This is the problem presented in the Sat Feb 15 2025 review session for Midterm 1. The presentation has been heavily streamlined, so you may notice some differences from what was presented on the board. **In particular the name of the language has been changed**, and the presentation of the definition is different, but it is still the same language.

Problem

Fix some alphabet Σ. Given a word $w \in \Sigma$, we let $\mathrm{filter}(w)$ denote w with its odd-indexed characters removed. For example, $\mathrm{filter}(a) = \varepsilon$, $\mathrm{filter}(ab) = b$, $\mathrm{filter}(abba) = ba$, and $\mathrm{filter}(\varepsilon) = \varepsilon$. For a language $L \subseteq \Sigma^*$, define

$$\mathrm{filter}(L) := \{\mathrm{filter}(x) \mid x \in L\}.$$

If L is regular, show that $\mathrm{filter}(L)$ is also regular.

Initial approaches

This is just some motivation to come up with the correct solution. If you just want to review the solution, you can skip to "The full solution".

My advice for solving a DFA problem is always the following:

- First, find *any* algorithm that solves the problem, so you know what you want your DFA to do.
- Then, try to make your algorithm more "DFA-like" by adding specificity.

Also, if we know that L is regular, we should actually give the DFA solving L a name. So suppose that $M = (Q, \delta, q_0, F)$ solves L.[1]

Here is a basic algorithm to check if $y \in \mathrm{filter}(L)$:

- Find all the $x \in \Sigma^*$ such that $\mathrm{filter}(x) = y$.
- For all said x, check if $x \in L$ (by running $M(x)$). If any such x is in L, then accept.

To get some concrete ideas, we should actually look at what the x look like. For the purposes of clarity, we will examine just the case $y = abc$:

- We could have $x = a?\,b?\,c$, where $?$ represents any character in Σ.
- We could also have $x = a?\,b?\,c?$.

An easier problem

The main difficulty is that there are two cases to check. For example, if we want to check whether $abc \in \mathrm{filter}(L)$, we need to check all the strings x of the form

- $a?\,b?\,c$
- $a?\,b?\,c?$

which makes the problem inherently more complicated. So we will first solve a simpler problem: if L is regular **and every string in L has even length**, then show that $\mathrm{filter}(L)$ is regular.

Here we only need to check the second case, because the first case consists of strings of odd length. This will allow the rest of our discussion to be a little simpler.

Here is the key observation: we can generate all of the possible strings x as we read in the characters of y.

[1] I am omitting Σ because we already know what the alphabet is: we fixed Σ earlier! And the DFA solving $\mathrm{filter}(L)$ will have the same Σ, so there really is no need to specify the language.

```
def generate_x(y):
  S = {ε} # This is the set of all possible x
  for σ ∈ y:
    S := {x + σ + char | x ∈ S, char ∈ Σ}
```

So when we encounter each $\sigma \in y$, for every char $\in \Sigma$, we create a new thread $x + \sigma + \text{char}$. And now we are very close to the solution, at least for this easier problem!

The point of the powerset

But before we try and formalize this idea, there is one last hiccup to take care of: the number of threads balloons, which means that a priori there is no reason to believe the number of states would be finite (which we need for a DFA).

But there is one important observation to make *for this problem*:[2] we want to check if *any* of the threads accepts, which means

1. We only care about the *states* of the threads,
2. and in particular, we do not care if there is more than one thread at the same state.

Armed with this knowledge, the intuition is as follows: whenever we see two threads at the same state, we should merge them. And what mathematical object automatically merges duplicates? Sets!

The solution to the easier problem

Now we actually formalize our ideas and write out a full solution *to the easier problem.*

Define a DFA $M' = (Q', \delta^*, {q_0}', F')$ where

- $Q' := \mathcal{P}(Q)$
- $\delta'(S, \sigma) = \{\delta(q, \sigma c) \mid q \in S, c \in \Sigma\}$
- $q_0' = \{q_0\}$
- $F' = \{S \mid S \cap F \neq \emptyset\}$.

What is this DFA really doing? Let's explain it step by step:

- Q': The state S is a subset Q, and it is the collection of threads that we are running. Each thread q is an individual state inside of Q.
- δ': Whenever we encounter a character σ in y, *for every current state* $q \in S$, we would like to create new threads by feeding σc into the DFA at state q for every $c \in \Sigma$. The collection of these threads is just $\{\delta(q, \sigma c) \mid q \in S, c \in \Sigma\}$, exactly as we defined $\delta'(S, \sigma)$.
- ${q_0}'$: Our initial thread is just q_0, since we initially start with $y = \varepsilon$ and the only **even length** string x such that $\text{filter}(x) = \varepsilon$ is $x = \varepsilon$.
- F': A set of threads S accepts precisely when it contains at least one accepting state. And this occurs precisely when $S \cap F \neq \emptyset$.

How to upgrade

To solve the full problem, where we do not assume that every $x \in L$ has even length, we have to handle the odd-length case.

Whenever we read in a character σ, just as we compute all the possible threads after appending σc (for every $c \in \Sigma$), we can compute all the possible threads after just appending σ. So suppose the threads we compute after just appending σ form a set T. Then at any step, $S \cup T$ consists of all the possible states that we could reach with *all* of the x.

[2]Meaning it does not always hold for *every* threading problem!

The main mental barrier to defining this T is the following: not knowing how to use T for future updates, or fear that storing T will somehow interfere with future updates. After all, our old solution worked because we were solely using S for updates, and as long as we preserve S, hopefully you believe that T is easily computable. But very roughly, the strategy is just to ignore T in δ. To do this, we need to store T *separately* from S.

All this will hopefully motivate the following.

The full solution

Suppose L is solved by (Q, δ, q_0, F). Then define a DFA (Q', δ', q_0', F') where

- $Q' := \mathcal{P}(Q)$
- $\delta'((S, T), \sigma) = (\{\delta(q, \sigma c) \mid q \in S, c \in \Sigma\}, \{\delta(q, \sigma) \mid q \in S\})$
- $q_0' := (\{q_0\}, \{\delta(q_0, \sigma) \mid \sigma \in \Sigma\})$
- $F' = \{(S, T) \mid (S \cup T) \cap F \neq \emptyset\}$.

Then (Q', δ', q_0', F') solves $\text{filter}(L)$.

What the states represent

The goal of this section is to clarify why the δ function is defined as it is and especially why q_0' — particularly the second component — is defined this way. It will also become the bedrock for the proof of correctness.

If the state after reading y is (S, T), S is the set of states achieved by the even-length x where $\text{filter}(x) = y$ and T is the set of states achieved by the odd-length x where $\text{filter}(x) = y$. In symbols,

$$S = \{\delta(q_0, x) \mid \text{filter}(x) = y, |x| \text{ is even}\}$$
$$T = \{\delta(q_0, x) \mid \text{filter}(x) = y, |x| \text{ is odd}\}.$$

Outline of proof of correctness

- The proof of correctness of this DFA essentially reduces to proving the claim above via induction on the length of y.
- Then we may conclude that
$$S \cup T = \{\delta(q_0, x) \mid \text{filter}(x) = y\}.$$
- To conclude, note that $(S \cup T) \cap F \neq \emptyset$ precisely when there is some $x \in \Sigma^*$ such that $\delta(q_0, x) \in F$ — which implies $x \in L$ — and $\text{filter}(x) = y$.