

Einführung in die Computerlinguistik und Sprachtechnologie

Vorlesung im WS 2009/10

Prof. Dr. Udo Hahn

Lehrstuhl für Computerlinguistik
Institut für Germanistische Sprachwissenschaft
Friedrich-Schiller-Universität Jena

Exemplarisches zum Parsing

- Parsing-Protokoll: Strukturbaum
- Parsing-Strategien
 - top-down vs. bottom-up
 - depth-first vs. breadth-first
 - Left corner
- Parsing und Ambiguität
 - lexikalische Ambiguität
 - strukturelle Ambiguität

2/45

der folgende Teil der Vorlesung
ist einer Ausarbeitung entnommen von

Prof. Harold Somers

Professor of Language Engineering
University of Manchester
Institute of Science & Technology
(UMIST)

1. Top-down with simple grammar

~~the man shot~~ an elephant

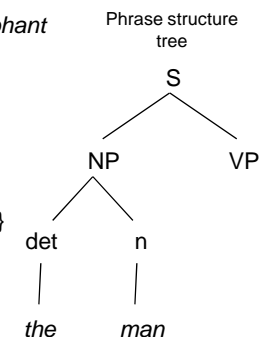
$S \rightarrow NP VP$

$NP \rightarrow det n$

$det \rightarrow \{an, the\}$

$n \rightarrow \{elephant, man\}$

~~$VP \rightarrow v$~~
 $VP \rightarrow v NP$



$S \rightarrow NP VP$

$NP \rightarrow det n$

$VP \rightarrow v$

$VP \rightarrow v NP$

Lexicon

$det \rightarrow \{an, the\}$

$n \rightarrow \{elephant, man\}$

$v \rightarrow shot$

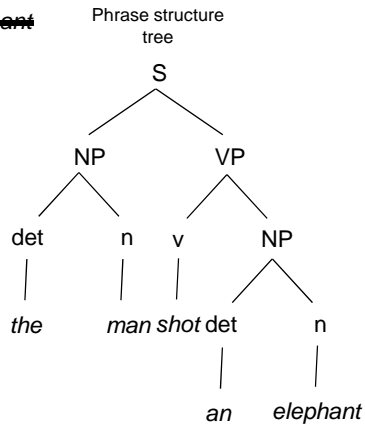
No more rules, but input is not completely accounted for...
So we must backtrack, and try the other VP rule

4/45

1. Top-down with simple grammar

~~the man shot an elephant~~

~~VP → v~~
 S → NP VP
 NP → det n
 det → {an, the}
 n → {elephant, man}



S → NP VP
 NP → det n
 VP → v
 VP → v NP

Lexicon
 det → {an, the}
 n → {elephant, man}
 v → shot

No more rules, and input is completely accounted for

5/45

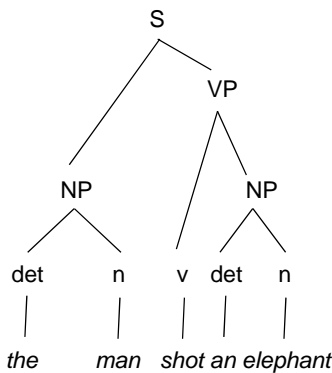
Breadth-first vs. Depth-first

- When we came to the VP rule we were faced with a choice of two rules
 - “Depth-first” means following the first choice through to the end
 - “Breadth-first” means keeping all your options open (e.g., by copying partial parse trees and by proceeding with different alternatives independently)

6/45

2. Bottom-up with simple grammar

det → {an, the}
 n → {elephant, man}
 v → shot
 NP → det n
 VP → v NP
 S → NP VP



S → NP VP
 NP → det n
 VP → v
 VP → v NP

Lexicon
 det → {an, the}
 n → {elephant, man}
 v → shot

We've reached the top, and input is completely accounted for

7/45

Same again but with lexical ambiguity

S → NP VP
 NP → det n
 VP → v
 VP → v NP

Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

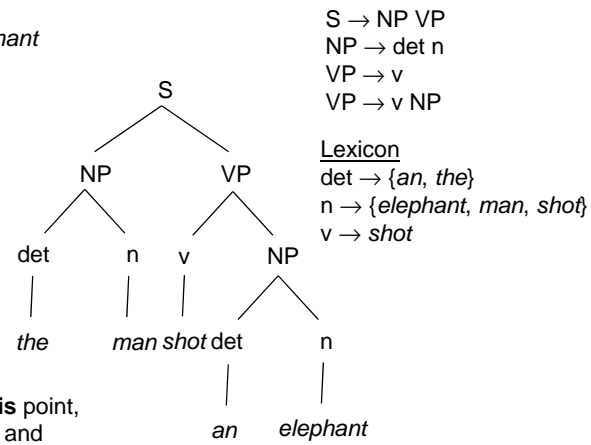
shot can be v or n

8/45

3. Top-down with lexical ambiguity

~~the man~~ shot an elephant

- S → NP VP
- NP → det n
- det → {an, the}
- n → {elephant, man}
- ~~VP → v NP~~
- VP → v NP



Same as before: at this point, we are looking for a v, and shot fits the bill; the n reading never comes into play

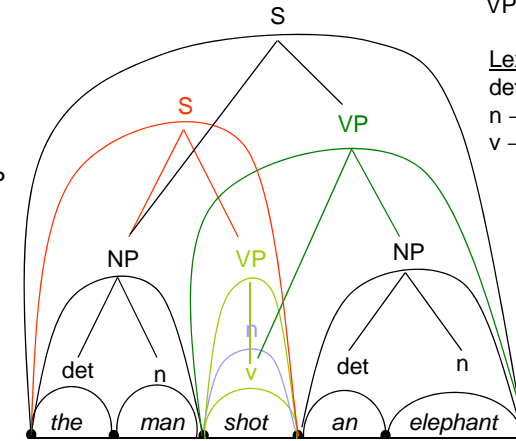
- S → NP VP
 - NP → det n
 - VP → v
 - VP → v NP
- Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

4. Bottom-up with lexical ambiguity

- det → {an, the}
- n → {elephant, man, shot}
- v → shot
- NP → det n
- VP → v
- VP → v NP
- S → NP VP

- S → NP VP
- NP → det n
- VP → v
- VP → v NP

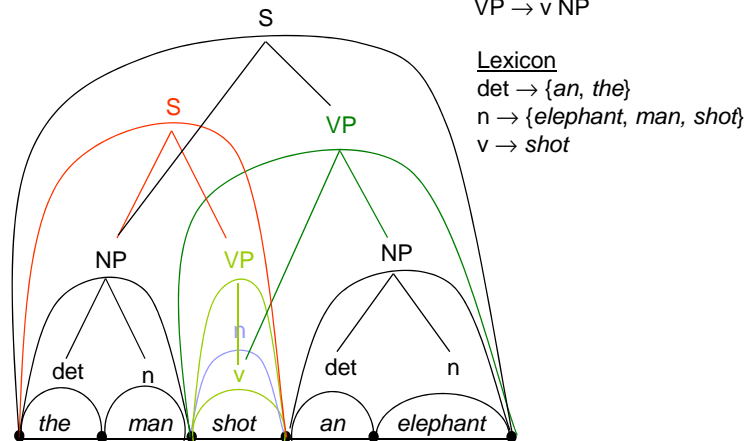
- Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot



Terminology:
 graph
 nodes
 arcs (edges)

4. Bottom-up with lexical ambiguity

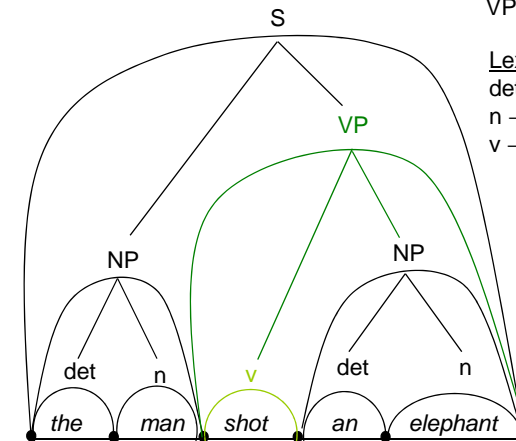
Let's get rid of all the unused arcs



- S → NP VP
 - NP → det n
 - VP → v
 - VP → v NP
- Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

4. Bottom-up with lexical ambiguity

Let's get rid of all the unused arcs



- S → NP VP
- NP → det n
- VP → v
- VP → v NP

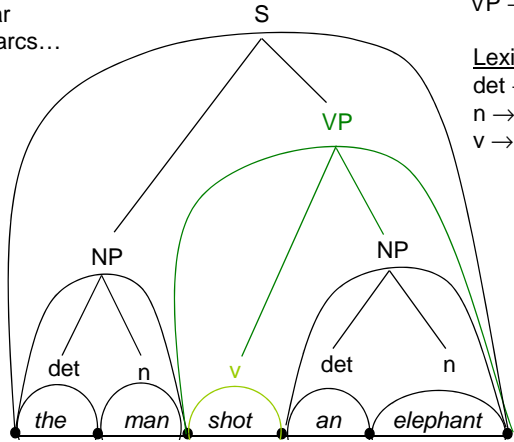
- Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

4. Bottom-up with lexical ambiguity

S → NP VP
 NP → det n
 VP → v
 VP → v NP

Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

And let's clear away all the arcs...

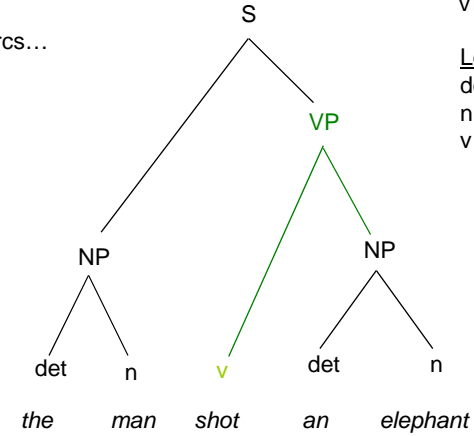


4. Bottom-up with lexical ambiguity

S → NP VP
 NP → det n
 VP → v
 VP → v NP

Lexicon
 det → {an, the}
 n → {elephant, man, shot}
 v → shot

And let's clear away all the arcs...



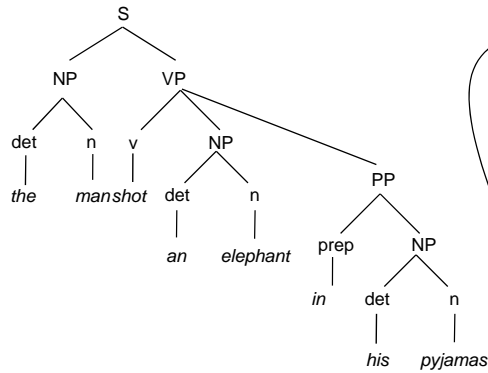
Same again but with structural ambiguity

the man shot an elephant in his pyjamas [remember: Catalan numbers!]

S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

Lexicon
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
 v → shot
 prep → in

We introduce a PP rule in two places



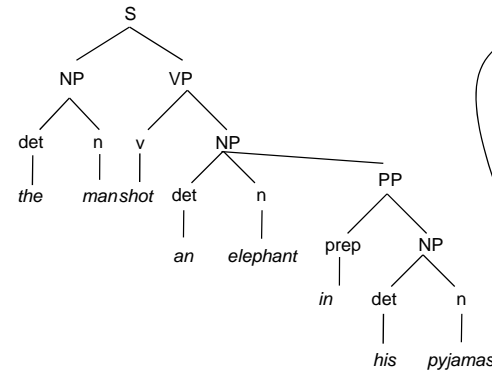
Same again but with structural ambiguity

the man shot an elephant in his pyjamas

S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

Lexicon
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
 v → shot
 prep → in

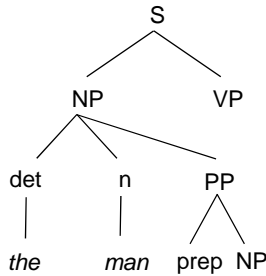
We introduce a PP rule in two places



5. Top-down with structural ambiguity

~~the man~~ shot an elephant in his pyjamas

S → NP VP
 NP → det n
~~NP → det n PP~~
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
 PP → prep NP
 prep → in



S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

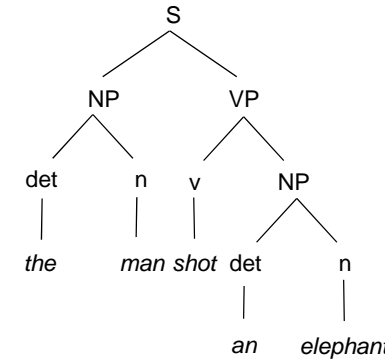
At this point, depending on our strategy (breadth-first vs. depth-first) we may consider the NP complete and look for the VP, or we may try the second NP rule. Let's see what happens in the latter case.

The next word, *shot*, isn't a prep, So this rule simply fails

5. Top-down with structural ambiguity

~~the man shot an elephant~~ in his pyjamas

S → NP VP
 NP → det n
~~NP → det n PP~~
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
~~VP → v~~
~~VP → v NP~~
 VP → v NP PP
 v → shot
~~NP → det n~~
 NP → det n PP
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}



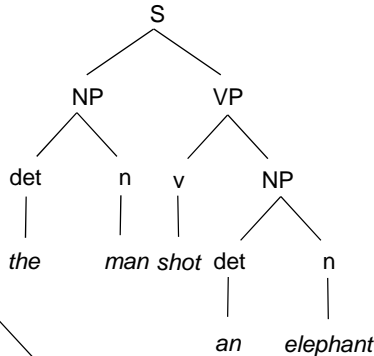
S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

As before, the first VP rule works, But does not account for all the input. Similarly, if we try the second VP rule, and the first NP rule ...

5. Top-down with structural ambiguity

~~the man shot an elephant~~ in his pyjamas

S → NP VP
 NP → det n
~~NP → det n PP~~
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
~~VP → v~~
~~VP → v NP~~
 VP → v NP PP
 v → shot
~~NP → det n~~
 NP → det n PP



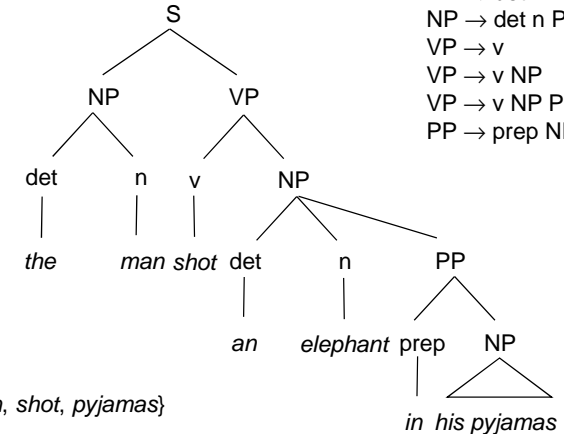
S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

So what do we try next?
 This?
 Or this?

5. Top-down with structural ambiguity (depth-first)

~~the man shot an elephant in his pyjamas~~

S → NP VP
 NP → det n
~~NP → det n PP~~
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
~~VP → v~~
~~VP → v NP~~
 VP → v NP PP
 v → shot
~~NP → det n~~
 NP → det n PP
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
 PP → prep NP
 prep → in

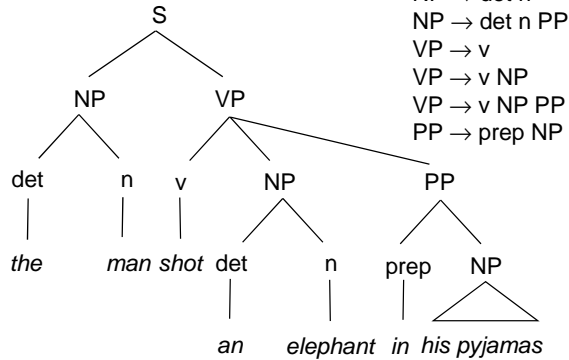


S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

5. Top-down with structural ambiguity (breadth-first)

~~the man shot an elephant in his pyjamas~~

~~S → NP VP~~
~~NP → det n~~
~~NP → det n PP~~
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
~~VP → v~~
~~VP → v NP~~
 VP → v NP PP
 v → shot
 NP → det n
 NP → det n PP
 det → {an, the, his}
 n → {elephant, man, shot, pyjamas}
 PP → prep NP
 prep → in



S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP

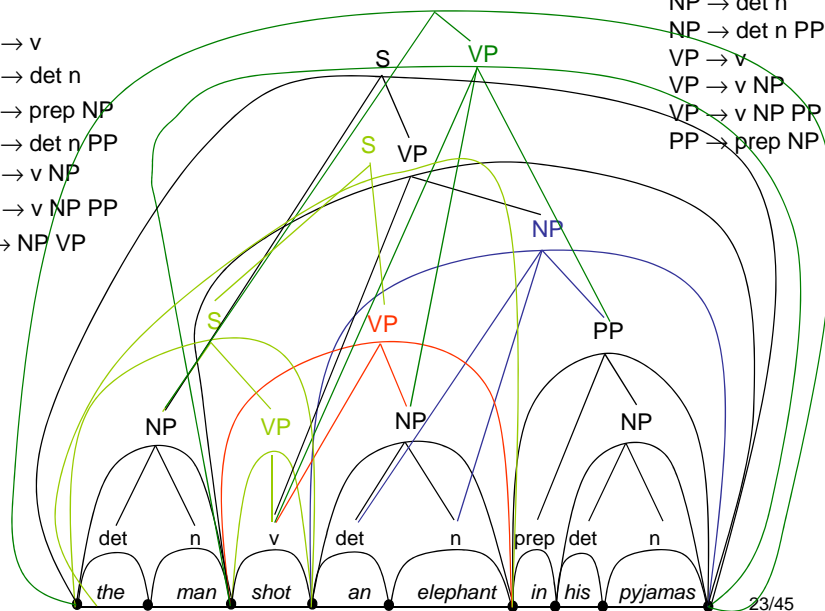
Recognizing ambiguity

- Notice how the choice of strategy determines which result we get (first).
- In both strategies, there are often rules left untried, on the list.
- If we want to know if our input is ambiguous, at some time we do have to follow these through.
- As you will see later, trying out alternative paths can be quite intensive

6. Bottom-up with structural ambiguity

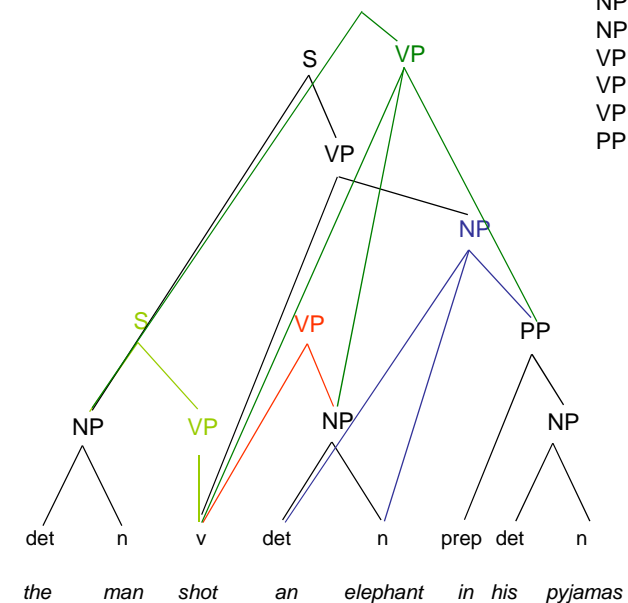
VP → v
 NP → det n
 PP → prep NP
 NP → det n PP
 VP → v NP
 VP → v NP PP
 S → NP VP

S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP



6. Bottom-up with structural ambiguity

S → NP VP
 NP → det n
 NP → det n PP
 VP → v
 VP → v NP
 VP → v NP PP
 PP → prep NP



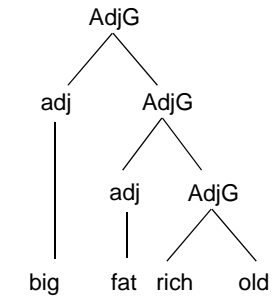
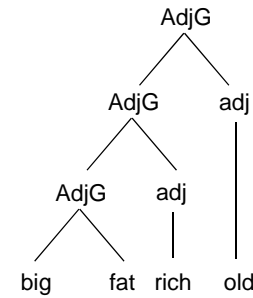
Recursive rules

- “Recursive” rules call themselves
- We already have some recursive rule pairs:
 - NP → det n PP
 - PP → prep NP
- Rules can be immediately recursive
 - AdjG → adj AdjG
 - (the) big fat ugly (man)

Recursive rules

Left recursive
 AdjG → AdjG adj
 AdjG → adj adj

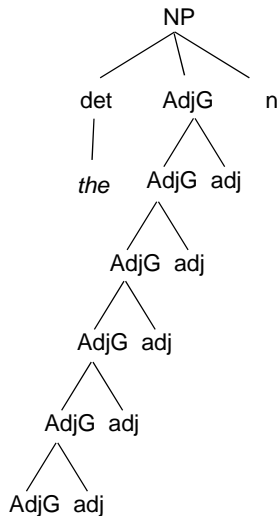
Right recursive
 AdjG → adj AdjG
 AdjG → adj adj



7. Top-down with left recursion

~~the big fat rich old man~~

~~NP → det n~~
 NP → det AdjG n
 AdjG → AdjG adj
 AdjG → adj



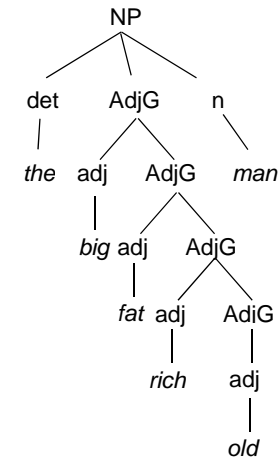
NP → det n
 NP → det AdjG n
 AdjG → AdjG adj
 AdjG → adj

You can't have left-recursive rules with a top-down parser, even if the non-recursive rule is first

7. Top-down with right recursion

~~the big fat rich old man~~

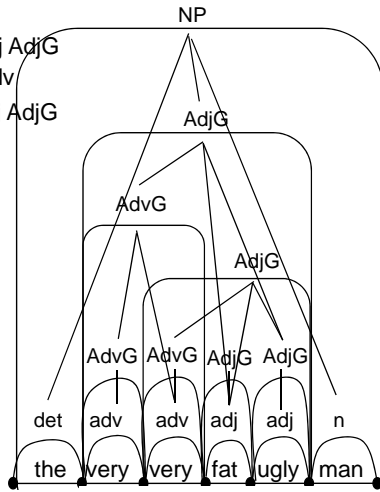
~~NP → det n~~
 NP → det AdjG n
 AdjG → adj AdjG
 AdjG → adj



NP → det n
 NP → det AdjG n
 AdjG → adj AdjG
 AdjG → adj

8. Bottom-up with left and right recursion

- AdjG → adj
- AdvG → adv
- AdjG → AdvG adj AdjG
- AdvG → AdvG adv
- AdjG → AdvG adj AdjG
- NP → det AdjG n



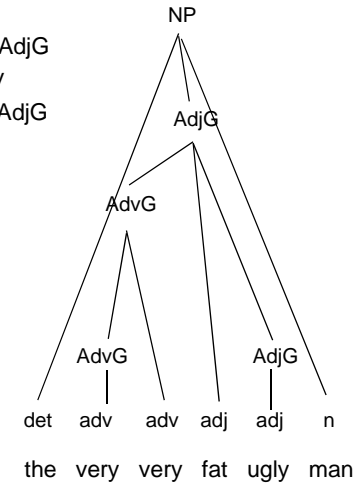
- NP → det n
- NP → det AdjG n
- AdjG → AdvG adj AdjG
- AdjG → adj
- AdvG → AdvG adv
- AdvG → adv

AdjG rule is right recursive,
AdvG rule is left recursive

Quite a few useless
paths, but overall
no difficulty

8. Bottom-up with left and right recursion

- AdjG → adj
- AdvG → adv
- AdjG → AdvG adj AdjG
- AdvG → AdvG adv
- AdjG → AdvG adj AdjG
- NP → det AdjG n



- NP → det n
- NP → det AdjG n
- AdjG → AdvG adj AdjG
- AdjG → adj
- AdvG → AdvG adv
- AdvG → adv

AdjG rule is right recursive,
AdvG rule is left recursive

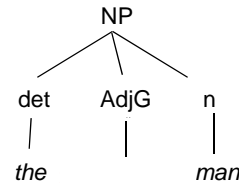
Empty rules

- For example
 - NP → det AdjG n
 - AdjG → adj AdjG
 - AdjG → ε
- Equivalent to
 - NP → det AdjG n
 - NP → det n
 - AdjG → adj
 - AdjG → adj AdjG
- Or
 - NP → det (AdjG) n
 - AdjG → adj (AdjG)

7. Top-down with empty rules

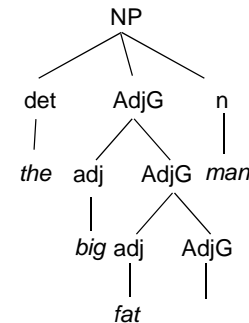
~~the~~ man

- NP → det AdjG n
- AdjG → adj AdjG
- AdjG → ε



~~the big fat~~ man

- NP → det AdjG n
- AdjG → adj AdjG
- AdjG → ε

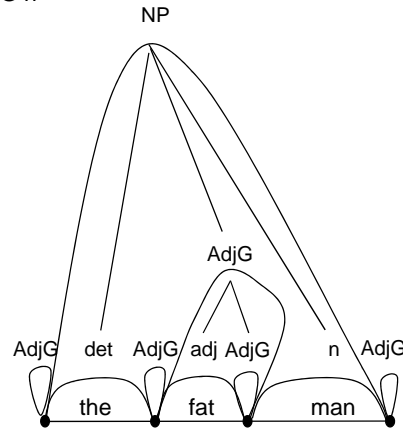


- NP → det AdjG n
- AdjG → adj AdjG
- AdjG → ε

8. Bottom-up with empty rules

AdjG $\rightarrow \epsilon$
 AdjG $\rightarrow \text{adj AdjG}$
 NP $\rightarrow \text{det AdjG n}$

NP $\rightarrow \text{det AdjG n}$
 AdjG $\rightarrow \text{adj AdjG}$
 AdjG $\rightarrow \epsilon$

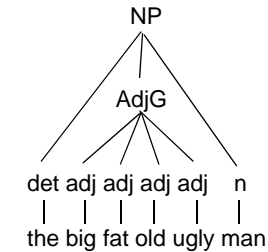


Lots of useless paths, especially in a long sentence, but otherwise no difficulty

33/45

Some additions to formalism

- Recursive rules build unattractive tree structures: you'd rather have flat trees with unrestricted numbers of daughters



- Kleene star
- AdjG $\rightarrow \text{adj}^*$

34/45

Some additions to formalism

- As grammars grow, the rule combinations multiply and it gets clumsy

NP $\rightarrow \text{det n}$
 NP $\rightarrow \text{det AdjG n}$
 NP $\rightarrow \text{det n PP}$
 NP $\rightarrow \text{det AdjG n PP}$
 NP $\rightarrow \text{n}$
 NP $\rightarrow \text{AdjG n}$
 NP $\rightarrow \text{n PP}$
 NP $\rightarrow \text{AdjG n PP}$

NP $\rightarrow (\text{det}) \text{n} (\text{AdjG}) \text{n} (\text{PP})$

35/45

Processing implications

- Parsing with Kleene star
 - Neatly combines empty rules and recursive rules

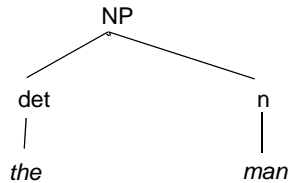
36/45

9. Top-down with Kleene star

$NP \rightarrow \text{det adj}^* n$

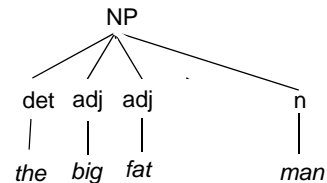
~~the~~ man

$NP \rightarrow \text{det adj}^* n$
 $\text{adj}^* = \text{adj adj}^*$
 $\text{adj}^* = \epsilon$



~~the big fat~~ man

$NP \rightarrow \text{det adj}^* n$
 $\text{adj}^* = \text{adj adj}^*$
 $\text{adj}^* = \epsilon$

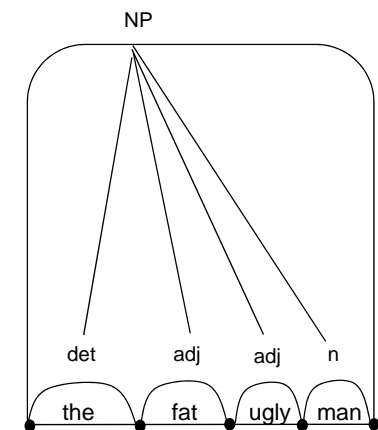
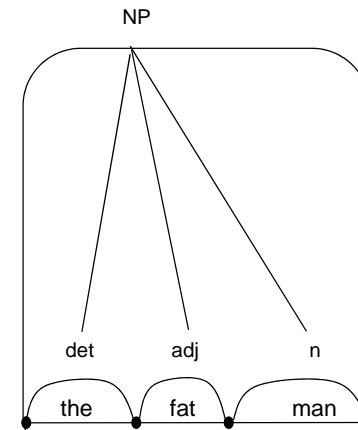


37/45

10. Bottom-up with Kleene star

$NP \rightarrow \text{det adj}^* n$

$NP \rightarrow \text{det adj}^* n$



38/45

Processing implications

- Parsing with bracketed symbols
 - Parser has to expand rules
 - either in a single pass beforehand
 - or (better) on the fly (as it comes to them)
 - So bracketing convention is just a convenience for rule-writers

$NP \rightarrow (\text{det}) n (\text{AdjG}) n (\text{PP})$

$NP \rightarrow \text{det } n (\text{AdjG}) n (\text{PP})$

$NP \rightarrow n (\text{AdjG}) n (\text{PP})$

39/45

Top down vs. bottom-up

- Bottom-up builds many useless trees
- Top-down can propose false trails, sometimes quite long, which are only abandoned when they reach the word level
 - Especially a problem if breadth-first
- Bottom-up very inefficient with empty rules
- Top-down CANNOT handle left-recursion
- Top-down cannot do partial parsing
 - Especially useful for speech
- Wouldn't it be nice to combine them to get the advantages of both?

40/45

Left-corner parsing

- The “left corner” of a rule is the first symbol after the rewrite arrow
 - e.g. in $S \rightarrow NP VP$, the left corner is NP.
- Left corner parsing starts bottom-up, taking the first item off the input and finding a rule for which it is the left corner.
- This provides a top-down prediction, but we continue working bottom-up until the prediction is fulfilled.
- When a rule is completed, apply the left-corner principle: is that completed constituent a left-corner?

41/45

9. Left-corner with simple grammar

the man shot an elephant

$NP \rightarrow \text{det } n$

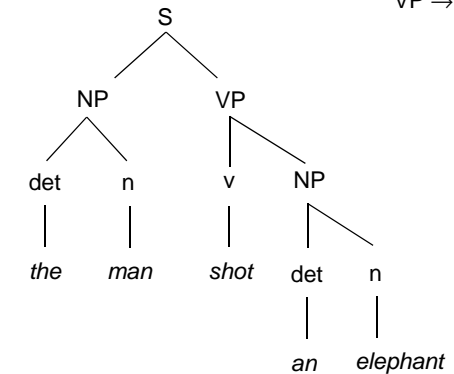
$S \rightarrow NP VP$

$VP \rightarrow v$

but text not all accounted for, so try

$VP \rightarrow v NP$

$NP \rightarrow \text{det } n$



$S \rightarrow NP VP$

$NP \rightarrow \text{det } n$

$VP \rightarrow v$

$VP \rightarrow v NP$

42/45