Syntactic Parsing

Phrase Chunking

- Find all non-recursive noun phrases (NPs) and verb phrases (VPs) in a sentence.
  - [NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs].
  - [NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September]

Phrase Chunking as Sequence Labeling

- Tag individual words with one of 3 tags
  - B (Begin) word starts new target phrase
  - I (Inside) word is part of target phrase but not the first word
  - O (Other) word is not part of target phrase
- Sample for NP chunking
  - He reckons the current account deficit will narrow to only # 1.8 billion in September.

Begin **Inside**
Other

Evaluating Chunking

- Per token accuracy does not evaluate finding correct full chunks. Instead use:

$$\text{Precision} = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}$$

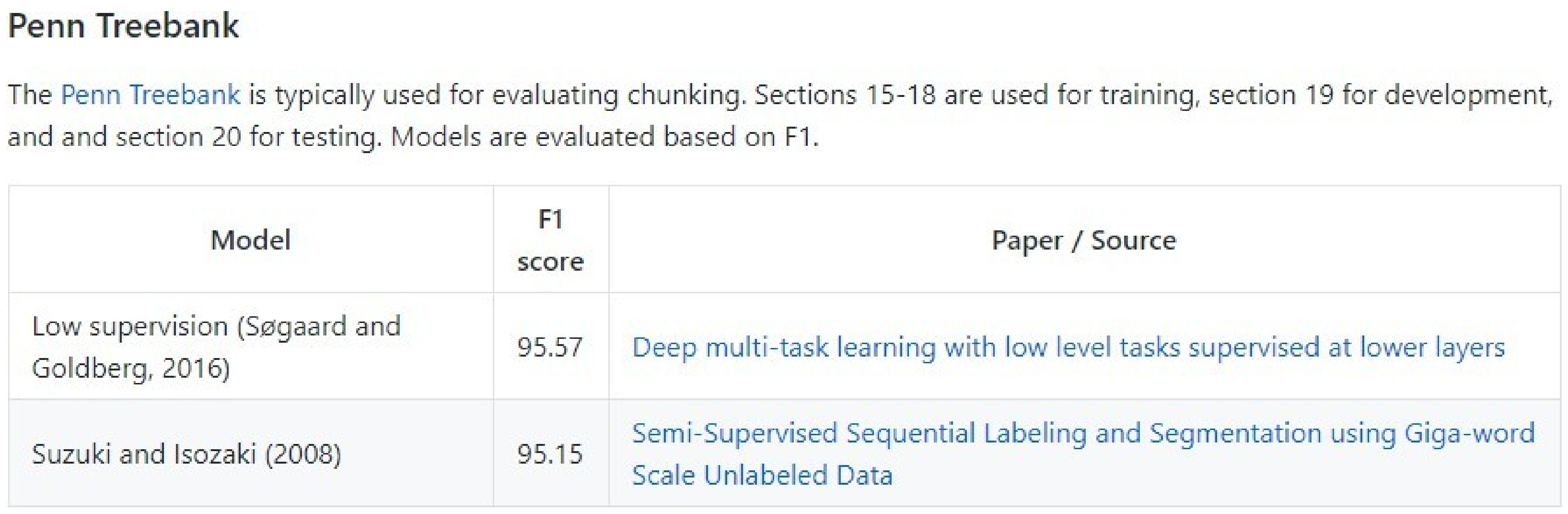
$$\text{Recall} = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}$$

- Take harmonic mean to produce a single evaluation metric called F measure.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \qquad F_1 = \frac{1}{(\frac{1}{P} + \frac{1}{R})/2} = \frac{2PR}{P + R}$$

Current Chunking Results

- Best system for NP chunking: F_1=96%
- Typical results for finding range of chunk types (CONLL 2000 shared task: NP, VP, PP, ADV, SBAR, ADJP)



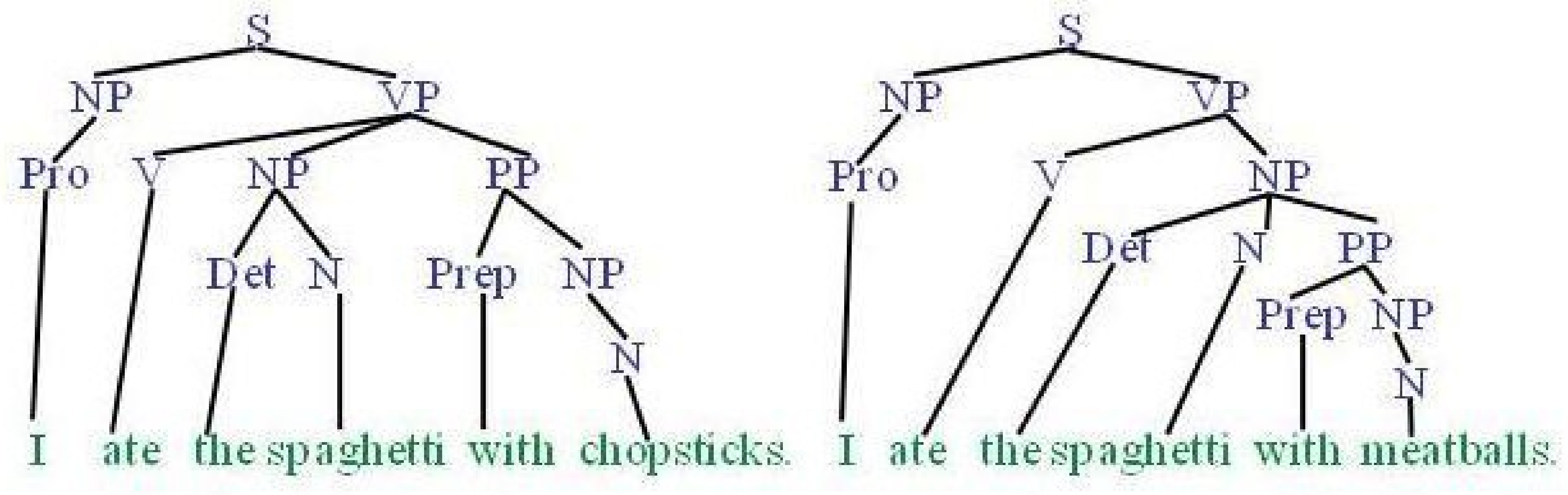
Penn Treebank

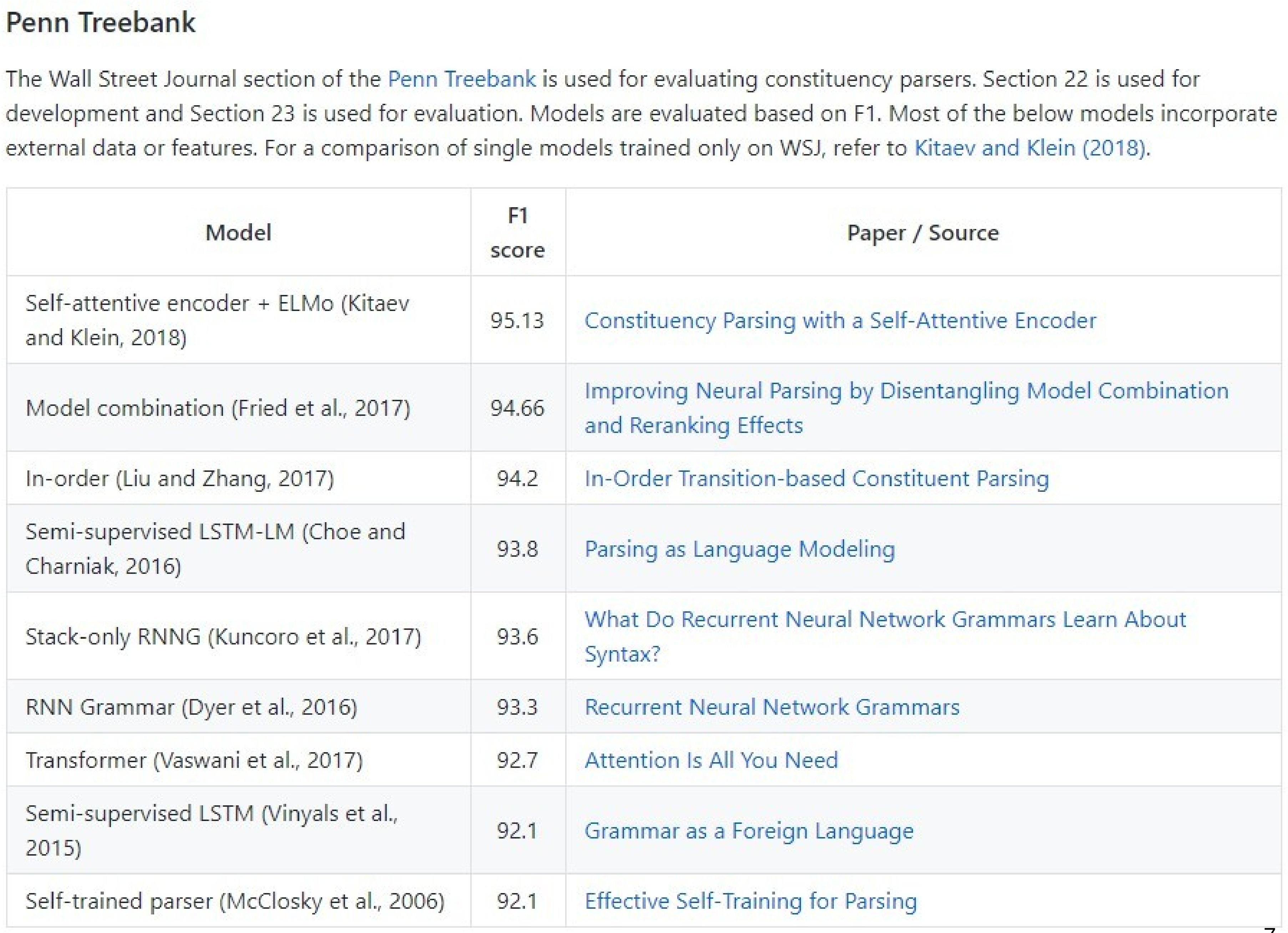
The Penn Treebank is typically used for evaluating chunking. Sections 15-18 are used for training, section 19 for development, and and section 20 for testing. Models are evaluated based on F1.

Model	F1 score	Paper / Source
Low supervision (Søgaard and Goldberg, 2016)	95.57	Deep multi-task learning with low level tasks supervised at lower layers
Suzuki and Isozaki (2008)	95.15	Semi-Supervised Sequential Labeling and Segmentation using Giga-word Scale Unlabeled Data

Syntactic Parsing

- Produce the correct syntactic parse tree for a sentence.


S
NP
VP
Pro
V
NP
PP
Det
N
Prep
NP
N
I ate the spaghetti with chopsticks.
S
NP
VP
Pro
V
NP
Det
N
PP
Prep
NP
N
I ate the spaghetti with meatballs.



Penn Treebank

The Wall Street Journal section of the Penn Treebank is used for evaluating constituency parsers. Section 22 is used for development and Section 23 is used for evaluation. Models are evaluated based on F1. Most of the below models incorporate external data or features. For a comparison of single models trained only on WSJ, refer to Kitaev and Klein (2018).

Model	F1 score	Paper / Source
Self-attentive encoder + ELMo (Kitaev and Klein, 2018)	95.13	Constituency Parsing with a Self-Attentive Encoder
Model combination (Fried et al., 2017)	94.66	Improving Neural Parsing by Disentangling Model Combination and Reranking Effects
In-order (Liu and Zhang, 2017)	94.2	In-Order Transition-based Constituent Parsing
Semi-supervised LSTM-LM (Choe and Charniak, 2016)	93.8	Parsing as Language Modeling
Stack-only RNNG (Kuncoro et al., 2017)	93.6	What Do Recurrent Neural Network Grammars Learn About Syntax?
RNN Grammar (Dyer et al., 2016)	93.3	Recurrent Neural Network Grammars
Transformer (Vaswani et al., 2017)	92.7	Attention Is All You Need
Semi-supervised LSTM (Vinyals et al., 2015)	92.1	Grammar as a Foreign Language
Self-trained parser (McClosky et al., 2006)	92.1	Effective Self-Training for Parsing

Context Free Grammars (CFG)

- *N* a set of ***non-terminal symbols*** (or ***variables***)
- a set of ***terminal symbols*** (disjoint from *N*)
- *R* a set of ***productions*** or ***rules*** of the form A → , where A is a non-terminal and is a string of symbols from (*N)**
- S, a designated non-terminal called the ***start symbol***

Simple CFG for ATIS English

Grammar

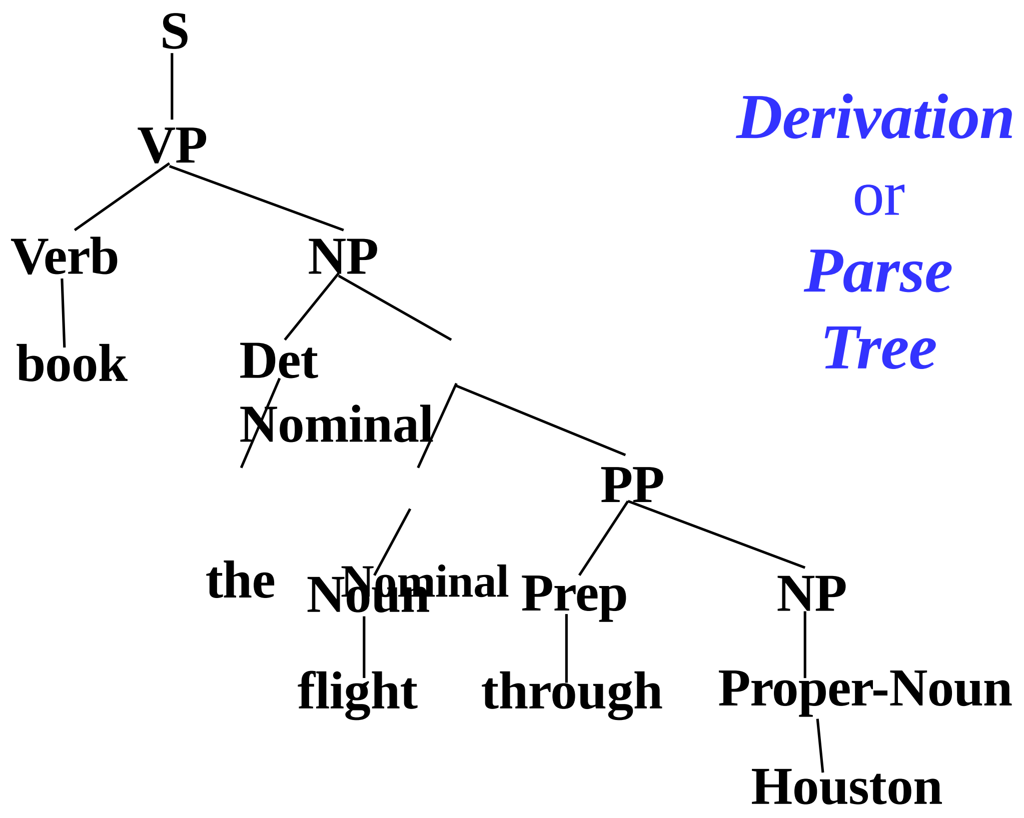
S → NP VP
S → Aux NP
VP S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

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

Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.



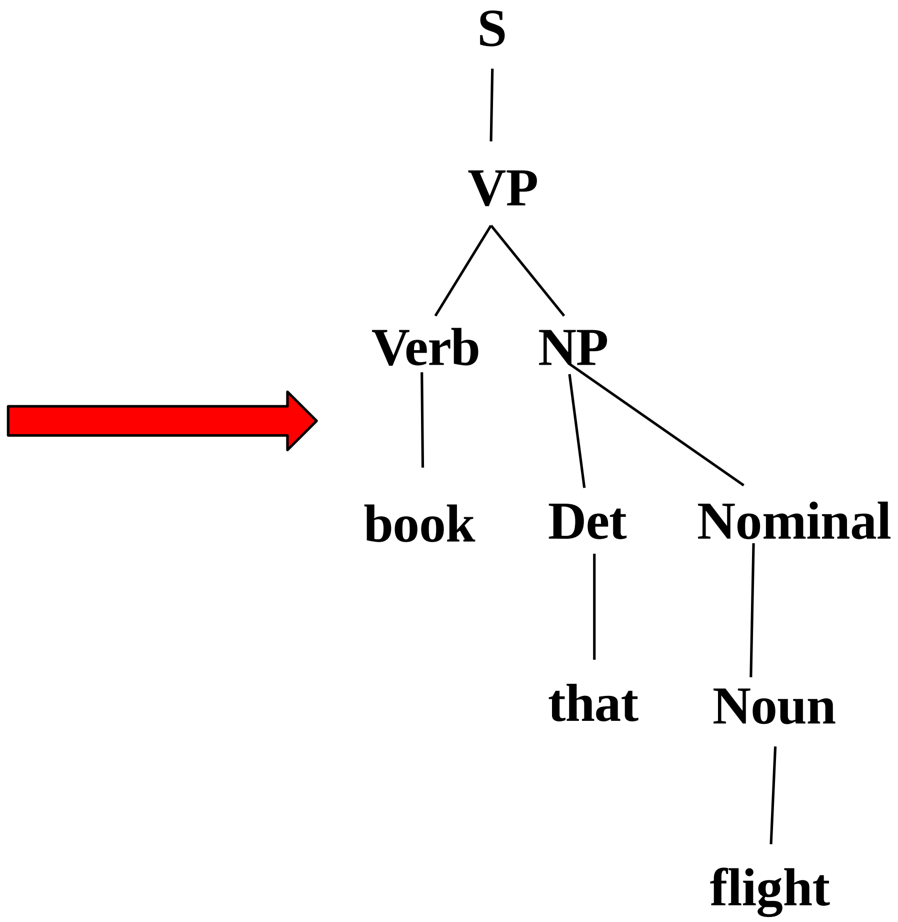
Derivation
or
Parse Tree

Parsing

- Given a string of non-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
    derivations from the terminal symbols in the string.

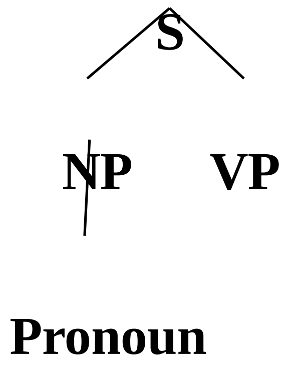
Parsing Example

book that flight

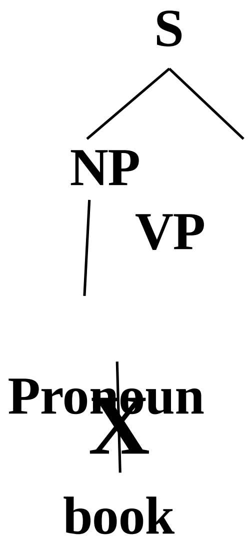


Top Down Parsing

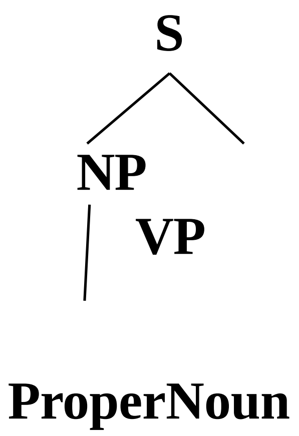
- Start searching space of derivations for the start symbol



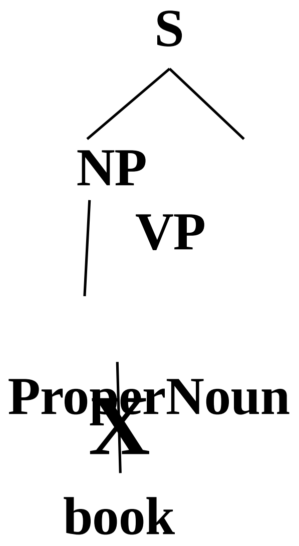
Top Down Parsing



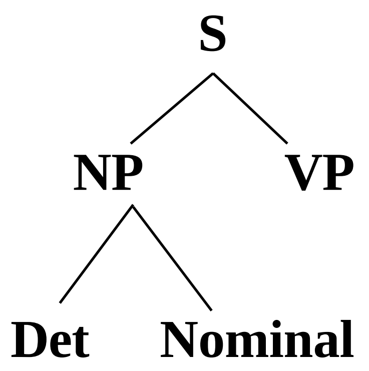
Top Down Parsing



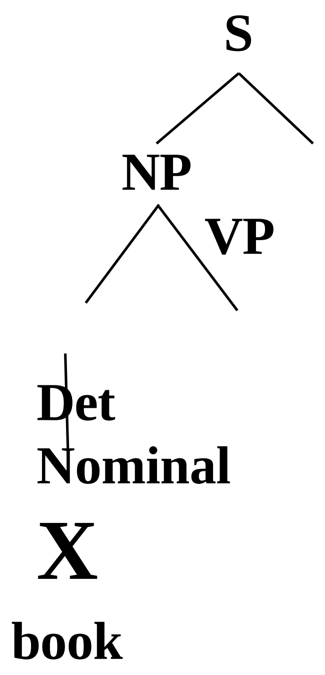
Top Down Parsing



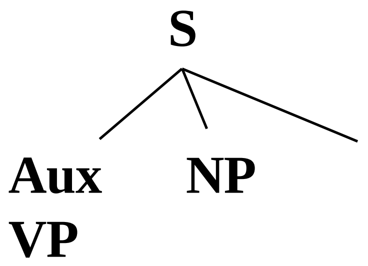
Top Down Parsing



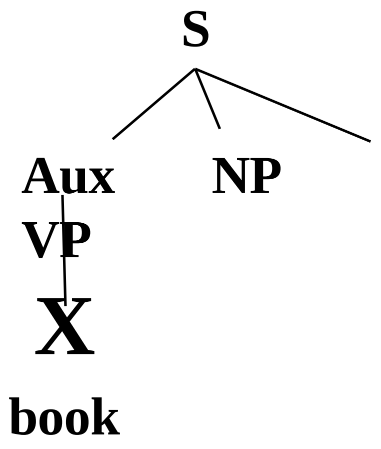
Top Down Parsing



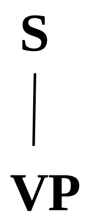
Top Down Parsing



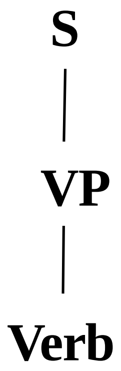
Top Down Parsing



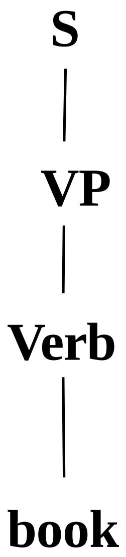
Top Down Parsing



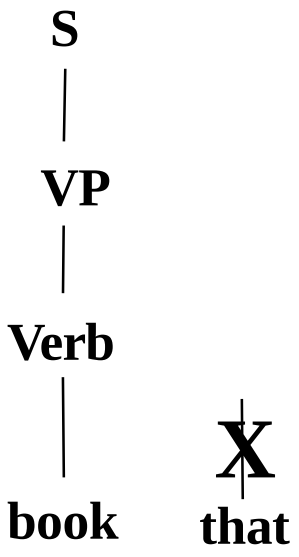
Top Down Parsing



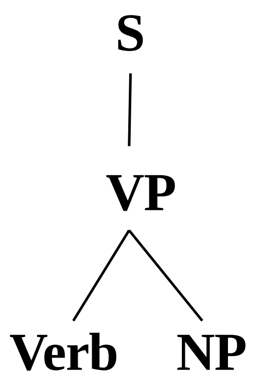
Top Down Parsing



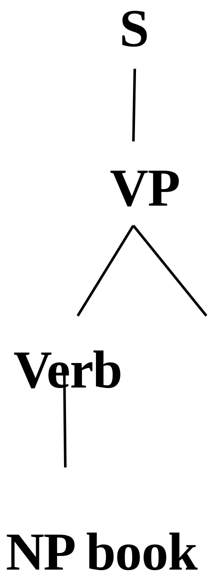
Top Down Parsing



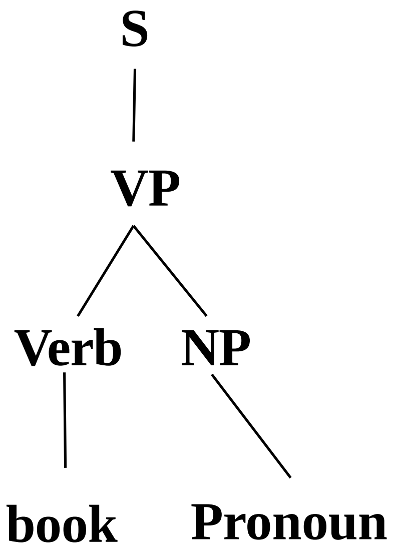
Top Down Parsing



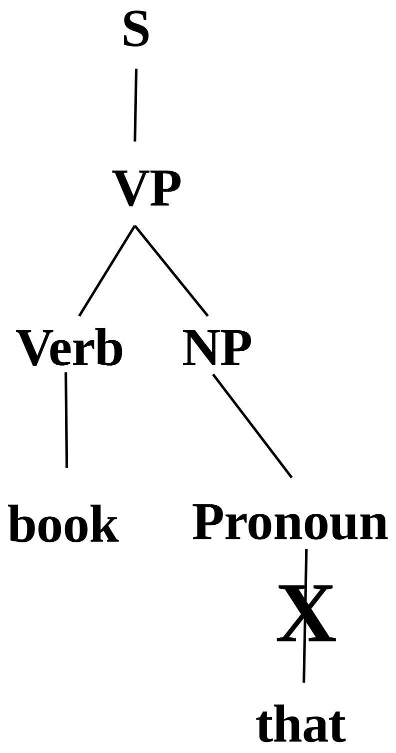
Top Down Parsing



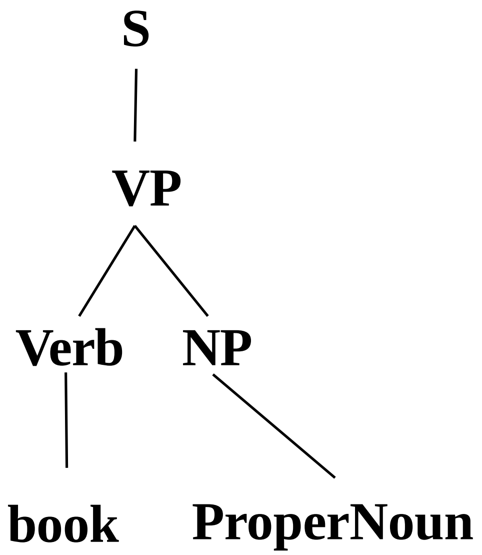
Top Down Parsing



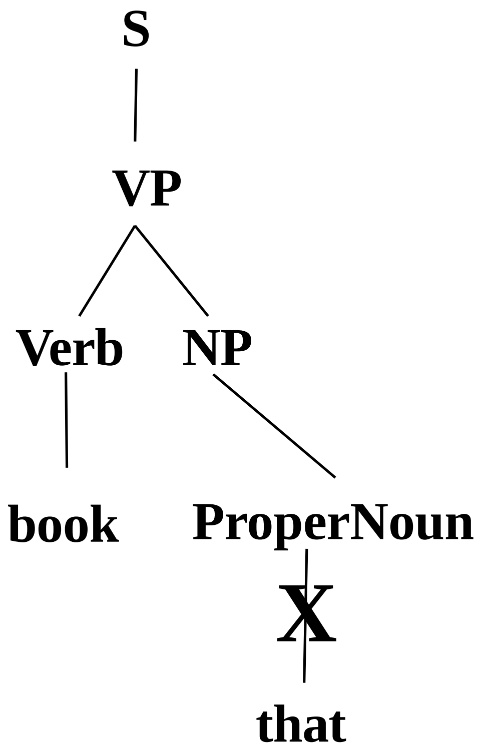
Top Down Parsing



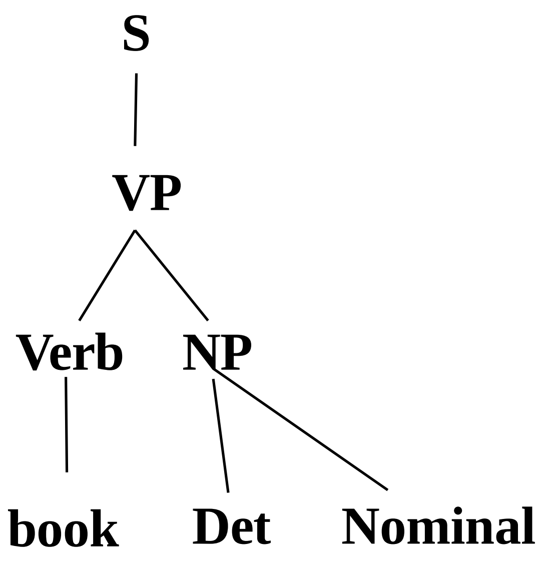
Top Down Parsing



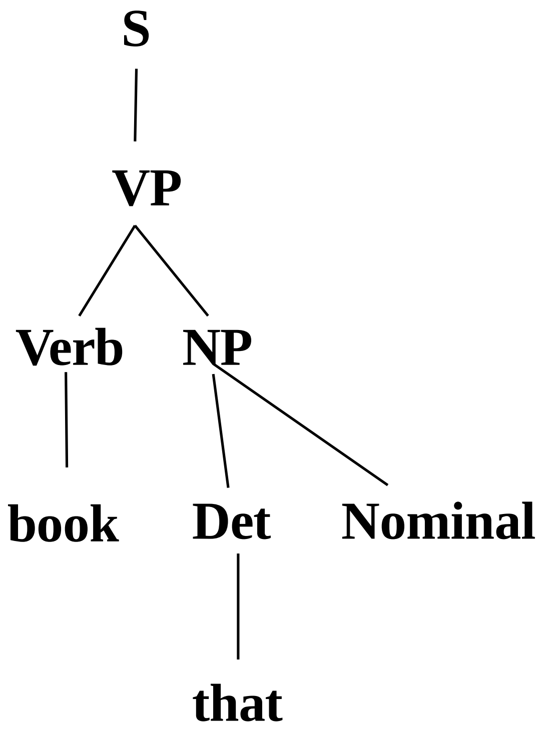
Top Down Parsing



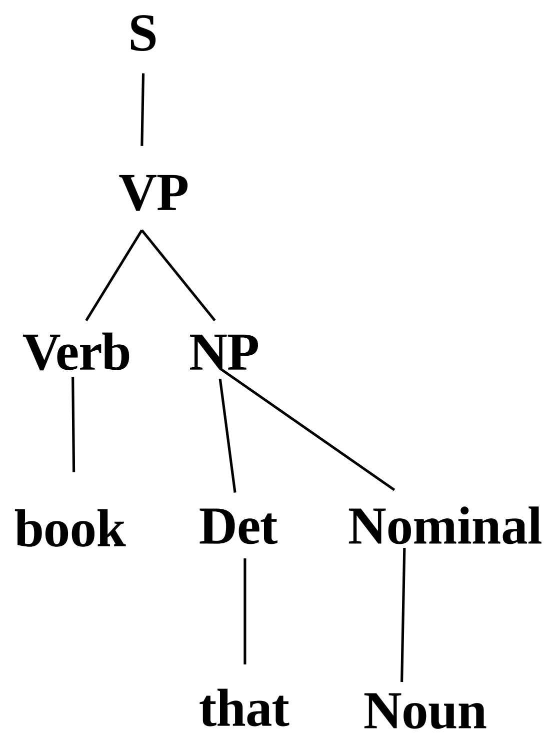
Top Down Parsing



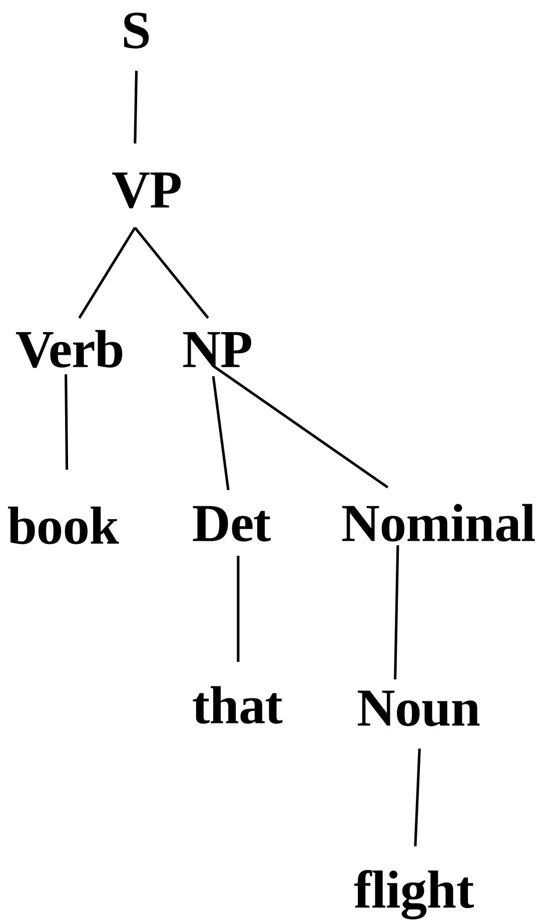
Top Down Parsing



Top Down Parsing



Top Down Parsing

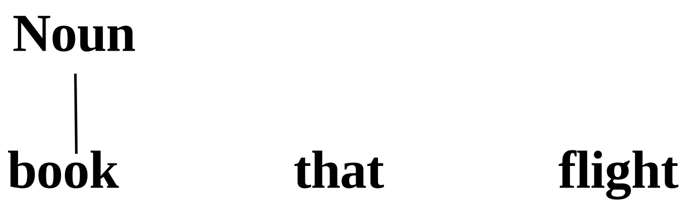


Bottom Up Parsing

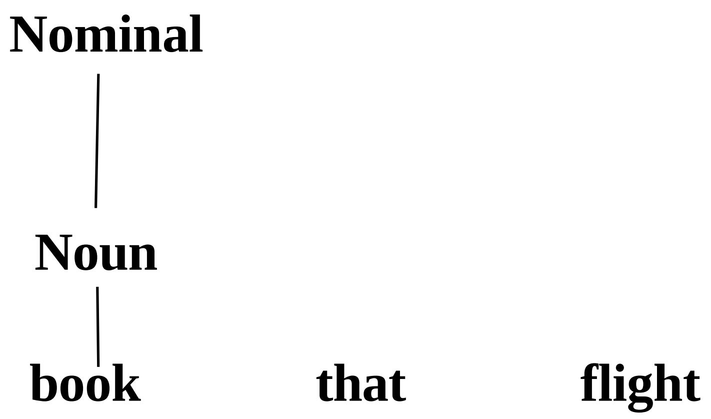
- **Start search space of reverse derivations from the terminal symbols in the string**

book **that** **flight**

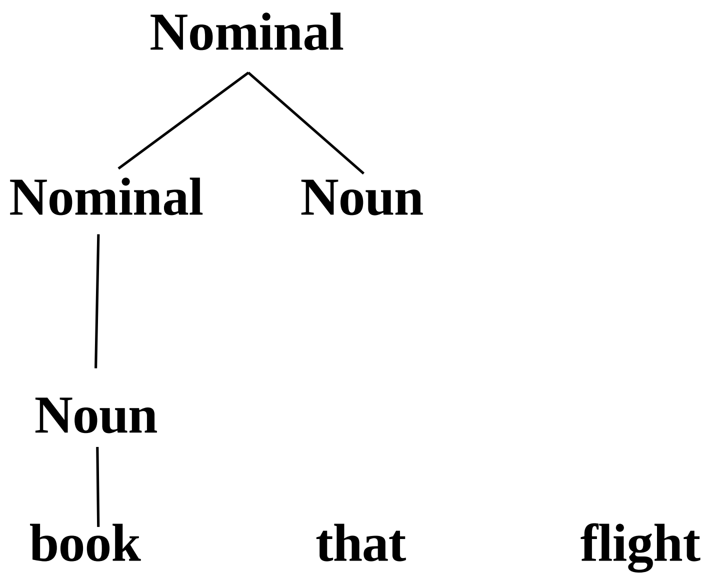
Bottom Up Parsing



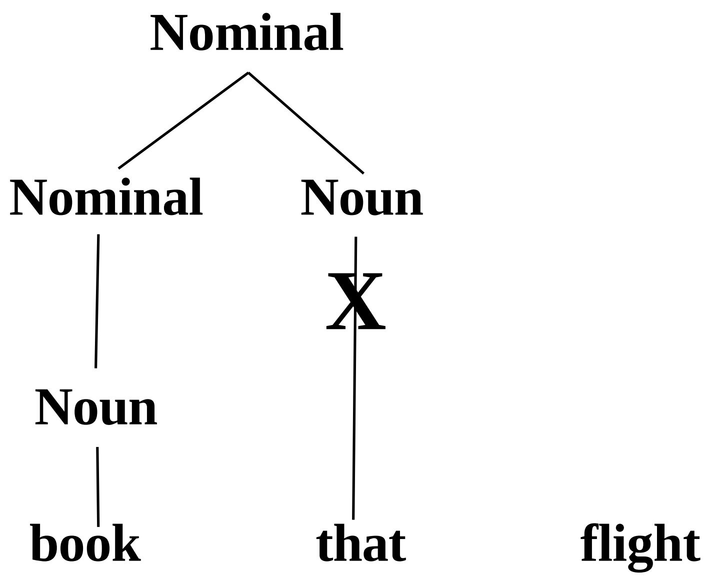
Bottom Up Parsing



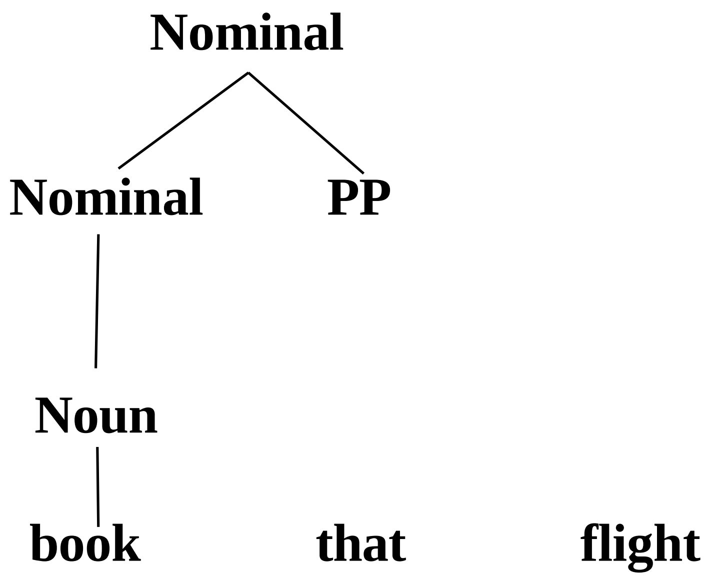
Bottom Up Parsing



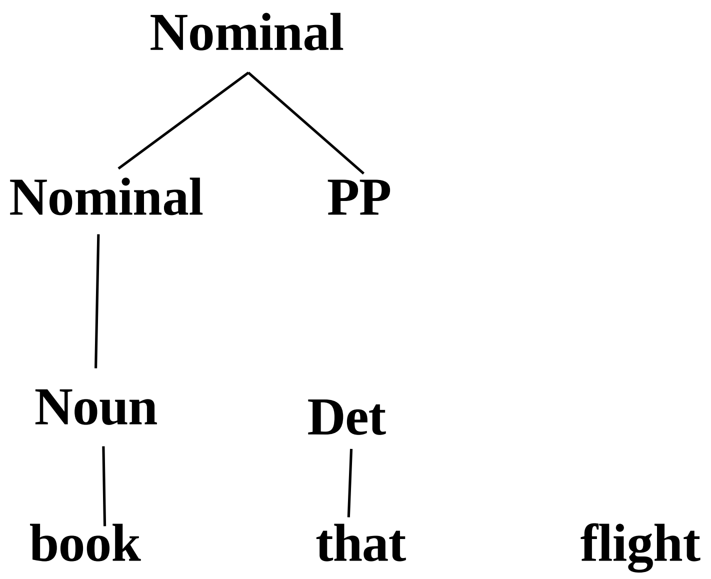
Bottom Up Parsing



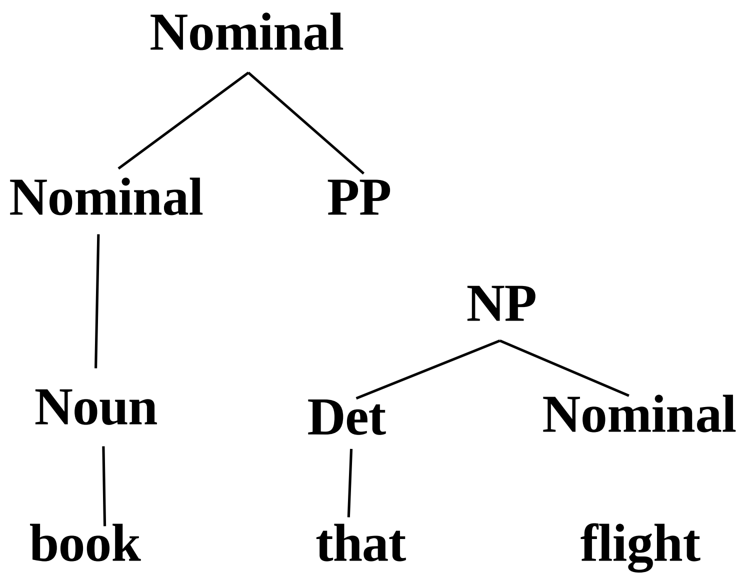
Bottom Up Parsing



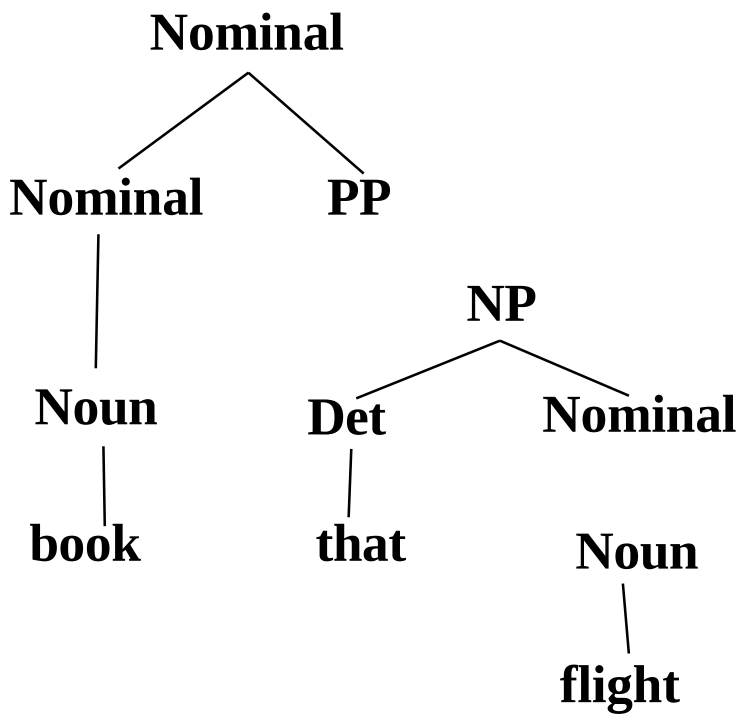
Bottom Up Parsing



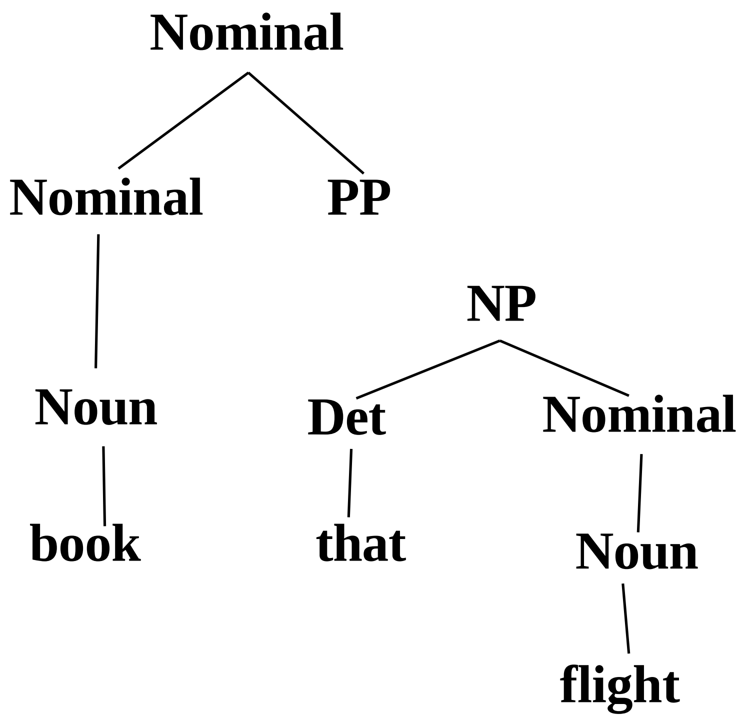
Bottom Up Parsing



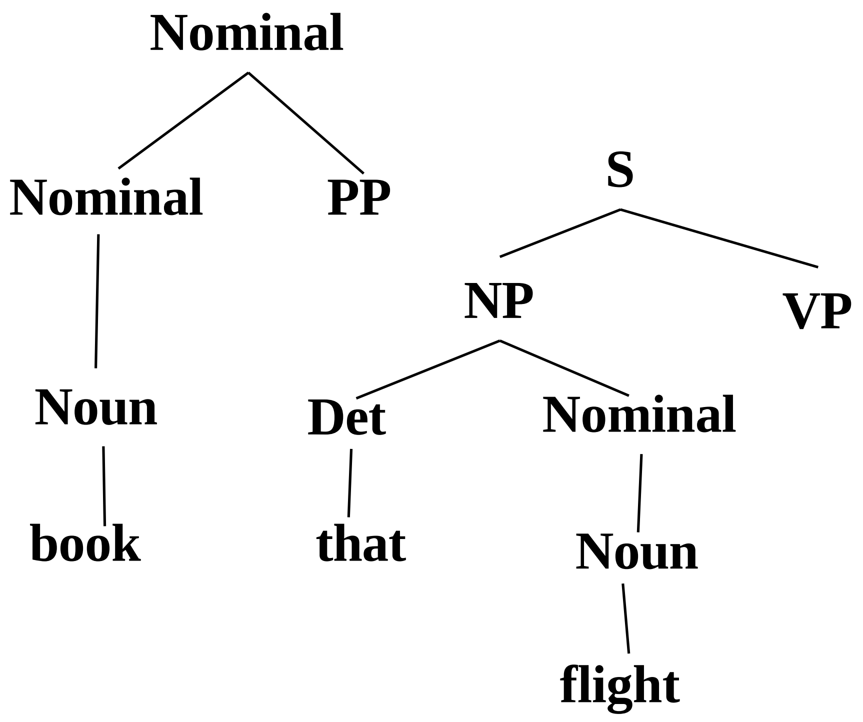
Bottom Up Parsing



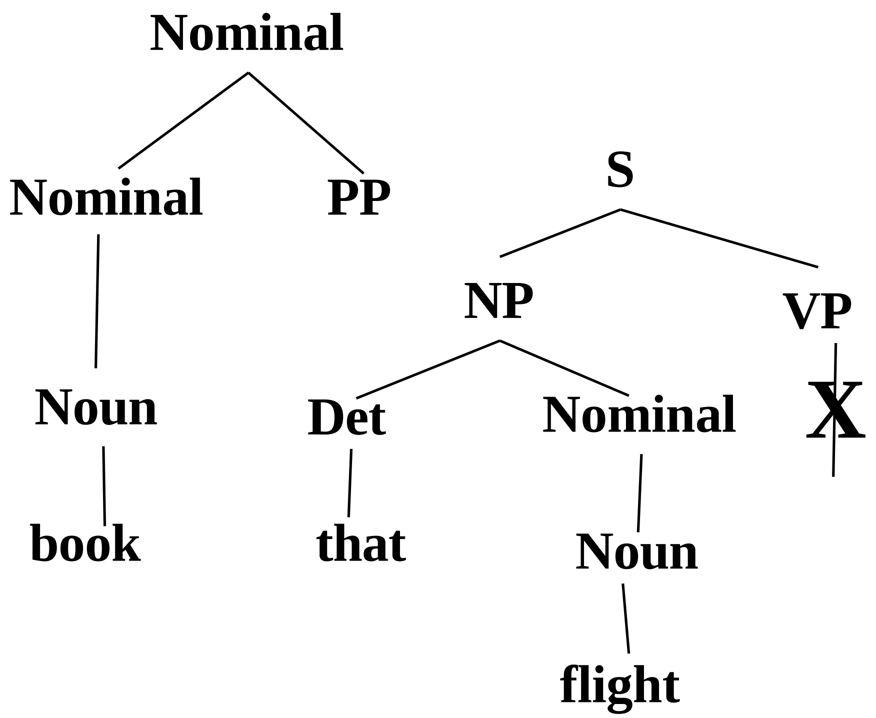
Bottom Up Parsing



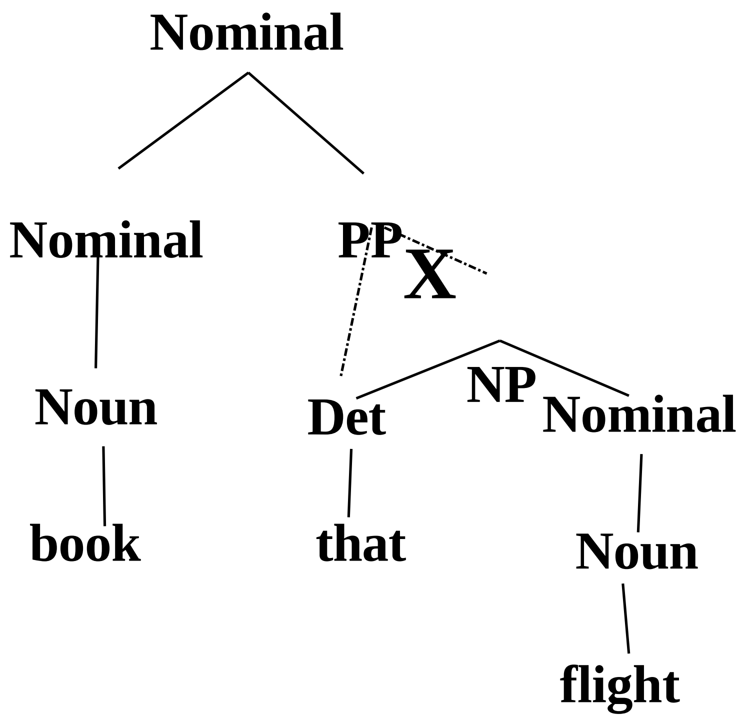
Bottom Up Parsing



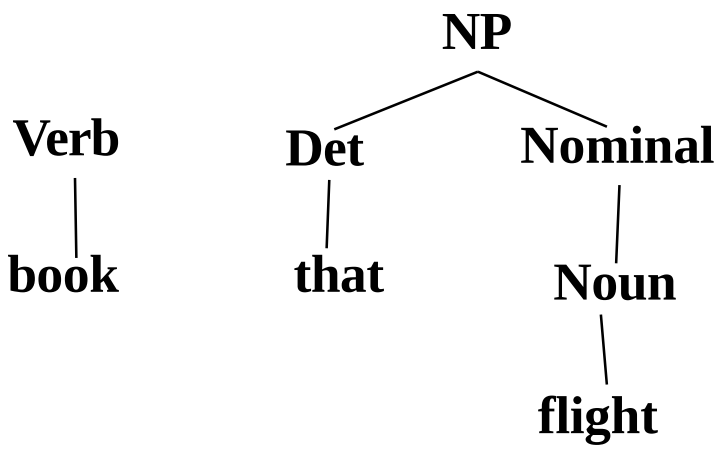
Bottom Up Parsing



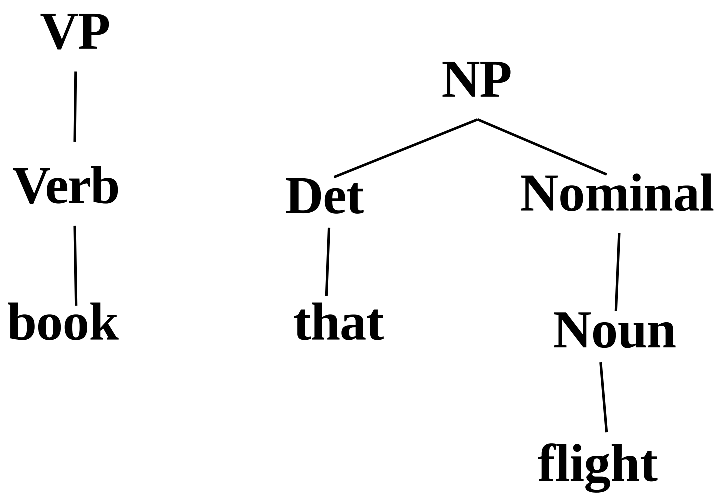
Bottom Up Parsing



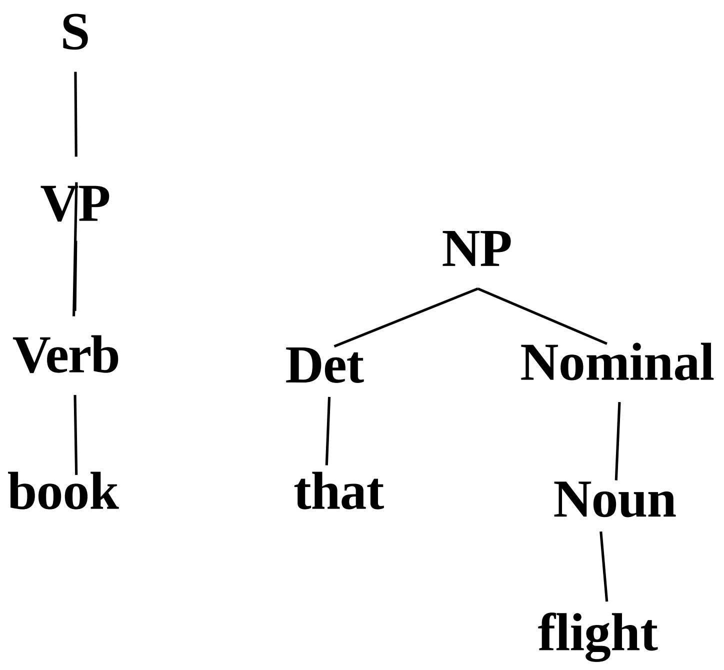
Bottom Up Parsing



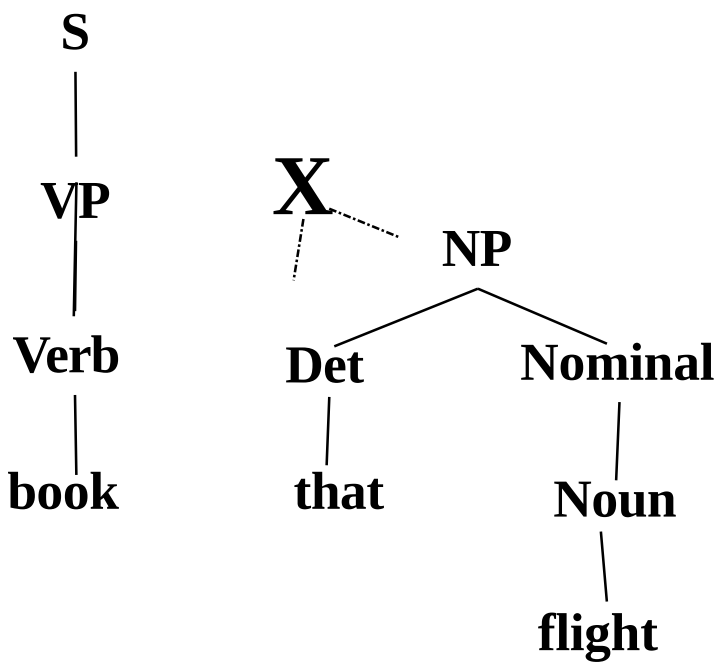
Bottom Up Parsing



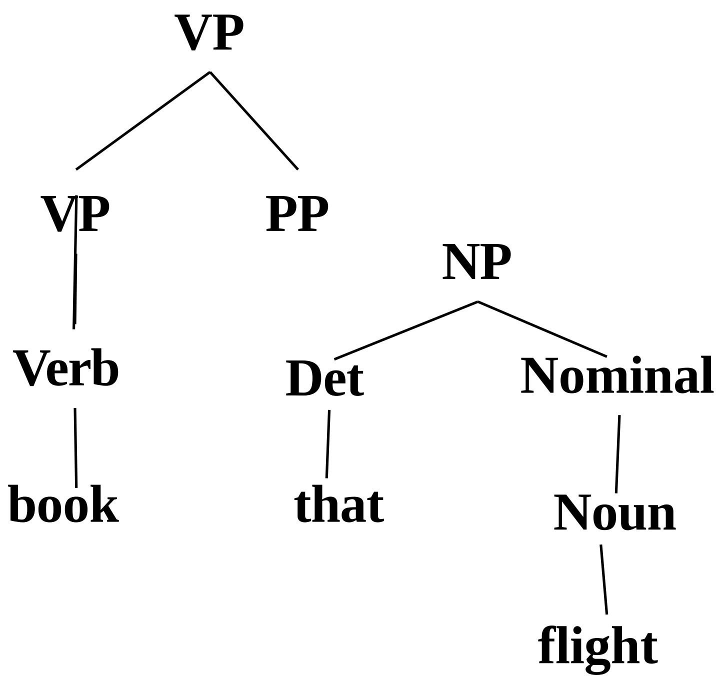
Bottom Up Parsing



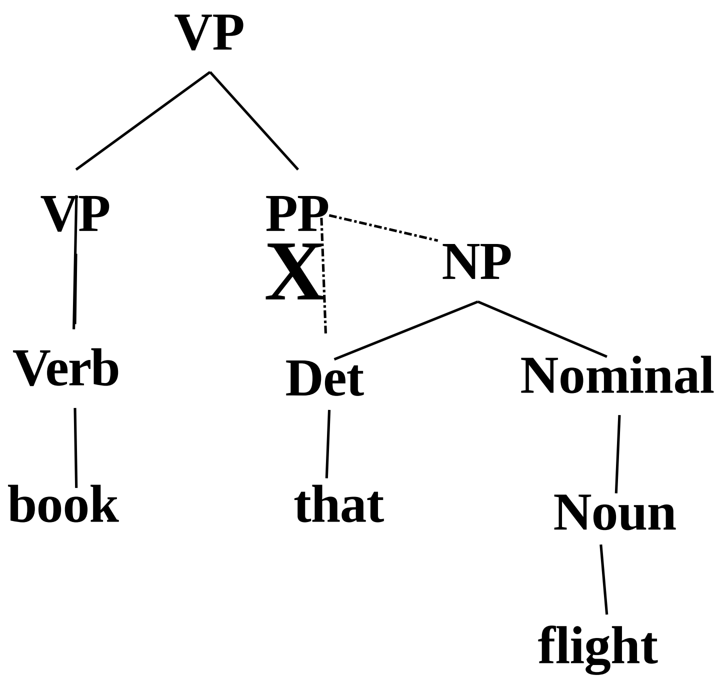
Bottom Up Parsing



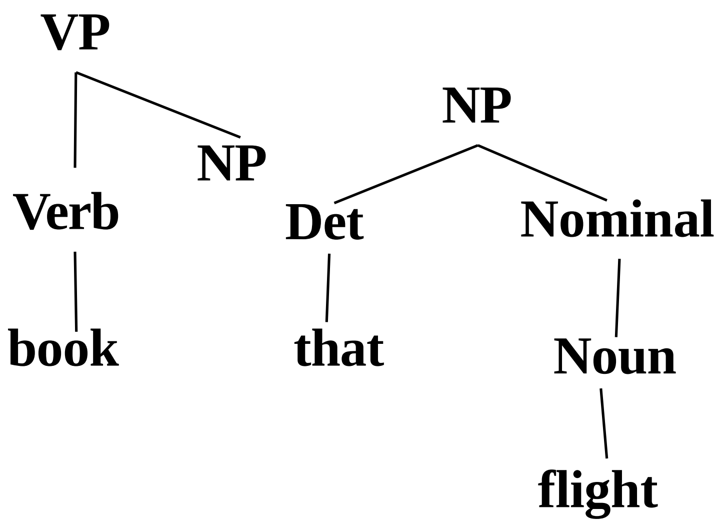
Bottom Up Parsing



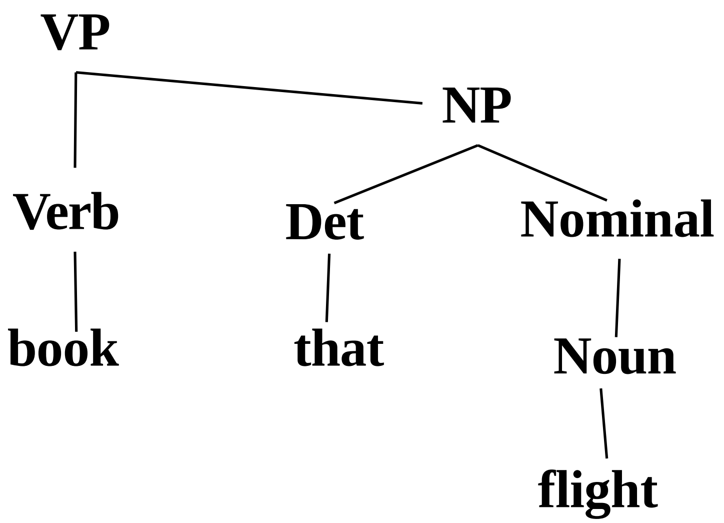
Bottom Up Parsing



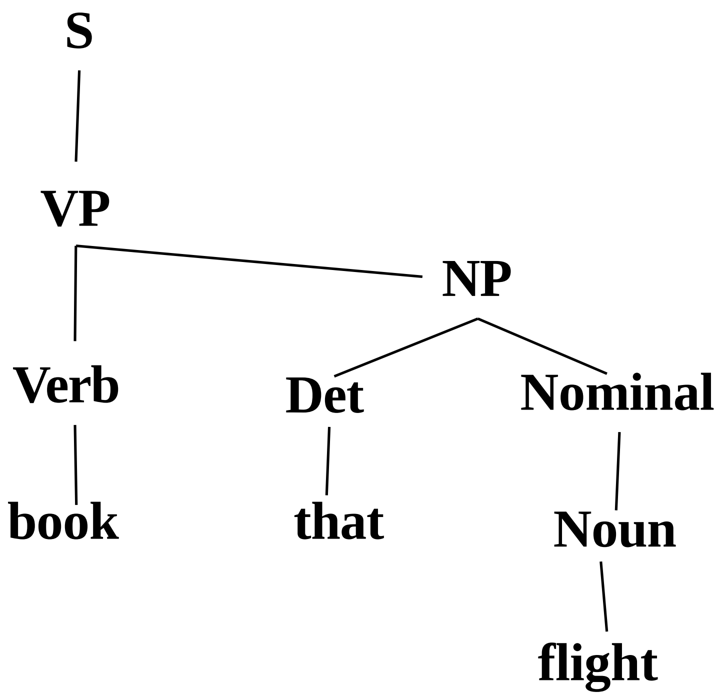
Bottom Up Parsing



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.

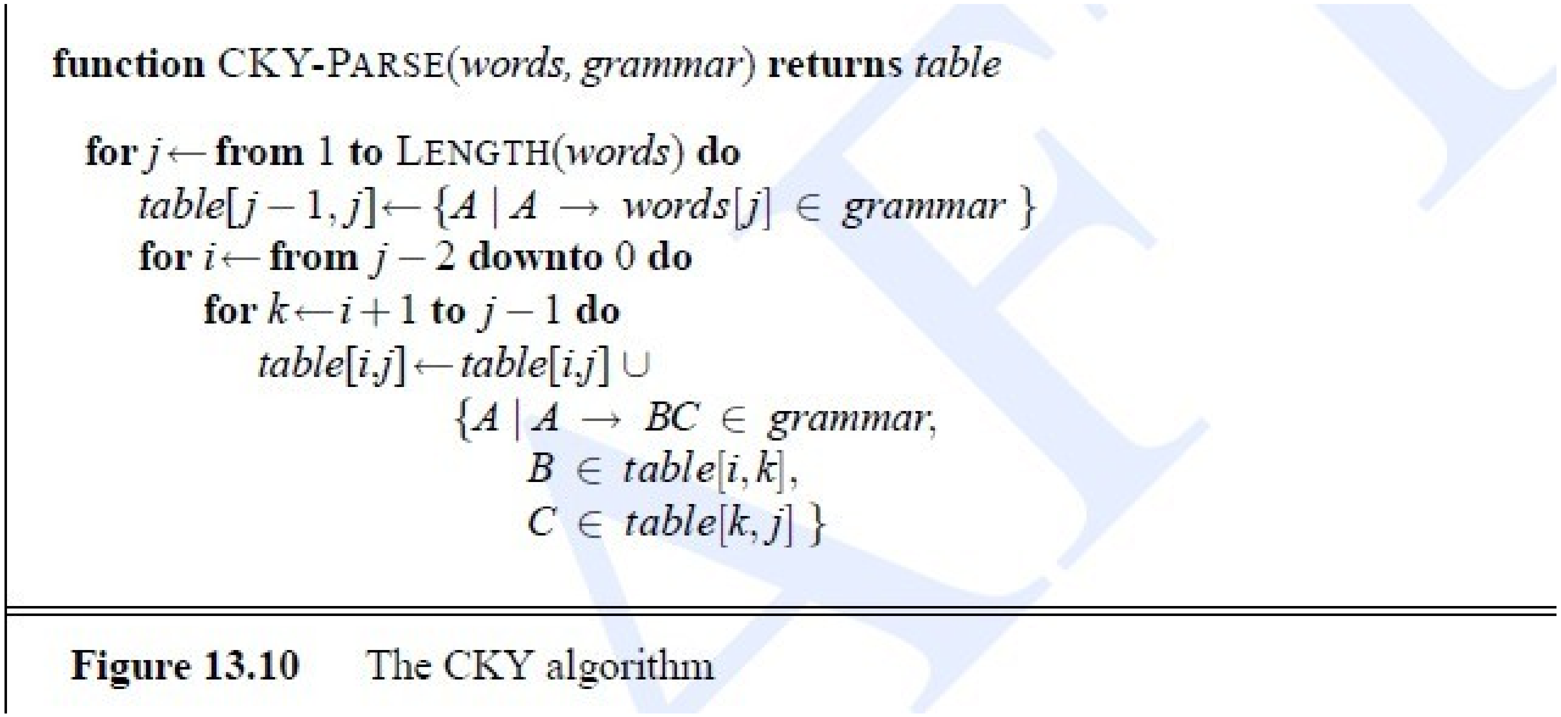
Dynamic Programming Parsing Methods

- **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 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).

CYK algorithm



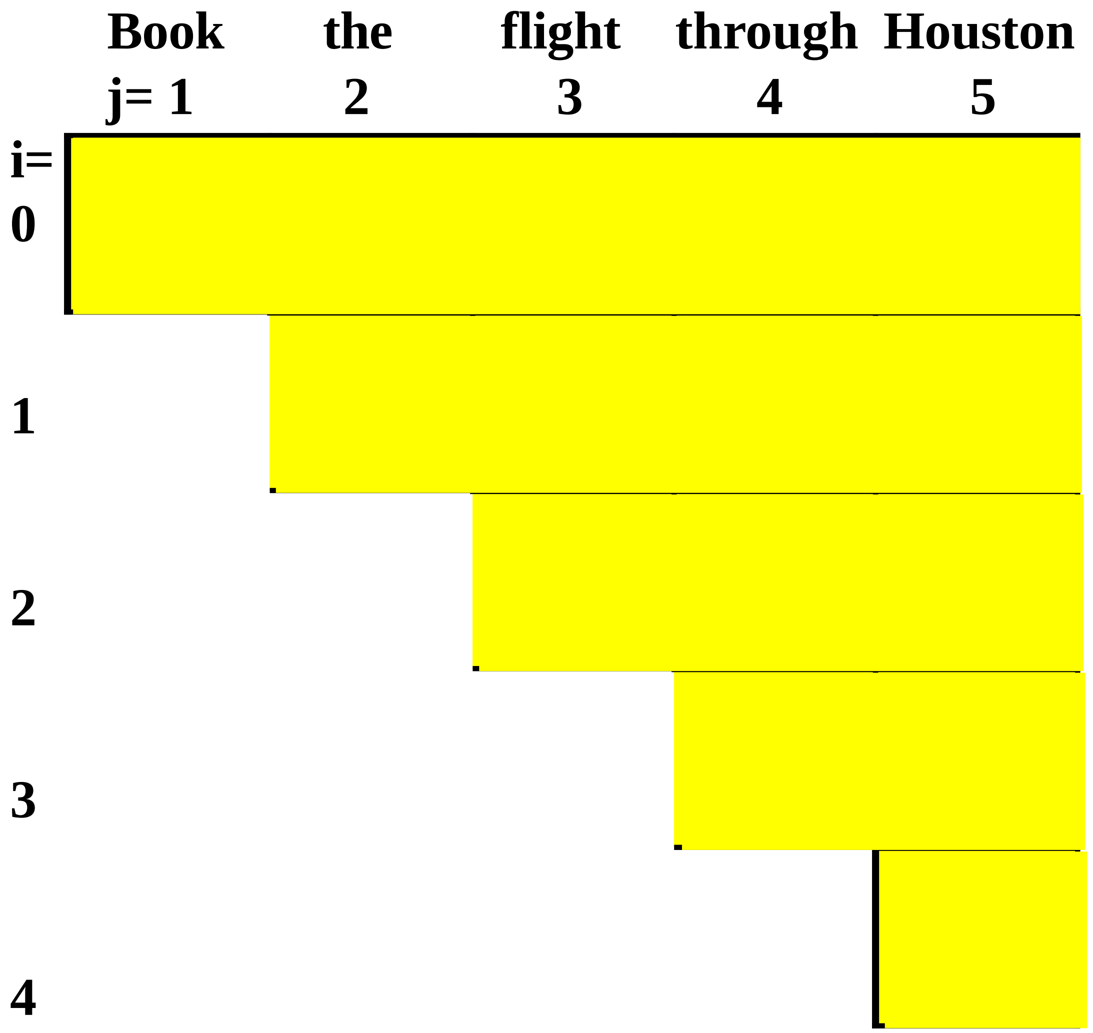
```
function CKY-PARSE(words, grammar) returns table

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

Figure 13.10 The CKY algorithm

ATIS English Grammar Conversion

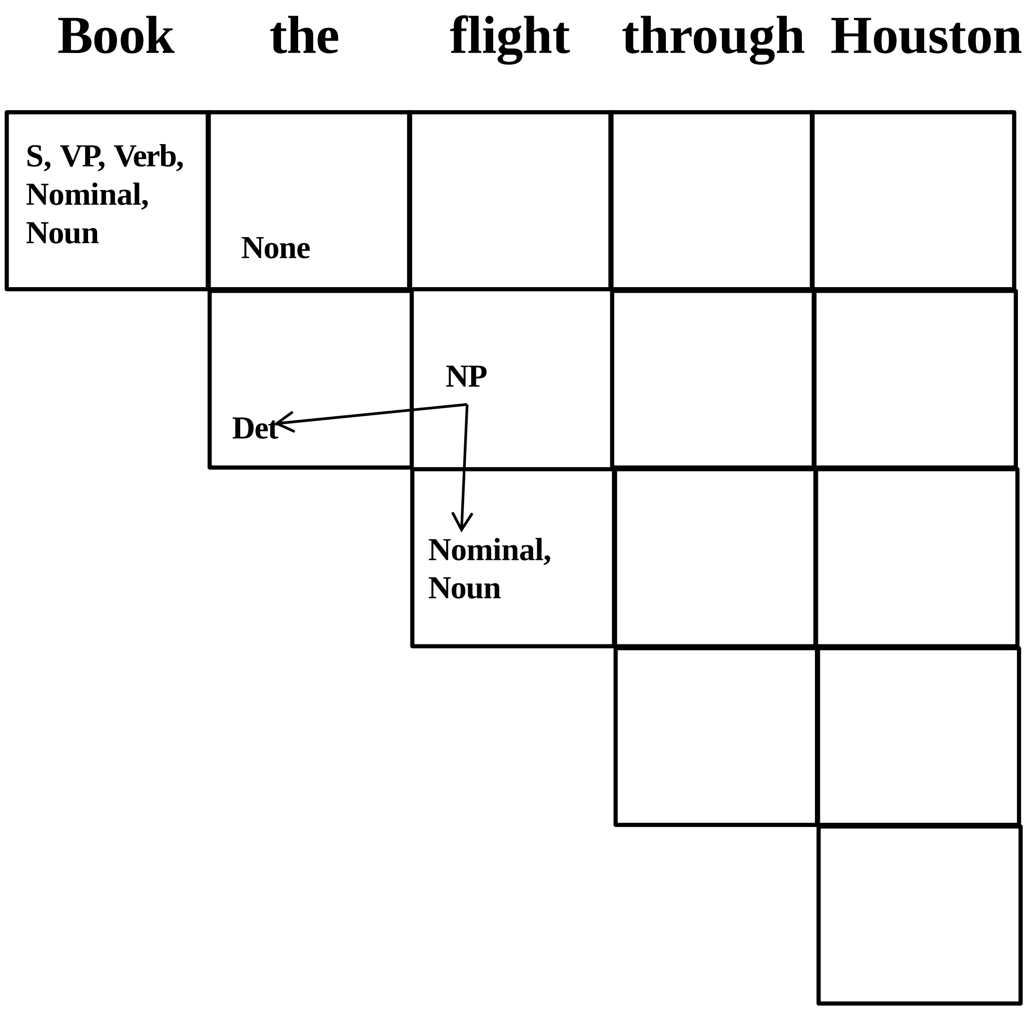
Original Grammar	**Chomsky Normal Form**
S → NP VP	**S → NP VP**
S → Aux NP VP	**S → X1 VP**
	X1 → Aux NP
S → VP	**S → book | include | prefer**
	S → Verb NP
	S → VP PP
NP → Pronoun	**NP → I | he | she | me**
NP → Proper-Noun	**NP → Houston | NWA**
NP → Det Nominal	**NP → Det Nominal**
Nominal → Noun	**Nominal → book | flight | meal | money**
Nominal → Nominal Noun	**Nominal → Nominal Noun**
Nominal → Nominal PP	**Nominal → Nominal PP**
VP → Verb	**VP → book | include | prefer**
VP → Verb NP	**VP → Verb NP**
VP → VP PP	**VP → VP PP**
PP → Prep NP	**PP → Prep NP**

CKY Parser

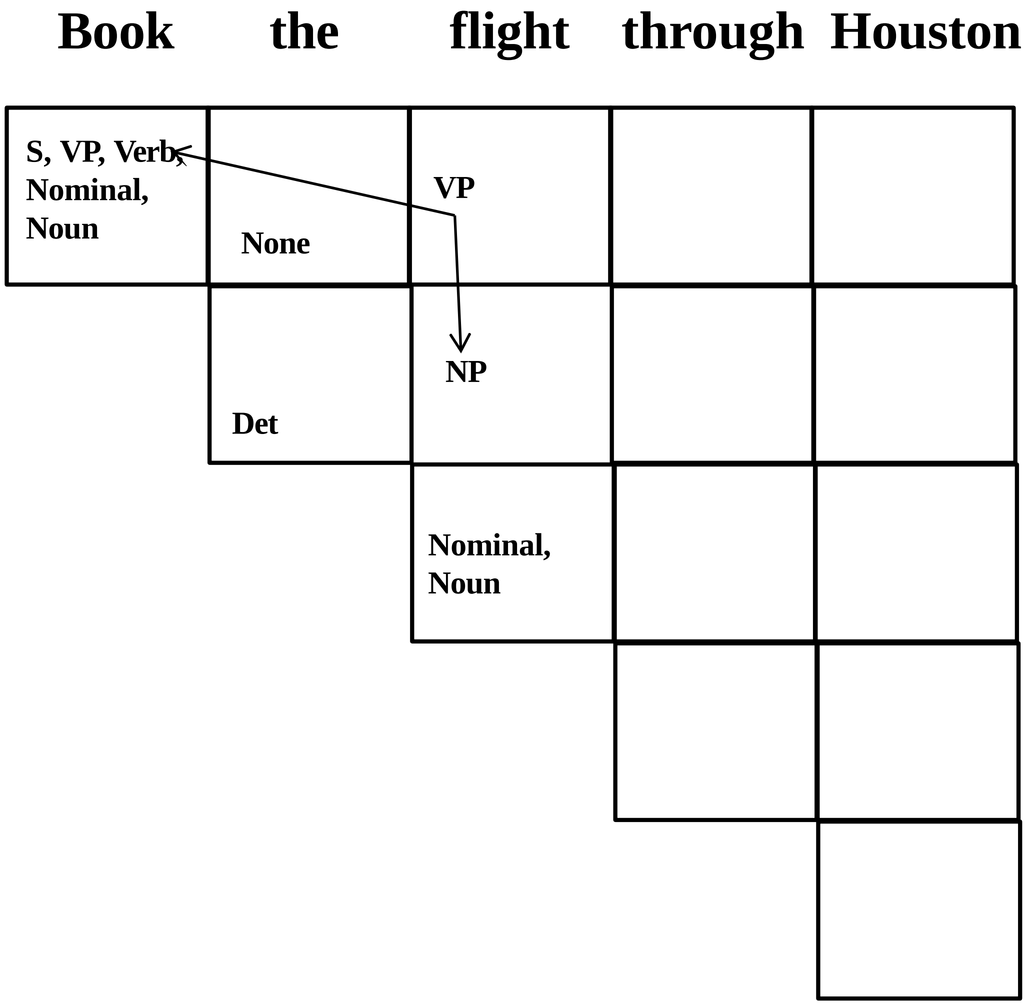


Cell[*i*,*j*] contains all constituents (non-terminals) covering words *i* +1 through *j*

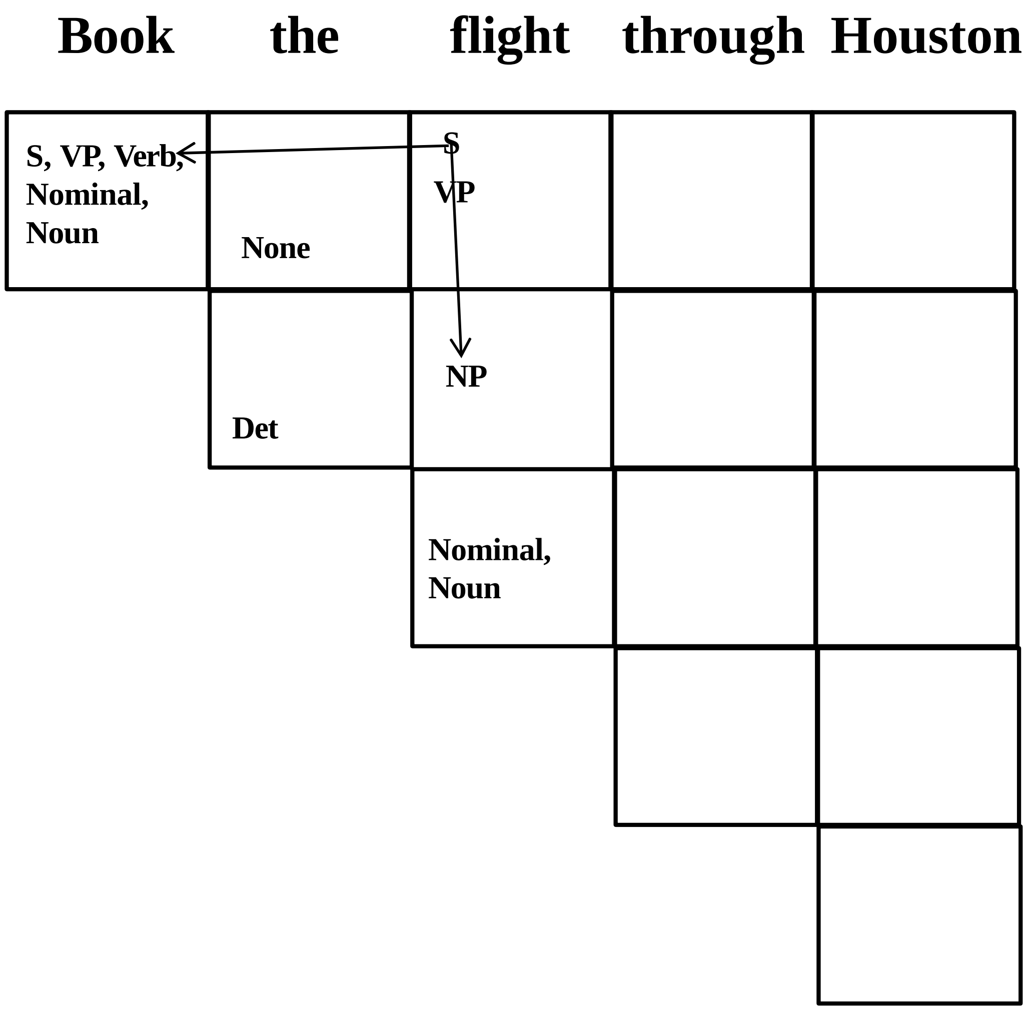
CKY Parser


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

CKY Parser


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

CKY Parser


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

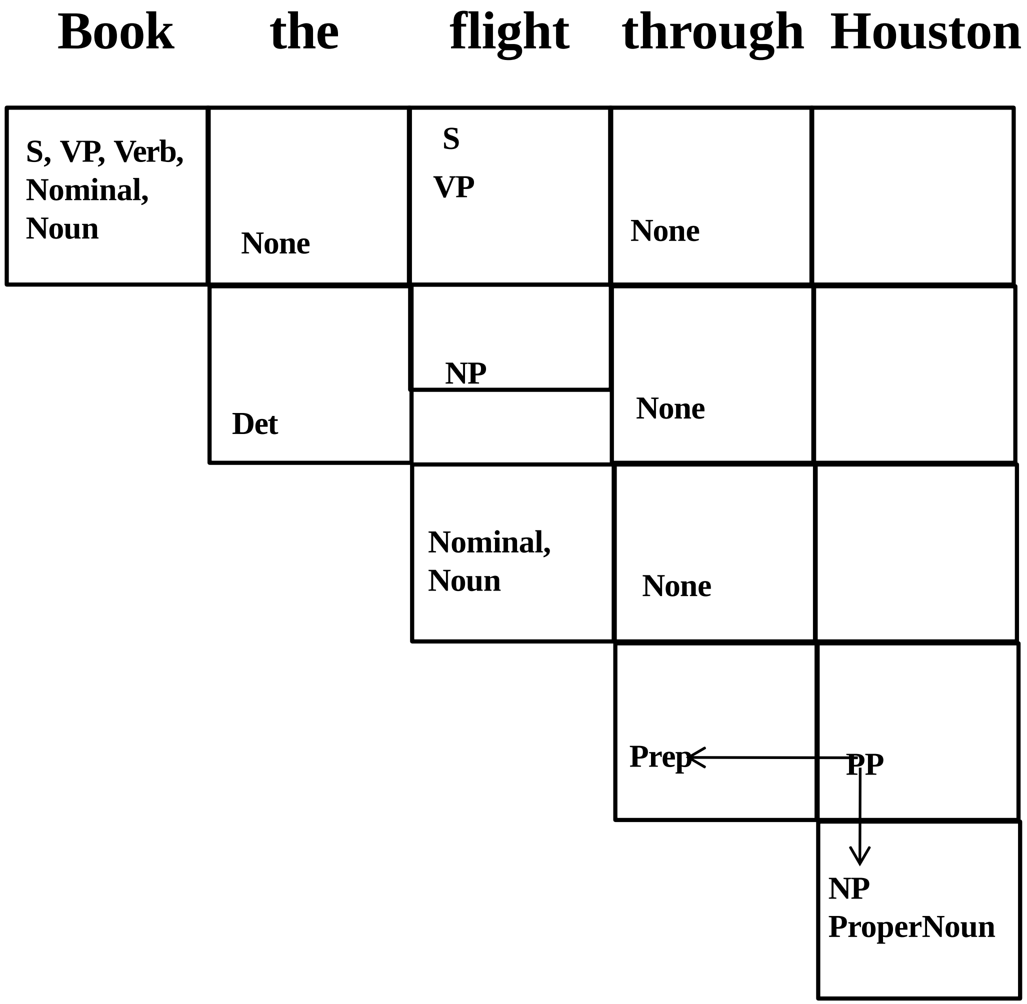
CKY Parser

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

CKY Parser

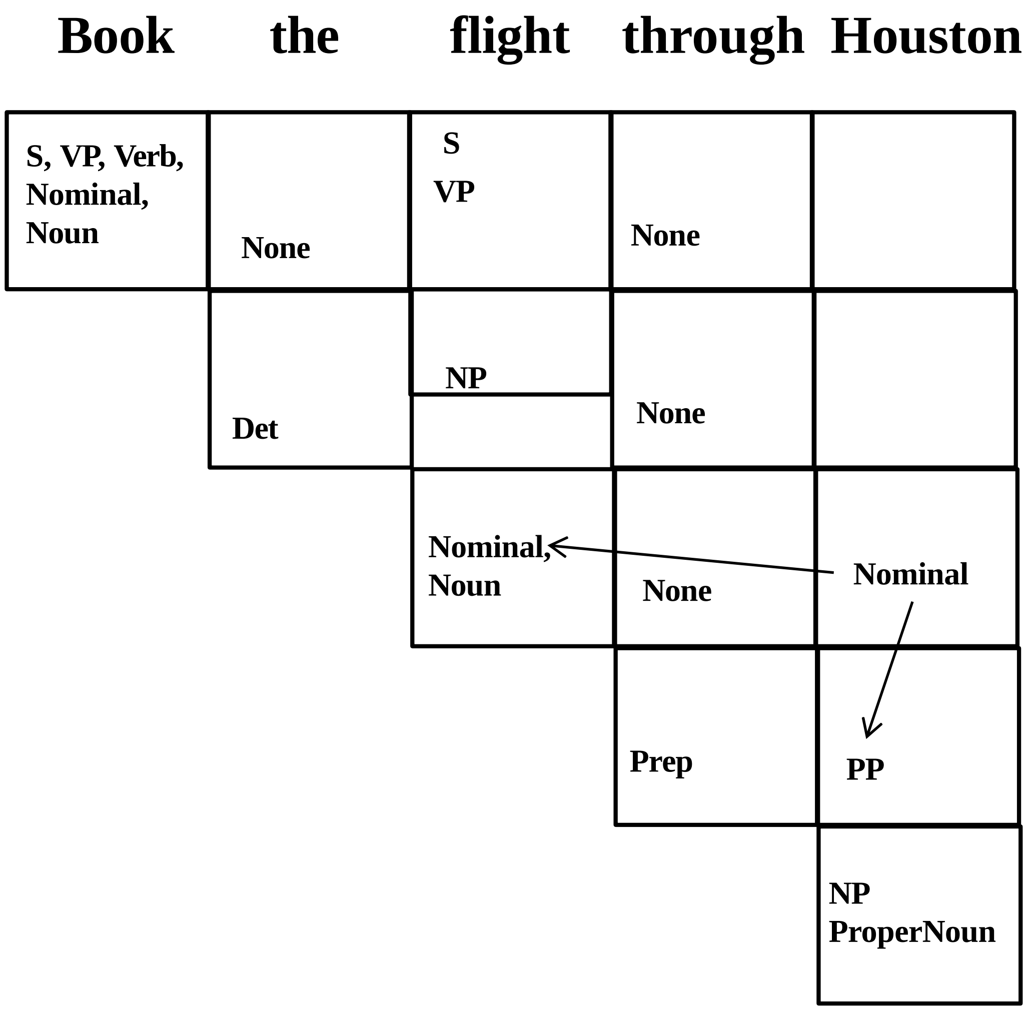
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	

CKY Parser



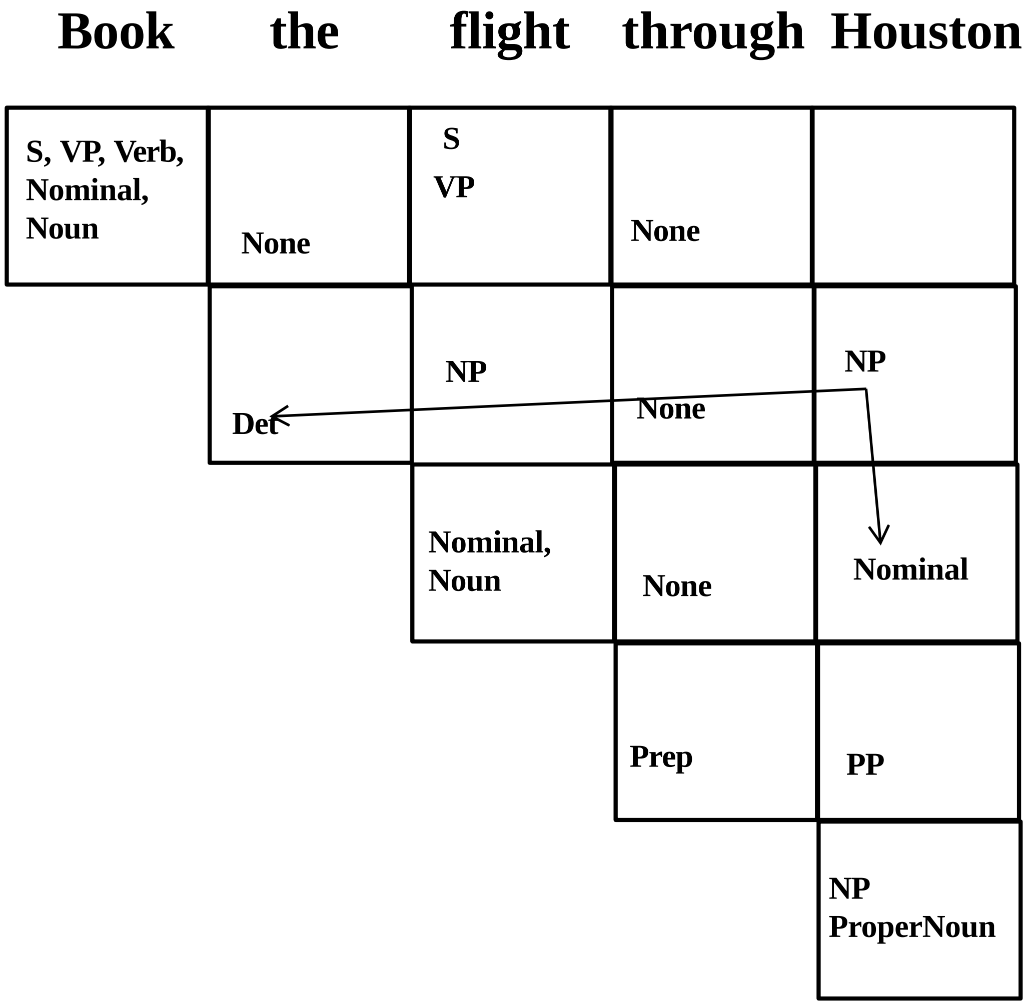
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	PP
				NP ProperNoun

CKY Parser



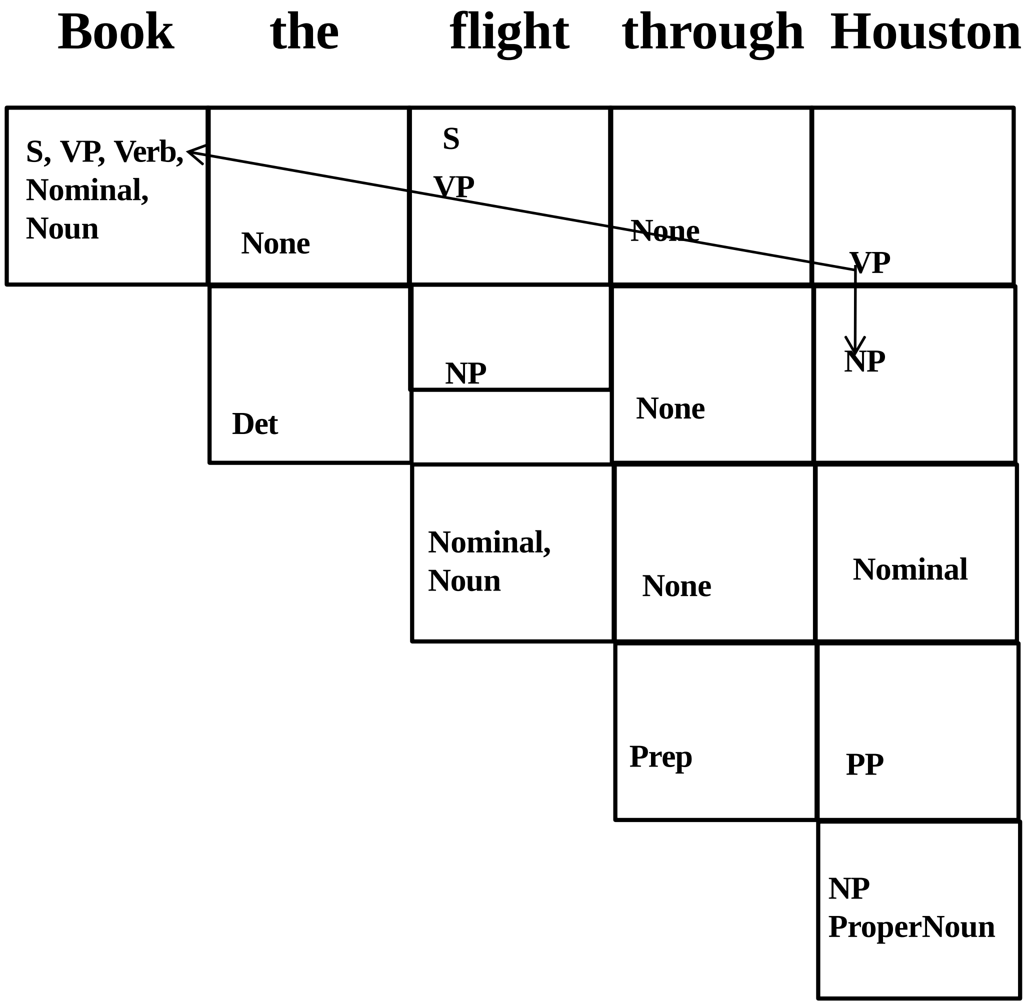
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser



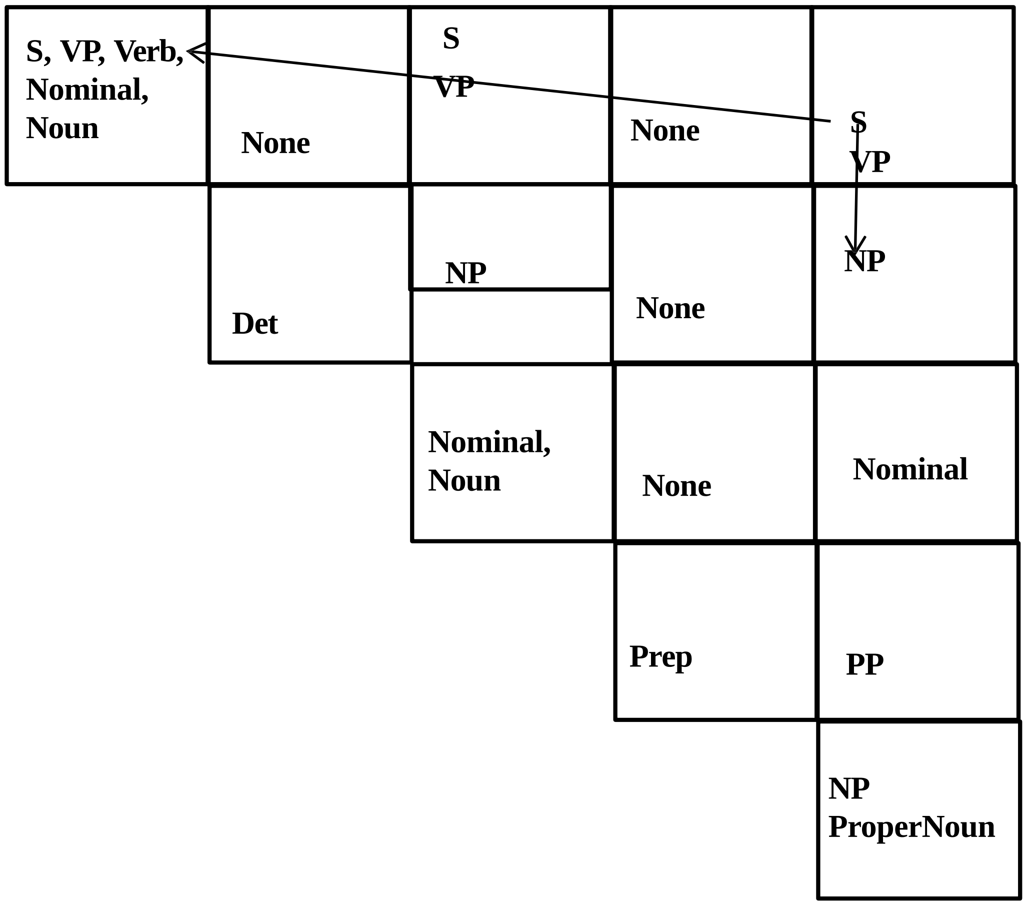
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser



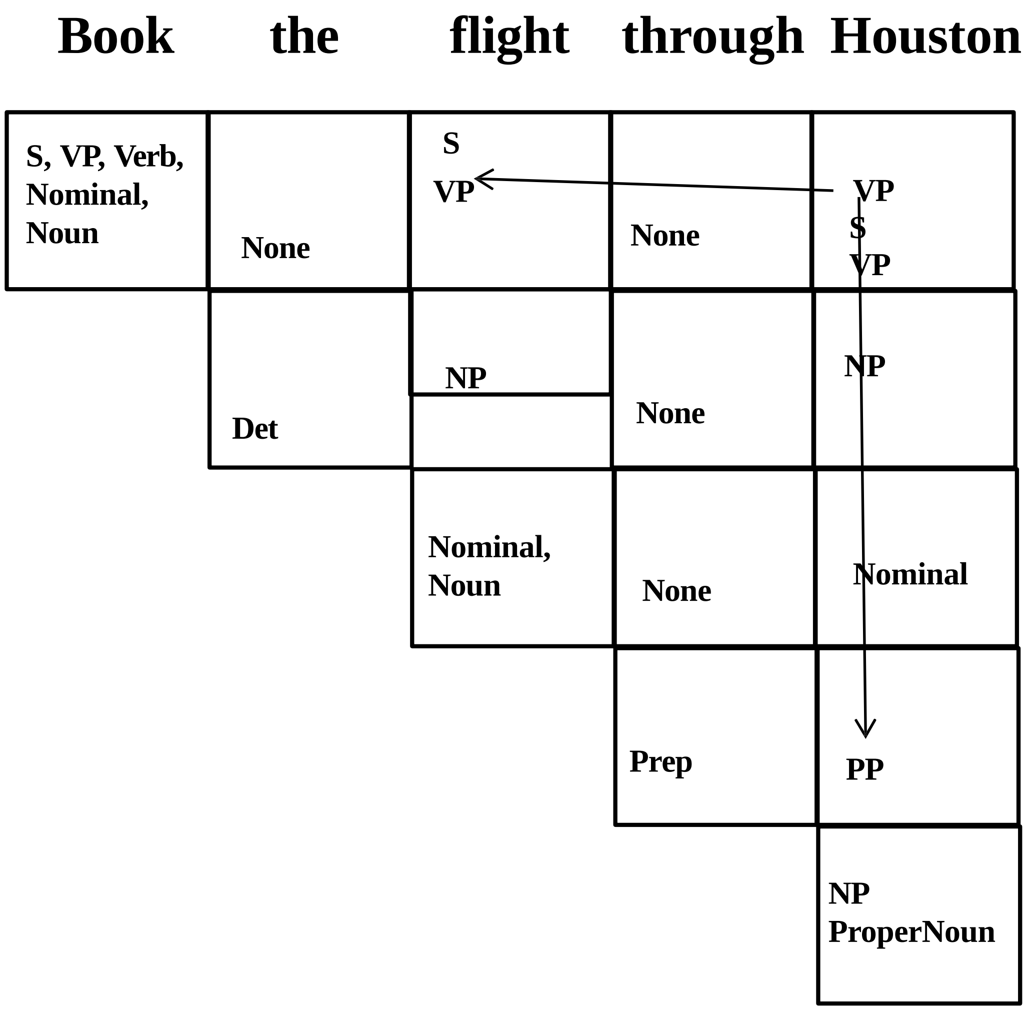
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser



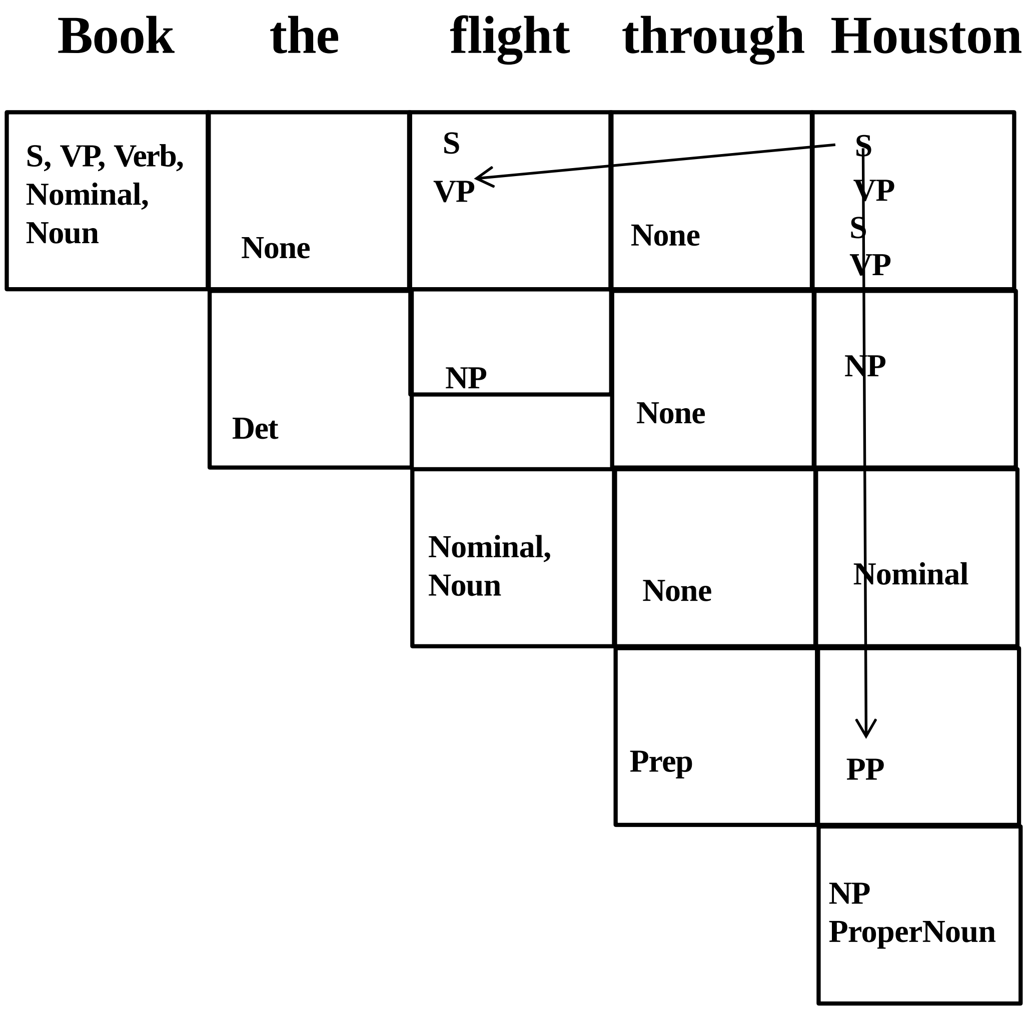
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

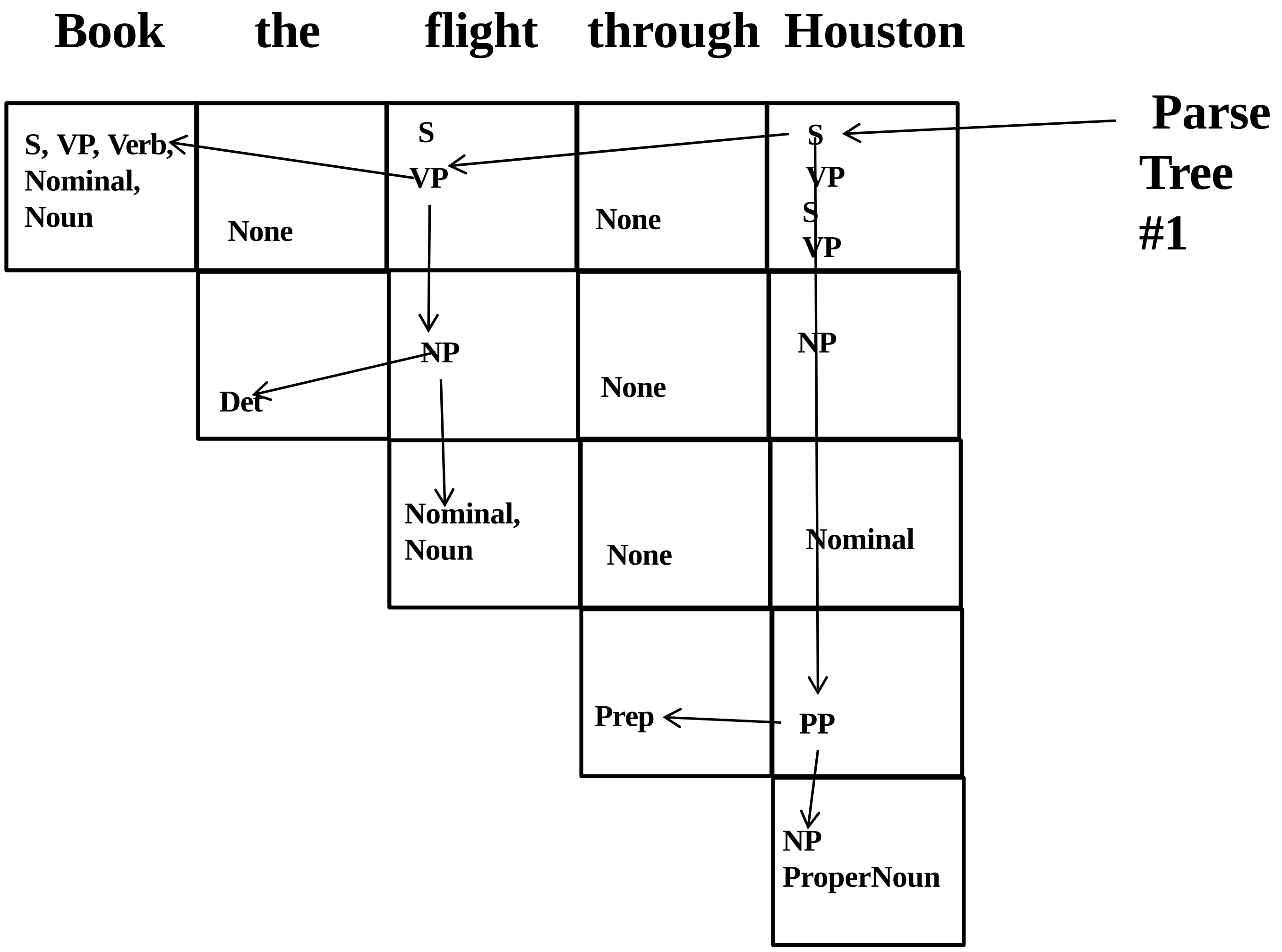


Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

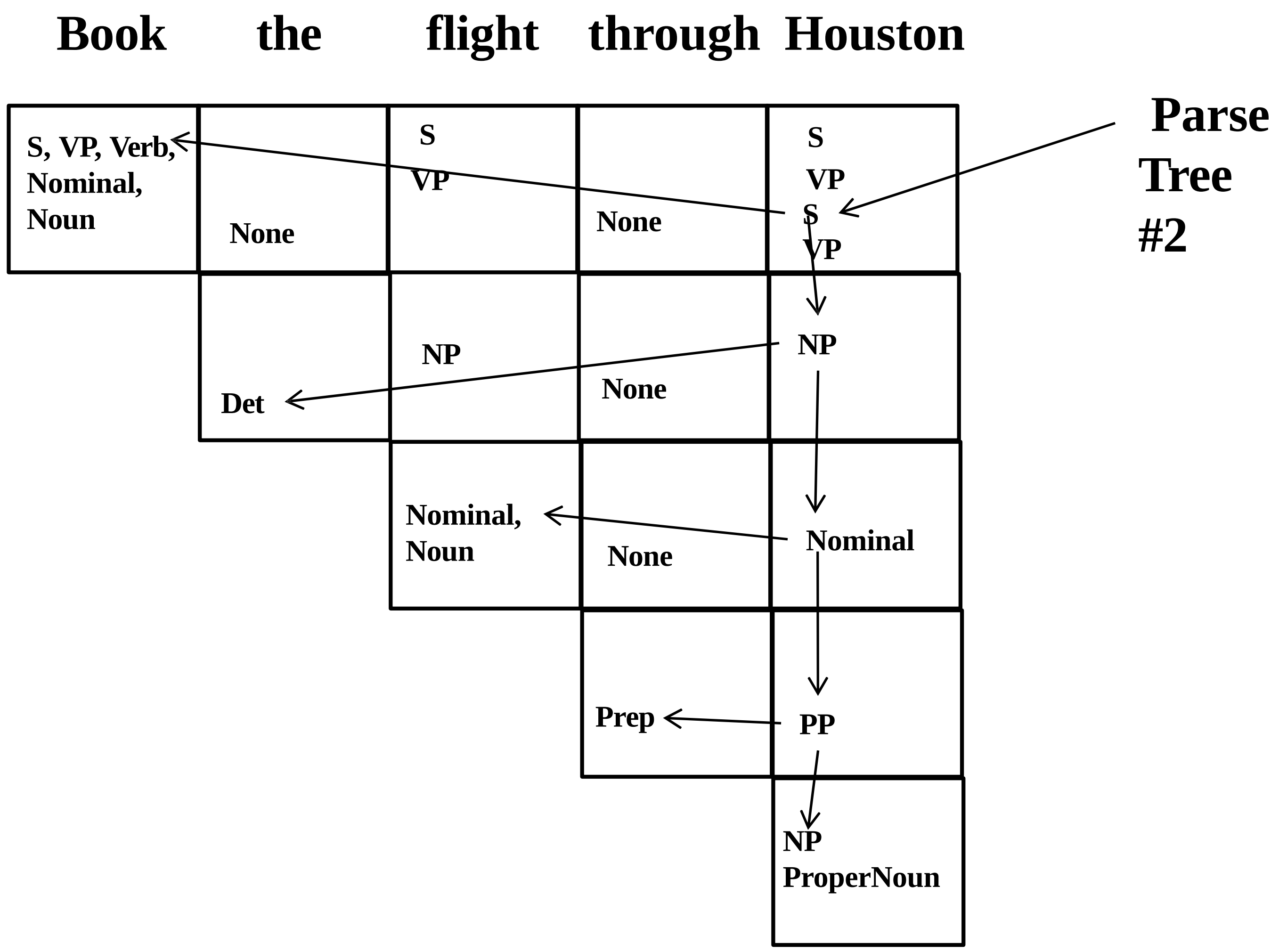

Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

CKY Parser



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

CKY Parser


Book
the
flight
through
Houston
Parse Tree #2
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
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 compute 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.

Syntactic Ambiguity

- Just produces all possible parse trees.
- Does not address the important issue of ambiguity resolution.

Issues with CFGs

- Addressing some grammatical constraints requires complex CFGs that do no compactly encode the given regularities.
- Some aspects of natural language syntax may not be captured at all by CFGs and require context-sensitivity (productions with more than one symbol on the LHS).

Agreement

- Subjects must agree with their verbs on person and number.
  - I am cold. You are cold. He is cold.
  - * I are cold * You is cold. *He am cold.
- Requires separate productions for each combination.
  - S → NP1stPersonSing VP1stPersonSing
  - S → NP2ndPersonSing VP2ndPersonSing
  - NP1stPersonSing → …
  - VP1stPersonSing → …
  - NP2ndPersonSing → …

Other Agreement Issues

- Pronouns have case (e.g. nominative, accusative) that must agree with their syntactic position.
  - I gave him the book. * I gave he the book.
  - He gave me the book. * Him gave me the book.
  - Los Angeles * Las Angeles
  - Las Vegas * Los Vegas
- Many languages have gender agreement.

Subcategorization

- Specific verbs take some types of arguments but not others.
  - Transitive verb: "found" requires a direct object
    - John found the ring. * John found.
  - Intransitive verb: "disappeared" cannot take one
    - John disappeared. * John disappeared the ring.
  - "gave" takes both a direct and indirect object
    - John gave Mary the ring. * John gave Mary. * John gave the ring.
  - "want" takes an NP, or non-finite VP or S
    - John wants a car. John wants to buy a car. John wants Mary to take the ring. * John wants.
- **Subcategorization frames** specify the range of argument types that a given verb can take.

Conclusions

- Syntax parse trees specify the syntactic structure of a sentence that helps determine its meaning.
  - John ate the spaghetti with meatballs with chopsticks.
  - How did John eat the spaghetti?
    What did John eat?
- CFGs can be used to define the grammar of a natural language.
- Dynamic programming algorithms allow computing a single parse tree in cubic time or all parse trees in exponential time.