Solutions for Week Eight

Concept Checks

i. What is a language? (I.e., what "type" does it have—is it a string? A function?)

A language is a (possibly infinite) set of strings. For a given alphabet Σ, a language is a subset of Σ*.

ii. Name at least two differences between DFAs and NFAs.

NFAs are non-deterministic, meaning they can have multiple transitions for the same (state, symbol) pair. They can also have zero transitions defined for a given (state, symbol) pair. Finally, they can transition on ε (the empty string) as well as the symbols of Σ.

You could also observe that while a DFA accepts a string if it ends in an accepting state, an NFA accepts a string if *any* computation ends in an accepting state.

iii. Give three different definitions for what a regular language is.

A regular language is a language recognized by a DFA; a language recognized by an NFA; or a language for which there is a regular expression.

iv. Name at least two closure properties of regular languages.

The regular languages are closed under complementation, union, intersection, concatenation, and Kleene star.

Designing Regular Expressions

Below are a list of alphabets Σ and languages over those alphabets. For each language, write a regular expression for that language.

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

One option is

$$\Sigma^* cab$$

This matches any string that begins with some number of characters of any type, then ends with *cab*. Isn't that easier than the NFA?

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

Here's one option:

$$a \cup b \cup a\Sigma^* a \cup b\Sigma^* b$$

This says "either match the single characters *a* and *b*, or match something that starts and ends with *a* with any number of characters in the middle, or match something that starts and ends with *b* with any number of characters in the middle."

You can condense this a bit using the *?* operator:

$$a(\Sigma^* a)? \cup b(\Sigma^* b)?$$

This means "if you see an *a*, either you're done, or you're going to see some number of characters and then another *a*" (and the same for *b*.)

iii. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ contains two *b*'s separated by exactly five characters $\}$. Write a regular expression for L.

Here's one option:

$$\Sigma^* b \Sigma^5 b \Sigma^*$$

This means "match any number of characters, then read a *b*, five characters, and another *b*, then as much as you'd like after that."

iv. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

One possibility is

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

If the characters alternate, they either start with an a or they start with a b. If they begin with a, then we'll then have some number of copies of the string ba, optionally ending with a final b. If they begin with a b, then we'll then have some number of copies of the string ab, optionally ending with a final a.

v. Let $\Sigma = \{a, b, c\}$ and let $L = \{ w \in \Sigma^* \mid w$ contains every character in Σ exactly once $\}$. Write a regular expression for L.

Alas, there's no easy way to write this one because those characters can appear in any order.

Here's one option:

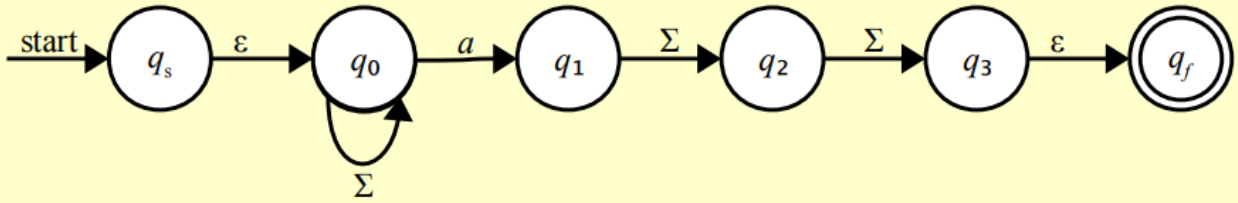
$$abc \cup acb \cup bac \cup bca \cup cab \cup cba$$

Why we asked this question: This question was designed to show off some common patterns and techniques in the design of regular expressions. Part (i) shows off how regexes match patterns at the beginning or end of a string (namely, by inserting Σ^* to cover prefixes or suffixes.) Part (ii) explores how regular expressions keep track of information like which character was used; namely, by repeating different patterns in different combinations. Part (iii) was designed to get you to see how a language looks when expressed in different formats, and this problem shows that regexes nicely capture this particular language. Part (iv) shows off an edge case whereby you sometimes have to insert a suffix to a regex to capture the idea that a pattern might end abruptly. Finally, part (v) of this problem was there to show that regexes do have some weaknesses, namely the inability to remember what they've seen without just exhaustively listing cases.

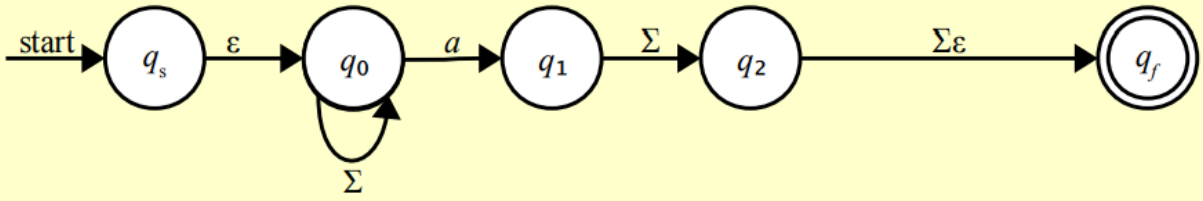
State Elimination

Using the state-elimination algorithm, convert this NFA into a regular expression. (You could just directly design a regular expression for this language, but we want you to specifically use the state elimination algorithm).

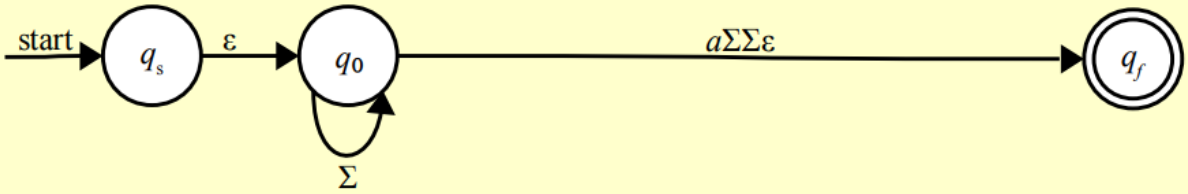
We begin by introducing a new start and accept state with the appropriate new ε-transitions:



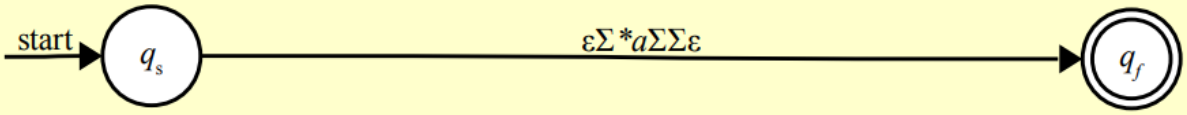
Let's eliminate state q_3 first. The only state with a transition into q_3 is q_2, and the only state q_2 has a transition into is q_f. Therefore, we'll add a direct link from q_3 to q_f labeled with the regular expression Σε (Σ for the entry into q_3 and ε for the exit from q_3 to q_f). This yields this setup:



Eliminating states q_2 and q_1, in that order, gives this configuration:



Now, we eliminate q_0. We can enter q_0 from q_s, stay around using the Σ self-loop, then leave by following the aΣΣε transition. This gives the following:



The final regex is εΣ*aΣΣε, which easily simplifies to Σ*aΣΣ. This makes sense – it means that the regex matches any number of characters, then something that ends with a followed by any two characters.

Why we asked this question: The beauty of the automata transformations we've seen so far is that it lets us study the same class of objects – the regular languages – using one of three totally different lenses. The constructions are a bit tricky, though, and to help you get a handle for how they work, we wanted you to practice working through them on a few special cases. This particular case, in our opinion, wasn't too tough of a sample input.

Closure Properties

Prove that the regular languages are closed under subtraction. That is, prove that if L_1 and L_2 are regular languages over some alphabet Σ, then the language $L_1 - L_2$ is also regular.

Proof: Let L_1 and L_2 be regular languages over Σ. Notice that the set $L_1 - L_2$ can equivalently be expressed as $\{ w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2 \}$. Since in this set we assume that $w \in \Sigma^*$, the condition that $w \notin L_2$ is equivalent to the condition that $w \in \overline{L_2}$. Consequently, we see that

$$L_1 - L_2 = \{ w \in \Sigma^* \mid w \in L_1 \wedge w \in \overline{L_2} \}.$$

Notice that this is in turn equivalent to the set $L_1 \cap \overline{L_2}$. The regular languages are closed under intersection and complementation, so this language is regular. Therefore, $L_1 - L_2$ is regular, as required. ■

Why we asked this question: Our exploration of the closure properties of the regular languages led us to discover the regular expressions, a totally new way of thinking about the regular languages that's completely divorced from the underlying machine model. This exercise was designed to get you thinking about these closure properties and how you might come up with your own.

The Myhill-Nerode Theorem

i. What is the formal definition of the statement $x \not\equiv_L y$? Explain it in plain English. Give an example of two strings x and y along with a language L where $x \not\equiv_L y$ holds.

The formal definition of $x \not\equiv_L y$ is

$$x \not\equiv_L y \quad \text{if} \quad \exists w \in \Sigma^*. (xw \in L \leftrightarrow yw \notin L).$$

In plain English, this means there's a string you can tack onto the end of both x and y such that exactly one of xw and yw will be in L.

As an example, the strings a and aa are distinguishable relative to $\{a^n b^n \mid n \in \mathbb{N}\}$; tacking on bb makes $aabb \in$ L and $abb \notin$ L.

ii. The proof hinges on the fact that if $x \not\equiv_L y$, then x and y cannot end in the same state when run through any DFA for a language L. We sketched a proof of this in class. Explain intuitively why this is the case.

Let x and y be strings where $x \not\equiv_L y$, and let w be a string that distinguishes x and y. If x and y end up in the same state in a DFA D for the language L, then the strings xw and yw will end up in the same state in D because once x and y have "collided," reading the same characters after reading x and y will cause the DFA to follow the same series of transitions. If the state that xw and yw end up in is accepting, then both xw and yw are accepted even though one of those strings isn't in L, and if that state is rejected the both xw and yw are rejected even though one of those strings *is* in L. Either way, we contradict the fact that D is a DFA for the language L.

iii. Explain, intuitively, why S has to be an infinite set for this proof to work.

The proof of the Myhill-Nerode theorem works by arguing that no matter how large of a DFA we build 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.

iv. Does anything in the proof require that S be a subset of L?

Nope! Not at all. We just need S to be a subset of Σ^*. This is really important, actually – 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.

Why we asked this question: This question was designed to get you engaged with the proof of the Myhill-Nerode theorem. The proof of the theorem is a bit tricky and nuanced, and the statement of the proof is quite complex. We hoped that these problems would make it easier for you to see what the theorem requires, what it doesn't require, and why it works.

Nonregular Languages Warmup

Let $\Sigma = \{1, \geq\}$ and consider the language $L = \{1^m \geq 1^n \mid m, n \in \mathbb{N} \text{ and } m \geq n\}$.

i. Give some specific examples of strings from the language L.

Here are a few strings from L: $1 \geq 1$, $1111 \geq 11$, $111 \geq$, $11111 \geq 11111$.

ii. Without using the Myhill-Nerode theorem, give an intuitive justification for why L isn't regular.

Intuitively, any DFA for this language will at some level need to keep track of how many 1's appear on the left-hand side of the $\geq$ sign. If the DFA can't do that, then when it sees the 1's appearing on the right-hand side of the $\geq$ sign, it won't be able to tell whether it saw more 1's the first time than the second time. However, the DFA only has finitely many states, so there's no way for it to exactly remember the number of 1's that it's seen.

iii. 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. As a hint, see if you can think of some 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 $S = \{1^n \mid n \in \mathbb{N}\}$. Notice that S is infinite because it contains one string per natural number and there are infinitely many natural numbers. We also claim that any two distinct strings from S are distinguishable relative to L. To see this, consider any two strings 1^m, $1^n \in S$ with $m \neq n$. Assume without loss of generality that $m > n$. Then $1^m \geq 1^m \in L$, but $1^n \geq 1^m \notin$ L. Therefore, we see that $1^m \not\equiv_L 1^n$. Consequently, by the Myhill-Nerode theorem, we know that L is not regular. ■

Why we asked this question: This question was designed to walk you through the thought process for showing that a particular language is nonregular. Part (i) asked you to take the (concise) definition of the language L and get a feel for what the strings in L actually look like. From there, we hoped you'd get a more concrete sense of what this language is. Part (ii) asked you to build an intuition for why the language isn't regular, which is a good idea because it forces you to think about what a DFA for the language would have to look like and what it would have to do. In part (iii), we asked you to formalize your reasoning so that you could say with certainty that the language is definitely not regular. As you work through more proofs of nonregularity going forward, we hope that you use some of the techniques from this problem to help guide your efforts.

Nonregular Languages

Here are some more problems to help you get used to proving that certain languages aren't regular.

i. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^m \mid n, m \in \mathbb{N} \text{ and } n \neq m \}$. Explain why this language is not the complement of the language $\{ a^n b^n \mid n \in \mathbb{N} \}$.

This language is not the complement of the language $\{ a^n b^n \mid n \in \mathbb{N} \}$ because the complement of $\{ a^n b^n \mid n \in \mathbb{N} \}$ contains strings like *ba* or *abba* that don't consist of a string of *a*'s followed by a string of *b*'s. However, the new language $\{ a^n b^m \mid n, m \in \mathbb{N} \text{ and } n \neq m \}$ doesn't contain strings like these, so it's not the complement of $\{ a^n b^n \mid n \in \mathbb{N} \}$.

ii. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^m \mid n, m \in \mathbb{N} \text{ and } n \neq m \}$. Prove that L is not regular.

Proof: Let $S = \{ a^n \mid n \in \mathbb{N} \}$. This set is infinite, since it contains one string per natural number. Moreover, we claim that all strings in S are distinguishable relative to L. To see this, consider any strings $a^n, a^m \in S$ where $n \neq m$. Then $a^n b^n \notin L$ but $a^m b^n \in L$. Thus a^n and a^m are distinguishable relative to L, and since they were chosen arbitrarily we conclude that any pair of strings in S are distinguishable relative to L. Therefore, by the Myhill-Nerode theorem, L is not regular. ■

iii. Let $\Sigma = \{a\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is a palindrome} \}$. Prove that L *is* regular.

Proof: All strings made only of a's are palindromes, since they're the same forwards and backwards. Thus $L = \Sigma^*$, which is a regular language. ■

iv. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is a palindrome} \}$. Prove that L is not regular.

Proof: Let $S = \{ a^n \mid n \in \mathbb{N} \}$. This set is infinite, since it contains one string per natural number. Moreover, we claim that all strings in S are distinguishable relative to L. To see this, consider any strings $a^n, a^m \in S$ where $n \neq m$. Then $a^n b a^n \in L$ but $a^m b a^n \notin L$. Thus a^n and a^m are distinguishable relative to L, and since they were chosen arbitrarily we conclude that any pair of strings in S are distinguishable relative to L. Therefore, by the Myhill-Nerode theorem, L is not regular. ■

Why we asked this question: Part (i) of this question was designed to highlight a nuance of complements of languages that hasn't come up in any other contexts yet. Part (ii) was then designed as a (hopefully) straightforward application of the Myhill-Nerode theorem. Part (iv) was designed as a more complex Myhill-Nerode argument because you need to make sure that you can actually distinguish the strings in the middle of the proof. As you saw in part (iii), the language is regular if you just have one character in your alphabet, so the proof in part (iv) would have to use the character *b* at least once.

Context-Free Grammars

i. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ has no a's or has no b's $\}$. Write a CFG for L.

One option is

$$S \to A \mid B$$
$$A \to aA \mid \varepsilon$$
$$B \to bB \mid \varepsilon$$

Any string in Σ^* that has no a's or no b's (for this choice of Σ) must consist purely of a's or purely of b's. This grammar works by having dedicated nonterminals for each case, then choosing which case to work with.

ii. Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ has at least one a and at least one b $\}$. Write a CFG for L.

One option is

$$S \to XaXbX \mid XbXaX$$
$$X \to aX \mid bX \mid \varepsilon$$

Here, X generates any possible string. This grammar works by, at the top level, placing down an a and a b and filling the gaps with arbitrary strings.

Another option is shown here:

$$S \to aX \mid bY$$
$$X \to aX \mid bZ$$
$$Y \to aZ \mid bY$$
$$Z \to aZ \mid bZ \mid \varepsilon$$

Here, each nonterminal stands for a particular combination of letters that have appeared so far: S means "nothing has been written yet," X means "only a's have been written," Y means "only b's have been written," and Z means "both a and b have been written." Each production rule then writes a single character and switches which nonterminal is written down. Only when we get to nonterminal Z – when both a and b have been written – are we allowed to expand the nonterminal out into ε and stop adding characters.

iii. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b a^n \mid n \in \mathbb{N} \}$. Write a CFG for L.

This one is all about finding a build ordering. We build both groups of a's simultaneously, working from the outside inward. Here's one grammar that does this:

$$S \to aSa \mid b$$

Notice the similarity between this grammar and the grammar for $\{a^n b^n \mid n \in \mathbb{N}\}$.

iv. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^{2n} \mid n \in \mathbb{N} \}$. Write a CFG for L.

This grammar generalizes the grammar for $\{a^n b^n \mid n \in \mathbb{N}\}$ and works by building the a's and b's at the same time, working from the outside inward:

$$S \to aSbb \mid \varepsilon$$

You might have noticed that the trick of building from the outside inward is pretty common in context-free grammars.

v. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^m \mid n, m \in \mathbb{N} \text{ and } n \leq m \leq 5n \}$. Write a CFG for L.

There are many ways we can do this. One option is shown below. It works by using the general pattern from before, except that whenever we place down an a, we place down between one and five b's on the other side:

$$S \to aSb \mid aSbb \mid aSbbb \mid aSbbbb \mid aSbbbbb \mid \varepsilon$$

This works, but is a bit lengthy. Here's another option that works by saying that whenever we put an a down on one side of the string, we have to put a b down on the other, followed by four optional b's:

$$S \to aSbXXXX \mid \varepsilon$$

$$X \to b \mid \varepsilon$$

Here, the nonterminal X means "either b or nothing at all."

vi. Let $\Sigma = \{a, b\}$ and let $L = \{ a^n b^m \mid n, m \in \mathbb{N} \text{ and } n \neq m \}$. Write a CFG for L.

The main insight here is that any string of this form consists of a string of the form $a^n b^n$ either with some extra a's tacked onto the front or some extra b's tacked onto the back. We can use these insights to build this grammar:

$S \to AE \mid EB$

$E \to aEb \mid \varepsilon$

$A \to aA \mid a$

$B \to bB \mid b$

Here, E generates all strings of the form $a^n b^n$, and A and B generate a^+ and b^+, respectively.

vii. Let $\Sigma = \{a, b, c\}$ and let $L = \{a^n b^m c^p \mid n, m, p \in \mathbb{N} \text{ and } n = m \text{ or } n = p \}$. Write a CFG for L.

I love this problem because it combines everything together. There are two options here: if $n = p$, then we need to build matching a's and c's on the outside and, when we're done, we fill in the middle with b's. If $n = m$, then we build matching a's and b's and append any number of c's. The main challenge is getting everything structured appropriately. Here's one option:

$S \to XC \mid Y$

$X \to aXb \mid \varepsilon$

$C \to cC \mid \varepsilon$

$Y \to aYc \mid B$

$B \to bB \mid \varepsilon$

Here, X generates strings of the form $a^n b^n$. Y generates strings of the form $a^n b^m c^n$ by generating the matching a's and c's on the outside and filling the middle with any number of c's.

viii. Let $\Sigma = \{a, b\}$ and let $L = \{a^n b^n \mid n \in \mathbb{N}\}$. Write a CFG for L^*, the Kleene closure of L.

The grammar given below works by starting with a grammar for $\{a^n b^n \mid n \in \mathbb{N}\}$ and then letting us place down zero or more copies of strings from that language:

$S \to ES \mid \varepsilon$

$E \to aEb \mid \varepsilon$

Notice that the start symbol S expands out into some string of E's, and each E expands to a string of the form $a^n b^n$.

ix. Let $\Sigma = \{a, b\}$ and let $L = \{a^n b^m \mid n, m \in \mathbb{N}$ and either $n=2m$ or $m=2n\}$. Write a CFG for L.

The key insight here is that there are two independent options to pick from: we either have $n = 2m$ or we have $m = 2n$, and each option is something we can build a CFG for:

$S \rightarrow X \mid Y$

$X \rightarrow aXbb \mid \varepsilon$

$Y \rightarrow aaYb \mid \varepsilon$

x. Let $\Sigma = \{a, b\}$ and let $L = \{a^n b^n \mid n \in \mathbb{N}\}$. Write a CFG for $\overline{L}$, the complement of L.

The main insight here is that any string not in L must either (1) not consist of a series of a's followed by a series of b's or (2) consist of a series of a's followed by a series of b's, but not have the same quantity of each. Strings in case (1) are just the strings containing ba as a substring. Strings in case (2) are generated by the CFG from part (vi) of this problem. Putting those insights together gives us the following:

$S \rightarrow XbaX \mid AE \mid EB$

$X \rightarrow aX \mid bX \mid \varepsilon$

$E \rightarrow aEb \mid \varepsilon$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Why we asked this question: CFGs are a powerful and expressive framework for defining languages, but they're much trickier to design and work with than regular expressions. This problem was designed to give you a ton of practice writing grammars and understanding the key techniques involved (look for a build order, store information in nonterminals, think recursively, etc.) Although these languages all seem to more or less be variation on a theme, as you can see, the shapes of the grammars necessary to generate those languages look quite different!