Finite Automata

Part One

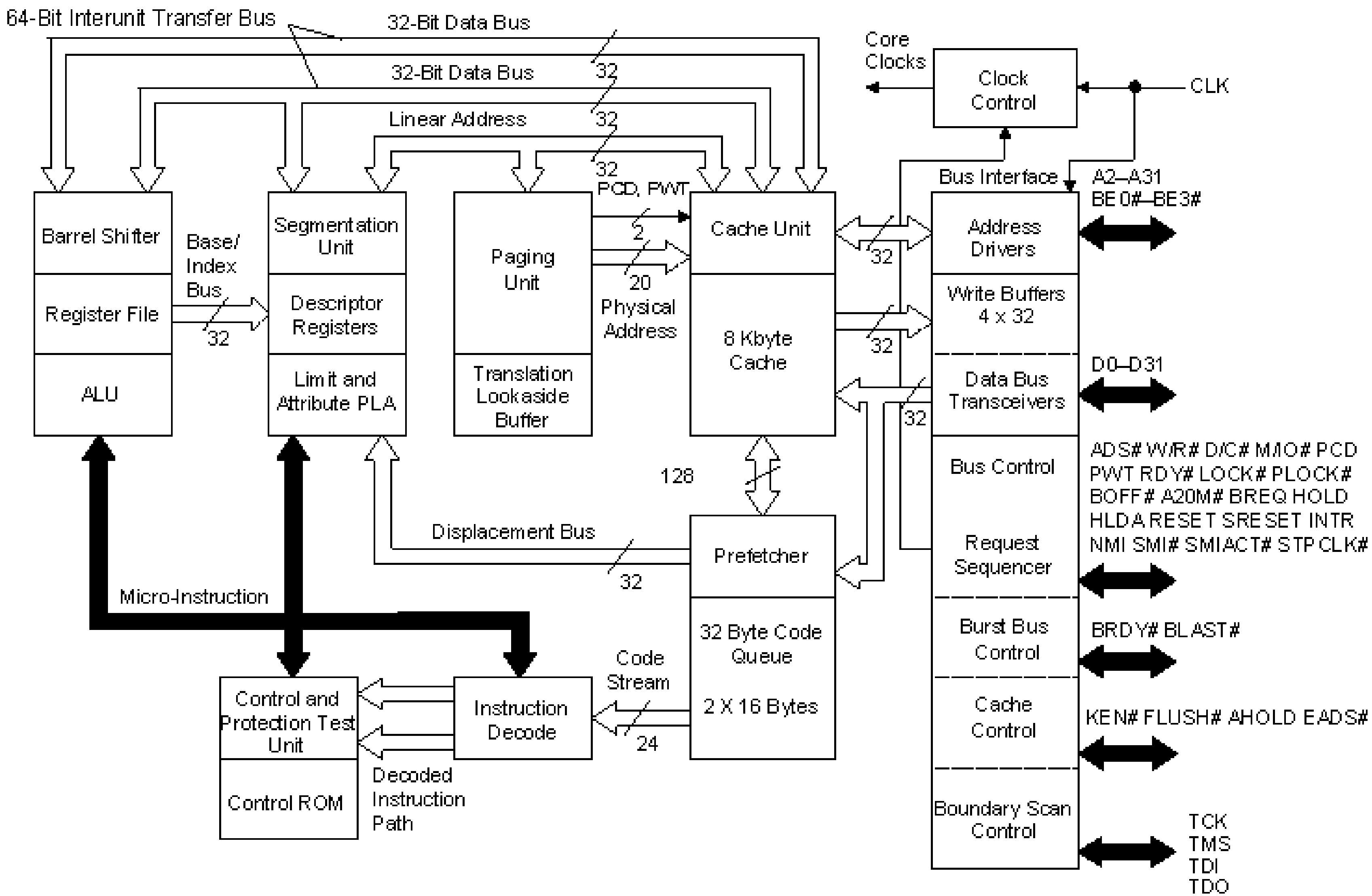
Computability Theory

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

Computers are Messy

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

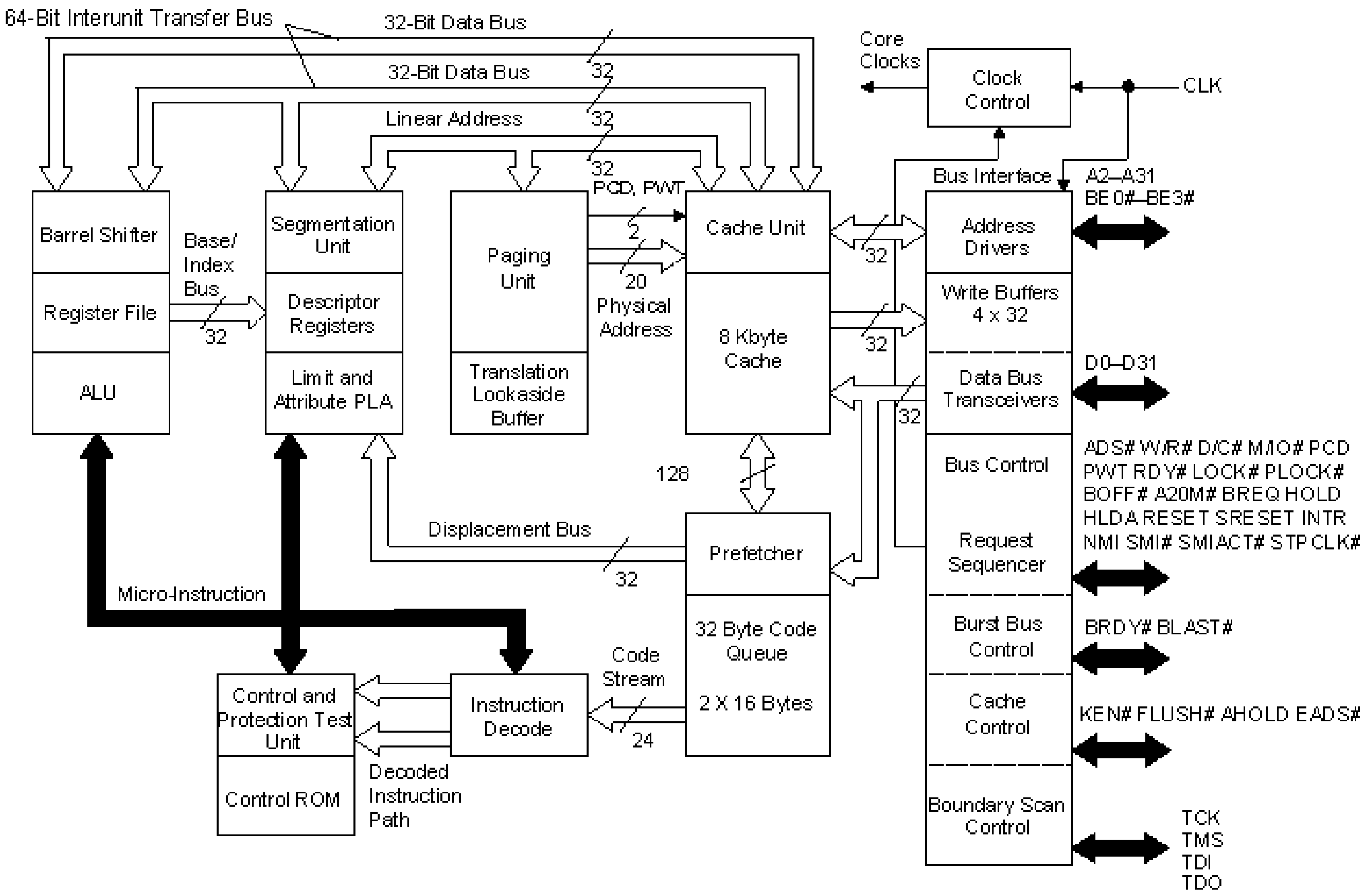
Computers are Messy

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

We need a simpler way of discussing computing machines.

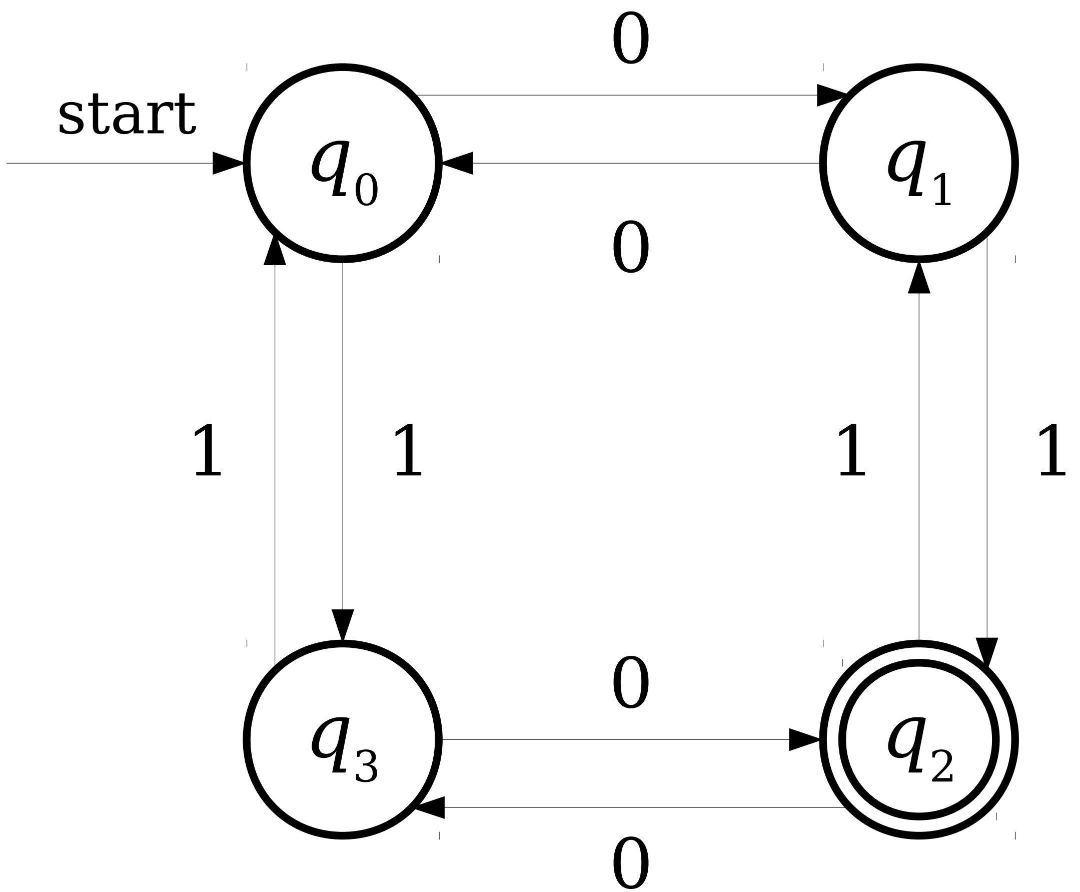
An ***automaton*** (plural: **automata**) is a mathematical model of a computing device.

Computers are Messy



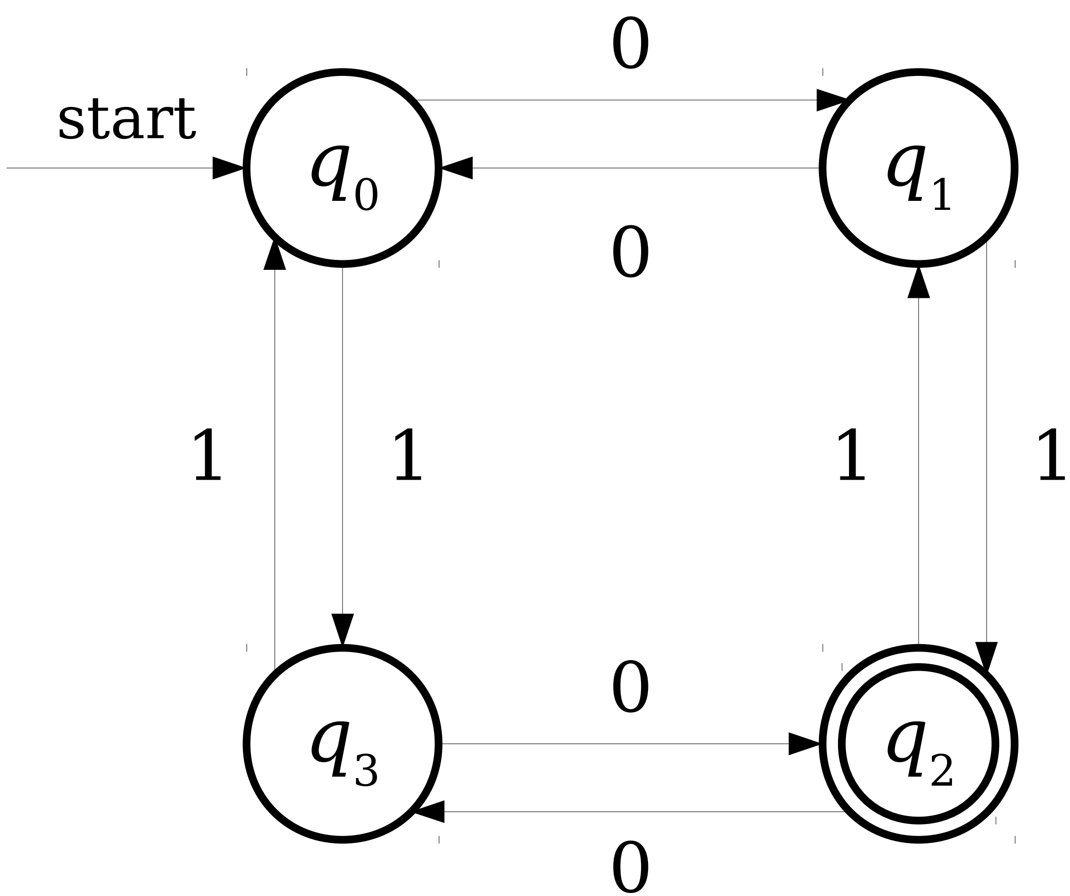
http://www.intel.com/design/intarch/prodbref/272713.htm

Automata are Clean



Computers are Messy

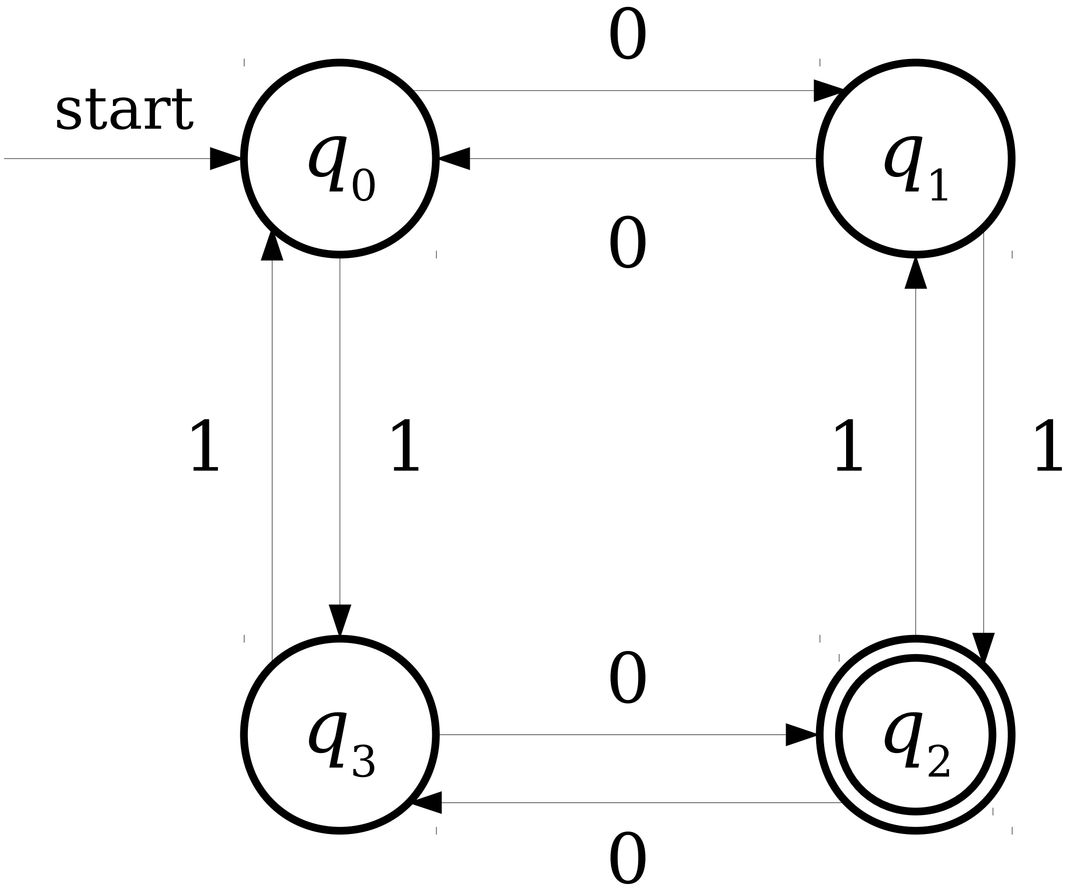
Automata are Clean



Computers are Messy

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

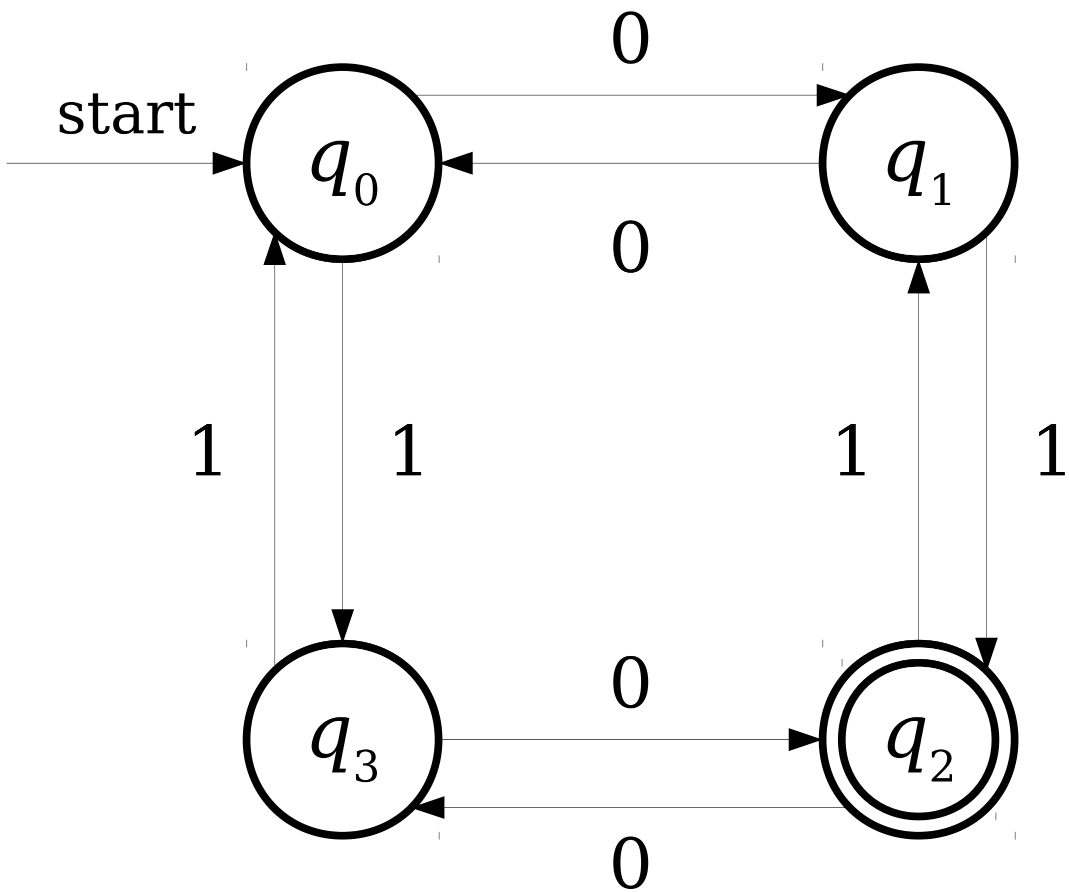
Automata are Clean



Computers are Messy

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

Automata are Clean



Why Build Models?

- 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?"

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 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: L is a language over Σ iff $L \subseteq \Sigma^*$.

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 both the choice of L and the choice of automaton.
- 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) output whether that string is contained in a particular language.

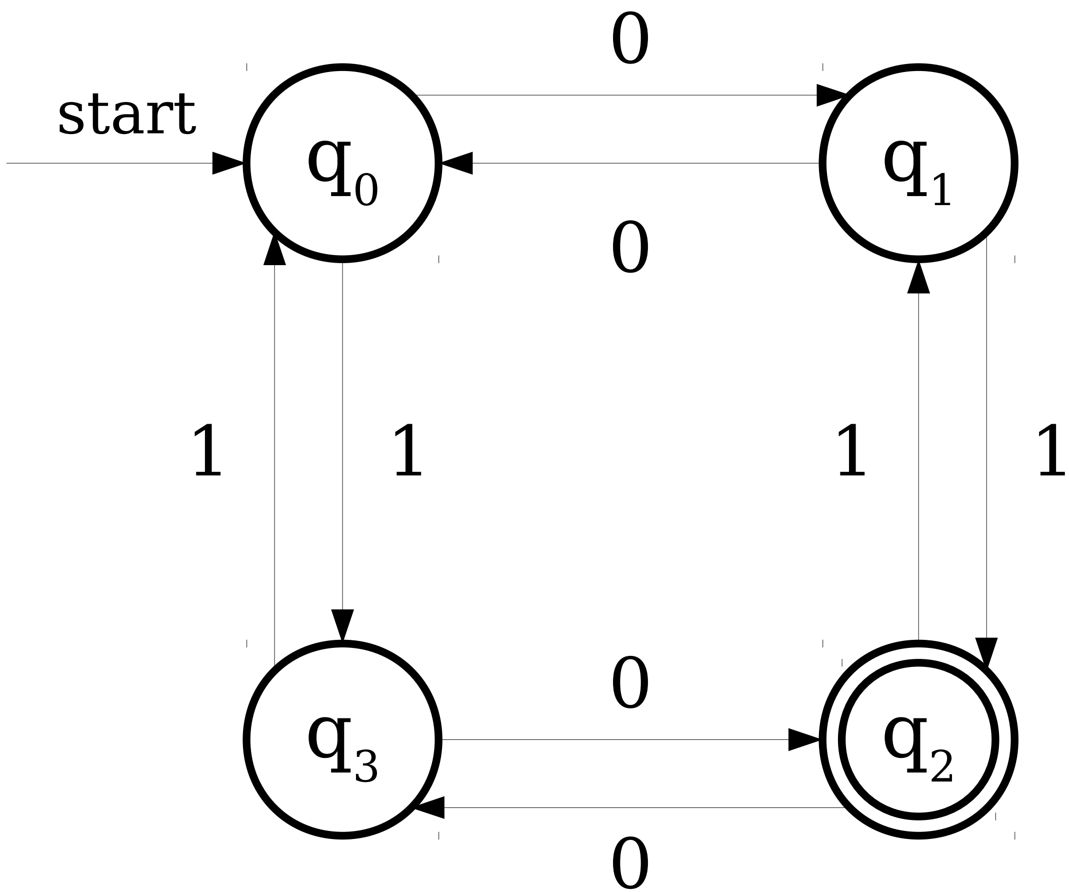
What problems can we solve with a computer?

Finite Automata

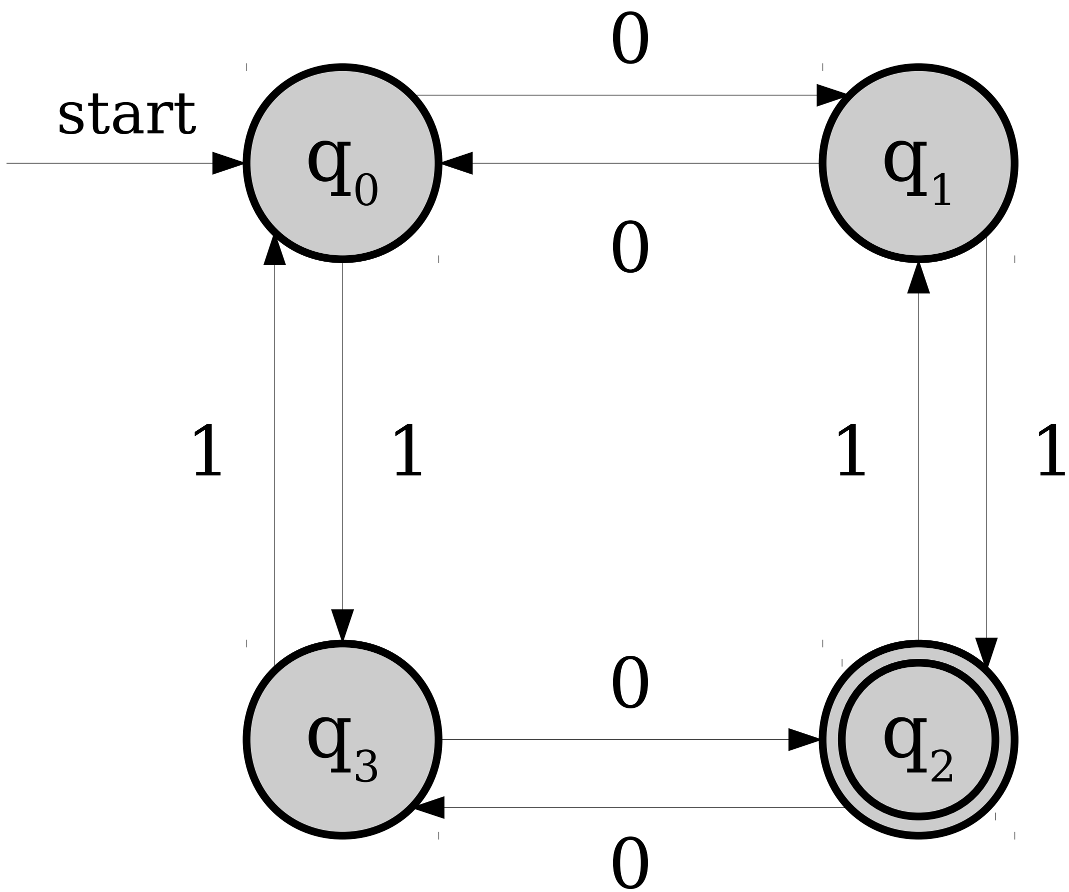
A ***finite automaton*** is a simple type of mathematical machine for determining whether a string is contained within some language.

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

A Simple Finite Automaton

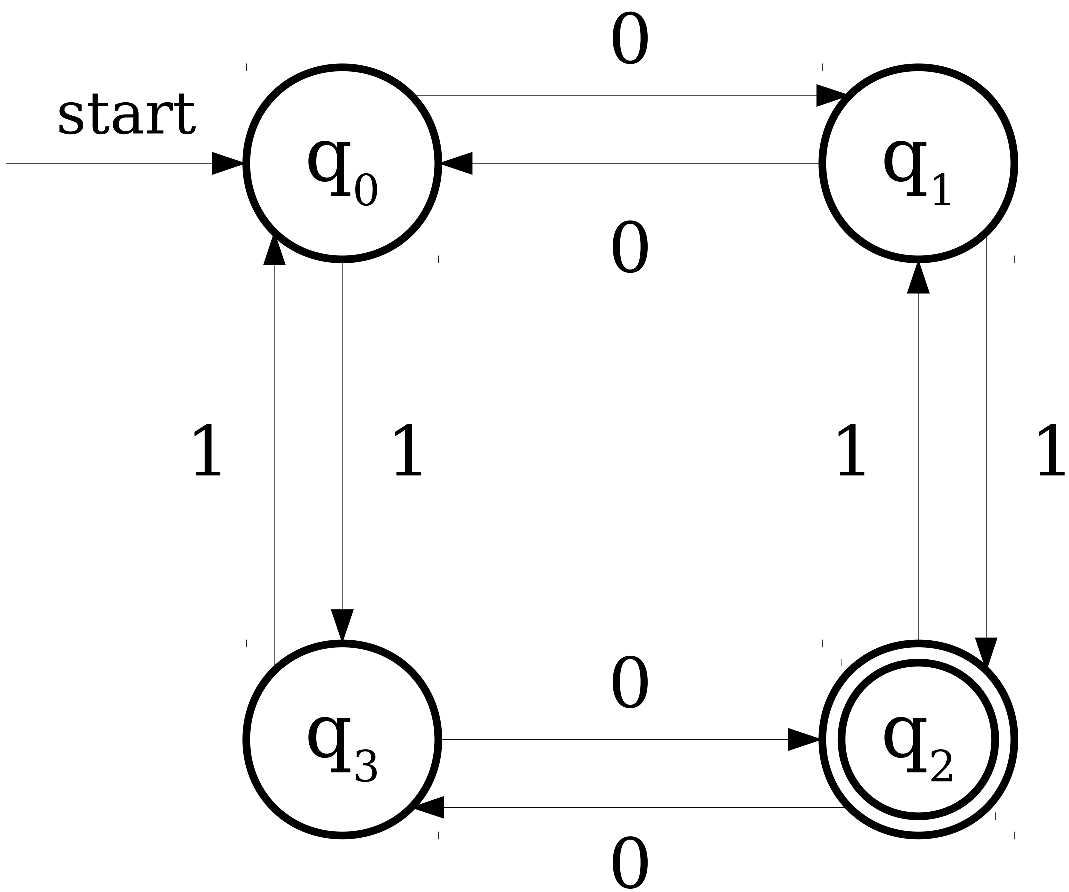


A Simple Finite Automaton

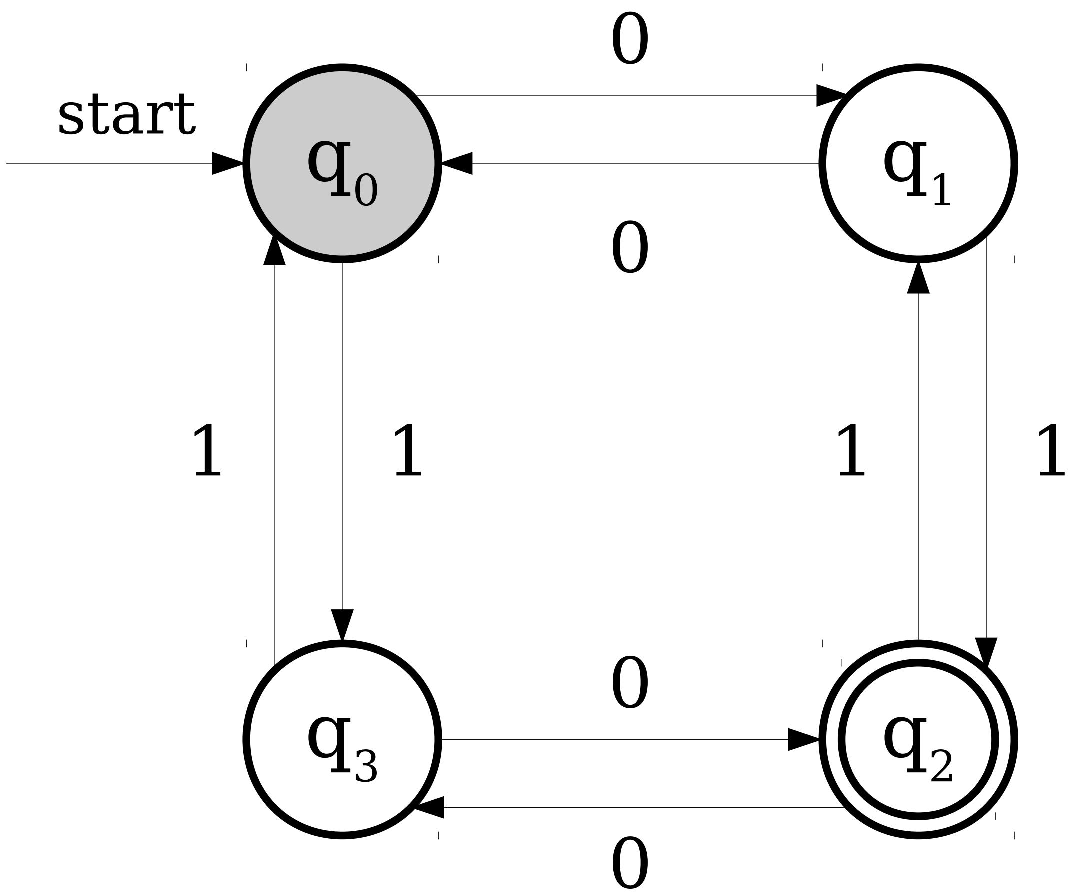


Each circle represents a state of the automaton.

A Simple Finite Automaton

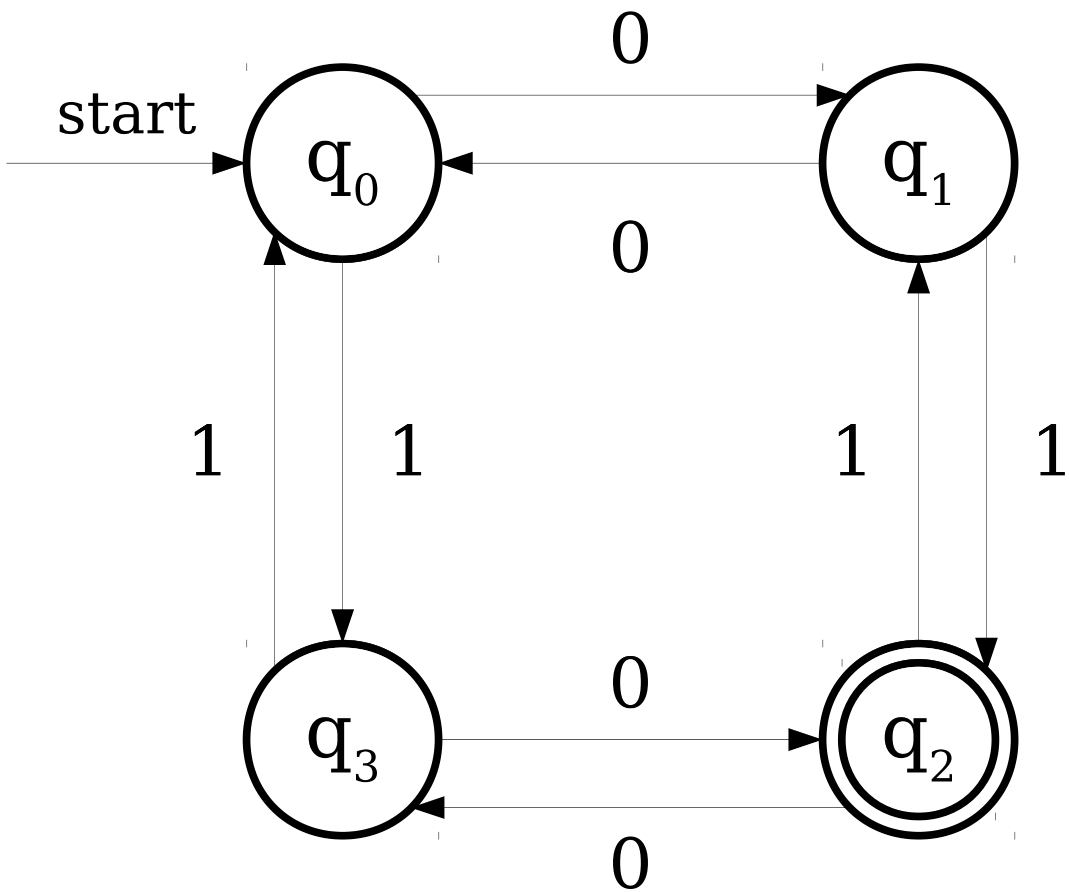


A Simple Finite Automaton

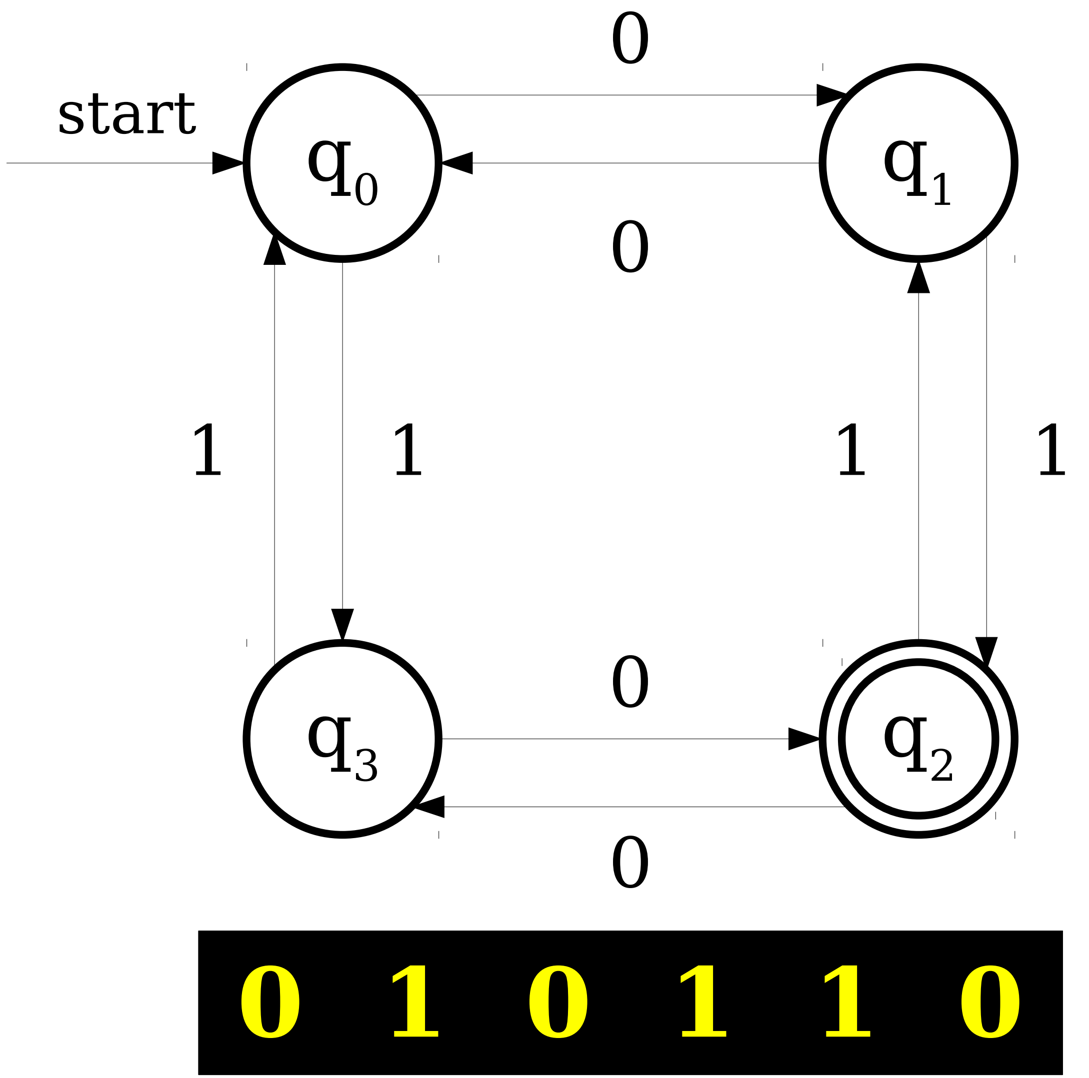


One special state is designated as the **start state**.

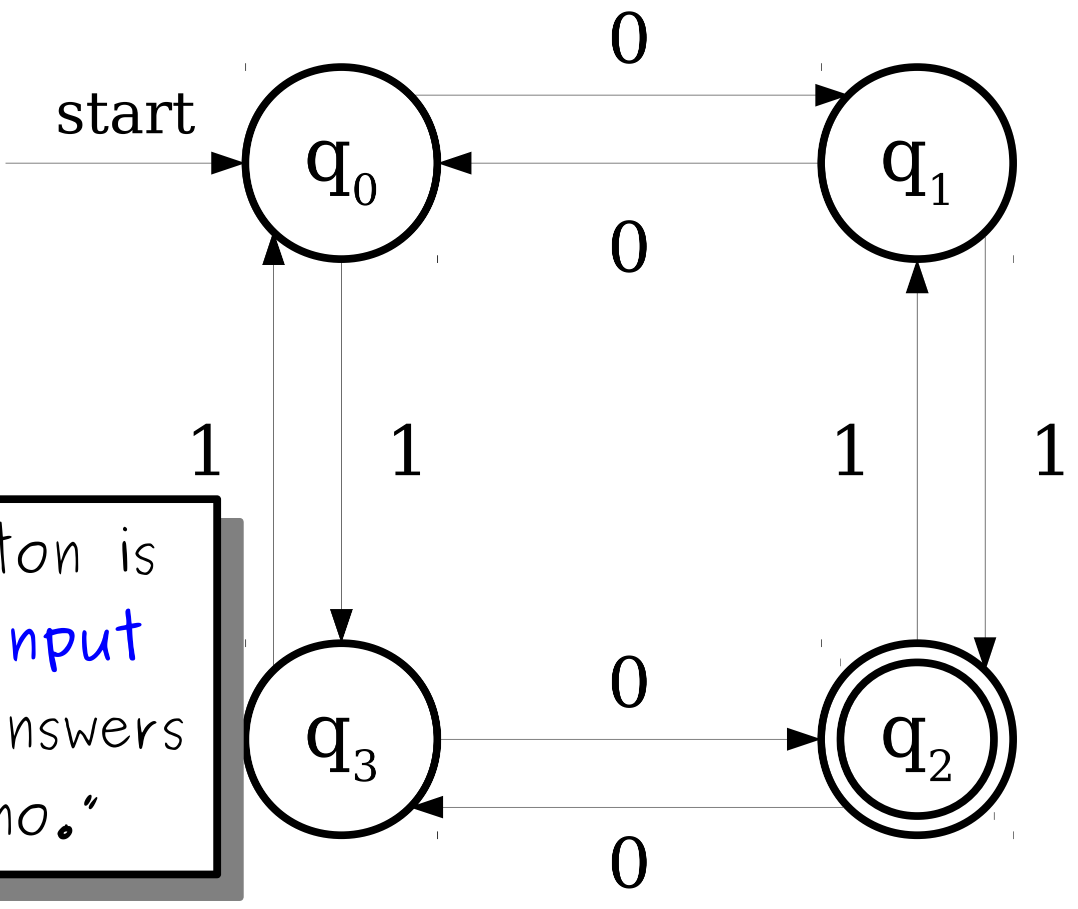
A Simple Finite Automaton



A Simple Finite Automaton



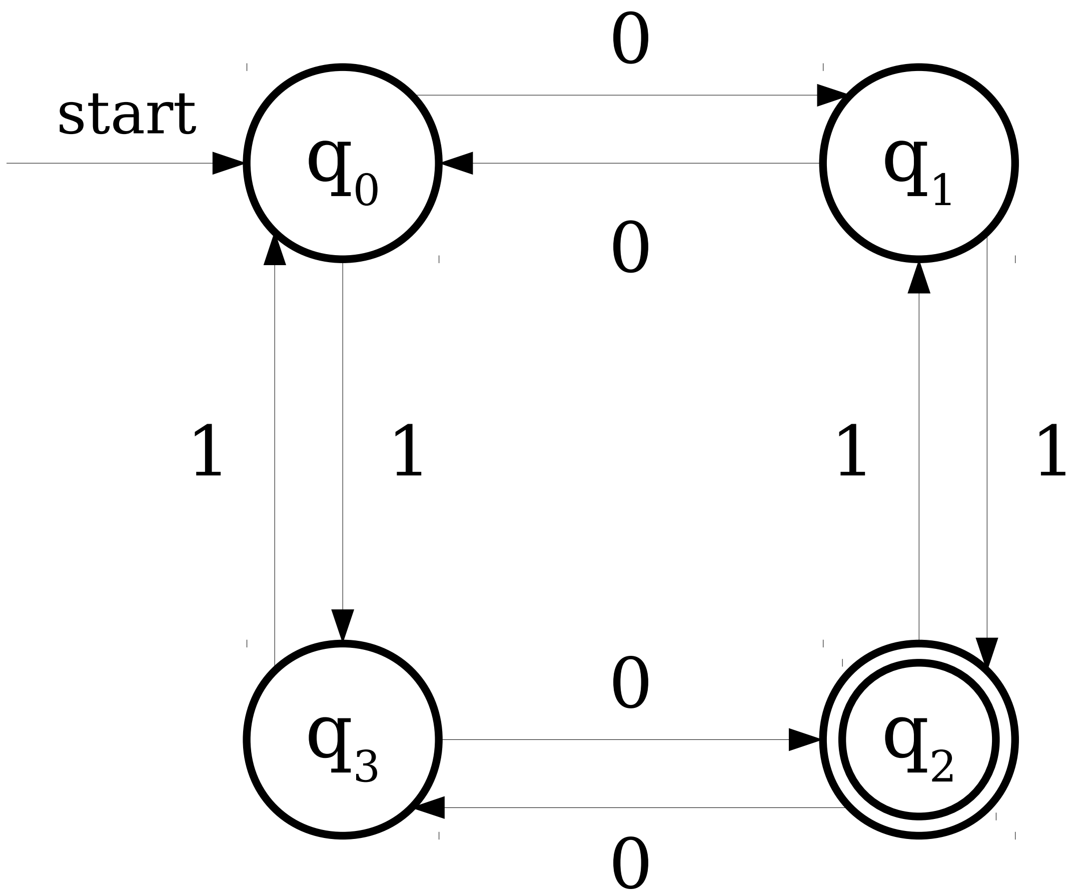
A Simple Finite Automaton



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

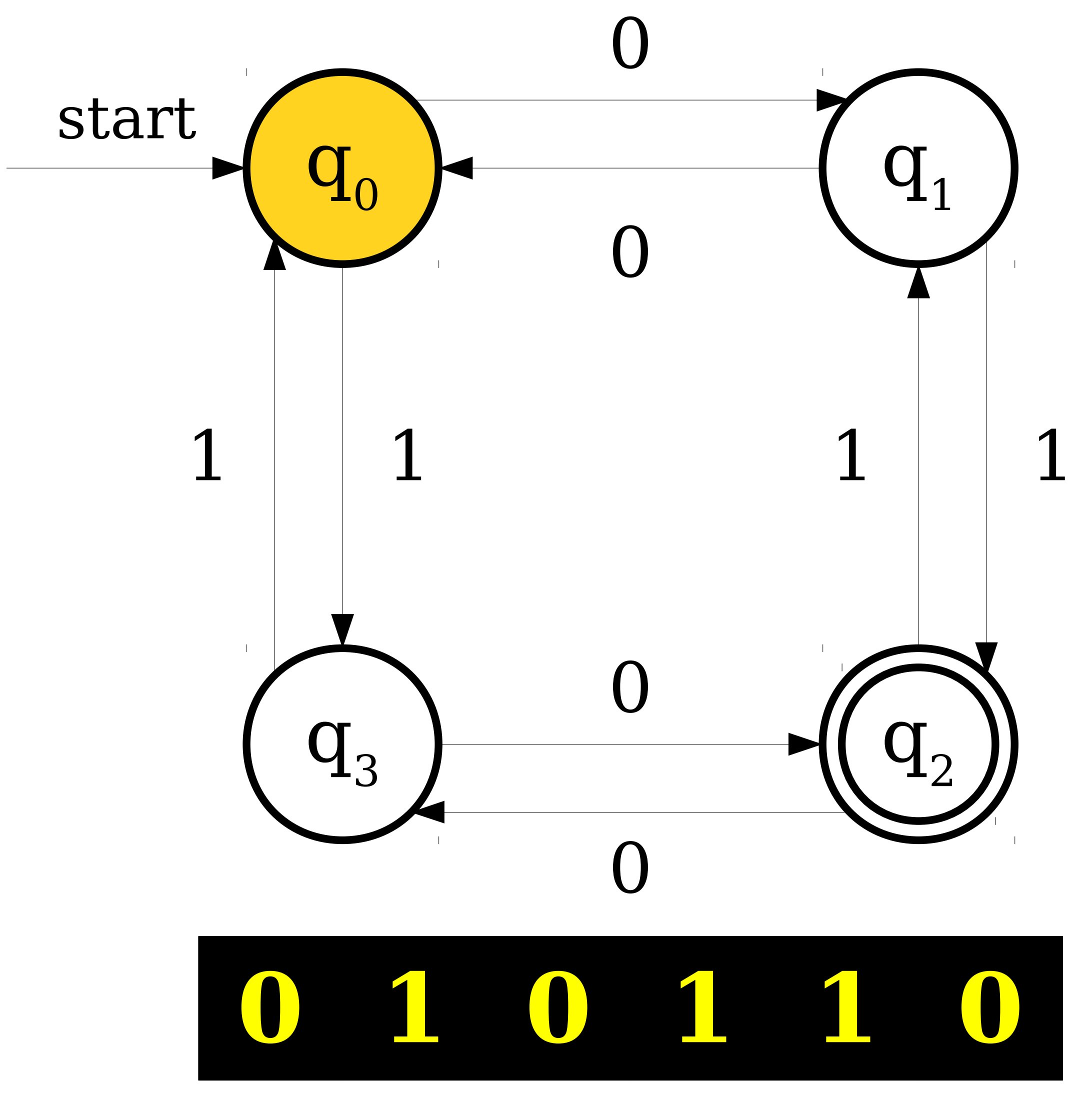
0 1 0 1 1 0

A Simple Finite Automaton

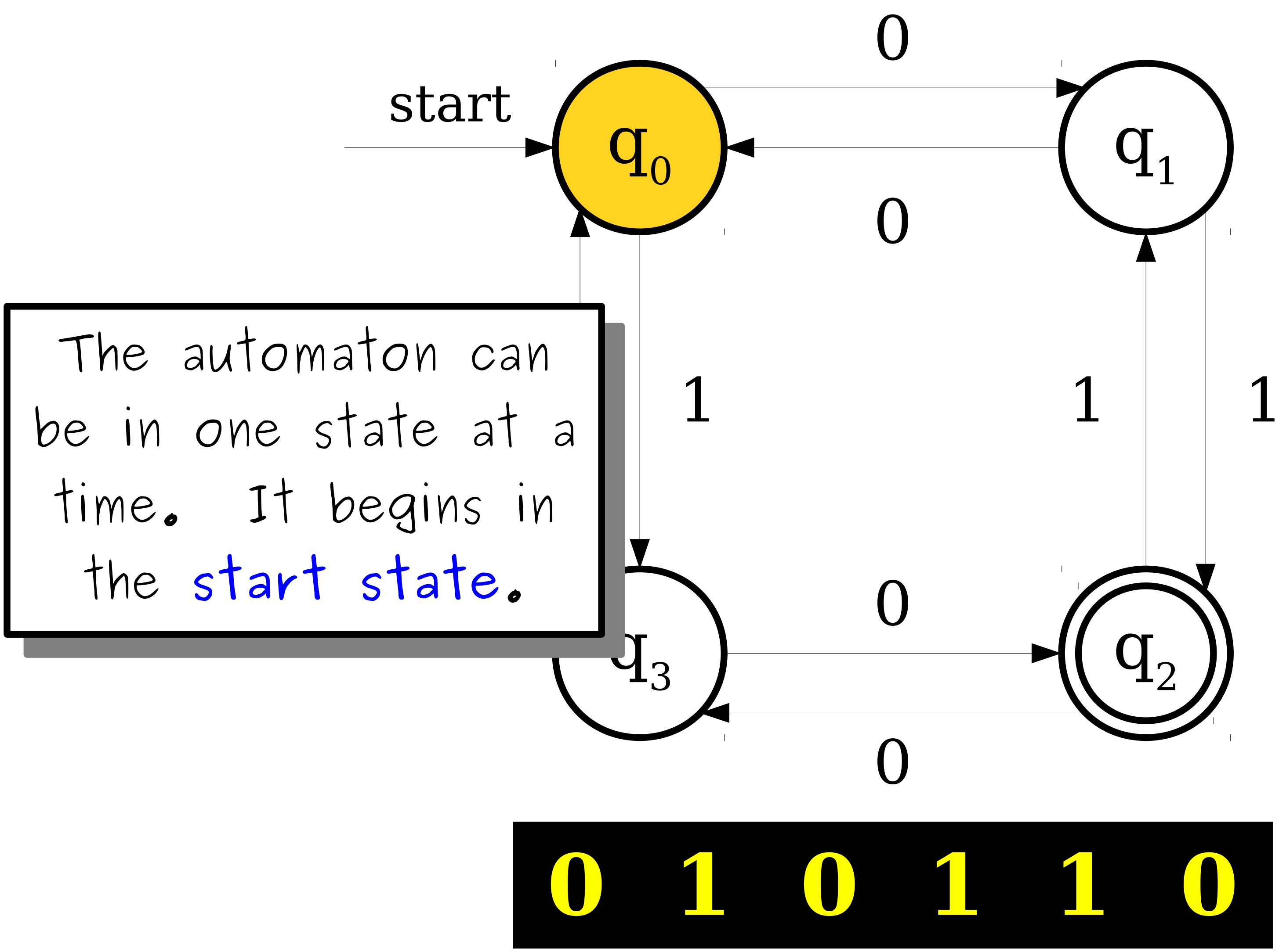


0 1 0 1 1 0

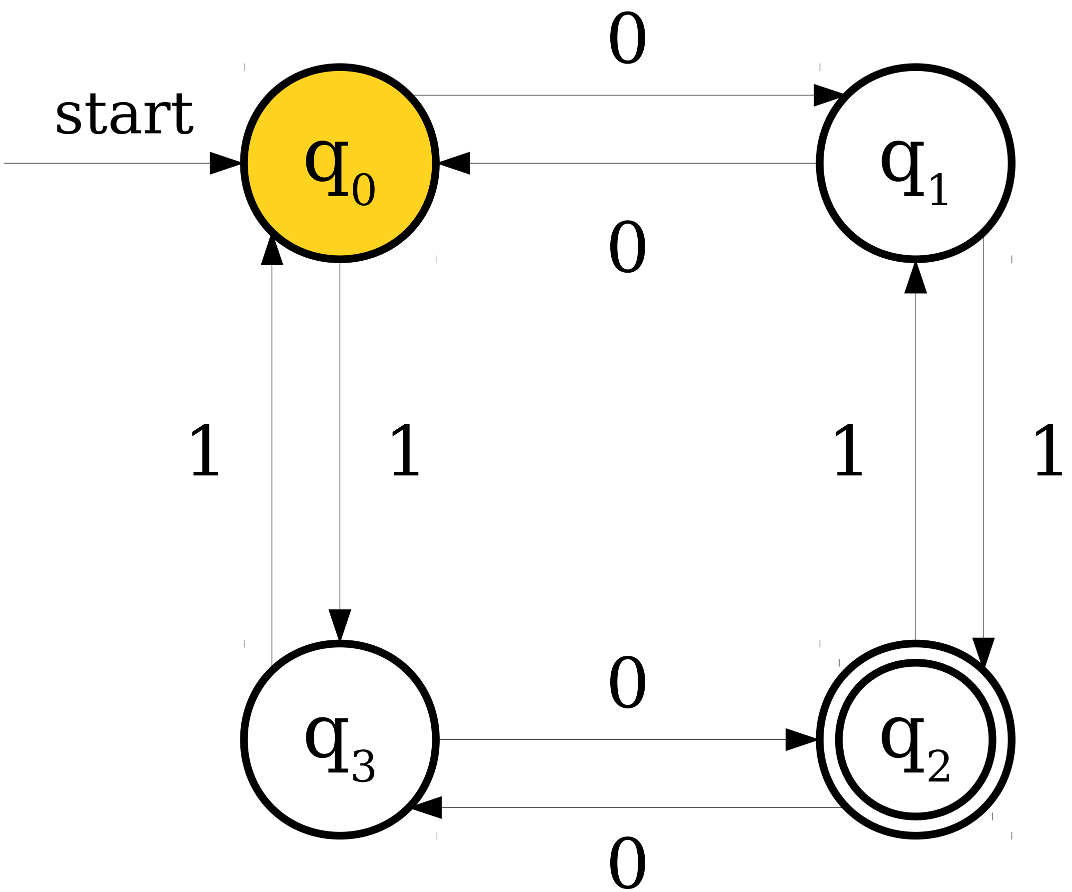
A Simple Finite Automaton



A Simple Finite Automaton

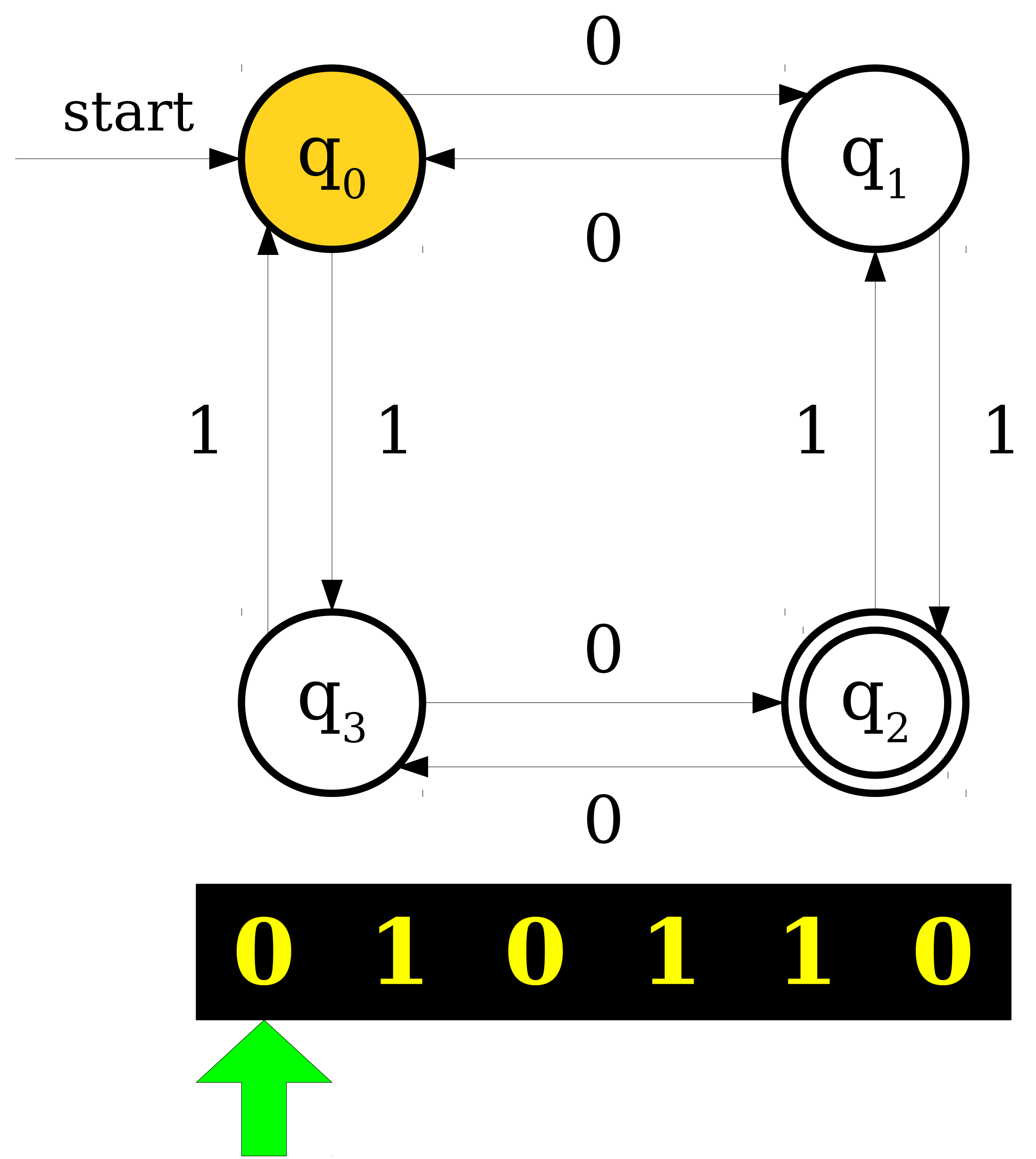


A Simple Finite Automaton

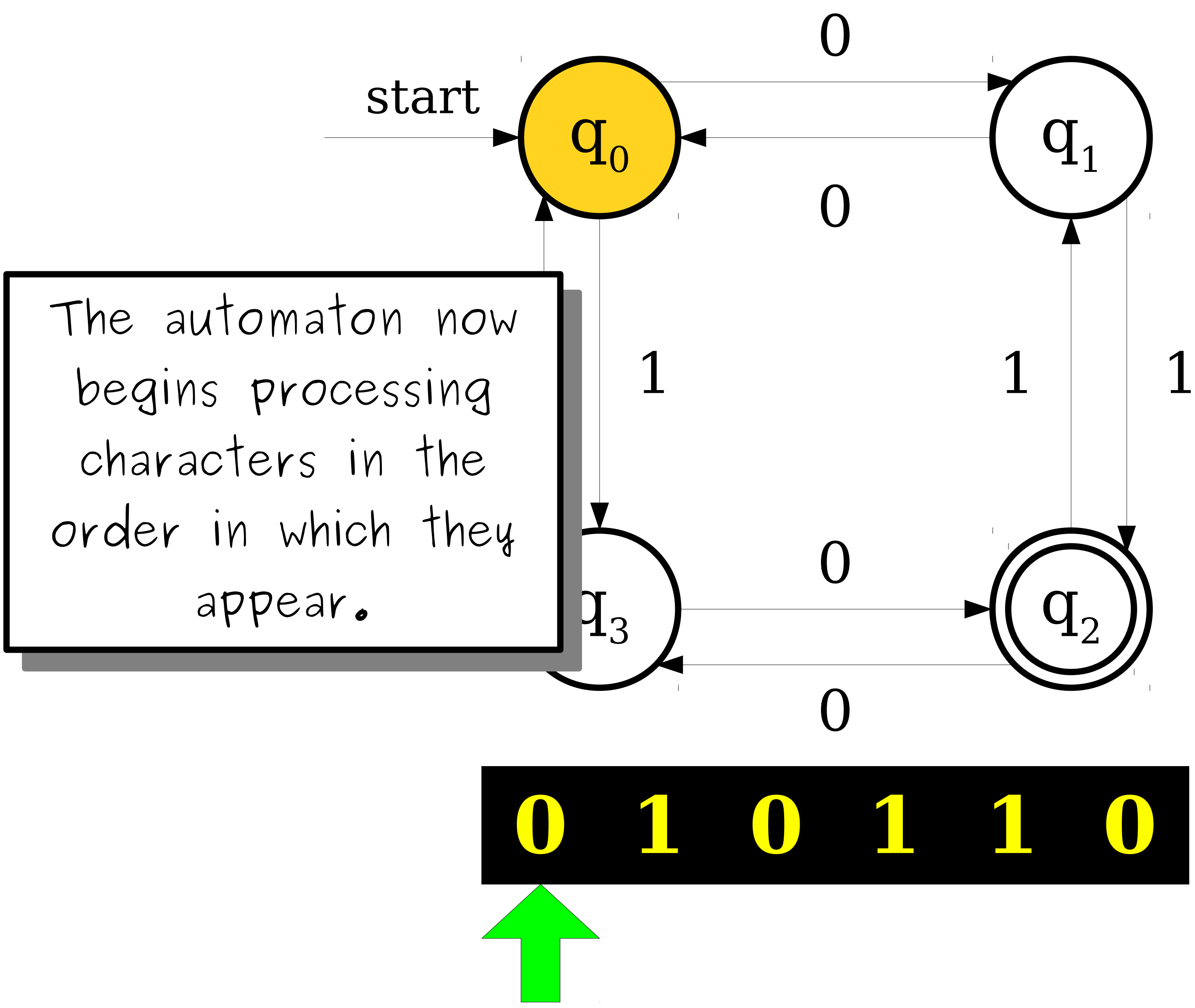


0 1 0 1 1 0

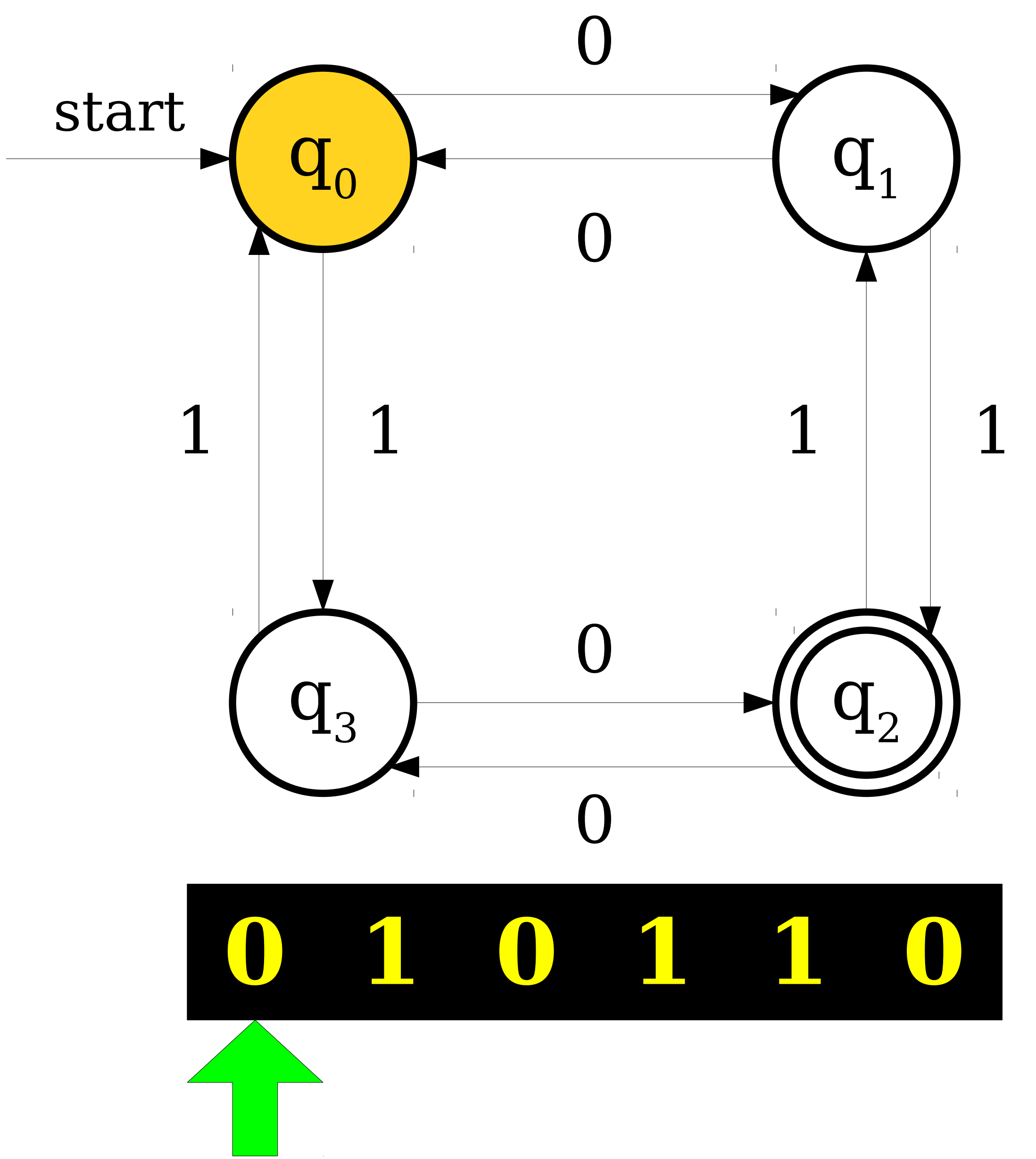
A Simple Finite Automaton



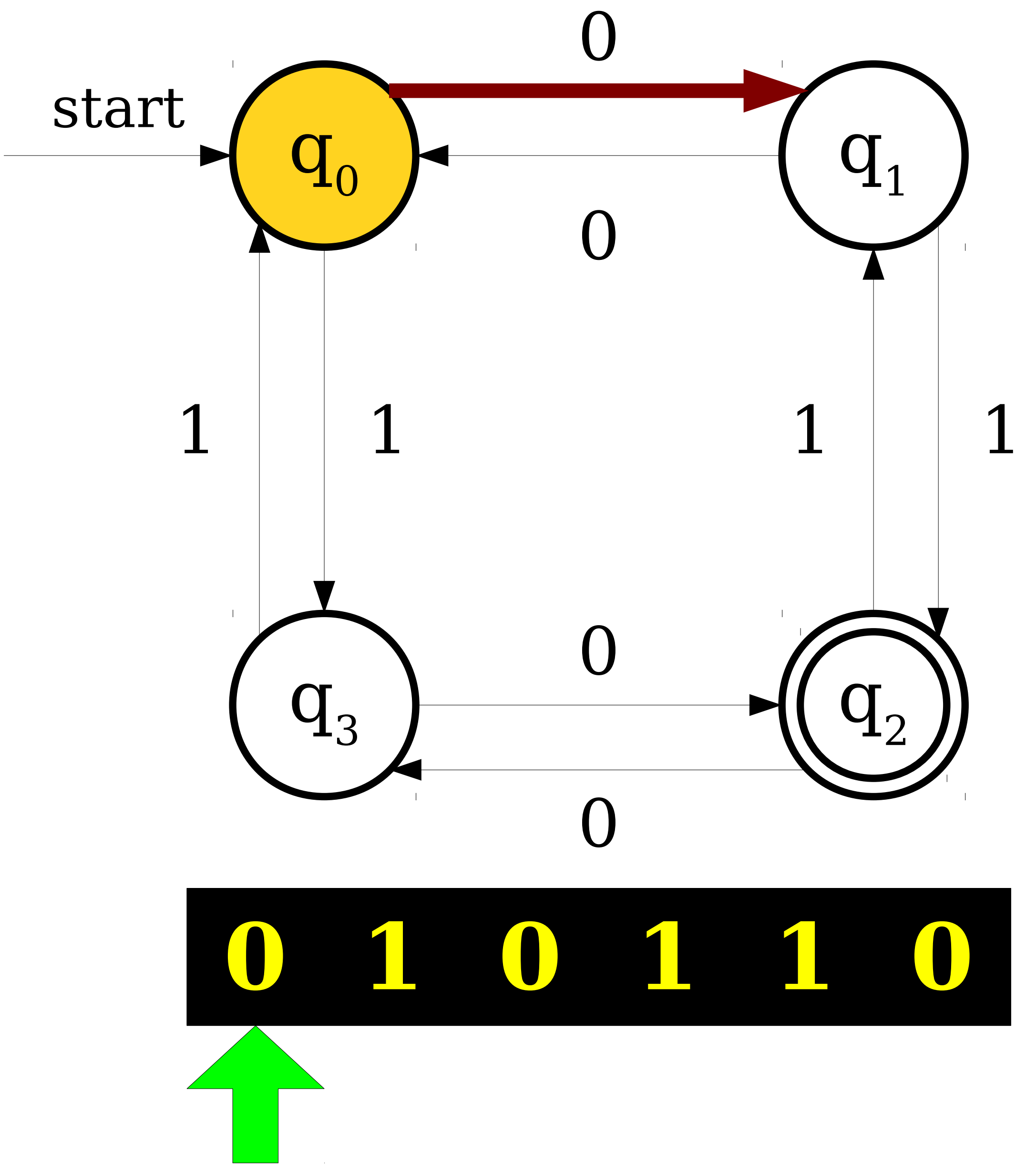
A Simple Finite Automaton



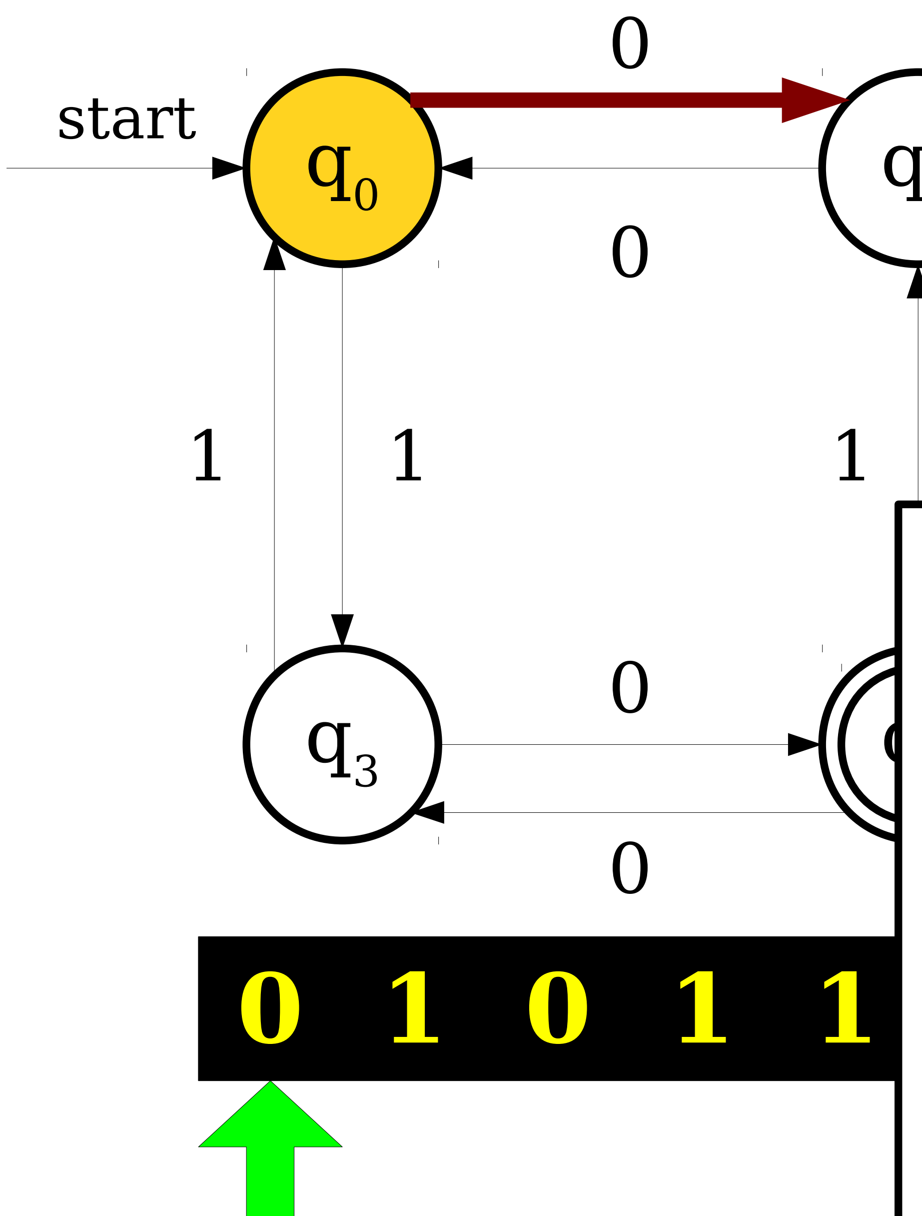
A Simple Finite Automaton



A Simple Finite Automaton

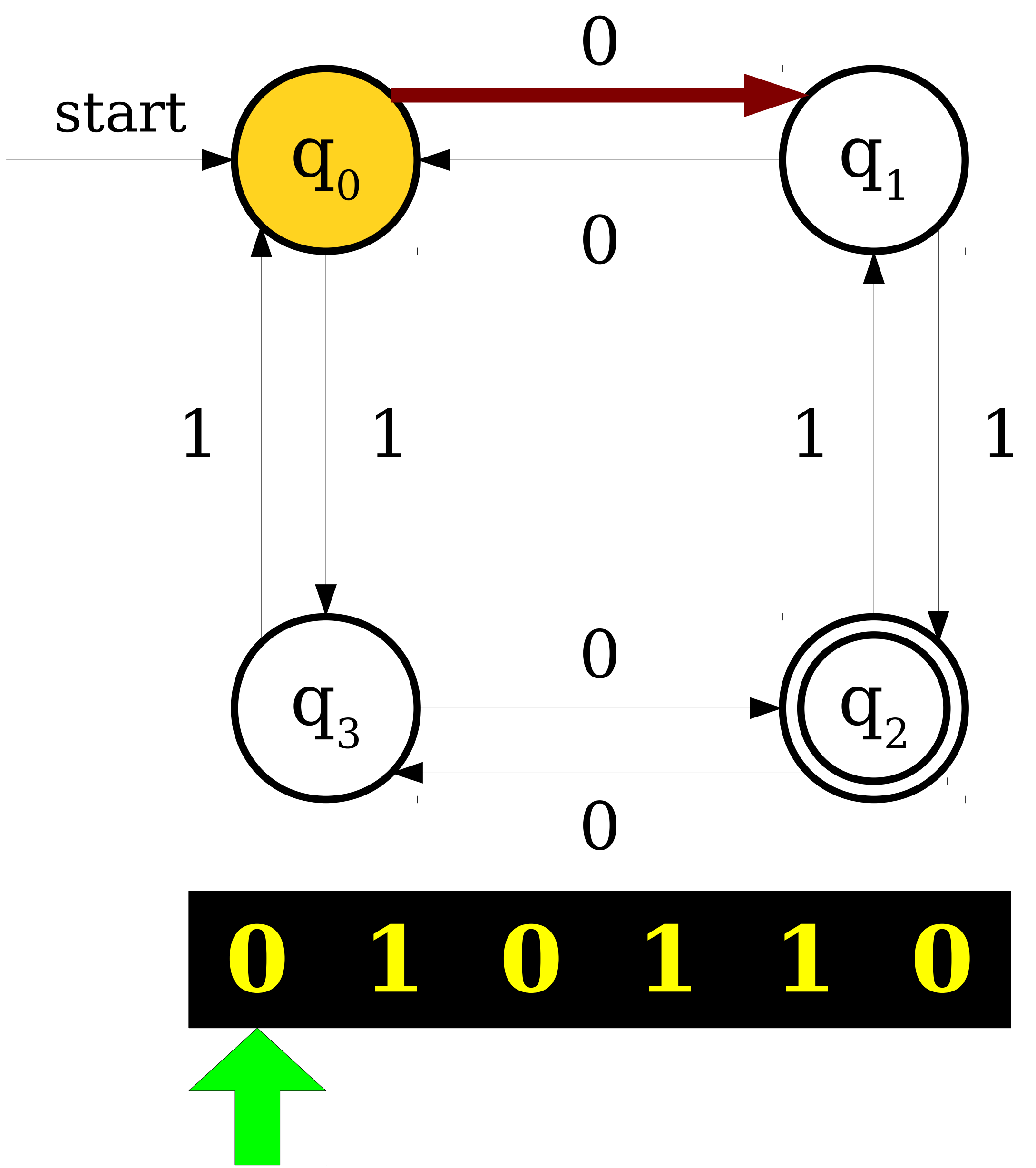


A Simple Finite Automaton

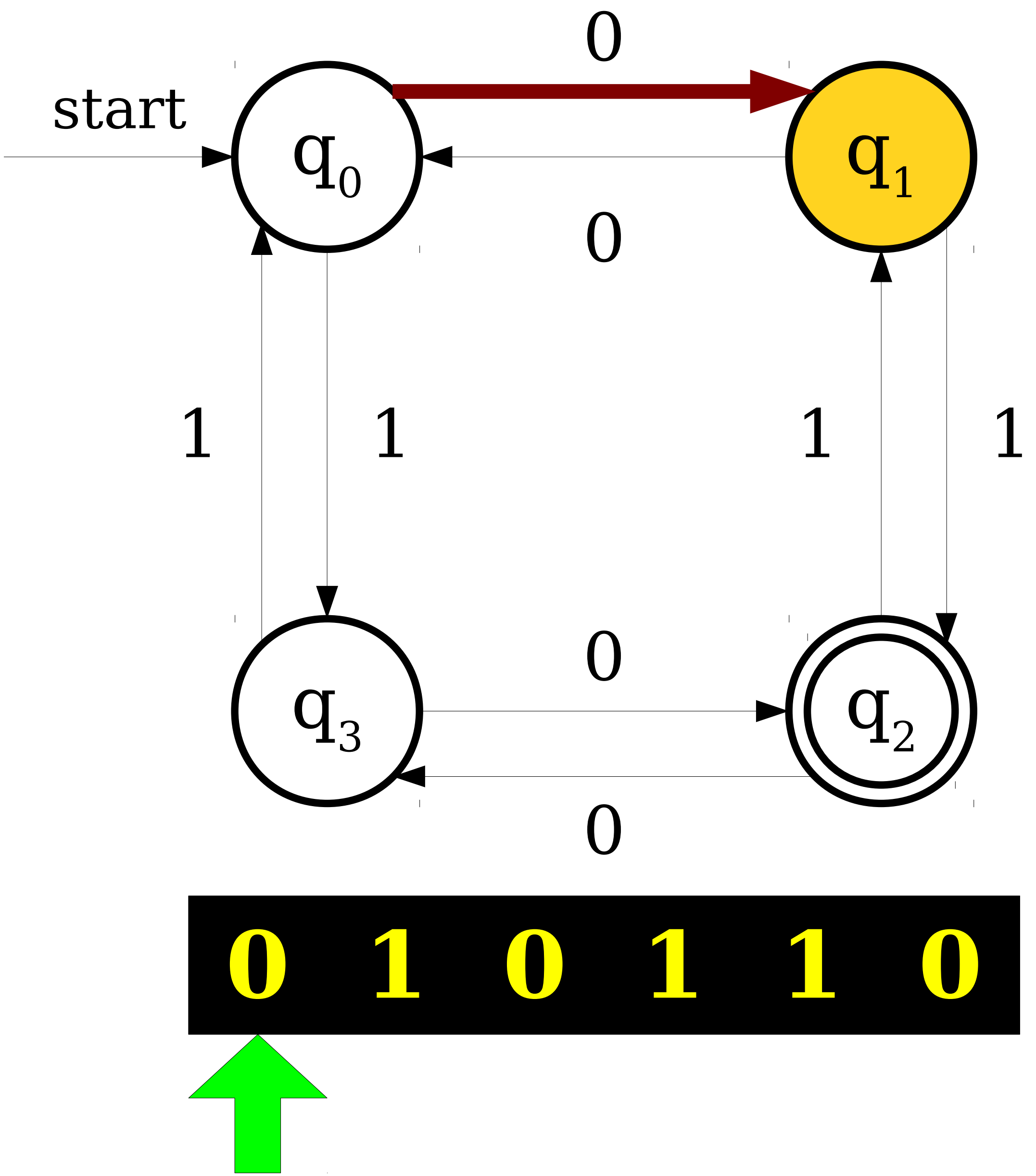


Each arrow in this diagram represents a **transition**. The automaton always follows the transition corresponding to the current symbol being read.

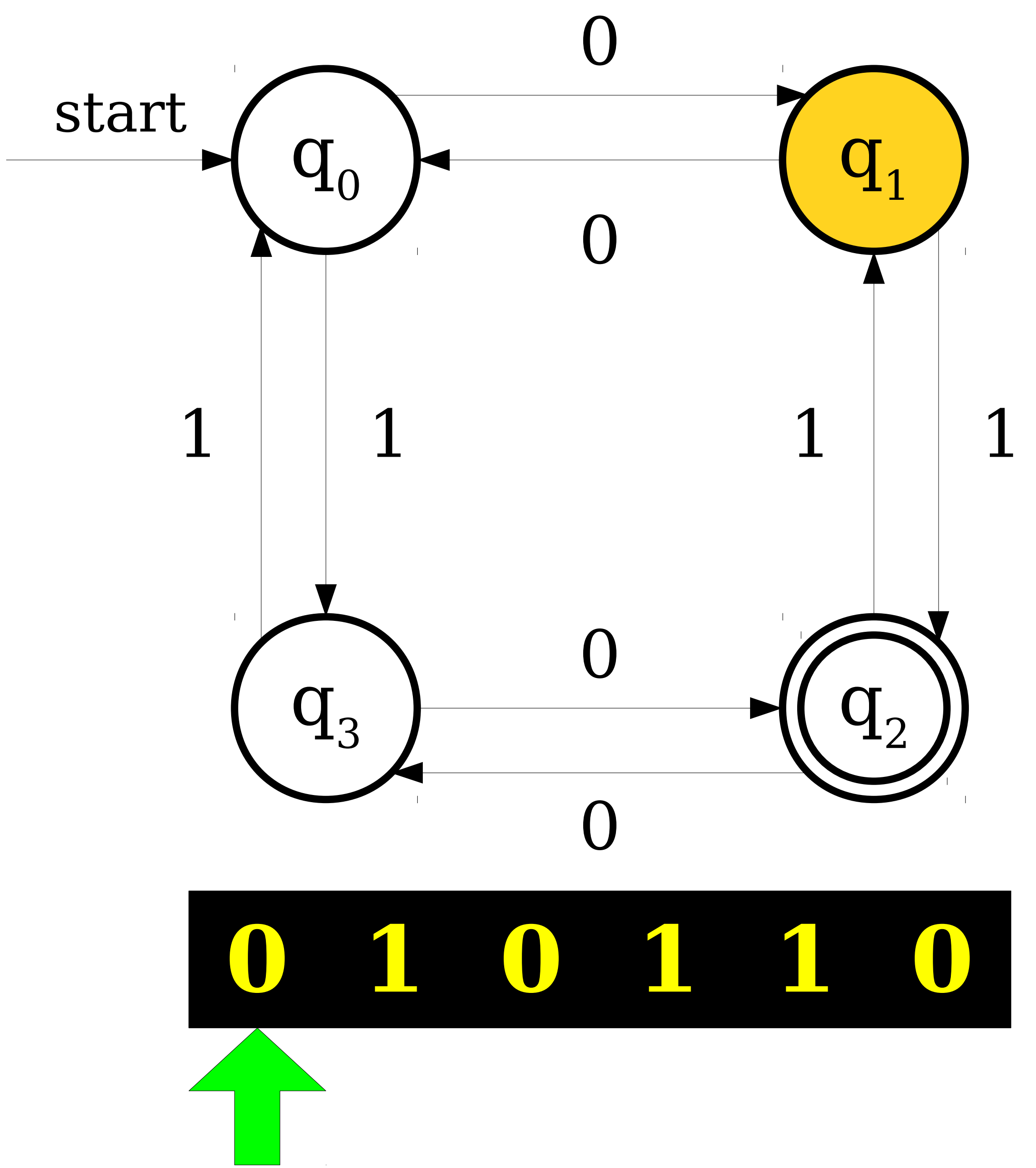
A Simple Finite Automaton



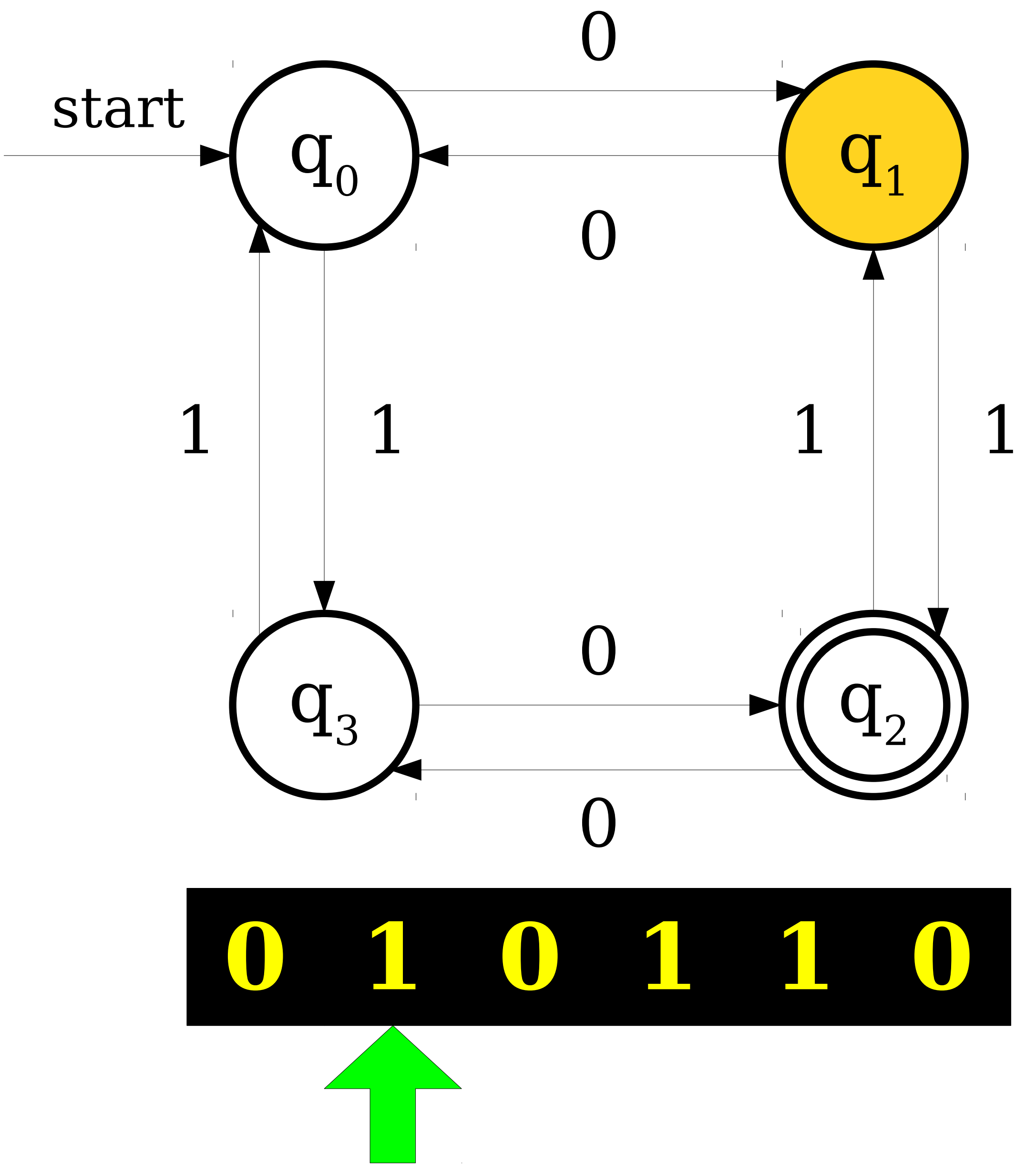
A Simple Finite Automaton



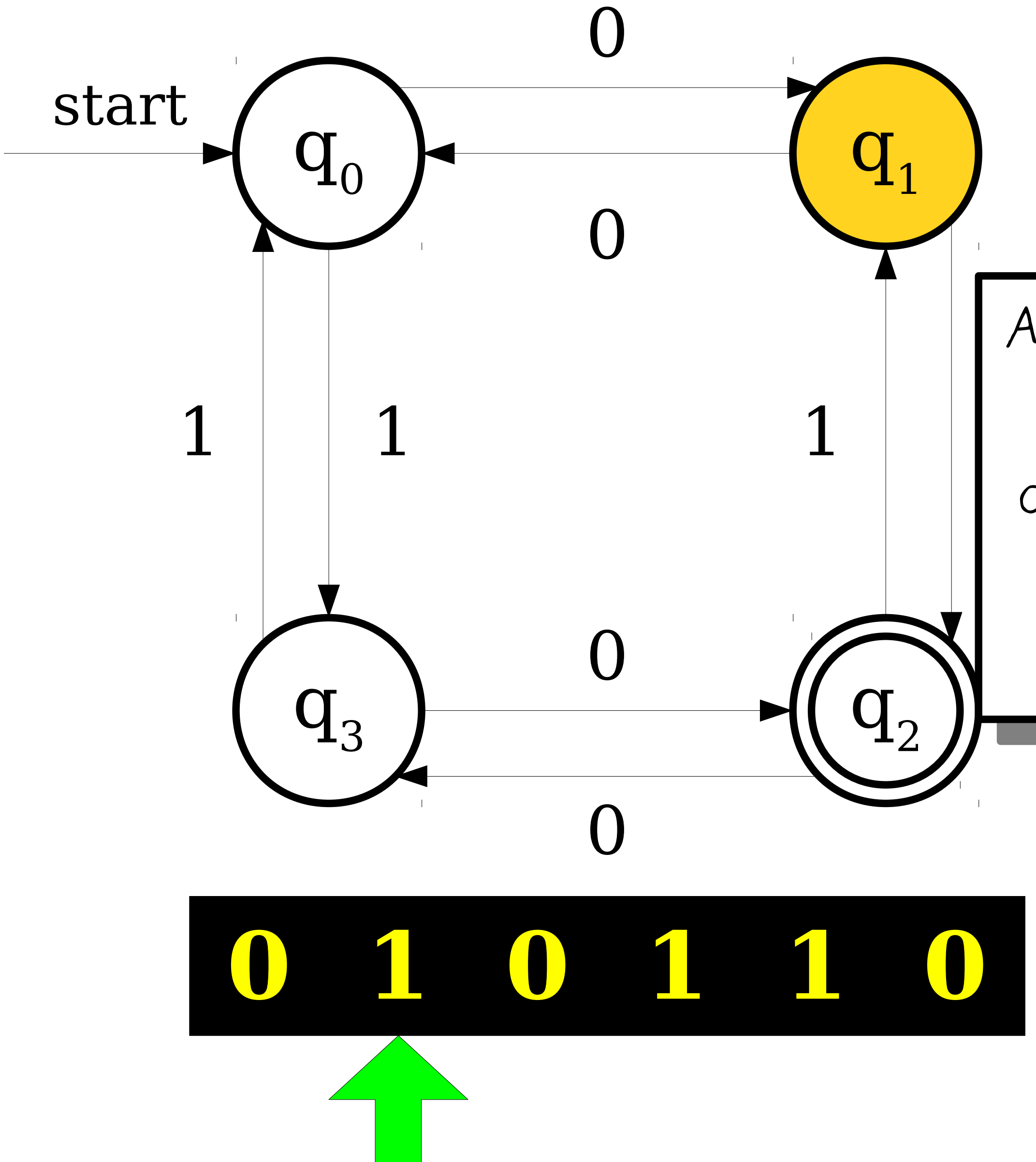
A Simple Finite Automaton



A Simple Finite Automaton

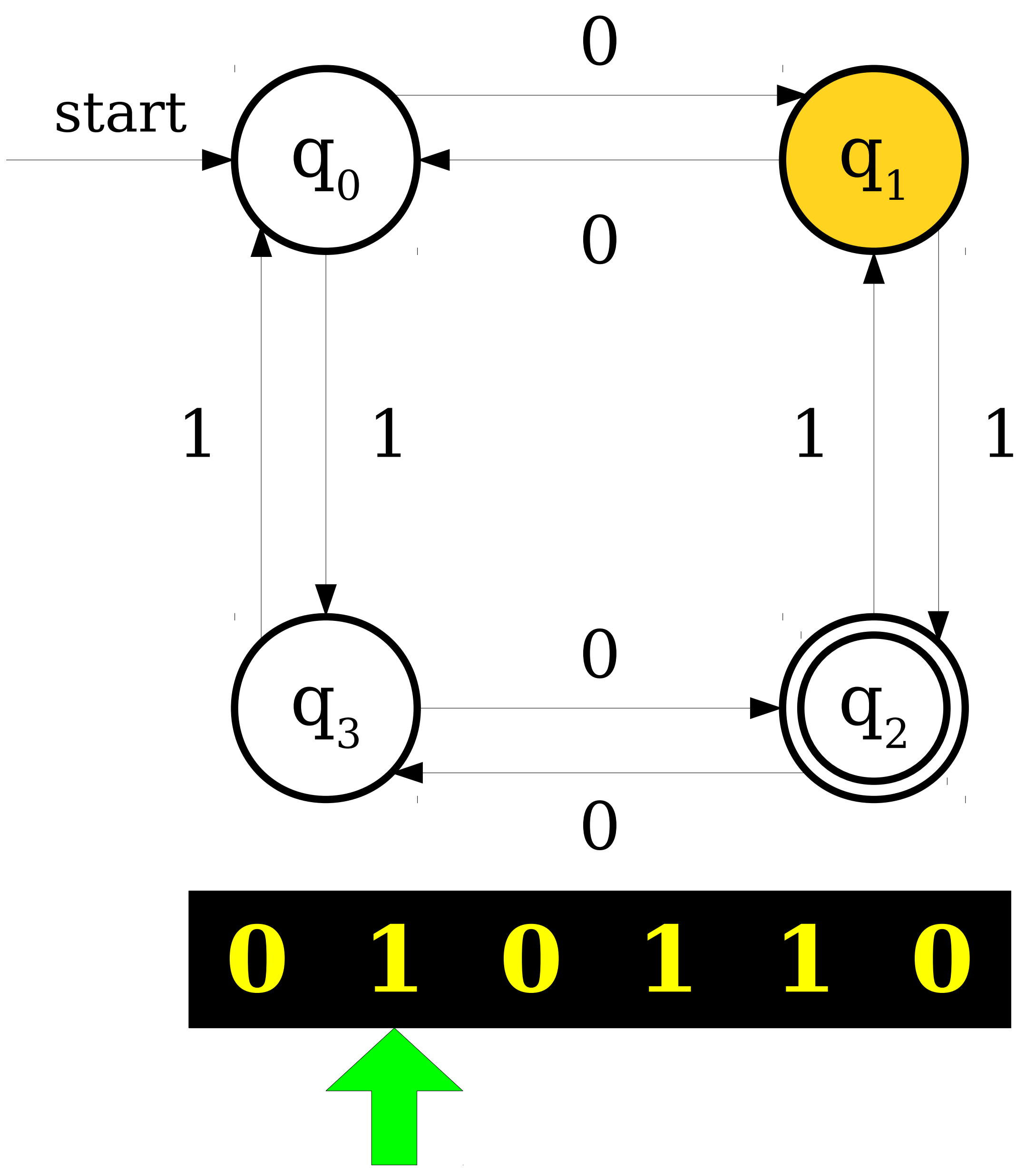


A Simple Finite Automaton

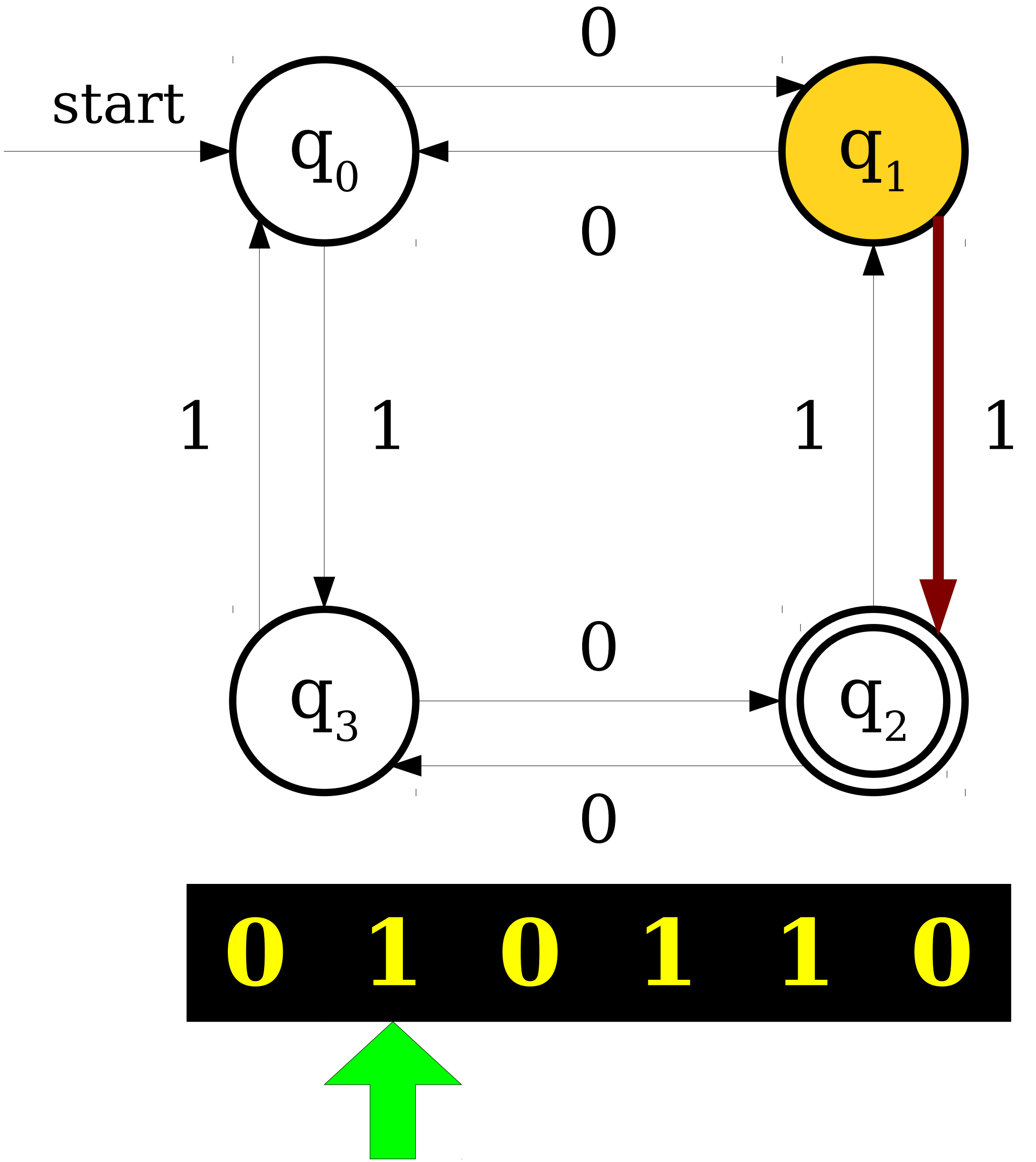


After transitioning, the automaton considers the next symbol in the input.

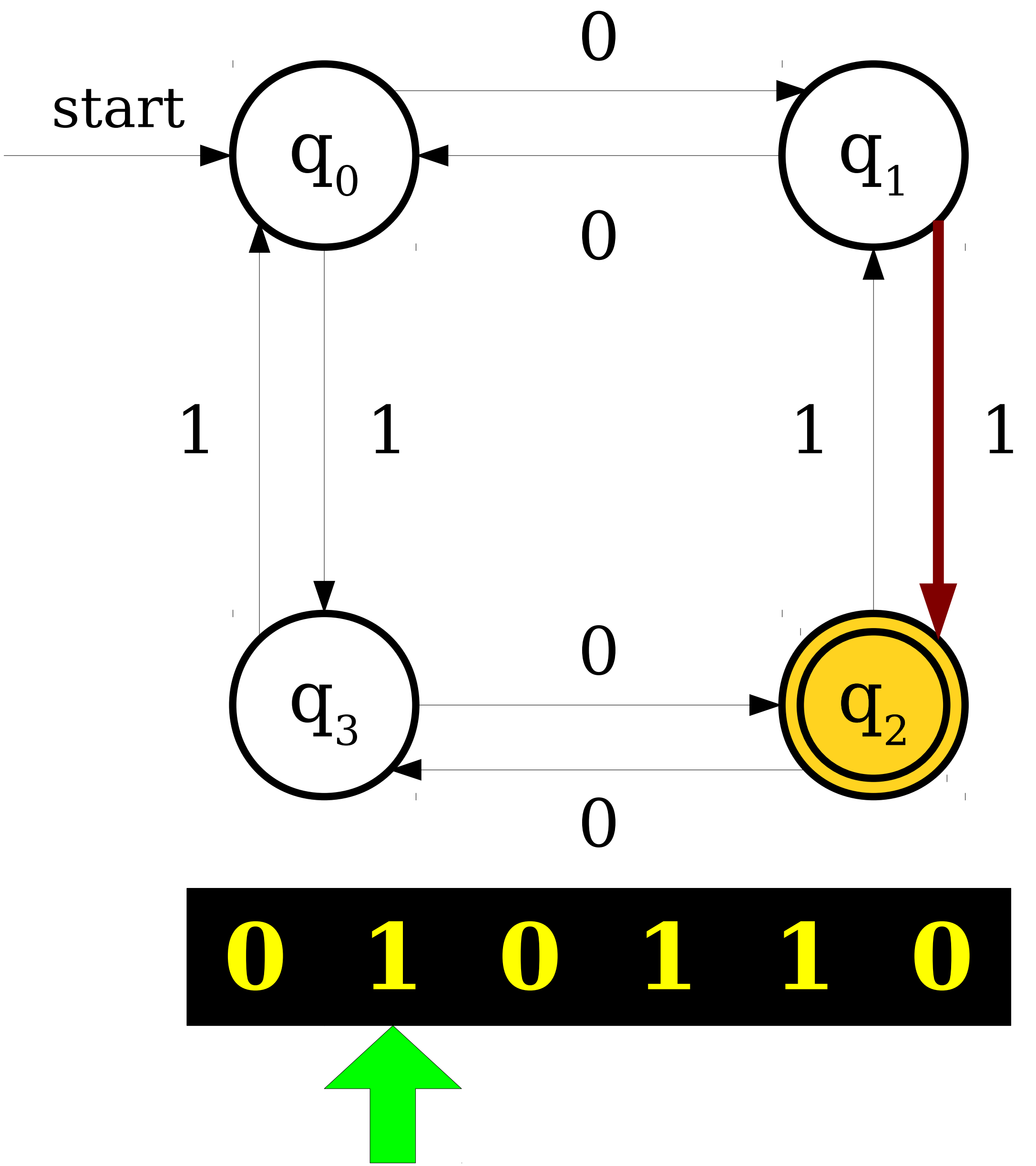
A Simple Finite Automaton



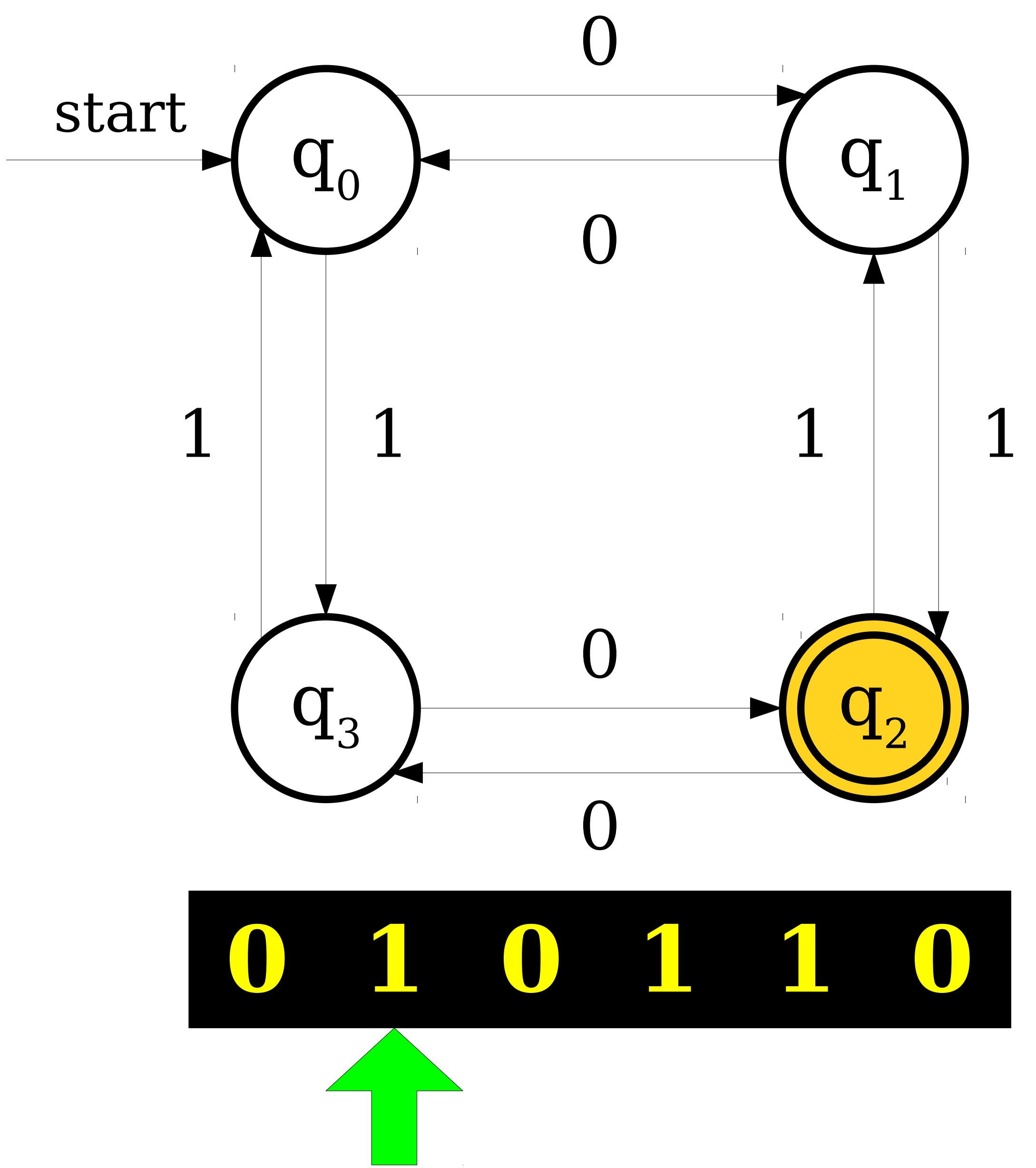
A Simple Finite Automaton



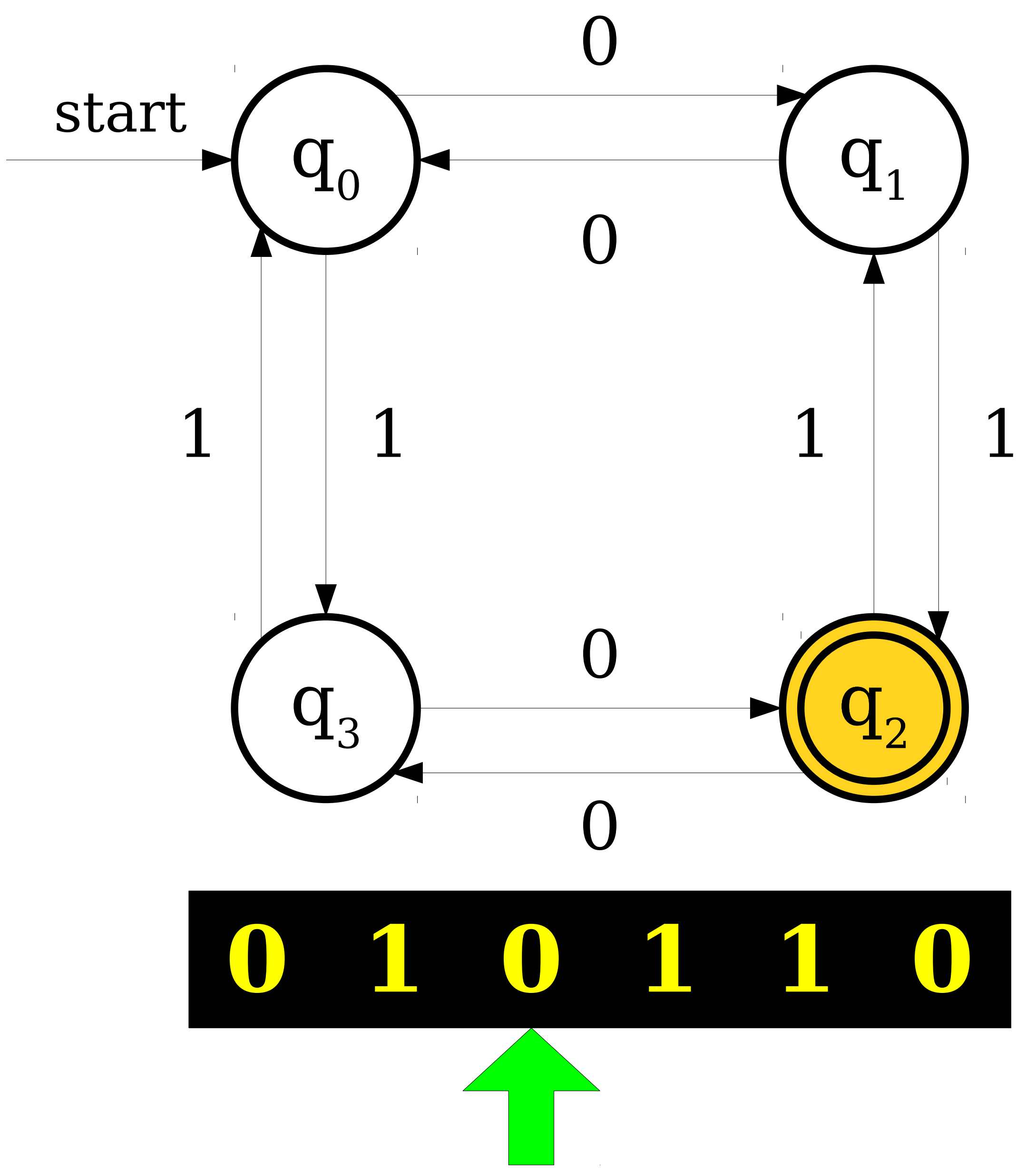
A Simple Finite Automaton



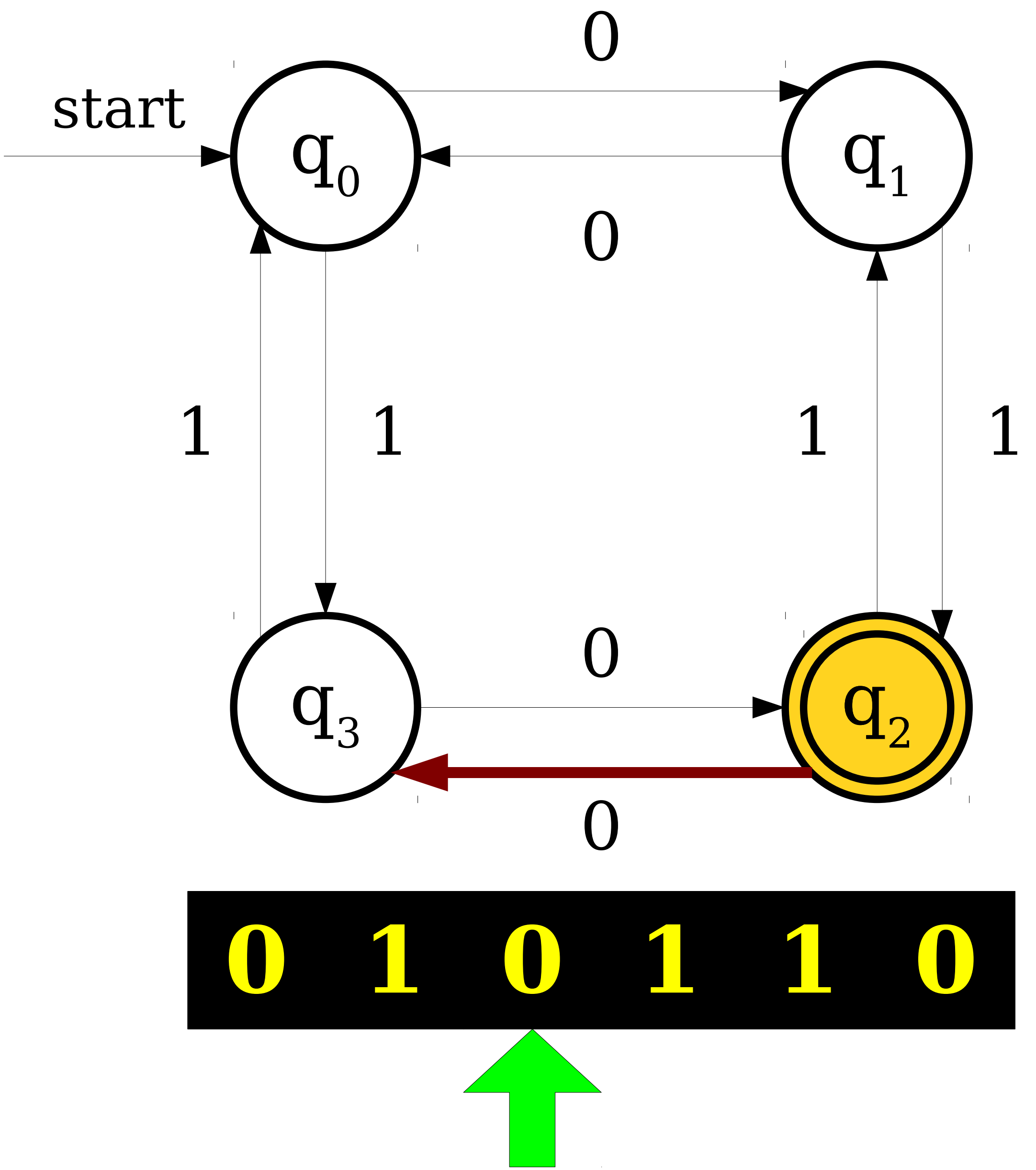
A Simple Finite Automaton



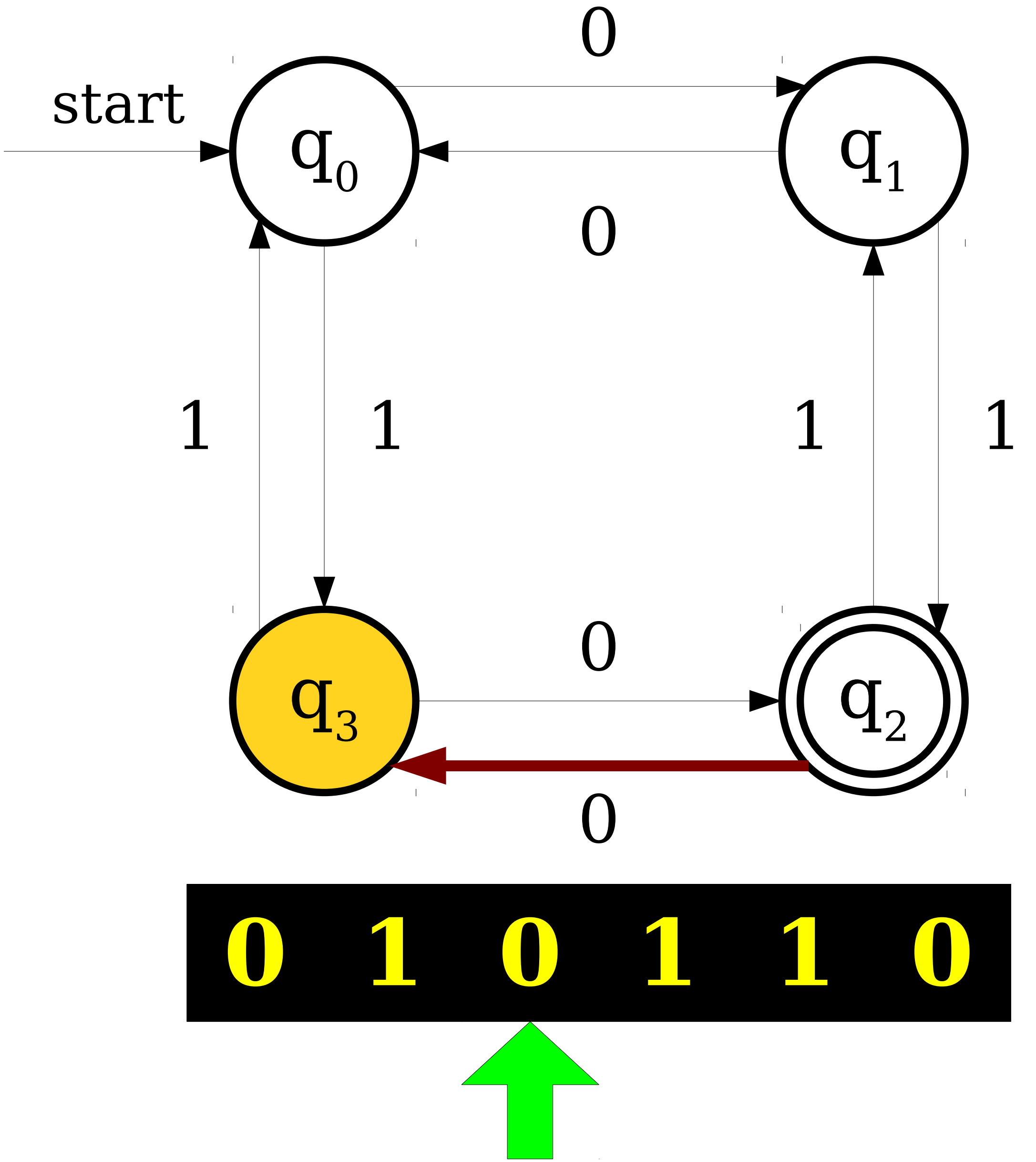
A Simple Finite Automaton



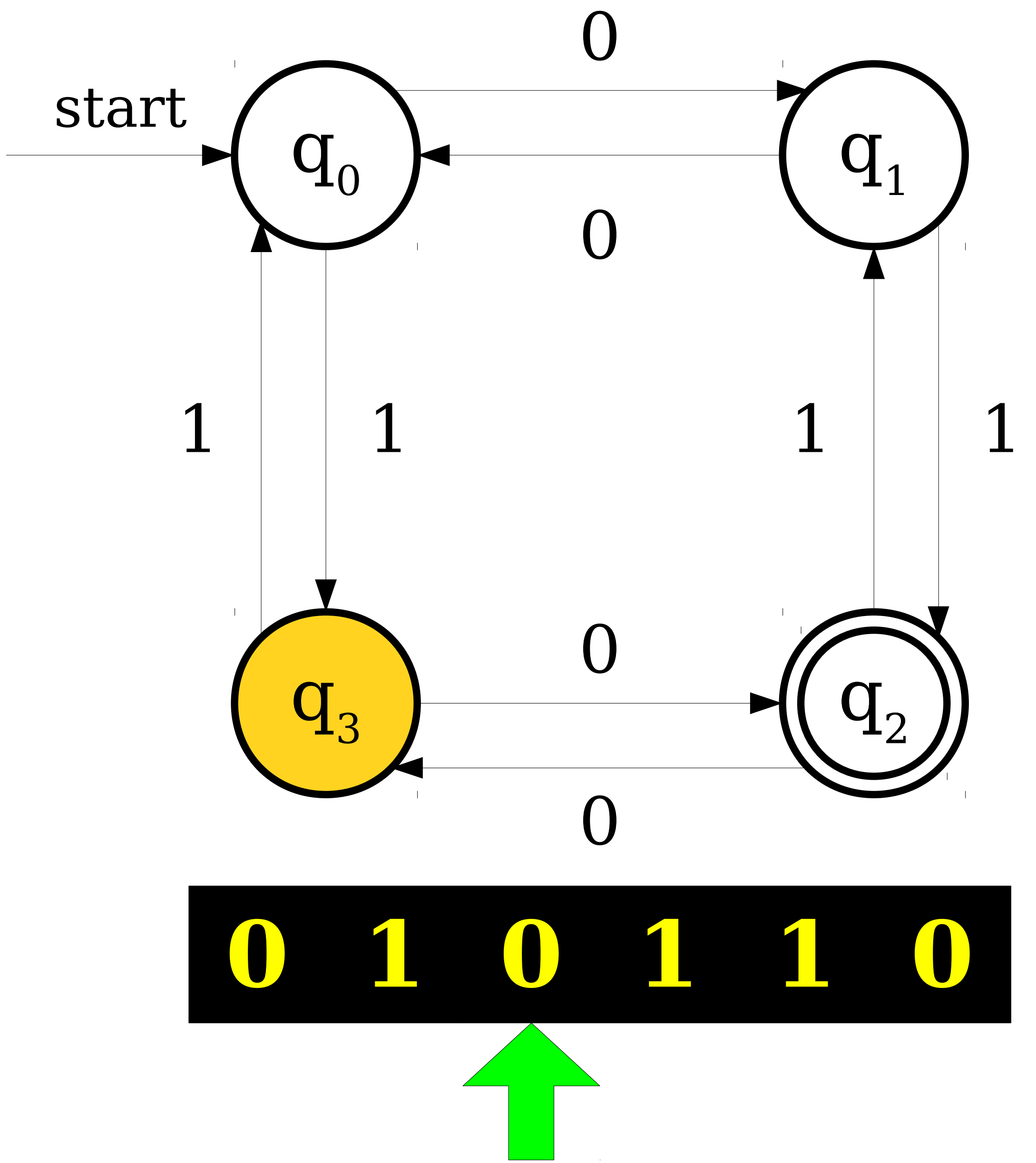
A Simple Finite Automaton



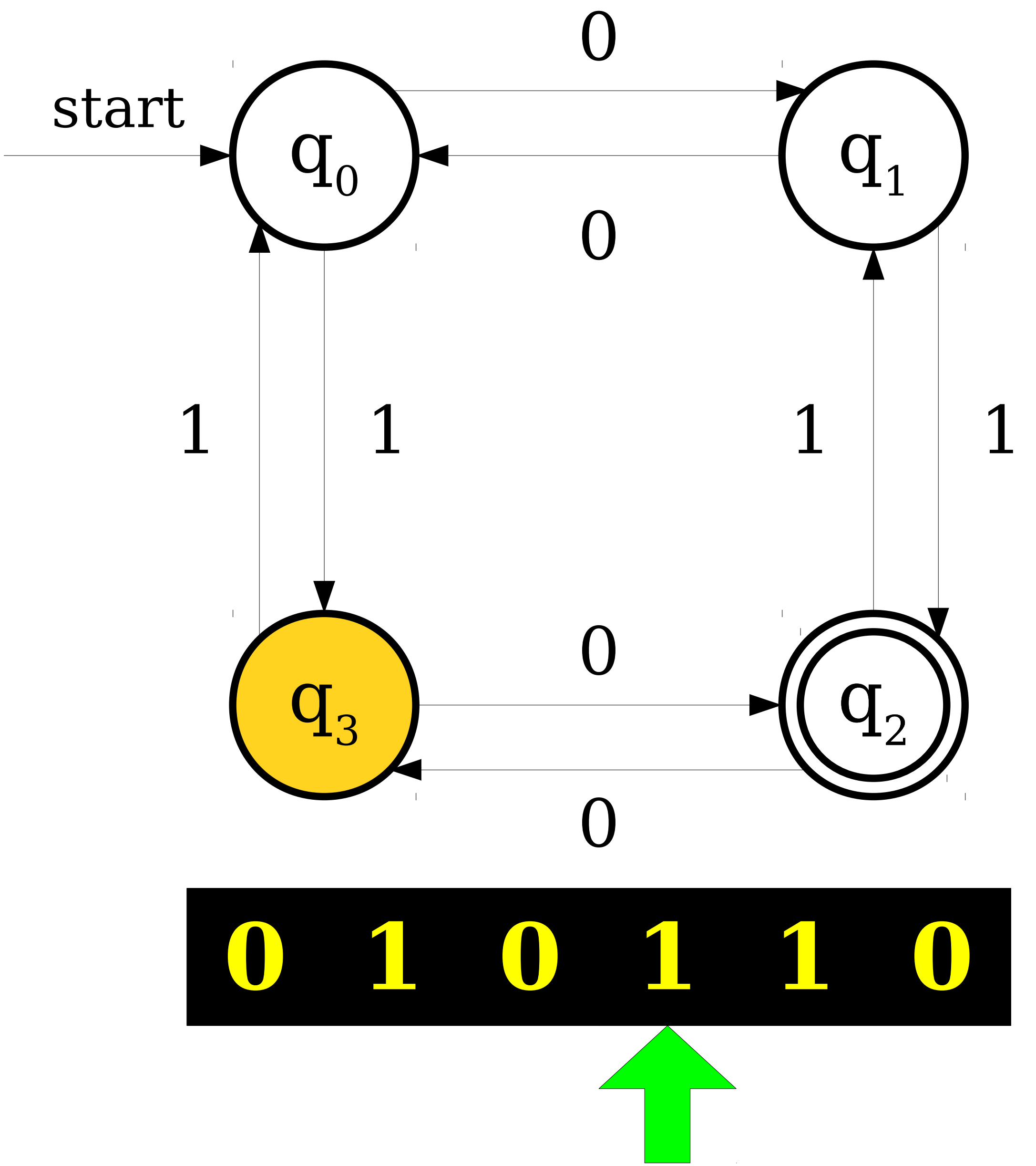
A Simple Finite Automaton



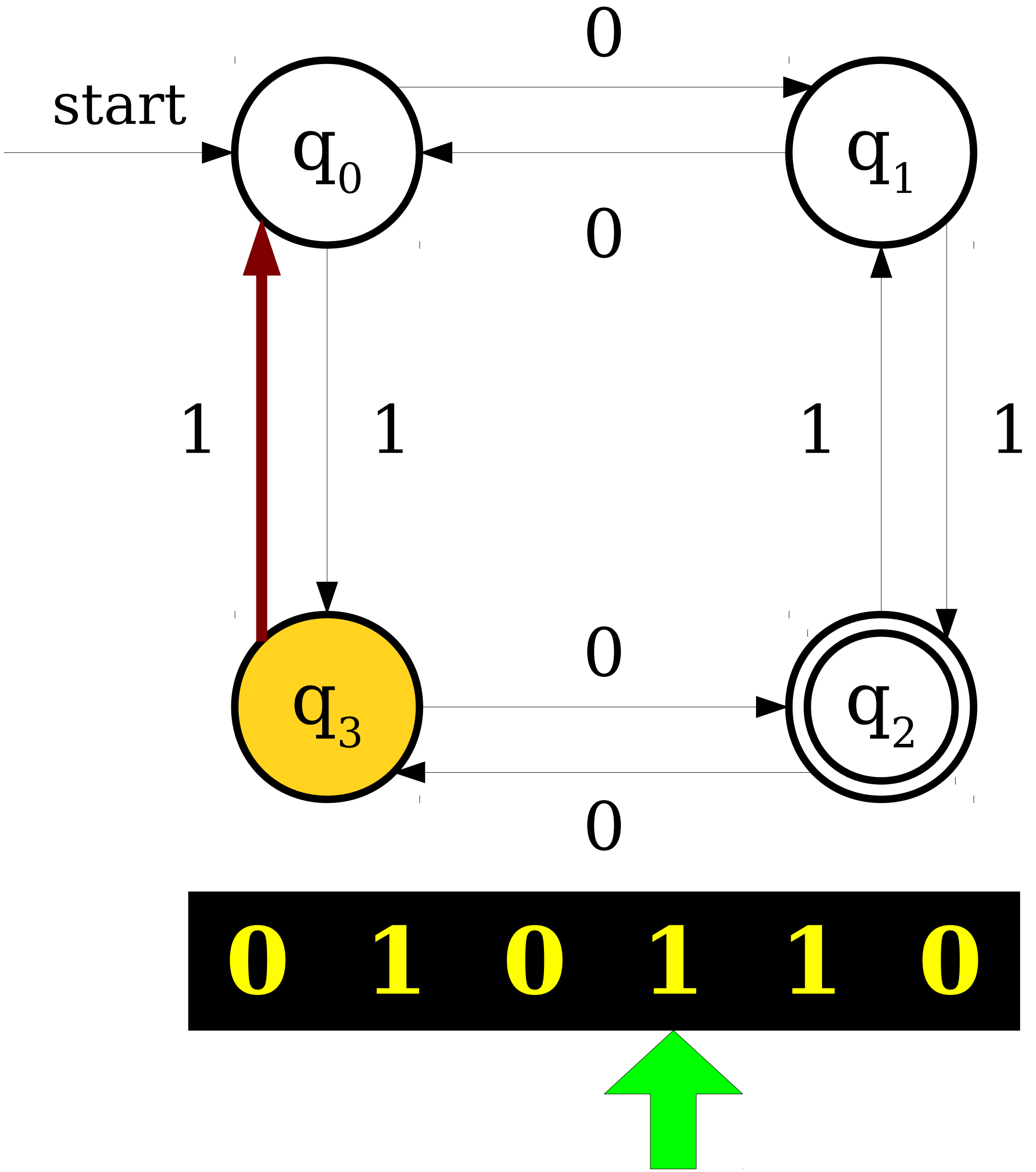
A Simple Finite Automaton



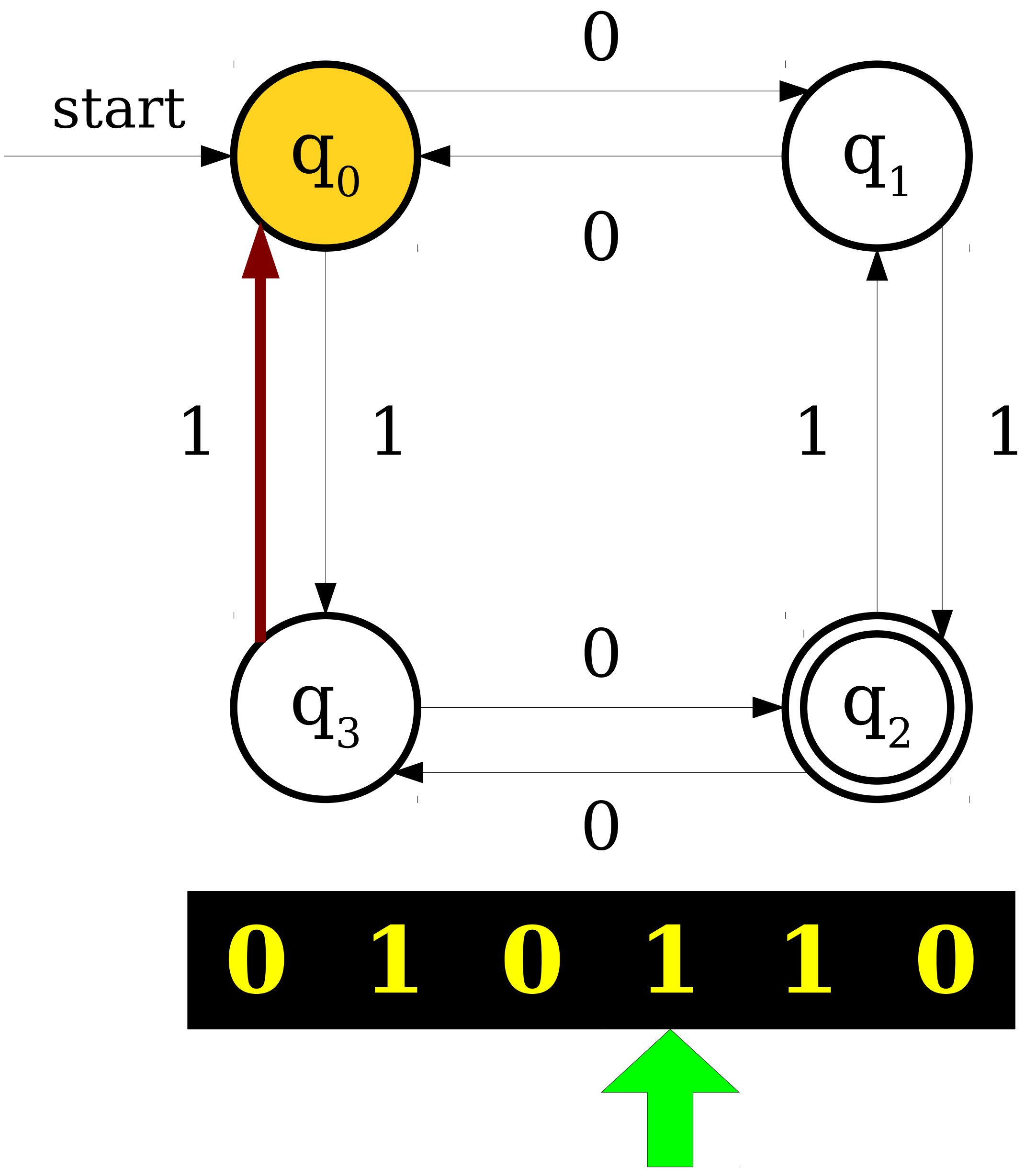
A Simple Finite Automaton



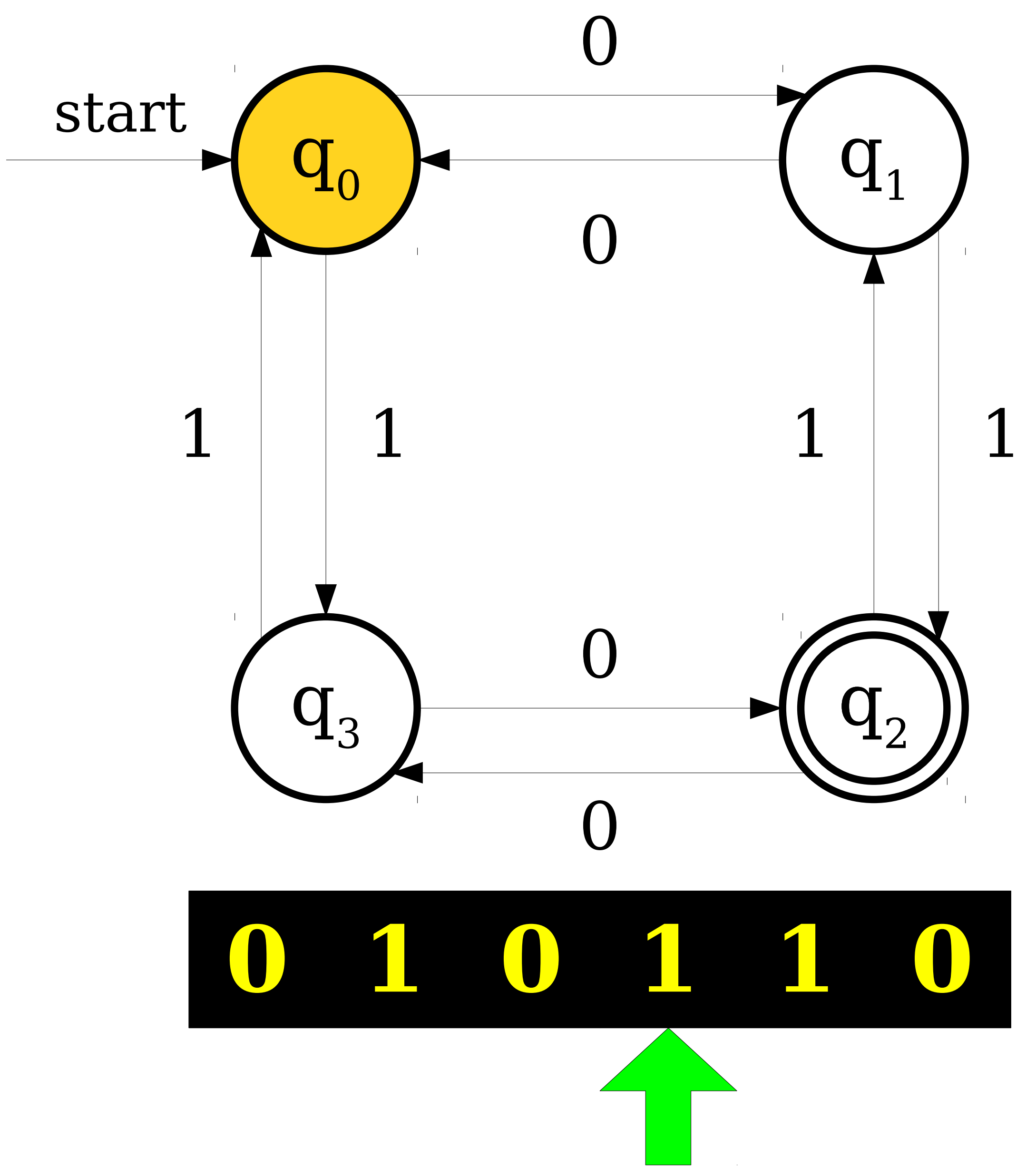
A Simple Finite Automaton



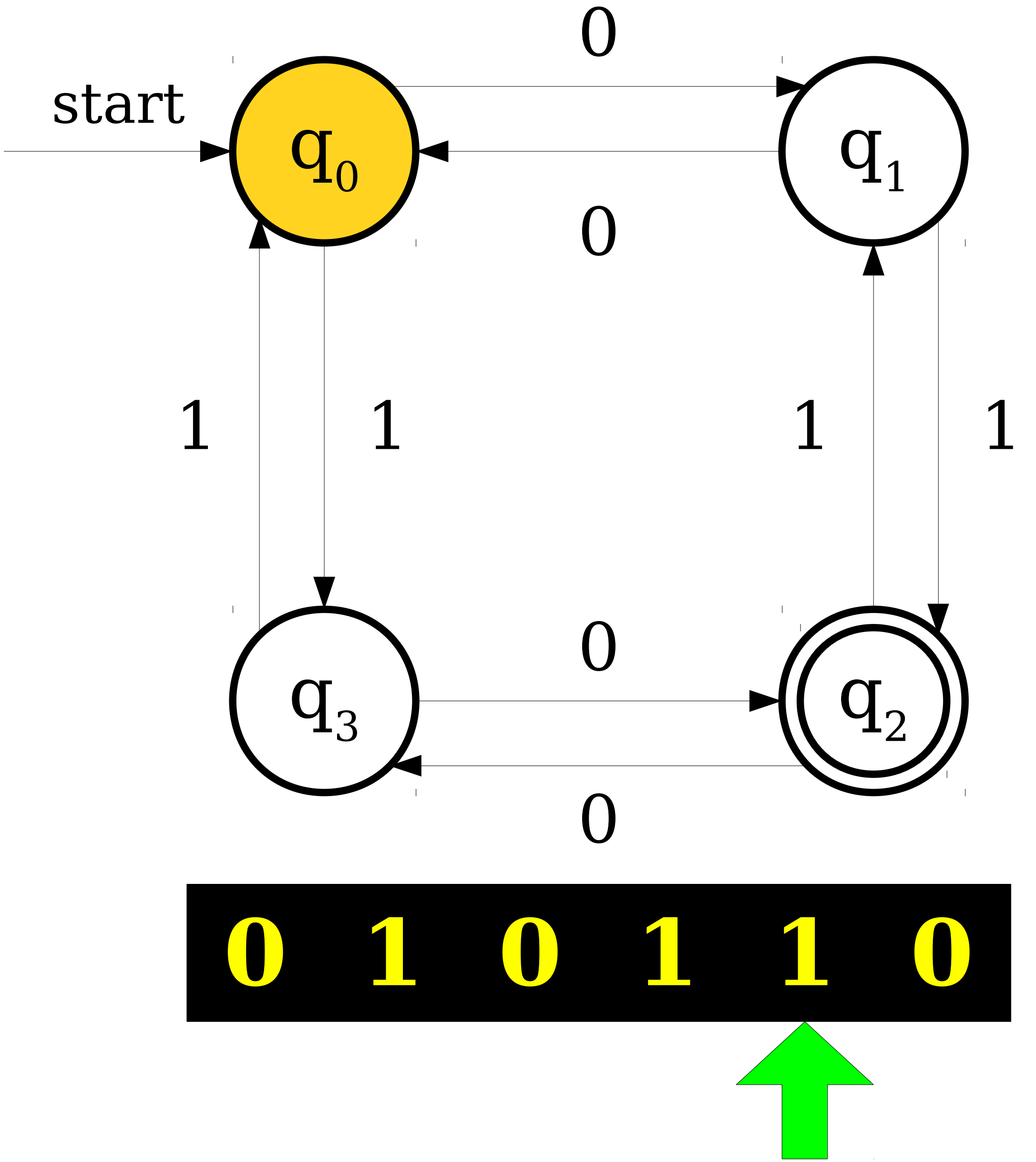
A Simple Finite Automaton



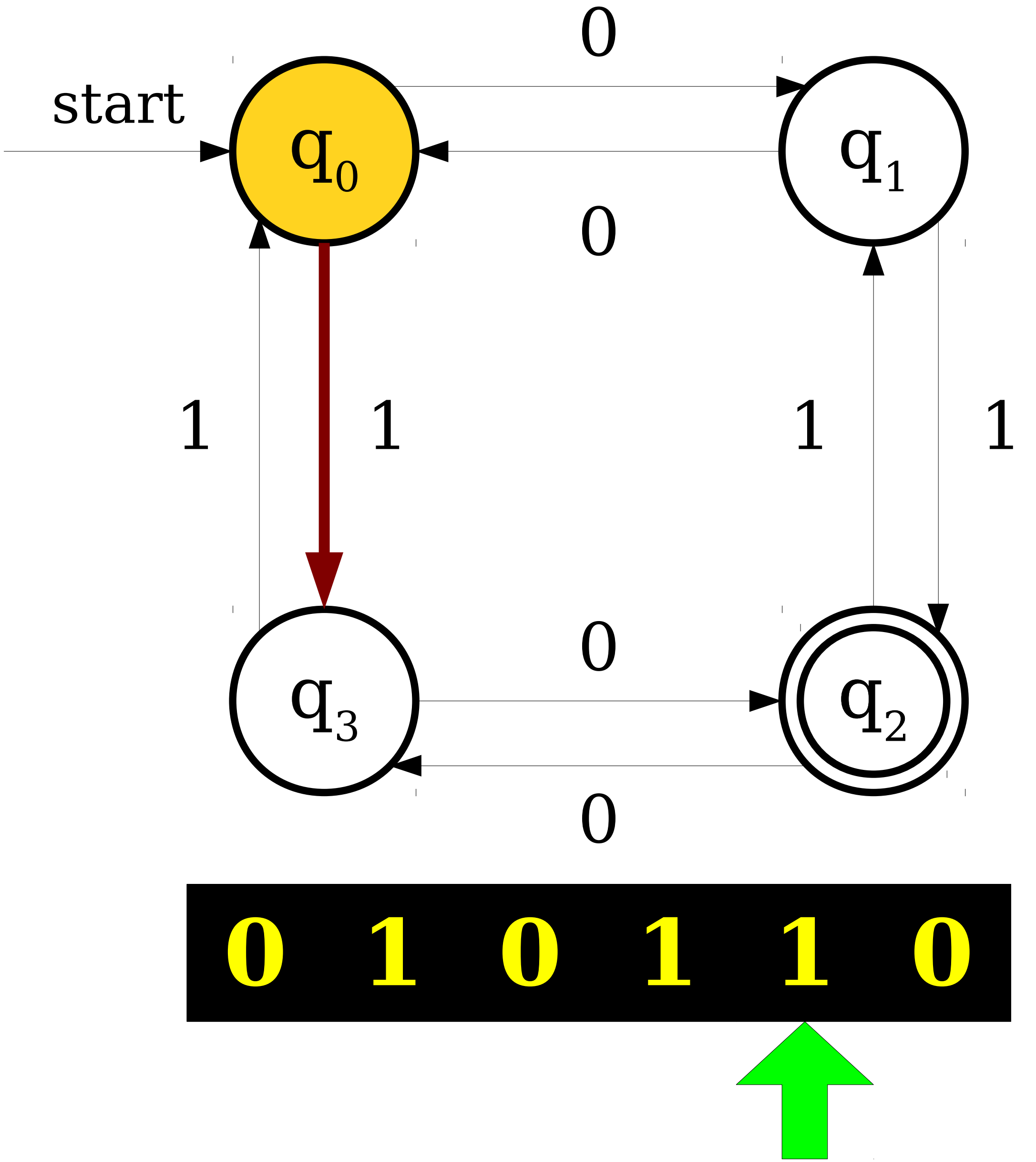
A Simple Finite Automaton



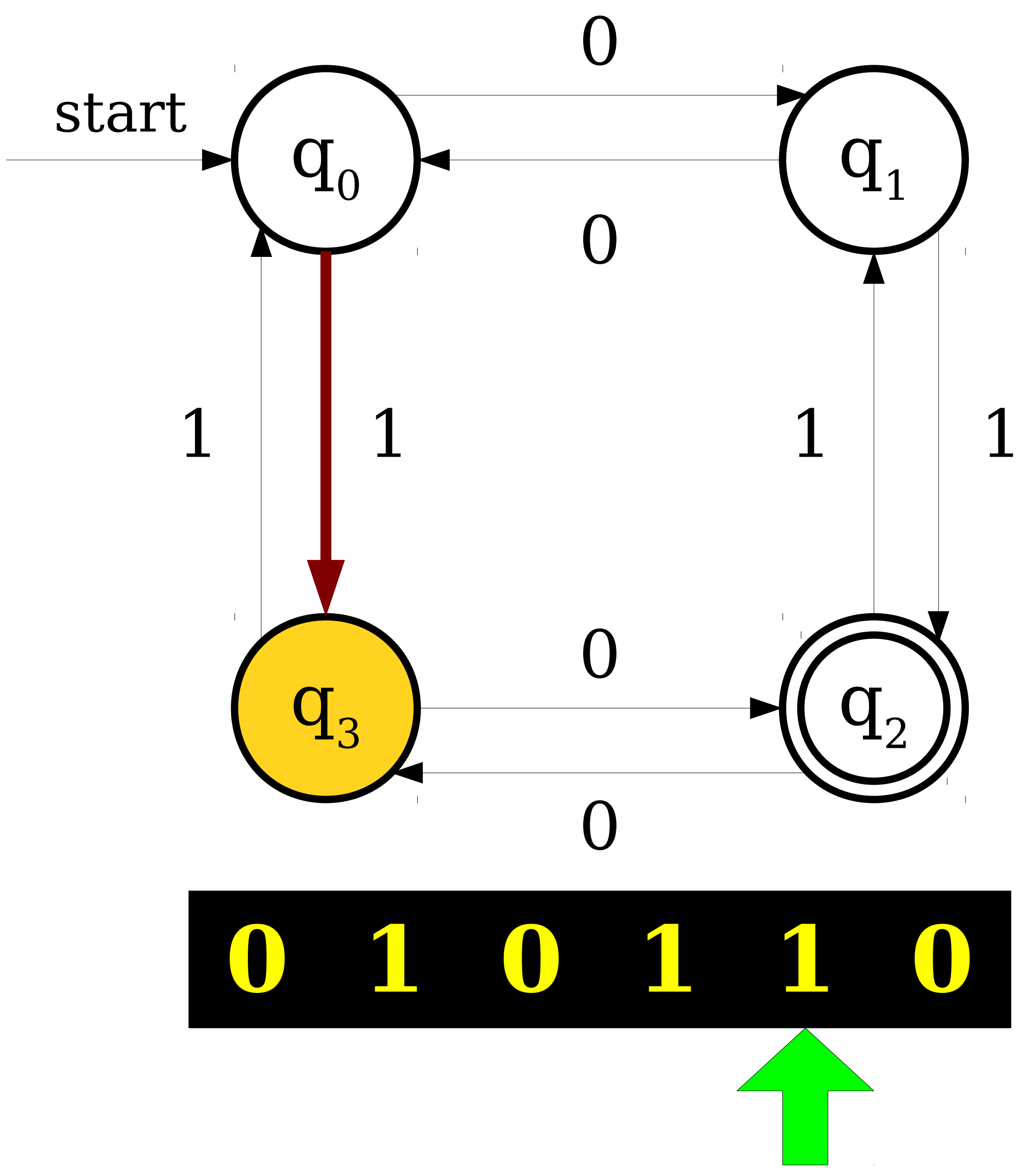
A Simple Finite Automaton



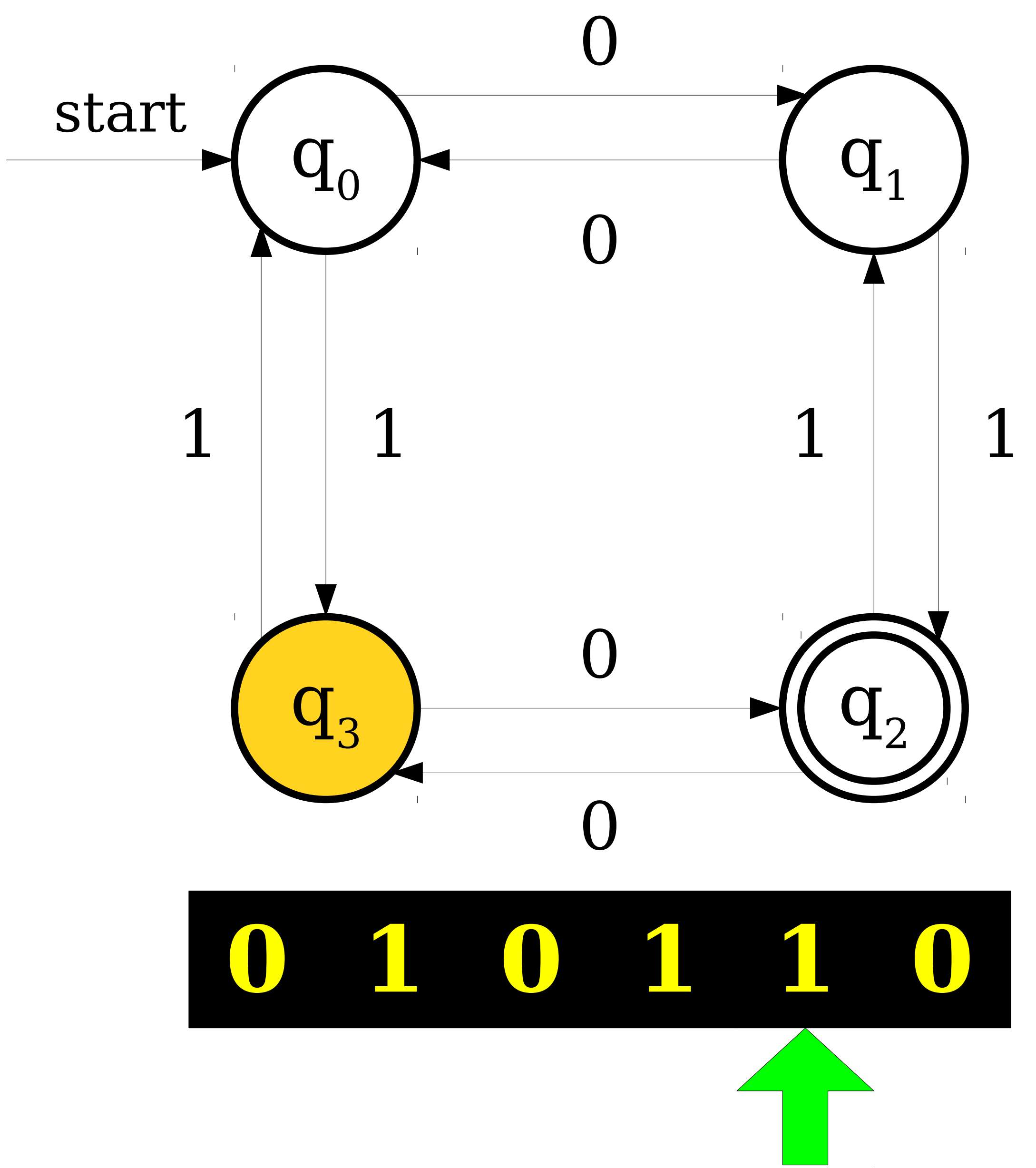
A Simple Finite Automaton



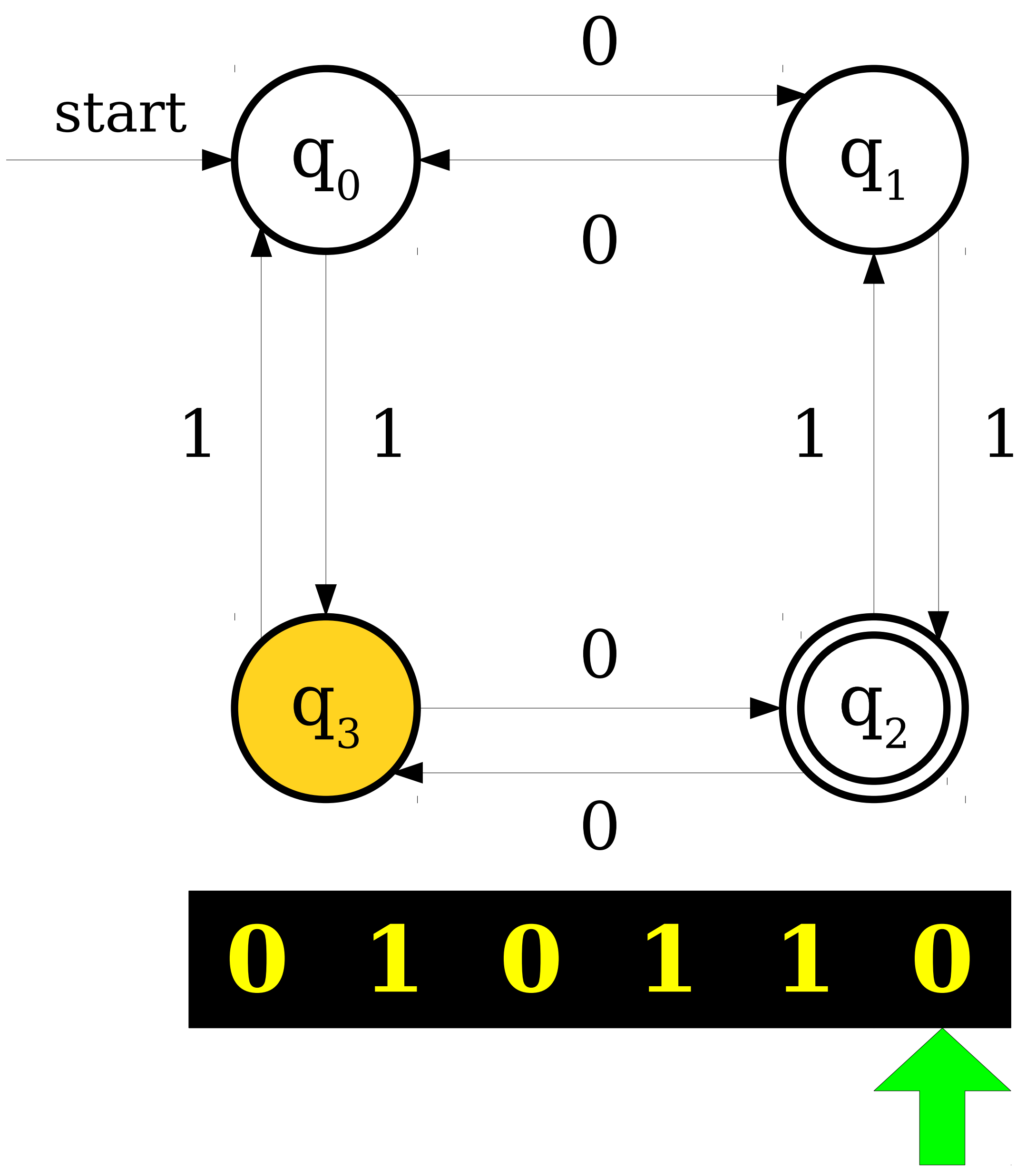
A Simple Finite Automaton



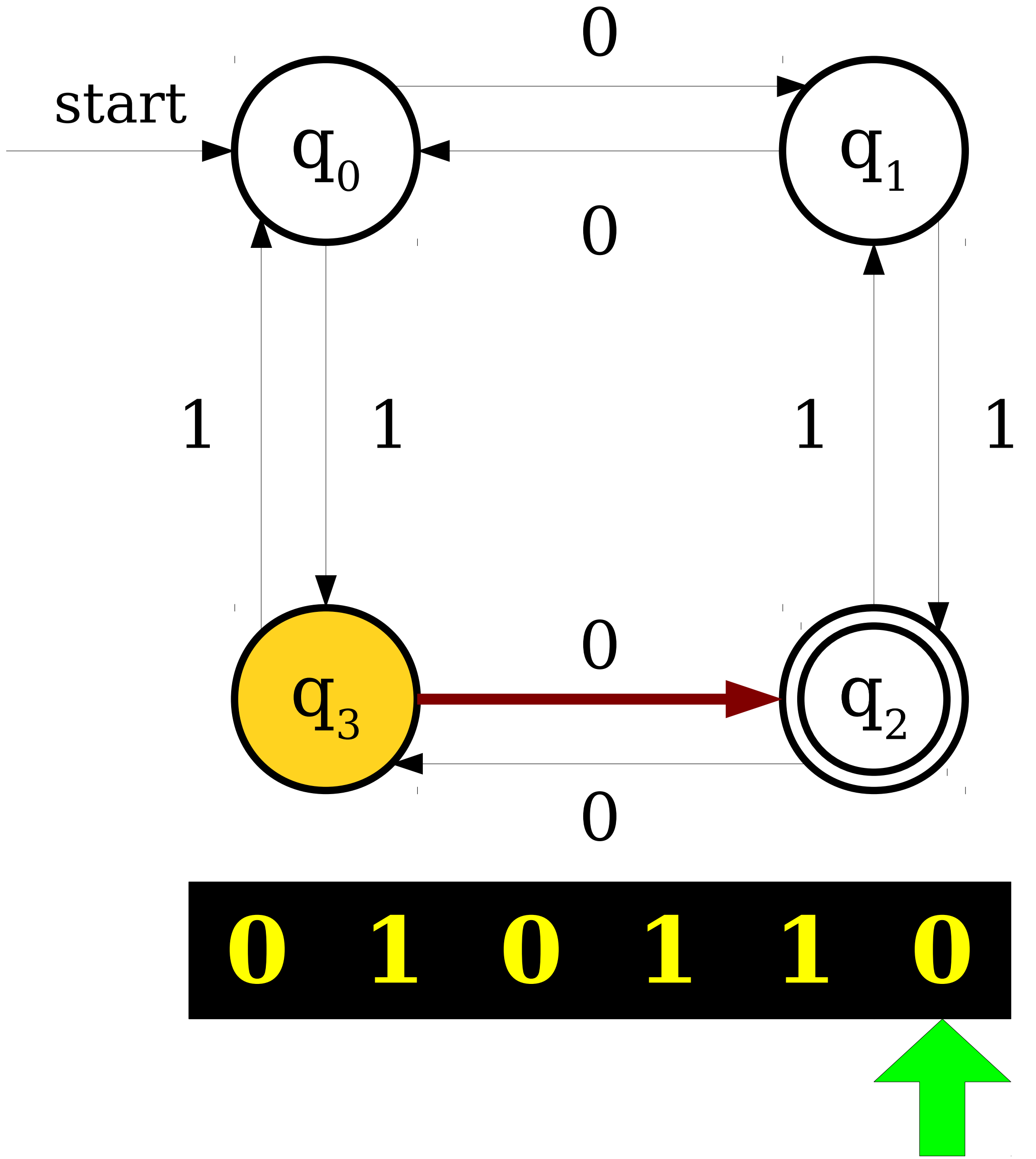
A Simple Finite Automaton



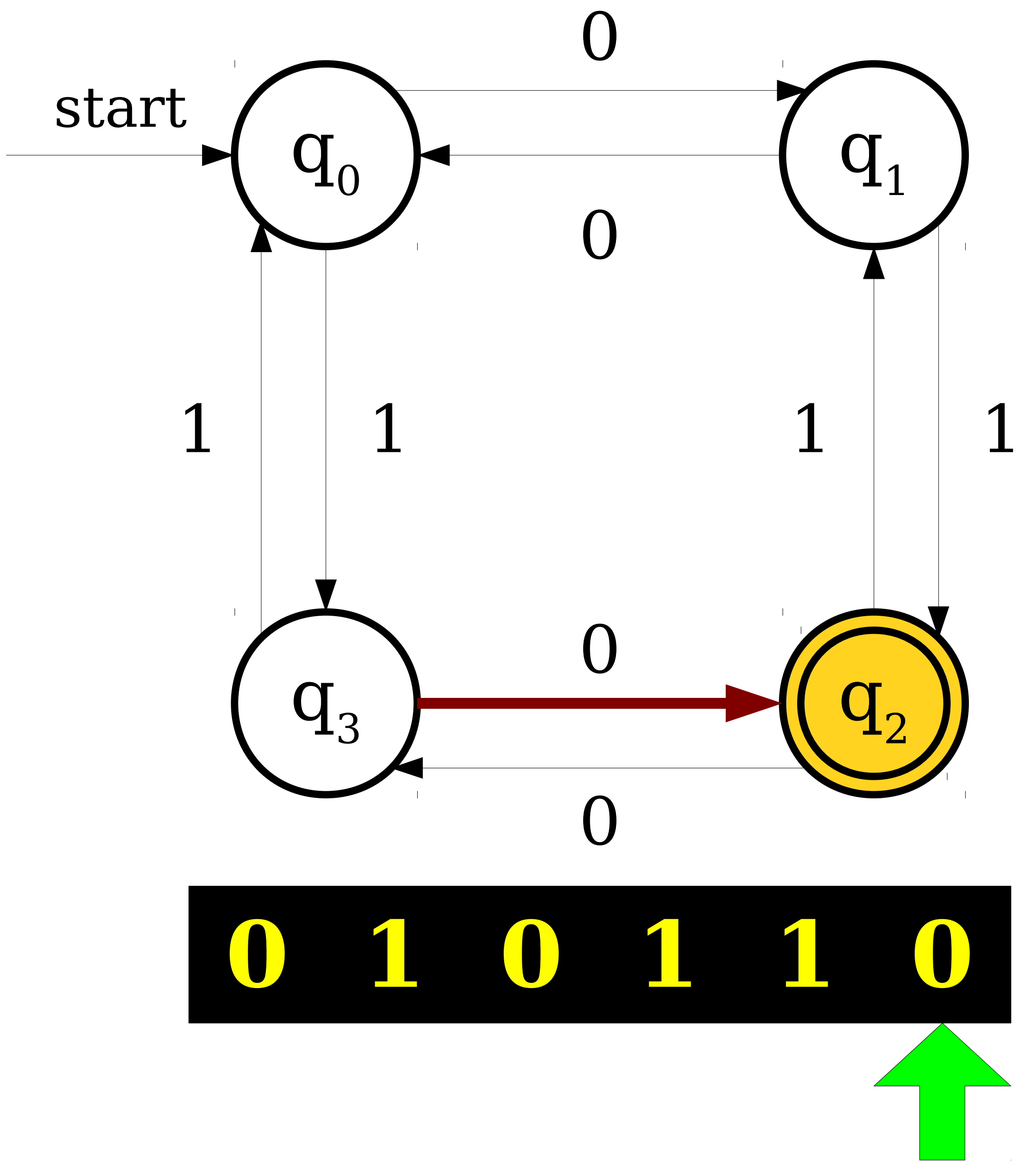
A Simple Finite Automaton



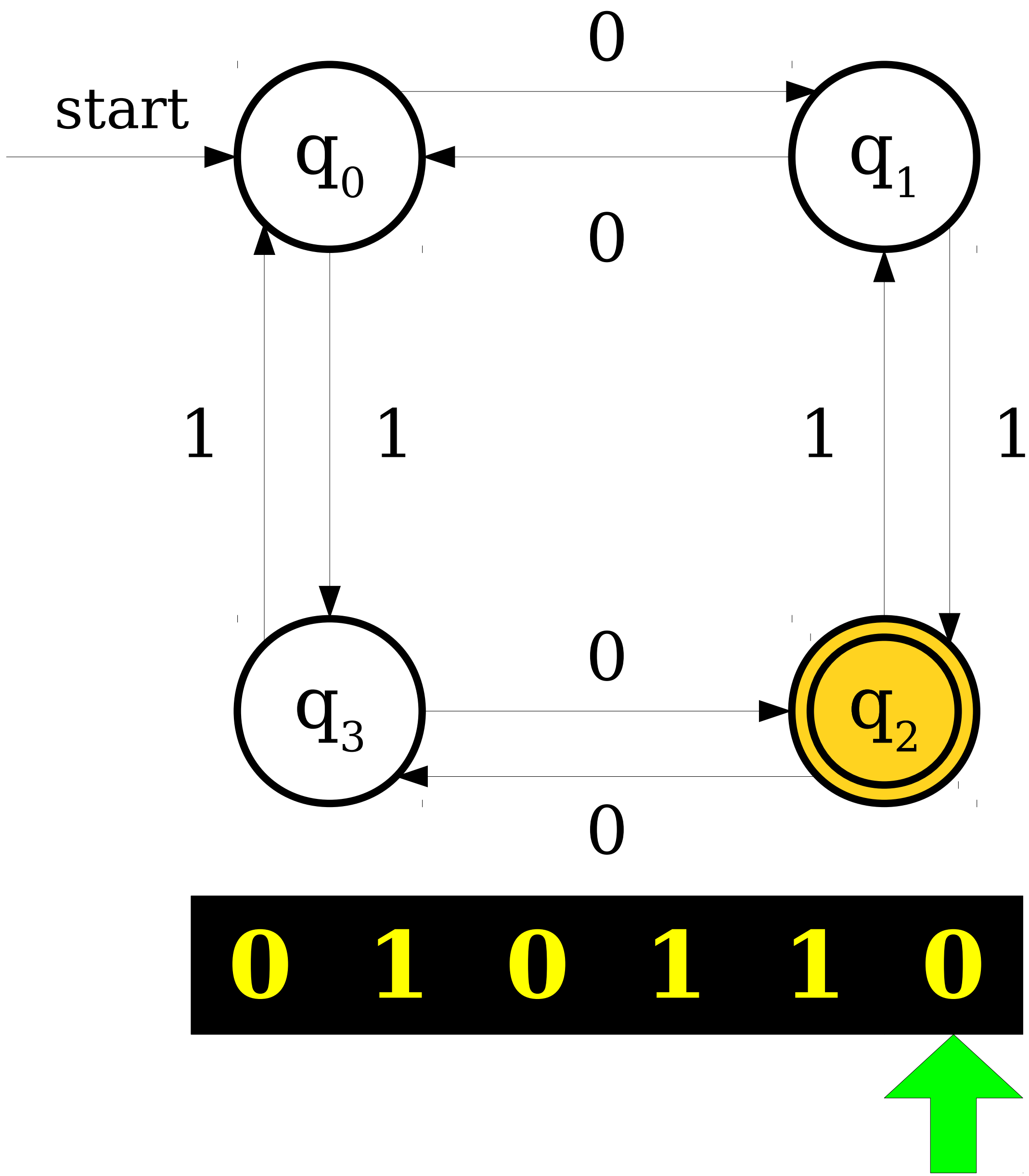
A Simple Finite Automaton



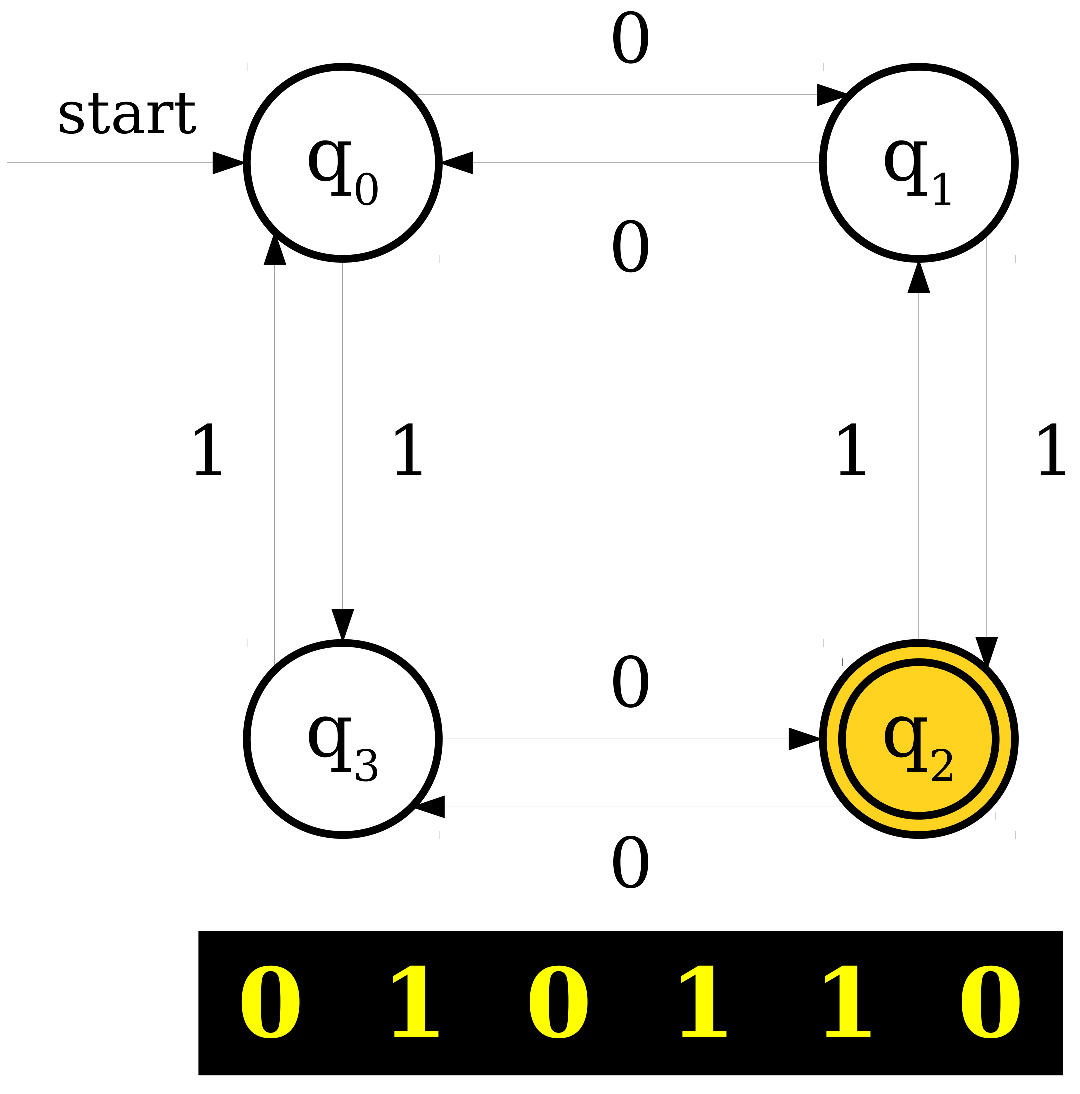
A Simple Finite Automaton



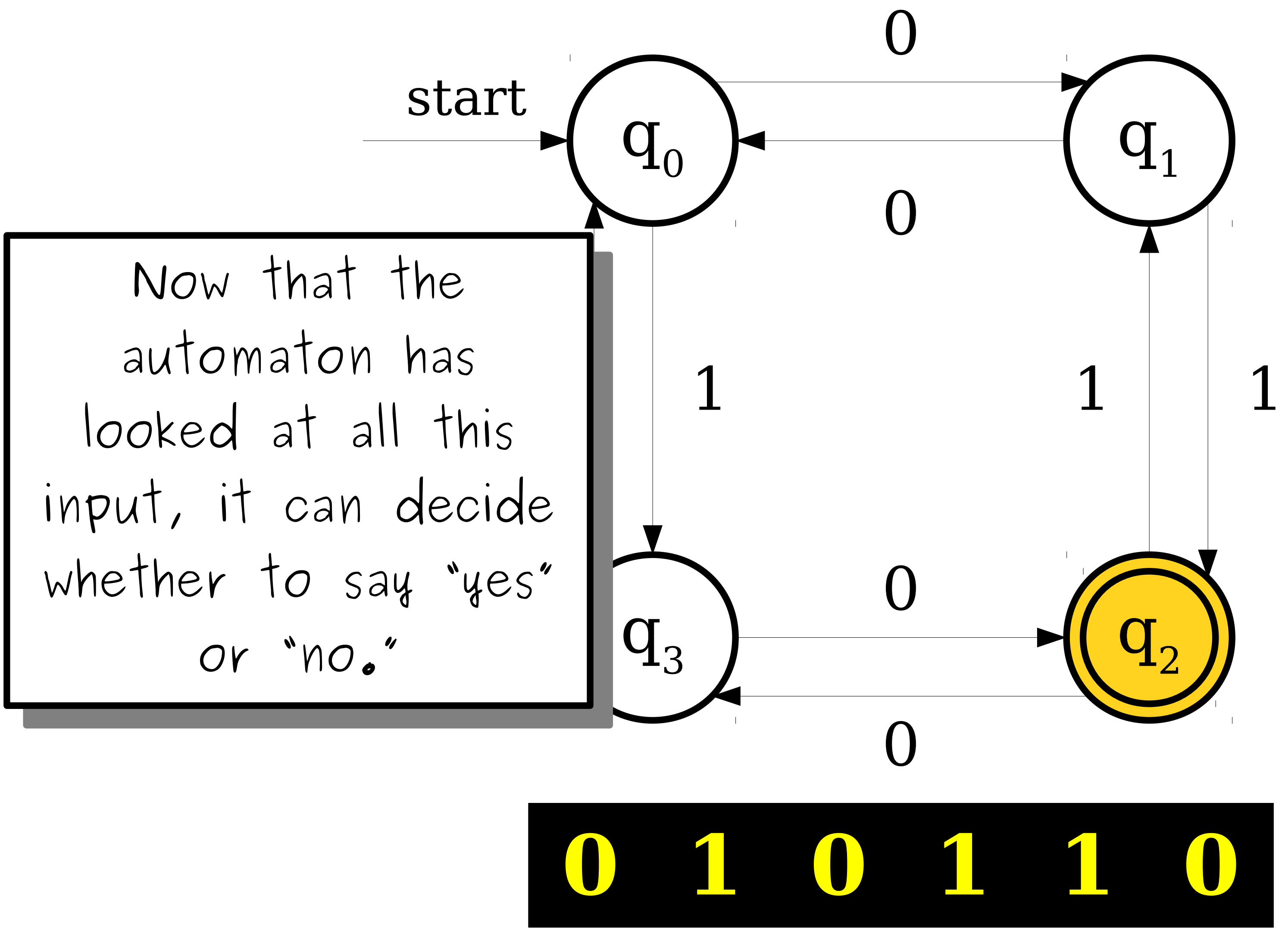
A Simple Finite Automaton



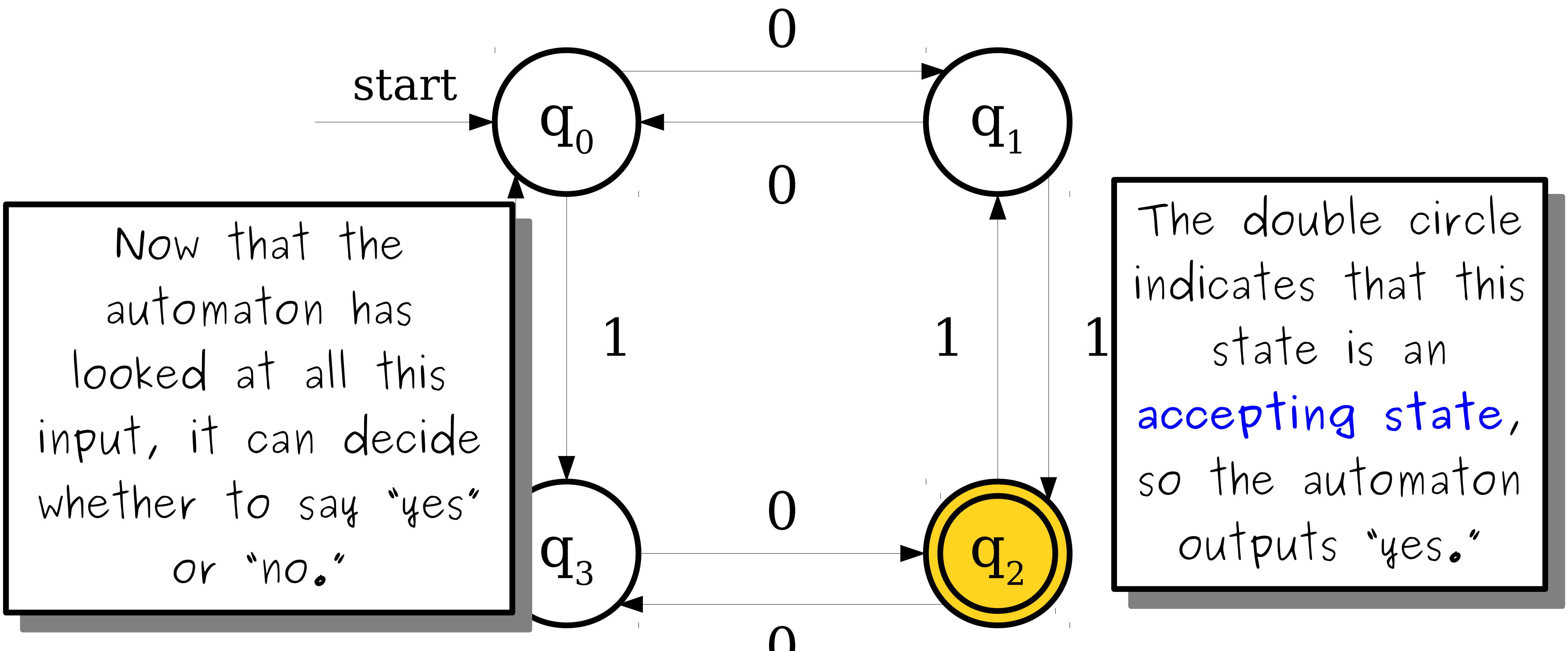
A Simple Finite Automaton



A Simple Finite Automaton

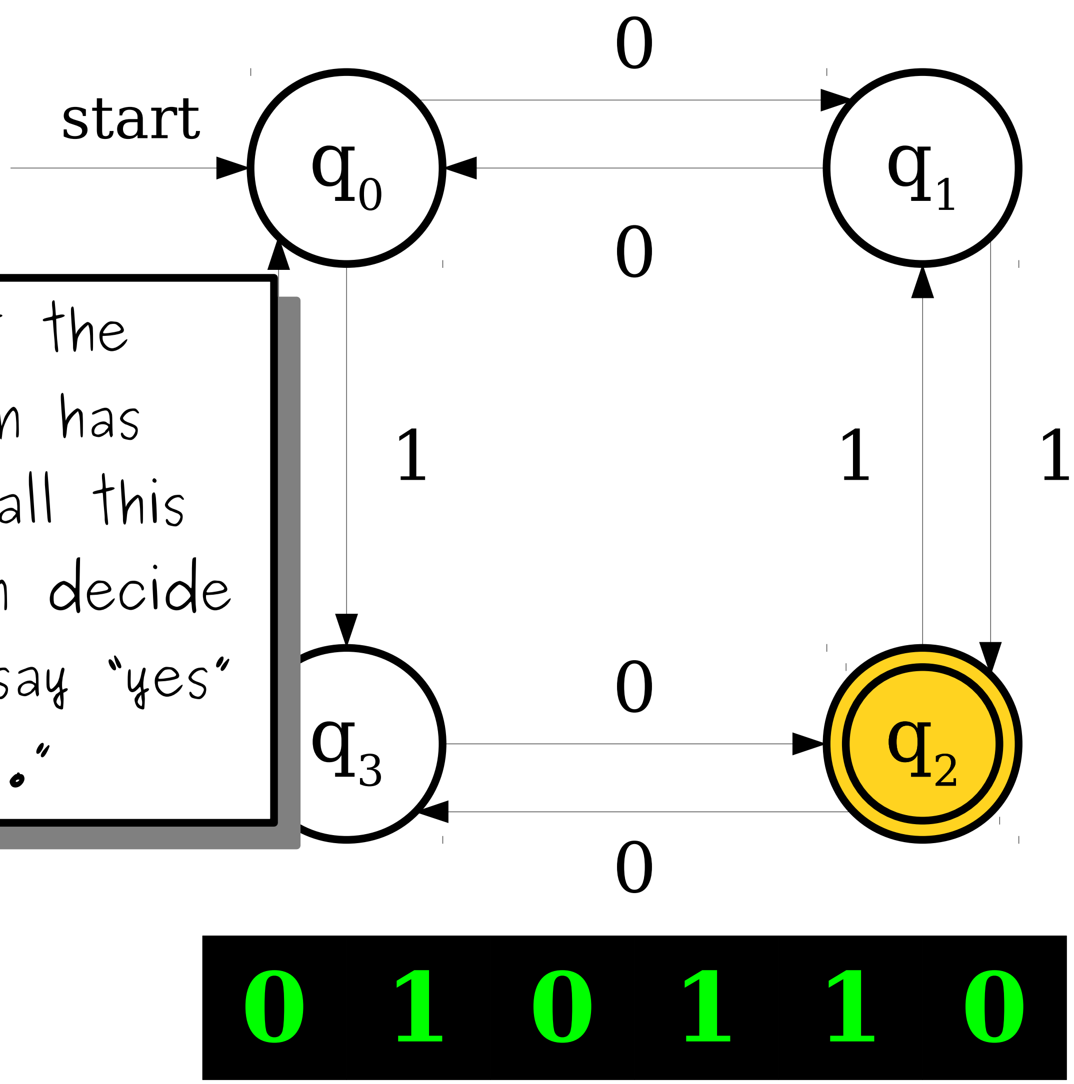


A Simple Finite Automaton



0 1 0 1 1 0

A Simple Finite Automaton

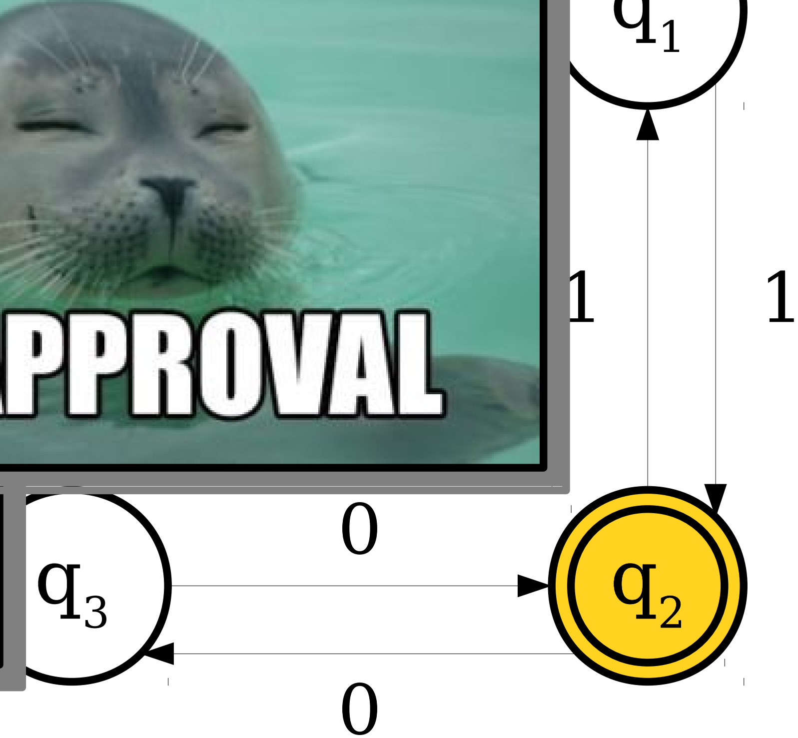


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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

A Simple Finite Automaton

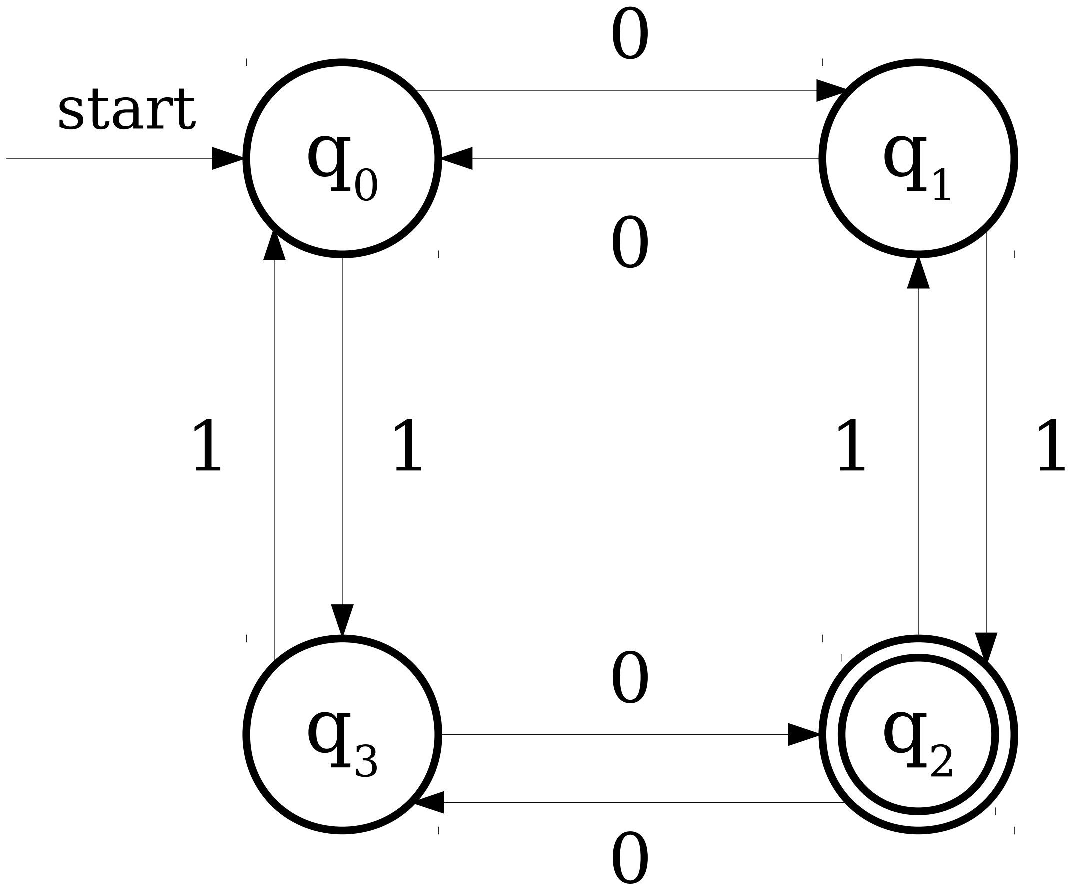
Now t
automa
looked
input, it
whether to say “yes”
or “no.”



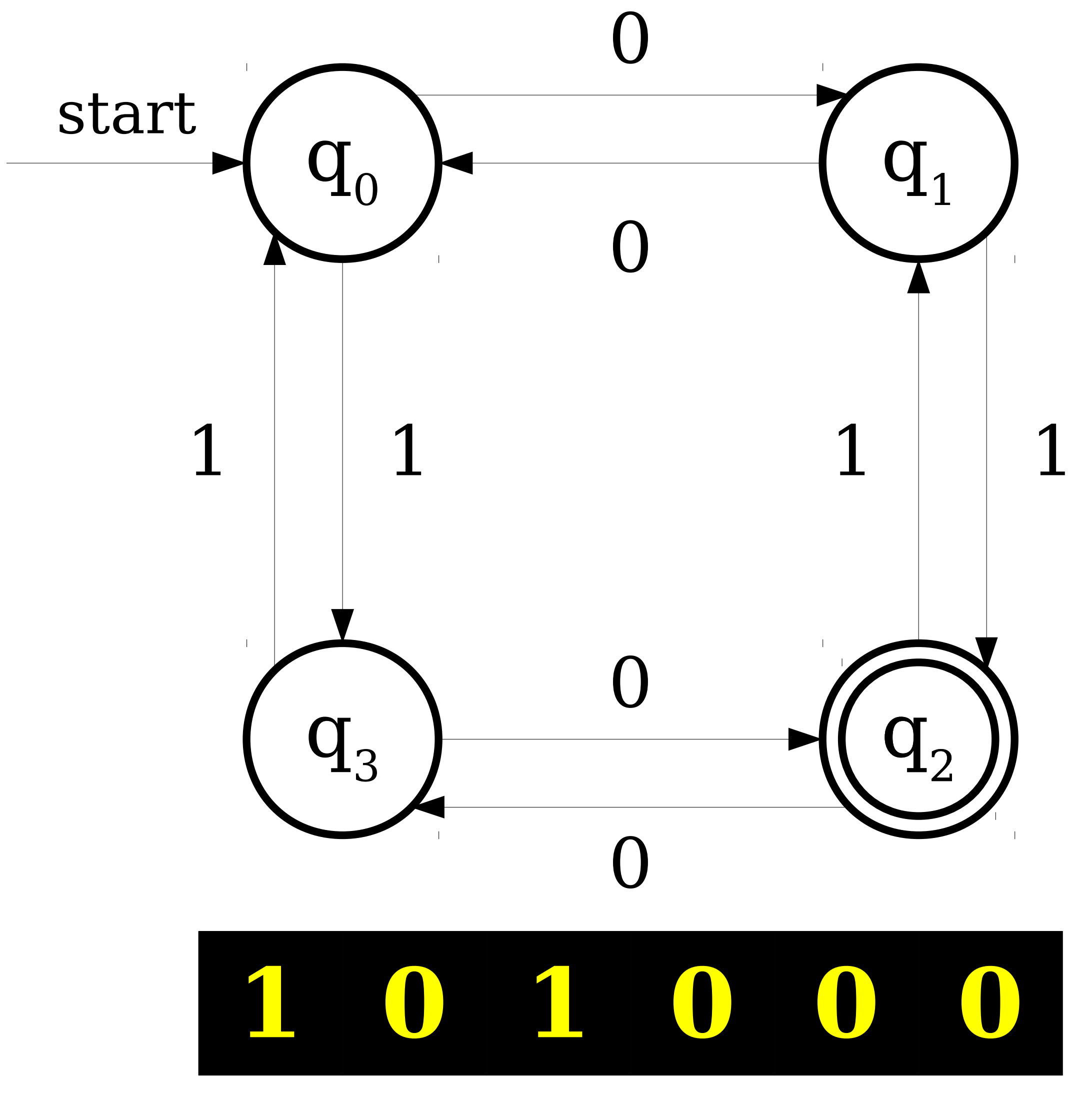
The double circle indicates that this state is an accepting state, so the automaton outputs “yes.”

0 1 0 1 1 0

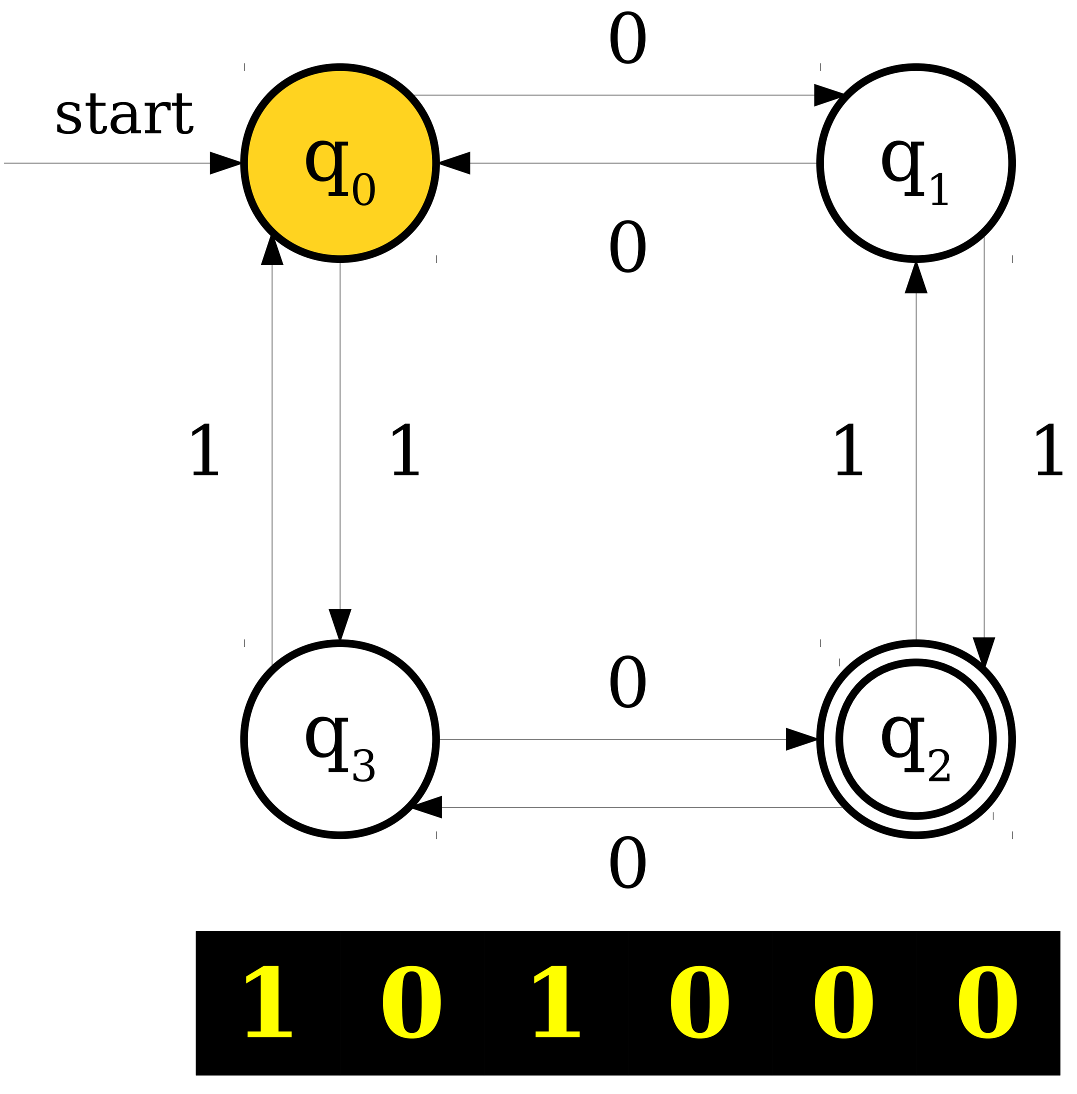
A Simple Finite Automaton



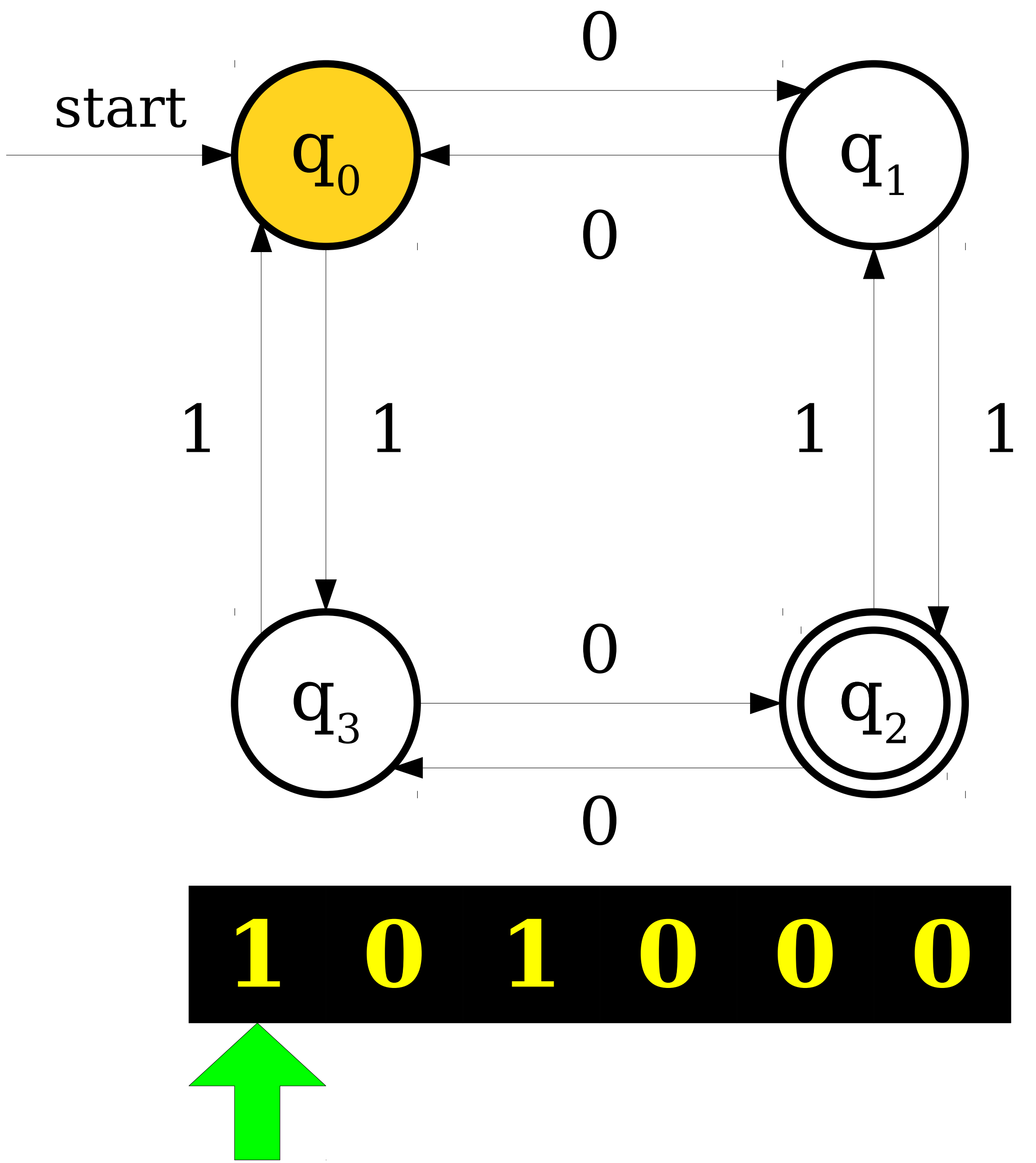
A Simple Finite Automaton



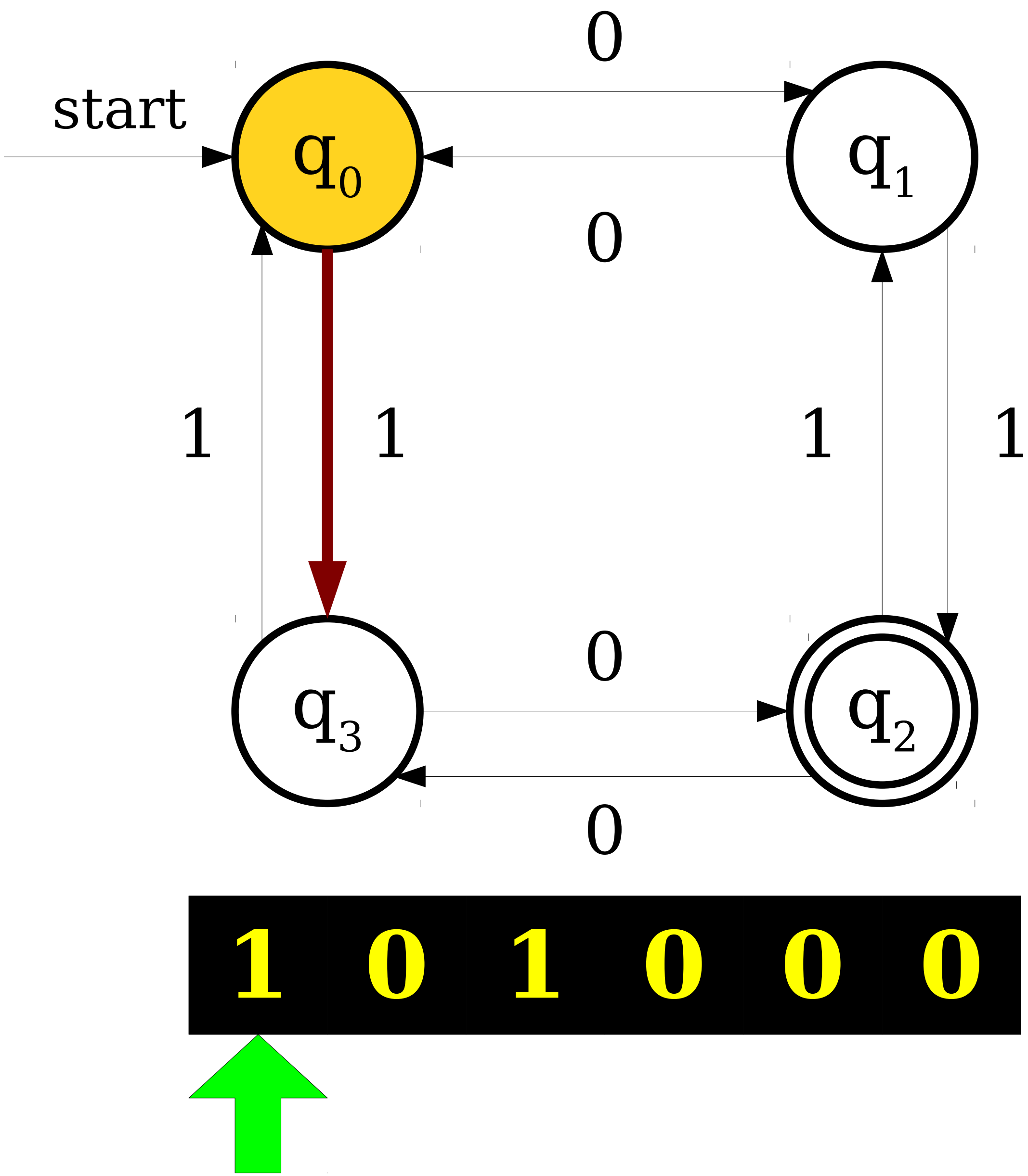
A Simple Finite Automaton



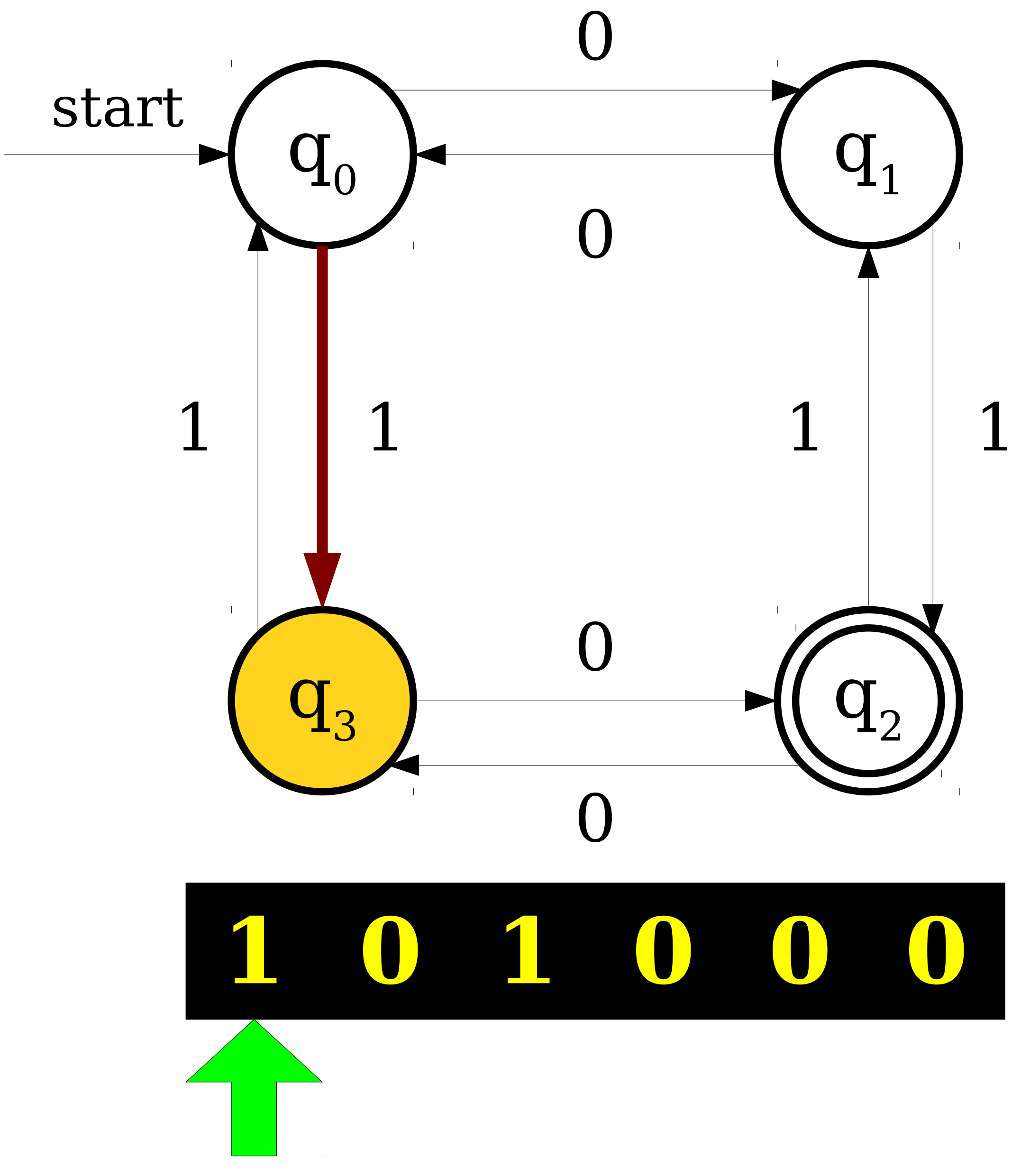
A Simple Finite Automaton



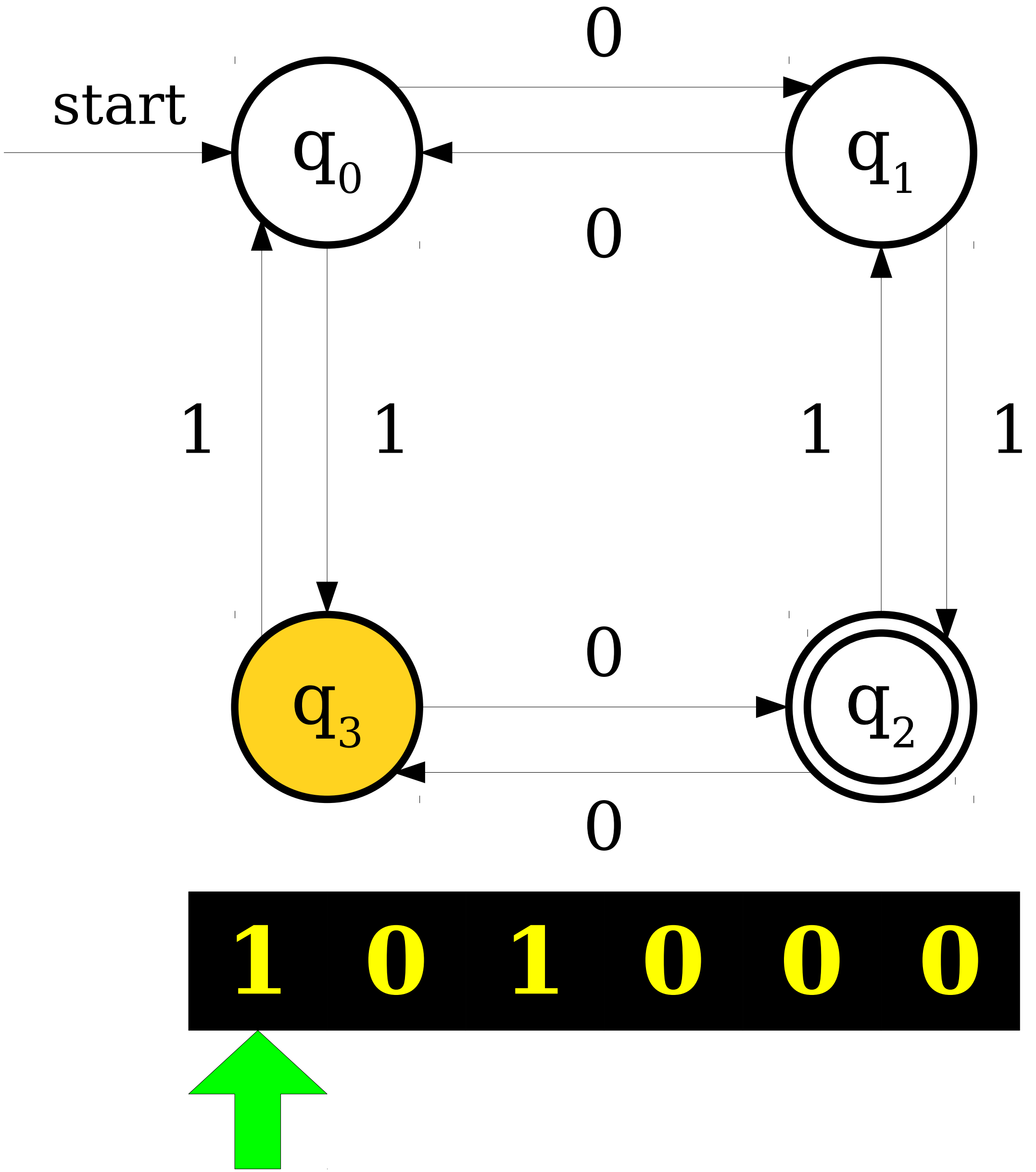
A Simple Finite Automaton



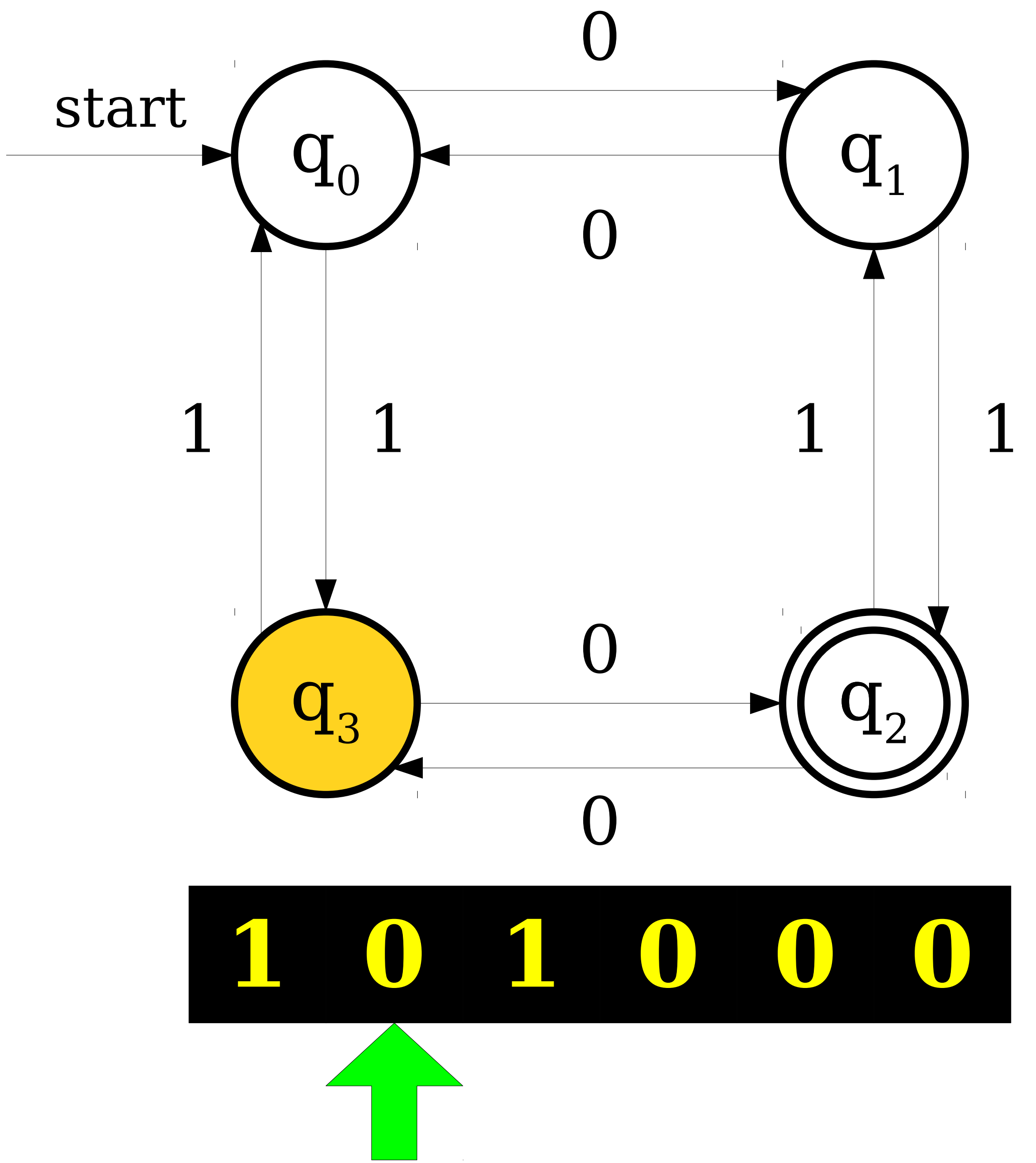
A Simple Finite Automaton



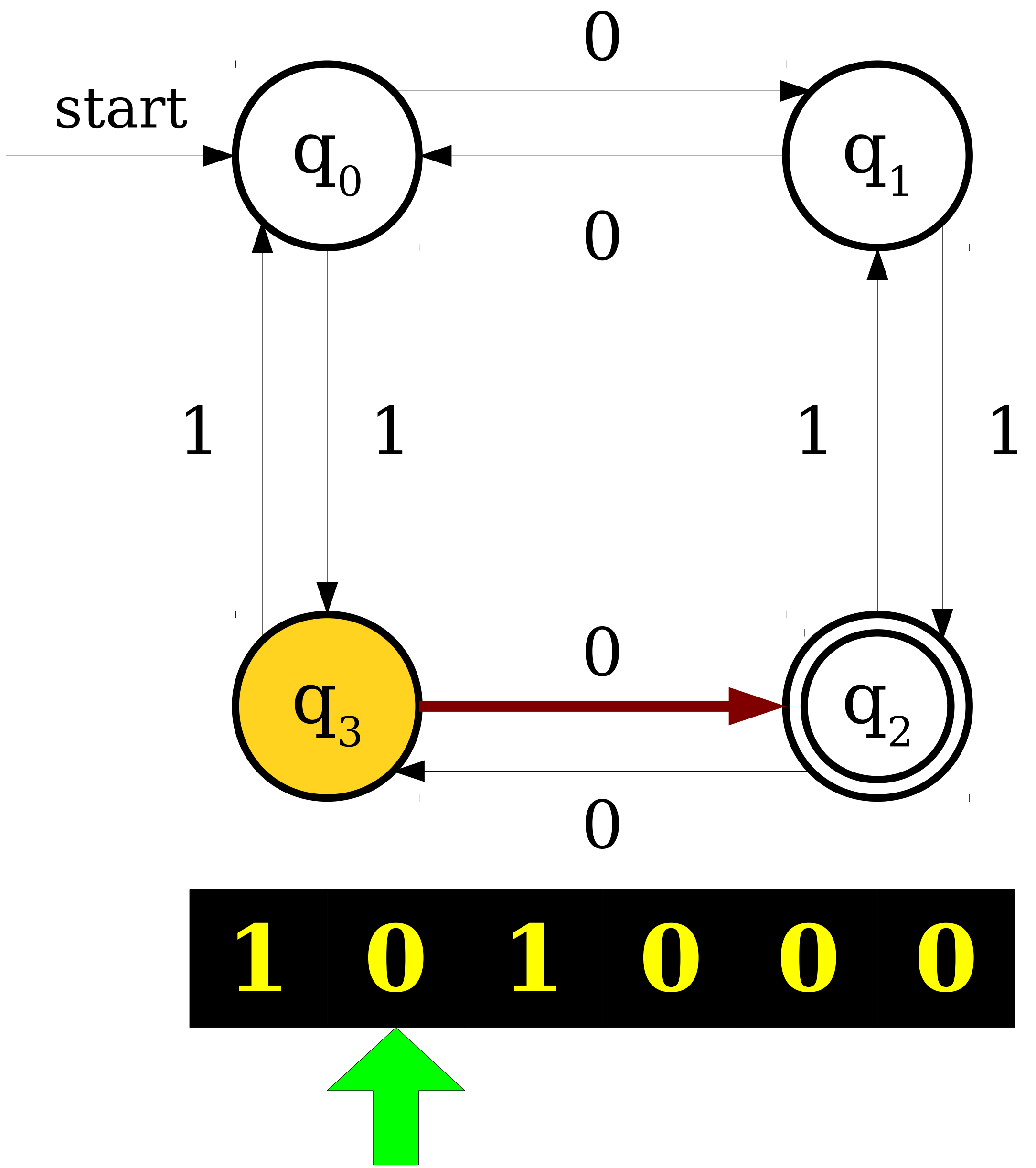
A Simple Finite Automaton



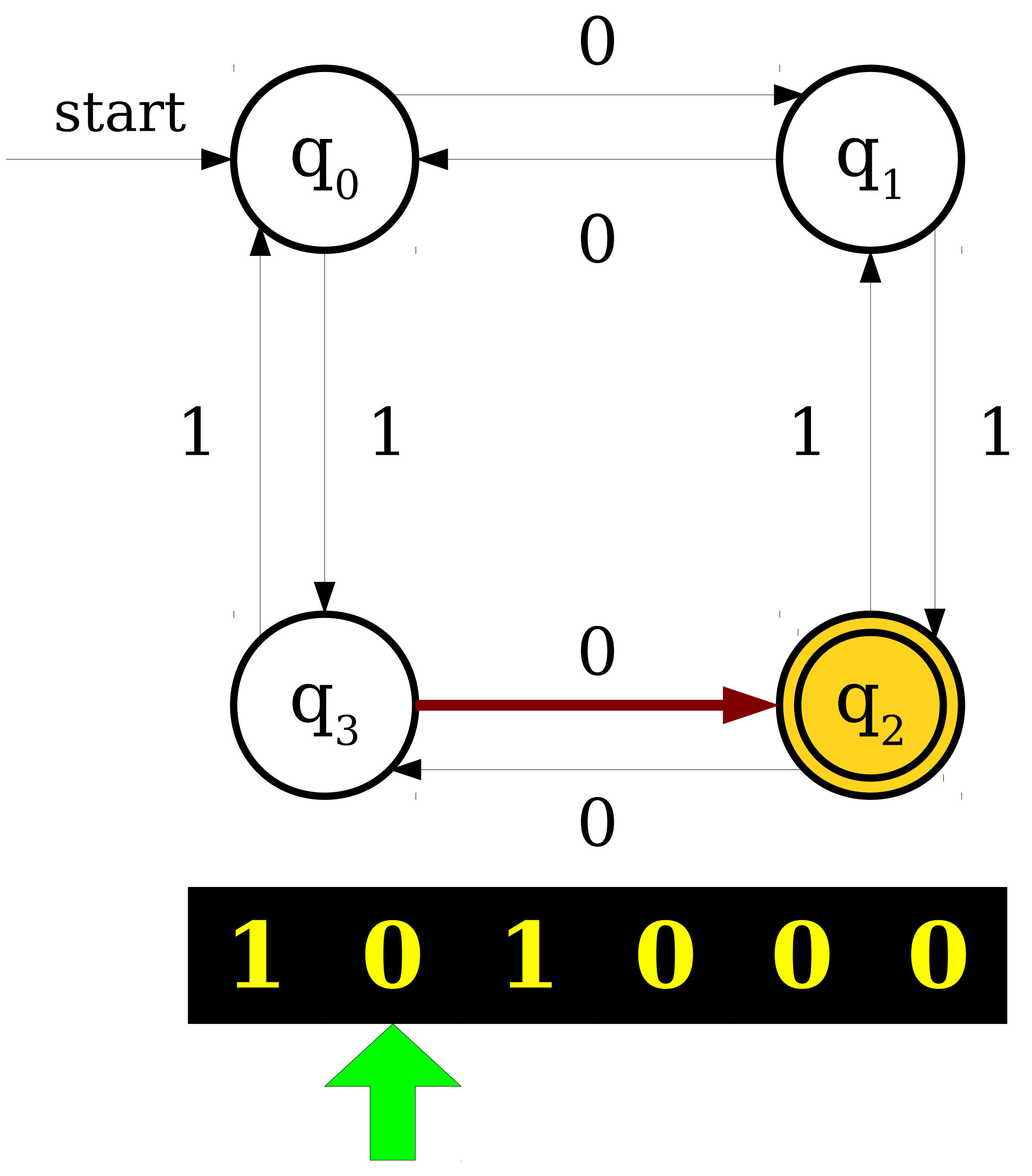
A Simple Finite Automaton



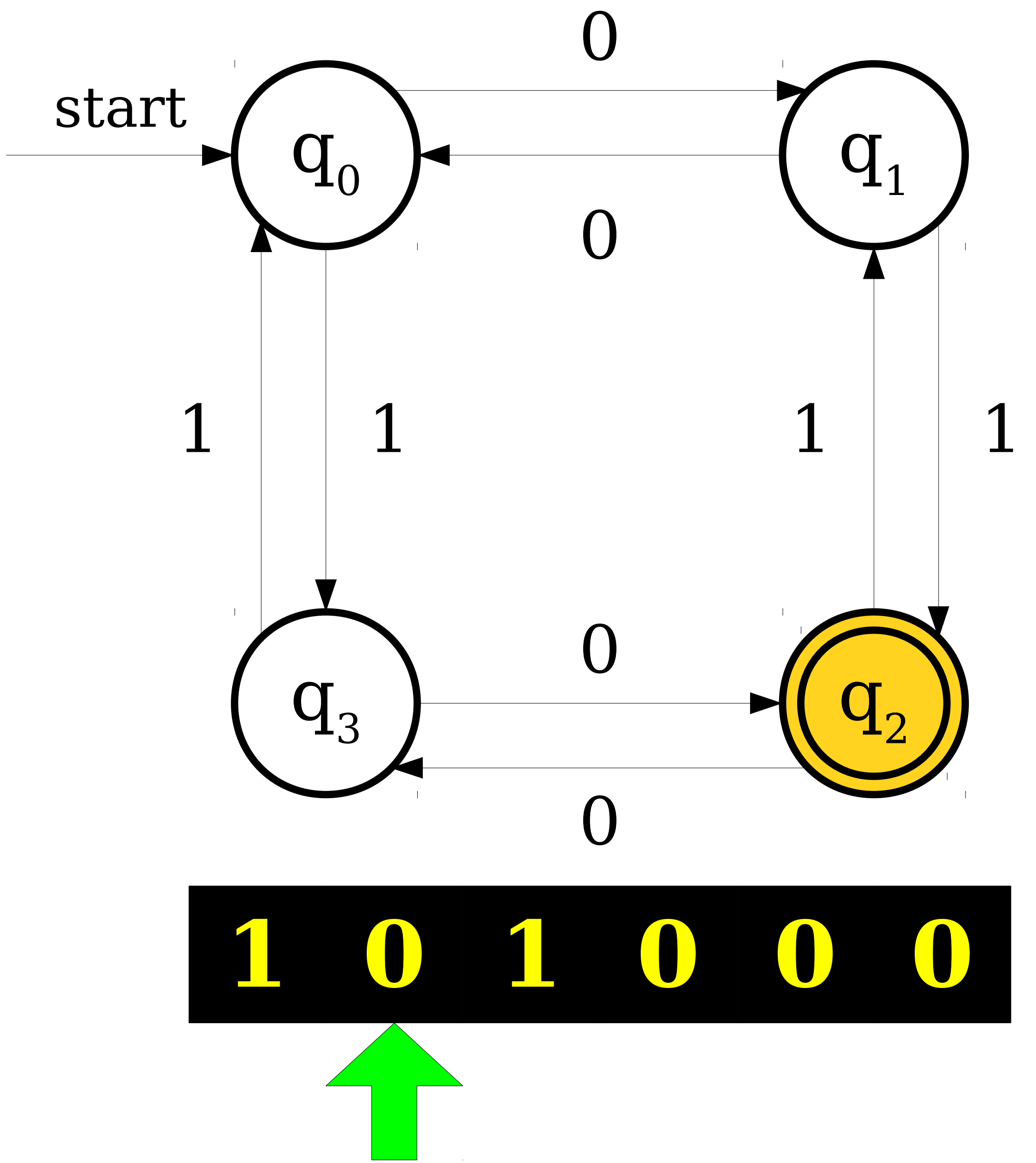
A Simple Finite Automaton



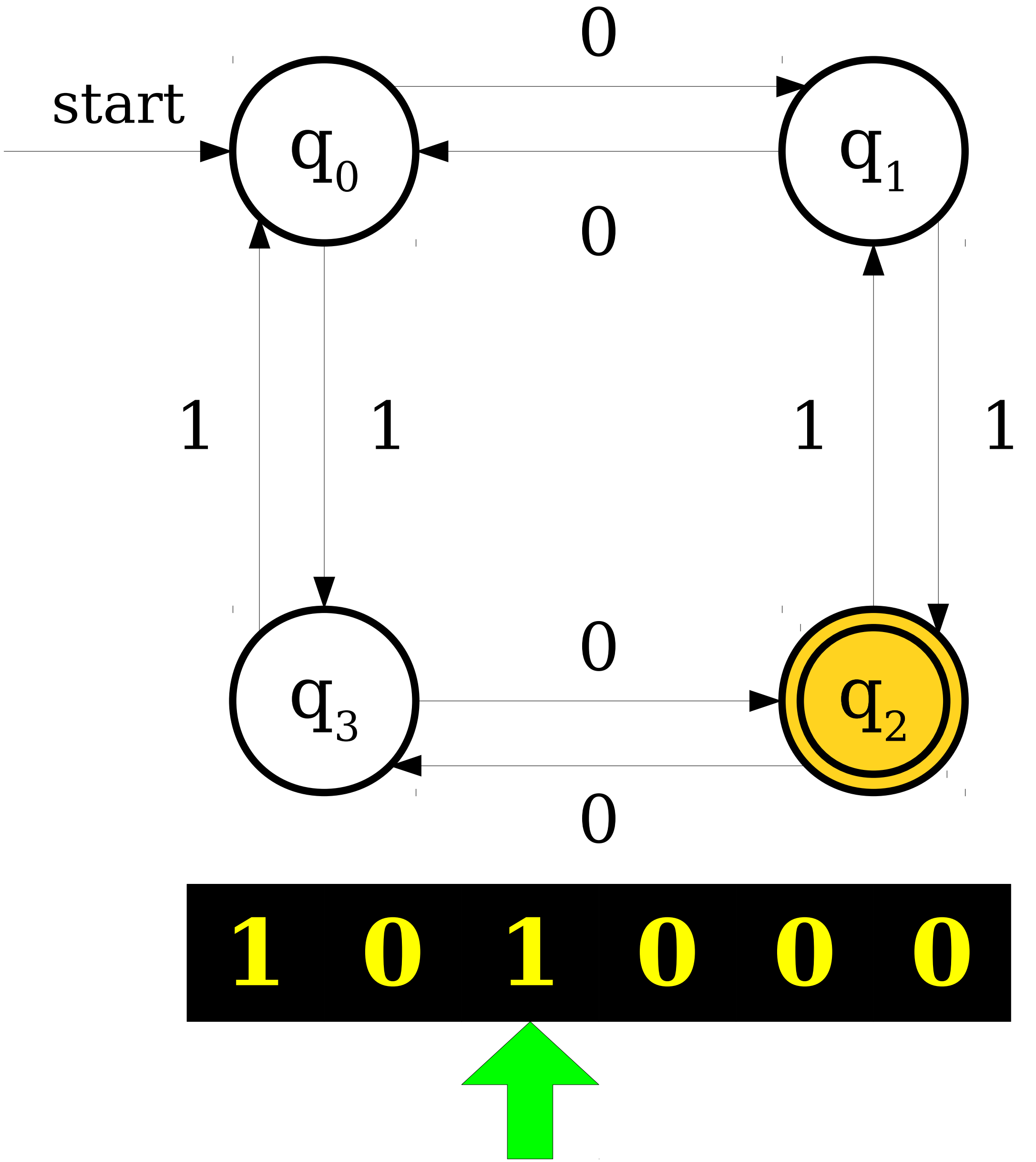
A Simple Finite Automaton



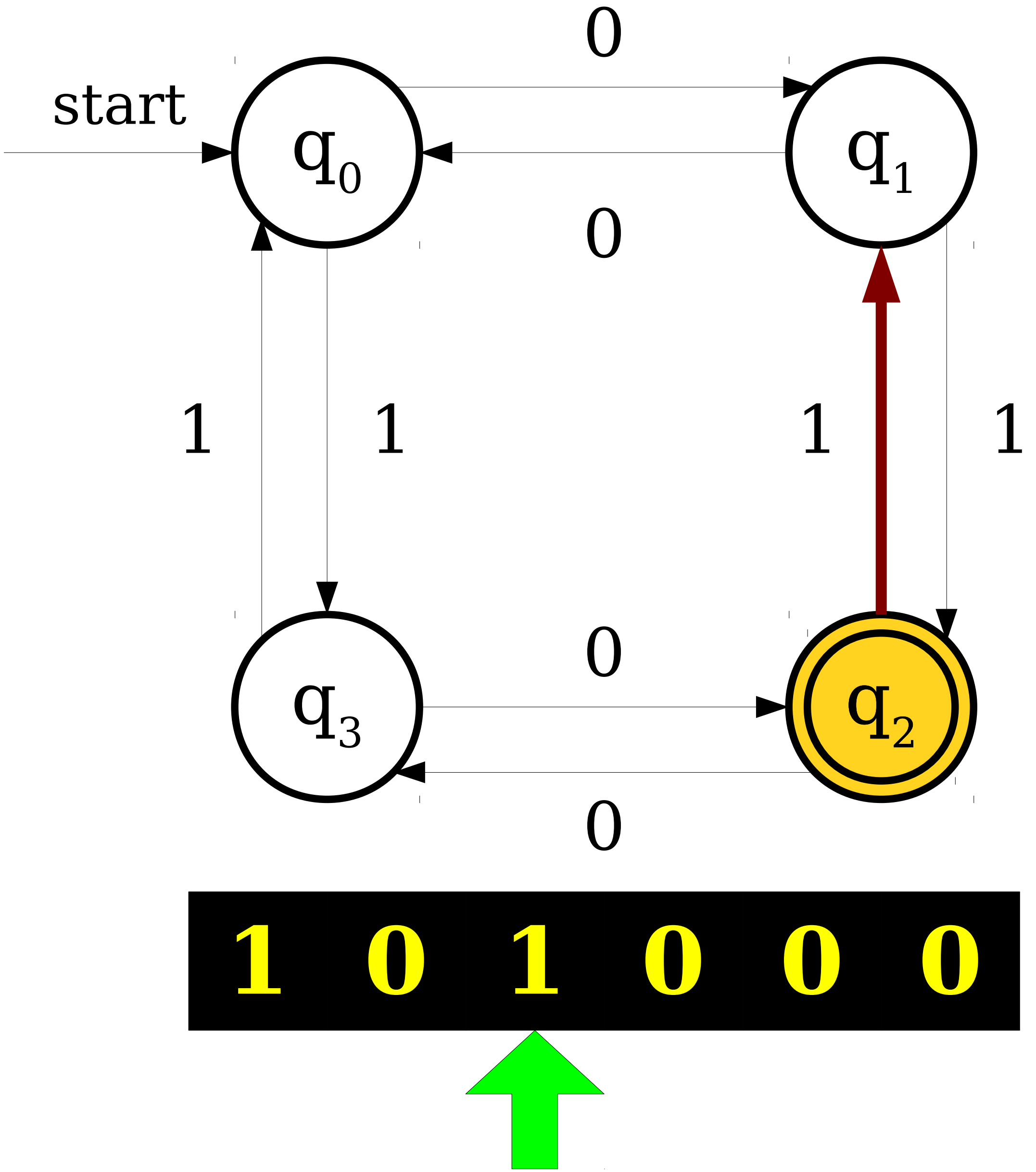
A Simple Finite Automaton



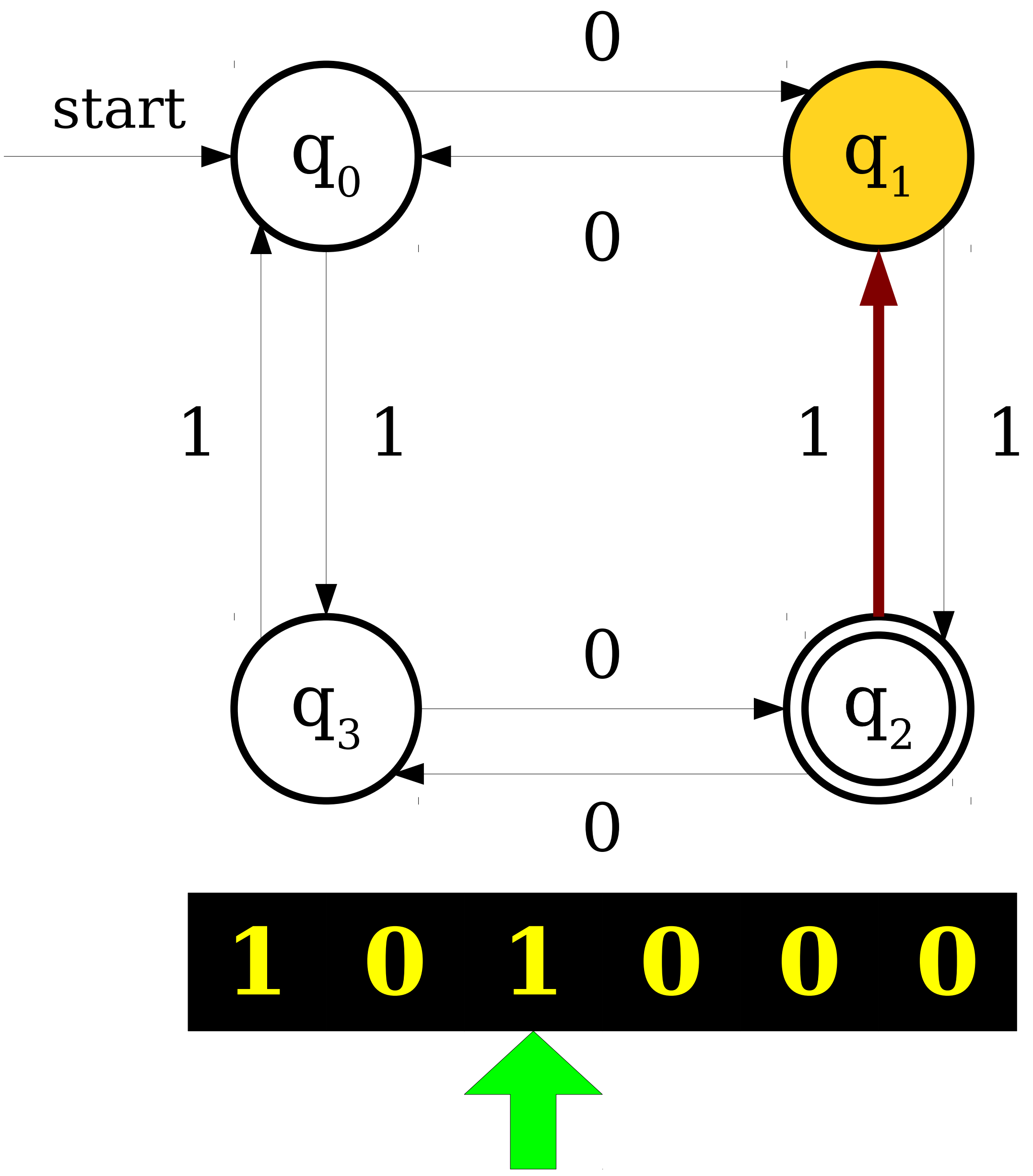
A Simple Finite Automaton



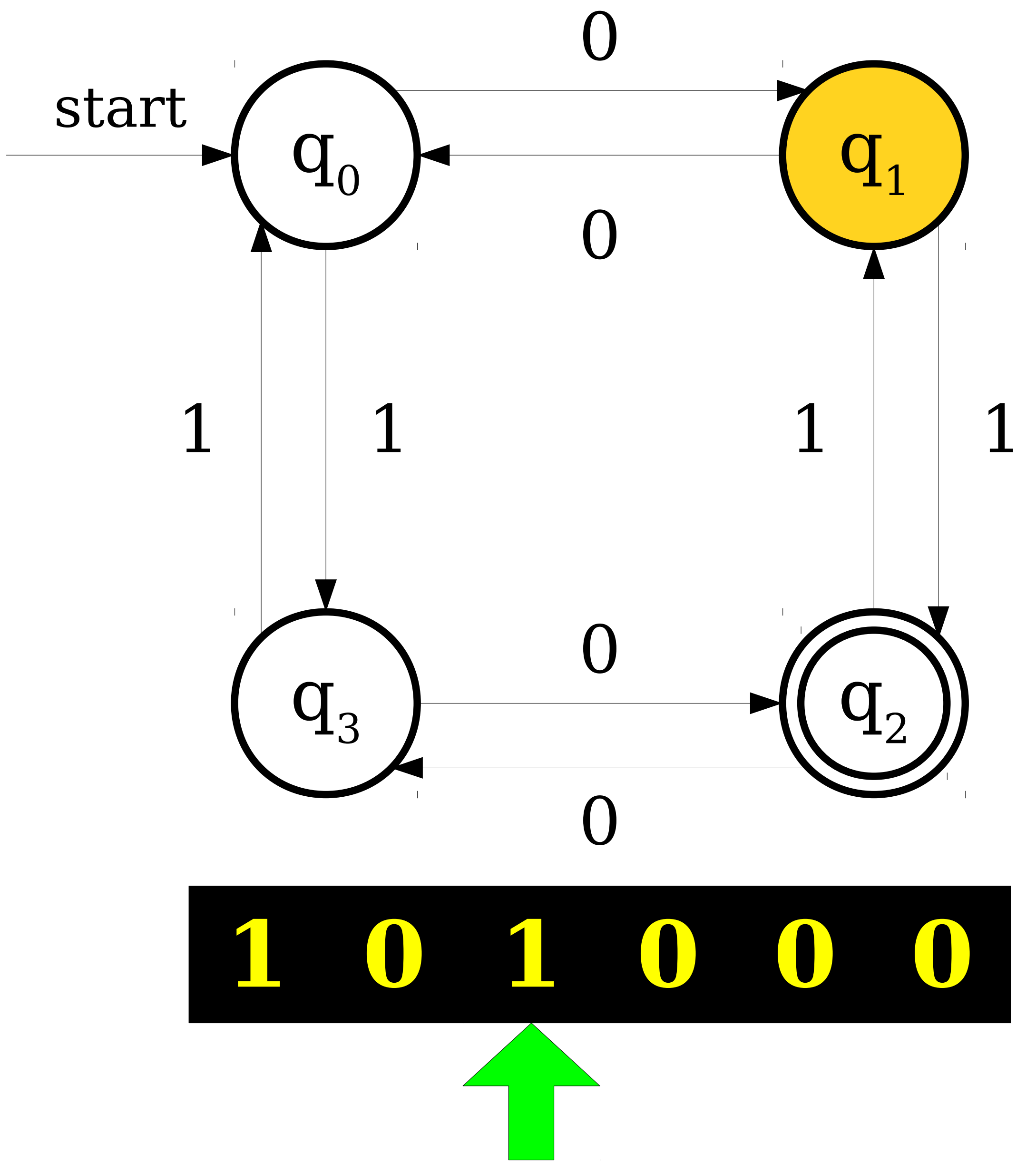
A Simple Finite Automaton



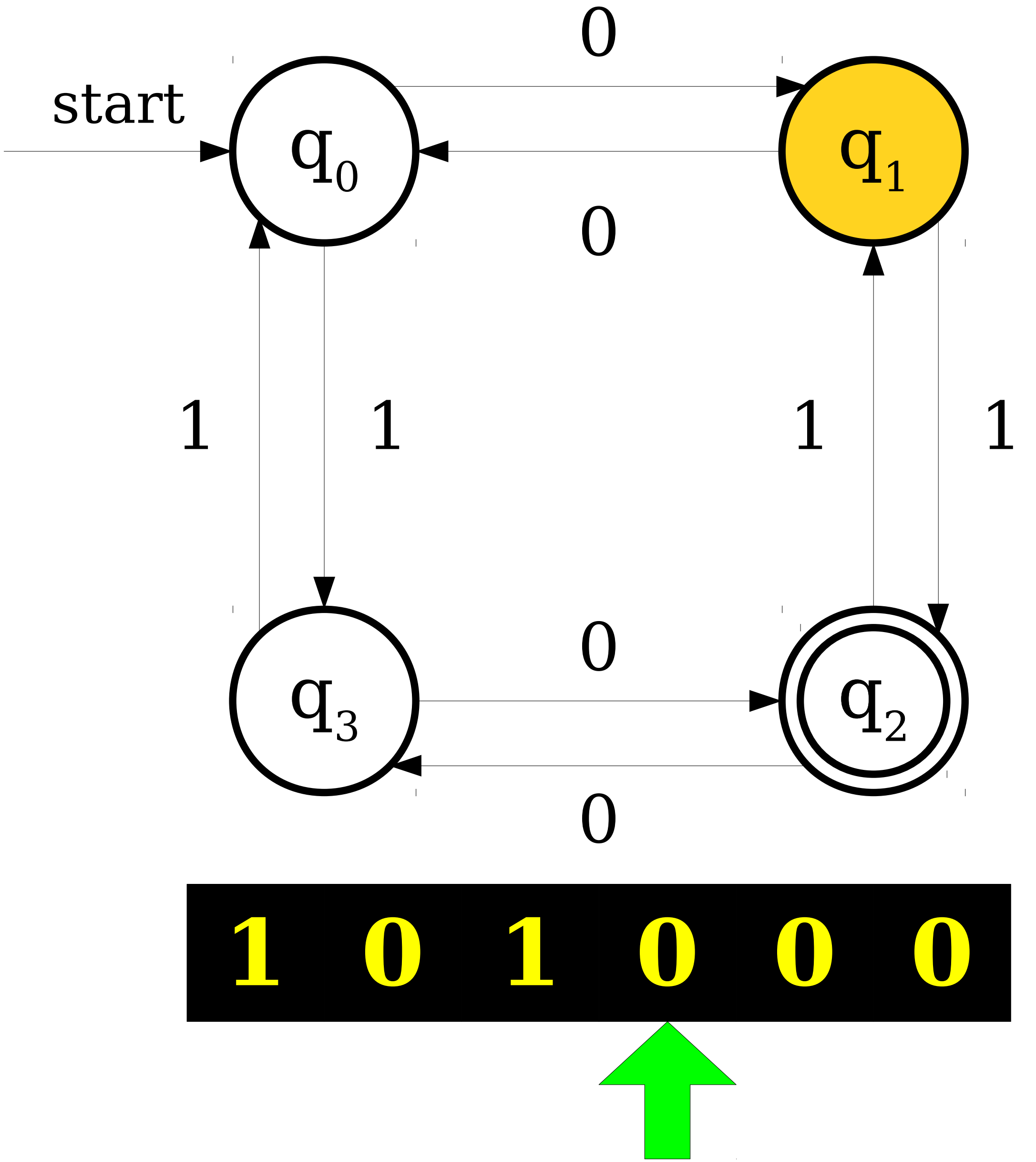
A Simple Finite Automaton



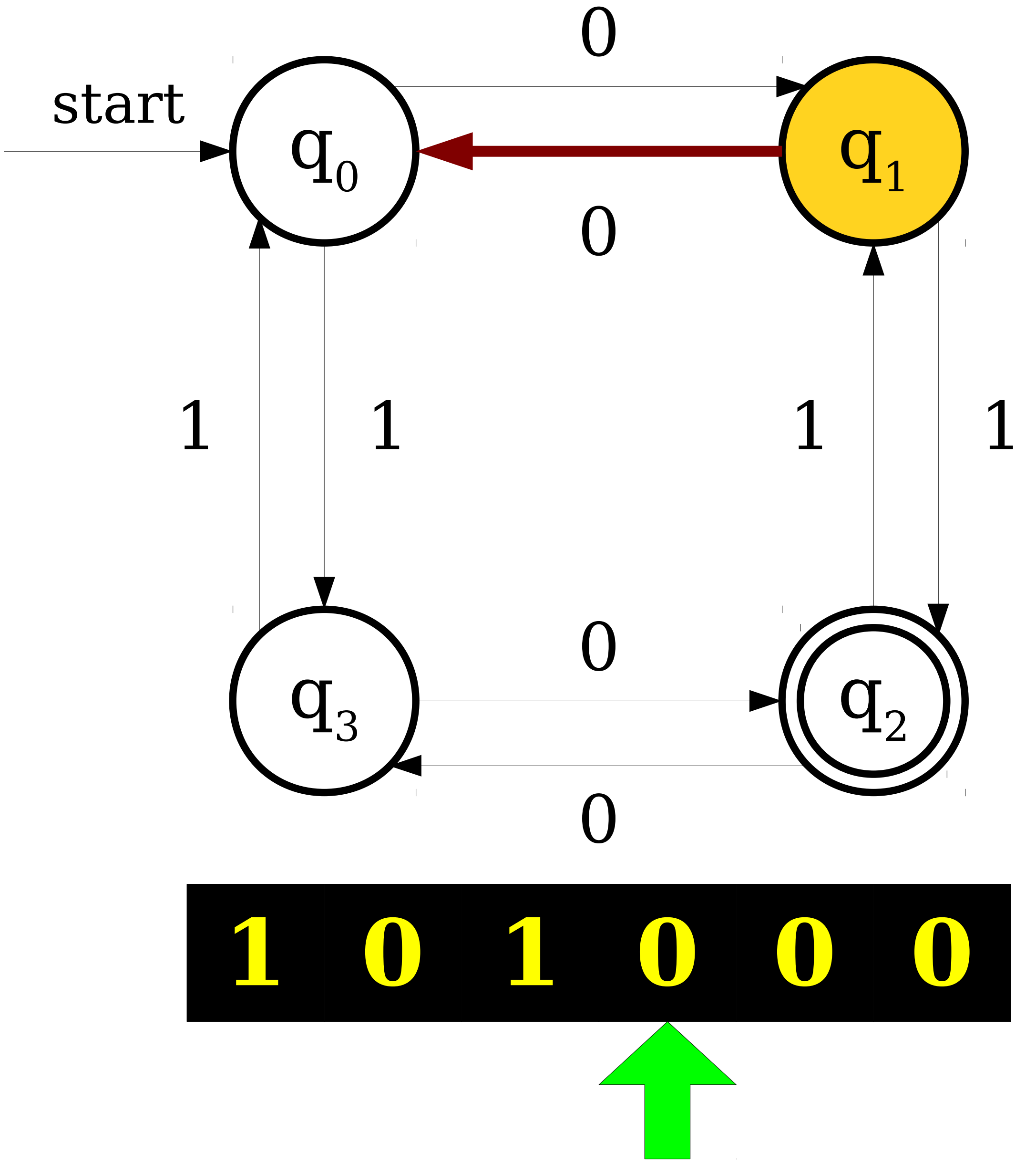
A Simple Finite Automaton



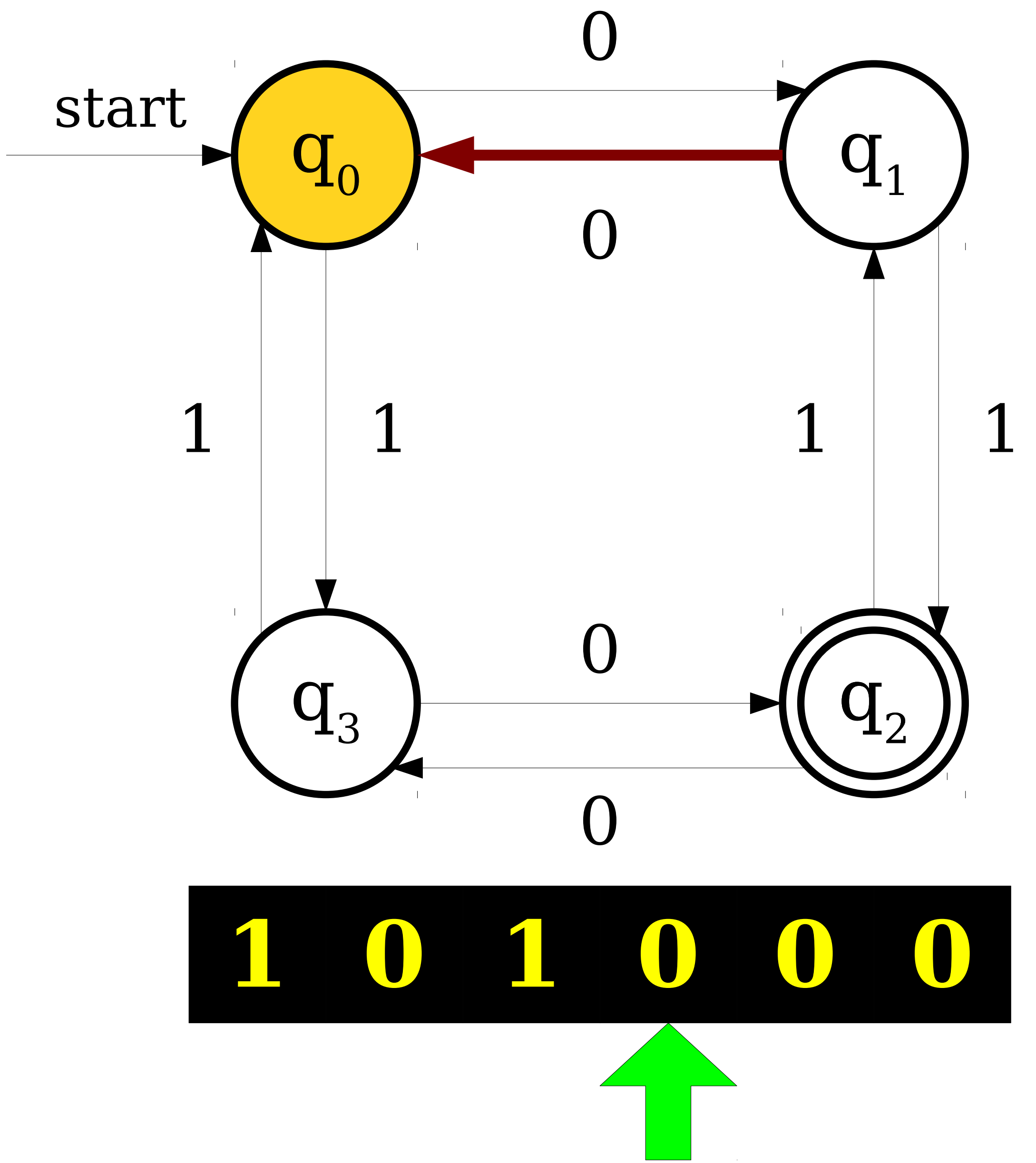
A Simple Finite Automaton



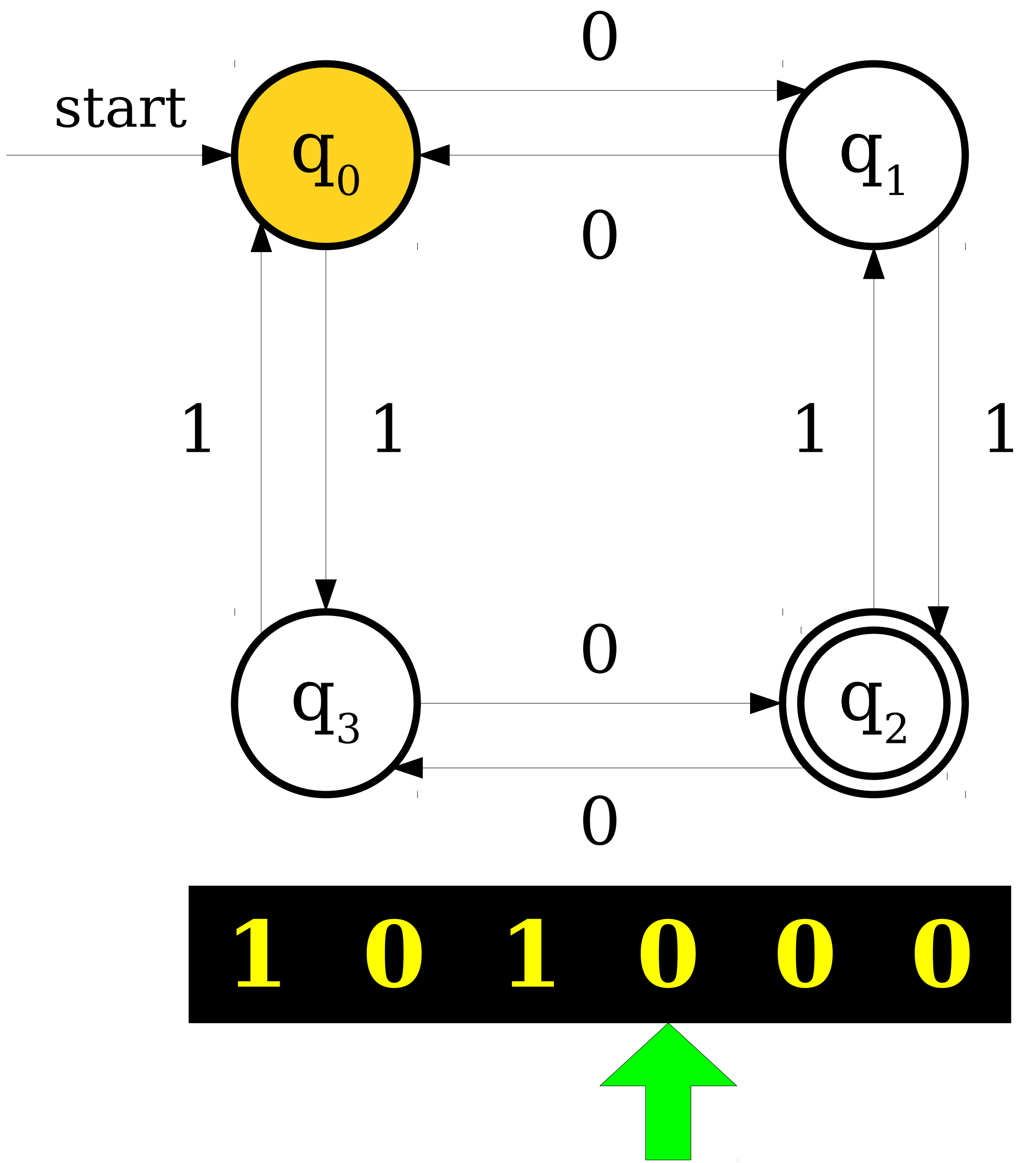
A Simple Finite Automaton



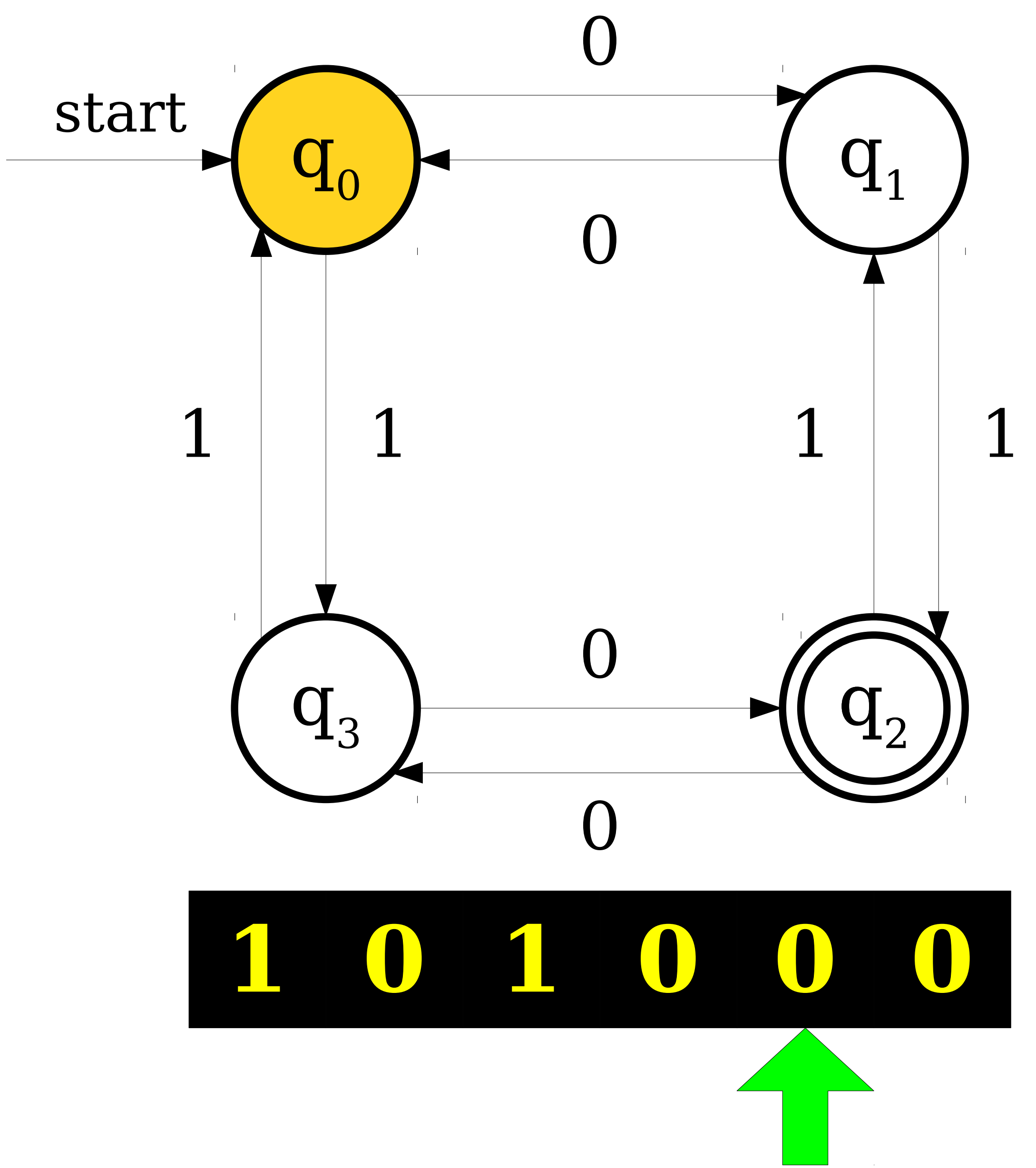
A Simple Finite Automaton



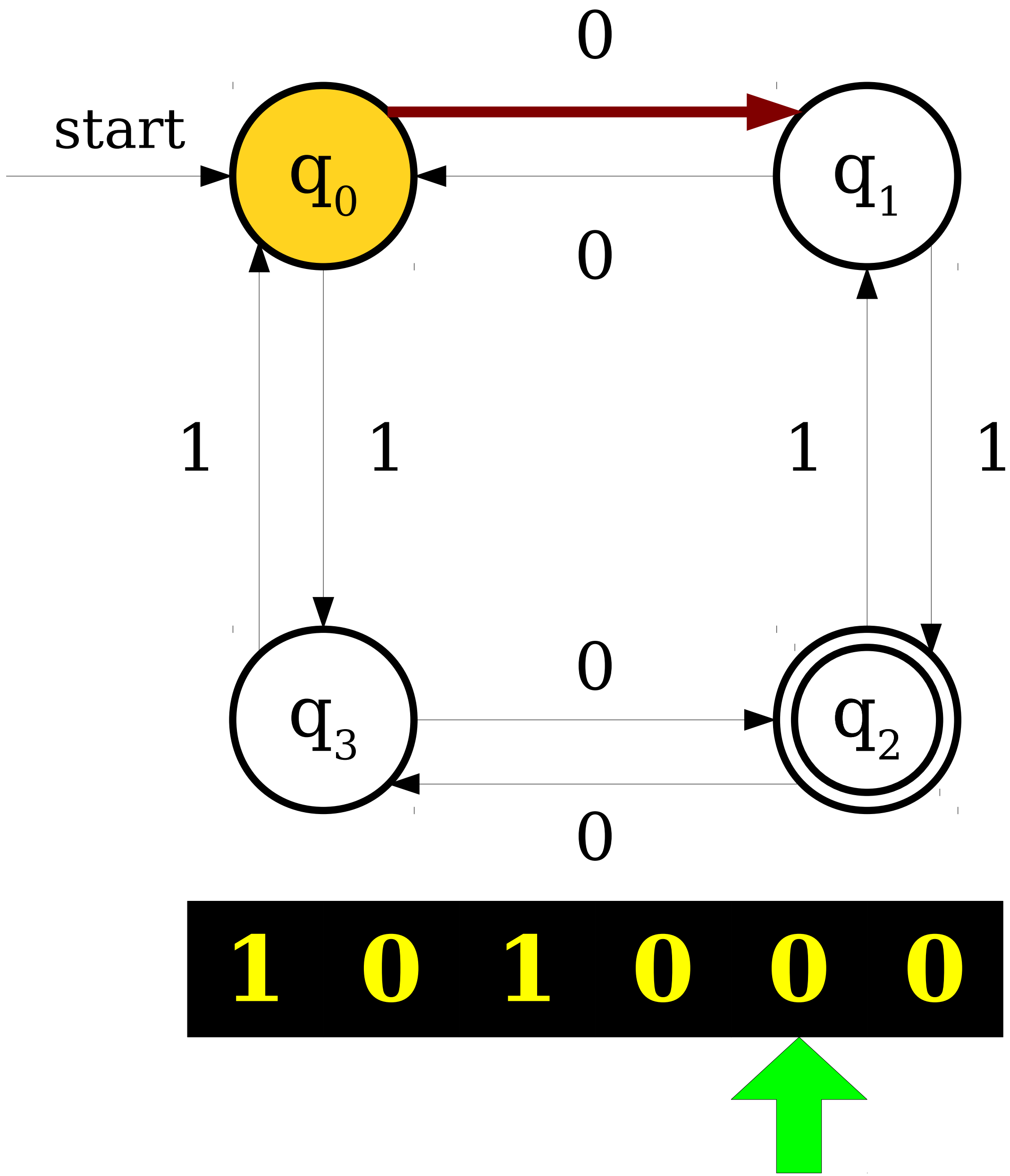
A Simple Finite Automaton



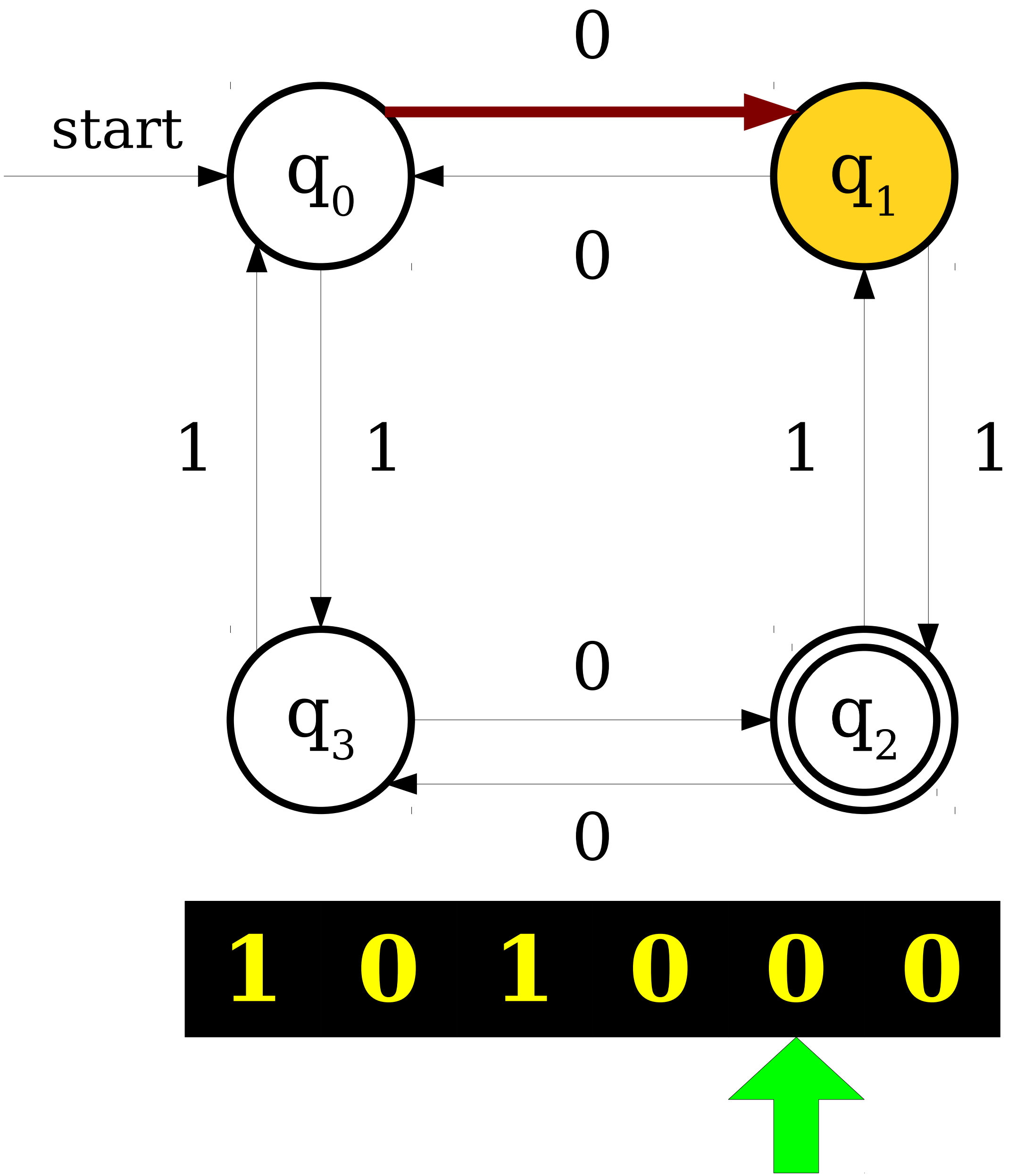
A Simple Finite Automaton



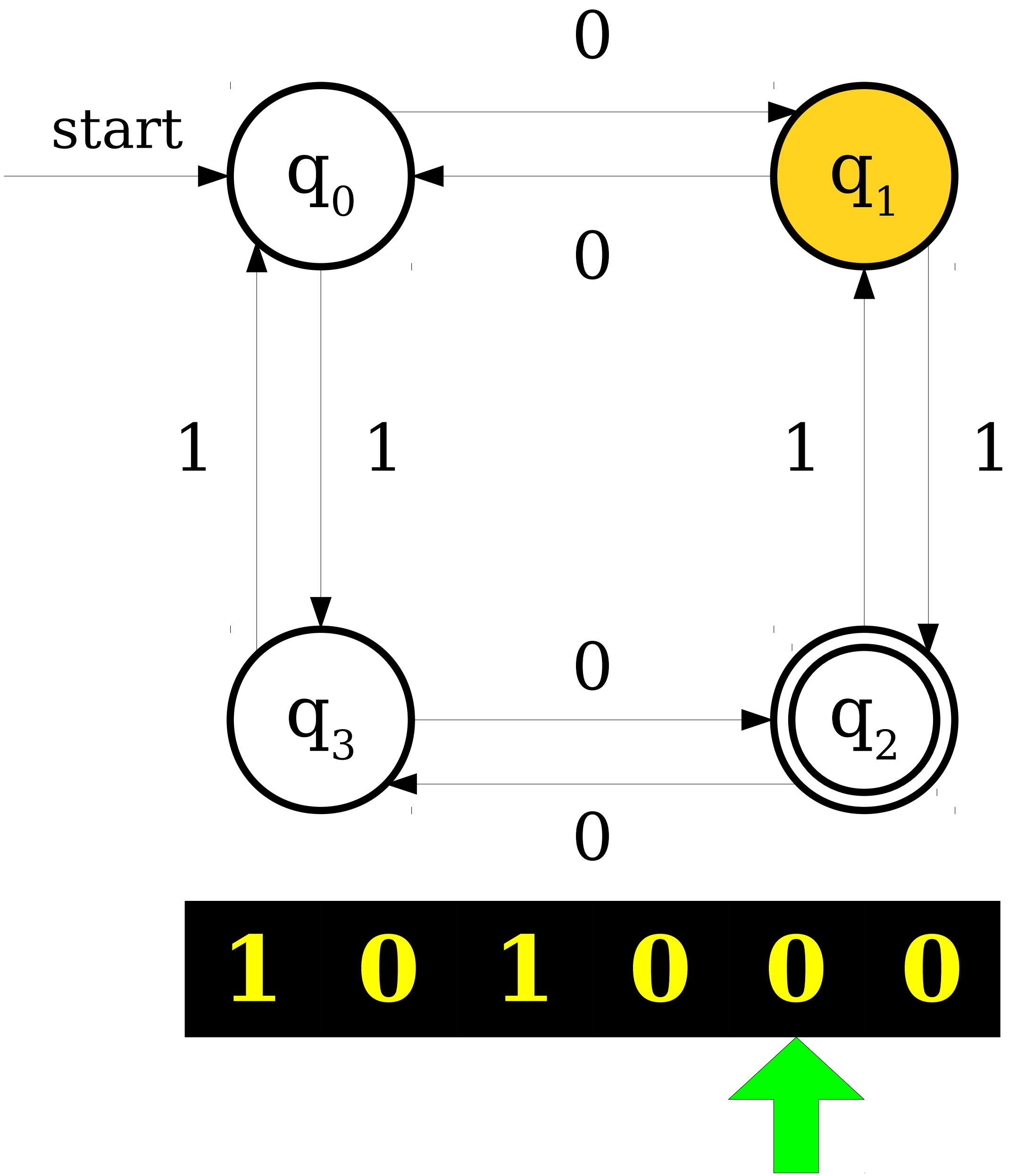
A Simple Finite Automaton



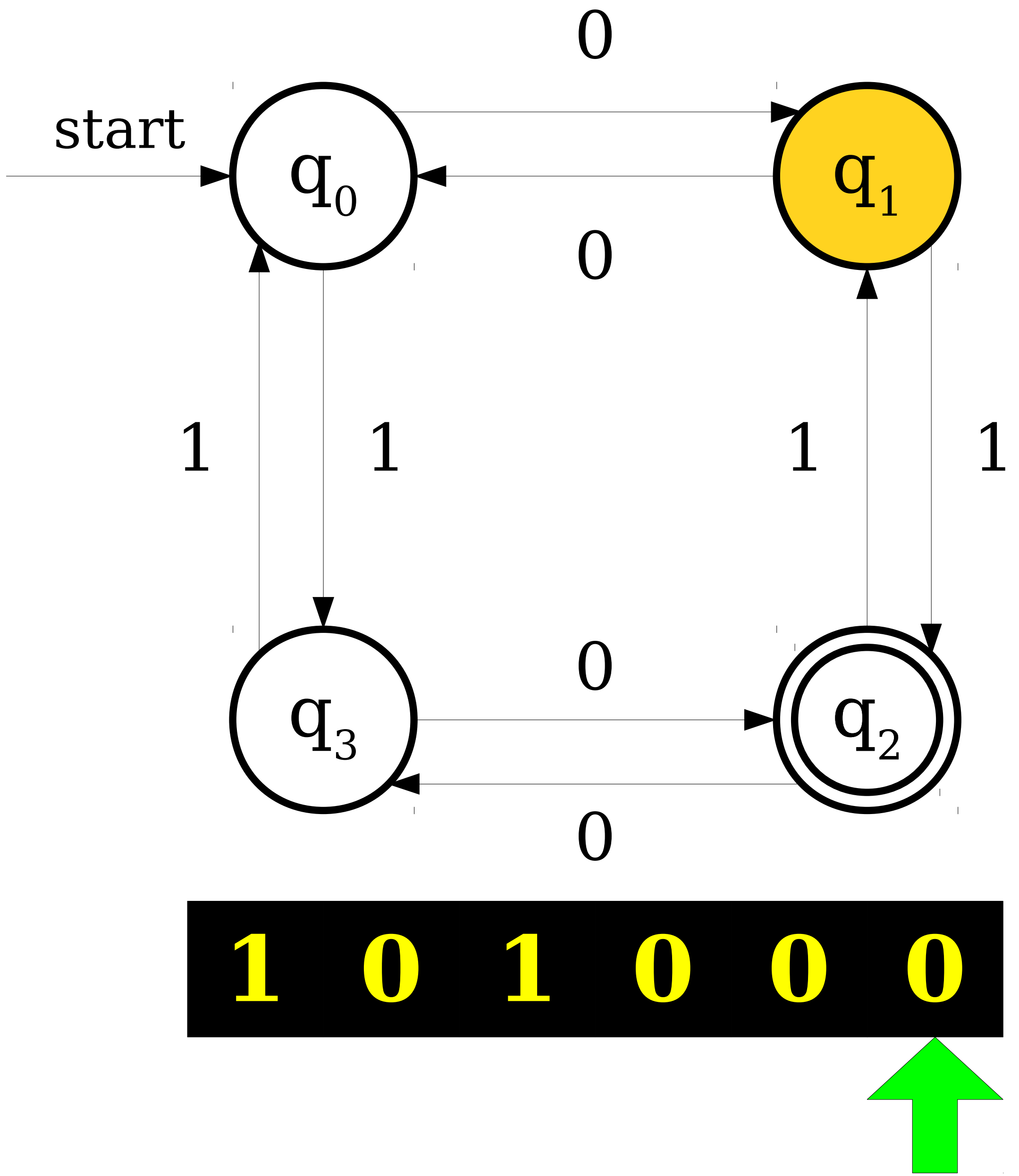
A Simple Finite Automaton



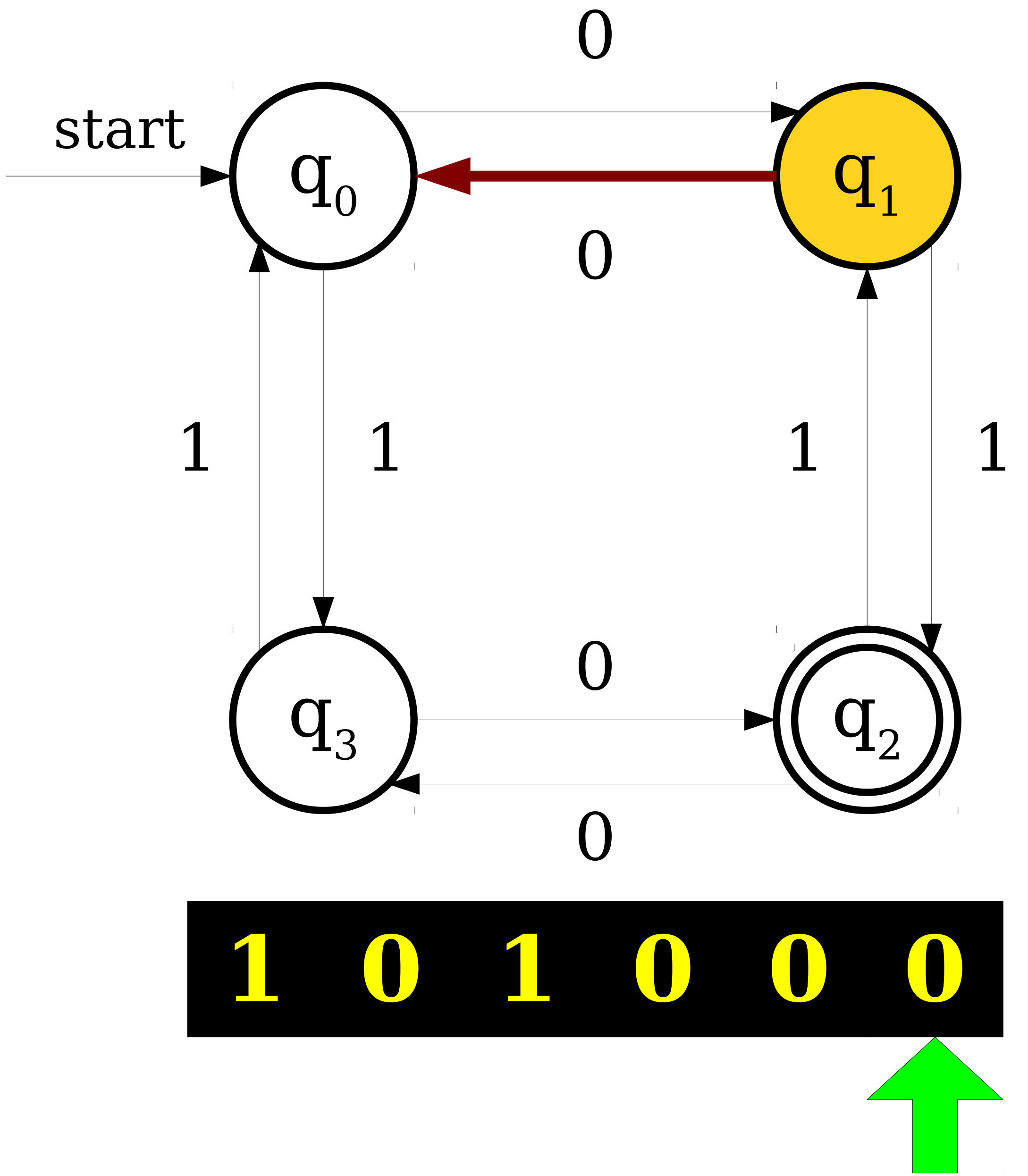
A Simple Finite Automaton



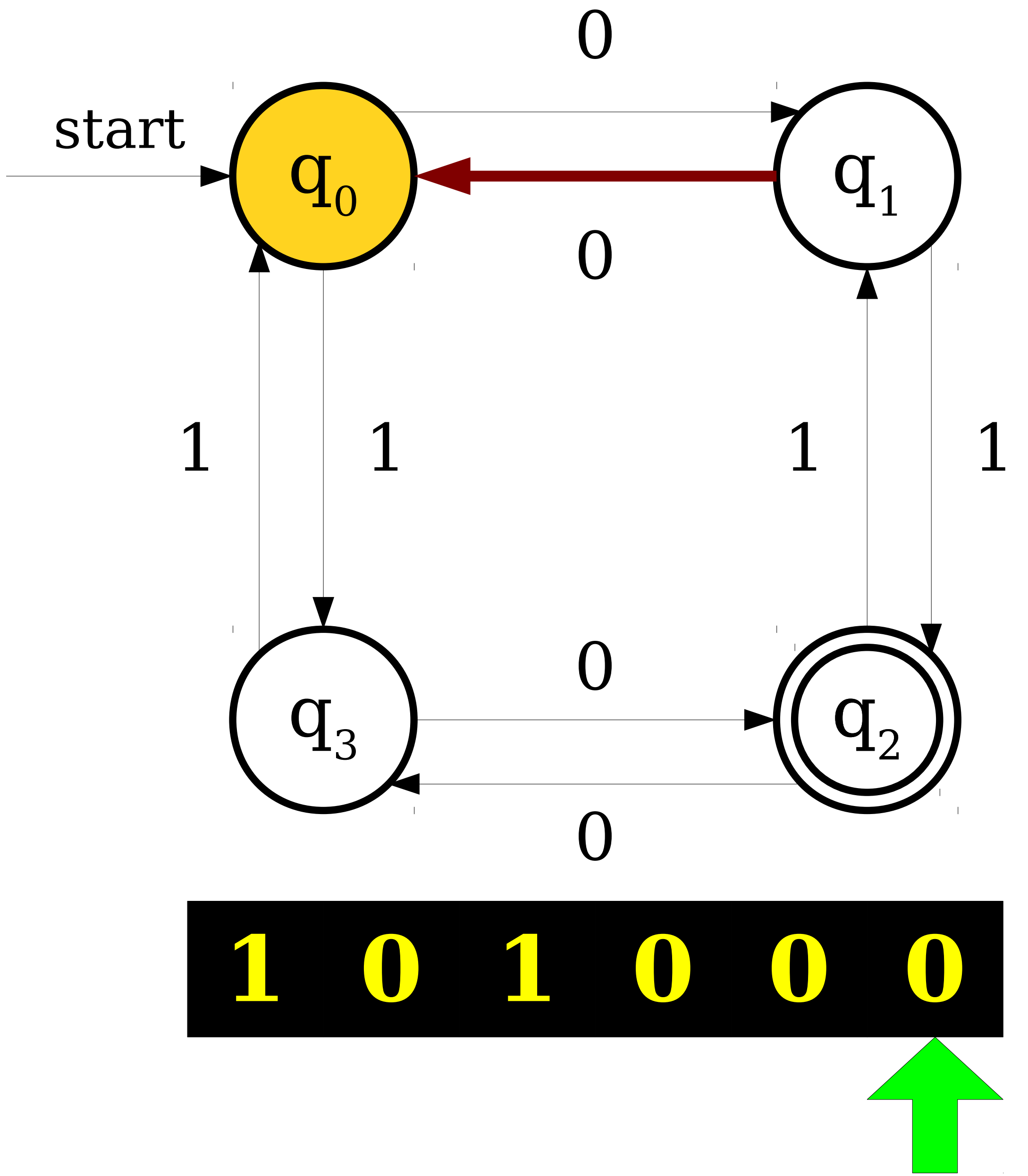
A Simple Finite Automaton



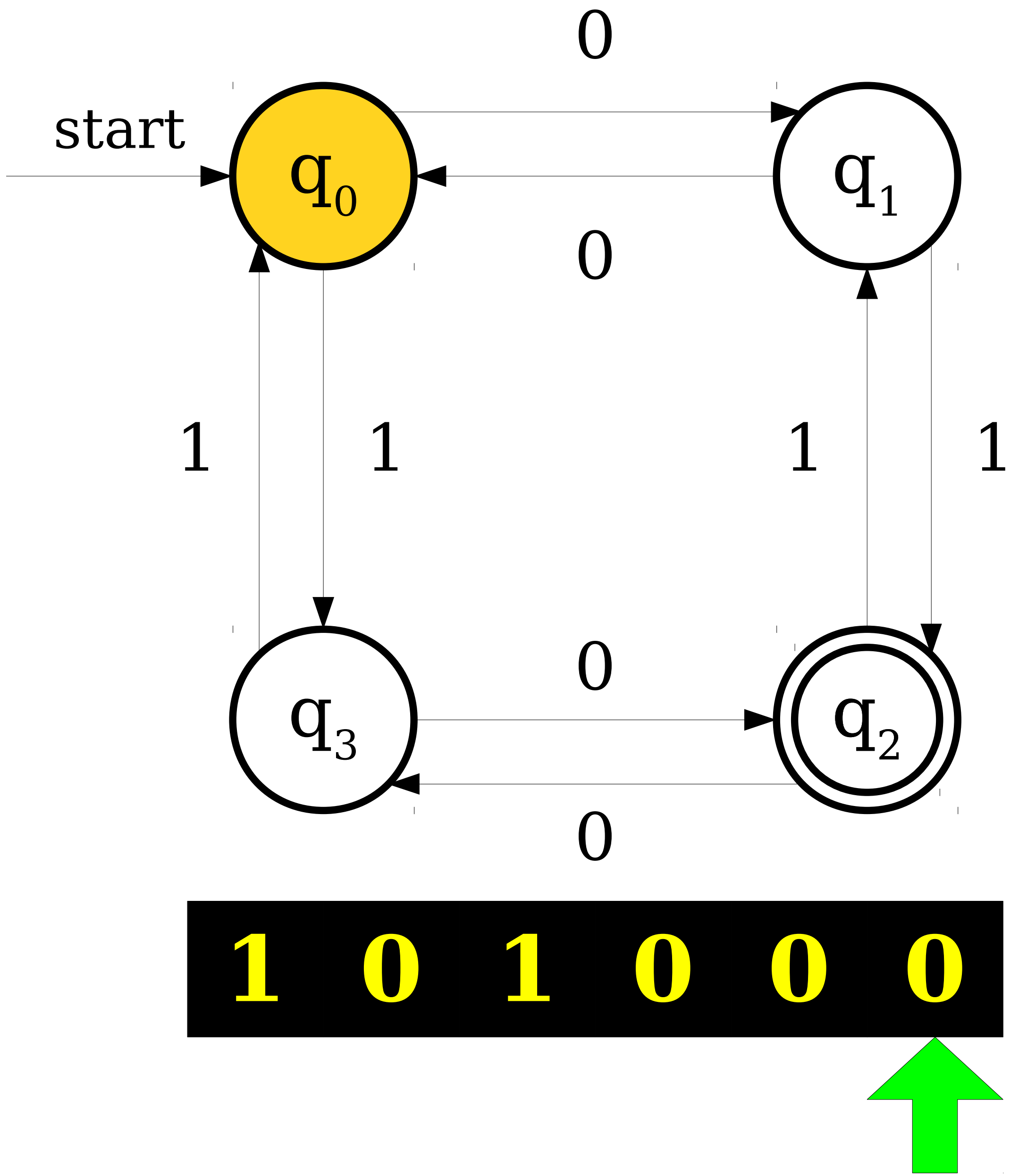
A Simple Finite Automaton



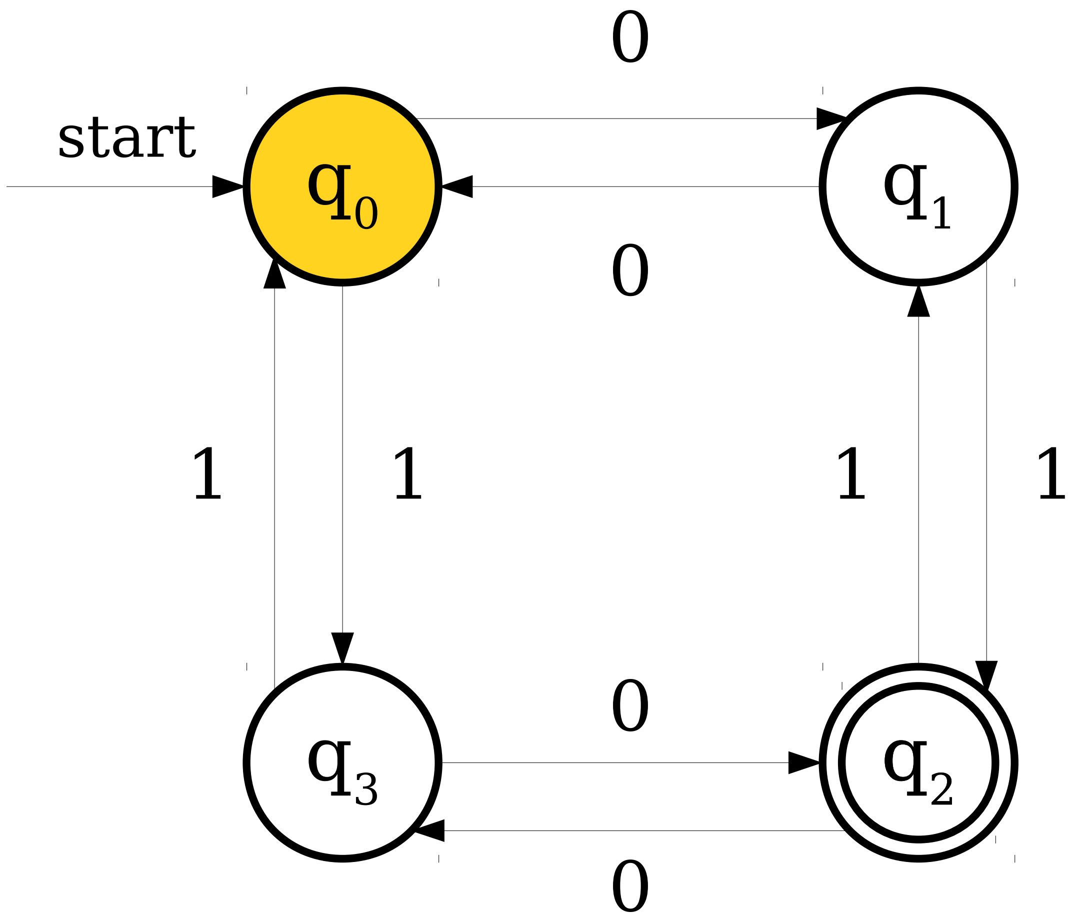
A Simple Finite Automaton



A Simple Finite Automaton

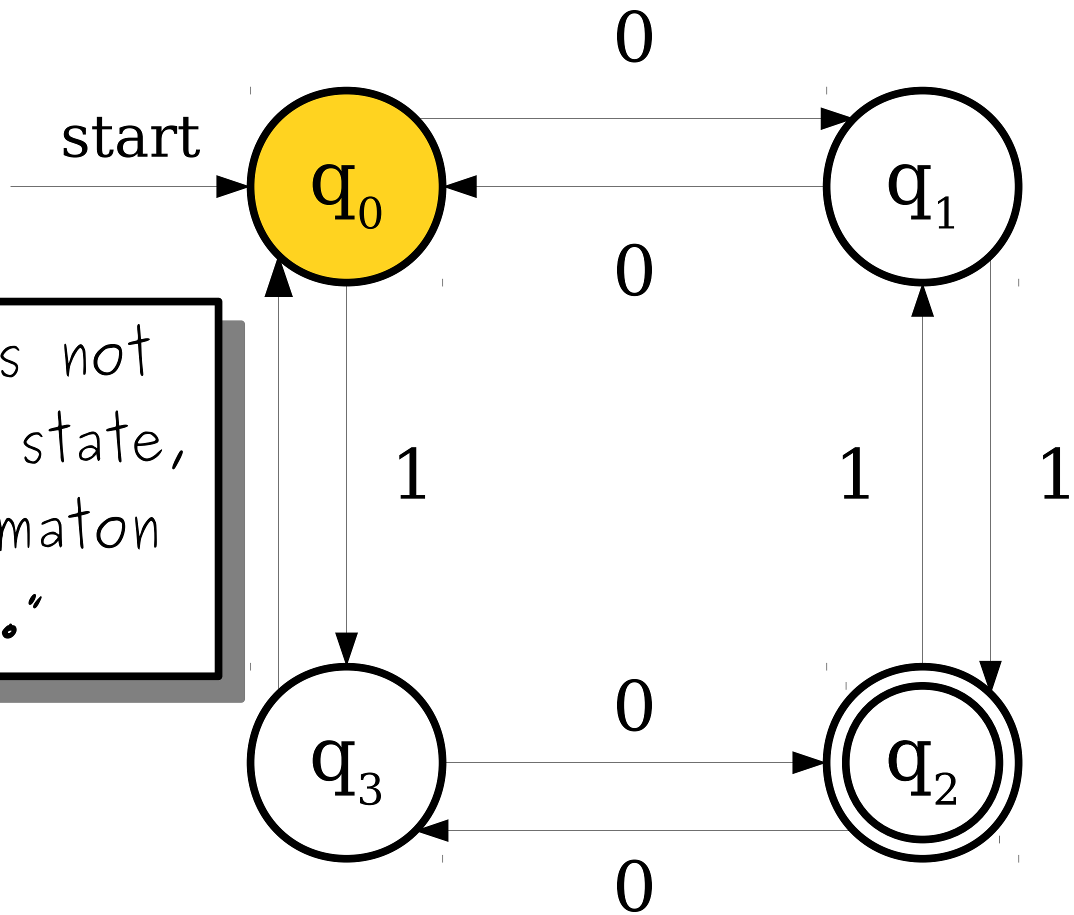


A Simple Finite Automaton



1 0 1 0 0 0

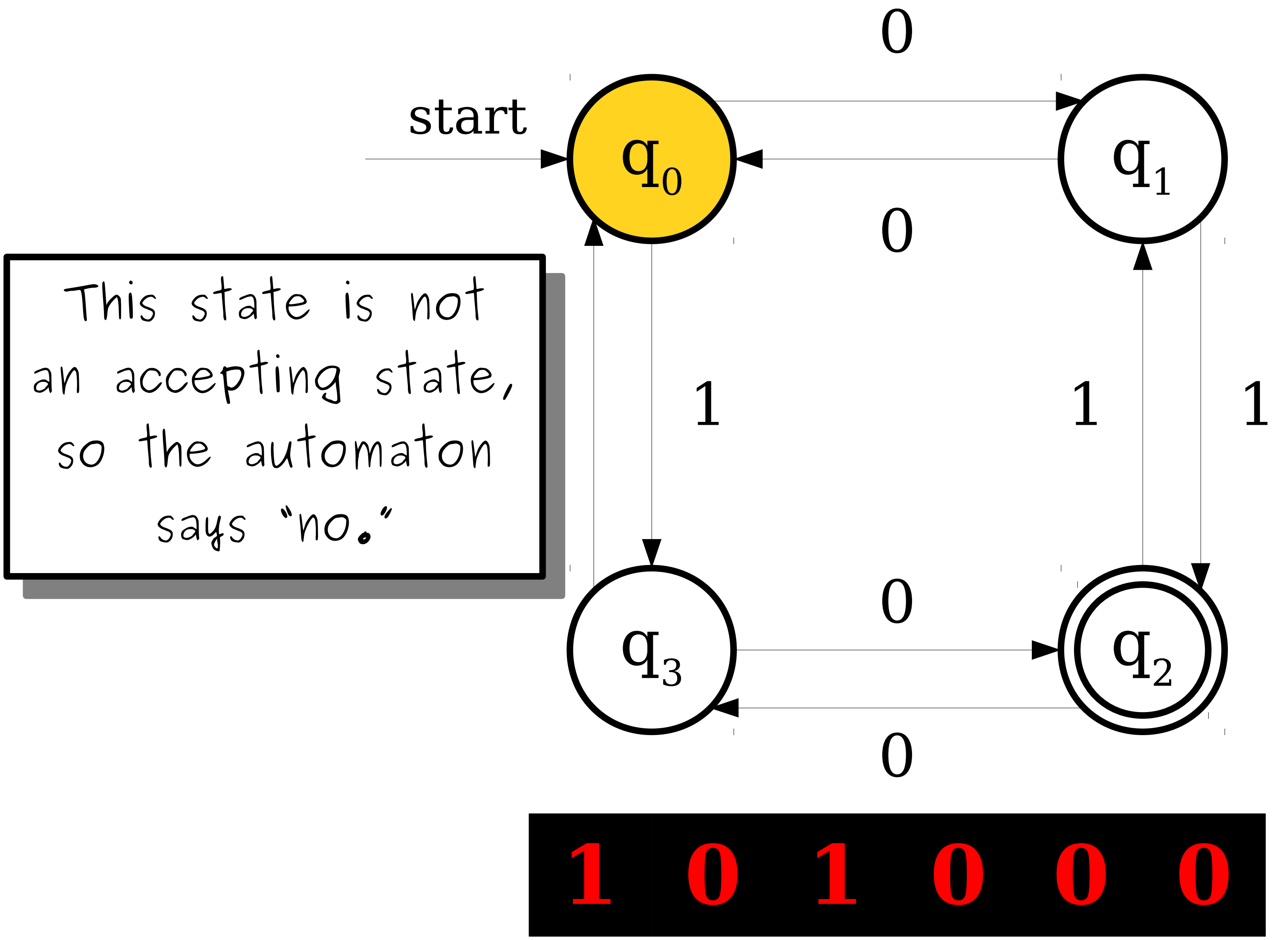
A Simple Finite Automaton



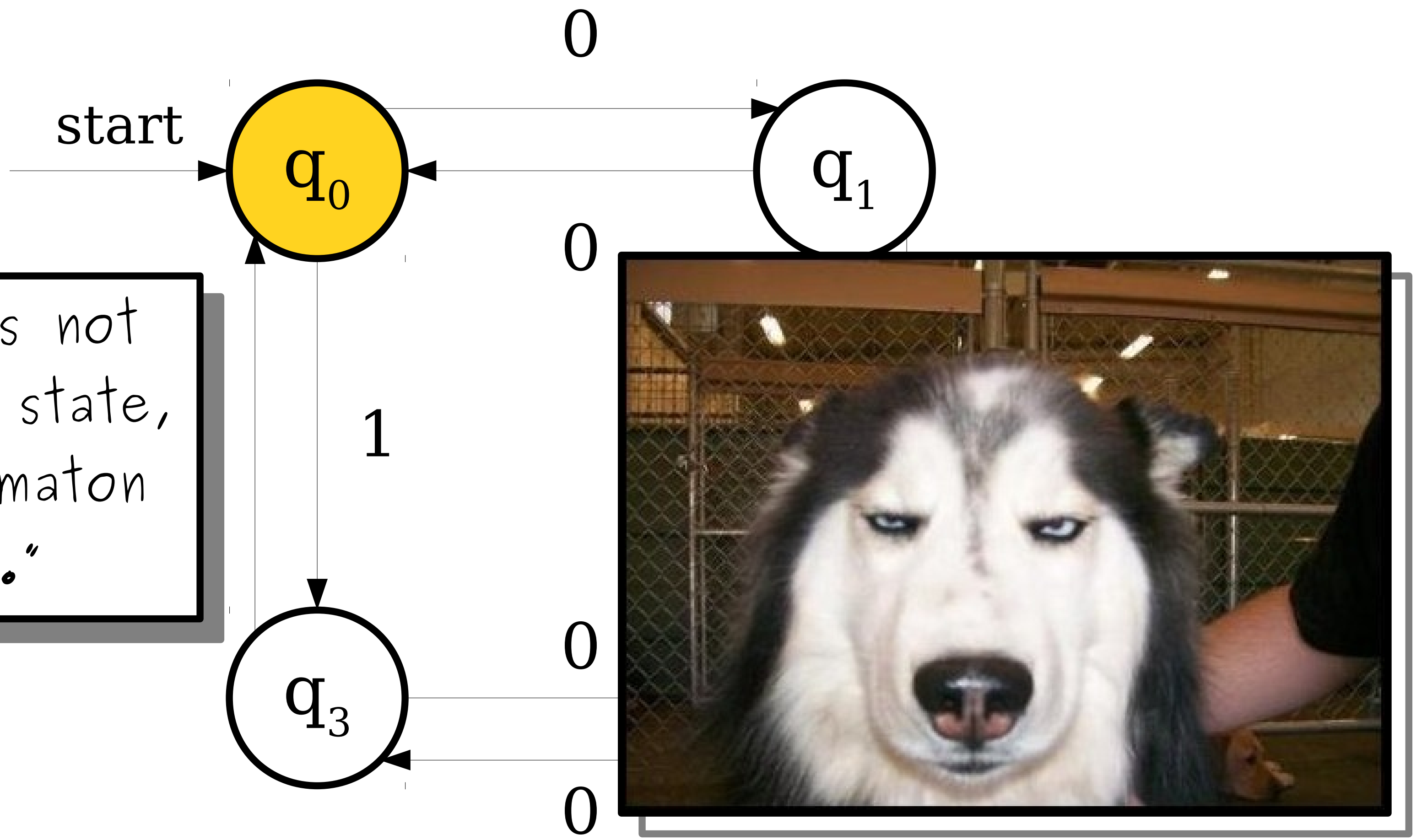
This state is not an accepting state, so the automaton says "no."

1 0 1 0 0 0

A Simple Finite Automaton



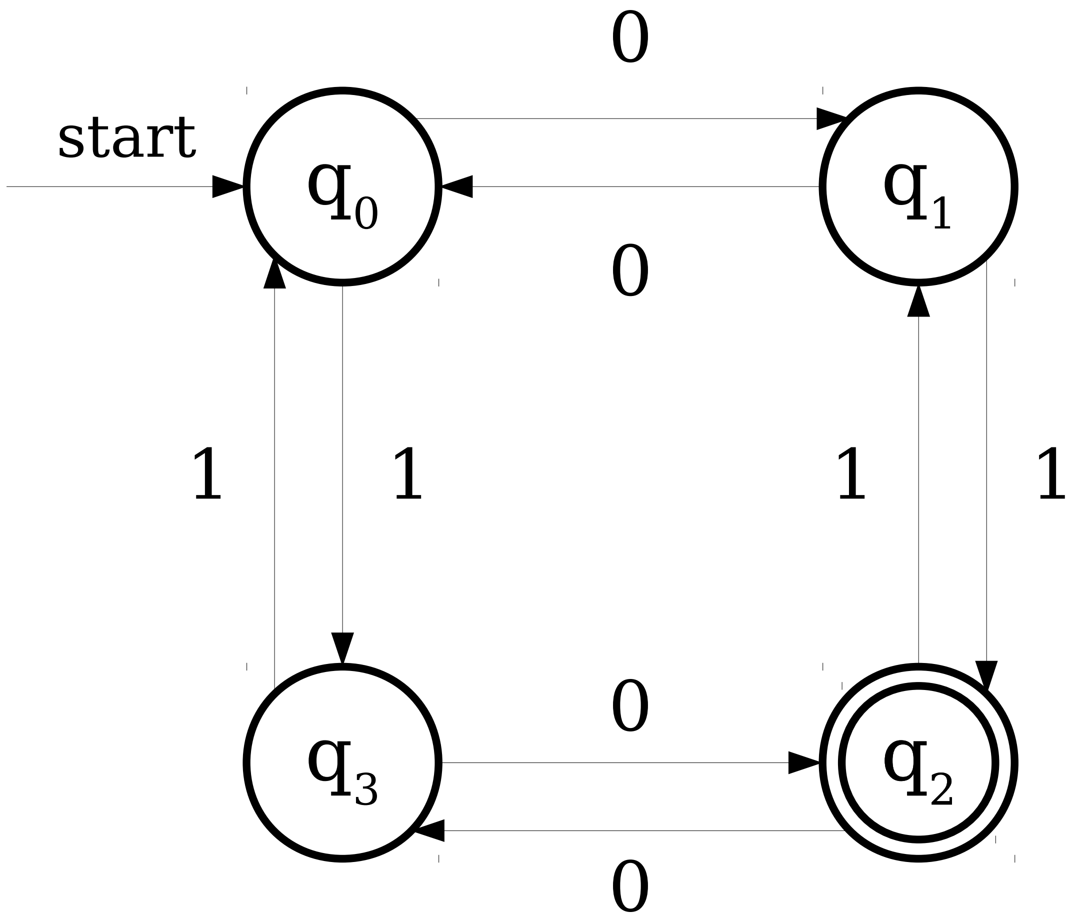
A Simple Finite Automaton



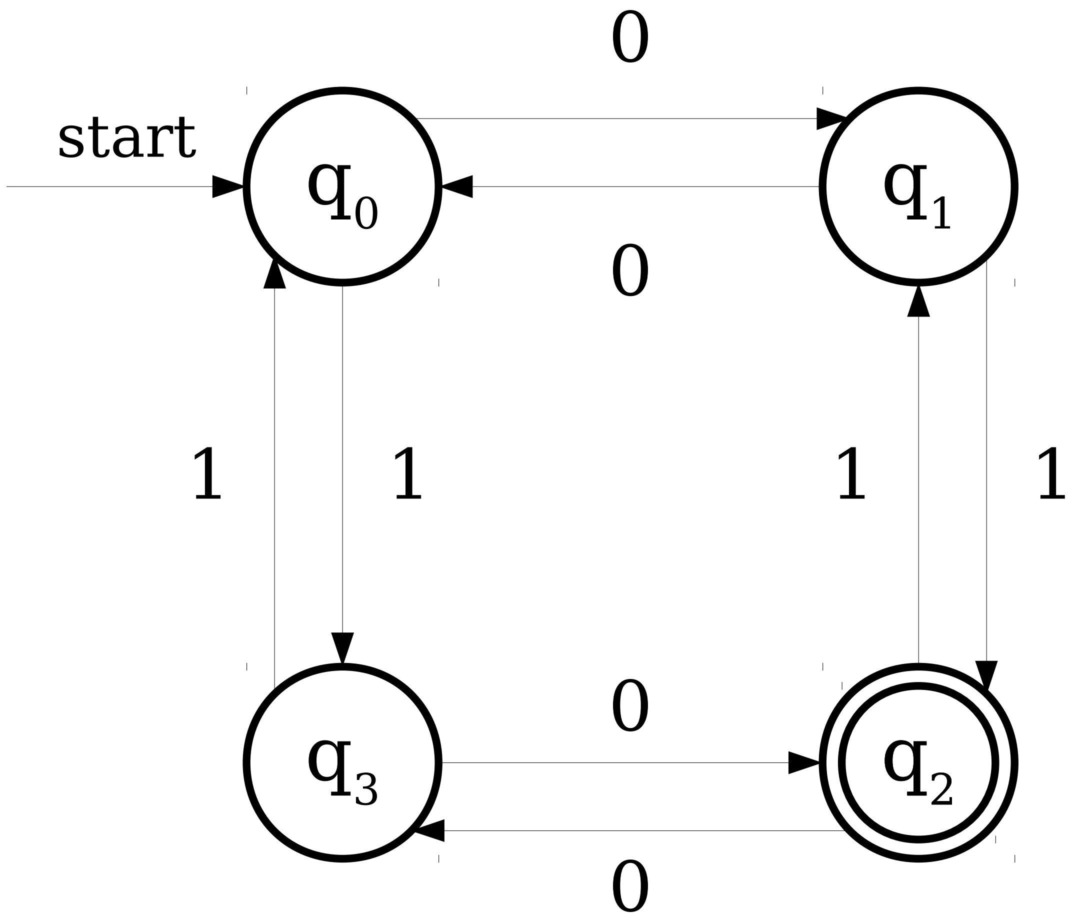
This state is not an accepting state, so the automaton says "no."

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

The midterm is tomorrow – good luck!

The Binary Relation Editor

Solution Sets

- All solution sets are now available in the filing cabinet.
- There's a typo in the PS4 checkpoint solutions – the checkpoint is worth 2 points, not 25 (sorry about that!)
- We will recycle all unclaimed solution sets for PS1 – PS3 early next week to make room for more solution sets; please stop by and pick them up!

Late Policy

- Up to this point, we've been a bit flexible with our late policy due to issues with Scoryst.
- Going forward, anything submitted past 2:15:00PM on the due date is late and uses up your one late period.
- Anything submitted past 2:15:00PM one class period after the due date will not be accepted.

Your Questions

“Can you please go over 'Translating into Logic' questions (Problem 8 from Pset 3)? Or do you have any tips/guidance on how to approach these types of questions?”

(So... this is awkward, but because the questions I was answering are actual problem set questions that I'm planning on reusing in the future, I can't really put the solutions up here. Go check out the video for details!)

“Can you talk about how to choose the correct base case for a proof by induction?”

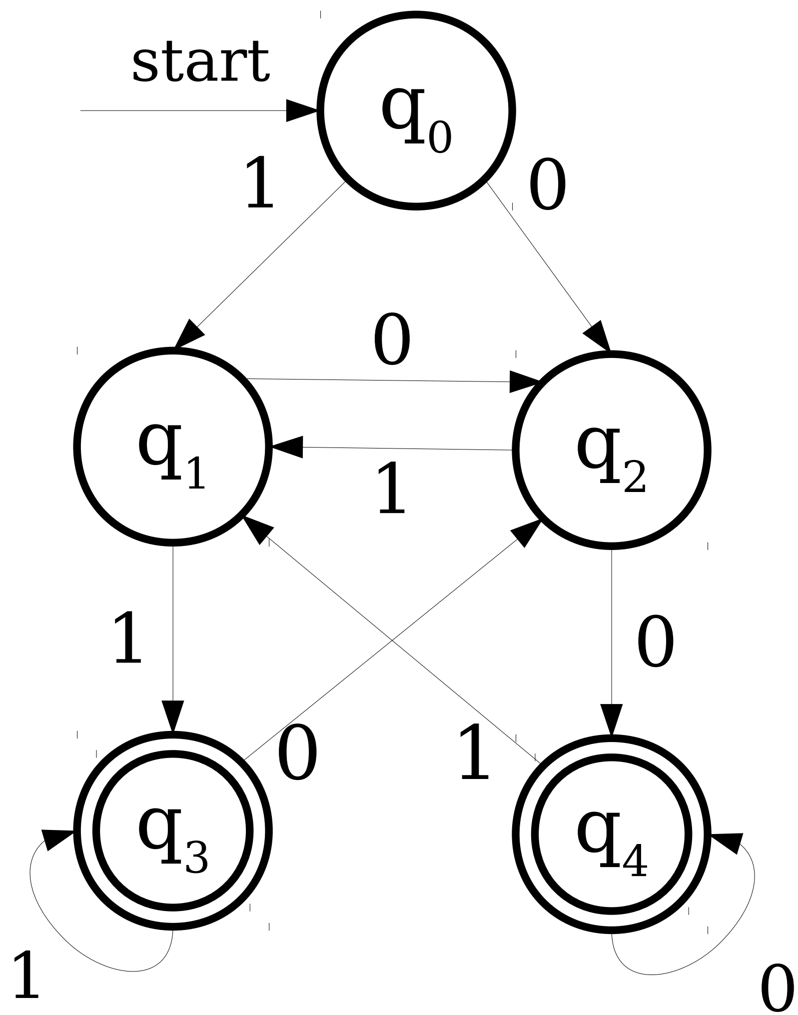
“How did you do on your CS103 midterm?”

“Can we please have some office hours over the weekend since the problem set is due on Monday?”

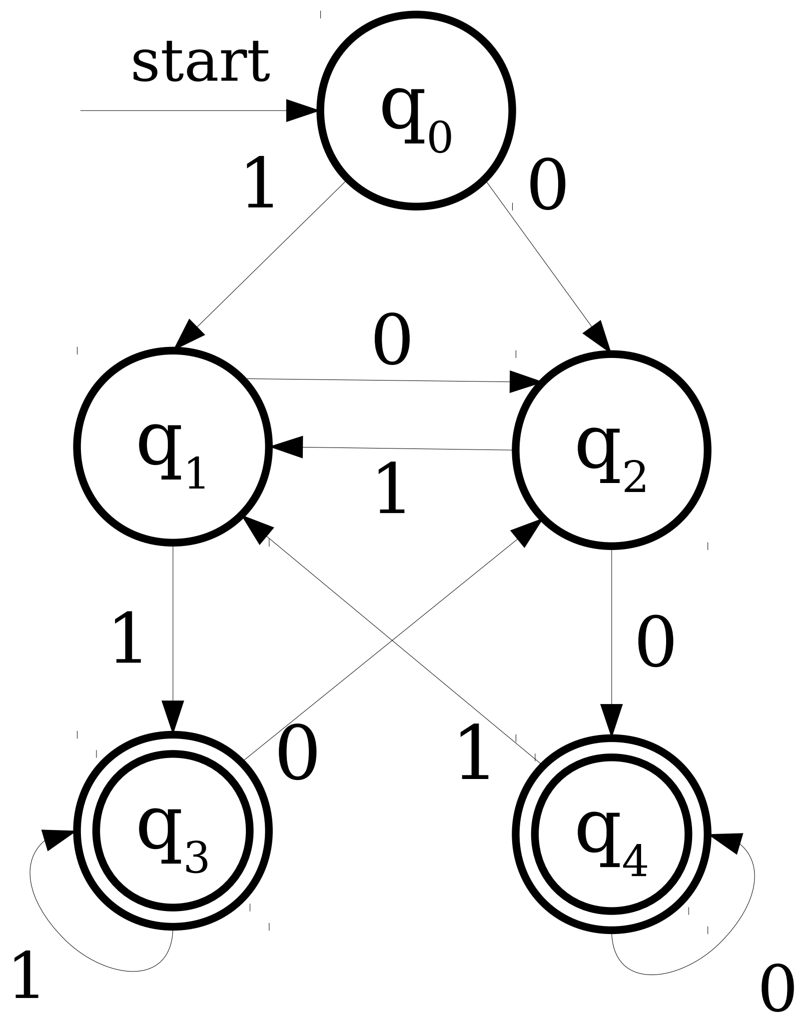
“What are some classes you wish you took as a student but never did?”

Back to CS103!

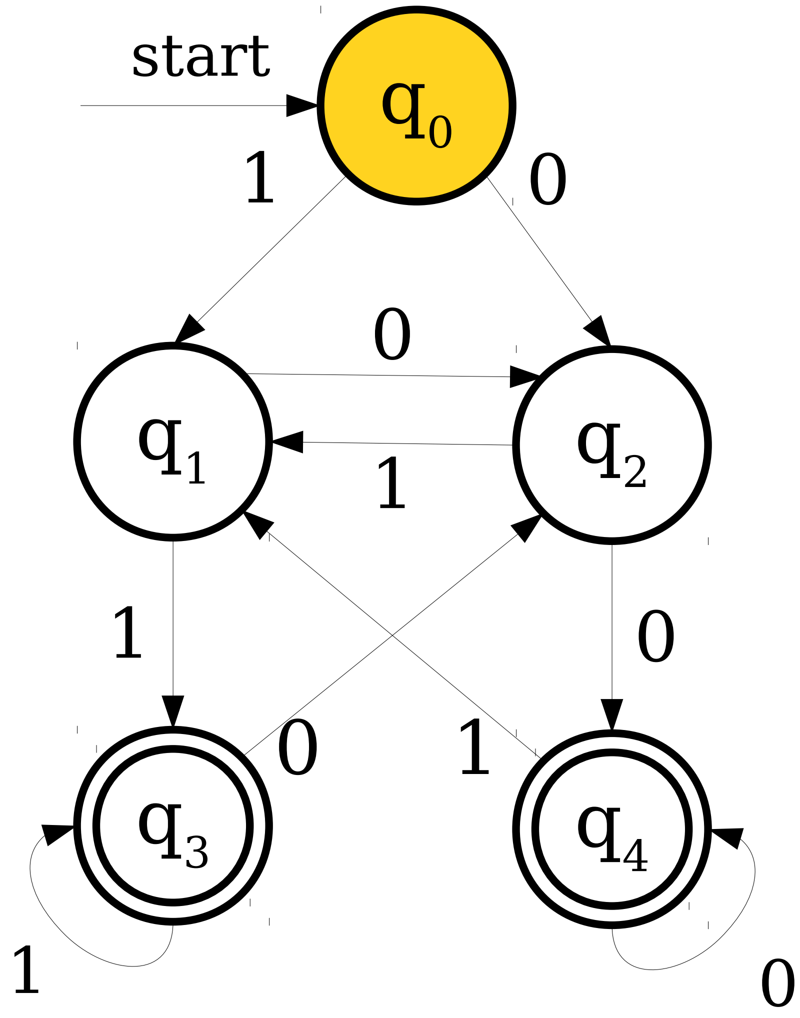
Accepting States, Revisited



Accepting States, Revisited

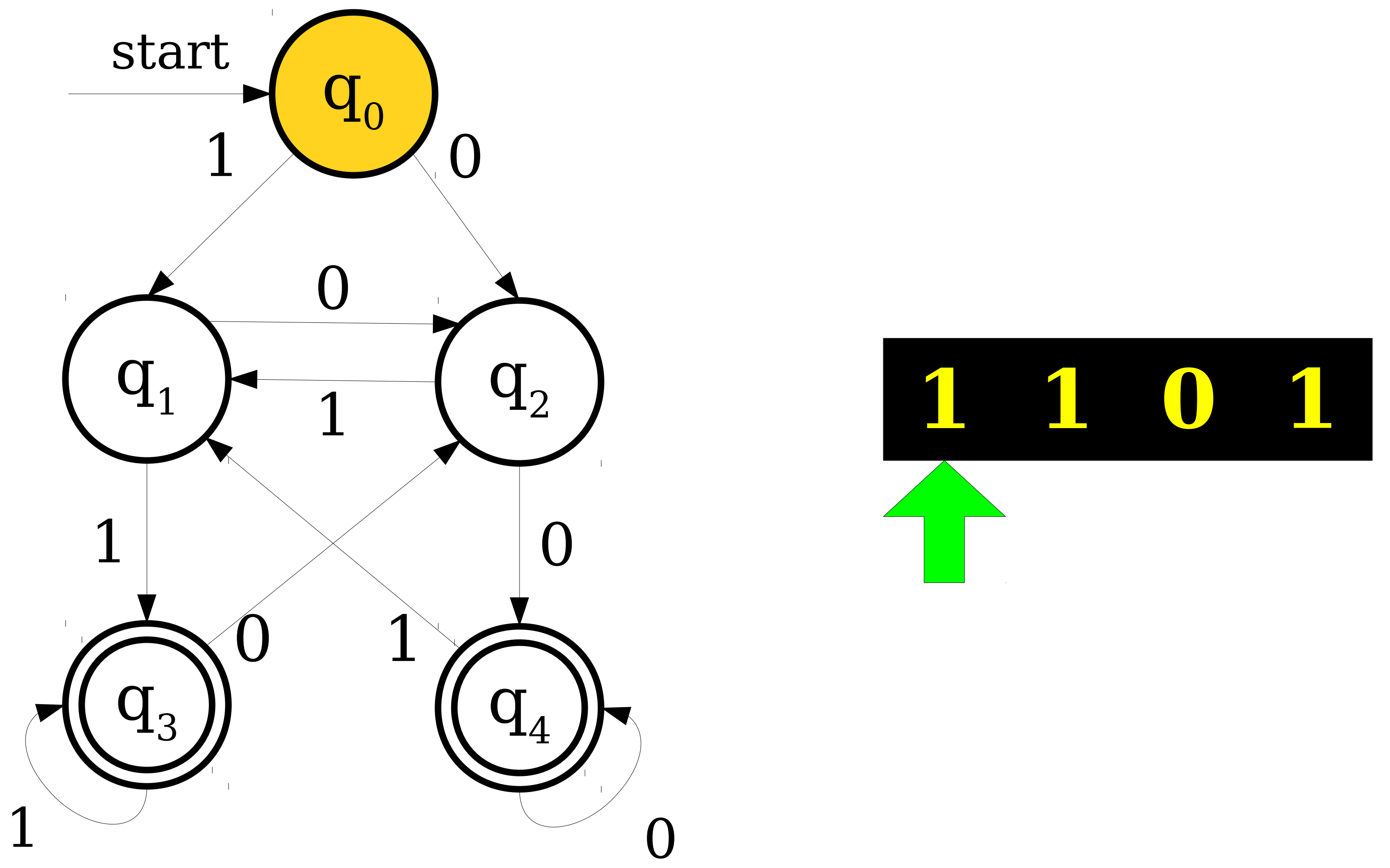


Accepting States, Revisited

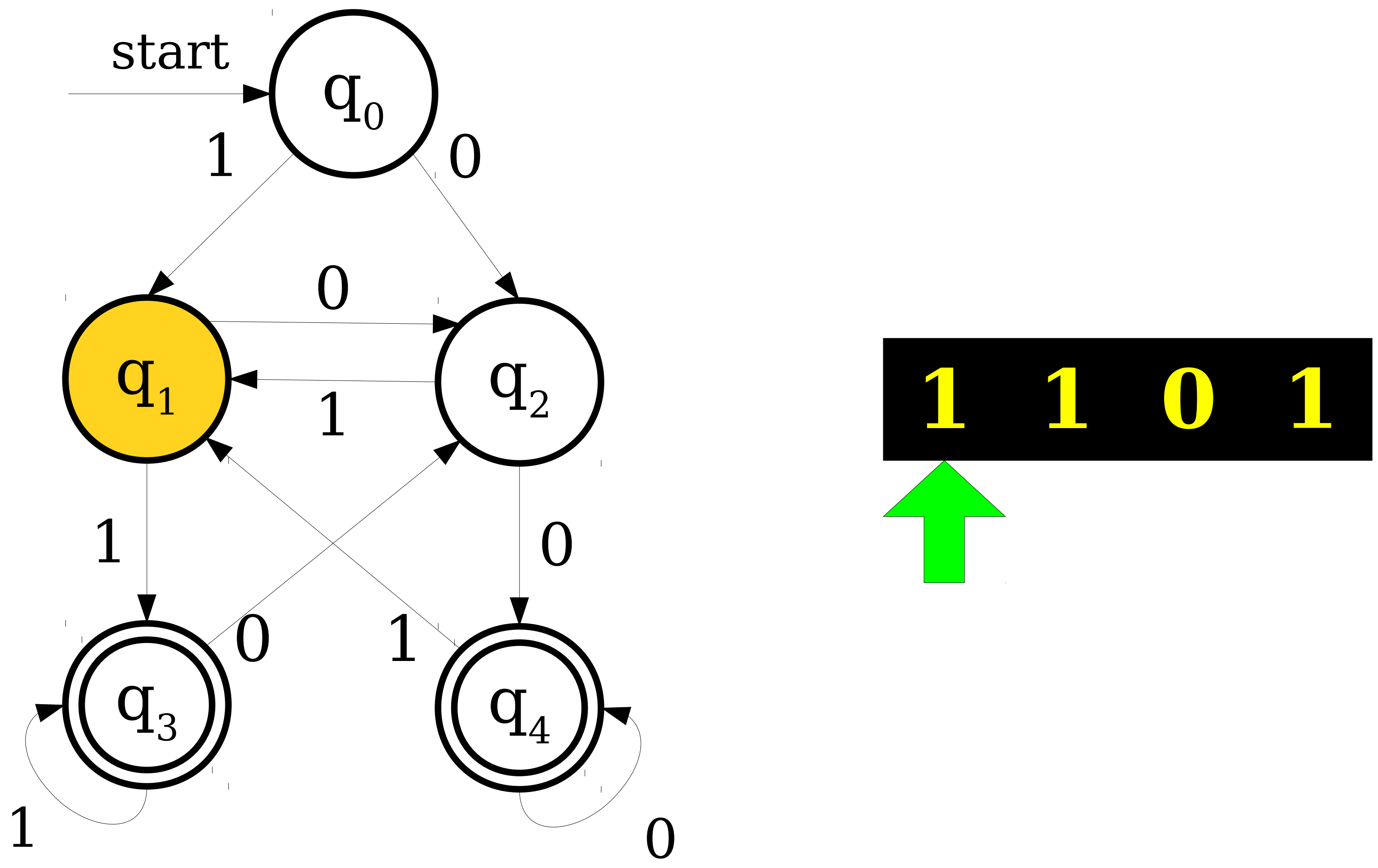


1 1 0 1

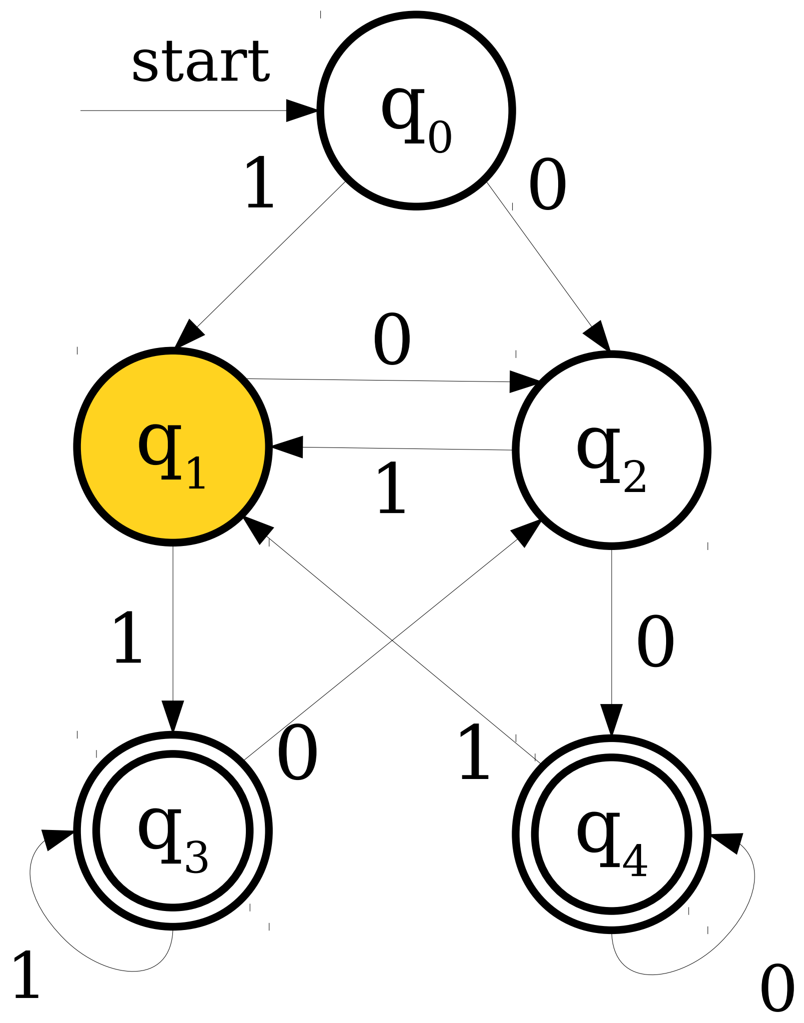
Accepting States, Revisited

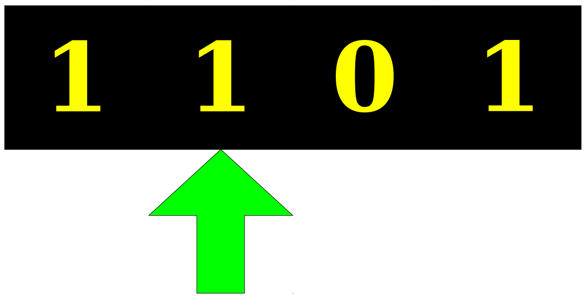


Accepting States, Revisited

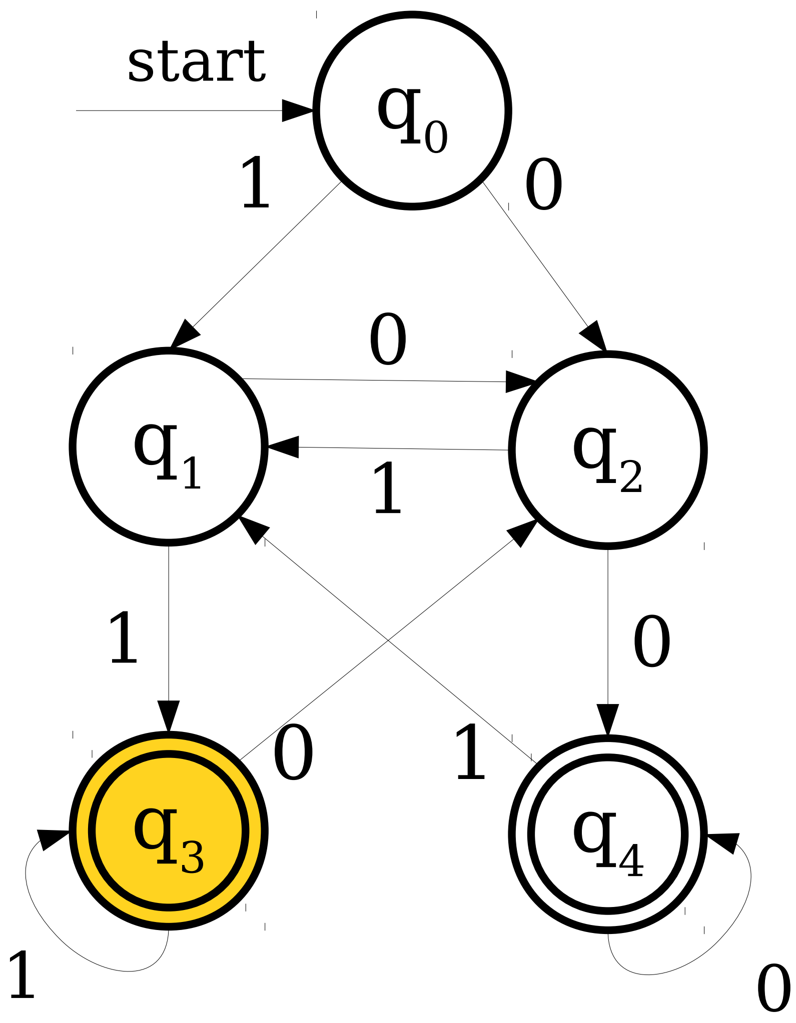


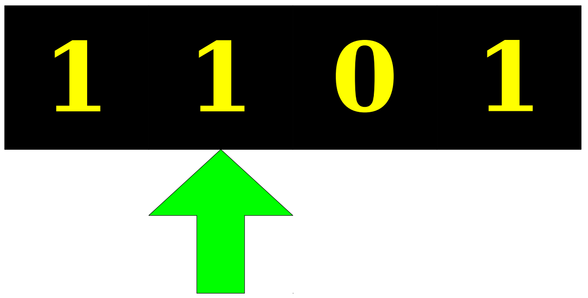
Accepting States, Revisited



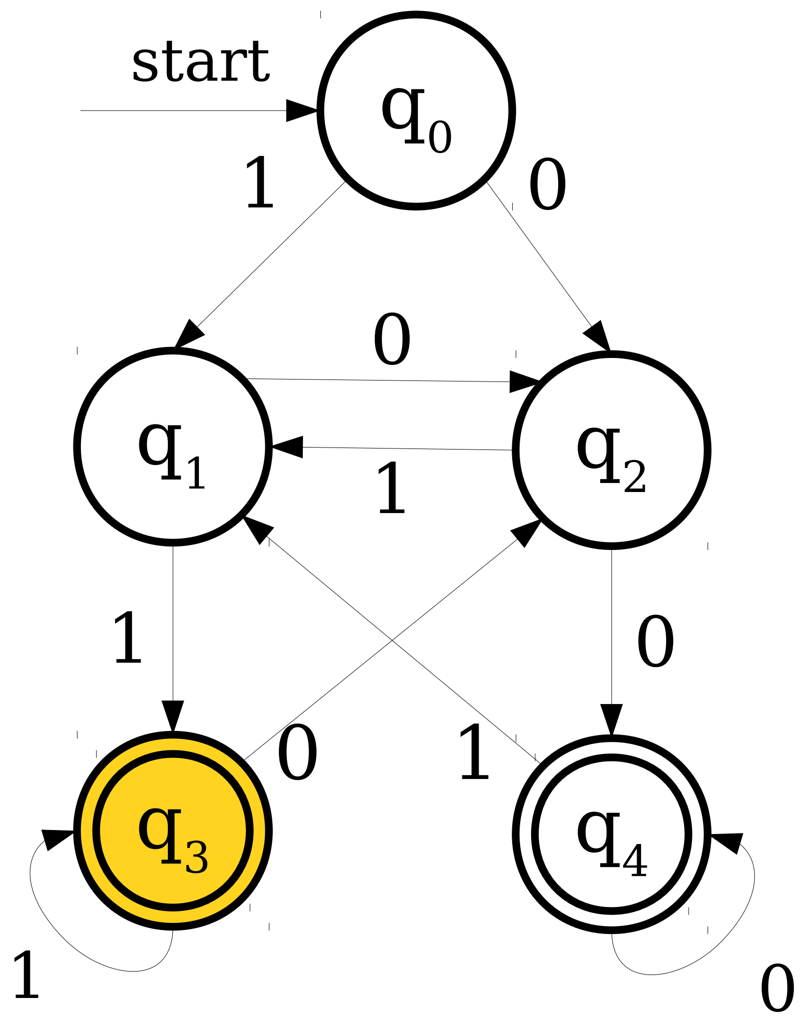


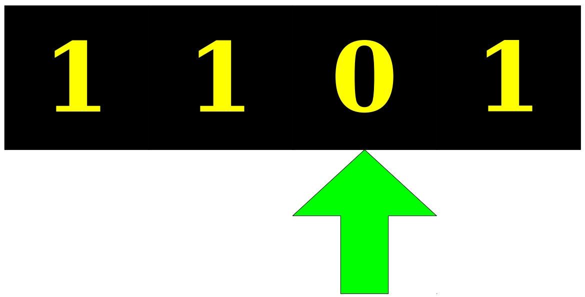
Accepting States, Revisited



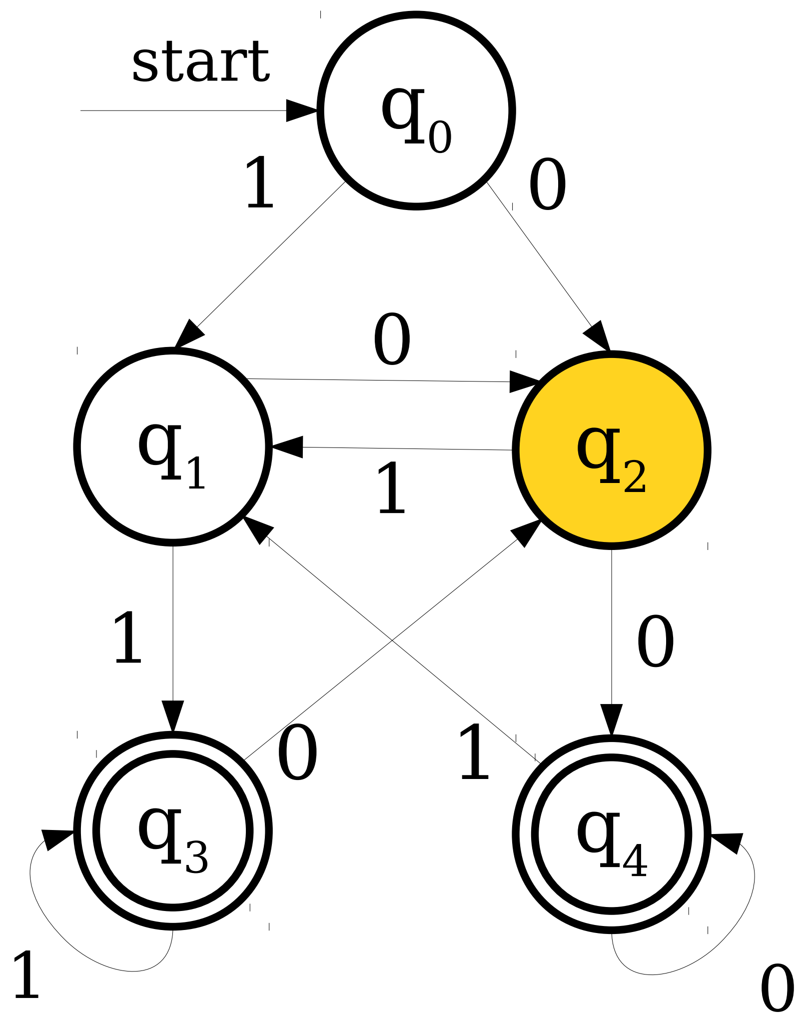


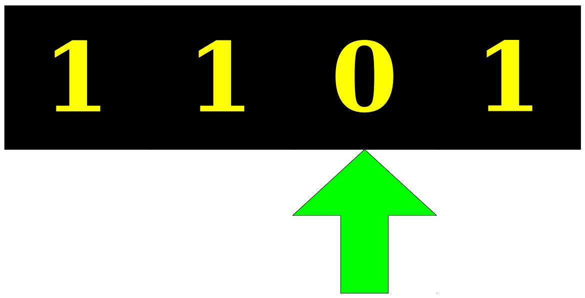
Accepting States, Revisited



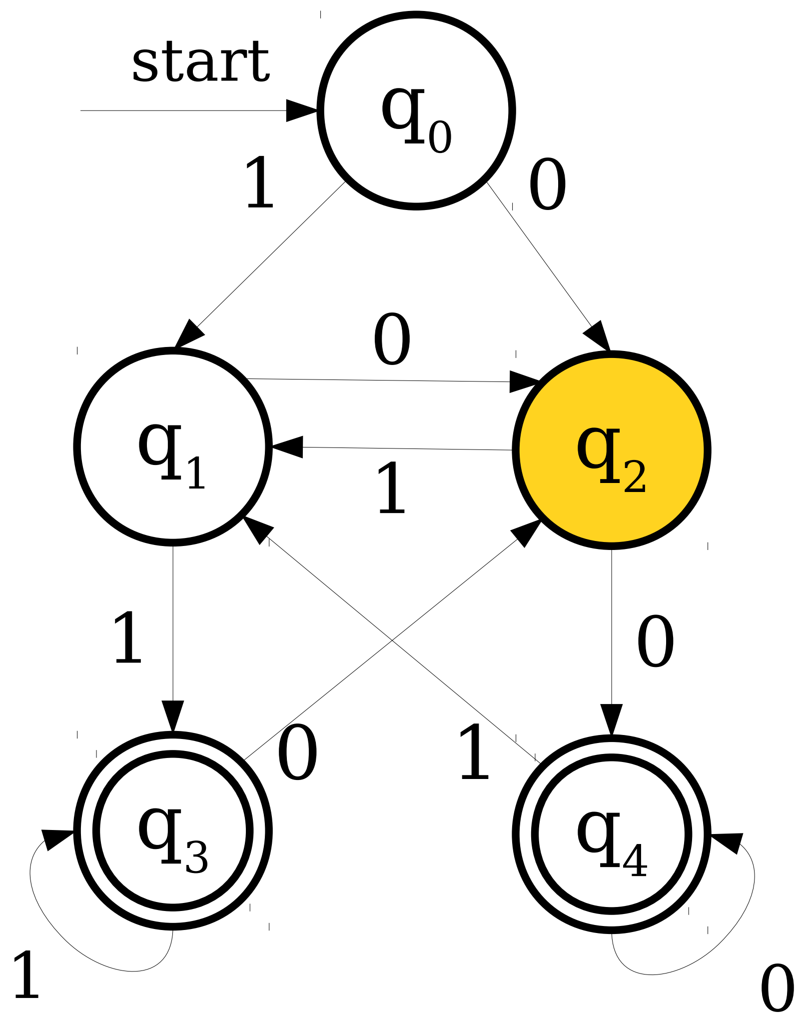


Accepting States, Revisited

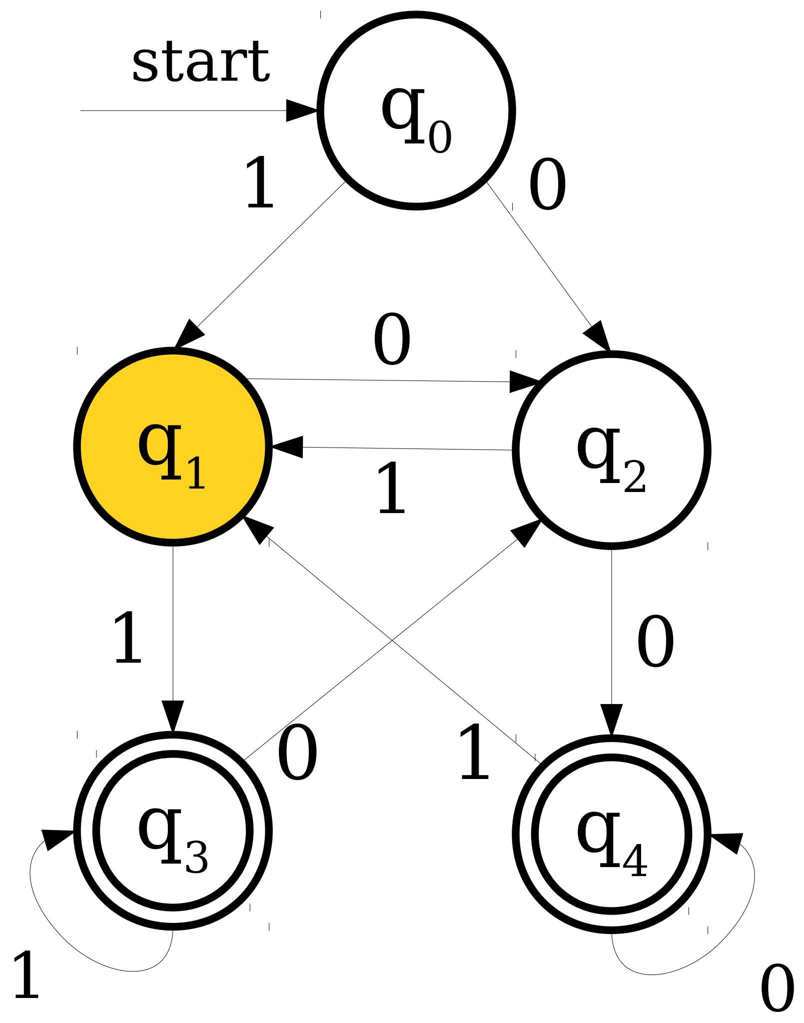




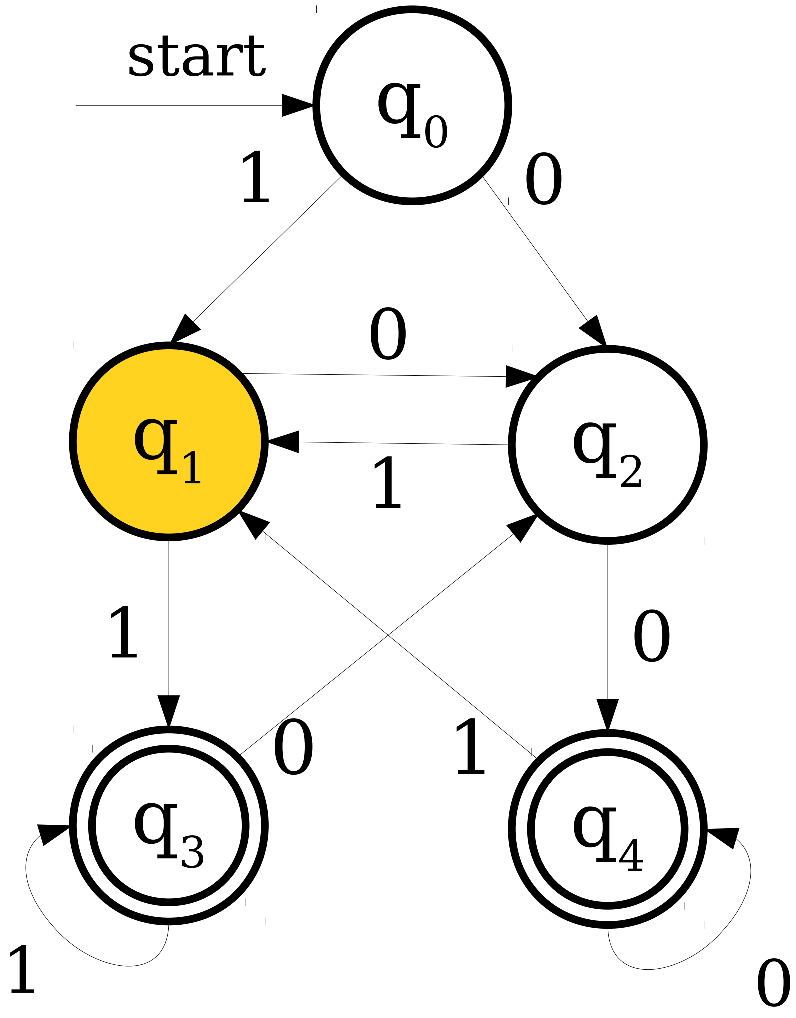
Accepting States, Revisited



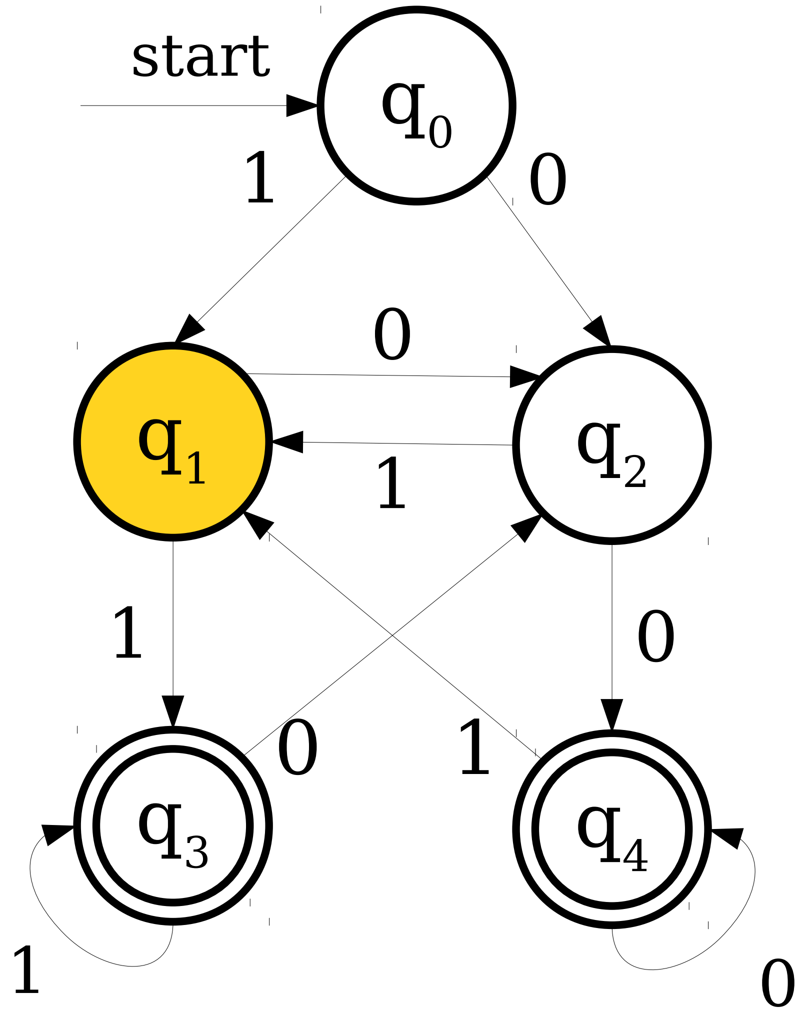
Accepting States, Revisited



Accepting States, Revisited

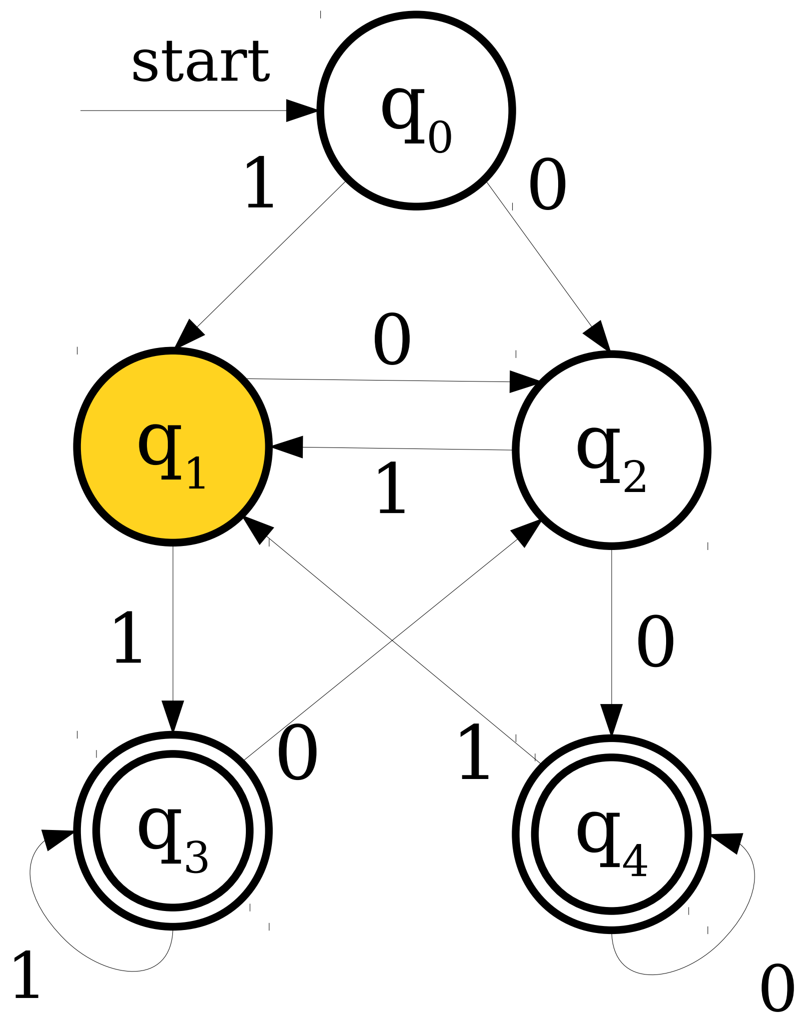


Accepting States, Revisited



1 1 0 1

Accepting States, Revisited

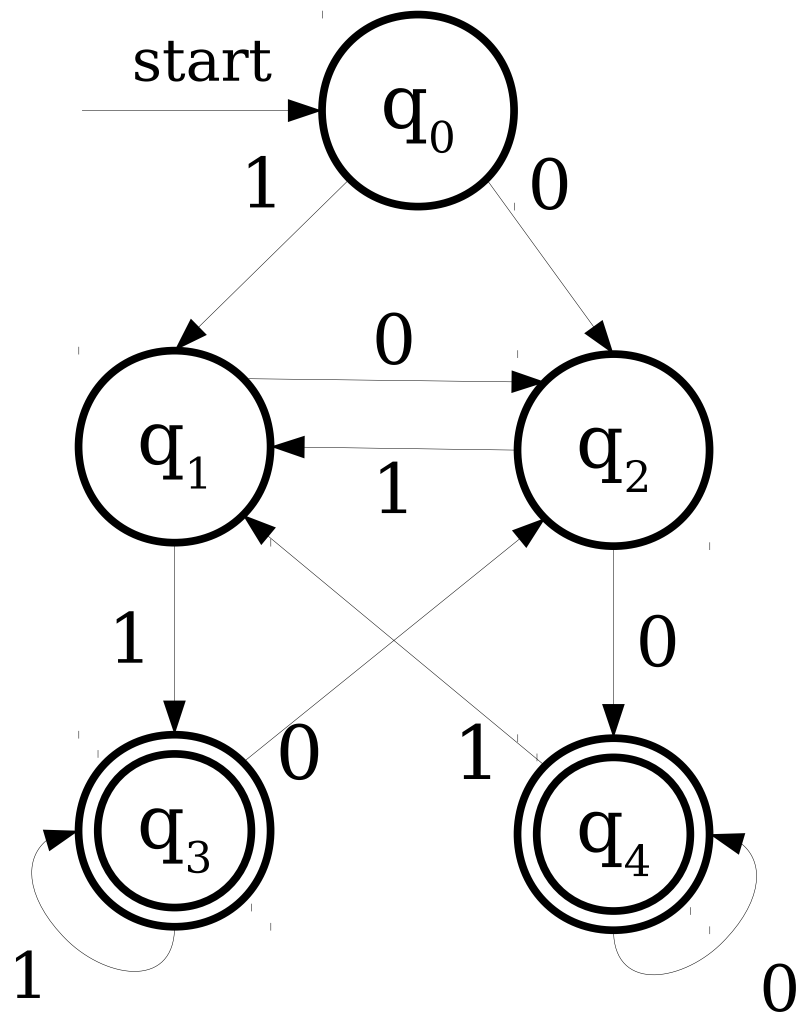


1 1 0 1

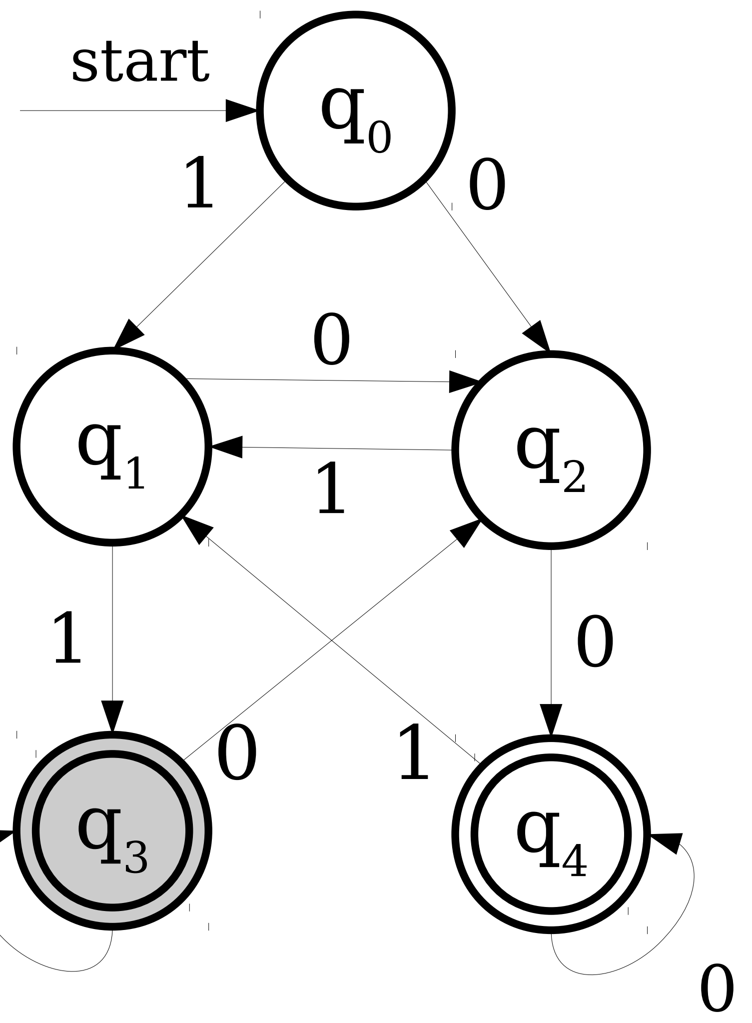
A finite automaton does ***not*** accept as soon as it enters an accepting state.

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

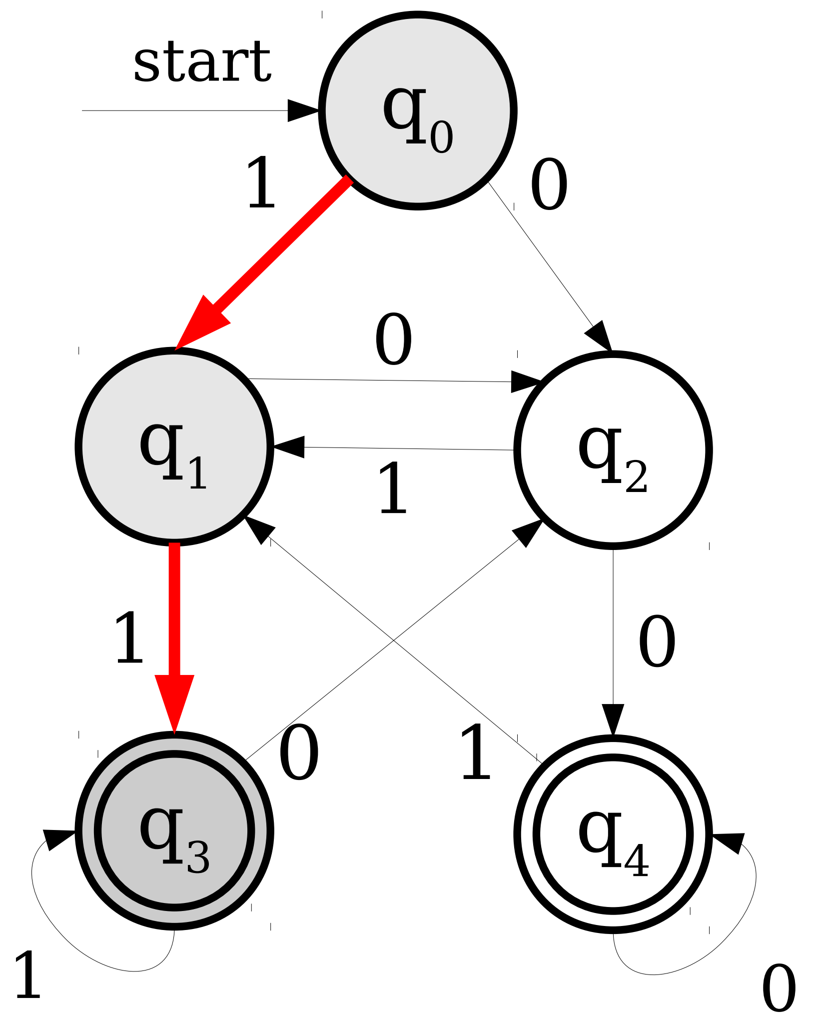
What Does This Accept?



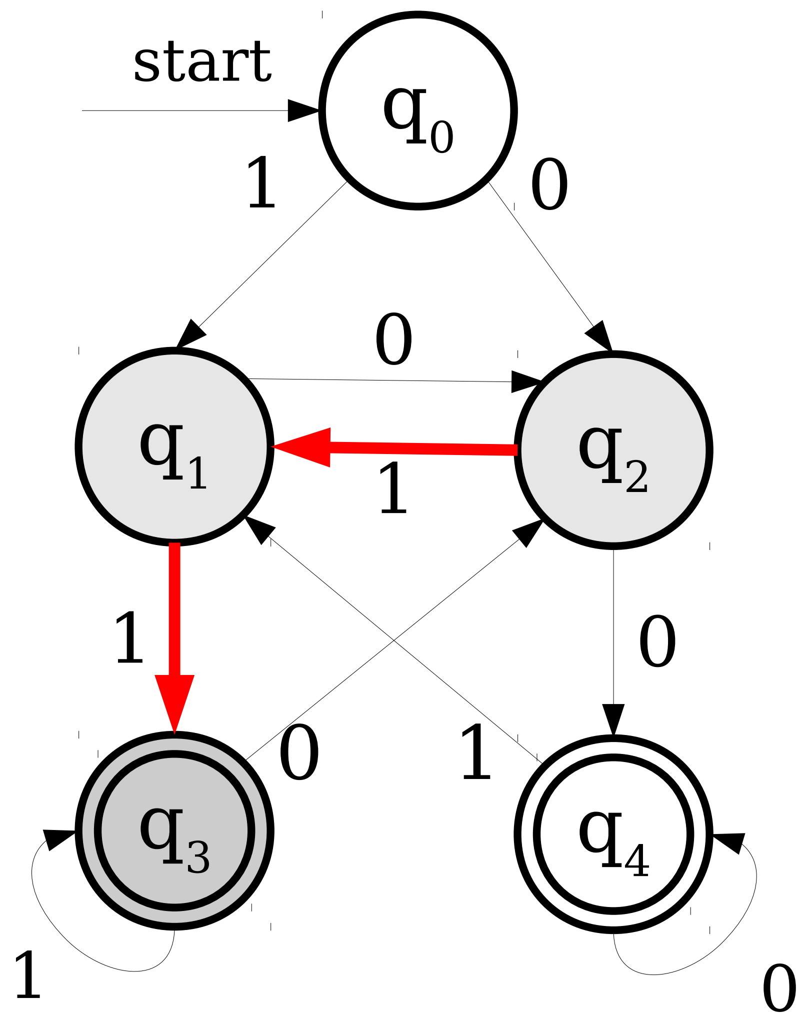
What Does This Accept?



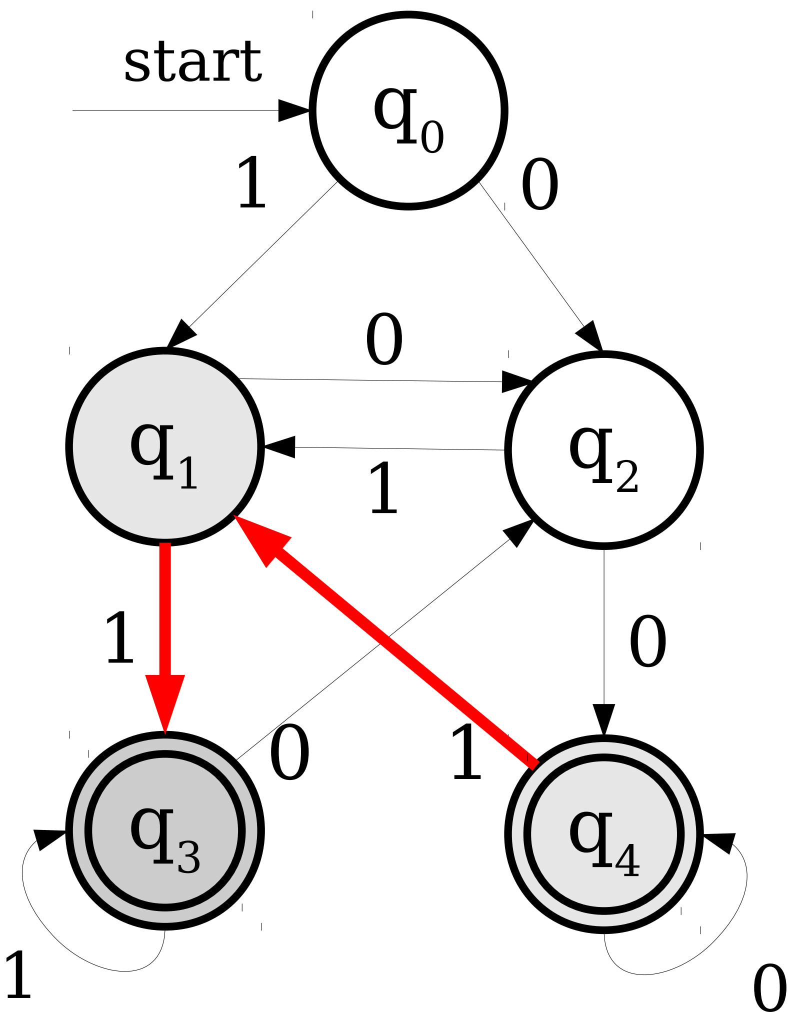
What Does This Accept?



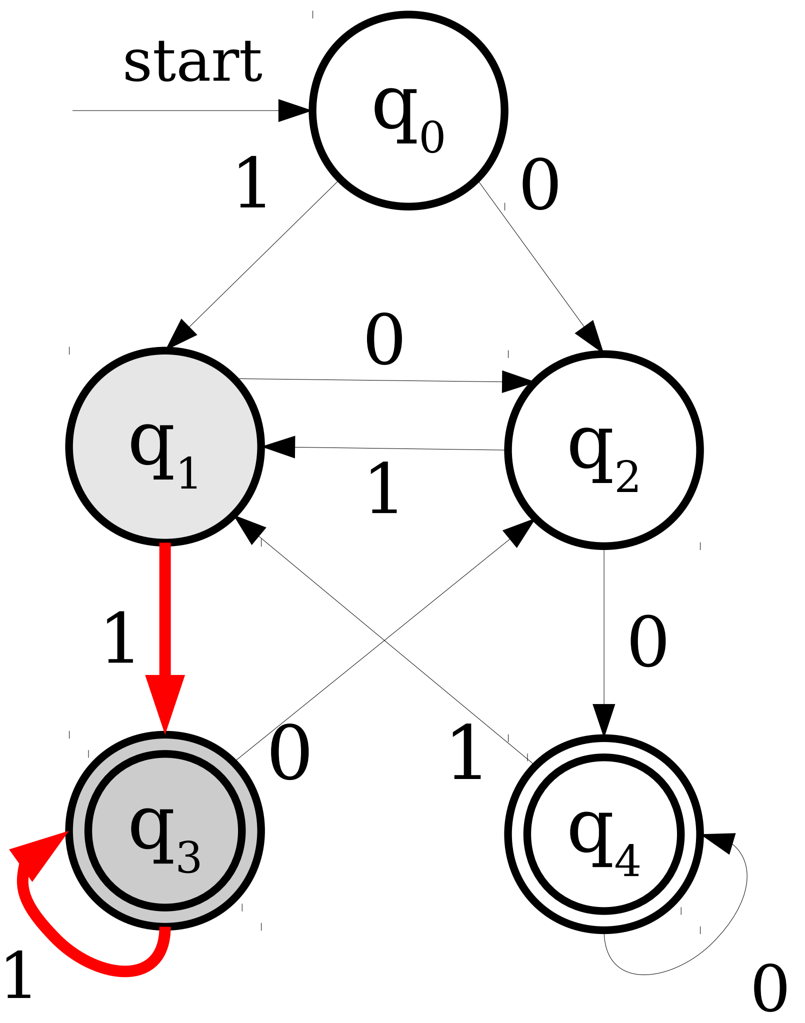
What Does This Accept?



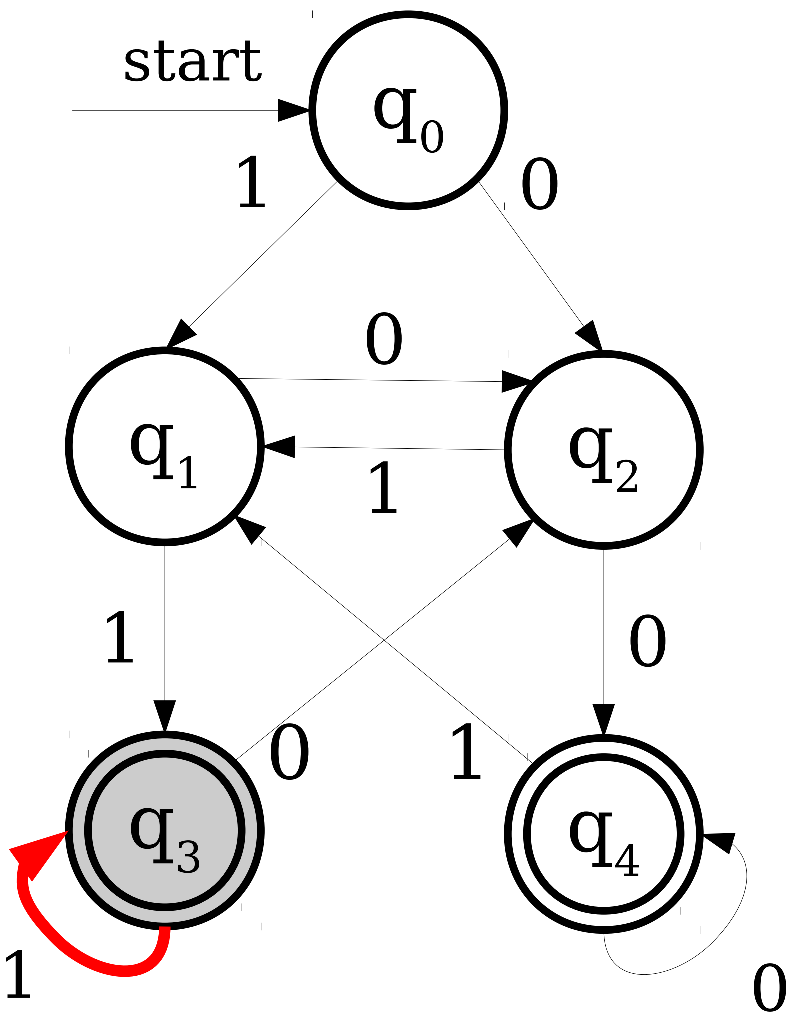
What Does This Accept?



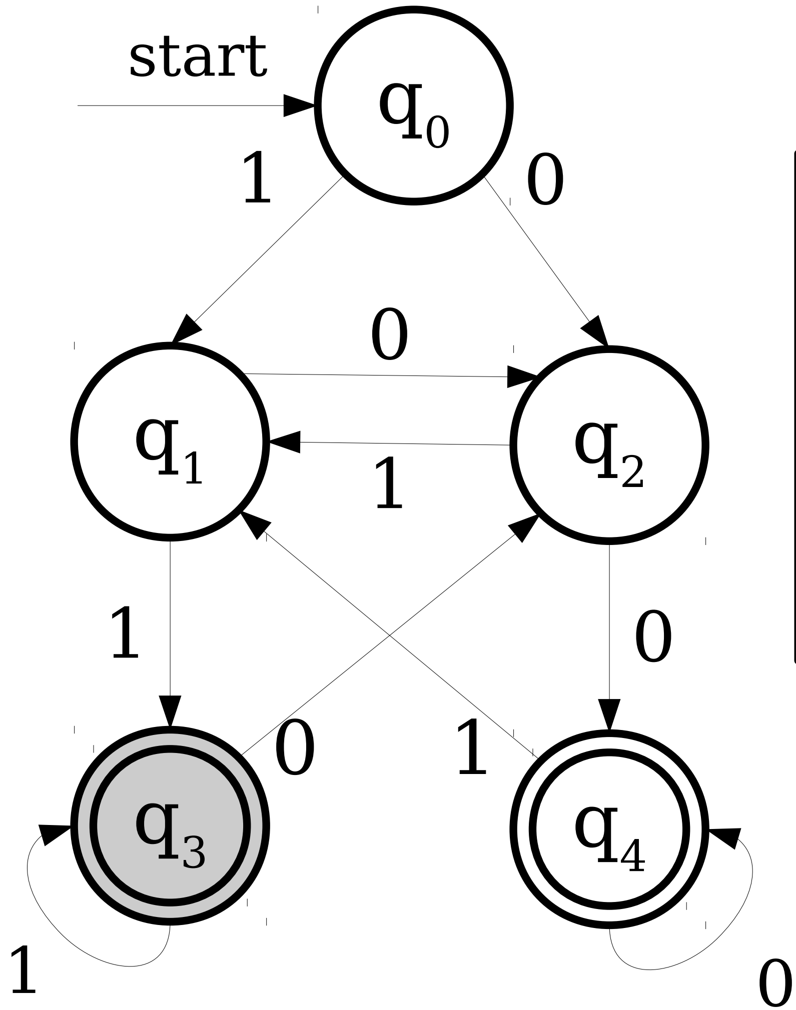
What Does This Accept?



What Does This Accept?

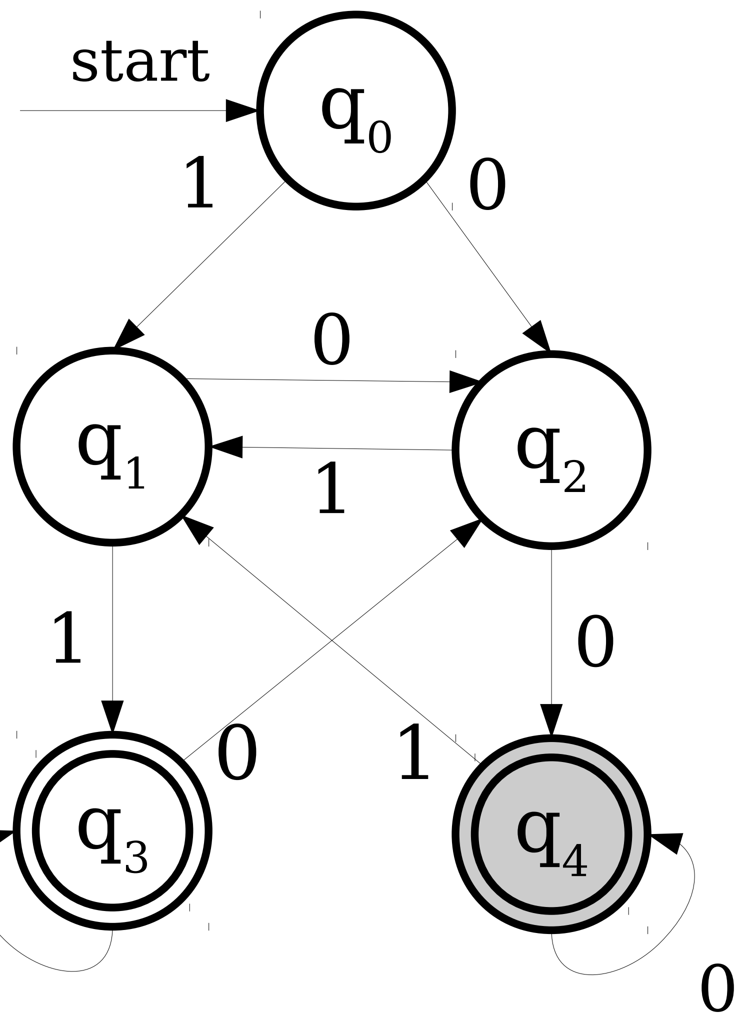


What Does This Accept?

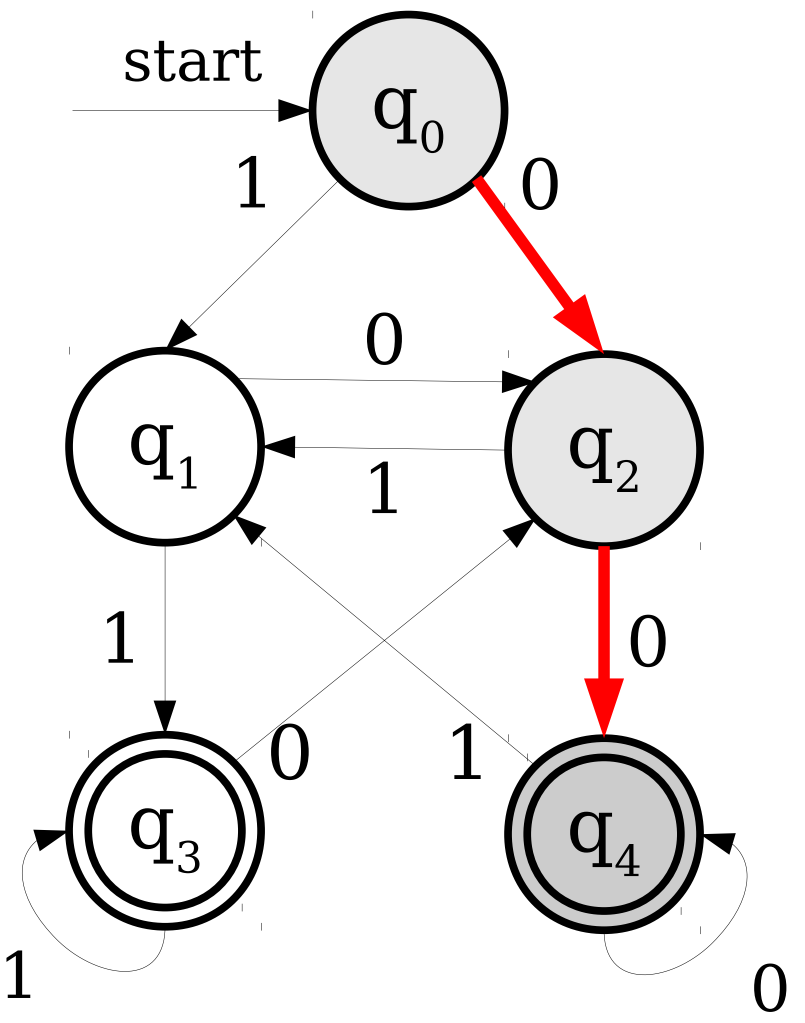


No matter where we start in the automaton, after seeing two 1's, we end up in accepting state q_3.

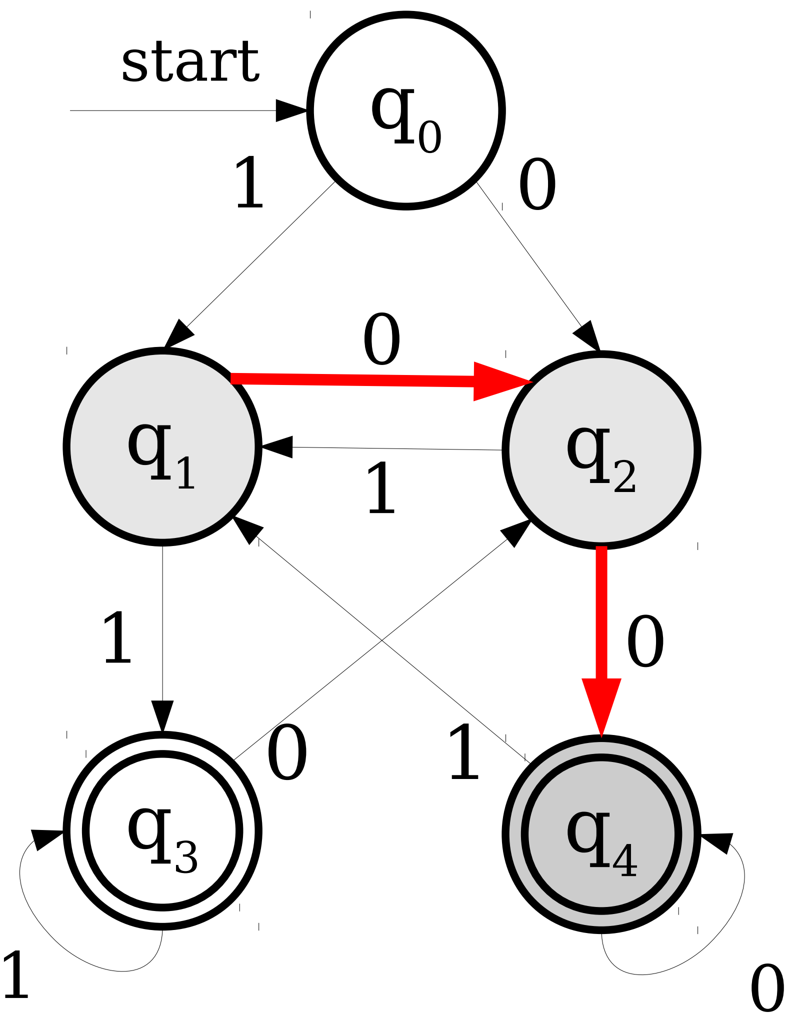
What Does This Accept?



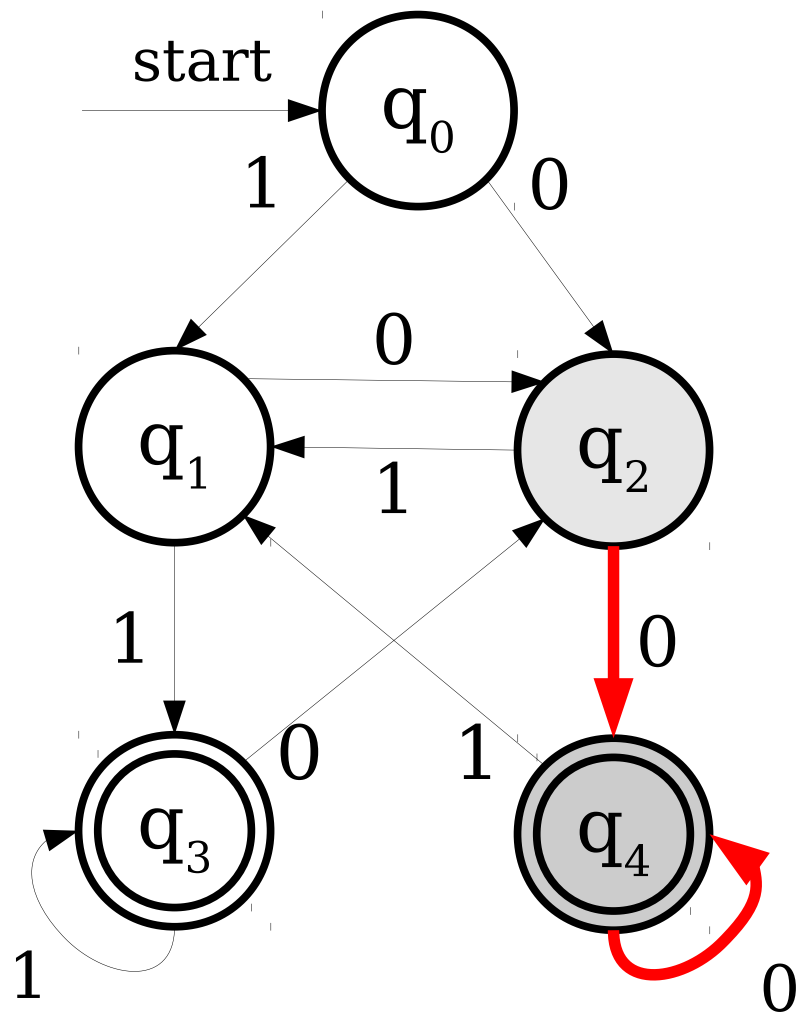
What Does This Accept?



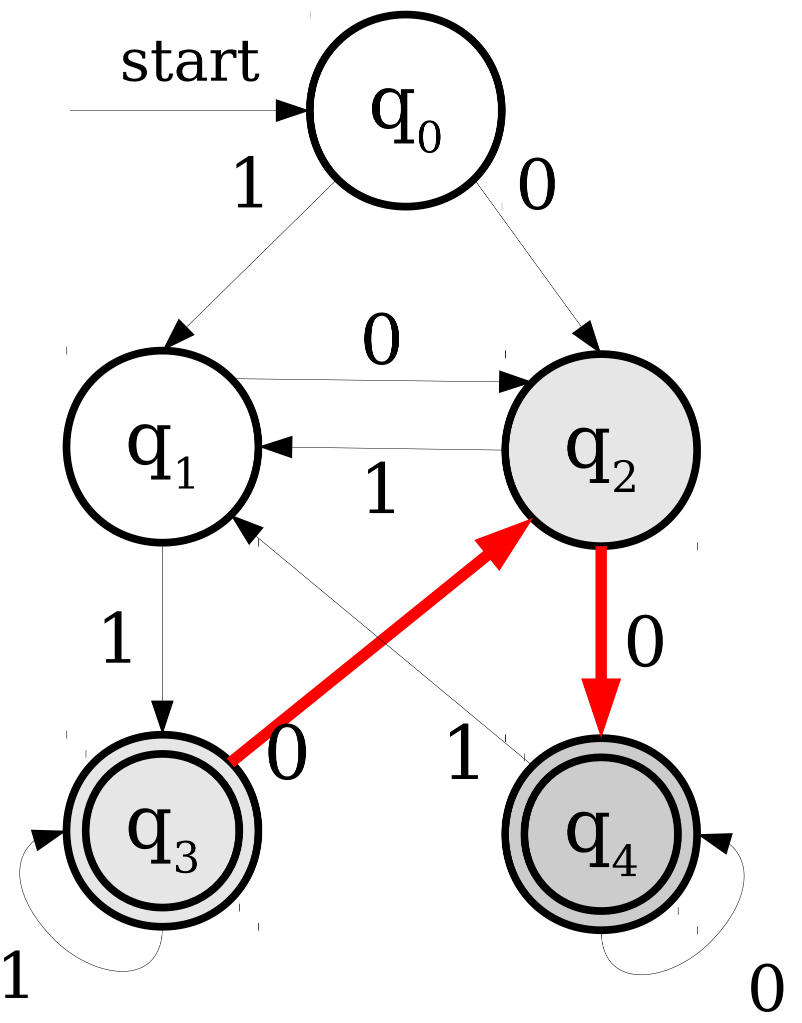
What Does This Accept?



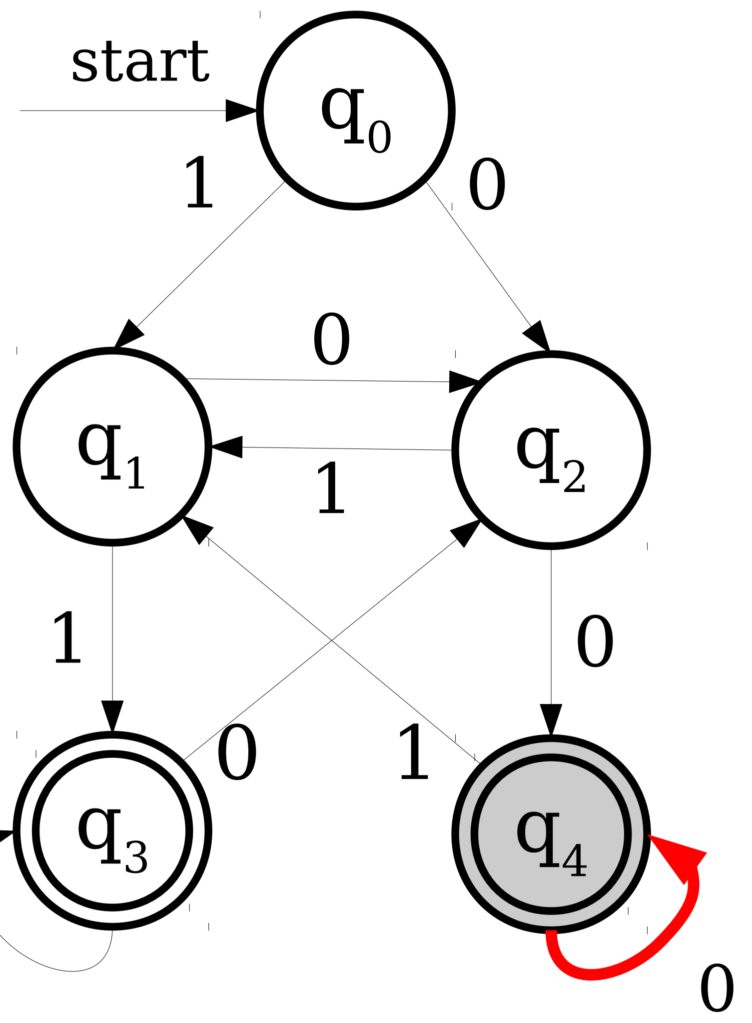
What Does This Accept?



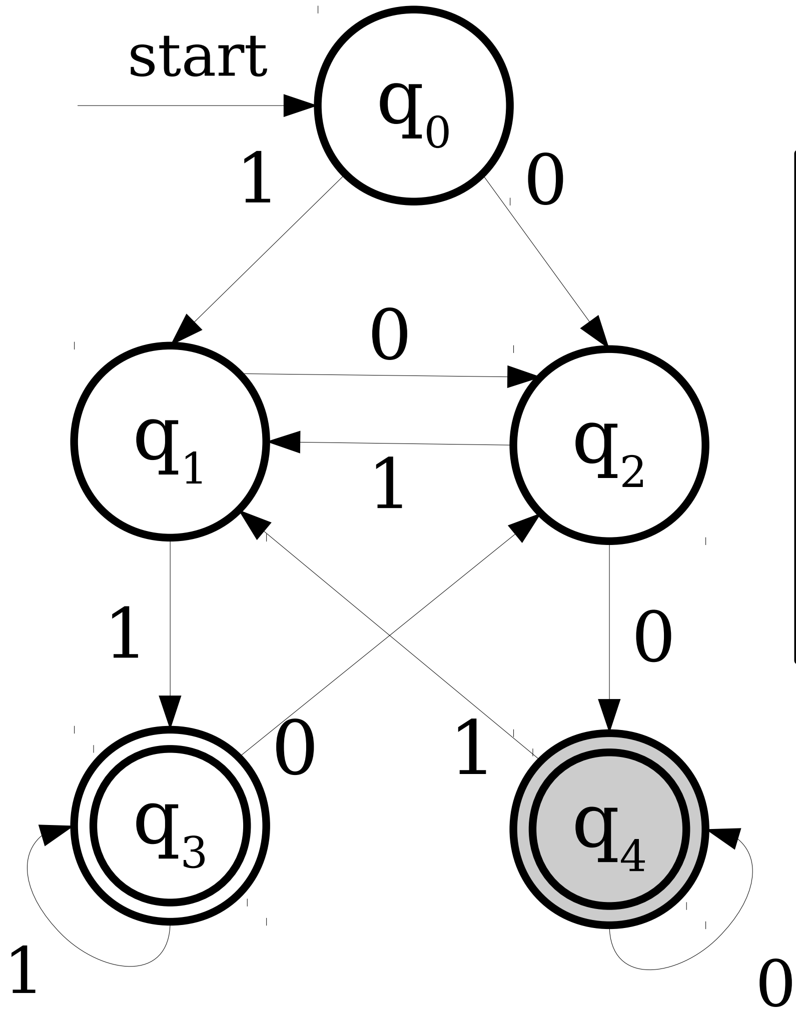
What Does This Accept?



What Does This Accept?

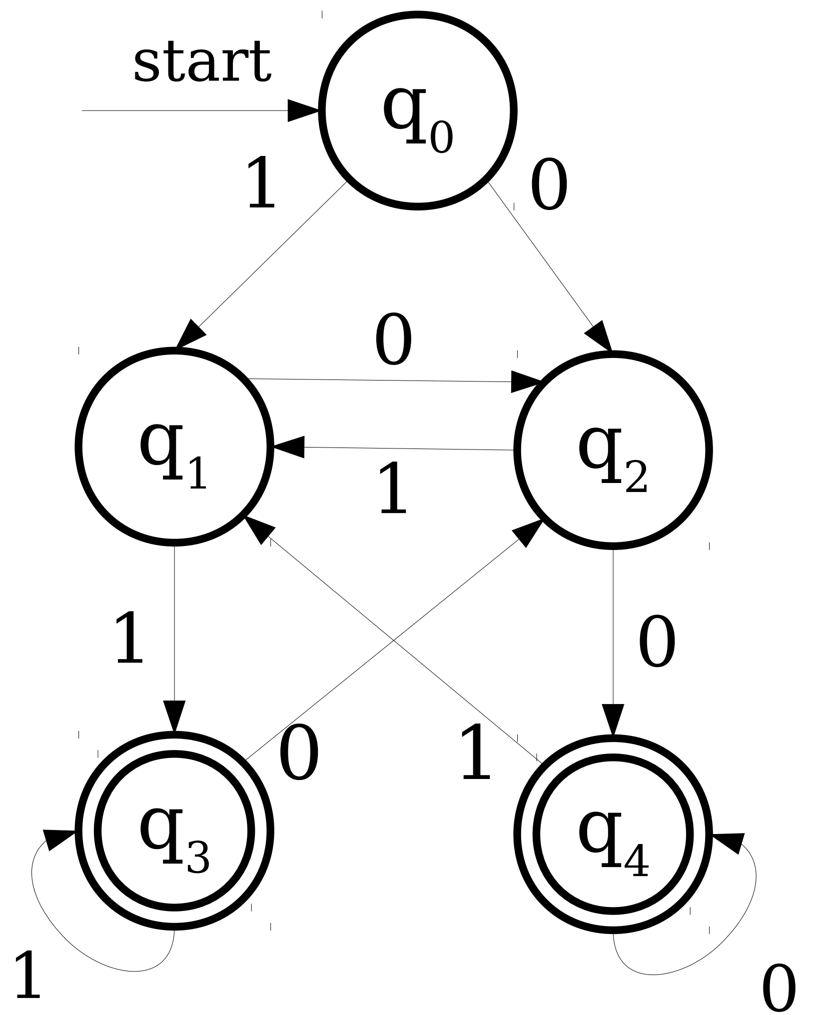


What Does This Accept?

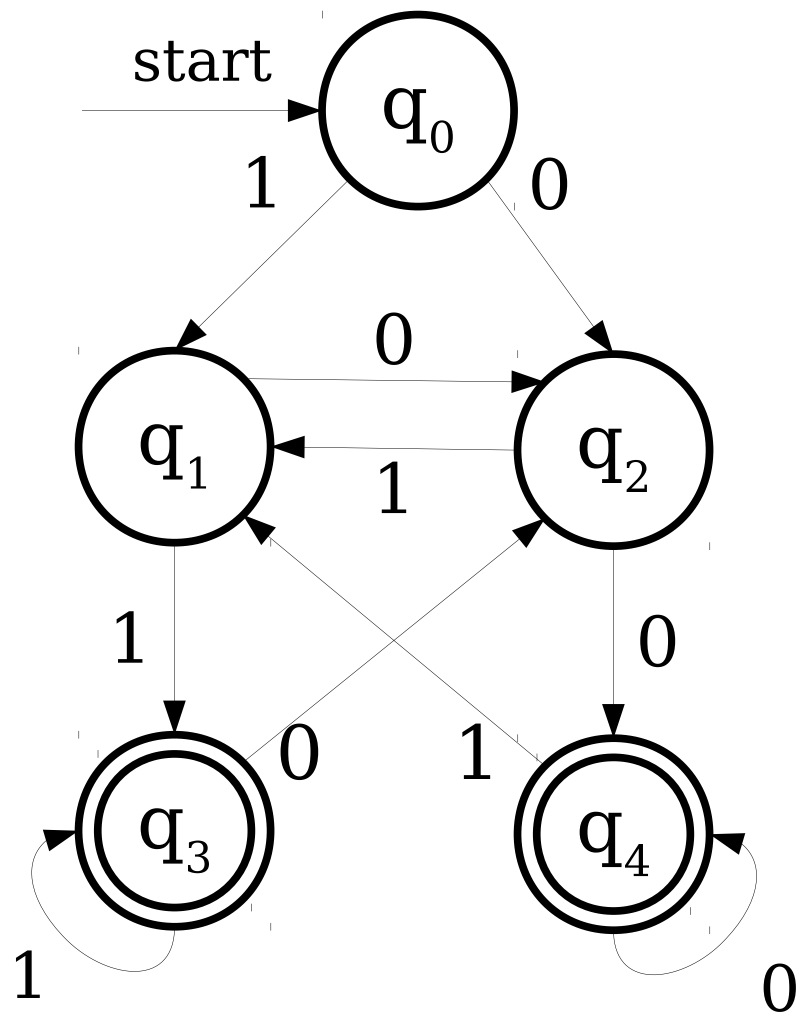


No matter where we start in the automaton, after seeing two 0's, we end up in accepting state q_5.

What Does This Accept?



What Does This Accept?



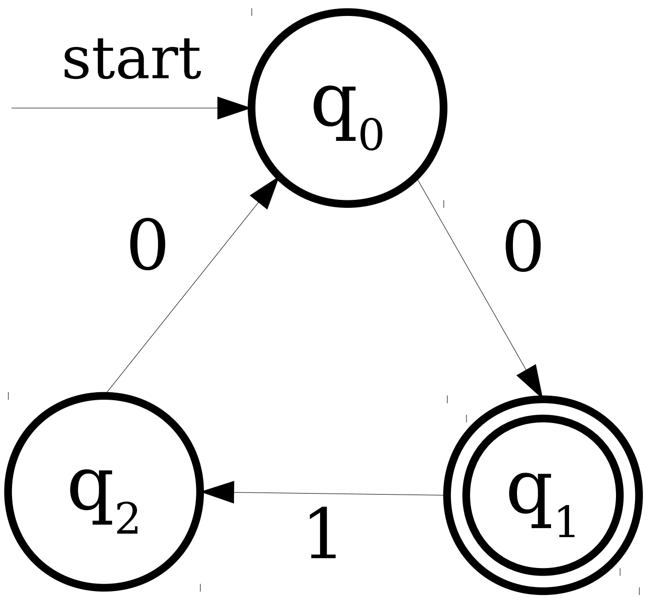
This automaton accepts a string iff it ends in 00 or 11.

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

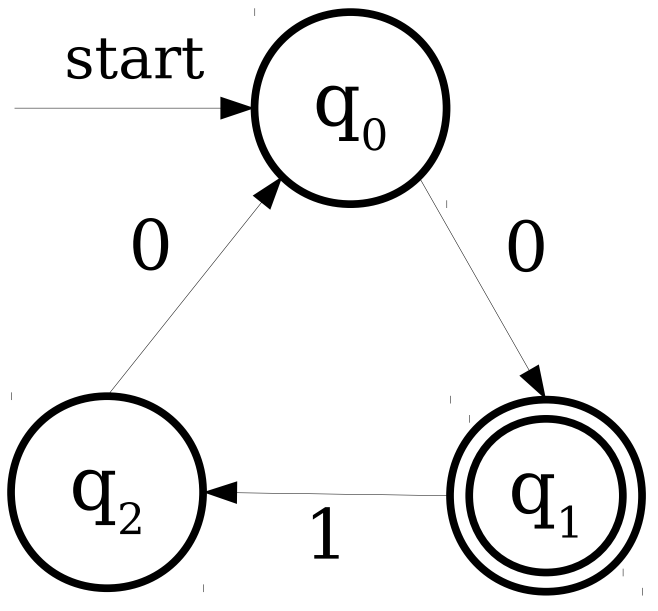
If D is an automaton, we denote the language of D as $\mathscr{L}(\boldsymbol{D})$.

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

A Small Problem

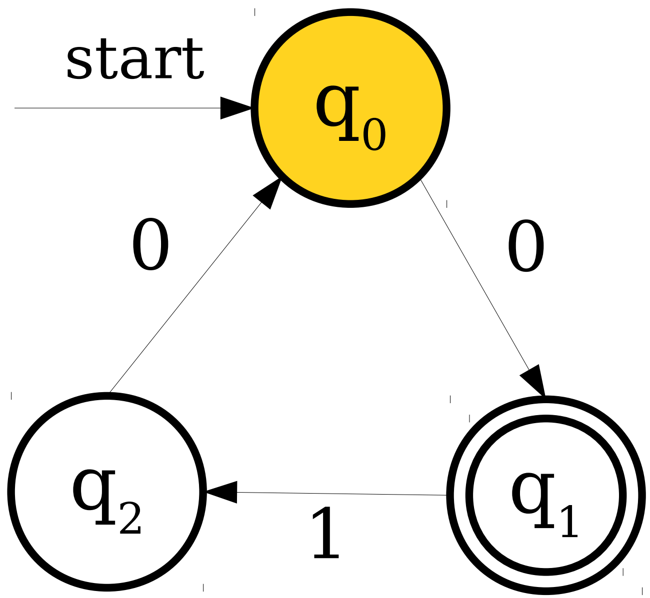


A Small Problem



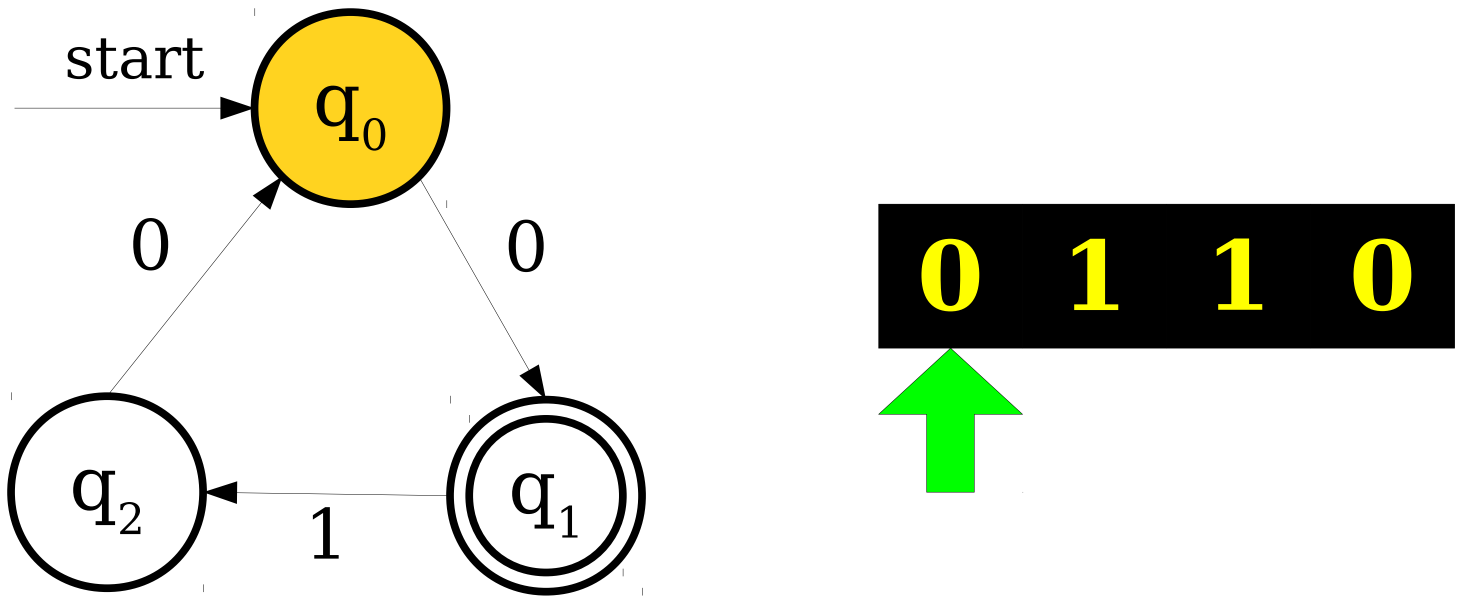
0	**1**	**1**	**0**

A Small Problem

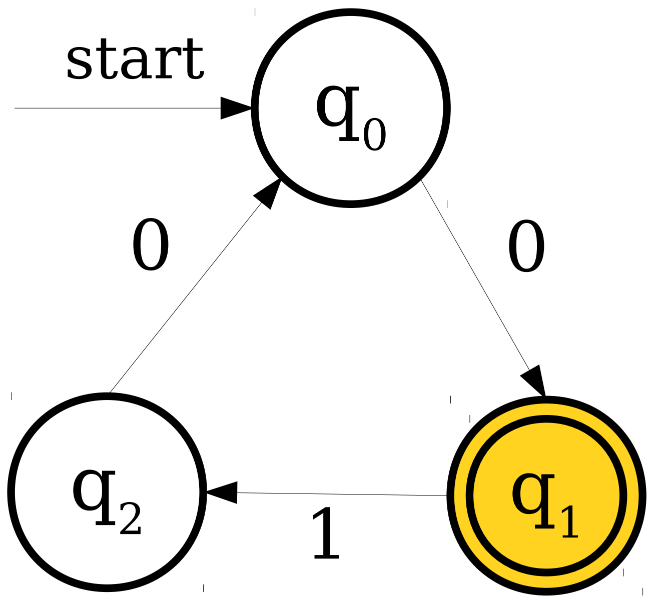


0 1 1 0

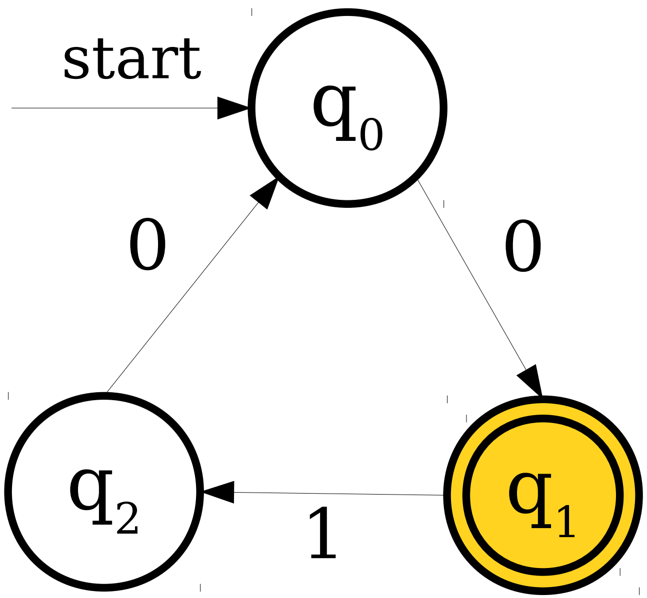
A Small Problem

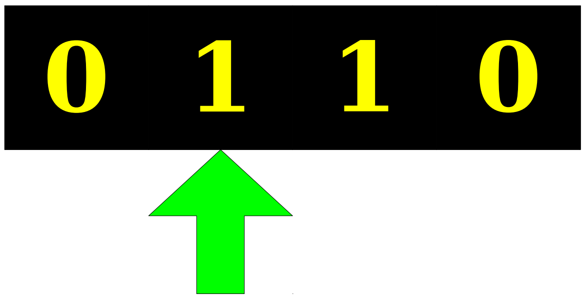


A Small Problem

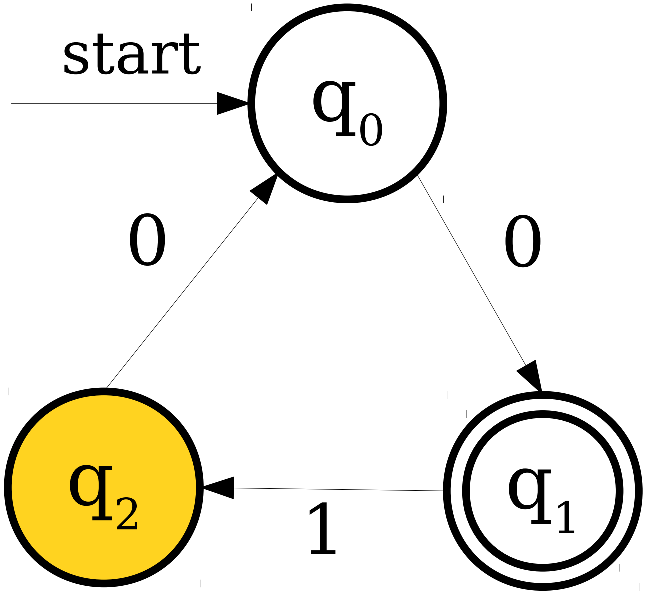


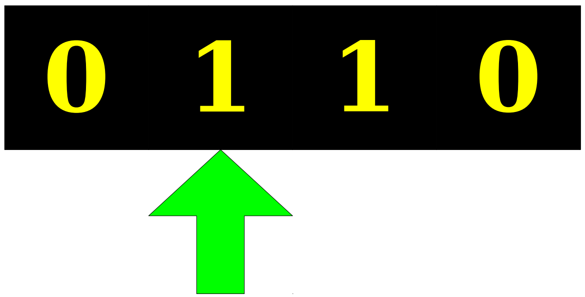
A Small Problem



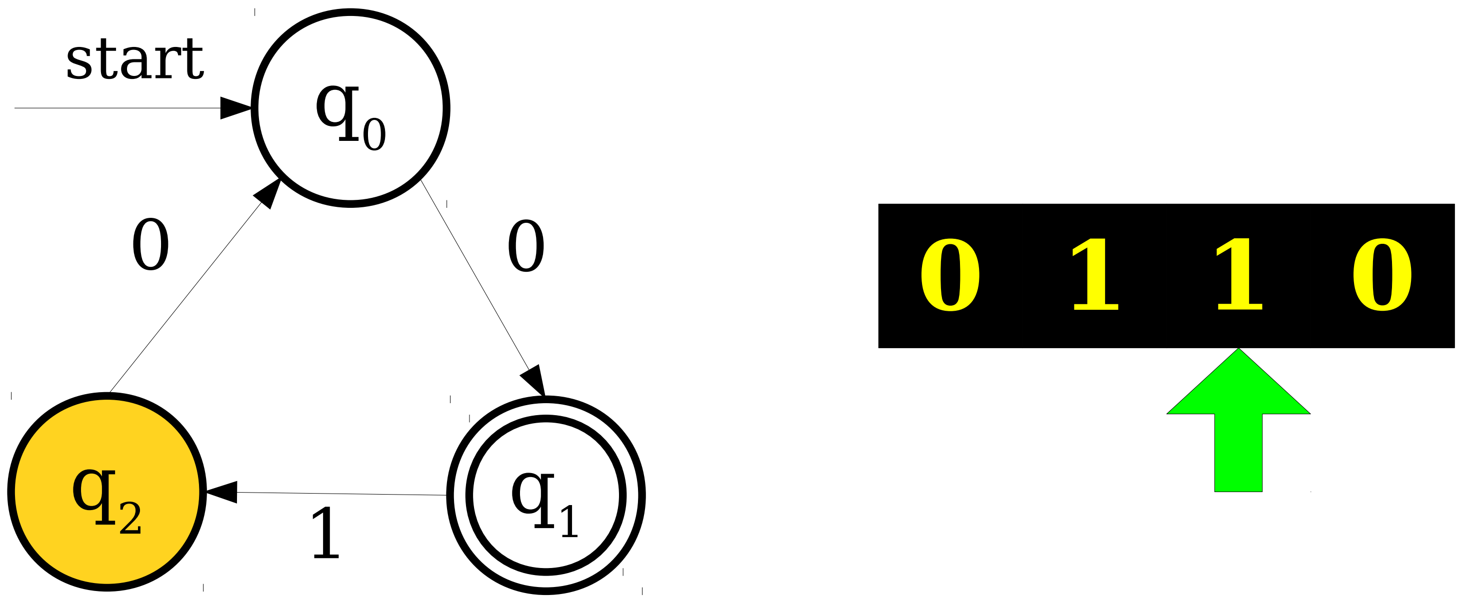


A Small Problem

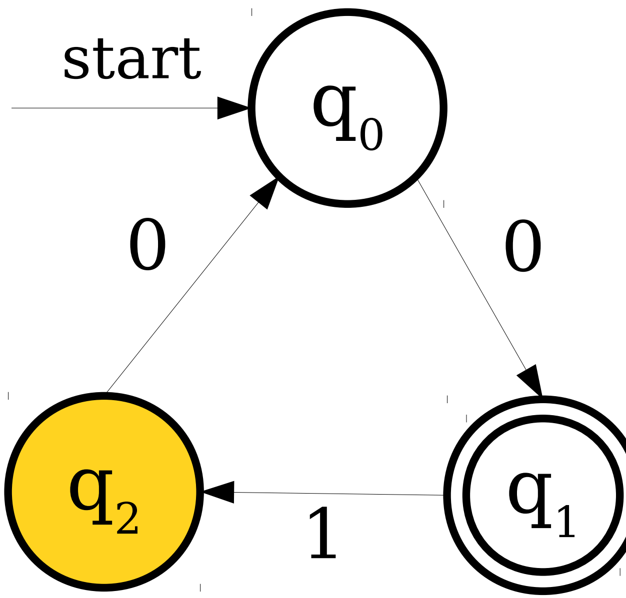




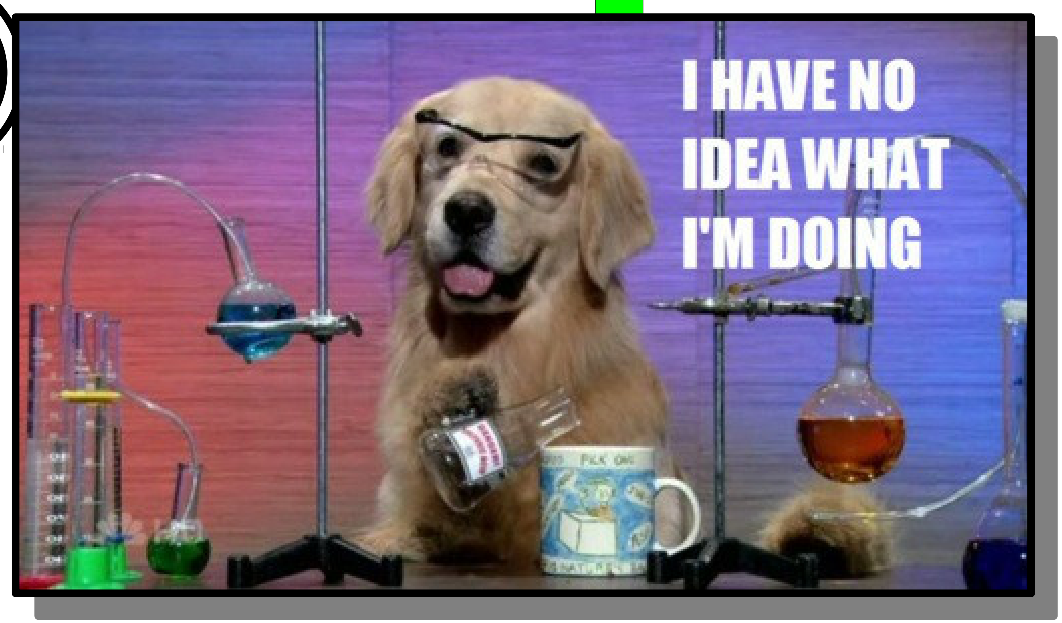
A Small Problem



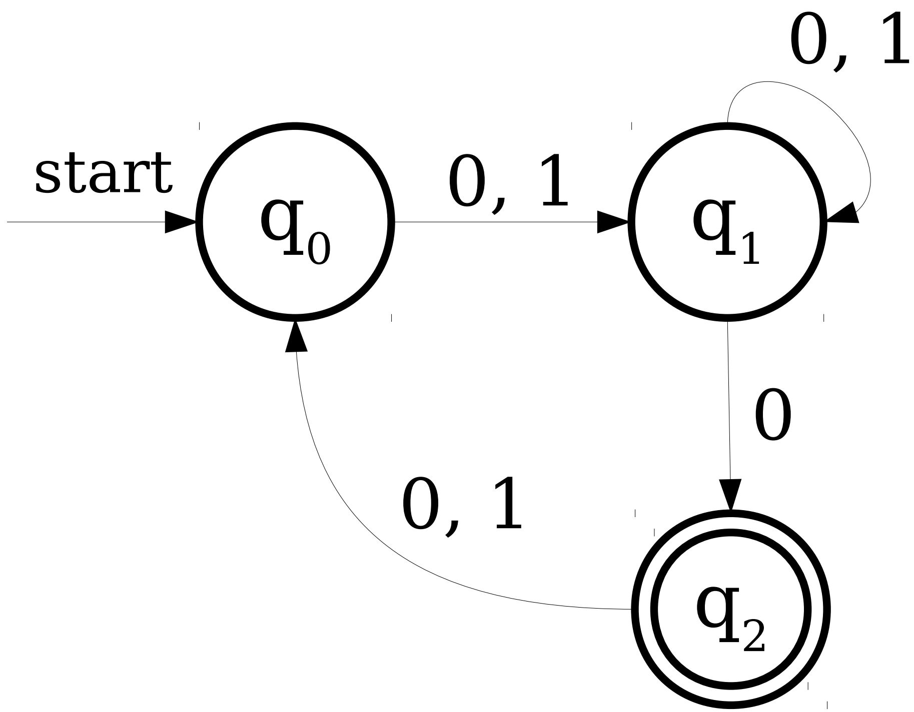
A Small Problem



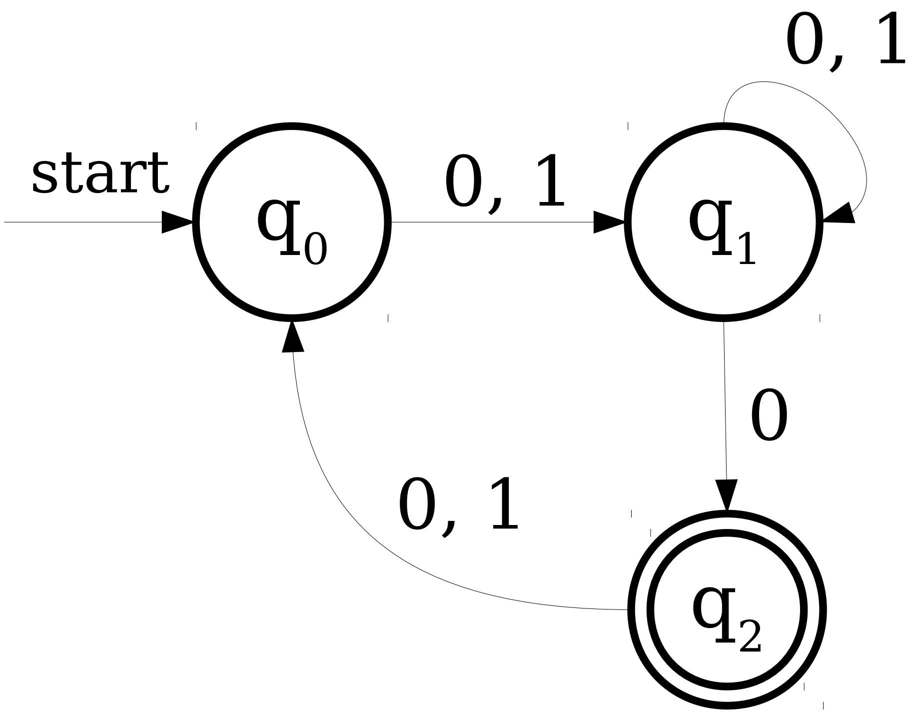
0 1 1 0

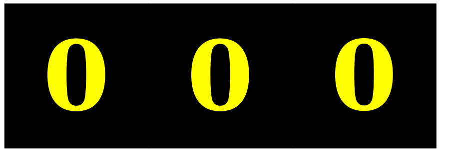


Another Small Problem

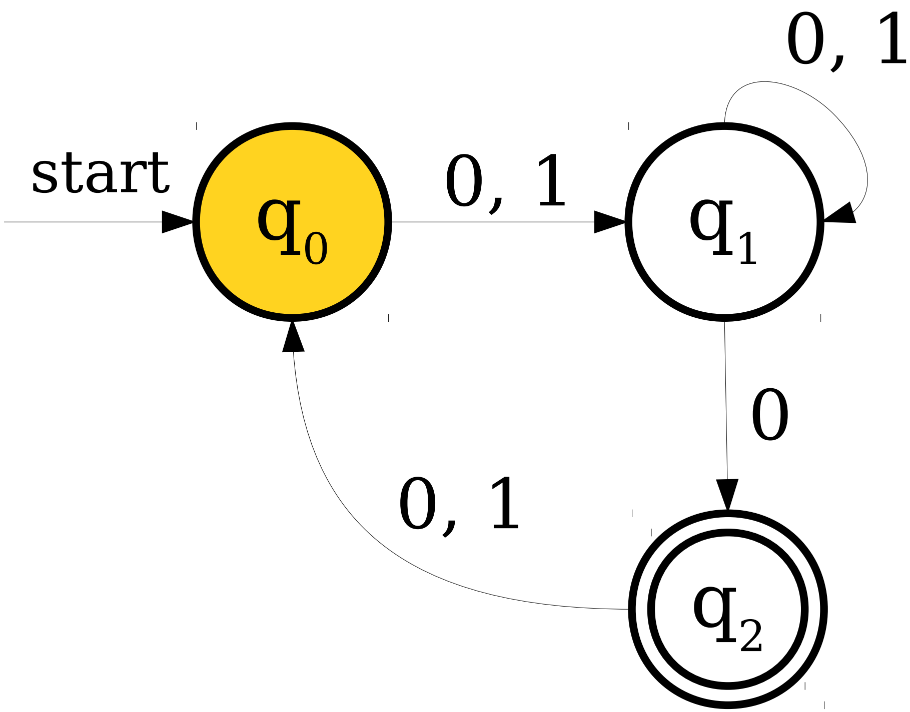


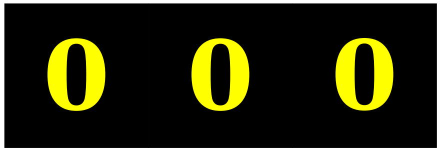
Another Small Problem



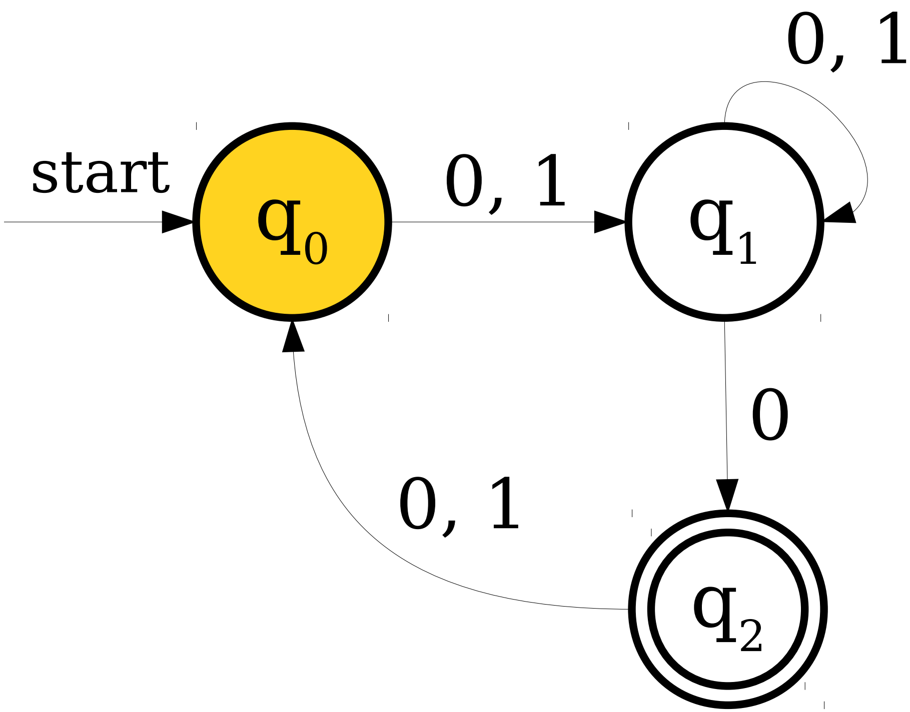


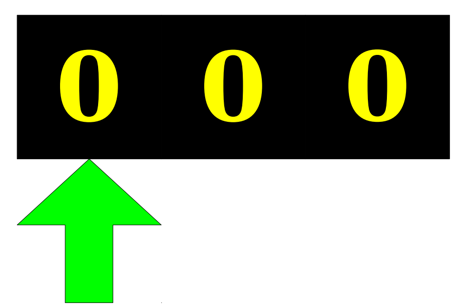
Another Small Problem



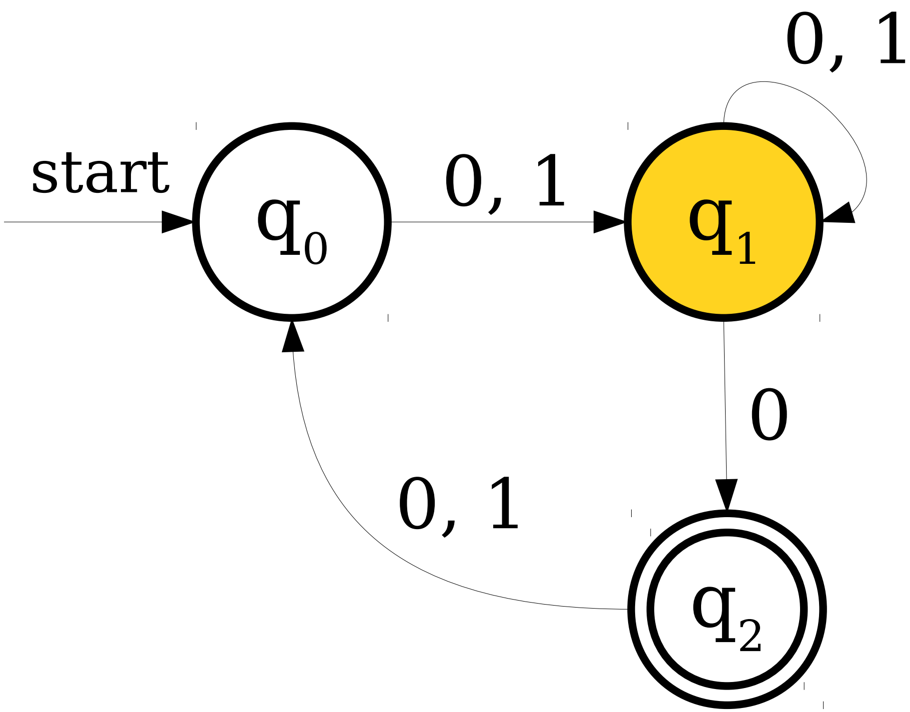


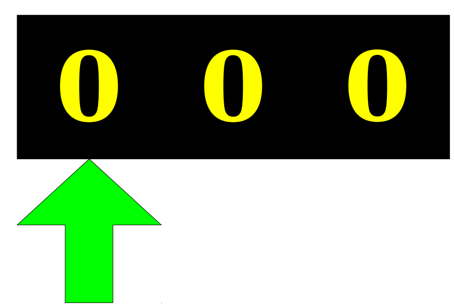
Another Small Problem



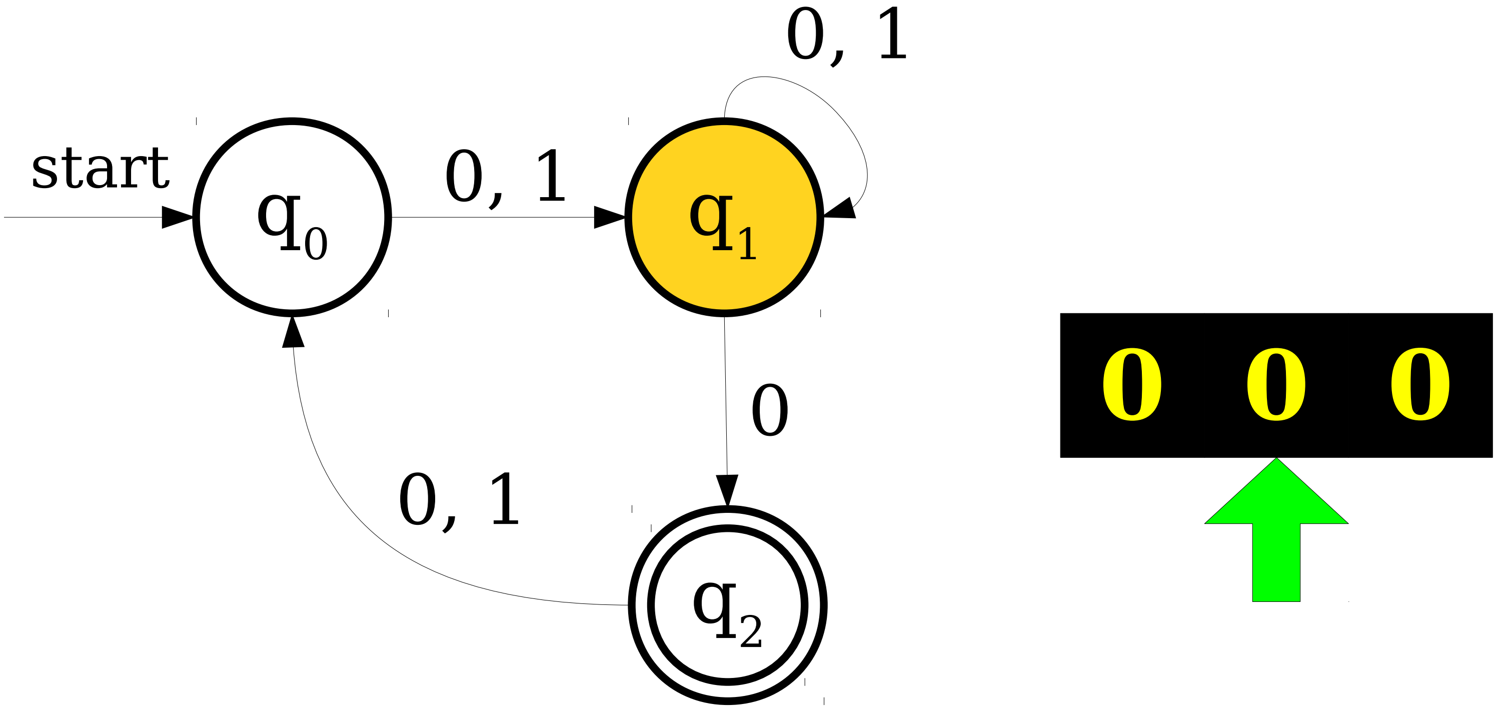


Another Small Problem

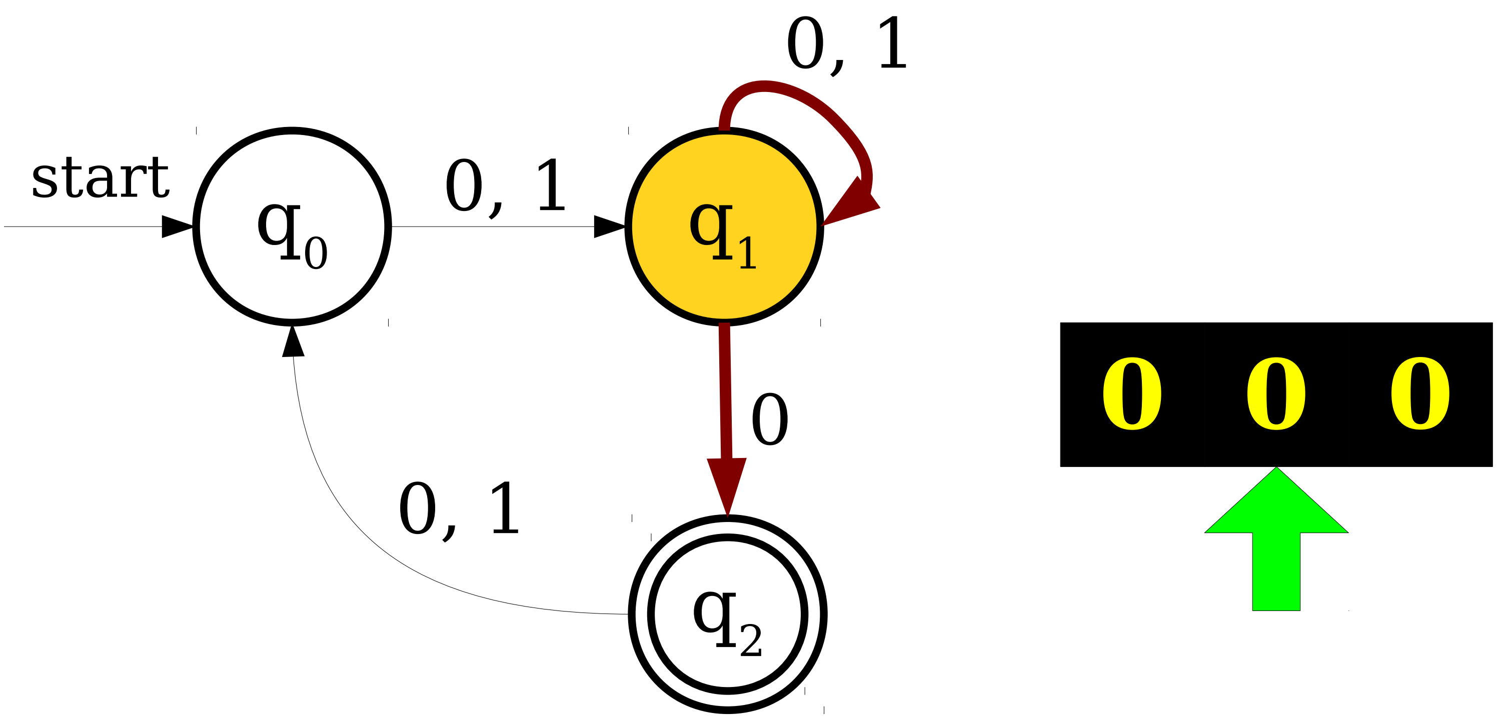




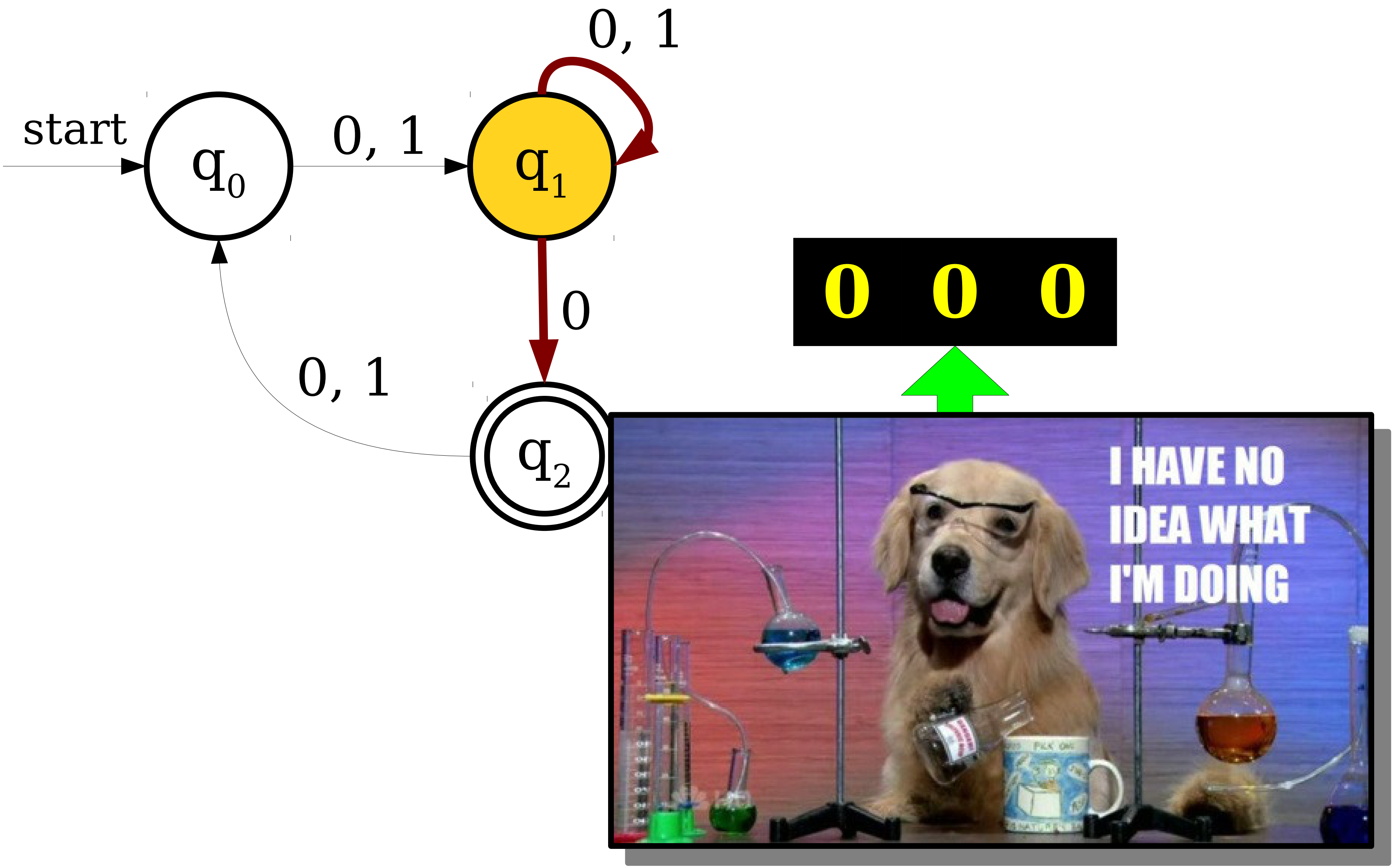
Another Small Problem



Another Small Problem



Another Small Problem



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

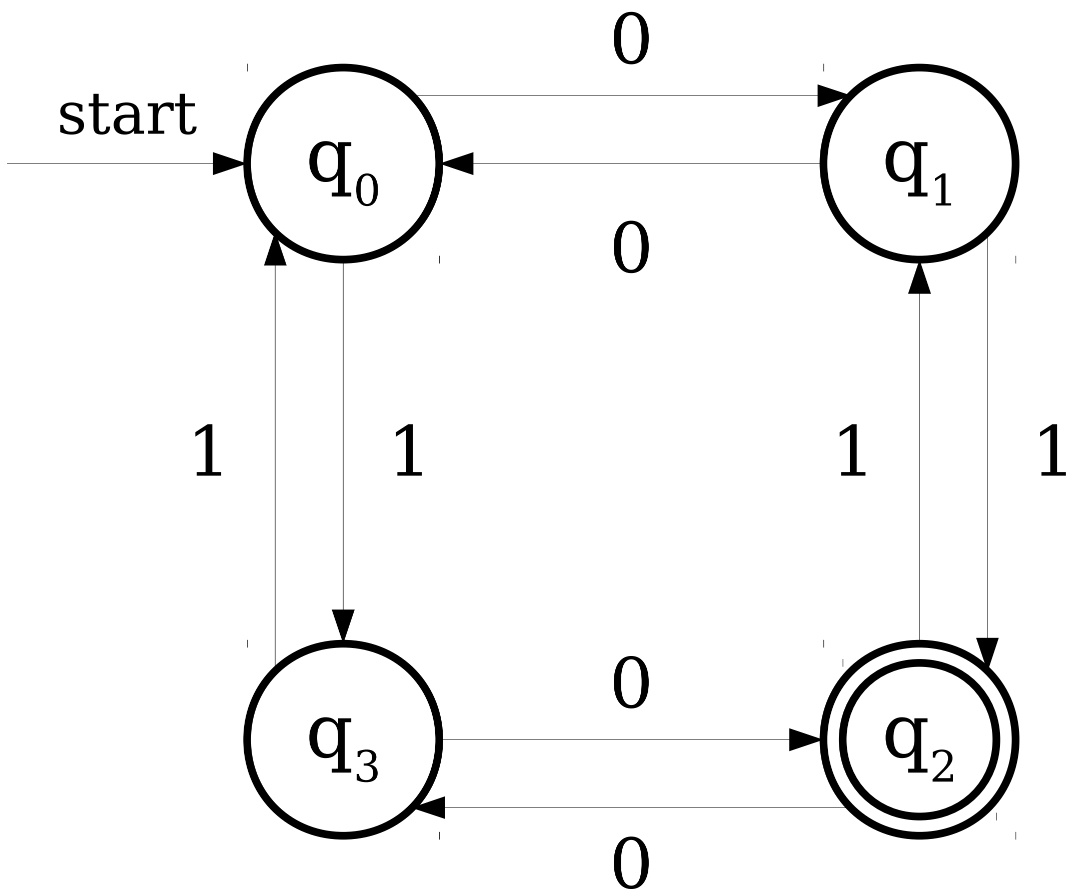
- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- DFAs are the simplest type of automaton that we will see in this course.

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.

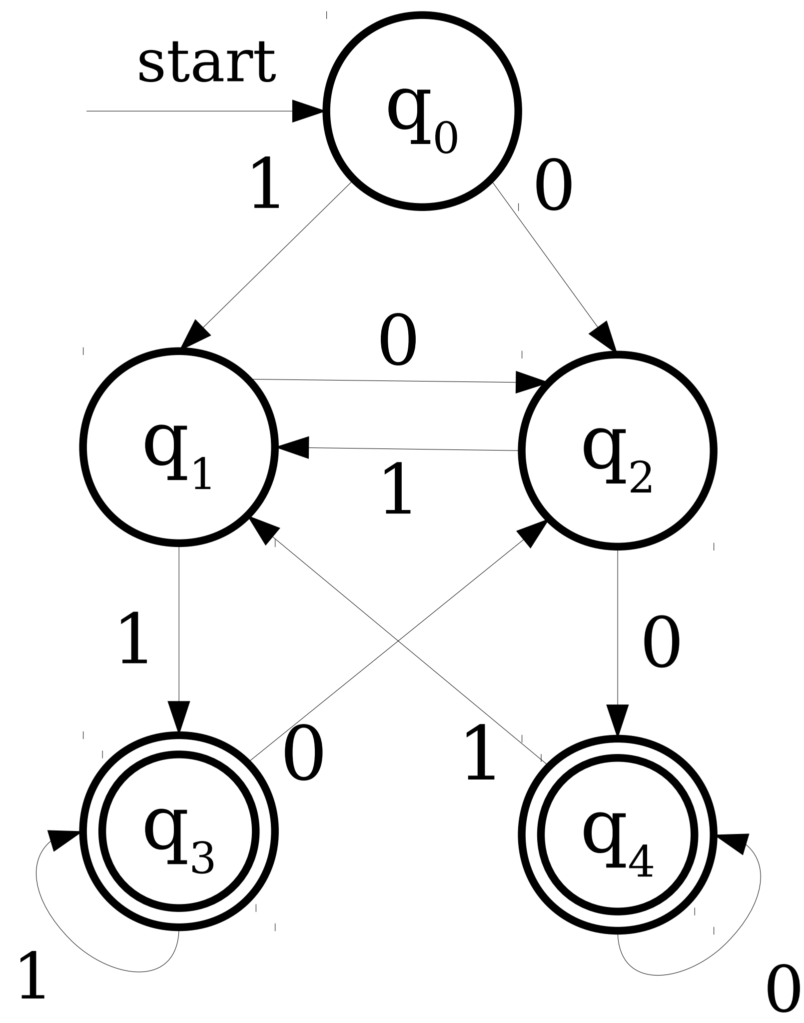
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?



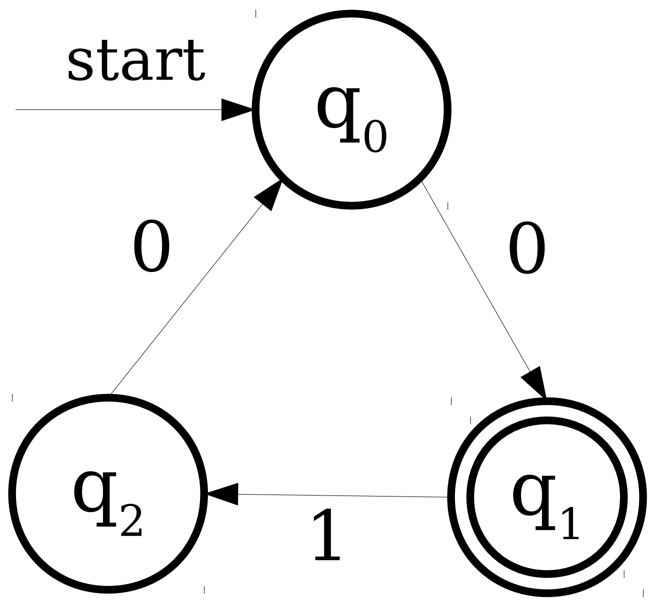
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?

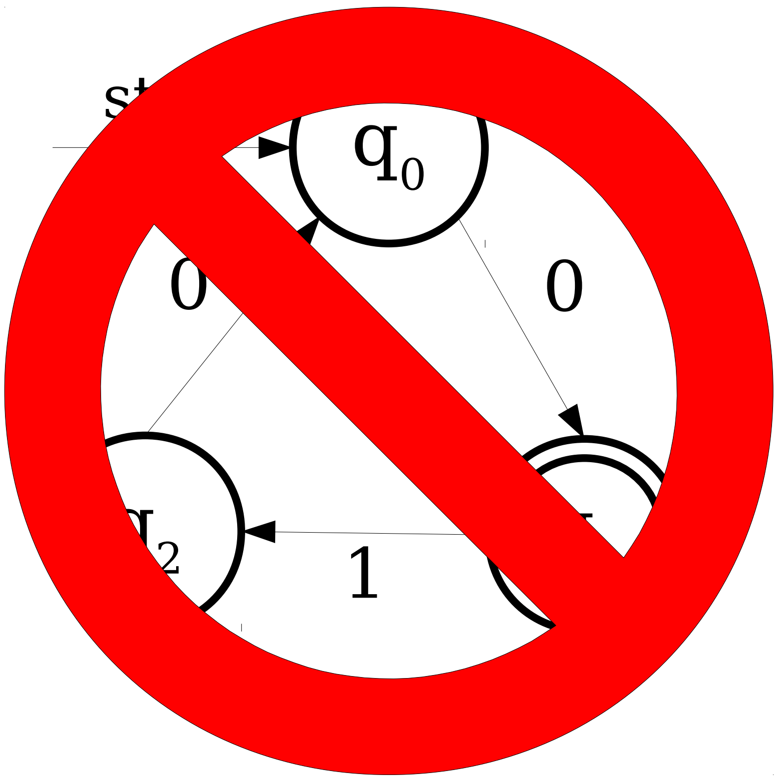


Is this a DFA over {0, 1}?

Is this a DFA over {0, 1}?

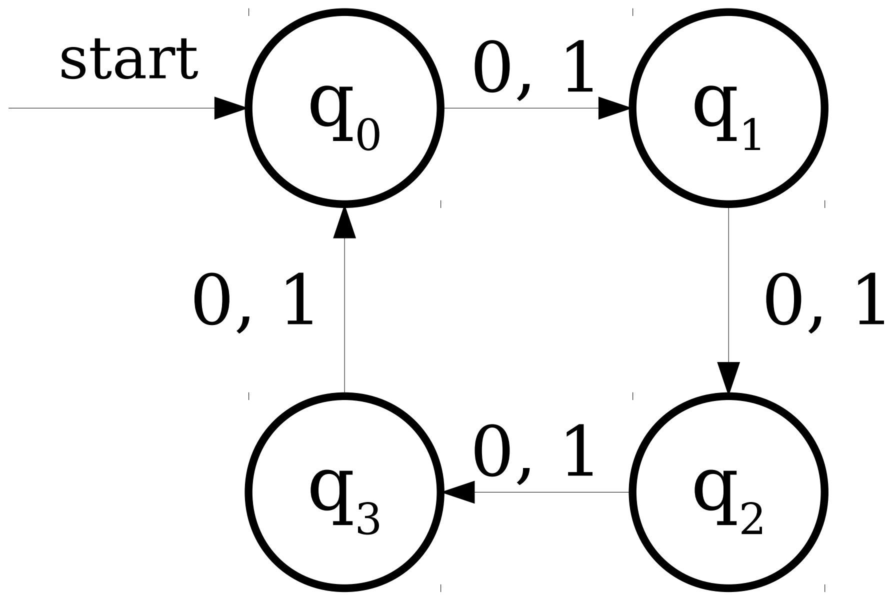


Is this a DFA over {0, 1}?



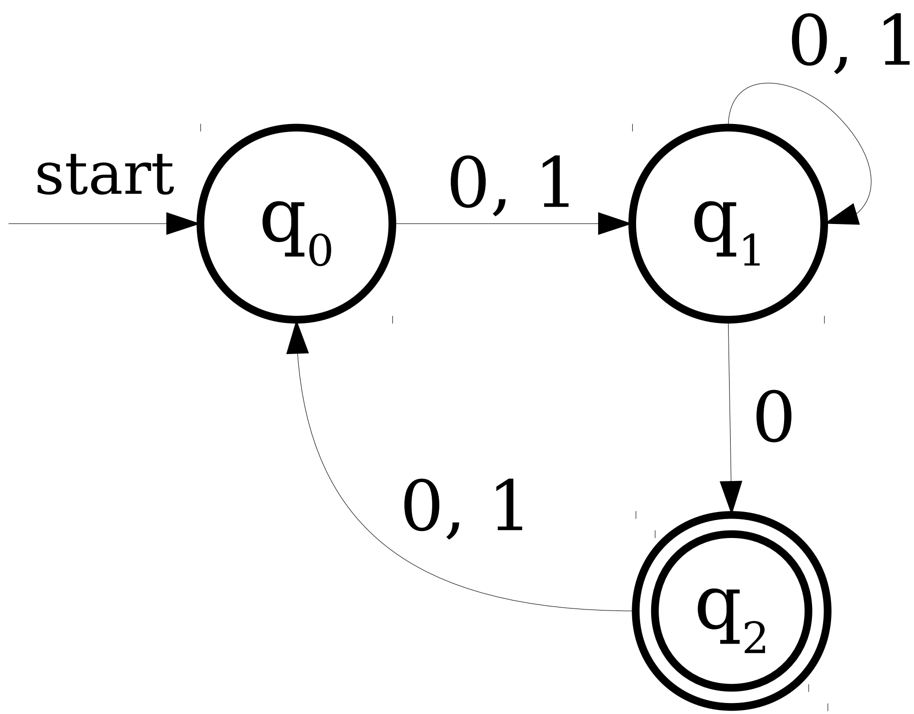
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?

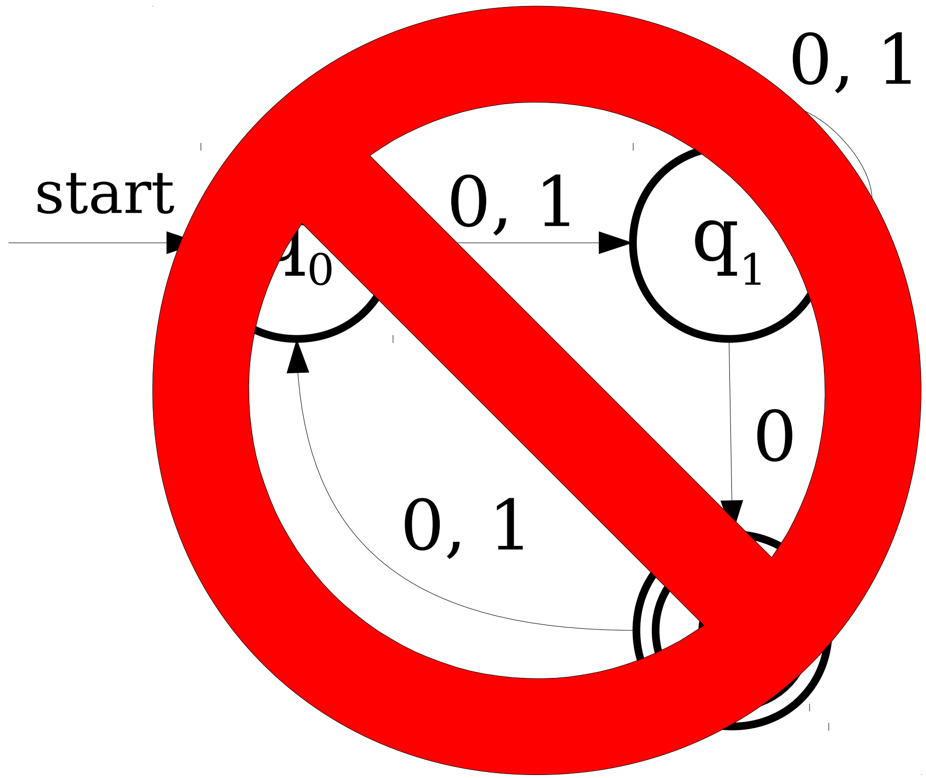


Is this a DFA over $\{0, 1\}$?

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

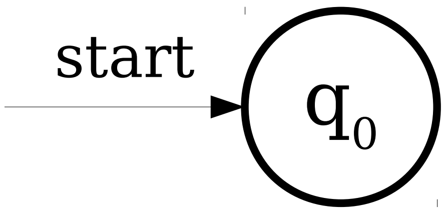
- At each point in its execution, the DFA can only remember what state it is in.
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only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

Recognizing Languages with DFAs

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

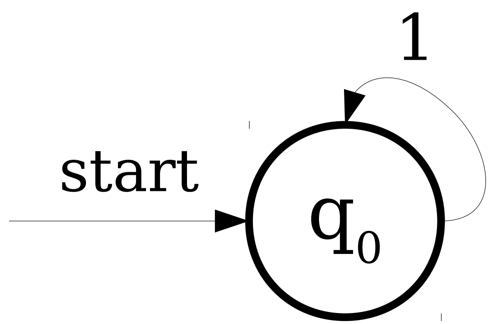
Recognizing Languages with DFAs

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



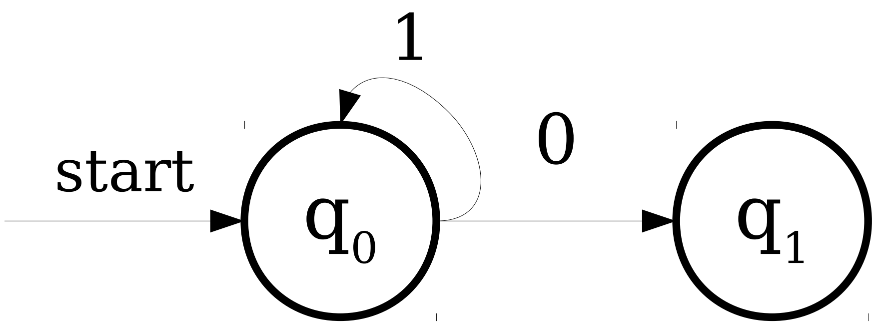
Recognizing Languages with DFAs

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



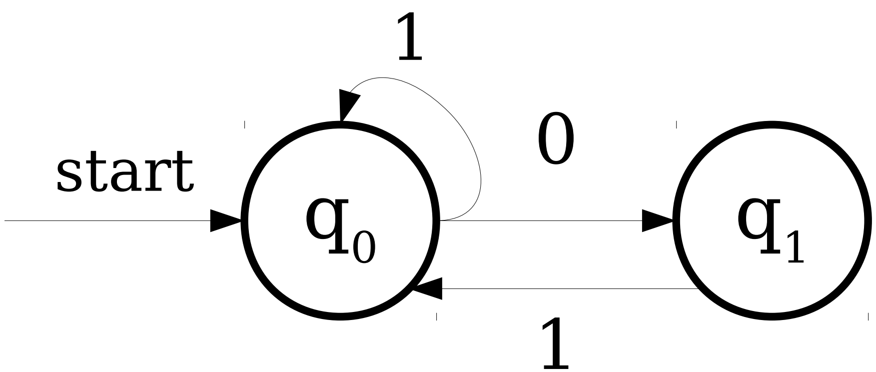
Recognizing Languages with DFAs

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



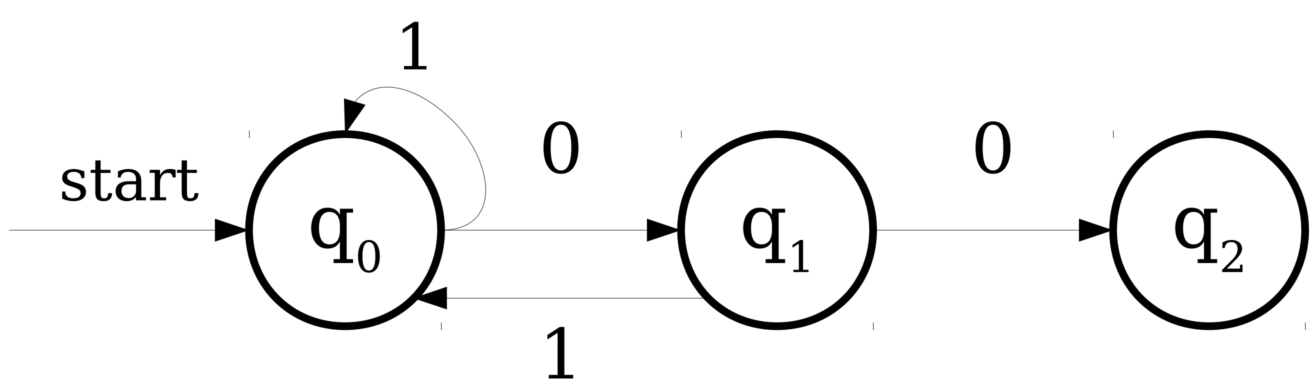
Recognizing Languages with DFAs

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



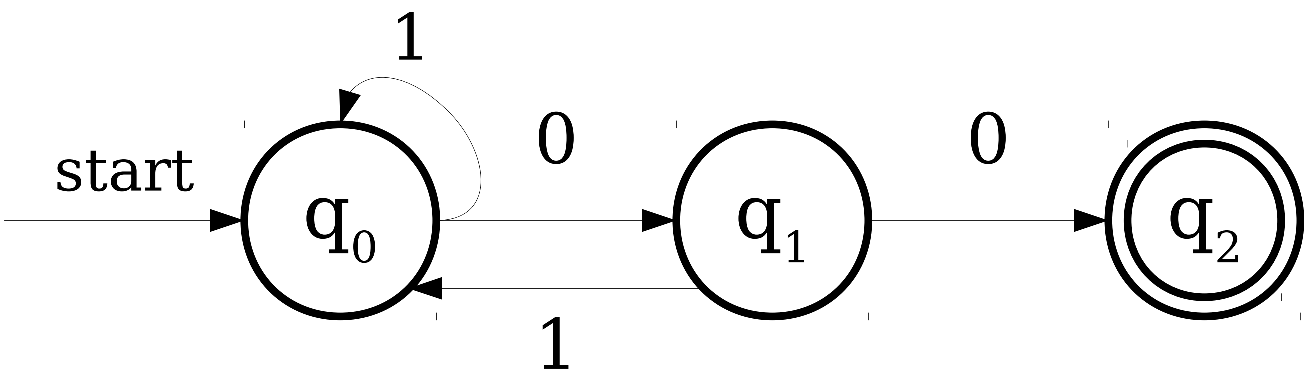
Recognizing Languages with DFAs

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



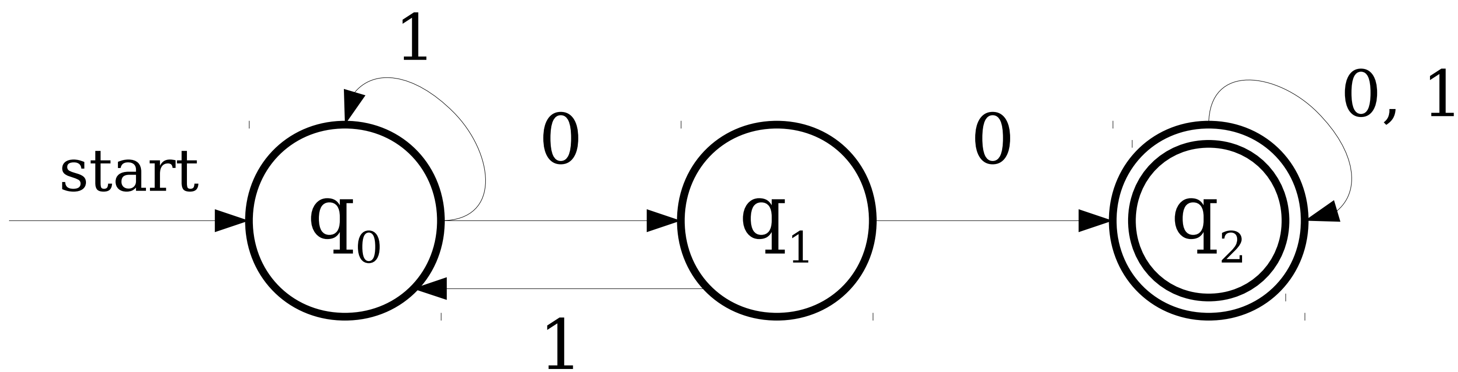
Recognizing Languages with DFAs

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



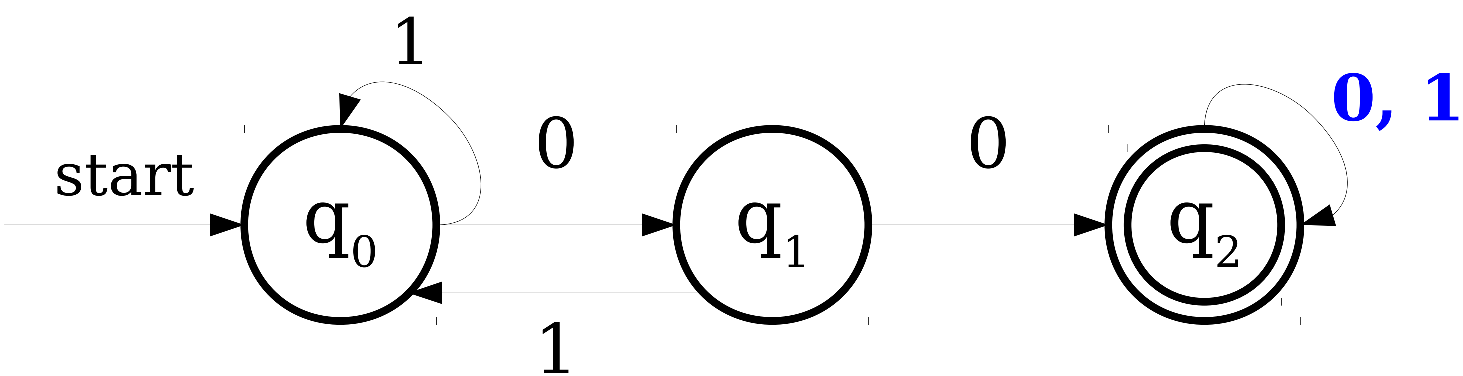
Recognizing Languages with DFAs

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



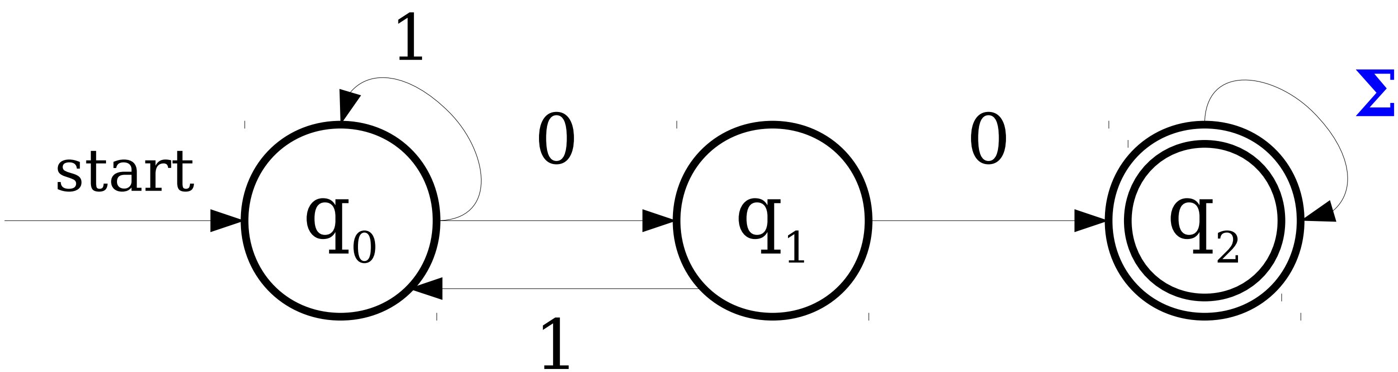
Recognizing Languages with DFAs

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



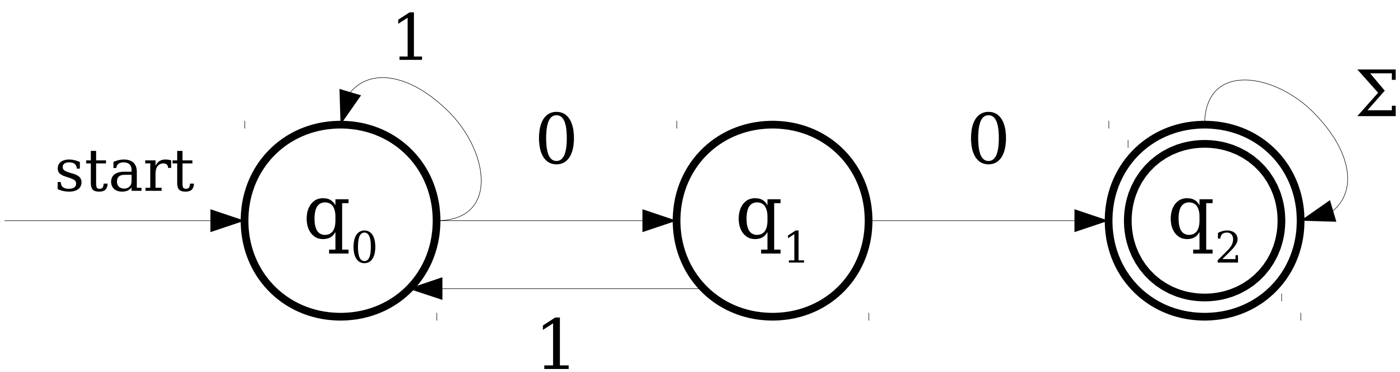
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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

Recognizing Languages with DFAs

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

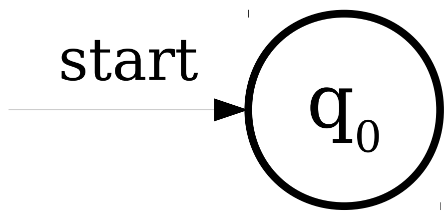
YES	**NO**
01	**1**
00**0**1	**0**0**1**
01**0**1**0**1**0**0**0**1	**0**0**0**0**1**

Recognizing Languages with DFAs

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

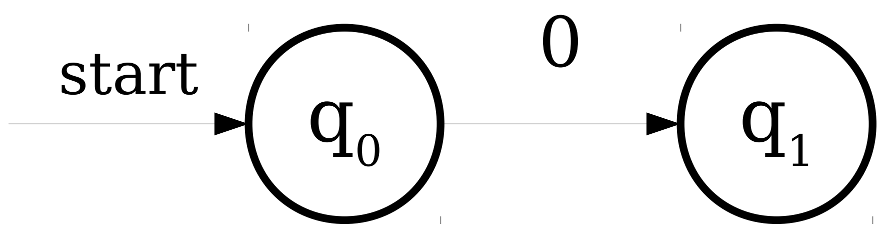
Recognizing Languages with DFAs

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



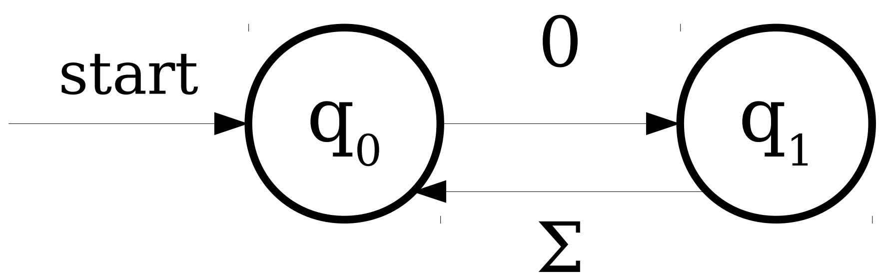
Recognizing Languages with DFAs

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



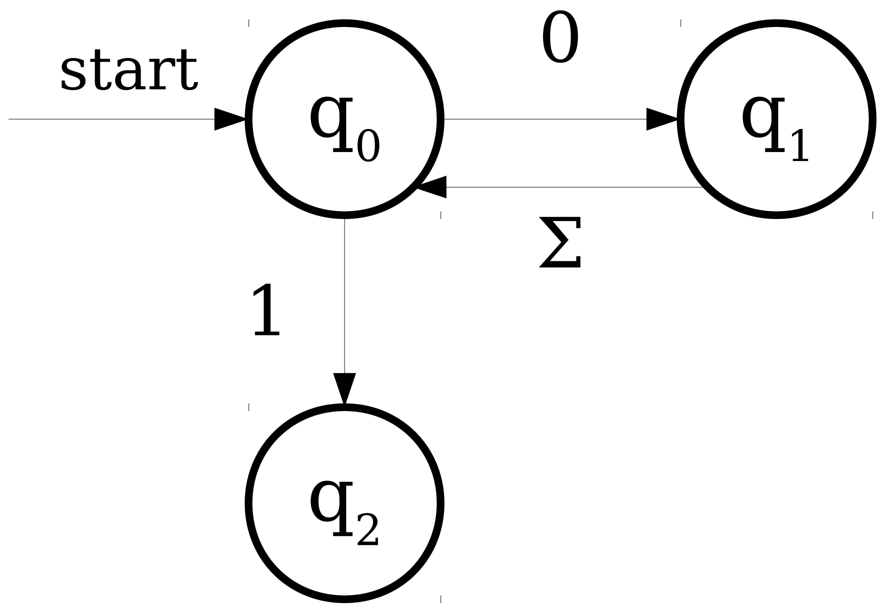
Recognizing Languages with DFAs

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



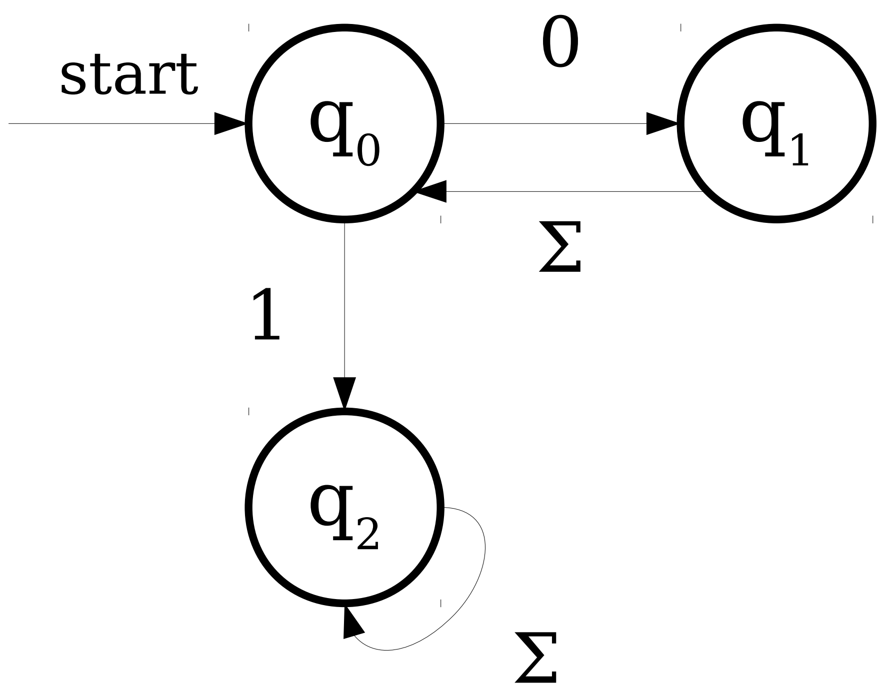
Recognizing Languages with DFAs

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



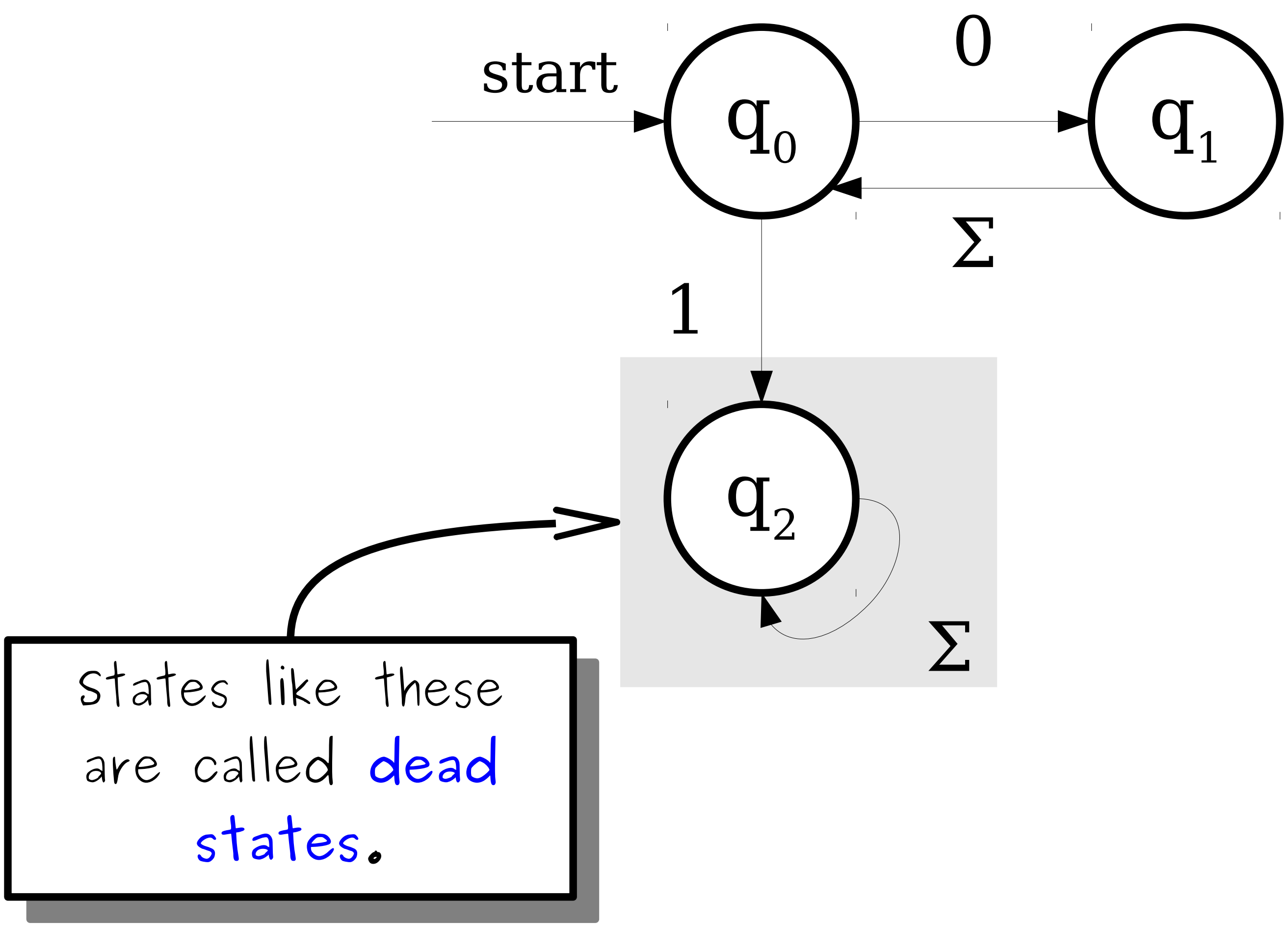
Recognizing Languages with DFAs

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



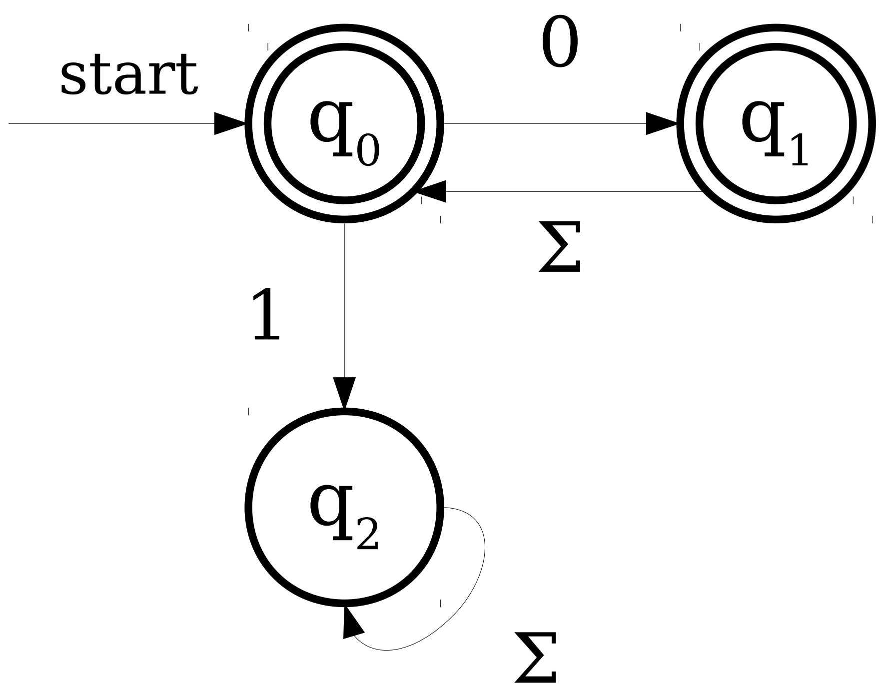
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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



Next Time

- **Regular Languages**
  - What is the expressive power of DFAs?
- **NFAs**
  - Automata with Magic Superpowers!
- **Nondeterminism**
  - Nondeterminisic computation.
  - Intuitions for nondeterminism.
  - Programming with nondeterminism.