

CS420
(Deterministic) Finite Automata

Monday, January 29, 2024

UMass Boston Computer Science

Announcements

- **HW**
 - Weekly; in/out Mon noon
 - HW 0 in, HW 1 out
 - ~3-4 questions, Paper-and-pencil proofs (no programming)
 - Discussing with classmates ok
 - Final answers written up and submitted individually
- **Lectures**
 - Slides posted
 - Closely follow the listed textbook chapters
- **Office Hours**
 - Wed 11:30-1pm (in person, McCormack 3rd floor, Rm 201)
 - Fri 11:30-1pm (zoom, access link from blackboard)
 - Let me know in advance if possible, but drop-ins also fine
 - TAs TBD

Last Time: How Mathematics Works

Today:

- “Facts” can have many different “shapes”!
- How do we USE known facts?
- How can we PROVE new facts?

Mathematician
(or student)

It's not easy to create the next
level ...
Preciseness is important

Proofs = Figure out how to (precisely) fit
known “facts” together



More **Theorems**

More **Axioms**

More **Definitions**

Theorem

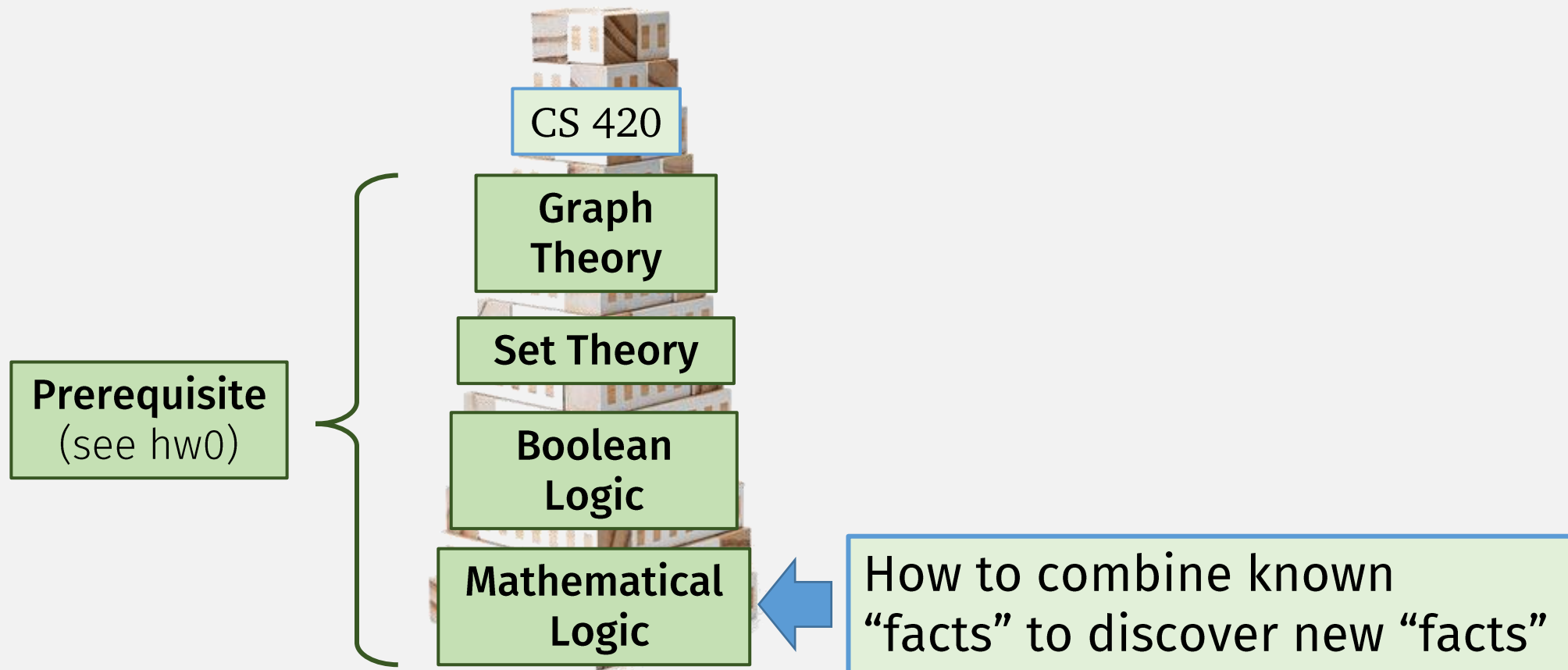
Theorem

Axioms

Definitions

“facts”

Last Time: How CS 420 Works



Mathematical Logic Operators

- **Conjunction** (AND, \wedge)
- **Disjunction** (OR, \vee)
- **Negation** (NOT, \neg)
- **Implication** (IF-THEN, \Rightarrow , \rightarrow)
- ...

This semester:


Must understand difference
between **Using** vs **Proving** a
mathematical statement!

Mathematical Statements: AND

Using:

- If we know $A \wedge B$ is TRUE, what do we know about A and B individually?
 - A is TRUE, and
 - B is TRUE

A	B	$A \wedge B$
True	True	True
True	False	False
False	True	False
False	False	False



Mathematical Statements: AND

Using:

- If we know $A \wedge B$ is TRUE, what do we know about A and B individually?
 - A is TRUE, and
 - B is TRUE

Proving:

- To prove $A \wedge B$ is TRUE:
 - Prove A is TRUE, and
 - Prove B is TRUE

A	B	$A \wedge B$
True	True	True
True	False	False
False	True	False
False	False	False



Mathematical Statements: IF-THEN

Using:

- If we know $P \rightarrow Q$ is TRUE, what do we know about P and Q individually?
 - Either P is FALSE, or
 - If we **prove** P is TRUE, then Q is TRUE (**modus ponens**)

Proving:

p	q	$p \rightarrow q$	
True	True	True	←
True	False	False	⊗
False	True	True	←
False	False	True	←

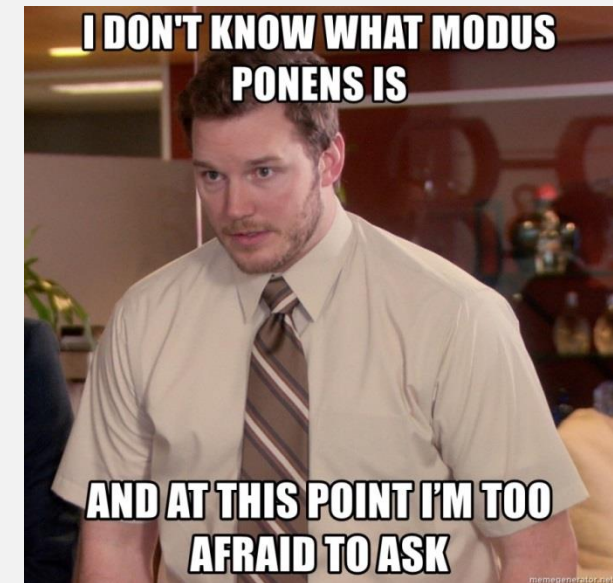
Using an IF-THEN statement: The “Modus Ponens” Inference Rule

Premises (if these statements are true)

- If P then Q
- P is TRUE

Conclusion (then we can say that this is also true)

- Q must also be TRUE



Mathematical Logic Operators: IF-THEN

Using:

- If we know $P \rightarrow Q$ is TRUE, what do we know about P and Q individually?
 - Either P is FALSE, or
 - If we **prove** P is TRUE, then Q is TRUE (**modus ponens**)

Proving:

- To prove $P \rightarrow Q$ is TRUE:
 - Either **Prove** P is FALSE (usu. hard or impossible), or
 - **Assume** (not prove!) P is TRUE, then **prove** Q is TRUE

p	q	$p \rightarrow q$	
True	True	True	←
True	False	False	
False	True	True	←
False	False	True	←

Example: Proving an IF-THEN Statement

Prove the following:

Proving IF-THEN

Using IF-THEN

• IF: If $x \geq 4$, then $2^x \geq x^2$

Assume this (AND stmt) is true

Using AND

AND: x is the sum of the squares of four positive integers

• THEN: $2^x \geq x^2$

Prove this is true

Proving:

To prove $P \rightarrow Q$ is TRUE:

Either Prove P is FALSE (usu. hard or impossible), or
Assume (not prove!) P is TRUE, then prove Q is TRUE

p	q	$p \rightarrow q$
True	True	True
True	False	False
False	True	True
False	False	True



Example: Proving an IF-THEN Statement

Prove: IF If $x \geq 4$, then $2^x \geq x^2$ AND x is the sum of the squares of four positive integers
THEN $2^x \geq x^2$

Proof:

Statement

1. $x = a^2 + b^2 + c^2 + d^2$
2. $a \geq 1; b \geq 1; c \geq 1; d \geq 1$

5. If $x \geq 4$, then $2^x \geq x^2$
6. $2^x \geq x^2$

Justification

1. Assumption (IF part of IF-THEN)
 2. Assumption (IF part of IF-THEN)
-
5. Assumption (IF part of IF-THEN)

Example: Proving an IF-THEN Statement

Prove: IF If $x \geq 4$, then $2^x \geq x^2$ AND x is the sum of the squares of four positive integers
THEN $2^x \geq x^2$

Proof:

Statement

- 1. $x = a^2 + b^2 + c^2 + d^2$
- 2. $a \geq 1; b \geq 1; c \geq 1; d \geq 1$
- 3. $a^2 \geq 1; b^2 \geq 1; c^2 \geq 1; d^2 \geq 1$

4. $x \geq 4$

5. If $x \geq 4$, then $2^x \geq x^2$

→ 6. $2^x \geq x^2$

Justification

- 1. Assumption (IF part of IF-THEN)
- 2. Assumption (IF part of IF-THEN)
- 3. By Stmt #2 & arithmetic laws
- 4. Stmts #1, #3, and arithmetic
- 5. Assumption (IF part of IF-THEN)
- 6. Stmts #4 and #5

Modus Ponens

If we can prove these:

- If P then Q

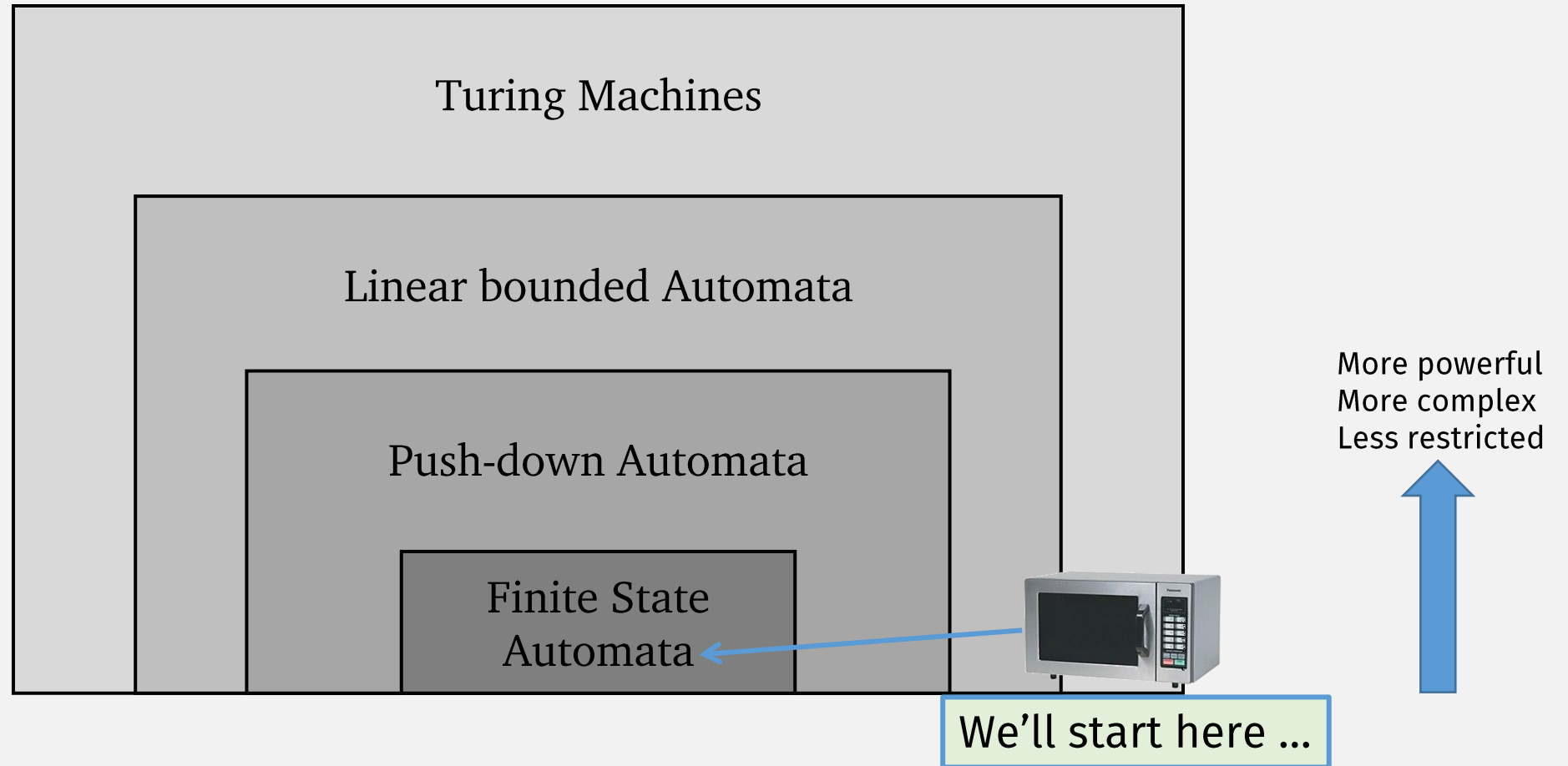
- P

Then we've proved:

- Q

←

Last Time: Models of Computation Hierarchy



Finite Automata: “Simple” Computation / “Programs”

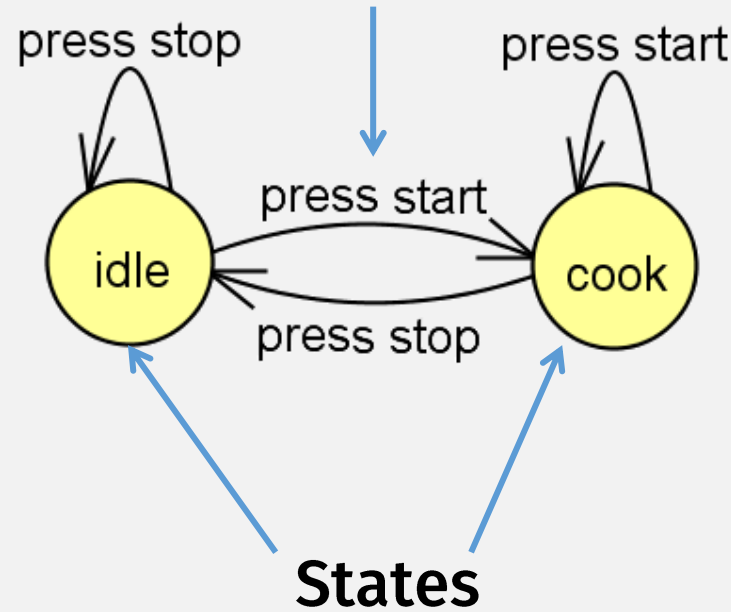


Finite Automata

- A **finite automata** or **finite state machine (FSM)** ...
- ... computes with a finite number of **states**

A Microwave Finite Automata

Input “symbols” change states
(possibly)



Finite Automata: Not Just for Microwaves

Finite Automata:
a common
programming pattern



State pattern

From Wikipedia, the free encyclopedia

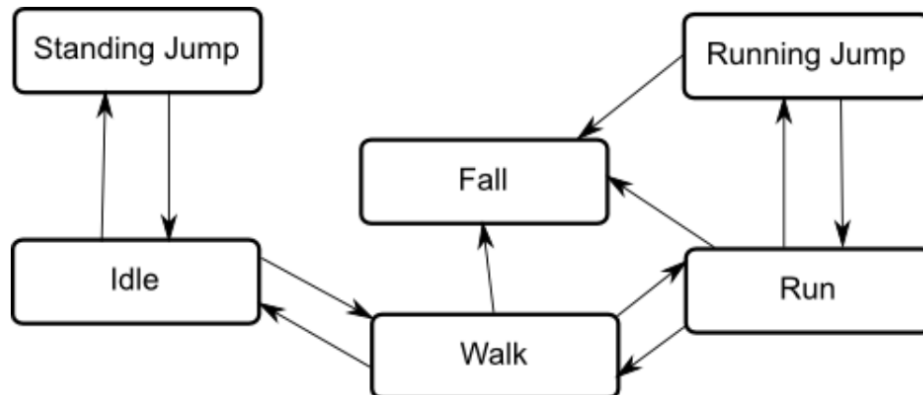
The **state pattern** is a [behavioral software design pattern](#) that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of [finite-state machines](#). The state pattern can be interpreted as a [strategy pattern](#), which is able to switch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

(More powerful) **Computation Simulating**
other (weaker) **Computation**
(a common theme this semester)

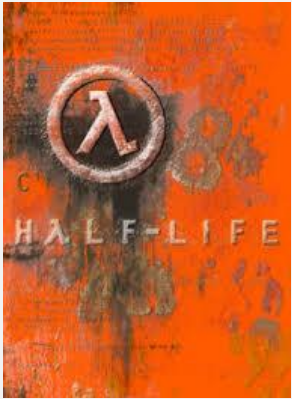
Video Games Love Finite Automata

The basic idea is that a character is engaged in some particular kind of action at any given time. The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc. These actions are referred to as **states**, in the sense that the character is in a “state” where it is walking, idling or whatever. In general, the character will have restrictions on the next state it can go to rather than being able to switch immediately from any state to any other. For example, a running jump can only be taken when the character is already running and not when it is at a standstill, so it should never switch straight from the idle state to the running jump state. The options for the next state that a character can enter from its current state are referred to as **state transitions**. Taken together, the set of states, the set of transitions and the variable to remember the current state form a **state machine**.

The states and transitions of a state machine can be represented using a graph diagram, where the nodes represent the states and the arcs (arrows between nodes) represent the transitions. You can think of the current state as being a marker or highlight that is placed on one of the nodes and can then only jump to another node along one of the arrows.



Finite Automata in Video Games



ValveSoftware / **halflife**

<> Code ⓘ Issues 1.6k 🔗 Pull requests 23 ⏮ Actions 📁 Projects 📖 Wiki

🔗 5d761709a3 ▾ [halflife](#) / [game_shared](#) / [bot](#) / [simple_state_machine.h](#)

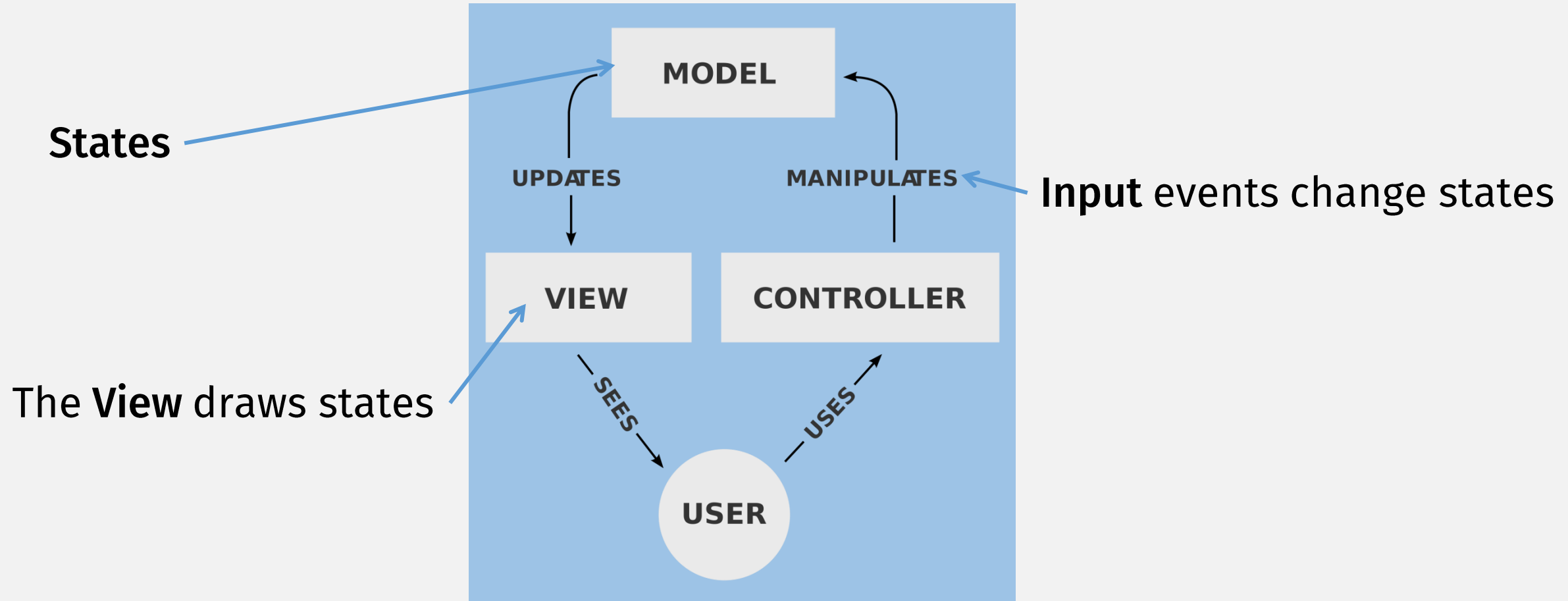
🐙 Alfred Reynolds initial seed of Half-Life 1 SDK

👤 0 contributors

85 lines (67 sloc) | 2.15 KB

```
1 // simple_state_machine.h
2 // Simple finite state machine encapsulation
3 // Author: Michael S. Booth (mike@turtlerockstudios.com), November 2003
4
5 #ifndef _SIMPLE_STATE_MACHINE_H_
6 #define _SIMPLE_STATE_MACHINE_H_
7
8 //-----
9 /**
10  * Encapsulation of a finite-state-machine state
11  */
12 template < typename T >
13 class SimpleState
```

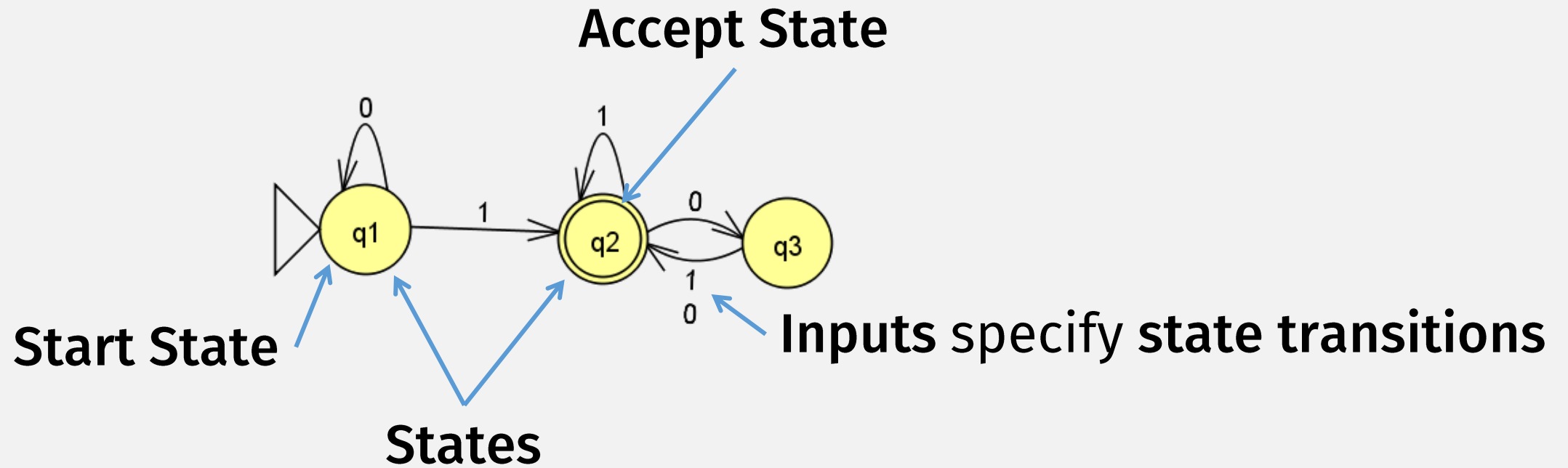
Model-view-controller (MVC) is an FSM



A Finite Automata is a “Program”

- A very limited “program” that uses finite memory
 - Actually, only 1 “cell” of memory!
 - States = the possible things that can be written to memory
- Finite Automata has different representations:
 - Code (wont use in this class)
 - State diagrams

Finite Automata state diagram



A Finite Automata = a “Program”

- A very limited program with finite memory
 - Actually, only 1 “cell” of memory!
 - States = the possible things that can be written to memory
- Finite Automata has different representations:
 - Code
 - State diagrams
 - Formal mathematical description (essentially like code)

Finite Automata: The Formal Definition

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \longrightarrow Q$ 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

This semester

Things in **bold** are precise formal definitions.

(remember them because they will appear frequently in hw, etc)

Analogy

This is the “programming language” definition for finite automata “programs”

Interlude: Sets and Sequences

- Both are: mathematical objects that group other objects
- **Members** of the group are called **elements**
- Can be: **empty**, finite, or infinite
- Can contain: **other sets or sequences**

Sets

- Unordered
- Duplicates not allowed
- Common notation: { }
- **Empty set** denoted: \emptyset or { }
- A **language** is a (possibly infinite) set of strings

Sequences

- Ordered
- Duplicates ok
- Common notation: (), or just commas
- **Empty sequence:** ()
- A **tuple** is a finite sequence
- A **string** is a finite sequence of characters

Set or Sequence ?

A **function** is ...

... a **set** of **pairs**
(1st of each pair from **domain**, 2nd from **range**)

... can write it in many ways: as a mapping, a table, ...

DEFINITION

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

set

1. Q is a finite set called the *states*,

**Set of pairs
(domain)**

2. Σ is a finite set called the *alphabet*,

set

3. $\delta: Q \times \Sigma \rightarrow Q$ is the *transition function*,

4. $q_0 \in Q$ is the *start state*, and

Set (range)

5. $F \subseteq Q$ is the *set of accept states*.

Don't know!
(states can be anything)

set

A **pair** is ... a **sequence** of 2 elements

Finite Automata: The Formal Definition

DEFINITION

5 components

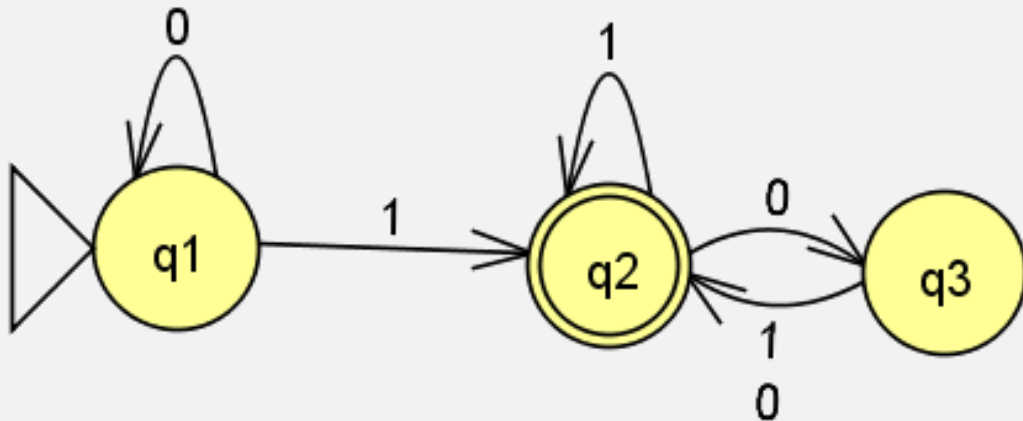
A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.

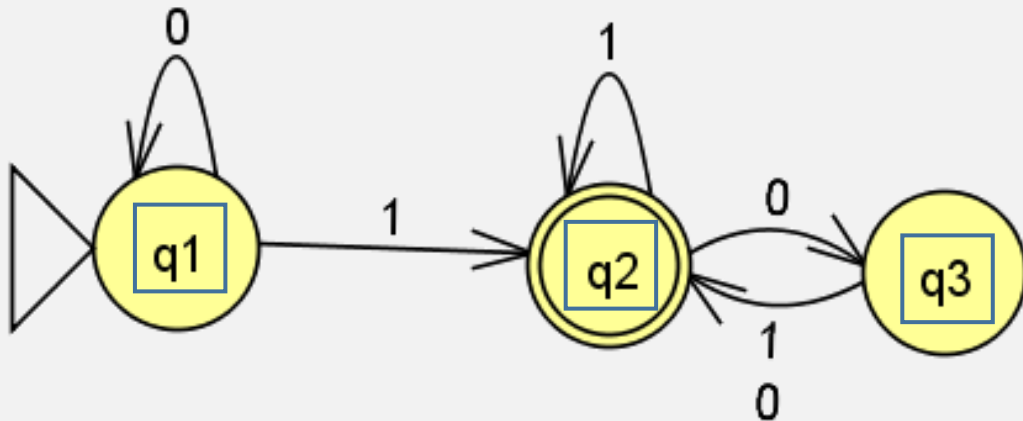
Note:
Not the same Q

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. δ is described as

braces =
set notation
(no duplicates)



Example: as state diagram

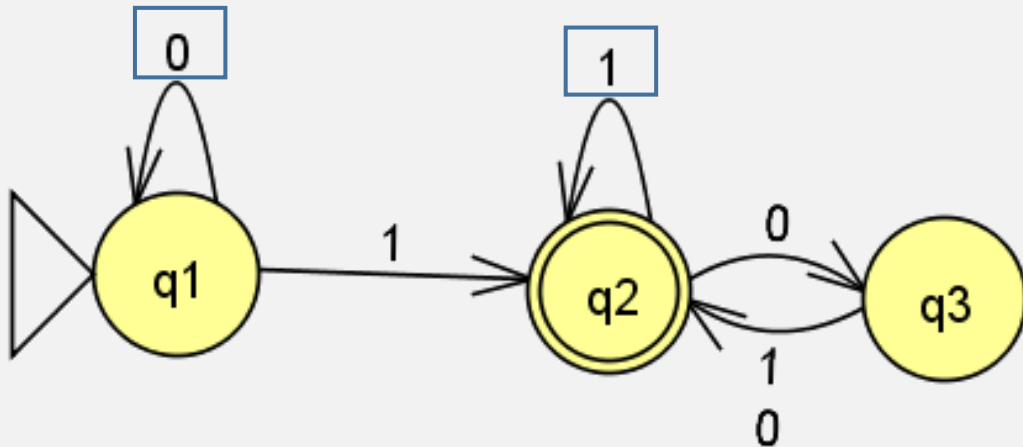
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

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

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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

Example: as formal description

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

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

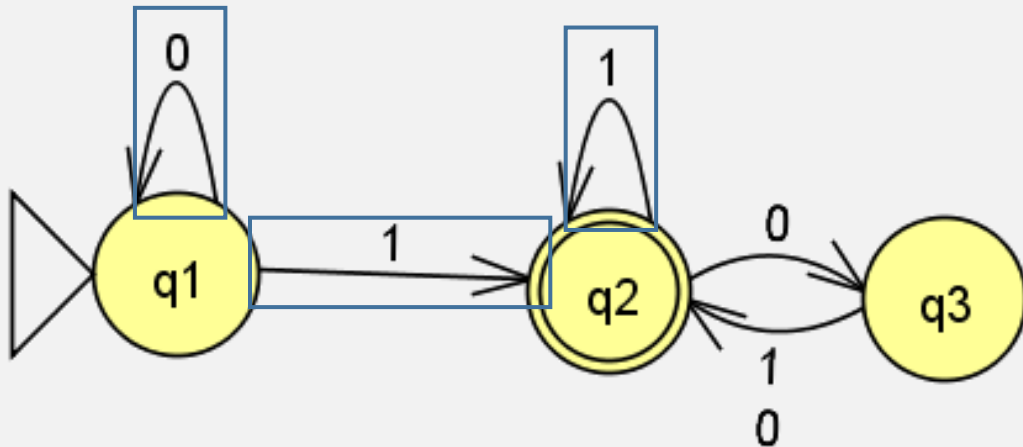
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

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

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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

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. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

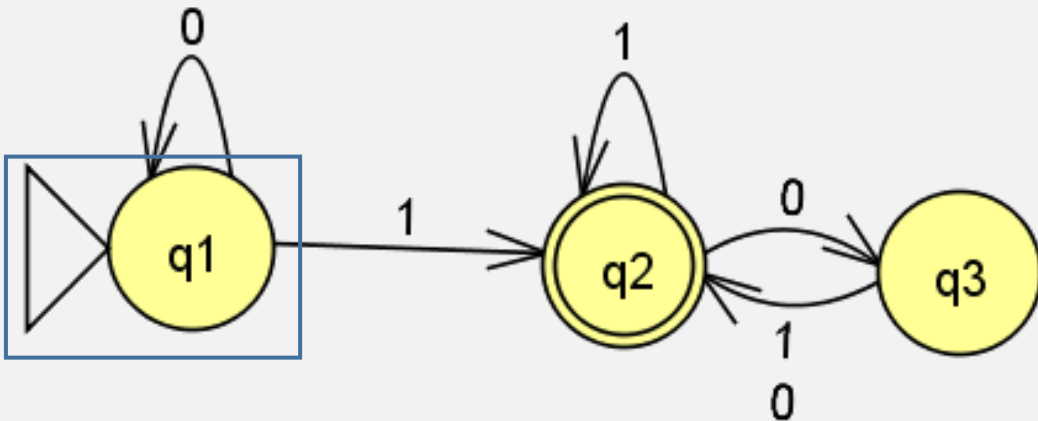
“If in this state” → q_2 (row header)
“And this is next input symbol” → 1 (column header)
“Then go to this state” → q_2 (cell content)

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

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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

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. δ is described as

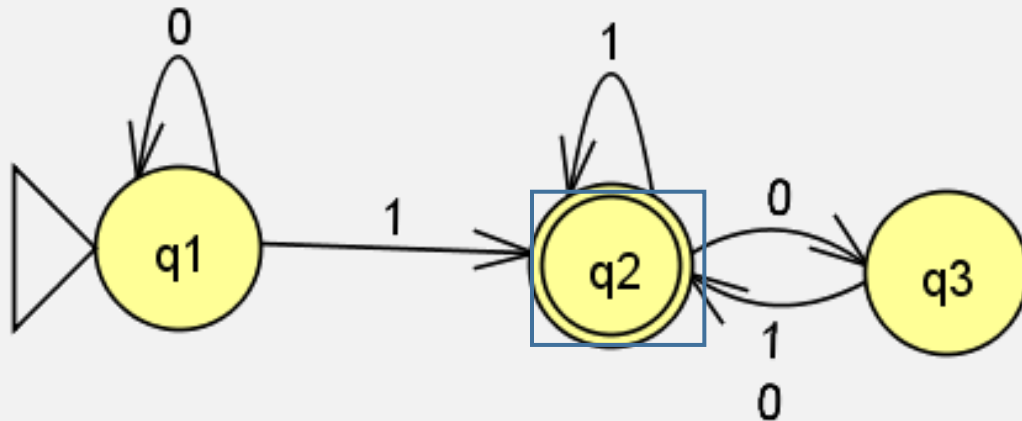
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

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

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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

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. δ is described as

	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, and

5. $F = \{q_2\}$.

DEFINITION

A *finite automaton* is a 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. $\delta: Q \times \Sigma \rightarrow Q$ 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 “Programming Language”

This analogy is a way to help your intuition

But don't confuse with **formal definitions**.

Programming Analogy

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. δ is described as

	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, and
5. $F = \{q_2\}$.