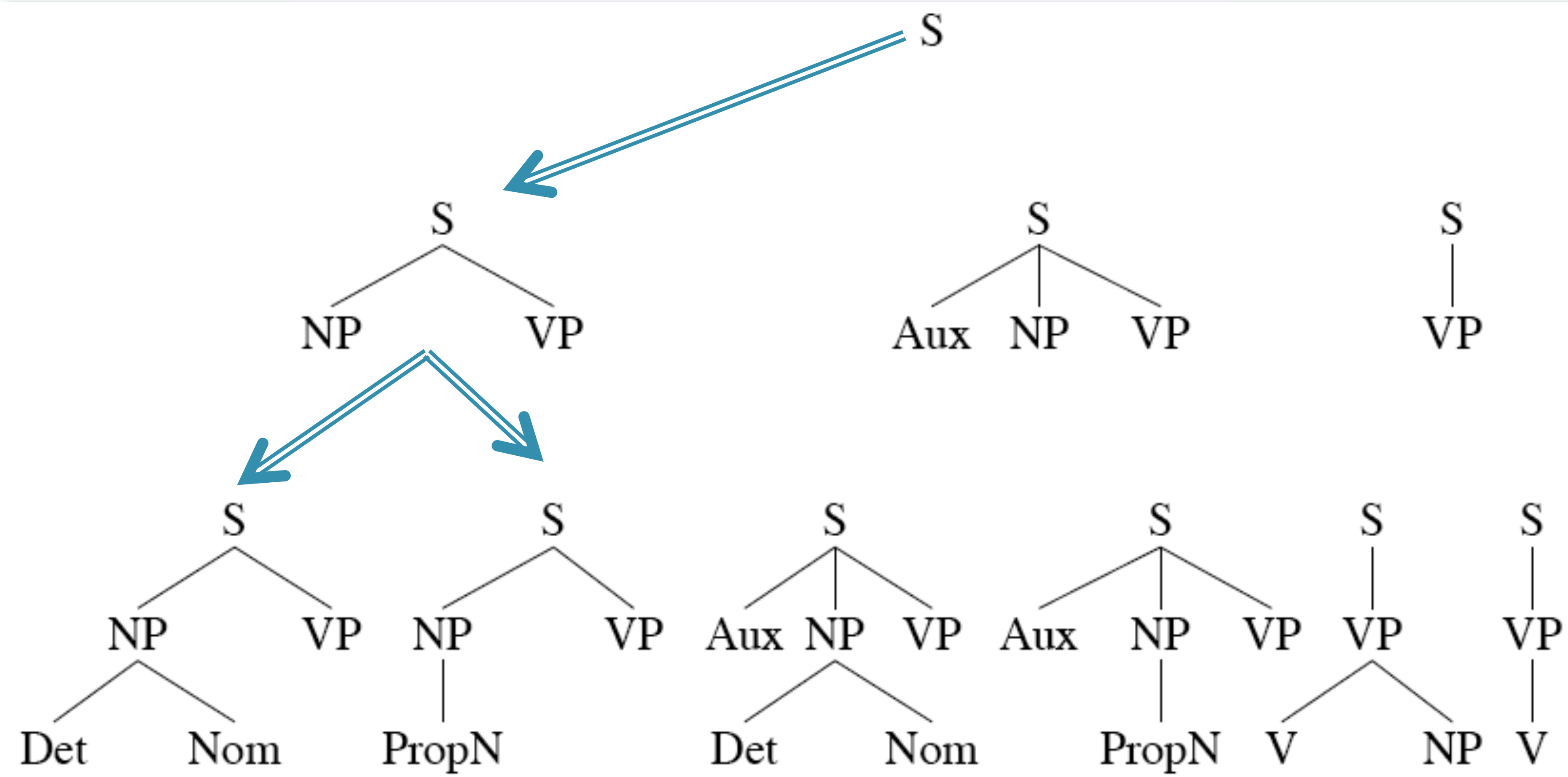
CKY Parsing

Ling571
Deep Processing Approaches to NLP
January 11, 2016

Roadmap

- Motivation:
  - Inefficiencies of parsing-as-search
- Strategy: Dynamic Programming
- Chomsky Normal Form
  - Weak and strong equivalence
- CKY parsing algorithm

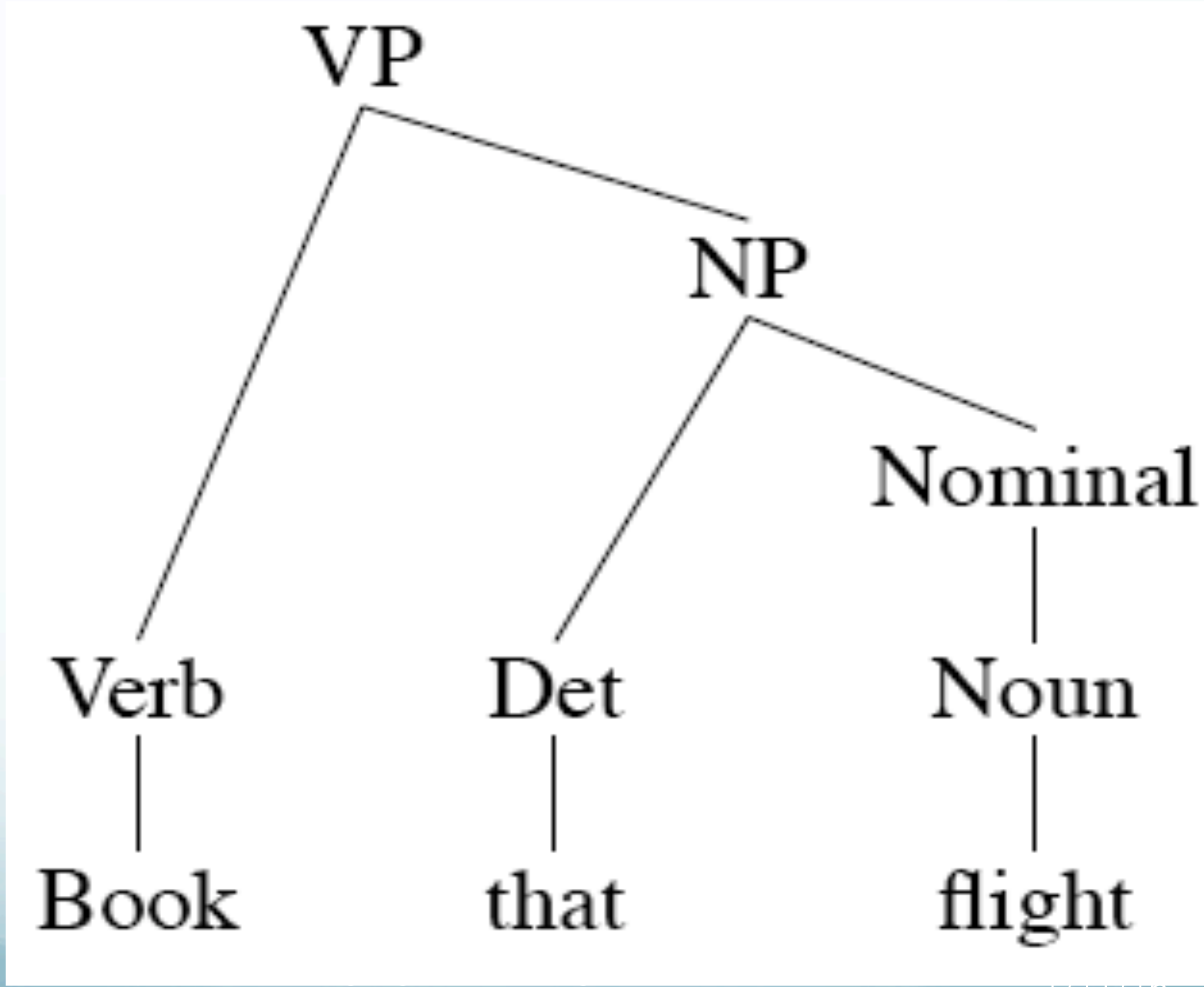
Top-down parsing (DFS)



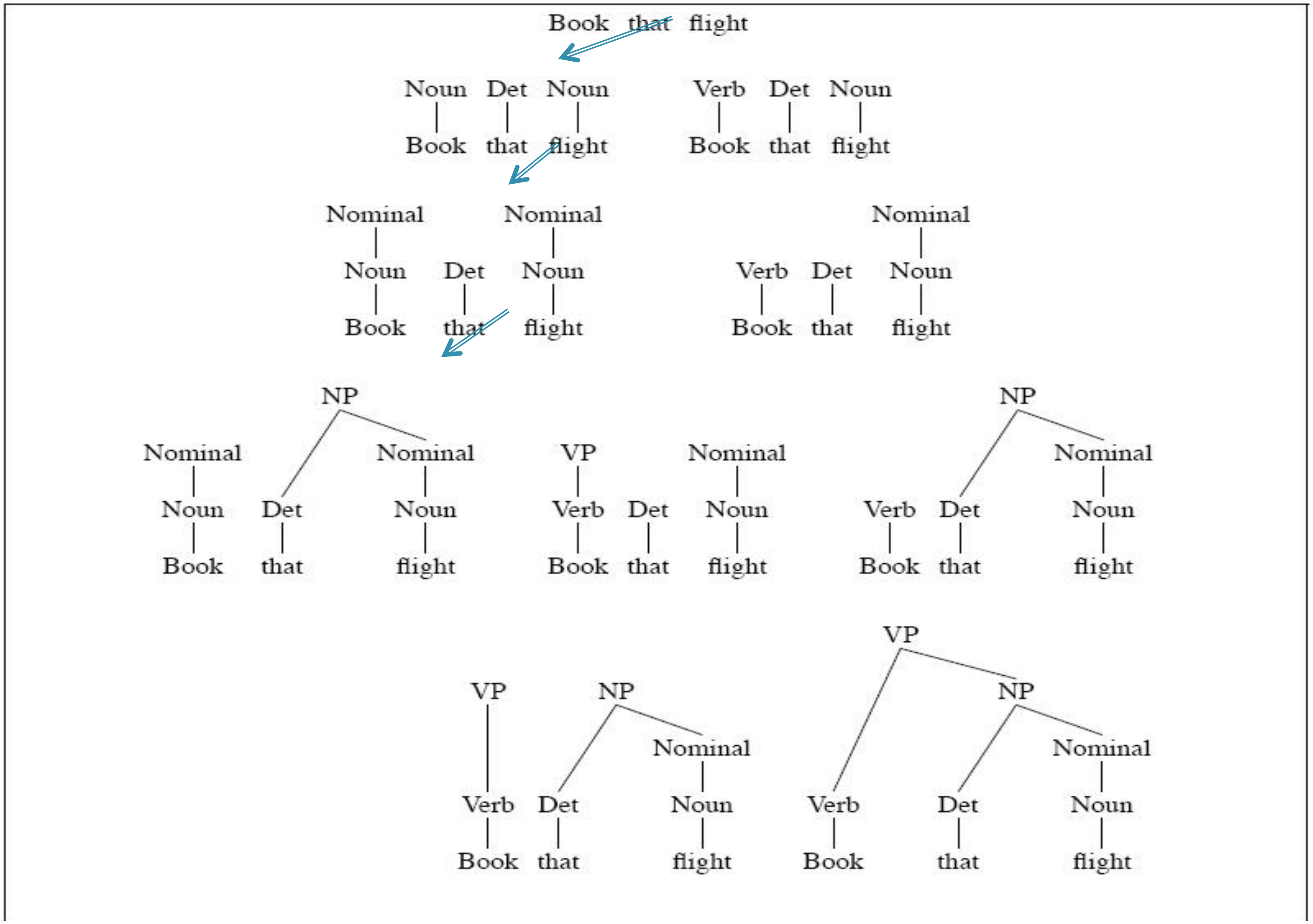
Bottom-Up Parsing

- Try to find all trees that span the input
  - Start with input string
    - Book that flight.
  - Use all productions with current subtree(s) on RHS
    - E.g., N → Book; V → Book
  - Stop when spanned by S (or no more rules apply)

Bottom-Up Search



Bottom-Up Search



Pros and Cons of Bottom-Up Search

- Pros:
  - Will not explore trees that don't match input
  - Recursive rules less problematic
  - Useful for incremental/ fragment parsing

- Cons:
  - Explore subtrees that will not fit full sentences

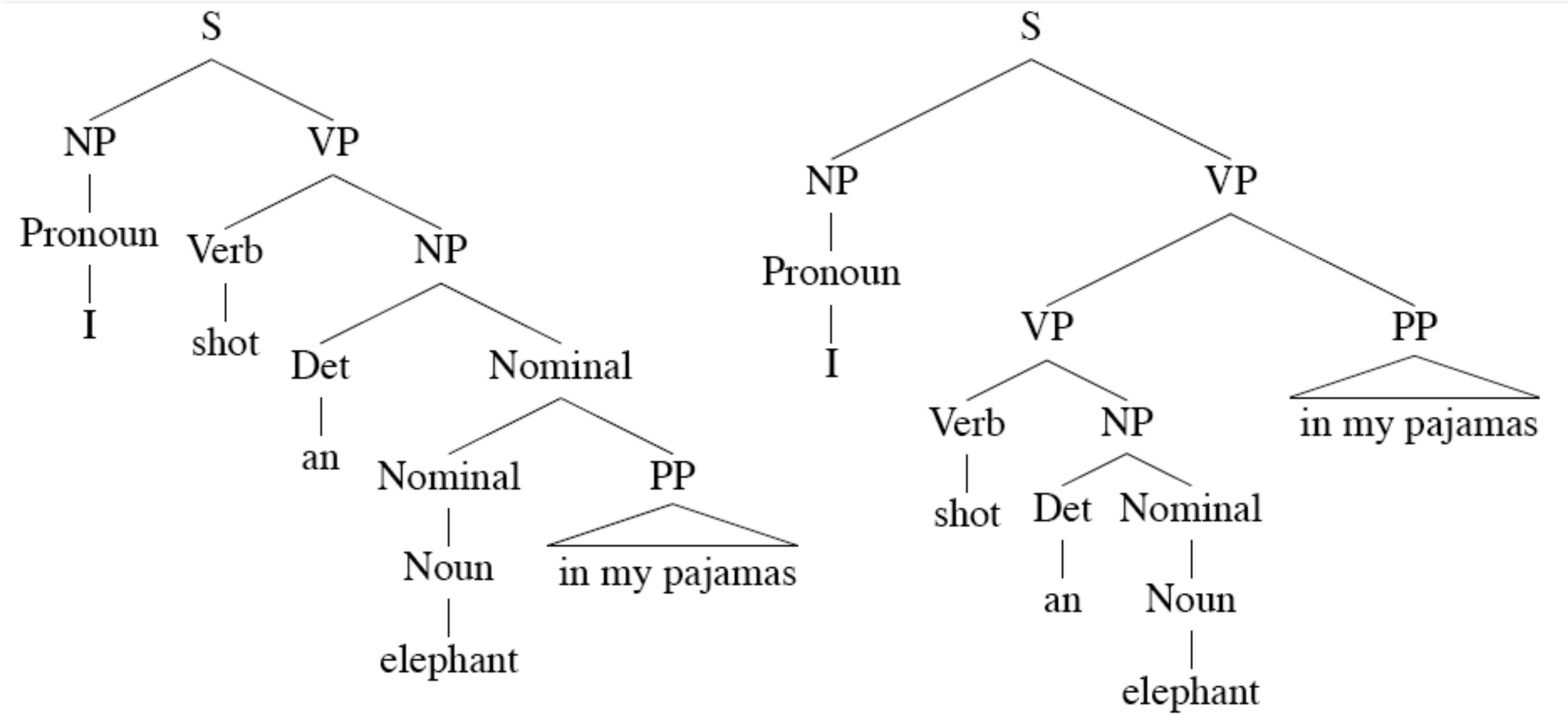
Parsing Challenges

- Ambiguity
- Repeated substructure
- Recursion

Parsing Ambiguity

- Many sources of parse ambiguity
  - Lexical ambiguity
    - Book/N; Book/V
  - Structural ambiguity: Main types:
    - Attachment ambiguity
      - Constituent can attach in multiple places
        - *I shot an elephant in my pyjamas*.
    - Coordination ambiguity
      - Different constituents can be conjoined
        - *Old men and women*

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

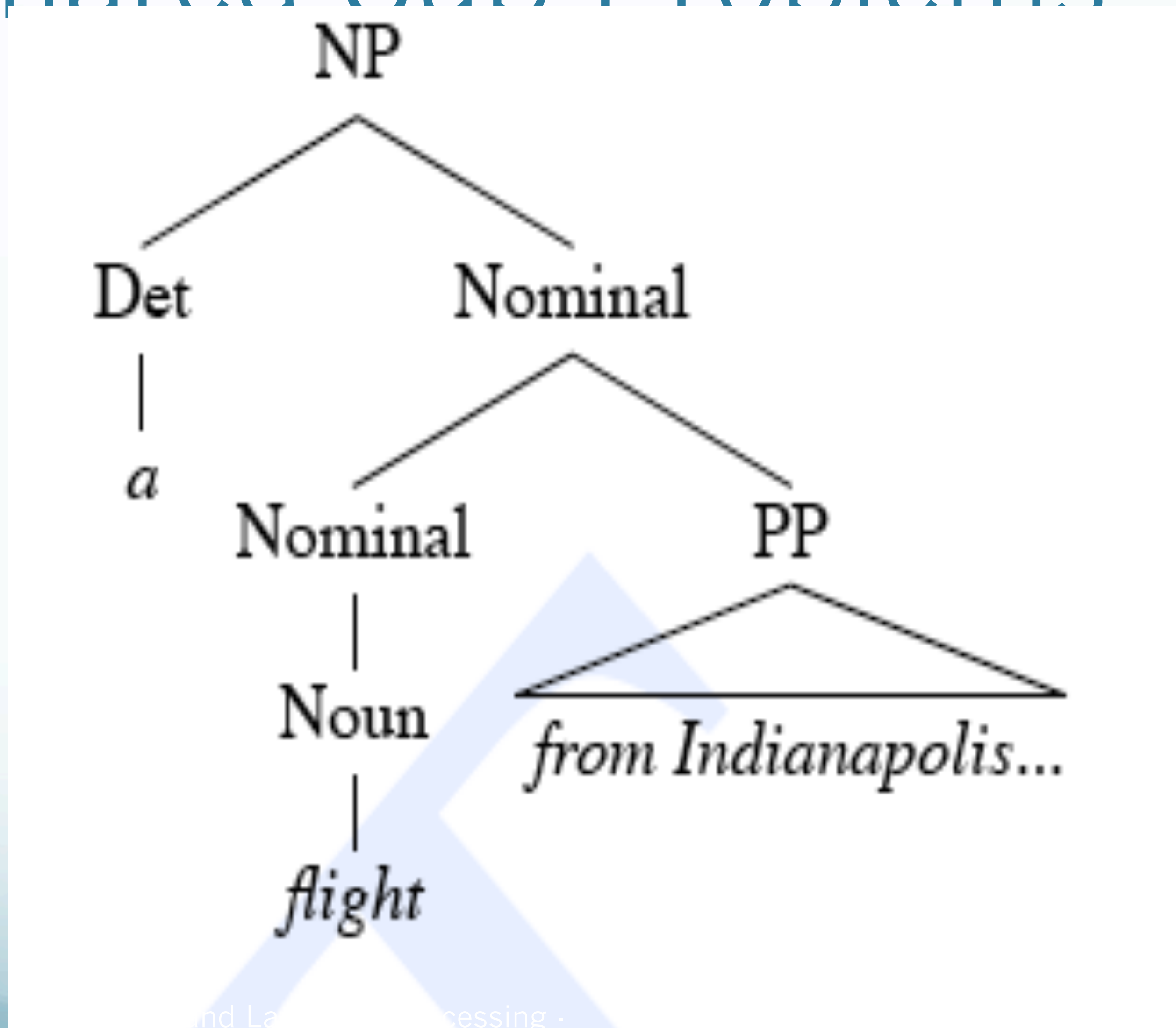
Disambiguation

- Global ambiguity:
  - Multiple complete alternative parses
  - Need strategy to select correct one
    - Approaches exploit other information
      - Statistical
        - Some prepositional structs more likely to attach high/low
        - Some phrases more likely, e.g., (old (men and women))
      - Semantic
      - Pragmatic
        - E.g., elephants and pyjamas
  - Alternatively, keep all
- Local ambiguity:
  - Ambiguity in subtree, resolved globally

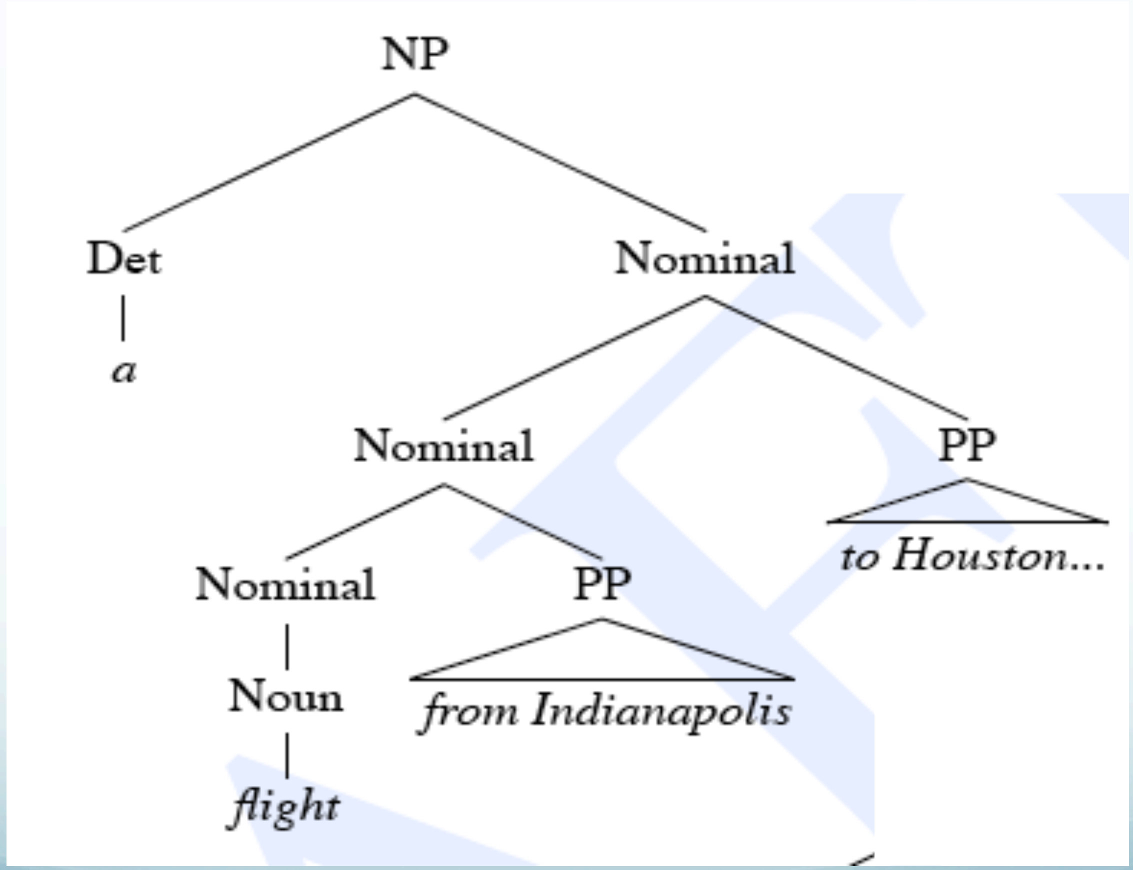
Repeated Work

- Top-down and bottom-up parsing both lead to repeated substructures
  - Globally bad parses can construct good subtrees
    - But overall parse will fail
    - Require reconstruction on other branch
  - No static backtracking strategy can avoid
- Efficient parsing techniques require storage of shared substructure
  - Typically with dynamic programming
- Example: *a flight from Indianapolis to Houston on TWA*

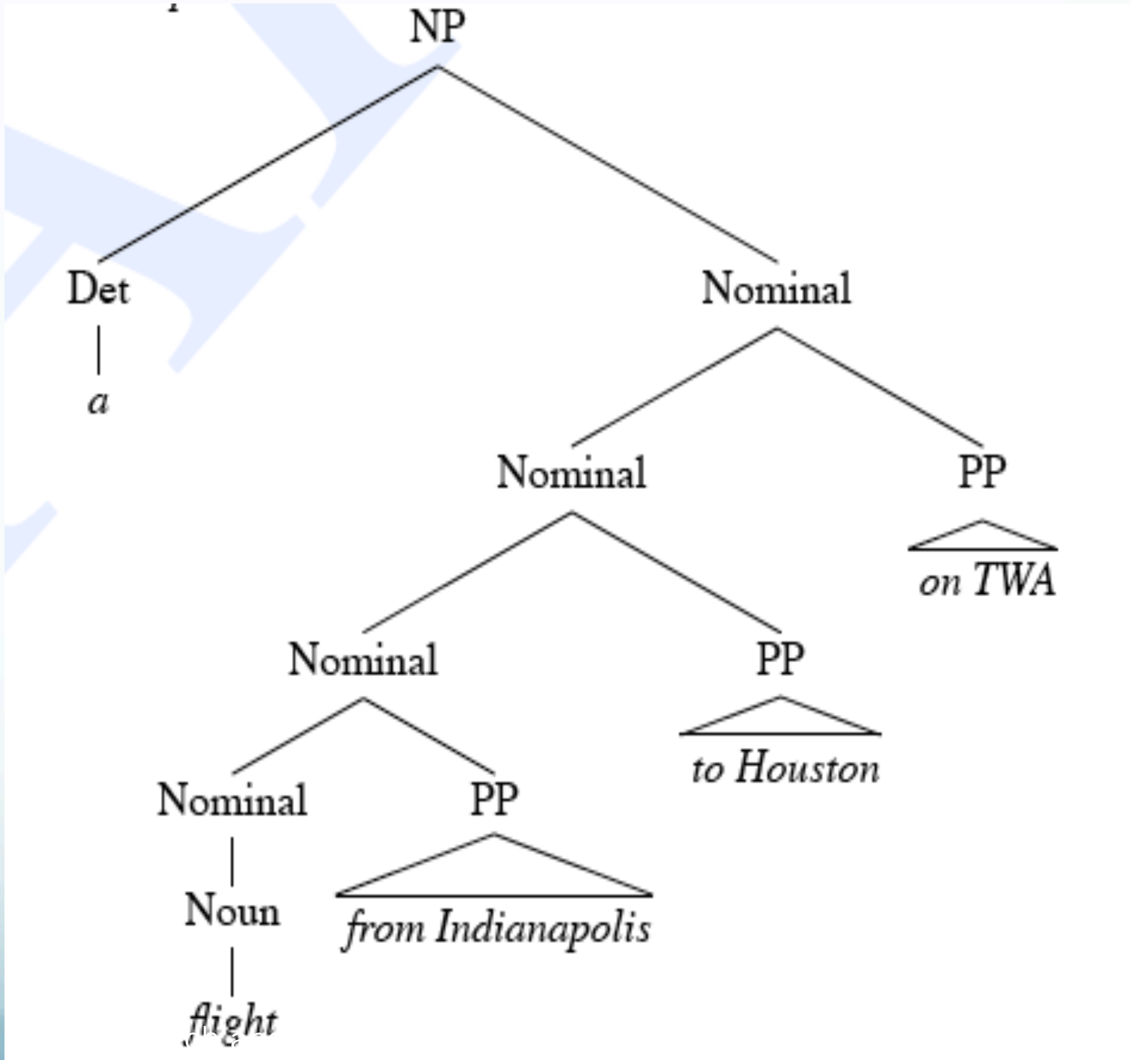
Shared Sub-Problems



Shared Sub-Problems



Shared Sub-Problems



Recursion

- Many grammars have recursive rules
  - E.g., S → S Conj S
- In search approaches, recursion is problematic
  - Can yield infinite searches
    - Esp., top-down

Dynamic Programming

- Challenge: Repeated substructure → Repeated work
- Insight:
  - Global parse composed of parse substructures
  - Can record parses of substructures
- Dynamic programming avoids repeated work by tabulating solutions to subproblems
  - Here, stores subtrees

Parsing w/Dynamic Programming

- Avoids repeated work
- Allows implementation of (relatively) efficient parsing algorithms
  - Polynomial time in input length
    - Typically cubic (n^3) or less
- Several different implementations
  - Cocke-Kasami-Younger (CKY) algorithm
  - Earley algorithm
  - Chart parsing

Chomsky Normal Form (CNF)

- CKY parsing requires grammars in CNF
- Chomsky Normal Form
  - All productions of the form:
    - A → B C, or
    - A → a
- However, most of our grammars are not of this form
  - E.g., S → Wh-NP Aux NP VP
- Need a general conversion procedure
  - Any arbitrary grammar can be converted to CNF

Grammar Equivalence and Form

- Grammar equivalence
  - Weak: Accept the same language, May produce different analyses
  - Strong: Accept same language, Produce same structure

CNF Conversion

- Three main conditions:
  - Hybrid rules:
    - INF-VP → to VP
  - Unit productions:
    - A → B
  - Long productions:
    - A → B C D

CNF Conversion

- Hybrid rule conversion:
  - Replace all terminals with dummy non-terminals
  - E.g., INF-VP → to VP
    - INF-VP → TO VP; TO → to
- Unit productions:
  - Rewrite RHS with RHS of all derivable non-unit productions
    - If $A \overset{*}{\Rightarrow} B$ and B → w, then add A → w

CNF Conversion

- Long productions:
  - Introduce new non-terminals and spread over rules
  - S → Aux NP VP
    - S → X1 VP; X1 → Aux NP

- For all non-conforming rules,
  - Convert terminals to dummy non-terminals
  - Convert unit productions
  - Binarize all resulting rules

$\mathscr{L}_1$ **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

- Cocke-Kasami-Younger parsing algorithm:
  - (Relatively) efficient bottom-up parsing algorithm based on tabulating substring parses to avoid repeated work
  - Approach:
    - Use a CNF grammar
    - Build an (n+1) x (n+1) matrix to store subtrees
      - Upper triangular portion
    - Incrementally build parse spanning whole input string

Dynamic Programming in CKY

- Key idea:
  - For a parse spanning substring [i,j] , there exists some k such there are parses spanning [i,k] and [k,j]
    - We can construct parses for whole sentence by building up from these stored partial parses
- So,
  - To have a rule A → B C in [i,j],
    - We must have B in [i,k] and C in [k,j], for some $i<k<j$
      - CNF grammar forces this for all $j>i+1$

CKY

- Given an input string S of length n,
  - Build table (n+1) x (n+1)
  - Indexes correspond to inter-word positions
    - E.g., 0 Book 1 That 2 Flight 3
- Cells [i,j] contain sets of non-terminals of ALL constituents spanning i,j
  - [j-1,j] contains pre-terminals
  - If [0,n] contains Start, the input is recognized

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**

$table[i,j] \leftarrow table[i,j] \cup$

$\{A \mid A \rightarrow BC \in grammar,$

$B \in table[i,k],$

$C \in table[k,j]\}$

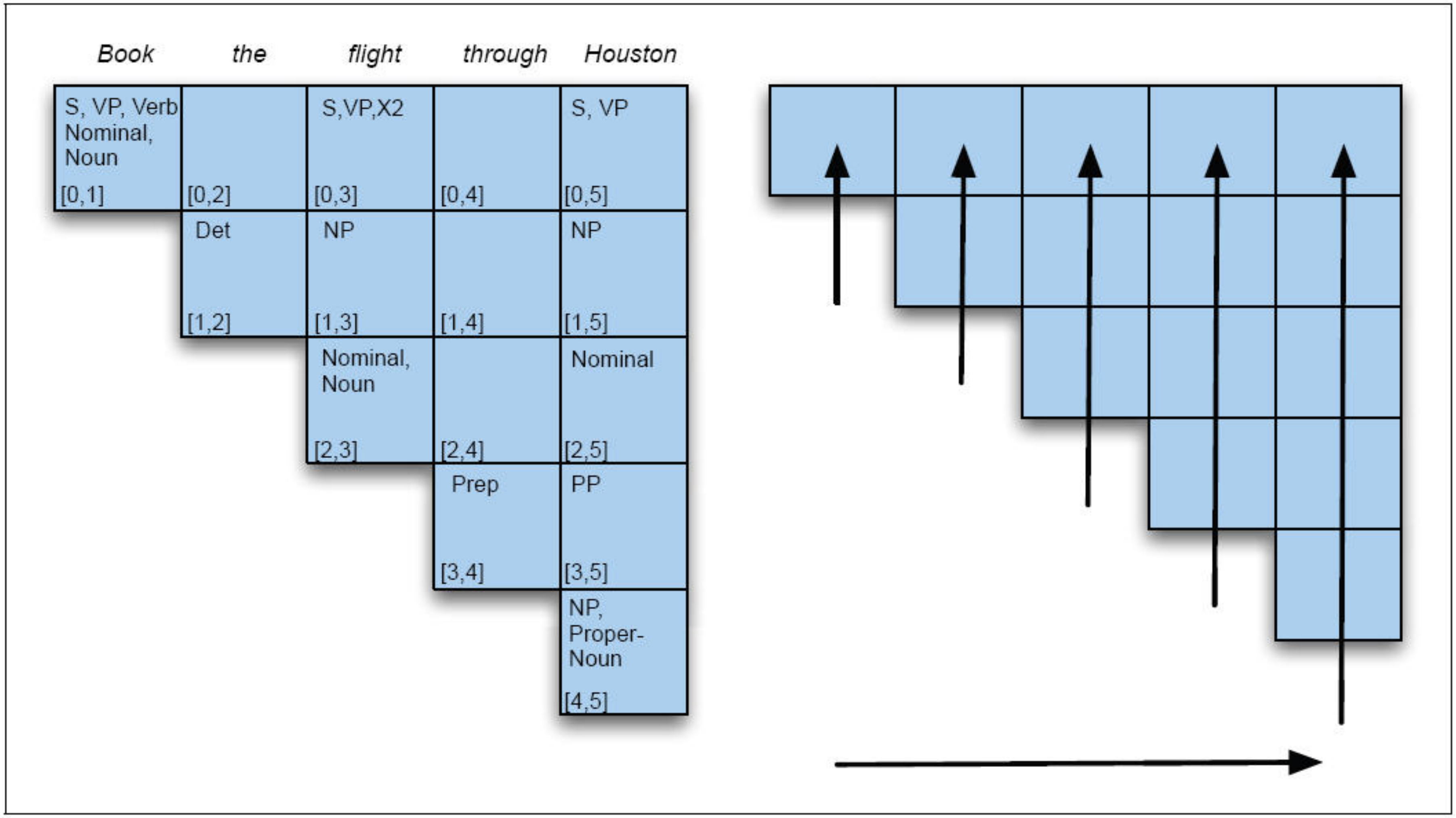
Is this a parser?

CKY Parsing

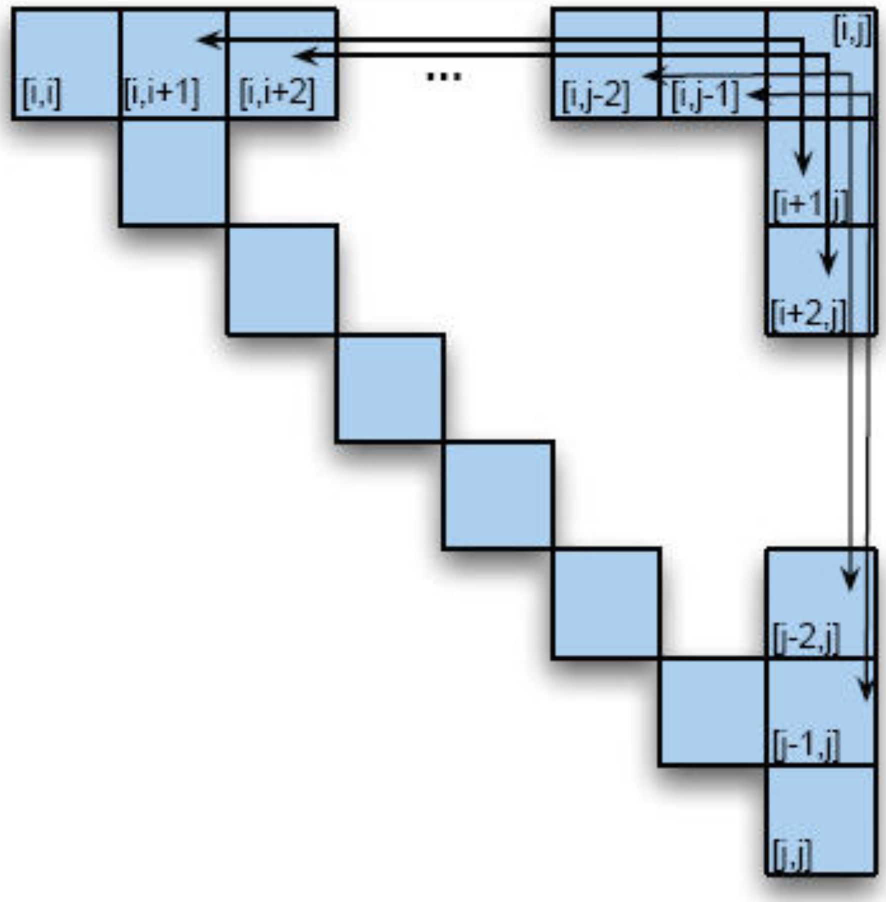
- Table fills:
  - Column-by-column
  - Left-to-right
  - Bottom-to-top
- Why?
  - Necessary info available (below and left)
  - Allows online sentence analysis
    - Works across input string as it arrives

CKY Table

- Book the flight through Houston



Filling CKY cell



0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]		
	Det [1,2]	NP [1,3]		
		NN, Nominal [2,3]		