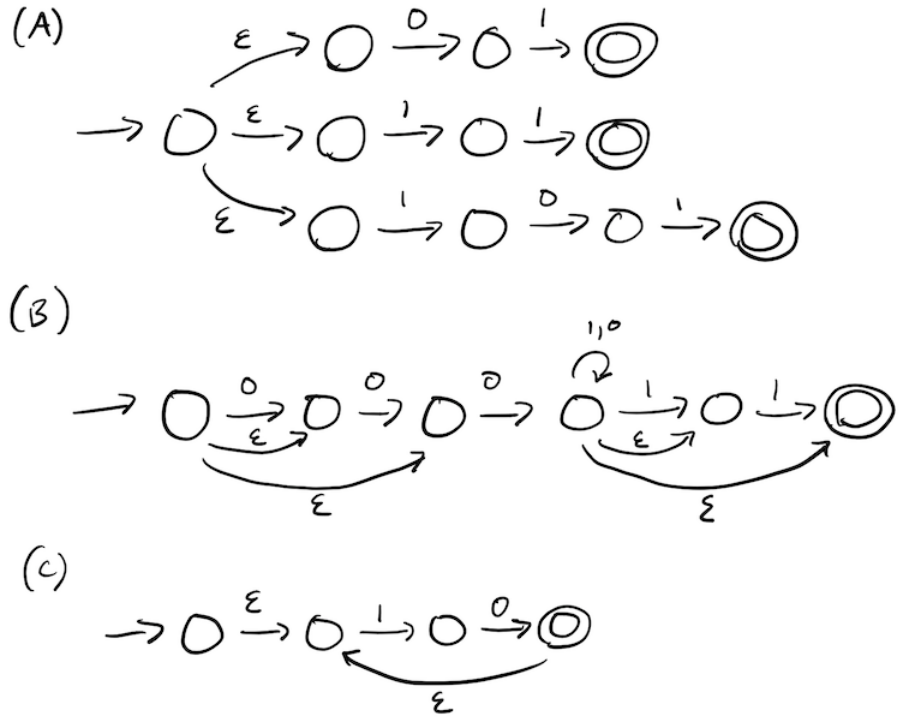
Alex Hubers
CSC:341 Spring 2024

Lab #2B

Due: February 3, 2022

#1 Interpretation



1. Determine whether the strings are accepted by the NFAs above.
   (a) 101
   (b) 0111
   (c) 1010
   (d) ϵ
2. Give a formal description of the languages that each of the NFAs above recognize.

#2 Nondeterministic Design Patterns

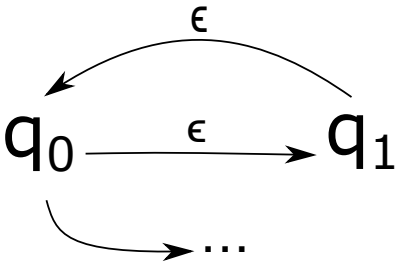
Construct an NFA that productively takes advantage of nondeterminism to recognize each of the following languages. Let $\Sigma = \{ 0, 1 \}$ for both languages.

- **Guessing**: $L_1 = \{ x \mid x \in \Sigma^*, x \text{ contains } 01010 \}$.
- **Language Union**: $L_2 = \{ x \mid x \in \Sigma^*, x \text{ has an odd number of 0s or ends in three 1s} \}$.

(Hint: build machines that recognize "odd 0s" and "ends in three 1s" separately and then combine them with nondeterminism!)

#3 Infinite Loops?

Consider the following partial NFA that features an "epsilon loop." The remainder of the NFA does some computation on q_0 that is irrelevant for our purposes.



And consider the following critique of this construction:

This NFA is impossible!
The epsilon loop will spawn an infinite number of threads of execution!

Explain in a few sentences why such a situation is "ok" by appealing to the formal definitions of a NFA. (Hint: do the definitions prescribe how we should execute a NFA?)

#4 Problem: All-NFAs

Definition 1 (all-NFA)**.** *An* all-*NFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ that behaves like an NFA except that it accepts $x \in \Sigma^*$ if every possible state that M could be in after reading input x is a state from F.*

Note, in contrast, an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove the following claim:

Claim 1. *A language is regular if and only it is recognized by some* all-*NFA.*

The definition of a regular language is on page 40 of Sipser.

Here are some steps that will lead you through the proof. Note that the only required steps you need to turn in are (f), and (g). Depending on how you think about the problem, the other steps may or may not help your thought process.

1. **Understand the Definitions**: create a small, toy all-NFA M that exhibits the essence of the all-NFA construction and give a pair of strings, one of which M accepts and the other M rejects. Give the execution traces of these strings on M.

2. **Assess the Claim**: read the claim. Identify what you must do (i.e., construct) in order to prove this claim.

3. **Explore the Design Space**: use your toy all-NFA M and convert it to an equivalent NFA. Likewise, create a small, toy NFA N and convert it into an equivalent all-NFA. Perform your conversion with an eye towards *generalizing the conversion process.* In other words, you should try to work mechanically on the structure of the machines rather than with knowledge of the particulars of their languages.

4. **State Essence**: your previous exploration should convince you that the claim is true. In a sentence or two, identify the "essence" as to why the claim is true. Try to appeal to the behaviors of NFAs/DFAs versus all-NFAs. Work through more examples if you are still unsure of the essence of this claim.

5. **Develop a Proof Strategy**: based on this essence, describe at a high-level plan how to (1) convert an all-NFA into a DFA/NFA and (2) convert an DFA/NFA into an all-NFA.

6. **Start with the Easy Case**: next, formally prove the "easy direction" of the claim.
7. **Finish It Off**: finally, finish the proof by proving the "hard direction" of the claim.