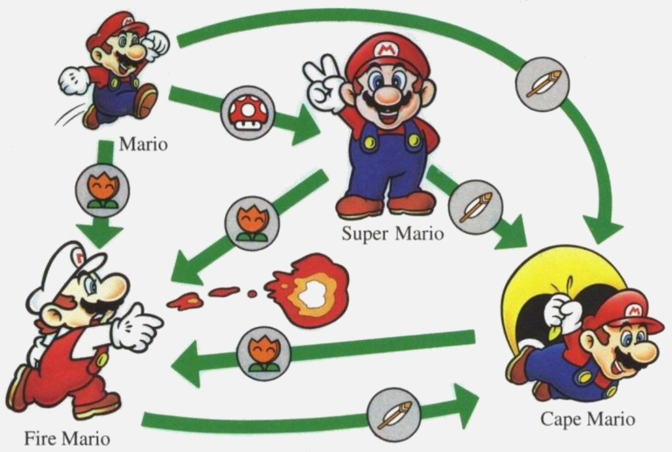
CS420

(Deterministic) Finite Automata

Thursday, September 8, 2022

UMass Boston Computer Science



Announcements

- **Quizzes**
  - Quiz 1 returned
  - Use gradescope issue ticket for questions / complaints
- **HW**
  - Weekly; in/out Sunday midnight
  - <u>HW 0 due Sunday 9/11 11:59pm EST</u>
  - ~4-5 questions, Paper-and-pencil proofs (no programming)
  - Discussing with classmates ok; Final answers written up / submitted individually
- **Lectures**
  - Not recorded but closely follow the listed textbook chapters
- **Office Hours**
  - Thurs 12:30-2pm (in person)
  - Fri 4-5:30pm (zoom, access link from blackboard)
  - Let me know if advance if possible, but drop-ins also fine

Last Time: The Theory of Computation …

Formally defines mathematical models of computation

In order to:

1. Make predictions (about computer programs)
   - If possible

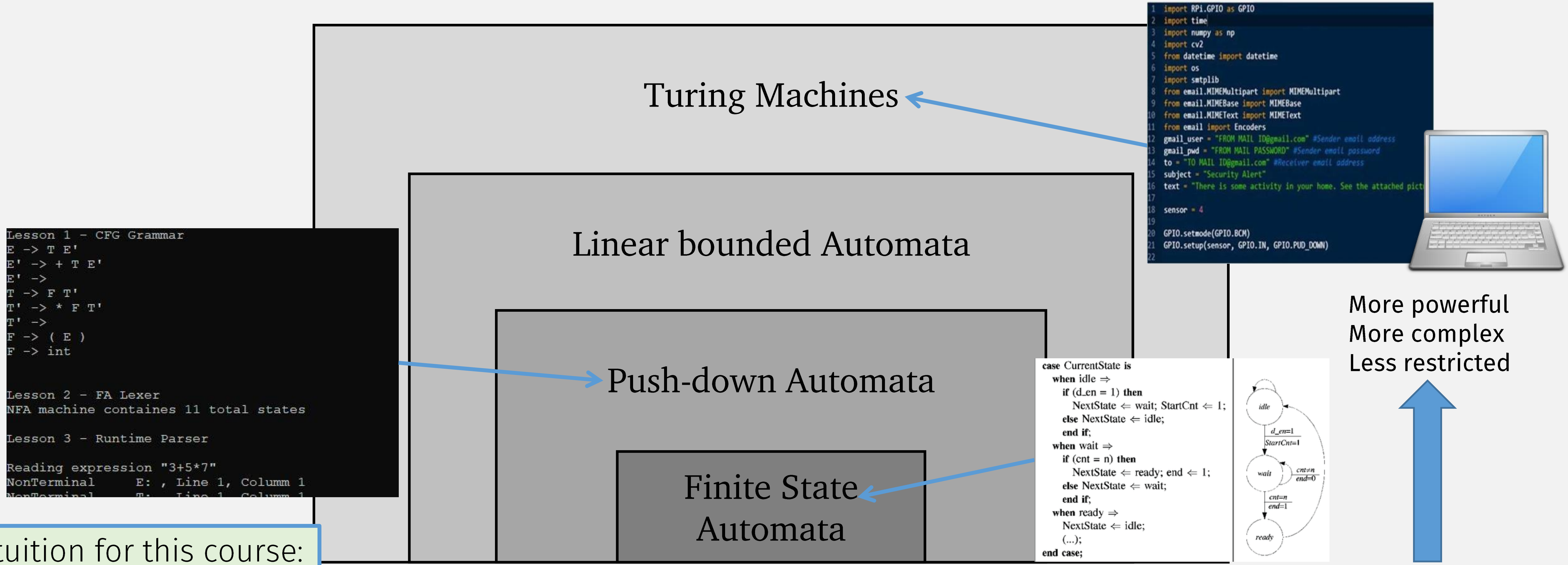
```
function( x, y, z, n) {
  if n > 2 && x^n + y^n == z^n {
    printf("hello, world!\n");
  }
}
```

Fermat's Last Theorem
(unknown for ~350 years, solved in 1990s)

2. Compare the models to each other
   - `Java` vs `Python`? The same?

3. Explore the limits of computation
   - What programs cannot be written?

Last Time: Computation = Programs!

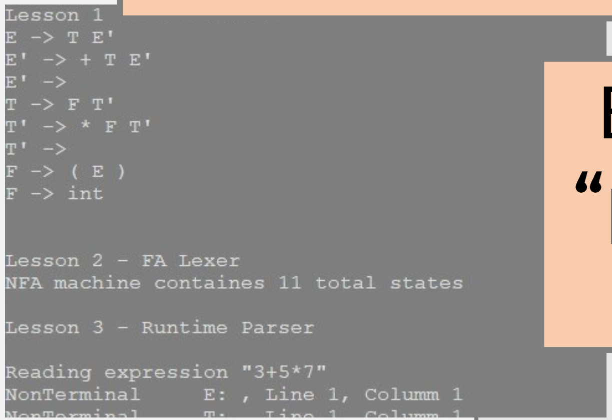


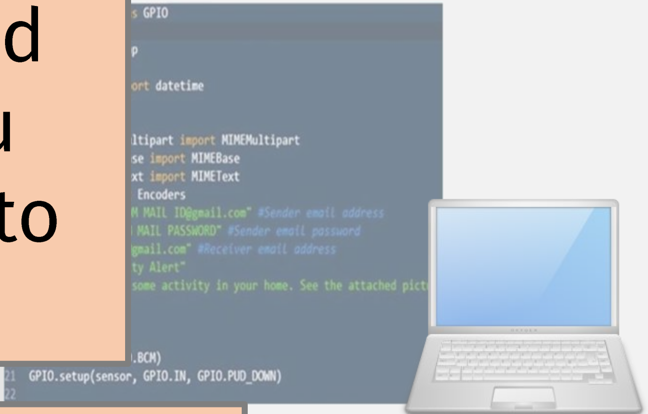
Intuition for this course:

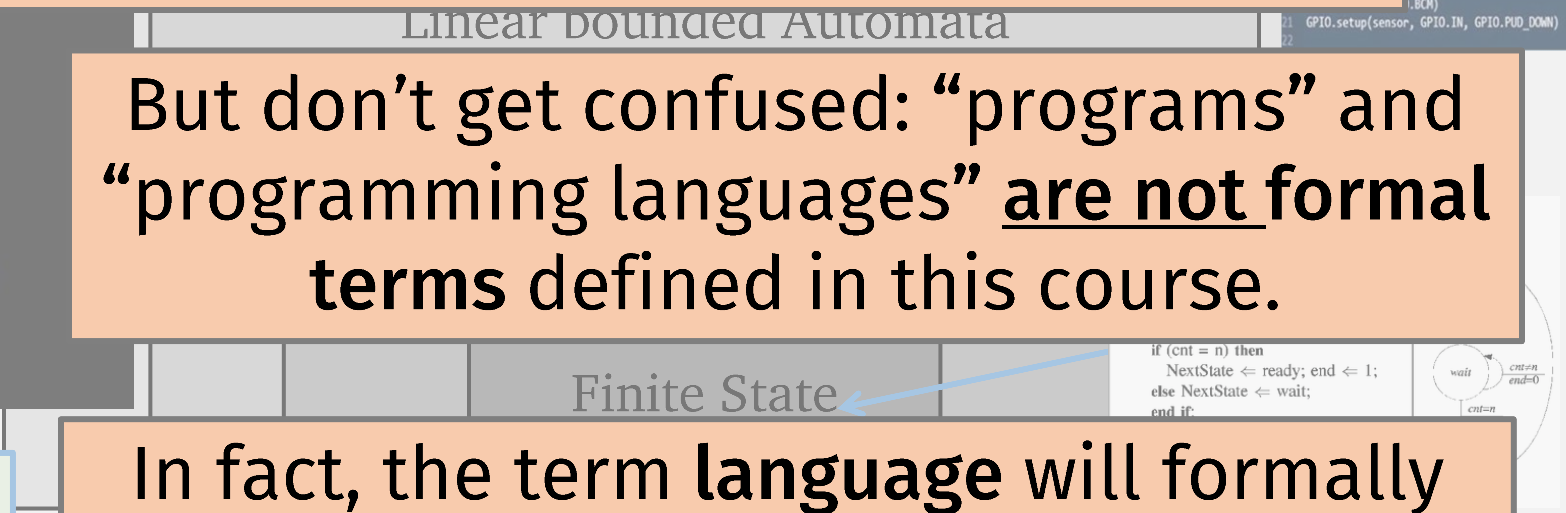
- A **model of computation** defines a **class of machines** (each box)
- Think of: a **class** of machines = a **"Programming Language"**!
- Think of: a **single** machine instance = a **"Program"**!

Last Time: Computation = Programs!

<u>Very important Note</u>: I use this “programs” and “programming language” <u>analogy</u> to help you understand CS420 topics, by comparing them to ideas you’ve seen before







But don’t get confused: “programs” and “programming languages” **<u>are not formal</u> terms** defined in this course.

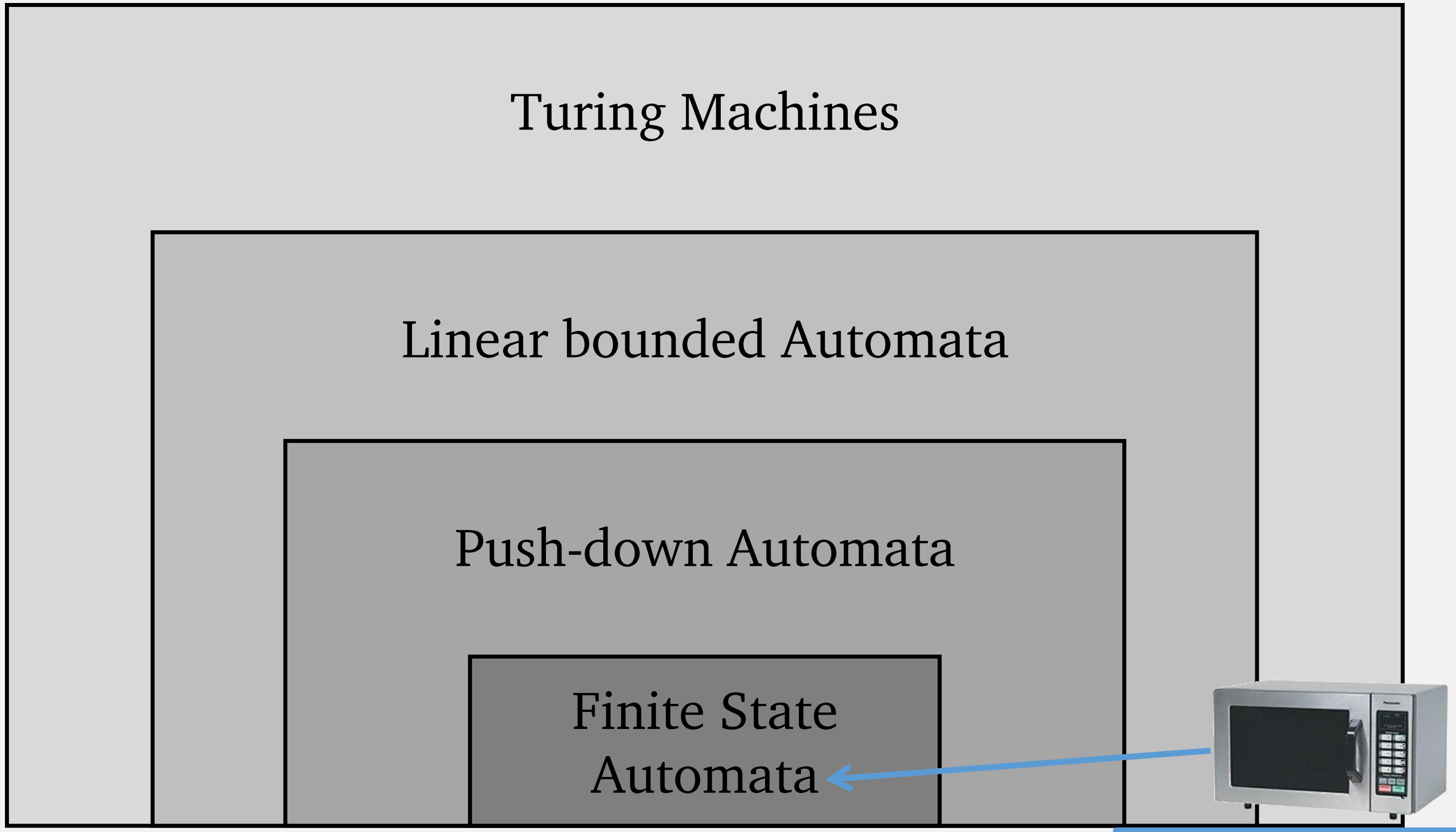
More powerful
More complex
Less restricted

In fact, the term **language** will formally mean something else (later)

Intuition for this course:

- **A model of compu**
- Think of: a **class** of machines = a “**Programming Language**”!
- Think of: a **single** machine instance = a “**Program**”!

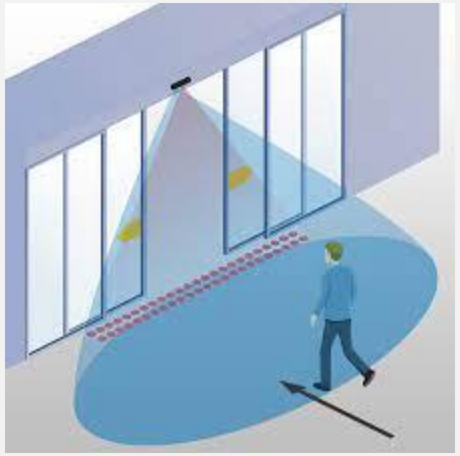
Last Time: Models of Computation Hierarchy



We’ll start here …

More powerful
More complex
Less restricted

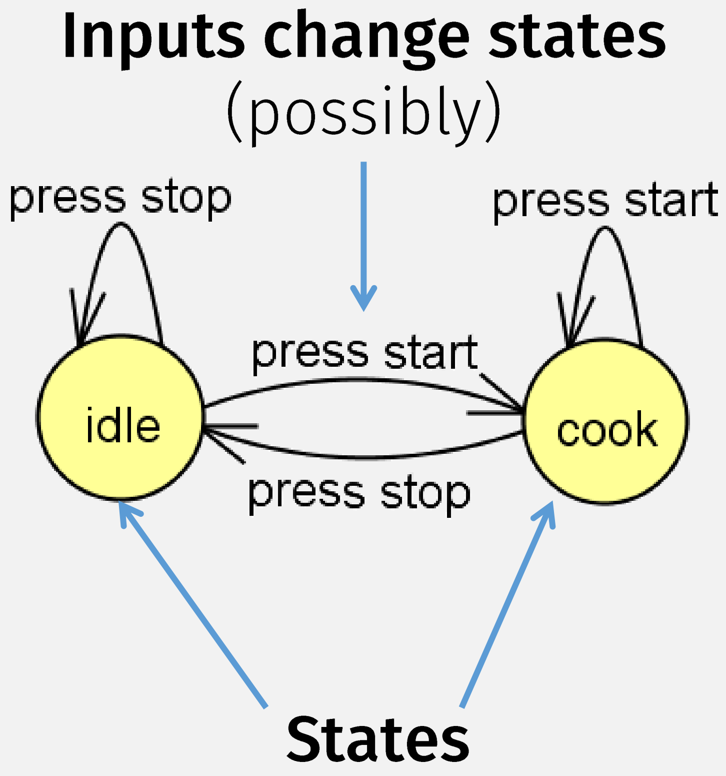
Finite Automata: “Simple” Computation / “Programs”



Finite Automata

- A **finite automata** or **finite state machine (FSM) …**
- … **computes** with a <u>finite</u> **number of states**

A Microwave Finite Automata



Finite Automata: Not Just for Microwaves

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.

Finite Automata:
a common
programming pattern

Computation Simulating Other Computation
(a common theme this semester)

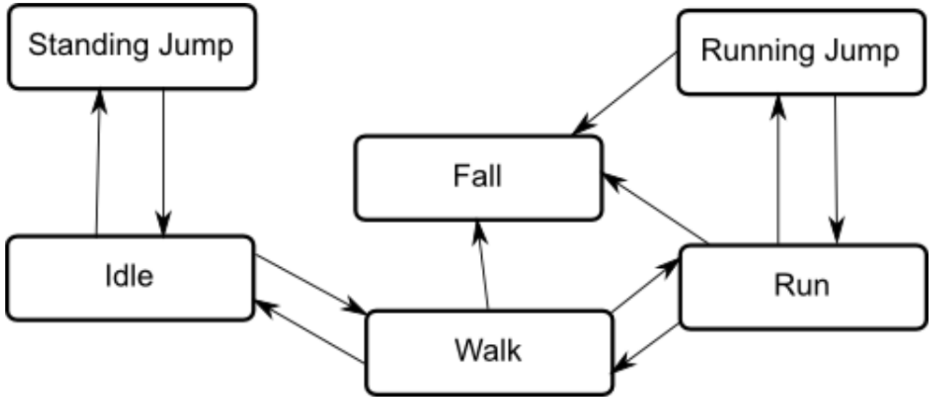
Video Games Love Finite Automata

Unity Documentation — Manual

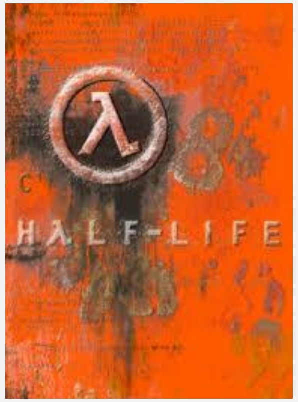
Unity User Manual 2020.3 (LTS) / Animation / Animator Controllers / Animation State Machines / State Machine Basics

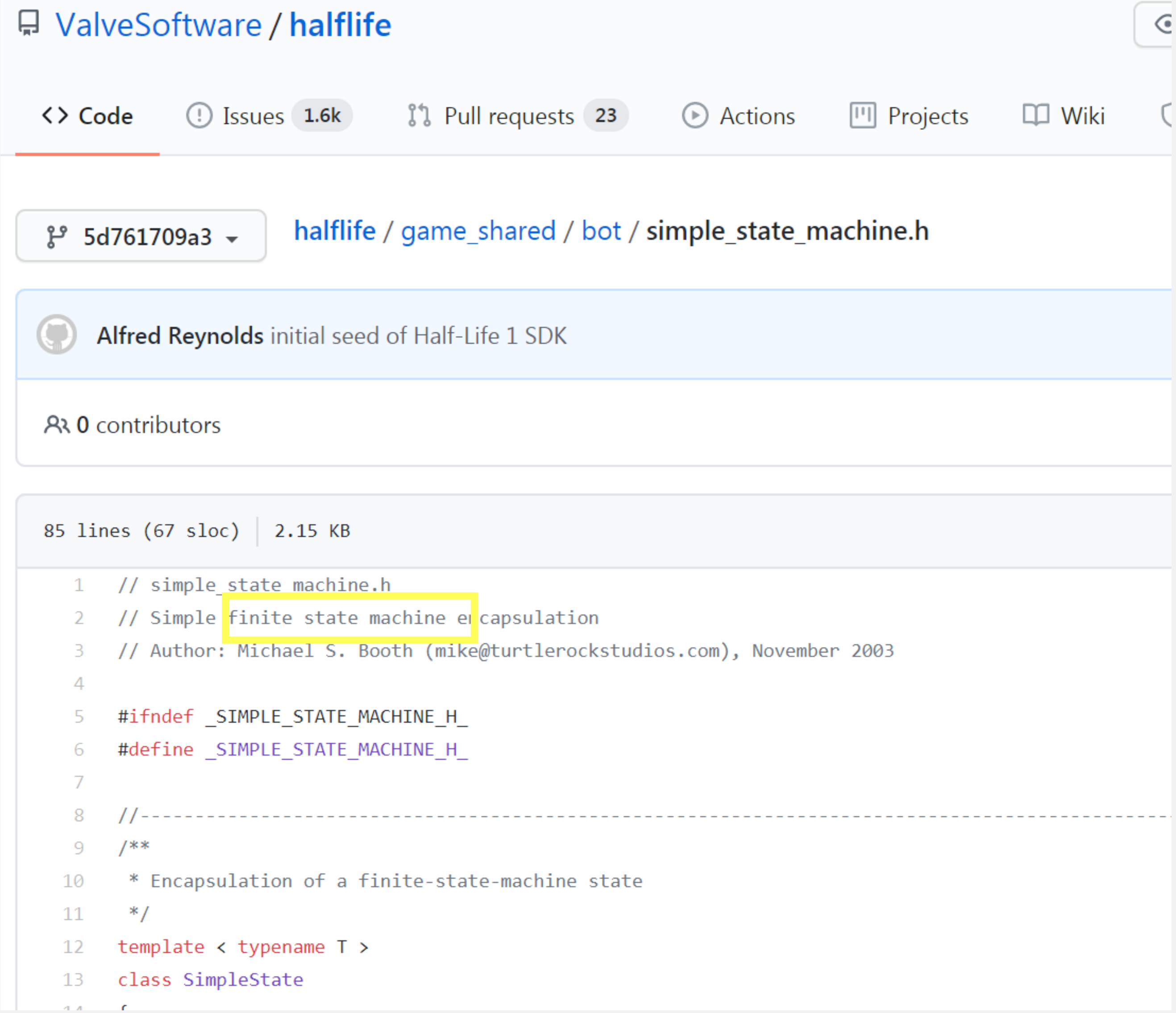
The basic idea is that a character is engaged in some particular kind of action at any given time. The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc. These actions are referred to as **states**, in the sense that the 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



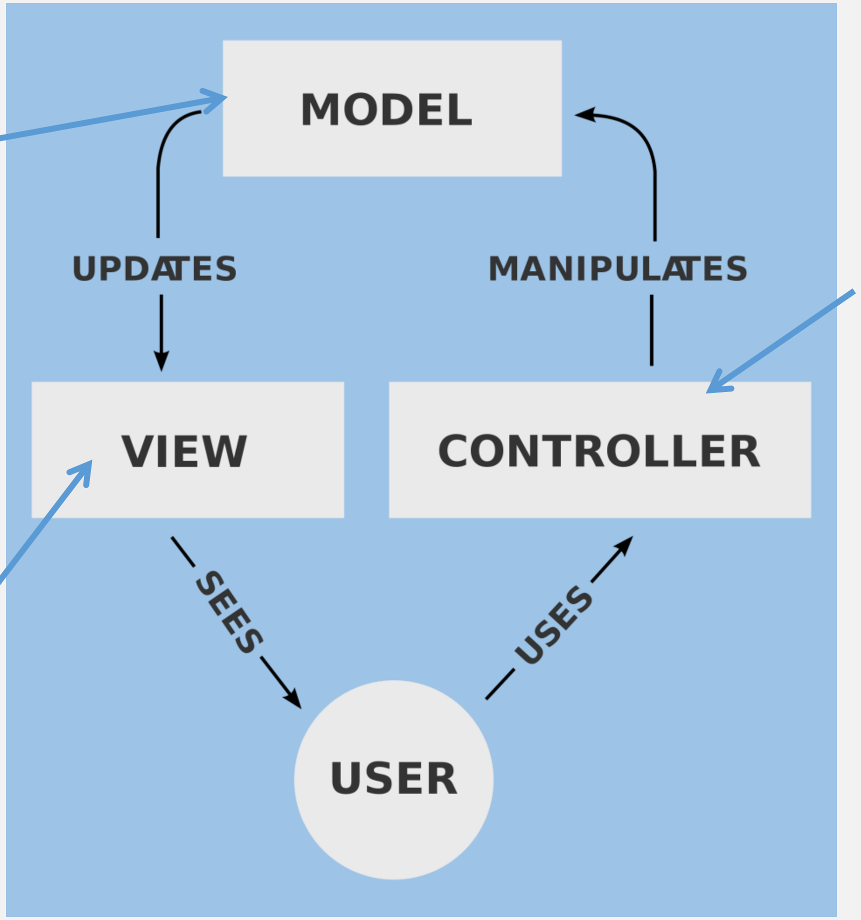


Model-view-controller (MVC) is an FSM

States

Inputs change states

The View draws states

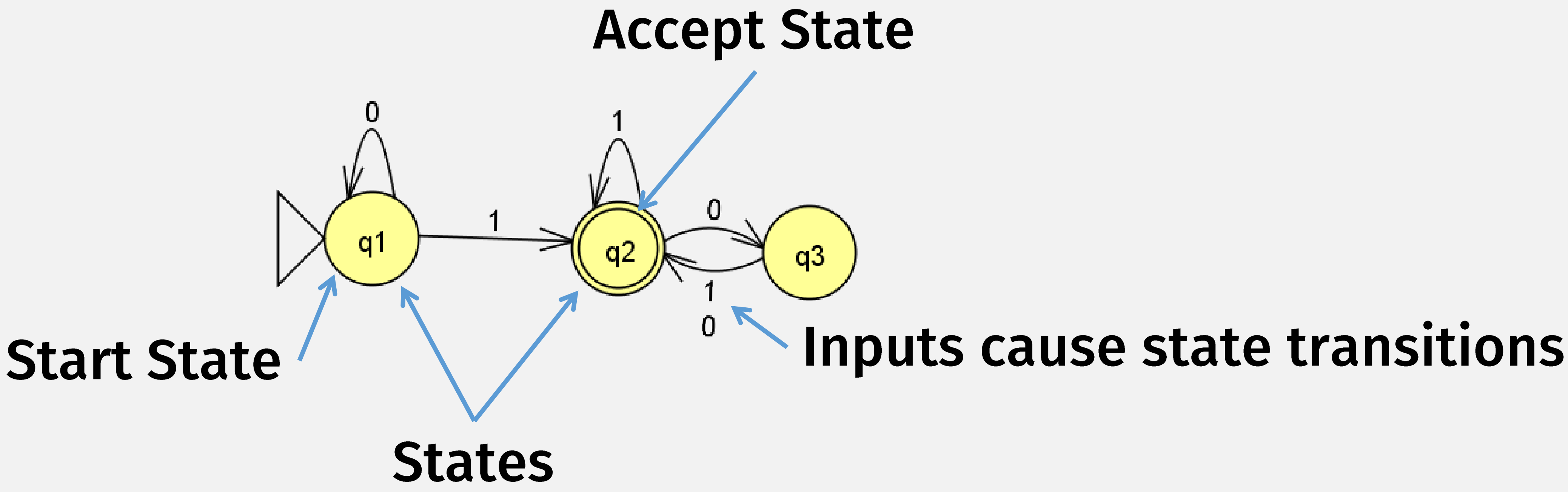


A Finite Automata = a "Program"

- 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

Finite Automata: The Formal Definition

5 components

DEFINITION

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

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

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	**Sequences**
• **Un**ordered	• Ordered
• Duplicates **not** allowed	• Duplicates ok
• Common notation: { }	• Common notation: ( ), or **just commas**
• “Empty set” denoted: Ø or { }	• “Empty sequence”: ( )
• A **language** is a (possibly infinite) set of strings	• 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 <u>mapping</u>, a <u>table</u>, …

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***.

sequence

set

set

Set of pairs (**domain**)

Set (**range**)

Don't know! (states can be anything)

set

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

Finite Automata: The Formal Definition

5 components

DEFINITION

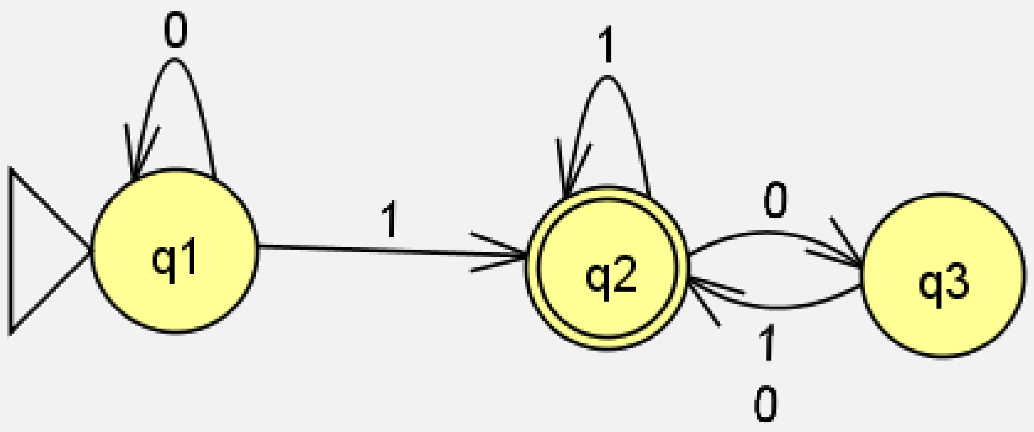
A ***finite automaton*** is a 5-tuple $\boxed{(Q, \Sigma, \delta, q_0, F)}$, where

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



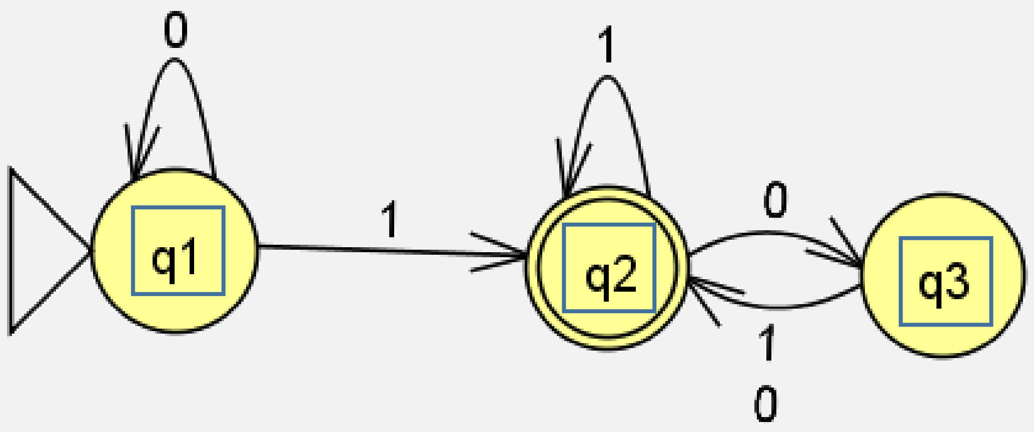
Example: as <u>state diagram</u>

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Note:
Not the same Q



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,

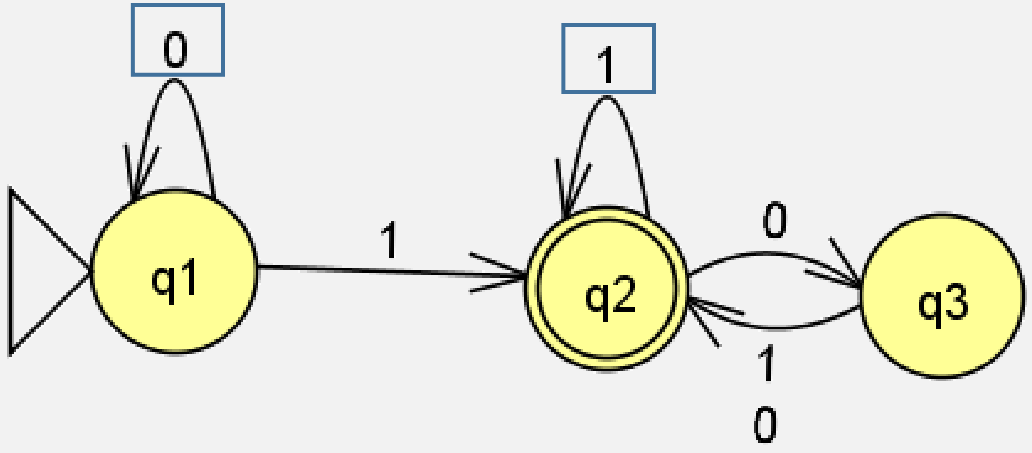
4. q_1 is the start state, and
5. $F = \{q_2\}$.

braces =
set notation
(no duplicates)

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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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

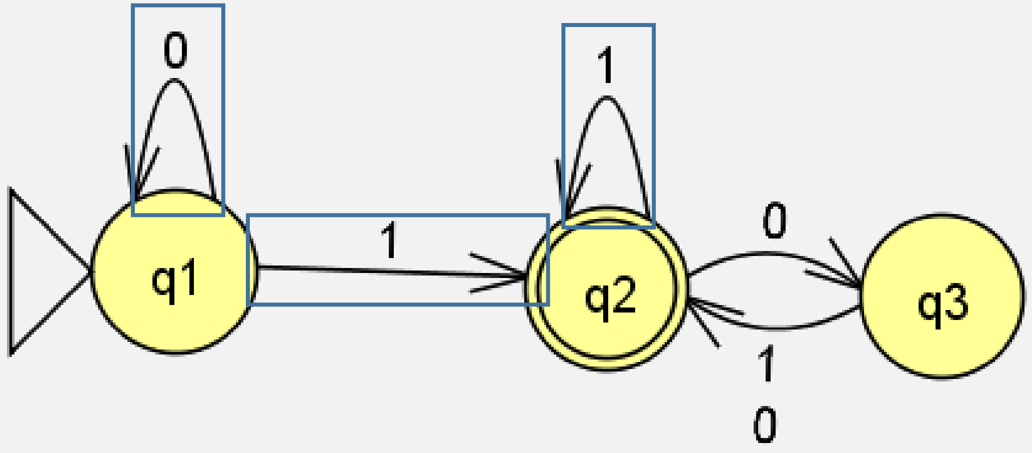
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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



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,

"And this is next input symbol"

"If in this state"

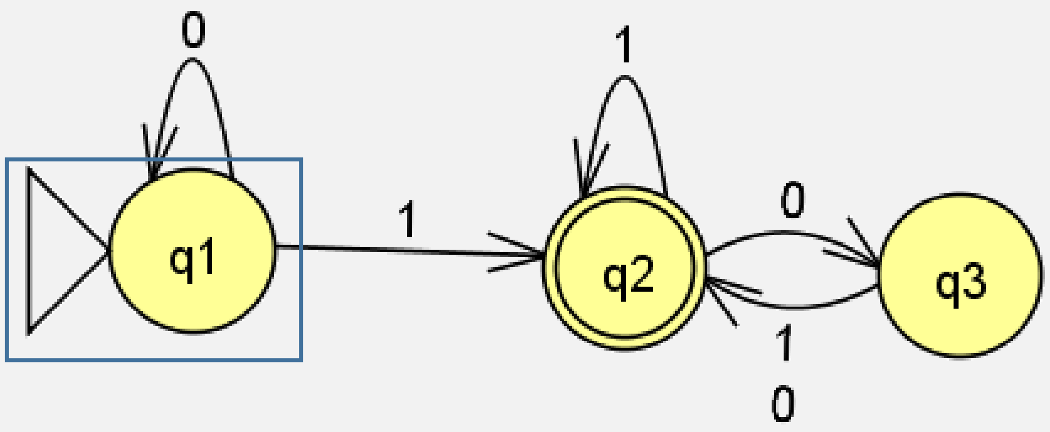
"Then go to this state"

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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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

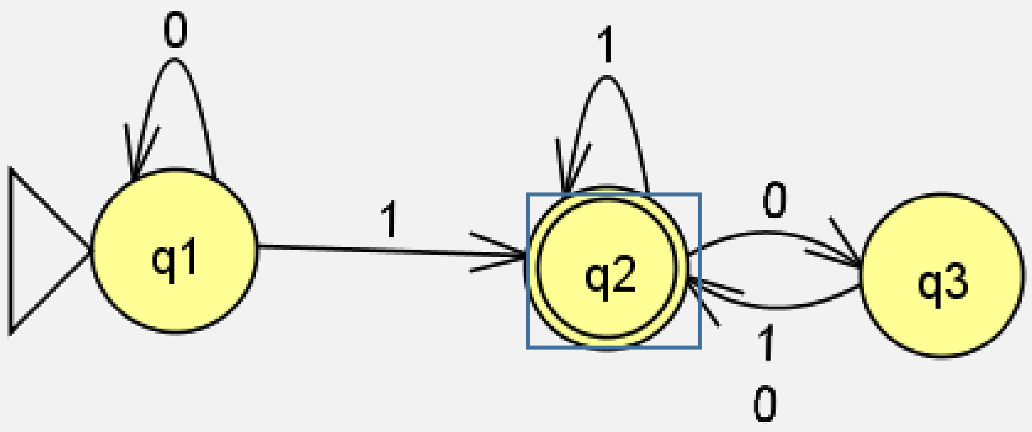
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

A "Programming Language"

Remember: **this is just way to help your intuition**

But **these are not formal terms.**
Don't get confused

Programming Analogy

Example: **as formal description**

A "Program"

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

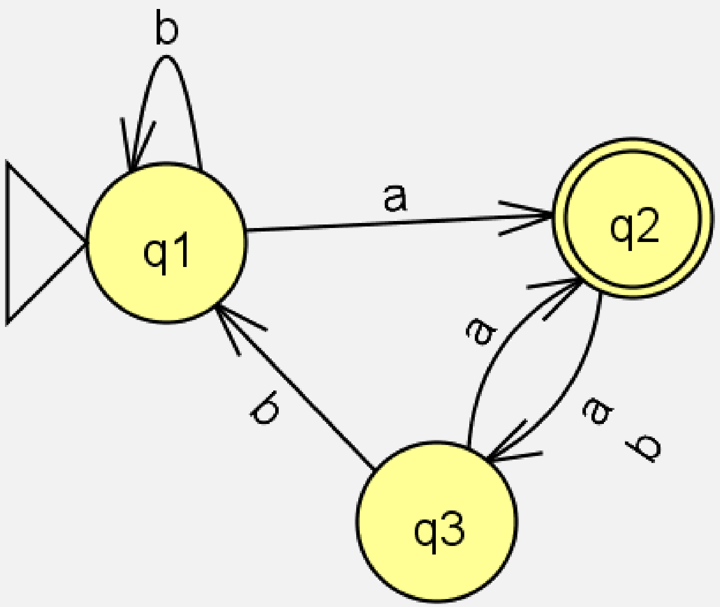
1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	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. $F = \{q_2\}$.

In-class Exercise

Come up with a formal description of the following machine:



DEFINITION

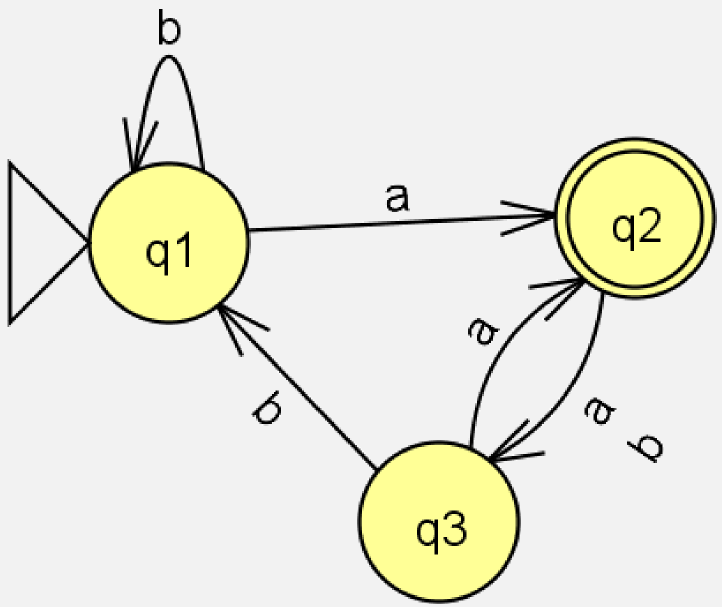
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

In-class Exercise: solution

- Q = {q1, q2, q3}
- Σ= { **a**, **b** }
- δ
  - δ(q1, **a**) = q2
  - δ(q1, **b**) = q1
  - δ(q2, **a**) = q3
  - δ(q2, **b**) = q3
  - δ(q3, **a**) = q2
  - δ(q3, **b**) = q1
- q_0 = q1
- F = {q2}

$$M = (Q, \Sigma, \delta, q_0, F)$$



A Computation Model is … (from lecture 1)

- Some **base definitions and axioms …**

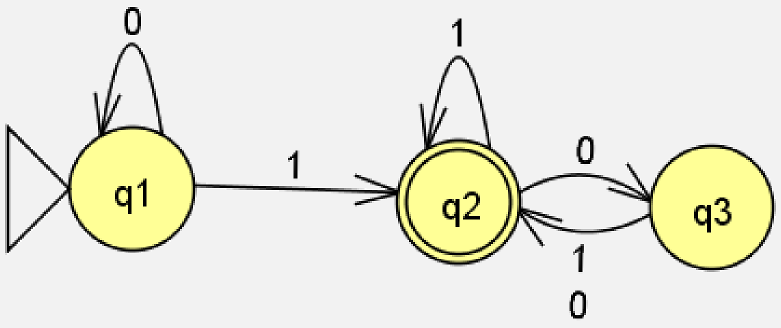
DEFINITION

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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

- And **rules that use the definitions …**

Computation with FSMs (JFLAP demo)

- **FSM:** 

- **Input:** “`1101`”

FSM Computation Model

Informally

- Program = a finite automata
- Input = string of chars, e.g. "`1101`"

To run a program:

- Start in "start state"
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - "**Accept**" if last state is "Accept" state
  - "**Reject**" otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$
- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}, \text{ for } i = 0, \ldots, n-1$

Let's come up with **nicer notation** to represent this part

- M ***accepts*** w if

  sequence of states $r_0, r_1, \ldots, r_n$ in Q exists ...

  with $r_n \in F$

Still a little verbose

Check-in Quiz 9/8

On gradescope