CYK algorithm

(Cook, Younger, Kasami)

Method for parsing (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.)
E.g. Grammar: (must be in CNF)
\[ \begin{align*}
S & \rightarrow AB \\
A & \rightarrow BB|a \\
B & \rightarrow AB|b
\end{align*} \]

String: bbabbb.

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

We calculate \( V_{00}, V_{11}, V_{22}, \ldots, V_{44} \) (in gen, \( V_{l-1,l-1} \)).

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 $k$, $i \leq k \leq j$
- look at every combo of a var in $V_{ik}$ + one in $V_{kj}$ and find all the vars that can produce that pair.
$V_{ij}$ is the set of all such vars.
Our example:

\[ V_{00} \hspace{1em} V_{11} \hspace{1em} V_{22} \hspace{1em} V_{33} \hspace{1em} V_{44} \]

\[ \begin{array}{cccc}
B & B & A & B \\
\end{array} \]

\[ V_{01} = \left( \text{var from } V_{00}, \text{ var from } V_{11} \right)^3 \]

\[ = \{ A^3 \} \quad \text{or} \quad \begin{array}{c}
A \\
\hline
B & B \\
\hline
b & b \\
\end{array} \]

\[ V_{12} = \emptyset, \quad V_{23} = \{ B^3 \} \quad V_{34} = \{ A^3 \} \]

\[ V_{02} = \emptyset, \] I can glue together

\[ V_{00}, V_{12} \] or \[ V_{01}, V_{22} \]

\[ \{ B^3 \}, \emptyset \quad \{ A^3 \}, \{ A^3 \} \]

\[ \emptyset, \emptyset \]

\[ V_{13} = \begin{bmatrix} V_{11} & V_{23} \\
B^3 & B^3 \\
A \\
\end{bmatrix} \quad \text{or} \quad \begin{bmatrix} V_{12} & V_{33} \\
\emptyset & \{ B^3 \} \\
\emptyset \\
\end{bmatrix} = \{ A^3 \} \]
\[ V_{24} = \begin{bmatrix} V_{22} & V_{34} \\ \{A^3\} & \{A^3\} \end{bmatrix} \text{ or } \begin{bmatrix} V_{23} & V_{44} \\ \{S,B^3\} & \{B^3\} \end{bmatrix} = \{A^3\} \]

\[ V_{03} = \begin{bmatrix} V_{00} & V_{13} \\ \{B^3,A^3\} & \{A^3,S,B^3\} \end{bmatrix} \text{ or } \begin{bmatrix} V_{01} & V_{23} \\ \{A^3,S,B^3\} & \{B^3\} \end{bmatrix} = \{S,B^3\} \]

\[ V_{14} = \begin{bmatrix} V_{11} & V_{24} \\ \{B^3\} & \{A^3\} \end{bmatrix} \text{ or } \begin{bmatrix} V_{12} & V_{34} \\ \{A^3\} & \{B^3\} \end{bmatrix} \text{ or } \begin{bmatrix} V_{13} & V_{44} \\ \{S,B^3\} & \{B^3\} \end{bmatrix} = \{S,B^3\} \]

\[ V_{04} = V_{00} V_{14} \text{ or } V_{01} V_{24} \text{ or } V_{02} V_{34} \text{ or } V_{03} V_{44} \]

\[ \{B^3,S,B^3\} \text{ or } \{A^3\} \text{ or } \{B^3\} \text{ or } \{S,B^3\} \text{ or } \{B^3\} \]

\[ S \not\in V_{04} = \{A^3\}, \text{ so } \boxed{\text{NO}} \]
Extra credit (on the scale of 1 HW assignment) (or procedure)

Write a computer program implementing this algorithm with comments send me code and some tests w/output.

Anytime before last day of class
Any programming language not designed to be obfuscating. *(Note added after 01/20)*

*Note added: There is a programming language I have in mind as "illegal" - it's name contains a 4 letter word.*

Pushdown automata:

(nondeterministic)

This is like a 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 another state
  b) Put stuff on the stack and take the top symbol off the stack
  c) Either take off the first character of string or not (\(\lambda\)-transition)

- If there is a way to end at a final state, we accept.
  convention: we don't care what is left on the stack.