Discussion 2

Discussion 2 - Thursday, June 19th

Reminders

1. Project 3 released, due **Wednesday, July 2nd @ 11:59 PM**

Topic List

- Regular Expressions
- NFAs and DFAs

Review - Regular Expressions

There are many patterns regex can describe aside from string literals.

- **Concatenation (and)**: `ab` We use this to accept something that satisfies a and b in th er, where a and b can denote sub-regex.
  - Ex. `a` matches "a", `b` matches "b", so `ab` matches "ab"
  - Ex. `(a|b)` matches "a" or "b", `c` matches "c", so `(a|b)c` matches "ac" or "bc"
- **Union (or)**: `a|b|c` We use this to accept something from given choices. **Note** that a, b, or c can also denote sub-regex if parentheses are specified.
  - Ex. `a|b|c` matches "a" or "b" or "c"
- **Precedence (parentheses)**: `(a)` are used to enforce order of evaluation and capture groups.
  - Ex. `a|bc` matches "a" or "bc". This is the same as `a|(bc)`
  - Ex. `(a|b)c` matches "ac" or "bc"
- **Sets**: `[abc]` We use this to accept one character from the given choices.
  - Ex. `[abc]` matches "a" or "b" or "c"
- **Ranges**: `[a-z]`, `[c-k]`, `[A-Z]`, `[0-9]` We use these ranges, also known as character classes, to accept characters within a specified range (inclusive).
  - Ex. `[a-z]` matches any lowercase letter
  - Ex. `[c-k]` matches letters c to k inclusive
  - Ex. `[A-Z]` matches any uppercase letter
  - Ex. `[0-9]` matches any digit
  - Ex. `[a-z0-9]` matches any lowercase letter or digit
- **Negation**: `[^abc]` `[^a-z]` `[^0-9]` We use these to exclude a set of characters.
  - Ex. `[^abc]` matches with any character other than "a", "b", or "c"
  - Ex. `[^a-z]` matches with any character that is not a lowercase letter

  - Ex. `[^0-9]` matches with any character that is not a digit
  - Note that the use of "^" differs from the beginning of a pattern
- **Meta Characters**: `\d`, `\D`, `\s`, `\w`, `\W` We use these characters to match on any of a particular type of pattern.
  - ex. `\d` matches any digit (equivalent to `[0-9]`)
  - ex. `\D` matches any character that is not a digit (equivalent to `[^0-9]`)
  - ex. `\s` matches any whitespace character (spaces, tabs, or newlines)
  - ex. `\w` matches any alphanumeric character from the basic Latin alphabet, including the underscore (equivalent to `[A-Za-z0-9_]`)
  - ex. `\W` matches any character that is not a word character from the basic Latin alphabet (equivalent to `[^A-Za-z0-9_]`)
- **Wildcard**: `.` We use this to match on **any** single character. Note: to use a literal `.`, we must escape it, i.e. `\.`
- **Repetitions**: `a*`, `a+`, `a?`, `a{3}`, `a{4,6}`, `a{4,}`, `a{,4}`:
  - Ex. `a*` matches with 0 or more a's
  - Ex. `a+` matches with at least one a
  - Ex. `a?` matches with 0 or 1 a
  - Ex. `a{3}` matches with exactly three a's
  - Ex. `a{4,6}` matches with 4, 5, or 6 a's
  - Ex. `a{4,}` matches with at least 4 a's
  - Ex. `a{,4}` matches with at most 4 a's
  - **Note**: a can denote a sub-regex
- **Partial Match**: `a` and `abc` These patterns can match any part of a string that contains the specified characters.
  - Ex. `a` matches "a", "ab," "yay," or "apple"
  - Ex. `abc` matches "abc", "abcdefg," "xyzabcjklm," or "abc123"
  - **Note**: They do not require the specified sequence to be at the beginning or end of the string
- **Beginning of a pattern**: `^hello` The string must begin with "hello".
  - Ex. `^hello` matches with "hellocliff" but does not match with "cliffhello"
- **End of a pattern**: `bye$` The string must end with "bye".
  - Ex. `bye$` matches with "cliffbye" but does not match with "byecliff"
- **Exact Match**: `^hello$` The string must be exactly "hello".
  - Ex. `^hello$` only matches "hello" and no other string
  - **Note**: Enforces both the beginning and end of the string

> **Question**: *Can every string pattern be expressed with a regex?*

Answer: No!

There are certain string patterns that **cannot** be expressed with regex. This is because regex is memoryless; as they cannot keep track of what they have already seen.

As an example, consider a pattern that represents all palindromes, e.g. "racecar". We can't track how many of each character we have previously seen (assuming our regex engine doesn't have backreferences).

Exercises - Regular Expressions

Write a regex pattern for each of the following scenarios (or explain why you cannot):

1. **Exactly** matches a string that alternates between capital & lowercase letters, starting with capital letters. Single-character strings with just one capital letter and empty strings should be allowed.
   - Includes: "AaBbCc", "DlFsPrOa", "HiWoRlD"
   - Excludes: "aAbBcC", "aaa", "123"
2. Matches a string that contains an even number of 3s, and then an odd number of 4s.
   - Includes: "3333444", "334", "3333334444444", "4"
   - Excludes: "34", "33344", "334444", "1111222"
3. Matches a string that contains a phone number following the format (XXX)-XXX-XXXX where X represents a digit.
   - Includes: "(123)-456-7890", "(111)-222-3333"
   - Excludes: "123-456-7890", "1234567890"
4. **Exactly** matches a string email following the format [Directory ID]@umd.edu where [Directory ID] is any sequence consisting of lowercase letters (a-z), uppercase letters (A-Z), or digits (0-9) with length >= 1.
   - Includes: "colemak123@umd.edu", "ArStDhNeIo@umd.edu", "b@umd.edu"
   - Excludes: "qwerty@gmail.com", "@umd.edu"
5. Matches a string that has more 7s, 8s, and 9s than 1s, 2s, and 3s.
   - Includes: "7891", "123778899", "12789", "8"
   - Excludes: "1", "271", "12399", "831"

Solutions

▼ Click here!

1. `/^([A-Z][a-z])*([A-Z])?$/` 2. `/(33)*4(44)*/` 3. `/\([0-9]{3}\)-[0-9]{3}-[0-9]{4}/` (Note, we have to escape the parenthesis with `\`) 4. `/^[a-zA-Z0-9]+@umd\.edu$/` (Note, we have to escape the period with `\`) 5. Cannot be represented with regular expressions, since there is no memory of which numbers have been previously used.

NFAs and DFAs

Notes:

Key differences between NFA and DFA

- All DFAs are NFAs, but not all NFAs are DFAs.
- NFA can have ε-transition(s) between states.
- NFA states can have multiple transitions going out of them using the same symbol.
- DFAs are computationally cheaper to process, but often harder to read compared to NFAs.

Exercises

Regex -> NFA

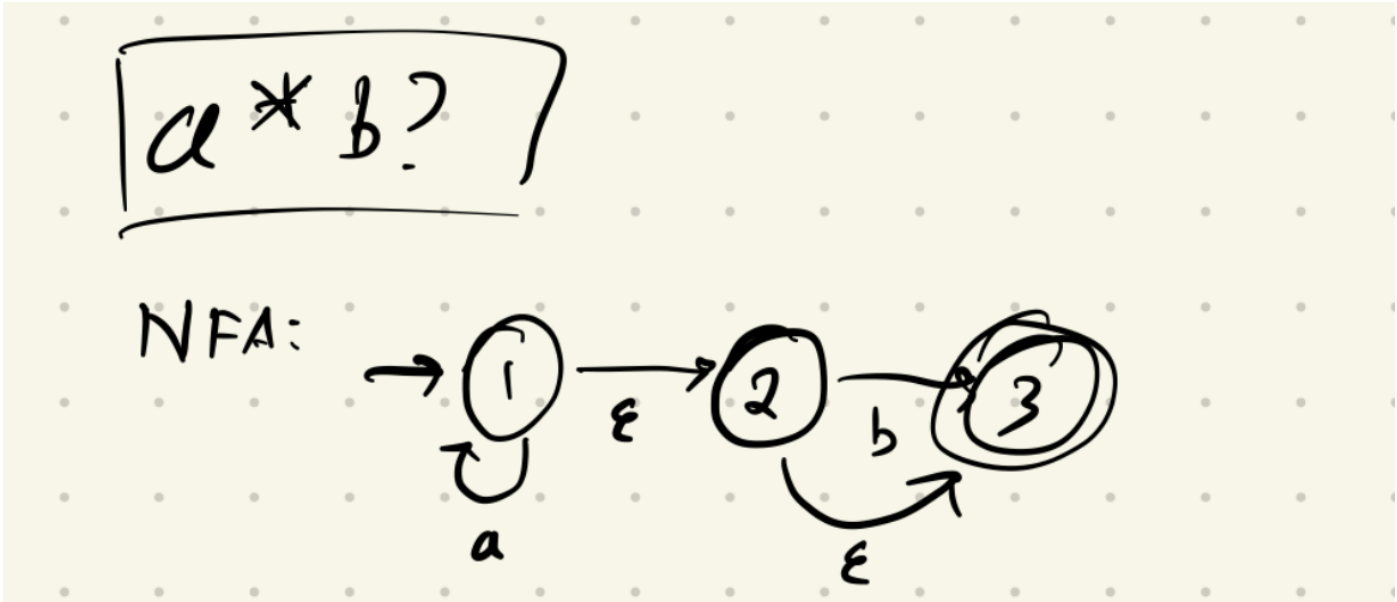
1. Consider the following regular expressions:

```
a) a*b?
b) (b|c)+
c) a*b?(b|c)+
```

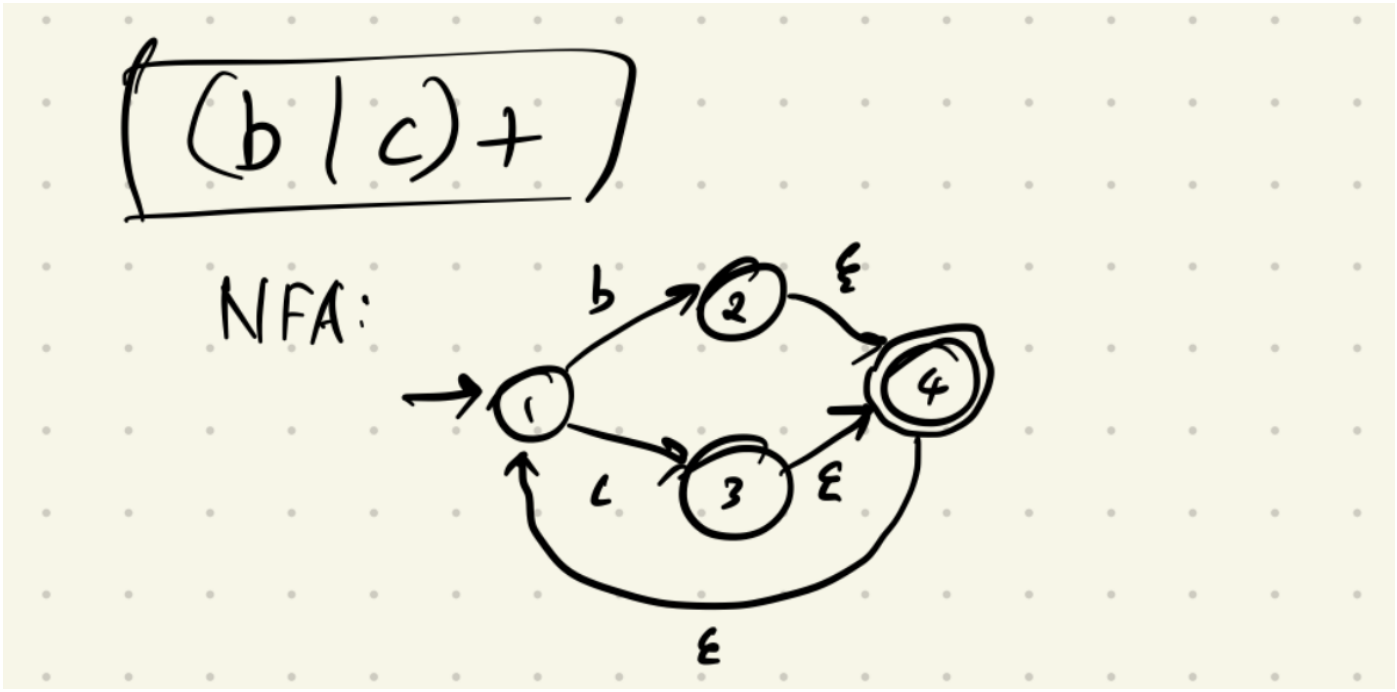
   - Convert each regex to an equivalent NFA
     - Note that there are many valid NFAs
   - Convert each NFA to its equivalent DFA
   - Compare your DFA with the person next to you
     - Are they the same?

▼ Solutions!

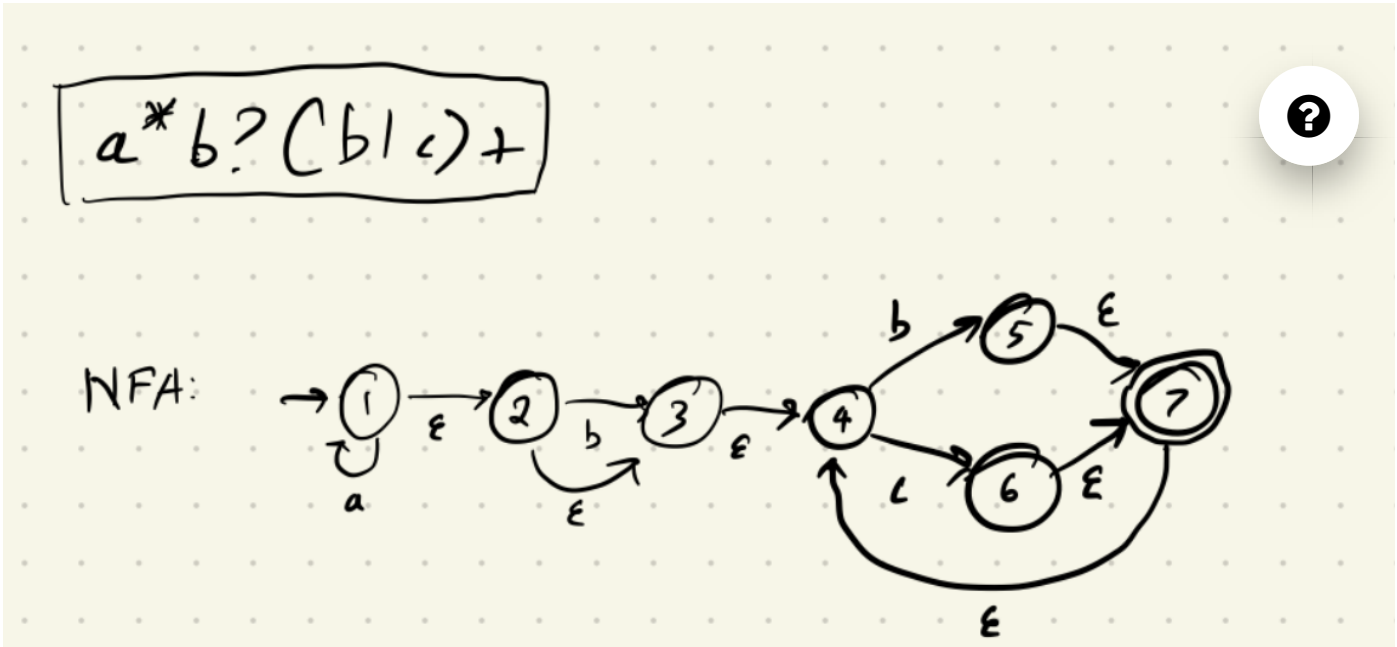
a)



b)



c)



Resources & Additional Readings

- Fall 2023 Python HOF + Regex discussion (https://github.com/cmsc330fall23/cmsc330fall23/tree/main/discussions/d2_hof_regex)
- Online Regular Expression Tester (https://regexr.com/)
- Regex Practice Problem Generator (https://apabla1.github.io/)
- Fall 2023 Discussion - NFA and DFA (https://github.com/cmsc330fall23/cmsc330fall23/tree/main/discussions/d3_nfa_dfa)
- Fall 2023 Discussion - NFA and DFA Conversion (https://github.com/cmsc330fall23/cmsc330fall23/tree/main/discussions/d4_nfa_dfa_conversion)