AB IdeaLab, Competitive Programming Team, Fall 18–Spring 19 February 8, 2019

Lecture 3: FSA/Regular Expressions

American Computer Science League, February Contest

Primary Editor: Sanjit Bhat *Secondary Editor: Alexander Sun*

1 Fun Facts

- Developed in 1951 by mathematician Stephen Cole Kleene.
- Ken Thompson (one of the guys who developed UNIX) used regular expressions on an early Unix editor. This eventually lead to its use in the famous UNIX tool grep.
- Applications include string searching algorithms, input verification, and search engines.
- You can even use it inside your programming editor to find where you've put stuff.

2 Background

What are regular expressions? According to Wikipedia, regular expressions (regex) are "a sequence of characters that define a search pattern." In other words, a regex defines a set of possible strings in a concise manner for some later purpose. For example, reali[sz]e defines the set {realize, realise} of possible strings. This set can be later used for cross-referencing American-English spellings with British-English spellings.

All the regex syntax you need to know. Regex includes *metacharacters* that define more complex types of string matching. The following is a list of all the regex metacharacters you need to know:

1. **|, or,** ∪ These are booleans that tell the processor to take the set union of the regexes on the left- and right-hand sides. For instance, gr(a|e)y, gray or grey, and gr(a∪e)y all define the set {gray, grey}.
2. λ The null or empty string.
3. **Quantification** Defining the number of something allowed to occur. Note that these all operate on a regex left of the operator.
   (a) **?** Zero or one. E.g., colou?r = {color, colour}.
   (b) ***** Zero or more. This is also called the Kleene Star (named after the inventor, Stephen Kleene).
   (c) + One or more.
4. **.** Wildcard (a fill in for any character). Combine . and * for a.*b, which accepts any string with a and b as the leftmost and rightmost characters, respectively, with an arbitrary number of arbitrary characters inbetween.
5. **[. . .]** Set of possible character matches. Think the reali[sz]e example above. This can get slightly more complex by using hyphens to define ranges of possible characters. E.g., [a-z] means every *lowercase* char from a to z; [abcx-z] means a, b, c, and x, y, z; and [a-cx-z] means a, b, c and x, y, z.
6. **[ˆ. . .]** Set of characters not contained withing the brackets. E.g., [ˆa-z] matches any character that is not a lowercase character from a to z.

7. **()** Just like in math, parentheses imply grouping. E.g., if we wanted the set {gray, grey}, gra|ey would give us {gra, ey}. Instead, using parentheses we can get gr(a|e)y, which gives us the correct regex. A more complex example is H(ä|ae?)ndel, which matches {Handel, Händel, Haendel}.

Order of operations: Kleene Star (*), concatenation (ab), and union(∪). Because Kleene Star has the highest priority, a.*b accepts a string with an arbitrary number of *several different arbitrary* characters (e.g., {acdb, ...}), as opposed to only an arbitrary number of a single arbitrary character (e.g., {accb, ...}).

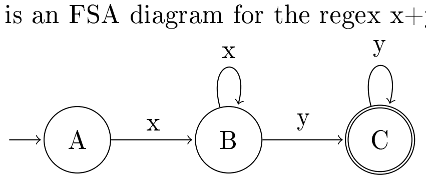
Practicing the syntax via identity proofs. To make sure you understand the syntax and order of operations, see if you can prove the following identities:

1. (a*)* = a*
2. aa* = a*a
3. aa* ∪λ = a*
4. a(b ∪ c) = ab ∪ ac
5. a(ba)* = (ab)*a
6. (a ∪ b)* = (a* ∪ b*)*
7. (a ∪ b)* =(a*b*)*
8. (a ∪ b)* = a*(ba*)*

How are regex interpreted by the computer? In a regex, there are two types of chars: literals and metacharacters. Literals define regular characters, while metacharacters indicate more nuanced behaviors. After creating a regex, a regex processor transforms the characters into an internal representation that can be thought of as a Finite State Automata (FSA). FSAs are an abstract concept in theoretical computer science consisting of the following:

1. A finite number of states, of which exactly one is active at any given time
2. Transition rules to change the active state
3. An initial state
4. One or more final states

We can draw an FSA by representing each state as a circle, the final state as a double circle, the start state as the only state with an incoming arrow, and the transition rules as labeled-edges connecting the states. For instance, the following is an FSA diagram for the regex x+y+:



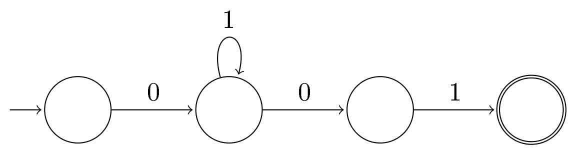
If you would like to learn more about FSAs, I recommend the Wikipedia page. Outside the ACSL bubble, automata and finiteness are an important field of research in theoretical CS. They connect back to problems such as P vs. NP and whether a program will stop in a reasonable amount of time or even in an infinite amount of time.

Testing regex syntax. If you would like to practice regex and have your code actually matched against strings, I recommend this website.

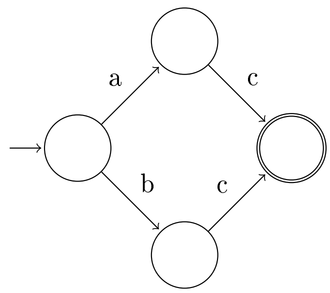
3 Exercises

3.1 Translate an FSA to a Regular Expression

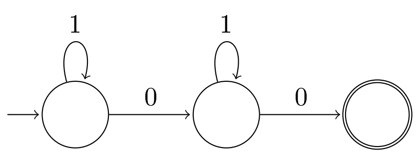
1. Find a simplified Regular Expression for the following FSA:

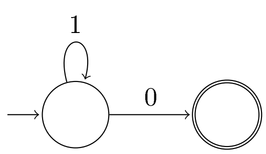


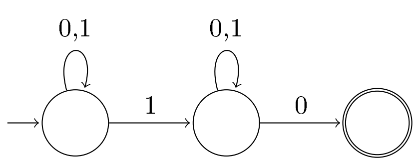
2. Find a simplified Regular Expression for the following FSA:

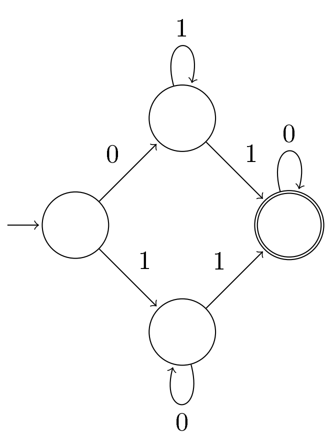


3. List all of the following FSAs which represent 1*01*0:

(a) 

(b) 

(c) 

(d) 

3.2 Simplify a Regular Expression

3.3 Determine which Regular Expressions or FSAs are equivalent

1. Which, if any, of the following Regular Expressions are equivalent?
   (a) (a∪b)(ab*)(b*∪a)
   (b) (aab*∪bab*)a
   (c) aab*∪bab*∪aaba∪bab*a
   (d) aab*∪bab*∪aab*a∪bab*a
   (e) a*∪b*

3.4 Determine which strings are accepted by either an FSA or a Regular Expression

1. Which of the following strings are accepted by the following Regular Expression "00*1*1U11*0*0"?
   (a) 0000001111111
   (b) 1010101010
   (c) 1111111
   (d) 0110
   (e) 10
2. Which of the following strings match the regular expression pattern "[A-D]*[a-d]*[0-9]"?
   (a) ABCD8
   (b) abcd5
   (c) ABcd9
   (d) AbCd7
   (e) X
   (f) abCD7
   (g) DCCBBBaaaa5
3. Which of the following strings match the regular expression pattern "Hi?g+h+[^a-ceiou]"?
   (a) Highb
   (b) HiiighS
   (c) HigghhhC
   (d) Hih
   (e) Hghe
   (f) Highd
   (g) HgggggghX

4 Solutions

4.1 Answers for Section 3.1

1. 01*01
2. (a|b)c or ac ∪ bc
3. a. The other choices correspond to 1*0, (0∪1)*1(0∪1)*0, and 01*10*∪10*10*

4.2 Answers for Section 3.3

1. B is different from the rest because it requires an ending 'a'. E is different from the rest because it doesn't allow for alternating a's and b's. C and D are different because of the third 'or' condition. Upon very close inspection, A and D are equivalent (check this carefully yourself). Therefore, A and D are the answers.

4.3 Answers for Section 3.4

1. 0000001111111 and 10
2. ABCD8, abcd5, ABcd9, and DCCBBBaaaa5
3. HigghhhC, Highd, and HggggggghX