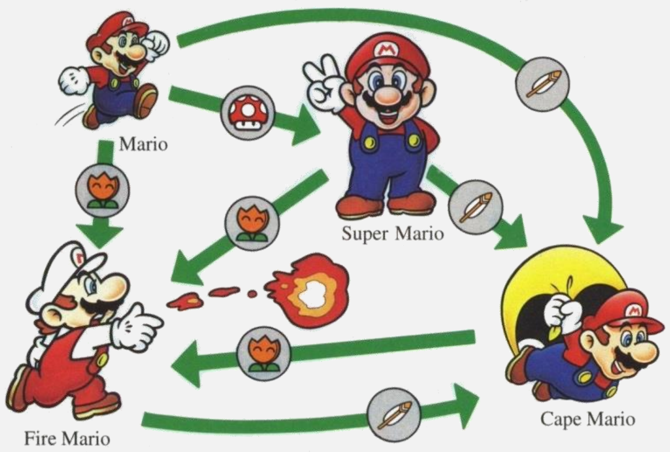
CS420

(Deterministic) Finite Automata

Wednesday, January 25, 2023

UMass Boston Computer Science



Announcements

- **Quizzes**
  - 15 min limit
  - 1/23 quiz "graded"
  - Use gradescope issue ticket for questions / complaints
- **HW**
  - Weekly; in/out Sun midnight
  - ~4-5 questions, Paper-and-pencil proofs (no programming)
  - Discussing with classmates ok; Final answers written up / submitted individually
  - HW 0 extended: due Tues 1/31 11:59pm EST
- **Lectures**
  - Slides posted
  - Closely follow the listed textbook chapters
  - Might be recorded?
- **Office Hours**
  - Wed 12:30-2pm (in person, McCormack 3rd floor, Rm 201)
  - Fri 12:30-2pm (zoom, access link from blackboard)
  - Let me know in advance if possible, but drop-ins also fine
  - TA TBD

Last Time: How Mathematics Works

More Theorems

More Axioms

More Definitions

Theorem

Theorem

Axioms

Definitions

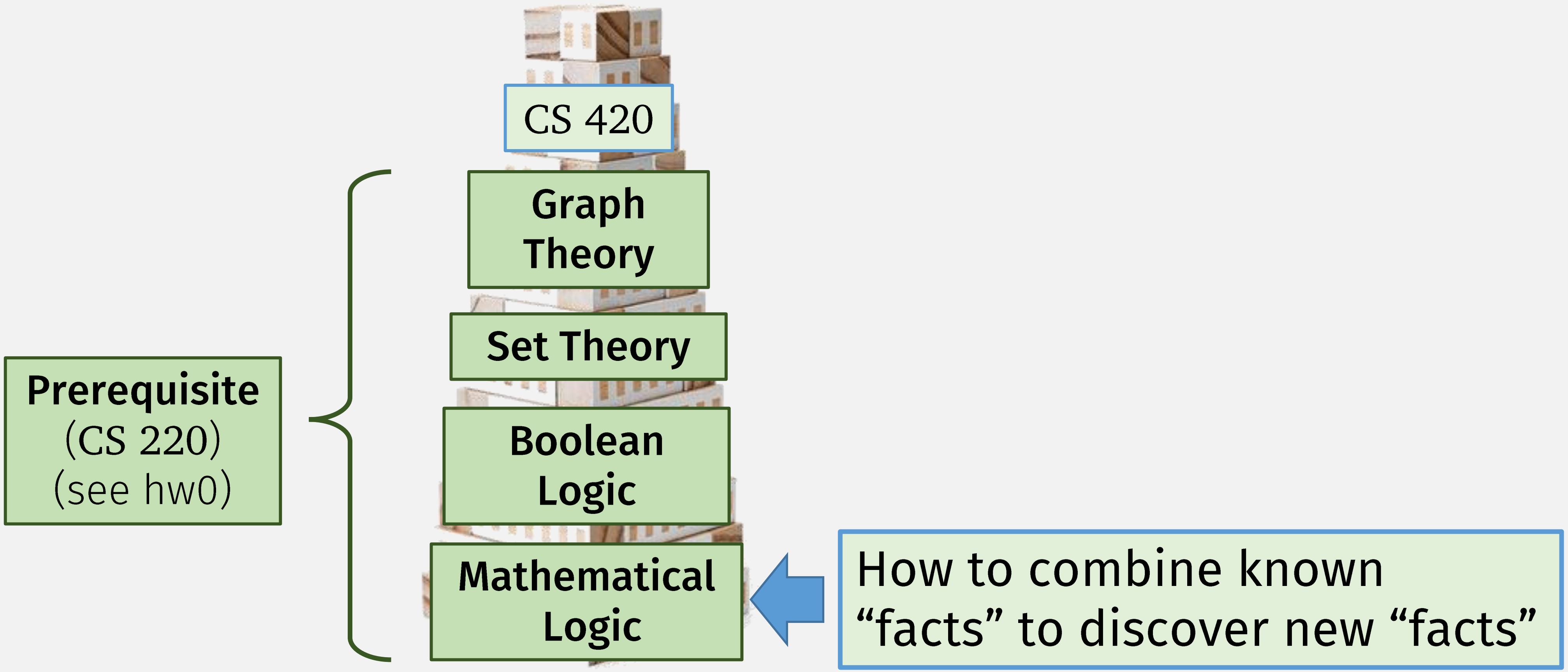
"facts"

Mathematician
(or student)

Actually, **it's not always so easy to create the next level … Preciseness is important**

Proofs = Figuring out how to (precisely) **fit known "facts" together**

How **CS 420** Works



Mathematical Logic Operators

- **Conjunction** (AND, $\land$)
- **Disjunction** (OR, $\lor$)
- **Negation** (NOT, $\neg$)
- **Implication** (IF-THEN, $\Rightarrow$, $\rightarrow$)

…

Mathematical Statements: $\mathbf{AND}$

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 P is TRUE, then Q is TRUE (**modus ponens**)

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

Proving:

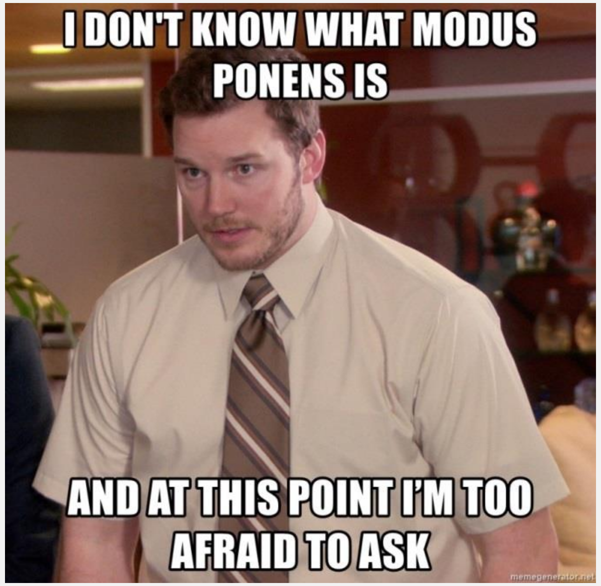
<u>Using</u> an **IF-THEN** statement:
The "Modus Ponens" Inference Rule

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

Proving:

- To prove $P \to Q$ is TRUE:
  - If P is FALSE, statement is always TRUE
  - Assume P is TRUE, then prove Q is TRUE

p	q	$p \to q$
True	True	True
True	False	False
False	True	True
False	False	True

Last Time: Deductive Proof Example

Prove the following:

Proving

- If: If $x \geq 4$, then $2^x \geq x^2$ ← Assume these are true

Using

- And: x is the sum of the squares of four positive integers

- Then: $2^x \geq x^2$ ← Prove this is true

Statements / Justifications Table

Last Time: Deductive Proof Example

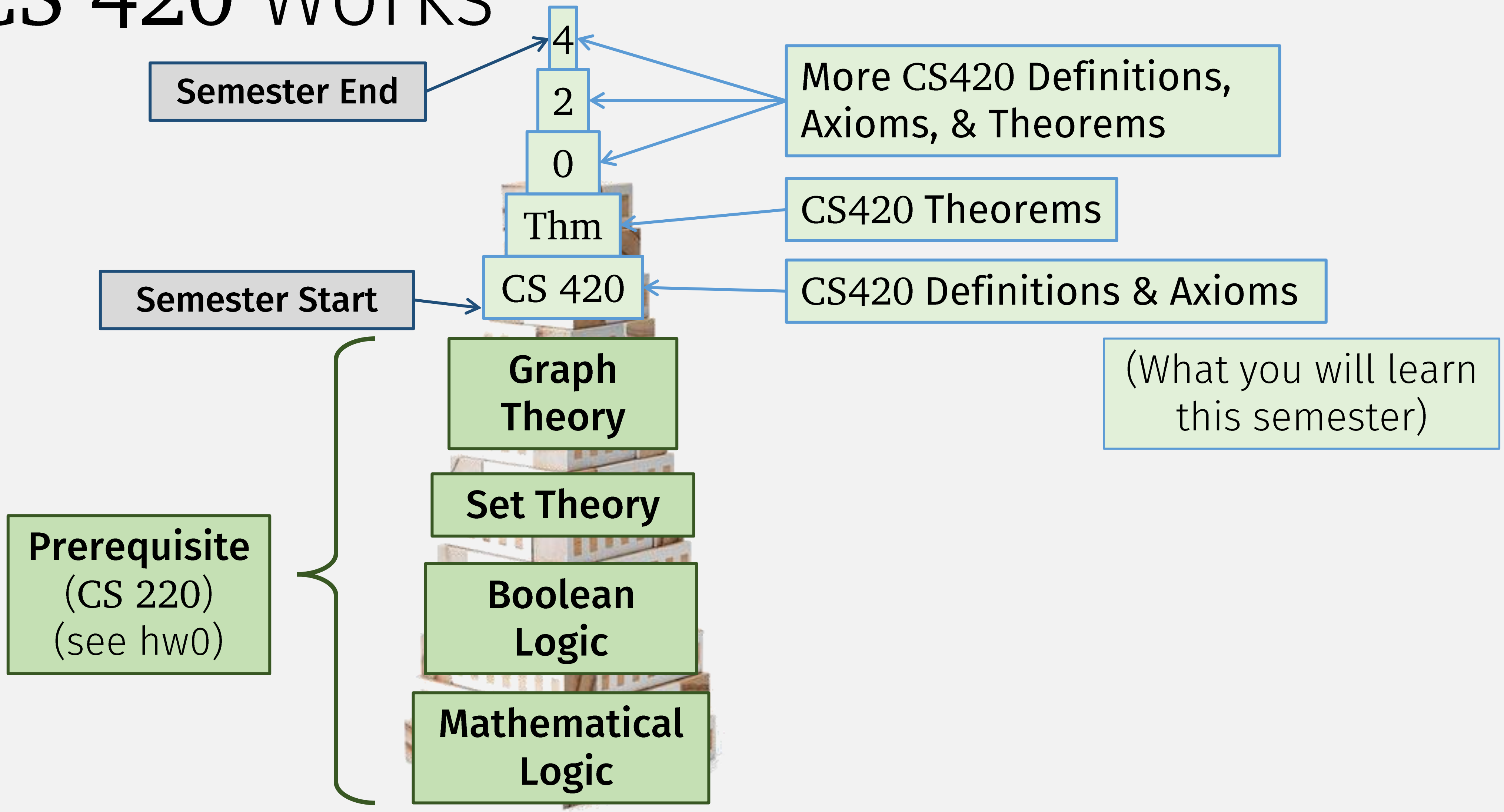
Prove: If If $x \geq 4$, then $2^x \geq x^2$ and x is the sum of the squares of four positive integers then $2^x \geq x^2$

Proof:

Statement	Justification
1. $x = a^2 + b^2 + c^2 + d^2$	1. Assumption
2. $a \geq 1;\ b \geq 1;\ c \geq 1;\ d \geq 1$	2. Assumption
3. $a^2 \geq 1;\ b^2 \geq 1;\ c^2 \geq 1;\ d^2 \geq 1$	3. By Stmt #2 & arithmetic laws
4. $x \geq 4$	4. Stmts #1, #3, and arithmetic
5. If $x \geq 4$, then $2^x \geq x^2$	5. Assumption
6. $2^x \geq x^2$	6. Stmts #4 and #5 Modus ponens

■

How **CS 420** Works



A Word of Advice

To **prove** a (new) **theorem** ...

... need to know all **axioms, definitions**, and (previous) **theorems** below it

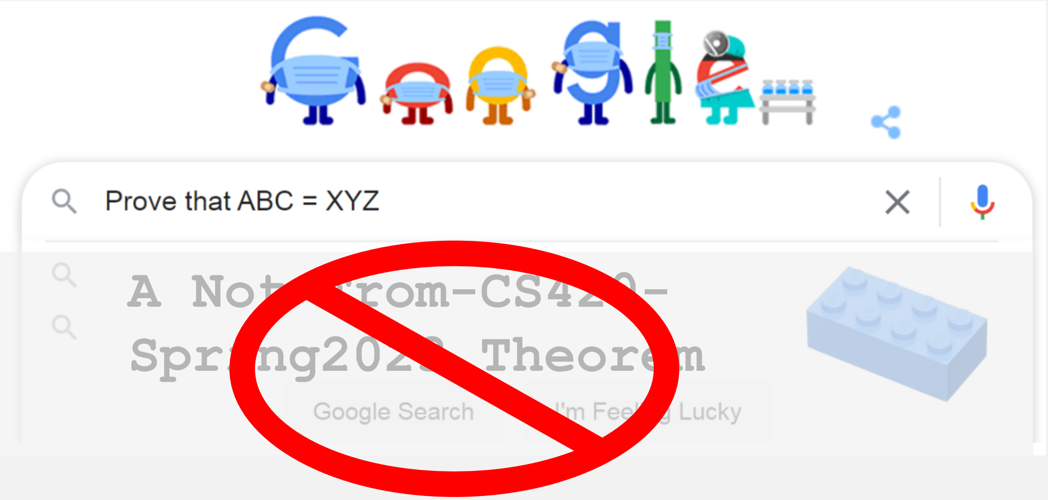
Important:
Do not fall behind
in this course

More Advice

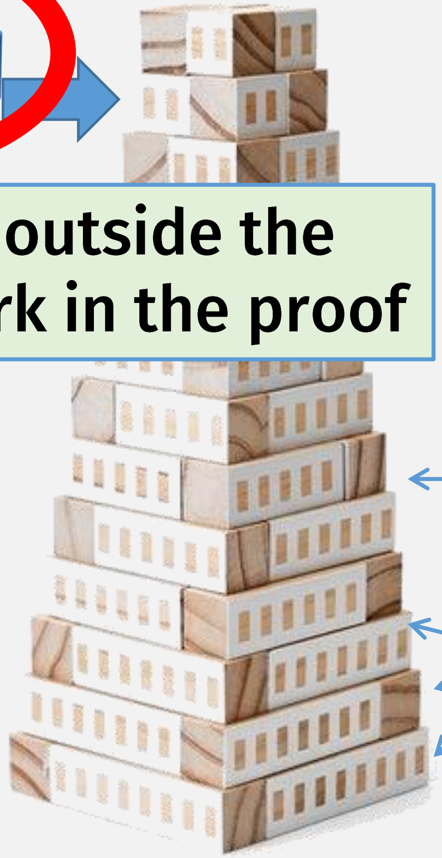
"Answer Hunting" won't work in CS420

HW 1, Problem 1

Prove that $ABC = XYZ$



"Blocks" from outside the course won't work in the proof



Hw is *graded* on your understanding of <u>how</u> to get to the answer, <u>not</u> the final answer itself!

... can be used to **prove** (new) **theorems** in <u>this course</u>

<u>Only</u> **axioms, definitions**, and **theorems** from <u>this course</u> ...

Textbooks

- Sipser. *Intro to Theory of Computation*, 3rd ed.

- Hopcroft, Motwani, Ullman. *Intro to Automata Theory, Languages, and Computation*, 3rd ed.

- Recommended but not required,
  - slides and lecture should be self-contained,
- Readings to accompany lectures will be posted

FYI: **Reading the readings correlated with good grades!**

All course info available on web site:
`https://www.cs.umb.edu/~stchang/cs420/s23`

Grading

- **HW**: 80%
  - Weekly: Out Monday, In Sunday
  - Approx. 12 assignments
  - Lowest grade dropped
- **Quizzes**: 5%
  - End of every lecture
  - To help everyone keep up
- **Participation:** 15%
  - Lecture, office hours, piazza
- No exams

- **A** range: 90-100
- **B** range: 80-90
- **C** range: 70-80
- **D** range: 60-70
- **F**: < 60

All course info available on web site:
`https://www.cs.umb.edu/~stchang/cs420/s23`

Late HW

- Is bad … try not to do it please
  - Grades get delayed
  - Can’t discuss solutions
  - You fall behind!

- <u>Late Policy:</u> **3 late days** to use during the semester

HW Collaboration Policy

Allowed

- Discussing HW with classmates (but must cite)
- Using other resources, e.g., youtube, other books, etc.
- Writing up answers on your own, from scratch, in your own words

Not Allowed

- Submitting someone else's answer
- It's still someone else's answer if:
  - variables are changed,
  - words are omitted,
  - or sentences rearranged …
- Using sites like Chegg, CourseHero, Bartleby, Study, etc.
- Can't use theorems or definitions not from this course

Honesty Policy

- 1st offense: zero on problem
- 2nd offense: zero on hw, reported to school
- 3rd offense+: F for course

Regret policy

- If you self-report an honesty violation, you'll only receive a zero on the problem and we move on.

All Up to Date Course Info

Survey, Schedule, Office Hours, HWs, …

See course website:

https://www.cs.umb.edu/~stchang/cs420/s23/

Last Time: The Theory of Computation …

Formally defines mathematical models of computation

In order to:

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

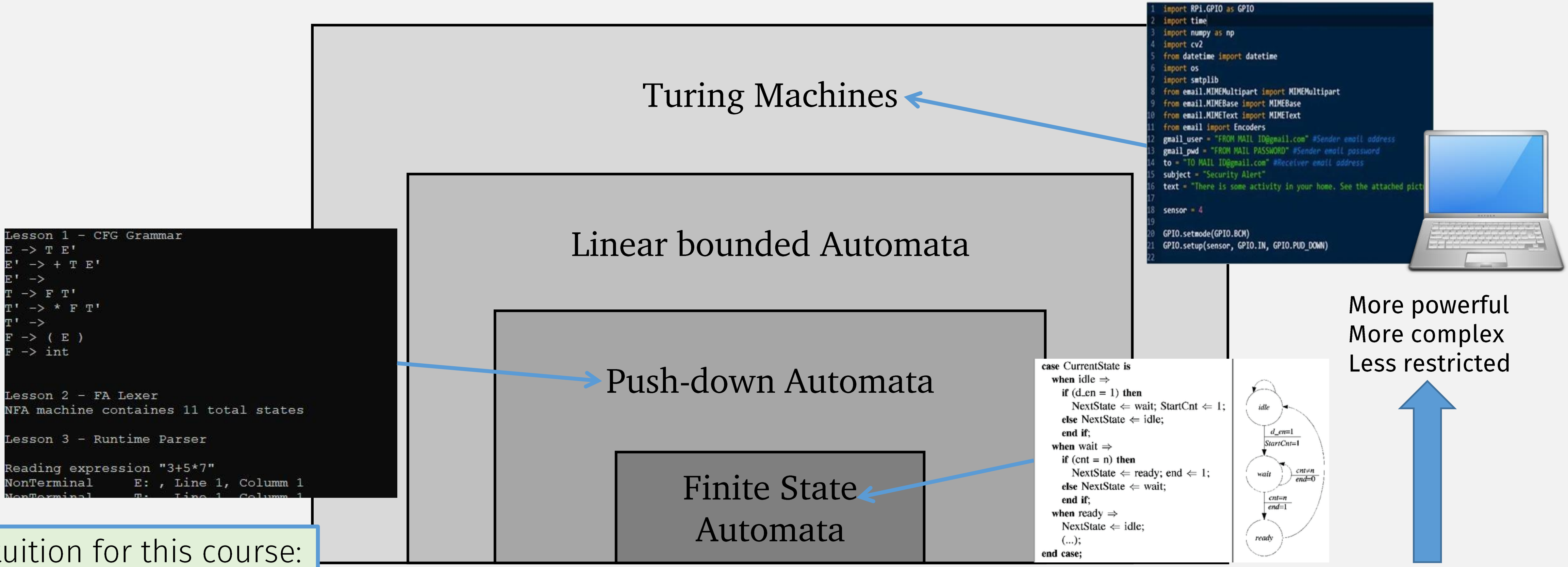
```
function( x, y, z, n) {
  if n > 2 && x^n + y^n == z^n {
    printf("hello, world!\n");
  }
}
```

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!

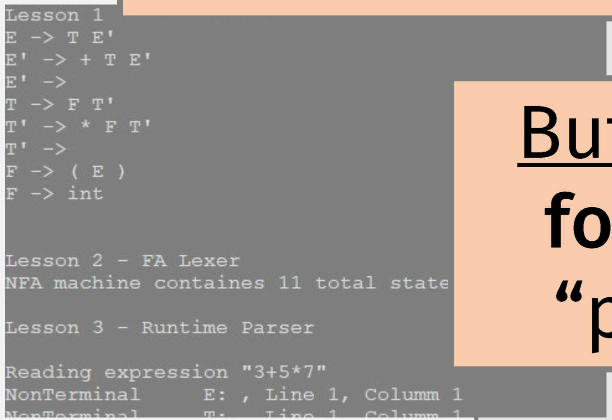


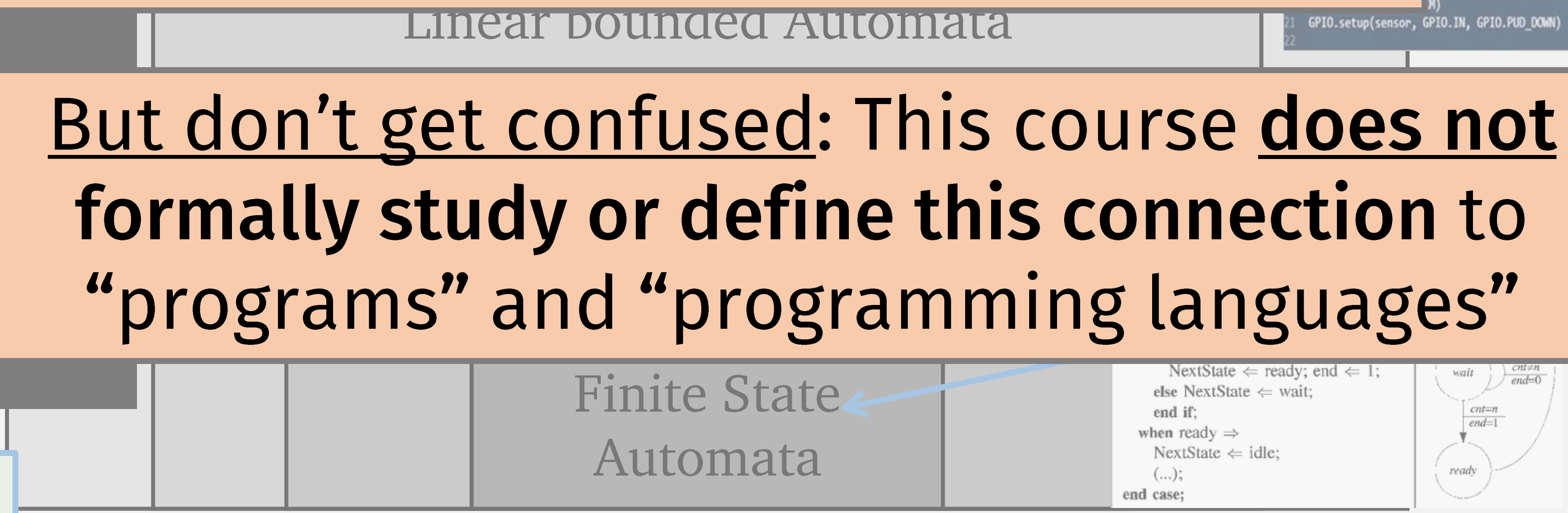
Intuition for this course:

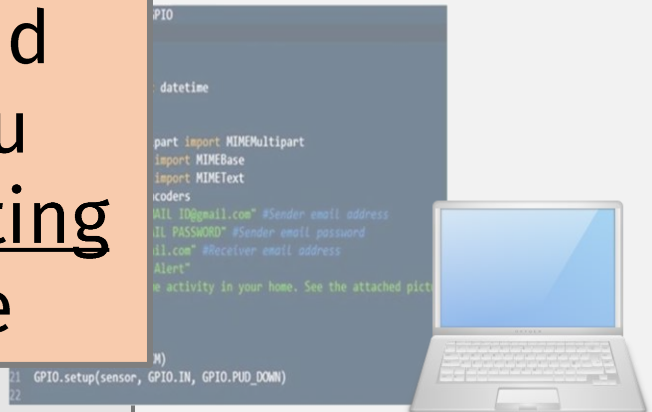
- A **model of computation** defines a **class of machines** (each box)
- Think of: a **class** of machines = a **"Programming Language"**!
- Think of: a **single** machine instance = a **"Program"**!

Last Time: Computation = Programs!

Very important Note: I use the "programs" and "programming language" analogy to help you understand CS420 formal concepts, by connecting them to real-world ideas you've seen before







ore powerful
ore complex
ss restricted

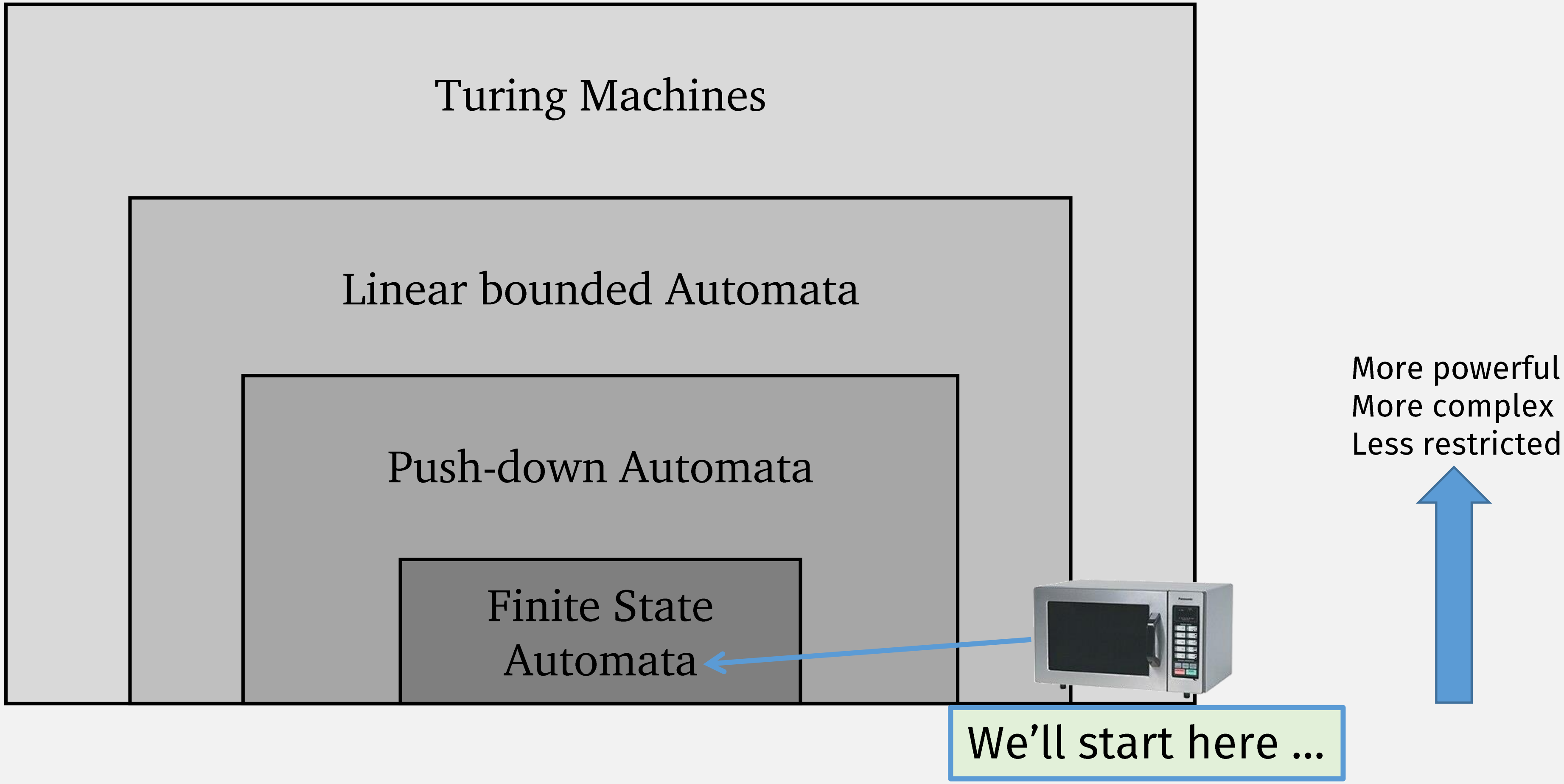
But don't get confused: This course **does not formally study or define this connection** to "programs" and "programming languages"

Intuition for this course:

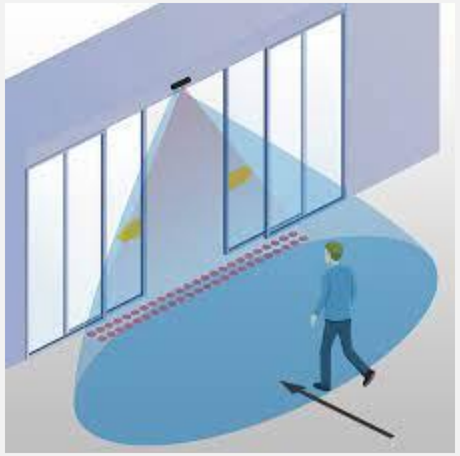
- **A model of compu**
- Think of: a **class o**
- Think of: a **single** machine instance = a "**Program**"

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

Last Time: Models of Computation Hierarchy



Finite Automata: “Simple” Computation / “Programs”



Quiz Preview

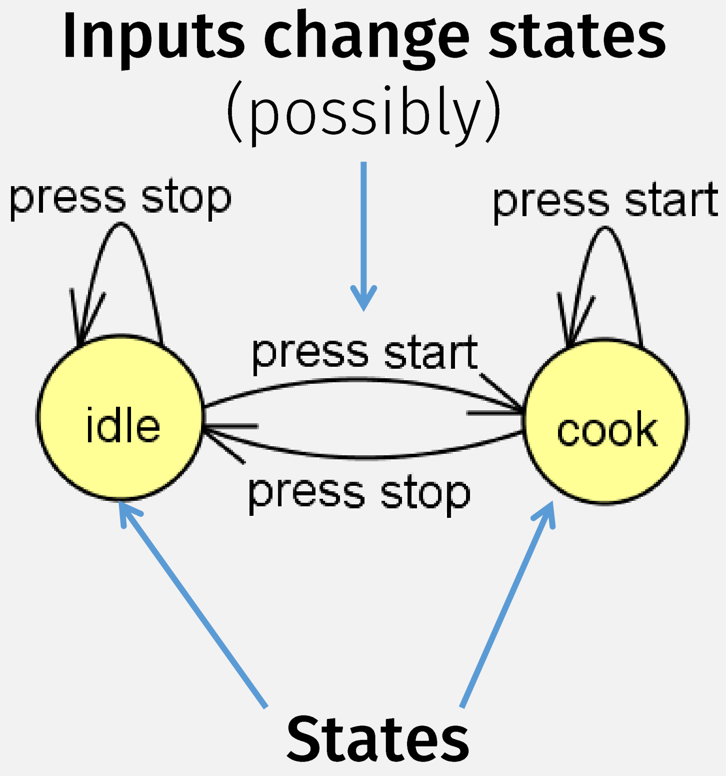
The formal definition of a **finite (state) automata** (FSM)

- has how many components?
- is what kind of mathematical object?

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

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

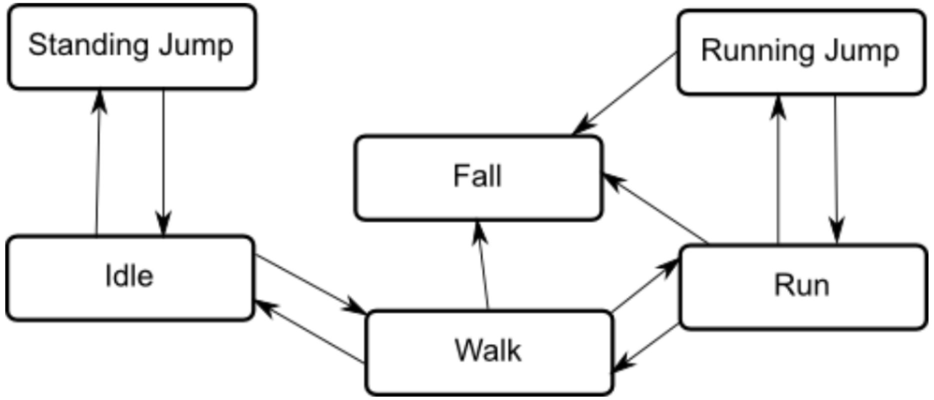
Video Games Love Finite Automata

Unity Documentation Manual

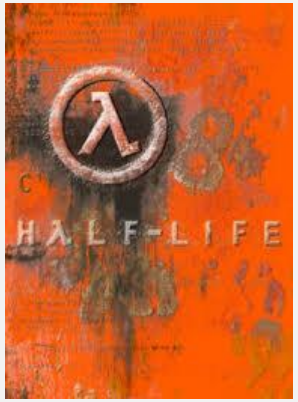
Unity User Manual 2020.3 (LTS) / Animation / Animator Controllers / Animation State Machines / State Machine Basics

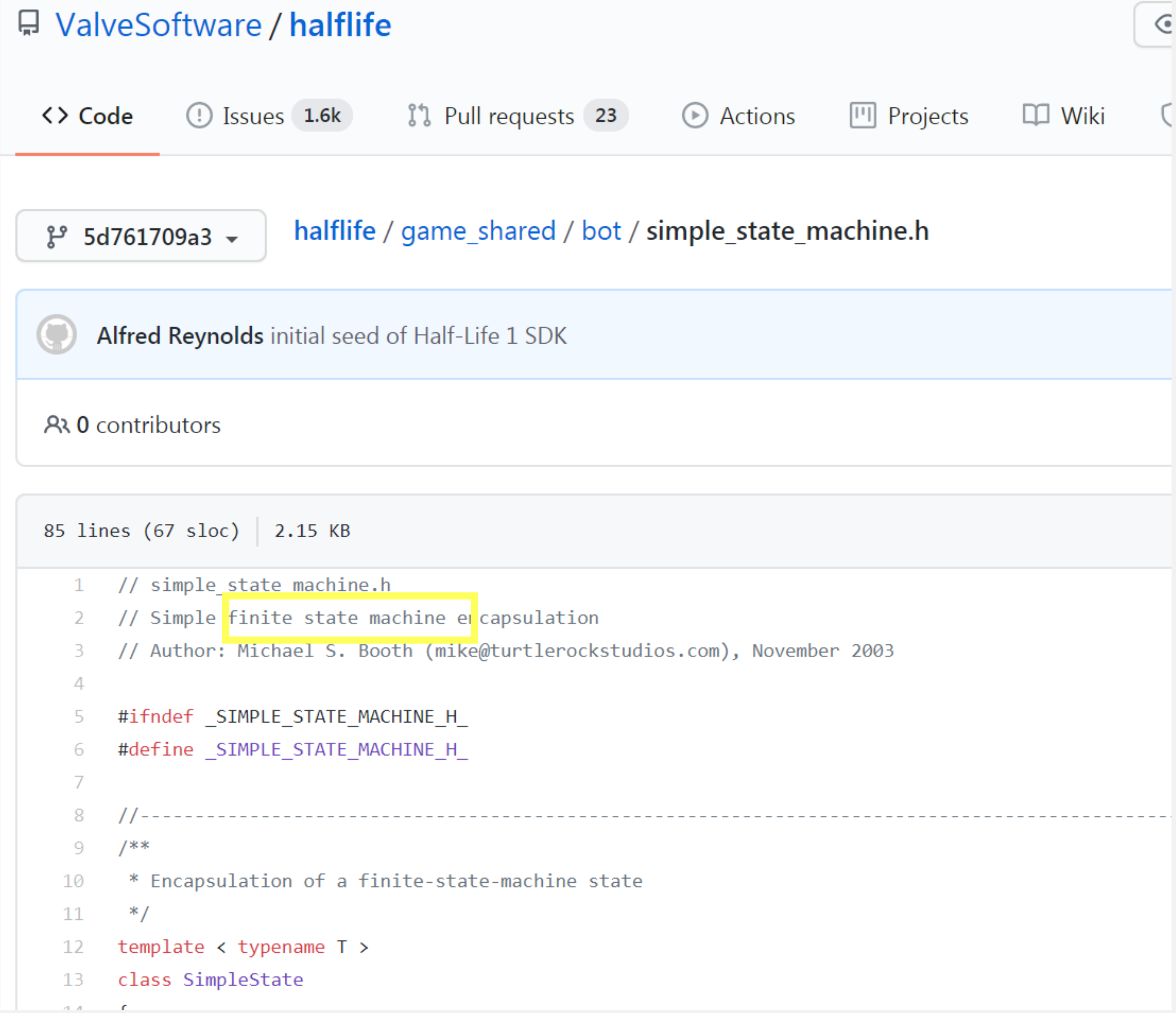
The basic idea is that a character is engaged in some particular kind of action at any given time. The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc. These actions are referred to as **states**, in the sense that the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



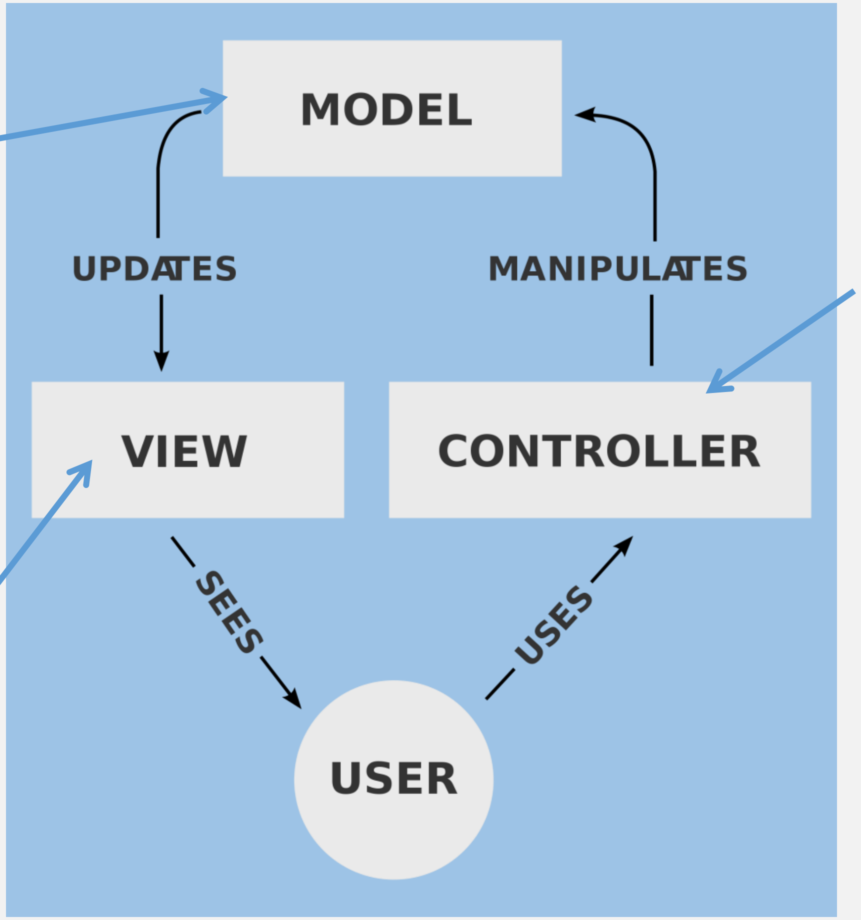
Finite Automata in Video Games





Model-view-controller (MVC) is an FSM

States



Inputs change states

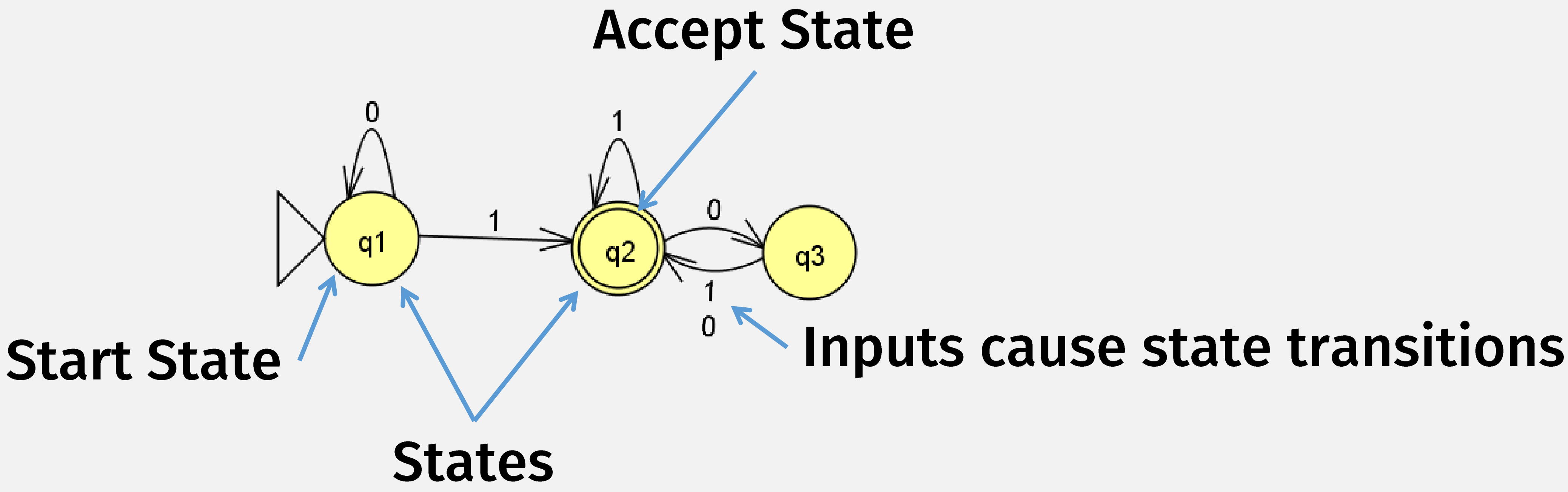
The View draws states

A Finite Automata, as a "Program"

- A **very limited "program"** that uses <u>finite</u> **memory**
  - Actually, **only** <u>1 "cell" of memory</u>!
  - **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 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 mapping, a table, …

DEFINITION

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

1. Q is a finite set called the ***states***, — **set**
2. Σ is a finite set called the ***alphabet***, — **set**
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***, — **Set** of pairs (**domain**) → **Set** (**range**)
4. $q_0 \in Q$ is the ***start state***, and — **Don't know!** (states can be anything)
5. $F \subseteq Q$ is the ***set of accept states***. — **set**

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

Finite Automata: The Formal Definition

5 components

DEFINITION

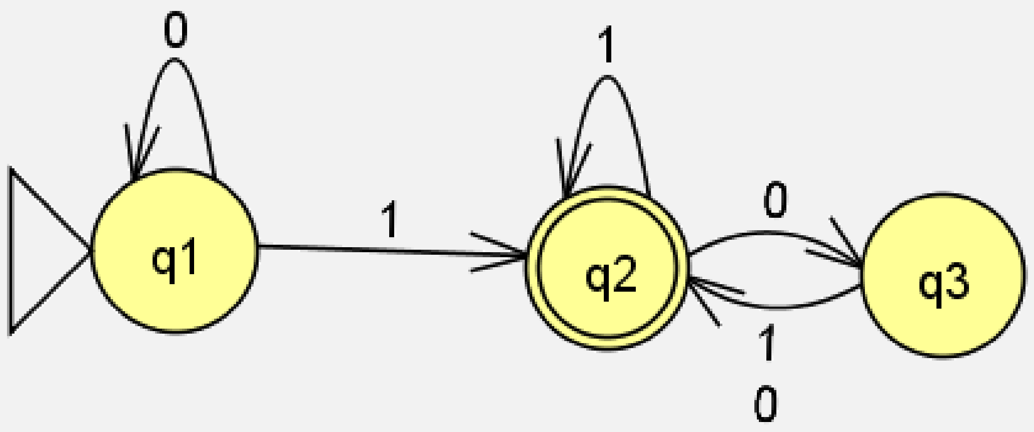
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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***.

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***.



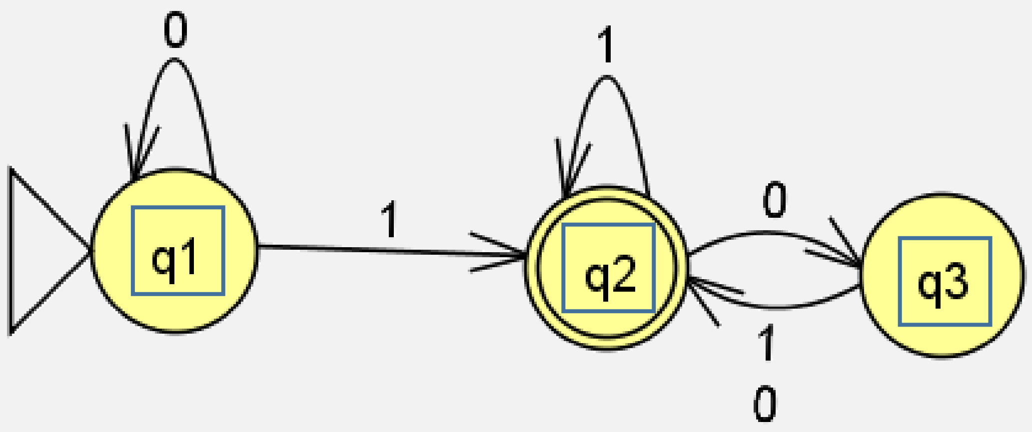
Example: as state diagram

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***.

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,

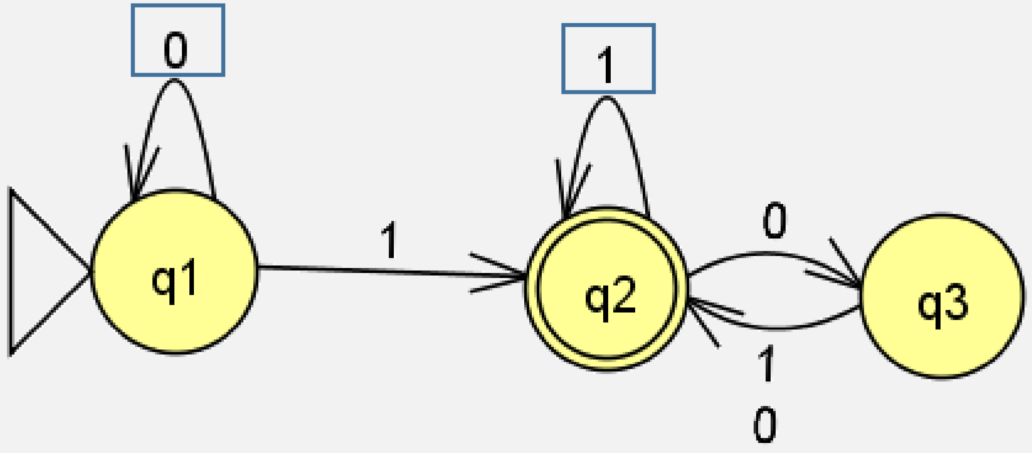
4. q_1 is the start state, and
5. $F = \{q_2\}$.

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. $\delta : Q \times \Sigma \longrightarrow Q$ 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

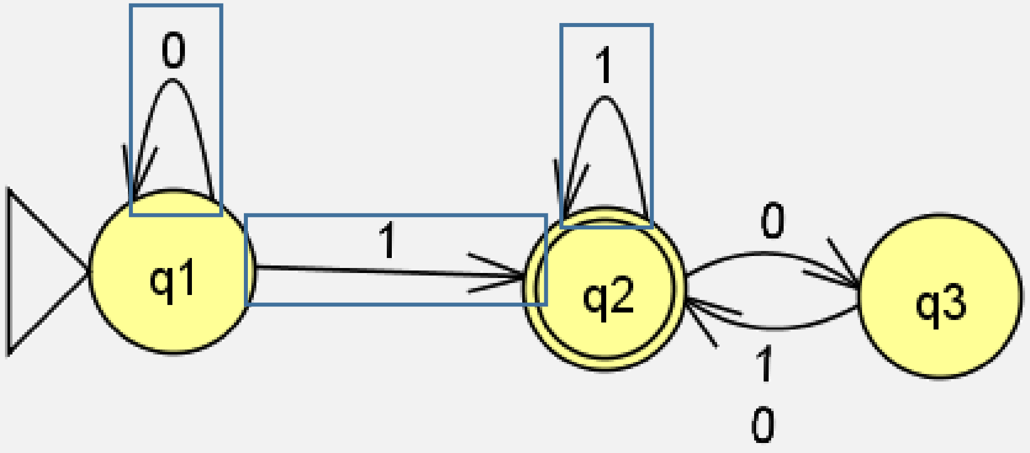
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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. $\delta \colon Q \times \Sigma \longrightarrow Q$ 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>

Example: as <u>formal description</u>

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

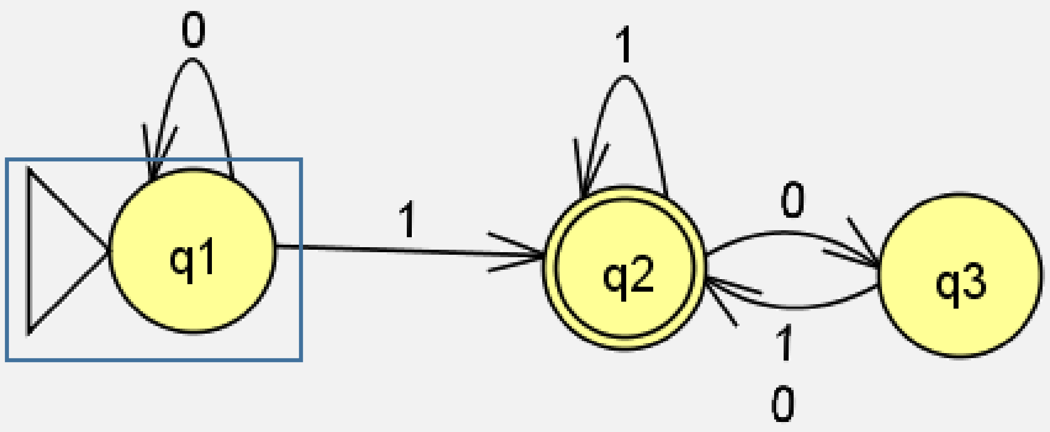
"Then go to this state"

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. $\delta : Q \times \Sigma \longrightarrow Q$ 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>

Example: as <u>formal description</u>

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

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

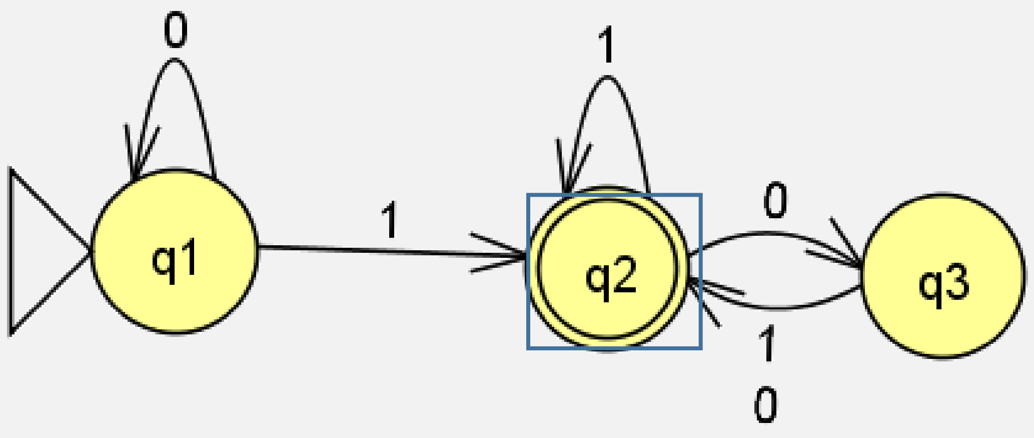
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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

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

A "Program" →

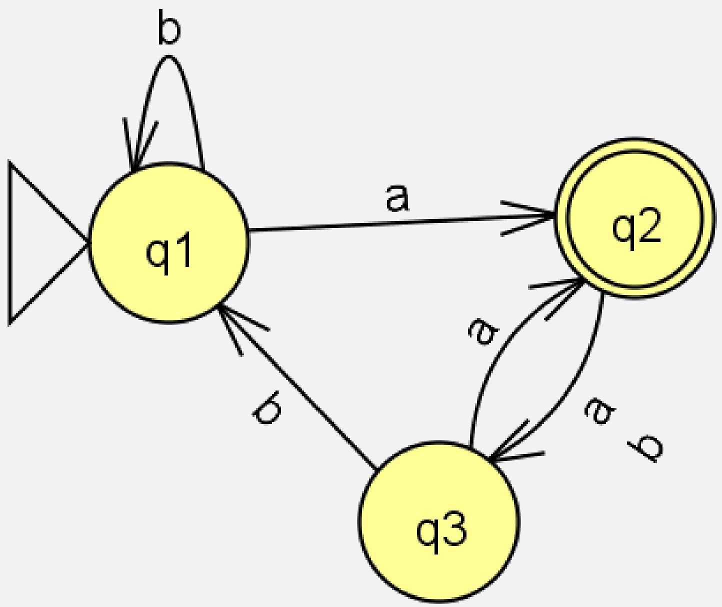
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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

Programming Analogy

In-class Exercise

Come up with a formal description of the following machine:



DEFINITION

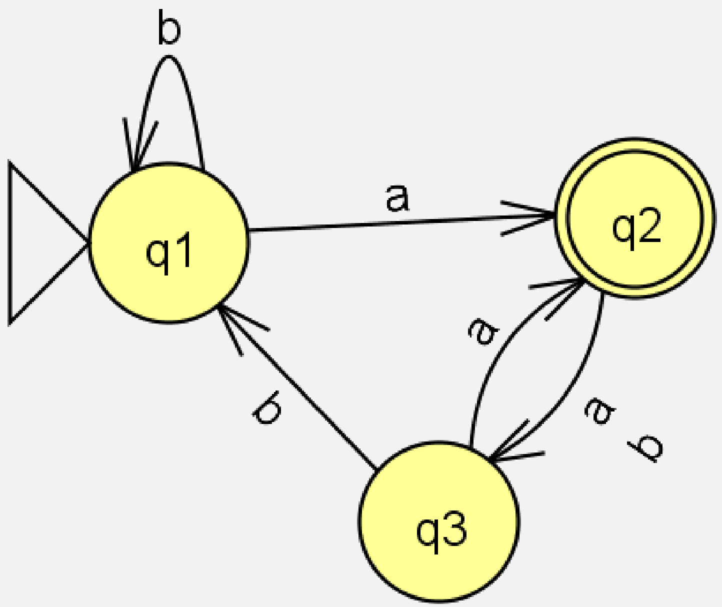
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ 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 …**

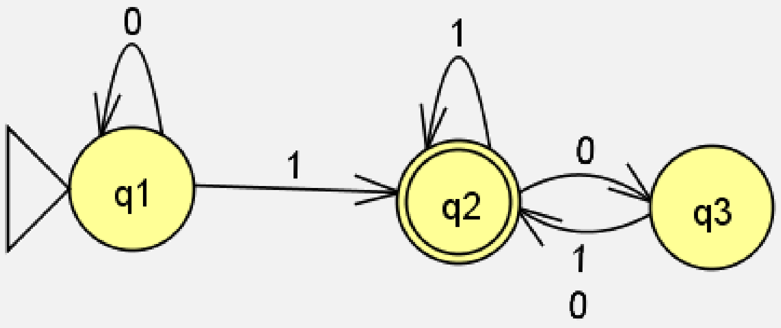
DEFINITION

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)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$
- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, 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 1/25

On gradescope