Regular Expression Examples

(Realize that many different RegExs can describe the same regular language.)

1. \((0 \cup 1) = \{ w \mid w \text{ is exactly 0 or 1} \}\) (literally only two strings are allowed: 0 and 1.)
2. \(0^*10^* = \{ w \mid w \text{ has exactly a single 1} \}\)
3. \(\sum^*1\sum^* = \{ w \mid w \text{ has at least one 1} \}\)
4. \(\sum^*101\sum^* = \{ w \mid w \text{ has the substring 101} \}\)
5. \((\sum\sum)^* = \{ w \mid w \text{ has an even length} \}\)
6. \(\sum(\sum\sum)^* = \{ w \mid w \text{ has odd length} \}\)
7. \(((\sum\sum\sum)^*) = \{ w \mid \text{the length of } w \text{ is a multiple of 3} \}\)
8. language of all strings ending with 0 = \(\sum^*0\)
9. language of all strings not ending with 11
   (So if the length is less than 2 we have no restriction, I choose to enumerate the possibilities.)
   \(= \varepsilon \cup 0 \cup 1 \cup \sum^*00 \cup \sum^*01 \cup \sum^*10\) (Enumerating all possibilities)
   \(= \varepsilon \cup \sum \cup \sum^*00 \cup \sum^*01 \cup \sum^*10\) (realized I could simplify)
   \(= \varepsilon \cup \sum \cup \sum^*0(00 \cup 01 \cup 10)\) (simplified even more)
10. language of all strings w such that w contains exactly one 1 and an even number of 0s.
    Let's jot a few down: 1, 001, 010, 100, 0001, 0010, 0100, 1000, and so on
    Now for the regular expression:
    \(= (00)^*1(00)^* \cup (00)^*010(00)^*\)
    (Think about it, for the count of 1s to be even, then viewing the 1 as splitting those 0s, we have
    then an even number of 0s both before and after OR we have an odd number of 1s both before
    and after (to maintain the total overall even count of 0s.))
11. Set of strings over \{0, 1, 2\} containing at least one 0 and at least one 1.
    (Again the problem is to identify categories of possibilities. We need a 0; we need a 1; Anything
    can be dispersed before and after those. We also need to make sure that at a minimum even if
    we have exactly one 0 and one 1, the order could be 01 or 10. Hence the two regular expressions
    below unioned together into one regular expression.
    \(= \sum^*0\sum^*1\sum^* \cup \sum^*1\sum^*0\sum^*\)
12. Back to our typical alphabet, \{0, 1\}, language of all strings containing at most one pair of
    consecutive 0s.
    I start by thinking about the minimal essence of what is required: the string may have no
    sequences of 0s or just a single 0, or exactly two 0s. I write the regular expression for just that
    part – ignoring the context in which this will be allowed.
    \((\varepsilon \cup 0 \cup 00) \leftarrow \text{just the kernel of the problem}\)
    Now I reason that 0 and 1 are allowed before and after this pattern, but with some restrictions so
    that I do not allow inadvertently a sequence of more than two 0s. Before this pattern, certainly
    any number of strictly 1s is allowed (that's 1*), but intermixed I could have a 0 as long as it is
    guaranteed to be followed by a 1 so that concatenating that with our kernel part will not result in
    a series of more than two 0s in sequence. Hence I have:
    \((1 \cup 01)^* (\varepsilon \cup 0 \cup 00)\)
    Using the same sort of reasoning I describe what is allowed after the kernel. And here is the
    final RegEx:
    \((1 \cup 01)^* (\varepsilon \cup 0 \cup 00) (1 \cup 10)^*\) (Admittedly this may not be the most elegant sol'n.)
13. language of all strings \( w \) such that \( w \) contains at most one 1 OR at least two 0s. (Implied: or both.)
   a) let's deal with the "at most one 1" \( \rightarrow 0^*10^* \cup 0^*(\varepsilon \cup 1)0^* \)
   b) now the "at least two 0s" \( \rightarrow \sum^*0\sum^*0\sum^* \)
   so since we are describe the language of all strings that belong in a or b, we use union:
   \( 0^*(\varepsilon \cup 1)0^* \cup \sum^*0\sum^*0\sum^* \)

14. Compare 12 to this problem. The language of all strings \( w \) such that \( w \) contains at most one 1 AND at least two 0s.
   Trying to use info of a & b from above will give a solution, albeit a convoluted line of reasoning:
   i) It is useful to find a & b as in problem 12, but in the end we seek the intersection of those regular expressions a & b. Not the union.
      Focusing on b) \( \sum^*0\sum^*0\sum^* \) we observe that at most only one of the three \( \sum^* \) is allowed to \( 0^*10^* \)
      while the other two are allowed to be \( 0^* \); otherwise we end up with too many 1s being allowed.
      (And all 0s is okay.)
      all \( 0s \rightarrow 000^* \)
      allowing the first \( \sum^* \) yields \( 0^*(0^*10^*)00^*0^* \rightarrow 0^*1000^* \)
      allowing the second \( \sum^* \) yields \( 0^*00^*(0^*10^*)00^* \rightarrow 00^*10^*0^* \)
      allowing the third \( \sum^* \) yields \( 0^*0^*(0^*10^*)00^* \rightarrow 0*0010^* \)
      Putting those three together we have now:
      \( 0^*(00^* \cup 001 \cup 010 \cup 100)0^* \)
   ii) Instead it may have been more productive just to think of this with a fresh mind. If at most one 1 is allowed and at least two 0s, then I can start enumerating the strings of this language taken in order of length:
      (None of zero length; none of length one)
      00, 001, 010, 100, ...
      and I realize others are the result of just interjecting more 0s within those string I have listed. So
      I have the kernel of the RegEx.
      \( (00 \cup 001 \cup 010 \cup 100) \)
      and since that may have any number of 0s before or after it:
      \( 0^*(00 \cup 001 \cup 010 \cup 100)0^* \)

15. For each of the examples given, how would you write the corresponding regular expression?
   1. all bit strings that begin with 0 and end with 1
   2. all bit strings whose number of 0's is a multiple of 5
   3. all bit strings with more 1's than 0's
   4. all bit strings with no consecutive 1's

**ANSWERS**

1. \( 0 (0 + 1)^* 1 \)
2. \( 1^* (0^* 0^* 1^* 0^* 1^* 0 1^* 0 1^*)^* \)
   *(don't forget that 0 is a multiple of 5!)*
3. trick question: you can't do this with a regular expression. How can you tell? the clue is that this language requires counting. Regular expressions can't 'count.'
This one is hard - You could start by writing something like this: \(0^* (1\ 0\ 0^*)^*\). That RE allows you to generate strings that start with any number of 0's. Then, notice that whenever you add a 1 to the string, it has to be followed by at least one 0. In this way, you won't be able to generate strings with consecutive 1's. However, this RE isn't correct. It doesn't allow us to generate strings that end in 1, and these strings should be part of the language described by the RE, because they fit the given definition. So, we can add \(0^* 1 (0\ 0^* 1)^*\) to our RE to create any strings that end in 1. In this part of the RE, as we generate strings, we always add at least one 0 before any 1. Again, there is no way to create strings with consecutive 1's. So, the answer is: \(0^* (1\ 0\ 0^*)^* + 0^* 1(0\ 0^* 1)^*\). Note that this is not the only correct answer.

16. language of \(w\) such that \(w\) does not equal 11 or 111

Think about what is allowed.

Enumerate some possibilities to focus on what is important to capture:

Are strings of length zero allowed? yes
Are strings of length one allowed? yes
Are strings of length two allowed? only those that are not 11
Are strings of length three allowed? only those that are not 111
Are strings of length greater than 3 allowed? yes.

So enumerating the first few of the strings in order of length:
\(\varepsilon, 0, 1, 00, 01, 10, 000, 001, 010, 011, 100, 101\), and all with length > three

Just line of thinking will yield a result, however, if you can view the problem in the way I am about to describe, it is even more cut and dry.

I claim there are two major categories of strings to handle.
1) those containing at least one 0, and
2) those that have no 0s.

Are all strings of category 1 allowed? Yes. And that RegEx is \(\sum^* 0 \sum^*\)
Are all string of category 2 allowed? No. Just those that do not have length 2 or 3. Hence that RegEx is \(\varepsilon \cup 1 \cup 11111^*\)

Therefore the overall expression is the following:
\(\sum^* 0 \sum^* \cup \varepsilon \cup 1 \cup 11111^*\)

17. \((10+0)^* (1+10)^*\)

\((10+0)^*\) will generate all strings that do not contain a pair of 1s, and \((1+10)^*\), the strings that do not contain a pair of 0s. So, the concatenation will generate all strings in which every occurrence of 00 precedes every occurrence of 11.

18. \(0^* (1+000^*)^* 0^*\)

We can generate a string which does not contain the occurrence 101 by making sure that any 0 in the middle (between two 1s) of the string must be paired with another 0.

19. \((e+0) (10)^* (e+0) (10)^* (e+1) (10)^* (e+1) + (e+0) (10)^* (e+1) (10)^* (e+0) (10)^* (e+0)\)

The both terms are just alternating 1's and 0s, eg \((e+0)(10)^*(e+1)\) where you are allowed to inert at most one extra 1 or 0 in between. We need two terms, depending on whether the double 1 or the double 0 comes first.

20. Find a regular expression corresponding to the language of all strings over the alphabet \{ a, b \} that contain exactly two a's.
Solution: A string in this language must have at least two a's. Since any string of b's can be placed in front of the first a, behind the second a and between the two a's, and since an arbitrary string of b's can be represented by the regular expression \( b^* \), \( b^* a b a b^* \) is a regular expression for this language.

21 Find a regular expression corresponding to the language of all strings over the alphabet \{ a, b \} that do not end with ab.

Solution: Any string in a language over \{ a, b \} must end in a or b. Hence if a string does not end with ab then it ends with a or if it ends with b the last b must be preceded by a symbol b. Since it can have any string in front of the last a or bb, \( (a + b)^* (a + bb) \) is a regular expression for the language.

22 Find a regular expression corresponding to the language of all strings over the alphabet \{ a, b \} that contain no more than one occurrence of the string aa.

Solution: If there is one substring aa in a string of the language, then that aa can be followed by any number of b. If an a comes after that aa, then that a must be preceded by b because otherwise there are two occurrences of aa. Hence any string that follows aa is represented by \( (b + ba)^* \). On the other hand if an a precedes the aa, then it must be followed by b. Hence a string preceding the aa can be represented by \( (b + ab)^* \). Hence if a string of the language contains aa then it corresponds to the regular expression \( (b + ab)^* (b + ba)^* \).

If there is no aa but at least one a exists in a string of the language, then applying the same argument as for aa to a, \( (b + ab)^* a(b + ba)^* \) is obtained as a regular expression corresponding to such strings.

If there may not be any a in a string of the language, then applying the same argument as for aa to \( A \), \( (b + ab)^* (b + ba)^* \) is obtained as a regular expression corresponding to such strings.

Altogether \( (b + ab)^* (A + a + aa) (b + ba)^* \) is a regular expression for the language.

23 Find a regular expression corresponding to the language of strings of even lengths over the alphabet of \{ a, b \}.

Solution: Since any string of even length can be expressed as the concatenation of strings of length 2 and since the strings of length 2 are aa, ab, ba, bb, a regular expression corresponding to the language is \( (aa + ab + ba + bb)^* \). Note that 0 is an even number. Hence the string \( \Lambda \) is in this language.

24 Describe as simply as possible in English the language corresponding to the regular expression \( a^* b^*(a^* b^* a^* b^* a^* \).

Solution: A string in the language can start and end with a or b, it has at least one b, and after the first b all the b's in the string appear in pairs. Any number of a's can appear any place in the string. Thus simply put, it is the set of strings over the alphabet \{ a, b \} that contain an odd number of b's.
25 Describe as simply as possible in English the language corresponding to the regular expression \((( a + b )^3)^* ( a + a + b )\).

**Solution:** \((( a + b )^3)^*\) represents the strings of length 3. Hence \((( a + b )^3)^*\) represents the strings of length a multiple of 3. Since \((( a + b )^3)^* ( a + b )\) represents the strings of length 3n + 1, where n is a natural number, the given regular expression represents the strings of length 3n and 3n + 1, where n is a natural number.

26: Describe as simply as possible in English the language corresponding to the regular expression \((b + ab)^* (a + ab)^*\).

**Solution:** \((b + ab)^*\) represents strings which do not contain any substring aa and which end in b, and \((a + ab)^*\) represents strings which do not contain any substring bb. Hence altogether it represents any string consisting of a substring with no aa followed by one b followed by a substring with no bb.

27. The set of all strings in which every pair of adjacent zeros appears before any pair of adjacent ones.

\((10+0)^* (1+10)^*\)

28. The set of all strings not containing 101 as a substring.

\(0^* (1^000^*)^* 0^*\)

29. The set of all strings with at most one pair of consecutive zeros and one pair of consecutive ones.

\((e+0)(10)^* (e+0)(10)(e+1)(10)^* (e+1) + (e+0)(10)^* (e+1)(10)(e+0)(10)^* (e+0)\)

The both terms are just alternating 1’s and 0s, eg \((e+0)(10)^* (e+1)\) where you are allowed to insert at most one extra 1 or 0 in between. We need two terms, depending on whether the double 1 or the double 0 comes first.

30. The language of all 0’s and 1’s having 1 at odd position

\(1((0+1))^*(1∧)\)