Formal Languages and Automata Theory

Lecture 1: Introduction

Course Website

- https://atu-se.github.io/courses/flat/

Reference

- Lecture Slides Based off of UC San Diego
- https://cseweb.ucsd.edu/classes/fa08/cse105

Meeting Times

- Monday 4:30-6, 6:30-8
- Wednesday 6:30-8

How to succeed

- Attend all lectures
- Take notes (not all materials will be distributed as slides. This class may have many problems that are worked out in class.
- Study your textbook
- Work as many practice problems as possible
- Do all assignments
- Ask questions

Academic Integrity

- You are encouraged to discuss the topics among yourselves.
- When noted, you may work in groups as described in the assignments. Otherwise, your work should be your own.
- Each student must hand in her/his own solution.
- You must adhere to all rules as outlined in the course syllabus

Bibliography

Required

- Michael Sipser: Introduction to the Theory of Computation, Third Edition, Thomson.

Supplemental

- *Book of Proof*
- *Introduction to Theory of Computation* (Maheshwari/Smid)

Introduction to Computability Theory

Lecture1: Finite Automata and Regular Languages

Introduction

Computer Science stems from two starting points:

Mathematics: What can be computed?

And what **cannot be computed?**

Electrical Engineering: How can we build computers?

Not in this course.

Introduction

Computability Theory deals with the profound mathematical basis for Computer Science, yet it has some interesting practical ramifications that I will try to point out sometimes.

The question we will try to answer in this course is:

“What can be computed? What Cannot be computed and where is the line between the two?”

Computational Models

A **Computational Model** is a mathematical object (Defined on paper) that enables us to reason about computation and to study the properties and limitations of computing.

We will deal with three principal computational models in increasing order of **Computational Power.**

Computational Models

We will deal with three principal models of computations:

1. Finite Automaton (in short FA).
   recognizes Regular Languages .
2. Stack Automaton.
   recognizes Context Free Languages .
3. Turing Machines (in short TM).
   recognizes Computable Languages .

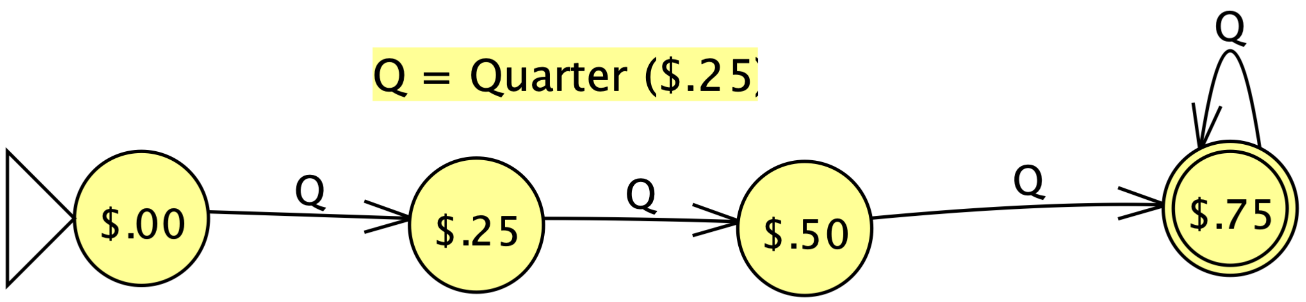
Alan Turing - A Short Detour

Dr. Alan Turing is one of the founders of Computer Science (he was an English Mathematician). Important facts:

1. "Invented" Turing machines.
2. "Invented" the **Turing Test.**
3. Broke the German submarine transmission coding machine "Enigma".
4. The movie *Imitation Game* is loosely based on his life.

Finite Automata - A Short Example

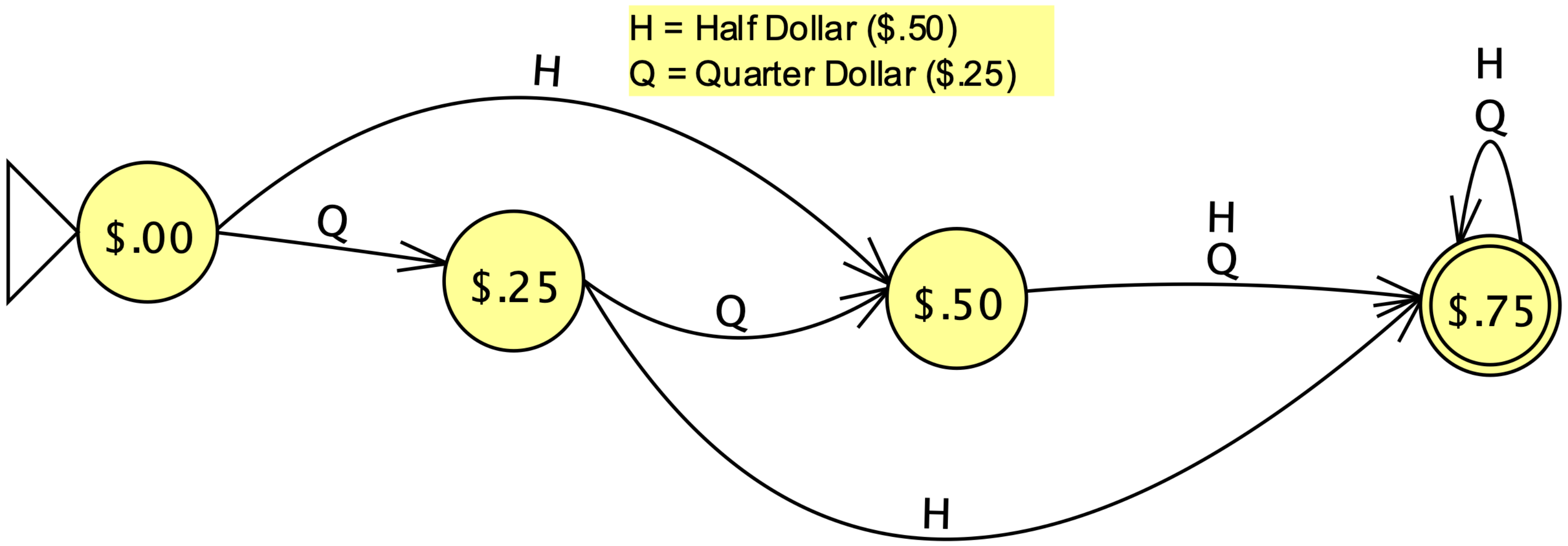
- The control of a washing machine is a very simple example of a finite automaton.
- The most simple washing machine accepts quarters and operation does not start until at least 3 quarters were inserted.
- Credit: Vadim Lyubasehvsky



- Accepts quarters
- Put in three (or more) quarters, should start washing.
- Put in less; it does nothing.

Finite Automata - A Short Example

- The control of a washing machine is a very simple example of a finite automaton.
- The most simple washing machine accepts quarters and operation does not start until at least 3 quarters were inserted.
- The second washing machine accepts 50 cents coins as well.

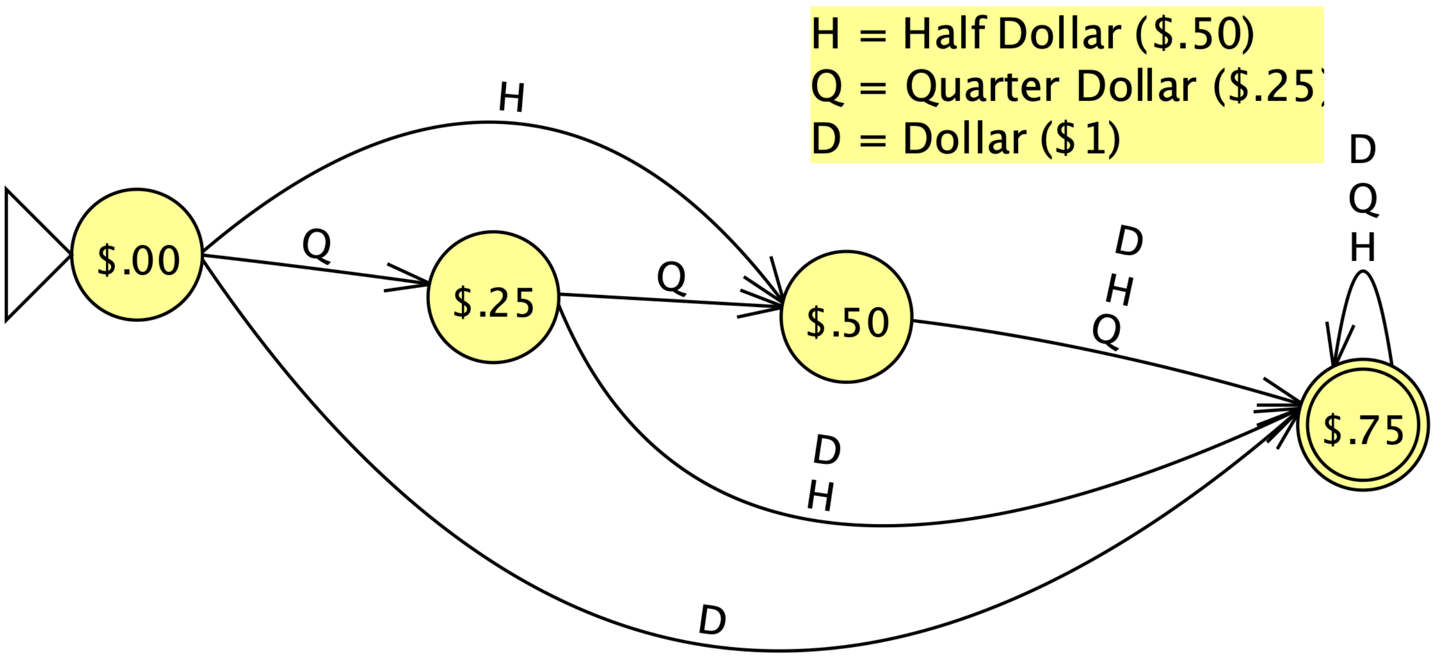


Washing Machine Example #2

Accepts quarters and fifty cent coins (half dollars)

Finite Automata - A Short Example

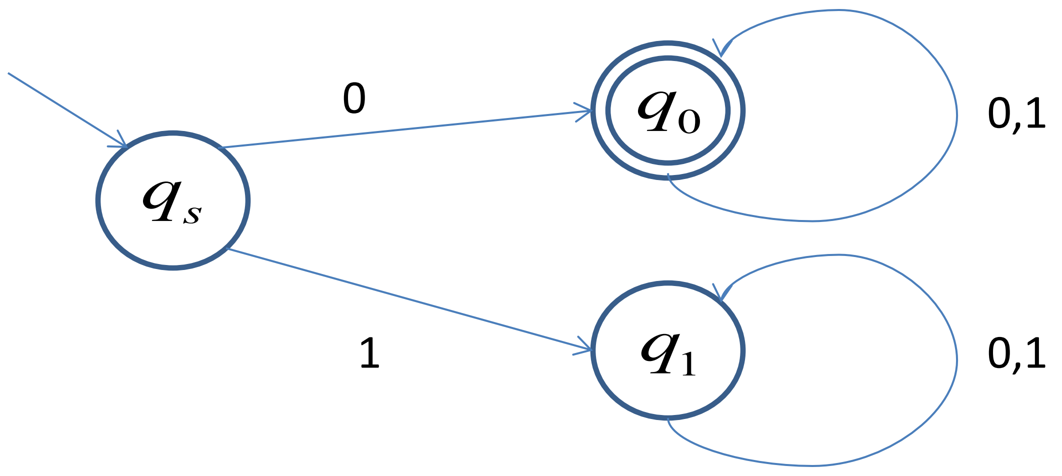
- The control of a washing machine is a very simple example of a finite automaton.
- The most simple washing machine accepts quarters and operation does not start until at least 3 quarters were inserted.
- The second washing machine accepts 50 cents coins as well.
- The most complex washing machine accepts $1 coins too.



Washing Machine Example #3

Accepts quarters, fifty cent coins (half dollars), and dollar coins

Finite Automaton - An Example



States: $Q = \{q_s, q_0, q_1\}$

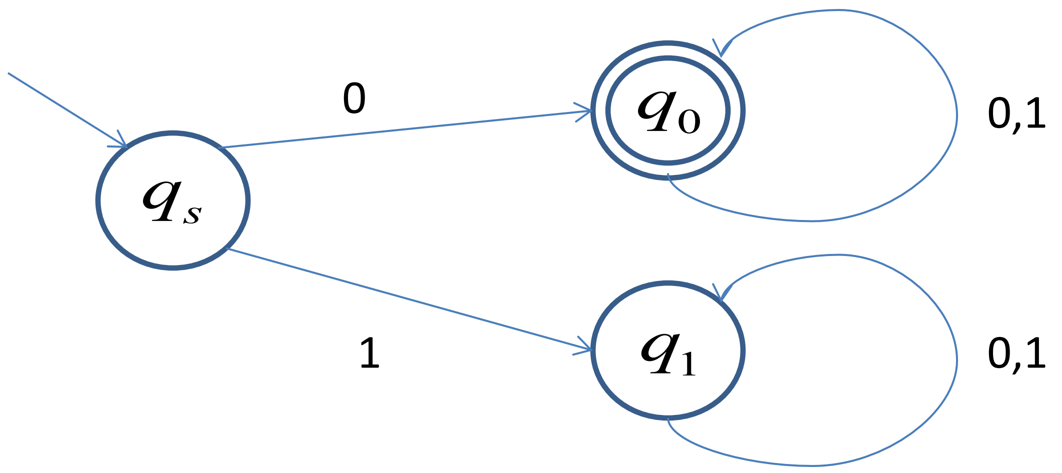
Initial State: q_s

,

Final State: q_0

,

Finite Automaton – An Example



Transition Function: $\delta(q_s,0)=q_0 \quad \delta(q_s,1)=q_1$

$\delta(q_0,0)=\delta(q_0,1)=q_0 \quad \delta(q_1,0)=\delta(q_1,1)=q_1$

Alphabet: $\{0,1\}$. **Note:** Each state has **all** transitions .

Accepted words: $0,00,01,000,001,\ldots$

Finite Automaton – Formal Definition

A ***finite automaton*** is a 5-tupple $\left(Q,\Sigma,\delta,q_0,F\right)$ 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***, and
5. $F \subseteq Q$ is the set of ***accept states***.

Observations

1. Each state has **a single transition** for each symbol in the alphabet.
2. Every FA has a computation for **every finite string** over the alphabet.

,

,

Examples

M_2 accepts all words (strings) ending with 1 over the alphabet {0, 1}.

The language ***recognized*** by M_2, called $L(M_2)$ satisfies: $L(M_2) = \{w \mid w \text{ ends with } 1\}$

<u>How to do it</u>

1. Find some simple examples (short accepted and rejected words)
2. Think what should each state "**remember"** (represent).
3. Draw the states with a proper name.
4. Draw transitions that preserve the states' **"memory"**.
5. Validate or correct.
6. Write a correctness argument.

The automaton

Correctness Argument:

The FA’s states encode the last input bit and *q1* is the only accepting state. *q1* is only reached when the last bit is a 1.

Examples

1. M_2 accepts all words (strings) ending with 1.

   $L(M_2) = \{w \mid w \text{ ends with } 1\}$.

2. M_3 accepts all words ending with 0.

3. M_4 accepts all words *not* ending with 0,

   including the **empty string** ε .

   This is the ***Complement Automaton*** of M_2 .

Examples (cont)

4. M_4 accepts all strings ***over alphabet*** $\{a, b\}$ *that start and end with the same **symbol*** .

5. M_5 accepts all words of the form $0^m 1^n$ where m, n are integers and $m, n > 0$.

6. M_6 accepts all words in $\{0,1,00,01,10\}$.

Languages

- Definition: A ***language*** is a set of strings over some alphabet .
- Examples:
  - $L_1 = \{0,1,10,1110001\}$
  - $L_2 = \{0^m 1^n \mid n, m \text{ are positive integers}\}$
  - $L_3 = \left\{\begin{array}{l}\text{bit strings whose binary} \\ \text{value is a multiple of 4}\end{array}\right\}$

Languages

- A language of an FA, M , $L(M)$, is the set of words (strings) that M **accepts**.
- If $La = L(M)$ we say that M ***recognizes*** La .
- **If La is recognized by some finite automaton A , La is a Regular Language.**

Some Questions

Q1: How do you prove that a language La is regular?

A1: By presenting an FA, M , satisfying $La = L(M)$.

Q2: How do you prove that a language La is not regular?

A2: **Hard!** to be answered on later in the course.

Q3: Why is it important?

A3: Recognition of a regular language requires a controller with bounded Memory.

The Regular Operations

Let A and B be 2 regular languages above the same alphabet, Σ. We define the 3 ***Regular Operations***:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1, x_2, \ldots, x_k \mid k \geq 0 \text{ and } x_k \in A\}$.

The Regular Operations

- given a language one can verify it is **regular** by presenting an FA that accept the language.
- **Regular Operations** give us a systematic way of constructing languages that are known to be regular: no verification is required.

The Regular Operations - Examples

$$A = \{good, bad\} \quad B = \{\mathrm{girl}, \mathrm{boy}\}$$

- $A \cup B = \{good, bad, girl, boy\}$
- $A \circ B = \{goodgirl, goodboy, badgirl, badboy\}$
- $A^* = \left\{ \begin{matrix} \varepsilon, good, bad, goodgood, goodbad, \\ goodgoodgoodbad, badbadgoodbad, \ldots \end{matrix} \right\}$

Theorem

The class of Regular languages, above the same alphabet, is **closed** under the ***union*** operation.

Meaning: The union of two regular languages is **Regular**.

.

Example

Consider $L_1 = \{w \mid w \text{ starts with } 1\}$, and
$L_2 = \{w \mid w \text{ ends with } 0\}$.

The union set is the set of all bit strings that either start with 1, or end with 0.

Each of these sets can be recognized by an FA with 3 states.

Can you construct an FA that recognizes the union set?

Proof idea

If A_1 and A_2 are regular, each has its own recognizing automaton N_1 and N_2 , respectively.

In order to prove that the language $A_1 \cup A_2$ is regular we have to construct an FA that accepts exactly the words in $A_1 \cup A_2$.

Note: N_1 followed by N_2 doesn't work.

Proof idea

- We construct an FA that **simulates** the computations of, N_1 and N_2 **simultaneously**.
- Each state of the simulating FA represents a pair of states of N_1 and of N_2 respectively.
- Can you define the transition function and the final state(s) of the simulating FA?

Wrap up

In this talk we:

1. Motivated the course.
2. Defined **Finite Automata (**Latin Pl. form of automaton)**.**
3. Learned how to deal with construction of automata and how to come up with a correctness argument.