CS420
Finite Automata and Regular Languages
Wed Jan 27, 2021
UMass Boston Computer Science
Programming in Linux: Basics
(15 minute crash course)
Code Demo

- stdin, stdout
- command line
- command line scripts
- Makefiles
A Makefile

setup: # install your language here (you can probably leave it blank)

run-hw0-stdio:
  racket hello.rkt # this line must start with a tab

run-hw0-alphabet:
  racket alphabet.rkt

run-hw0-powerset:
  racket powerset.rkt

run-hw0-xml:
  racket xml.rkt

Grader Preinstalled langs: Python, Java, C, C++, JS, Racket

Targets (see hw for names)

Commands to run (these files better exist)
HW 0 Questions?
Last time: The Theory of Computation ... 

• Creates and studies mathematical models of computers 

• In order to: 
  • Make predictions about computer programs 
  • Explore the limits of computation
Last time: Levels of Computational Power

We’ll start here
Finite Automata

or

Deterministic Finite Automata (DFA)
State Machine
Finite State Machine (FSM)
Finite Automata: A computational model for ...
A Microwave Finite Automata

Inputs change states (possibly)

press stop  press start

press start  press stop

idle  cook

States
Finite Automata: Not Just for Microwaves

Finite Automata: a common programming pattern

State pattern

From Wikipedia, the free encyclopedia

The state pattern is a behavioral software design pattern that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of finite-state machines. The state pattern can be interpreted as a strategy pattern, which is able to switch a strategy through invocations of methods defined in the pattern's interface.
Video Games Love Finite Automata
Finite Automata in Video Games

// simple_state_machine.h
// Simple finite state machine encapsulation
// Author: Michael S. Booth (mike@turtlerockstudios.com), November 2003

#ifndef _SIMPLE_STATE_MACHINE_H_
#define _SIMPLE_STATE_MACHINE_H_

/**
 * Encapsulation of a finite-state-machine state
 */

template <typename T>
class SimpleState

Model-view-controller (MVC) is a FSM

States

Inputs change states

The View Draws states
A Finite Automata is a Computer!

• A very limited computer with finite memory
  • Memory = states

• In this class, we’ll formally study automata as:
  • State diagrams
  • Formal mathematical model
  • Code simulations of the mathematical model
Finite Automata state diagram

Start State

States

Accept State

Inputs cause state transitions
A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

5 components
A **finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the **states**,  
2. \(\Sigma\) is a finite set called the **alphabet**,  
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(F \subseteq Q\) is the **set of accept states**.

**Example:** as state diagram
**Definition 1.5**

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

1. \( Q \) is a finite set called the **states**,  
2. \( \Sigma \) is a finite set called the **alphabet**,  
3. \( \delta: Q \times \Sigma \rightarrow Q \) is the **transition function**,  
4. \( q_0 \in Q \) is the **start state**, and  
5. \( F \subseteq Q \) is the **set of accept states**.

**Example: as formal description**

\[ M_1 = (Q, \Sigma, \delta, q_1, F), \text{ where} \]

1. \( Q = \{q_1, q_2, q_3\} \),  
2. \( \Sigma = \{0, 1\} \),  
3. \( \delta \) is described as

\[
\begin{array}{c|cc}
0 & 1 & \\
\hline
q_1 & q_1 & q_2 \\
q_2 & q_3 & q_2 \\
q_3 & q_2 & q_2 \\
\end{array}
\]

4. \( q_1 \) is the start state, and  
5. \( F = \{q_2\} \).

**Example: as state diagram**
**Definition 1.5**

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

1. \(Q\) is a finite set called the **states**,
2. \(\Sigma\) is a finite set called the **alphabet**,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,
4. \(q_0 \in Q\) is the **start state**, and
5. \(F \subseteq Q\) is the **set of accept states**.

**Example: as formal description**

\[
M_1 = (Q, \Sigma, \delta, q_1, F),
\]

where

1. \(Q = \{q_1, q_2, q_3\}\),
2. \(\Sigma = \{0, 1\}\), \textbf{Possible inputs}
3. \(\delta\) is described as

<table>
<thead>
<tr>
<th>()</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>(q_1)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_3)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_2)</td>
<td>(q_2)</td>
</tr>
</tbody>
</table>

4. \(q_1\) is the start state, and
5. \(F = \{q_2\}\).

**Example: as state diagram**
**Definition 1.5**

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

1. \(Q\) is a finite set called the **states**,  
2. \(\Sigma\) is a finite set called the **alphabet**,  
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(F \subseteq Q\) is the **set of accept states**.

**Example: as formal description**

\[M_1 = (Q, \Sigma, \delta, q_1, F),\]

1. \(Q = \{q_1, q_2, q_3\}\),  
2. \(\Sigma = \{0, 1\}\),  
3. \(\delta\) is described as:  

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

   - "If in this state"  
   - "And this is next input symbol"  
   - "Then go to this state"  

4. \(q_1\) is the start state, and  
5. \(F = \{q_2\}\).
**Definition 1.5**

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

1. \(Q\) is a finite set called the **states**,  
2. \(\Sigma\) is a finite set called the **alphabet**,  
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(F \subseteq Q\) is the **set of accept states**.

**Example: as formal description**

\[M_1 = (Q, \Sigma, \delta, q_1, F)\], where

1. \(Q = \{q_1, q_2, q_3\}\),  
2. \(\Sigma = \{0,1\}\),  
3. \(\delta\) is described as

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>(q_1)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_3)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_2)</td>
<td>(q_2)</td>
</tr>
</tbody>
</table>

4. \(q_1\) is the start state, and  
5. \(F = \{q_2\}\).
**Definition 1.5**

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

1. \(Q\) is a finite set called the **states**,
2. \(\Sigma\) is a finite set called the **alphabet**,
3. \(\delta : Q \times \Sigma \rightarrow Q\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(F \subseteq Q\) is the **set of accept states**.

**Example: as formal description**

\[ M_1 = (Q, \Sigma, \delta, q_1, F), \]

where

1. \(Q = \{q_1, q_2, q_3\}\),
2. \(\Sigma = \{0, 1\}\),
3. \(\delta\) is described as

\[
\begin{array}{c|cc}
   & 0 & 1 \\
\hline
q_1 & q_1 & q_2 \\
q_2 & q_3 & q_2 \\
q_3 & q_2 & q_2 \\
\end{array}
\]

4. \(q_1\) is the start state, and
5. \(F = \{q_2\}\).
Precise Terminology is Important

• **Currently**, these terms are all equivalent:
  • Finite State Machine (FSM), State Machine
  • Finite Automaton, Automaton
  • Deterministic Finite Automata (DFA)

• They describe a specific kind of FSM, defined in Definition 1.5

• **Eventually**, we’ll learn about many different variations of finite automata:
  • Deterministic Finite Automata (DFA)
  • Non-Deterministic Finite Automata (NFA)
  • Generalized Non-Deterministic Finite Automata (GNFA)

• **Then**, these terms will generally describe the class of machines studied in Ch 1
  • Finite State Machine (FSM), State Machine
  • Finite Automaton, Automaton

• **But** all these machines are related; they are equivalent in “power”
“Running” an FSM “Program” (JFLAP demo)

- **FSM:**

  ![FSM Diagram]

- **Program:** “1101”
The Computation Model

**Informally**
- **Computer** = some finite automata
- **Program** = input string of chars
- **Start** in “start state”
- 1 char at a time, follow **transition** table to change states
- **Result** =
  - “Accept” if last state is “Accept” state
  - “Reject” otherwise

**Formally (i.e., mathematically)**
- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1w_2 \cdots w_n$
- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$
- $M$ accepts $w$ if sequence of states $r_0, r_1, \ldots, r_n$ in $Q$ exists with $r_n \in F$
Terminology

• $M$ accepts $w$

• $M$ recognizes language $A$
  if $A = \{ w | M$ accepts $w \}$

• A language is called a regular language
  if some finite automaton recognizes it.
Proving that a language is regular
Kinds of Mathematical Proof

• Proof by construction
  • Construct the mathematical object in question

• Proof by contradiction

• Proof by induction
Proving that a language is regular

• (Usually) requires creating a FSM

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

• States = the machine’s memory!
  • Finite amount of memory: must be allocated in advance
  • Think about what information must be remembered.

• Example: machine accepts strings with even number of 0s
  • Two states: 1) seen even number of 0s, 2) seen odd number of 0s

• Input may only be read once

• Must decide accept/reject after that
In-class exercise 1

• Come up with a formal description of the following machine:

DEFINITION 1.5

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

1. \(Q\) is a finite set called the **states**, 
2. \(\Sigma\) 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**.
In-class exercise 2

• Design machine $M$ that recognizes: \{w | w has exactly three 1’s\}

• Where $\Sigma = \{0, 1\}$,

• Remember:

**Definition 1.5**

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

1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.
Check-in Quiz 1/27

See Gradescope