PARSING

Ing. R. Tedesco. PhD, AA 20-21

(mostly from: Speech and Language Processing - Jurafsky and Martin)

Today

- Parsing with CFGs
- Ambiguity

Parsing

- Parsing with CFGs refers to the task of assigning *proper* trees to input strings
- *Proper* here means a tree that covers all and only the elements of the input and has an *S* at the top
- First step: find all the admissible trees
- Second step - eliminate ambiguities: select the correct tree

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices
- We'll see some basic (meaning bad) methods

For Now

- Assume…
  - You have all the words already in some buffer
  - The input isn't POS tagged
  - We won't worry about morphological analysis
  - All the words are known

  - These are all problematic in various ways, and would have to be addressed in real applications.

Top-Down Search

- Since we're trying to find trees rooted with an *S* (Sentences), why not start with the rules that give us an *S*.
- Then we can work our way down from there to the words.

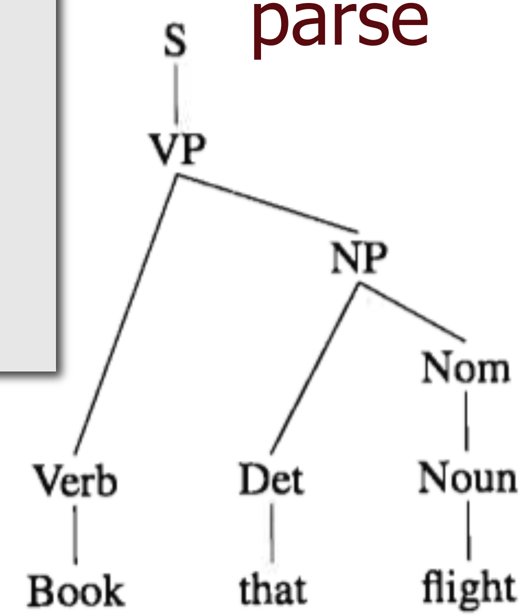
Example

- E.g.: “book that flight”

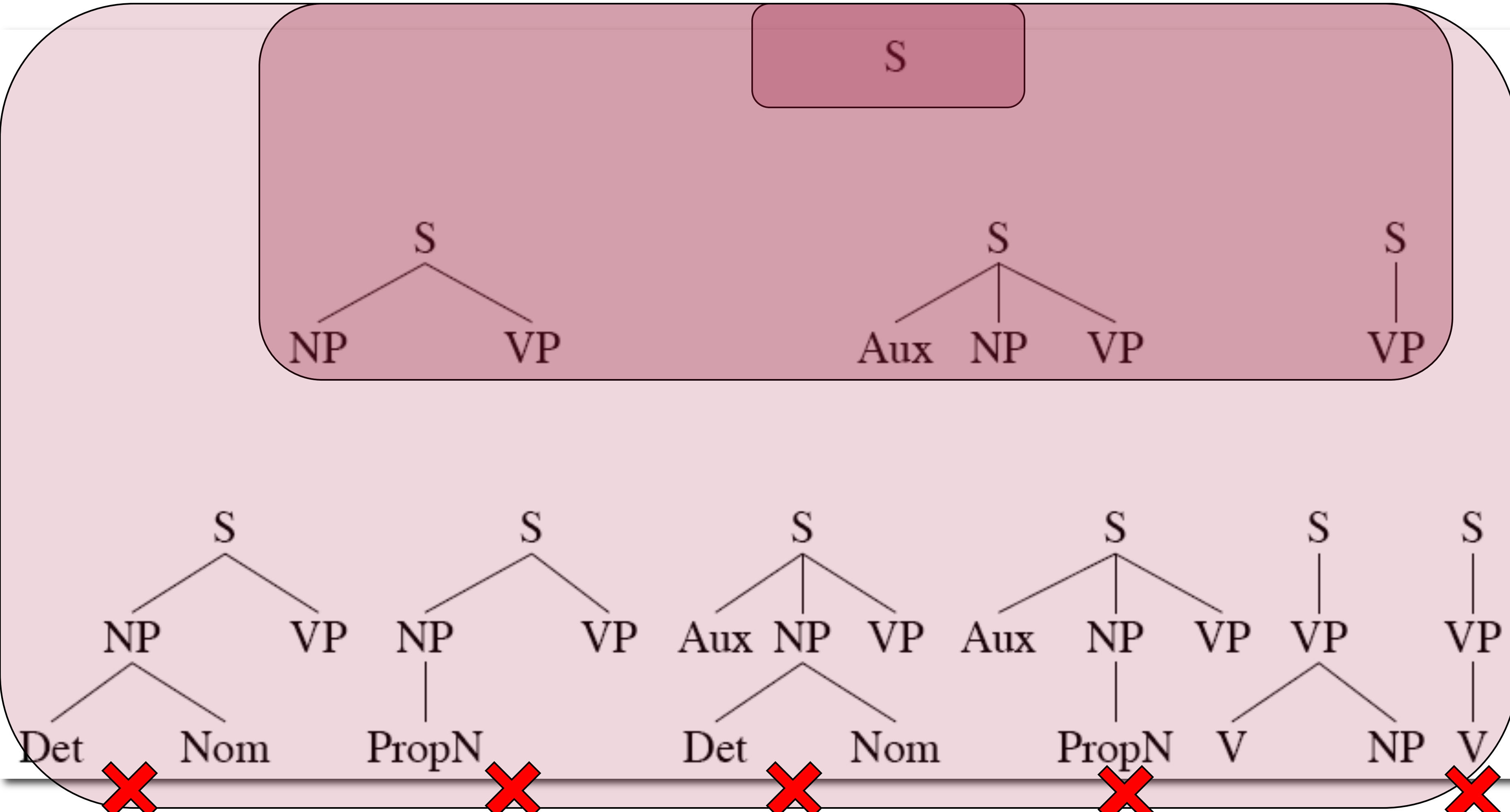
Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

- Grammar

- Correct parse

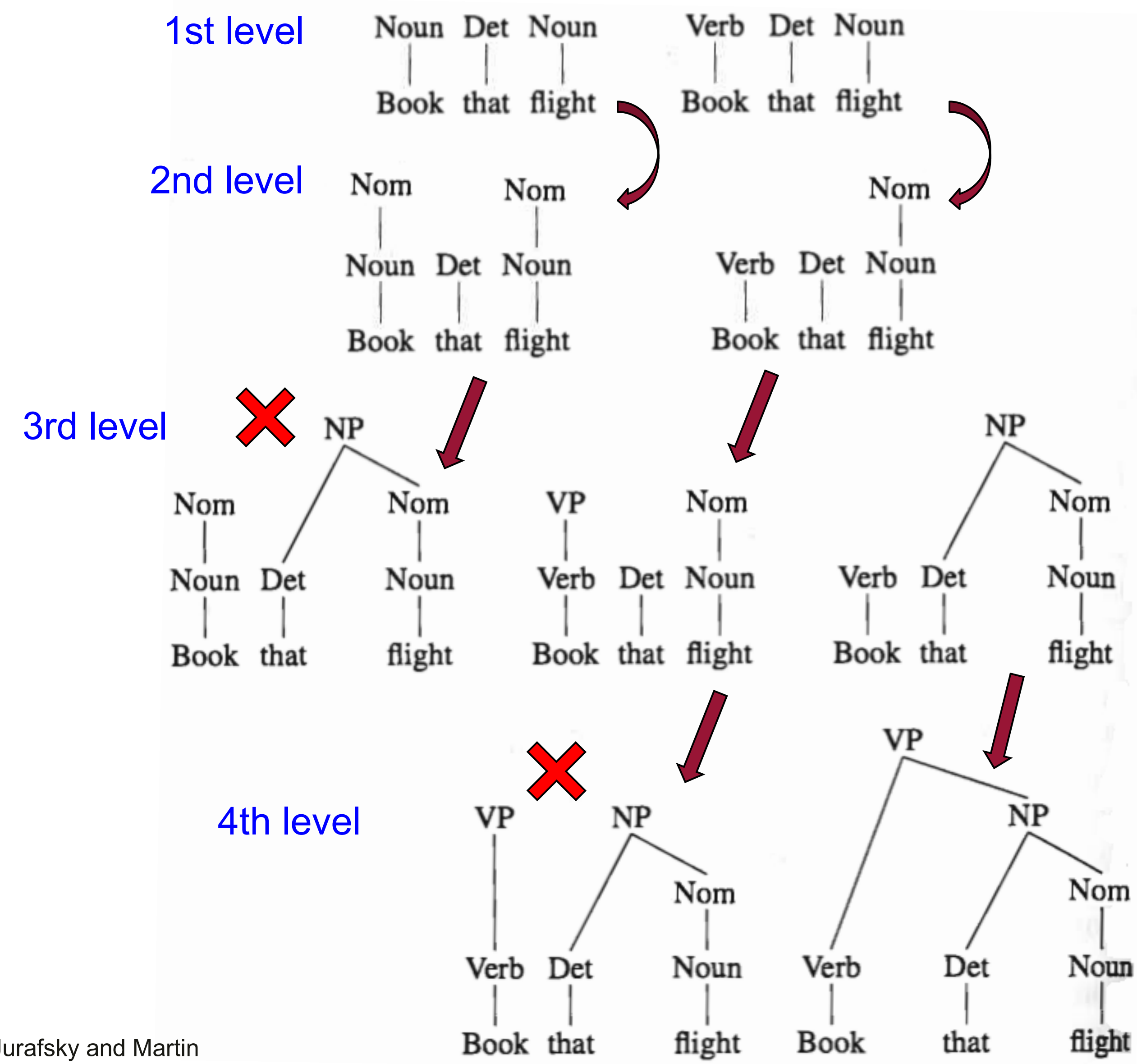


Top Down Space (partial)



Bottom-Up Parsing

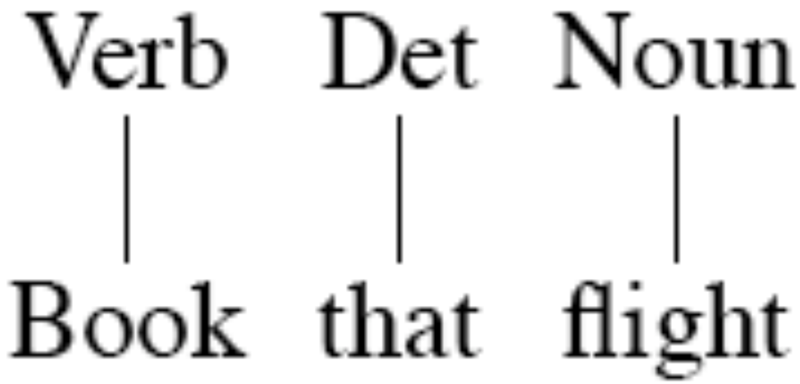
- But, we also want trees that cover the input words.
- So we might also start with trees that link up with the words in the right way.
- Then work your way up from there to larger and larger trees.



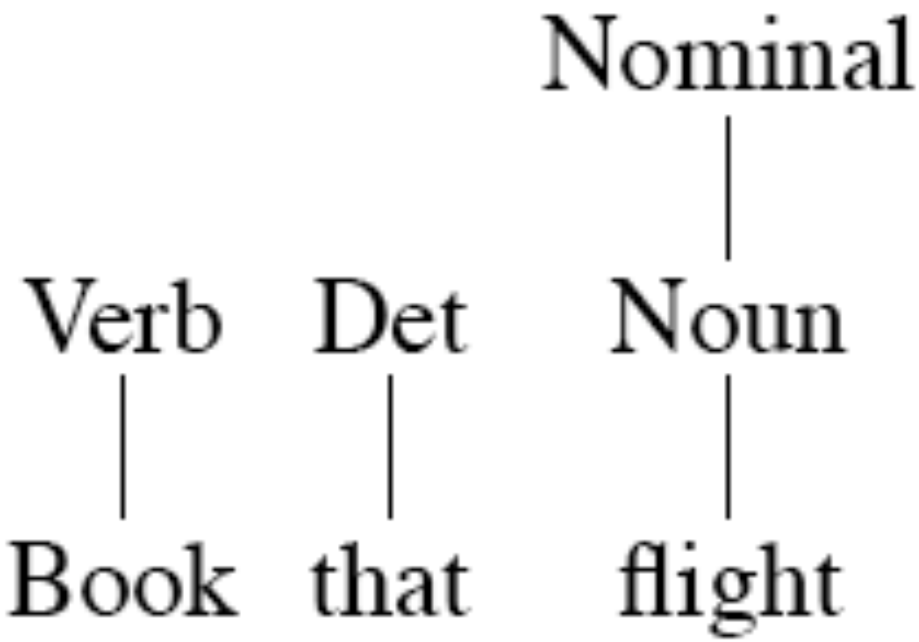
Bottom-Up Search

Book that flight

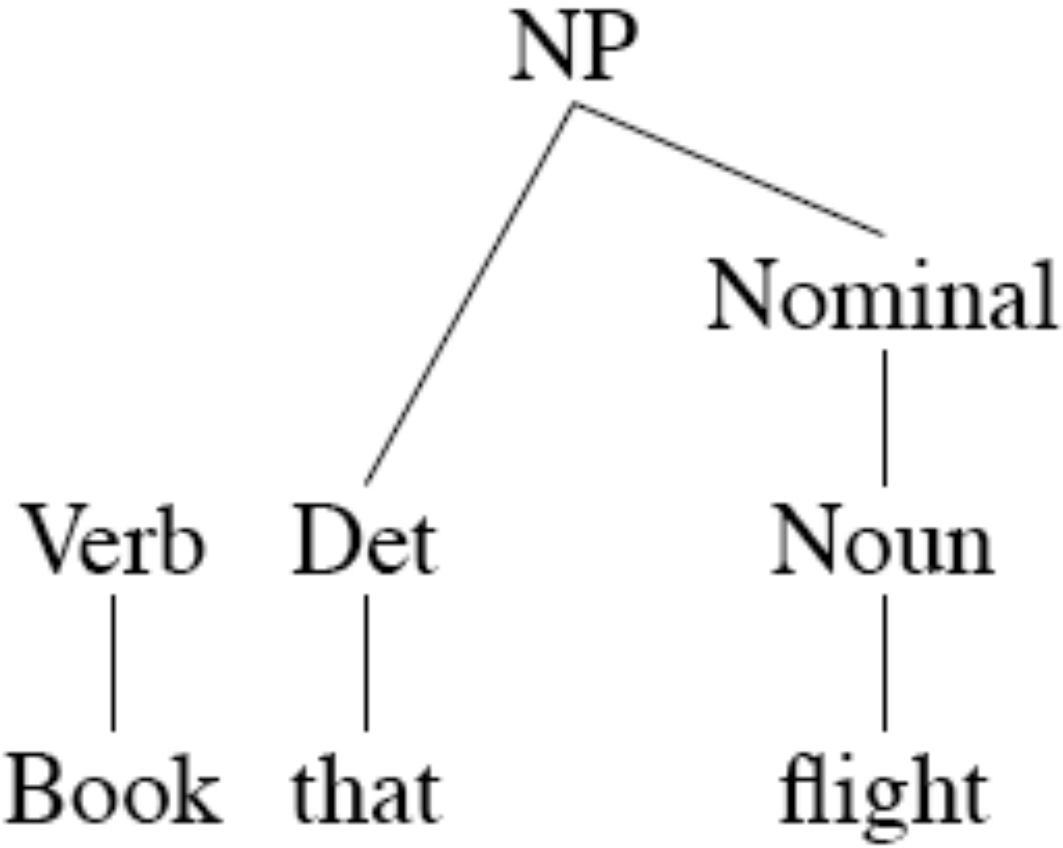
Bottom-Up Search



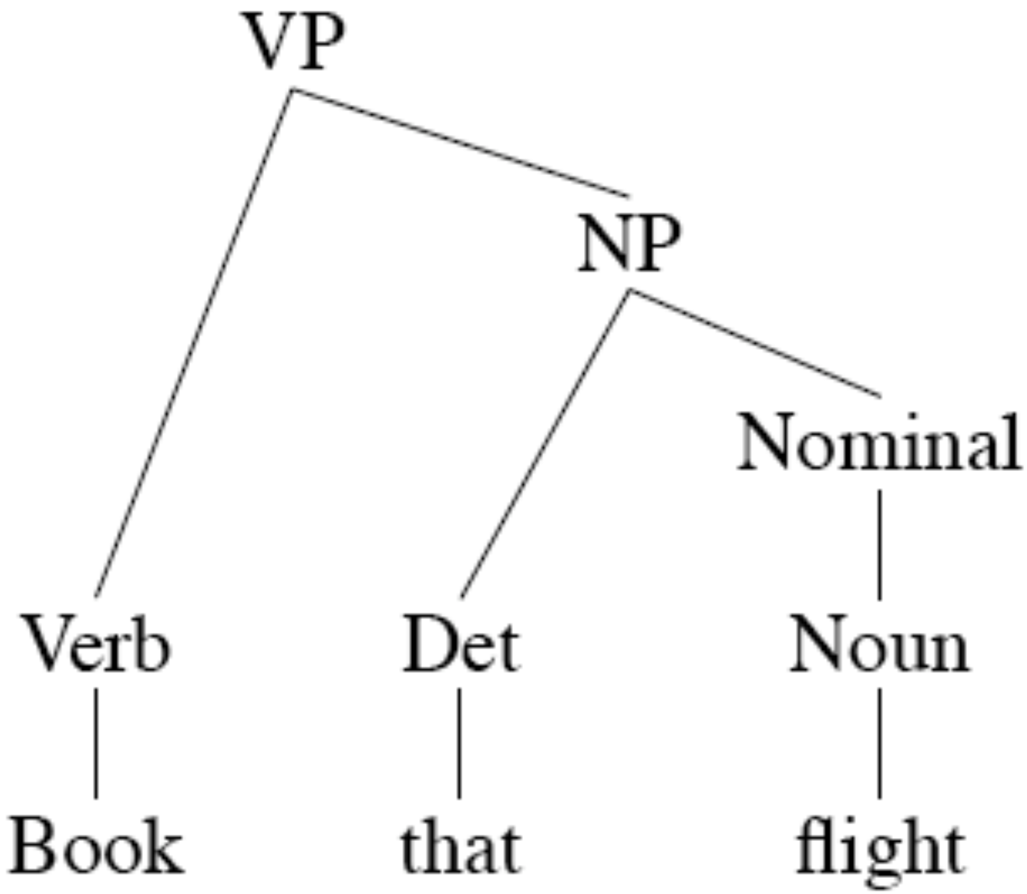
Bottom-Up Search



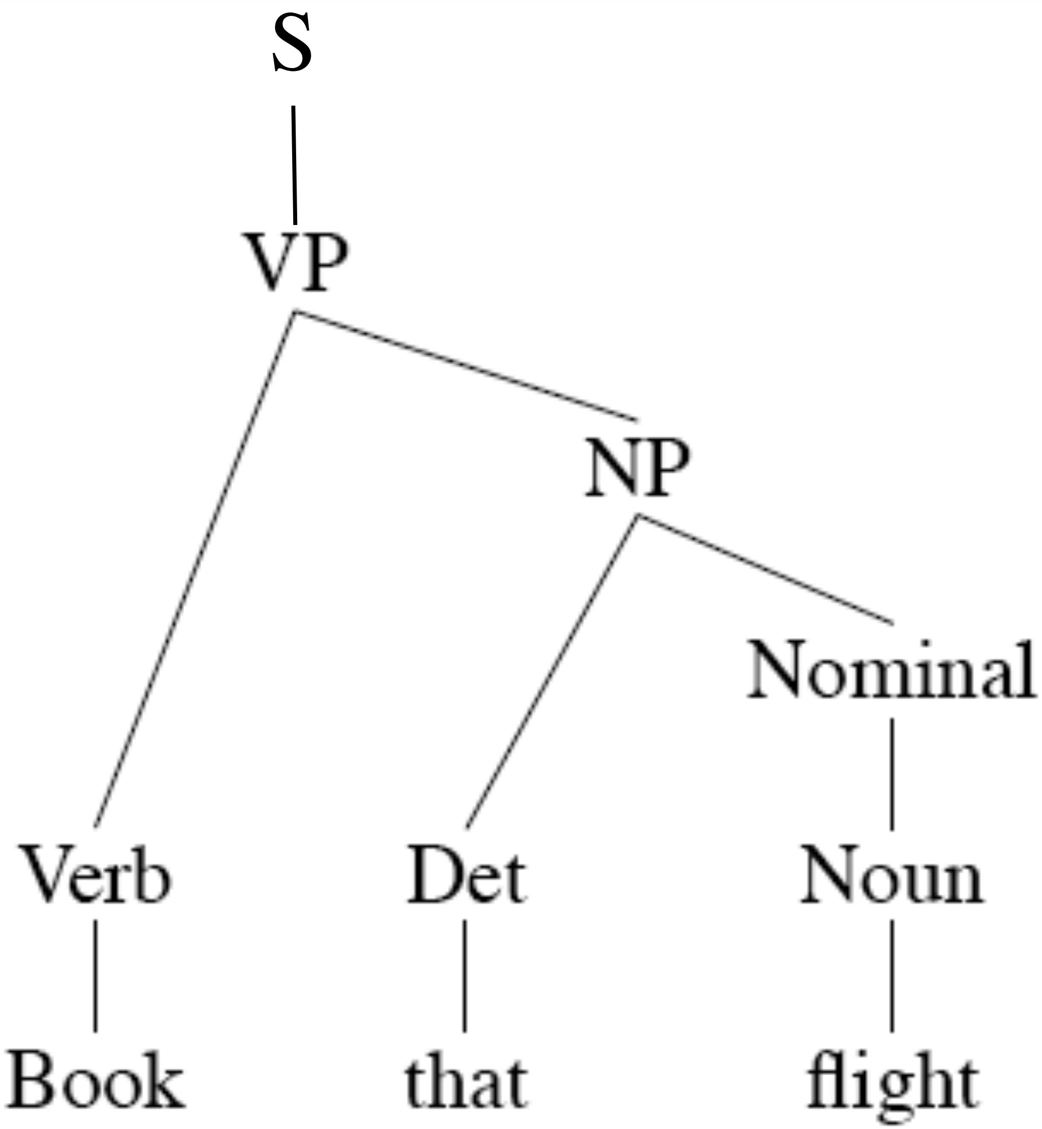
Bottom-Up Search



Bottom-Up Search



Bottom-Up Search



Top-Down and Bottom-Up

- Top-down
  - Only returns trees consistent with the grammar (they all start from *S*)
  - But also generates trees that are not consistent with any of the words (they do not cover all and only the sentence words)
- Bottom-up
  - Only forms trees consistent with the words (they cover all and only the sentence words)
  - But also generates subtrees that can't reach *S*

Control

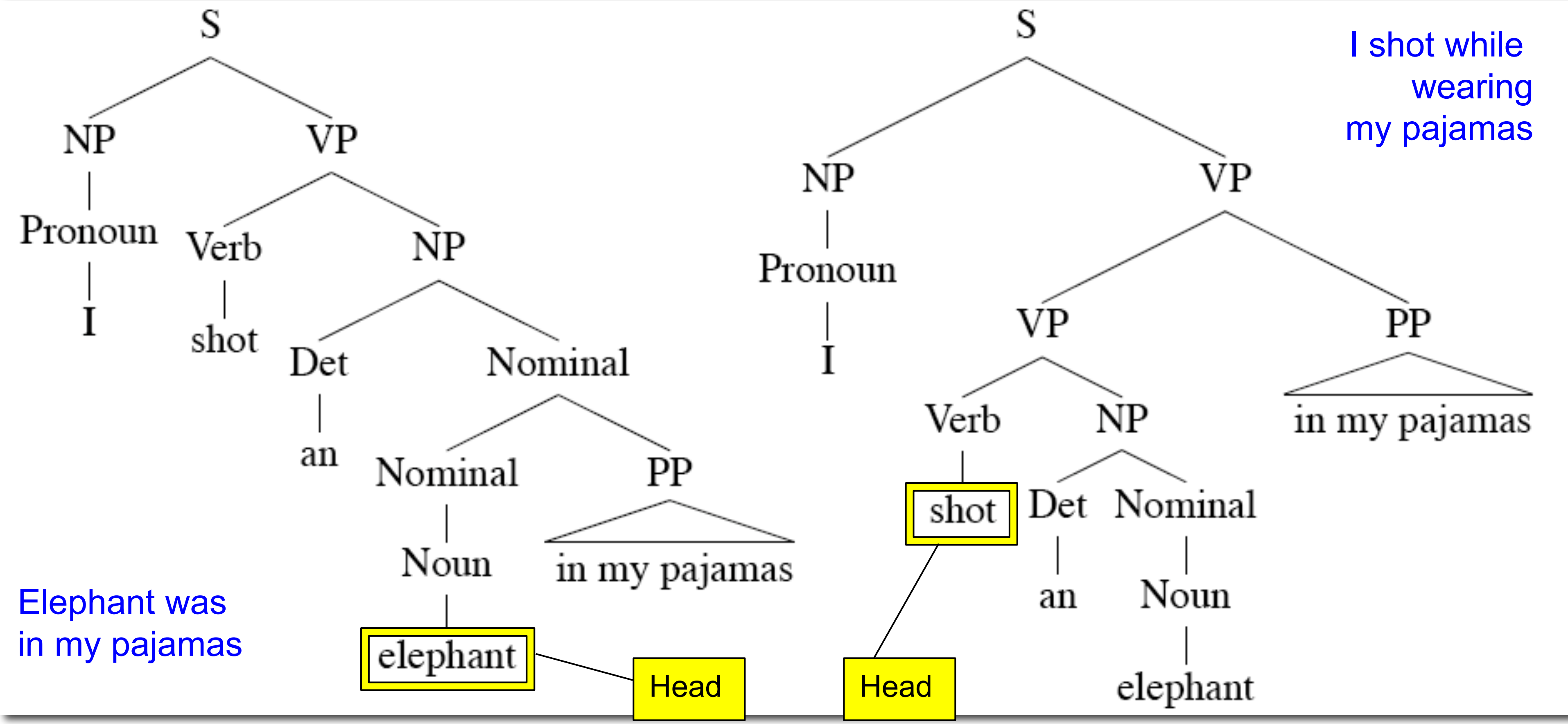
- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node
- One approach is called backtracking.
  - Make a choice, if it works out then fine
  - If not then back up and make a different choice

Problems

- Backtracking methods are doomed because of two inter-related problems
  - **Structural ambiguity**: the grammar assigns more than one possible parse to a phrase
  - **Repeated parsing** of subtrees

Ambiguity

- Structural ambiguity: the grammar assigns more than one possible parse to a phrase
  - "One morning, I shot an elephant in my pajamas. How he got in my pajamas I don't know" G. Marx, 1930

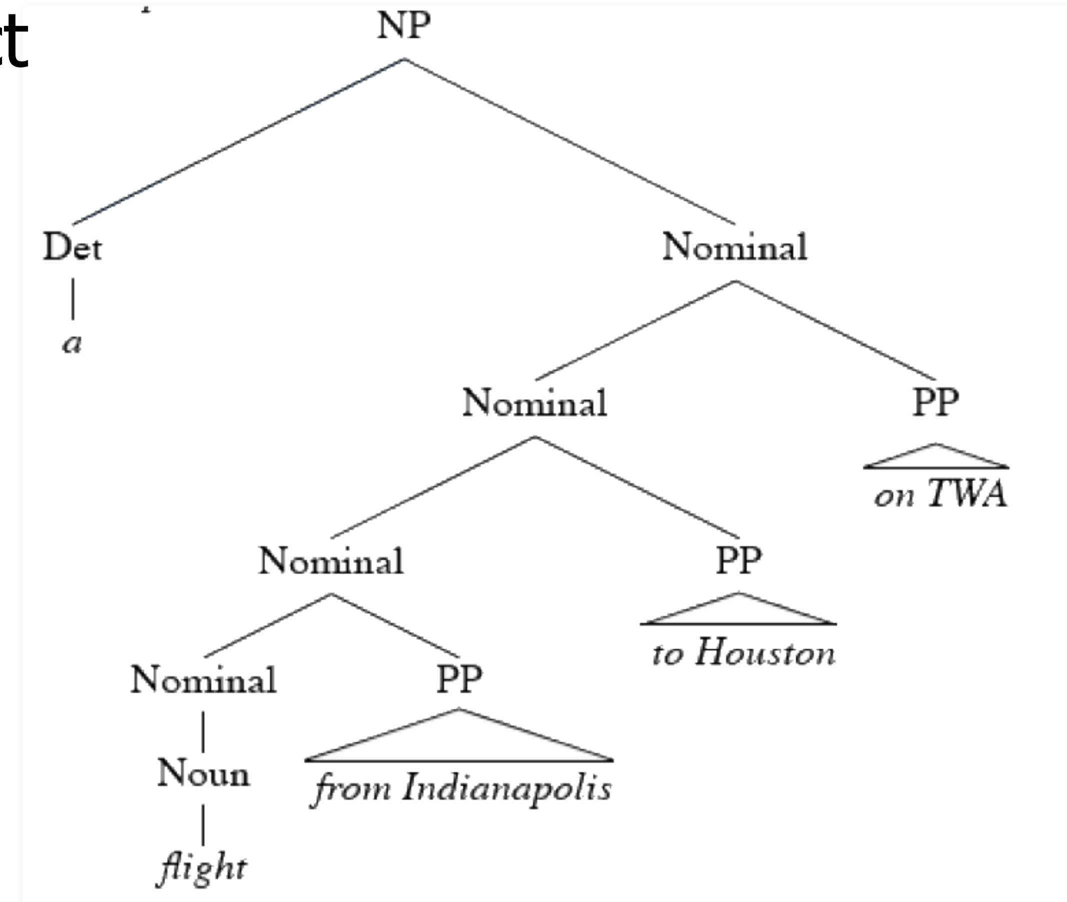


Repeated parsing of subtrees

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don't want to redo work we've already done.
  - Unfortunately, naïve backtracking will lead to duplicated work.

Repeated parsing of subtrees

- Consider
  - "A flight from Indianapolis to Houston on TWA"
  - Here is the correct parse

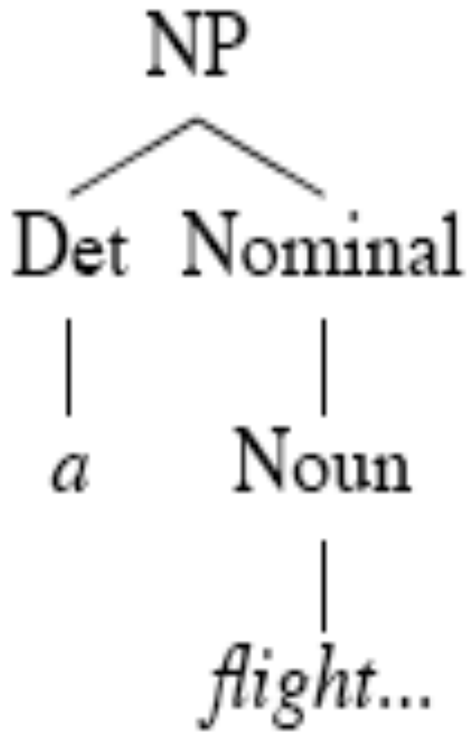


Repeated parsing of subtrees

- Assume a top-down parse making choices among the various Nominal rules.
- In particular, between these two
  - *Nominal* → *Noun*
  - *Nominal* → *Nominal PP*
- Statically choosing the rules in this order leads to the following bad results...

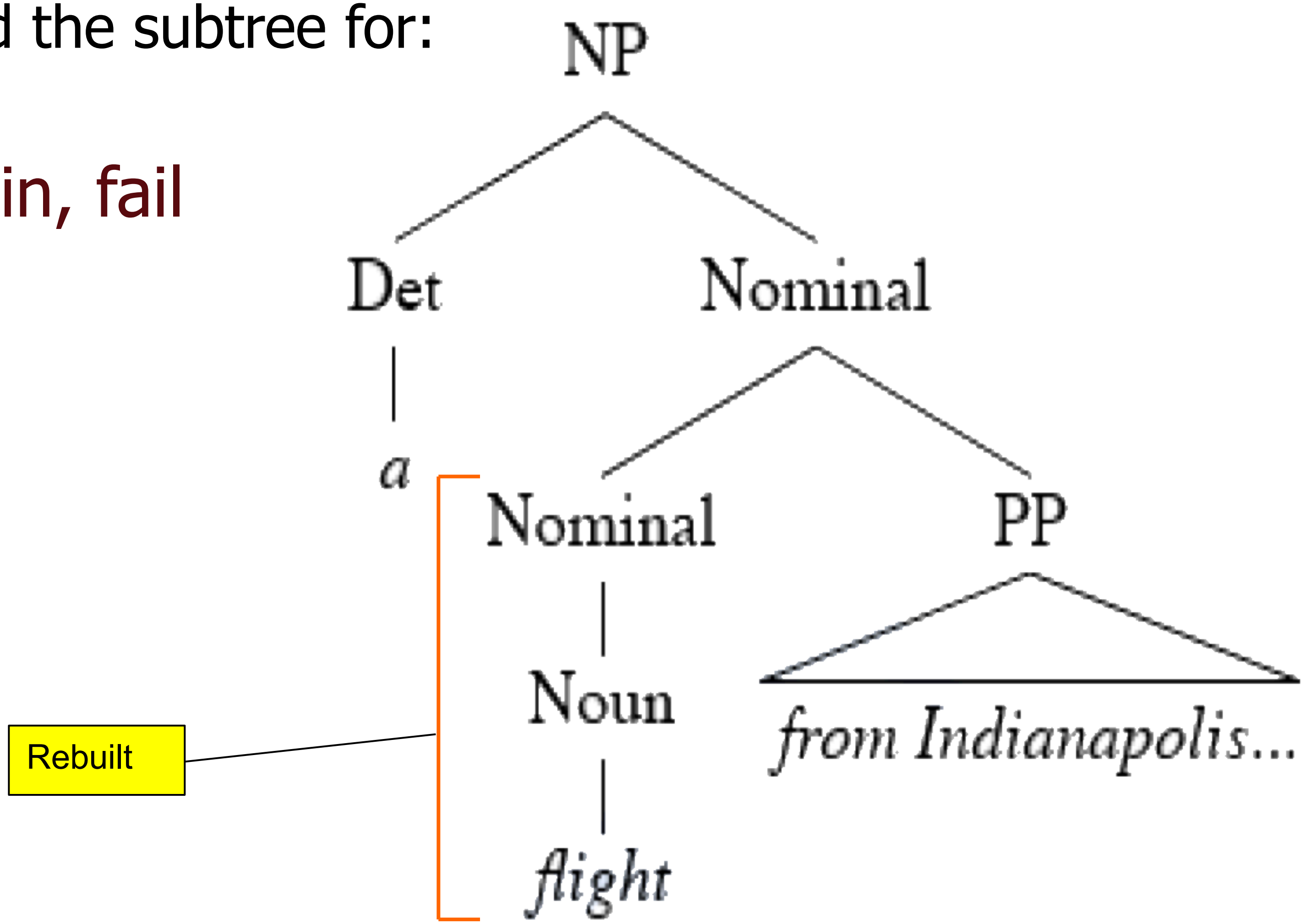
Repeated parsing of subtrees

- First attempt: *Nominal* → *Noun*
- Failed, as the tree does not cover all the input
- Backtrack…



Repeated parsing of subtrees

- Second attempt: *Nominal* → *Nominal PP*
  - Rebuild the subtree for: *flight*
- But, again, fail



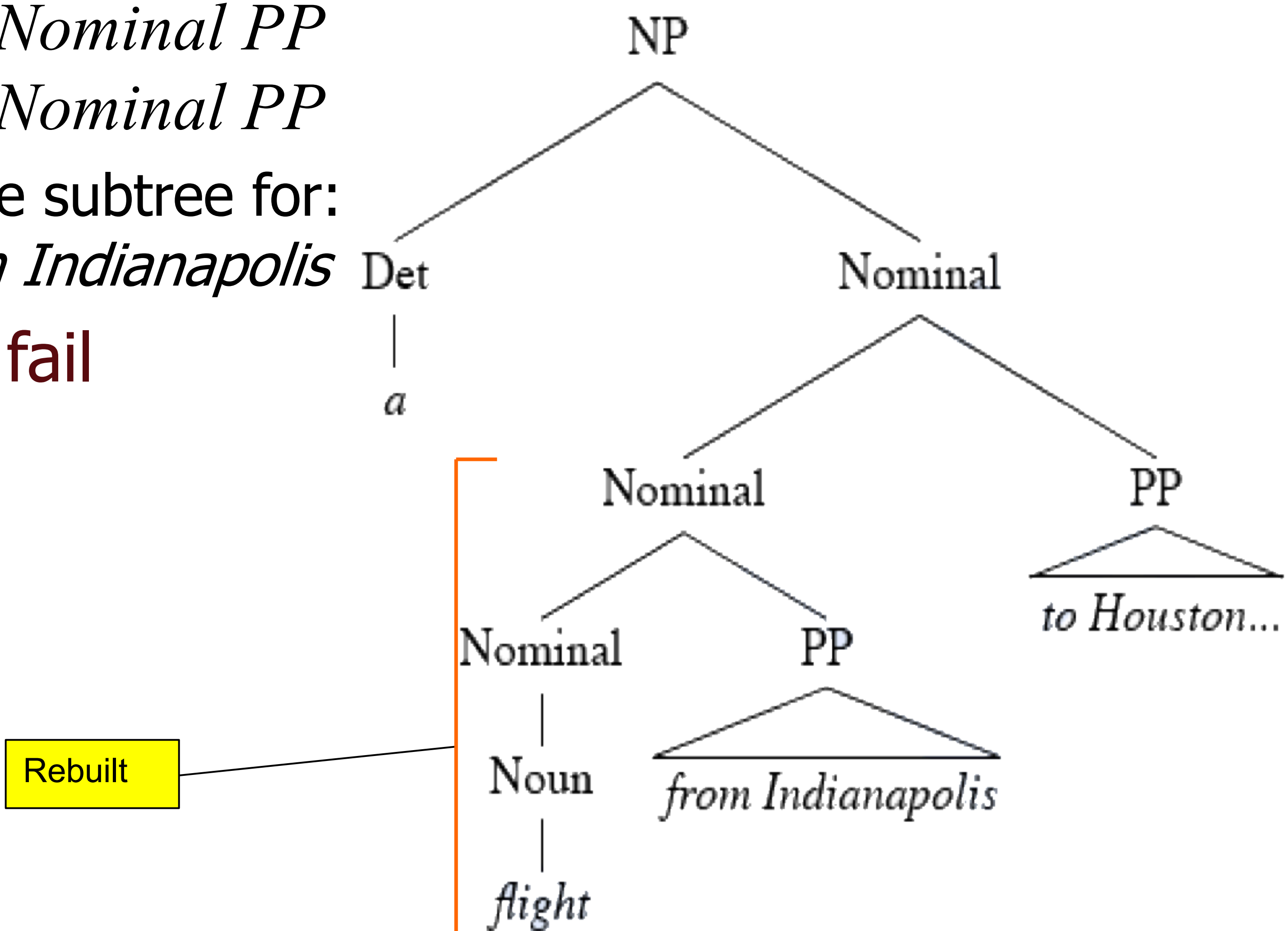
Repeated parsing of subtrees

- Third attempt:

  Nominal → Nominal PP

  Nominal → Nominal PP

  - Rebuild the subtree for: *flight from Indianapolis*
- But, again, fail



Repeated parsing of subtrees

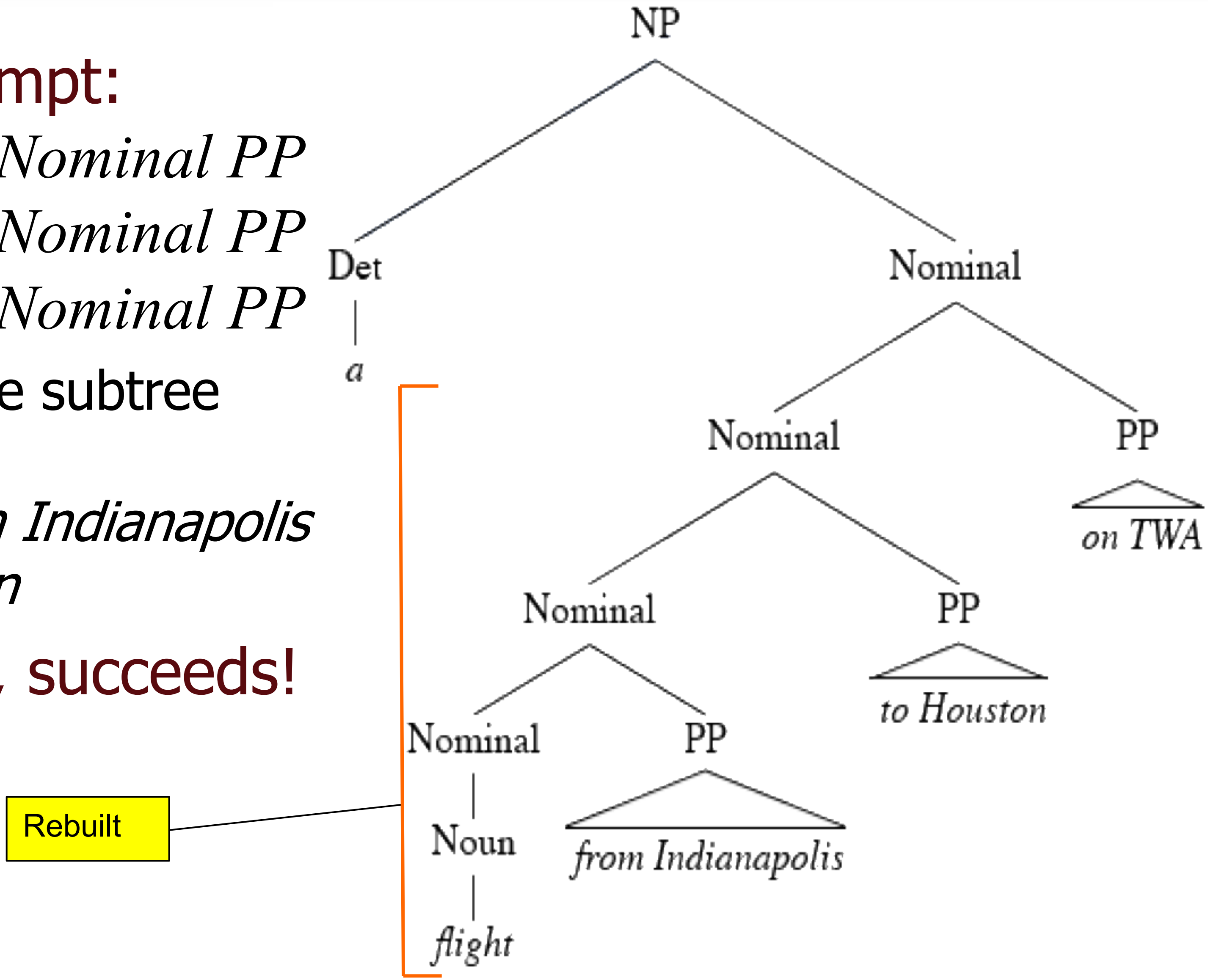
- Fourth attempt:

  Nominal → Nominal PP

  Nominal → Nominal PP

  Nominal → Nominal PP

  - Rebuild the subtree for:
    flight from Indianapolis to Houston
- And, finally, succeeds!



Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Efficiently store ambiguous structures with shared sub-parts.
- Two important approaches that roughly correspond to top-down and bottom-up approaches.
  - CKY
  - Earley

Ambiguity

- Did we solve it?
- No…
  - We still have multiple S structures.
  - CKY and Earley both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

Ambiguity

- In most cases, humans don't notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- We'll try to model that with probabilities.