Chapter 9: FSAs and Reg Exs

WMC CS CLUB

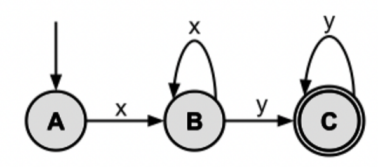
March 24, 2022

§1 Basics of FSA and Regular Expressions

A **Finite State Automaton** (FSA) is a mathematical model of computation comprising all 4 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; and 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.

Example 1.1

Parse the following FSA diagram.



Explanation. In the above FSA, there are three states: A, B, and C. The only way to go from state A to B is by seeing the letter x. Once in state B, there are two transition rules: seeing the letter y will cause the FSA to make C the active state, and seeing an x will keep B as the active state. State C is a final state so if the string being parsed is completed and the FSA is in State C, the input string is said to be accepted by the FSA. In State C, seeing any additional letter y will keep the machine in state C. The FSA above will accept strings composed of one or more x's followed by one or more y's (e.g., xy, xxy, xxxyy, xyyy, xxyyyy).

A **Regular Expression** (RE) is an algebraic representation of an FSA. Think of it as an abstraction of FSA, as if numbers are an abstraction of counting. For example, the regular expression corresponding to the FSA given above is xx*yy*. The rules for forming a RE are as shown below. Figure 1 on the right shows some common identities of Regular Expressions.

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

Figure 1. RE Identities

1. The null string (λ) is a RE.
2. If the string a is in the input alphabet, then it is a RE.
3. if a and b are both REs, then so are the strings built up using the following rules:
   a) **CONCATENATION**. "ab" (a followed by b).
   b) **UNION**. "a ∪ b" or "a | b" (a or b).
   c) **CLOSURE**. "a*" (a repeated zero or more times). This is known as the Kleene Star.

Remark 1.2 (Order of Precedence)**.** The order of precedence for Regular Expression operators is: Kleene Star, concatenation, and then union. For example, "dca*b" generates strings dcb, dcab, dcaab, etc. However, "d(ca)*b" generates strings db, dcab, dcacab, etc.

§2 RegEx in Practice

Programmers use Regular Expressions (usually referred to as regex) extensively for expressing patterns to search for. All modern programming languages have regular expression libraries. Here are the syntax rules that we will use.

Pattern	Description
|	A vertical bar separates alternatives. For example, gray | grey can match "gray" or "grey".
*	The asterisk indicates zero or more occurrences of the preceding element. For example, ab*c matches "ac", "abc", "abbc", "abbbc", and so on.
?	The question mark indicates zero or one occurrences of the preceding element. For example, colou?r matches both "color" and "colour".
+	The plus sign indicates one or more occurrences of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac".
.	The wildcard. matches any character. For example, a.b matches any string that contains an "a", then any other character, and then a "b" such as "a7b", "a&b", or "arb", but not "abbb". Therefore, a.*b matches any string that contains an "a" and a "b" with 0 or more characters in between. This includes "ab", "acb", or "a123456789b".
[]	A bracket expression matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z".
[^]	Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.
()	Parentheses define a sub-expression. For example, the pattern `H(ä|ae?)ndel` matches "Handel", "Händel", and "Haendel".

Example 2.1

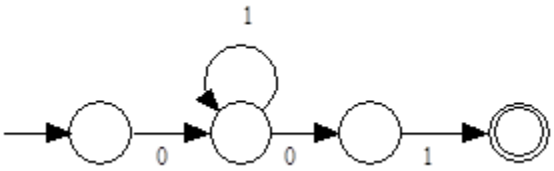
Which of the following strings match the RE pattern "[A-D]*[a-d]*[0-9]"?
1. ABCD8 **2.** abcd5 **3.** ABcd9 **4.** AbCd7 **5.** X **6.** abCD7
7. DCCBBBaaaa5

Explanation. The pattern describes strings the start with zero or more uppercase letters A, B, C, or D (in any order), followed by zero or more lowercase letter a, b, c, or d (in any order), followed by a single digit. The strings that are represented by this pattern are 1, 2, 3, and 7.

§3 Practice Problems

§3.1 Problems

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



2. Which of the following strings are accepted by the following Regular Expression "00*1*1 U 11*0*0"?
   A. 000000111111 **B.** 1010101010 **C.** 1111111 **D.** 0110 **E.** 10

3. Which of the following strings match the RE pattern `Hi?g+h+[^a-ceiou]`?
   1. Highb **2.** HiiighS **3.** HigghhhC **4.** Hih **5.** Hghe **6.** Highd
   7. HgggggghX

§3.2 Answer Key

1. The expression 01*01 is read directly from the FSA. It is in its most simplified form.

2. This Regular Expression parses strings described by the union of 00*1*1 and 11*0*0. The RE 00*1*1 matches strings starting with one or more 0s followed by one or more 1s: 01, 001, 0001111, and so on. The RE 11*0*0 matches strings with one or more 1s followed by one or more 0s: 10, 1110, 1111100, and so on. In other words, strings of the form: 0s followed by some 1s; or 1s followed by some 0s. Choice **A** and **E** following this pattern.

3. The ? indicates 0 or 1 "i"s. The + indicates 1 or more "g"s followed by 1 or more "h"s. The ∧ indicates that the last character cannot be lower-case a, b, c, e, i, o, or u. The strings that are represented by this pattern are **3**, **6**, and **7**.