Regex Theory: Write a (Simple) Regex Compiler

Eckel, TJHSST AI2, Spring 2023

Background & Explanation

Put simply, your goal in this assignment is to implement the process from the previous assignment in code. Specifically, you'll get a regular expression, you'll convert it to a DFA, and then the DFA will be ready to run on input strings in a single left-to-right pass to determine a match.

This is **much simpler** than the full-featured regex we've used in the practical regex assignments. The grammar contains only the following:

- ab to specify two tokens in sequence (token a followed by token b)
- a|b to specify two different options (token a OR token b)
- (ab) to group a regular expression into a single token
- a* to specify any number, including zero, of a token (token a any number of times)
- a+ to specify any number, excluding zero, of a token (token a at least once)
- a? to specify an optional token (token a once or not at all)

…and does not contain any character classes. We also are only looking to see if the entire string matches or not, so you don't have to worry about finding a match on a substring.

This probably goes without saying, but **you may not import re to accomplish this assignment.** No fair using compiled regular expressions to compile regular expressions!

Input/Output Specification

You'll receive three command-line arguments – a language, a regular expression, and a txt file with test cases. You will compile the regular expression to a DFA and then run it on all the test cases. The required output is just like the Deterministic Finite Automata assignment – you'll print out a table showing the state connections, a list of final nodes, and the results of tests on the given strings.

Sample Runs

To help you out, I've given a couple of sample runs where I'm outputting what my computer does at each stage of the conversion process. **This is not required for you** – you only need to output the end. But hopefully this will help you debug.

On each of these sample runs, note that it is quite likely that you won't end up with your states numbered precisely as mine have been. This is ok. You should be able to take your own states and find a corresponding state to each of them in my output (the fancy mathematical term for this is that your solutions don't need to be identical to mine but they do need to be **isomorphic** to mine).

Finally, I want to point out a choice I made here that you don't have to also do, but that might be helpful. Note that during the NFA stages, I've made each state be a dict of characters to **tuples** so that I can store multiple connections with the same character. When we get to DFA, I convert back to the format I showed in the DFA assignment.

```
> compile_regex.py ab a|(ab|b*a)* dfa_ex_tests.txt

Converted to NFAwe:
*           eps         a           b
0           (2,)        (1,)        _
1           _           _           _
2           (5, 1)      (3,)        _
3           _           _           (2,)
4           _           (2,)        _
5           (4,)        _           (5,)
Final: [1]

Converted to NFA:
*           a           b
0           (1, 3, 2)   (5,)
1           _           _
2           (3, 2)      (5,)
3           _           (2,)
4           (2,)        _
5           (2,)        (5,)
Final: [0, 1, 2]

While converting to DFA, here are my new states: {(1, 2, 3): 6, (2, 3): 7, (2, 5): 8}
Converted to DFA:
*           a           b
0           6           5
1           _           _
2           7           5
3           _           2
4           2           _
5           2           5
6           7           8
7           7           8
8           7           5
Final: [0, 1, 2, 6, 7, 8]

While cleaning, after removing states unreachable from start:
*           a           b
0           6           5
2           7           5
5           2           5
6           7           8
7           7           8
8           7           5
Final: [0, 2, 6, 7, 8]

While removing redundant nodes, each tuple represents a pair of redundant nodes:
{(6, 7), (2, 8)}
{(0, 2)}
set()
^ The final 'set()' is on the last time through my redundancy search loop where no redundancies are found.
Cleaned DFA (everything after THIS POINT is the output YOU need to have):
*           a           b
0           6           5
5           0           5
6           6           0
Final: [0, 6]
True  a
False b
True  aa
False bb
True  aba
False bab
True  abba
True  baab
True  ababa
False babab
False abbbbb
True  baaaaa
True  abbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbba
True  baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab
True  abbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaa
False baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabb
```

```
> compile_regex.py ab a(a|b*)a|b+ dfa_ex_tests.txt

Converted to NFAwe:
*           eps         a           b
0           _           (2,)        (5,)
1           _           _           _
2           (4,)        (3,)        _
3           _           (1,)        _
4           (3,)        _           (4,)
5           (1,)        _           (5,)
Final: [1]

Converted to NFA:
*           a           b
0           (2,)        (5,)
1           _           _
2           (3, 1)      (4,)
3           (1,)        _
4           (1,)        (4,)
5           _           (5,)
Final: [1, 5]

While converting to DFA, here are my new states: {(1, 3): 6}
Converted to DFA:
*           a           b
0           2           5
1           _           _
2           6           4
3           1           _
4           1           4
5           _           5
6           1           _
Final: [1, 5, 6]

While cleaning, after removing states unreachable from start:
*           a           b
0           2           5
1           _           _
2           6           4
4           1           4
5           _           5
6           1           _
Final: [1, 5, 6]

While removing redundant nodes, each tuple represents a pair of redundant nodes:
set()
^ The final 'set()' is on the last time through my redundancy search loop where no redundancies are found.
Cleaned DFA (everything after THIS POINT is the output YOU need to have):
*           a           b
0           2           5
1           _           _
2           6           4
4           1           4
5           _           5
6           1           _
Final: [1, 5, 6]
False a
True  b
True  aa
True  bb
True  aba
False bab
True  abba
False baab
False ababa
False babab
False abbbbb
False baaaaa
True  abbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbba
False baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab
False abbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaa
False baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabb

Process finished with exit code 0
```

```
> compile_regex.py abc (abc?|(cc)+)* dfa_ex_tests_2.txt   (THIS IS A VERY BAD REGEX, but it demonstrates redundancy well)

Converted to NFAwe:
*           eps         a           b           c
0           (2,)        _           _           _
1           _           _           _           _
2           (1,)        (3,)        _           (6,)
3           _           _           (4,)        _
4           (2,)        _           _           (2,)
5           (2,)        _           _           (7,)
6           _           _           _           (5,)
7           _           _           _           (5,)
Final: [1]

Converted to NFA:
*           a           b           c
0           (3,)        _           (6,)
1           _           _           _
2           (3,)        _           (6,)
3           _           (4,)        _
4           (3,)        _           (2, 6)
5           (3,)        _           (7, 6)
6           _           _           (5,)
7           _           _           (5,)
Final: [0, 1, 2, 4, 5]

While converting to DFA, here are my new states: {(2, 6): 8, (6, 7): 9, (5, 6): 10, (5, 6, 7): 11}
Converted to DFA:
*           a           b           c
0           3           _           6
1           _           _           _
2           3           _           6
3           _           4           _
4           3           _           8
5           3           _           9
6           _           _           5
7           _           _           5
8           3           _           10
9           _           _           5
10          3           _           11
11          3           _           11
Final: [0, 1, 2, 4, 5, 8, 10, 11]

While cleaning, after removing states unreachable from start:
*           a           b           c
0           3           _           6
3           _           4           _
4           3           _           8
5           3           _           9
6           _           _           5
8           3           _           10
9           _           _           5
10          3           _           11
11          3           _           11
Final: [0, 4, 5, 8, 10, 11]

While removing redundant nodes, each tuple represents a pair of redundant nodes:
{(10, 11), (6, 9)}
{(8, 10), (0, 5)}
{(4, 8)}
set()
^ The final 'set()' is on the last time through my redundancy search loop where no redundancies are found.
Cleaned DFA (everything after THIS POINT is the output YOU need to have):
*           a           b           c
0           3           _           6
3           _           4           _
4           3           _           4
6           _           _           0
Final: [0, 4]
False a
False b
False c
True  ab
False ac
True  cc
True  abc
True  abcc
True  abccc
True  abcababcababababcabcababc
True  abcababababccababccccccababcccab
False abbabccabccabcabcc
```

These last two are complex enough that I haven't provided all the output. Be sure to check the test cases to make sure your True and False outputs match. If you have problems with these and this isn't enough info to help, let me know!

```
> compile_regex.py "abcd" "(a|b*)c|(a+d*c+|a+(cc)+)?b" dfa_ex_tests_3.txt

*               a               b               c               d
0               15              16              1               _
1               _               _               _               _
4               _               4               1               _
10              _               _               11              10
11              _               1               11              _
15              17              _               18              10
16              _               4               1               _
17              17              _               19              10
18              _               1               20              _
19              _               1               20              _
20              _               1               19              _
Final nodes: {16, 1, 18}
False
False a
True  b
True  c
False d
False aa
False bb
False cc
False dd
False abc
True  acb
False bdc
False abcdb
True  bbbbbbc
True  adcb
True  addcccb
True  acccb
False adddb
False dcba
False cca
False cccbbbbba
False cbbbb

Process finished with exit code 0
```

```
> compile_regex.py "abcd" "(a|b|c)+d(a|b|c|d)*|(a|b)+(a|b|d)*c(a|b|c|d)*|a(a|c|d)*b(a|b|c|d)*" dfa_ex_tests_3.txt

*               a               b               c               d
0               17              18              4
                                                                _
3               3               3               3               3
4               4               4               4               3
17              17              21              22              23
18              18              18              25              26
21              21              21              28              29
22              22              28              22              32
23              23              29              32              23
25              25              25              25              37
26              26              26              37              26
28              28              28              28              41
29              29              29              41              29
32              32              41              32              32
37              37              37              37              37
41              41              41              41              41
Final nodes: {3, 21, 22, 23, 25, 26, 28, 29, 32, 37, 41}
False
False a
False b
False c
False d
False aa
False bb
False cc
False dd
True  abc
True  acb
True  bdc
True  abcdb
True  bbbbbbc
True  adcb
True  addcccb
True  acccb
True  adddb
False dcba
False cca
False cccbbbbba
False cbbbb

Process finished with exit code 0
```

Specification

Submit a single Python script to the link on the course website.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your code takes in 3 command line arguments, as specified above, and compiles and runs a regex, as specified above. Pay careful attention to the output you should have; note that you don't need everything from my sample output.
- **YOUR CODE MAY NOT IMPORT `re`.** No fair using compiled regexes to compile regexes.