COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Dependency Parsing II

CMSC 470

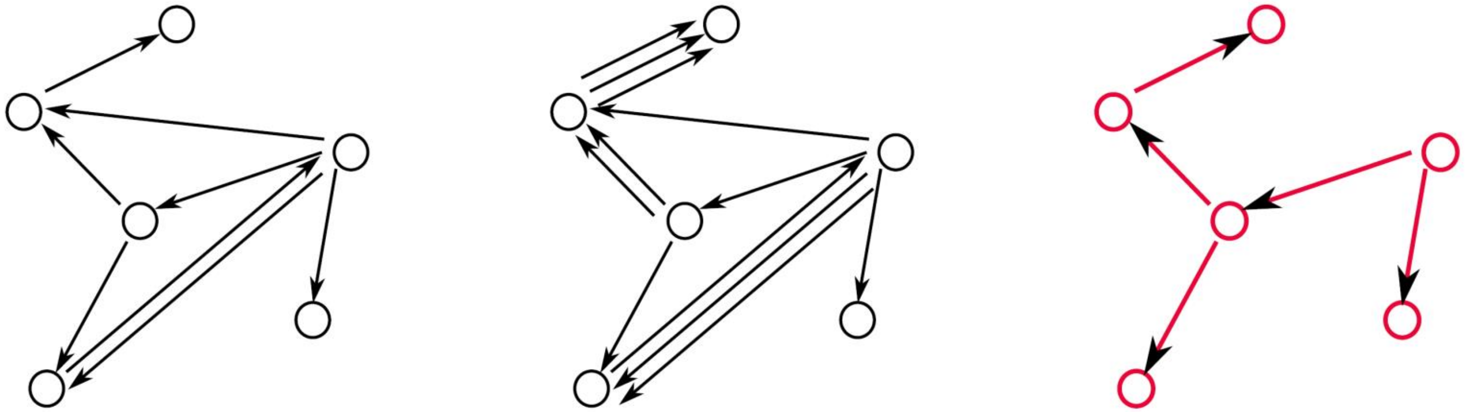
Marine Carpuat

Graph-based Dependency Parsing

Slides credit: Joakim Nivre

Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - G' is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



Dependency Parsing as Finding the Maximum Spanning Tree

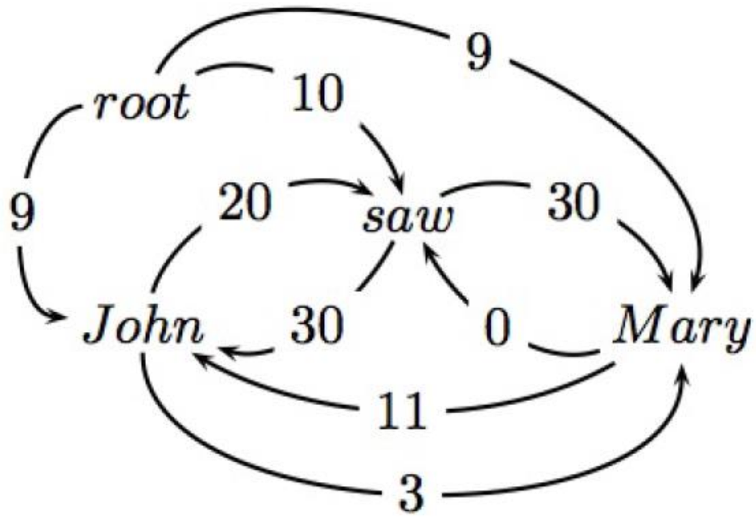
- Views parsing as finding the best directed spanning tree
  - of multi-digraph that captures all possible dependencies in a sentence
  - needs a score that quantifies how good a tree is

- Assume we have an **arc factored** model

  i.e. weight of graph can be factored as sum or product of weights of its arcs

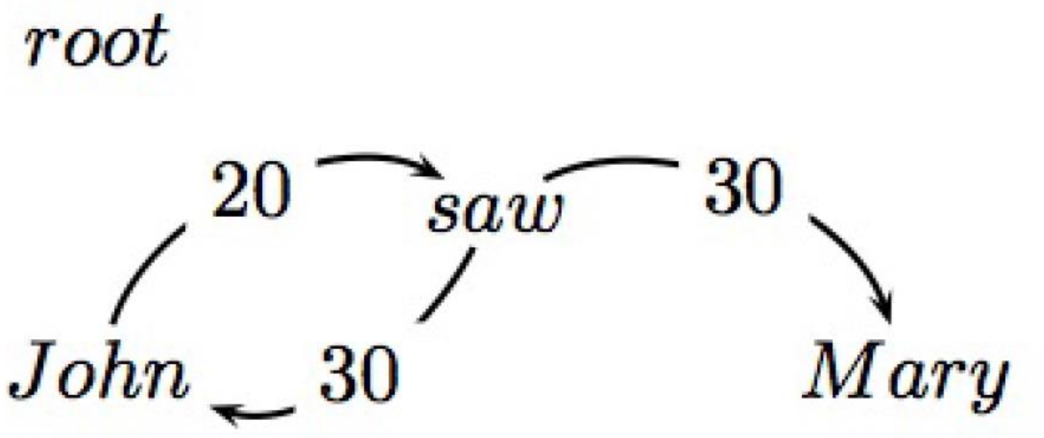
- Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us
  - Recursive algorithm
  - Naïve implementation: O(n^3)

Chu-Liu-Edmonds illustrated
(for unlabeled dependency parsing)



Chu-Liu-Edmonds illustrated

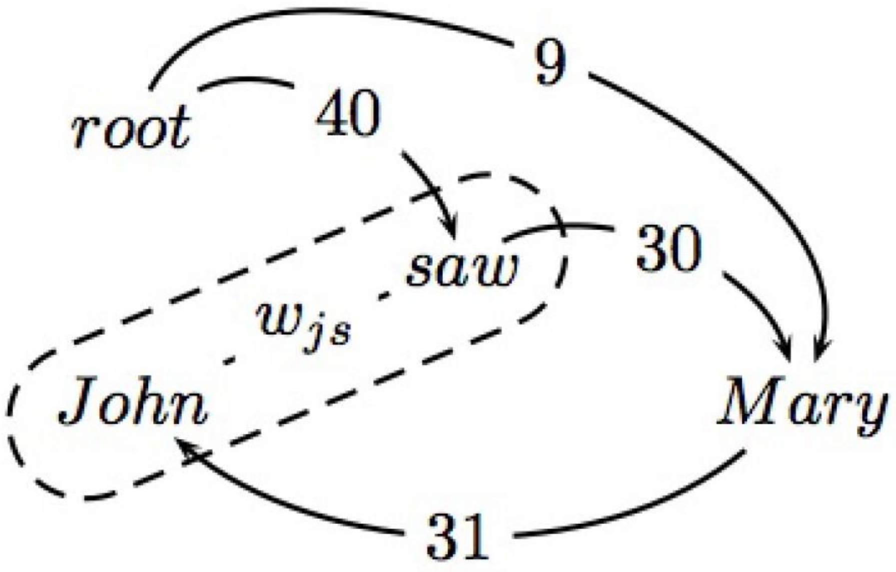
- Find highest scoring incoming arc for each vertex



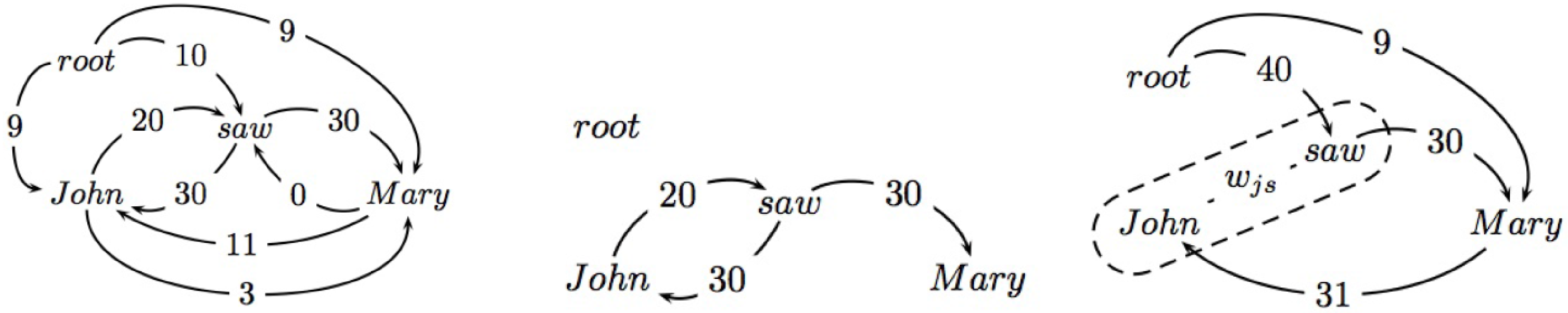
- If this is a tree, then we have found MST!!

Chu-Liu-Edmonds illustrated

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle

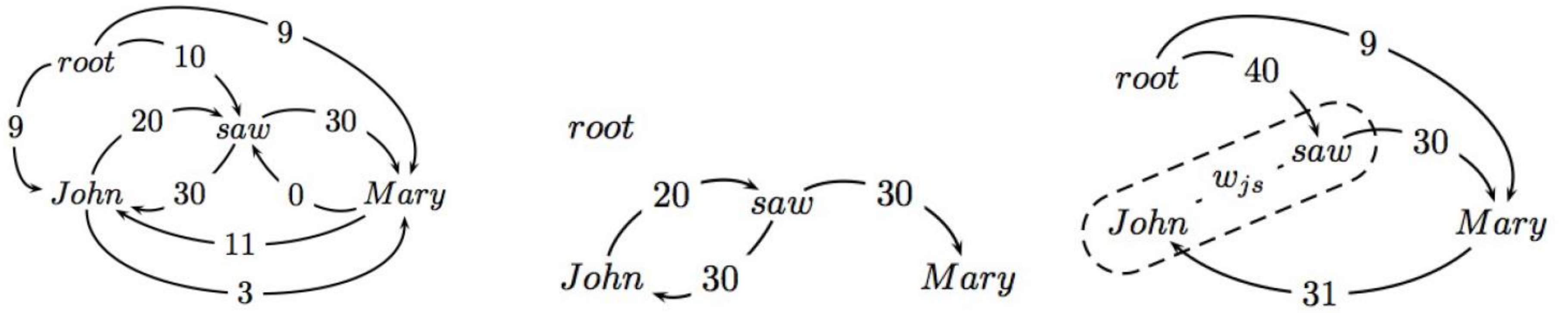


Chu-Liu-Edmonds illustrated



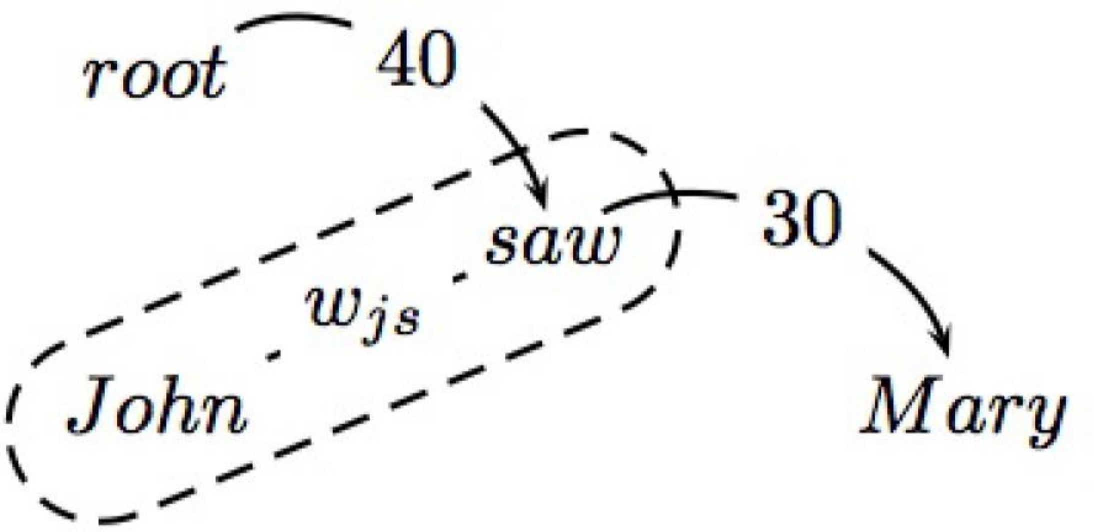
- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John $\rightarrow$ Mary is 3 and saw $\rightarrow$ Mary is 30

Chu-Liu-Edmonds illustrated



- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root $\rightarrow$ saw $\rightarrow$ John is 40 (**)
  - root $\rightarrow$ John $\rightarrow$ saw is 29

- This is a tree and the MST for the contracted graph!!



- Go back up recursive call and reconstruct final graph

Chu-Liu-Edmonds algorithm

```
function MaxSpanningTree(G=(V,E), root, score) returns spanning tree

    F ← []
    T' ← []
    score' ← []
    for each v ∈ V do
      bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
      F ← F ∪ bestInEdge
      for each e=(u,v) ∈ E do
         score'[e] ← score[e] − score[bestInEdge]

      if T=(V,F) is a spanning tree then return it
      else
        C ← a cycle in F
        G' ← Contract(G, C)
        T' ← MaxSpanningTree(G', root, score')
        T ← Expand(T', C)
        return T

function Contract(G, C) returns contracted graph

function Expand(T, C) returns expanded graph
```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

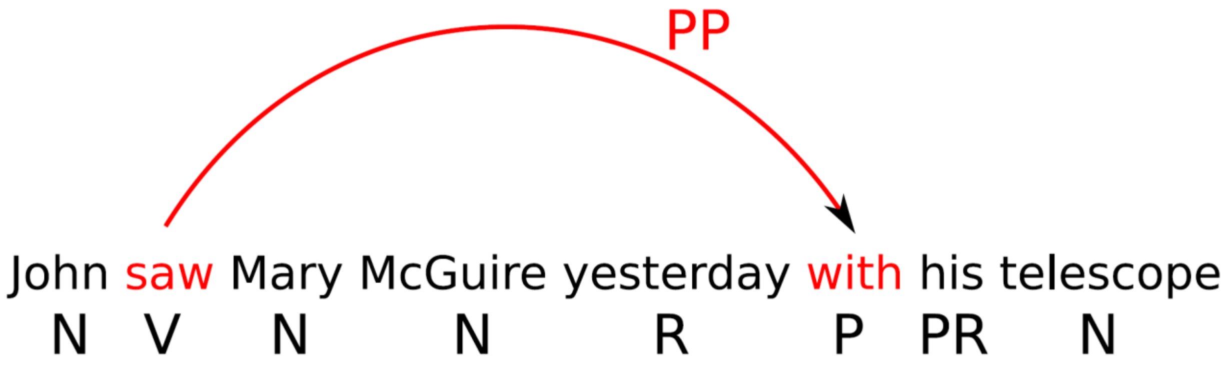
For dependency parsing, we will view arc weights as linear classifiers

Weight of arc from head **i** to dependent **j**, with label **k**

$$w_{ij}^{k} = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- Arc weights are a linear combination of features of the arc, $\mathbf{f}$, and a corresponding weight vector $\mathbf{w}$
- Raised to an exponent (simplifies some math ...)
- What arc features?

Example of classifier features



- Features from [McDonald et al. 2005]:
  - Identities of the words w_i and w_j and the label l_k

head=saw & dependent=with

Typical classifier features

- Word forms, lemmas, and parts of speech of the headword and its dependent
- Corresponding features derived from the contexts before, after and between the words
- Word embeddings
- The dependency relation itself
- The direction of the relation (to the right or left)
- The distance from the head to the dependent
- …

How to score a graph G using features?

Arc-factored model assumption

By definition of arc weights as linear classifiers

$$
\begin{aligned}
G &= \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \arg\max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \arg\max_{G \in T(G_x)} \sum_{(i,j,k) \in G} \mathbf{w} \cdot \mathbf{f}(i,j,k) \\
&= \arg\max_{G \in T(G_x)} \mathbf{w} \cdot \sum_{(i,j,k) \in G} \mathbf{f}(i,j,k) = \arg\max_{G \in T(G_x)} \mathbf{w} \cdot \mathbf{f}(G)
\end{aligned}
$$

Learning parameters with the Structured Perceptron

Training data: $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{|\mathcal{T}|}$

1. $\mathbf{w}^{(0)} = 0;\ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $G' = \arg\max_{G'} \mathbf{w}^{(i)} \cdot \mathbf{f}(G')$
5. if $G' \neq G_t$
6. $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{f}(G_t) - \mathbf{f}(G')$
7. $i = i + 1$
8. return $\mathbf{w}^i$

This is the exact same perceptron algorithm as for multiclass classification, sequence labeling

$$\hat{y} = \text{argmax}_{\hat{y} \in \mathcal{Y}(x)} w \cdot \phi(x, \hat{y})$$

Algorithm 40 STRUCTUREDPERCEPTRONTRAIN(**D**, *MaxIter*)

```
w ← 0                                             // initialize weights
for iter = 1 ... MaxIter do
   for all (x,y) ∈ D do
      ŷ ← argmax_{ŷ∈𝒴(x)} w · φ(x, ŷ)             // compute prediction
      if ŷ ≠ y then
         w ← w + φ(x, y) − φ(x, ŷ)                 // update weights
      end if
   end for
end for
return w                                          // return learned weights
```

Algorithm from CIML chapter 17

Comparing dependency parsing algorithms

Transition-based

- Locally trained
- Use greedy search algorithm
- Can define features over a rich history of parsing decisions

Graph-based

- Globally trained
- Use exact search algorithm
- Can only define features over a limited history of parsing decisions to maintain arc-factored assumption

Dependency Parsing: what you should know

- Interpreting dependency trees
- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition systems: arc standard, arc eager
  - Oracle algorithm: how to obtain a transition sequence given a tree
  - How to construct a multiclass classifier to predict parsing actions
  - What transition-based parsers can and cannot do
  - That transition-based parsers provide a flexible framework that allows many extensions
    - such as RNNs vs feature engineering, non-projectivity (but I don’t expect you to memorize these algorithms)
- Graph-based dependency parsing
  - Chu-Liu-Edmonds algorithm
  - Stuctured perceptron

Parsing with Context Free Grammars

Agenda

- Grammar-based parsing with CFGs
  - CKY algorithm
- Dealing with ambiguity
  - Probabilistic CFGs

Sample Grammar

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

Grammar-based parsing: CKY

Grammar-based Parsing

- Problem setup
  - Input: string **and a CFG**
  - Output: parse tree assigning proper structure to input string
- “Proper structure”
  - Tree that covers all and only words in the input
  - Tree is rooted at an S
  - Derivations obey rules of the grammar
  - Usually, more than one parse tree…

Parsing Algorithms

- Two naive algorithms:
  - Top-down search
  - Bottom-up search
- A “real” algorithm:
  - CKY parsing

Top-Down Search

- Observation
  - trees must be rooted with an S node
- Parsing strategy
  - Start at top with an S node
  - Apply rules to build out trees
  - Work down toward leaves

Bottom-Up Search

- Observation
  - trees must cover all input words

- Parsing strategy
  - Start at the bottom with input words
  - Build structure based on grammar
  - Work up towards the root S

Top-Down vs. Bottom-Up

- Top-down search
  - Only searches valid trees
  - But, considers trees that are not consistent with any of the words
- Bottom-up search
  - Only builds trees consistent with the input
  - But, considers trees that don’t lead anywhere

Parsing as Search

- Search involves controlling choices in the search space
  - Which node to focus on in building structure
  - Which grammar rule to apply
- General strategy: backtracking
  - Make a choice, if it works out then fine
  - If not, back up and make a different choice

Shared Sub-Problems

- Observation
  - ambiguous parses still share sub-trees

- We don’t want to redo work that’s already been done
- Unfortunately, naïve backtracking leads to duplicate work

Efficient Parsing with the CKY (Cocke Kasami Younger) Algorithm

- Solution: Dynamic programming
- Intuition: store partial results in tables
  - Thus avoid repeated work on shared sub-problems
  - Thus efficiently store ambiguous structures with shared sub-parts
- We'll cover one example
  - CKY: roughly, bottom-up

CKY Parsing: CNF

- CKY parsing requires that the grammar consist of binary rules in Chomsky Normal Form
  - All rules of the form:

$$A \rightarrow B\ C$$
$$D \rightarrow w$$

  - What does the tree look like?

CKY Parsing with Arbitrary CFGs

- What if my grammar has rules like $VP \rightarrow NP\ PP\ PP$
  - Problem: can't apply CKY!
  - Solution: rewrite grammar into CNF
    - Introduce new intermediate non-terminals into the grammar

$$A \rightarrow B\ C\ D \quad \Longrightarrow \quad \begin{array}{l} A \rightarrow X\ D \\ X \rightarrow B\ C \end{array}$$

(Where X is a symbol that doesn't occur anywhere else in the grammar)

Sample Grammar

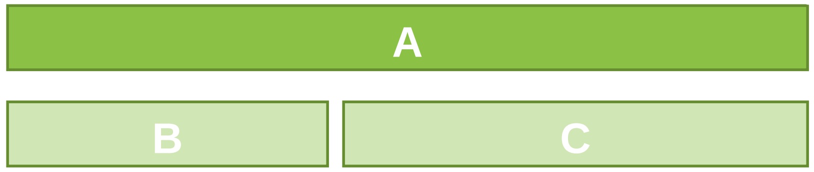
Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP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

CNF Conversion

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

CKY Parsing: Intuition

- Consider the rule D → w
  - Terminal (word) forms a constituent
  - Trivial to apply
- Consider the rule A → B C
  - “If there is an A somewhere in the input, then there must be a B followed by a C in the input”
  - First, precisely define span [*i*, *j*]
  - If A spans from *i* to *j* in the input then there must be some *k* such that *i*<*k*<*j*
  - Easy to apply: we just need to try different values for *k*



CKY Parsing: Table

- Any constituent can conceivably span [*i*, *j*] for all $0 \le i < j \le N$, where *N* = length of input string
  - We need half of an *N* × *N* table to keep track of all spans

- Semantics of table: cell [*i*, *j*] contains A iff A spans *i* to *j* in the input string
  - must be allowed by the grammar!

TO:

FROM:		1	2	3	4	5	6
	0	0–1	0–2	0–3	0–4	0–5	0–6
	1		1–2	1–3	1–4	1–5	1–6
	2			2–3	2–4	2–5	2–6
	3				3–4	3–5	3–6
	4					4–5	4–6
	5						5–6

CKY Parsing: Table-Filling

- In order for A to span [*i*, *j*]
  - A $\rightarrow$ B C is a rule in the grammar, and
  - There must be a B in [*i*, *k*] and a C in [*k*, *j*] for some *i*<*k*<*j*

- Operationally
  - To apply rule A $\rightarrow$ B C, look for a B in [*i*, *k*] and a C in [*k*, *j*]
  - In the table: look left in the row and down in the column

TO:

FROM:	1	2	3	4	5	6
0	0–1	0–2	0–3	0–4	0–5	0–6
1		1–2	1–3	1–4	1–5	1–6
2			2–3	2–4	2–5	2–6
3				3–4	3–5	3–6
4					4–5	4–6
5						5–6

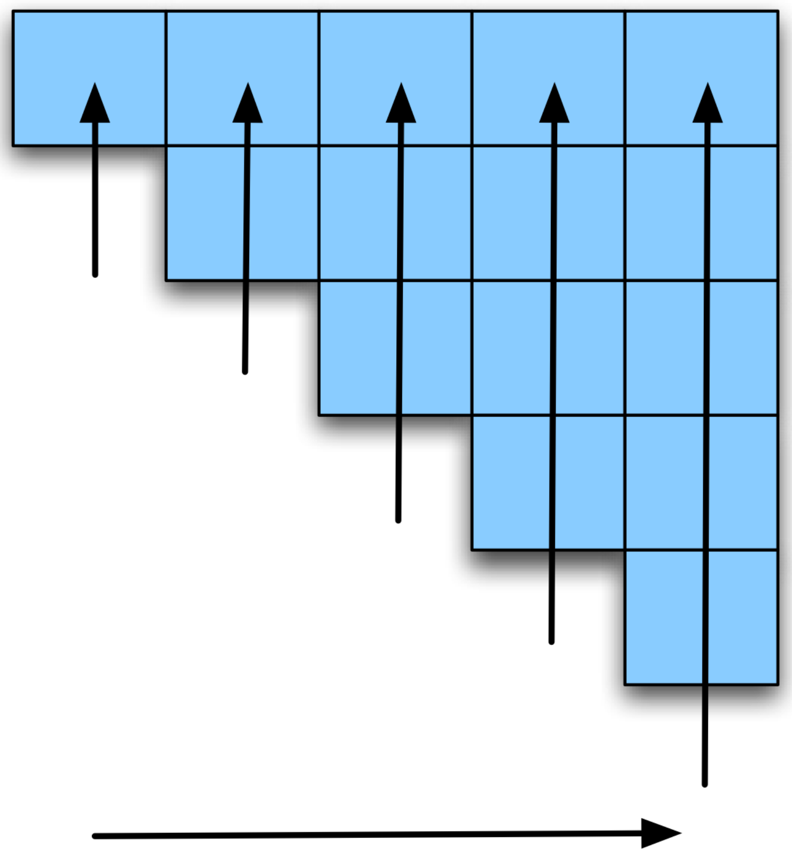
CKY Parsing: Canonical Ordering

- Standard CKY algorithm:
  - Fill the table a column at a time, from left to right, bottom to top
  - Whenever we're filling a cell, the parts needed are already in the table (to the left and below)

- Nice property: processes input left to right, word at a time

CKY Parsing: Ordering Illustrated

Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [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 Algorithm

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

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**

$\boxed{table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}}$

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: Example

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]	? [1,5]
		Nominal, Noun [2,3]	[2,4]	? [2,5]
			Prep [3,4]	? [3,5]
				NP, Proper-Noun [4,5]

Filling column 5

CKY: Example

Recall our CNF grammar:

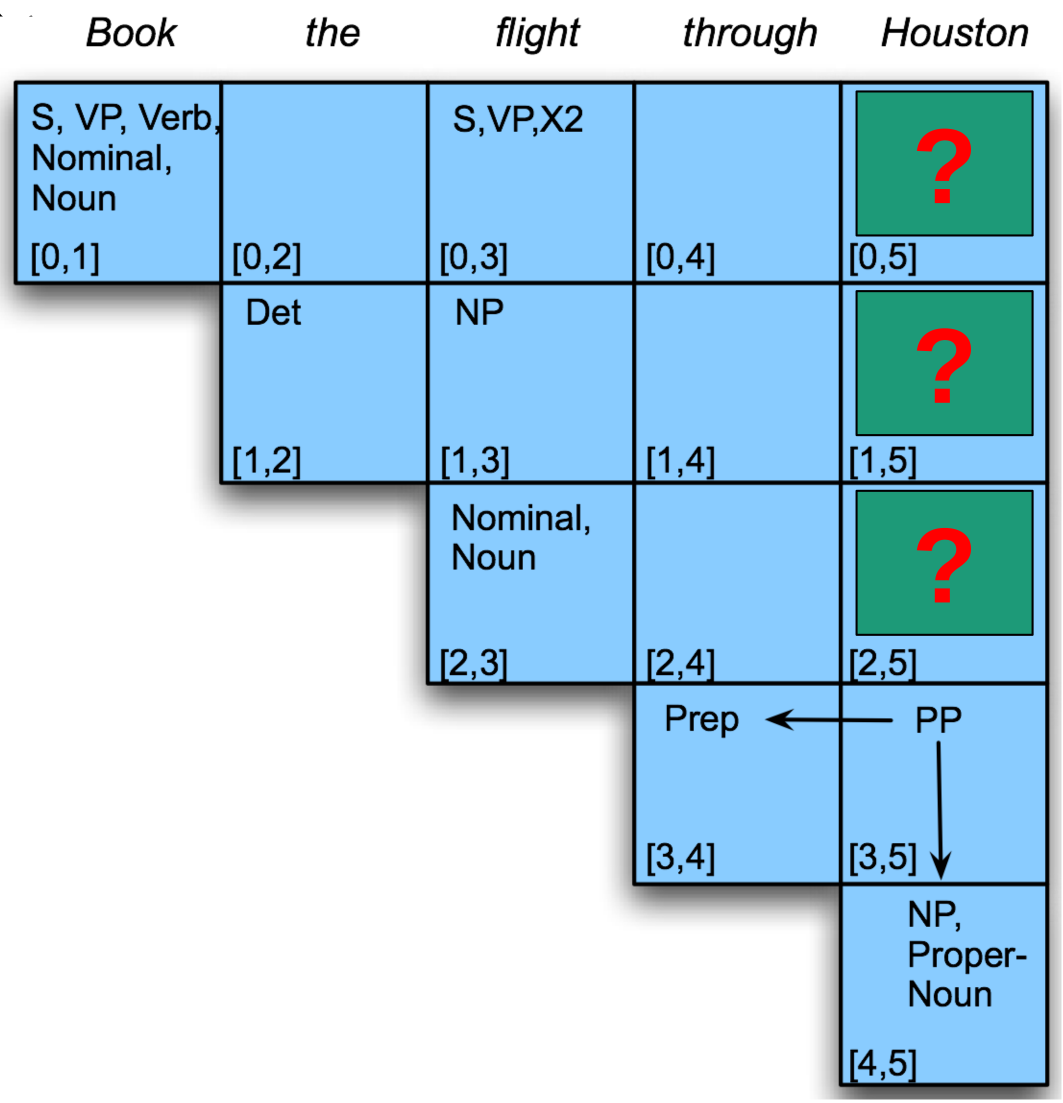
ar	
	$S \rightarrow NP\ VP$
	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
	$NP \rightarrow I \mid she \mid me$
un	$NP \rightarrow TWA \mid Houston$
al	$NP \rightarrow Det\ Nominal$
	$Nominal \rightarrow book \mid flight \mid meal \mid money$
al Noun	$Nominal \rightarrow Nominal\ Noun$
al PP	$Nominal \rightarrow Nominal\ PP$
	$VP \rightarrow book \mid include \mid prefer$
	$VP \rightarrow Verb\ NP$
	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
	$VP \rightarrow Verb\ PP$
	$VP \rightarrow VP\ PP$
NP	$PP \rightarrow Preposition\ NP$

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]	? [1,5]
		Nominal, Noun [2,3]	[2,4]	? [2,5]
			Prep [3,4]	? [3,5]
				NP, Proper-Noun [4,5]

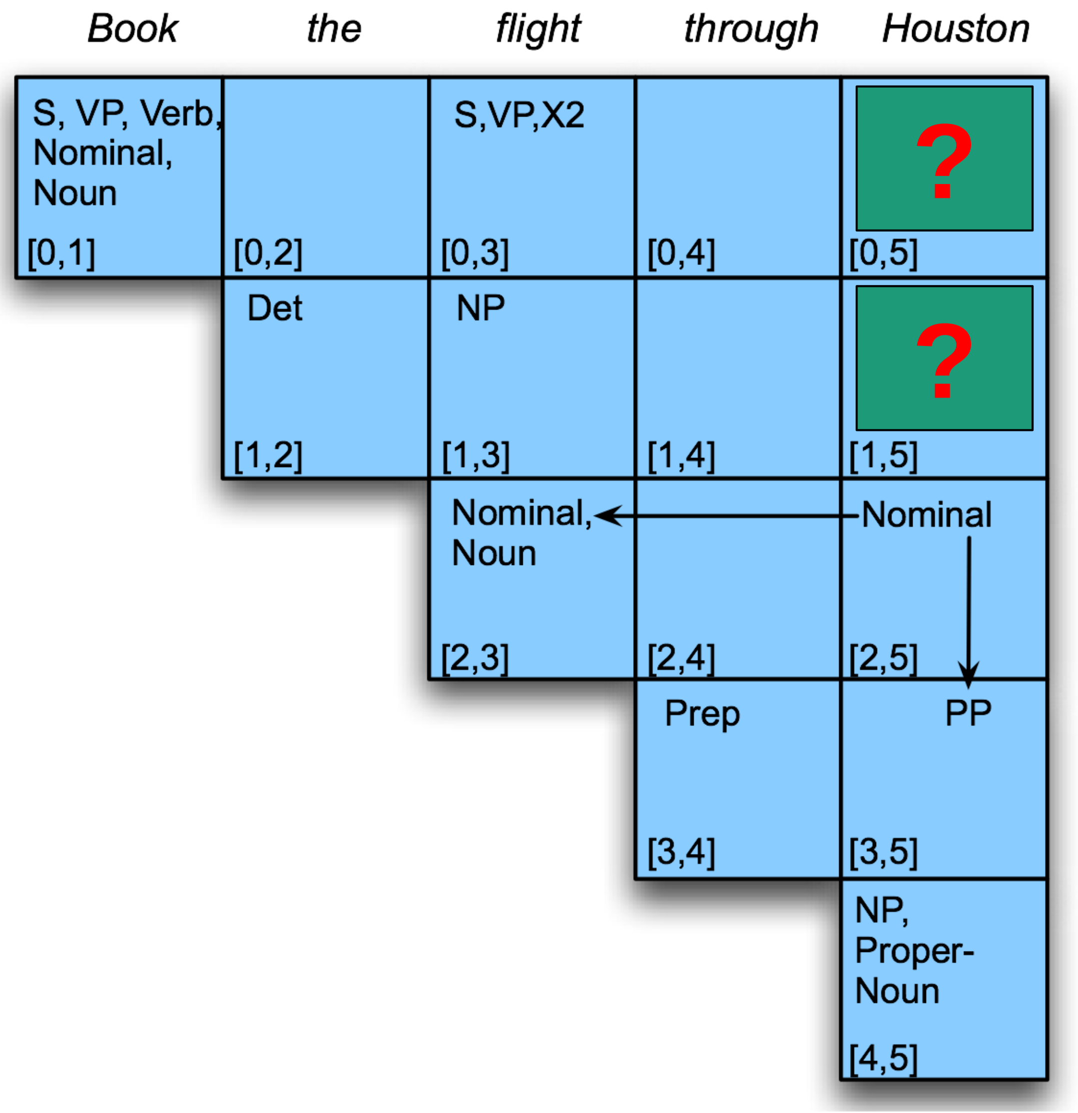
CKY: Example

Recall our CNF grammar:

S → *NP VP*
S → *X1 VP*
X1 → *Aux NP*
S → *book* | *include* | *prefer*
S → *Verb NP*
S → *X2 PP*
S → *Verb PP*
S → *VP PP*
NP → *I* | *she* | *me*
NP → *TWA* | *Houston*
NP → *Det Nominal*
Nominal → *book* | *flight* | *meal* | *money*
Nominal → *Nominal Noun*
Nominal → *Nominal PP*
VP → *book* | *include* | *prefer*
VP → *Verb NP*
VP → *X2 PP*
X2 → *Verb NP*
VP → *Verb PP*
VP → *VP PP*
PP → *Preposition NP*



CKY: Example

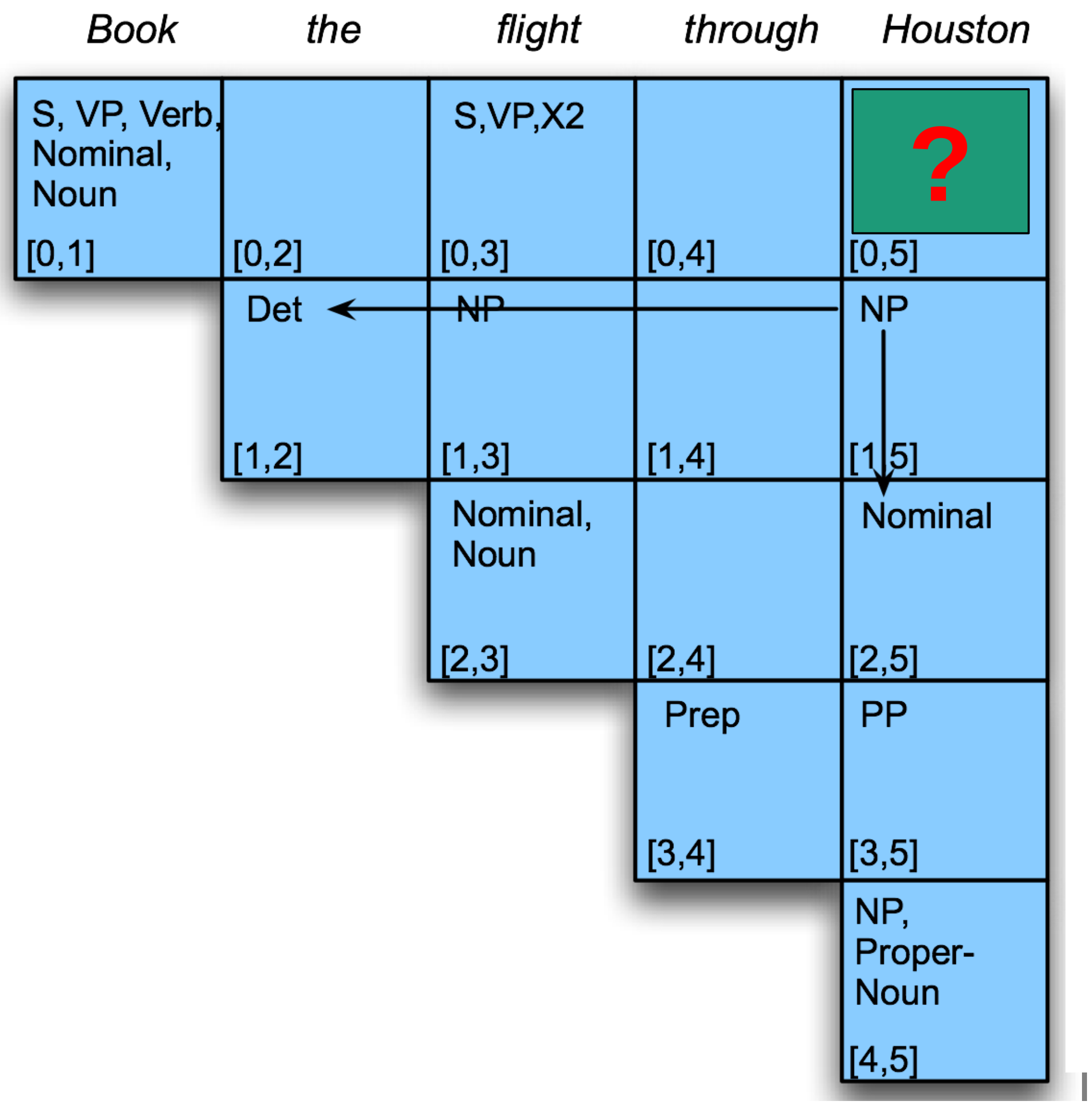


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
?
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
?
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

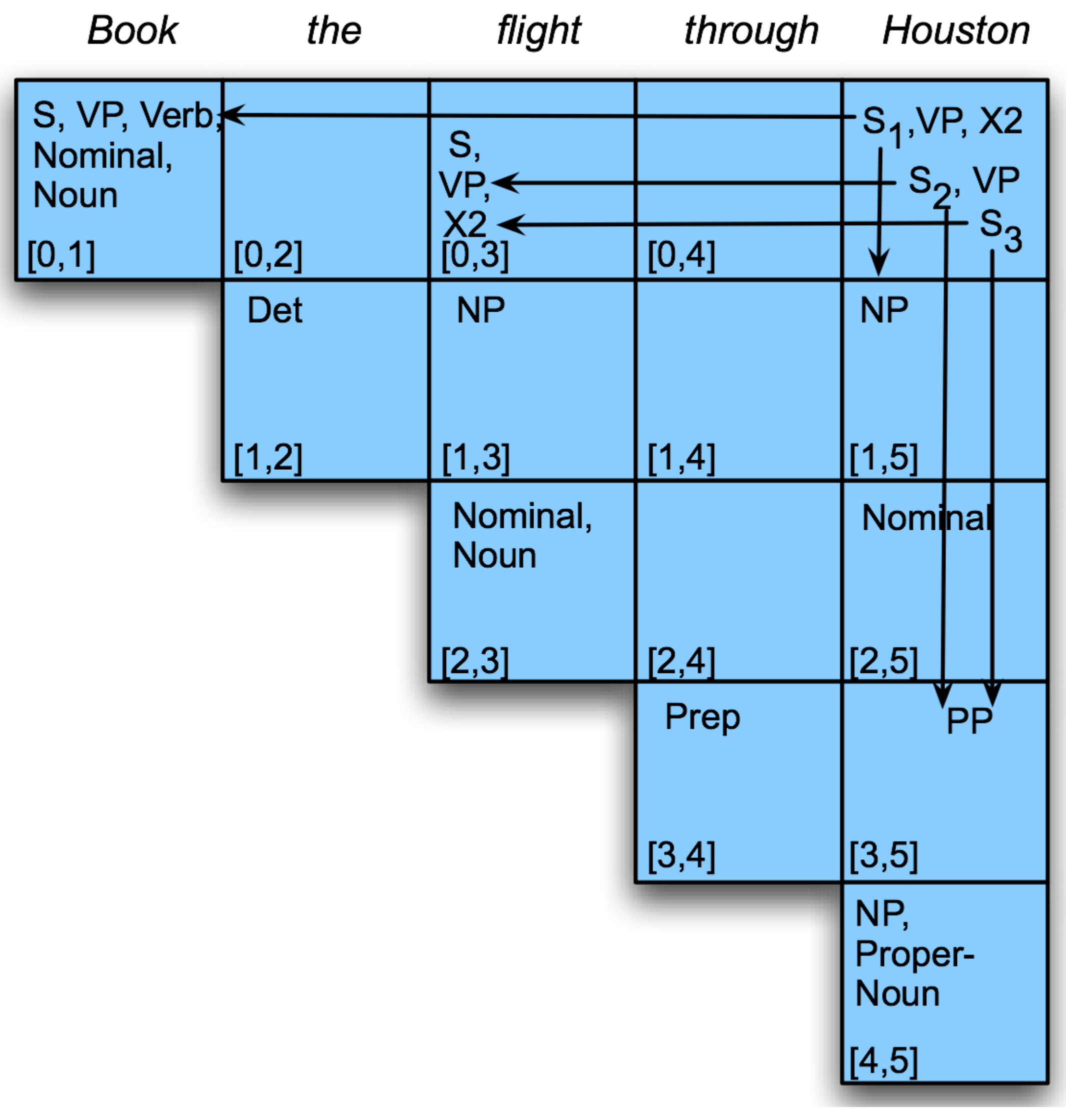
CKY: Example

Recall our CNF grammar:

$S \rightarrow NP\ VP$
$S \rightarrow X1\ VP$
$X1 \rightarrow Aux\ NP$
$S \rightarrow book \mid include \mid prefer$
$S \rightarrow Verb\ NP$
$S \rightarrow X2\ PP$
$S \rightarrow Verb\ PP$
$S \rightarrow VP\ PP$
$NP \rightarrow I \mid she \mid me$
$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$
$VP \rightarrow X2\ PP$
$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$



CKY: Example



Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S, VP, X2
[0,3]
[0,4]
S1,VP, X2
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 Parsing: Recognize or Parse

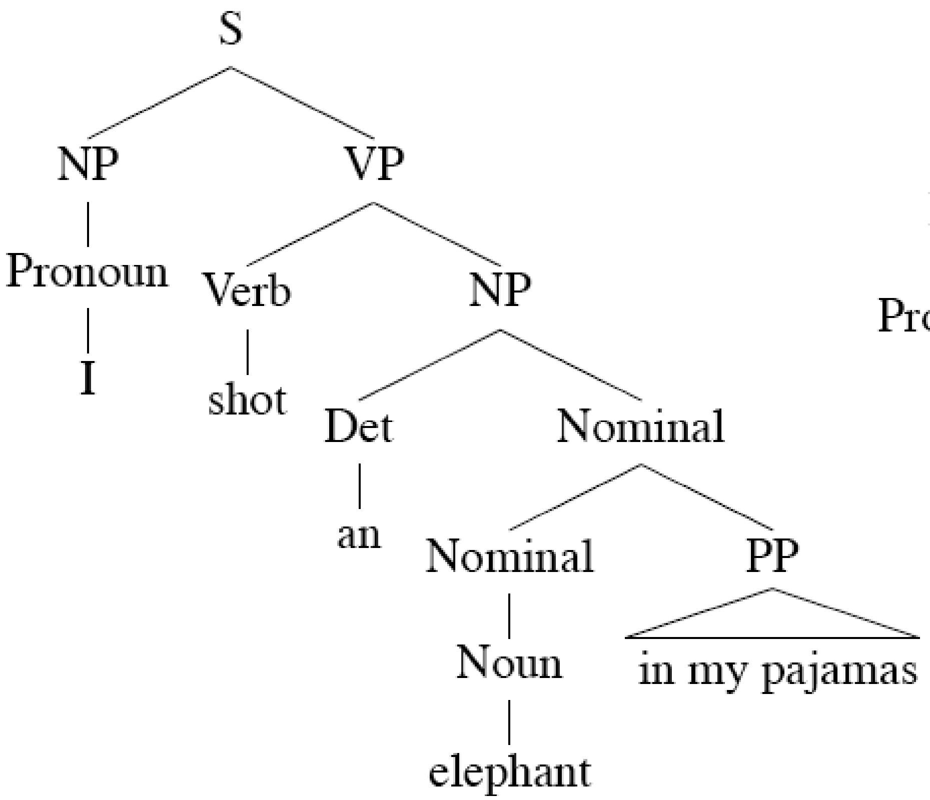
- Recognizer
  - Output is binary
  - Can the complete span of the sentence be covered by an S symbol?

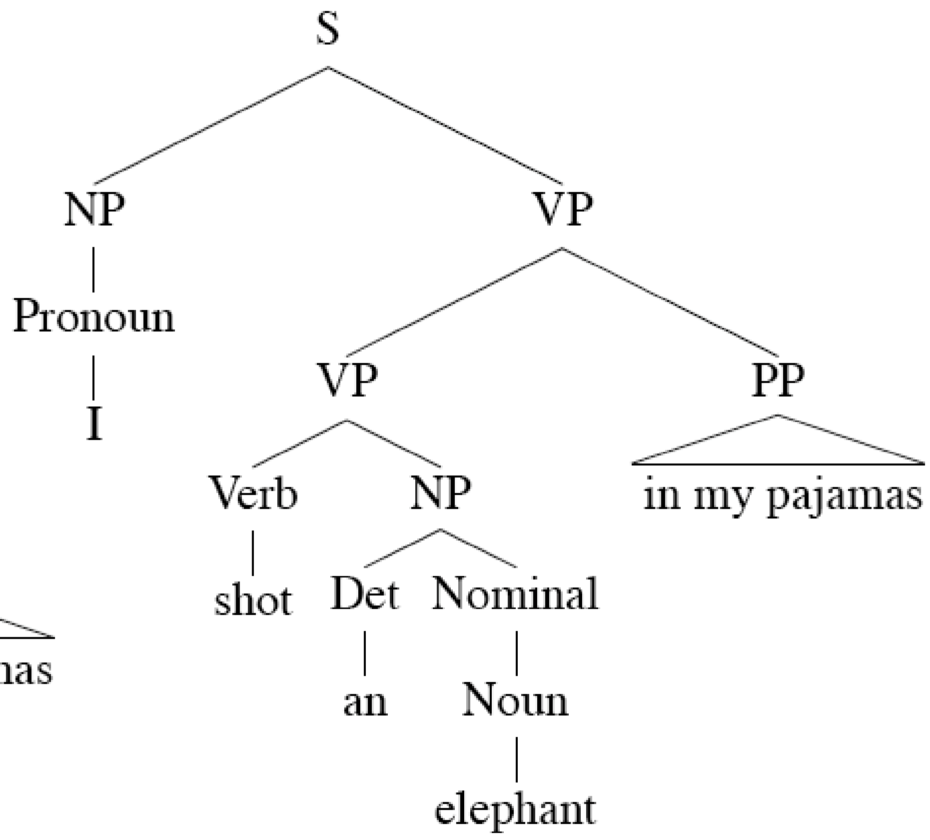
- Parser
  - Output is a parse tree
  - From recognizer to parser: add backpointers!

Ambiguity

- CKY can return multiple parse trees
  - Plus: compact encoding with shared sub-trees
  - Plus: work deriving shared sub-trees is reused
  - Minus: algorithm doesn’t tell us which parse is correct!

Ambiguity


S
NP
VP
Pronoun
Verb
NP
I
shot
Det
Nominal
an
Nominal
PP
Noun
in my pajamas
elephant


S
NP
VP
Pronoun
VP
PP
I
Verb
NP
in my pajamas
shot
Det
Nominal
an
Noun
elephant

PROBABILISTIC Context-free grammars

Simple Probability Model

- A derivation (tree) consists of the bag of grammar rules that are in the tree
  - The probability of a tree is the product of the probabilities of the rules in the derivation.

$$P(T,S) = \prod_{node \in T} P(rule(n))$$

Rule Probabilities

- What's the probability of a rule?
- Start at the top...
  - A tree should have an *S* at the top. So given that we know we need an *S*, we can ask about the probability of each particular *S* rule in the grammar: P(particular rule | S)
- In general we need
  for each rule in the grammar $$P(\alpha \rightarrow \beta \mid \alpha)$$

Training the Model

- We can get the estimates we need from a treebank

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

For example, to get the probability for a particular *VP* rule:

1. count all the times the rule is used
2. divide by the number of *VP*s overall.

Parsing (Decoding)

How can we get the best (most probable) parse for a given input?

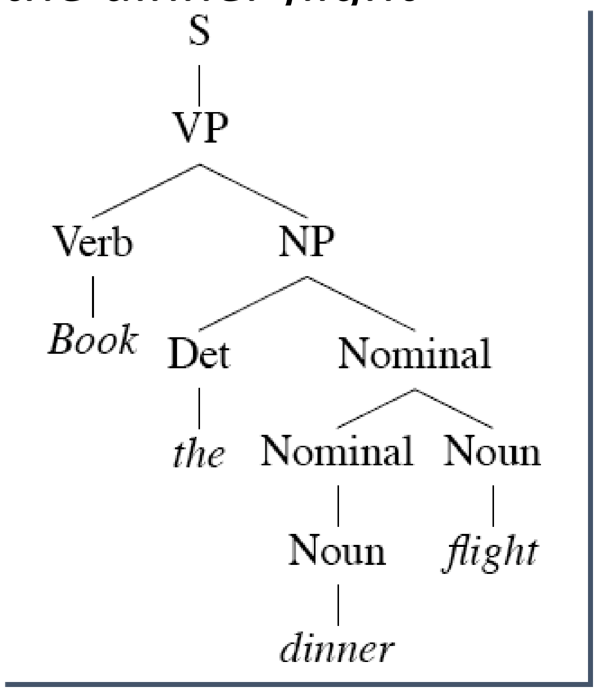
1. Enumerate all the trees for a sentence

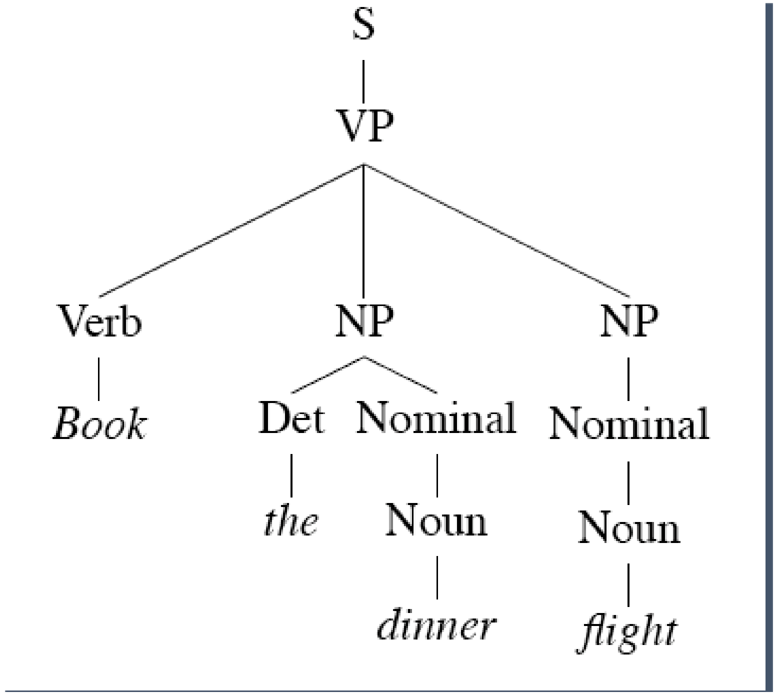
2. Assign a probability to each using the model

3. Return the argmax

Example

- Consider...
  - *Book the dinner flight*





Examples

- These trees consist of the following rules.

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

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$

$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

Dynamic Programming

- Of course, as with normal parsing we don’t really want to do it that way...
- Instead, we need to exploit dynamic programming
  - For the parsing (as with CKY)
  - And for computing the probabilities and returning the best parse (as with Viterbi)

Probabilistic CKY

- Store probabilities of constituents in the table
  - table[i,j,A] = probability of constituent A that spans positions i through j in input

- If A is derived from the rule $A \rightarrow B\ C$:
  - table[i,j,A] = P($A \rightarrow B\ C$ | A) * table[i,k,B] * table[k,j,C]
  - Where
    - P($A \rightarrow B\ C$ | A) is the rule probability
    - table[i,k,B] and table[k,j,C] are already in the table given the way that CKY operates

- Only store the MAX probability over all the A rules.

Probabilistic CKY

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                               and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
        table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
              for all { A | A → BC ∈ grammar,
                          and table[i,k,B] > 0 and table[k,j,C] > 0 }
                    if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                       table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                       back[i,j,A] ← {k,B,C}
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Grammar-based parsing with CFGs summary

- CKY algorithm finds all the parses of a given sentence efficiently
  - Using dynamic programming
- Probabilistic CFGs help deal with ambiguity
  - Requires computing probability of rules based on their frequency in the training data
- Lexicalized grammars help improve performance further