### A Recursive Parser

```python
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:

```python
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]
```

### Memory: Theory

- 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.
  - What about sparsity?

### Time: Theory

- How much time will it take to parse?
  - Have to fill each cache element (at worst)
  - Each time the cache fails, we have to:
    - Iterate over each rule X -> Y Z and split point k
    - Do constant work for the recursive calls
  - Total time: |rules| * n^3
  - Cubic time
  - Something like 5 sec for an unoptimized parse of a 20 word sentences
Unary Rules

- **Unary rules?**

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

Same-Span Reachability

```
NX  SD  X  RRC  LST  CONJ  P  NAC
ADJP  ADVP  FRAG  INTJ  NP  PP  PRN  QP  S  SBAR  UCP  VP  WHNP  WHADJP  SBARQ  WHPP  WHADVP
```

Same-Span Reachability

```
ADJP  ADVP  FRAG  INTJ  NP  PP  PRN  QP  S  SBAR  UCP  VP  WHNP  WHADJP  SBARQ  WHPP  WHADVP
```

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

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

Alternating Layers

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

A Bottom-Up Parser (CKY)

- **Can also organize things bottom-up**

```
betweenScore(a)  
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])  
                for (j = i + diff)  
                    for (X -> YZ : rule)  
                        for (k : [i+1, j-1])  
                            score[X][i][j] = max(score[X][i][j],  
                                score[X][i][j] *  
                                score[Y][i][k] *  
                                score[Z][k][j])
```

Efficient CKY

- **Lots of tricks to make CKY efficient**
  - Most 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 critical:
    - Many X[i,j] can be suppressed on the basis of the input
      string
    - We’ll see this next class as figures-of-merit or A* heuristics
Memory: Practice

- Memory:
  - Still requires memory to hold the score table

- Pruning:
  - score[X][i][j] can get too large (when?)
  - can instead keep beams scores[i][j] which only record scores for the top K symbols found to date for the span [i,j]

Time: Theory

- How much time will it take to parse?
  - For each diff (<= n)
    - For each i (<= n)
      - For each rule X → Y Z
        - Do constant work
  - Total time: |rules|*n^3

Runtime: Practice

- Parsing with the vanilla treebank grammar:
  - Why’s it worse in practice?
    - Longer sentences “unlock” more of the grammar
    - All kinds of systems issues don’t scale

Rule State Reachability

- Example: NP CC •
  - 1 Alignment
  - Observed exponent: 3.6

- Example: NP CC NP •
  - n Alignments
  - Many states are more likely to match larger spans!

(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.

A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
  - Nodes: positions between words
  - Edges: spans of words with labels, represent the set of trees over those words rooted at x
  - A chart: records which edges we’ve built
  - An agenda: a holding pen for edges (a queue)
- We’re going to figure out:
  - What edges can we build?
  - All the ways we built them.
Word Edges
- An edge found for the first time is called discovered. Edges go into the agenda on discovery.
- To initialize, we discover all word edges.

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

CHART [EMPTY]

The “Fundamental Rule”
- When we pop edges off of the agenda:
  - Check for unary projections (NNS → critics, NP → NNS)
  - Y[i,j] with X → Y forms X[i,j]
  - Combine with edges already in our chart (this is sometimes called the fundamental rule)
  - Y[i,j] and Z[j,k] with X → Y Z form X[i,k]
  - Enqueue resulting edges (if newly discovered)
  - Record backtraces (called traversals)
  - Stick the popped edge in the chart

Queries a chart must support:
- Is edge X[i,j] in the chart?
- What edges with label Y end at position j?
- What edges with label Z start at position i?

Unary Projection
- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it

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

CHART [EMPTY]

Exploiting Substructure
- Each edge records all the ways it was built (locally)
  - Can recursively extract trees
  - A chart may represent too many parses to enumerate (how many?)

Order Independence
- A nice property:
  - It doesn’t matter what policy we use to order the agenda (FIFO, LIFO, random).

  Why? Invariant: before popping an edge:
  - Any edge X[i,j] that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
  - Convince yourselves this invariant holds!

  This will not be true once we get weighted parsers.
Empty Elements

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

```python
I want John to parse this sentence
```

- These are easy to add to our chart parser:
  - For each position $i$, add the “word” edge $\varepsilon$-$[i,i]$
  - Add rules like $\text{NP} \to \varepsilon$ to the grammar
  - That’s it!

---

I like to parse empties

```
\varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \\
\text{NP} \quad \text{VP}
```

I want John to parse this sentence

```
\varepsilon \varepsilon \varepsilon \varepsilon \varepsilon \\
\text{NP} \quad \text{VP}
```

I want [ ] to parse this sentence