CS112: Theory of Computation (LFA)

Lecture 2: Finite automata

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

March 4, 2025

Table of contents

Section 1

Previously on CS112

Proofs

For every even number $n > 3$, there is a $3 -$ *regular* graph with n nodes. A graph is k-regular if every node has degree k.
We will use a proof by construction.

- Many theorems say that a specific type of object exists. One way to prove it exists is by constructing it
- May sound weird, but this is by far the most common proof technique we will use in this course
- We may be asked to show that some property is true. We may need to construct a model which makes it clear that this property is true

Proofs

- Place the nodes in a circle and then connect each node to the ones next to it, which gives us a 2-regular graph
- Then connect each node to the one opposite it and you are done
- This is guaranteed to work because if the number of nodes is even, the opposite node will always get hit exactly once

Proofs

Prove $\sqrt{2}$ is irrational

Proof by contradiction, assume it is rational

- Rational numbers can be written as m/n for integer m, n
- Assume with no loss of generality we reduce the fraction $\sqrt{2} = m/n$
- This means that m and n cannot both be even
- Let's do some math:

$$n\sqrt{2} = m$$
$$2n^2 = m^2$$

- This mean that m^2 is even so m must be even

Section 2

Context setup

Context setup

Corresponding to Sipser 1.1

Context setup

- The theory of computation begins with a question: What is a computer?
- Silly question, as everyone know what a computer is
- But these real computers are quite complicated so it is hard to make a mathematical theory on them
- Instead, we use an idealized computer called a **computational model**
- As with any model in science, a computational model may be accurate in some ways but perhaps not in others
- We will use several different computational models, depending on the features we want to focus on
- We begin with the simplest model **finite automata**

Section 3

Finite Automata

What is a FA

- Finite automata are good models for computers with an extremely limited amount of memory.
- What can a computer do with such a small memory? **A lot**
- Let us take a real-world example: *automatic door*

Real world example

Imagine a supermarket automatic door. It has two pads or sensors, one in front and one in the back.
The controller is either of two states: *open* or *closed* There are four input conditions: *front*, *rear*, *both* and *neither*.
The controller moves from state to state, based on input it receives.

Real world example

		input signal			
		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

Figure: State transition table for an automatic door controller

Real world example

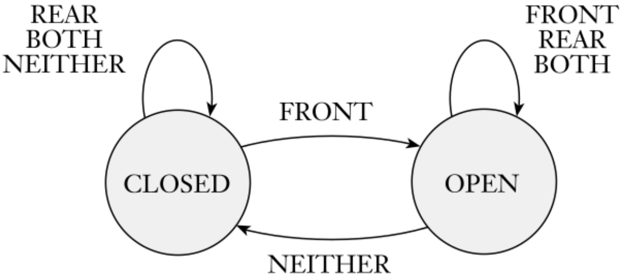

Figure: State diagram for an automatic door controller

Real world example

- This controller is a computer that has just a single bit of memory, capable of recording which of the two states the controller is in.
- More complicated controllers with more memory: elevator controller, coffee machine, etc ...

Formal Definition

- Finite automata and their probabilistic counterpart Markov chains are useful tools when we are attempting to recognize patterns in data.
- We will now take a closer look at finite automata from a mathematical perspective.
- We will develop a precise definition of a finite automaton, terminology for describing and manipulating finite automata
- Next we touch theoretical results that describe their power and limitations.

Formal Definition

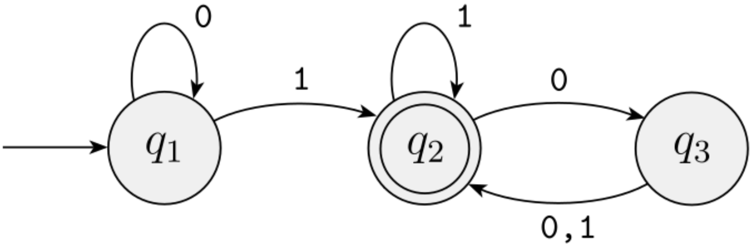

Figure: A finite automaton called M_1 that has three states

Formal Definition

If we feed feed the input string 1101 into the machine M_1 the processing proceeds as follows:

1. Start in state q_1
2. Read 1, follow transition from q_1 to q_2
3. Read 1, follow transition from q_2 to q_2
4. Read 0, follow transition from q_2 to q_3
5. Read 1, follow transition from q_3 to q_2
6. Accept because M_1 is in an accept state q_2 at the end of the input

Formal Definition

Definition

A finite automaton is 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q is a finite set called the states
2. Σ is a finite set called the alphabet
3. $\delta : Q \times \Sigma \to Q$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

Formal Definition

We can describe M_1 formally as $M_1 = (Q, \Sigma, \delta, q_0, F)$ where:

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is described

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state
5. $F = \{q_2\}$

Formal Definition

Definition

If A is the set of all strings that machine M accepts, we say that A is the language of machine M and write $L(M) = A$

We say that M **recognizes** A or that M **accepts** A. Because the term accept has different meanings when we refer to machines accepting strings and machines accepting languages, we prefer the term recognize for languages in order to avoid confusion.

Examples

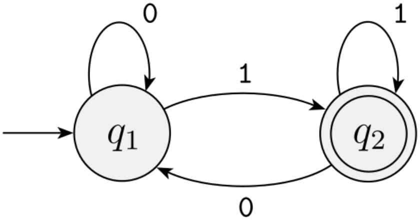

Figure: A finite automaton called M_2

In formal description M_2 is $(\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$ where transition function is:

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

Examples

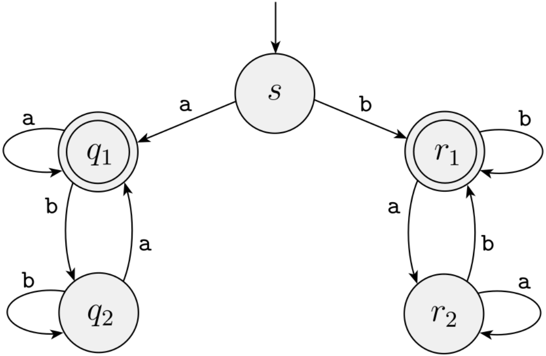

Figure: A finite automaton called M_4

M_4 operates on alphabet $\Sigma = \{a, b\}$ and accepts all strings that start and end with *a* or that start and end with *b*.

Examples

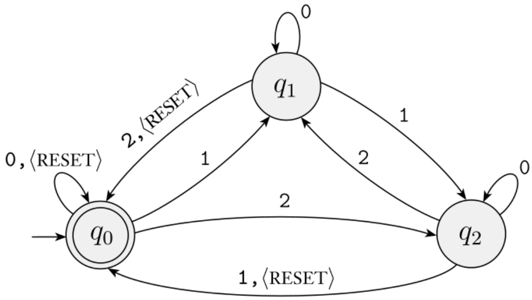

Figure: A finite automaton called M_5

Machine M_5 has alphabet $\Sigma = \{< RESET >, 0, 1, 2\}$ and keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives the $< RESET >$ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3.

Experiment

Remark

A good way to begin understanding any machine is to try it on some sample input strings. When you do these "experiments" to see how the machine is working, its method of functioning often becomes apparent.

Section 4

Computation

Formal Definition

Now we formalise finite automaton's computation as follows:
Let $M = (Q, \Sigma, \delta, q_o, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where eacg w_i is a member of Σ.

Definition

Then M **accepts** w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$
3. $r_n \in F$

Definition

A language is called a regular language if some finite automaton recognizes it.

Examples

Let's take:

$$w = 10 < RESET > 22 < RESET > 012$$

Then M_5 accepts w according to the formal definition because it exists:

$$q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$$

which satisfies the three conditions. The language is:

$L(M_5) = \{w|\text{the sum of the symbols in w modulo 3, except that} < RESET > \text{resets the count}\}$

As M_5 recognizes this language, it is a regular language.

Approach

- Automaton design is a creative process, so there is no general recipe.
- One recommended approach is to **put yourself in the place of the machine you are trying to design**
- Pretending that you are the machine is a psychological trick that helps engage your whole mind in the design process.

Section 6

Closure under regular operations

Regular operations

- Up until now we we introduced and defined finite automata and regular languages.
- Now we investigate some of their proprieties
- We define three operations on languages, called the **regular operations** and use them to study properties of the regular languages

Regular operations

Definition

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- Union: $A \cup B = \{x|x \in A \text{ or } x \in B\}$
- Concatenation: $A \circ B = \{xy|x \in A \text{ and } y \in B\}$
- Star: $A^* = \{x_1x_2 \dots x_k|k \geq 0 \text{ and each } x_i \in A\}$

empty string

empty string ϵ is always a member of A^*, no matter what A is.

Example

Let Σ be $\{a, b, \ldots, z\}$. If $A = \{good, bad\}$ and $B = \{boy, girl\}$ then:

$$\begin{aligned}
A \cup B =& \{good, bad, boy, girl\} \\
A \circ B =& \{goodboy, goodgirl, badboy, badgirl\} \\
A^* =& \{\epsilon, good, bad, goodgood, goodbad, \\
& badgood, badbad, goodgoodgood, goodgoodbad, \\
& goodbadgood, goodbadbad, \ldots\}
\end{aligned}$$

Closure under regular operations

- Generally speaking, a collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.
- Next we show that the collection of regular languages is closed under all three of the regular operations

Closure under union

Theorem

The class of regular languages is closed under the union operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Closure under union

Proof idea:

- Because A_1 and A_2 are regular, we know that some finite automaton M_1 recognizes A_1 and some finite automaton M_2 recognizes A_2
- To prove $A_1 \cup A_2$ is regular we need a finite automaton called M that recognize $A_1 \cup A_2$. This is a proof by **construction**
- This FA M must accept an input string if either M_1 or M_2 accepts it. So we simulate somehow M_1 and M_2
- Cannot be done in sequential order because once a symbol has been read then it is gone
- So we simulate M_1 and M_2 simultaneously by remembering the pair of states
- If size (i.e., number of states) of M_1 is k_1 and size of M_2 is k_2 then we have $k_1 \times k_2$ pairs

Closure under union

Theorem

The class of regular languages is closed under the union operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Proof.

Without the loss of generality we assume that M_1 and M_2 have the same alphabet. Let M_1 recognize A_1 where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and let M_2 recognize A_2 where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ We construct M to recognize $A_1 \cup A_2$ where $M = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2)\}$
- Σ is the same alphabet as for M_1 and M_2.
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$

□

Closure under concatenation

Theorem

The class of regular languages is closed under the concatenation operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.

Closure under concatenation

Proof idea:

- we can start with finite automata M_1 and M_2 recognizing the regular languages A_1 and A_2
- We must construct M such that it accept first piece as M_1 does and next the last part of the input as M_2 does. However we do not know where to break the input To tackle this problem we need a new technique called **nondeterminism**