Handout 3: The CKY Algorithm

1. `g0.g` and `g0.lex`:

S → NP VP
NP → Det N
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP

a Det
book N V
flight N
I NP
in P
May NP

2. Sentence "I book a flight in May"

a. Number the positions between words (and at start and end)

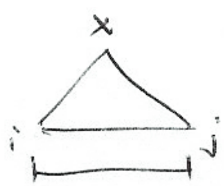
$$_0\ \mathrm{I}\ _1\ \mathrm{book}\ _2\ \mathrm{a}\ _3\ \mathrm{flight}\ _4\ \mathrm{in}\ _5\ \mathrm{May}\ _6$$

b. "a" can be a Det: $_2\mathrm{Det}_3$

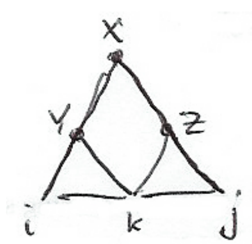
c. "flight" can be an N: $_3\mathrm{N}_4$

d. $_2\mathrm{Det}_3$ and $_3\mathrm{N}_4$ can be combined to make an NP: $_2\mathrm{NP}_4$

3. Phrases (nodes): labeled spans $_iX_j$



4. Combining two phrases: $_iX_j$ is created by combining $_iY_k$ and $_kZ_j$, provided there is a rule $X \to Y Z$



5. Top-down parsing

a. Can we make $_0\mathrm{S}_6$?

b. Possible split points: 1, 2, 3, 4, 5

c. Possible rules (only one, here): S → NP VP

   d. Consider each split point + rule. E.g., split point 2, rule S $\rightarrow$ NP VP.
   e. Recurse: Can we make $_0\mathrm{NP}_2$ and $_2\mathrm{VP}_6$?

6. We will end up asking (e.g.) "Can we make $_3\mathrm{N}_4$" many times
   a. Caching: the first time we ask, we compute the answer, then record it in the table. After that, we can just look it up in the table.
   b. Dynamic programming: anticipate what the questions are going to be, make sure the answer is computed <u>before</u> the first time the question is asked.

7. CKY is dynamic programming
   a. We asked for **children** $_0\mathrm{NP}_2$ and $_2\mathrm{VP}_6$ in order to build **parent** $_0\mathrm{S}_6$. Children are always smaller (number of words covered).
   b. Span **width** is number of words covered: $j - i$
   c. Visit cells in order of increasing width. Guarantees that all children have already been computed when needed.
   d. Initialize by filling in lowest cells

```
for i in range(0,5):
    for pos in lex.parts(word[i]):
        cell[i, i+1].add(pos)
```

   e. Main loop:

```
for width in range(2, 6):
    for i in range(0, 6-width+1):
        fill_cell(i, i+width)
```

   f. where:

```
def fill_cell (i,j):
    for k in range(i+1, j):
        for Y in cell[i,k]:
            for (X -> Y Z) in g.continuations(Y):
                if Z in cell[k,j]:
                    cell[i,j].add(X)
```

   g. Success if `S` is in `cell[0,6]`

8. Alternative loop (book version)
   a. Visit columns left to right (increasing value of j), decreasing value of i (increasing width) within a column
   b. Since first child always has smaller j than parent, it is visited before parent
   c. Since second child always has smaller width than parent, it is visited before parent

d. Initialization + main loop:

```
for j in range(1,6):
    for pos in lex.parts(word[j-1]):
        cell[j-1, j].add(pos)
    for width in range(2, j+1):
        fill_cell(j-width, j)
```

9. Parsing (rather than just recognition)

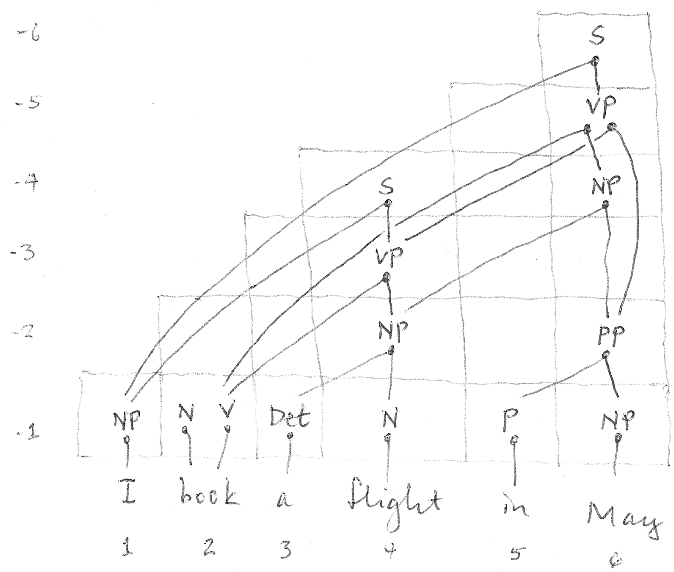
a. Keep track of the expansions $[{}_iY_k, {}_kZ_j]$ that were used to form ${}_iX_j$

b. If there is more than one, keep track of them all

c. After the chart is filled, **unwind**: pull out trees in all possible ways

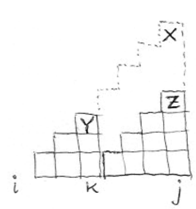
d. Eventually – use semantics and/or probabilities to disambiguate

10. Graphically



a. Rows have same width, columns have same j

b. To find i for a cell: down diagonally to the left



11. Grammar must be in Chomsky Normal Form (CNF)

- **a.** Lexical rules: Pos $\rightarrow$ word
- **b.** Binary rules: X $\rightarrow$ Y Z
- **c.** Nothing else allowed