Finite Automata

Part Two

Recap from Last Time

Strings

- An ***alphabet*** is a finite set of symbols called ***characters***.
  - Typically, we use the symbol **Σ** to refer to an alphabet.
- A ***string over an alphabet Σ*** is a finite sequence of characters drawn from Σ.
- Example: If Σ = {**a**, **b**}, some valid strings over Σ include

  a

  aabaaabbabaaabaaaabbb

  abbababba

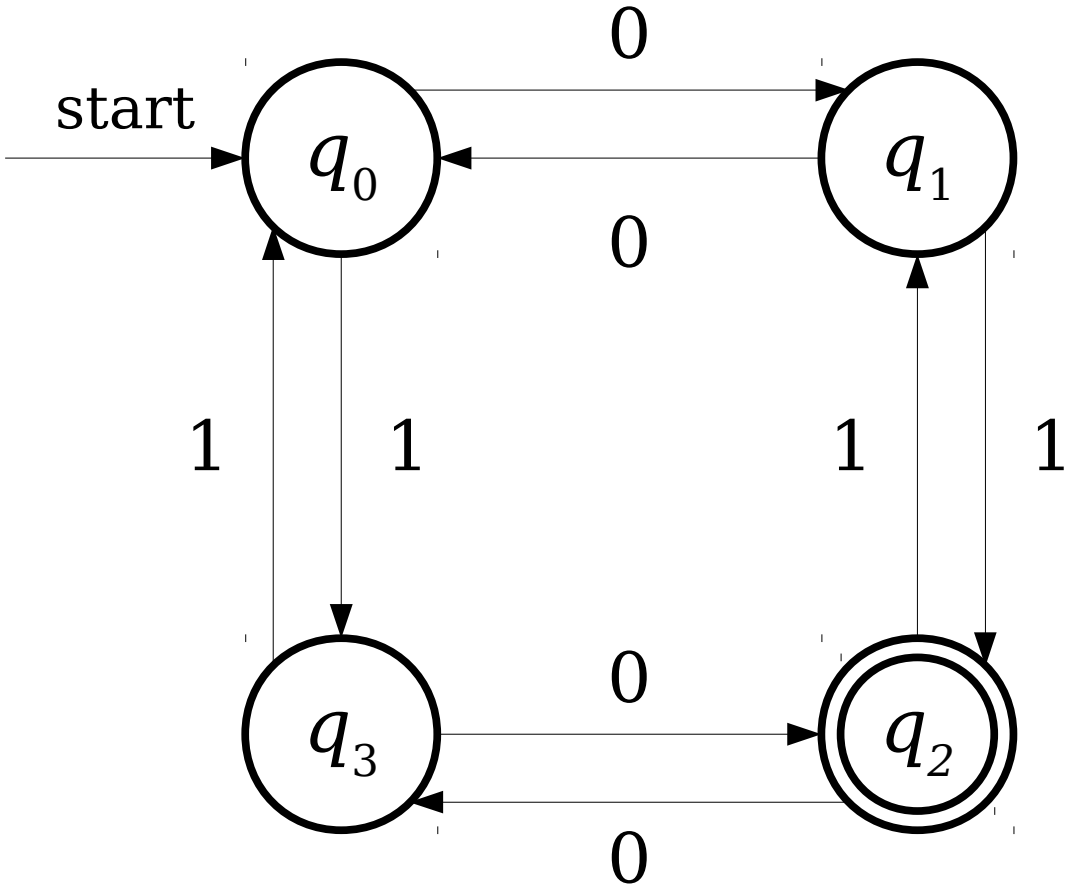
- The ***empty string*** contains no characters and is denoted **ε**.

Languages

- A ***formal language*** is a set of strings.
- We say that L is a ***language over Σ*** if it is a set of strings over Σ.
- Example: The language of palindromes over $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ is the set

  $\{\varepsilon, \texttt{a}, \texttt{b}, \texttt{c}, \texttt{aa}, \texttt{bb}, \texttt{cc}, \texttt{aaa}, \texttt{aba}, \texttt{aca}, \texttt{bab}, \ldots\}$
- The set of all strings composed from letters in Σ is denoted **Σ***.
- Formally: L is a language over Σ iff $L \subseteq \Sigma^*$.

A Simple Finite Automaton



0 1 0 1 1 0

The ***language of an automaton*** is the set of strings that it accepts.

If D is an automaton, we denote the language of D as $\mathscr{L}(D)$.

$$\mathscr{L}(D) = \{\ w \in \Sigma^* \mid D \text{ accepts } w\ \}$$

DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- DFAs are the simplest type of automaton that we will see in this course.

DFAs, Informally

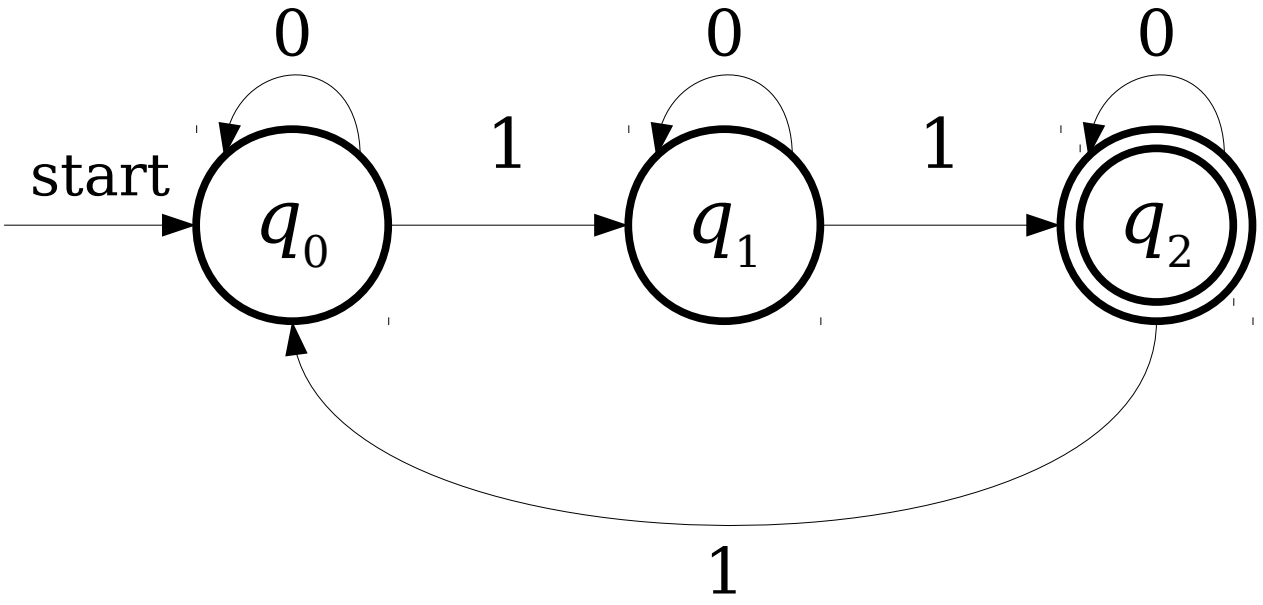
- A DFA is defined relative to some alphabet Σ.
- For each state in the DFA, there must be ***exactly one*** transition defined for each symbol in Σ.
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

Designing DFAs

- At each point in its execution, the DFA can only remember what state it is in.
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only finitely many different things the machine can remember.

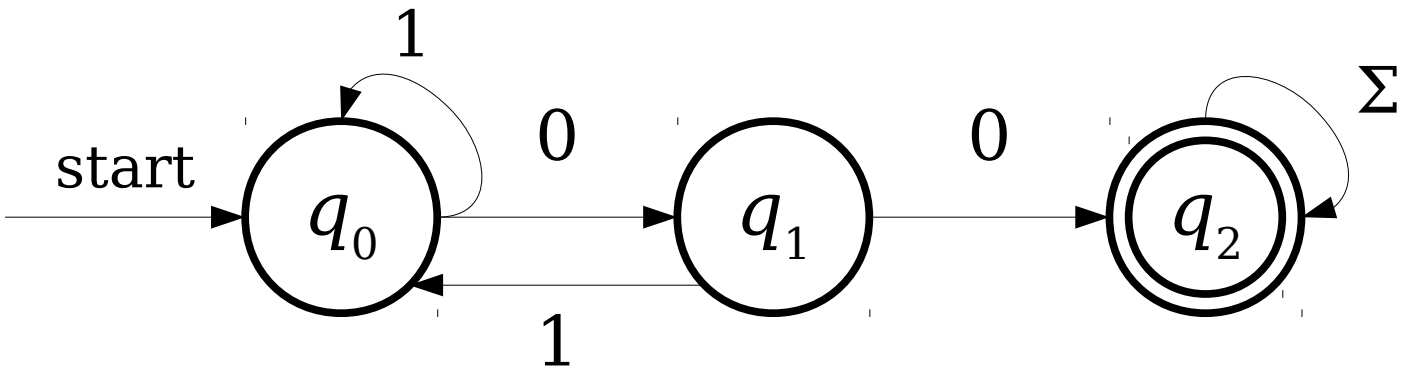
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* |$ the number of 1's in w is congruent to two modulo three $\}$



Recognizing Languages with DFAs

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ contains } \texttt{00} \text{ as a substring} \}$



New Stuff!

More Elaborate DFAs

$L = \{ w \in \Sigma^* \mid w \text{ is a C-style comment } \}$

Suppose the alphabet is

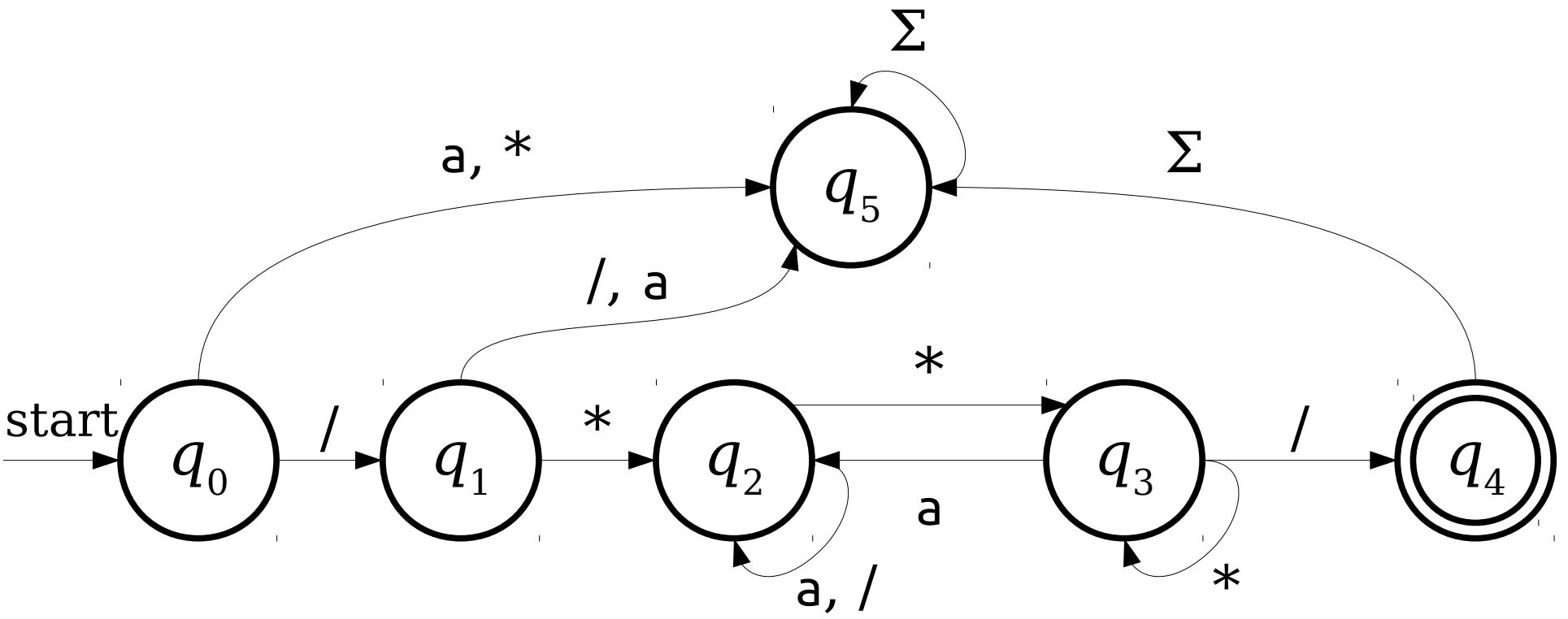
$\Sigma = \{$ a, *, / $\}$

Try designing a DFA for comments! Some test cases:

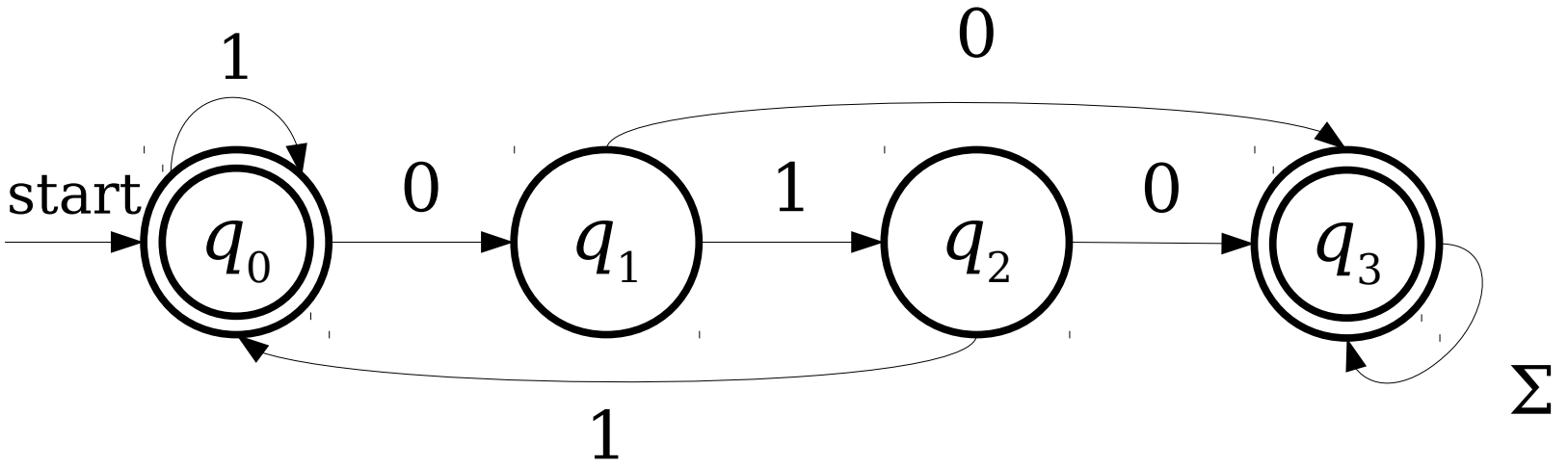
ACCEPTED	REJECTED
/*a*/	/**
/**/	/**/a/*aa*/
/***/	aaa/**/
/*aaa*aaa*/	/*/
/*a/a*/	/**a/

More Elaborate DFAs

$L = \{\ w \in \{\mathtt{a}, \mathtt{*}, \mathtt{/}\}^* \mid w \text{ represents a C-style comment}\ \}$



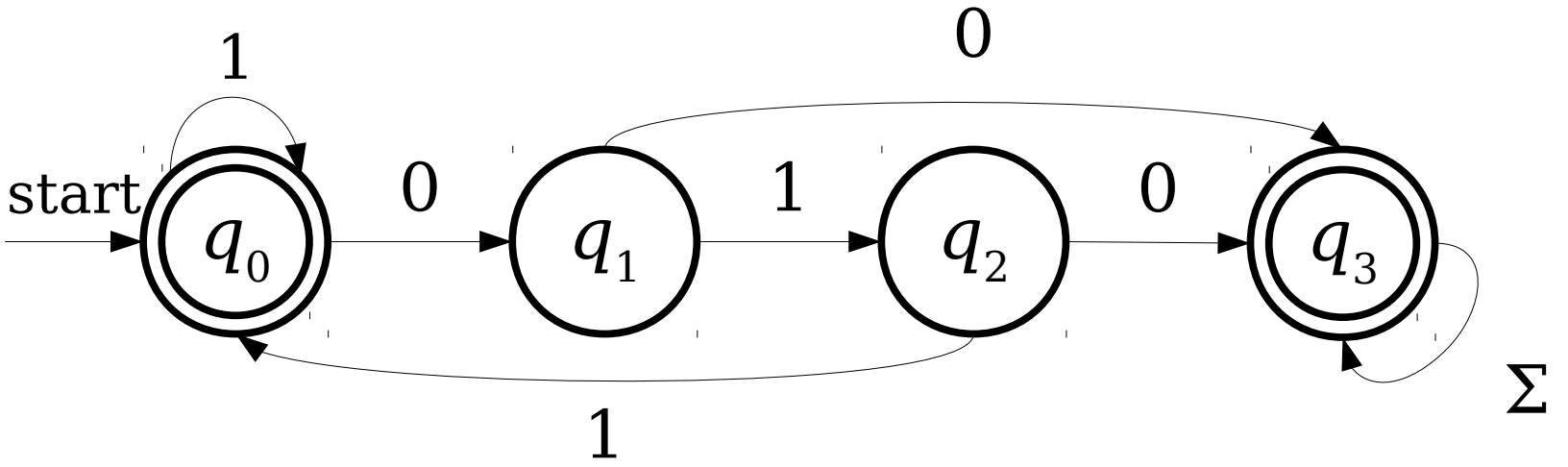
Tabular DFAs



	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

These stars indicate accepting states.

Tabular DFAs



	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

Since this is the first row, it's the start state.

Code? In a Theory Course?

```
int kTransitionTable[kNumStates][kNumSymbols] = {
     {0, 0, 1, 3, 7, 1, …},
      …
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    …
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input)
        state = kTransitionTable[state][ch];
    return kAcceptTable[state];
}
```

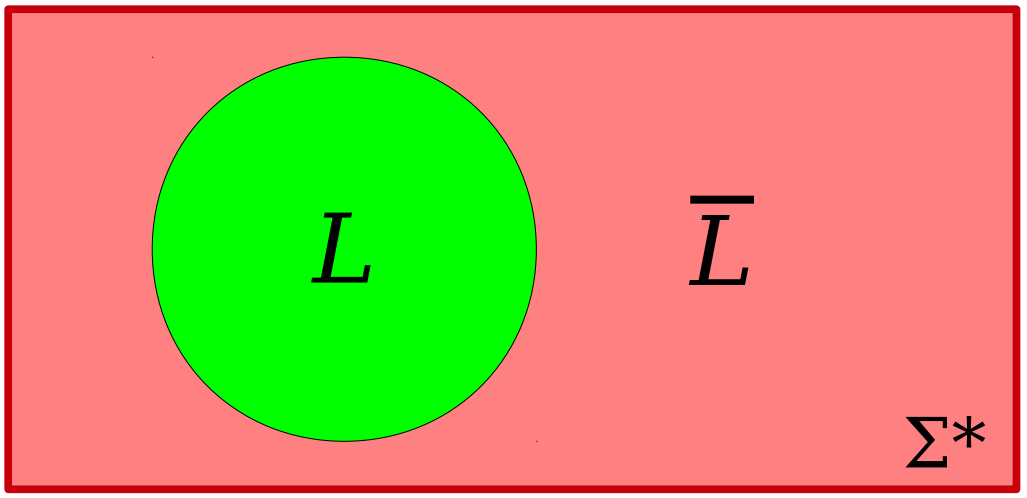
The Regular Languages

A language L is called a ***regular language*** if there exists a DFA D such that $\mathscr{L}(D) = L$.

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \Sigma^* - L$$

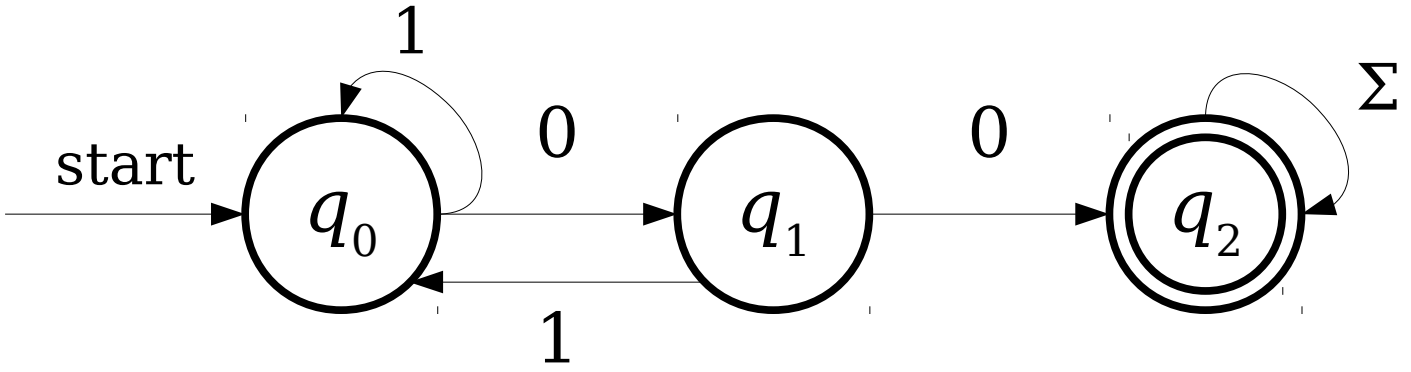


Complementing Regular Languages

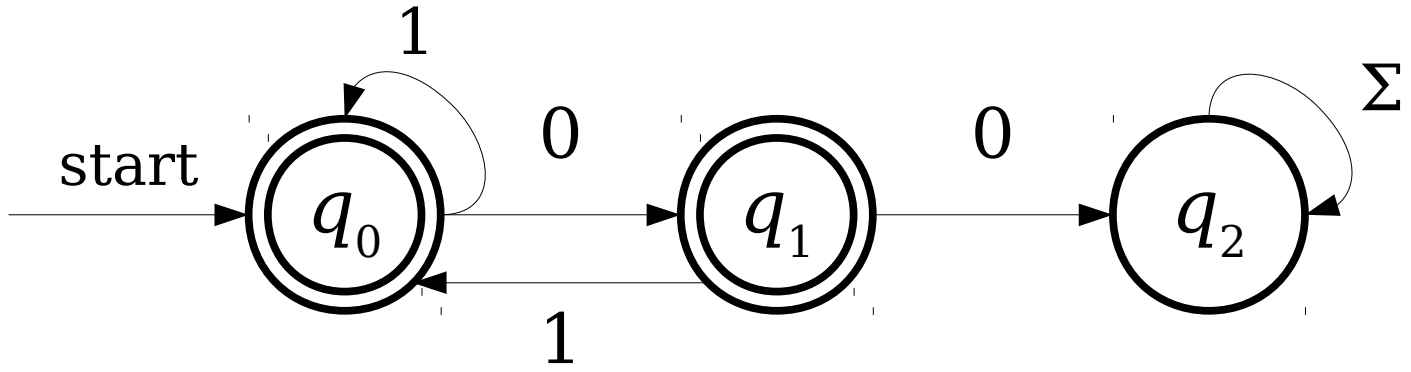
- Recall: A ***regular language*** is a language accepted by some DFA.
- ***Question:*** If L is a regular language, is $\overline{L}$ a regular language?
- If the answer is "yes," then there must be some way to construct a DFA for $\overline{L}$.
- If the answer is "no," then some language L can be accepted by a DFA, but $\overline{L}$ cannot be accepted by any DFA.

Complementing Regular Languages

$L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring} \}$

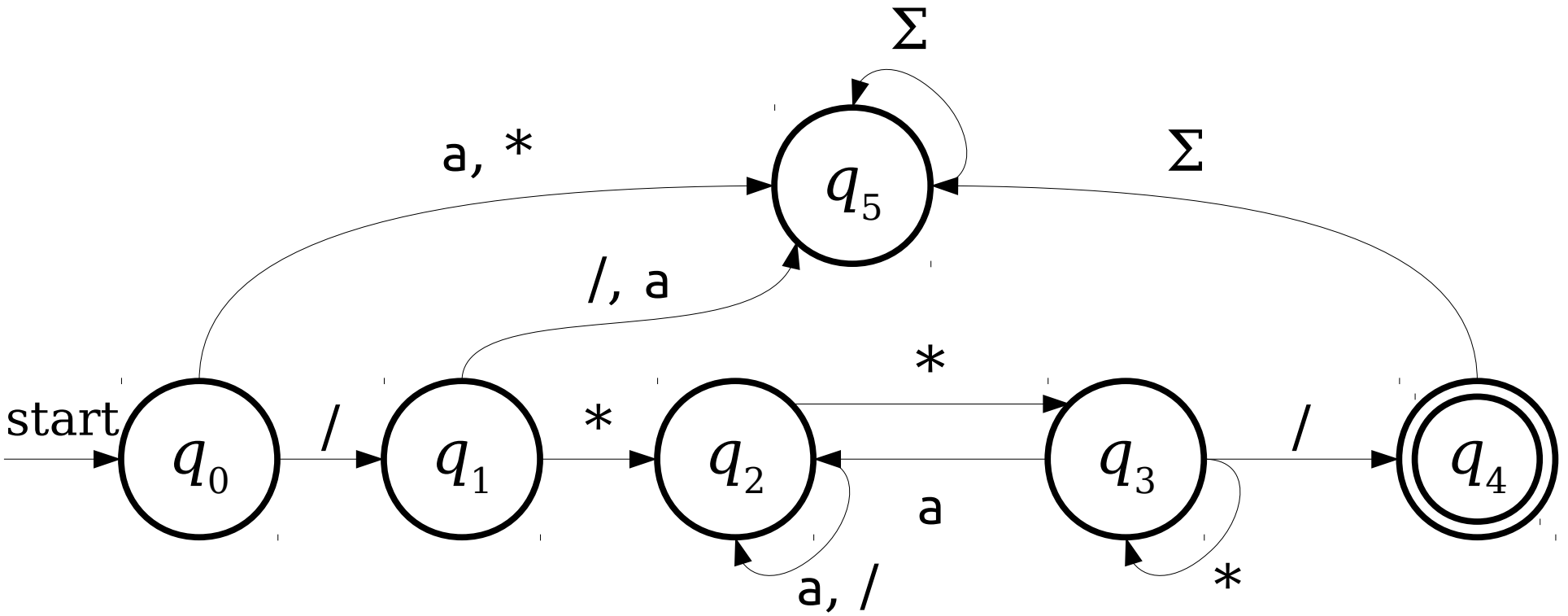


$\overline{L} = \{ w \in \{0, 1\}^* \mid w \textbf{\textit{ does not }} \text{contain } 00 \text{ as a substring} \}$



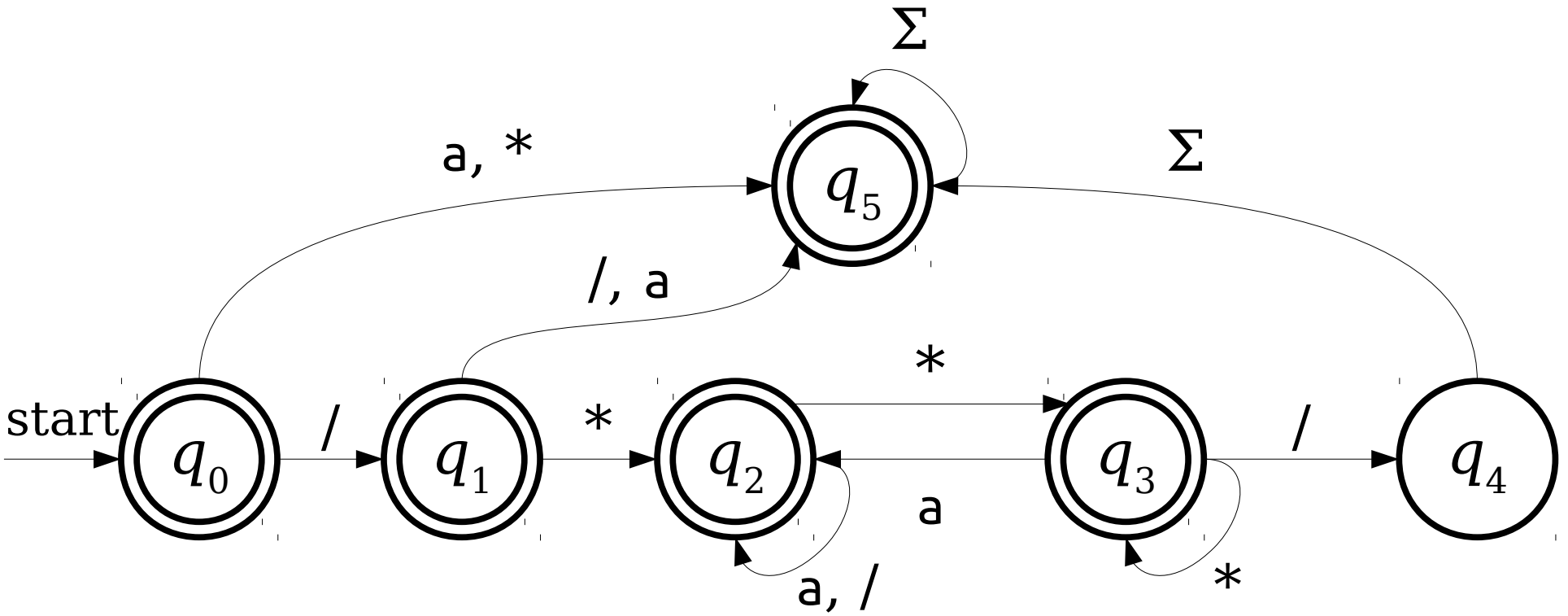
More Elaborate DFAs

$L = \{\ w \mid w$ is a C-style comment $\}$



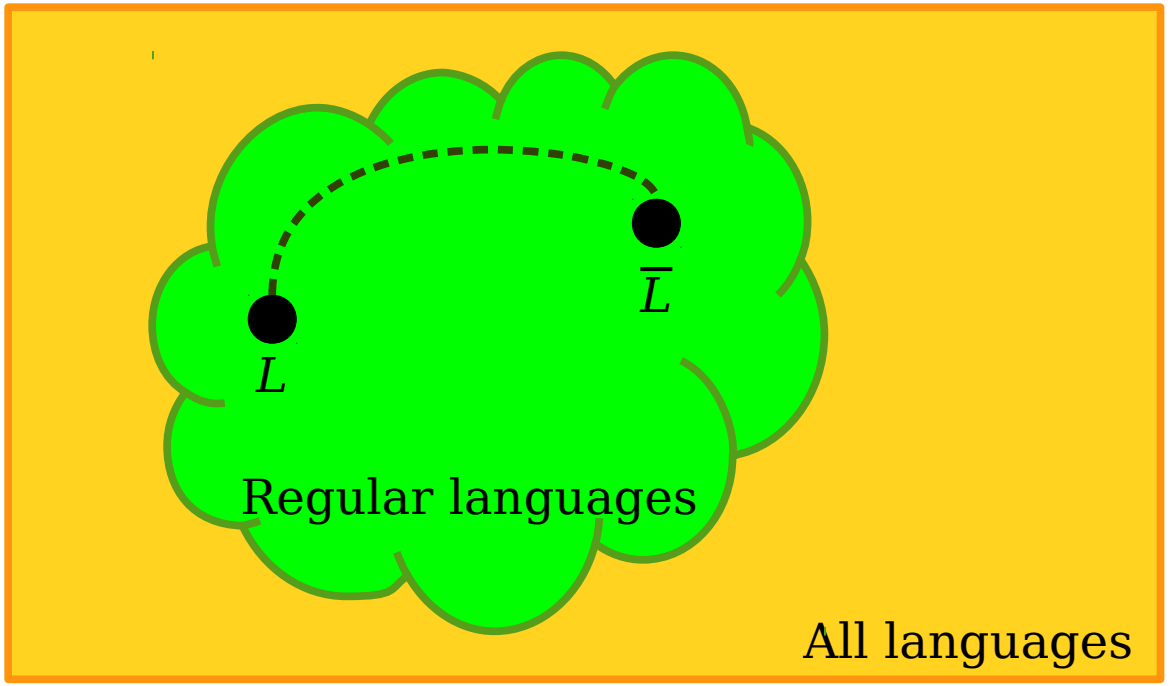
More Elaborate DFAs

$\overline{L} = \{\ w \mid w$ is ***not*** a C-style comment $\}$



Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are ***closed under complementation***.



Time-Out For Announcements!

Midterms!

- You're done with the midterm! Woohoo!
- We're going to grade them on Saturday and try to have them ready to return on Monday.
- We'll release solutions along with statistics. However, if you'd like to ask us about individual questions on the exam in the meantime, feel free to do so.

Problem Set Five

- Problem Set Five is due on Friday.
- Feel free to stop by office hours with questions or to ask over on Piazza.
- Check out the “Guide to Inductive Proofs” handout we released on Monday. It's got a bunch of useful information!

Your Questions

“You've mentioned multiple times that you wished you'd taken more humanities/social science classes when you were an undergrad. If you could next quarter, which ones would you take?”

A few come to mind:

Ethics and Politics of Public Service (highly cross-listed)

Negotiation (also highly cross-listed)

Global Human Geography (HISTORY 106A/B)

Fiction Writing (English 90)

“What's the hardest problem you've solved?”

I'm worried that this framing restricts me to math and CS-type problems, since most interesting “problems” out there can't be “solved” by anyone. In that vein, I think the hardest problem I ever solved was the one on my CS106X final where Julie made a typo that turned a relatively easy problem into a really hard one. ☺

I'm working on a lot of different things that are hard - making CS more inclusive, making CS easier to learn, making hard CS topics easier to understand, etc. - but I definitely haven't solved them. I'm more proud of the work I've done there, though, than of any problem I've worked on that can definitively be solved.

“Why do we say if and only if, instead of just only if?”

The statement “P only if Q,” at least in mathematics, means “P implies Q.” In plain English, “only if” feels like it means “precisely when,” but for whatever reason we don't do that in math. C'est la vie.

For example, the statement “a graph is planar only if it's four-colorable” just means that every planar graph is four-colorable. There are plenty of graphs that aren't planar even though they're four-colorable.

Back to CS103!

NFAs

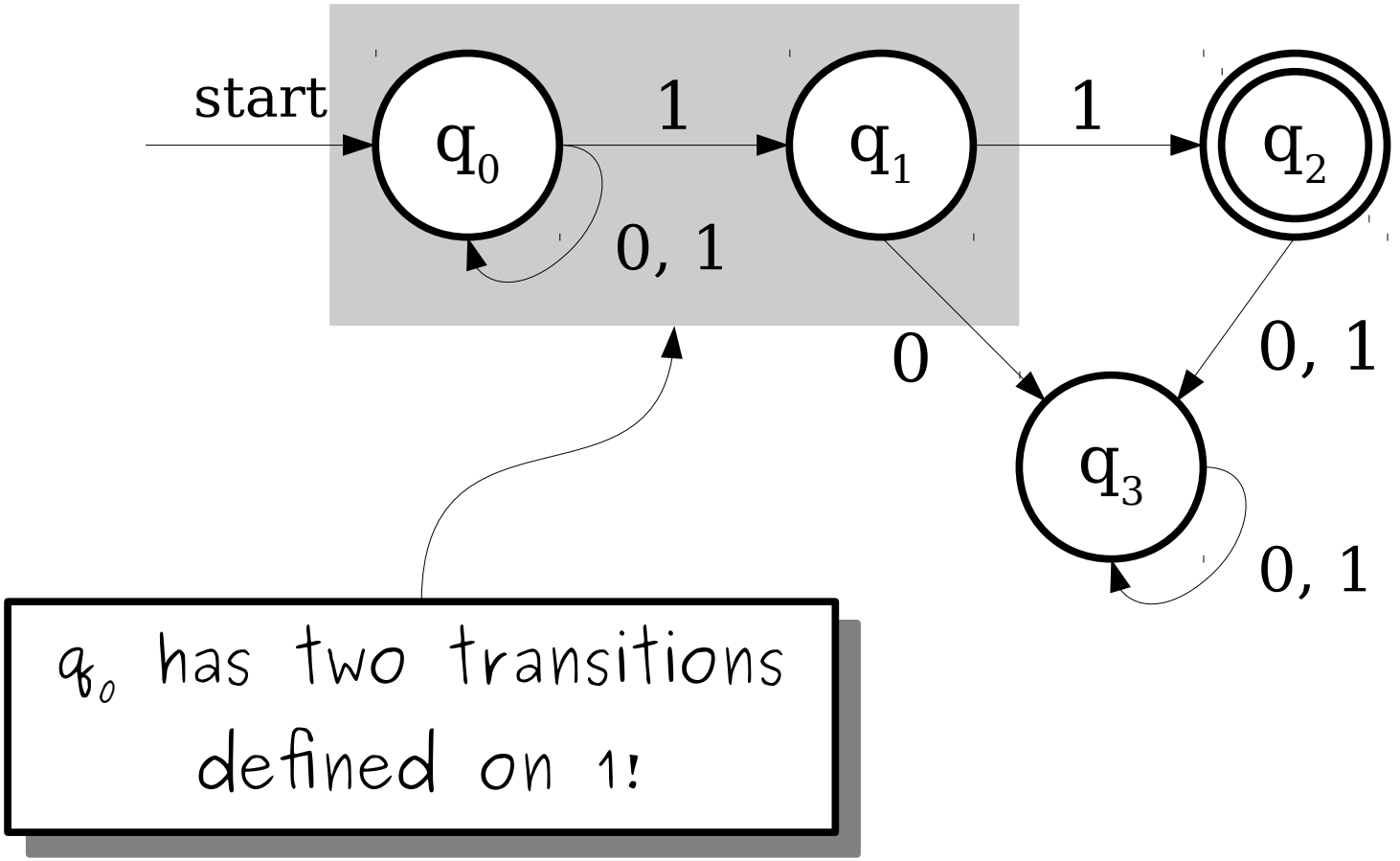
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.

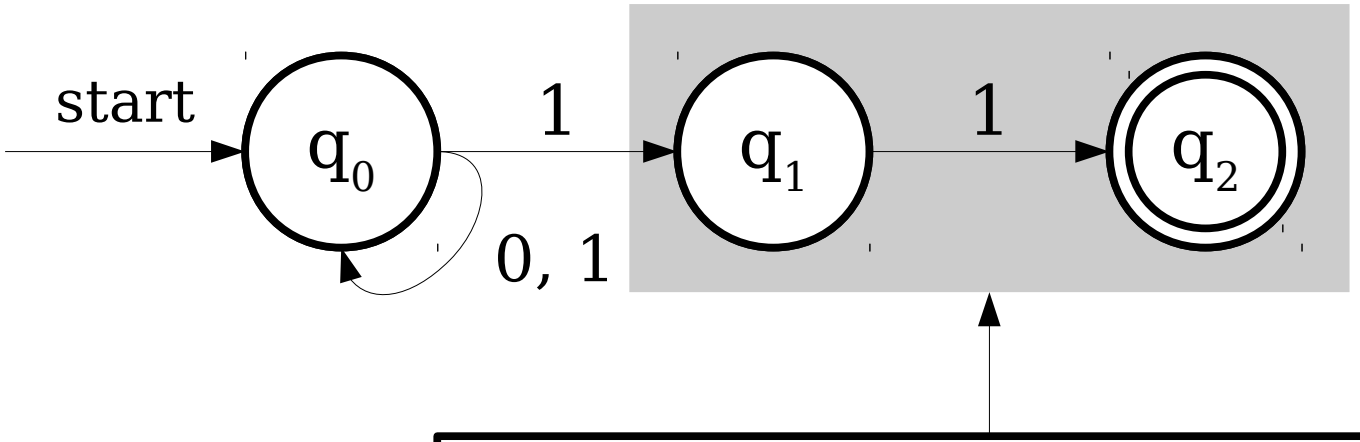
(Non)determinism

- A model of computation is ***deterministic*** if at every point in the computation, there is exactly one choice that can make.
- The machine accepts if that series of choices leads to an accepting state.
- A model of computation is ***nondeterministic*** if the computing machine may have multiple decisions that it can make at one point.
- The machine accepts if ***any*** series of choices leads to an accepting state.

A Simple NFA

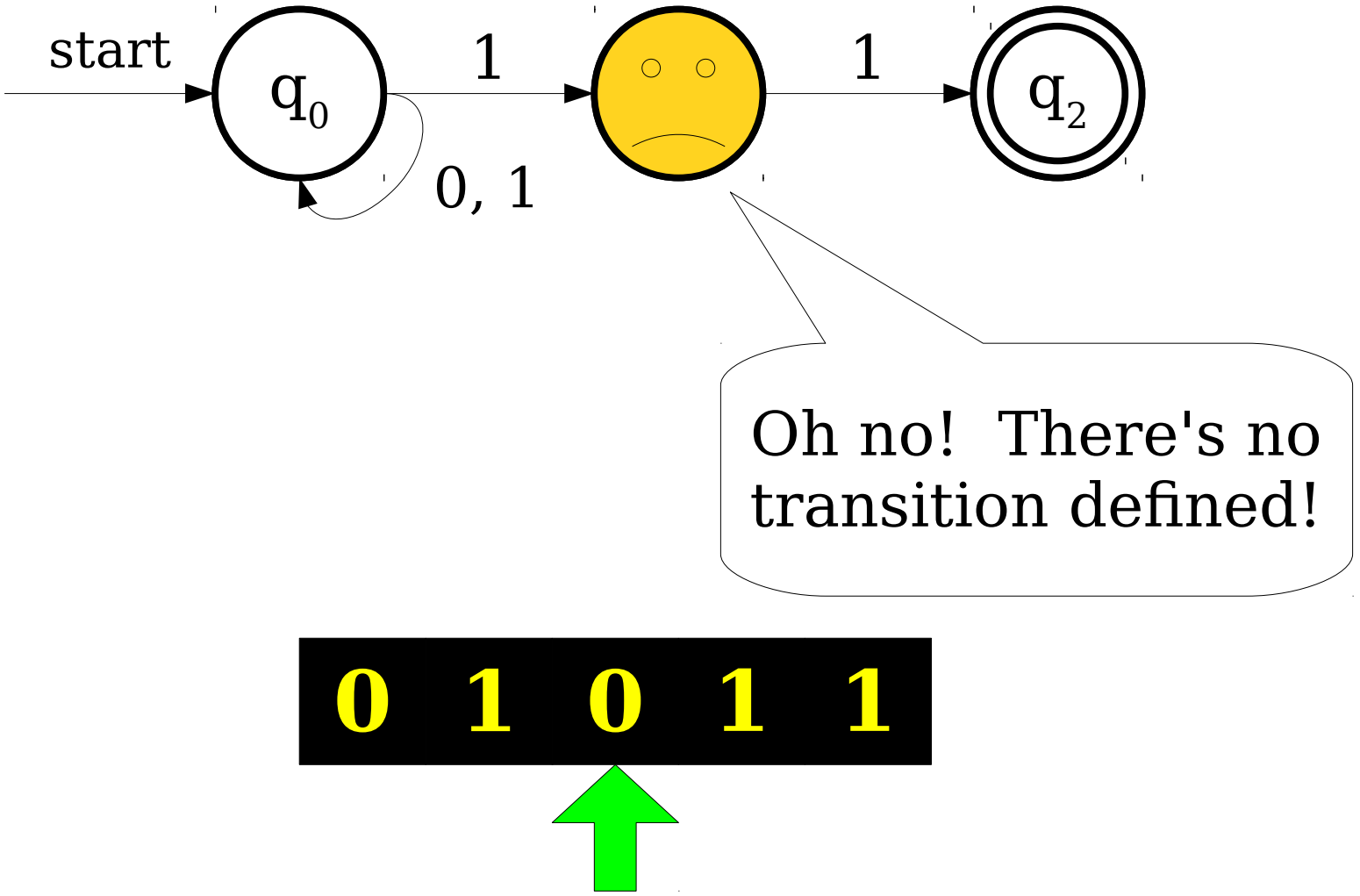

start
q_0
q_1
q_2
q_3
1
1
0, 1
0
0, 1
0, 1
q_0 has two transitions defined on 1!

A More Complex NFA



If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.

A More Complex NFA



NFA Acceptance

- An NFA N accepts a string w if there is some series of choices that lead to an accepting state.
- Let $LeadsToAccept(N, c, w)$ mean "the series of choices c takes N into an accept state when run on w."
- Then

$$\boldsymbol{N \text{ accepts } w \leftrightarrow \exists c.\ LeadsToAccept(N, c, w)}$$

- Consequently,

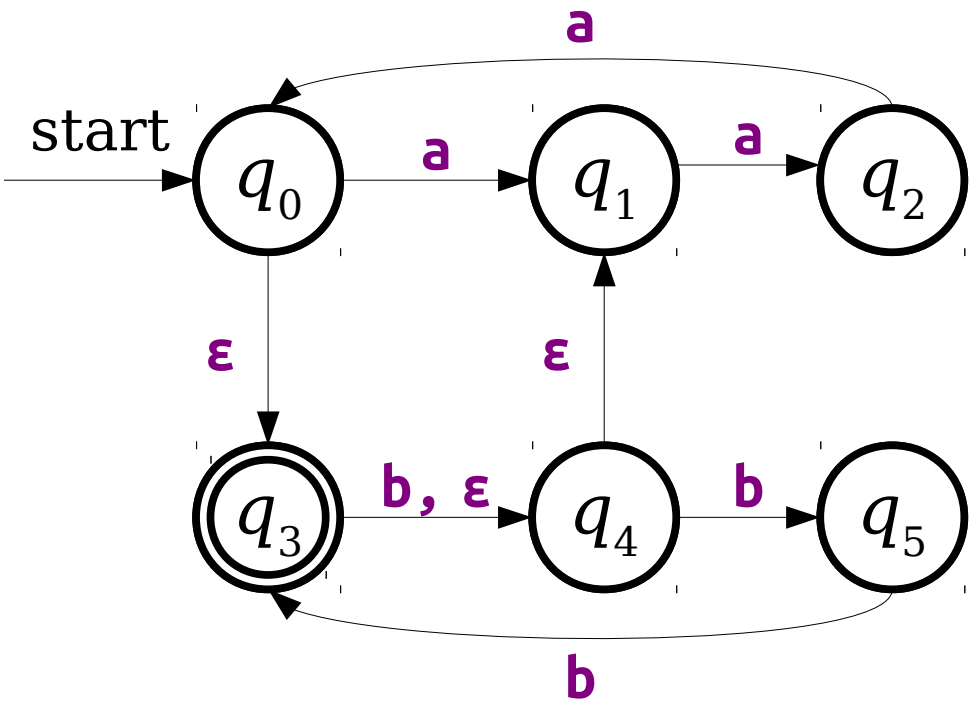
$$\boldsymbol{N \text{ rejects } w \leftrightarrow \forall c.\ \neg LeadsToAccept(N, c, w)}$$

ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



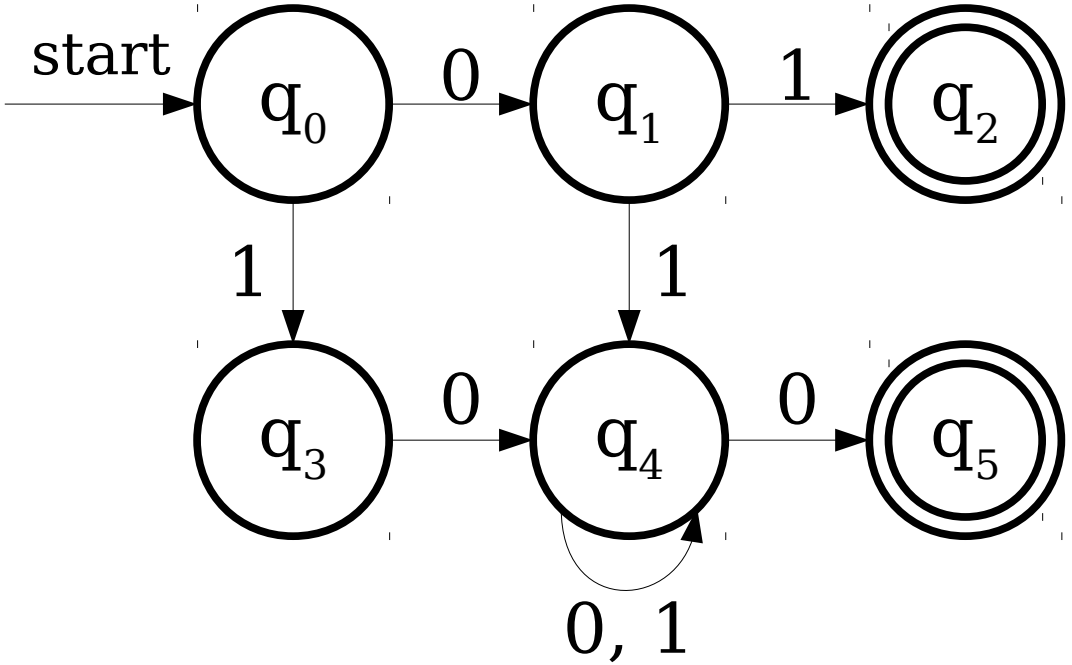
ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
- NFAs are not *required* to follow ε-transitions. It's simply another option at the machine's disposal.

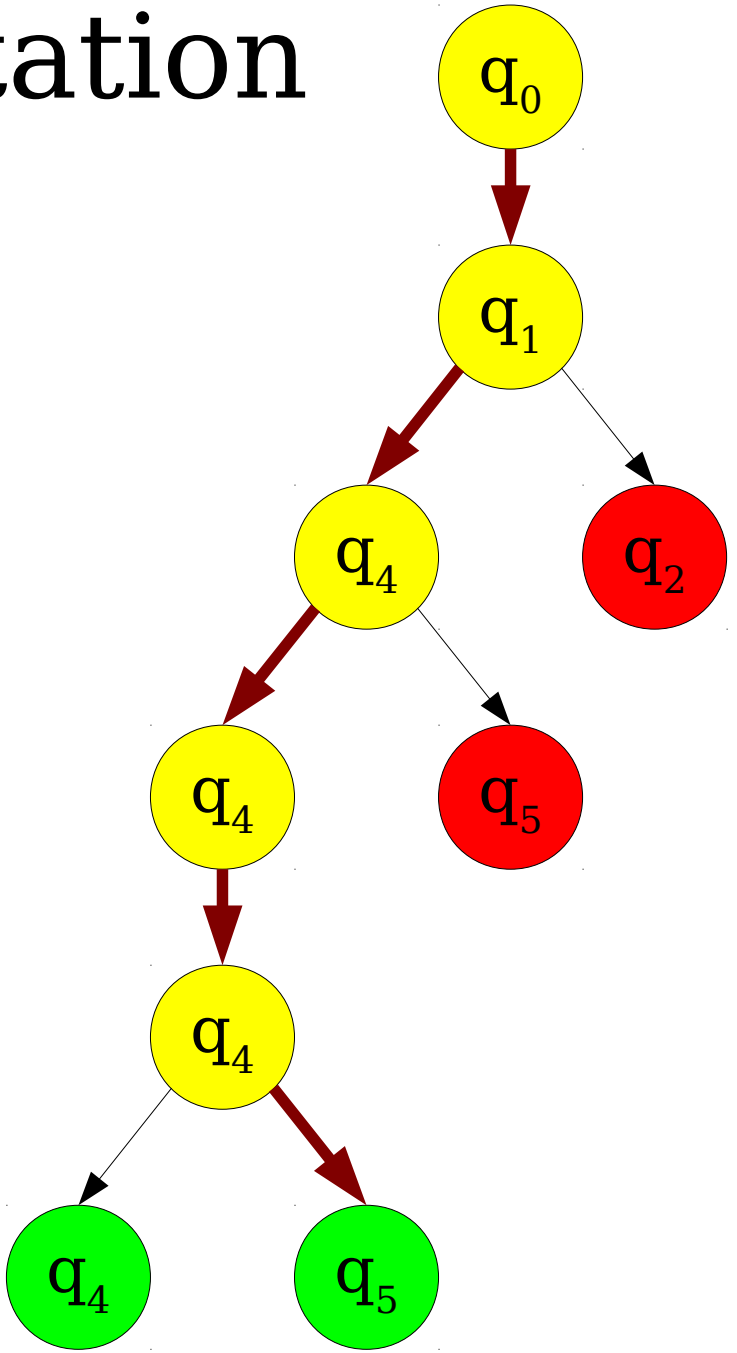
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers.
- How can we build up an intuition for them?
- Three approaches:
  - **Tree computation**
  - **Perfect guessing**
  - **Massive parallelism**

Tree Computation



0 1 0 1 0



Nondeterminism as a Tree

- At each decision point, the automaton clones itself for each possible decision.
- The series of choices forms a directed, rooted tree.
- At the end, if any active accepting states remain, we accept.

Perfect Guessing

- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess the correct choice of moves to make.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- No known physical analog for this style of computation – this is totally new!

Massive Parallelism

- An NFA can be thought of as a DFA that can be in many states at once.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Nondeterministic machines can be thought of as machines that can try any number of options in parallel.

So What?

- We will turn to these three intuitions for nondeterminism more later in the quarter.
- Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:
  - Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?
  - Can any problem that can be solved by a nondeterministic machine be solved ***efficiently*** by a deterministic machine?
- The answers vary from automaton to automaton.

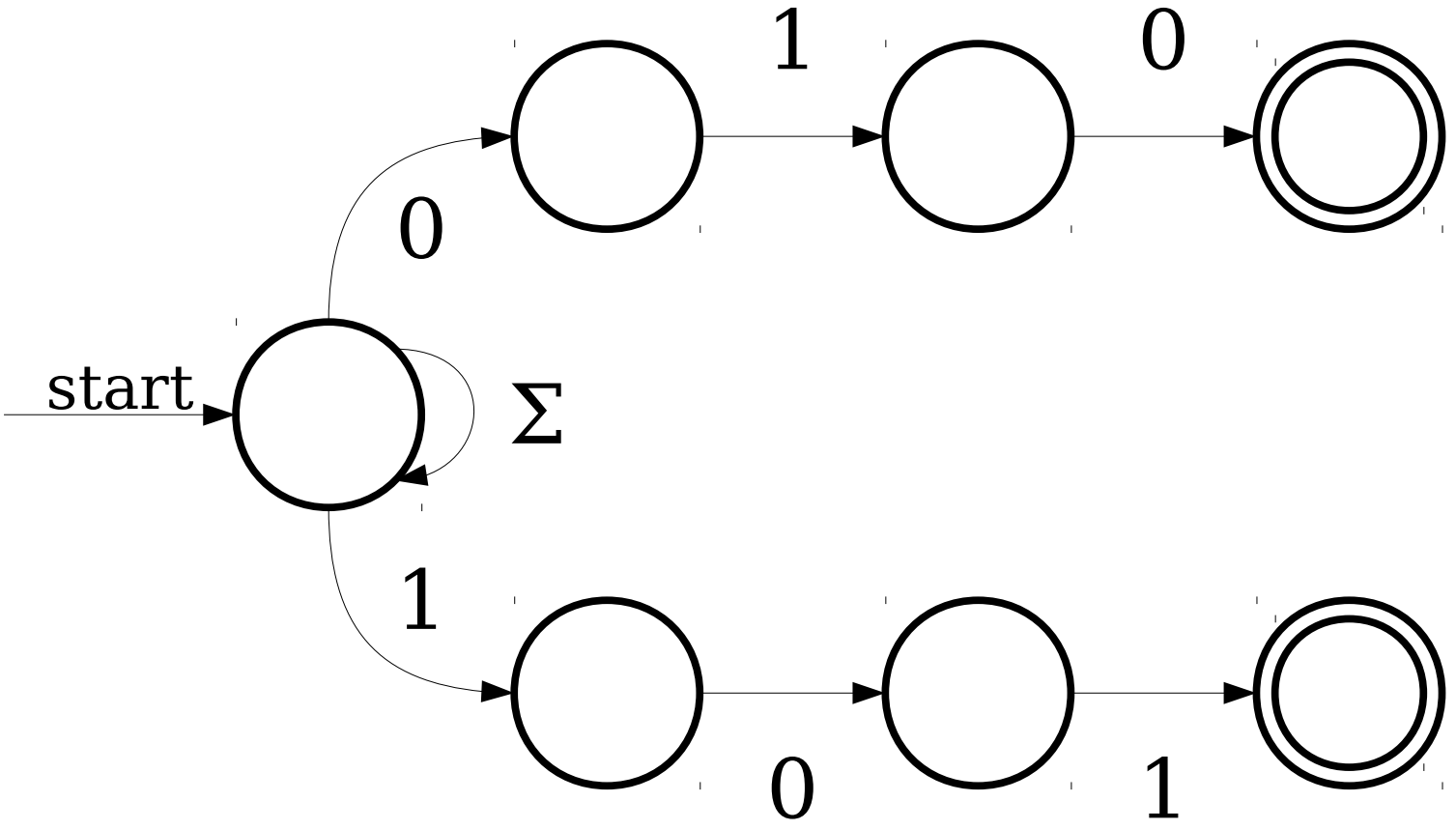
Designing NFAs

Designing NFAs

- When designing NFAs, *embrace the nondeterminism!*
- Good model: ***Guess-and-check***:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase corresponds to trying lots of different options.
- The *check* phase corresponds to filtering out bad guesses or wrong options.

Guess-and-Check

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101} \}$



Guess-and-Check

$L = \{ w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$