1. Nonregular Languages Review

a. What problems do nonregular languages correspond to?

> Nonregular languages represent problems that cannot be represented with an NFA, DFA, or regex, that is, cannot be solved with a computer with a finite amount of memory.

b. Intuitively, why is $E = \{a^n b^n \mid n \in \mathbb{N}\}$ **not regular**? Meanwhile, intuitively, why is the language $L = \{a^n b^n \mid n \in \mathbb{N} \text{ and } n \leq 103\}$ **regular**?

> E is not regular because we need to keep track of a number that can grow arbitrarily large (the difference between the number of a's and b's we've seen), so it cannot be solved with finite memory.
>
> Meanwhile, L is regular because the number we need to keep track of is limited. It can be no more than 103.
>
> Also, while regular languages can be infinite, all finite languages are regular, and L is a finite language.

c. For some language L over Σ and strings x and y, the formal definition of the statement "x and y are distinguishable relative to L", denoted by $x \not\equiv_L y$, is $\exists w \in \Sigma^*. \ (xw \in L \leftrightarrow yw \notin L)$. Explain this definition in plain English.

> This means "there's a string w you can add onto the end of x and y so that exactly one of the resulting strings (xw or yw, but not both) will be in L."

d. Explain the definition of a distinguishing set for L: $\forall x \in S. \ \forall y \in S. \ (x \neq y \to x \not\equiv_L y)$

Given an arbitrary language, what is the smallest distinguishing set for it?

> A distinguishing set S for a language L is a set where any two distinct elements from the set are distinguishable relative to the language.
>
> The smallest distinguishing set for any language is the empty set. The definition of distinguishing set is vacuously true.

e. For the language $L = \{a^n b^n \mid n \in \mathbb{N}\}$, give an example of two strings x and y where $x \not\equiv_L y$ is **true**. Give an example of two strings x and y where $x \not\equiv_L y$ is **false**.

$x \not\equiv_L y$ for $x = a$ and $y = aa$. Adding b to the end of both strings results in $ab \in L$ and $aab \notin L$.

$x \equiv_L y$ for $x = aba$ and $y = b$. Adding any string to the end of both strings results in two strings that are definitely not in the language since they break the pattern.

2. Proving Languages are Not Regular

The Myhill-Nerode theorem says the following:

> Let L be a language over Σ. If there is a set $S \in \Sigma^*$ such that
> - S contains infinitely many strings, and
> - every pair of distinct strings $x, y \in S$ are distinguishable relative to L, that is, $x \not\equiv_L y$,
>
> then L is not a regular language.

a. Explain intuitively why S has to be an infinite set for this theorem to work.

> The proof of the Myhill-Nerode theorem works by arguing that no matter how many states we have in a DFA for a language L, we can always find a larger number of pairwise distinguishable strings. If we have infinitely many strings in S, we can always ensure that we have more strings in S than there are states in any proposed DFA for L. On the other hand, if S is finite, this line of reasoning only works on DFAs that have fewer than $|S|$ states.

b. Does S have to be a subset of L? Why or why not?

> Nope, not at all! We just need S to be a subset of Σ^*. This is really important: when you're trying to show that a language isn't regular, you don't need to limit your search for distinguishable strings purely to strings in L. You can use any strings you'd like.

c. Give an example of a distinguishing set for the language $L = \{a^n b^n \mid n \in \mathbb{N}\}$.

> The example we discussed in lecture was $\{a^n \mid n \in \mathbb{N}\}$. This is the set $\{\varepsilon, a, aa, aaa, ...\}$

d. Let's practice using the theorem. Let $\Sigma = \{a, b\}$ and let $L = \{b^n a^m \mid n, m \in \mathbb{N} \text{ and } n \neq m\}$.

(1) Explain why L is not the complement of the language $\{a^n b^n \mid n \in \mathbb{N}\}$.

> The complement of $\{a^n b^n \mid n \in \mathbb{N}\}$ contains strings like aab or $abba$ that don't consist of a string of b's followed by a string of a's, but these strings aren't in L.

(2) Give an intuitive justification for why L isn't regular – what would we need to "remember" that would not fit in a finite amount of memory?

Similarly to the language $\{a^n b^n \mid n \in \mathbb{N}\}$, we need to keep track of how many b's we've seen before seeing our first a, so that we can ensure that the number of a's is not the same. This number could be arbitrarily large.

(3) Use the Myhill-Nerode theorem to prove that L isn't regular. You'll need to find an infinite set of strings that are pairwise distinguishable relative to L. Finding this set is the difficult part of any nonregular language proof. Think of some category of strings that would have to be treated differently by any DFA for L, then see what happens if you gather all of them together into a set.

Proof: Let $\mathrm{S} = \{b^n \mid n \in \mathbb{N}\}$. We will prove that S is infinite and is a distinguishing set for L.

To see that S is infinite, note that it contains one string per natural number.

To see that any pair of strings in S are distinguishable relative to L, pick any two strings $b^n, b^m \in S$ where $n \neq m$. Then, note that $b^n a^n \notin L$ but $b^m a^n \in L$. We see that $a^n \not\equiv_L a^m$, as required.

Because S is an infinite distinguishing set for L, by the Myhill-Nerode theorem, L is not regular. ■

3. Writing Regular Expressions

Here are some tips for writing regular expressions:

- Think about ways to simplify the problem. Is there a choice between multiple options, which you could represent with $\cup$? Is there some way to split strings in this language into multiple parts or sections, which you could concatenate?
- Try writing out example strings in the language. A regex can only generate arbitrarily long strings using the * operator. Look out for a repeating pattern that you can star.

To practice with regular expressions, write a regular expression for each of these languages.

a. Let $\Sigma = \{a, b, c\}$ and let $L = \{w \in \Sigma^* \mid w \text{ ends in } cab\}$.

> One option is
> $$\Sigma^* cab$$
> This matches any string that begins with some number of characters of any type, then ends with cab.

b. Let $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same}\}$.

> Here's an option:
> $$a \cup b \cup a\Sigma^* a \cup b\Sigma^* b$$
> This says "match a, or b, or something that starts and ends in a with any number of characters in the middle, or something that starts and ends in b with any number of characters in the middle."
>
> We can condense this using the ? operator:
> $$a(\Sigma^* a)? \cup b(\Sigma^* b)?$$

c. Let $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid \text{some substring of } w \text{ consists of two } b\text{s separated by five characters }\}$.

> Here's a way to do this:
> $$\Sigma^* b\Sigma^5 b\Sigma^*$$
> This says "match any number of characters, then b, then 5 characters, then b, then any number of characters."

d. Let $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid w \text{ does not contain two consecutive } a\text{s or } b\text{s}\}$. (Hint: Write out some strings in this language. What do you notice?)

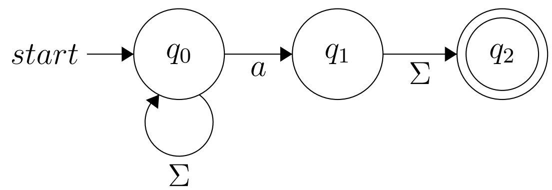
Here's one option:

$$(ba)^*b? \cup (ab)^*a?$$

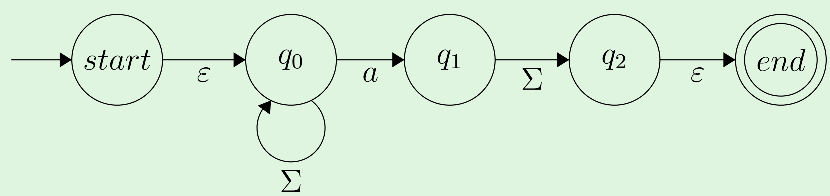
The definition means that we'll need to alternate between a's and b's.

4. The State Elimination Algorithm

Let's practice the state elimination algorithm, which converts an NFA into a regular expression. Consider this NFA:



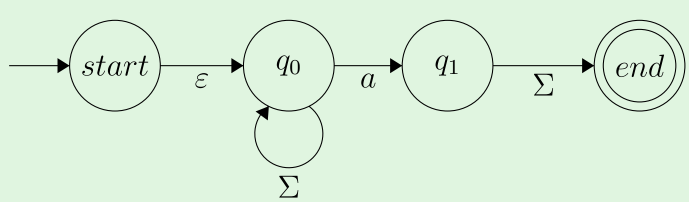
a. Prepare the NFA for the state elimination algorithm by adding two new states, q_{start} and q_{end}, adding an ε transition from q_{start} to the old start state, adding an ε transition from all of the accept states to q_{end}, marking all of the accept states as no longer-accepting, and marking the new end state as accepting.



To eliminate a state q, identify all pairs of states q_{in} and q_{out} where there's a transition from q_{in} to q and from q to q_{out}, then add shortcut edges from q_{in} to q_{out} to bypass state q. Remember that q_{in} and q_{out} may be the same state.

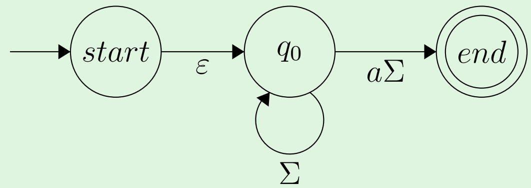
b. Eliminate state q_2 from the NFA.

We only have to add a transition between one pair of qualifying states in this case, from q_1 to *end*. The transitions we'd have to take to eliminate these states are Σ and then ε. The regular expression we should add onto that transition is $\Sigma\varepsilon$, which is the same thing as Σ.



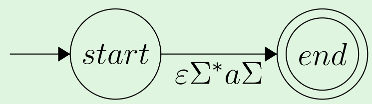
c. Eliminate state q_1 from the NFA.

The only pair of states we need to add a transition between are q_0 and *end*. We combine the two regular expressions to see how to label the new transition, giving us $a\Sigma$. Intuitively, we can get from q_0 to *end* by reading one a and then one of any character.



d. Eliminate state q_0 from the NFA. What is the final regex?

Here, there is only one pair of states we need to add a transition between, *start* and *end*; but because there is a transition on q_0 to itself, we have to introduce the Kleene star into the regular expression.



Now that we have only our two states left, we're left with the regular expression $\varepsilon\Sigma^* a\Sigma$, which can be simplified to $\Sigma^* a\Sigma$.