UC Irvine Math Circle
Finite Automata

Isaac Goldbring and Michael Hehmann*

February 26, 2024

1 Finite automata

Here is a picture of a **(deterministic) finite automaton**:

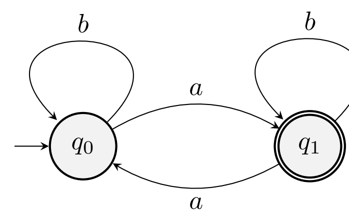


Figure 1: A simple finite automaton

The two circles represent the **states** of the machine. Here, the states are labeled q_0 and q_1. You input to the machine a **string** of a's and b's, e.g. $aaba$. The machine then begins a "computation", starting in the state which has the unlabeled arrow pointing to it, the **initial state**, then **transitions** from state to state by following the arrows that correspond to the letters in the string. So, in our example, upon input $aaba$, here is the "computation":

- The machine starts in state q_0.
- It follows the a arrow from state q_0 to state q_1.
- It then follows the a arrow from state q_1 to state q_0.
- It then follows the b arrow from state q_0 to state q_1.
- It finally follows the a arrow from state q_0 to state q_1.

Once the computation has concluded, the machine **accepts** if the machine ended up in an **accept state** (that is, a state with a double circle) and **rejects** otherwise.

Problem 1: Does the machine in Figure 1 accept upon input string $aaba$?

Solution. Yes, the computation halts in q_1, an accepting state.

Problem 2: Consider the following machine:

(a) What sequence of states does the machine go through if you input 11001?

*This work was based off a Math Circle written by Dillon Zhi at UCLA several years ago.

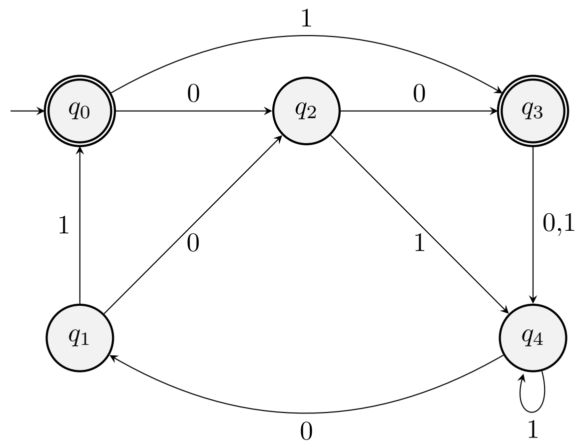


Figure 2: A slightly more complicated machine

(b) Does the machine accept 11001?

(c) If you input the **empty string** ε, does the machine accept?

Solution. (a) The machine goes through $q_0 \to q_3 \to q_4 \to q_1 \to q_2 \to q_4$.

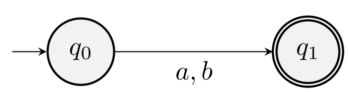
(b) It does not accept 11001 as q_4 is not an accept state.

(c) Yes, as the initial state q_0 is also an accept state.

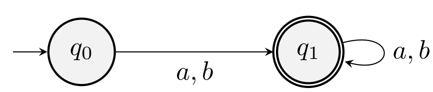
The name finite automaton refers to the fact that there are only finitely many states and the machine is self-operating. Each finite automaton has an **alphabet**, which is the set of letters it takes. In Figure 1, the alphabet was $\{a, b\}$ while the machine from Figure 2 had alphabet $\{0, 1\}$, Each state of a finite automaton has *exactly* one arrow leading out of it for each letter of its alphabet. (That is what makes these automata **deterministic**; by relaxing this condition, one arrives at **nondeterministic** automata.)

Problem 3: For the alphabet $\{a, b\}$, which of the machines below are *not* deterministic finite automata?

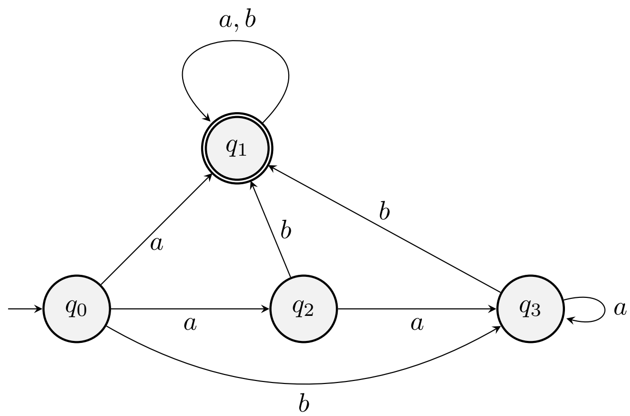
(a)



(b)



(c)



Solution. (a) Not a DFA as state q_1 does not have transitions for all alphabet symbols.

(b) This is a DFA.

(c) Not a DFA as state q_0 has two distinct transitions for a.

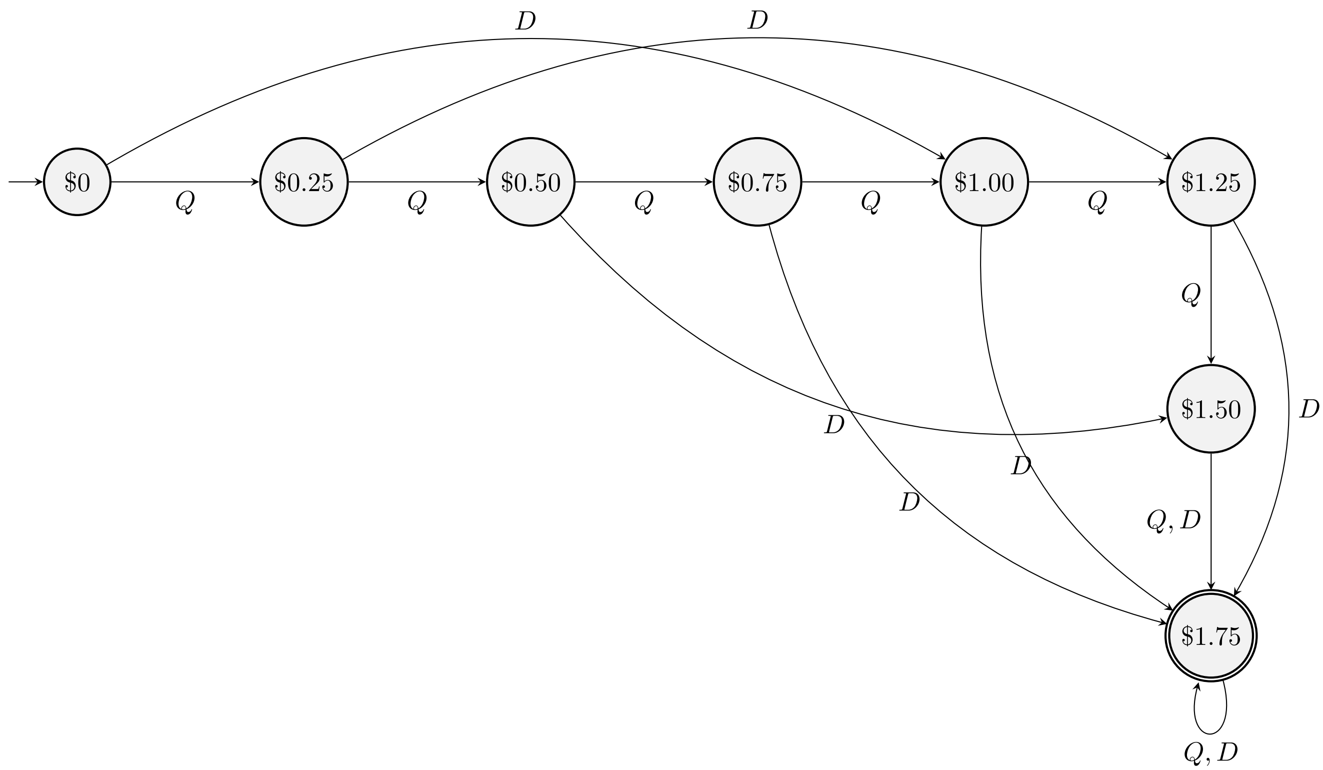
Problem 4: Suppose you are designing a vending machine which accepts only `Quarters` and `DollarBills`. The machine should only dispense a snack if at least \$1.75 has been inserted during a transaction, otherwise the machine rejects the transaction and does not dispense. Your job is to design a finite automaton to control this machine.

(a) Think about what *alphabet* this automaton should work with. How can you represent a transaction as a string?

(b) Draw a finite automaton which can determine when the vending machine should dispense a snack.

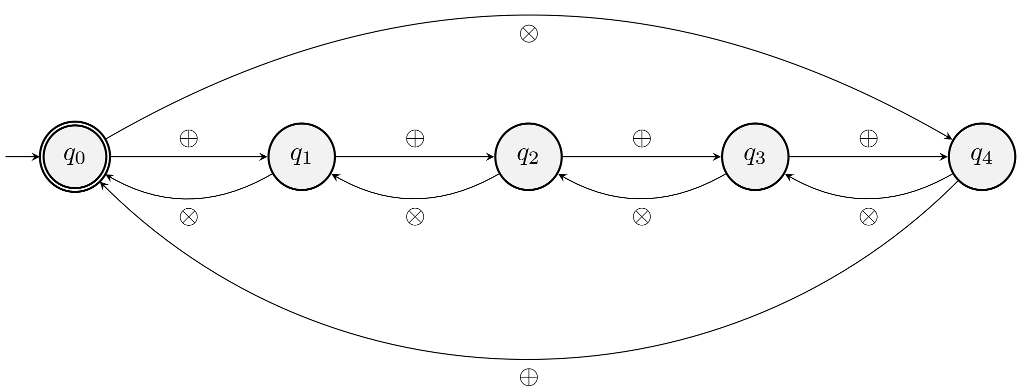
Solution. (a) The alphabet is $A = \{Q, D\}$. We think of a string w over A as encoding a transaction with instances of Q corresponding to depositing a `Quarter` and instances of D corresponding to depositing a `DollarBill`. Each state in our machine will correspond to the total amount of money that has been deposited so far in our processing of w.

(b)



Problem 5: Draw a finite automata that accepts only those strings over the alphabet $\{\oplus, \otimes\}$ for which the number of $\oplus$'s minus the number of $\otimes$'s in the string is a multiple of 5.

Solution. We will have 5 states q_i, $0 \leq i < 5$ where being in state q_i denotes that the number of $\oplus$'s minus the number of $\otimes$'s in the string scanned so far is congruent to i modulo 5.



2 Regular languages

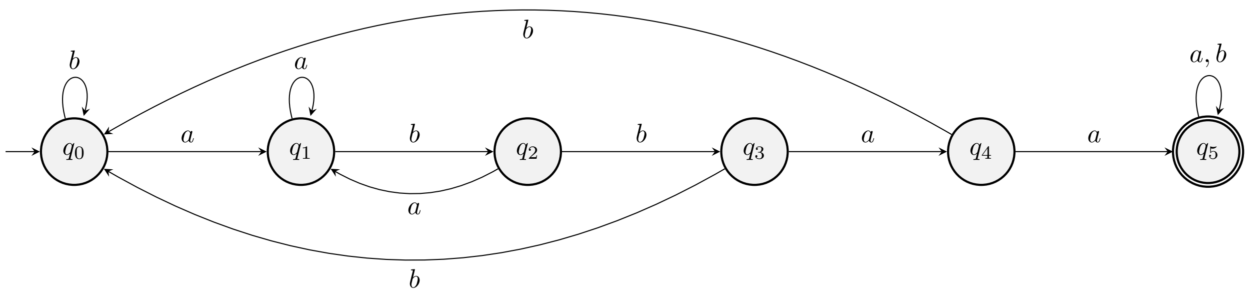
Given a finite alphabet A, we let A^* denote the set of strings from A. A **language over** A is a subset of A^*, that is, a collection of strings from the alphabet A. The collection strings accepted by a finite automaton is called the **language recognized** by the automaton and a language is called **regular** if it is the language recognized by some finite automaton.

Problem 6: Give a concrete description of the language recognized by the finite automaton from Figure 1.

Solution. The language accepted by the machine in Figure 1 consists of all those strings which end with a substring of the form $a^{2k+1}b^*$ where $k \geq 0$ and b^* denotes a string of zero or more b's.

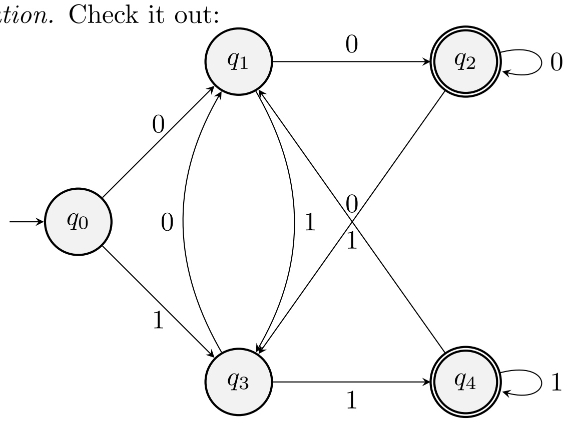
Problem 7: Draw a finite automaton with 6 states recognizing the language over the alphabet $\{a, b\}$ that contains $abbaa$ as a substring at least once.

Solution. Check it out:



Problem 8: Draw a finite automaton recognizing the language of strings over the alphabet $\{0, 1\}$ that end with either 00 or 11.

Solution. Check it out:



The next problems establish some closure properties of regular languages.

Problem 9: Suppose that L is a regular language. Show that the complement of L (consisting of those strings that do *not* belong to L) is also regular. (Hint: This is actually incredibly easy!)

Solution. Let M be the DFA which recognizes L. Let $\overline{M}$ be the machine which has the same states, transitions, and initial state as M, but swaps the accepting and non-accepting states. Then $\overline{M}$ recognizes the complement of L.

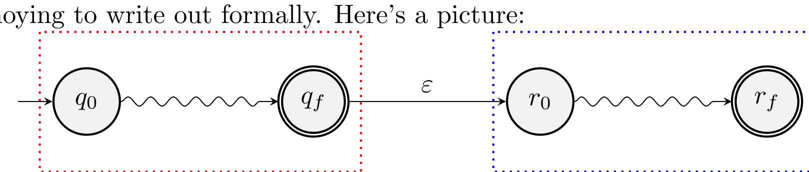
Problem 10: Suppose that L_1 and L_2 are regular languages. Prove that the **union** $L_1 \cup L_2$ (consisting of all strings that belong to either L_1 or L_2, perhaps both) and the **concatenation** $L_1 L_2$ (consisting of all strings formed by placing a string from L_1 next to a string from L_2) are also regular languages.

Solution. Let M_i recognize L_i. We need to construct a machine M to recognize $L_1 \cup L_2$. The idea is to take a product of M_1 and M_2, which has states labeled by pairs (q, r) where q is a state of M_1 and r a state of M_2. The initial state is (q_0, r_0), the product of the initial states of M_1 and M_2. We have a transition from (q_i, r_i) to (q_j, r_j) upon scanning symbol a if in M_1 we transition from q_i to q_j when scanning a and in M_2 we transition from r_i to r_j upon scanning a. Finally, the accept states are (q, r) where at least one of q or r is an accept state in their respective machines. Formally, if $M_i = \langle Q_i, \delta_i, q_0^i, F_i \rangle$ where Q_i is the set of states, $\delta_i : Q_i \times A \to Q_i$ the transition map, q_0^i the initial state, and $F_i \subseteq Q_i$ the set of accepting states, then

$$M = \langle Q_1 \times Q_2, \delta, (q_0^1, q_0^2), (F_1 \times Q_2) \cup (Q_1 \times F_2) \rangle$$

where $\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$.

Now we construct a machine M to recognize $L_1 L_2$. The idea here is to "concatenate" the machines by grafting M_2 onto M_1, placing copies of M_2 with start state placed at each accept state of M_1. Probably annoying to write out formally. Here's a picture:



3 The pumping lemma

How do you prove that a language is not regular? The main technique is the **Pumping Lemma**, which states that if L is a regular language, then there is some integer n such that, for all strings w that belong to the language whose length is at least n, there are substrings x, y, and z of w such that:

- $w = xyz$
- the length of xy is no more than n
- the length of y is at least 1, and
- for all $k \geq 0$, $xy^k z$ also belongs to L.

Here, y^k is y stringed together k times. So for strings in regular languages, as long as a word is long enough, some "interior" portion of the word can be "pumped up" as many times as you wish and the resulting word still belongs to the language.

Problem 11: Use the pumping lemma to prove that the following languages are *not* regular:

(a) The language L over the alphabet $\{a, b\}$ consisting of all strings $a^k b^k$ for $k \geq 0$.

(b) The language L over the alphabet $\{a, b\}$ consisting of all those strings that have an equal number of a's and b's.

(c) The language L over alphabet $\{a, b\}$ consisting of all *palindromes*, that is, strings which are the same when reversed.

Solution. (a) Suppose L was regular and let n be the pumping constant. Consider the string $w = a^n b^n$, of length $2n \geq n$. By the Pumping Lemma, we can write $w = xyz$ with $|xy| \leq n$, $|y| \geq 1$, and $xy^k z \in L$ for all $k \geq 0$. Since $|xy| \leq n$, it is a substring of a^n and thus y consists only of a's, and as $|y| \geq 1$, contains at least one a. Say $x = a^s$, $y = a^t$ and so $a^n = xya^{n-s-t}$ for $s + t \leq n$ and $t \geq 1$. Then $xy^2z = a^s a^{2t} a^{n-s-t} b^n = a^{n+t} b^n$, but as $n + t > n$, this is not in L, a contradiction.

Alternatively, we can "pump down" by noting that $xy^0z = xz$ must be in L. But this is clearly impossible.

(b) Same proof as part (a) works. The point is that the pumping lemma conclusion must hold for *all* strings $w \in L$ of length at least n, and so one is free to choose a particularly nice string of length $\geq n$ which contains an equal number a's and b's, for example $a^n b^n$, where it is easy to apply the lemma.

(c) Suppose L was regular and let n be the pumping constant. We wish to choose a palindrome for which it would be easy to apply the pumping lemma. Choose $w = a^n b^{2n} a^n$, which has length $4n \geq n$.

By the Pumping Lemma, we can write $w = xyz$ with $|xy| \leq n$, $|y| \geq 1$, and $xy^kz \in L$ for all $k \geq 0$. Since $|xy| \leq n$, it is a substring of a^n and thus y consists only of a's, and as $|y| \geq 1$, contains at least one a. Pumping y tells us that $xy^kz \in L$ for any $k \geq 0$. But xy^2z is a string of the form $a^m b^{2n} a^n$ where $m > n$ and this is no longer a palindrome, a contradiction.

Problem 12: Prove the pumping lemma. Here are some steps to help out. First, suppose that L is recognized by a machine with n states. Suppose w is a word in L of length bigger than n.

(a) Conclude that the machine must repeat a state q when processing the first n symbols from w.

(b) Let x denote the substring of w processed before state q was reached and let y denote the substring of w processed in between the first and second occurrences of q. Finally, let z be the "rest" of w. (Draw a picture!) Show that these x, y, and z are as desired.

Solution. (a) In processing the first n symbols of w, we start at initial state q_0 and then make n state transitions to visit a total of $n + 1$ many states. Since the machine has n states, pigeonhole principle tells use that there is some state q visited at least twice.

(b) The occurrences of q are in the processing of the first n symbols from w giving $|xy| \leq n$. Since these are two distinct occurrences of q, y cannot be the empty string, so $|y| \geq 1$. Finally, any string of the form xy^kz must be accepted by this machine, as we can process x arriving at state q, process any amount of y^k by following the computation loop from the first occurrence of q to the second occurrence k times, and then finish processing z.