CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2013
Assignment #5
30 points + 5 extra credit
Due: October 1, 2013

Part 1: Reading and Exercises

Compilers - Principles, Techniques, & Tools, 2nd Edition, Sections 4.1, 4.2, 4.3
Homework Exercises – laboratory and answer the questions at the end of the assignment.

Introduction

In this assignment, you implement the language of the string (a|b)*(a|b)b(a|b) using non-deterministic (NFA) and deterministic (DFA) finite automata. You are given a main program (main.c), scanner (scan.l) and two parsers (gramNFA.y gramDFA.y) that already accept (a|b)*. You modify the two yacc files to correctly implement the grammar for the language. You test your compilers using three test programs (test1.c test2.c test3.c) with a script file (hw05_test.sh). You also implement the error production and keep track of the number of syntax errors.

Part 2: Laboratory

From a console window, make a directory on your computer and call it hw05.
mkdir EECS337/caseid/hw05/ ; where caseid is YOUR Case ID, enter all in lower case

Download a copy of: hw05_caseid.tar file to the hw05 directory from http://blackboard.case.edu/ in the EECS337 homework assignment area. To untar the file, type the command.
tar xvf hw05_caseid.tar

The following files will be created in the current working directory.
Makefile
gramDFA.y
gramNFA.y
hw05_test.sh
main.c
scan.l
test1.c
test2.c
test3.c
yystype.h

Look at the test files (test1.c test2.c test3.c) and examine the input strings. The file test1.c contains one string that is accepted by the language of the string.
aba

The file test2.c contains strings that are accepted by the language of the string and the last few examples have number characters embedded in the strings, which create syntax errors.
aba
abb
bba
bbb
aaba
…
b1b2a
b1b2b

The file test3.c contains four strings that are not accepted by the language of the string:
aaa
aab
baa
bab

Look at the Makefile. Notice the line with all, creates two executable versions (hw05estNFA, hw05testDFA) for testing. These two versions share the same yystype.h, main.c and scan.l files and use two different yacc files (gramNFA.y and gramDFA.y). **So be sure to edit the gramNFA.y and gramDFA.y files for testing each implementation.** The Makefile also copies the y.output file to either yNFA.output or yDFA.output. To build one version at a time then use one of the commands below.

make hw05testNFA
make hw05testDFA

Look at the hw05_test.sh Unix script file. It does a make clean and builds of both versions and then runs the same tests using the two executable versions.

```
#
#      hw05_test.sh
#
make clean
make
#
#      test the NFA grammar
#
./hw05testNFA +yydebug test1.c
./hw05testNFA test2.c
./hw05testNFA test3.c
#
#      test the DFA grammar
#
./hw05testDFA +yydebug test1.c
./hw05testDFA test2.c
./hw05testDFA test3.c
```

Test the hw05testNFA and hw05testDFA programs by using each of the following commands:
./hw05testNFA test1.c
./hw05testNFA test2.c
./hw05testNFA test3.c
./hw05testDFA test1.c
./hw05testDFA test2.c
./hw05testDFA test3.c

Out of the box, the output from the test files should be successful for test1.c and test3.c and a syntax error on test2.c. When you correctly implement the assignment you should have test1.c and test2.c run successfully and test3.c should give four syntax errors. Out of the box, your output should be:

```
./hw05testNFA test1.c
for caseid start time: Thu Sep 19 07:18:20 2013
NFA A0: accept
./hw05testNFA test2.c
for caseid start time: Thu Sep 19 07:18:20 2013
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept

^
syntax error
Error: yyparse 1
./hw05testNFA test3.c
for caseid start time: Thu Sep 19 07:18:20 2013
NFA A0: accept
NFA A0: accept
NFA A0: accept
NFA A0: accept
```

You are now ready to solve the laboratory assignment.

Part 3: Laboratory Assignment
Edit the main.c file and change all strings with caseid to your Case ID. Save the file.

Edit the scan.l file and add two regular expressions to pass the characters “a” and “b”. Change the dot (.) expression to return nothing. Save, build and test the files again. The syntax error from test2.c should now be eliminated because it returns only a and b to the parser and skip everything else.

Design the NFA state transition table for the string (a|b)*(a|b)b(a|b). Change the state table into an NFA grammar. Now edit the gramNFA.y file. Change the productions to implement the grammar for using non-deterministic (NFA) finite automata. The accepting state is always empty.

Build the program using the following commands:
make clean
make hw05testNFA

Test the hw05testNFA program by using each of the following commands:
./hw05testNFA test1.c *// success*
./hw05testNFA test2.c *// success*
./hw05testNFA test3.c *// syntax error*

Design the DFA state transition table for the string (a|b)*(a|b)b(a|b). Change the state table into a DFA grammar. Now edit the gramDFA.y file. Change the productions to implement the grammar using deterministic (DFA) finite automata. Remember there is only one path for each symbol out of each state in DFA and include the EOL or EOF as an input symbol.

Build the program using the following commands:
make clean
make hw05testDFA

Test the hw05testDFA program by using each of the following commands:
./hw05testDFA test1.c *// success*
./hw05testDFA test2.c *// success*
./hw05testDFA test3.c *// syntax error*

Part 4: Syntax Error Handling
Edit the yystype.h file and add the variable below to the DATA structure and the extern statement to the scan.l external declarations . Save the file.

```
        int     errors;

        extern void yysync( void);
```

Edit the main.c file and inside the main_exit routine replace the `return 0;` with the code below. Save the file.

```
/*
 *      check if compiler errors
 */
        if( data.errors)
                fprintf( stderr, "Error: compiler errors: %d\n", data.errors);
        return( data.errors);
```

Edit the scan.l file and add the code below at the end of the file after the %%. Save the file.

```
/*
 *      resync the parser with a syntax error
 */
void yysync( void)
{
        int c;
        data.errors++;
        while (( c = getc( yyin )) != EOF)
        {
                if( c == '\n')
                {
                    unput( c);
                    break;
                }
        }
}
```

Edit the gramNFA.y and gramDFA.y files and add to the "line" production the error production shown below. The word "error" is a non-terminal symbol for a parser error. The function yysync is in the scan.l file to read characters to EOF or to a new line character. If it sees a new line it "unput" that character and returns. The macro "yyerrok" tells the parser to continue parsing the file. This allows the parser to resynchronize on a per line bases. Save the files.

```
                | error '\n' { yysync(); yyerrok; }
```

Build the program using the following commands:
make clean
make

When the error handling code is complete and working then execute the command below. This should now continue parsing the test3.c file and generate four errors that are reported from the main_exit routine, shown below.

./hw05testNFA test3.c

```
for caseid start time: Thu Sep 19 07:48:34 2013

^
syntax error

^
syntax error
```

```
^
syntax error

^
syntax error
Error: compiler errors: 4
Error: exit: 4
```

Part 4 A: Laboratory Extra Credit (5 points)
Notice the syntax error messages do not include any previous string information. Edit the files (yystype.h, gram[ND]FA.y, scan.l) to report a meaningful syntax error message. For example your output should look like:

./hw05testNFA test3.c

```
for caseid start time: Mon Sep 23 07:47:44 2013

aaa
  ^
syntax error

aab
  ^
syntax error

baa
  ^
syntax error

bab
  ^
syntax error
Error: compiler errors: 4
Error: exit: 4
```

When done, upload **ONLY** your scan.l file to blackboard in the assignment area. Submit it with the comments **hw05** with your **CaseID** on the title line. The extra credit will be graded on-line and your score will be added to your assignment score. When you generate your laboratory output file in Part 5, show the improved syntax error handling. Mark your output file with **EXTRA CREDIT** and turn in with your assignment.

Part 5: Laboratory Output Generation
When all your lab assignments have been completed execute the Unix script file "./hw05_test.sh". To redirect the standard error (stderr) and standard output (stdout) to a file use the following command:
./hw05_test.sh &> hw05_test.txt

Print out the hw05_test.txt output file. Put your name, assignment number and date on it. Answer the questions at the end of the assignment.

Your final directory should look like:

```
EECS337/caseid/hw05/Makefile
EECS337/caseid/hw05/gramDFA.y
EECS337/caseid/hw05/gramNFA.y
EECS337/caseid/hw05/hw05_caseid.tar
EECS337/caseid/hw05/hw05_test.sh
EECS337/caseid/hw05/hw05_test.txt
EECS337/caseid/hw05/hw05testDFA
EECS337/caseid/hw05/hw05testNFA
EECS337/caseid/hw05/main.c
EECS337/caseid/hw05/scan.l
EECS337/caseid/hw05/test1.c
EECS337/caseid/hw05/test2.c
EECS337/caseid/hw05/test3.c
EECS337/caseid/hw05/yDFA.output
EECS337/caseid/hw05/yNFA.output
EECS337/caseid/hw05/yystype.h
```

Part 6: Laboratory Questions

1) What happens if you remove the [\n] lex expression and action {…} from scan.l?

2) Why does test2.c generate a syntax error when you build it out of the box?

3) Why does test3.c NOT generate a syntax error when you build it out of the box?

4) After implementation, explain why one of your files (yNFA.output yDFA.output) is larger?

5) How many bytes are in your executable programs (hw05testNFA hw05testDFA)?

6) What is a reason to write your grammar in DFA over NFA inside a yacc parser?

7) In yacc, how do you define the starting state?

8) In yacc, what does an accepting state have that is not found in a transition state?

9) Write your yacc NFA productions for solving language [(a|b)*(a|b)b(a|b)].

10) Write your yacc DFA productions for solving language [(a|b)*(a|b)b(a|b)].