PARSING

David Kauchak
CS159 – Fall 2014

Admin

Assignment 3

Quiz #1
- High: 36
- Average: 33 (92%)
- Median: 33.5 (93%)
- Next one will probably be a bit harder ☺

Parsing

Parsing is the field of NLP interested in automatically determining the syntactic structure of a sentence

parsing can also be thought of as determining what sentences are "valid" English sentences

Parsing

We have a grammar, determine the possible parse tree(s)

Let's start with parsing with a CFG (no probabilities)

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

I eat sushi with tuna

approaches?
algorithms?

Parsing

Top-down parsing

- ends up doing a lot of repeated work
- doesn't take into account the words in the sentence until the end!

Bottom-up parsing

- constrain based on the words
- avoids repeated work (dynamic programming)
- CKY parser

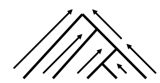
Parsing

Top-down parsing

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

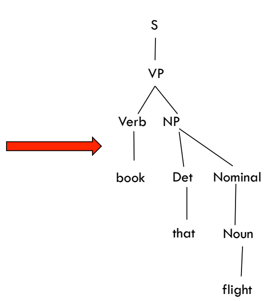
Bottom-up parsing

- 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

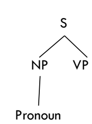


Parsing Example

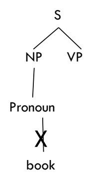
book that flight



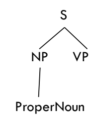
Top Down Parsing



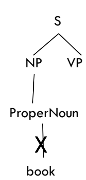
Top Down Parsing



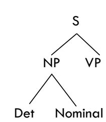
Top Down Parsing



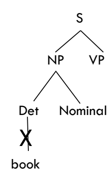
Top Down Parsing



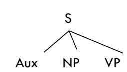
Top Down Parsing



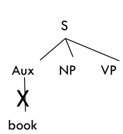
Top Down Parsing



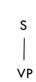
Top Down Parsing



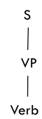
Top Down Parsing



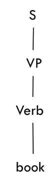
Top Down Parsing



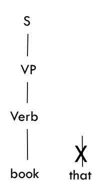
Top Down Parsing



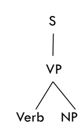
Top Down Parsing



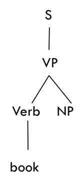
Top Down Parsing



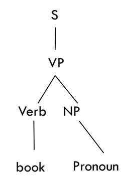
Top Down Parsing



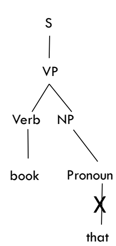
Top Down Parsing



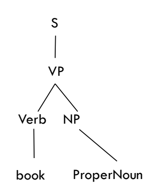
Top Down Parsing



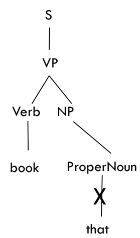
Top Down Parsing



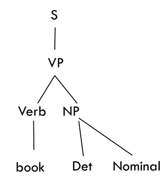
Top Down Parsing



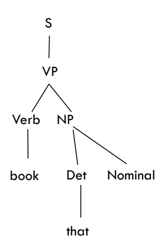
Top Down Parsing



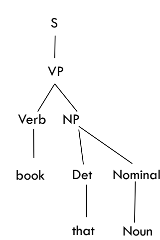
Top Down Parsing



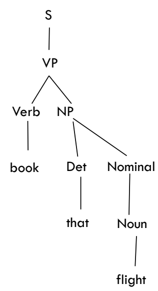
Top Down Parsing



Top Down Parsing



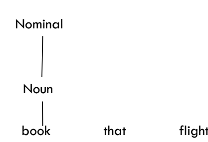
Top Down Parsing



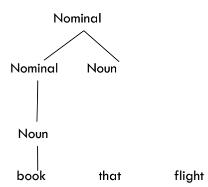
Bottom Up Parsing

Bottom Up Parsing

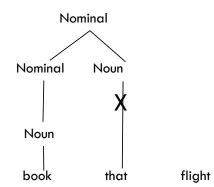
Bottom Up Parsing



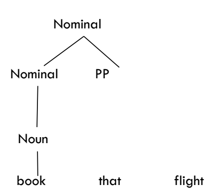
Bottom Up Parsing



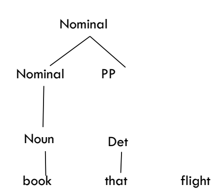
Bottom Up Parsing



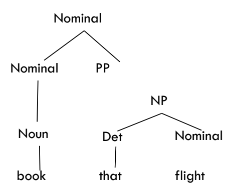
Bottom Up Parsing



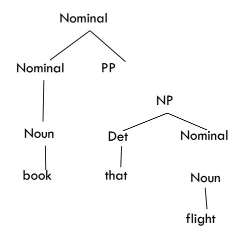
Bottom Up Parsing



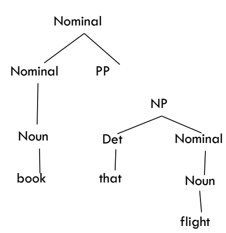
Bottom Up Parsing



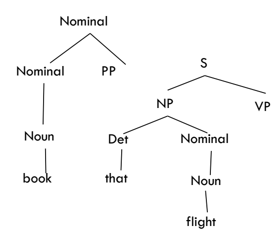
Bottom Up Parsing



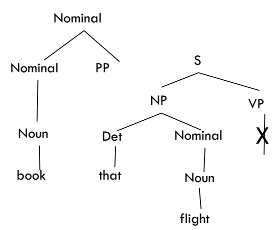
Bottom Up Parsing



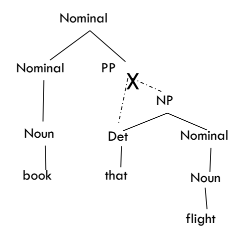
Bottom Up Parsing



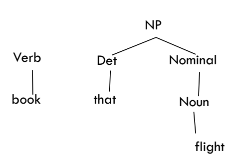
Bottom Up Parsing



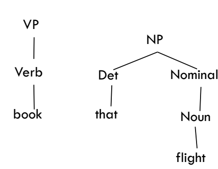
Bottom Up Parsing



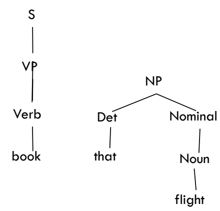
Bottom Up Parsing



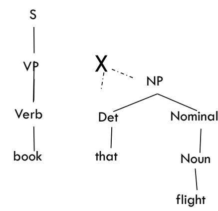
Bottom Up Parsing



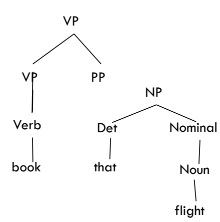
Bottom Up Parsing



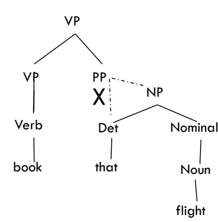
Bottom Up Parsing



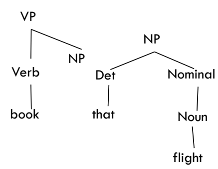
Bottom Up Parsing



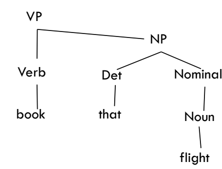
Bottom Up Parsing



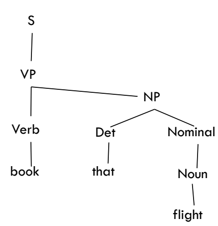
Bottom Up Parsing



Bottom Up Parsing



Bottom Up Parsing



Parsing

Pros/Cons?

- Top-down:
  - Only examines parses that could be valid parses (i.e. with an S on top)
  - Doesn't take into account the actual words!
- Bottom-up:
  - Only examines structures that have the actual words as the leaves
  - Examines sub-parses that may NOT result in a valid parse!

Why is parsing hard?

Actual grammars are large

Lots of ambiguity!

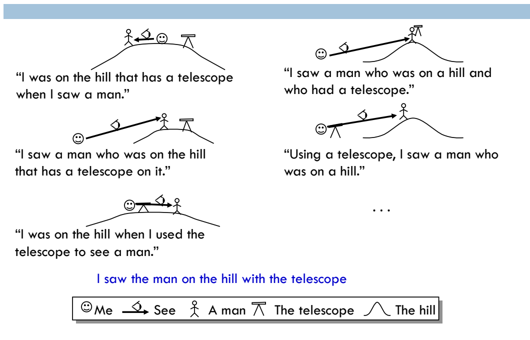
- Most sentences have many parses
- Some sentences have a lot of parses
- Even for sentences that are not ambiguous, there is often ambiguity for subtrees (i.e. multiple ways to parse a phrase)

Why is parsing hard?

I saw the man on the hill with the telescope

What are some interpretations?

Structural Ambiguity Can Give Exponential Parses



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.

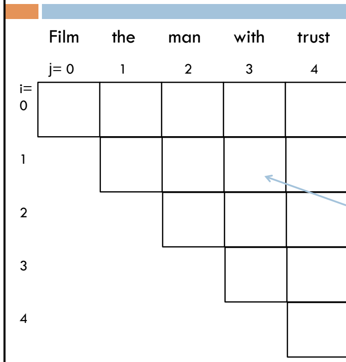
Dynamic Programming Parsing Methods

CKY (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.

Earley parser is based on top-down parsing and does not require normalizing grammar but is more complex.

These both fall under the general category of **chart parsers** which retain completed constituents in a chart

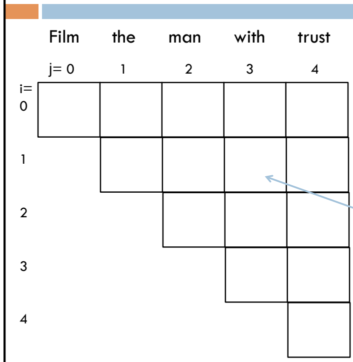
CKY parser: the chart



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

what does this cell represent?

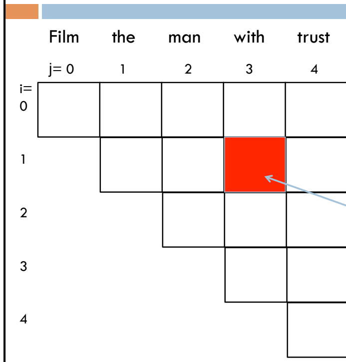
CKY parser: the chart



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

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

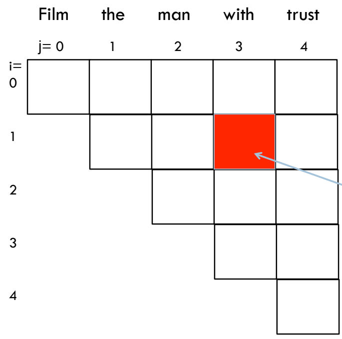
CKY parser: the chart



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

how could we figure this out?

CKY parser: the chart

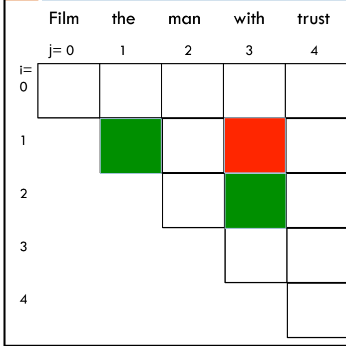


Cell[*i,j*] contains all constituents covering words *i* through *j*

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

VP -> VB NP
NP -> DT NN

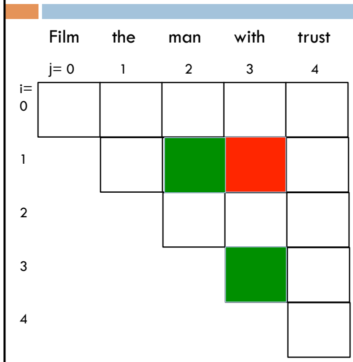
CKY parser: the chart



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"

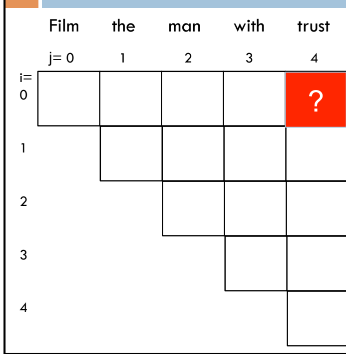
CKY parser: the chart



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"

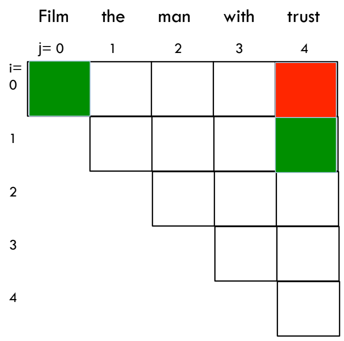
CKY parser: the chart



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

What combinations do we need to consider when trying to put constituents here?

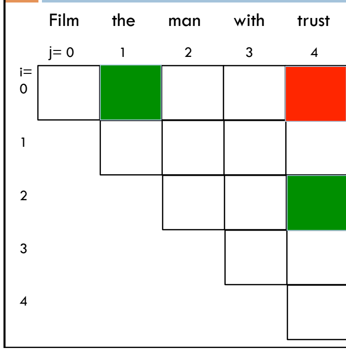
CKY parser: the chart



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 trust”

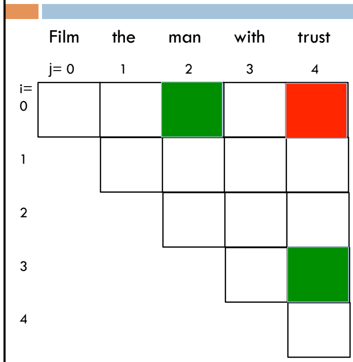
CKY parser: the chart



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”

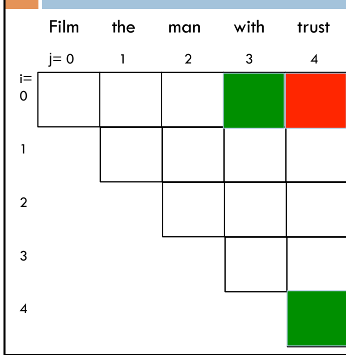
CKY parser: the chart



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”

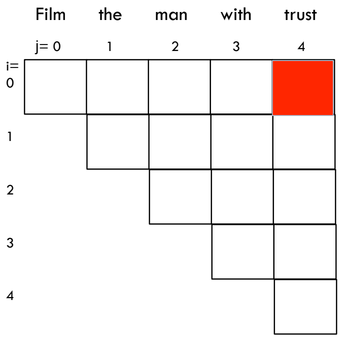
CKY parser: the chart



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”

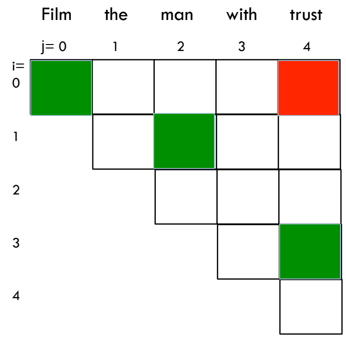
CKY parser: the chart



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

What if our rules weren't binary?

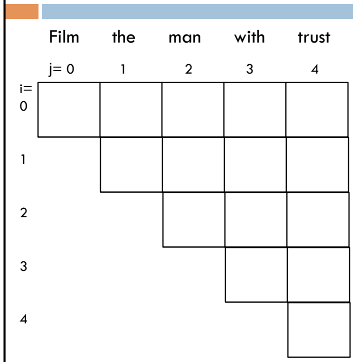
CKY parser: the chart



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"

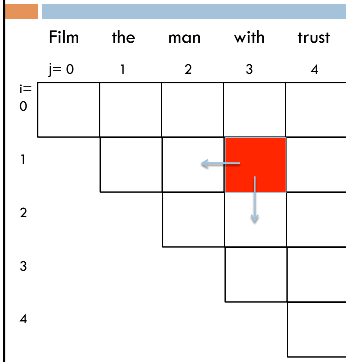
CKY parser: the chart



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

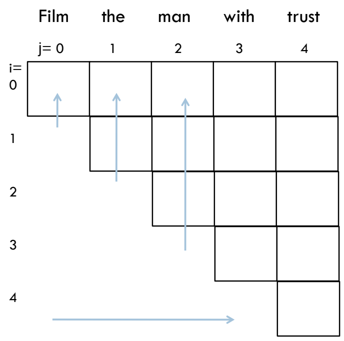
What order should we fill the entries in the chart?

CKY parser: the chart



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

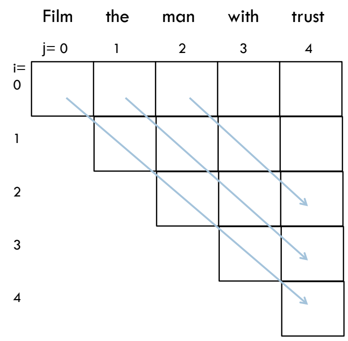
CKY parser: the chart



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

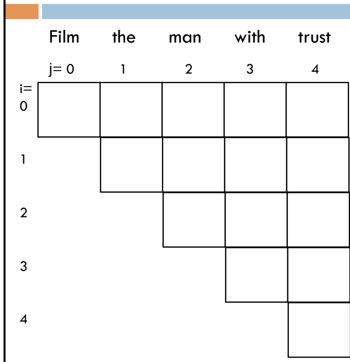
S -> VP
VP -> VB NP
VP -> VP2 PP
VP2 -> VB NP
NP -> DT NN
NP -> NN
NP -> NP PP
PP -> IN NP
DT -> the
IN -> with
VB -> film
VB -> trust
NN -> man
NN -> film
NN -> trust

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



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

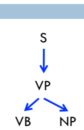
- If there is an **S** in the upper right corner

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

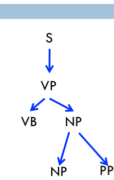
CKY: retrieving the parse

	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

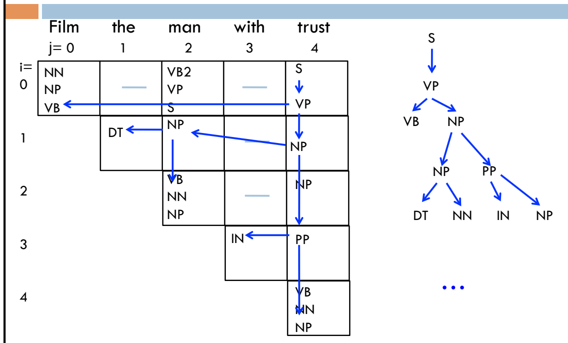


CKY: retrieving the parse

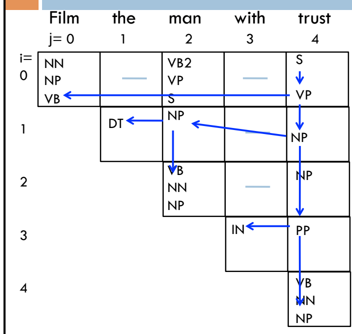
	Film j= 0	the 1	man 2	with 3	trust 4
i= 0	NN NP VB	—	VB2 VP S	—	S VP
1		DT	NP	—	NP
2			VB NN NP	—	NP
3				IN	PP
4					VB NN NP



CKY: retrieving the parse

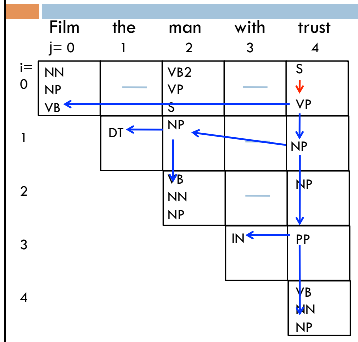


CKY: retrieving the parse



Where do these arrows/ references come from?

CKY: retrieving the parse

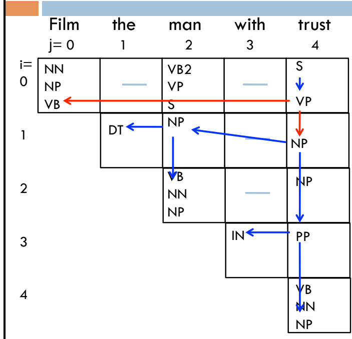


S → VP

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



VP → VB NP

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

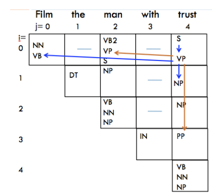
The references represent the smaller constituents we used to build this constituent

CKY: retrieving the parse

i \ j	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

S → VP
VP → VB NP
VP → VP2 PP
VP2 → VB NP
NP → DT NN
NP → NN
…

We get a CNF parse tree

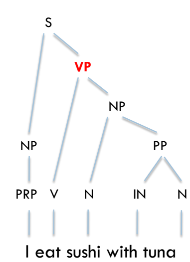
Actual grammar

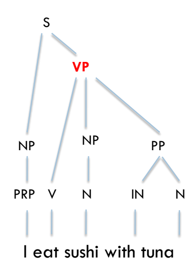
S → VP
VP → VB NP
VP → VB NP PP
NP → DT NN
NP → NN
…

but want one for the actual grammar

Ideas?

Parsing ambiguity





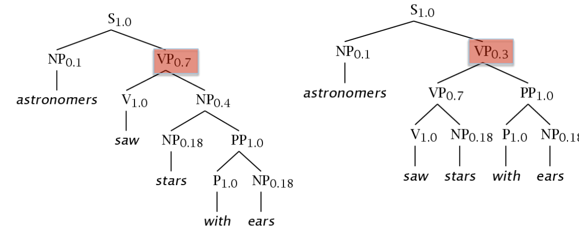
S → NP VP
NP → PRP
NP → N PP
VP → V NP
VP → V NP PP
PP → IN N
PRP → I
V → eat
N → sushi
N → tuna
IN → with

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



= 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

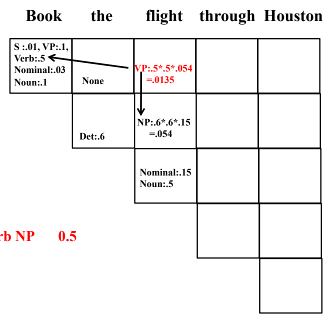
Probabilistic CKY Parser

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

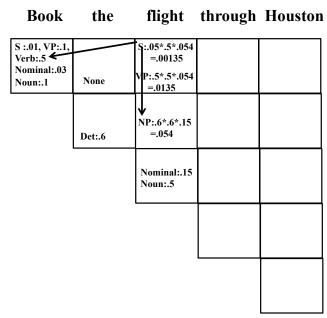
What is the probability of the VP?

Probabilistic CKY Parser

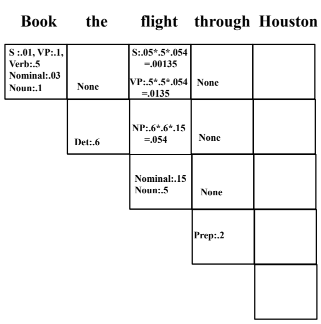


VP → Verb NP 0.5

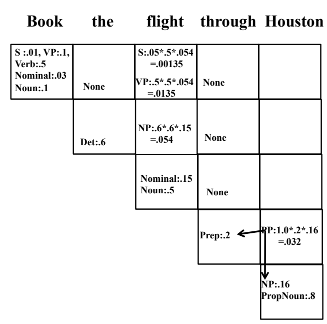
Probabilistic CKY Parser



Probabilistic CKY Parser



Probabilistic CKY Parser



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

Which parse do we pick?

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

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