

Problem 1(a)

Find the Regex for the complement language of $L(aa^*bb^*)$

The essence of the regex is 1 or more a followed by 1 or more b , so the complement is a string that has zero a and/or zero b , or any instance of ba , because it violates the order.

So the regex is: $a^* + b^* + (a + b)^* ba (a + b)^*$

Problem 4

Show that if a language family is closed under union and complementation, it must also be closed under intersection.

If $L = L_1 \cap L_2$, then $L = \overline{\overline{L_1} \cup \overline{L_2}}$ which is $\overline{\overline{L_1} \cup \overline{L_2}}$, by DeMorgan's law. This is expressed using only union and complementation, which regular languages are closed over, so regular languages must also be closed under intersection because it can be expressed via other closed operations

Problem 10

Show that regular languages are closed under symmetric difference

The symmetric difference is defined by $(L_1 - (L_1 \cap L_2)) \cup (L_2 - (L_1 \cap L_2))$, creating a set of elements that are in only one of the two input sets.

Regular languages are closed under difference, as if $L = L_1 - L_2$, then $L = L_1 \cap \overline{L_2}$

We just showed that regular languages are closed under union, using the fact that they are closed under intersection and complement, so because we can represent the symmetric difference exclusively using operators under which regular languages are closed, symmetric difference is also closed

Problem 3

Find an algorithm to determine whether a given regular language L is a palindrome language.

The general idea is that the NFA that accepts a string in L should accept its reverse, too. We know that regular languages are effectively closed under reversal, so this is possible.

For an NFA M that defines L , construct another NFA M^R as follows:

1. Create a copy of M and reverse all edges, call this M^R
2. Add a new state I and a new state F to M^R

3. Add epsilon transitions from I to all accepting states in M^R
4. Add epsilon transitions from the initial state of M^R to F
5. Designate F as the only accepting state, and I as the only initial state

This generates an NFA M^R which accepts L^R . A language is palindromic if $L(M) = L(M \cap M^R)$. This is decidable, being an instance of the equivalence problem, so you can apply the algorithm to solve the equivalence problem and decide whether or not L is palindromic using this equivalence

Problem 6

Show that there exists an algorithm to determine whether L_1 is a proper subset of L_2 for any regular languages L_1 and L_2 .

L_1 is a proper subset of L_2 iff $L_1 \subseteq L_2$ and $L_1 \neq L_2$. $L_1 \subseteq L_2$ is decidable, as it is equivalent to $L_1 \cap \overline{L_2} = \emptyset$, which uses exclusively operations that regular languages are effectively closed under. $L_1 \neq L_2$ is also decidable, as equivalence is decidable and you can take the opposite of whatever the decision is to find non-equivalence.

Thus, there exists an algorithm to determine proper subset-ness, as there exists an algorithm to determine whether L_1 is a subset of L_2 and an algorithm to determine if they are equal.

Problem 15

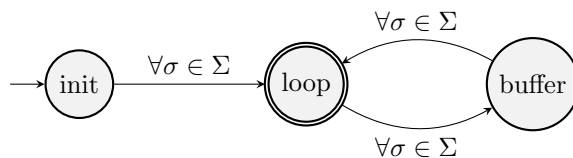
Find an algorithm to determine whether a regular language L contains a finite number of even-length strings.

Let $\Sigma = \{\sigma_1, \sigma_2, \dots\}$ be the alphabet over which L is constructed.

Let $M = (Q, \Sigma, \delta, q_o, F)$ be an NFA where $Q = \{\text{init}, \text{loop}, \text{buffer}\}$, $q_o = \text{init}$, $F = \text{loop}$, and δ is defined as follows:

$\delta(\text{init}, \sigma) = \text{loop}$ for all $\sigma \in \Sigma$, $\delta(\text{loop}, \sigma) = \text{buffer}$ for all $\sigma \in \Sigma$, and $\delta(\text{buffer}, \sigma) = \text{loop}$ for all $\sigma \in \Sigma$.

Why not include a diagram at this point:



Call this M - it accepts any odd-length string over Σ

Let $L_{\text{odd}} = L(M)$, a language which consists of all odd-length strings over Σ .

A language L consists of a finite number of even-length strings if $L - L_{\text{odd}}$ is finite, which is a decidable problem as mentioned above (accomplished via a depth-first search to identify cycles in an accepting DFA or NFA)

Problem 2

Show that the language $\mathbf{L} = \{a^n b^k c^n : n \geq 0, k \geq n\}$ is **not regular**.

Assume L is regular. For some value of p , strings of length $\geq p$ can be expressed in the form $w = xy^kz$, where y is nonempty, $k \geq 0$ and $w \in L$, by the pumping lemma.

Let $w = a^n b^n c^n$. w is in L , as k can be greater than **or equal** to n .

You can select substrings xyz in three ways:

1. y contains only a 's, or y contains only c 's
2. y contains only b 's
3. y contains both b 's and (a 's or c 's), or y contains all three.

In case 1, pumping y more than one time will upset the balance, as there will be an unequal number of a 's and c 's

In case 3, the order will be violated. If y contains two unique types of characters, then pumping y multiple times will result in a 's coming before b 's, or c 's coming before b 's, which violates the structure of the language

In case 2, the exact statement of the pumping lemma becomes important. We get to pump y any number of times, **including zero times**. So if we select y to contain exclusively b 's, then you could pump y zero times, resulting in the number of b 's being less than n . This violates the structure.

So no matter how you choose substrings to satisfy $w = xyz$ when $w = a^n b^n c^n$, you cannot pump y in a way that generates another string in L .

Because $w = a^n b^n c^n$ is a string in L , but cannot be pumped, L cannot be regular.

Problem 5(c)

Prove the irregularity of: $\mathbf{L} = \{a^n b^l a^k : n = l \text{ or } l \neq k\}$

We can represent L as $L_1 \cup L_2$ where:

$$L_1 = \{a^n b^n a^k\} \text{ and } L_2 = \{a^n b^l a^k : n \neq k\}$$

We know L_1 is not regular. $a^n b^n$ is a perfectly valid string in L_1 , yet it cannot be pumped. Any selection of a single type of character upsets balance, choosing a substring with both a 's and b 's upsets order.

So we are unioning a potentially regular language L_2 with a definitely irregular language L_1 . This will produce an irregular language. The hypothetical NFA that could accept $L_1 \cup L_2$ will not be able to accept strings from L_1 , but those strings are also in L .

Because an NFA cannot possibly accept $L_1 \cup L_2$, it must be irregular.

Another fun way to express this is to assert that irregular languages are closed under union, and L_1 is irregular.

I'll spare you the repetition of the pumping lemma which I just spit out last problem. The key is that given **any** string in L , we can pump it **zero or more** times to make another string in L .

So if you hand me a^n , for some sufficiently large n , the string is in L but I cannot pump any of its substrings.

The string is in L by setting $n = n$, $l = 0$ and $k = 0$, but when I want to pump the only thing I can select is some number of a 's.

No matter how many times I repeat a , it will always be true that $l = k$, because they are both 0.

Sure, you could pump the entirety of a^n 0 times, so the string is empty, which is a string in L , but you need to be able to pump it any amount of times you want, so any value other than 0 generates a string that is not in L .

Because there is no substring of a^n any number of times that can be pumped

I'll spare you the repetition of the pumping lemma which I just spit out last problem. The key is that given **any** string in L , we can pump it **zero or more** times to make another string in L .

So if you hand me $a^{v+1}b^va^{v+1}$, for some sufficiently large v , the string is in L but I cannot pump any of its substrings.

If my pumping substring contains both a 's and b 's, pumping it multiple times violates the order of the string, so my pumping string must exclusively consist of a 's or b 's.

If my substring comes exclusively from the first term, a^n , then pumping it more than zero times will result in $n > l$, which violates $n = l$.

If my substring comes exclusively from the middle term, b^l , then I could pump a single character for it to generate a string in L , but I need to be able to pump it an arbitrary amount of times. Pumping it any number of times more than 1 results in

I'll spare you the repetition of the pumping lemma which I just spit out last problem. The key is that given **any** string in L , we can pump it **zero or more** times to make another string in L .

So if you hand me $a^n b^l$, you have handed me a string in L where $k = 0$.

I cannot pump this string. I either need to achieve $n = l$ or $l \neq k$

If my goal is to pump n to equal l I can't do it. If my pumping string has both symbols in it the order will be violated, and if it doesn't I can pump one string $l + 1$ times and it's over. They can't possibly be equal.

If my goal is to pump so that $l \neq k$, that's a little easier. I just pump the a^n part freely, right?

Wrong. Take a look at the string $a^n b^l a^l$.

If my substring contains only symbols below

Big case 1: $n = l$

When $n = l$ you can just select any section of a^k to be your pumping string, which will generate more strings in L , which makes this language smell a little regular. However, k can be 0, and you can't pump a zero-length string.

But it gets deeper, if $k = 0$ and $n = l$ then l must be greater than 0 in order for the total string length to be greater than our pumping constant, so it must be the case that $l \neq k$, which forms...

Big case 2: $l \neq k$

When $l \neq k$, we must show that any substring can be pumped to force l to equal k .

There are two subcases to consider. First, you don't want to include any part of a^k in your pumping substring because that will obviously ruin your chances of making $l = k$, which is our strategy in this contradiction, so we can safely ignore that case.

So we arrive at.

Big case 2 subcase 1: your pumping string contains only symbols from a^n