Finite Automata

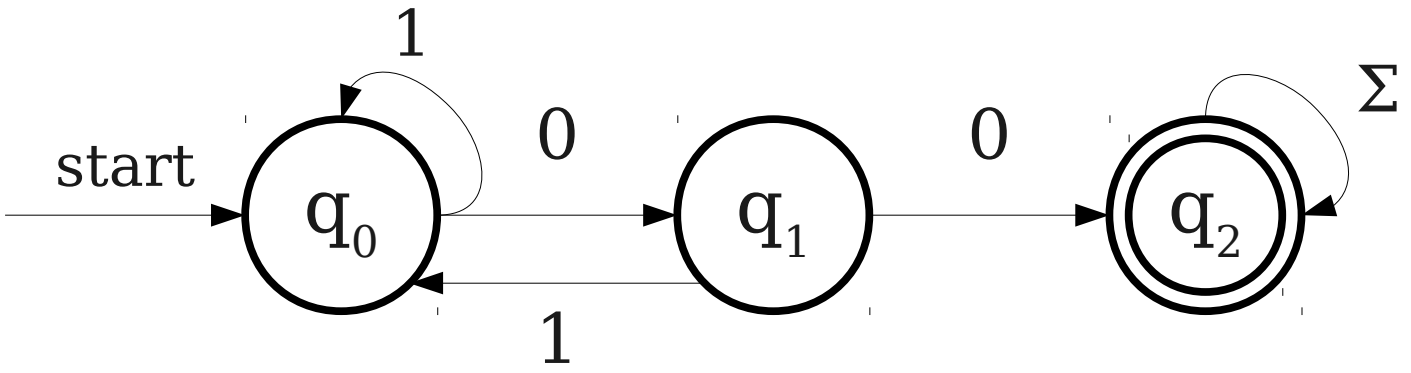
Part Two

DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- 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 the alphabet.
- There is a unique start state.
- There are zero or more accepting states.

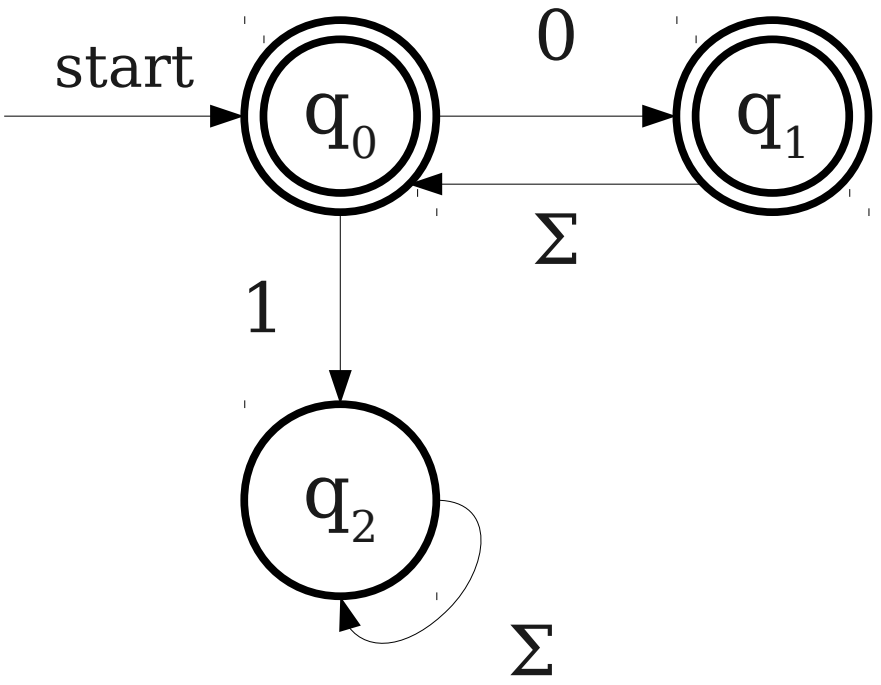
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* |$ every other character of w, starting with the first character, is 0 $\}$



More Elaborate DFAs

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

Suppose the alphabet is

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

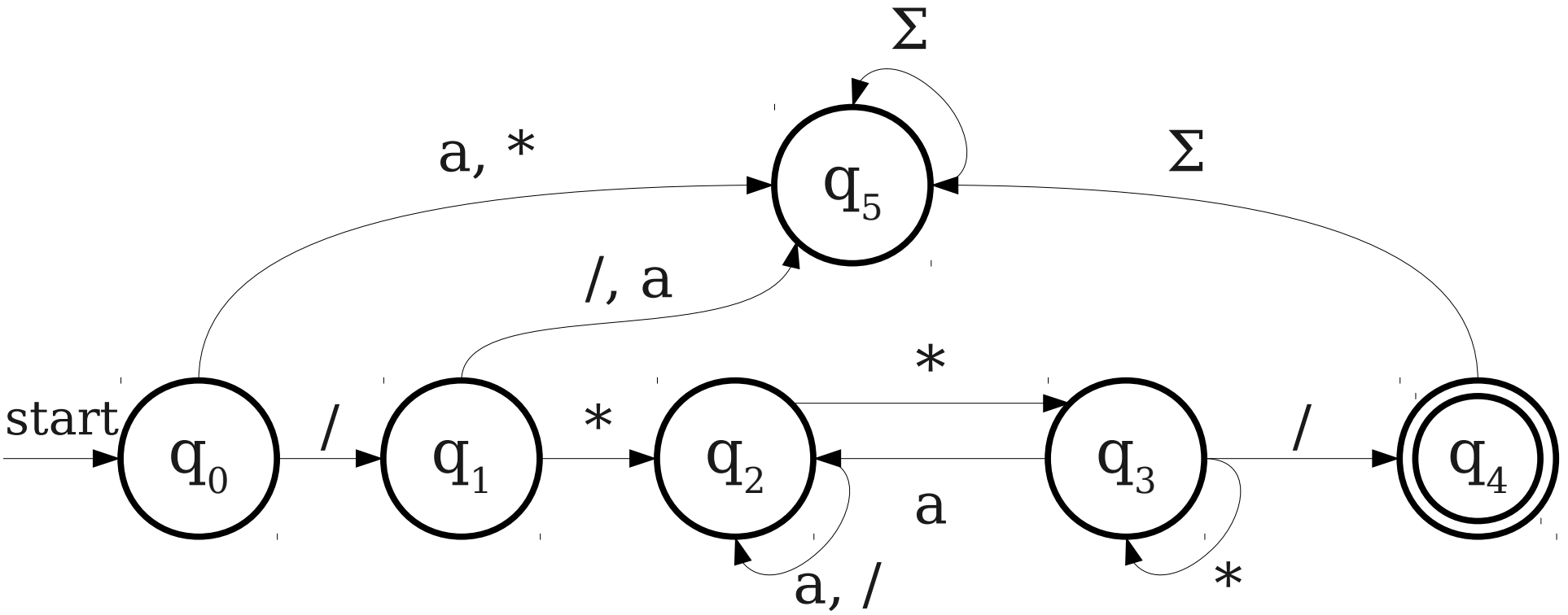
Try designing a DFA for comments!

Some test cases:

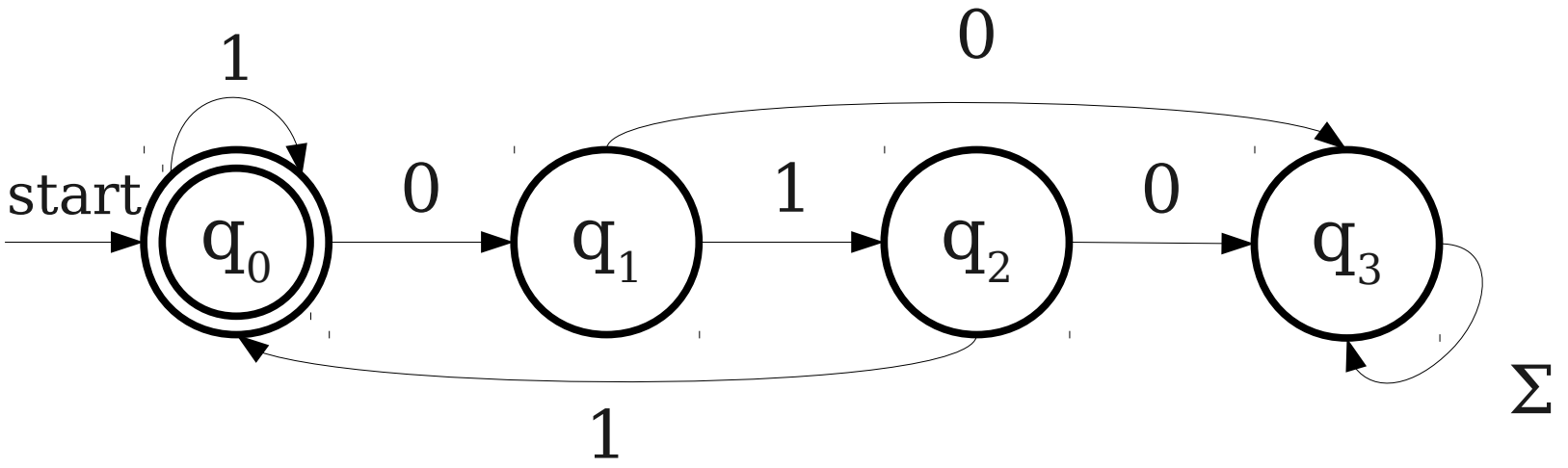
ACCEPTED	REJECTED
/*a*/	/**
/**/	/**/a
/***/	aaa/**/
/*aaa*aaa*/	/*/

More Elaborate DFAs

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



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

The star indicates that this is an accepting state.

Code? In a Theory Course?

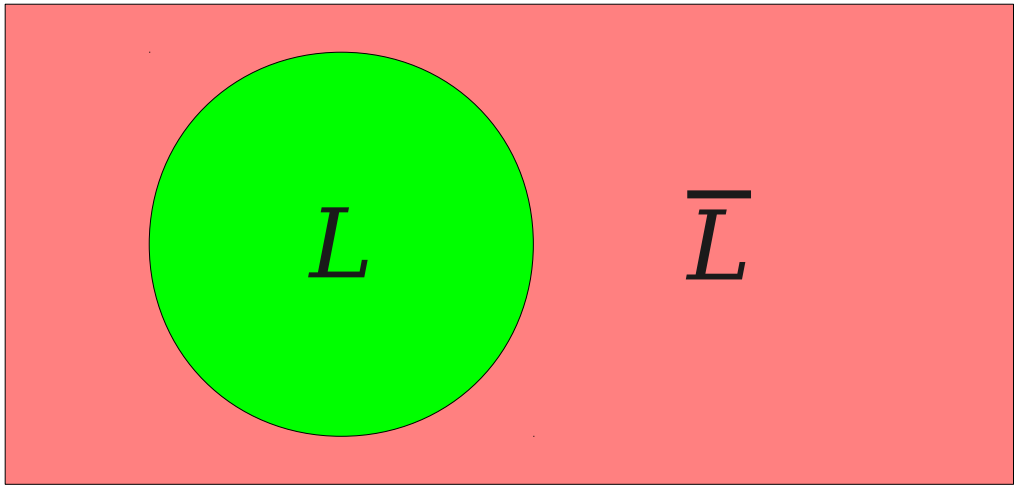
```
int kTransitionTable[kNumStates][kNumSymbols] = {
     {0, 0, 1, 3, 7, 1, …},
      …
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    …
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input)
        state = kTransitionTable[state][ch];
    return kAcceptTable[state];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

A language L is called a **regular language** iff 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:

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

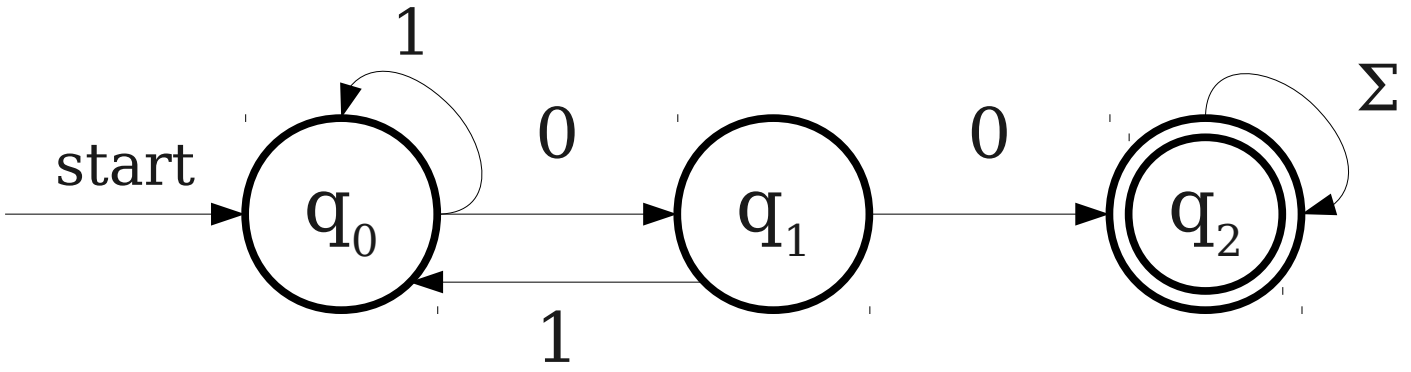


Complementing Regular Languages

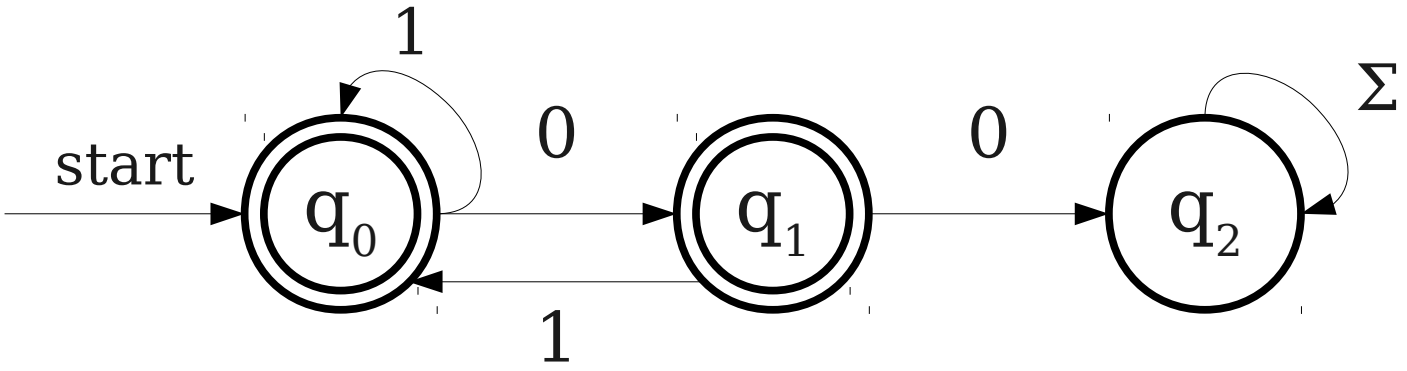
- Recall: A **regular language** is a language accepted by some DFA.
- **Question:** If L is a regular language, is $\overline{L}$ a regular language?
- If the answer is "yes," then there must be some way to construct a DFA for $\overline{L}$.
- If the answer is "no," then some language L can be accepted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

Complementing Regular Languages

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

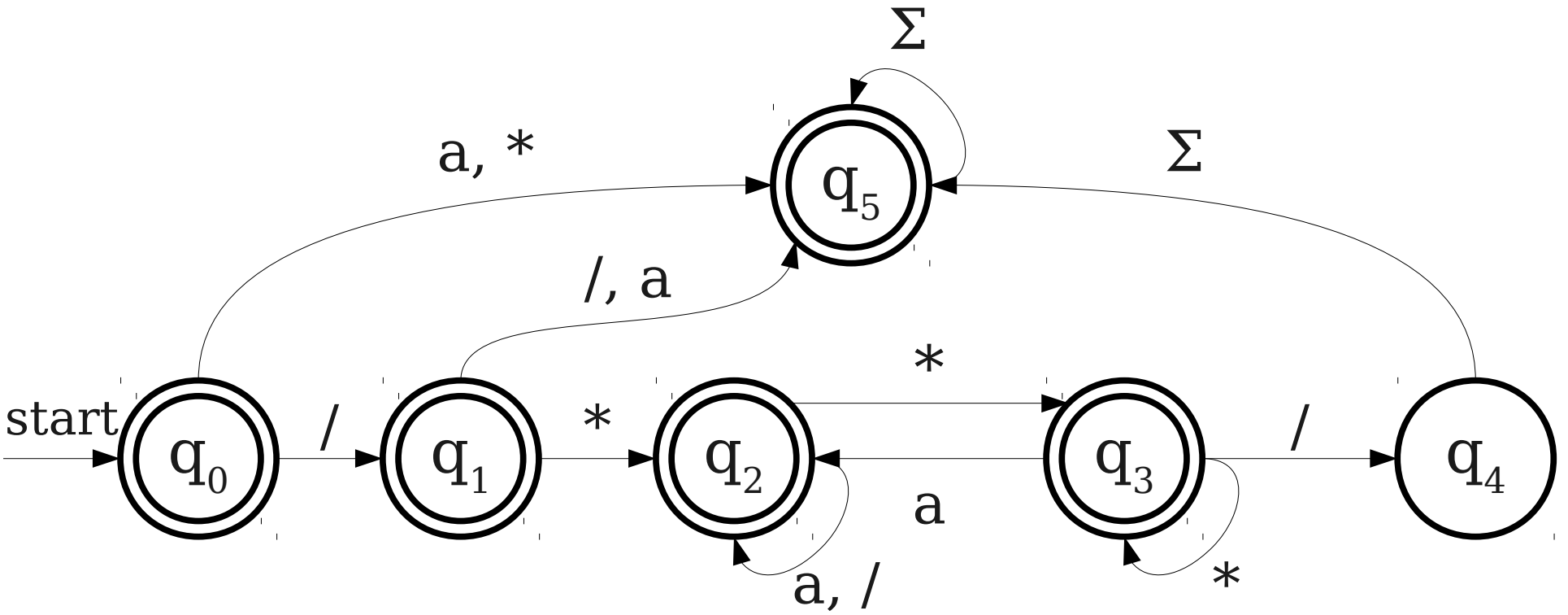


$\overline{L} = \{ w \in \{0, 1\}^* \mid w \textbf{ does not } \text{contain } 00 \text{ as a substring} \}$



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.
- If we begin with a regular language and complement it, we end up with a regular language.
- This is an example of a **closure property of regular languages**.
  - The regular languages are **closed under complementation**.
  - We'll see more such properties later on.

Time-Out For Announcements!

Solutions Released

- Solutions released for
  - Problem Set 4 checkpoint.
  - Practice Midterm 1.
  - Practice Midterm 2.
- ***Please review the solution sets for the Problem Set 4 checkpoint***. There are some common mistakes you probably want to review.

Midterm Logistics

- Midterm review sessions this weekend.
  - **Saturday, 2:15PM – 4:15PM:** Michael holding an on-campus review session.
  - **Sunday, 8PM – 10PM:** Dilli holding a review session for SCPD students.
- Neither time works for you? Let us know so we can try to schedule something extra!
- We have a Google Moderator page to see what questions to answer during the review session; link will be posted later today.

Your Questions

“The only difference between predicates and functions is that the first 'returns' booleans and the second 'returns' objects. Are booleans not objects? If they aren't, why aren't they, and if they are, how are predicates and functions different?”

NFAs

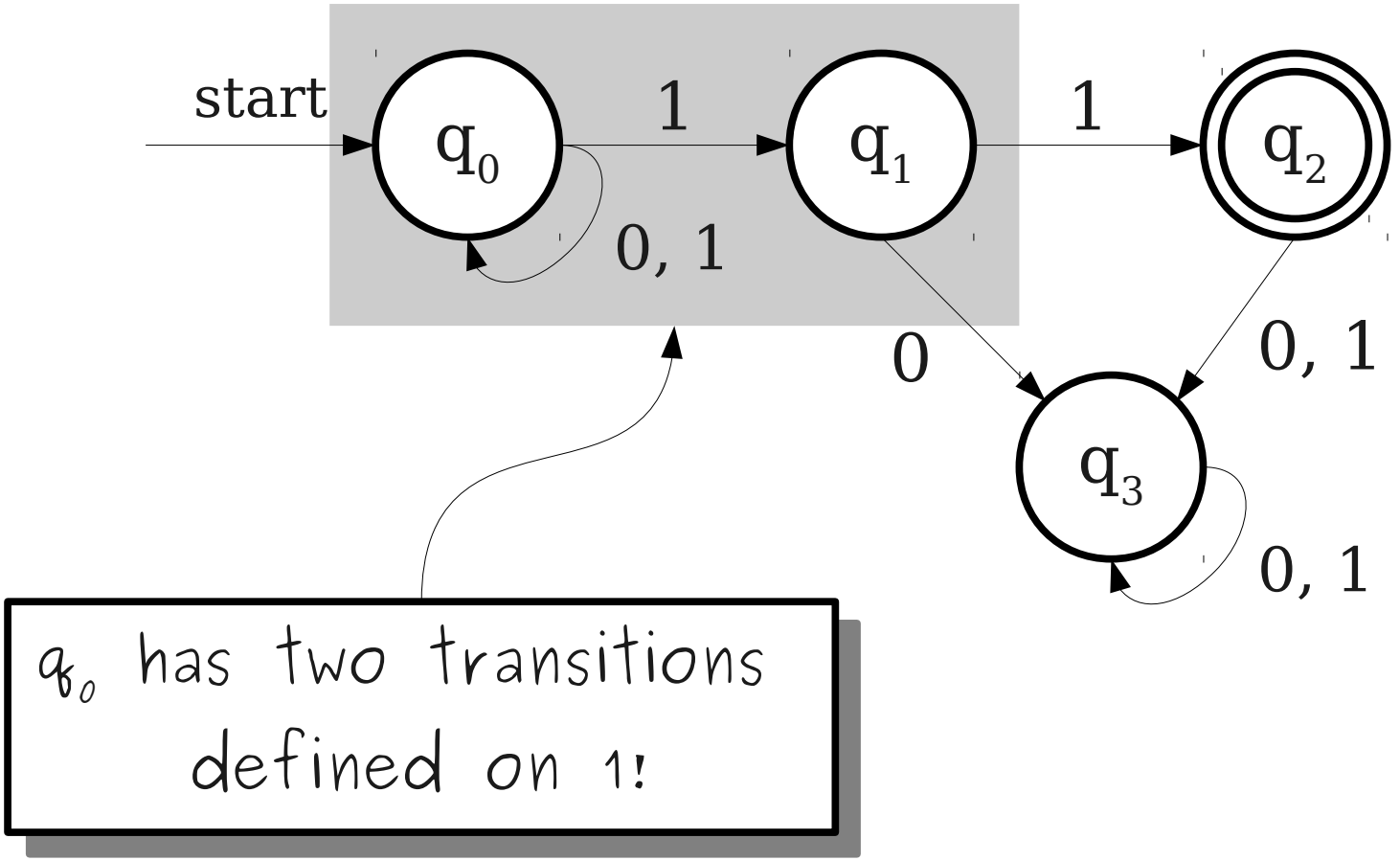
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Conceptually similar to a DFA, but equipped with the vast power of **nondeterminism**.

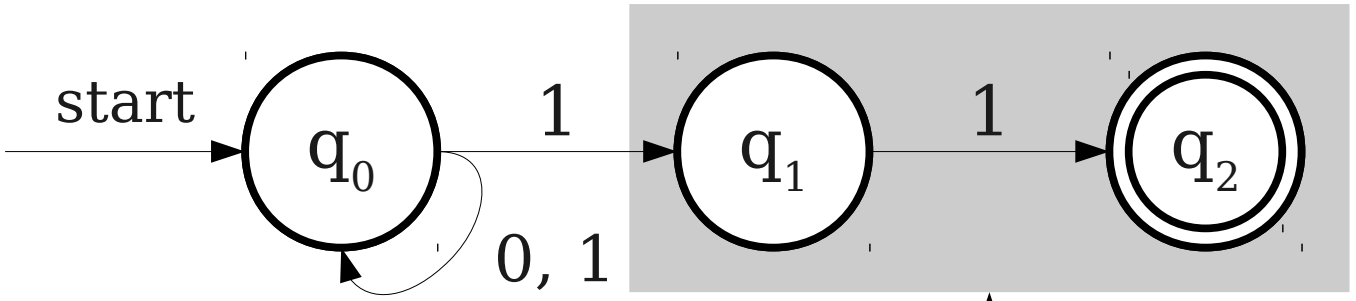
(Non)determinism

- 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

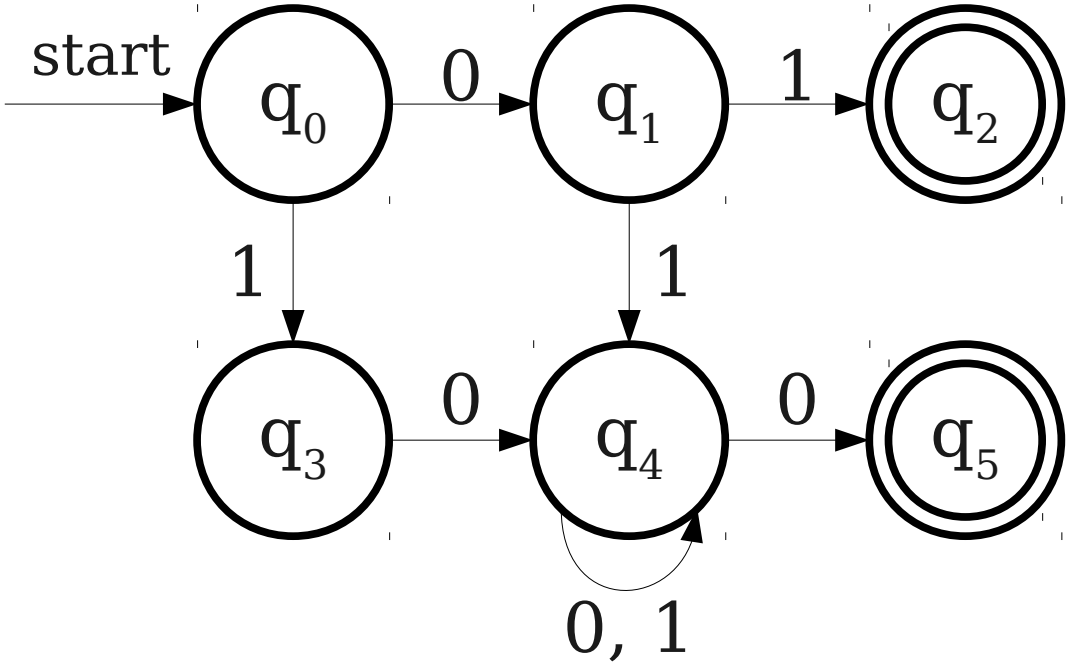


If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.

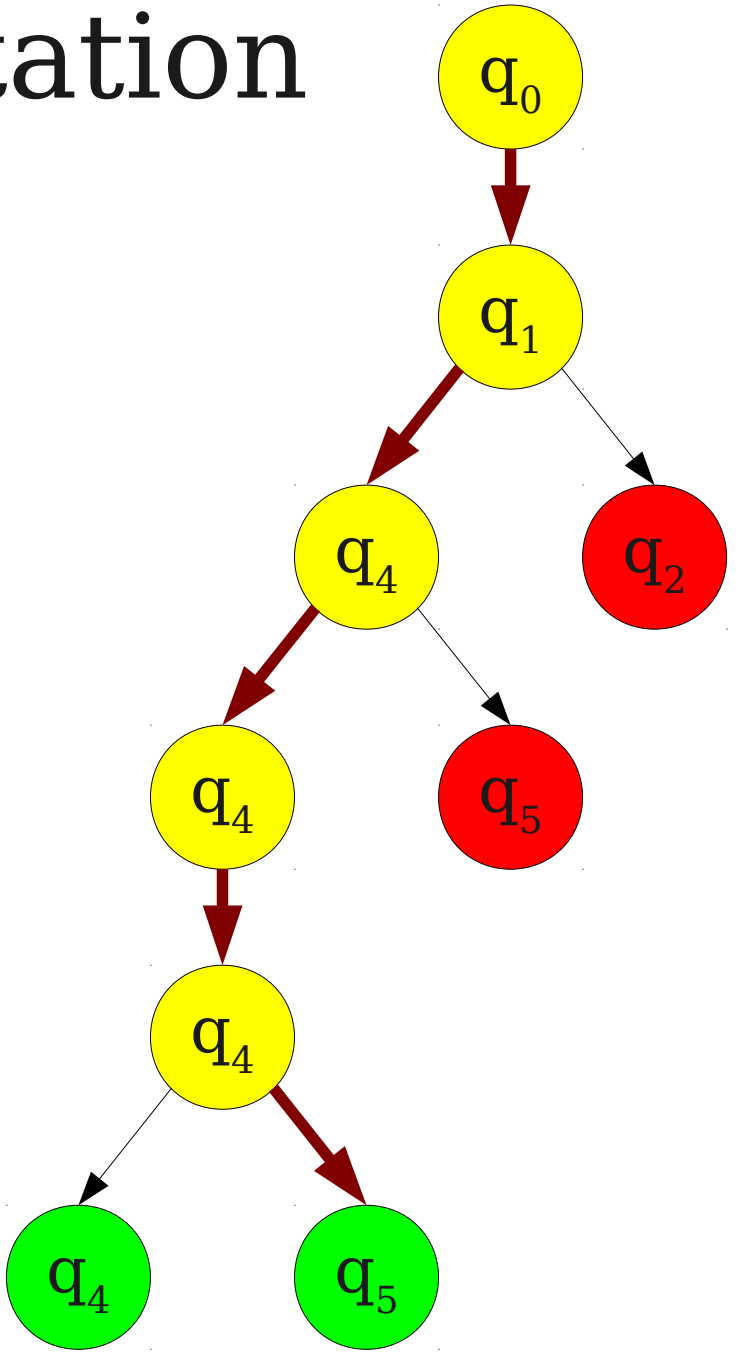
Intuiting Nondeterminism

- 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.
- Idea: Machine can always guess a path that leads to an accepting state if one exists.
- No known physical analog for this style of computation.

Massive Parallelism

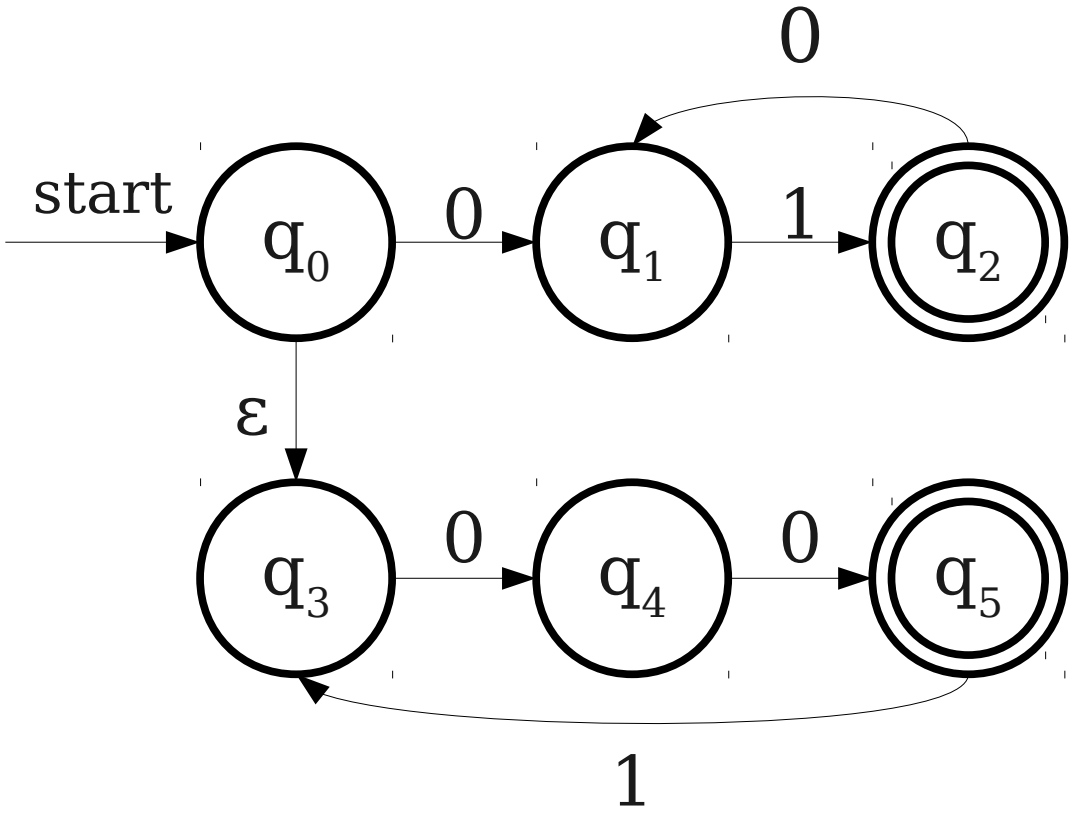
- An NFA can be thought of as a DFA that can be in many states at once.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  - No fixed limit on processors; makes multicore machines look downright wimpy!

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.

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.

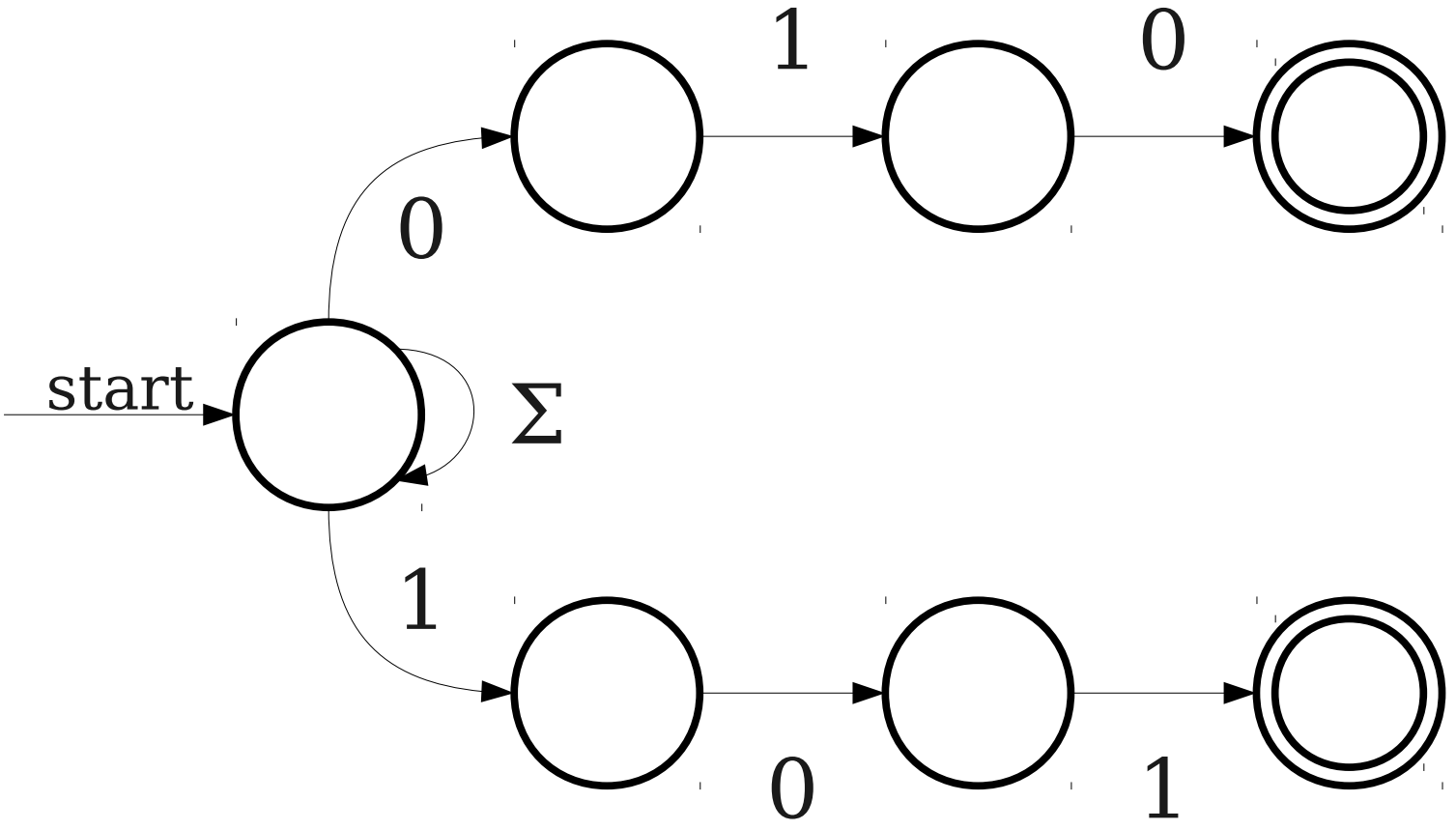


Designing NFAs

- When designing NFAs, *embrace the nondeterminism!*
- Good model: **Guess-and-check**:
  - Have the machine *nondeterministically guess* what the right choice is.
  - Have the machine *deterministically check* that the choice was correct.

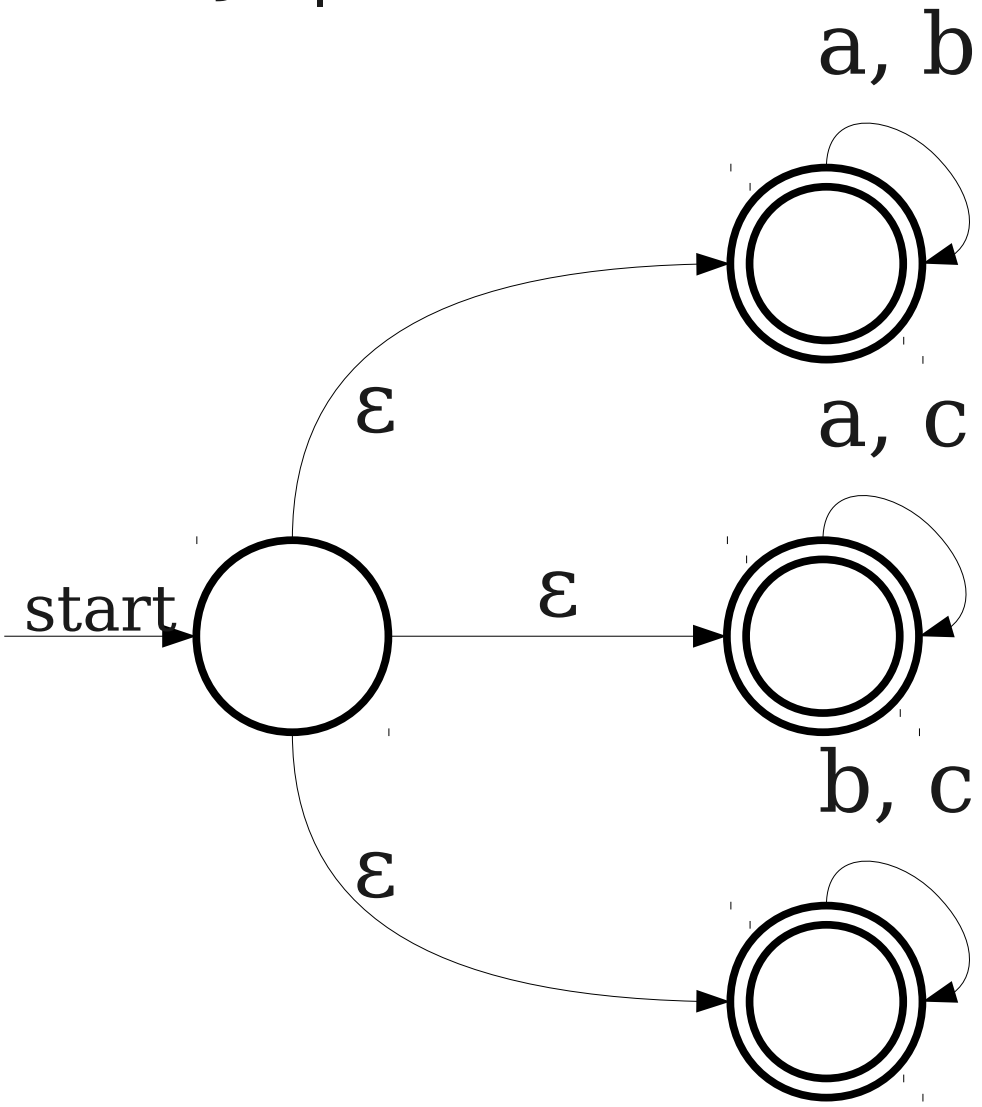
Guess-and-Check

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



Guess-and-Check

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



Next Time

- **Equivalence of NFAs and DFAs**
  - How are DFAs powerful enough to match NFAs?
- **Additional Closure Properties**
  - Closure under union, intersection, concatenation, and Kleene star!
- **Regular Expressions I**
  - A different formalism for regular languages.