Elements of Syntax for Parsing

CS 115B

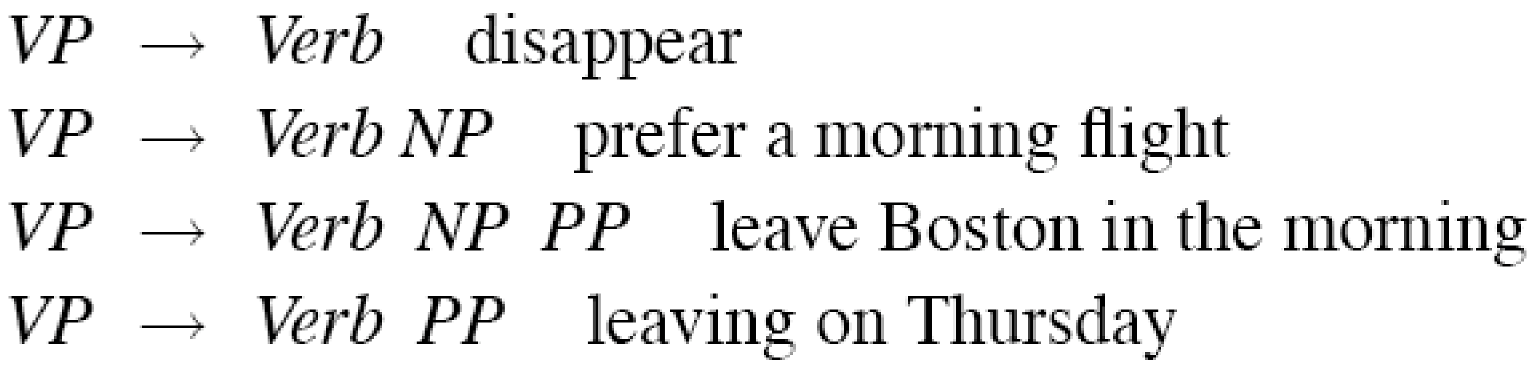
Fundamentals of NLP 1

March 5, 2025

Brandeis University

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.

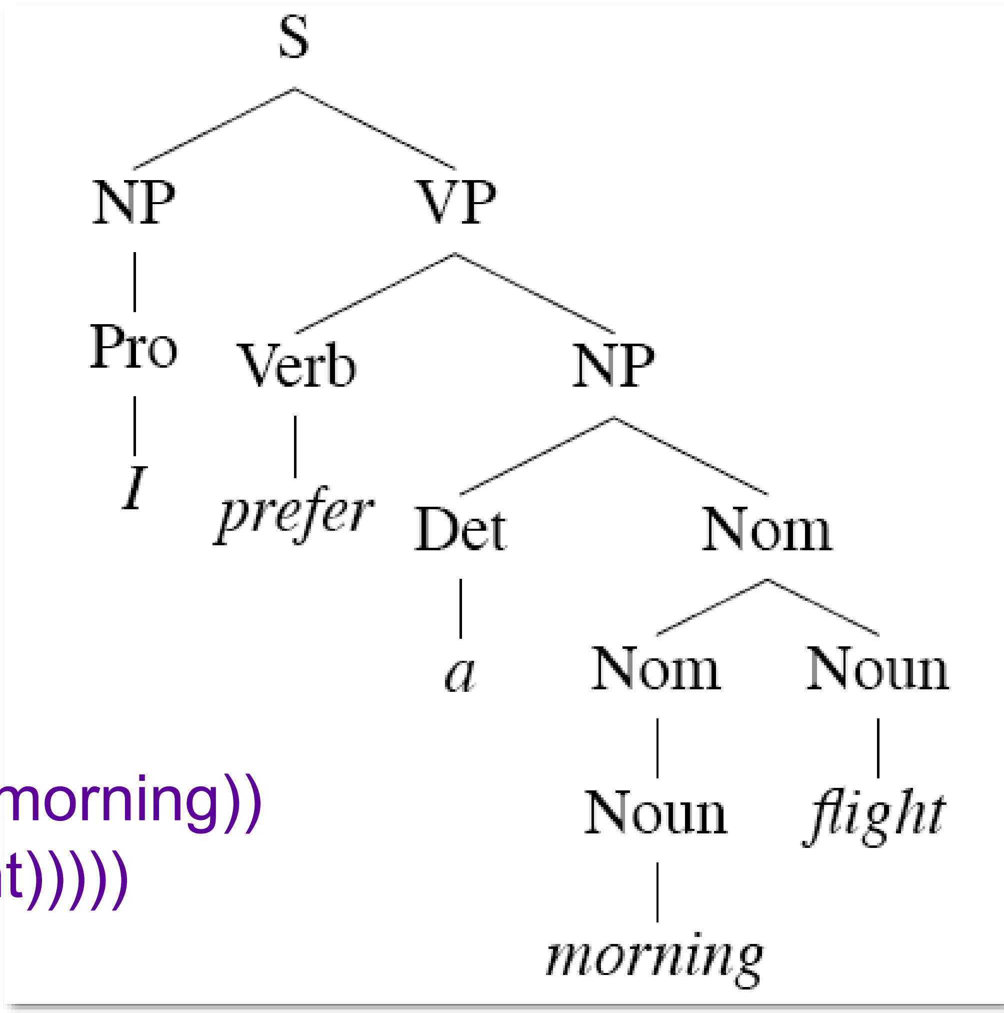
Summary

- 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...

$$
\begin{array}{l}
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP}
\end{array}
$$

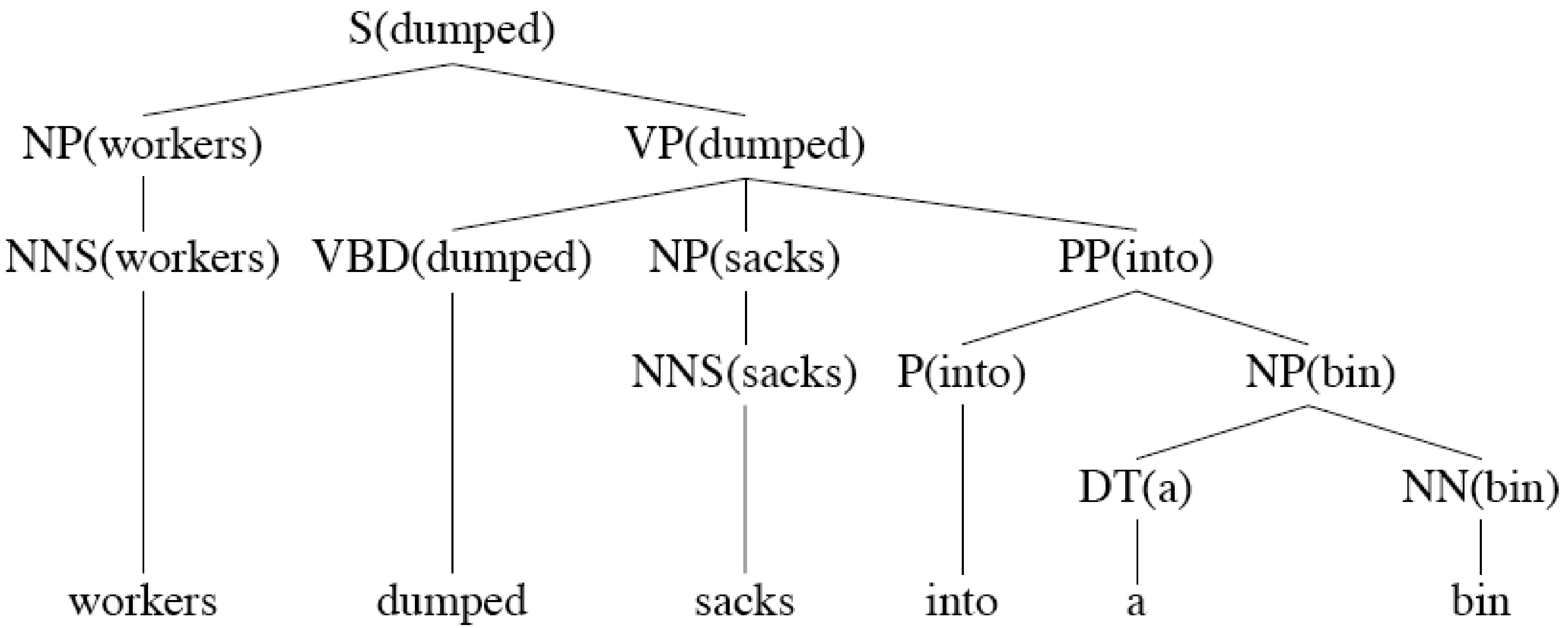
Treebank Uses

- 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.

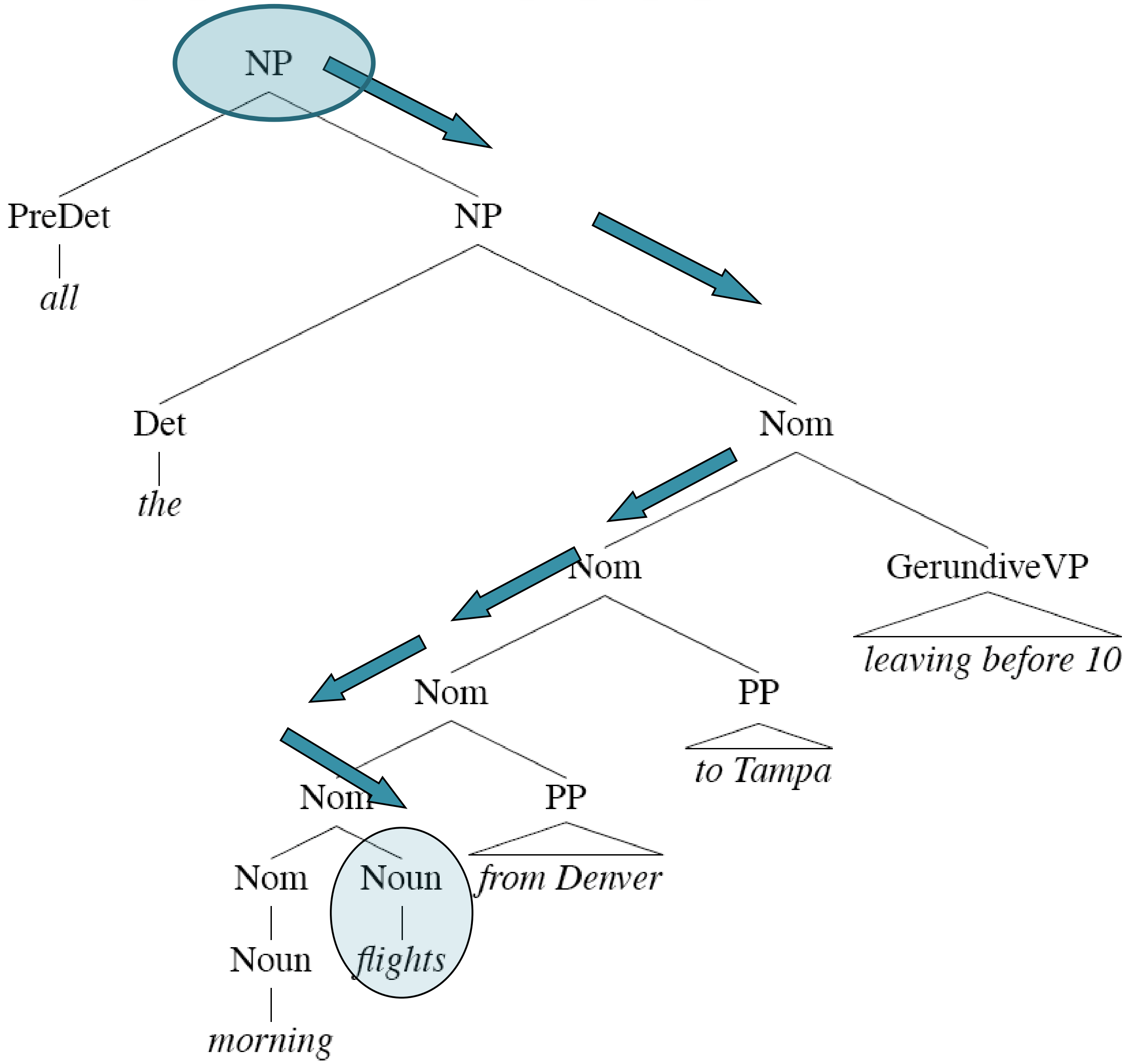
Lexically Decorated Tree



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



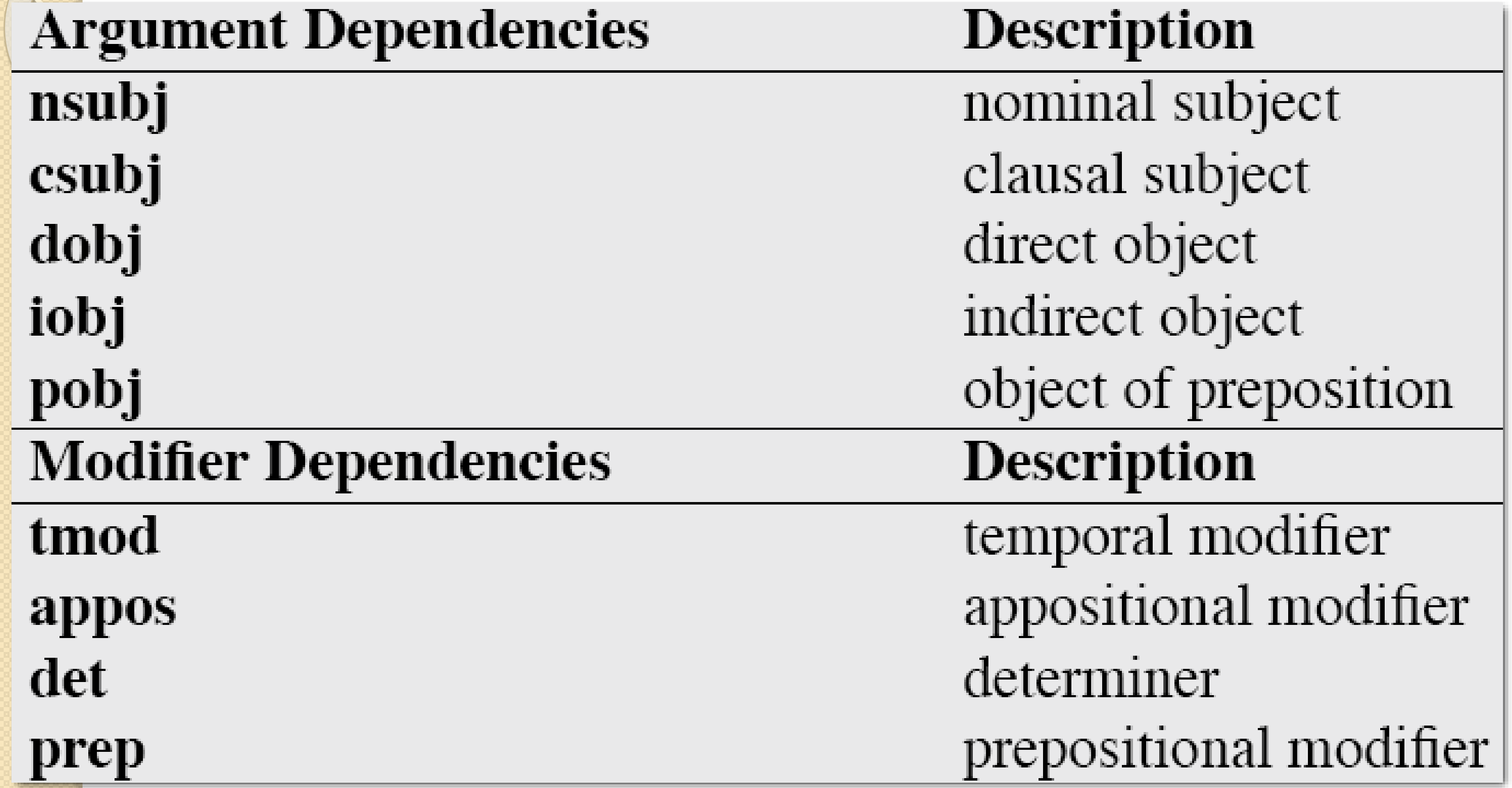
Treebank Uses

- 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

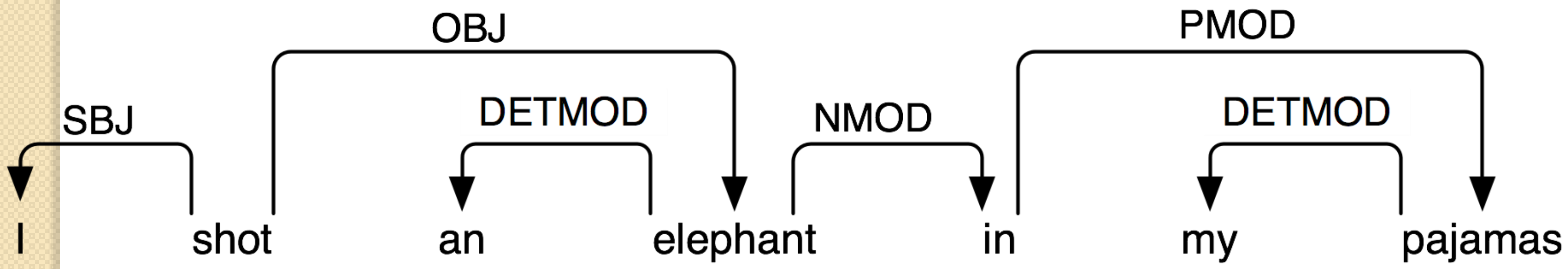
- In CFG-style phrase-structure grammars the main focus is on *constituents* and *ordering*.
- But it turns out you can get a lot done with just labeled relations among the words in an utterance.
- In a dependency grammar framework, a parse is a tree where
  - The nodes stand for the words in an utterance
  - 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

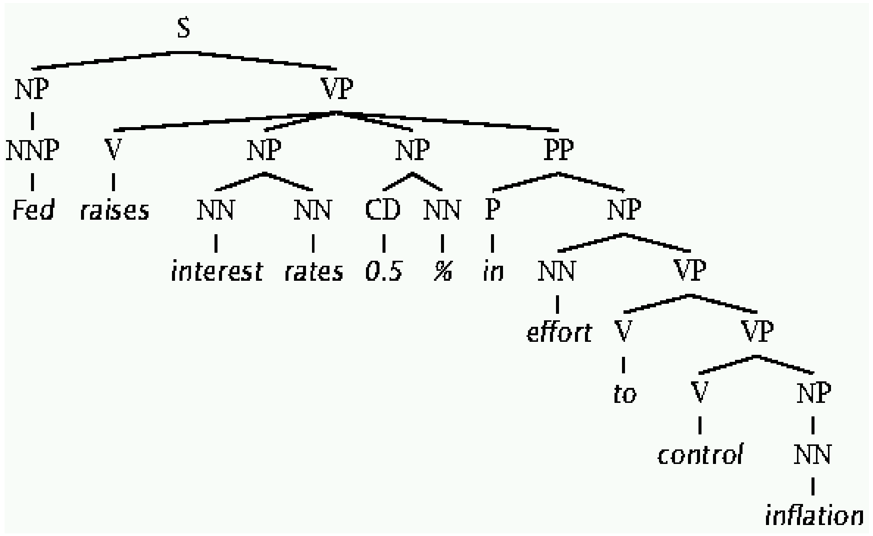
- The dependency approach has a number of advantages over full phrase-structure parsing.
  - It deals well with free word order languages where the constituent structure is quite fluid
  - Parsing is *much faster* than with CFG-based parsers
  - Dependency structure often captures the syntactic relations 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.

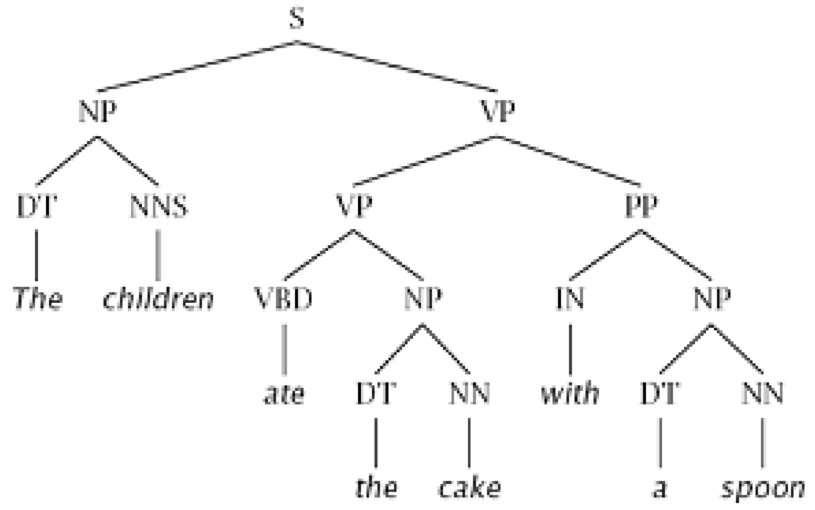
1. Phrase structure

- Phrase structure trees organize sentences into *constituents* or *brackets.*
- Each constituent gets a label.
- The constituents are nested in a tree form.
- Linguists can and do argue about the details.
- Lots of ambiguity …



Constituency Tests

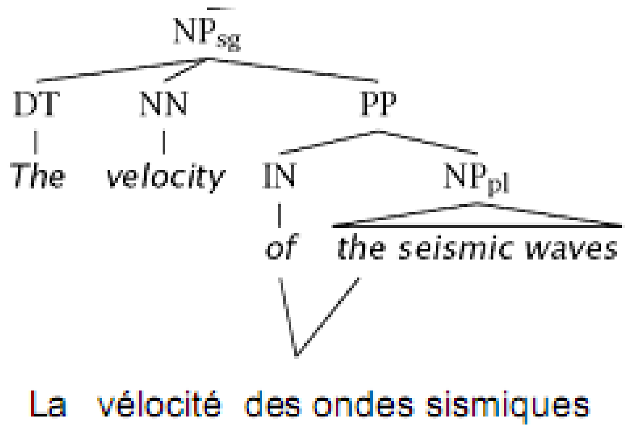
- How do we know what nodes go in the tree?
- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic grounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction
- Cross-linguistic arguments



Conflicting Tests

Constituency isn't always clear.

- Phonological Reduction:
  - I will go → I'll go
  - I want to go → I wanna go
  - a le centre → au centre



- Coordination
  - He went to and came from the store.

Classical NLP: Parsing

- Write symbolic or logical rules:

Grammar (CFG)		Lexicon
ROOT → S	NP → NP PP	NN → interest
S → NP VP	VP → VBP NP	NNS → raises
NP → DT NN	VP → VBP NP PP	VBP → interest
NP → NN NNS	PP → IN NP	VBZ → raises
		...

- Use deduction systems to prove parses from words
  - Minimal grammar on "Fed" sentence: 36 parses
  - Simple, 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses
  - With hand-built grammar, ~30% of sentences have no parse
- This scales very badly.
  - Hard to produce enough rules for every variation of language (coverage)
  - Many, many parses for each valid sentence (disambiguation)

Ambiguity examples

Part of speech ambiguities

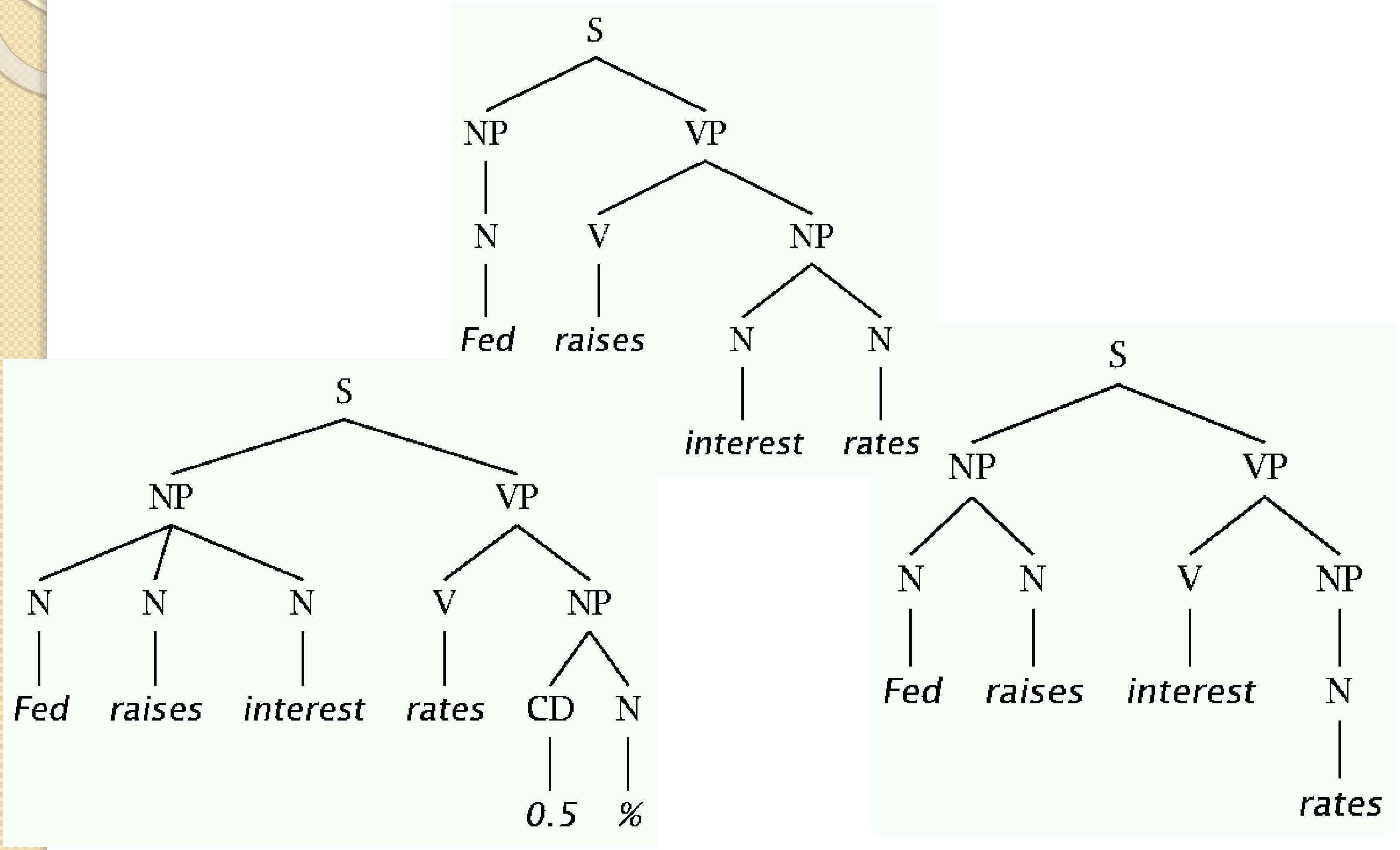
		VB				
	VBZ	VBP	VBZ			
NNP	NNS	NN	NNS	CD	NN	
Fed	*raises*	*interest*	*rates*	*0.5*	*%*	*in effort to control inflation*

Syntactic attachment ambiguities

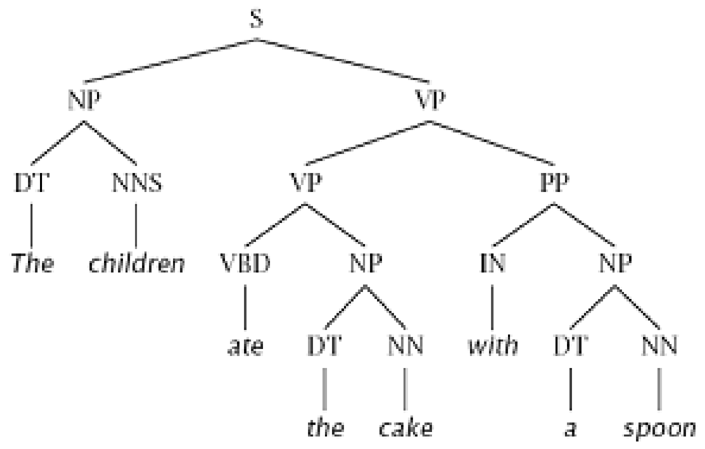
Word sense ambiguities: Fed → *"federal agent"*
interest → *a feeling of wanting to know or learn more*

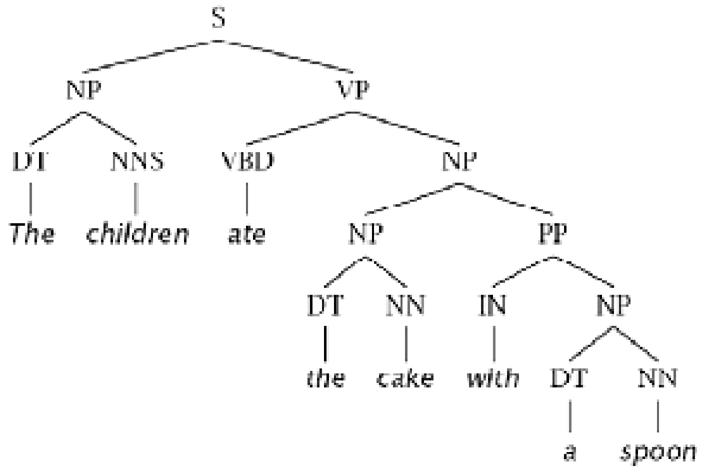
Semantic interpretation ambiguities above the word level

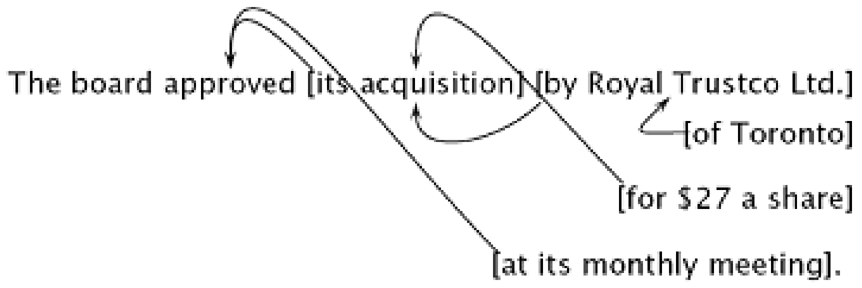
The bad effects of V/N ambiguities



Ambiguities: PP Attachment






The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for $27 a share]
[at its monthly meeting].

Attachments

- I cleaned the dishes from dinner.
- I cleaned the dishes with detergent.
- I cleaned the dishes in my pajamas.
- I cleaned the dishes in the sink.

Syntactic Ambiguities 1

- Prepositional Phrases

  They cooked the beans in the pot on the stove with handles.

- Particle vs. Preposition

  The puppy tore up the staircase.

- Complement Structure

  The tourists objected to the guide that they couldn' t hear.

  She knows you like the back of her hand.

- Gerund vs. Participial Adjective

  Visiting relatives can be boring.

  Changing schedules frequently confused passengers.

Syntactic Ambiguities 2

- Modifier scope within NPs

  impractical design requirements

  plastic cup holder

- Multiple gap constructions

  The chicken is ready to eat.

  The contractors are rich enough to sue.

- Coordination scope

  Small rats and mice can squeeze into holes or cracks in the wall.

Classical NLP Parsing: The problem and its solution

- Very constrained grammars attempt to limit unlikely/weird parses for sentences
  - But the attempt makes the grammars not robust: many sentences have no parse
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses
- Solution: We need mechanisms that allow us to find the <u>*most likely*</u> parse(s)
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but to still quickly find the best parse(s)

Polynomial-time Parsing with Context Free Grammars

Parsing

Computational task:

Given a set of grammar rules and a sentence, find a valid parse of the sentence (efficiently)

Naively, you could try all possible trees until you get to a parse tree that conforms to the grammar rules, that has "S" at the root, and that has the right words at the leaves.

But that takes exponential time in the number of words.

Aspects of parsing

- Running a grammar backwards to find possible structures for a sentence
- Parsing can be viewed as a search problem
- Parsing is a hidden data problem
- For the moment, we want to examine *all* structures for a string of words
- We can do this bottom-up or top-down
    - This distinction is independent of depth-first or breadth-first search – we can do either both ways
    - We search by building a *search tree* which his distinct from the *parse tree*

Human parsing

- Humans often do ambiguity maintenance
  - *Have the police … eaten their supper?*
  - *come in and look around.*
  - *taken out and shot.*

- But humans also commit early and are "garden pathed" :
  - *The man who hunts ducks out on weekends.*
  - *The cotton shirts are made from grows in Mississippi.*
  - *The horse raced past the barn fell.*

A phrase structure grammar

- S → NP VP
- VP → V NP
- VP → V NP PP
- NP → NP PP
- NP → N
- NP → e
- NP → N N
- PP → P NP

N → cats
N → claws
N → people
N → scratch
V → scratch
P → with

- By convention, S is the start symbol, but in the PTB, we have an extra node at the top (ROOT, TOP)

Phrase structure grammars = context-free grammars

- G = (T, N, S, R)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
- A grammar G generates a language L.

Probabilistic or stochastic context-free grammars (PCFGs)

- G = (T, N, S, R, P)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
    - P(R) gives the probability of each rule.

$$\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

- A grammar G generates a language model L.

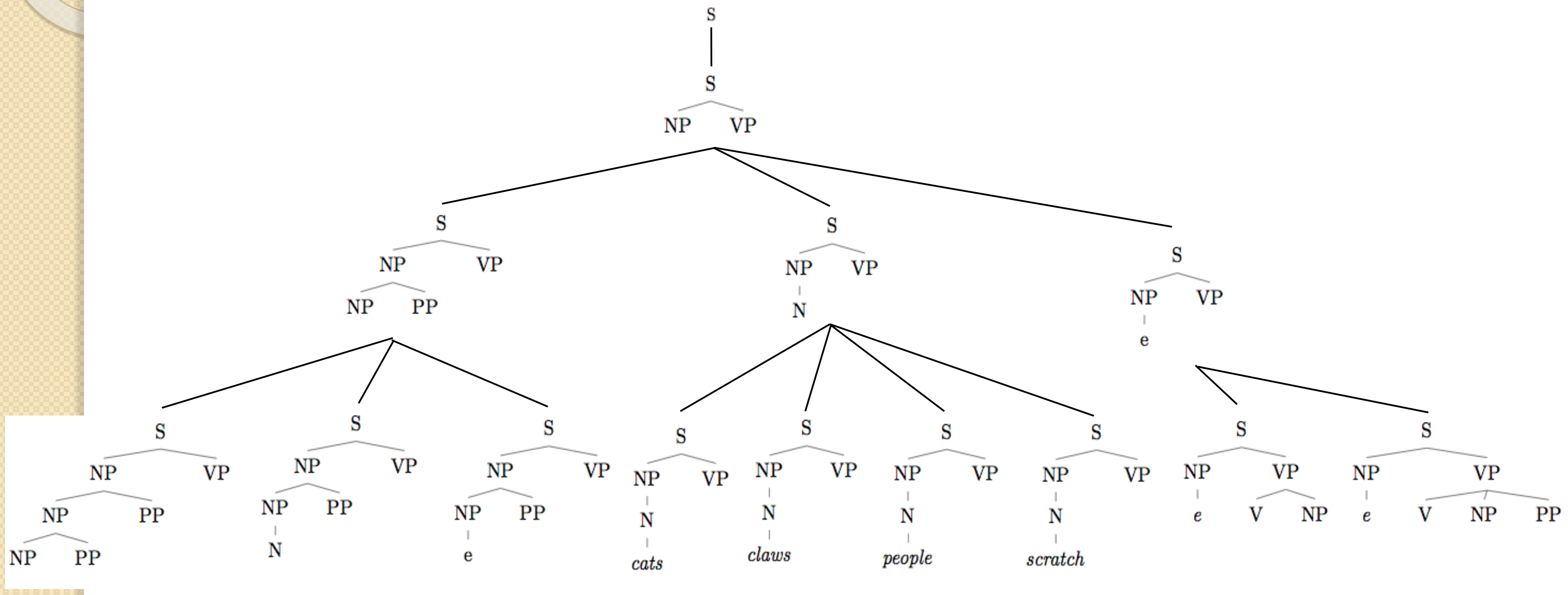
Soundness and completeness

- A parser is *sound* if every parse it returns is valid/correct
- A parser *terminates* if it is guaranteed to not go off into an infinite loop
- A parser is *complete* if for any given grammar and sentence, it is sound, produces every valid parse for that sentence, and terminates
- (For many purposes, we settle for sound but incomplete parsers: e.g., probabilistic parsers that return a *k*-best list.)

Top-down parsing

- Top-down parsing is goal directed
- A top-down parser starts with a list of constituents to be built. The top-down parser rewrites the goals in the goal list by matching one against the LHS of the grammar rules, and expanding it with the RHS, attempting to match the sentence to be derived.
- If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering.

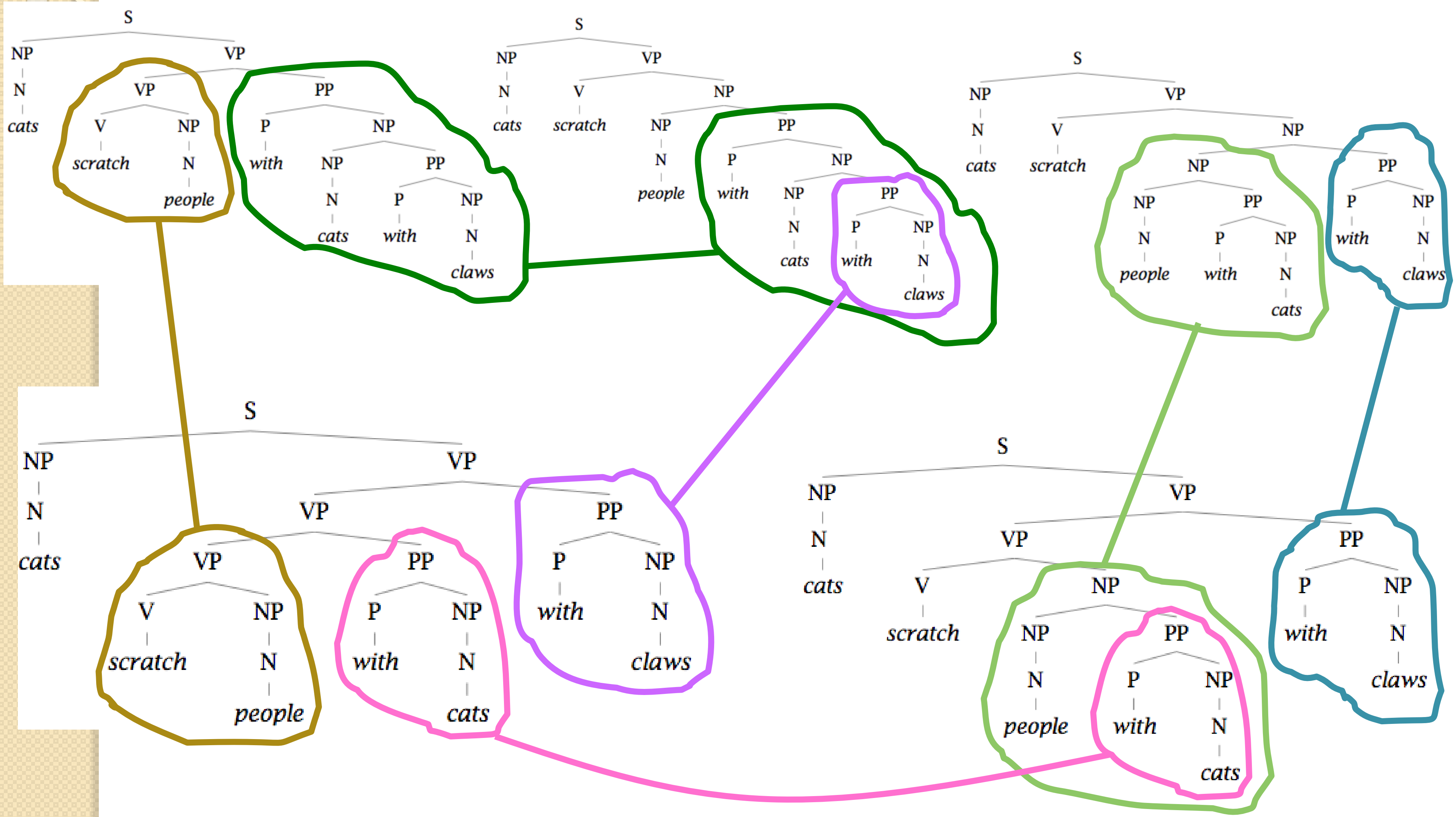
Top-down parsing



Problems with top-down parsing

- Left recursive rules
- A top-down parser will do badly if there are many different rules for the same LHS. Consider if there are 600 rules for S, 599 of which start with NP, but one of which starts with V, and the sentence starts with V.
- Useless work: expands things that are possible top-down but not there
- Top-down parsers do well if there is useful grammar-driven control: search is directed by the grammar
- Top-down is hopeless for rewriting parts of speech (preterminals) with words (terminals). In practice that is always done bottom-up as lexical lookup.
- Repeated work: anywhere there is common substructure

Repeated work…



Bottom-up parsing

- Bottom-up parsing is data directed
- The initial goal list of a bottom-up parser is the string to be parsed. If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
- Parsing is finished when the goal list contains just the start category.
- If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering.
- The standard presentation is as *shift-reduce parsing*.

Problems with bottom-up parsing

- Unable to deal with empty categories: termination problem, unless rewriting empties as constituents is somehow restricted (but then it's generally incomplete)
- Useless work: locally possible, but globally impossible.
- Inefficient when there is great lexical ambiguity (grammar-driven control might help here)
- Conversely, it is data-directed: it attempts to parse the words that are there.
- Repeated work: anywhere there is common substructure

Chomsky Normal Form

- All rules are of the form X → Y Z or X → w.
- A transformation to this form doesn' t change the weak generative capacity of CFGs.
  - With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
  - Unaries/empties are removed recursively
  - N-ary rules introduce new nonterminals:
    - VP → V NP PP becomes VP → V @VP-V and @VP-V → NP PP
- In practice it' s a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries can be trickier
- But it makes parsing easier/more efficient

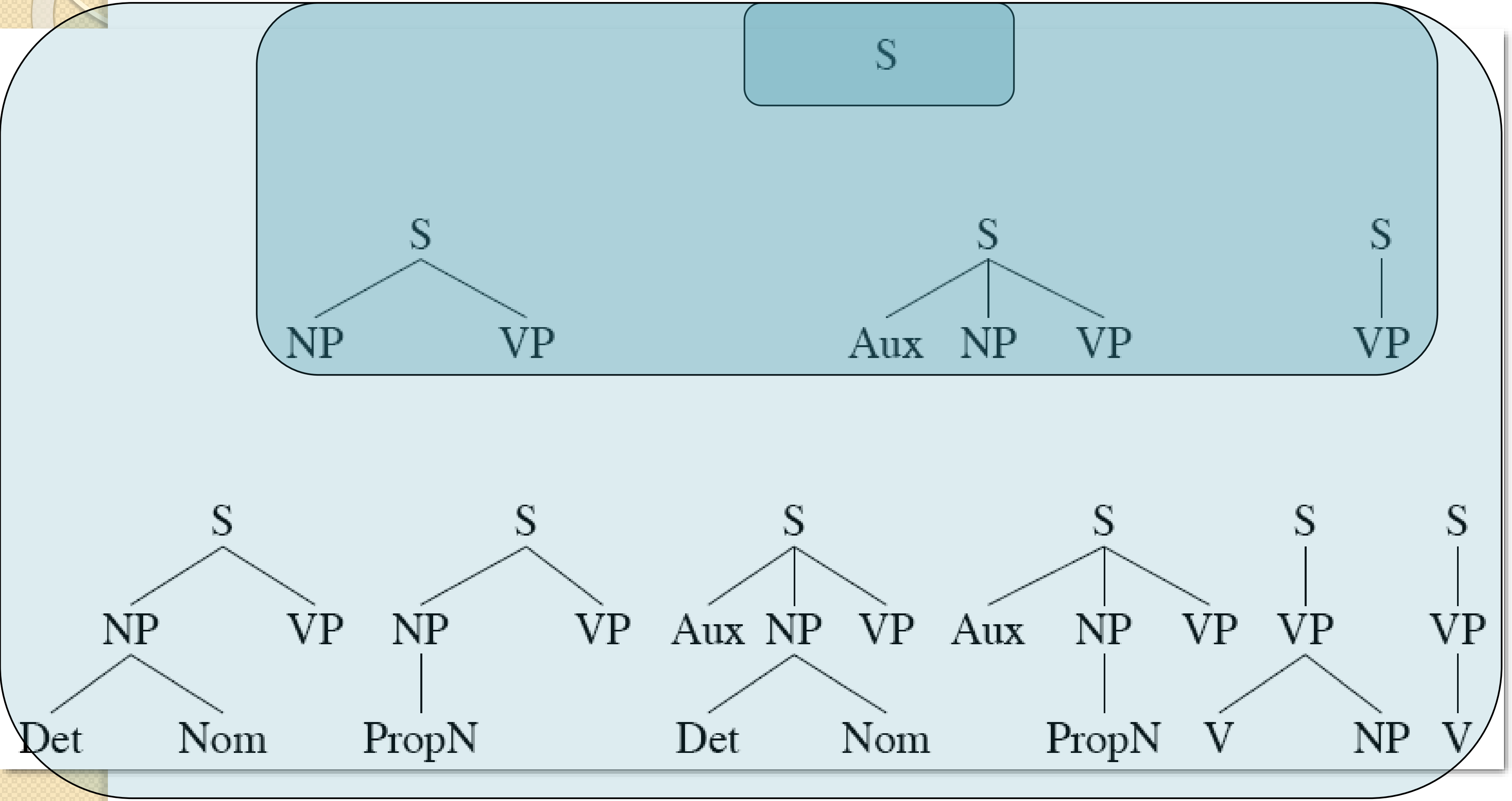
For Now

- 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



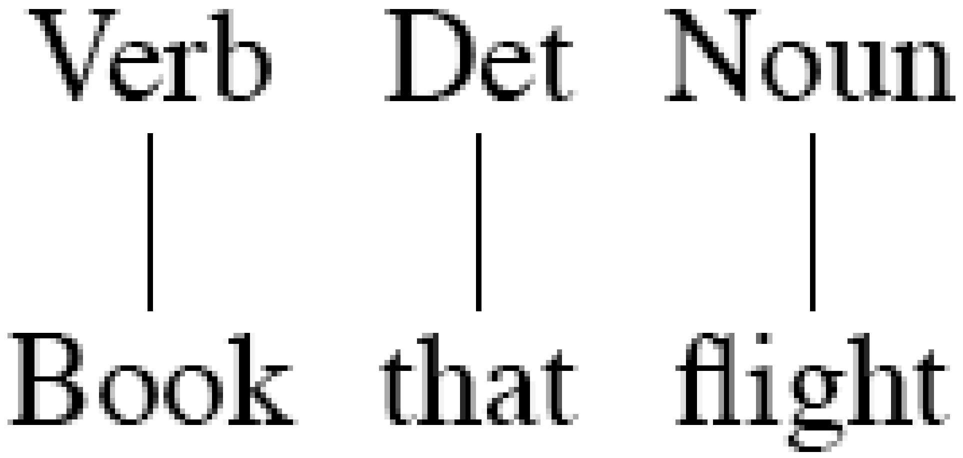
Bottom-Up Parsing

- 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.

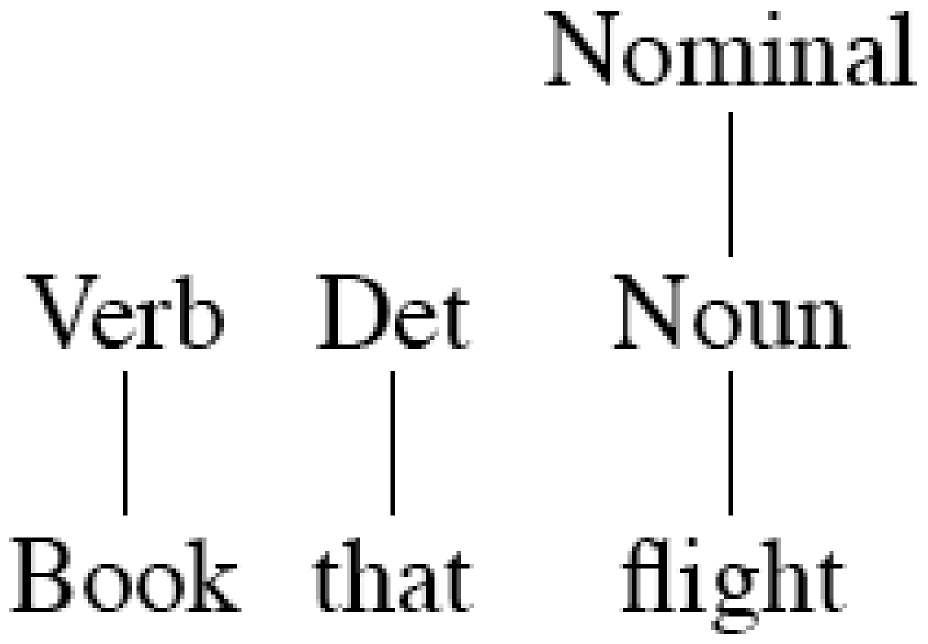
Bottom-Up Search

Book that flight

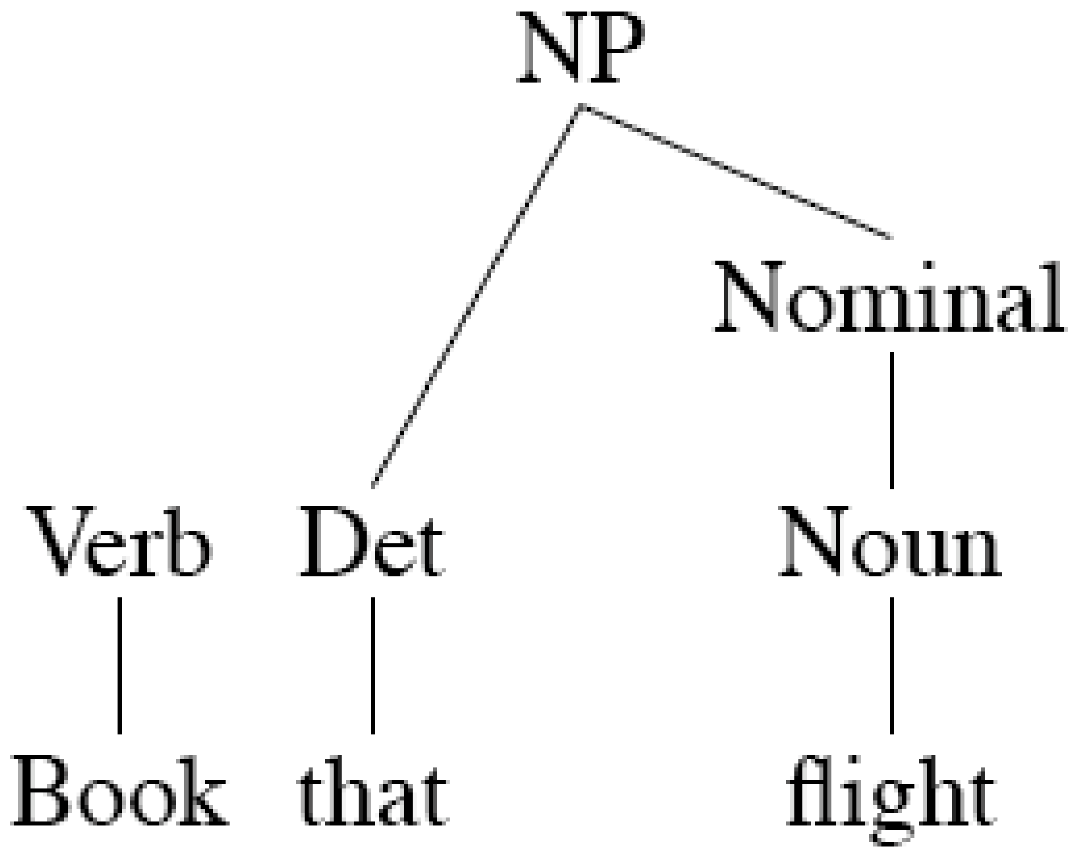
Bottom-Up Search



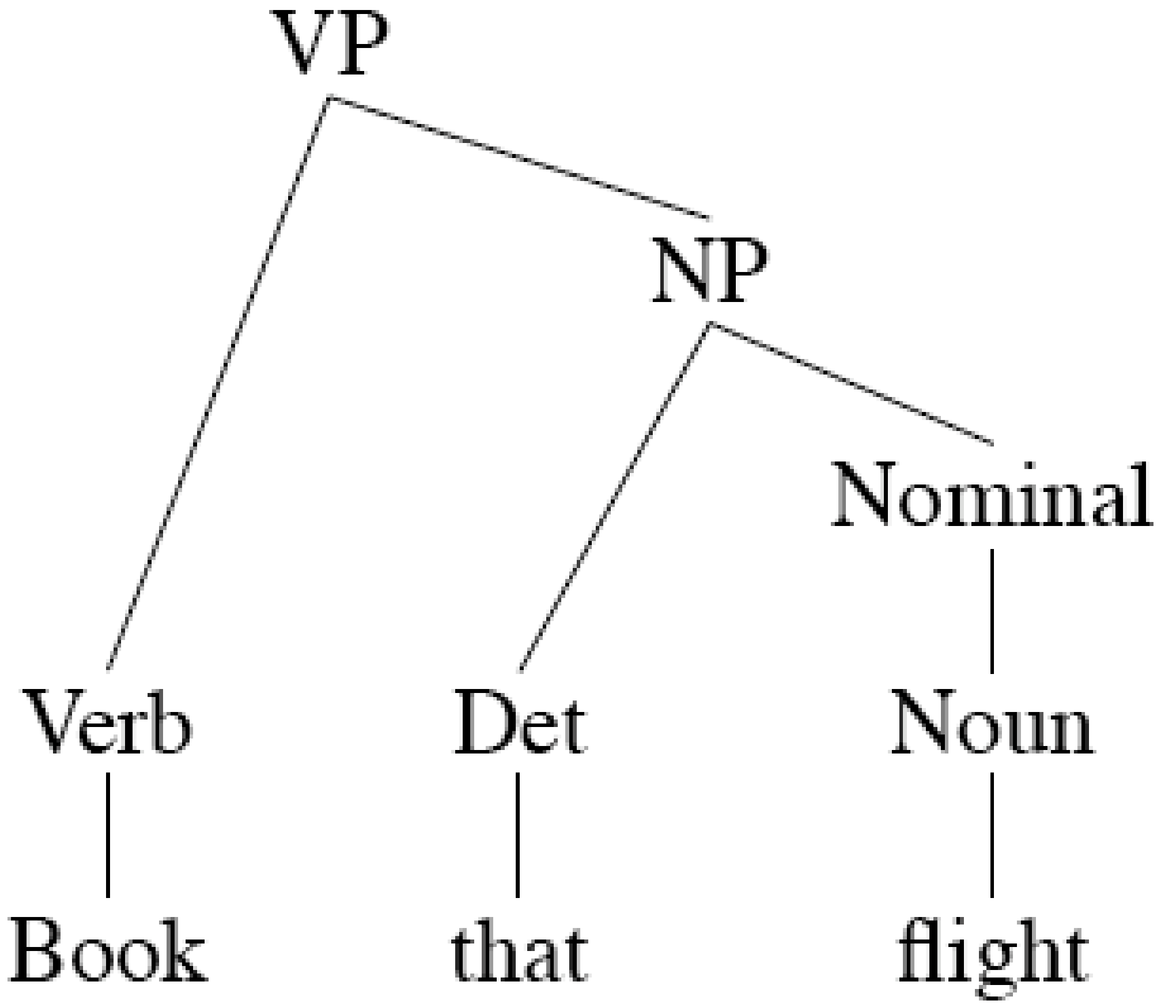
Bottom-Up Search



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

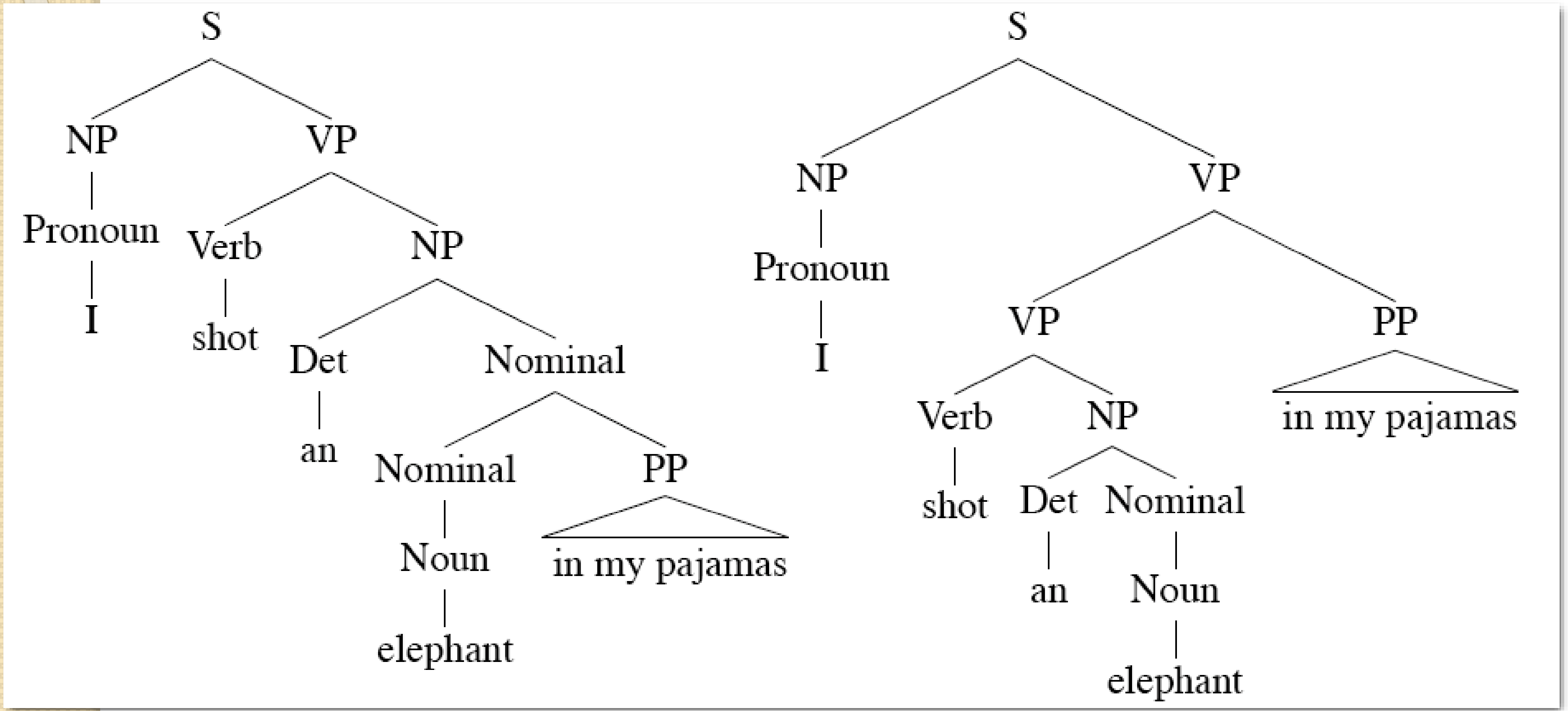
Control

- 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

Problems

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

Ambiguity

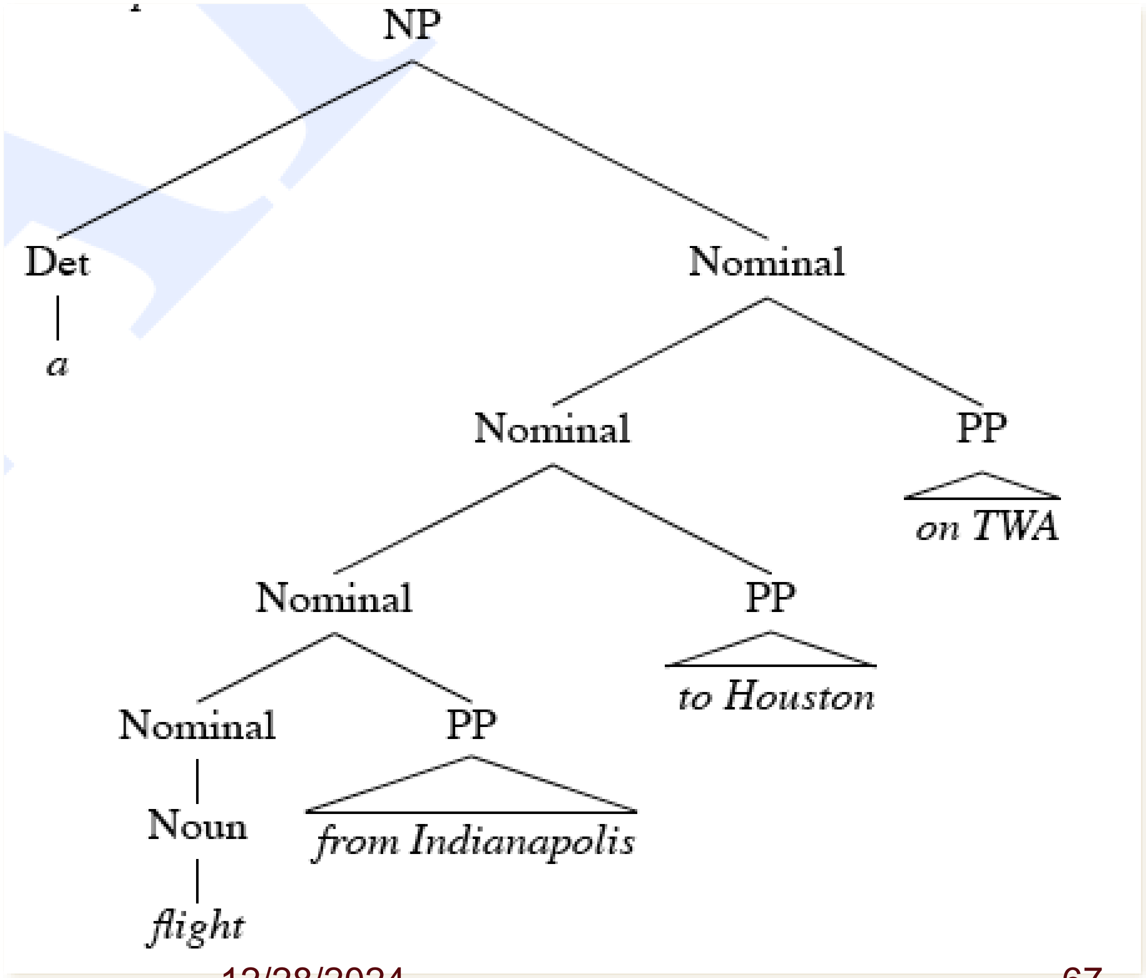

S
NP
VP
Pronoun
I
Verb
shot
NP
Det
an
Nominal
Nominal
Noun
elephant
PP
in my pajamas
S
NP
Pronoun
I
VP
VP
Verb
shot
NP
Det
an
Nominal
Noun
elephant
PP
in my pajamas

Shared Sub-Problems

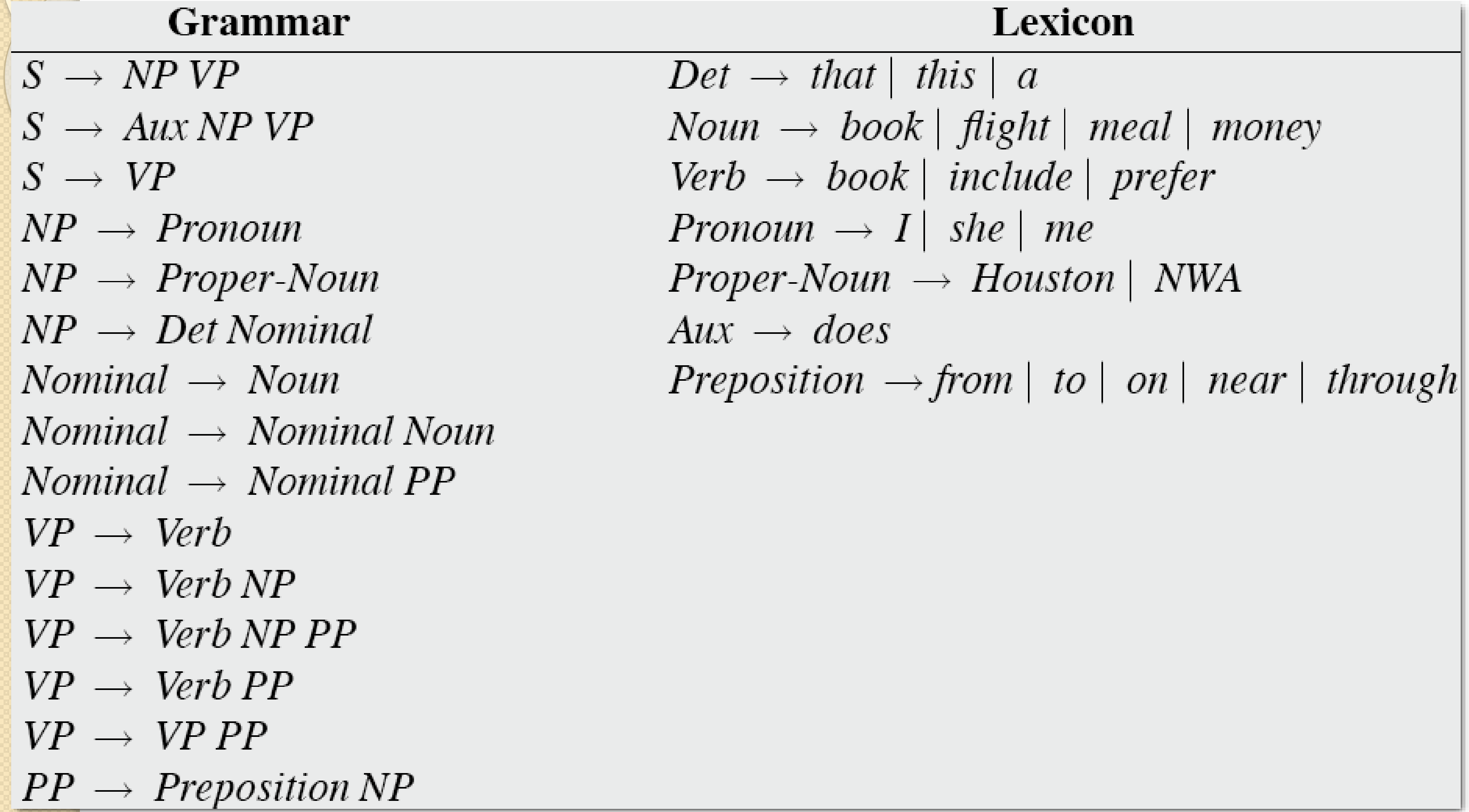
- No matter what kind of search (top-down or bottom-up or mixed) that we choose...
  - We can't afford to redo work we've already done.
  - Without some help naïve backtracking will lead to such duplicated work.

Shared Sub-Problems

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



Sample L1 Grammar

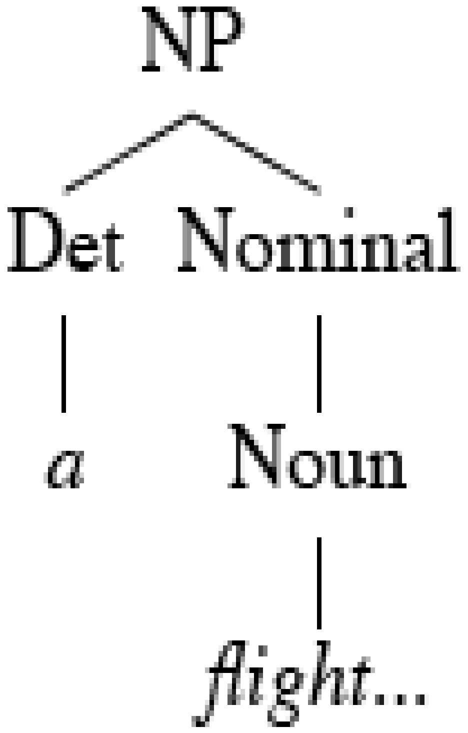


Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP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

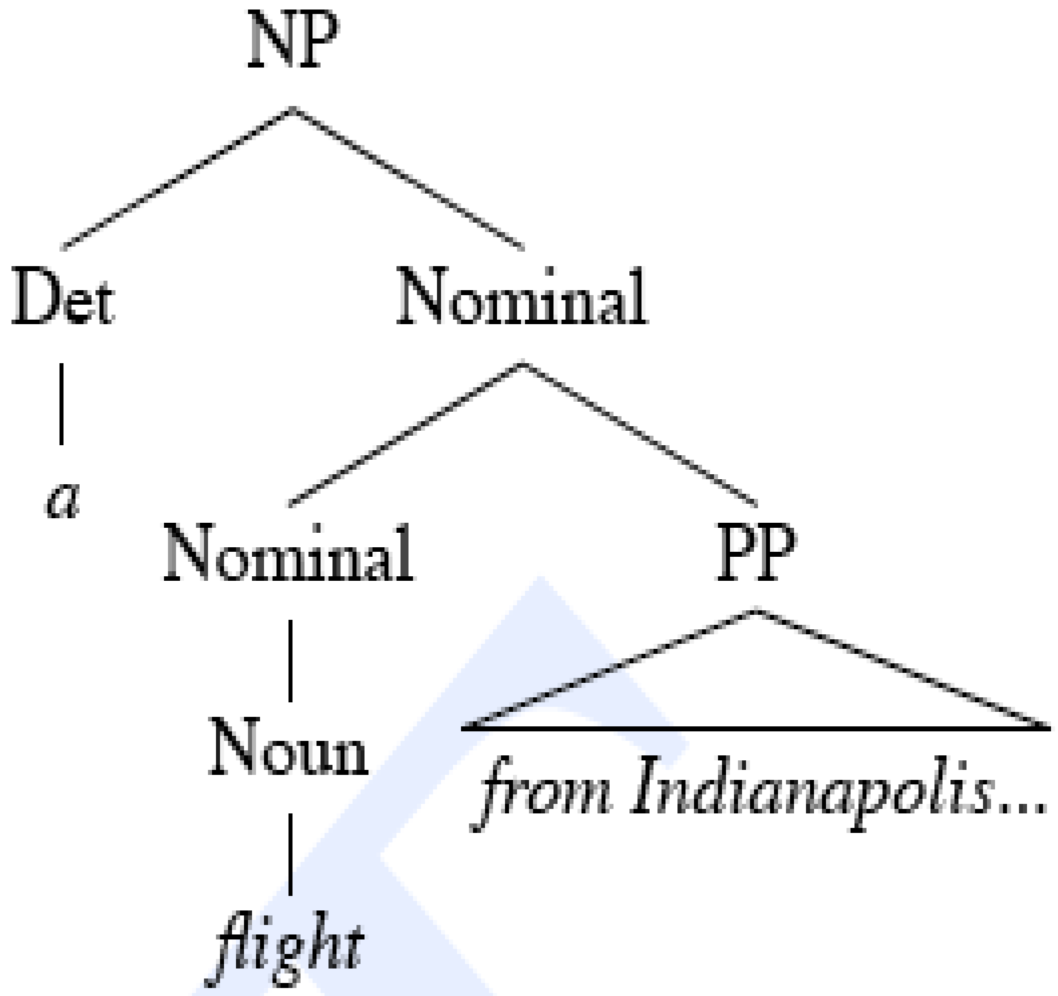
Shared Sub-Problems

- 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...

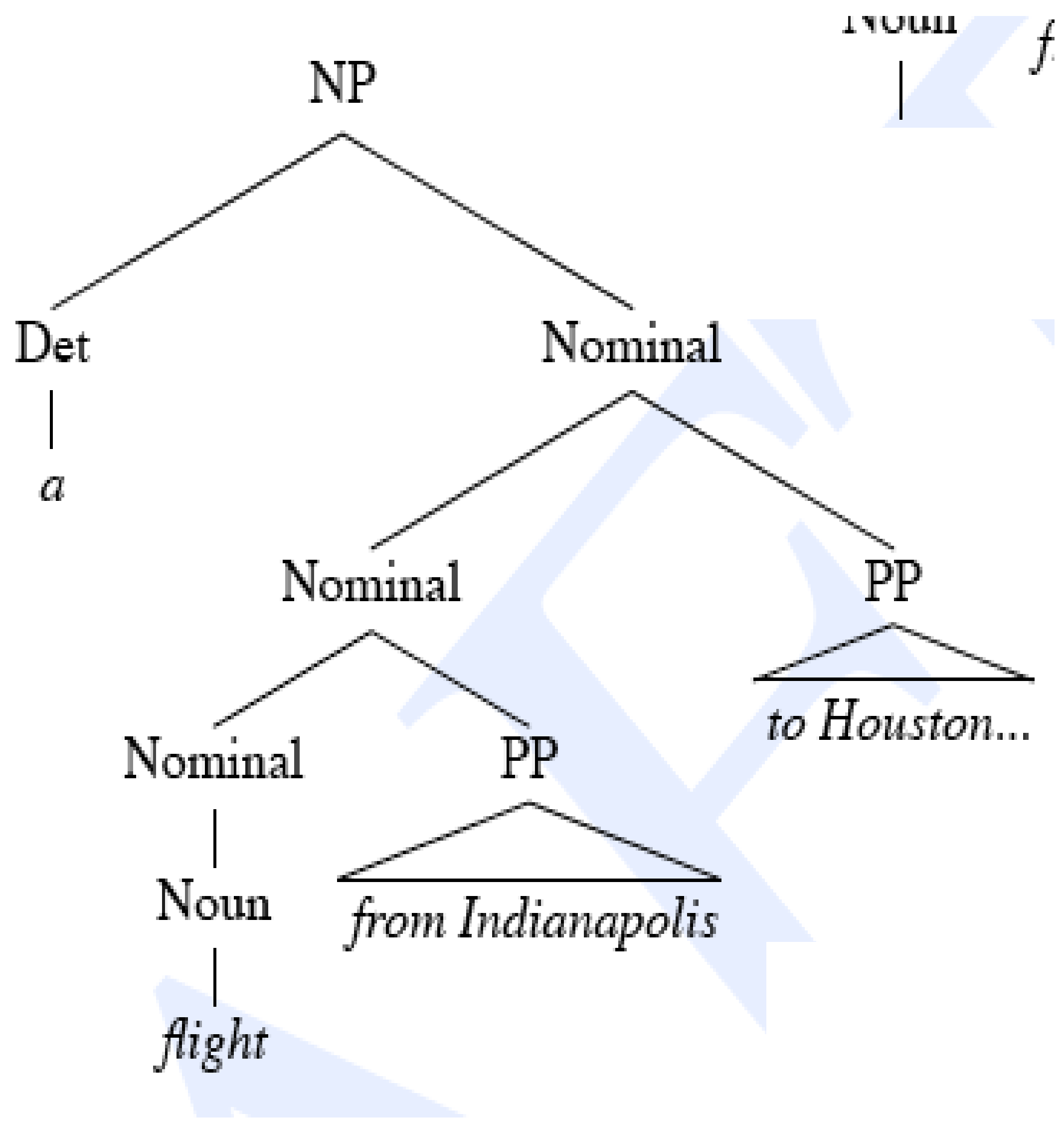
Shared Sub-Problems



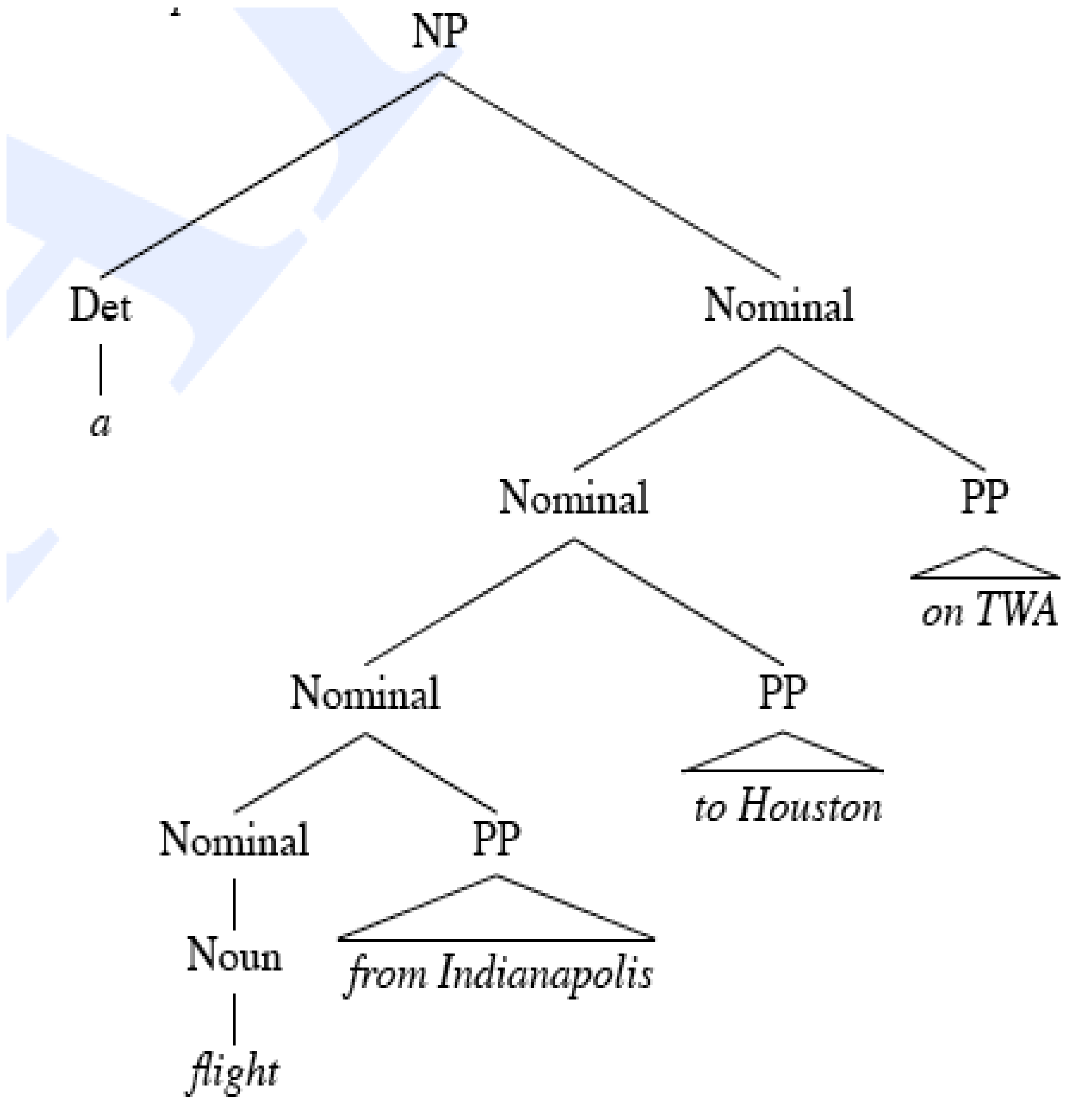
Shared Sub-Problems



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

- 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 $A \rightarrow B\ C$ 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

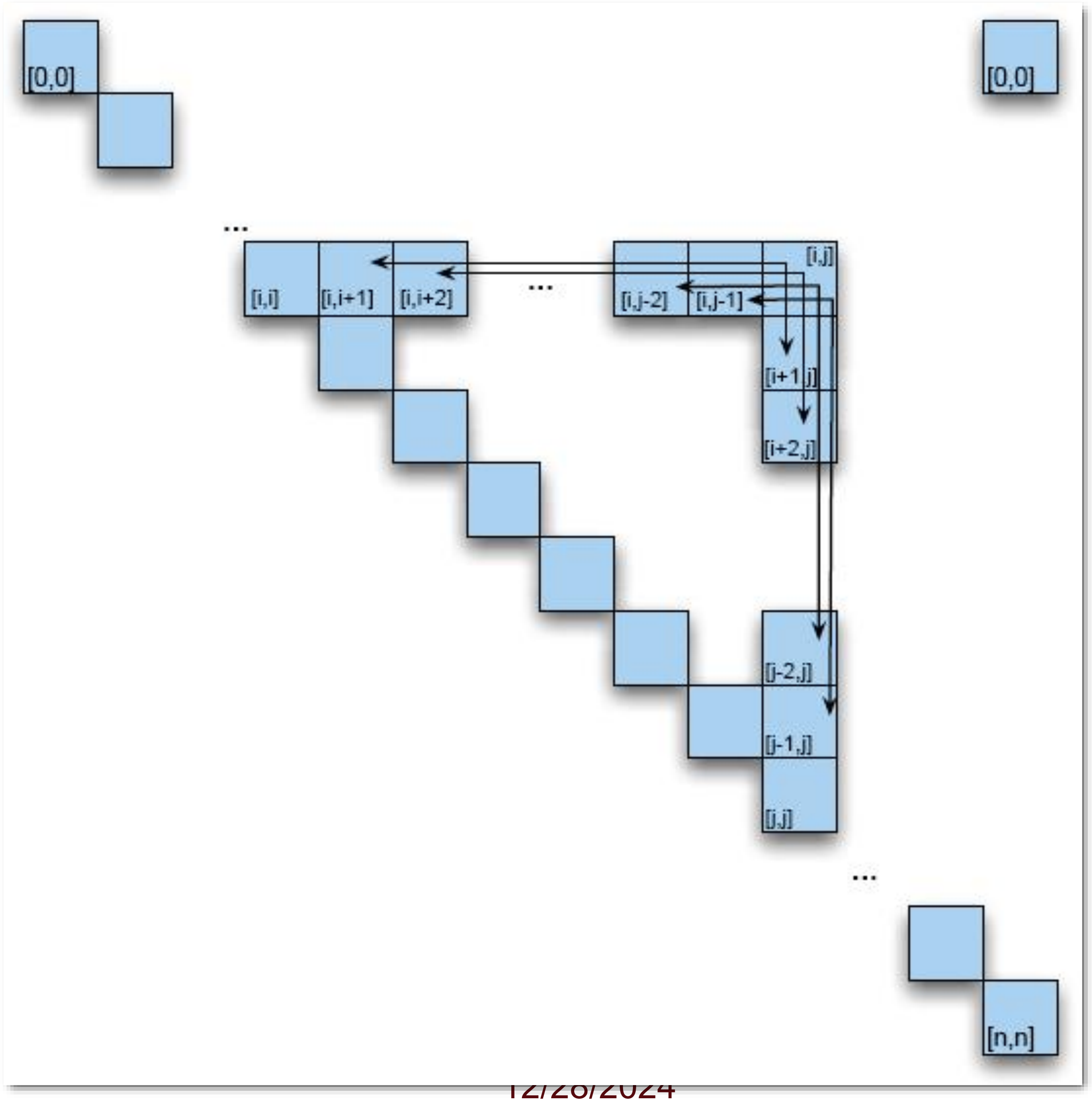
  $A \rightarrow B\ C$ is a rule in the grammar THEN
- There must be a B in [i,k] and a C in [k,j] for some k such that i<k<j

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

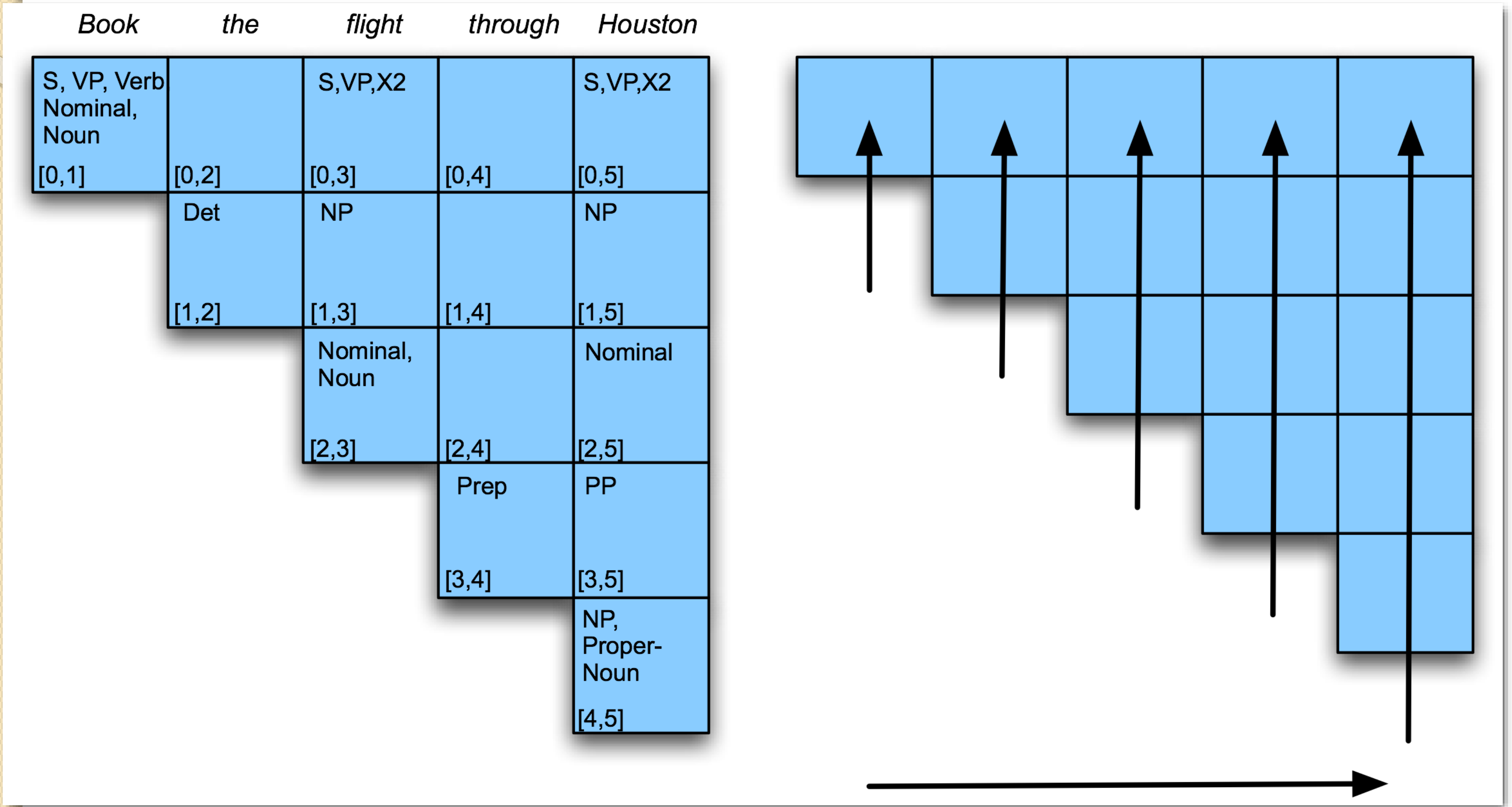


CKY Algorithm

```
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
              $table[i,j] \leftarrow table[i,j] \cup$
                          $\{A \mid A \rightarrow BC \in grammar,$
                              $B \in table[i,k],$
                              $C \in table[k,j]\}$
```

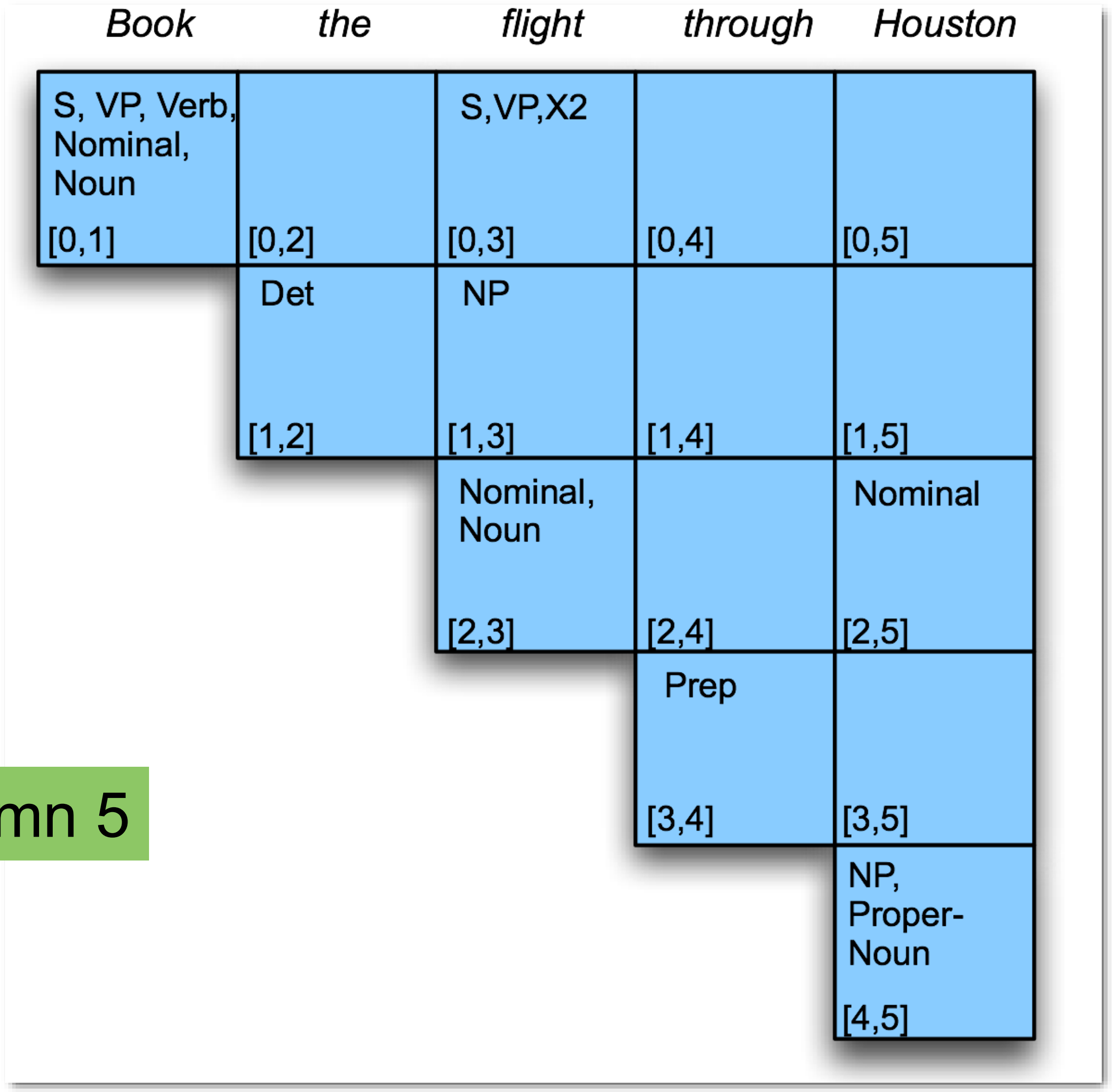
What’s the complexity of this?

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

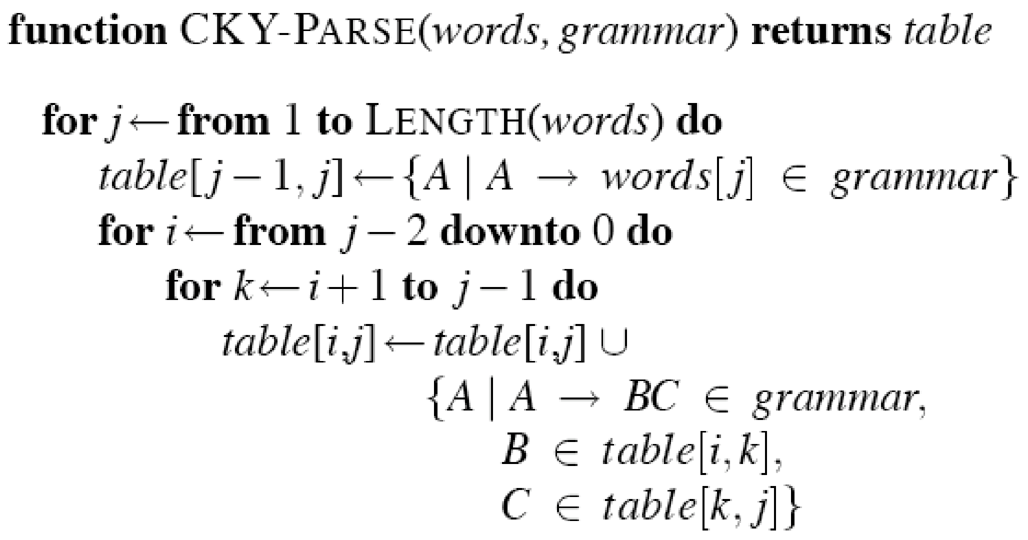


Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	[3,5]
				NP, Proper-Noun [4,5]

Filling column 5

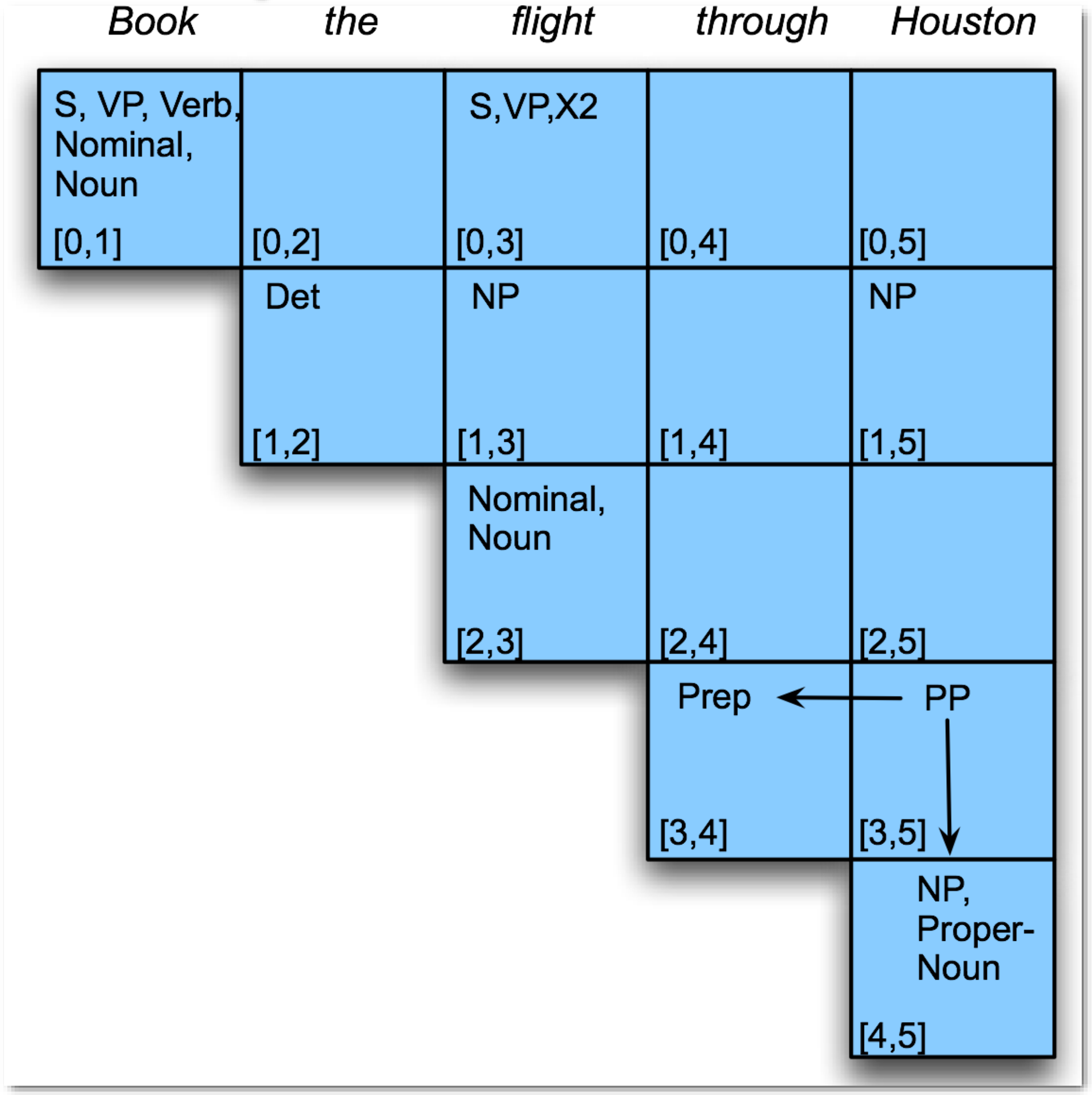
Example

- 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)



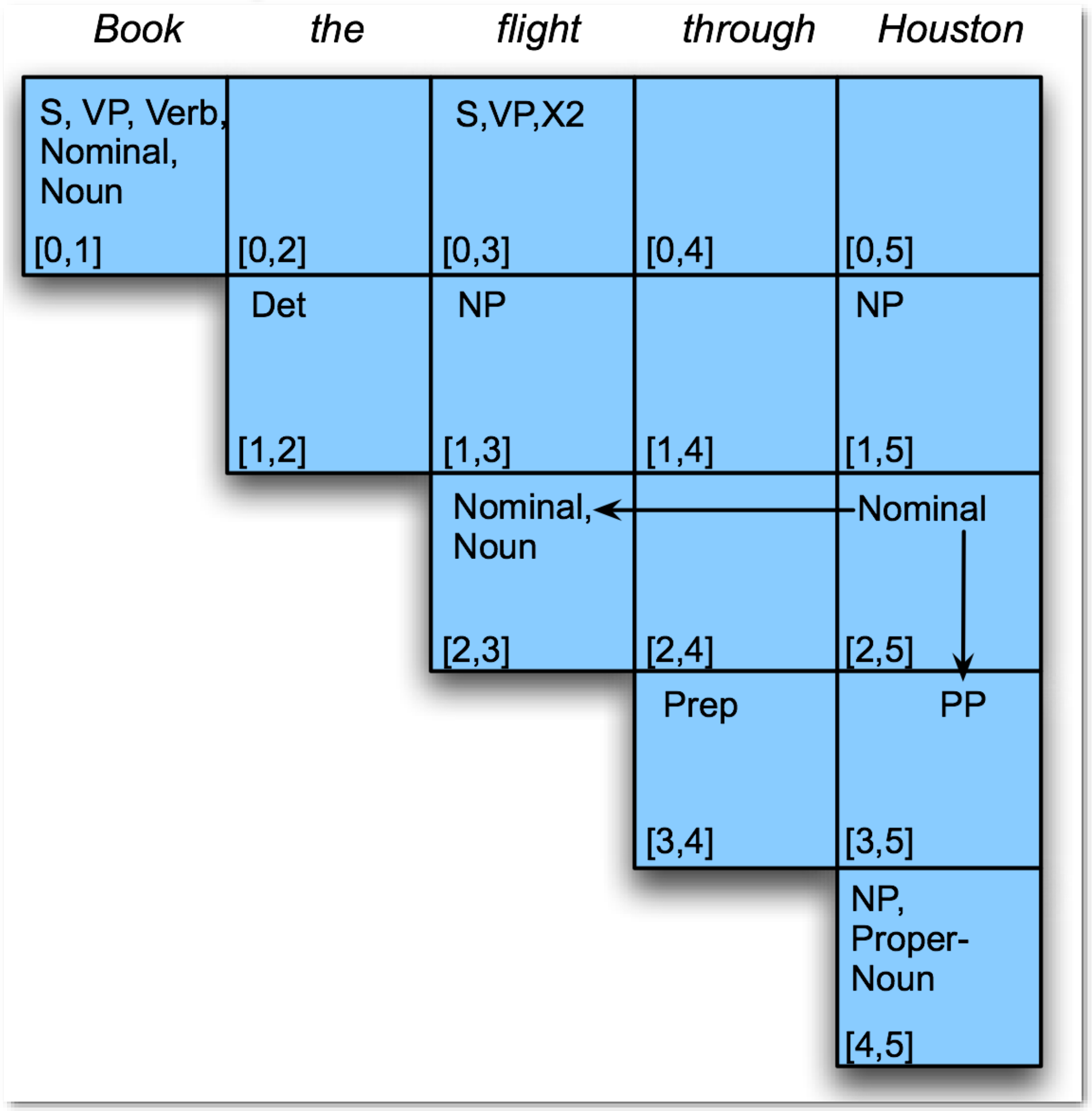
```
function CKY-PARSE(words, grammar) returns table

  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



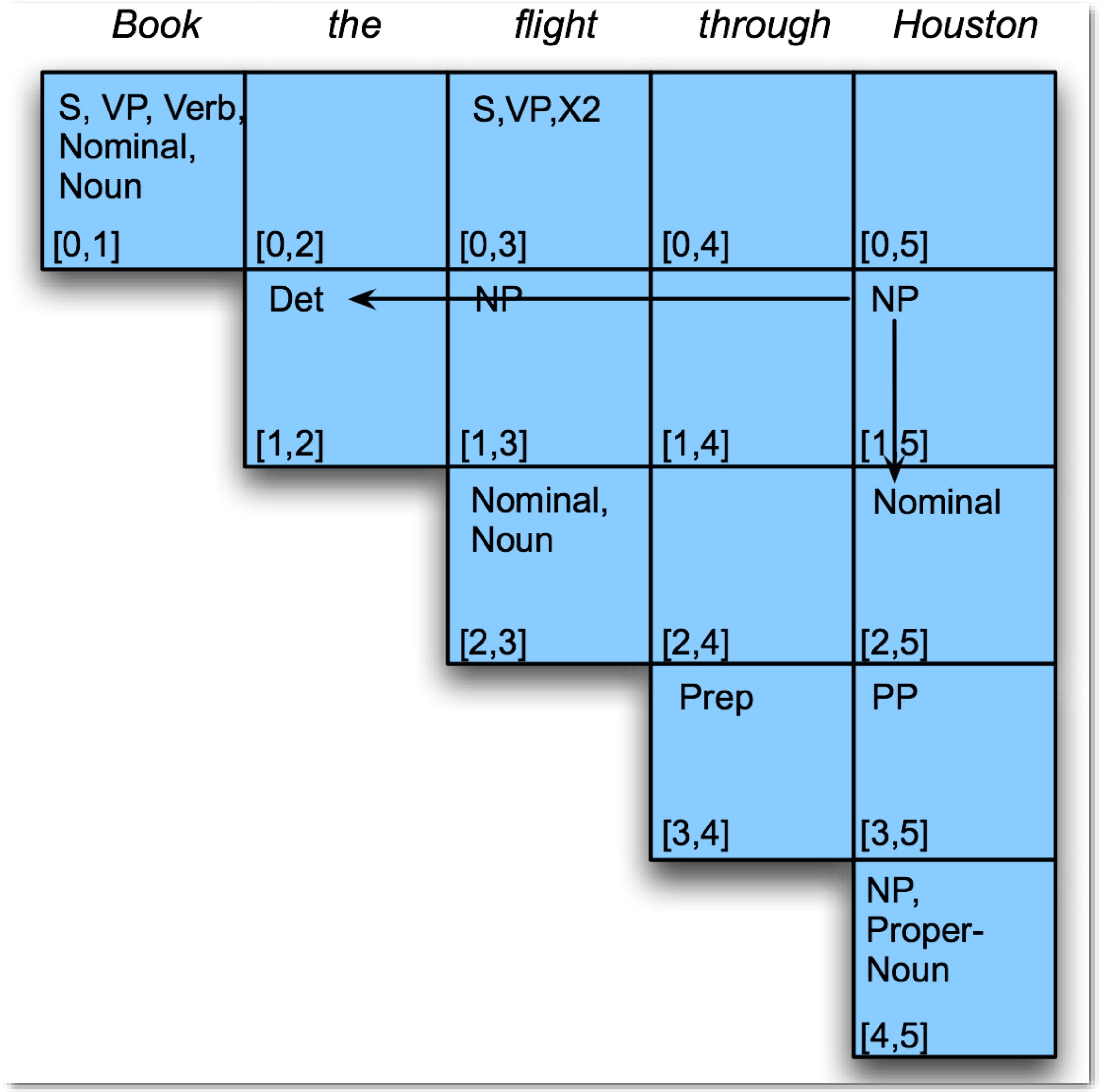
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	[2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

Example



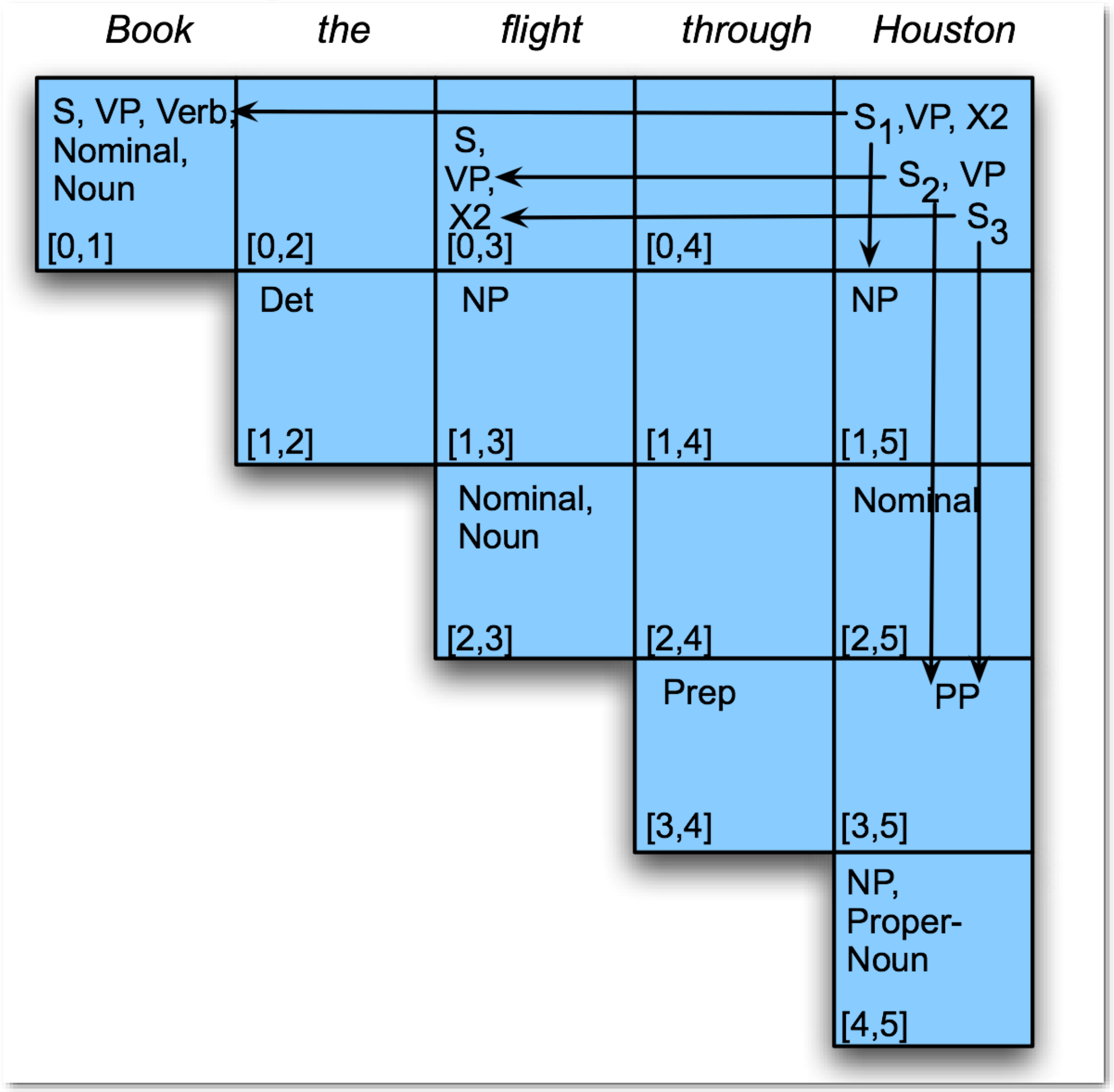
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

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 imagines 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

Earley Parsing

- Allows arbitrary CFGs
- Where CKY is bottom-up, Earley is top-down
- Fills a table in a single sweep over the input words
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

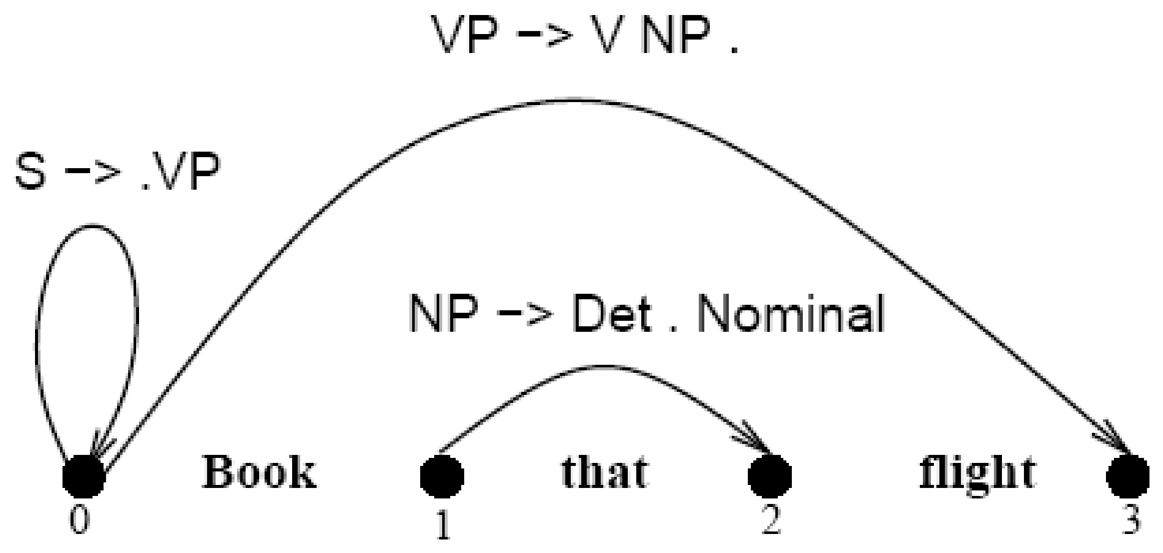
Dynamic Programming

- A standard T-D parser would reanalyze A FLIGHT 4 times, always in the same way
- A DYNAMIC PROGRAMMING algorithm uses a table (the CHART) to avoid repeating work
- The Earley algorithm also
  - Does not suffer from the left-recursion problem
  - Solves an exponential problem in $O(n^3)$

The Chart

- The Earley algorithm uses a table (the CHART) of size N+1, where N is the length of the input
  - Table entries sit in the `gaps' between words
- Each entry in the chart is a list of
  - Completed constituents
  - In-progress constituents
  - Predicted constituents
- All three types of objects are represented in the same way as STATES

THE CHART: GRAPHICAL REPRESENTATION



States

- A state encodes two types of information:
  - How much of a certain rule has been encountered in the input
  - Which positions are covered
  - A → α, [X,Y]
- DOTTED RULES
  - VP → V NP •
  - NP → Det • Nominal
  - S → • VP

Examples

Example states in parsing *Book that flight.*

- S $\rightarrow$ · VP, [0,0]
  - the first 0 indicates that the constituent begins at the start of the input
  - the second 0 indicates that the dot is here as well, and thus indicates a top-down prediction
- NP $\rightarrow$ Det · Nominal, [1,2]
  - the NP begins at position 1
  - the dot is at position 2
  - Det has thus been successfully parsed
  - Nominal is thus predicted next
- VP $\rightarrow$ V NP ·, [0,3]
  - a successful VP parse of the entire input

Success

- The parser has succeeded if entry N+1 of the chart contains the state
    - S → α •, [0,N]

THE ALGORITHM

- The algorithm loops through the input without backtracking, at each step performing three operations:
  - PREDICTOR: add predictions to the chart
  - COMPLETER: Move the dot to the right when looked-for constituent is found
  - SCANNER: read in the next input word

THE ALGORITHM: CENTRAL LOOP

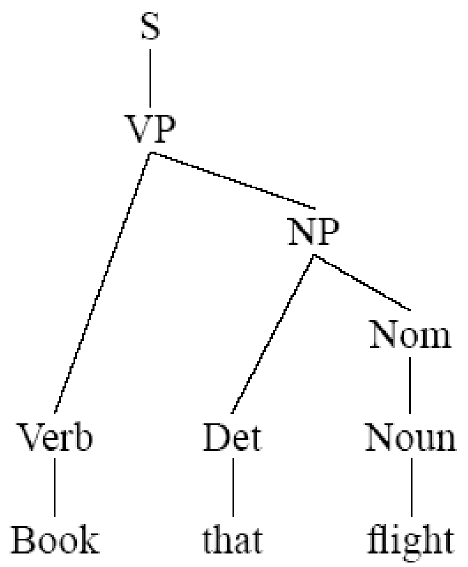
```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE(($\gamma \rightarrow \bullet S$, [0,0]), chart[0])
  for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
                NEXT-CAT(state) is not a part of speech then
          PREDICTOR(state)
      elseif INCOMPLETE?(state) and
                NEXT-CAT(state) is a part of speech then
          SCANNER(state)
      else
          COMPLETER(state)
    end
  end
  return(chart)
```

EARLEY ALGORITHM: THE THREE OPERATORS

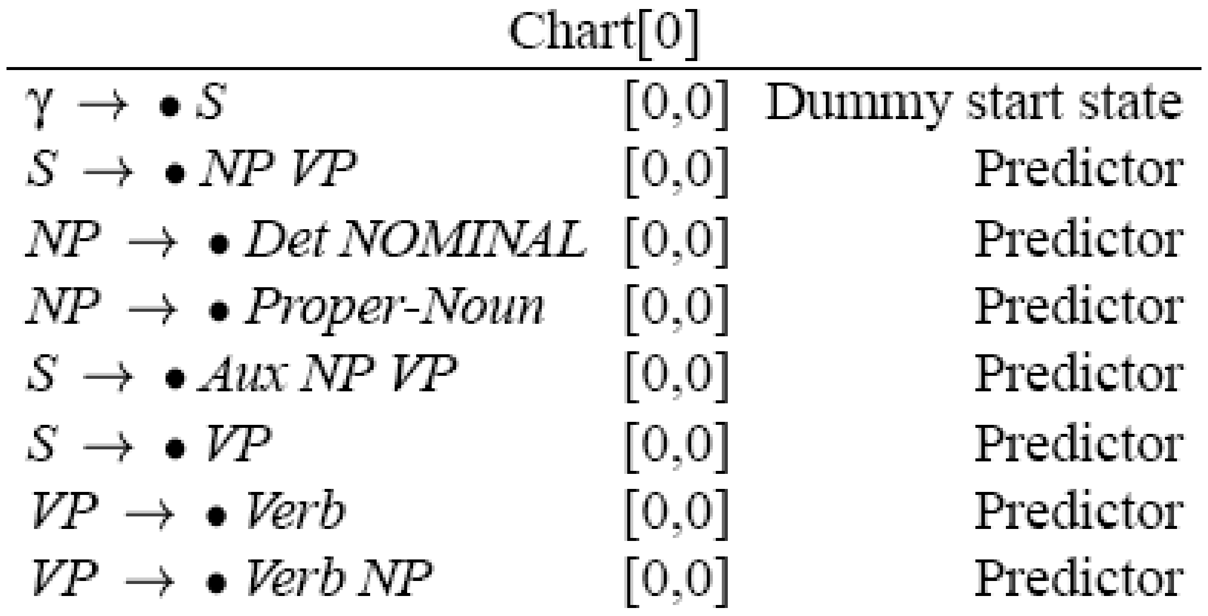
```
procedure PREDICTOR(($A \rightarrow \alpha \bullet B \beta$, $[i, j]$))
    for each ($B \rightarrow \gamma$) in GRAMMAR-RULES-FOR($B$, grammar) do
        ENQUEUE(($B \rightarrow \bullet \gamma$, $[j, j]$), chart[j])
    end

procedure SCANNER(($A \rightarrow \alpha \bullet B \beta$, $[i, j]$))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE(($B \rightarrow word[j]$, $[j, j+1]$), chart[j+1])

procedure COMPLETER(($B \rightarrow \gamma \bullet$, $[j, k]$))
    for each ($A \rightarrow \alpha \bullet B \beta$, $[i, j]$) in chart[j] do
        ENQUEUE(($A \rightarrow \alpha B \bullet \beta$, $[i, k]$), chart[k])
    end

procedure ENQUEUE(state, chart-entry)
    if state is not already in chart-entry then
        PUSH(state, chart-entry)
    end
```

EXAMPLE, AGAIN

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$



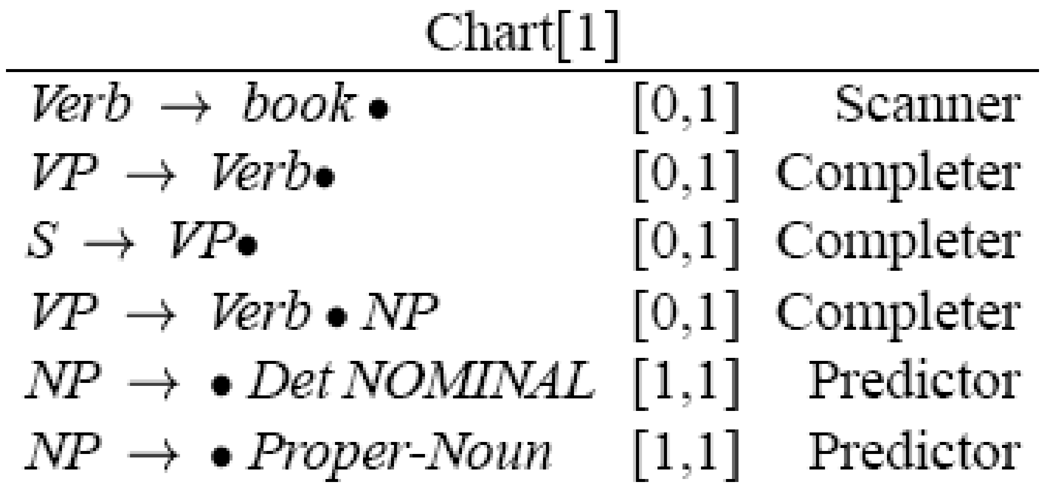
EXAMPLE:
BOOK THAT FLIGHT



Chart[0]

$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
$NP \rightarrow \bullet Det\ NOMINAL$	[0,0]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet VP$	[0,0]	Predictor
$VP \rightarrow \bullet Verb$	[0,0]	Predictor
$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor

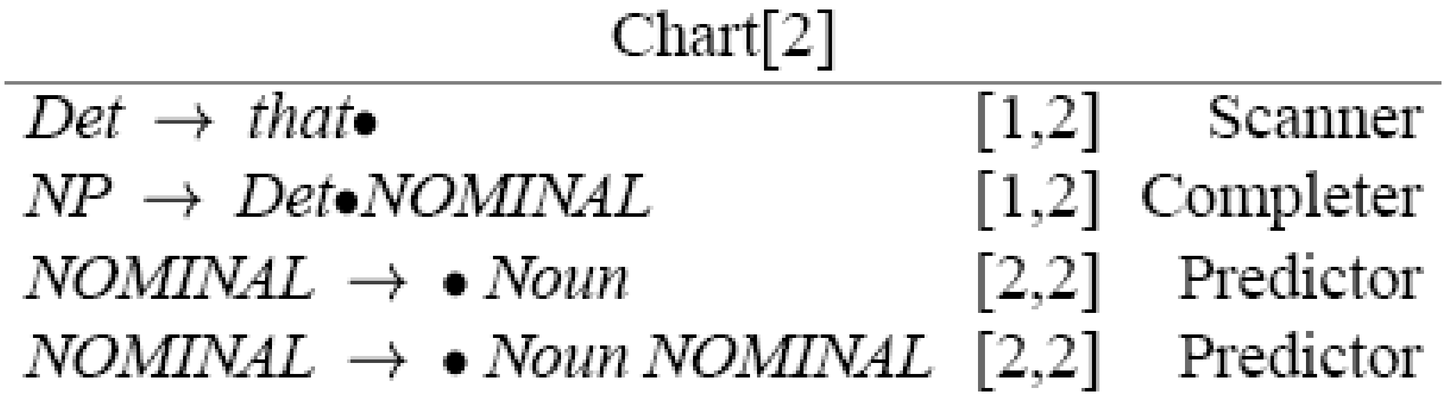
EXAMPLE:
BOOK THAT FLIGHT (II)



Chart[1]

$Verb \rightarrow book \bullet$	[0,1]	Scanner
$VP \rightarrow Verb \bullet$	[0,1]	Completer
$S \rightarrow VP \bullet$	[0,1]	Completer
$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
$NP \rightarrow \bullet Det\ NOMINAL$	[1,1]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor

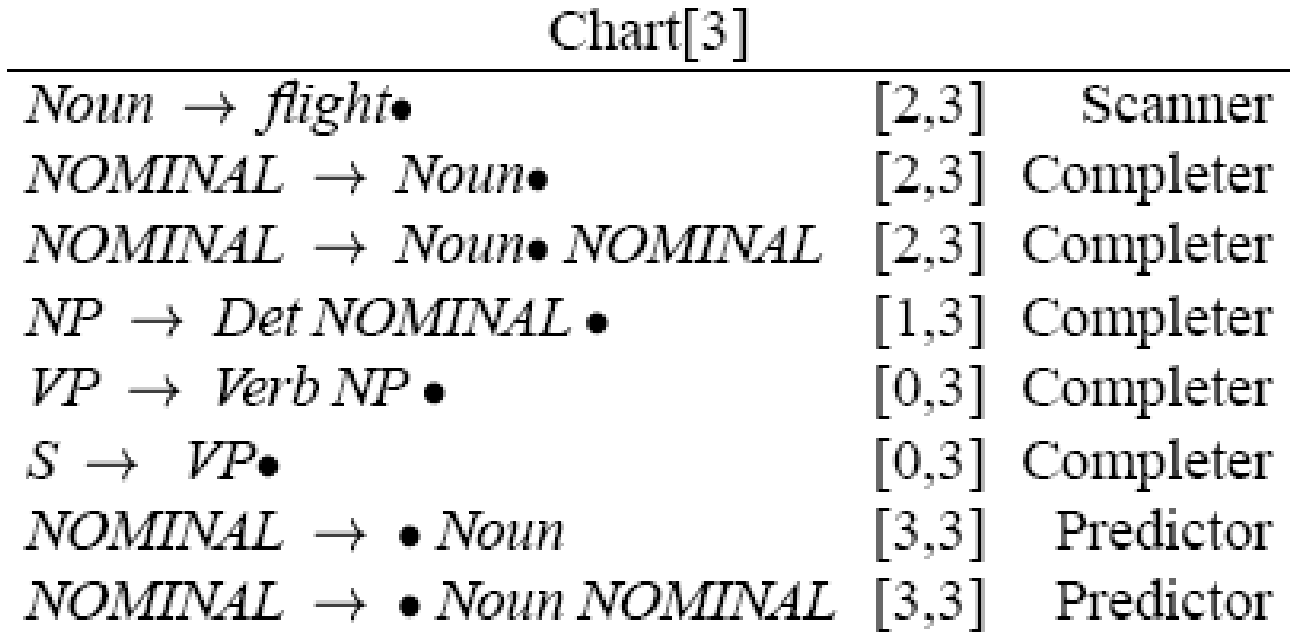
EXAMPLE:
BOOK THAT FLIGHT (III)



Chart[2]

$Det \rightarrow that \bullet$	[1,2]	Scanner
$NP \rightarrow Det \bullet NOMINAL$	[1,2]	Completer
$NOMINAL \rightarrow \bullet\, Noun$	[2,2]	Predictor
$NOMINAL \rightarrow \bullet\, Noun\ NOMINAL$	[2,2]	Predictor

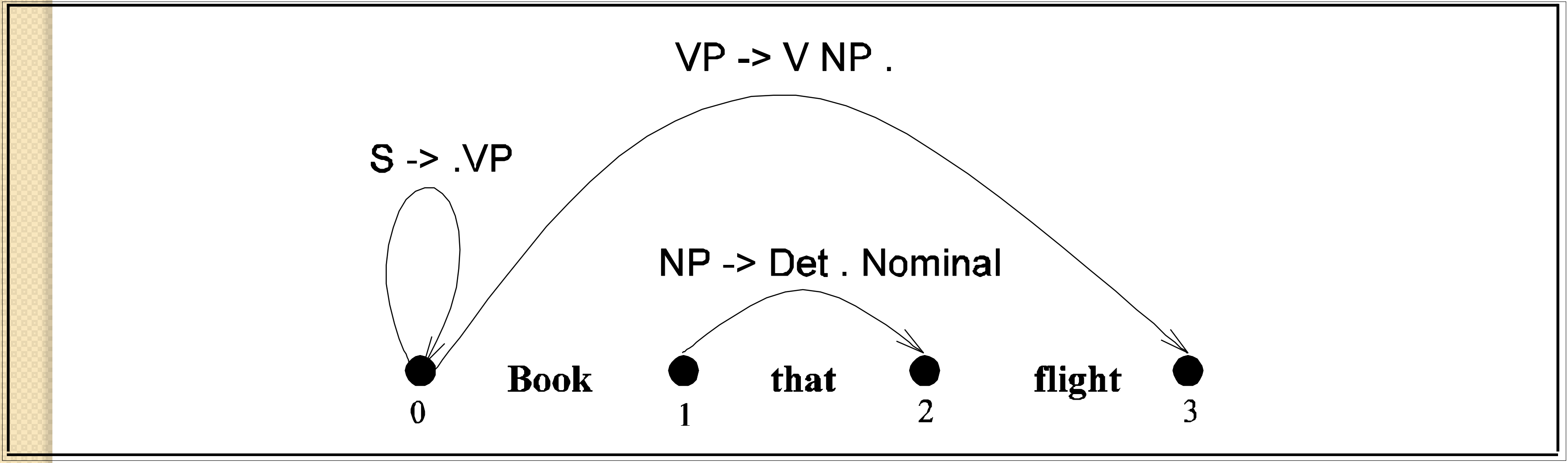
EXAMPLE:
BOOK THAT FLIGHT (IV)



Chart[3]

$Noun \rightarrow flight\bullet$	[2,3]	Scanner
$NOMINAL \rightarrow Noun\bullet$	[2,3]	Completer
$NOMINAL \rightarrow Noun\bullet\ NOMINAL$	[2,3]	Completer
$NP \rightarrow Det\ NOMINAL\bullet$	[1,3]	Completer
$VP \rightarrow Verb\ NP\bullet$	[0,3]	Completer
$S \rightarrow VP\bullet$	[0,3]	Completer
$NOMINAL \rightarrow \bullet\ Noun$	[3,3]	Predictor
$NOMINAL \rightarrow \bullet\ Noun\ NOMINAL$	[3,3]	Predictor

Graphically



Earley

- As with most dynamic programming approaches, the answer is found by looking in the table in the right place.
- In this case, there should be an S state in the final column that spans from 0 to n+1 and is complete.
- If that's the case you're done.
  - S –> α · [0,n+1]

Earley Algorithm

- March through chart left-to-right.
- At each step, apply 1 of 3 operators
  - Predictor
    - Create new states representing top-down expectations
  - Scanner
    - Match word predictions (rule with word after dot) to words
  - Completer
    - When a state is complete, see what rules were looking for that completed constituent

Earley's example 1
Predict - Scan- Complete

John called Sue from Denver

S -> . NP VP
NP -> . NP PP
P -> .V NP
VP -> .VP PP
PP -> . P NP
NP -> . John
NP -> . Sue
NP -> . Denver
V -> . called
V ->. sue
P -> . from

S -> . NP VP
NP -> . NP PP
NP -> . John
NP -> . Sue
NP -> . Denver

S -> NP .VP
NP -> NP . PP
VP -> .V NP
VP -> .VP PP
PP -> . P NP
NP -> John .
NP -> . Sue
NP -> . Denver
V -> . called
V ->. sue
P -> . from

Earley's example 2

John **called** Sue from Denver

S -> NP .VP	S -> NP .VP	S -> NP .VP
NP -> NP . PP	NP -> NP . PP	NP -> NP . PP
VP -> .V NP	VP -> .V NP	**VP -> V . NP**
VP -> .VP PP	VP -> .VP PP	
PP -> . P NP	PP -> . P NP	**V -> called .**
V -> . called	**V -> . called**	
V ->. sue	V ->. sue	
P -> . from	P -> . from	

Earley's example 3

John called **Sue** from Denver

S -> NP .VP
NP -> NP . PP
VP -> V . NP
VP -> .VP PP
PP -> . P NP
NP -> . John
NP -> . Sue
NP -> . Denver

NP -> . Sue

S -> NP VP .
S -> NP .VP
NP -> NP . PP
VP -> V NP .
VP -> VP . PP
NP -> Sue .

Earley's example 4

John called Sue **from Denver**

S -> NP .VP
NP -> NP . PP
VP -> V . NP
VP -> VP . PP
PP -> . P NP
P -> . from

P -> . from

S -> NP .VP
NP -> NP . PP
VP -> VP . PP
PP -> P . NP
P -> from .

NP -> . John
NP -> . Sue
NP -> . Denver

NP -> . Denver

NP -> Denver .
PP -> P NP .
NP -> NP PP .
VP -> VP PP .
VP -> V NP .
S -> NP VP .

Predictor

- Given a state
  - With a non-terminal to right of dot
  - That is not a part-of-speech category
  - Create a new state for each expansion of the non-terminal
  - Place these new states into same chart entry as generated state, beginning and ending where generating state ends.
  - So predictor looking at
    - S -> .VP [0,0]
  - results in
    - VP -> .Verb [0,0]
    - VP -> .Verb NP [0,0]

Scanner

- Given a state
  - With a non-terminal to right of dot
  - That is a part-of-speech category
  - If the next word in the input matches this part-of-speech
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at
    - VP -> .Verb NP [0,0]
  - If the next word, "book", can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart!

Completer

- Applied to a state when its dot has reached right end of role.
- Parser has discovered a category over some span of input.
- Find and advance all previous states that were looking for this category
  - copy state, move dot, insert in current chart entry
- Given:
  - NP -> Det Nominal . [1,3]
  - VP -> Verb. NP [0,1]
- Add
  - VP -> Verb NP . [0,3]

Earley: how do we know we are done?

- How do we know when we are done?
- Find an S state in the final column that spans from 0 to n+1 and is complete.
- If that's the case you're done.
  - S –> $\alpha \cdot$ [0,n+1]

Earley

- So sweep through the table from 0 to n+1…
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

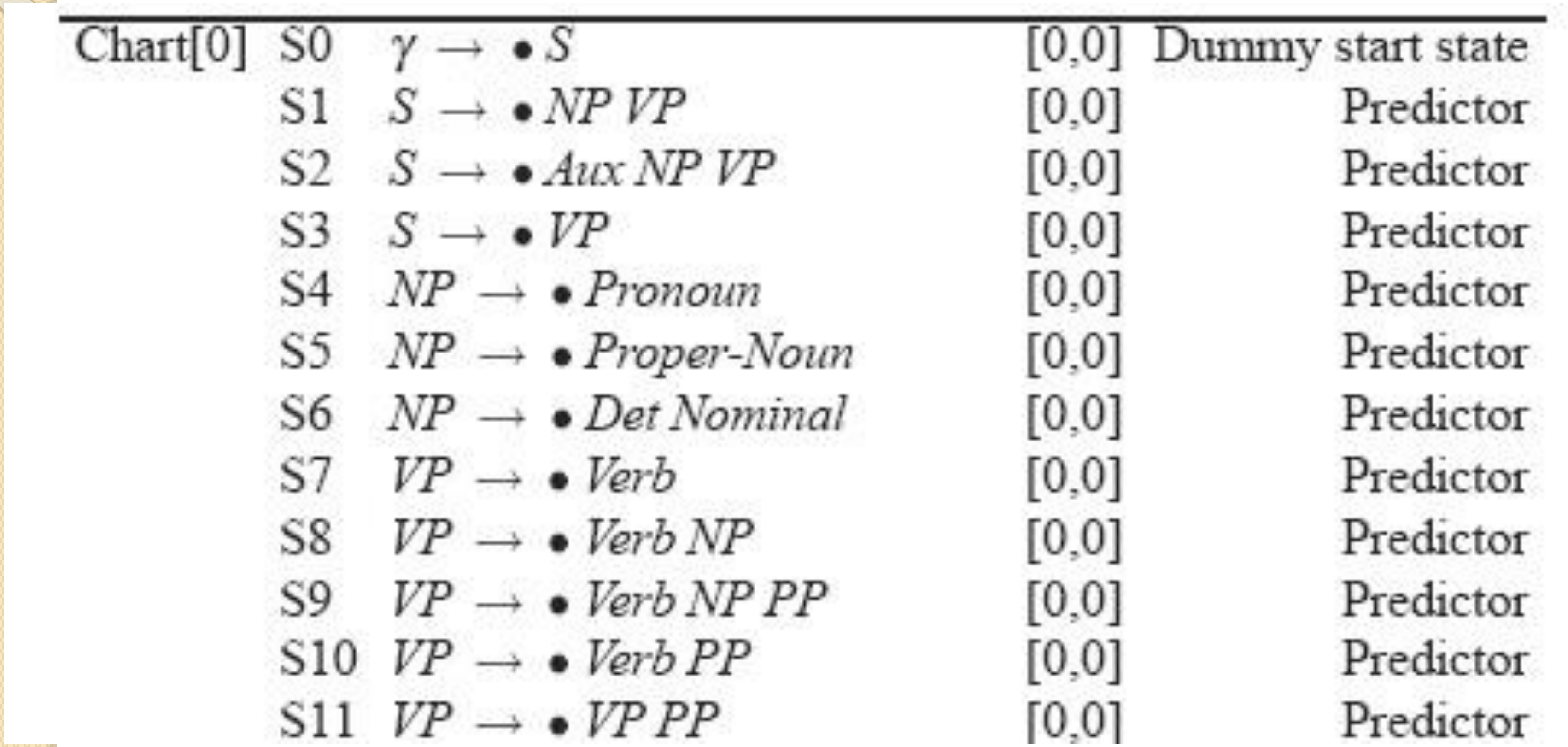
Earley

- More specifically…
  1. Predict all the states you can upfront
  2. Read a word
     1. Extend states based on matches
     2. Add new predictions
     3. Go to 2
  3. Look at N+1 to see if you have a winner

Example

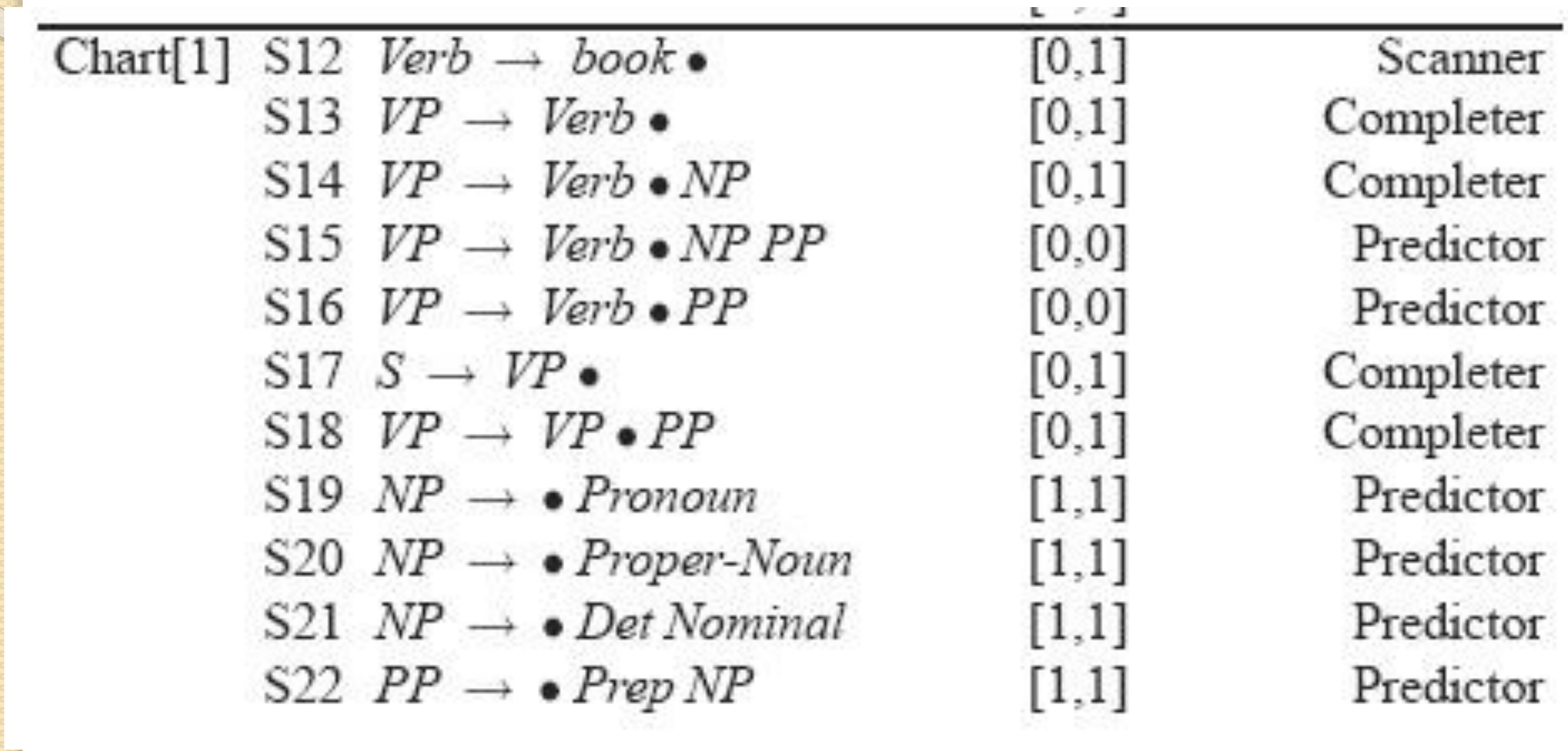
- Book that flight
- We should find… an S from 0 to 3 that is a completed state…

Example



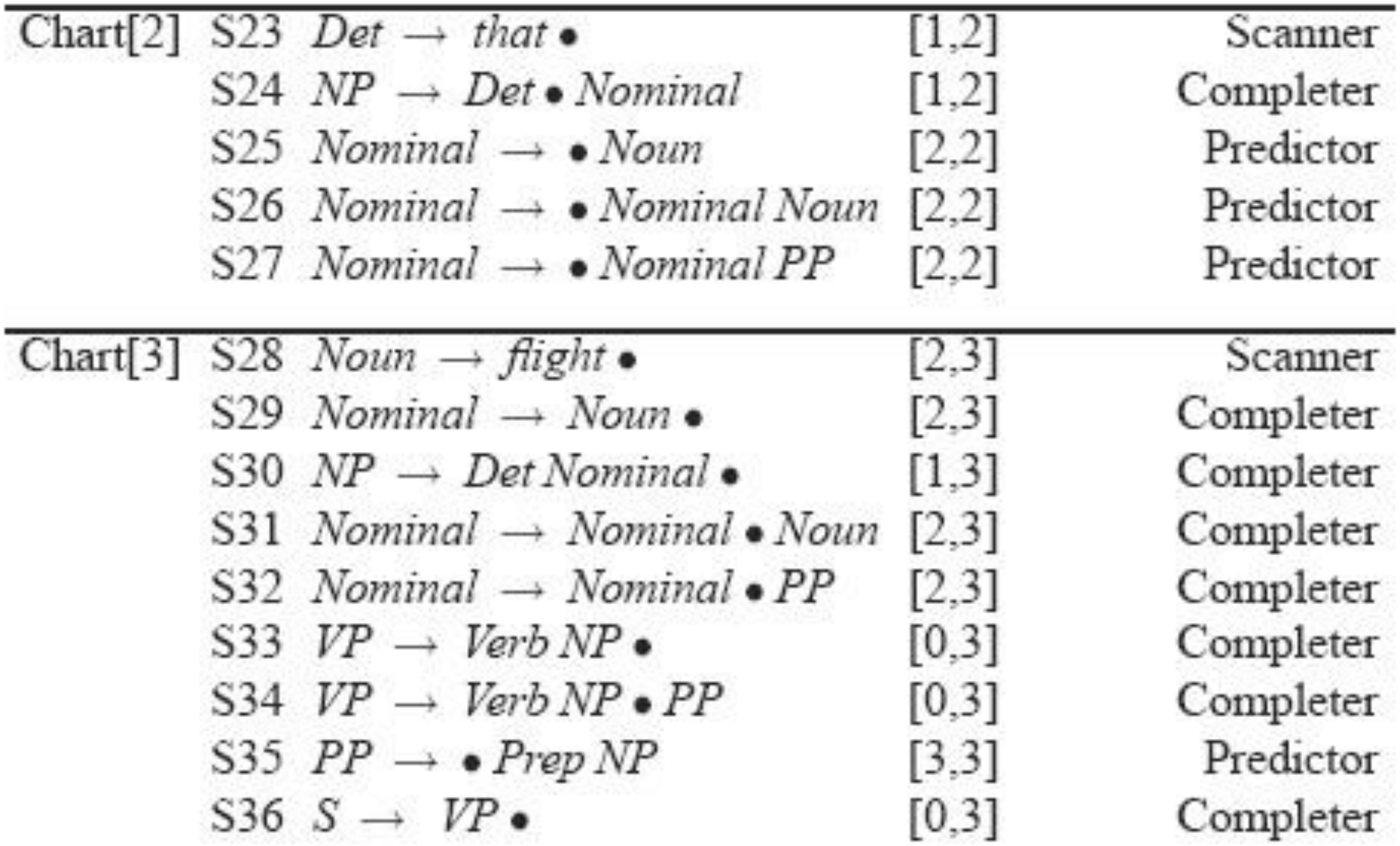
Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

Example



Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,0]	Predictor
	S16	$VP \rightarrow Verb \bullet PP$	[0,0]	Predictor
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor

Example



Chart[2]	S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
	S24	$NP \rightarrow Det \bullet Nominal$	[1,2]	Completer
	S25	$Nominal \rightarrow \bullet Noun$	[2,2]	Predictor
	S26	$Nominal \rightarrow \bullet Nominal\ Noun$	[2,2]	Predictor
	S27	$Nominal \rightarrow \bullet Nominal\ PP$	[2,2]	Predictor
Chart[3]	S28	$Noun \rightarrow flight \bullet$	[2,3]	Scanner
	S29	$Nominal \rightarrow Noun \bullet$	[2,3]	Completer
	S30	$NP \rightarrow Det\ Nominal \bullet$	[1,3]	Completer
	S31	$Nominal \rightarrow Nominal \bullet Noun$	[2,3]	Completer
	S32	$Nominal \rightarrow Nominal \bullet PP$	[2,3]	Completer
	S33	$VP \rightarrow Verb\ NP \bullet$	[0,3]	Completer
	S34	$VP \rightarrow Verb\ NP \bullet PP$	[0,3]	Completer
	S35	$PP \rightarrow \bullet Prep\ NP$	[3,3]	Predictor
	S36	$S \rightarrow VP \bullet$	[0,3]	Completer

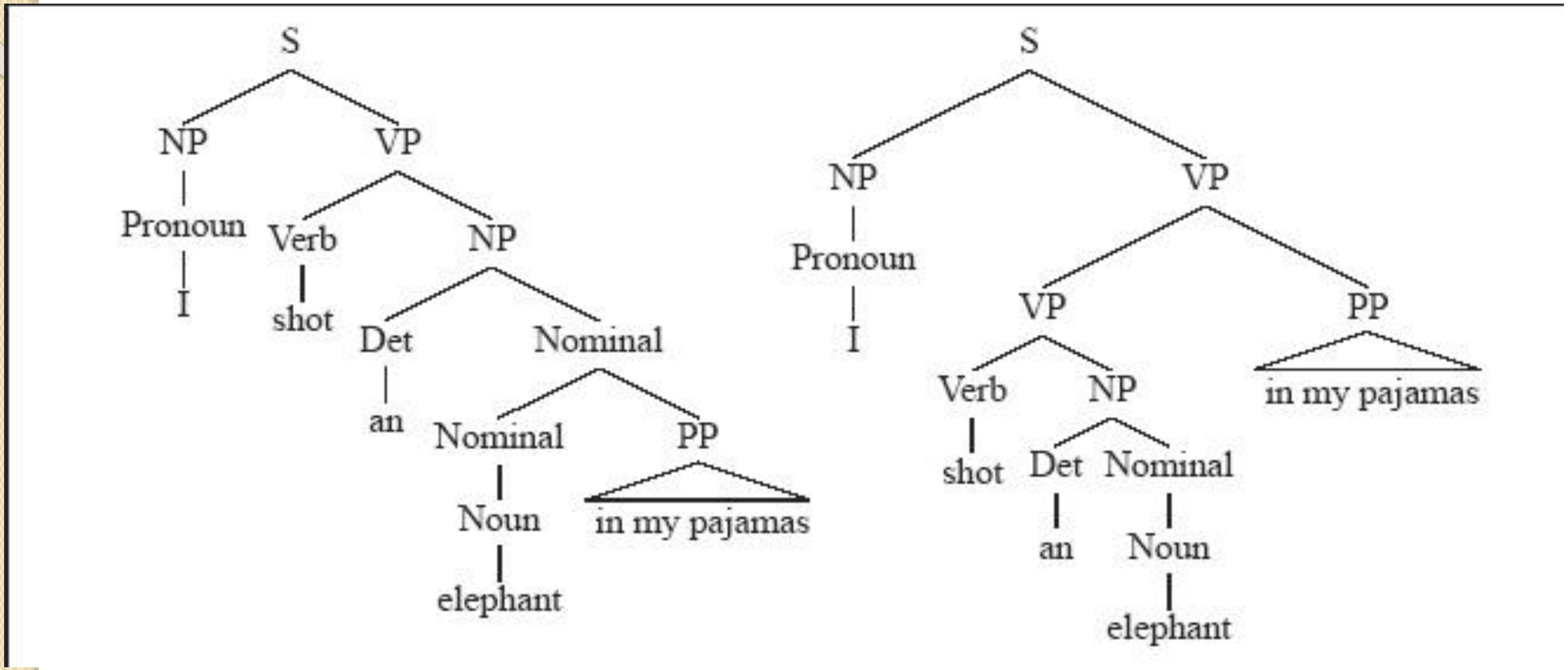
Details

- What kind of algorithms did we just describe (both Earley and CKY)
  - Not parsers – recognizers
    - The presence of an S state with the right attributes in the right place indicates a successful recognition.
    - But no parse tree… no parser
    - That’s how we solve (not) an exponential problem in polynomial time

Back to Ambiguity

- Did we solve it?

Ambiguity


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

Converting Earley from Recognizer to Parser

- With the addition of a few pointers we have a parser
- Augment the "Completer" to point to where we came from.

Augmenting the chart with structural information

Chart[1]

	Verb → *book* •	[0,1]	Scanner	
S8	*VP* → *Verb* •	[0,1]	Completer	
S9	*S* → *VP* •	[0,1]	Completer	S8
S10	*VP* → *Verb* • *NP*	[0,1]	Completer	S9
S11	*NP* → • *Det NOMINAL*	[1,1]	Predictor	S8
S12	*NP* → • *Proper-Noun*	[1,1]	Predictor	
S13				

Chart[2]

Det → *that* •	[1,2]	Scanner
NP → *Det* • *NOMINAL*	[1,2]	Completer
NOMINAL → • *Noun*	[2,2]	Predictor
NOMINAL → • *Noun NOMINAL*	[2,2]	Predictor

Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table
- We just need to read off all the backpointers from every complete S in the last column of the table
- Find all the S -> X . [0,N+1]
- Follow the structural traces from the Completer
- Of course, this won't be polynomial time, since there could be an exponential number of trees
- So we can at least represent ambiguity efficiently

Statistical Parsing

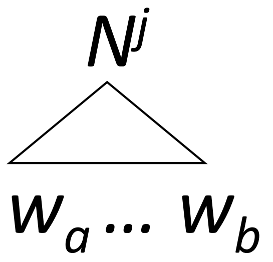
- Statistical parsing uses a probabilistic model of syntax in order to assign probabilities to each parse tree.
- Provides principled approach to resolving syntactic ambiguity.
- Allows supervised learning of parsers from tree-banks of parse trees provided by human linguists.
- Also allows unsupervised learning of parsers from unannotated text, but the accuracy of such parsers has been limited.

Probabilistic Context Free Grammar (PCFG)

- A PCFG is a probabilistic version of a CFG where each production has a probability.
- Probabilities of all productions rewriting a given non-terminal must add to 1, defining a distribution for each non-terminal.
- String generation is now probabilistic where production probabilities are used to non-deterministically select a production for rewriting a given non-terminal.

PCFGs – Notation

- $w_{1n} = w_1 \ldots w_n$ = the word sequence from 1 to n (sentence of length n)
- w_{ab} = the subsequence $w_a \ldots w_b$
- N^j_{ab} = the nonterminal N^j dominating $w_a \ldots w_b$



- We'll write $P(N^i \rightarrow \zeta^j)$ to mean $P(N^i \rightarrow \zeta^j \mid N^i)$
- We'll want to calculate $\max_t P(t \Rightarrow^* w_{ab})$

The probability of trees and strings

- P(w_{1n}, *t*) -- The probability of tree is the product of the probabilities of the rules used to generate it.

$$P(w_{1n}, t) = \prod_{\{R = X \rightarrow AB\} \in t} P(R) \prod_{\{R = X \rightarrow w_i\} \in t} P(R)$$

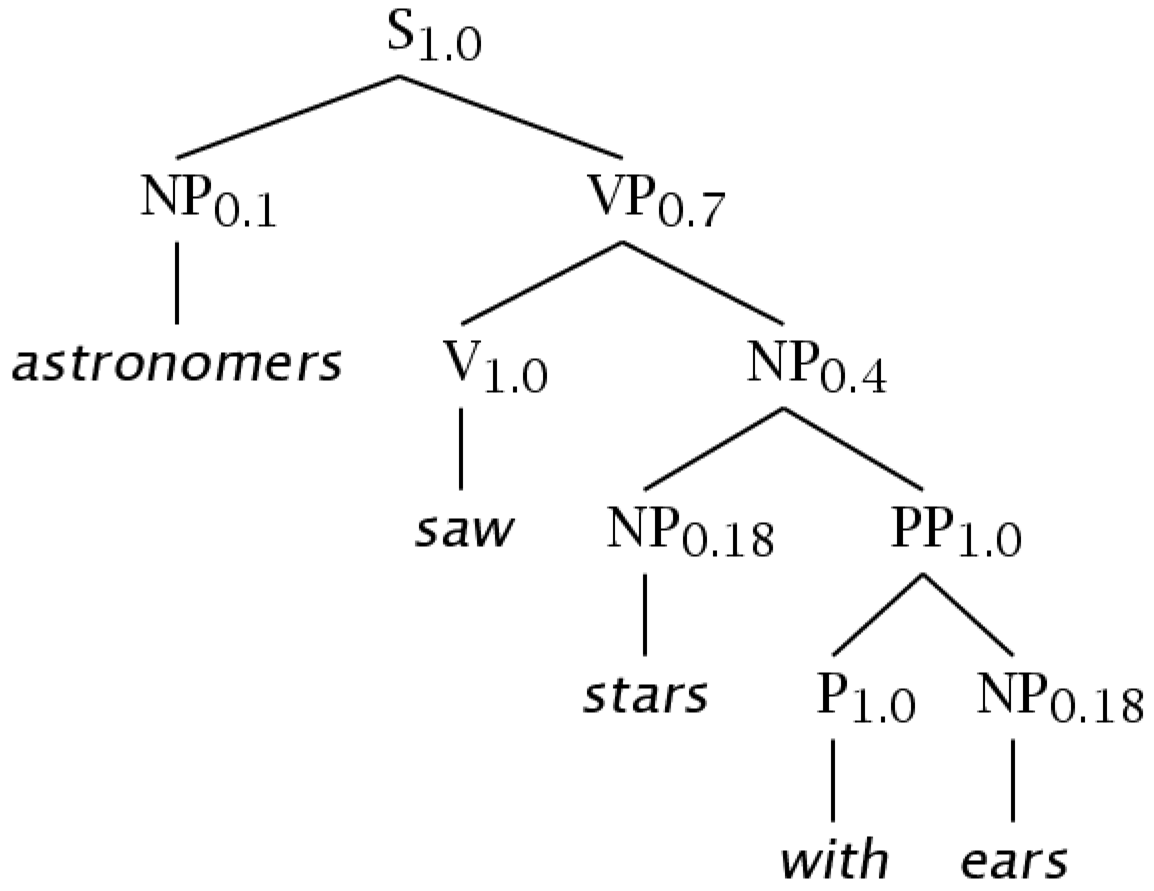
- P(w_{1n}) -- The probability of the string is the sum of the probabilities of the trees which have that string as their yield

$P(w_{1n}) = \Sigma_t P(w_{1n}, t)$ where *t* is a parse of w_{1n}

Example: A Simple PCFG (in Chomsky Normal Form)

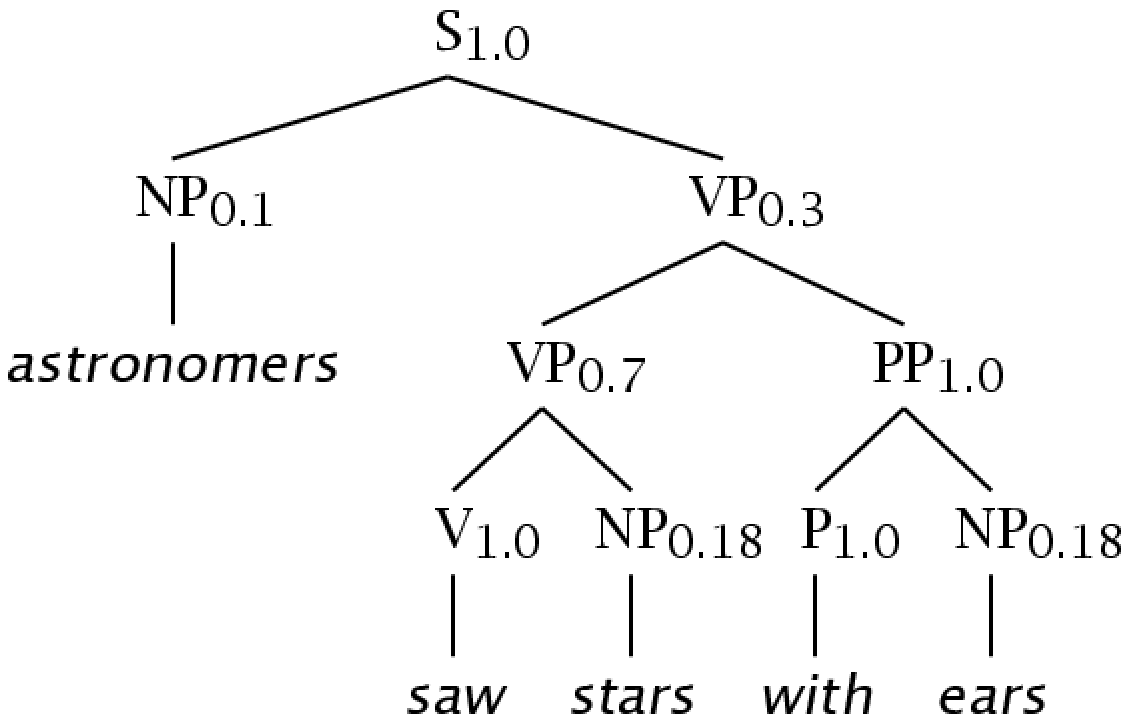
S	→	NP VP	1.0	NP	→	NP PP	0.4
VP	→	V NP	0.7	NP	→	*astronomers*	0.1
VP	→	VP PP	0.3	NP	→	*ears*	0.18
PP	→	P NP	1.0	NP	→	*saw*	0.04
P	→	*with*	1.0	NP	→	*stars*	0.18
V	→	*saw*	1.0	NP	→	*telescope*	0.1

t_1:


$S_{1.0}$
$NP_{0.1}$
$VP_{0.7}$
astronomers
$V_{1.0}$
$NP_{0.4}$
saw
$NP_{0.18}$
$PP_{1.0}$
stars
$P_{1.0}$
$NP_{0.18}$
with
ears

$$P(t_1) =$$

t_2:


$S_{1.0}$
$NP_{0.1}$
$VP_{0.3}$
astronomers
$VP_{0.7}$
$PP_{1.0}$
$V_{1.0}$
$NP_{0.18}$
$P_{1.0}$
$NP_{0.18}$
saw
stars
with
ears

Tree and String Probabilities

- w_{15} = *astronomers saw stars with ears*
- $P(t_1)$ = 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18
  * 1.0 * 1.0 * 0.18
  = 0.0009072
- $P(t_2)$ = 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18
  * 1.0 * 1.0 * 0.18
  = 0.0006804
- $P(w_{15})$ = $P(t_1)$ + $P(t_2)$
  = 0.0009072 + 0.0006804
  = 0.0015876

Simple PCFG for ATIS English

Grammar	Prob	
S → NP VP	0.8	
S → Aux NP VP	0.1	+ 1.0
S → VP	0.1	
NP → Pronoun	0.2	
NP → Proper-Noun	0.2	+ 1.0
NP → Det Nominal	0.6	
Nominal → Noun	0.3	
Nominal → Nominal Noun	0.2	+ 1.0
Nominal → Nominal PP	0.5	
VP → Verb	0.2	
VP → Verb NP	0.5	+ 1.0
VP → VP PP	0.3	
PP → Prep NP	1.0	

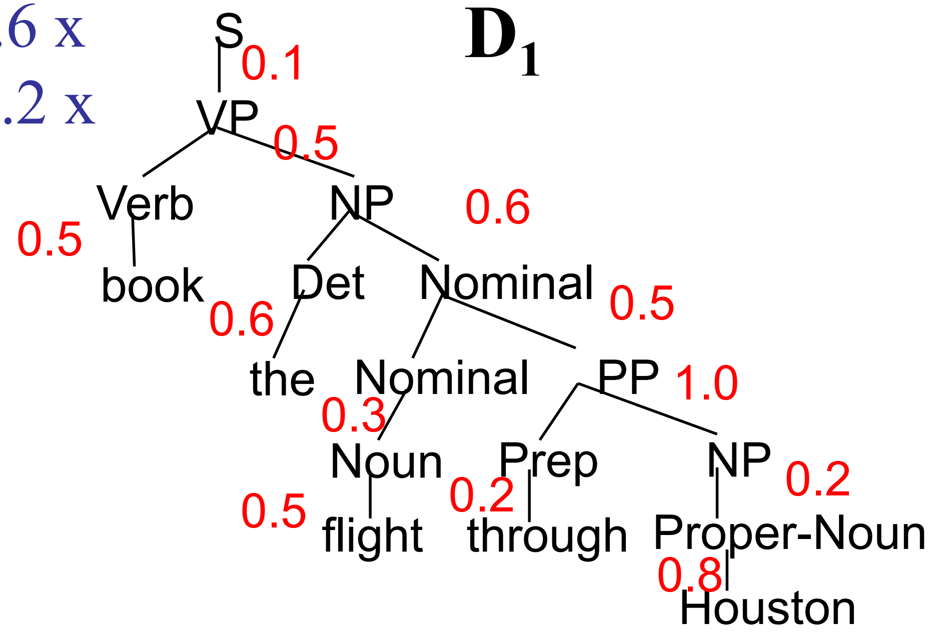
Lexicon

Det →	the	a	that	this	
	0.6	0.2	0.1	0.1	
Noun →	book	flight	meal	money	
	0.1	0.5	0.2	0.2	
Verb →	book	include	prefer		
	0.5	0.2	0.3		
Pronoun →	I	he	she	me	
	0.5	0.1	0.1	0.3	
Proper-Noun →	Houston	NWA			
	0.8	0.2			
Aux →	does				
	1.0				
Prep →	from	to	on	near	through
	0.25	0.25	0.1	0.2	0.2

Sentence Probability

- Assume productions for each node are chosen independently.
- Probability of derivation is the product of the probabilities of its productions.

$$P(D_1) = 0.1 \times 0.5 \times 0.5 \times 0.6 \times 0.6 \times 0.5 \times 0.3 \times 1.0 \times 0.2 \times 0.2 \times 0.5 \times 0.8 = 0.0000216$$

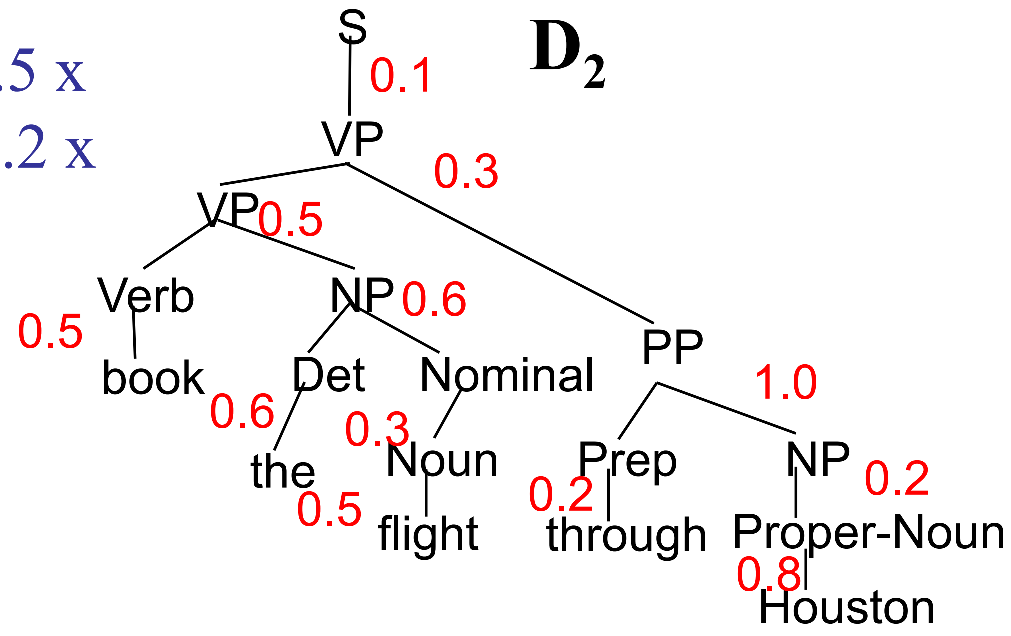


Syntactic Disambiguation

- Resolve ambiguity by picking most probable parse tree.

$P(D_2) = 0.1 \times 0.3 \times 0.5 \times 0.6 \times 0.5 \times 0.6 \times 0.3 \times 1.0 \times 0.5 \times 0.2 \times 0.2 \times 0.8$

$= 0.00001296$



Sentence Probability

- Probability of a sentence is the sum of the probabilities of all of its derivations.

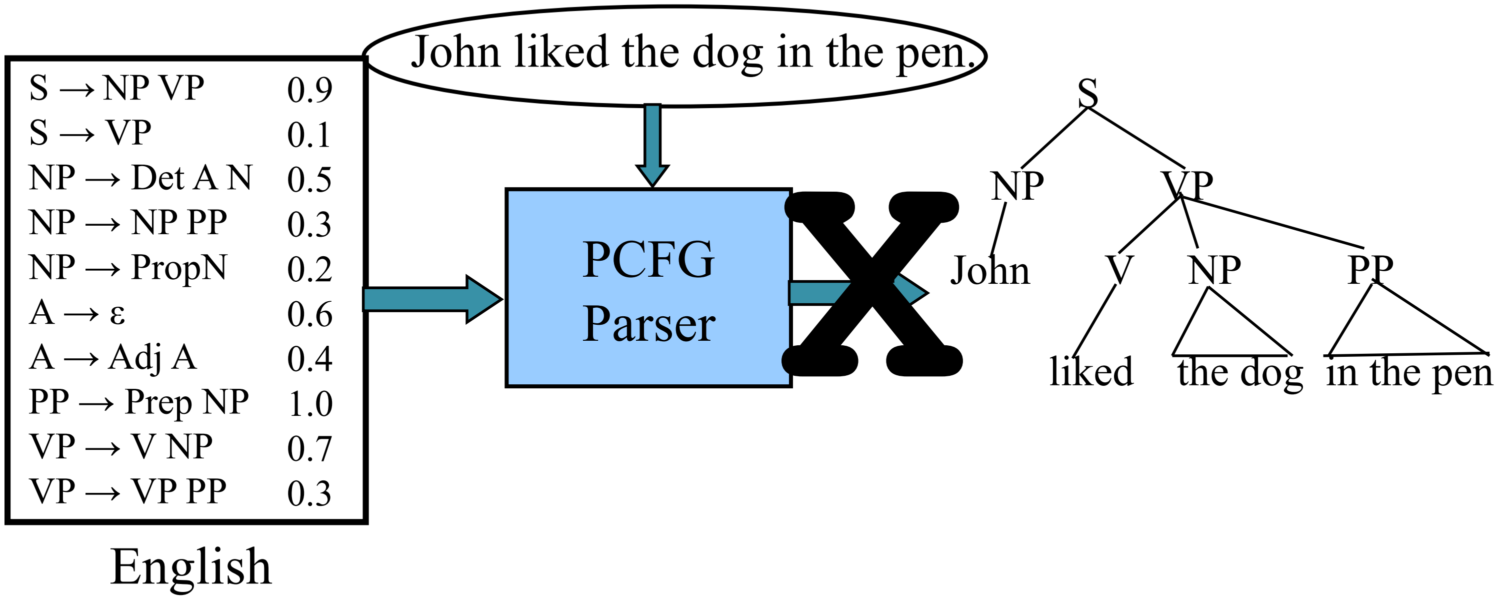
P("book the flight through Houston") =
$P(D_1) + P(D_2) = 0.0000216 + 0.00001296$
$= 0.00003456$

Three Useful PCFG Tasks

- Observation likelihood: To classify and order sentences.
- Most likely derivation: To determine the most likely parse tree for a sentence.
- Maximum likelihood training: To train a PCFG to fit empirical training data.

PCFG: Most Likely Derivation

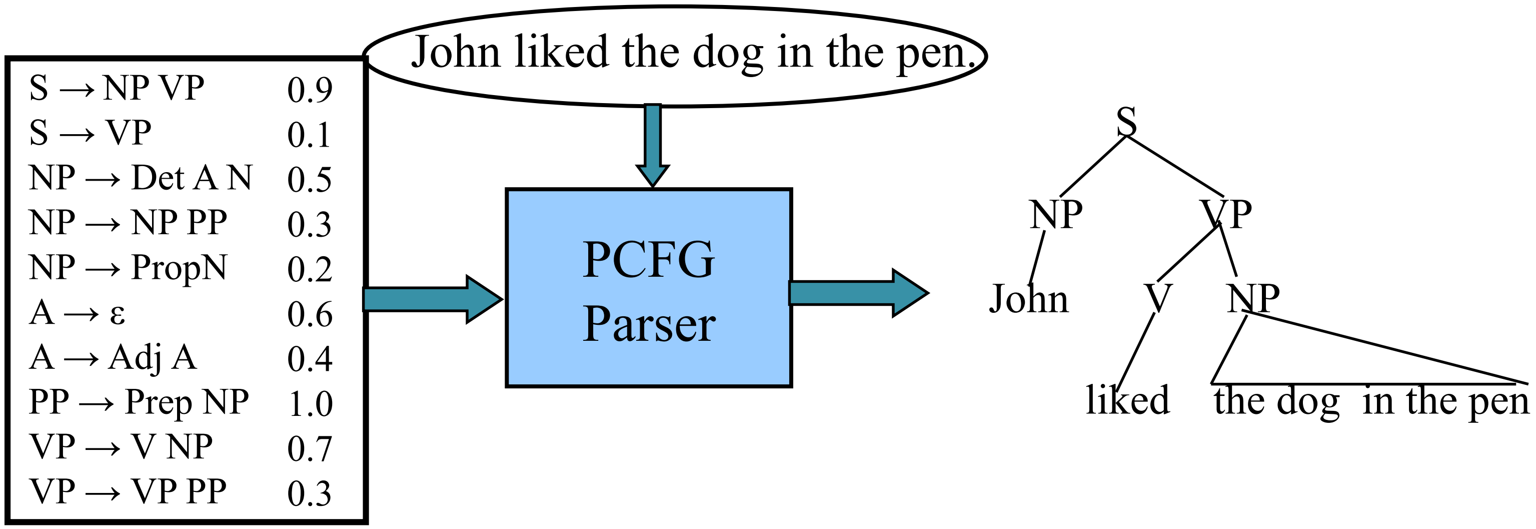
- There is an analog to the Viterbi algorithm to efficiently determine the most probable derivation (parse tree) for a sentence.



S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

PCFG: Most Likely Derivation

- There is an analog to the Viterbi algorithm to efficiently determine the most probable derivation (parse tree) for a sentence.



$S \rightarrow NP\ VP$	0.9
$S \rightarrow VP$	0.1
$NP \rightarrow Det\ A\ N$	0.5
$NP \rightarrow NP\ PP$	0.3
$NP \rightarrow PropN$	0.2
$A \rightarrow \varepsilon$	0.6
$A \rightarrow Adj\ A$	0.4
$PP \rightarrow Prep\ NP$	1.0
$VP \rightarrow V\ NP$	0.7
$VP \rightarrow VP\ PP$	0.3

English

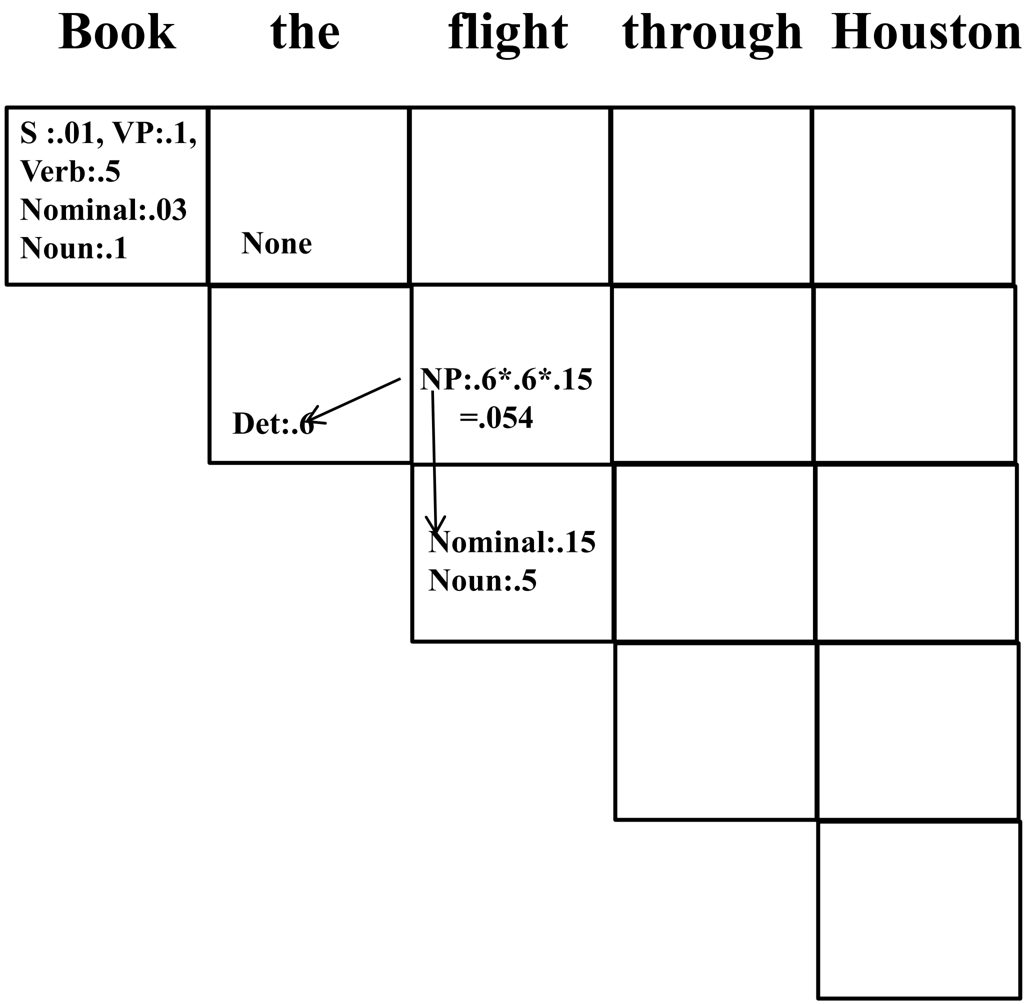
Probabilistic CKY

- CKY can be modified for PCFG parsing by including in each cell a probability for each non-terminal.
- Cell[*i*,*j*] must retain the *most probable* derivation of each constituent (non-terminal) covering words *i* +1 through *j* together with its associated probability.
- When transforming the grammar to CNF, must set production probabilities to preserve the probability of derivations.

Probabilistic Grammar Conversion

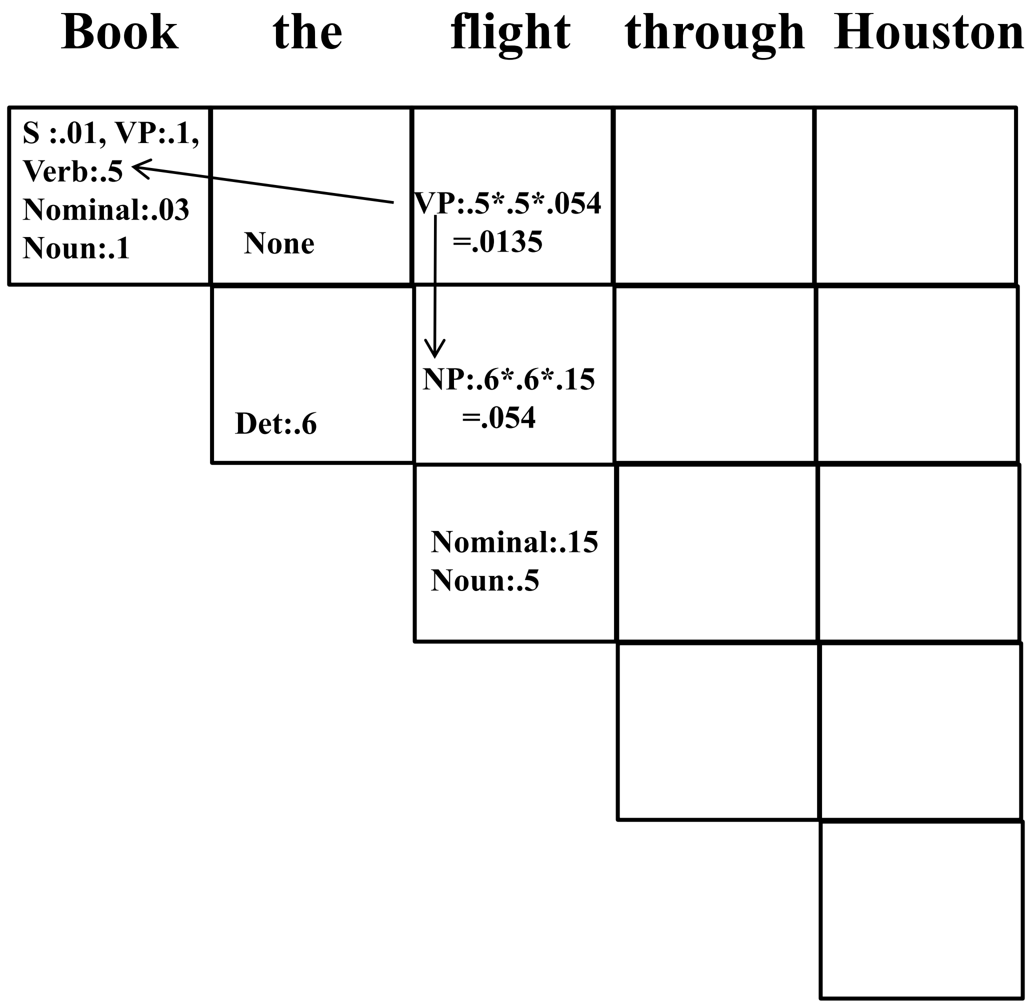
Original Grammar		Chomsky Normal Form	
S → NP VP	0.8	S → NP VP	0.8
S → Aux NP VP	0.1	S → X1 VP	0.1
		X1 → Aux NP	1.0
S → VP	0.1	S → book | include | prefer	0.01 0.004 0.006
		S → Verb NP	0.05
		S → VP PP	0.03
NP → Pronoun	0.2	NP → I | he | she | me	0.1 0.02 0.02 0.06
NP → Proper-Noun	0.2	NP → Houston | NWA	0.16 .04
NP → Det Nominal	0.6	NP → Det Nominal	0.6
Nominal → Noun	0.3	Nominal → book | flight | meal | money	0.03 0.15 0.06 0.06
Nominal → Nominal Noun	0.2	Nominal → Nominal Noun	0.2
Nominal → Nominal PP	0.5	Nominal → Nominal PP	0.5
VP → Verb	0.2	VP → book | include | prefer	0.1 0.04 0.06
VP → Verb NP	0.5	VP → Verb NP	0.5
VP → VP PP	0.3	VP → VP PP	0.3
PP → Prep NP	1.0	PP → Prep NP	1.0

Probabilistic CKY Parser



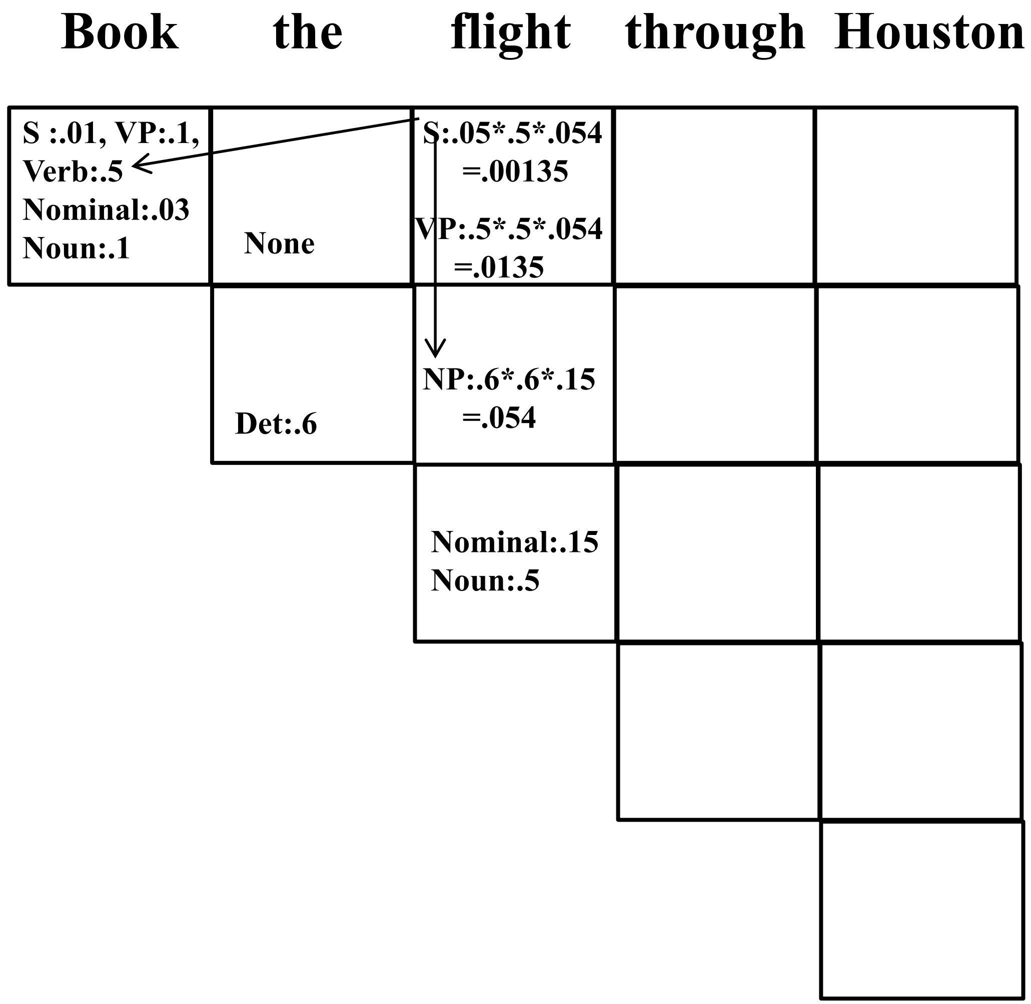
Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None			
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

Probabilistic CKY Parser



Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	VP:.5*.5*.054 =.0135		
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

Probabilistic CKY Parser



Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135		
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	
			Prep:.2	

Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

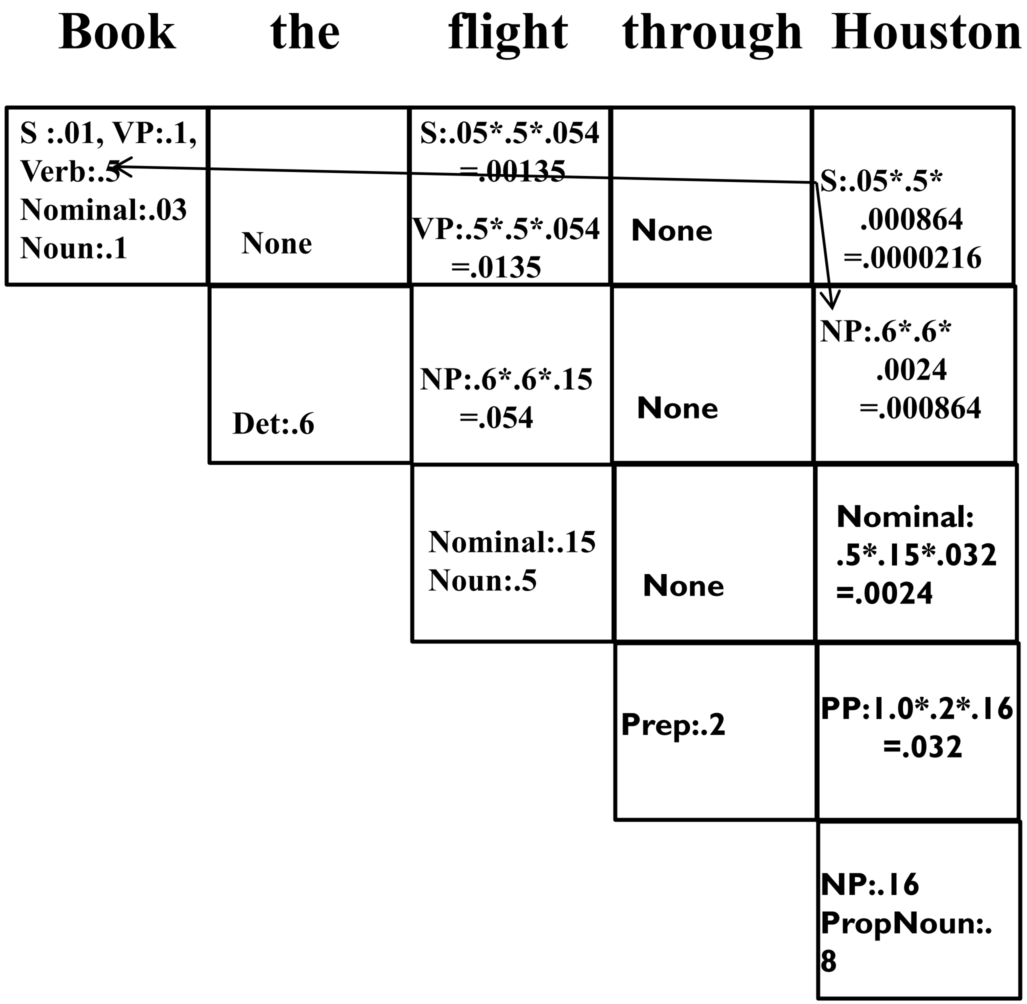
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Probabilistic CKY Parser

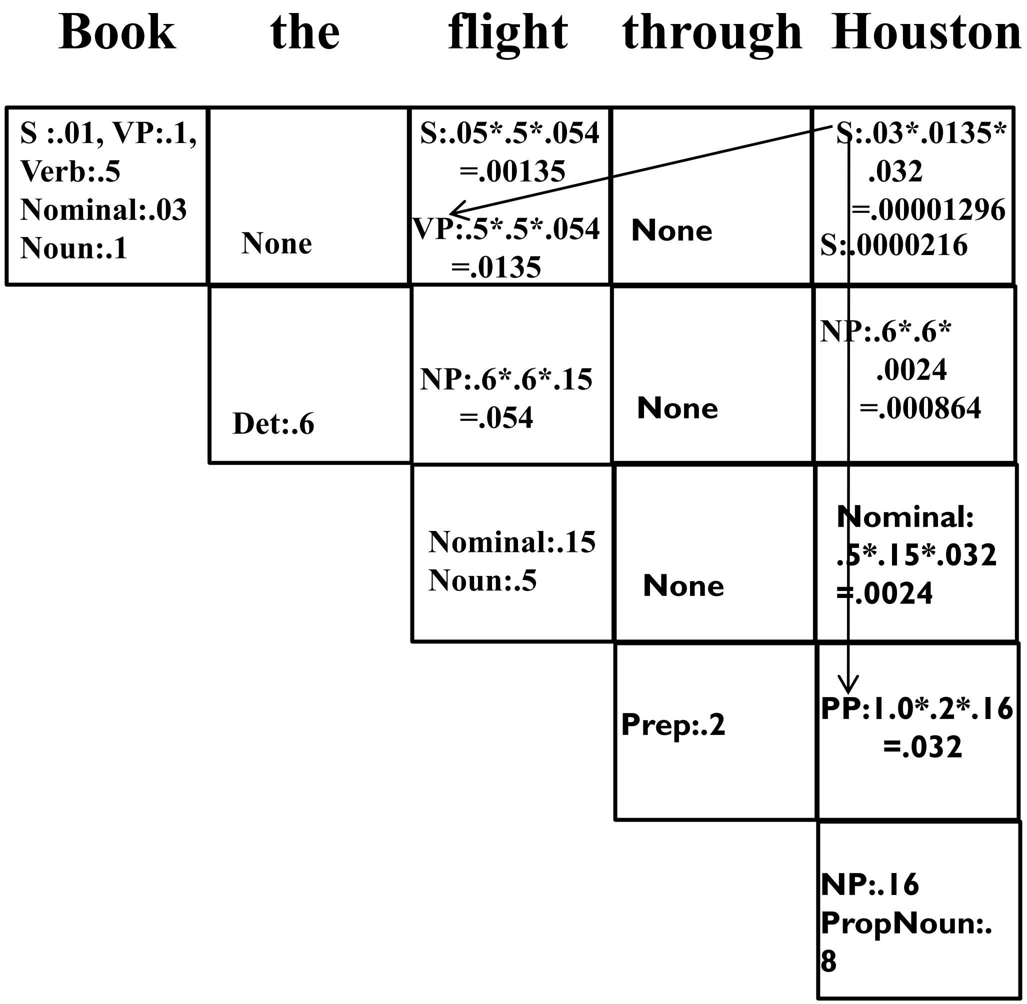
Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Probabilistic CKY Parser



Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S:.05*.5* .000864 =.0000216
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Probabilistic CKY Parser



Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S:.03*.0135* .032 =.00001296 S:.0000216
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

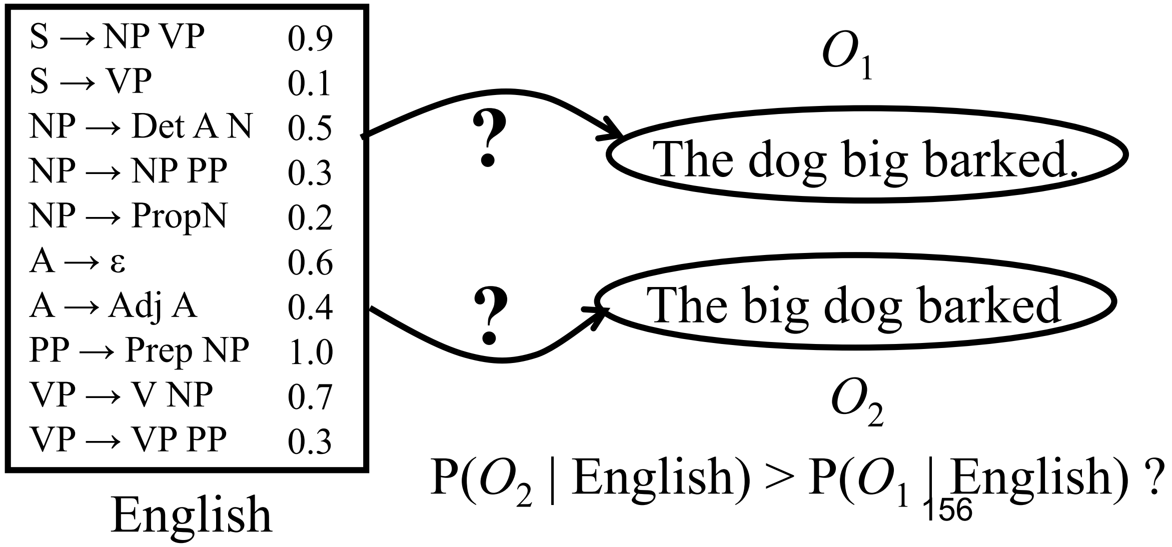
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S:.0000216
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Pick most probable parse, i.e. take max to combine probabilities of multiple derivations of each constituent in each cell.

PCFG: Observation Likelihood

- There is an analog to Forward algorithm for HMMs called the Inside algorithm for efficiently determining how likely a string is to be produced by a PCFG.
- Can use a PCFG as a language model to choose between alternative sentences for speech recognition or machine translation.



S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

English

P(O_2 | English) > P(O_1 | English) ?

Inside Algorithm

- Use CKY probabilistic parsing algorithm but combine probabilities of multiple derivations of any constituent using **addition** instead of **max**.

Probabilistic CKY Parser for Inside Computation

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S:..00001296 S:.0000216
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:. 8

Probabilistic CKY Parser for Inside Computation

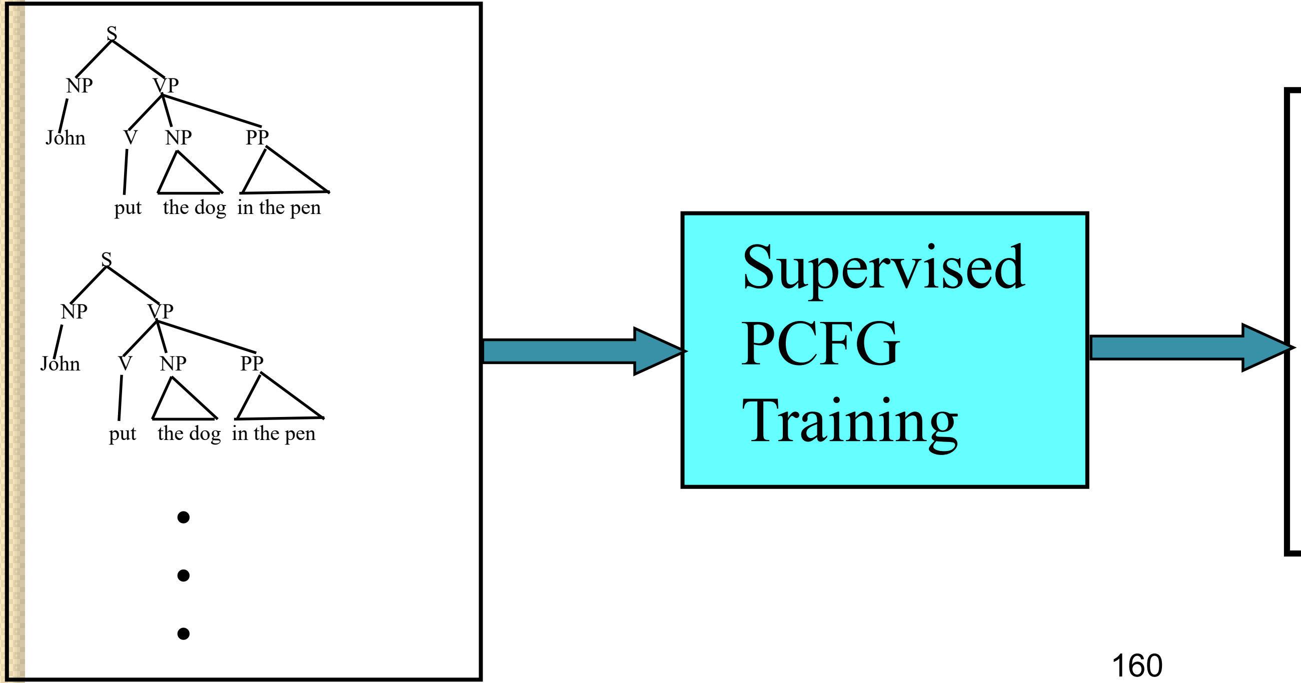
Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S: .00001296 +.0000216 =.00003456
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Sum probabilities of multiple derivations of each constituent in each cell.

PCFG: Supervised Training

- If parse trees are provided for training sentences, a grammar and its parameters can be can all be estimated directly from counts accumulated from the tree-bank (with appropriate smoothing).

Tree Bank



S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

English

Estimating Production Probabilities

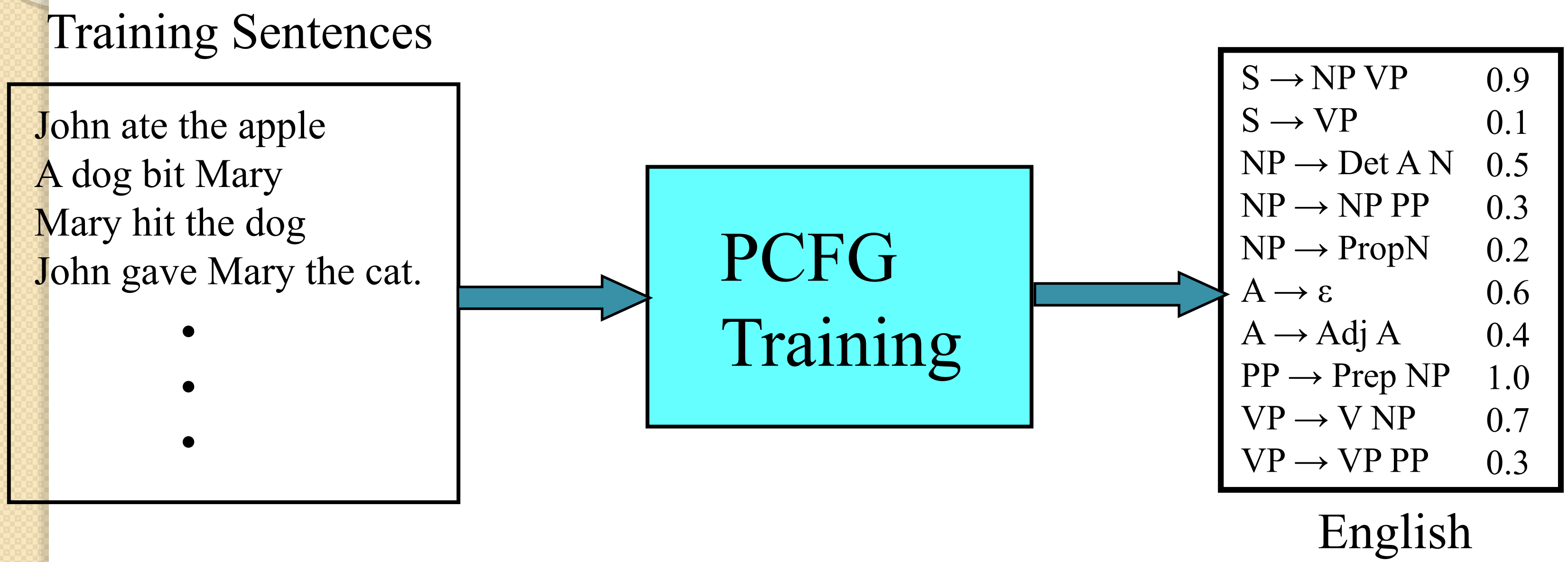
- Set of production rules can be taken directly from the set of rewrites in the treebank.
- Parameters can be directly estimated from frequency counts in the treebank.

$$P(\alpha \rightarrow \beta \mid \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{count}(\alpha \rightarrow \gamma)} = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}$$

PCFG: Maximum Likelihood Training

- Given a set of sentences, induce a grammar that maximizes the probability that this data was generated from this grammar.
- Assume the number of non-terminals in the grammar is specified.
- Only need to have an unannotated set of sequences generated from the model. Does not need correct parse trees for these sentences. In this sense, it is unsupervised.

PCFG: Maximum Likelihood Training



S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

Inside-Outside

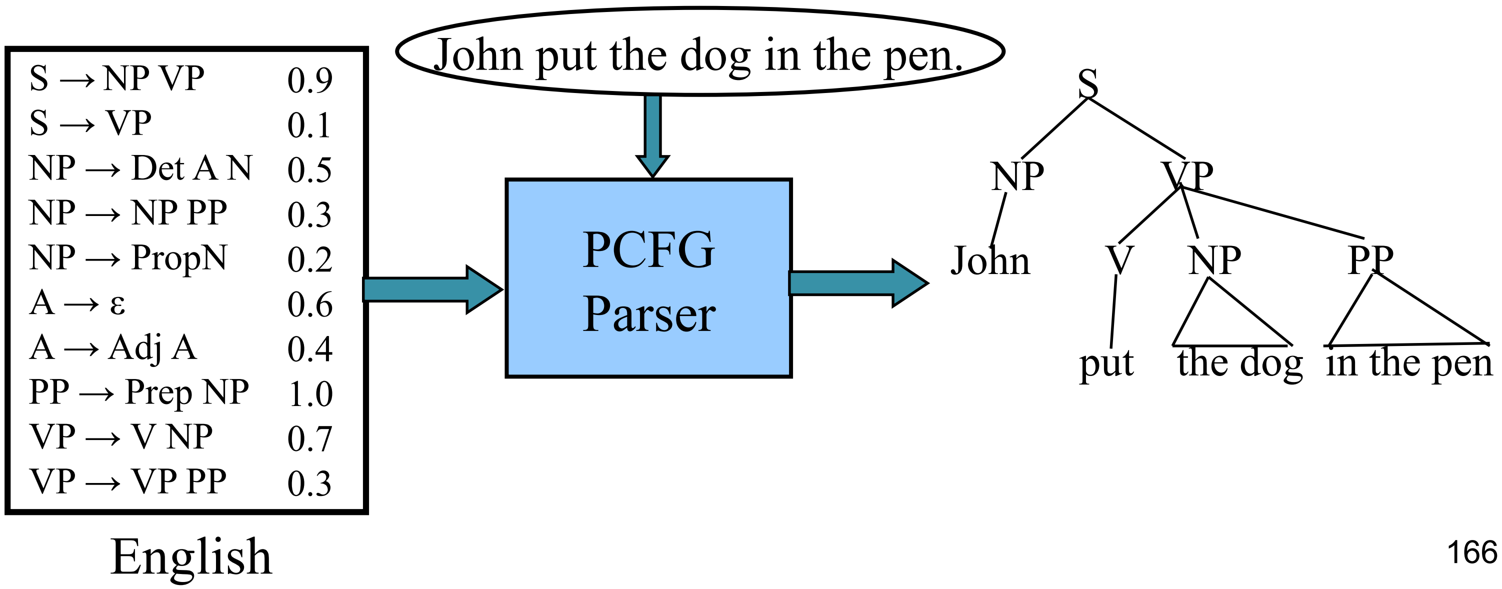
- The **Inside-Outside algorithm** is a version of EM for unsupervised learning of a PCFG.
  - Analogous to Baum-Welch (forward-backward) for HMMs
- Given the number of non-terminals, construct all possible CNF productions with these non-terminals and observed terminal symbols.
- Use EM to iteratively train the probabilities of these productions to locally maximize the likelihood of the data.
  - See Manning and Schütze text for details
- Experimental results are not impressive, but recent work imposes additional constraints to improve unsupervised grammar learning.

Vanilla PCFG Limitations

- Since probabilities of productions do not rely on specific words or concepts, only general structural disambiguation is possible (e.g. prefer to attach PPs to Nominals).
- Consequently, vanilla PCFGs cannot resolve syntactic ambiguities that require semantics to resolve, e.g. ate with fork vs. meatballs.
- In order to work well, PCFGs must be lexicalized, i.e. productions must be specialized to specific words by including their head-word in their LHS non-terminals (e.g. VP-ate).

Example of Importance of Lexicalization

- A general preference for attaching PPs to NPs rather than VPs can be learned by a vanilla PCFG.
- But the desired preference can depend on specific words.

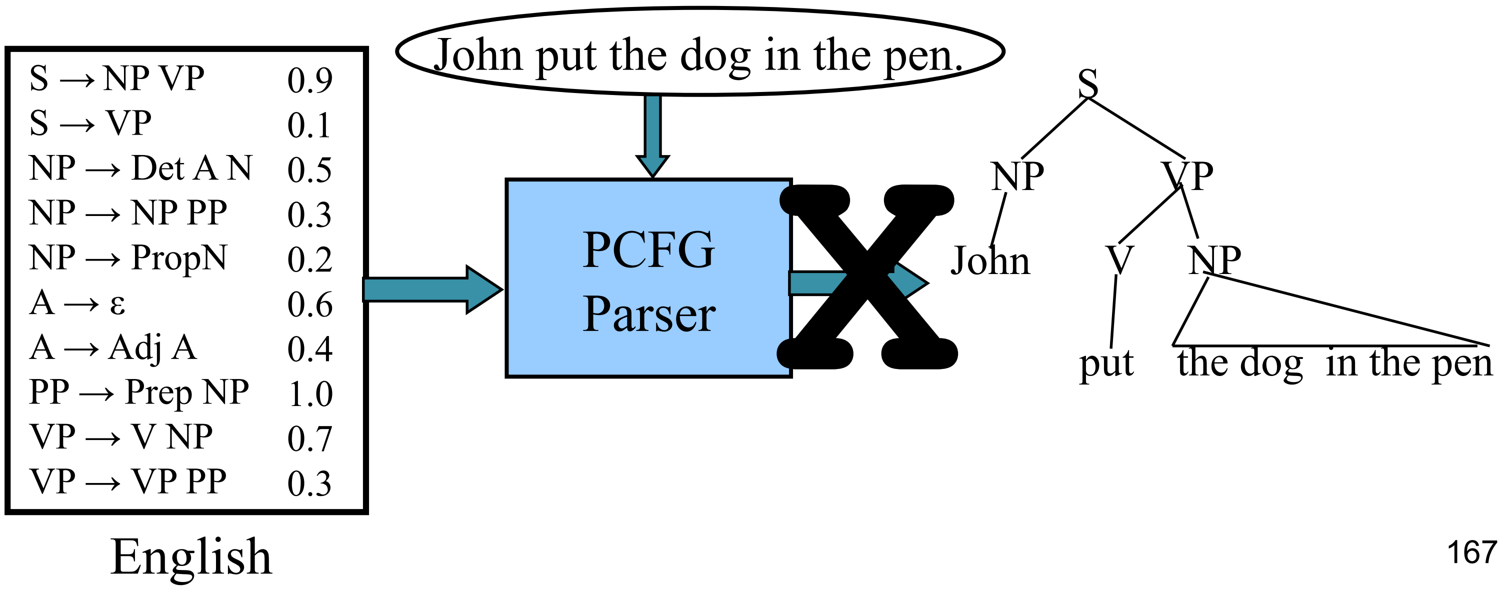


S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

English

Example of Importance of Lexicalization

- A general preference for attaching PPs to NPs rather than VPs can be learned by a vanilla PCFG.
- But the desired preference can depend on specific words.



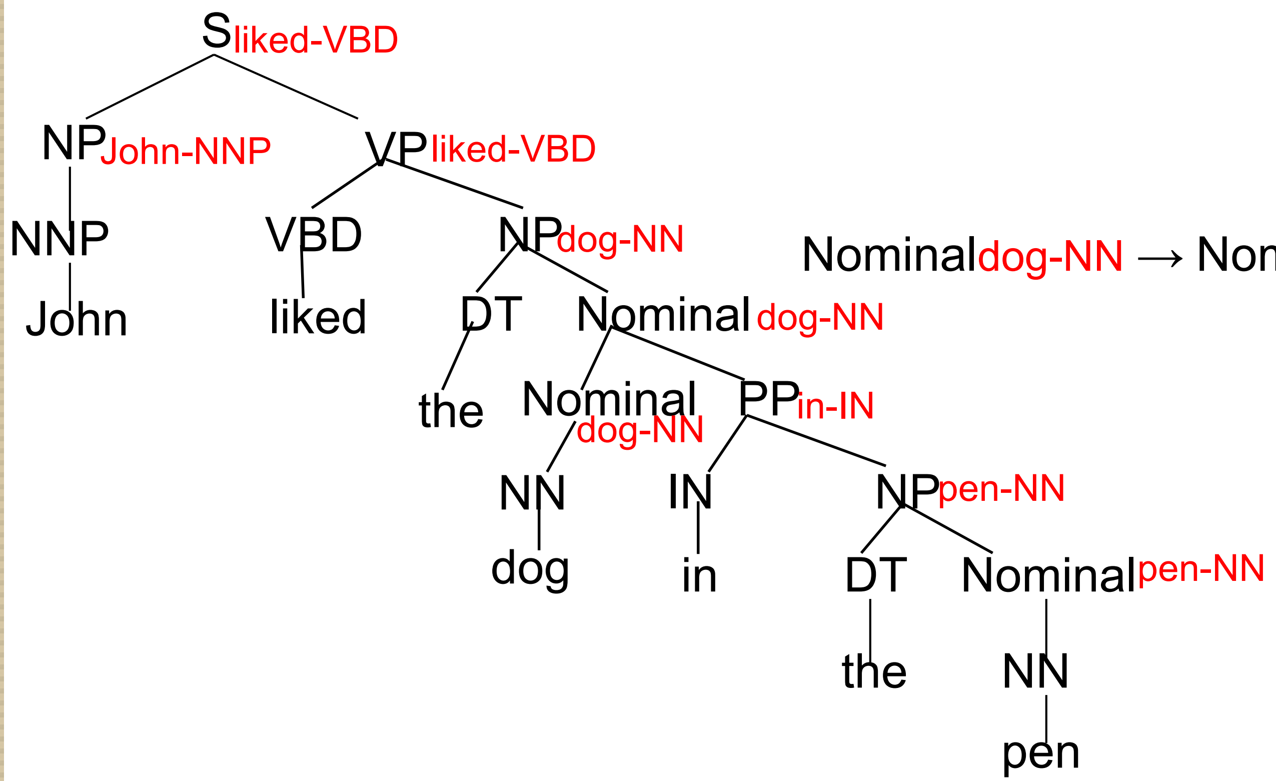
S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

Head Words

- Syntactic phrases usually have a word in them that is most "central" to the phrase.
- Linguists have defined the concept of a lexical **head** of a phrase.
- Simple rules can identify the head of any phrase by percolating head words up the parse tree.
  - Head of a VP is the main verb
  - Head of an NP is the main noun
  - Head of a PP is the preposition
  - Head of a sentence is the head of its VP

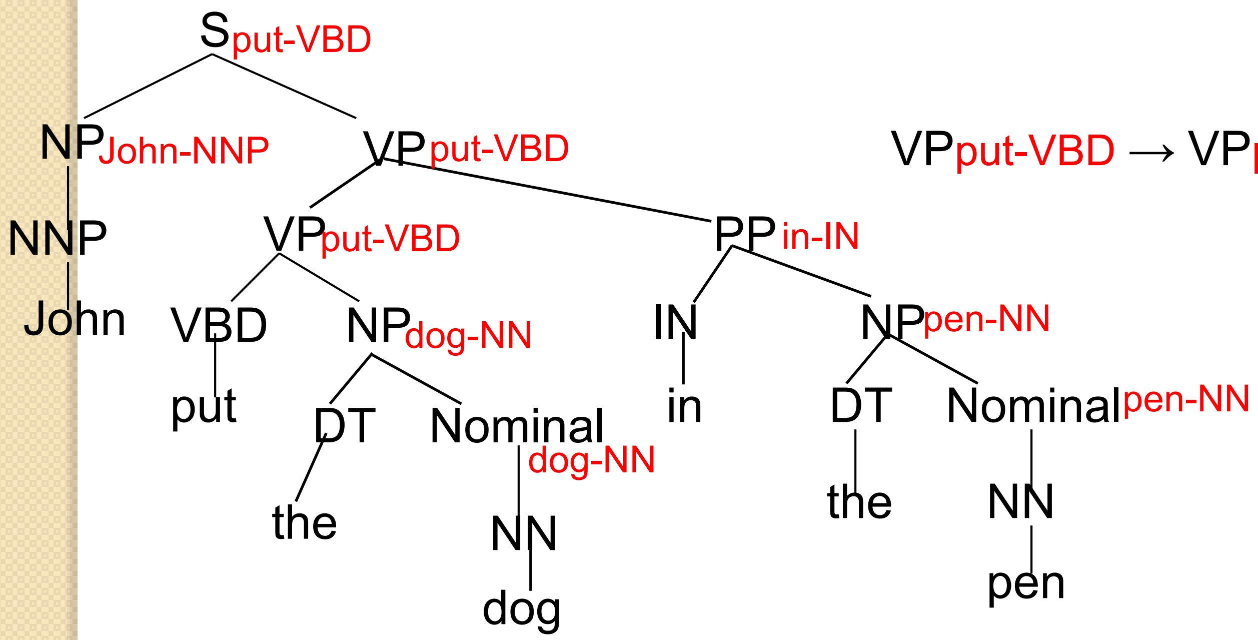
Lexicalized Productions

- Specialized productions can be generated by including the head word and its POS of each non-terminal as part of that non-terminal's symbol.



$\text{Nominal}_{\text{dog-NN}} \rightarrow \text{Nominal}_{\text{dog-NN}}\ \text{PP}_{\text{in-IN}}$

Lexicalized Productions



$\text{VP}_{\text{put-VBD}} \rightarrow \text{VP}_{\text{put-VBD}}\ \text{PP}_{\text{in-IN}}$

Parameterizing Lexicalized Productions

- Accurately estimating parameters on such a large number of very specialized productions could require enormous amounts of treebank data.
- Need some way of estimating parameters for lexicalized productions that makes reasonable independence assumptions so that accurate probabilities for very specific rules can be learned.

Collins Parser

- Collins (1999) parser assumes a simple generative model of lexicalized productions.
- Models productions based on context to the left and the right of the head daughter.
    - $\mathrm{LHS} \rightarrow \mathrm{L}_n \mathrm{L}_{n-1} \ldots \mathrm{L}_1 \mathrm{H}\ \mathrm{R}_1 \ldots \mathrm{R}_{m-1} \mathrm{R}_m$
- First generate the head (H) and then repeatedly generate left (L_i) and right (R_i) context symbols until the symbol STOP is generated.

Sample Production Generation

VPput-VBD → VBDput-VBD NPdog-NN PPin-IN

Note: Penn treebank tends to have fairly flat parse trees that produce long productions.

VPput-VBD → STOP VBDput-VBD NPdog-NN PPin-IN STOP

L_1 H R_1 R_2 R_3

P_L(STOP | VPput-VBD) * P_H(VBD | Vpput-VBD)*
P_R(NPdog-NN | VPput-VBD)*
P_R(PPin-IN | VPput-VBD) * P_R(STOP | VPput-VBD)

Estimating Production Generation Parameters

- Estimate P_H, P_L, and P_R parameters from treebank data.

$$P_R(\text{PPin-IN} \mid \text{VPput-VBD}) = \frac{\text{Count(PPin-IN right of head in a VPput-VBD production)}}{\text{Count(symbol right of head in a VPput-VBD)}}$$

$$P_R(\text{NPdog-NN} \mid \text{VPput-VBD}) = \frac{\text{Count(NPdog-NN right of head in a VPput-VBD production)}}{\text{Count(symbol right of head in a VPput-VBD)}}$$

- Smooth estimates by linearly interpolating with simpler models conditioned on just POS tag or no lexical info.

$$\begin{aligned} smP_R(\text{PPin-IN} \mid \text{VPput-VBD}) = {} & \lambda_1 P_R(\text{PPin-IN} \mid \text{VPput-VBD}) \\ & + (1-\lambda_1)\,(\lambda_2 P_R(\text{PPin-IN} \mid \text{VPVBD}) + \\ & (1-\lambda_2)\,P_R(\text{PPin-IN} \mid \text{VP})) \end{aligned}$$

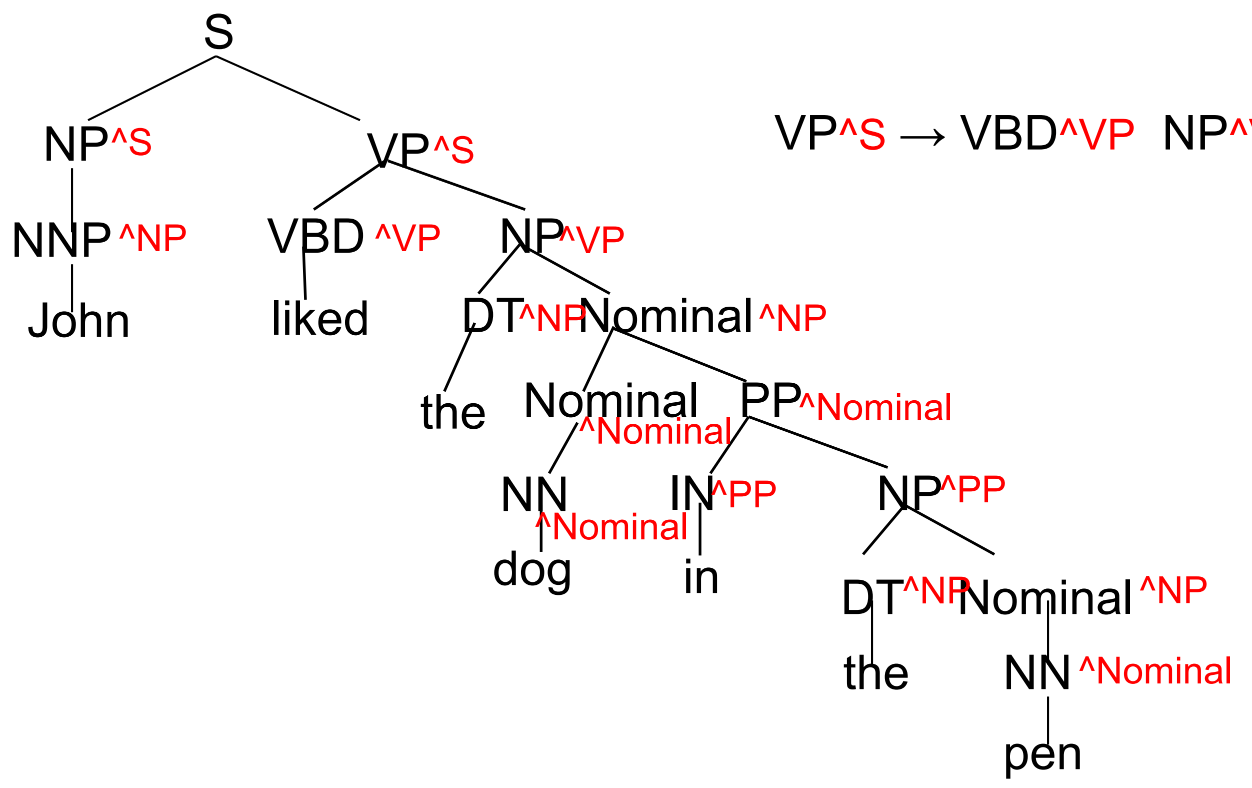
Missed Context Dependence

- Another problem with CFGs is that which production is used to expand a non-terminal is independent of its context.
- However, this independence is frequently violated for normal grammars.
    - NPs that are subjects are more likely to be pronouns than NPs that are objects.

Splitting Non-Terminals

- To provide more contextual information, non-terminals can be split into multiple new non-terminals based on their parent in the parse tree using parent annotation.
  - A subject NP becomes NP^S since its parent node is an S.
  - An object NP becomes NP^VP since its parent node is a VP

Parent Annotation Example



$VP\text{\textasciicircum}S \rightarrow VBD\text{\textasciicircum}VP\ \ NP\text{\textasciicircum}VP$

Split and Merge

- Non-terminal splitting greatly increases the size of the grammar and the number of parameters that need to be learned from limited training data.
- Best approach is to only split non-terminals when it improves the accuracy of the grammar.
- May also help to merge some non-terminals to remove some un-helpful distinctions and learn more accurate parameters for the merged productions.
- Method: Heuristically search for a combination of splits and merges that produces a grammar that maximizes the likelihood of the training treebank.

Treebanks

- English Penn Treebank: Standard corpus for testing syntactic parsing consists of 1.2 M words of text from the Wall Street Journal (WSJ).
- Typical to train on about 40,000 parsed sentences and test on an additional standard disjoint test set of 2,416 sentences.
- Chinese Penn Treebank: 100K words from the Xinhua news service.
- Other corpora existing in many languages, see the Wikipedia article "Treebank"

First WSJ Sentence

```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

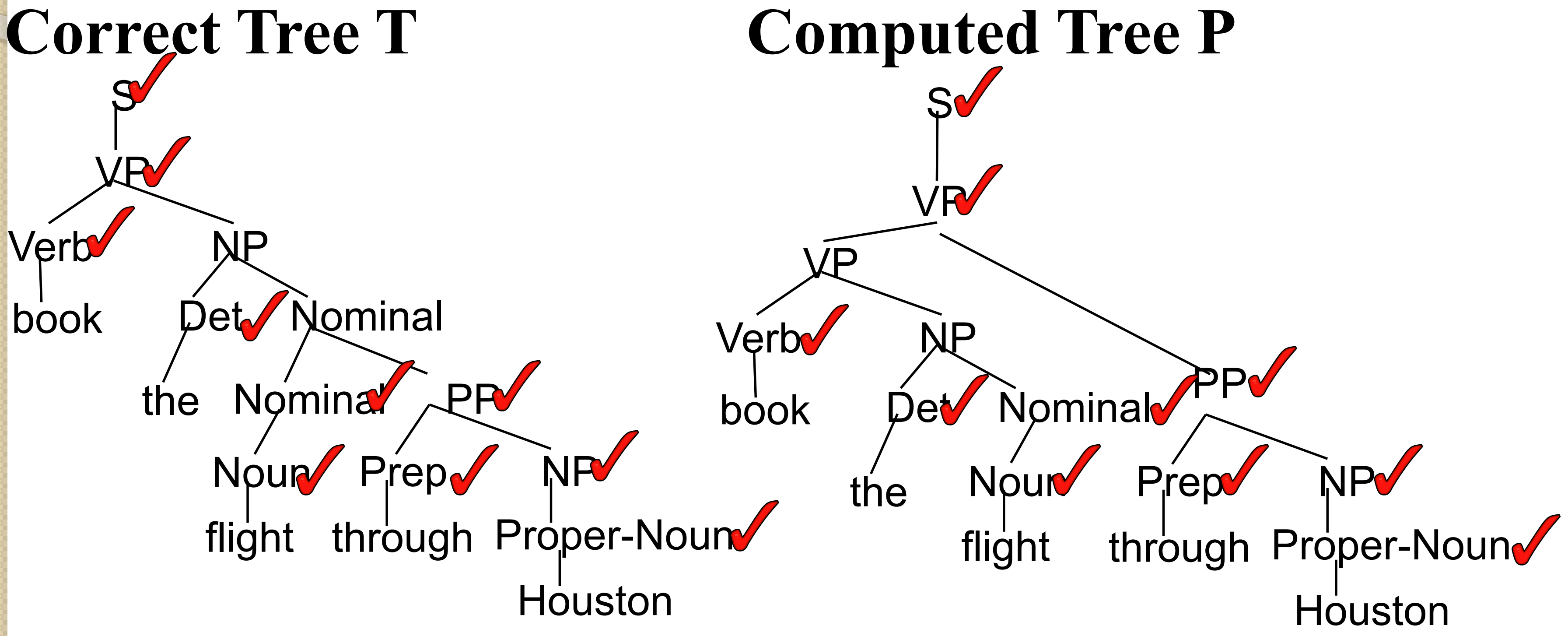
WSJ Sentence with Trace (NONE)

```
( (S
    (NP-SBJ (DT The) (NNP Illinois) (NNP Supreme) (NNP Court) )
    (VP (VBD ordered)
      (NP-1 (DT the) (NN commission) )
      (S
        (NP-SBJ (-NONE- *-1) )
        (VP (TO to)
          (VP
            (VP (VB audit)
              (NP
                (NP (NNP Commonwealth) (NNP Edison) (POS 's) )
                (NN construction) (NNS expenses) ))
            (CC and)
            (VP (VB refund)
              (NP (DT any) (JJ unreasonable) (NNS expenses) ))))))
    (. .) ))
```

Parsing Evaluation Metrics

- PARSEVAL metrics measure the fraction of the constituents that match between the computed and human parse trees. If *P* is the system's parse tree and *T* is the human parse tree (the "gold standard"):
  - ***Recall*** = (# correct constituents in *P*) / (# constituents in *T*)
  - ***Precision*** = (# correct constituents in *P*) / (# constituents in *P*)
- ***Labeled Precision*** and ***labeled recall*** require getting the non-terminal label on the constituent node correct to count as correct.
- F_1 is the harmonic mean of precision and recall.

Computing Evaluation Metrics



# Constituents: 12

# Constituents: 12

# Correct Constituents: 10

Recall = 10/12= 83.3%   Precision = 10/12=83.3% F_1 = 83.3%

Treebank Results

- Results of current state-of-the-art systems on the English Penn WSJ treebank are slightly greater than 90% labeled precision and recall.

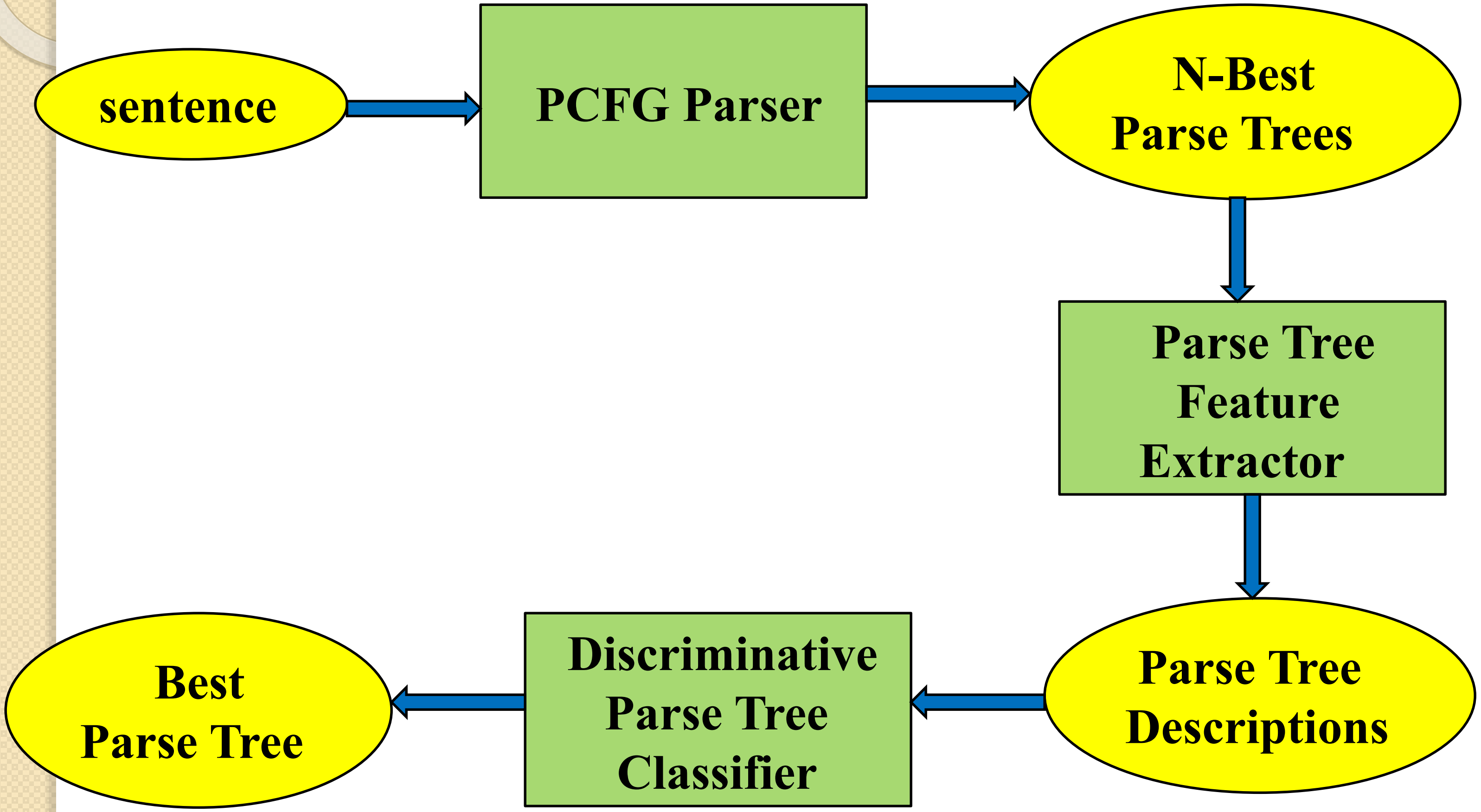
Discriminative Parse Reranking

- Motivation: Even when the top-ranked parse not correct, frequently the correct parse is one of those ranked highly by a statistical parser.
- Use a discriminative classifier that is trained to select the best parse from the N-best parses produced by the original parser.
- Reranker can exploit global features of the entire parse whereas a PCFG is restricted to making decisions based on local info.

2-Stage Reranking Approach

- Adapt the PCFG parser to produce an ***N-best list*** of the most probable parses in addition to the most-likely one.
- Extract from each of these parses, a set of global features that help determine if it is a good parse tree.
- Train a discriminative classifier (e.g. logistic regression) using the best parse in each N-best list as positive and others as negative.

Parse Reranking



Sample Parse Tree Features

- Probability of the parse from the PCFG.
- The number of parallel conjuncts.
  - “the bird in the tree and the squirrel on the ground”
  - “the bird and the squirrel in the tree”
- The degree to which the parse tree is right branching.
  - English parses tend to be right branching (cf. parse of “Book the flight through Houston”)
- Frequency of various tree fragments, i.e. specific combinations of 2 or 3 rules.

Evaluation of Reranking

- Reranking is limited by ***oracle accuracy***, i.e. the accuracy that results when an omniscient oracle picks the best parse from the N-best list.
- Typical current oracle accuracy is around F_1=97%
- Reranking can generally improve test accuracy of current PCFG models a percentage point or two.

Other Discriminative Parsing

- There are also parsing models that move from generative PCFGs to a fully discriminative model, e.g. ***max margin parsing*** (Taskar *et al*., 2004).
- There is also a recent model that efficiently reranks all of the parses in the complete (compactly-encoded) parse forest, avoiding the need to generate an N-best list (***forest reranking***, Huang, 2008).

Human Parsing

- Computational parsers can be used to predict human reading time as measured by tracking the time taken to read each word in a sentence.
- Psycholinguistic studies show that words that are more probable given the preceding lexical and syntactic context are read faster.
  - John put the dog in the pen with a lock.
  - John put the dog in the pen with a bone in the car.
  - John liked the dog in the pen with a bone.
- Modeling these effects requires an ***incremental*** statistical parser that incorporates one word at a time into a continuously growing parse tree.

Garden Path Sentences

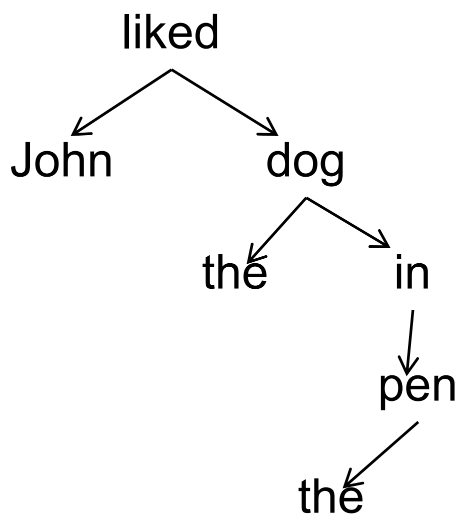
- People are confused by sentences that seem to have a particular syntactic structure but then suddenly violate this structure, so the listener is "lead down the garden path".
  - The horse raced past the barn fell.
    - vs. The horse raced past the barn broke his leg.
  - The complex houses married students.
  - The old man the sea.
  - While Anna dressed the baby spit up on the bed.
- Incremental computational parsers can try to predict and explain the problems encountered parsing such sentences.

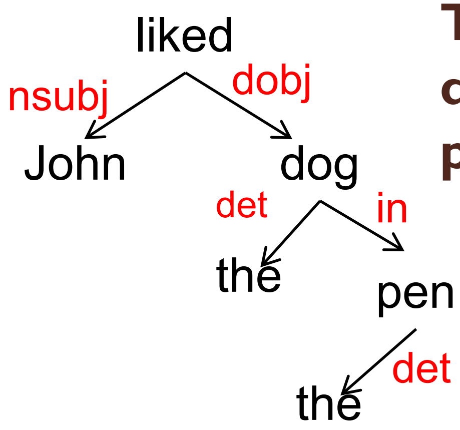
Center Embedding

- Nested expressions are hard for humans to process beyond 1 or 2 levels of nesting.
    - The rat the cat chased died.
    - The rat the cat the dog bit chased died.
    - The rat the cat the dog the boy owned bit chased died.
- Requires remembering and popping incomplete constituents from a stack and strains human short-term memory.
- Equivalent "tail embedded" (tail recursive) versions are easier to understand since no stack is required.
    - The boy owned a dog that bit a cat that chased a rat that died.

Dependency Grammars

- An alternative to phrase-structure grammar is to define a parse as a directed graph between the words of a sentence representing ***dependencies*** between the words.

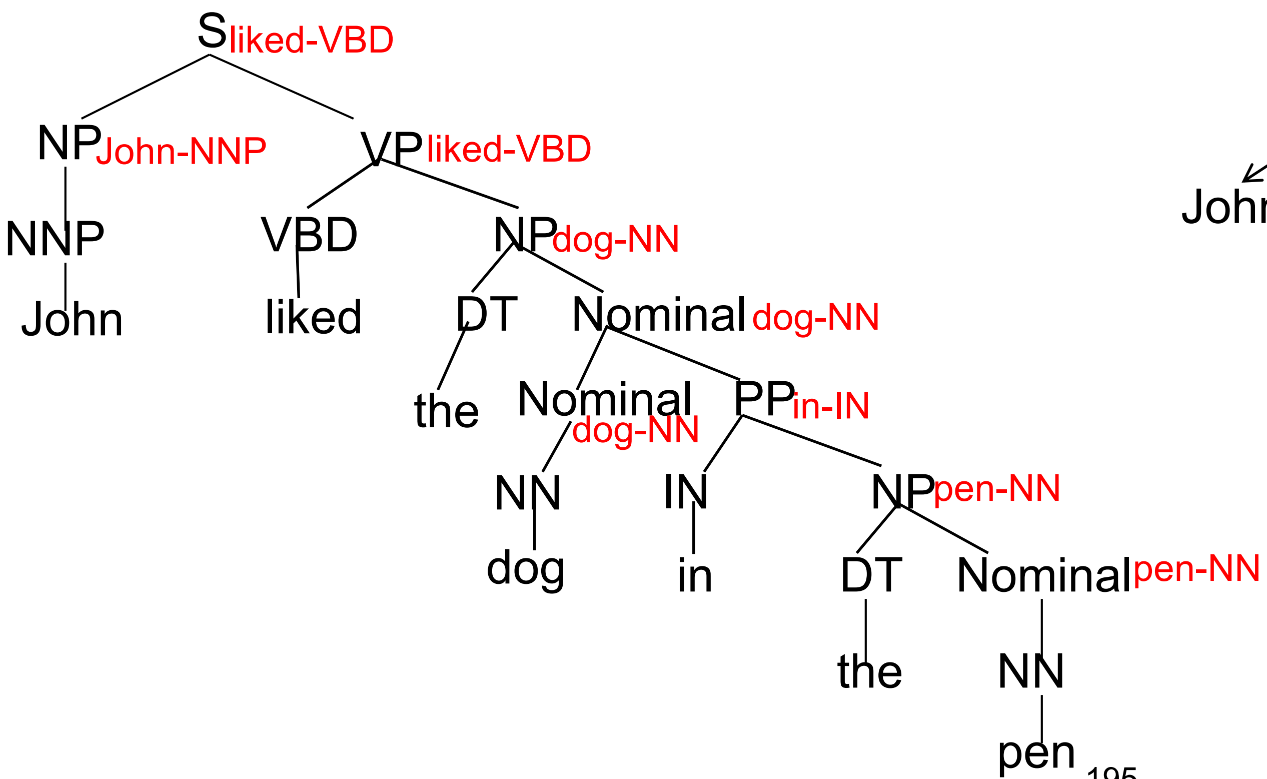


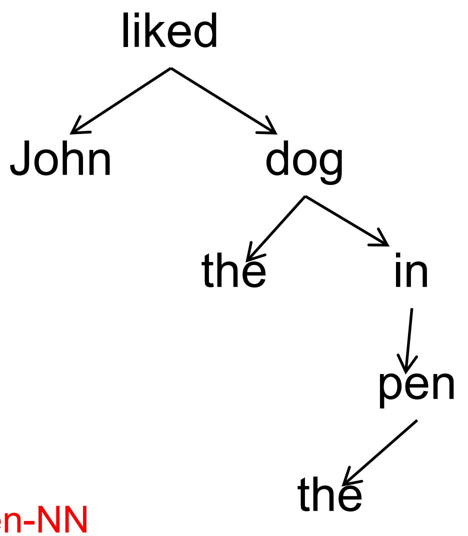


Typed dependency parse

Dependency Graph from Parse Tree

- Can convert a phrase structure parse to a dependency tree by making the head of each non-head child of a node depend on the head of the head child.





Unification Grammars

- In order to handle agreement issues more effectively, each constituent has a list of features such as number, person, gender, etc. which may or not be specified for a given constituent.
- In order for two constituents to combine to form a larger constituent, their features must ***unify***, i.e. consistently combine into a merged set of features.
- Expressive grammars and parsers (e.g. HPSG) have been developed using this approach and have been partially integrated with modern statistical models of disambiguation.

Mildly Context-Sensitive Grammars

- Some grammatical formalisms provide a degree of context-sensitivity that helps capture aspects of NL syntax that are not easily handled by CFGs.
- Tree Adjoining Grammar (TAG) is based on combining tree fragments rather than individual phrases.
- Combinatory Categorial Grammar (CCG) consists of:
  - Categorial Lexicon that associates a syntactic and semantic category with each word.
  - Combinatory Rules that define how categories combine to form other categories.

Statistical Parsing Conclusions

- Statistical models such as PCFGs allow for probabilistic resolution of ambiguities.
- PCFGs can be easily learned from treebanks.
- Lexicalization and non-terminal splitting are required to effectively resolve many ambiguities.
- Current statistical parsers are quite accurate but not yet at the level of human-expert agreement.