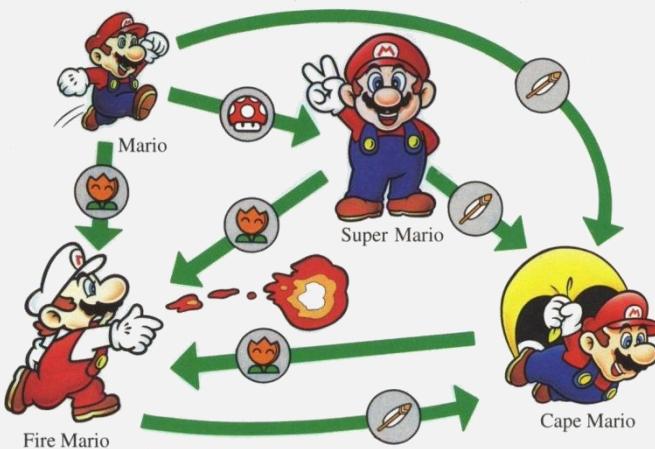


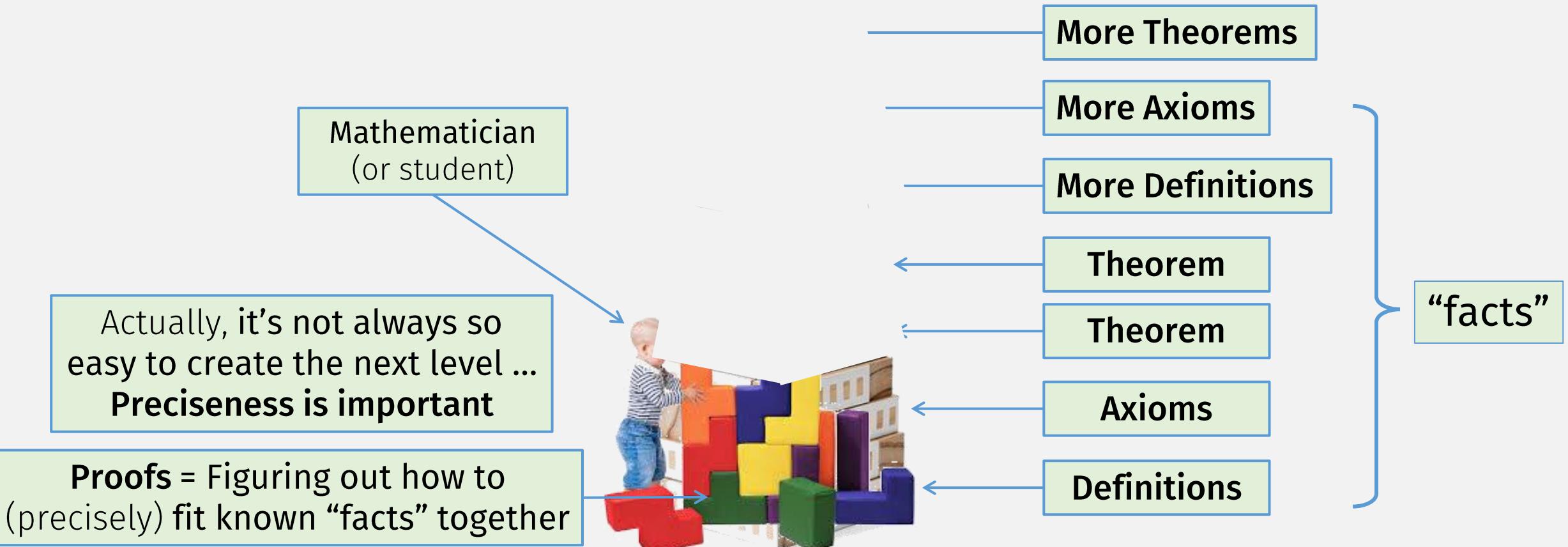
# CS420 (Deterministic) Finite Automata

Wednesday, January 25, 2023

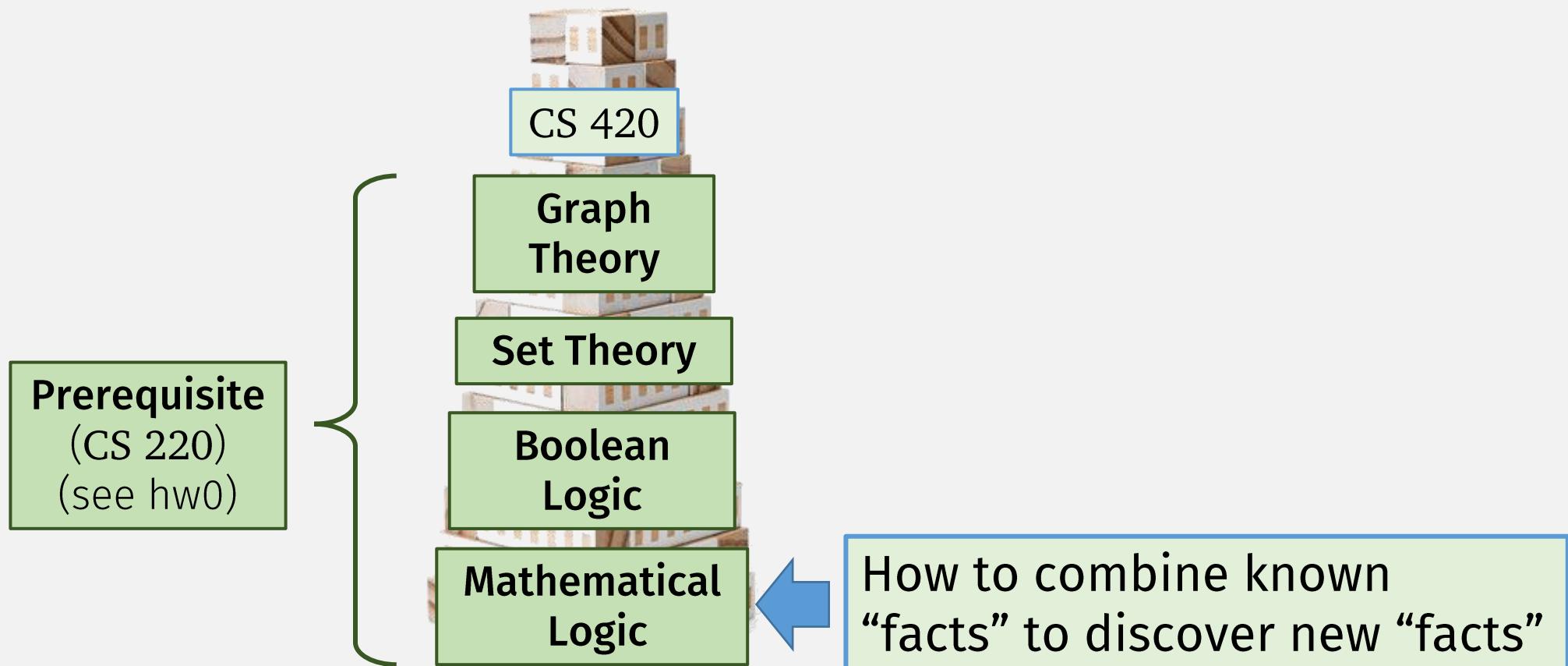
UMass Boston Computer Science



# Last Time: How Mathematics Works



# How CS 420 Works



# Mathematical Logic Operators

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

# 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  $P$  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



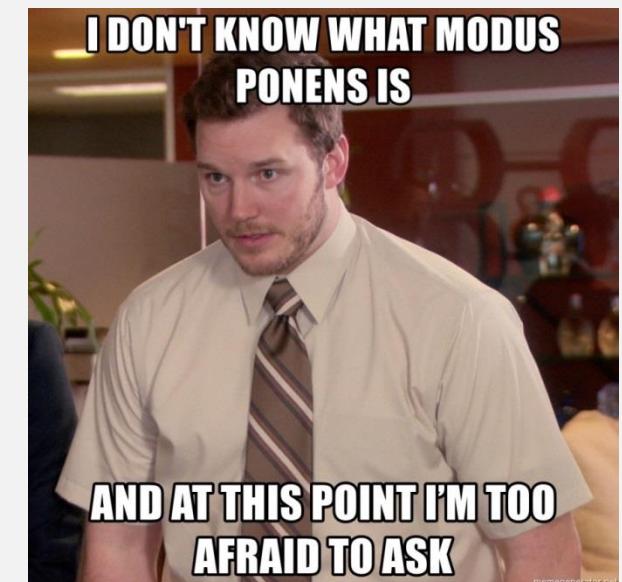
# Using an IF-THEN statement: The “Modus Ponens” Inference Rule

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

## Proving:

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

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



9

# Last Time: Deductive Proof Example

Prove the following:

Proving

- If: If  $x \geq 4$ , then  $2^x \geq x^2$  ← Assume these are true
- And:  $x$  is the sum of the squares of four positive integers
- Then:  $2^x \geq x^2$  ← Prove this is true

Using

# Last Time: Deductive Proof Example

Prove: 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

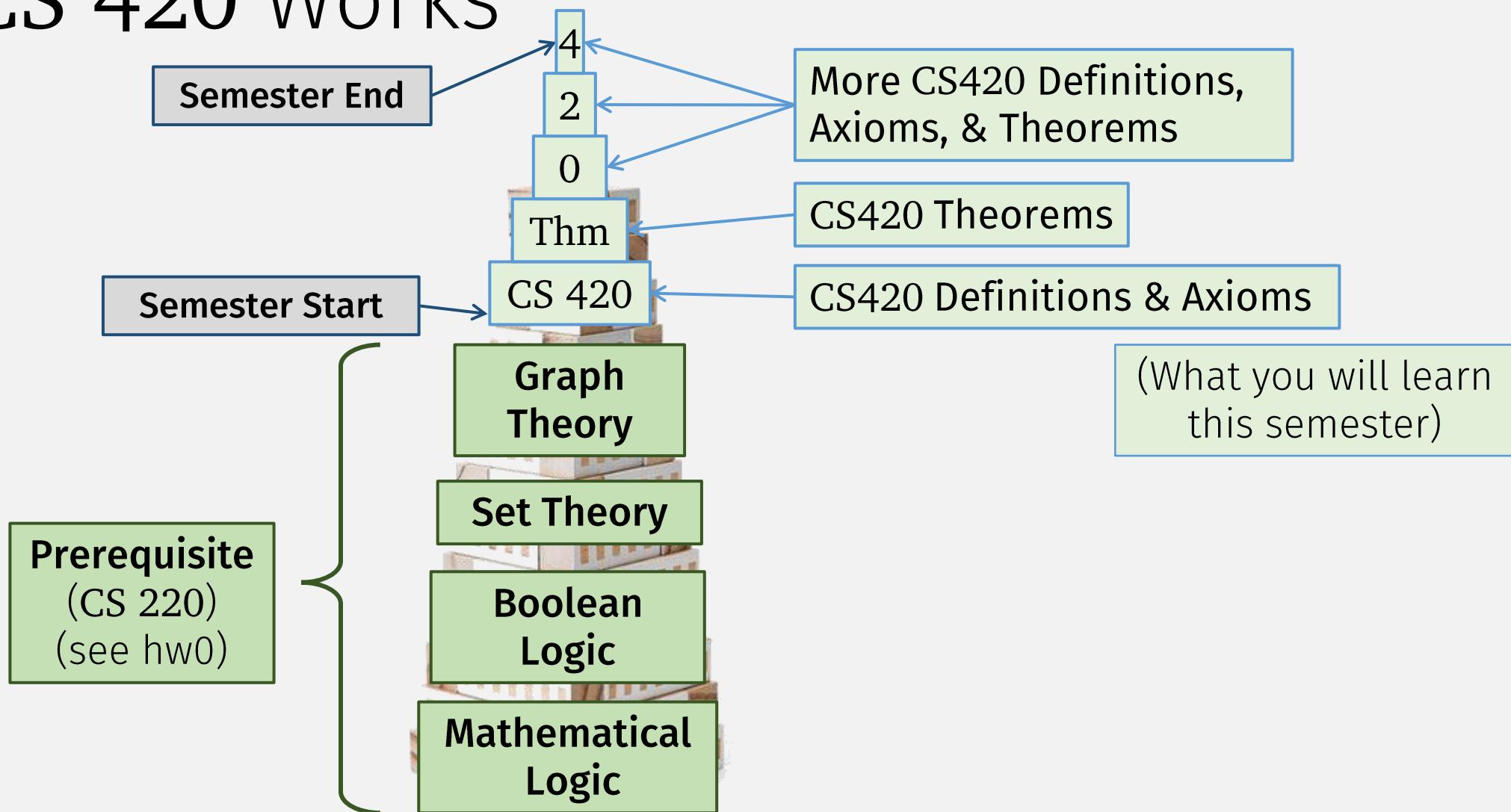
1.  $x = a^2 + b^2 + c^2 + d^2$
2.  $a \geq 1; b \geq 1; c \geq 1; d \geq 1$
3.  $a^2 \geq 1; b^2 \geq 1; c^2 \geq 1; d^2 \geq 1$
4.  $x \geq 4$
5. If  $x \geq 4$ , then  $2^x \geq x^2$
6.  $2^x \geq x^2$

## Justification

1. Assumption
2. Assumption
3. By Stmt #2 & arithmetic laws
4. Stmts #1, #3, and arithmetic
5. Assumption
6. Stmts #4 and #5      Modus ponens



# How CS 420 Works



# A Word of Advice

**Important:**  
**Do not fall behind**  
in this course



To prove a (new) theorem ...  
... need to know all axioms,  
definitions, and (previous)  
theorems below it

# More Advice

“Answer Hunting”  
won’t work in CS420



“Blocks” from outside the  
course won’t work in the proof

HW graded on  
understanding of  
how you got to the  
answer, not the final  
answer itself!



HW 1, Problem 1

Prove that  $ABC = XYZ$



Prove that  $ABC = XYZ$

A Note from CS420 –  
Spring2020 Theorem

Google Search I'm Feeling Lucky



... can be used to prove (new)  
theorems in this course

Only axioms, definitions, and  
theorems from this course...

# How to Do Well in this Course

- Learn the “building blocks”
  - I.e., axioms, definitions, and theorems
- To solve a problem (prove a new theorem) ...  
... think about how to (precisely) combine existing “blocks”
- HW problems graded on how you got to the answer,  
not on the final answer itself!
- Don't Fall Behind!
  - Start HW Early (HW 0 due Sunday 11:59pm EST)
- Participate and Engage
  - Lecture
  - Office Hours
  - Message Boards (piazza)

# Textbooks

- Sipser. *Intro to Theory of Computation*, 3<sup>rd</sup> ed.
- Hopcroft, Motwani, Ullman. *Intro to Automata Theory, Languages, and Computation*, 3<sup>rd</sup> 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!
- Late Policy: 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

- 1<sup>st</sup> offense: zero on problem
- 2<sup>nd</sup> offense: zero on hw, reported to school
- 3<sup>rd</sup> 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?

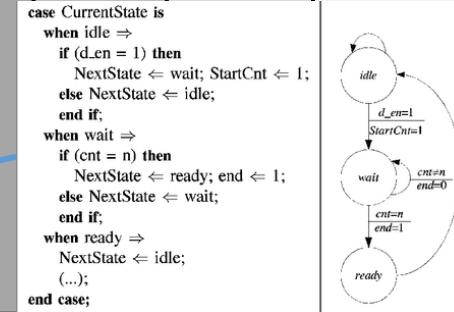
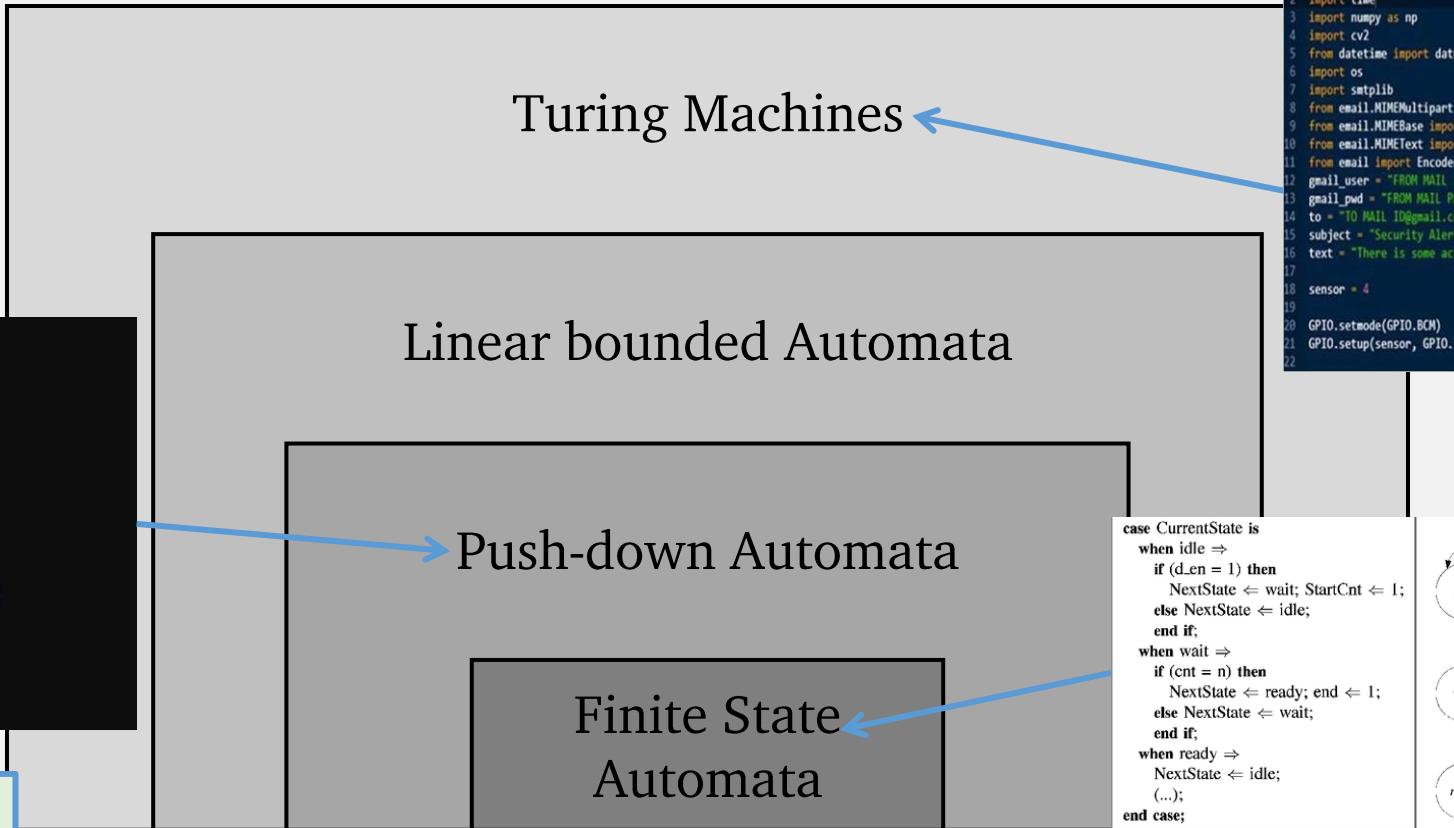
# Last Time: Computation = Programs!

Lesson 1 - CFG Grammar  
E → T E'  
E' → + T E'  
E' →  
T → F T'  
T' → \* F T'  
T' →  
F → ( E )  
F → int

Lesson 2 - FA Lexer  
NFA machine contains 11 total states

Lesson 3 - Runtime Parser  
Reading expression "3+5\*7"  
NonTerminal E: , Line 1, Column 1  
NonTerminal t: , Line 1, Column 1

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

```
Lesson 1
E -> T E'
E' -> + T E'
E' ->
T -> F T'
T' -> * F T'
T' ->
F -> ( E )
F -> int

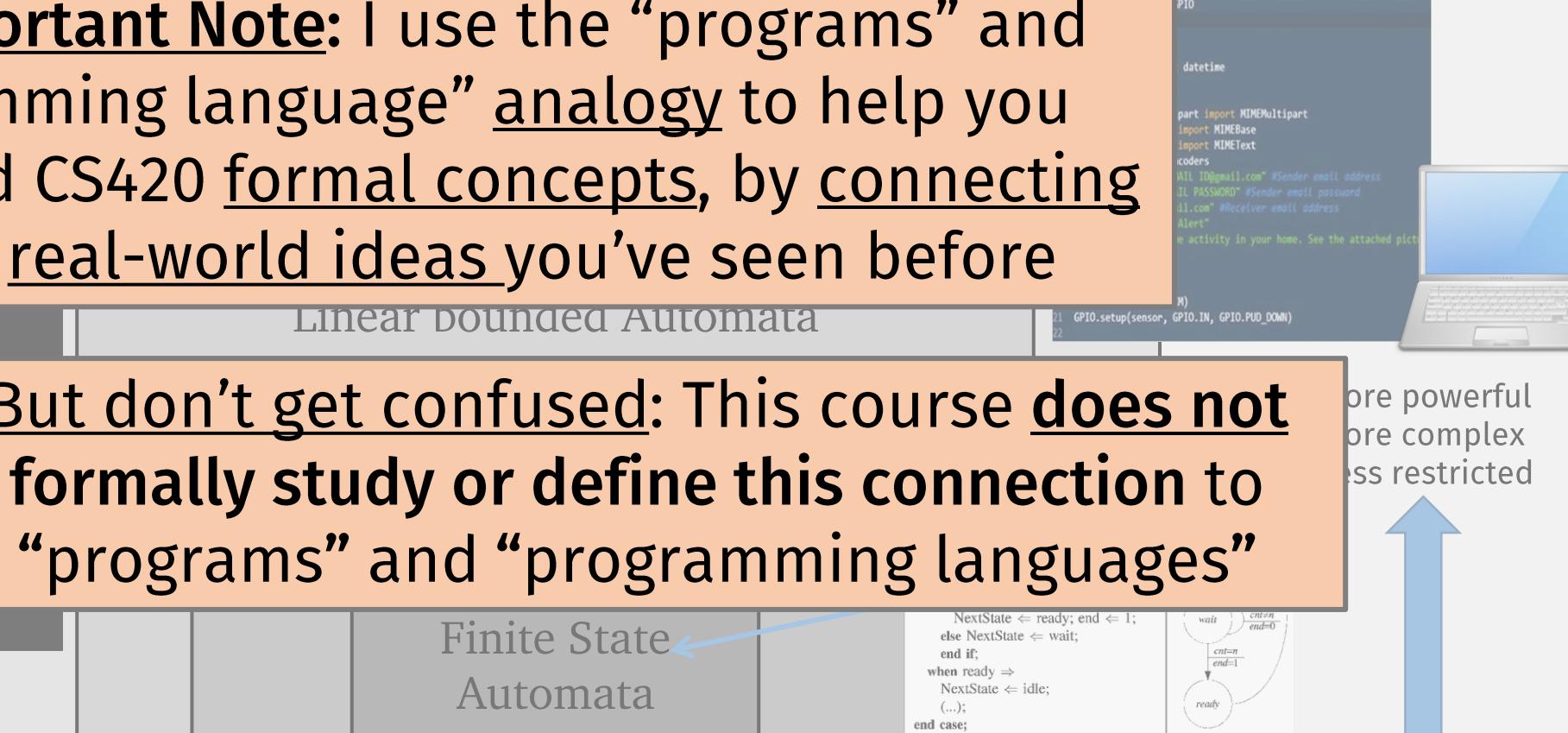
Lesson 2 - FA Lexer
NFA machine contains 11 total states

Lesson 3 - Runtime Parser
Reading expression "3+5*7"
NonTerminal   E: , Line 1, Column 1
NonTerminal   t: , Line 1, Column 1
```

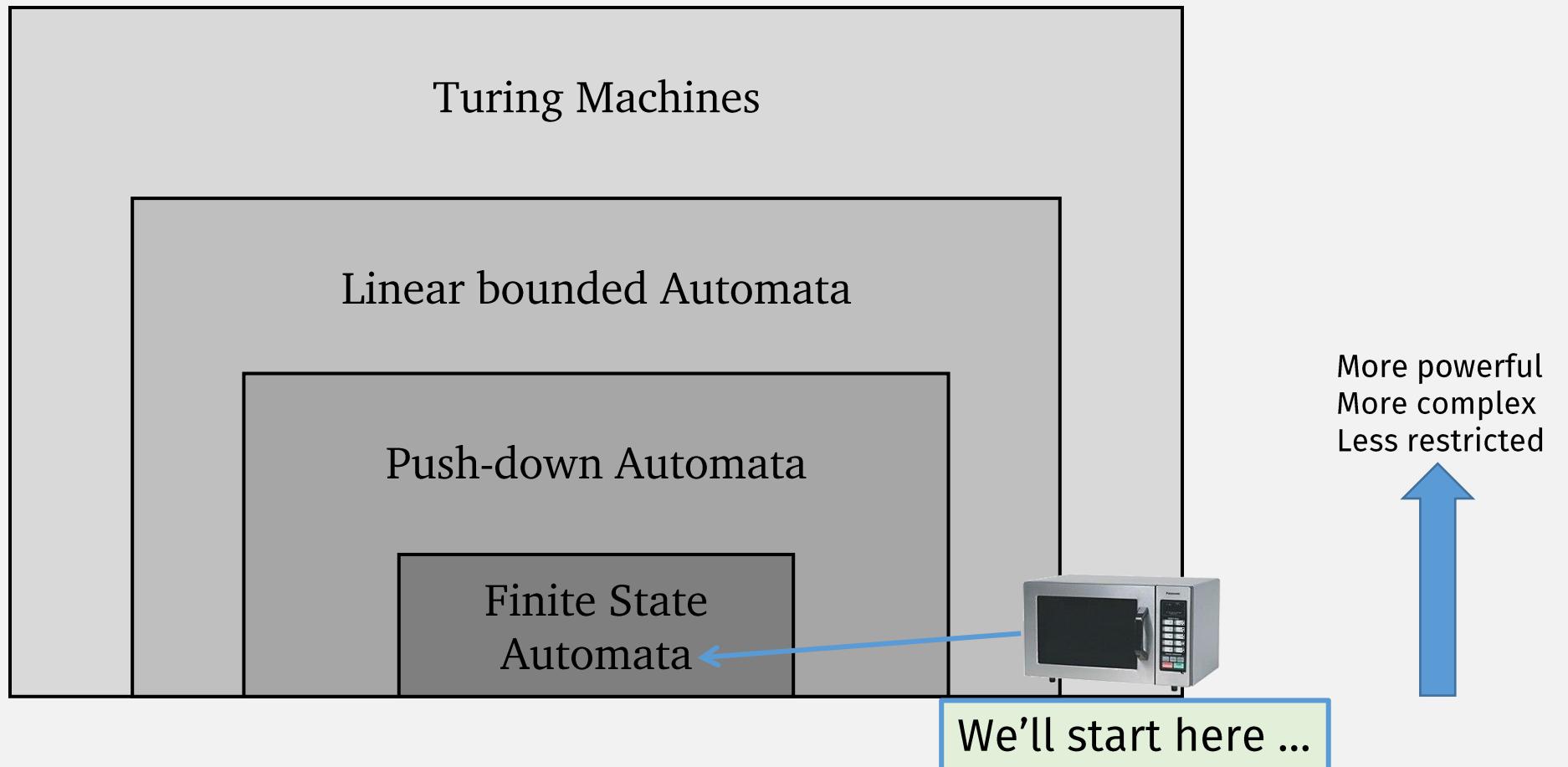
Intuition for this course:

- A model of computation
- Think of: a class of machines
- Think of: a single machine instance – a **program**:

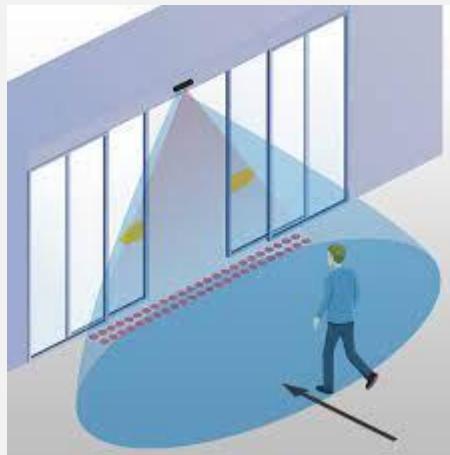
In fact, the term **language** will formally mean something else (later)



# Last Time: Models of Computation Hierarchy



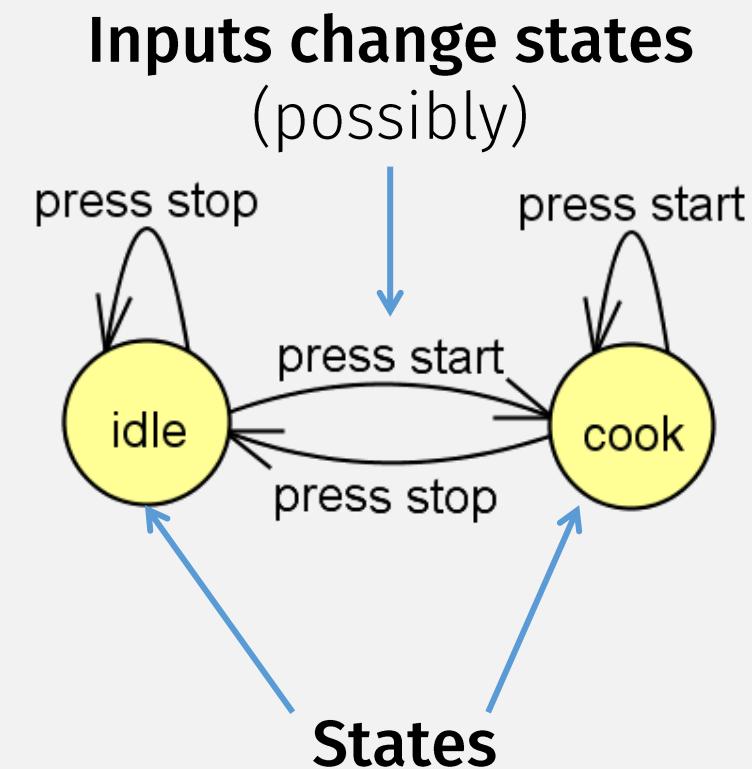
# Finite Automata: “Simple” Computation / “Programs”



# Finite Automata

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

# A Microwave Finite Automata



# 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

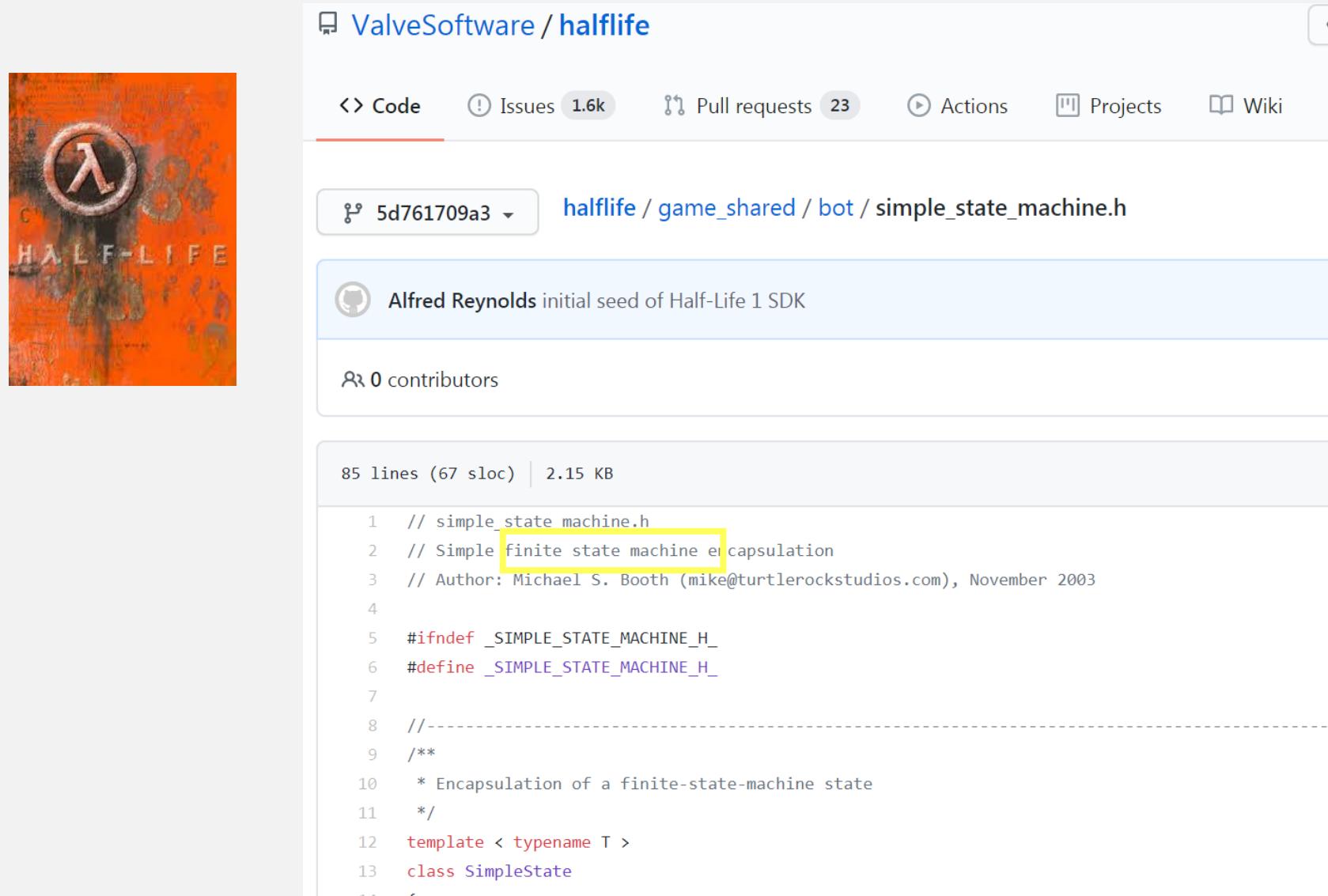
The screenshot shows a section from the Unity Documentation Manual about State Machine Basics. The URL is [Unity User Manual 2020.3 \(LTS\) / Animation / Animator Controllers / Animation State Machines / State Machine Basics](#). The text explains that a character is in a state like walking or idling, and can transition to another state like running or jumping. It uses the term "states" to refer to the character's condition and "state transitions" to refer to the possible moves between states. The text also mentions restrictions on transitions, such as not being able to run jump directly from idle. Below the text is a state machine diagram.

```
graph TD; Idle[Idle] --> StandingJump[Standing Jump]; Idle --> Walk[Walk]; Walk --> Fall[Fall]; Walk --> Run[Run]; Run --> Fall; Run --> StandingJump; Fall --> RunningJump[Running Jump]; RunningJump --> Fall; RunningJump --> StandingJump;
```

The diagram illustrates a state machine with the following states and transitions:

- States:** Idle, Walk, Run, Fall, Standing Jump, Running Jump.
- Transitions:**
  - Idle to Standing Jump
  - Idle to Walk
  - Walk to Fall
  - Walk to Run
  - Run to Fall
  - Run to Standing Jump
  - Fall to Running Jump
  - Running Jump to Fall
  - Running Jump to Standing Jump

# 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

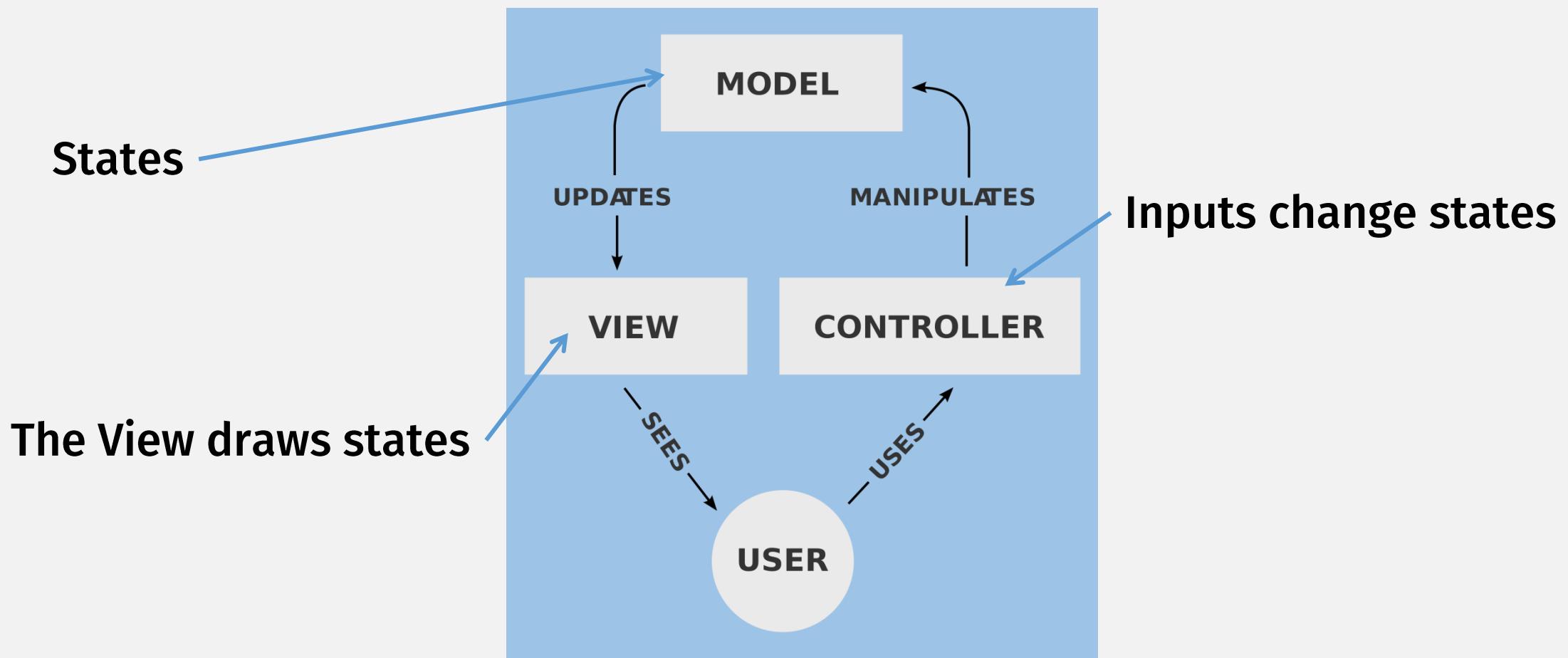
Alfred Reynolds initial seed of Half-Life 1 SDK

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
```

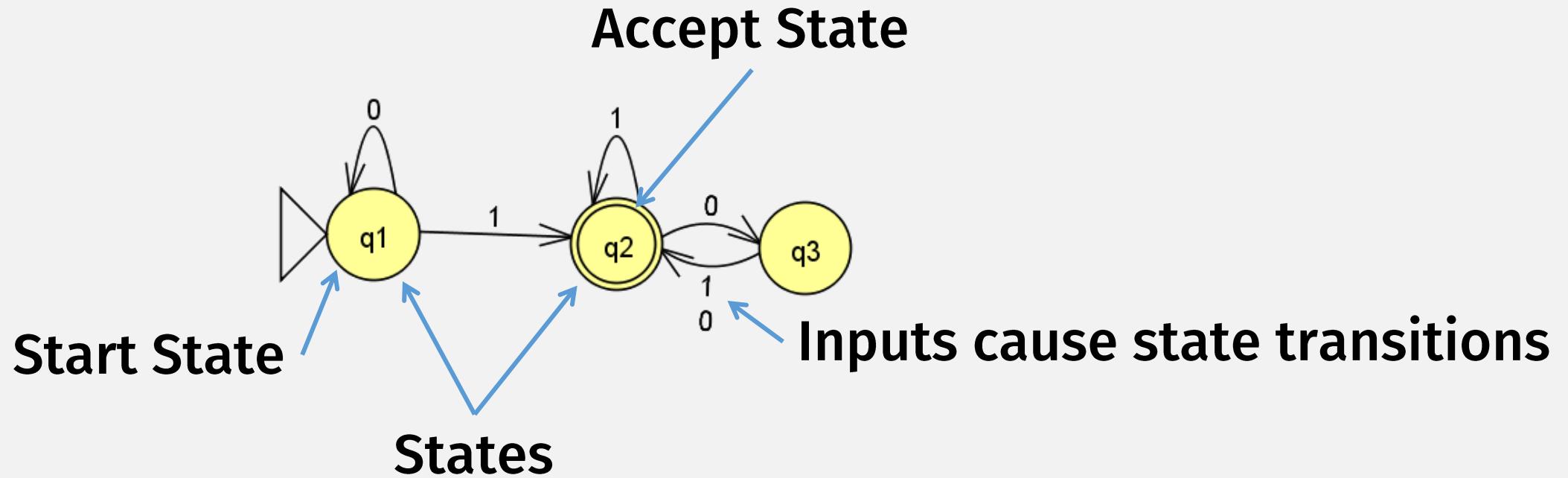
# Model-view-controller (MVC) is an FSM



# A Finite Automata, as 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

## DEFINITION

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



# 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

(1<sup>st</sup> of each pair from **domain**, 2<sup>nd</sup> 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.  $\Sigma$  is a finite set called the *alphabet*,

sequence

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

Don't know!  
(states can be anything)

4.  $q_0 \in Q$  is the *start state*, and Set (range)

set

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

A pair is ... a sequence of 2 elements

# 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.  $\Sigma$  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*.

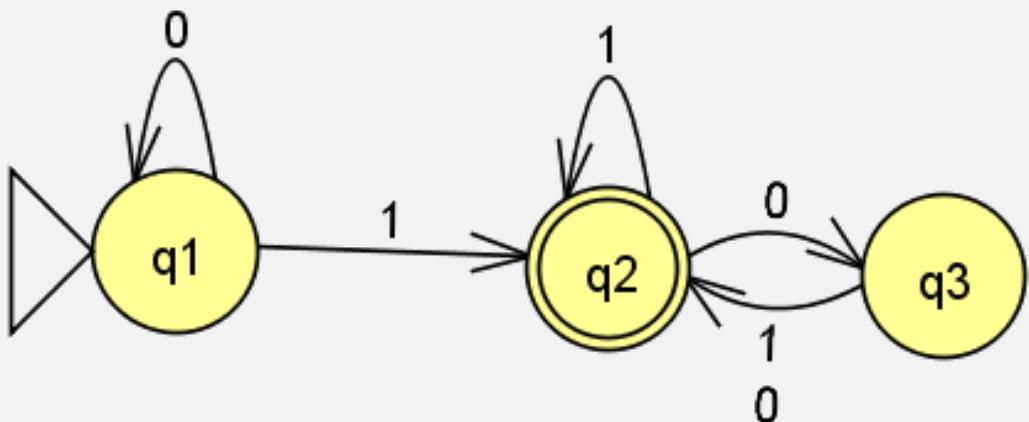
5 components



## 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.  $\Sigma$  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***.

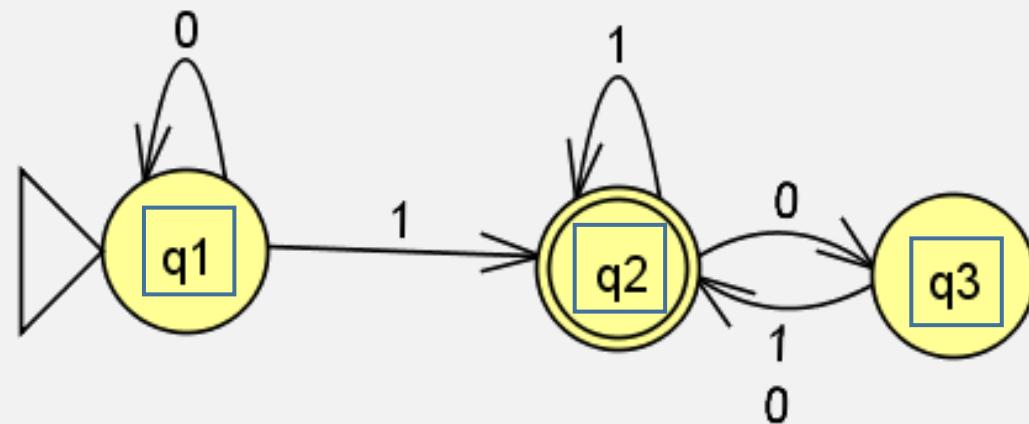


Example: as state diagram

## DEFINITION

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

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

braces =  
set notation  
(no duplicates)

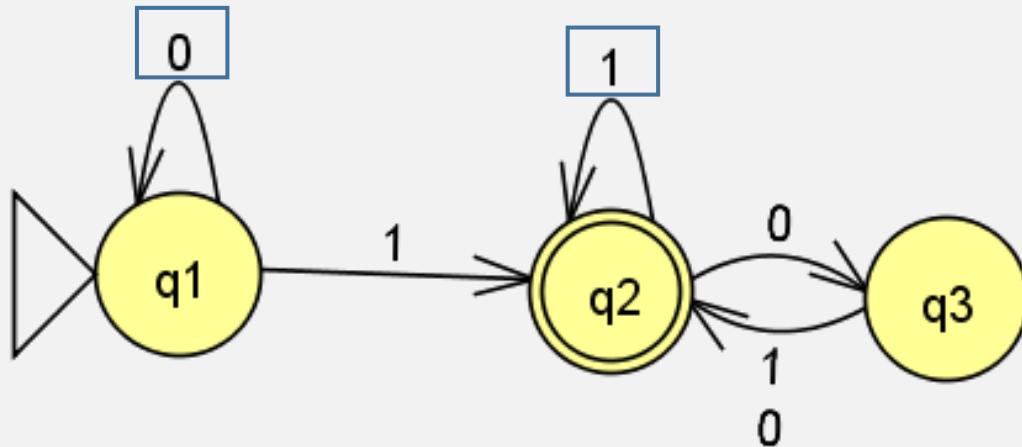
	0	1
$q_1$	$q_1$	$q_2$
$q_2$	$q_3$	$q_2$
$q_3$	$q_2$	$q_2$

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.  $\Sigma$  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

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.  $\delta$  is described as

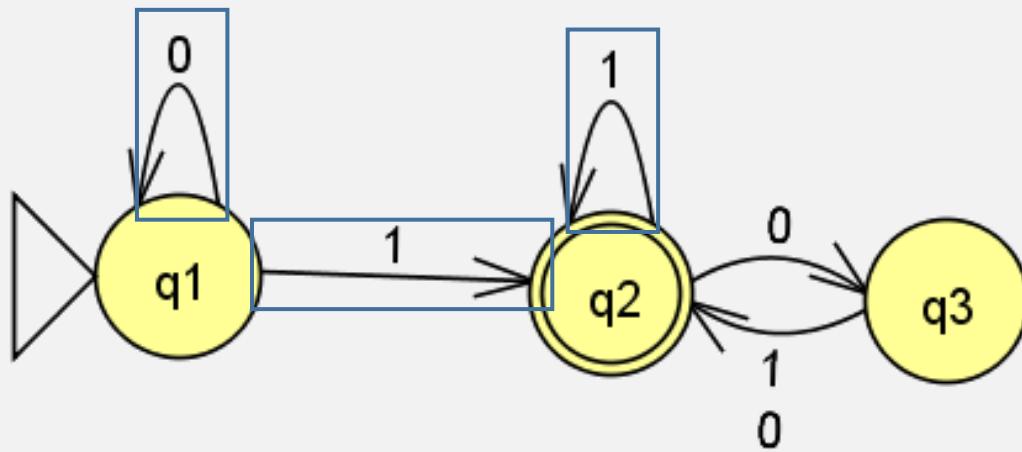
	0	1
$q_1$	$q_1$	$q_2$
$q_2$	$q_3$	$q_2$
$q_3$	$q_2$	$q_2$

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.  $\Sigma$  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

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

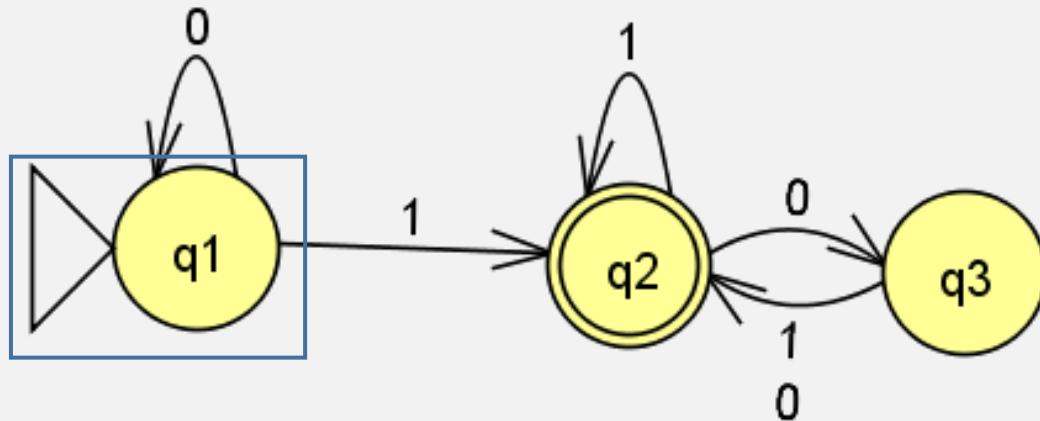
	0	1
<b>“If in this state”</b>	$q_1$	$q_1$
$q_1$	$q_3$	$q_2$
$q_2$	$q_2$	<b>“Then go to this state”</b>
$q_3$	$q_2$	<b>“And this is next input symbol”</b>

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.  $\Sigma$  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

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

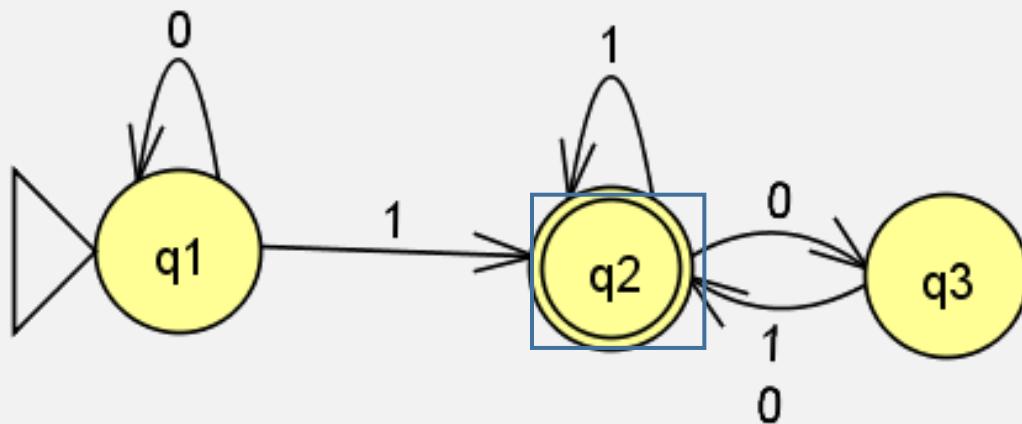
	0	1
$q_1$	$q_1$	$q_2$
$q_2$	$q_3$	$q_2$
$q_3$	$q_2$	$q_2$

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.  $\Sigma$  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

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

	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.  $\Sigma$  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 “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

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

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

	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

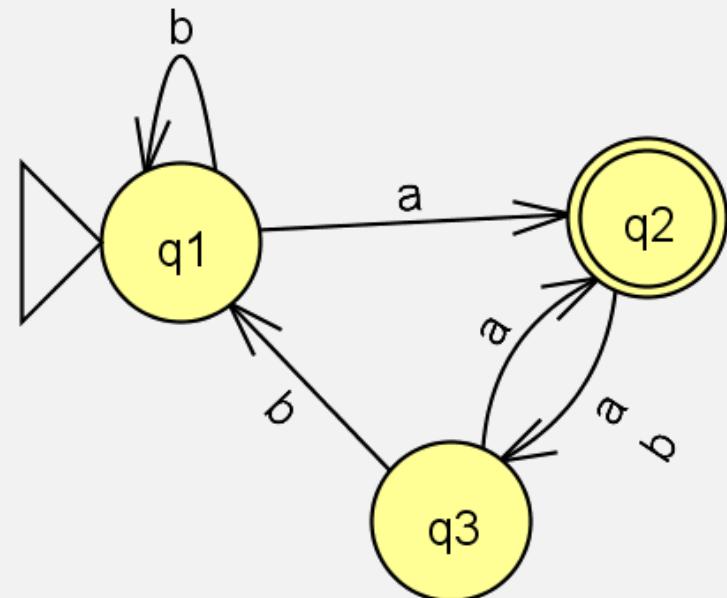
Come up with a formal description of the following machine:

## DEFINITION

---

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

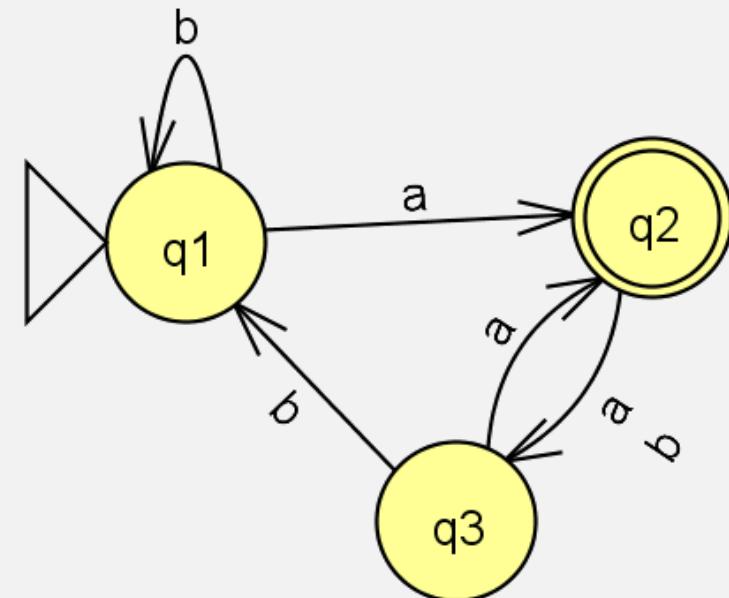
1.  $Q$  is a finite set called the **states**,
2.  $\Sigma$  is a finite set called the **alphabet**,
3.  $\delta: Q \times \Sigma \rightarrow Q$  is the **transition function**,
4.  $q_0 \in Q$  is the **start state**, and
5.  $F \subseteq Q$  is the **set of accept states**.



# In-class Exercise: solution

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- $\delta$ 
  - $\delta(q_1, a) = q_2$
  - $\delta(q_1, b) = q_1$
  - $\delta(q_2, a) = q_3$
  - $\delta(q_2, b) = q_3$
  - $\delta(q_3, a) = q_2$
  - $\delta(q_3, b) = q_1$
- $q_0 = q_1$
- $F = \{q_2\}$

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



# A Computation Model is ... (from lecture 1)

- Some base definitions and axioms ...

## DEFINITION

---

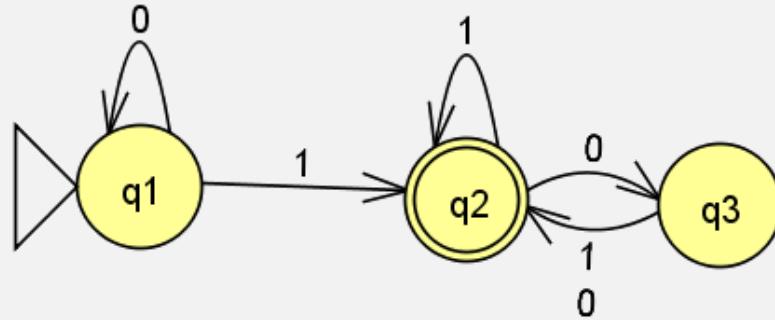
A **finite automaton** is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

1.  $Q$  is a finite set called the **states**,
2.  $\Sigma$  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**.

- 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  $i = 0, \dots, n - 1$

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

- $M$  **accepts**  $w$  if sequence of states  $r_0, r_1, \dots, r_n$  in  $Q$  exists ... with  $r_n \in F$

**Still a little verbose**

# **Check-in Quiz 1/25**

On gradescope