FSAs and Regular Expressions

ACSL Contest #3

What is an FSA?

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*
4. one or more *final states.*

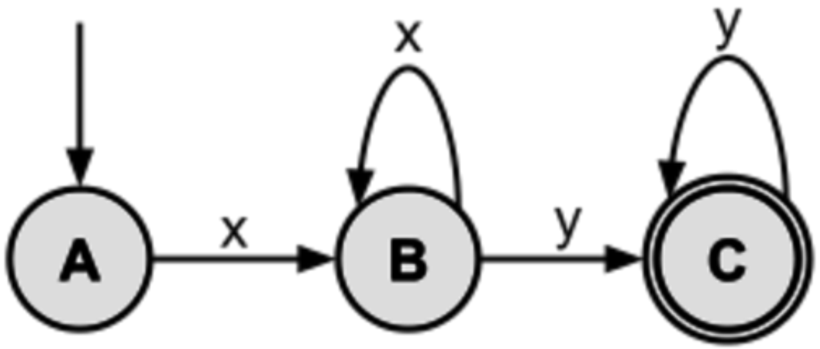
We can draw an FSA by representing each state as a circle, the final state as a double circle, the start state as the only state with an incoming arrow, and the transition rules as labeled-edges connecting the states. When labels are assigned to states, they appear inside the circle representing the state.

In this category, FSAs will be limited to parsing strings. That is, determining if a string is valid or not.

FSA Basics

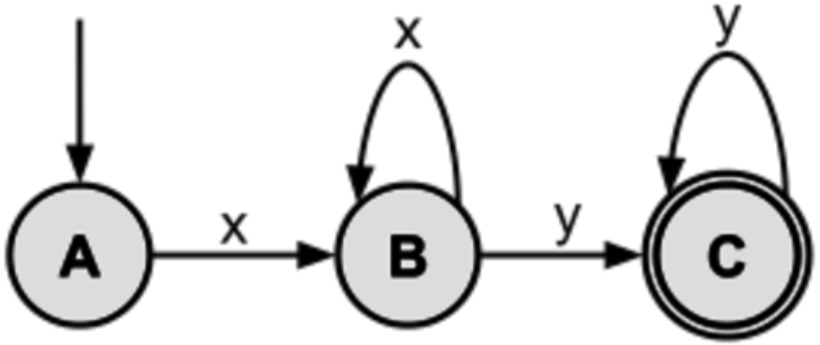
Let’s take a look at an FSA used to parse strings consisting of x’s and y’s:

- Consists of three states: A (initial state), B, C (final state)
- 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.


$$\begin{tikzcd} \arrow[r] & A \arrow[r, "x"] & B \arrow[loop above, "x"] \arrow[r, "y"] & C \arrow[loop above, "y"] \end{tikzcd}$$

FSA Basics

- 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 to the right will accept strings composed of one or more x's followed by one or more y's (e.g., xy, xxy, xxxyy, xyyy, xxyyyy).



Regular Expressions

A Regular Expression (RE) is an algebraic representation of an FSA. For example, the regular expression corresponding to the first FSA from the last slide is xx*yy*.

The rules for forming a Regular Expression (RE) are as follows:

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. "aUb" or "a|b" (a or b).

    c. CLOSURE. "a*" (a repeated zero or more times). This is known as the Kleene Star.

Regular Expressions Order of Precedence

The order of precedence for Regular Expression operators is:

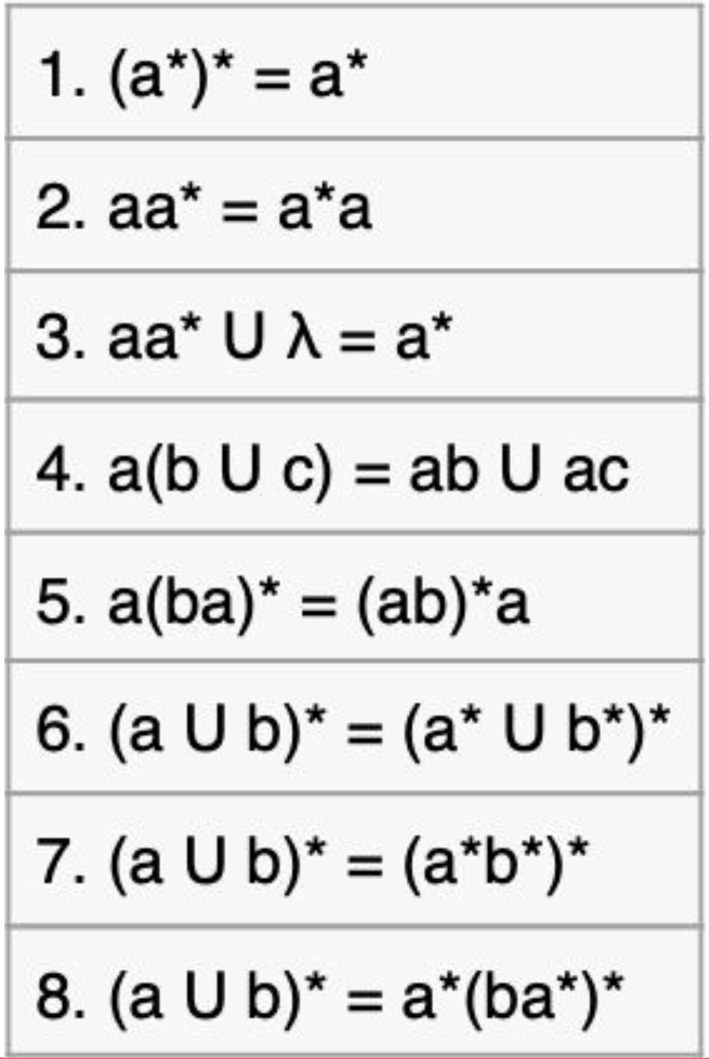
1. Kleene Star
2. Concatenation
3. Union.

Similar to standard Algebra, parentheses can be used to group sub-expressions.

- For example, "dca*b" generates strings dcb, dcab, dcaab, and so on, whereas "d(ca)*b" generates strings db, dcab, dcacab, dcacacab, and so on.

If we have a Regular Expression, then we can mechanically build an FSA to accept the strings which are generated by the Regular Expression. Conversely, if we have an FSA, we can mechanically develop a Regular Expression which will describe the strings which can be parsed by the FSA. For a given FSA or Regular Expression, there are many others which are equivalent to it. A "most simplified" Regular Expression or FSA is not always well defined.

Regular Expression Identities



1. $(a^*)^* = a^*$
2. $aa^* = a^*a$
3. $aa^* \cup \lambda = a^*$
4. $a(b \cup c) = ab \cup ac$
5. $a(ba)^* = (ab)^*a$
6. $(a \cup b)^* = (a^* \cup b^*)^*$
7. $(a \cup b)^* = (a^*b^*)^*$
8. $(a \cup b)^* = a^*(ba^*)^*$

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.

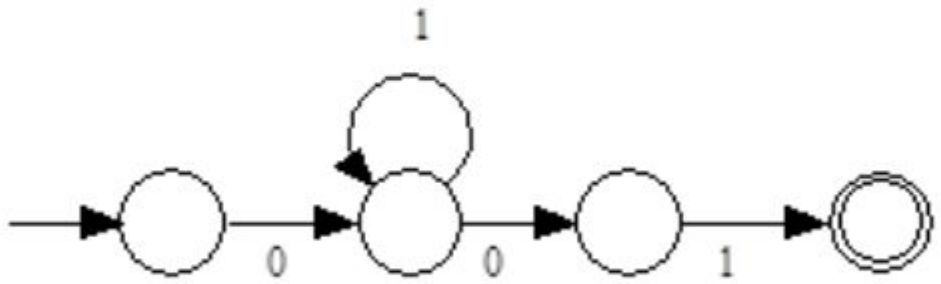
Unfortunately, the specific syntax rules vary depending on the specific implementation, programming language, or library in use. Interactive websites for testing regexes are a useful resource for learning regexes by experimentation. An excellent online tool is https://regex101.com/. A very nice exposition is Pattern Matching with Regular Expressions from the Automate the Boring Stuff book and online course.

Pattern	Description
|	As described above, a vertical bar separates alternatives. For example, gray|grey can match "gray" or "grey".
*	As described above, 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", as does [a-cx-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.
( )	As described above, parentheses define a sub-expression. For example, the pattern H(ä|ae?)ndel matches "Handel", "Händel", and "Haendel".

Sample Problem 1

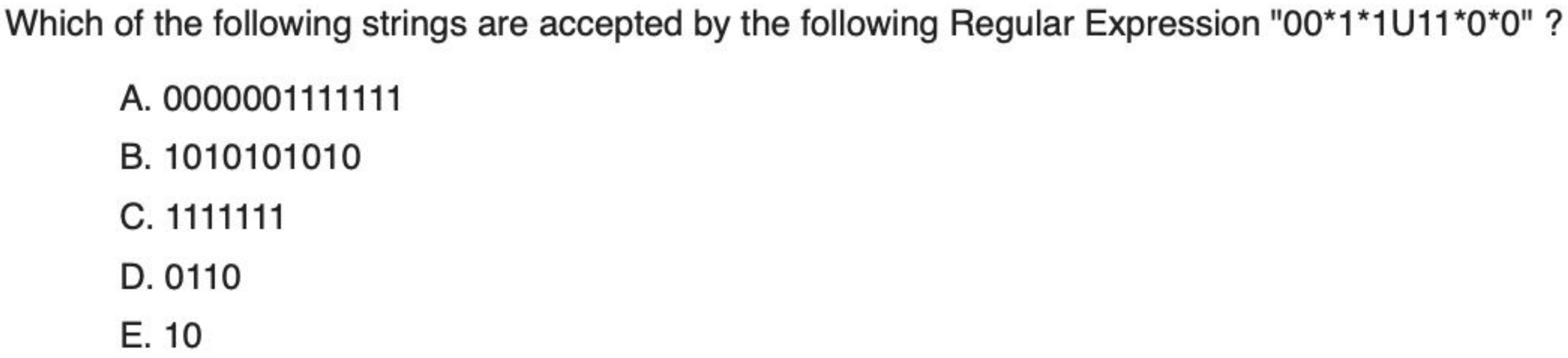
Note: Typical problems in the category will include: translate an FSA to a Regular Expression; simplify a Regular Expression; determine which Regular Expressions or FSAs are equivalent; and determine which strings are accepted by either an FSA or a Regular Expression.

Find a simplified Regular Expression for the following FSA:



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

Sample Problem 2



Which of the following strings are accepted by the following Regular Expression "00*1*1U11*0*0" ?

A. 0000001111111

B. 1010101010

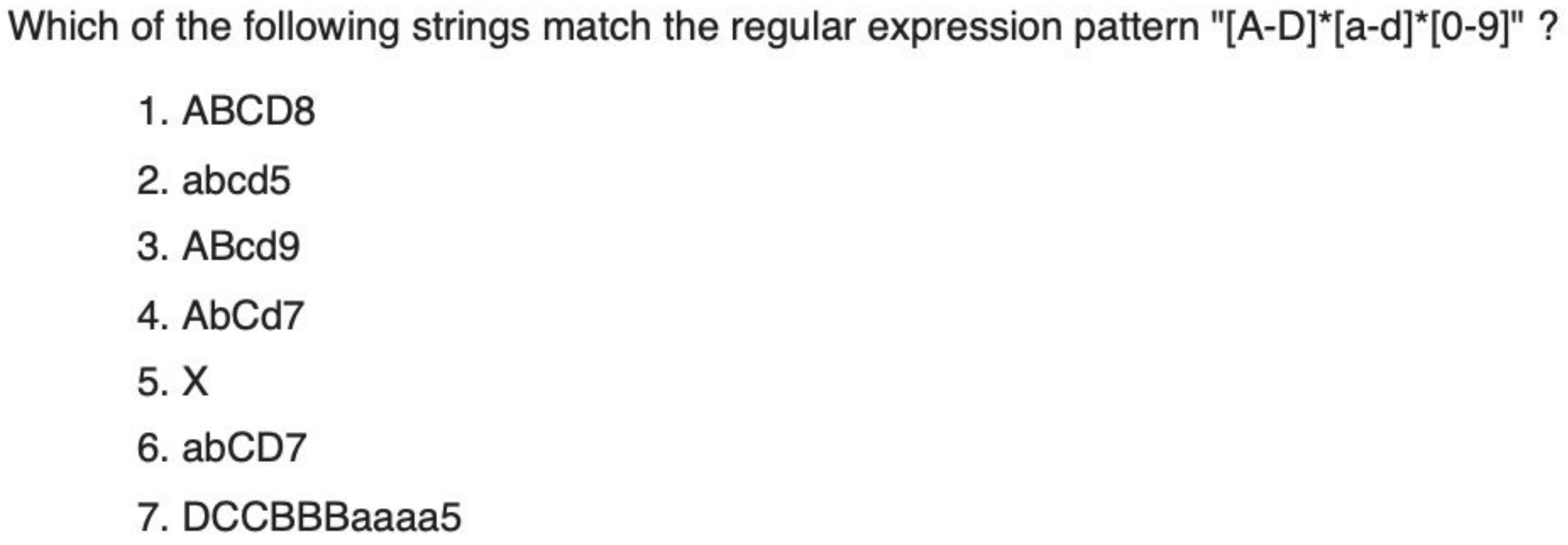
C. 1111111

D. 0110

E. 10

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.

Sample Problem 3


Which of the following strings match the regular expression pattern "[A-D]*[a-d]*[0-9]" ?

1. ABCD8
2. abcd5
3. ABcd9
4. AbCd7
5. X
6. abCD7
7. DCCBBBaaaa5

Solution: 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.

Sample Problem 4

Which of the following strings match the regular expression pattern "Hi?g+h+[^a-ceiou]" ?

1. Highb
2. HiiighS
3. HigghhhC
4. Hih
5. Hghe
6. Highd
7. HgggggghX

Solution: 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.

Past Contests: Senior

5. **Regular Expressions and FSA's**
Which of the following strings are accepted by the following regular expression? List all.

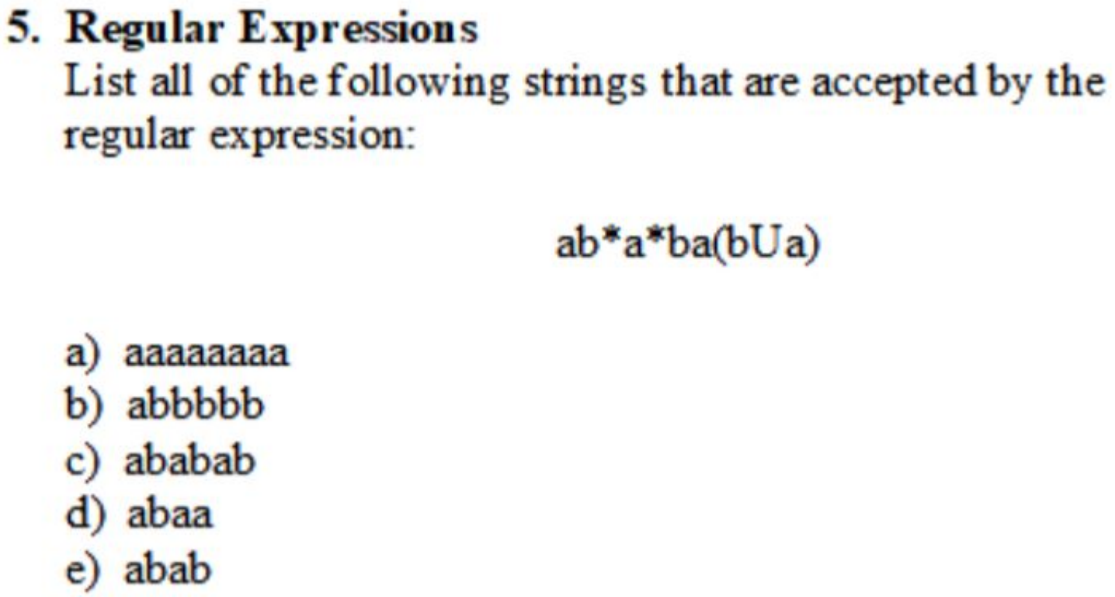
$$a\; b\; b^*(a \cup b^*)\; a\; (b^* \cup a^*)$$

a) *ababbaab*
b) *ababa*
c) *aaabb*
d) *abbbbab*
e) *abbaababbaa*

5. **Regular Expressions and FSA's**
a. fails at the last b.
b. fails after the second b.
c. fails after the first a.
e. fails at the fourth a.

5. d

Past Contests: Intermediate

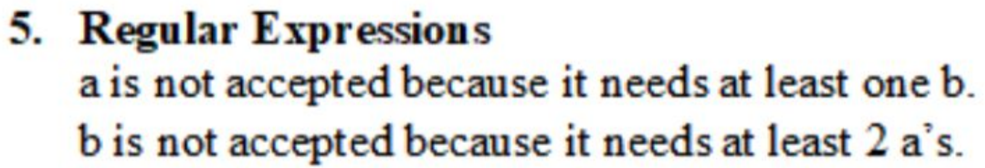


5. Regular Expressions

List all of the following strings that are accepted by the regular expression:

ab*a*ba(bUa)

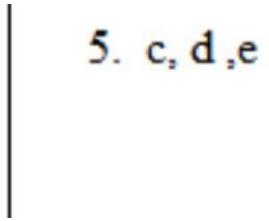
a) aaaaaaaa
b) abbbbb
c) ababab
d) abaa
e) abab



5. Regular Expressions

a is not accepted because it needs at least one b.

b is not accepted because it needs at least 2 a's.



5. c, d ,e