COMS W4115
Programming Languages and Translators
Homework Assignment 2

Prof. Ronghui Gu
Columbia University

Due March 9th, 2020
at 11:59PM PM

Submit your assignment as a zipped directory for question 1, a `.mll` file for question 2, and a single PDF file with the rest of the homework problems written legibly, on Courseworks.

Do this assignment alone. You may consult the instructor or a TA, but not other students. You may not consult solutions from earlier semesters.

1. **20pt** Extend the three-slide "calculator" example shown in the OCaml slides (the source is also available on the class website) to accept variables named with identifiers consisting of lowercase letters, assignment to those variables, and sequencing using the ";" operator. For example,

   ```
   foo = 3; bar = baz = 6; foo * bar + baz
   ```

   should print "24"

   Use a string-to-integer Map to track variable variables. Add tokens to the parser and scanner for representing assignment, sequencing, and variable names.

   The ocamllex rule for the variable names, which converts the letters a–z into the corresponding literals, is

   ```
   | ['a'-'z']+ as id { VARIABLE(id) }
   ```

   The new ast.mli file is

   ```
   type operator = Add | Sub | Mul | Div
   type expr =
       Binop of expr * operator * expr
     | Lit of int
     | Seq of expr * expr
     | Asn of string * expr
     | Var of string
   ```

   Make sure your code compiles without warnings

2. **10pt** Write a regular expression for lexing floating point constants in a file called `scanner.mll`. Your tokenizing rule should be called `lex_float`. Your code should build without warnings when `ocamlbuild scanner.native` is run from the terminal. Your executable should read from `stdin` and print the successfully lexed pattern to `stdout`. In order to do so, include the following code at the end of `scanner.mll`.

   ```
   {
     let buf = Lexing.from_channel stdin in
     let f = lex_float buf in
     print_endline f
   }
   ```

   You should lex floats according to the following (after K&R):

   > A floating constant consists of an integer part, a decimal point, a fraction part, an `e` or an `E`, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the `e`/`E` and the exponent (not both) may be missing.

   Hint: make sure your regular expression accepts floating constants such as `1. 0.5e-15 .3E+3 .2 1e5 3.5E-4` but not integer constants such as `42`

3. **10pt** Draw a DFA for a scanner that recognizes and distinguishes the following set of keywords. Draw accepting states with double lines and label them with the name of the keyword they accept. Follow the definition of a DFA given in class.

   ```
   fun function a let asp open full fell operator
   ```

4. **20pt** Construct nondeterministic finite automata for the following regular expressions using Thompson's algorithm (Algorithm 3.23, p. 159, also shown in class), then use the subset construction algorithm to construct DFAs for them using Algorithm 3.20 (p. 153, also shown in class).

   (a) $a\,a^*\,b$
   (b) $(ab \mid b)^* a$
   (c) $((a \mid \epsilon)\,ba)^*$

   Number the NFA states; use the numbers to label DFA states while performing subset construction, e.g., like Figure 3.35 (p. 155).

5. **20pt** Using this grammar, whose three terminals are *a*, *b*, and *c*,

$$E \to a\ F\ b$$
$$E \to c$$
$$F \to E\ a\ F$$
$$F \to E$$

   (a) Construct a rightmost derivation for *aacacbb* and show the handle of each right-sentential form.

   (b) Show the steps of a shift-reduce (bottom-up) parser corresponding to this rightmost derivation.

   (c) Show the concrete parse tree that would be constructed during this shift-reduce parse.

6. **20pt** Build the LR(0) automaton for the following ambiguous grammar. **if**, **else**, and **null** are terminals; the third rule indicates *T* may be the empty string. Indicate the state in which the shift/reduce conflict appears.

$$S' \to S$$
$$S \to \textbf{if}\ S\ T$$
$$S \to \textbf{null}$$
$$T \to$$
$$T \to \textbf{else}\ S$$

Check your work by running "ocamlyacc -v" on the grammar below and looking through the "output" file. Include annotated snippets from the .output file that confirm your answer.


```
%token IF ELSE NULL
%start s
%type <int>s

%%

s : IF s t      { 0 }
  | NULL        { 0 }

t : /* empty */ { 0 }
  | ELSE s      { 0 }
```

Acknowledgment

The assignment is based on the materials designed by Prof. Stephen A. Edwards.