CS420
Finite Automata and Regular Languages

Instructor: Stephen Chang
Mon Sept 14, 2020
UMass Boston Computer Science
HW 0 Questions?
Quick Poll: Regular Expressions

The Good, the Bad, the ... Weird?

*REgEx Golf:* You try to match one group but not the other. `/m` / `i`/j/ matches Star Wars subtitles but not Star Trek. COOL.

*META-REgEx Golf:* So I wrote a program that plays REGex golf with arbitrary lists. Uh oh.

*META-META-REgEx Golf:* But I lost my code, so I'm grepping for files that look like REGex golf solvers.

...and beyond:

Really, this is all /META-META-REgEx Golf/. Now you have infinite problems. No, I had those already.
Deterministic Finite Automata (DFAs)
A computational model for ...
A Finite Automata (or State Machine)

Inputs change states (possibly)

Press stop

Press start

States

Press start

Press stop

Press stop

Press start

idle

cook
Finite Automata: Not Just for Microwaves

Finite Automata: a common programming pattern
Finite Automata in: Video Games

ValveSoftware / halflife

halflife / game_shared / bot / simple_state_machine.h

Alfred Reynolds initial seed of Half-Life 1 SDK

0 contributors

85 lines (67 sloc) | 2.15 KB

// 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_

#ifndef SIMPLE_STATE_MACHINE_H_
#endif

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

template <typename T>
class SimpleState
Model-view-controller (MVC) is a FSM

States

Inputs change states

View Draw states
Finite Automata in this class: state diagram

Start State → q1

q1 → q2 (Input: 0, Transition: 1)
q2 → q3 (Input: 1, Transition: 0)
q3 → Accept State (Input: 0, Transition: 1)

States

Inputs cause state transitions
JFLAP demo: “Running” an FSM “Program”

• FSM:

• Program: “1101”
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.
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**.

\[ M_1 = (Q, \Sigma, \delta, q_1, F), \] 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. \(F = \{q_2\}\).
**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*.

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

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. \(F = \{q_2\}\).
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**.

$M_1 = (Q, \Sigma, \delta, q_1, F)$, 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. $F = \{q_2\}$. 
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**.

$M_1 = (Q, \Sigma, \delta, q_1, F)$, 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**.

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

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. \(F = \{q_2\}\).
In-class exercise

- 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. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(F \subseteq Q\) is the **set of accept states**.
Terminology

• These are all equivalent:
  • Finite State Machine (FSM)
  • Finite Automaton, Automata, Automaton
  • State Machine

• They generally describe the class of machines studied in Ch 1

• What I just introduced:
  • Deterministic Finite Automata (DFA)

• A specific kind of FSM, corresponding to Definition 1.5

• At this point in the course all terms on this slide are the same
  • But they wont be later
Math vs Its Code Representation

• In CS420 we use code to explore mathematical objects

• But it’s important to understand the distinction

• E.g., a set is an **abstract** mathematical object
  • contains other math objects like: strings, nums, characters, and other sets!

• A set’s (data) **representation** in code can take many forms:
  • e.g., a list, an array, a space-separated string

• This course teaches abstract mathematical concepts
  • It is up to you how to represent the math as code and data!
## Math vs Representation, Examples

<table>
<thead>
<tr>
<th>Abstract Math Concept</th>
<th>Possible Data Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td></td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td></td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td></td>
</tr>
<tr>
<td>Finite automata</td>
<td></td>
</tr>
<tr>
<td>Abstract Math Concept</td>
<td>Possible Data Representation</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Numbers</td>
<td>Int, BigInt, float, double</td>
</tr>
<tr>
<td>Set</td>
<td></td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td></td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td></td>
</tr>
<tr>
<td>Finite automata</td>
<td></td>
</tr>
</tbody>
</table>
### Math vs Representation, Examples

<table>
<thead>
<tr>
<th><strong>Abstract Math Concept</strong></th>
<th><strong>Possible Data Representation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>Int, BigInt, float, double</td>
</tr>
<tr>
<td>Set</td>
<td>List, array, tree</td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td></td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td></td>
</tr>
<tr>
<td>Finite automata</td>
<td></td>
</tr>
</tbody>
</table>
## Math vs Representation, Examples

<table>
<thead>
<tr>
<th>Abstract Math Concept</th>
<th>Possible Data Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>Int, BigInt, float, double</td>
</tr>
<tr>
<td>Set</td>
<td>List, array, tree</td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td>Struct, object, list</td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td></td>
</tr>
<tr>
<td>Finite automata</td>
<td></td>
</tr>
</tbody>
</table>
### Math vs Representation, Examples

<table>
<thead>
<tr>
<th>Abstract Math Concept</th>
<th>Possible Data Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>Int, BigInt, float, double</td>
</tr>
<tr>
<td>Set</td>
<td>List, array, tree</td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td>Struct, object, list</td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td>Function, dict, map, hash, tree</td>
</tr>
<tr>
<td>Finite automata</td>
<td></td>
</tr>
</tbody>
</table>
## Math vs Representation, Examples

<table>
<thead>
<tr>
<th>Abstract Math Concept</th>
<th>Possible Data Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>Int, BigInt, float, double</td>
</tr>
<tr>
<td>Set</td>
<td>List, array, tree</td>
</tr>
<tr>
<td>Tuple (i.e., a small finite set)</td>
<td>Struct, object, list</td>
</tr>
<tr>
<td>Function, i.e., a set of pairs</td>
<td>Function, dict, map, hash, tree</td>
</tr>
<tr>
<td>Finite automata</td>
<td>XML str, &lt;your choice here&gt;</td>
</tr>
</tbody>
</table>
“Running a Program” on a Finite Automata

• Program = an input string of characters

• Start in “Start State”

• One char at a time, follow transition table to change states

• Result of running the program:
  • “Accept” the input if last state is an “Accept State”
  • “Reject” otherwise
Formal Definition of “Computation”

\[ M = (Q, \Sigma, \delta, q_0, F) \quad \text{a finite automaton} \]
\[ w = w_1w_2 \cdots w_n \quad \text{a string where each } w_i \text{ is a member of the alphabet } \Sigma. \]

\[ M \text{ accepts } w \text{ if a sequence of states } r_0, r_1, \ldots, r_n \text{ in } Q \text{ exists with three conditions:} \]

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

Condition 1 machine starts in the start state.
Condition 2 machine goes from state to state according to the transition function.
Condition 3 machine accepts its input if it ends up in an accept state.
Terminology

- $M$ accepts $w$

- $M$ recognizes language $A$
  \[
  \text{if } A = \{ w \mid M \text{ accepts } w \} 
  \]

- A language is called a \textit{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

- Often 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 example

• Design machine $M$ that recognizes: \{w \mid w \text{ 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

https://www.gradescope.com/courses/160337/assignments/650219
End of Class survey 9/14

https://forms.gle/pZqmX3urYRN5sn3t5