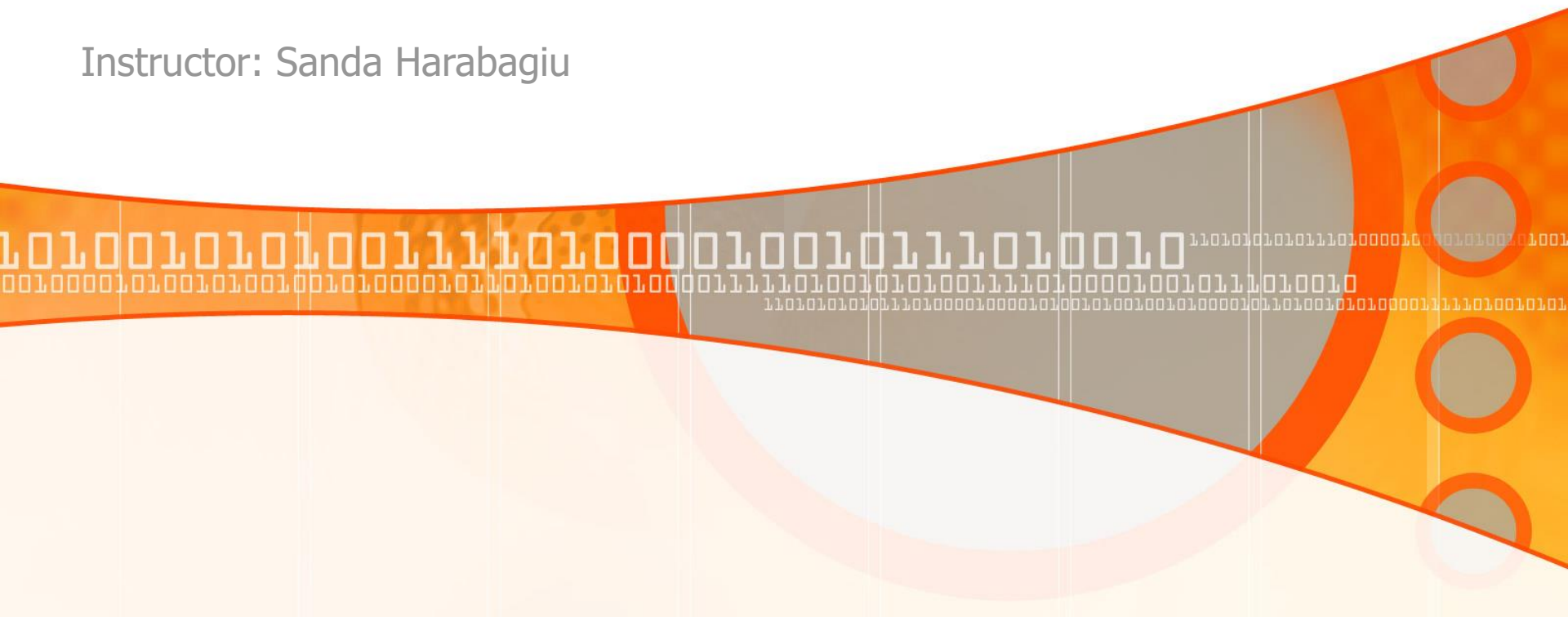
Natural Language Processing
CS 6320

Lecture 10

Syntactic Parsing

Instructor: Sanda Harabagiu



Introduction into Syntactic Parsing

- *Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. We consider the structures assigned by context-free grammars (CFGs).*
  - CFGs *are based on a purely* ***declarative formalism****, as they don't specify how the* parse tree *for a given sentence should be computed. We therefore need to specify algorithms that employ these grammars to efficiently produce correct parses (trees).*

- We discuss why parsing is difficult:
  - *Structural ambiguity!!!*
  - *Parsing Algorithm:*
    - Cocke-Kasami-Younger (CKY) algorithm
  - Partial parsing

Structural Ambiguity

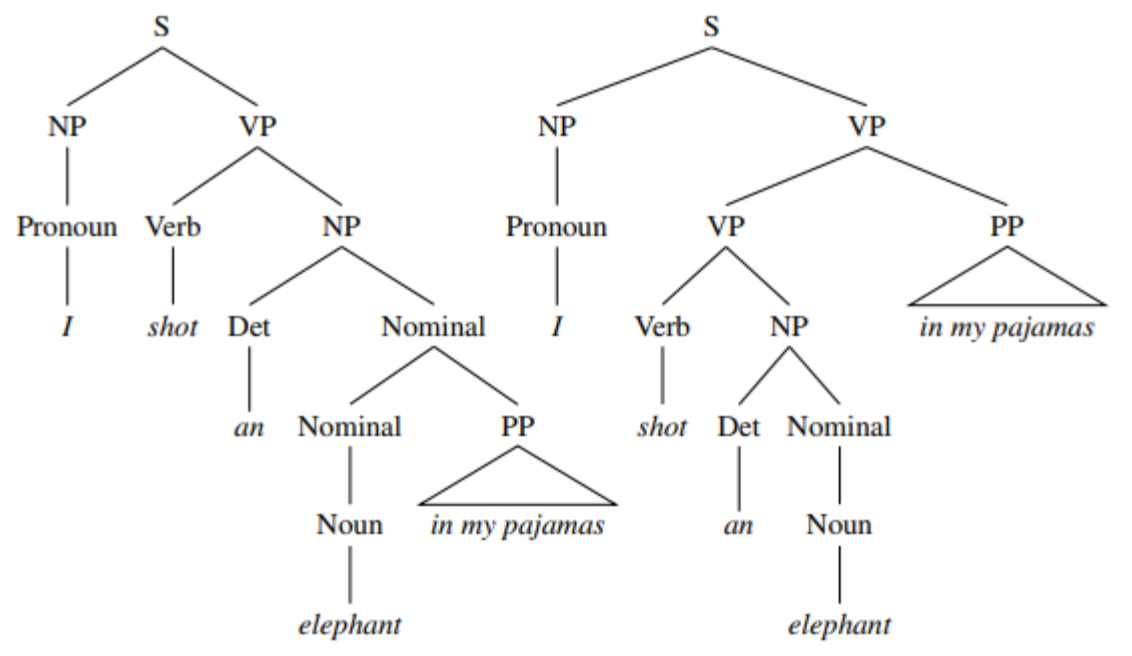
- **DEFINITION:** *Structural ambiguity occurs when the grammar can assign more than one parse to a sentence.*

Example: Miniature English grammar & lexicon

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

SENTENCE: I shot an elephant in my pajamas.

Two different parses



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

SENTENCE: I shot an elephant in my pajamas.

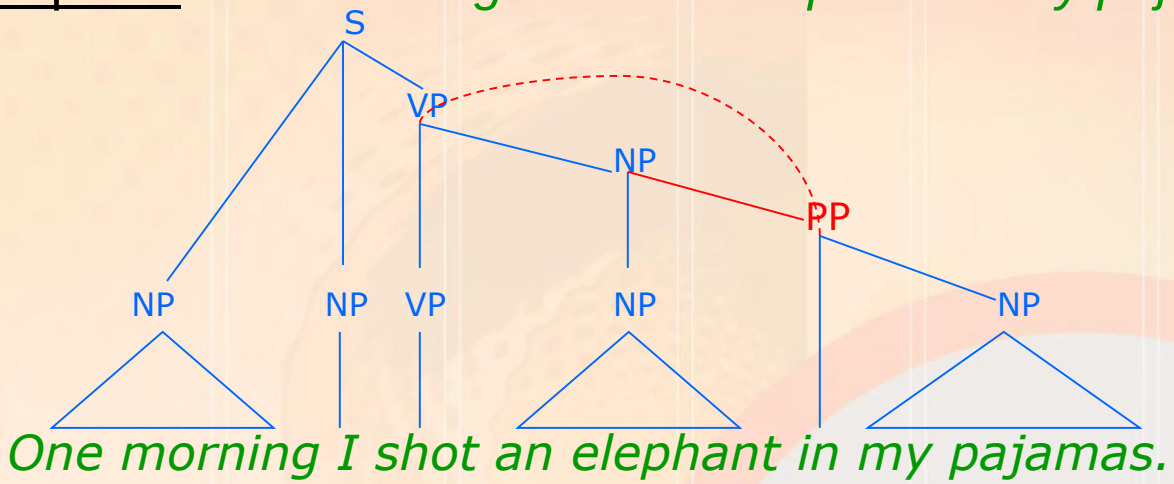
- *Groucho Marx's well-known line as Captain Spaulding in Animal Crackers is ambiguous because the phrase in my pajamas can be part of the NP headed by elephant or a part of the verb phrase headed by shot.*

Structural Ambiguity

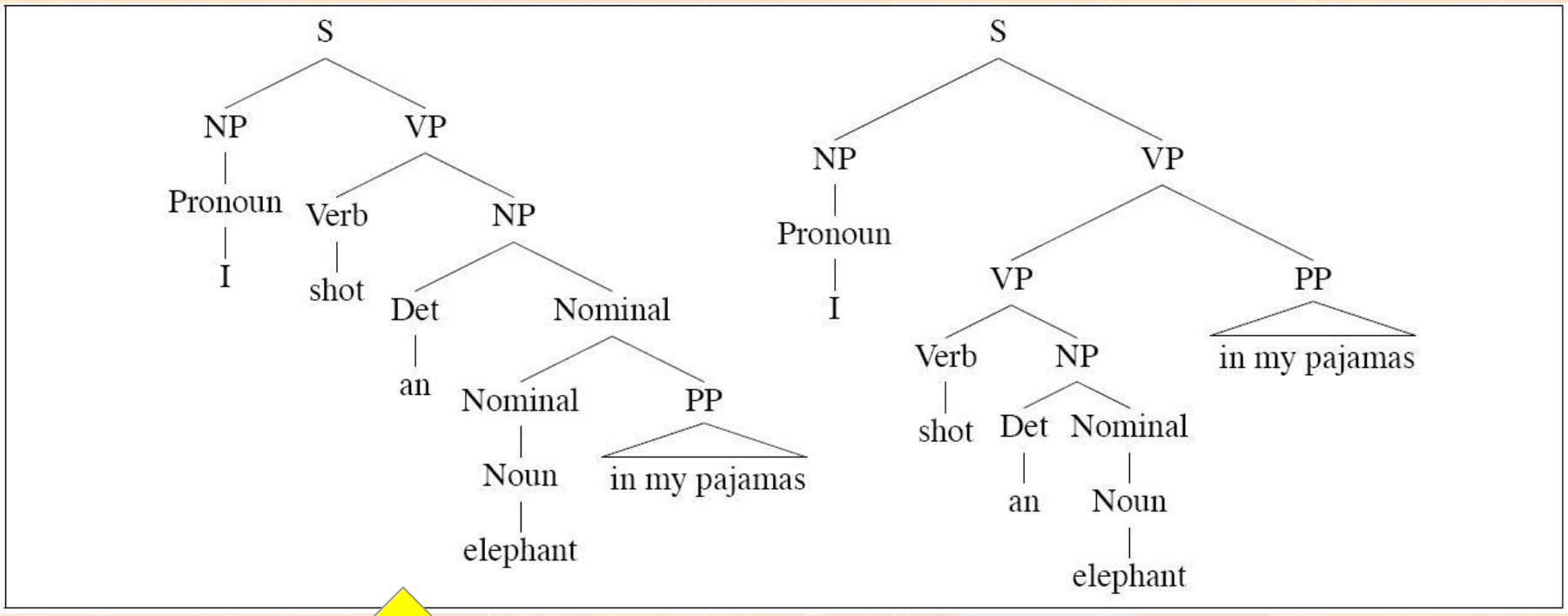
Structural ambiguity comes in many forms. Two particularly common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.

Definition: A sentence has an **attachment ambiguity** if a particular constituent can be attached to the parse tree at more than one place.

Example 1: *One morning I shot an elephant in my pajamas.*



Parse trees for an ambiguous sentence



Humorous reading!!!

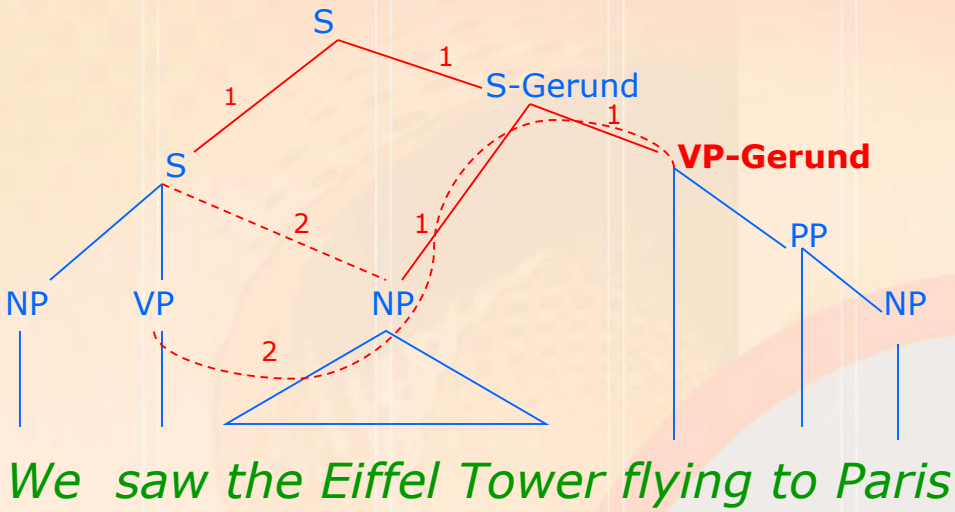
Captain Spalding did the shooting in his pajamas

Attachment Ambiguity

Various kinds of *adverbial phrases* are also subject to this kind of ambiguity.

Example 2: *We saw the Eiffel Tower flying to Paris.*

The gerundive-VP *flying to Paris* can be part of a gerundive sentence whose subject is *the Eiffel Tower* or it can be an adjunct modifying the VP headed by *saw.*



Coordination Ambiguity

In **coordination ambiguity** there are different sets of phrases that can be conjoined by a conjunction like *and*.

Example: the phrase *old men and women* can be bracketed as:

[old [men and women]], referring to *old men* and *old women*, or as

[old men] and [women], in which case it is only the men who are old.

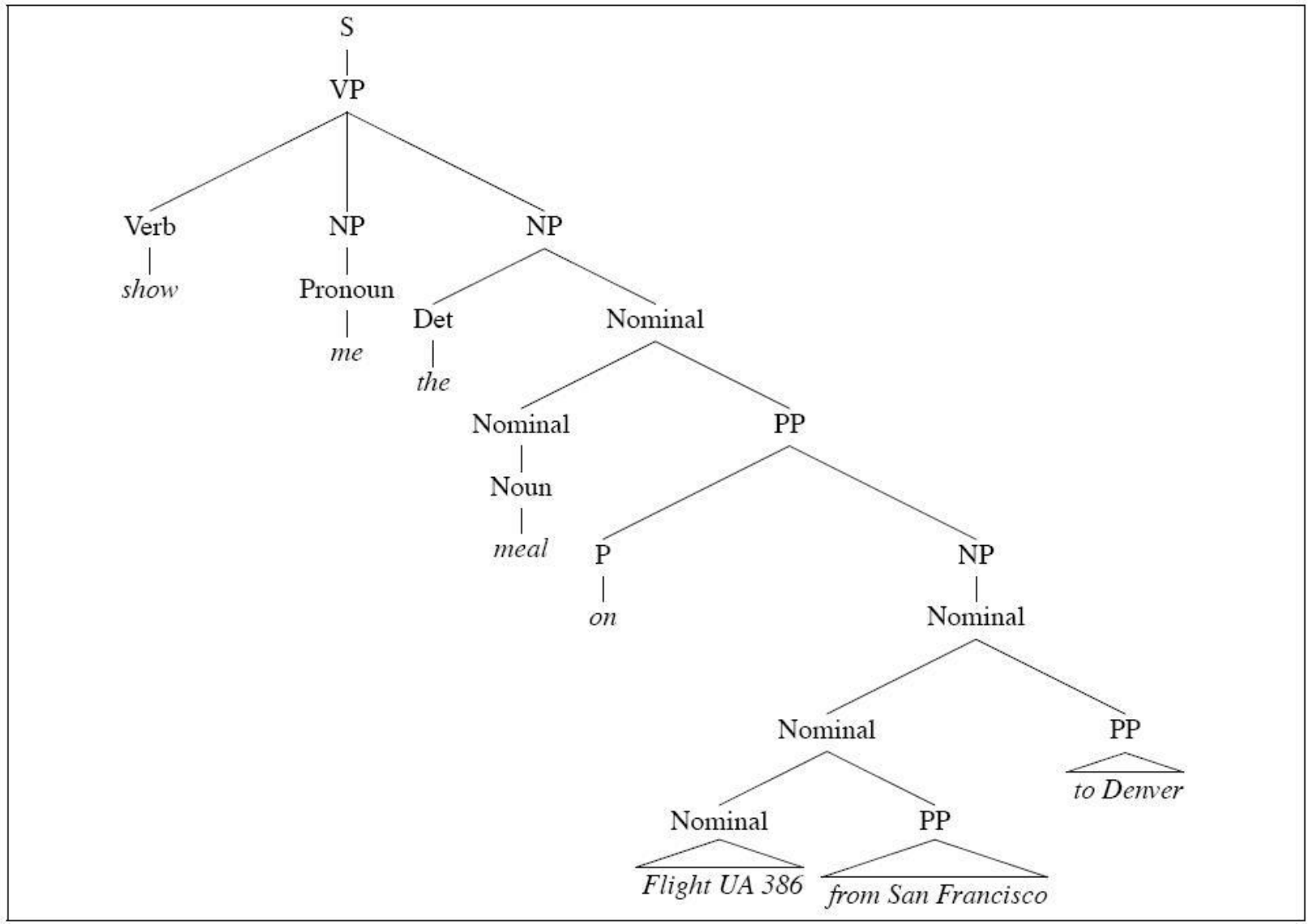
Syntactic ambiguity: number of parses grows exponentially!

Example: *Show me the meal on Flight UA 386 from San Francisco to Denver.*

VP → VP PP and Nominal → Nominal PP

Reasonable Parse

Show me the meal on Flight UA 386 from San Francisco to



CKY Parsing

Dynamic Programming Parsing Methods

The CKY algorithm is a classic example of the dynamic programming paradigm. It is a bottom-up parser.

- Requirements: the grammars used with CKY parsing must be in **Chomsky normal form** (CNF).
- CNF are restricted to rules of the form

  $A \rightarrow B\ C$, or $A \rightarrow w$.

The right-hand side of each rule must expand to either two non-terminals or to a single terminal.

- This single restriction gives rise to an extremely simple and elegant *table-based* parsing method.

Conversion to CNF 1/2

- How can we convert a generic CFG into one represented in CNF?

- There are three situations we need to address in any generic grammar:

1. rules that mix terminals with non-terminals on the right-hand side,
2. rules that have a single non-terminal on the right, and
3. rules where the right-hand side's length is greater than two.

Solutions:

A. The remediation for rules that mix terminals and non-terminals is to simply introduce **a new dummy non-terminal** that covers only the original terminal. Example: a rule for an infinitive verb phrase such as *INF-VP → to VP* would be replaced by two rules: *INF-VP → TO VP* and *TO → to.*

B. Rules with a single non-terminal on the right are called **unit productions**. Unit productions are eliminated by rewriting the right-hand side of the original rules with the right-hand side of all the non-unit production rules that they ultimately lead to. If $A \Rightarrow^* B$ by a chain of one or more unit productions, and $B \rightarrow \gamma$ is a non-unit production in our grammar, then we add $A \rightarrow \gamma$ for each such rule in the grammar, and discard all the intervening unit productions.

Conversion to CNF 2/2

C. Rules with right-hand sides longer than 2 are remedied through the introduction of new non-terminals that spread the longer sequences over several new productions.

Example: the rule *S → Aux NP VP* can be replaced by the two rules *S → X1 VP* and *X1 → Aux NP*.

- The entire conversion process can be summarized as follows:

1. *Copy all conforming rules to the new grammar unchanged,*
2. *Convert terminals within rules to dummy non-terminals,*
3. *Convert unit-productions*
4. *Binarize all rules and add to new grammar.*

	$\mathcal{L}_1$ Grammar	$\mathcal{L}_1$ in CNF
1	$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
3	$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
		$X1 \rightarrow Aux\ NP$
2	$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
		$S \rightarrow Verb\ NP$
		$S \rightarrow X2\ PP$
		$S \rightarrow Verb\ PP$
		$S \rightarrow VP\ PP$
2	$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
2	$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
2	$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
2	$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
1	$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
1	$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
	$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
1	$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
3	$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
		$X2 \rightarrow Verb\ NP$
1	$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
1	$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
1	$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

Lexicon

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

CKY Recognition

- Starting Point: When the grammar is in CNF, each non-terminal node above the part-of-speech level in a parse tree will have exactly **two daughters**.
- Representation: A simple two-dimensional matrix can be used to encode the structure of an entire tree.

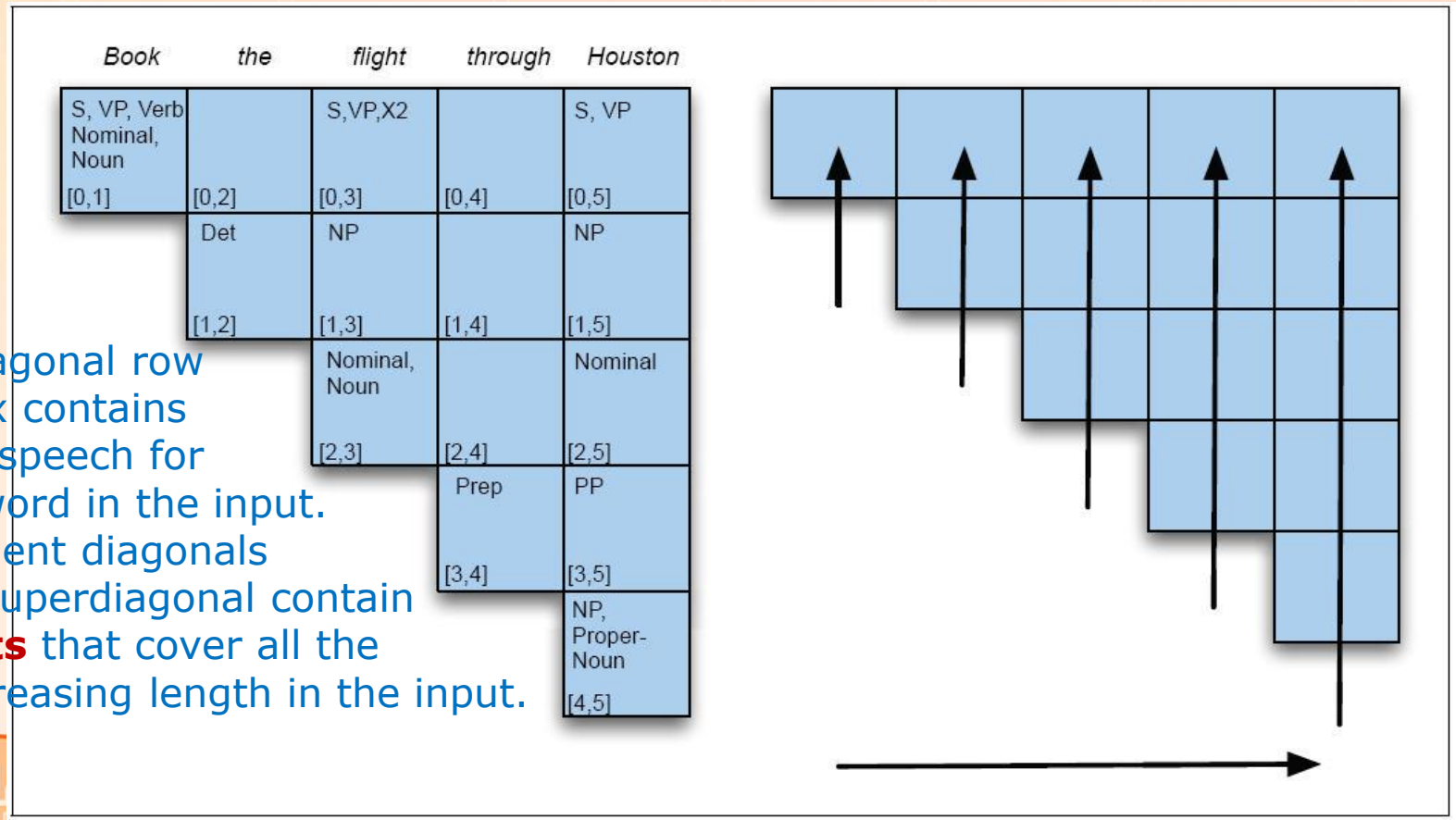
For a sentence of length *n*, we will be working with the upper-triangular portion of an (*n*+1) x (*n*+1) matrix.

- Each cell [*i, j*] in this matrix contains a set of **non-terminals** that represent all the constituents that span positions *i* through *j* of the input.
- Since our indexing scheme begins with 0, it's natural to think of the **indexes as pointing at the gaps between the input words.**

- The cell that represents the entire input resides in position [0,*n*] in the matrix.

CKY Recognition

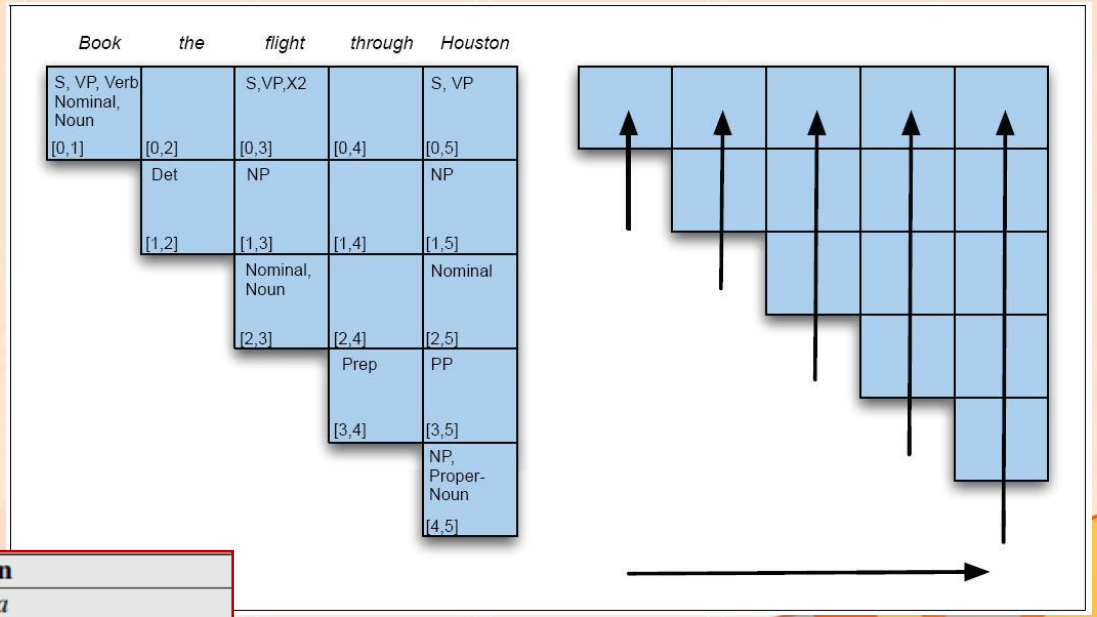
- For each constituent represented by an entry [*i*, *j*] in the table there must be a position in the input, *k*, where it can be split into two parts such that *i*<*k*<*j*.
- Given such a *k*, the first constituent [*i*, *k*] must lie to the left of entry [*i*, *j*] somewhere along row *i*, and the second entry [*k*, *j*] must lie beneath it, along column *j*.



The superdiagonal row in the matrix contains the parts of speech for each input word in the input. The subsequent diagonals above that superdiagonal contain **constituents** that cover all the spans of increasing length in the input.

How does it work?

- *Given this setup, CKY recognition consists of **filling the parse table** in the right way.*
- *To do this, we'll proceed in a **bottom-up fashion** so that at the point where we are filling any cell [i, j], the cells containing the parts that could contribute to this entry (i.e., the cells **to the left** and **the cells below**) have already been filled.*



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

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

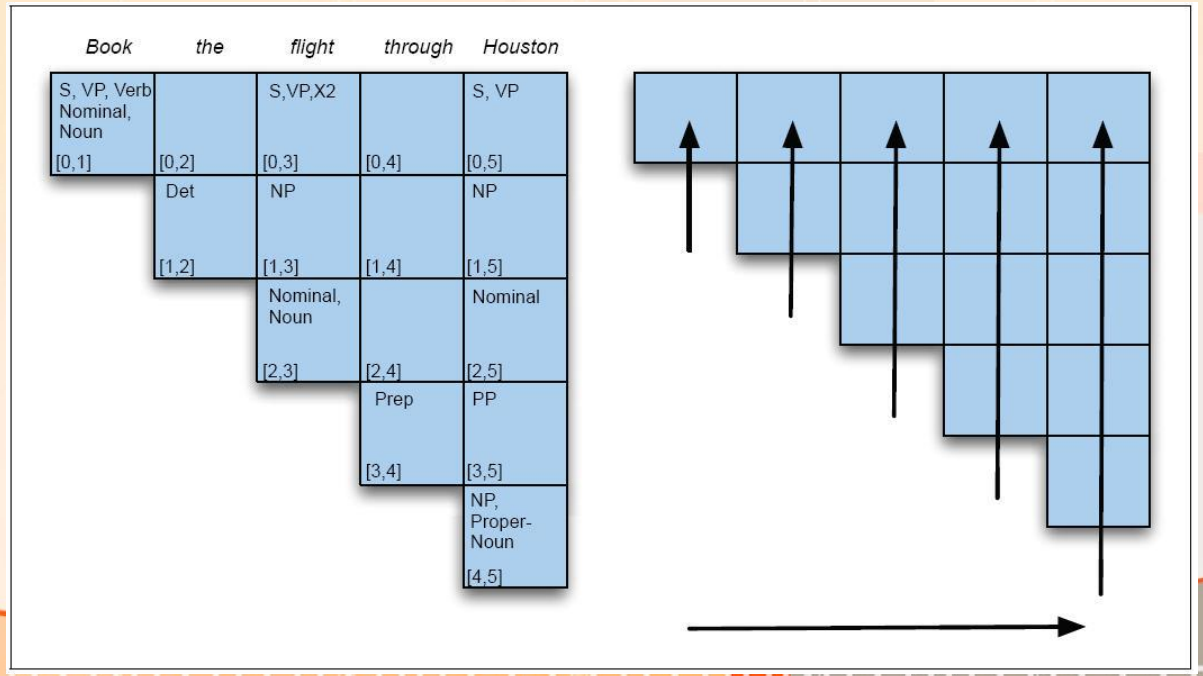
function CKY-PARSE (words, grammar) **returns** table

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

The **outermost loop** of the algorithm iterates over the columns,

the **second loop** iterates over the rows, from the bottom up.

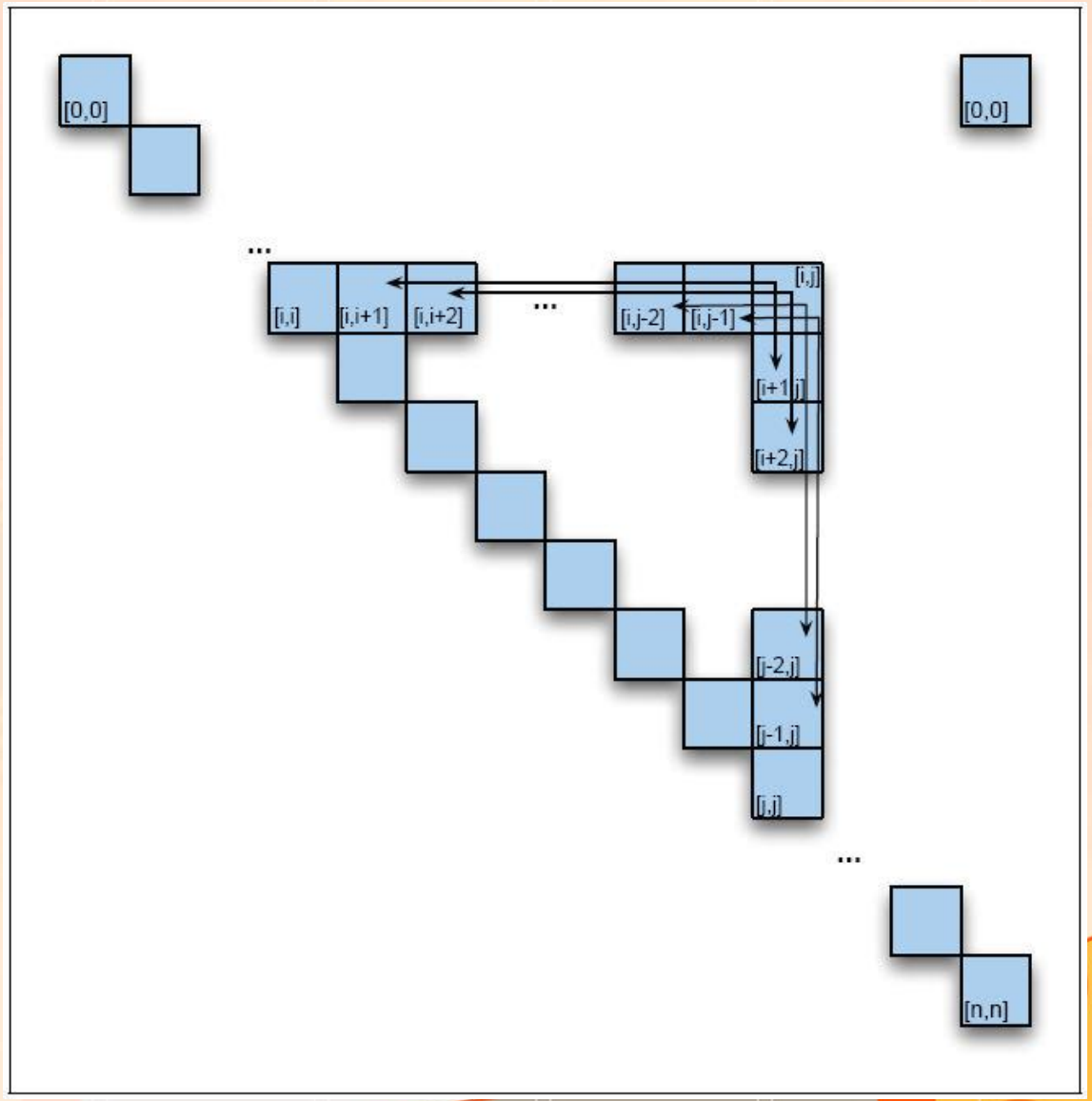
The purpose of the **inner-most loop** is to range over all the places where a substring spanning ***i*** to ***j*** in the input might be split in two.
As ***k*** ranges over the places where the string can be split, the pairs of cells we consider move, in lockstep, to the right along row ***i*** and down along column ***j***.



All the ways to fill the [I,j] cell in the CKY table

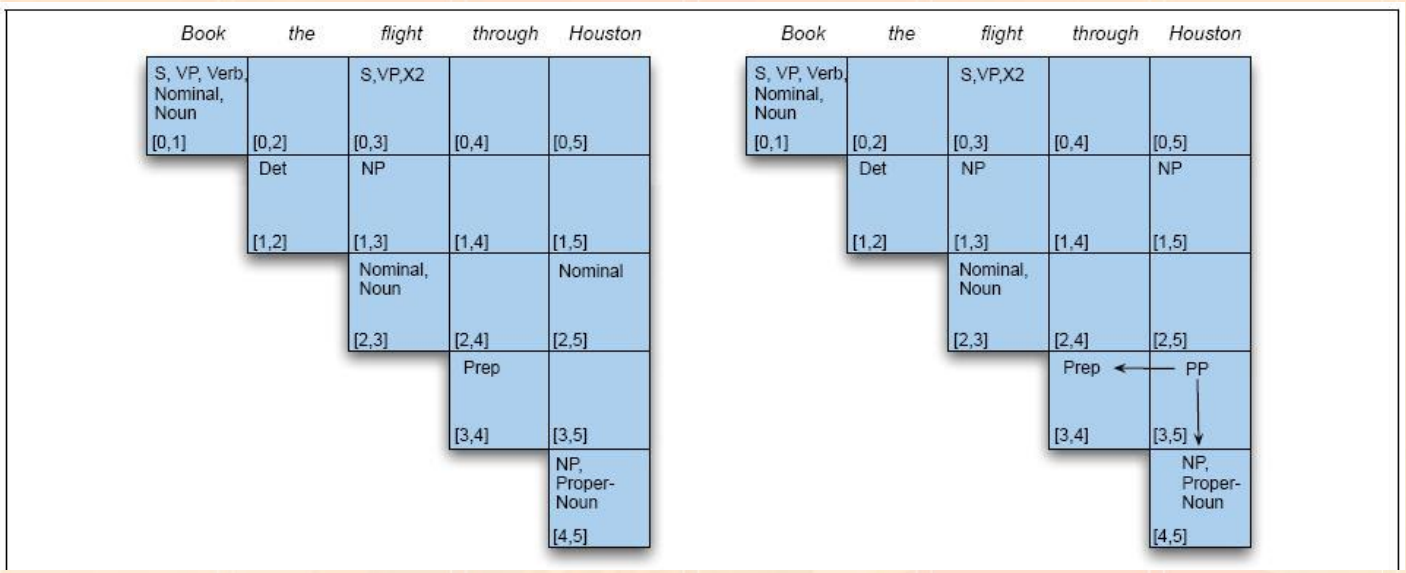
At each such split, the algorithm considers whether the contents of the two cells can be combined in a way that is sanctioned by a rule in the grammar.

- *If such a rule exists, the non-terminal on its left-hand side is entered into the table*



Example 1 (1/3)

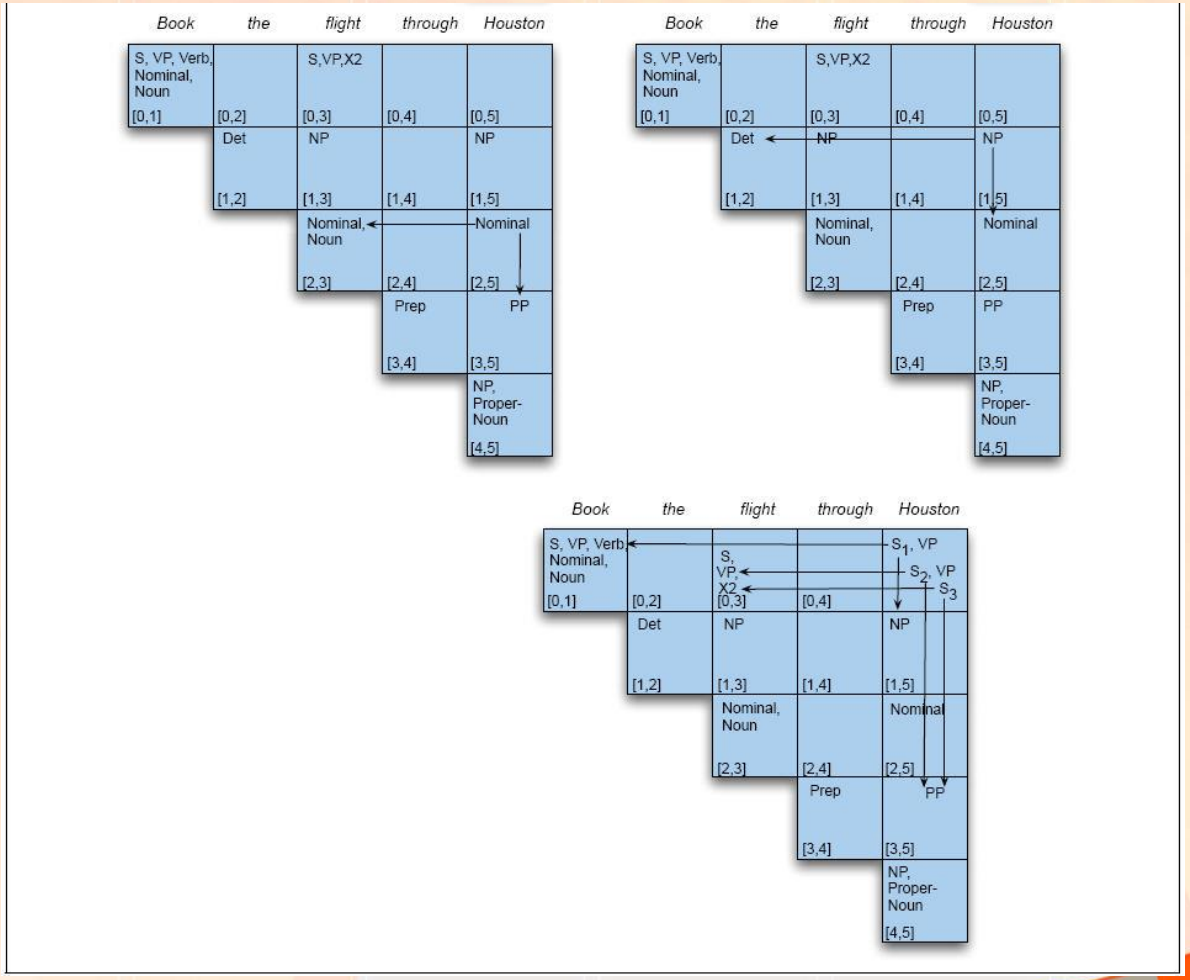
Show how the five cells of column 5 of the table are filled after the word *Houston* is read.



The arrows point out the two spans that are being used to add an entry to the table.

Example 1 (2/3)

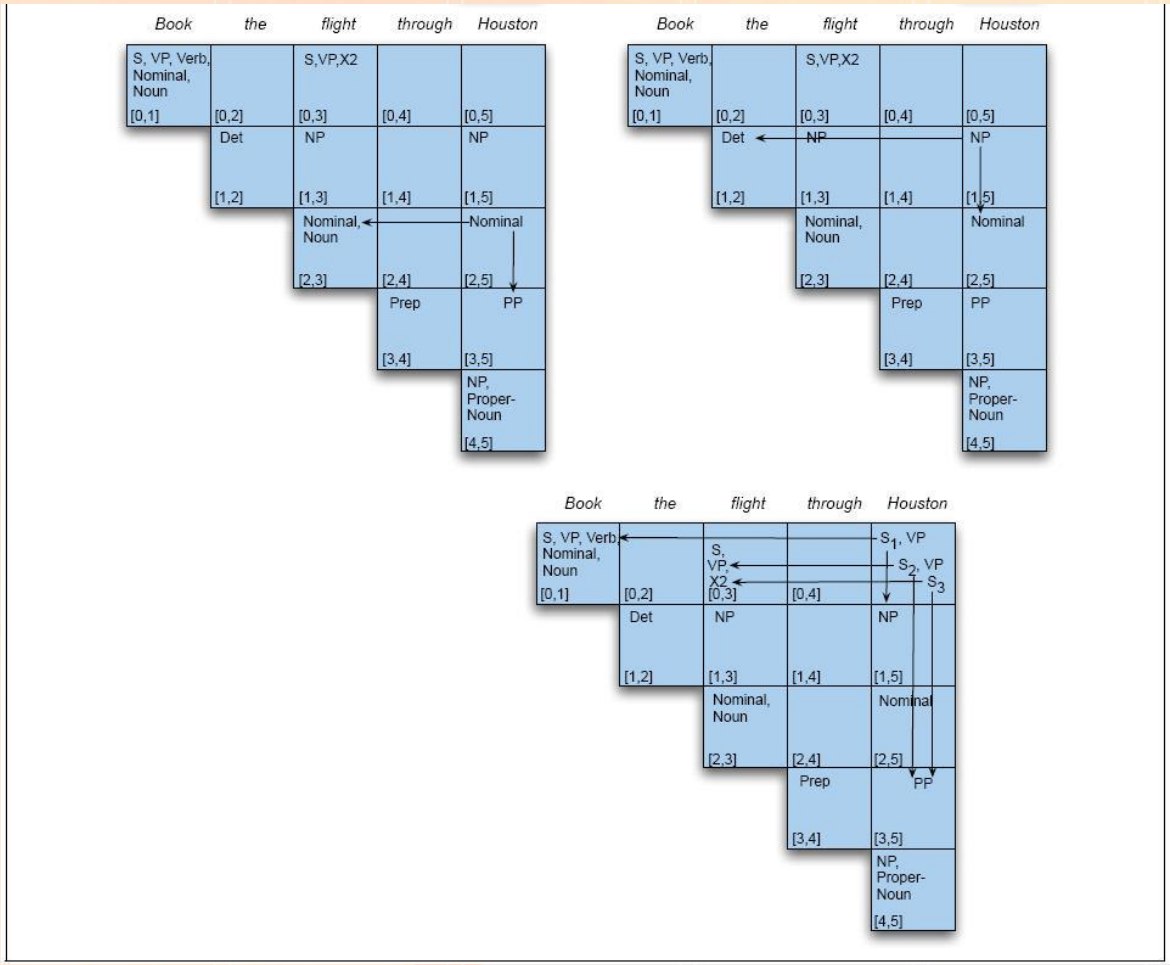
Show how the five cells of column 5 of the table are filled after the word *Houston* is read.



The arrows point out the two spans that are being used to add an entry to the table.

Example 1 (3/3)

Show how the five cells of column 5 of the table are filled after the word *Houston* is read.



Note that the action in cell [0, 5] indicates the presence of three alternative parses for this input, one where the *PP* modifies the *flight*, one where it modifies the *booking*, and one that captures the second argument in the original ***VP → Verb NP PP*** rule, now captured indirectly with the *VP → X2 PP* rule.

Example 2: CYK Algorithm

The	flights	leave	on	time
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

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

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

LENGTH (words) = 5

Example 2: CYK Algorithm

The	flights	leave	on	time
DET [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

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

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

LENGTH (words) = 5

j = 1
table[0,1]=DET

Example 2: CYK Algorithm

The	flights	leave	on	time
DET [0,1]	NP [0,2]	[0,3]	[0,4]	[0,5]
	Noun, Verb,VP [1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

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

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

LENGTH (words) = 5

j = 2
table[1,2]=Noun, Verb, VP
i = 0
k = 1
table[0,2]= NP
(A=NP B=DET C=Noun)

Example 2: CYK Algorithm

The	flights	leave	on	time
DET [0,1]	NP [0,2]	S [0,3]	[0,4]	[0,5]
	Noun, Verb, VP [1,2]	[1,3]	[1,4]	[1,5]
		Noun, Verb, VP [2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

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

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

j = 3
table[2,3]=Noun, Verb, VP
i = 1 to 0

i=1
k = 2
table[1,3]=
?table[1,2]♦ ?table[2,3]
(A=Noun|Verb B=Noun|Verb)
NOTHING

i=0
k = 1 to 2 k=1
table[0,3]= NOTHING
?table[0,1]♦ ?table[1,3]
(A=DET B=Nothing)
k=2 table[0,3]= S
?table[0,2]♦ ?table[2,3]
(A=NP B=Noun|Verb| VP)

Example 2

The	flights	leave	on	time
DET [0,1]	NP [0,2]	S [0,3]	[0,4]	[0,5]
	Noun, Verb [1,2]	[1,3]	[1,4]	[1,5]
		Noun, Verb, VP [2,3]	[2,4]	[2,5]
			Prep [3,4]	[3,5]
				[4,5]

```
function CKY-PARSE (words, grammar) returns table
for j ← 1 to LENGTH (words) do
    table[j-1,j] ← {A| A→ words [j] ∈ grammar}
    for i ← j-1 downto 0 do
        for k ← i+1 to j-1 do
            table[i,j] ← table[i,j] ∪
                {A| A → B C ∈ grammar,
                    B ∈ table[i,k],
                    C ∈ table[k,j] }
```

j = 4
table[3,4]=Prep
i = 2 to 0

i=0 k = 1 to 3 k=1
table[0,4]= NOTHING
?table[0,1]♦ ?table[1,4]
(A=DET B=NOTHING)

i=0 k = 1 to 3 k=2
table[0,4]= NOTHING
?table[0,2]♦ ?table[2,4]
(A=NP B=NOTHING)

i=0 k = 1 to 3 k=3
table[0,4]= NOTHING
?table[0,3]♦ ?table[3,4]
(A=S B=PREP)

i=2 k = 3
table[2,4]= NOTHING
?table[2,3]♦ ?table[3,4]
(A=Noun|Verb|VP B=Prep)

i=1 k = 2 to 3 k=2
table[1,4]= NOTHING
?table[1,2]♦ ?table[2,4]
(A=Noun|Verb B=NOTHING)

i=1 k = 2 to 3 k=3
table[1,4]= NOTHING
?table[1,3]♦ ?table[3,4]
(A=NOTHING B=Prep)

Example 2

The	flights	leave	on	time
DET [0,1]	NP [0,2]	S [0,3]	[0,4]	[0,5]
	Noun, Verb [1,2]	[1,3]	[1,4]	[1,5]
		Noun, Verb, VP [2,3]	[2,4]	VP [2,5]
			Prep [3,4]	PP [3,5]
				Noun, NP VP,Verb [4,5]

j = 5
table[3,4]=Noun|NP|Verb|VP
i = 3 to 0

i=3 k = 4
table[3,5]= PP
?table[3,4]♦ ?table[4,5]
(A=Prep B=Noun|**NP**|VP|Verb)

i=2 k = 3 to 4 k=3
table[2,5]= VP
?table[2,3]♦ ?table[3,5]
(A=Noun|Verb|VP B=PP)

i=2 k = 3 to 4 k=4
table[2,5]= VP ∪ NOTHING
?table[2,4]♦ ?table[4,5]
(A=NOTHING B=Noun|NP|VP|Verb)

i=1 k = 2 to 4 k=2
table[1,5]= NOTHING
?table[1,2]♦ ?table[2,5]
(A=Noun|Verb B=VP)

i=1 k = 2 to 4 k=3
table[1,5]= NOTHING
?table[1,3]♦ ?table[3,5]
(A=NOTHINNG B=PP)

i=1 k = 2 to 4 k=4
table[1,5]= NOTHING
?table[1,4]♦ ?table[4,5]
(A=NOTHING B=Noun|NP|VP|Verb)

Example 2

The	flights	leave	on	time
DET [0,1]	NP [0,2]	S [0,3]	[0,4]	**S** [0,5]
	Noun, Verb [1,2]	[1,3]	[1,4]	[1,5]
		Noun, Verb, VP [2,3]	[2,4]	VP [2,5]
			Prep [3,4]	PP [3,5]
				Noun, NP VP,Verb [4,5]

j = 5
table[3,4]=Noun|NP|Verb|VP
i = 3 to 0

i=0 k = 1 to 4 k=1
table[0,5]= NOTHING
?table[0,1]♦ ?table[1,5]
(A=Det B=NOTHING)

i=0 k=2
table[0,5]= **S**
?table[0,2]♦ ?table[2,5]
(A=NP B=VP)

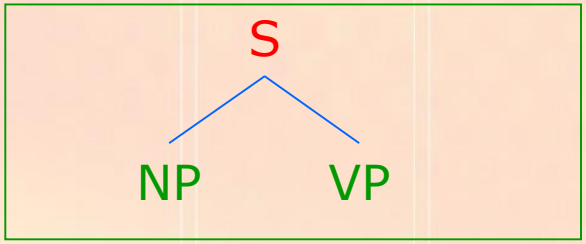
i=0 k=3
table[0,5]= S ∪ NOTHING
?table[0,3]♦ ?table[3,5]
(A=S B=PP)

i=0 k=4
table[0,5]= S ∪ NOTHING
?table[0,4]♦ ?table[4,5]
(A=NOTHING B=Noun|NP|VP|Verb)

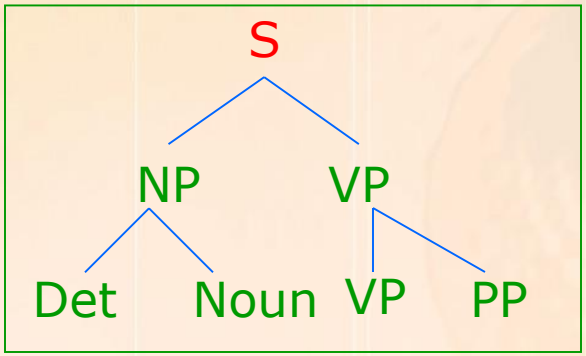
Example 2: What is the parse ???

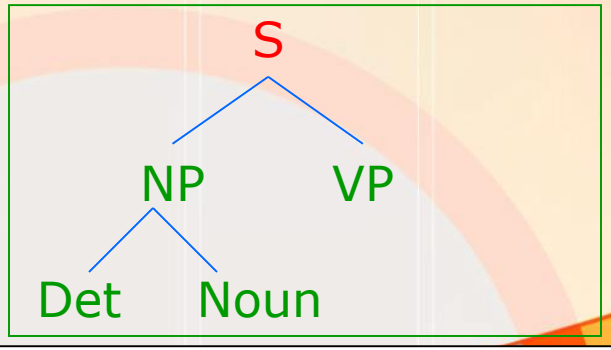
The	flights	leave	on	time
DET [0,1]	NP [0,2]	S [0,3]	[0,4]	**S** [0,5]
	Noun, Verb [1,2]	[1,3]	[1,4]	[1,5]
		Noun, Verb, VP [2,3]	[2,4]	VP [2,5]
			Prep [3,4]	PP [3,5]
				Noun, NP VP,Verb [4,5]

i=0 k=2
table[0,5]= **S**
?table[0,2]♦ ?table[2,5]
(A=NP B=VP)



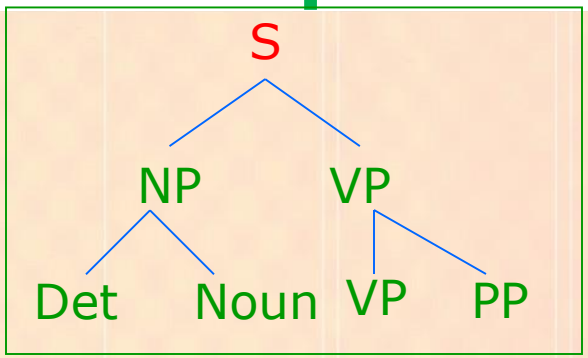
table[0,2]= NP
(A=NP B=DET C=Noun)

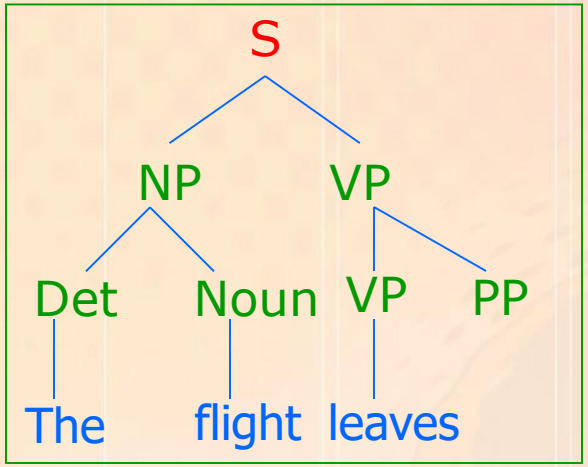


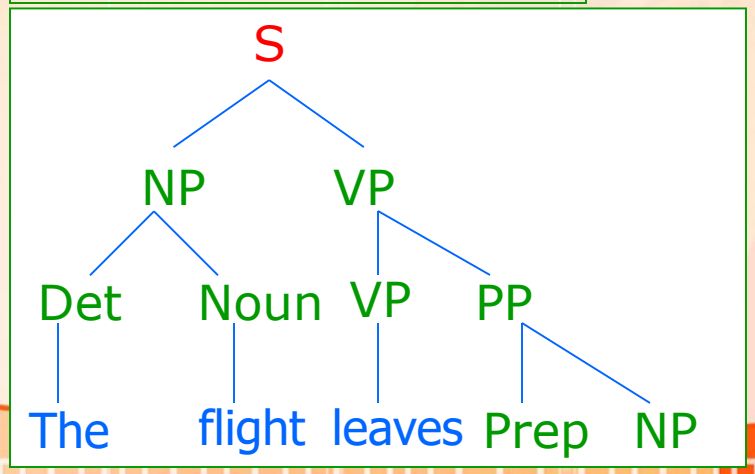


i=2 k = 3 to 4 k=3
table[2,5]= VP
?table[2,3]♦ ?table[3,5]
(A=Noun|Verb|**VP** B=PP)

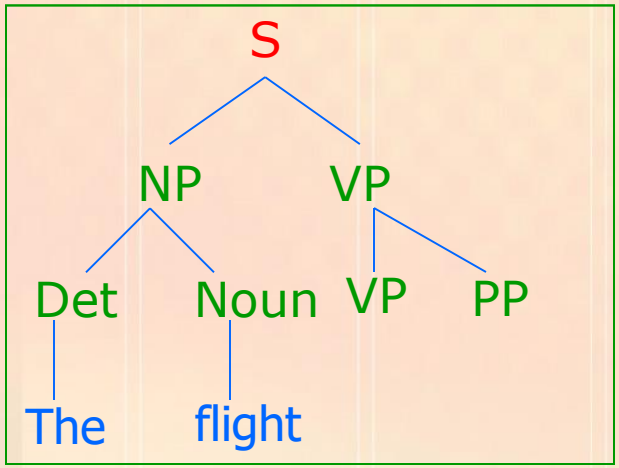
Example 2: What is the parse ???







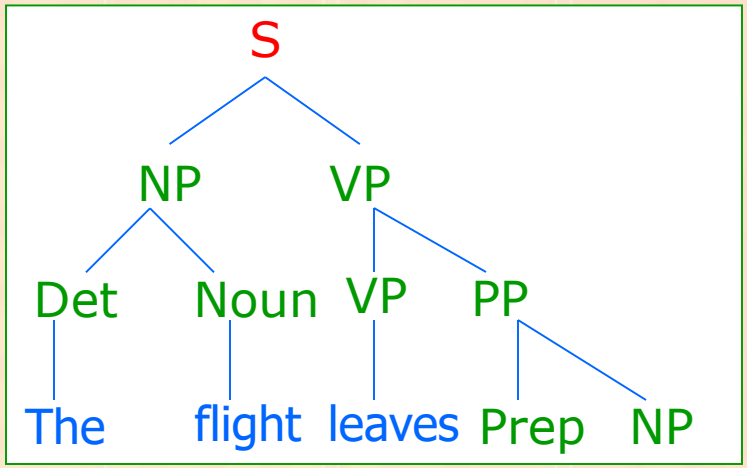
table[0,2]= NP
(A=NP B=DET C=Noun)



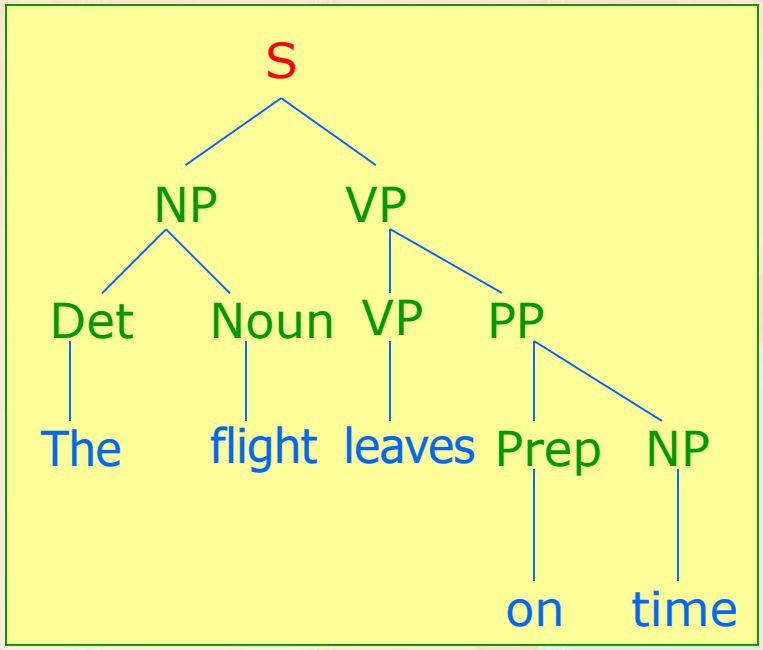
i=2 k = 3 to 4 k=3
table[2,5]= VP
?table[2,3]♦ ?table[3,5]
(A=Noun|Verb|**VP** B=PP)

i=3 k = 4
table[3,5]= PP
?table[3,4]♦ ?table[4,5]
(A=Prep B=Noun|**NP**|VP|Verb)

Final Parse



i=3 k = 4
table[3,5]= PP
?table[3,4]♦ ?table[4,5]
(A=Prep B=Noun|**NP**|VP|Verb)



CKY Parsing – Observations 1/2

- The algorithm described is a recognizer, not a parser!!!

Why???? for it to succeed it simply has to find an *S* in cell [0, *N*].

- To turn it into a parser capable of returning all possible parses for a given input, we'll make two simple changes to the algorithm:

1. the first change is to augment the entries in the table so that each non-terminal is paired with pointers to the table entries from which it was derived (more or less as shown in the previous Figures),
2. the second change is to permit multiple versions of the same non-terminal to be entered into the table With these changes, the completed *table contains all the possible parses* for a given input.

- Returning an arbitrary single parse consists of choosing an *S* from cell [0, *n*] and then recursively retrieving its component constituents from the table.

CKY Parsing – Observations 2/2

- Returning all the parses for a given input may incur considerable cost.
- there may be an exponential number of parses associated with a given input. In such cases, returning all the parses will have an unavoidable exponential cost.
- We can also think about retrieving the best parse for a given input by further augmenting the table to contain the *probabilities* of each entry.
- Retrieving the most probable parse consists of running a suitably modified version of the Viterbi algorithm over the completed parse table.

Partial Parsing

- Many language-processing tasks simply do not require complex, complete parse trees for all inputs. For these tasks, a **partial parse**, or **shallow parse**, of input sentences may be sufficient.
  - For example, **information extraction** systems generally do not extract *all* the possible information from a text; they simply identify and classify the segments in a text that are likely to contain valuable information.
- There are many different approaches to partial parsing.
  - Some approaches make use of cascades of FSTs to try to produce representations that closely approximate syntactic trees.
  - These approaches typically produce flatter trees. This flatness arises from the fact that such approaches generally defer decisions that may require semantic or contextual factors, such a prepositional phrase attachments, coordination ambiguities, and nominal compound analyses. Nevertheless the intent is to produce parse-trees that link all the major constituents in an input.

Chunk Parsing

- At the other end of the spectrum is a style of partial parsing known as **chunking**.
- Definition: Chunking is the process of identifying and classifying the flat non-overlapping segments of a sentence that **constitute the basic non-recursive phrases** corresponding to the major parts-of-speech found in most wide-coverage grammars.

This set typically includes NPs, VPs, APs, and PPs.

Since chunked texts lack a hierarchical structure, a simple bracketing notation is sufficient to denote the location and the type of the chunks in a given example.

Example: [$_{NP}$The morning flight] [$_{PP}$from] [$_{NP}$Denver] [$_{VP}$has arrived.]

This bracketing notation makes clear the two fundamental tasks that are involved in chunking: finding the non-overlapping extents of the chunks, and assigning the correct label to the discovered chunks.

Syntactic Phrase

- *The details of what constitutes a syntactic phrase for any given system varies according to the syntactic theories underlying the system and whether the phrases are being derived from a treebank.*
- *What constitutes a syntactic base phrase depends on the application (and whether the phrases come from a treebank).*
- *Base phrases of a given type do not recursively contain any constituents of the same type. Eliminating this kind of recursion leaves us with the problem of determining the boundaries of the non-recursive phrases.*
- In most approaches, base-phrases include the headword of the phrase, along with any pre-head material within the constituent, while crucially excluding any post-head material.
  - Eliminating post-head modifiers from the major categories automatically removes the need to resolve attachment ambiguities.
- Example: *a flight from Indianapolis to Houston on TWA* is reduced to the following:

  [$_{NP}$ a flight] [$_{PP}$ from] [$_{NP}$ Indianapolis] [$_{PP}$ to] [$_{NP}$ Houston] [$_{PP}$ on] [$_{NP}$ TWA]

Learning-Based Approaches to Chunking 1/5

- *State-of-the-art approaches to chunking use supervised machine learning to train a chunker by using annotated data as a training set and training any sequence labeler.*

The standard way to do this has come to be called **IOB tagging** and is accomplished by introducing tags to represent the beginning (B) and internal (I) parts of each chunk , as well as those elements of the input that are outside (O) ANY CHUNK.

- Under this scheme, the size of the tagset is (2*n*+1) where *n* is the number of categories to be classified.

- Example: *The morning flight from Denver has arrived*

  B_NP I_NP I_NP B_PP B_NP B_VP I_VP

Learning-Based Approaches to Chunking 2/5

The same sentence with only the base-NPs tagged illustrates the role of the O tags.

The morning flight from Denver has arrived

B_NP I_NP I_NP O B_NP O O

Note: *that there is no explicit encoding of the end of a chunk in this scheme*;

- the end of any chunk is implicit in any transition from an I or B, to a B tag, or from an I to an O tag.

- This encoding reflects the notion that when sequentially labeling words, it is generally quite a bit easier (at least in English) to detect the beginning of a new chunk than it is to know when a chunk has ended.

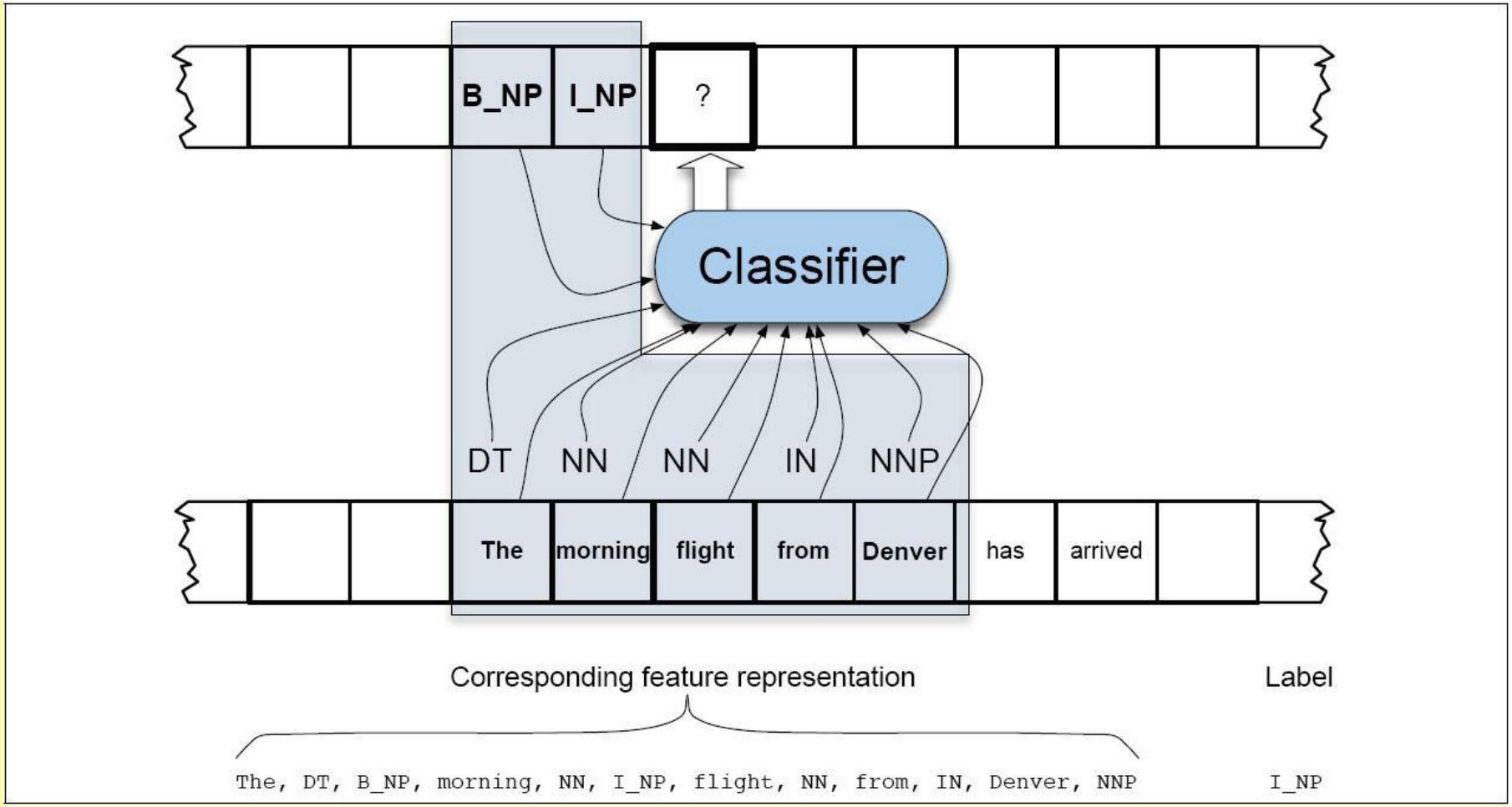
- *Since annotation efforts are expensive and time consuming, chunkers usually rely on existing treebanks like the Penn Treebank , extracting syntactic phrases from the full parse constituents of a sentence, finding the appropriate heads and then including the material to the left of the head, ignoring the text to the right.*

Learning-Based Approaches to Chunking 3/5

Having extracted a training corpus from a treebank, we must now cast the training data into a form that's useful for training classifiers.

- Each input can be represented as a set of features extracted from **a context window** that surrounds the word to be classified. Using a window that extends two words before, and two words after the word being classified seems to provide reasonable performance.

- Features extracted from this window include: the words themselves, their parts-of-speech, as well as the chunk tags of the preceding inputs in the window.



- During training, the classifier would be provided with a training vector consisting of the values of 12 features (using Penn Treebank tags) as shown.

More considerations

The best current systems achieve an F-measure of around 96% on the task of base-NP chunking. The exact choice of learning approach seems to have little impact on these results; a wide-range of machine learning approaches achieve essentially the same results (Cardie et al., 2000).

- Examples that involve pre-nominal modifiers and conjunctions.

[$_{NP}$ Late arrivals and departures] are commonplace during winter.

[$_{NP}$ Late arrivals] and [$_{NP}$ cancellations] are commonplace during winter.

In the first example, *late* is shared by both *arrivals* and *departures* yielding a single long base-NP.

In the second example, *late* is not shared and modifies *arrivals* alone, thus yielding two base-NPs.

Distinguishing these two situations, and others like them, requires access to semantic and context information unavailable to current chunkers.

Summary

- **The two major ideas introduced in this chapter are those of parsing and partial parsing.**

❑ *the main points we covered about these ideas:*

- *Structural ambiguity is a significant problem for parsers. Common sources of structural ambiguity include PP-attachment, coordination ambiguity, and noun-phrase bracketing ambiguity.*
- *Dynamic programming parsing algorithms, such as CKY, use a table of partial parses to efficiently parse ambiguous sentences.*
- *CKY restricts the form of the grammar to Chomsky normal form (CNF).*
- *Many practical problems, including information extraction problems, can be solved without full parsing.*
- *Partial parsing and chunking are methods for identifying shallow syntactic constituents in a text.*
- *State-of-the-art methods for partial parsing use supervised machine learning techniques.*