Parsing

Slides by James Martin, adapted by Diana Inkpen
for CSI 5386 @ uOttawa

Verb Phrases

- English *VP*s consist of a head verb along with 0 or more following constituents which we'll call *arguments*.

VP	$\rightarrow$	$Verb$	disappear
VP	$\rightarrow$	$Verb\ NP$	prefer a morning flight
VP	$\rightarrow$	$Verb\ NP\ PP$	leave Boston in the morning
VP	$\rightarrow$	$Verb\ PP$	leaving on Thursday

Subcategorization

- Even though there are many valid VP rules in English, not all verbs are allowed to participate in all those VP rules.
- We can *subcategorize* the verbs in a language according to the sets of VP rules that they participate in.
- This is just an elaboration on the traditional notion of transitive/intransitive.
- Modern grammars have many such classes

Subcategorization

- Sneeze: John sneezed
- Find: Please find [a flight to NY]$_{NP}$
- Give: Give [me]$_{NP}$[a cheaper fare]$_{NP}$
- Help: Can you help [me]$_{NP}$[with a flight]$_{PP}$
- Prefer: I prefer [to leave earlier]$_{TO\text{-}VP}$
- Told: I was told [United has a flight]$_{S}$
- …

Programming Analogy

- It may help to view things this way
  - Verbs are functions or methods
  - They participate in specify the number, position, and type of the arguments they take...
    - That is, just like the formal parameters to a method.

Subcategorization

- *John sneezed the book
- *I prefer United has a flight
- *Give with a flight

- As with agreement phenomena, we need a way to formally express these facts

Why?

- Right now, the various rules for VPs *overgenerate*.
  - They permit the presence of strings containing verbs and arguments that don't go together
  - For example
  - VP -> V NP therefore

    Sneezed the book is a VP since "sneeze" is a verb and "the book" is a valid NP

Possible CFG Solution

- Possible solution for agreement.
- Can use the same trick for all the verb/VP classes.

- SgS -> SgNP SgVP
- PlS -> PlNp PlVP
- SgNP -> SgDet SgNom
- PlNP -> PlDet PlNom
- PlVP -> PlV NP
- SgVP ->SgV Np
- …

CFG Solution for Agreement

- It works and stays within the power of CFGs
  - But it is a fairly ugly one
- And it doesn’t scale all that well because of the interaction among the various constraints explodes the number of rules in our grammar.

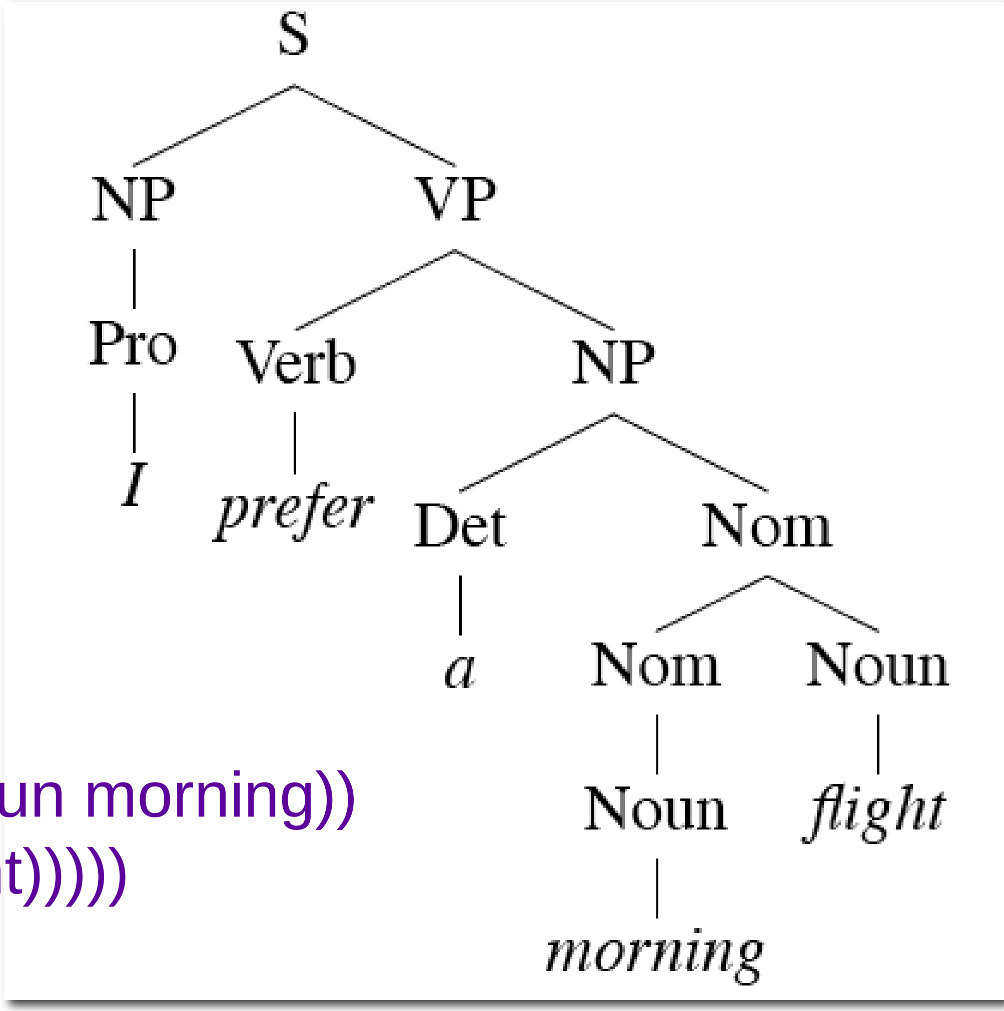
Summary

- CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.
- But there are problems
  - That can be dealt with adequately, although not elegantly, by staying within the CFG framework.
- There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power)
  - LFG, HPSG, Construction grammar, XTAG, etc.
  - Chapter 15 explores one approach (feature unification) in more detail

Treebanks

- Treebanks are corpora in which each sentence has been paired with a parse structure (presumably the correct one).
- These are generally created
  1. By first parsing the collection with an automatic parser
  2. And then having human annotators hand correct each parse as necessary.
- This generally requires detailed annotation guidelines that provide a POS tagset, a grammar, and instructions for how to deal with particular grammatical constructions.

Parens and Trees



(S (NP (Pro I))
(VP (Verb prefer)
(NP (Det a)
(Nom (Nom (Noun morning))
(Noun flight)))))

Penn Treebank

- Penn TreeBank is a widely used treebank.

```
( (S (‘‘ ‘‘)
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those)(NNS assets)))))))))))))
    (, ,) (’’ ’’)
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Most well known part is the Wall Street Journal section of the Penn TreeBank.

- 1 M words from the 1987-1989 Wall Street Journal.

Treebank Grammars

- Treebanks implicitly define a grammar for the language covered in the treebank.
- Simply take the local rules that make up the sub-trees in all the trees in the collection and you have a grammar
  - The WSJ section gives us about 12k rules if you do this
- Not complete, but if you have decent size corpus, you will have a grammar with decent coverage.

Treebank Grammars

- Such grammars tend to be very flat due to the fact that they tend to avoid recursion.
  - To ease annotator's burden, among things
- For example, the Penn Treebank has ~4500 different rules for VPs. Among them...

$$VP \rightarrow VBD\ PP$$
$$VP \rightarrow VBD\ PP\ PP$$
$$VP \rightarrow VBD\ PP\ PP\ PP$$
$$VP \rightarrow VBD\ PP\ PP\ PP\ PP$$

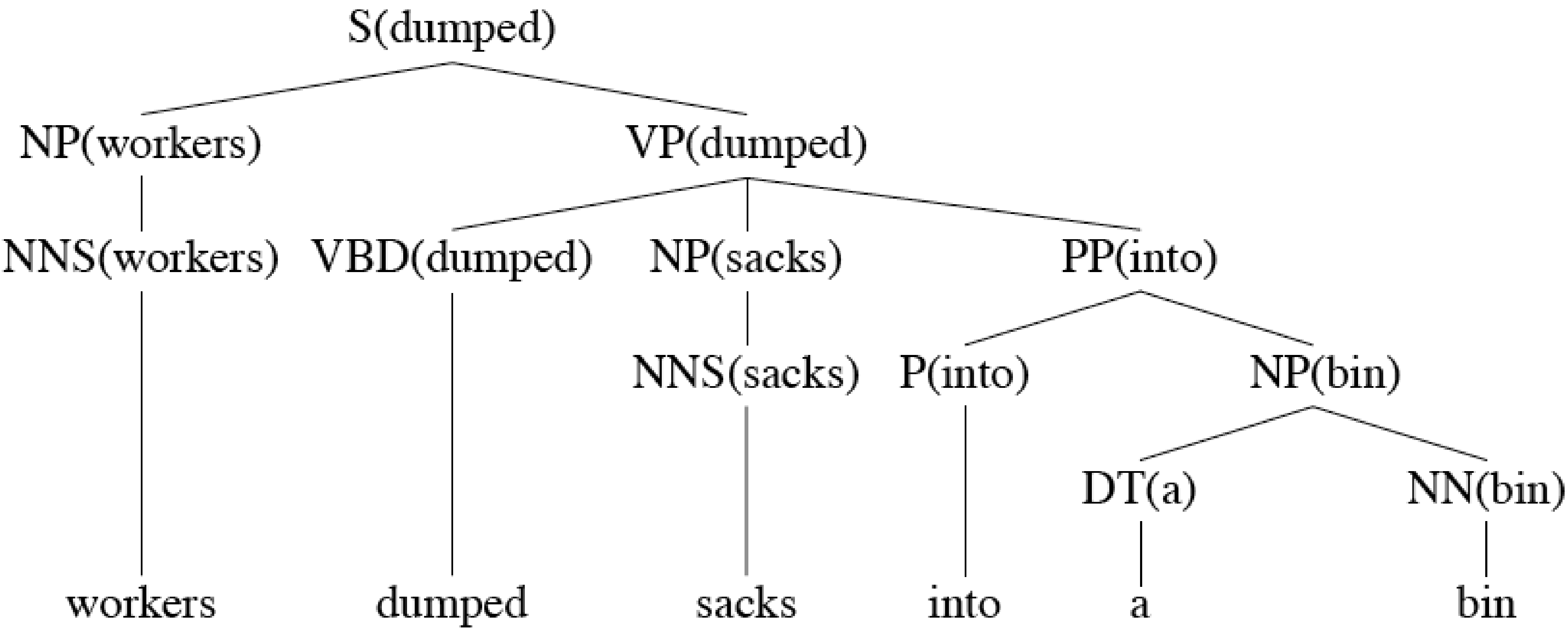
Treebank Uses

- Treebanks (and head-finding) are particularly critical to the development of statistical parsers
  - Chapter 14
    - We will get there
- Also valuable to *Corpus Linguistics*
  - Investigating the empirical details of various constructions in a given language
    - How often do people use various constructions and in what contexts...
    - Do people ever say X ...

Head Finding

- Finding heads in treebank trees is a task that arises frequently in many applications.
  - As we'll see it is particularly important in statistical parsing
- We can visualize this task by annotating the nodes of a parse tree with the heads of each corresponding node.

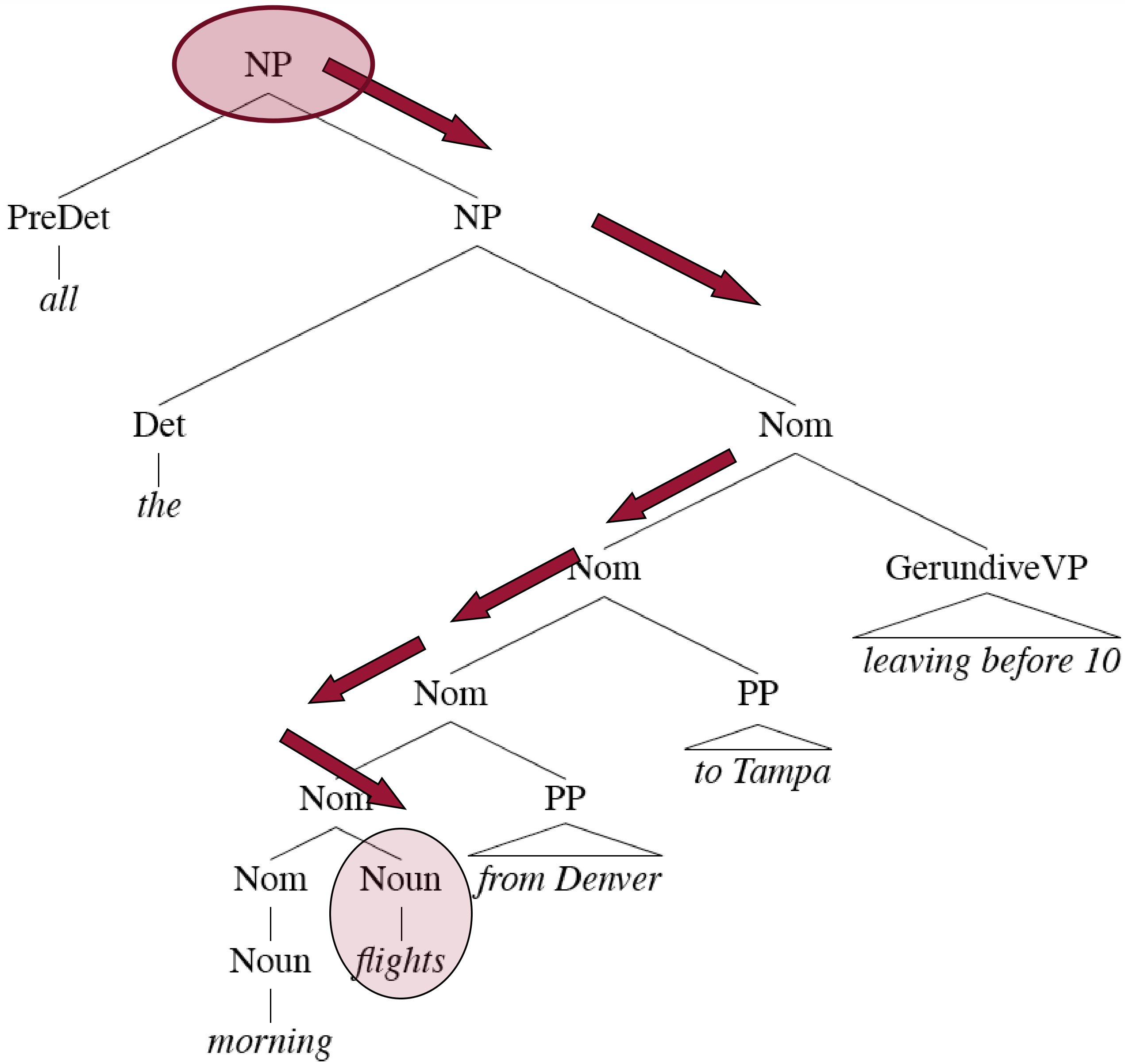
Lexically Decorated Tree



Head Finding

- Given a tree, the standard way to do head finding is to use a simple set of tree traversal rules specific to each non-terminal in the grammar.

Noun Phrases



Treebank Uses

- Treebanks (and head-finding) are particularly critical to the development of statistical parsers
  - Chapter 14
- Also valuable to *Corpus Linguistics*
  - Investigating the empirical details of various constructions in a given language

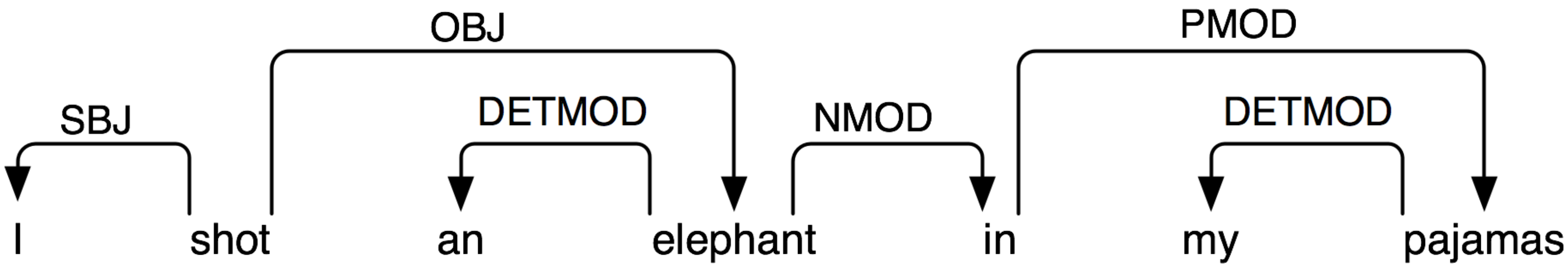
Dependency Grammars

- In CFG-style phrase-structure grammars the main focus is on *constituents* and *ordering*.
- But it turns out you can get a lot done with just labeled relations among the words in an utterance.
- In a dependency grammar framework, a parse is a tree where
  - The nodes stand for the words in an utterance
  - The links between the words represent dependency relations between pairs of words.
    - Relations may be typed (labeled), or not.

Dependency Relations

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Dependency Parse



Dependency Parsing

- The dependency approach has a number of advantages over full phrase-structure parsing.
  - It deals well with free word order languages where the constituent structure is quite fluid
  - Parsing is *much faster* than with CFG-based parsers
  - Dependency structure often captures the syntactic relations needed by later applications
    - CFG-based approaches often extract this same information from trees anyway

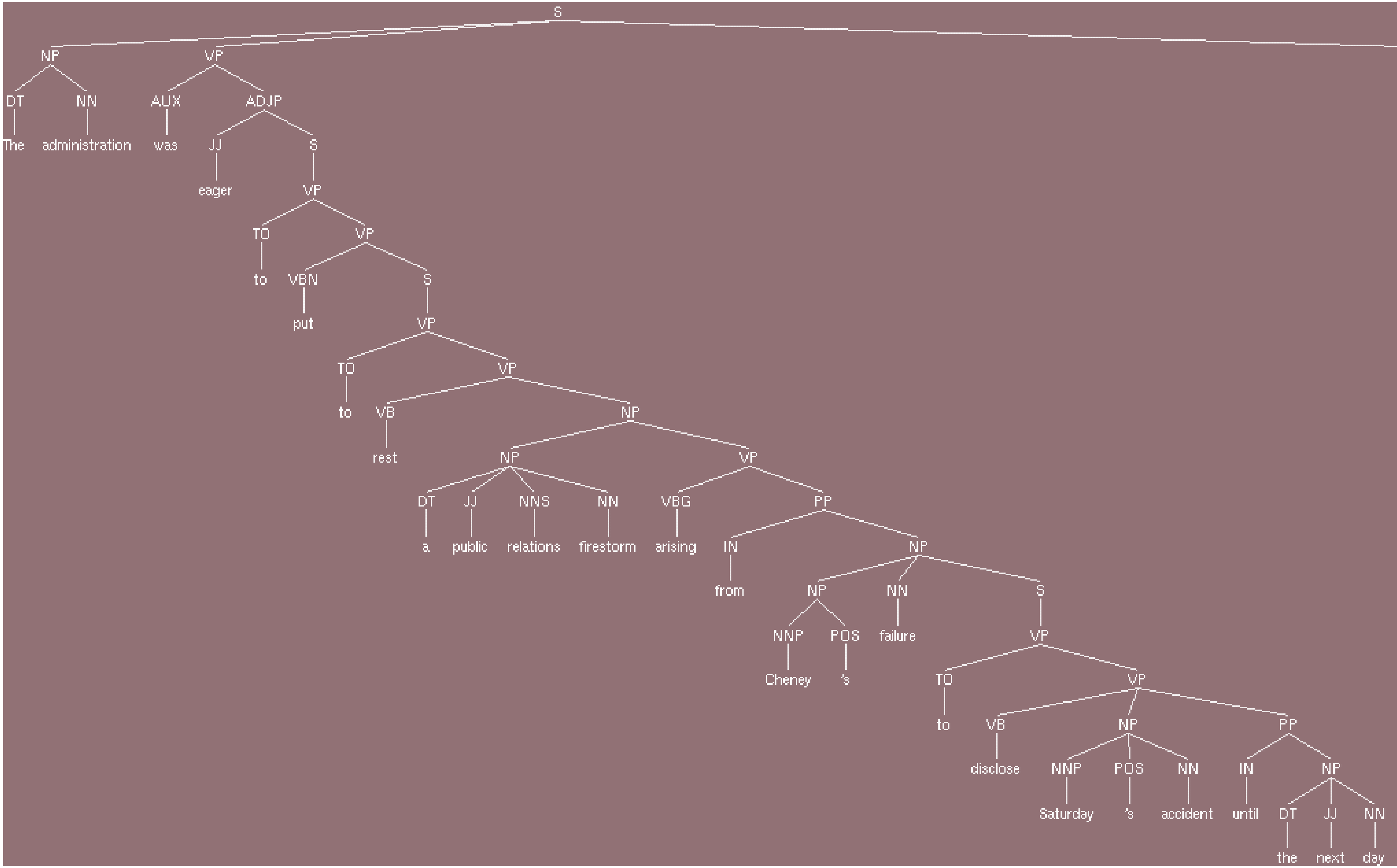
Summary

- Context-free grammars can be used to model various facts about the syntax of a language.
- When paired with parsers, such grammars consititute a critical component in many applications.
- Constituency is a key phenomena easily captured with CFG rules.
  - But agreement and subcategorization do pose significant problems
- Treebanks pair sentences in corpus with their corresponding trees.

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
- It doesn't actually mean that the system can select the correct tree from among all the possible trees

Automatic Syntactic Parse



For Now

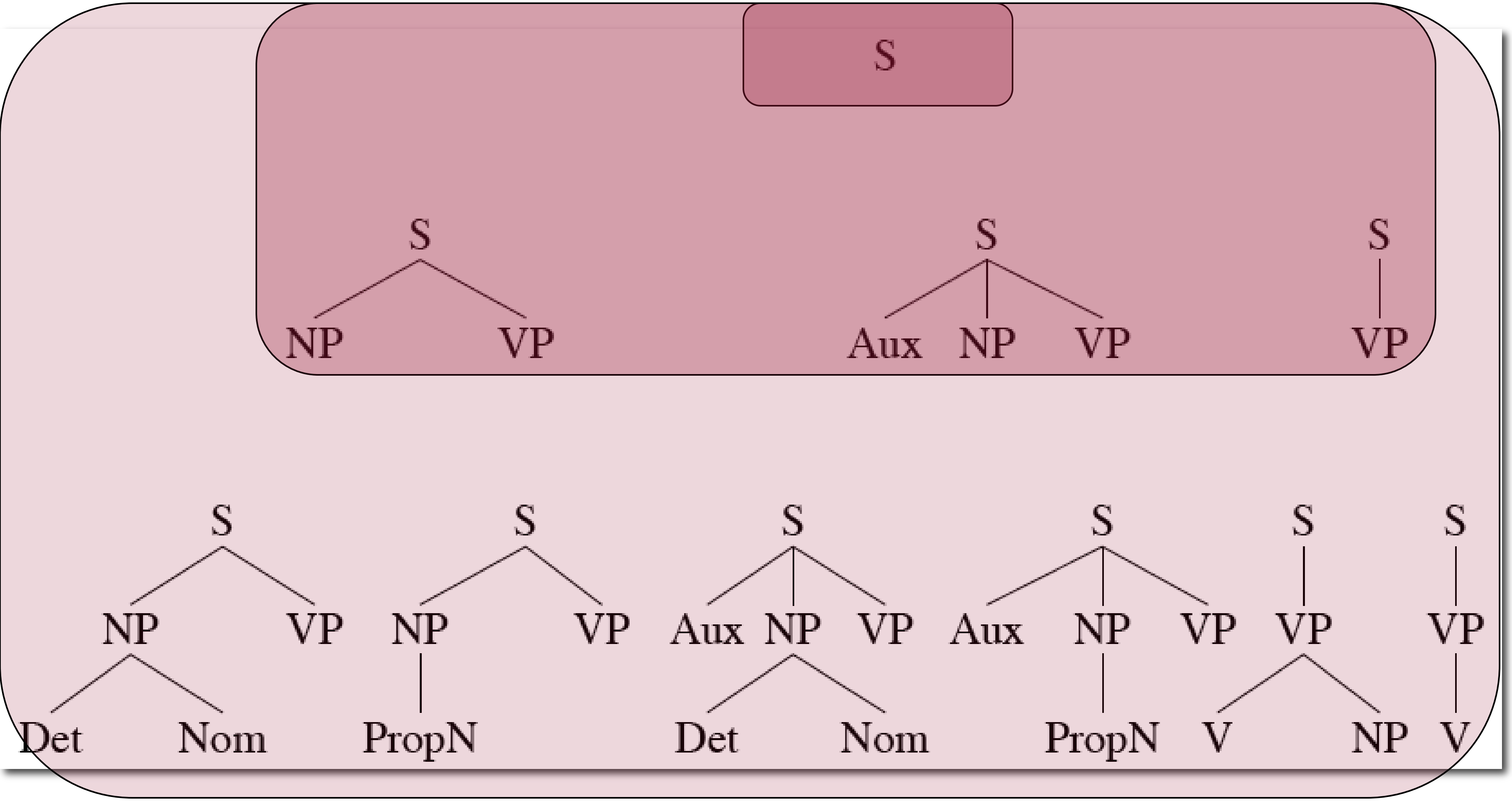
- Assume…
  - You have all the words already in some buffer
  - The input is not POS tagged prior to parsing
  - We won't worry about morphological analysis
  - All the words are known

  - 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.

Top Down Space



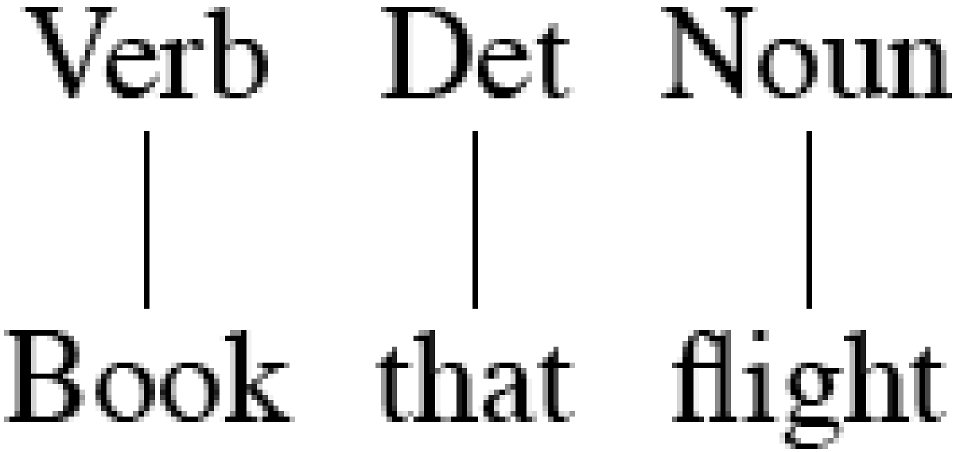
Bottom-Up Parsing

- Of course, 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.

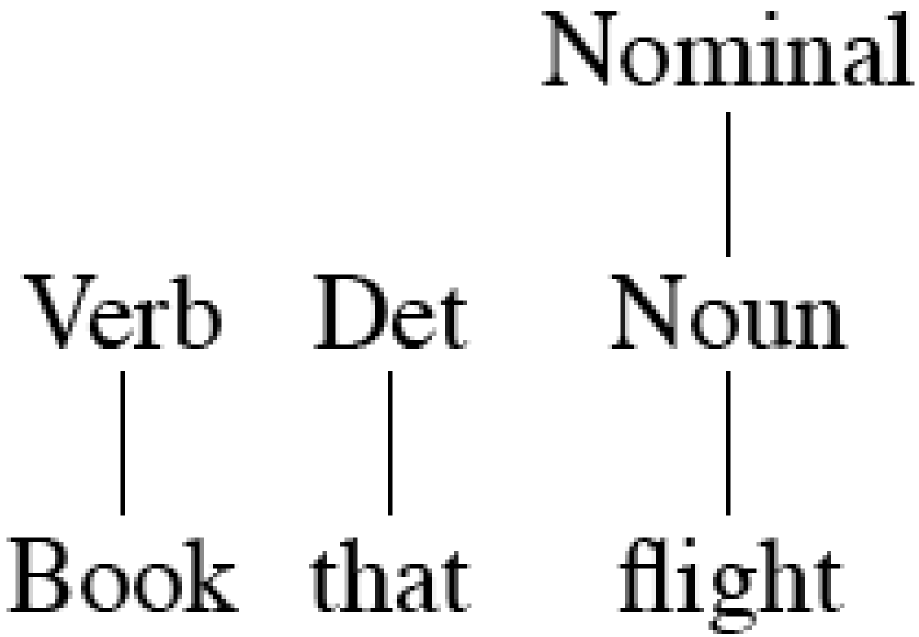
Bottom-Up Search

Book that flight

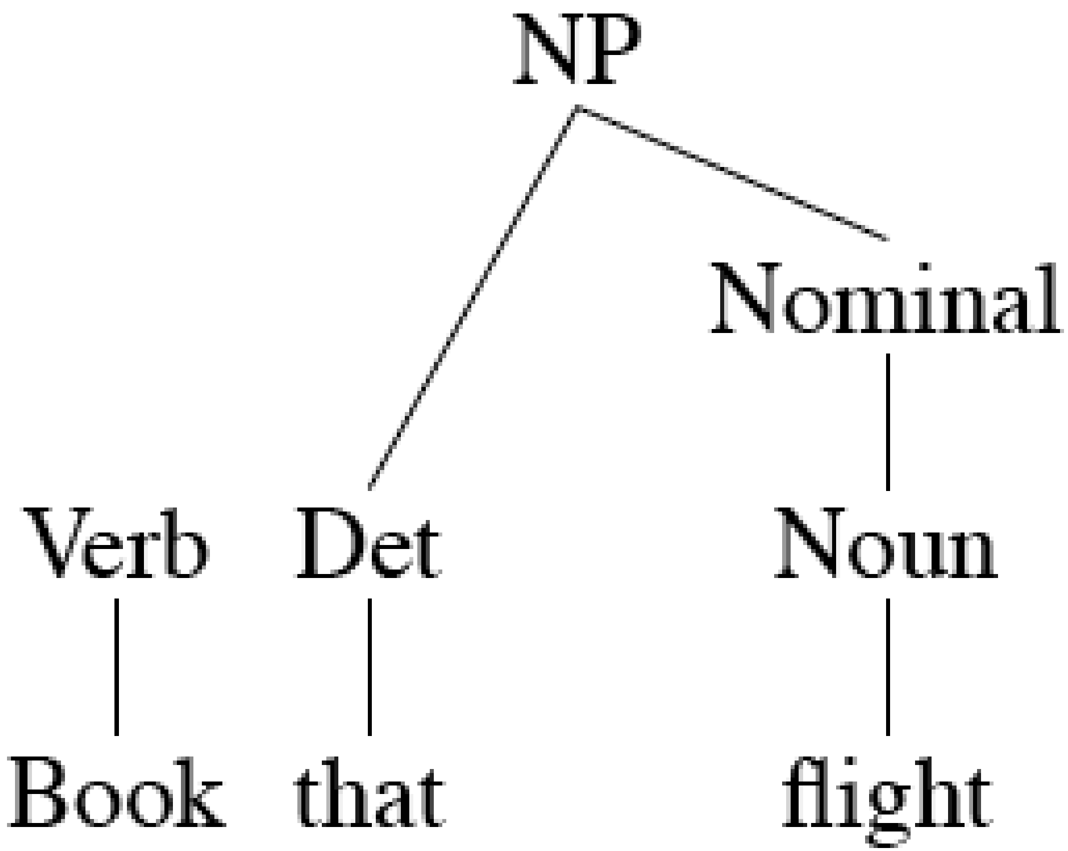
Bottom-Up Search



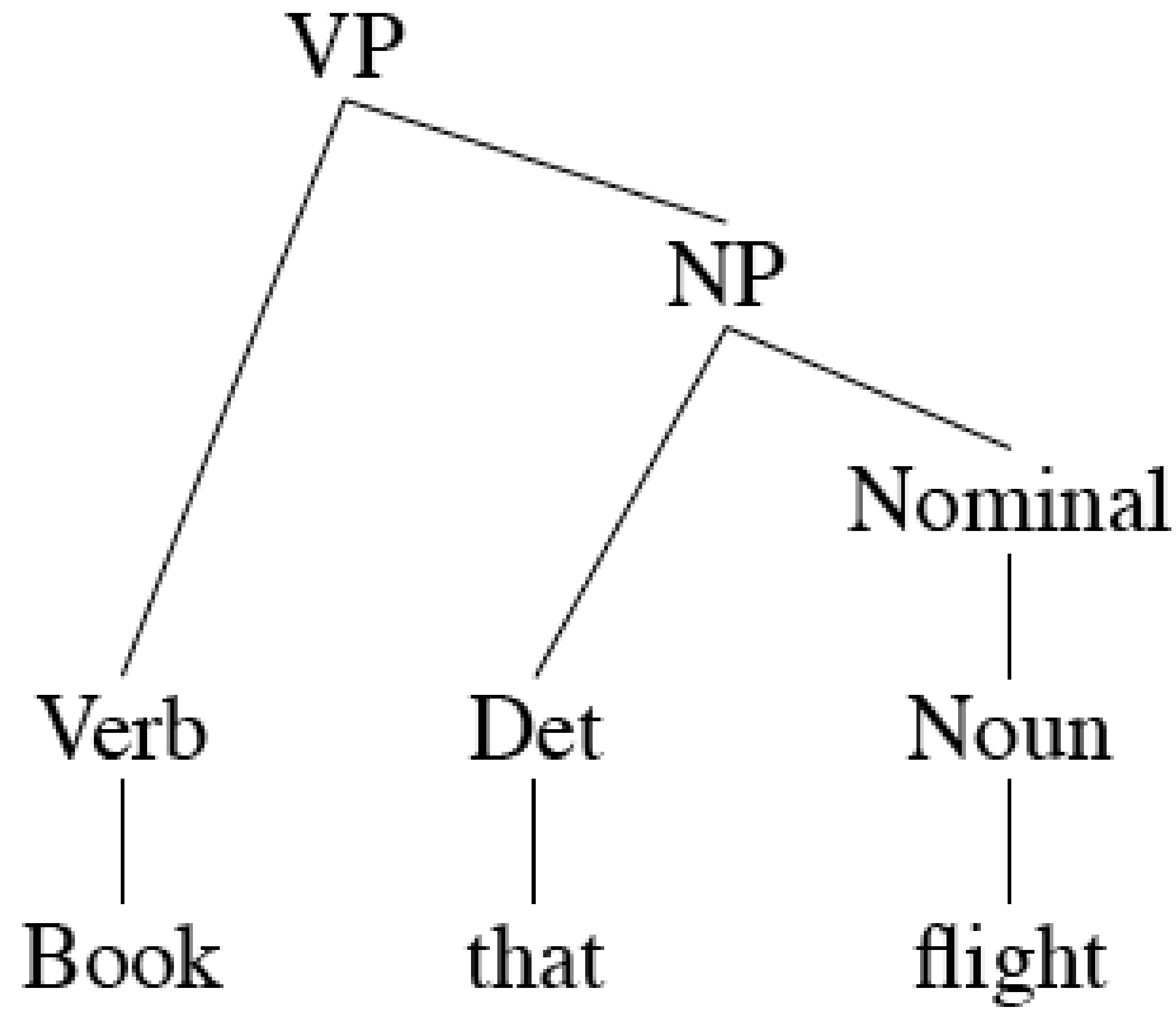
Bottom-Up Search



Bottom-Up Search



Bottom-Up Search



Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers (i.e. S's)
  - But also suggests trees that are not consistent with any of the words
- Bottom-up
  - Only forms trees consistent with the words
  - But suggests trees that make no sense globally

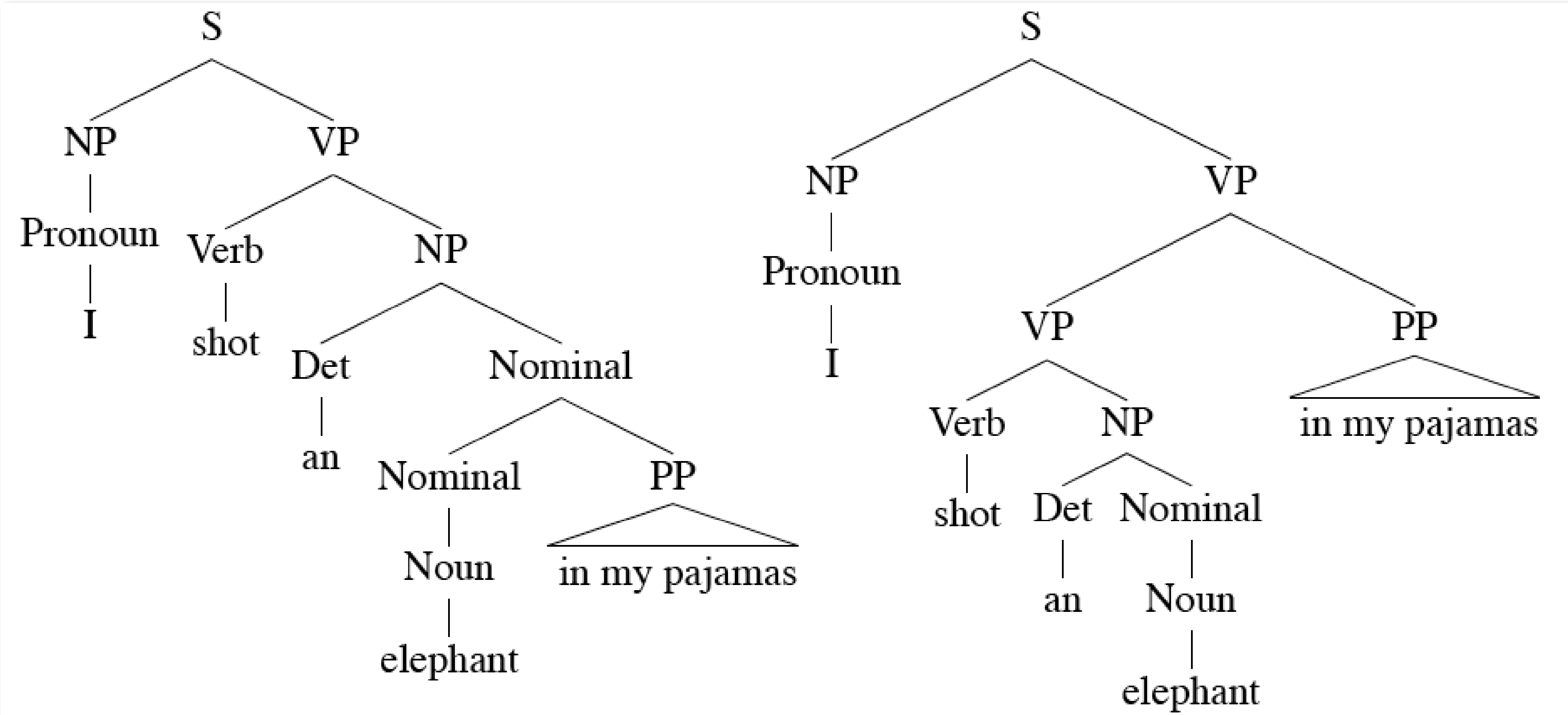
Control

- 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
    - Same as with ND-Recognize

Problems

- Even with the best filtering, backtracking methods are doomed because of two inter-related problems
  - Ambiguity and search control (choice)
  - Shared subproblems

Ambiguity



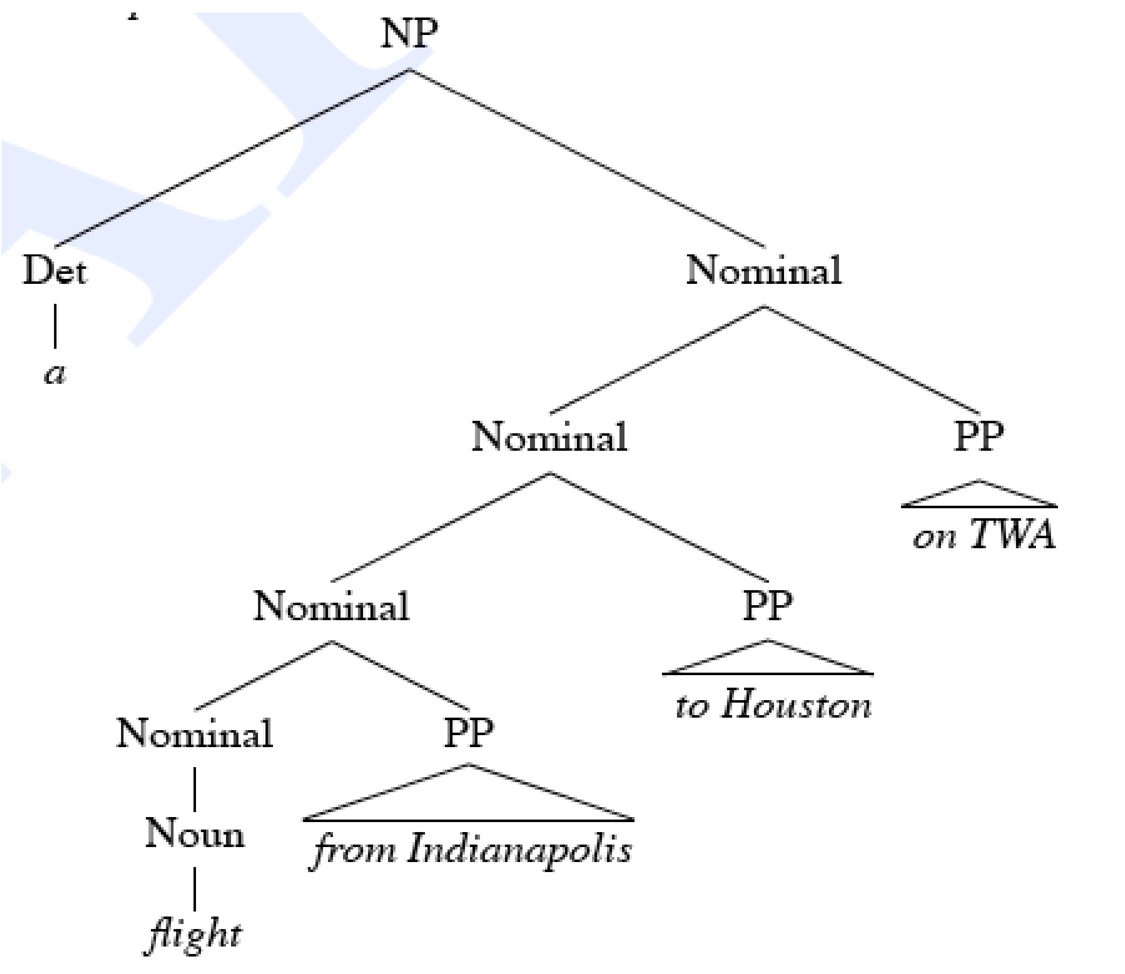
S
NP
VP
Pronoun
Verb
NP
I
shot
Det
Nominal
an
Nominal
PP
Noun
in my pajamas
elephant
S
NP
VP
Pronoun
VP
PP
I
Verb
NP
in my pajamas
shot
Det
Nominal
an
Noun
elephant

Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose...
  - We can’t afford to redo work we’ve already done.
  - Without some help naïve backtracking will lead to such duplicated work.

Shared Sub-Problems

- Consider
  - *A flight from Indianapolis to Houston on TWA*



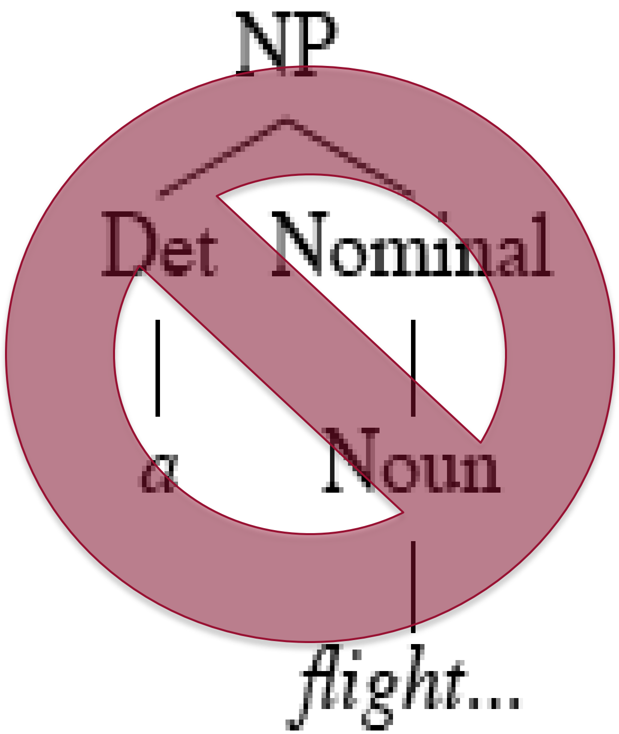
Sample L1 Grammar

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

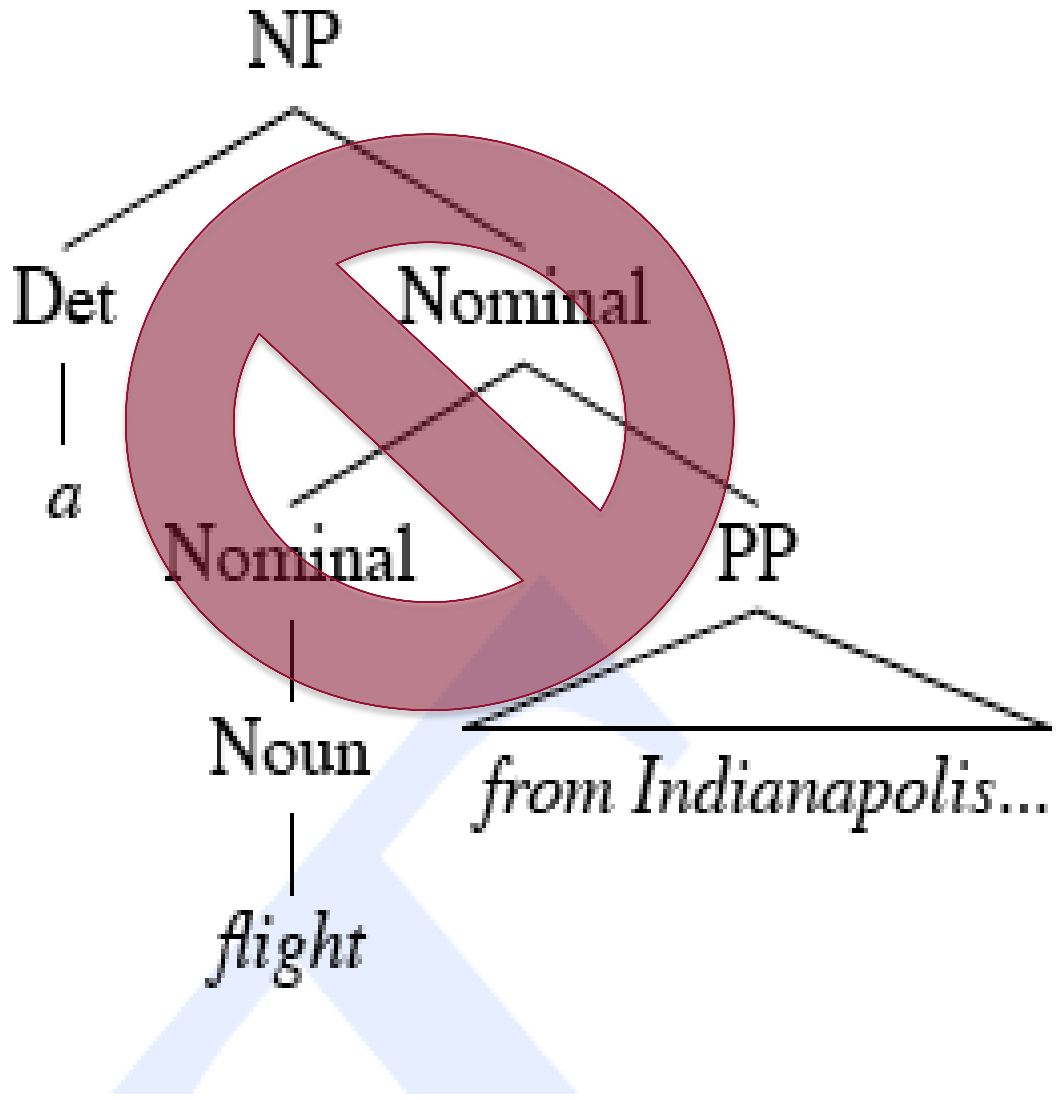
Shared Sub-Problems

- Assume a top-down parse that has already expanded the *NP* rule (dealing with the Det)
- Now its 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 behavior...

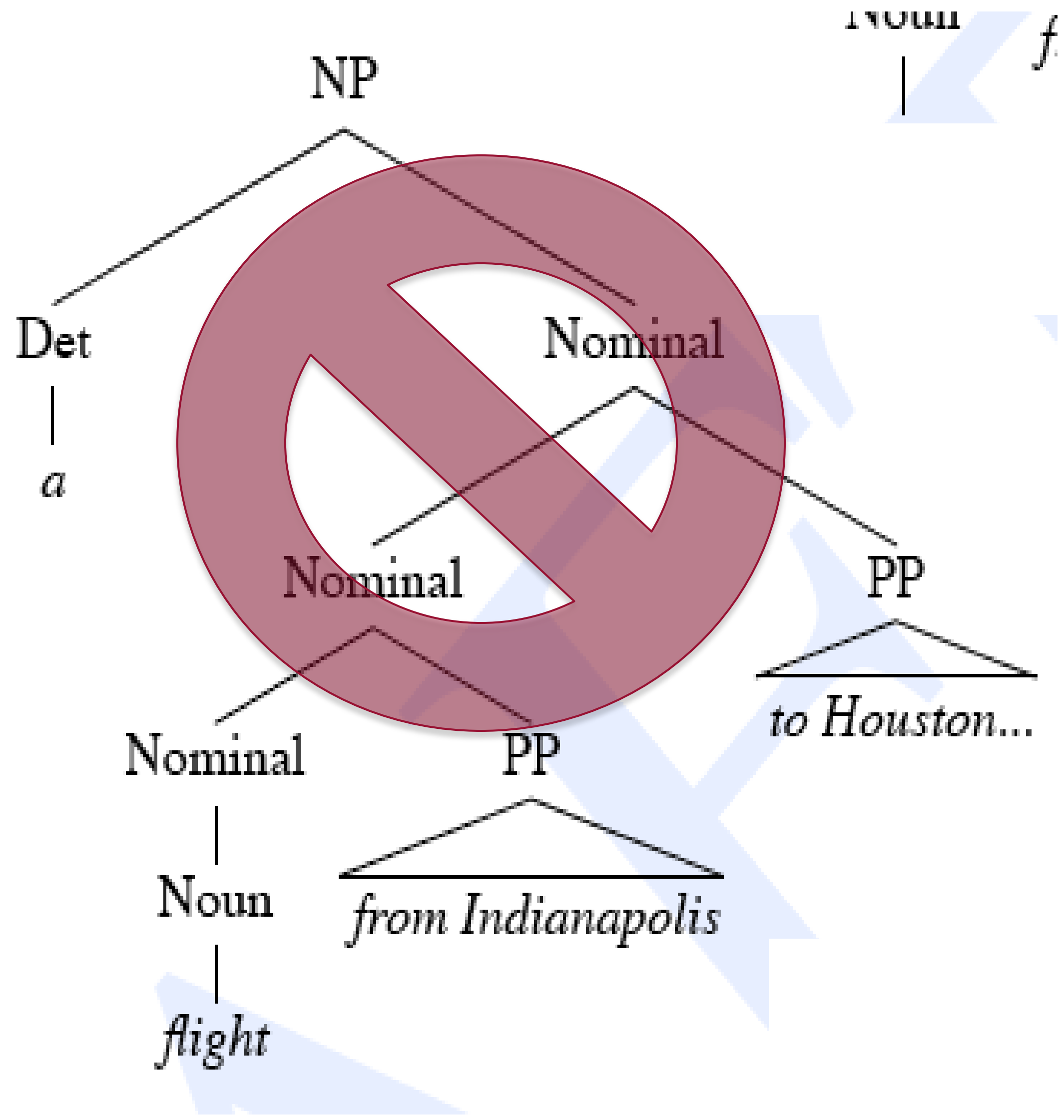
Shared Sub-Problems



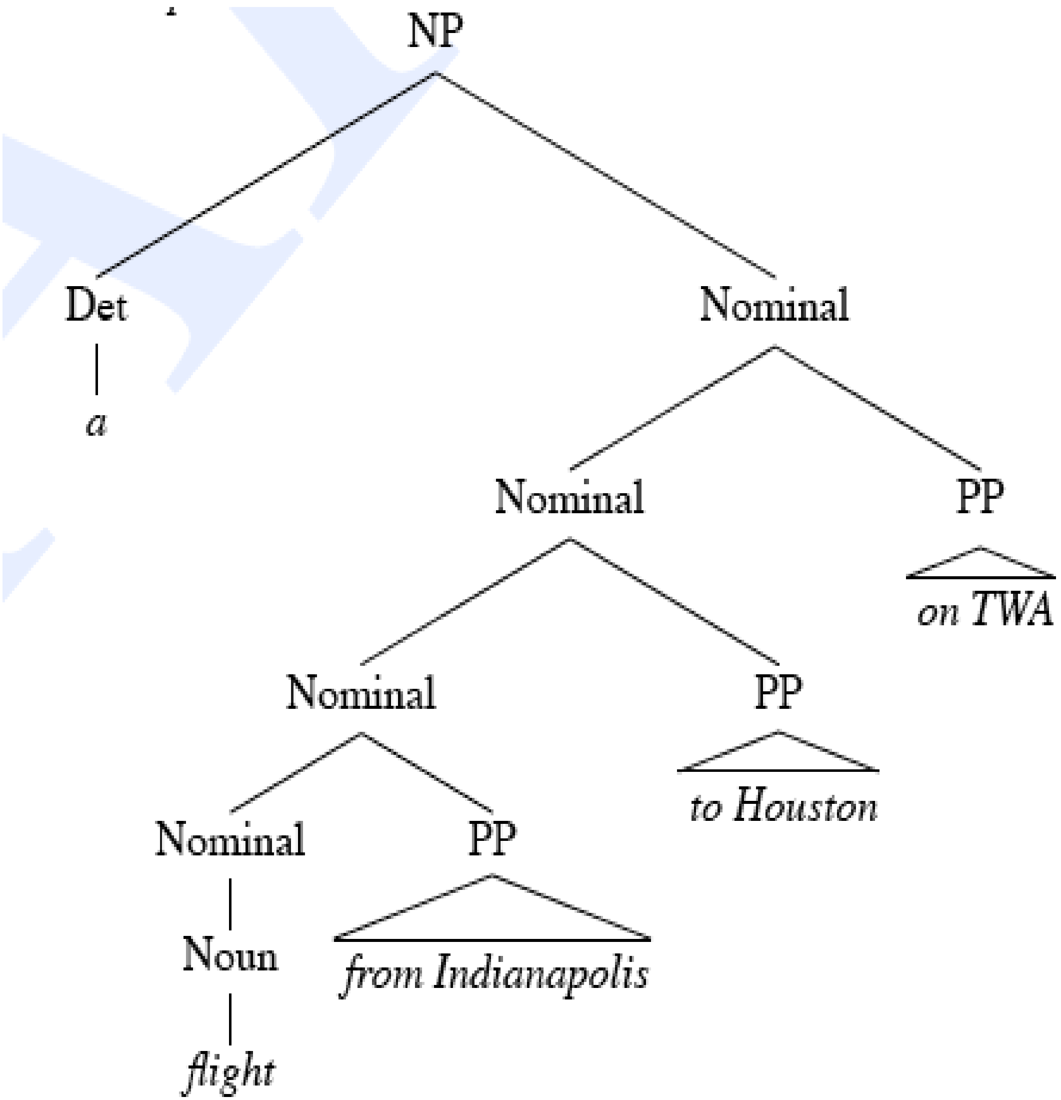
Shared Sub-Problems



Shared Sub-Problems



Shared Sub-Problems



Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Solve exponential problems in polynomial time (well not really)
  - Efficiently store ambiguous structures with shared sub-parts.
- We'll cover two approaches that roughly correspond to top-down and bottom-up approaches.
  - CKY
  - Earley

CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more on this later)
- Consider the rule $A \rightarrow BC$
  - If there is an A somewhere in the input generated by this rule then there must be a B followed by a C in the input.
  - If the A spans from i to j in the input then there must be some k st. i<k<j
    - In other words, the B splits from the C someplace after the i and before the j.

CKY

- Let's build *a table* so that an *A* spanning from i to j in the input is placed in cell [i,j] in the table.
  - So a non-terminal spanning an entire string will sit in cell [0, n]
    - Hopefully it will be an *S*
- Now we know that the parts of the A must go from i to k and from k to j, for some k

CKY

- Meaning that for a rule like A → B C we should look for a B in [i,k] and a C in [k,j].
- In other words, if we think there might be an A spanning i,j in the input… AND

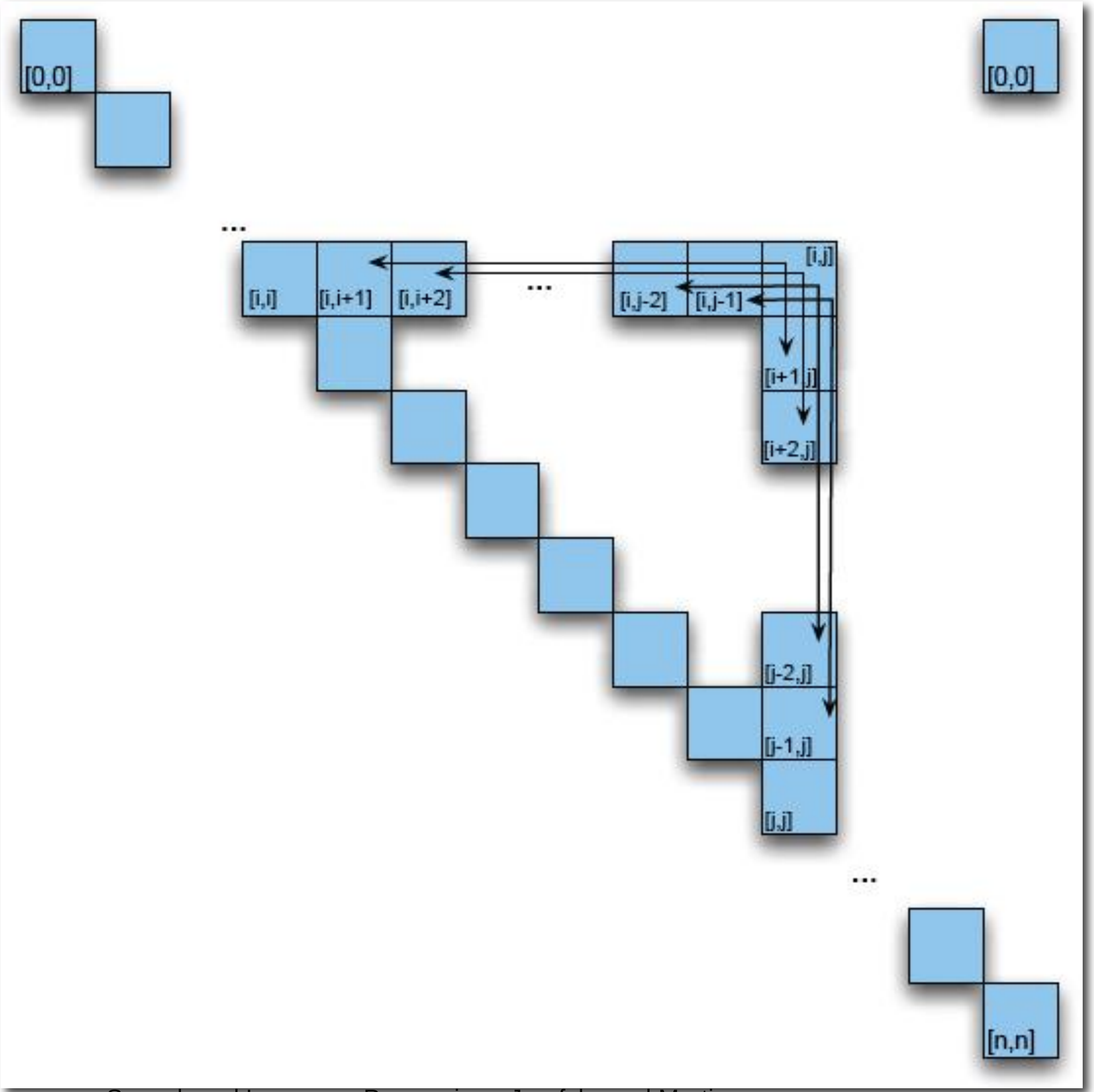
  A → B C is a rule in the grammar THEN
- There must be a B in [i,k] and a C in [k,j] for some k such that i<k<j

What about the B and the C?

CKY

- So to fill the table loop over the cells [i,j] values in some systematic way
  - Then for each cell, loop over the appropriate k values to search for things to add.
  - Add all the derivations that are possible for each [i,j] for each k

CKY Table



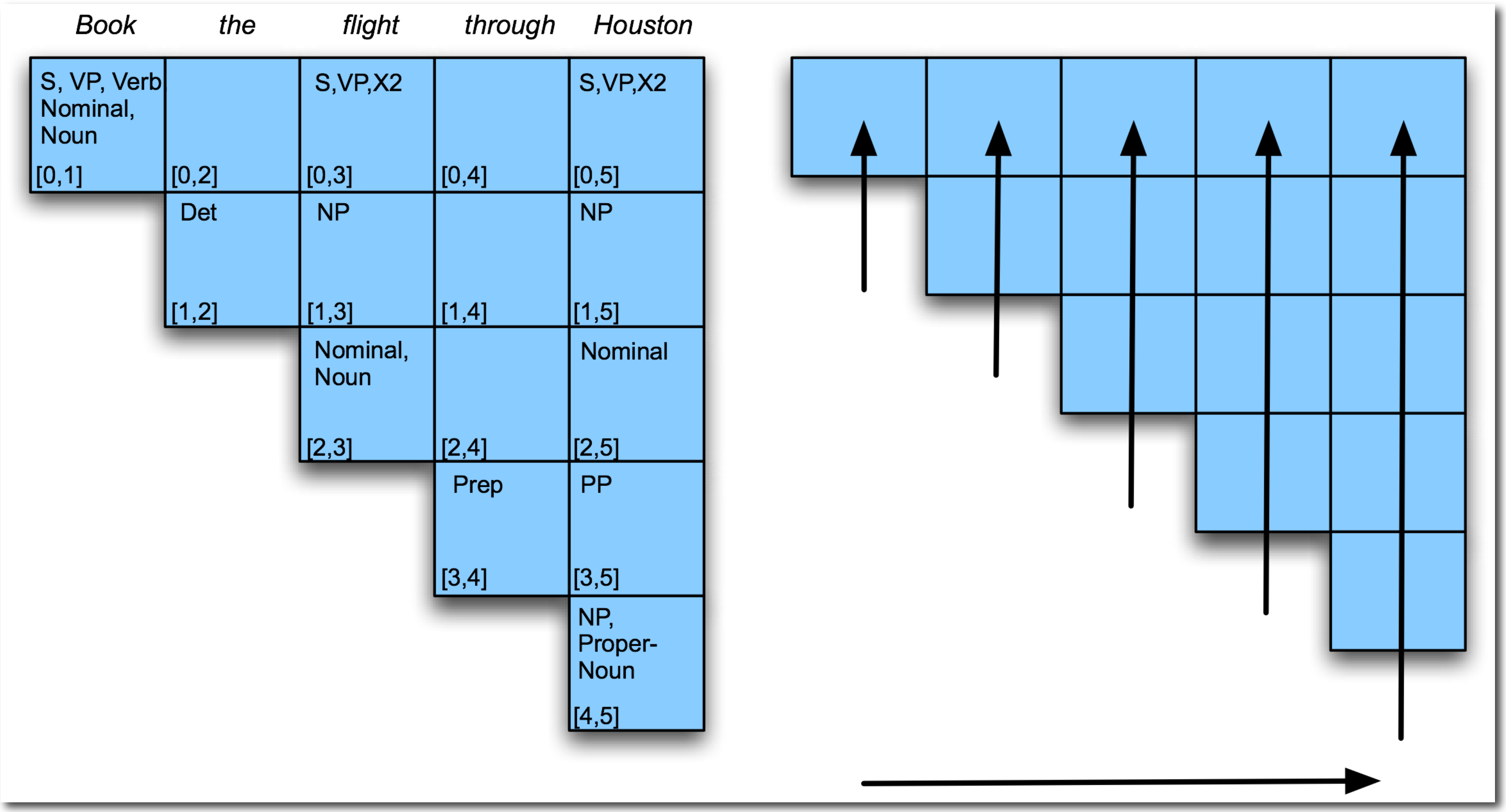
CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

What's the complexity of this?

Example


Book
the
flight
through
Houston
S, VP, Verb Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
S,VP,X2
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

Example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	[3,5]
				NP, Proper-Noun [4,5]

Filling column 5

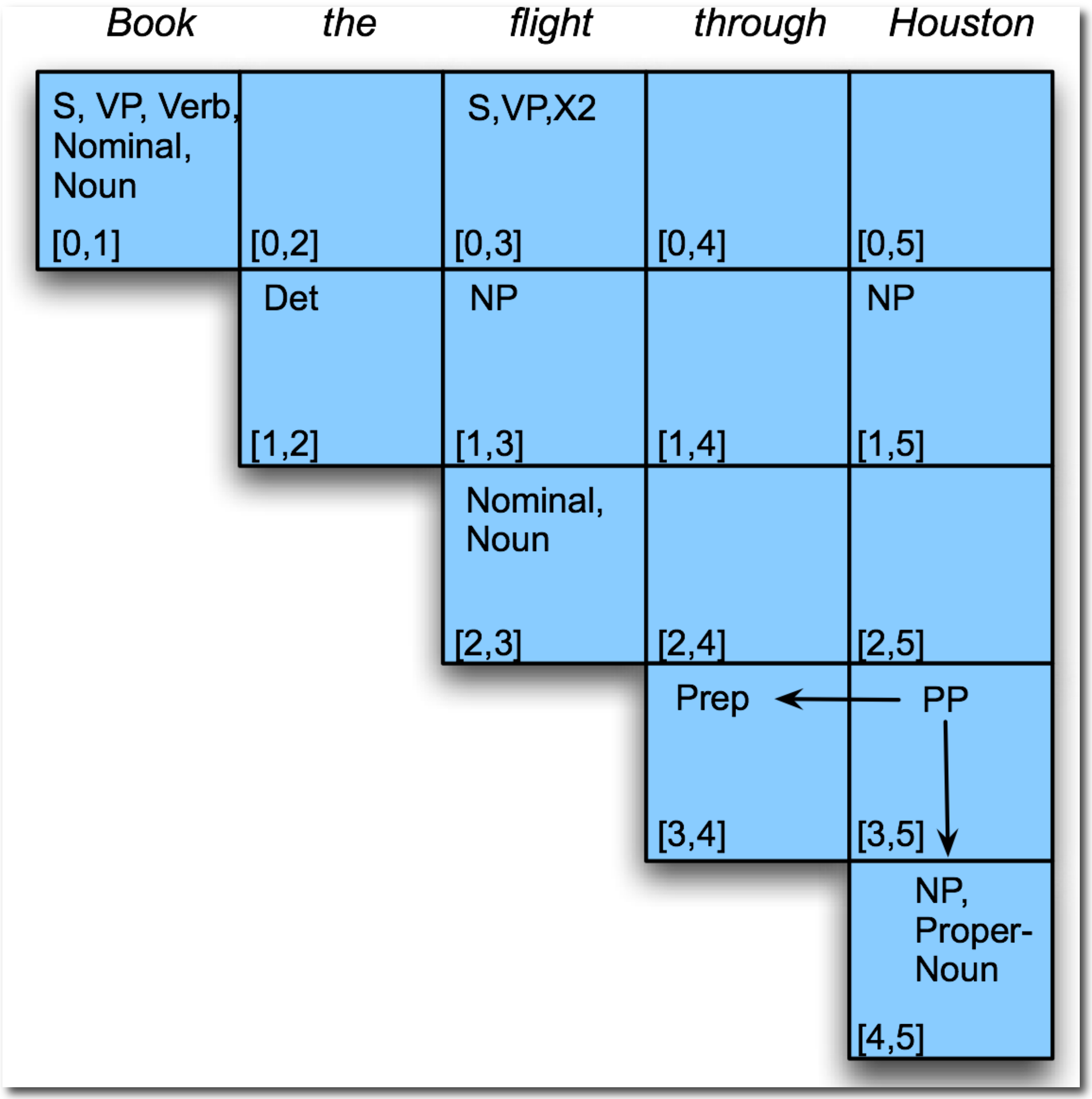
Example

- Filling column 5 corresponds to processing word 5, which is *Houston*.
  - So *j* is 5.
  - So *i* goes from 3 to 0 (3,2,1,0)

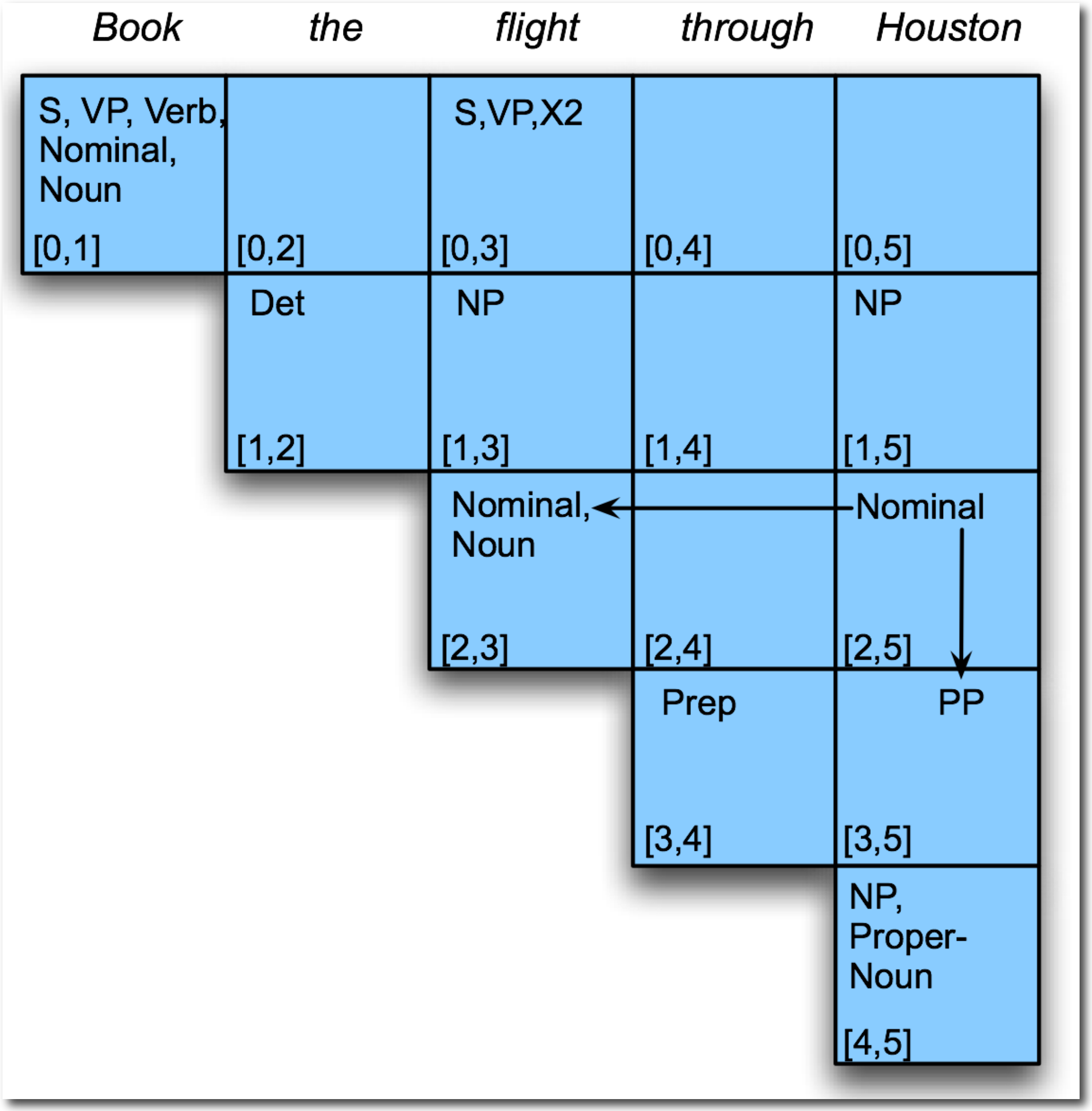
function CKY-PARSE(*words, grammar*) **returns** *table*

  for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
    $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
    for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
      for $k \leftarrow i+1$ **to** $j-1$ **do**
        $table[i,j] \leftarrow table[i,j] \cup$
$$\{A \mid A \rightarrow BC \in grammar, B \in table[i,k], C \in table[k,j]\}$$

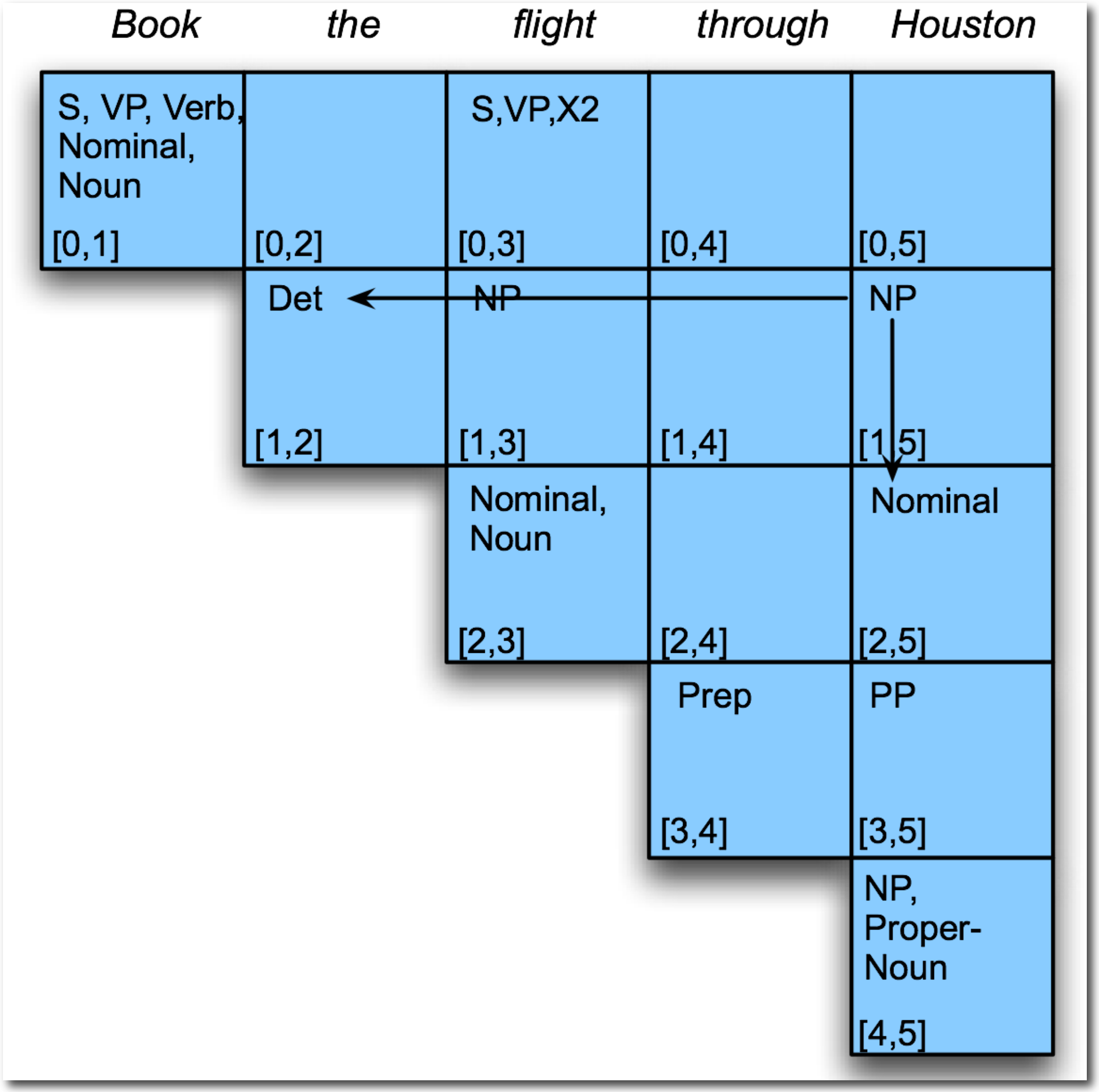
Example


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

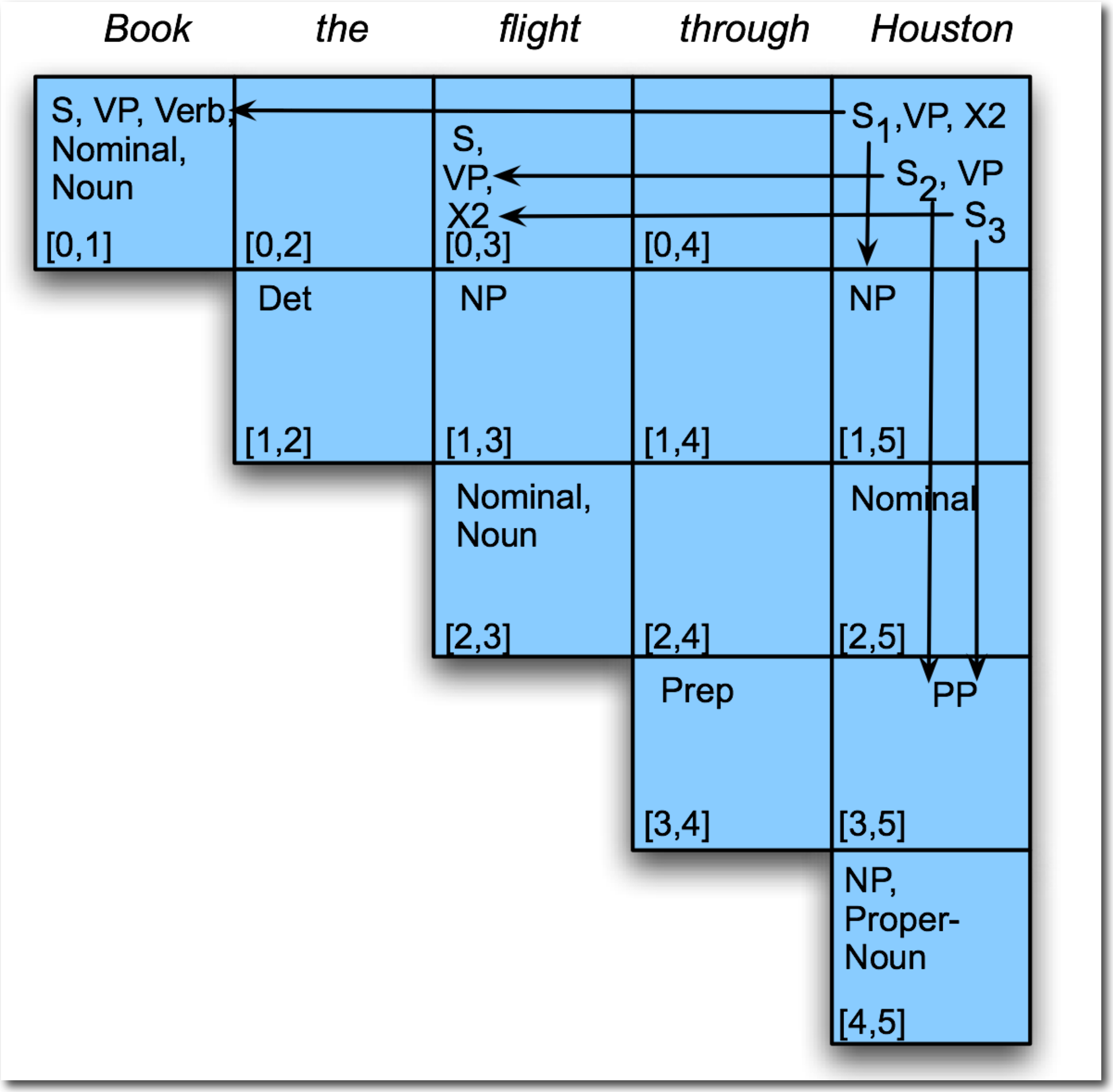
Example


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

Example


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

Example


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S, VP, X2
[0,3]
[0,4]
S1,VP, X2
S2, VP
S3
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

Example

- Since there's an *S* in [0,5] we have a valid parse.
- Are we done? We we sort of left something out of the algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup \{A \mid A \rightarrow BC \in grammar, B \in table[i,k], C \in table[k,j]\}$

CKY Notes

- Since it’s bottom up, CKY hallucinates a lot of silly constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy
  - Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

CKY Notes

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we' re filling a cell, the parts needed to fill it are already in the table (to the left and below)
  - It' s somewhat natural in that it processes the input a left to right a word at a time
    - Known as online
  - Can you think of an alternative strategy?