Finite Automata

Part One

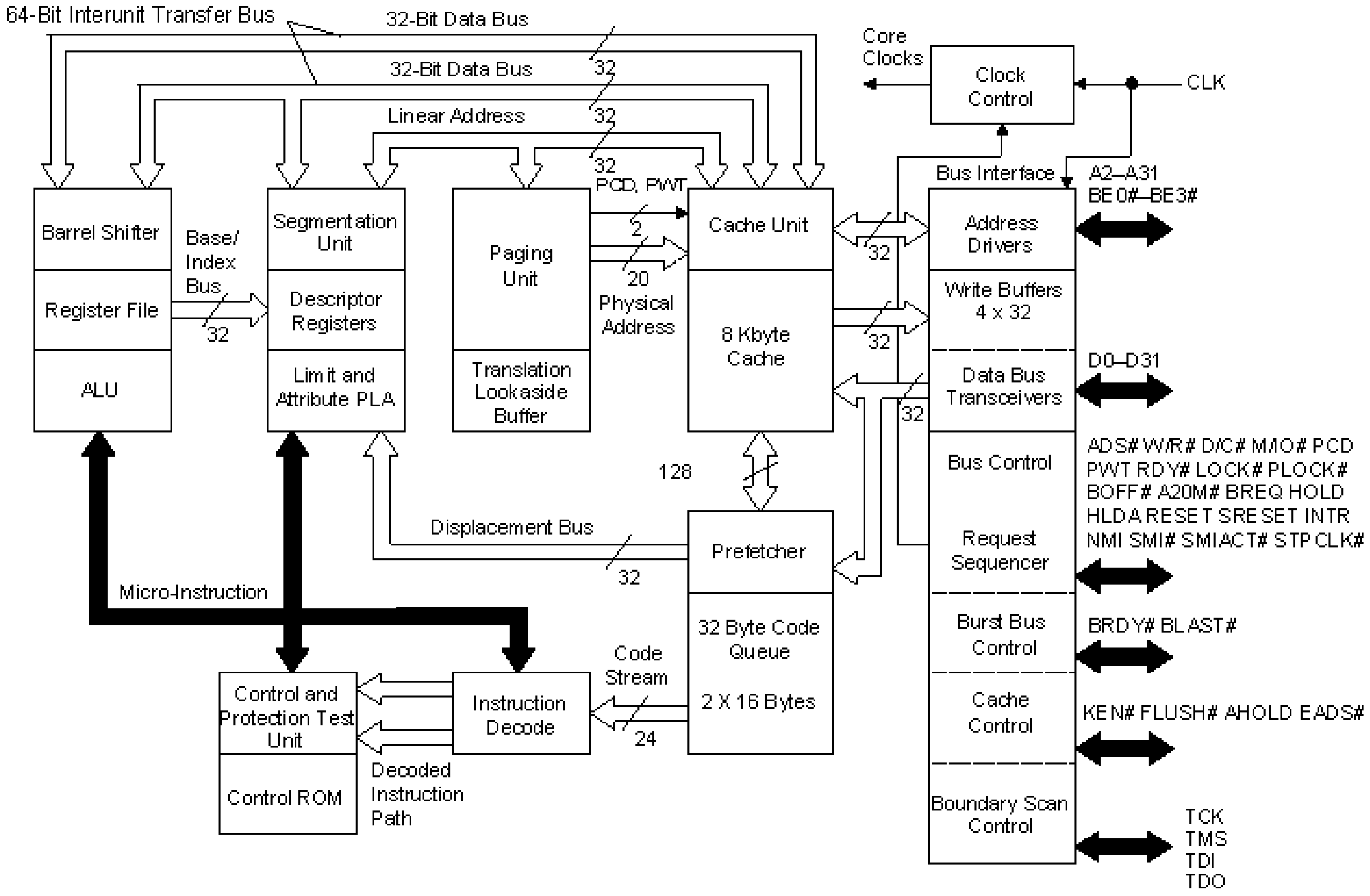
Computability Theory

What problems can we solve with a computer?

What problems can we solve with a computer?

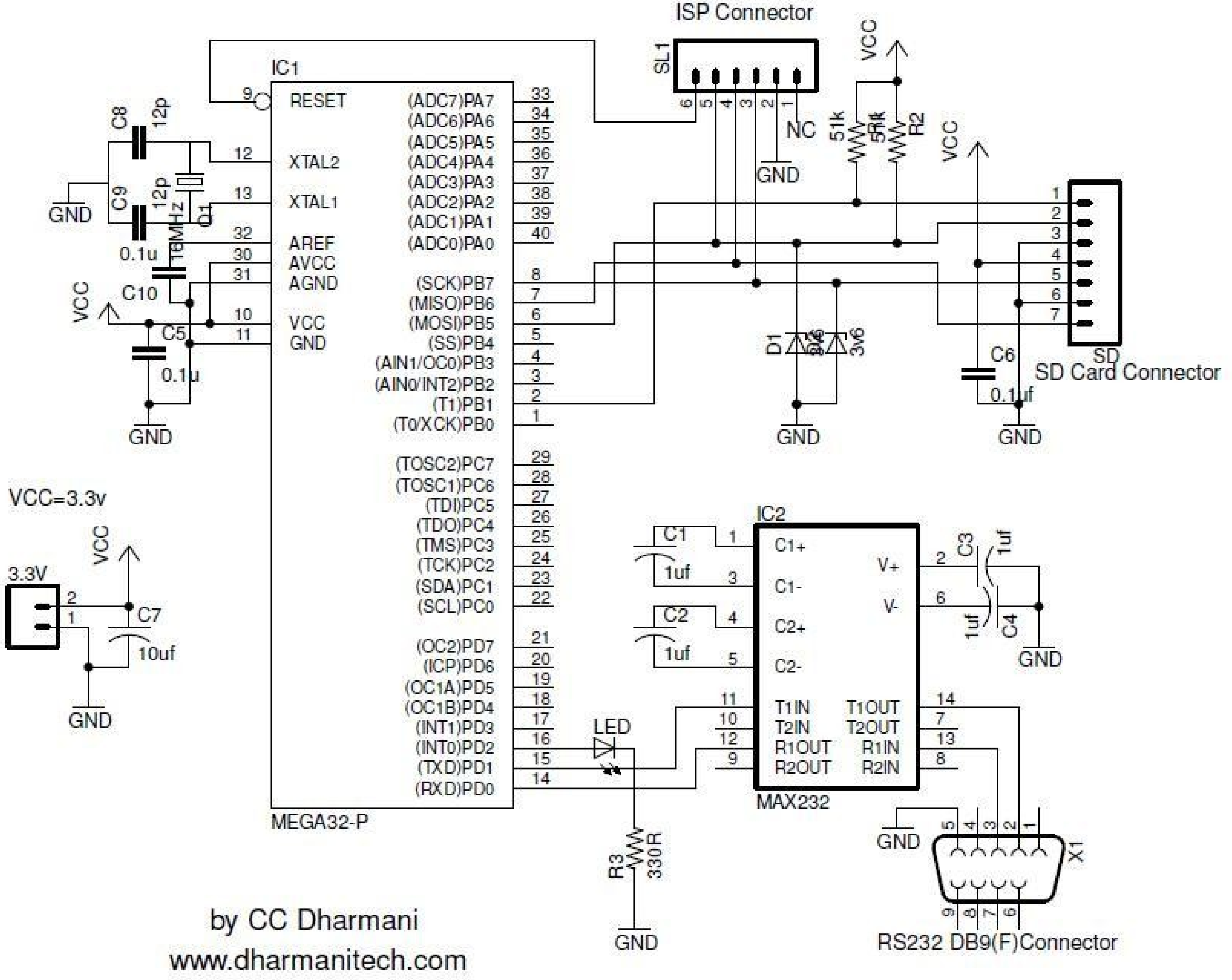
What kind of computer?

Computers are Messy



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

Computers are Messy



by CC Dharmani
www.dharmanitech.com

microSD/SD Card interface with ATmega32 Ver_2.3

http://www.dharmanitech.com/

Computers are Messy

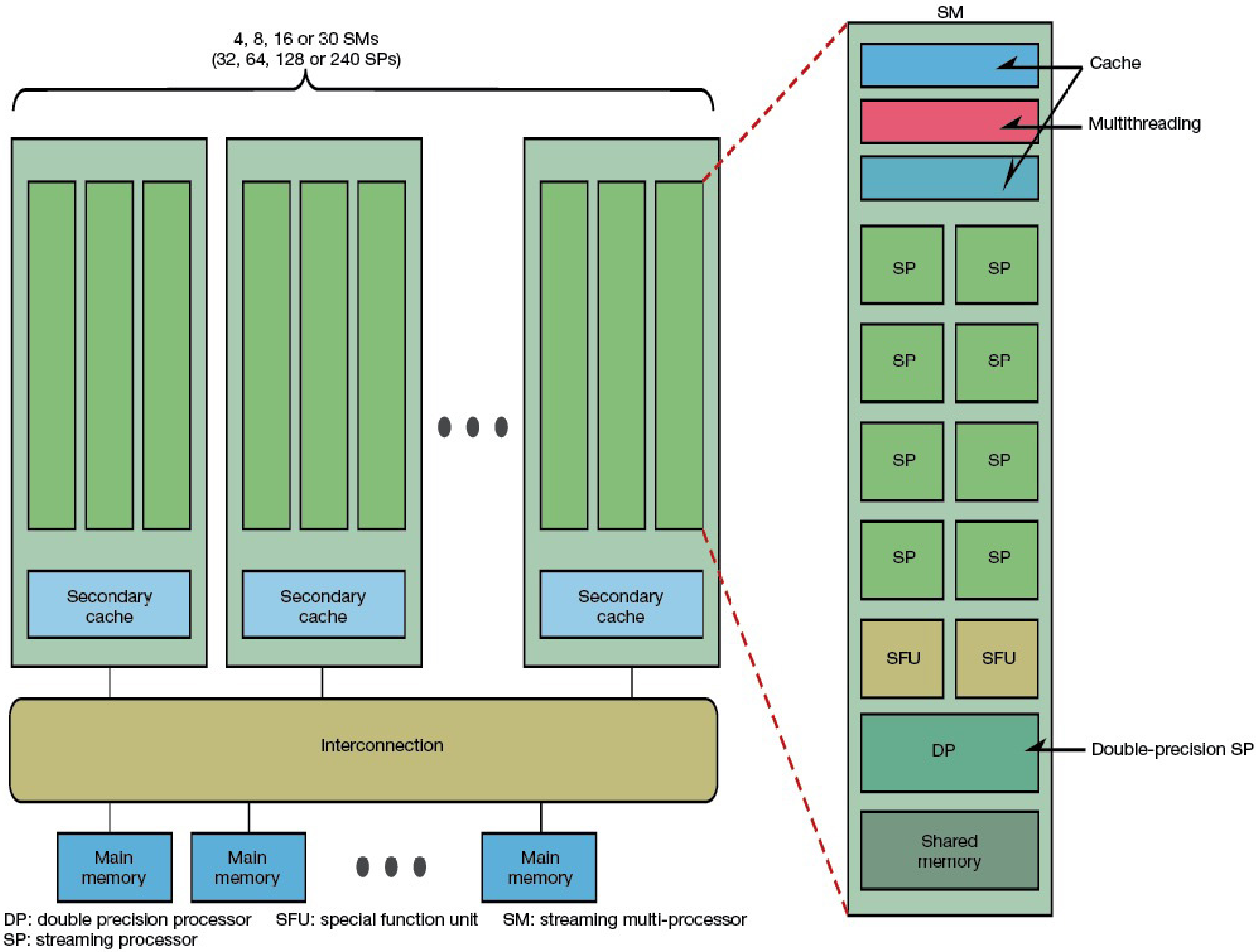
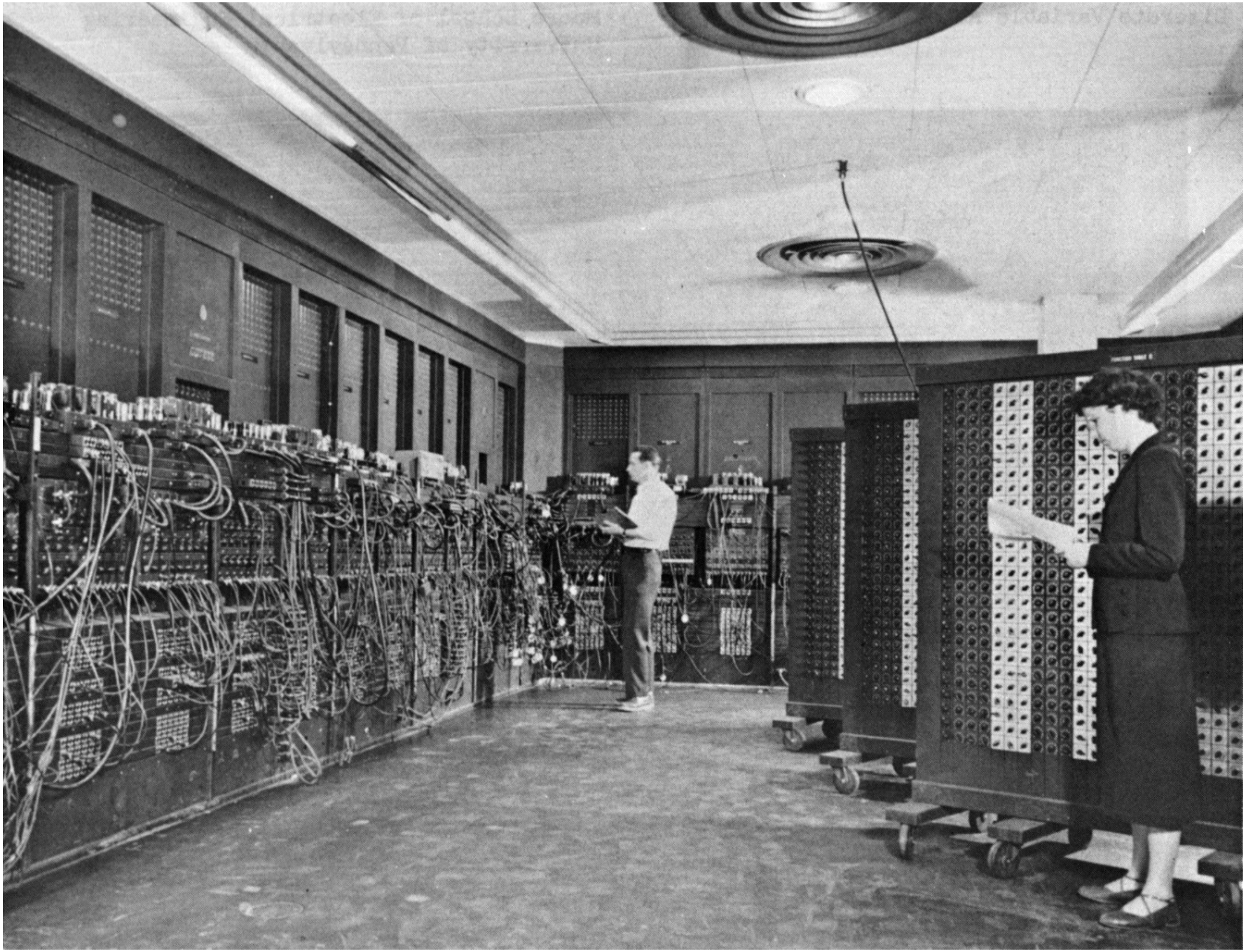


Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM)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

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

Computers are Messy

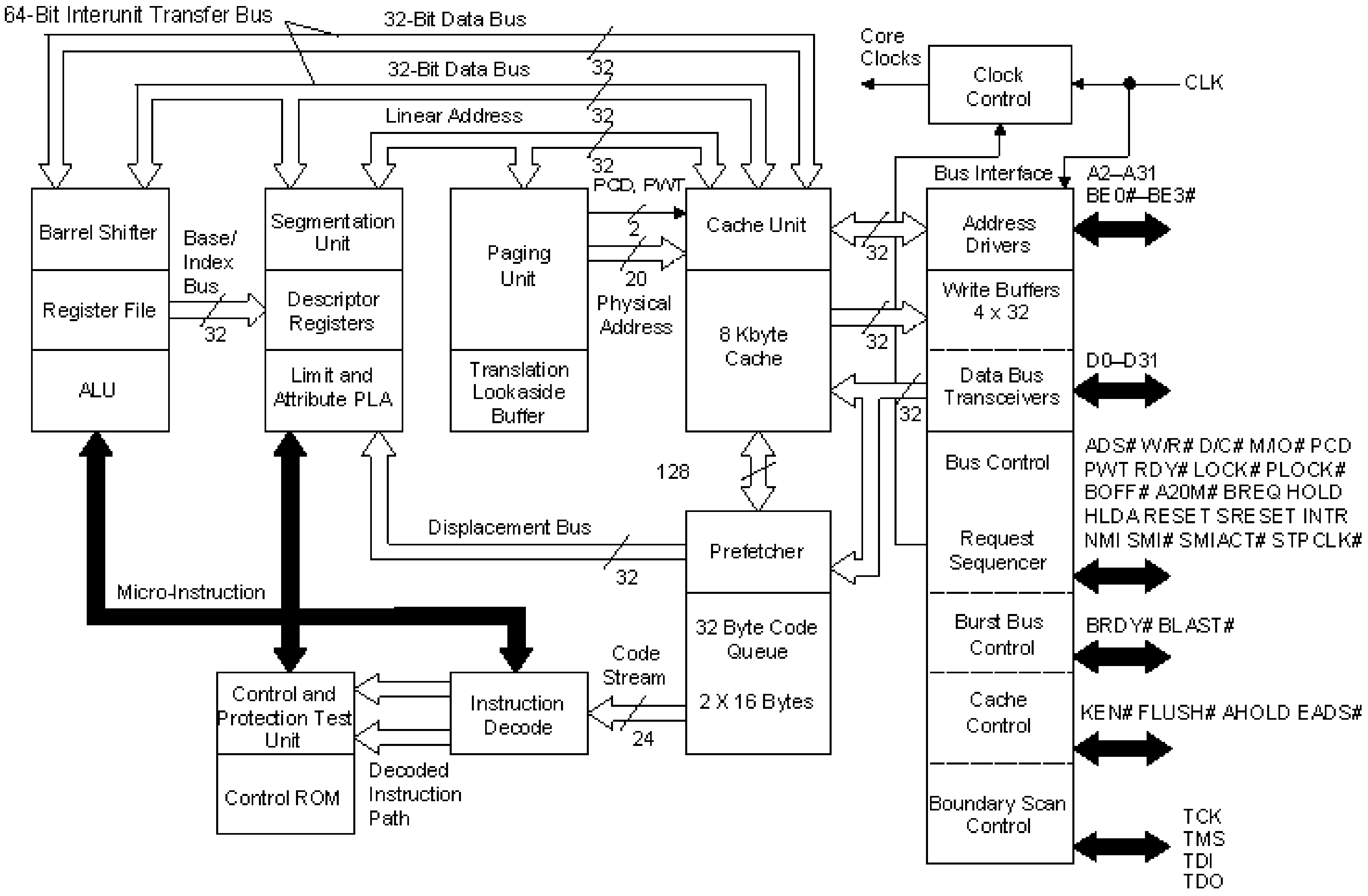


http://en.wikipedia.org/wiki/File:Eniac.jpg

We need a simpler way of discussing computing machines.

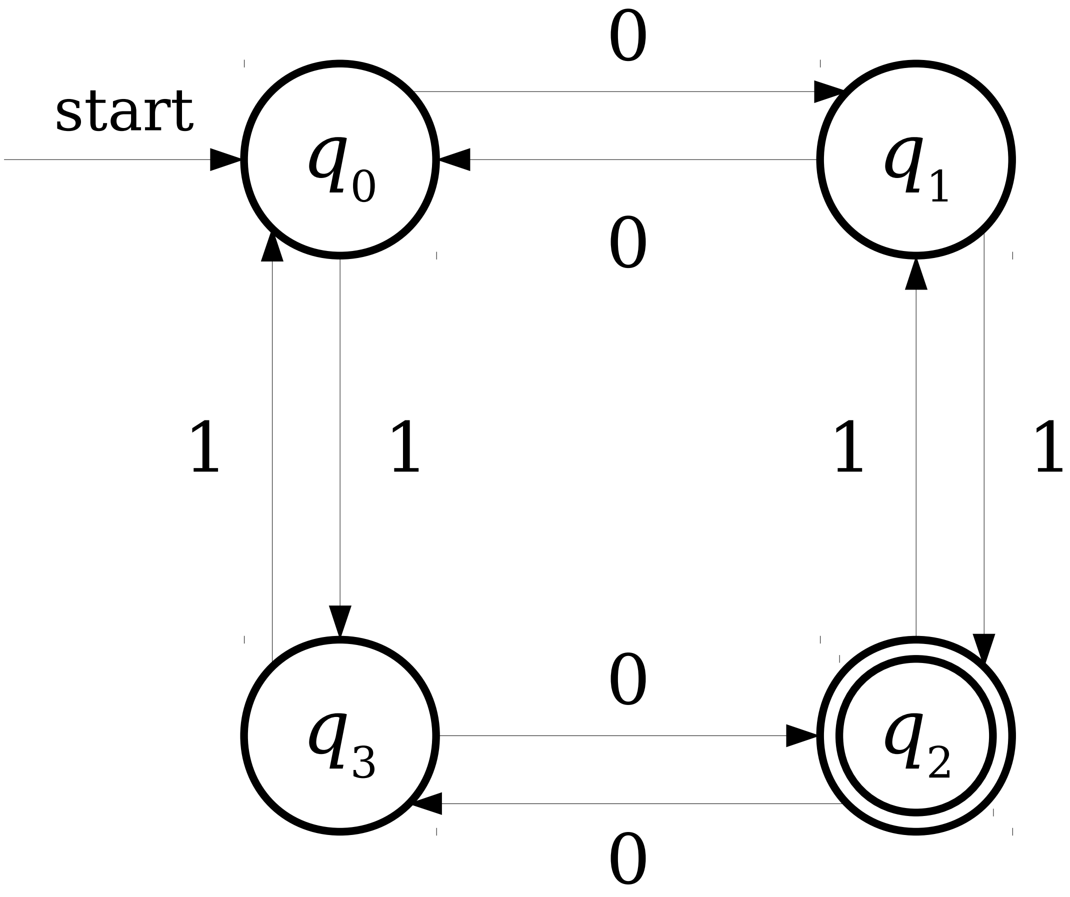
An ***automaton*** (plural: ***automata***) is a mathematical model of a computing device.

Computers are Messy

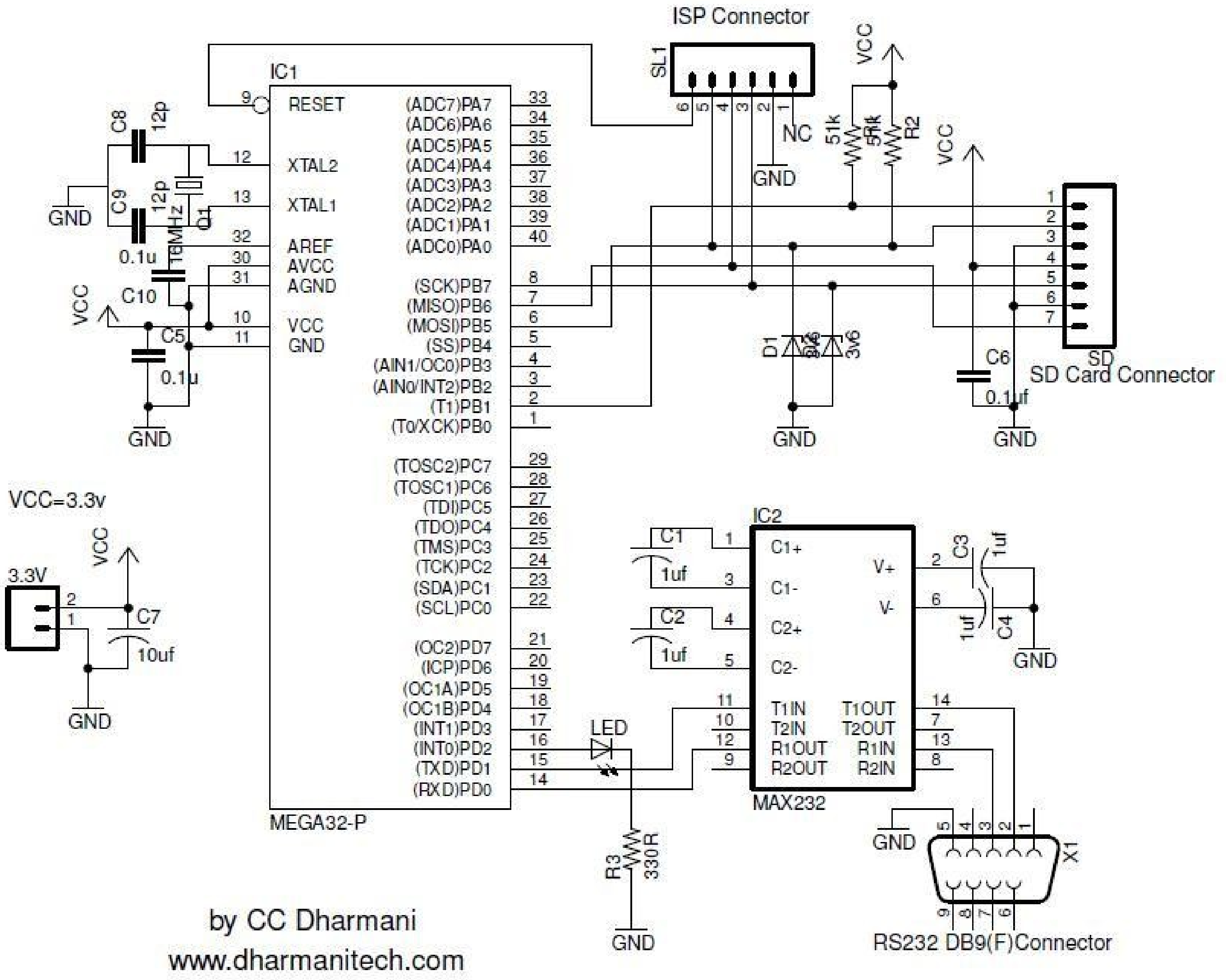


http://www.intel.com/design/intarch/prodbref/272713.htm

Automata are Clean



Computers are Messy

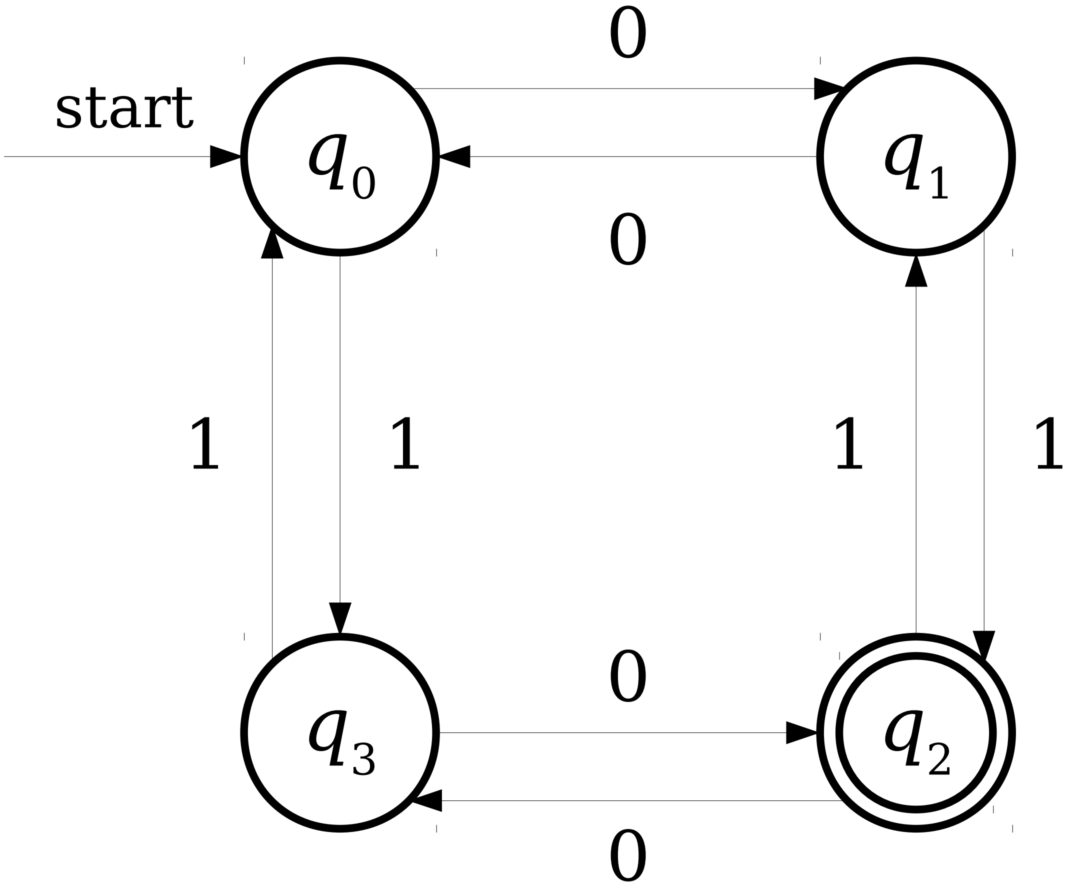


by CC Dharmani
www.dharmanitech.com

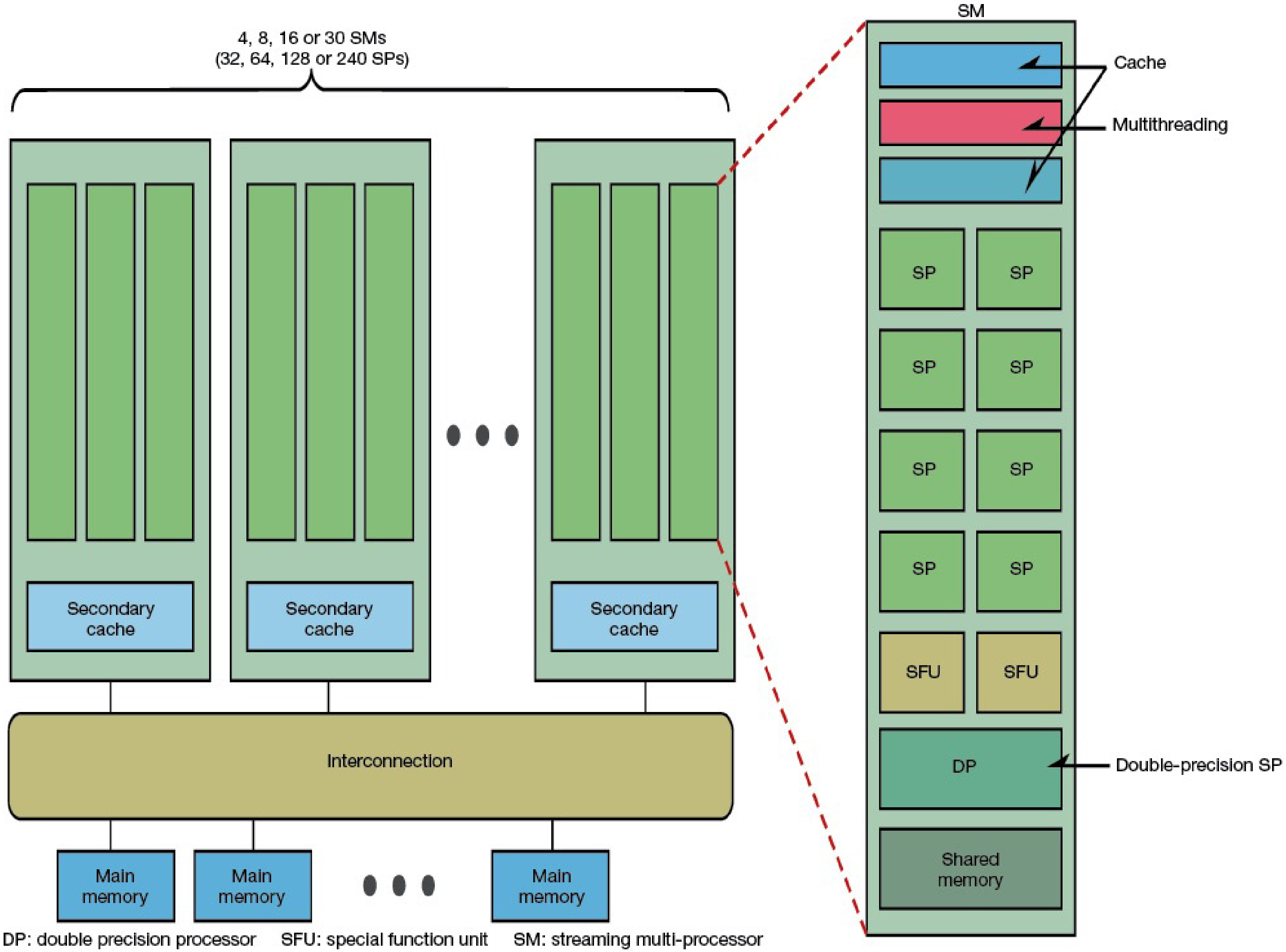
microSD/SD Card interface with ATmega32 Ver_2.3

http://www.dharmanitech.com/

Automata are Clean



Computers are Messy

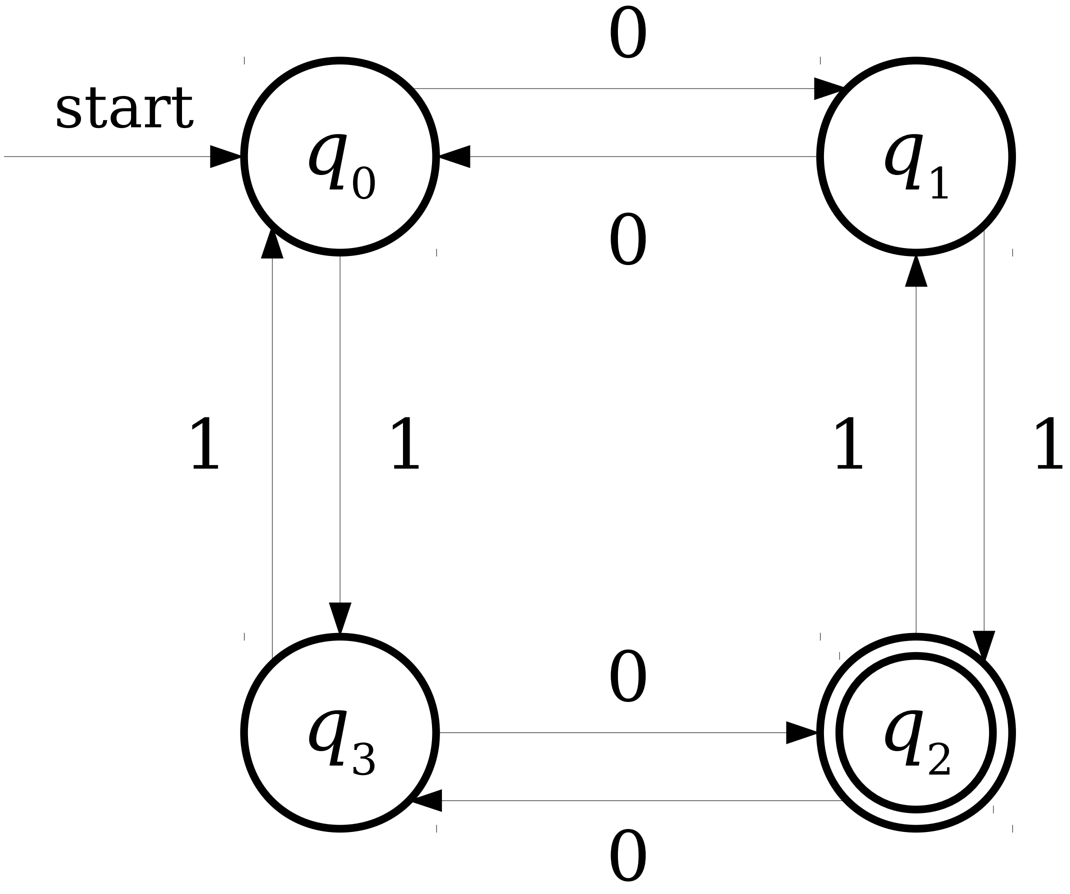


DP: double precision processor SFU: special function unit SM: streaming multi-processor
SP: streaming processor

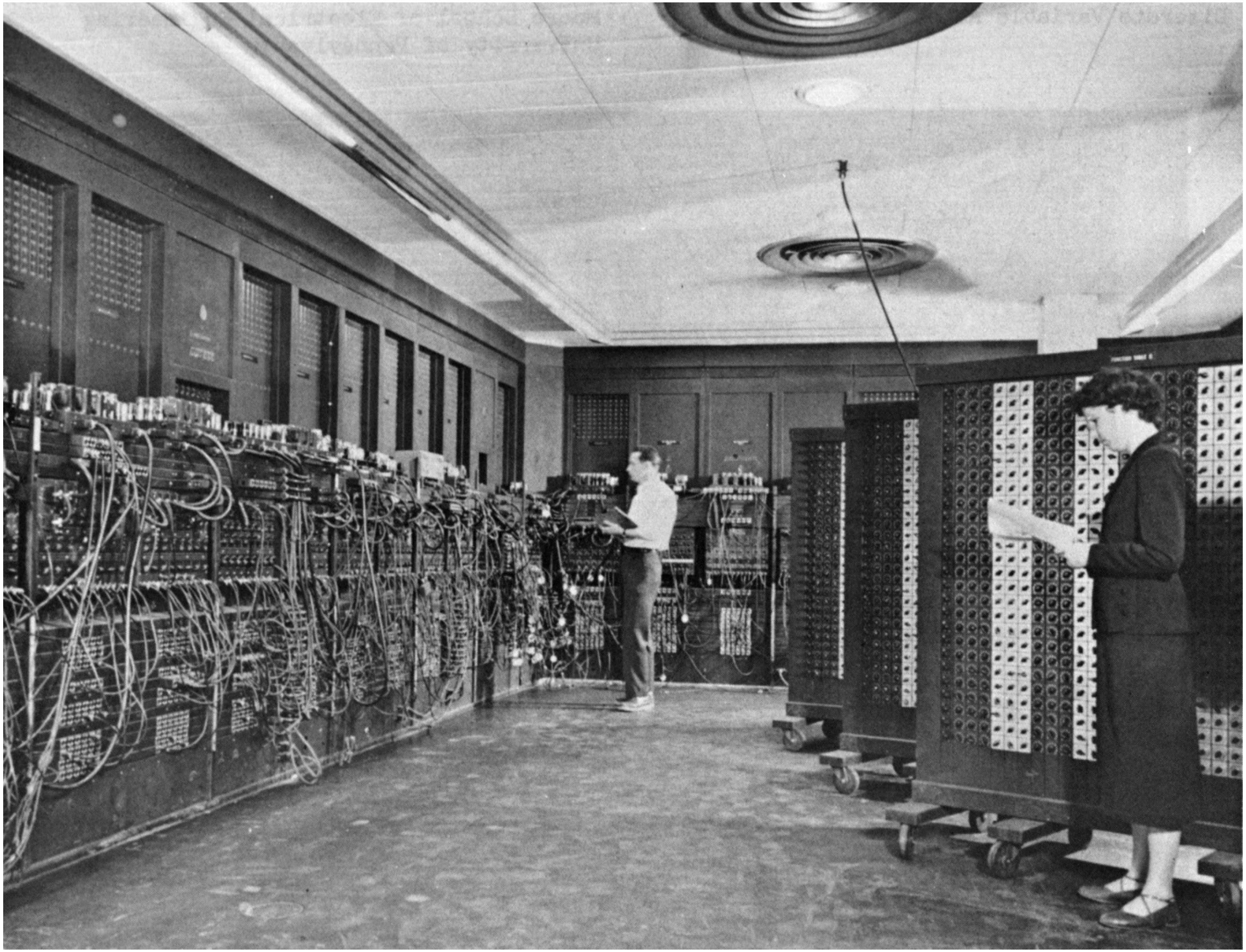
Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM)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

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

Automata are Clean

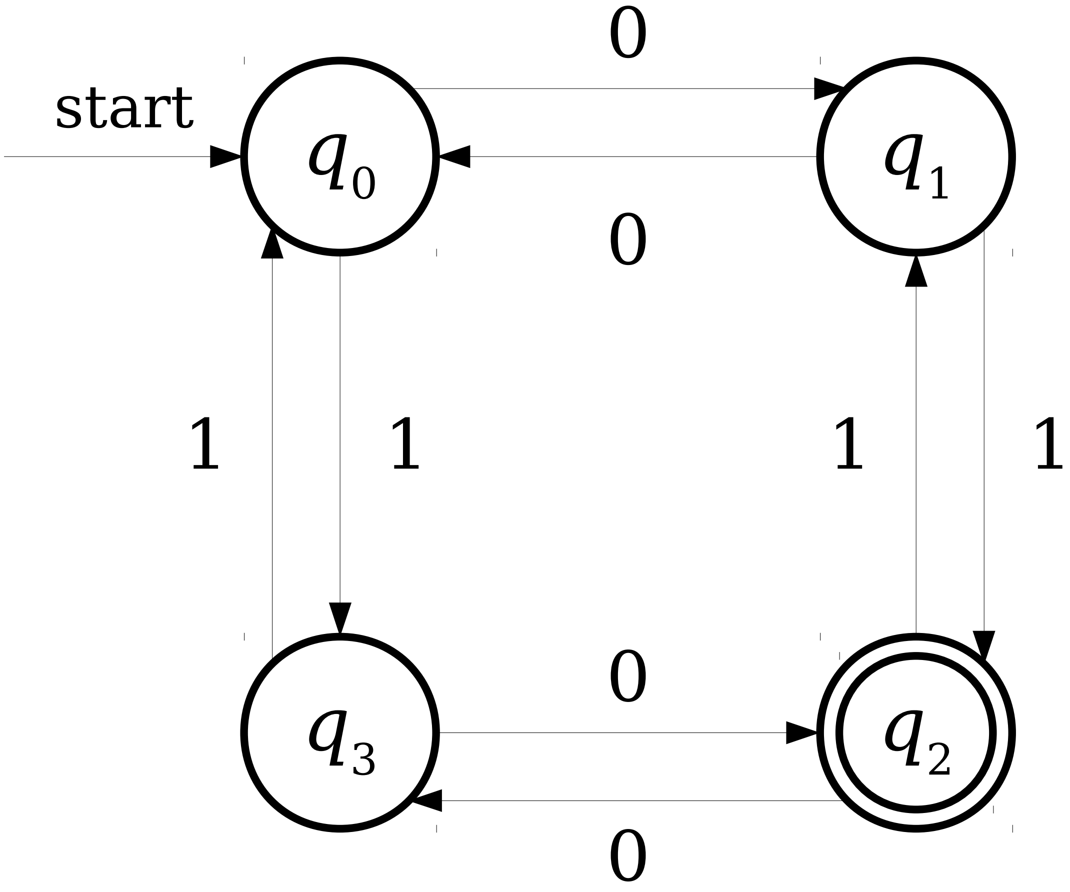


Computers are Messy



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 $L \subseteq \Sigma^*$.

To Recap

- ***Languages*** are sets of strings, and
- ***strings*** are sequences of characters, and
- ***characters*** are elements of an alphabet.

The Model

- ***Fundamental Question:*** For what languages L can you design an automaton that takes as input a string, then decides whether the string is in L?
- The answer depends on the choice of L, the choice of automaton, and the definition of "determines."
- In answering this question, we'll go through a whirlwind tour of models of computation and see how this seemingly abstract question has very real and powerful consequences.

To Summarize

- An ***automaton*** is an idealized mathematical computing machine.
- A ***language*** is a set of strings, a ***string*** is a (finite) sequence of characters, and a ***character*** is an element of an alphabet.
- ***Goal:*** Figure out in which cases we can build automata for particular languages.

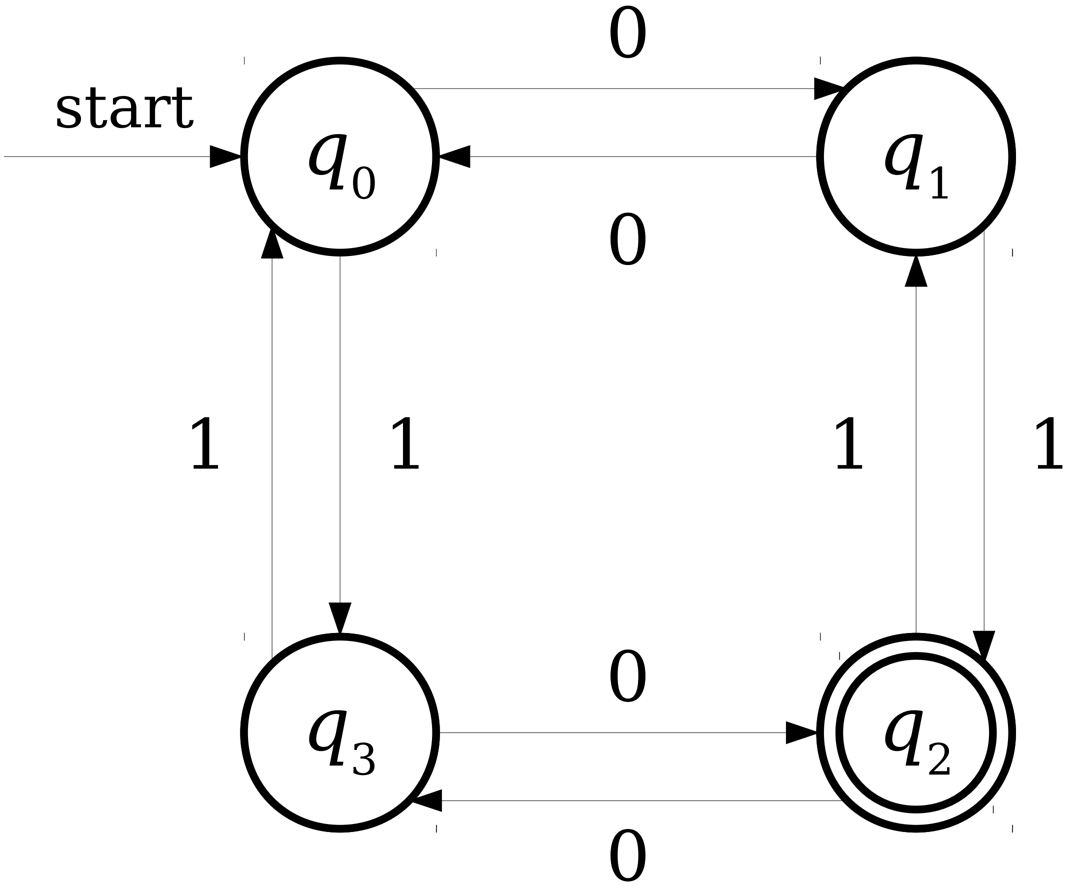
What problems can we solve with a computer?

Finite Automata

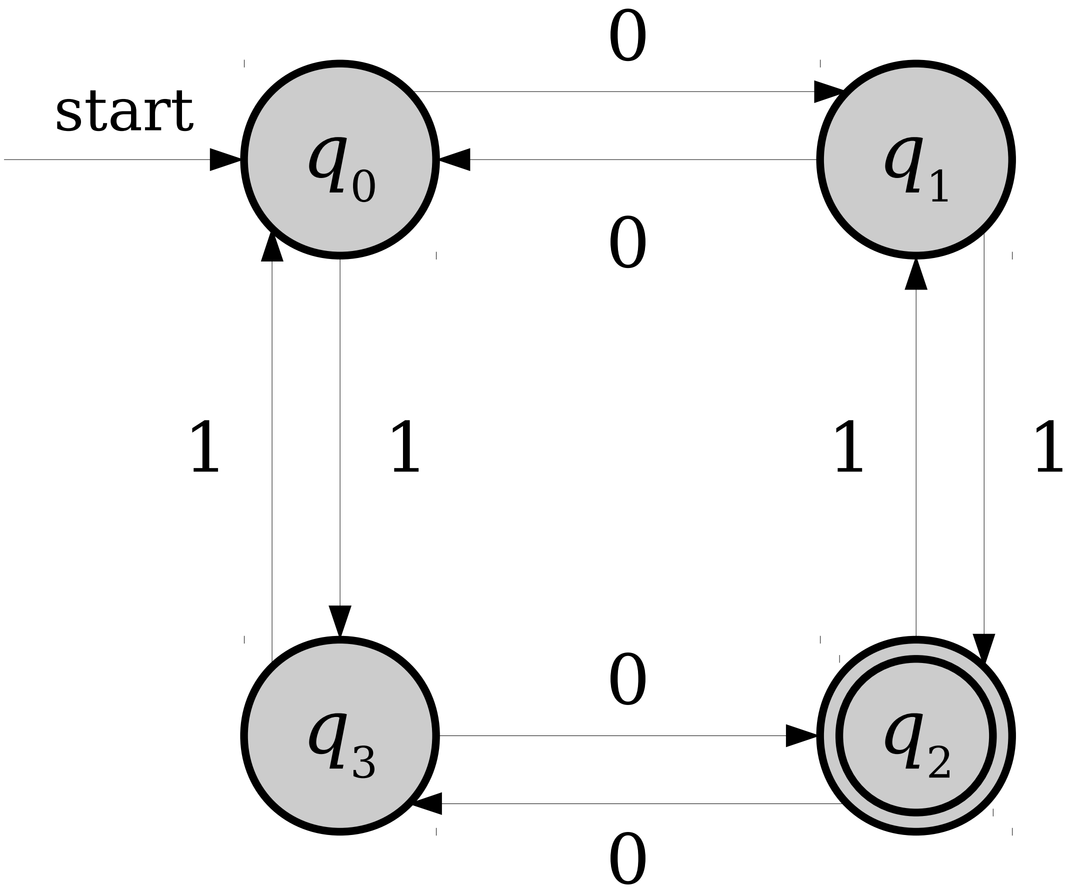
A ***finite automaton*** is a simple type of mathematical machine for determining whether a string is contained within some language.

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

A Simple Finite Automaton

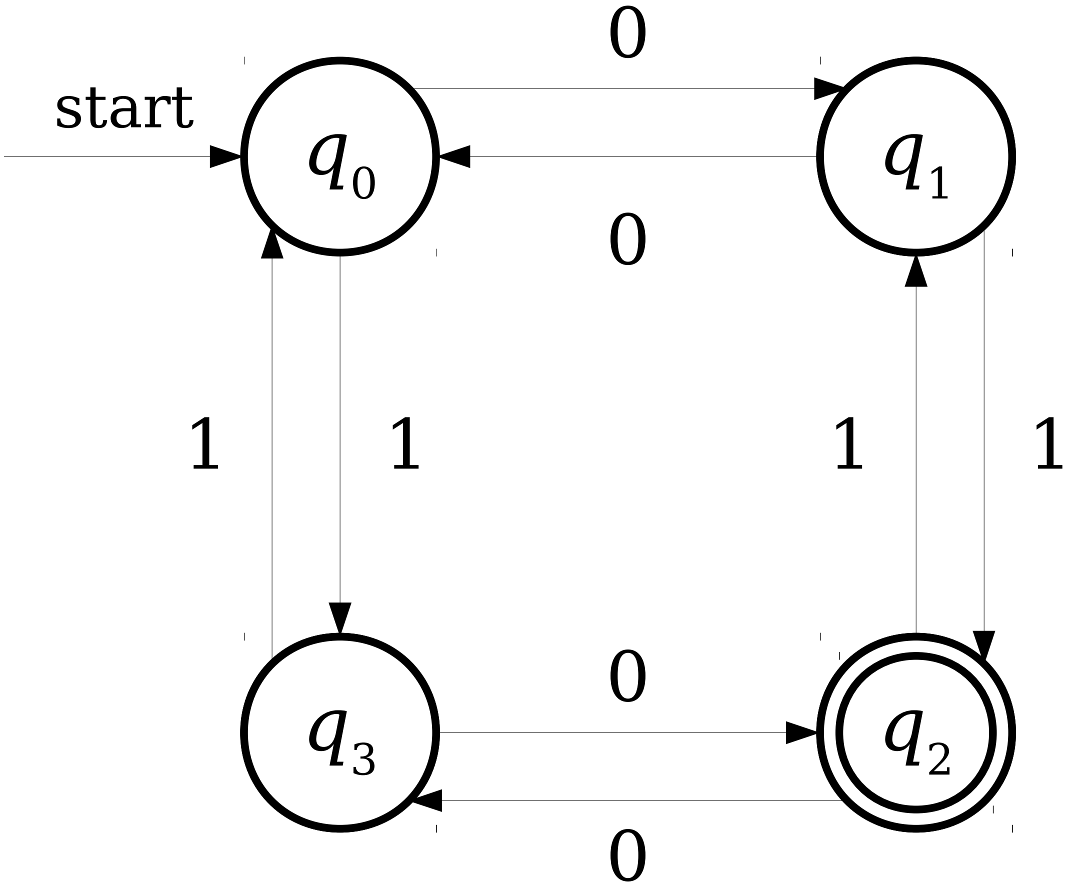


A Simple Finite Automaton

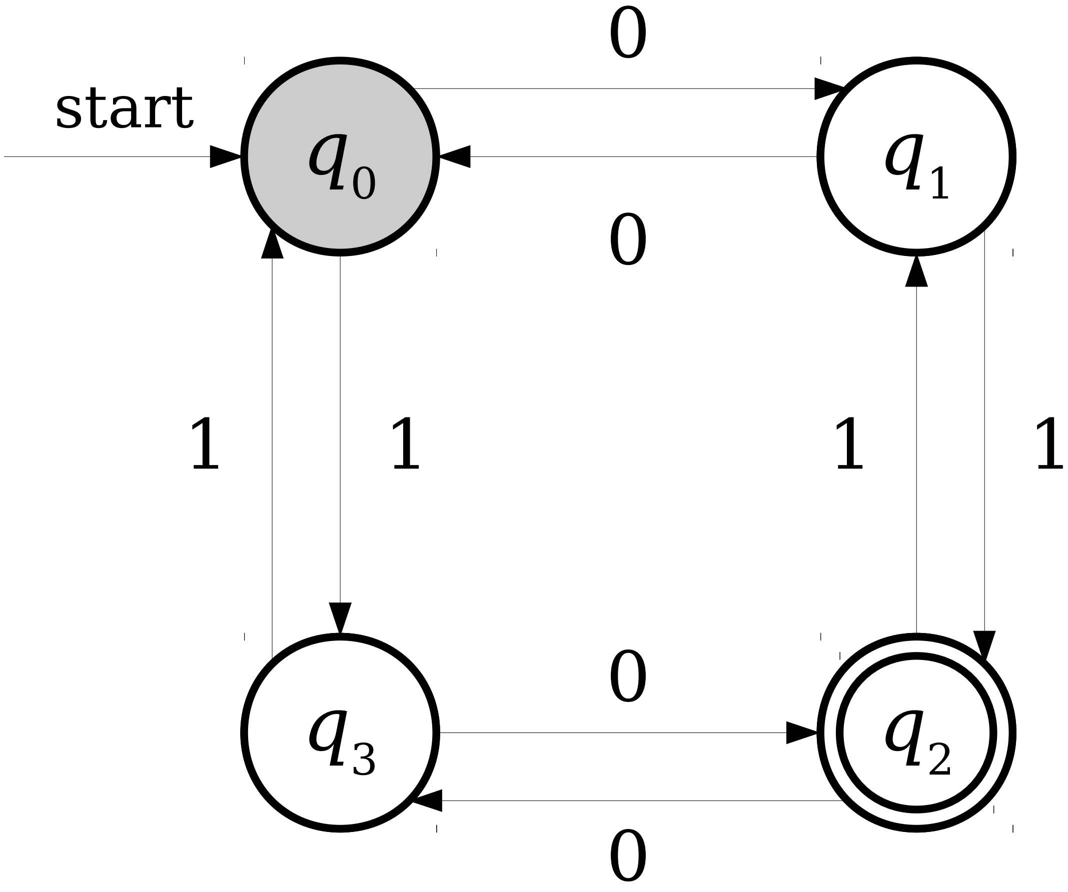


Each circle represents a **state** of the automaton.

A Simple Finite Automaton

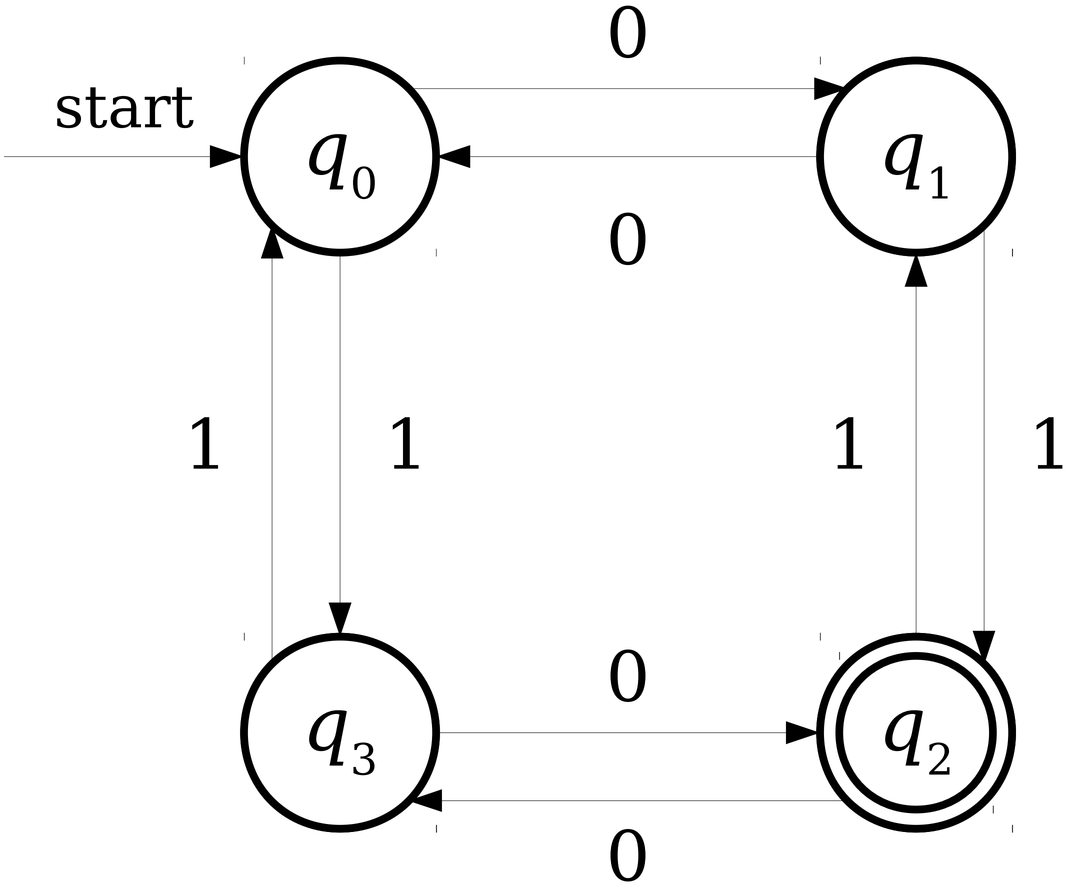


A Simple Finite Automaton

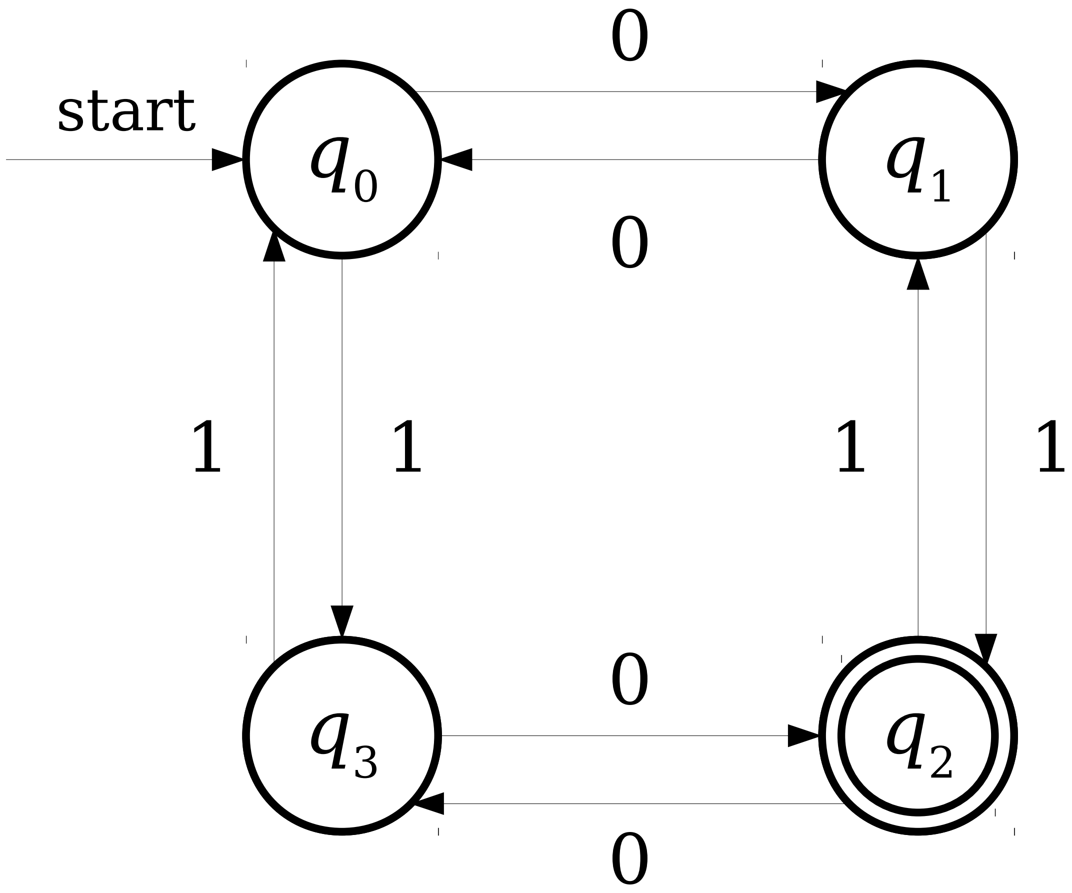


One special state is designated as the **start state**.

A Simple Finite Automaton

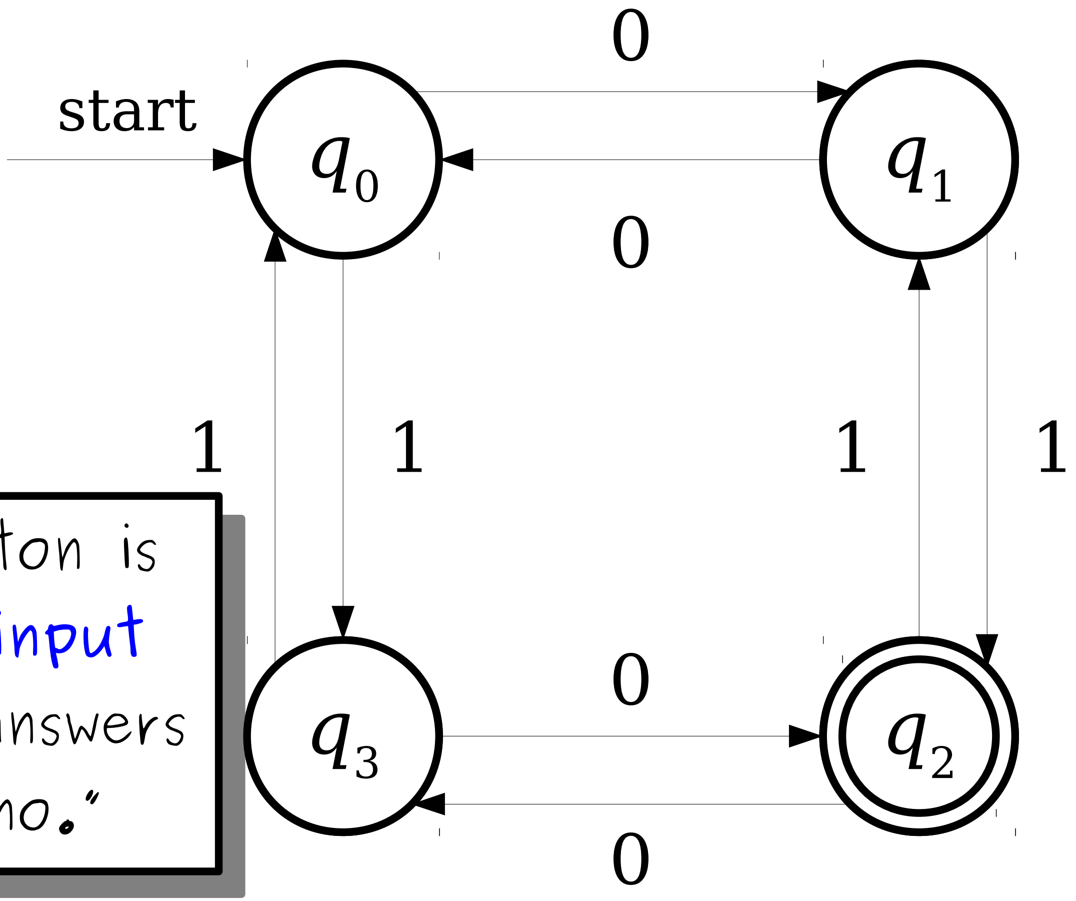


A Simple Finite Automaton



0 1 0 1 1 0

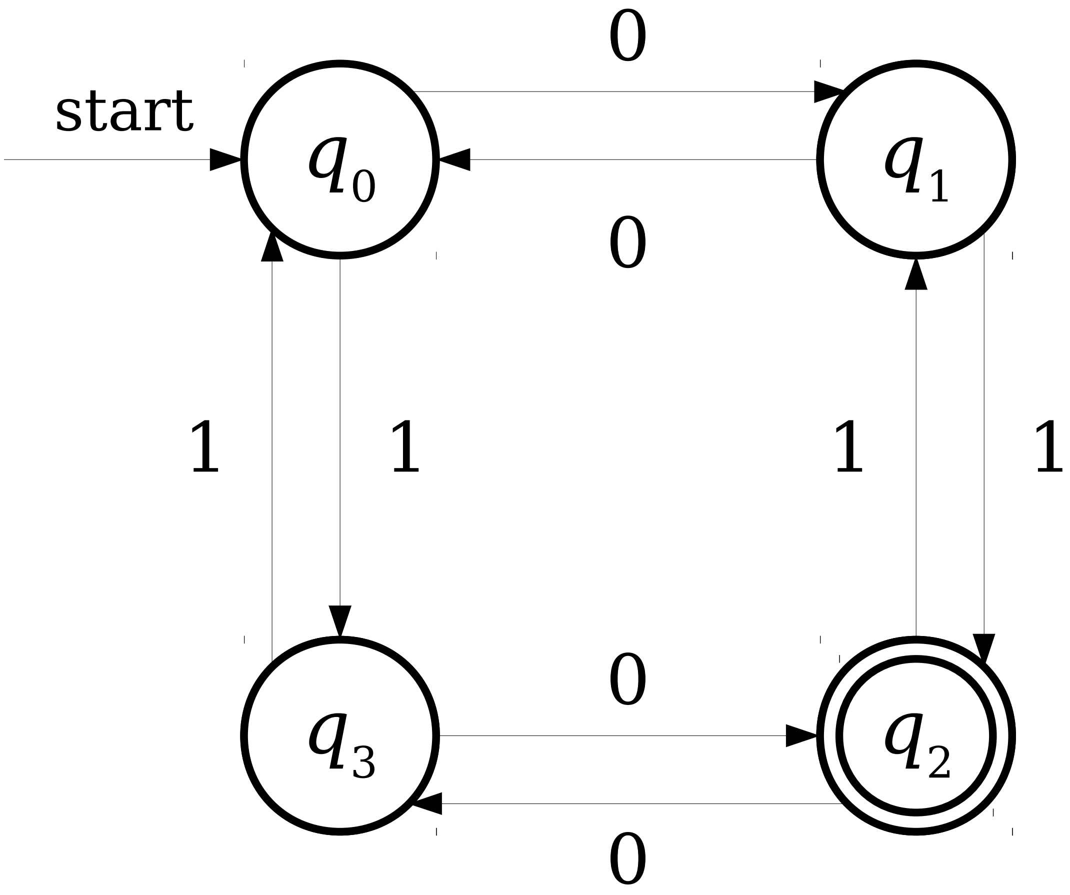
A Simple Finite Automaton



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

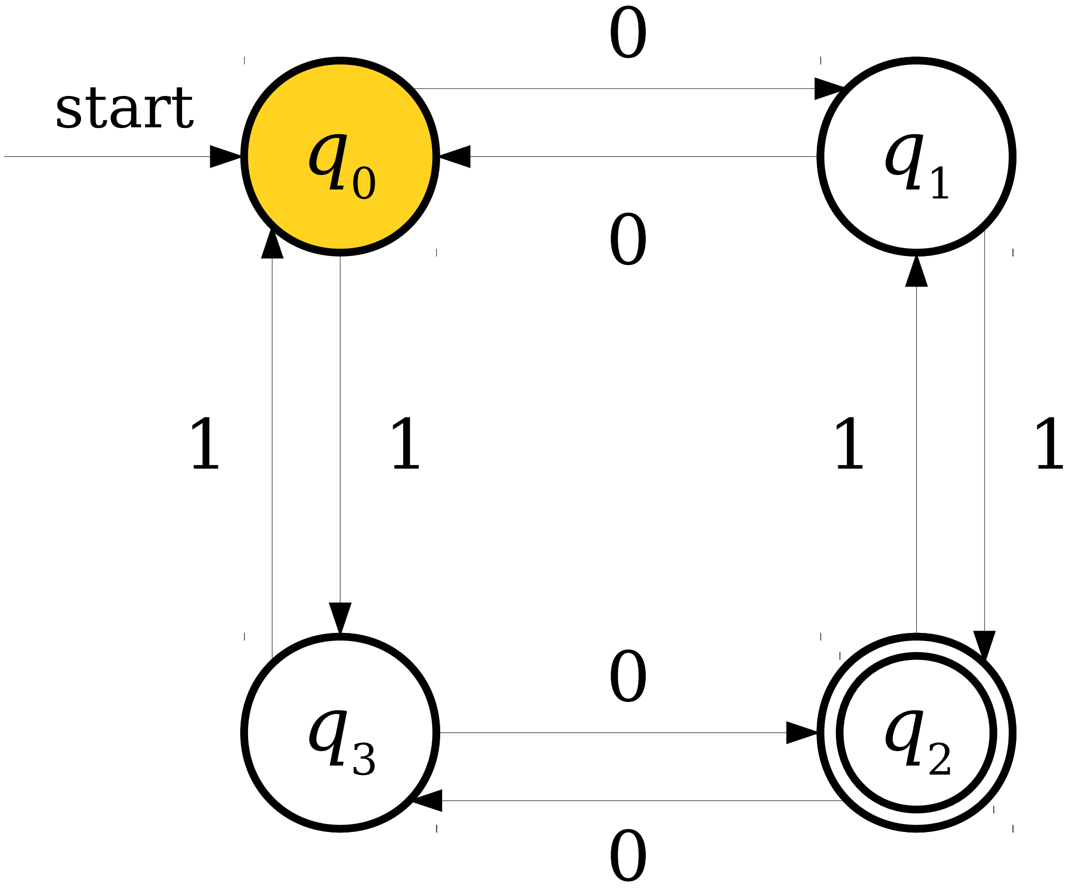
0 1 0 1 1 0

A Simple Finite Automaton



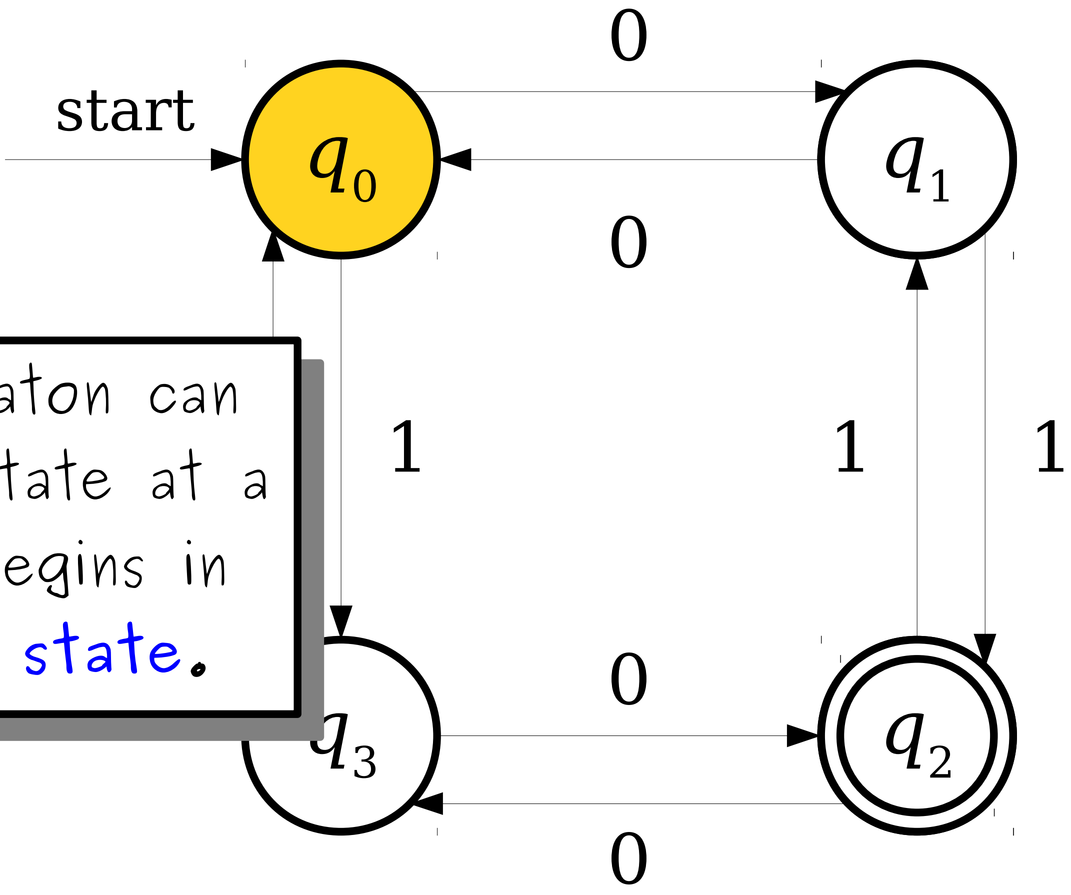
0 1 0 1 1 0

A Simple Finite Automaton



0 1 0 1 1 0

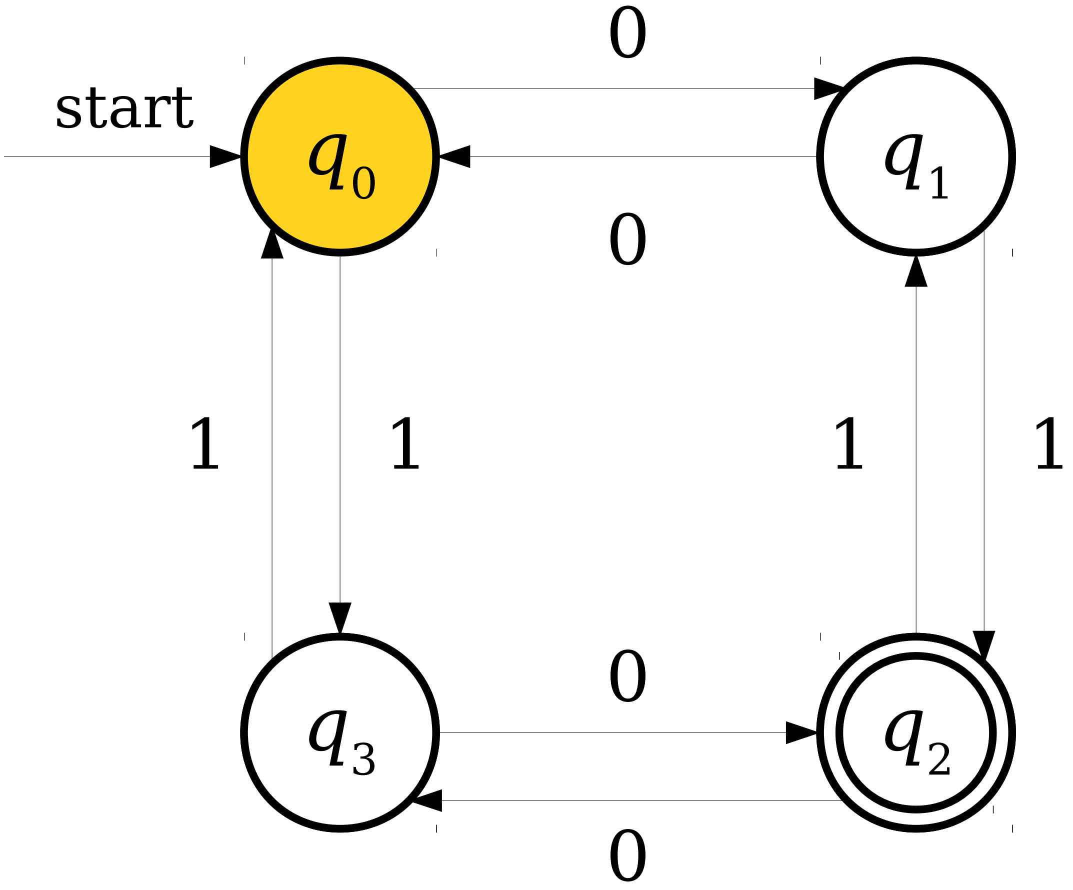
A Simple Finite Automaton



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

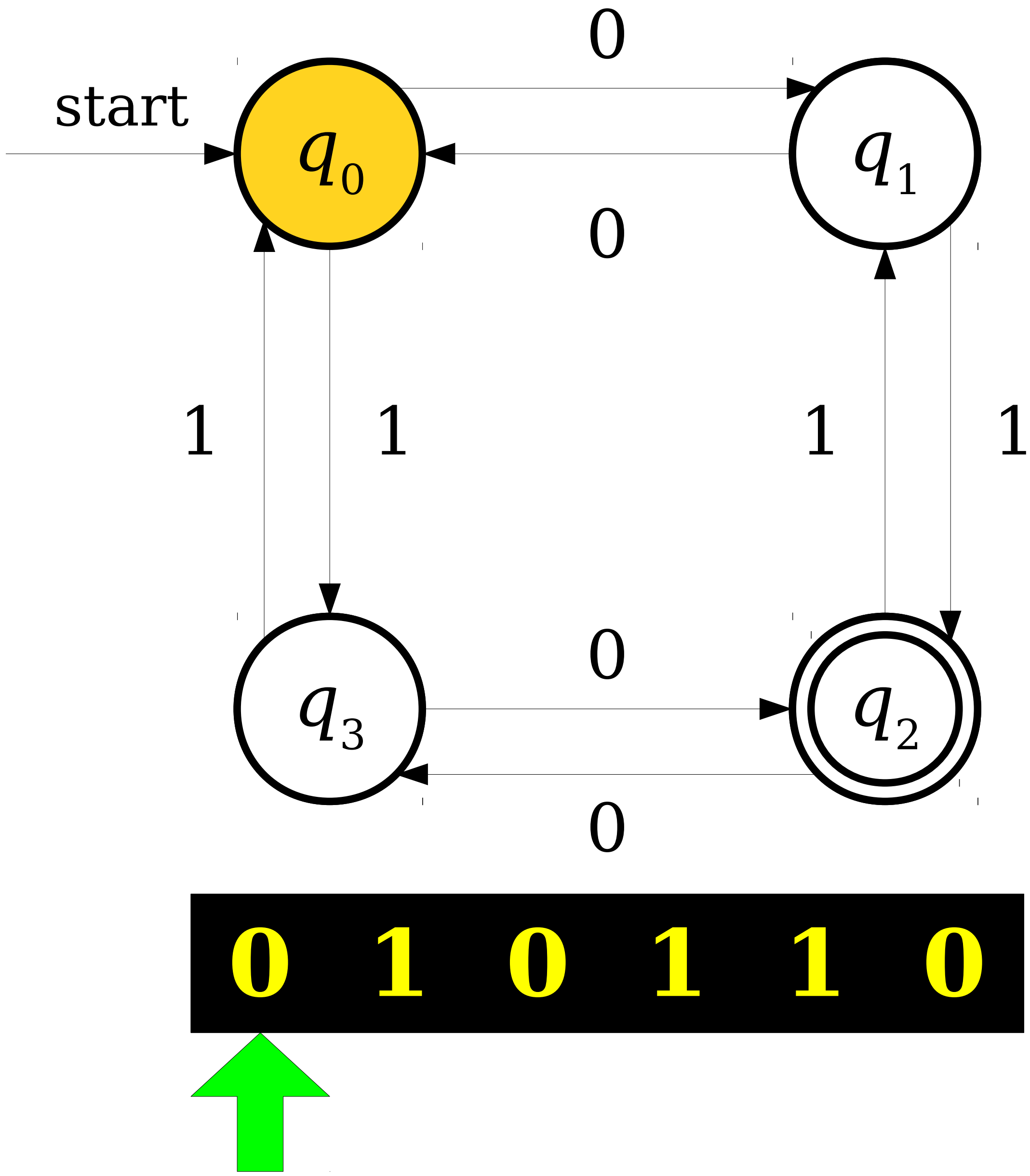
0 1 0 1 1 0

A Simple Finite Automaton

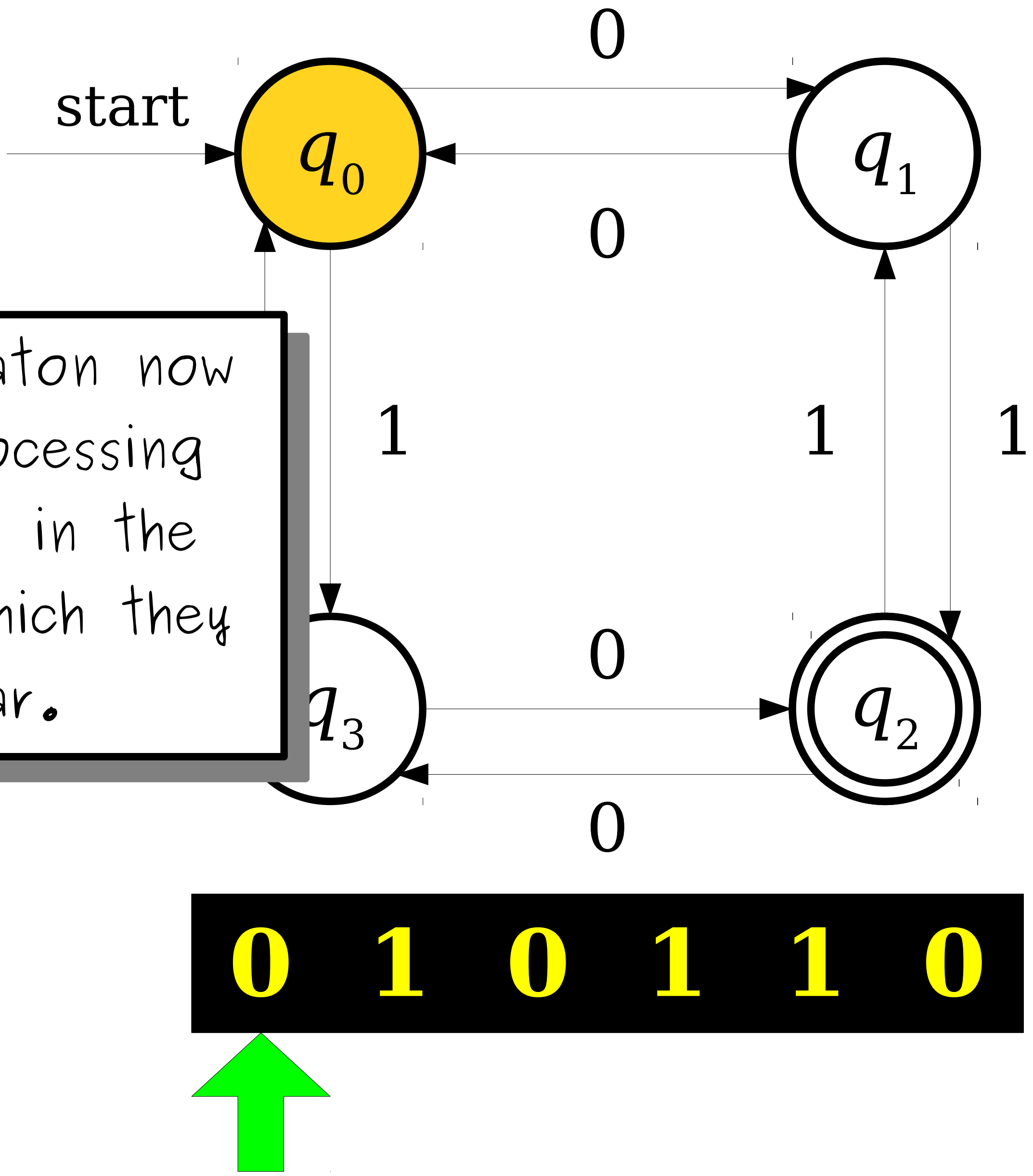


0 1 0 1 1 0

A Simple Finite Automaton

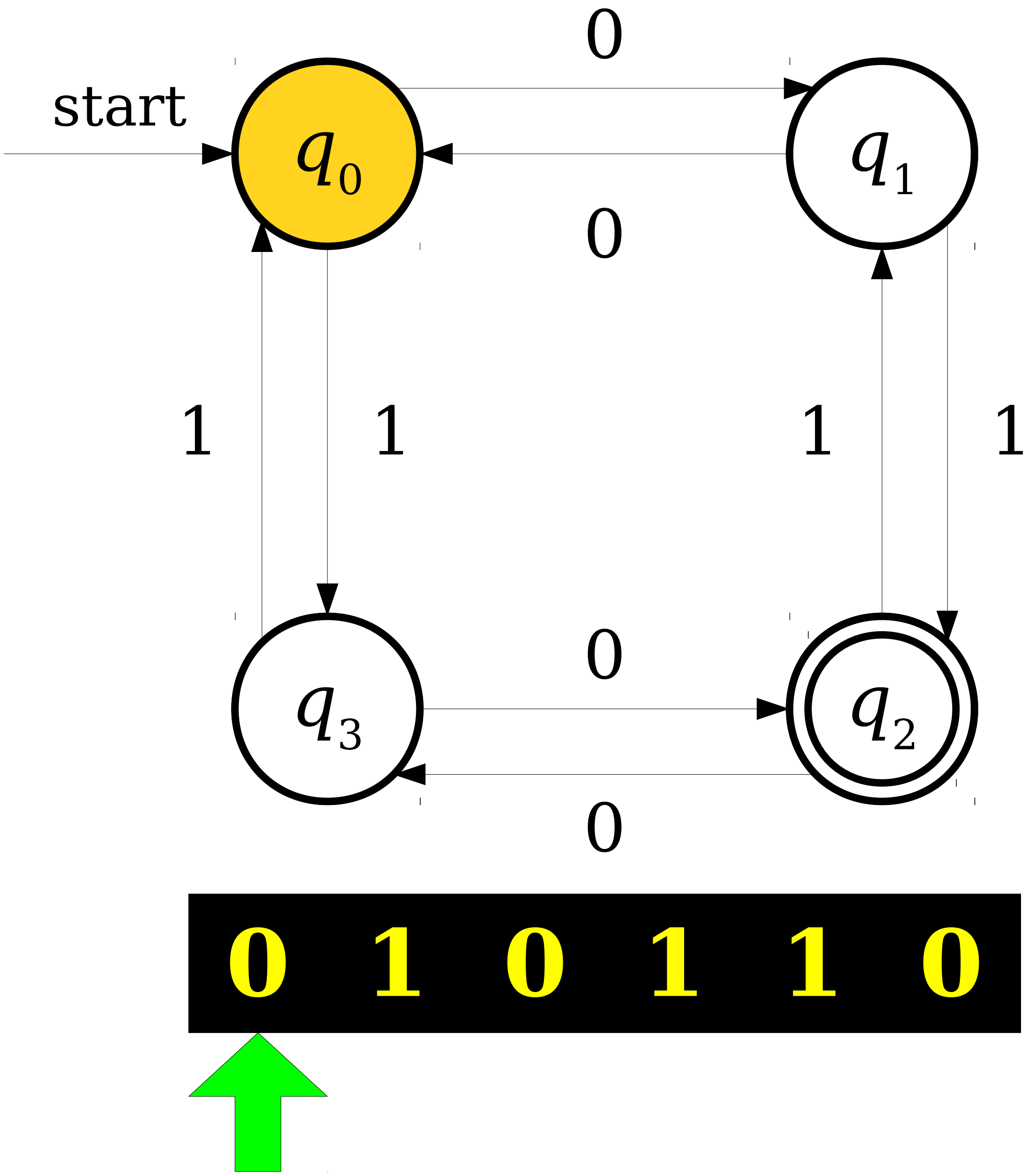


A Simple Finite Automaton

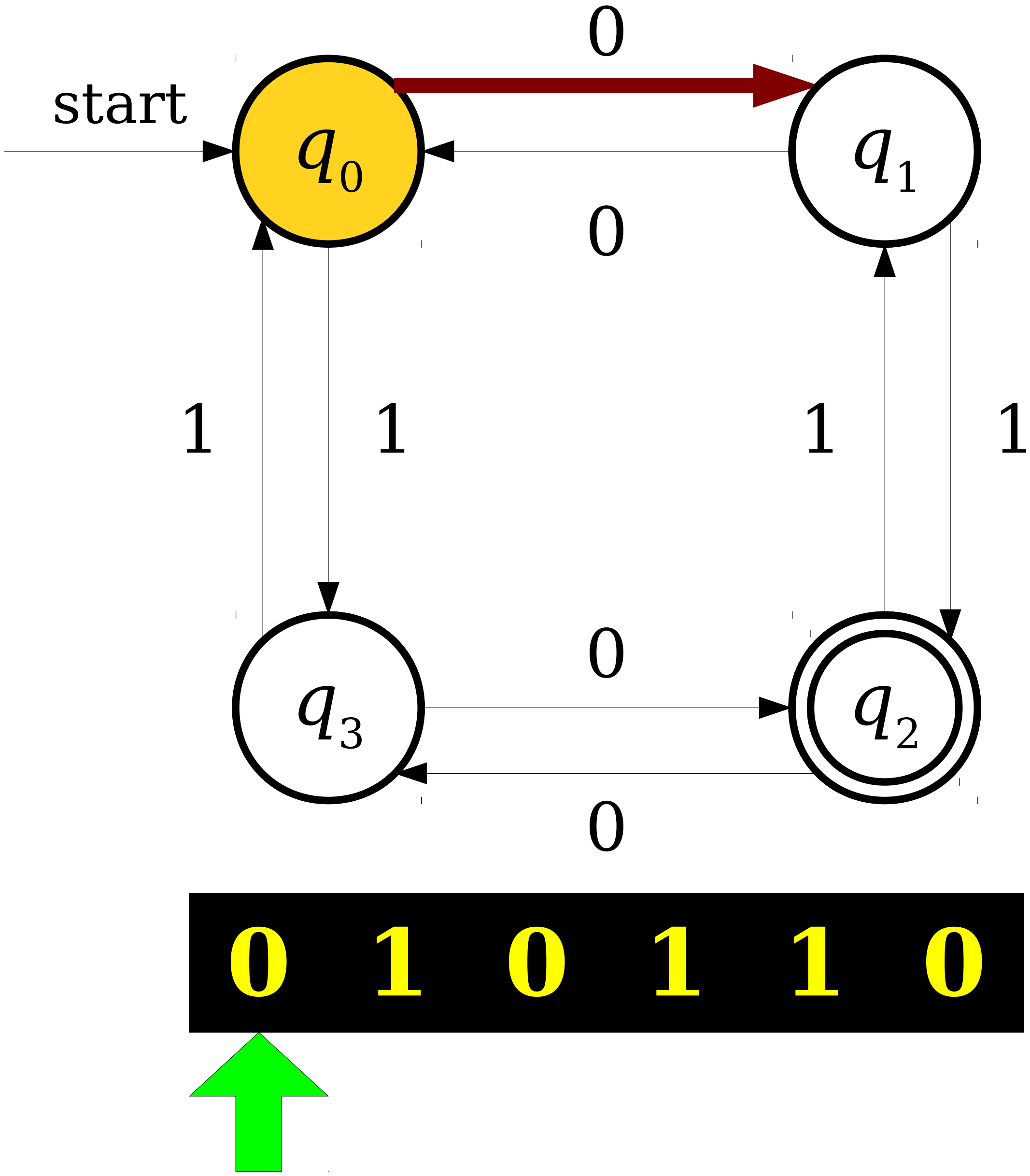


The automaton now begins processing characters in the order in which they appear.

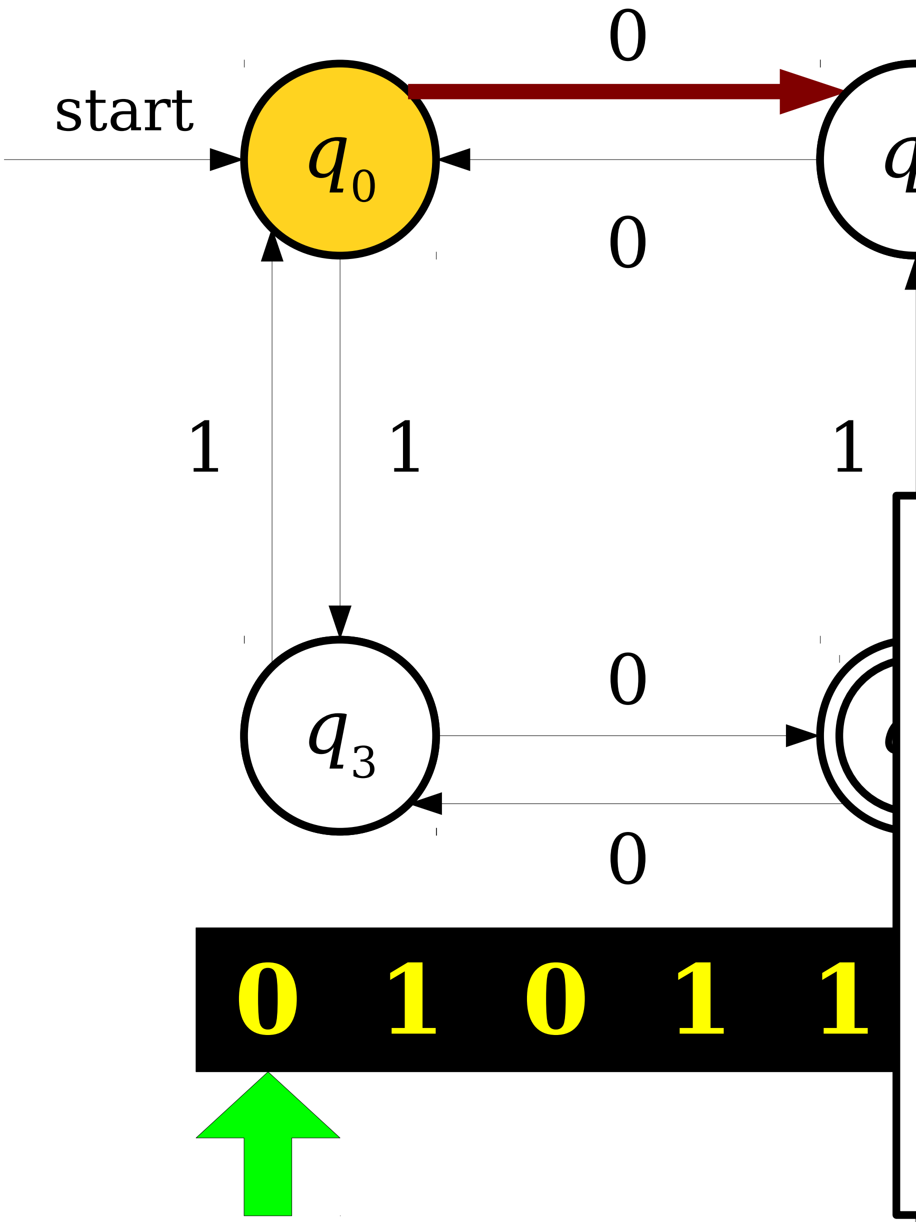
A Simple Finite Automaton



A Simple Finite Automaton

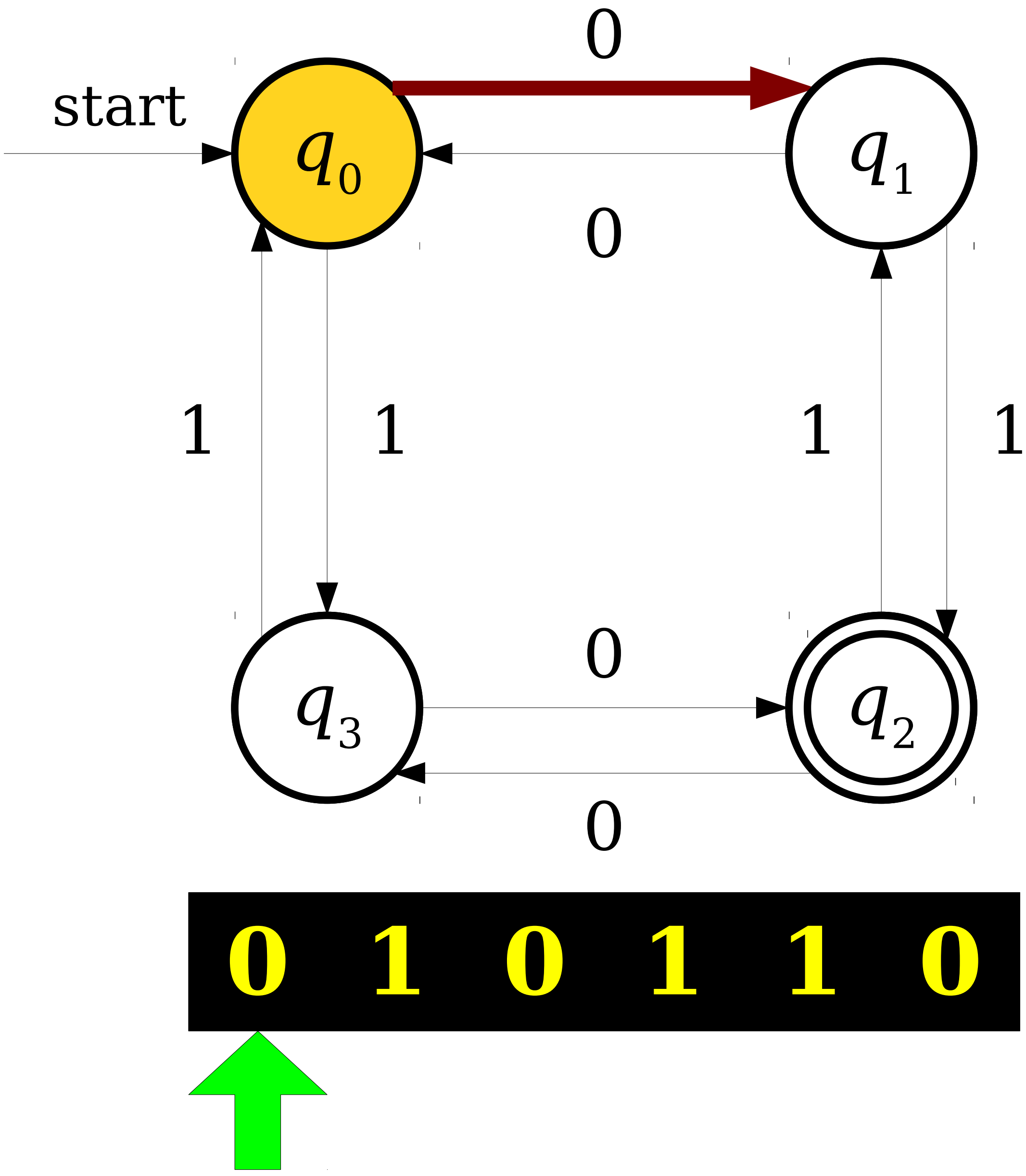


A Simple Finite Automaton

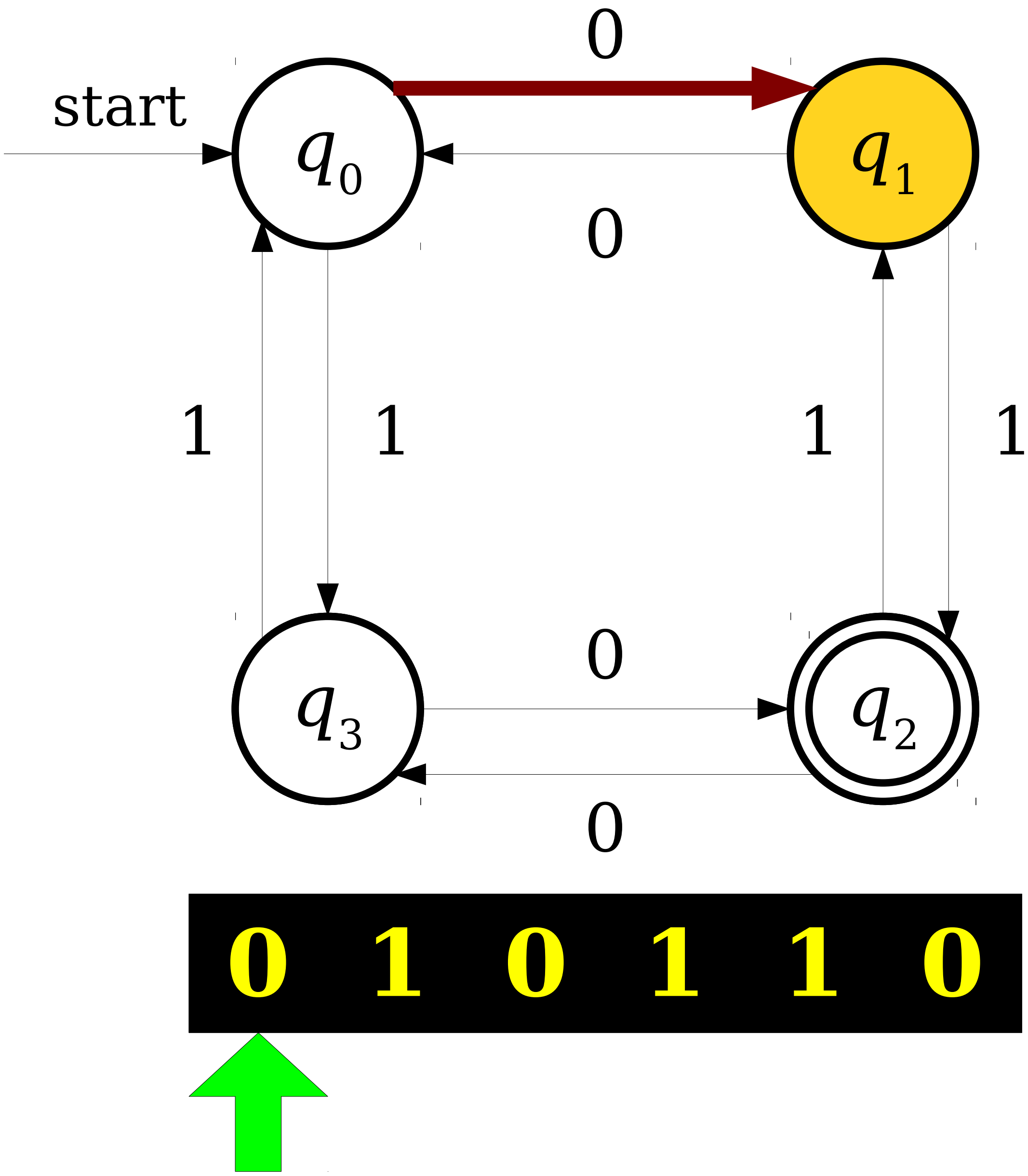


Each arrow in this diagram represents a **transition**. The automaton always follows the transition corresponding to the current symbol being read.

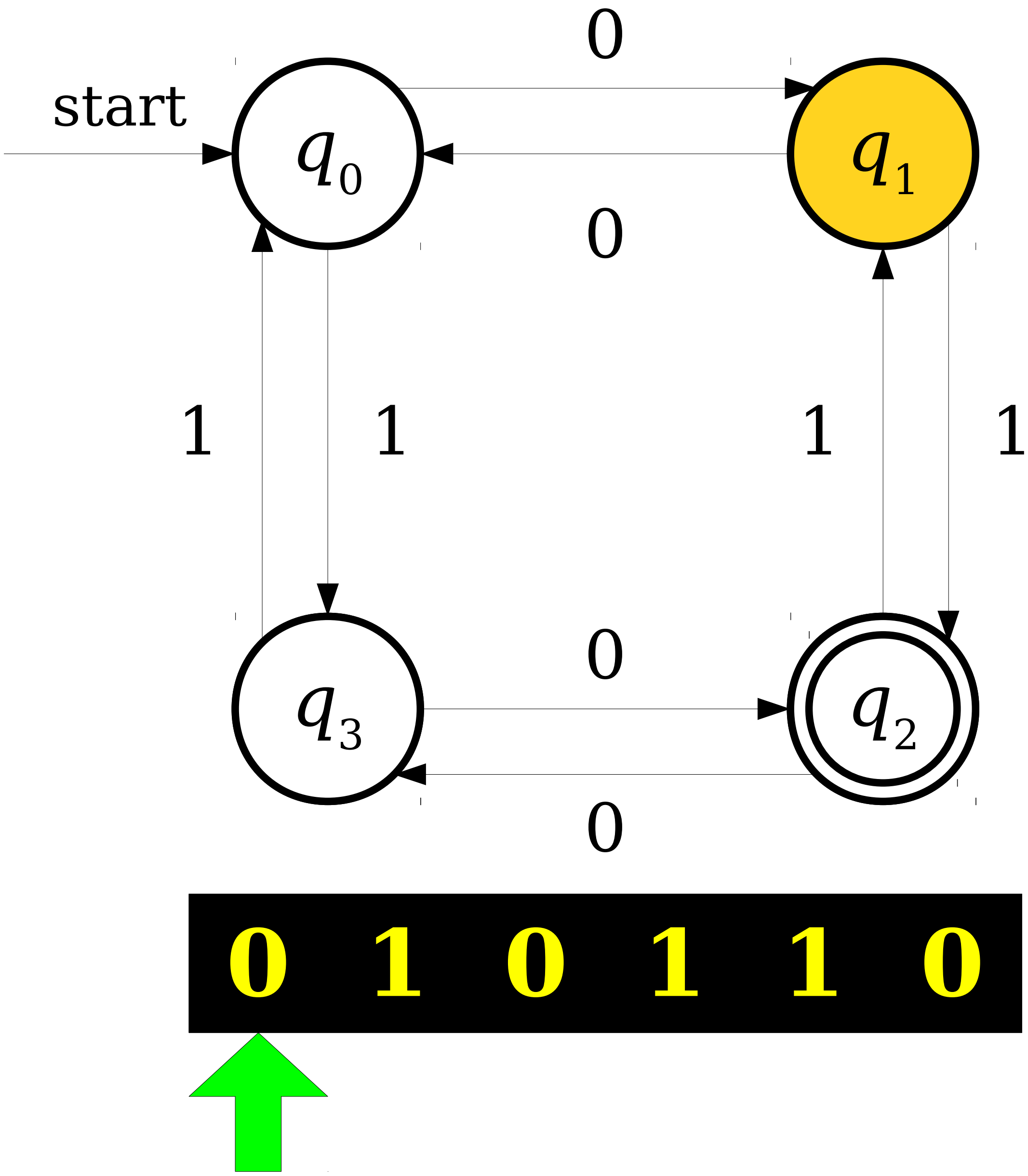
A Simple Finite Automaton



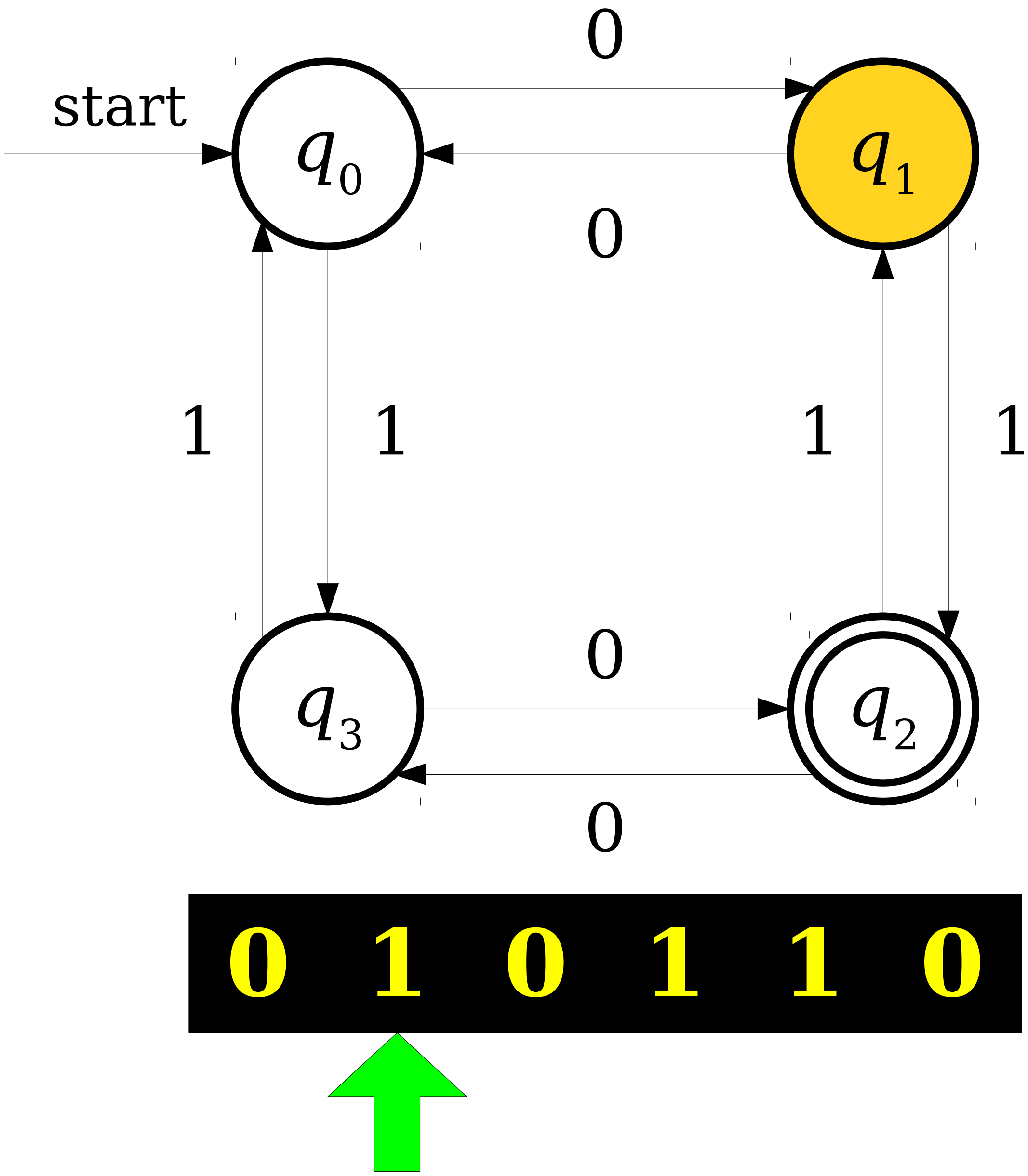
A Simple Finite Automaton



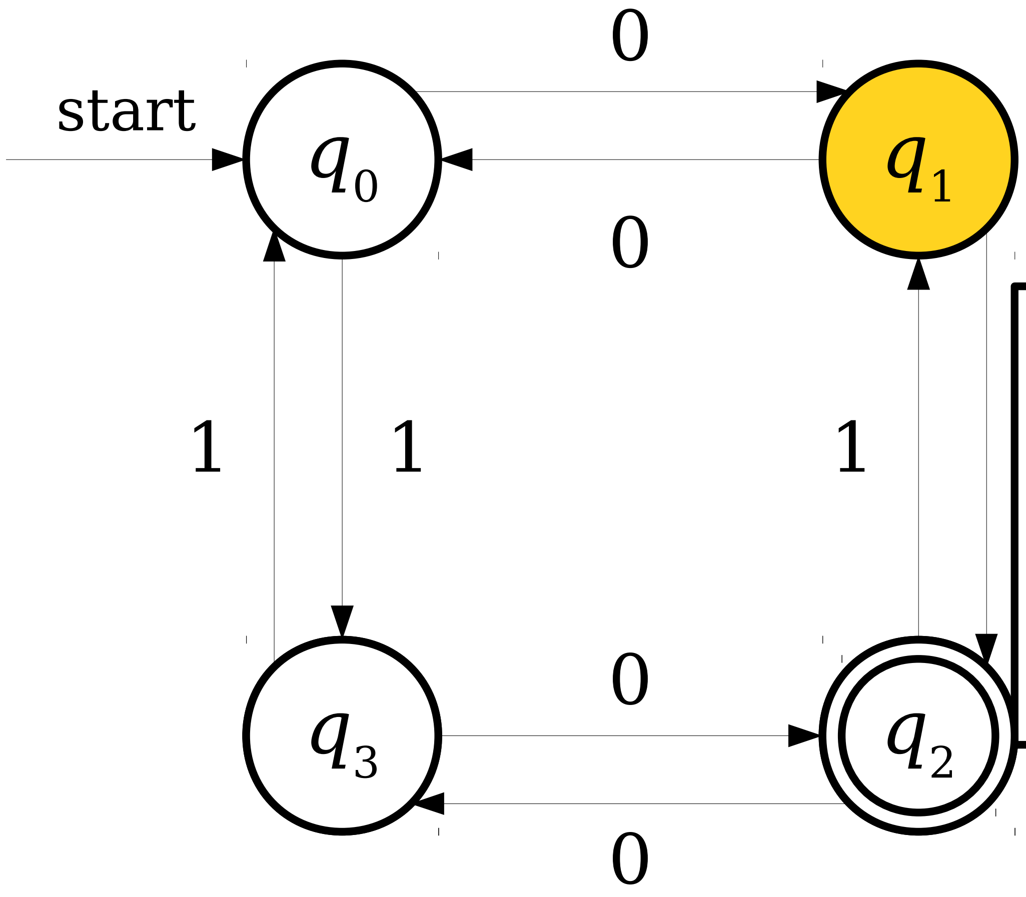
A Simple Finite Automaton



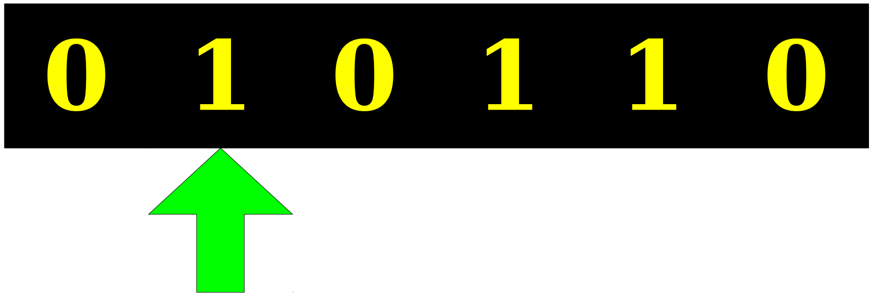
A Simple Finite Automaton



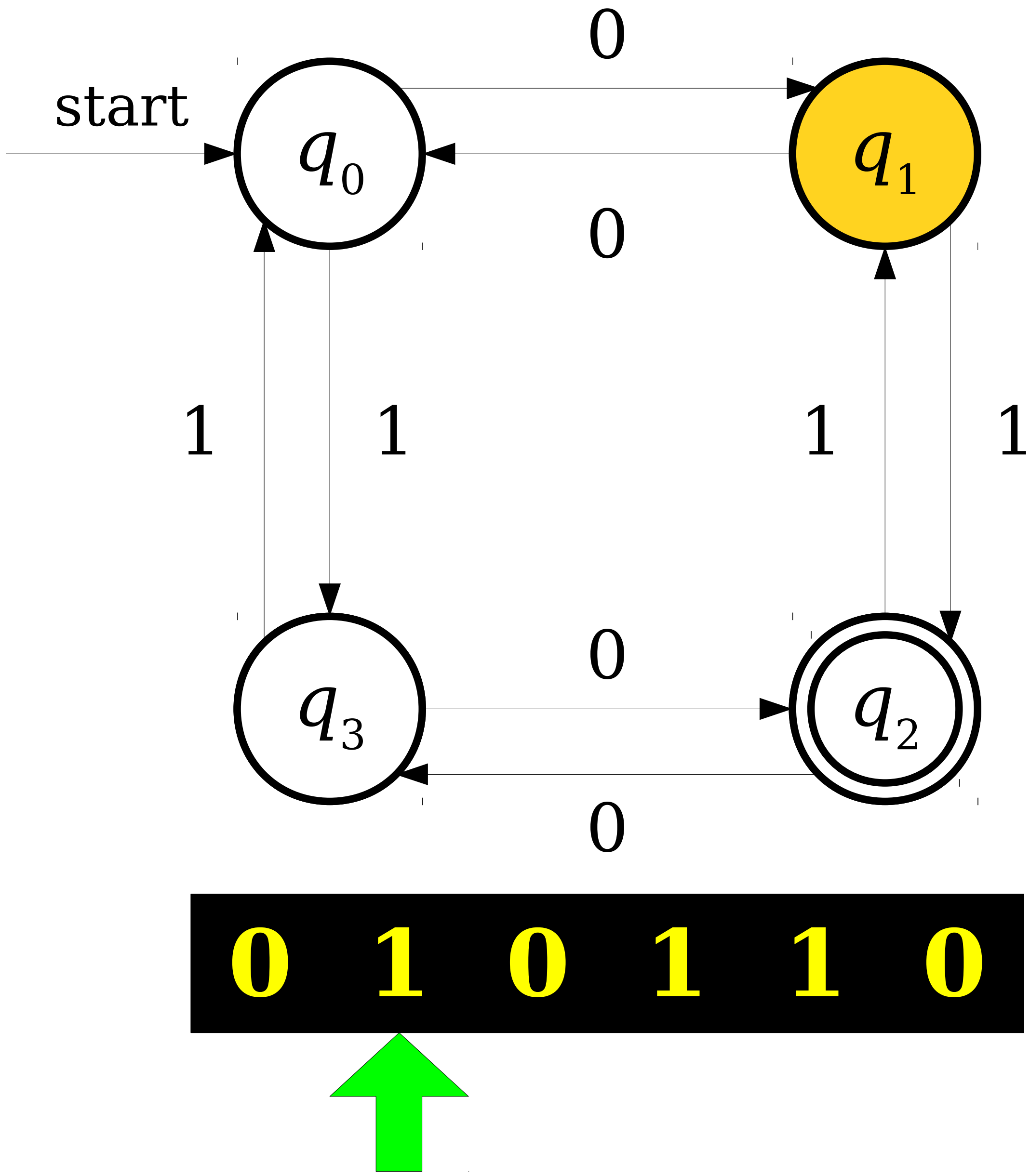
A Simple Finite Automaton



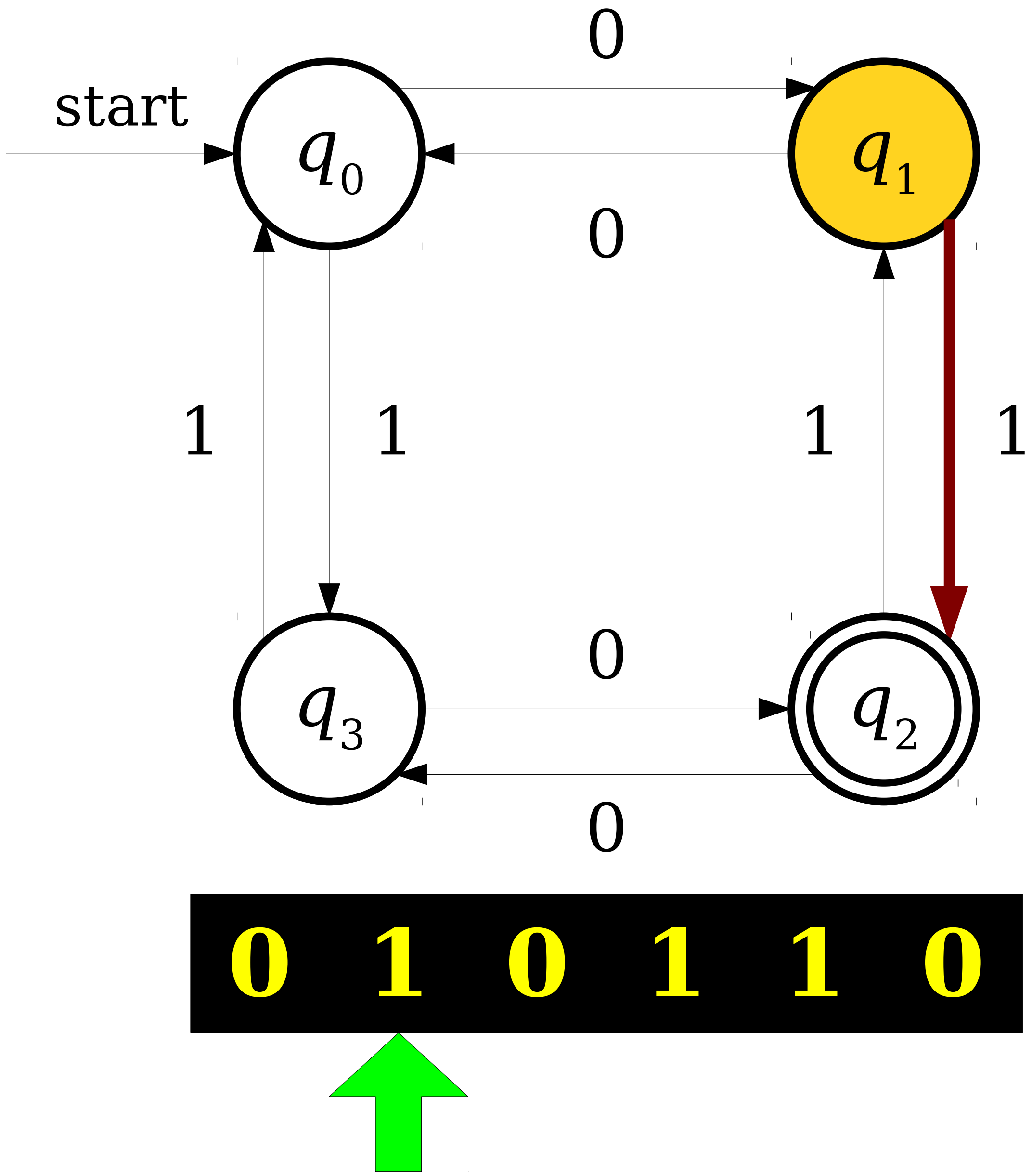
After transitioning, the automaton considers the next symbol in the input.



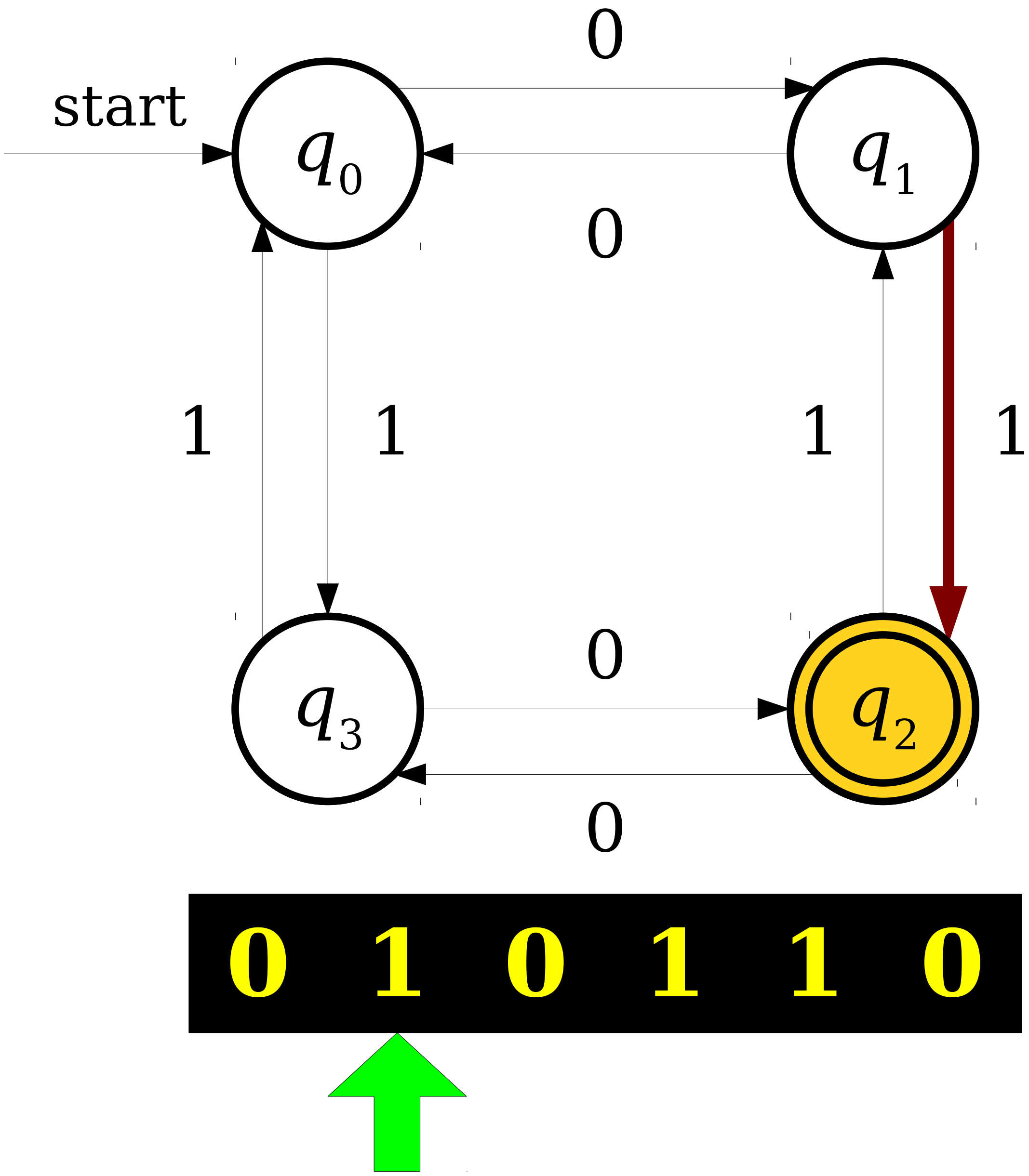
A Simple Finite Automaton



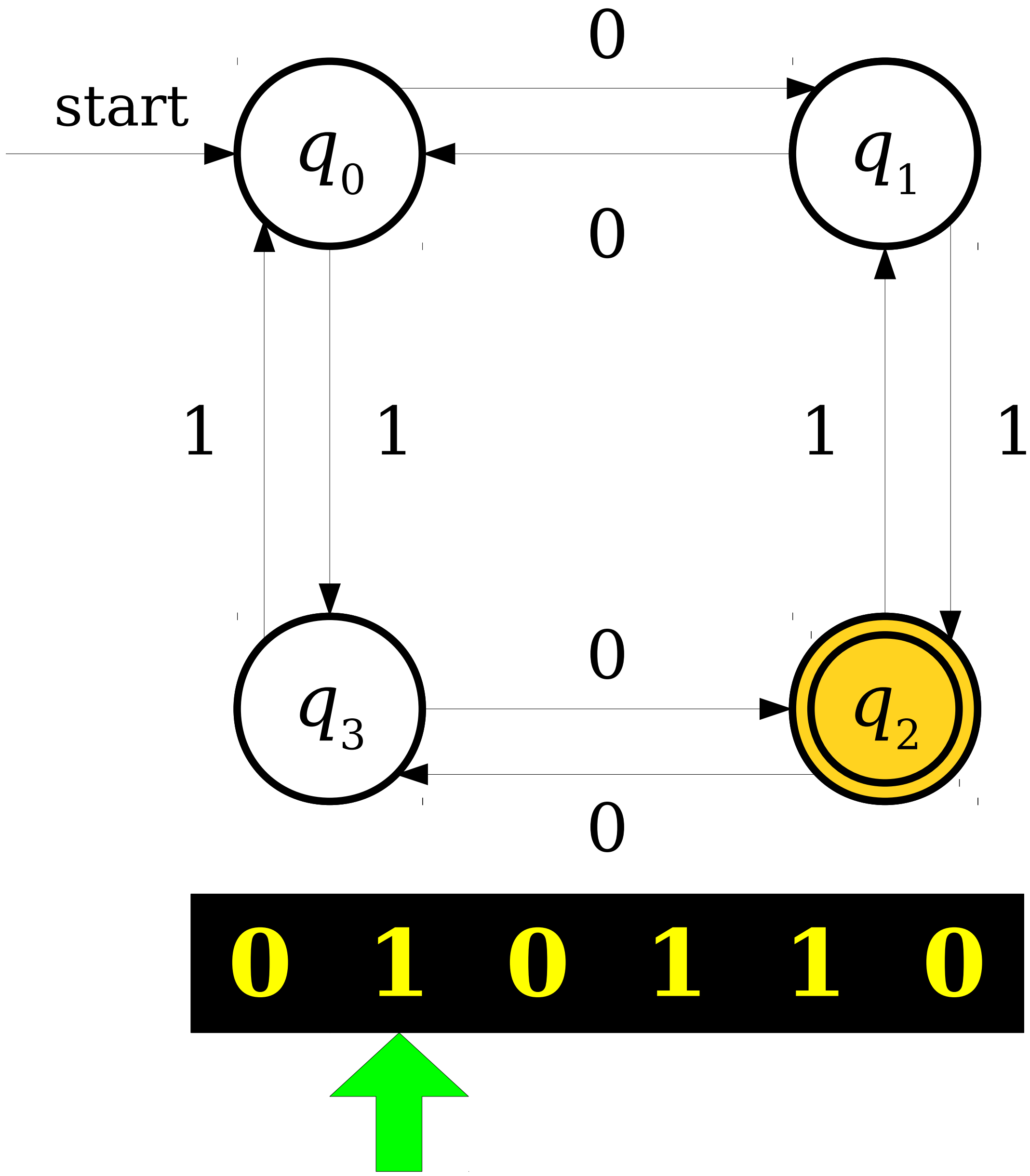
A Simple Finite Automaton



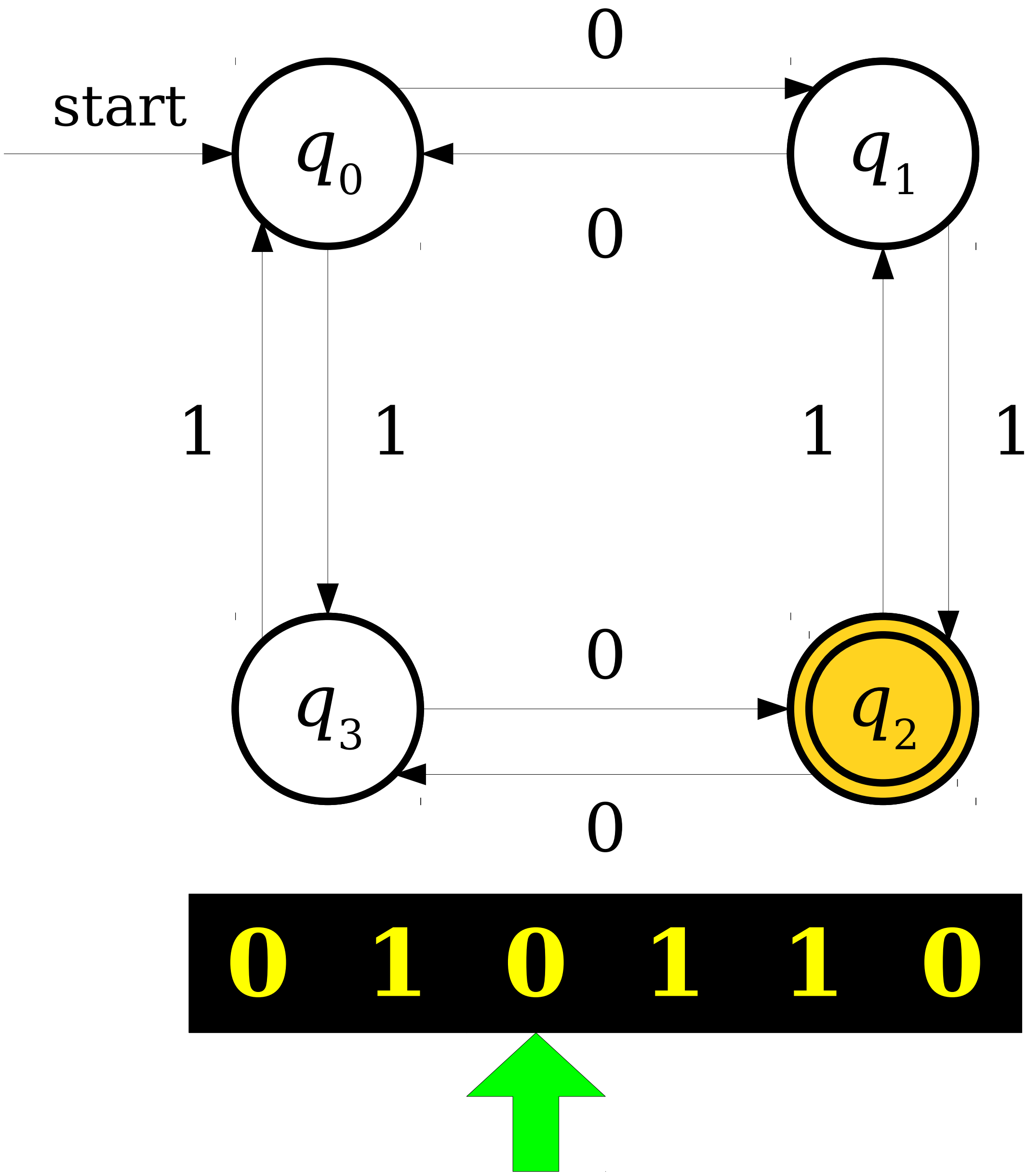
A Simple Finite Automaton



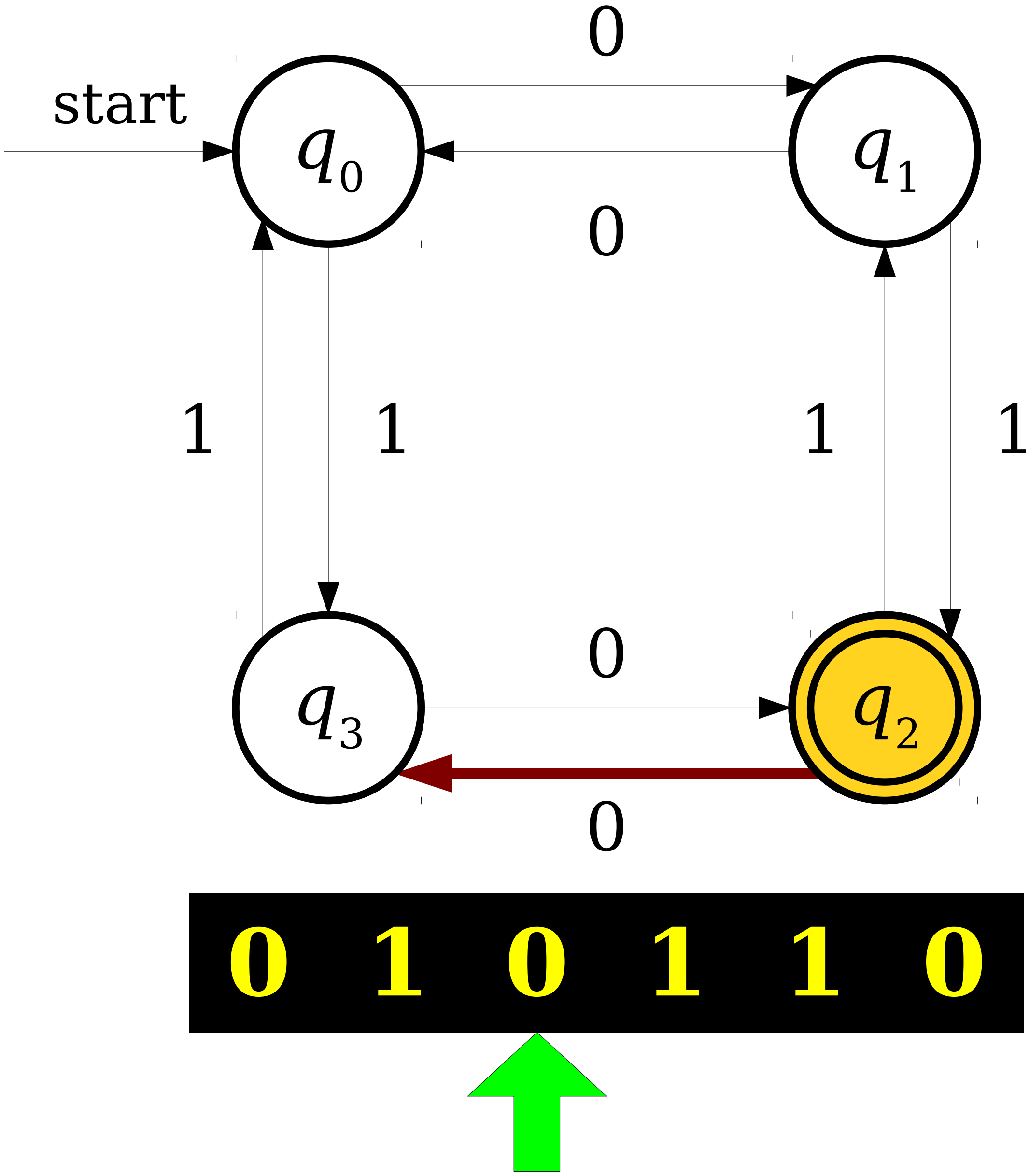
A Simple Finite Automaton



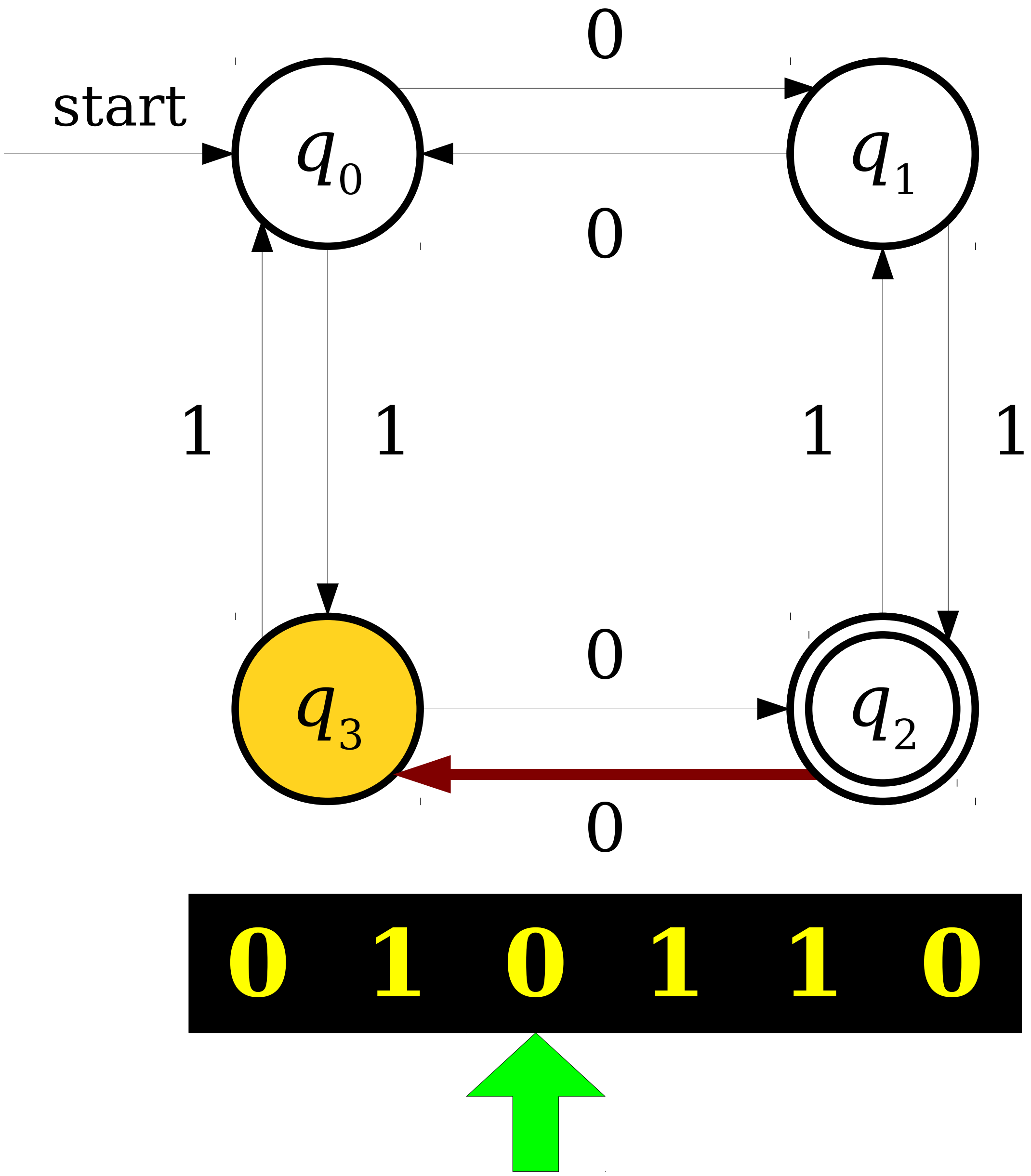
A Simple Finite Automaton



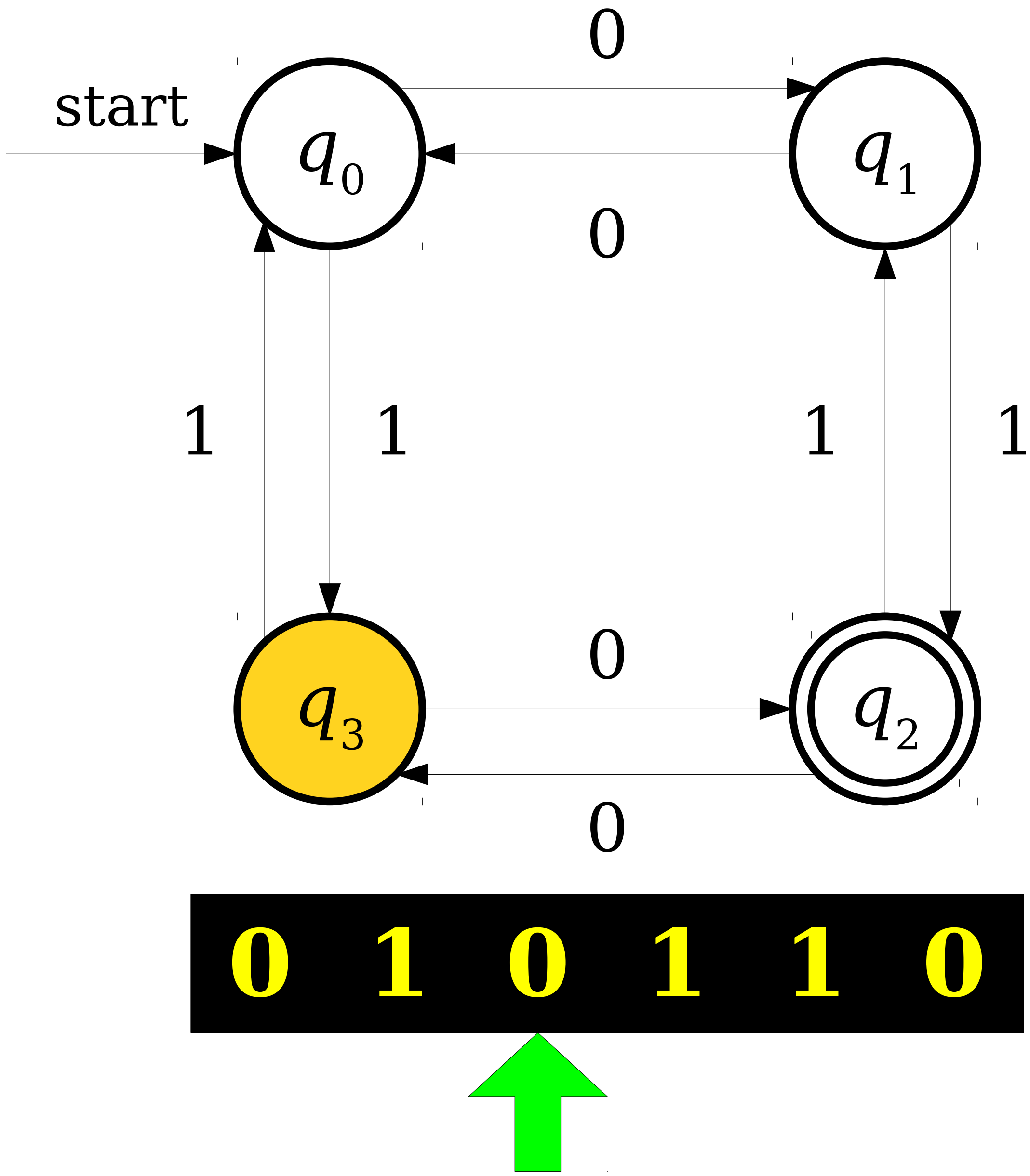
A Simple Finite Automaton



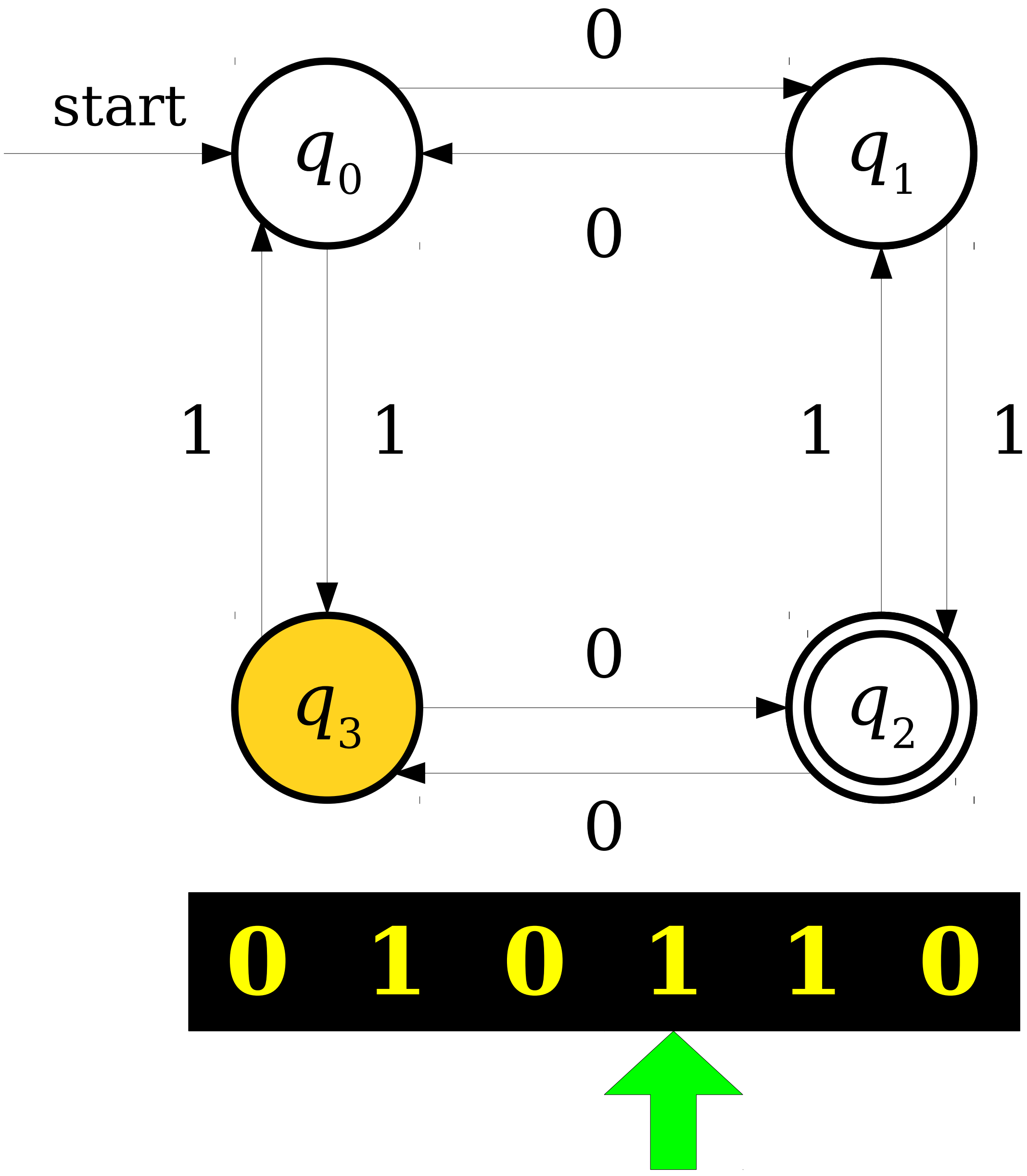
A Simple Finite Automaton



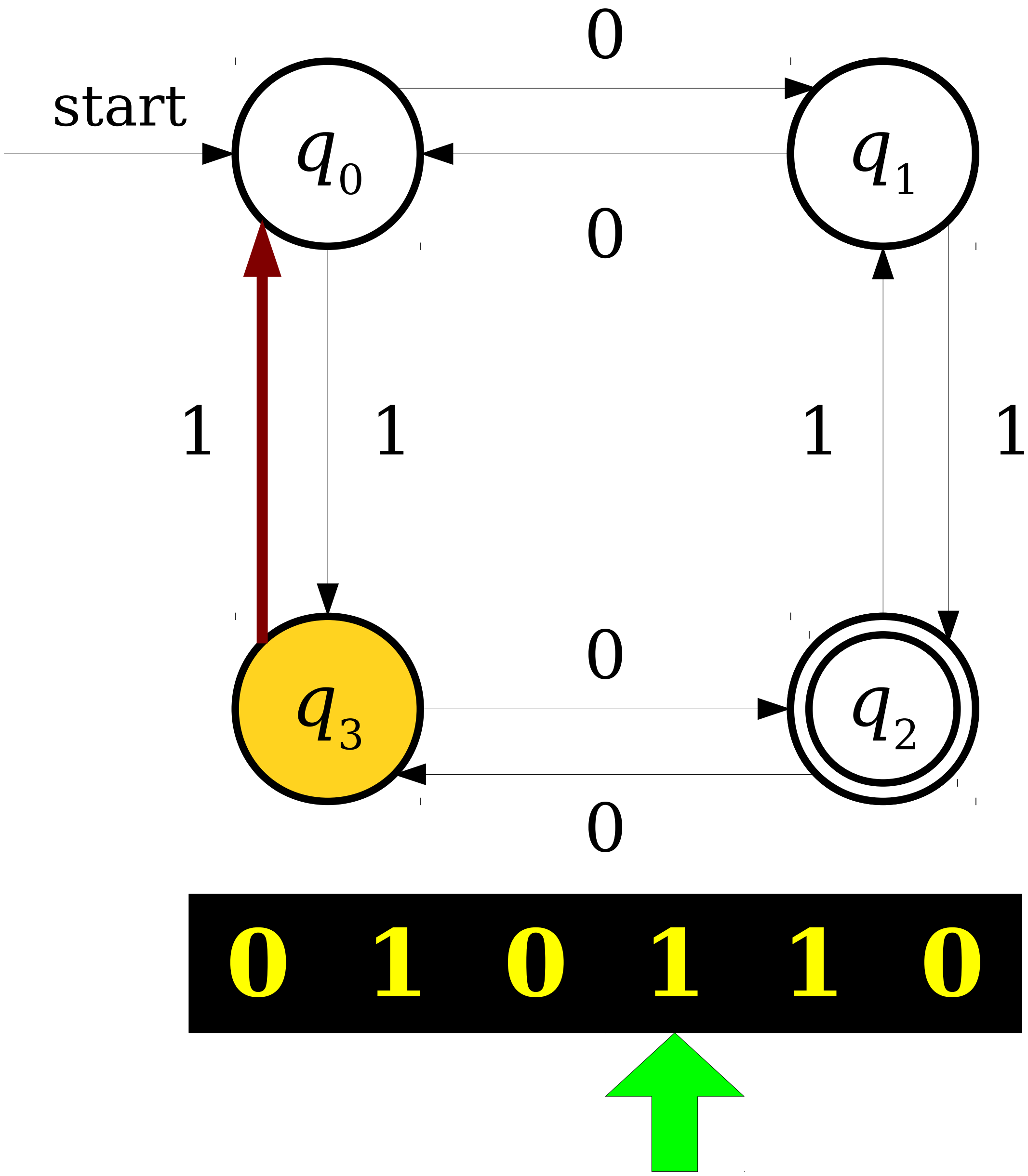
A Simple Finite Automaton



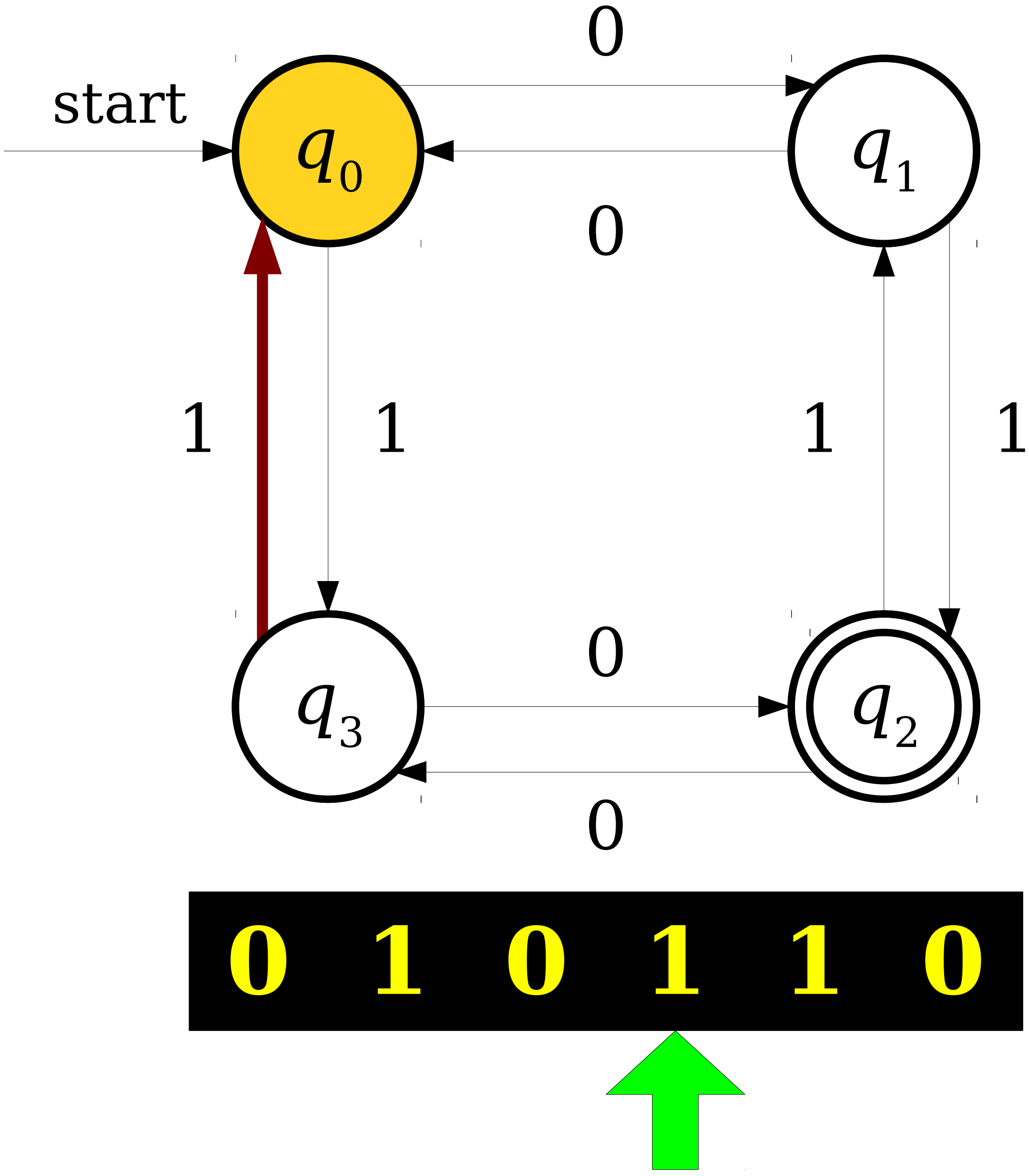
A Simple Finite Automaton



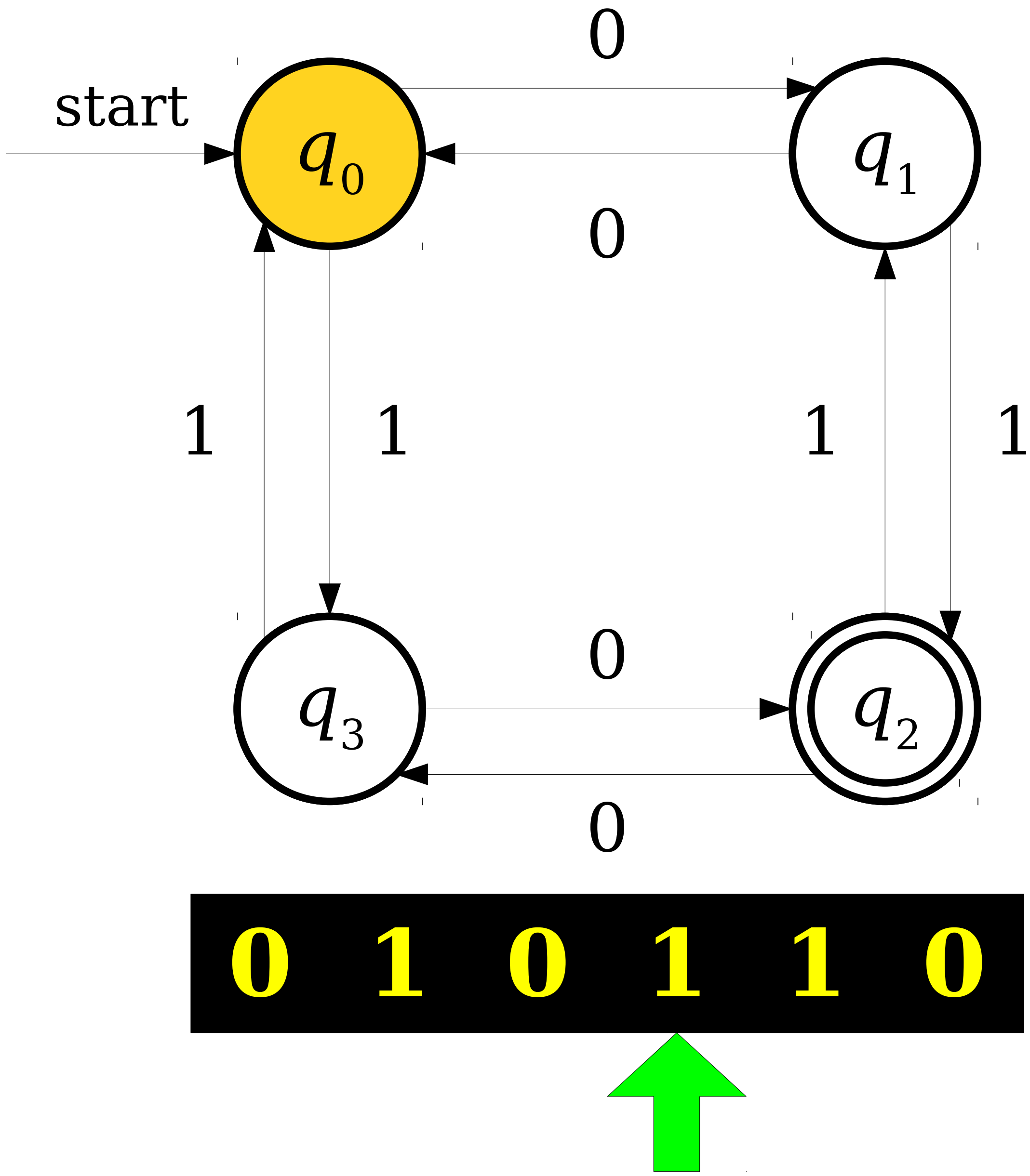
A Simple Finite Automaton



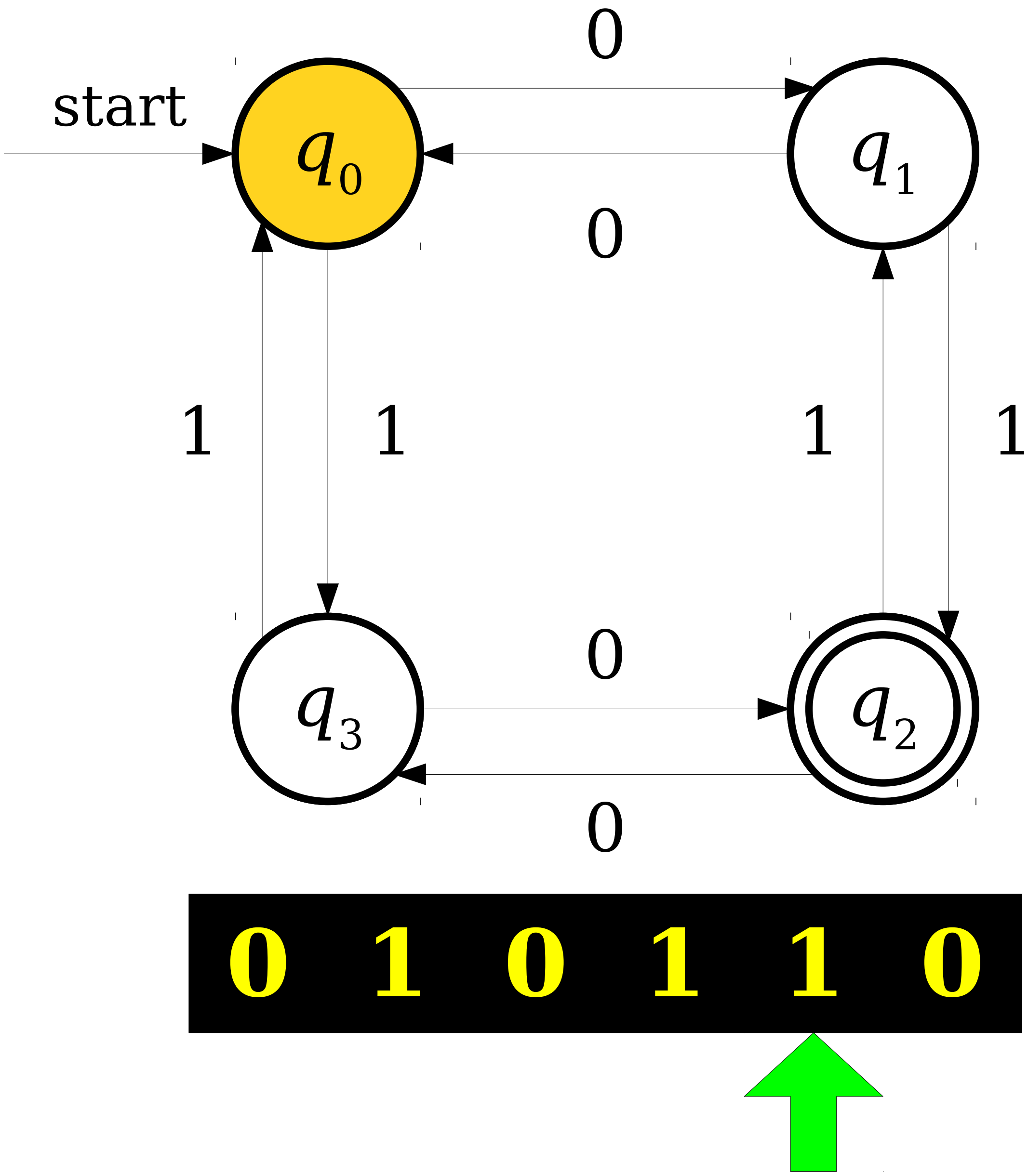
A Simple Finite Automaton



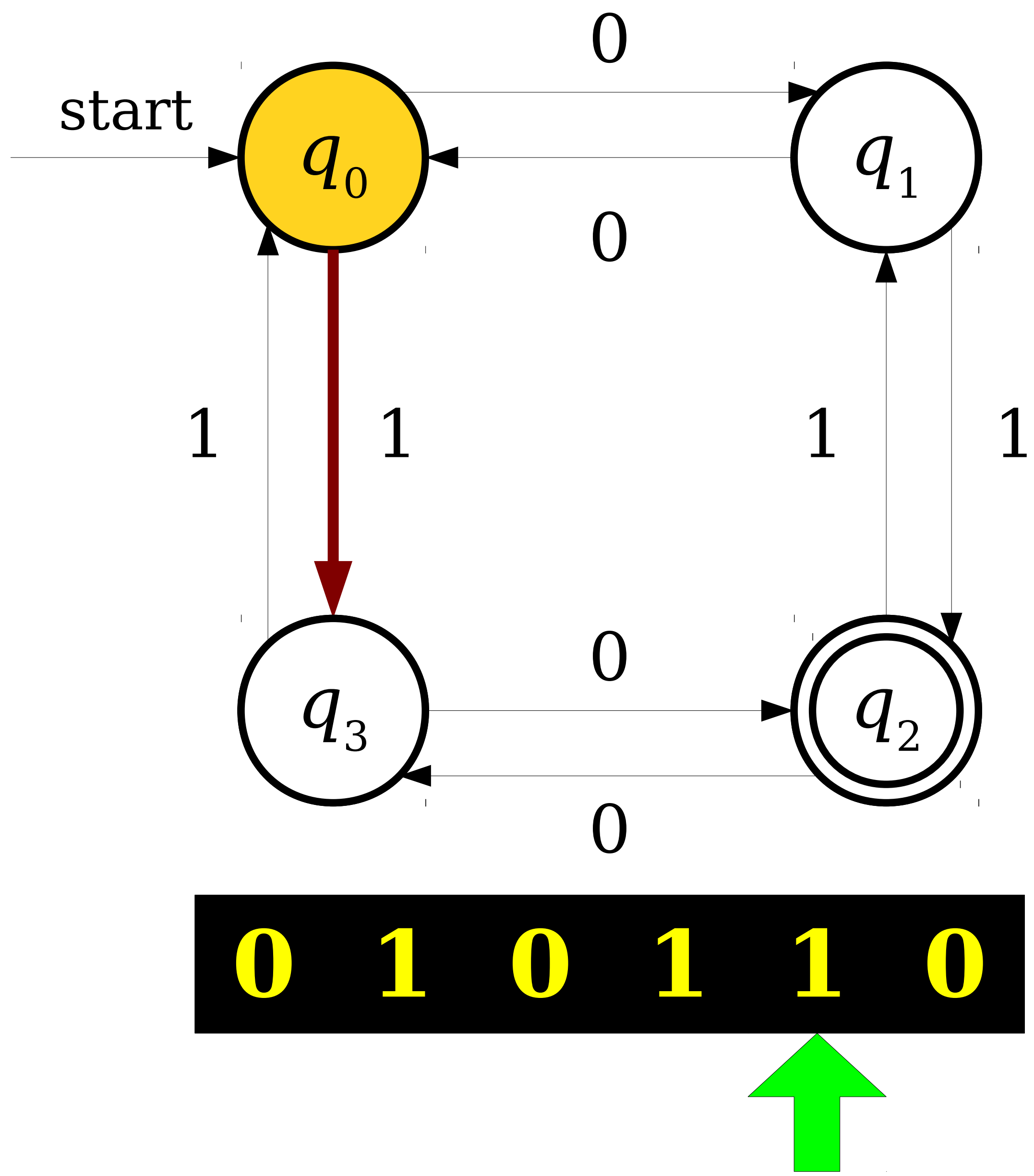
A Simple Finite Automaton



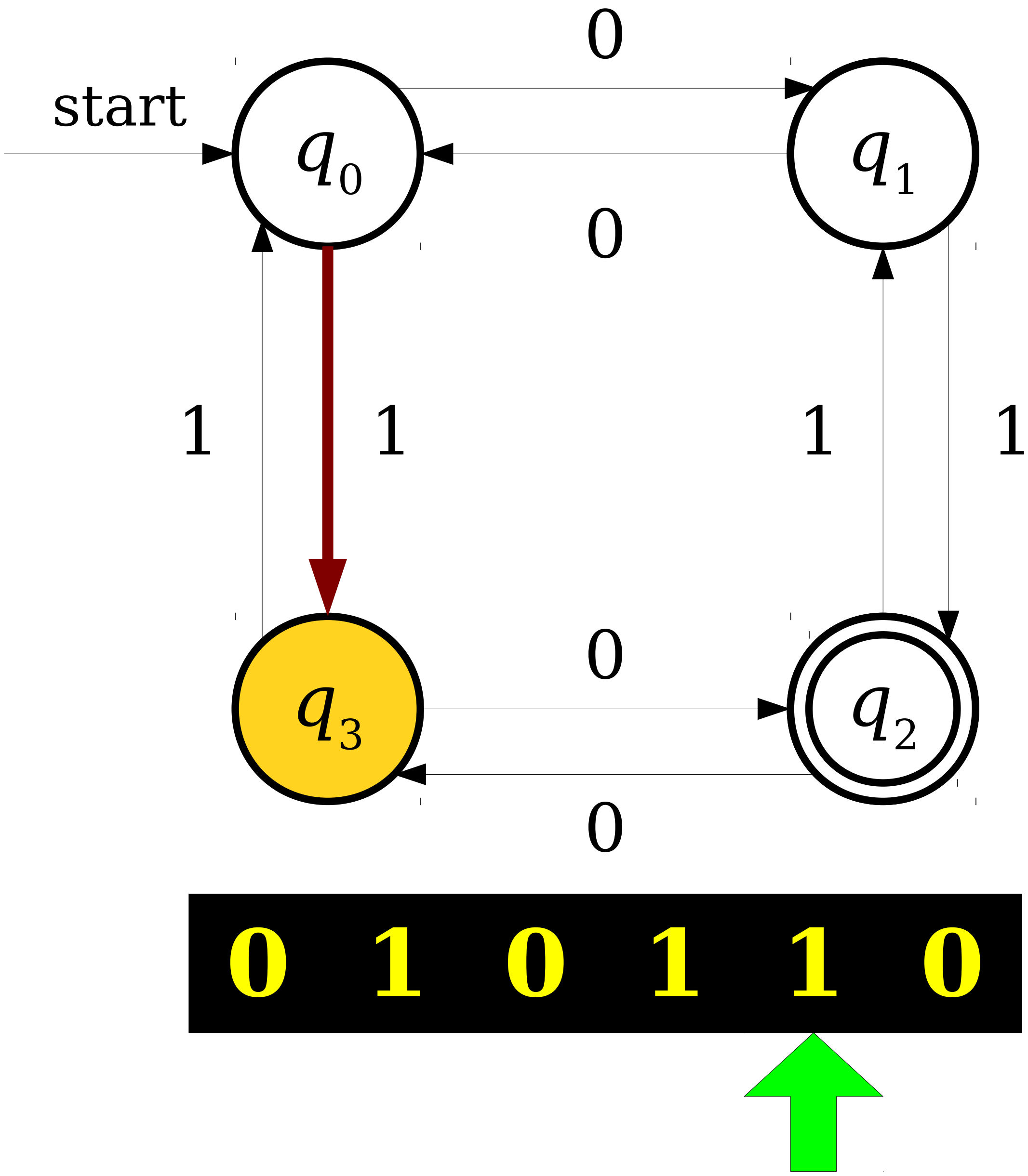
A Simple Finite Automaton



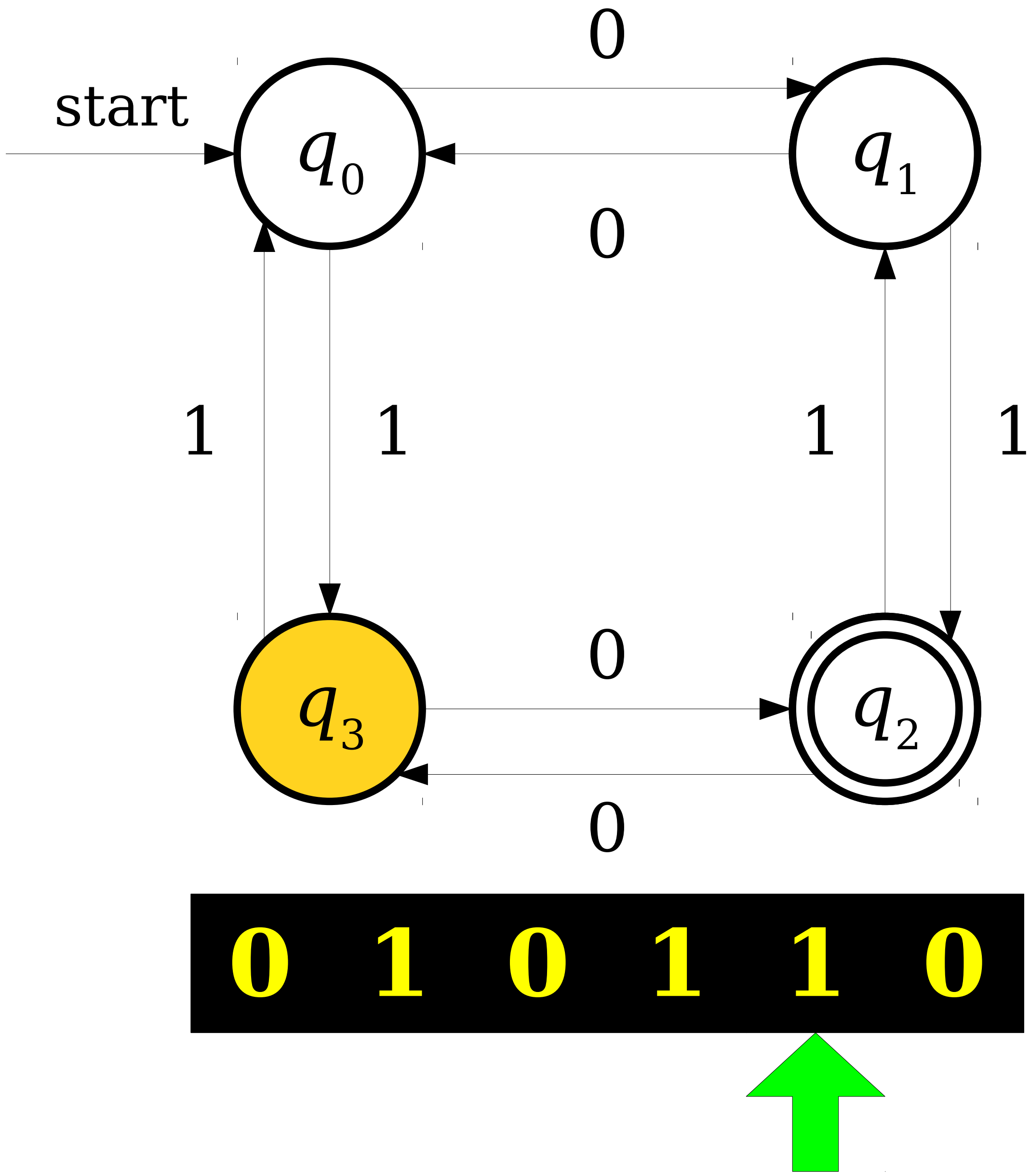
A Simple Finite Automaton



A Simple Finite Automaton

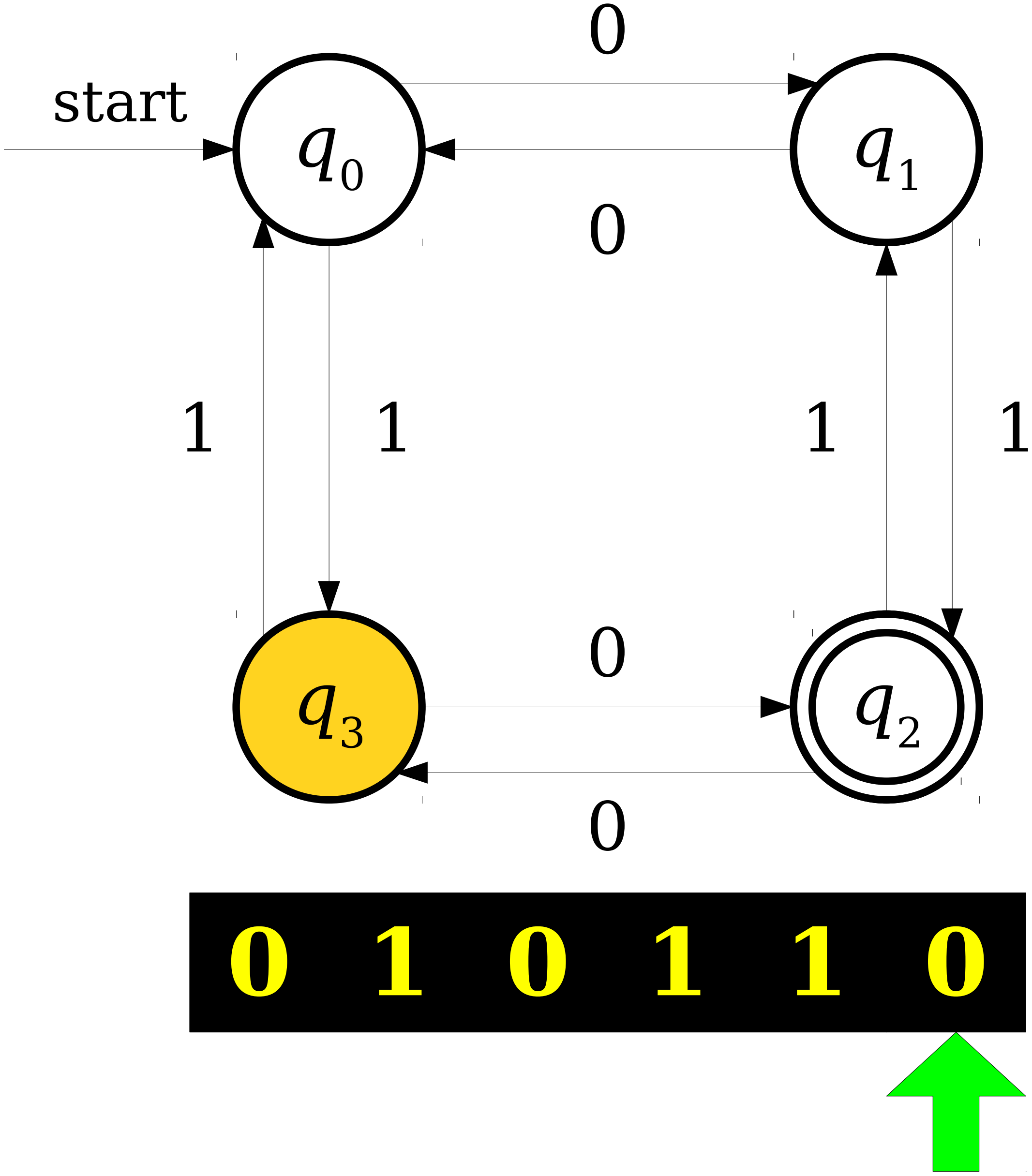


A Simple Finite Automaton

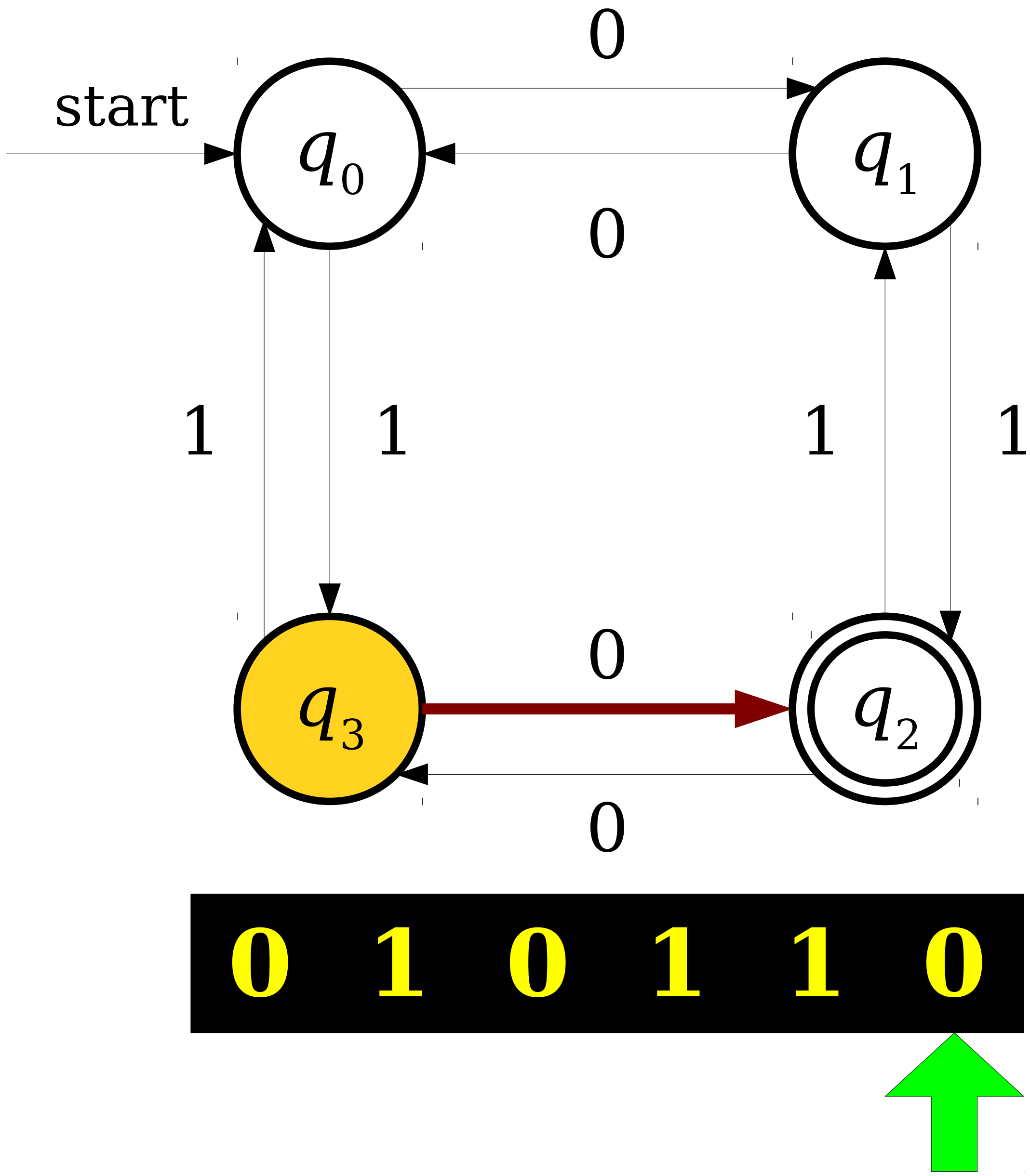


0 1 0 1 1 0

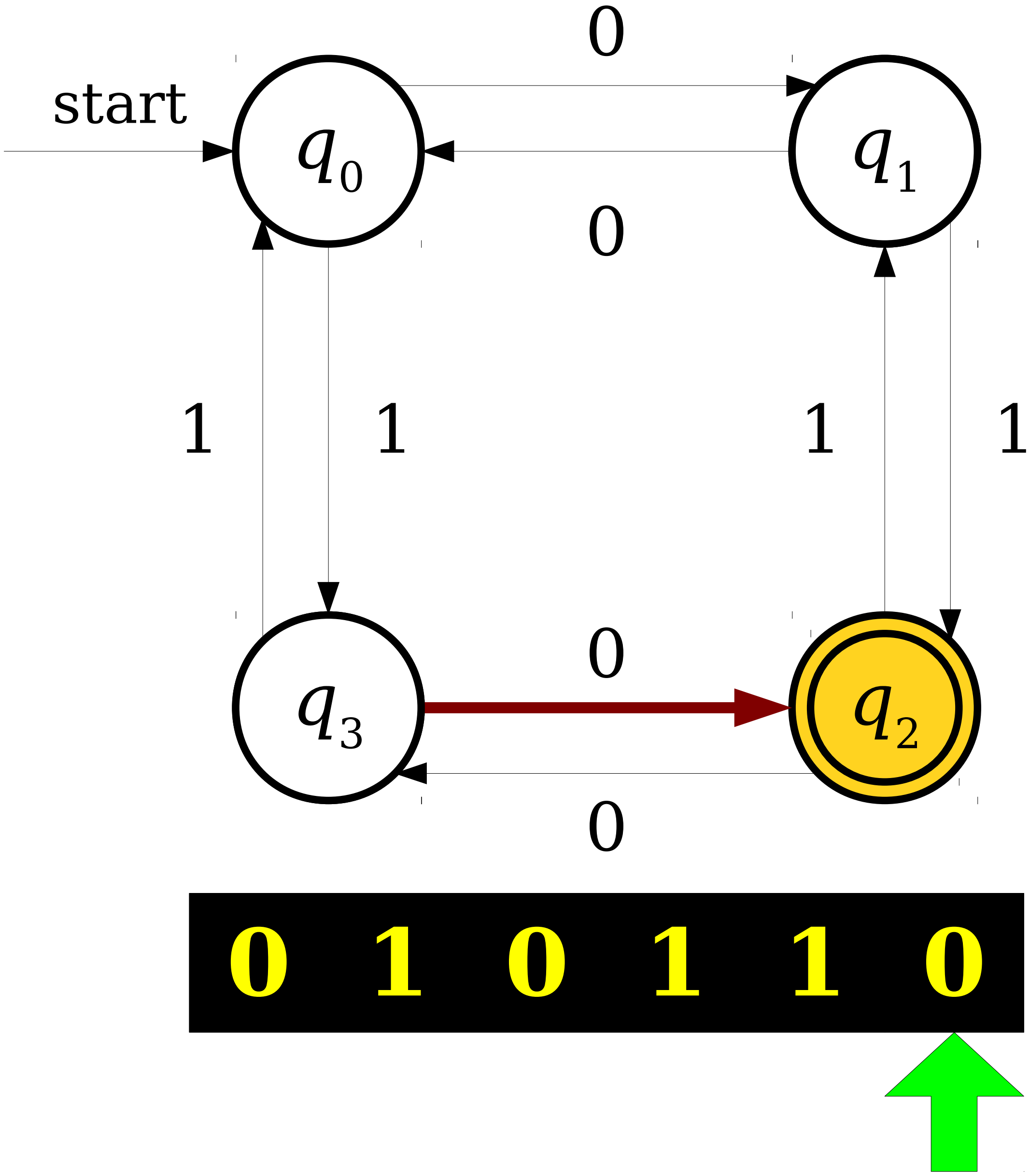
A Simple Finite Automaton



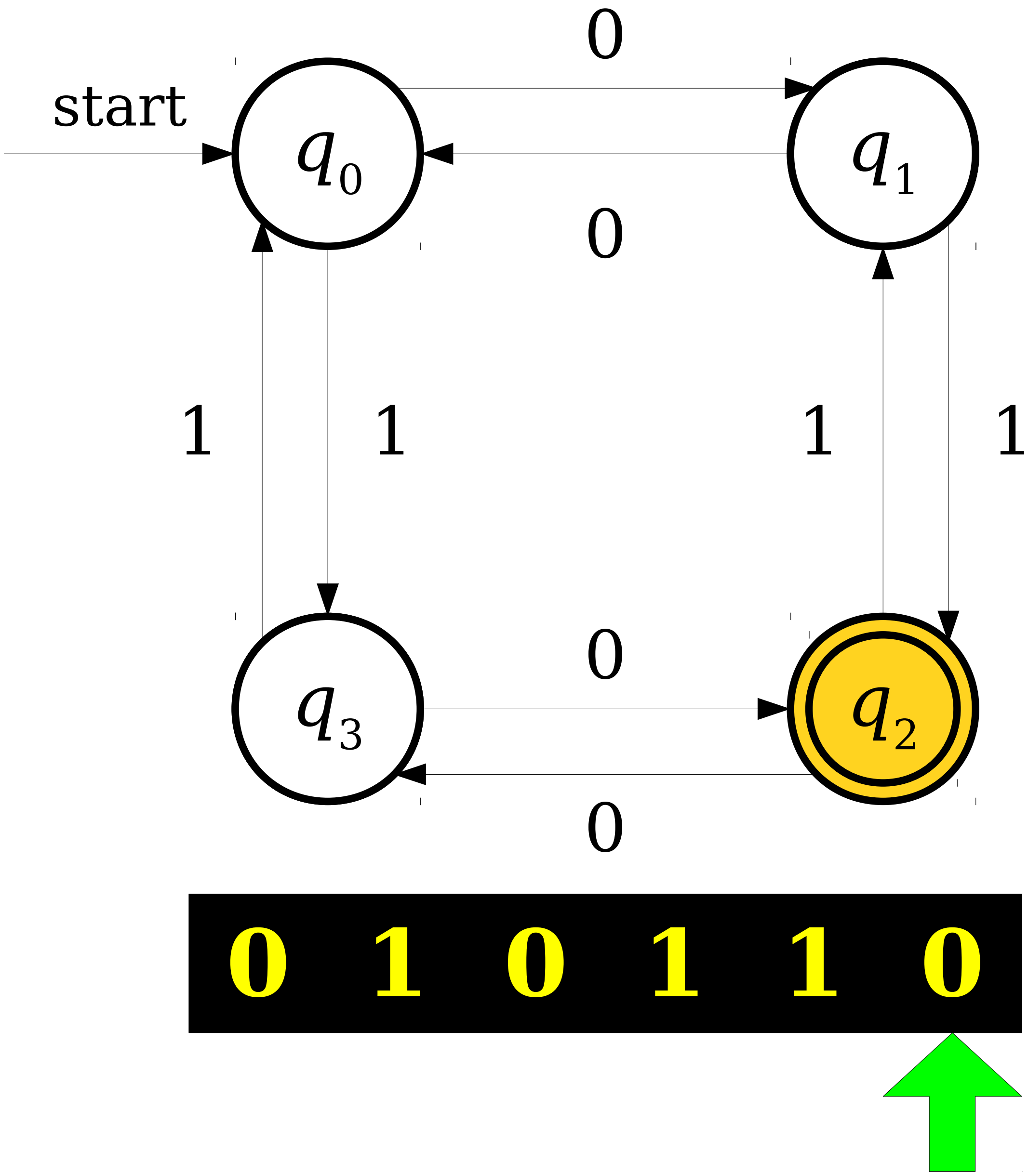
A Simple Finite Automaton



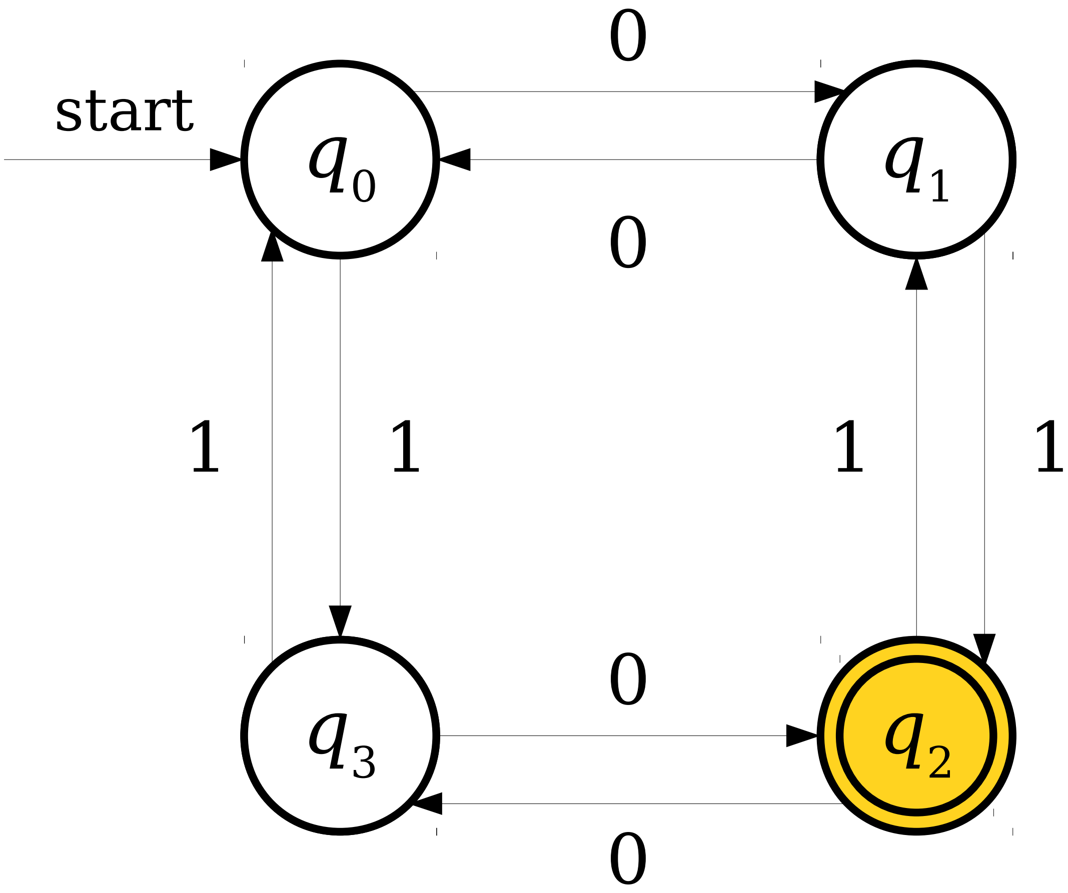
A Simple Finite Automaton



A Simple Finite Automaton

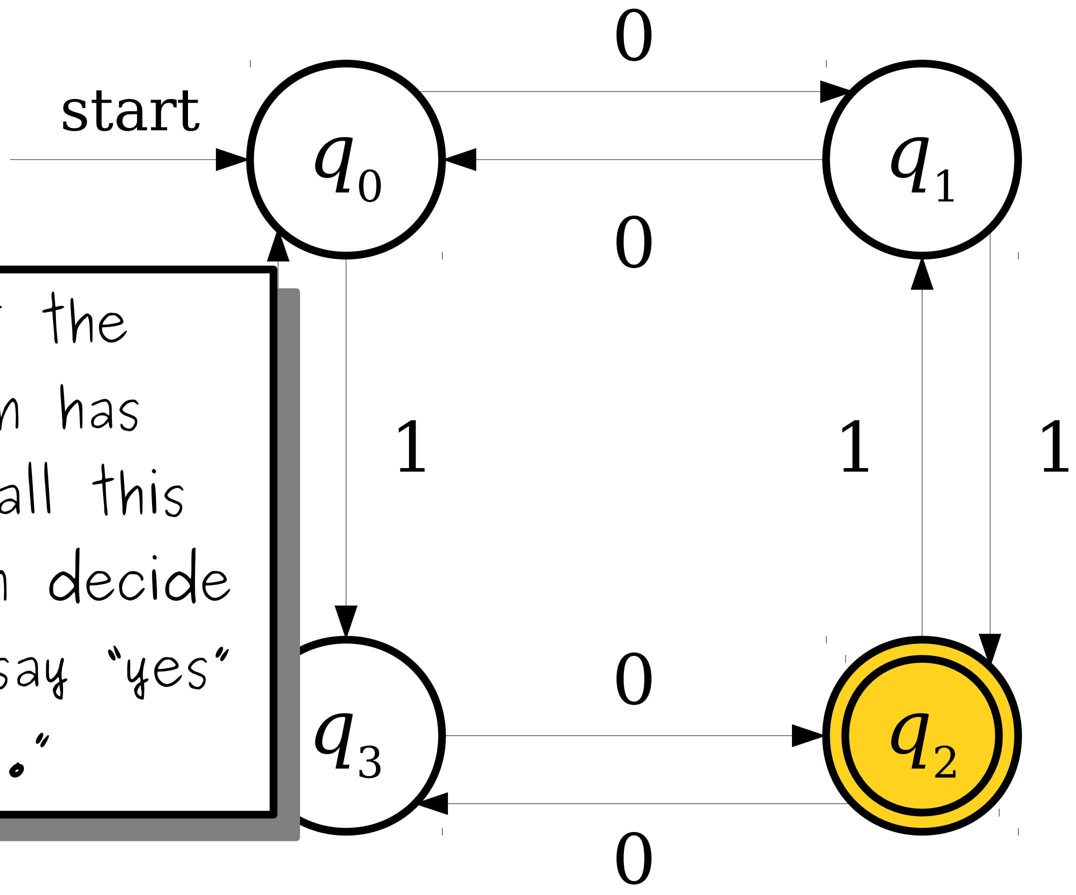


A Simple Finite Automaton



0 1 0 1 1 0

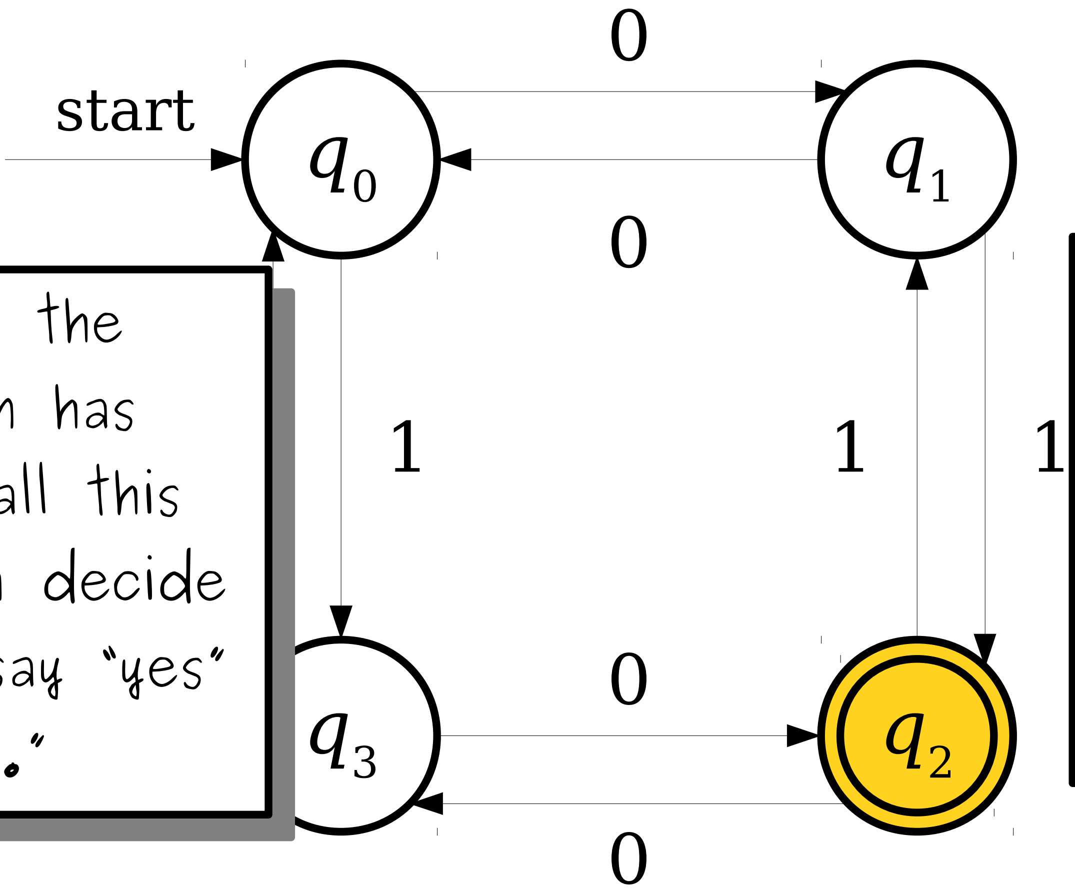
A Simple Finite Automaton



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

0 1 0 1 1 0

A Simple Finite Automaton

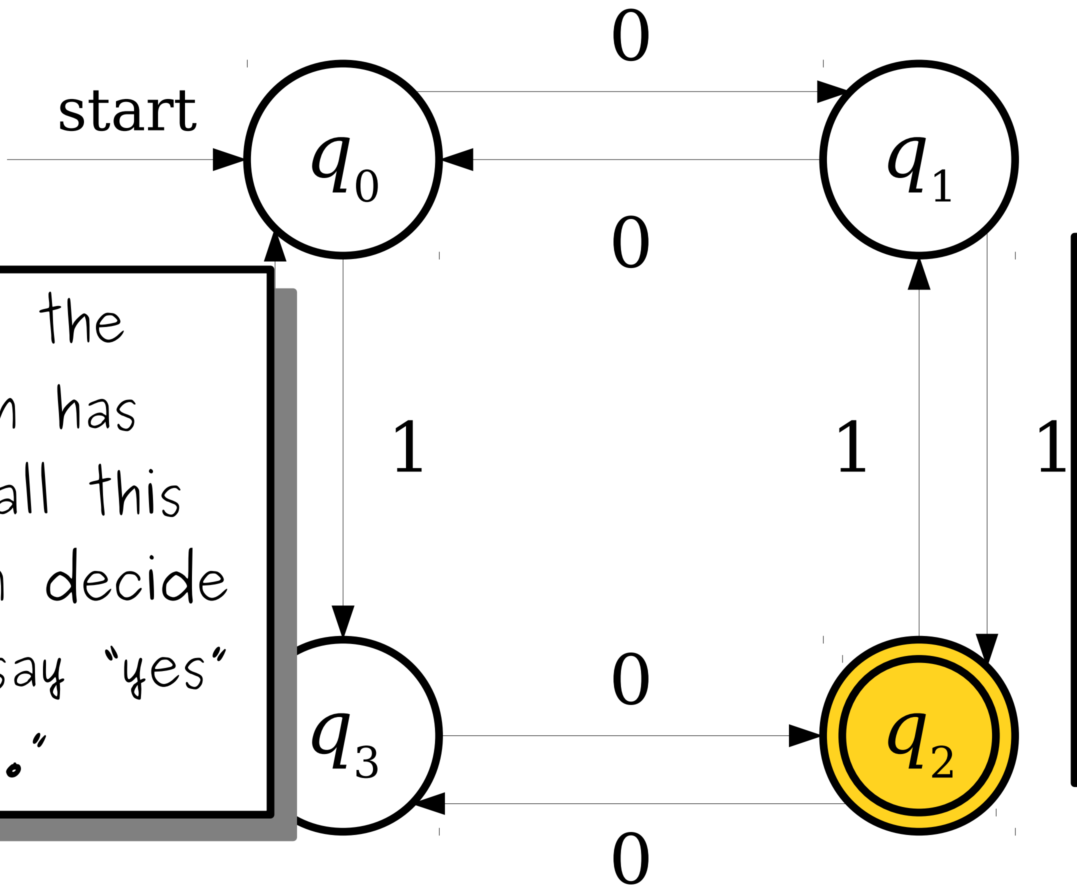


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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



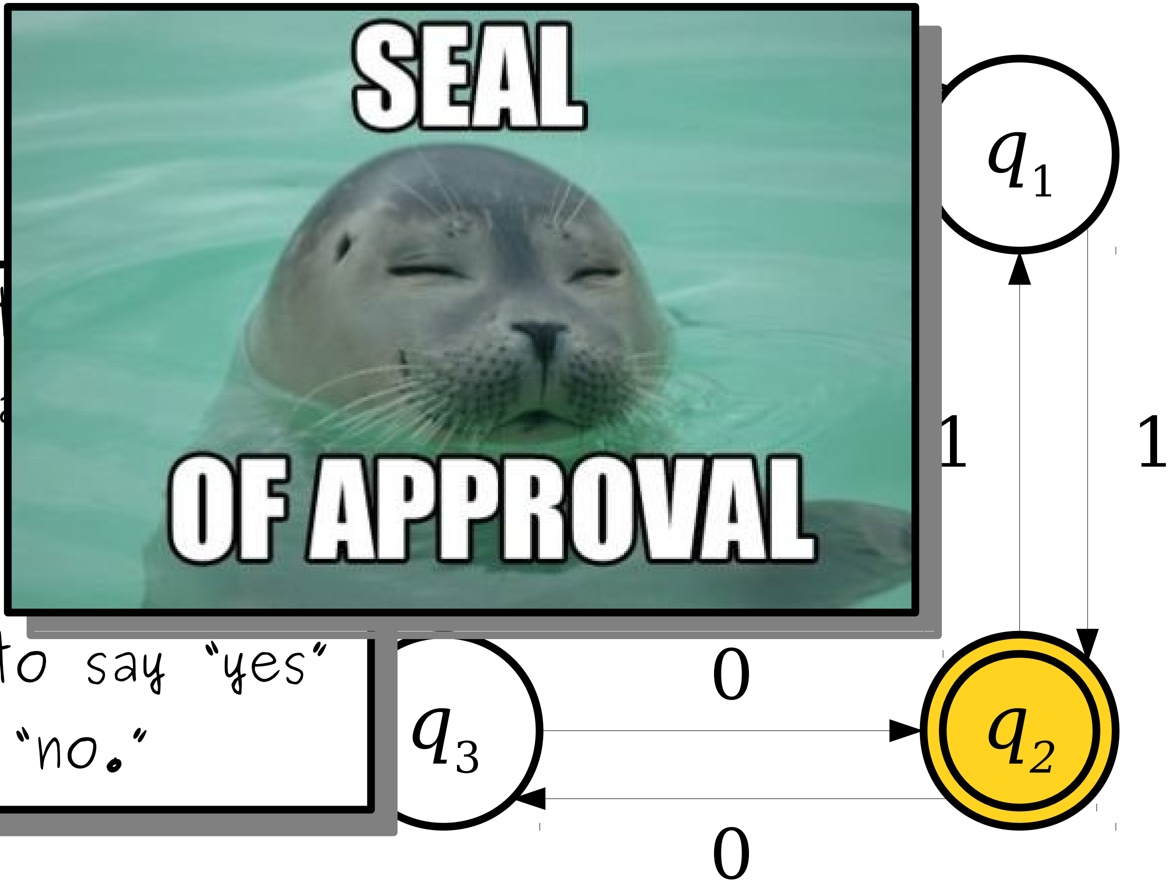
Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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

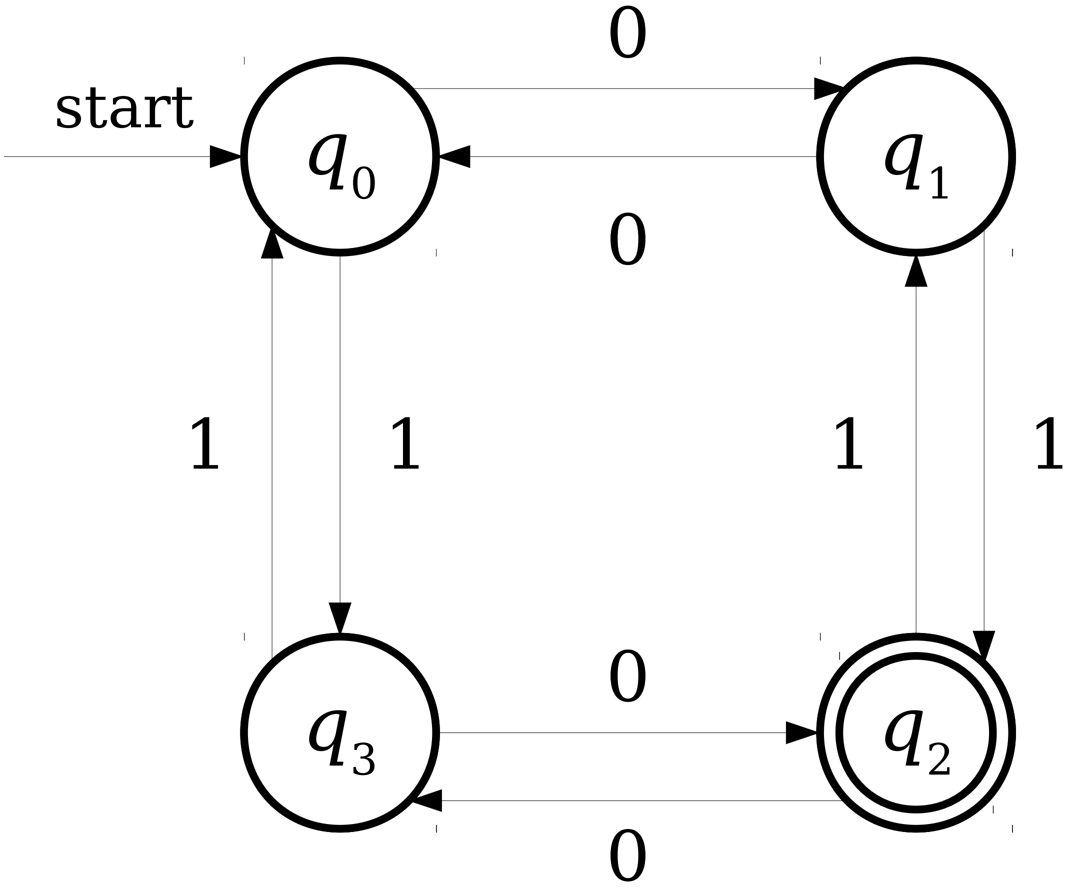
Now t
automa
looked
input, it
whether to say "yes"
or "no."



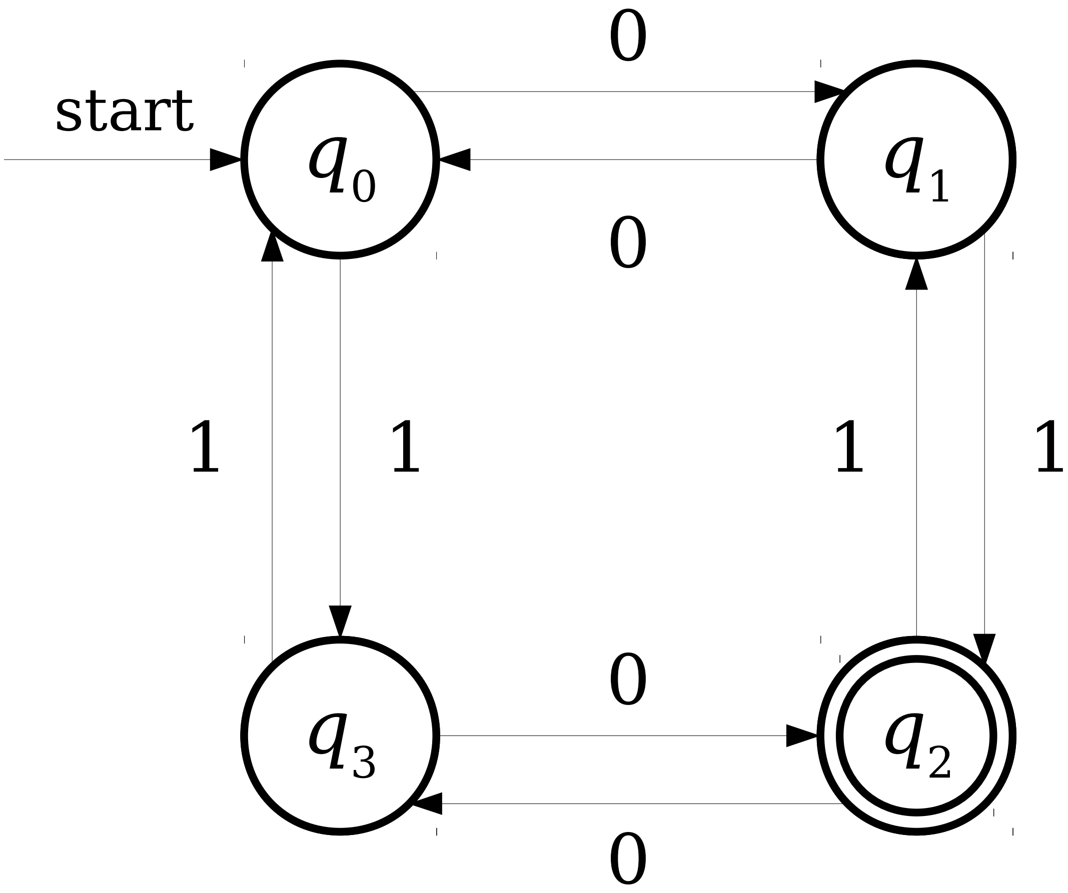
The double circle indicates that this state is an **accepting state**, so the automaton outputs "yes."

0 1 0 1 1 0

A Simple Finite Automaton

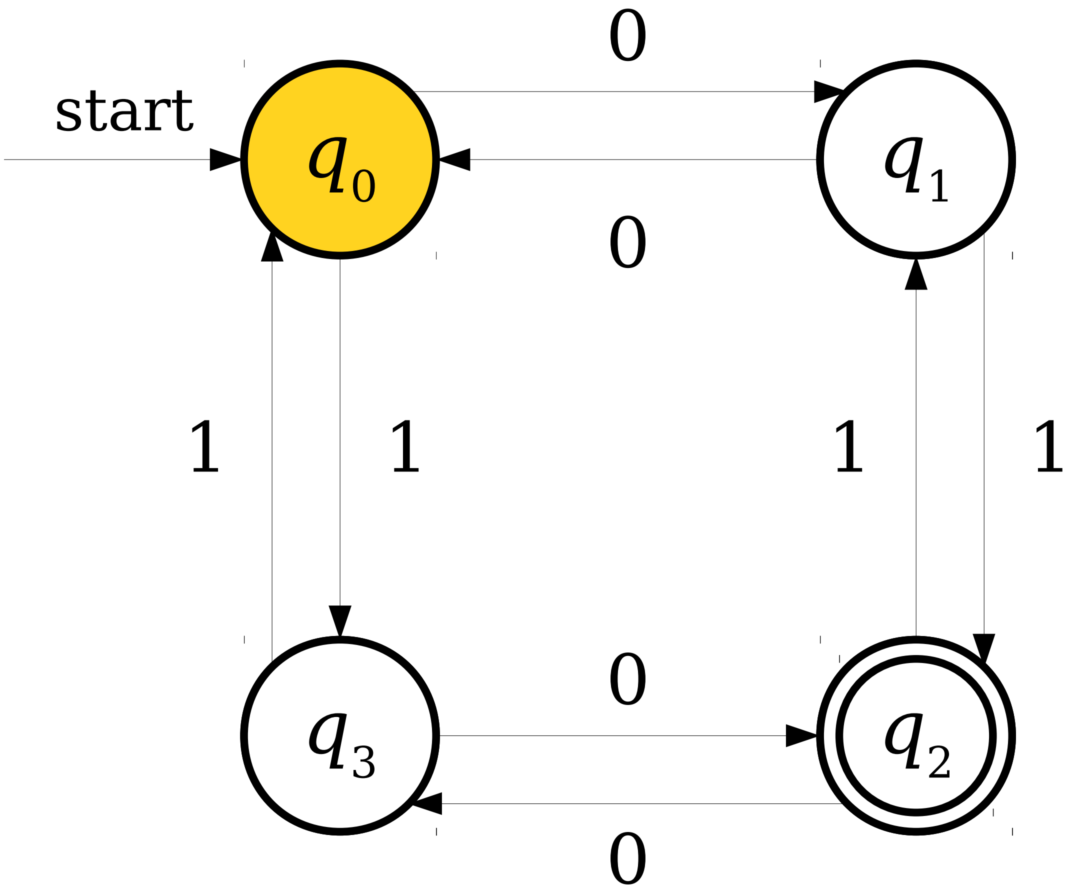


A Simple Finite Automaton



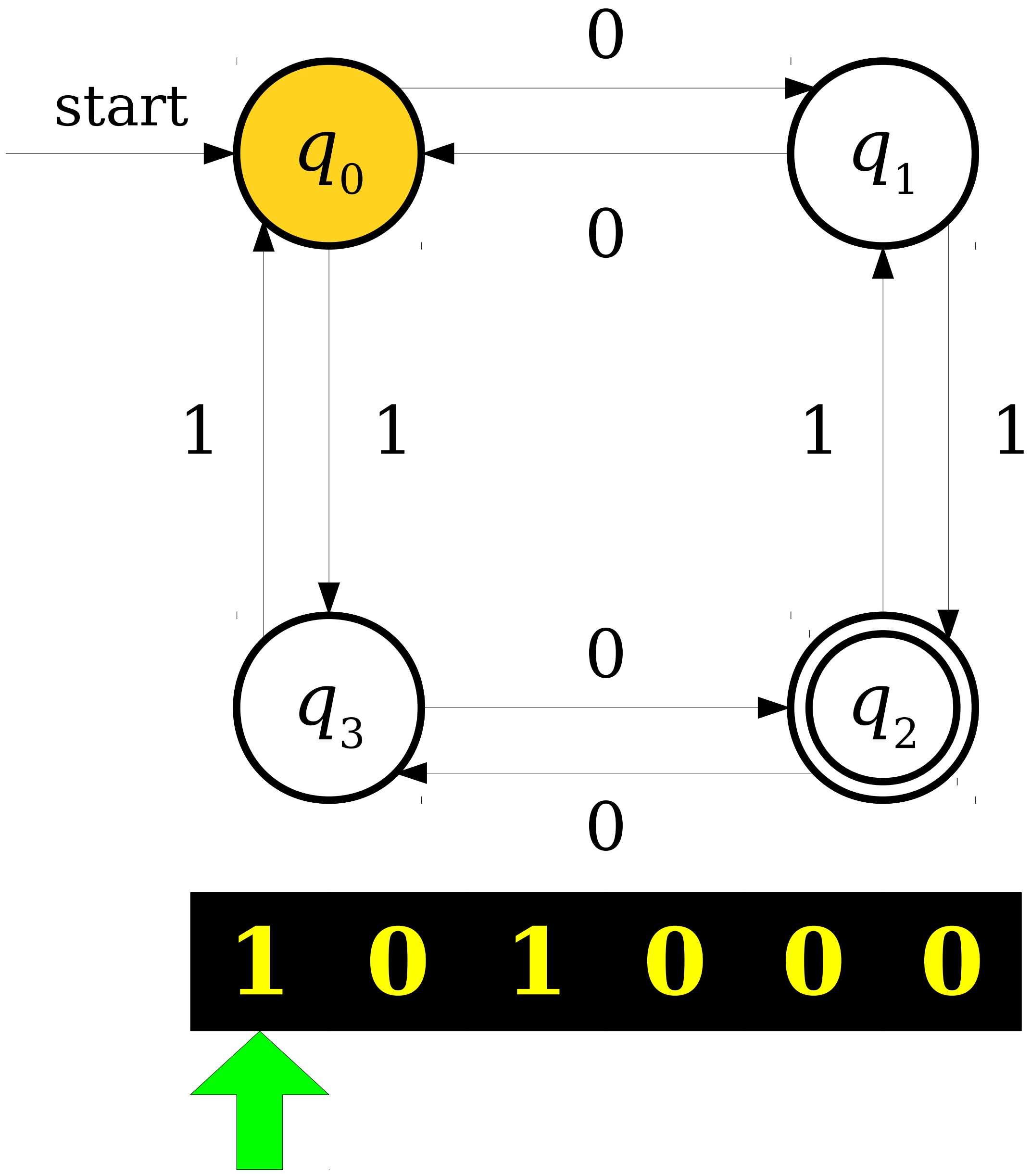
1 0 1 0 0 0

A Simple Finite Automaton

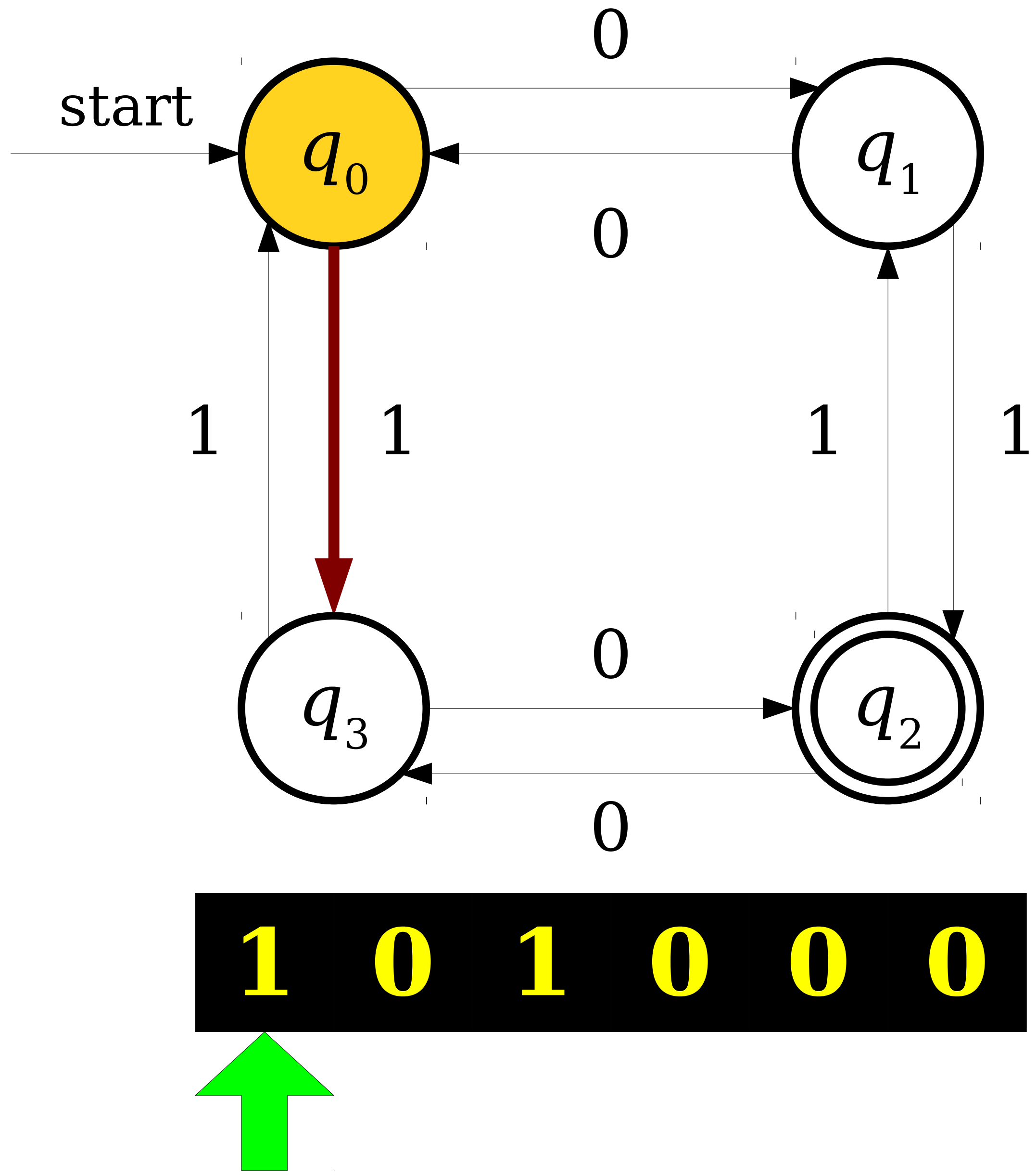


1 0 1 0 0 0

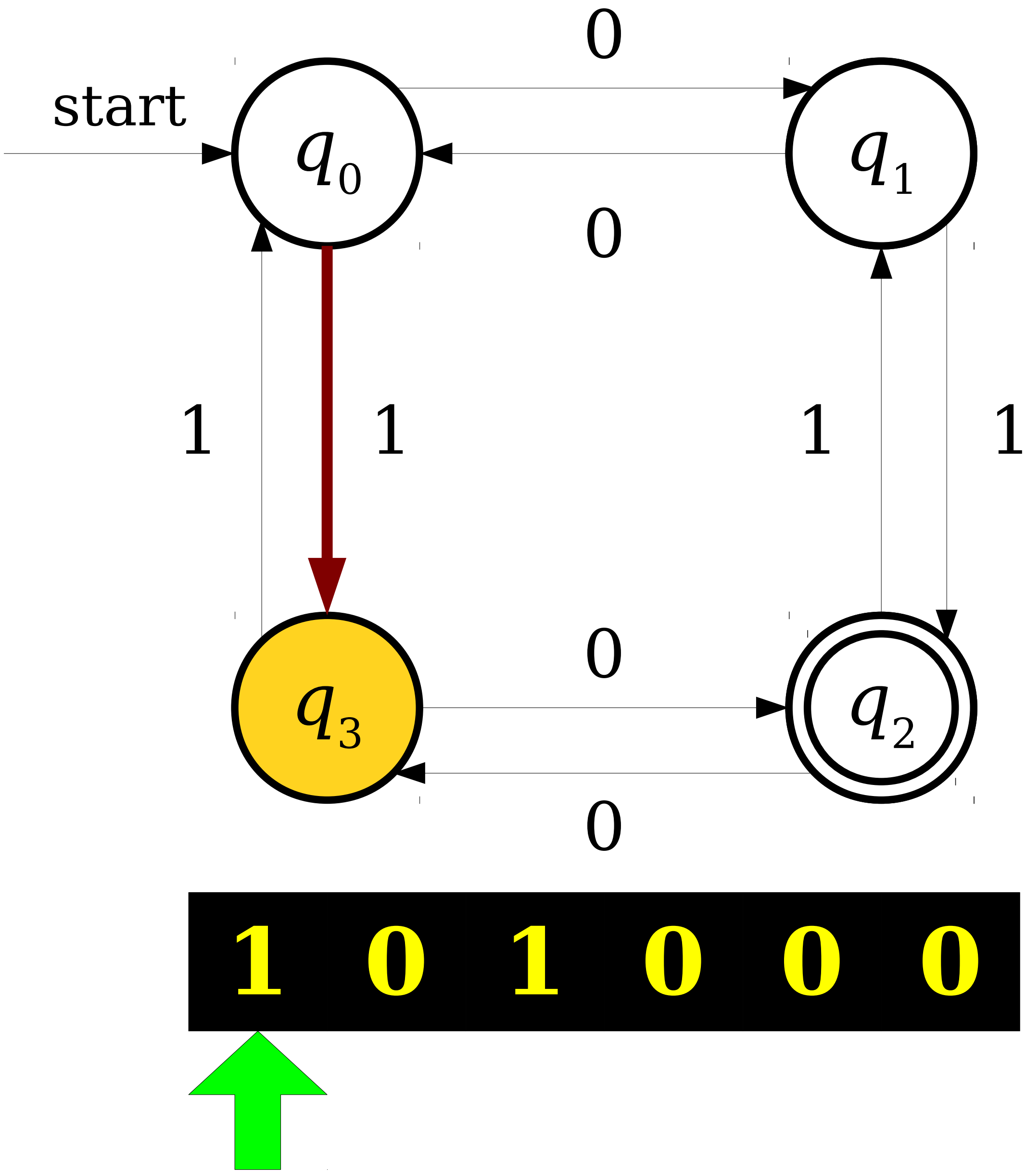
A Simple Finite Automaton



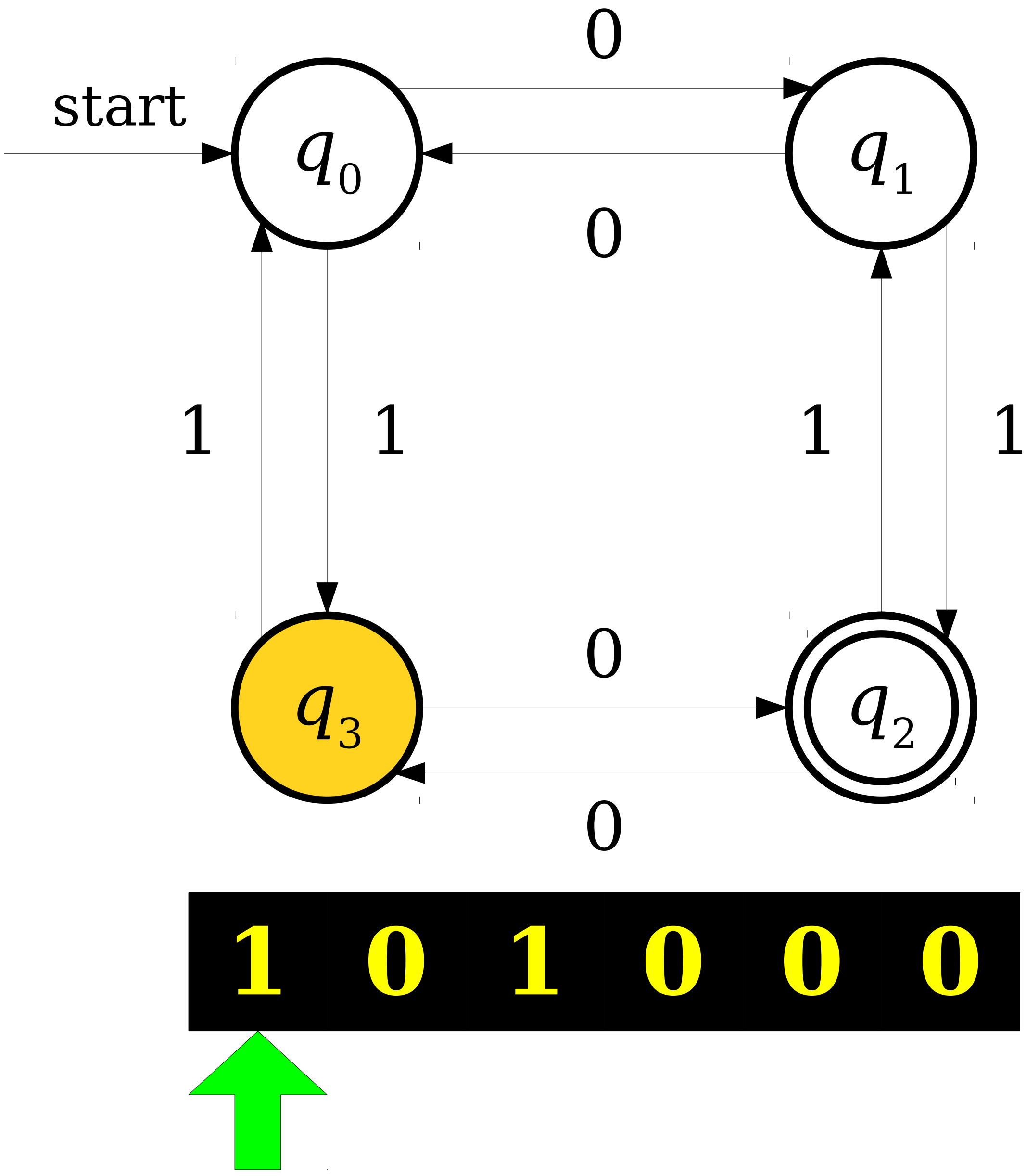
A Simple Finite Automaton



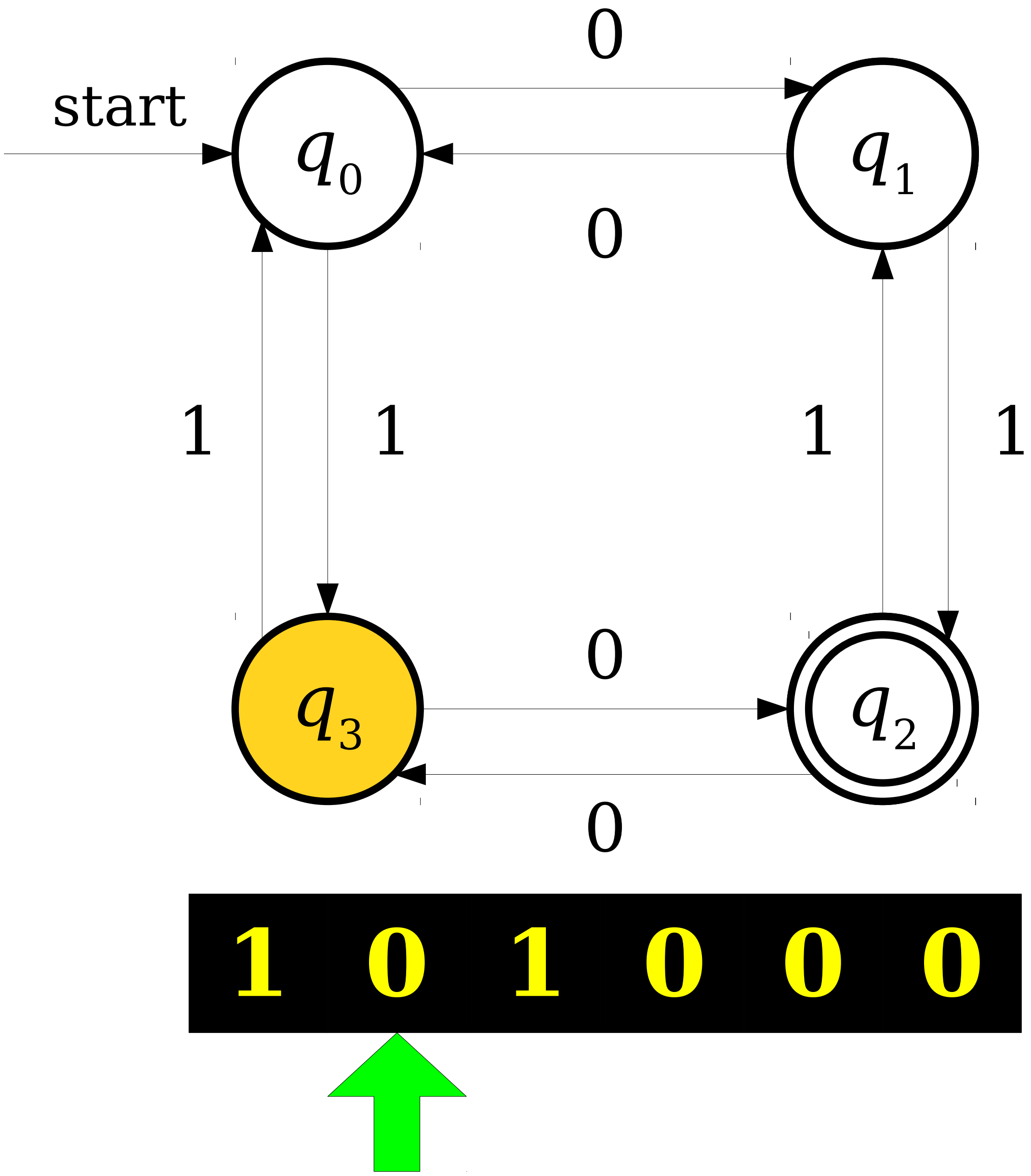
A Simple Finite Automaton



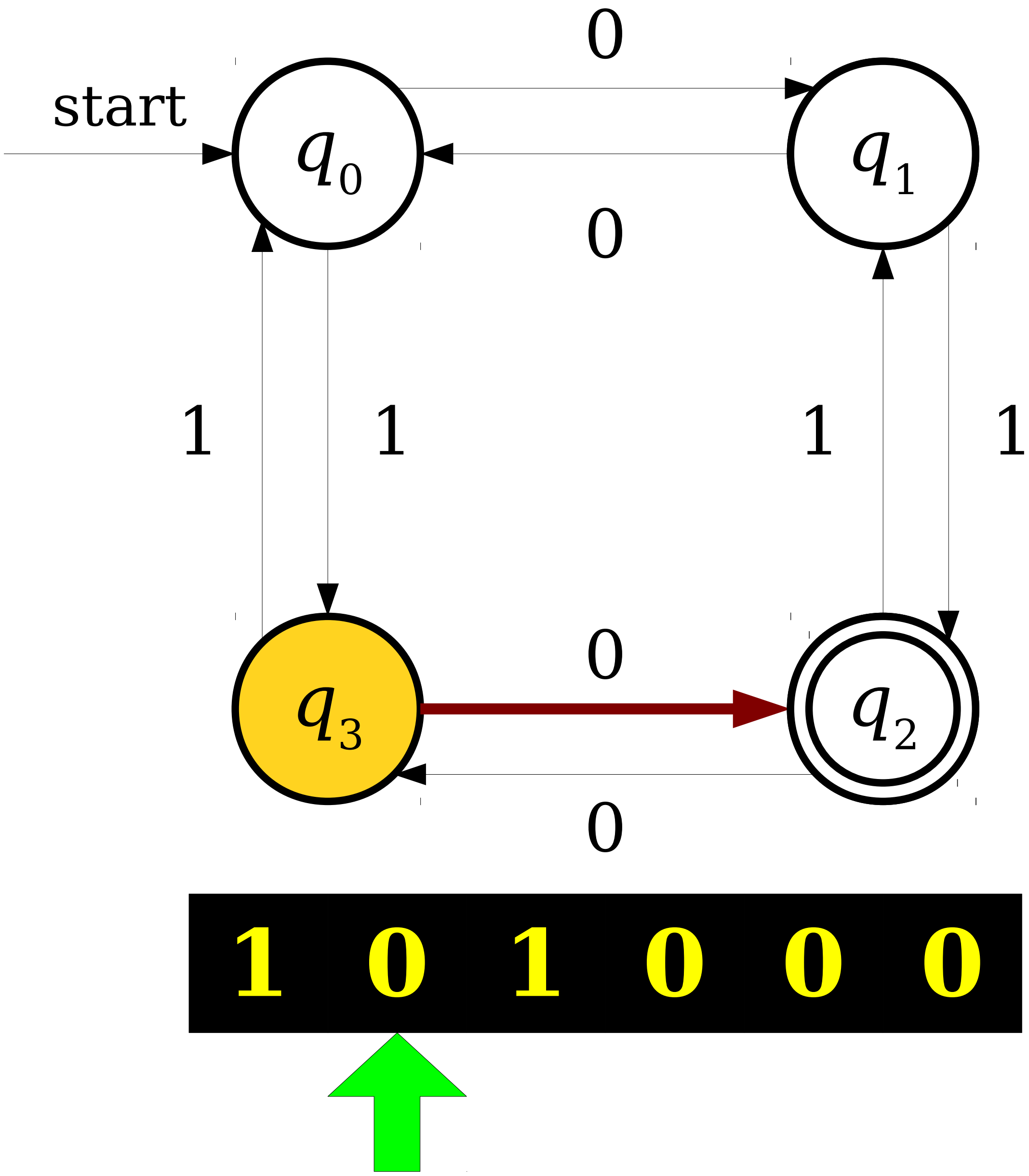
A Simple Finite Automaton



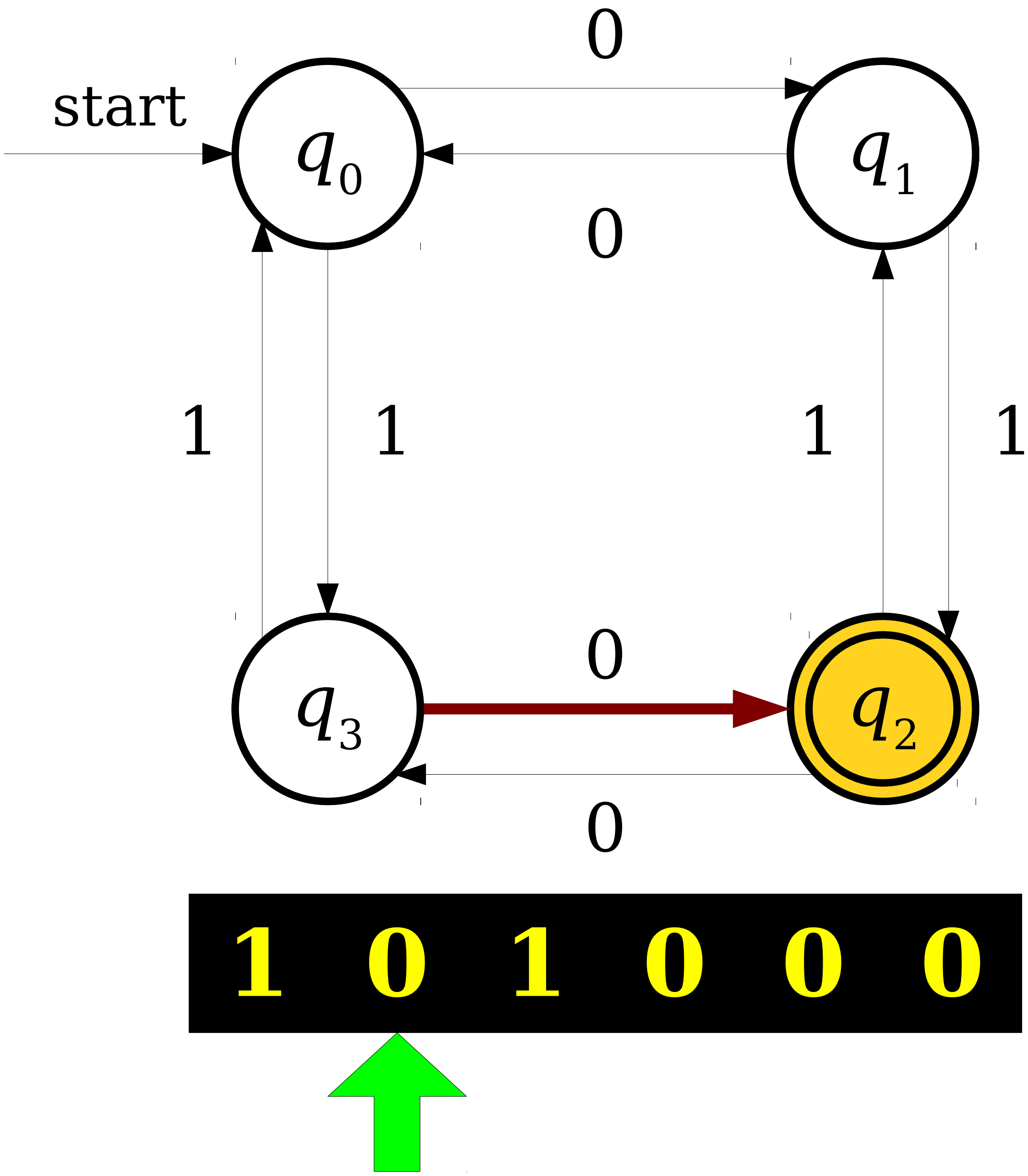
A Simple Finite Automaton



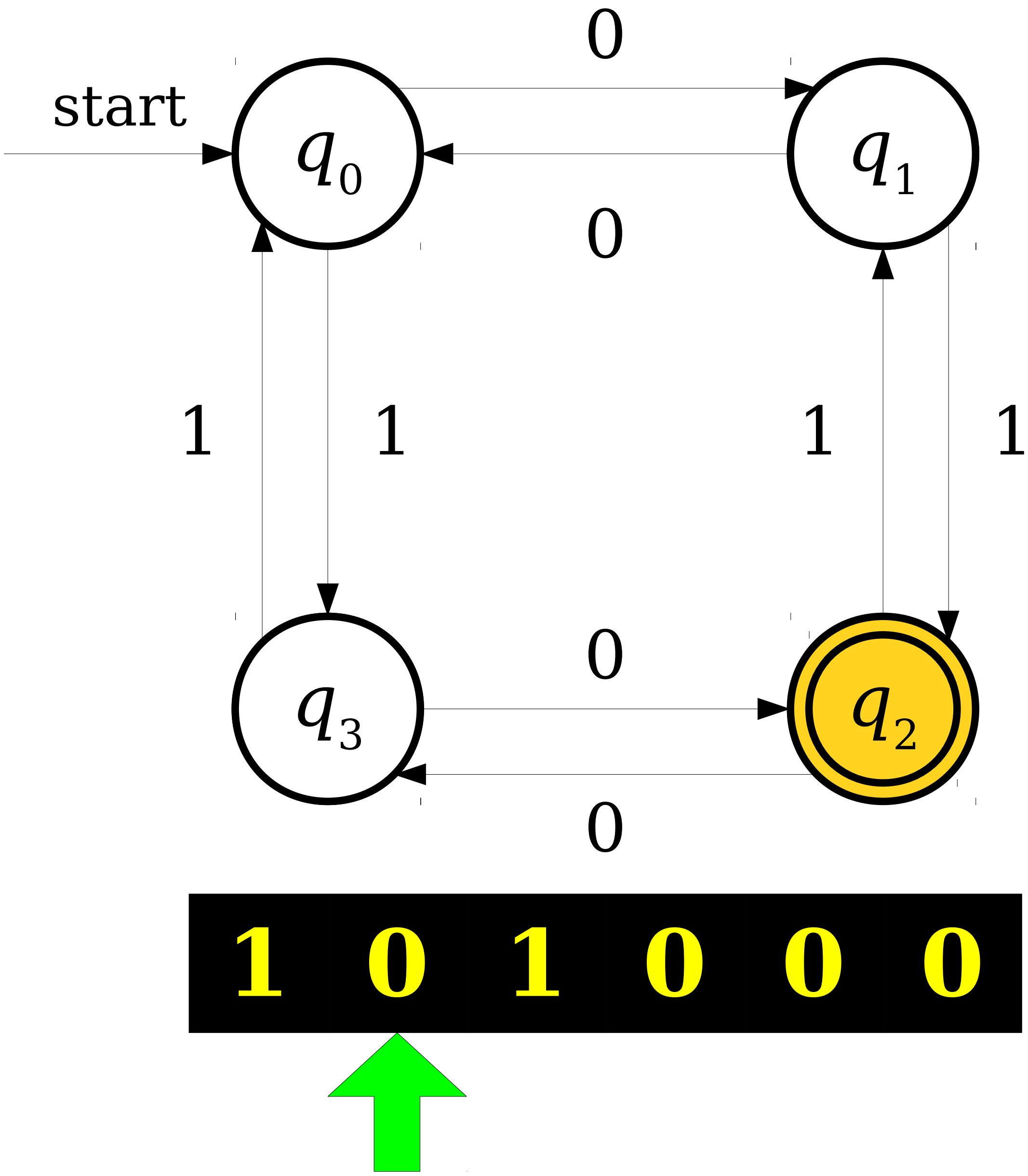
A Simple Finite Automaton



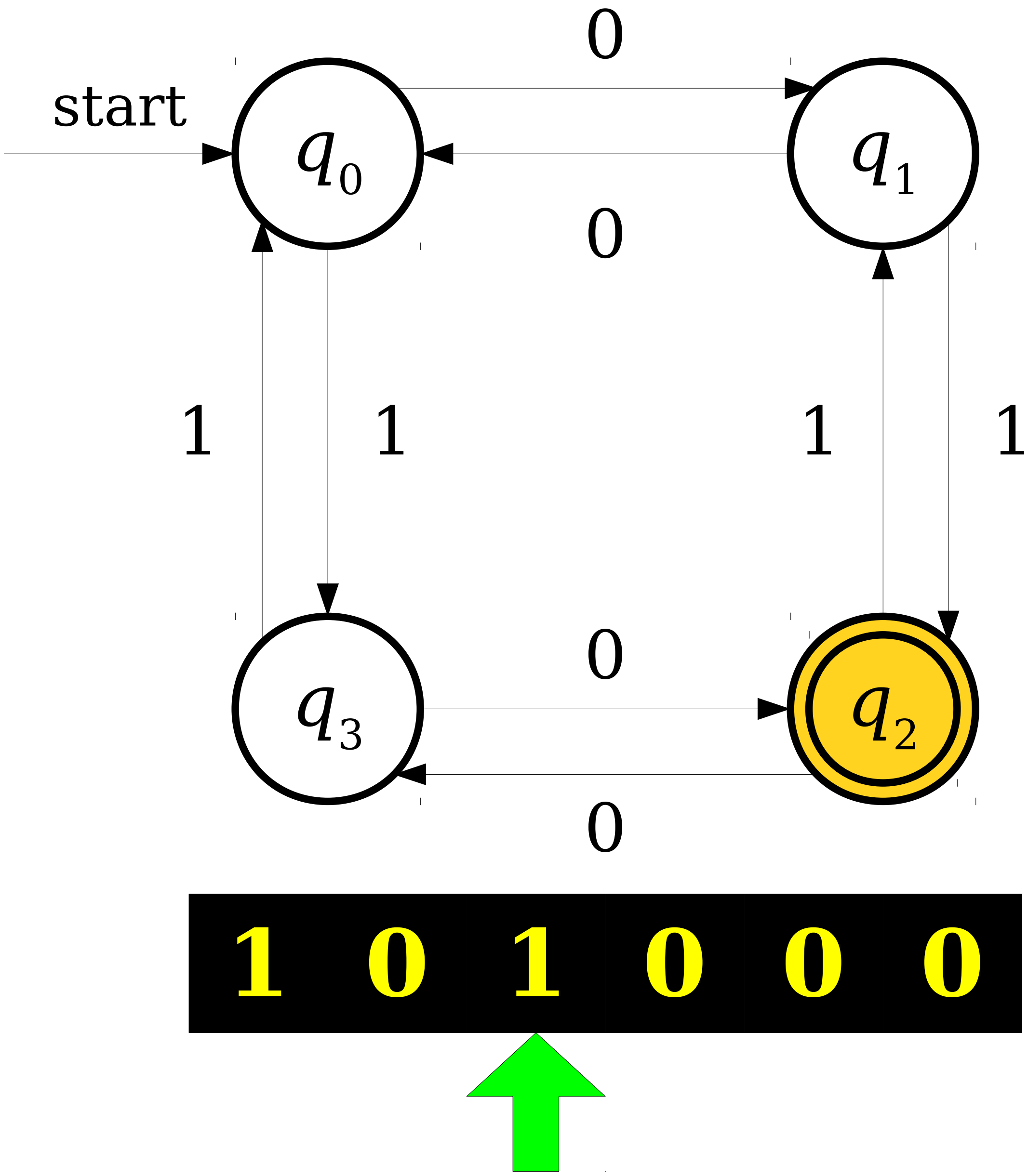
A Simple Finite Automaton



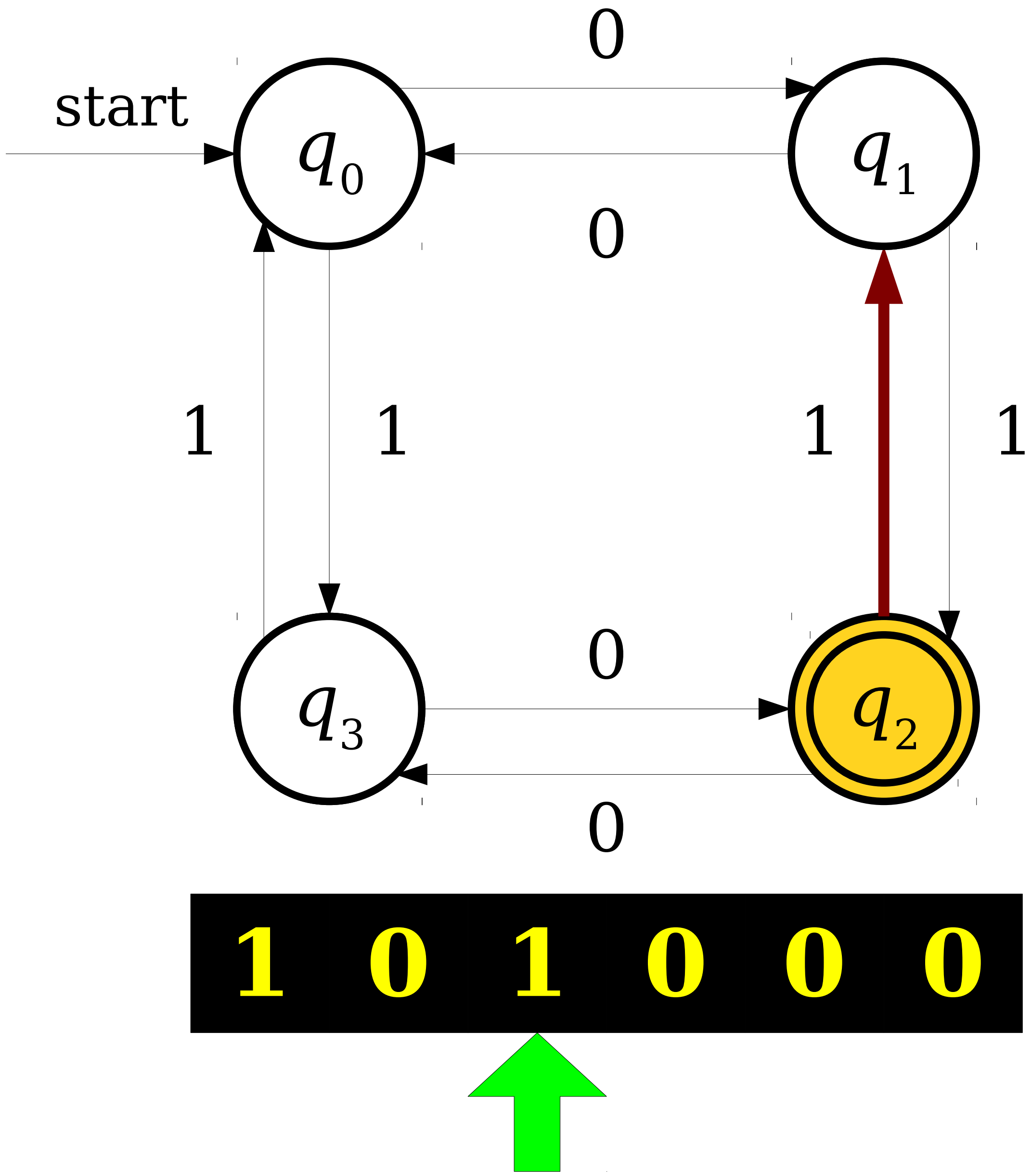
A Simple Finite Automaton



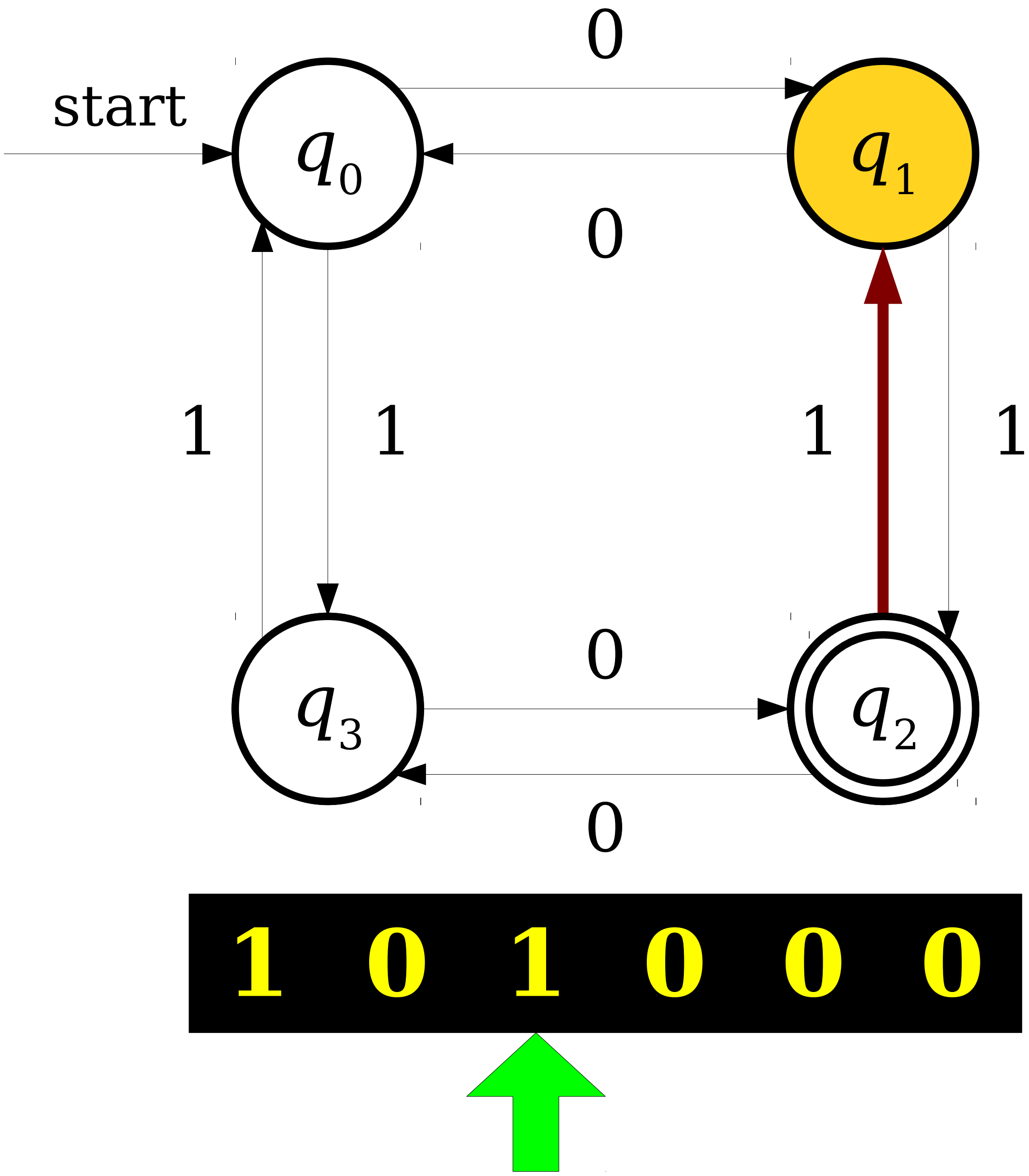
A Simple Finite Automaton



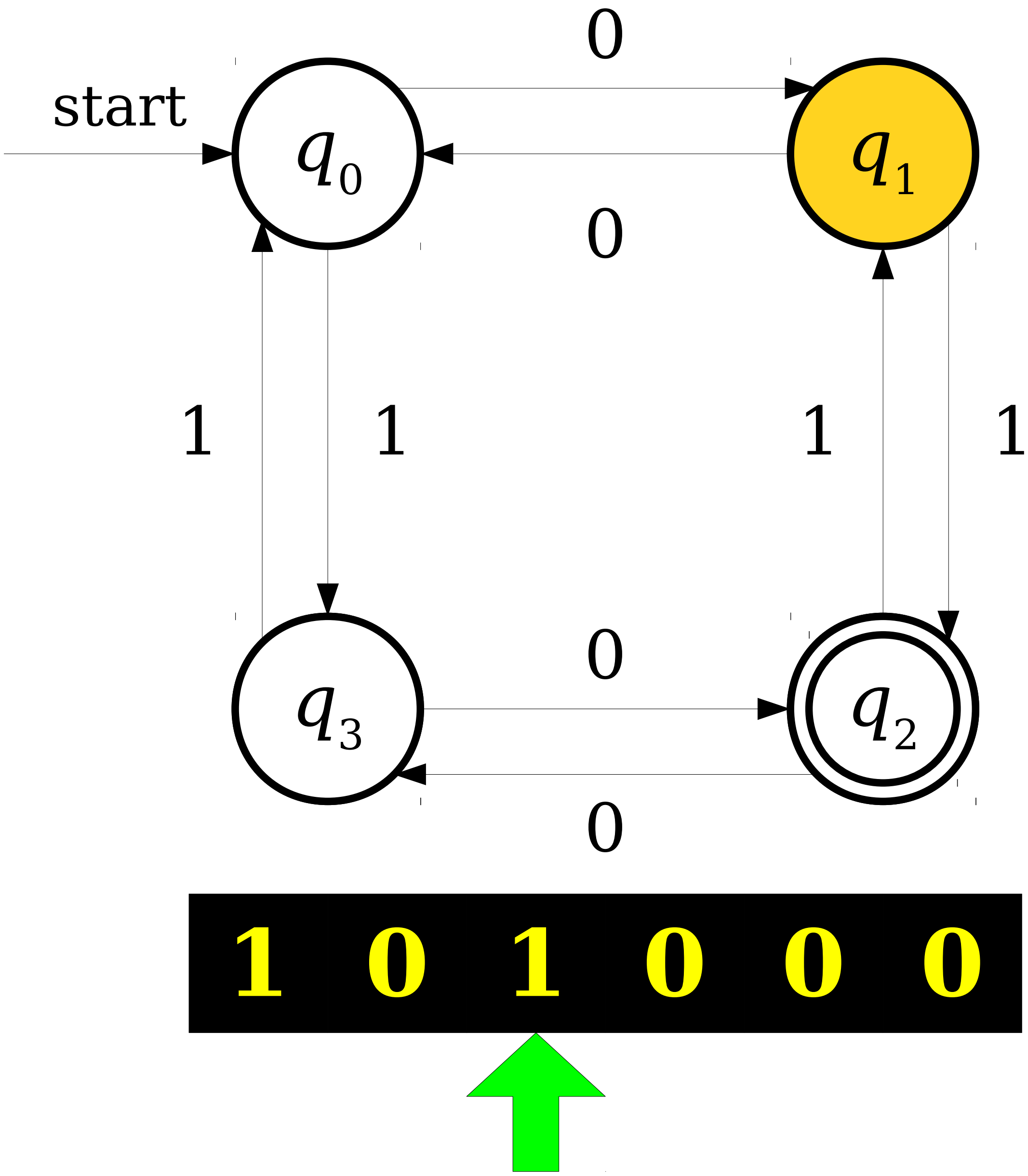
A Simple Finite Automaton



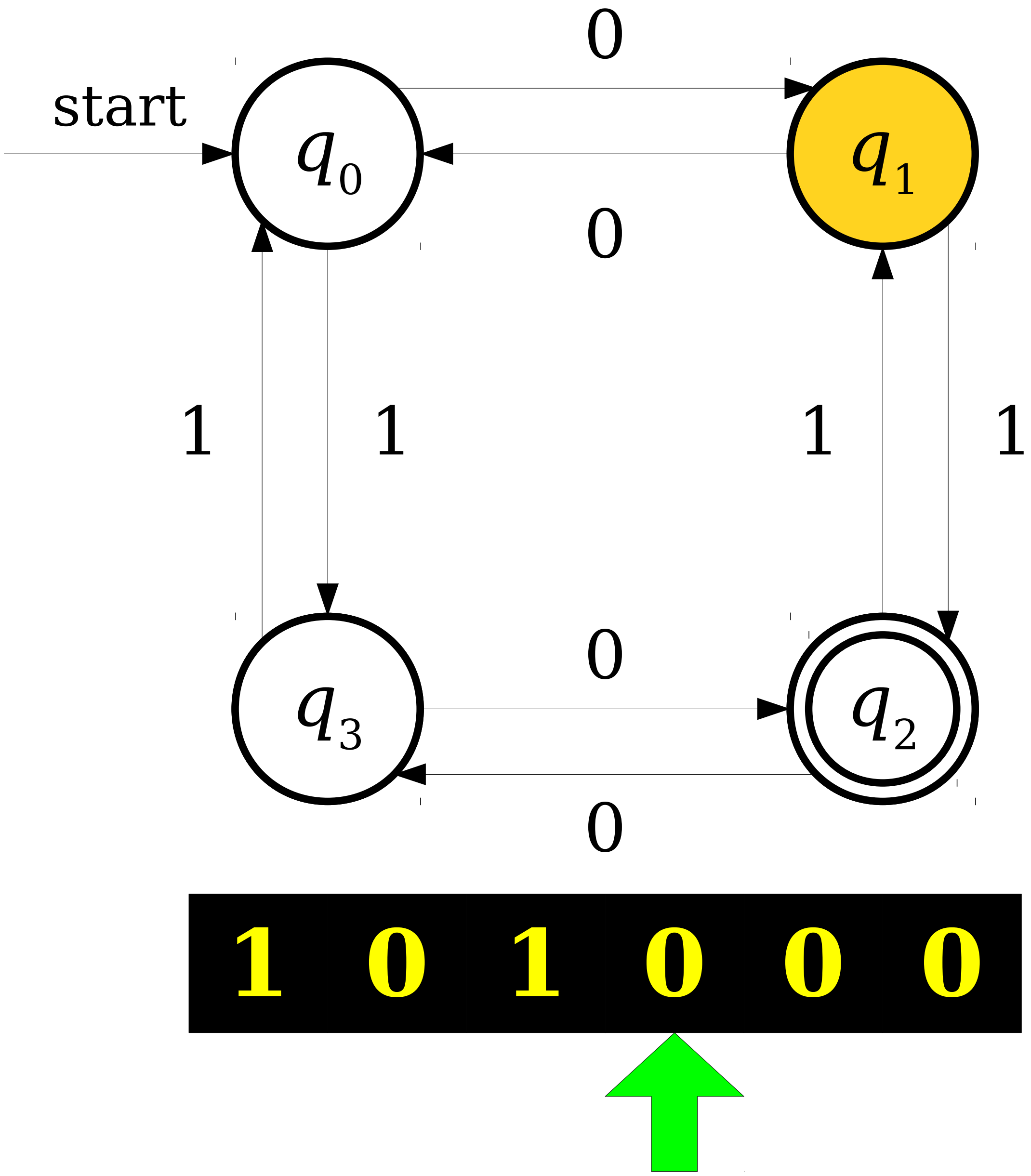
A Simple Finite Automaton



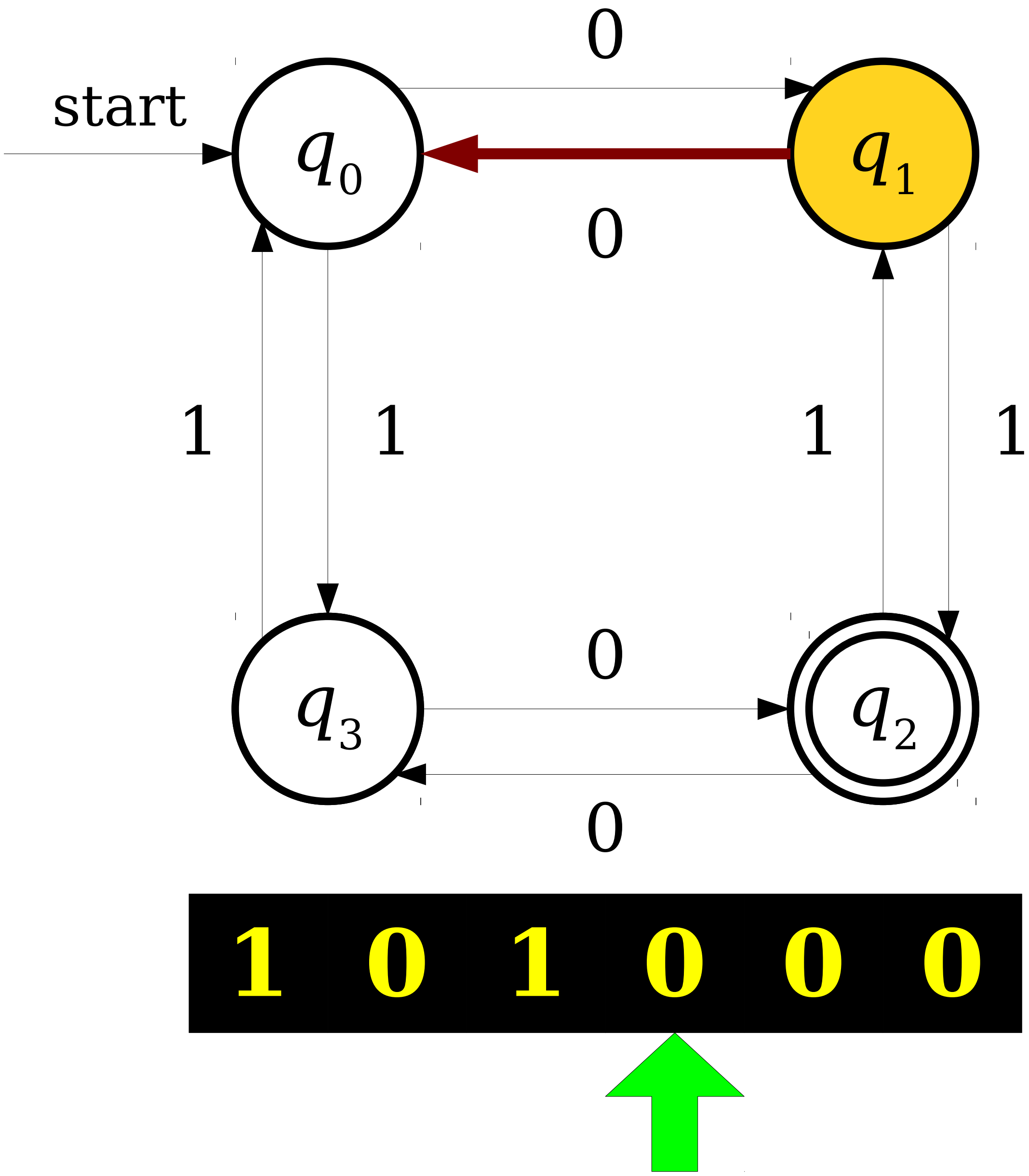
A Simple Finite Automaton



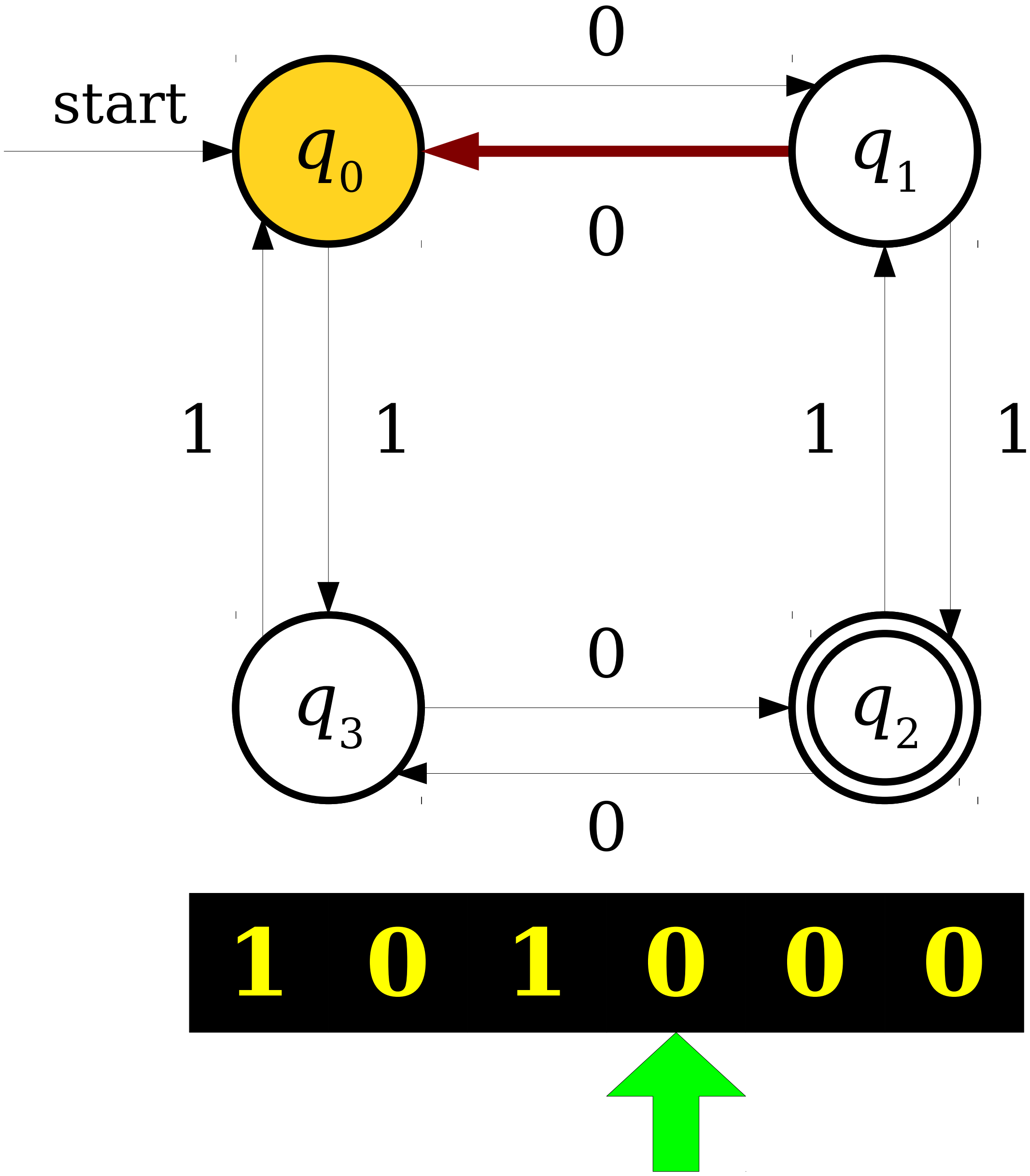
A Simple Finite Automaton



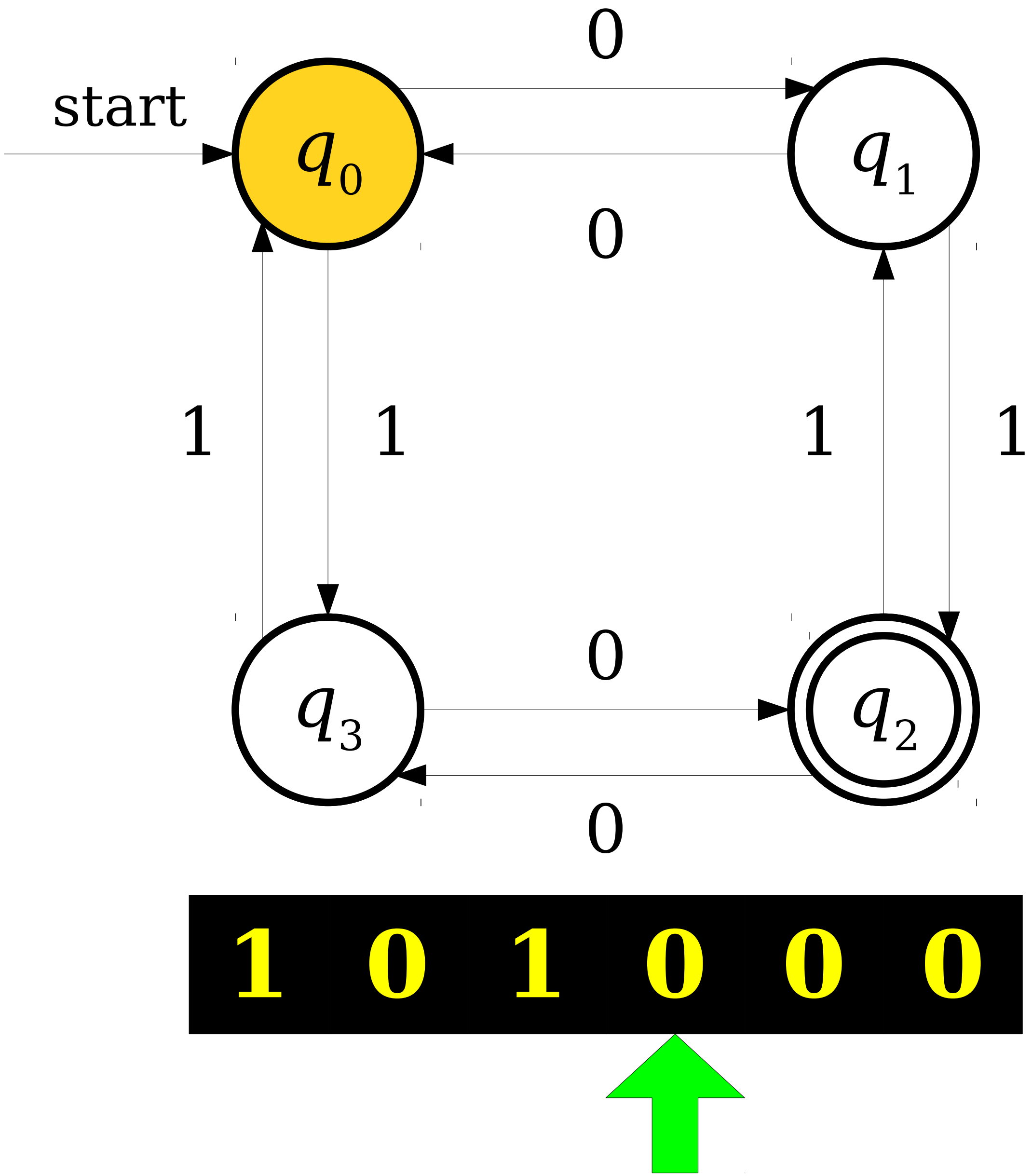
A Simple Finite Automaton



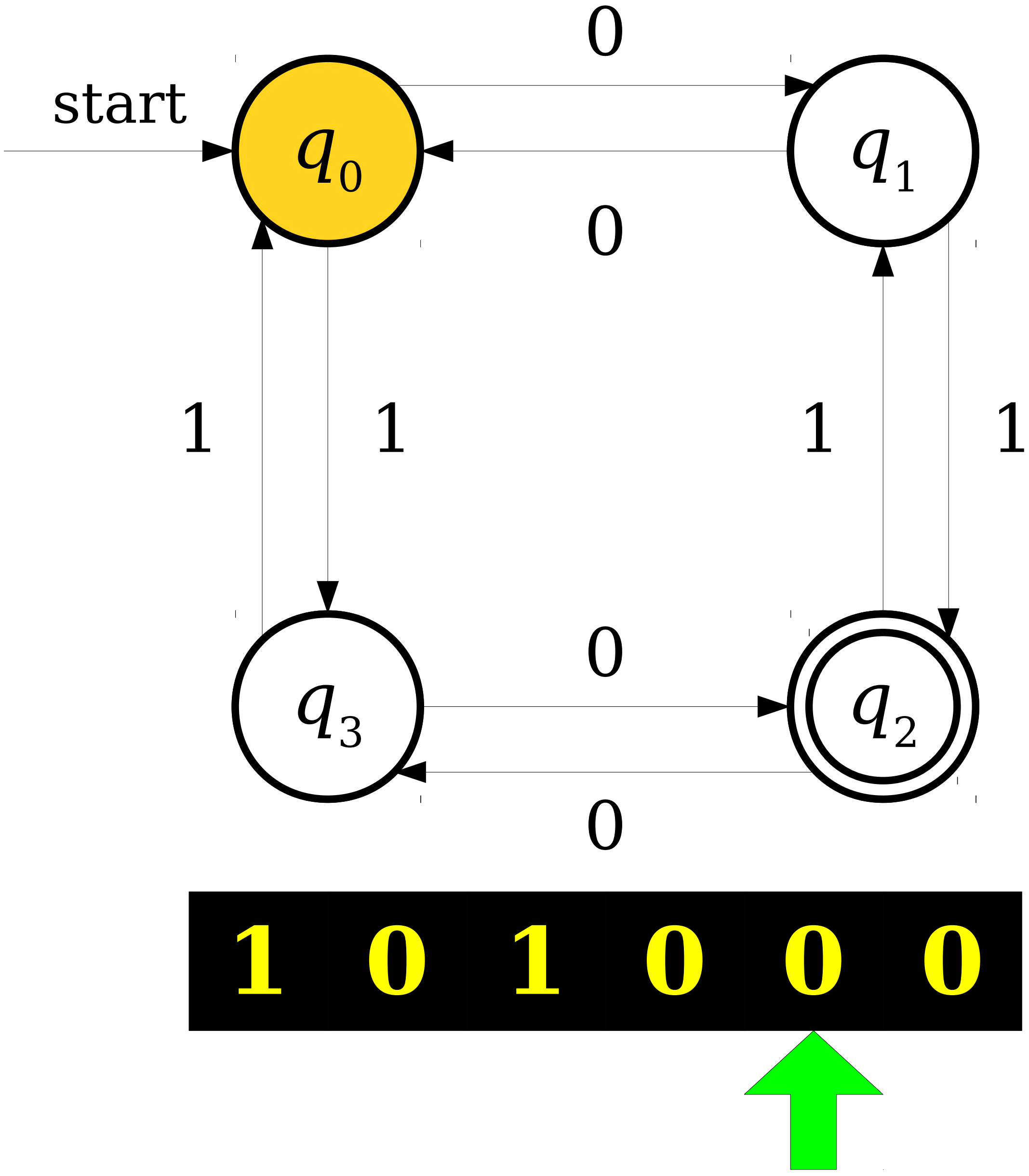
A Simple Finite Automaton



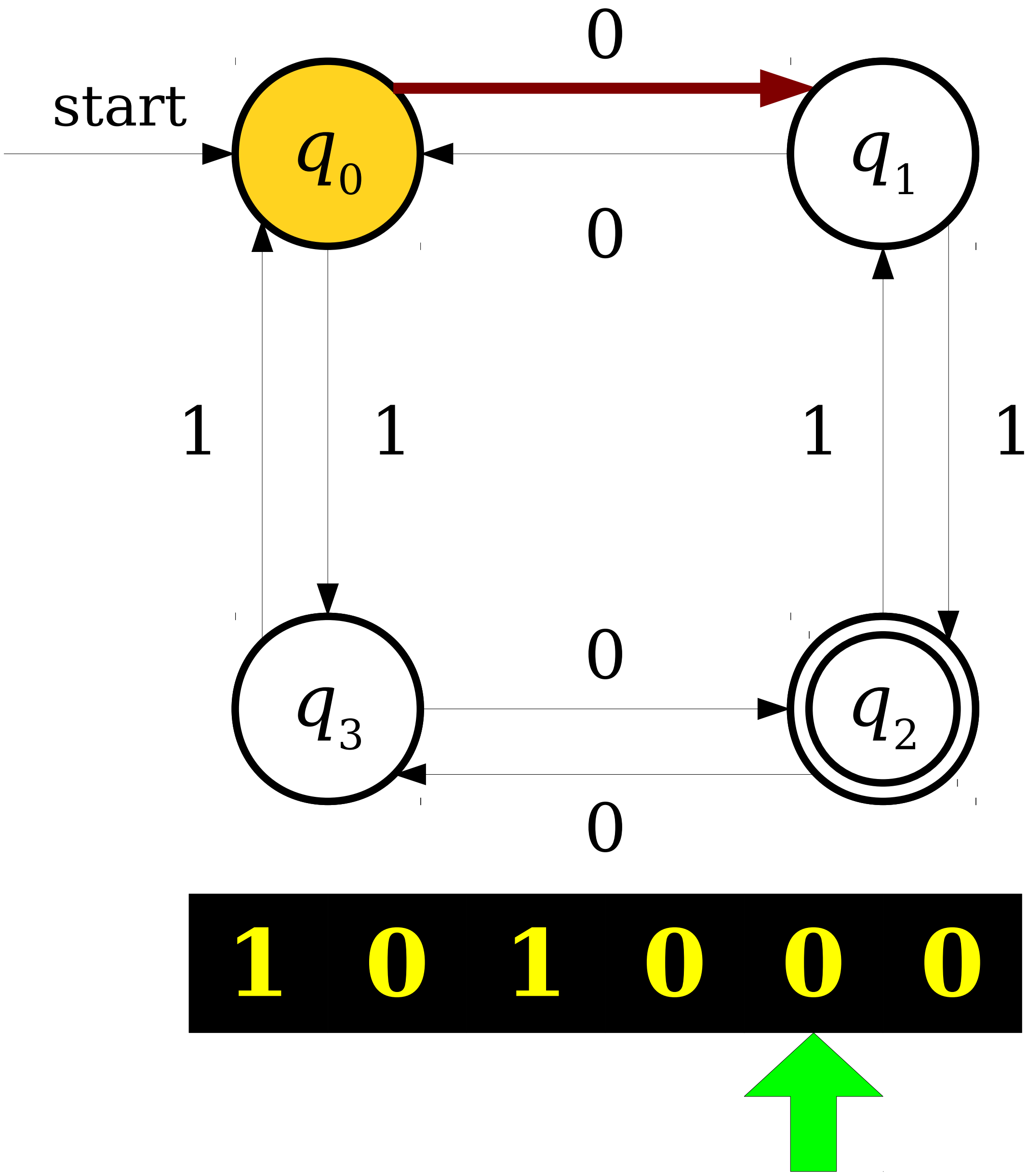
A Simple Finite Automaton



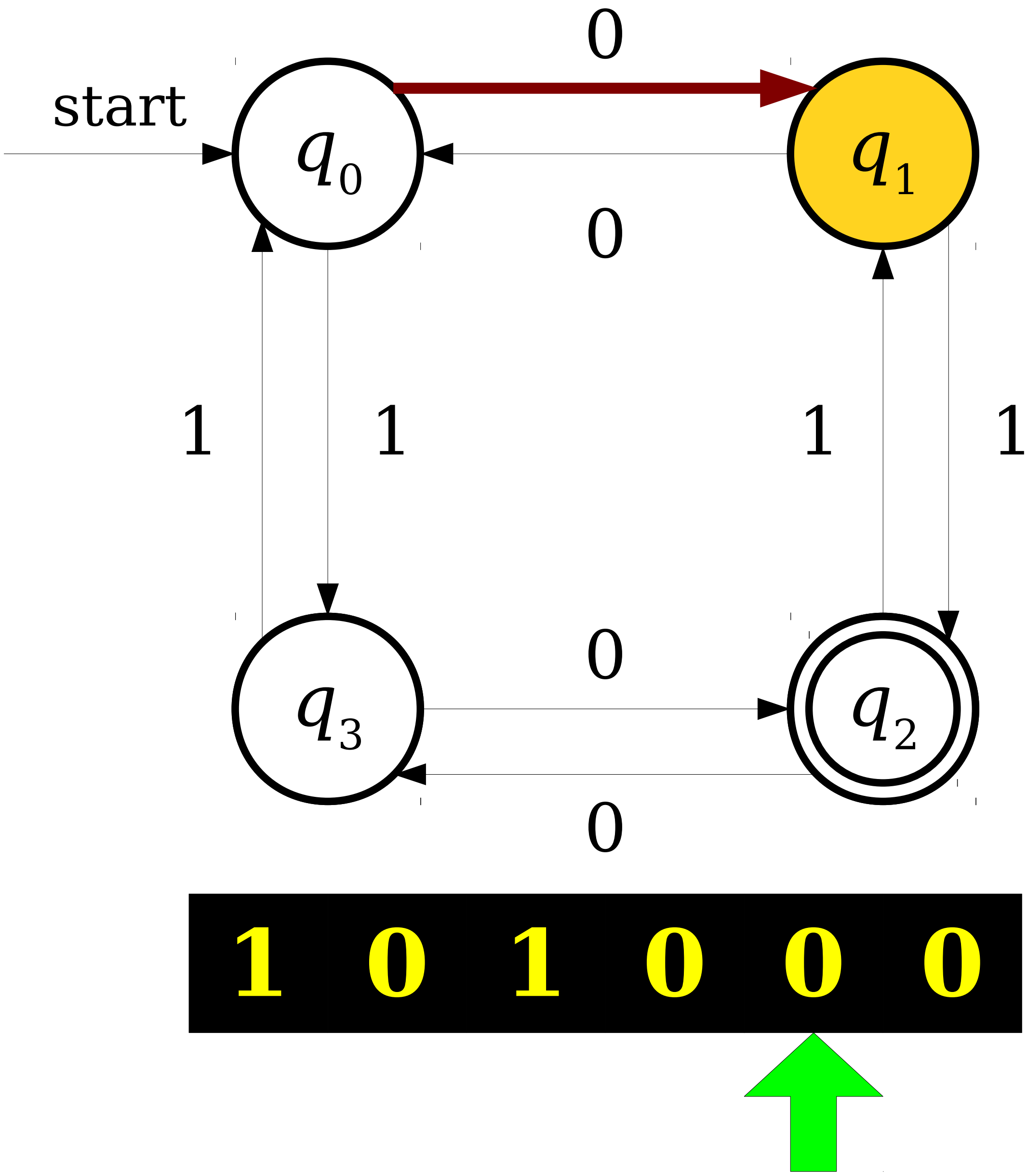
A Simple Finite Automaton



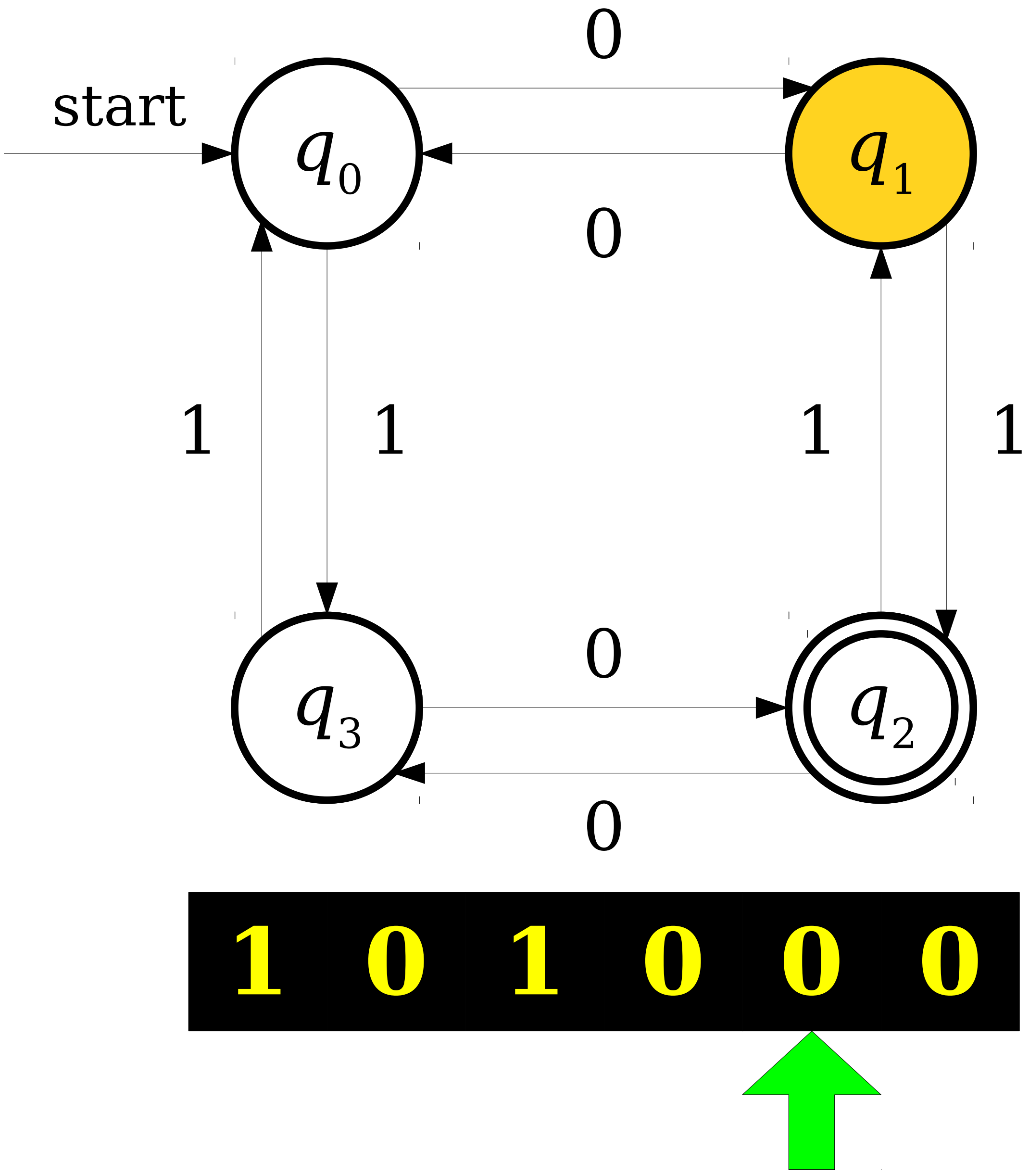
A Simple Finite Automaton



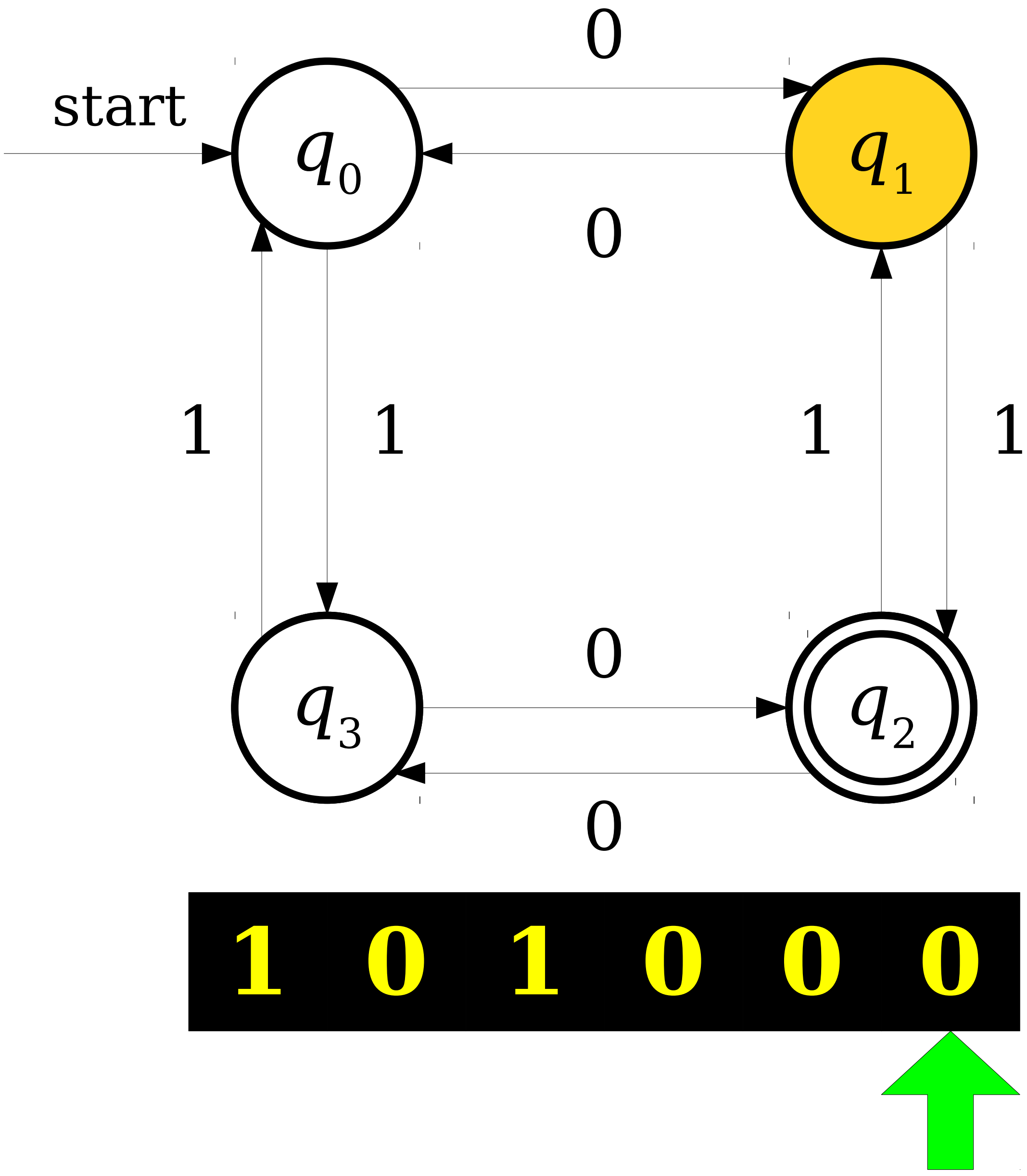
A Simple Finite Automaton



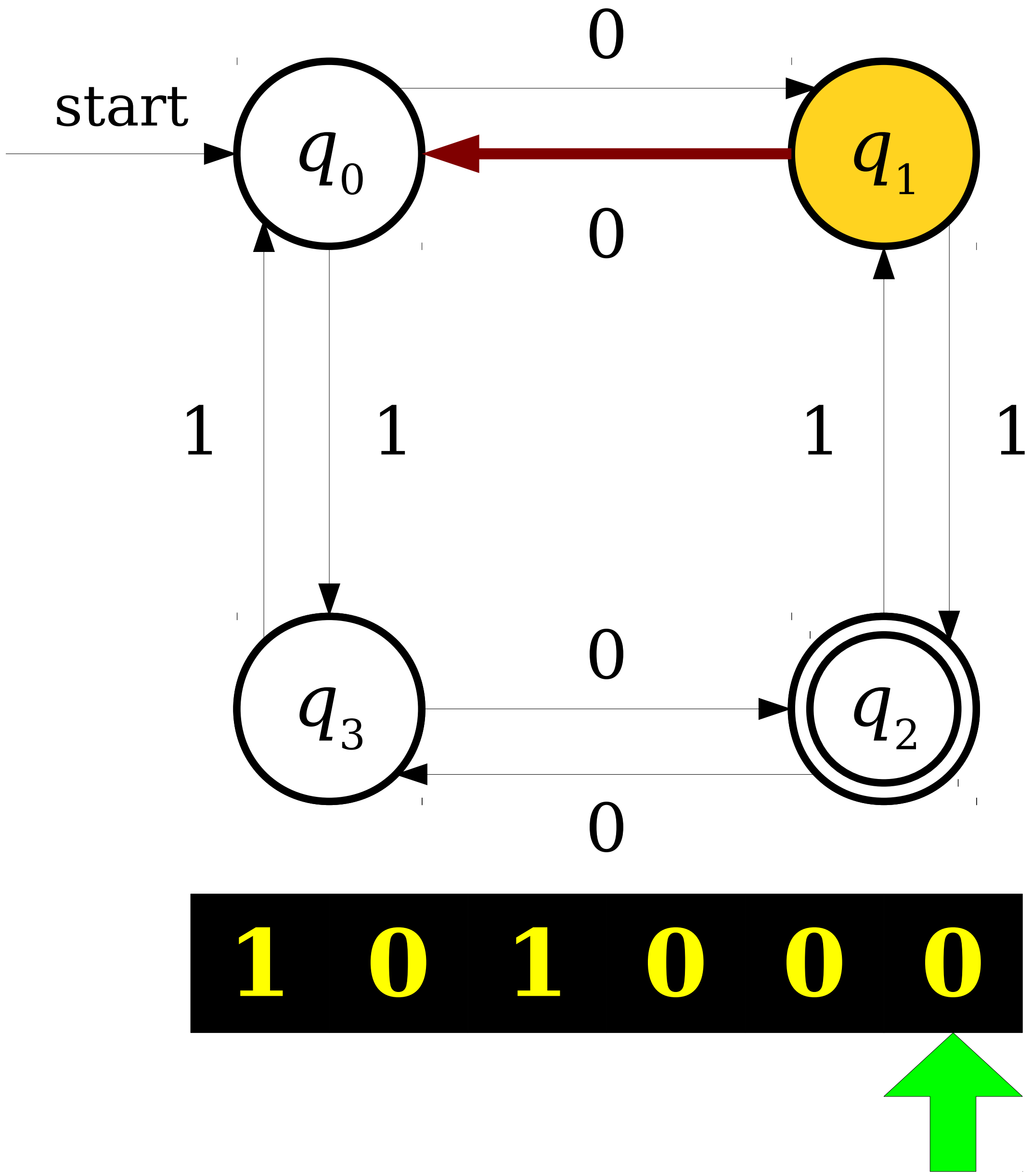
A Simple Finite Automaton



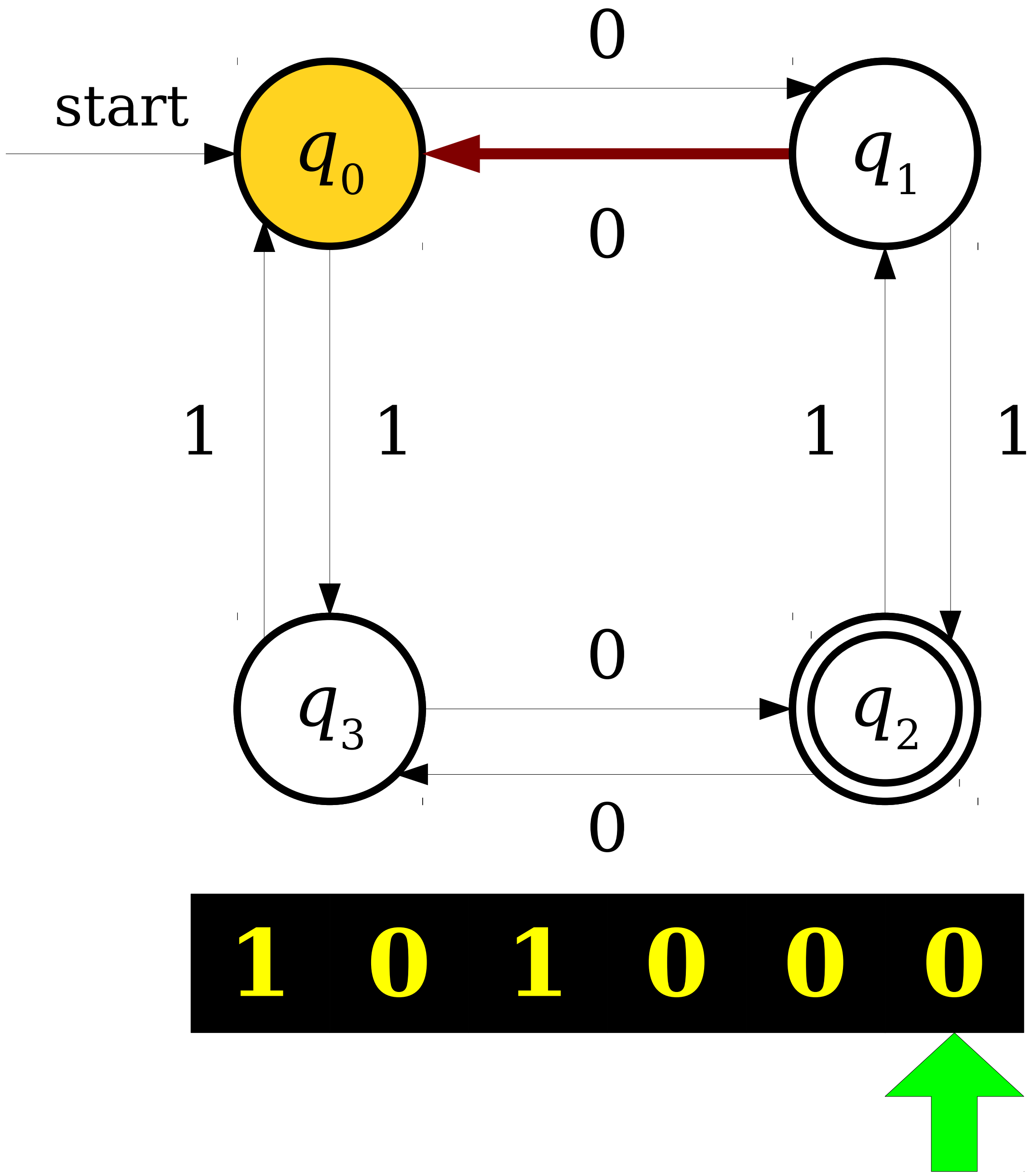
A Simple Finite Automaton



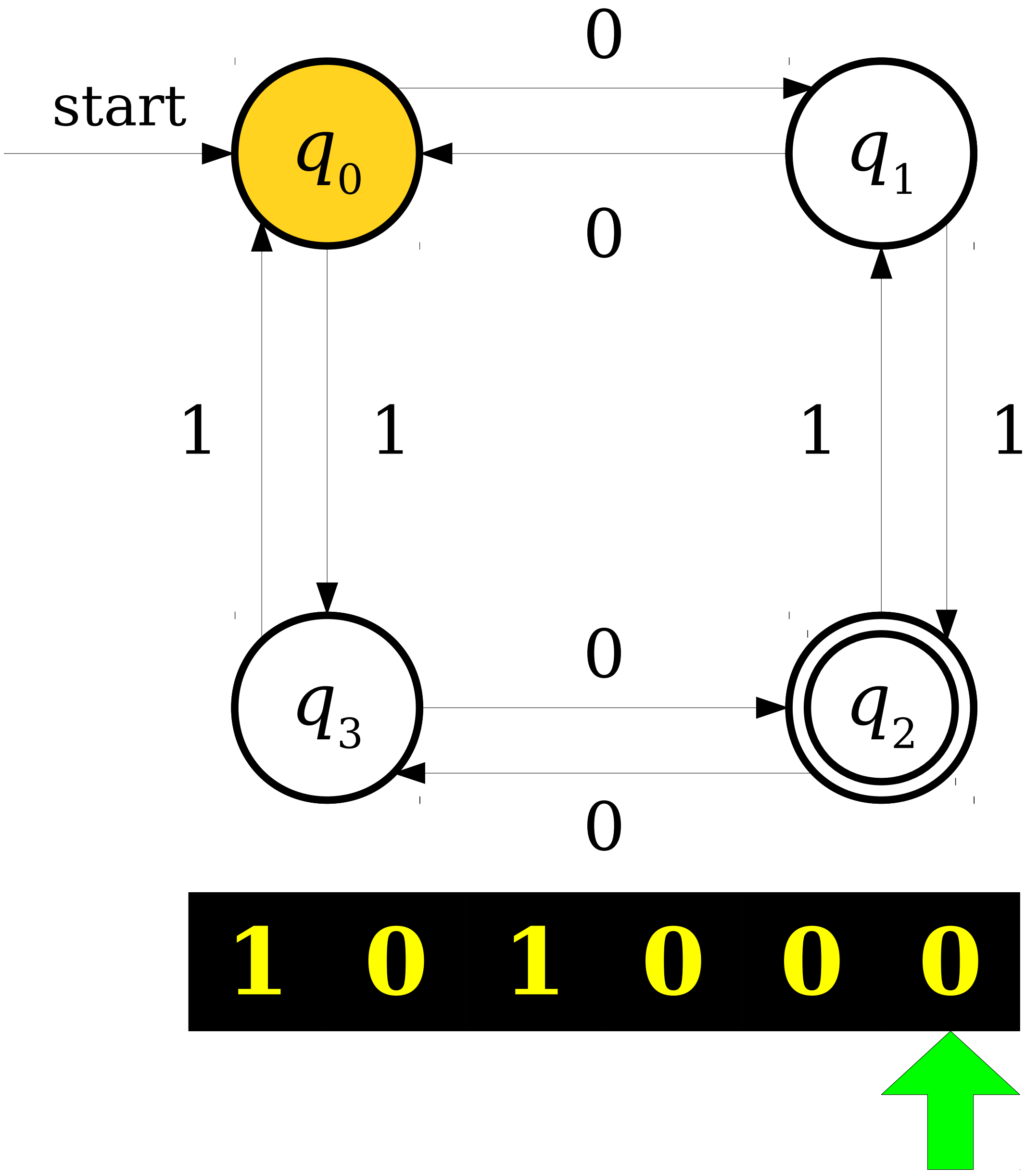
A Simple Finite Automaton



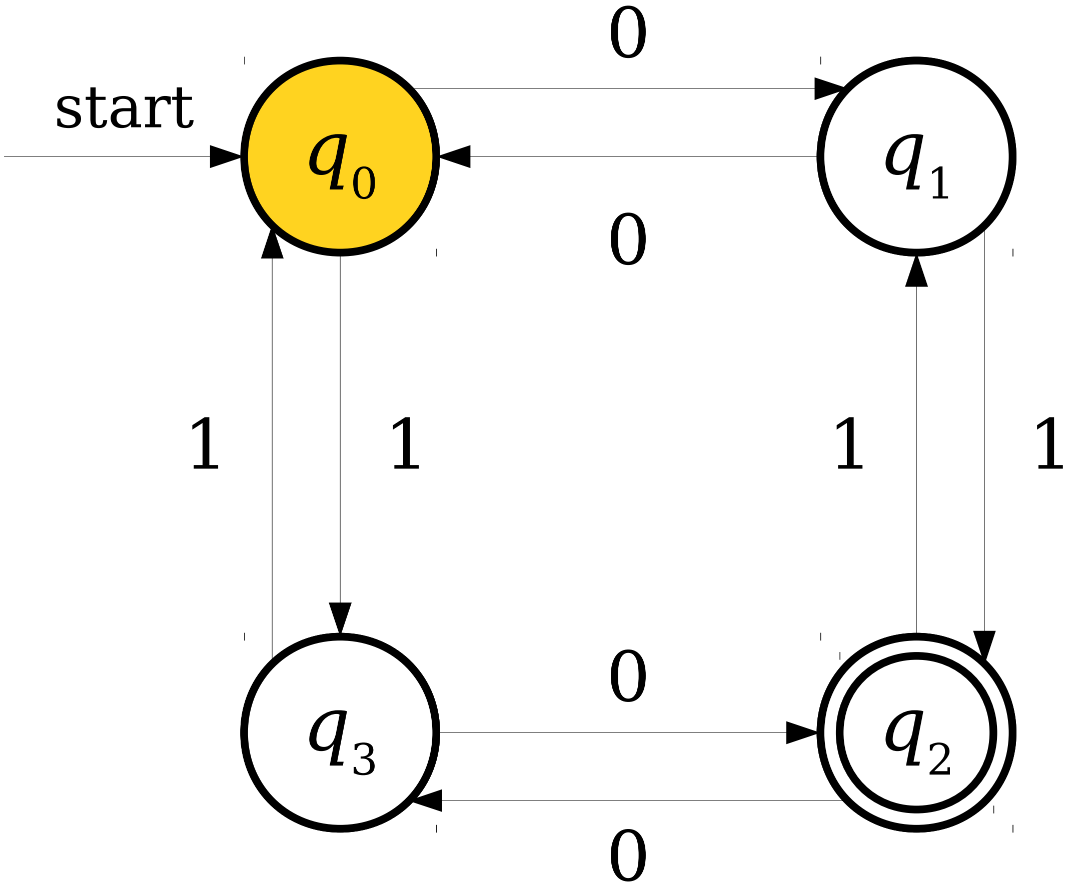
A Simple Finite Automaton



A Simple Finite Automaton

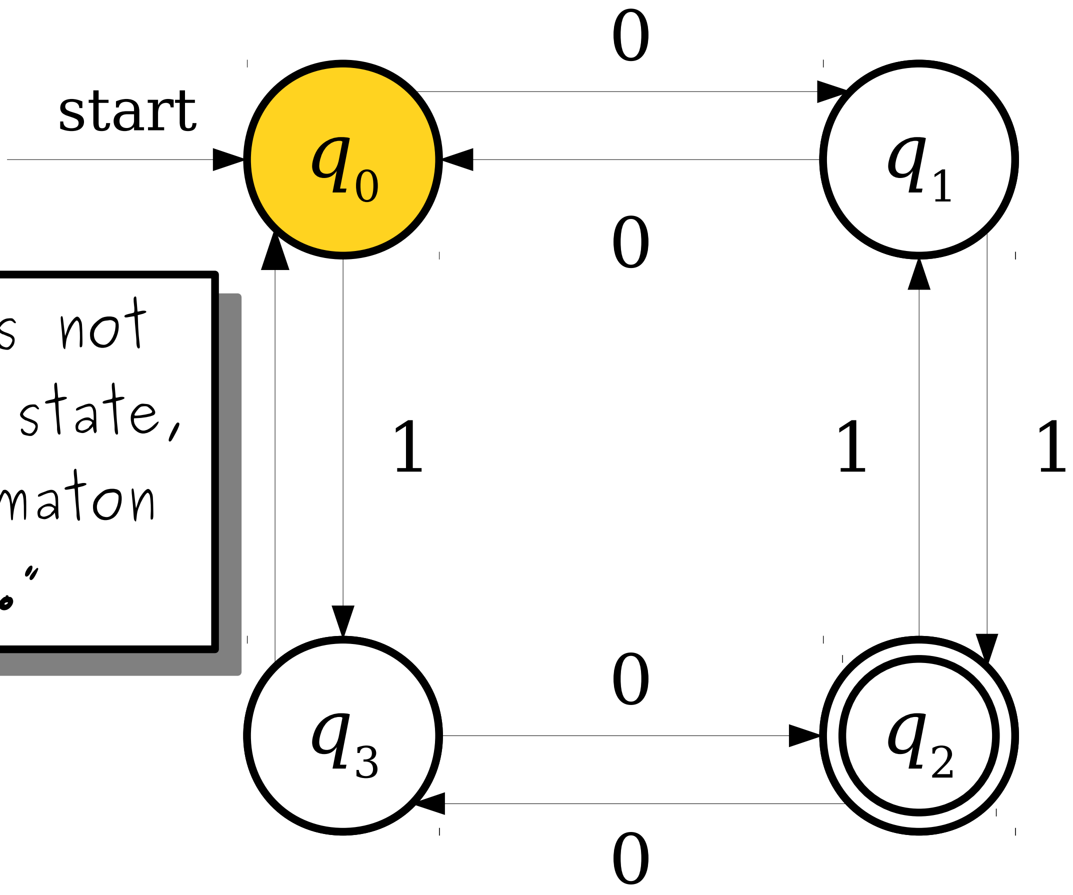


A Simple Finite Automaton



1 0 1 0 0 0

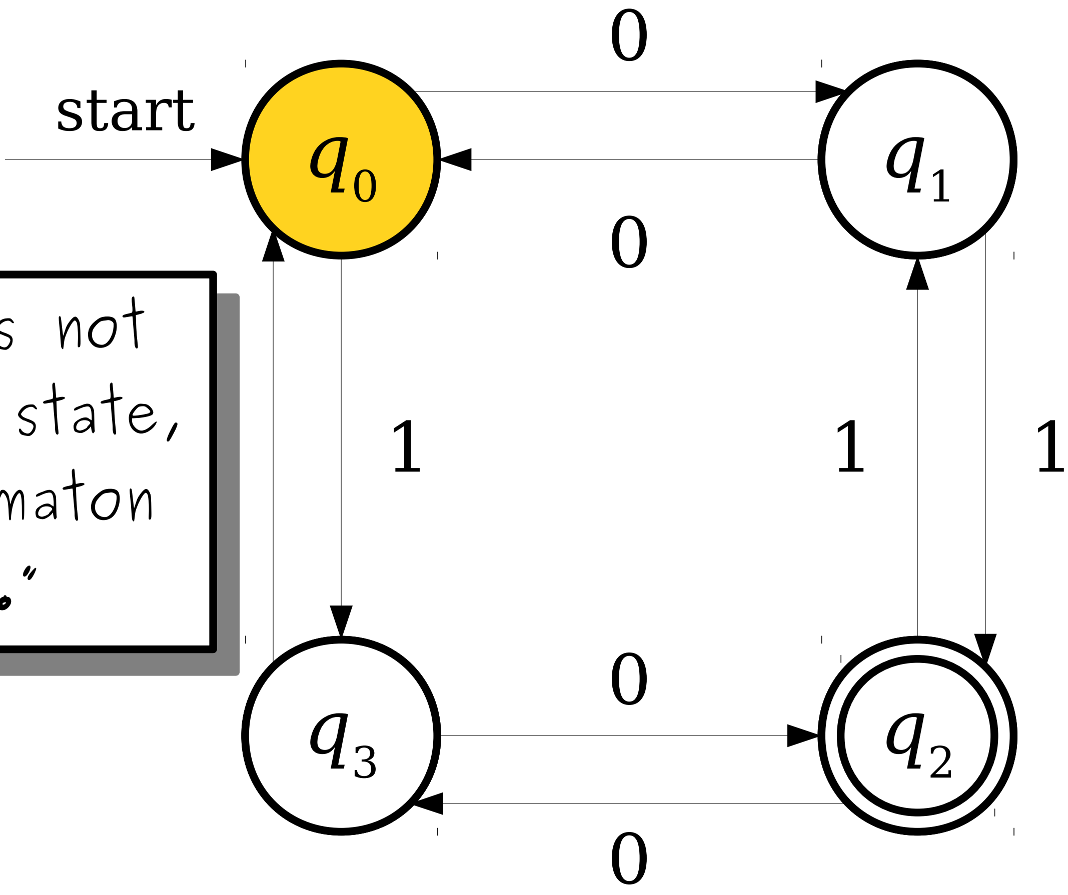
A Simple Finite Automaton



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

1 0 1 0 0 0

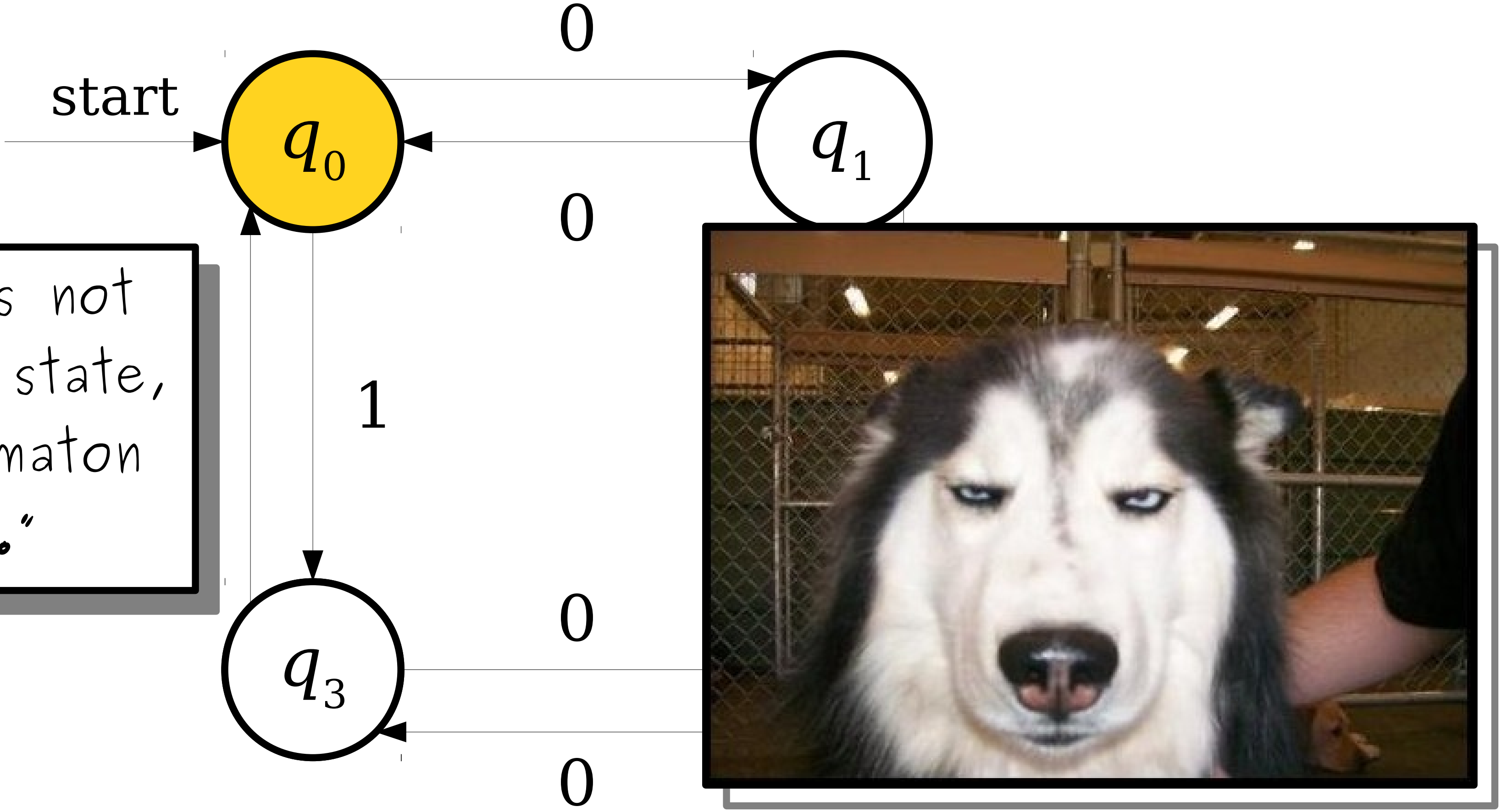
A Simple Finite Automaton



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

1 0 1 0 0 0

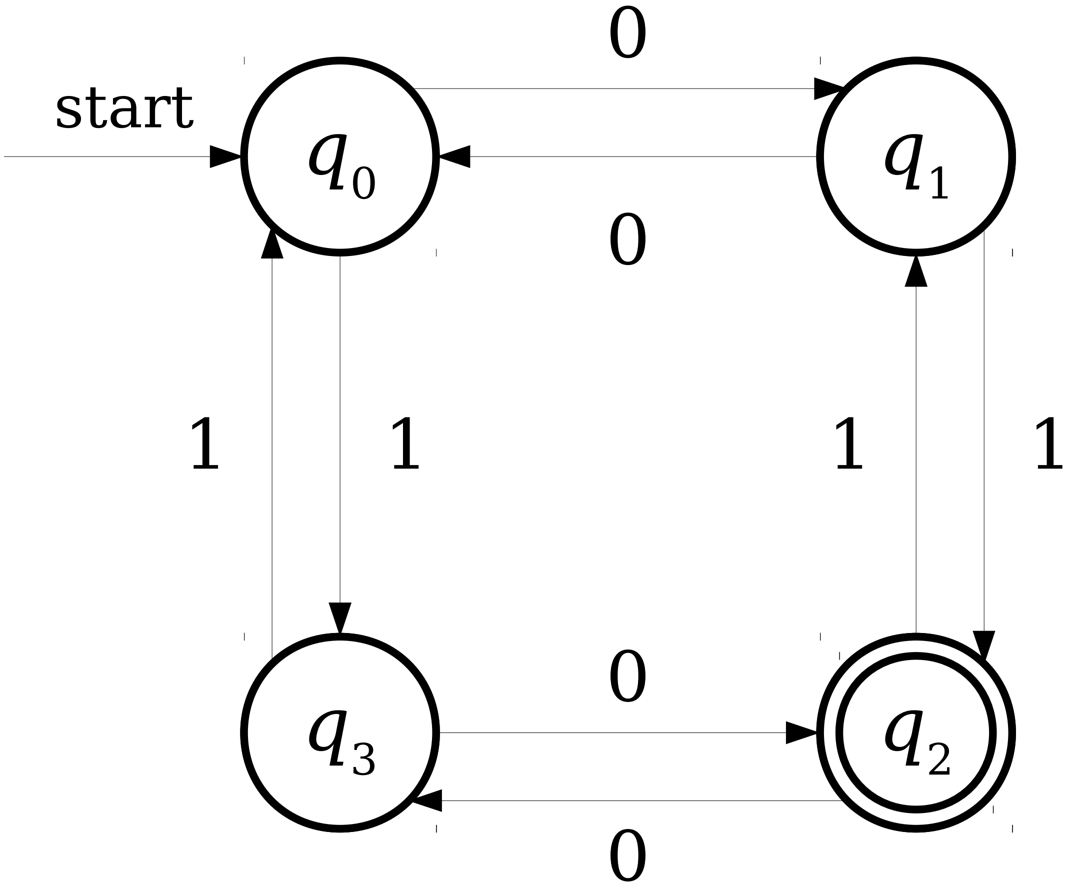
A Simple Finite Automaton



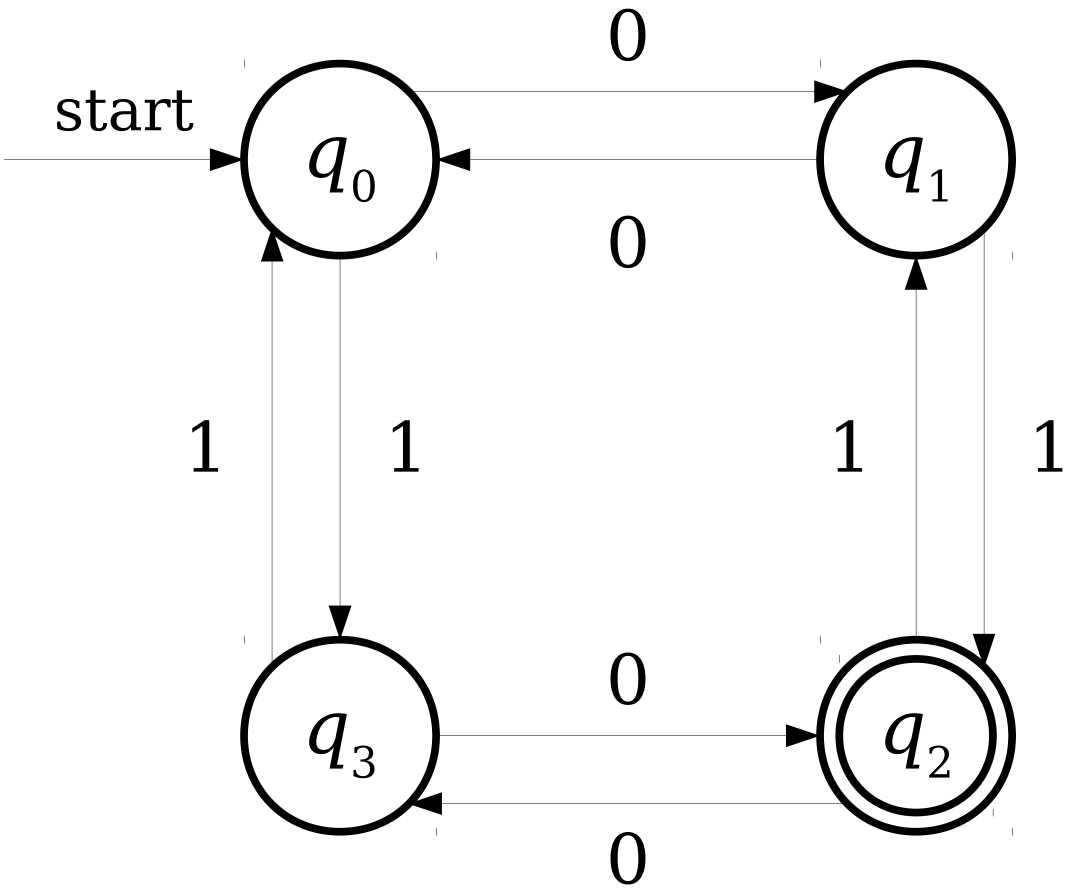
This state is not an accepting state, so the automaton says "no."

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.

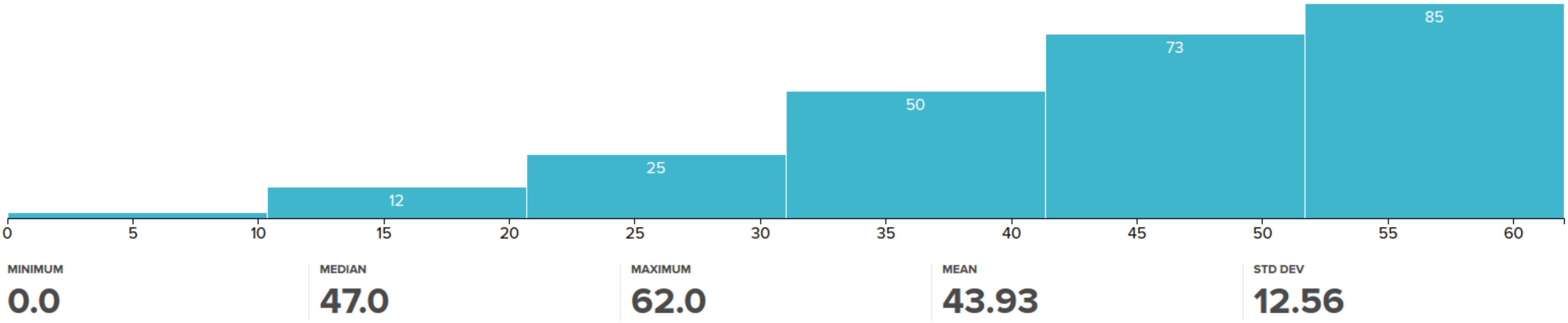
Time-Out For Announcements!

Problem Sets

- Problem Set Four was due at the start of class today.
    - You have three 24-hour late days to use throughout the quarter however you’d like.
- Problem Set Five goes out today. It’s due next Friday at the start of class.
    - Play around with just about everything we’ve seen so far: graphs, binary relations, functions, cardinality, the pigeonhole principle, and induction!
    - There is no checkpoint problem.

Problem Set Three

- PS3 has been graded. Here's the distribution:



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
0.0	47.0	62.0	43.93	12.56

- We're happy to chat with you about this problem set, the feedback you received, and how to tune and nudge things going forward. Feel free to stop by OH!

Your Questions

“Is there a documentary or book that really changed the way you thought about something?”

Yes! The books “Unlocking the Clubhouse” and “Whistling Vivaldi” completely changed how I think about questions of identity and belonging. And “Jiro Dreams of Sushi” was a surprisingly good exploration of the benefits and drawbacks of a singleminded focus on perfection in one domain.

“Why do you hate philosophy/philosophers so much?”

I'm sorry that I've given that impression! Many of the foundational ideas in computer science and mathematics are due to philosophers like Frege, Chomsky, and Russell. I think it's hugely valuable to explore notions of truth and meaning and would definitely recommend taking courses to learn more about that!

“Why should I or shouldn't I take a gap year if I REALLY love CS but have issues adjusting to the pace here?”

Taking a gap year can be a great experience if you're strategic about when you take it and what you want to accomplish, but it's certainly not for everyone. I typically would not recommend taking a gap year just because of the pace – I'm not sure that would necessarily fix the underlying issue. If you're in a position like this and want to chat, please let me know! I'm happy to offer advice.

“Why do you use the number 137 everywhere?”

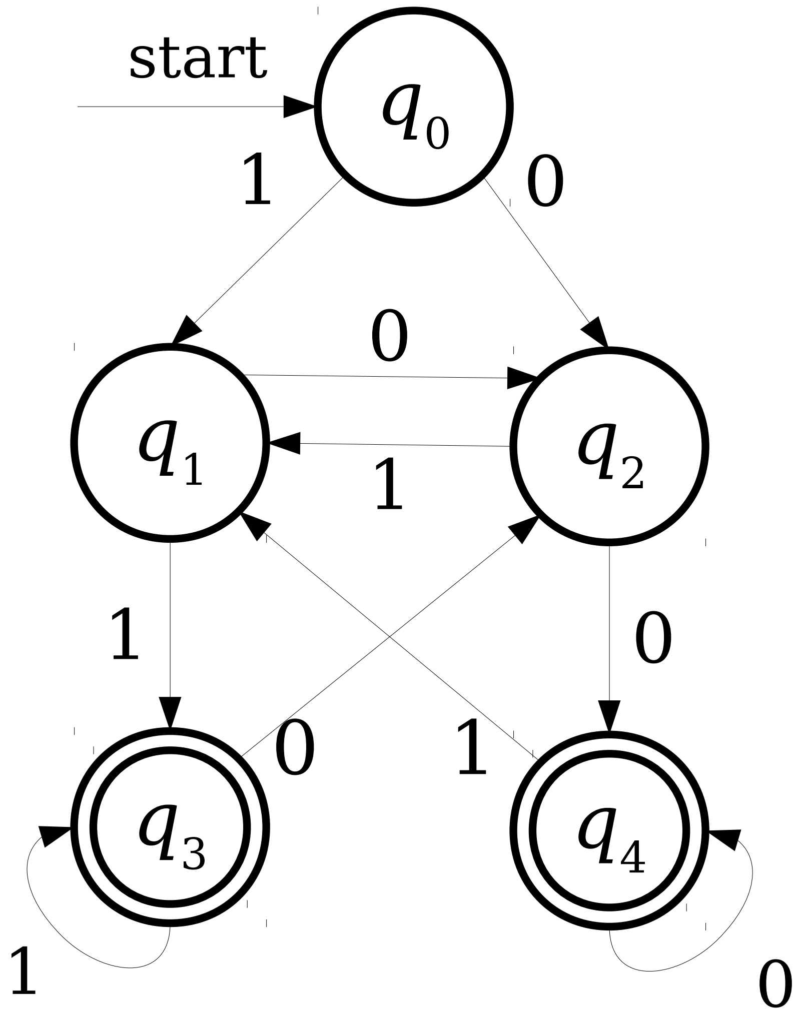
The number 137 is very close to the reciprocal of the fine-structure constant in physics:

$$\alpha = \frac{1}{4\pi\varepsilon_0}\frac{e^2}{\hbar c} = \frac{\mu_0}{4\pi}\frac{e^2 c}{\hbar} = \frac{k_e e^2}{\hbar c} = \frac{c\mu_0}{2R_K} = \frac{e^2}{4\pi}\frac{Z_0}{\hbar}$$

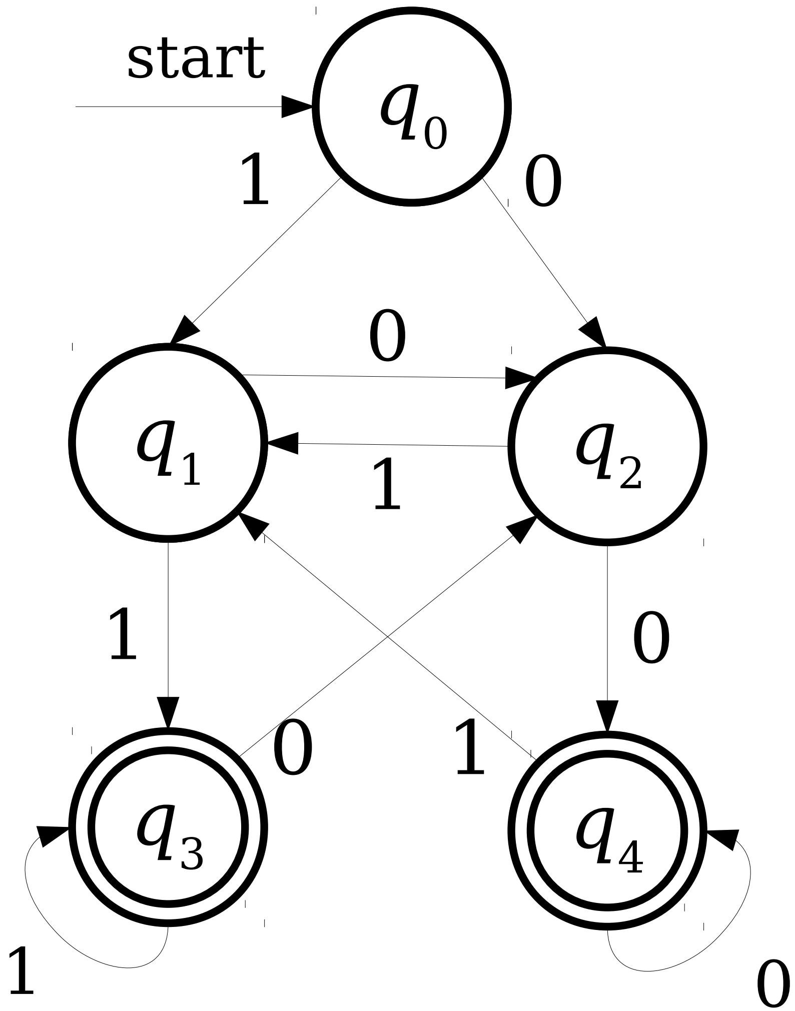
It has a fun history. Plus, it's a great “nothing-up-my-sleeve” number that (usually) indicates that the particular number doesn't have any meaning.

Back to CS103!

Accepting States, Revisited

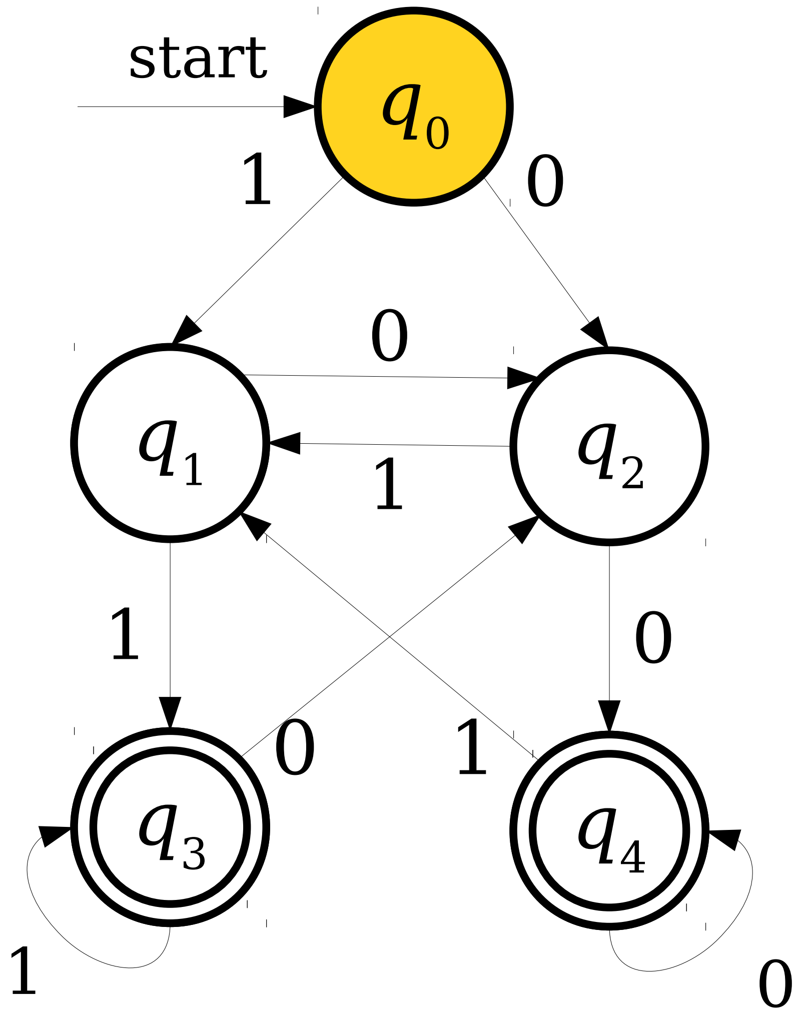


Accepting States, Revisited



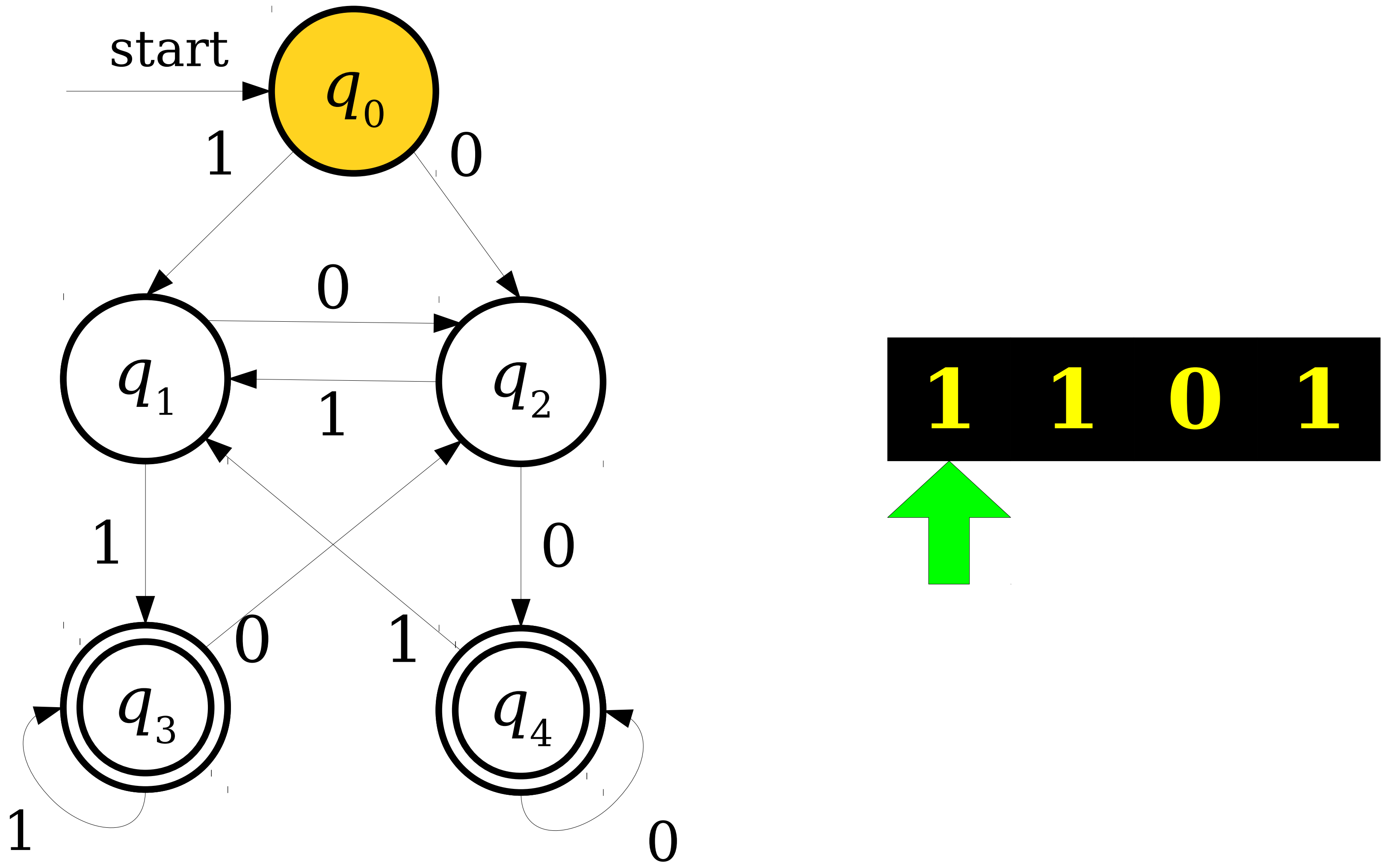
1 1 0 1

Accepting States, Revisited

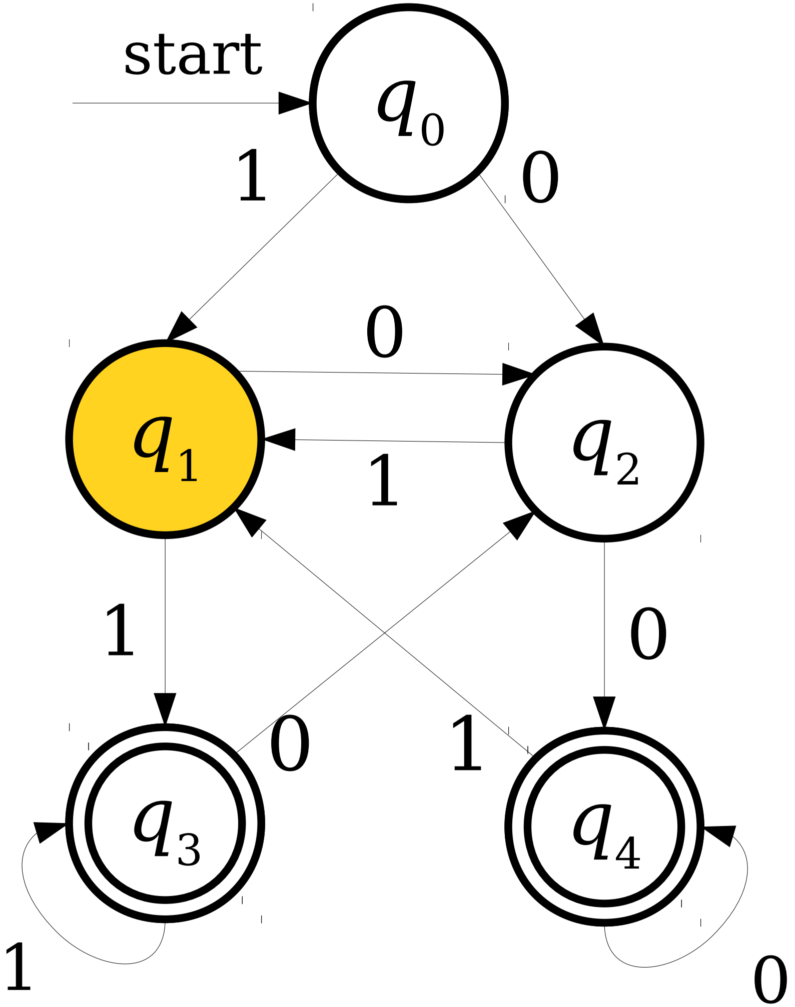


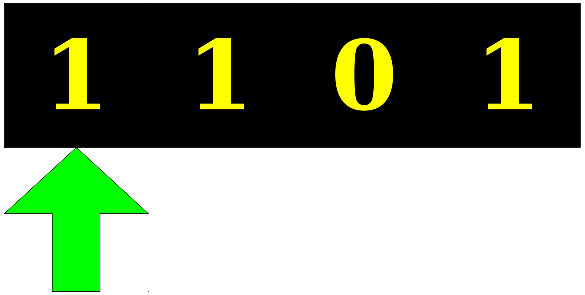
1 1 0 1

Accepting States, Revisited

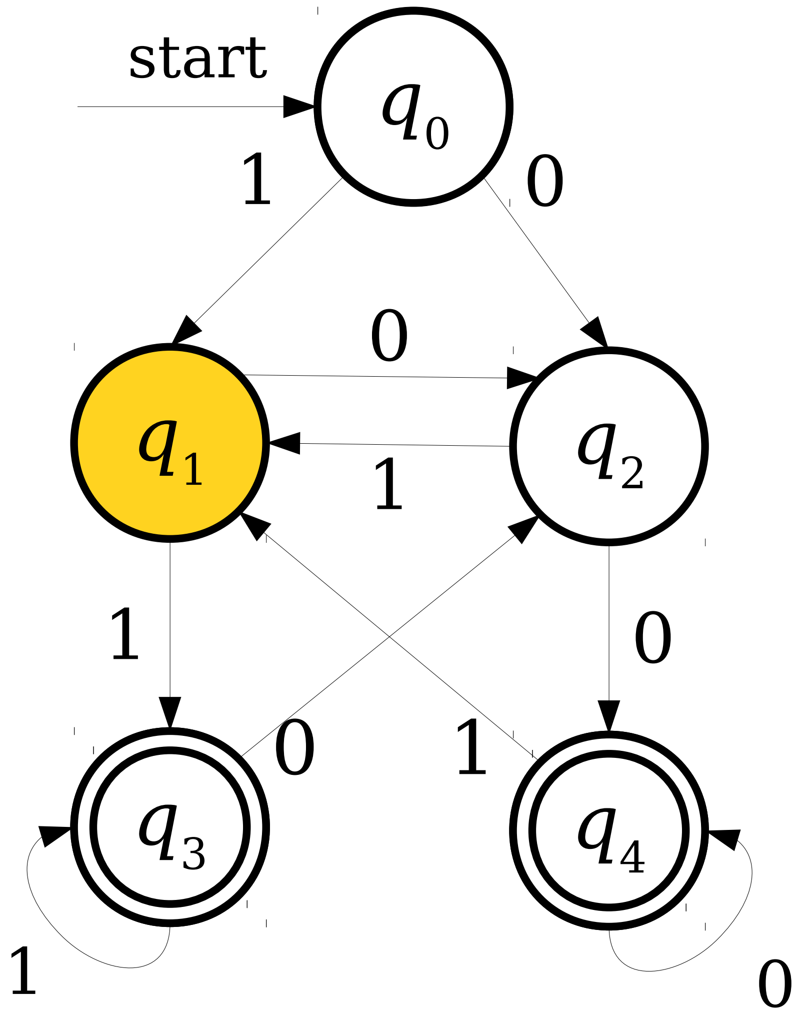


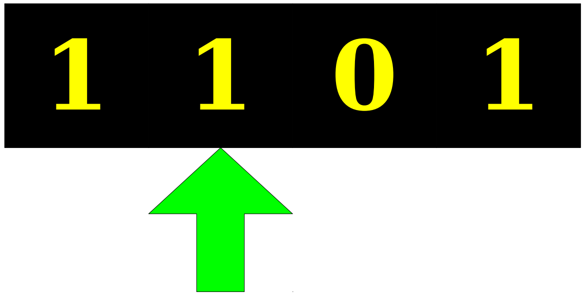
Accepting States, Revisited



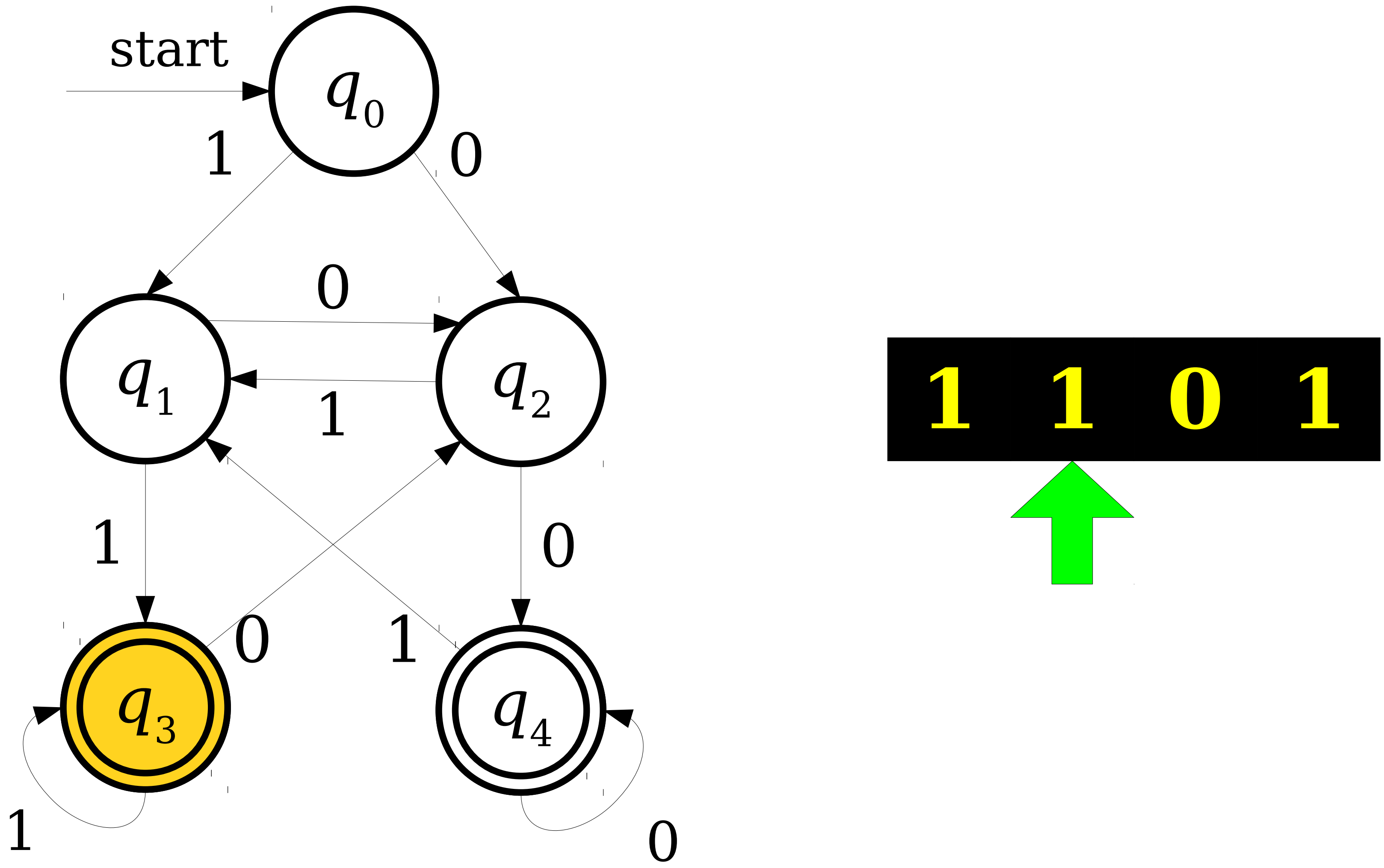


Accepting States, Revisited

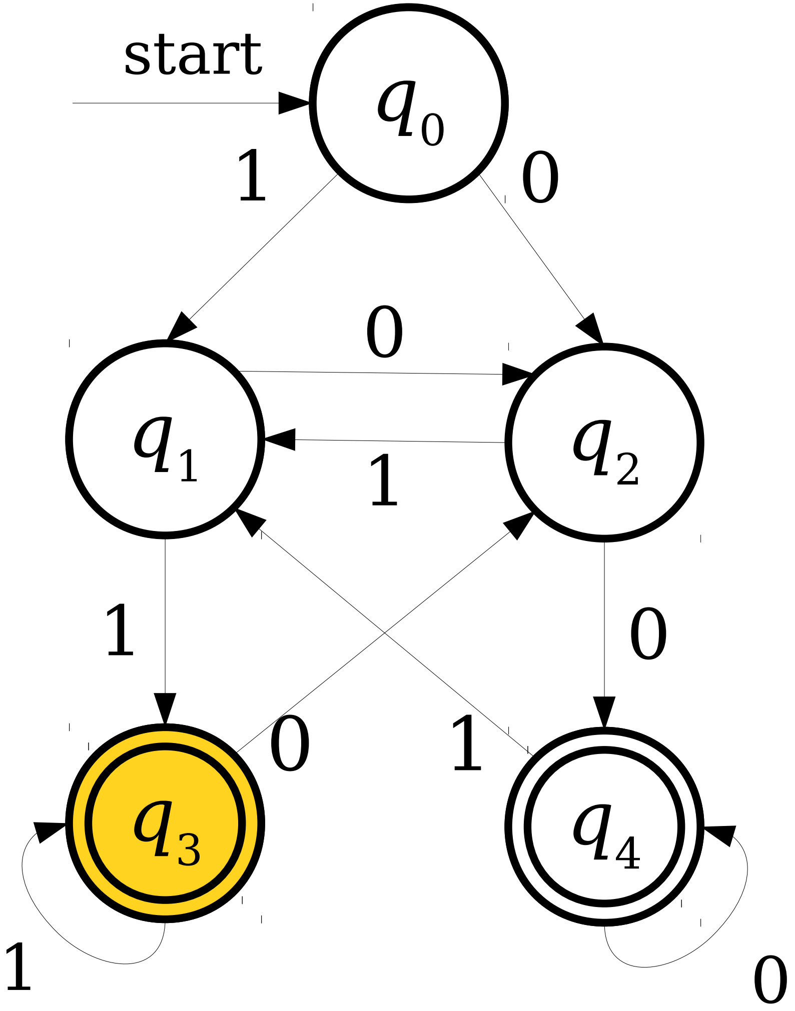


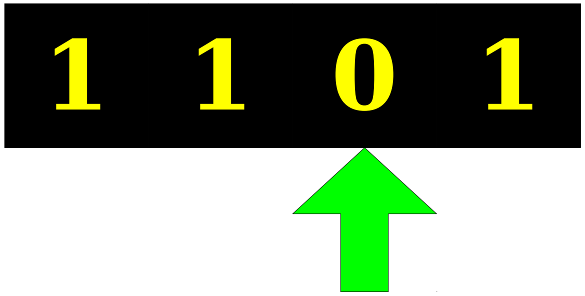


Accepting States, Revisited

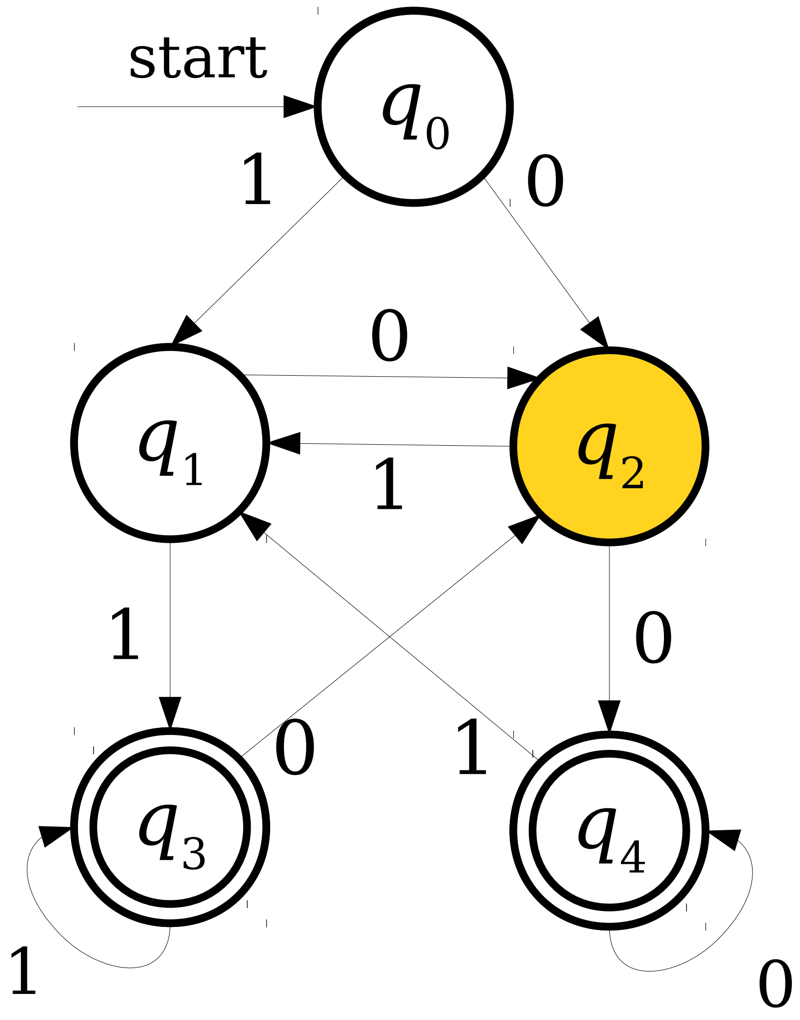


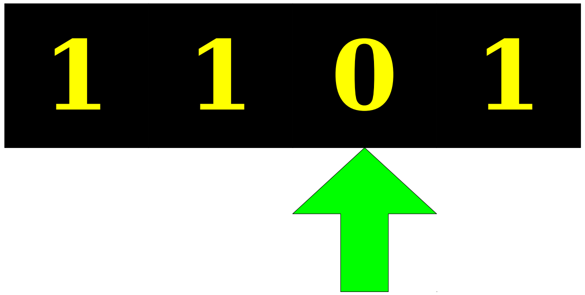
Accepting States, Revisited



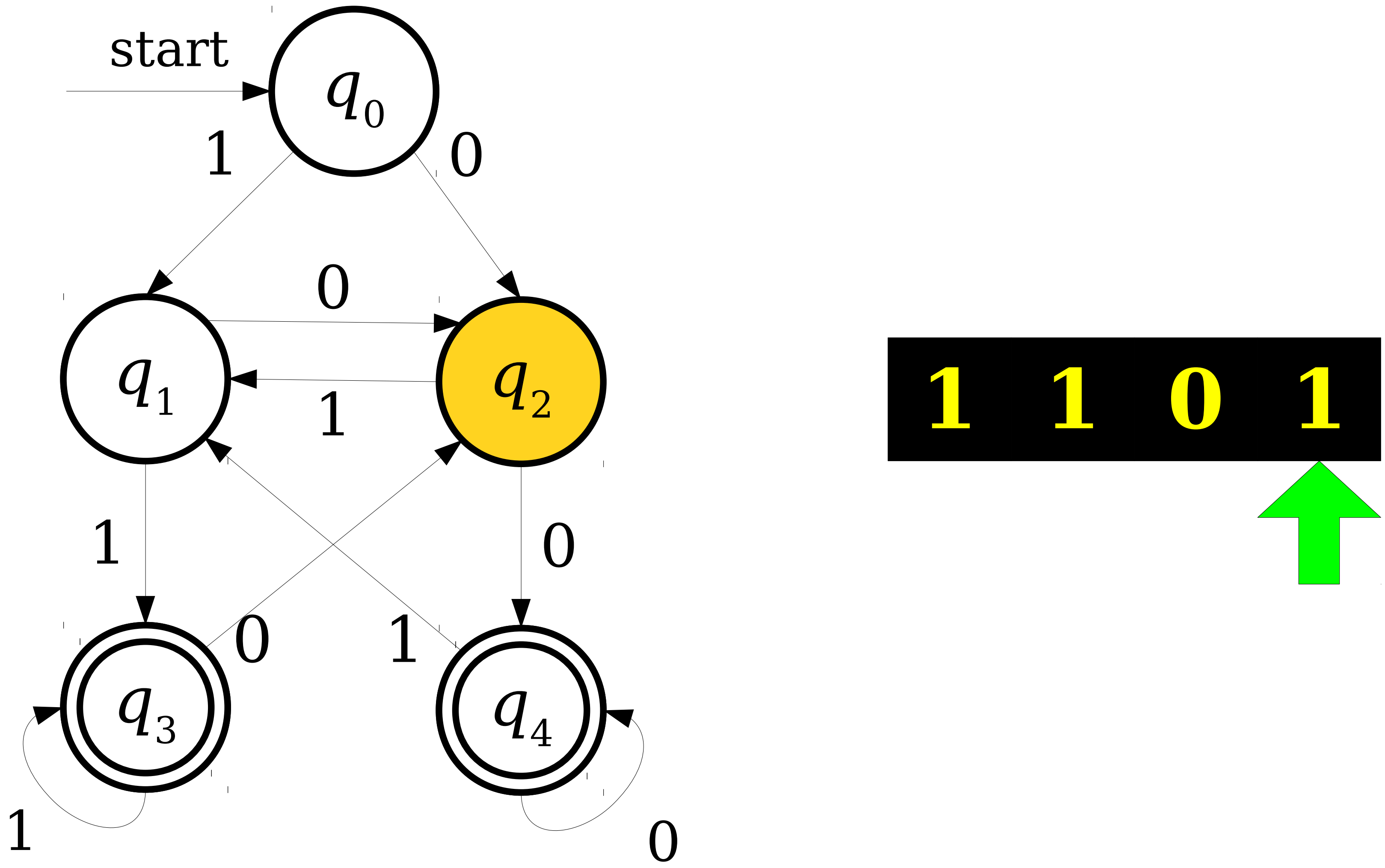


Accepting States, Revisited

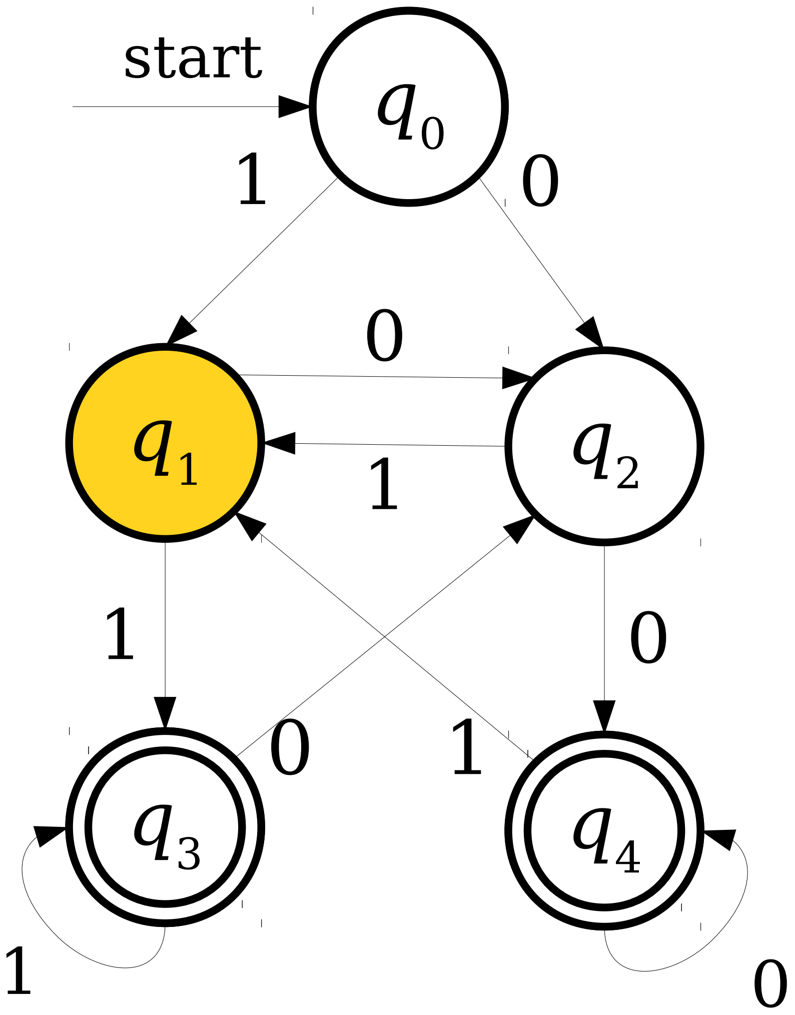


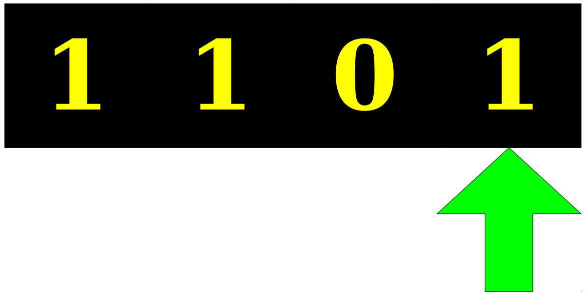


Accepting States, Revisited

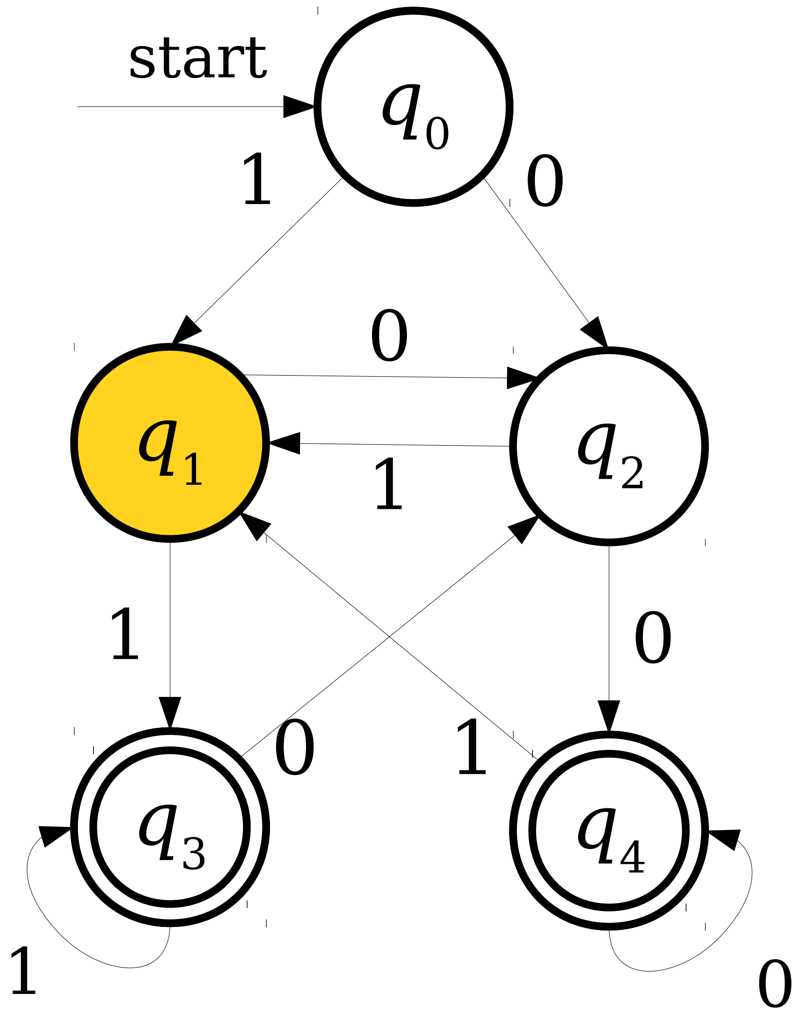


Accepting States, Revisited



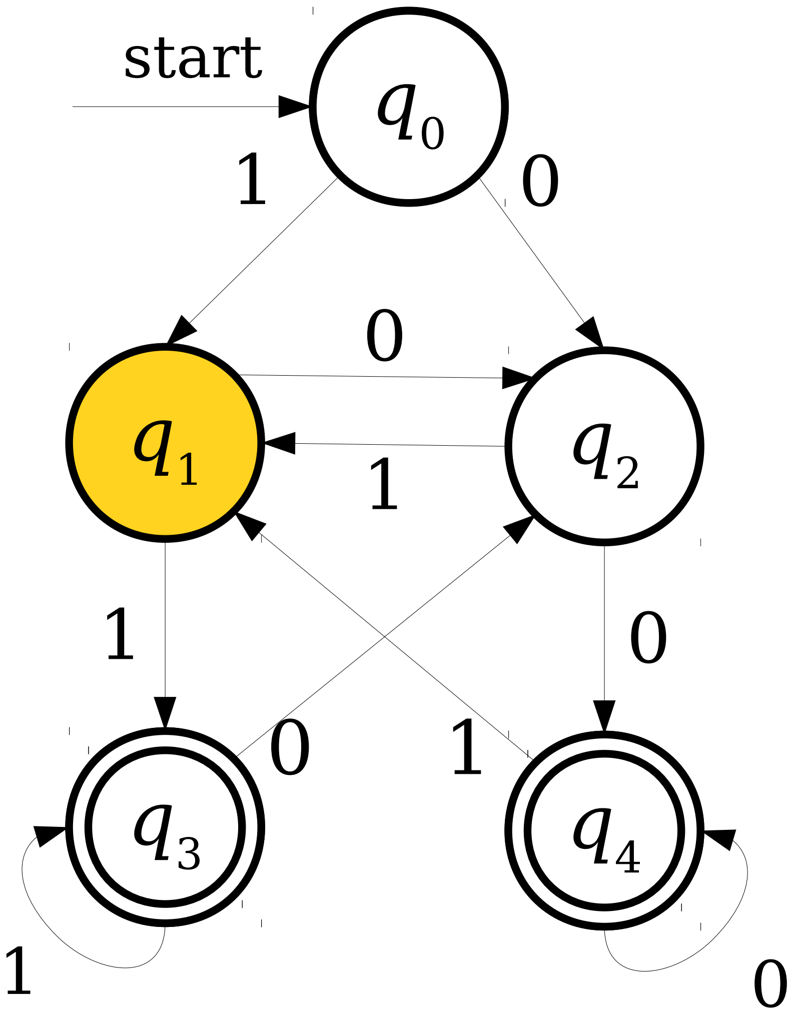


Accepting States, Revisited



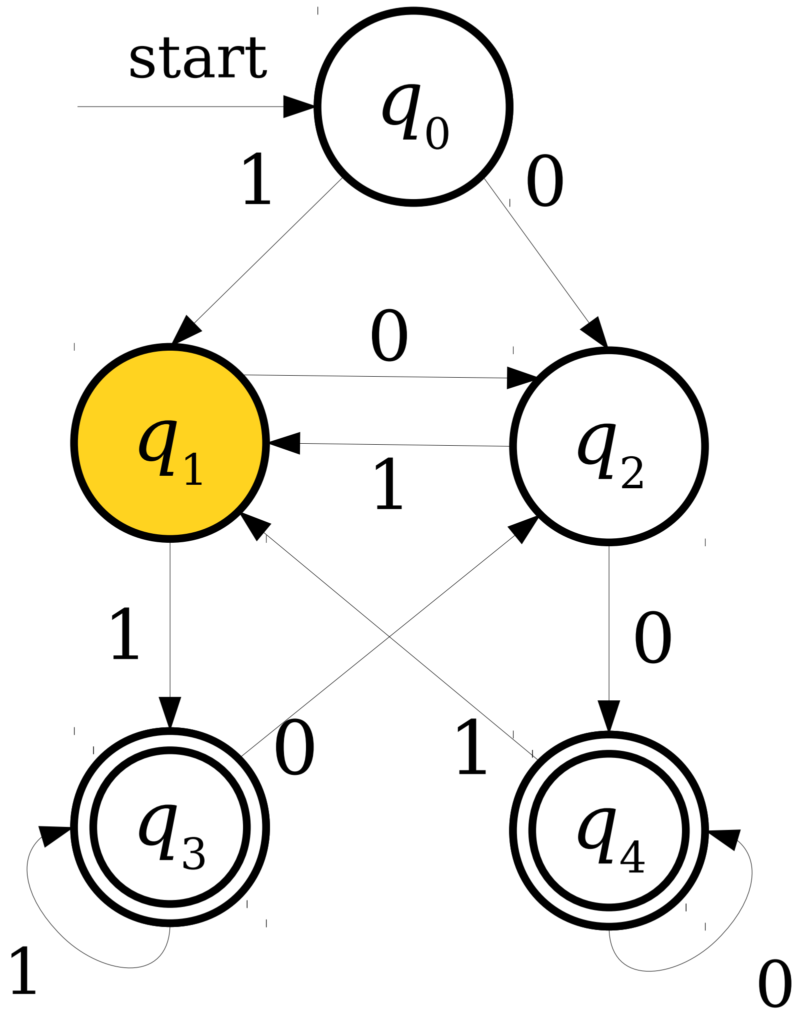
1 1 0 1

Accepting States, Revisited



1 1 0 1

Accepting States, Revisited

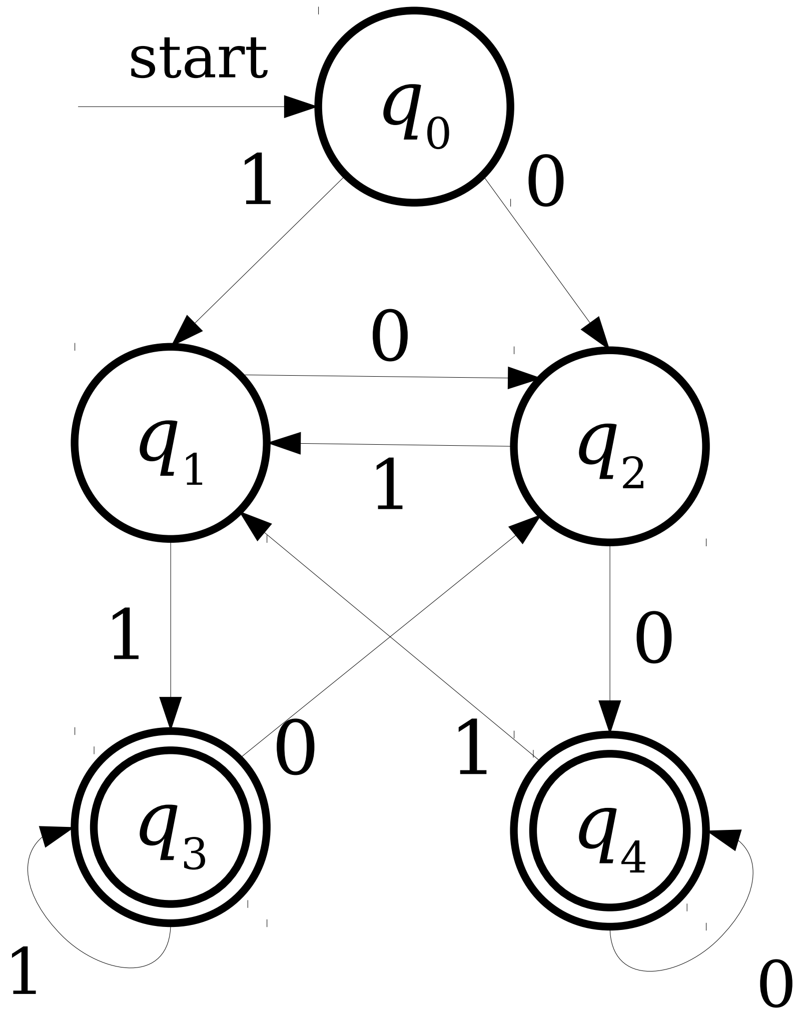


1 1 0 1

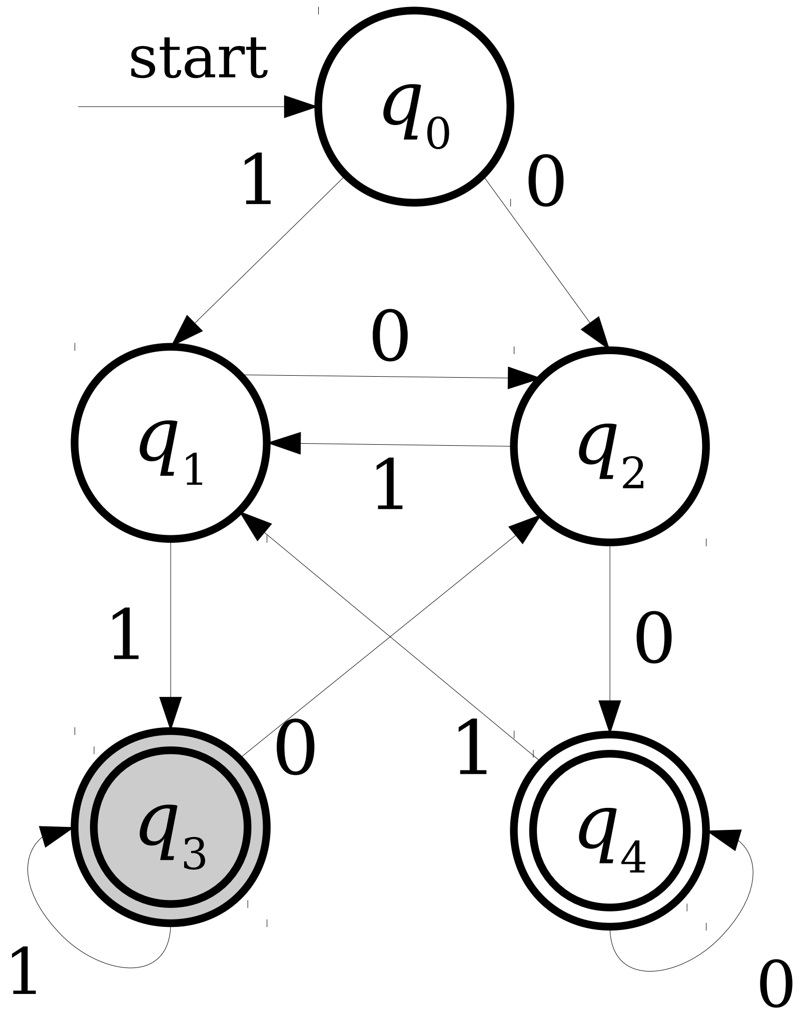
A finite automaton does ***not*** accept as soon as it enters an accepting state.

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

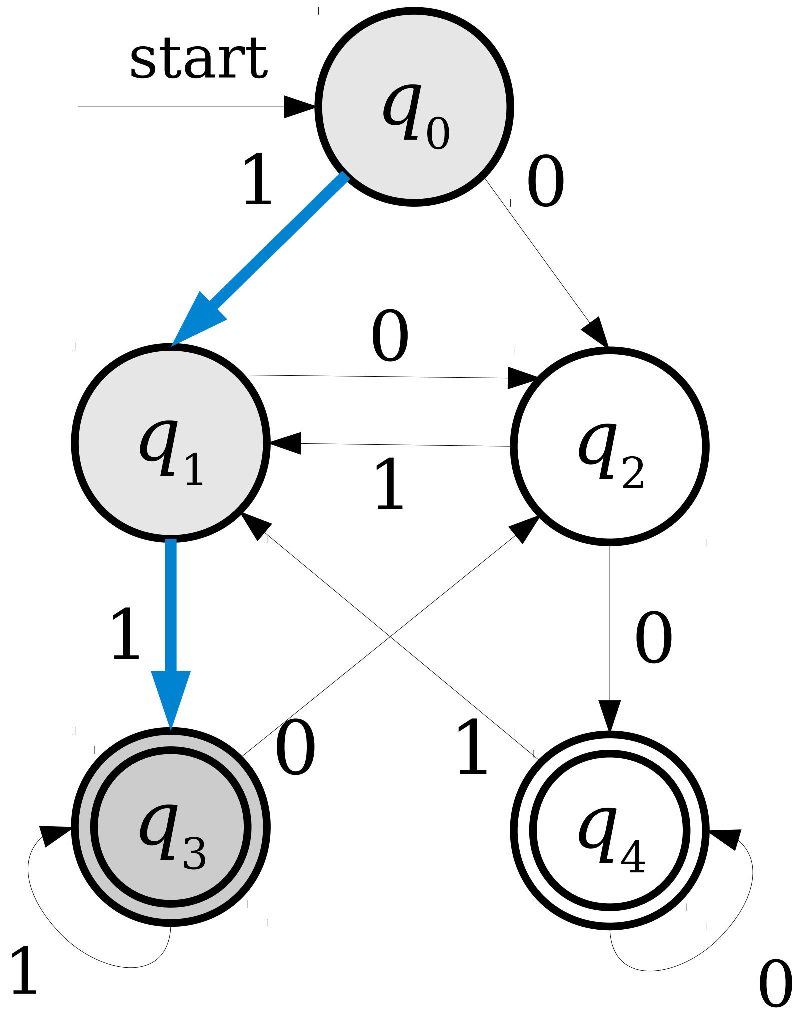
What Does This Accept?



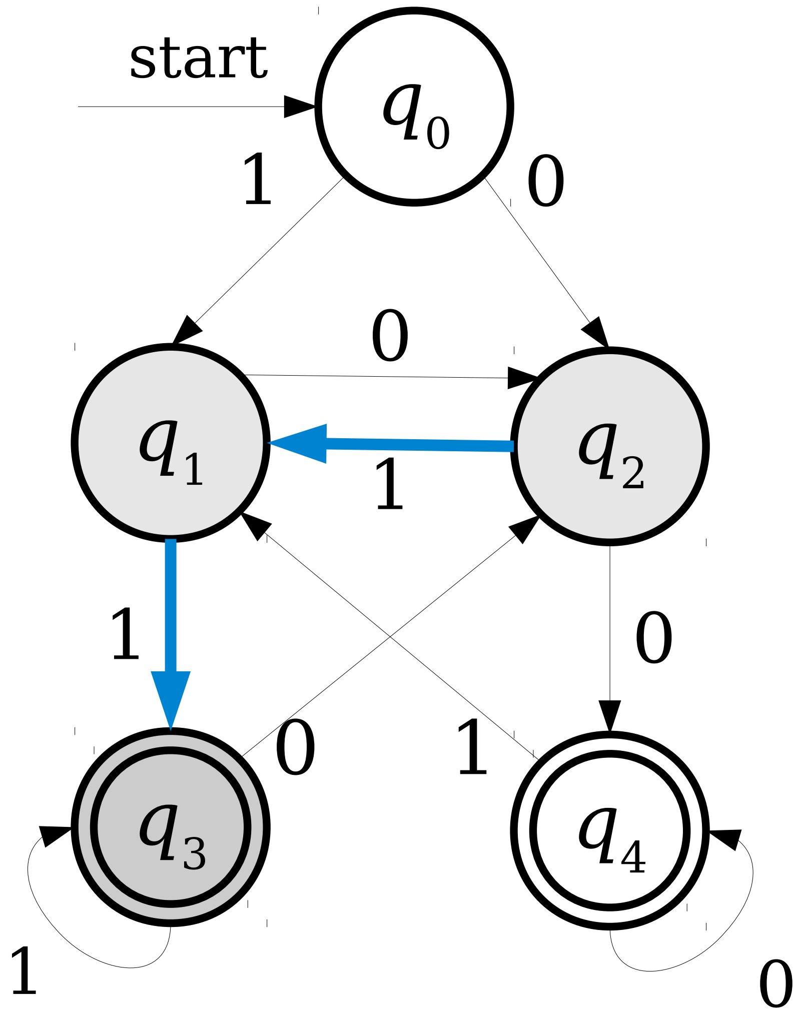
What Does This Accept?



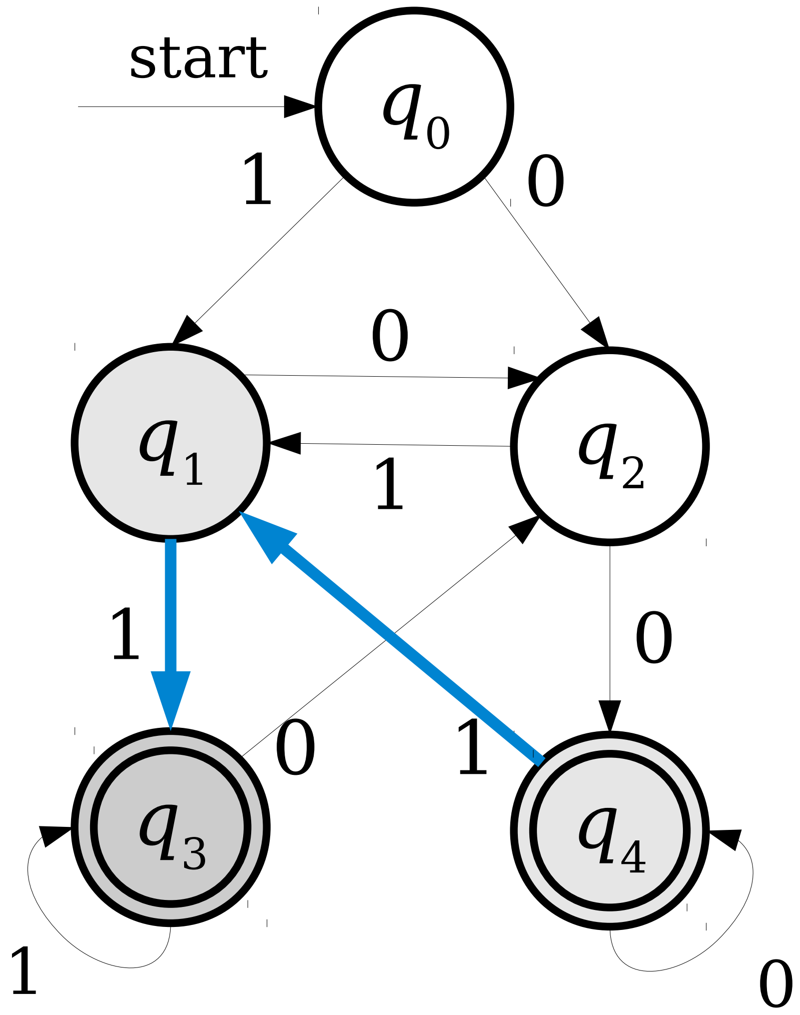
What Does This Accept?



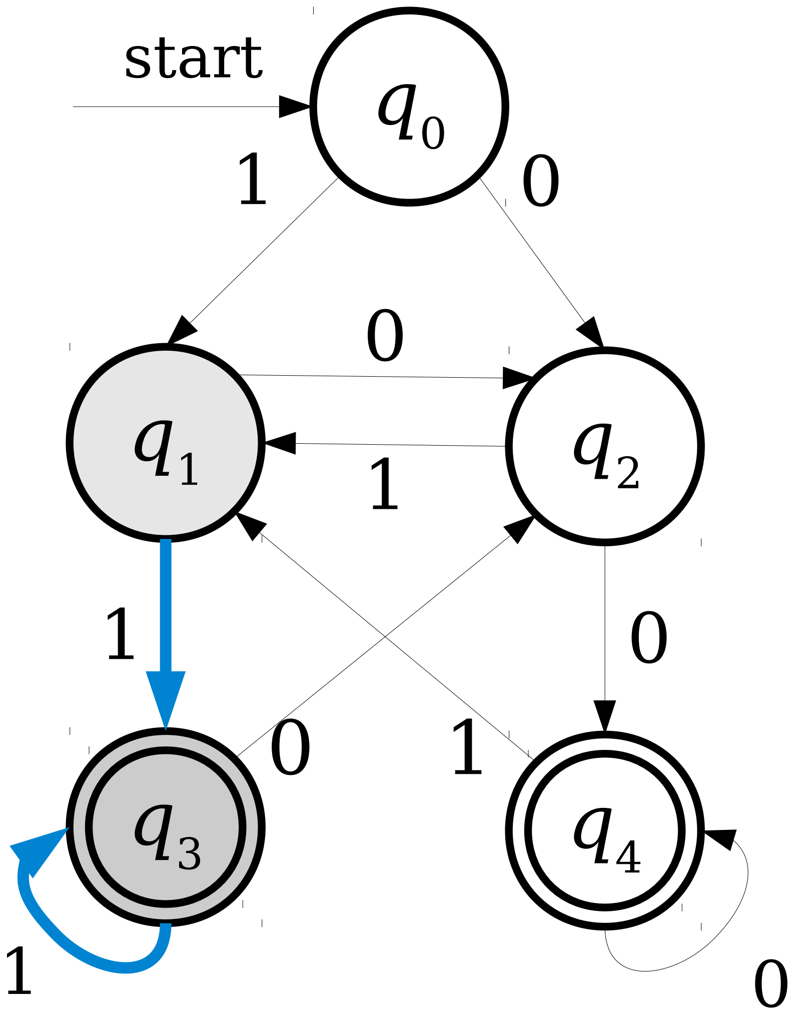
What Does This Accept?



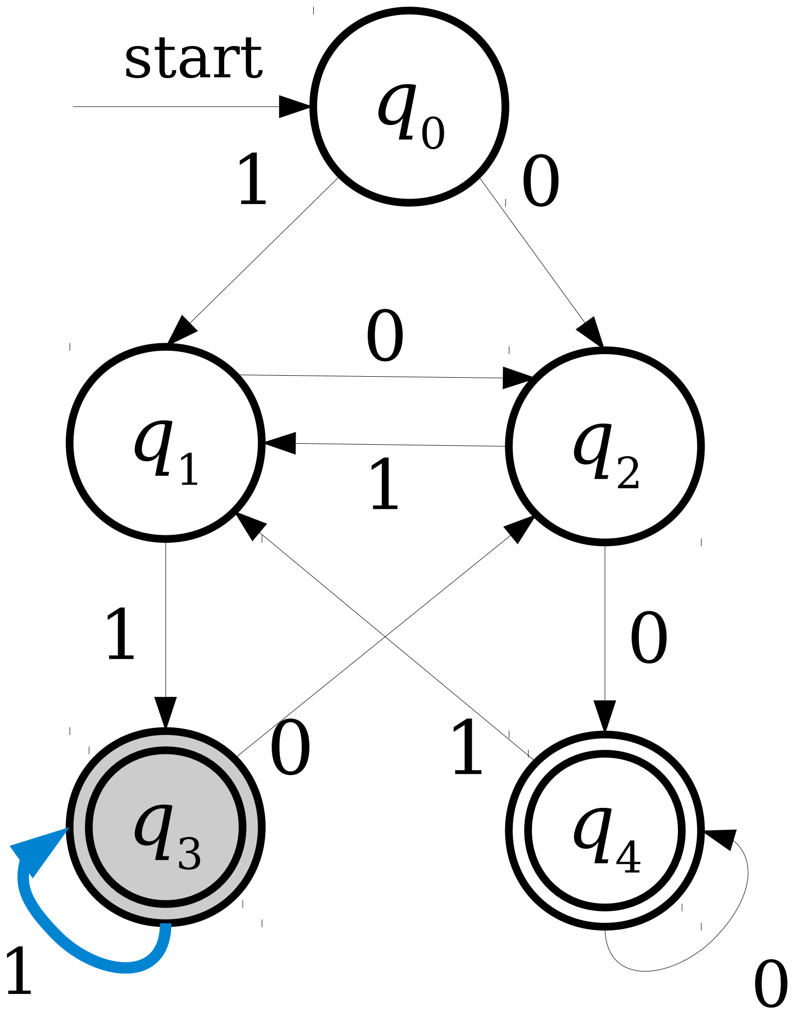
What Does This Accept?



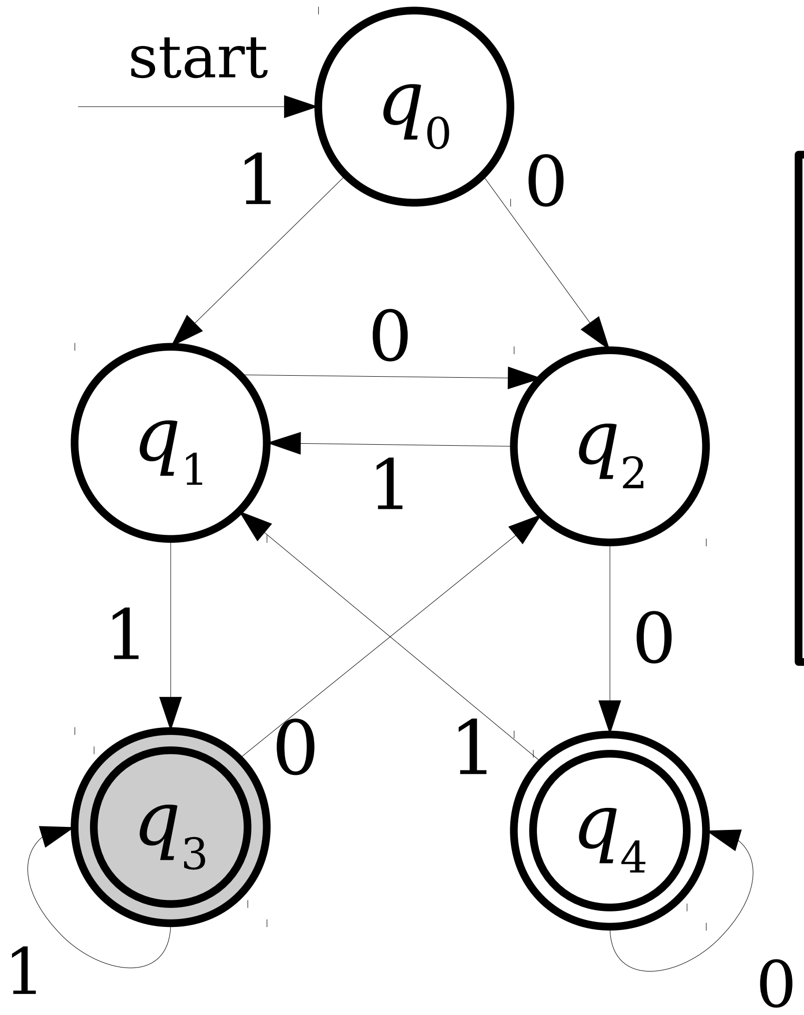
What Does This Accept?



What Does This Accept?

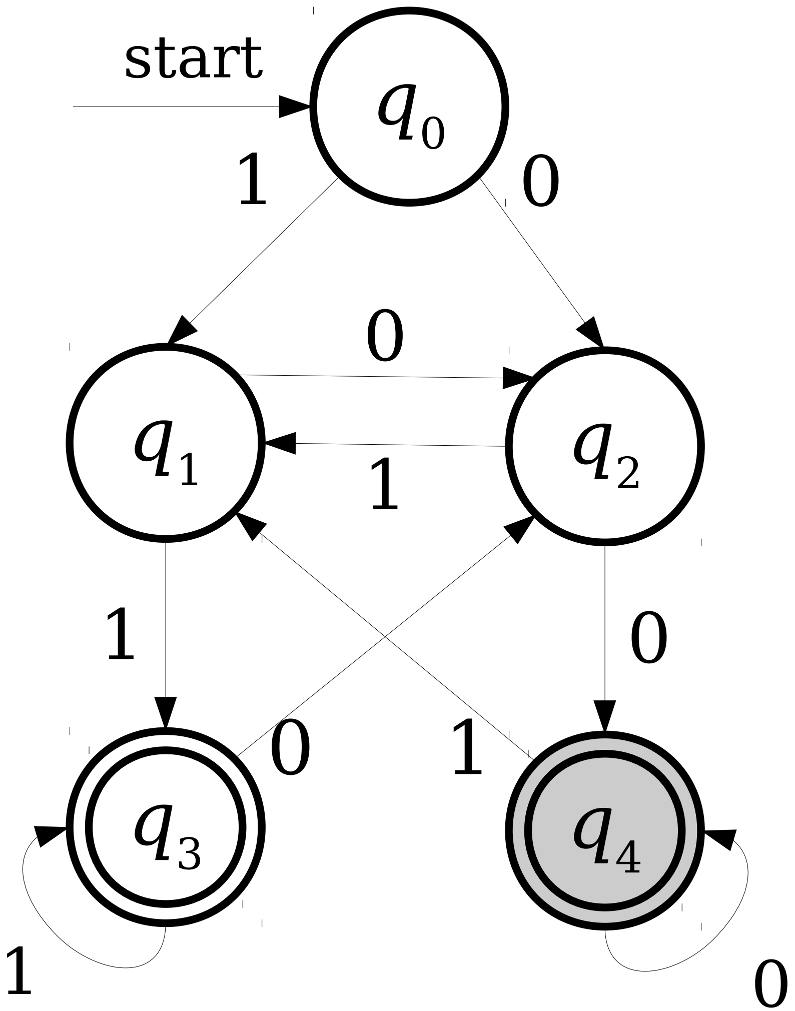


What Does This Accept?

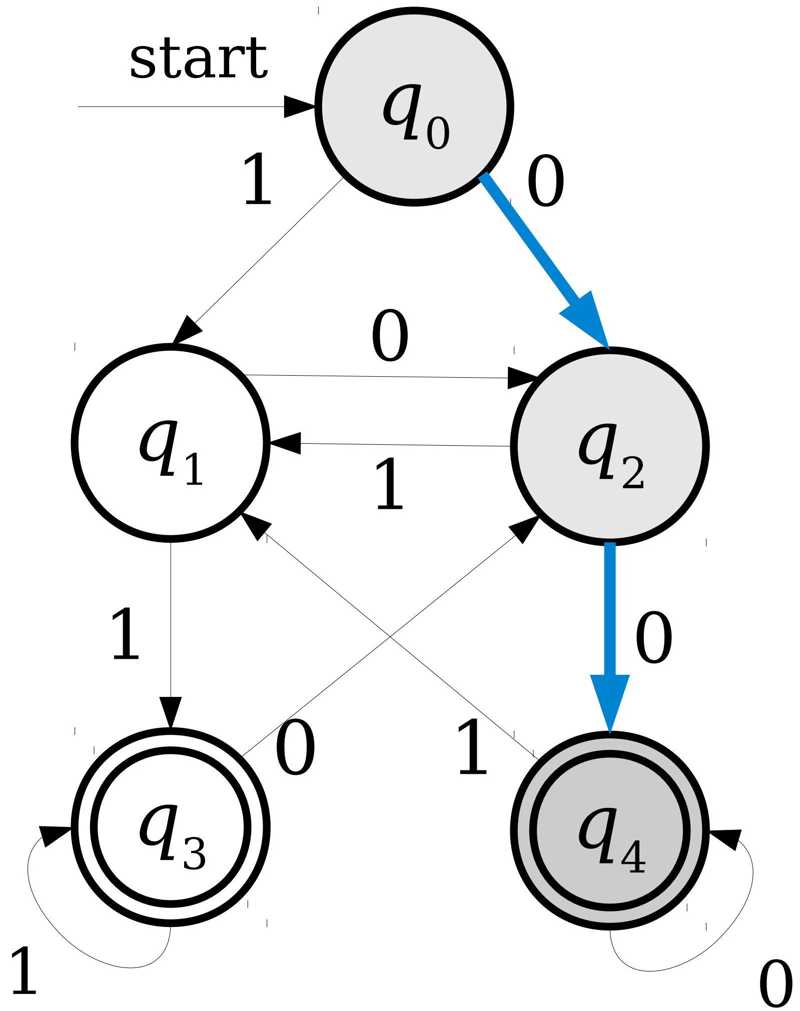


No matter where we start in the automaton, after seeing two 1's, we end up in accepting state q_3.

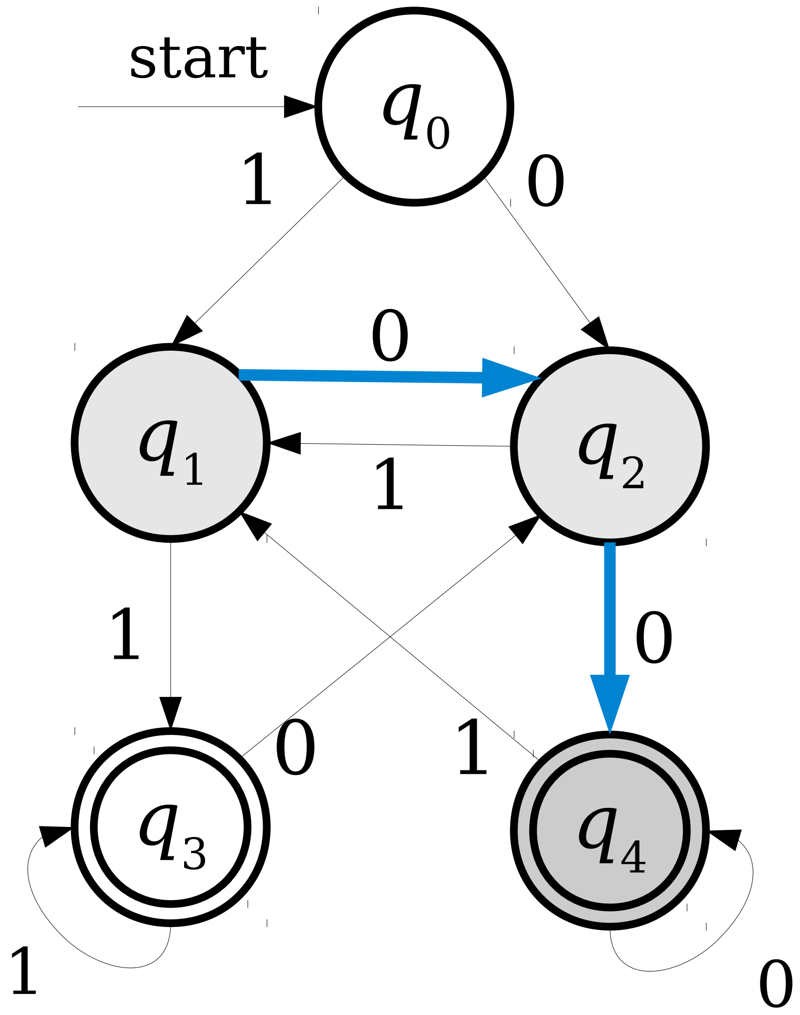
What Does This Accept?



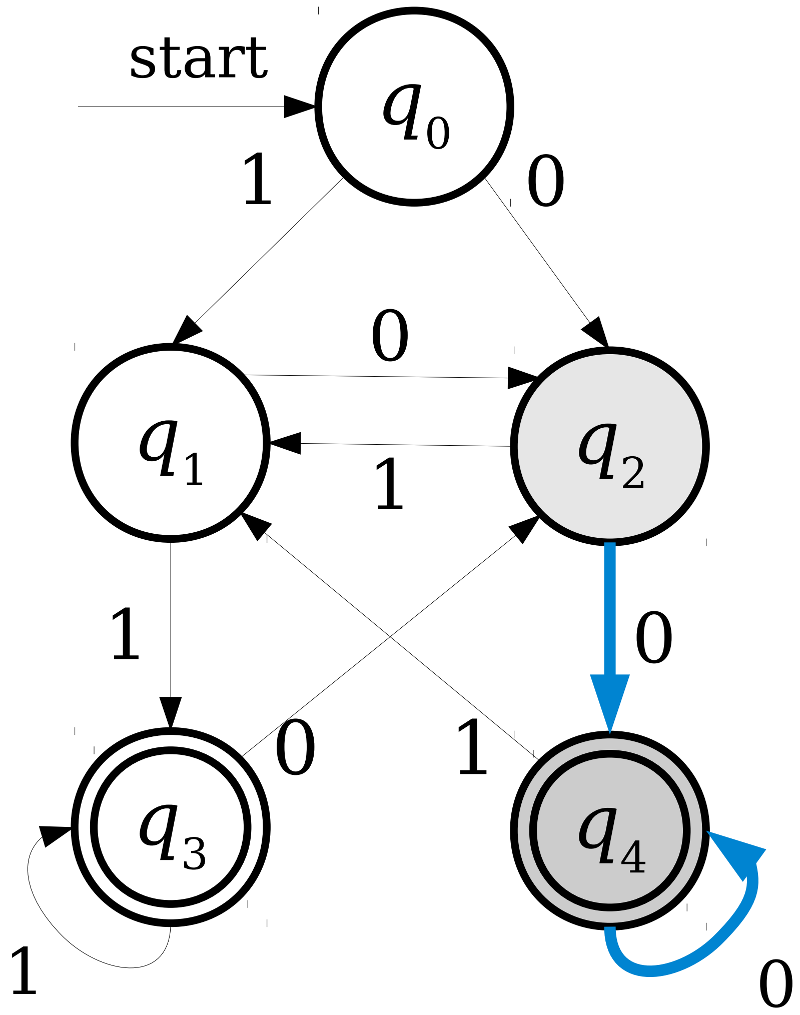
What Does This Accept?



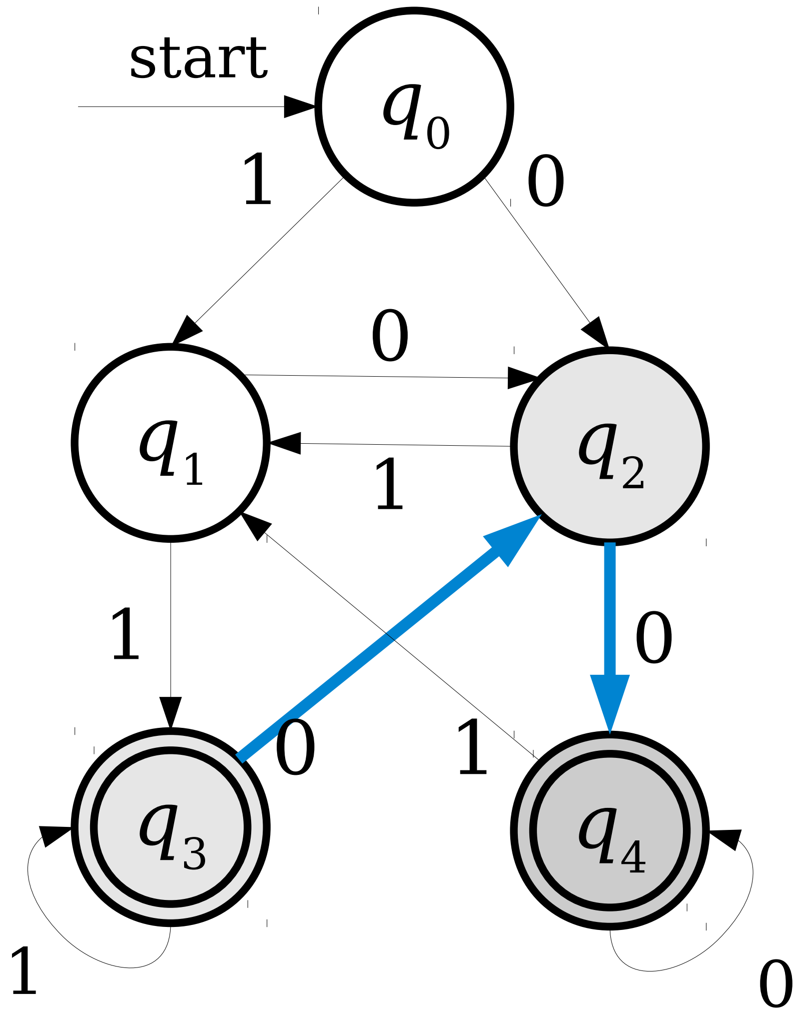
What Does This Accept?



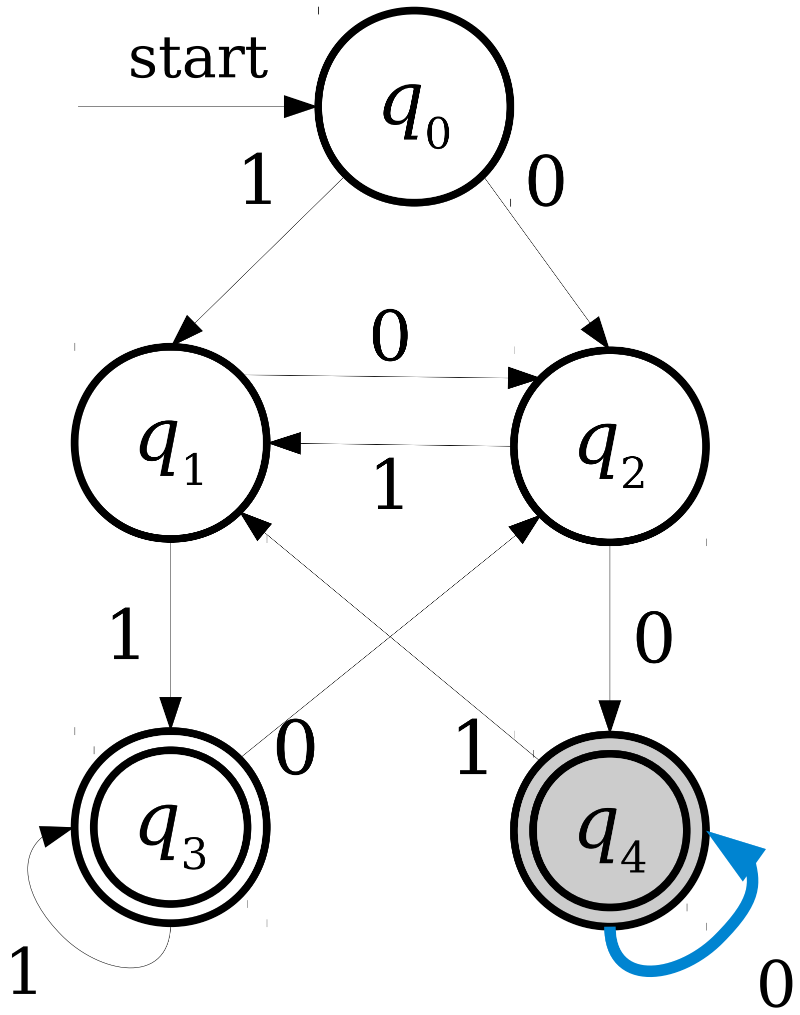
What Does This Accept?



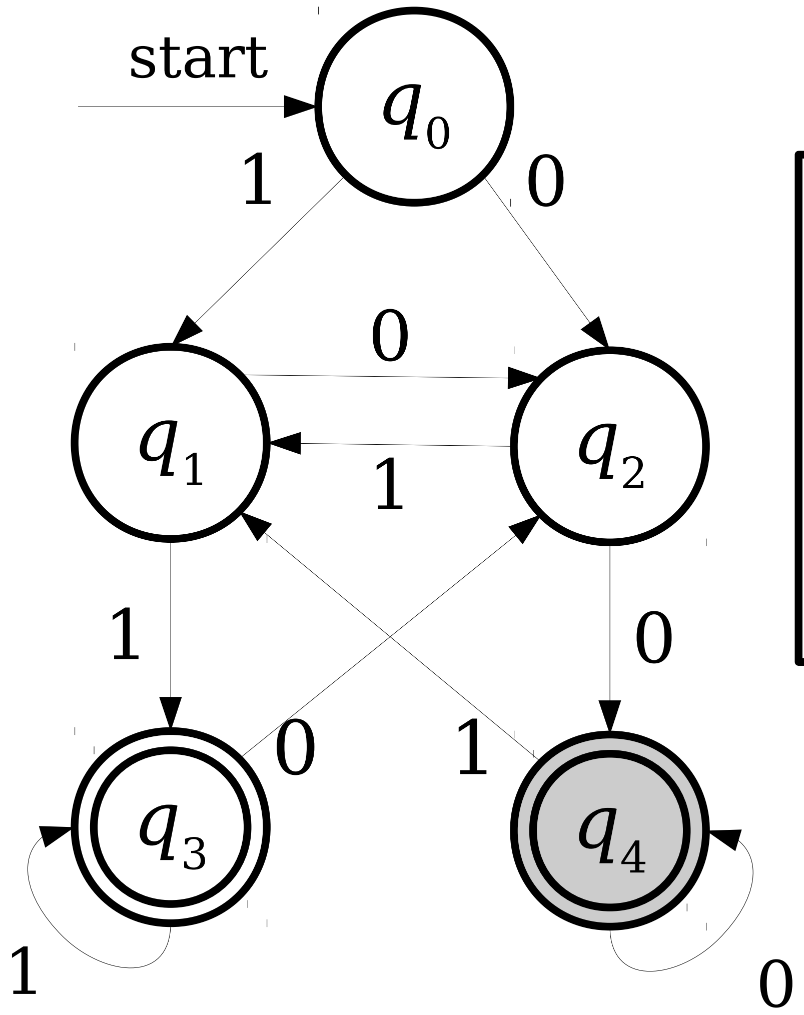
What Does This Accept?



What Does This Accept?

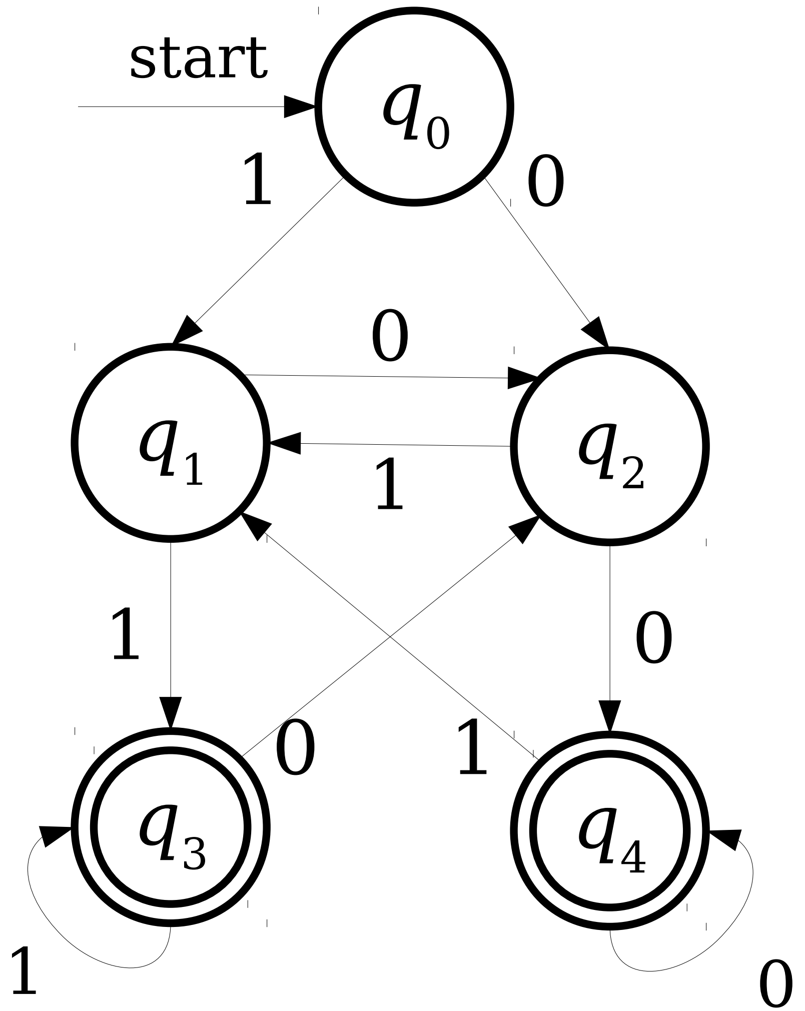


What Does This Accept?

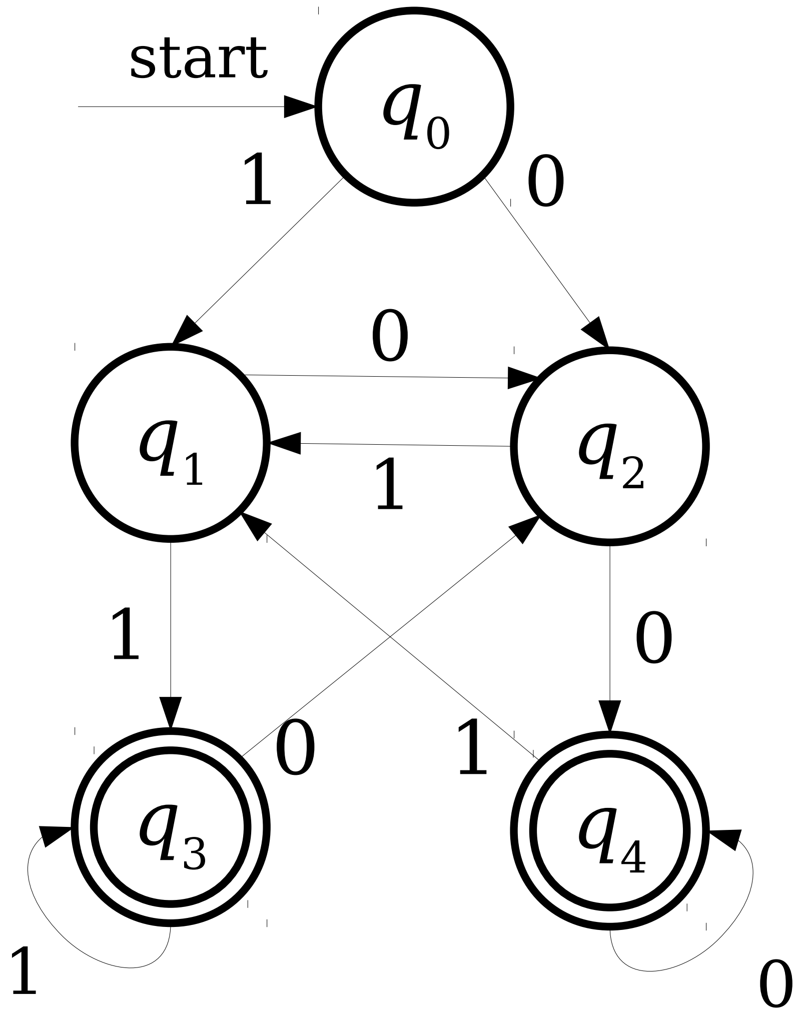


No matter where we start in the automaton, after seeing two 0's, we end up in accepting state q_5.

What Does This Accept?



What Does This Accept?



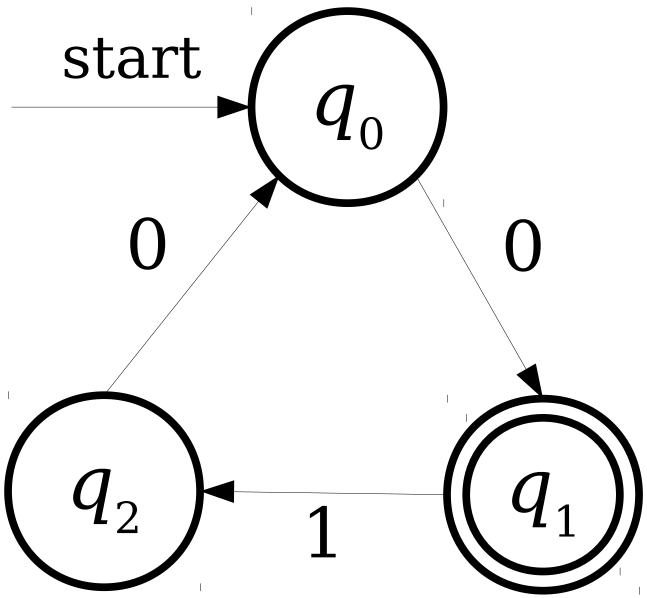
This automaton accepts a string iff the string ends in 00 or 11.

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

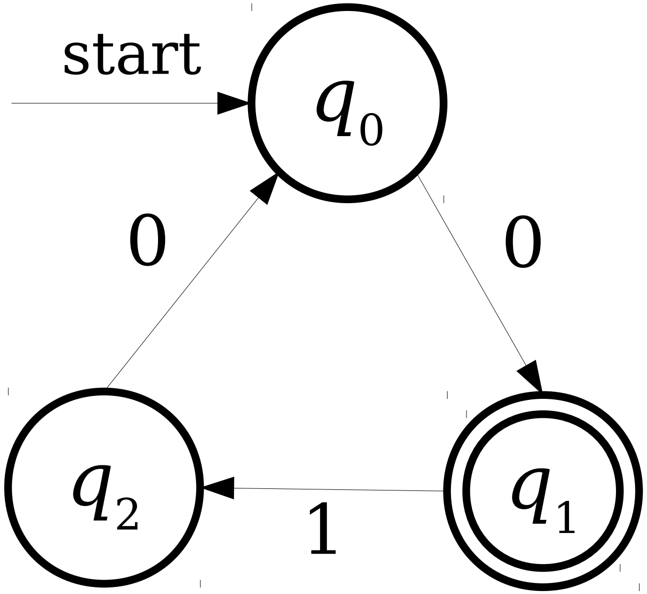
If D is an automaton, we denote the language of D as $\mathscr{L}$**(*D*)**.

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

A Small Problem

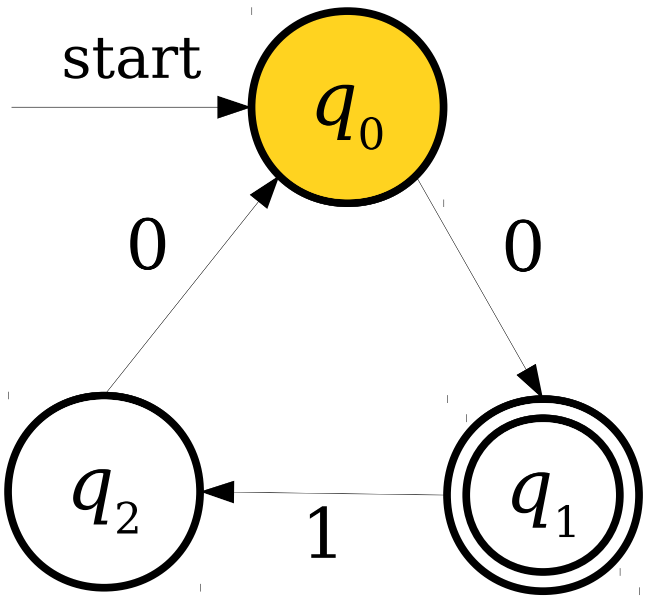


A Small Problem



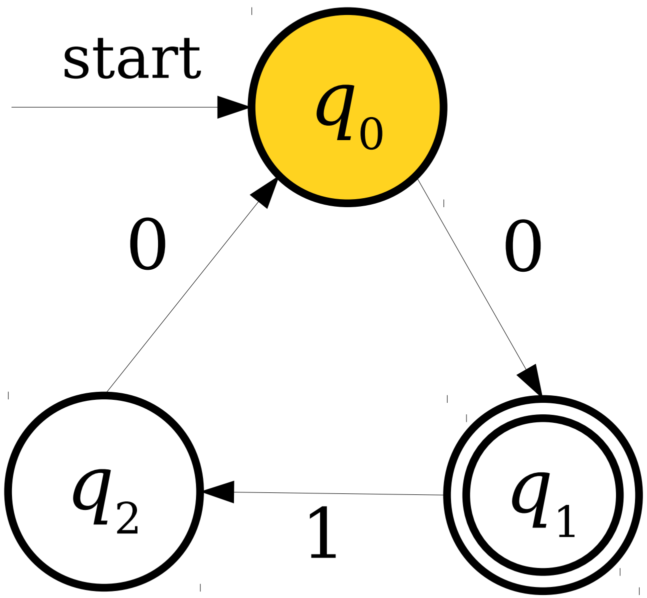
0 1 1 0

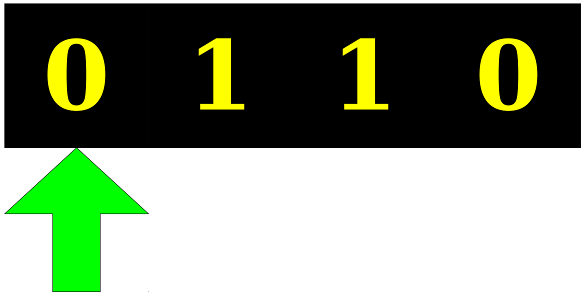
A Small Problem



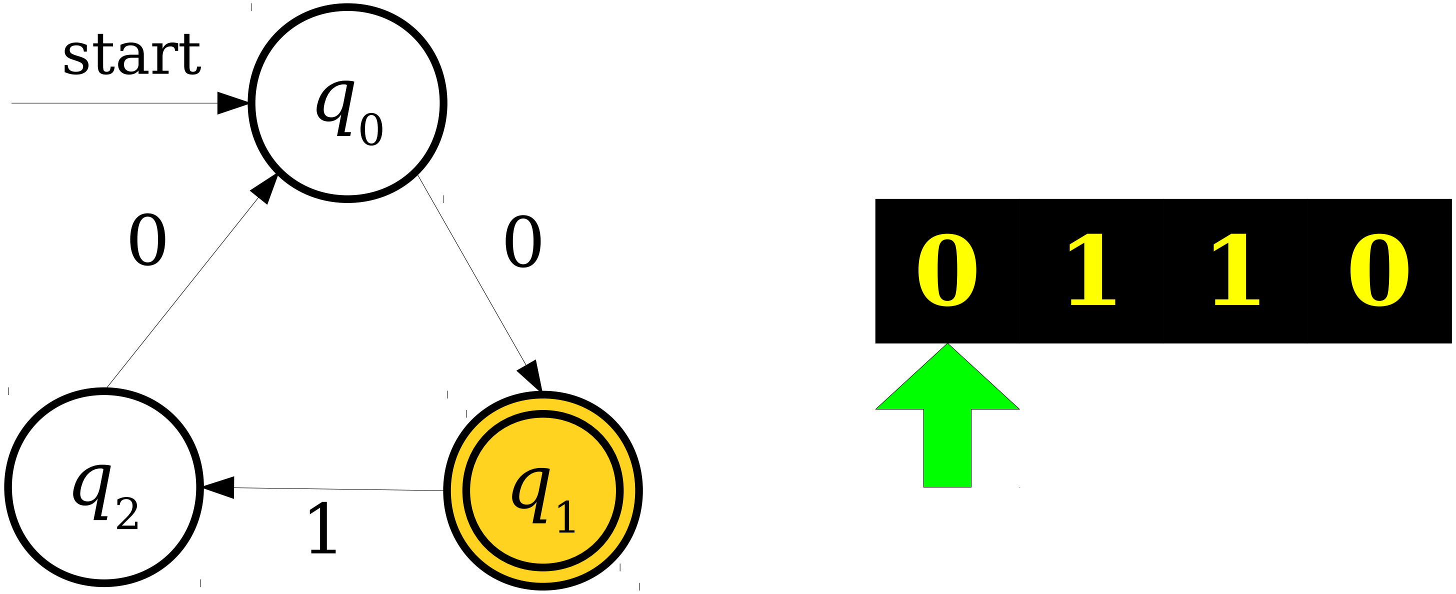
0 1 1 0

A Small Problem

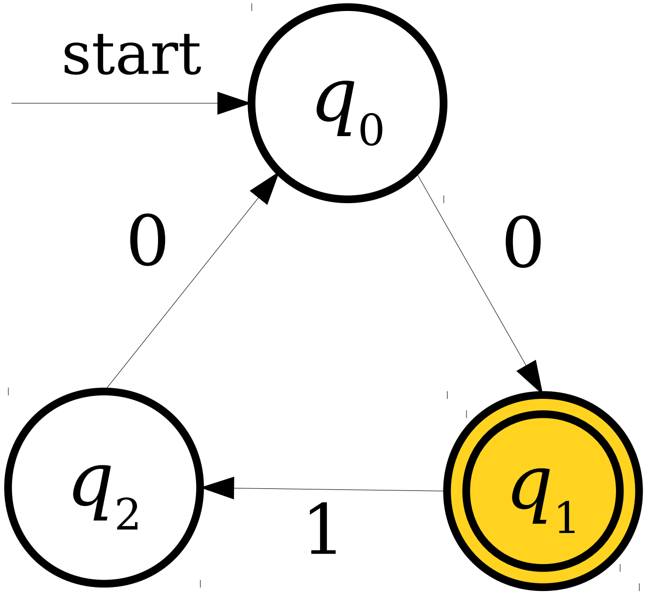


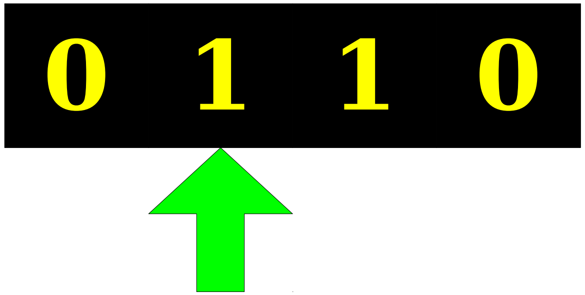


A Small Problem

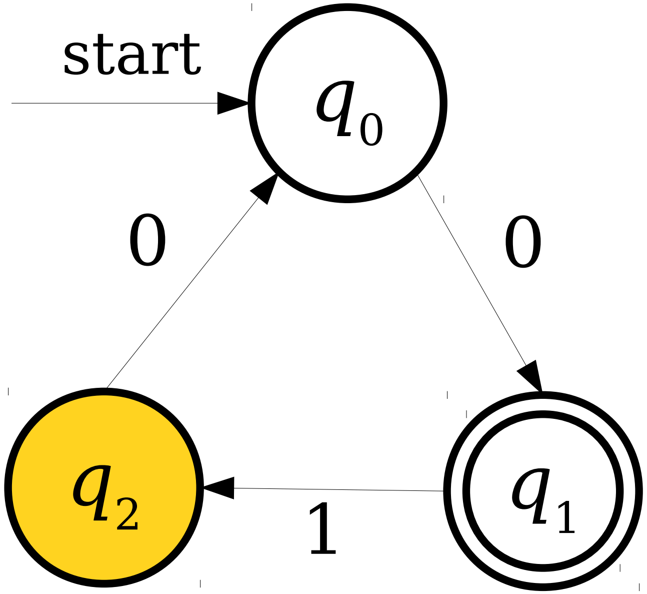


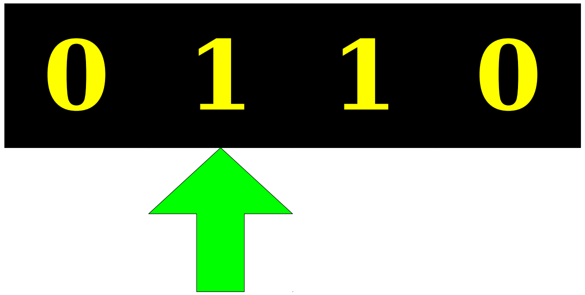
A Small Problem





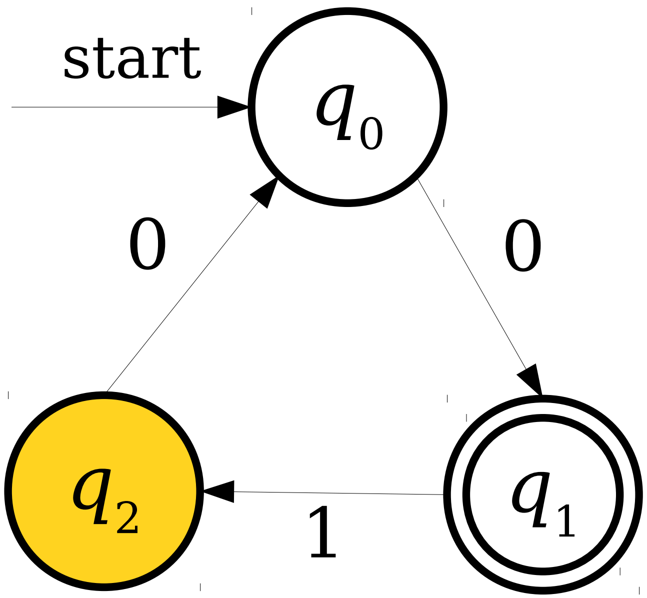
A Small Problem

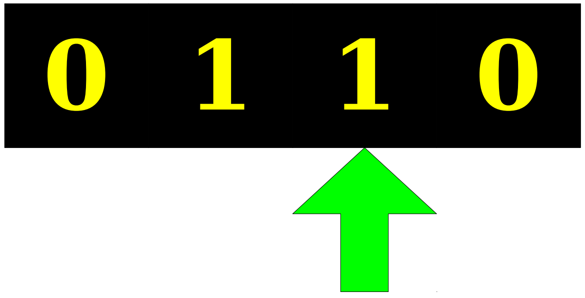




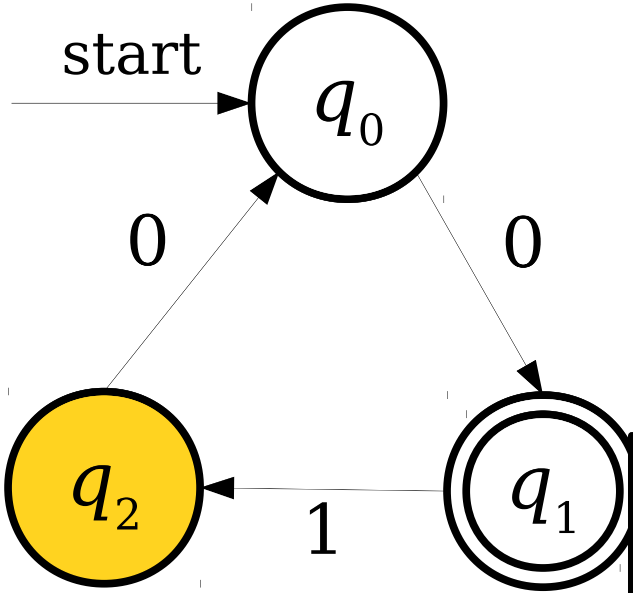
0 1 1 0

A Small Problem

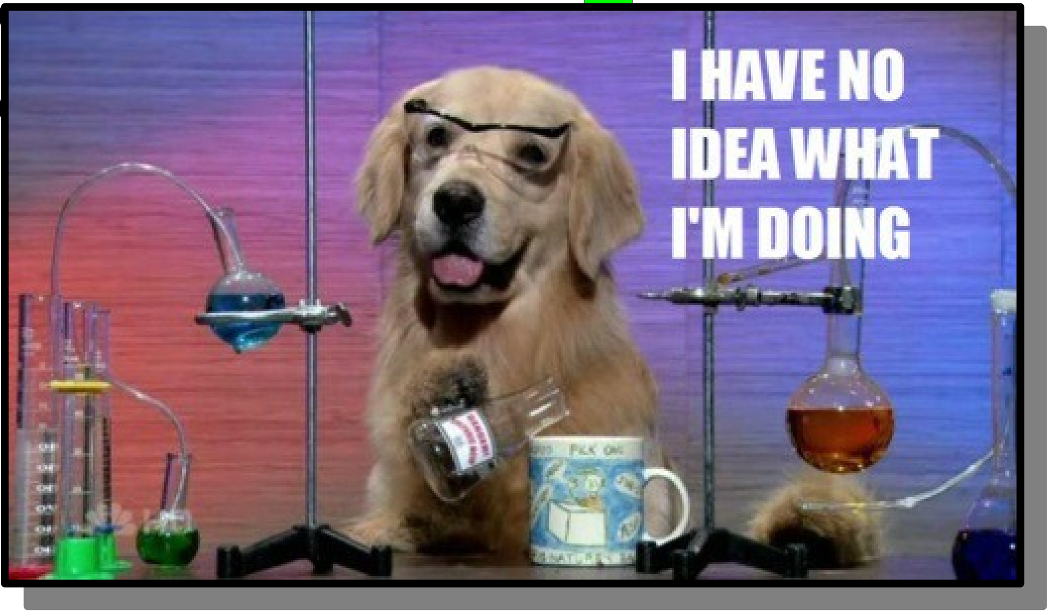




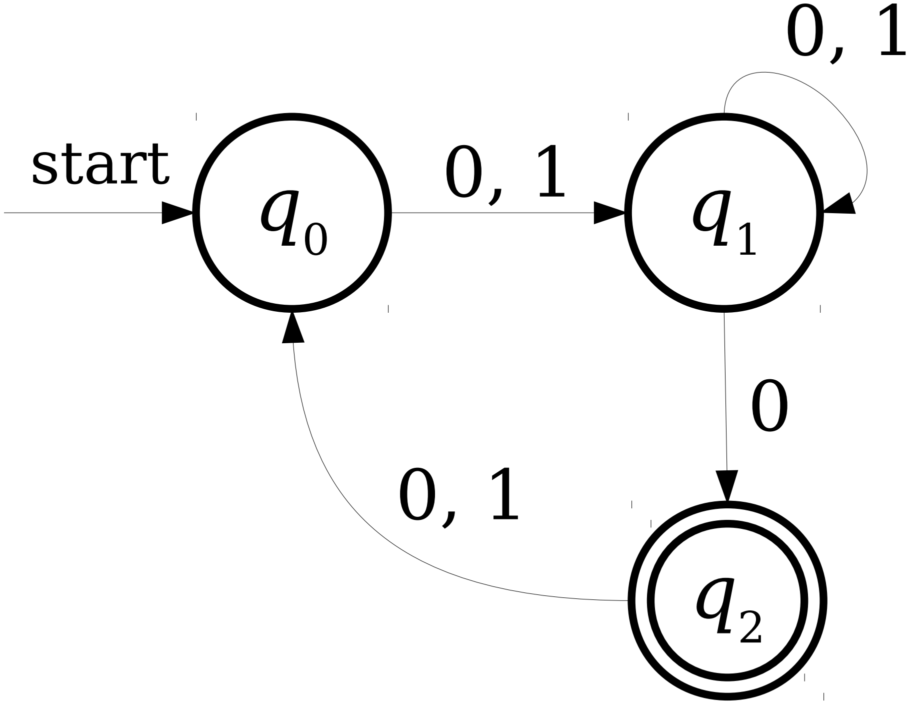
A Small Problem



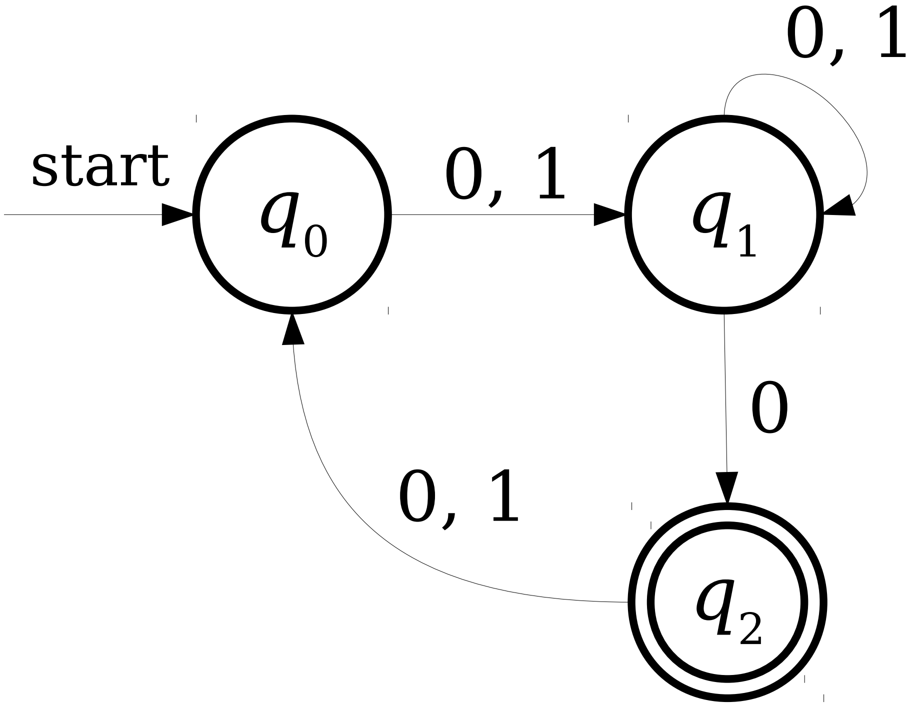
0 1 1 0



Another Small Problem

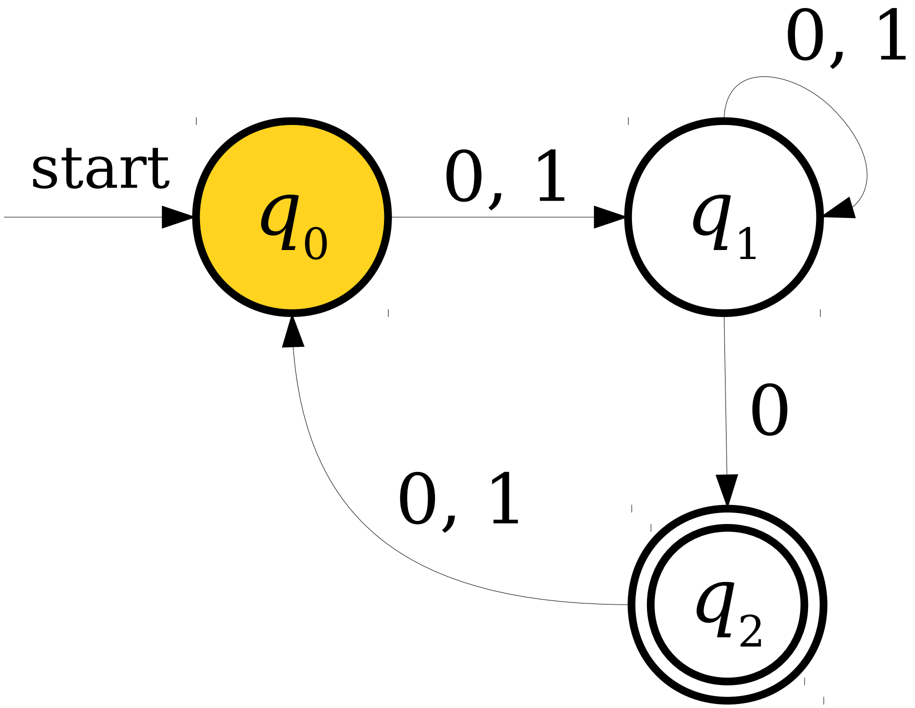


Another Small Problem



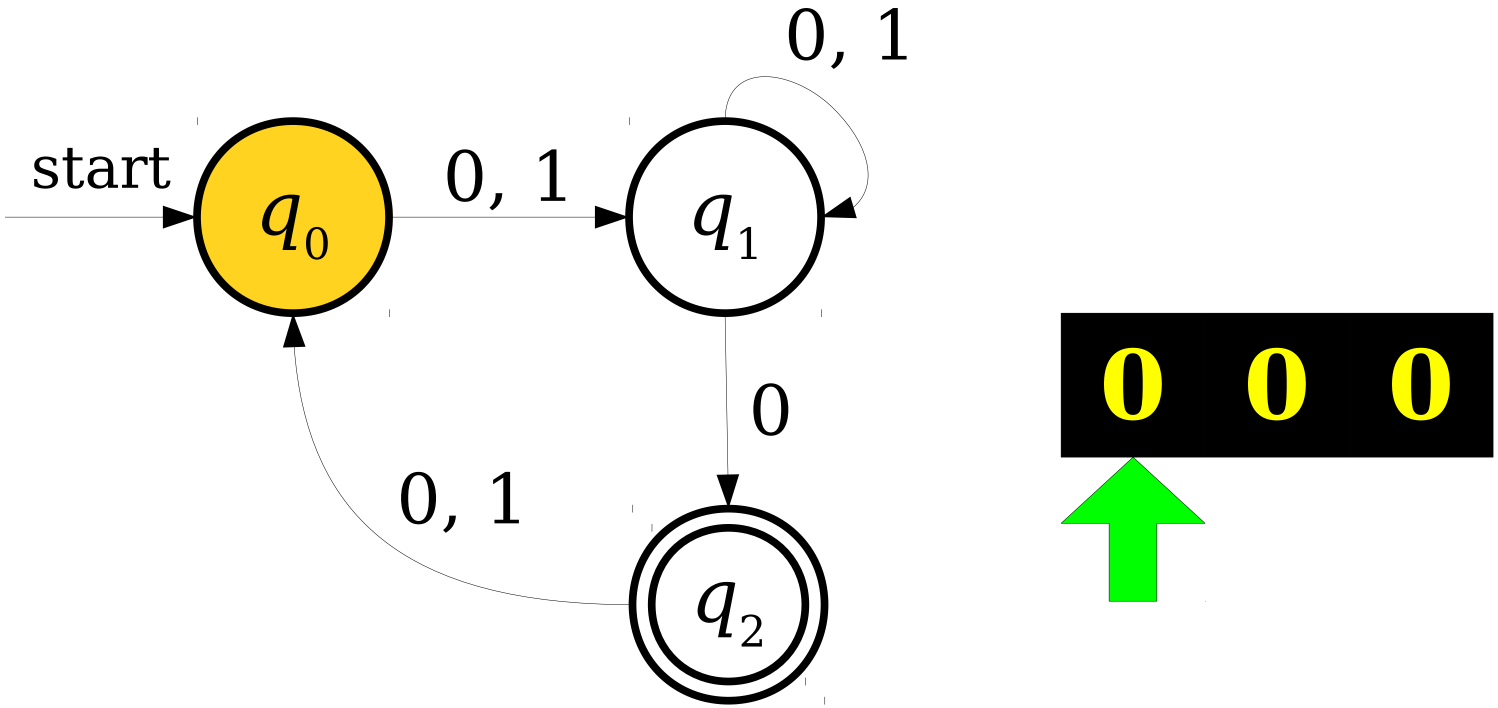
0 0 0

Another Small Problem

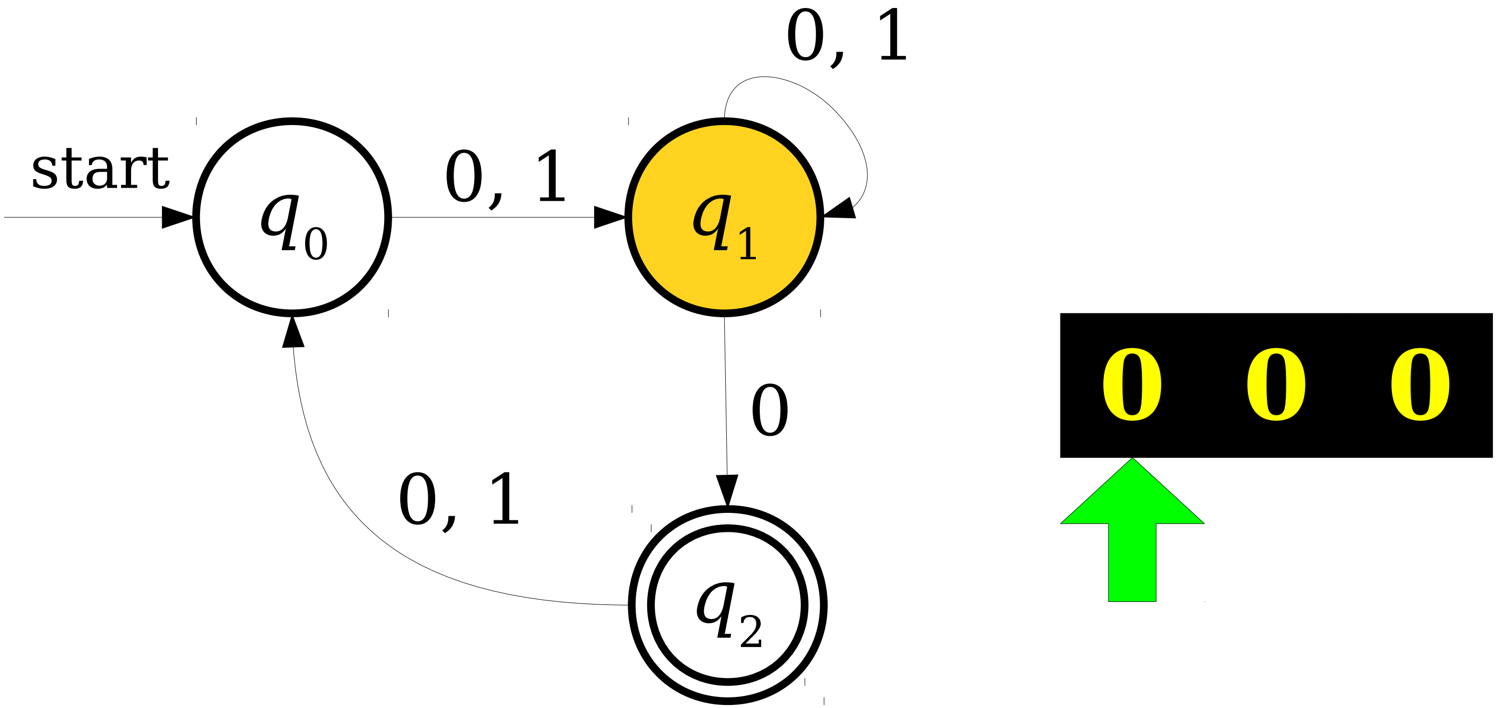


0 0 0

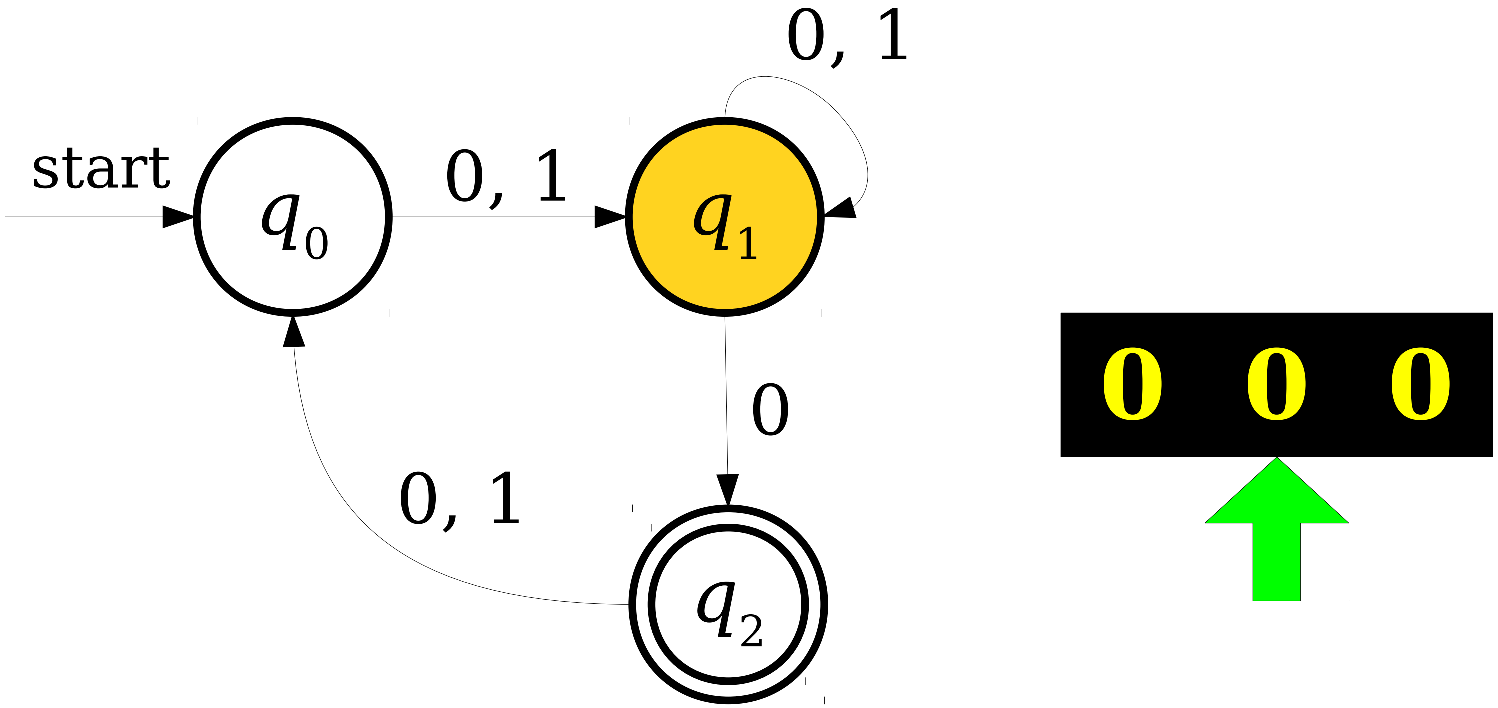
Another Small Problem



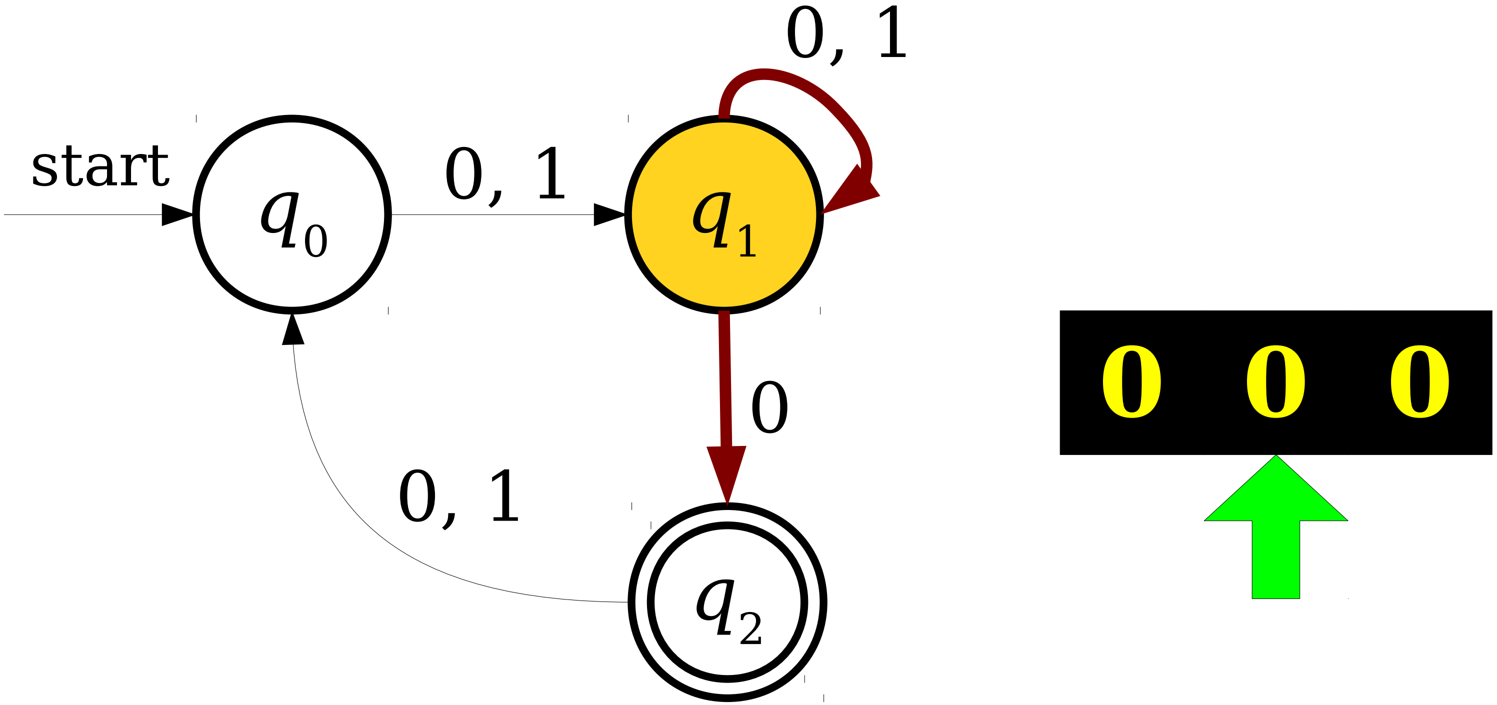
Another Small Problem



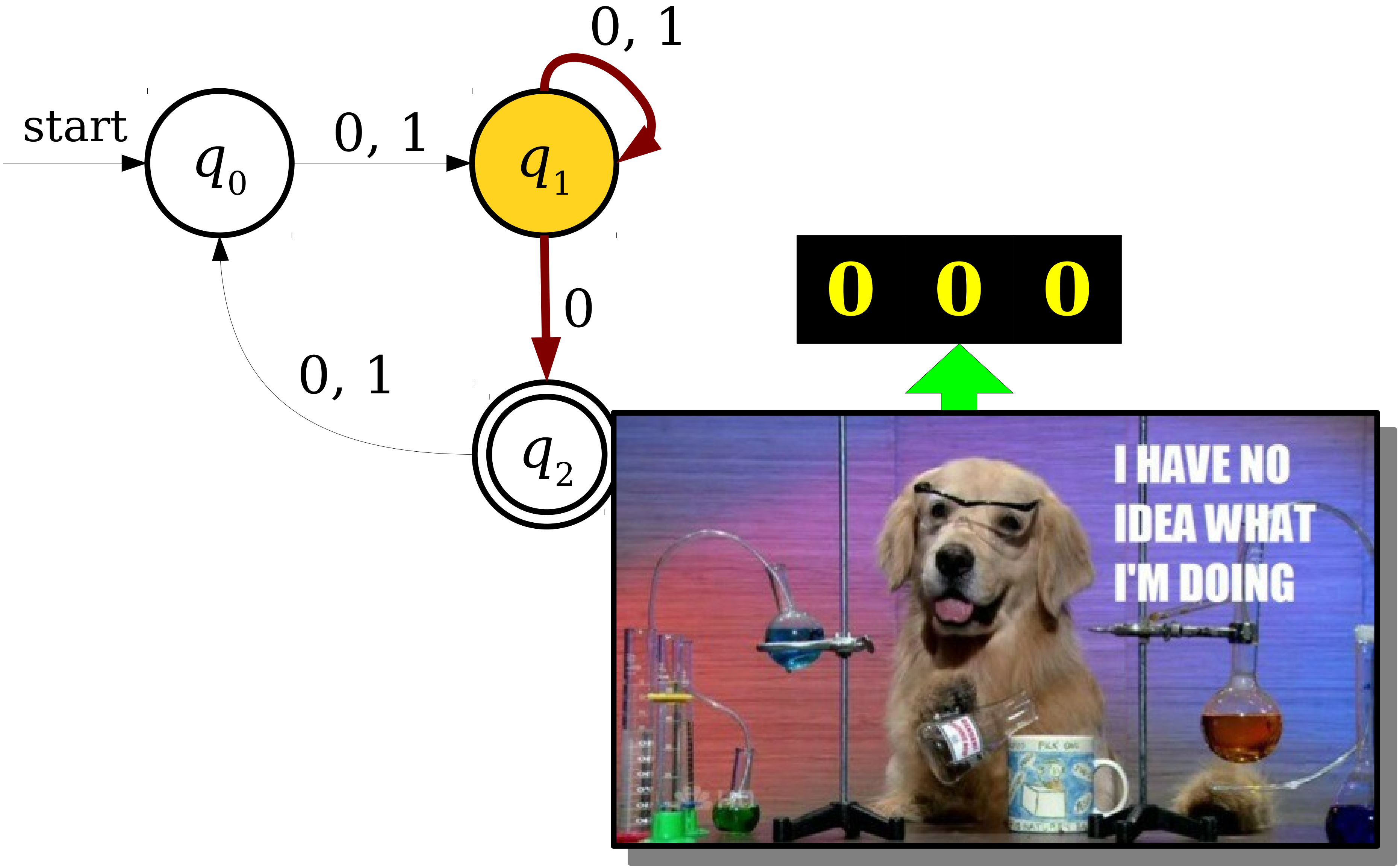
Another Small Problem



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

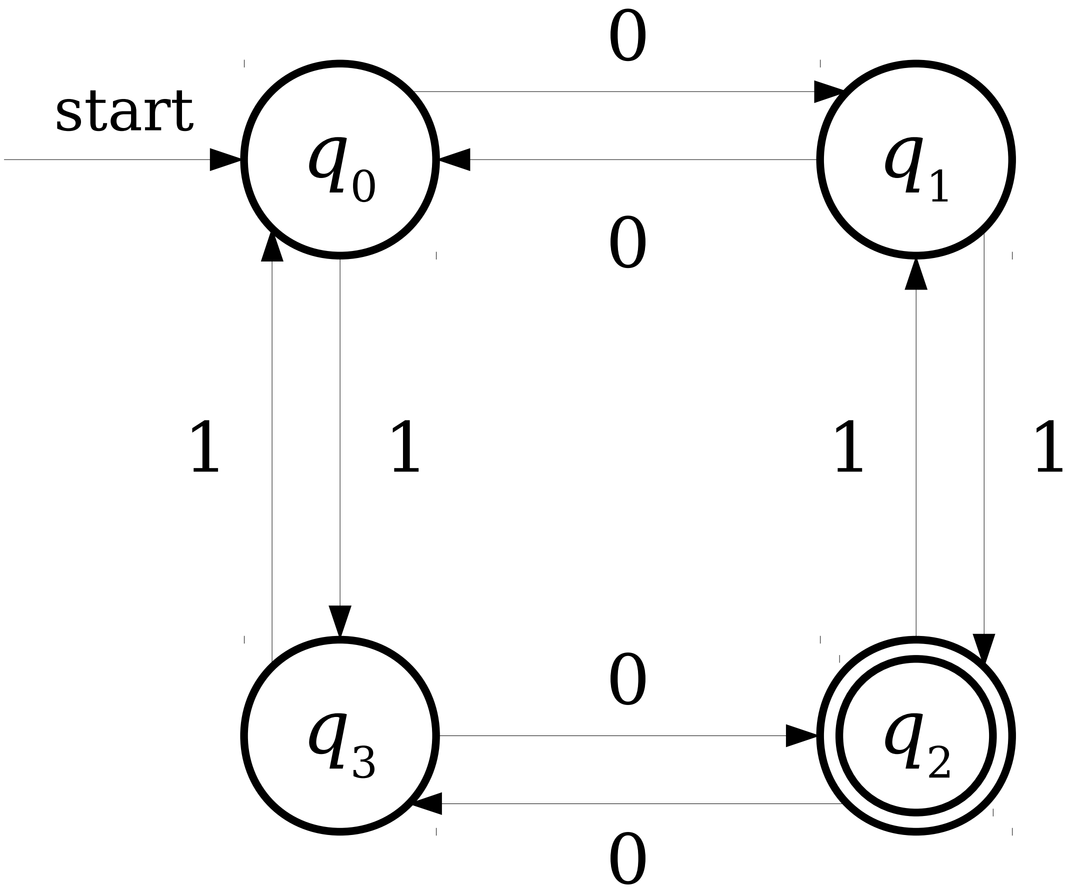
- A ***DFA*** is a
  - ***D***eterministic
  - ***F***inite
  - ***A***utomaton
- DFAs are the simplest type of automaton that we will see in this course.

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.

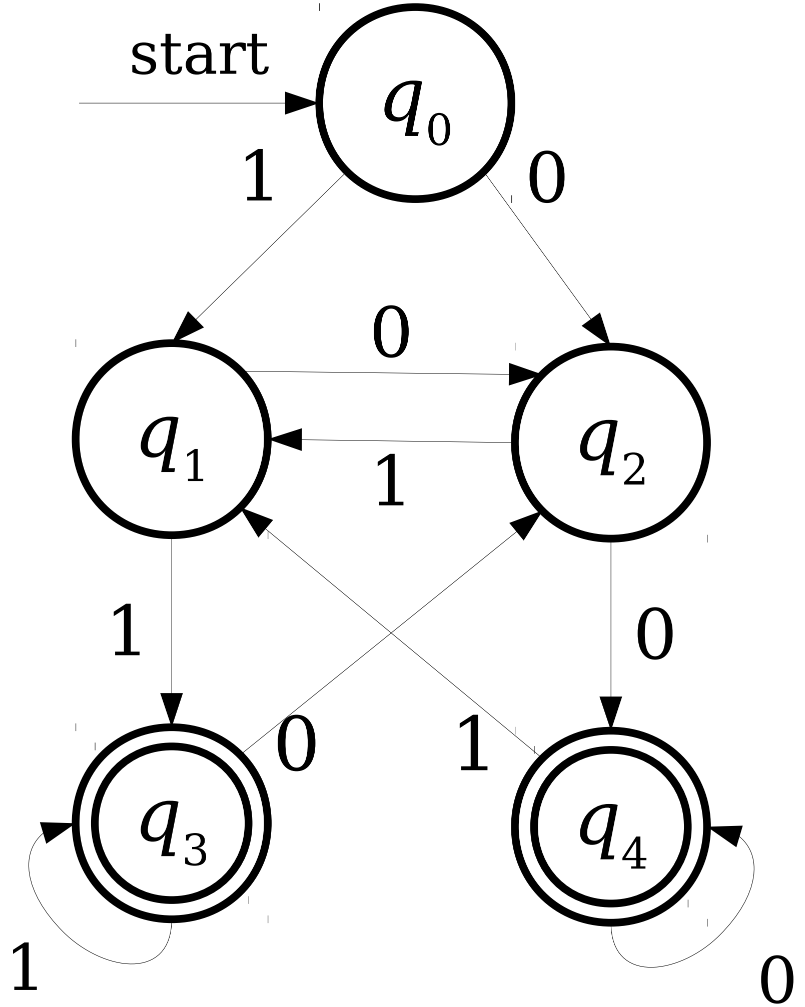
Is this a DFA over {0, 1}?

Is this a DFA over {0, 1}?



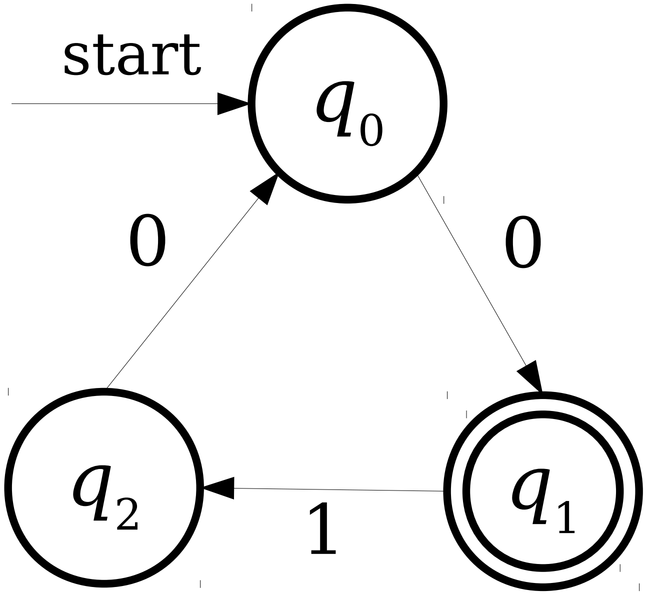
Is this a DFA over $\{0, 1\}$?

Is this a DFA over {0, 1}?

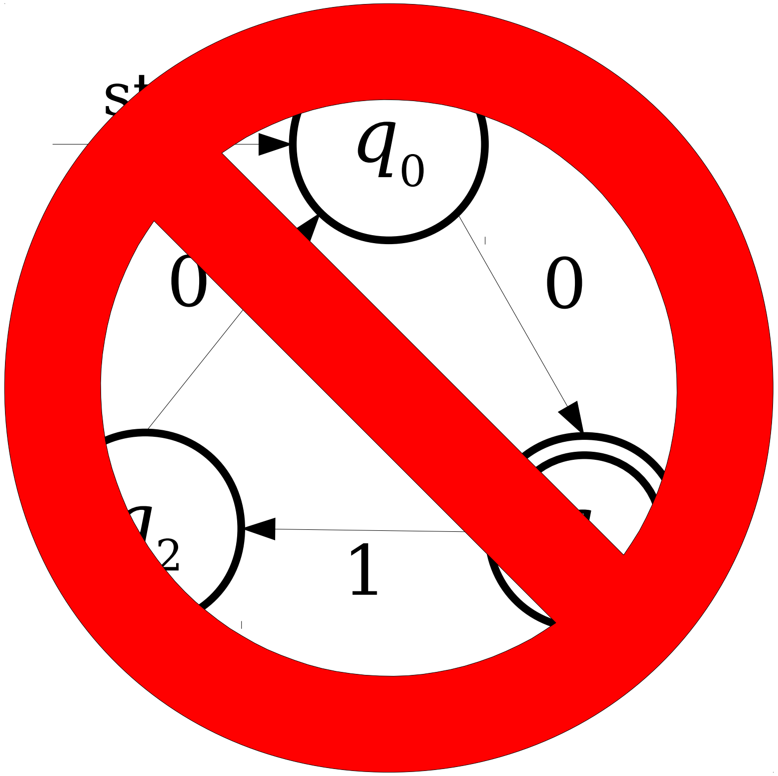


Is this a DFA over {0, 1}?

Is this a DFA over {0, 1}?

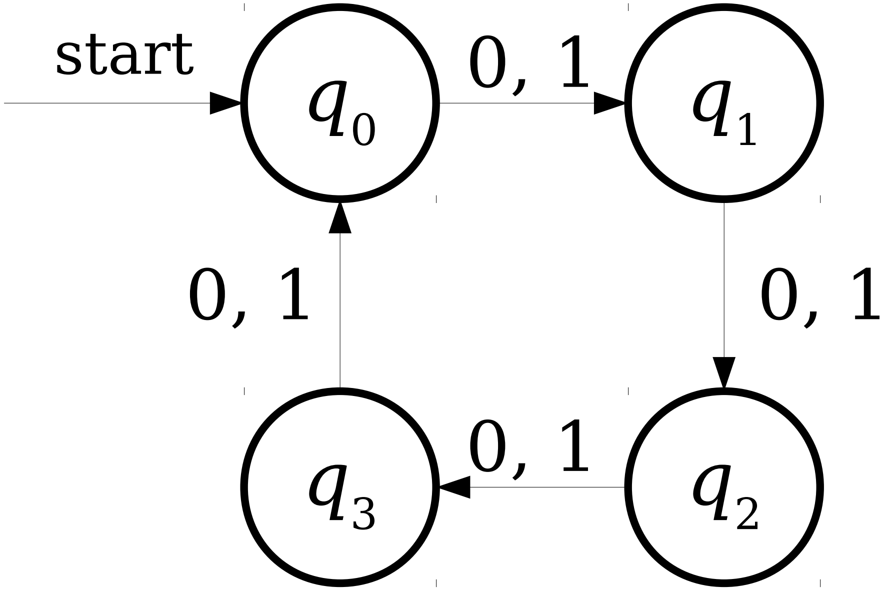


Is this a DFA over {0, 1}?



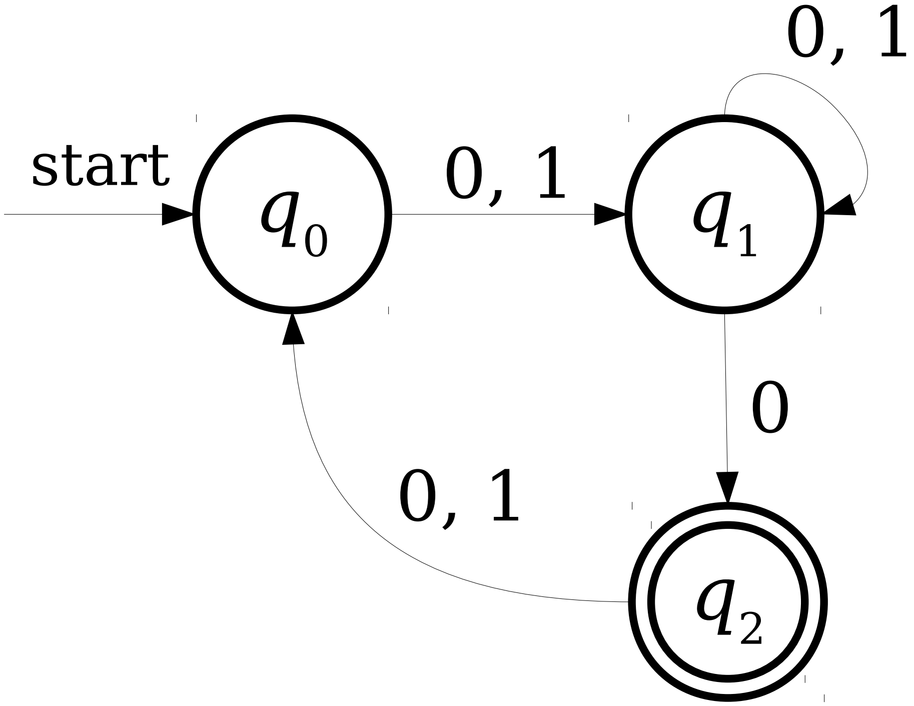
Is this a DFA over {0, 1}?

Is this a DFA over {0, 1}?

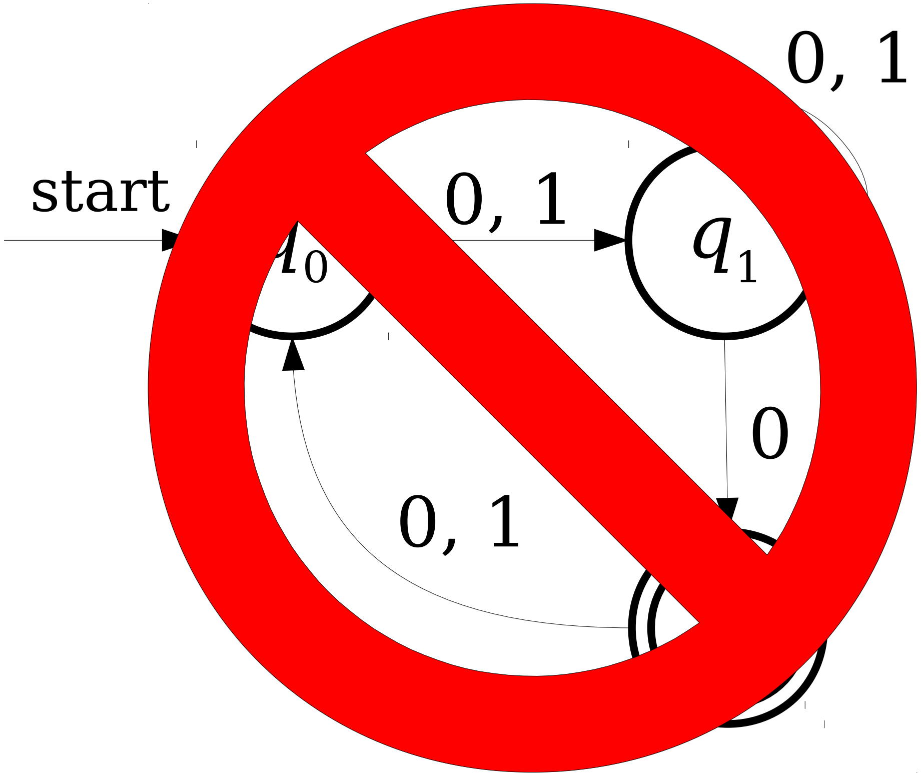


Is this a DFA over {0, 1}?

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

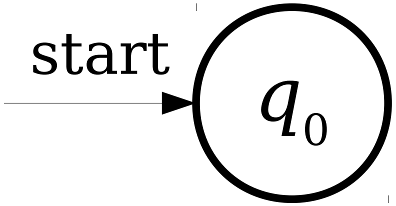
- At each point in its execution, the DFA can only remember what state it is in.
- ***DFA Design Tip:*** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only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

Recognizing Languages with DFAs

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

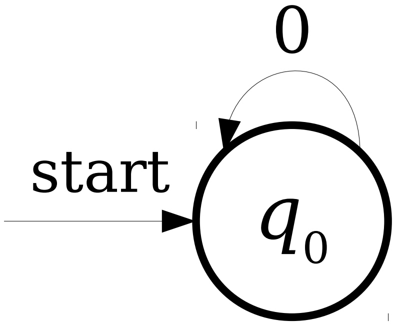
Recognizing Languages with DFAs

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



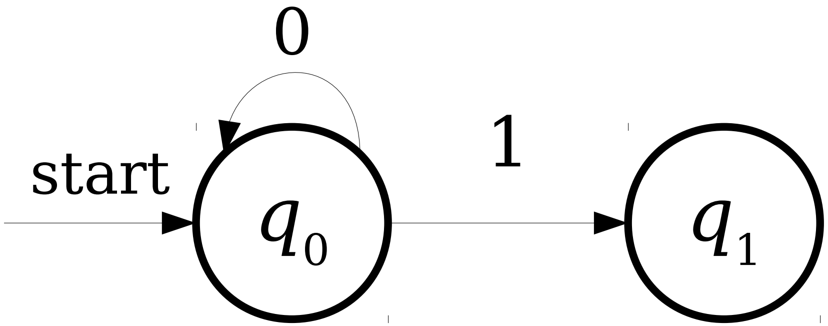
Recognizing Languages with DFAs

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



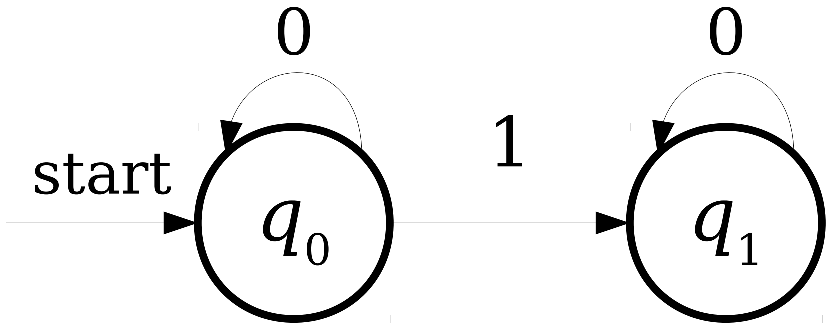
Recognizing Languages with DFAs

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



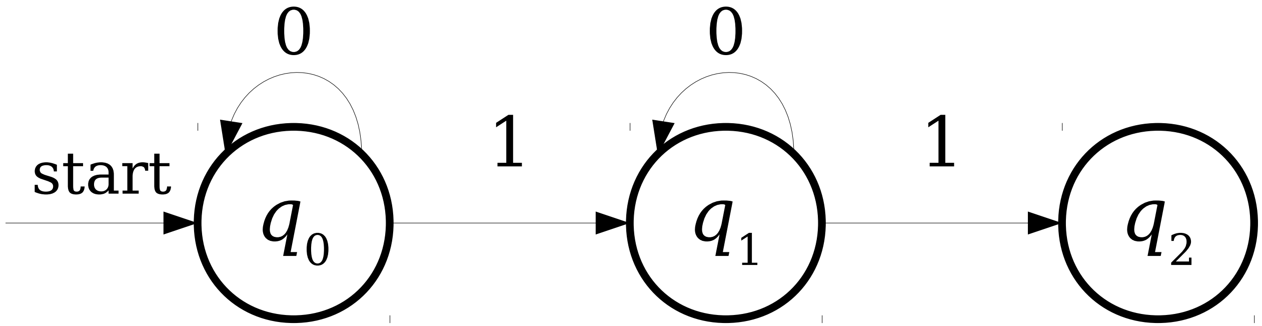
Recognizing Languages with DFAs

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



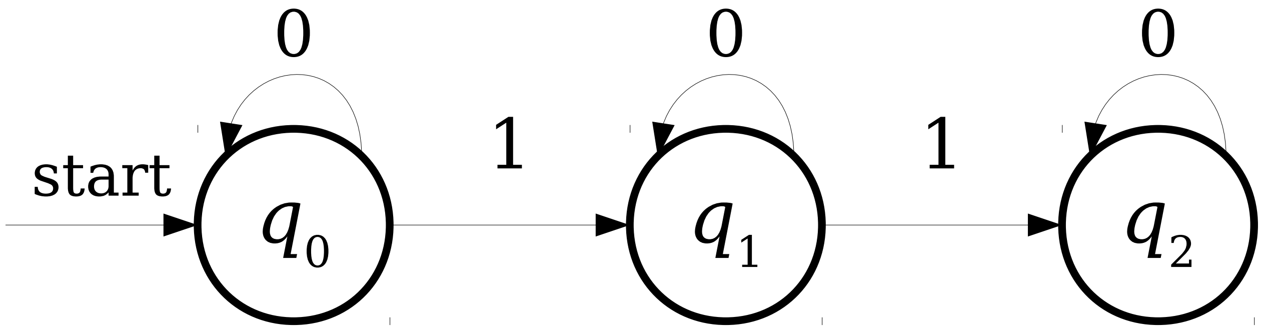
Recognizing Languages with DFAs

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



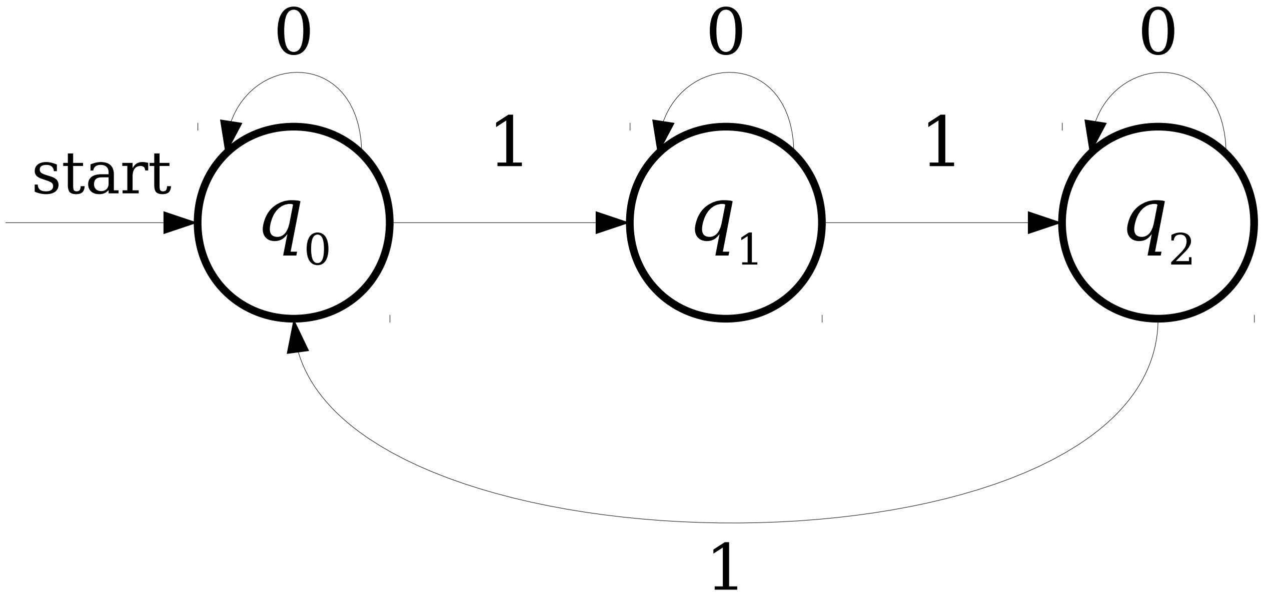
Recognizing Languages with DFAs

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



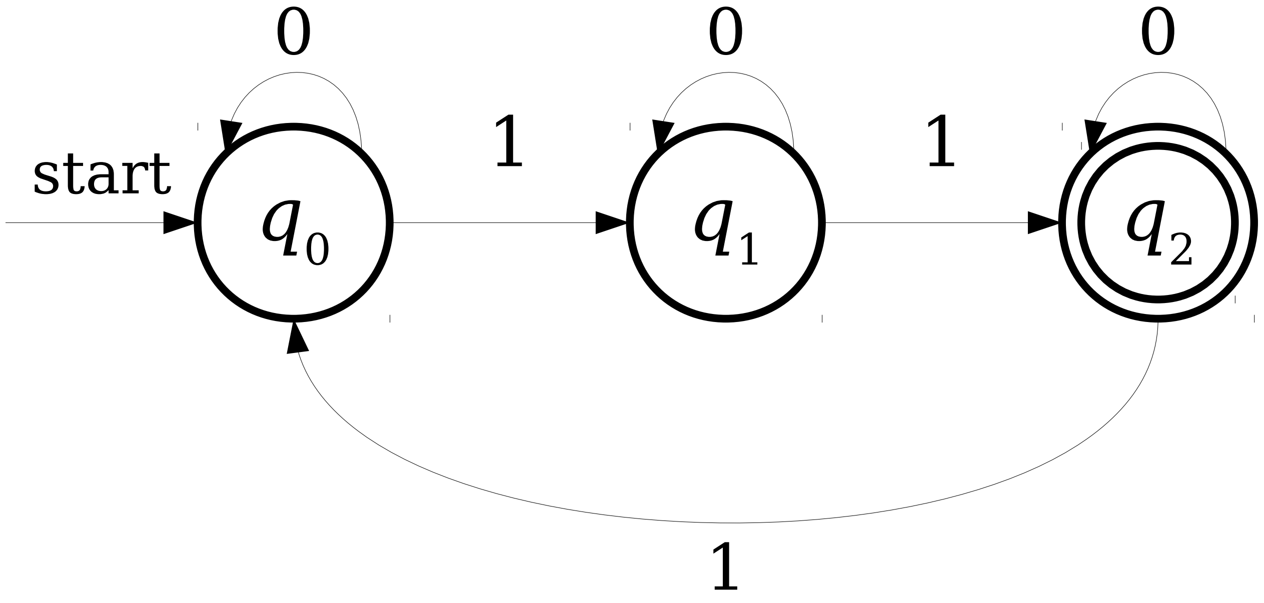
Recognizing Languages with DFAs

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



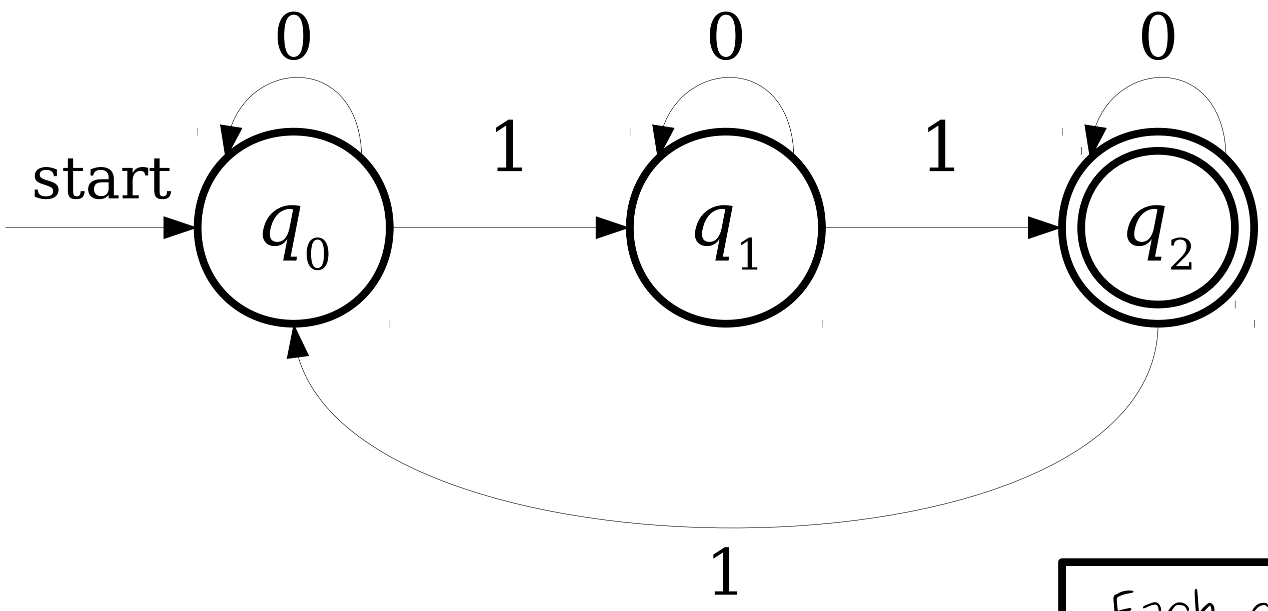
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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



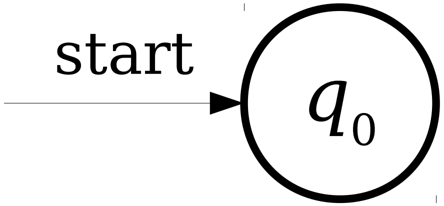
Each state remembers the remainder of the number of 1's seen so far modulo three.

Recognizing Languages with DFAs

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

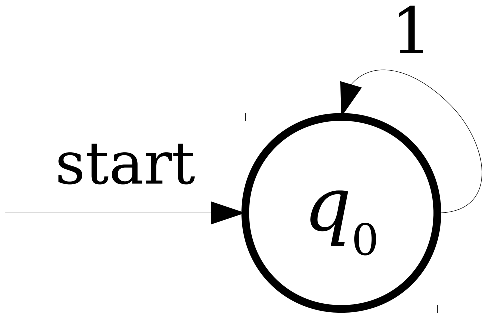
Recognizing Languages with DFAs

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



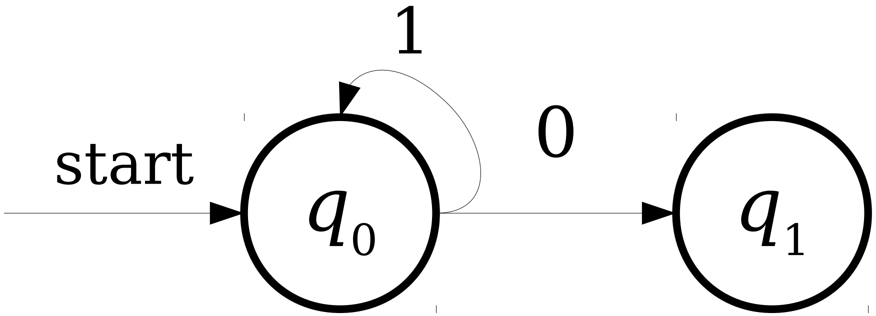
Recognizing Languages with DFAs

L = { w ∈ {0, 1}* | w contains 00 as a substring }



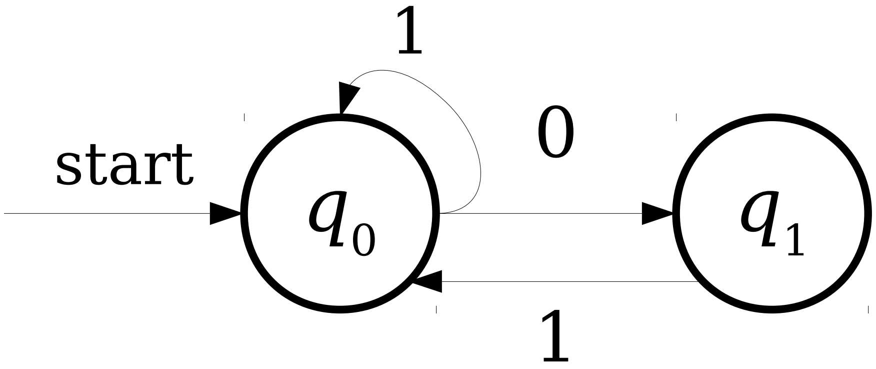
Recognizing Languages with DFAs

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



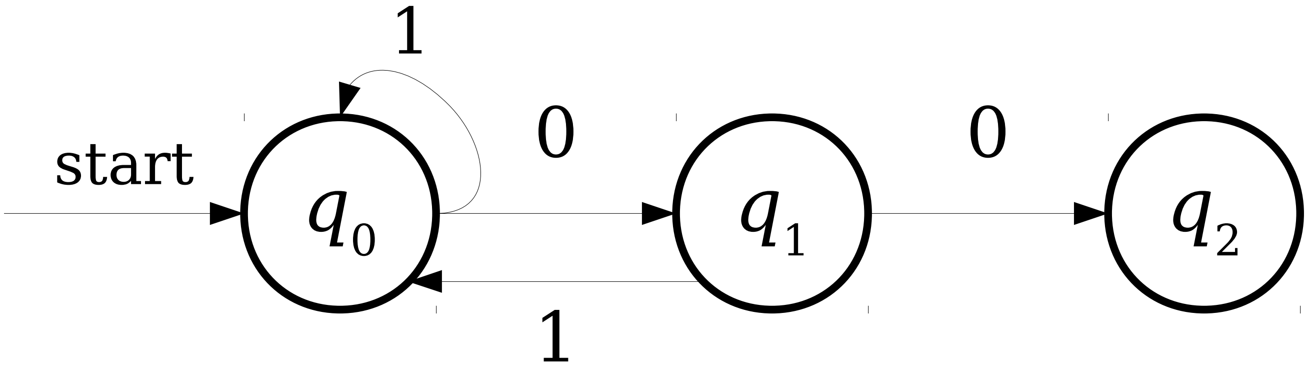
Recognizing Languages with DFAs

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



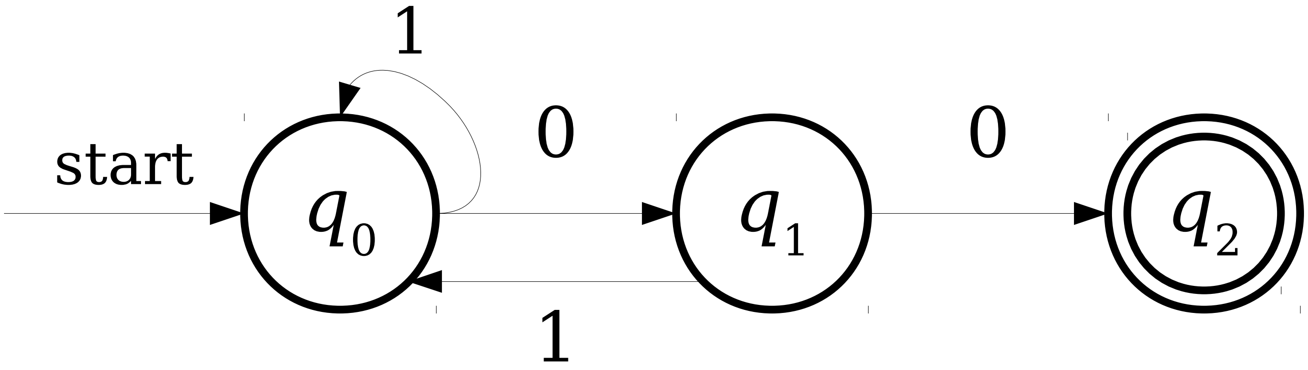
Recognizing Languages with DFAs

L = { w ∈ {0, 1}* | w contains 00 as a substring }



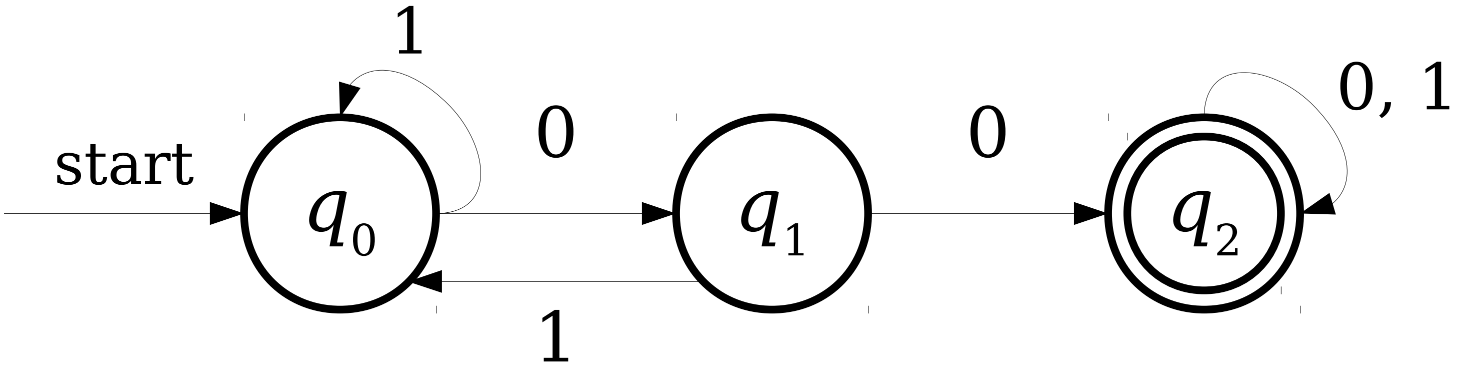
Recognizing Languages with DFAs

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



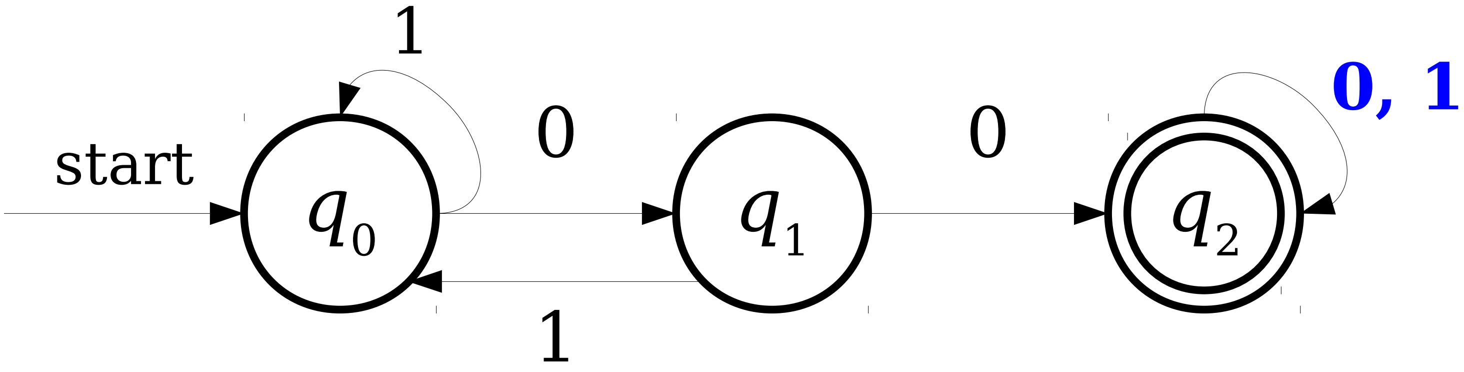
Recognizing Languages with DFAs

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



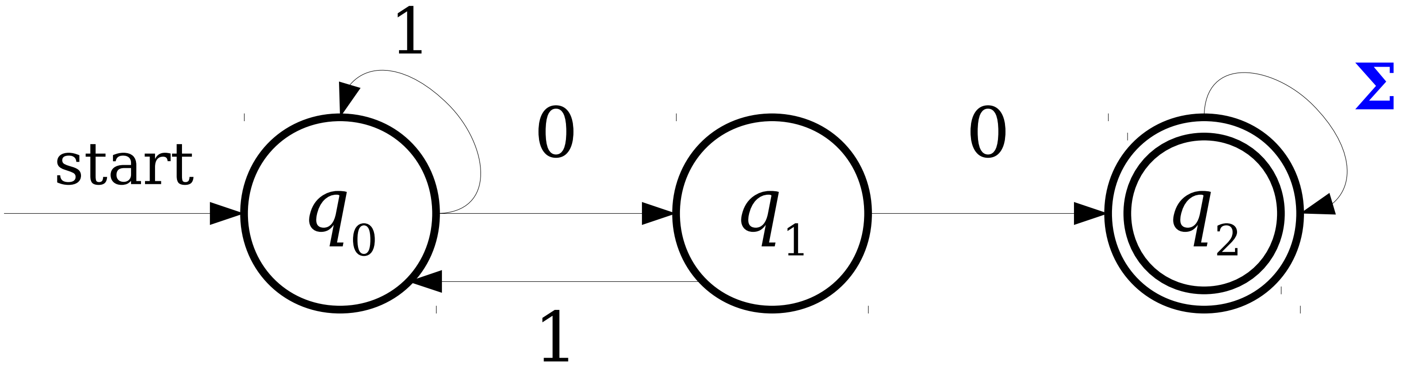
Recognizing Languages with DFAs

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



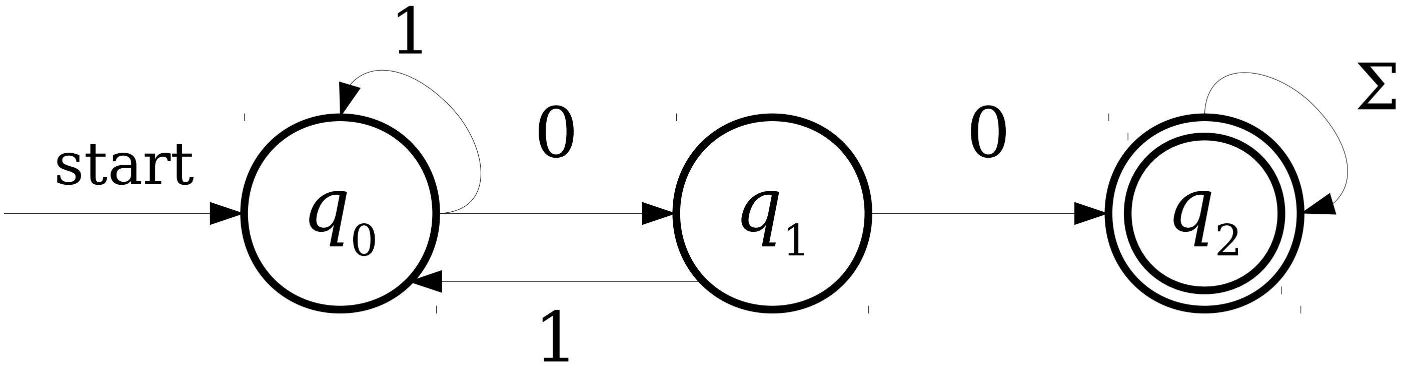
Recognizing Languages with DFAs

L = { w ∈ {0, 1}* | w contains 00 as a substring }



Recognizing Languages with DFAs

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



More Elaborate DFAs

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

Let's have the a symbol be a placeholder for "some character that isn't a star or slash."

Try designing a DFA for comments! Here's some test cases to help you check your work:

Accepted:

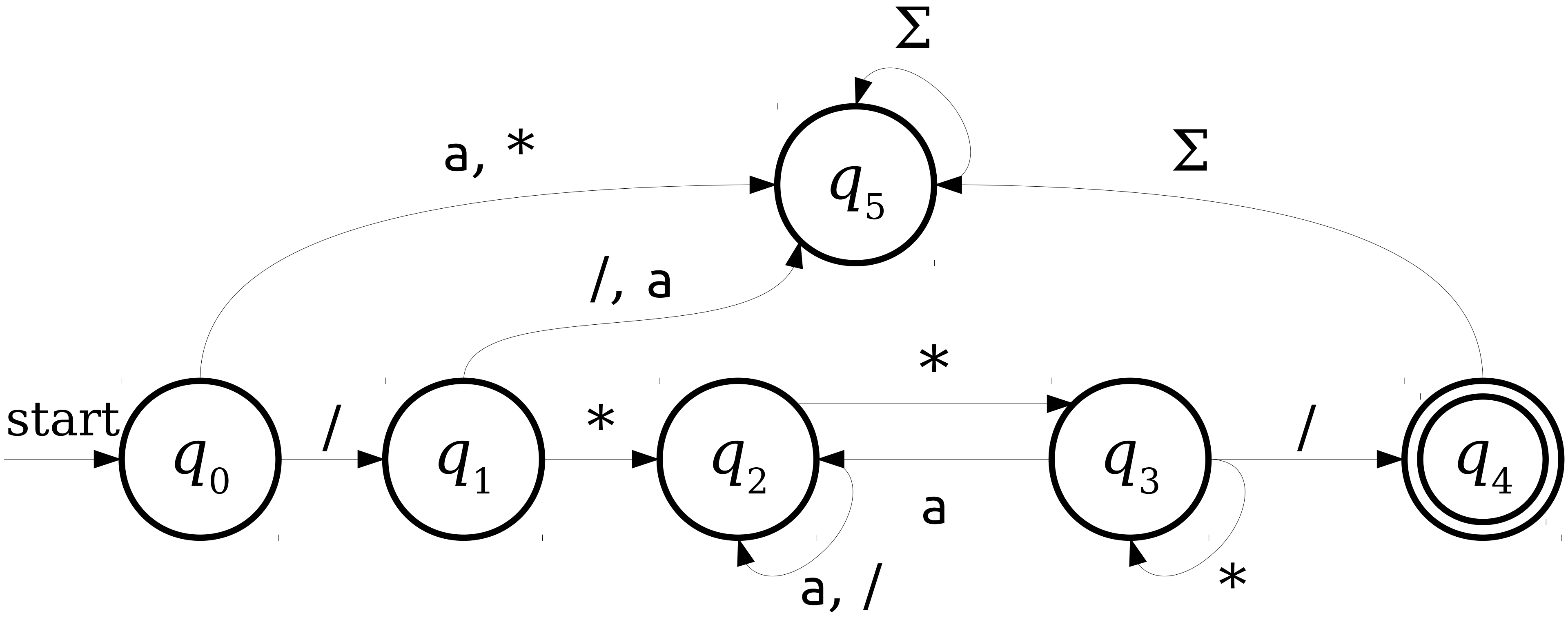
/*a*/
/**/
/***/
/*aaa*aaa*/
/*a/a*/

Rejected:

/**
/**/a/*aa*/
aaa/**/aa
/*/
/**a/
//aaaa

More Elaborate DFAs

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


Σ
a, *
Σ
q_5
/, a
*
start
q_0
/
q_1
*
q_2
q_3
/
q_4
a
a, /
*