# CS371N: Natural Language Processing
# Lecture 17: Parsing II

Greg Durrett
TEXAS
The University of Texas at Austin

## Announcements

- A4 due today
- Midterm Thursday:
  - One 8.5"x11" notesheet
  - No calculators
  - Multiple choice, short-answer, and long-answer

## Recap: PCFGs

Grammar (CFG)

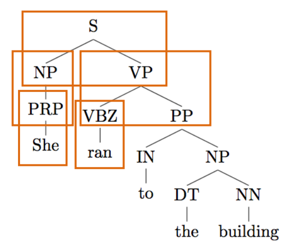
| | | | |
|---|---|---|---|
| ROOT → S | 1.0 | NP → NP PP | 0.3 |
| S → NP VP | 1.0 | VP → VBP NP | 0.7 |
| NP → DT NN | 0.2 | VP → VBP NP PP | 0.3 |
| NP → NN NNS | 0.5 | PP → IN NP | 1.0 |

Lexicon

| | |
|---|---|
| NN → interest | 1.0 |
| NNS → raises | 1.0 |
| VBP → interest | 1.0 |
| VBZ → raises | 1.0 |

- Context-free grammar: symbols which rewrite as one or more symbols
- Lexicon consists of "preterminals" (POS tags) rewriting as terminals (words)
- CFG is a tuple (N, T, S, R): N = nonterminals, T = terminals, S = start symbol (generally a special ROOT symbol), R = rules
- PCFG: probabilities associated with rewrites, normalize by source symbol

## Recap: Learning PCFGs




→

| | |
|---|---|
| S → NP VP | 1.0 |
| NP → PRP | 0.5 |
| NP → DT NN | 0.5 |
| ... | |

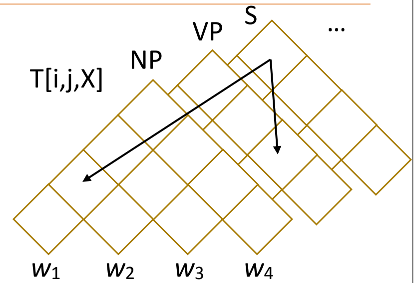
- Maximum likelihood PCFG for a set of labeled trees: count and normalize! Same as HMMs / Naive Bayes

# Recap: CKY

- Chart: T[i,j,X] = best score for X over (i, j)
- Base: T[i,i+1,X] = log P(X → $w_i$)
- Loop over all split points k, apply rules X -> Y Z to build X in every possible way
- Recurrence:

$$T[i,j,X] = \max_{k} \max_{r: X \to X1\ X2} T[i,k,X1] + T[k,j,X2] + \log P(X \to X1\ X2)$$

- Runtime: $O(n^3G)$  G = grammar constant




S[0,4] => NP[0,2] VP[2,4]

# Parser Evaluation

# Parser Evaluation

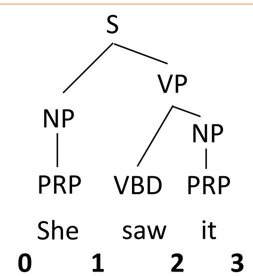
- View a parse as a set of labeled *brackets* / constituents

S(0,3)

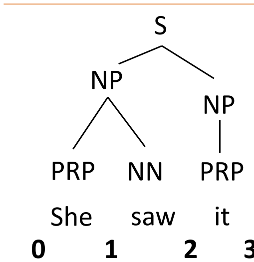
NP(0,1)

PRP(0,1) (but standard evaluation *does not count POS tags*)

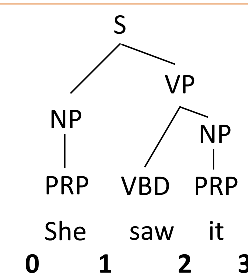
VP(1,3), VBD(1,2), NP(2,3), PRP(2,3)




# Parser Evaluation




**S(0,3)**,
**NP(0,2)**,
**NP(2,3)**,
~~PRP(0,1),~~
~~NN(1,2),~~
~~PRP(2,3)~~




**S(0,3)**,
**NP(0,1)**,
**VP(1,3)**,
**NP(2,3)**,
~~PRP(0,1),~~
~~VBD(1,2),~~
~~PRP(2,3)~~

- Precision: number of correct predictions / number of predictions = 2/3
- Recall: number of correct predictions / number of golds = 2/4
- F1: harmonic mean of precision and recall = $(1/2 * ((2/4)^{-1} + (2/3)^{-1}))^{-1}$

= 0.57 (closer to min)

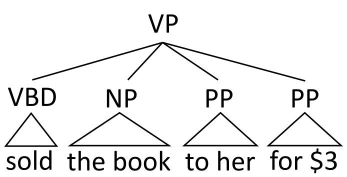
# Results

- Standard dataset for English: Penn Treebank (Marcus et al., 1993)
- “Vanilla” PCFG: ~71 F1
- Best PCFGs for English: ~90 F1
- State-of-the-art discriminative models (using unlabeled data): 95 F1
- Other languages: results vary widely depending on annotation + complexity of the grammar

# Grammar Preprocessing

# Binarization

- To parse efficiently, we need our PCFGs to be at most binary (not CNF)




$P(VP \rightarrow VBD\ NP\ PP\ PP) = 0.2$

$P(VP \rightarrow VBZ\ PP) = 0.1$

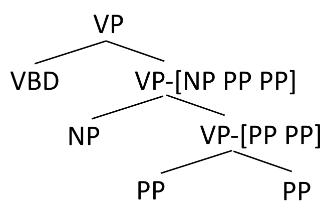
...

- Solution: transform the trees. Introduce intermediate special symbols that rewrite deterministically

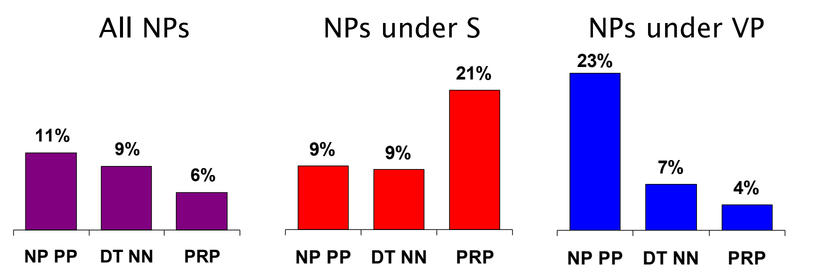
$P(VP \rightarrow VBD\ VP\text{-}[NP\ PP\ PP]) = 0.2$

$P(VP\text{-}[NP\ PP\ PP] \rightarrow NP\ VP\text{-}[PP\ PP]) = 1.0$

$P(VP\text{-}[PP\ PP] \rightarrow PP\ PP) = 1.0$

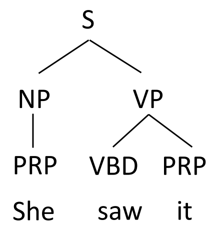



# PCFG Independence Assumptions

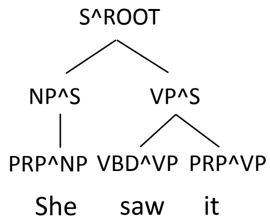



- Language is not context-free: NPs in different contexts rewrite differently
- $[They]_{NP}$ received $[the\ package\ of\ books]_{NP}$

# Vertical Markovization




Basic tree (v = 1)

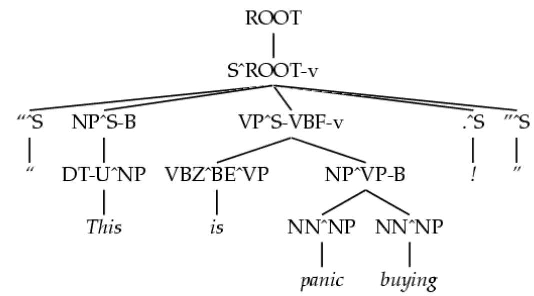



v = 2 Markovization

- Why is this a good idea?

# Annotated Tree

- Augment the grammar: deterministically transform symbols to be “less context free” (binarization not shown here)



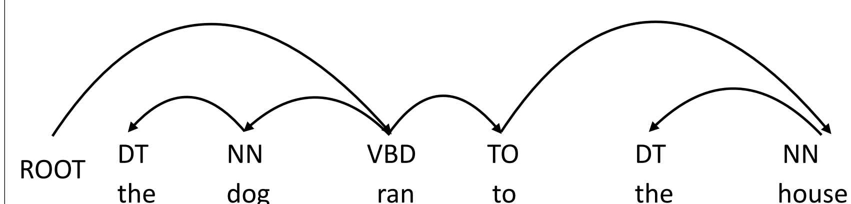

- 75 F1 with basic PCFG => 86.3 F1 with this highly customized PCFG (SOTA was 90 F1 at the time, but with more complex methods)

Klein and Manning (2003)

# Dependency Parsing

# Dependency Parsing

- Dependencies: syntactic structure is defined by relations between words
  - Head (parent, governor) connected to dependent (child, modifier)
  - Each word has exactly one parent except for the ROOT symbol, dependencies must form a directed acyclic graph



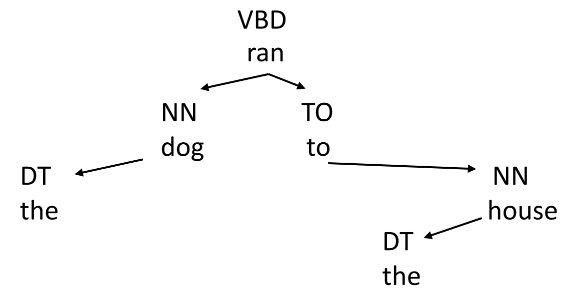

- POS tags same as before, usually run a tagger first as preprocessing

# Why are they defined this way?

- Constituency tests:
  - Substitution by *proform*: the dog *did so [ran to the house]*, *he [the dog]* ran to the house
  - Clefting (*It was [to the house] that the dog ran…*)
- Dependency: verb is the root of the clause, everything else follows from that
  - No notion of a VP!

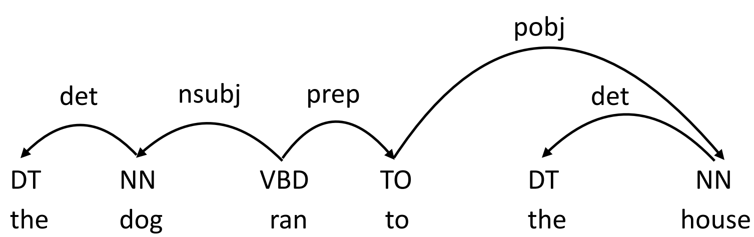
# Dependency Parsing

- Still a notion of hierarchy! Subtrees often align with constituents



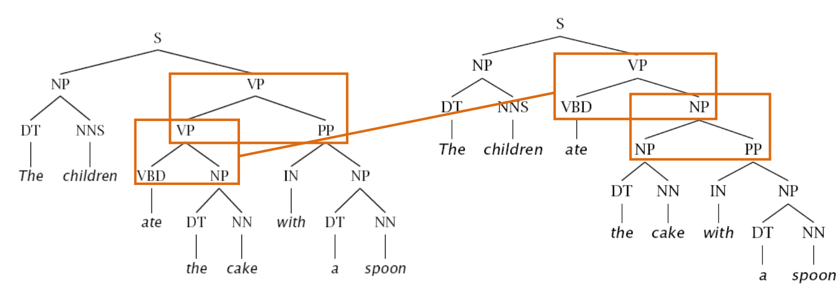

# Dependency Parsing

- Can label dependencies according to syntactic function
- Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)




# Dependency vs. Constituency: PP Attachment

- Constituency: several rule productions need to change



# Dependency vs. Constituency: PP Attachment

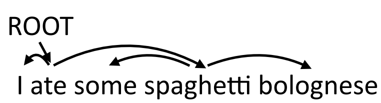
- Dependency: one word (with) assigned a different parent


- corenlp.run: *spoon* is child instead of *with*. This is just a different formalism
- More predicate-argument focused view of syntax
- "What's the main verb of the sentence? What is its subject and object?" — easier to answer under dependency parsing

# Parsers Today

# Modern Parsers

- Shift-reduce parsers: parsers that construct a tree from a sentence via a greedy sequence of operations. similar to parsing algorithms for compilers:



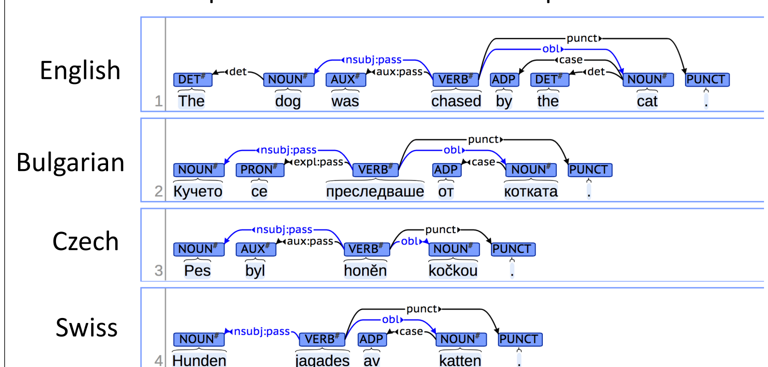

Shift, Shift, Left-arc, Shift, Shift, Left-arc, Shift, Right-arc, Right-arc, Right-arc

I <- ate | some <- spaghetti | spaghetti -> bolognese | ate -> spaghetti | ROOT -> ate

- These parsers historically worked less well. But with neural networks, they're pretty good and **very** fast!

# Universal Dependencies

- Annotate dependencies with the same representation in many languages




http://universaldependencies.org/

# Reflections on Structure

- What is the role of it now?
- Systems still make these kinds of judgments, just not explicitly
- To improve systems, do we need to understand what they do?