John Pennig

Homework 5. CSCI 330
Due date: Monday February 17th
Building a Recursive Descent parser.2

Consider the following grammar:
I ➔ iES | iESeS E➔EoG | G G➔x|y|z|w S➔ s | dLb L➔s | sL

Q 1. Construct derivations for the strings ixoyowdssbes and ixoys

I ➔ iESeS ➔iEoGSeS ➔ iEoGoGSeS➔ iGoGoGSeS ➔ ixoGoGSeS ➔ ixoyoGSeS ➔ ixoyowGSes ➔ixoyowSeS➔ ixoyowdLbeS➔ ixoyowdsLbeS ➔ ixoyowdssbeS ➔ ixoyowdssbes

I ➔ iES ➔ iEoGS ➔ iGoGS ➔ ixoGS ➔ ixoyS ➔ ixoys

Q 2. Identify the grammar rules that have Left Recursion. Rewrite those rules to eliminate the Left Recursion.

E➔EoG | G has Left Recursion because of the E symbol appearing as the first symbol on the right side of the production rule. We can use a new symbol of E' to rewrite this production rule to no longer have left recursion.

The new grammar would result in

E ➔ GE' and

E' ➔ oGE' | ε

where ε is an empty string.

Q 3. Consider the process of building a top-down parser for the rewritten grammar. We will have a function for each non-terminal. The input to each of these functions is a string that has to be derived. Describe your strategy (logic to be employed) in each of these functions, looking only at the first symbol in the input string. In other words, what action would the function take for each possible terminal symbol in the language (i, e, o, x, y, z, w, s, d and b). Do we have a unique action for each symbol, or do we find symbols for which there is more than one possible action?

In each of these functions it looks at the next input symbol to be derived. Such as the start symbol has to produce i first. Next it continues into the E function since I➔ iES. It continues parsing until a character has been consumed and then continues down the stack until the next character is parsed. The symbols that have more than one possible action are s which can be produced from S or L, L can produce sL or just s, and I can produce iES or iESeS.

Q 4. Consider the functions from Q 3 where we had more than one possible action. You will find rules where a non-terminal generates two or more strings which share a common prefix. (These would fail the pairwise disjointness test.) Rewrite these rules using Left Factoring to remove these common prefixes.

There are two functions where we can apply Left Factoring. I ➔ iES | iESeS and L ➔ s | sL. These can be rewritten to I ➔ iESI' where I' ➔ es | ε and L ➔ sL' where L' ➔ L | ε where ε is an empty string.

Q 5. Rewrite the affected functions from Q 3 to verify that we have a unique course of action for every terminal symbol.

Now, the L grammar no longer has to look ahead to know what to produce, it always produces sL' where the s is consumed before it is passed to the L' function which has to only check if the string continues, if it does it passes it back to the L function where it can consume another s or continue up the recursion stack.

Q 6. Implement the parser using LISP. The code should be in your GitHub repository, in the folder HWork5.

Q 7. To test the code generate seven strings of different lengths between 5 and 20 that can be generated by this grammar. Then perturb these strings to get seven strings of different lengths that cannot be generated by this grammar. Start a script session in the HWork5 folder, run sbcl and load the file. Test the program on all your 14 test strings.

ixoys
ixoyowdssbes
izdsb
iwdsssb
iyozses
iyoyoyoyoydssbes
izses

Q 8 (Extra Credit: generating error messages) Modify your program from Q 6 to generate specific error messages. This will require adding additional parameters, and adding addition conditions in the functions. A simple error message is "Unexpected symbol at position <x>". More specific error messages will require additional logic.
What to submit: Code and script file in GitHub. Answers to Q1 through Q 5 in D2L. **For extra credit,** in D2L, add an answer to Q 8 explaining how you generated error messages, and what changes were made to LISP code. Create a second script file to show the error messages.