Problem 1(a)

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

The essense 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, becuase 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 $L = \overline{\overline{L_1 \cap L_2}}$ 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 union 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 intersect and complement, so because we can represent the symmetric difference exclusively using operators unver 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 reglangs are effectively closed under reversal, so this is possible. This is hard to show algorithmically. You can't feed every string through a machine and see if its reverse is also accepted. Instead, we'll create a machine that accepts the reverse of a language, and see if those machines define the same language. Here goes!

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 $L(M) = L(M^R)$. 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.

The equivalence problem is decidable because $L(M) = L(M^R)$ is equivalent to showing that $L(M) \subseteq L(M^R)$ and $L(M^R) \subseteq L(M)$. That is decidable, as we will show in problem 6.

So overall, the algorithm is to generate M^R as directed above and compare it to M.

Problem 6

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

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 only uses operations that reglangs 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.

Equality being decidable is given by $L(M) = L(M^R) \iff L(M) \subseteq L(M^R)$ and $L(M^R) \subseteq L(M)$

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, both of which we have construted here.

Problem 15

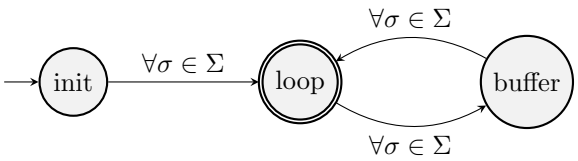
Find an algorithm to determine whether a regular language L contains a finite number of even-length strings.

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, accomplished via a depth-first search to identify cycles in an accepting DFA or NFA.

So because we can construct L_{odd} which is the subset of odd-length strings from L, and regular languages are closed under difference, we can perform the algorithm to decide whether or not $L - L_{\text{odd}}$ is finite, thus deciding whether or not L contains a finite number of even-length strings.

Problem 2

Show that the language $\mathbf{L = \left\{a^n b^k c^n \ : \ n \geq 0, k \geq n\right\}}$ **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 begin 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: $\mathbf{L = \left\{a^n b^l a^k \ : \ n = l \text{ or } l \neq k\right\}}$

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

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

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.