Speech and Language Processing

Chapter 12

Constituency Parsing

Today

- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity
  - CKY parsing
  - (Earley)
  - Shallow

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

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices
- We'll start with some basic (meaning bad) methods before moving on to the one that you need to know

For Now

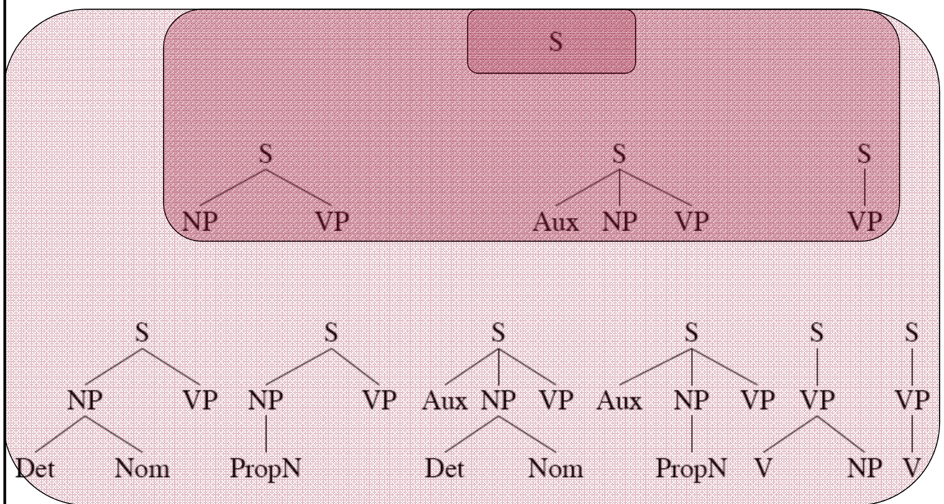
- Assume…
  - You have all the words already in some buffer
  - The input isn't POS tagged
  - We won't worry about morphological analysis
  - All the words are known

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

Top-Down Search

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

Top Down Space



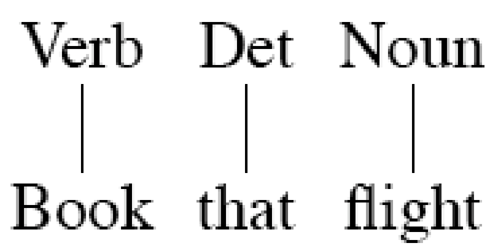
Bottom-Up Parsing

- 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.

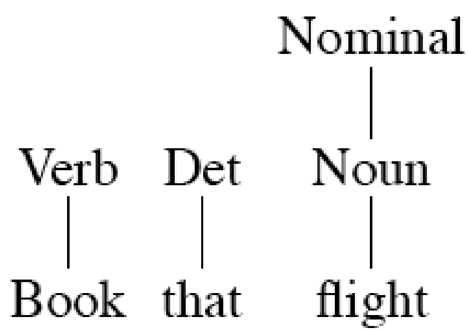
Bottom-Up Search

Book that flight

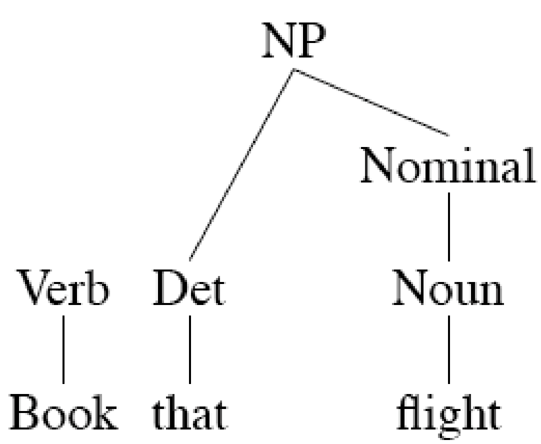
Bottom-Up Search



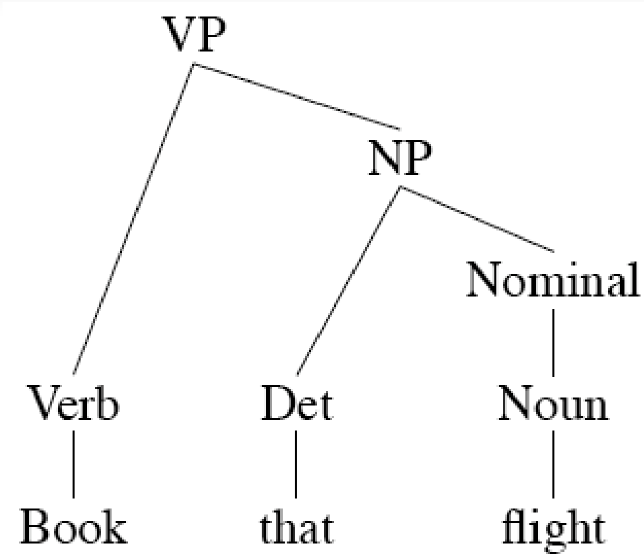
Bottom-Up Search



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

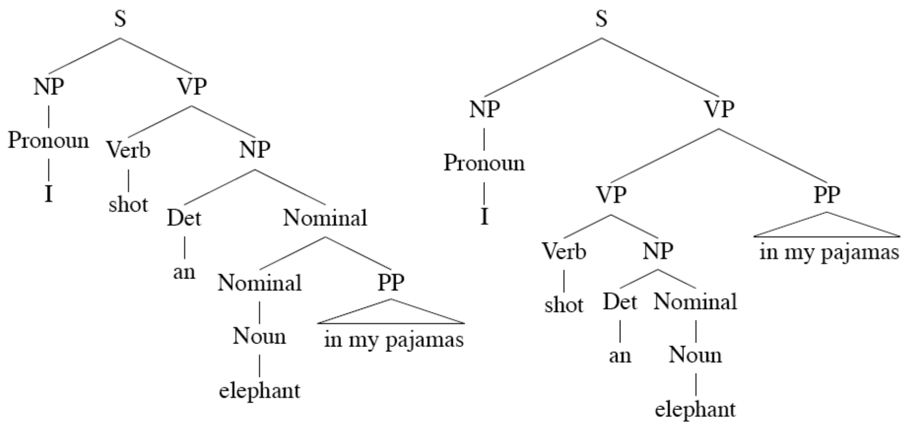
Control

- 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
  - Shared subproblems

Ambiguity



Example types of ambiguity

- POS
- Attachment
  - PP
  - Coordination (*old dogs and cats*)

Shared Sub-Problems

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

Review

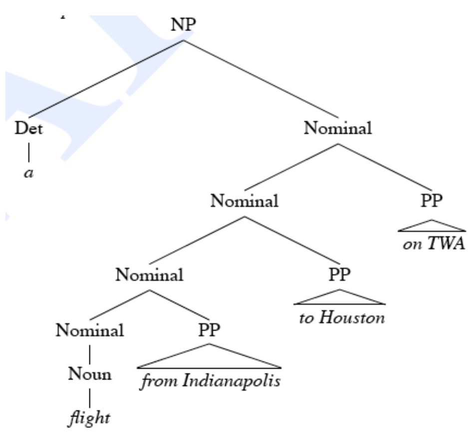
- Formal Grammars
  - CFG – what, why, why not?
  - Dependency
  - Treebanks
- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity

"The old dog the footsteps of the young."

S → NP VP	VP → V
S → Aux NP VP	*VP -> V PP*
S -> VP	*PP -> Prep NP*
NP → Det Nom	N → old | dog | footsteps | young
NP →PropN	V → dog | eat | sleep | bark | meow
Nom -> Adj N	Aux → does | can
Nom → N	Prep →from | to | on | of
Nom → N Nom	PropN → Fido | Felix
Nom → Nom PP	Det → that | this | a | the
VP → V NP	Adj -> old | happy| young

Shared Sub-Problems

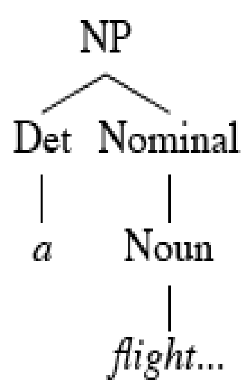
- Consider
  - A flight from Indianapolis to Houston on TWA



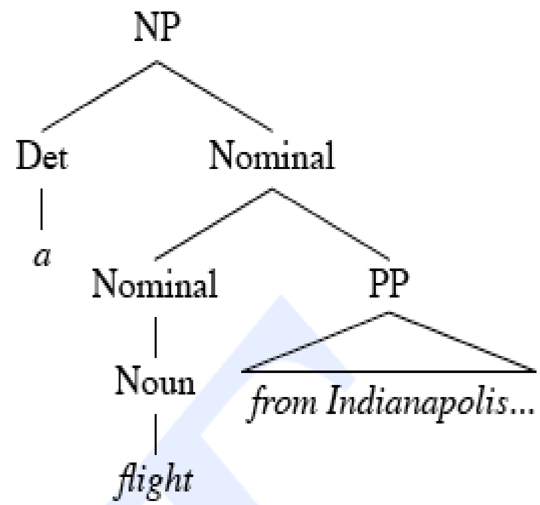
Shared Sub-Problems

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

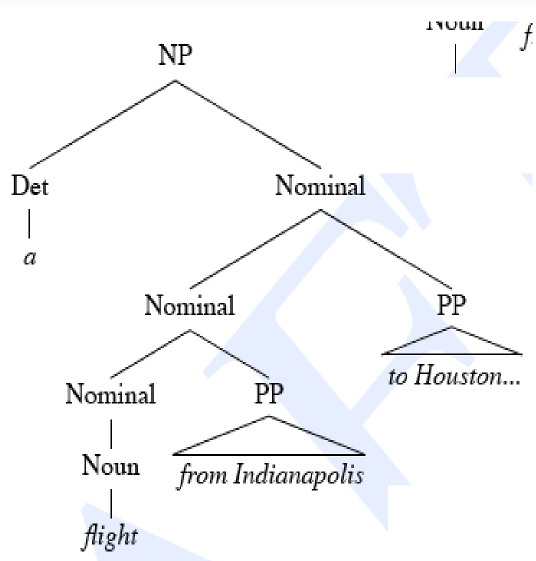
Shared Sub-Problems



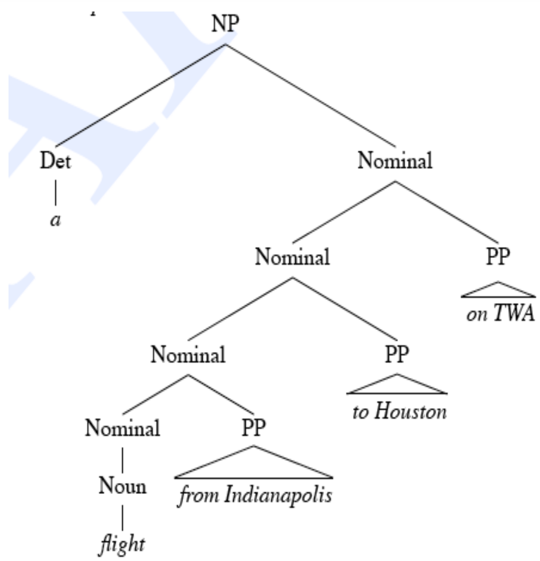
Shared Sub-Problems



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
  - Efficiently store ambiguous structures with shared sub-parts.
- Two approaches roughly correspond to bottom-up and top-down approaches.
  - CKY
  - Earley

CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule *A* $\rightarrow$ *BC*
  - If there is an A somewhere in the input 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
    - Ie. The B splits from the C someplace.

Problem

- What if your grammar isn't binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary… any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

Problem

- More specifically, we want our rules to be of the form

  A $\rightarrow$ B C

  Or

  A $\rightarrow$ *w*

  That is, rules can expand to either 2 non-terminals or to a single terminal.

Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules.
  - So… *S → A B C turns into*

  S → X C and

  X → A B

  Where X is a symbol that doesn't occur anywhere else in the the grammar.

Sample L1 Grammar

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

CNF Conversion

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

CKY

- So 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 an *S*
- If we build the table bottom-up, we'll 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 $i<k<j$

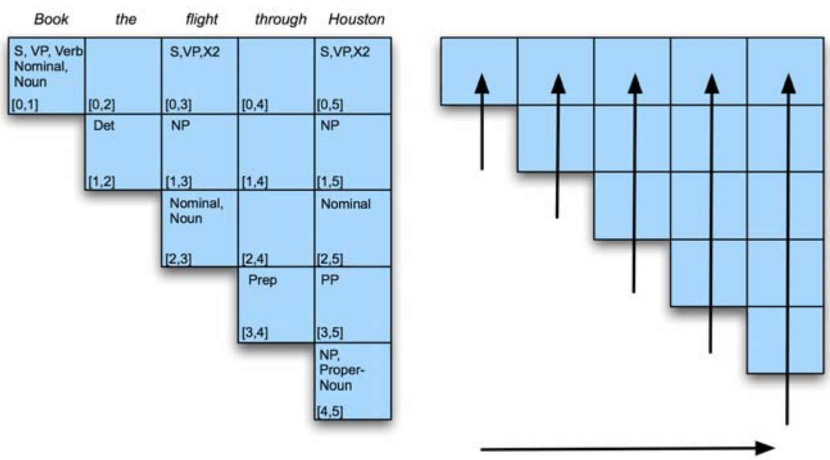
CKY

- So to fill the table loop over the cell[i,j] values in some systematic way
  - What constraint should we put on that systematic search?
  - For each cell, loop over the appropriate k values to search for things to add.

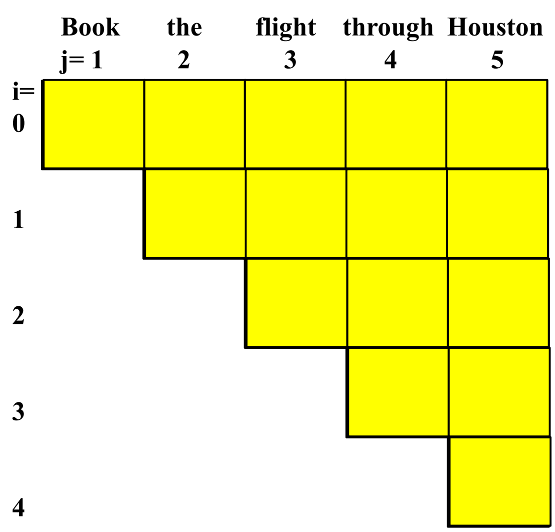
Note

- 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

Example

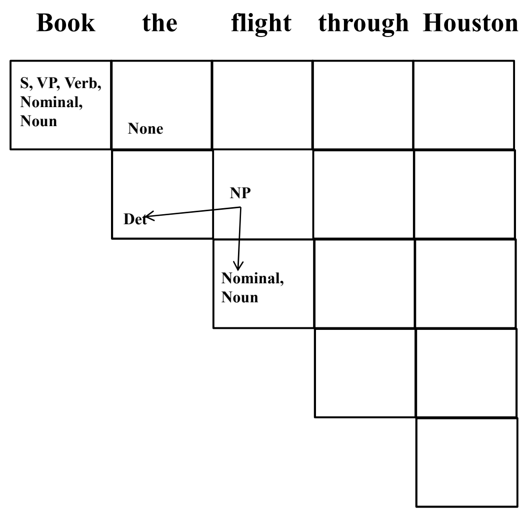


CKY Parser



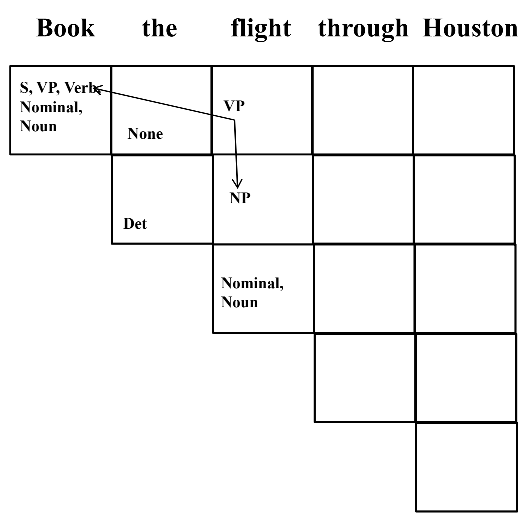
Cell[*i,j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

CKY Parser



41

CKY Parser



42

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP		
	Det	NP		
		Nominal, Noun		

43

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP, X2		
	Det	NP		
		Nominal, Noun		

44

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	

45

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	PP
				NP ProperNoun

46

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

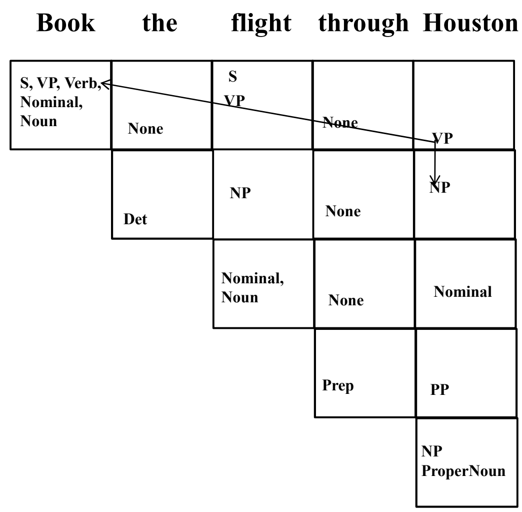
47

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

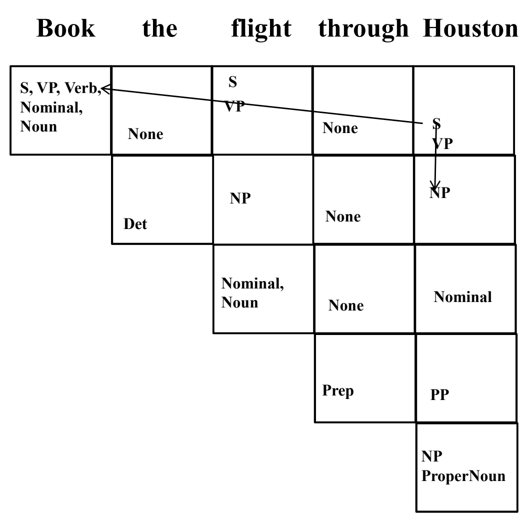
48

CKY Parser



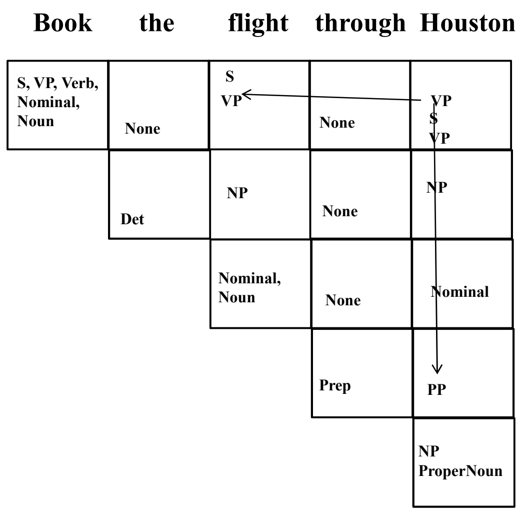
49

CKY Parser

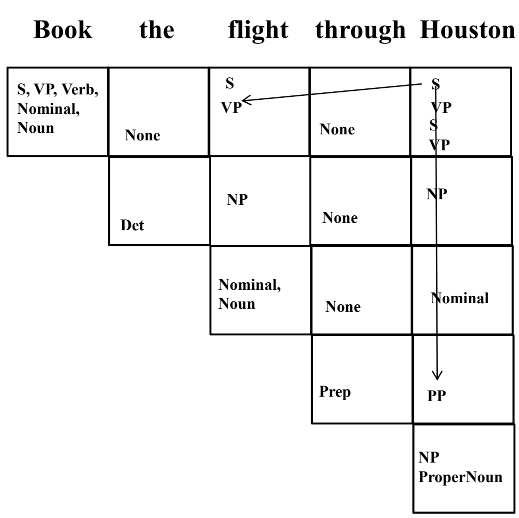


50

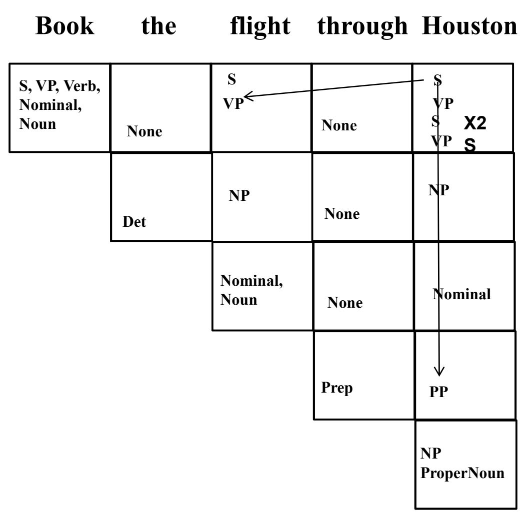
CKY Parser



CKY Parser

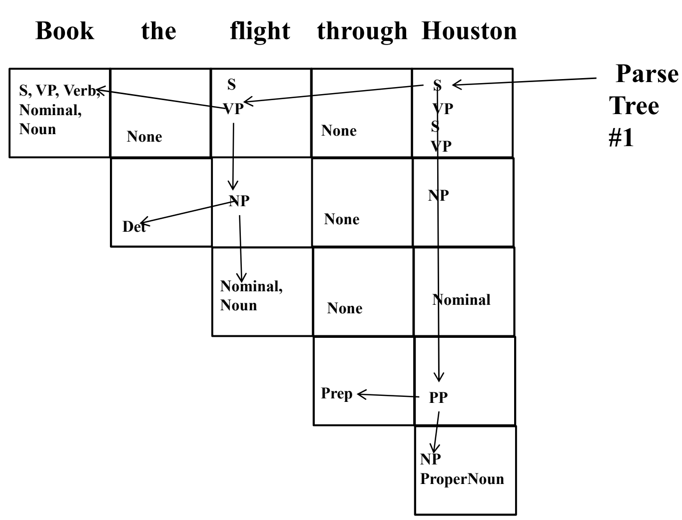


CKY Parser



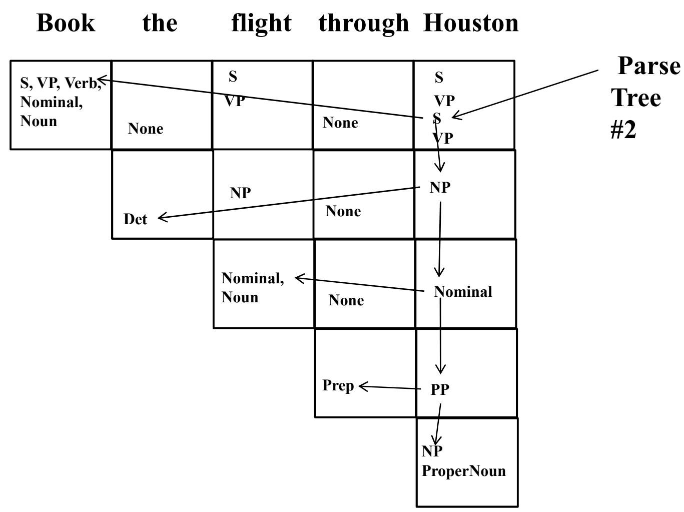
53

CKY Parser



54

CKY Parser

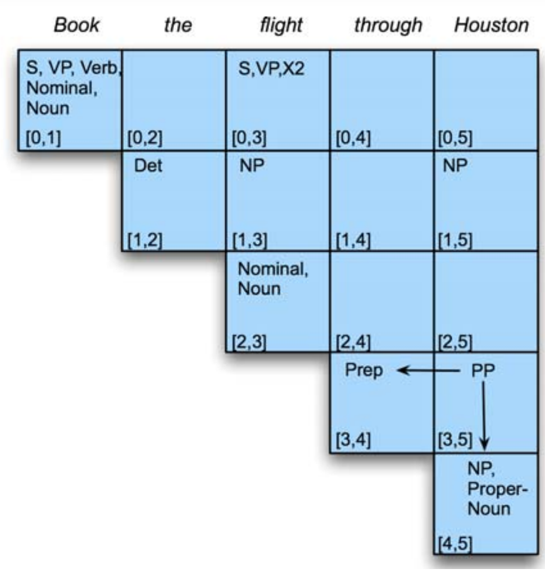


Example

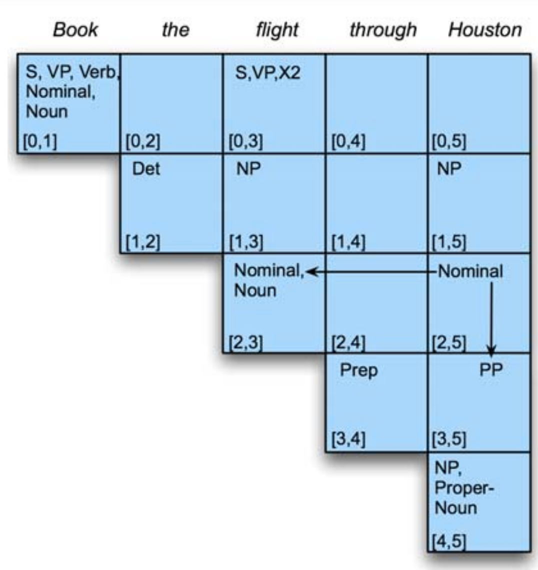
	Book	the	flight	through	Houston
	S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
		Det [1,2]	NP [1,3]	[1,4]	[1,5]
			Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
				Prep [3,4]	[3,5]
					NP, Proper-Noun [4,5]

Filling column 5

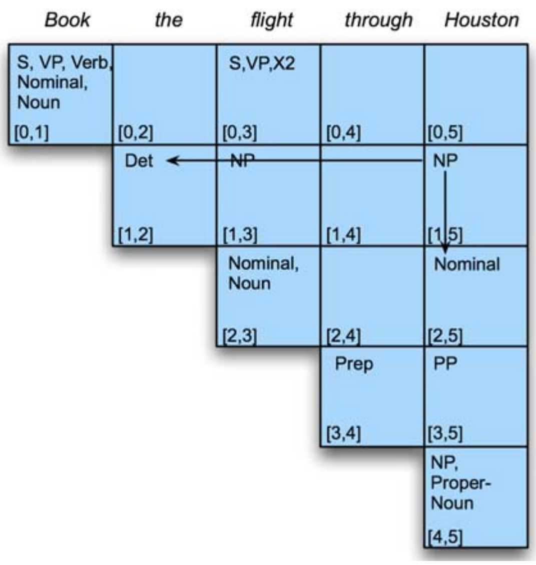
Example



Example

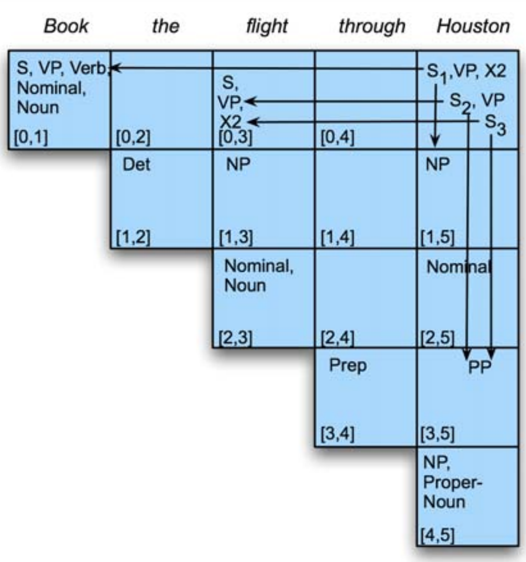


Example



9/25/2019 Speech and Language Processing - Jurafsky and Martin

Example



9/25/2019 Speech and Language Processing - Jurafsky and Martin

CKY Notes

- Since it's bottom up, CKY populates the table with a lot of phantom 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.

Earley Parsing

- Allows arbitrary CFGs
- Top-down control
- Fills a table in a single sweep over the input
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

Back to Ambiguity

- Did we solve it?

9/25/2019 Speech and Language Processing - Jurafsky and Martin 63

Ambiguity

- No…
  - Both CKY and Earley will result in multiple S structures for the [0,N] table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

9/25/2019 Speech and Language Processing - Jurafsky and Martin 64

Ambiguity

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

Shallow or Partial Parsing

- Sometimes we don't need a complete parse tree
  - Information extraction
  - Question answering
- But we would like more than simple POS sequences

Chunking

- Find major but unembedded constituents like NPs, VPs, AdjPs, PPs
  - Most common task: NP chunking of base NPs
  - [NP I] saw [NP the man] on [NP the hill] with [NP a telescope]
  - No attempt to identify full NPs – no recursion, no post-head words
  - No overlapping constituents
  - E.g., if we add PPs or VPs, they may consist only of their heads, e.g. [PP on]

Approaches: RE Chunking

- Use regexps to identify constituents, e.g.
  - NP → (DT) NN* NN
  - Find longest matching chunk
  - Hand-built rules
  - No recursion but can cascade to approximate true CF parser, aggregating larger and larger constituents

Approaches: Tagging for Chunking

- Require annotated corpus
- Train classifier to classify each element of input in sequence (e.g. IOB Tagging)
  - B (beginning of sequence)
  - I (internal to sequence)
  - O (outside of any sequence)
  - No end-of-chunk coding – it's implicit
  - Easier to detect the beginning than the end

  Book/B_VP that/B_NP flight/I_NP quickly/O

Summary and Limitations

- Sometimes shallow parsing is enough for task
- Performance quite accurate

Distribution of Chunks in CONLL Shared Task

Label	Category	Proportion (%)	Example
NP	Noun Phrase	51	*The most frequently cancelled flight*
VP	Verb Phrase	20	*may not arrive*
PP	Prepositional Phrase	20	*to Houston*
ADVP	Adverbial Phrase	4	*earlier*
SBAR	Subordinate Clause	2	*that*
ADJP	Adjective Phrase	2	*late*

Summing Up

- Parsing as search: what search strategies to use?
  - Top down
  - Bottom up
  - How to combine?
- How to parse as little as possible
  - Dynamic Programming
- Shallow Parsing

72