Finite Automata

Part One

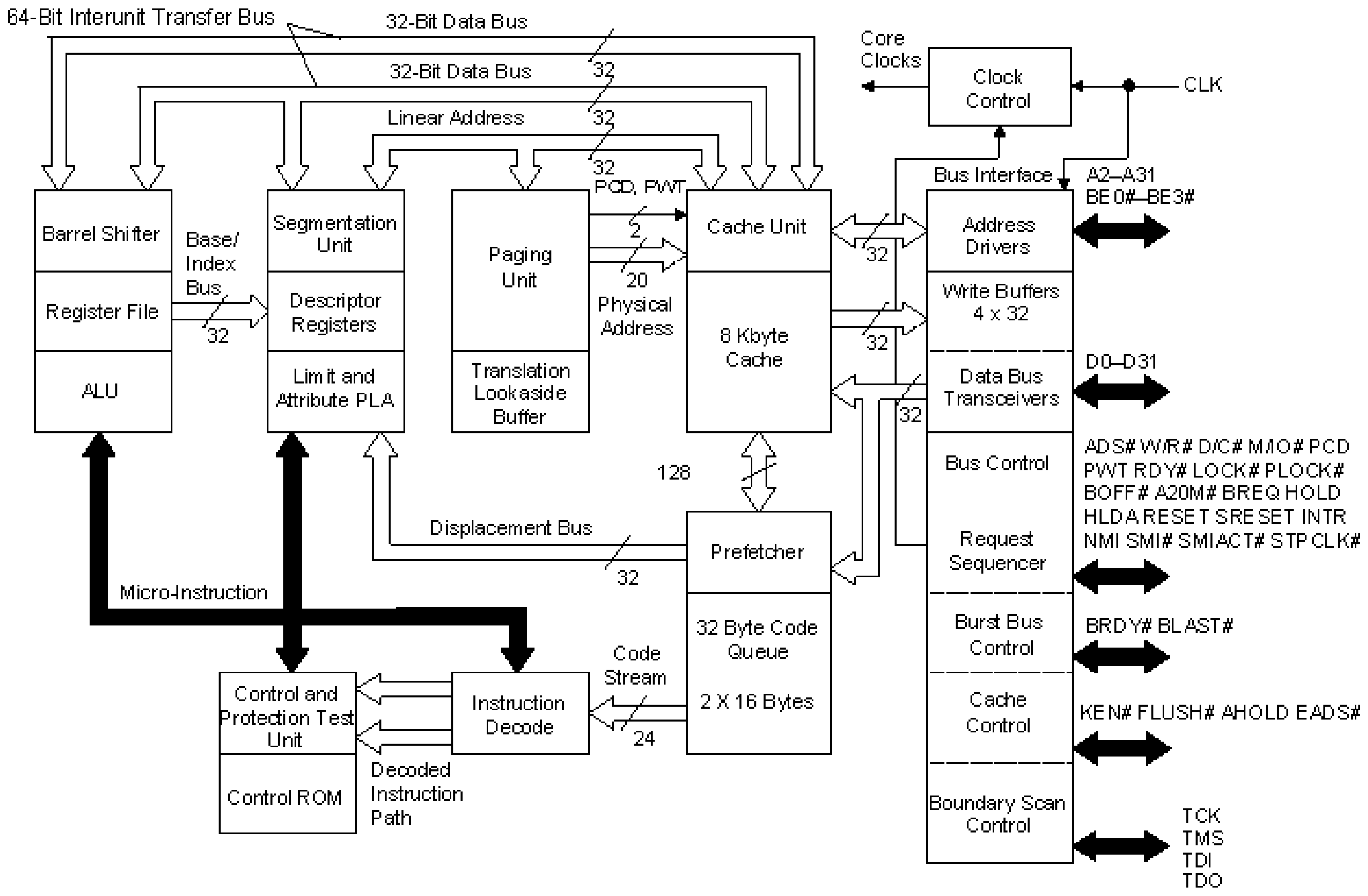
Computability Theory

What problems can we solve with a computer?

What problems can we solve with a computer?

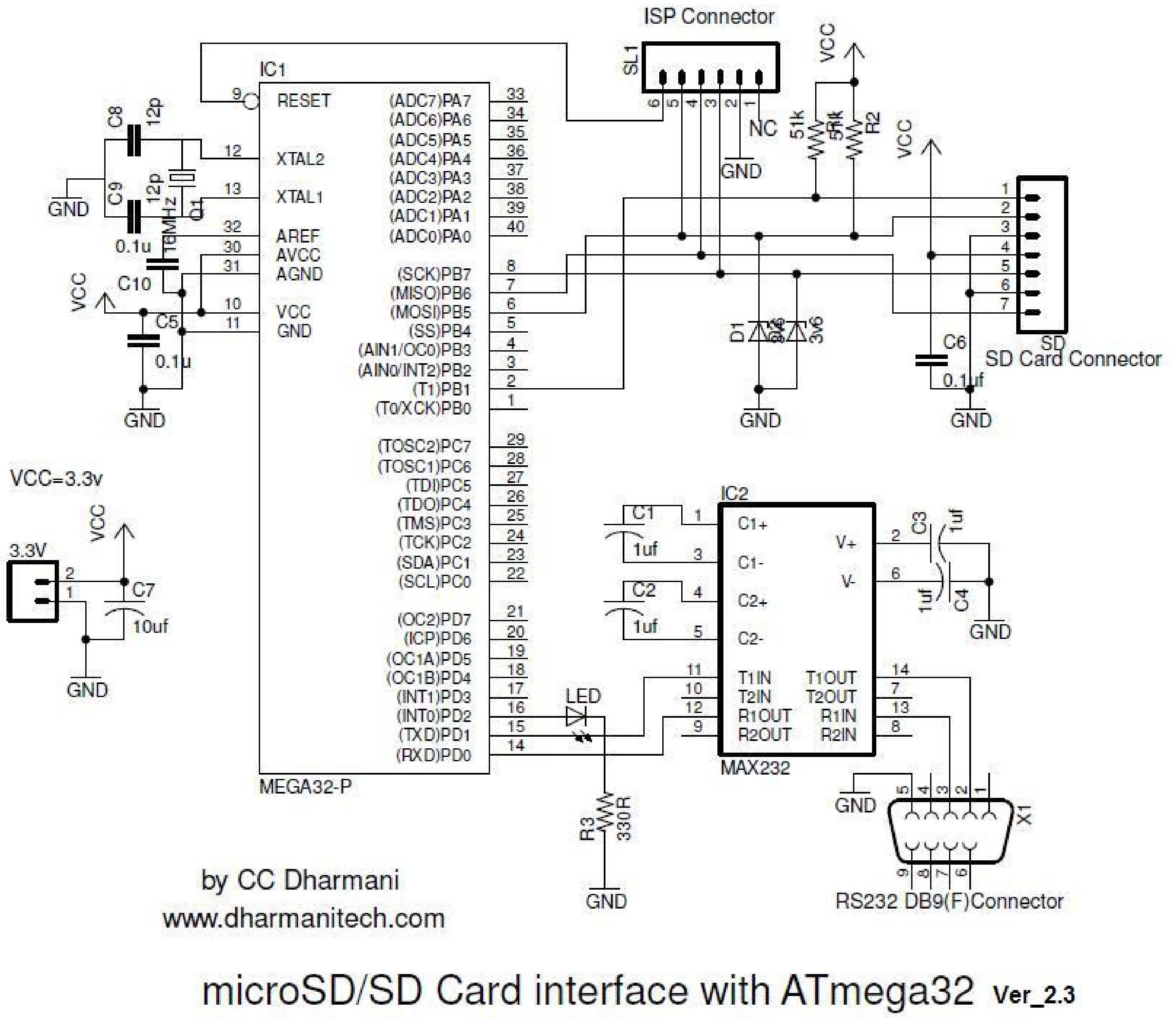
What kind of computer?

Computers are Messy



http://www.intel.com/design/intarch/prodbref/272713.htm

Computers are Messy



http://www.dharmanitech.com/

Computers are Messy

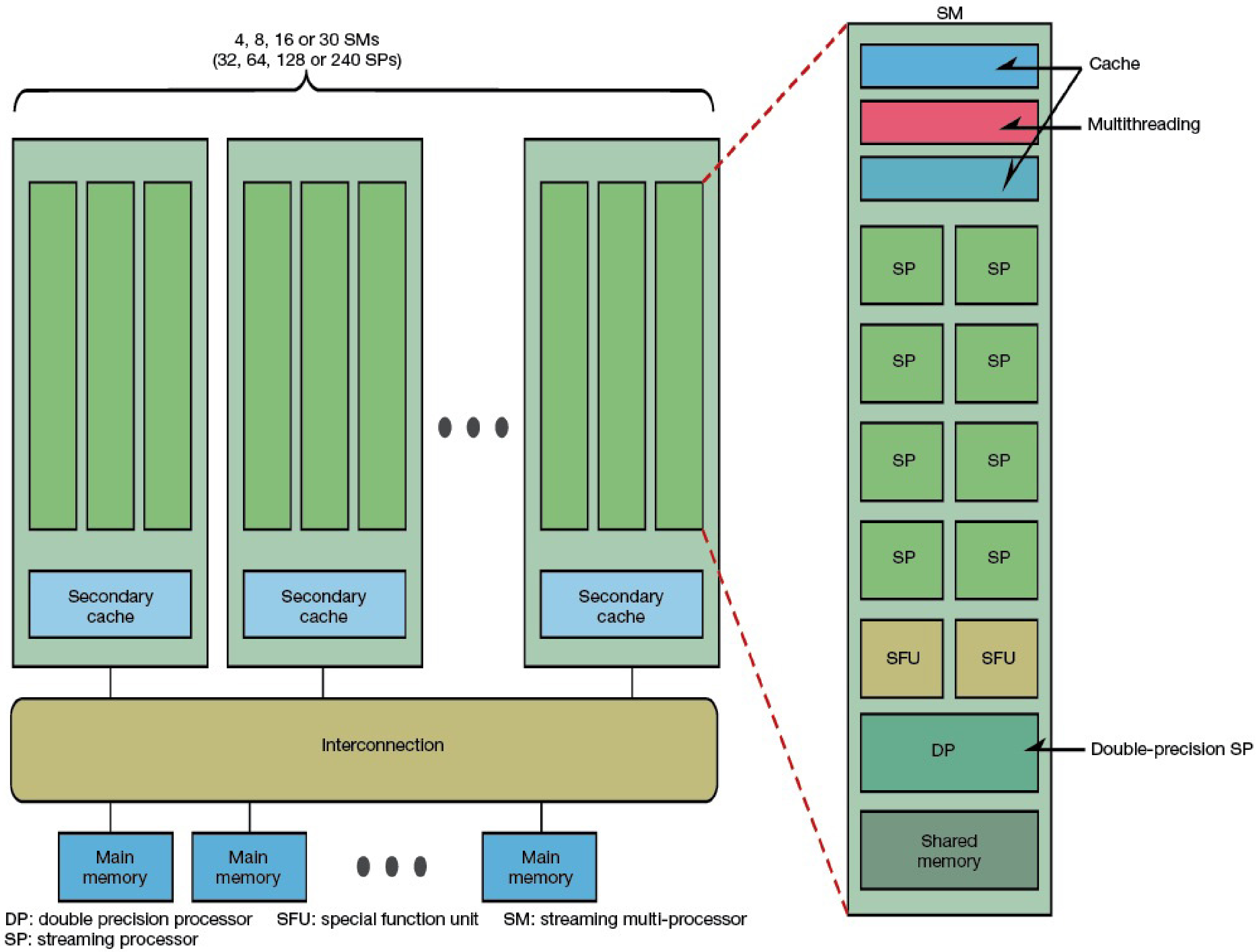
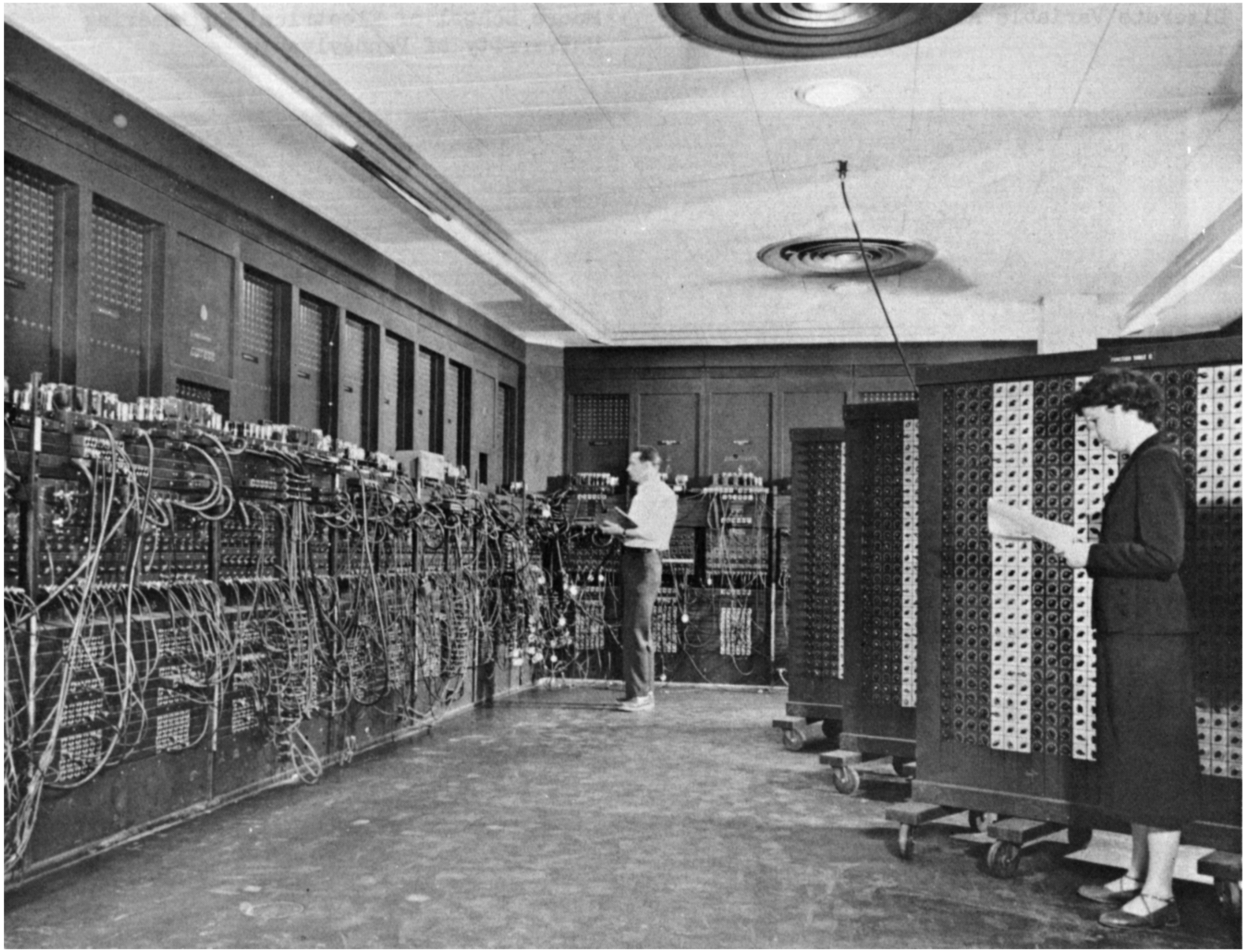


Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM) added.

http://techon.nikkeibp.co.jp/article/HONSHI/20090119/164259/

Computers are Messy

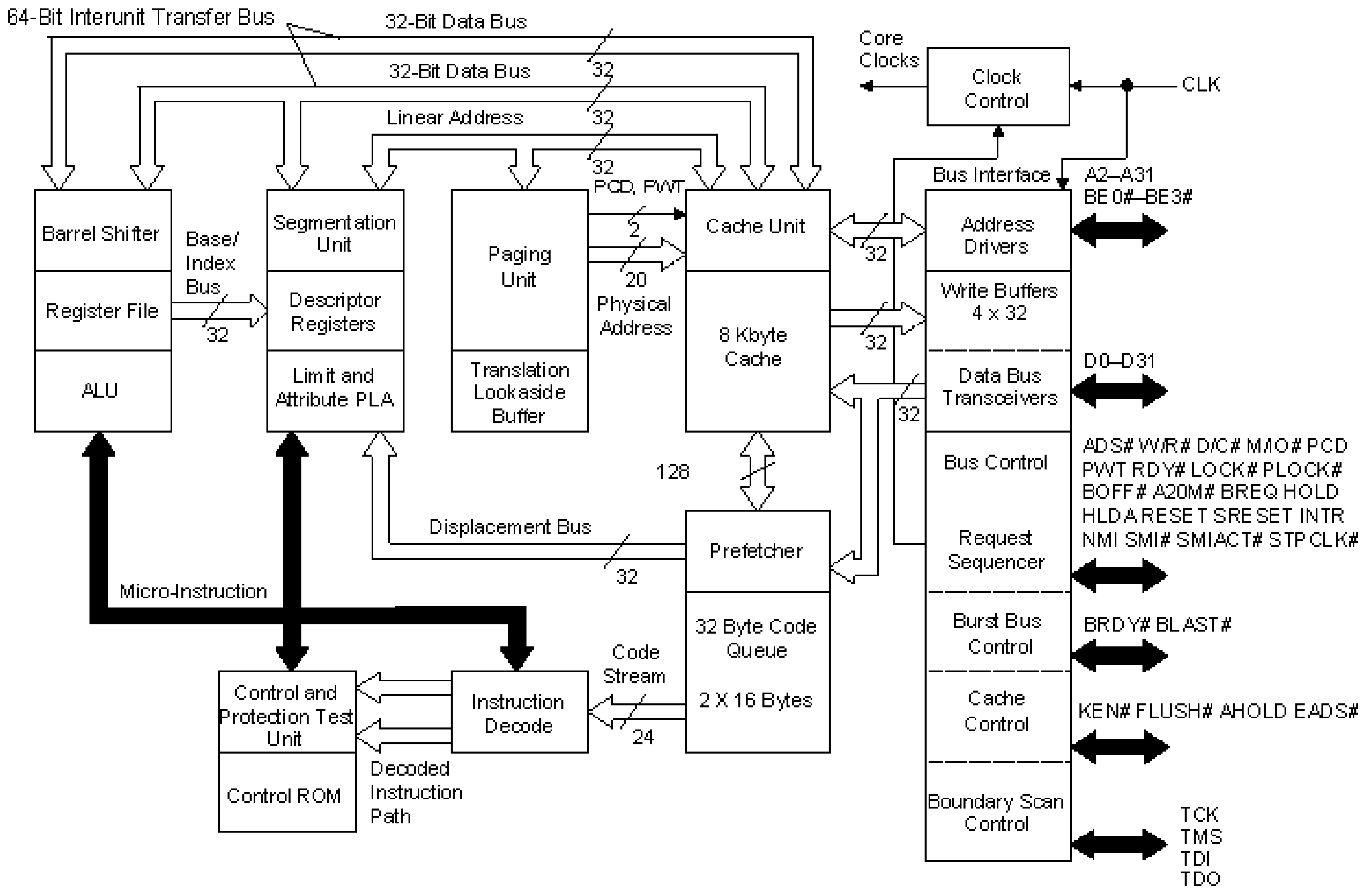


http://en.wikipedia.org/wiki/File:Eniac.jpg

We need a simpler way of discussing computing machines.

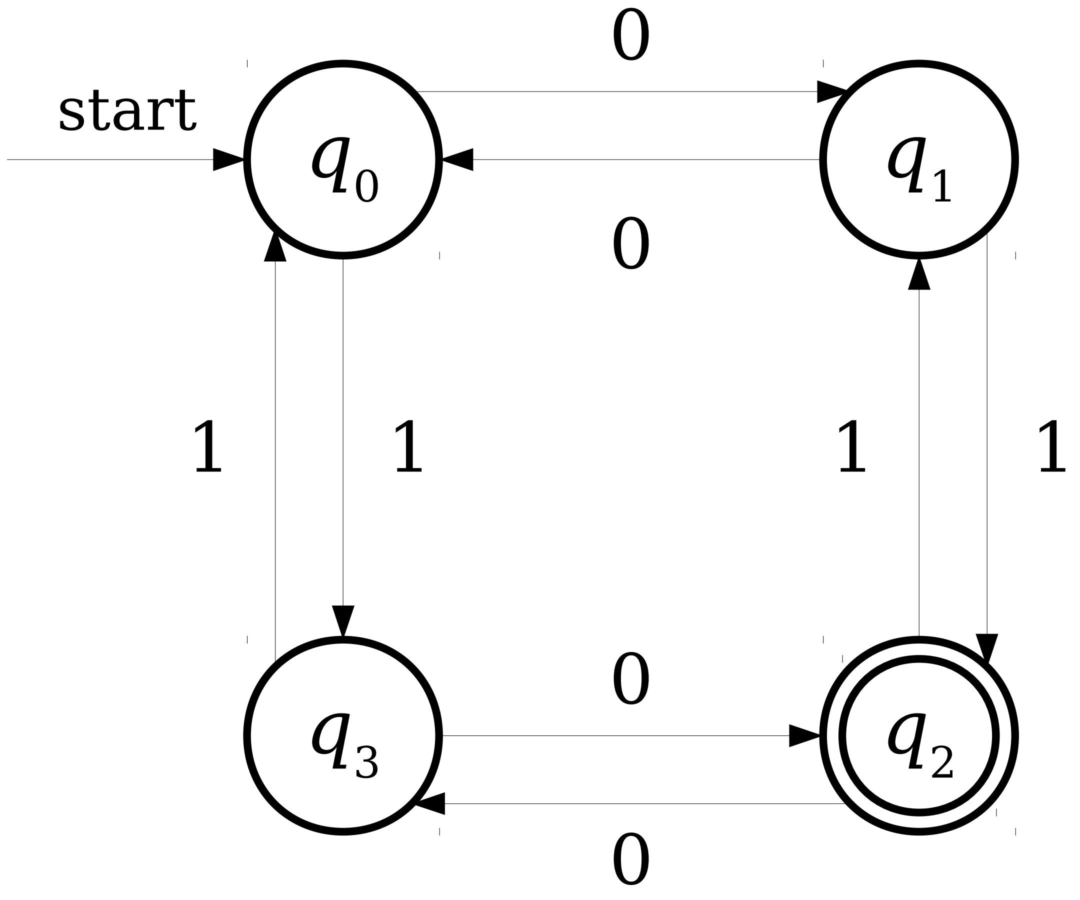
An ***automaton*** (plural: ***automata***) is a mathematical model of a computing device.

Computers are Messy

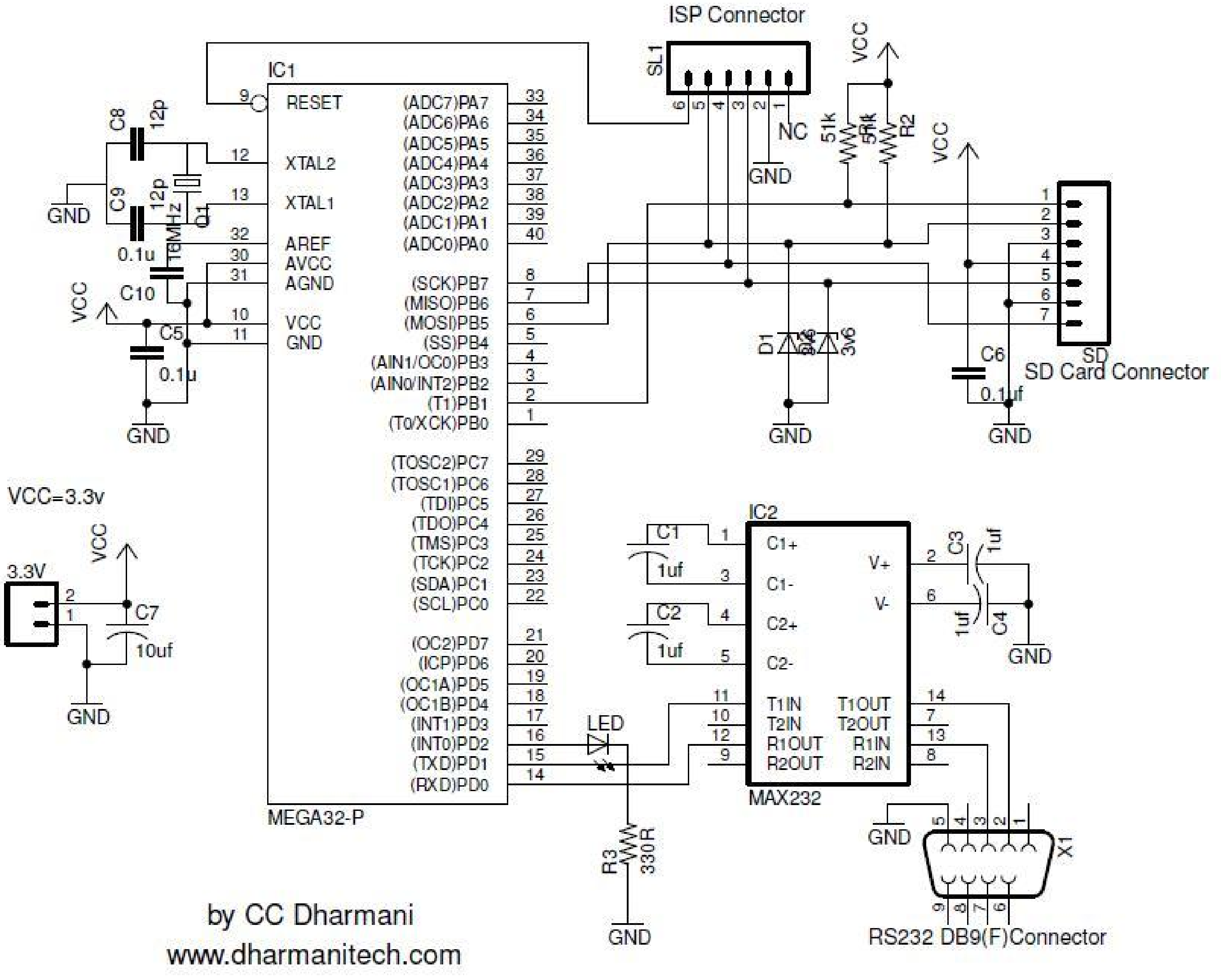


http://www.intel.com/design/intarch/prodbref/272713.htm

Automata are Clean



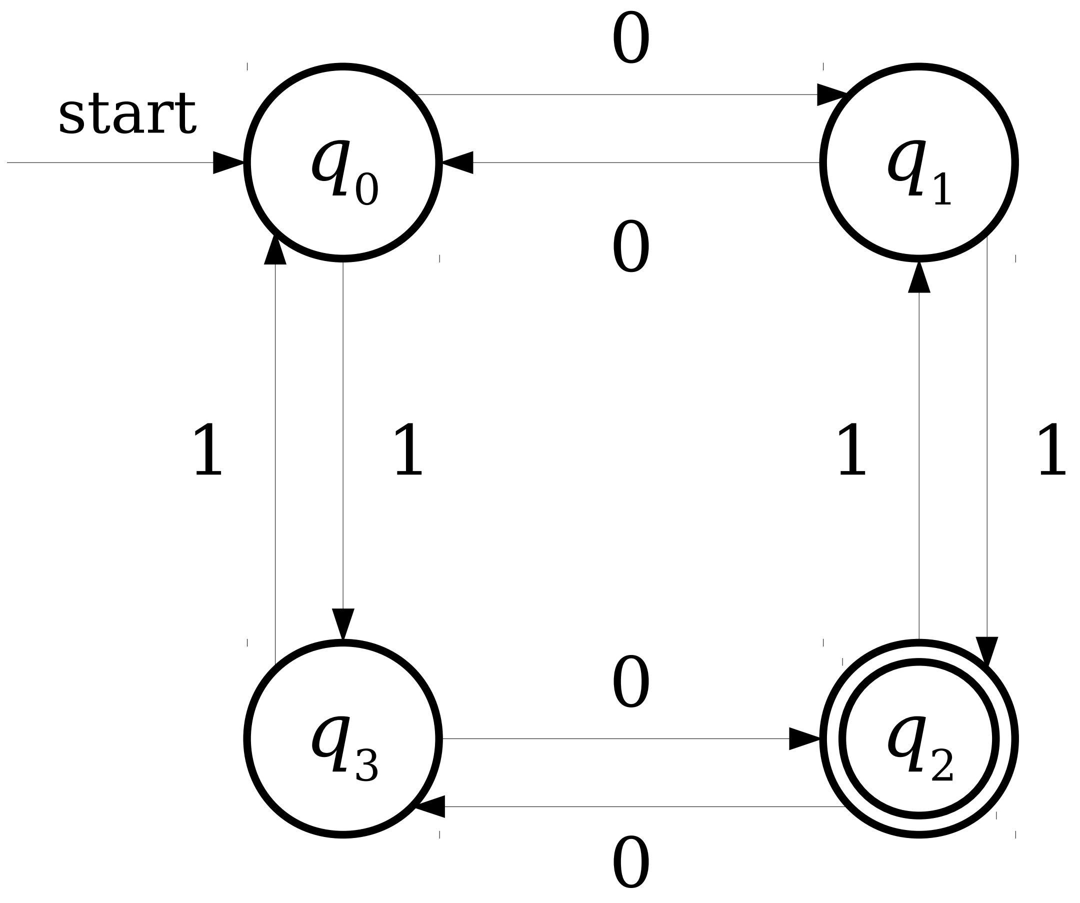
Computers are Messy



microSD/SD Card interface with ATmega32 Ver_2.3

http://www.dharmanitech.com/

Automata are Clean



Computers are Messy

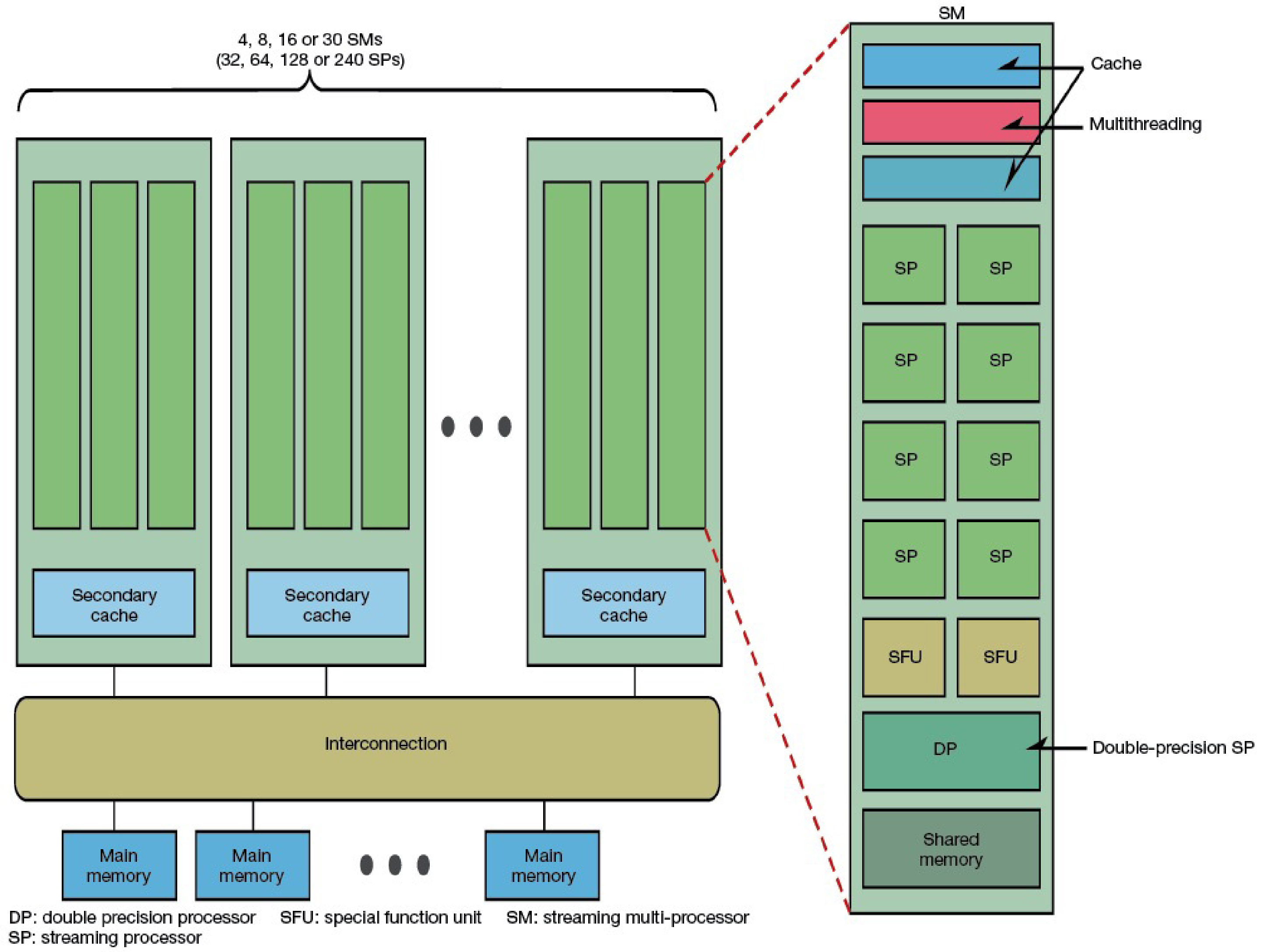
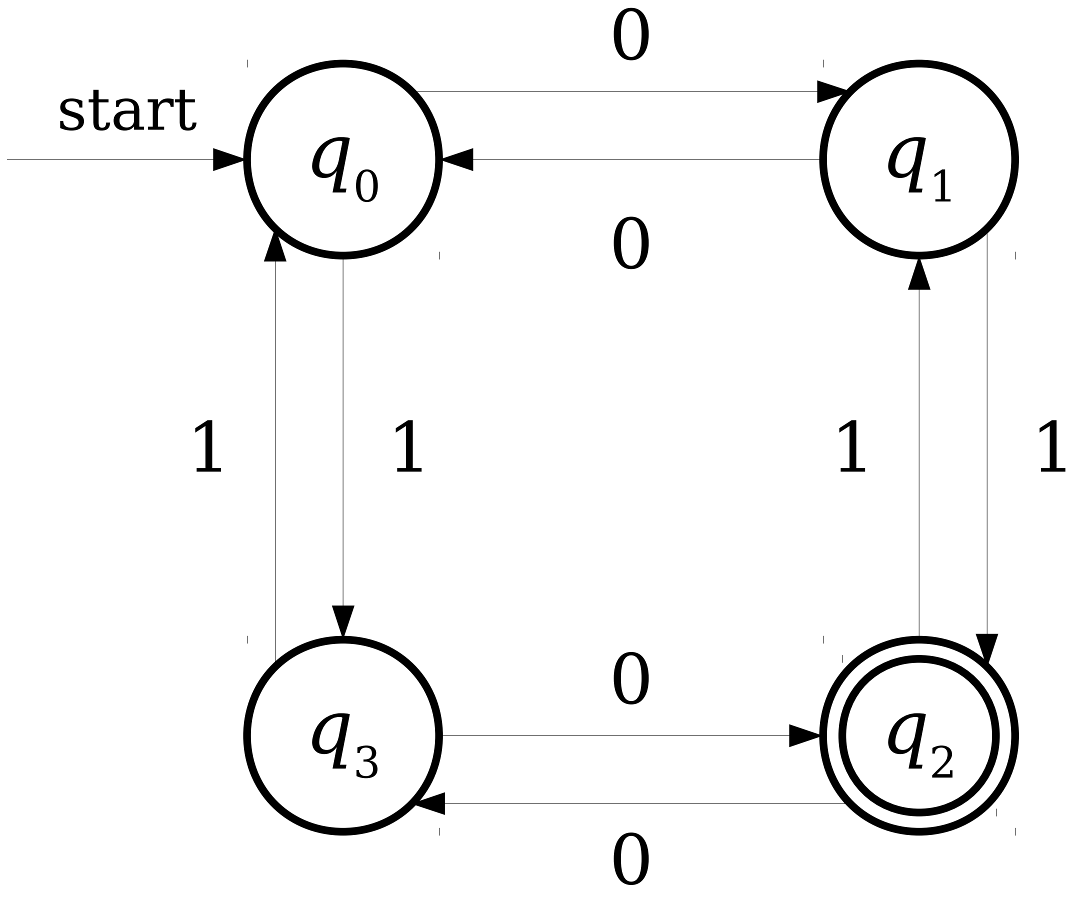


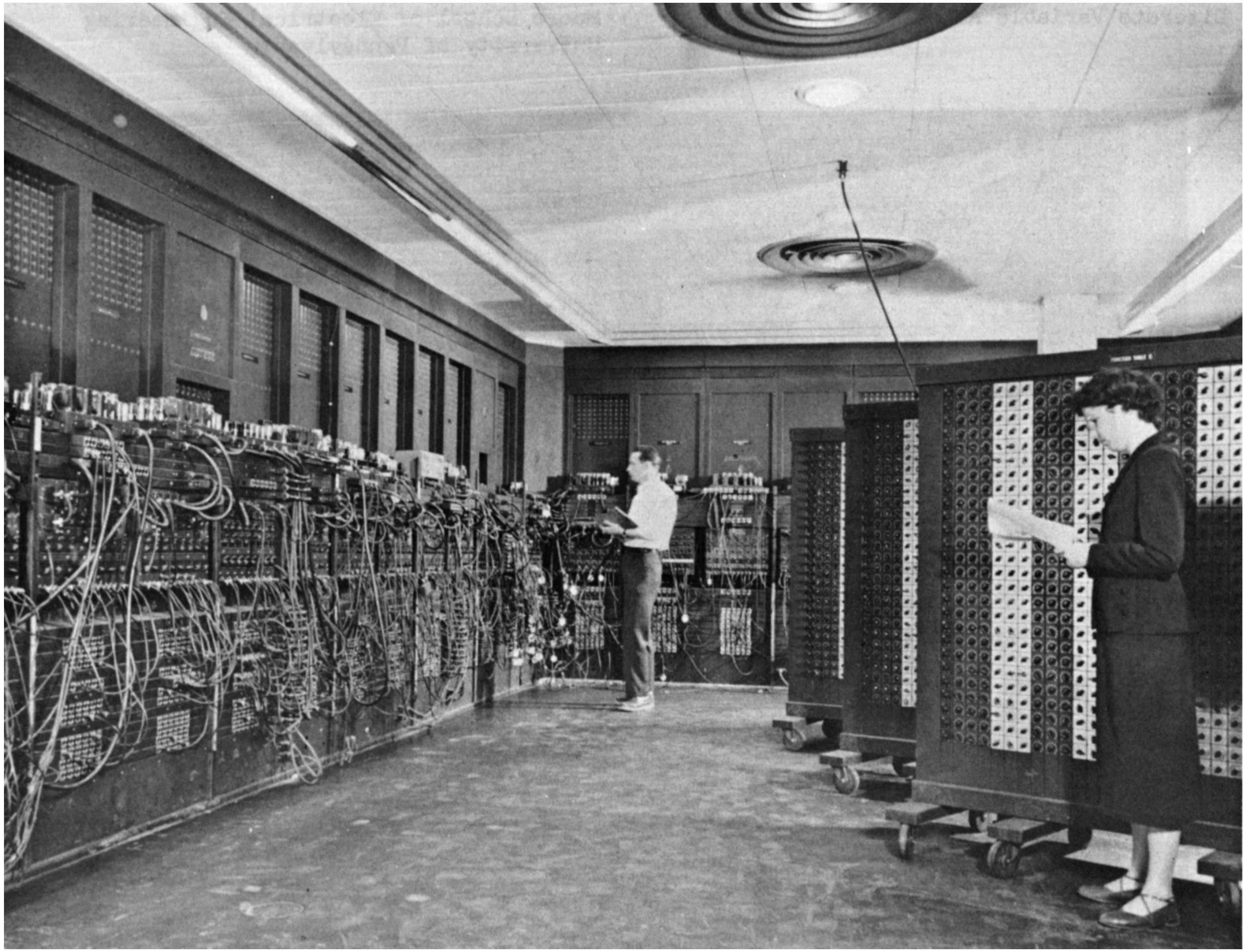
Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM) added.

http://techon.nikkeibp.co.jp/article/HONSHI/20090119/164259/

Automata are Clean

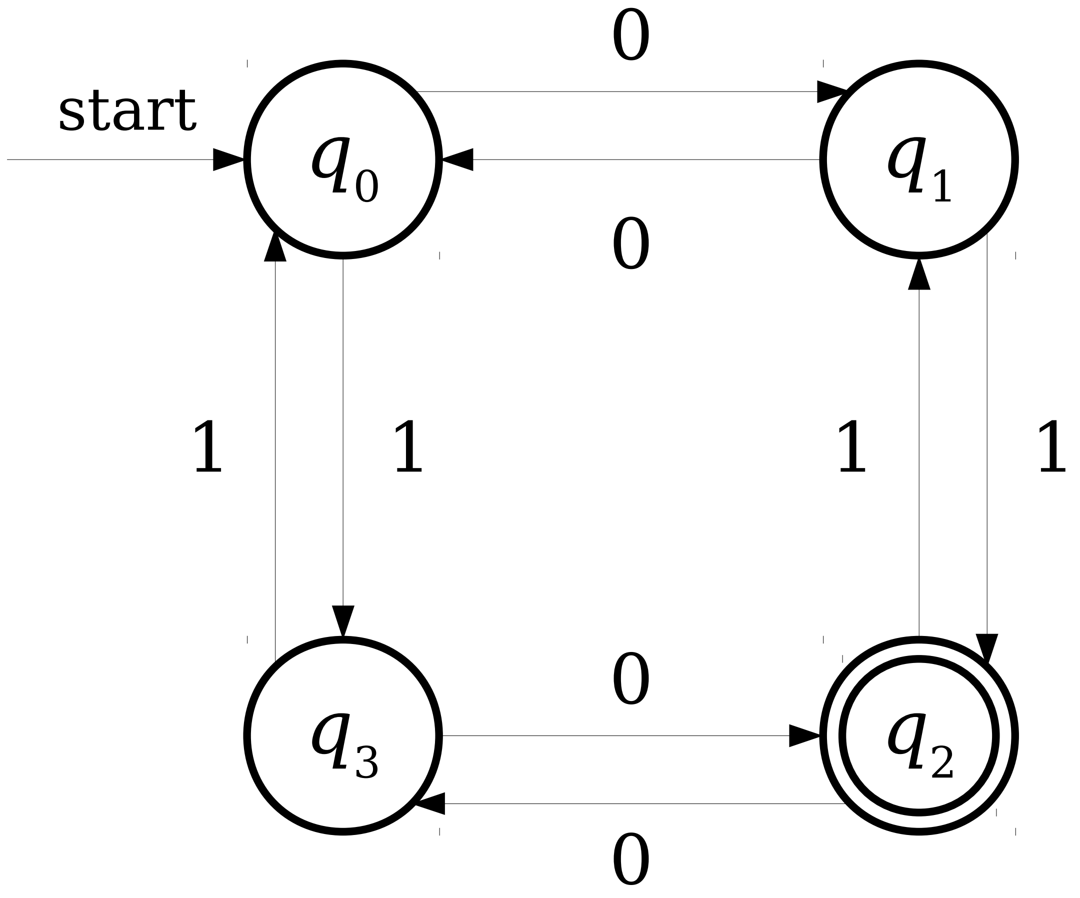


Computers are Messy



http://en.wikipedia.org/wiki/File:Eniac.jpg

Automata are Clean



Why Build Models?

- **Mathematical simplicity.**
  - It is significantly easier to manipulate our abstract models of computers than it is to manipulate actual computers.
- **Intellectual robustness.**
  - If we pick our models correctly, we can make broad, sweeping claims about huge classes of real computers by arguing that they're just special cases of our more general models.

Why Build Models?

- The models of computation we will explore in this class correspond to different conceptions of what a computer could do.
- ***Finite automata*** (next two weeks) are an abstraction of computers with finite resource constraints.
  - Provide upper bounds for the computing machines that we can actually build.
- ***Turing machines*** (later) are an abstraction of computers with unbounded resources.
  - Provide upper bounds for what we could ever hope to accomplish.

What problems can we solve with a computer?

What problems can we solve with a computer?

What is a "problem?"

Problems with Problems

- Before we can talk about what problems we can solve, we need a formal definition of a "problem."
- We want a definition that
  - corresponds to the problems we want to solve,
  - captures a large class of problems, and
  - is mathematically simple to reason about.
- No one definition has all three properties.

Formal Language Theory

Strings

- An ***alphabet*** is a finite, nonempty 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**}, here are some valid strings over Σ:

  a **aabaaabbabaaabaaaabbb** **abbababba**

- The ***empty string*** has no characters and is denoted **ε**.
- Calling attention to an earlier point: since all strings are finite sequences of characters from Σ, you cannot have a string of infinite length.

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 Σ = {**a**, **b**, **c**} is the set
    - {ε, **a**, **b**, **c**, **aa**, **bb**, **cc**, **aaa**, **aba**, **aca**, **bab**, … }
- The set of all strings composed from letters in Σ is denoted **Σ***.
- Formally, we say that L is a language over Σ if $L \subseteq \Sigma^*$.

The Model

- ***Fundamental Question:*** Given an alphabet Σ and a language L over Σ, in what cases can we build an automaton that determines which strings are in L?
- The answer depends on the choice of L, the choice of automaton, and the definition of "determines."
- The entire rest of the quarter will be dedicated to answering these questions.

To Summarize

- An ***automaton*** is an idealized mathematical computing machine.
- A ***language*** is a set of strings.
- The automata we will study will accept as input a string and (attempt to) determine whether that string is contained in a particular language.

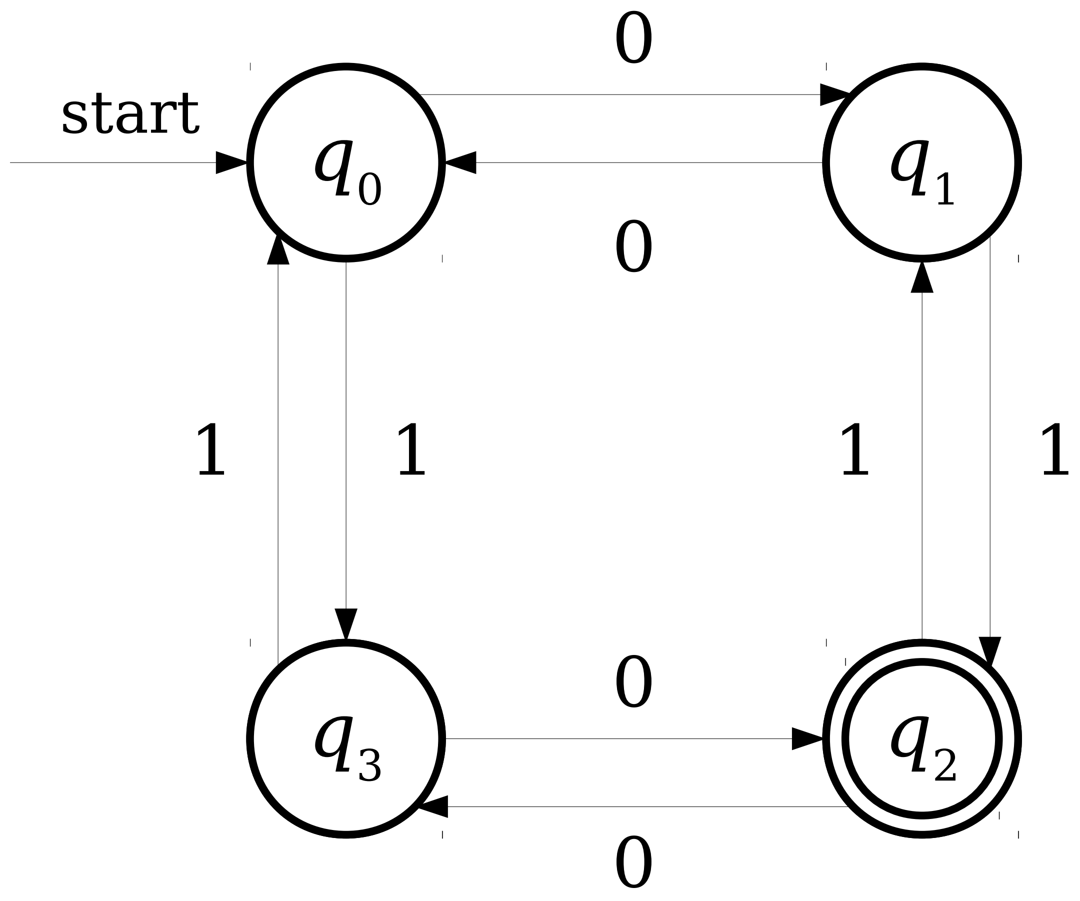
What problems can we solve with a computer?

Finite Automata

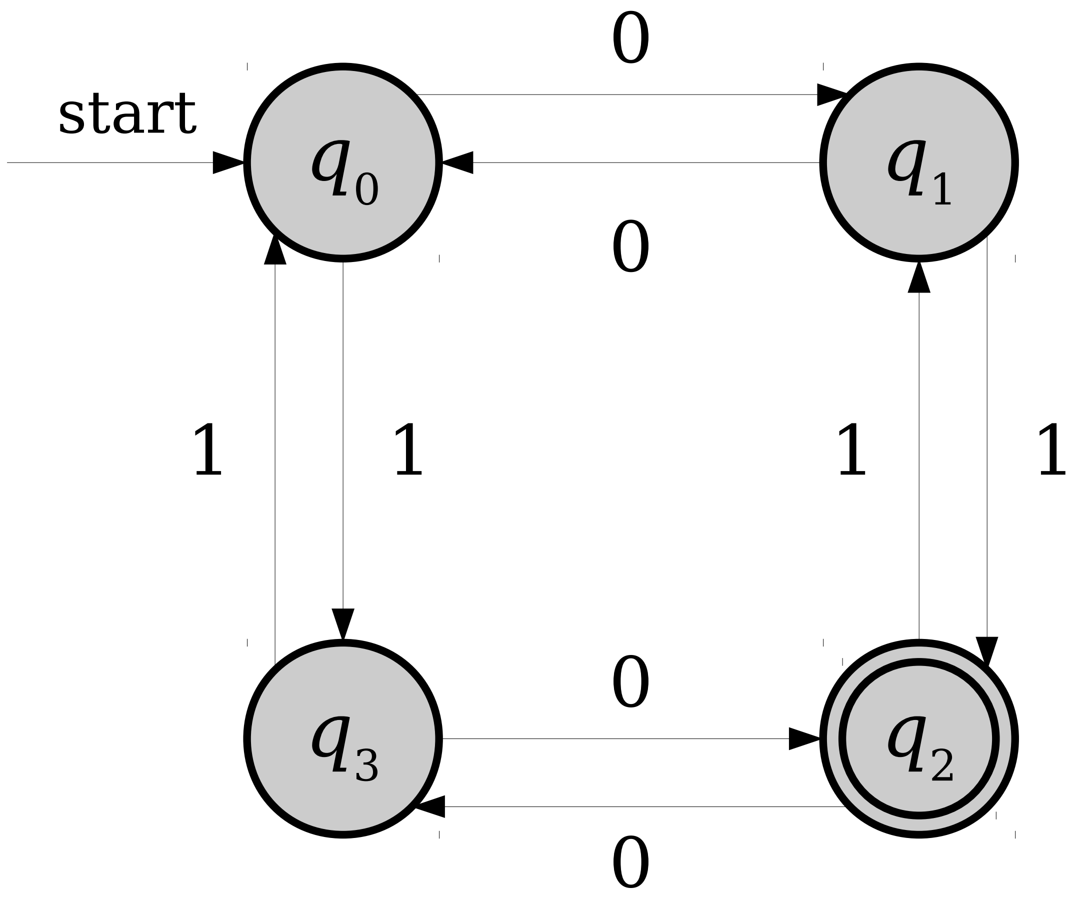
A ***finite automaton*** is a simple type of mathematical machine for determining whether a string is contained within some language.

Each finite automaton consists of a set of ***states*** connected by ***transitions***.

A Simple Finite Automaton



A Simple Finite Automaton



Each circle represents a state of the automaton.

A Simple Finite Automaton

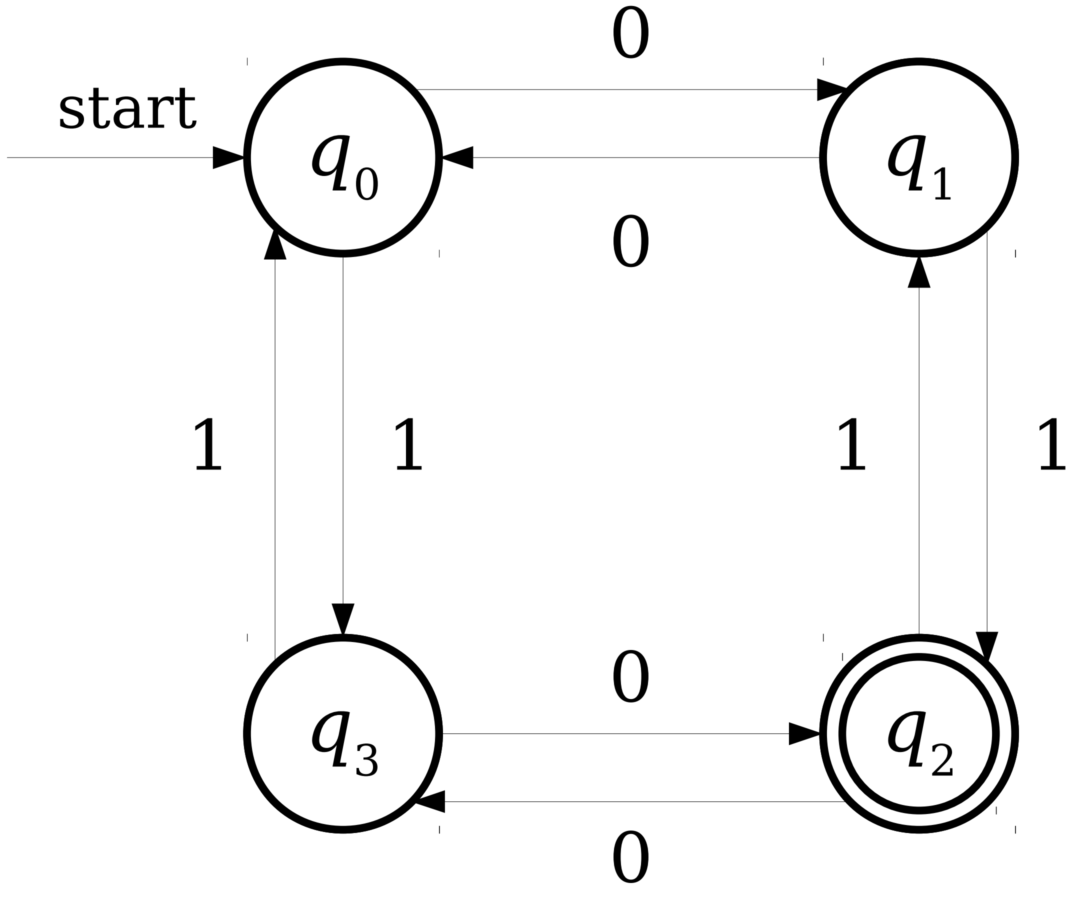
$$\begin{tikzcd}
{} \arrow[r, "\text{start}"] & q_0 \arrow[r, shift left, "0"] \arrow[d, shift left, "1"] & q_1 \arrow[l, shift left, "0"] \arrow[d, shift left, "1"] \\
& q_3 \arrow[u, shift left, "1"] \arrow[r, shift left, "0"] & q_2 \arrow[l, shift left, "0"] \arrow[u, shift left, "1"]
\end{tikzcd}$$

A Simple Finite Automaton

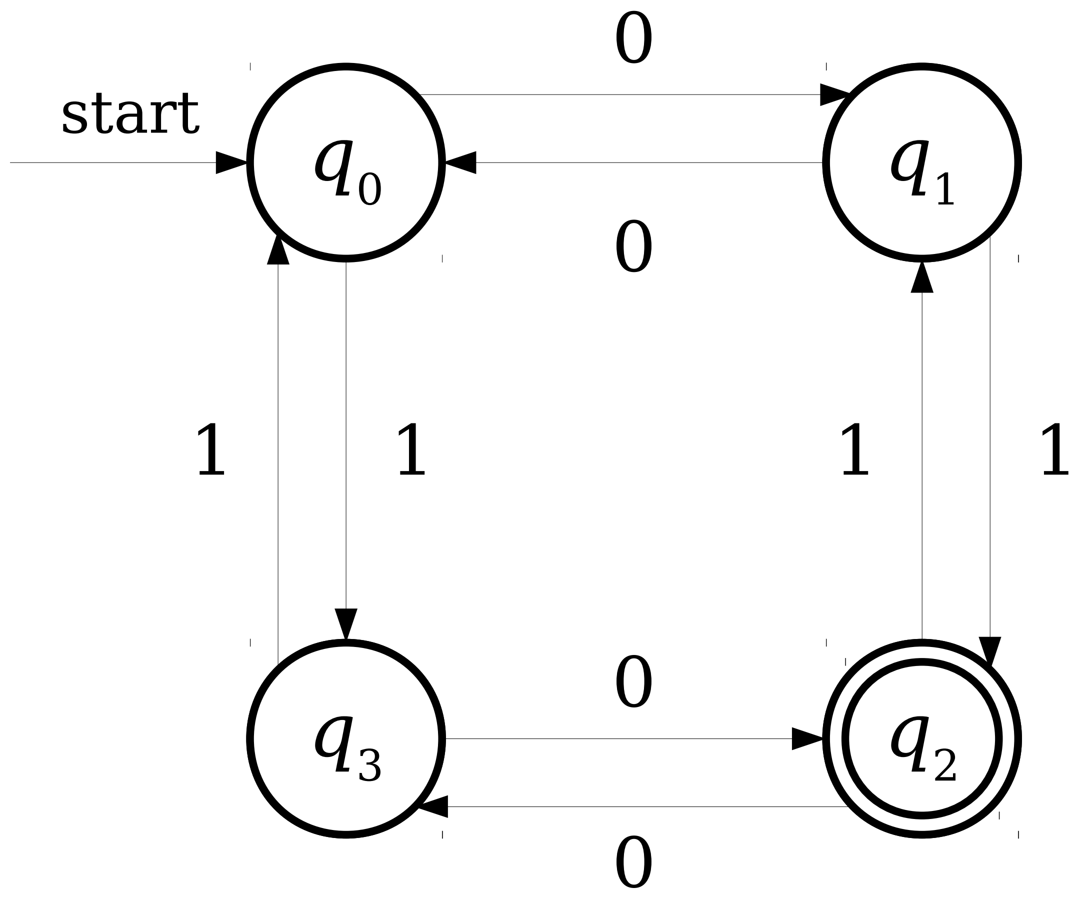
$$\begin{tikzcd}
{} \arrow[r, "\text{start}"] & q_0 \arrow[r, shift left, "0"] \arrow[d, shift left, "1"] & q_1 \arrow[l, shift left, "0"] \arrow[d, shift left, "1"] \\
 & q_3 \arrow[u, shift left, "1"] \arrow[r, shift left, "0"] & q_2 \arrow[l, shift left, "0"] \arrow[u, shift left, "1"]
\end{tikzcd}$$

One special state is designated as the **start state**.

A Simple Finite Automaton

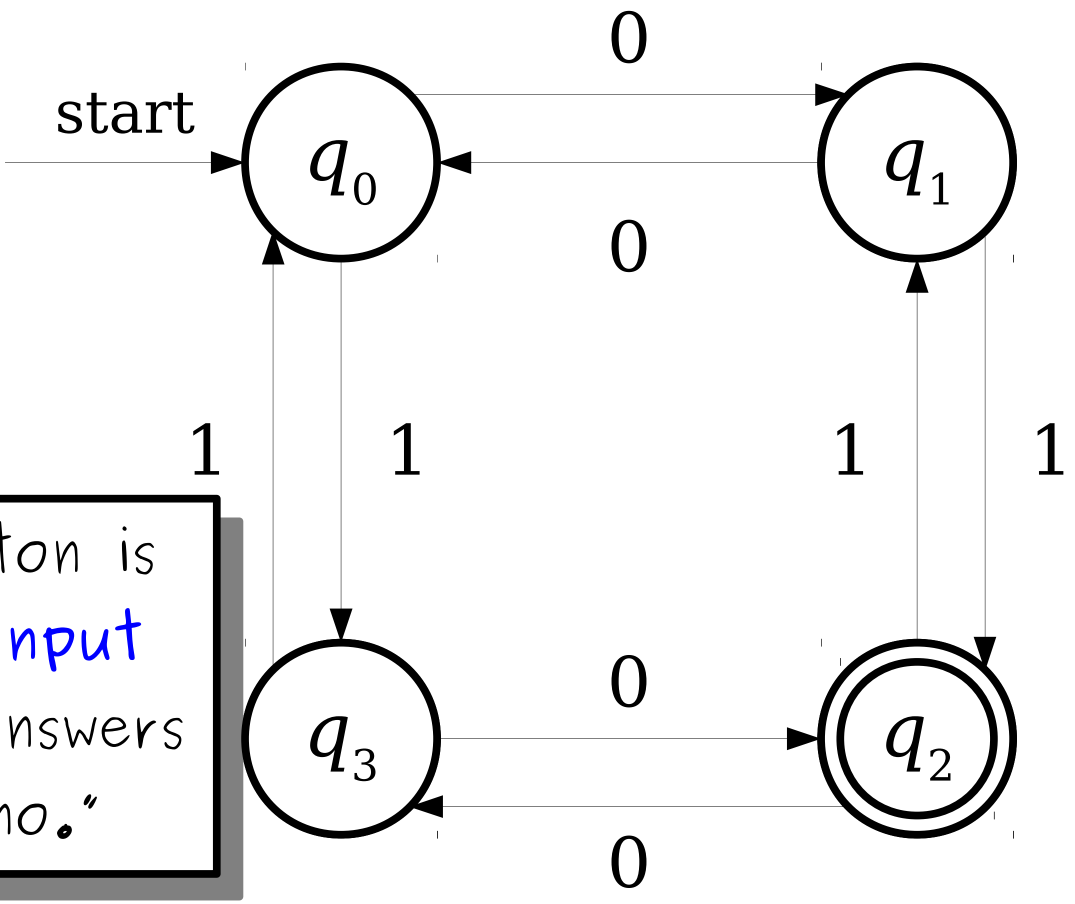


A Simple Finite Automaton



0 1 0 1 1 0

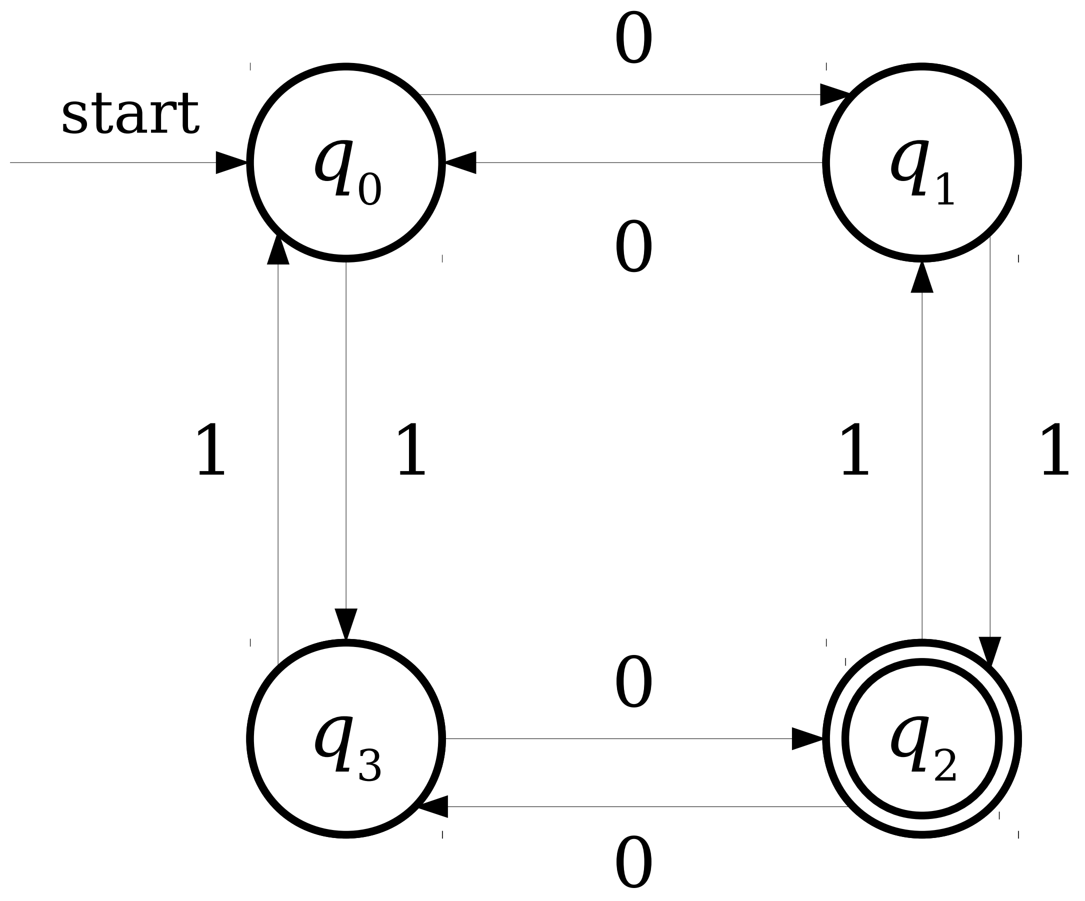
A Simple Finite Automaton



The automaton is run on an input string and answers "yes" or "no."

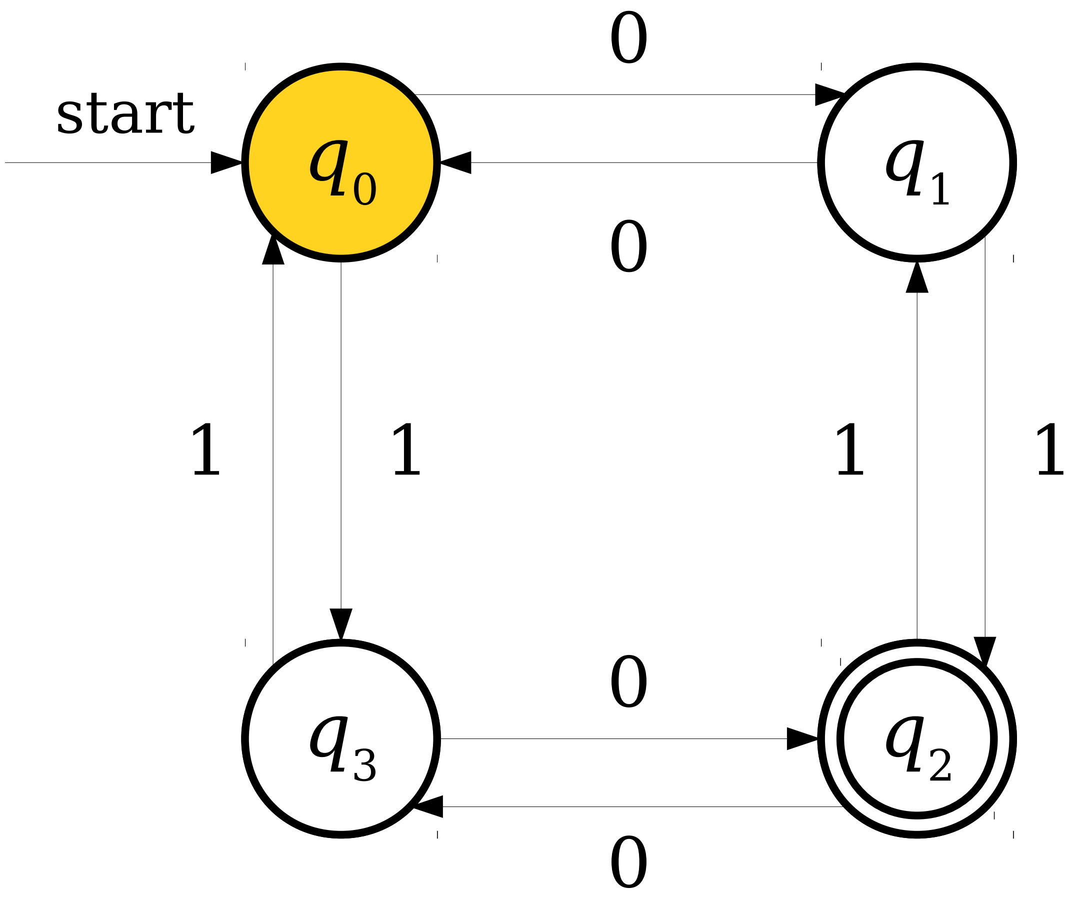
0 1 0 1 1 0

A Simple Finite Automaton



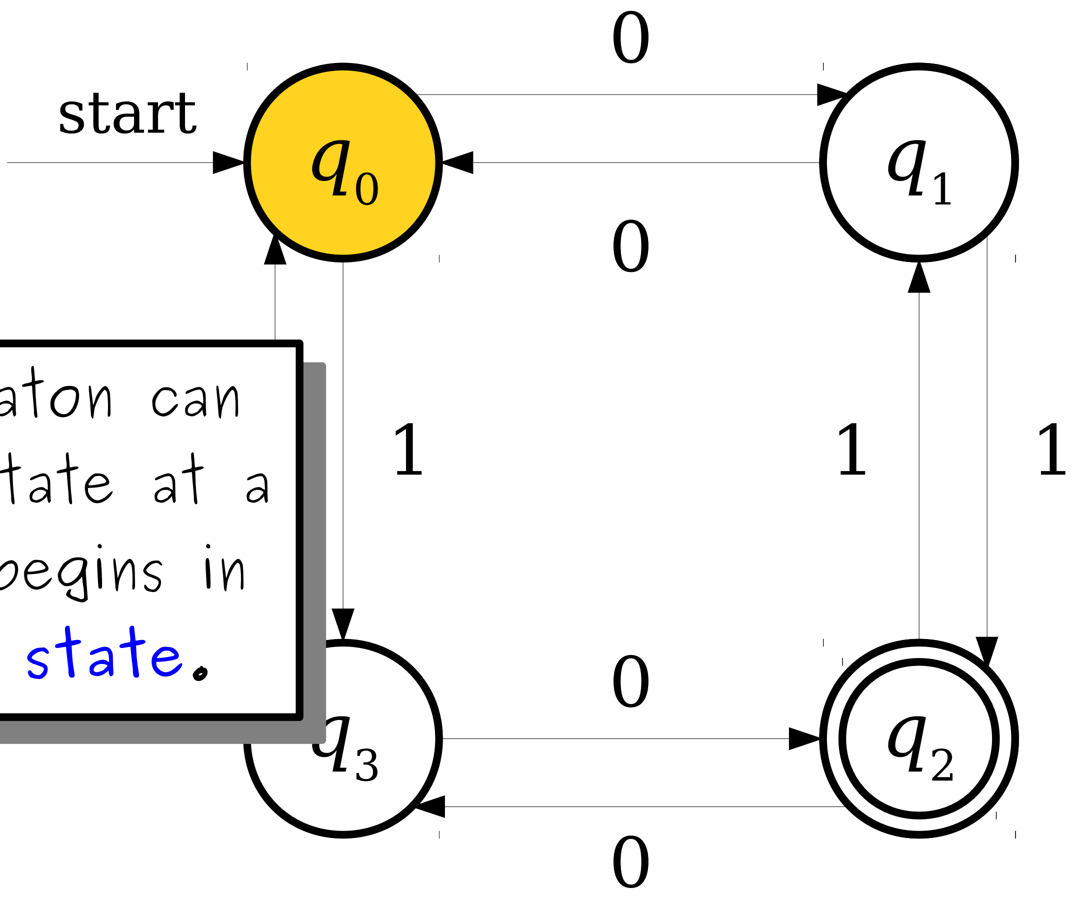
0 1 0 1 1 0

A Simple Finite Automaton



0 1 0 1 1 0

A Simple Finite Automaton



The automaton can be in one state at a time. It begins in the start state.

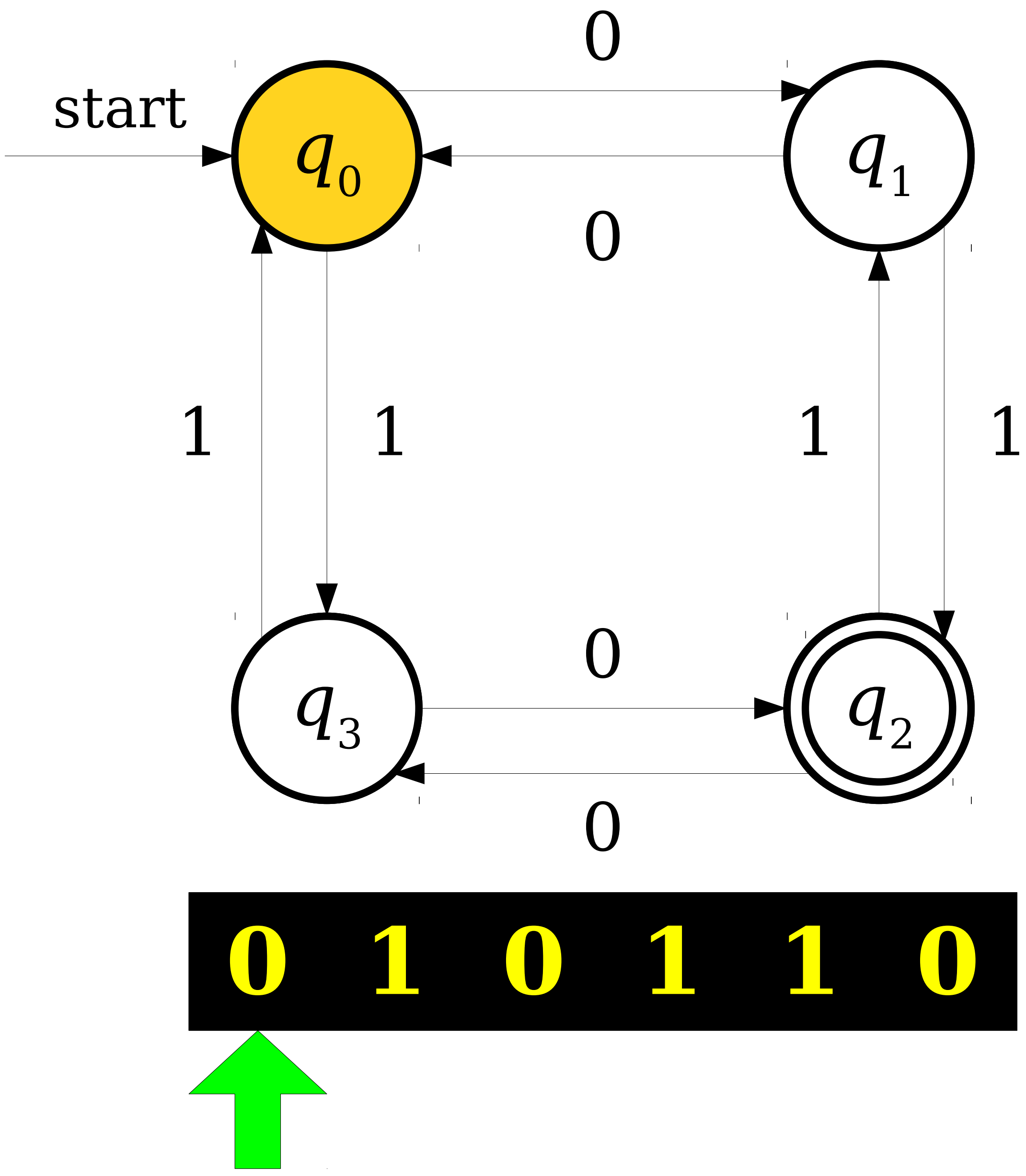
0 1 0 1 1 0

A Simple Finite Automaton

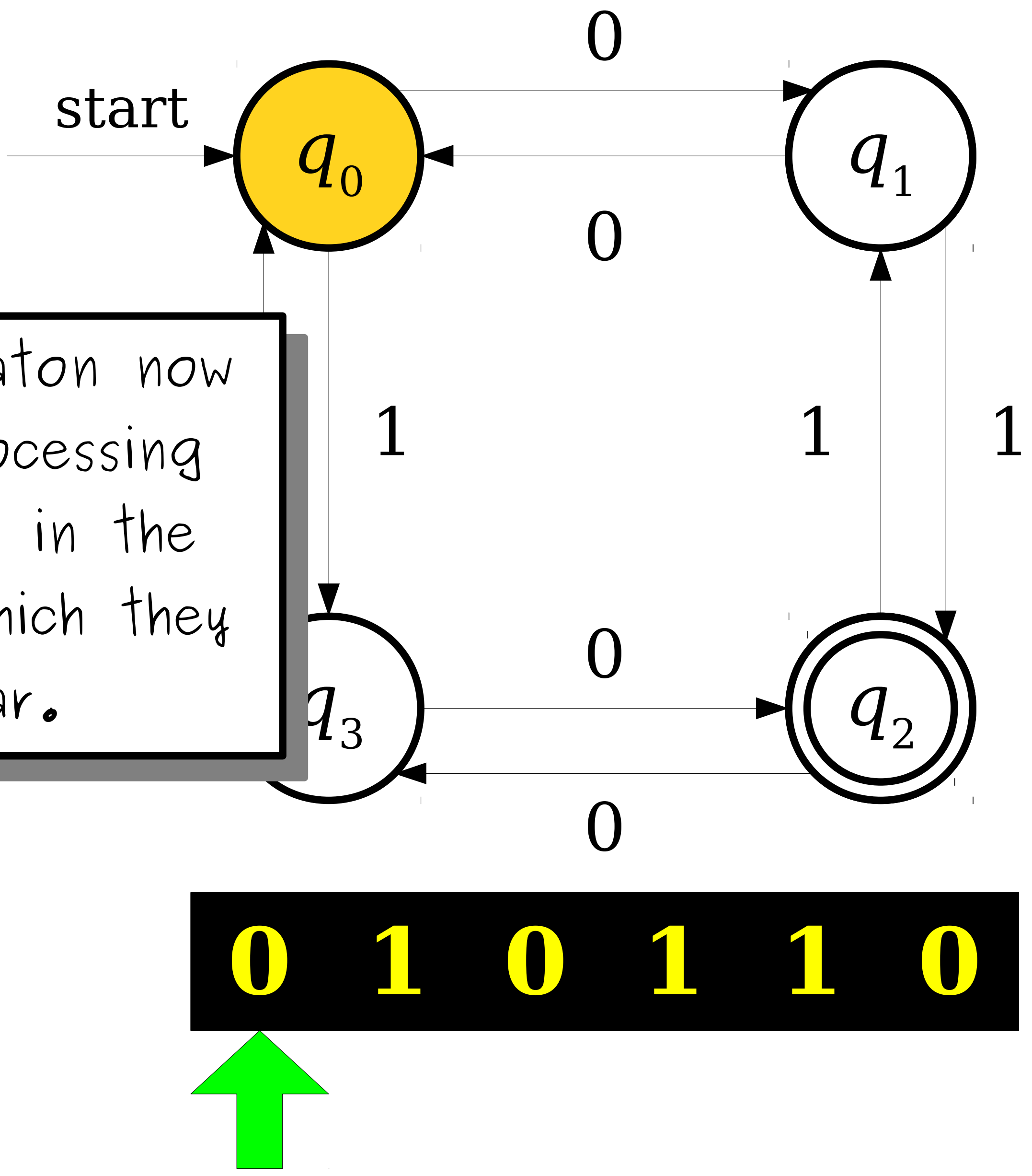
$$\begin{tikzcd}
{} \arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

0 1 0 1 1 0

A Simple Finite Automaton

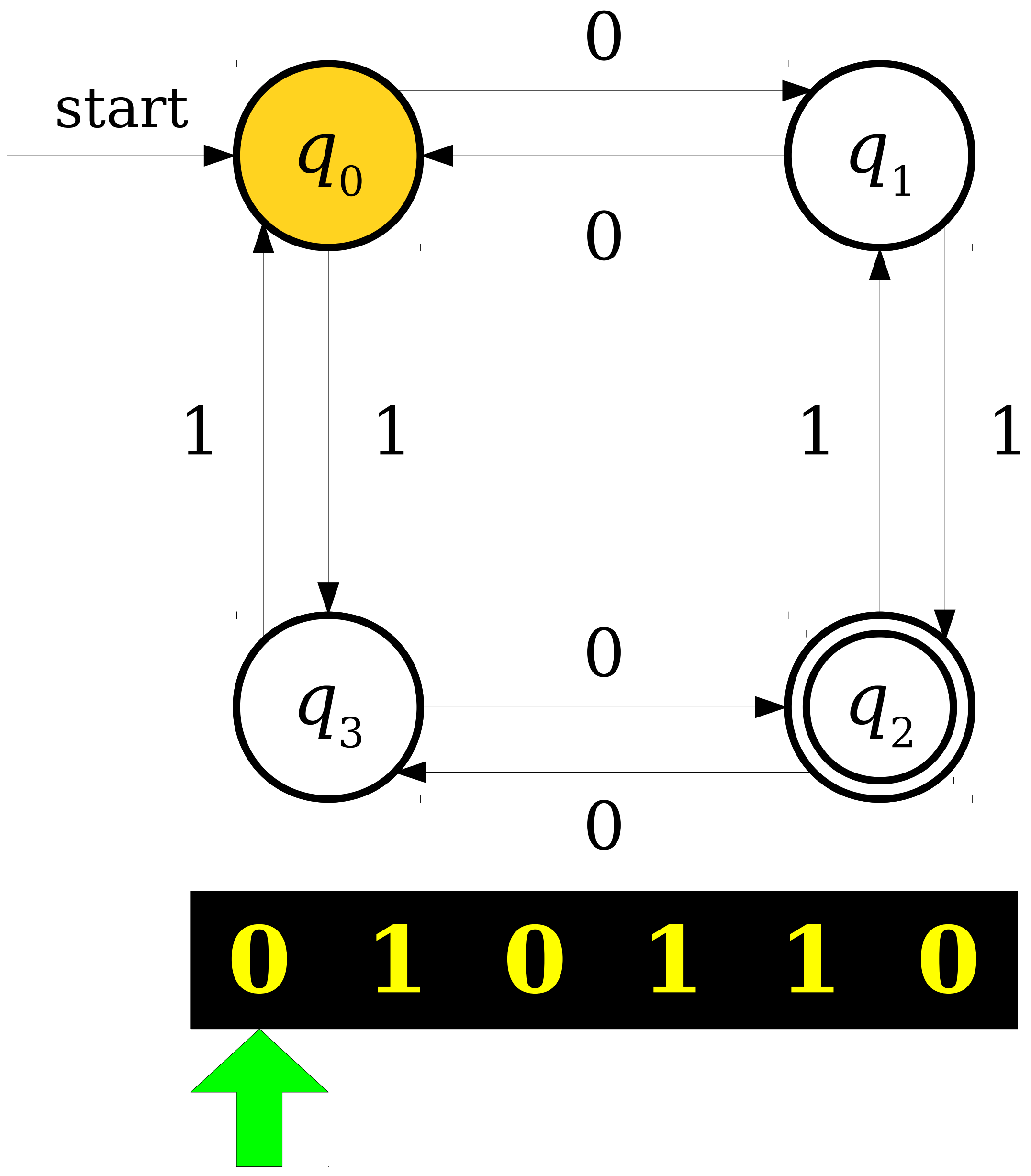


A Simple Finite Automaton

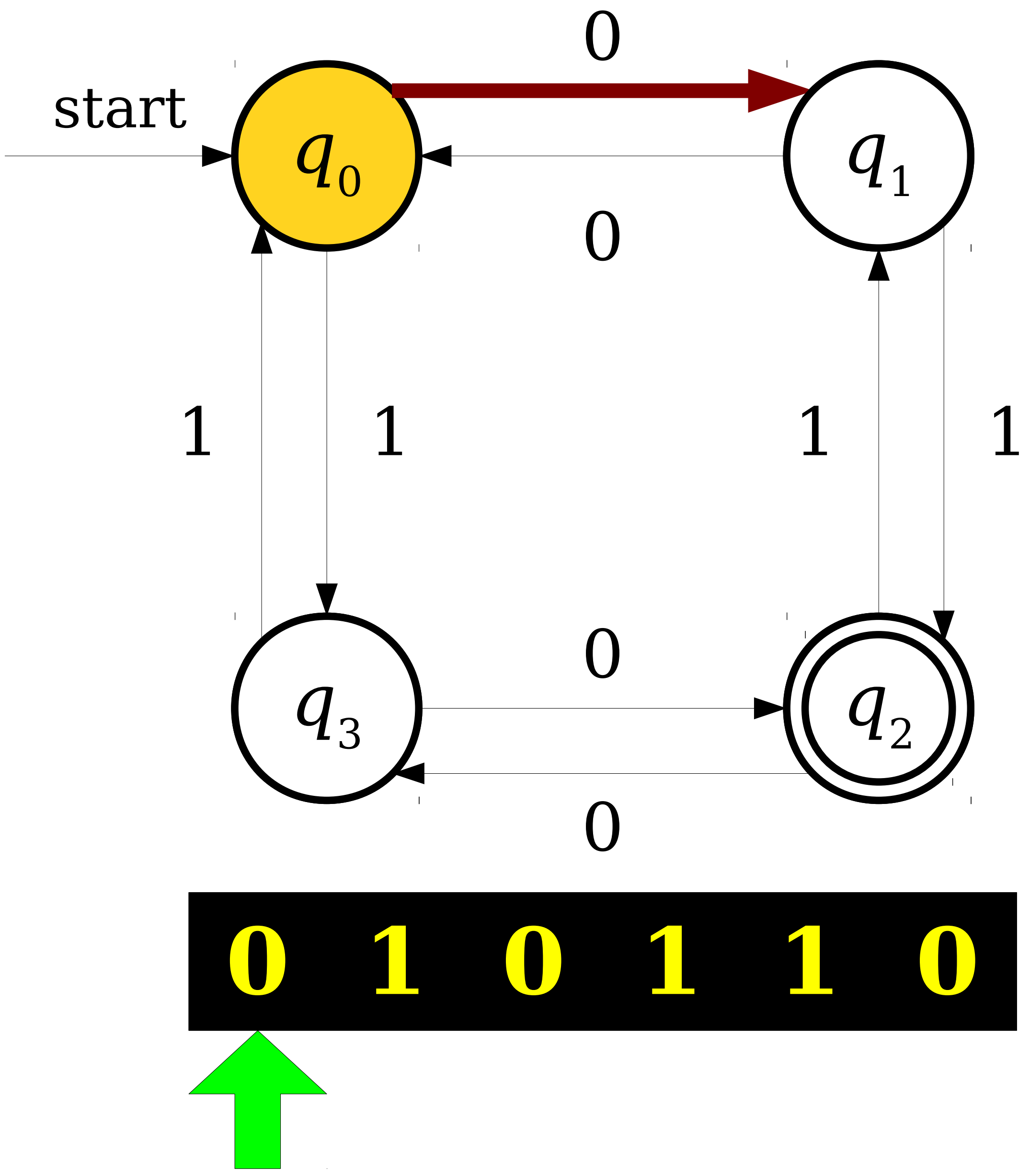


The automaton now begins processing characters in the order in which they appear.

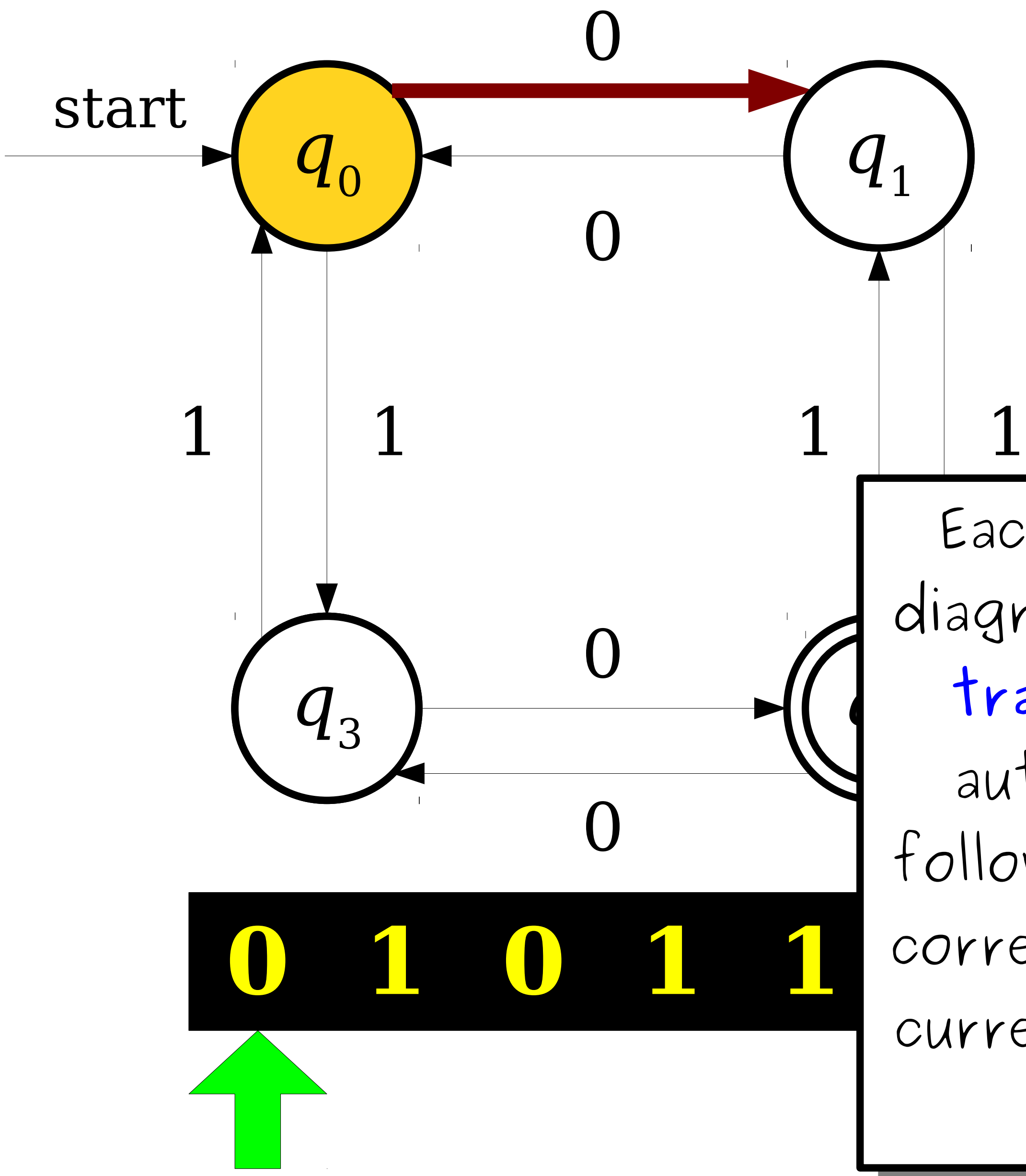
A Simple Finite Automaton



A Simple Finite Automaton

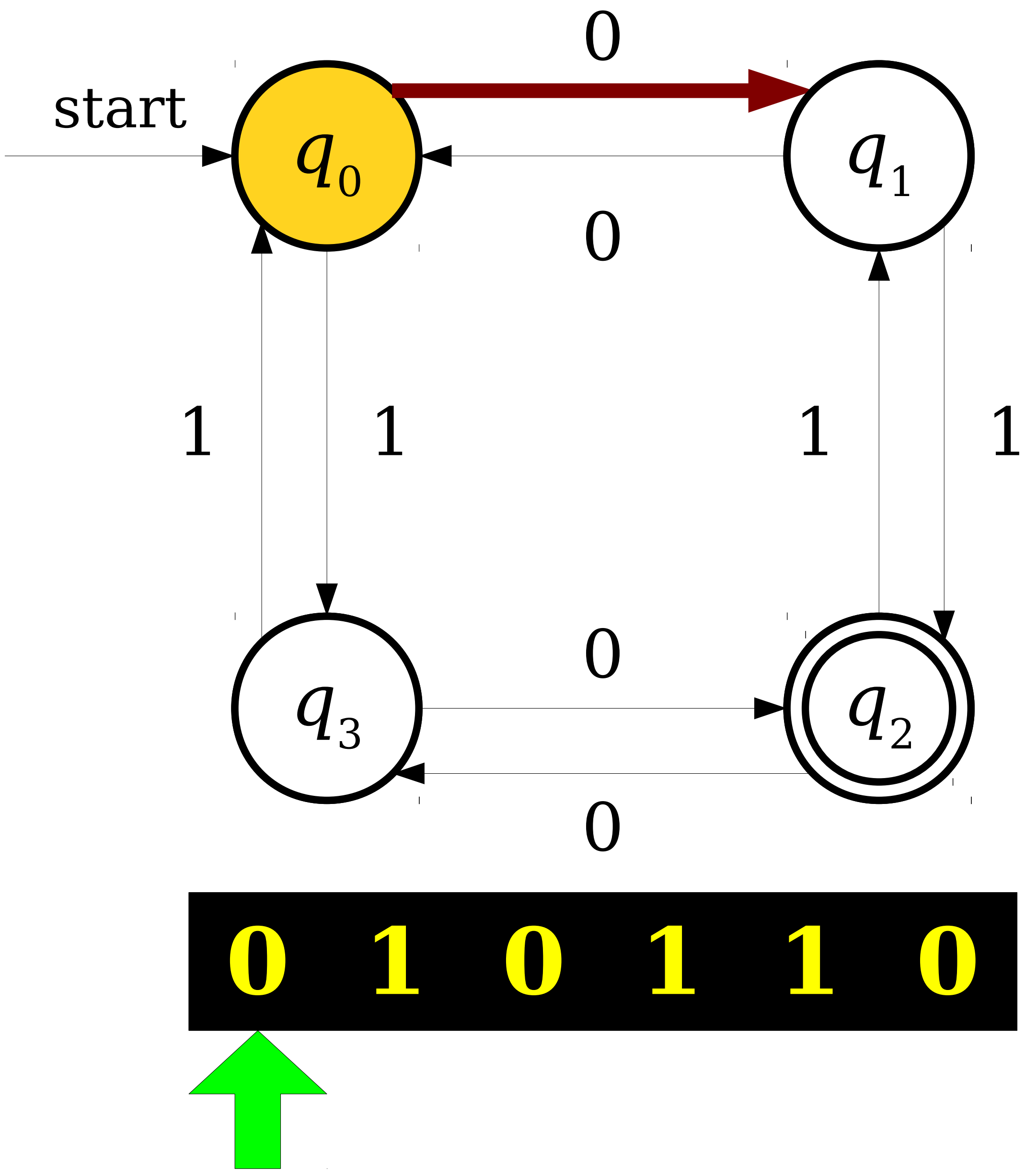


A Simple Finite Automaton

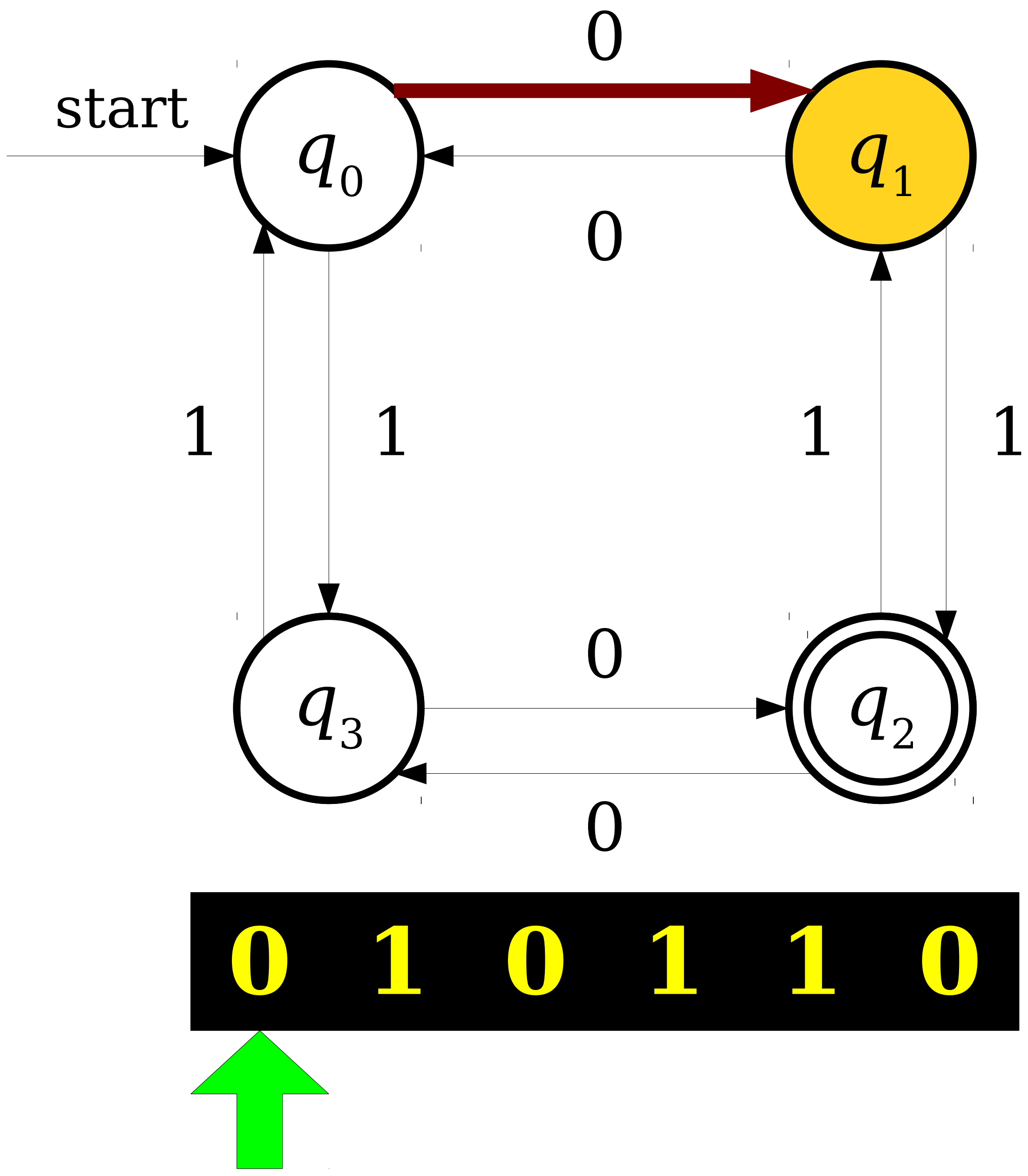


Each arrow in this diagram represents a transition. The automaton always follows the transition corresponding to the current symbol being read.

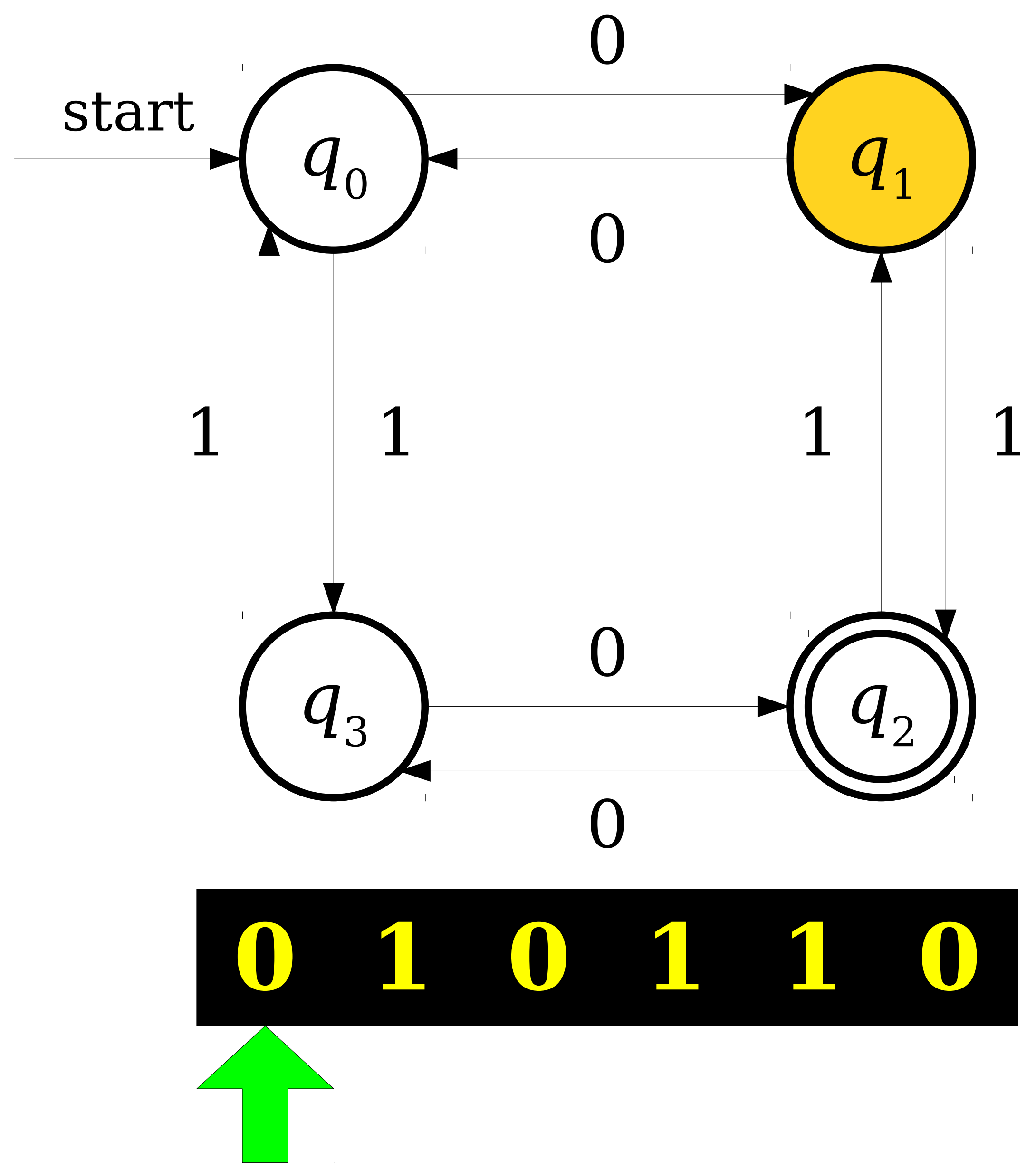
A Simple Finite Automaton



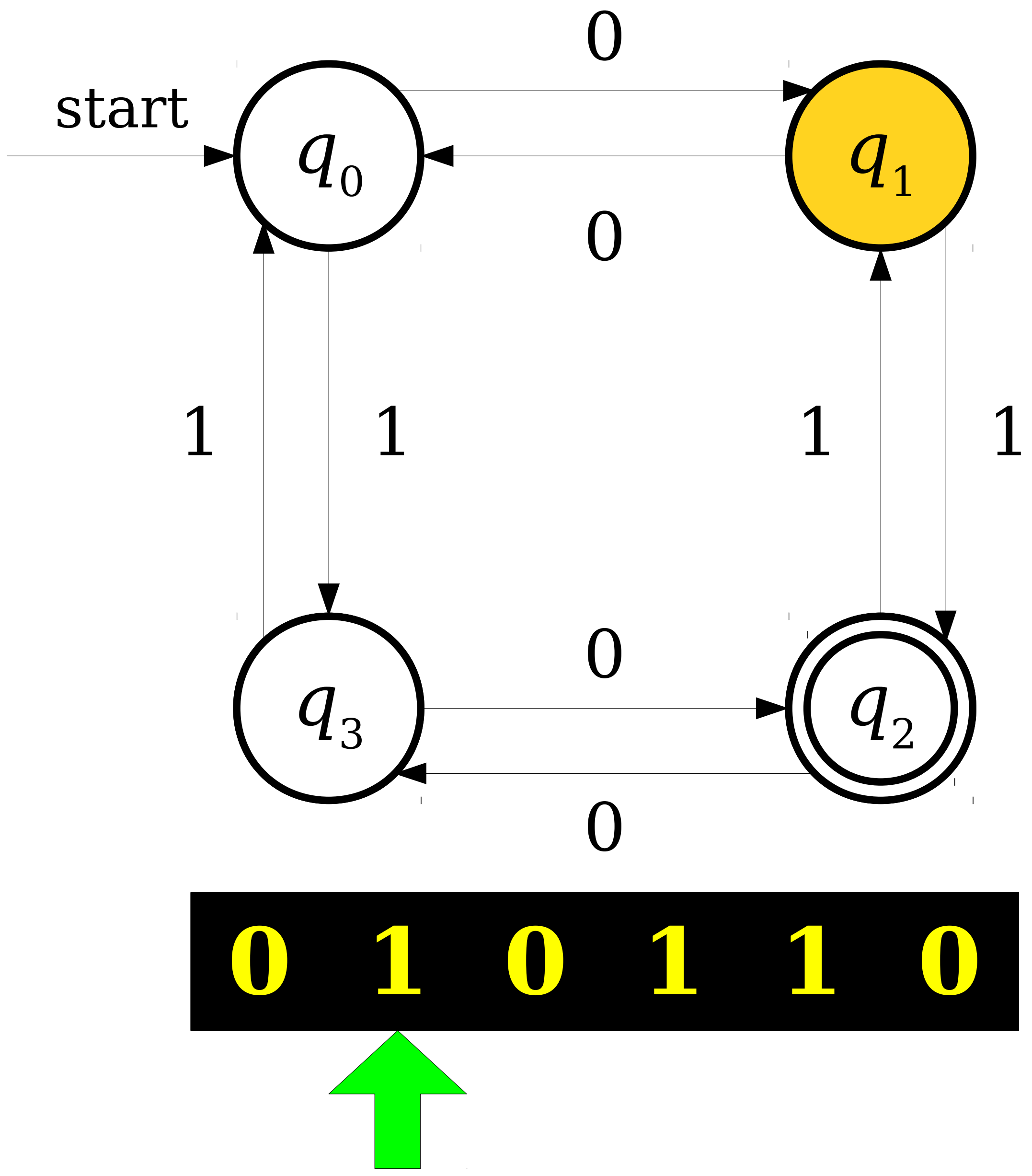
A Simple Finite Automaton



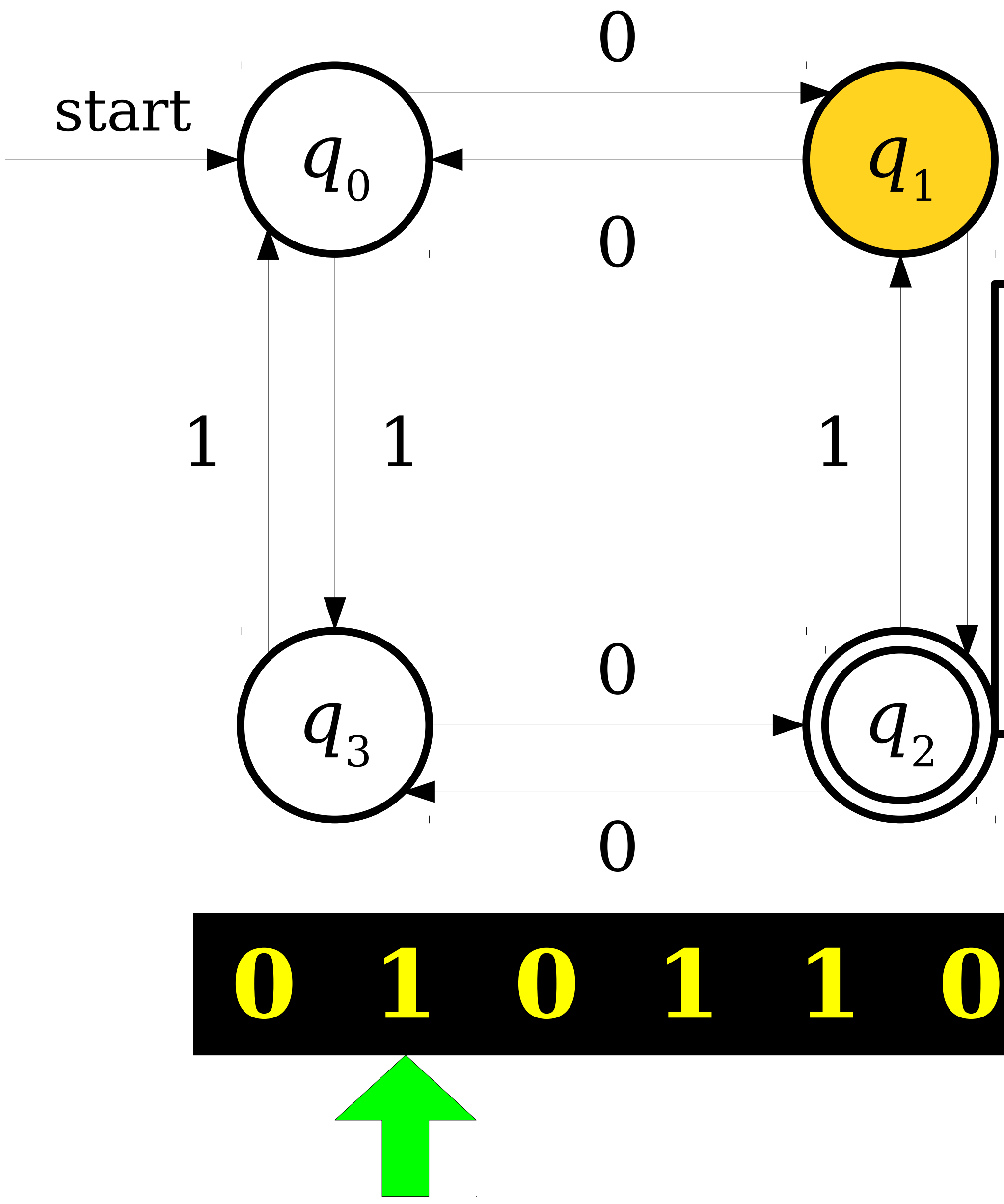
A Simple Finite Automaton



A Simple Finite Automaton

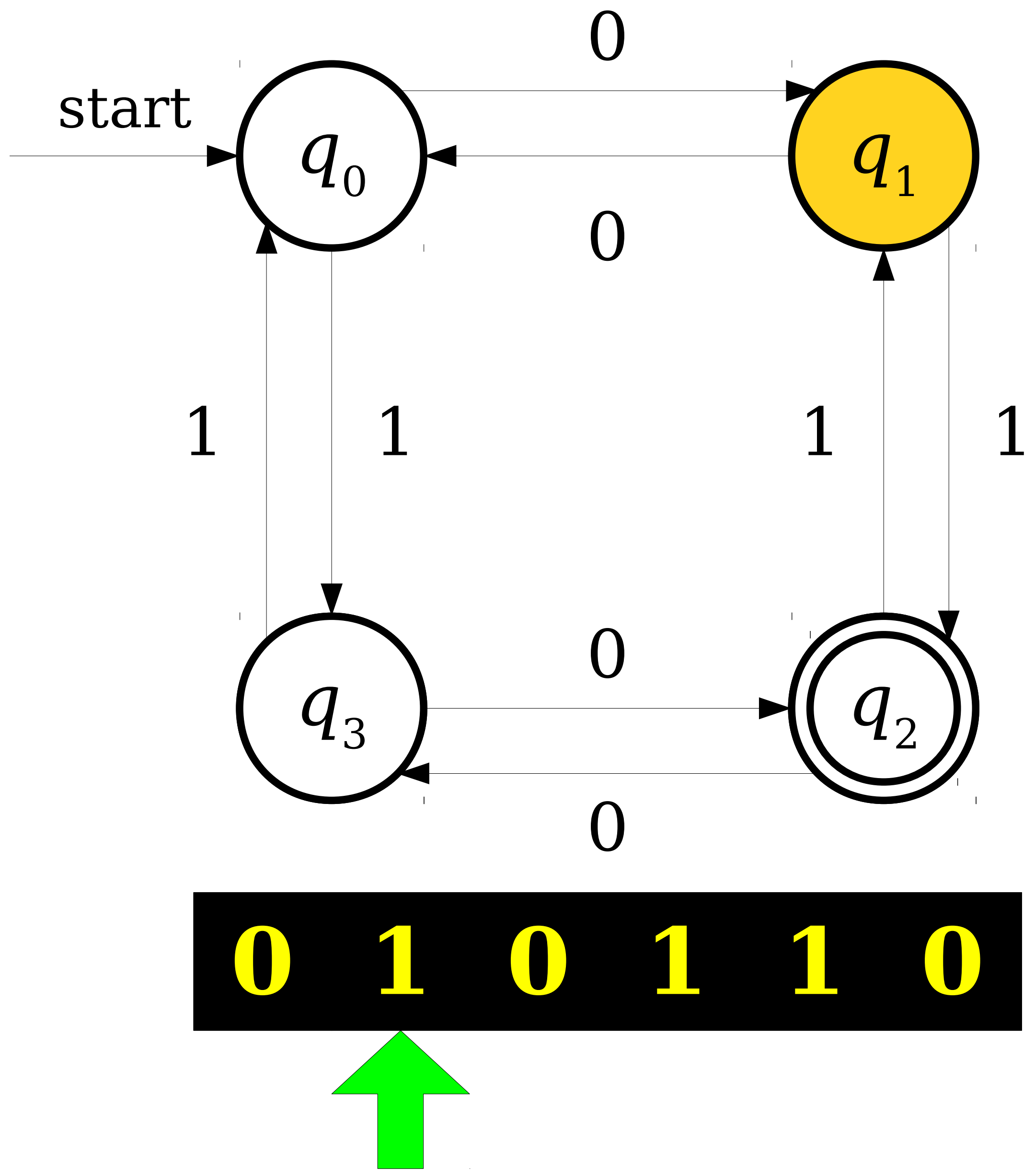


A Simple Finite Automaton

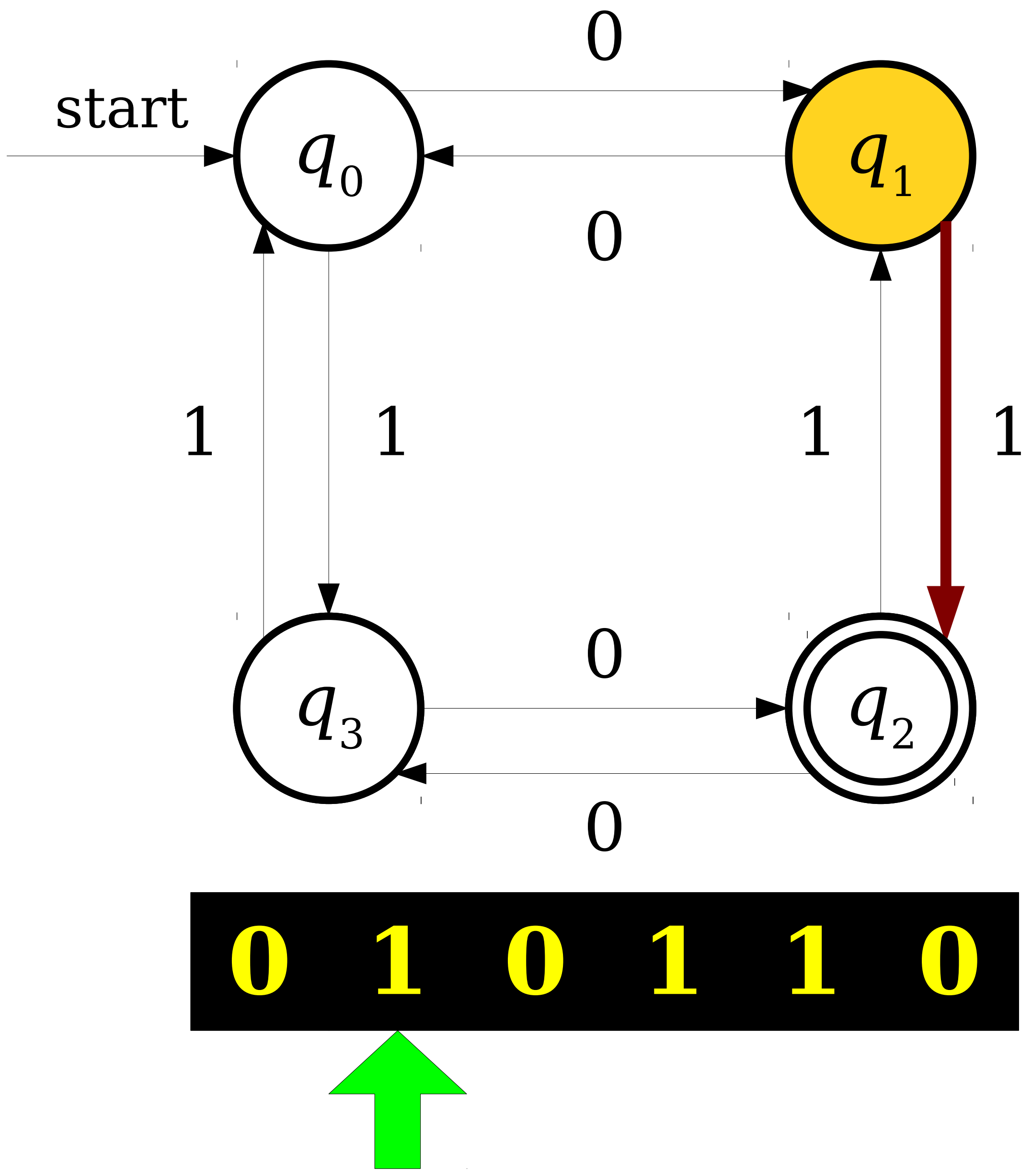


After transitioning, the automaton considers the next symbol in the input.

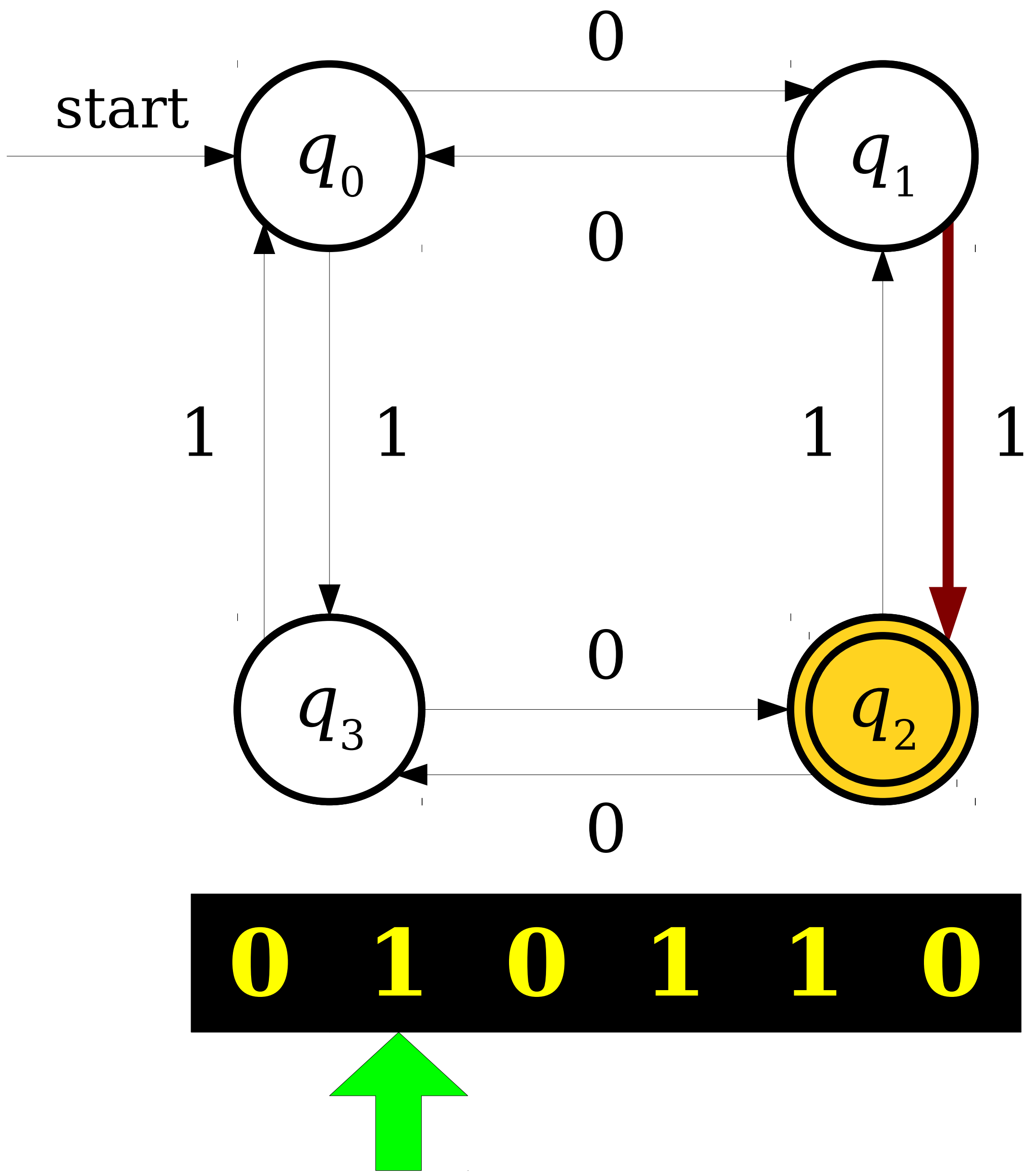
A Simple Finite Automaton



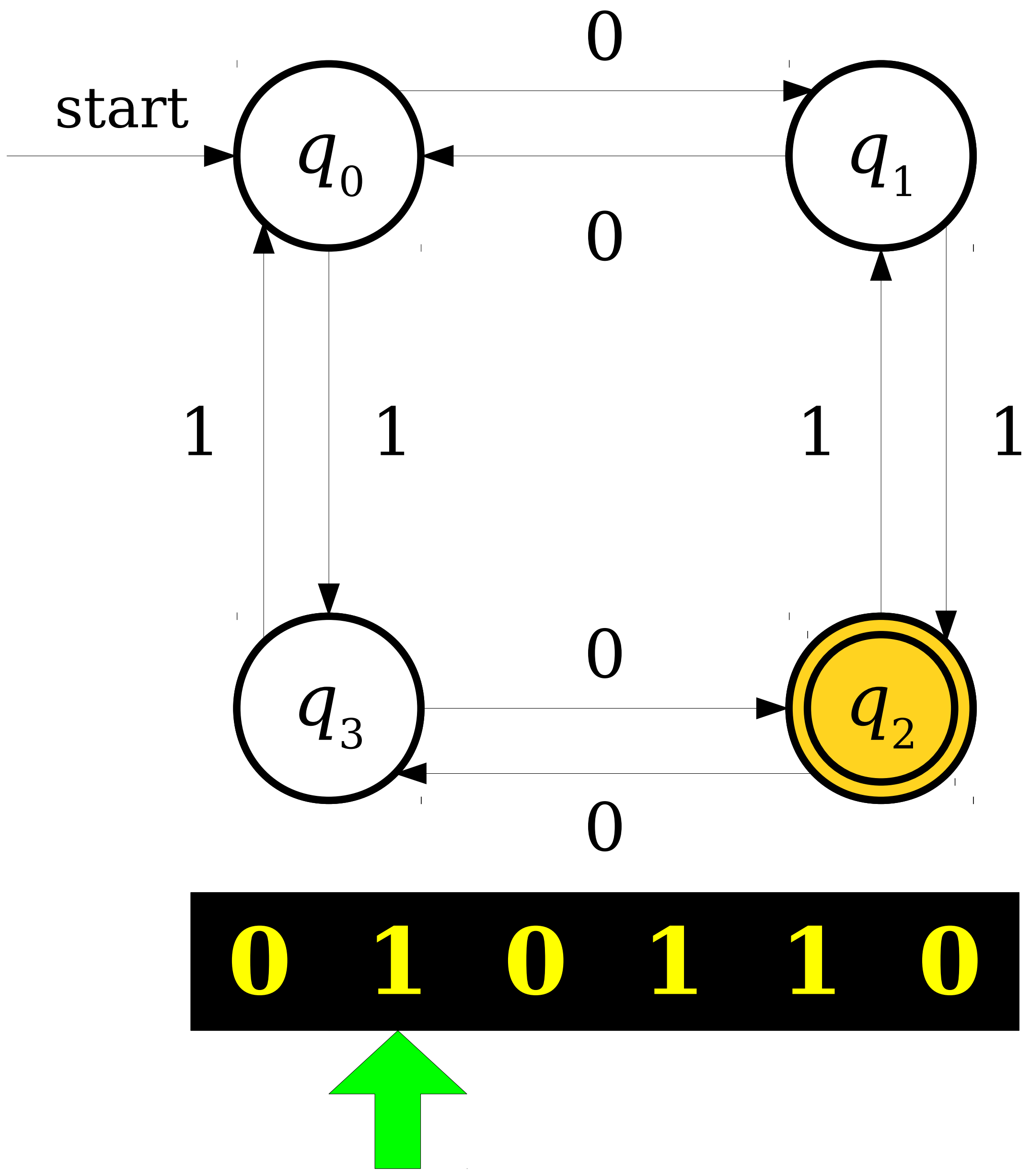
A Simple Finite Automaton



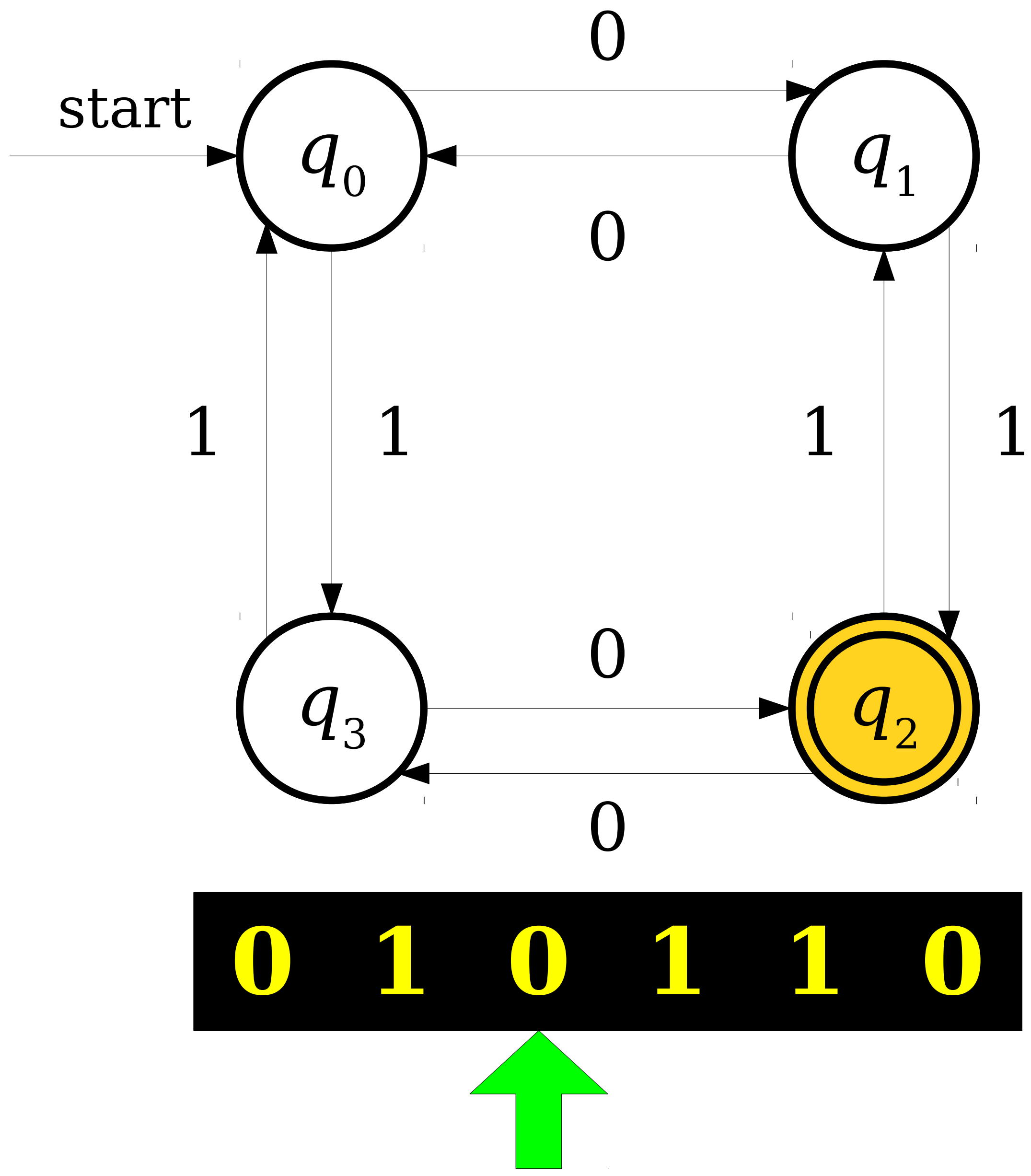
A Simple Finite Automaton



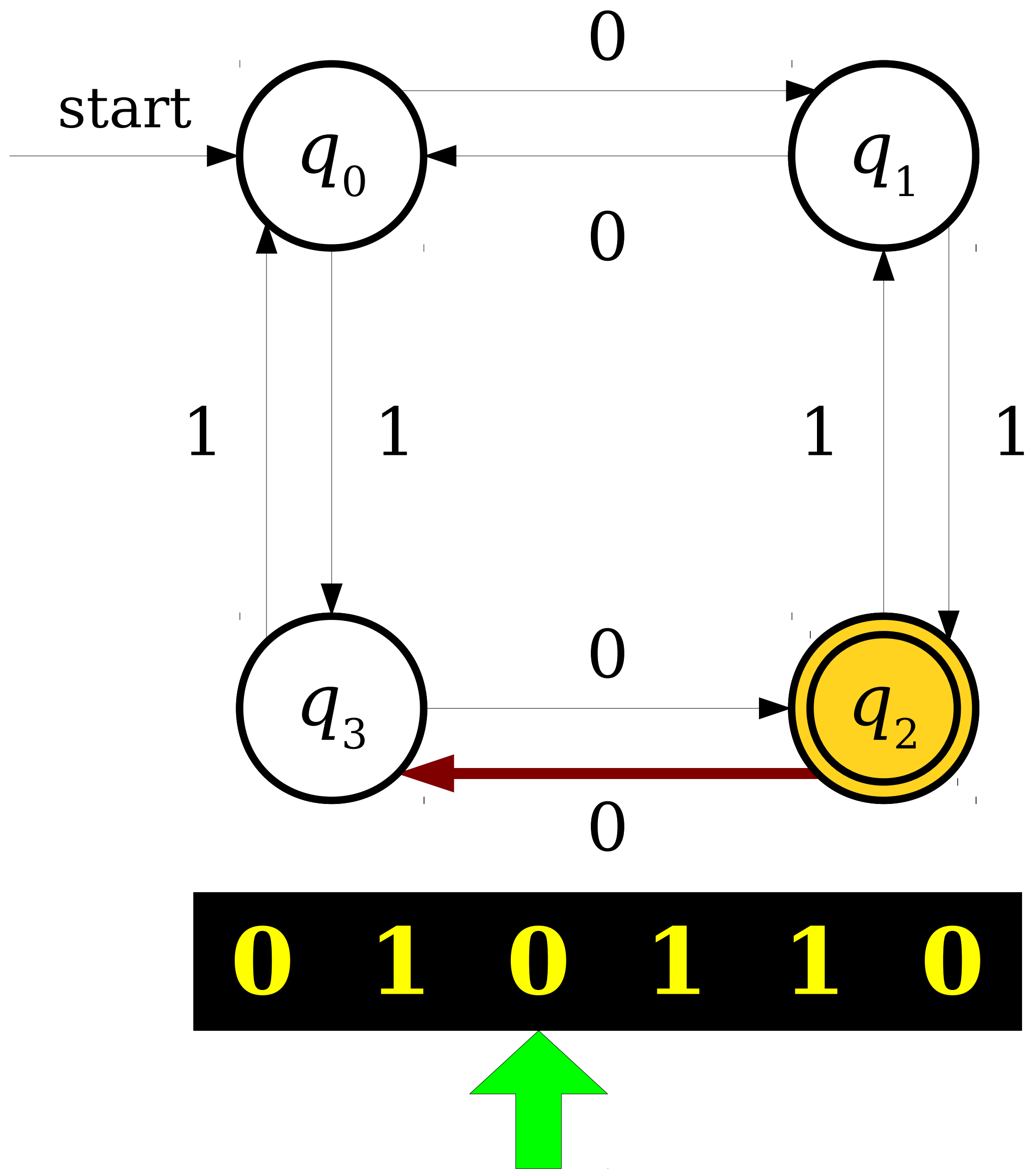
A Simple Finite Automaton



A Simple Finite Automaton



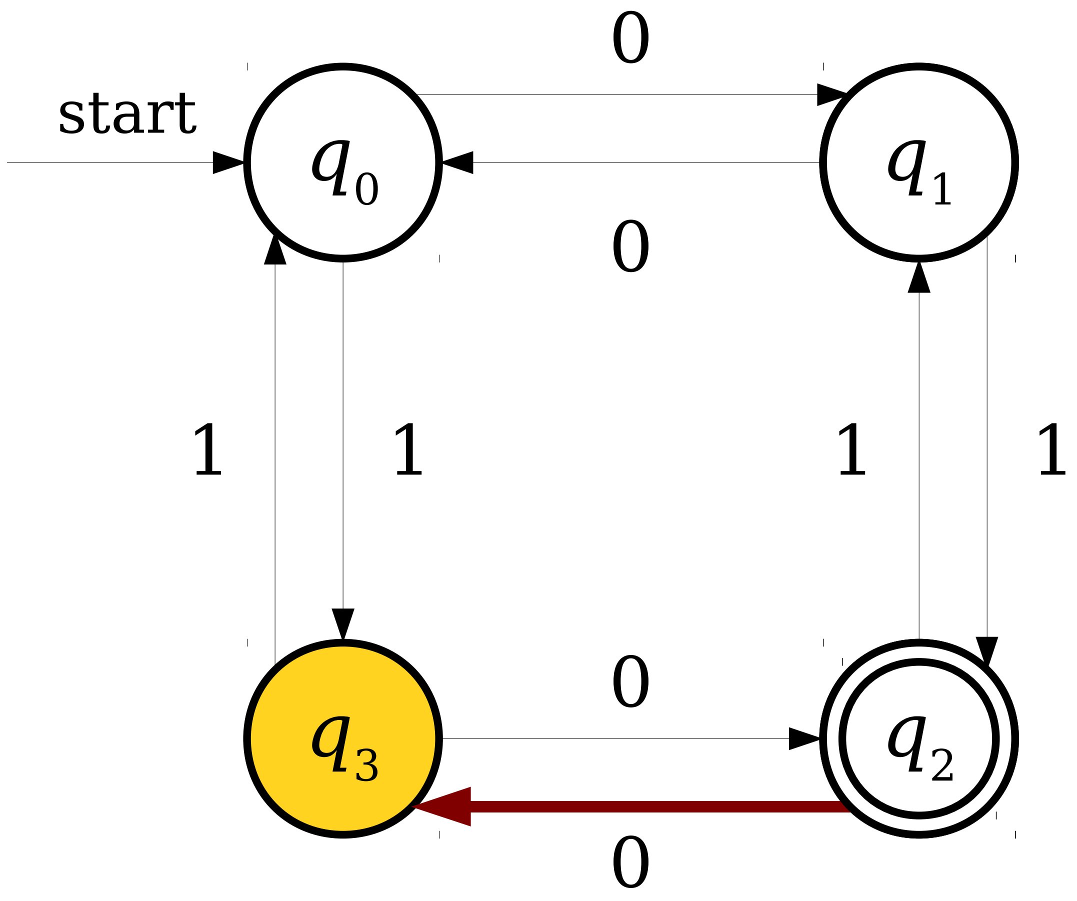
A Simple Finite Automaton



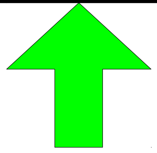
$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[r, bend left, "0"] \arrow[u, bend left, "1"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

A Simple Finite Automaton

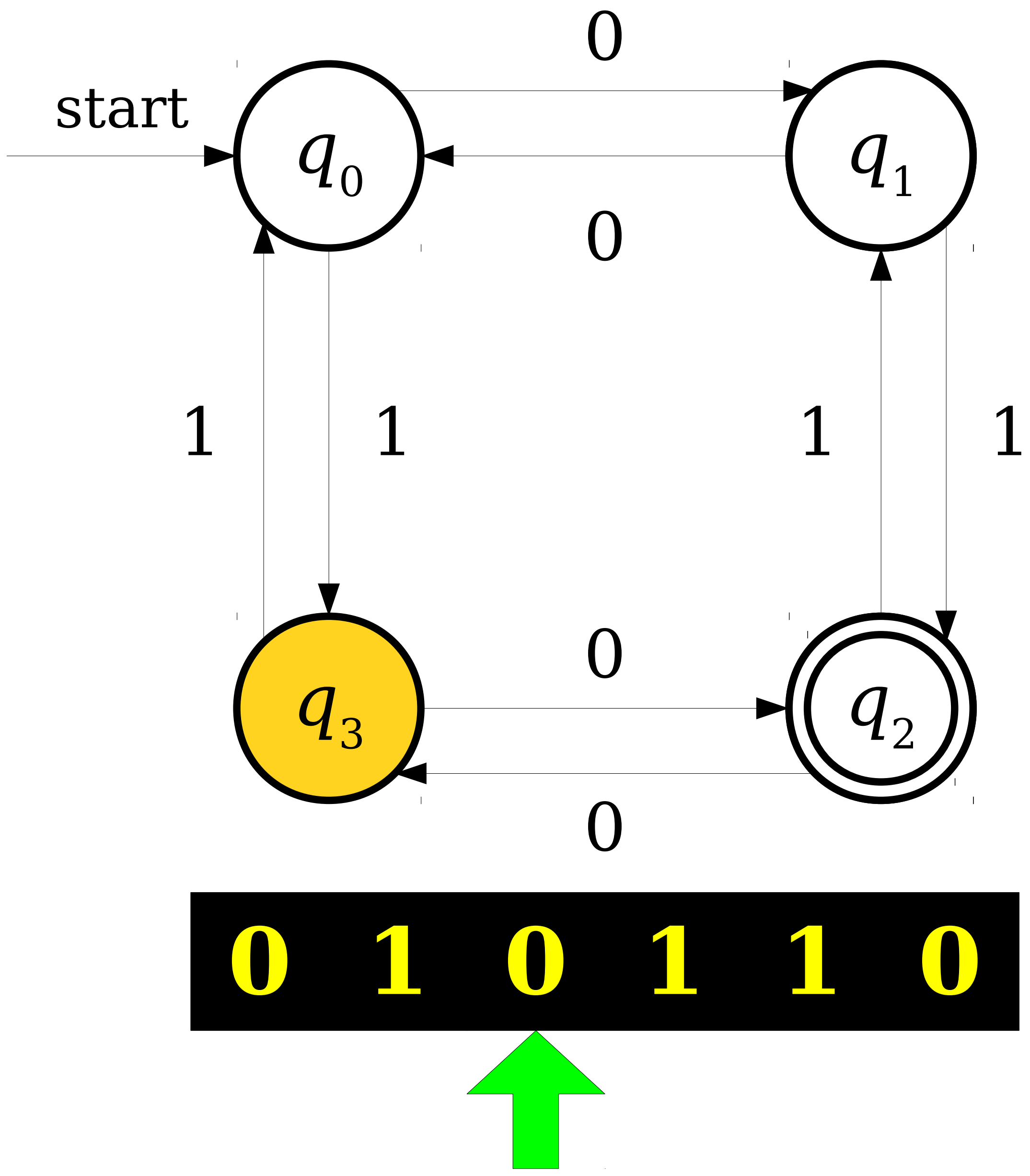
$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[rr, bend left, "0"] \arrow[d, bend left, "1"] & & q_1 \arrow[ll, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[rr, bend left, "0"] & & q_2 \arrow[ll, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$



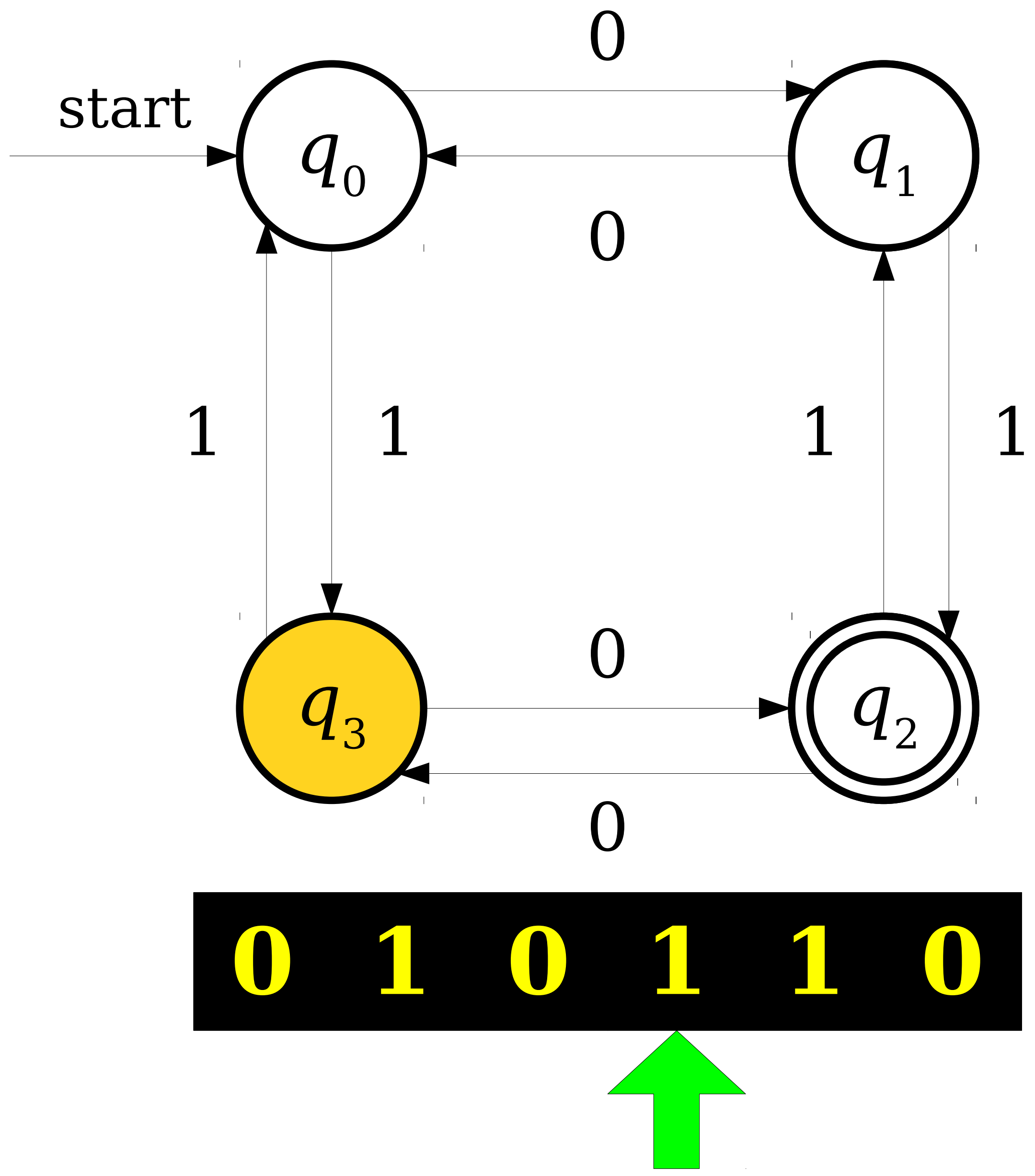
0 1 0 1 1 0



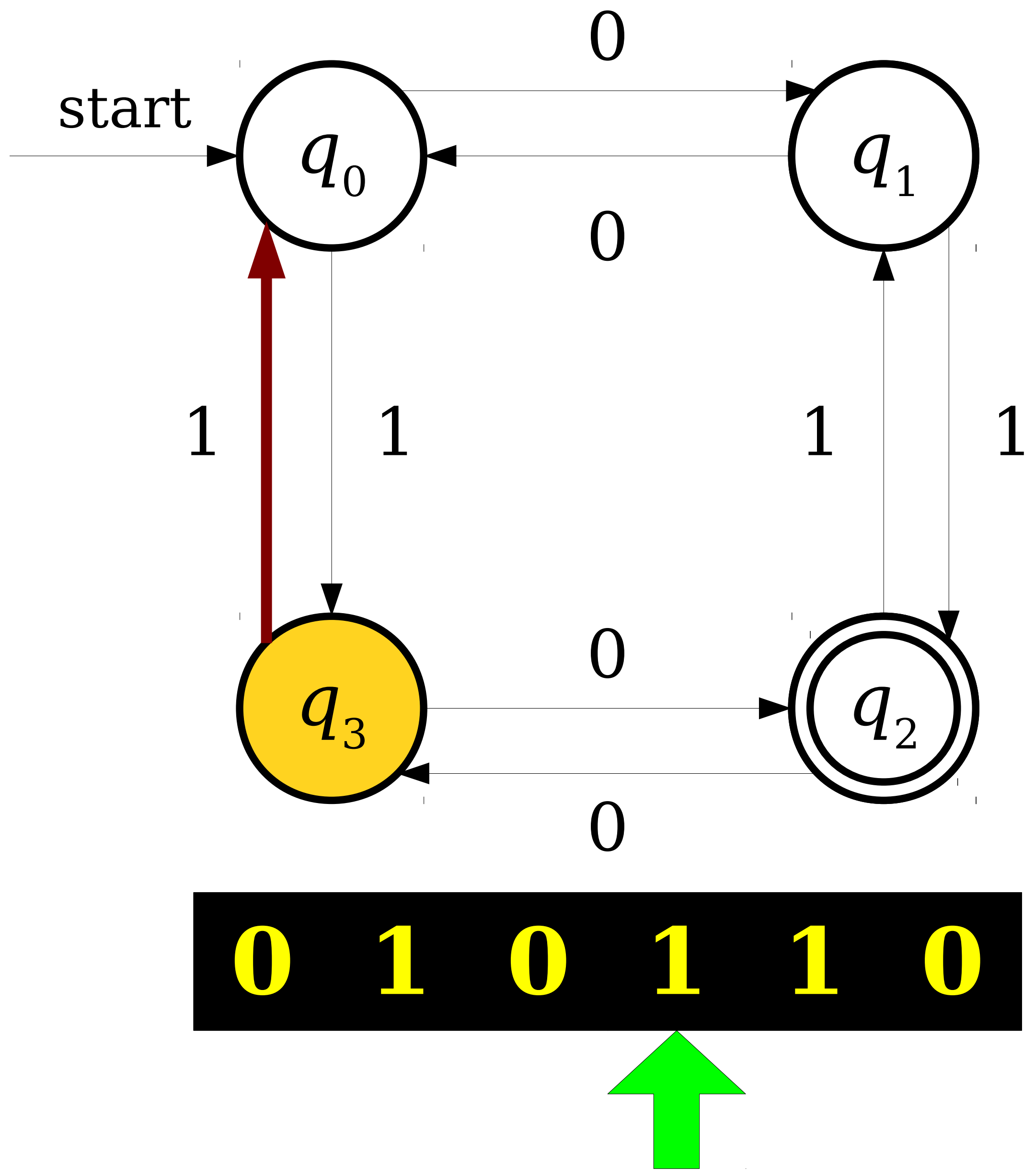
A Simple Finite Automaton



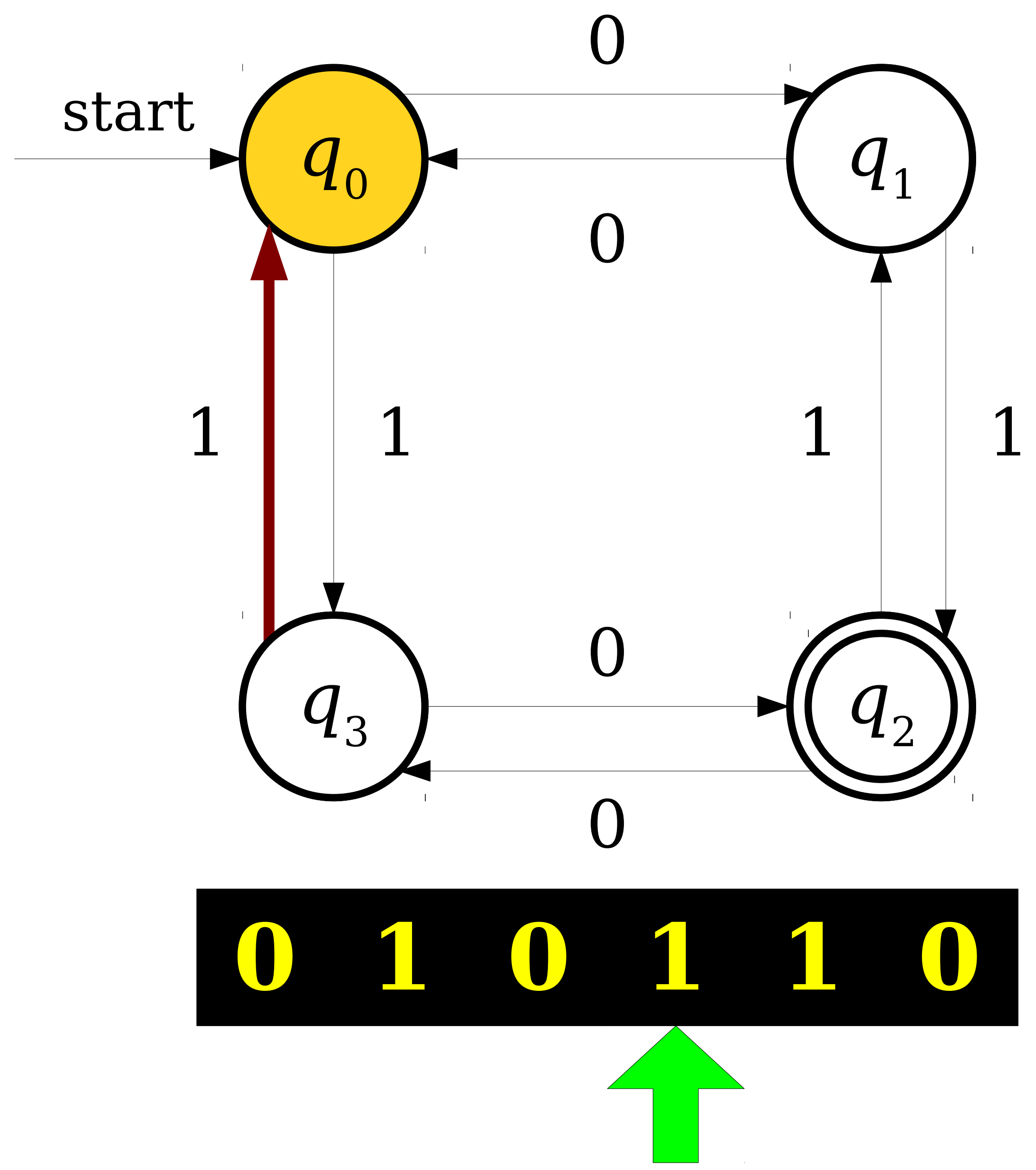
A Simple Finite Automaton



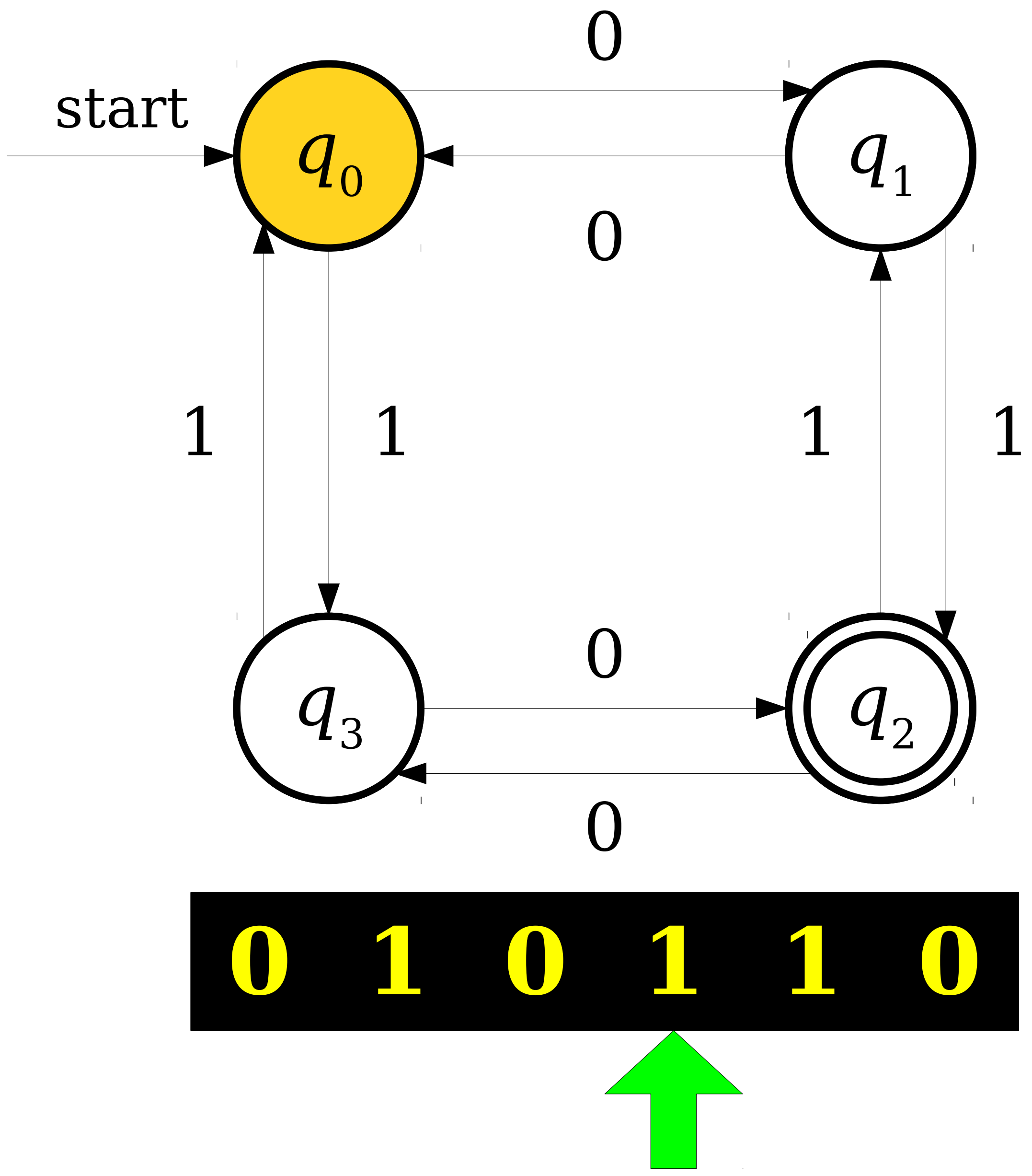
A Simple Finite Automaton



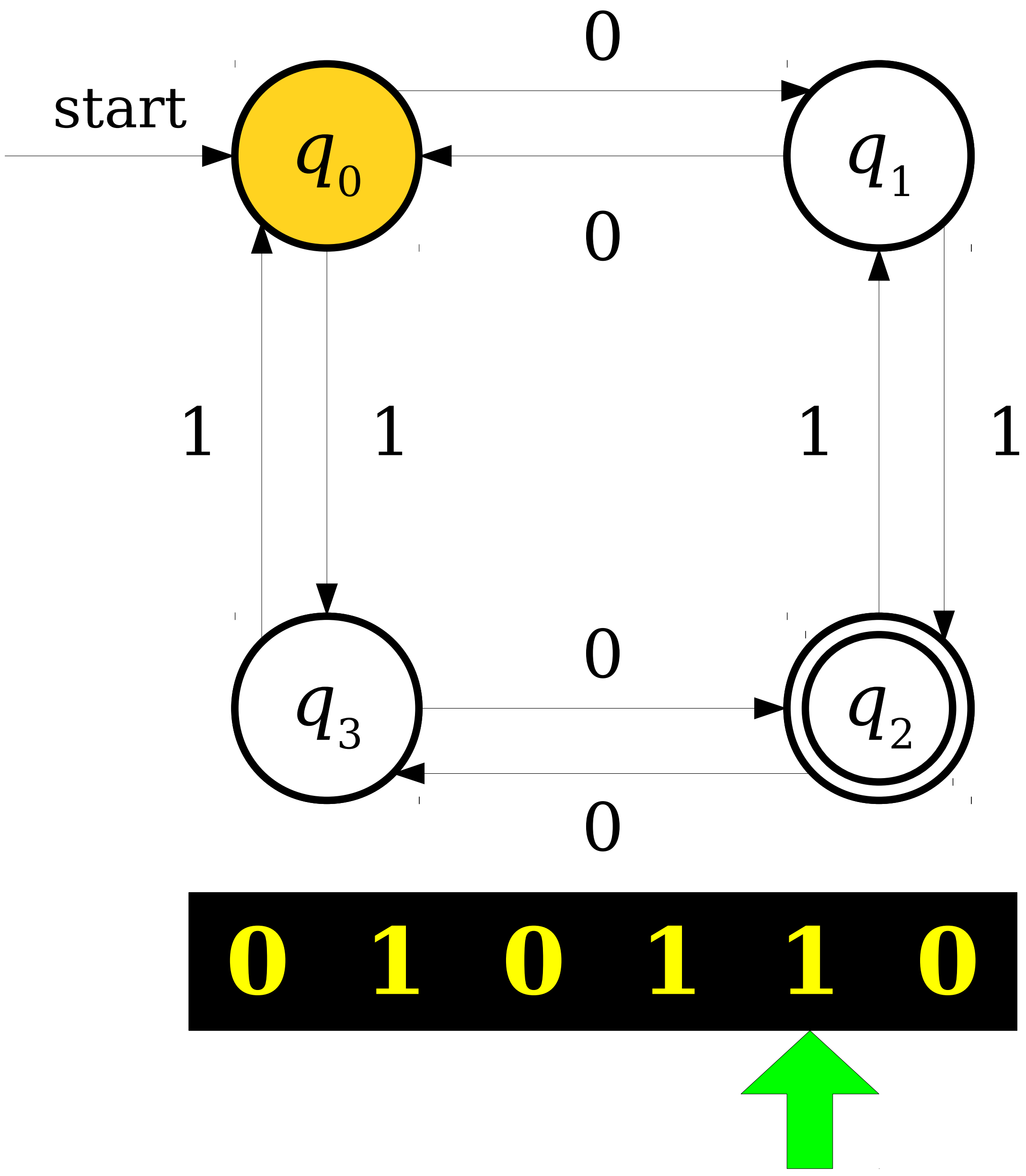
A Simple Finite Automaton



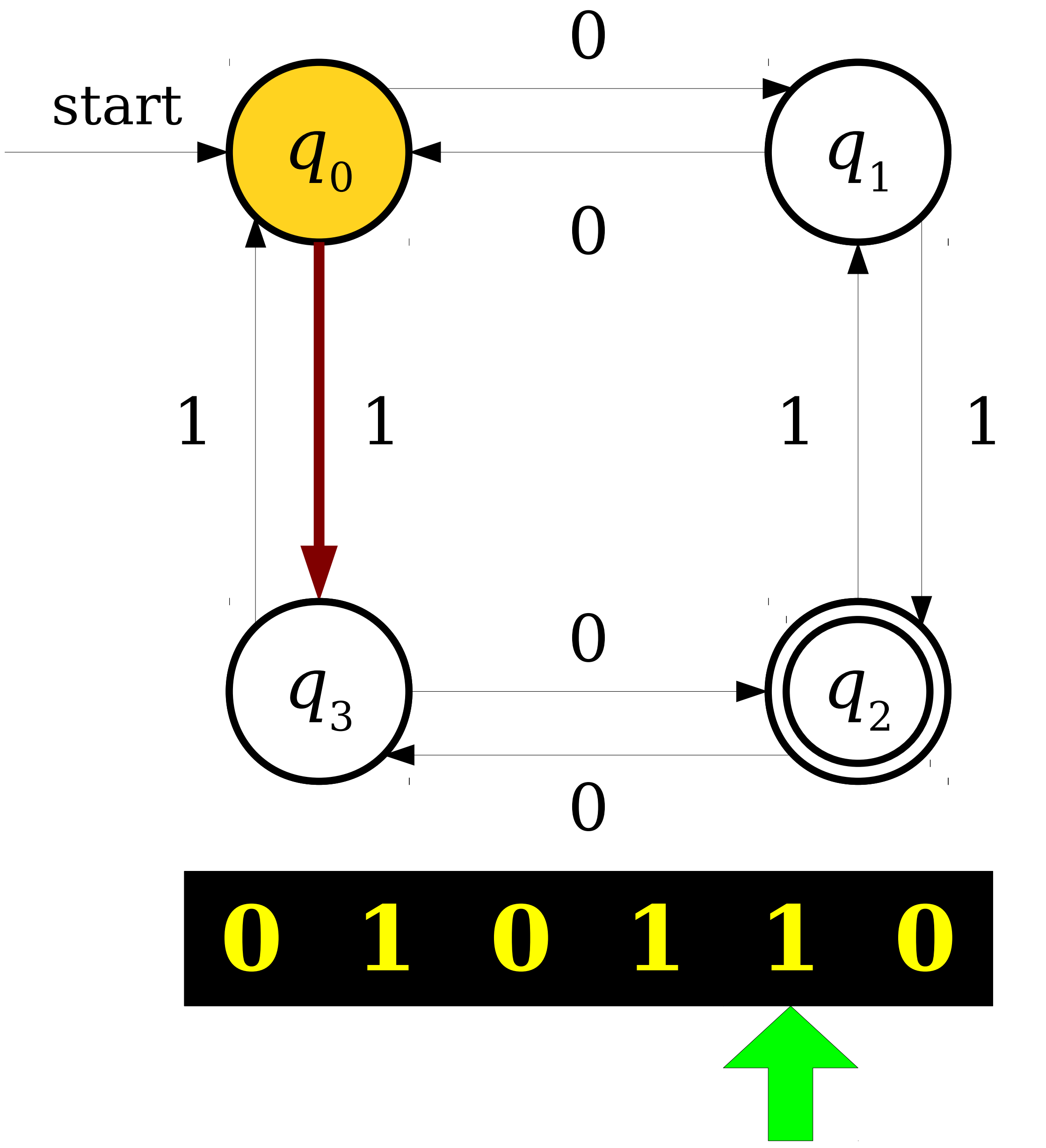
A Simple Finite Automaton



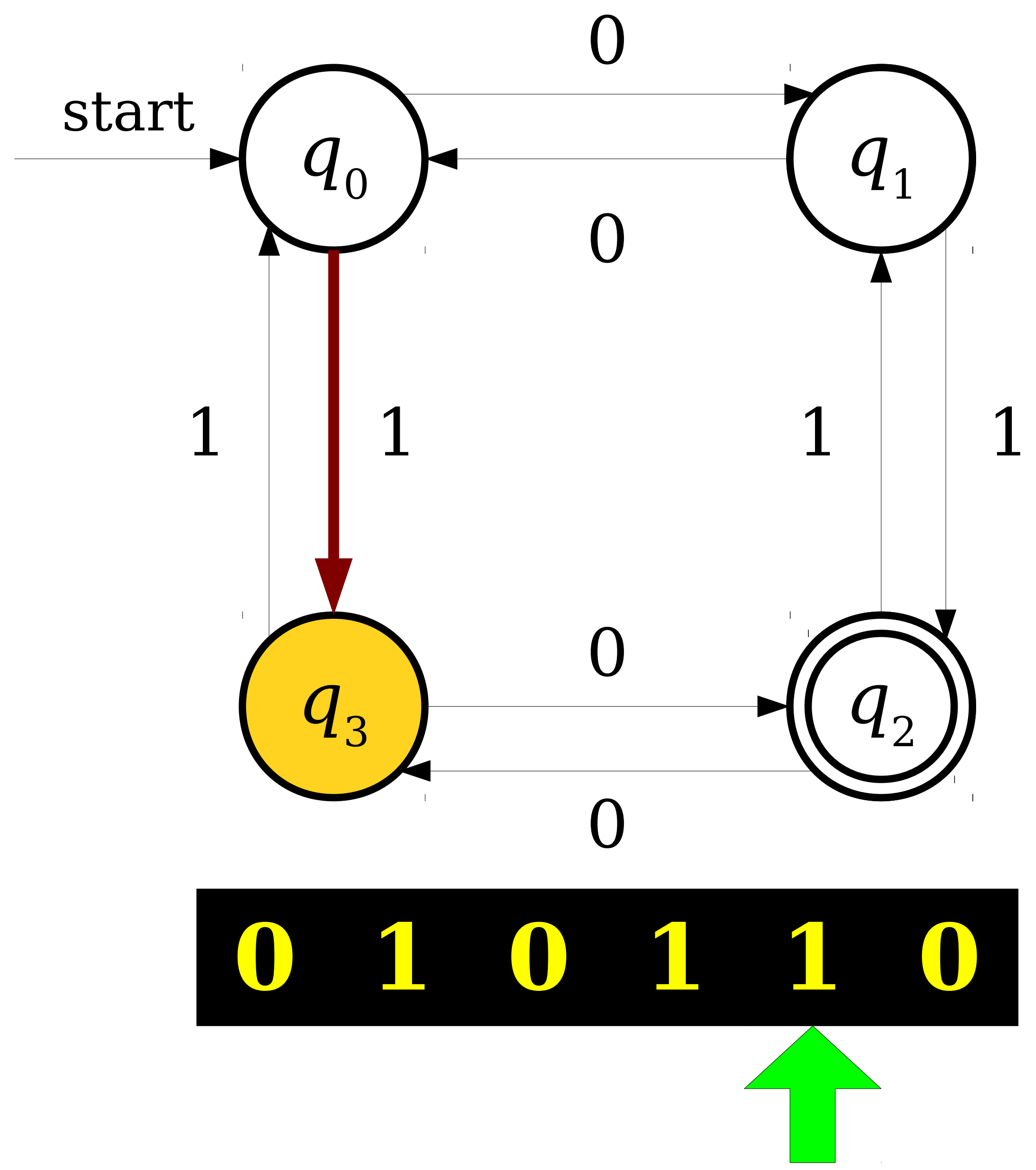
A Simple Finite Automaton



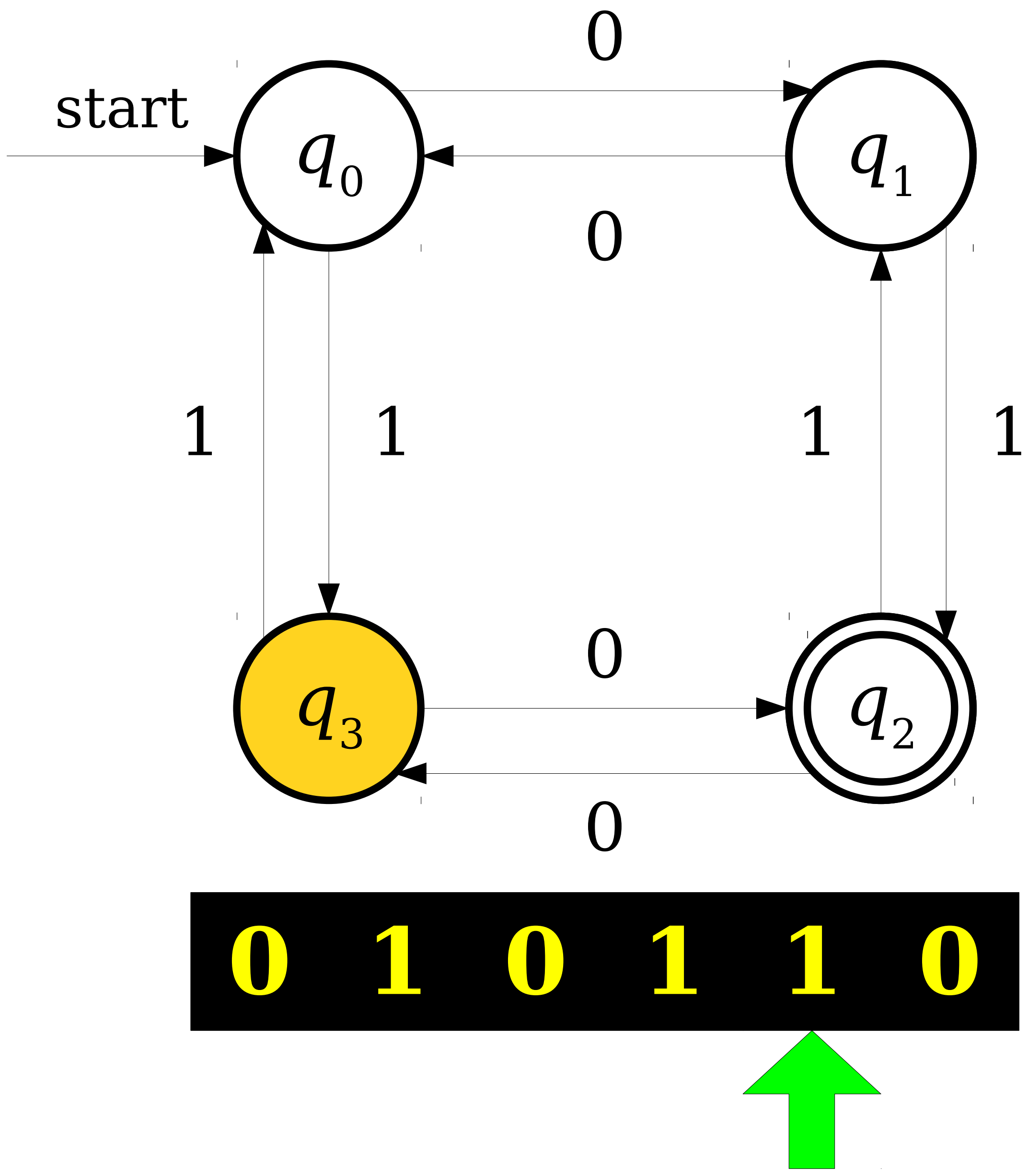
A Simple Finite Automaton



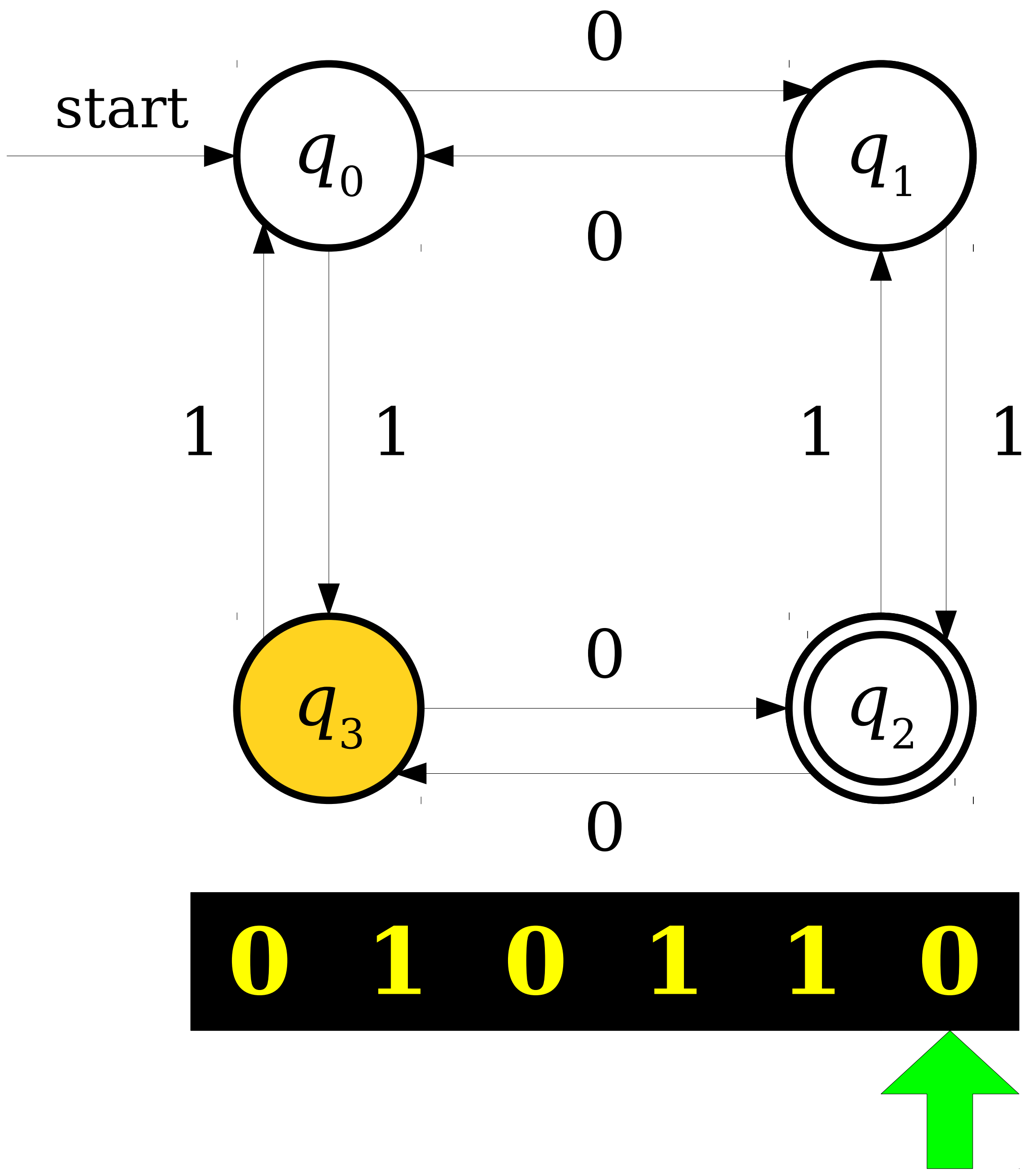
A Simple Finite Automaton



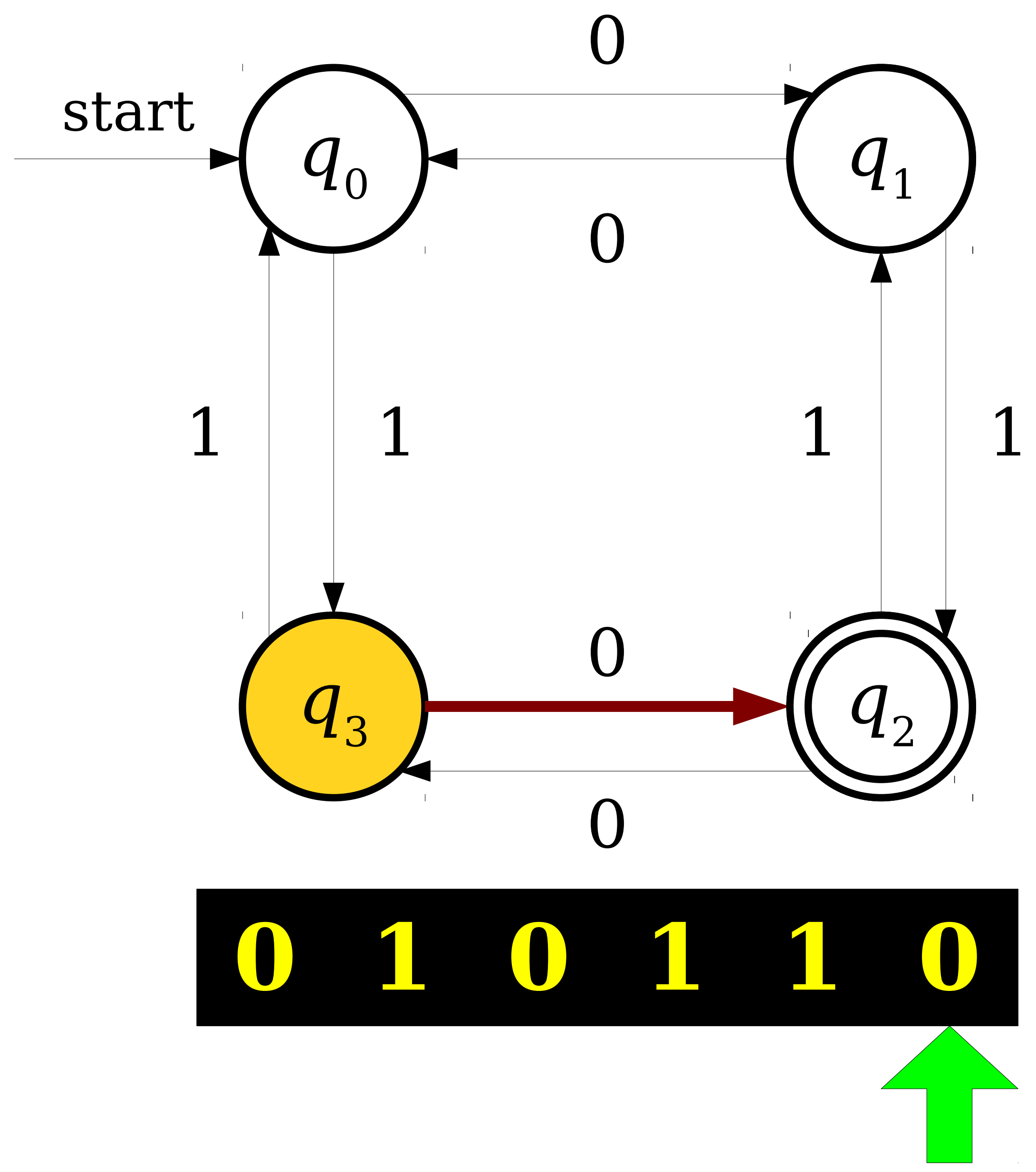
A Simple Finite Automaton



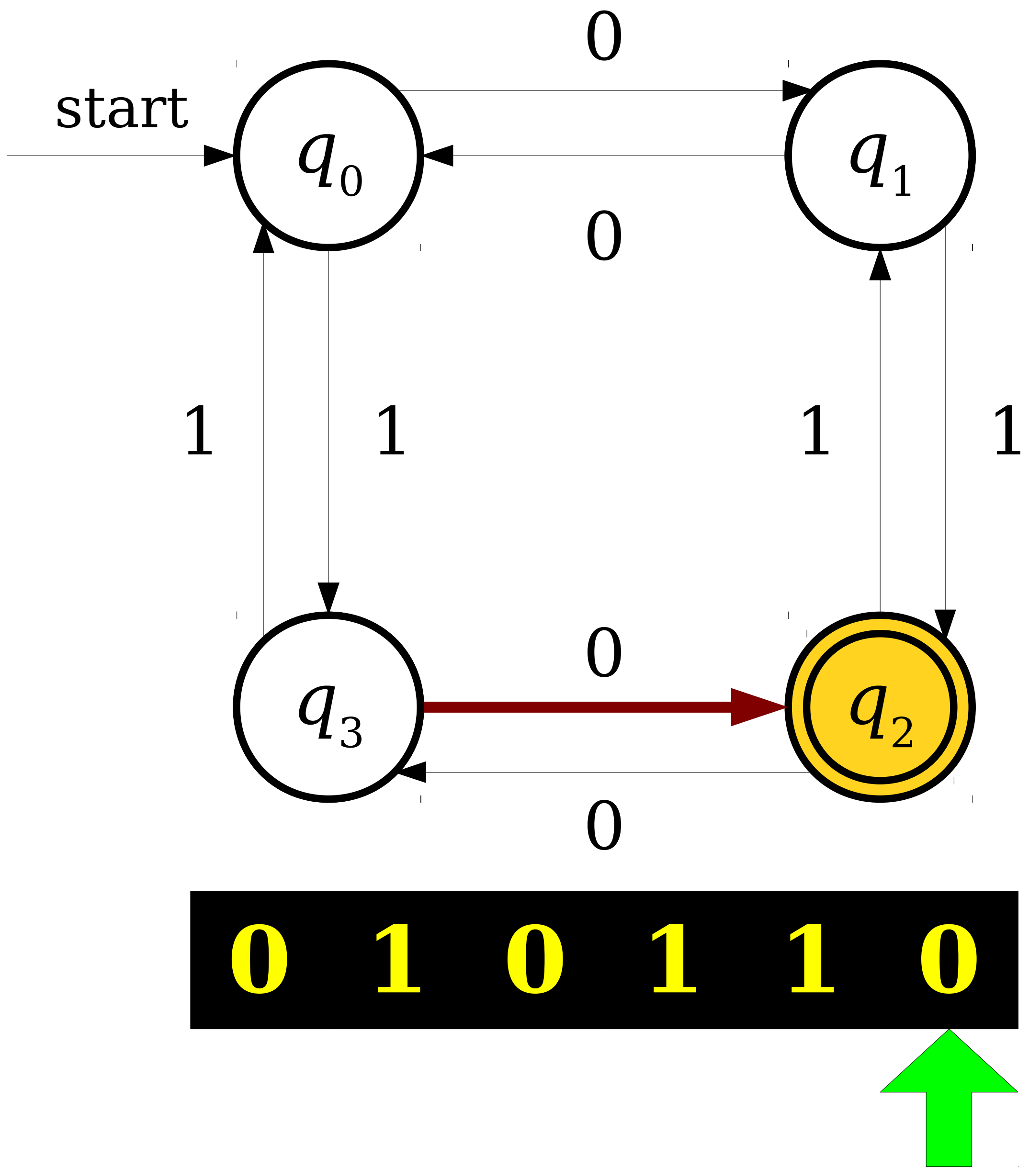
A Simple Finite Automaton



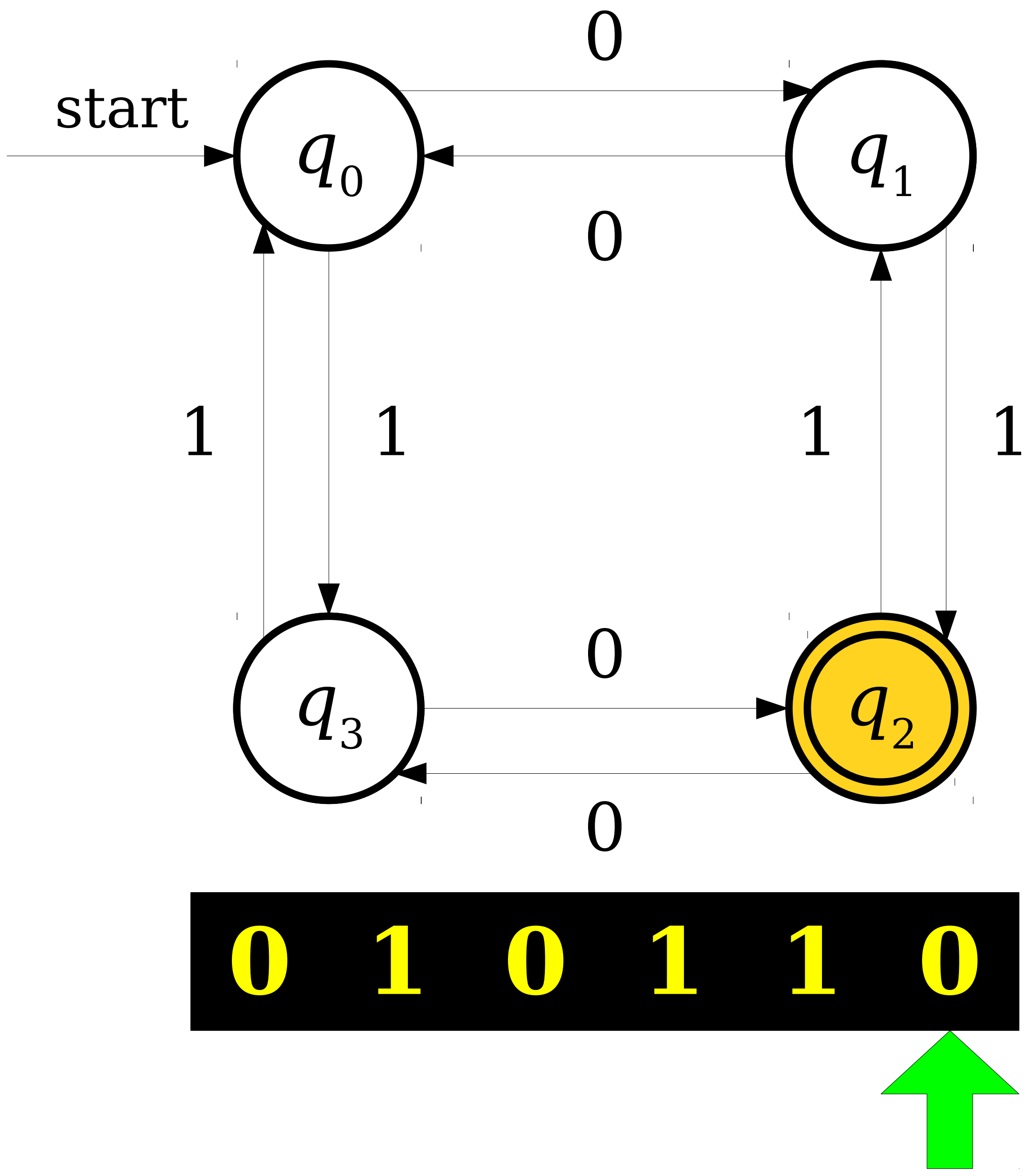
A Simple Finite Automaton



A Simple Finite Automaton



A Simple Finite Automaton



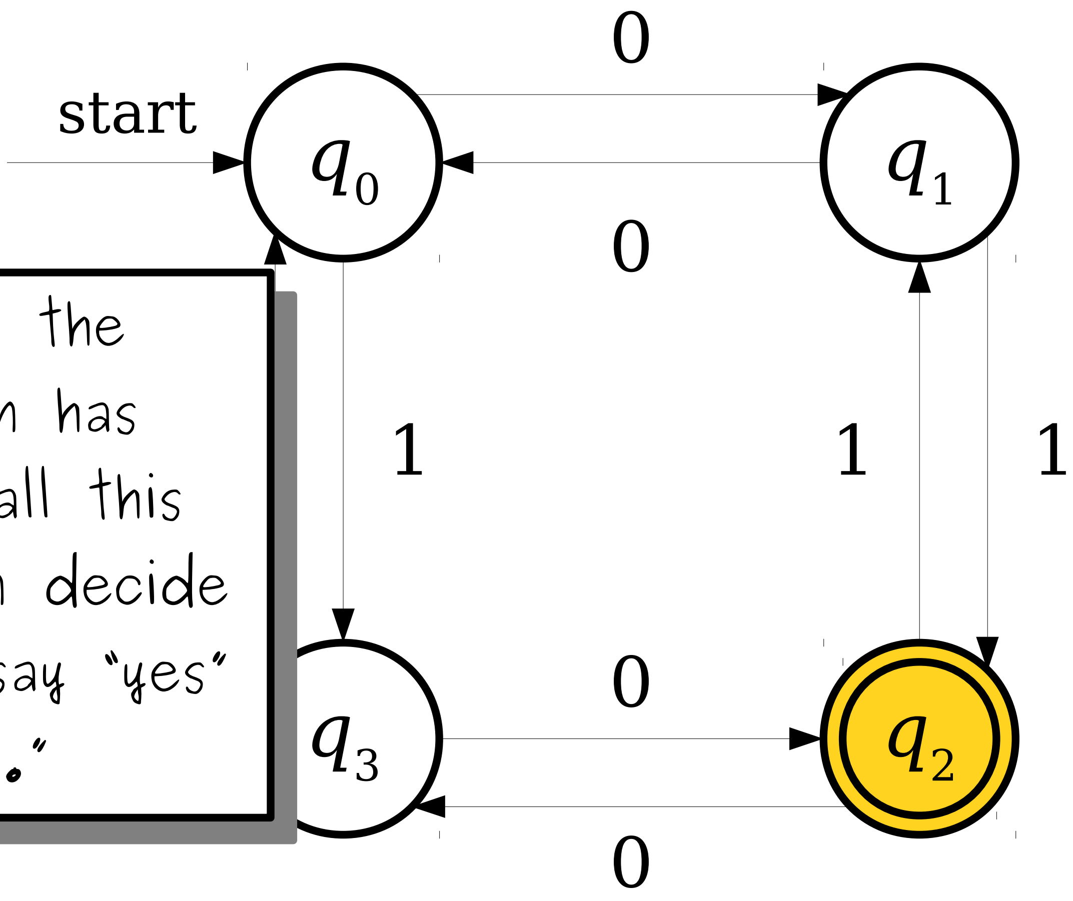
$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

A Simple Finite Automaton

$$\begin{tikzcd}
{} \arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[u, bend left, "1"] \arrow[l, bend left, "0"]
\end{tikzcd}$$

0 1 0 1 1 0

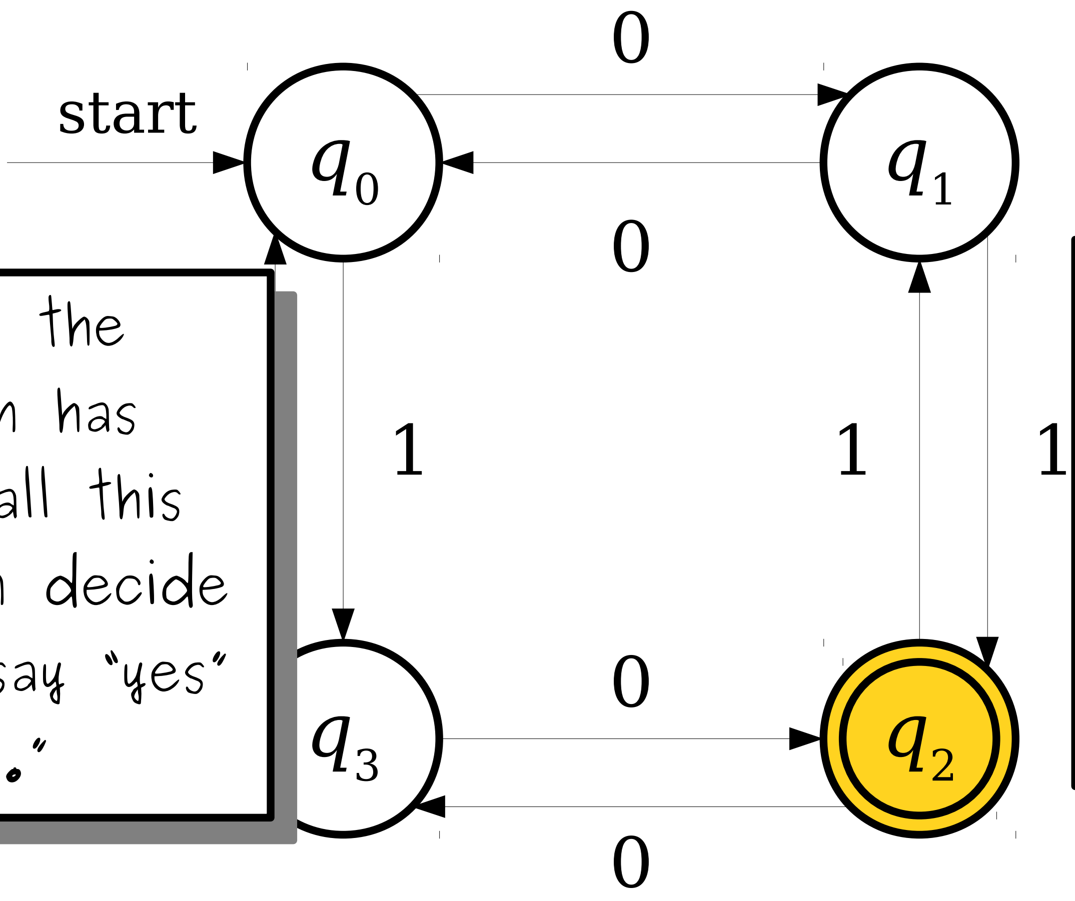
A Simple Finite Automaton



Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

0 1 0 1 1 0

A Simple Finite Automaton

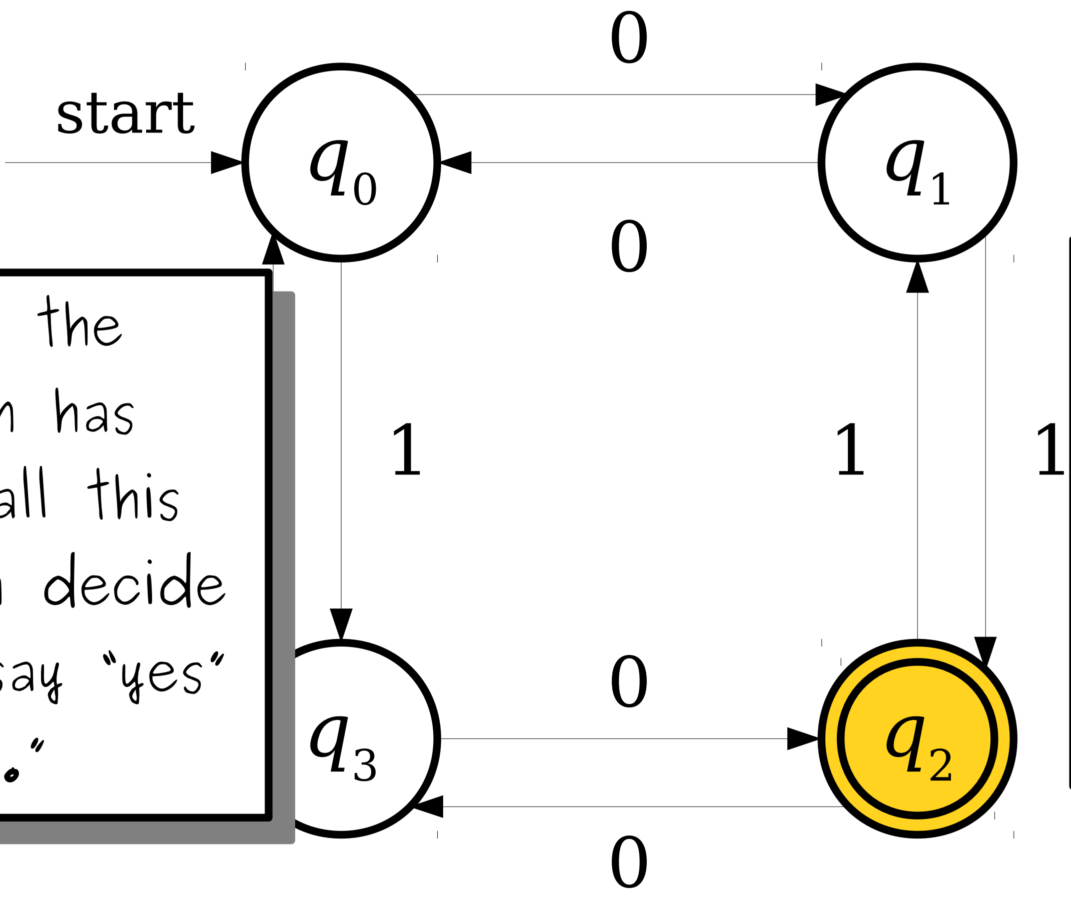


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

The double circle indicates that this state is an **accepting state**, so the automaton outputs "yes."

0 1 0 1 1 0

A Simple Finite Automaton


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, "1"] \\
& q_3 \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, "1"]
\end{tikzcd}$$

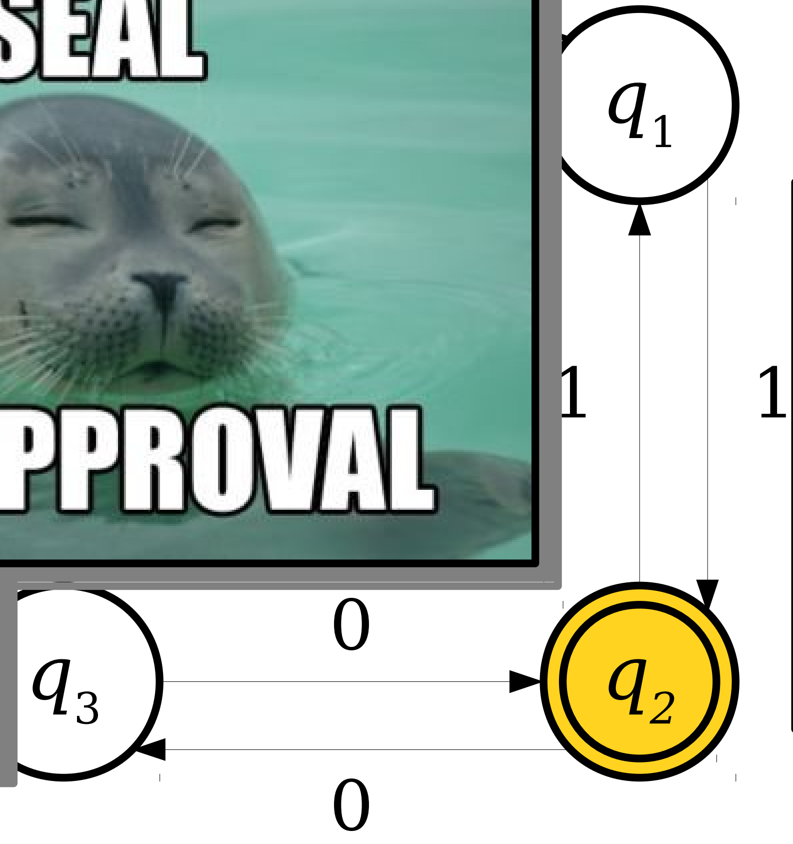
Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

The double circle indicates that this state is an **accepting state**, so the automaton outputs "yes."

0 1 0 1 1 0

A Simple Finite Automaton

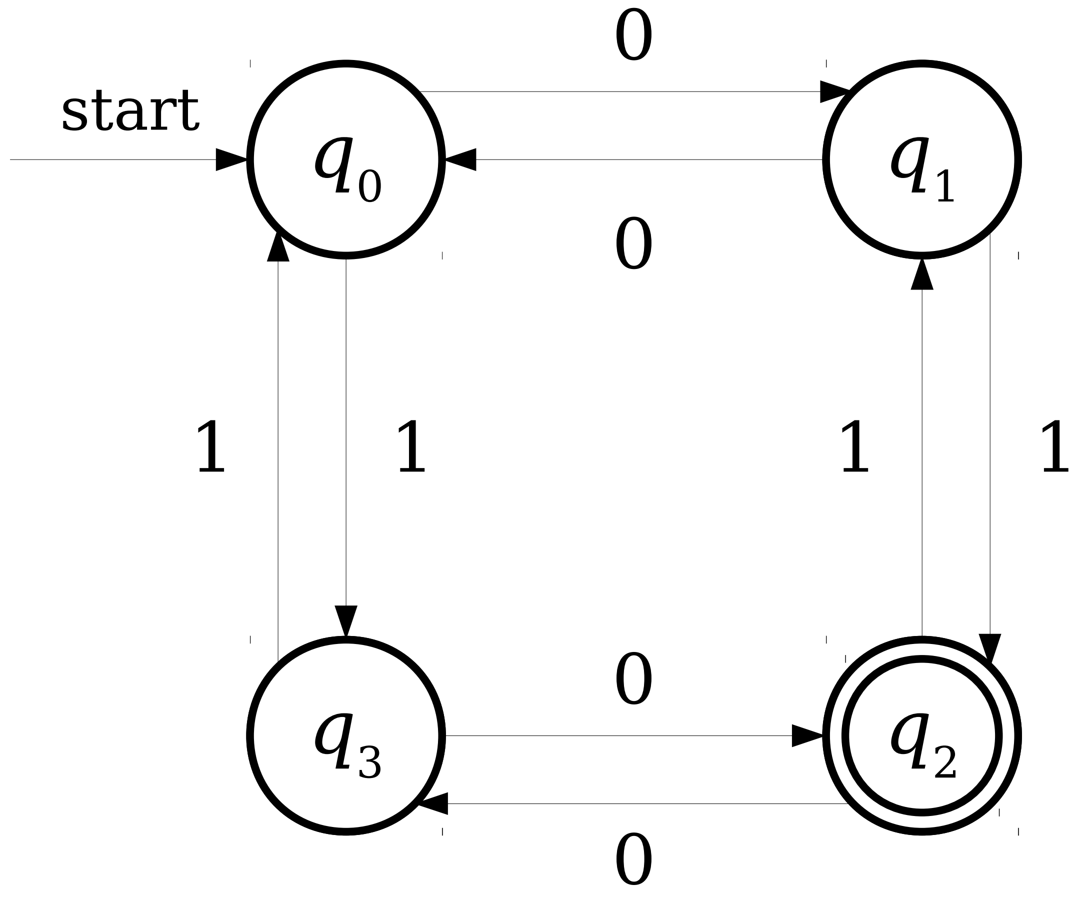
Now t
automa
looked
input, it
whether to say "yes"
or "no."



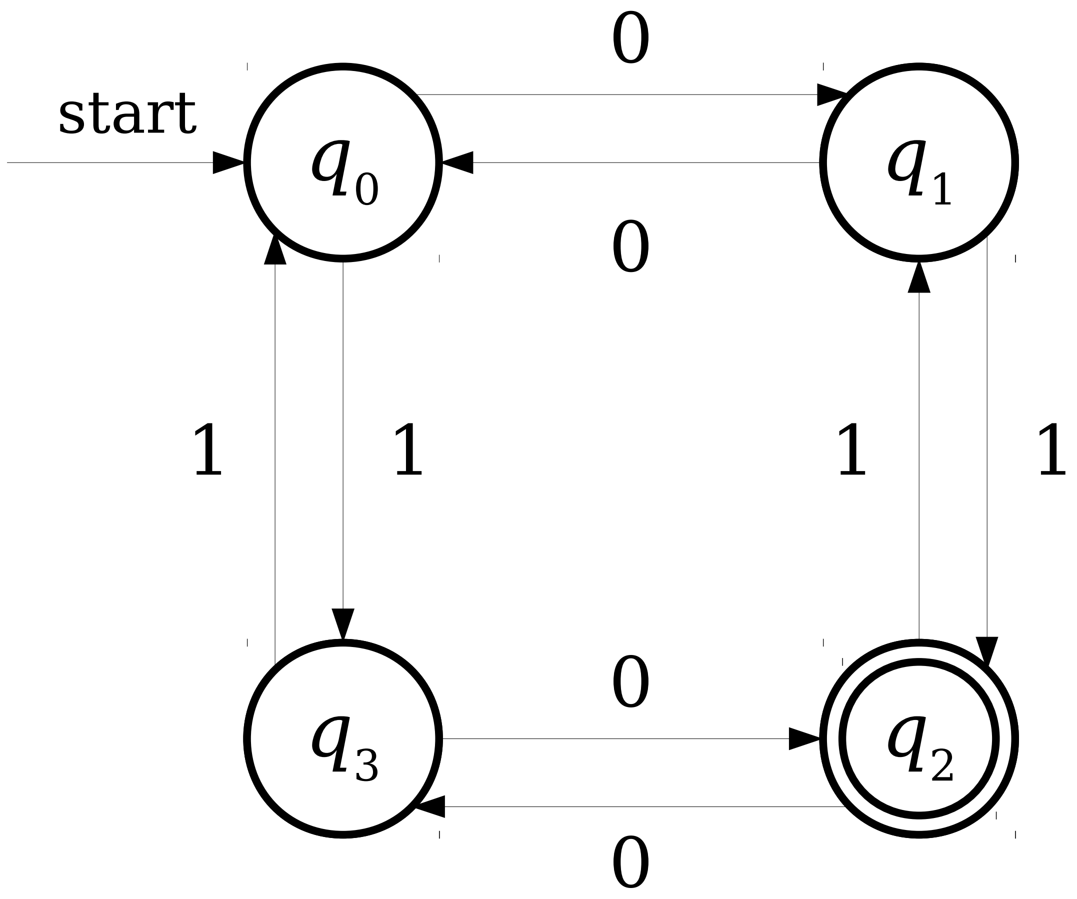
The double circle indicates that this state is an accepting state, so the automaton outputs "yes."

0 1 0 1 1 0

A Simple Finite Automaton

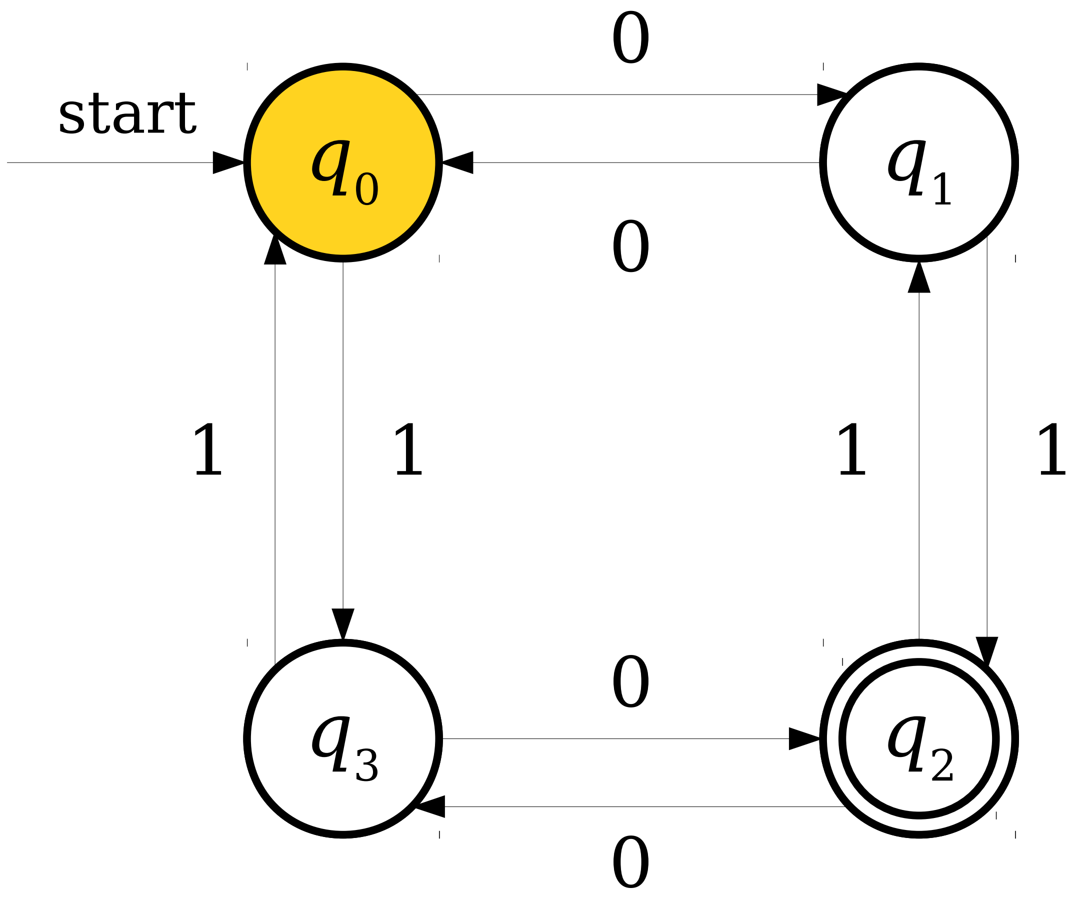


A Simple Finite Automaton



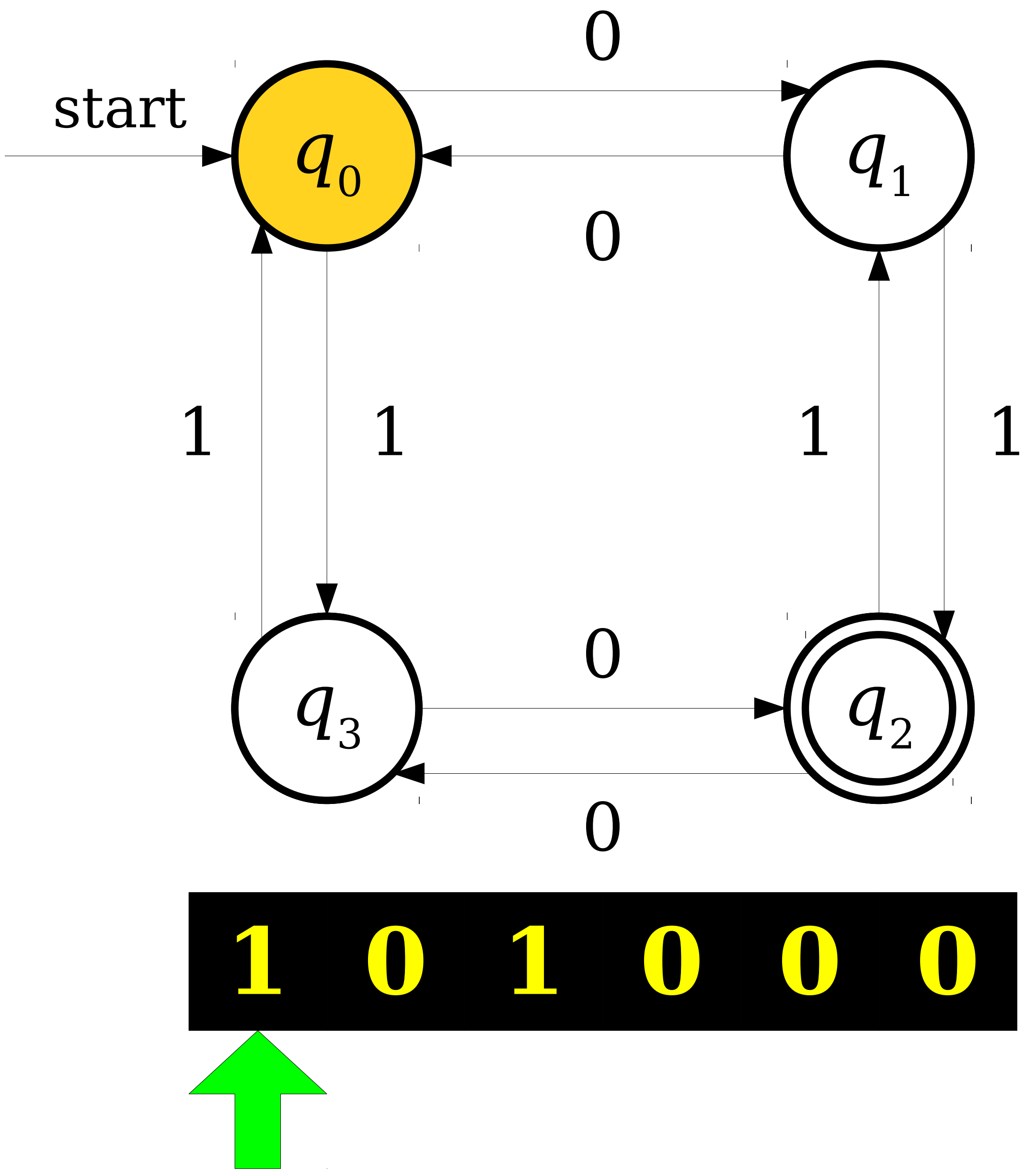
1 0 1 0 0 0

A Simple Finite Automaton



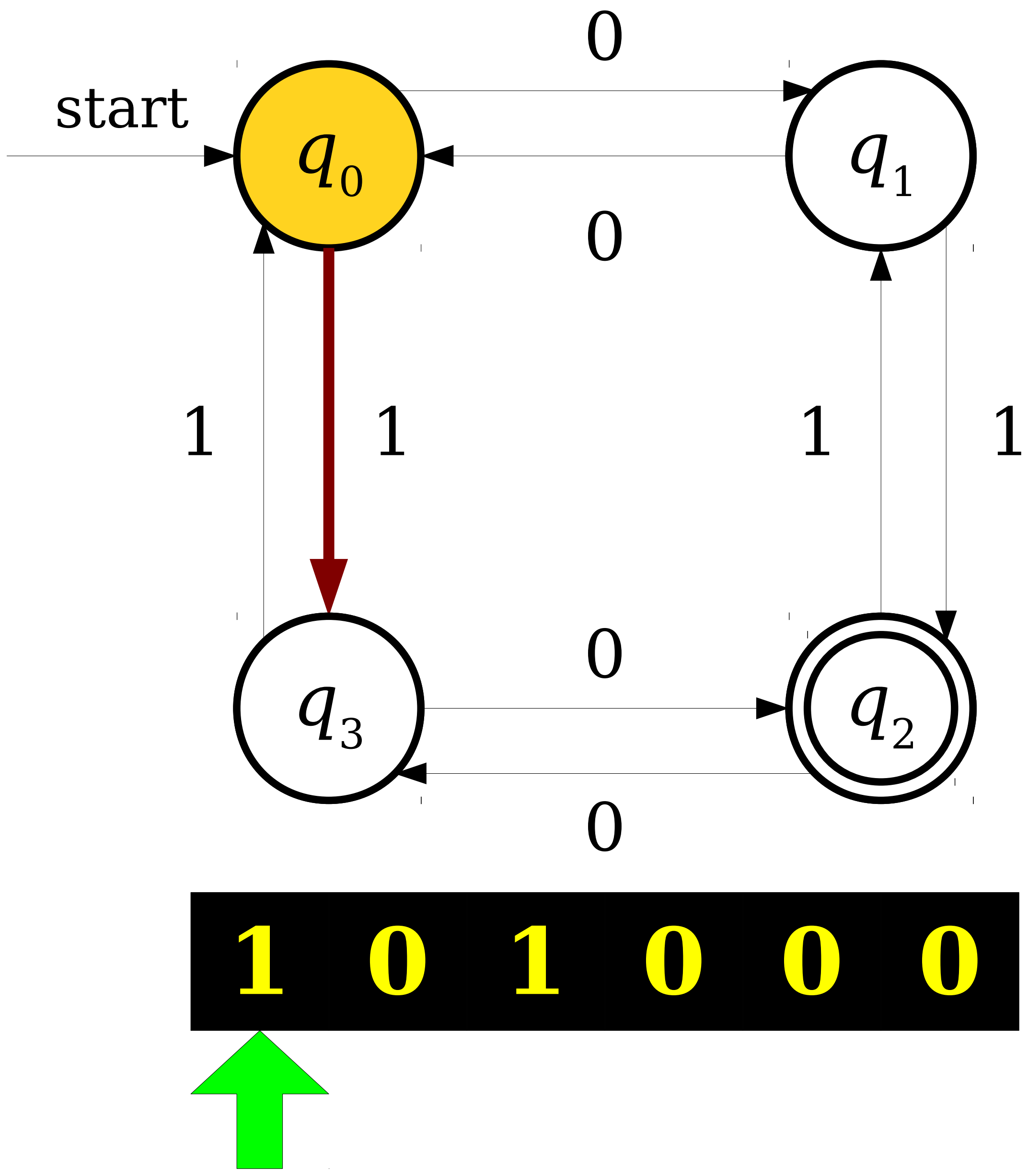
1 0 1 0 0 0

A Simple Finite Automaton

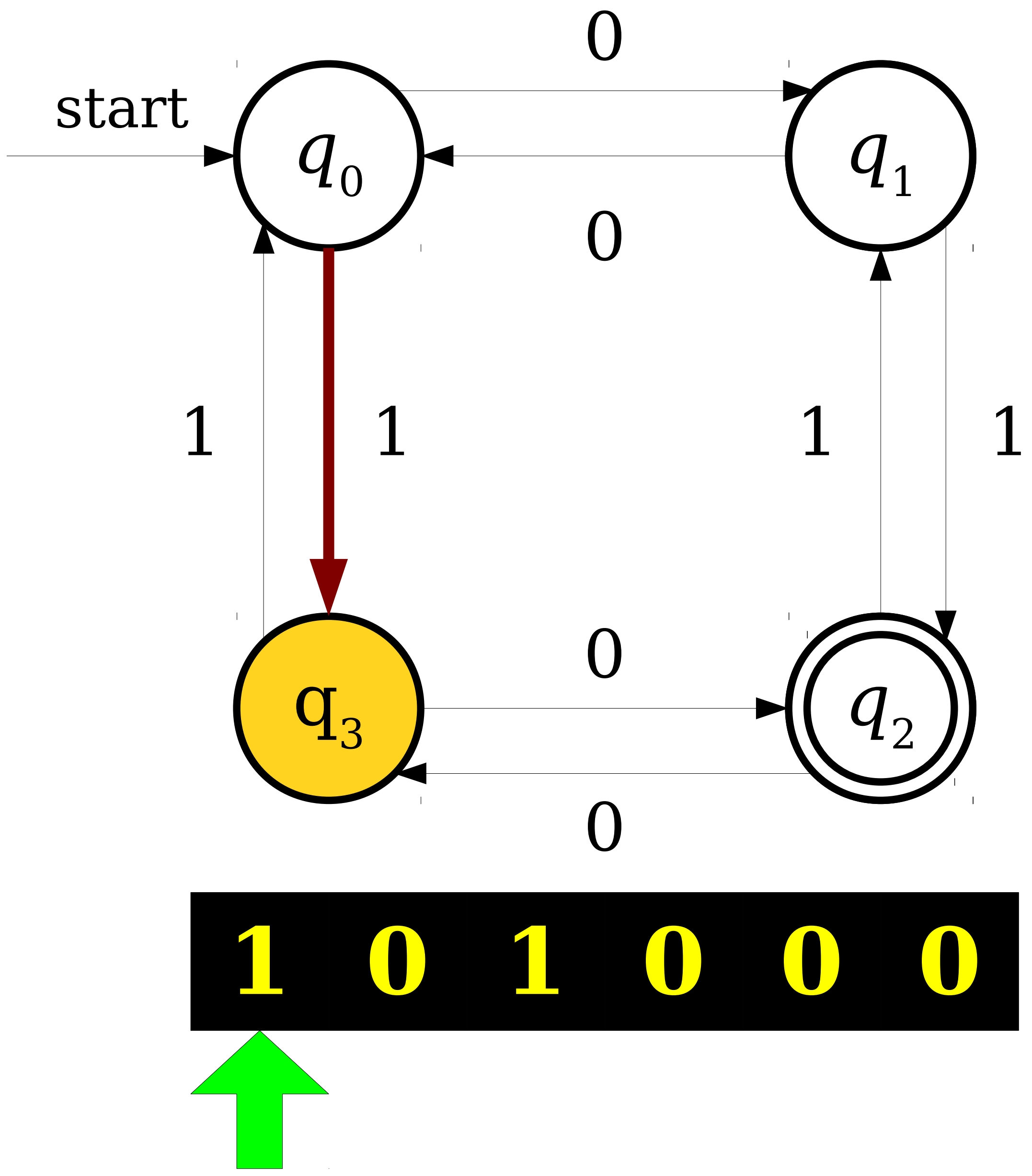


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[u, bend left, "1"] \arrow[l, bend left, "0"]
\end{tikzcd}$$

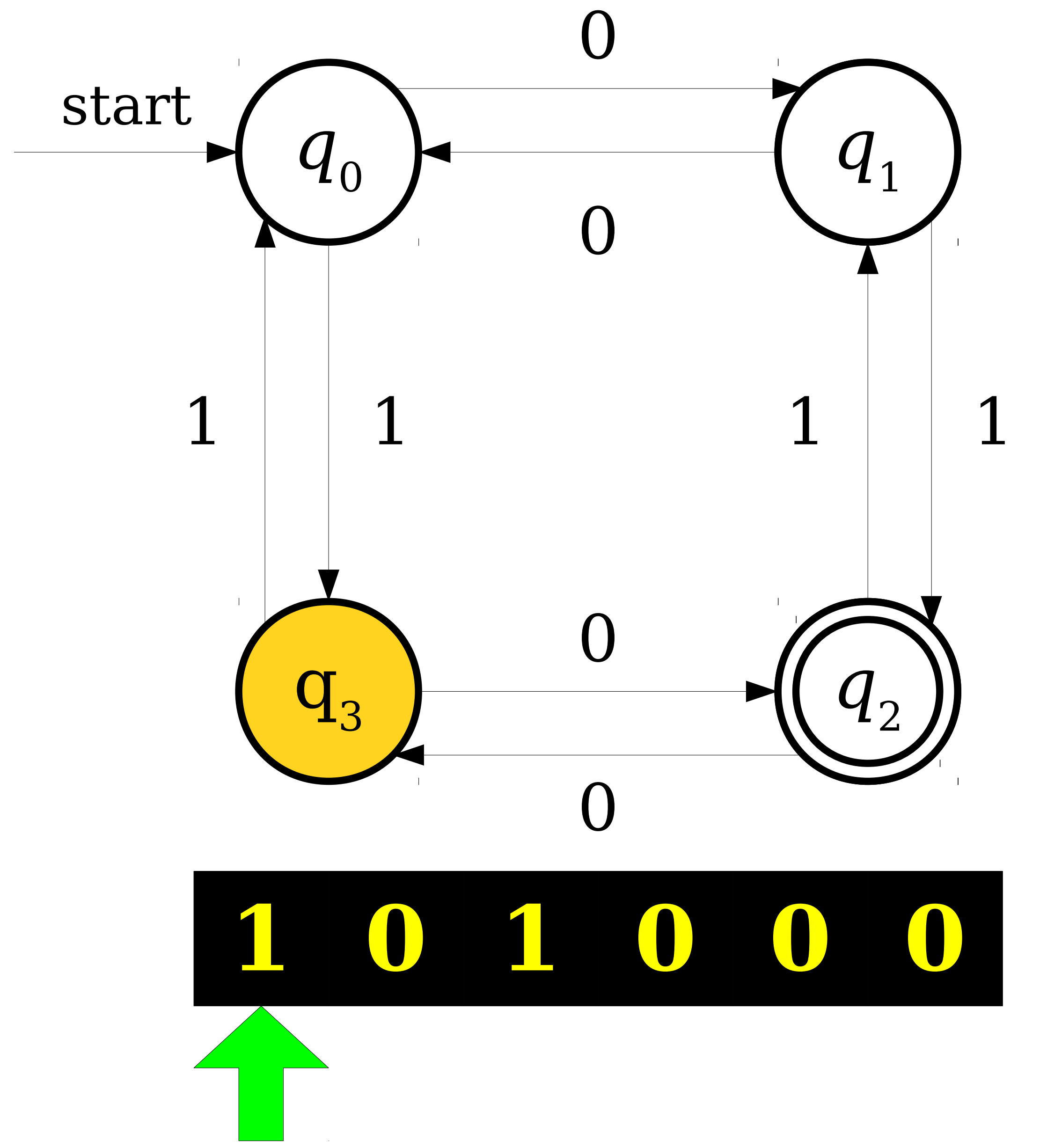
A Simple Finite Automaton



A Simple Finite Automaton

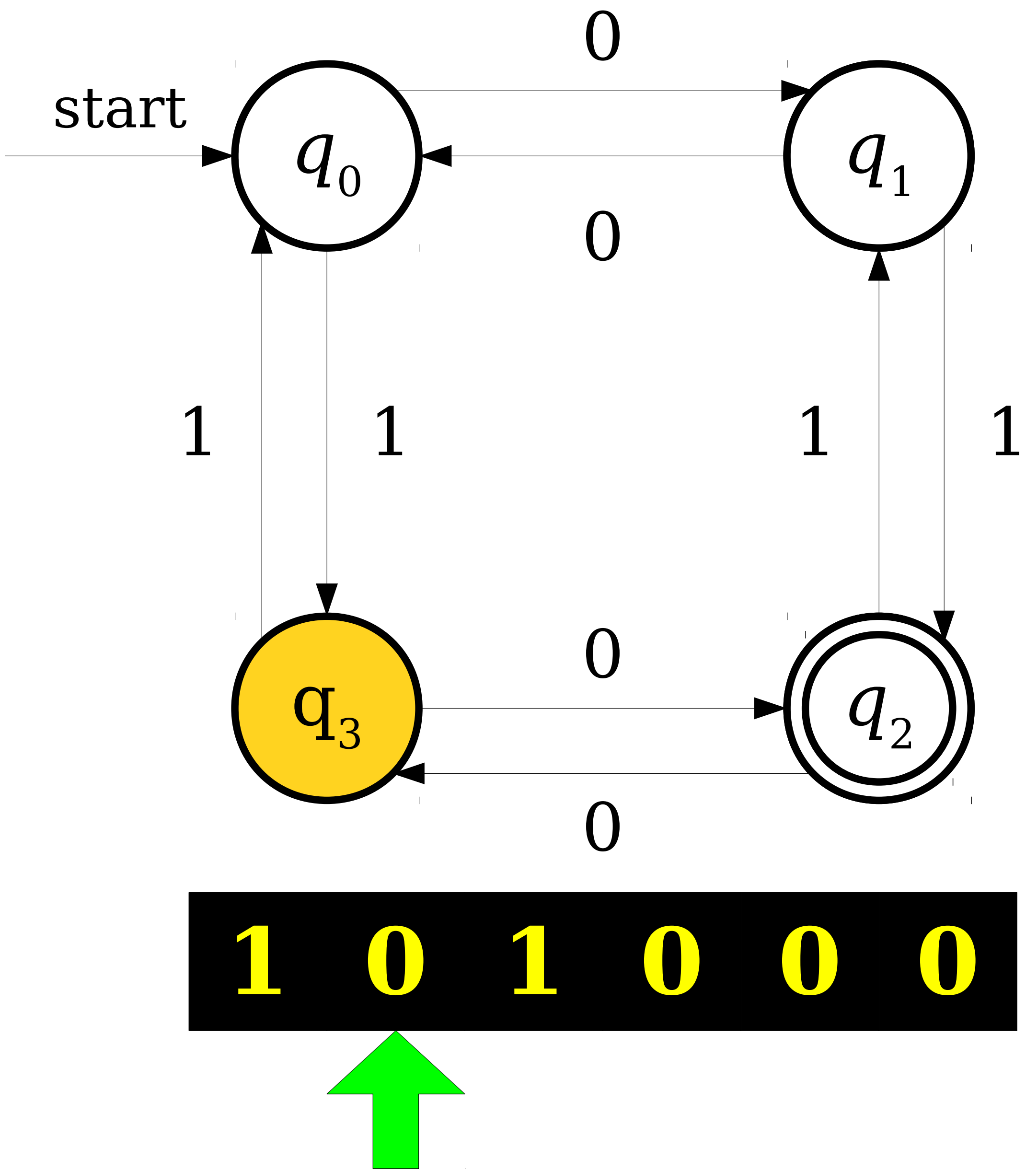


A Simple Finite Automaton

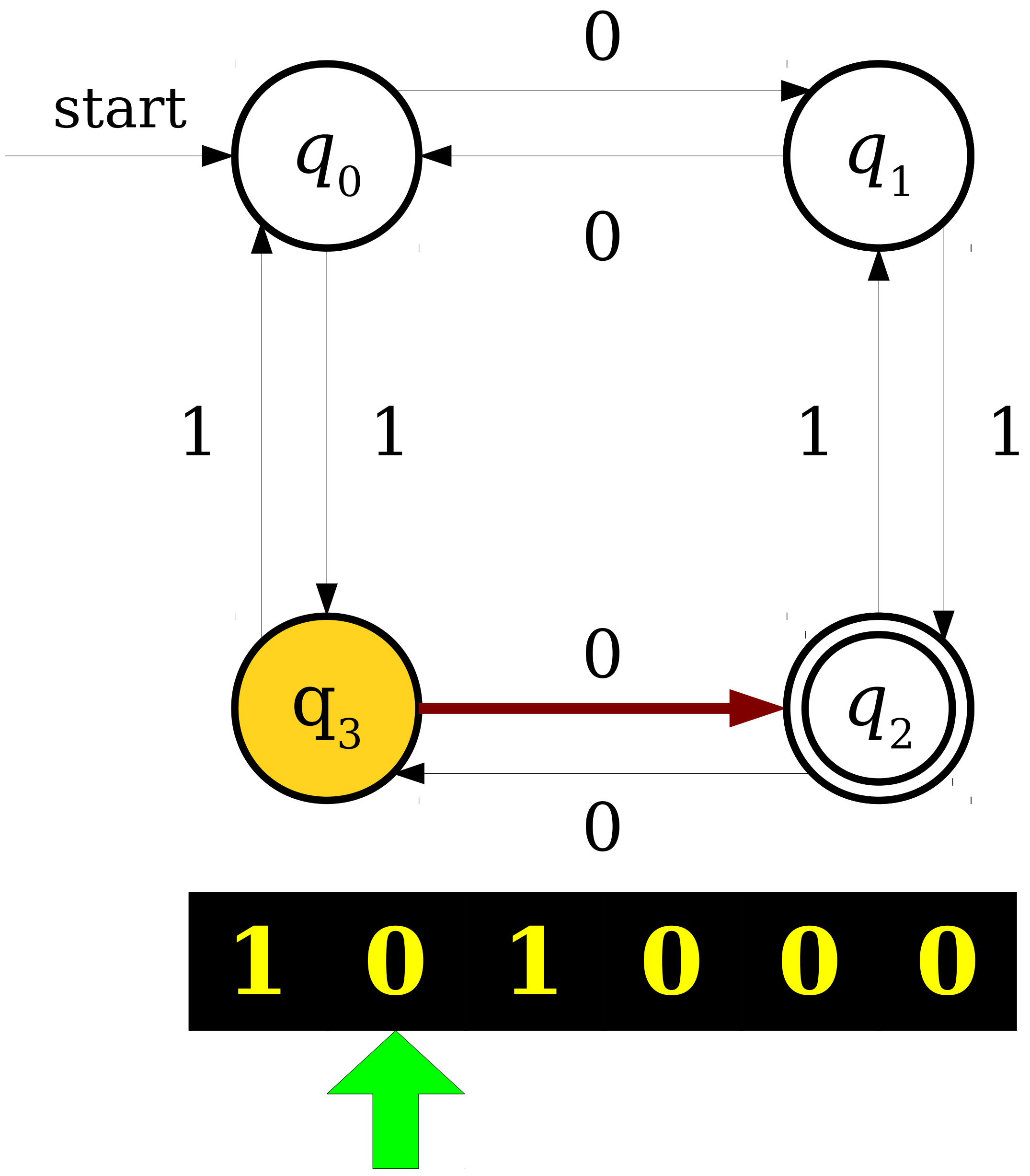


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

A Simple Finite Automaton

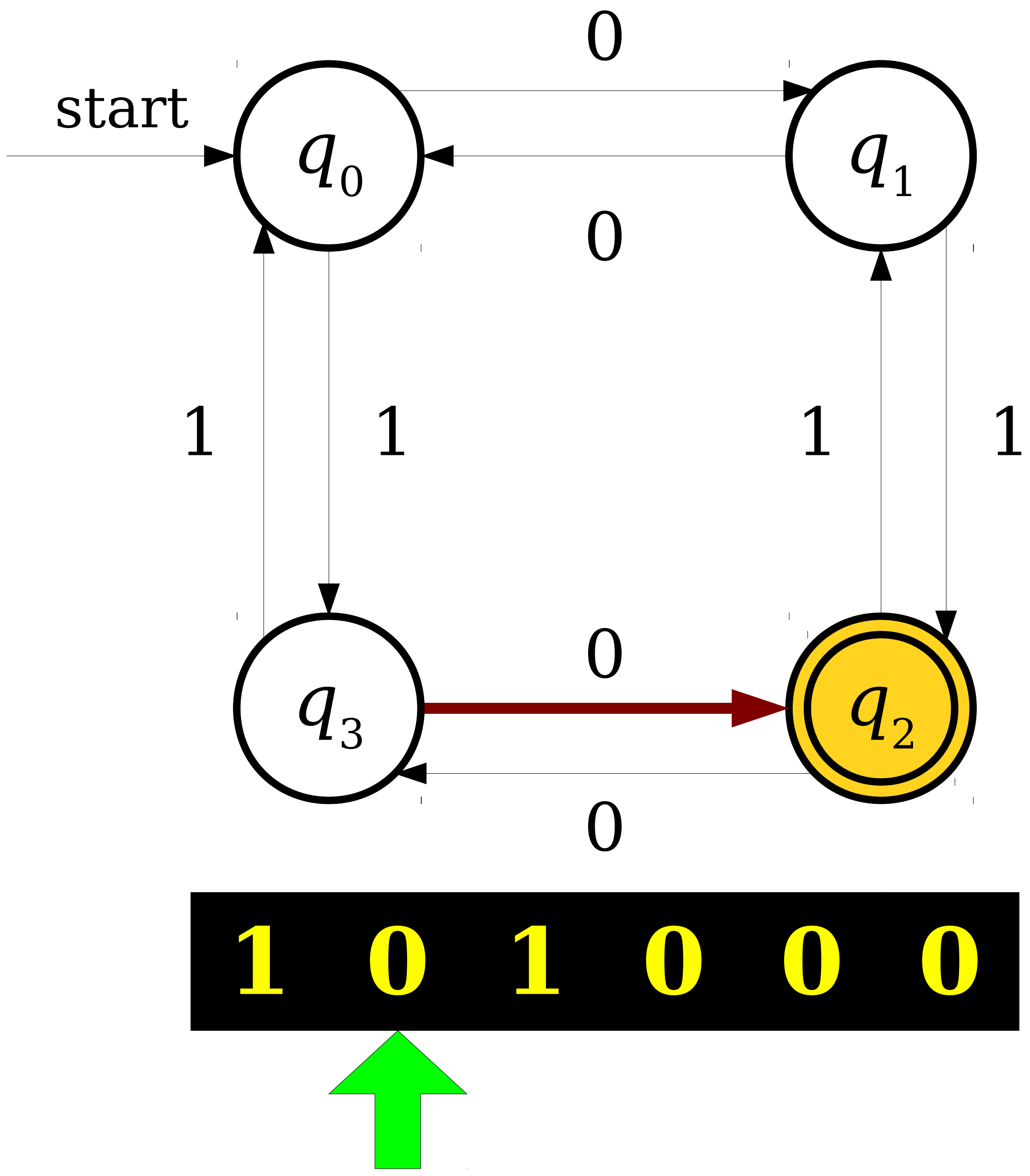


A Simple Finite Automaton

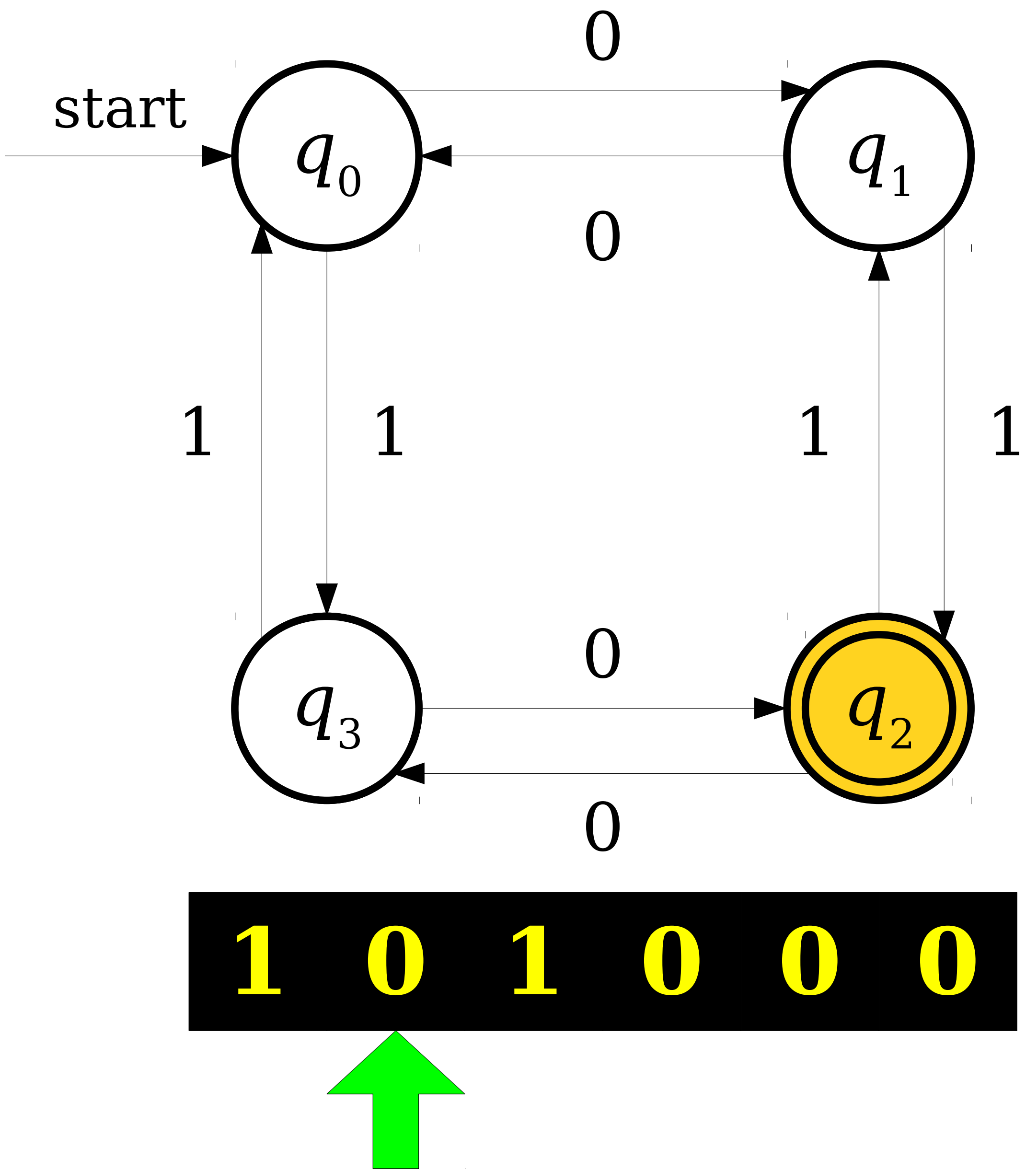


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

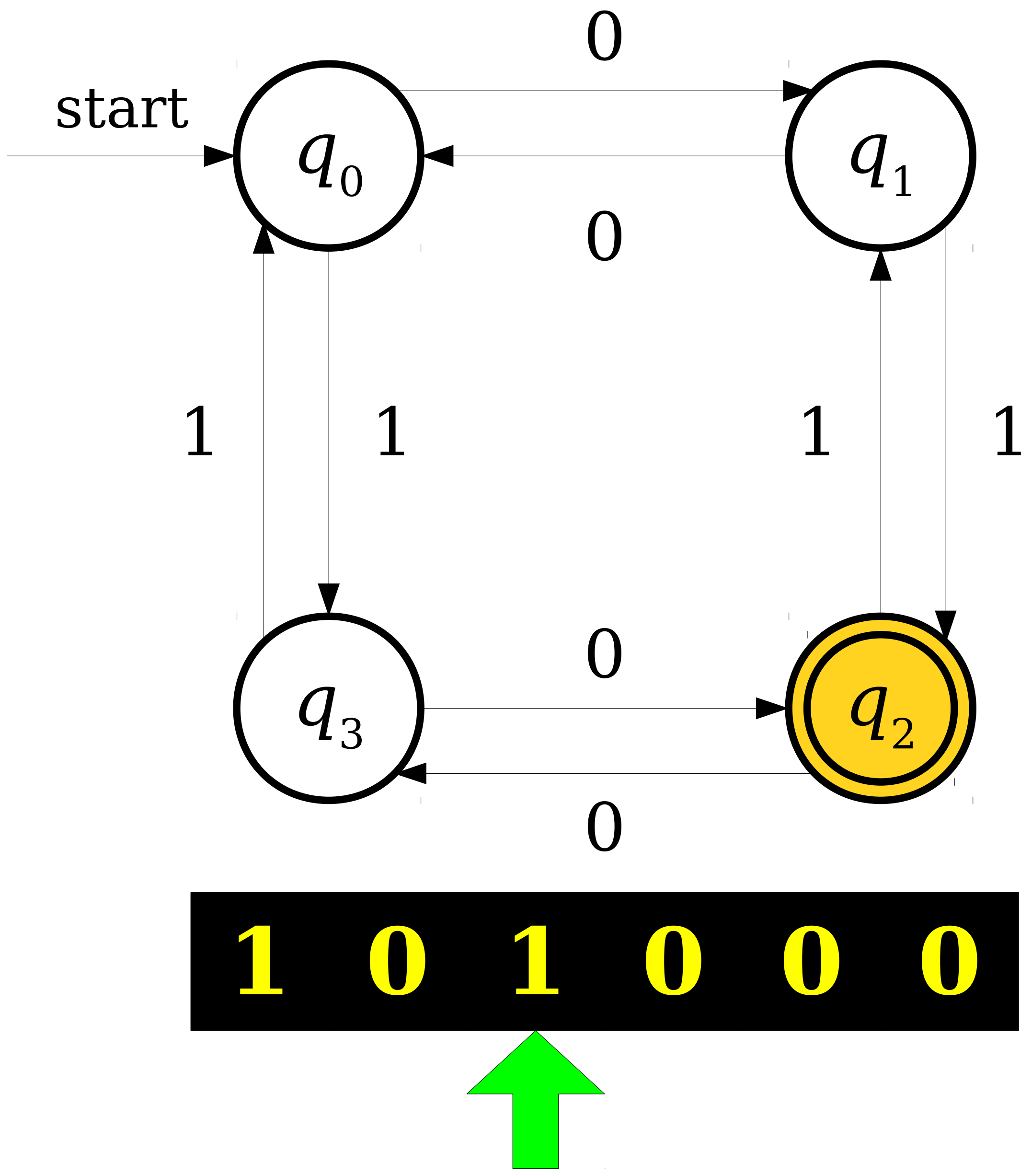
A Simple Finite Automaton



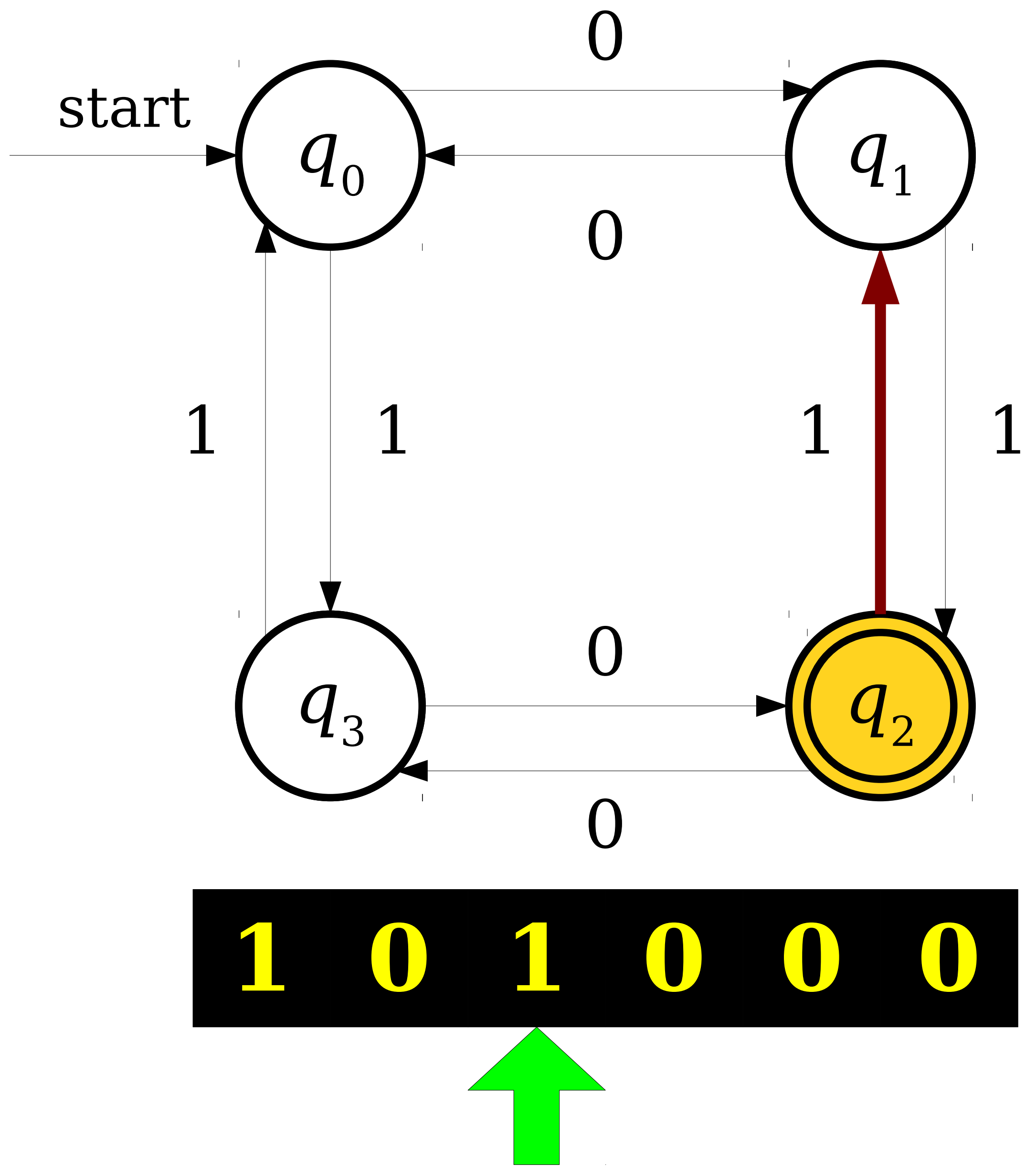
A Simple Finite Automaton



A Simple Finite Automaton

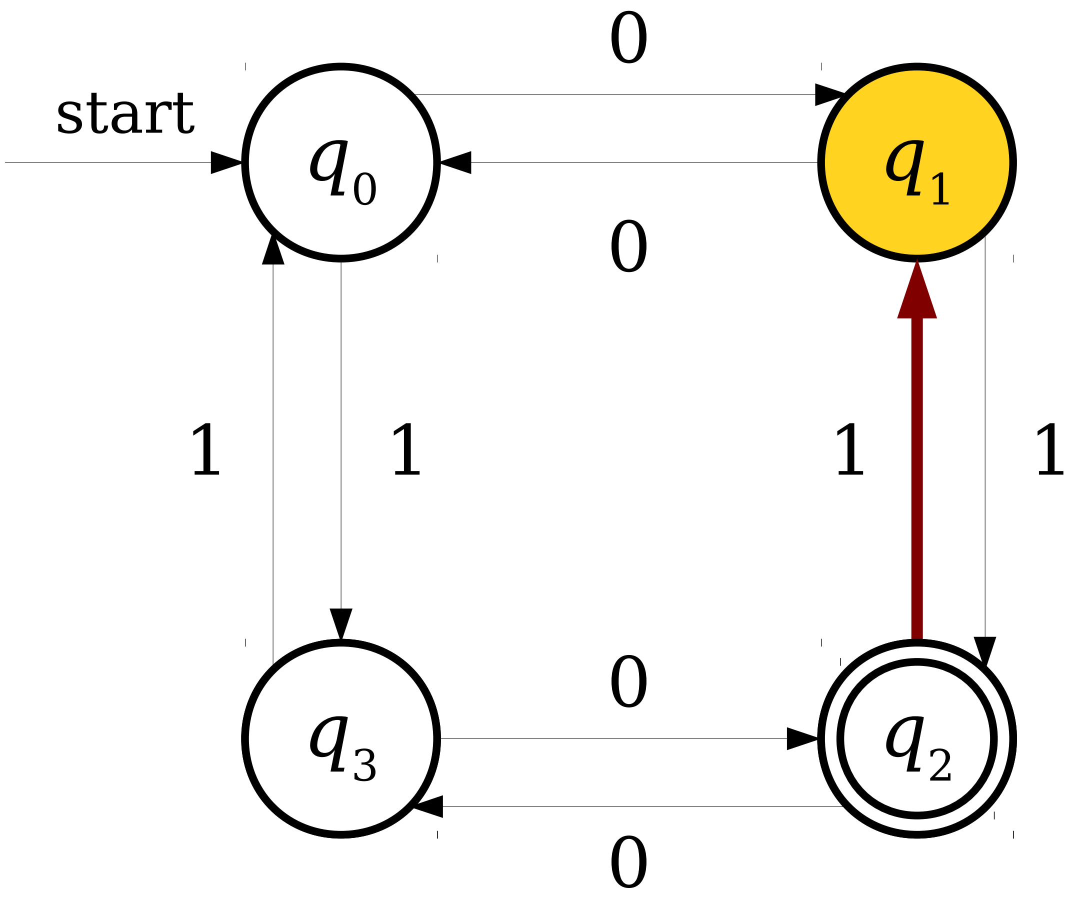


A Simple Finite Automaton

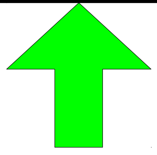


A Simple Finite Automaton

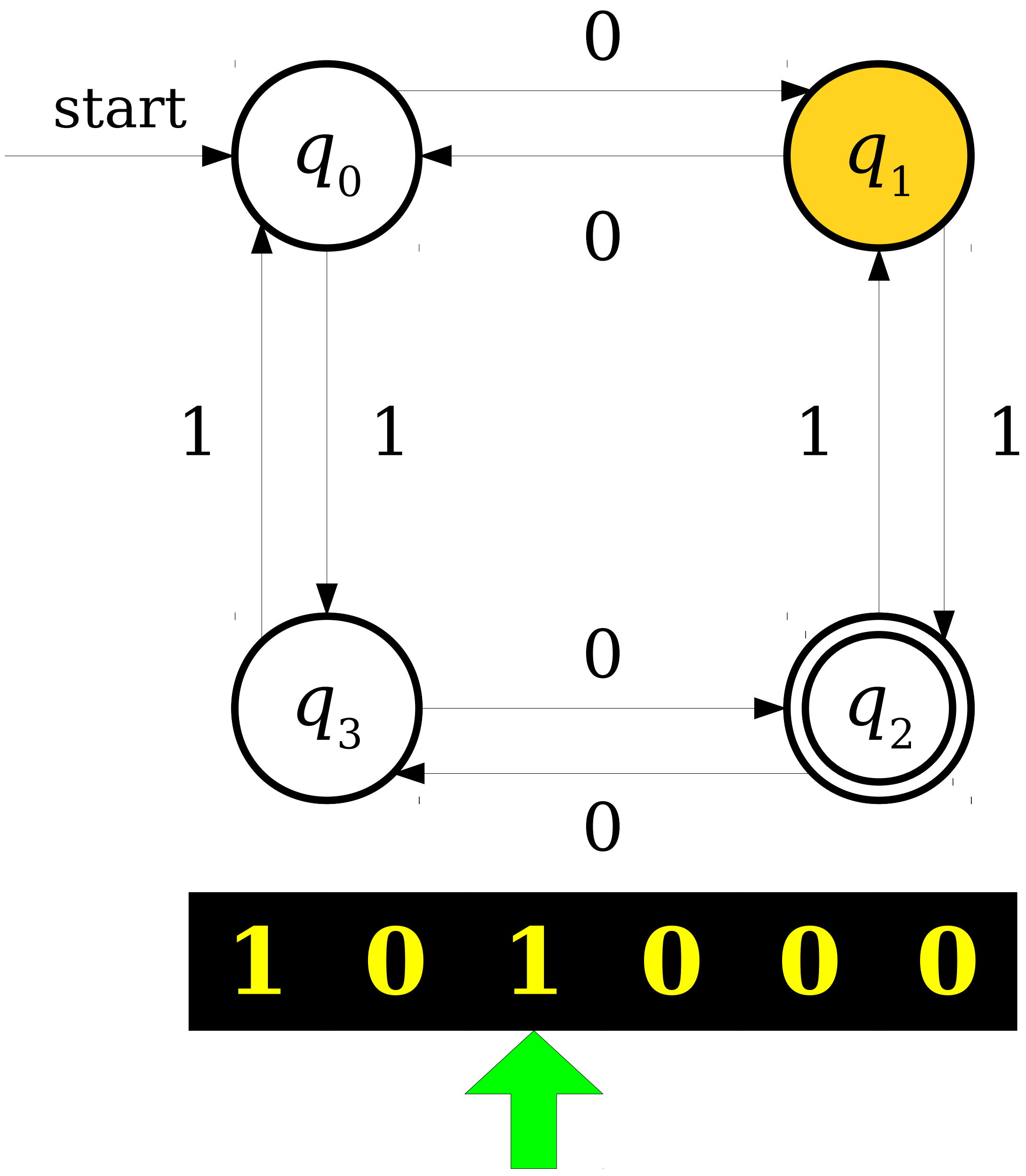
$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, shift left, "0"] \arrow[d, shift left, "1"] & q_1 \arrow[l, shift left, "0"] \arrow[d, shift left, "1"] \\
& q_3 \arrow[u, shift left, "1"] \arrow[r, shift left, "0"] & q_2 \arrow[l, shift left, "0"] \arrow[u, shift left, "1"]
\end{tikzcd}$$



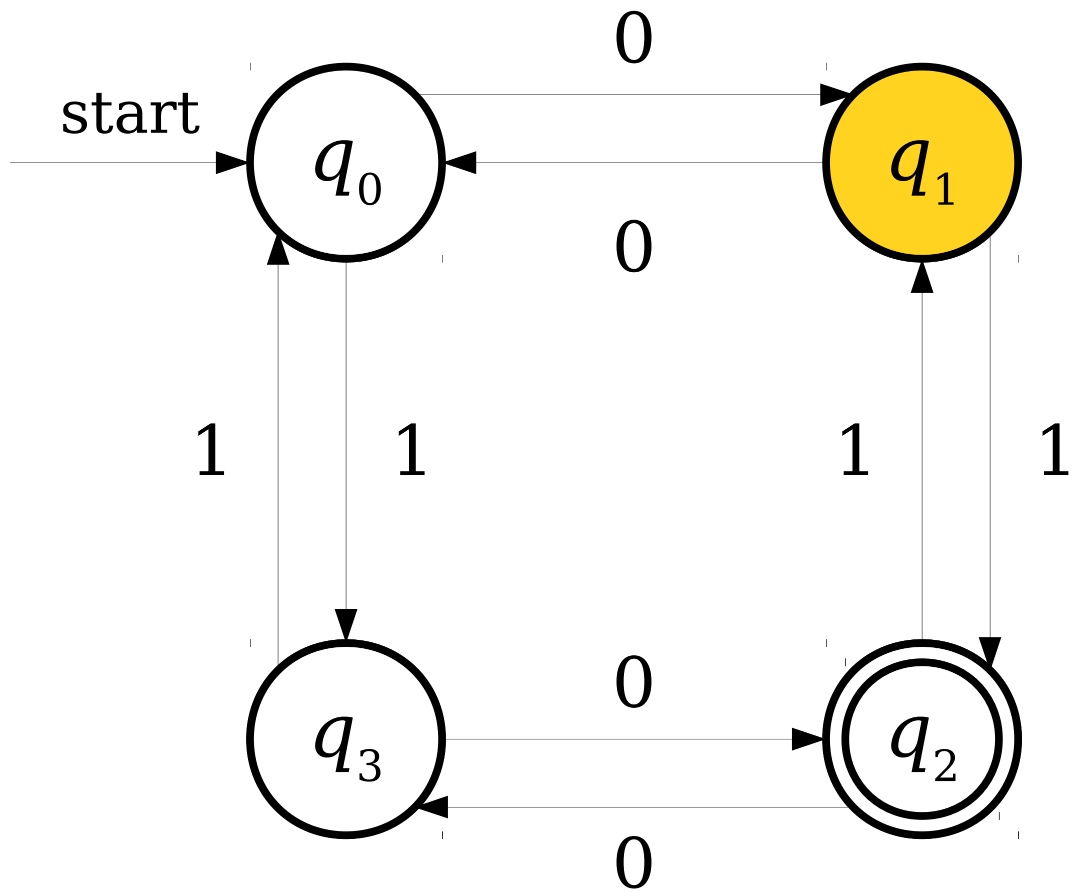
1 0 1 0 0 0

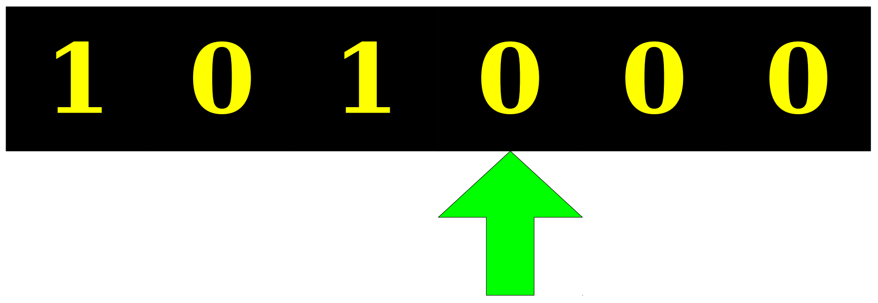


A Simple Finite Automaton

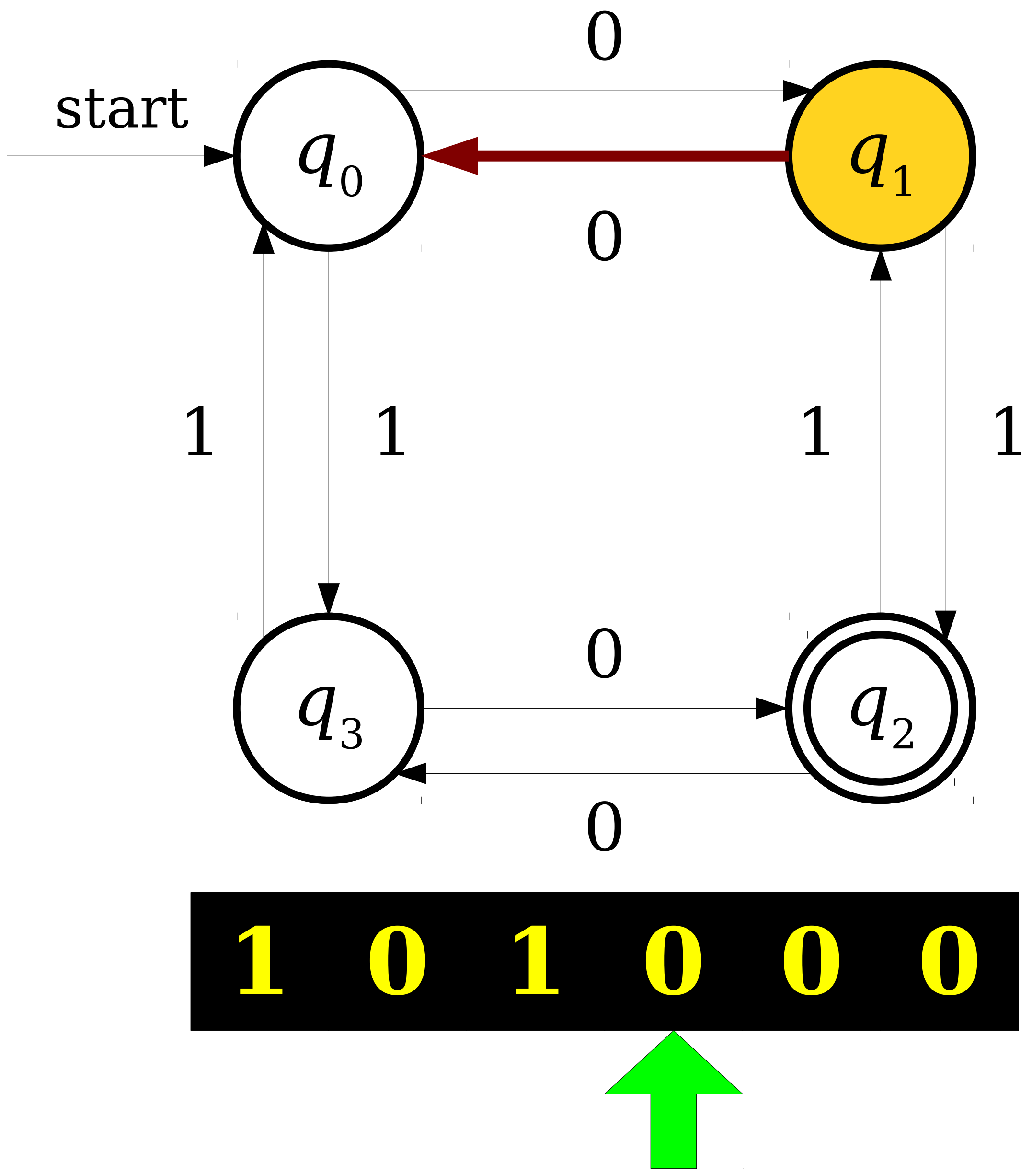


A Simple Finite Automaton

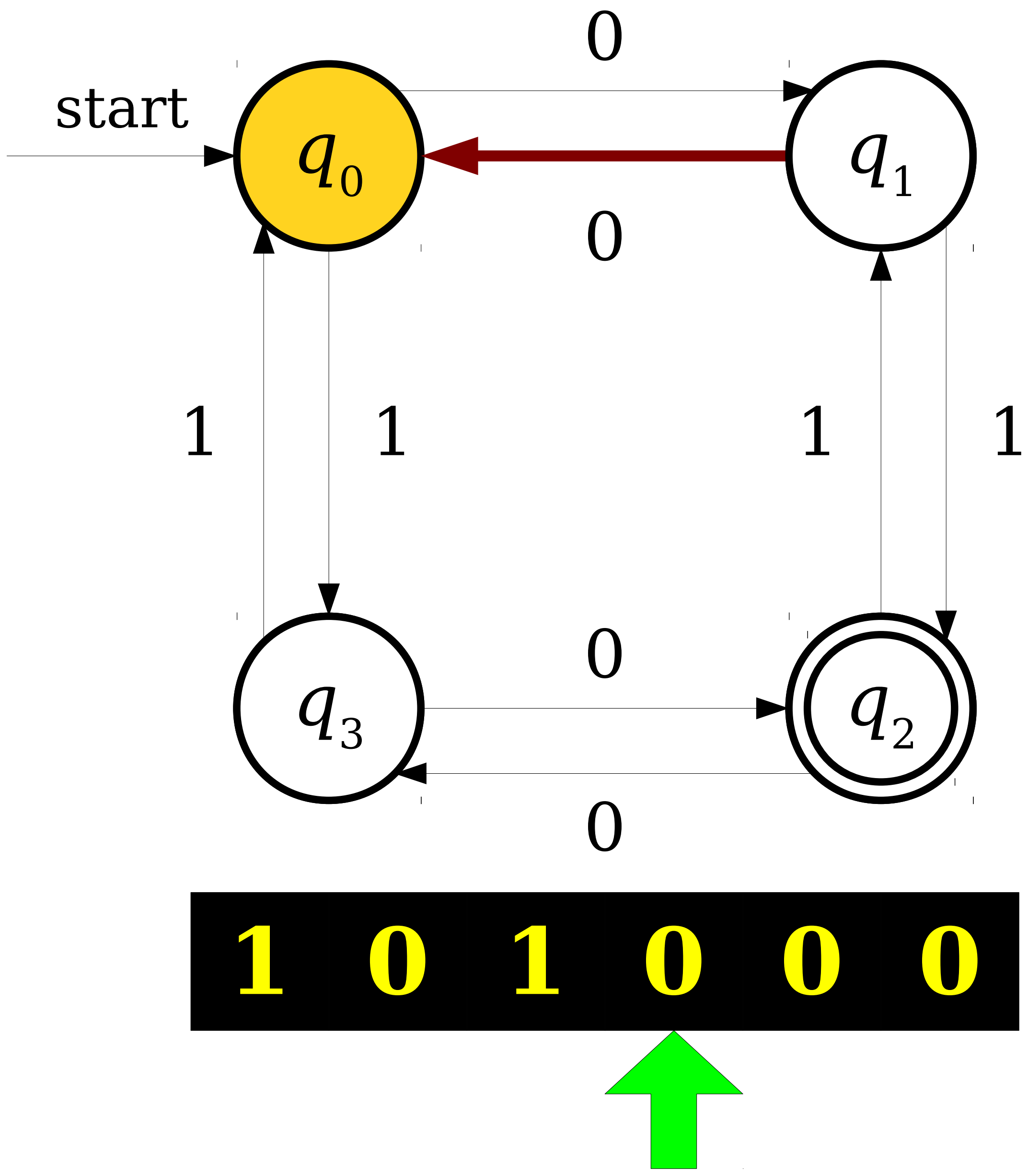




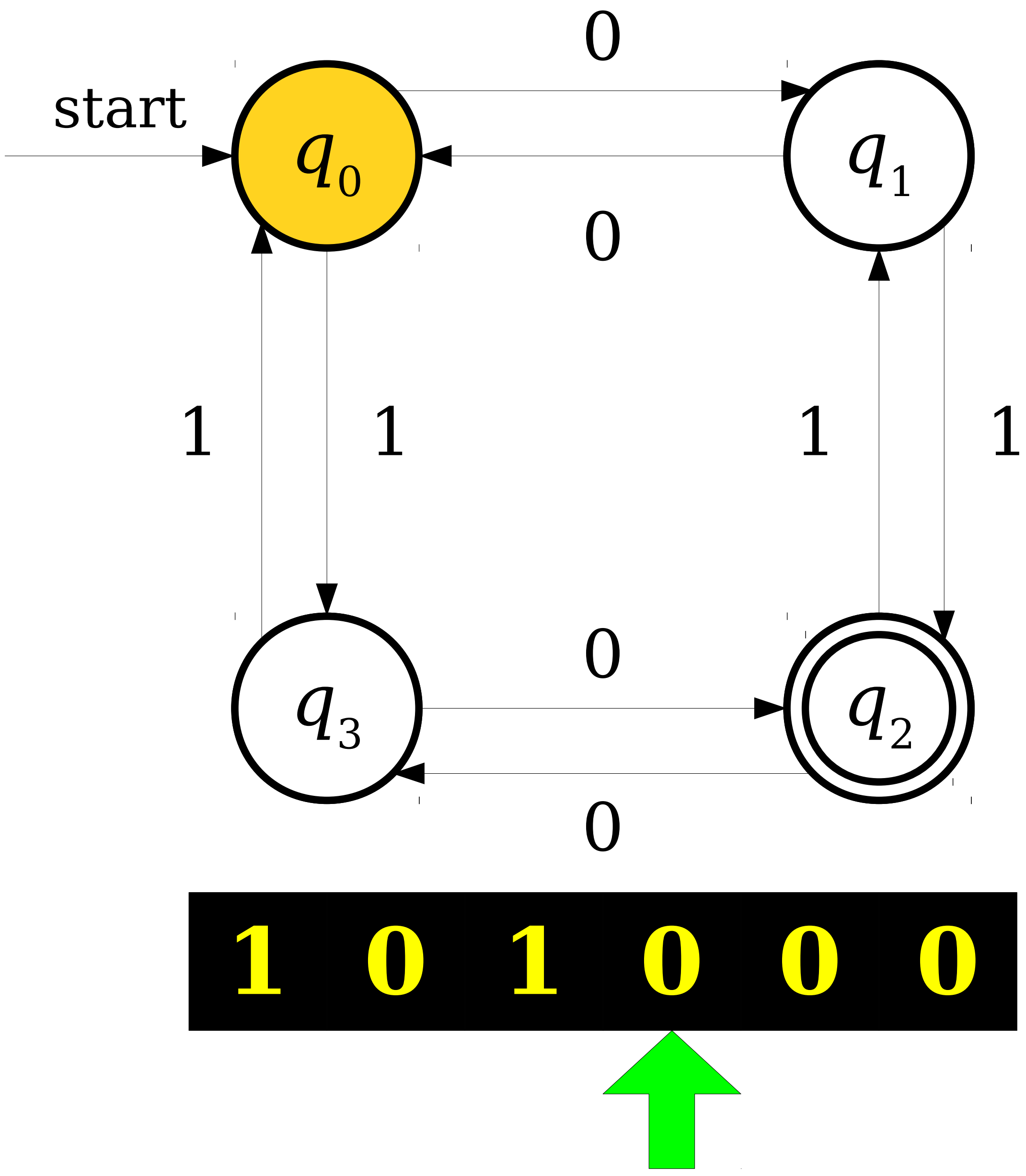
A Simple Finite Automaton



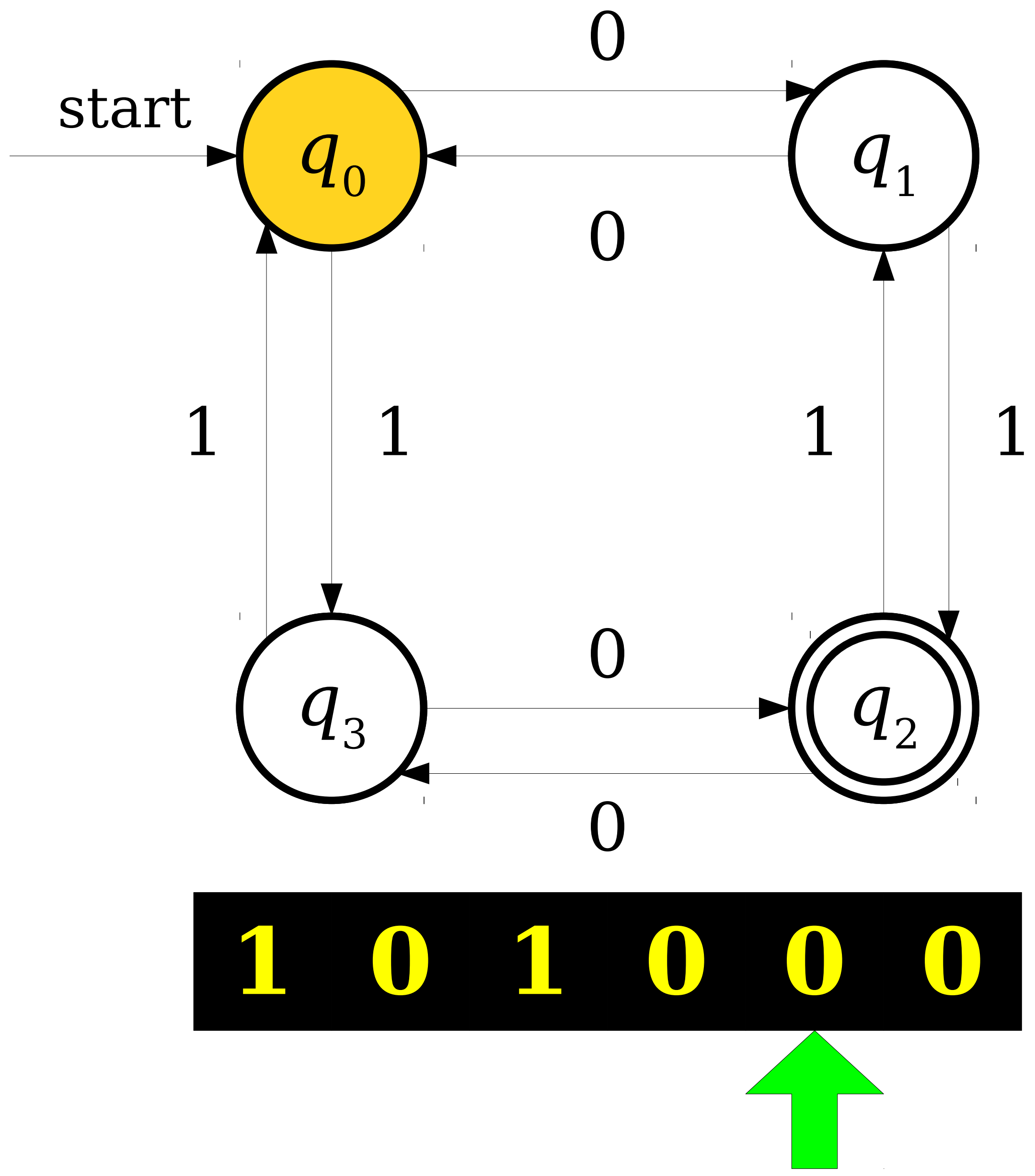
A Simple Finite Automaton



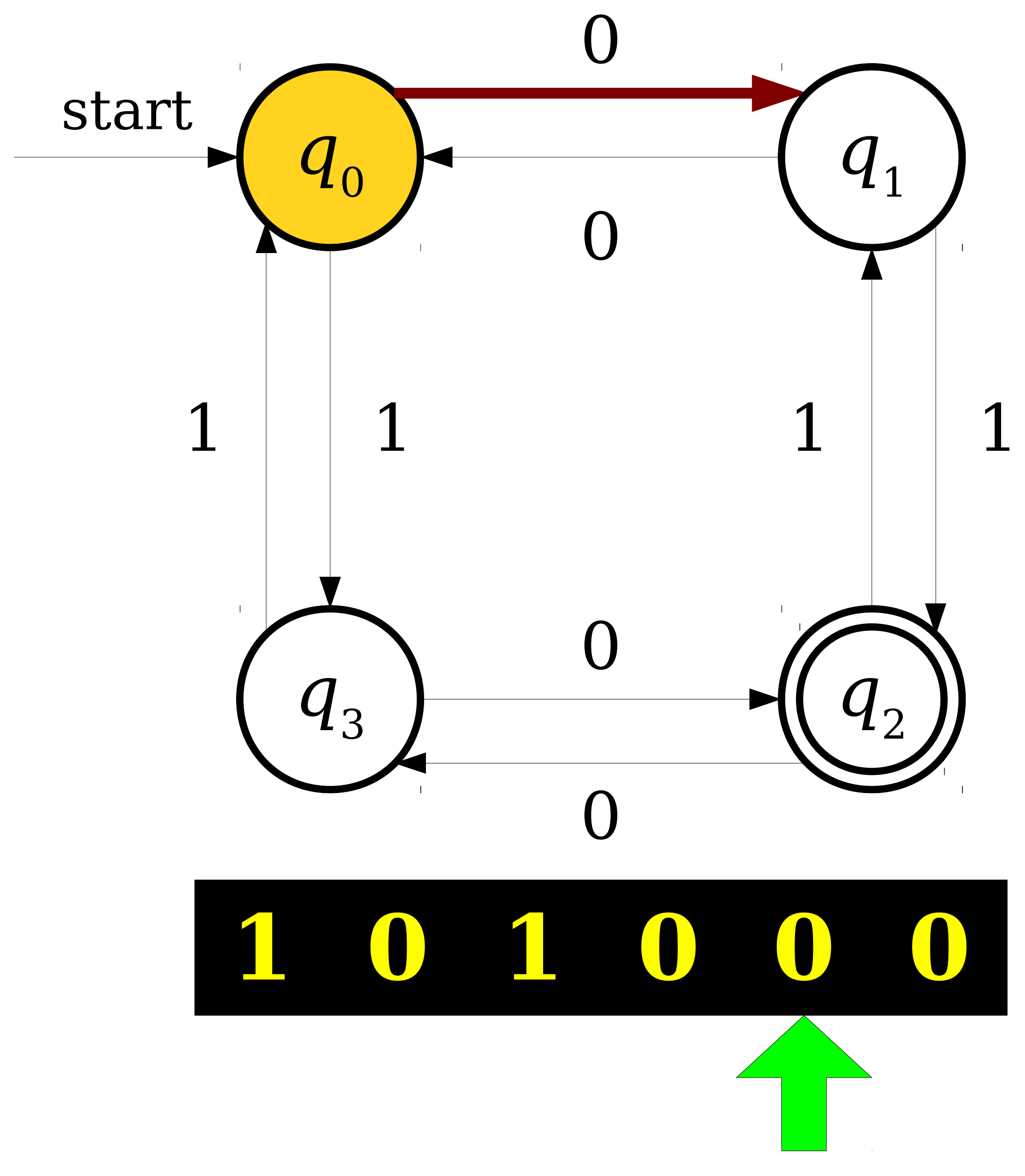
A Simple Finite Automaton



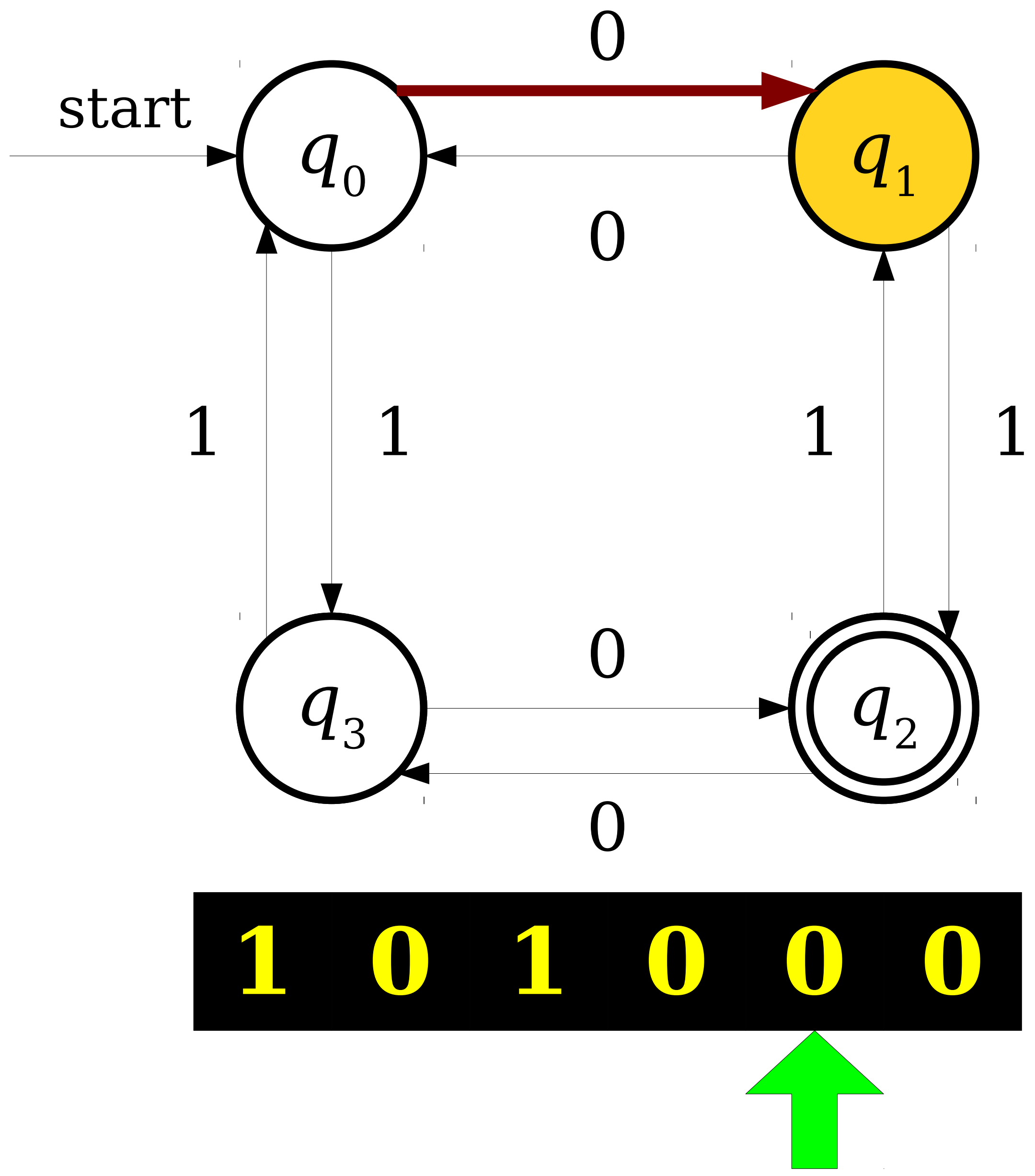
A Simple Finite Automaton



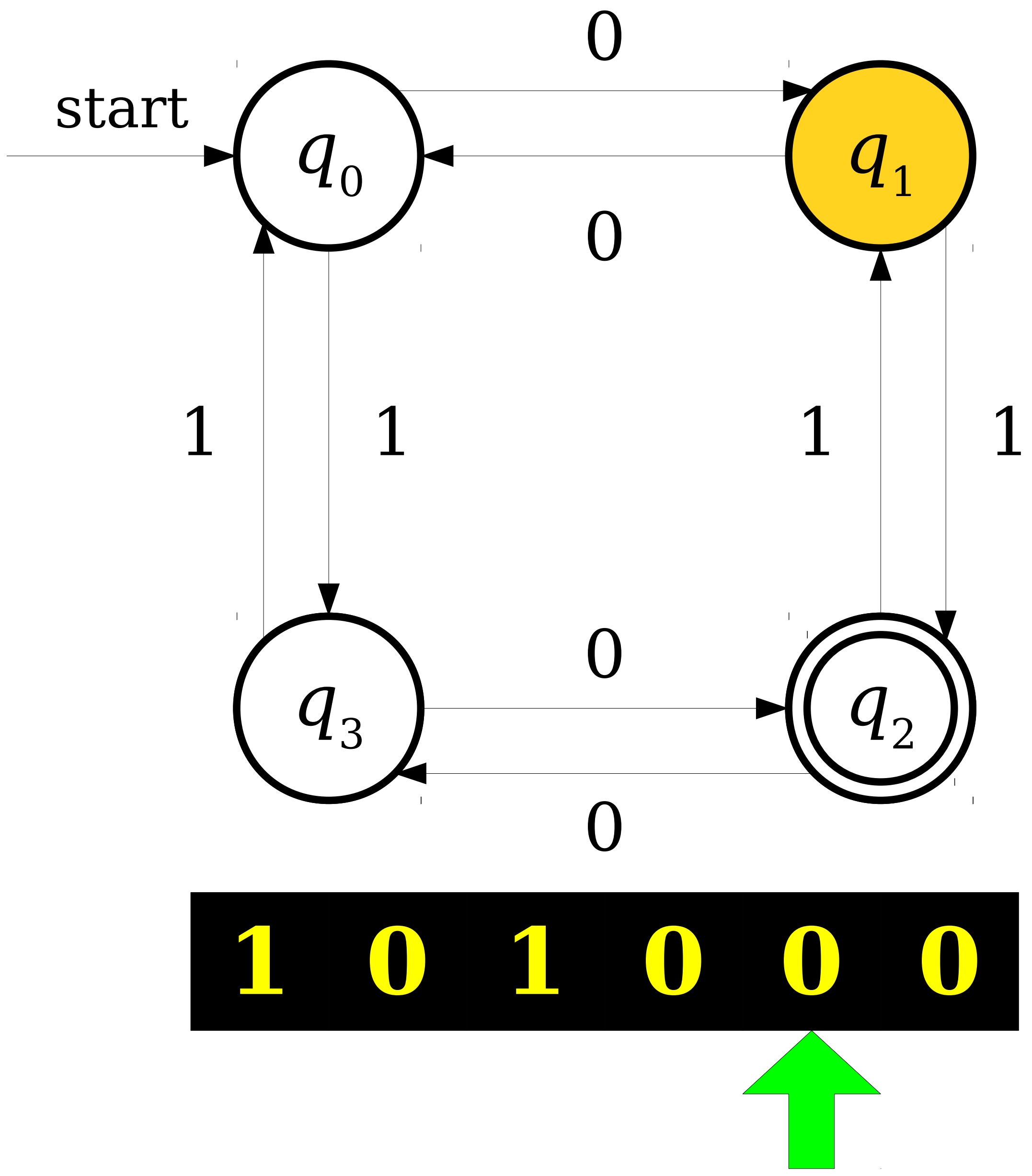
A Simple Finite Automaton



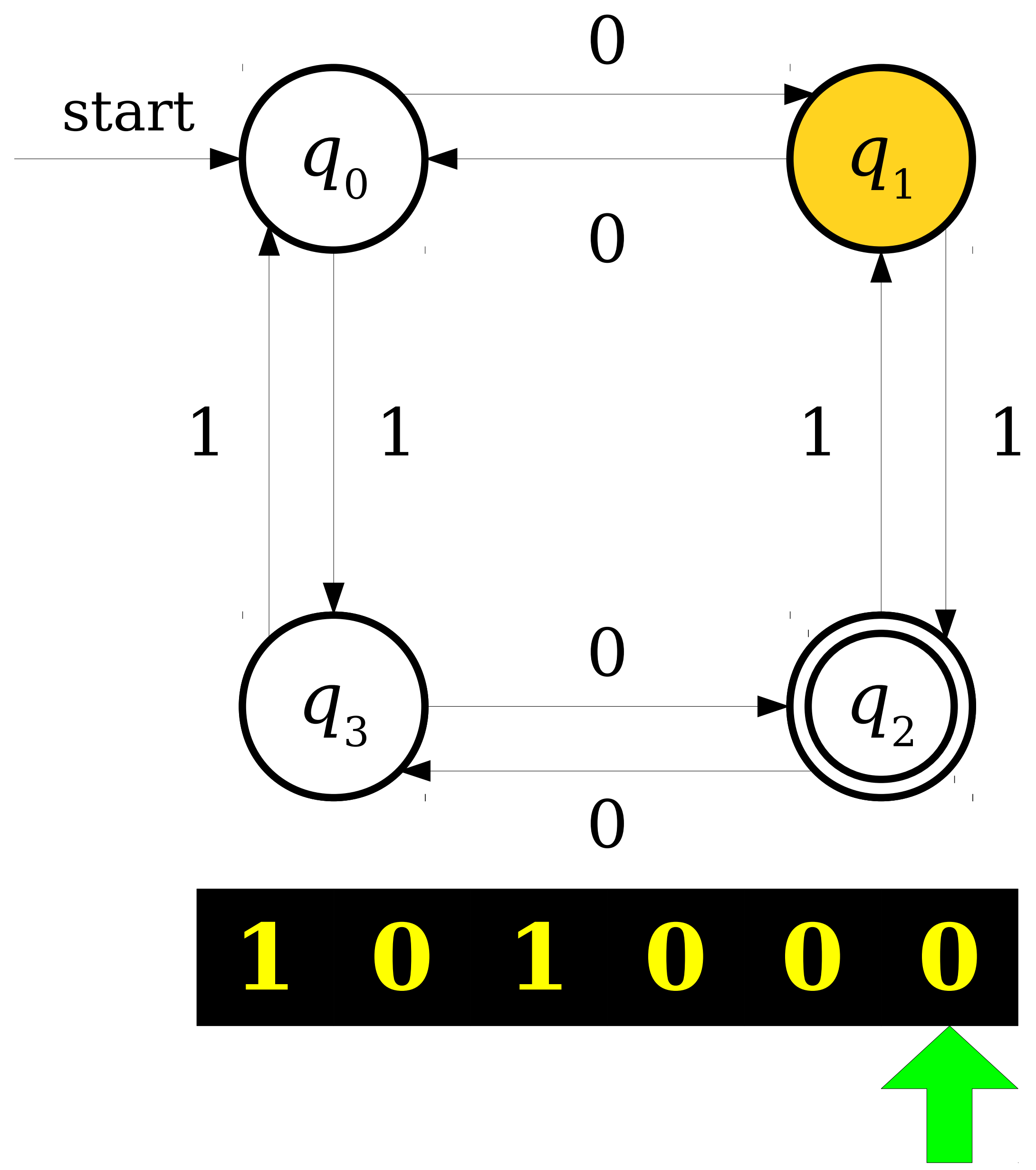
A Simple Finite Automaton



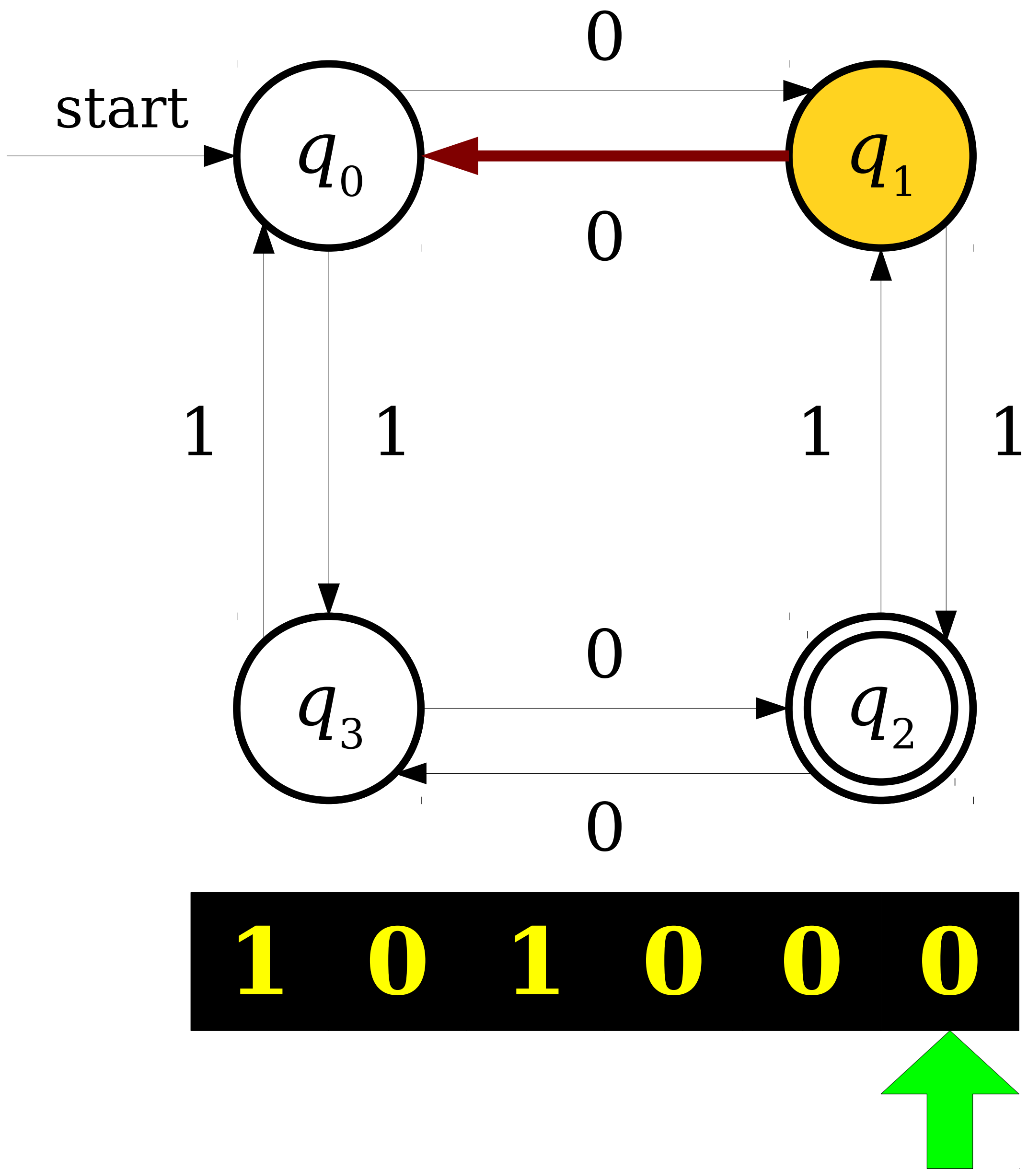
A Simple Finite Automaton



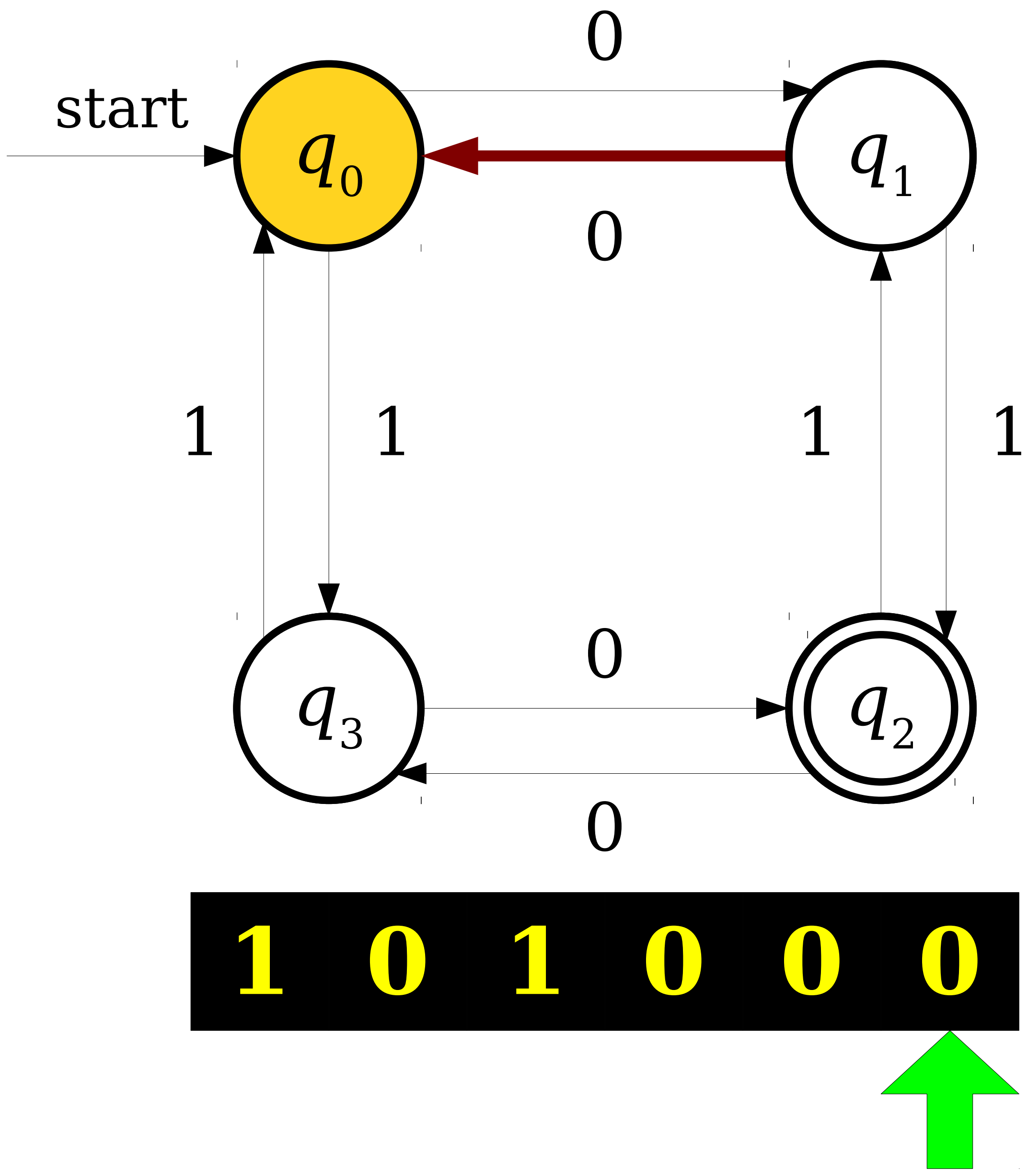
A Simple Finite Automaton



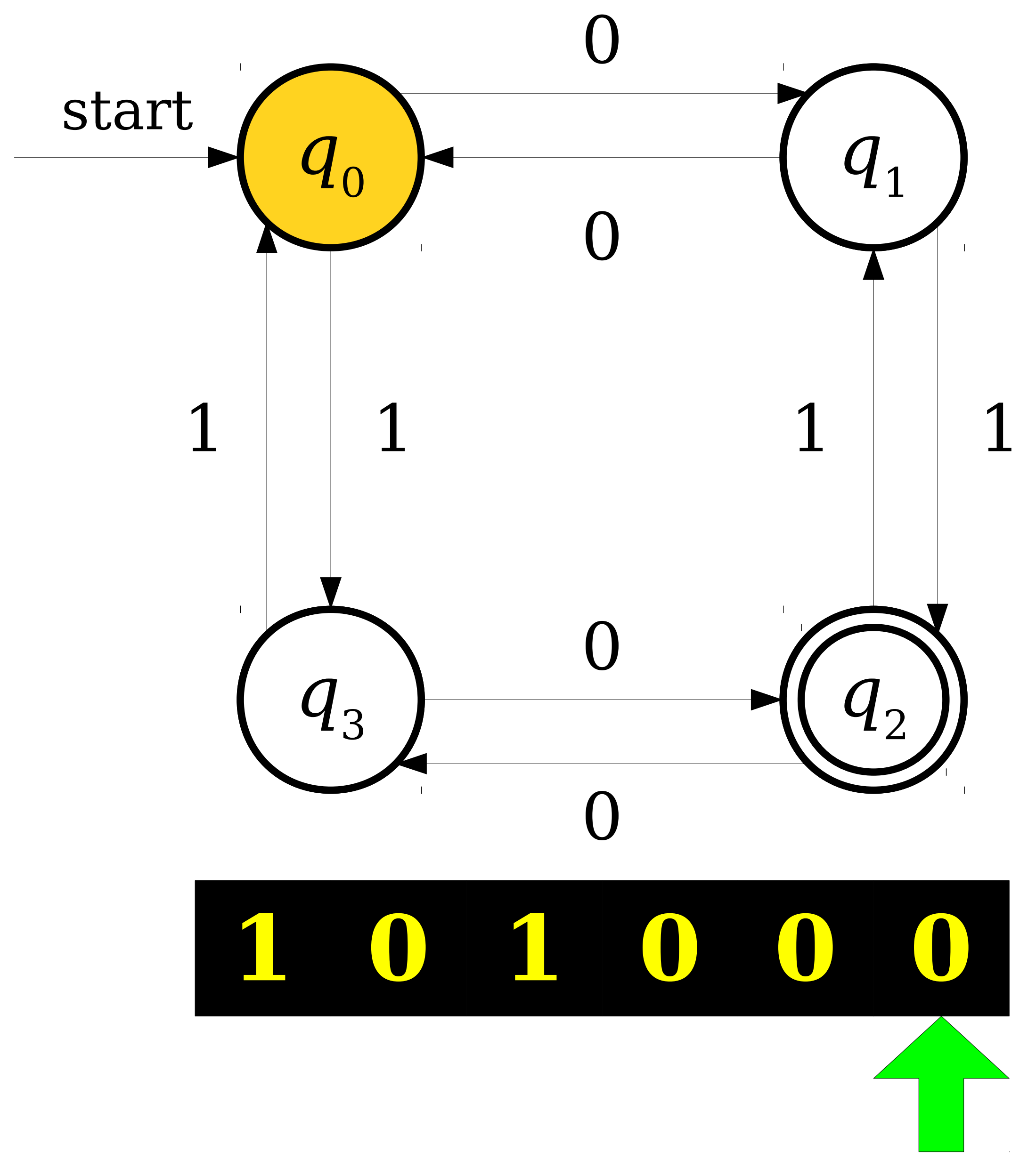
A Simple Finite Automaton



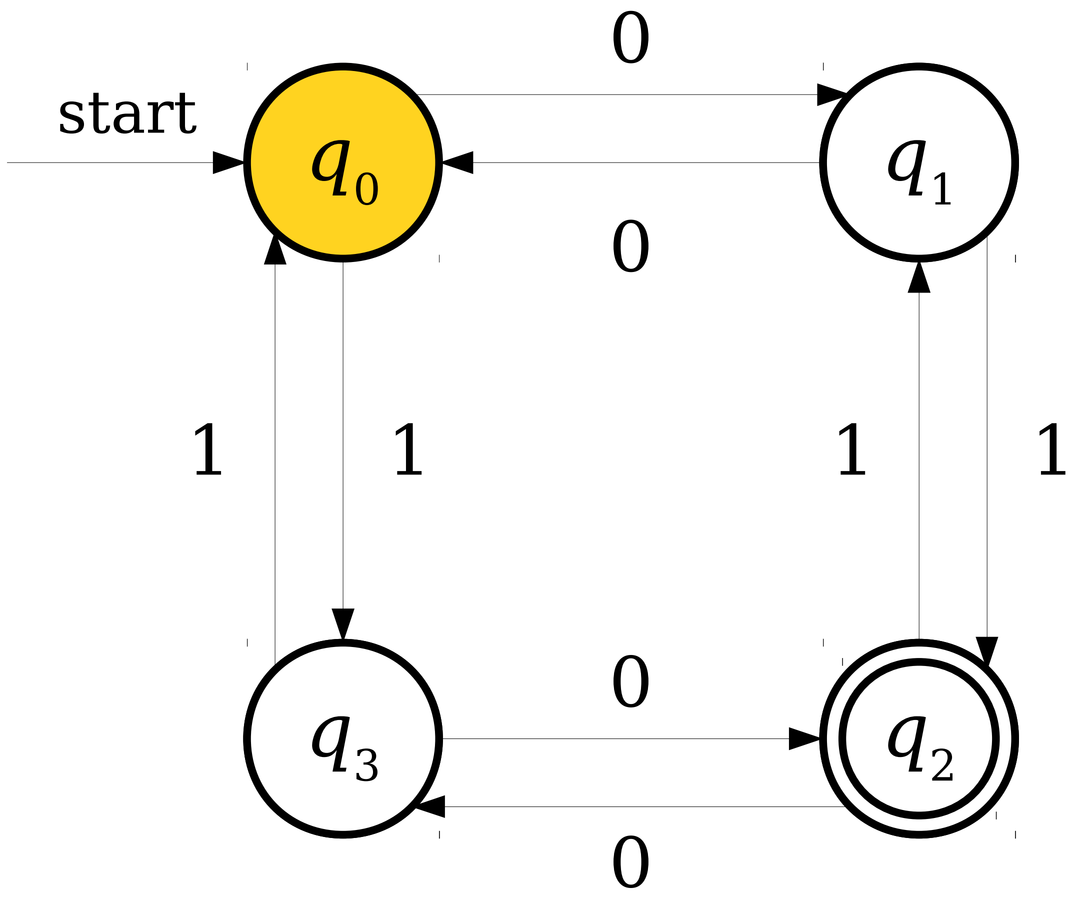
A Simple Finite Automaton



A Simple Finite Automaton



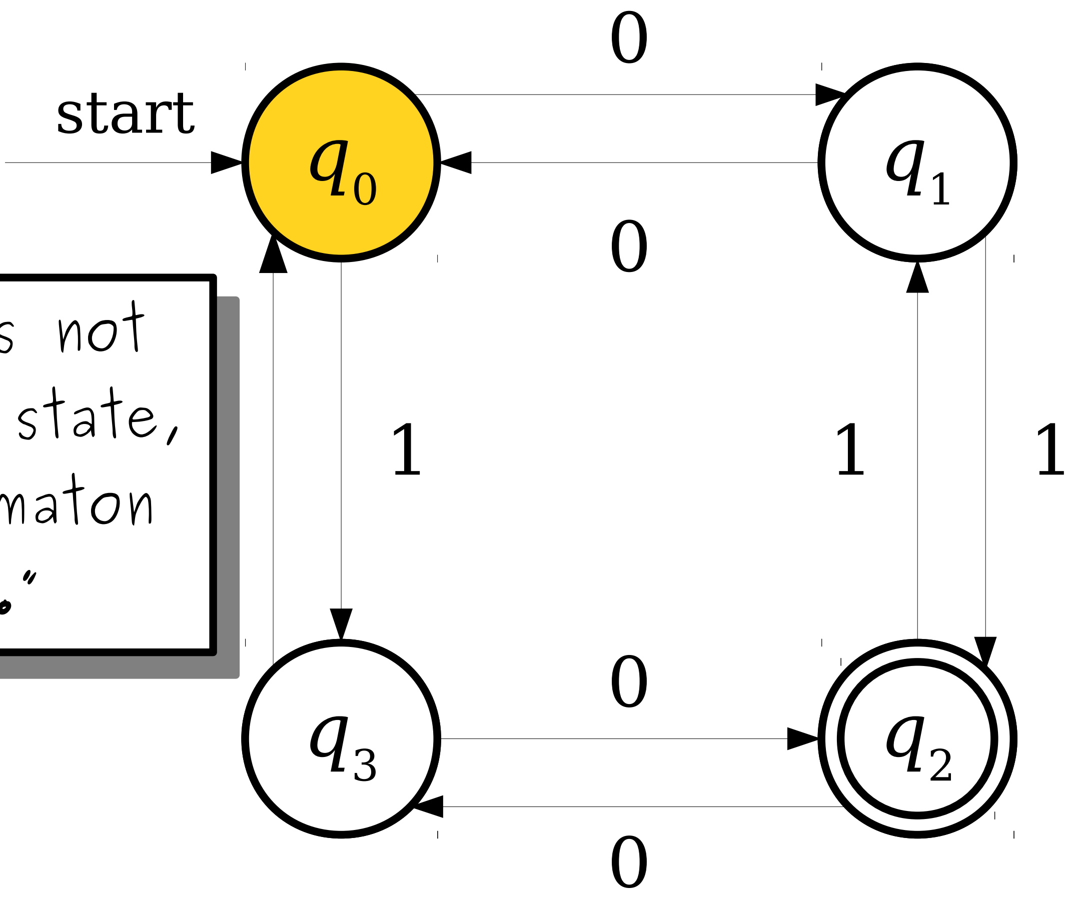
A Simple Finite Automaton



$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

1 0 1 0 0 0

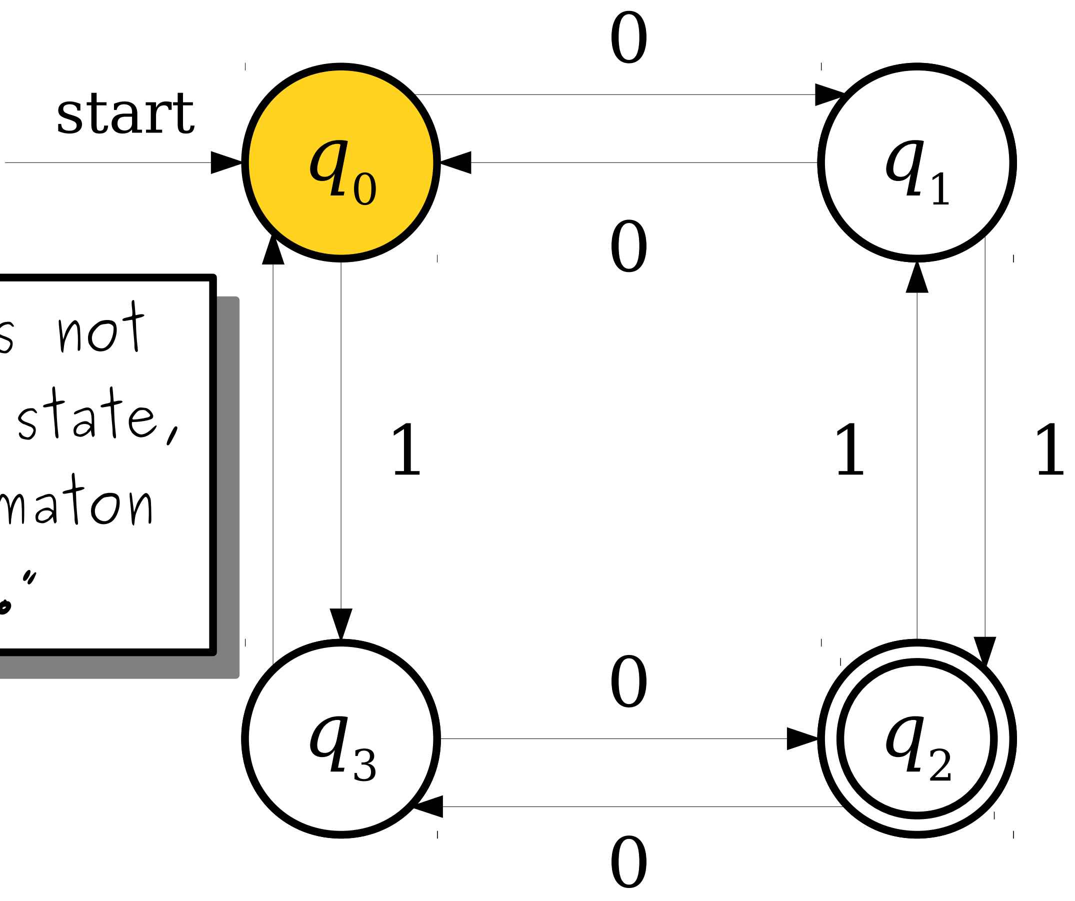
A Simple Finite Automaton


$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[r, bend left, "0"] \arrow[d, "1"] & q_1 \arrow[l, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u] \arrow[r, "0"] & \boxed{q_2} \arrow[l, bend left, "0"] \arrow[u, "1"]
\end{tikzcd}$$

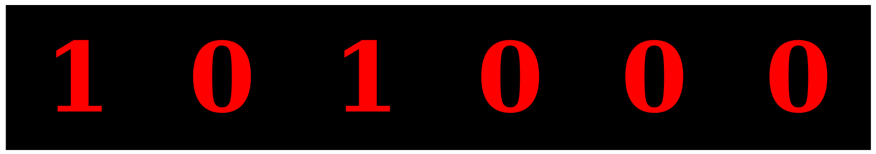
This state is not an accepting state, so the automaton says "no."

1 0 1 0 0 0

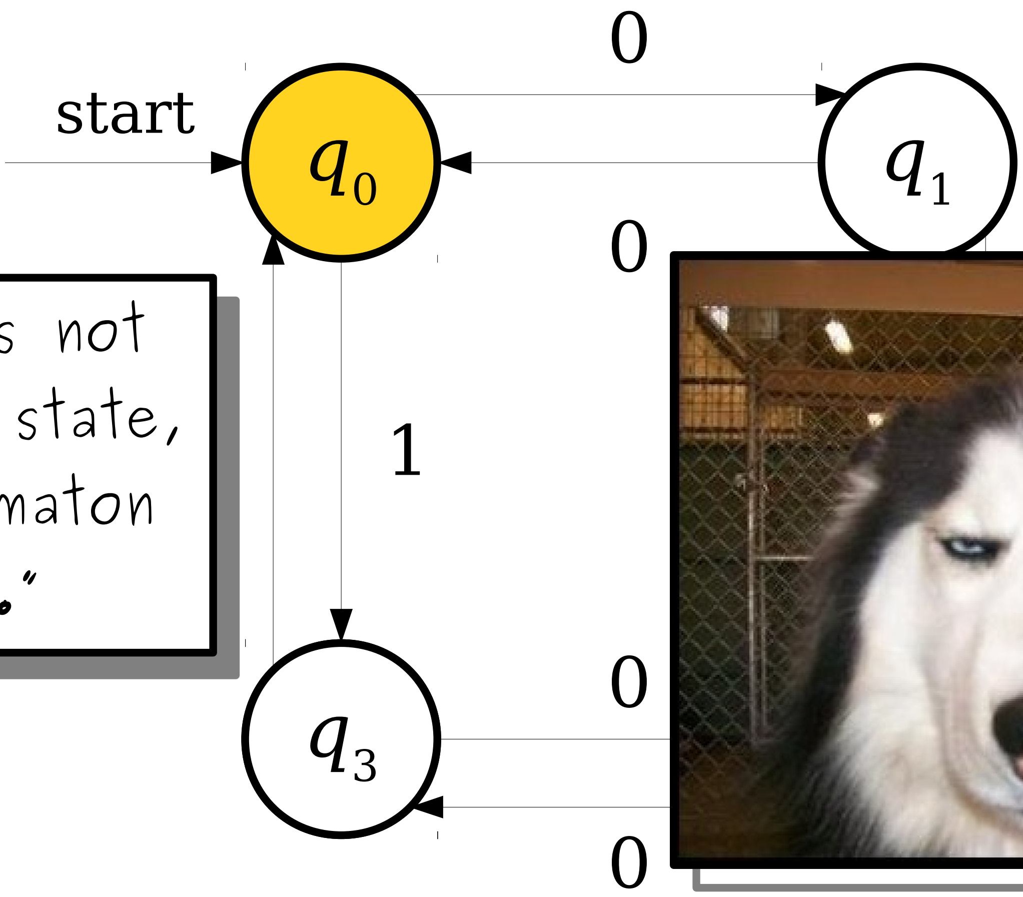
A Simple Finite Automaton



This state is not an accepting state, so the automaton says "no."



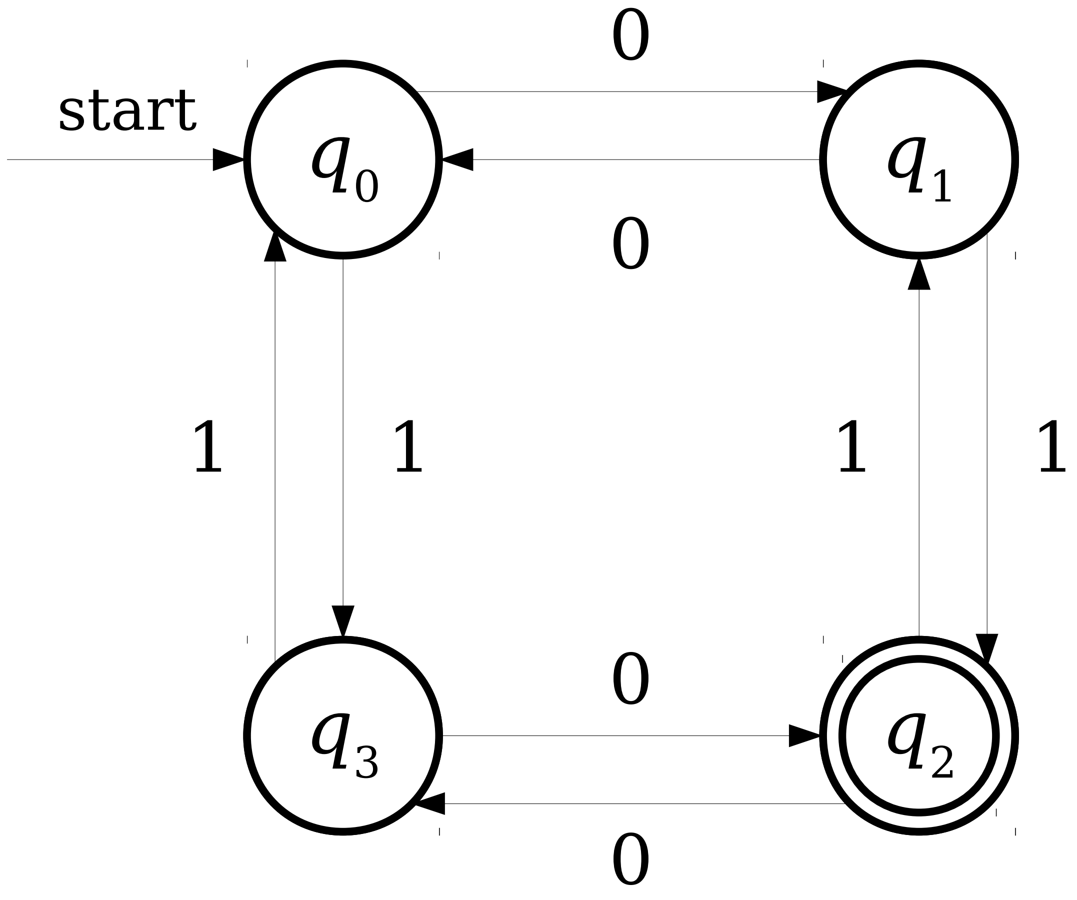
A Simple Finite Automaton



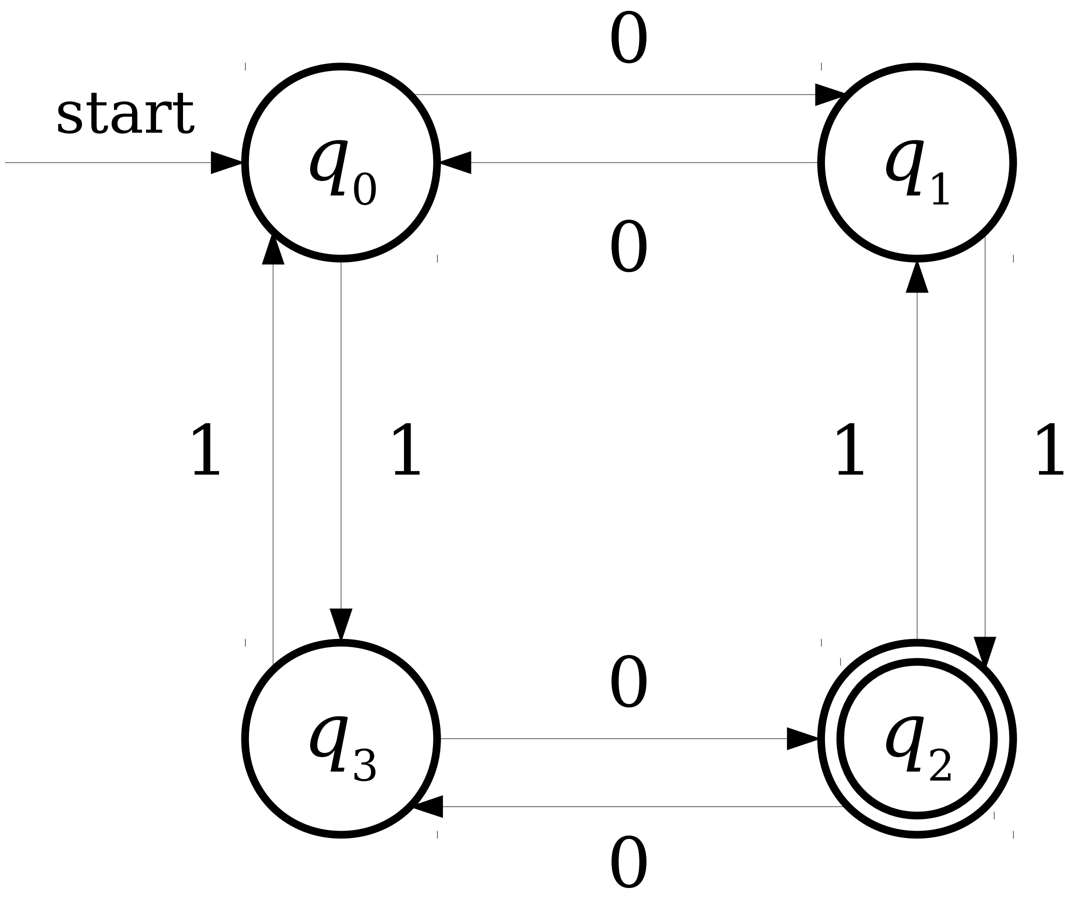
This state is not an accepting state, so the automaton says "no."

1 0 1 0 0 0

A Simple Finite Automaton



A Simple Finite Automaton



Try it yourself! Does the automaton **accept** (say yes) or **reject** (say no)?

1 1 0 1 1 1 0 0

The Story So Far

- A ***finite automaton*** is a collection of ***states*** joined by ***transitions***.
- Some state is designated as the ***start state***.
- Some states are designated as ***accepting states***.
- The automaton processes a string by beginning in the start state and following the indicated transitions.
- If the automaton ends in an accepting state, it ***accepts*** the input.
- Otherwise, the automaton ***rejects*** the input.

Time-Out For Announcements!

Solution Sets

- We'll be moving solution sets down to the Gates basement near the exterior stairwell for the weekend.

  `<rant style="because-i-care-about-you">`

  Please pick up solution sets! They're hugely valuable study resources. You get to see more examples of good proofs and learn the motivations behind the problems. I honestly don't understand why anyone wouldn't pick up the solutions; even if you got a perfect score on the problem sets, it's super valuable to see polished solutions and the motivation behind the problems.

  I've been getting some pressure from the CS department to reduce wasted paper, so going forward we're going to start printing fewer solution sets.

  `</rant>`

Problem Set Five

- Problem Set Four was due at the start of class today.
  - Want to use late days? Turn it in by Monday at the start of lecture.
- Problem Set Five goes out now and is due next Friday.
  - There is no checkpoint. You have a midterm to study for!
  - I'm aware that the exam is on Monday. I've done my best to pare down the problem set to a reasonable size. I'd at least look over it today, though I suspect most of you are going to start it on Tuesday.

Midterm Logistics

- Our first midterm is this upcoming Monday from 7PM – 10PM.
- Room locations divvied up by last (family) name:
  - `Abd - Lin`: Go to **Cubberly Auditorium**.
  - `Liu - Raj`: Go to **370-370**.
  - `Ram - Zhu`: Go to **420-040**.
- Closed-book, closed-computer, limited-notes.
  - You can have a double-sided 8.5” × 11” sheet of notes when you take the exam.
- Topic coverage is PS1 – PS3 and Lectures 00–08.

Extra Practice

- We have released
  - three sets of extra practice problems (EPP1 – EPP3), with solutions;
  - one set of challenge problems, without solutions;
  - a practice midterm, with solutions; and
  - a packet of review problems in CS103A.
- Need more practice? Please let us know!

Skills for the Midterm

- The four key skills you'll need for the midterm exam are the following:
  - ***Proof setup and structure:*** Can you set up and execute proofs of common types of statements (universal statements, existential statements, and implications)?
  - ***Understanding and manipulating first-order logic:*** Can you translate statements in first-order logic? Can you negate first-order statements? Do you understand the nuances of the connectives?
  - ***Writing proofs involving formal definitions:*** Can you provide a proof that rigorously calls back to the terms and definitions we've defined so far? Can you write a proof of a statement in first-order logic (and, importantly, can you do so in plain English?)
  - ***Understanding the definitions and intuitions covered so far:*** Do you remember the definitions of injections, surjections, bijections, equivalence relations, cardinality, etc.? Do you have an intuition for them?

Your Questions

“What do you do to relax when you're stressed? Any advice for dealing with exam stress?”

Personally? I like baking bread and cooking Korean food. Kneading bread dough is extremely cathartic. Highly recommended, and very tasty!

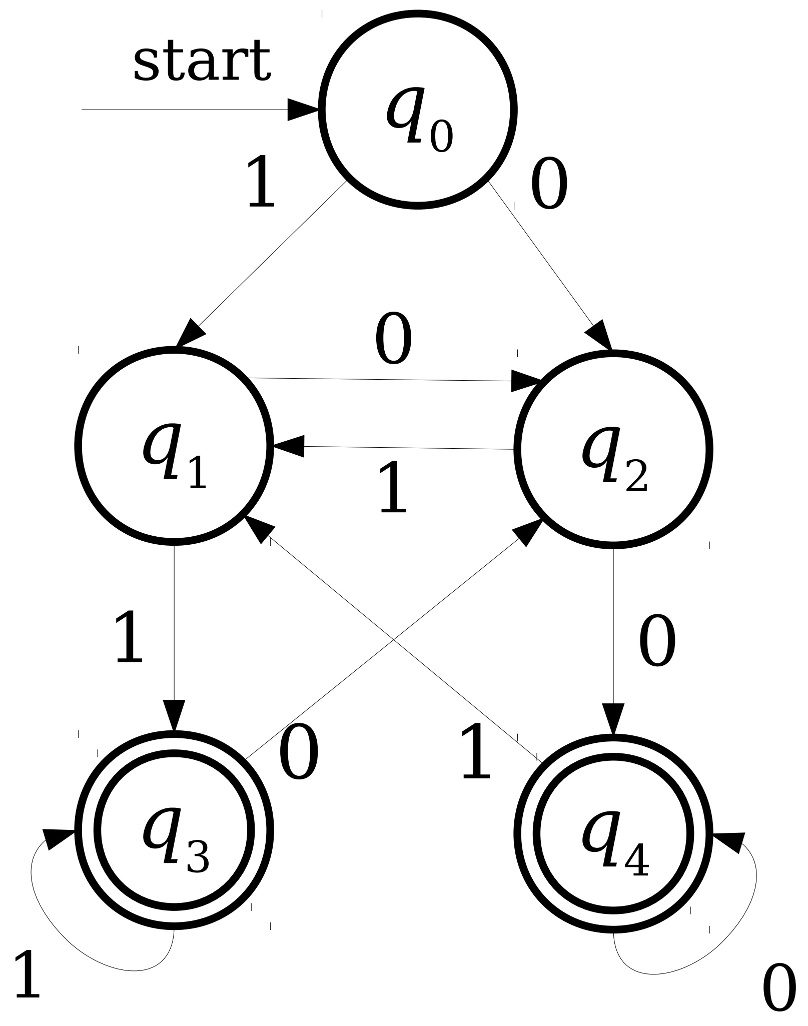
As for dealing with exam stress: make sure that you're getting support from the people around you. From experience, going in alone can make you miserable. Find ways to take breaks every now and then to destress and do something fun. Oh, and exercise. Exercise is good. Especially useful if you keep eating all the bread that you're baking. ☺

“So what's your deal with the number 137?”

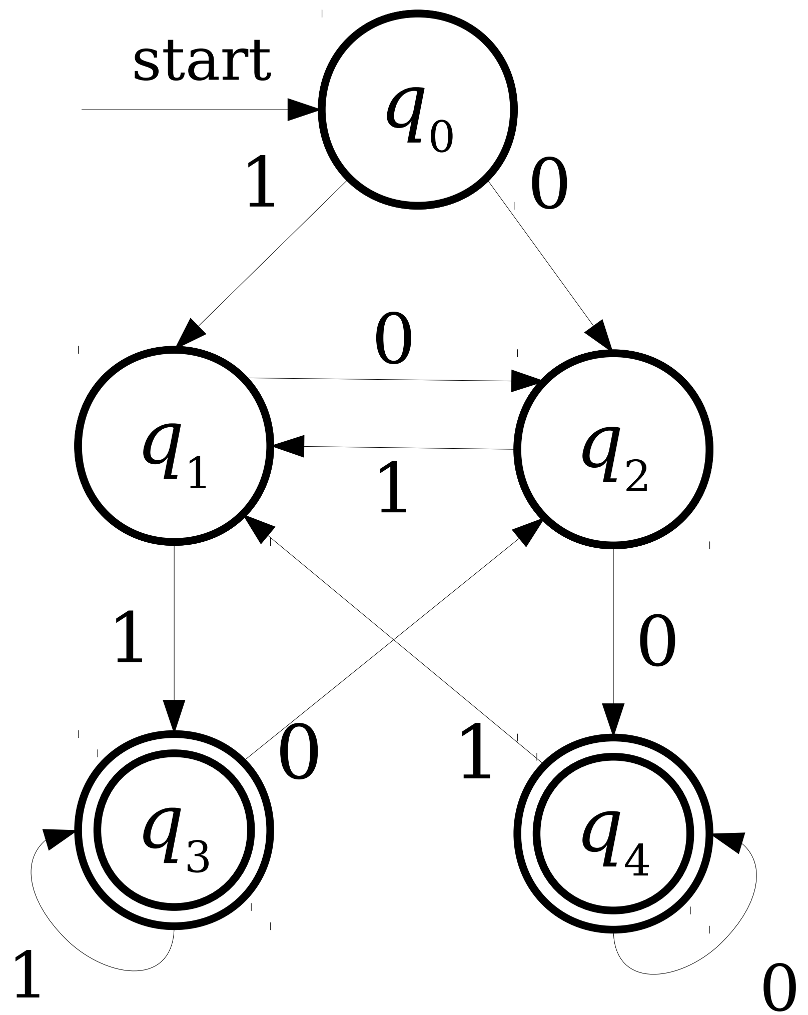
$$\boxed{\alpha = \frac{1}{4\pi\epsilon_0}\frac{e^2}{\hbar c}}$$

Back to CS103!

Accepting States, Revisited

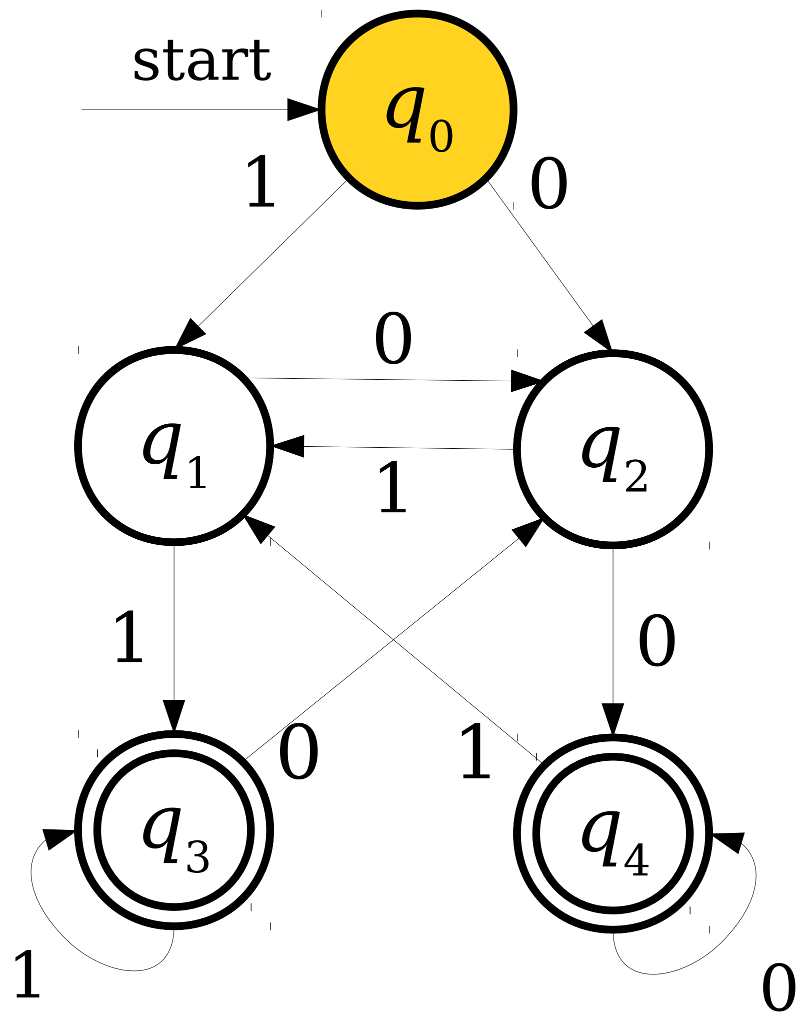


Accepting States, Revisited



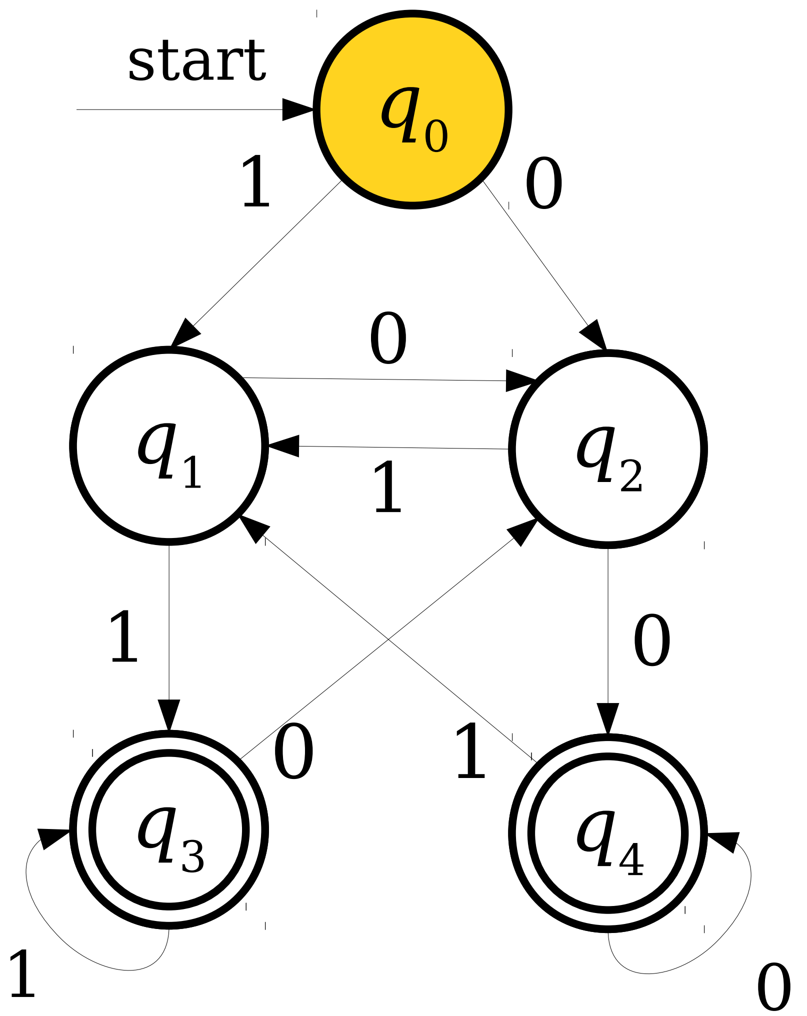
1 1 0 1

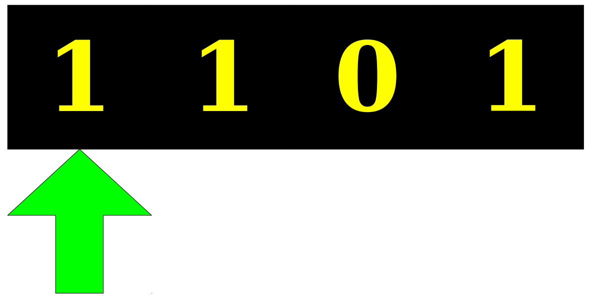
Accepting States, Revisited



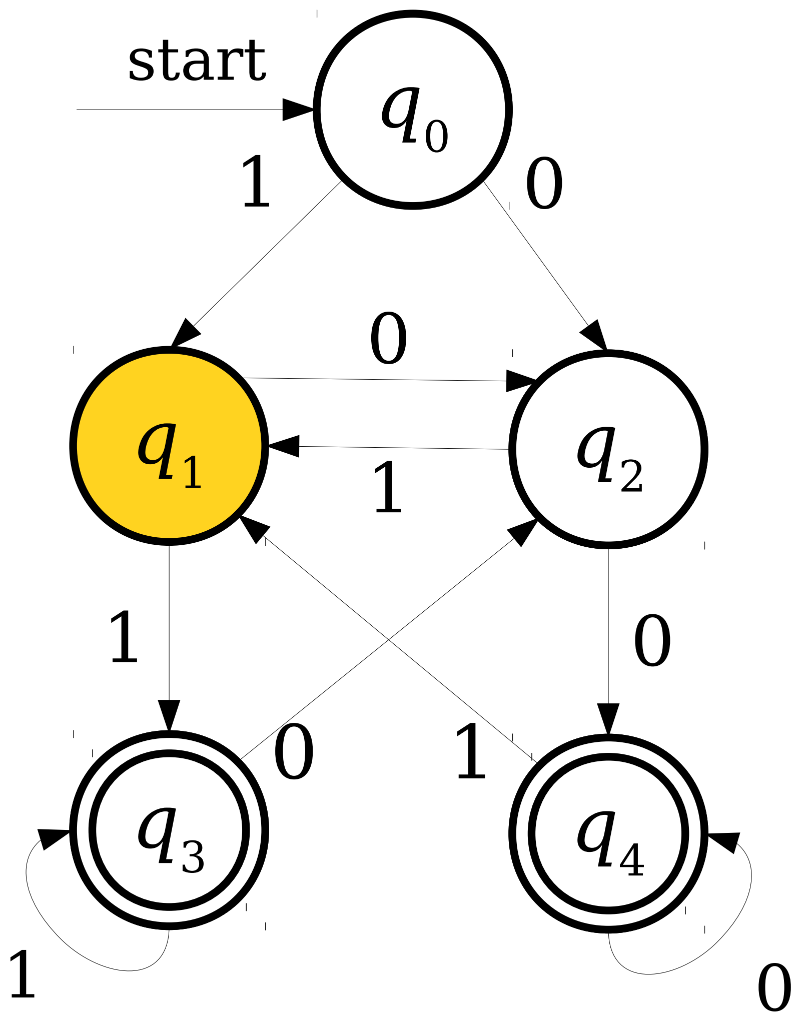
1 1 0 1

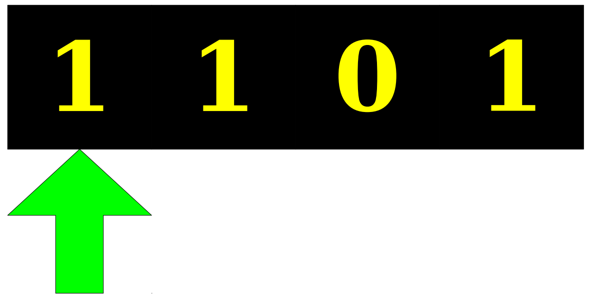
Accepting States, Revisited



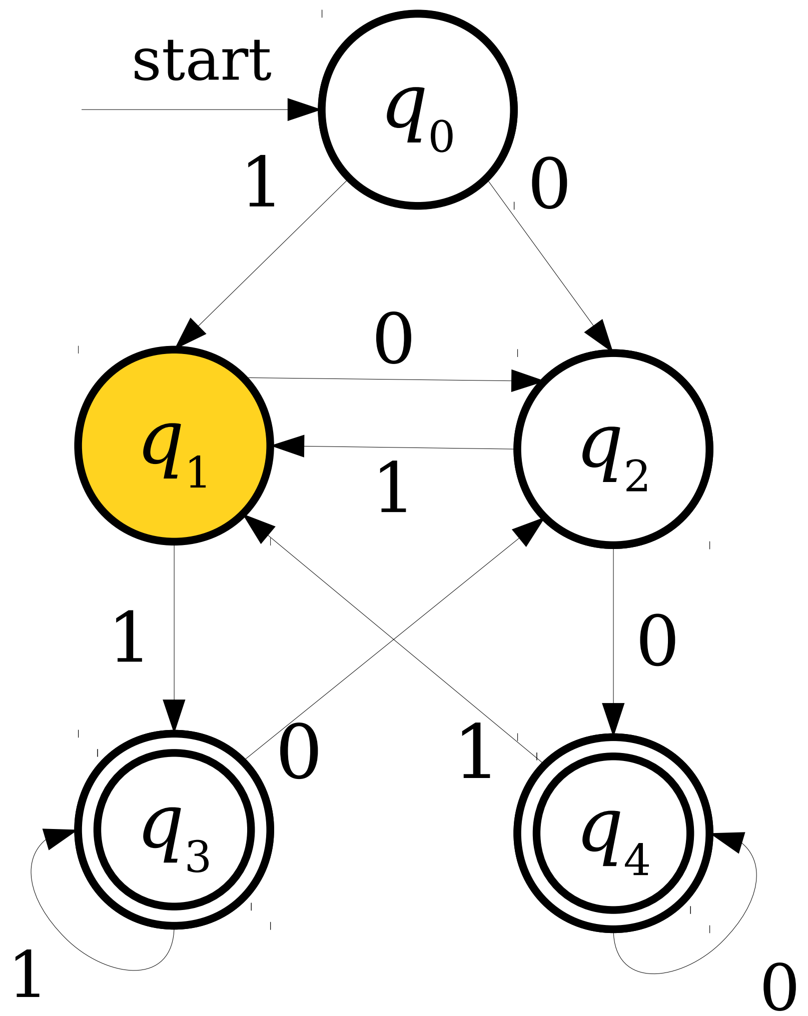


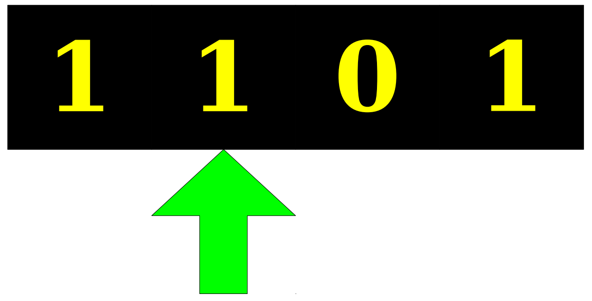
Accepting States, Revisited



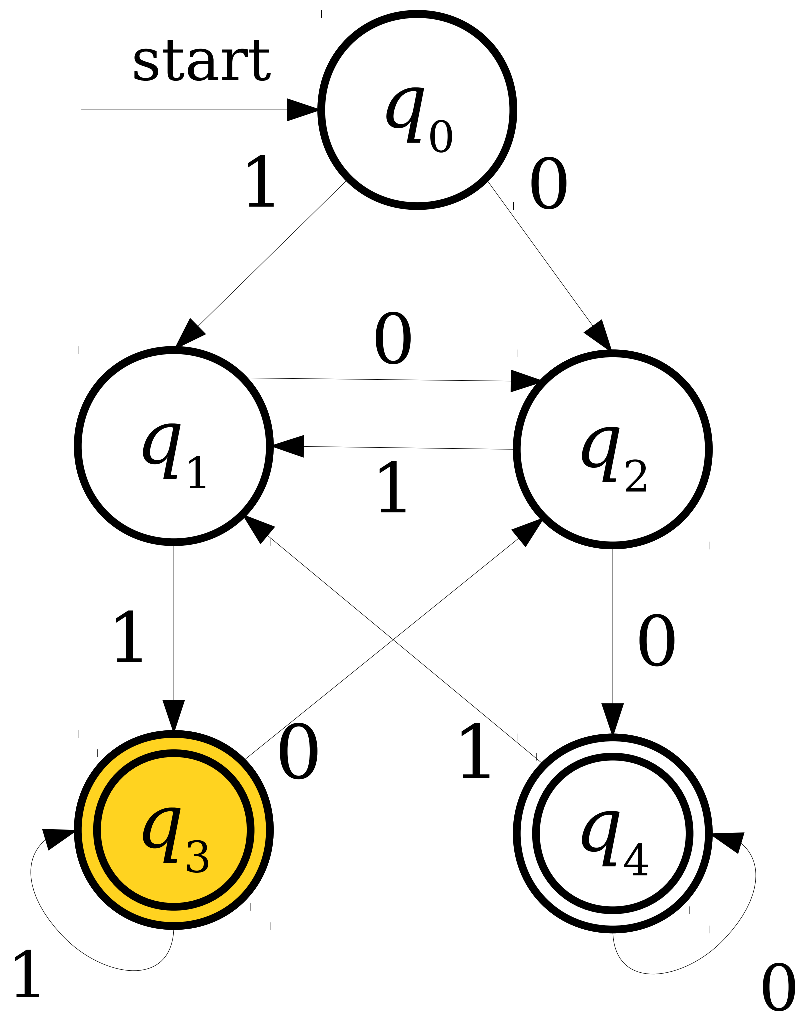


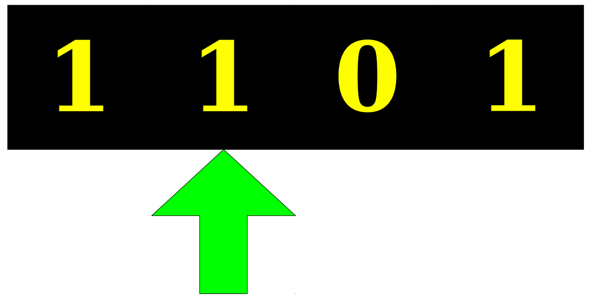
Accepting States, Revisited



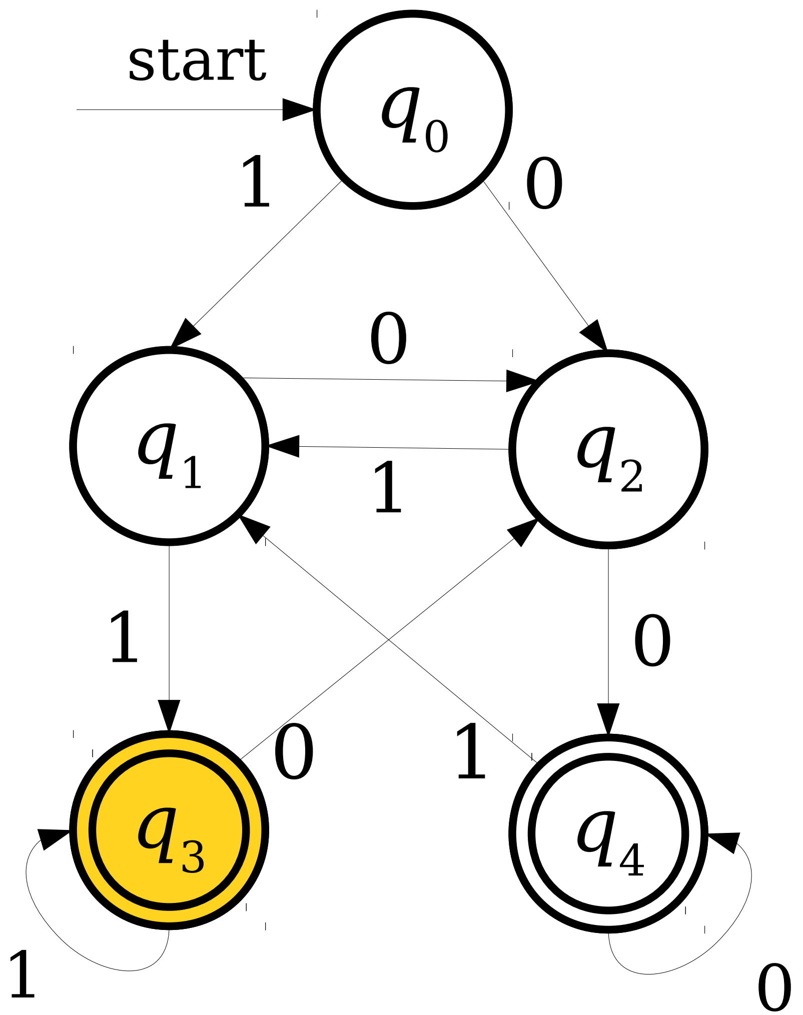


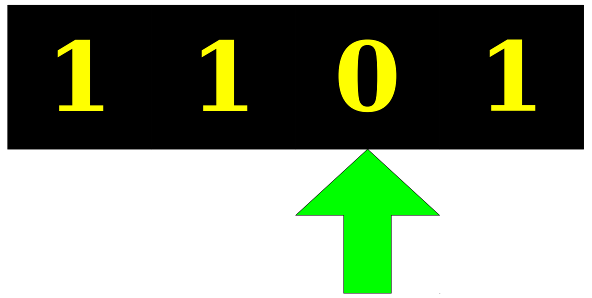
Accepting States, Revisited



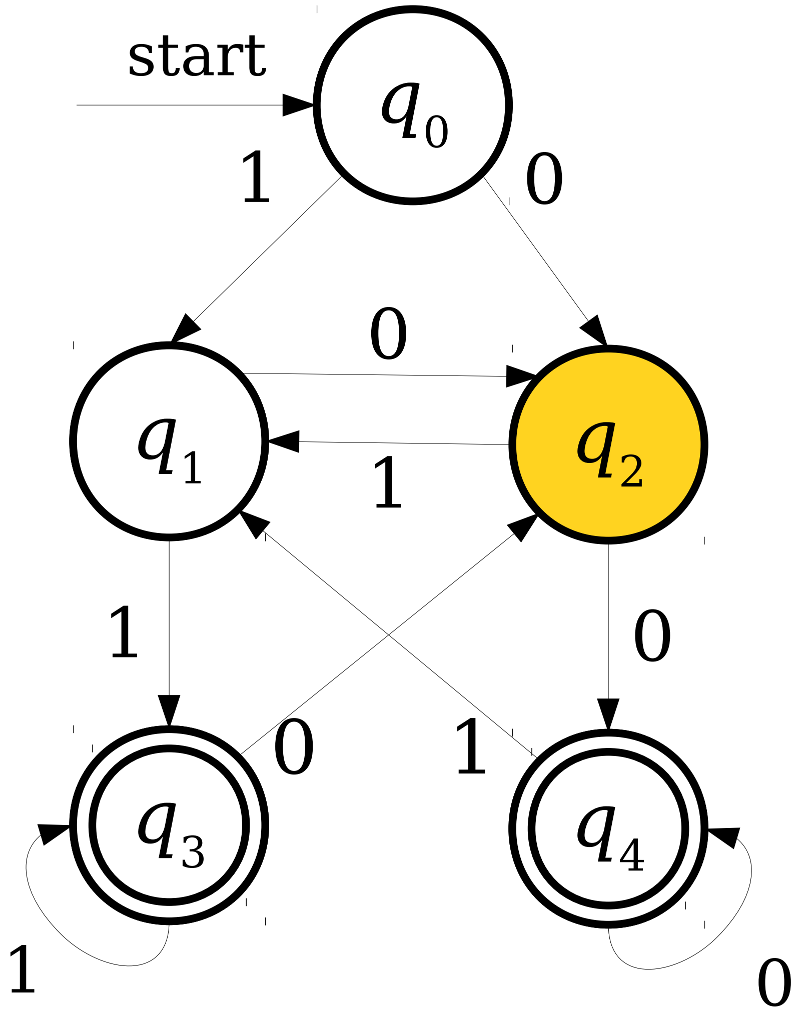


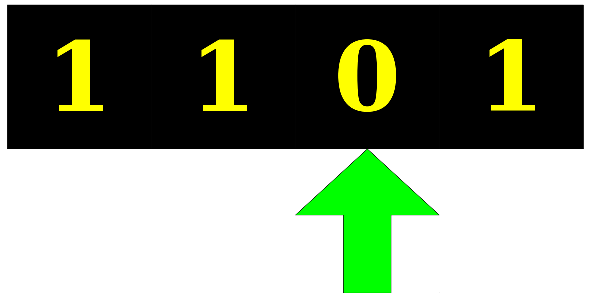
Accepting States, Revisited



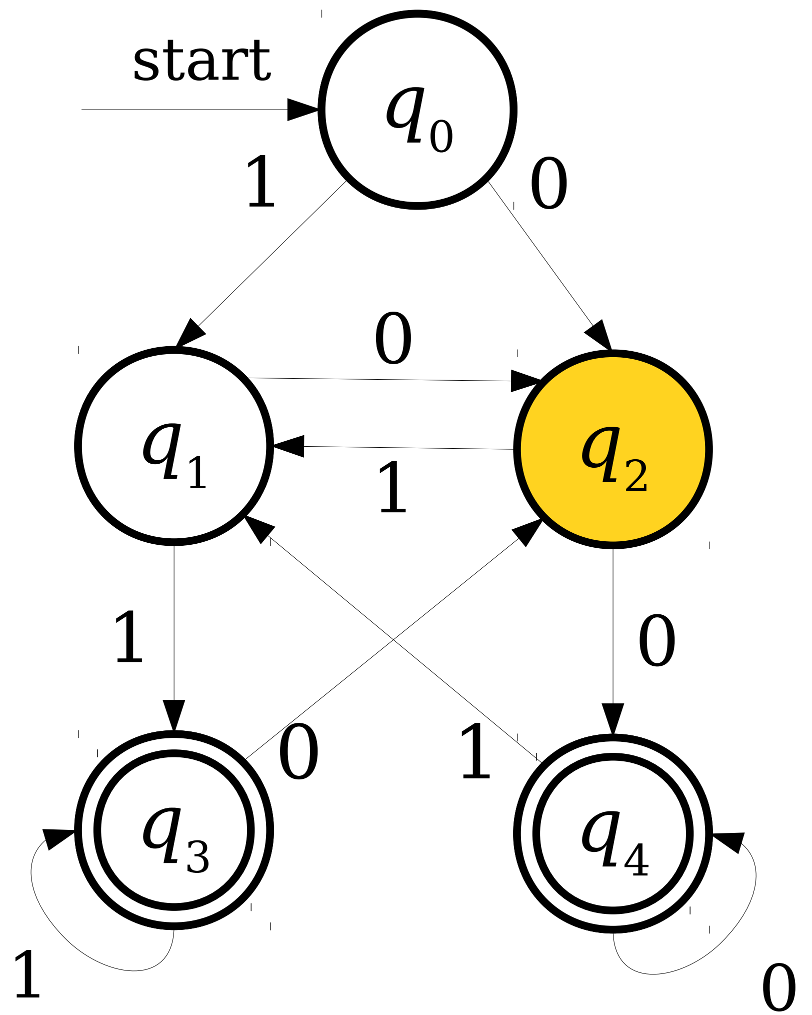


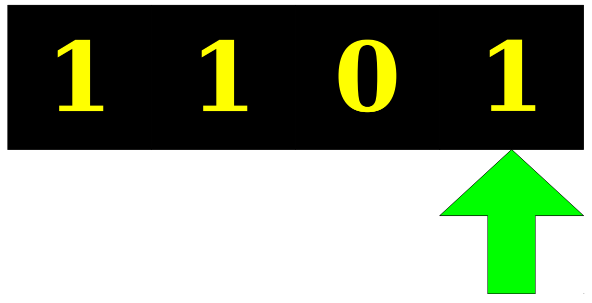
Accepting States, Revisited



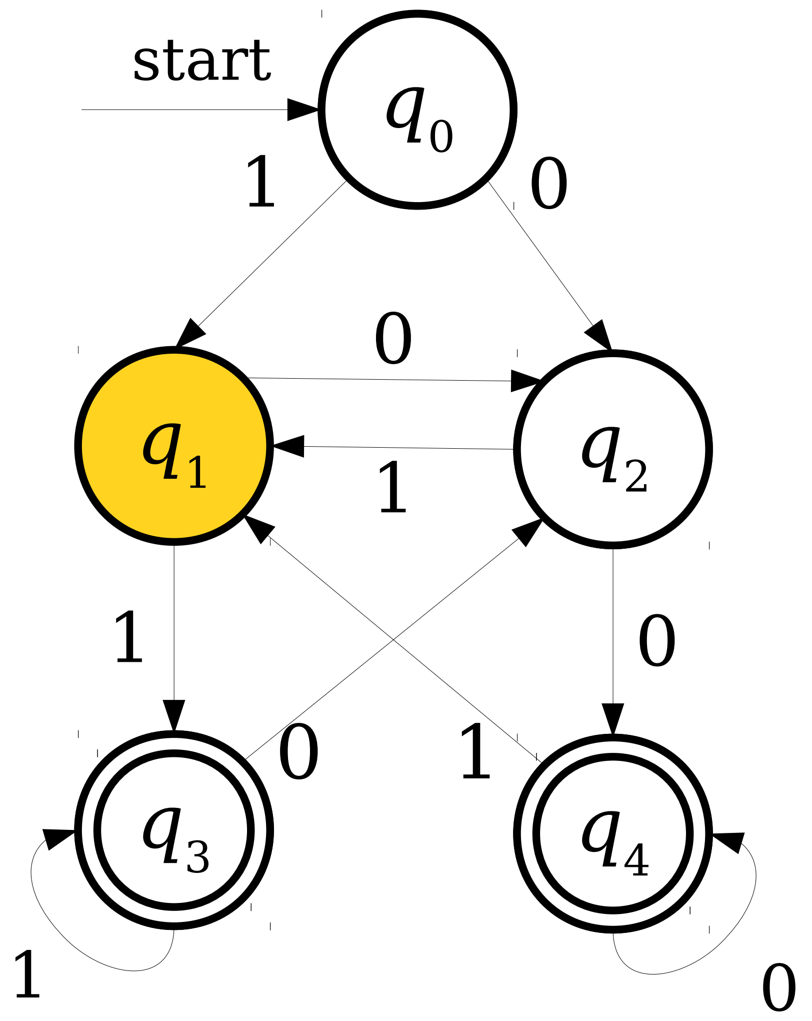


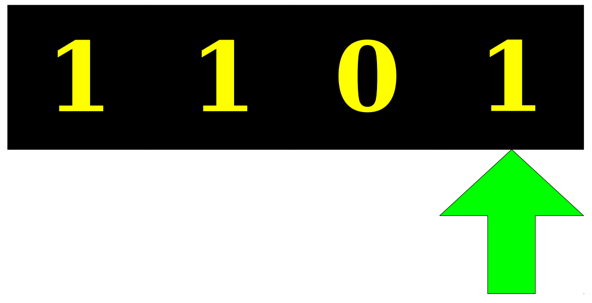
Accepting States, Revisited



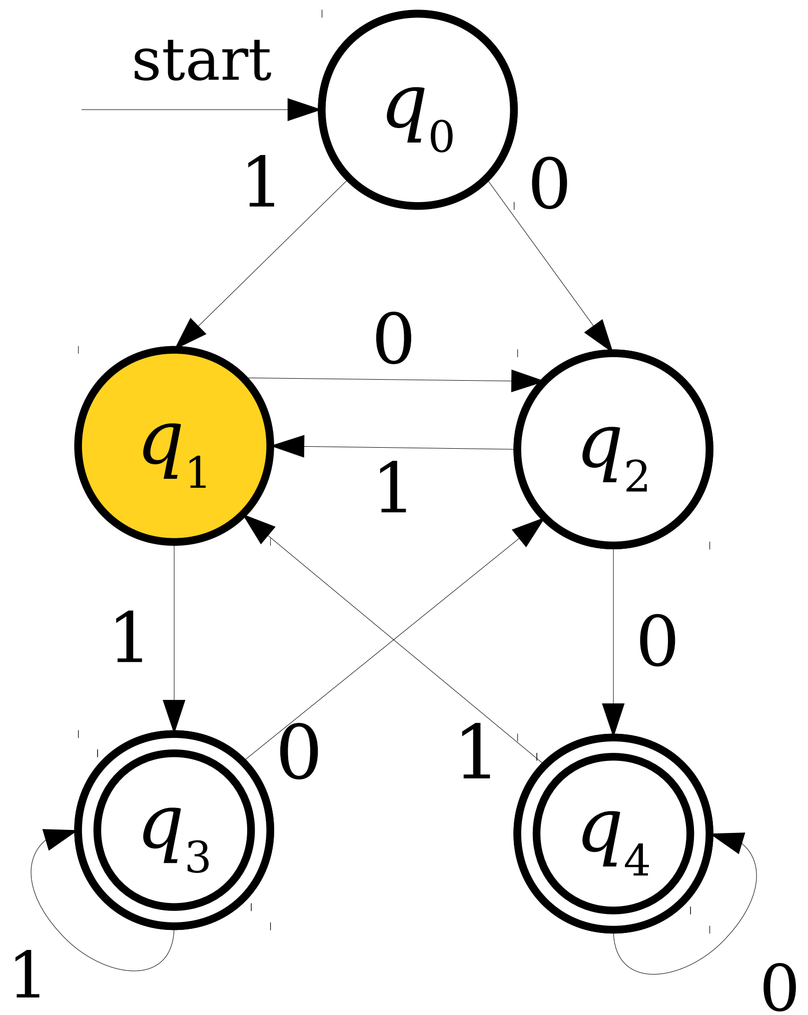


Accepting States, Revisited



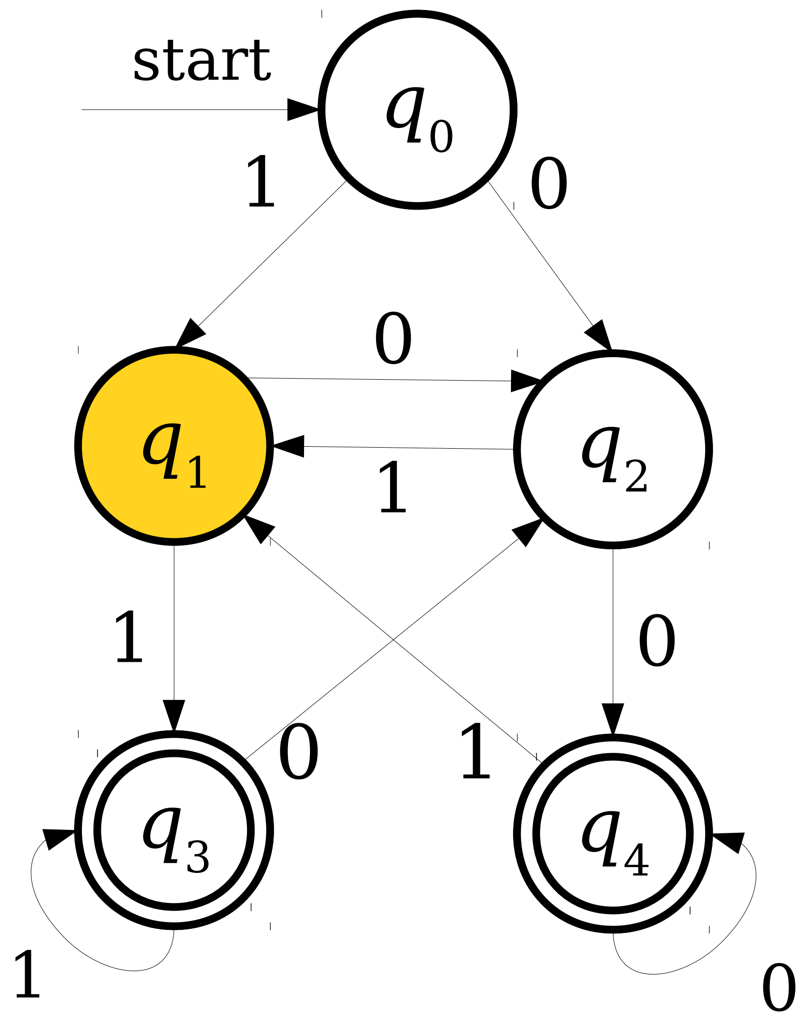


Accepting States, Revisited



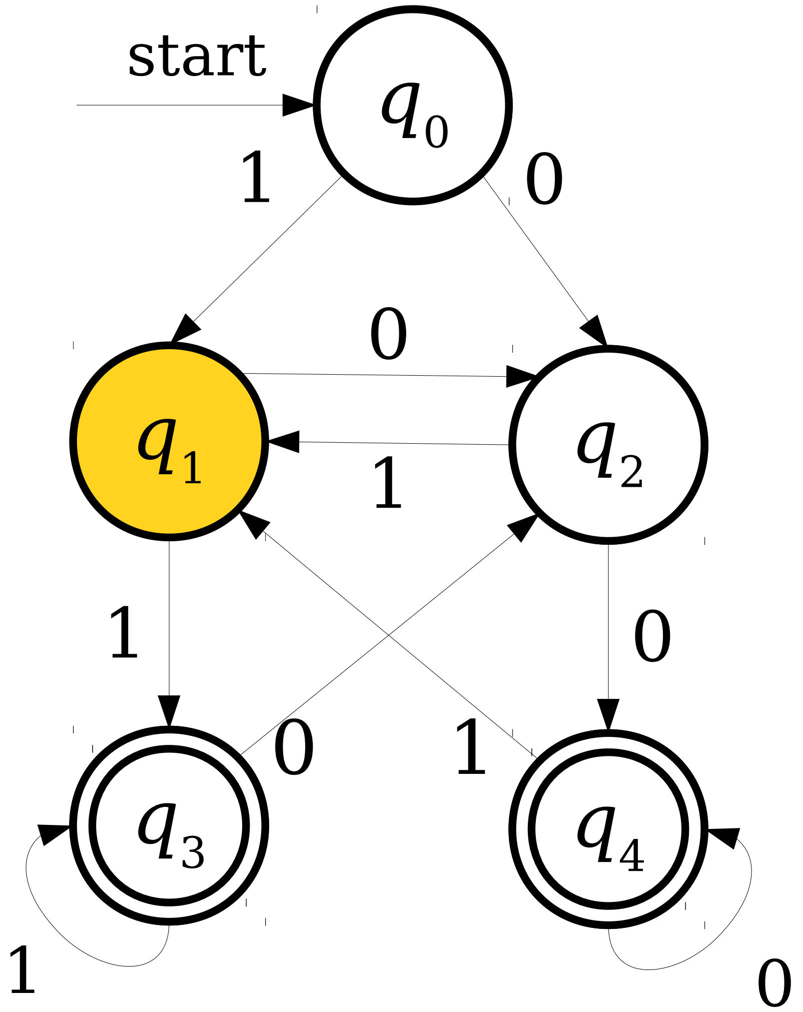
1 1 0 1

Accepting States, Revisited



1 1 0 1

Accepting States, Revisited

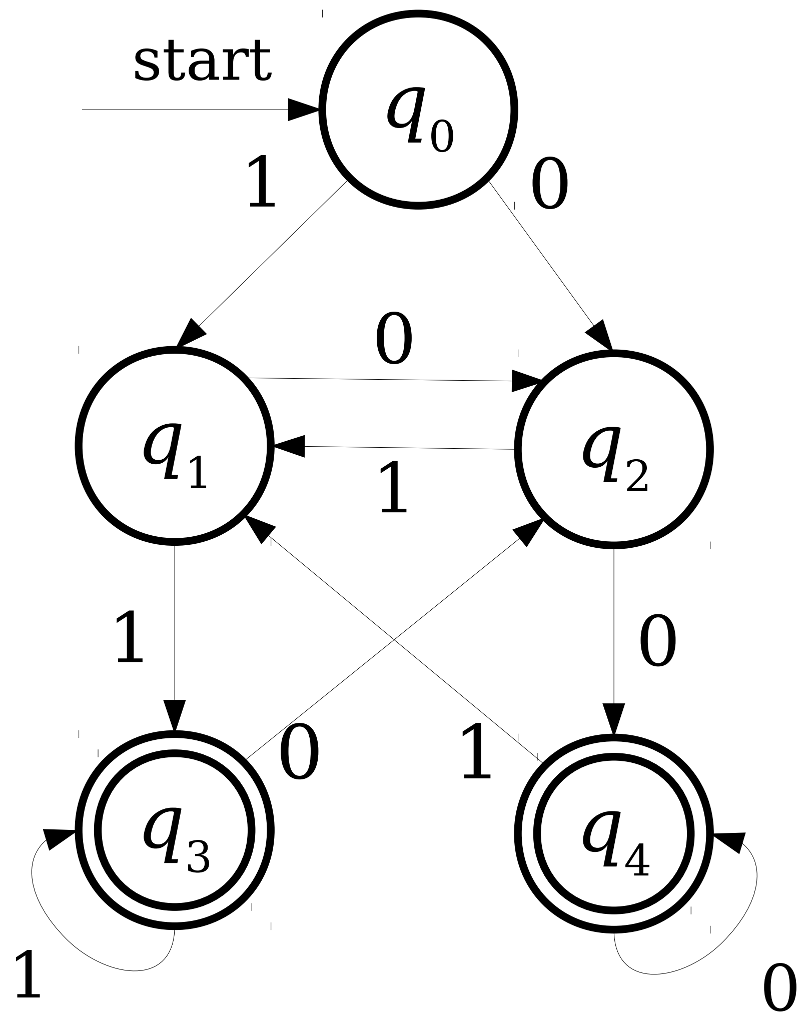


1 1 0 1

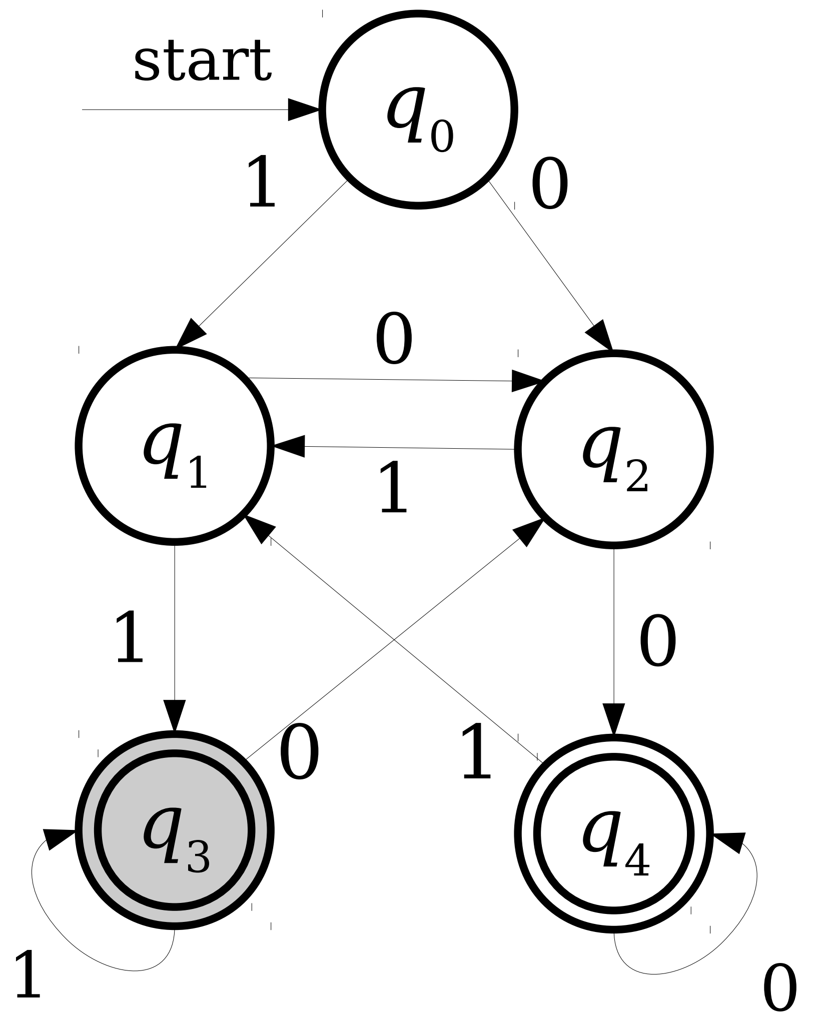
A finite automaton does ***not*** accept as soon as it enters an accepting state.

A finite automaton accepts if it ***ends*** in an accepting state.

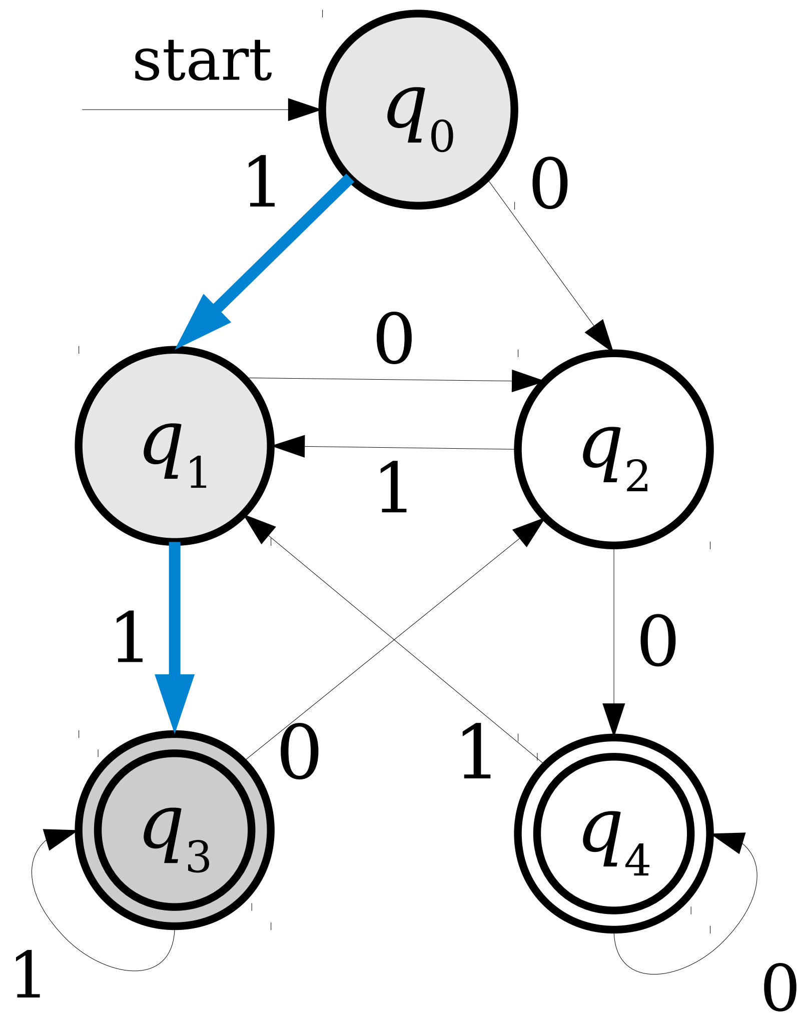
What Does This Accept?



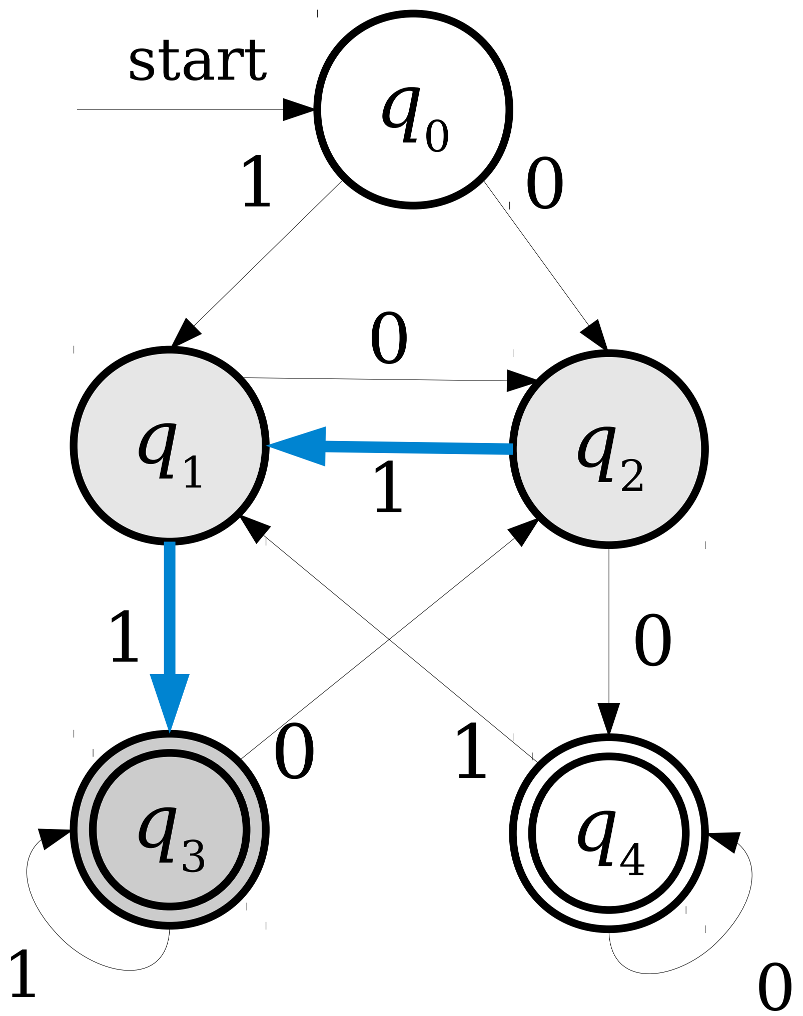
What Does This Accept?



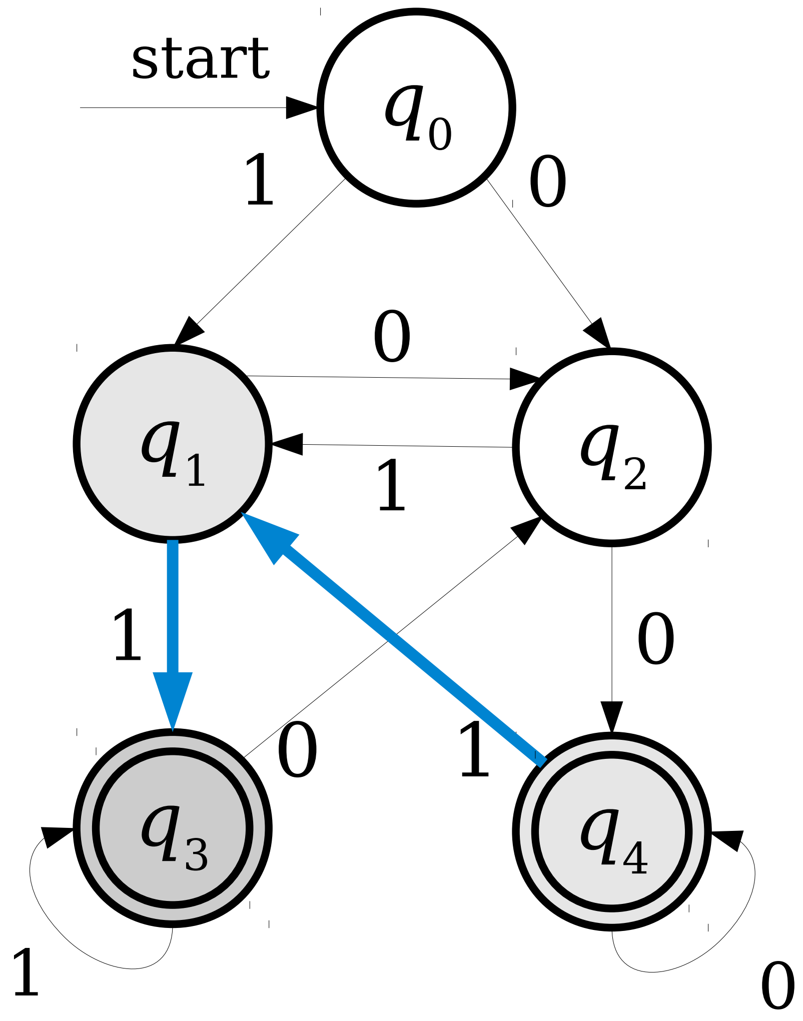
What Does This Accept?



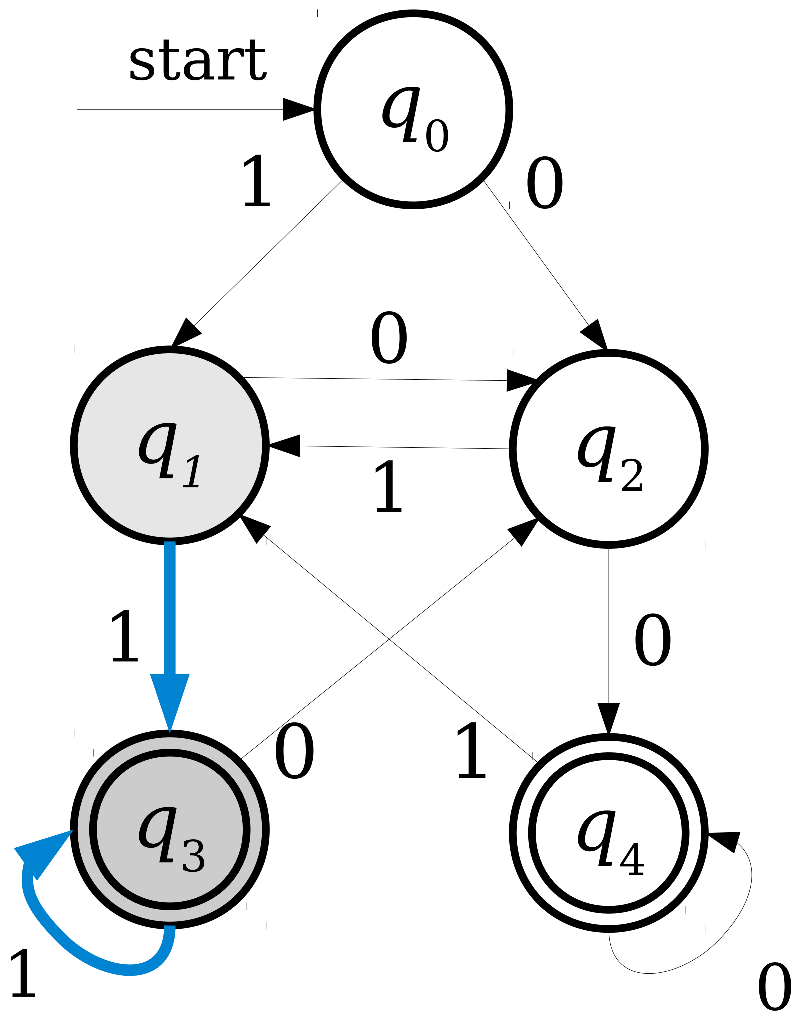
What Does This Accept?



What Does This Accept?

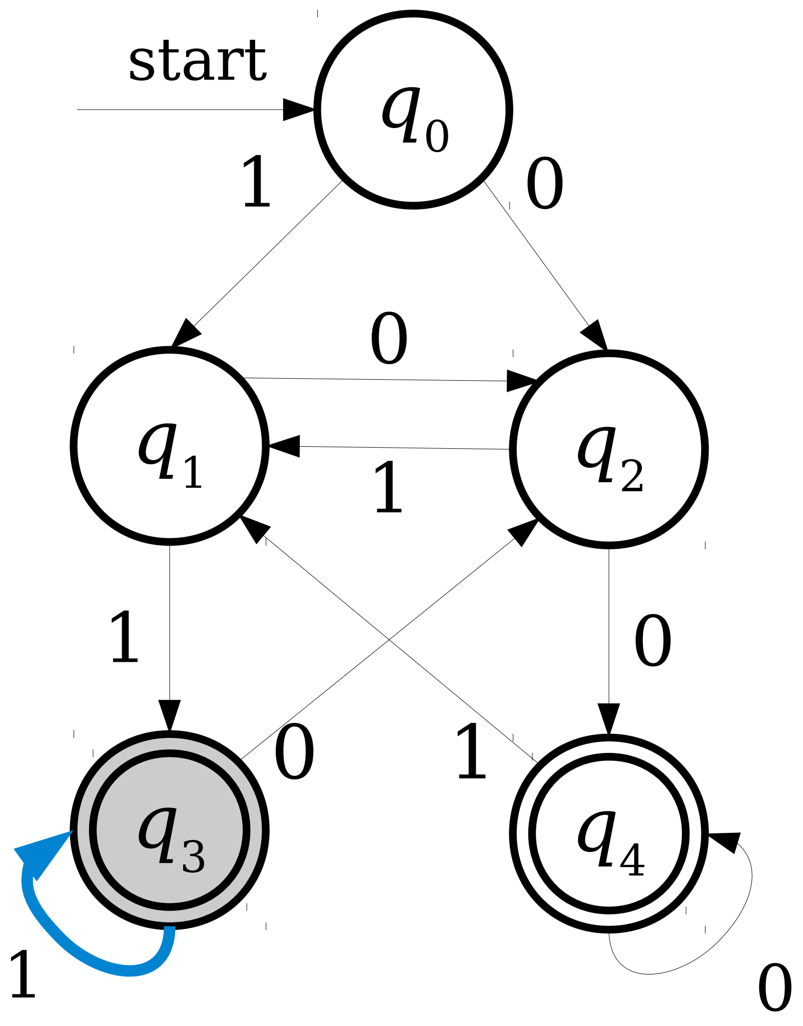


What Does This Accept?

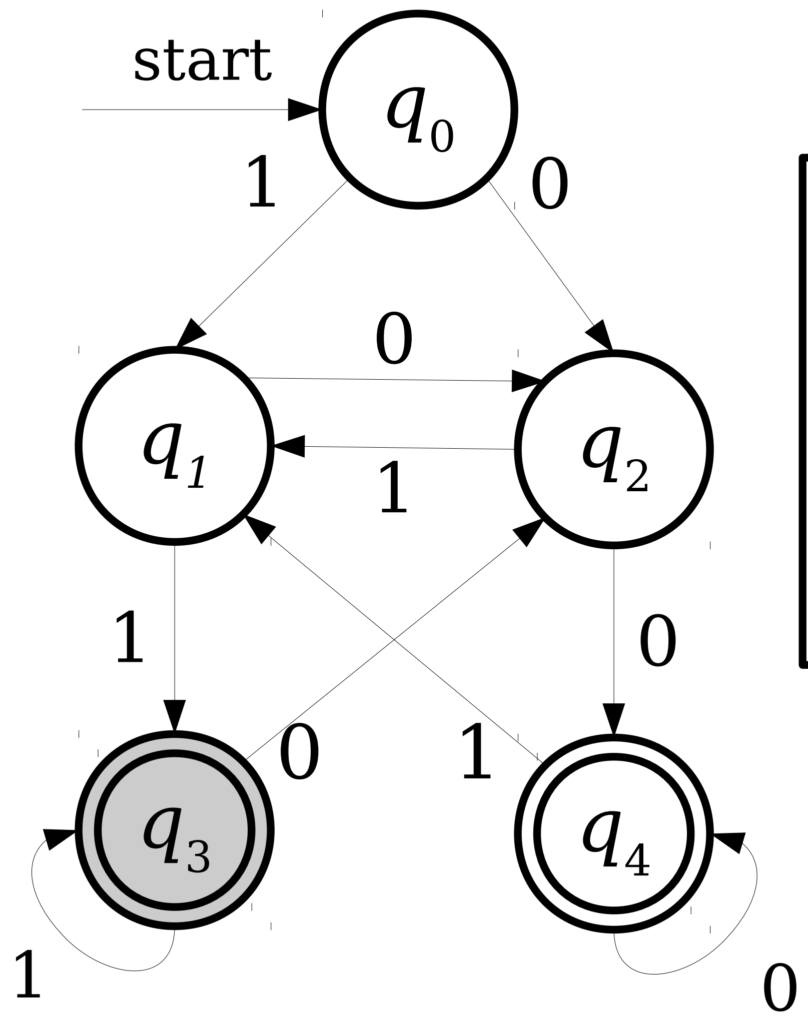


$$\begin{tikzcd}
 & q_0 \arrow[dl, "1"'] \arrow[dr, "0"] & \\
q_1 \arrow[rr, bend left, "0"] \arrow[d, "1"] & & q_2 \arrow[ll, "1"] \arrow[d, "0"] \\
q_3 \arrow[loop left, "1"] \arrow[urr, "0"] & & q_4 \arrow[loop right, "0"] \arrow[ull, "1"]
\end{tikzcd}$$

What Does This Accept?

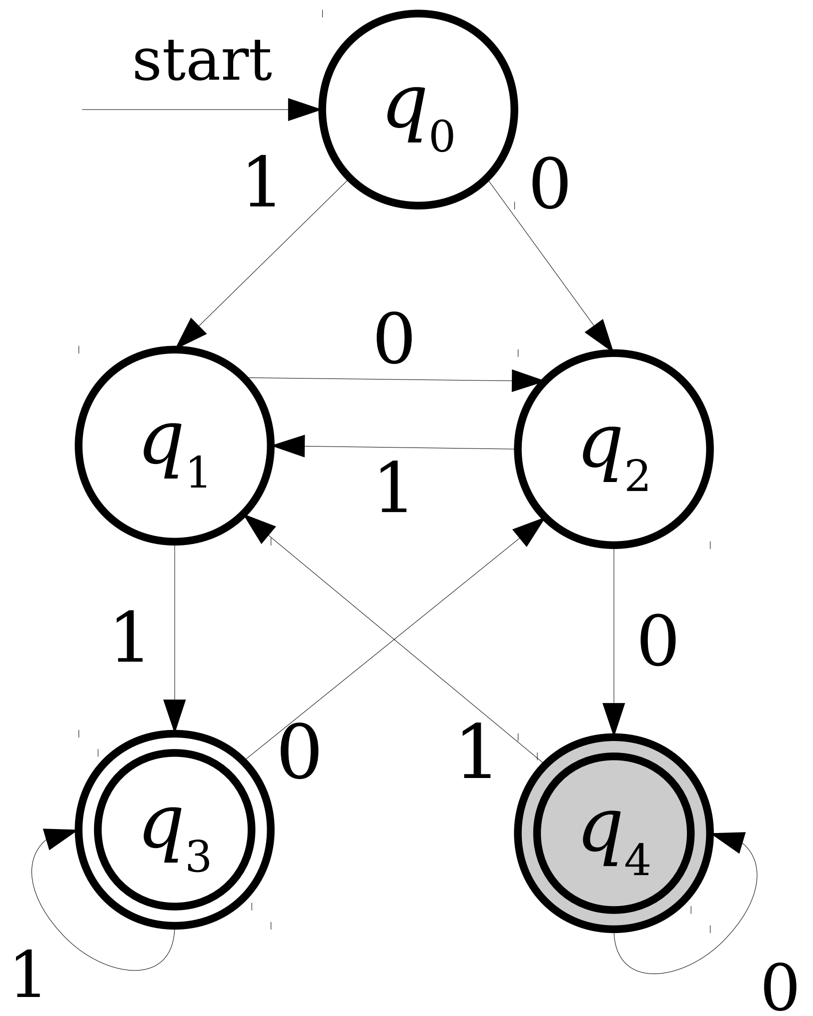


What Does This Accept?

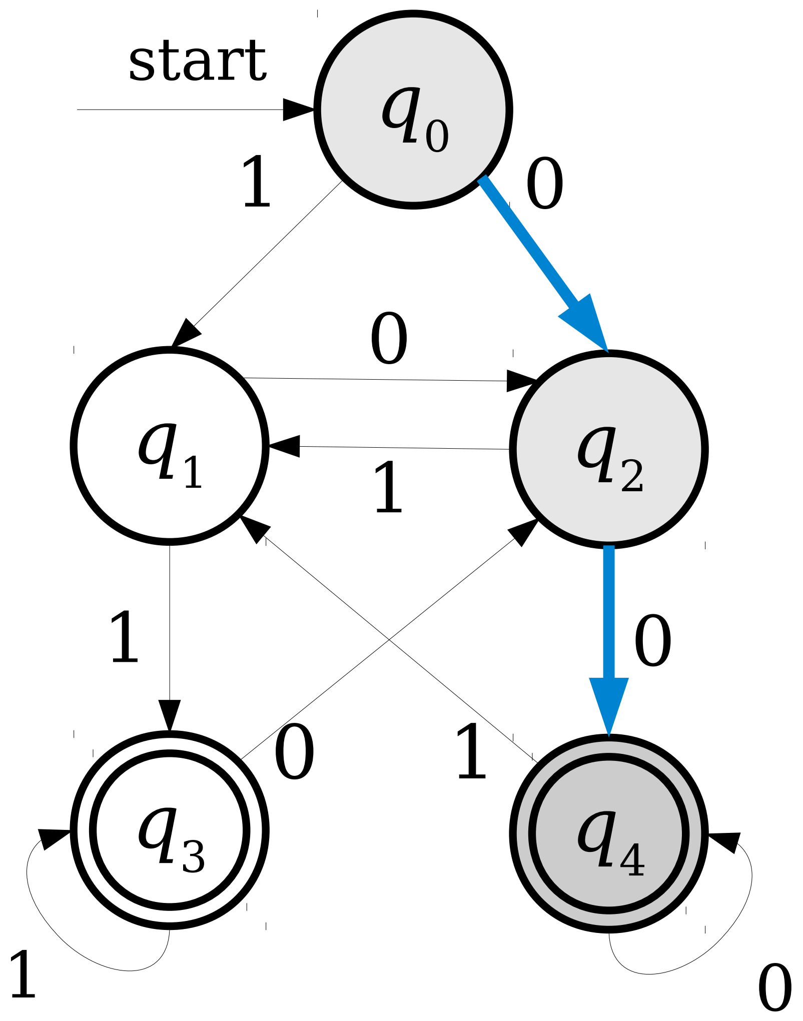

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[dl, "1"'] \arrow[dr, "0"] & \\
q_1 \arrow[rr, shift left, "0"] \arrow[d, "1"] & & q_2 \arrow[ll, shift left, "1"] \arrow[d, "0"] \\
q_3 \arrow[urr, "0"] \arrow[loop left, "1"] & & q_4 \arrow[ull, "1"'] \arrow[loop right, "0"]
\end{tikzcd}$$

No matter where we start in the automaton, after seeing two 1's, we end up in accepting state q_3.

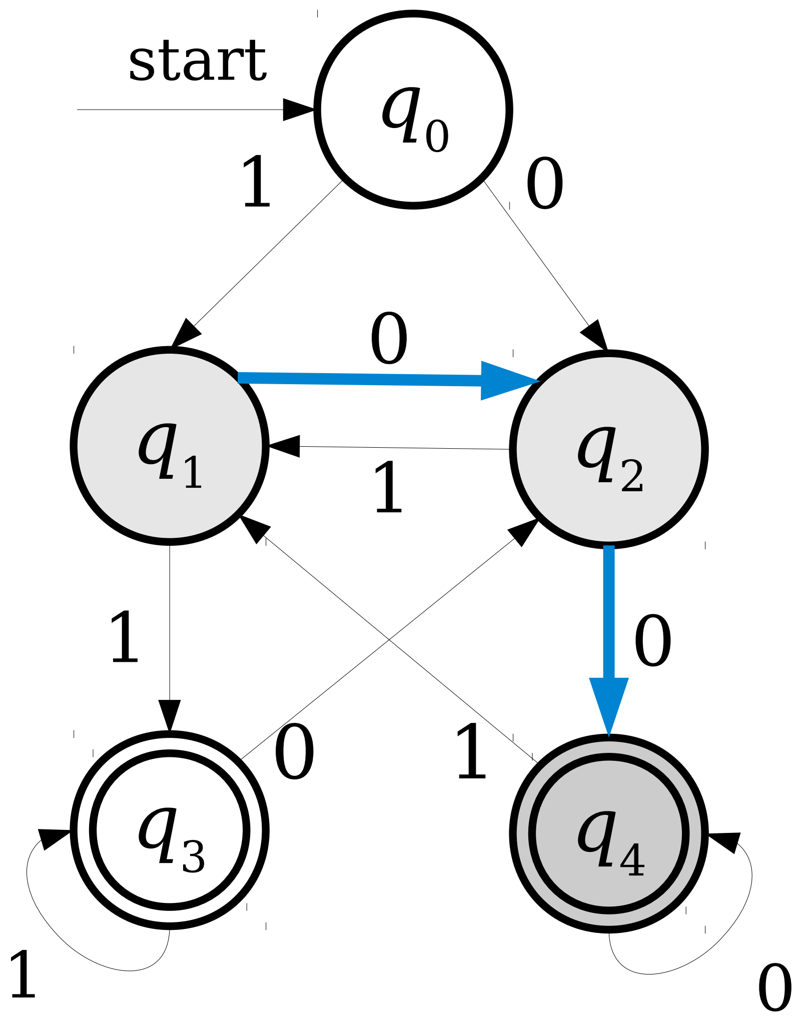
What Does This Accept?



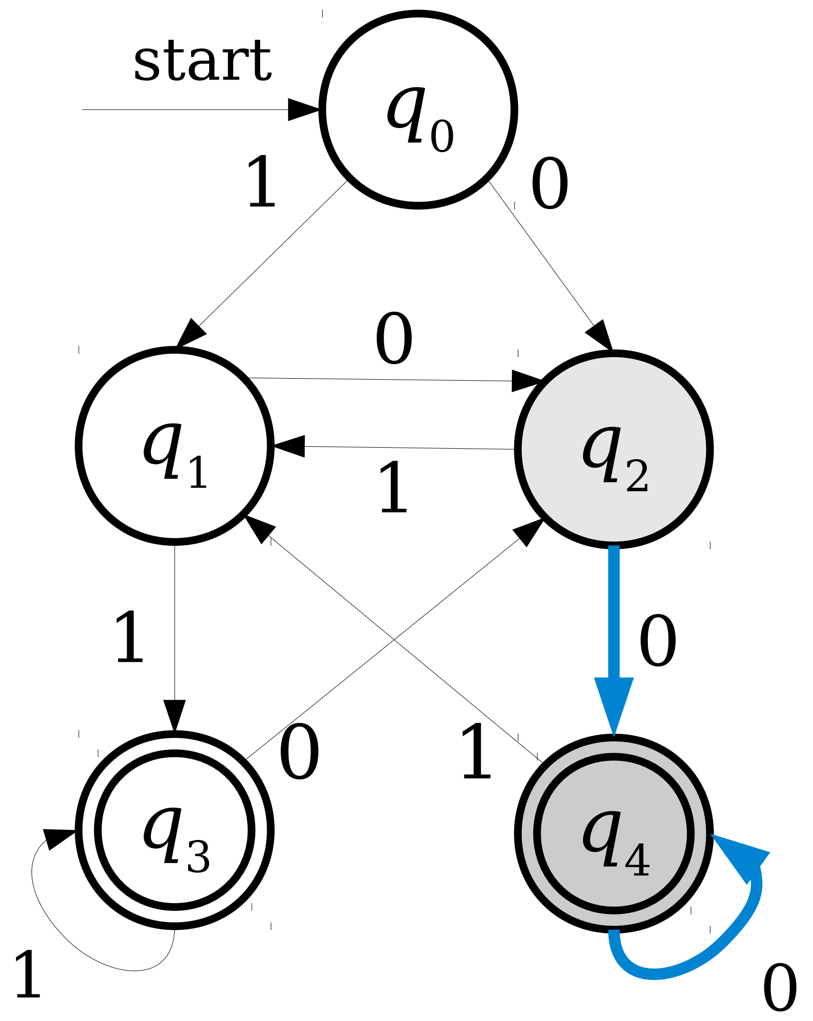
What Does This Accept?



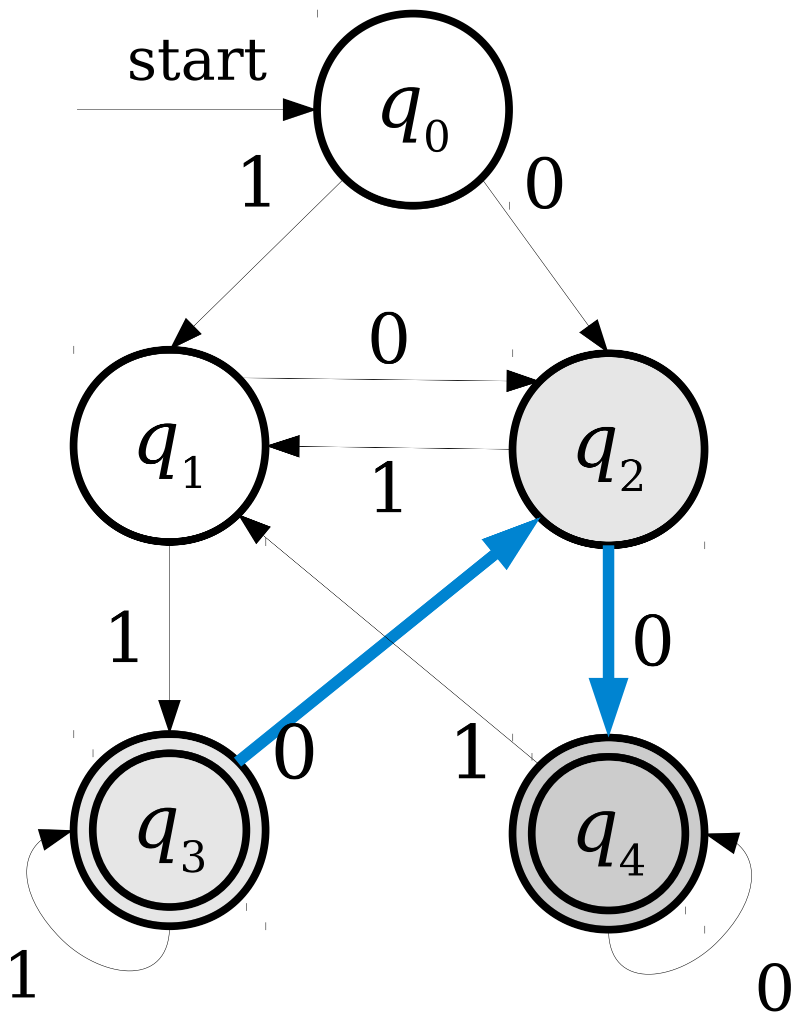
What Does This Accept?



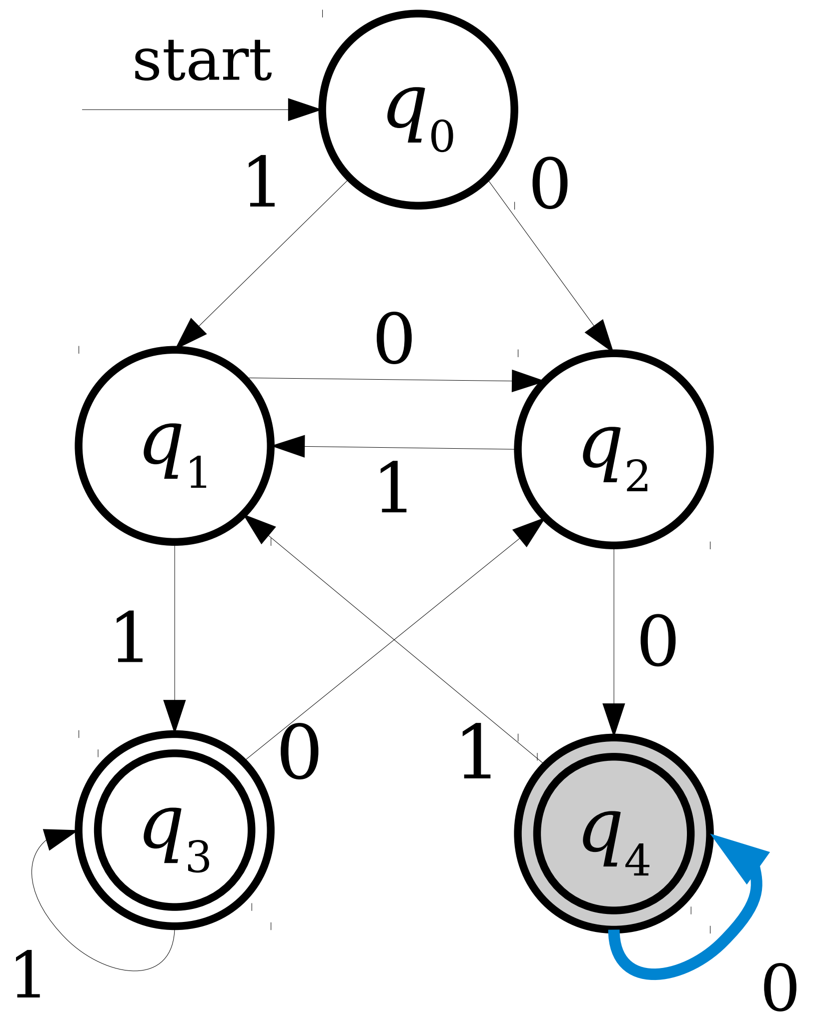
What Does This Accept?



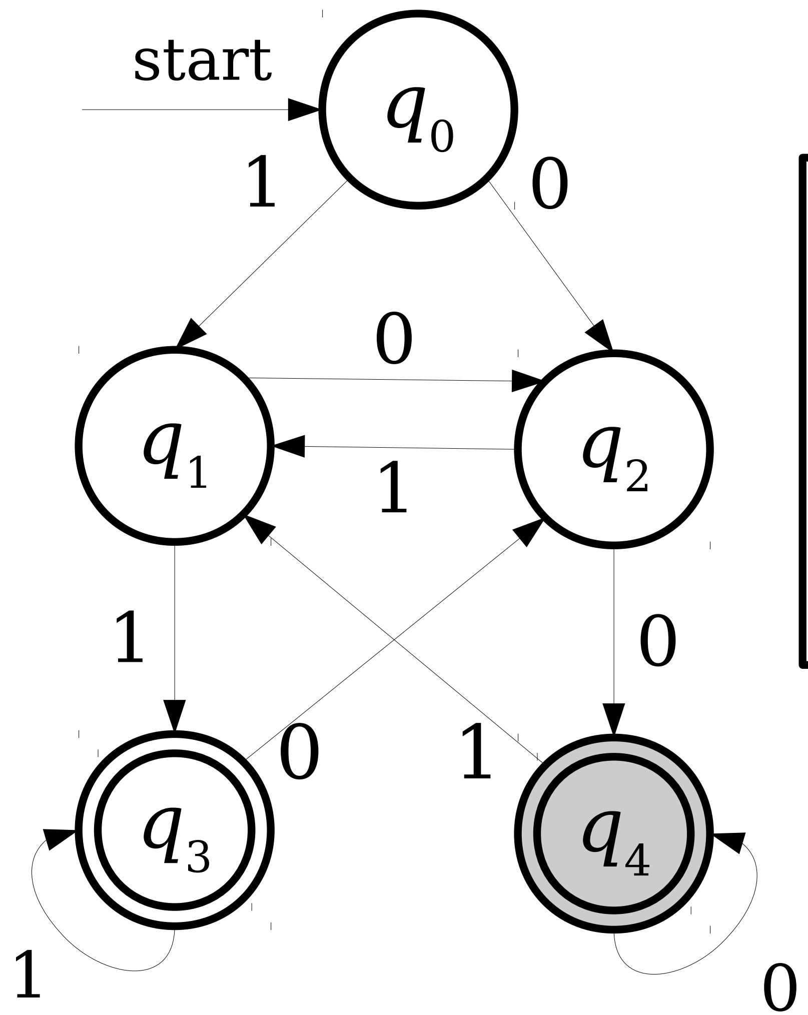
What Does This Accept?



What Does This Accept?

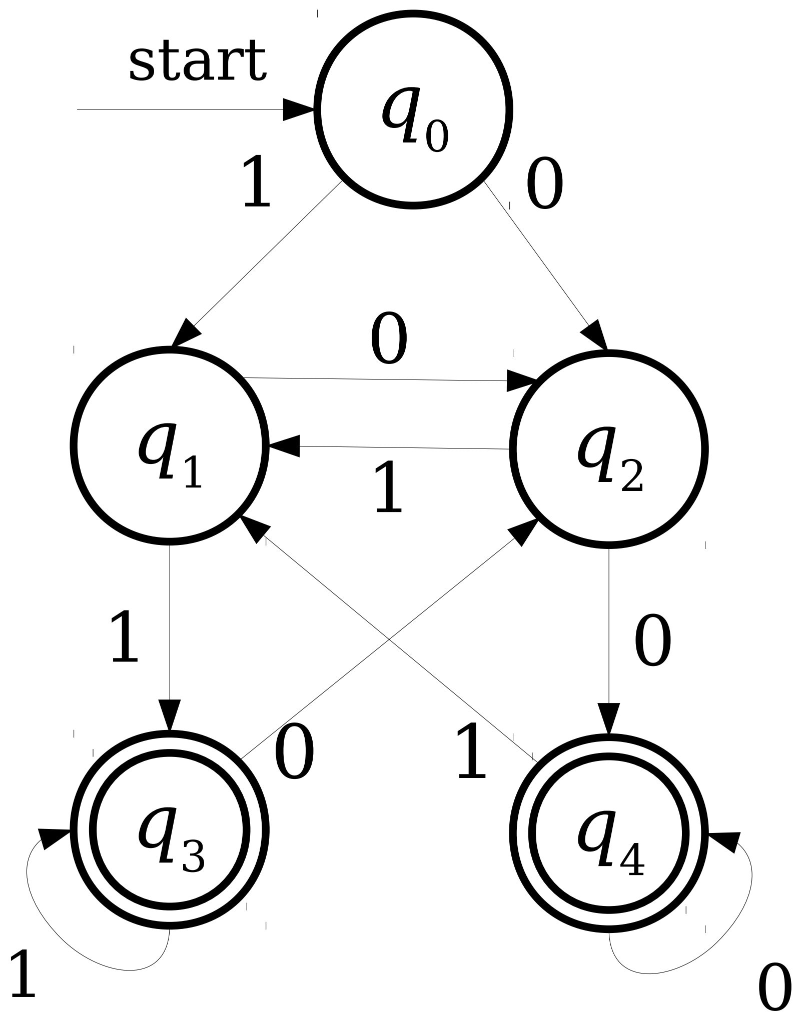


What Does This Accept?

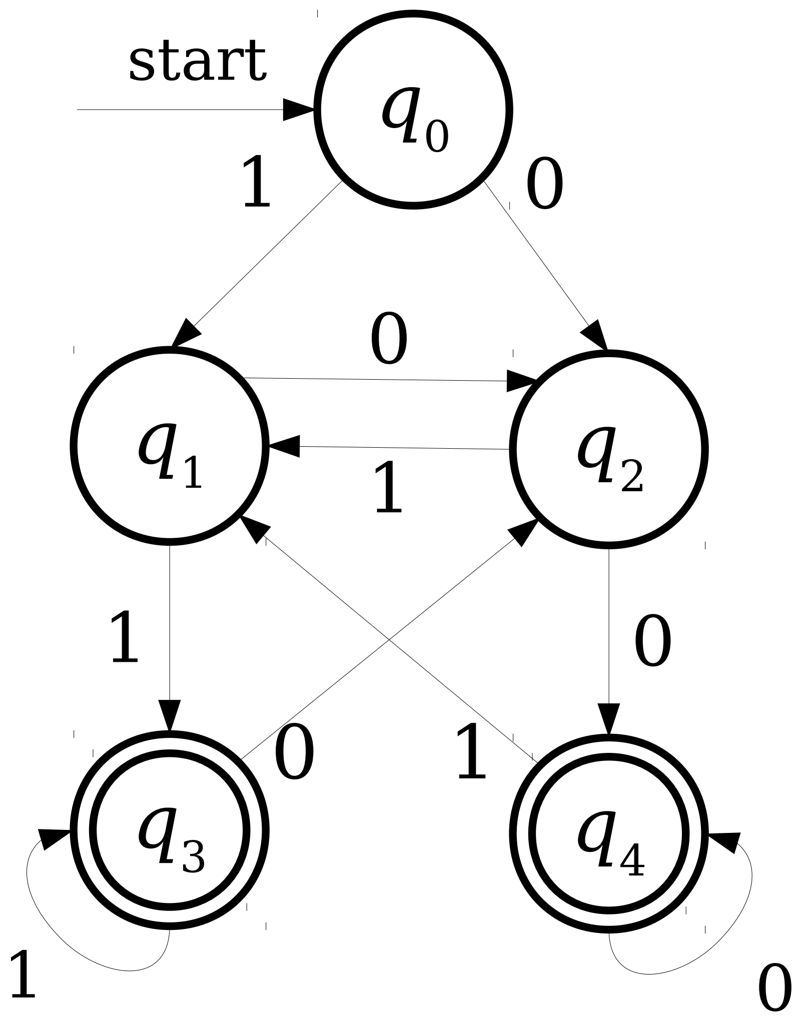


No matter where we start in the automaton, after seeing two 0's, we end up in accepting state q_5.

What Does This Accept?



What Does This Accept?



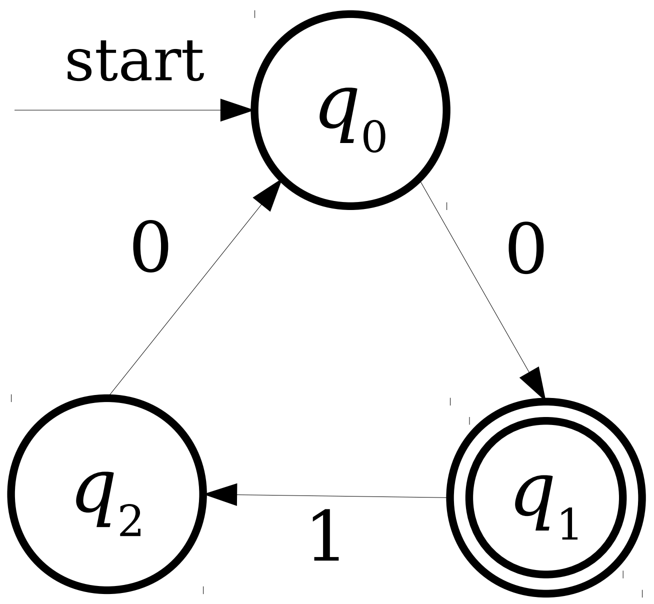
This automaton accepts a string iff the string ends in 00 or 11.

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

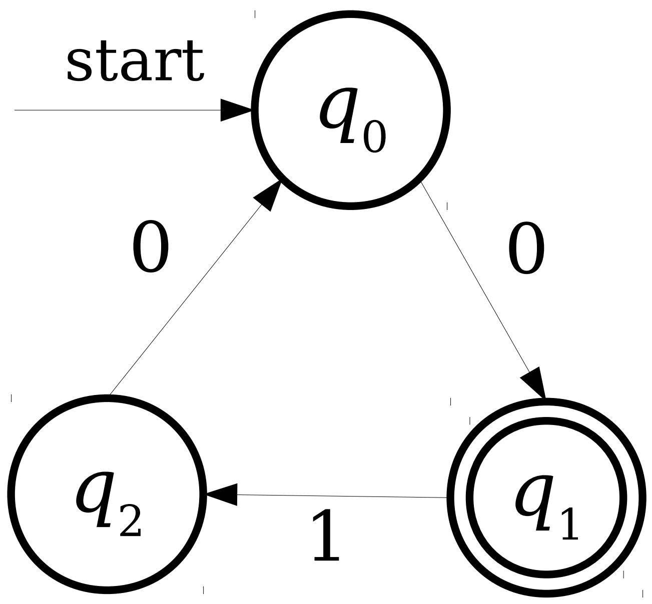
If D is an automaton, we denote the language of D as $\mathscr{L}$**(*D*)**.

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

A Small Problem

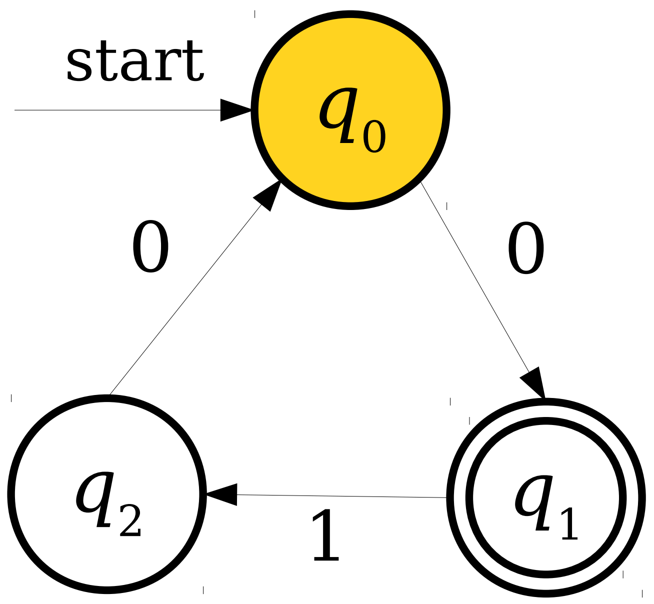


A Small Problem



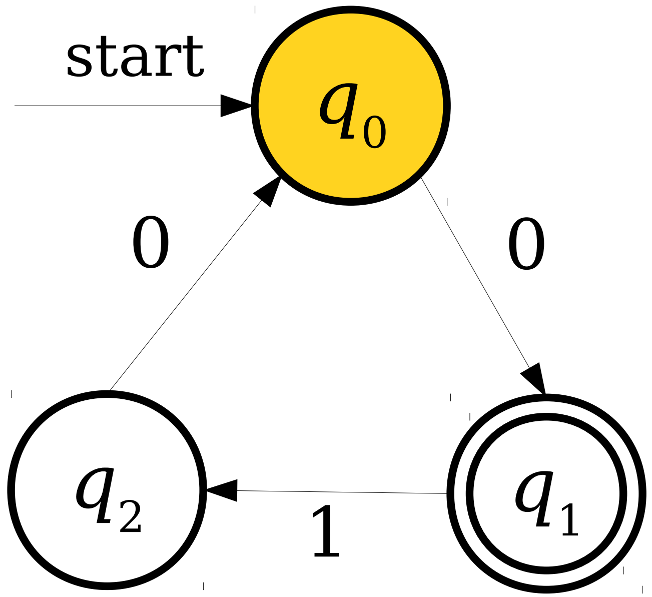
0 1 1 0

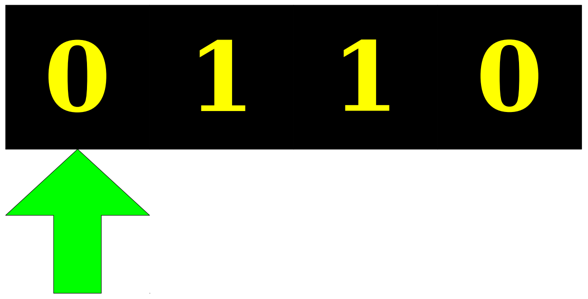
A Small Problem



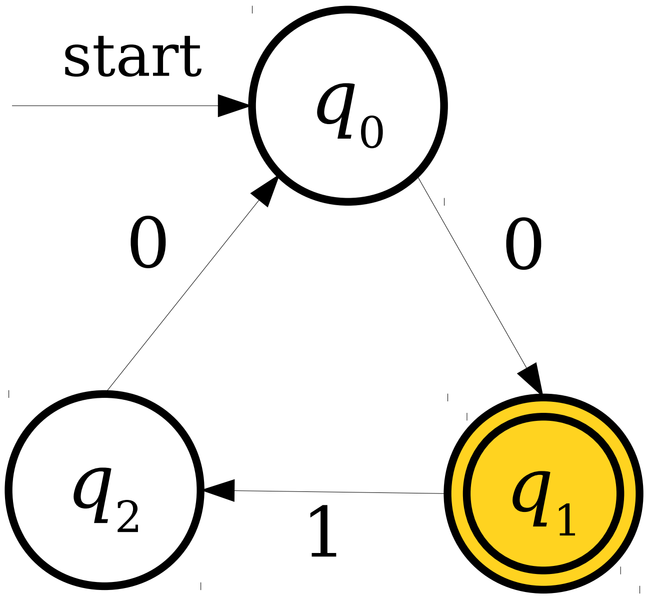
0 1 1 0

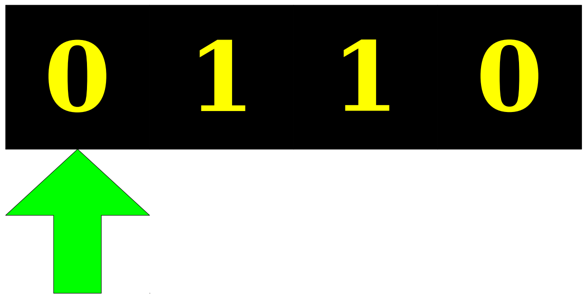
A Small Problem



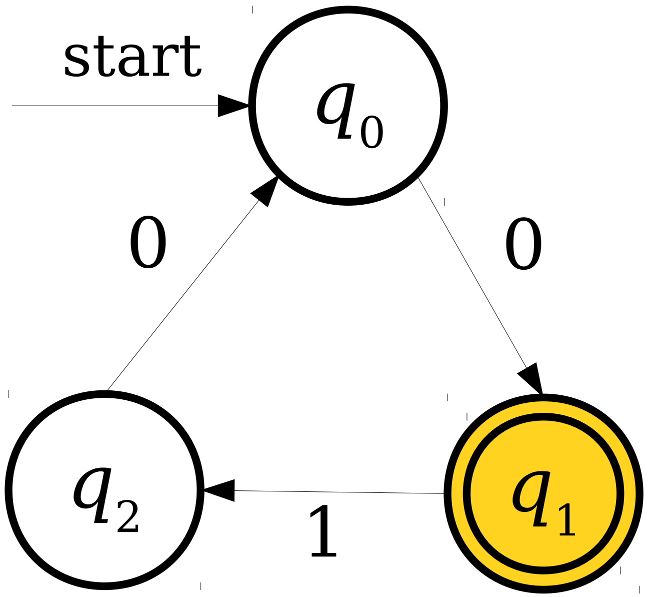


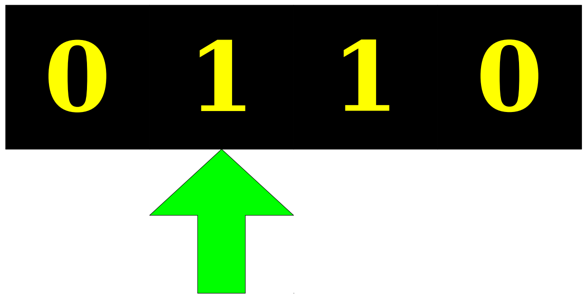
A Small Problem



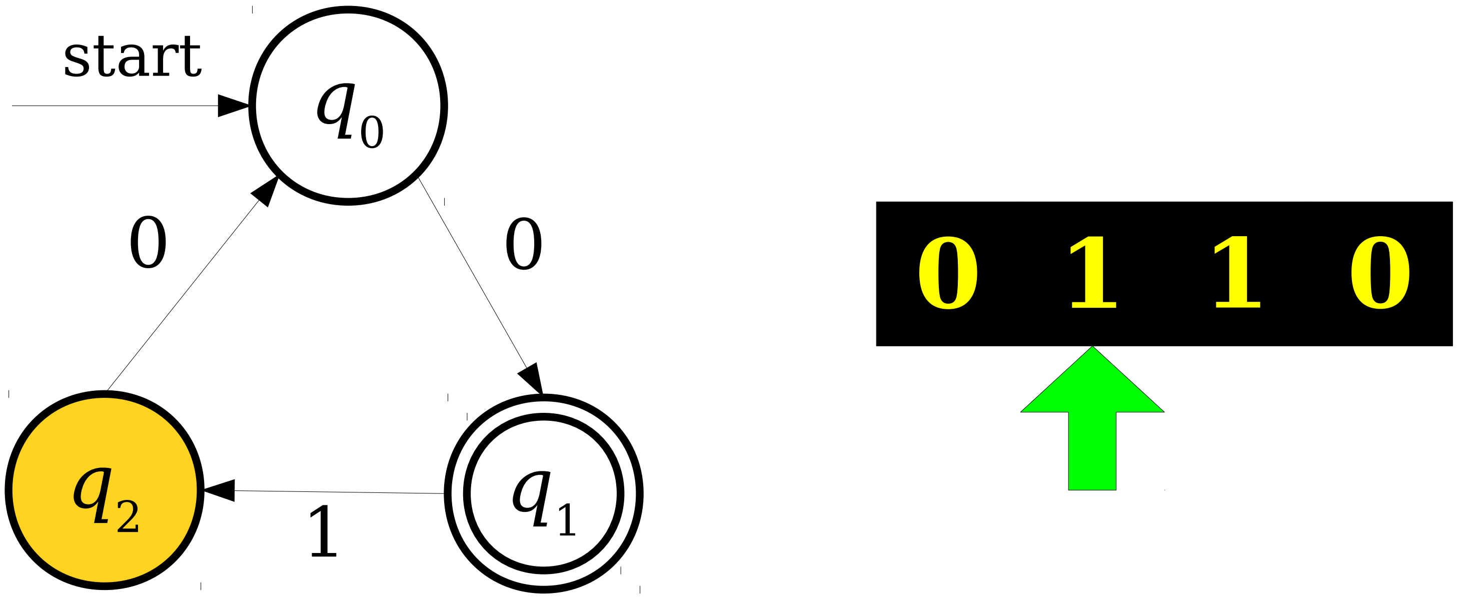


A Small Problem

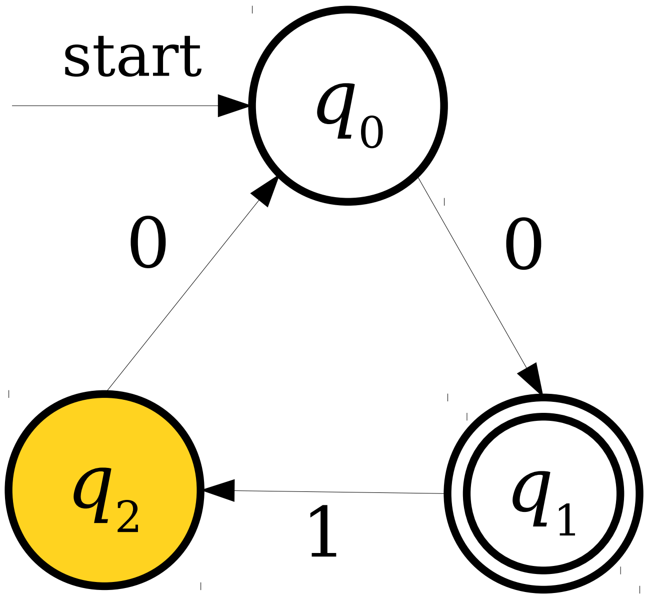


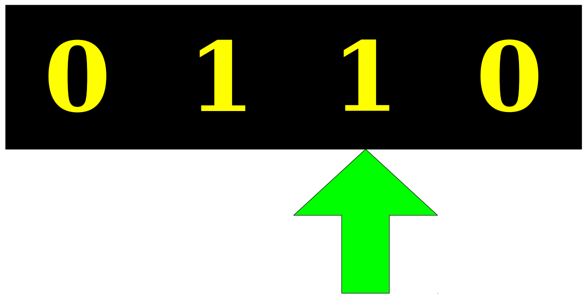


A Small Problem

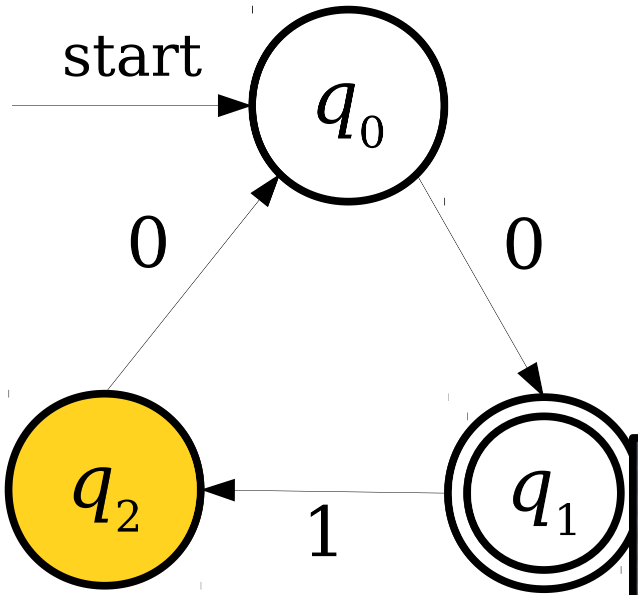


A Small Problem

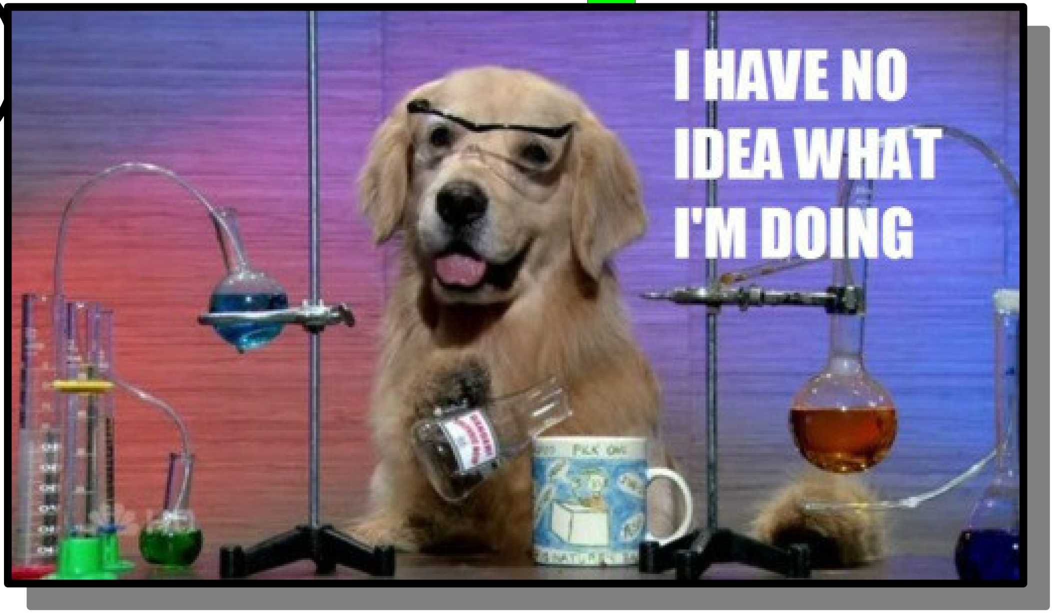




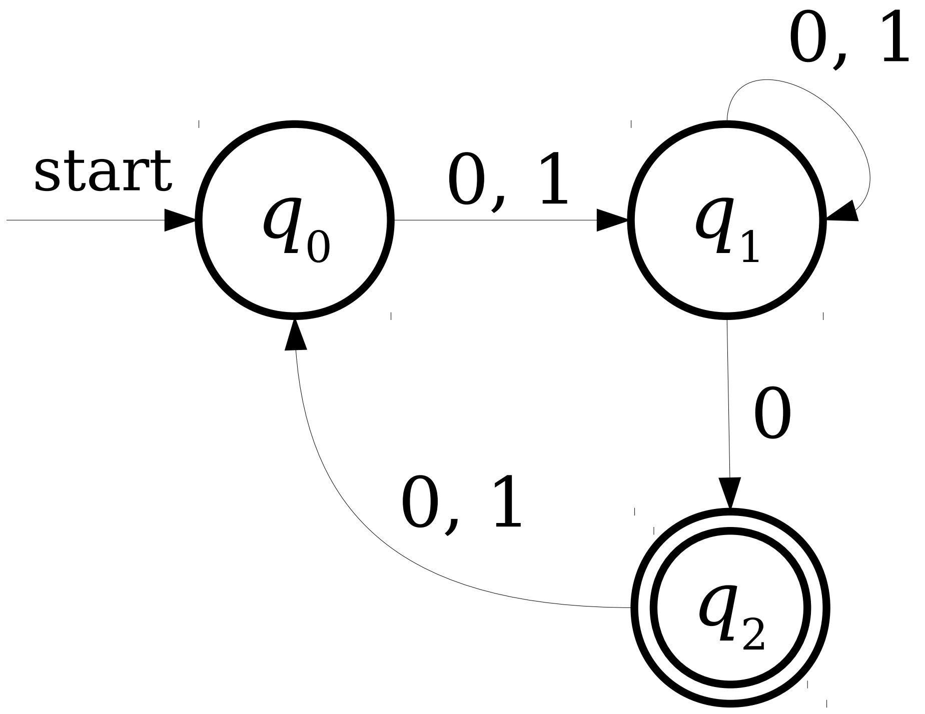
A Small Problem



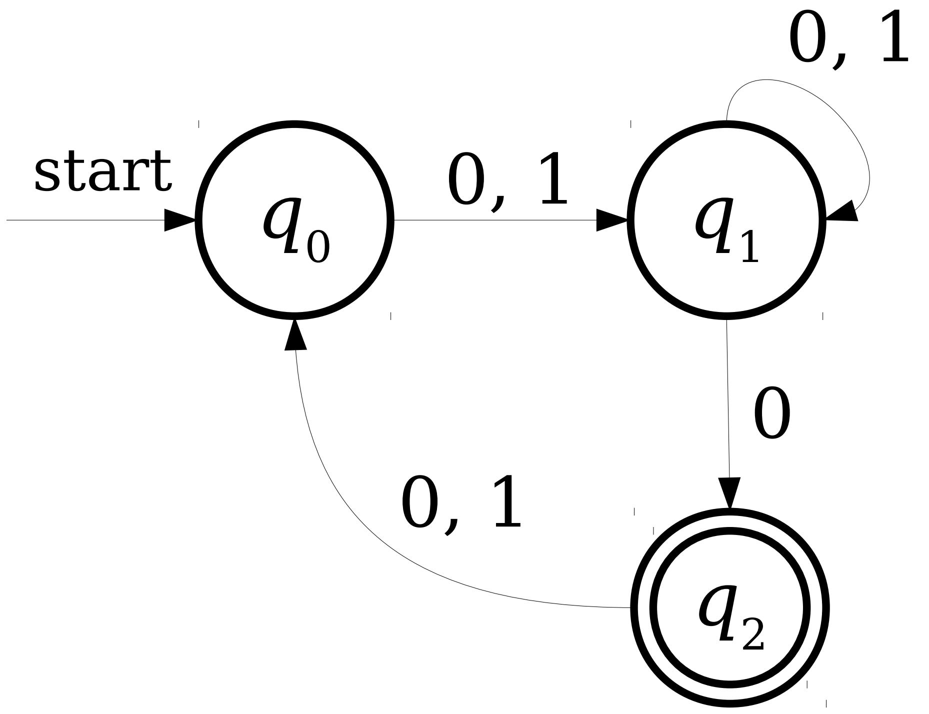
0 1 1 0

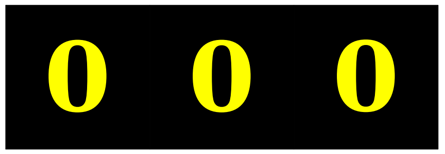


Another Small Problem

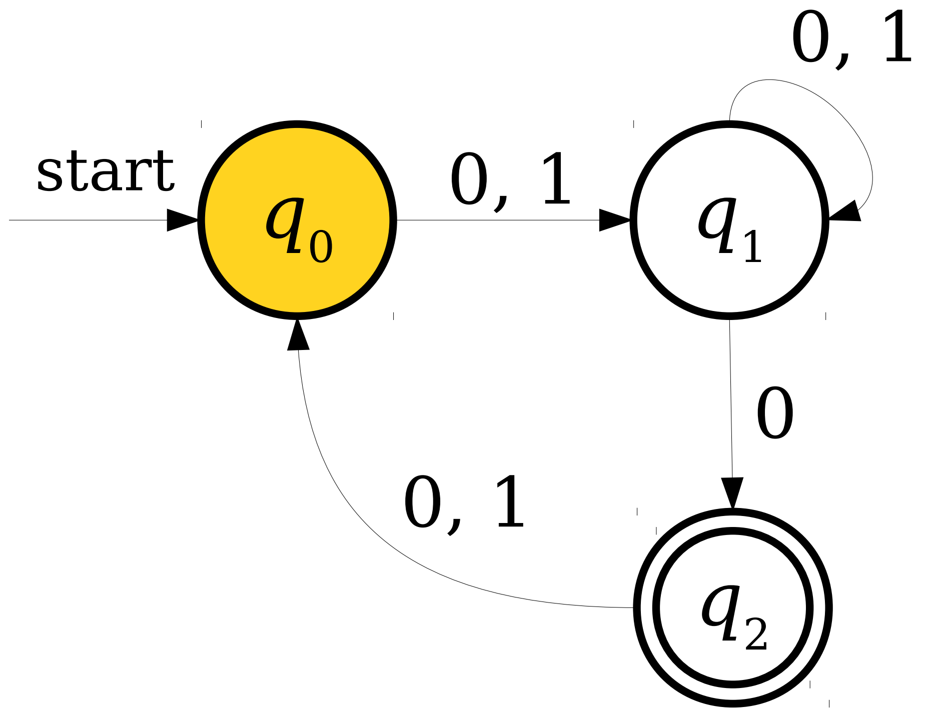


Another Small Problem



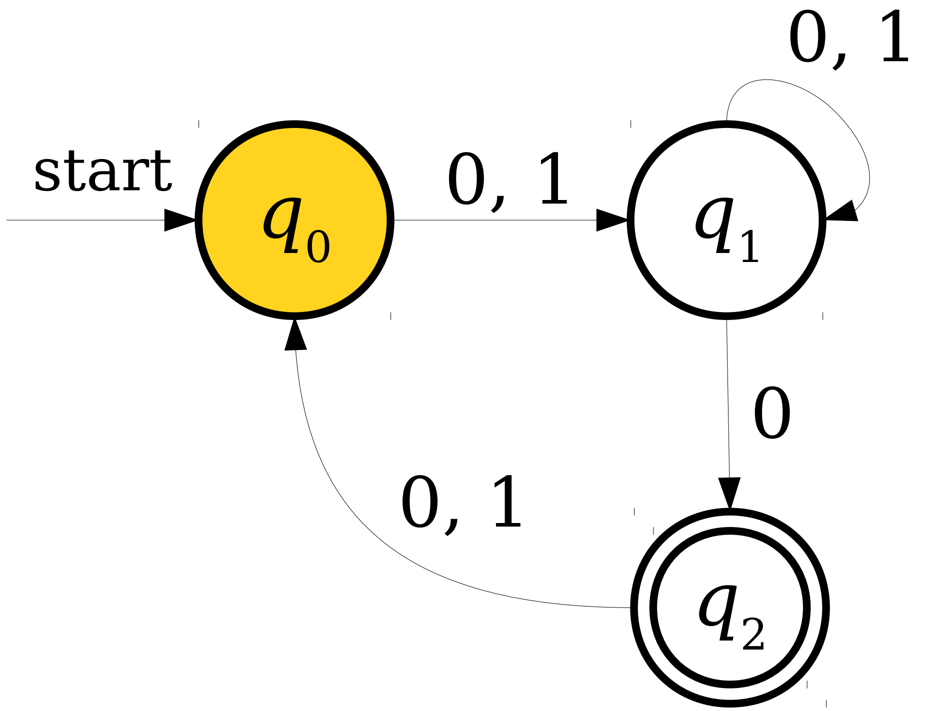


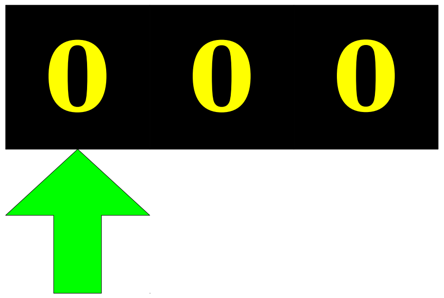
Another Small Problem



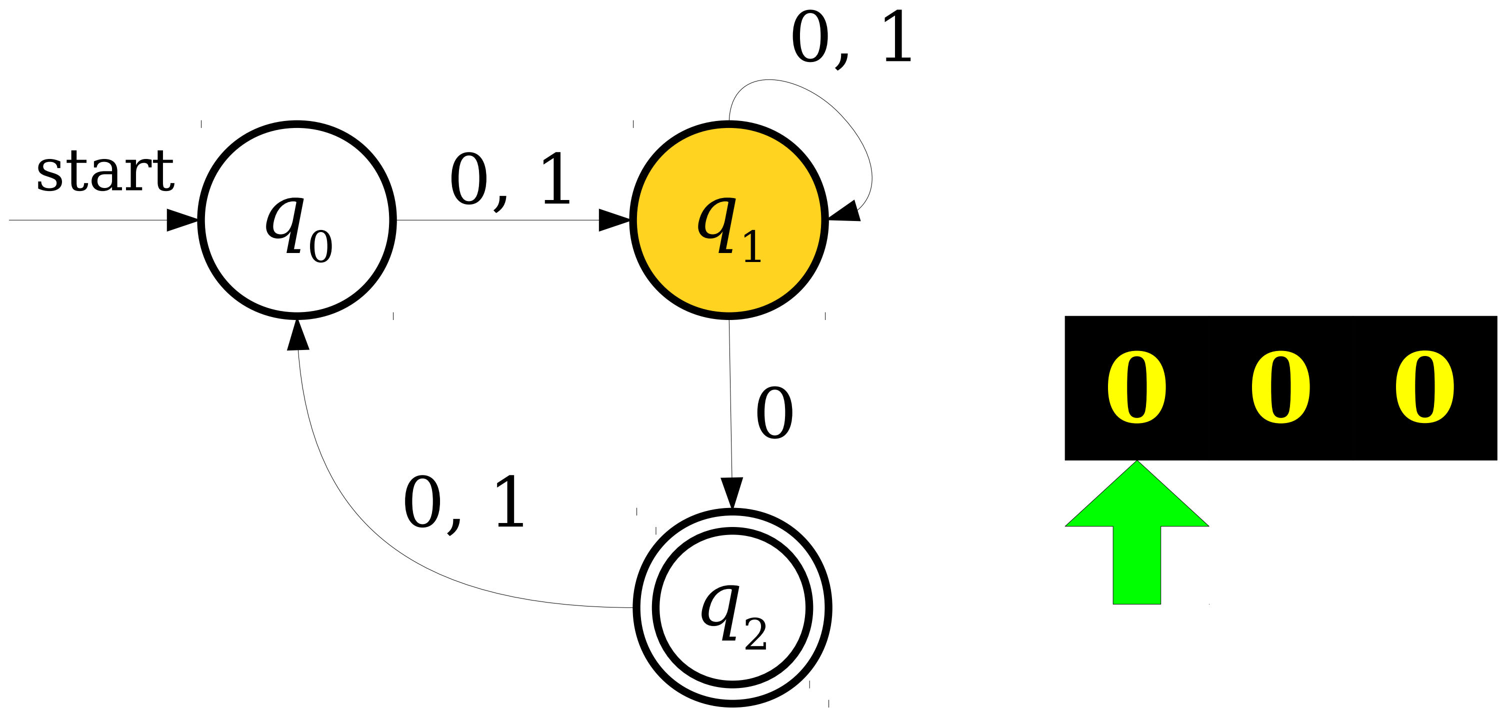
0 0 0

Another Small Problem

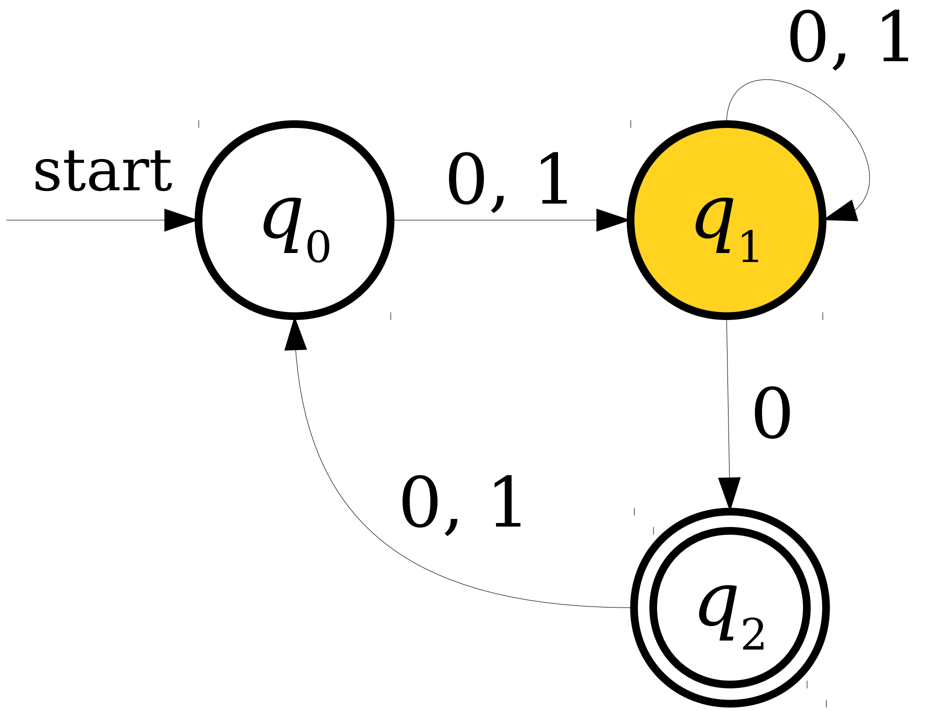


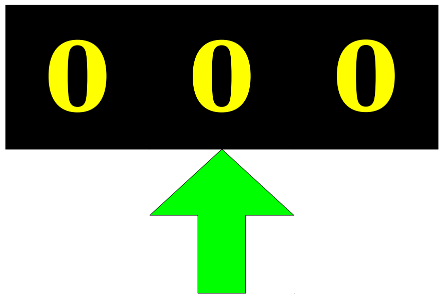


Another Small Problem

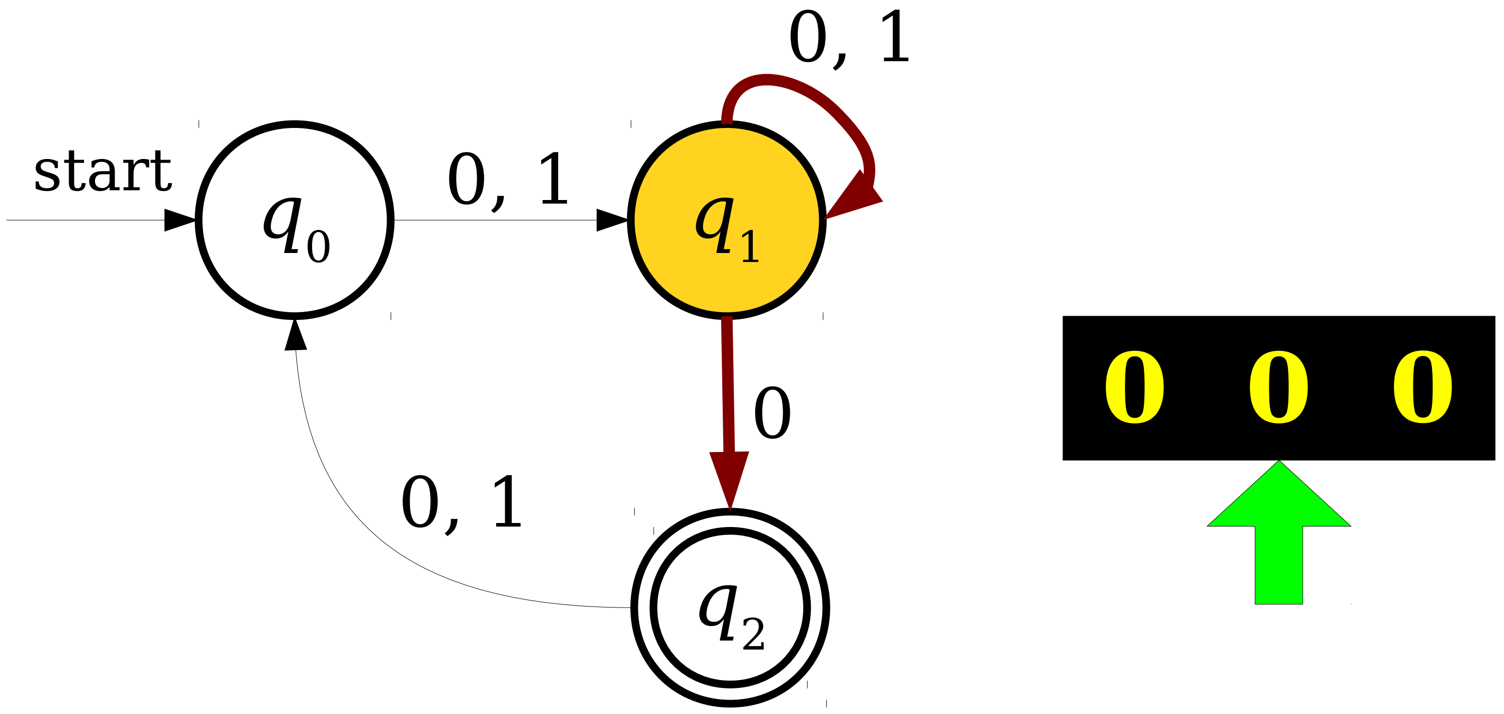


Another Small Problem



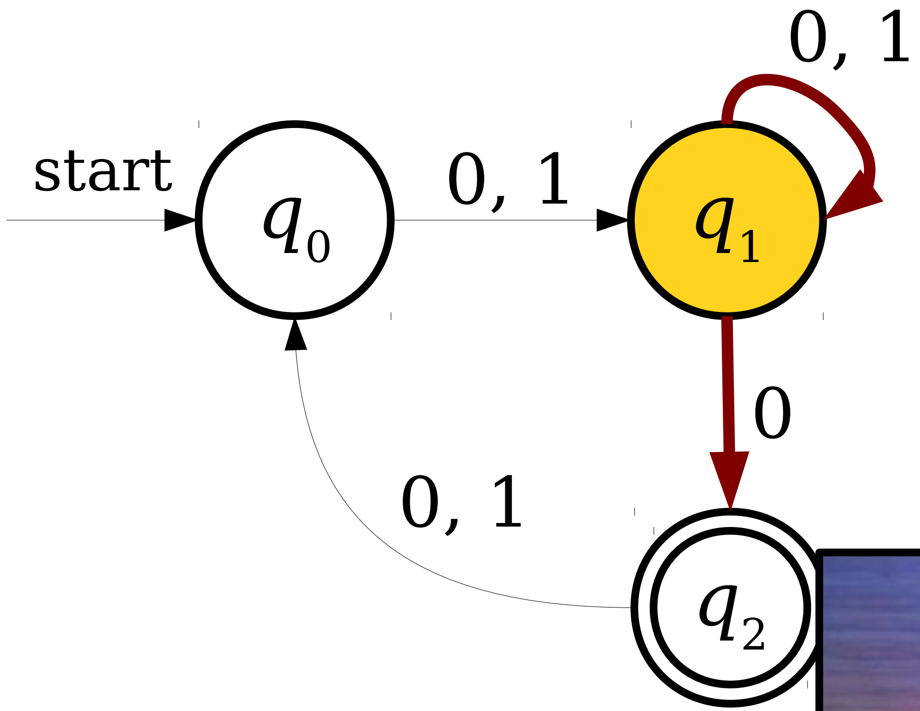


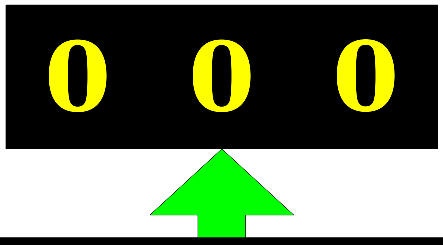
Another Small Problem



Another Small Problem

$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, "{0,\ 1}"] & q_1 \arrow[loop above, "{0,\ 1}"] \arrow[d, "0"] \\
& & q_2 \arrow[ul, bend left, "{0,\ 1}"]
\end{tikzcd}$$





The Need for Formalism

- In order to reason about the limits of what finite automata can and cannot do, we need to formally specify their behavior in *all* cases.
- All of the following need to be defined or disallowed:
  - What happens if there is no transition out of a state on some input?
  - What happens if there are *multiple* transitions out of a state on some input?

DFAs

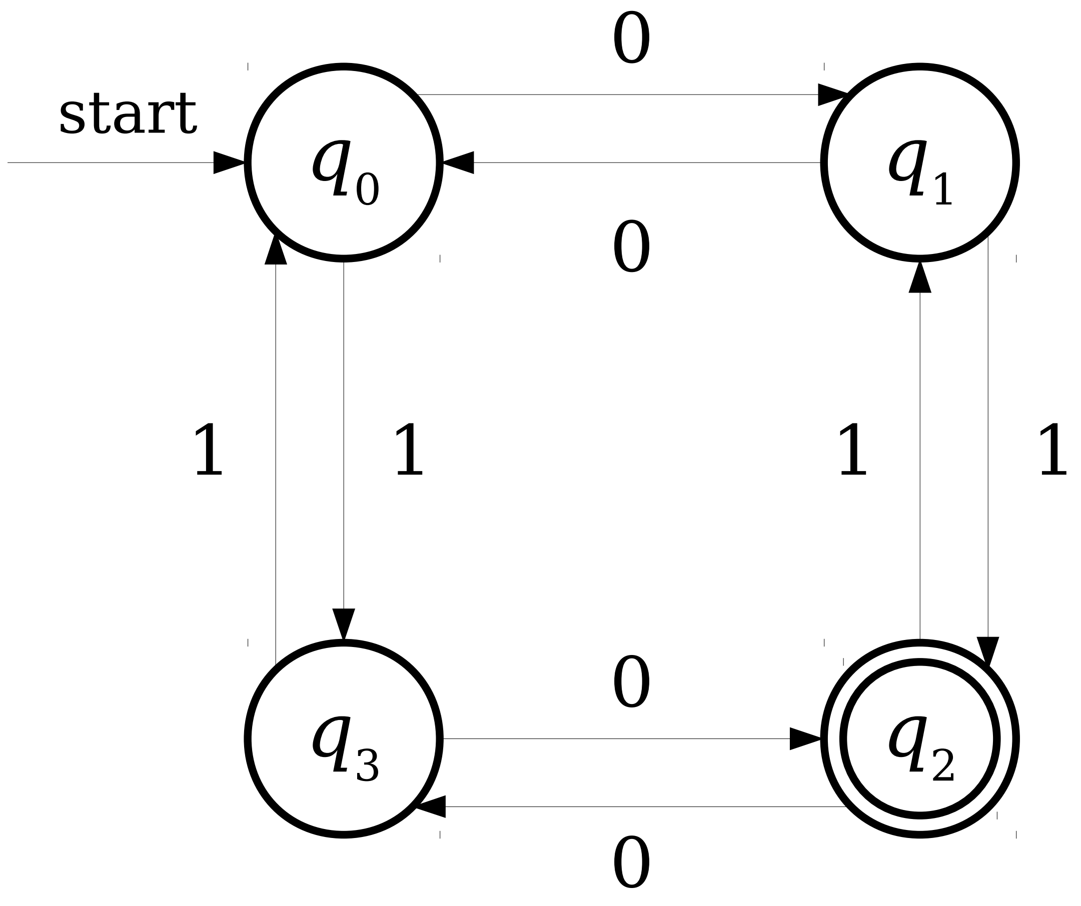
- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- DFAs are the simplest type of automaton that we will see in this course.

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.

Is this a DFA over {0, 1}?

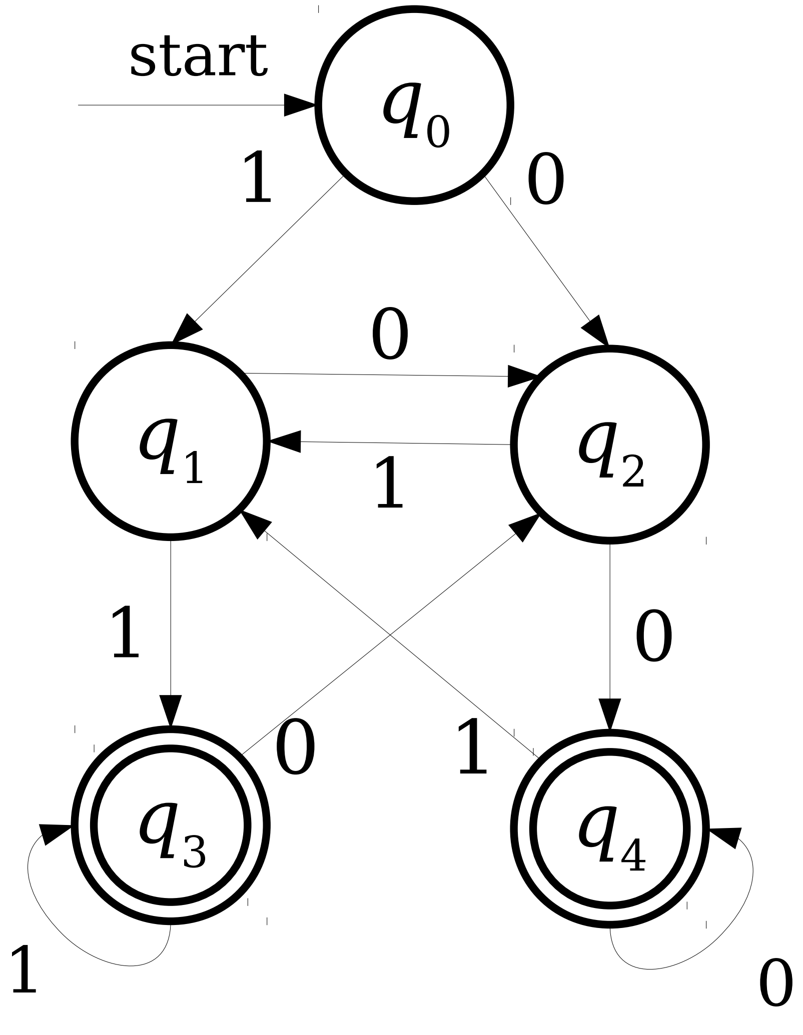
Is this a DFA over {0, 1}?



$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

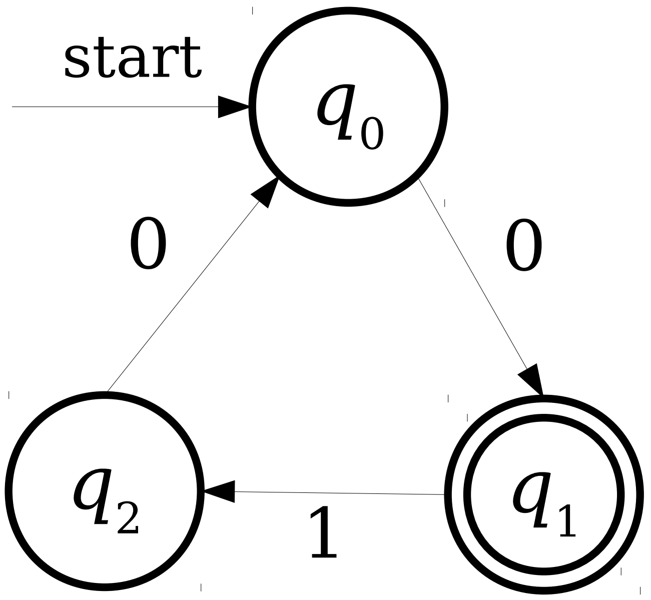
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?

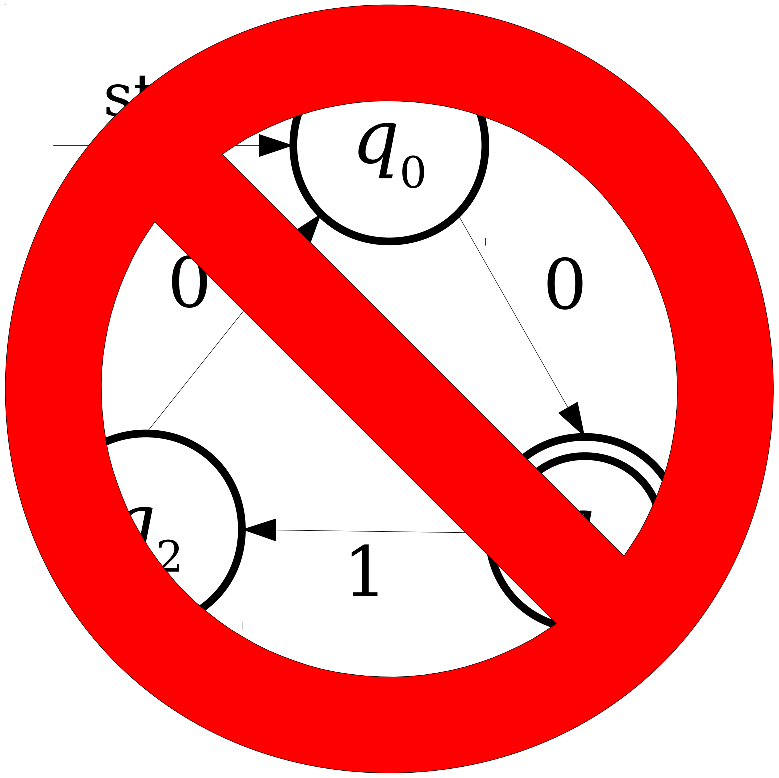


Is this a DFA over {0, 1}?

Is this a DFA over {0, 1}?

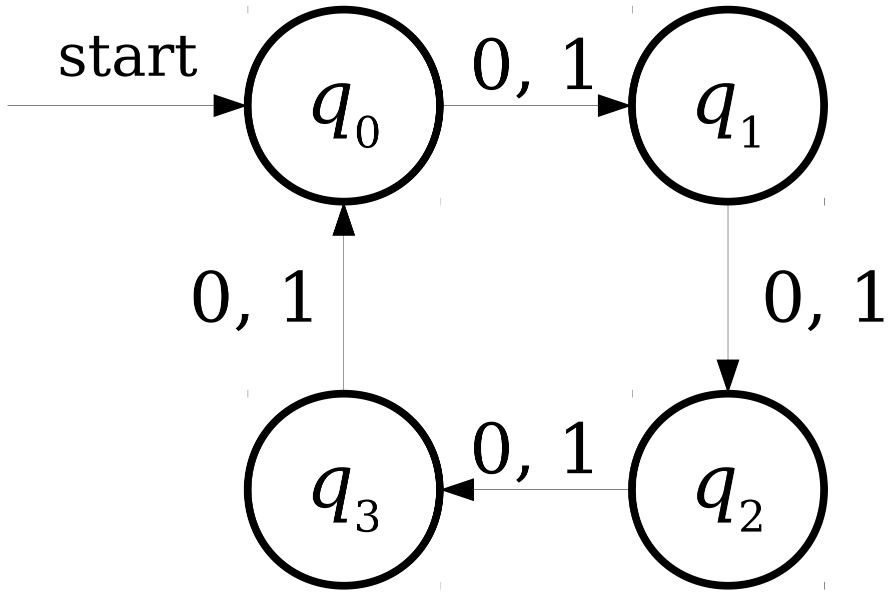


Is this a DFA over {0, 1}?



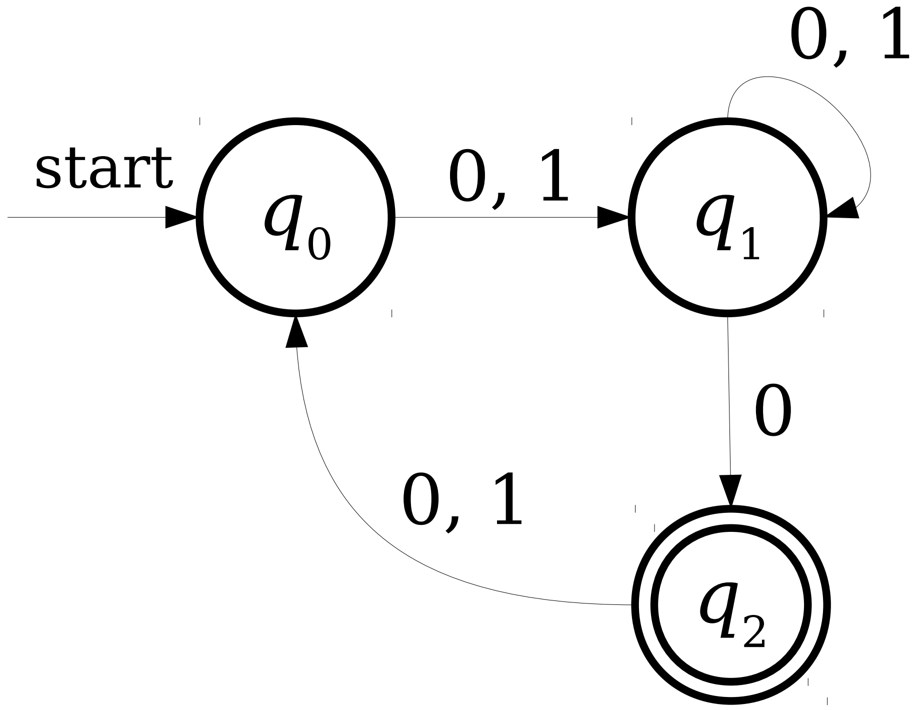
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?

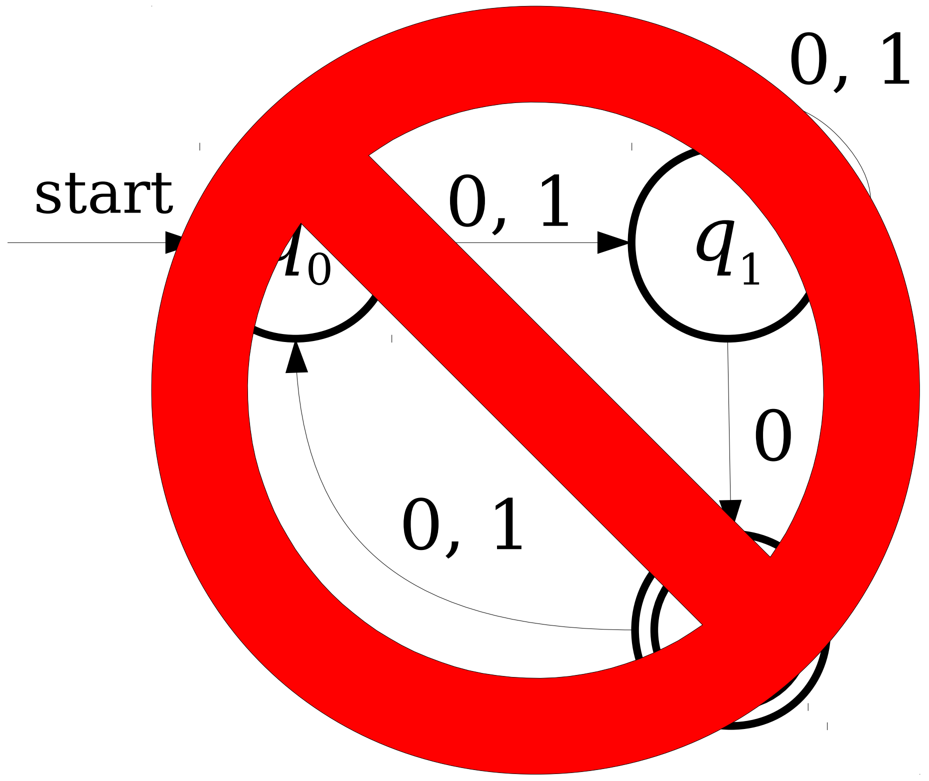


Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?



Is this a DFA over {0, 1}?



Is this a DFA?

Is this a DFA?

Is this a DFA?

Drinking **F**amily of **A**ardvarks

Designing DFAs

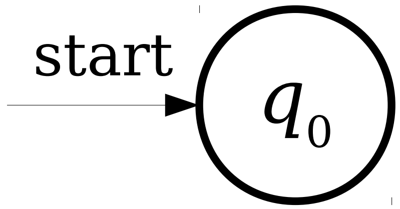
- At each point in its execution, the DFA can only remember what state it is in.
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only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

Recognizing Languages with DFAs

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

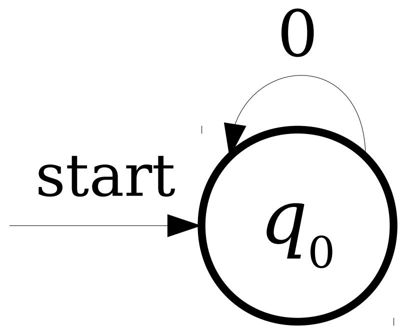
Recognizing Languages with DFAs

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



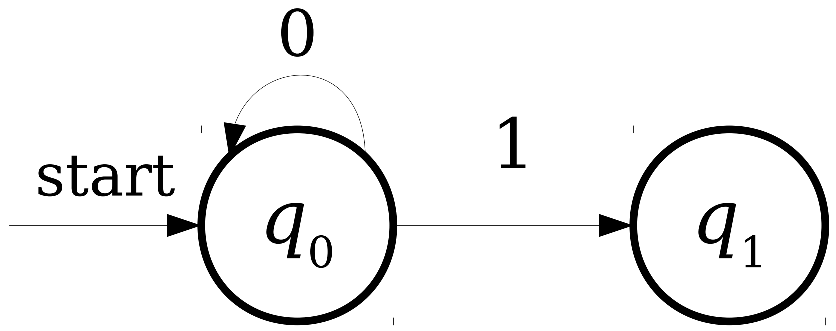
Recognizing Languages with DFAs

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



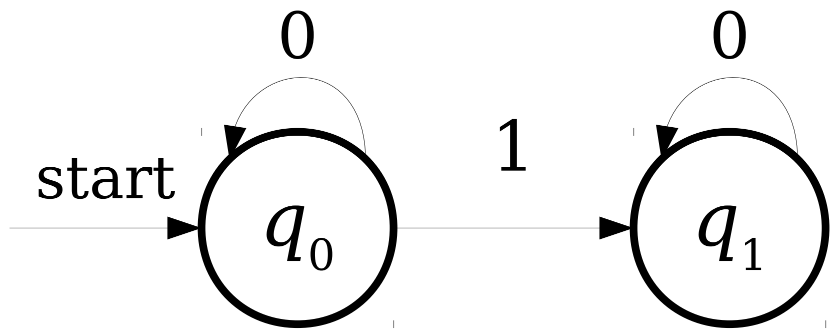
Recognizing Languages with DFAs

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



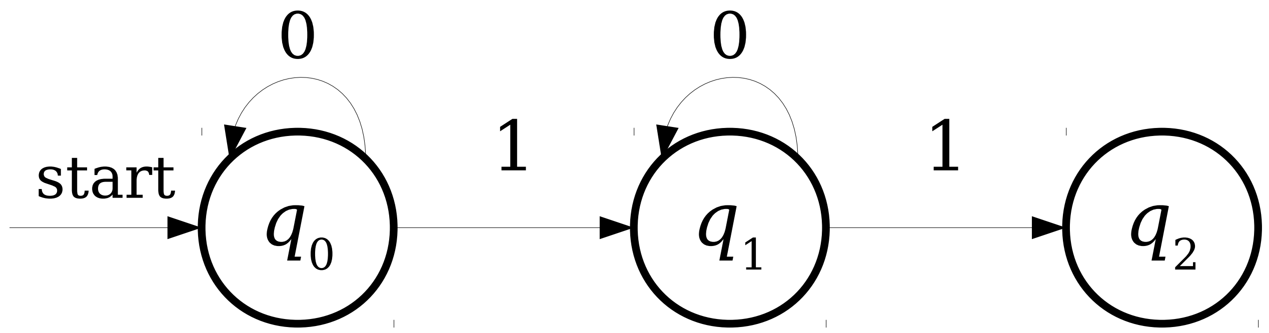
Recognizing Languages with DFAs

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


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[loop above, "0"] \arrow[r, "1"] & q_1 \arrow[loop above, "0"]
\end{tikzcd}$$

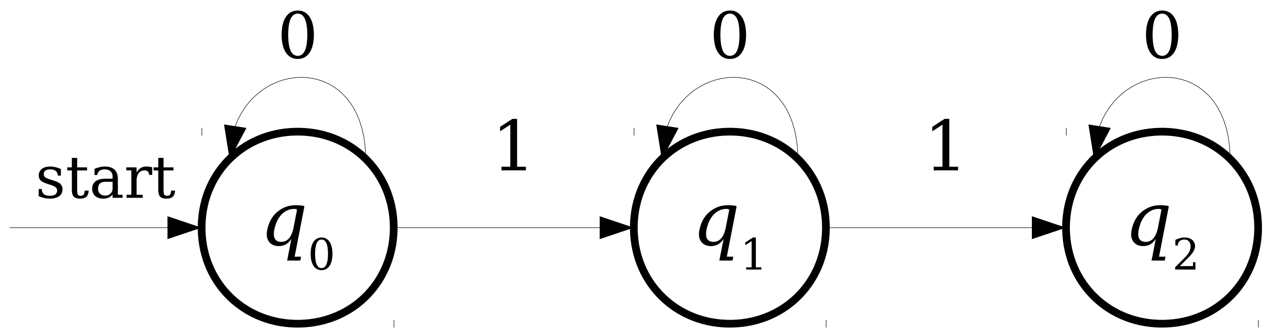
Recognizing Languages with DFAs

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



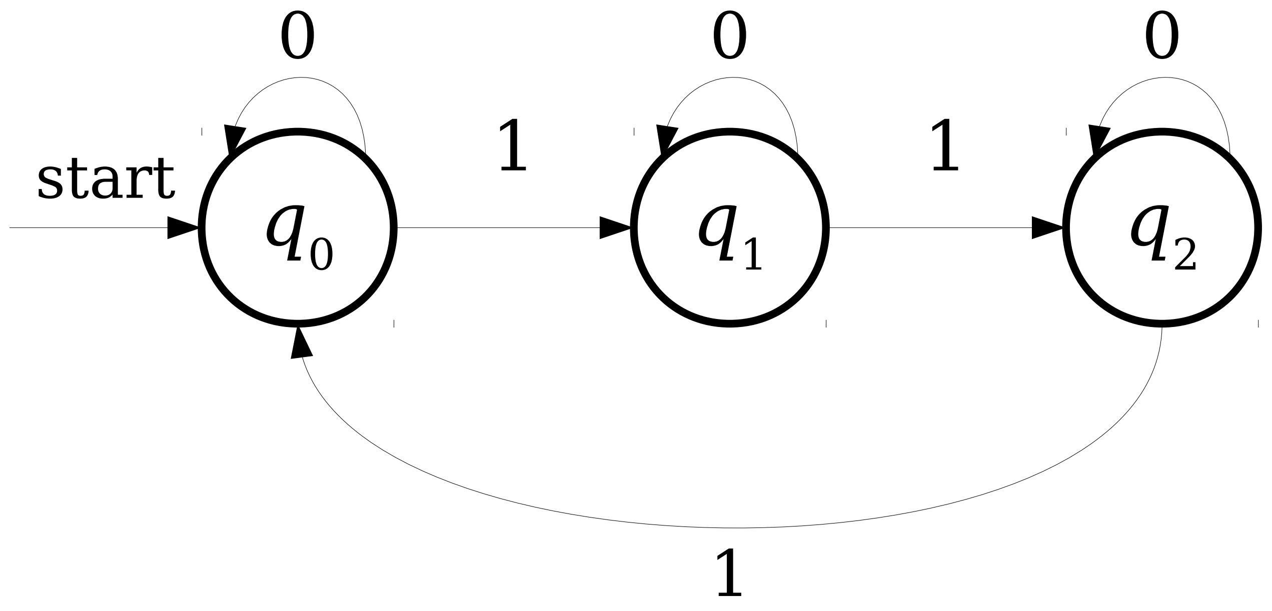
Recognizing Languages with DFAs

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


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, "1"] \arrow[loop above, "0"] & q_1 \arrow[r, "1"] \arrow[loop above, "0"] & q_2 \arrow[loop above, "0"]
\end{tikzcd}$$

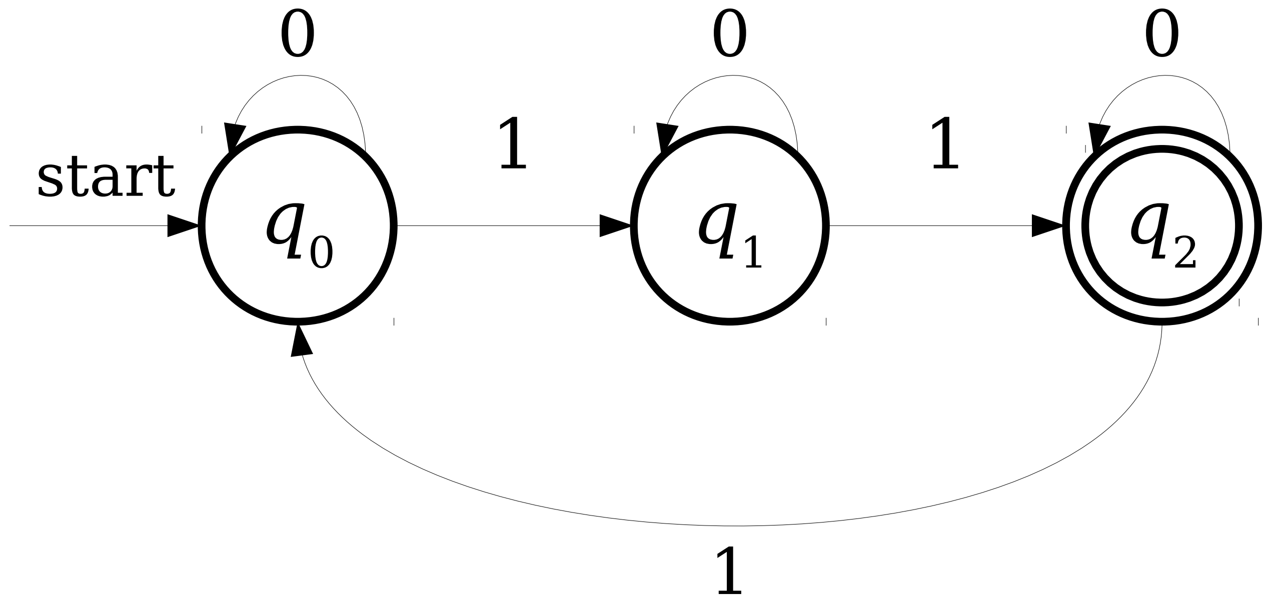
Recognizing Languages with DFAs

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



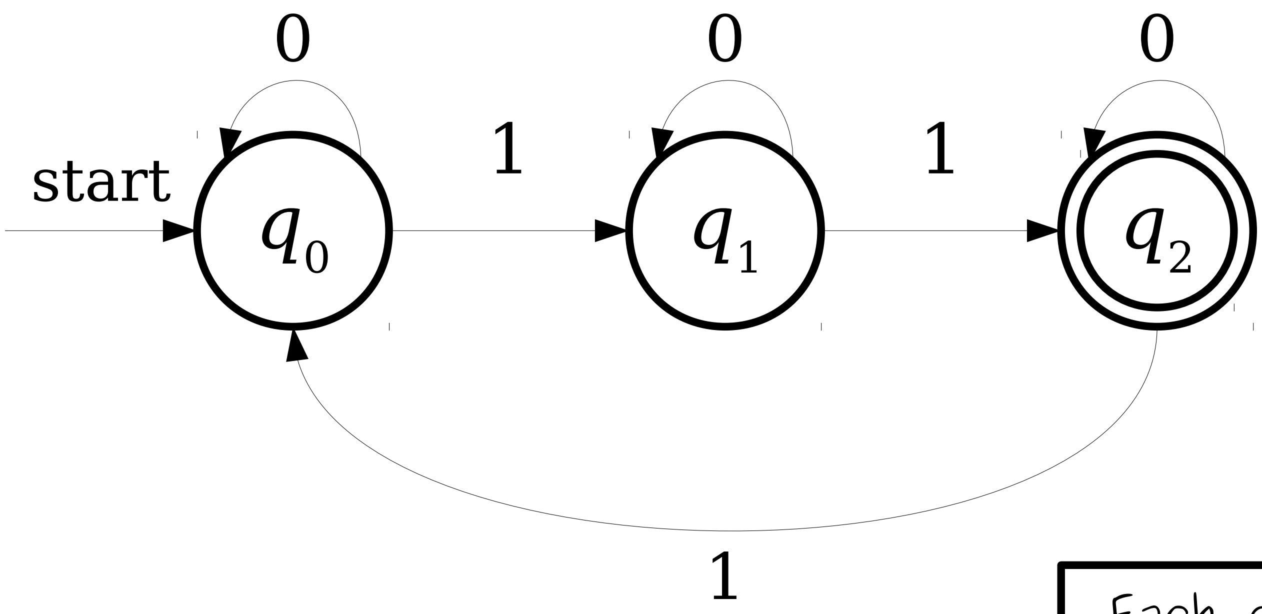
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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


$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[loop above, "0"] \arrow[r, "1"] & q_1 \arrow[loop above, "0"] \arrow[r, "1"] & \boxed{q_2} \arrow[loop above, "0"] \arrow[ll, bend left, "1"]
\end{tikzcd}$$

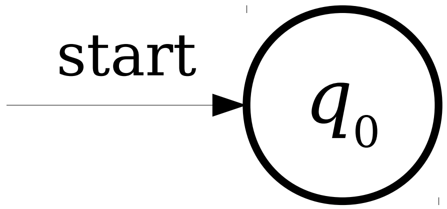
Each state remembers the remainder of the number of 1's seen so far modulo three.

Recognizing Languages with DFAs

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

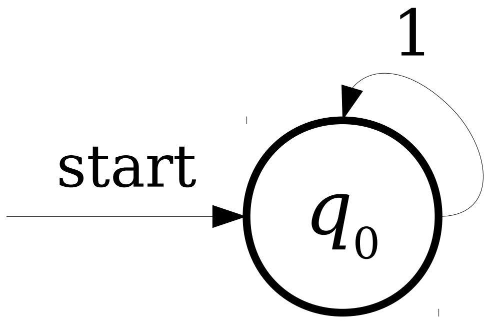
Recognizing Languages with DFAs

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



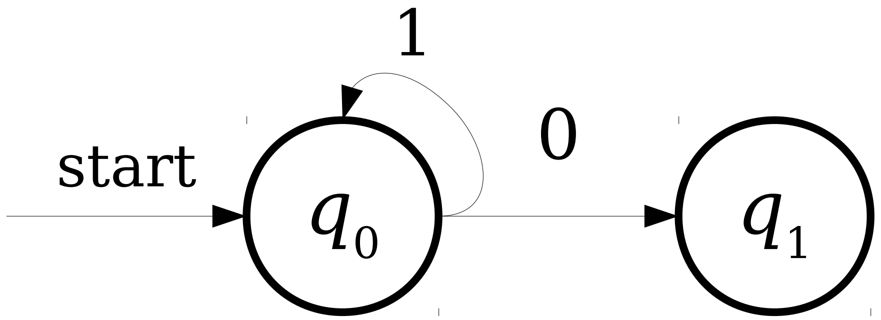
Recognizing Languages with DFAs

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



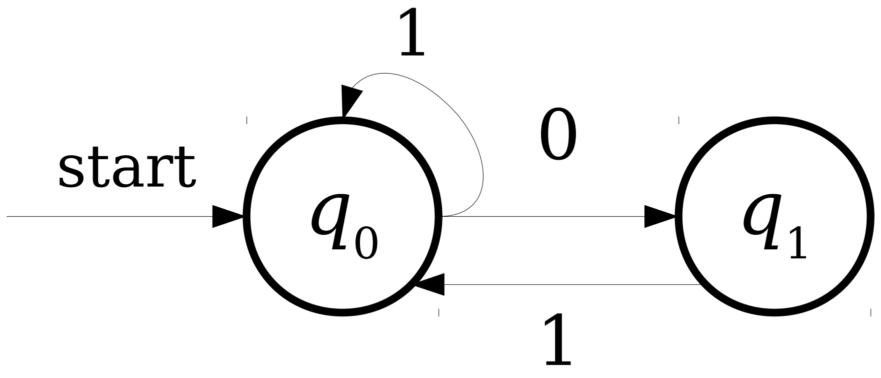
Recognizing Languages with DFAs

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



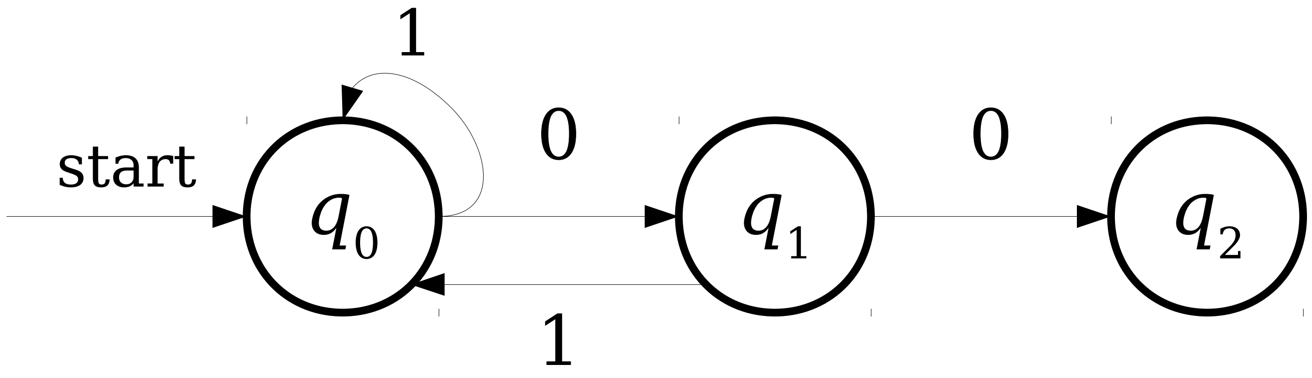
Recognizing Languages with DFAs

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



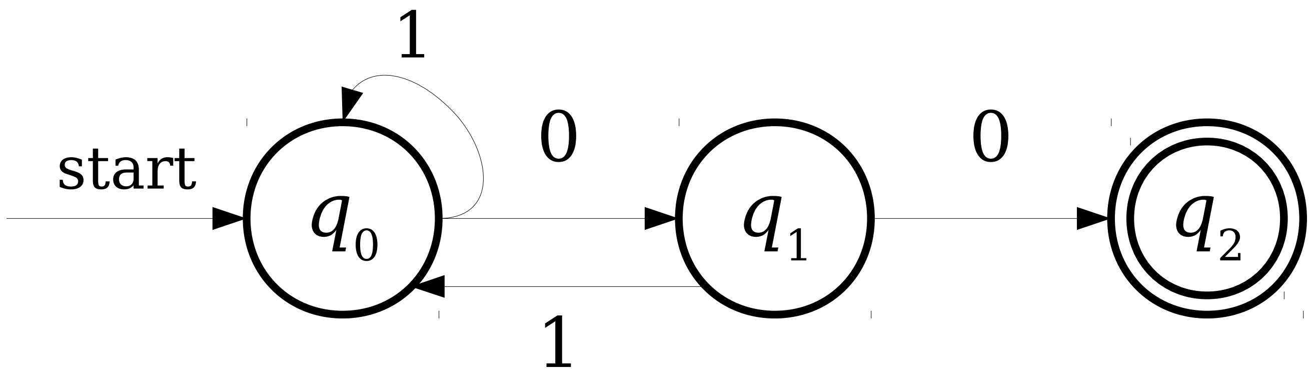
Recognizing Languages with DFAs

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



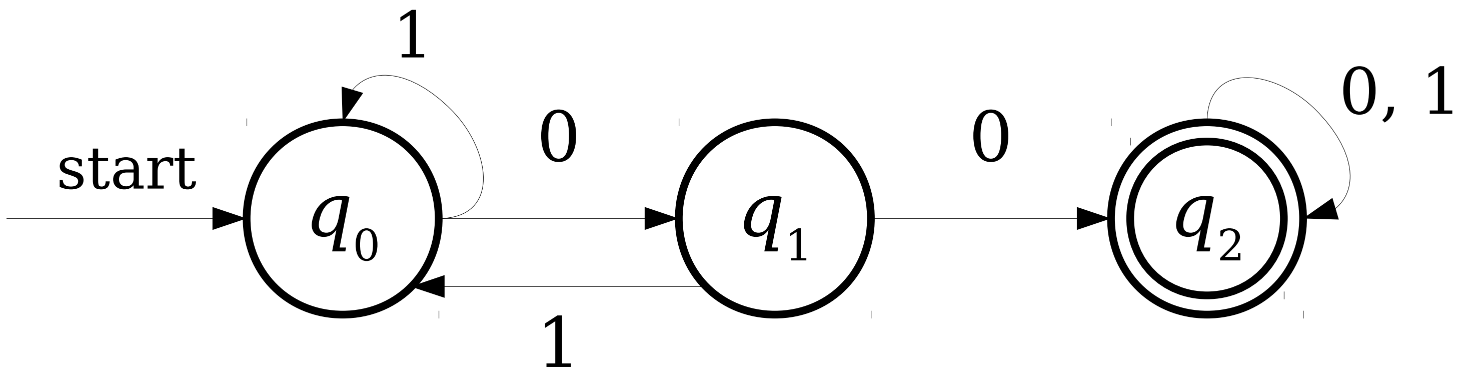
Recognizing Languages with DFAs

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



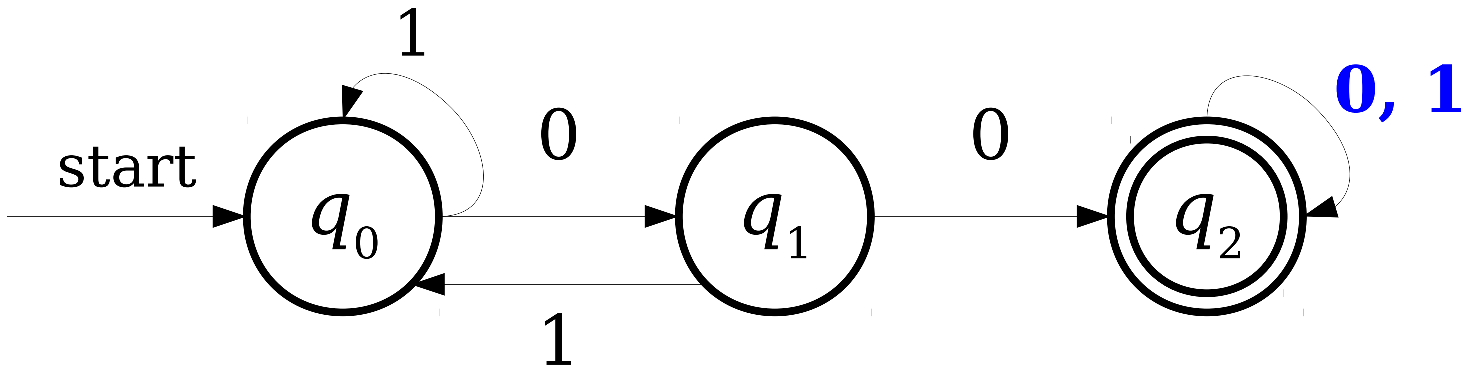
Recognizing Languages with DFAs

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



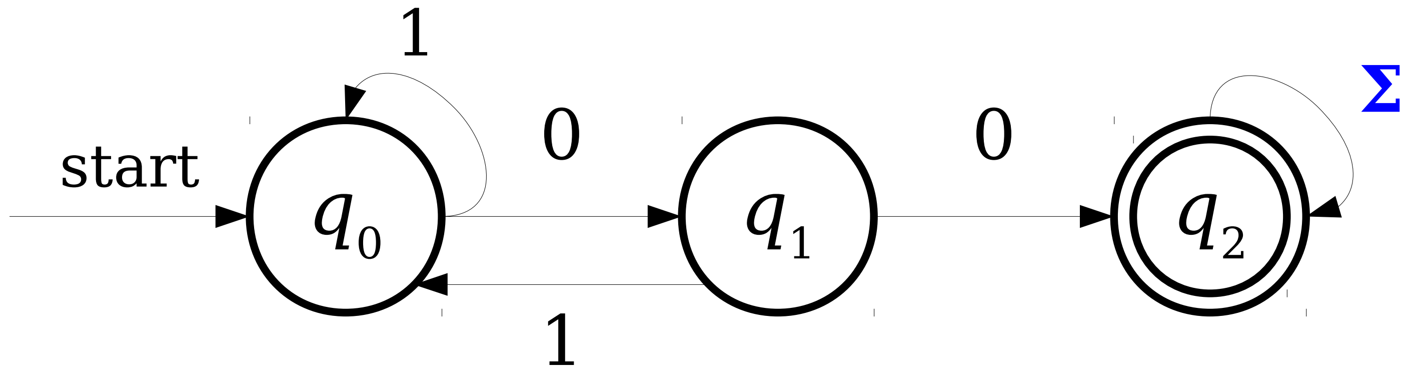
Recognizing Languages with DFAs

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


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[loop above, "1"] \arrow[r, "0"] & q_1 \arrow[l, bend left, "1"] \arrow[r, "0"] & q_2 \arrow[loop above, "0, 1"]
\end{tikzcd}$$

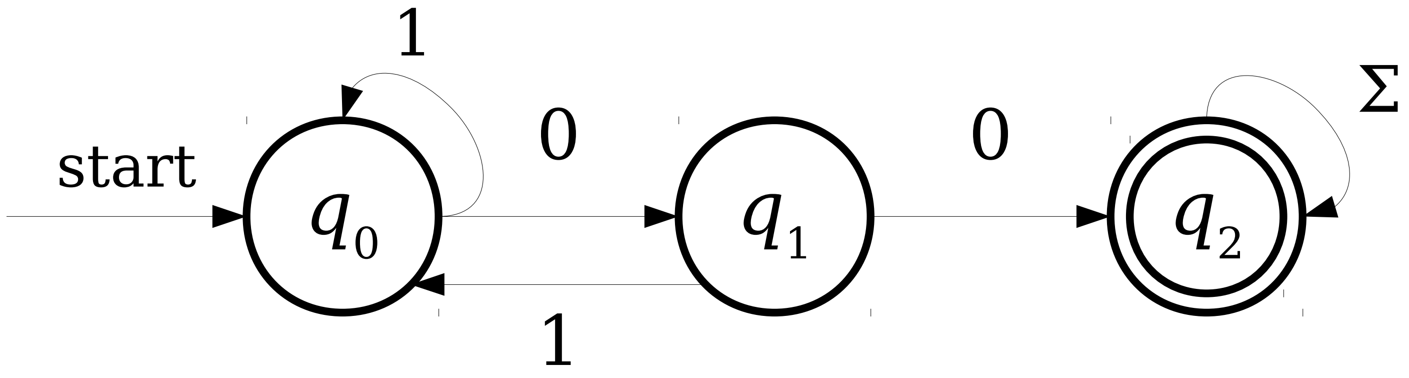
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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



More Elaborate DFAs

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

Suppose the alphabet is

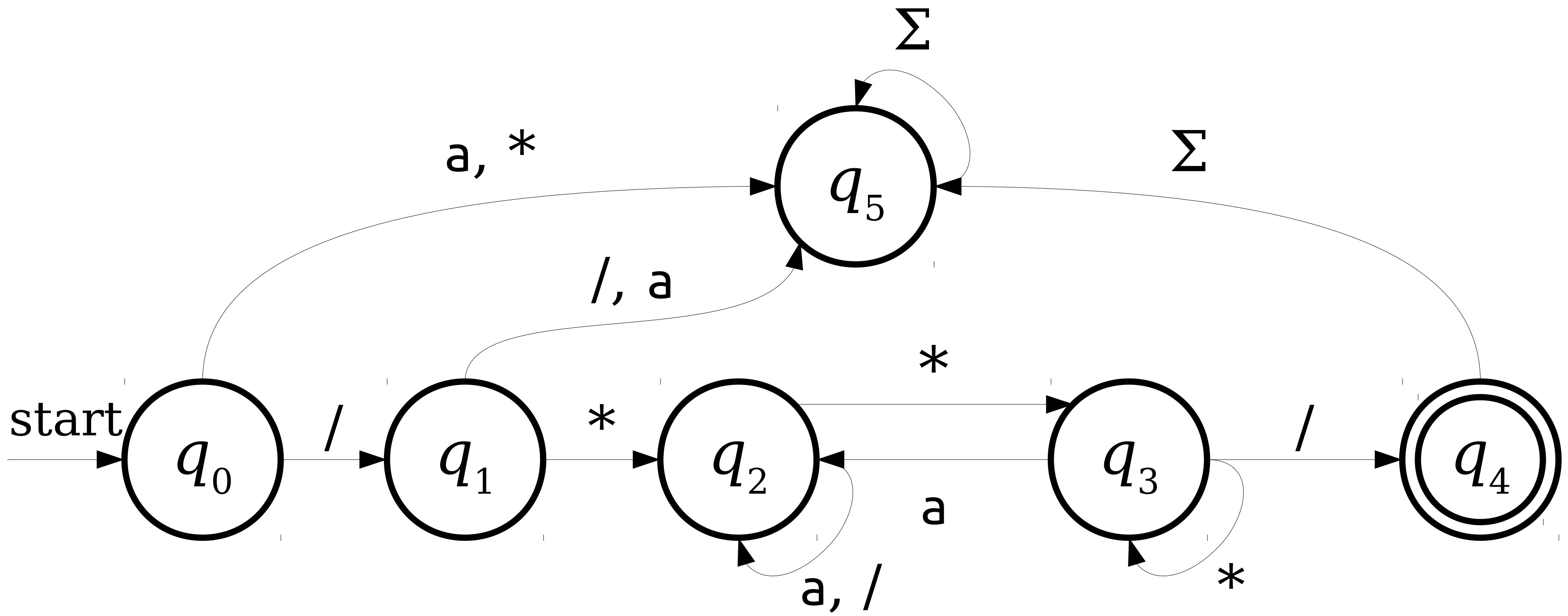
$\Sigma = \{\ \texttt{a}, \texttt{*}, \texttt{/}\ \}$

Try designing a DFA for comments! Some test cases:

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

More Elaborate DFAs

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


Σ
a, *
Σ
q₅
/, a
start
q₀
/
q₁
*
q₂
*
q₃
/
q₄
a
a, /
*