron and `metal wood' clubs"**

Vs. this task…

Person: Put the blue block on the pyramid

System: I'm going to have to clear off the pyramid. Oops, I can't do that – a pyramid can't support the block.

OK, move it onto the red block.

OK.

What supports the blue block?

The red block.

Key questions

- What do we have to *know* in order to get the job done?
- And then – how do we *represent* this knowledge?
- And then – how do we *compute* with this representation?
- (cf. David Marr's notions)

Answers defined in terms of characteristics of 'the task'

- Information extraction
  - Function is communication of factual information
  - Typically only parts of the text are relevant
  - Typically only part of a relevant sentence is relevant
  - Only predicate-argument structure needed (at a superficial level)
  - No modeling of author or audience

"Logical" Form

- Context-independent meaning
  - Produced directly from the syntax
  - Ignores the utterance context
- Example: *The ball is red*
  - Assigning an exact (contextual) meaning requires knowing *which ball*
  - Logical form an *intermediate step* in full meaning representation

Logical Form [2]

- Includes *indexical terms*
  - Pronouns (e.g., *I, you*)
  - Generic NP (e.g., *a ball, the ball*)
  - Any term whose exact denotation can only be determined from context
- Logical form allows compact representation of indexical terms
  - e.g. (RED1 <THE b1 BALL>) vs. (OR b1 b4 b12 b45 …)

Events

- To retrieve an exact meaning, we must combine LF with a particular context or *event*
- An event might be represented as a set of objects and relations:
  {(BALL B0005), (PERSON P86), (OWNS P86 B0005)}

Word Senses & Ambiguity

- Q: Can the basic unit of LF be a word?
- A: No, words have different *senses*
- Example: *go* has many senses (to move, depart, pass, vanish, reach, extend, …)
- Senses are organized into an *ontology*

Word Senses [2]

- Ontology
  - Example: Aristotle's classes
    - substance (physical objects)
    - quantity (e.g., numbers)
    - quality (e.g., being red)
    - Others: relation, place, time, position, state, action, affection
  - Important: actions, events
    - Provide a structure for organizing the interpretation of sentences

Actions and Events

- *We lifted the box. It was hard work.*
  - The pronoun *it* refers to the whole action (not just *the box*)
- *We lifted the box. It was heavy.*
  - The pronoun *it* refers to *the box*

Semantic Ambiguity

- Parallel to syntactic ambiguity
  - *Happy [cats and dogs] live on the farm*
  - *[Happy cats] and dogs live on the farm*
- Independent of syntactic structure
  - *Every boy loves a dog*
  - "all boys love a single dog"
  - "foreach boy, there is a dog he loves"

Logical Form Language

- Similar to first-order predicate calculus (FOPC)
- Constants: word senses
- Terms: constants that describe objects in the world
- Predicates: constants that describe relations or properties
- Propositions: predicate + terms

Predicates

- *Fido is a dog*
  (DOG1 FIDO1)
  unary predicate
- *Sue loves Jack*
  (LOVES1 SUE1 JACK1)
  binary predicate
- We shall place this into an event structure:

Event(Loves1 :Agent Sue1 :Patient Jack1
Time: present)

Word Senses

- Proper names: terms
  JACK1
- Common nouns: unary predicates
  (DOG1 <>)
- Verbs: n-ary predicates (really n?)
  (BREAK1 <> <>)

Operators

- Logical Operators
  - *not*, *or*, *and*, *if*, *only if*, …
- Logical form supports two kinds of operators:
  - as word senses (if the operator is part of the utterance)
  - as logical operators (if the operator isn't part of the utterance)

Operators [2]

- Examples
  - *Jack loves Sue or Jack loves Mary*
    (OR1 (LOVES1 JACK1 SUE1)(LOVES1 JACK1 MARY1))
  - *Jack loves Sue, Bill loves Mary*
    (& (LOVES1 JACK1 SUE1)(LOVES1 BILL1 MARY1))

Quantifiers

- FOPC: only universal and existential quantifiers: ∀, ∃
- English: much larger range: (Is this true?)
  - *all*, *some*, *most*, *many*, *a few*, *the*, …
- Generalized Quantifiers
  (<quantifier> <variable> : <restriction-proposition>
  <body-proposition>)

Quantifiers [2]

- *Most dogs bark*
  (MOST1 d1:(DOG1 d1)(BARKS1 d1))
- *Most barking things are dogs*
  (MOST1 d1:(BARKS d1)(DOG1 d1))
- *The dog barks*
  (THE x:(DOG1 x)(BARKS1 x))

Plural Forms [2]

- Distributive reading
  The dogs bark
  "There is a set of dogs, and each one barks"
- Collective reading
  The dogs met at the corner
  "*There is a set of dogs, and each one met at the corner"

Ambiguous Plurals

- Some sentences allow both collective and distributive readings

  Two guys bought a stereo
  "Each guy bought a stereo"
  "The two guys bought a stereo together"