CS622
(Deterministic) Finite Automata

Wednesday, January 31, 2024

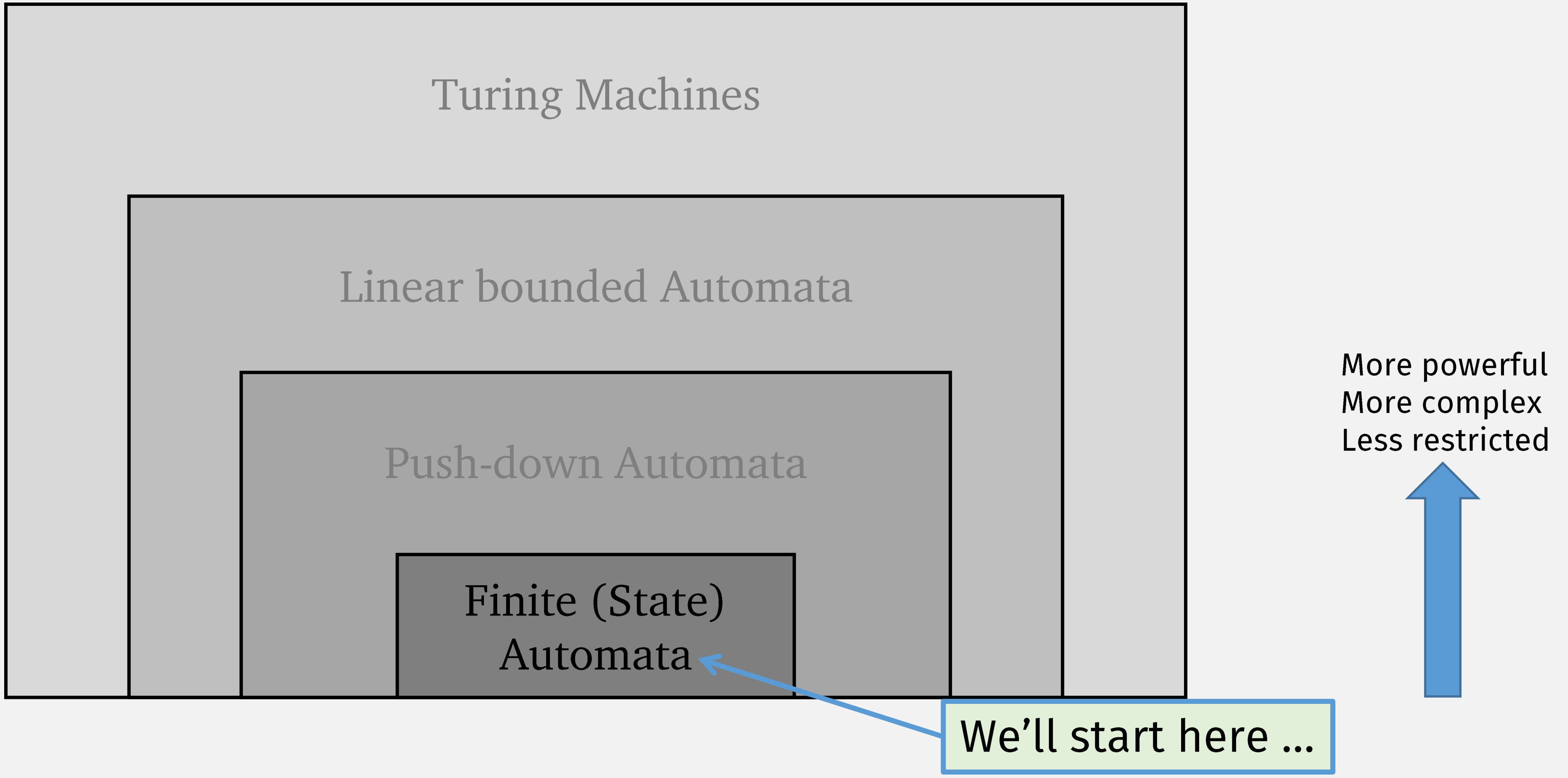
UMass Boston Computer Science

Announcements

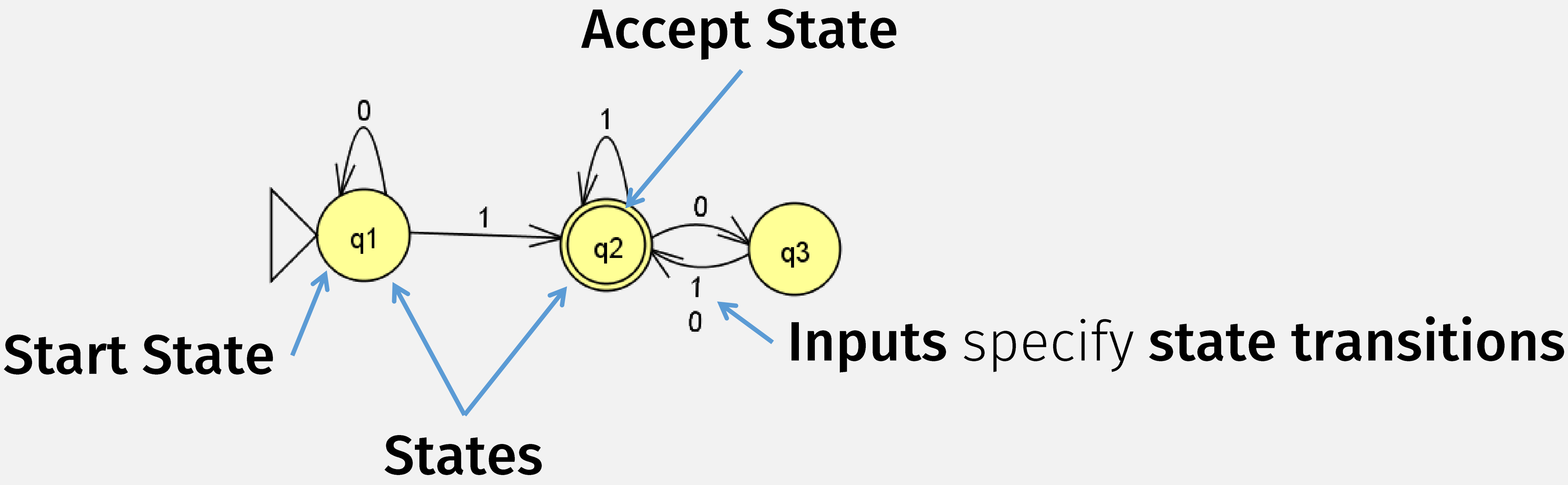
- HW 1
  - due date extended: Mon 2/12, 12pm EST (noon)

- Please ask all HW questions on Piazza!
  - So all course staff can see,
  - and entire class can benefit
  - Please do not email course staff with HW questions

Last Time: Models of Computation Hierarchy



Finite Automata state diagram



Analogy: Finite Automata is a "Program"

- A **restricted "program"** with access to <u>**finite**</u> **memory**
  - Only <u>**1 "cell"** of **memory**</u>**!**
  - Possible **contents** of **memory = # of states**

- **Finite Automata has different representations:**
  - **Code** (wont use in this class)
  - ➢ **State diagrams**

Analogy: Finite Automata is a "Program"

- A **restricted "program"** with access to <u>**finite**</u> **memory**
  - Only <u>**1 "cell"** of **memory**</u>!
  - Possible **contents** of **memory = # of states**

- **Finite Automata has different representations:**
  - **Code** (wont use in this class)
  - **State diagrams**
  - ➢ **Formal math description** (like code, just a different "programming lang")

Finite Automata: The Formal Definition

DEFINITION

deterministic

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

(DFA)

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***.

This semester

Things in **bold** have **precise formal definitions.**
(be sure to look up and review the definition whenever you are unsure)

Analogy

This is the "**programming language**" for **(deterministic) finite automata** "programs"

Finite Automata: The Formal Definition

Set or **sequence**?

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***.

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
- Notation: { }
- **Empty set** written: Ø or { }
- A **language** is a (possibly infinite) set of strings

A **set** used a lot in this course

Sequences

- Ordered
- Duplicates ok
- Notation: **varies:** ( ), **comma,** or **append**
- **Empty sequence:** ( )
- A **tuple** is a finite sequence
- A **string** is a finite sequence of characters

sequences used a lot in this course

Set or Sequence ?

A **function** is … | … a **set** of **pairs** (1st of each pair **from domain**, 2nd **from range**) | … has many representations: 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

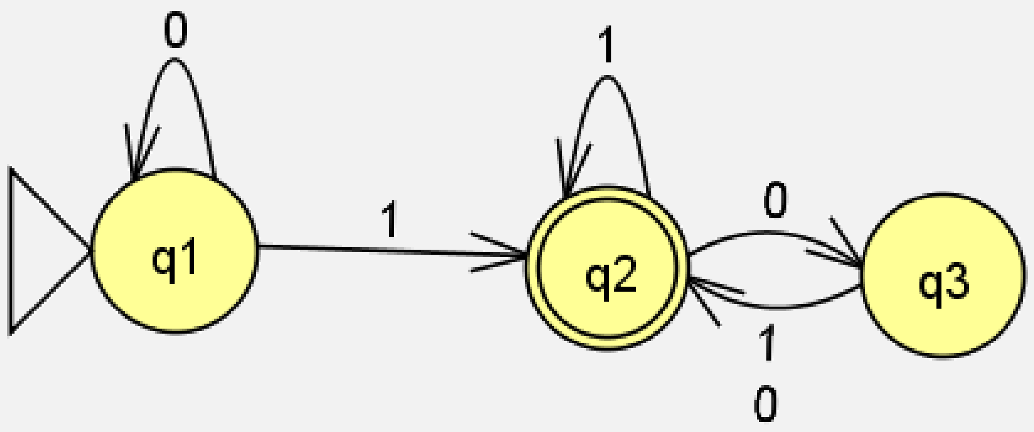
A ***finite automaton*** is a 5-tuple $\boxed{(Q, \Sigma, \delta, q_0, F)}$, where

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



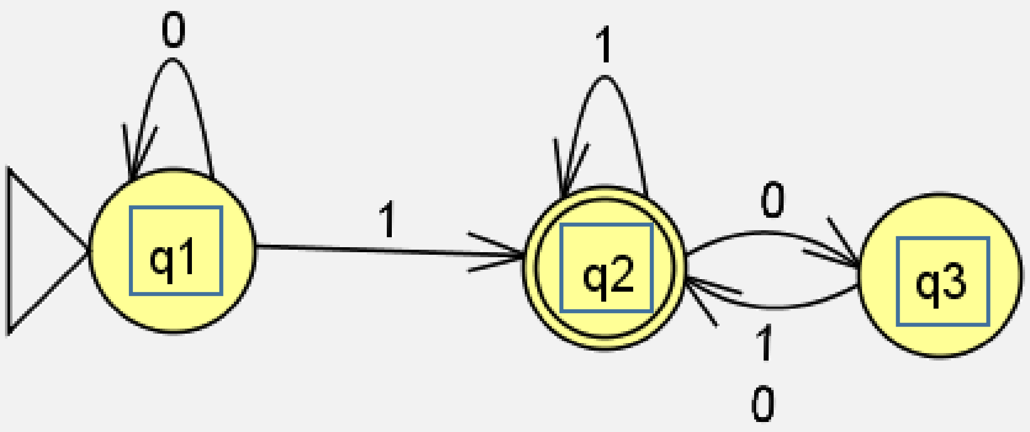
An **Example** (as **state diagram**)

DEFINITION

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

1. $\boxed{Q \text{ is a finite set called the } \textbf{\textit{states}},}$
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



An **Example** (as **state diagram**)

An **Example** (as **formal description**)

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

1. $\boxed{Q = \{q_1, q_2, q_3\},}$
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,

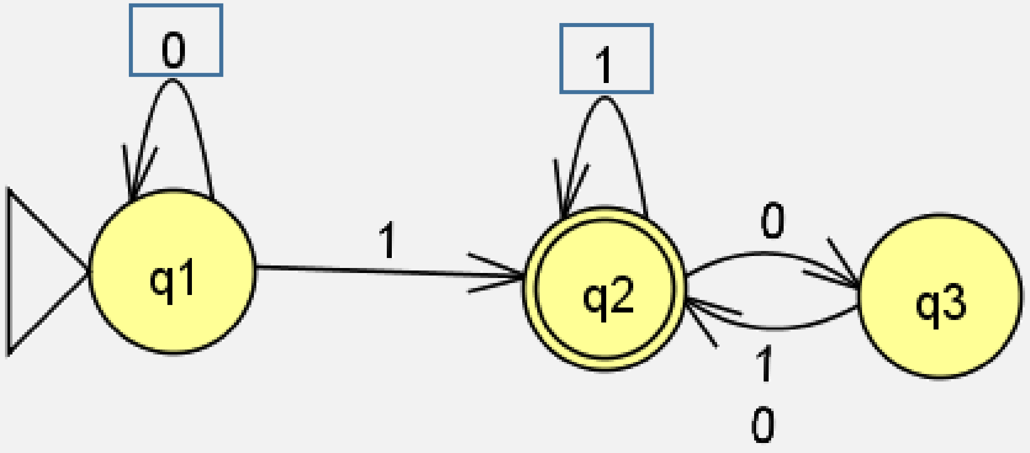
4. q_1 is the start state, and
5. $F = \{q_2\}$.

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***.



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

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

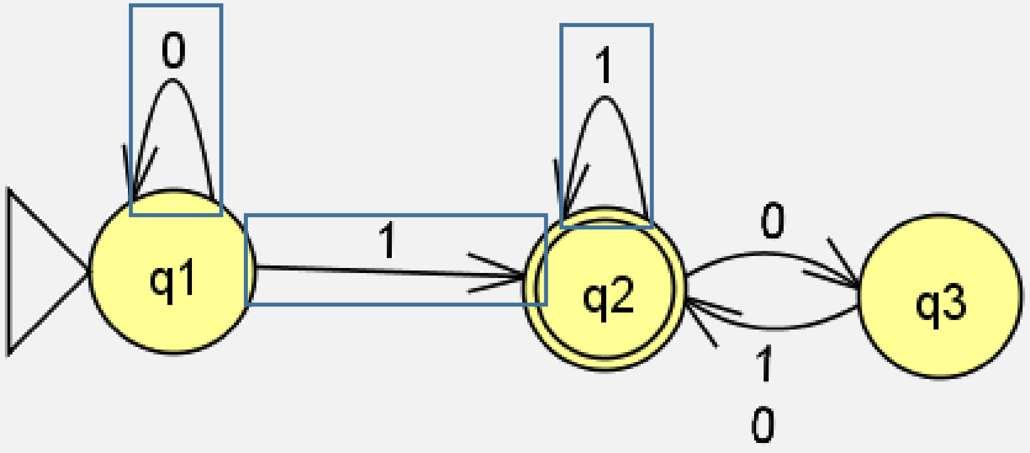
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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***.



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

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

"And this is next input symbol"

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

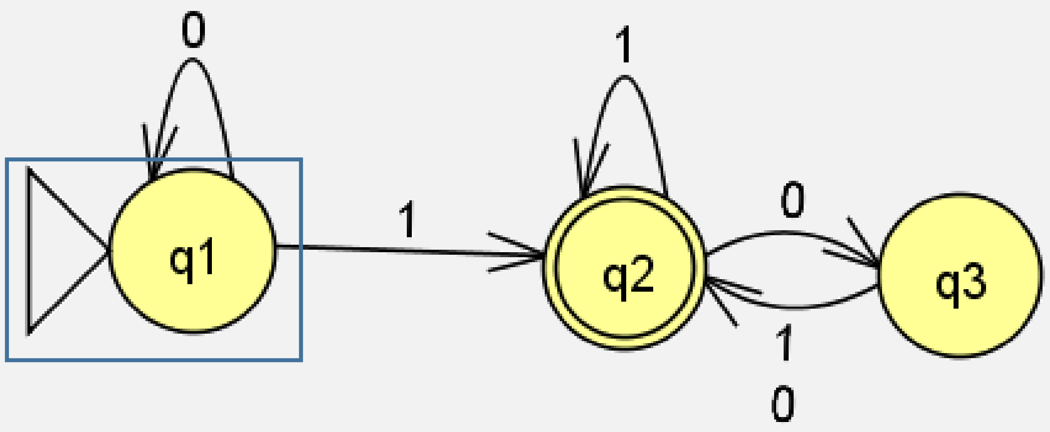
"If in this state" "Then go to this state"

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***.



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

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

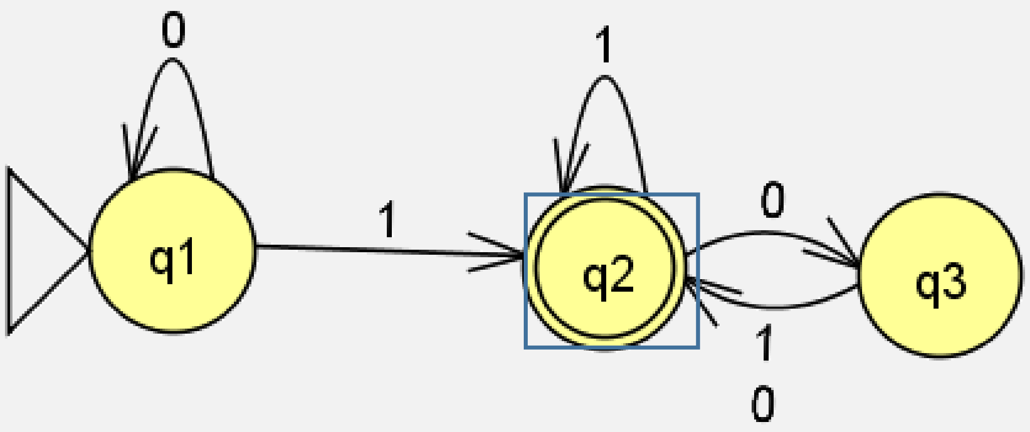
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

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



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

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***.

An **Example** (as **formal description**)

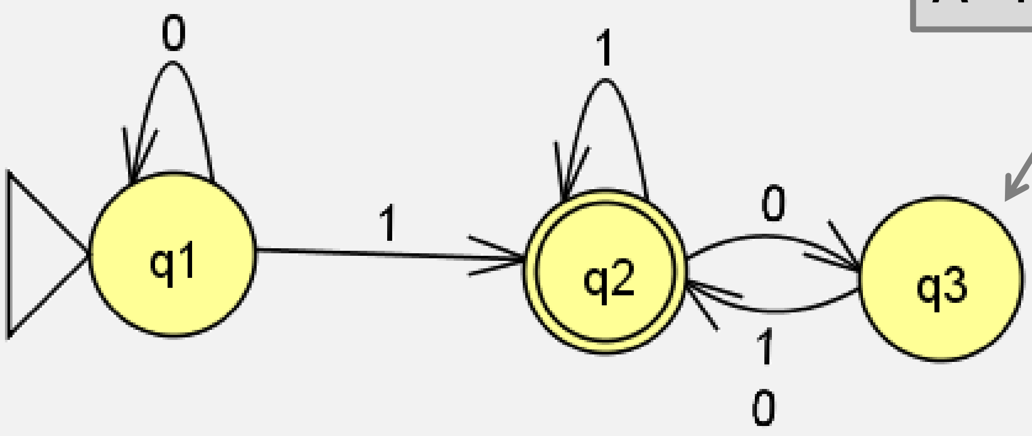
$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

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. $F = \{q_2\}$.

A "Program"



"Programming" Analogy

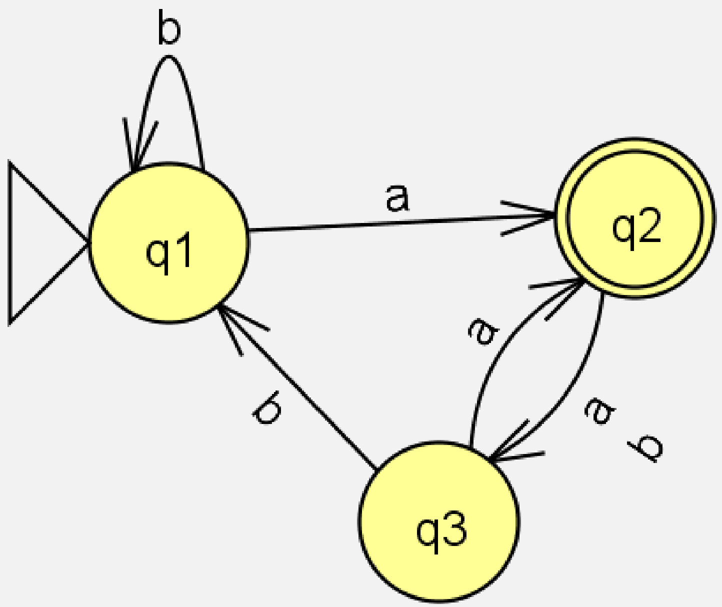
This "analogy" is meant to help your intuition

But it's important not to confuse with **formal definitions**.

Submit in-class work in gradescope!

In-class Exercise

Come up with a formal description of the following machine:



DEFINITION

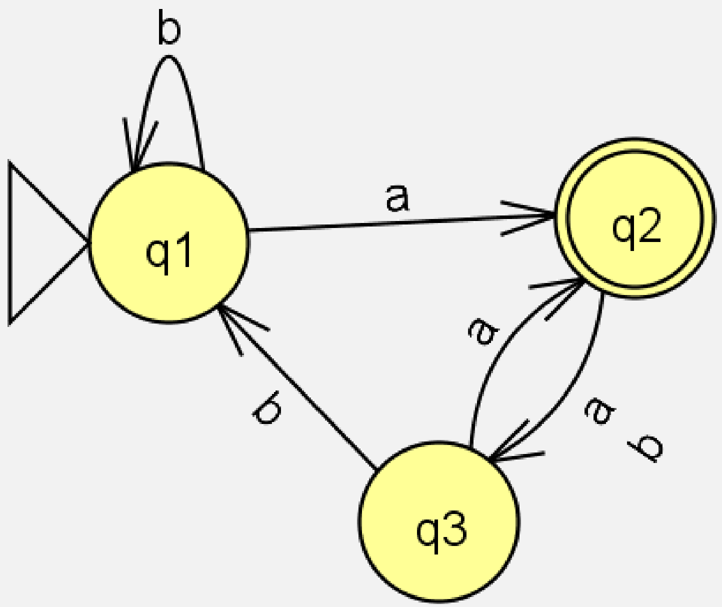
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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 **definitions** …

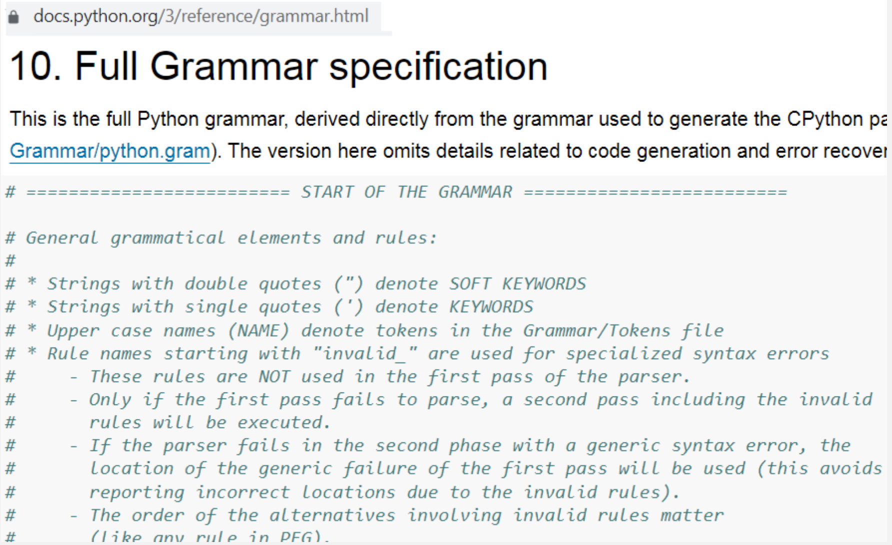
> e.g., A **Natural Number** is either
> - Zero
> - a Natural Number + 1

- And **rules** that describe how to **compute** with the **definitions** …

> To **add** two **Natural Numbers:**
> - Add the ones place of each num
> - Carry anything over 10
> - Repeat for each of remaining digits …

A Computation Model is … (from lecture 1)

- Some **definitions** …



docs.python.org/3/reference/grammar.html

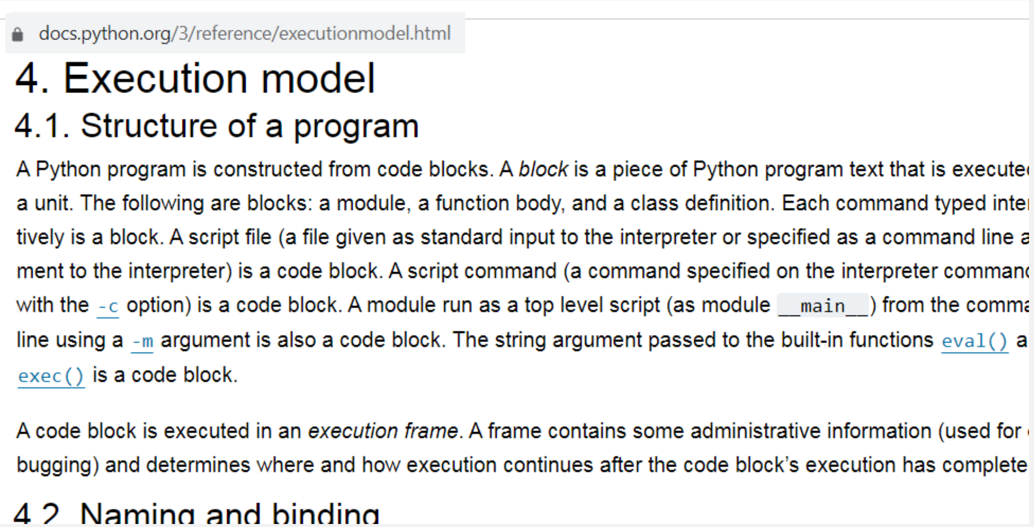
10. Full Grammar specification

This is the full Python grammar, derived directly from the grammar used to generate the CPython pa
Grammar/python.gram). The version here omits details related to code generation and error recover

```
# ======================== START OF THE GRAMMAR ========================

# General grammatical elements and rules:
#
# * Strings with double quotes (") denote SOFT KEYWORDS
# * Strings with single quotes (') denote KEYWORDS
# * Upper case names (NAME) denote tokens in the Grammar/Tokens file
# * Rule names starting with "invalid_" are used for specialized syntax errors
#     - These rules are NOT used in the first pass of the parser.
#     - Only if the first pass fails to parse, a second pass including the invalid
#       rules will be executed.
#     - If the parser fails in the second phase with a generic syntax error, the
#       location of the generic failure of the first pass will be used (this avoids
#       reporting incorrect locations due to the invalid rules).
#     - The order of the alternatives involving invalid rules matter
#       (like any rule in PEG).
```

- And **rules** that describe how to **compute** with the **definitions** …



docs.python.org/3/reference/executionmodel.html

4. Execution model

4.1. Structure of a program

A Python program is constructed from code blocks. A *block* is a piece of Python program text that is execute
a unit. The following are blocks: a module, a function body, and a class definition. Each command typed inte
tively is a block. A script file (a file given as standard input to the interpreter or specified as a command line a
ment to the interpreter) is a code block. A script command (a command specified on the interpreter comman
with the `-c` option) is a code block. A module run as a top level script (as module `__main__`) from the comma
line using a `-m` argument is also a code block. The string argument passed to the built-in functions `eval()` a
`exec()` is a code block.

A code block is executed in an *execution frame*. A frame contains some administrative information (used for
bugging) and determines where and how execution continues after the code block's execution has complete

4.2. Naming and binding

A Computation Model is … (from lecture 1)

- Some **definitions** …

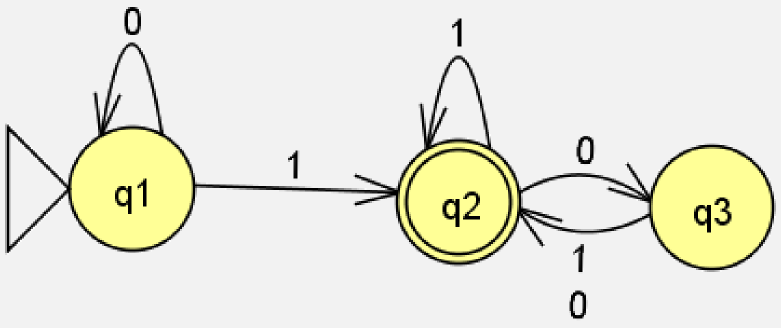
DEFINITION

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 describe how to **compute** with the **definitions** …

Computation with DFAs (JFLAP demo)

- DFA: 

- Input: “1101”

FSM Computation Rules

HINT: to better understand the math, always work out concrete examples

Informally

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

To run a computation / "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)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

Define variables $r_0, \ldots, r_n$, representing sequence of states in the computation

- $r_0 = q_0$

e.g., i=1, $r_1 = \delta(r_0, w_1)$

$r_2 = \delta(r_1, w_2)$ ...

- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$

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 $r_n \in F$

This is still a little verbose / informal