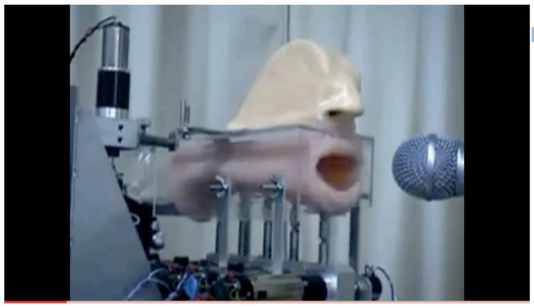


http://www.youtube.com/watch?v=dD_NdnYrDzY

PARSING 2

David Kauchak
CS159 – Spring 2011

some slides adapted from Ray Mooney

Admin

- Quiz 1 (out of 32)
  - High: 31
  - Average: 26
- Assignment 3 will be out soon
- Watson vs. Humans

Parsing

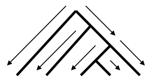
- Given a CFG and a sentence, determine the possible parse tree(s)

I eat sushi with tuna

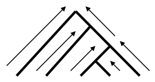
```
S -> NP  VP
NP -> PRP
NP -> N PP
VP -> V NP
VP -> V NP PP
PP -> IN N
PRP -> I
V -> eat
N -> sushi
N -> tuna
IN -> with
```

Parsing

- Top-down parsing
  - start at the top (usually S) and apply rules
  - matching left-hand sides and replacing with right-hand sides



- Bottom-up parsing
  - start at the bottom (i.e. words) and build the parse tree up from there
  - matching right-hand sides and replacing with left-hand sides



Dynamic Programming Parsing

- To avoid extensive repeated work you must cache intermediate results, specifically found constituents
- Caching (memoizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs
- Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where n is the length of the input string.

CKY

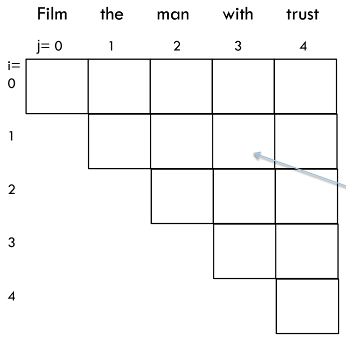
- First grammar must be converted to **Chomsky normal form (CNF)**
  - We'll allow all unary rules, though
- Parse bottom-up storing phrases formed from all substrings in a triangular table (chart)

CNF Grammar

```
S -> VP
VP -> VB NP
VP -> VB NP PP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> trust
NN -> man
NN -> film
NN -> trust
```

```
S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> trust
NN -> man
NN -> film
NN -> trust
```

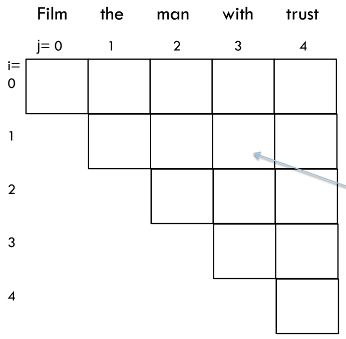
CKY parser: the chart



Cell[*i*,*j*] contains all constituents covering words *i* through *j*

what does this cell represent?

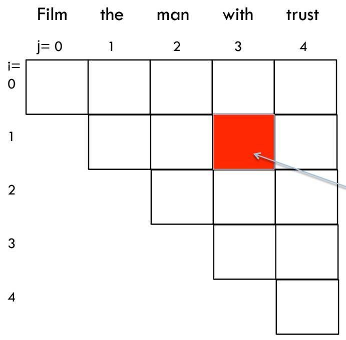
CKY parser: the chart



Cell[*i*,*j*] contains all constituents covering words *i* through *j*

all constituents spanning 1-3 or "the man with"

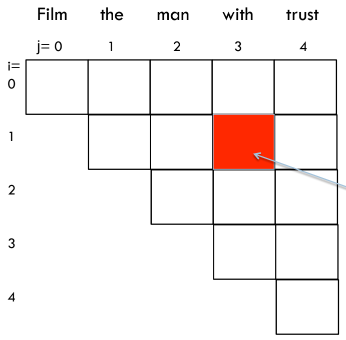
CKY parser: the chart



Cell[*i*,*j*] contains all constituents covering words *i* through *j*

how could we figure this out?

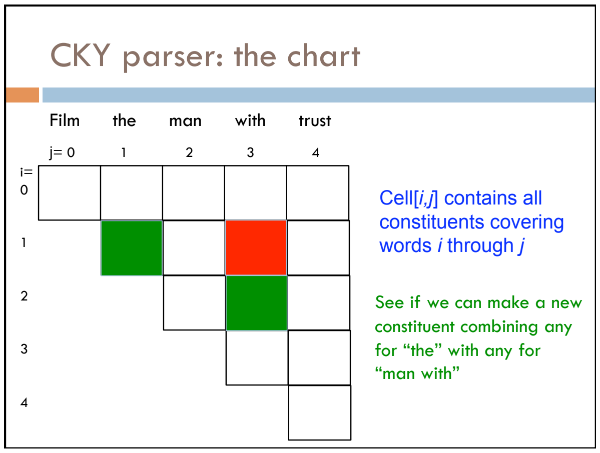
CKY parser: the chart



Cell[*i*,*j*] contains all constituents covering words *i* through *j*

Key: rules are binary and only have two constituents on the right hand side

VP -> VB NP
NP -> DT NN

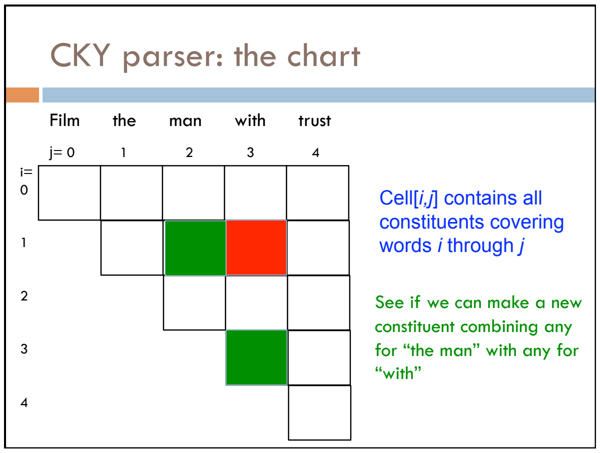


CKY parser: the chart

i=	Film j= 0	the 1	man 2	with 3	trust 4
0					
1		🟩		🟥	
2				🟩	
3					
4					

Cell[*i,j*] contains all constituents covering words *i* through *j*

See if we can make a new constituent combining any for “the” with any for “man with”

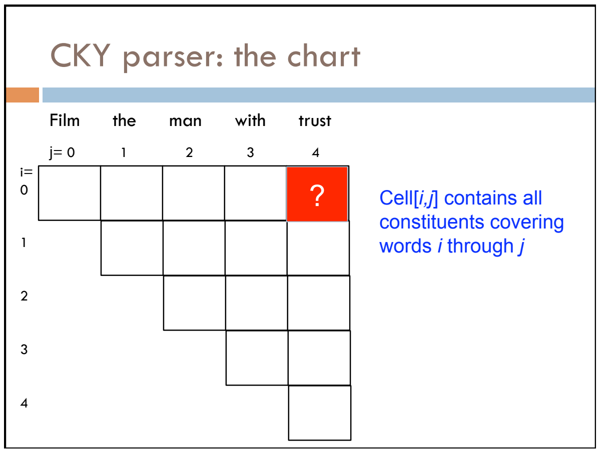


CKY parser: the chart

i=	Film j= 0	the 1	man 2	with 3	trust 4
0					
1			🟩	🟥	
2					
3				🟩	
4					

Cell[*i,j*] contains all constituents covering words *i* through *j*

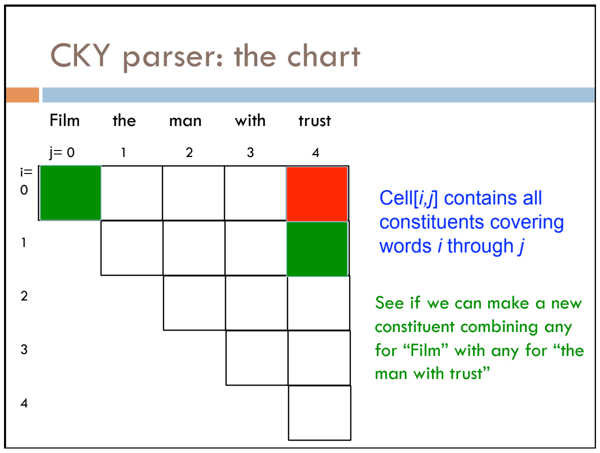
See if we can make a new constituent combining any for “the man” with any for “with”



CKY parser: the chart

i=	Film j= 0	the 1	man 2	with 3	trust 4
0					🟥 ?
1					
2					
3					
4					

Cell[*i,j*] contains all constituents covering words *i* through *j*



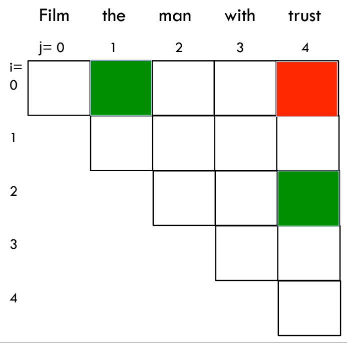
CKY parser: the chart

i=	Film j= 0	the 1	man 2	with 3	trust 4
0	🟩				🟥
1					🟩
2					
3					
4					

Cell[*i,j*] contains all constituents covering words *i* through *j*

See if we can make a new constituent combining any for “Film” with any for “the man with trust”

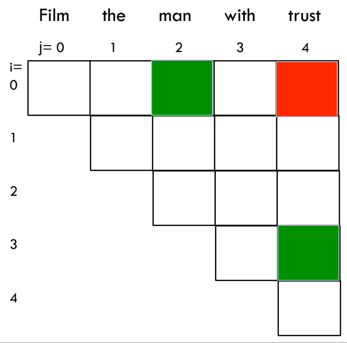
CKY parser: the chart



Cell[*i,j*] contains all constituents covering words *i* through *j*

See if we can make a new constituent combining any for “Film the” with any for “man with trust”

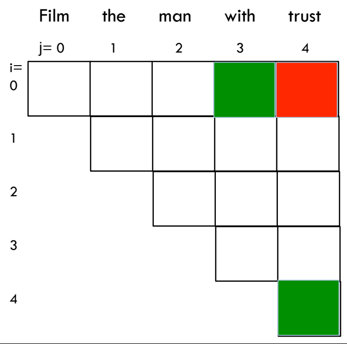
CKY parser: the chart



Cell[*i,j*] contains all constituents covering words *i* through *j*

See if we can make a new constituent combining any for “Film the man” with any for “with trust”

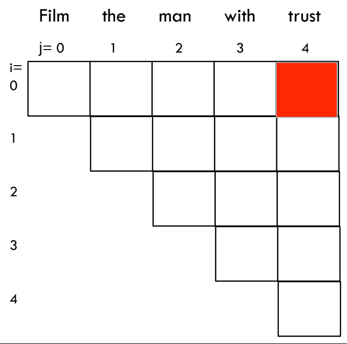
CKY parser: the chart



Cell[*i,j*] contains all constituents covering words *i* through *j*

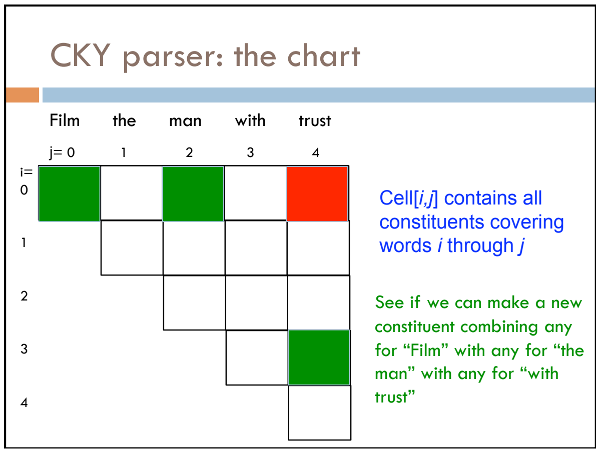
See if we can make a new constituent combining any for “Film the man with” with any for “trust”

CKY parser: the chart



Cell[*i,j*] contains all constituents covering words *i* through *j*

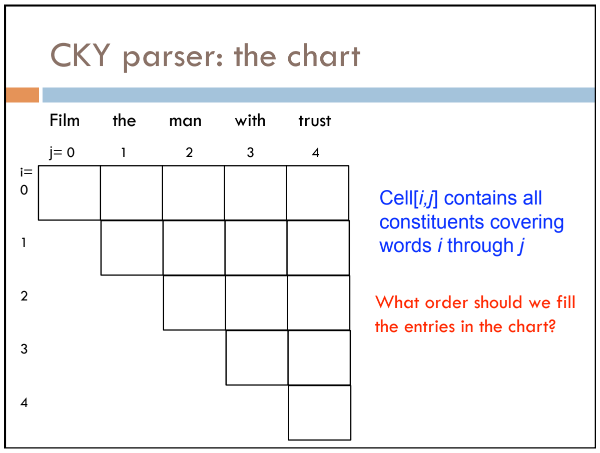
What if our rules weren’t binary?



CKY parser: the chart

Cell[*i,j*] contains all constituents covering words *i* through *j*

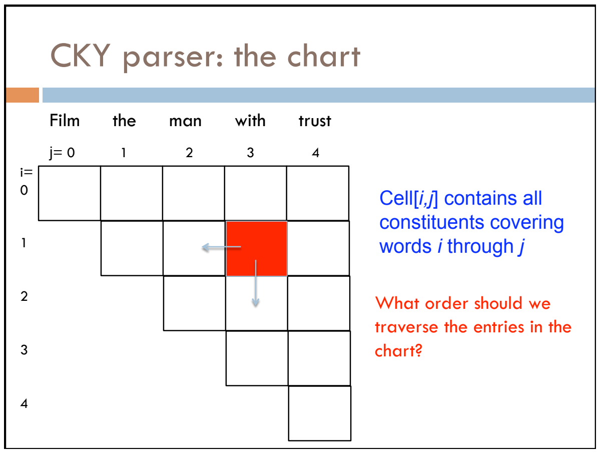
See if we can make a new constituent combining any for "Film" with any for "the man" with any for "with trust"



CKY parser: the chart

Cell[*i,j*] contains all constituents covering words *i* through *j*

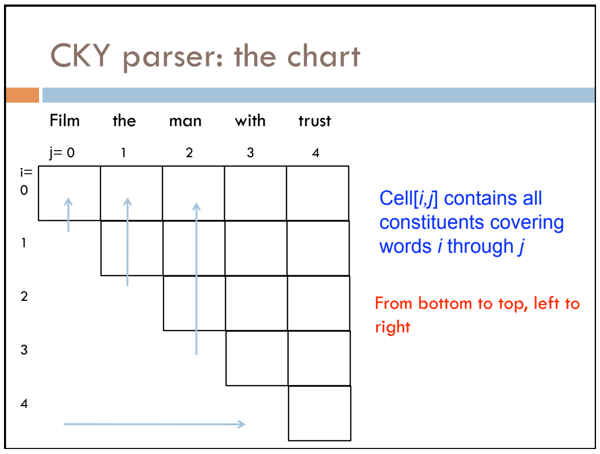
What order should we fill the entries in the chart?



CKY parser: the chart

Cell[*i,j*] contains all constituents covering words *i* through *j*

What order should we traverse the entries in the chart?

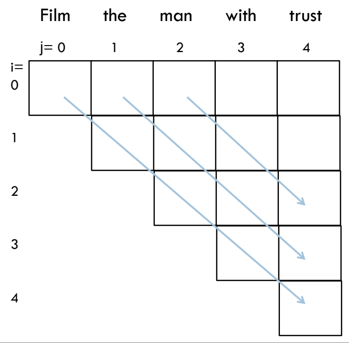


CKY parser: the chart

Cell[*i,j*] contains all constituents covering words *i* through *j*

From bottom to top, left to right

CKY parser: the chart



Cell[*i*,*j*] contains all constituents covering words *i* through *j*

Top-left along the diagonals moving to the right

CKY parser: unary rules

S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> trust
NN -> man
NN -> film
NN -> trust

- Often, we will leave unary rules rather than converting to CNF
- Do these complicate the algorithm?
  - Must check whenever we add a constituent to see if any unary rules apply

CKY parser: the chart

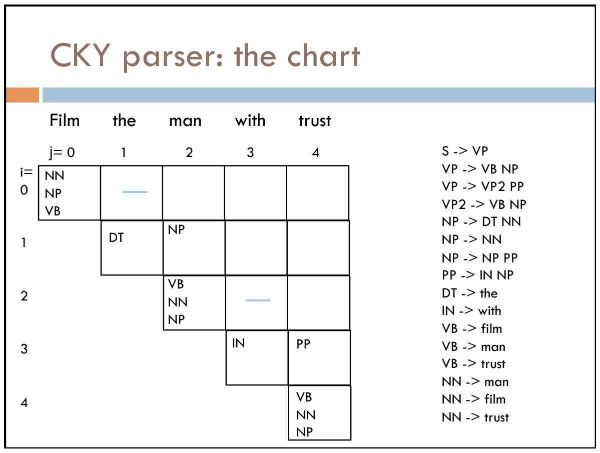
	Film j= 0	the 1	man 2	with 3	trust 4
i= 0					
1					
2					
3					
4					

S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust

CKY parser: the chart

	Film j= 0	the 1	man 2	with 3	trust 4
i= 0	NN NP VB				
1		DT			
2			VB NN NP		
3				IN	
4					VB NN NP

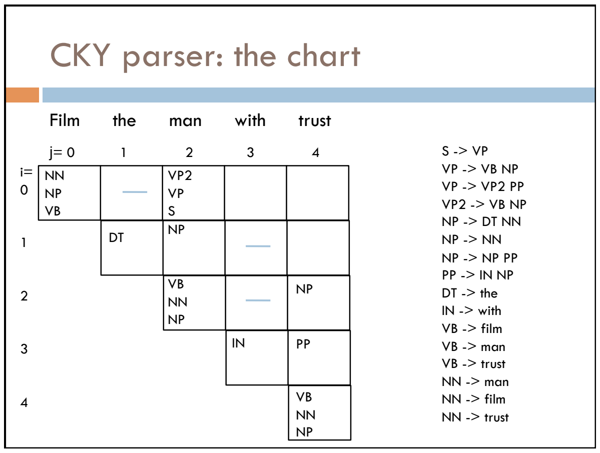
S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust



CKY parser: the chart

	Film	the	man	with	trust
i=	j= 0	1	2	3	4
0	NN NP VB	—			
1		DT	NP		
2			VB NN NP	—	
3				IN	PP
4					VB NN NP

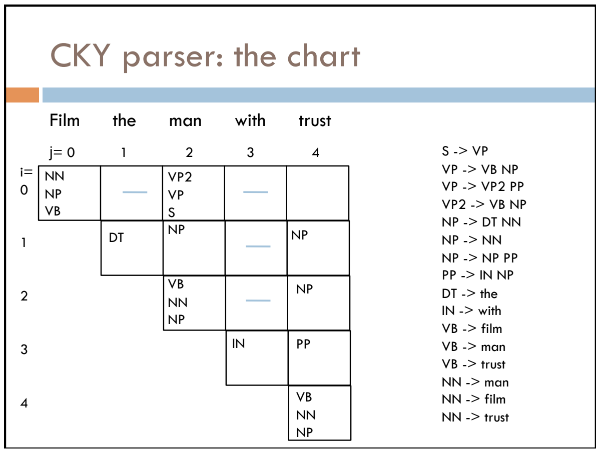
S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust



CKY parser: the chart

	Film	the	man	with	trust
i=	j= 0	1	2	3	4
0	NN NP VB	—	VP2 VP S		
1		DT	NP	—	
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

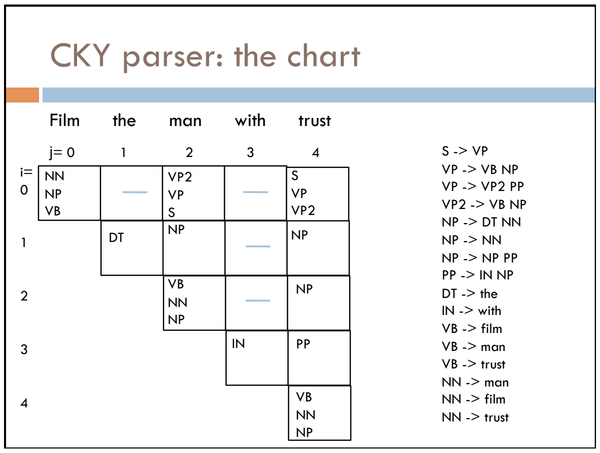
S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust



CKY parser: the chart

	Film	the	man	with	trust
i=	j= 0	1	2	3	4
0	NN NP VB	—	VP2 VP S	—	
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust



CKY parser: the chart

	Film	the	man	with	trust
i=	j= 0	1	2	3	4
0	NN NP VB	—	VP2 VP S	—	S VP VP2
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> man
VB -> trust
NN -> man
NN -> film
NN -> trust

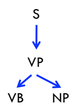
CKY: some things to talk about

- After we fill in the chart, how do we know if there is a parse?
  - If there is an **S** in the upper right corner
- What if we want an actual tree/parse?

	j= 0	1	2	3	4
i= 0	NN VB	—	VB2 VP S	—	S VP
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

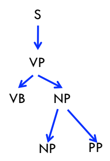
CKY: retrieving the parse

	Film j= 0	the 1	man 2	with 3	trust 4
i= 0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP



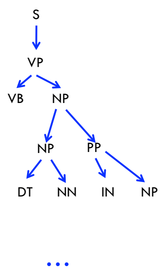
CKY: retrieving the parse

	Film j= 0	the 1	man 2	with 3	trust 4
i= 0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP		NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

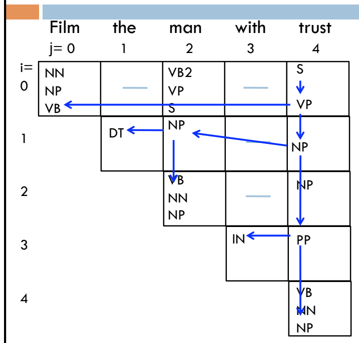


CKY: retrieving the parse

	Film j= 0	the 1	man 2	with 3	trust 4
i= 0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP		NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP



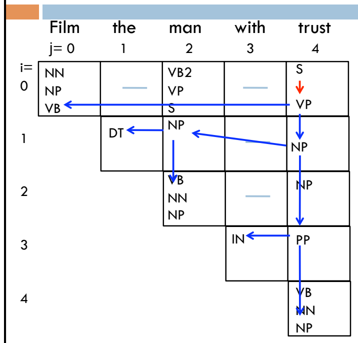
CKY: retrieving the parse



i=	Film j= 0	the 1	man 2	with 3	trust 4
0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP		NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

Where do these arrows/ references come from?

CKY: retrieving the parse



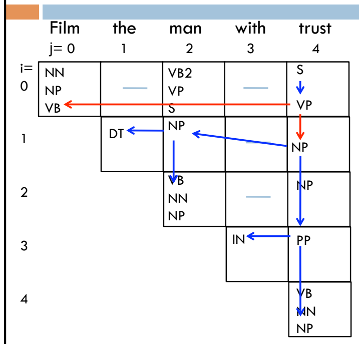
i=	Film j= 0	the 1	man 2	with 3	trust 4
0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP		NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

S -> VP

To add a constituent in a cell, we're applying a rule

The references represent the smaller constituents we used to build this constituent

CKY: retrieving the parse



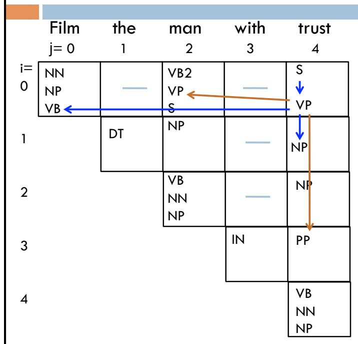
i=	Film j= 0	the 1	man 2	with 3	trust 4
0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP		NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

VP -> VB NP

To add a constituent in a cell, we're applying a rule

The references represent the smaller constituents we used to build this constituent

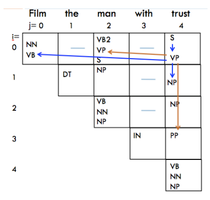
CKY: retrieving the parse



i=	Film j= 0	the 1	man 2	with 3	trust 4
0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP

What about ambiguous parses?

CKY: retrieving the parse



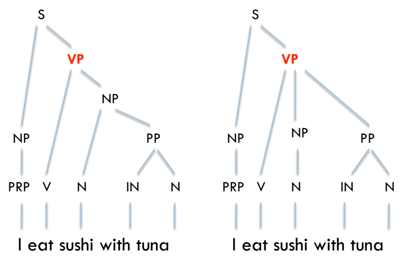
- We can store multiple derivations of each constituent
- This representation is called a "parse forest"
- It is often convenient to leave it in this form, rather than enumerate all possible parses. Why?

CKY: some things to think about

CNF	Actual grammar
S -> VP VP -> VB NP VP -> VP2 PP VP2 -> VB NP NP -> DT NN NP -> NN …	S -> VP VP -> VB NP VP -> VB NP PP NP -> DT NN NP -> NN …
We get a CNF parse tree	but want one for the actual grammar

Ideas?

Parsing ambiguity



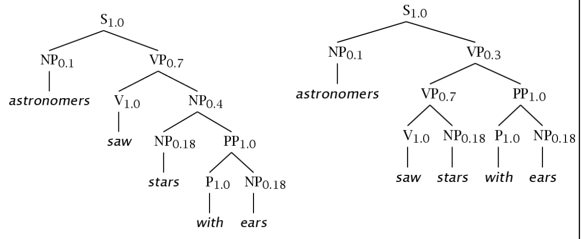
S -> NP VP
NP -> PRP
NP -> N PP
VP -> V NP
VP -> V NP PP
PP -> IN N
PRP -> I
V -> eat
N -> sushi
N -> tuna
IN -> with

How can we decide between these?

A Simple PCFG

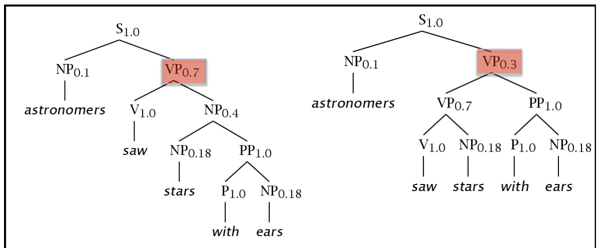
Probabilities!

S	→	NP VP	1.0	NP	→	NP PP	0.4
VP	→	V NP	0.7	NP	→	*astronomers*	0.1
VP	→	VP PP	0.3	NP	→	*ears*	0.18
PP	→	P NP	1.0	NP	→	*saw*	0.04
P	→	*with*	1.0	NP	→	*stars*	0.18
V	→	*saw*	1.0	NP	→	*telescope*	0.1



Just like n-gram language modeling, PCFGs break the sentence generation process into smaller steps/probabilities

The probability of a parse is the product of the PCFG rules



= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18
* 1.0 * 1.0 * 0.18
= 0.0009072

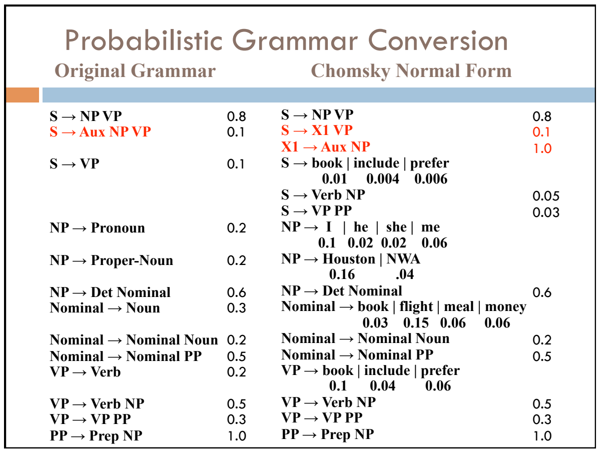
= 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18
* 1.0 * 1.0 * 0.18
= 0.0006804

Parsing with PCFGs

- How does this change our CKY algorithm?
  - We need to keep track of the probability of a constituent
- How do we calculate the probability of a constituent?
  - Product of the PCFG rule times the product of the probabilities of the sub-constituents (right hand sides)
  - Building up the product from the bottom-up
- What if there are multiple ways of deriving a particular constituent?
  - max: pick the most likely derivation of that constituent

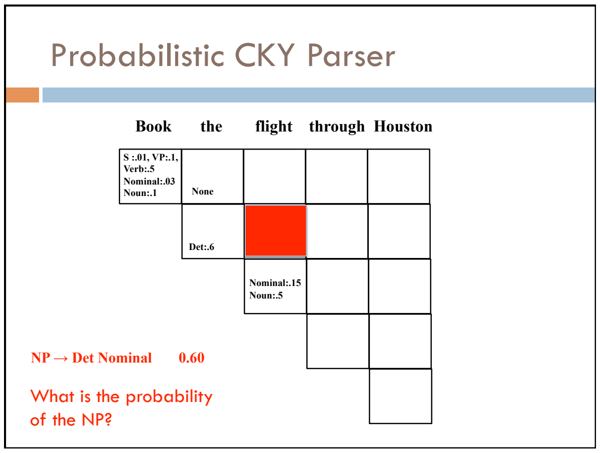
Probabilistic CKY

- Include in each cell a probability for each non-terminal
- Cell[*i*,*j*] must retain the *most probable* derivation of each constituent (non-terminal) covering words *i* through *j*
- When transforming the grammar to CNF, must set production probabilities to preserve the probability of derivations



Probabilistic Grammar Conversion

Original Grammar		Chomsky Normal Form	
S → NP VP	0.8	S → NP VP	0.8
S → Aux NP VP	0.1	S → X1 VP	0.1
		X1 → Aux NP	1.0
S → VP	0.1	S → book | include | prefer 0.01 0.004 0.006	
		S → Verb NP	0.05
		S → VP PP	0.03
NP → Pronoun	0.2	NP → I | he | she | me 0.1 0.02 0.02 0.06	
NP → Proper-Noun	0.2	NP → Houston | NWA 0.16 .04	
NP → Det Nominal	0.6	NP → Det Nominal	0.6
Nominal → Noun	0.3	Nominal → book | flight | meal | money 0.03 0.15 0.06 0.06	
Nominal → Nominal Noun	0.2	Nominal → Nominal Noun	0.2
Nominal → Nominal PP	0.5	Nominal → Nominal PP	0.5
VP → Verb	0.2	VP → book | include | prefer 0.1 0.04 0.06	
VP → Verb NP	0.5	VP → Verb NP	0.5
VP → VP PP	0.3	VP → VP PP	0.3
PP → Prep NP	1.0	PP → Prep NP	1.0

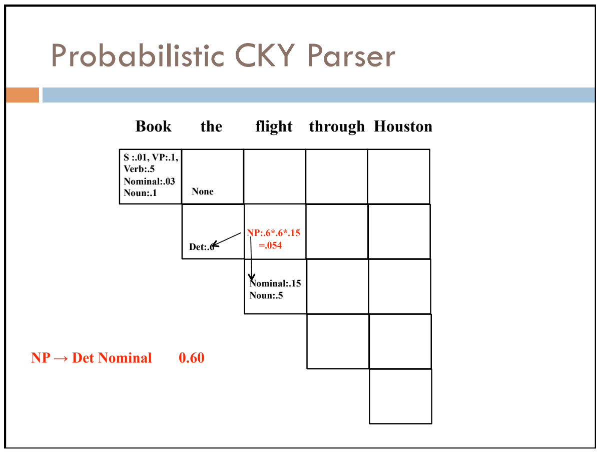


Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None			
	Det:.6	🟥		
		Nominal:.15 Noun:.5		

NP → Det Nominal 0.60

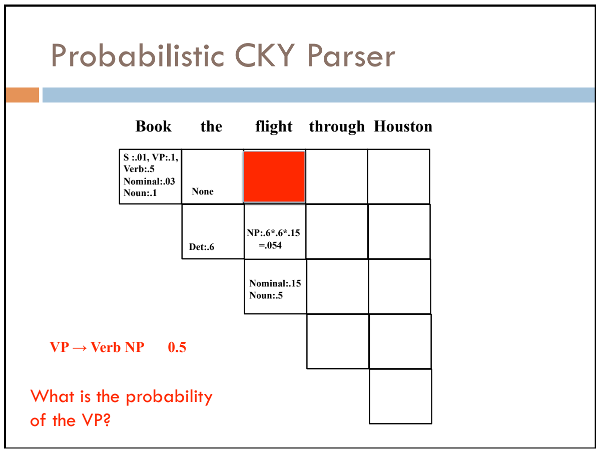
What is the probability of the NP?



Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None			
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

NP → Det Nominal 0.60

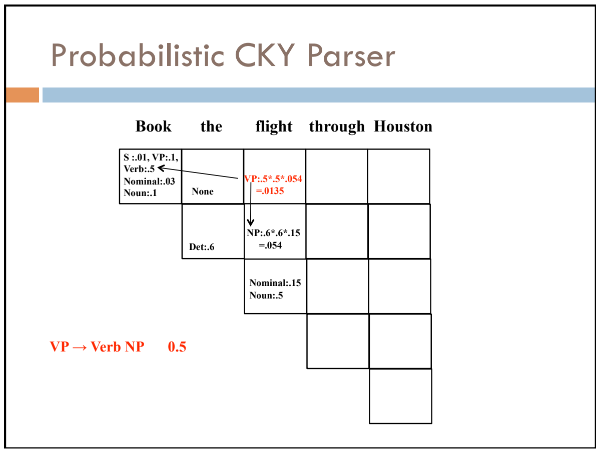


Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	🟥		
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

VP → Verb NP 0.5

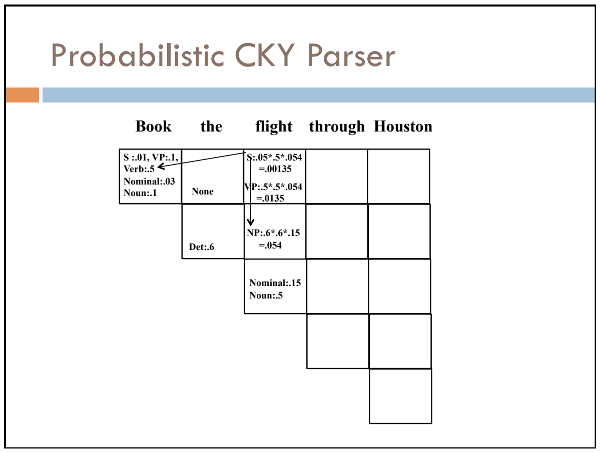
What is the probability of the VP?



Probabilistic CKY Parser

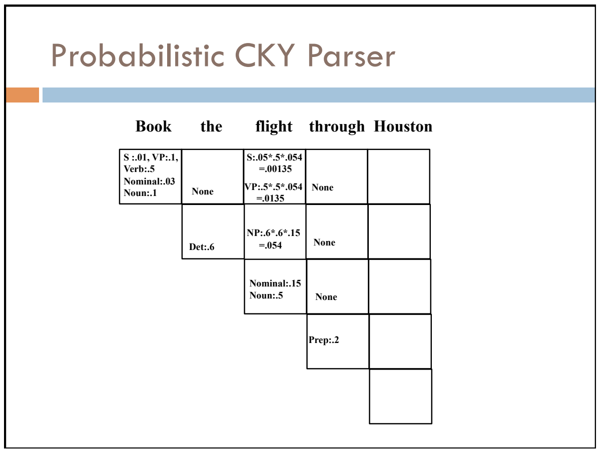
Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	VP:.5*.5*.054 =.0135		
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		

VP → Verb NP 0.5



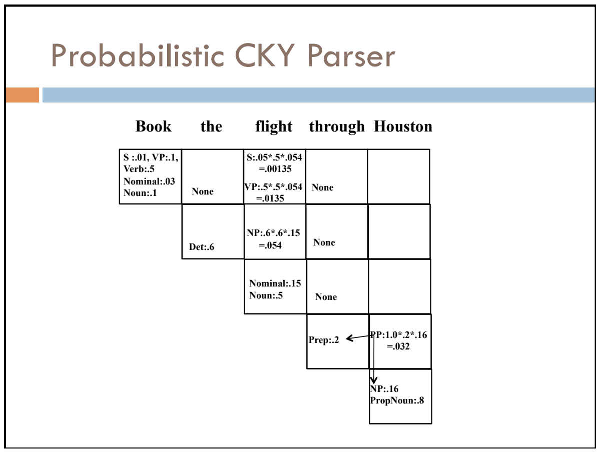
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135		
	Det:.6	NP:.6*.6*.15 =.054		
		Nominal:.15 Noun:.5		



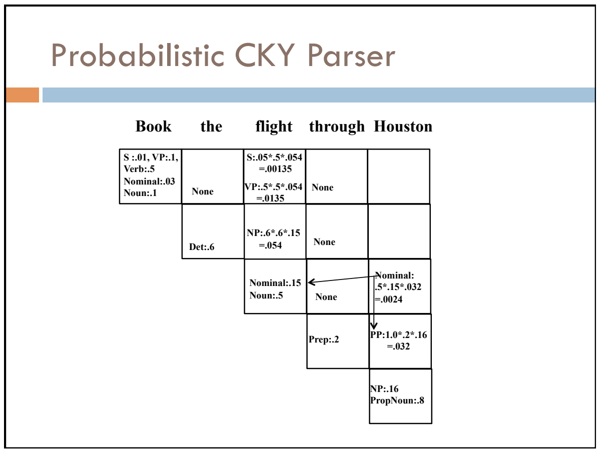
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	
			Prep:.2	



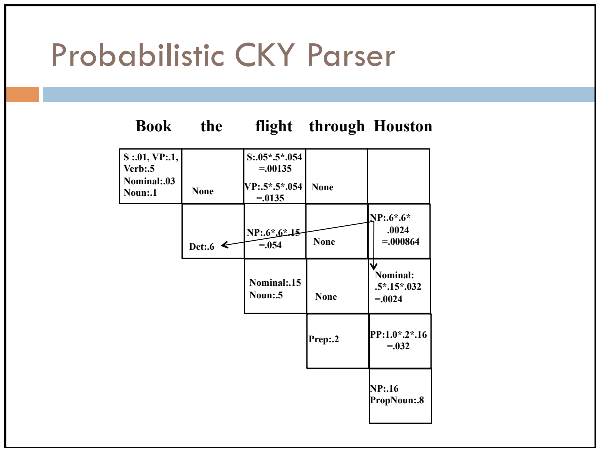
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8



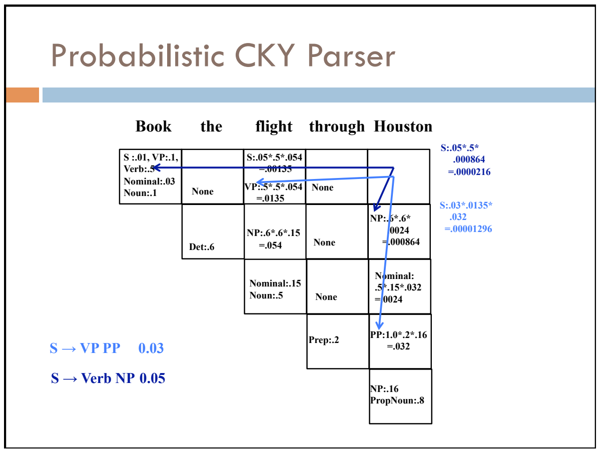
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8



Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8



Probabilistic CKY Parser

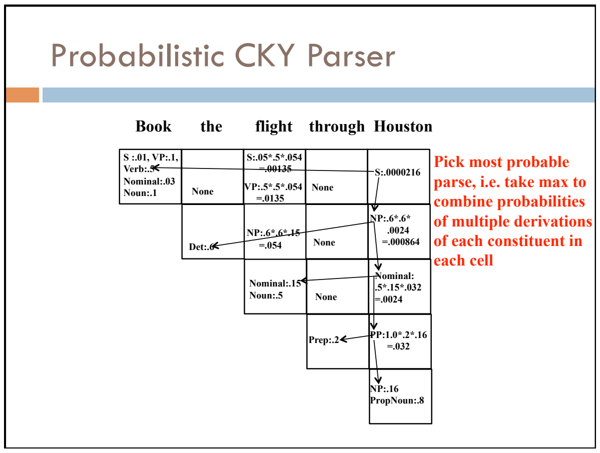
Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

S:.05*.5*
.000864
=.0000216

S:.03*.0135*
.032
=.00001296

S → VP PP 0.03

S → Verb NP 0.05



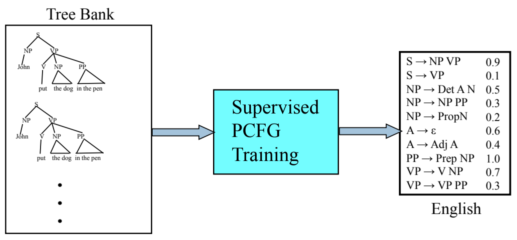
Probabilistic CKY Parser

Book	the	flight	through	Houston
S :.01, VP:.1, Verb:.5 Nominal:.03 Noun:.1	None	S:.05*.5*.054 =.00135 VP:.5*.5*.054 =.0135	None	S:.0000216
	Det:.6	NP:.6*.6*.15 =.054	None	NP:.6*.6* .0024 =.000864
		Nominal:.15 Noun:.5	None	Nominal: .5*.15*.032 =.0024
			Prep:.2	PP:1.0*.2*.16 =.032
				NP:.16 PropNoun:.8

Pick most probable parse, i.e. take max to combine probabilities of multiple derivations of each constituent in each cell

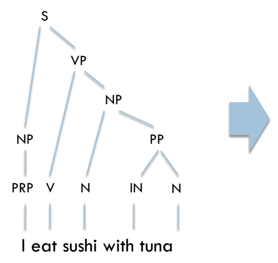
PCFG: Training

- If we have example parsed sentences, how can we learn a set of PCFGs?



S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

Extracting the rules



S -> NP VP
NP -> PRP
PRP -> I
VP -> V NP
V -> eat
NP -> N PP
N -> sushi
PP -> IN N
IN -> with
N -> tuna

What CFG rules occur in this tree?

Estimating PCFG Probabilities

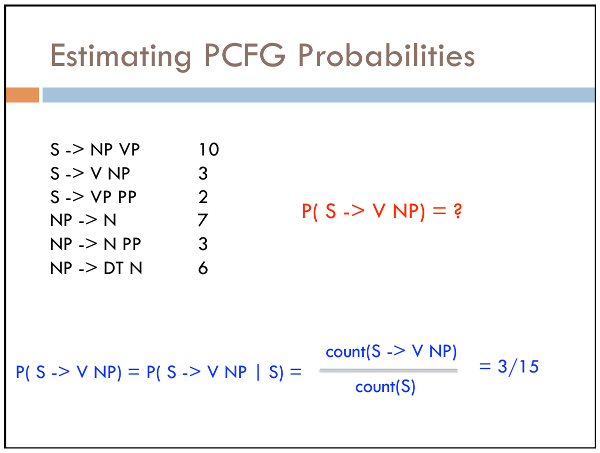
- We can extract the rules from the trees
- Then, we can count the probabilities using MLE

$$P(\alpha \to \beta \mid \alpha) = \frac{\text{count}(\alpha \to \beta)}{\sum_{\gamma} \text{count}(\alpha \to \gamma)} = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)}$$

Estimating PCFG Probabilities

S -> NP VP	10
S -> V NP	3
S -> VP PP	2
NP -> N	7
NP -> N PP	3
NP -> DT N	6

P(S -> V NP) = ?

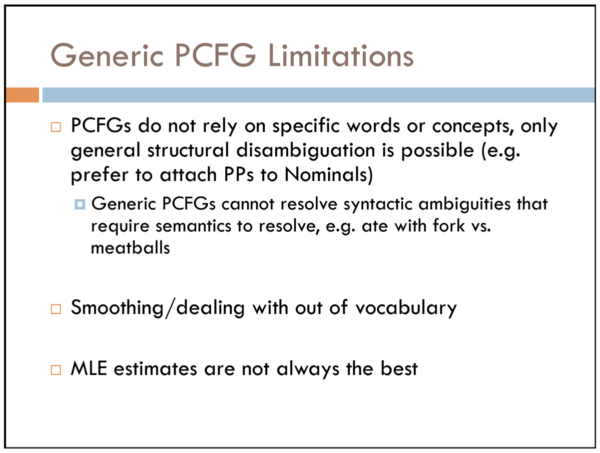


Estimating PCFG Probabilities

S -> NP VP	10
S -> V NP	3
S -> VP PP	2
NP -> N	7
NP -> N PP	3
NP -> DT N	6

P(S -> V NP) = ?

$$P(\text{ S -> V NP}) = P(\text{ S -> V NP } \mid \text{ S}) = \frac{\text{count(S -> V NP)}}{\text{count(S)}} = 3/15$$



Generic PCFG Limitations

- PCFGs do not rely on specific words or concepts, only general structural disambiguation is possible (e.g. prefer to attach PPs to Nominals)
  - Generic PCFGs cannot resolve syntactic ambiguities that require semantics to resolve, e.g. ate with fork vs. meatballs
- Smoothing/dealing with out of vocabulary
- MLE estimates are not always the best