Seven Lectures on Statistical Parsing

Christopher Manning
LSA Linguistic Institute 2007
LSA 354
Lecture 2

Attendee information

Please put on a piece of paper:

- Name:
- Affiliation:
- "Status" (undergrad, grad, industry, prof, ...):
- Ling/CS/Stats background:
- What you hope to get out of the course:
- Whether the course has so far been too fast, too slow, or about right:

Assessment

Phrase structure grammars = context-free grammars

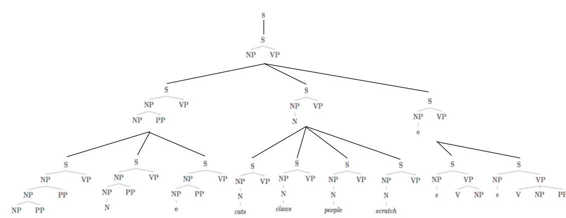
- G = (T, N, S, R)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
- A grammar G generates a language L.

A phrase structure grammar

- S → NP VP
- VP → V NP
- VP → V NP PP
- NP → NP PP
- NP → N
- NP → e
- NP → N N
- PP → P NP
- N → cats
- N → claws
- N → people
- N → scratch
- V → scratch
- P → with

- By convention, S is the start symbol, but in the PTB, we have an extra node at the top (ROOT, TOP)

Top-down parsing



Bottom-up parsing

- Bottom-up parsing is data directed
- The initial goal list of a bottom-up parser is the string to be parsed. If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
- Parsing is finished when the goal list contains just the start category.
- If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering.
- The standard presentation is as *shift-reduce parsing*.

Shift-reduce parsing: one path

	cats scratch people with claws	
cats	*scratch people with claws*	SHIFT
N	*scratch people with claws*	REDUCE
NP	*scratch people with claws*	REDUCE
NP *scratch*	*people with claws*	SHIFT
NP V	*people with claws*	REDUCE
NP V *people*	*with claws*	SHIFT
NP V N	*with claws*	REDUCE
NP V NP	*with claws*	REDUCE
NP V NP *with*	*claws*	SHIFT
NP V NP P	*claws*	REDUCE
NP V NP P *claws*		SHIFT
NP V NP P N		REDUCE
NP V NP P NP		REDUCE
NP V NP PP		REDUCE
NP VP		REDUCE
S		REDUCE

What other search paths are there for parsing this sentence?

Soundness and completeness

- A parser is *sound* if every parse it returns is valid/correct
- A parser *terminates* if it is guaranteed to not go off into an infinite loop
- A parser is *complete* if for any given grammar and sentence, it is sound, produces every valid parse for that sentence, and terminates
- (For many purposes, we settle for sound but incomplete parsers: e.g., probabilistic parsers that return a *k*-best list.)

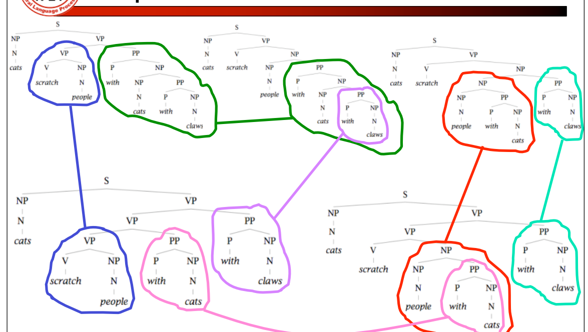
Problems with bottom-up parsing

- Unable to deal with empty categories: termination problem, unless rewriting empties as constituents is somehow restricted (but then it's generally incomplete)
- Useless work: locally possible, but globally impossible.
- Inefficient when there is great lexical ambiguity (grammar-driven control might help here)
- Conversely, it is data-directed: it attempts to parse the words that are there.
- **Repeated work:** anywhere there is common substructure

Problems with top-down parsing

- Left recursive rules
- A top-down parser will do badly if there are many different rules for the same LHS. Consider if there are 600 rules for S, 599 of which start with NP, but one of which starts with V, and the sentence starts with V.
- Useless work: expands things that are possible top-down but not there
- Top-down parsers do well if there is useful grammar-driven control: search is directed by the grammar
- Top-down is hopeless for rewriting parts of speech (preterminals) with words (terminals). In practice that is always done bottom-up as lexical lookup.
- **Repeated work:** anywhere there is common substructure

Repeated work…



Principles for success: take 1

- If you are going to do parsing-as-search with a grammar as is:
  - Left recursive structures must be found, not predicted
  - Empty categories must be predicted, not found
- Doing these things doesn't fix the repeated work problem:
  - Both TD (LL) and BU (LR) parsers can (and frequently do) do work exponential in the sentence length on NLP problems.

Principles for success: take 2

- Grammar transformations can fix both left-recursion and epsilon productions
- Then you parse the same language but with different trees
- Linguists tend to hate you
  - But this is a misconception: they shouldn't
  - You can fix the trees post hoc:
    - The transform-parse-detransform paradigm

Principles for success: take 3

- Rather than doing parsing-as-search, we do parsing as dynamic programming
- This is the most standard way to do things
  - Q.v. CKY parsing, next time
- It solves the problem of doing repeated work
- But there are also other ways of solving the problem of doing repeated work
  - Memoization (remembering solved subproblems)
    - Also, next time
  - Doing graph-search rather than tree-search.

Human parsing

- Humans often do ambiguity maintenance
  - *Have the police … eaten their supper?*
  - *come in and look around.*
  - *taken out and shot.*
- But humans also commit early and are "garden pathed":
  - *The man who hunts ducks out on weekends.*
  - *The cotton shirts are made from grows in Mississippi.*
  - *The horse raced past the barn fell.*

Polynomial time parsing of PCFGs

Probabilistic or stochastic context-free grammars (PCFGs)

- G = (T, N, S, R, P)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
    - P(R) gives the probability of each rule.

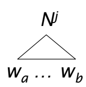
$$\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

- A grammar G generates a language model L.

$$\sum_{\gamma \in T^*} P(\gamma) = 1$$

PCFGs – Notation

- $w_{1n} = w_1 \ldots w_n$ = the word sequence from 1 to n (sentence of length n)
- w_{ab} = the subsequence $w_a \ldots w_b$
- N^j_{ab} = the nonterminal N^j dominating $w_a \ldots w_b$



- We'll write $P(N^i \rightarrow \zeta^j)$ to mean $P(N^i \rightarrow \zeta^j \mid N^i)$
- We'll want to calculate $\max_t P(t \Rightarrow^* w_{ab})$

The probability of trees and strings

- $P(t)$ -- The probability of tree is the product of the probabilities of the rules used to generate it.
- $P(w_{1n})$ -- The probability of the string is the sum of the probabilities of the trees which have that string as their yield

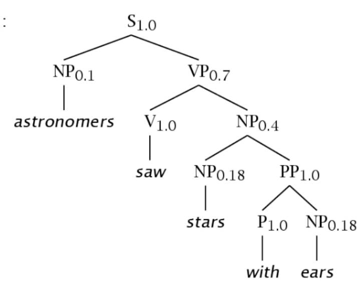
$P(w_{1n}) = \Sigma_j P(w_{1n}, t)$ where t is a parse of w_{1n}

$= \Sigma_j P(t)$

A Simple PCFG (in CNF)

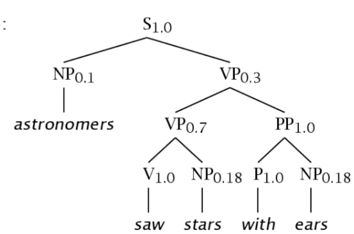
S	→	NP VP	1.0	NP	→	NP PP	0.4
VP	→	V NP	0.7	NP	→	*astronomers*	0.1
VP	→	VP PP	0.3	NP	→	*ears*	0.18
PP	→	P NP	1.0	NP	→	*saw*	0.04
P	→	*with*	1.0	NP	→	*stars*	0.18
V	→	*saw*	1.0	NP	→	*telescope*	0.1

t_1:



312

t_2:



313

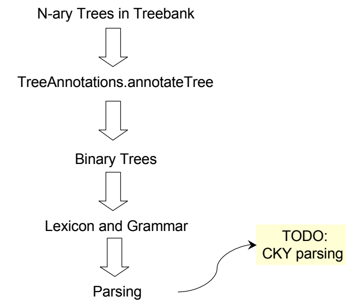
Tree and String Probabilities

- w_{15} = *astronomers saw stars with ears*
- $P(t_1)$ = 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18
  = 0.0009072
- $P(t_2)$ = 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18
  = 0.0006804
- $P(w_{15})$ = $P(t_1)$ + $P(t_2)$
  = 0.0009072 + 0.0006804
  = 0.0015876

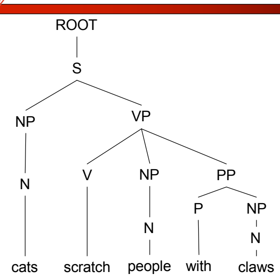
Chomsky Normal Form

- All rules are of the form X → Y Z or X → w.
- A transformation to this form doesn't change the weak generative capacity of CFGs.
  - With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
  - Unaries/empties are removed recursively
  - N-ary rules introduce new nonterminals:
    - VP → V NP PP becomes VP → V @VP-V and @VP-V → NP PP
- In practice it's a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries can be trickier
- But it makes parsing easier/more efficient

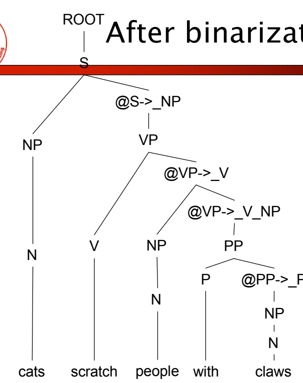
Treebank binarization

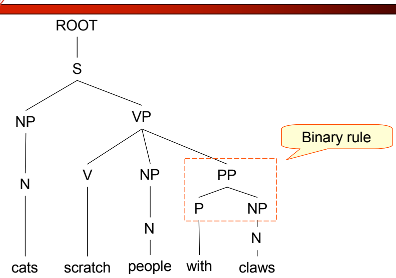


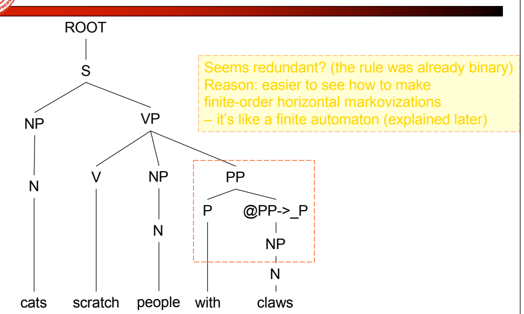
An example: before binarization…

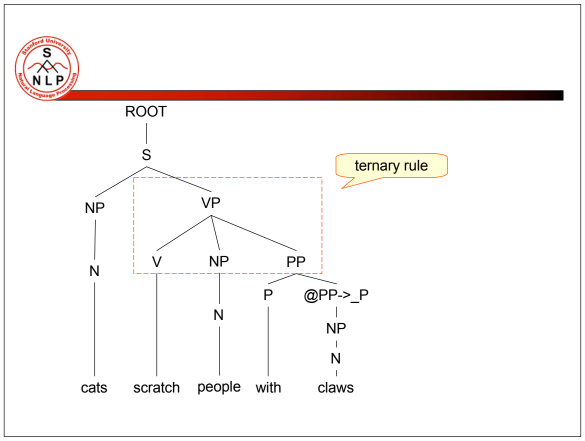


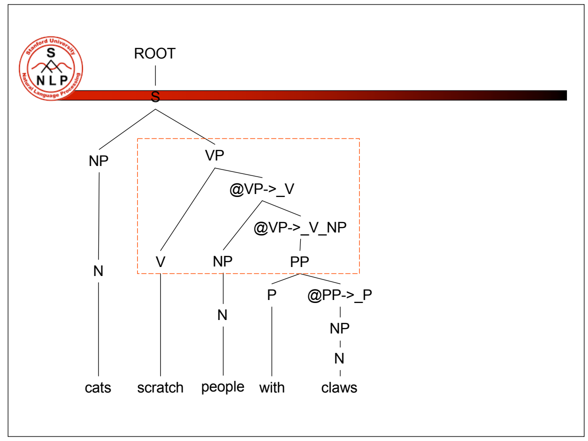
After binarization..

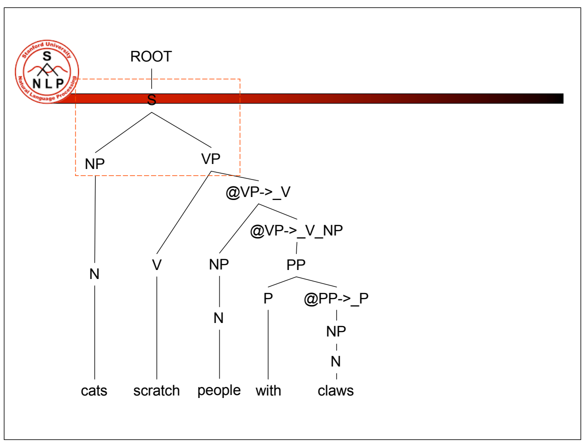


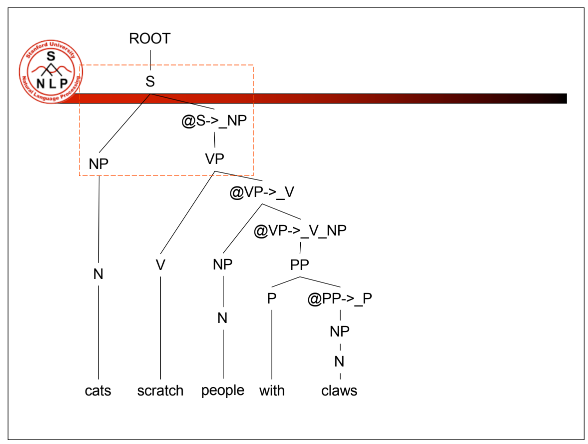


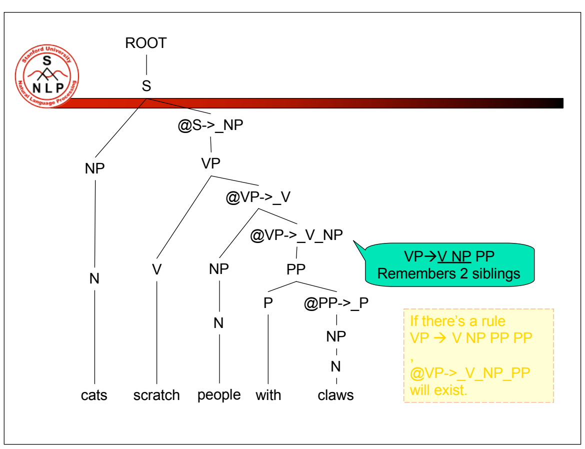




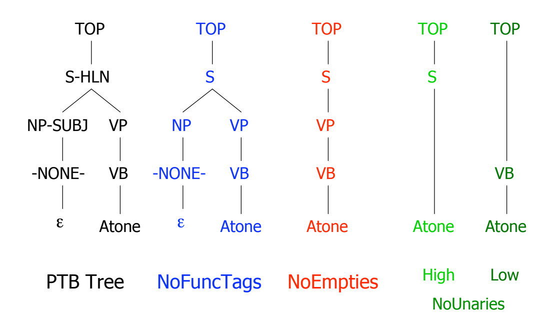








Treebank: empties and unaries

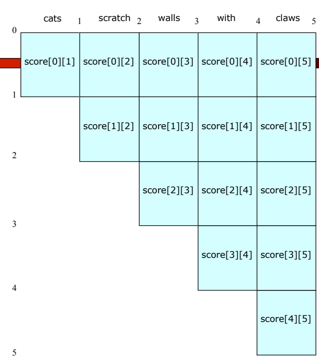


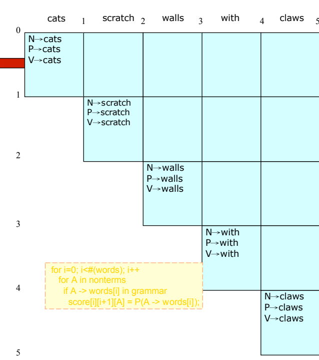
The CKY algorithm (1960/1965)

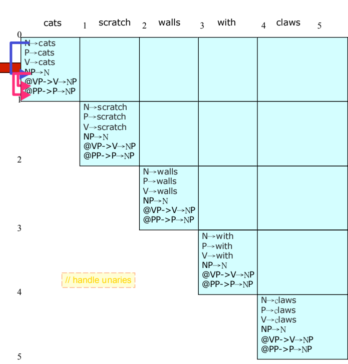
```
function CKY(words, grammar) returns most probable parse/prob
  score = new double[#(words)+1][#(words)+][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]]
  for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i])
    //handle unaries
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
          prob = P(A->B)*score[i][i+1][B]
          if(prob > score[i][i+1][A])
            score[i][i+1][A] = prob
            back[i][i+1] [A] = B
            added = true
```

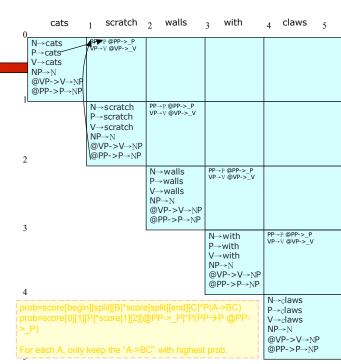
The CKY algorithm (1960/1965)

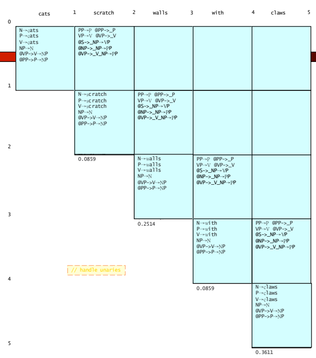
```
for span = 2 to #(words)
  for begin = 0 to #(words)- span
    end = begin + span
    for split = begin+1 to end-1
      for A,B,C in nonterms
        prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
        if(prob > score[begin][end][A])
          score[begin]end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
      //handle unaries
      boolean added = true
      while added
        added = false
        for A, B in nonterms
          prob = P(A->B)*score[begin][end][B];
          if(prob > score[begin][end] [A])
            score[begin][end] [A] = prob
            back[begin][end] [A] = B
            added = true
return buildTree(score, back)
```



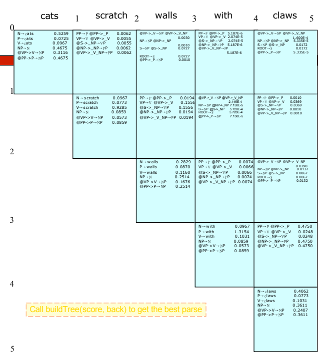







cats 1 scratch 2 walls 3 with 4 claws 5
0 1 2 3 4 5
N→cats
P→cats
V→cats
NP→N
@VP->V→NP
@PP->P→NP
PP→P @PP->_P
VP→V @VP->_V
@S->_NP→VP
@NP->_NP→PP
@VP->_V_NP→PP
N→scratch
P→scratch
V→scratch
NP→N
@VP->V→NP
@PP->P→NP
0.0859
N→walls
P→walls
V→walls
NP→N
@VP->V→NP
@PP->P→NP
0.2514
N→with
P→with
V→with
NP→N
@VP->V→NP
@PP->P→NP
0.0859
N→claws
P→claws
V→claws
NP→N
@VP->V→NP
@PP->P→NP
0.3611
// handle unaries

....


cats 1 scratch 2 walls 3 with 4 claws 5
0 1 2 3 4 5
N→cats 0.5259
P→cats 0.0725
V→cats 0.0967
NP→N 0.4675
@VP->V→NP 0.3116
@PP->P→NP 0.4675
PP→P @PP->_P 0.0062
VP→V @VP->_V 0.0055
@S->_NP→VP 0.0055
@NP->_NP→PP 0.0062
@VP->_V_NP→PP 0.0062
@VP->_V→NP @VP->_V_NP 0.0030
NP→NP @NP->_NP 0.0010
S→NP @S->_NP 0.0727
ROOT→S 0.0727
@PP->_P→NP 0.0010
PP→P @PP->_P 5.187E-6
VP→V @VP->_V 2.074E-5
@S->_NP→VP 2.074E-5
@NP->_NP→PP 5.187E-6
@VP->_V_NP→PP 5.187E-6
@VP->_V→NP @VP->_V_NP 1.600E-4
NP→NP @NP->_NP 5.335E-5
S→NP @S->_NP 0.0172
ROOT→S 0.0172
@PP->_P→NP 5.335E-5
N→scratch 0.0967
P→scratch 0.0773
V→scratch 0.9285
NP→N 0.0859
@VP->V→NP 0.0573
@PP->P→NP 0.0859
PP→P @PP->_P 0.0194
VP→V @VP->_V 0.1556
@S->_NP→VP 0.1556
@NP->_NP→PP 0.0194
@VP->_V_NP→PP 0.0194
@VP->_V→NP @VP->_V_NP 2.145E-4
NP→NP @NP->_NP 7.150E-5
S→NP @S->_NP 5.720E-4
ROOT→S 5.720E-4
@PP->_P→NP 7.150E-5
PP→P @PP->_P 0.0010
VP→V @VP->_V 0.0369
@S->_NP→VP 0.0369
@NP->_NP→PP 0.0010
@VP->_V_NP→PP 0.0010
N→walls 0.2829
P→walls 0.0870
V→walls 0.1160
NP→N 0.2514
@VP->V→NP 0.1676
@PP->P→NP 0.2514
PP→P @PP->_P 0.0074
VP→V @VP->_V 0.0066
@S->_NP→VP 0.0066
@NP->_NP→PP 0.0074
@VP->_V_NP→PP 0.0074
@VP->_V→NP @VP->_V_NP 0.0398
NP→NP @NP->_NP 0.0132
S→NP @S->_NP 0.0062
ROOT→S 0.0062
@PP->_P→NP 0.0132
N→with 0.0967
P→with 1.3154
V→with 0.1031
NP→N 0.0859
@VP->V→NP 0.0573
@PP->P→NP 0.0859
PP→P @PP->_P 0.4750
VP→V @VP->_V 0.0248
@S->_NP→VP 0.0248
@NP->_NP→PP 0.4750
@VP->_V_NP→PP 0.4750
N→claws 0.4062
P→claws 0.0773
V→claws 0.1031
NP→N 0.3611
@VP->V→NP 0.2407
@PP->P→NP 0.3611
Call buildTree(score, back) to get the best parse