PaDE20-MadsL-Ass2

malj Ljungberg

September 2020

1 Exercise 1.

Review for the last class: Exercise 2.4 and Exercise 2.5 from the PLC book. see machine.java and incomp2.fs

2 Exercise 2.

Regular expressions: Exercise 2.1 from the BCD book.
Answer:

- All number-strings that have the value 42.
  - (42)
- All number-strings that do not have the value 42.
  - ([123567890]*[13456790]*)
- All number-strings that have a value that is strictly greater than 42.
  - ([4-9]+[3-9]*—[5-9]+[0-9]*—[1-9]+[0-9]+[0-9]+[0-9]*)

  //I'm not sure if this captures 50 it should.

3 Exercise 3.

NFA and DFA: Exercise 2.2 from the BCD book.

4 Exercise 4.

NFA and DFA: Exercise 3.2 from the PLC book – (a(ba?)*—(b(b*a?))+)

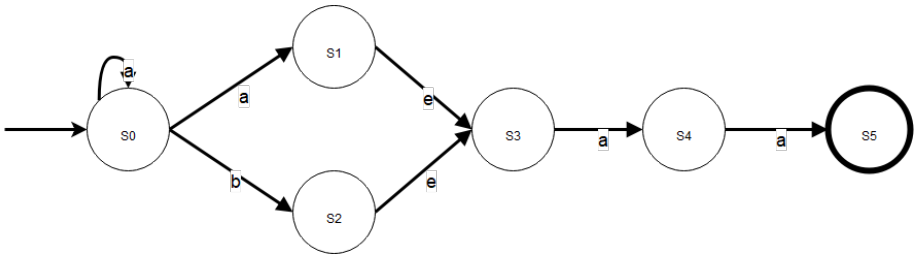

Figure 1: 2-2 NFA

5 Exercise 5.

(A little lexing project) A minimal lexer specification is provided below.

```
{ // starting

module Hello_fslex
open FSharp.Text.Lexing
open System
}

rule Tokenize = parse
  | ['0'-'9']       { LexBuffer<char>.LexemeString lexbuf }
  | _               { failwith "Lexer error: illegal symbol" }


{ // ending

[<EntryPoint>]
let main argv =
      printfn "Hello World from FsLex!\n\nPlease pass a digit:"
      let input = Console.ReadLine()
      let res=Tokenize (LexBuffer<char>.FromString input)
      printfn "The lexer recognizes %s" res
      0

}
```

1. Read the specification. What are the regular expressions involved, and which semantic values are they associated with?

   - Answer: a regular expression that recognises any integer. any number between 0-9 not seperated by commas, periods, any other non integer value.

2. Generate the lexer out of the specification using a command prompt. Which additional file is generated during the process? How many states are there by the automaton of the lexer?

    - Answer: One file is generated a .fs version of the .fsl file used. number of states in the file generated are 3. And my guess to the 3 states are the time where there is a number, any other char non integer - fail state, and no input given ie. empty string.

3. Draw a DFA corresponding to the lexer. Hint: One of the states should indicate errors.

4. Modify the lexer specification file so that it transforms a text with floating-point numbers to another one where all texts are the same except that any floating-point number, deonoted by fp, becomes 0.1*fp.

    Hint: You may consider using [+-]?([0-9]*[.])?[0-9]+ as the regular expressions for floating-point numbers. This will match, for example, 12, 12.45, or .67. You may also use a different regular expression that recognizes not 12 or .67, but it must be able to recognize numbers with a digit in the middle, like 12.45.

    For testing, you can generate the lexer and run it with an input "hello 2.7", the output should be "hello 0.27".

5. Run the generated lexer on a piece C++ code used in the Robot Operating System (ROS): $https://github.com/ros/ros_tutorials/blob/noetic-devel/turtlesim/tutorials/draw_square.cpp$

    Hint: Suppose you can run your lexer with a command "mylex", then you can use a pipeline to get the output: cat draw_square.cpp — mylex.

    - I could not get this to work don't know how to feed the file (draw_square.cpp to the lexer. - (on windows using cli.

Your handin for this exercise should include (1) the lexer specification file; and (2) the transfomed text from running the lexer on draw_square.cpp.