Einführung in die Computerlinguistik und Sprachtechnologie

Vorlesung im WiSe 2018/19
(B-GSW-12)

Prof. Dr. Udo Hahn

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

http://www.julielab.de

Exemplarisches zum Parsing

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

der folgende Teil der Vorlesung enthält Teile 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 → NP VP

NP → det n

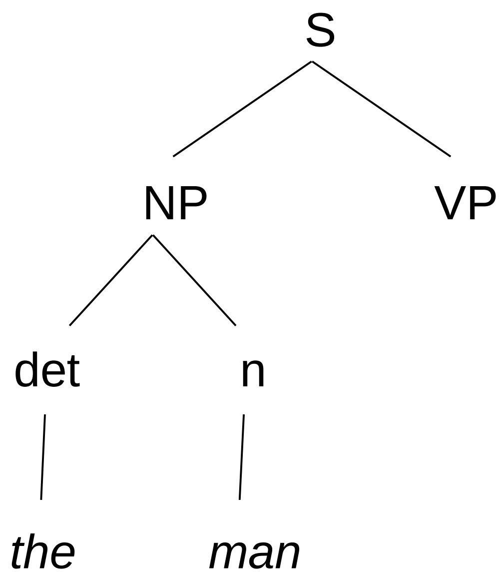
det → {*an*, *the*}

n → {*elephant*, *man*}

~~VP → v~~

VP → v NP

Phrase structure tree



S → NP VP
NP → det n
VP → v
VP → v NP

<u>Lexicon</u>
det → {*an*, *the*}
n → {*elephant*, *man*}
v → *shot*

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

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 → v~~

$VP \rightarrow v\ NP$

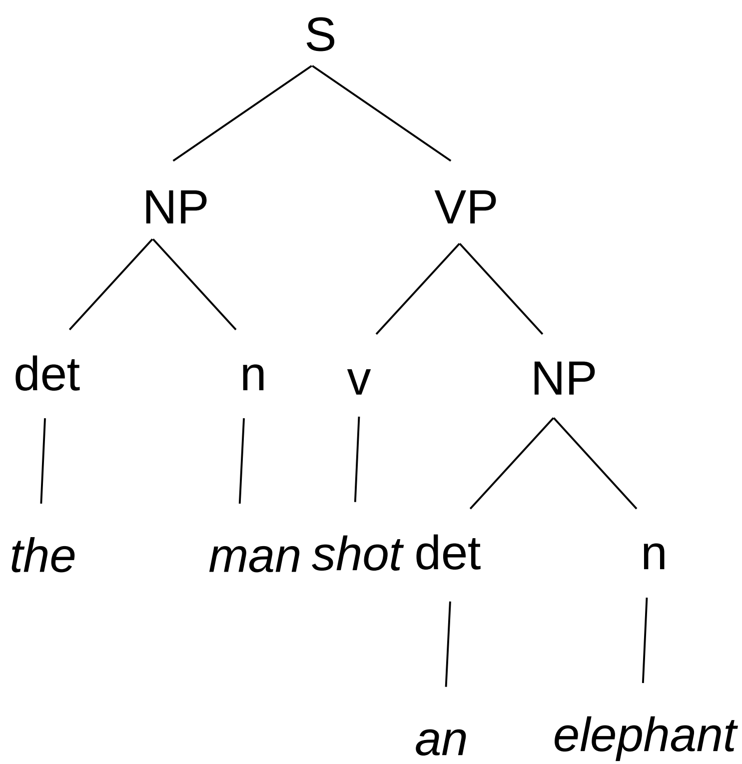
v $\rightarrow$ *shot*

$NP \rightarrow det\ n$

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

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

Phrase structure tree



$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$VP \rightarrow v$
$VP \rightarrow v\ NP$

<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man*}
v $\rightarrow$ *shot*

No more rules, and input *is* completely accounted for

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
    - In case a dead end occurs, we must backtrack
  - "Breadth-first" means keeping all your options open (e.g., by copying partial parse trees and by proceeding with different alternatives independently)
    - In case a dead end occurs, the current alternative "dies" (is invalidated)

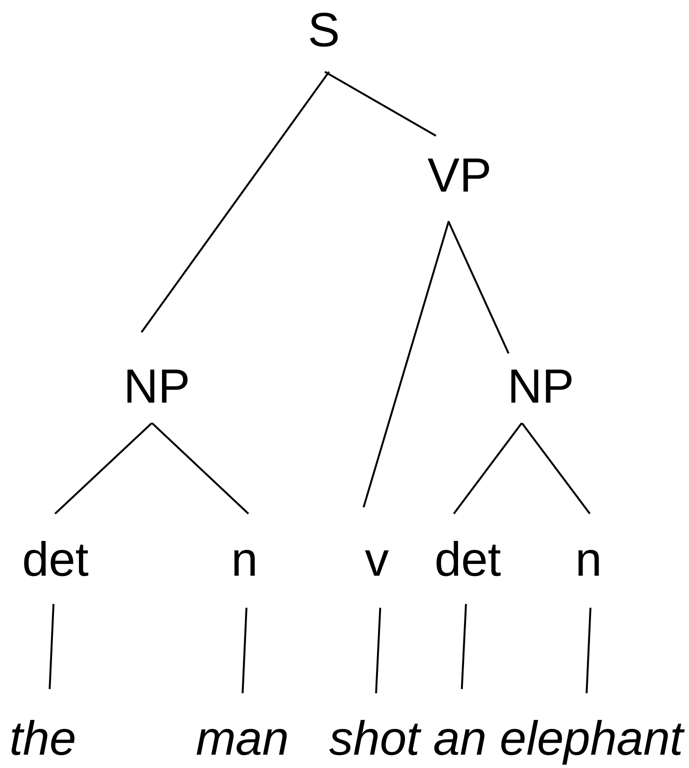
2. Bottom-up with simple grammar

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

<u>Lexicon</u>
det → {*an*, *the*}
n → {*elephant*, *man*}
v → *shot*

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

Eine Beispielgrammatik (CFG)

(1-a) S ⇨ NP VP

(1-b) NP ⇨ Det Nom

(1-c) VP ⇨ Verb NP

(1-d) Det ⇨ { der, den }

(1-e) Nom ⇨ { Präsident, Vertrag }

(1-f) Verb ⇨ { unterschreibt }

Einfacher Bottom-Up-CFG-Parser

(Algorithmenskizze)

<u>Eingabe</u>:

a. ein Satz bestehend aus *n* lexikalischen Einheiten einer (natürlichen) Sprache

b. eine kontextfreie Phrasenstruktur-Grammatik (deren terminale Symbole lexikalische Einheiten sind)

1. Prüfe für alle *n* lexikalischen Einheiten des Satzes (von links nach rechts):

   a. es existiert eine Regel, deren rechte Seite die jeweilige lexikalische Einheit als Element enthält ; falls nicht, gehe zu (5.b)

   b. konstruiere für jede so identifizierte lexikalische Einheit einen Graphen, dessen Wurzel das lexikalische Kategoriensymbol der linken Seite dieser Regel als Etikett erhält und die lexikalische Einheit aus der rechten Seite der Regel als Etikett des direkten Nachfolgers der Wurzel hat.

   c. jeder dieser „lexikalischen" Graphen hat einen Abdeckungsindex [i…i], der die Abdeckung des jeweiligen *i*-ten lexikalischen Element beschreibt (i=1..n).

   d. ordne die Graphen in aufsteigender Folge ihrer *n* Abdeckungsindices.

2. Finde iterativ alle Regeln,

   a. deren rechte Seite (bestehend aus *k* Kategoriensymbolen) vollständig mit den Etiketten der Wurzeln einer *k*-elementigen Teilfolge von Graphen (ebenfalls Kategoriensymbole) übereinstimmt, wobei

   b. die Abdeckungsindices der *k*-elementigen Teilfolge von Graphen dicht (also ohne Wort- oder Phrasenlücken) und numerisch aufsteigend geordnet sind.

3. Konstruiere für jede so gefundene *k*-elementige Teilfolge von Graphen einen die gefundenen Graphen enthaltenden neuen Graphen,

   a. dessen Wurzel das komplexe syntaktische Kategoriensymbol der linken Seite der identifizierten Regel als Etikett erhält und die Wurzelknoten der identifizierten *k*-elementigen Teilfolge von Graphen aus der korrespondierenden rechten Seite dieser Regel als direkte Nachfolger des neuen Wurzelknotens hat, wobei die Präzedenzordnung zwischen diesen Teilgraphen sich aus der aufsteigenden Ordnung ihrer Abdeckungsindices ergibt.

   b. der Abdeckungsindex des neu gebildeten Graphen berechnet sich wie folgt:

      (1) die untere Grenze ergibt sich aus der unteren Grenze des ersten Graphen der total geordneten *k*-elementigen Folge von Graphen,

      (2) die obere Grenze ergibt sich aus der oberen Grenze des k-ten Graphen der total geordneten *k*-elementigen Folge von Graphen.

4. Iteriere Schritte (2.) und (3.) solange, bis ein Graph mit dem Startsymbol der Grammatik (S) als Etikett einer Wurzel gebildet ist, die Vorgänger aller Knoten des Graphen ist und den Eingabesatz komplett abgedeckt (d.h. S alle lexikalische Kategorien dominiert und damit einen Abdeckungsindex [1..n] hat. Falls kein entsprechender Graph gebildet werden kann, gehe zu (5.b)

5. <u>Ausgabe</u>:

   a. der Syntaxbaum für den Eingabesatz, falls S alle *n* lexikalische Kategorien dominiert

   b. „kein Satz der geg. Grammatik", sonst

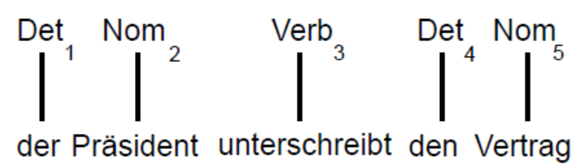
Einfacher Bottom-Up-CFG-Parser

(Algorithmenskizze)

Eingabe:

a. ein Satz bestehend aus *n* lexikalischen Einheiten einer (natürlichen) Sprache
b. eine kontextfreie Phrasenstruktur-Grammatik (deren terminale Symbole lexikalische Einheiten sind)

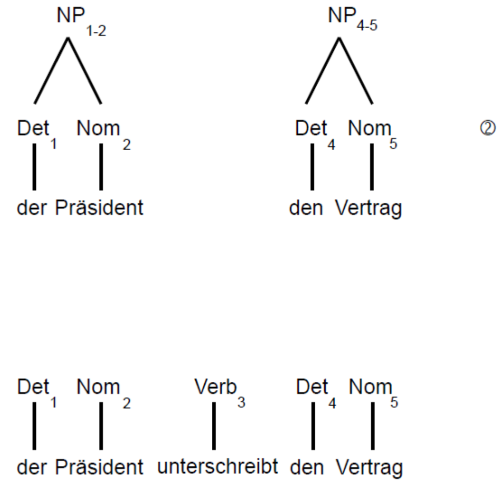
1. Prüfe für alle *n* lexikalischen Einheiten des Satzes (von links nach rechts):
   a. es existiert eine Regel, deren rechte Seite die jeweilige lexikalische Einheit als Element enthält ; falls nicht, gehe zu (5.b)
   b. konstruiere für jede so identifizierte lexikalische Einheit einen Graphen, dessen Wurzel das lexikalische Kategoriensymbol der linken Seite dieser Regel als Etikett erhält und die lexikalische Einheit aus der rechten Seite der Regel als Etikett des direkten Nachfolgers der Wurzel hat.
   c. jeder dieser „lexikalischen" Graphen hat einen Abdeckungsindex [i…i], der die Abdeckung des jeweiligen *i*-ten lexikalischen Element beschreibt (i=1..n).
   d. ordne die Graphen in aufsteigender Folge ihrer *n* Abdeckungsindices.



①

Einfacher Bottom-Up-CFG-Parser

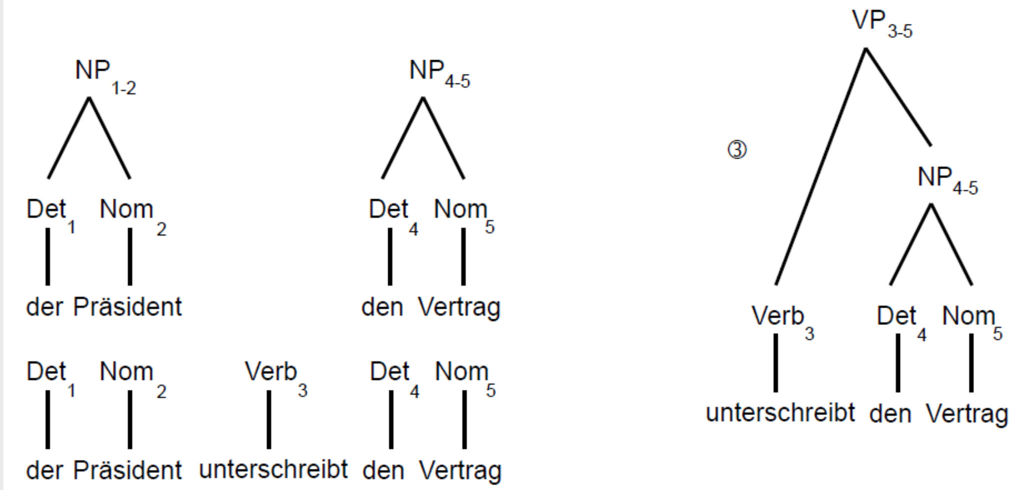
(Algorithmenskizze)



2. Finde iterativ alle Regeln,
   a. deren rechte Seite (bestehend aus *k* Kategoriensymbolen) vollständig mit den Etiketten der Wurzeln einer *k*-elementigen Teilfolge von Graphen (ebenfalls Kategoriensymbole) übereinstimmt, wobei
   b. die Abdeckungsindices der *k*-elementigen Teilfolge von Graphen dicht (also ohne Wort- oder Phrasenlücken) und numerisch aufsteigend geordnet sind.
3. Konstruiere für jede so gefundene *k*-elementige Teilfolge von Graphen einen die gefundenen Graphen enthaltenden neuen Graphen,
   a. dessen Wurzel das komplexe syntaktische Kategoriensymbol der linken Seite der identifizierten Regel als Etikett erhält und die Wurzelknoten der identifizierten *k*-elementigen Teilfolge von Graphen aus der korrespondierenden rechten Seite dieser Regel als direkte Nachfolger des neuen Wurzelknotens hat, wobei die Präzedenzordnung zwischen diesen Teilgraphen sich aus der aufsteigenden Ordnung ihrer Abdeckungsindices ergibt.
   b. der Abdeckungsindex des neu gebildeten Graphen berechnet sich wie folgt:
      (1) die untere Grenze ergibt sich aus der unteren Grenze des ersten Graphen der total geordneten *k*-elementigen Folge von Graphen,
      (2) die obere Grenze ergibt sich aus der oberen Grenze des k-ten Graphen der total geordneten *k*-elementigen Folge von Graphen.
4. Iteriere Schritte (2.) und (3.) solange, bis ein Graph mit dem Startsymbol der Grammatik (S) als Etikett einer Wurzel gebildet ist, die Vorgänger aller Knoten des Graphen ist und den Eingabesatz komplett abgedeckt (d.h. S alle lexikalische Kategorien dominiert und damit einen Abdeckungsindex [1..n] hat. Falls kein entsprechender Graph gebildet werden kann, gehe zu (5.b)
5. <u>Ausgabe</u>:
   a. der Syntaxbaum für den Eingabesatz, falls S alle *n* lexikalische Kategorien dominiert
   b. „kein Satz der geg. Grammatik“, sonst

Einfacher Bottom-Up-CFG-Parser

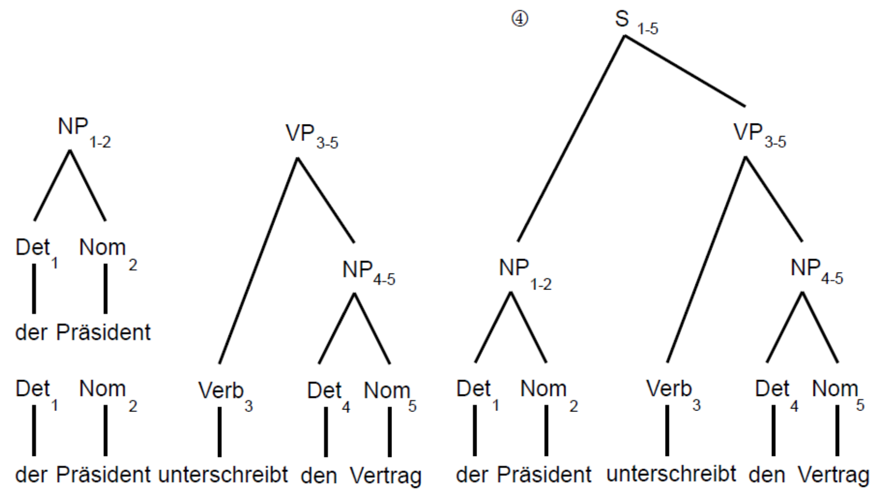
(Algorithmenskizze)



2. Finde iterativ alle Regeln,
   a. deren rechte Seite (bestehend aus *k* Kategoriensymbolen) vollständig mit den Etiketten der Wurzeln einer *k*-elementigen Teilfolge von Graphen (ebenfalls Kategoriensymbole) übereinstimmt, wobei
   b. die Abdeckungsindices der *k*-elementigen Teilfolge von Graphen dicht (also ohne Wort- oder Phrasenlücken) und numerisch aufsteigend geordnet sind.
3. Konstruiere für jede so gefundene *k*-elementige Teilfolge von Graphen einen die gefundenen Graphen enthaltenden neuen Graphen,
   a. dessen Wurzel das komplexe syntaktische Kategoriensymbol der linken Seite der identifizierten Regel als Etikett erhält und die Wurzelknoten der identifizierten *k*-elementigen Teilfolge von Graphen aus der korrespondierenden rechten Seite dieser Regel als direkte Nachfolger des neuen Wurzelknotens hat, wobei die Präzedenzordnung zwischen diesen Teilgraphen sich aus der aufsteigenden Ordnung ihrer Abdeckungsindices ergibt.
   b. der Abdeckungsindex des neu gebildeten Graphen berechnet sich wie folgt:
      (1) die untere Grenze ergibt sich aus der unteren Grenze des ersten Graphen der total geordneten *k*-elementigen Folge von Graphen,
      (2) die obere Grenze ergibt sich aus der oberen Grenze des k-ten Graphen der total geordneten *k*-elementigen Folge von Graphen.
4. Iteriere Schritte (2.) und (3.) solange, bis ein Graph mit dem Startsymbol der Grammatik (S) als Etikett einer Wurzel gebildet ist, die Vorgänger aller Knoten des Graphen ist und den Eingabesatz komplett abgedeckt (d.h. S alle lexikalische Kategorien dominiert und damit einen Abdeckungsindex [1..n] hat. Falls kein entsprechender Graph gebildet werden kann, gehe zu (5.b)
5. <u>Ausgabe</u>:
   a. der Syntaxbaum für den Eingabesatz, falls S alle n lexikalische Kategorien dominiert
   b. „kein Satz der geg. Grammatik", sonst

Einfacher Bottom-Up-CFG-Parser

(Algorithmenskizze)



2. Finde iterativ alle Regeln,
   a. deren rechte Seite (bestehend aus *k* Kategoriensymbolen) vollständig mit den Etiketten der Wurzeln einer *k*-elementigen Teilfolge von Graphen (ebenfalls Kategoriensymbole) übereinstimmt, wobei
   b. die Abdeckungsindices der *k*-elementigen Teilfolge von Graphen dicht (also ohne Wort- oder Phrasenlücken) und numerisch aufsteigend geordnet sind.
3. Konstruiere für jede so gefundene *k*-elementige Teilfolge von Graphen einen die gefundenen Graphen enthaltenden neuen Graphen,
   a. dessen Wurzel das komplexe syntaktische Kategoriensymbol der linken Seite der identifizierten Regel als Etikett erhält und die Wurzelknoten der identifizierten *k*-elementigen Teilfolge von Graphen aus der korrespondierenden rechten Seite dieser Regel als direkte Nachfolger des neuen Wurzelknotens hat, wobei die Präzedenzordnung zwischen diesen Teilgraphen sich aus der aufsteigenden Ordnung ihrer Abdeckungsindices ergibt.
   b. der Abdeckungsindex des neu gebildeten Graphen berechnet sich wie folgt:
      (1) die untere Grenze ergibt sich aus der unteren Grenze des ersten Graphen der total geordneten *k*-elementigen Folge von Graphen,
      (2) die obere Grenze ergibt sich aus der oberen Grenze des k-ten Graphen der total geordneten *k*-elementigen Folge von Graphen.
4. Iteriere Schritte (2.) und (3.) solange, bis ein Graph mit dem Startsymbol der Grammatik (S) als Etikett einer Wurzel gebildet ist, die Vorgänger aller Knoten des Graphen ist und den Eingabesatz komplett abgedeckt (d.h. S alle lexikalische Kategorien dominiert und damit einen Abdeckungsindex [1..n] hat. Falls kein entsprechender Graph gebildet werden kann, gehe zu (5.b)
5. <u>Ausgabe</u>:
   a. der Syntaxbaum für den Eingabesatz, falls S alle n lexikalische Kategorien dominiert
   b. „kein Satz der geg. Grammatik", sonst

Same again but with lexical ambiguity

S $\rightarrow$ NP VP
NP $\rightarrow$ det n
VP $\rightarrow$ v
VP $\rightarrow$ v NP

<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man*, *shot*}
v $\rightarrow$ *shot*

shot can be v or n

3. Top-down with lexical ambiguity

~~the man~~ shot an elephant

$S \rightarrow NP\ VP$

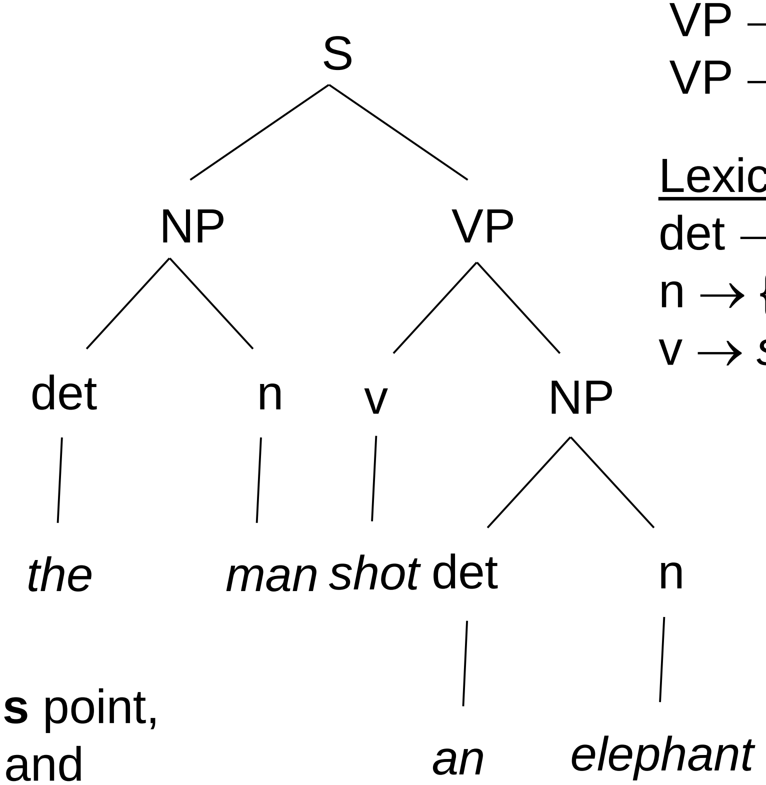
$NP \rightarrow det\ n$

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*, *shot*}
v $\rightarrow$ *shot*

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

4. Bottom-up with lexical ambiguity

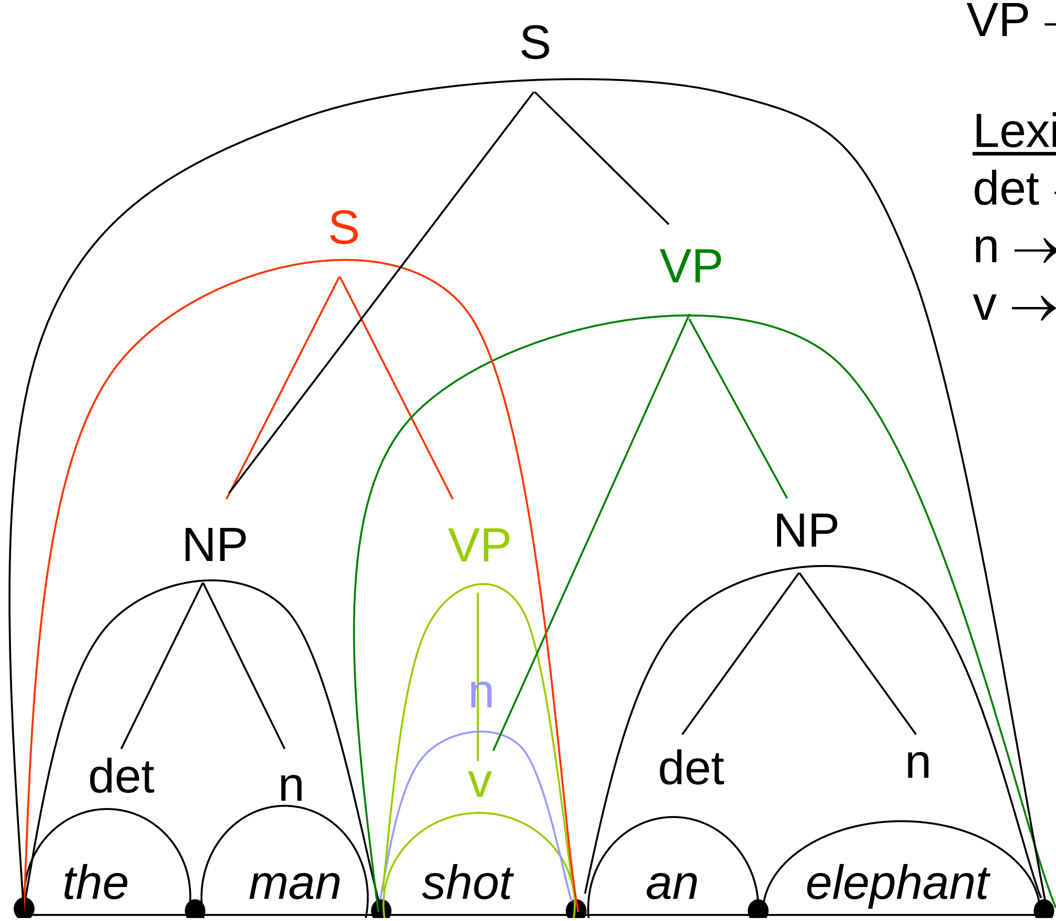
det → {*an*, *the*}
n → {*elephant*, *man*, *shot*}
v → *shot*

NP → det n

VP → v

VP → v NP

S → NP VP



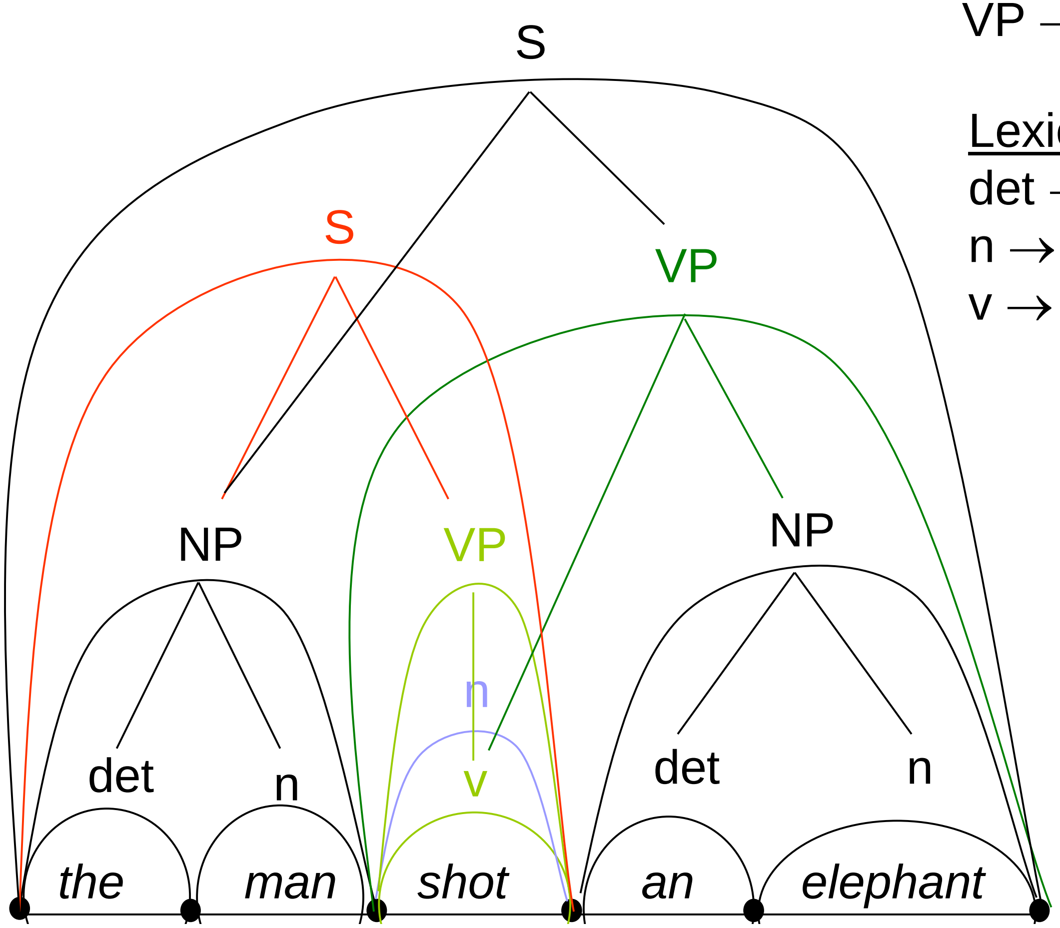
S → NP VP
NP → det n
VP → v
VP → v NP

<u>Lexicon</u>
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



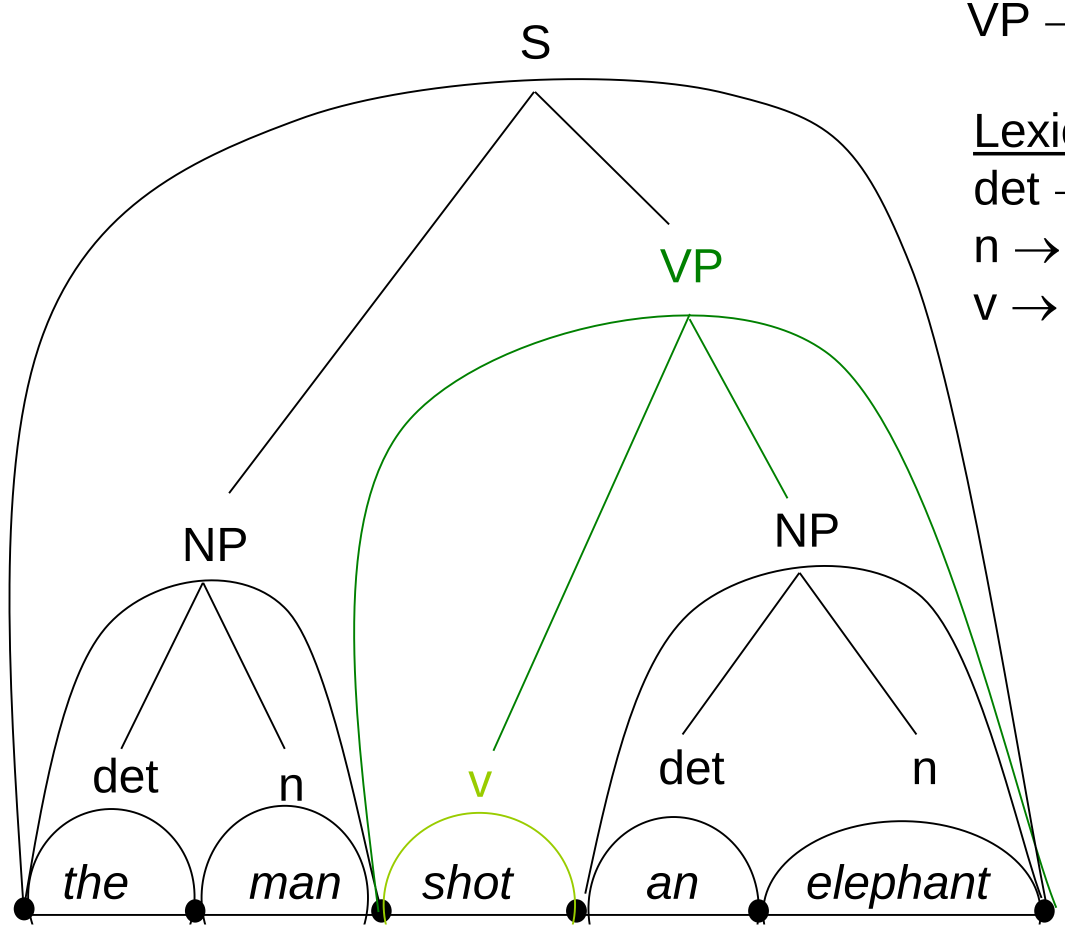
$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$VP \rightarrow v$
$VP \rightarrow v\ NP$

<u>Lexicon</u>

det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man, shot*}
v $\rightarrow$ *shot*

4. Bottom-up with lexical ambiguity

Let's get rid of all the unused arcs



S $\rightarrow$ NP VP
NP $\rightarrow$ det n
VP $\rightarrow$ v
VP $\rightarrow$ v NP

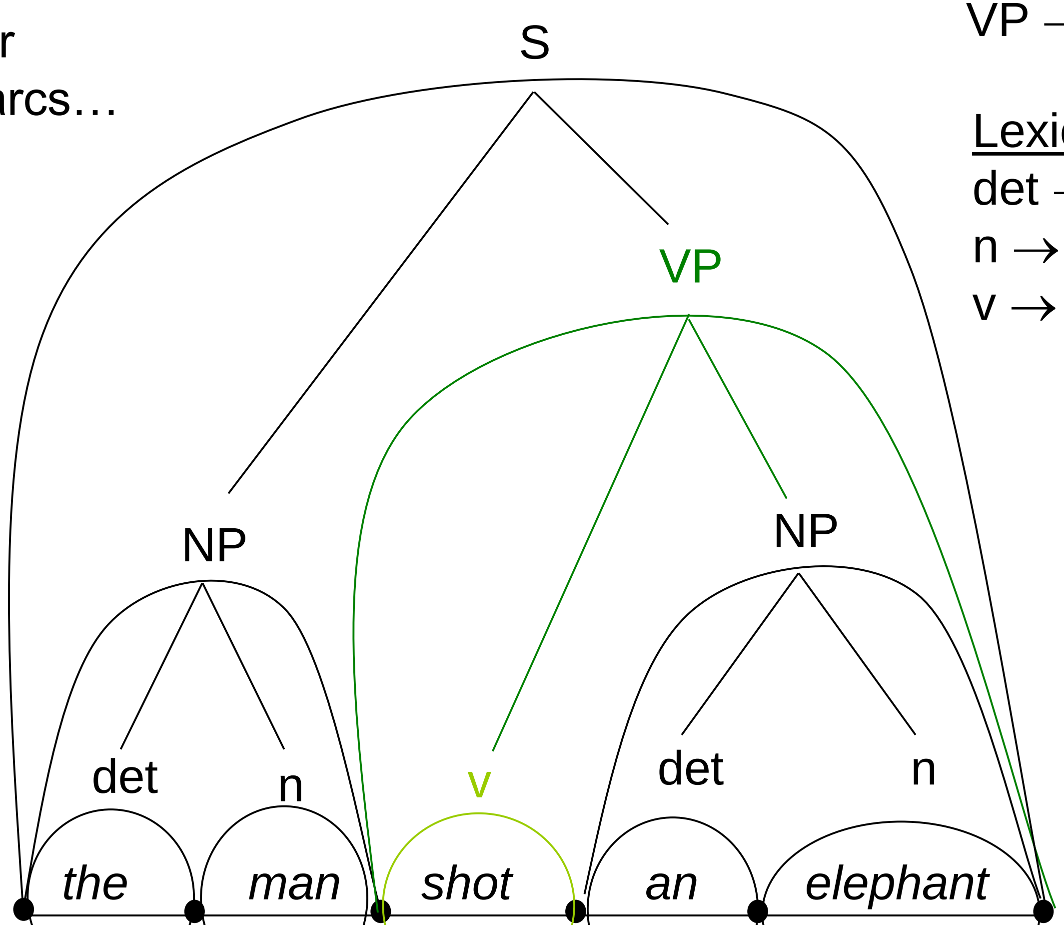
<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man, shot*}
v $\rightarrow$ *shot*

4. Bottom-up with lexical ambiguity

$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$VP \rightarrow v$
$VP \rightarrow v\ NP$

<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man, shot*}
v $\rightarrow$ *shot*

And let's clear away all the arcs…

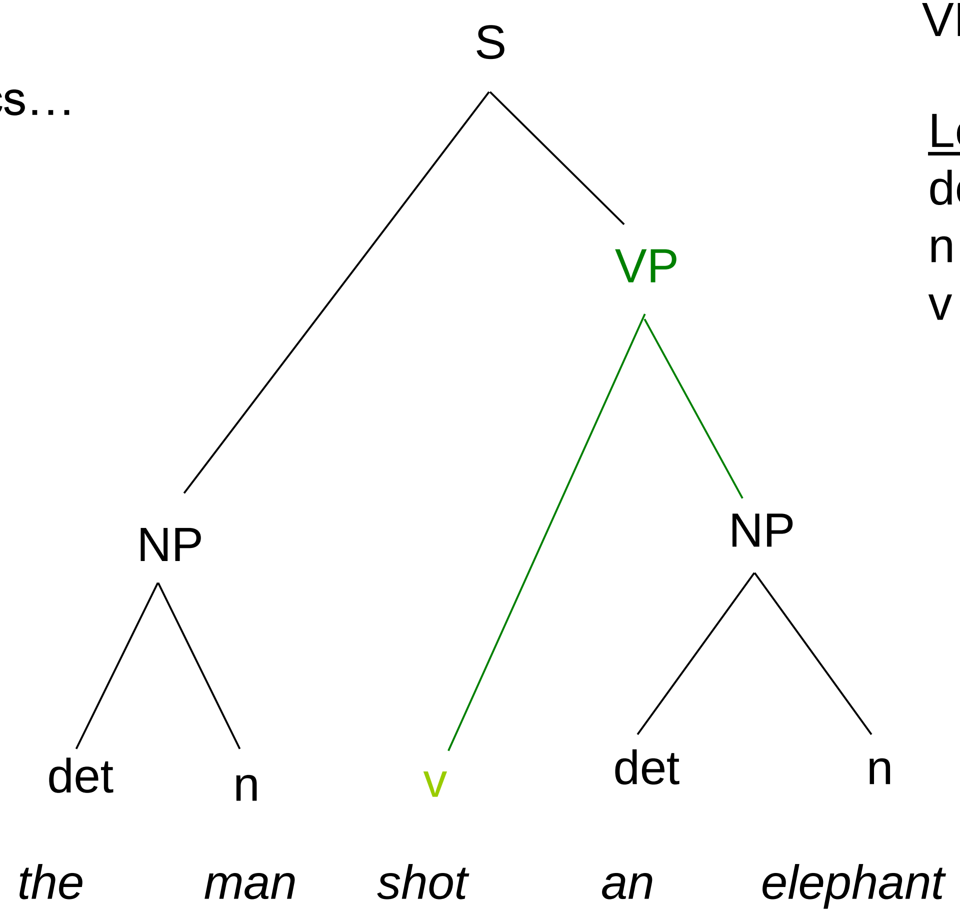


4. Bottom-up with lexical ambiguity

$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$VP \rightarrow v$
$VP \rightarrow v\ NP$

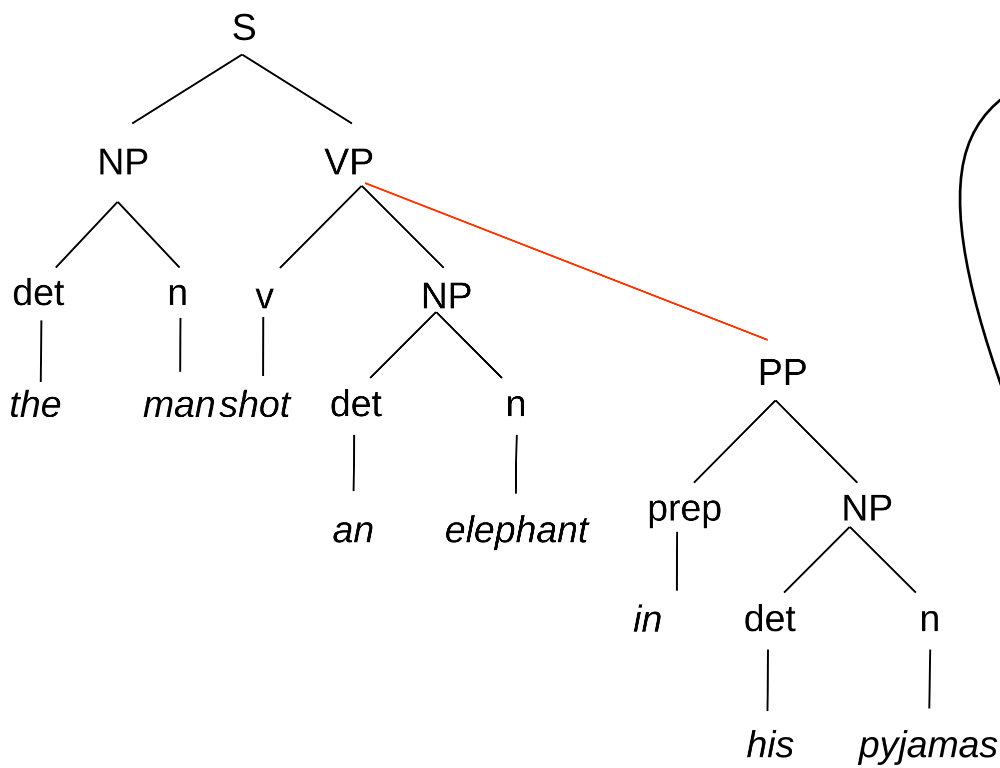
<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*}
n $\rightarrow$ {*elephant*, *man, shot*}
v $\rightarrow$ *shot*

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!]



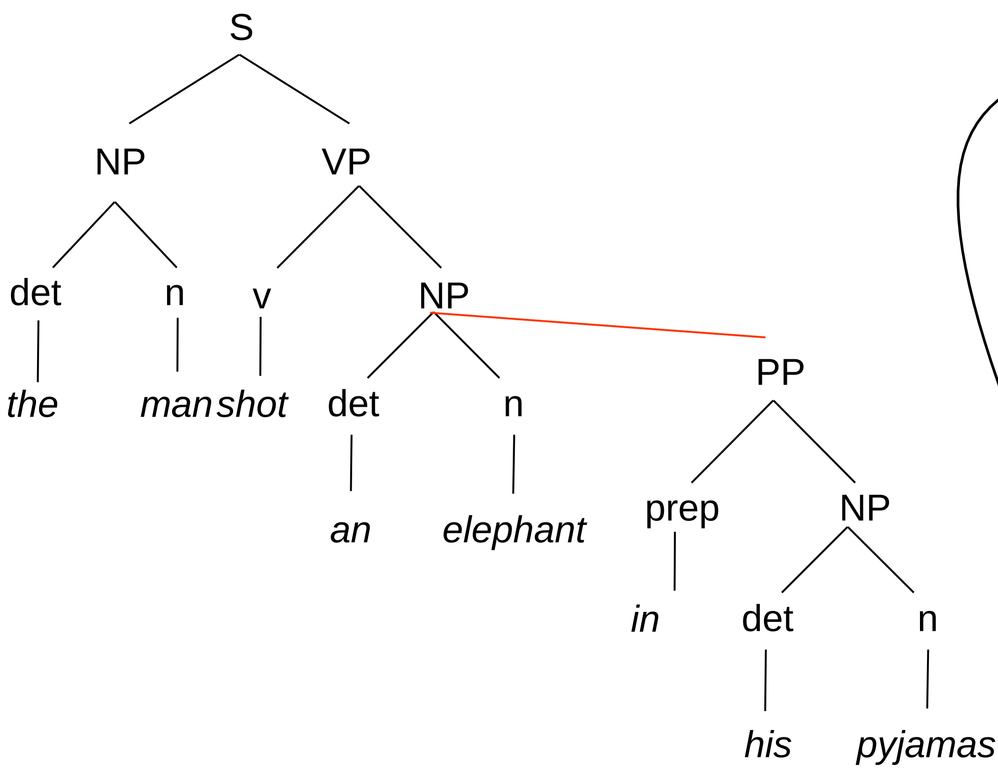
S $\rightarrow$ NP VP
NP $\rightarrow$ det n
NP $\rightarrow$ det n PP
VP $\rightarrow$ v
VP $\rightarrow$ v NP
VP $\rightarrow$ v NP PP
PP $\rightarrow$ prep NP

<u>Lexicon</u>
det $\rightarrow$ {*an*, *the*, *his*}
n $\rightarrow$ {*elephant*, *man*, *shot*, *pyjamas*}
v $\rightarrow$ *shot*
prep $\rightarrow$ *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

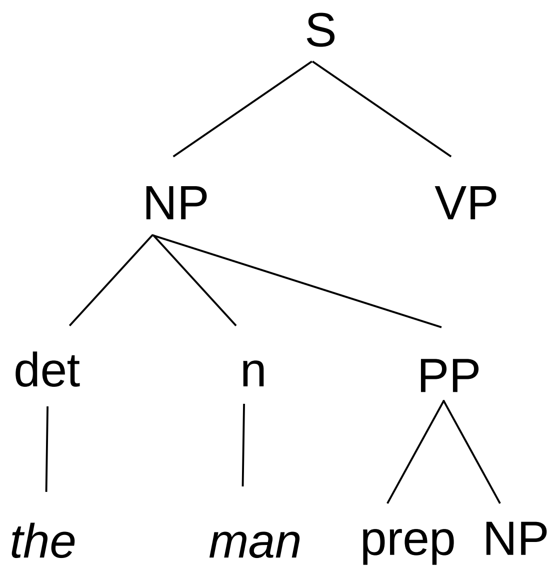
<u>Lexicon</u>
det → {*an*, *the*, *his*}
n → {*elephant*, *man*, *shot*, *pyjamas*}
v → *shot*
prep → in

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

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

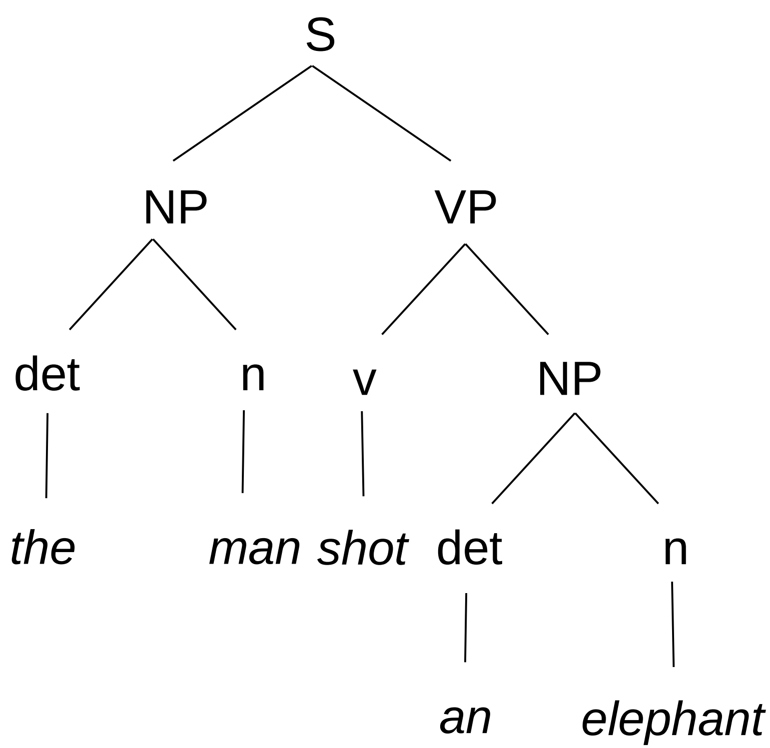
At this point, depending on our strategy (breadth-first vs. depth-first) we may consider the NP complete and look for the VP,
or we may try the second NP rule.
Let's see what happens in the latter case.

5. Top-down 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

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*}

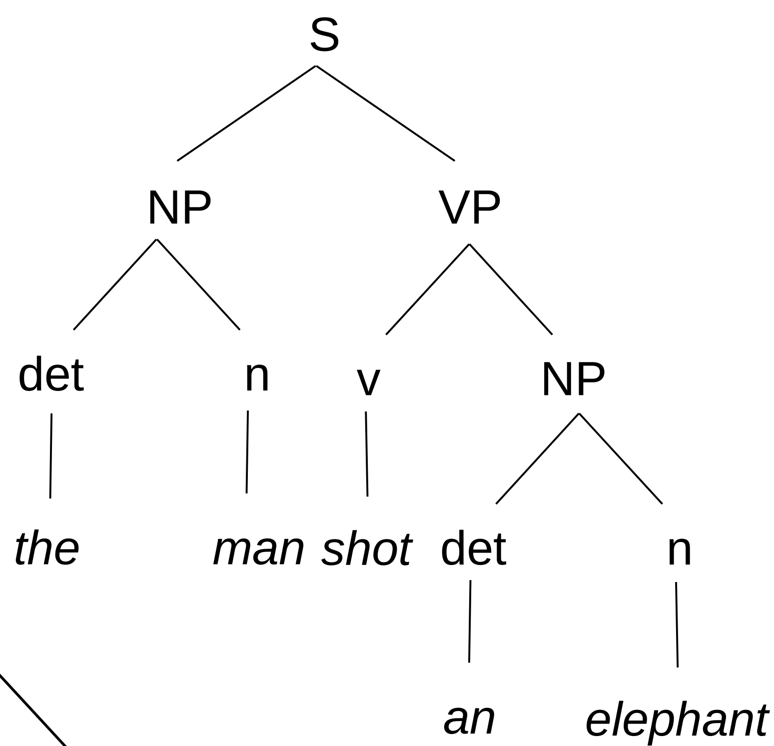


As before, the first VP rule works,
But does not account for all the input.
Similarly, if we try the second VP rule, and the first NP rule …

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



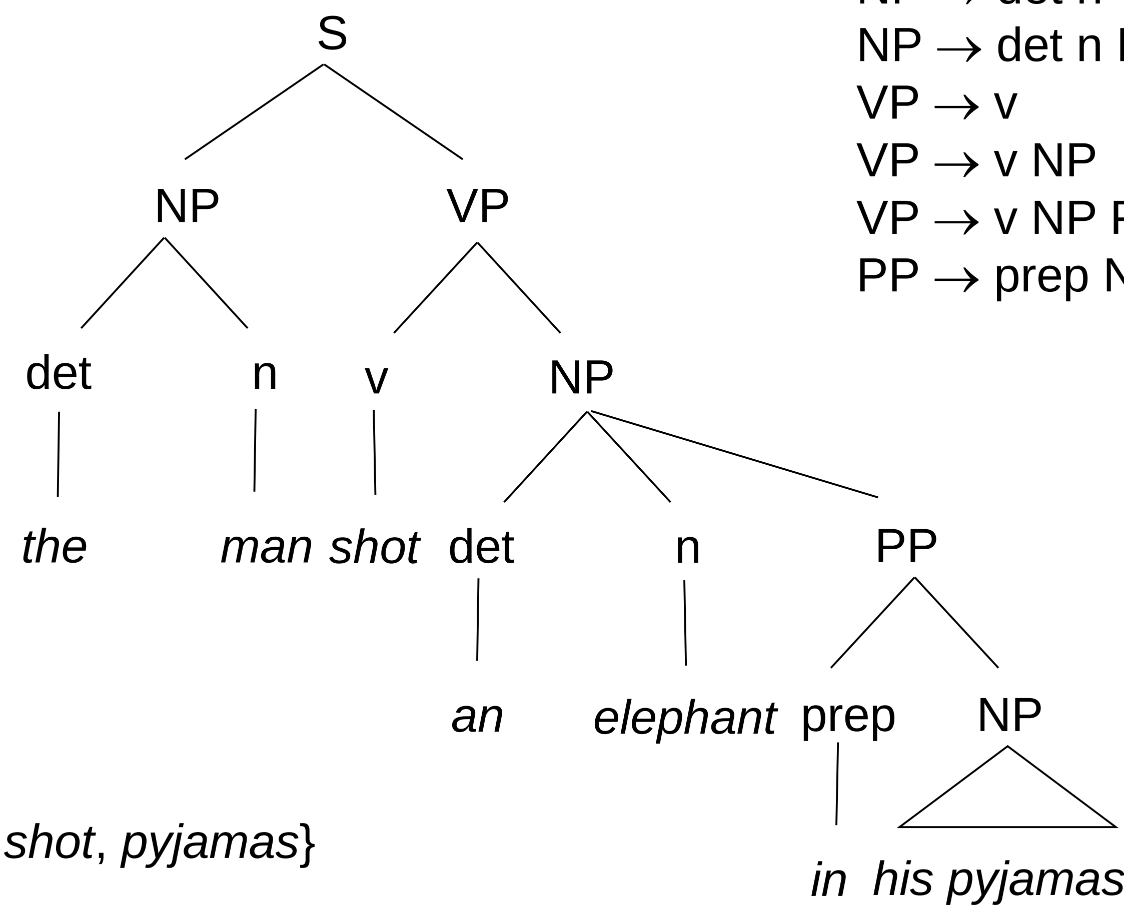
So what do we try next?
This?
Or this?

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 (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*

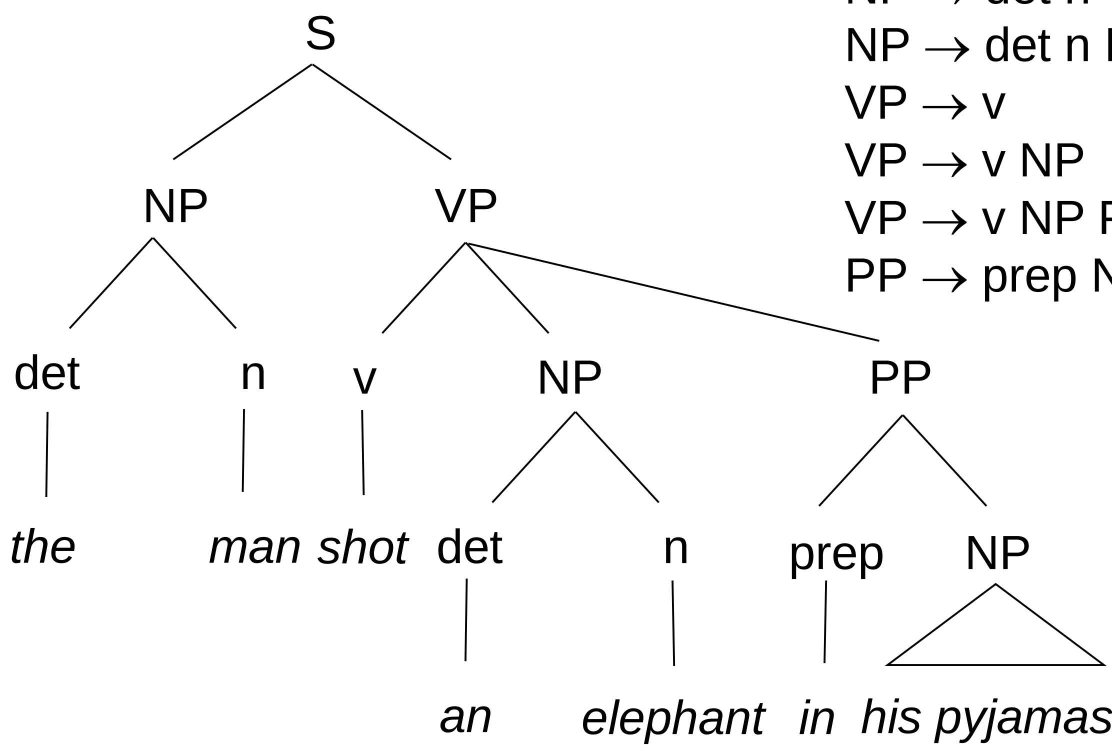


S → NP VP
NP → det n
NP → det n PP
VP → v
VP → v NP
VP → v NP PP
PP → prep NP

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*

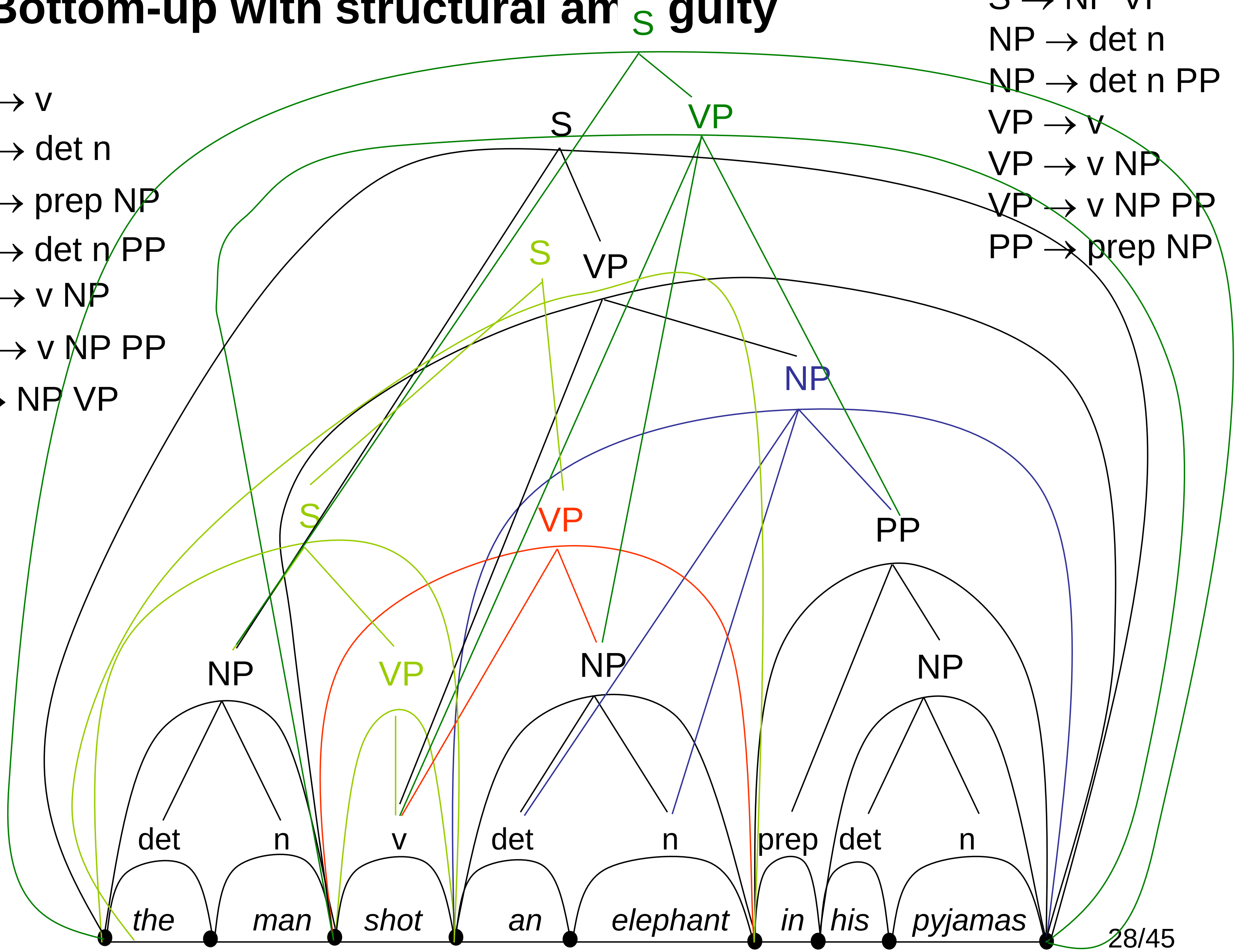


S → NP VP
NP → det n
NP → det n PP
VP → v
VP → v NP
VP → v NP PP
PP → prep NP

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

$S \rightarrow NP\ VP$

$NP \rightarrow det\ n$

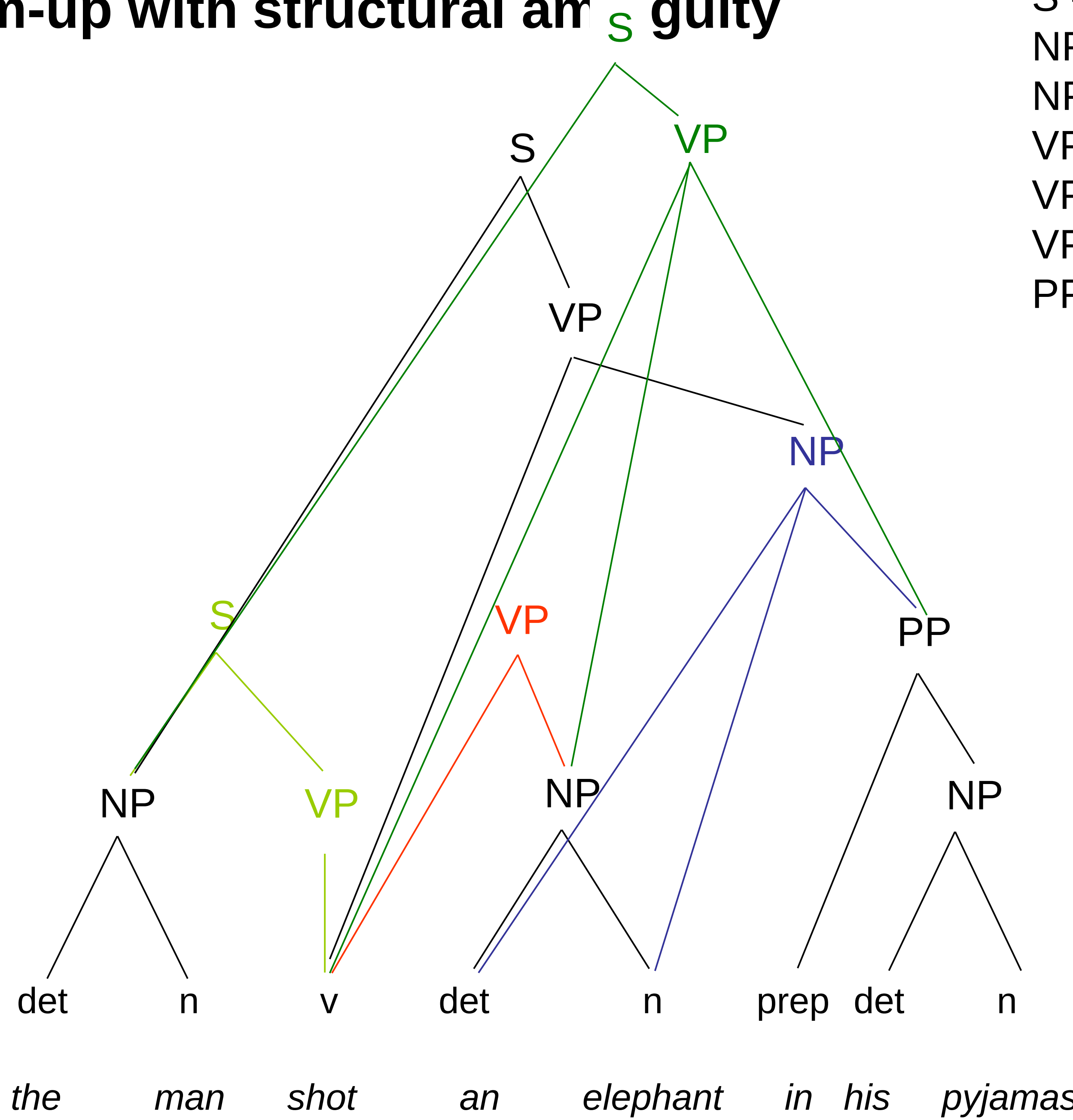
$NP \rightarrow det\ n\ PP$

$VP \rightarrow v$

$VP \rightarrow v\ NP$

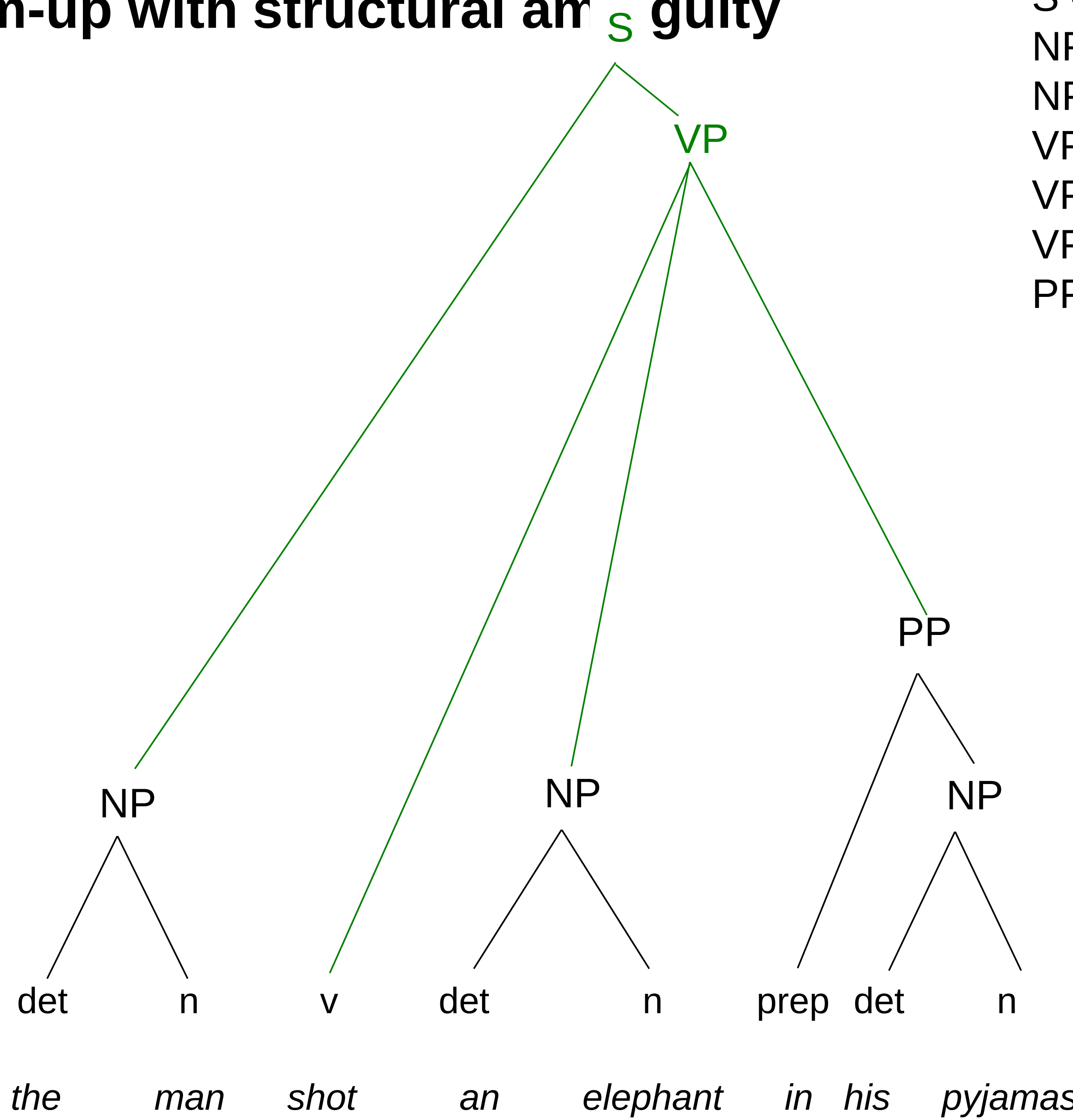
$VP \rightarrow v\ NP\ PP$

$PP \rightarrow prep\ NP$



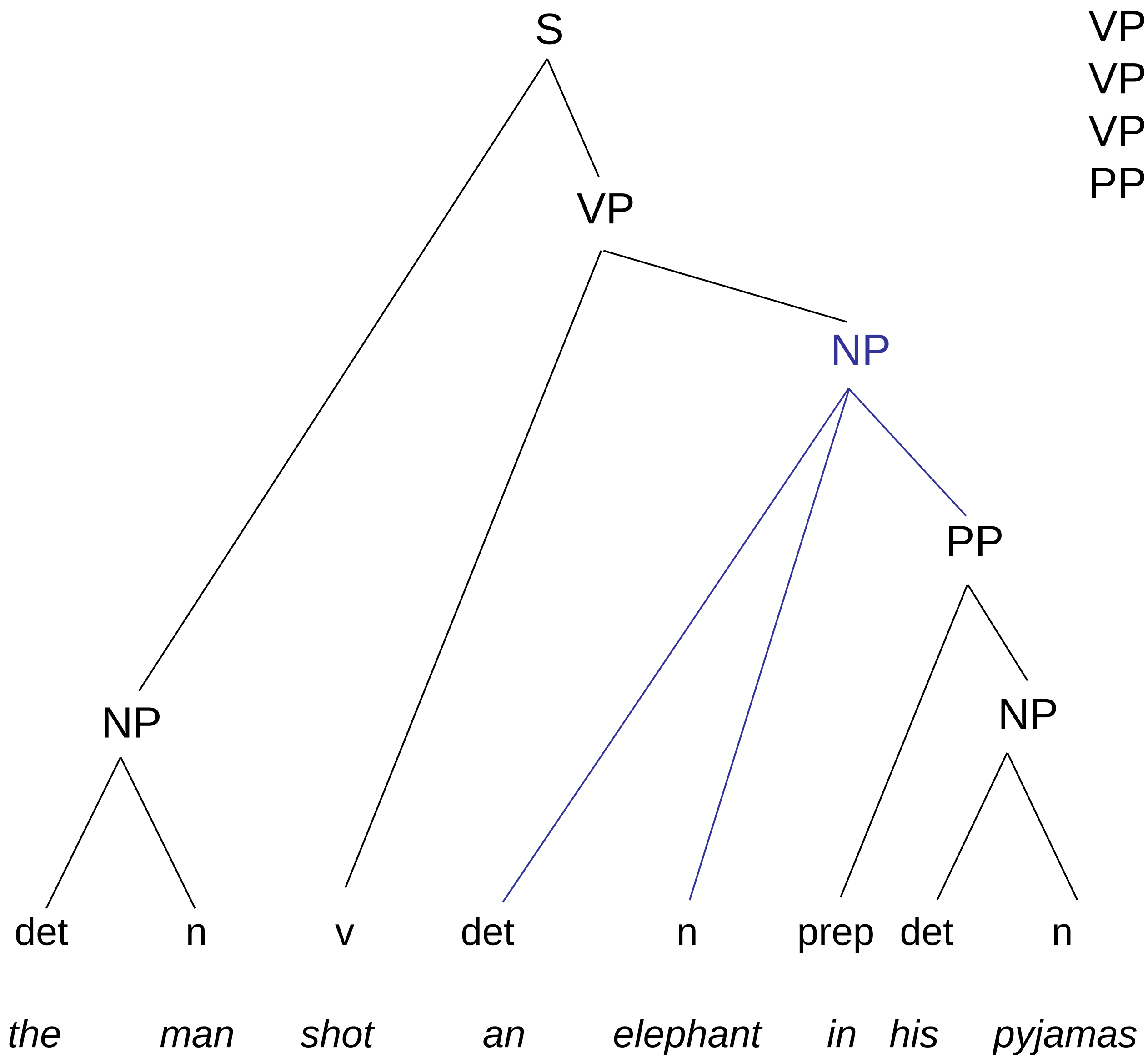
6. Bottom-up with structural ambiguity

$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$NP \rightarrow det\ n\ PP$
$VP \rightarrow v$
$VP \rightarrow v\ NP$
$VP \rightarrow v\ NP\ PP$
$PP \rightarrow prep\ NP$



6. Bottom-up with structural ambiguity

$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$NP \rightarrow det\ n\ PP$
$VP \rightarrow v$
$VP \rightarrow v\ NP$
$VP \rightarrow v\ NP\ PP$
$PP \rightarrow prep\ NP$



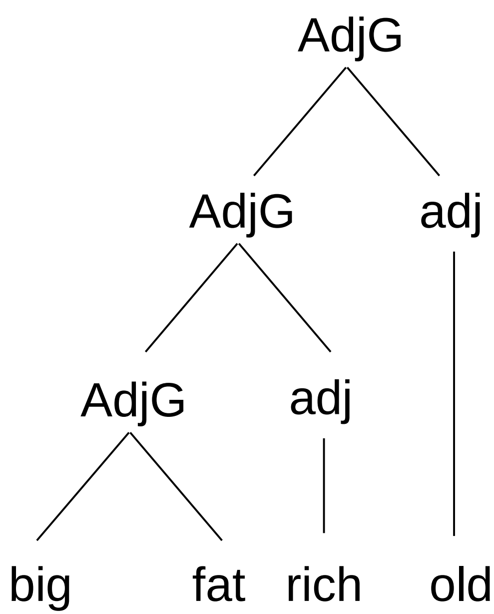
Recursive rules

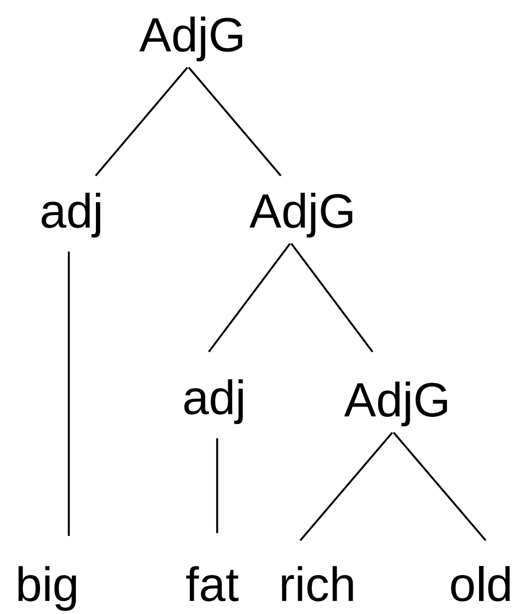
- “Recursive” rules call themselves
- We already have some recursive rule pairs:
  - NP $\rightarrow$ det n PP
  - PP $\rightarrow$ prep NP
- Rules can be immediately recursive
  - AdjG $\rightarrow$ adj AdjG
    - (*the*) *big fat ugly* (*man*)

Recursive rules

<u>Left recursive</u>
AdjG → AdjG adj
AdjG → adj adj

<u>Right recursive</u>
AdjG → adj AdjG
AdjG → adj adj



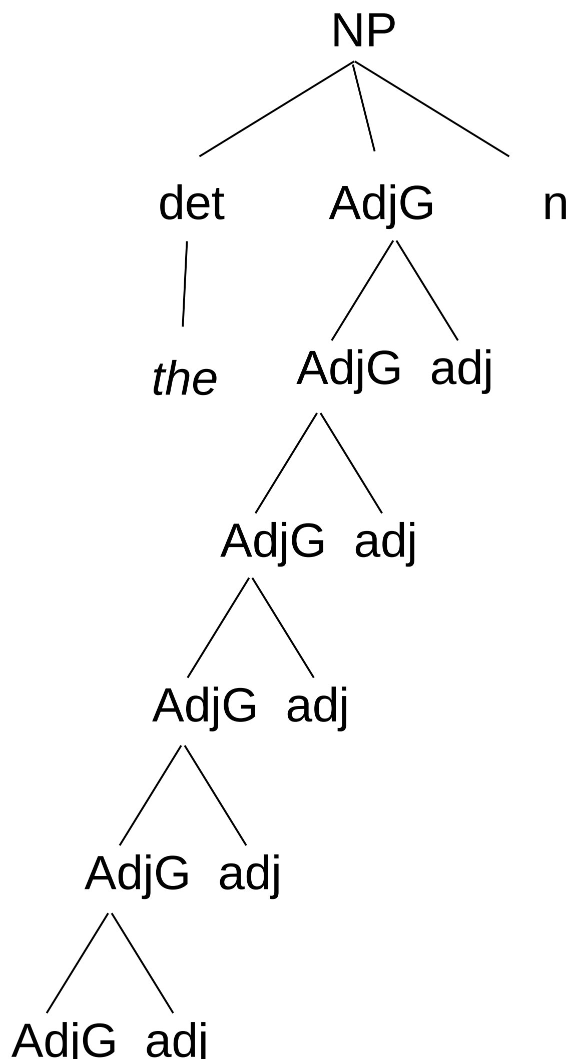


7. Top-down with left recursion

NP $\rightarrow$ det n
NP $\rightarrow$ det AdjG n
AdjG $\rightarrow$ AdjG adj
AdjG $\rightarrow$ adj

~~*the*~~ *big fat rich old man*

~~NP $\rightarrow$ det n~~
NP $\rightarrow$ det AdjG n
AdjG $\rightarrow$ AdjG adj
AdjG $\rightarrow$ adj



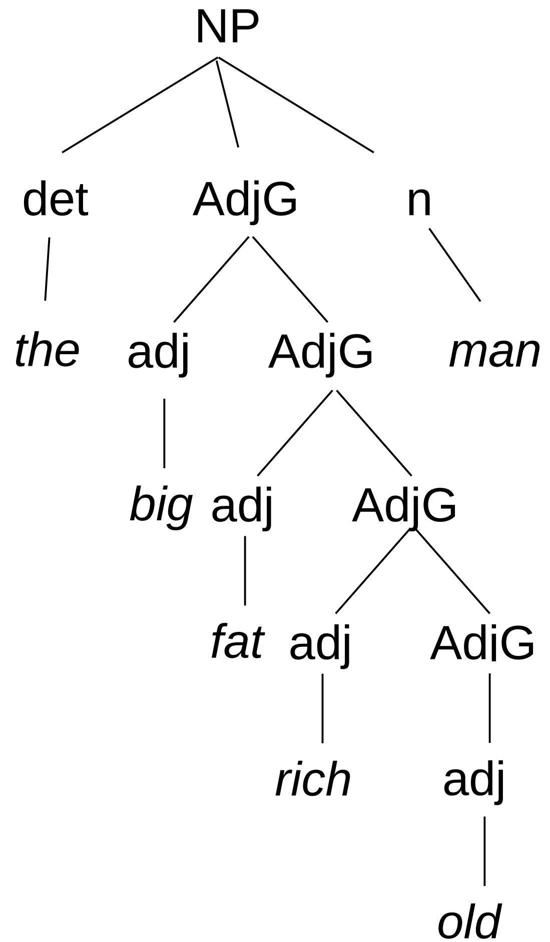
You can't have left-recursive rules with a top-down parser, even if the non-recursive rule is first

7. Top-down with right recursion

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

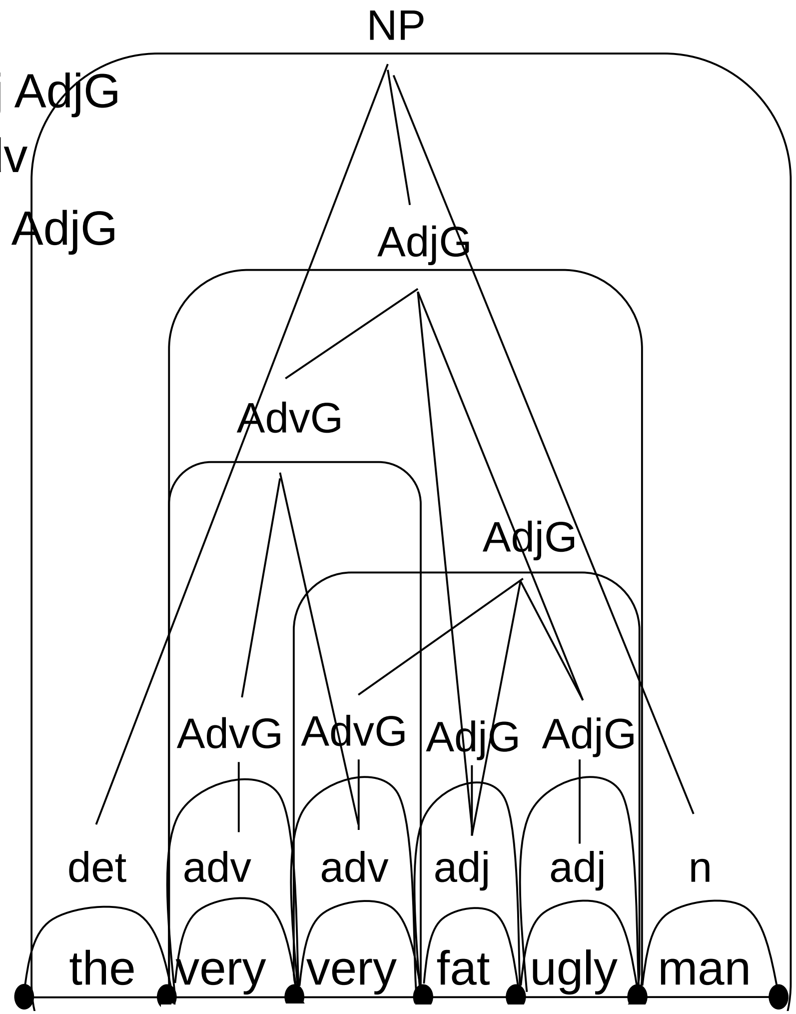
~~*the big fat rich old man*~~

~~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 $\rightarrow$ adj

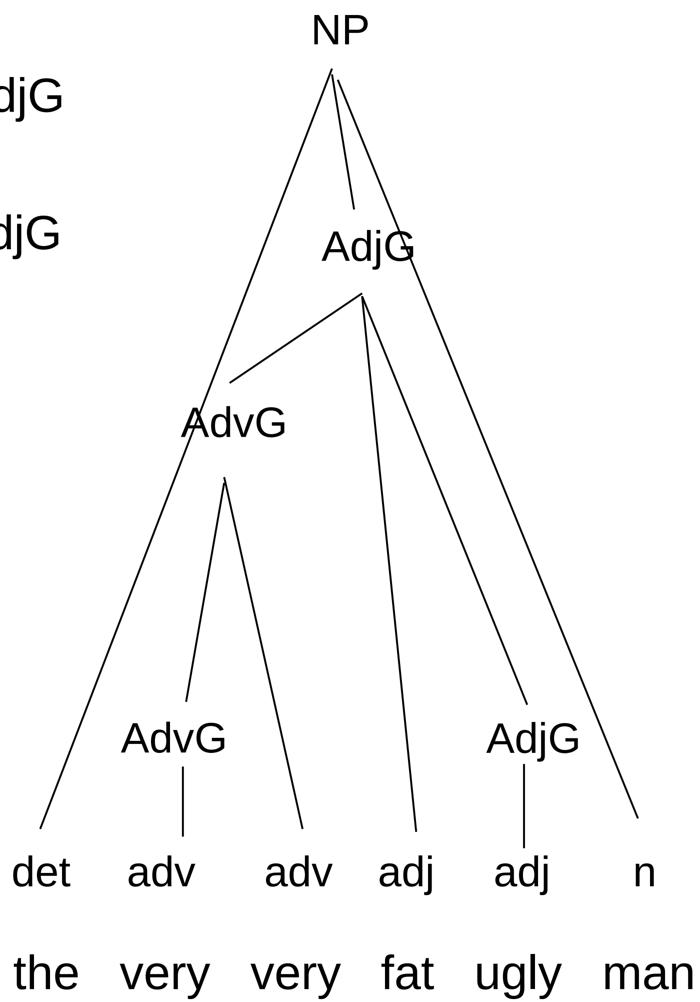
AdvG $\rightarrow$ adv

AdjG $\rightarrow$ AdvG adj AdjG

AdvG $\rightarrow$ AdvG adv

AdjG $\rightarrow$ AdvG adj AdjG

NP $\rightarrow$ det AdjG n



NP $\rightarrow$ det n
NP $\rightarrow$ det AdjG n
AdjG $\rightarrow$ AdvG adj AdjG
AdjG $\rightarrow$ adj
AdvG $\rightarrow$ AdvG adv
AdvG $\rightarrow$ adv

AdjG rule is right recursive,
AdvG rule is left recursive

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

- 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?

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?

9. Left-corner with simple grammar

the man shot an elephant

NP → det n

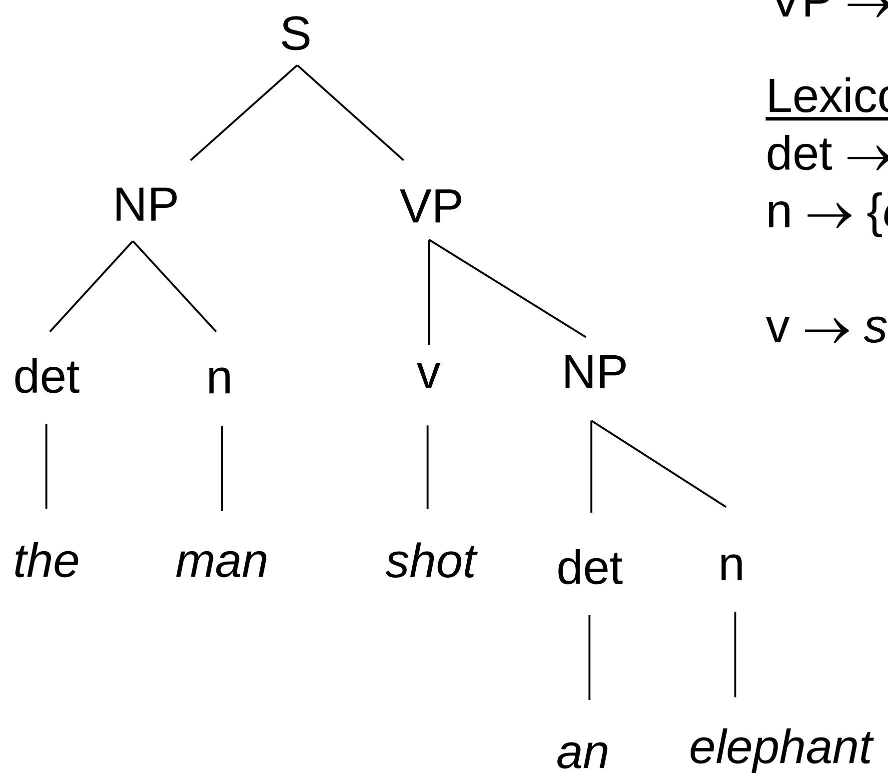
S → NP VP

VP → v

but text not all accounted for, so try

VP → v NP

NP → det n



S → NP VP
NP → det n
VP → v
VP → v NP

<u>Lexicon</u>
det → {*an*, *the*, *his*}
n → {*elephant*, *man*, *shot*, *pyjamas*}
v → *shot*

Left-Corner-Parsing – Basic Idea

- Left-corner combines bottom-up and top-down strategies in the following sense.
- Given a rule: $k_0 \longrightarrow k_1 k_2 \ldots k_n$
- Normal bottom-up: all k1 to kn must be recognized before applying the rule
- Left-corner: it suffices that k_1 is recognized
- k2 to kn and the dominating nodes of k_1 are predicted top-down

Quelle: Mike Rosner, Dec 2011 [Lecture 17: Hybrid Algorithms: Left Corner Parsing]
http://staff.um.edu.mt/mros1/csa3202/pdf/left_corner.pdf

Left-Corner Relation– Formally

- We define the relation $\angle$ between nonterminals such that $B \angle A$ if and only if there is a rule $A \rightarrow B\alpha$, where α denotes some sequence of grammar symbols.
- The transitive and reflexive closure of $\angle$ is denoted by $\angle^*$, which is called the left-corner relation.
- Informally, we have that $B \angle^* A$ if and only if it is possible to have a spine in some parse tree in which B occurs below A (or B = A).
- Possible left corners of all non-terminal categories can be determined in advance and placed in a table.

Quelle: Mike Rosner, Dec 2011 [Lecture 17: Hybrid Algorithms: Left Corner Parsing] http://staff.um.edu.mt/mros1/csa3202/pdf/left_corner.pdf

Left-Corner Relation– Table

$S \rightarrow NP\ VP$
$NP \rightarrow det\ n$
$VP \rightarrow v$
$VP \rightarrow v\ NP$

Lexicon
det $\rightarrow$ {*an*, *the*, *his*}
n $\rightarrow$ {*elephant*, *man*,
shot, *pyjamas*}
v $\rightarrow$ *shot*

Cat	Left Corners
S	NP, det, *an, the, his*
NP	det, *an, the, his*
VP	v, *shot*

Left-Corner-Parsing – Algorithm

To parse a constituent of type C:

1. Accept a word W from input and determine K, its category.
2. Complete C:
   - If K=C, exit with success; otherwise
   - Find a constituent whose expansion begins with K. Call that CC. For instance, if K= **det** , CC could be **NP** since we have rule **NP → det n**
   - Recursively left-corner parse all the remaining elements of the expansion of CC (in this case, **n**).
   - Put CC in place of K, and return to step 2

Quelle: Mike Rosner, Dec 2011 [Lecture 17: Hybrid Algorithms: Left Corner Parsing]
http://staff.um.edu.mt/mros1/csa3202/pdf/left_corner.pdf