Parsing

Recommended Reading: Ch. 12-14th

Jurafsky & Martin 2nd edition

PI Disclosure: This set includes adapted material from Rada Mihalcea, Raymond Mooney and Dan Jurafsky

Today

- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity
  - CKY parsing
  - Early algorithm

Parsing with CFGs

- Parsing with CFGs refers to the task of assigning proper trees to input strings
- Proper: 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 with CFGs

- As with everything of interest, parsing involves a search
- We'll start with some basic methods:
  - Top down parsing
  - Bottom up parsing
- Real algorithms:
  - Cocke-Kasami-Younger (CKY)
  - Earley parser

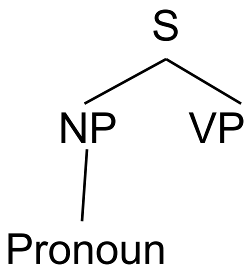
For Now

- 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

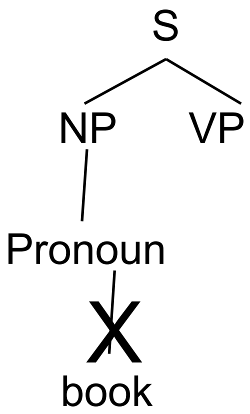
Top-Down Search

- 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.
- As an example let’ s parse the sentence:
  - Book that flight

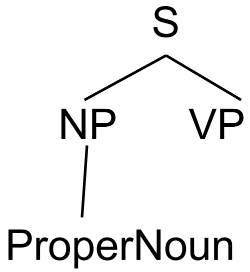
Top Down Parsing



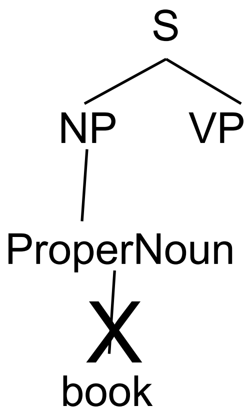
Top Down Parsing



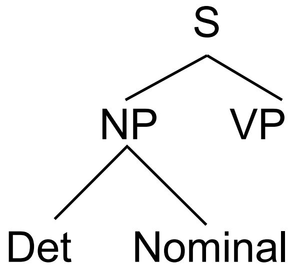
Top Down Parsing



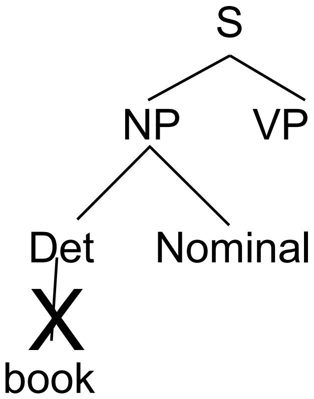
Top Down Parsing



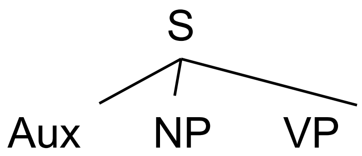
Top Down Parsing



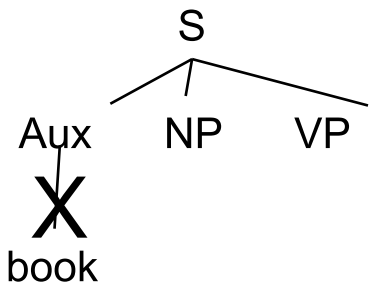
Top Down Parsing



Top Down Parsing



Top Down Parsing



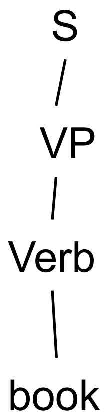
Top Down Parsing

S
/
VP

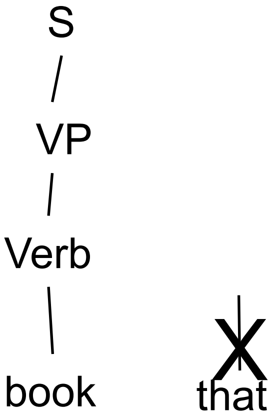
Top Down Parsing

S
/
VP
/
Verb

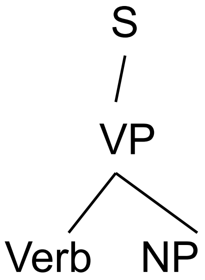
Top Down Parsing



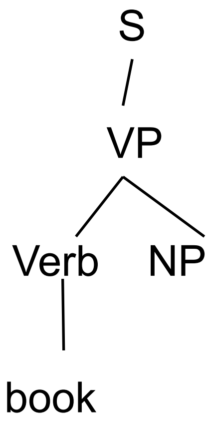
Top Down Parsing



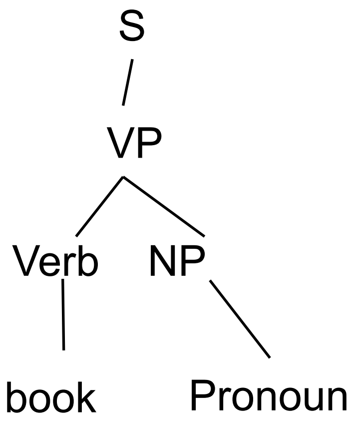
Top Down Parsing



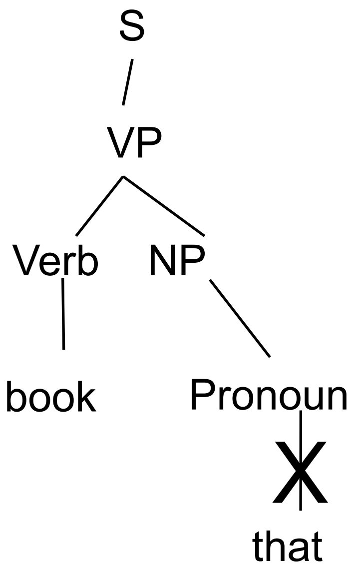
Top Down Parsing



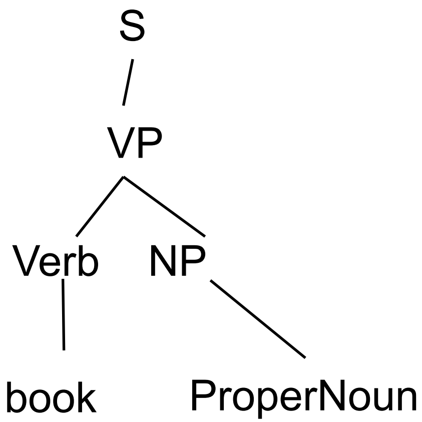
Top Down Parsing



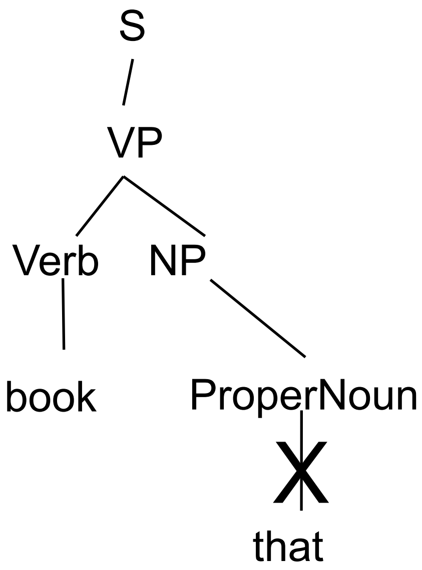
Top Down Parsing



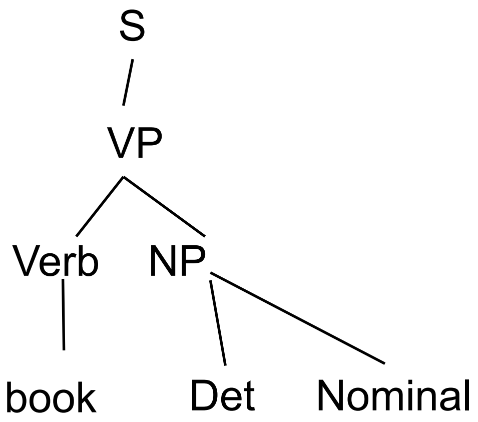
Top Down Parsing



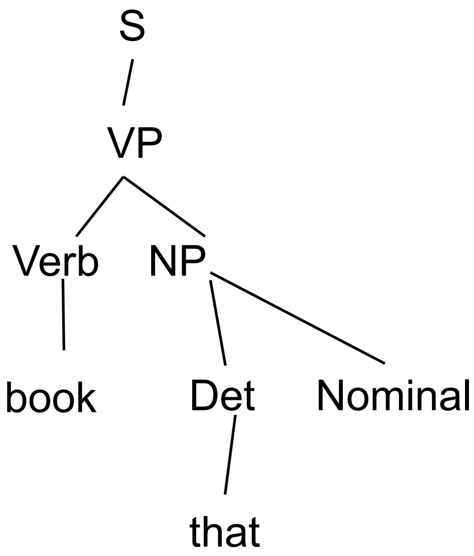
Top Down Parsing



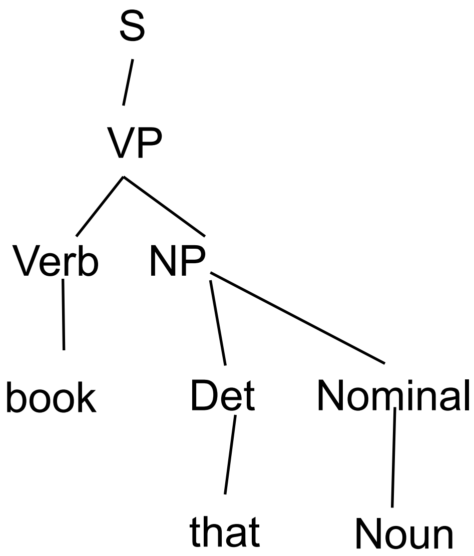
Top Down Parsing



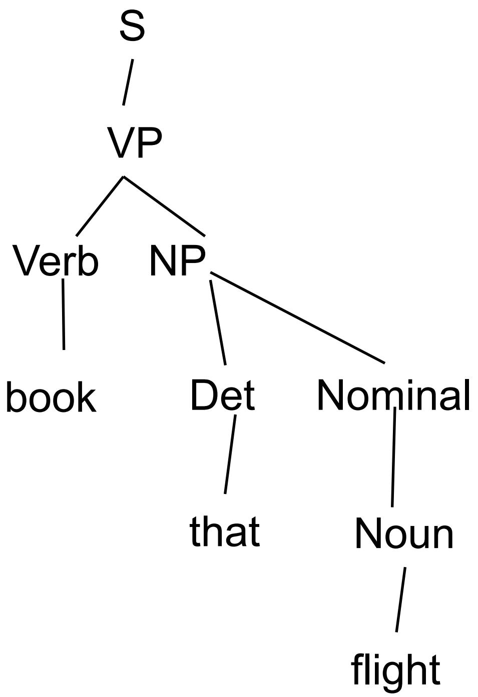
Top Down Parsing



Top Down Parsing



Top Down Parsing



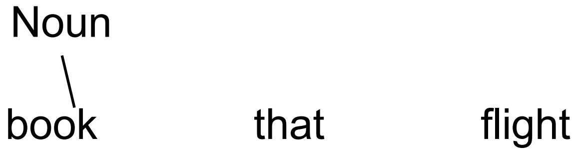
Bottom-Up Parsing

- 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.

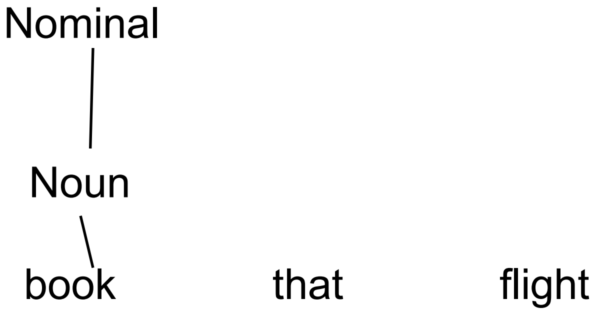
Bottom Up Parsing

book that flight

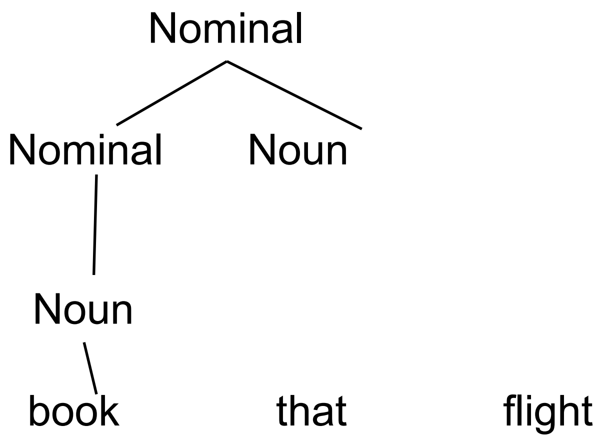
Bottom Up Parsing



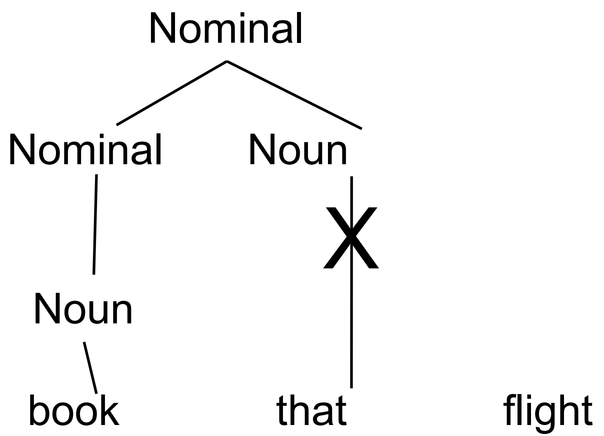
Bottom Up Parsing



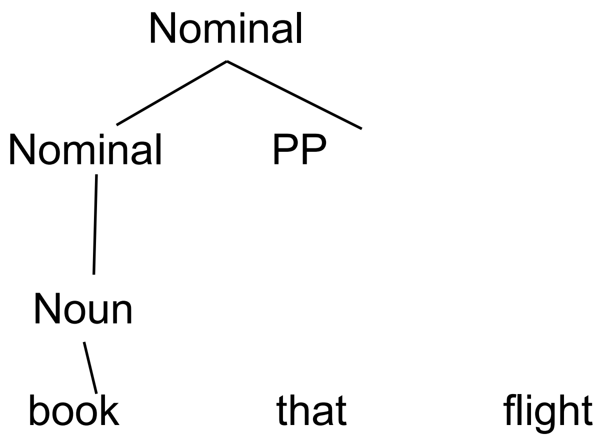
Bottom Up Parsing



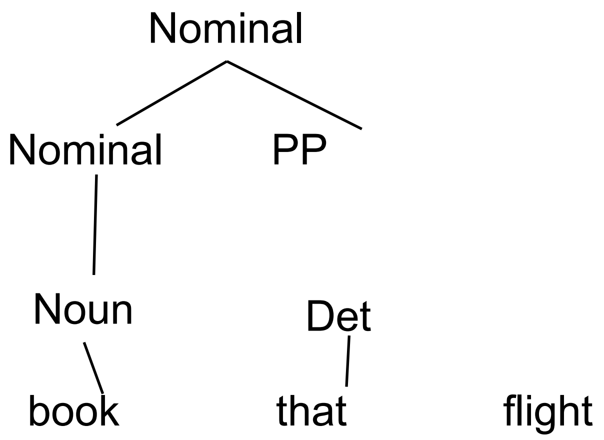
Bottom Up Parsing



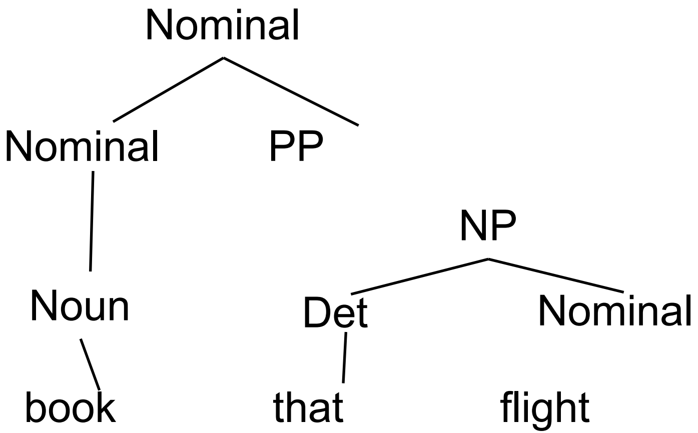
Bottom Up Parsing



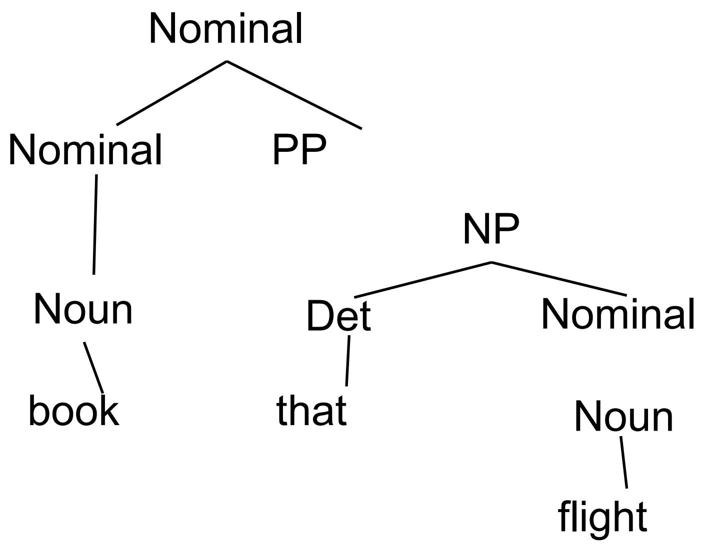
Bottom Up Parsing



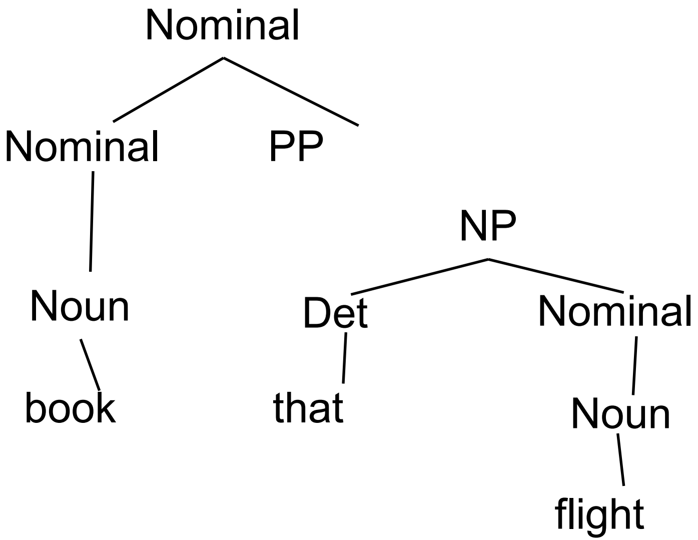
Bottom Up Parsing



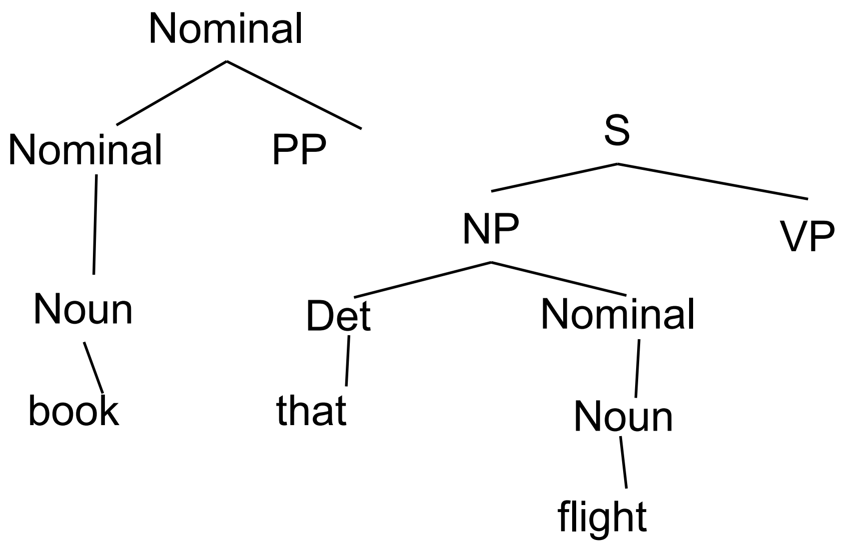
Bottom Up Parsing



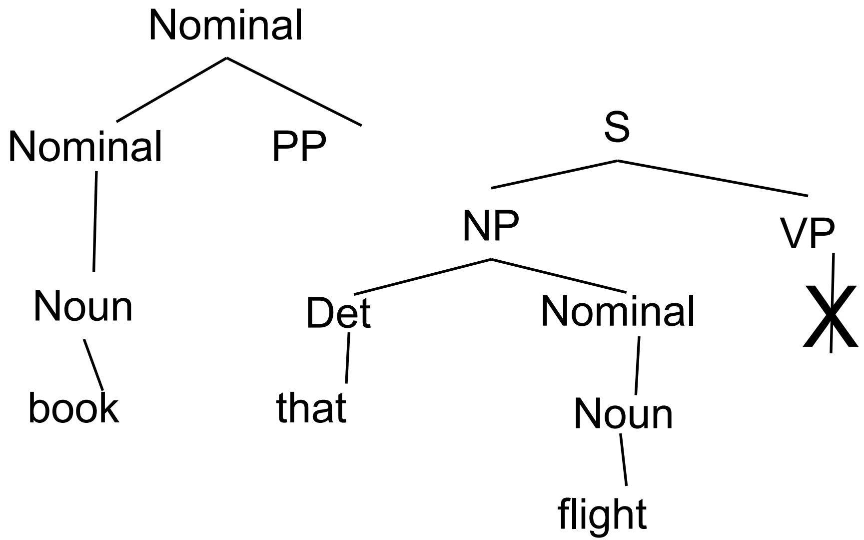
Bottom Up Parsing



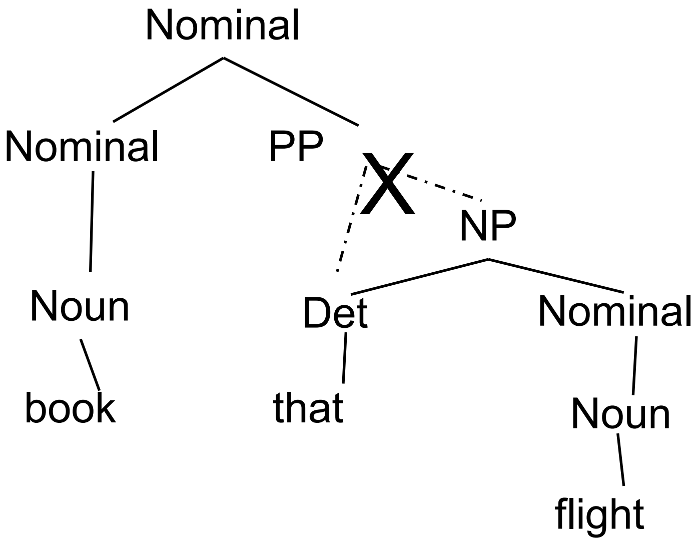
Bottom Up Parsing



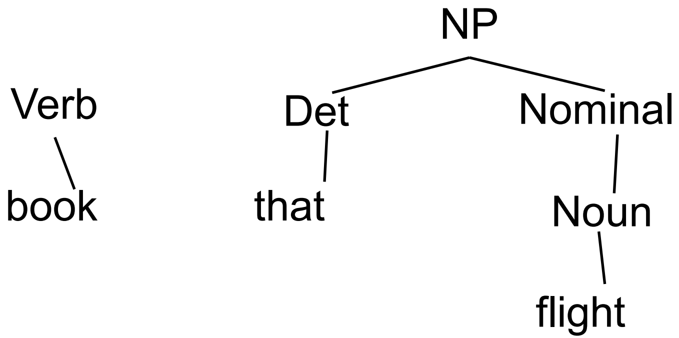
Bottom Up Parsing



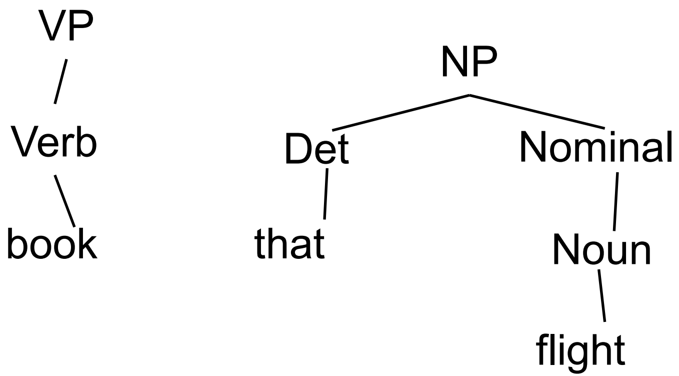
Bottom Up Parsing



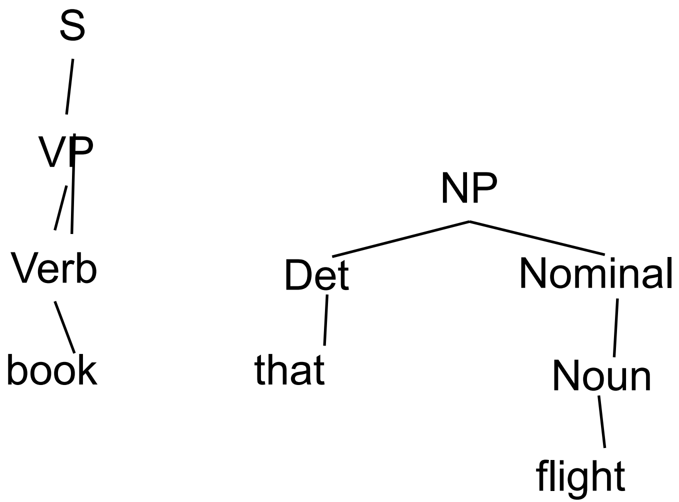
Bottom Up Parsing



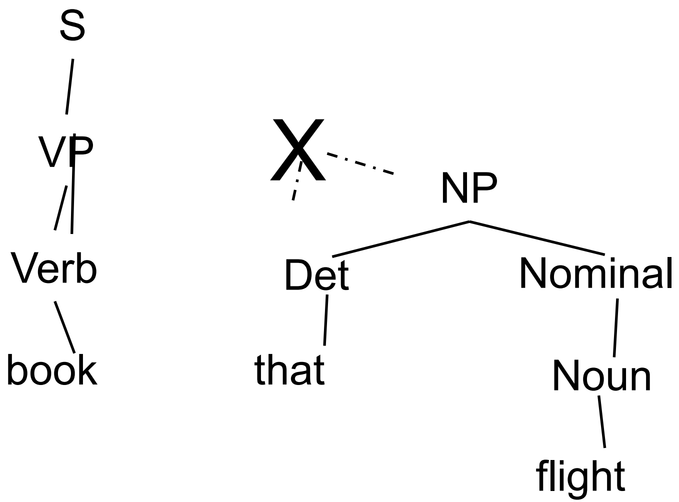
Bottom Up Parsing



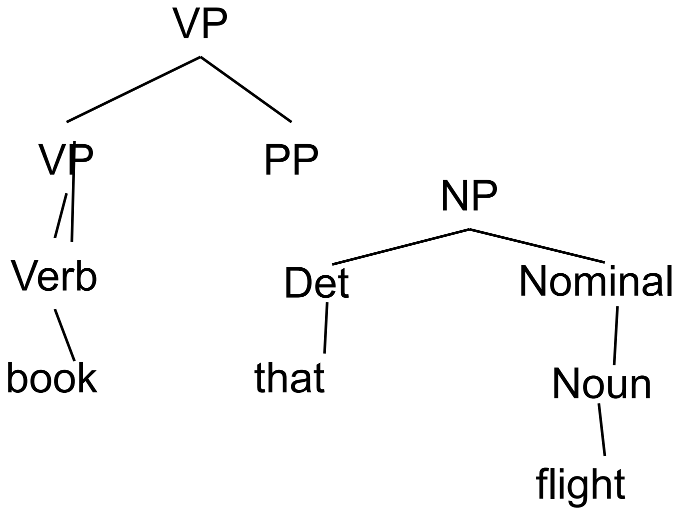
Bottom Up Parsing



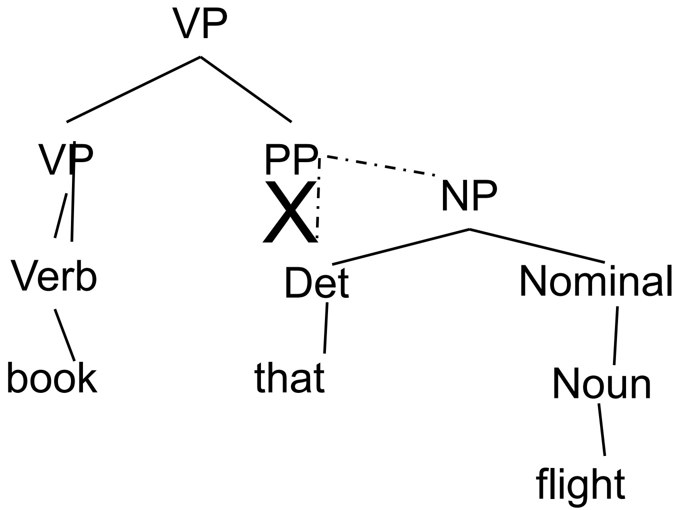
Bottom Up Parsing



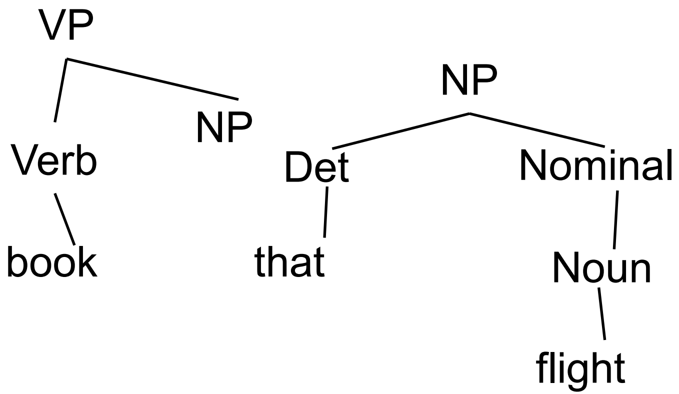
Bottom Up Parsing



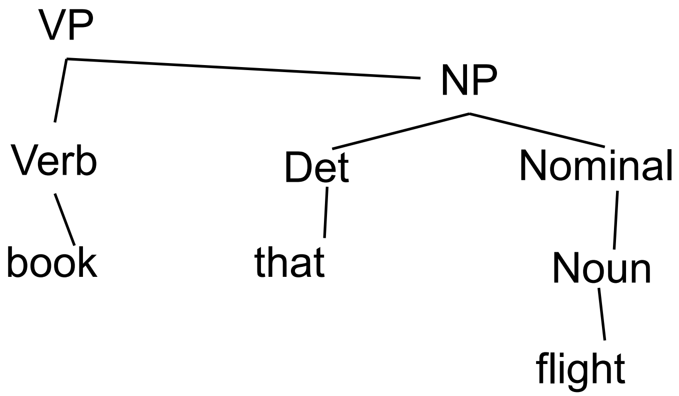
Bottom Up Parsing



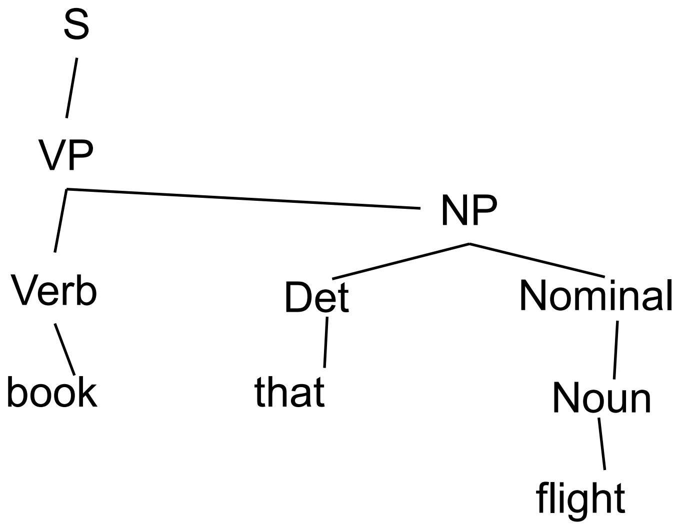
Bottom Up Parsing



Bottom Up Parsing



Bottom Up Parsing



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

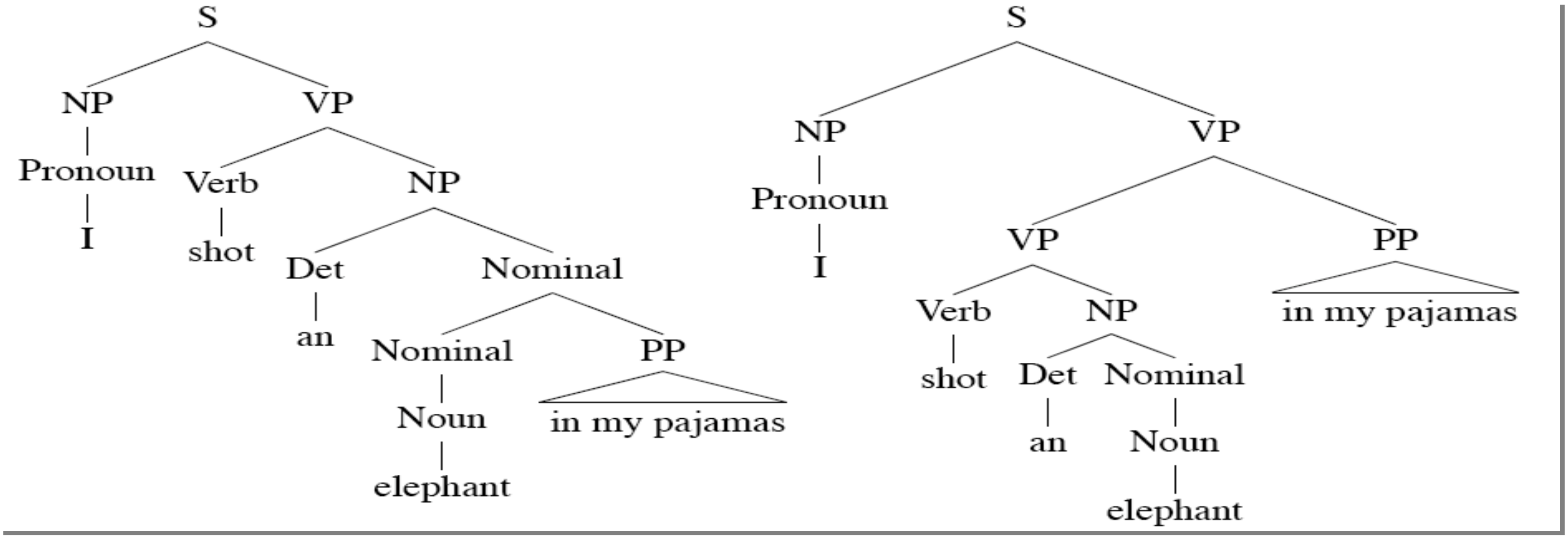
Control

- 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 ambiguity
  - Attachment ambiguity
  - Coordination ambiguity

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

Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Efficiently store ambiguous structures with shared sub-parts.
- We'll cover two approaches that roughly correspond to top-down and bottom-up approaches:
  - Cocke-Kasami-Younger (CKY)
  - Earley parser

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 s.t. $i<k<j$
    - 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?

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… $S \rightarrow A\ B\ C$ *turns into*

$S \rightarrow X\ C$ *and*

$X \rightarrow A\ B$

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 Parsing: Intuition

- Consider the rule $D \rightarrow w$
  - Terminal (word) forms a constituent
  - Trivial to apply
- Consider the rule $A \rightarrow B\ C$
  - If there is an A somewhere in the input then there must be a B followed by a C in the input
  - First, precisely define span $[\ i, j\ ]$
  - If A spans from i to j in the input then there must be some k such that $i<k<j$
  - Easy to apply: we just need to try out different values for k

CKY Parsing: Table

- Any constituent can conceivably span [i, j] for all $0 \le i < j \le N$, where N = length of input string
  - We need an $N \times N$ table to keep track of all spans…
  - But we only need half of the table
- Semantics of table: cell [i, j] contains A iff A spans i to j in the input string
  - Of course, must be allowed by the grammar!

CKY Parsing: Table-Filling

- So let's fill this table…
  - And look at the cell [*0, N*]: which means?
- But how?

CKY Parsing: Table-Filling

- In order for A to span [i, j]:
  - $A \rightarrow B\ C$ is a rule in the grammar, and
  - There must be a B in [i, k] and a C in [k, j] for some $i<k<j$
- Operationally:
  - To apply rule $A \rightarrow B\ C$, look for a B in [i, k] and a C in [k, j]
  - In the table: look left in the row and down in the column

CKY Algorithm

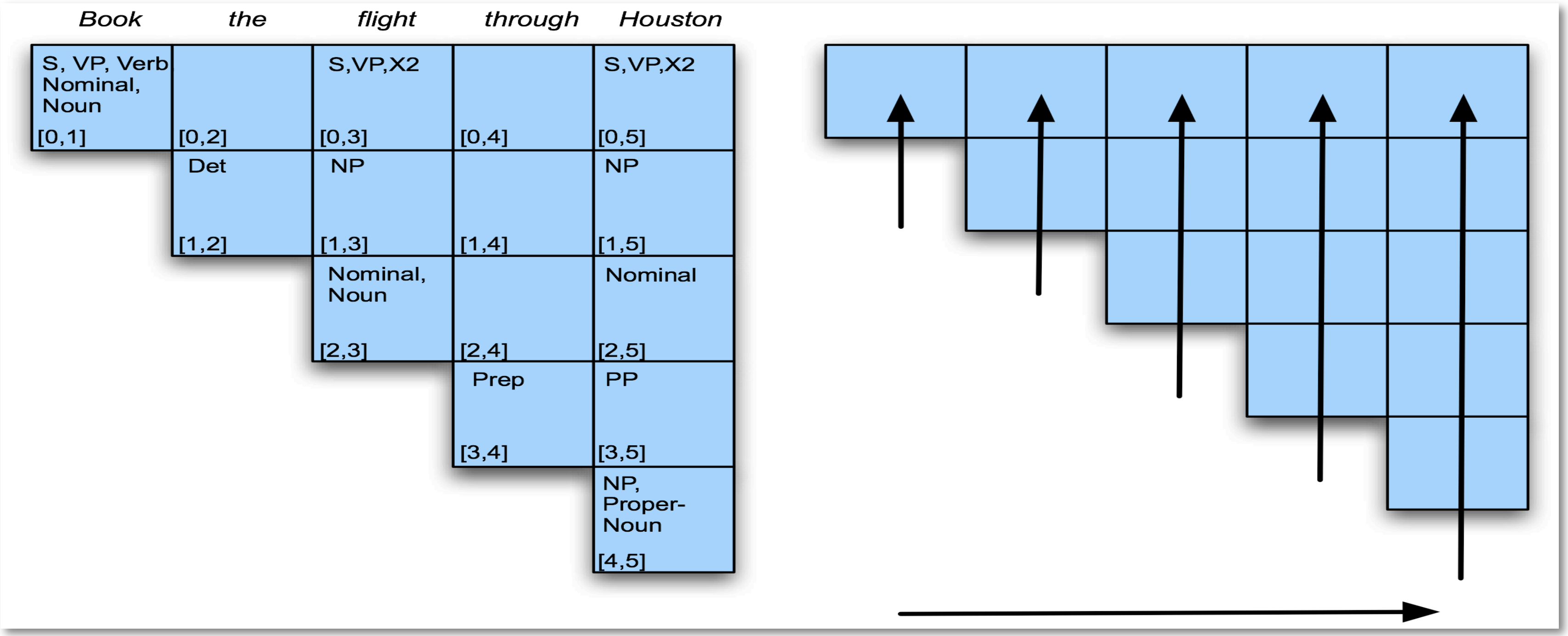
function CKY-PARSE(*words, grammar*) **returns** *table*

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

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 left to right a word at a time
    - Known as online

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]

CKY Parser Example

CKY Notes

- Since it’s bottom up, CKY populates the table with a lot of phantom constituents.
  - 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.
- Is there a parsing algorithm for arbitrary CFGs that combines dynamic programming and top-down control?

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 a set of states (s_i):
    - A grammar rule
    - Information about progress made in completing the sub-tree represented by the rule
    - Span of the sub-tree

States/Locations

- $S \to \bullet\ VP$, [0,0]
- A VP is predicted at the start of the sentence
- $NP \to Det \bullet Nominal$, [1,2]
- An NP is in progress; the Det goes from 1 to 2
- $VP \to V\ NP \bullet$, [0,3]
- A VP has been found starting at 0 and ending at 3

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 and is complete. That is,
  - $S \rightarrow \alpha \bullet [0,N]$
- If that' s the case you' re done.

Earley

- So sweep through the table from 0 to N…
  - 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. Generate new predictions
     3. Go to step 2
  3. When you' re out of words, look at the chart to see if you have a winner

Earley

- Proceeds incrementally, left-to-right
  - Before it reads word 5, it has already built all hypotheses that are consistent with first 4 words
  - Reads word 5 & attaches it to immediately preceding hypotheses. Might yield new constituents that are then attached to hypotheses immediately preceding *them* …
  - E.g., attaching D to $A \to B\ C\ .\ D\ E$ gives $A \to B\ C\ D\ .\ E$
  - Attaching E to that gives $A \to B\ C\ D\ E\ .$
  - Now we have a complete A that we can attach to hypotheses immediately preceding the A, etc.

Earley

- Three Main Operators:
  - Predictor: If state s_i has a non terminal to the right we add to s_i all alternatives to generate the non terminal
  - Scanner: when there is POS to the right of the dot in s_i then scanner will try to match it with an input word and if a successful match is found the new state will be added to s_i
  - Completer: if the dot is at the end of the production then the completer looks for all states looking for the non terminal that has been found and advances the position of the dot for those states.

Core Earley Code

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

  ENQUEUE((γ → • 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 Code

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

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

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

Example

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

Chart[0] $_0$Book $_1$ the $_2$ flight $_3$

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

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

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	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
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

Charts[2] and [3]

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
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
S37	$VP \rightarrow VP \bullet PP$	[0,3]	Completer

Efficiency

- For such a simple example, there seems to be a lot of useless stuff in there.
- Why?

  - It's predicting things that aren't consistent with the input
  - That's the flipside to the CKY problem.

Details

- As with CKY that isn't a parser until we add the backpointers so that each state knows where it came from.

Back to Ambiguity

- Did we solve it?

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.

Ambiguity

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