CYK algorithm

(Cocke, Younger, Kasami)

Method for parsing (~~see~~ seeing how a string can be derived from a grammar) in cubic time (i.e. approx. $k|w|^3$ time, $|w|$ = length of string), k = size of grammar)

This is an example of bottom-up parsing; we try to put together pieces that could be the bottom of deriv tree and try to match them together to make the whole deriv tree.

(Brute force was top-down; we start w/ start symbol and try to build the deriv tree from there.)

bbabb

E.g. Grammar: (must be in CNF)

$S \to AB$

$A \to BB \mid a$

$B \to AB \mid b$

string: bbabb.

Idea: Let $V_{ij} = \{$ all variables from which I can get the string starting at letter i and ending at letter j $\}$

We calculate

$V_{00}, V_{11}, V_{22}, \ldots, V_{44}$ (in gen, $V_{\ell-1,\ell-1}$)

Then

$V_{01}, V_{12}, \ldots, V_{34}$

Then

$V_{02}, \ldots, V_{24}$,

until we get to

V_{04}

If S is one of the vars in V_{04}, the answer is yes, otherwise no.

(If we want not just yes/no but a derivation tree, then we store along w/each var in V_{ij} a partial deriv tree w/ that var.)

How do we calculate these things?

To calculate V_{ii} - we find all vars that can produce letter i.

To calculate V_{ij} $(i \neq j)$ -

- look at each k, $i \leq k \cancel{\leq} j$ $k+1, j$
  - look at every combo of a var in V_{ik} + one in V_{kj} and ~~check if~~ find all the vars that can produce that pair.

V_{ij} is the set of all such vars.

Our example:

$$\begin{array}{ccccc} V_{00} & V_{11} & V_{22} & V_{33} & V_{44} \\ \| & \| & \| & \| & \| \\ B & B & A & B & B \end{array}$$

$V_{01} = \{ (\text{var from } V_{00}, \text{ var from } V_{11}) \}$

$= \{A\}$ (or )

$V_{12} = \emptyset$, $V_{23} = \{S, B\}$ $V_{34} = \{A\}$

$V_{02} =$ I can glue together

$\cancel{V_{00}, V_{12}}$ or V_{01}, V_{22}

$\cancel{\{B\}, \emptyset}$ $\qquad$ $\{A\}$ $\{A\}$

$\emptyset$ $\qquad$ $\emptyset$

$= \emptyset$,

$$V_{13} = \left[\begin{array}{ccc} V_{11} \; V_{23} & \text{or} & V_{12} \; V_{33} \\ \{B\} \; \{S,B\} & & \emptyset \; \{B\} \\ A & & \emptyset \end{array} \right] = \{A\}$$

$$
V_{24} = \left[\begin{array}{ccc}
V_{22}\,V_{34} & \text{or} & V_{23}\,V_{44} \\
\{A\}\,\{A\} & & \{S,B\}\,\{B\} \\
\emptyset & & \{A\}
\end{array}\right] = \{A\}
$$

$$
V_{03} = \left(\begin{array}{ccccc}
V_{00}\,V_{13} & \text{or} & V_{01}\,V_{23} & \text{or} & V_{02}\,V_{33} \\
\{B\}\{A\} & & \{A\}\{S,B\} & & \emptyset\ \{B\} \\
\emptyset & & \{S,B\} & & \emptyset
\end{array}\right) = \{S,B\}
$$

$$
V_{14} = \left(\begin{array}{ccccc}
V_{11}\,V_{24} & \text{or} & V_{12}\,V_{34} & \text{or} & V_{13}\,V_{44} \\
\{B\}\ \{A\} & & \emptyset\ \{A\} & & \{A\}\ \{B\} \\
\emptyset & & \emptyset & & \{\cancel{A}S,B\}
\end{array}\right) = \{S,B\}
$$

$$
\begin{array}{lccccccc}
V_{04} = & V_{00}\,V_{14} & \text{or} & V_{01}\,V_{24} & \text{or} & V_{02}\,V_{34} & \text{or} & V_{03}\,V_{44} \\
& \{B\}\{S,B\} & & \{A\}\{A\} & & \emptyset & & \{S,B\}\{B\} \\
& A & & \emptyset & & \emptyset & & \cancel{\emptyset}\,A
\end{array}
$$

$$
\| \\
\{A\}
$$

$S \notin V_{04} = \{A\}$, so $\boxed{\text{NO}}$

Extra credit (on the scale of 1 HW assignment)

Write a computer program (or procedure) with comments implementing this algorithm, send me code and ~~a~~ some tests w/output.

Anytime before last day of class

Any programming language not designed to be obfuscating. ~~(Note added after cl~~

(Note added ~~after class~~: There is a programming language I have in mind as illegal - it's name contains a 4 letter word.)

Pushdown automata:

This is like a (nondeterministic) finite automaton, but w/ a stack - data storage w/o maximum capacity, but you can only see the top thing at any given point (you can remove the top thing).

- We start at a start state (with a special symbol on the stack)
- Depending on what's ~~the~~ state we're in, what character is at beginning of the string, and what is at top of stack, we

  a) ~~move to a have~~ move to another state

  b) ~~Put stuff on the~~ take the top symbol off the stack and put stuff on the stack

  convention

  c) either take off the first char of string or not (λ-transition)

- If there is a way to end at a final state having eaten the whole string we accept.

  convention - we don't care what is left on the stack.