Exercise 1

In formal language theory, languages are represented as sets, and we use set operations to manipulate them. This problem asks you to recall basic set theory from CMPU 145.

Consider the following sets:

\[ A = \{a, b\} \]
\[ B = \{c, d\} \]
\[ C = \{a, b, c, d, e\} \]

a. Is \( B \in C \)? Is \( B \subseteq C \)?

b. What is \( A \cap B \)? How about \( A \cup B \)?

c. What is \( C - A \)?

d. What is \( A \times B \)?
Exercise 2

Consider the following deterministic finite automaton (DFA), \( M \):

![DFA Diagram]

a. What is the start state of \( M \)?

b. What is the set of accept (final) states of \( M \)?

c. What sequence of states does \( M \) go through on input \( bbaa \)? Does \( M \) accept this input? Why or why not?

d. Describe the set of strings accepted by \( M \).
Exercise 3

There are many ways to tile a 2 × 8 checkerboard with dominoes, two of which are shown to the right.

Notice that the horizontal dominoes must appear as stacked pairs. We can encode each tiling from left to right as a string made from the characters I and B, where I denotes a vertical domino and B denotes two horizontal dominoes. The top tiling here would be represented as BIBIII and the bottom tiling as IBBBBI.

Let Σ = {B, I}. Design the state diagram of a DFA for the language \( \{ w \in \Sigma^* \mid w \text{ represents a domino tiling of a } 2 \times 8 \text{ checkerboard} \} \).

Figure 1: Two domino tilings
Exercise 4

You're taking a walk with your dog, Dug. Dug is a vicious beast, so he's on a short leash, which keeps the distance between you at most two steps. You both start at the same position. (Dug's sitting on your feet.)

Consider the alphabet $\Sigma = \{ y, d \}$. A string in $\Sigma^*$ can be thought of as a series of events in which either you or Dug moves forward one step. For example, the string $dyd$ means Dug takes two step forward, then you take one step forward.

Design the state diagram of a DFA for the language $L = \{ w \in \Sigma^* \mid w$ describes a series of steps where you and Dug are never more than two units apart$\}$. 

Figure 2: Dug
Exercise 5

You can approximate the number of syllables in an English word by counting the number of vowels in the word (including y), except for

- vowels that have vowels directly before them, and
- the letter e, if it appears by itself at the end of a word.

For example:

- program has two vowels and two syllables: pro·gram.
- peach has two vowels, but they're consecutive so it's only one syllable.
- deduce has two syllables since the final e does not count by our rule
- oboe has two syllables. It ends with an e, but the e is preceded by another vowel.
- why has one syllable since y counts as a vowel.
- enqueue has two syllables, en·queue, since the consecutive vowels ueue all count as one.

This approach isn't always correct, e.g., it will say that area has two syllables, but it's a good approximation.

Let \( \Sigma = \{a, b, c, \ldots, z\} \). Design the state diagram of a DFA for the language \( \{w \in \Sigma^* \mid w \text{ has at least two syllables according to the above heuristic}\} \). To make your diagram simpler, you can label some transitions as vowels or consonants rather than write all the letters.

The strings in this language don't need to be English words, e.g., it include the nonsense word wekruvbsd.f.
Exercise 6

The formal mathematical definition of a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where:

- \(Q\) is a finite set whose elements are called \textit{states};
- \(\Sigma\) is a non-empty finite set whose elements are called \textit{characters} (or symbols);
- \(\delta : Q \times \Sigma \rightarrow Q\) is the \textit{transition function}, described below;
- \(q_0 \in Q\) is the start state;
- \(F \subseteq Q\) is the set of accepting (or final) states.

When we’ve drawn DFAs, we’ve represented the transitions by arrows labeled with characters. In this formal definition, the transition function \(\delta\) is what specifies these transitions. Specifically, for any state \(q \in Q\) and any symbol \(a \in \Sigma\), the transition from state \(q\) on symbol \(a\) is given by \(\delta(q, a)\).

This question explores some properties of this definition:

a. Is it possible for a DFA to have no states? If so, define a DFA with no states as a 5-tuple and explain why it meets the above requirements. If not, explain why this is not possible.

To define a machine using a 5-tuple, use this template: “Let \(D = (Q, \Sigma, \delta, q_0, F)\), where \(Q = \ldots, \Sigma = \ldots,\) etc.” For this problem, please don’t define the transition function \(\delta\) with a diagram or a table. Instead, define it like a mathematical function.

b. Is it possible for a DFA to have no accepting states? If so, define a DFA with no accepting states as a 5-tuple, and explain why it meets the above requirements. If not, explain why this is not possible.
c. In class, we said that a DFA must obey the rule that for any state and any symbol, there has to be exactly one transition defined on that symbol. What part of the definition guarantees this?

d. Is it possible for a DFA to have an unreachable state (i.e., a state that is never entered, regardless of what string you run the DFA on)? If so, define a DFA with an unreachable state as a 5-tuple, and explain why it meets the above requirements. If not, explain why this is not possible.