CS-AD 220 – Spring 2016

Natural Language Processing

Session 17: 2-Apr-16

Prof. Nizar Habash

Some slides are adapted from Jurafsky and Martin's course slides on Speech and Language Processing

NYUAD Course CS-AD 220 – Spring 2016

Natural Language Processing

Assignment #3 : POS Tagging and Parsing

Assigned Mar 31, 2016 / Due Apr 17, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the development of a dependency parser and a part-of-speech (POS) tagger for English. The assignment accounts for 15% of the full grade. It consists of five exercises. **There is also a bonus exercise that can count for up to 5% of the full grade**. The additional exercise consists of a parsing competition on an unseen test set. Participation earns 2%. The first, second and third ranked systems earn additional 3%, 2% and 1%, respectively.

Assignment #3 posted on NYU Classes

START EARLY!

DEADLINE PUSHED FORWARD TO APR 17

Moving Legislative Day Class

- Spring Break is March 18 – 25, 2016
- Sat March 26, 2016 is a Legislative *Thursday*
- Move to

Today, Now, Here

Treebanks

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

Penn Treebank

- Penn TreeBank is a widely used treebank.
  - Most well known is the Wall Street Journal section of the Penn TreeBank.
    - 1 M words from the 1987-1989 Wall Street Journal.

```
( (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) ))
   (. .) ))
```

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.
- Not complete, but if you have decent size corpus, you'll 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.

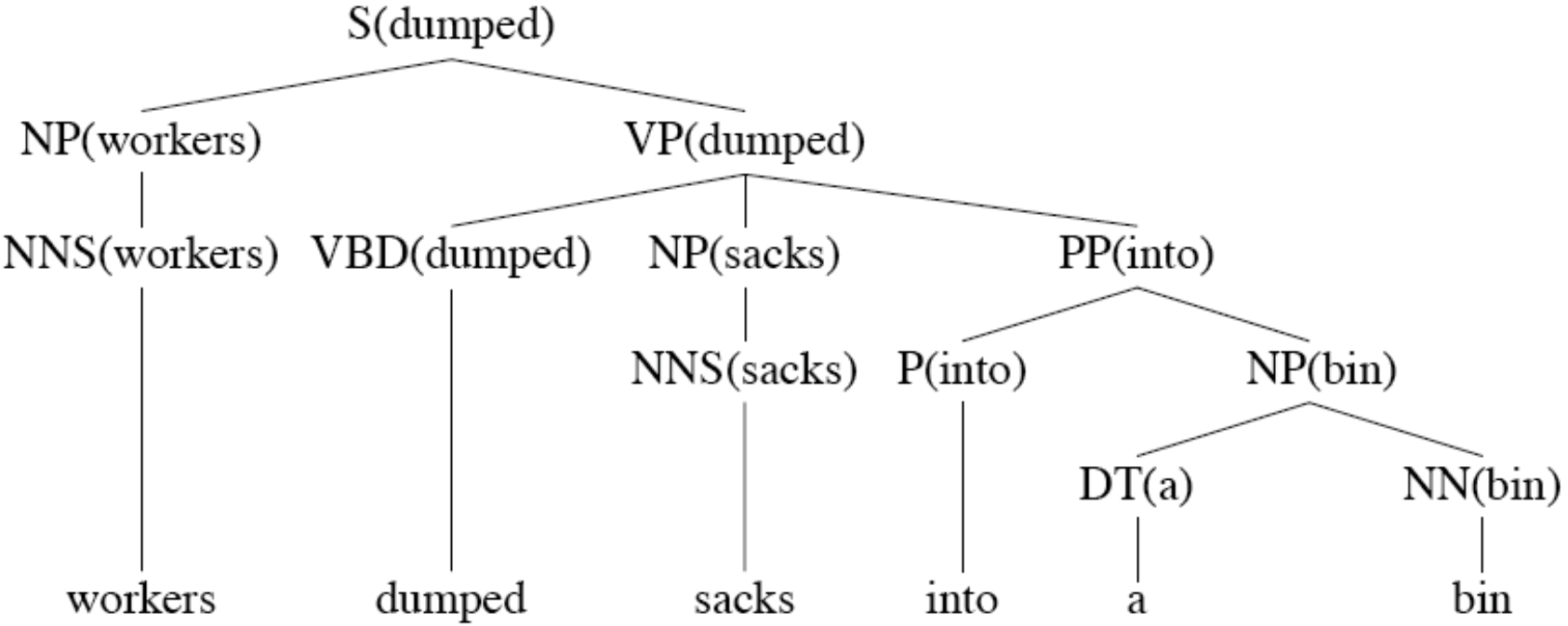
- For example, the Penn Treebank has 4500 different rules for VPs. Among them...

$$\begin{aligned}
\text{VP} &\rightarrow \text{VBD PP} \\
\text{VP} &\rightarrow \text{VBD PP PP} \\
\text{VP} &\rightarrow \text{VBD PP PP PP} \\
\text{VP} &\rightarrow \text{VBD PP PP PP PP}
\end{aligned}$$

Heads in Trees

- Finding heads in treebank trees is a task that arises frequently in many applications.
  - 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.

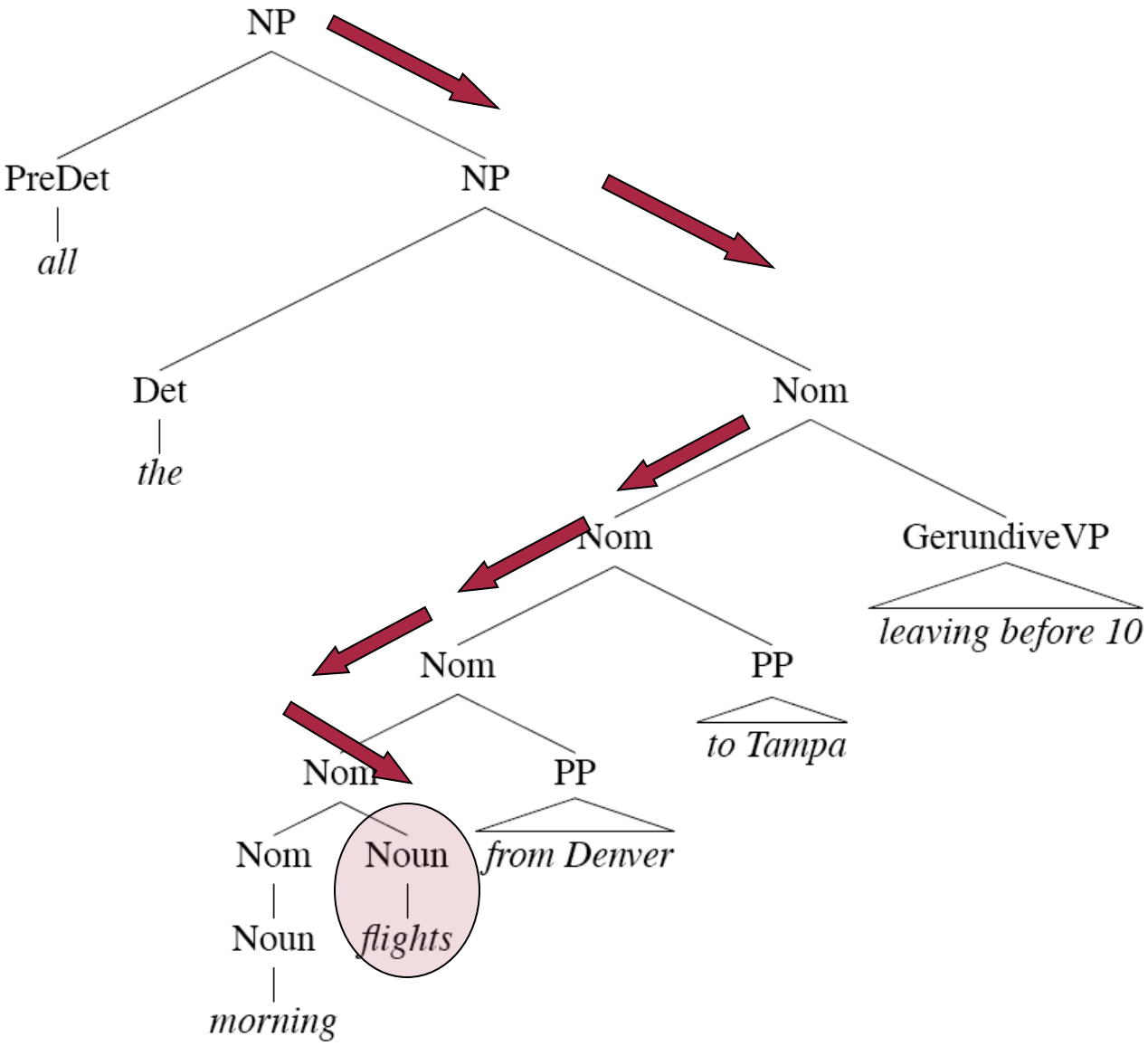
Lexically Decorated Tree



Head Finding

- 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.
  - aka head-percolation rules

Noun Phrases



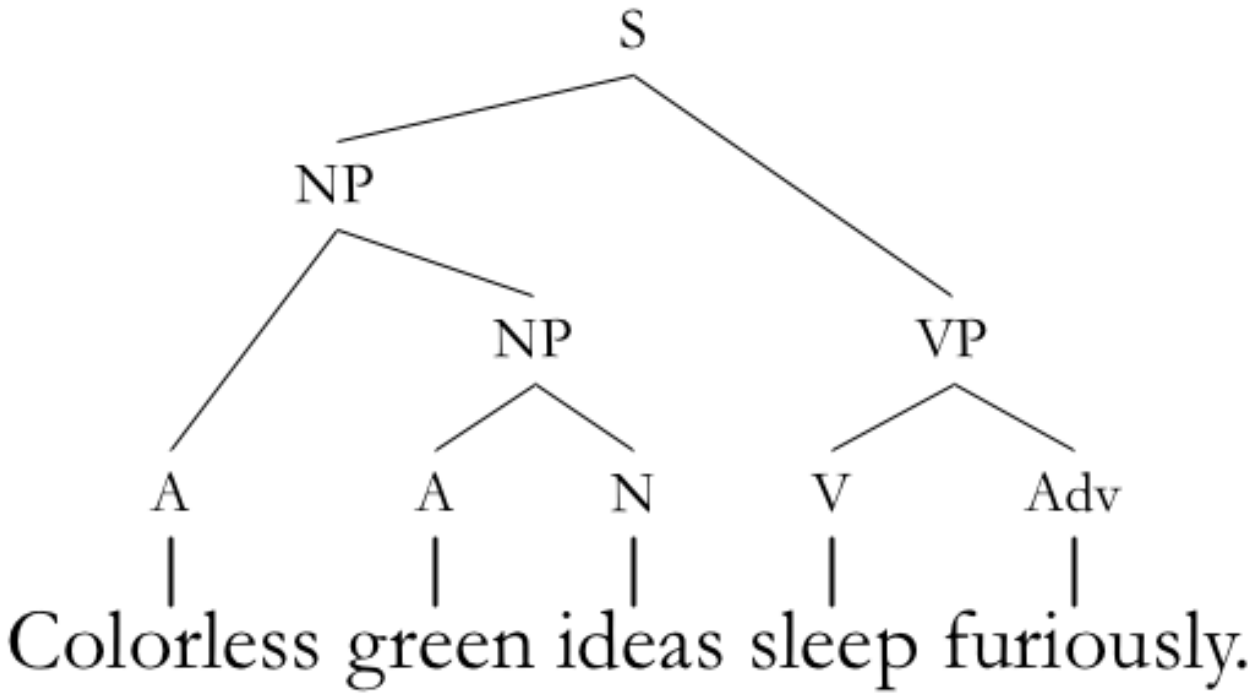
Treebank Uses

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

Types of Syntactic Analyses

- Phrase Structure Parsing
  - aka constituency parsing
- Dependency Parsing
- Chunking
  - Base-phrase chunking

Syntactic Analysis



- Famous example of how a sentence can be grammatical but its semantics nonsensical.
  - From “Syntactic Structures” (Chomsky, 1957)

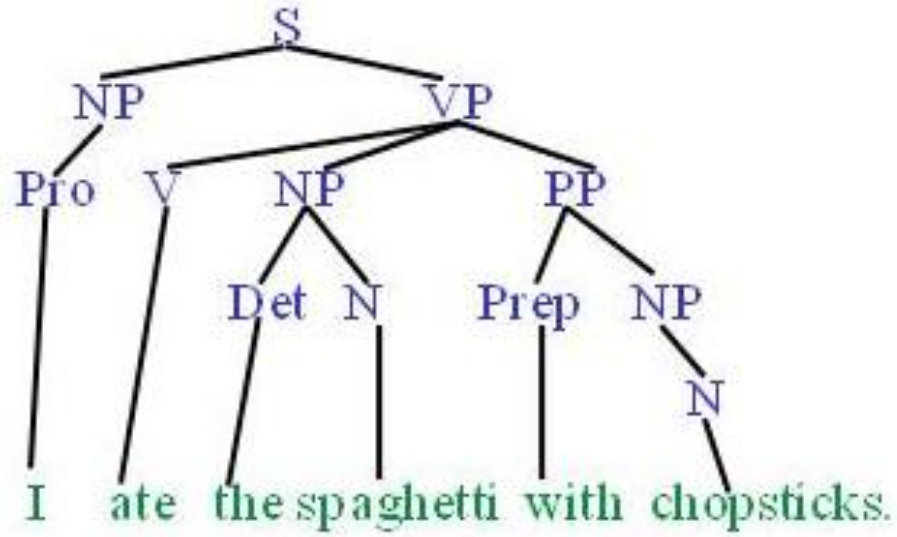
Syntactic Parsing

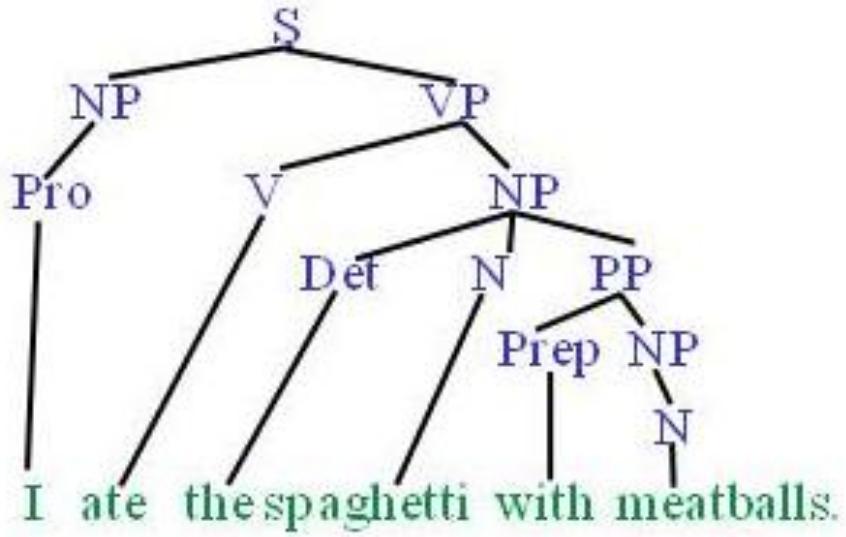
- Produce the correct syntactic parse tree for a sentence.

I ate the spaghetti with chopsticks. I ate the spaghetti with meatballs.

Syntactic Parsing

- Produce the correct syntactic parse tree for a sentence.





Simple CFG for ATIS* English

Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

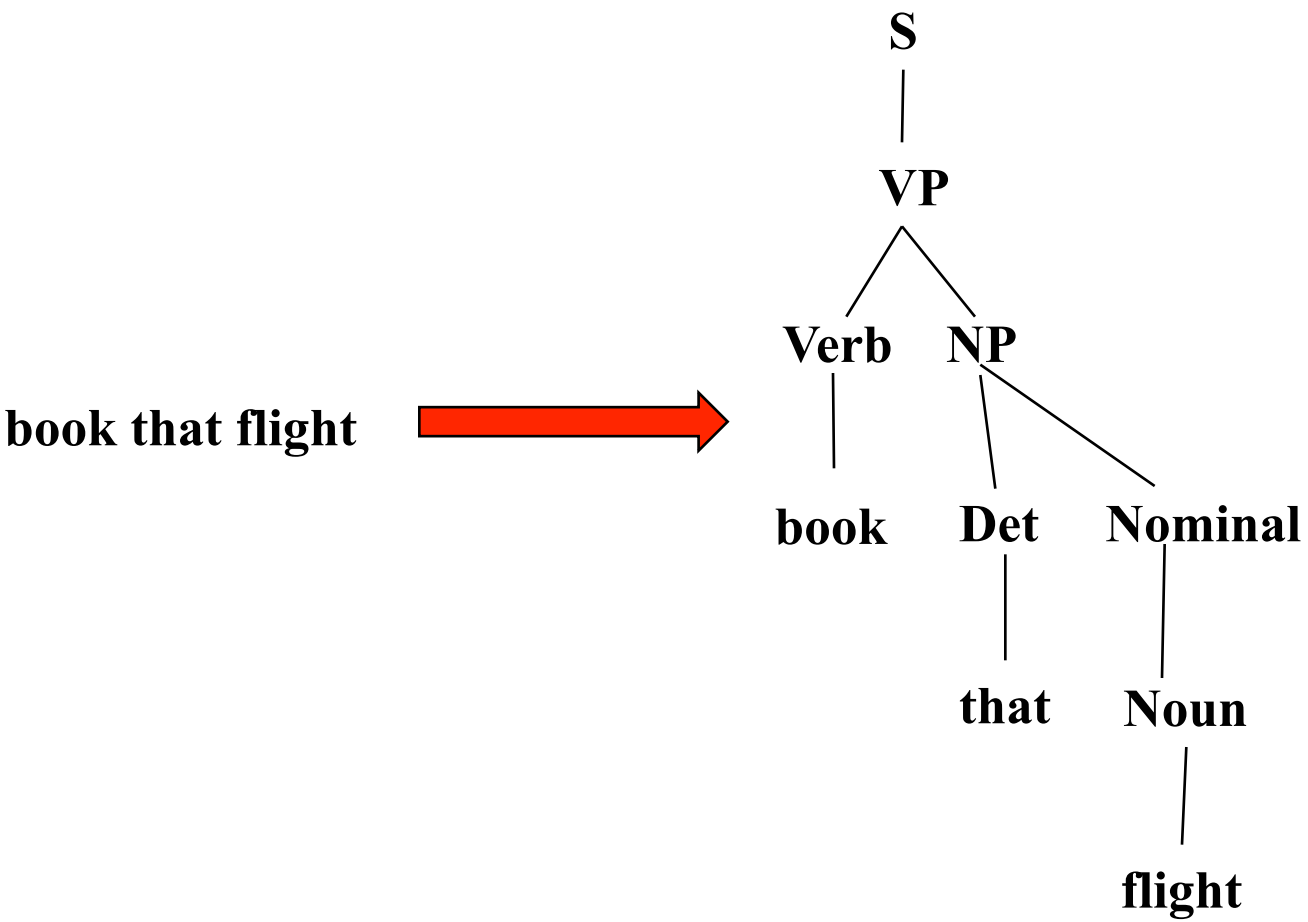
Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

*Air Travel Information Systems Project

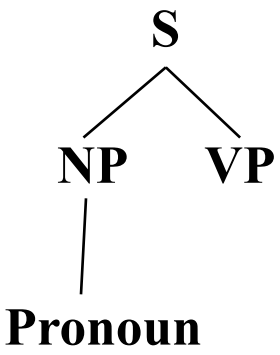
Parsing

- Given a string of terminals and a CFG, determine if the string can be generated by the CFG.
  - Also return a parse tree for the string
  - Also return all possible parse trees for the string
- Must search space of derivations for one that derives the given string.
  - **Top-Down Parsing**: Start searching space of derivations for the start symbol.
  - **Bottom-up Parsing**: Start search space of reverse deivations from the terminal symbols in the string.

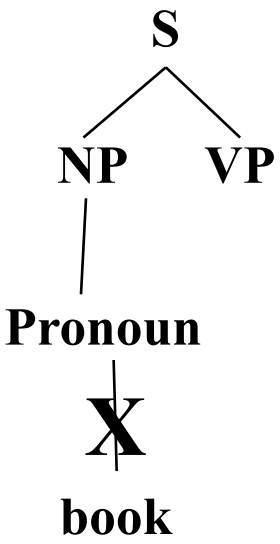
Parsing Example



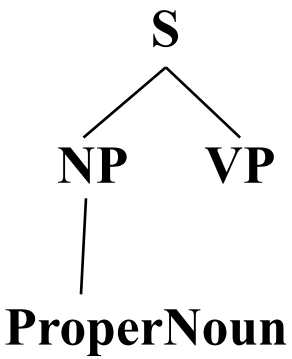
Top Down Parsing



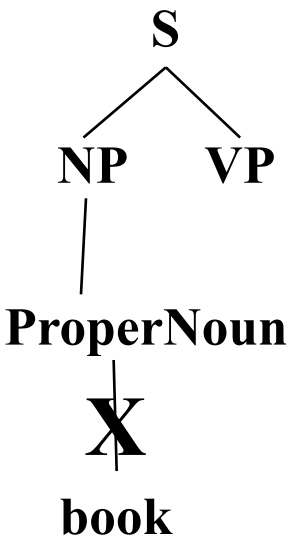
Top Down Parsing



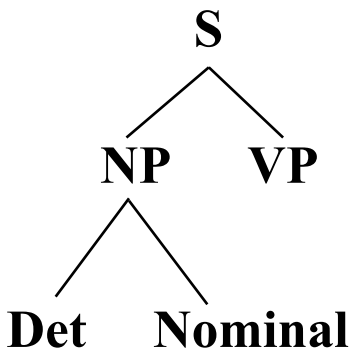
Top Down Parsing



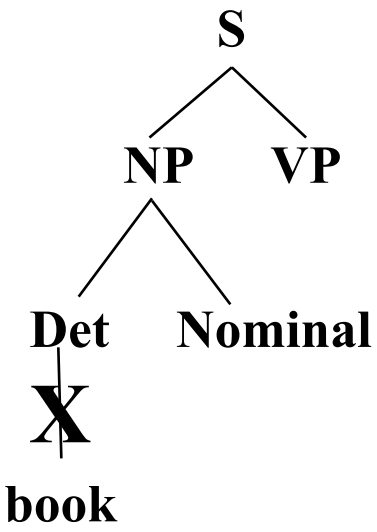
Top Down Parsing



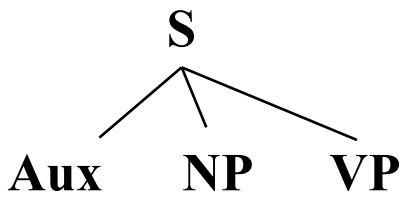
Top Down Parsing



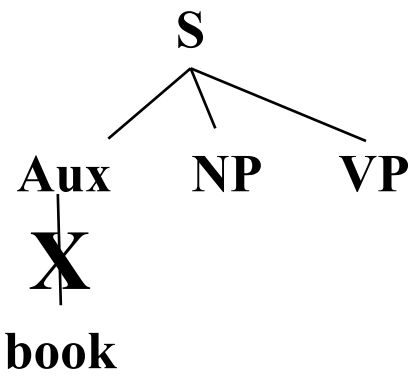
Top Down Parsing



Top Down Parsing

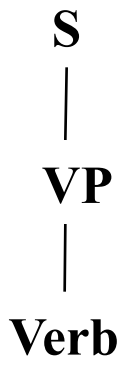


Top Down Parsing



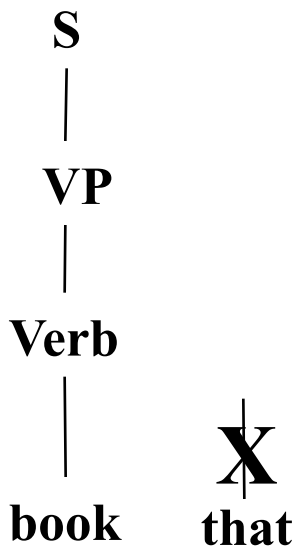
Top Down Parsing

Top Down Parsing

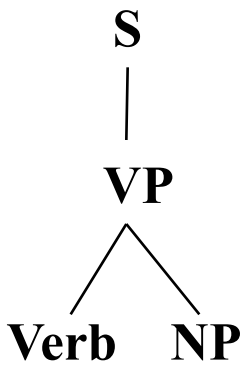


Top Down Parsing

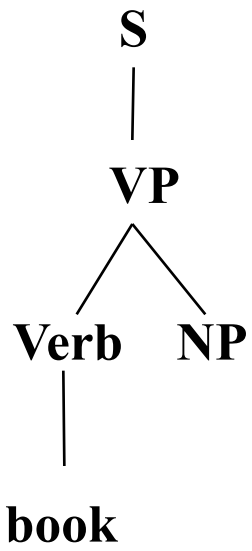
Top Down Parsing



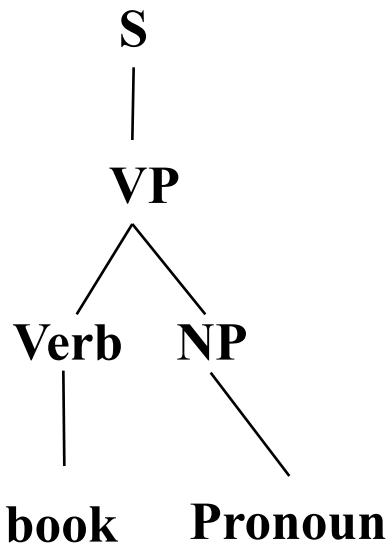
Top Down Parsing



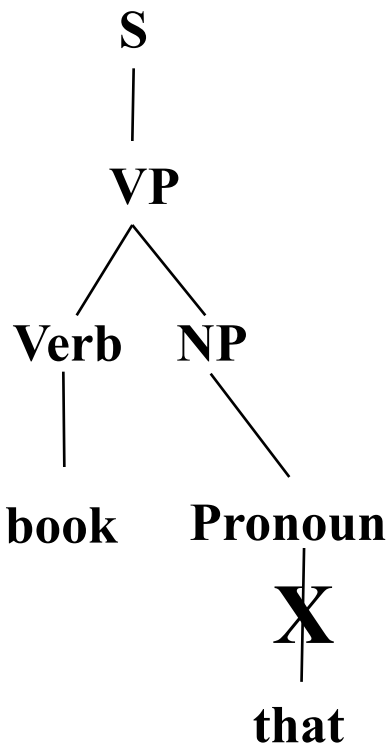
Top Down Parsing



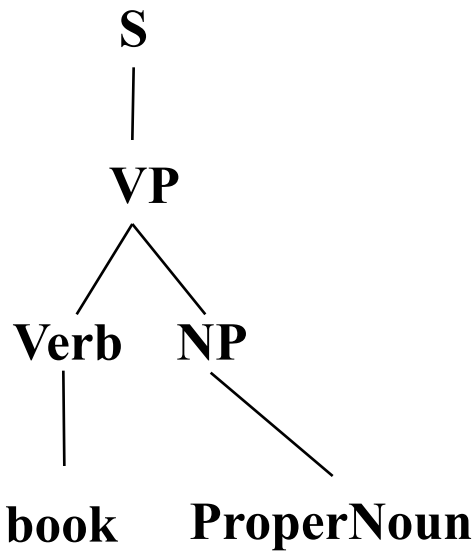
Top Down Parsing



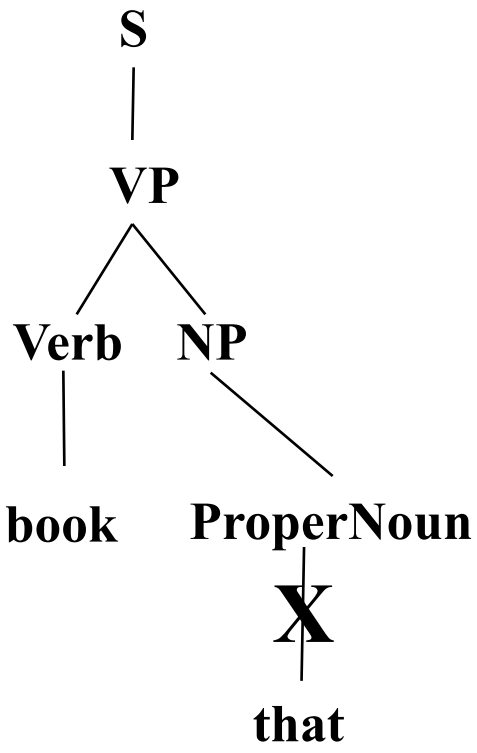
Top Down Parsing



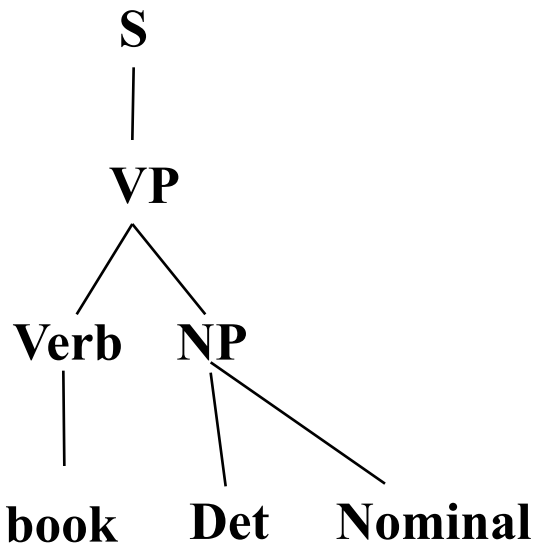
Top Down Parsing



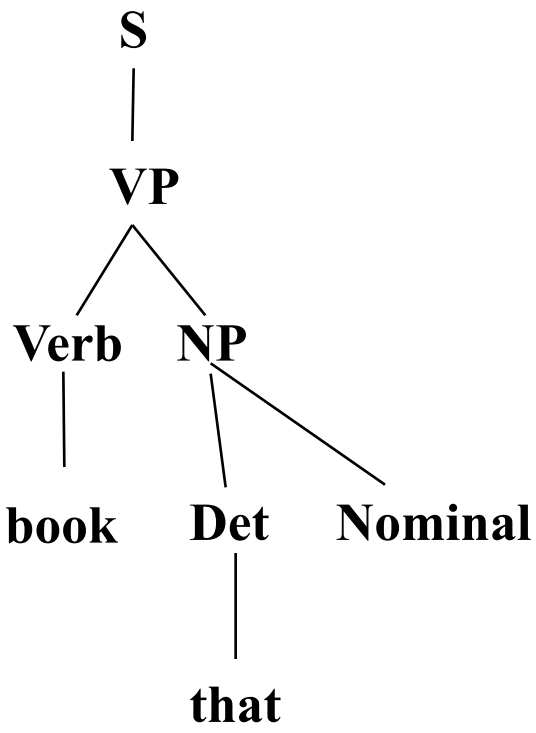
Top Down Parsing



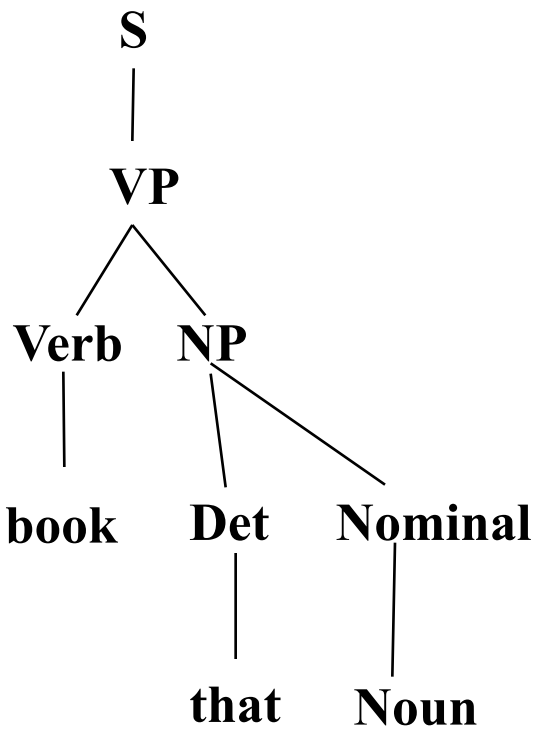
Top Down Parsing



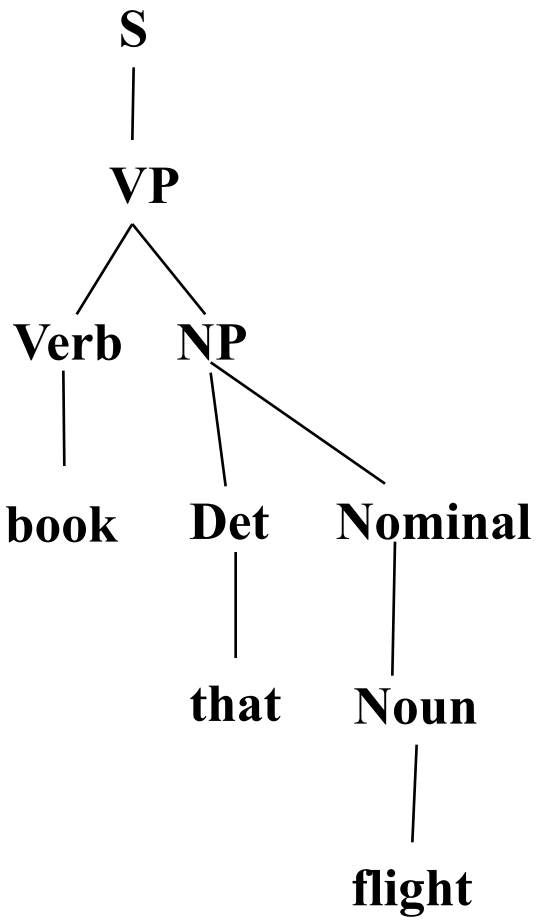
Top Down Parsing



Top Down Parsing



Top Down Parsing

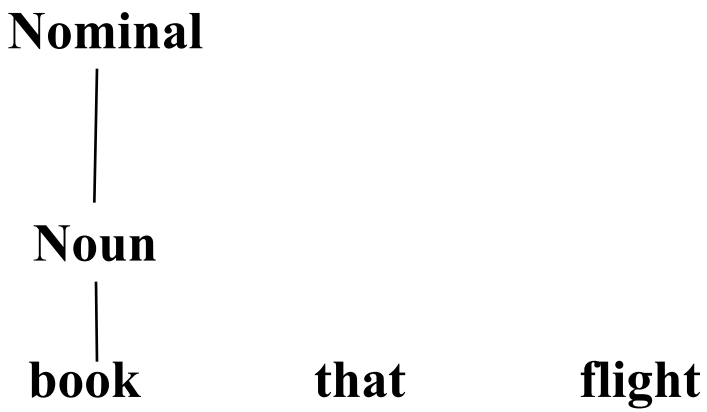


Bottom Up Parsing

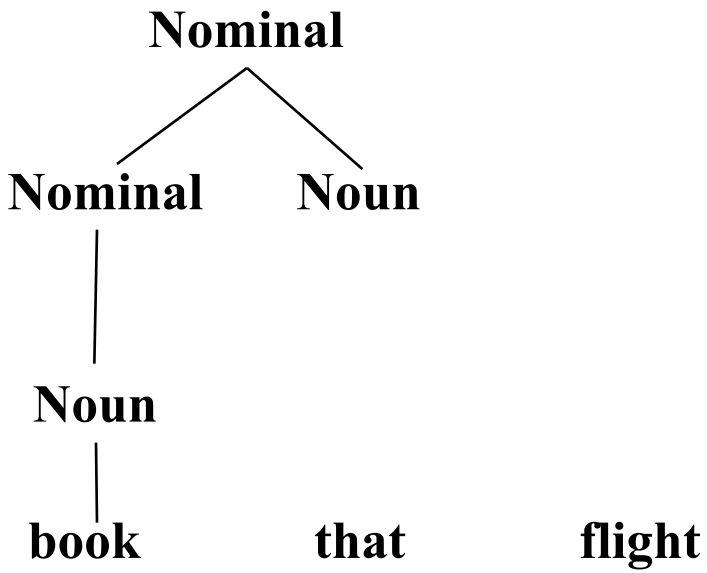
book **that** **flight**

Bottom Up Parsing

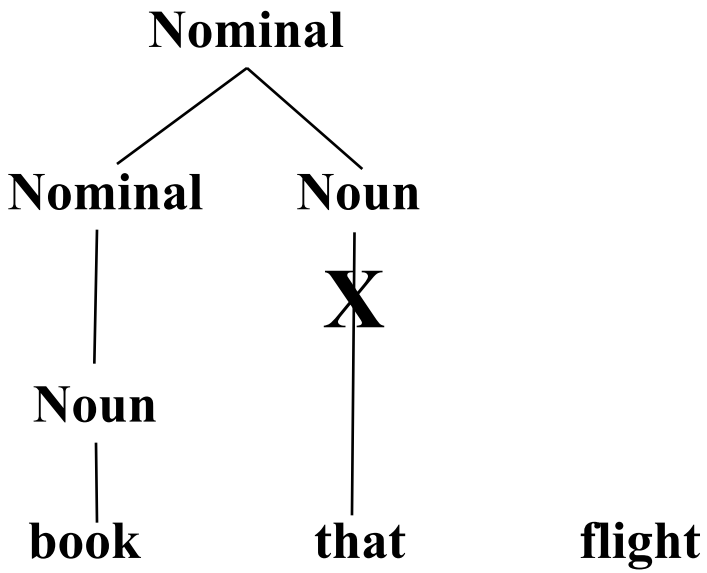
Bottom Up Parsing



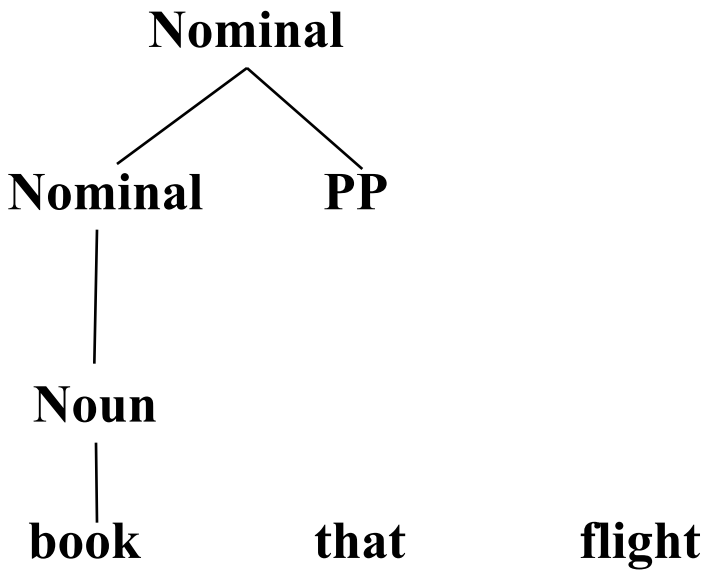
Bottom Up Parsing



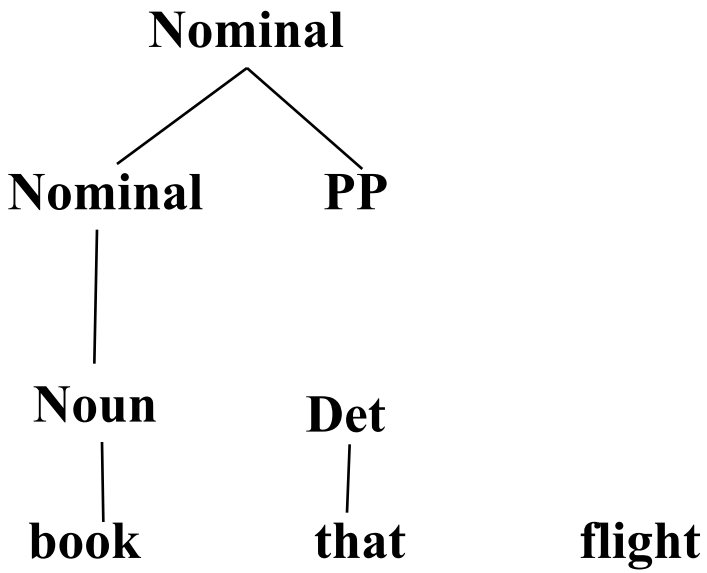
Bottom Up Parsing



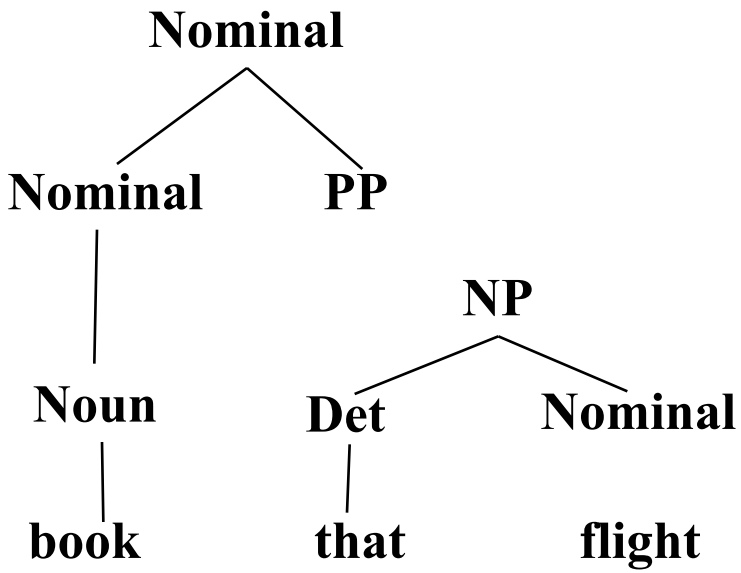
Bottom Up Parsing



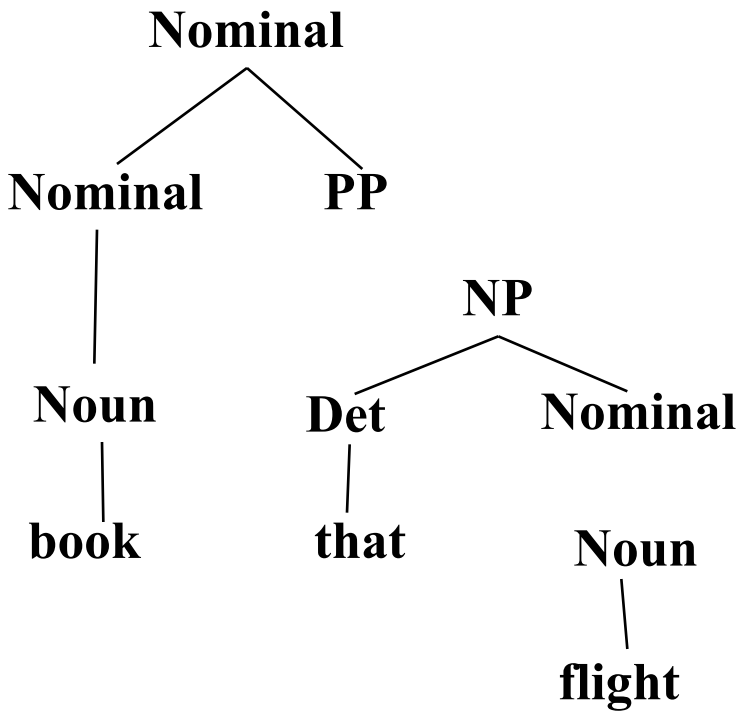
Bottom Up Parsing



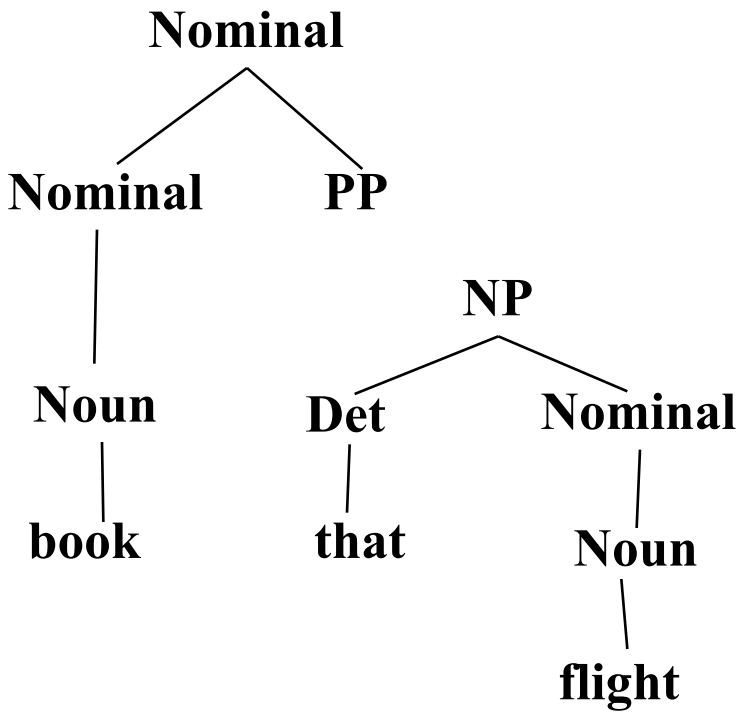
Bottom Up Parsing



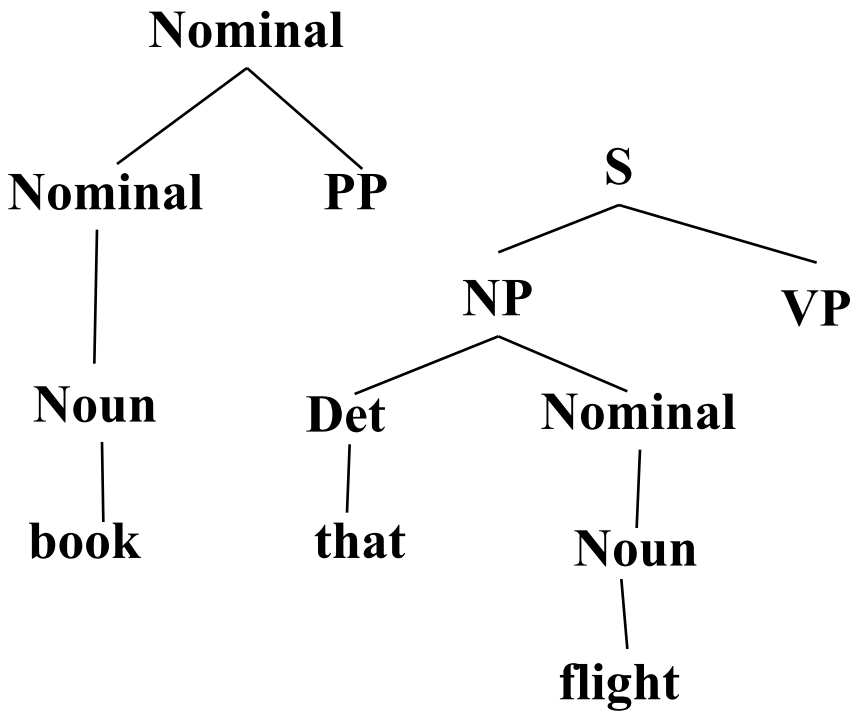
Bottom Up Parsing



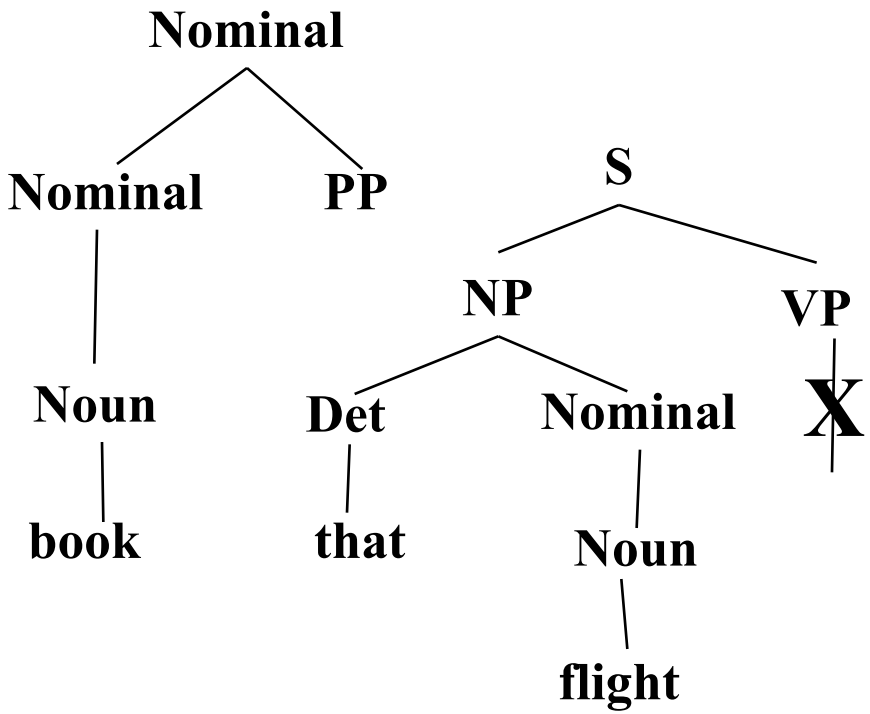
Bottom Up Parsing



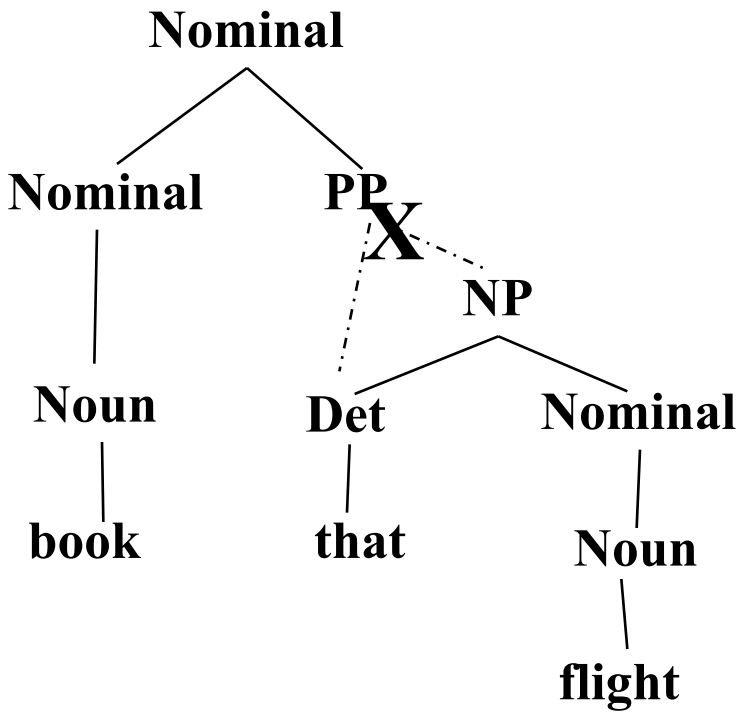
Bottom Up Parsing



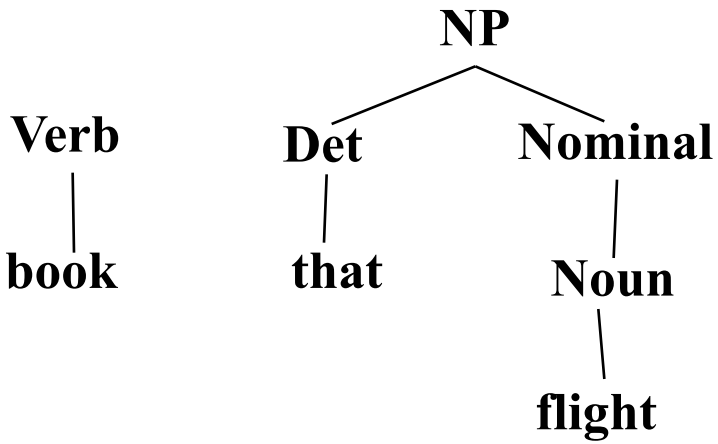
Bottom Up Parsing



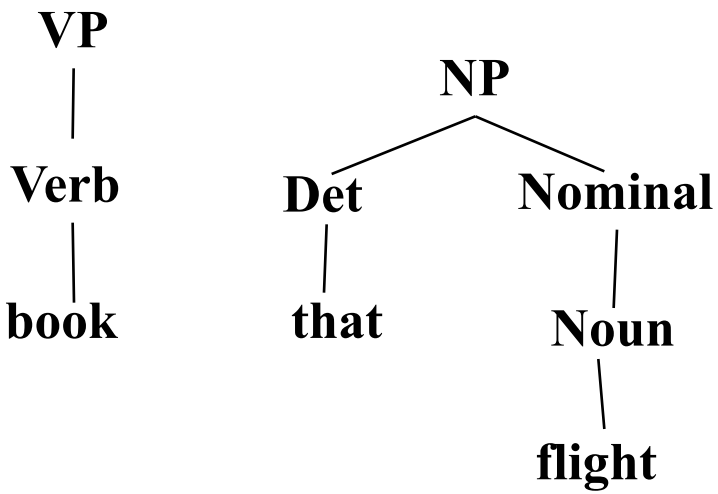
Bottom Up Parsing



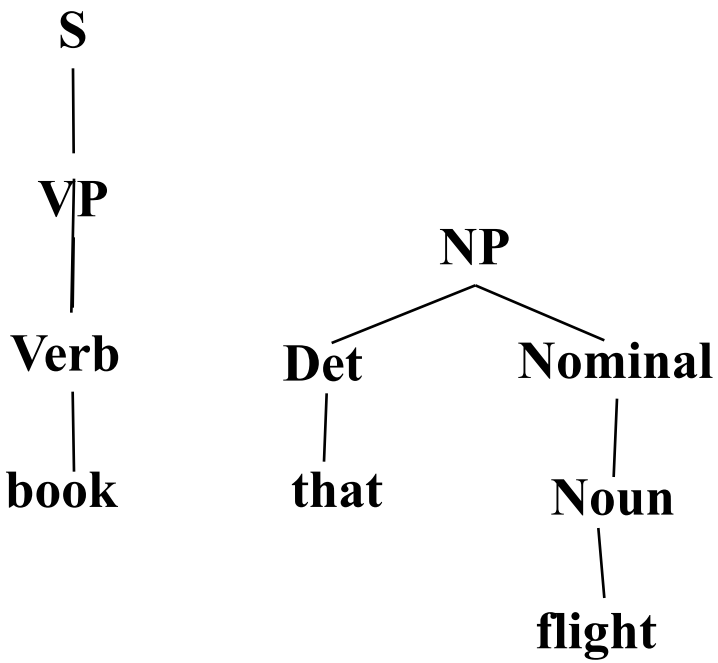
Bottom Up Parsing



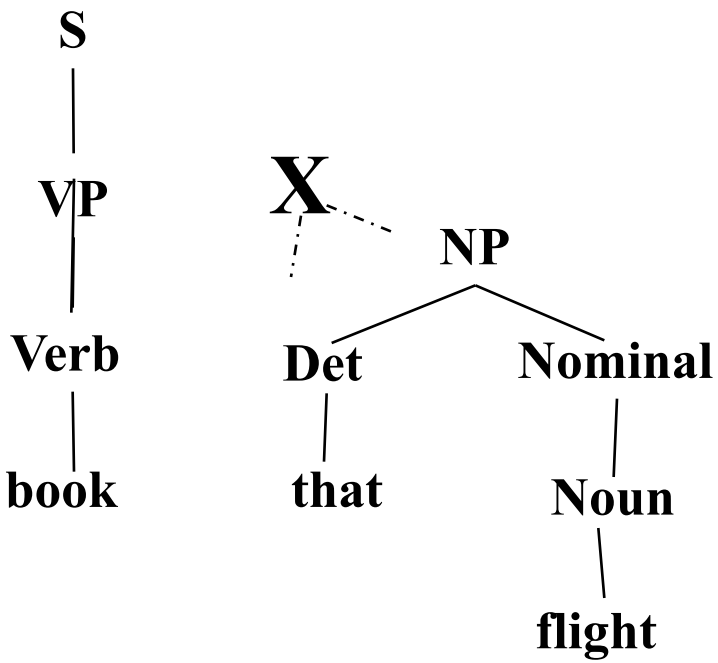
Bottom Up Parsing



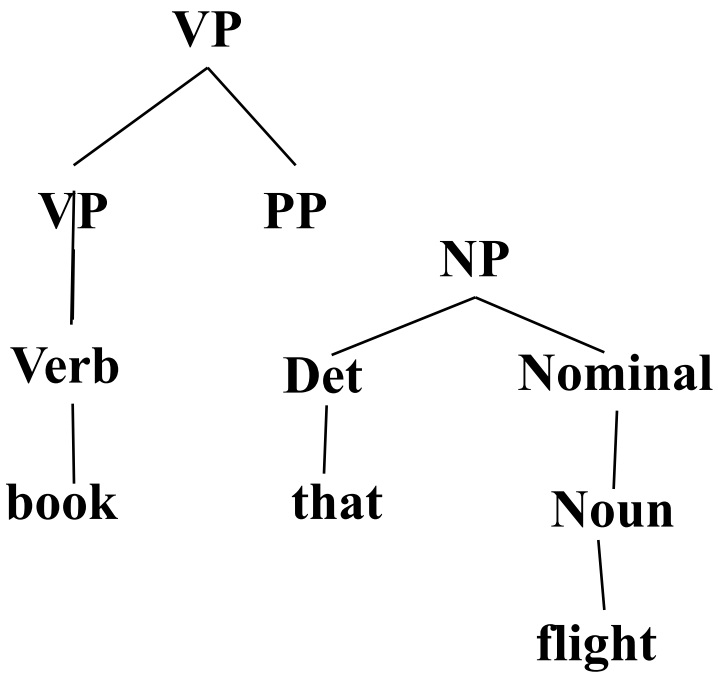
Bottom Up Parsing



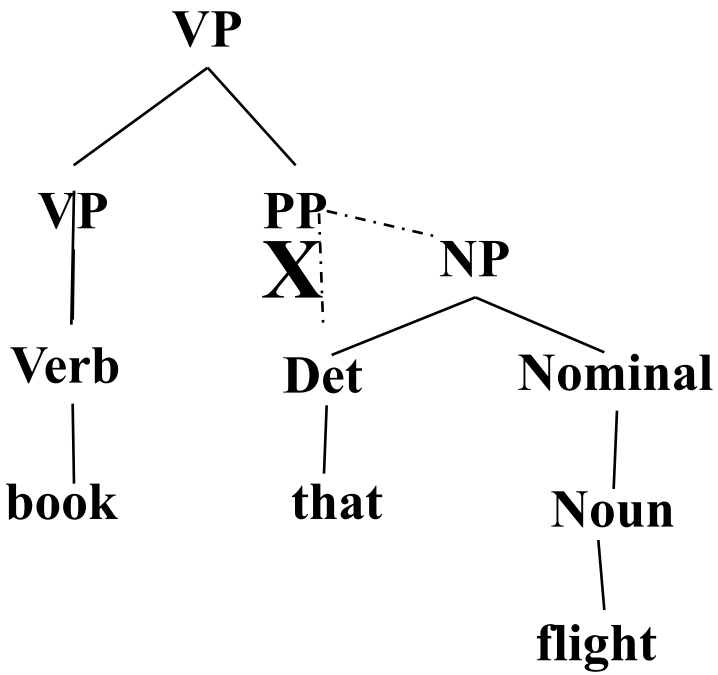
Bottom Up Parsing



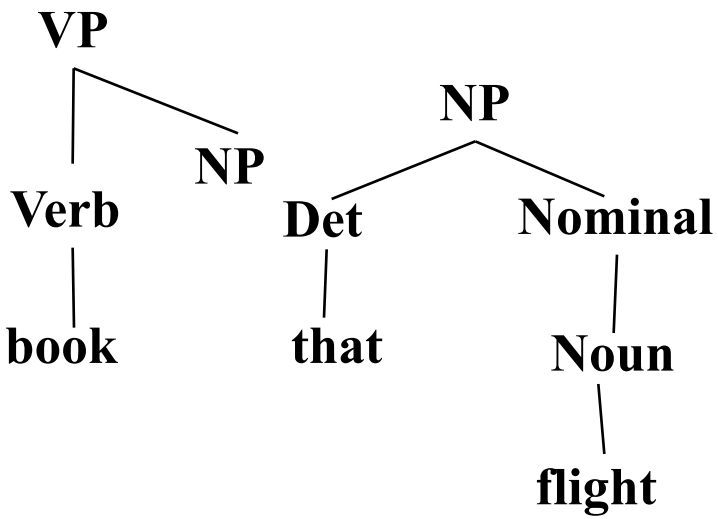
Bottom Up Parsing



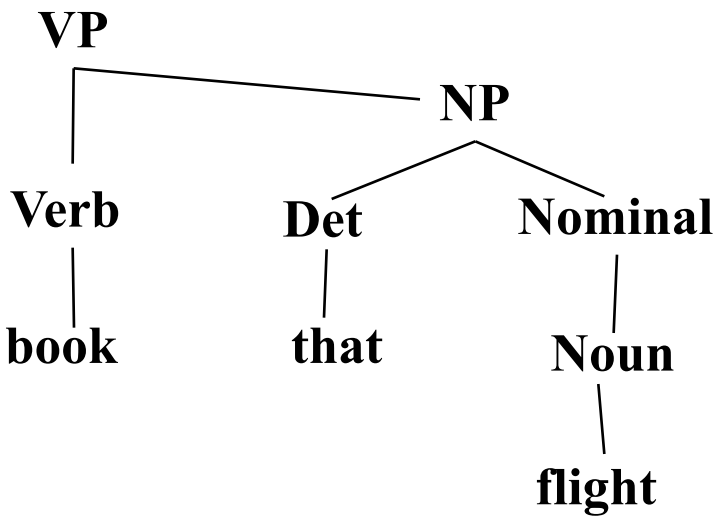
Bottom Up Parsing



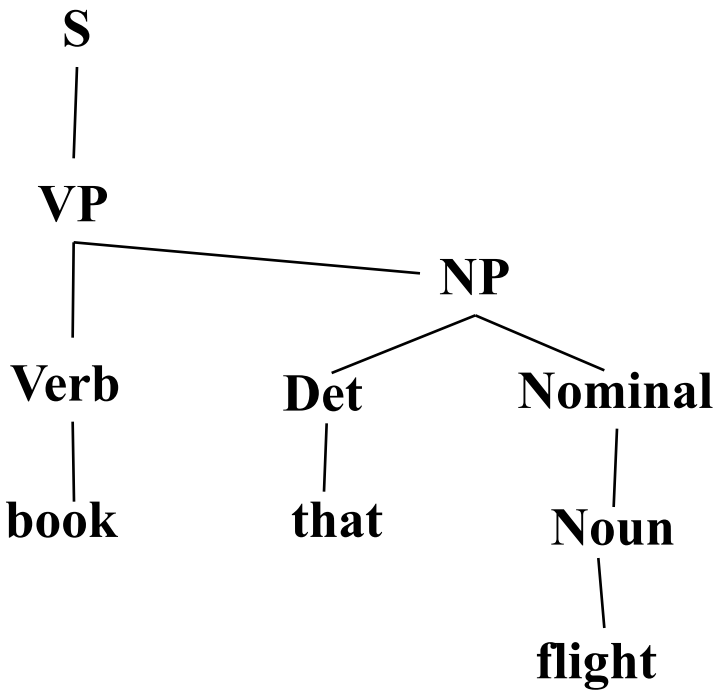
Bottom Up Parsing



Bottom Up Parsing



Bottom Up Parsing



Top Down vs. Bottom Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memoizing) critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.
- Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where n is the length of the input string.

Dynamic Programming Parsing Methods

- **CKY** (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.
- **Earley parser** is based on top-down parsing and does not require normalizing grammar but is more complex.
- More generally, **chart parsers** retain completed phrases in a chart and can combine top-down and bottom-up search.

CKY

- First, the grammar must be converted to **Chomsky normal form (CNF)** in which productions must have either exactly 2 non-terminal symbols on the RHS or 1 terminal symbol (lexicon rules).
- Parse bottom-up storing phrases formed from all substrings in a triangular table (chart).

ATIS English Grammar Conversion

Original Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

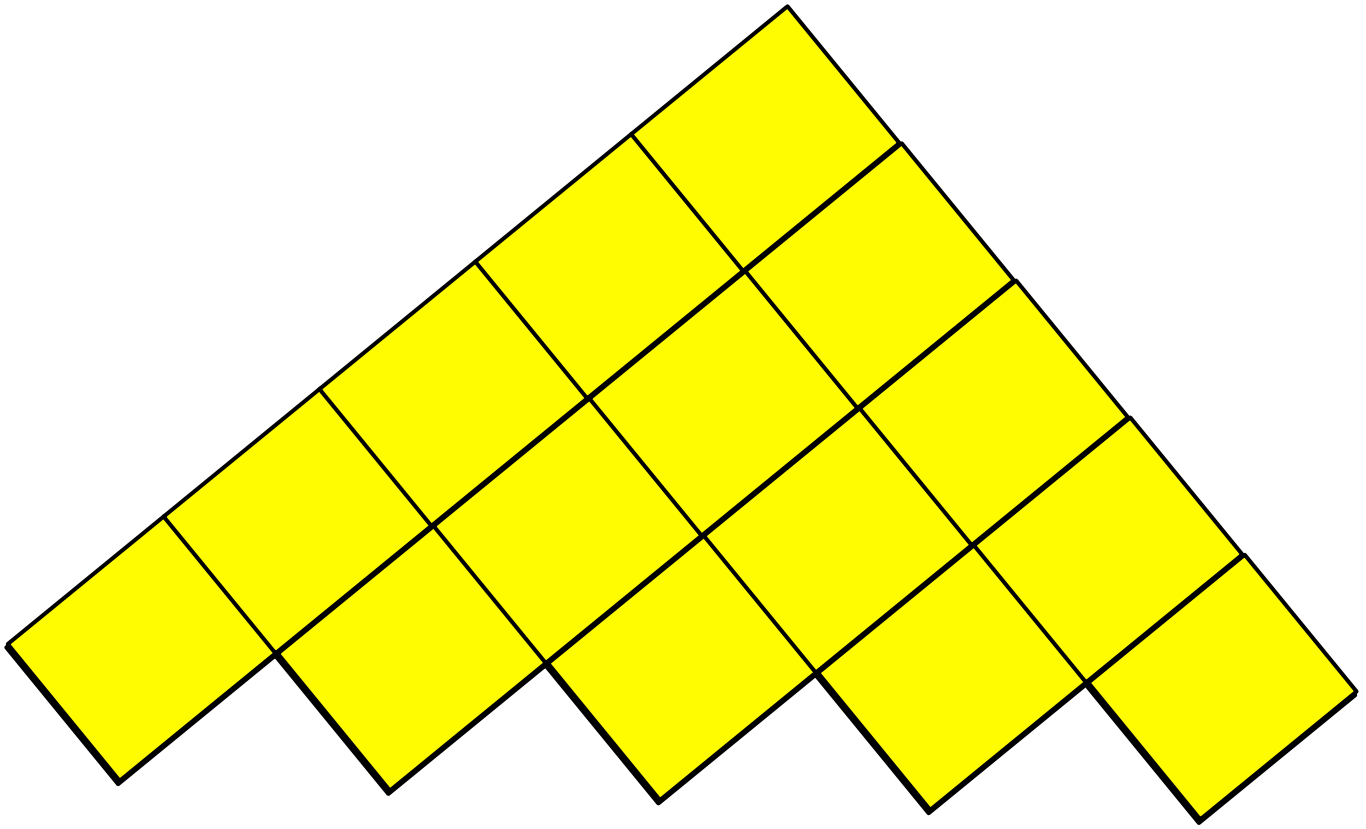
Chomsky Normal Form

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP

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

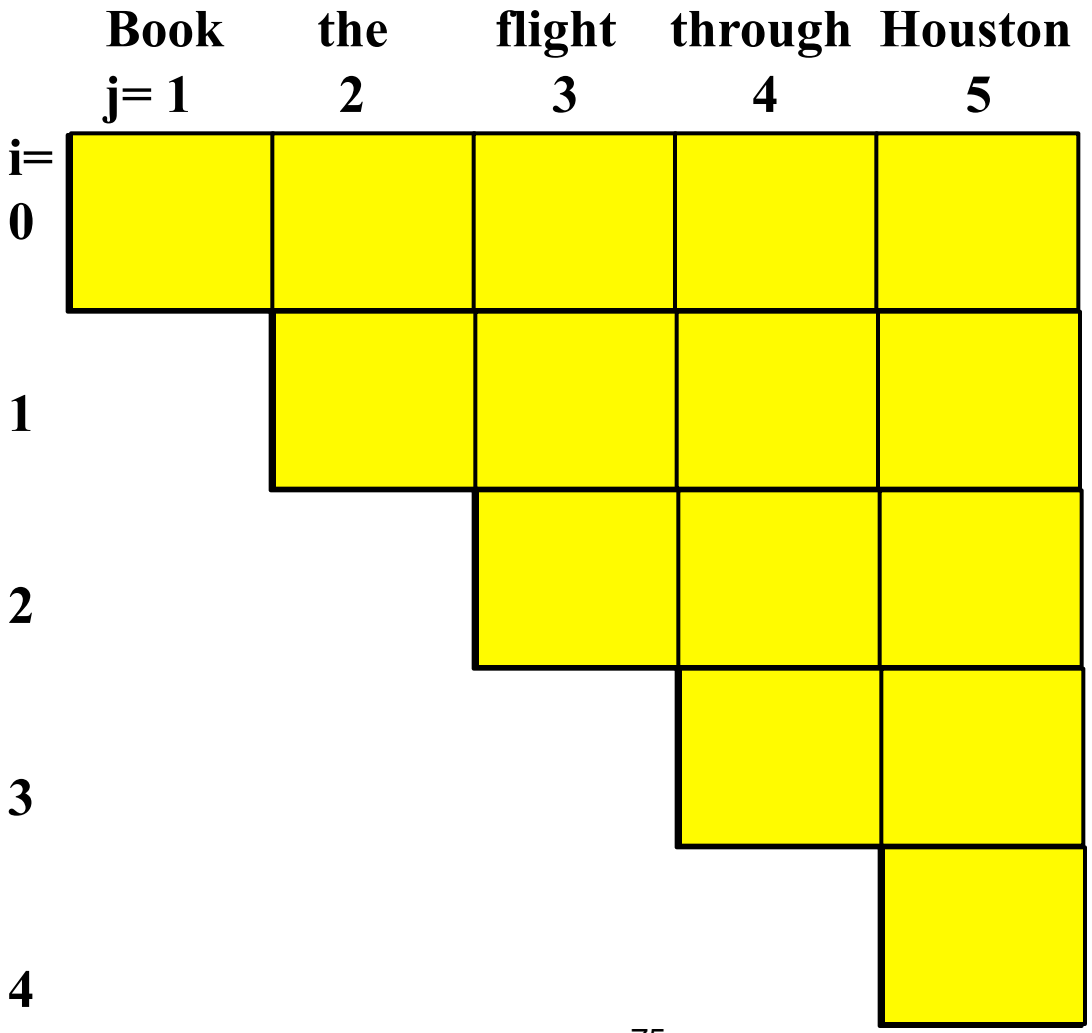
for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid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

CKY Parser



Book the flight through Houston

CKY Parser

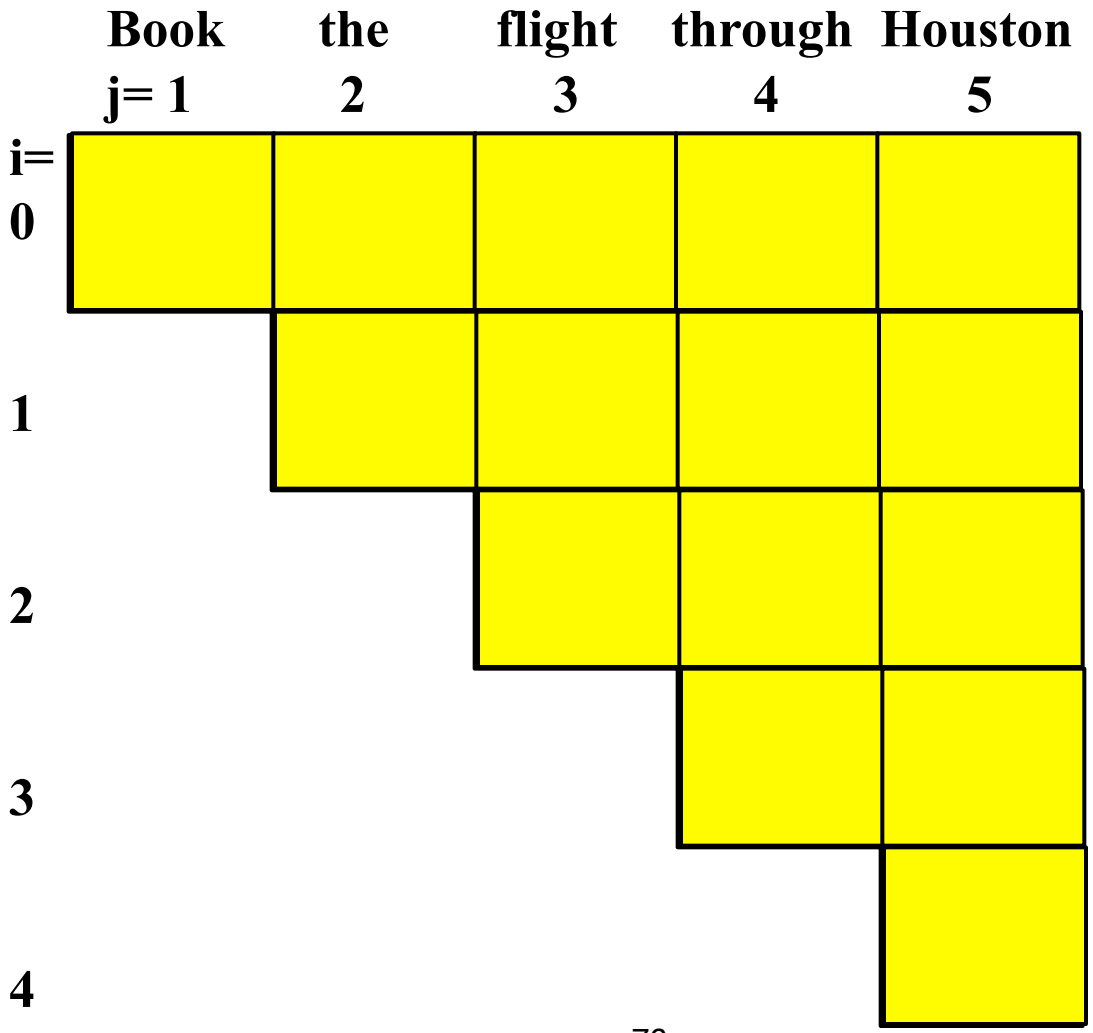


Cell[*i*,*j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

CKY Parser

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

for j ← from 1 to LENGTH(words) do
    table[j − 1, j] ← {A | A → words[j] ∈ grammar}
    for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
            table[i,j] ← table[i,j] ∪
                {A | A → BC ∈ grammar,
                     B ∈ table[i,k],
                     C ∈ table[k, j]}
```

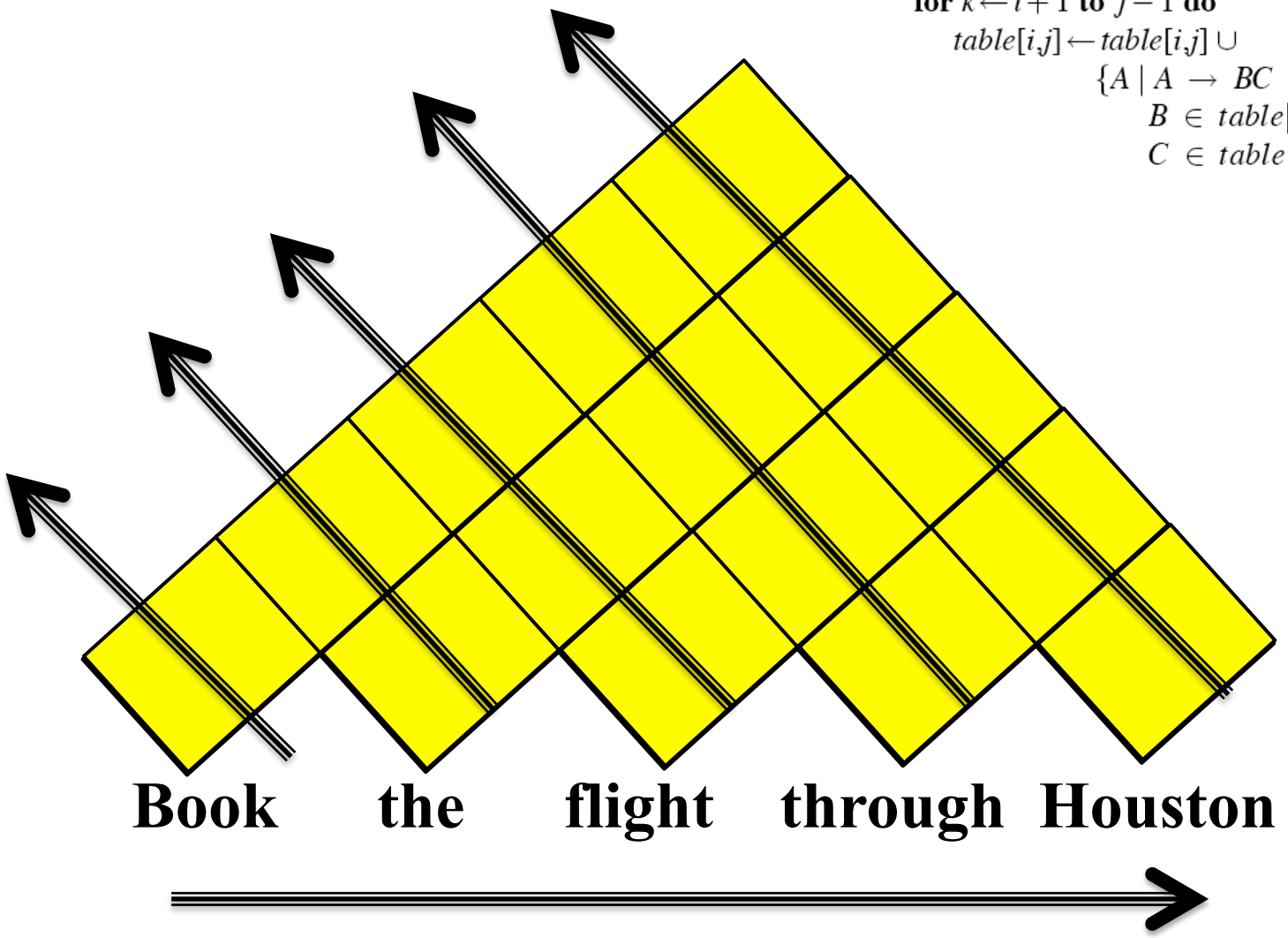


Cell[*i,j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

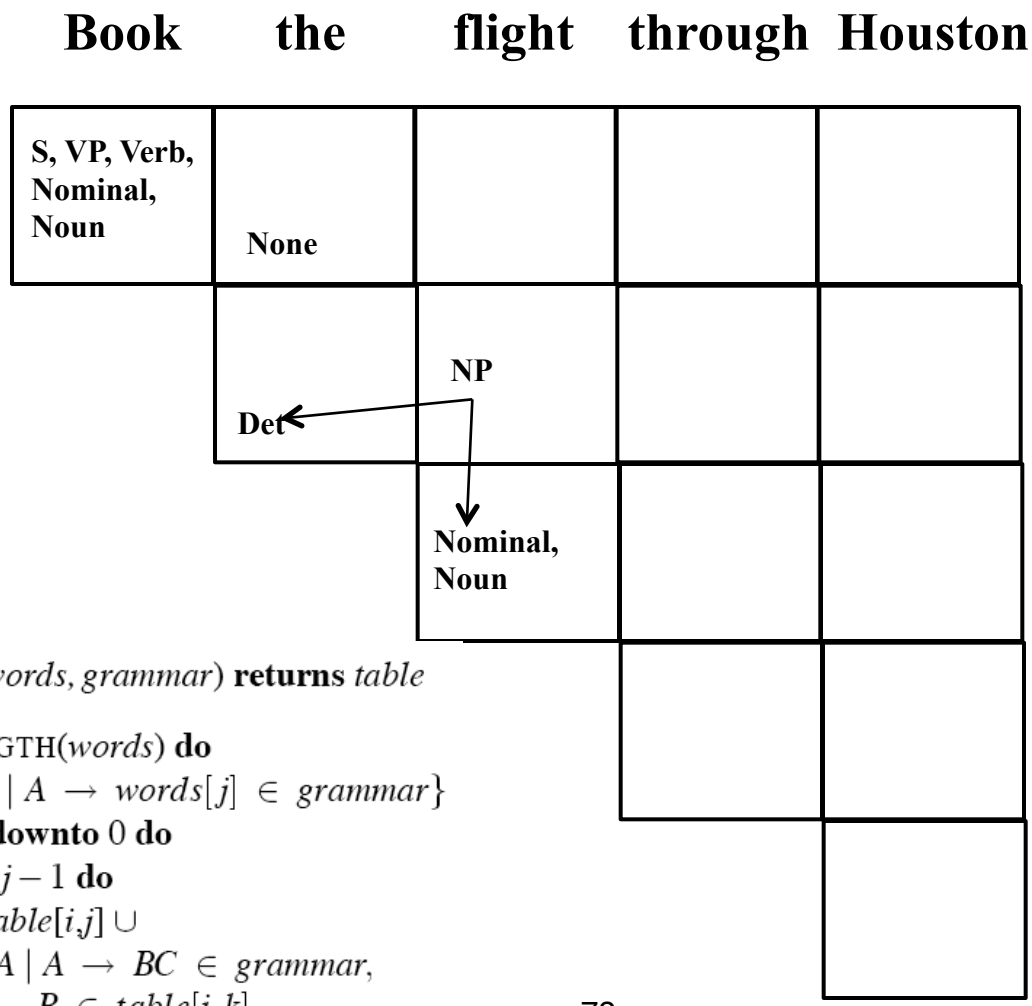
CKY Parser

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

for j ← from 1 to LENGTH(words) do
    table[j − 1, j] ← {A | A → words[j] ∈ grammar}
    for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
            table[i,j] ← table[i,j] ∪
                {A | A → BC ∈ grammar,
                     B ∈ table[i,k],
                     C ∈ table[k, j]}
```



CKY Parser

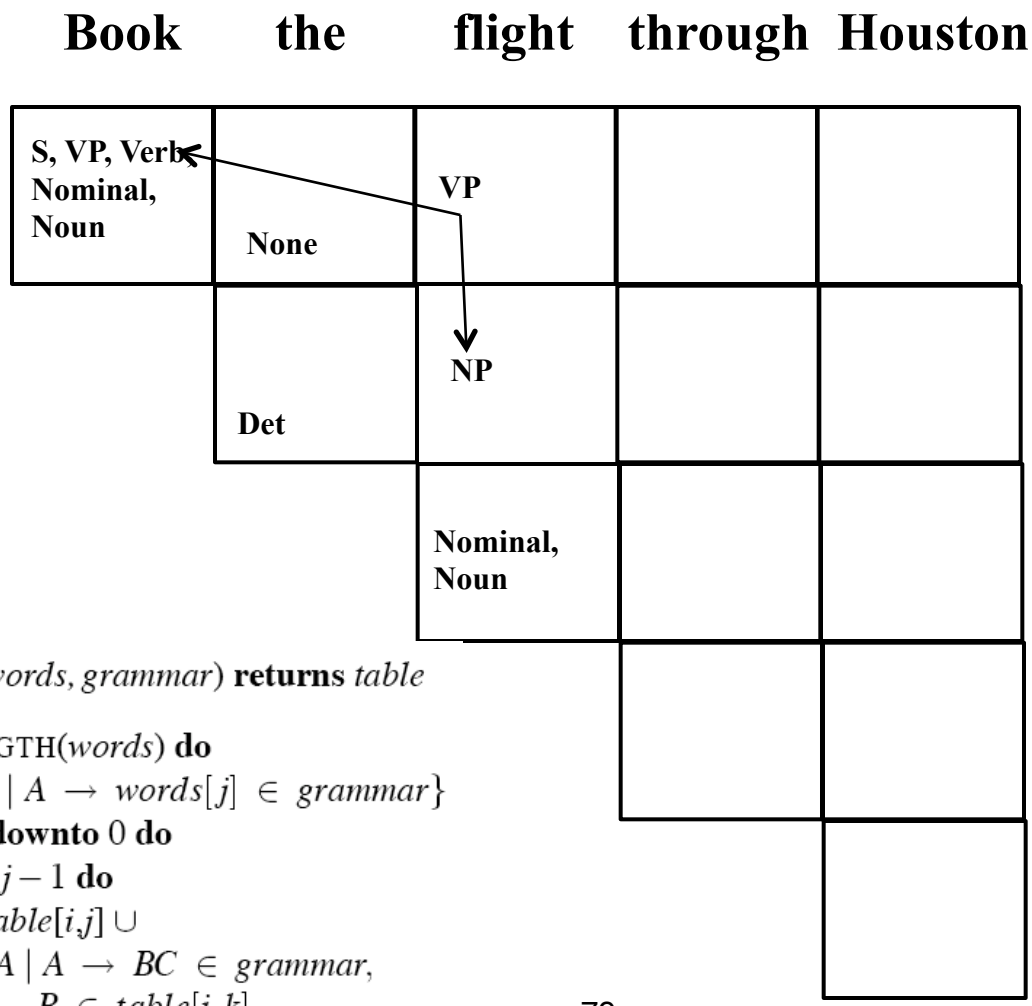



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

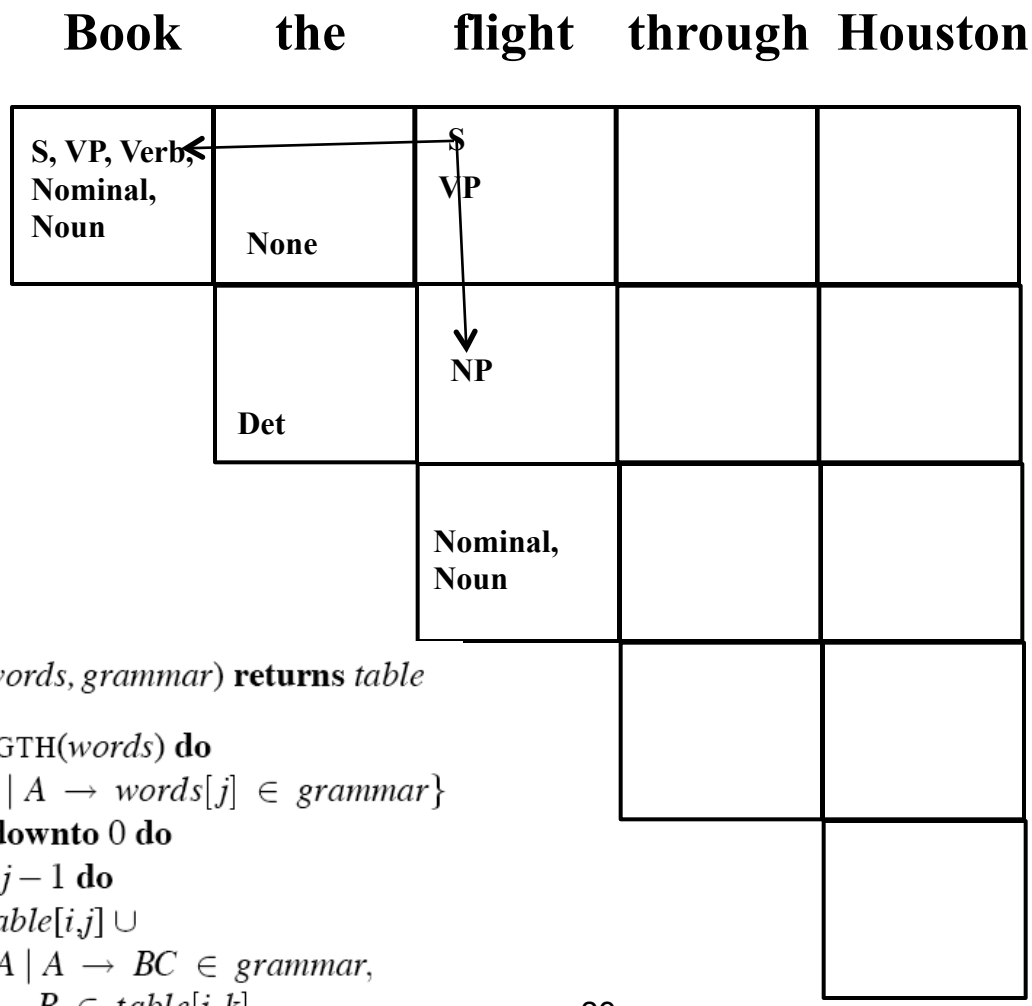



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser



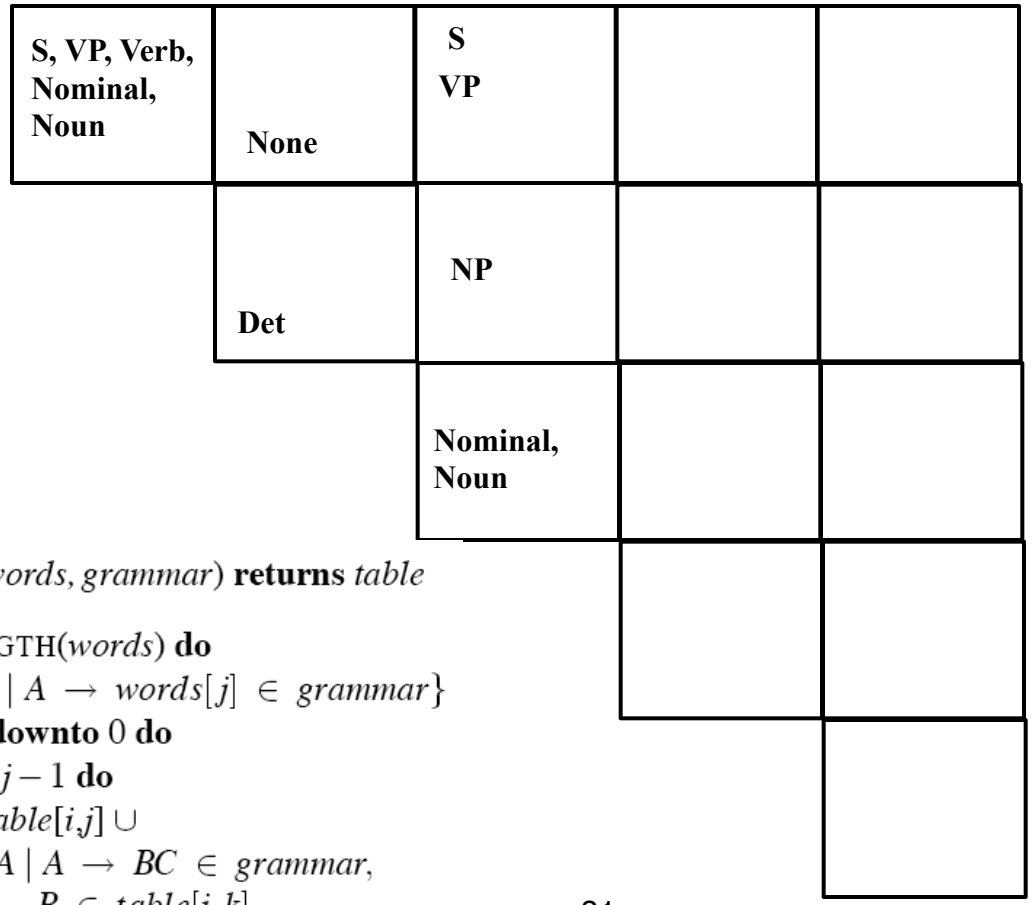

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

Book the flight through Houston




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

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

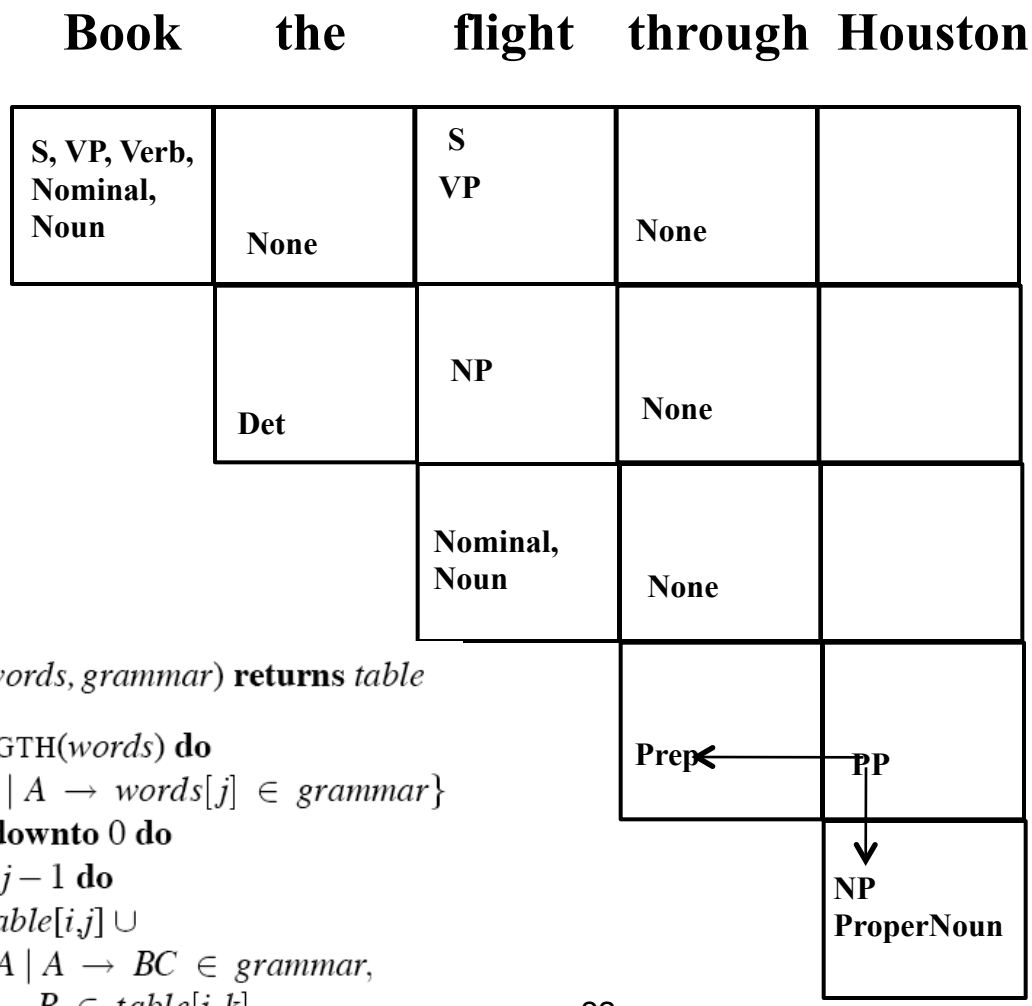
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	

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

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                              B ∈ table[i,k],
                              C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

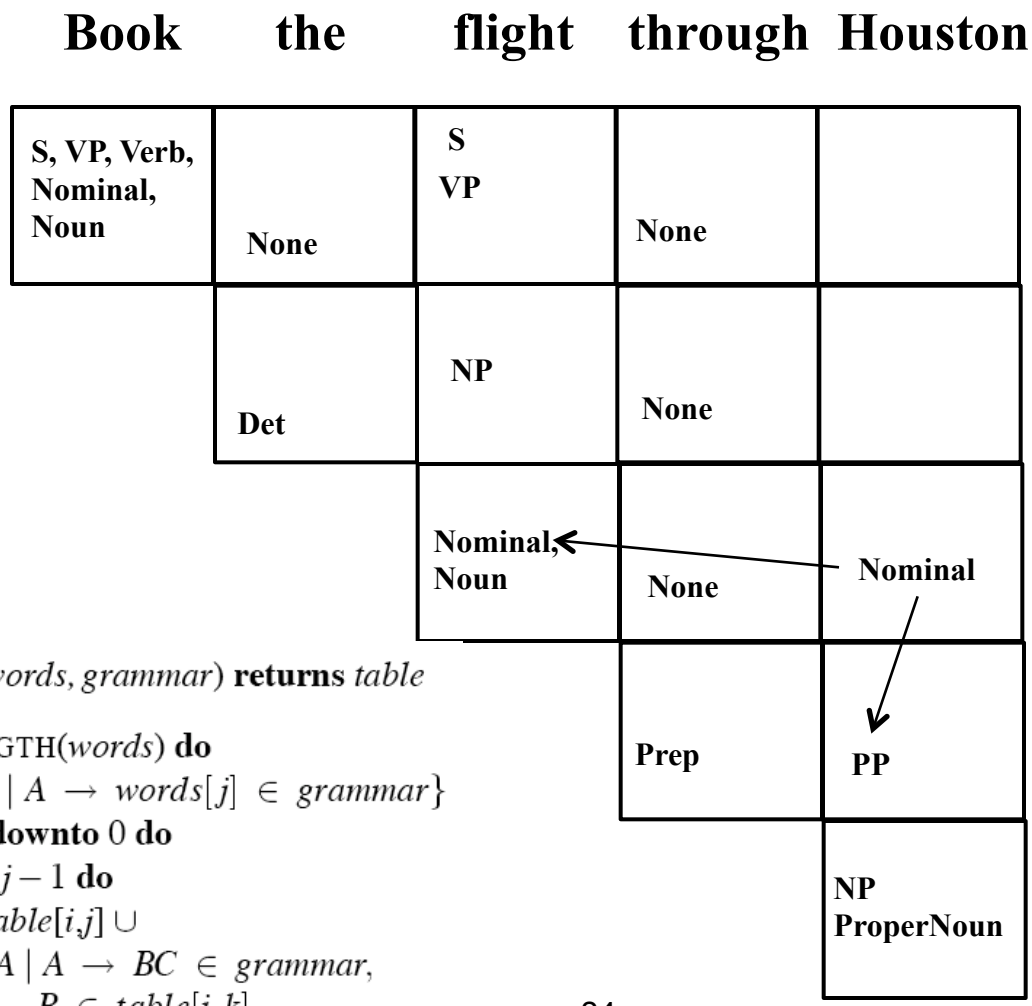



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

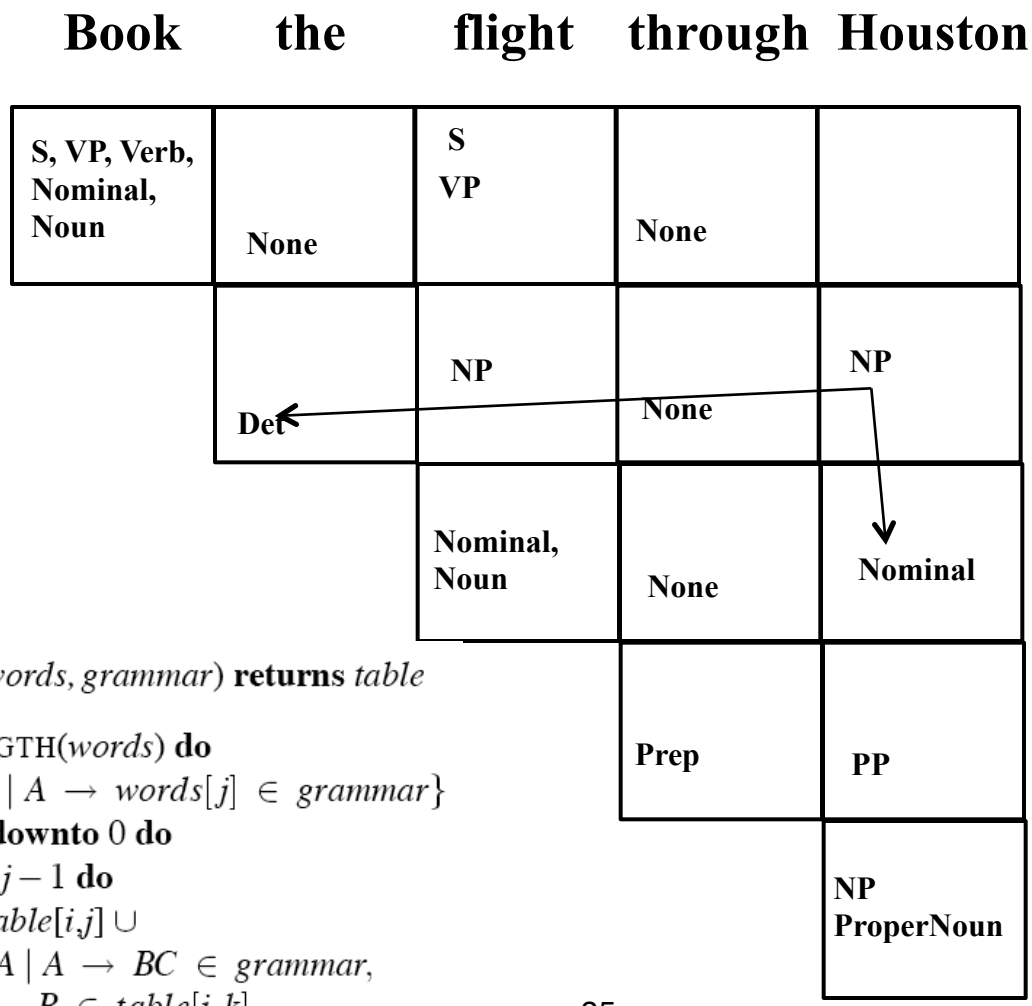



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

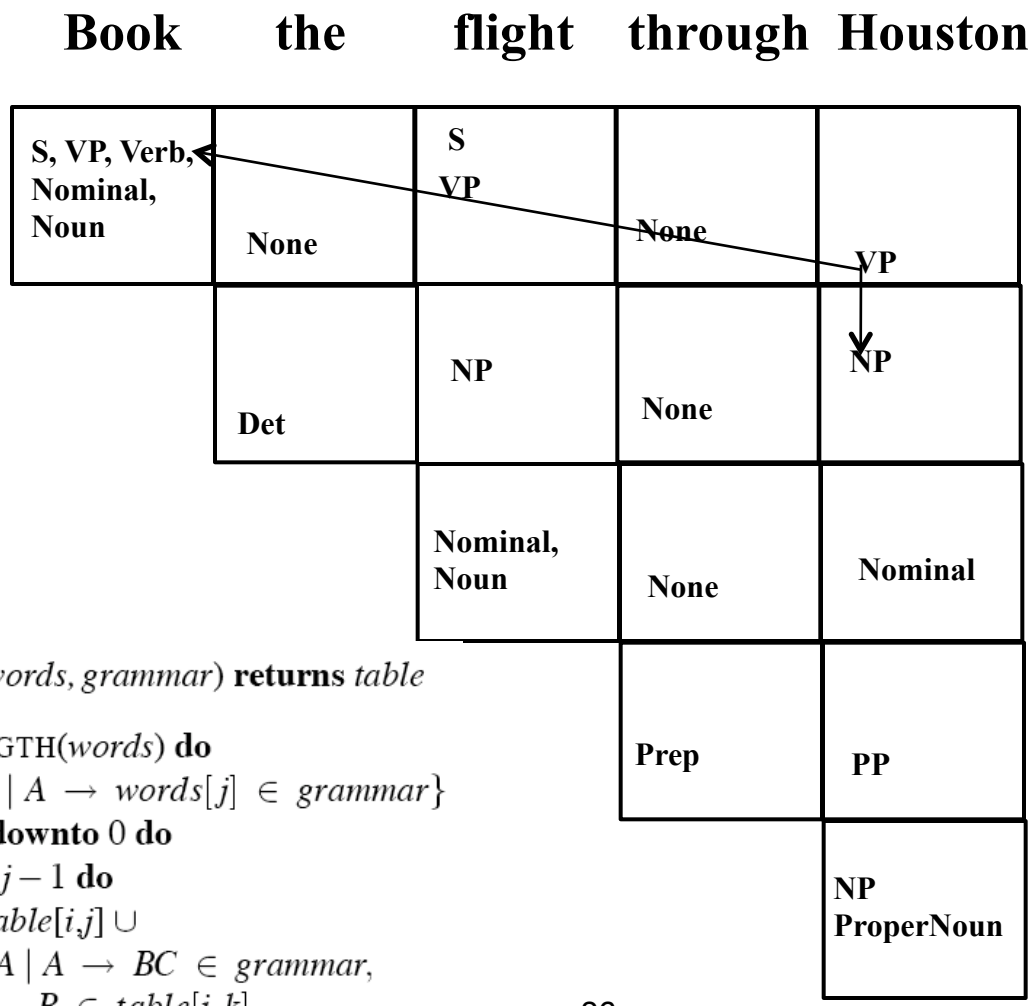



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

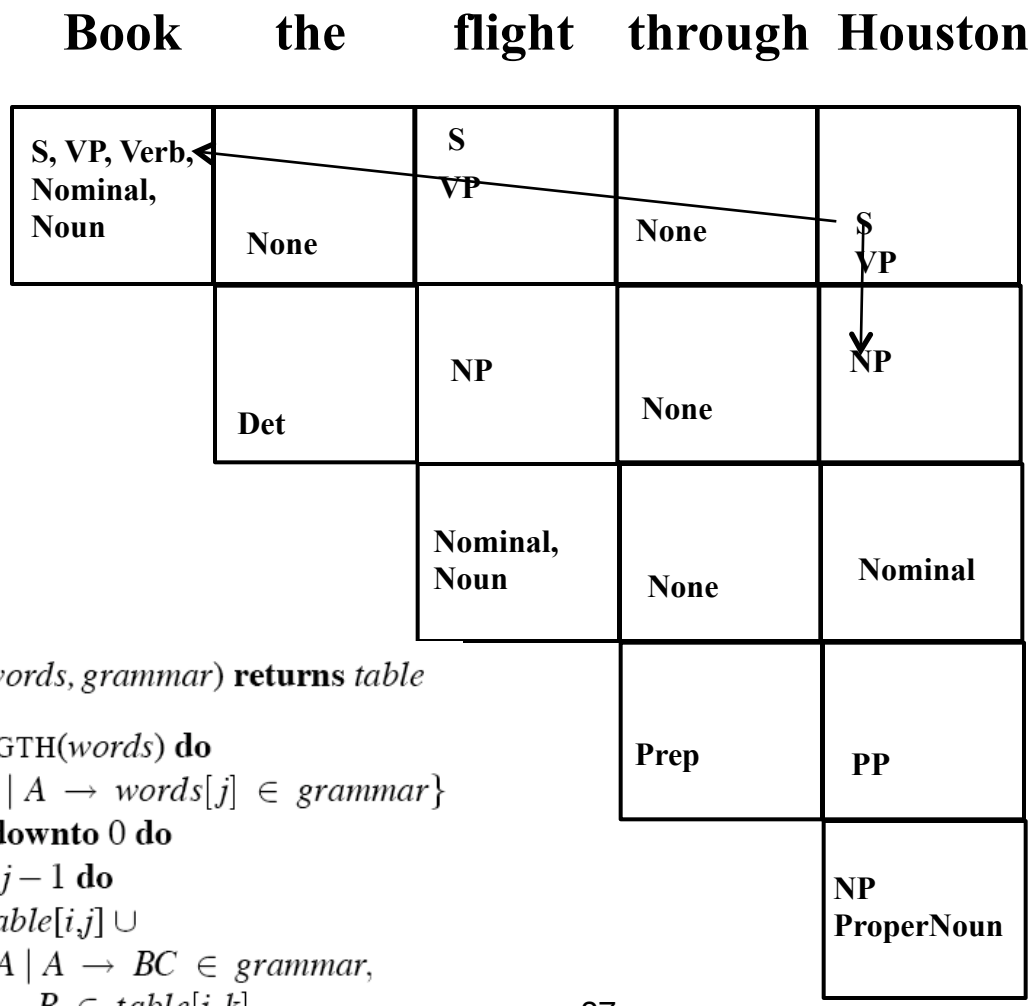



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                              B ∈ table[i,k],
                              C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

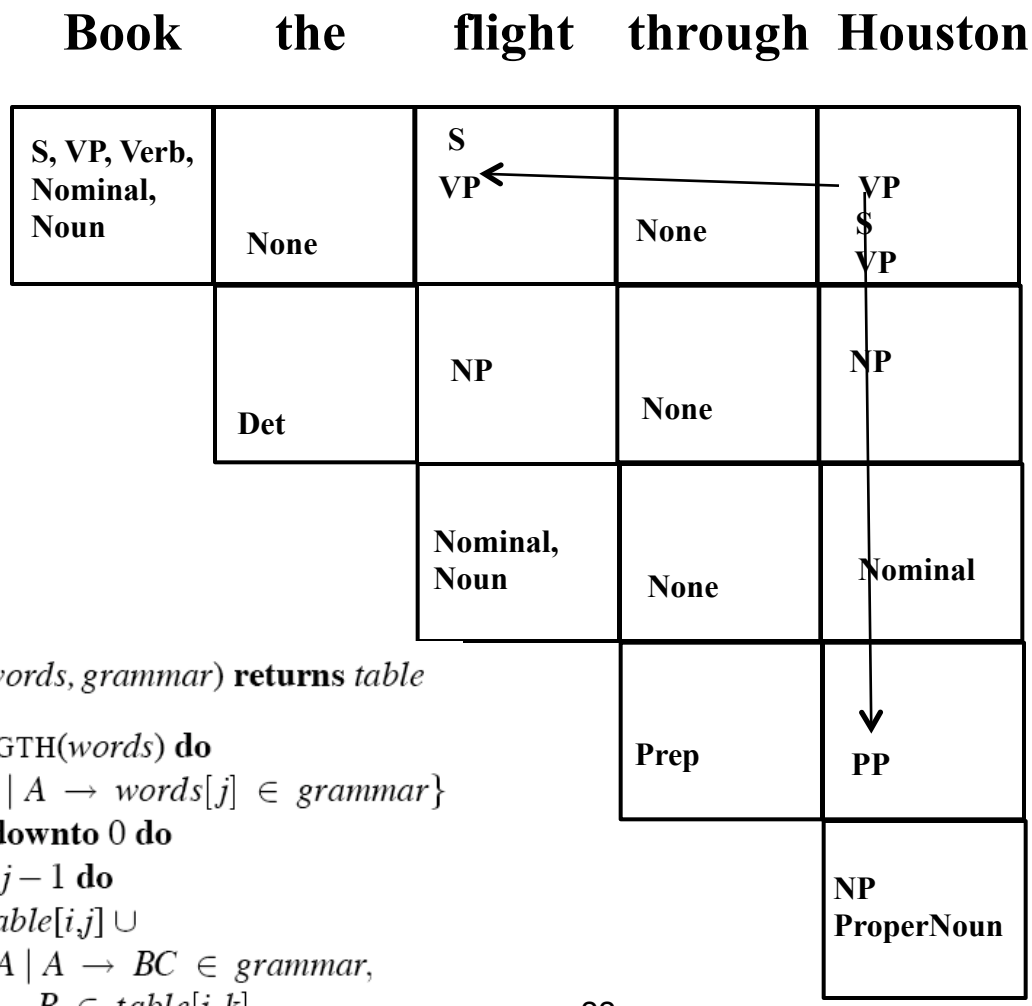



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                              B ∈ table[i,k],
                              C ∈ table[k, j]}
```

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

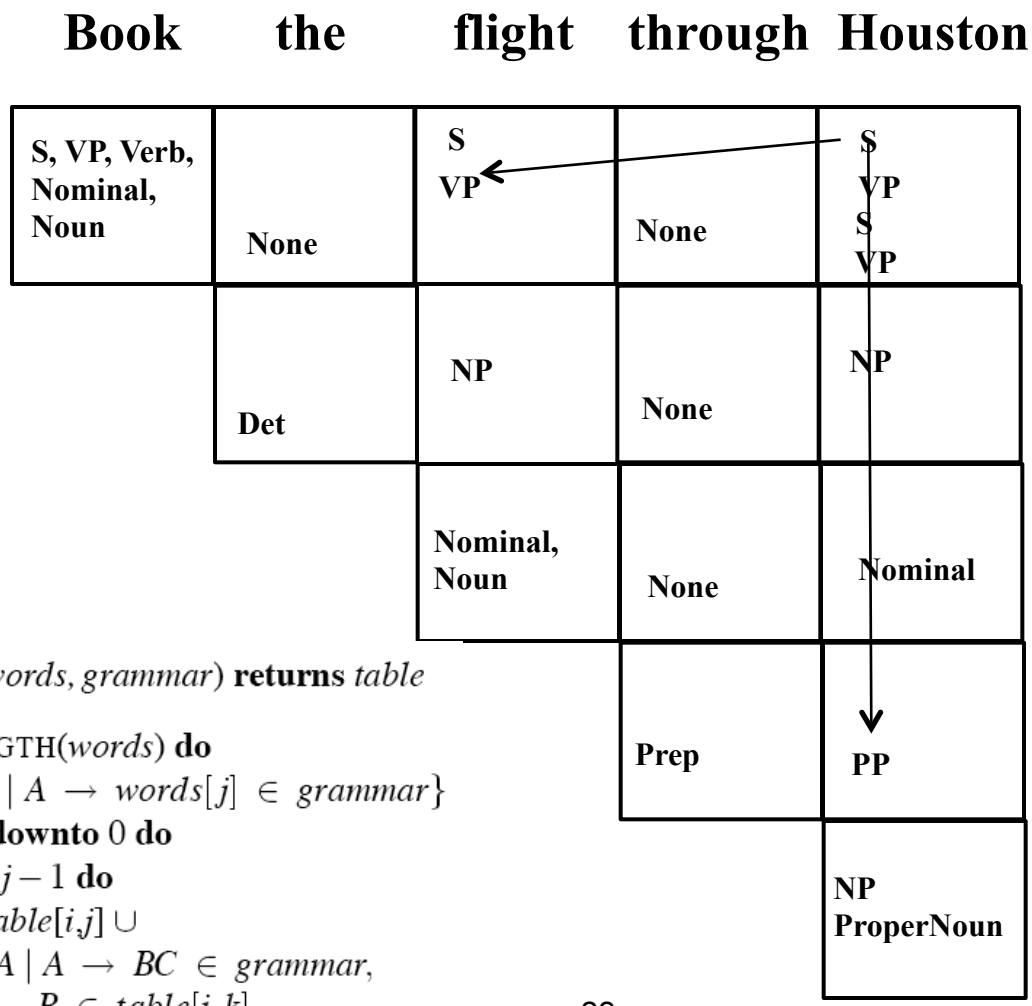


function CKY-PARSE(*words, grammar*) **returns** *table*

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

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser

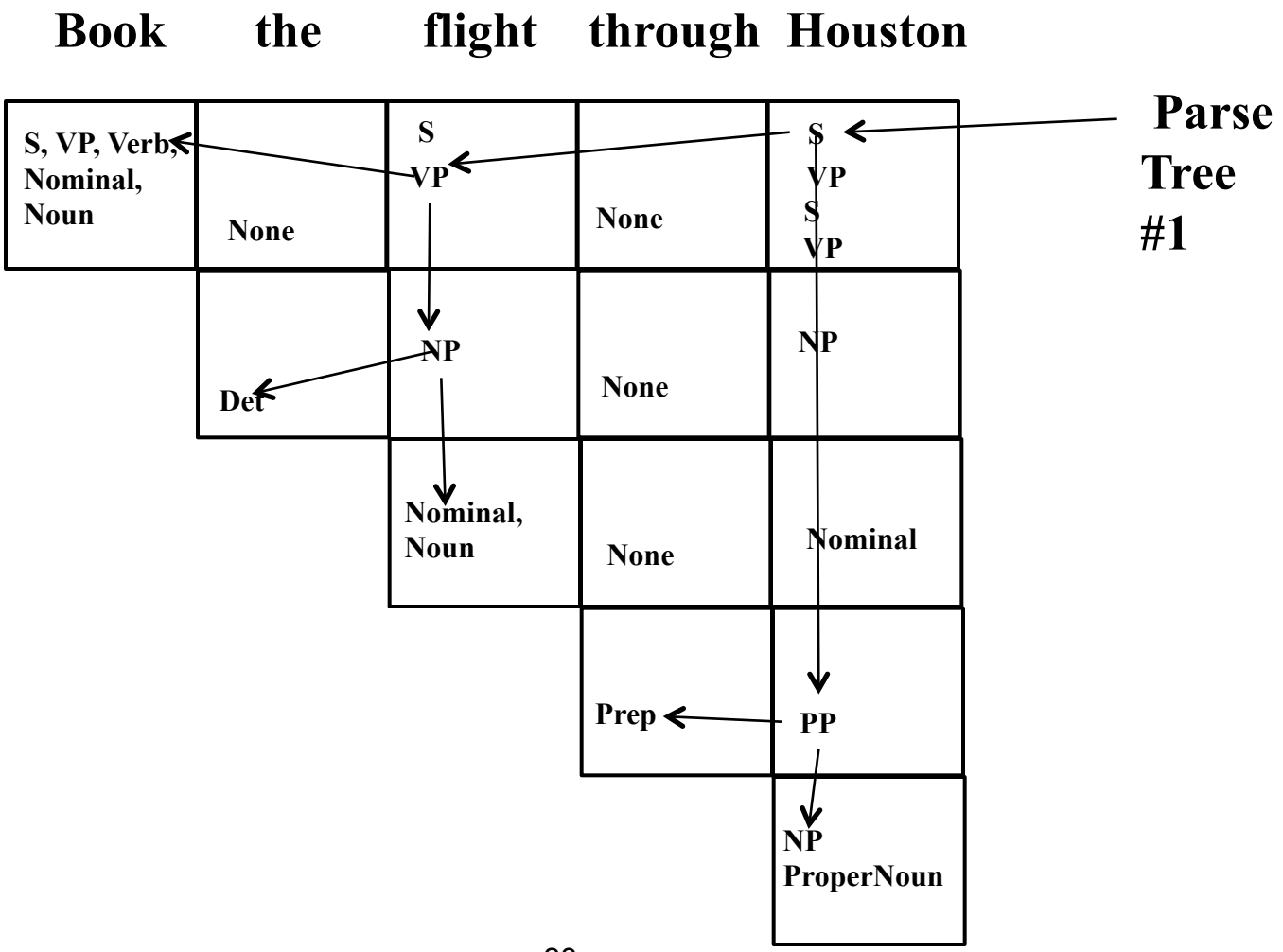


function CKY-PARSE(*words, grammar*) **returns** *table*

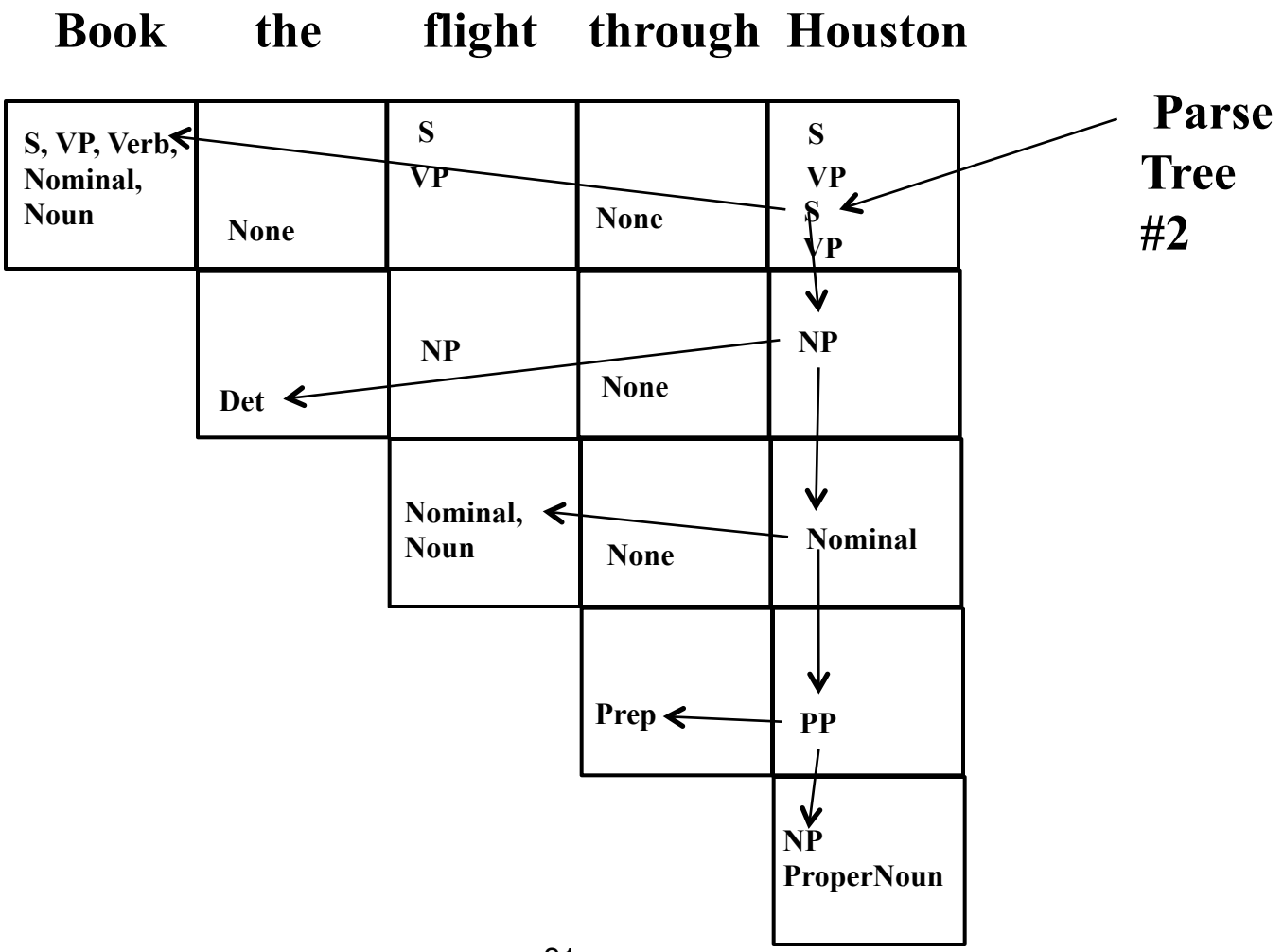
for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid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

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include |…
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight |…
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | …
VP → Verb NP
VP → VP PP
PP → Prep NP
Det → the
Noun → book | flight |…
Prep → in | on | through | …

CKY Parser


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Parse Tree #1
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

CKY Parser


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Parse Tree #2
NP
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

Complexity of CKY (recognition)

- There are $(n(n+1)/2) = O(n^2)$ cells
- Filling each cell requires looking at every possible split point between the two non-terminals needed to introduce a new phrase.
- There are $O(n)$ possible split points.
- Total time complexity is $O(n^3)$

Complexity of CKY (all parses)

- Previous analysis assumes the number of phrase labels in each cell is fixed by the size of the grammar.
- If computing all derivations for each non-terminal, the number of cell entries can expand combinatorially.
- Since the number of parses can be exponential, so is the complexity of finding all parse trees.

Effect of CNF on Parse Trees

- Parse trees are for CNF grammar not the original grammar.
- A post-process can repair the parse tree to return a parse tree for the original grammar.

Midterm and Assignment#2

- Assignment #2
  - Average 97; Min=89; Max=100
- Midterm
  - Average 68; Min=49; Max=97
  - Model answer

Next Time

- Read J+M Chap 13
- Come with questions about Assignment #3