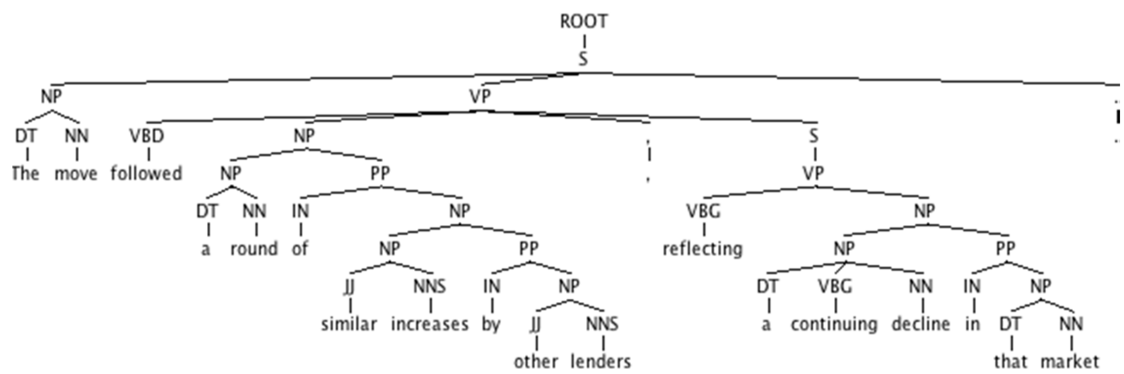
Natural Language Processing

Parsing I

Dan Klein – UC Berkeley

Syntax

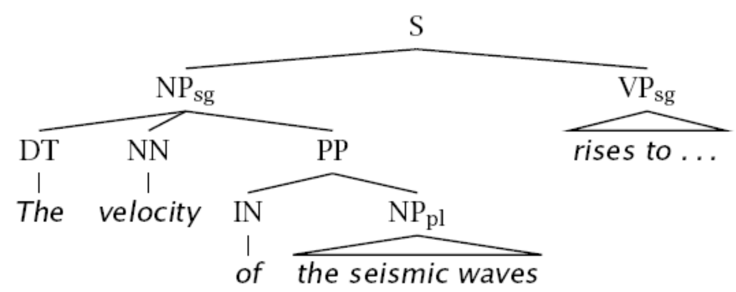
Parse Trees



The move followed a round of similar increases by other lenders, reflecting a continuing decline in that market

Phrase Structure Parsing

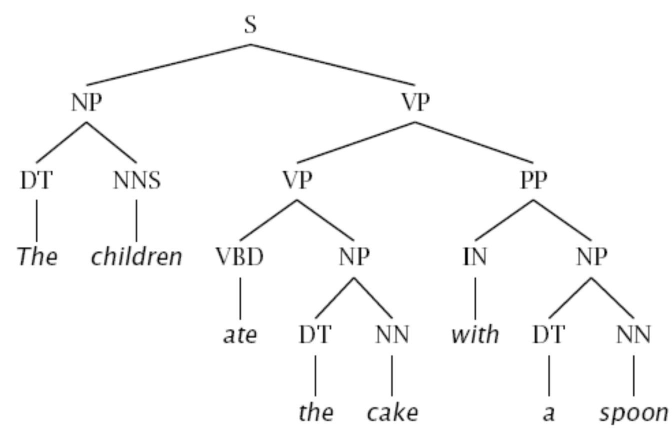
- Phrase structure parsing organizes syntax into *constituents* or *brackets*
- In general, this involves nested trees
- Linguists can, and do, argue about details
- Lots of ambiguity
- Not the only kind of syntax…



new art critics write reviews with computers

Constituency Tests

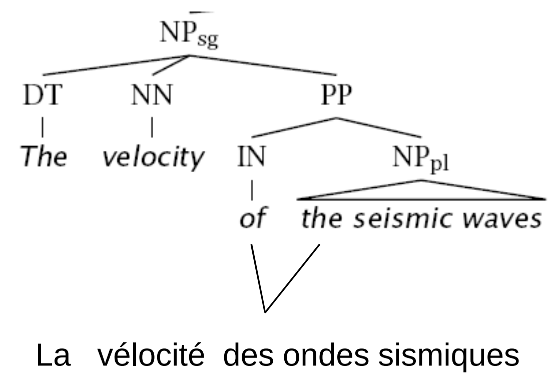
- How do we know what nodes go in the tree?
- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic gounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction



- Cross-linguistic arguments, too

Conflicting Tests

- Constituency isn't always clear
  - Units of transfer:
    - think about ~ penser à
    - talk about ~ hablar de
  - Phonological reduction:
    - I will go → I'll go
    - I want to go → I wanna go
    - a le centre → au centre
  - Coordination
    - He went to and came from the store.



Classical NLP: Parsing

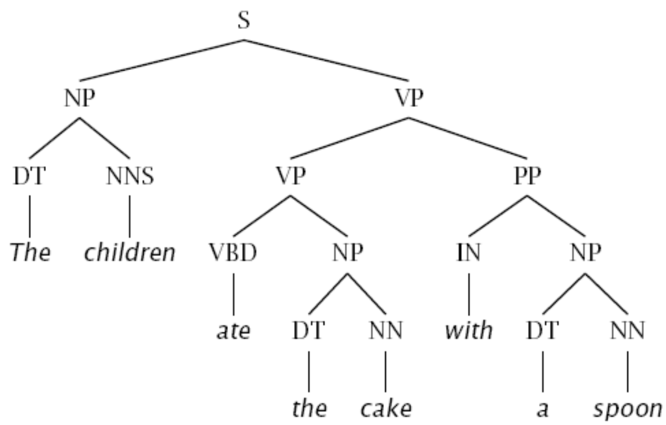
- Write symbolic or logical rules:

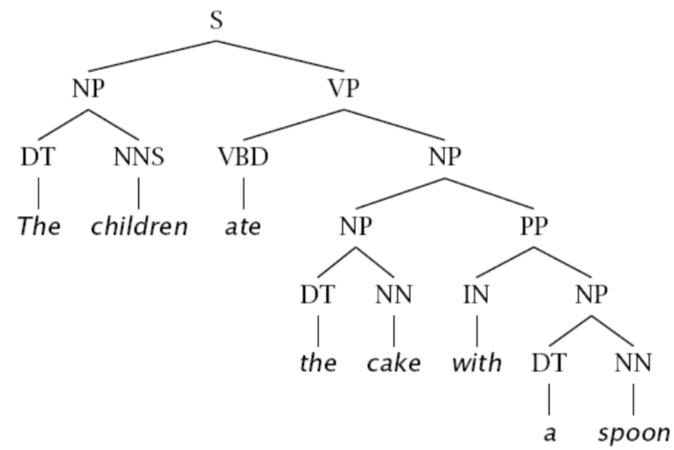
Grammar (CFG)		Lexicon
ROOT → S	NP → NP PP	NN → interest
S → NP VP	VP → VBP NP	NNS → raises
NP → DT NN	VP → VBP NP PP	VBP → interest
NP → NN NNS	PP → IN NP	VBZ → raises
		…

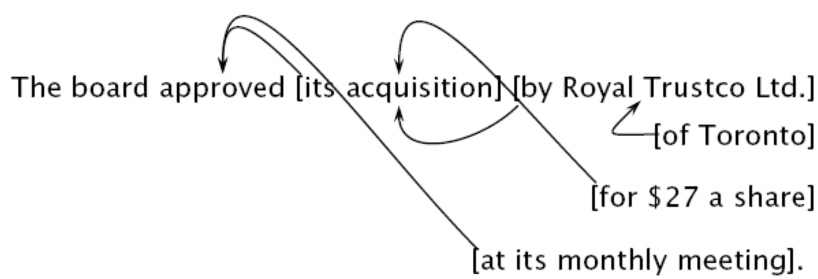
- Use deduction systems to prove parses from words
  - Minimal grammar on “Fed raises” sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses
- This scaled very badly, didn’t yield broad-coverage tools

Ambiguities

Ambiguities: PP Attachment







Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink

Syntactic Ambiguities I

- Prepositional phrases:
  They cooked the beans in the pot on the stove with handles.

- Particle vs. preposition:
  The puppy tore up the staircase.

- Complement structures
  The tourists objected to the guide that they couldn’t hear.
  She knows you like the back of her hand.

- Gerund vs. participial adjective
  Visiting relatives can be boring.
  Changing schedules frequently confused passengers.

Syntactic Ambiguities II

- Modifier scope within NPs
  impractical design requirements
  plastic cup holder

- Multiple gap constructions
  The chicken is ready to eat.
  The contractors are rich enough to sue.

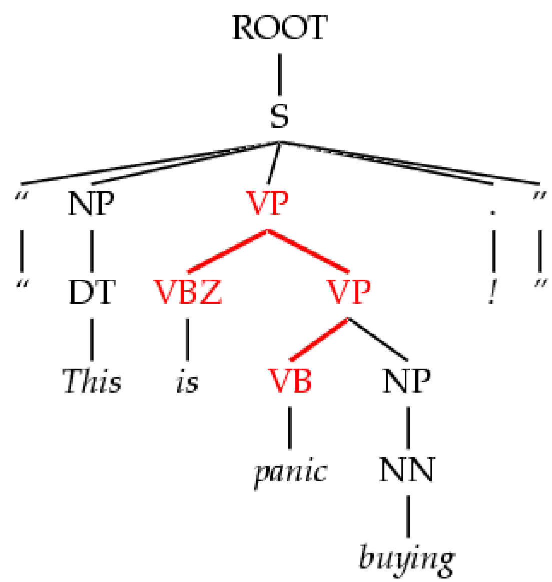
- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.

Dark Ambiguities

- *Dark ambiguities*: most analyses are shockingly bad (meaning, they don't have an interpretation you can get your mind around)

This analysis corresponds to the correct parse of

"This will panic buyers ! "



- Unknown words and new usages
- Solution: We need mechanisms to focus attention on the best ones, probabilistic techniques do this

PCFGs

Probabilistic Context-Free Grammars

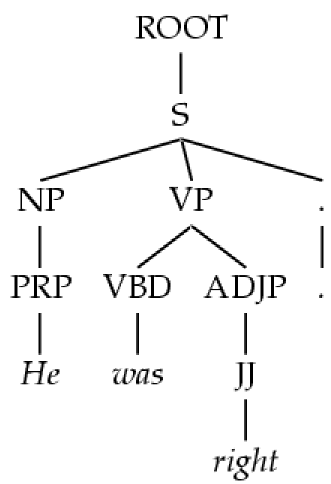
- A context-free grammar is a tuple <*N, T, S, R*>
  - *N* : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - *T* : the set of terminals (the words)
  - *S* : the start symbol
    - Often written as ROOT or TOP
    - *Not* usually the sentence non-terminal S
  - *R* : the set of rules
    - Of the form $X \rightarrow Y_1 Y_2 \ldots Y_k$, with $X, Y_i \in N$
    - Examples: $S \rightarrow NP\ VP$, $VP \rightarrow VP\ CC\ VP$
    - Also called rewrites, productions, or local trees
- A PCFG adds:
  - A top-down production probability per rule $P(Y_1 Y_2 \ldots Y_k \mid X)$

Treebank Sentences

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (PP against
                         (NP Arizona real estate loans)))))
         ,
         (S-ADV (NP-SBJ *)
                (VP reflecting
                    (NP (NP a continuing decline)
                        (PP-LOC in
                                (NP that market))))))
     .))
```

Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):



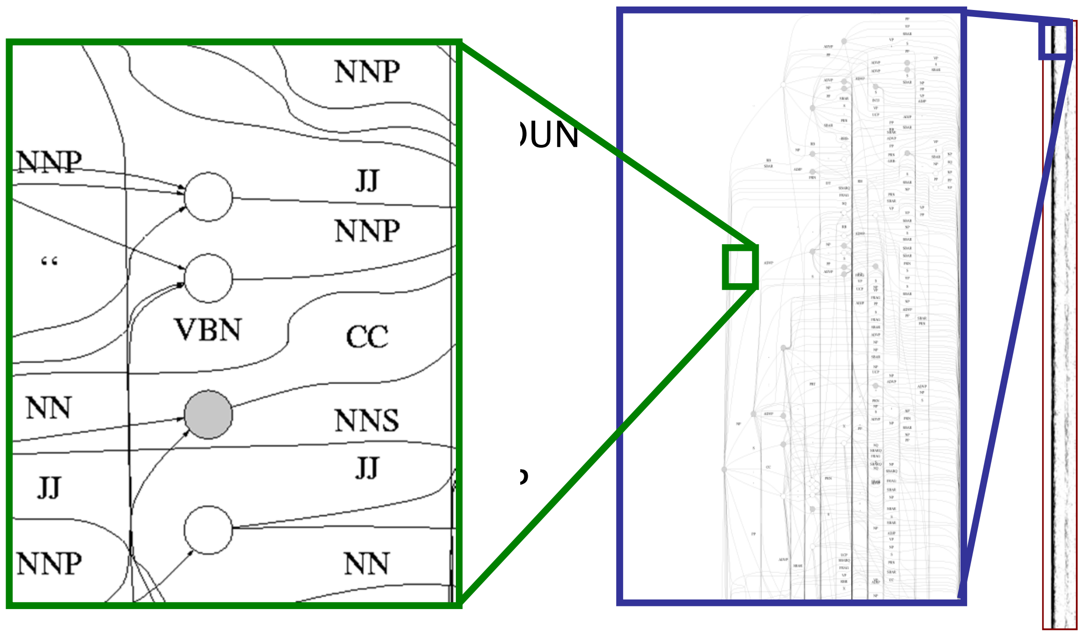
ROOT → S	1
S → NP VP .	1
NP → PRP	1
VP → VBD ADJP	1
…..	

- Better results by enriching the grammar (e.g., lexicalization).
- Can also get reasonable parsers without lexicalization.

Treebank Grammar Scale

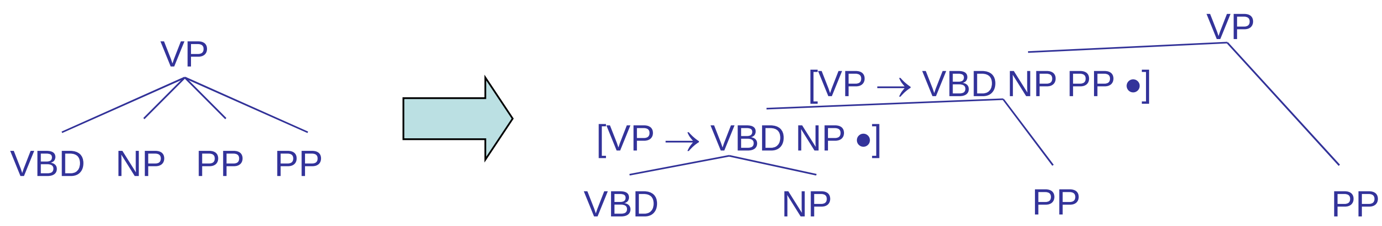
- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller

NP



Chomsky Normal Form

- Chomsky normal form:
  - All rules of the form X → Y Z or X → w
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals



    - Unaries / empties are "promoted"
  - In practice it's kind of a pain:
    - Reconstructing n-aries is easy
    - Reconstructing unaries is trickier
    - The straightforward transformations don't preserve tree scores
  - Makes parsing algorithms simpler!

CKY Parsing

A Recursive Parser

```
bestScore(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max score(X->YZ) *
                   bestScore(Y,i,k) *
                   bestScore(Z,k,j)
```

- Will this parser work?
- Why or why not?
- Memory requirements?

A Memoized Parser

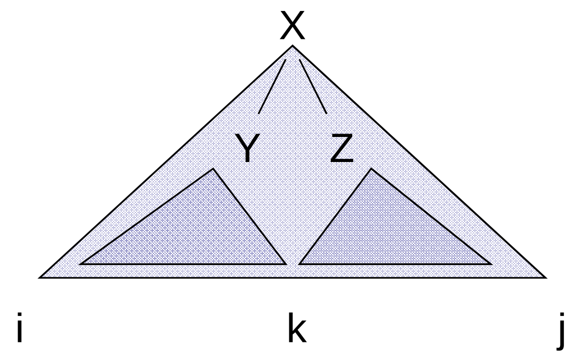
- One small change:

```
bestScore(X,i,j,s)
   if (scores[X][i][j] == null)
       if (j = i+1)
           score = tagScore(X,s[i])
       else
           score = max score(X->YZ) *
                         bestScore(Y,i,k) *
                         bestScore(Z,k,j)
       scores[X][i][j] = score
   return scores[X][i][j]
```

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```
bestScore(s)
   for (i : [0,n-1])
     for (X : tags[s[i]])
       score[X][i][i+1] =
           tagScore(X,s[i])
   for (diff : [2,n])
     for (i : [0,n-diff])
       j = i + diff
       for (X->YZ : rule)
         for (k : [i+1, j-1])
           score[X][i][j] = max score[X][i][j],
                                score(X->YZ) *
                                score[Y][i][k] *
                                score[Z][k][j]
```



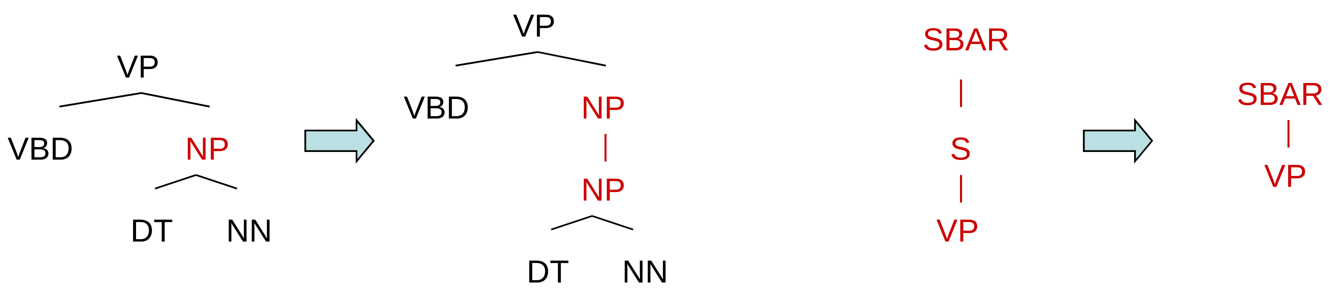
Unary Rules

- Unary rules?

```
bestScore(X,i,j,s)
   if (j = i+1)
        return tagScore(X,s[i])
   else
        return max max score(X->YZ) *
                       bestScore(Y,i,k) *
                       bestScore(Z,k,j)
                   max score(X->Y) *
                       bestScore(Y,i,j)
```

CNF + Unary Closure

- We need unaries to be non-cyclic
  - Can address by pre-calculating the *unary closure*
  - Rather than having zero or more unaries, always have exactly one



  - Alternate unary and binary layers
  - Reconstruct unary chains afterwards

Alternating Layers

```
bestScoreB(X,i,j,s)
      return max max score(X->YZ) *
                      bestScoreU(Y,i,k) *
                      bestScoreU(Z,k,j)



bestScoreU(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max max score(X->Y) *
                      bestScoreB(Y,i,j)
```

Analysis

Memory

- How much memory does this require?
  - Have to store the score cache
  - Cache size: |symbols|*n^2 doubles
  - For the plain treebank grammar:
    - X ~ 20K, n = 40, double ~ 8 bytes = ~ 256MB
    - Big, but workable.

- Pruning: Beams
  - score[X][i][j] can get too large (when?)
  - Can keep beams (truncated maps score[i][j]) which only store the best few scores for the span [i,j]

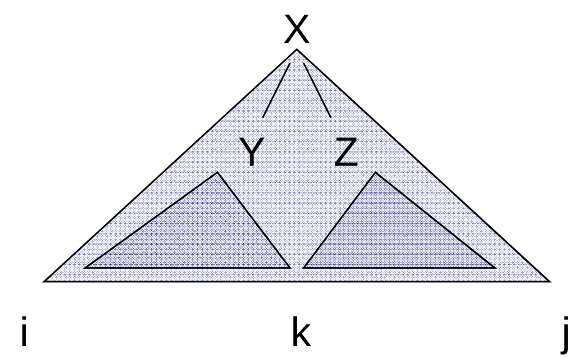
- Pruning: Coarse-to-Fine
  - Use a smaller grammar to rule out most X[i,j]
  - Much more on this later…

Time: Theory

- How much time will it take to parse?

  - For each diff (<= n)
    - For each i (<= n)
      - For each rule X → Y Z
        - For each split point k

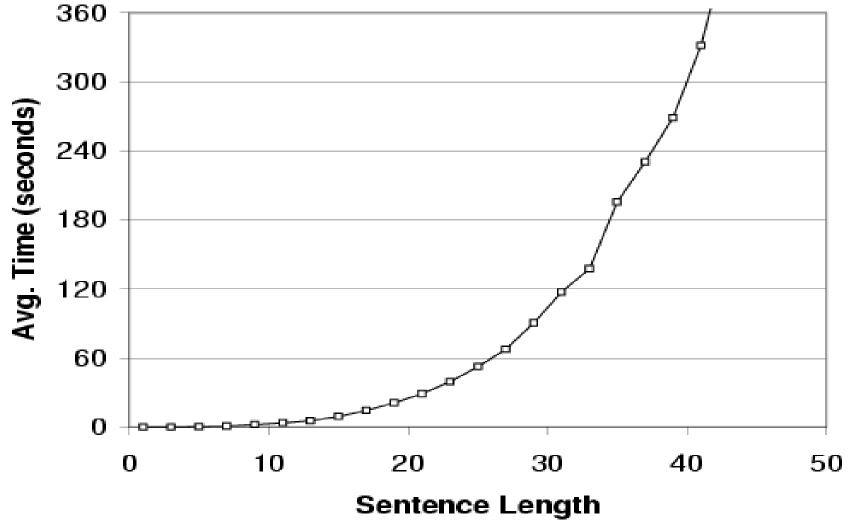
          Do constant work



  - Total time: |rules|*n^3
  - Something like 5 sec for an unoptimized parse of a 20-word sentences

Time: Practice

- Parsing with the vanilla treebank grammar:



~ 20K Rules

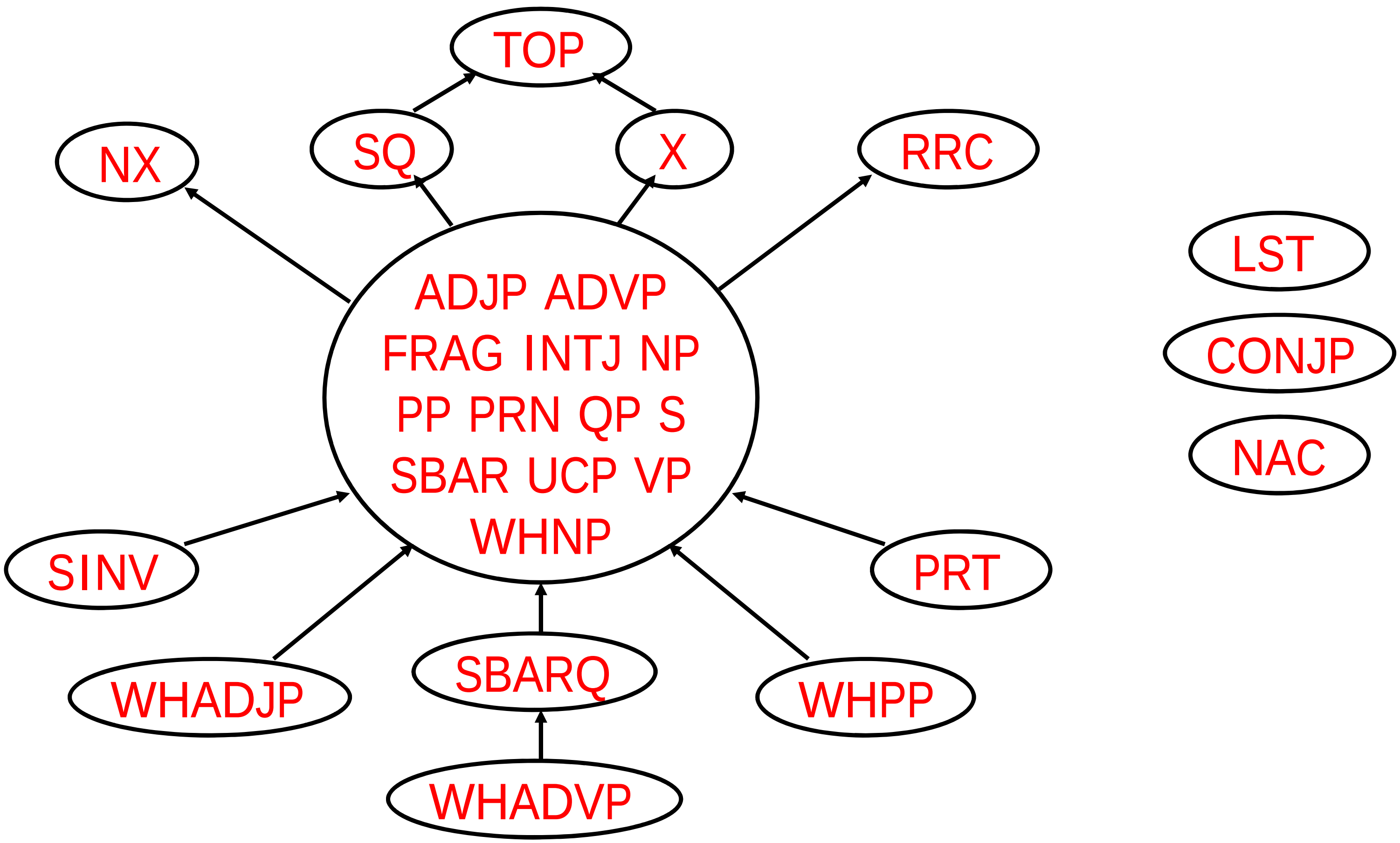
(not an optimized parser!)

Observed exponent:

3.6

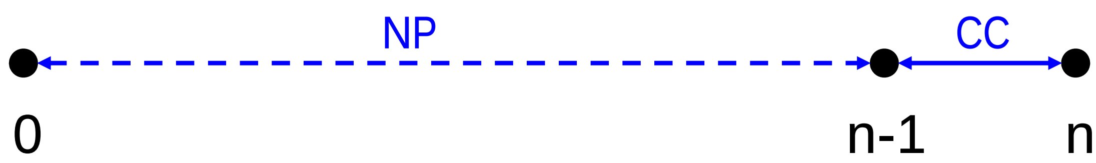
- Why's it worse in practice?
  - Longer sentences "unlock" more of the grammar
  - All kinds of systems issues don't scale

Same-Span Reachability



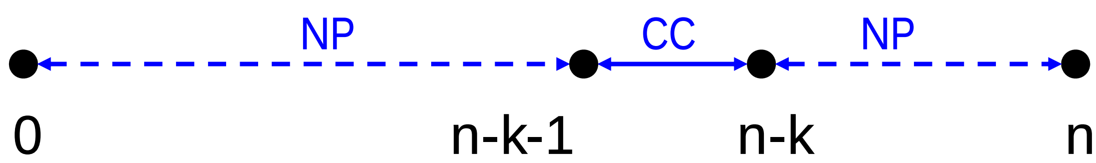
Rule State Reachability

Example: NP CC •



1 Alignment

Example: NP CC NP •



n Alignments

- Many states are more likely to match larger spans!

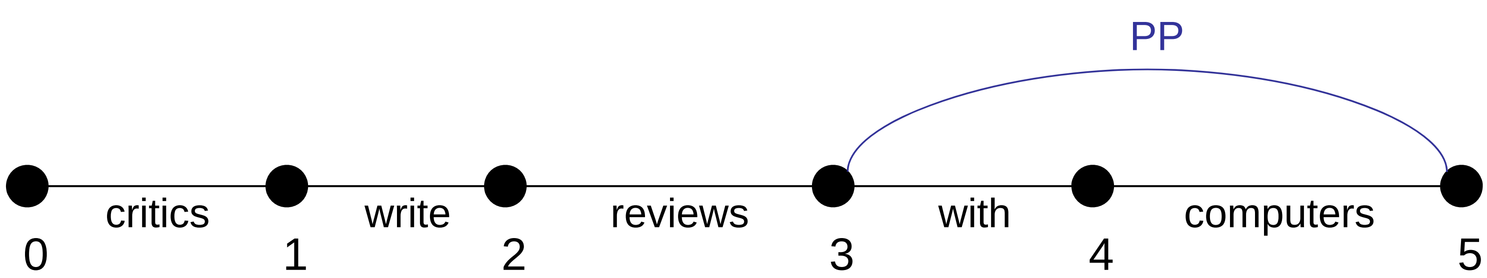
Efficient CKY

- Lots of tricks to make CKY efficient
  - Some of them are little engineering details:
    - E.g., first choose k, then enumerate through the Y:[i,k] which are non-zero, then loop through rules by left child.
    - Optimal layout of the dynamic program depends on grammar, input, even system details.
  - Another kind is more important (and interesting):
    - Many X:[i,j] can be suppressed on the basis of the input string
    - We'll see this next class as figures-of-merit, A* heuristics, coarse-to-fine, etc

Agenda-Based Parsing

Agenda-Based Parsing

- Agenda-based parsing is like graph search (but over a hypergraph)
- Concepts:
  - Numbering: we number fenceposts between words
  - "Edges" or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we've expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)



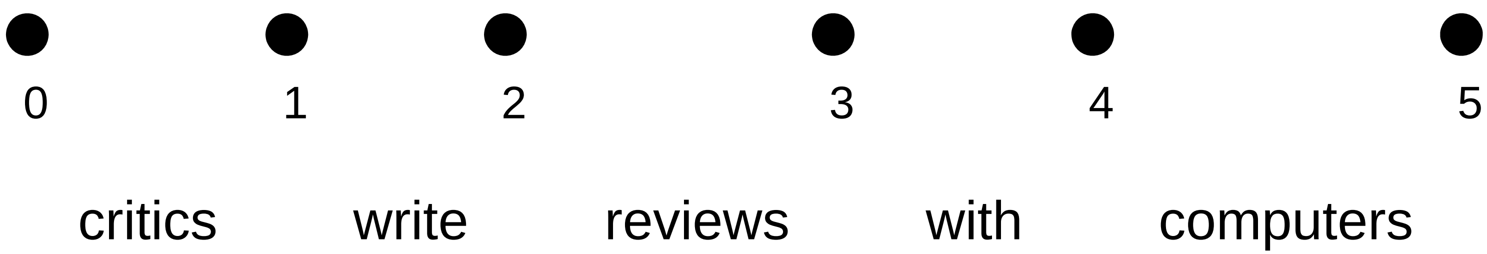
Word Items

- Building an item for the first time is called discovery. Items go into the agenda on discovery.
- To initialize, we discover all word items (with score 1.0).

AGENDA

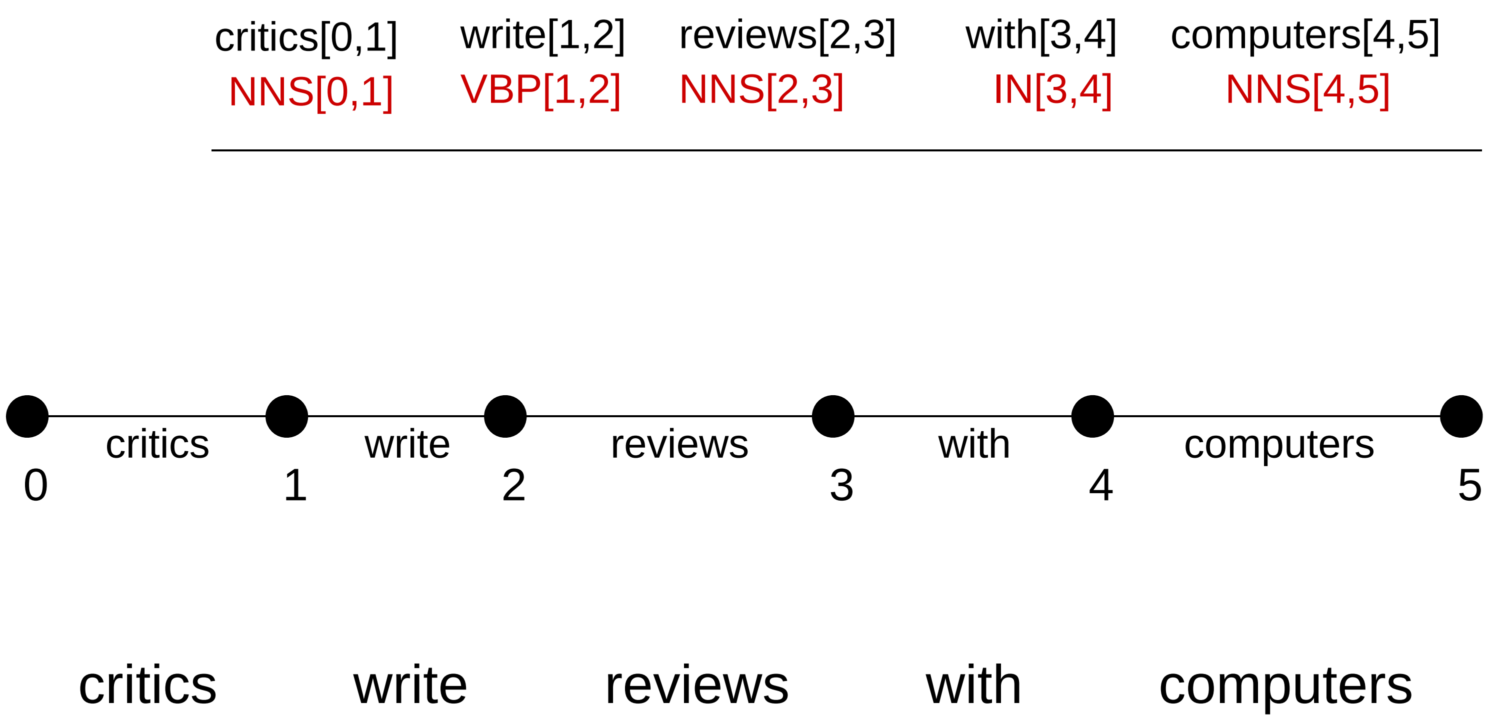
critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

CHART [EMPTY]



Unary Projection

- When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda



Item Successors

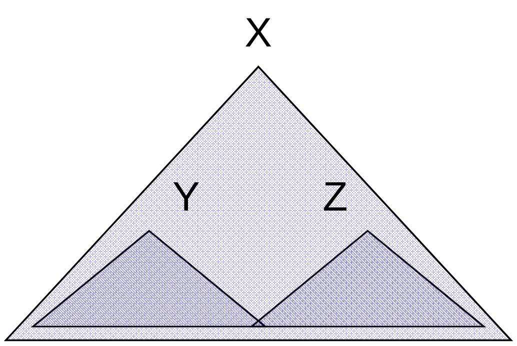
- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

$$Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j]$$

  - Hypergraph successors: combine with items already in our chart

$$Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y\ Z \text{ form } X[i,k]$$

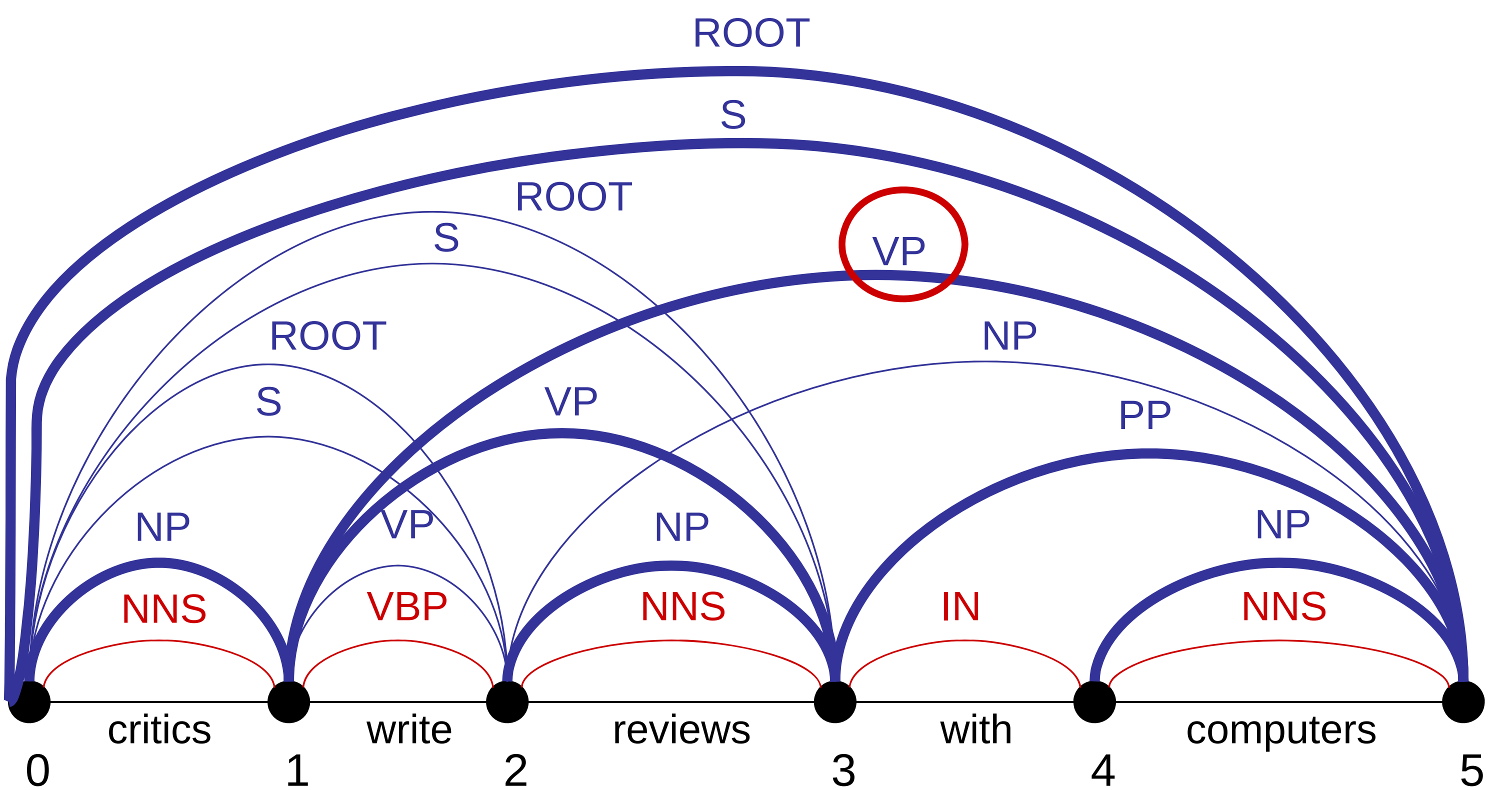
  - Enqueue / promote resulting items (if not in chart already)
  - Record backtraces as appropriate
  - Stick the popped edge in the chart (closed set)
- Queries a chart must support:
  - Is edge X:[i,j] in the chart? (What score?)
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?



An Example

NNS[0,1] VBP[1,2] NNS[2,3] IN[3,4] NNS[3,4] NP[0,1] VP[1,2] NP[2,3] NP[4,5] S[0,2]

VP[1,3] PP[3,5] ROOT[0,2] S[0,3] VP[1,5] NP[2,5] ROOT[0,3] S[0,5] ROOT[0,5]



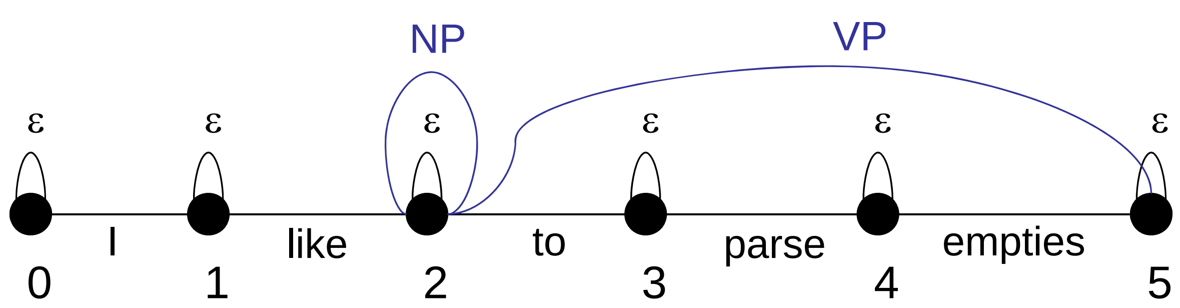
Empty Elements

- Sometimes we want to posit nodes in a parse tree that don't contain any pronounced words:

I want you to parse this sentence

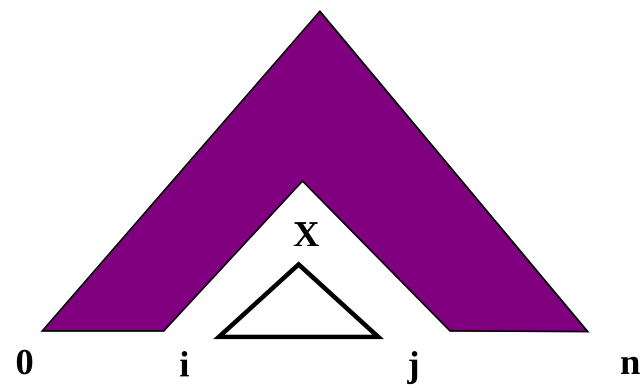
I want [   ] to parse this sentence

- These are easy to add to a chart parser!
  - For each position i, add the "word" edge ε:[i,i]
  - Add rules like NP → ε to the grammar
  - That's it!



UCS / A*

- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A* search also well defined
- You can also speed up the search without sacrificing optimality
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]



(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don't know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.