GRAMMARS

David Kauchak
CS457 – Spring 2011

some slides adapted from Ray Mooney

Admin

- Assignment 2
- Assignment 3
  - Technically due Sunday Oct. 16 at midnight
  - Work in pairs
  - Any programming language
  - Given example output

Constituency

- Parts of speech can be thought of as the lowest level of syntactic information
- Groups *words* together into categories

________ likes to eat candy.

What can/can't go here?

Constituency

________ likes to eat candy.

nouns	determiner nouns
Dave	The man
Professor Kauchak	The boy
Dr. Suess	The cat

pronouns	determiner nouns +
He	The man that I saw
She	The boy with the blue pants
	The cat in the hat

Constituency

- Words in languages tend to form into functional groups (parts of speech)
- Groups of words (aka phrases) can also be grouped into functional groups
  - often some relation to parts of speech
  - though, more complex interactions
- These phrase groups are called constituents

Common constituents

He likes to eat candy.

noun phrase | verb phrase

The man in the hat ran to the park.

noun phrase | verb phrase

Common constituents

The man in the hat ran to the park.

noun phrase | prepositional phrase | prepositional phrase

noun phrase | verb phrase

Common constituents

The man in the hat ran to the park.

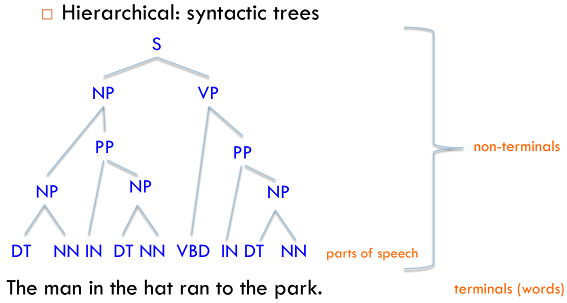
noun phrase | prepositional phrase | noun phrase

noun phrase | prepositional phrase

verb phrase

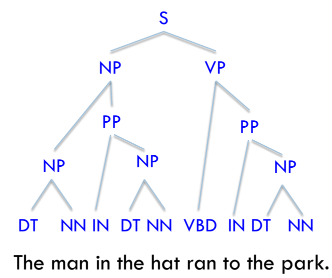
Syntactic structure

- Hierarchical: syntactic trees



Syntactic structure

(S (NP (NP (DT the) (NN man)) (PP (IN in) (NP (DT the) (NN hat)))) (VP (VBD ran) (PP (TO to (NP (DT the) (NN park))))))



Syntactic structure

(S (NP (NP (DT the) (NN man)) (PP (IN in) (NP (DT the) (NN hat)))) (VP (VBD ran) (PP (TO to (NP (DT the) (NN park))))))

```
(S
  (NP
    (NP (DT the) (NN man))
    (PP (IN in)
      (NP (DT the) (NN hat))))
  (VP (VBD ran)
    (PP (TO to)
      (NP (DT the) (NN park))))))
```

Syntactic structure

- A number of related problems:
  - Given a sentence, can we determine the syntactic structure?
  - Can we determine if a sentence is grammatical?
  - Can we determine how *likely* a sentence is to be grammatical? to be an English sentence?
  - Can we generate candidate, grammatical sentences?

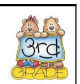
Grammars

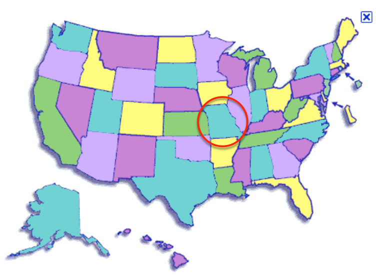
What is a grammar (3^{rd} grade again…)?

Grammars

- Grammar is a set of structural rules that govern the composition of sentences, phrases and words
- Lots of different kinds of grammars:
  - regular
  - context-free
  - context-sensitive
  - recursively enumerable
  - transformation grammars

States





What is the capitol of this state? Jefferson City (Missouri)

Context free grammar

- How many people have heard of them?
- Look like:

$$S \rightarrow NP\ VP$$

left hand side (single symbol) right hand side (one or more symbols)

Formally…

G = (NT, T, P, S)

- NT: finite set of nonterminal symbols
- T: finite set of terminal symbols, NT and T are disjoint
- P: finite set of productions of the form
  $A \rightarrow \alpha$, $A \in V$ and $\alpha \in (T \cup NT)^*$
- $S \in NT$: start symbol

CFG: Example

- Many possible CFGs for English, here is an example (fragment):
  - S → NP VP
  - VP → V NP
  - NP → DetP N | AdjP NP
  - AdjP → Adj | Adv AdjP
  - N → boy | girl
  - V → sees | likes
  - Adj → big | small
  - Adv → very
  - DetP → a | the

Grammar questions

- Can we determine if a sentence is grammatical?
- Given a sentence, can we determine the syntactic structure?
- Can we determine how likely a sentence is to be grammatical? to be an English sentence?
- Can we generate candidate, grammatical sentences?

Which of these can we answer with a CFG? How?

Grammar questions

- Can we determine if a sentence is grammatical?
  - Is it accepted/recognized by the grammar
  - Applying rules right to left, do we get the start symbol?
- Given a sentence, can we determine the syntactic structure?
  - Keep track of the rules applied…
- Can we determine how likely a sentence is to be grammatical? to be an English sentence?
  - Not yet… no notion of "likelihood" (probability)
- Can we generate candidate, grammatical sentences?
  - Start from the start symbol, randomly pick rules that apply (i.e. left hand side matches)

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | the

S

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | the

NP VP

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | **the**

DetP N VP

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | **likes**
Adj → big | small
Adv → very
DetP → a | the

the boy VP

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | the

the boy likes NP

Derivations in a CFG

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | the

the boy likes a girl

Derivations in a CFG; Order of Derivation Irrelevant

S → NP VP
VP → V NP
NP → DetP N | AdjP NP
AdjP → Adj | Adv AdjP
N → boy | girl
V → sees | likes
Adj → big | small
Adv → very
DetP → a | the

DetP N VP

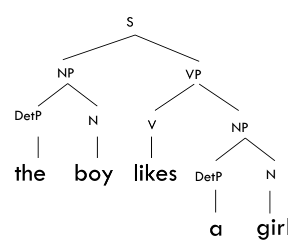
the boy VP   DetP N likes NP

the boy likes a girl

Derivations of CFGs

- String rewriting system: we derive a string
- But derivation history represented by phrase-structure tree

the boy likes a girl



Parsing

- Parsing is the field of NLP interested in automatically determining the syntactic structure of a sentence
- parsing can be thought of as determining what sentences are "valid" English sentences
- As a by product, we often can get the structure

Parsing

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

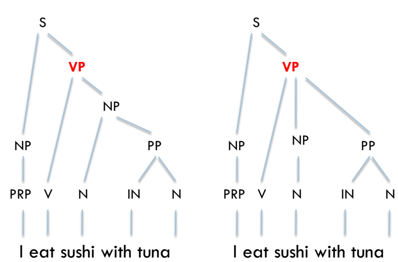
S -> NP VP
NP -> N
NP -> PRP
NP -> N PP
VP -> V NP
VP -> V NP PP
PP -> IN N
PRP -> I
V -> eat
N -> sushi
N -> tuna
IN -> with

I eat sushi with tuna

What parse trees are possible for this sentence?

What if the grammar is much larger?

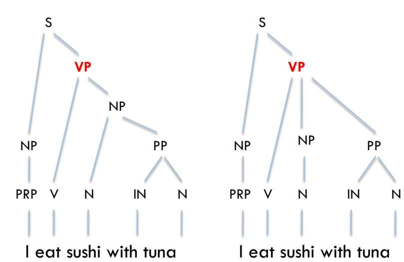
Parsing



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

What is the difference between these parses?

Parsing ambiguity



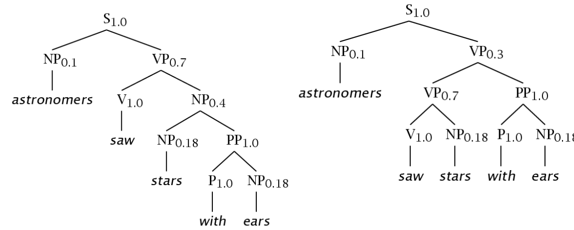
S -> NP VP
NP -> PRP
NP -> N PP
VP -> V NP
VP -> V NP PP
PP -> IN N
PRP -> I
V -> eat
N -> sushi
N -> tuna
IN -> with

How can we decide between these?

A Simple PCFG

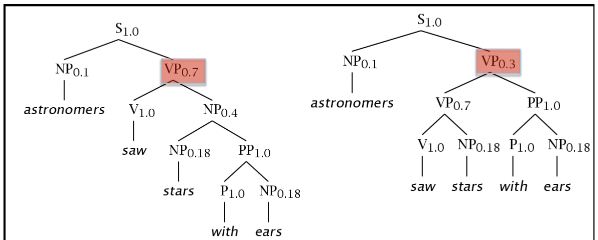
Probabilities!

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

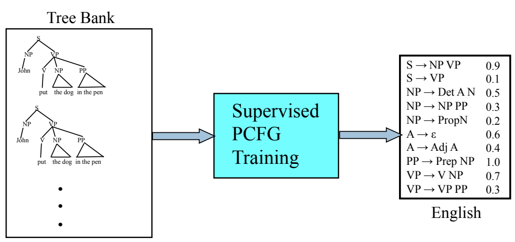
= 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18
= 0.0006804

Parsing problems

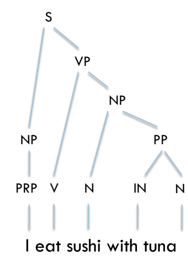
- Pick a model
  - e.g. CFG, PCFG, …
- Train (or learn) a model
  - What CFG/PCFG rules should I use?
  - Parameters (e.g. PCFG probabilities)?
  - What kind of data do we have?
- Parsing
  - Determine the parse tree(s) given a sentence

PCFG: Training

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



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

S -> NP VP
NP -> PRP
PRP -> I
VP -> V NP
V -> eat
NP -> N PP
N -> sushi
…

S	→	NP VP	1.0
VP	→	V NP	0.7
VP	→	VP PP	0.3
PP	→	P NP	1.0
P	→	*with*	1.0
V	→	*saw*	1.0

How do we go from the extracted CFG rules to PCFG rules?

Estimating PCFG Probabilities

- Extract the rules from the trees
- Calculate the probabilities using MLE

$$\alpha \rightarrow \beta \quad \Rightarrow \quad p(\alpha \rightarrow \beta \mid \alpha)$$

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

Estimating PCFG Probabilities

	Occurrences
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) = ?

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

Grammar Equivalence

- Weak equivalence: grammars generate same set of strings
  - Grammar 1: NP → DetP N and DetP → a | the
  - Grammar 2: NP → a N | NP → the N
- Strong equivalence: grammars have same set of derivation trees
  - With CFGs, possible only with useless rules
  - Grammar 2: NP → a N | NP → the N
  - Grammar 3: NP → a N | NP → the N, DetP → many

Normal Forms

- There are weakly equivalent normal forms (Chomsky Normal Form, Greibach Normal Form)
- A CFG is in Chomsky Normal Form (CNF) if all productions are of one of two forms:
  - A → BC with A, B, C nonterminals
  - A → a, with A a nonterminal and a a terminal
- Every CFG has a weakly equivalent CFG in CNF

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

Probabilistic Grammar Conversion

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

Grammar questions

- Can we determine if a sentence is grammatical?
- Given a sentence, can we determine the syntactic structure?
- Can we determine how likely a sentence is to be grammatical? to be an English sentence?
- Can we generate candidate, grammatical sentences?

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?

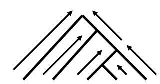
Parsing

- 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

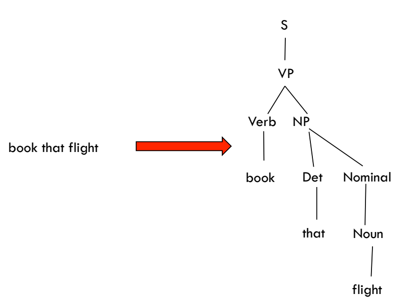
Parsing

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

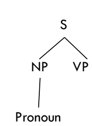
- Bottom-up parsing
  - start at the bottom (i.e. words) and build the parse tree up from there
  - matching right-hand sides and replacing with left-hand sides



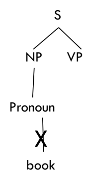
Parsing Example



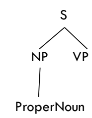
Top Down Parsing



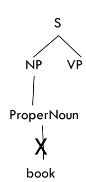
Top Down Parsing



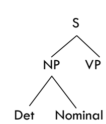
Top Down Parsing



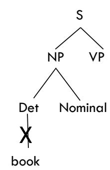
Top Down Parsing



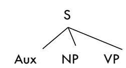
Top Down Parsing



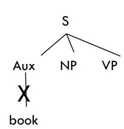
Top Down Parsing



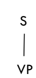
Top Down Parsing



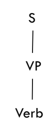
Top Down Parsing



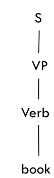
Top Down Parsing



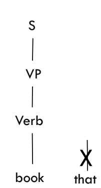
Top Down Parsing



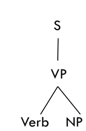
Top Down Parsing



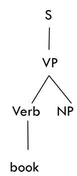
Top Down Parsing



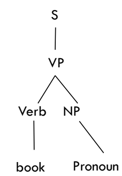
Top Down Parsing



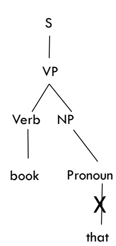
Top Down Parsing



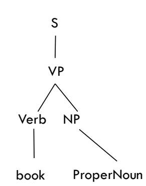
Top Down Parsing



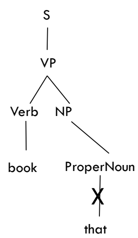
Top Down Parsing



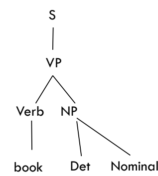
Top Down Parsing



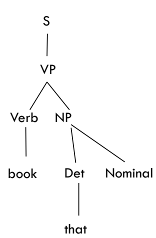
Top Down Parsing



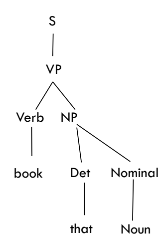
Top Down Parsing



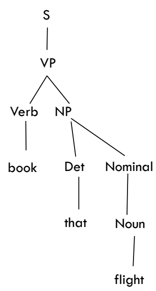
Top Down Parsing



Top Down Parsing



Top Down Parsing

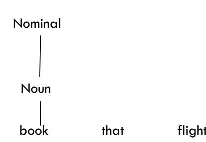


Bottom Up Parsing

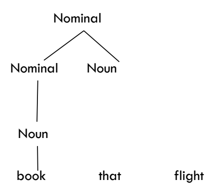
book that flight

Bottom Up Parsing

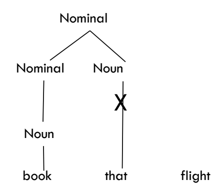
Bottom Up Parsing



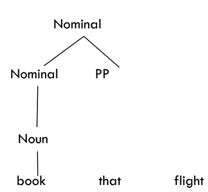
Bottom Up Parsing



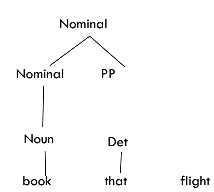
Bottom Up Parsing



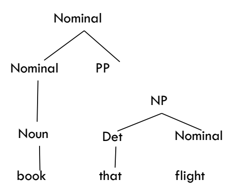
Bottom Up Parsing



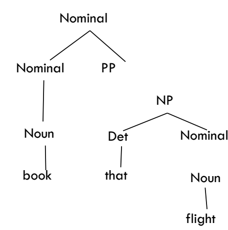
Bottom Up Parsing



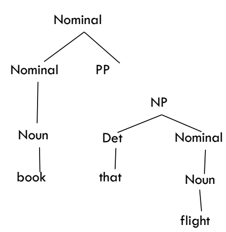
Bottom Up Parsing



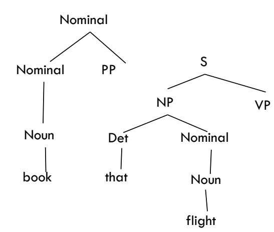
Bottom Up Parsing



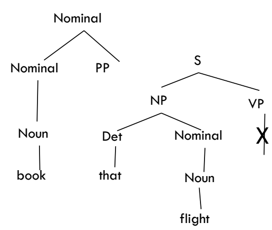
Bottom Up Parsing



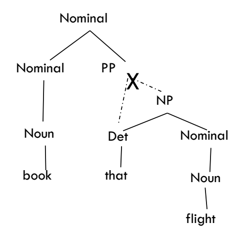
Bottom Up Parsing



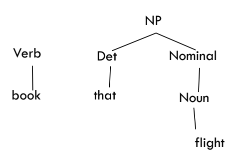
Bottom Up Parsing



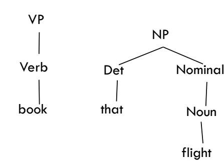
Bottom Up Parsing



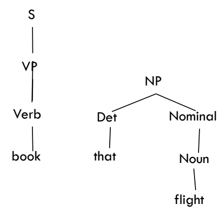
Bottom Up Parsing



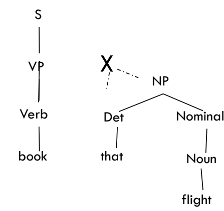
Bottom Up Parsing



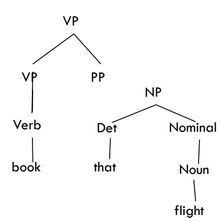
Bottom Up Parsing



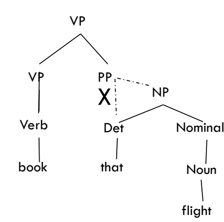
Bottom Up Parsing



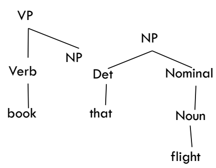
Bottom Up Parsing



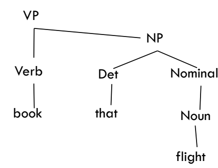
Bottom Up Parsing



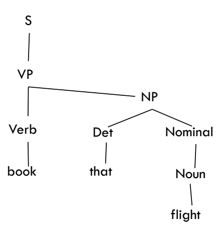
Bottom Up Parsing



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?

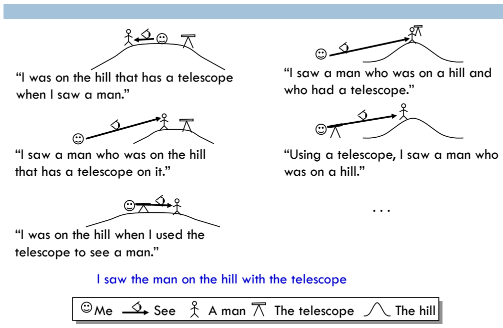
- Actual grammars are large
- Lots of ambiguity!
  - Most sentences have many parses
  - Some sentences have a lot of parses
  - Even for sentences that are not ambiguous, there is often ambiguity for subtrees (i.e. multiple ways to parse a phrase)

Why is parsing hard?

I saw the man on the hill with the telescope

What are some interpretations?

Structural Ambiguity Can Give Exponential Parses



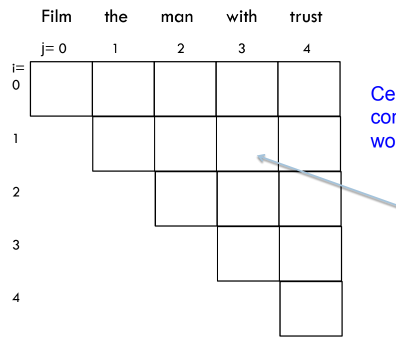
Dynamic Programming Parsing

- 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

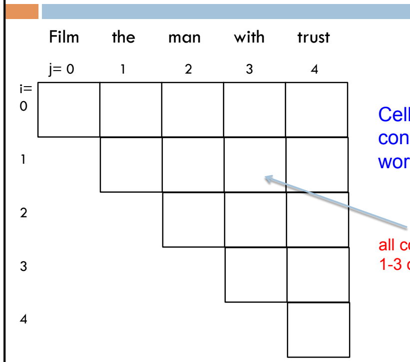
CKY parser: the chart



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

what does this cell represent?

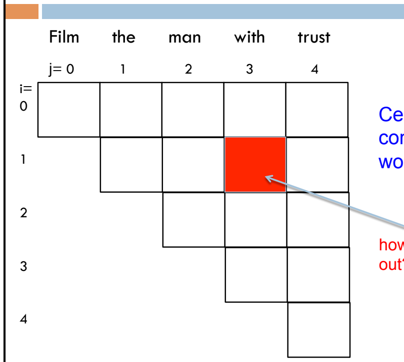
CKY parser: the chart



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

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

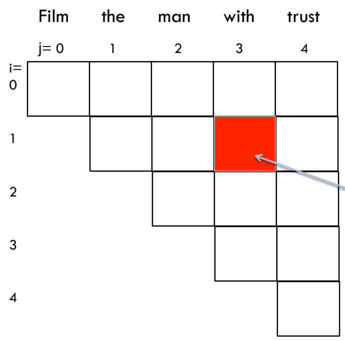
CKY parser: the chart



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

how could we figure this out?

CKY parser: the chart

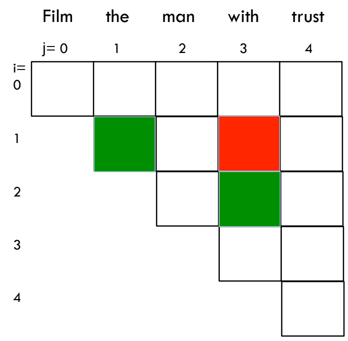


Cell[*i,j*] contains all constituents covering words *i* through *j*

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

VP -> VB NP
NP -> DT NN

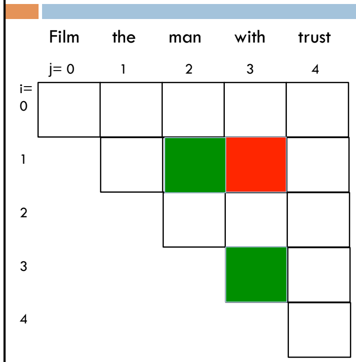
CKY parser: the chart



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"

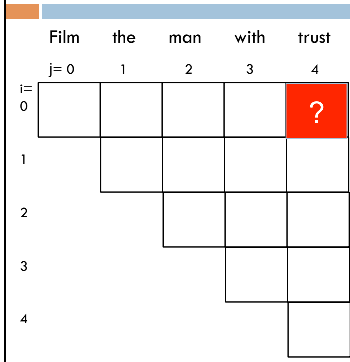
CKY parser: the chart



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

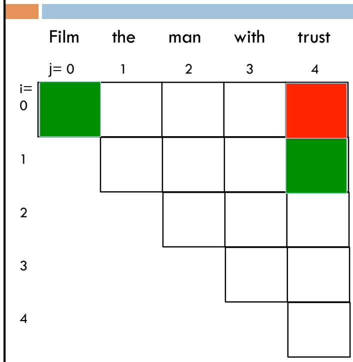
See if we can make a new constituent combining any for "the man" with any for "with"

CKY parser: the chart



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

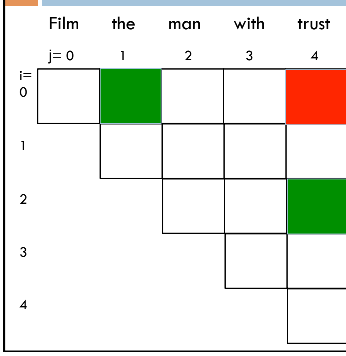
CKY parser: the chart



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"

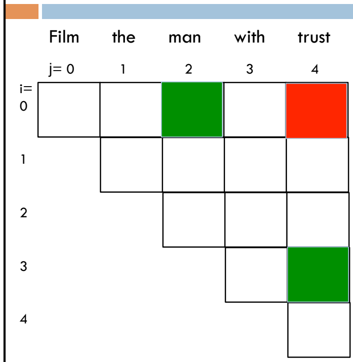
CKY parser: the chart



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"

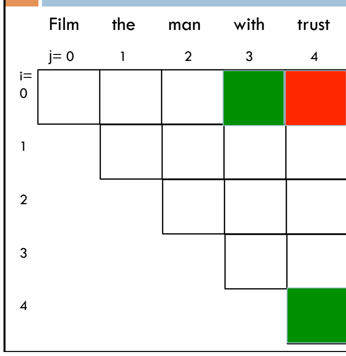
CKY parser: the chart



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"

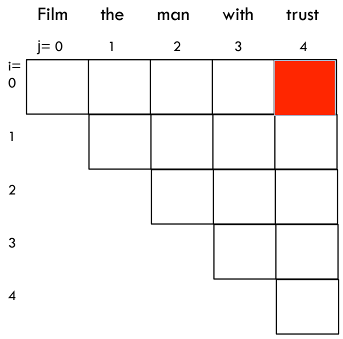
CKY parser: the chart



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"

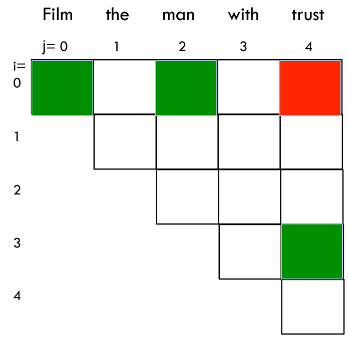
CKY parser: the chart



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

What if our rules weren't binary?

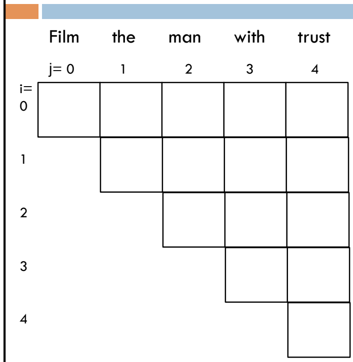
CKY parser: the chart



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"

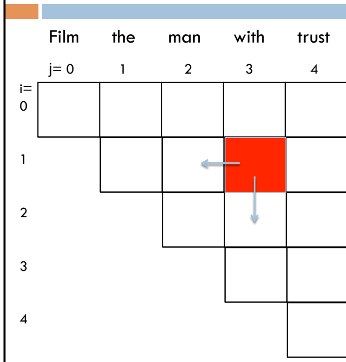
CKY parser: the chart



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

What order should we fill the entries in the chart?

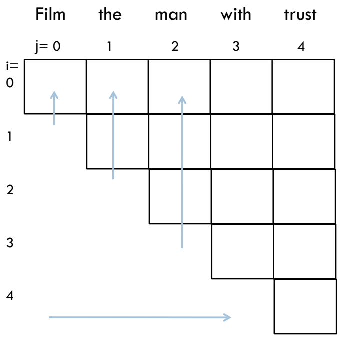
CKY parser: the chart



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

What order should we traverse the entries in the chart?

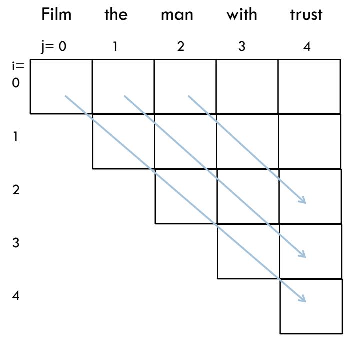
CKY parser: the chart



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*

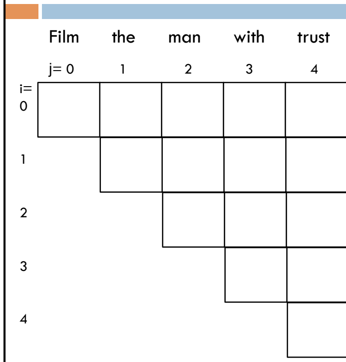
Top-left along the diagonals moving to the right

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



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