Bonus Problem Set #01 — Given On: Apr 03

CS2200 – Languages, Machines and Computations — Due On: May 04 (end of the day)

1. Aim of this assignment is to build a `grep` like program which can check for strings in a file matching a regular expression. Write programs for each of the following tasks. You can choose any programming language of your choice (C/C++ would be preferred). All your programs must compile and run correctly in department DCF Linux machines.

   (a) (25 points) Takes an alphabet set Σ and regular expression $\alpha \in \Sigma^*$ and outputs an NFA N such that $L(N) = L(\alpha)$.

   (b) (20 points) Takes an NFA N and outputs an equivalent DFA M.

   (c) (25 points) Takes a DFA M and outputs a minimal DFA M' accepting the same language as M.

   (d) (10 points) Takes a DFA M and a string x outputs YES if and only if $x \in L(M)$.

   (e) (20 points) Combine the above procedures to build a program that takes a regular expression α and a text file F as input and outputs the set of all words in F that match α. In this part of the problem, the text file name and regular expression **must** be passed as program arguments using `argc,argv`. Assume Σ as the letters in the English language (i.e. `a-z` and `A-Z`). Your program can consider punctuation marks as a separator.

   Input/Output representation Following are the format in which finite automata and regular expression are to be described. Strictly adhere to these formats as your programs will be tested using our test inputs files which will be in this format.

   This format must be followed for both input and output.

   - **Representation of regular expression**
     1. First line : Alphabet set Σ with each alphabet separated by space.
     2. Second line : Regular expression.

     The regular expression must be a string of non-zero length containing alphabets from Σ with operations `+` (for union), `*` (for star closure). Parentheses '`(`', '`)`' must be supported. Expressions of length more than 1 that are to be concatenated must be parenthesised. Use '`@`' to denote the string of zero length ϵ and '`%`' to denote the empty language $\emptyset$.
     Example 1:

     ```
     a b
     (a+b)*
     ```

     Example 2:

     ```
     0 1
     (0+@)(1+@)
     ```

   - **Representation of finite automata**
     1. First line : No of states. If this is n, then the states must be numbered from `0` to `n-1`.
     2. Second line : Alphabet set Σ. Each alphabet must be seperated by a space.
     3. Third line : Accept states. States must be separated by a space in case of NFA. State `0` is assumed as the start state.
     4. Subsequent lines representing transitions. Each transition must start in a new line. A typical transition looks like `i j t` where $\texttt{i},\texttt{j} \in \{0, 1, \ldots, n-1\}$ and $\texttt{t} \in \Sigma \cup \{\texttt{@}\}$ and $\delta(\texttt{i},\texttt{t}) = \texttt{j}$
        Example : $\Sigma = \{a, b\}$, automata accepting even number of ones. A representation is as follows.

```
2
a b
1
0 0 a
0 1 b
1 0 b
1 1 a
```

Instructions and Guidelines

- A couple of test cases will be uploaded in course moodle page. Each part must be implemented in a separate file as there will be separate test cases for each of them. For the first four parts, input is to be taken from console. To avoid repeated typing, enter the input in a single file and use IO-redirection (`<`) feature to pass the input to the program. For example, if the file `fa1.txt` contains the description of example DFA given above, then do `./prog < fa1.txt` to test your program.
- Create a zip file containing all these files along with a `makefile` containing instructions for compilation and creation of executables. The `makefile` must produce executables for each part and must be named as "your-roll-no" followed by the part. For example, if your roll number if `CS15B001`, the executable for say part (c) must be named as `cs15b001c`. Please note that executable name must be in small case. Testing of programs will be automated. *Any deviation from this naming convention can cause your program to be not evaluated.* Please name the zip file with your roll number before uploading in course moodle page.
- You are suggested to implement the above for $\Sigma = \{$`a,b`$\}$ first and then generalise your implementation for an arbitrary Σ.
- In part (e), while passing regular expression, you may need to enclose the regular expression in double quotes to prevent parsing by shell. For example, `./prog "(a+b)*" textfile.txt`.
- Your submission will not be evaluated if there are compilation errors. Collaboration is not allowed. Any form of plagiarism will will be referred to the Institute's disciplinary committee.