

Compilers Lab, Spring term 2019

Task 2

ϵ -NFA to DFA
&
Tokenization using ANTLR

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/3khKuW1S3WNrvPpx1>.
- .
- Good luck! =D

1 TOKENIZATION USING ANTLR

In this part, you are required to implement the regular definitions for the following languages. **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_1.py” & “task_2_1.g4” & output file name “task_2_1_result.txt”.

The ANTLR file (.g4) should contain the regular definitions for example:

```
1 // test.g4 file
2 grammar test;
3
4 NEWLINE      : [\r\n]+;
5 INT          : [0-9]+;
6 OPERATOR     : ('*' | '/' | '+' | '-');
7
8 expr         : INT OPERATOR INT
9               | OPERATOR INT
10              | INT;
11
12 start        : (expr)*;
```

The Python file should contain the code to tokenize the languages given & outputs the tokens, for example 100 + 5 * 30 would be:

```
1 100 INT
2 + OPERATOR
3 5 INT
4 * OPERATOR
5 30 INT
```

1. Create the regular definitions for Assembly x86 for 8086 processor.
 - a) Instructions allowed are: AAA, ADD & INC.
 - b) REGs allowed are AX, BX, CX, DX.
 - c) Memory structure: only one REG inside square brackets ex. [AX].
 - d) Immediate only positive numbers and binary numbers ex. 56, 0101100. b
 - e) Follow sample input on MET website for accepted structures (you may exclude other structures not present in sample folder).

2 ϵ -NFA TO DFA

In this part, you are required to implement the conversion from ϵ -NFA to DFA using epsilon closure. **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_2.py” & output file name “task_2_2_result.txt”.

Listing 1: Format

Line #1 state(s) separated by commas, the dead state is represented by "DEAD"
e.g.: A , B, C, ... , DEAD
Line #2 alphabet separated by commas
e.g.: a ,b, c, etc.
Line #3 start state
e.g.: A
Line #4 final state(s) separated by commas
e.g.: A, ,B, C, ...
Line #5 transition(s) in a tuple form separated by commas (state , alphabet , result state)
e.g.: (A, a, B), (A, b, C), (B, a, DEAD), ...

For example $(s|\epsilon t)^*$ the input will be:

```
1 q0,q1,q2,q3,q4,q5,q6,q7,q8
2 ,s,t
3 q0
4 q8
5 (q0, , q8), (q0, , q1), (q1, , q2), (q1, , q4), (q2, s, q3), (q3, , q7), (q4, , q5),
   ↪ (q5, t, q6), (q6, , q7), (q7, , q1), (q7, , q8)
```

The output would be:

```
1 A, B, C
2 s,t
3 A
4 A, B, C
5 (A, s, B), (A, t, C), (B, s, B), (B, t, C), (C, s, B), (C, t, C)
```