Chart Parsing: The Horrific CYK Algorithm

Informatics 2A: Lecture 20

Shay Cohen

31 October 2017

Last Class

Constituents and Phrases: A phrase inherits the category of its head.

Ambiguity: A sentence can have multiple parse trees (or multiple POS analyses)

Recursive descent Parsing: a top-down parser. Iterate through all rules, in a depth-first style, until a parse that matches the sentence is found. Exhuastive search, essentially.

Shift-Reduce Parsing: Two operations – SHIFT: put a word with its POS tag on the stack. REDUCE: Take a sequence of top symbols on the stack and pop them if they match with a right-hand side of a rule, and then place the left-hand side on the top of the stack.

Why not LL(1)? We have ambiguous grammars!

The CYK Algorithm

1. Problems with Parsing as Search
    - Grammar Restructuring
    - Problems with Parsing as Search

2. The CYK Algorithm
    - Parsing as Dynamic Programming
    - The CYK Algorithm
    - Properties of the Algorithm

The CYK Algorithm

The CKY Algorithm

Grammar Restructuring

Deterministic parsing (e.g., LL(1)) aims to address a limited amount of local ambiguity – the problem of not being able to decide uniquely which grammar rule to use next in a left-to-right analysis of the input string.

By re-structuring the grammar, the parser can make a unique decision, based on a limited amount of look-ahead.

Recursive Descent parsing also demands grammar restructuring, in order to eliminate left-recursive rules that can get it into a hopeless loop.

Left Recursion

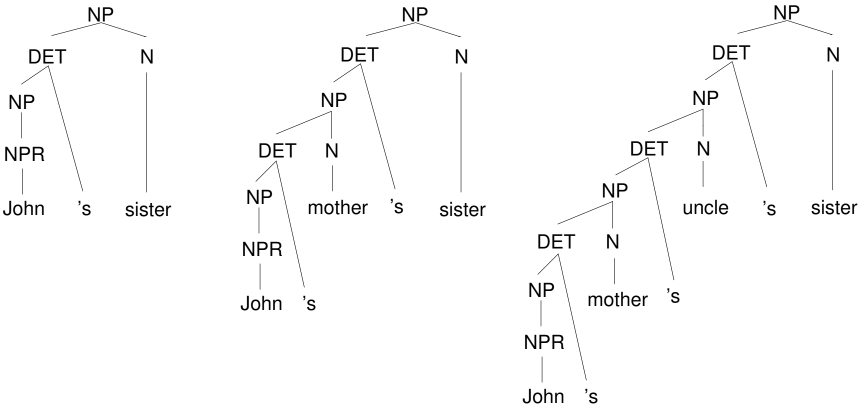
But grammars for natural human languages should be revealing, re-structuring the grammar may destroy this. (Indirectly) left-recursive rules are needed in English.

NP → DET N
NP → NPR
DET → NP 's

These rules generate NPs with possessive modifiers such as:

John's sister
John's mother's sister
John's mother's uncle's sister
John's mother's uncle's sister's niece

Left Recursion



We don't want to re-structure our grammar rules just to be able to use a particular approach to parsing. Need an alternative.

How many parses are there?

If our grammar is ambiguous (inherently, or by design) then how many possible parses are there?

How many parses are there?

If our grammar is ambiguous (inherently, or by design) then how many possible parses are there?

In general: an infinite number, if we allow unary recursion.

How many parses are there?

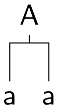
If our grammar is ambiguous (inherently, or by design) then how many possible parses are there?

In general: an infinite number, if we allow unary recursion.

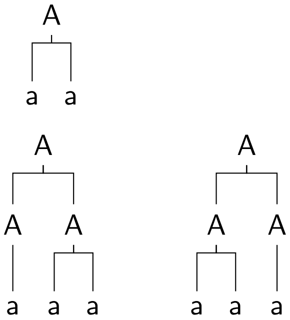
More specific: suppose that we have a grammar in Chomsky normal form. How many possible parses are there for a sentence of n words? Imagine that every nonterminal can rewrite as every pair of nonterminals (A$\rightarrow$BC) and every nonterminal (A$\rightarrow$a)

1. n
2. n^2
3. $n \log n$
4. $\frac{(2n)!}{(n+1)!n!}$

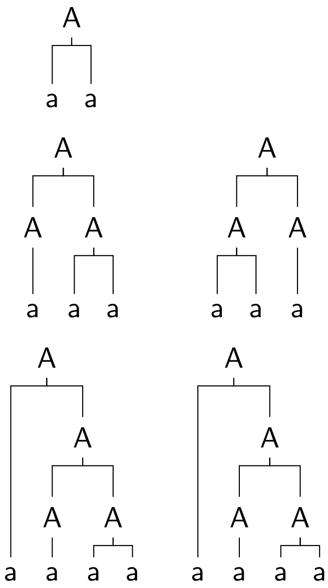
How many parses are there?



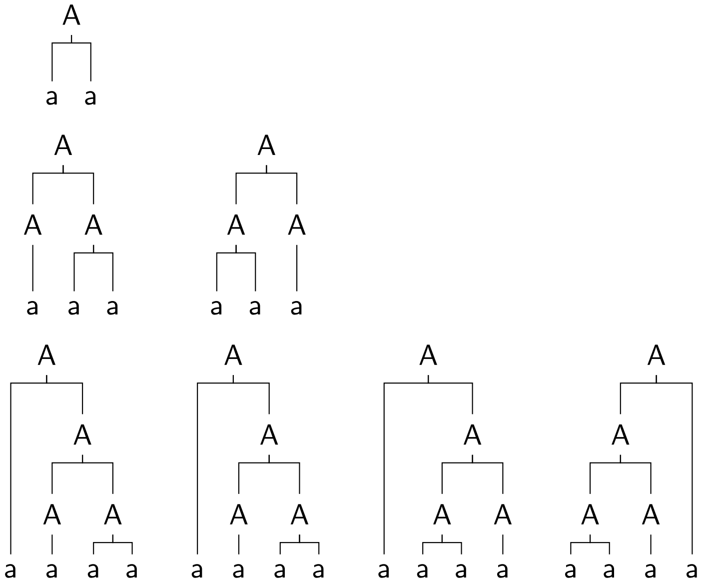
How many parses are there?



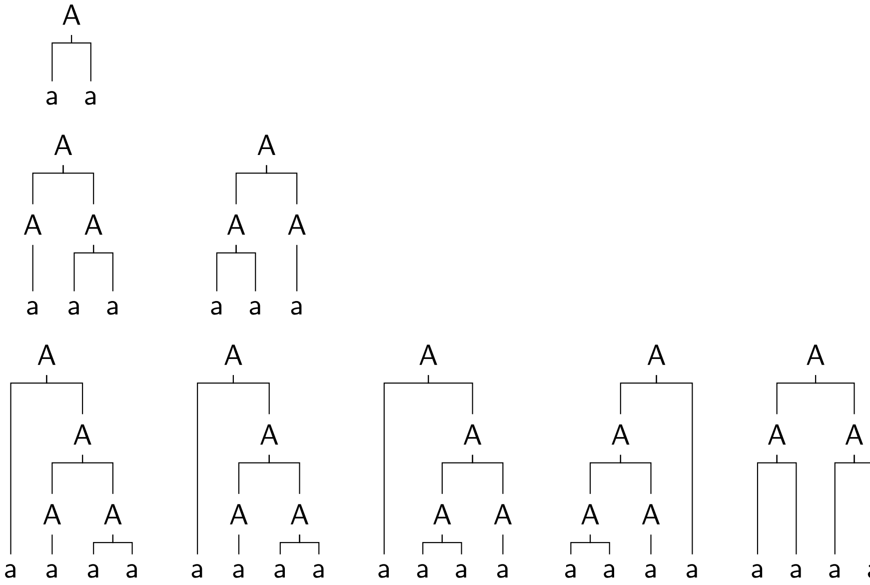
How many parses are there?



How many parses are there?



How many parses are there?



How many parses are there?

Intution. Let $C(n)$ be the number of binary trees over a sentence of length n. The root of this tree has two subtrees: one over k words ($1 \leq k < n$), and one over $n - k$ words. Hence, for all values of k, we can combine any subtree over k words with any subtree over $n - k$ words:

$$C(n) = \sum_{k=1}^{n-1} C(k) \times C(n-k)$$

How many parses are there?

Intution. Let $C(n)$ be the number of binary trees over a sentence of length n. The root of this tree has two subtrees: one over k words ($1 \leq k < n$), and one over $n - k$ words. Hence, for all values of k, we can combine any subtree over k words with any subtree over $n - k$ words:

$$C(n) = \sum_{k=1}^{n-1} C(k) \times C(n-k)$$

$$C(n) = \frac{(2n)!}{(n+1)!n!}$$

These numbers are called the Catalan numbers. They're big numbers!

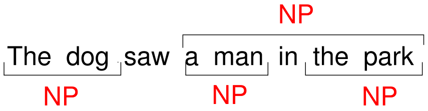
n	1	2	3	4	5	6	8	9	10	11	12
$C(n)$	1	1	2	5	14	42	132	429	1430	4862	16796

Problems with Parsing as Search

1. A **recursive descent parser** (top-down) will do badly if there are many different rules for the same LHS. Hopeless for rewriting parts of speech (preterminals) with words (terminals).
2. A **shift-reduce parser** (bottom-up) does a lot of useless work: many phrase structures will be locally possible, but globally impossible. Also inefficient when there is much lexical ambiguity.
3. Both strategies do repeated work by re-analyzing the same substring many times.

We will see how chart parsing solves the re-parsing problem, and also copes well with ambiguity.

With a CFG, a parser should be able to avoid re-analyzing sub-strings because the analysis of any sub-string is independent of the rest of the parse.



The parser's exploration of its search space can exploit this independence if the parser uses dynamic programming.

Dynamic programming is the basis for all chart parsing algorithms.

Parsing as Dynamic Programming

- Given a problem, systematically fill a table of solutions to sub-problems: this is called memoization.
- Once solutions to all sub-problems have been accumulated, solve the overall problem by composing them.
- For parsing, the sub-problems are analyses of sub-strings and correspond to constituents that have been found.
- Sub-trees are stored in a chart (aka well-formed substring table), which is a record of all the substructures that have ever been built during the parse.

Solves **re-parsing problem**: sub-trees are looked up, not re-parsed!
Solves **ambiguity problem**: chart implicitly stores all parses!

Depicting a Chart

A chart can be depicted as a matrix:

- Rows and columns of the matrix correspond to the start and end positions of a span (ie, starting right before the first word, ending right after the final one);
- A cell in the matrix corresponds to the sub-string that starts at the row index and ends at the column index.
- It can contain information about the type of constituent (or constituents) that span(s) the substring, pointers to its sub-constituents, and/or predictions about what constituents might follow the substring.

CYK Algorithm

CYK (Cocke, Younger, Kasami) is an algorithm for recognizing and recording constituents in the chart.

- Assumes that the grammar is in Chomsky Normal Form: rules all have form $A \rightarrow BC$ or $A \rightarrow w$.
- Conversion to CNF can be done automatically.

NP	$\rightarrow$	Det Nom	NP	$\rightarrow$	Det Nom
Nom	$\rightarrow$	N | OptAP Nom	Nom	$\rightarrow$	*book* | *orange* | AP Nom
OptAP	$\rightarrow$	ϵ | OptAdv A	AP	$\rightarrow$	*heavy* | *orange* | Adv A
A	$\rightarrow$	*heavy* | *orange*	A	$\rightarrow$	*heavy* | *orange*
Det	$\rightarrow$	*a*	Det	$\rightarrow$	*a*
OptAdv	$\rightarrow$	ϵ | *very*	Adv	$\rightarrow$	*very*
N	$\rightarrow$	*book* | *orange*			

CYK: an example

Let's look at a simple example before we explain the general case.

Grammar Rules in CNF

NP	$\rightarrow$	Det Nom
Nom	$\rightarrow$	*book* | *orange* | AP Nom
AP	$\rightarrow$	*heavy* | *orange* | Adv A
A	$\rightarrow$	*heavy* | *orange*
Det	$\rightarrow$	*a*
Adv	$\rightarrow$	*very*

(N.B. Converting to CNF sometimes breeds duplication!)

Now let's parse: *a very heavy orange book*

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a					
1	very					
2	heavy					
3	orange					
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very					
2	heavy					
3	orange					
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv			
2	heavy					
3	orange					
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv			
2	heavy			A,AP		
3	orange					
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv	AP		
2	heavy			A,AP		
3	orange					
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv	AP		
2	heavy			A,AP		
3	orange				Nom,A,AP	
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv	AP		
2	heavy			A,AP	Nom	
3	orange				Nom,A,AP	
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det				
1	very		Adv	AP	Nom	
2	heavy			A,AP	Nom	
3	orange				Nom,A,AP	
4	book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

	1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0 a	Det			NP	
1 very		Adv	AP	Nom	
2 heavy			A,AP	Nom	
3 orange				Nom,A,AP	
4 book					

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	
1	very		Adv	AP	Nom	
2	heavy			A,AP	Nom	
3	orange				Nom,A,AP	
4	book					Nom

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	
1	very		Adv	AP	Nom	
2	heavy			A,AP	Nom	
3	orange				Nom,A,AP	Nom
4	book					Nom

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	
1	very		Adv	AP	Nom	
2	heavy			A,AP	Nom	Nom
3	orange				Nom,A,AP	Nom
4	book					Nom

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	
1	very		Adv	AP	Nom	Nom
2	heavy			A,AP	Nom	Nom
3	orange				Nom,A,AP	Nom
4	book					Nom

Filling out the CYK chart

$_0$ a $_1$ very $_2$ heavy $_3$ orange $_4$ book $_5$

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	NP
1	very		Adv	AP	Nom	Nom
2	heavy			A,AP	Nom	Nom
3	orange				Nom,A,AP	Nom
4	book					Nom

CYK: The general algorithm

function CKY-Parse(*words*, *grammar*) **returns** *table* **for**

$j \leftarrow$ **from** 1 **to** $\textsc{Length}(words)$ **do**

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

$\quad$ **for** $i \leftarrow$ **from** $j-2$ **downto** 0 **do**

$\quad\quad$ **for** $k \leftarrow i+1$ **to** $j-1$ **do**

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

CYK: The general algorithm

function CKY-Parse(*words*, *grammar*) **returns** *table* **for**

$j \leftarrow$**from** 1 **to** LENGTH(*words*) **do** — loop over the columns

  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$ — fill bottom cell

  for $i \leftarrow$**from** $j-2$ **downto** 0 **do** — fill row i in column j

    for $k \leftarrow i+1$ **to** $j-1$ **do** — loop over split locations between i and j

      $table[i, j] \leftarrow table[i, j] \cup$

        $\{A \mid A \rightarrow BC \in grammar,$

          $B \in table[i, k]$

          $C \in table[k, j]\}$

Check the grammar for rules that link the constituents in $[i, k]$ with those in $[k, j]$. For each rule found store LHS in cell $[i, j]$.

A succinct representation of CKY

We have a Boolean table called Chart, such that $\mathrm{Chart}[A, i, j]$ is true if there is a sub-phrase according the grammar that dominates words i through words j

Build this chart recursively, similarly to the Viterbi algorithm:

For $j > i + 1$:

$$\mathrm{Chart}[A, i, j] = \bigvee_{k=i+1}^{j-1} \bigvee_{A \to B\ C} Chart[B, i, k] \wedge Chart[C, k, j]$$

Seed the chart, for $i + 1 = j$:
$\mathrm{Chart}[A, i, i + 1] = \mathrm{True}$ if there exists a rule $A \to w_{i+1}$ where w_{i+1} is the $(i + 1)$th word in the string

- So far, we just have a chart recognizer, a way of determining whether a string belongs to the given language.
- Changing this to a parser requires recording which existing constituents were combined to make each new constituent.
- This requires another field to record the one or more ways in which a constituent spanning (i,j) can be made from constituents spanning (i,k) and (k,j). (More clearly displayed in graph representation, see next lecture.)
- In any case, for a fixed grammar, the CYK algorithm runs in time $O(n^3)$ on an input string of n tokens.
- The algorithm identifies all possible parses.

Even without converting a grammar to CNF, we can draw *CYK-style* parse charts:

		1 *a*	2 *very*	3 *heavy*	4 *orange*	5 *book*
0	a	Det			NP	NP
1	very		OptAdv	OptAP	Nom	Nom
2	heavy			A,OptAP	Nom	Nom
3	orange				N,Nom,A,AP	Nom
4	book					N,Nom

(We haven't attempted to show ϵ-phrases here. Could in principle use cells below the main diagonal for this . . .)

However, CYK-style parsing will have run-time worse than $O(n^3)$ if e.g. the grammar has rules $A \rightarrow BCD$.

Second example

Grammar Rules in CNF

$S \rightarrow NP\ VP$	$Nominal \rightarrow book \mid flight \mid money$
$S \rightarrow X1\ VP$	$Nominal \rightarrow Nominal\ noun$
$X1 \rightarrow Aux\ VP$	$Nominal \rightarrow Nominal\ PP$
$S \rightarrow book \mid include \mid prefer$	$VP \rightarrow book \mid include \mid prefer$
$S \rightarrow Verb\ NP$	$VPVerb \rightarrow NP$
$S \rightarrow X2$	$VP \rightarrow X2\ PP$
$S \rightarrow Verb\ PP$	$X2 \rightarrow Verb\ NP$
$S \rightarrow VP\ PP$	$VP \rightarrow Verb\ NP$
$NP \rightarrow TWA \mid Houston$	$VP \rightarrow VP\ PP$
$NP \rightarrow Det\ Nominal$	$PP \rightarrow Preposition\ NP$
$Verb \rightarrow book \mid include \mid prefer$	$Noun \rightarrow book \mid flight \mid money$

Let's parse *Book the flight through Houston*!

Second example

Grammar Rules in CNF

$S \rightarrow NP\ VP$	$Nominal \rightarrow book\|flight\|money$
$S \rightarrow X1\ VP$	$Nominal \rightarrow Nominal\ noun$
$X1 \rightarrow Aux\ VP$	$Nominal \rightarrow Nominal\ PP$
$S \rightarrow book\|include\|prefer$	$VP \rightarrow book\|include\|prefer$
$S \rightarrow Verb\ NP$	$VPVerb \rightarrow NP$
$S \rightarrow X2$	$VP \rightarrow X2\ PP$
$S \rightarrow Verb\ PP$	$X2 \rightarrow Verb\ NP$
$S \rightarrow VP\ PP$	$VP \rightarrow Verb\ NP$
$NP \rightarrow TWA\|Houston$	$VP \rightarrow VP\ PP$
$NP \rightarrow Det\ Nominal$	$PP \rightarrow Preposition\ NP$
$Verb \rightarrow book\|include\|prefer$	$Noun \rightarrow book\|flight\|money$

Let's parse *Book the flight through Houston*!

Second example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0, 1]				

Second example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0, 1]				
	Det [1, 2]			

Second example

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

Second example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun $[0, 1]$				
	Det $[1, 2]$			
		Nominal, Noun $[2, 3]$		
			Prep $[3, 4]$	

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

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

Second example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0, 1]	[0, 2]	S, VP, X2 [0, 3]	[0, 4]	S_1, VP, X2, S_2, VP, S_3 [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]

Visualizing the Chart

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0, 1]	[0, 2]	S, VP, X2 [0, 3]	[0, 4]	S_1, VP, X2, S_2, VP, S_3 [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]

Visualizing the Chart

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0, 1]	[0, 2]	S, VP, X2 [0, 3]	[0, 4]	S_1, VP, X2, S_2, VP, S_3 [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]

Dynamic Programming as a problem-solving technique

- Given a problem, systematically fill a table of solutions to sub-problems: this is called memoization.
- Once solutions to all sub-problems have been accumulated, solve the overall problem by composing them.
- For parsing, the sub-problems are analyses of sub-strings and correspond to constituents that have been found.
- Sub-trees are stored in a chart (aka well-formed substring table), which is a record of all the substructures that have ever been built during the parse.

Solves **re-parsing problem**: sub-trees are looked up, not re-parsed!
Solves **ambiguity problem**: chart implicitly stores all parses!

A Tribute to CKY (part 1/3)

You, my CKY algorithm,
dictate every parser's rhythm,
if Cocke, Younger and Kasami hadn't bothered,
all of our parsing dreams would have been shattered.

You are so simple, yet so powerful,
and with the proper semiring and time,
you will be truthful,
to return the best parse - anything less would be a crime.

With dynamic programming or memoization,
you are one of a kind,
I really don't need to mention,
if it werent for you, all syntax trees would be behind.

A Tribute to CKY (part 2/3)

Failed attempts have been made to show there are better,
for example, by using matrix multiplication,
all of these impractical algorithms didn't matter
you came out stronger, insisting on just using summation.

All parsing algorithms to you hail,
at least those with backbones which are context-free,
you will never become stale,
as long as we need to have a syntax tree.

It doesn't matter that the C is always in front,
or that the K and Y can swap,
you are still on the same hunt,
maximizing and summing, nonstop.

A Tribute to CKY (part 3/3)

Every Informatics student knows you intimately,
they have seen your variants dozens of times,
you have earned that respect legitimately,
and you will follow them through their primes.

CKY, going backward and forward,
inside and out,
it is so straightforward -
You are the best, there is no doubt.

Summary

- Parsing as search is inefficient (typically exponential time).
- Alternative: use dynamic programming and memoize sub-analysis in a chart to avoid duplicate work.
- The chart can be visualized as as a matrix.
- The CYK algorithm builds a chart in $O(n^3)$ time. The basic version gives just a recognizer, but it can be made into a parser if more info is recorded in the chart.

Reading:	J&M (2nd ed), Chapter. 13, Sections 13.3–13.4 NLTK Book, Chapter. 8 (*Analyzing Sentence Structure*), Section 8.4
Next lecture:	the Earley parser or dynamic programming for top-down parsing