German University in Cairo
Media Engineering and Technology
Assoc. Prof. Haythem Ismail

Compilers Lab, Spring term 2019

Task 1

Regular expression to NFA

Please read the following instructions carefully:

- Read Rules & regulations first
- It is **YOUR responsibility** to ensure that you have:
  - Submitted before the deadline.
  - Submitted the correct file(s).
  - Submitted the correct file(s) names.
  - Submitted correct logic of the task as it will be tested both publicly & privately.
  - Submitted your code in the format XX_XXXX_lab_1.zip where XX_XXXX is your ID for example 34_8000_lab_1.zip if your ID is 3 digits, append a zero to the left to be 34_0800_lab_1.zip to the correct google form link https://goo.gl/forms/Y7hr9BFDKCqHr1U22.

  .
- Good luck! =D

1 Generating Regular Expressions

In this part, you are required to:

- Generate the regular expressions.
- Suppose that the alphabets Σ is the set of ASCII characters.
- Follow the exact file name & output file name for each one.
- You will need to use folder task1 on MET website and you must follow the sample file in each and every ".py" file.
- You must follow the sample file provided for the command line format in each of your code files.

The output file should contain the strings that matches the regex under each other for example:

```
xxxxyyyyzzzz
xyz
xxyzz
```

1. Generate a regex to match the string "aabb" anywhere in each line.
   File name "task_1_1.py" & Output file name "task_1_1_result.txt"
2. Generate a regex to match one or more repetitions of the string "aabb" anywhere in each line where the sub string may be repeated for instance "aabbaabb" would be considered one match not two
   File name "task_1_2.py" & Output file name "task_1_2_result.txt"
3. Generate a regex to match one or more repetitions of the string "aabb" or "dc" where either can be repeat multiple times in the string.
   File name "task_1_3.py" & Output file name "task_1_3_result.txt"
4. Generate a regex to match any consecutive symbols {1, 2, 3, +, -, *, /, ., =} that is at least two and at most four characters long.
   File name "task_1_4.py" & Output file name "task_1_4_result.txt"
5. Generate a regex to match any consecutive symbol from digits preceded by a "=" (does not match '=')
   File name "task_1_5.py" & Output file name "task_1_5_result.txt"
6. Generate a regex to match any character that is not a 1 or 2 or + and is followed by an "=". (does not match '=')
   File name "task_1_6.py" & Output file name "task_1_6_result.txt"

7. Generate a regex to match a string that begins with a letter or more of the alphabet or ends with a digit or more of the alphabet.
   File name “task_1_7.py” & Output file name “task_1_7_result.txt”

8. Generate a regex using captures (whether named or not) to scan for struct (C-like) pointers and to print them appending “_new” to them. The format of a struct in C is “struct STRUCTNAME *STRUCTPOINTER”.
   File name “task_1_8.py” & Output file name “task_1_8_result.txt”

Example:
static struct rv7xx_power_info * rv770_get_pi(struct amdgpu_device *adev);

Result:
static struct rv7xx_power_info * rv770_get_pi_new(struct amdgpu_device *adev_new);

2 Regular expressions to NFA

In this part, you are required to implement Thompson's Construction that converts a regular expressions to it's equivalent NFA. Suppose that the alphabets Σ is the set of ASCII characters. **following the exact output file name for each one** & with the following format, you must follow the sample file on MET as well

Follow the exact file name "task_2.py" & output file name "task_2_result.txt".

Listing 1: Format

```
Line #1 state(s) separated by commas
e.g.: q0 ,q1, q2, ..., qn
Line #2 alphabet separated by commas
e.g.: a ,b, c, etc.
Line #3 start state
e.g.: q0
Line #4 final state(s) separated by commas
e.g.: q0 ,q1, q2, ..., qn
Line #5 transition(s) in a tuple form separated by commas
(start state, alphabet, result state in array form)
e.g.: (q0, a, [q0, q1, q3]), (q1, b, [q0])
```

For example (s|ε t)* would be:

```
q0,q1,q2,q3,q4,q5,q6,q7,q8
 ,s,t
q0
q8
(q0, , [q8]), (q0, , [q1]),  (q1, , [q2]), (q1, , [q4]),  (q2, s, [q3]), (q3, , [q7]),
 ↪  (q4, , [q5]), (q5, t, [q6]), (q6, , [q7]), (q7, , [q1]),  (q7, , [q8])
```

For epsilon transitions, use a space character
Supported:

- r = (s)
- r = st
- r = s|t
- r = s*
- r = s+
- r = s?
- r = ϵ

Convert following Regular expressions to NFA: (those are only test cases)

1. ϵ
2. a
3. ab
4. (ab)+
5. (a|b)*
6. (a*|b*)*
7. $((\epsilon|a)b?)*$
8. (a|b)*abb(a|b)*
9. (0|(1(01*(00)*0)*1)*)*