Syntactic Parsing

dynamic programming

Syntactic parsing

The task of recognizing a sentence and assigning a syntactic structure to it.

Parse trees are useful in:

- grammar checking
- semantic analysis
- question answering

  What books were written by British women authors before 1800?

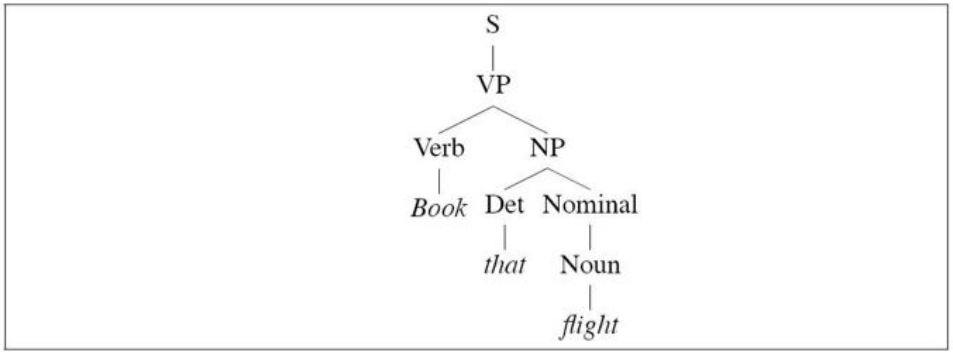
- need to know the subject is *what books*
- by-adjunct: *British women authors*

Parsing as Search

Finite-state automata: Recognition algorithms were formalized as search algorithms

- depth-first search
- breadth-first search

In syntactic parsing, we're searching through the space of possible parses and finding a parse, or all parses.

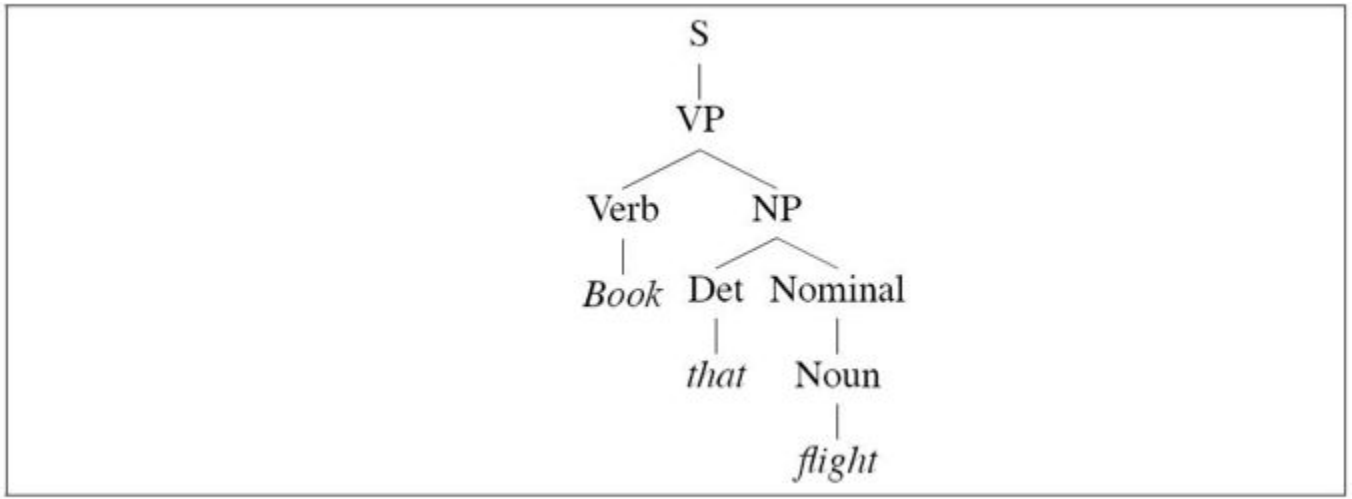


L_1 miniature grammar

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

L_1 miniature grammar

Given this grammar, the correct parse for the sentence is



Parsing as Search

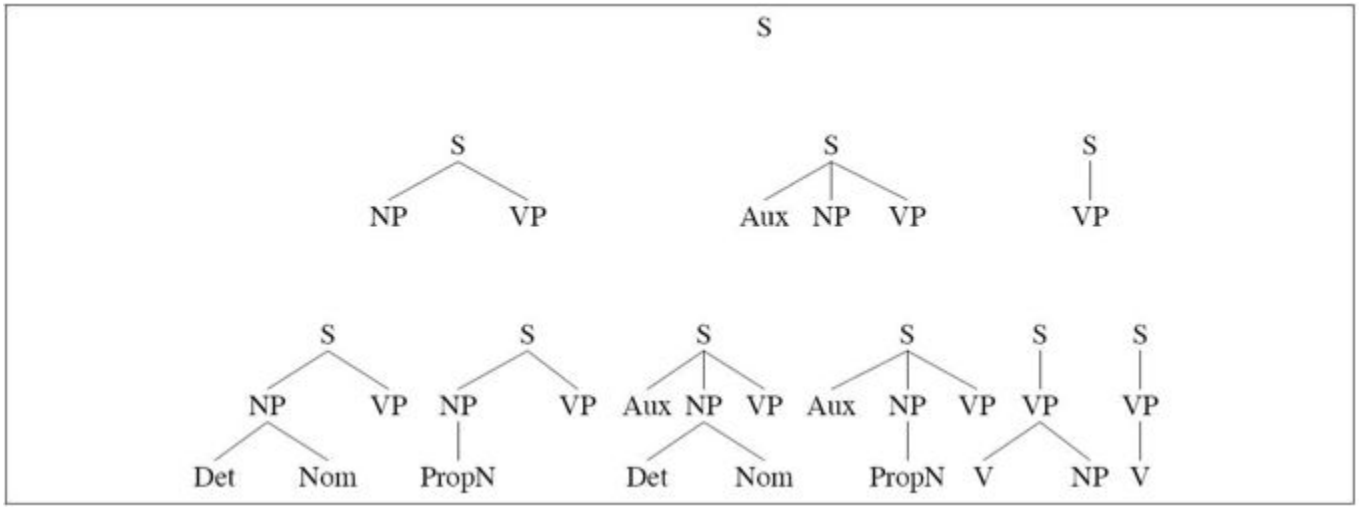
Our goal: Given the grammar, find all the trees whose root is the start symbol S and that cover exactly the words in the input.

Two kinds of constraints:

- Top-down:
  - All parses must begin with S (goal-directed search)
- Bottom-up:
  - All parses must yield the words *Book that flight* (data-directed search)

Top-down parsing

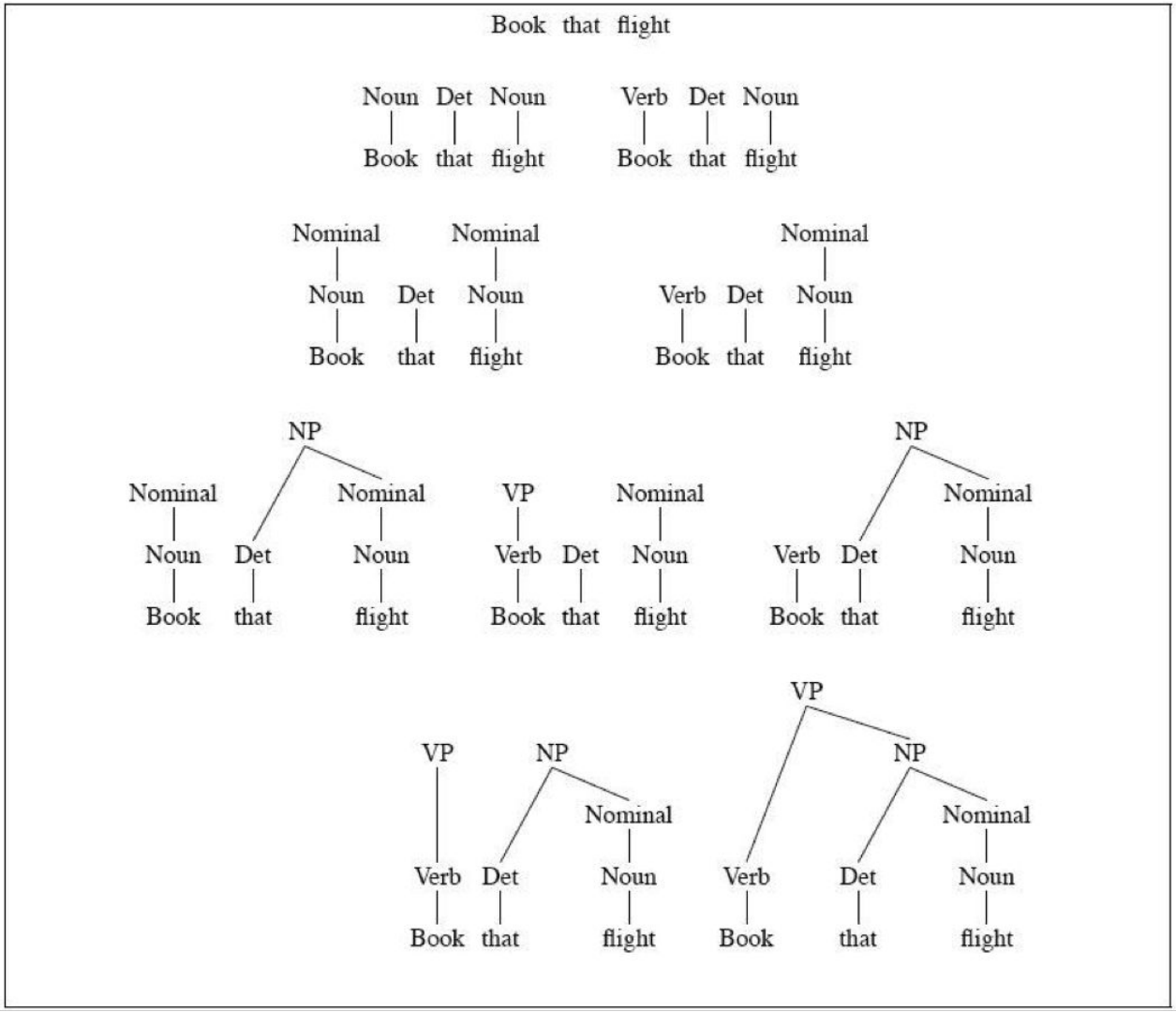
Search for a parse tree by trying to build from the root node S down to the leaves.



Bottom-up parsing

The earliest known parsing algorithm and is used in **shift-reduce parsers** common for programming languages. (Yngve 1955)

- Start with the POS and try to apply rules bottom-up until you reach an S.
- Any false starts that cannot be continued up are terminated.


Book that flight
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Nominal
Noun
Book
Det
that
Nominal
Noun
flight
Nominal
Verb
Book
Det
that
Noun
flight
NP
Nominal
Noun
Book
Det
that
Nominal
Noun
flight
VP
Verb
Book
Det
that
Nominal
Noun
flight
NP
Verb
Book
Det
that
Nominal
Noun
flight
VP
Verb
Book
NP
Det
that
Nominal
Noun
flight
VP
Verb
Book
NP
Det
that
Nominal
Noun
flight

Top-down vs. Bottom-up parsing

Top-down strategy:

- Never wastes time exploring trees that cannot result in an S.
- Never explores subtrees that cannot find a place in some S-rooted tree.

Bottom-up strategy:

- Wildly generates trees that may have no hope of fitting in an S-rooted tree.

Top-down vs. Bottom-up parsing

Top-down strategy:

- Spends considerable time on S-trees that are inconsistent with the input.
- Generates trees before even examining the input.

Bottom-up strategy:

- Never suggests trees that are not at least locally grounded in the input.

Structural ambiguity

We have seen POS ambiguity (*book* can be a verb or a noun)

Structural ambiguity is when a grammar assigns **more than one possible parse** to a sentence.

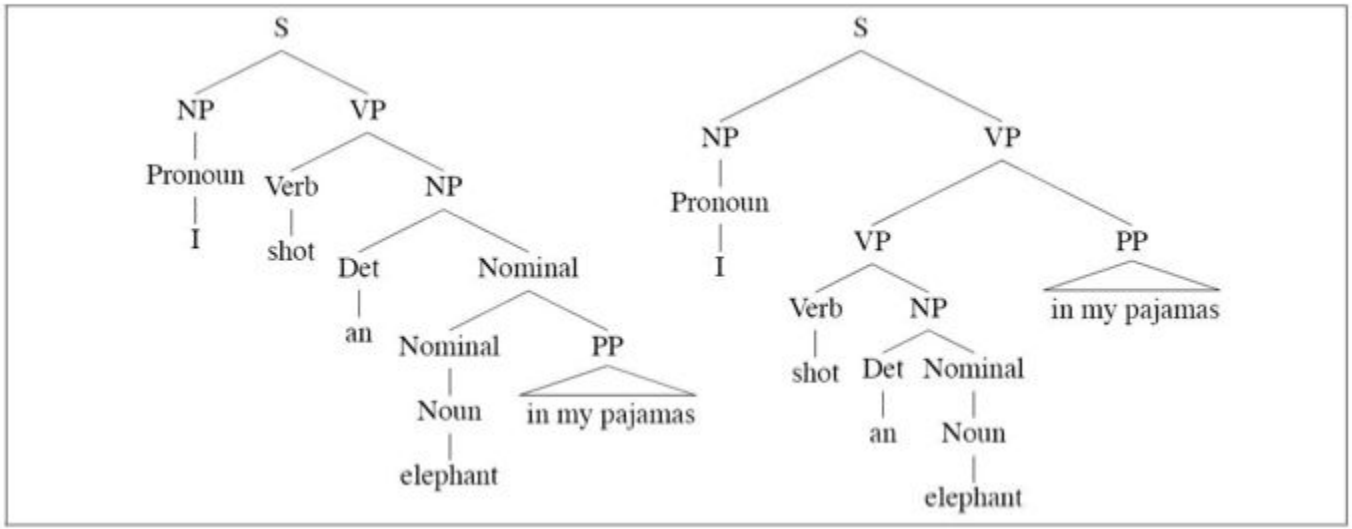
Structural ambiguity

One morning I shot an elephant in my pajamas.

How he got into my pajamas I don’t know.

Structural ambiguity

Attachment ambiguity: *Elephant in my pajamas* or *shot in my pajamas*?



Structural ambiguity

Adverbial phrase attachment ambiguity:

> *We saw the **Eiffel Tower flying to Paris.***
>
> ***We saw** the Eiffel Tower **flying to Paris.***

Coordination ambiguity:

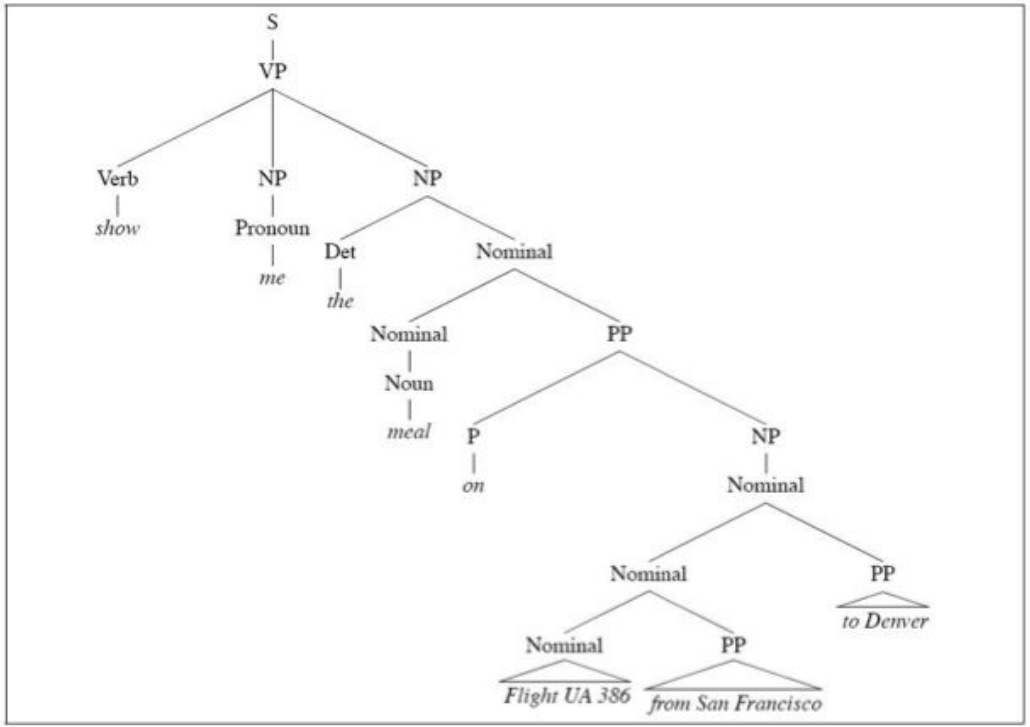
> ***[old [men and women]]*** `// both are old`
>
> ***[old men]** and **[women]*** `// only the men are old`

Structural ambiguity

14 parses for this sentence!

- VP → VP PP
- Nominal → Nominal PP

The number of parses grow **exponentially**, at the same rate as the number of parenthesizations of arithmetic expressions.



Structural ambiguity

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

Dynamic programming parsing

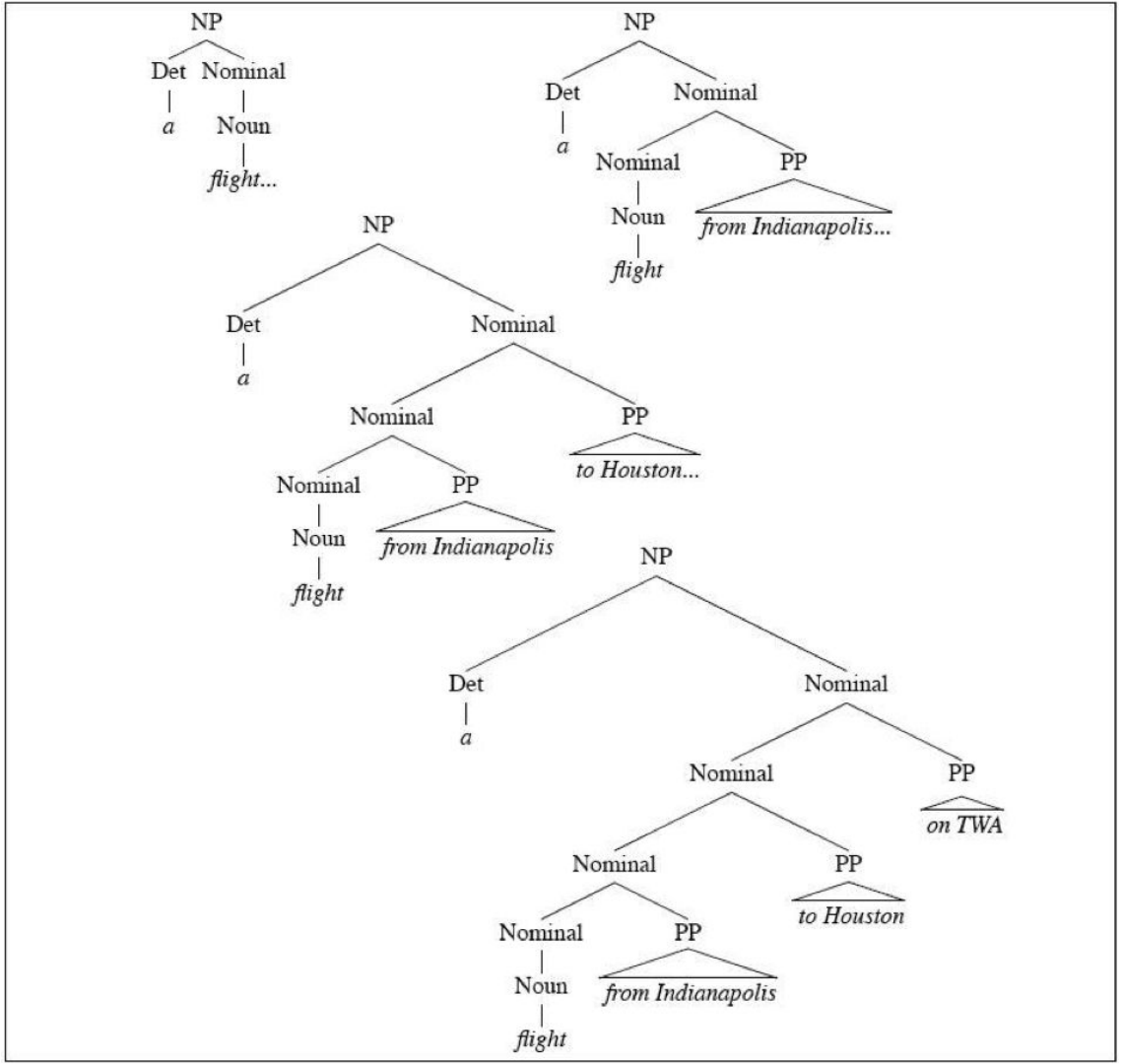
Both top-down and bottom-up parsing require that we explore all possible parse trees in parallel.

We can use the efficient **state space search** algorithms that we saw for FSA

- Depth-first search
- Breadth-first search
- or any sort of agenda-based approach where an agenda can tell us what trees to consider next.

Unfortunately this leads to a lot of *duplicate effort.*

False starts of a top-down depth-first left-to-right backtracking parser.



Dynamic programming parsing

Systematically fill in tables of solutions to sub-problems. When complete, the table contains the solution to all the sub-problems needed to solve the problem as a whole.

In parsing, solutions to sub-problems are going to be **subtrees for each potential constituent** in the input.

These subtrees then are *looked up* and not re-parsed.

CKY parsing

Also known as the Cocke-Kasami-Younger algorithm.

One requirement: The grammar must be in **Chomsky normal form (CNF)** (Chomsky, 1963).

A grammar is in CNF if

- it is ϵ-free
- each production is either of the form **A** → **B C** or **A** → **a** for some terminal a.

These grammars are **binary branching**.

CNF conversion

Assuming ϵ-freeness, deal with the following cases:

- RHS mixes terminals and nonterminals
  - Replace INF → to VP with INF → TO VP and TO → to
- RHS is a single nonterminal (unit production / unary rule)
  - if A → … → B and B → γ then we add A → γ and discard the intervening unary rules
- RHS is more than 2 nonterminals (N-ary rule)
  - A → B C γ then introduce A → X_1 γ and X_1 → B C and keep iterating

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

CKY recognition

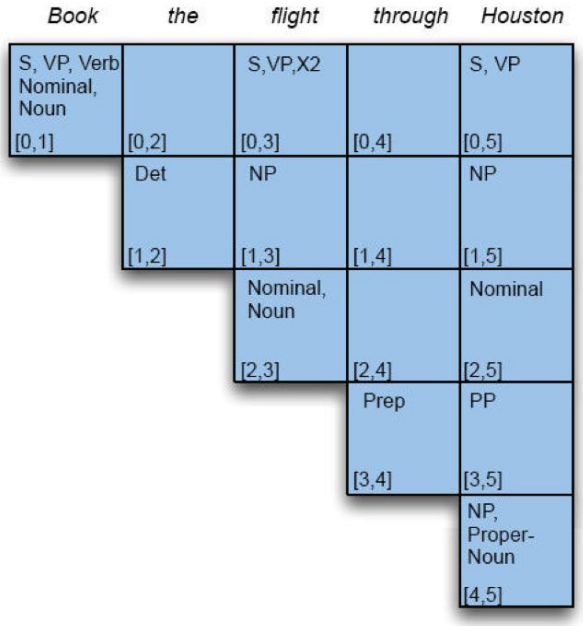
In dynamic programming parsing, our “solutions to sub-problems” are going to be **subtrees for each *potential* constituent** in the input.

$_0$ Book $_1$ that $_2$ flight $_3$

CKY recognition

All possible constituent spans:

- $_0$ Book $_1$
- $_1$ that $_2$
- $_2$ flight $_3$
- $_0$ Book $_1$ that $_2$
- $_1$ that $_2$ flight $_3$
- $_0$ Book $_1$ that $_2$ flight $_3$



We can use the upper-triangular portion of an (n+1) x (n+1) matrix to represent all constituents that span *i* through *j* of the input.

CKY recognition

In any tree, each node above the POS level will have exactly two children. Why?

$$_0\text{Book}\ _1\text{that}\ _2\text{flight}\ _3$$

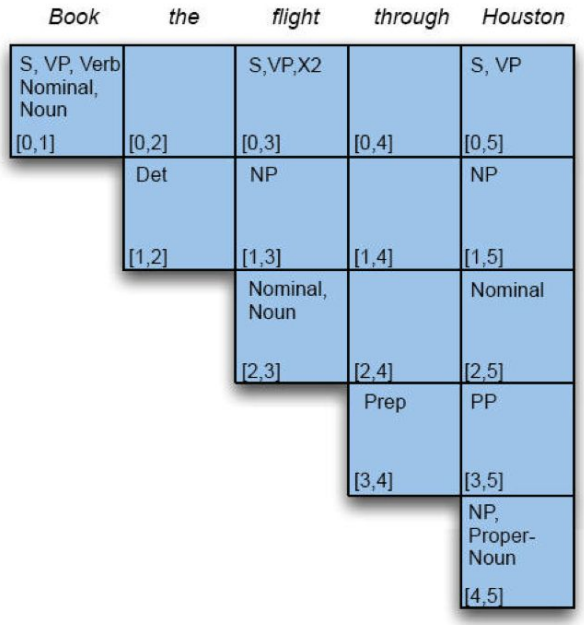
Therefore each constituent span must have a corresponding position *k*, $i < k < j$ where it can be split into two constituents.

$$_i A_j \rightarrow {}_i B_k\ C_j$$

CKY recognition

Our task is simply to fill the parse table in the right way.

We will proceed in a bottom-up fashion, forming constituents for *[i, j]* out of already filled constituents of *[i, k]* and *[k, j]* considering all possible *k* such that $i < k < j$.

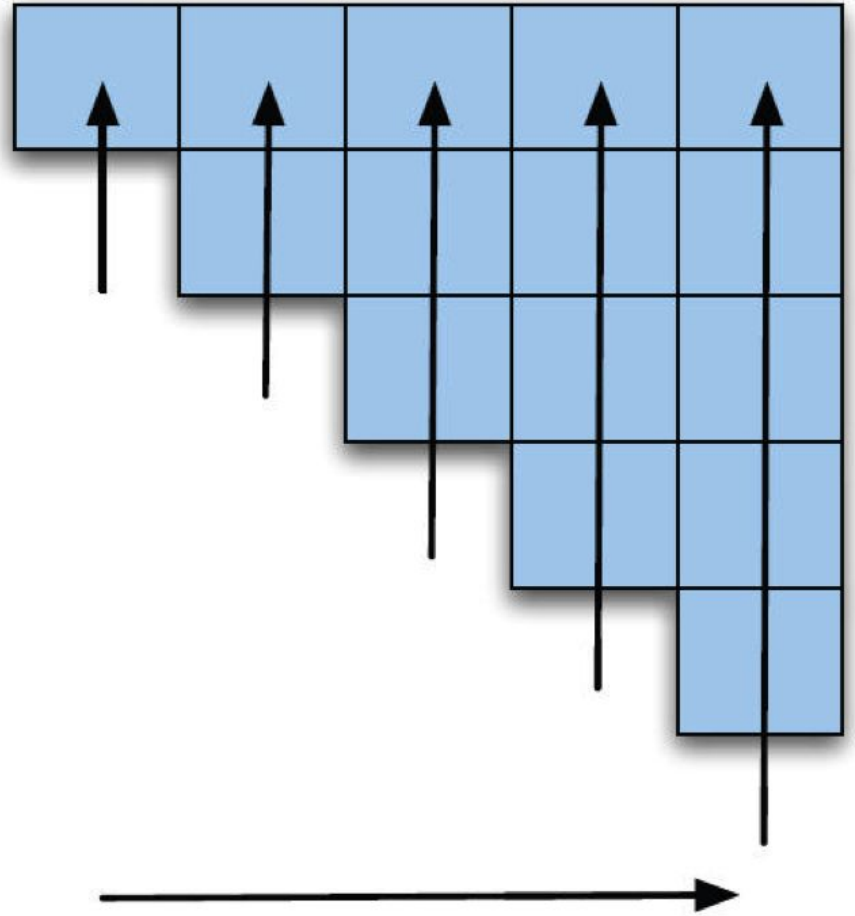


i < k < j

i < k

k < j

Book	*the*	*flight*	*through*	*Houston*
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]



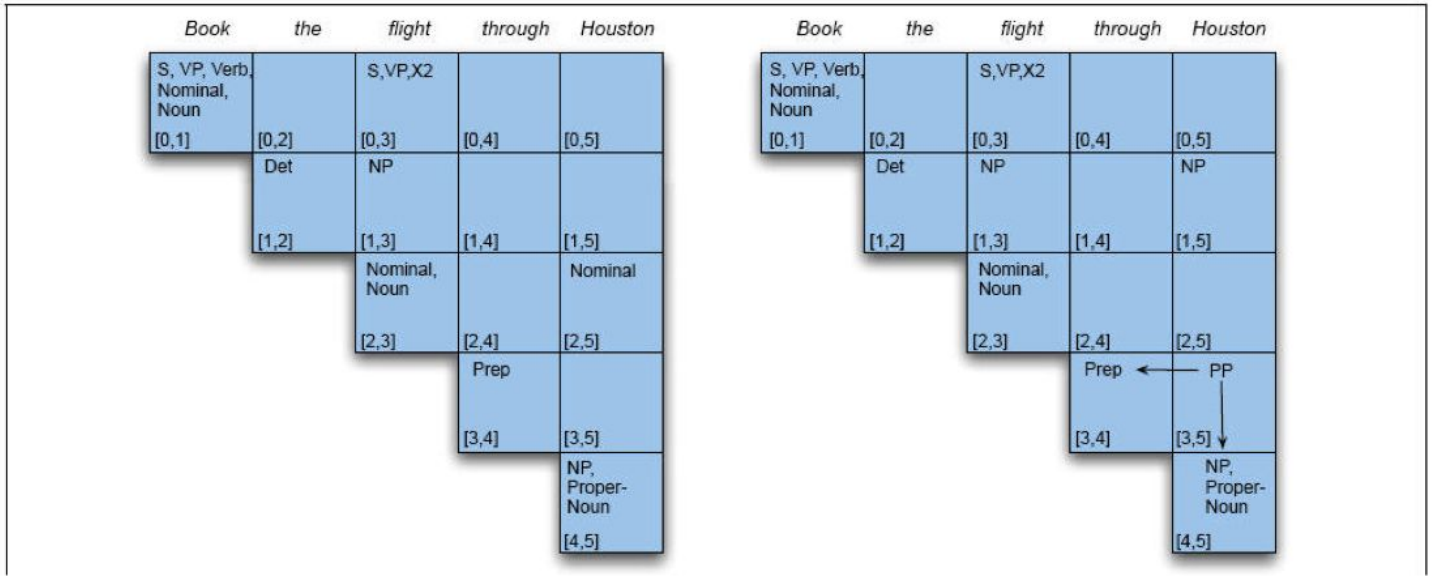
CKY recognition

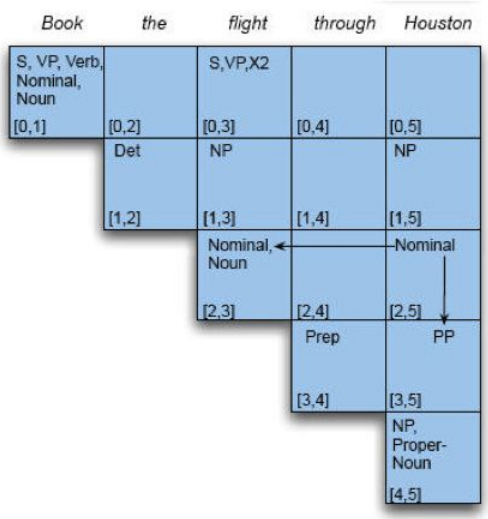
function CKY-PARSE(*words*, *grammar*) **returns** *table*

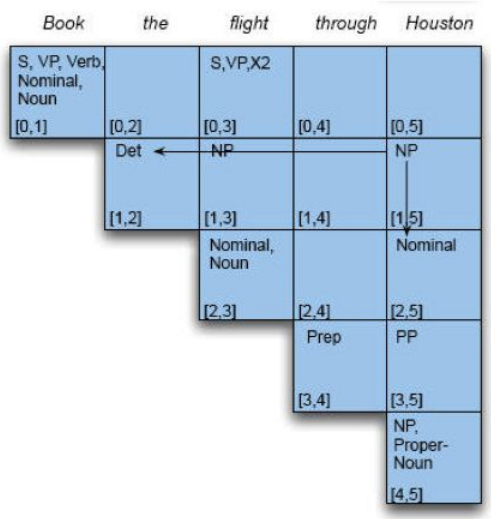
for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid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

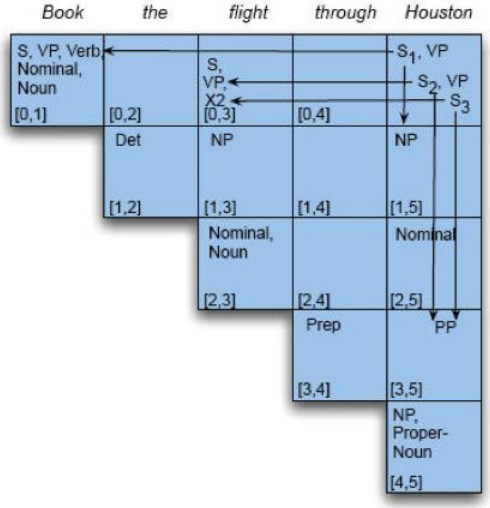
CKY recognition

After the word *Houston* is read, this is how we fill the 5 cells in the corresponding column.



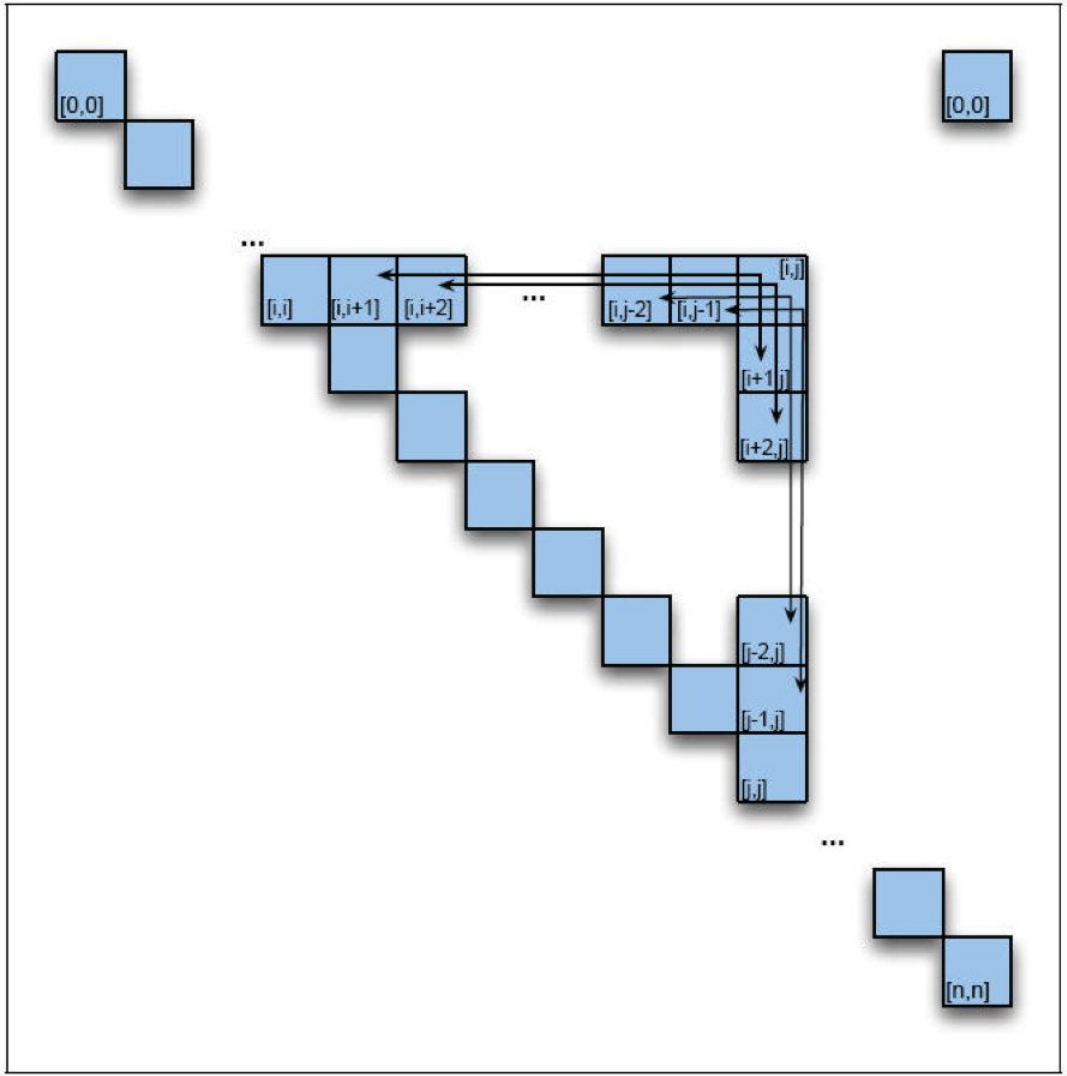

Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S, VP, X2
[0,3]
[0,4]
S1, VP
S2, VP
S3
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY recognition

All the ways to fill the [i, j]*th* cell in the CKY table.



CKY parsing

This algorithm builds a set of nonterminals that can span [i, j] of the input.

In this way, it is a **recognizer**, not a parser. When it discovers an S for the *[0, N]* span of the input, it recognizes the sentence as being **parseable**.

CKY parsing

If we want a parser (that will return all possible parses of a sentence) we must make changes to the algorithm:

1. Every nonterminal in a cell has two **pointers** to the constituents from which it was derived
2. Allow for multiple versions of the same nonterminal to be entered into one cell of the table.

Statistical Parsing

Probabilistic CKY

Statistical parsing

We will see that it is possible to build probabilistic models of syntactic knowledge and use some of this probabilistic knowledge in efficient probabilistic parsers.

Why probabilistic?

- **Disambiguation**: Deal with syntactic ambiguity (attachment/coordination ambiguity) in a principled way and resolve them by choosing the most probable interpretation.
- **Language modeling**: Define a probability distribution over sentences that is better than n-grams (accounting for longer range dependencies)

Probabilistic CFG (PCFG)

A PCFG differs from a CFG by augmenting each rule with a conditional probability

$A \rightarrow \beta$ [p]

Here p expresses the probability that A will be expanded to the sequence β

$P(A \rightarrow \beta)$ or $P(A \rightarrow \beta \mid A)$ or $P(\text{RHS} \mid \text{LHS})$

Thus, we have a probability distribution for every nonterminal LHS

$$\sum_{\beta} P(A \rightarrow \beta) = 1$$

Grammar		Lexicon
S → *NP VP*	[.80]	*Det* → *that* [.10] | *a* [.30] | *the* [.60]
S → *Aux NP VP*	[.15]	*Noun* → *book* [.10] | *flight* [.30]
S → *VP*	[.05]	| *meal* [.15] | *money* [.05]
NP → *Pronoun*	[.35]	| *flights* [.40] | *dinner* [.10]
NP → *Proper-Noun*	[.30]	*Verb* → *book* [.30] | *include* [.30]
NP → *Det Nominal*	[.20]	| *prefer*; [.40]
NP → *Nominal*	[.15]	*Pronoun* → *I* [.40] | *she* [.05]
Nominal → *Noun*	[.75]	| *me* [.15] | *you* [.40]
Nominal → *Nominal Noun*	[.20]	*Proper-Noun* → *Houston* [.60]
Nominal → *Nominal PP*	[.05]	| *NWA* [.40]
VP → *Verb*	[.35]	*Aux* → *does* [.60] | *can* [40]
VP → *Verb NP*	[.20]	*Preposition* → *from* [.30] | *to* [.30]
VP → *Verb NP PP*	[.10]	| *on* [.20] | *near* [.15]
VP → *Verb PP*	[.15]	| *through* [.05]
VP → *Verb NP NP*	[.05]	
VP → *VP PP*	[.15]	
PP → *Preposition NP*	[1.0]	

Probabilistic CFG (PCFG)

A probabilistic CFG is defined as follows (Booth, 1969)

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta\ [p]$, where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$, and p is a number between 0 and 1 expressing $P(\beta \mid A)$
S	a designated **start symbol**

Probabilistic CFG (PCFG)

How are PCFGs used?

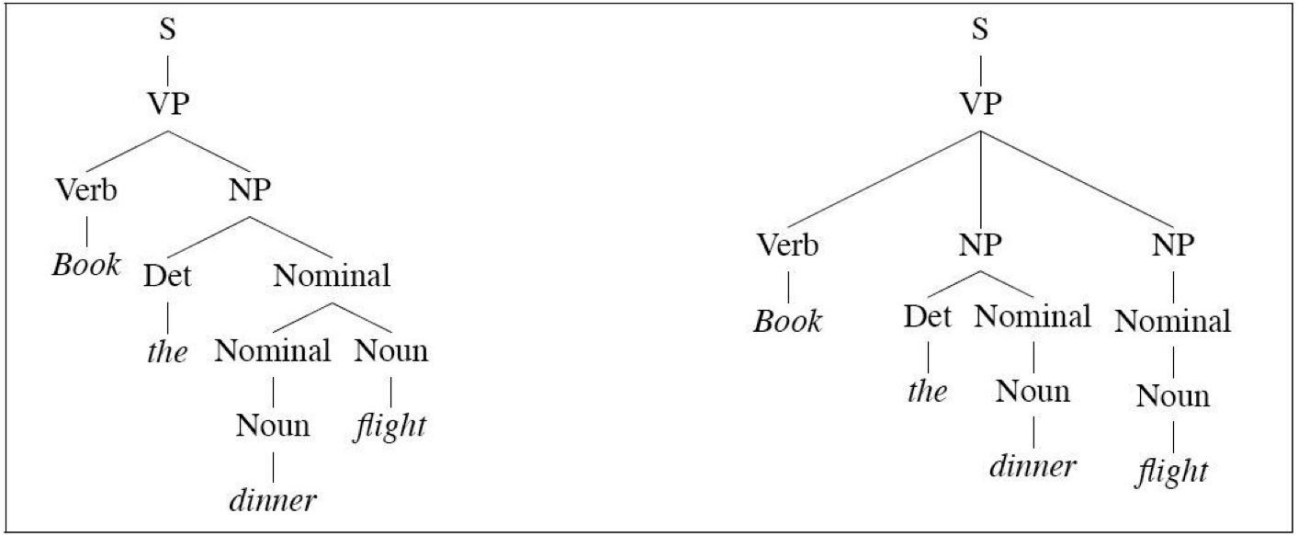
- They are used to estimate the probability of a parse tree (disambiguation)
- They are used to estimate the probability of a sentence or a constituent (language modeling)

Probabilistic CFG (PCFG)

Disambiguation: A PCFG assigns a probability to each parse tree T of a sentence.

The probability of a parse tree is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes (each rule is of the form $\text{LHS}_i \rightarrow \text{RHS}_i$)

$$P(T, S) = \prod_{i=1}^{n} P(\text{RHS}_i | \text{LHS}_i)$$



Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

Probabilistic CFG (PCFG)

Why does it make sense to pick the tree with the highest probability?

$$
\begin{aligned}
T^* &= \text{argmax}_{T \text{ s.t. } S = \text{yield(T)}} P(T|S) \\
&= \text{argmax}_{T \text{ s.t. } S = \text{yield(T)}} \frac{P(S|T)P(T)}{P(S)} \\
&= \text{argmax}_{T \text{ s.t. } S = \text{yield(T)}} P(S|T)P(T) \\
&= \text{argmax}_{T \text{ s.t. } S = \text{yield(T)}} P(T)
\end{aligned}
$$

Probabilistic CFG (PCFG)

Language modeling: A PCFG defines a probability distribution over sentences

$$P(S) = \sum_{T \text{ s.t. } S = \text{yield(T)}} P(T, S)$$

$$= \sum_{T \text{ s.t. } S = \text{yield(T)}} P(T)$$

It can model long range dependencies better than n-grams

the contract ended with a loss of 7 cents after trading as low as 9 cents

Probabilistic CKY

Our probabilistic parsing objective is to produce the most likely parse T for a given sentence S

$$T^* = \text{argmax}_{T \text{ s.t. } S = \text{yield(T)}} P(T)$$

First described by Ney (1991).

Probabilistic CKY

As with deterministic CKY, assume that our probabilistic grammar is in CNF.

➔ Each production is either of the form **A** → **B C** or **A** → **a** for some terminal a.

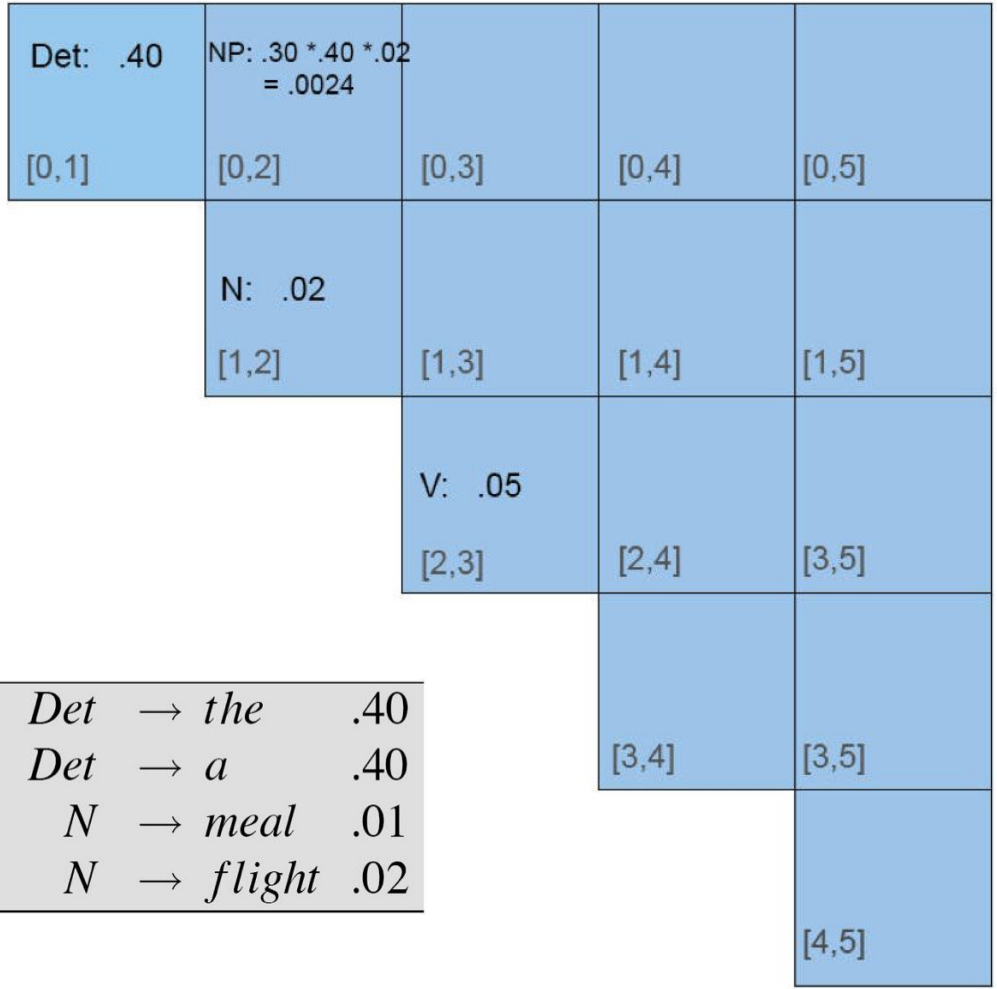
In deterministic CKY each cell **table[i,j]** contained a set of constituents that could span the input words from i to j

$_0$ Book $_1$ the $_2$ flight $_3$ through $_4$ Houston $_5$

In probabilistic CKY each cell **table[i,j,A]** is the maximum probability of a constituent A that spans words i through j.

function PROBABILISTIC-CKY(*words*,*grammar*) **returns** most probable parse
and its probability

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{ A \mid A \rightarrow words[j] \in grammar \}$
    $table[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{ A \mid A \rightarrow BC \in grammar,$
        and $table[i,k,B] > 0$ **and** $table[k,j,C] > 0 \}$
        if $(table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C])$ **then**
          $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
          $back[i,j,A] \leftarrow \{k, B, C\}$
  return BUILD_TREE(*back*[1, LENGTH(*words*), *S*]), *table*[1, LENGTH(*words*), *S*]



S	$\rightarrow$ $NP\ VP$	.80	Det	$\rightarrow$ *the*	.40
NP	$\rightarrow$ $Det\ N$	.30	Det	$\rightarrow$ *a*	.40
VP	$\rightarrow$ $V\ NP$	.20	N	$\rightarrow$ *meal*	.01
V	$\rightarrow$ *includes*	.05	N	$\rightarrow$ *flight*	.02

Learning with PCFG

The simplest way: Use the **Penn Treebank**, a collection of parse trees in English, Chinese and other languages.

$$P(\alpha \rightarrow \beta|\alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

Unsupervised learning is also an option. The **Inside Outside algorithm** (Lari and Young, 1990) is an instance of the expectation maximization framework and a generalization of the Forward Backward algorithm that we used for unsupervised learning of HMMs.

Problems with PCFGs

Poor independence assumptions: PCFGs assume that rule expansions are independent (that's why the probabilities are multiplied).

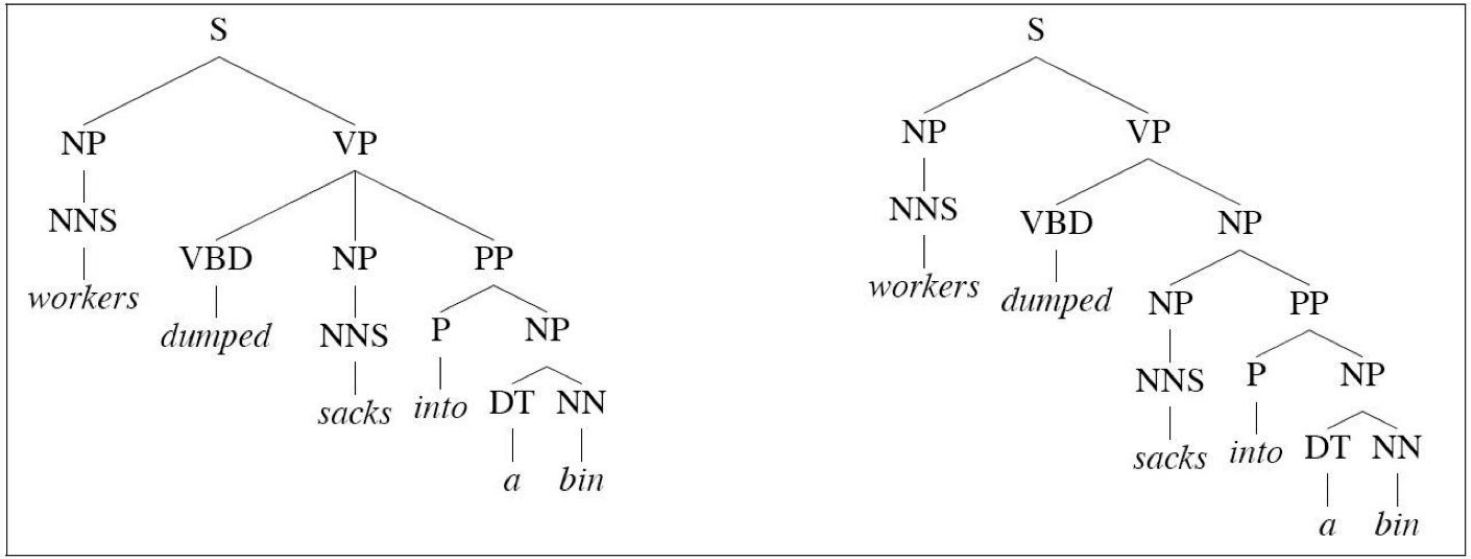
Fact: Subject-NPs are much more likely to be pronouns than object-NPs.

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

But PCFGs are context-free therefore they don't allow a rule probability to be conditioned on surrounding context.

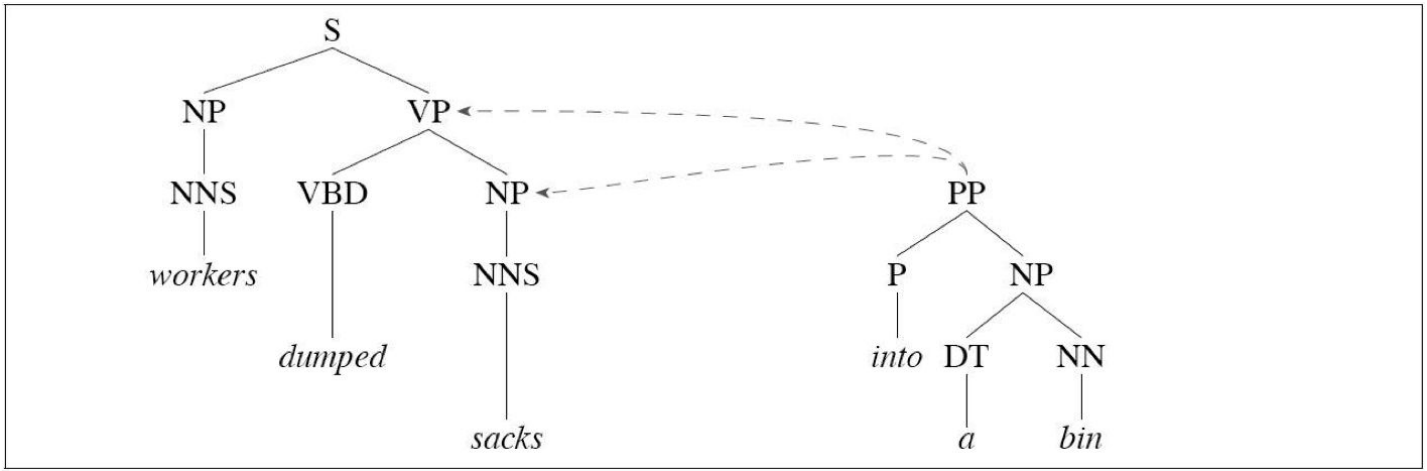
Problems with PCFGs

Lack of lexical conditioning: PCFGs do not model syntactic facts about specific words.



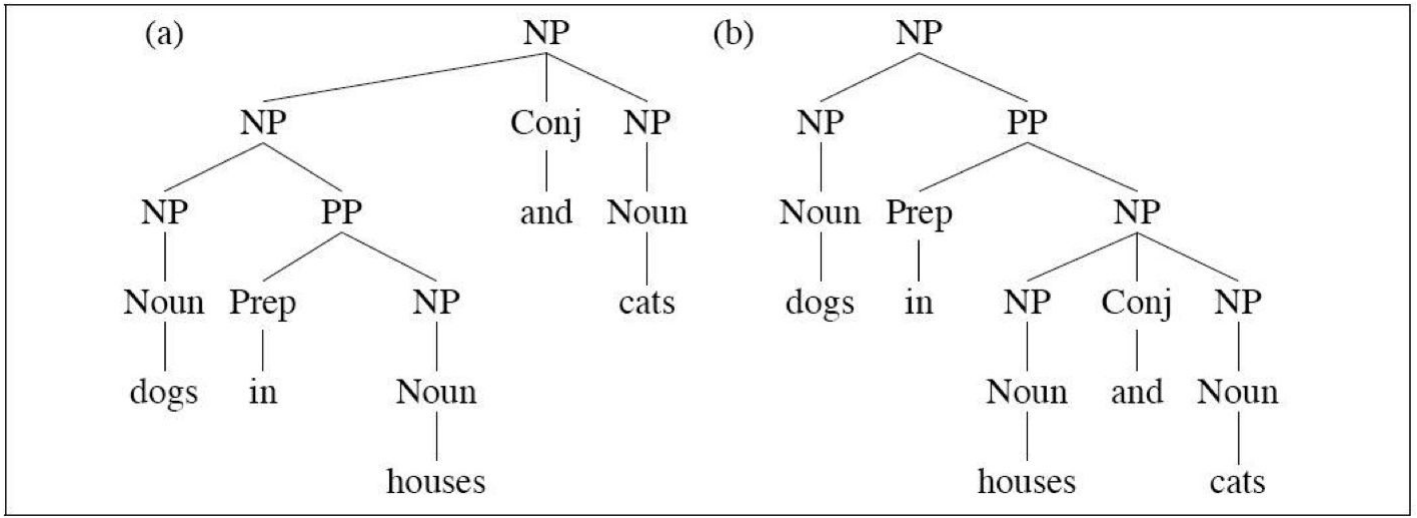
Problems with PCFGs

Lack of lexical conditioning: PCFGs do not model syntactic facts about specific words.



Problems with PCFGs

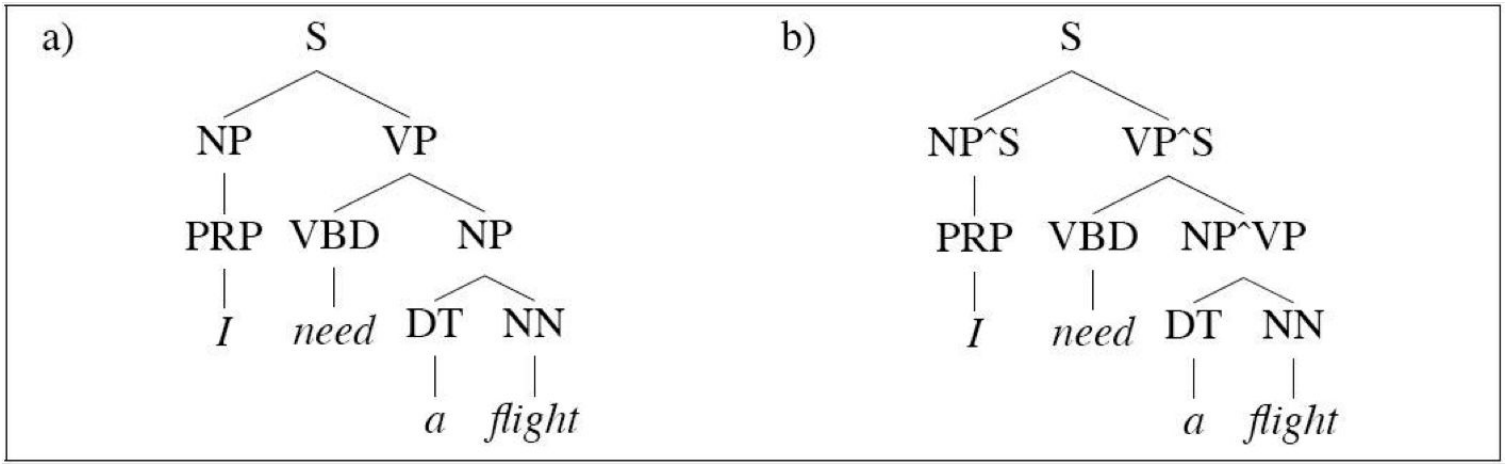
Lack of lexical conditioning: PCFGs do not model syntactic facts about specific words.



Solutions to problems with PCFGs

Nonterminal splitting: $NP_{subject} \rightarrow PRP$ and $NP_{object} \rightarrow PRP$

Parent annotation is one way to achieve this:



Solutions to problems with PCFGs

Nonterminal splitting: $NP_{subject} \rightarrow PRP$ and $NP_{object} \rightarrow PRP$

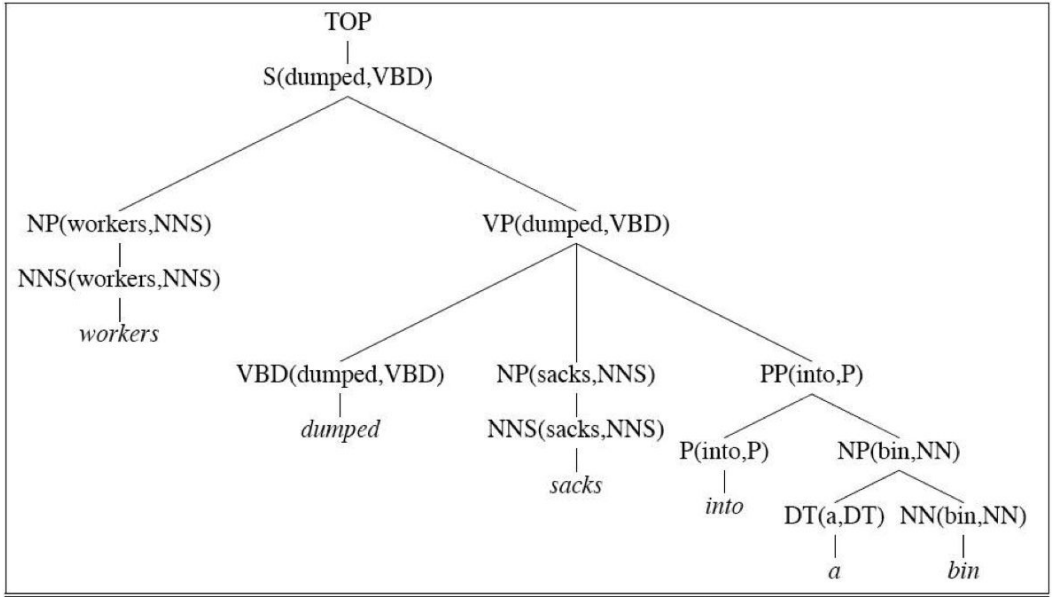
Expectation maximization is another way: The **Split-and-Merge parser** (Petrov et al. 2006) starts with an X-bar grammar and searches for optimal splits and merges of nonterminals that maximize the likelihood of the training set treebank.

Solutions to problems with PCFGs

Lexicalized PCFG:

VP(dumped) →

VP(dumped) NP(sacks) PP(into)



Internal Rules				Lexical Rules		
TOP	→	S(dumped,VBD)		NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)	VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)		NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS) PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)	DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)	NN(bin,NN)	→	bin

Evaluating parsers

PARSEVAL: Measure how much the constituents in the hypothesis parse tree look like the constituents in a gold-standard parse (Black et al, 1991).

Labeled precision: Percentage of constituents predicted that are correct.

Labeled recall: Percentage of gold standard constituents that are predicted correctly.

Cross-brackets: The number of constituents for which the system parse has ((A B) C) but the gold-standard has (A (B C)).

Evaluating parsers

The state-of-the-art parser performance on the Penn treebank is around 90% recall, 90% precision and about 1% cross-bracketed constituents per sentence.

Evalb: The publicly available implementation of PARSEVAL (Sekine and Collins, 1997).

Partial Parsing

Partial parsing

Many NLP tasks do not require whole parse trees for all inputs. For example, in named-entity recognition we might only care about noun phrases.

Chunking: The task of identifying and classifying the non-overlapping segments of a sentence that constitute the basic (non-recursive) phrases, e.g. NP, VP, ADJP, PP.

$$[_{NP} \text{ The morning flight}] [_{PP} \text{ from}] [_{NP} \text{ Denver}] [_{VP} \text{ has arrived}]$$

1. Find the extents of the chunks
2. Label the discovered chunks

Partial parsing

In general not all words have to fall in some chunk. Often we only care about Base-NP constituents.

[$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived

Base-NP constituents are NP constituents that do not contain other NPs.

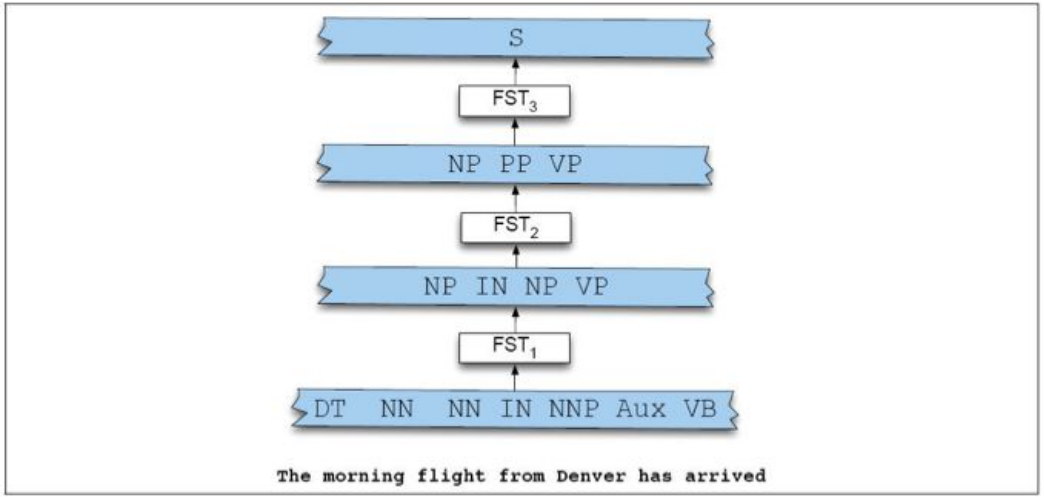
[$_{NP}$ The morning flight Denver] has arrived

Partial parsing

Chunk are non-recursive, therefore they can be modeled using FSTs.

- NP → (DT) NN* NN
- NP → NNP
- VP → VB
- VP → Aux VB
- PP → IN NP
- S → PP* NP PP* VP PP*

Turn each rule into FST



Partial parsing

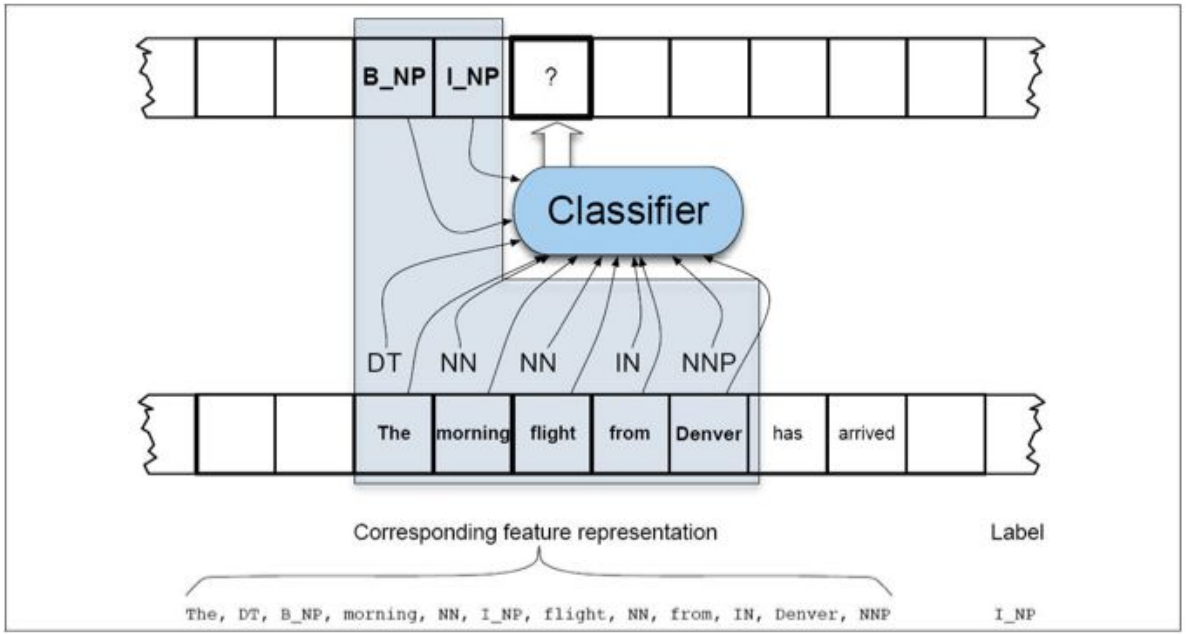
Machine learning approach: Treat chunking as a sequence classification task (like POS tagging) and train on an annotated training set.

IOB tagging: Introduce tags to represent the beginning (B) and internal (I) parts of each chunks, as well as those elements of the input that are outside (O) any chunk.

The	morning	flight	from	Denver	has	arrived
B_{NP}	I_{NP}	I_{NP}	O	B_{NP}	O	O

Partial parsing

At this point any sequence classification technique that we have already seen will work.



Partial parsing evaluation

Word-by-word accuracy metric of POS tagging is not appropriate anymore.

Precision: The percentage of system-provided chunks that were correct (correct = both the boundaries and the label is right).

Recall: The percentage of chunks in the data that were correctly identified by the system.

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

Partial parsing evaluation

F-measure is a way to combine precision and recall.

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The β parameter weights the importance of recall and precision: β > 1 favors recall, β < 1 favors precision.

$$F_1 = \frac{2PR}{P + R}$$

Partial parsing evaluation

Most machine learning systems using chunks such as below achieve F-measures in the .92 to .94 range.

Label	Category	Proportion (%)	Example
NP	Noun Phrase	51	*The most frequently cancelled flight*
VP	Verb Phrase	20	*may not arrive*
PP	Prepositional Phrase	20	*to Houston*
ADVP	Adverbial Phrase	4	*earlier*
SBAR	Subordinate Clause	2	*that*
ADJP	Adjective Phrase	2	*late*