Finite Automata


BRACE YOURSELF
MIDTERM IS COMING
quickmeme.com

Midterm Logistics

- Midterm is next **Tuesday, February 12** from **7PM – 10PM** (location TBA).
  - Open-book, open-note, open-computer, closed-network.
  - Covers material up through and including Wednesday's lecture.
- Practice exam available now; solutions will be released on Wednesday.
- If you need to take the exam at an alternate time, email the course staff no later than Wednesday at 12:50PM.

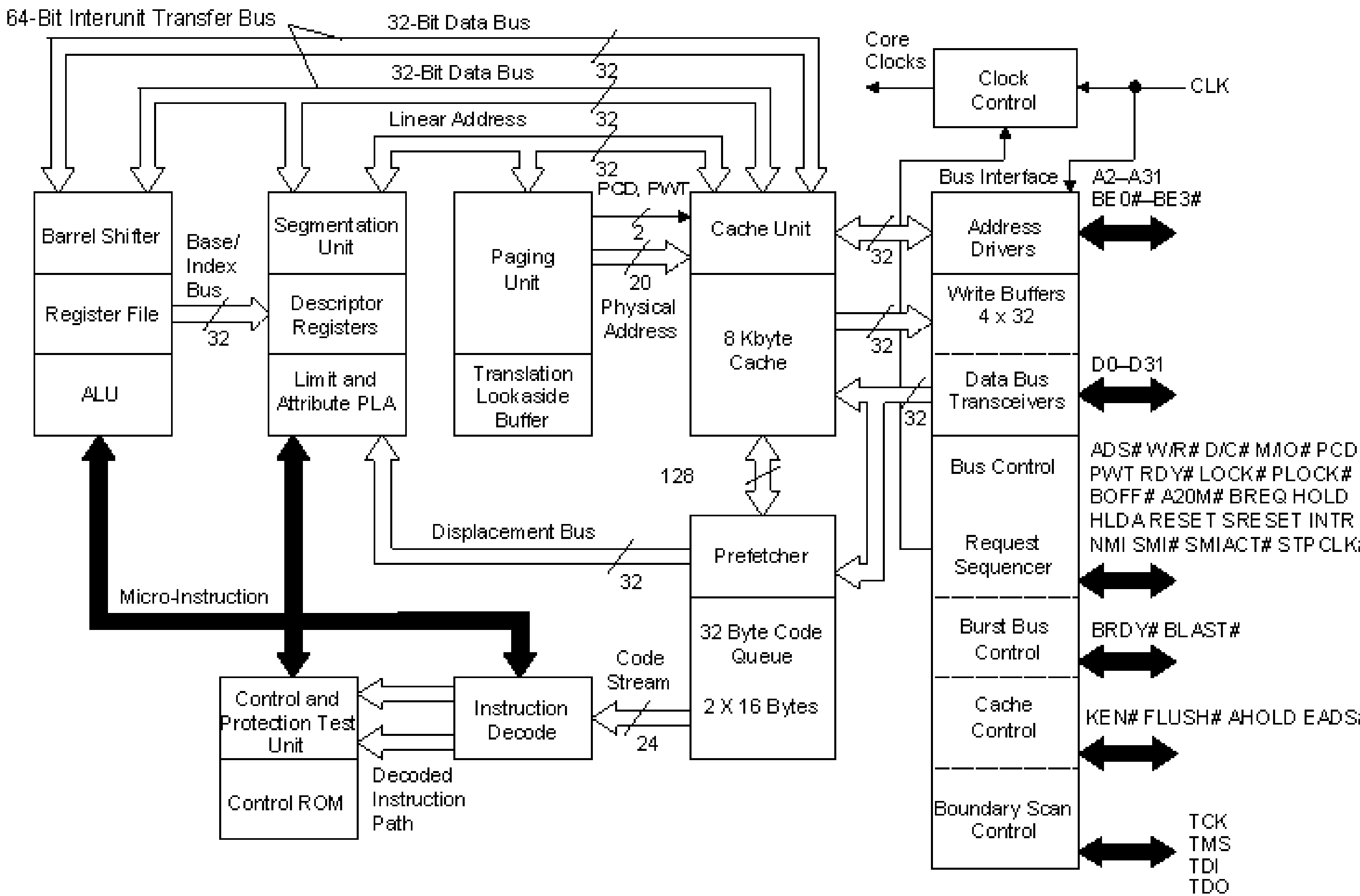
Computability Theory

What problems can we solve with a computer?

What problems can we solve with a computer?

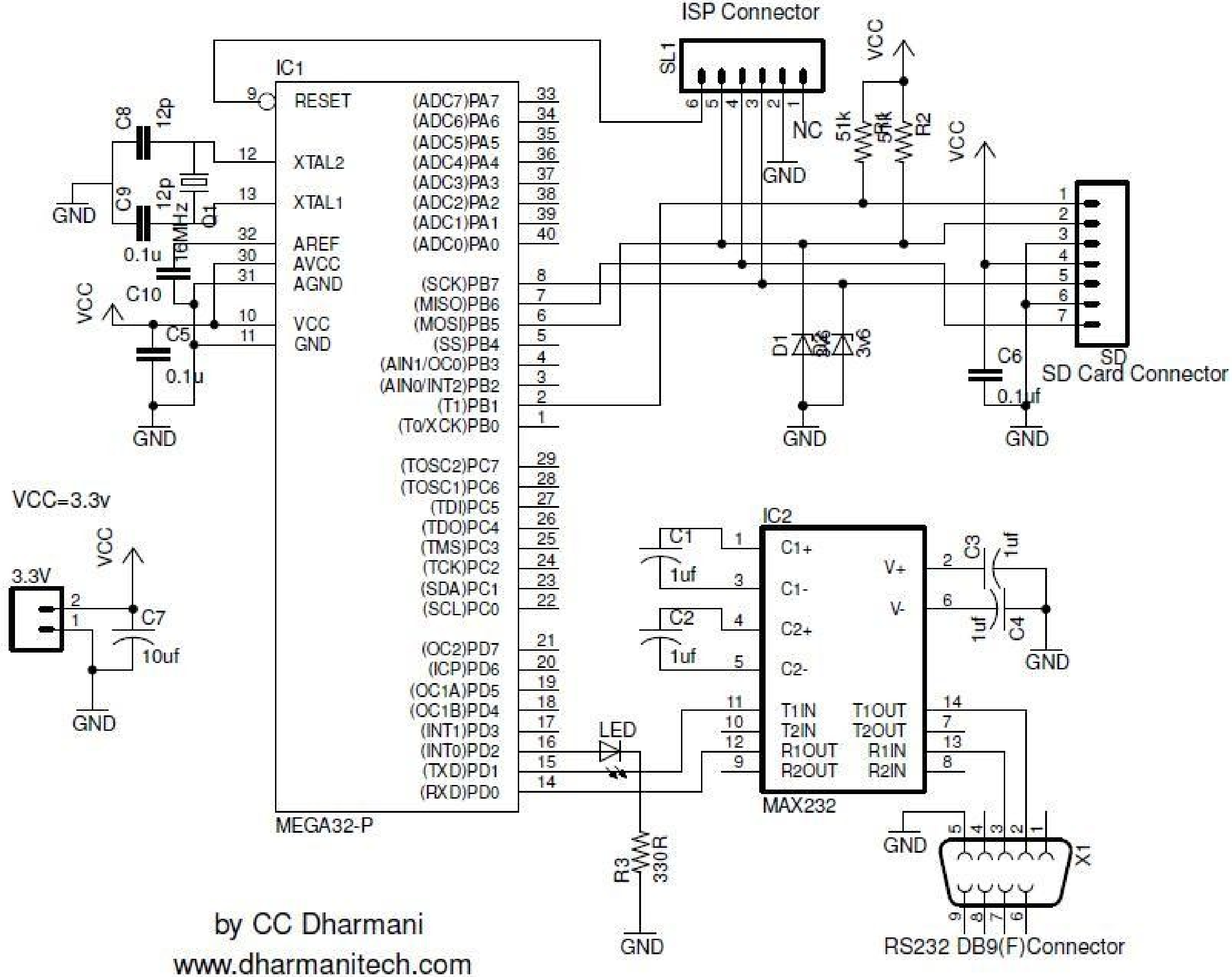
What kind of
computer?

Computers are Messy



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

Computers are Messy



microSD/SD Card interface with ATmega32 Ver_2.3

http://www.dharmanitech.com/

Computers are Messy

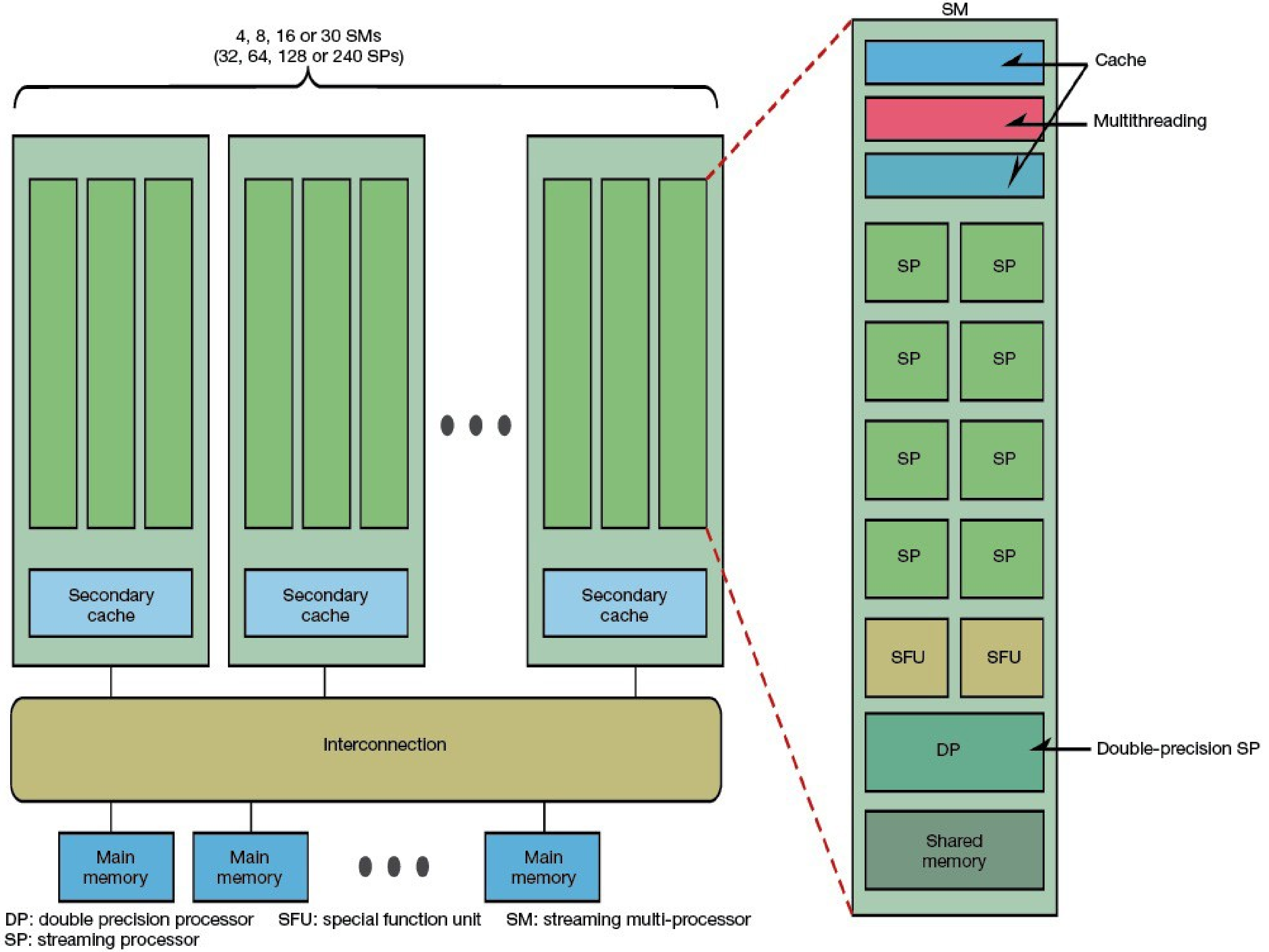
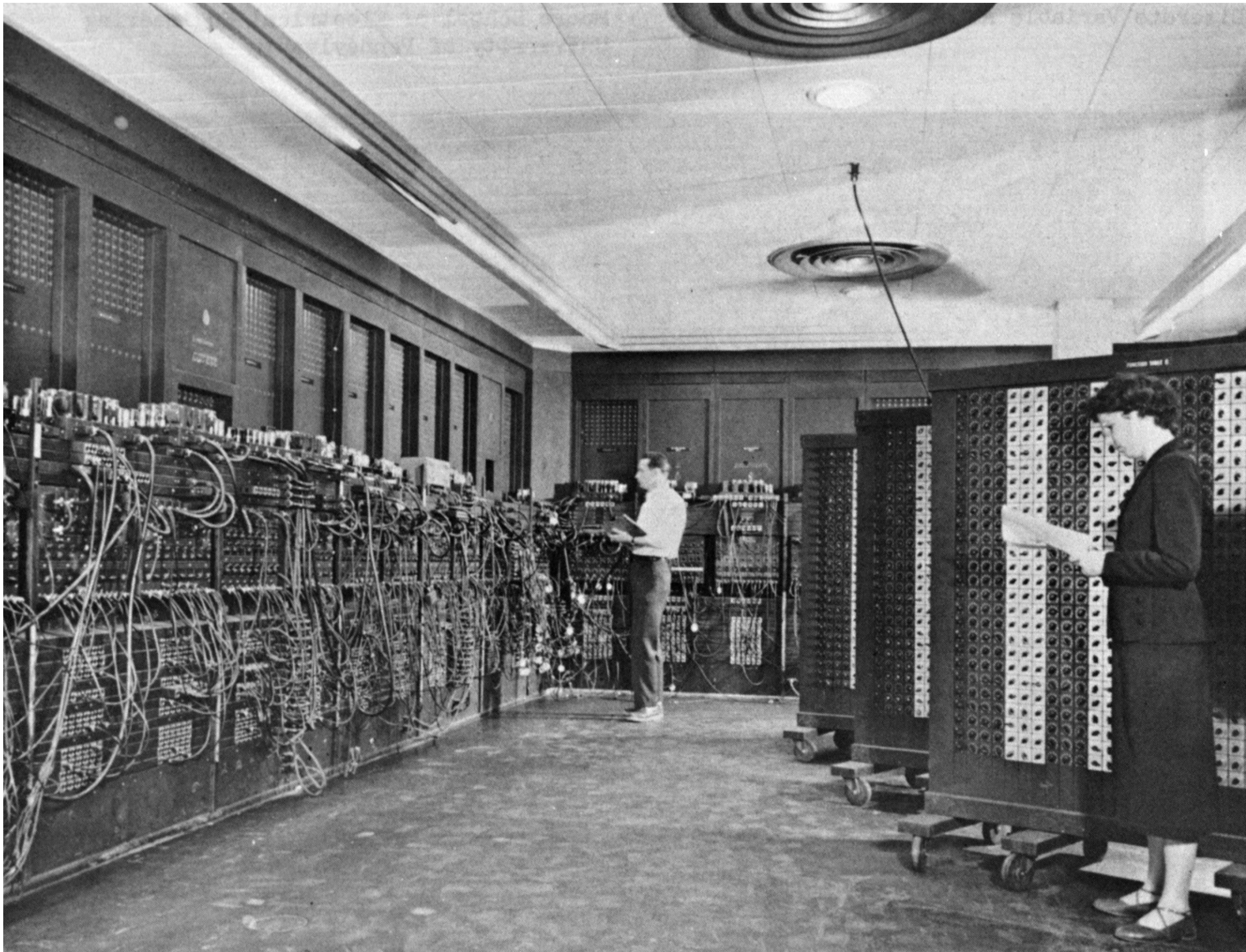


Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM)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

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

Computers are Messy



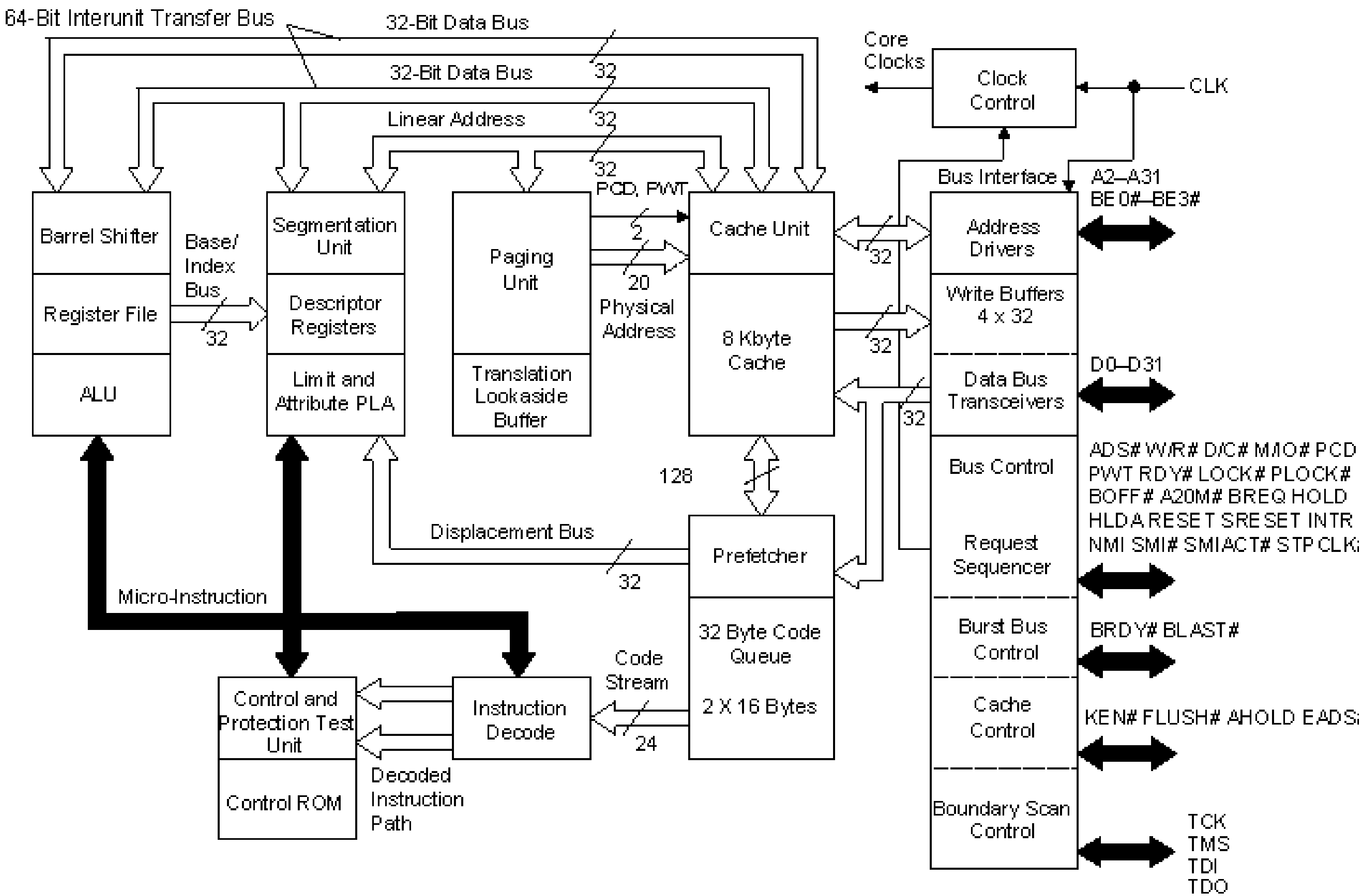
http://en.wikipedia.org/wiki/File:Eniac.jpg

We need a simpler way of
discussing computing machines.

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

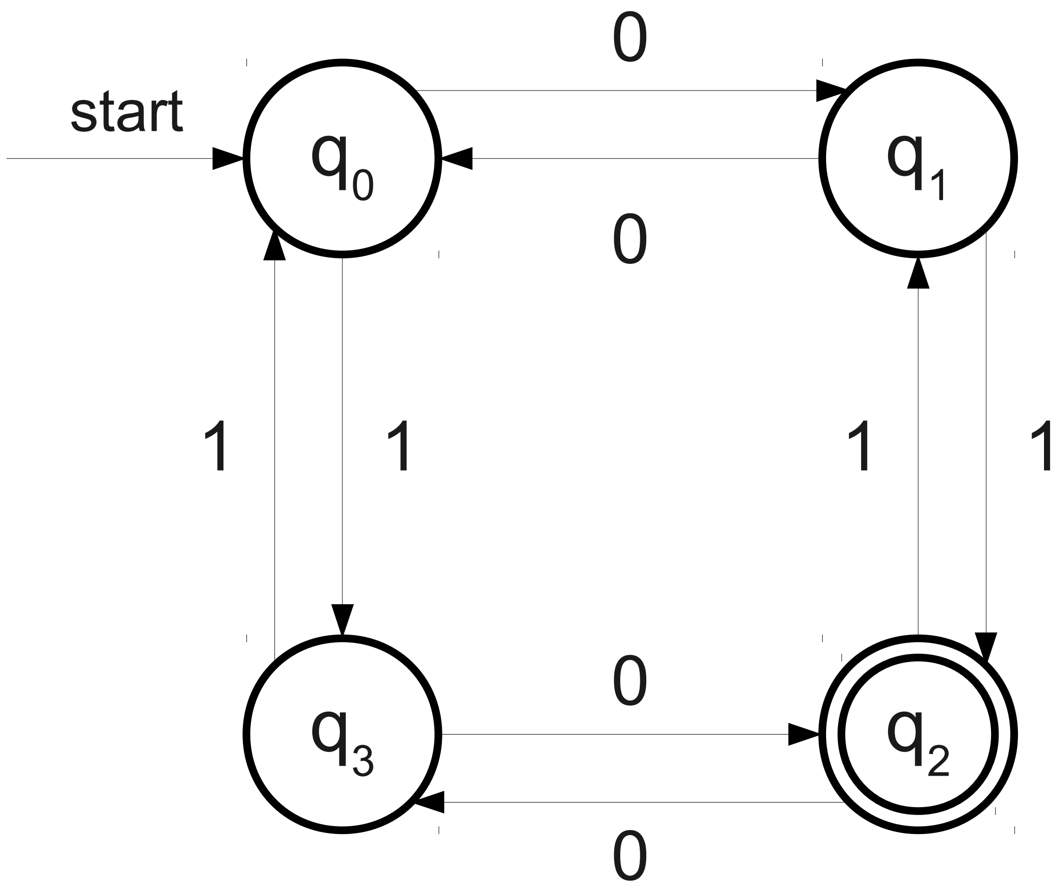
Automata make it possible to reason about computability by **abstracting away** the implementation complexity of real computing systems.

Computers are Messy

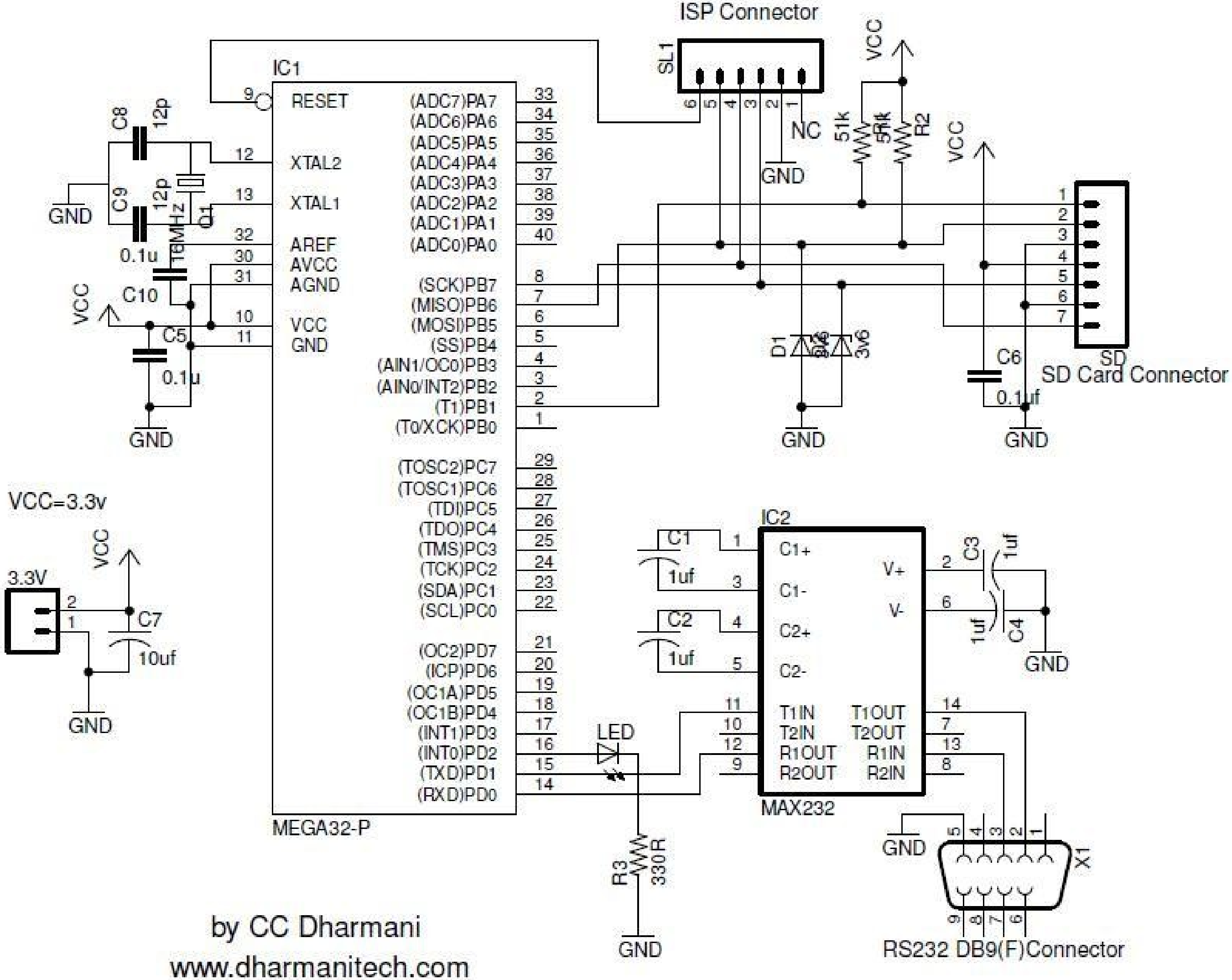


http://www.intel.com/design/intarch/prodbref/272713.htm

Automata are Clean



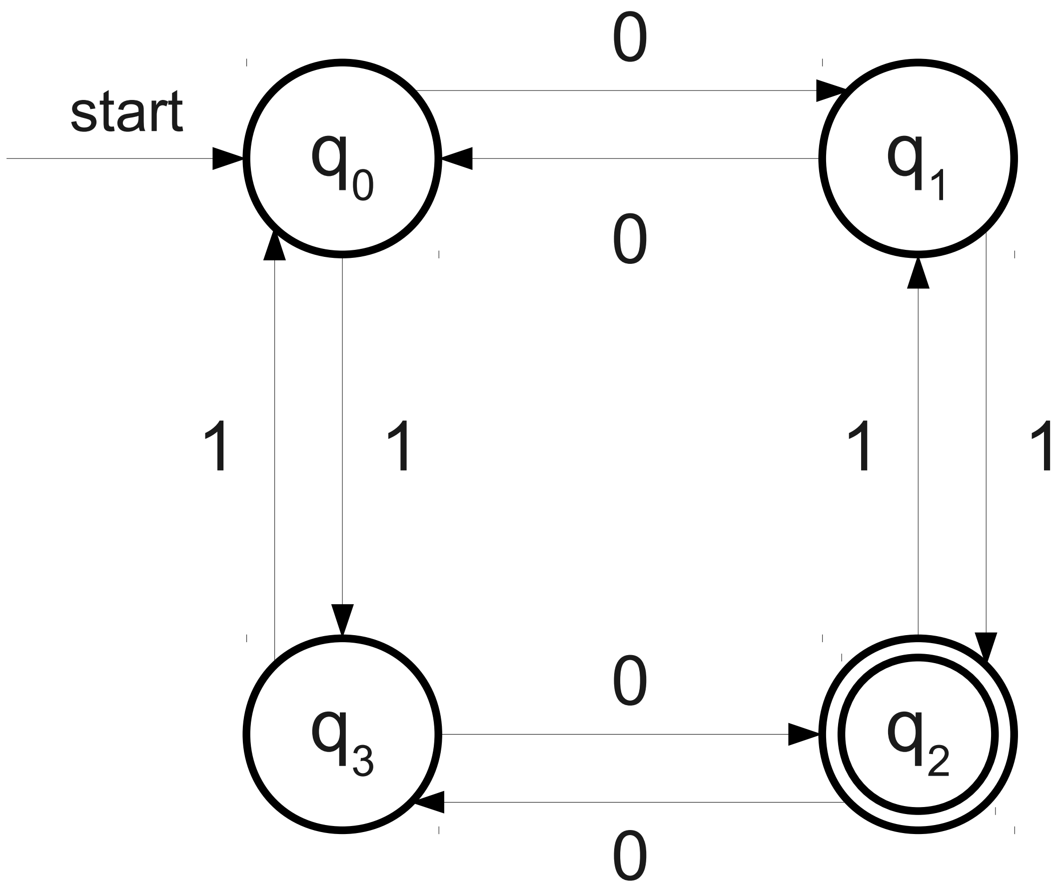
Computers are Messy



microSD/SD Card interface with ATmega32 Ver_2.3

http://www.dharmanitech.com/

Automata are Clean



Computers are Messy

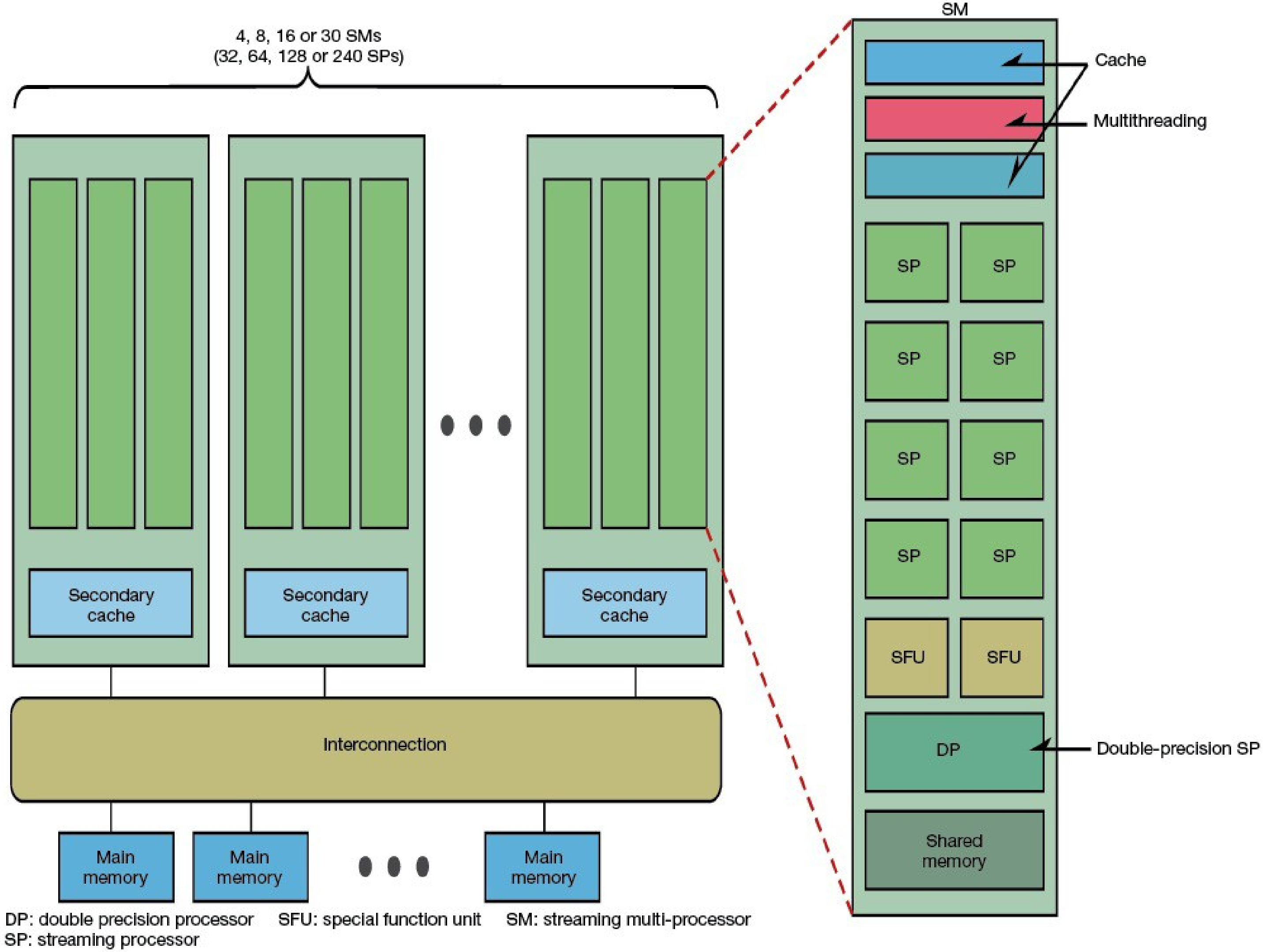
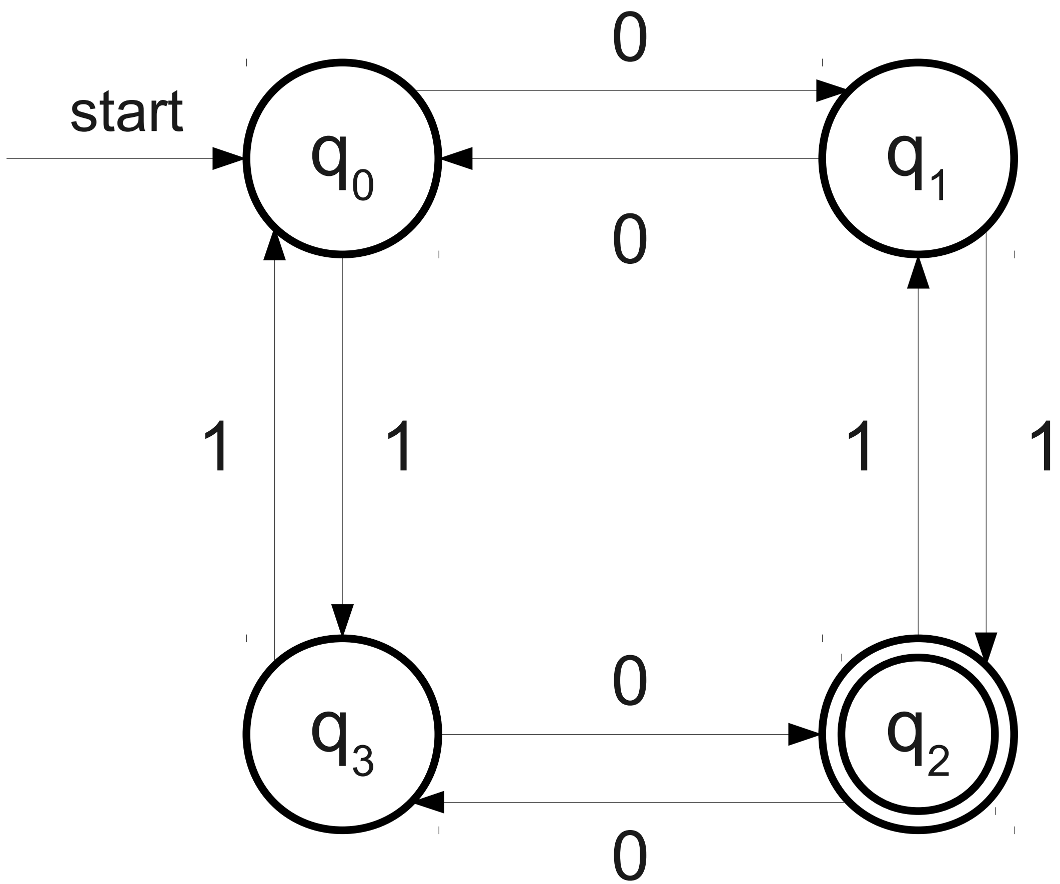


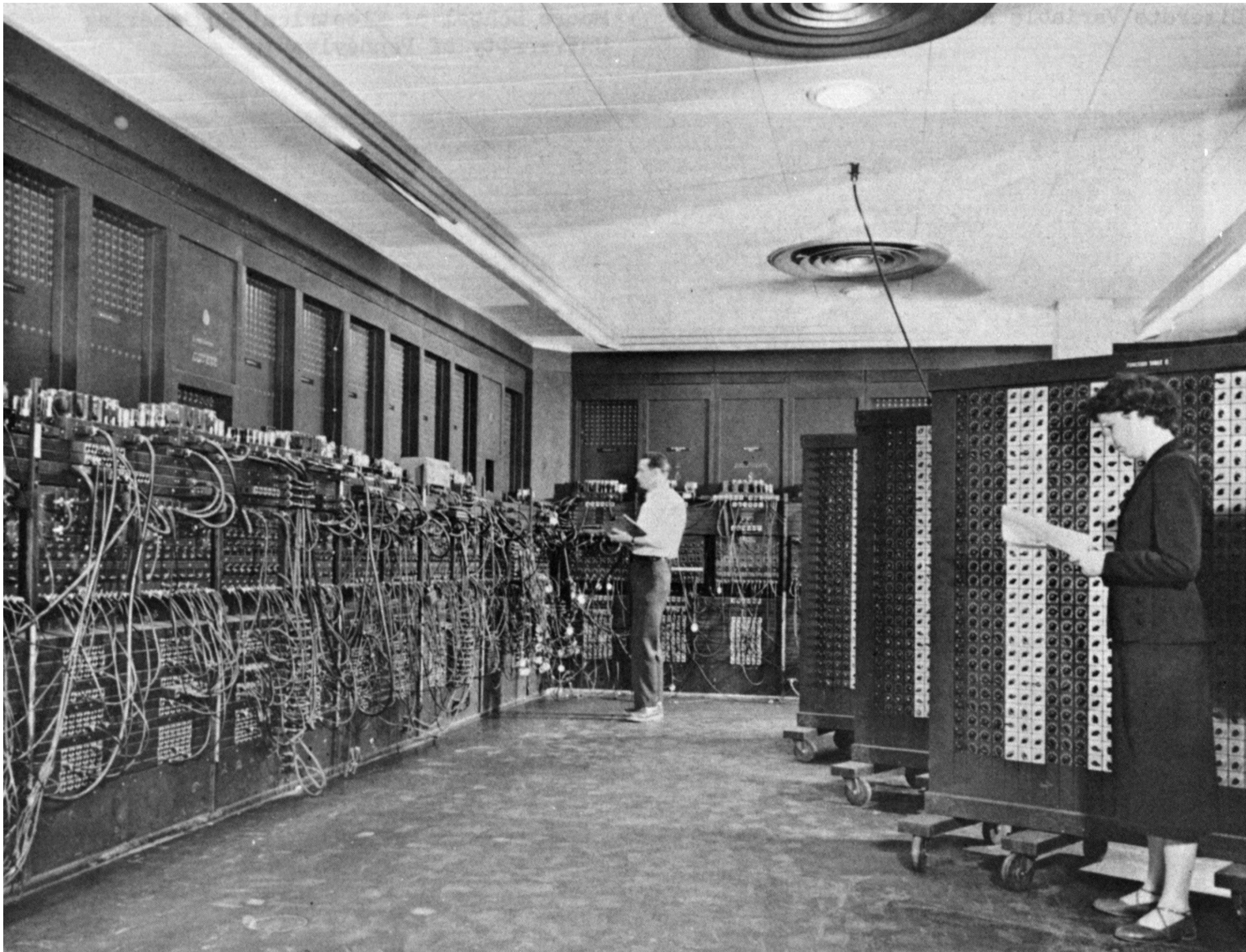
Fig 2 Covering Everything from PCs to Supercomputers NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM)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

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

Automata are Clean

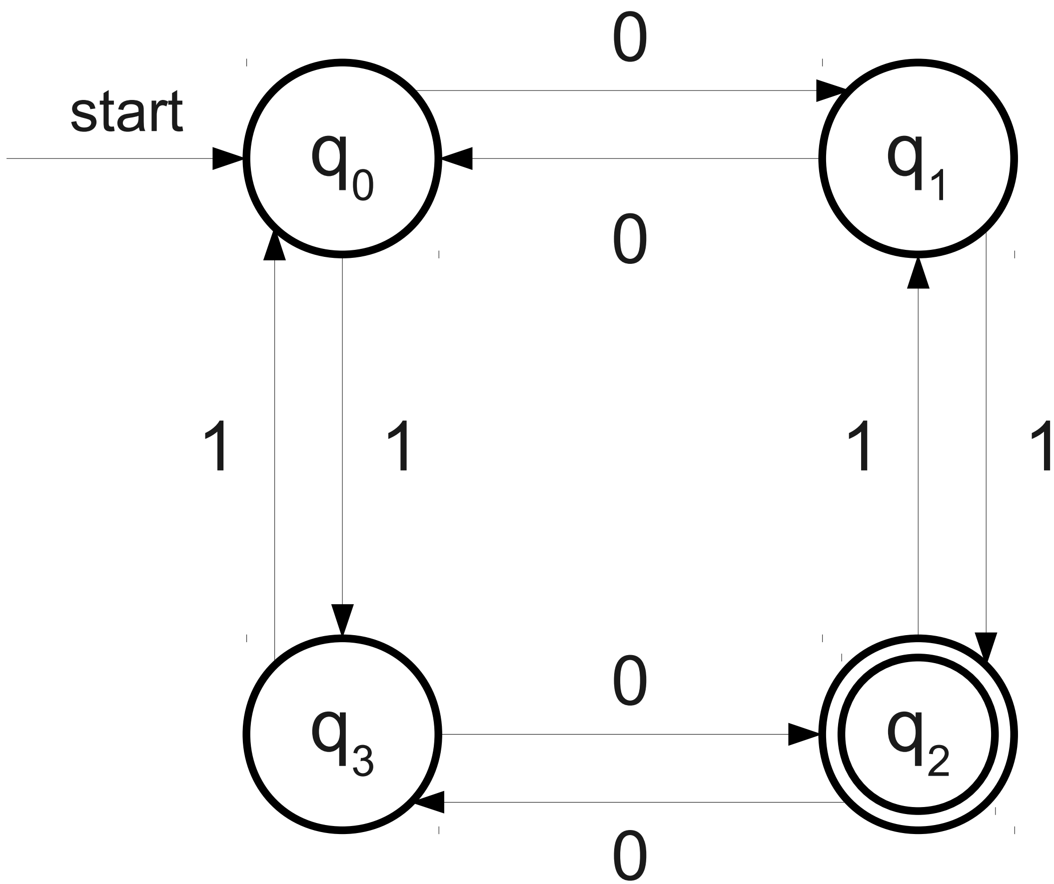


Computers are Messy



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

Automata are Clean



Why Build Models?

- The models of computation we will explore in this class correspond to different conceptions of what a computer could do.
- **Finite automata** (this week) are an abstraction of computers with finite resource constraints.
  - Provide upper bounds for the computing machines that we can actually build.
- **Pushdown automata** and **Turing machines** (after this week) 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.

Decision Problems

- In this class, we will consider **decision problems**, problems with yes/no answers.
- Examples:
  - Does 137 + 42 have 3 as a divisor?
  - Is P the shortest path from u to v?
  - SAT.

Decision and Function Problems

- Decision problems do not encompass all possible problems.
  - Example: “What is 2 + 2?” is not a decision problem.
- These more general problems are called **function problems**.
- For now, we'll ignore function problems. We'll revisit them toward the end of the quarter.

Why Decision Problems

- Why restrict ourselves to decision problems?
- Many nice mathematical properties:
  - All answers are just one bit, so machines can produce answers more easily.
  - No need to worry about what formats the answers will be provided in.
  - Easy to use as subroutines.
- If we can't solve a decision problem, the question must be so hard that we can't even get a one-bit answer back!

How do we encode problems?

Strings

- An **alphabet** is a finite set of **characters**.
  - Typically, we use the symbol **Σ** to refer to an alphabet.
- A **string** is a finite sequence of characters drawn from some alphabet.
- Example: If $\Sigma = \{0, 1\}$, some valid strings include
  - 0
  - 11101001000010000001
  - 11011100101110111
- 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 formed from characters in Σ.
- Example: The language of palindromes over Σ = {0, 1, 2} is the set

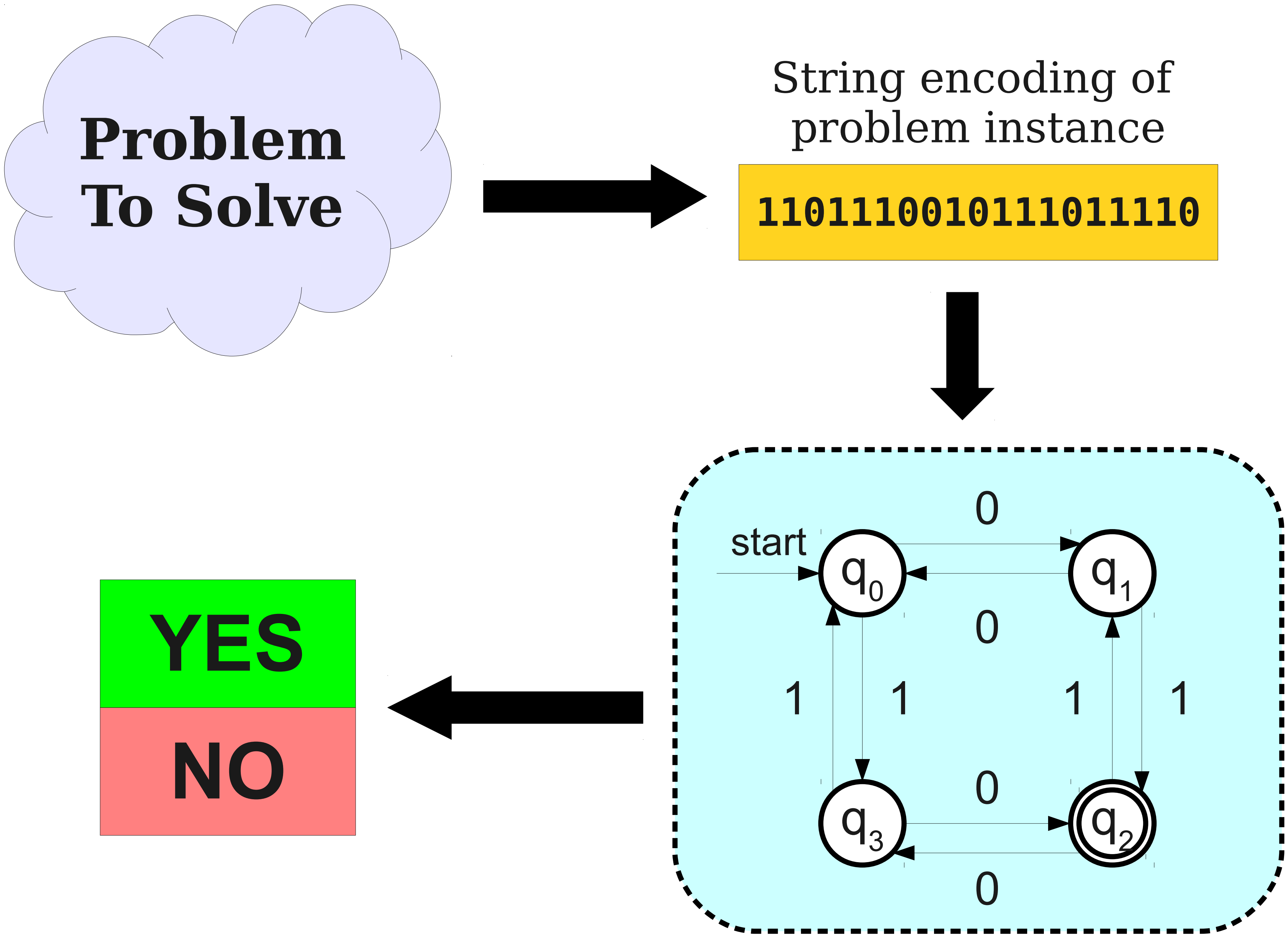
  {ε, 0, 1, 2, 00, 11, 22, 000, 010, 020, 101, … }
- The set of all strings composed from letters in Σ is denoted **Σ***.
- L is a language over Σ iff $L \subseteq \Sigma^*$.

Decision Problems and Languages

- Languages give a compact and flexible way to encode decision problems:

  Any decision problem can be represented by a language of strings encoding inputs to which the answer is "yes."

- All the automata we will discuss in this class will be machines for answering the question "is string x in language L?"


Problem To Solve
String encoding of problem instance
11011100101110111110
start
q0
q1
q2
q3
0
0
0
0
1
1
1
1
YES
NO

To Summarize

- An **automaton** is an idealized mathematical computing machine.
- A **language** is a set of strings.
- A **decision problem** is a yes/no question (though it can be quite complex).
- The automata we will study will accept as input a string and (attempt to) output whether that string is contained in a particular language.

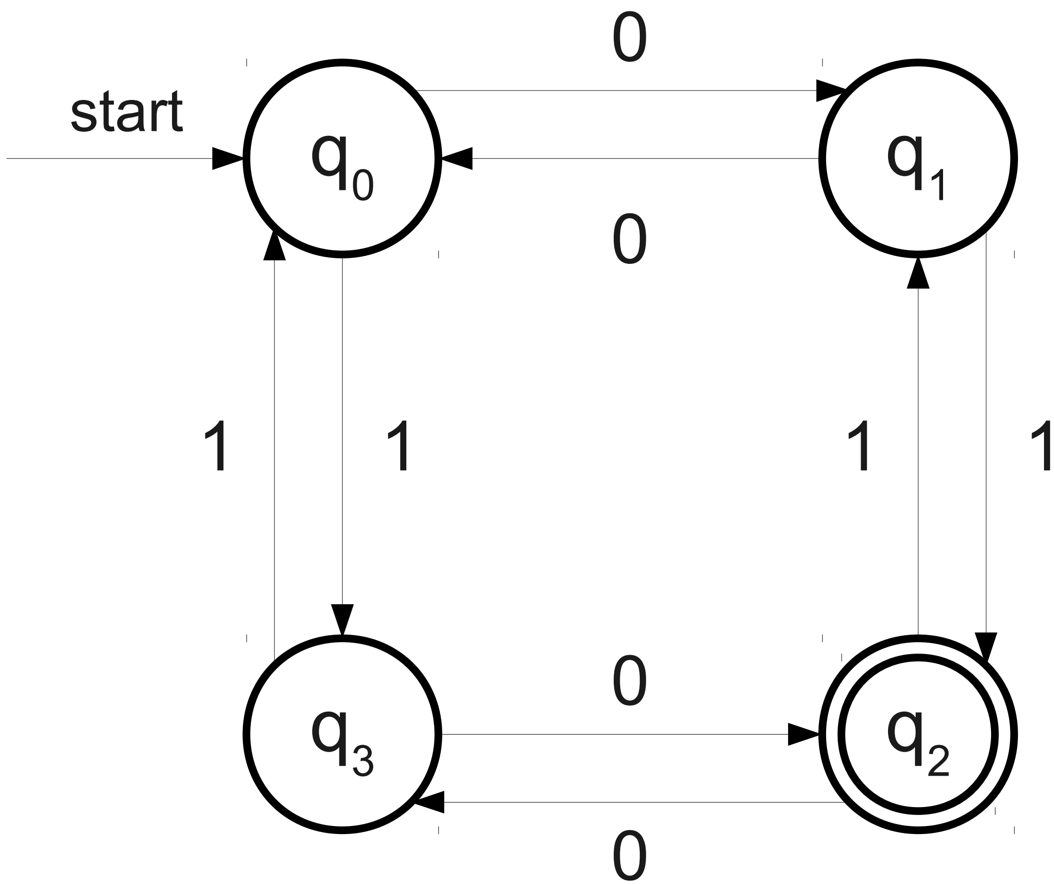
What problems can we solve with a computer?

Finite Automata

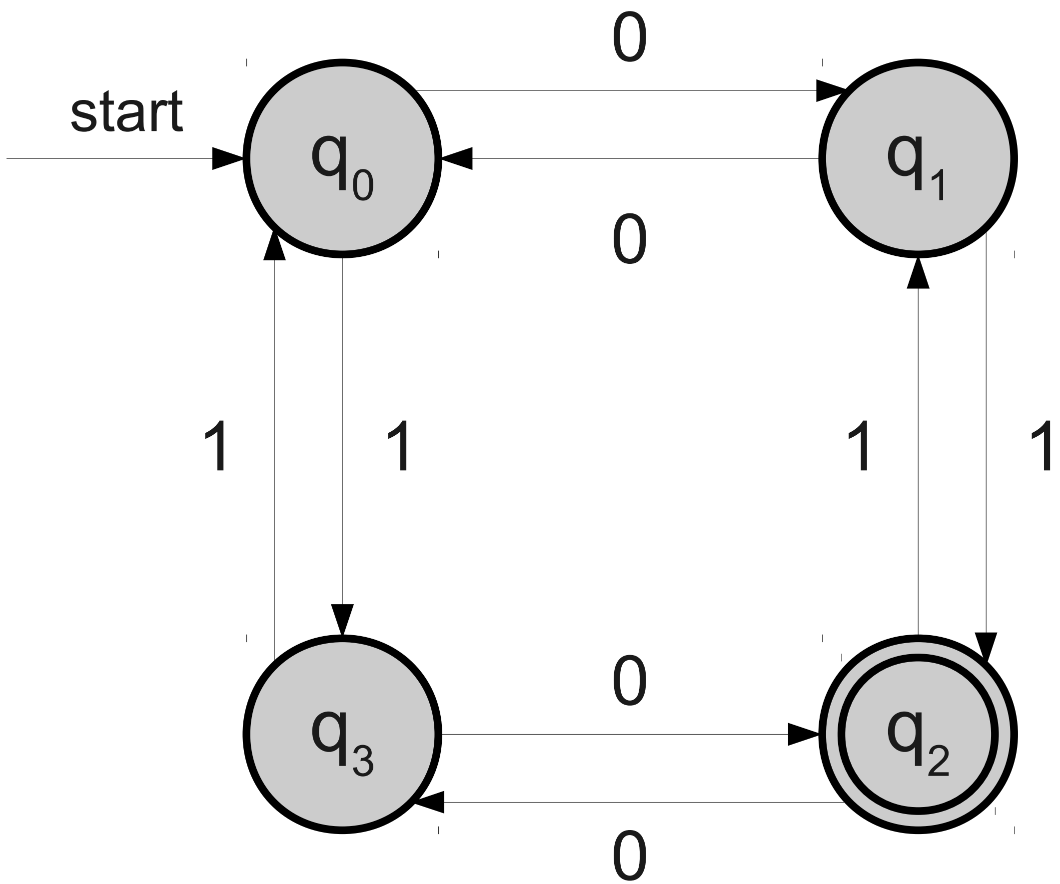
A **finite automaton** is a mathematical machine for determining whether a string is contained within some language.

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

A Simple Finite Automaton

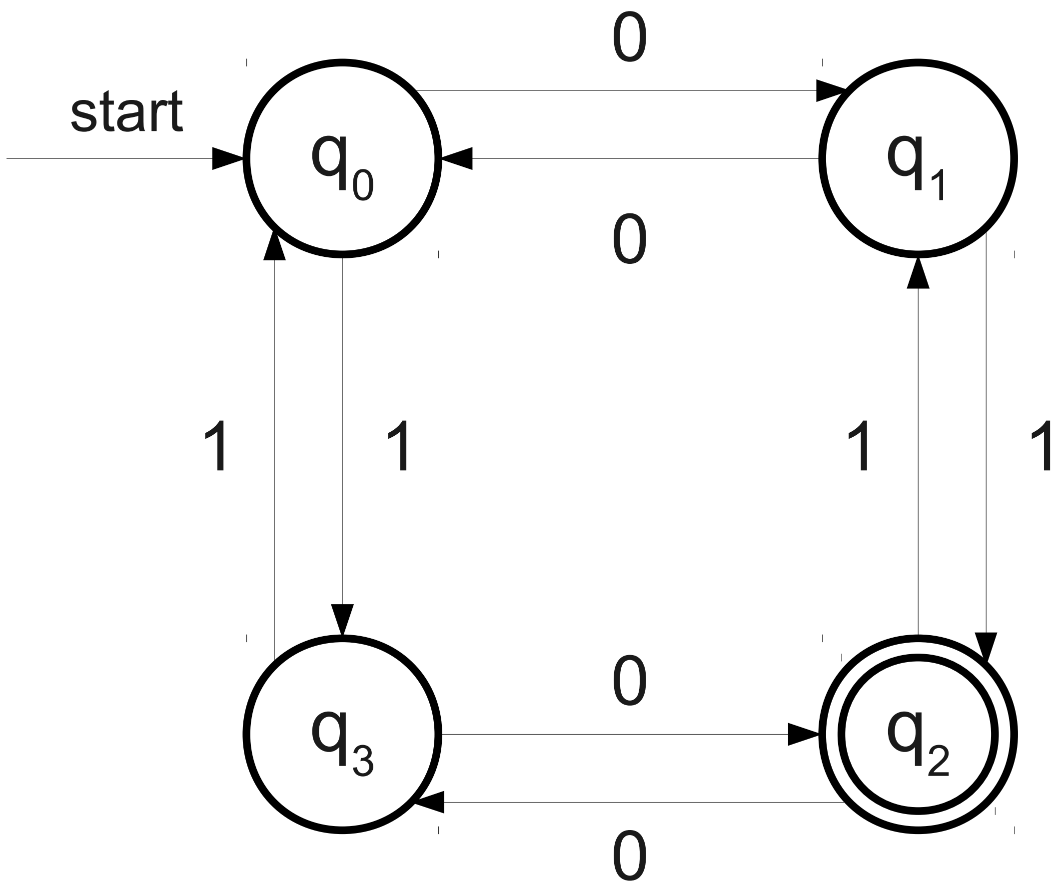


A Simple Finite Automaton

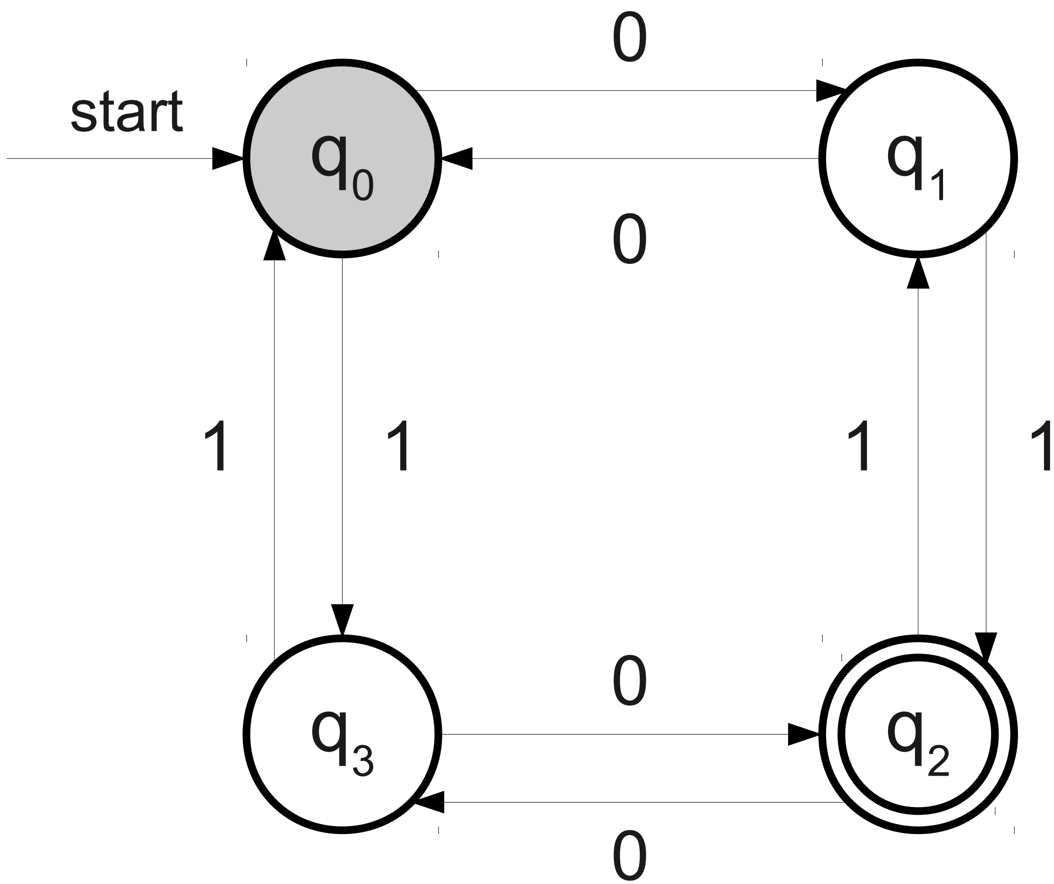


Each circle represents a **state** of the automaton.

A Simple Finite Automaton

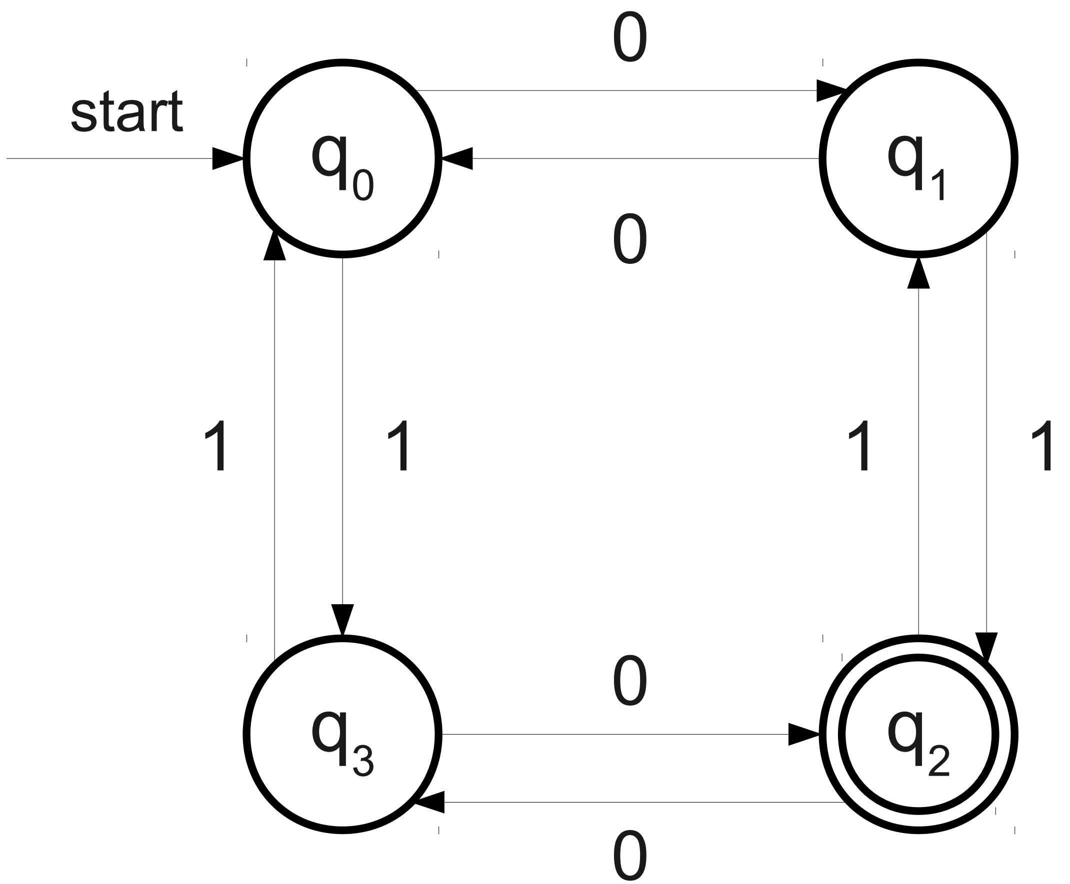


A Simple Finite Automaton

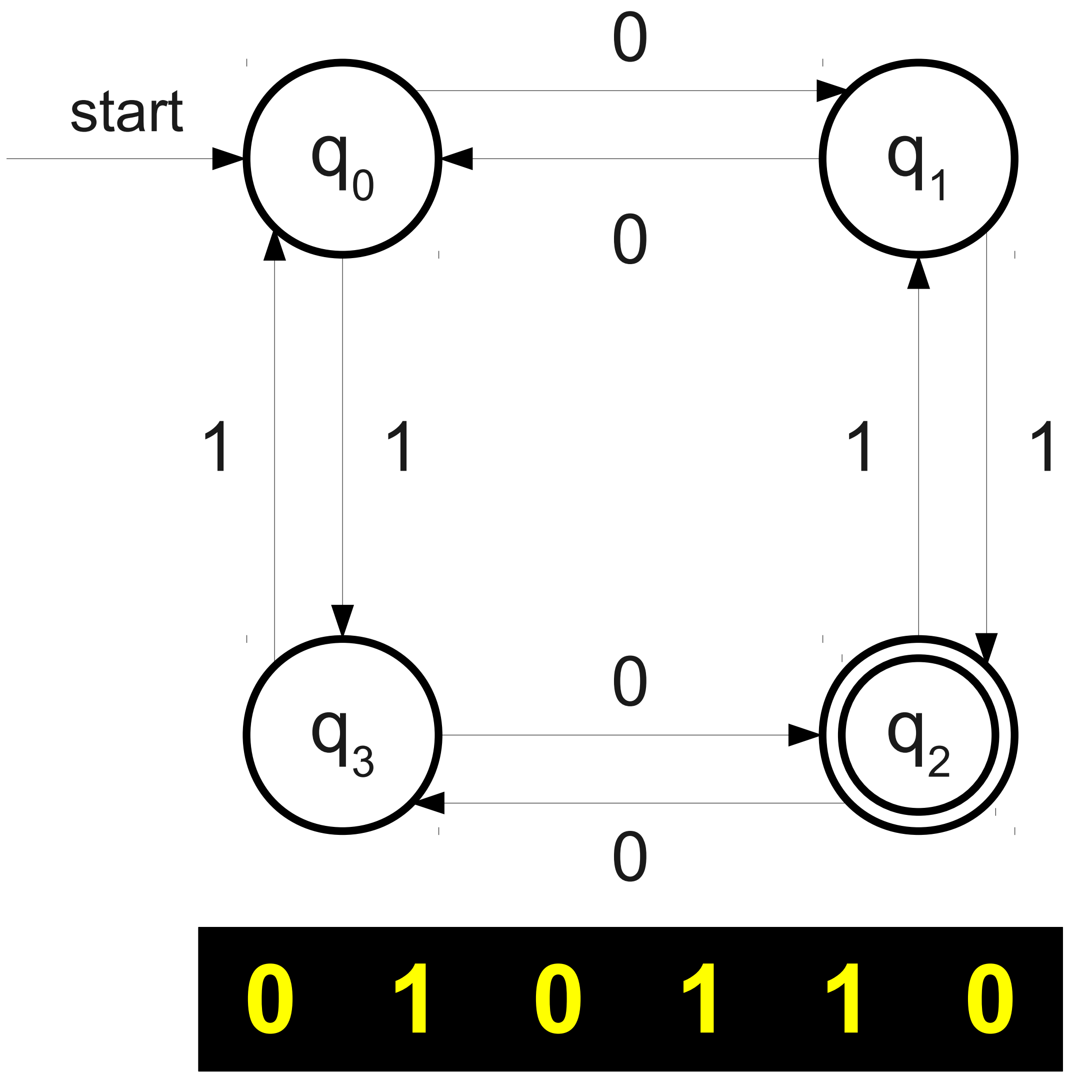


One special state is designated as the **start state**.

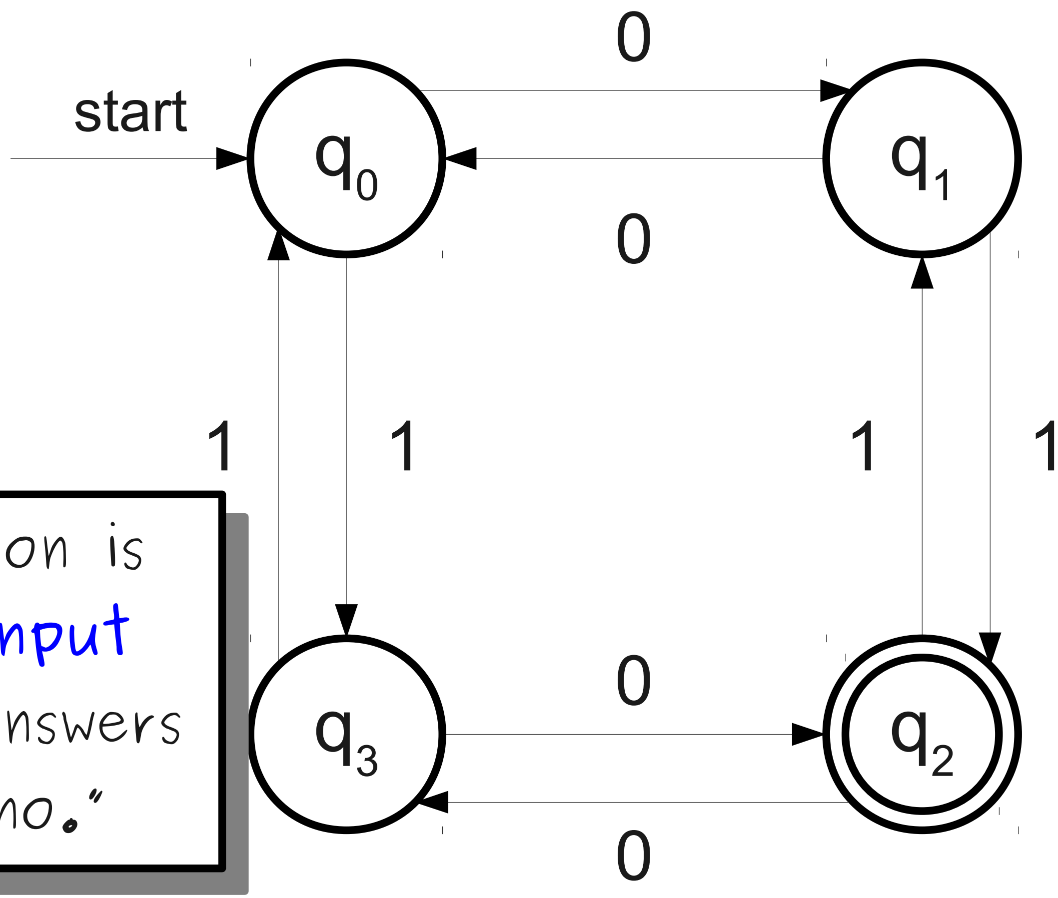
A Simple Finite Automaton



A Simple Finite Automaton



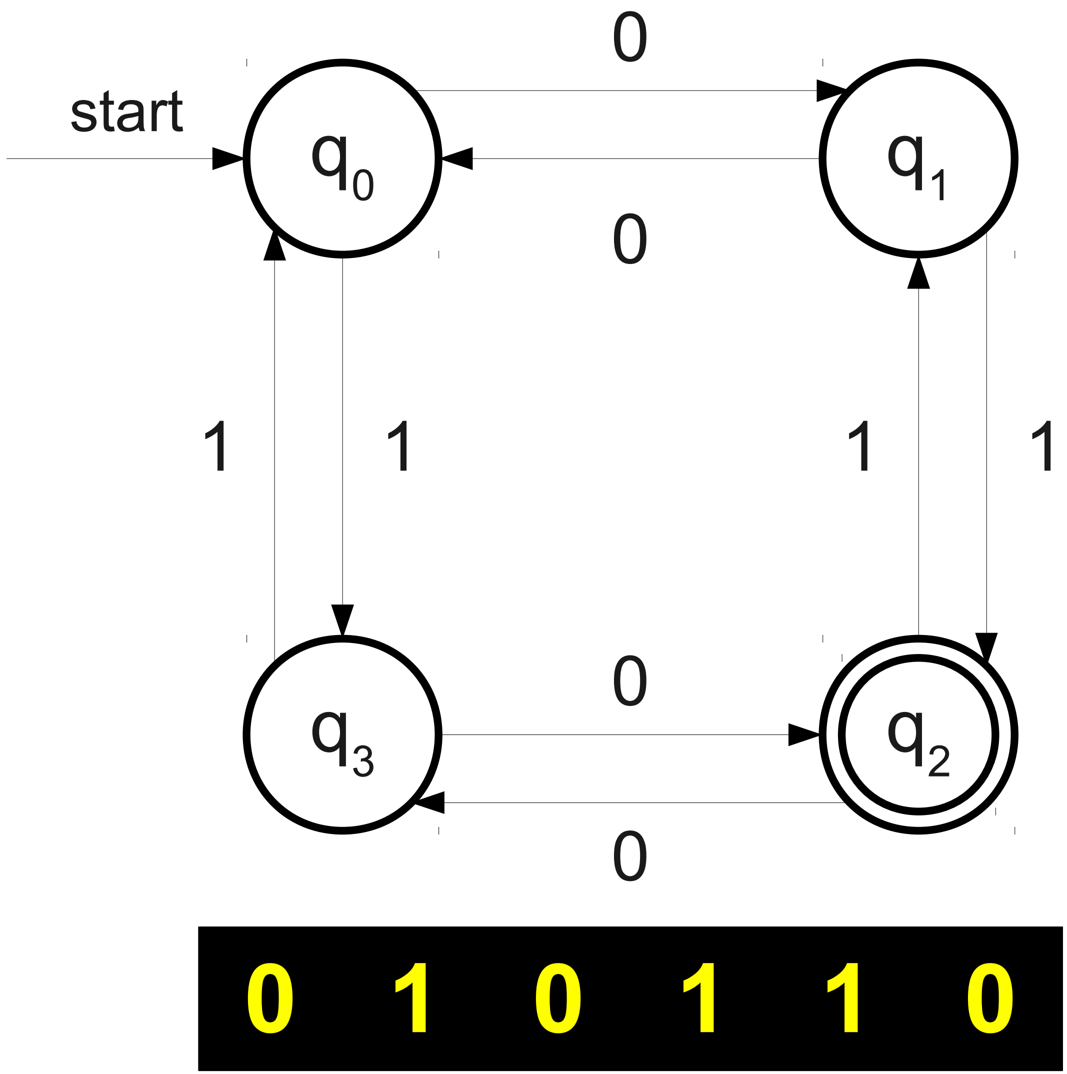
A Simple Finite Automaton



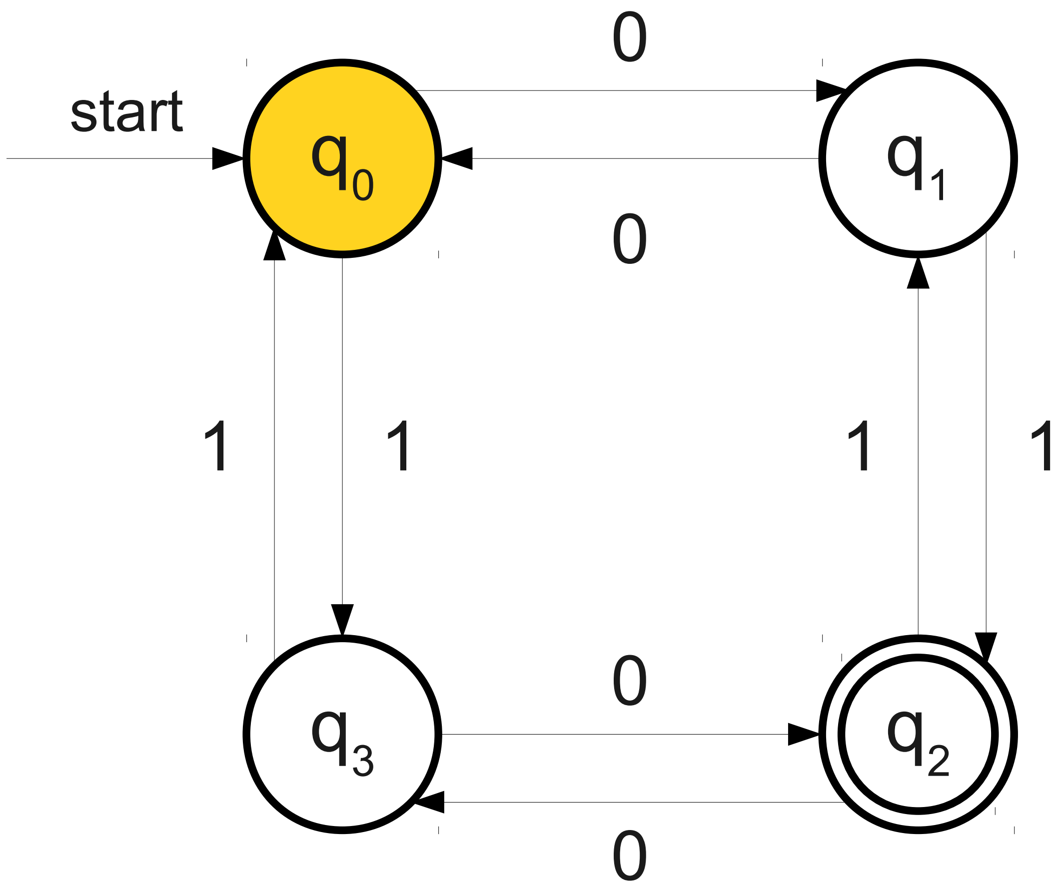
The automaton is run on an **input string** and answers "yes" or "no."

0	1	0	1	1	0

A Simple Finite Automaton

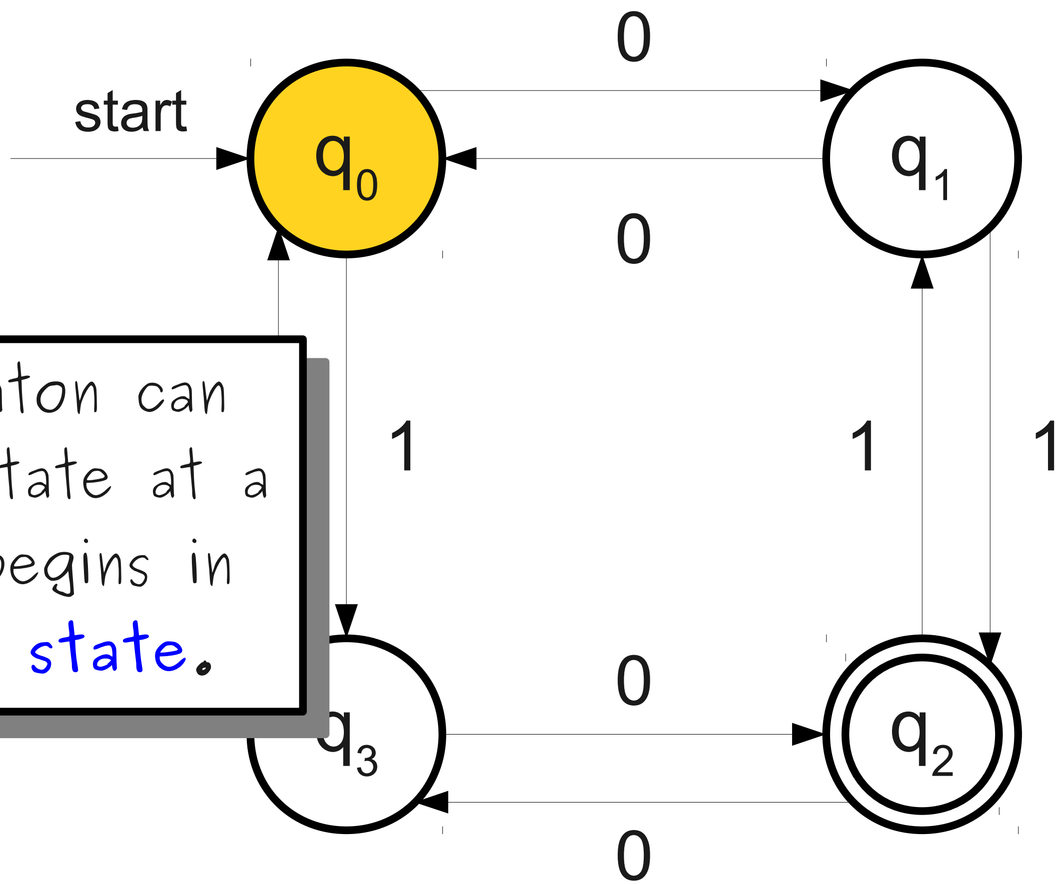


A Simple Finite Automaton



0 1 0 1 1 0

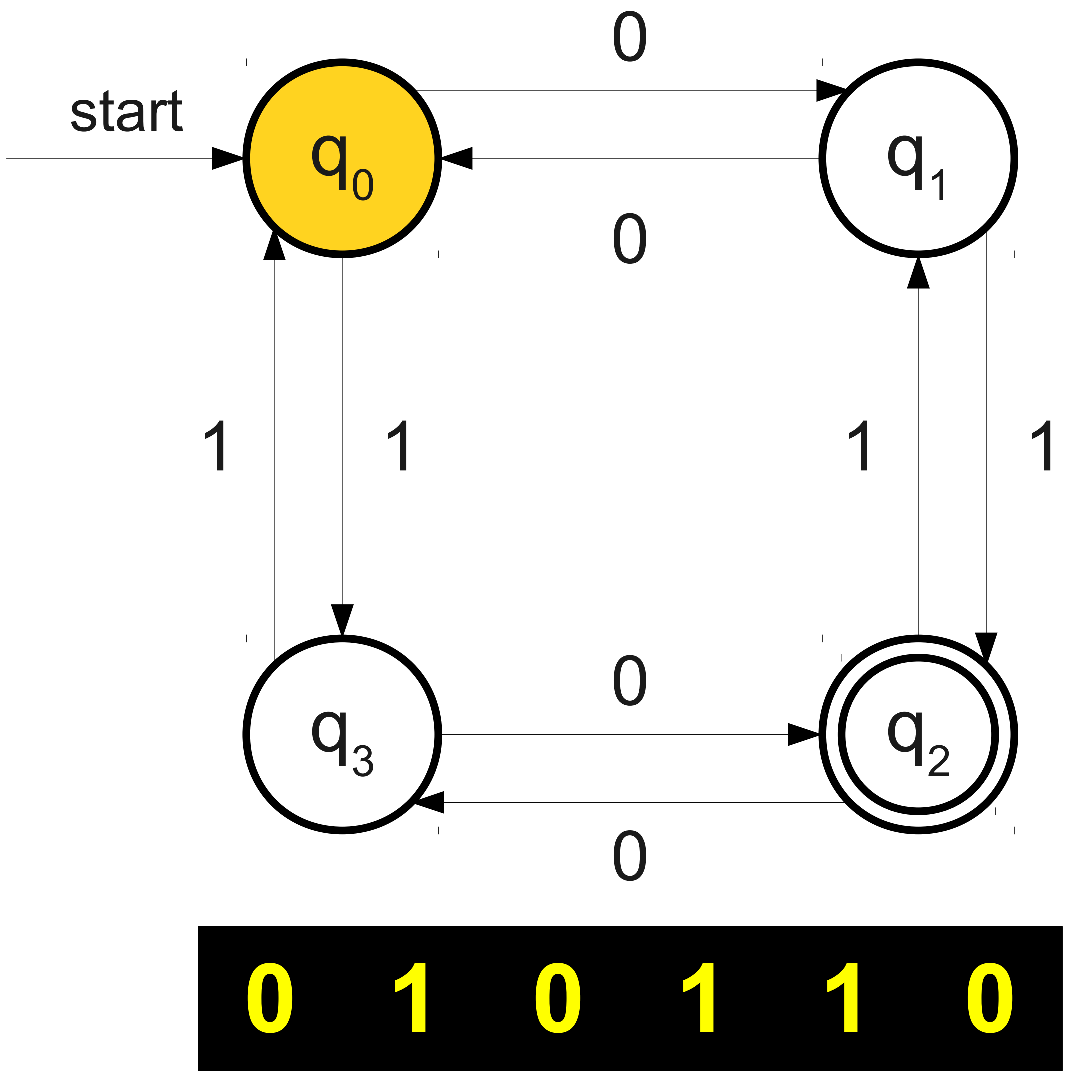
A Simple Finite Automaton



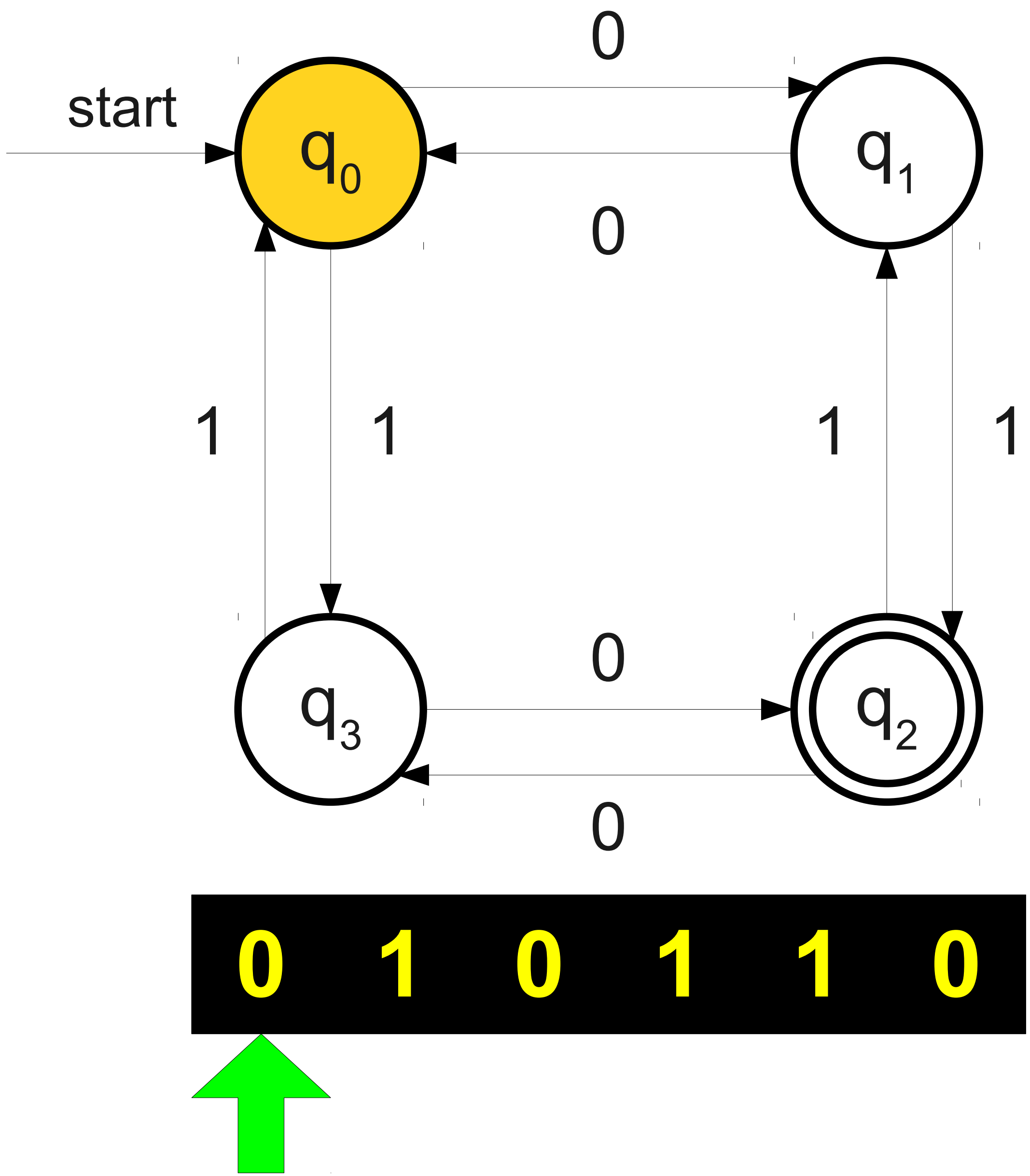
The automaton can be in one state at a time. It begins in the start state.

0 1 0 1 1 0

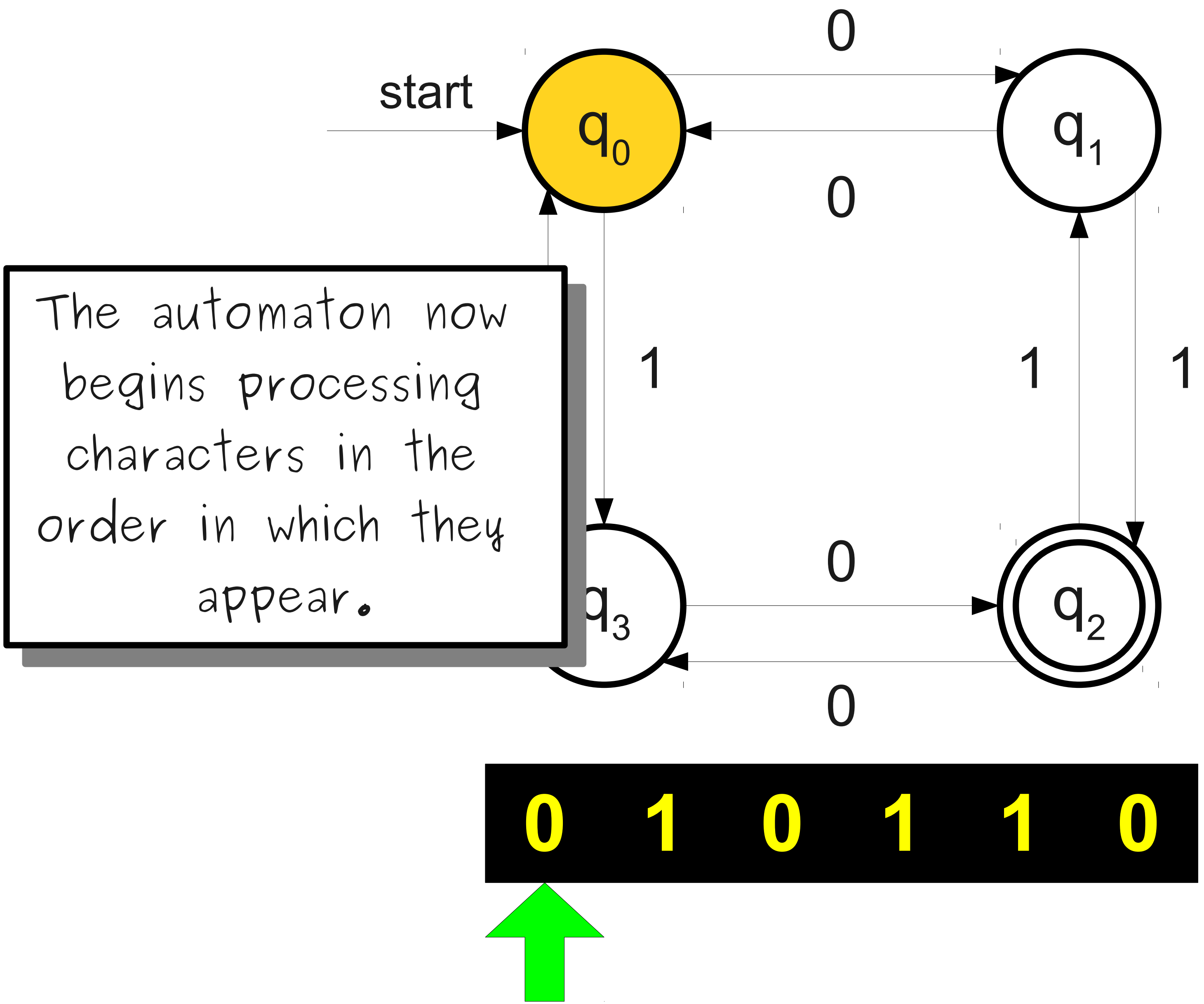
A Simple Finite Automaton



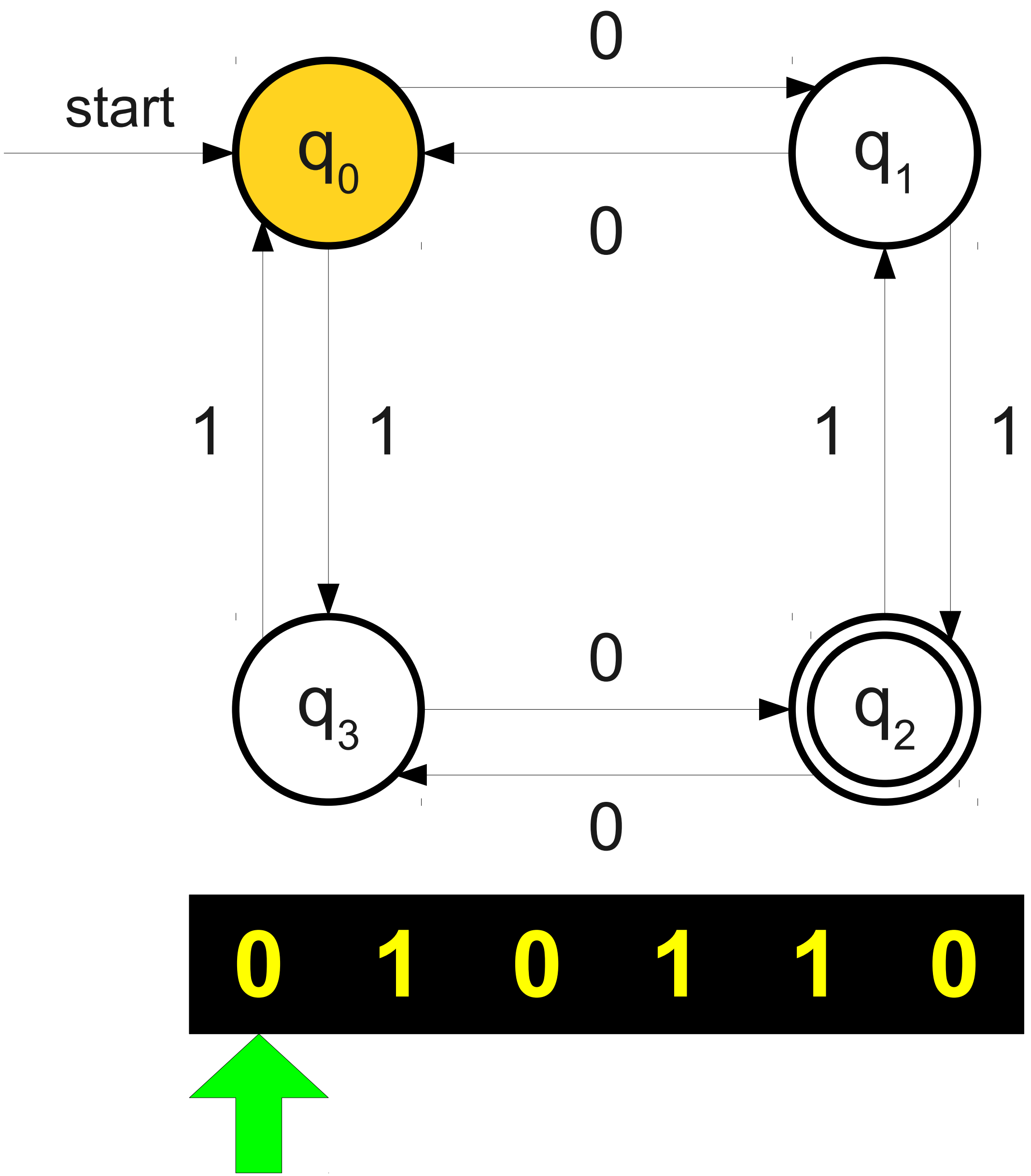
A Simple Finite Automaton



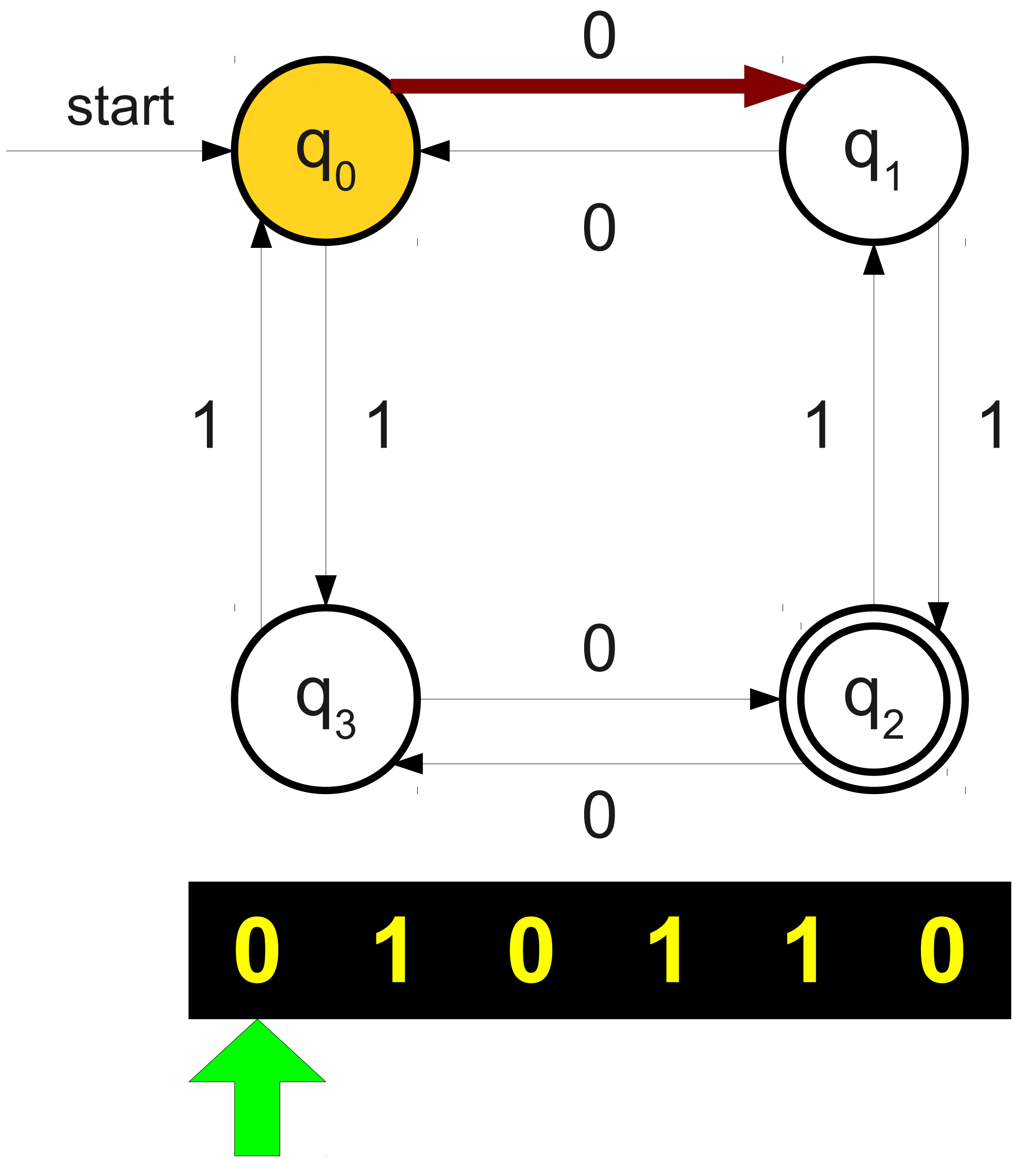
A Simple Finite Automaton



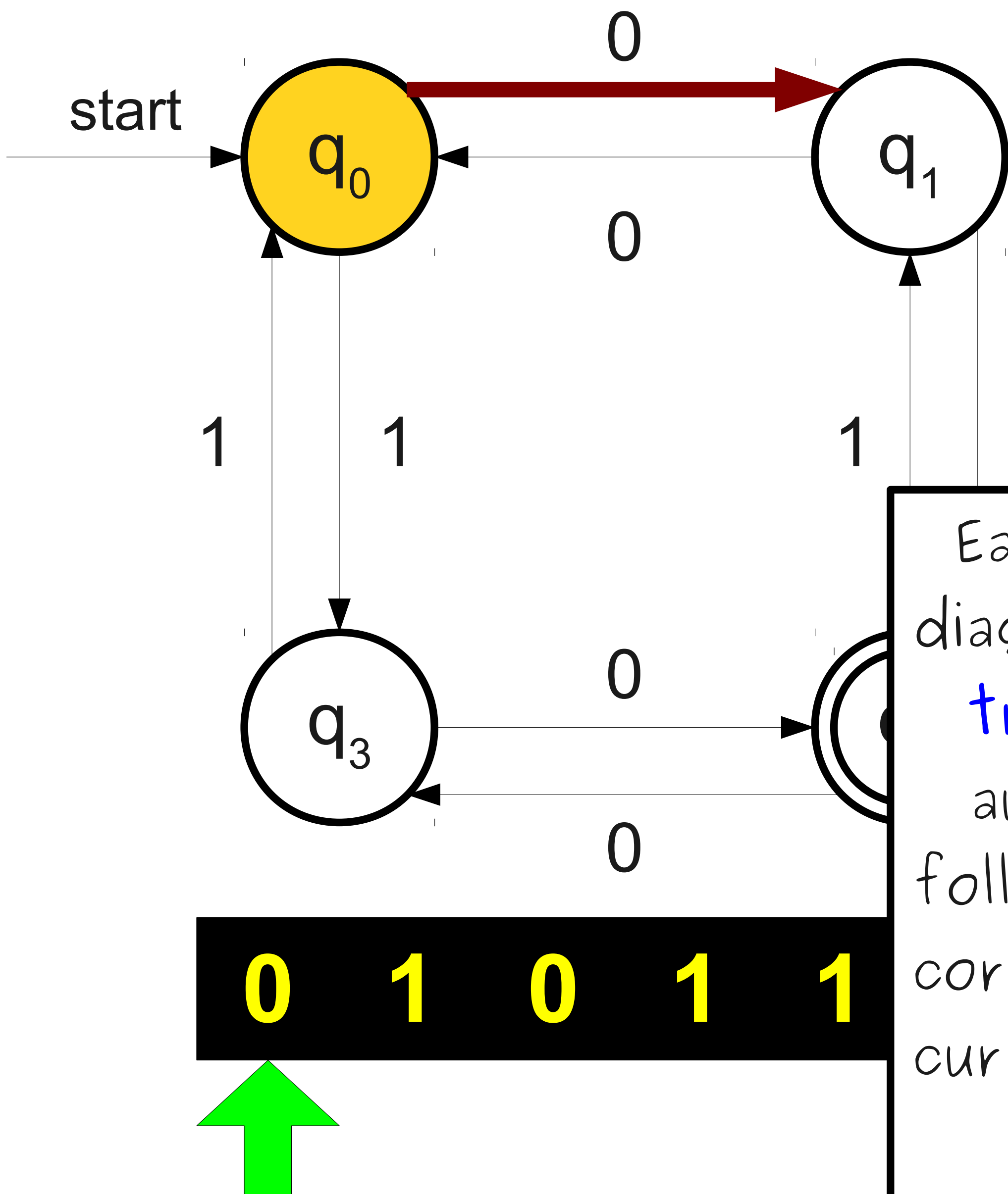
A Simple Finite Automaton



A Simple Finite Automaton

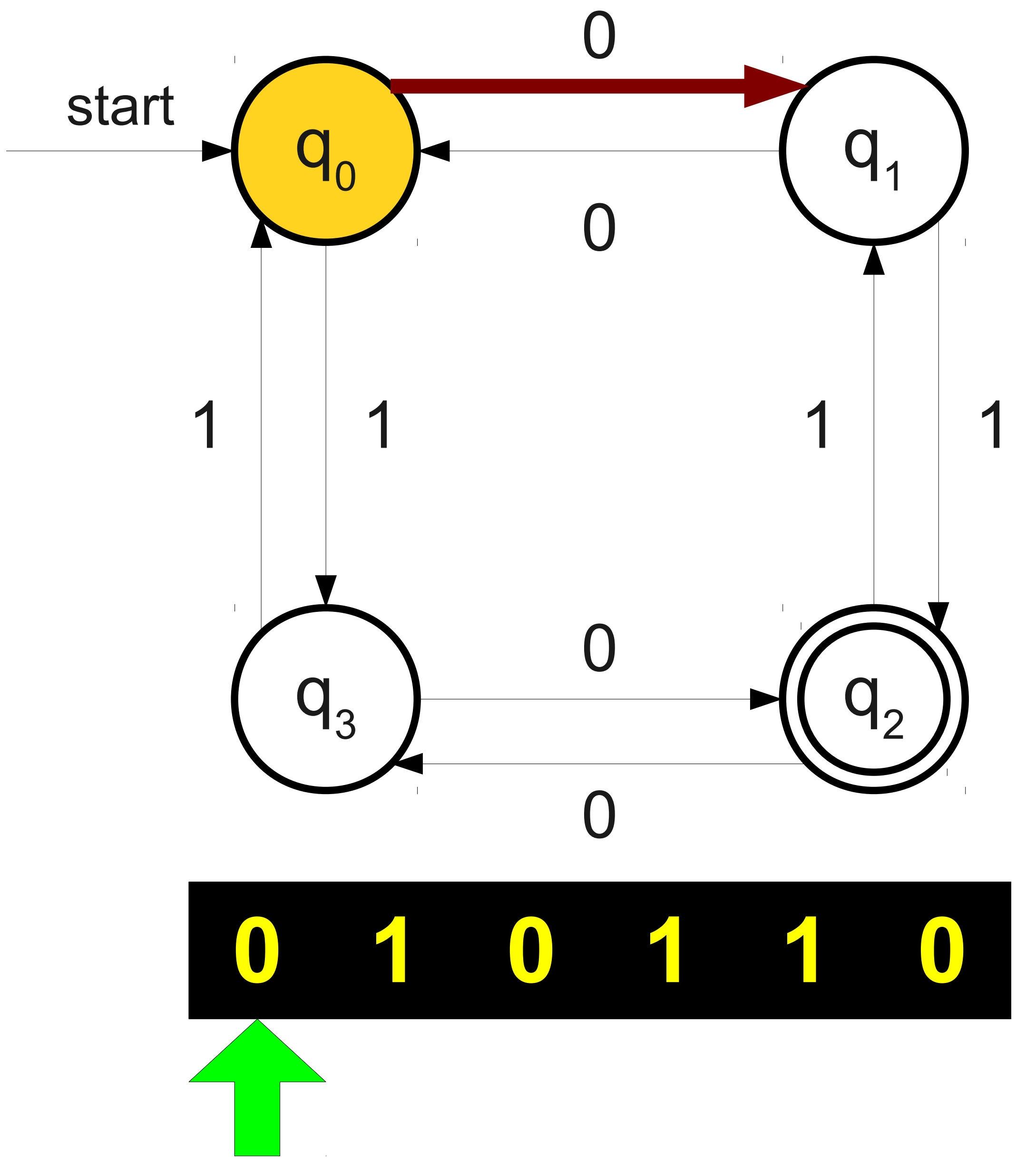


A Simple Finite Automaton

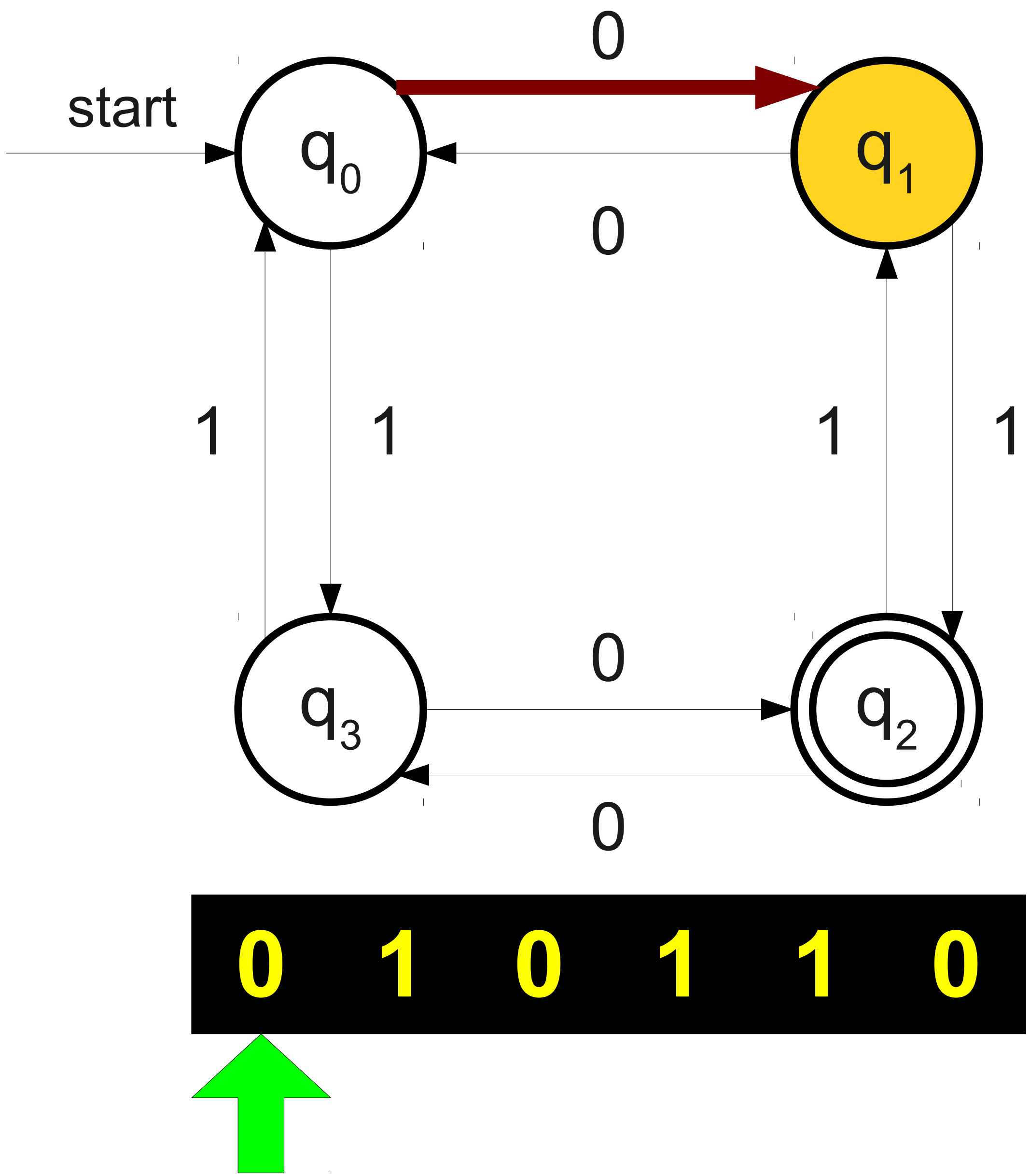


Each arrow in this diagram represents a **transition**. The automaton always follows the transition corresponding to the current symbol being read.

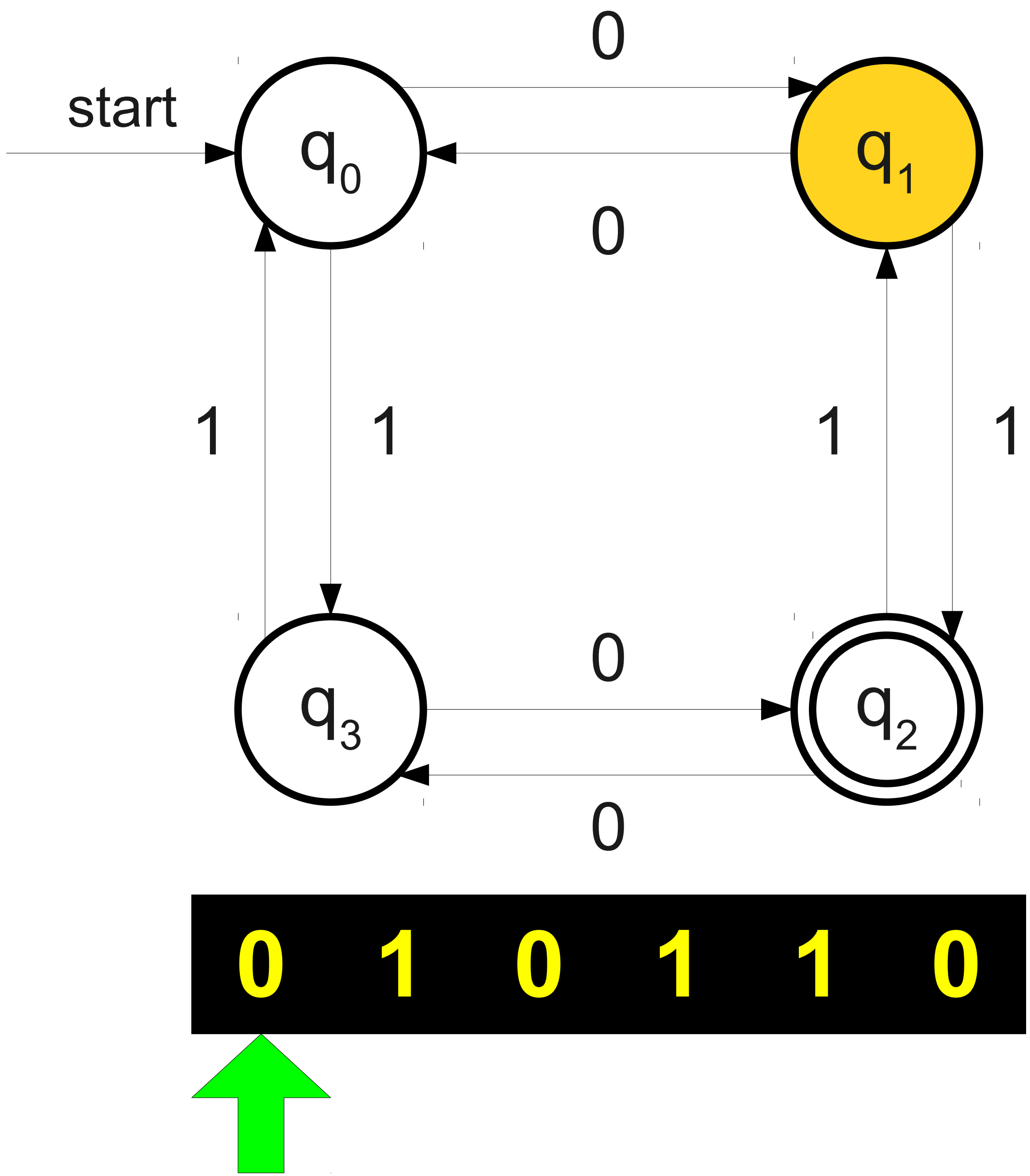
A Simple Finite Automaton



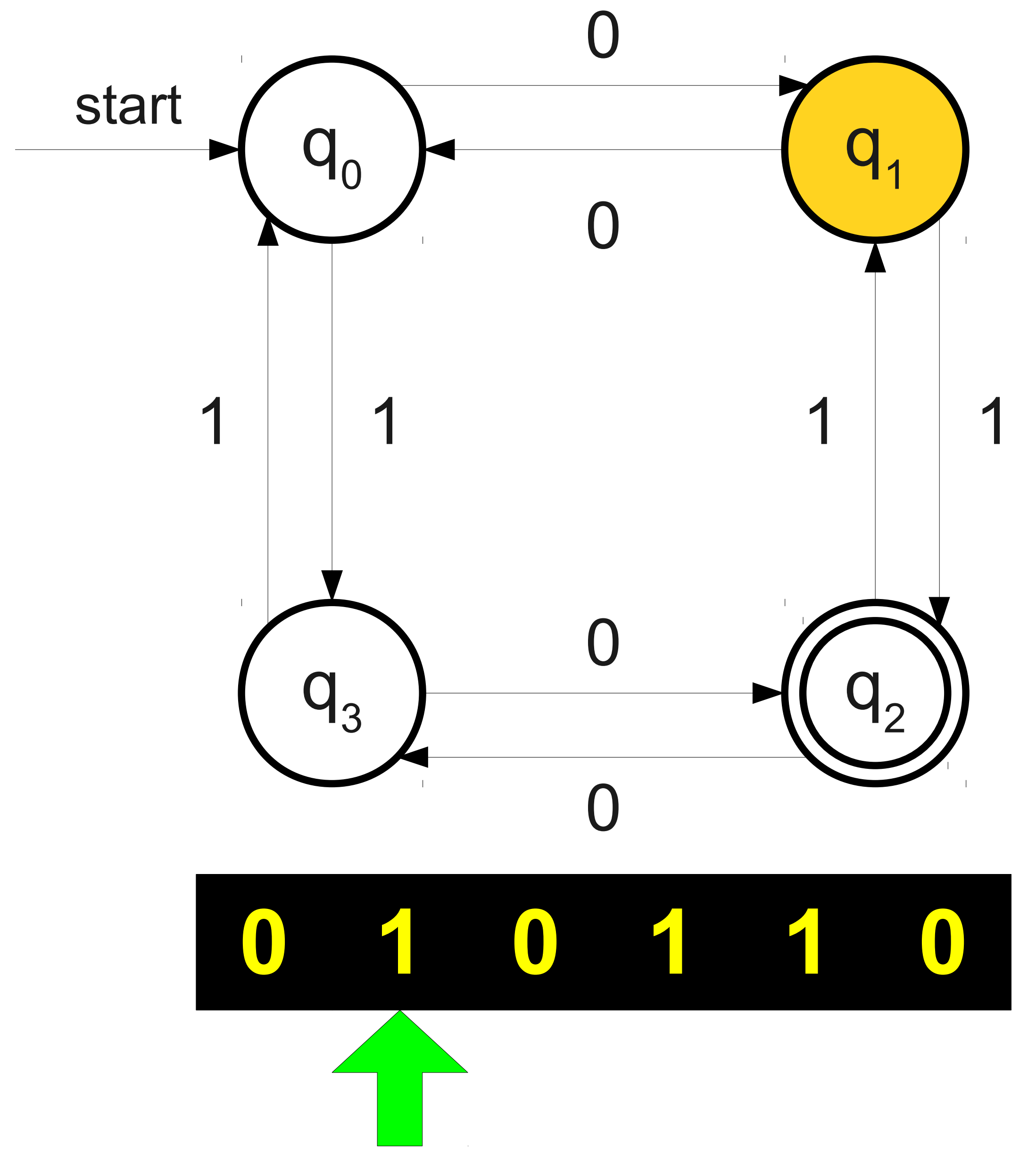
A Simple Finite Automaton



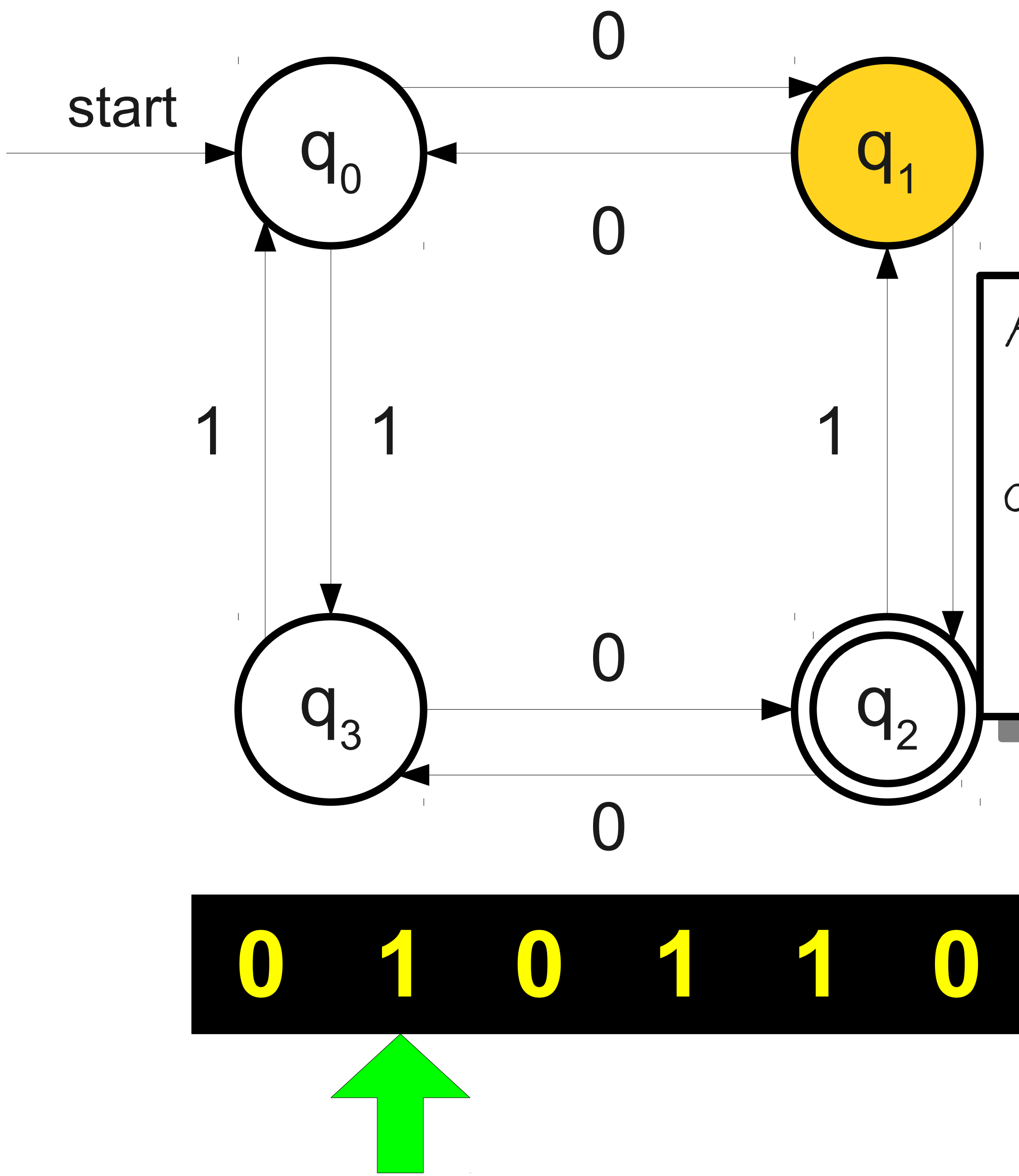
A Simple Finite Automaton



A Simple Finite Automaton

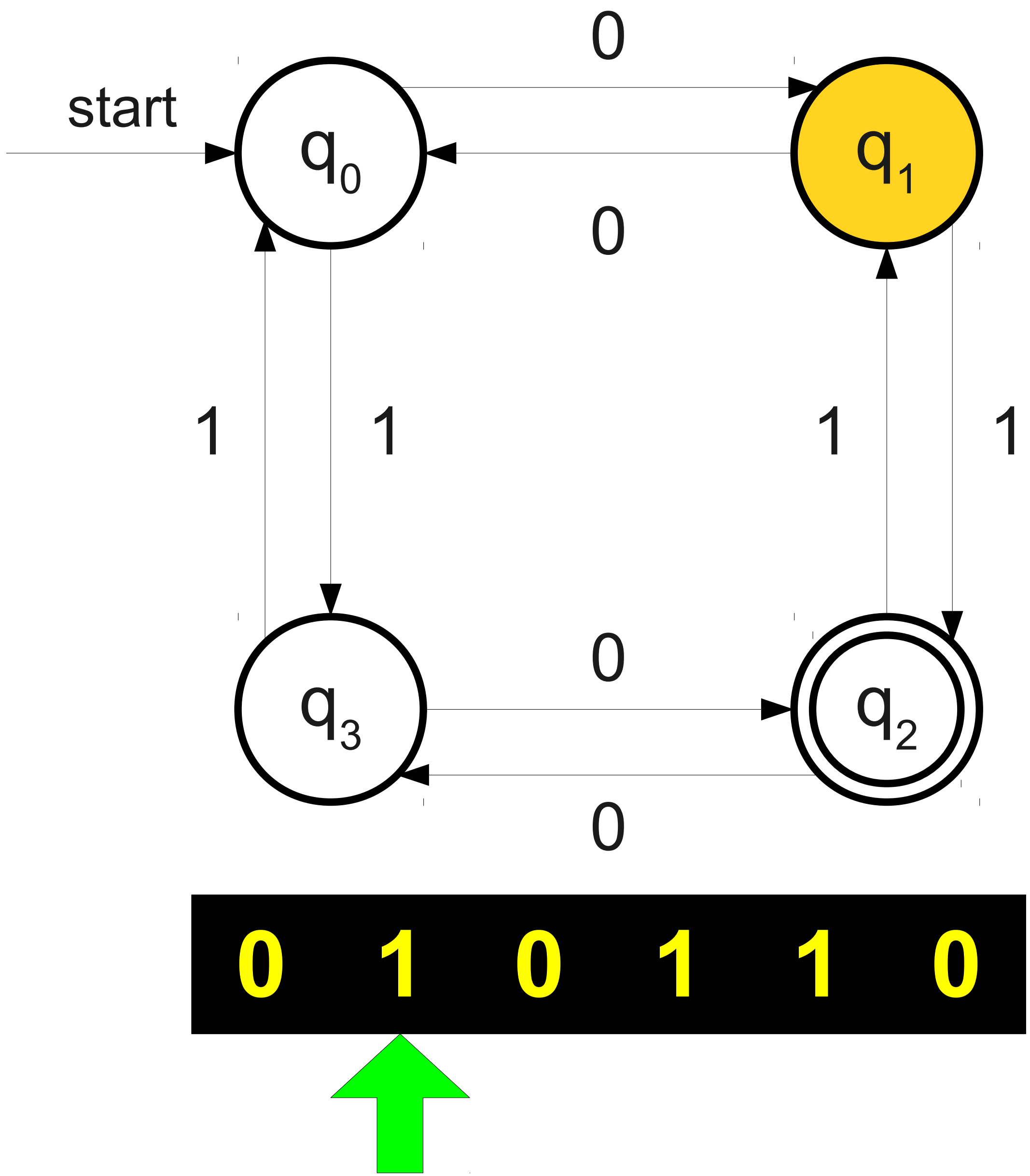


A Simple Finite Automaton

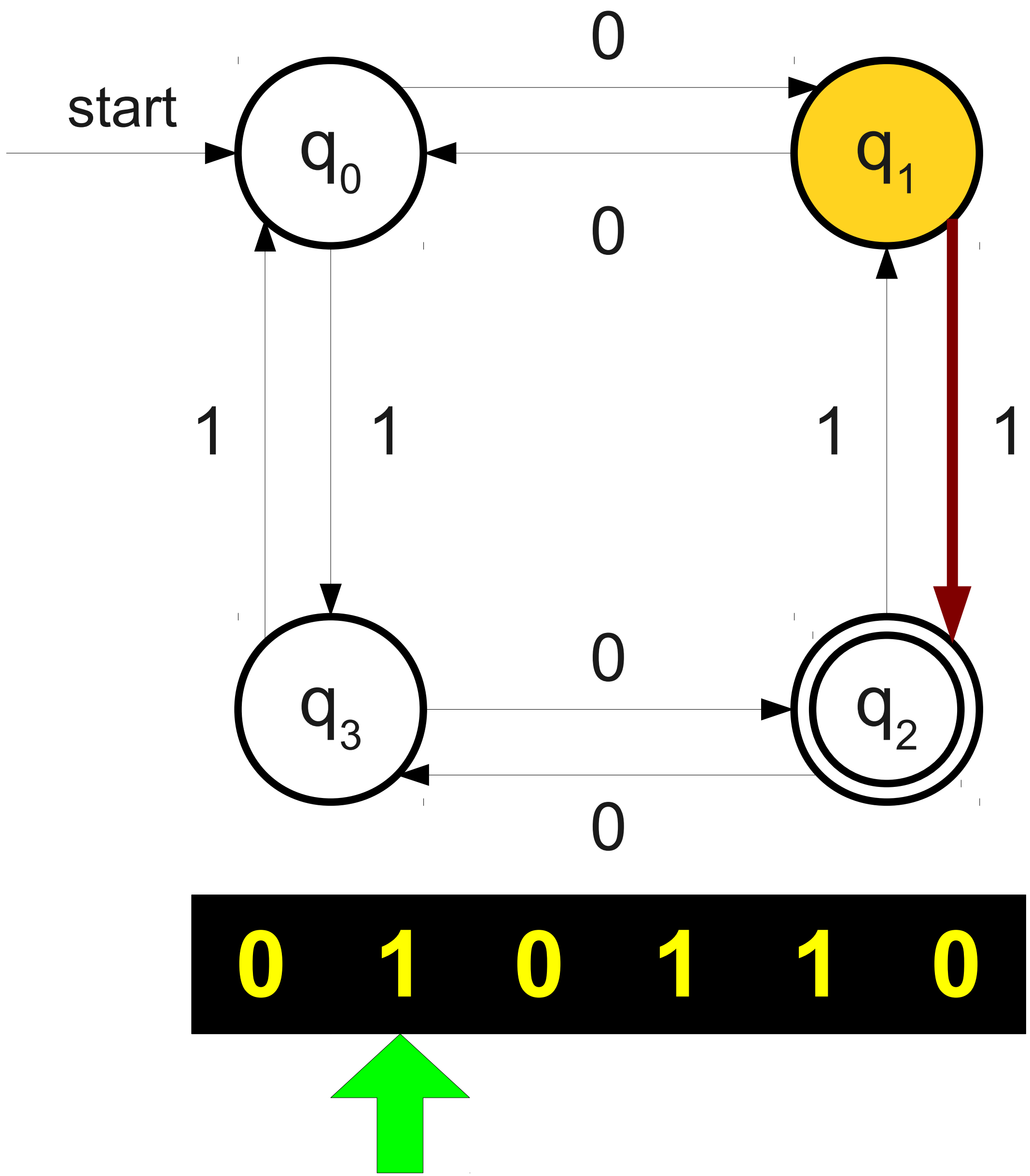


After transitioning, the automaton considers the next symbol in the input.

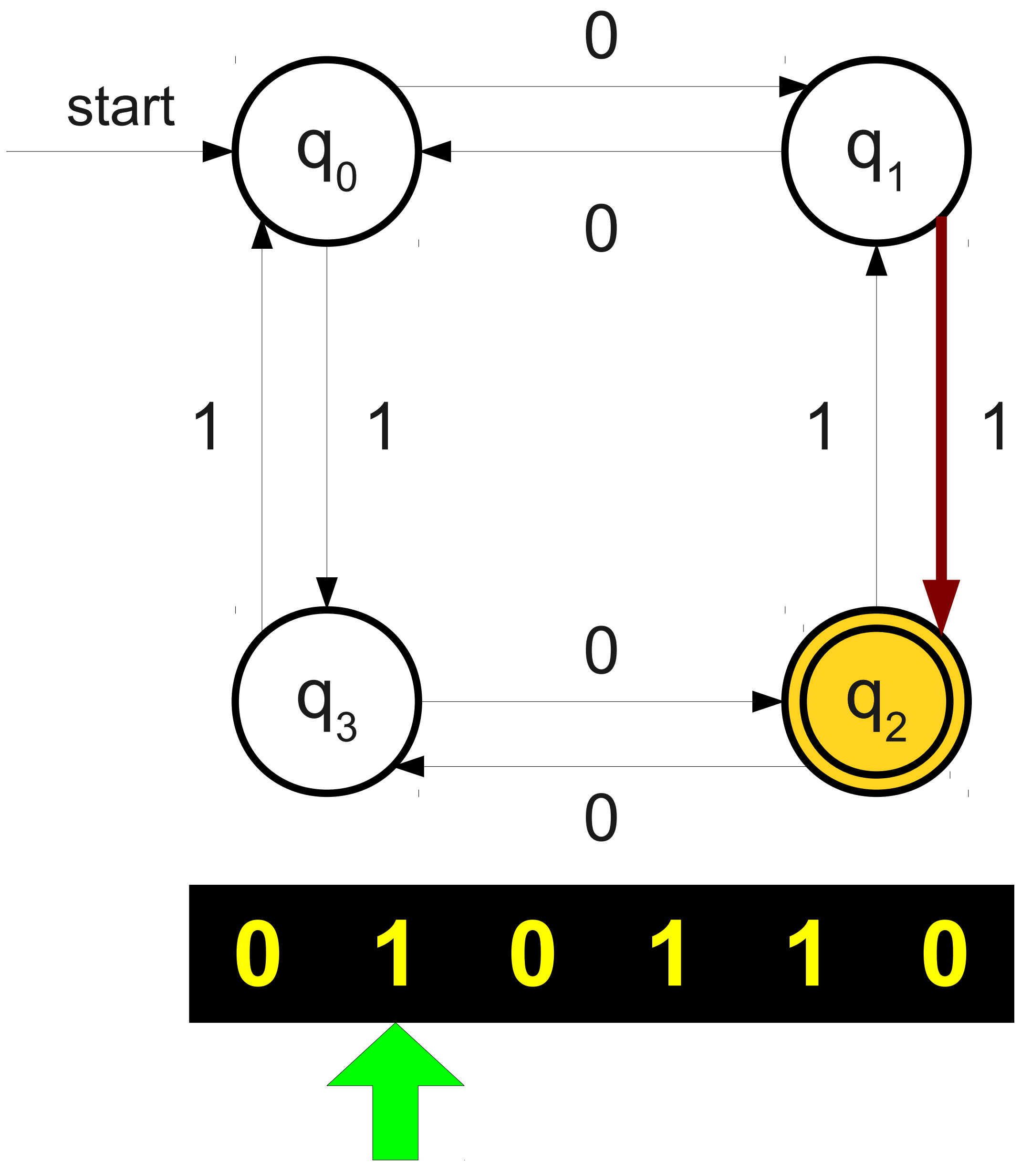
A Simple Finite Automaton



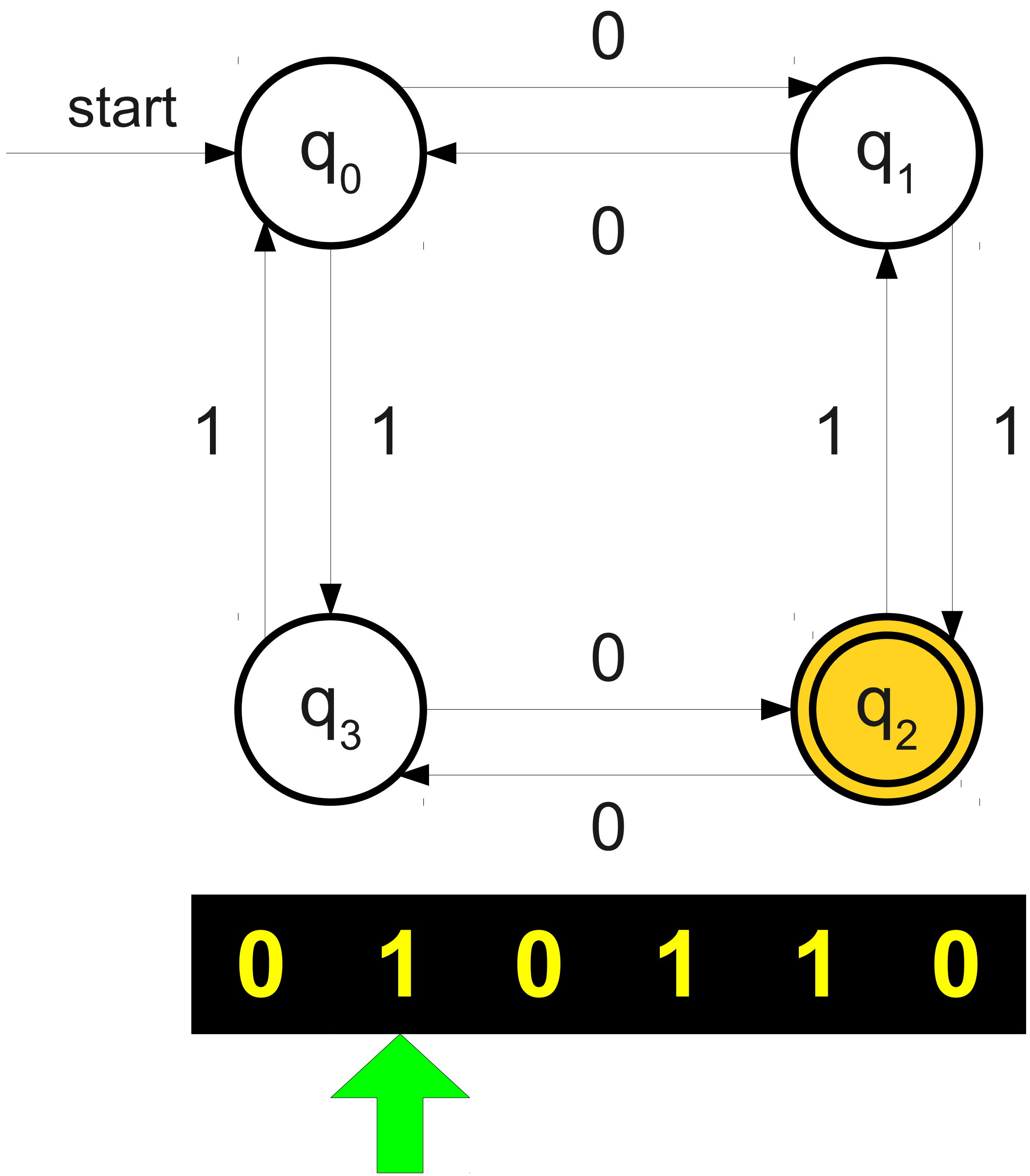
A Simple Finite Automaton



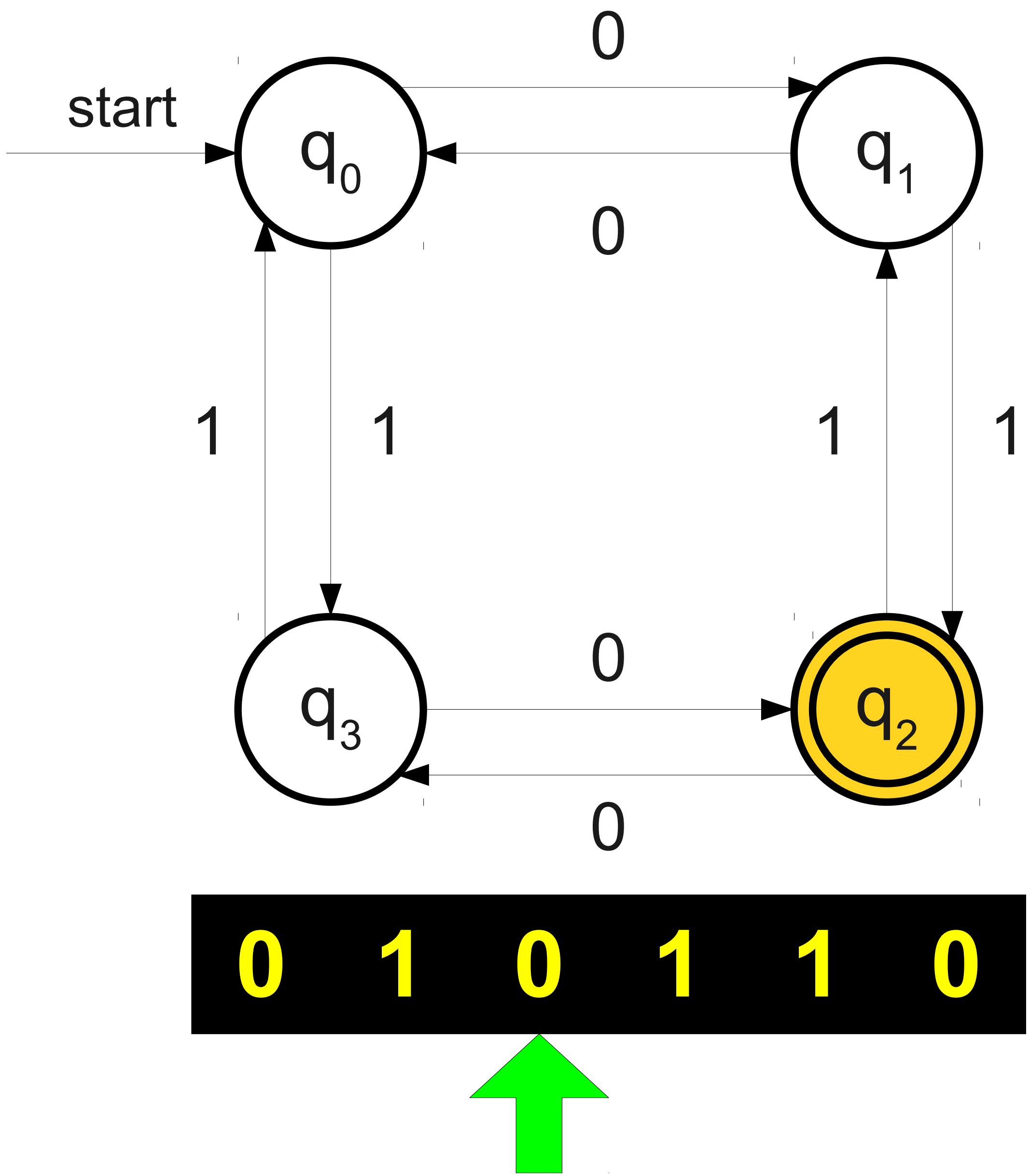
A Simple Finite Automaton



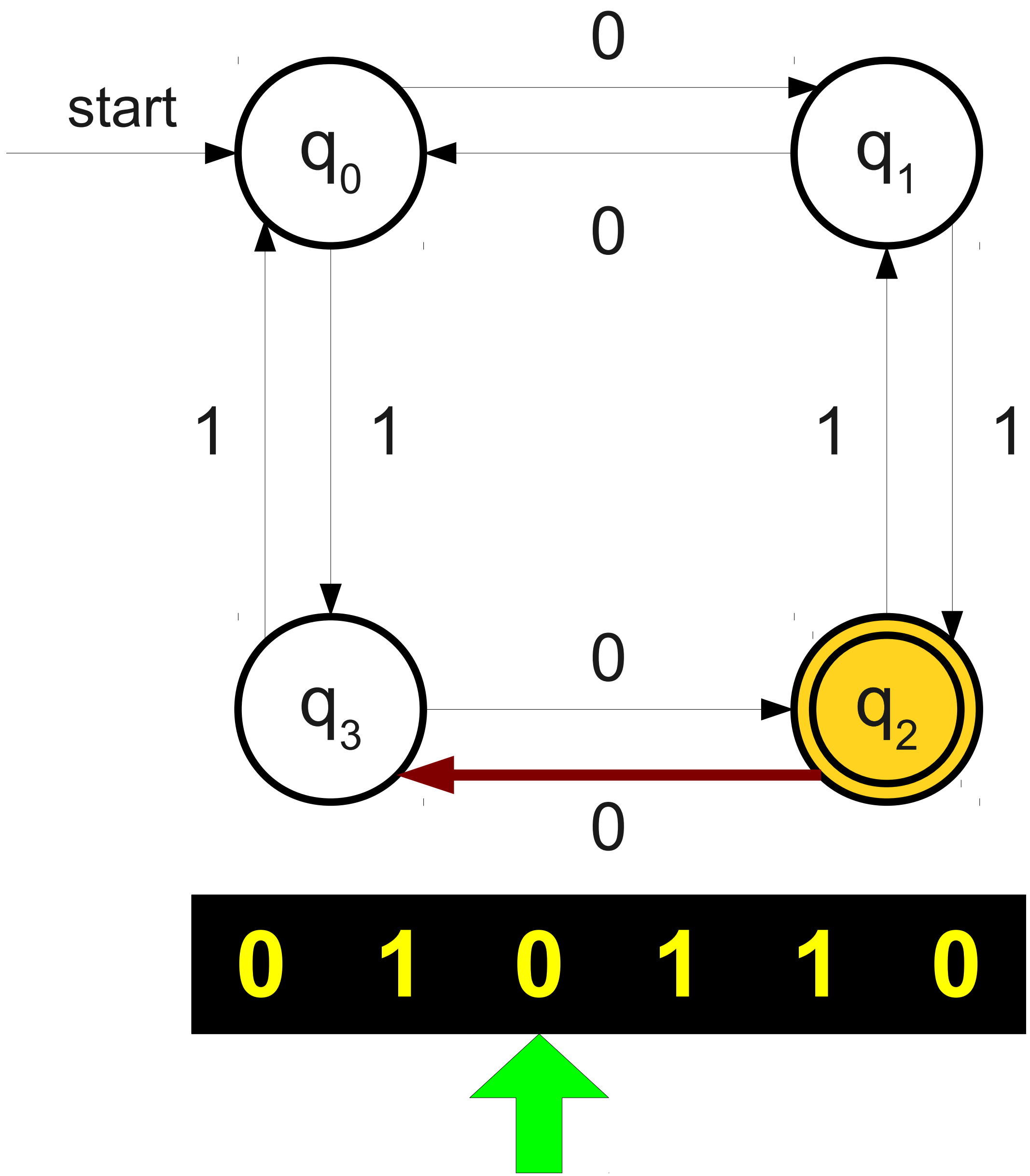
A Simple Finite Automaton



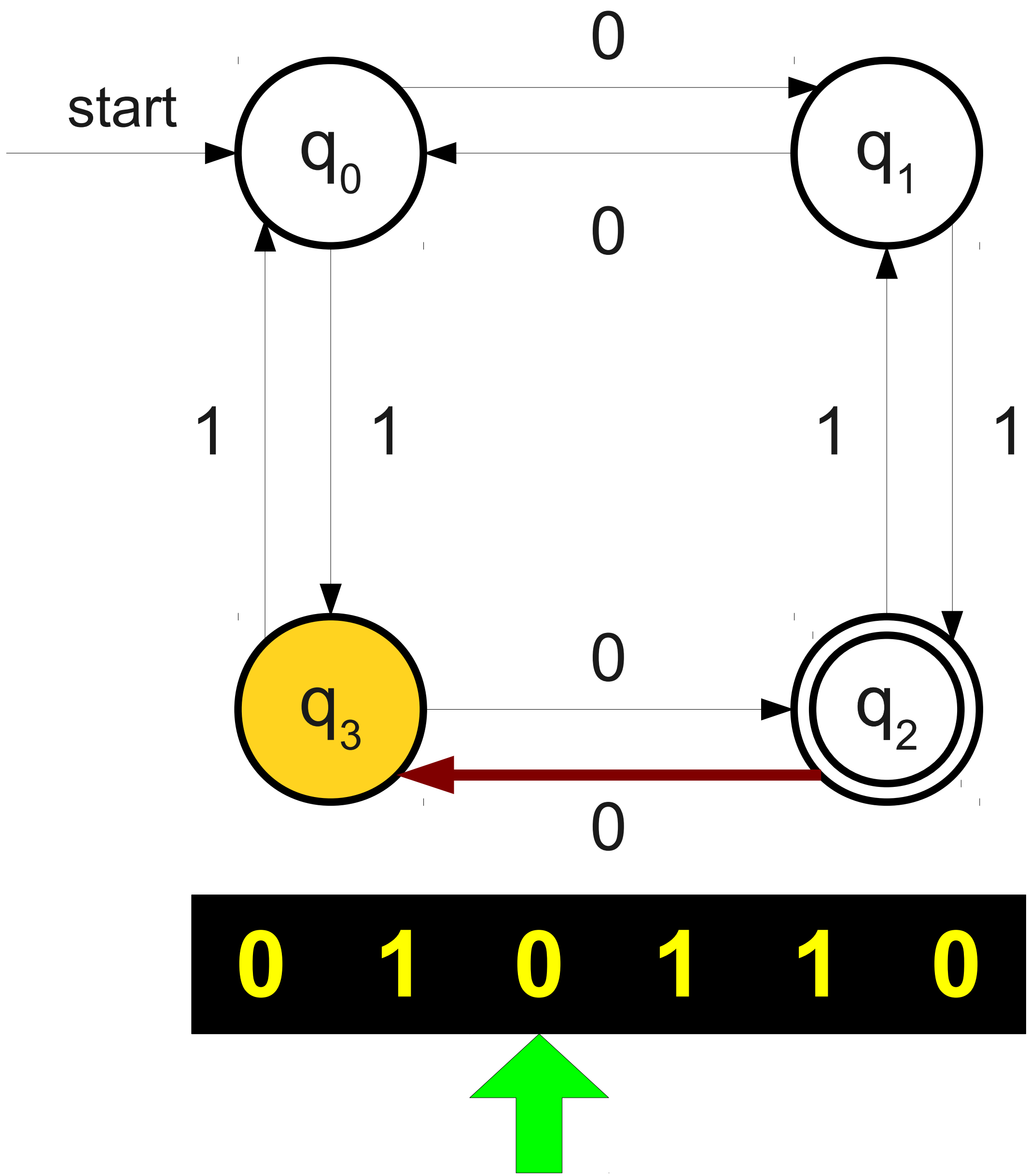
A Simple Finite Automaton



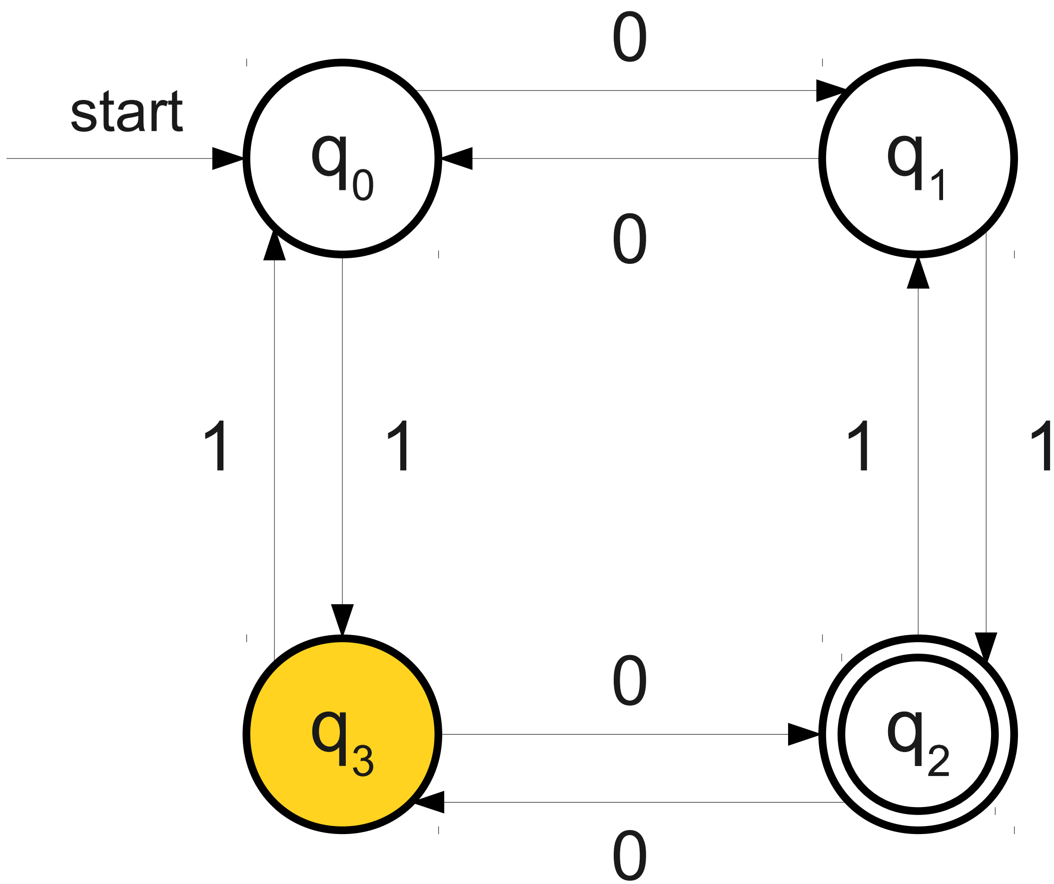
A Simple Finite Automaton



A Simple Finite Automaton

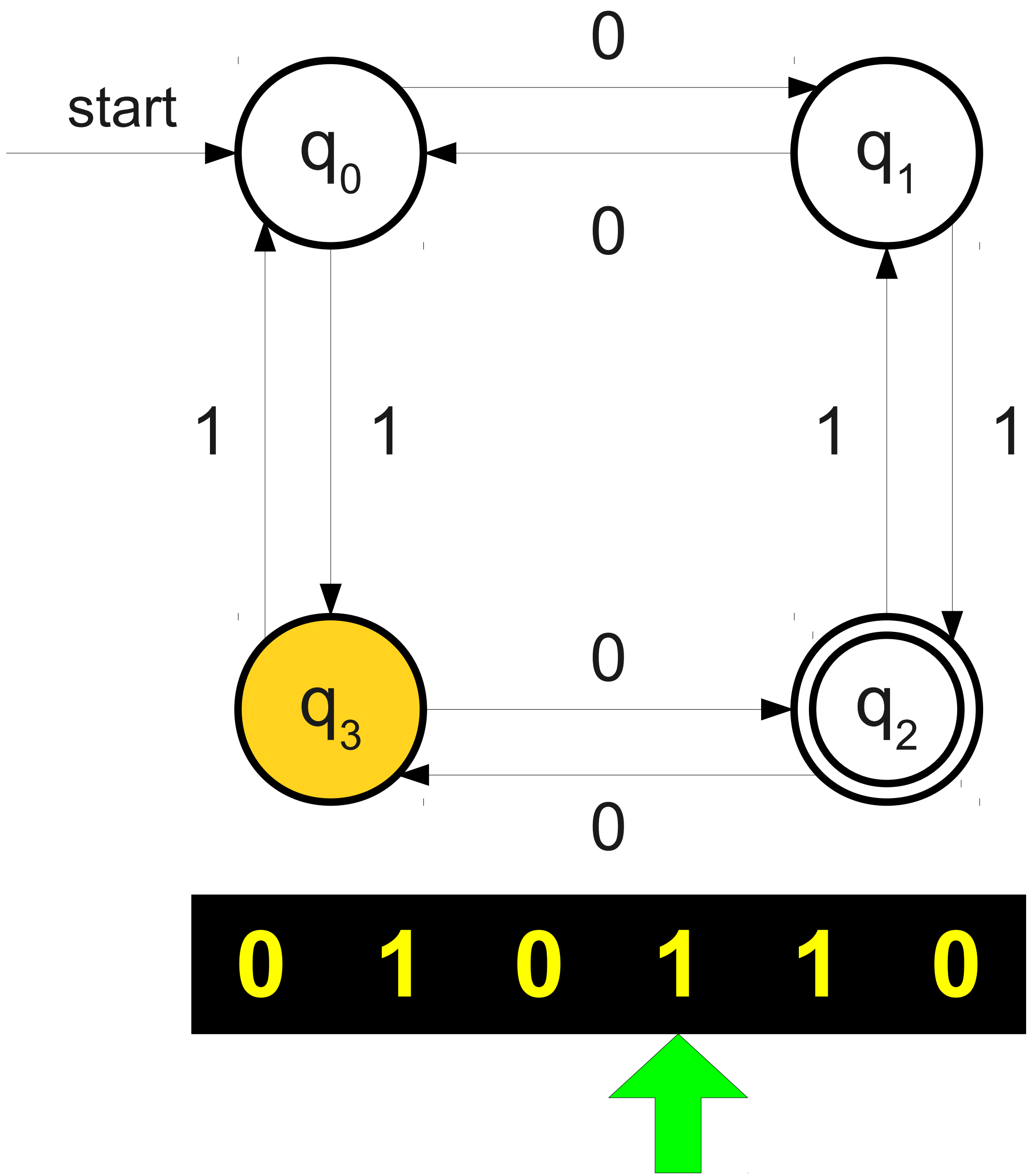


A Simple Finite Automaton

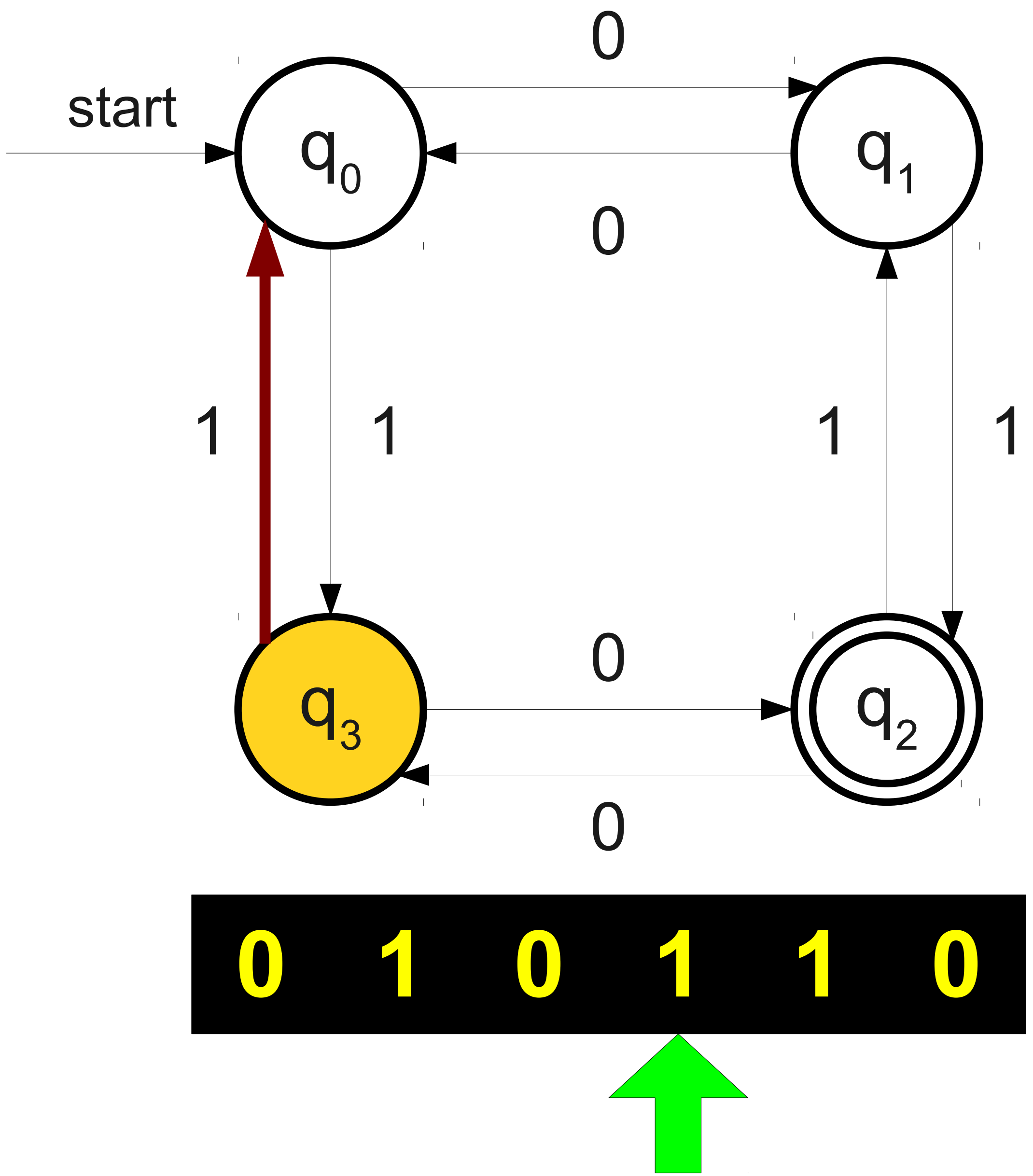


0	1	0	1	1	0

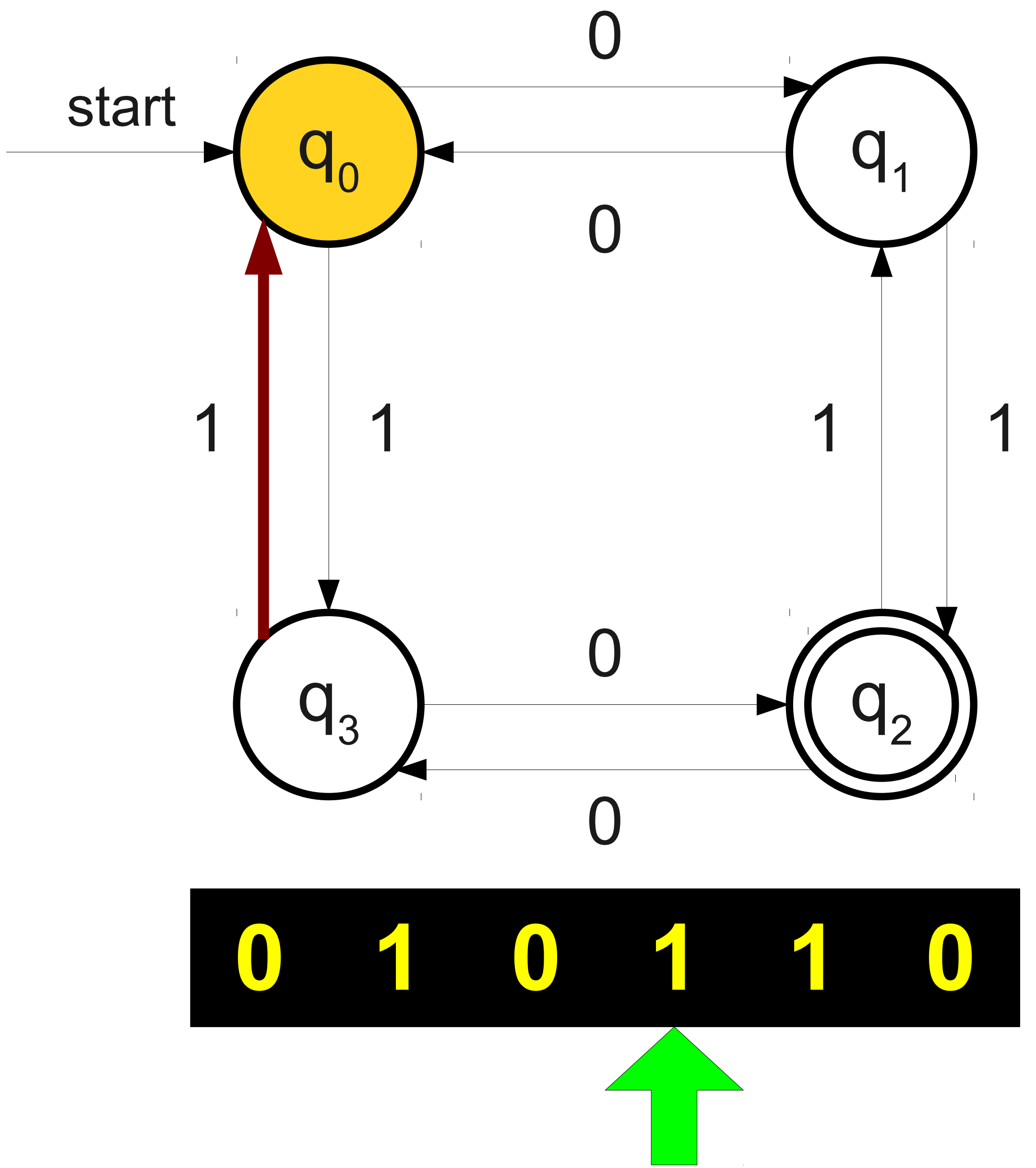
A Simple Finite Automaton



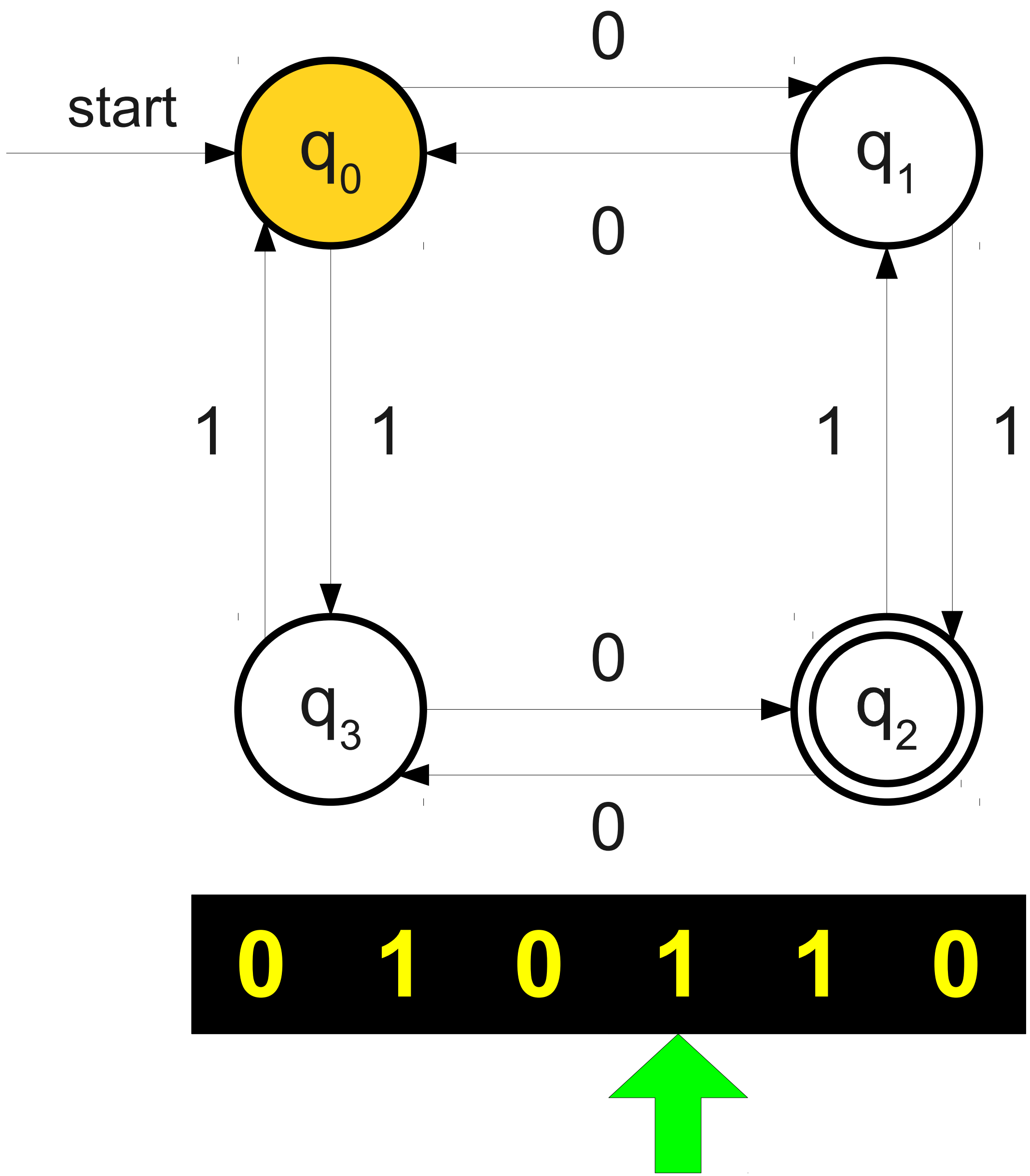
A Simple Finite Automaton



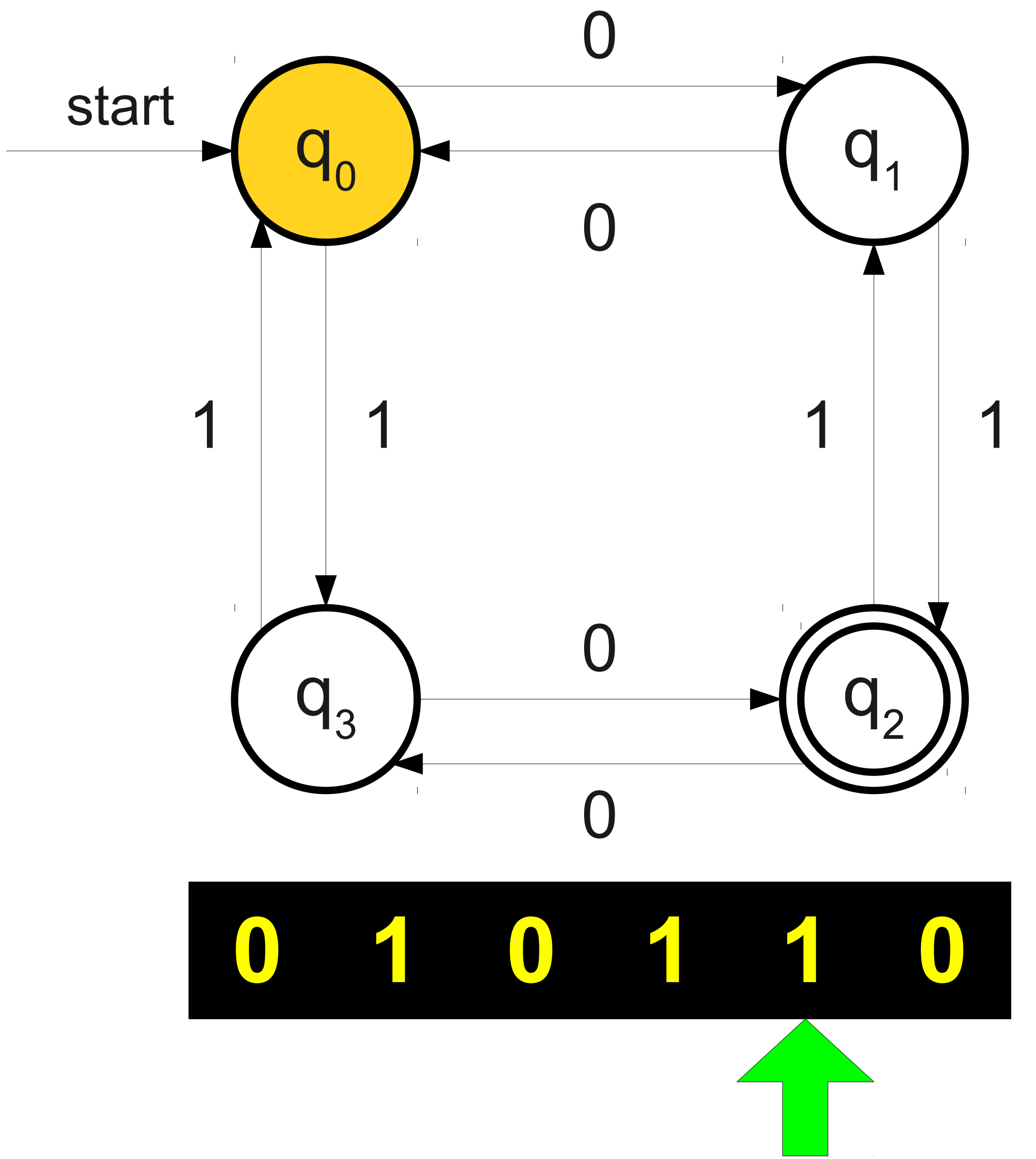
A Simple Finite Automaton



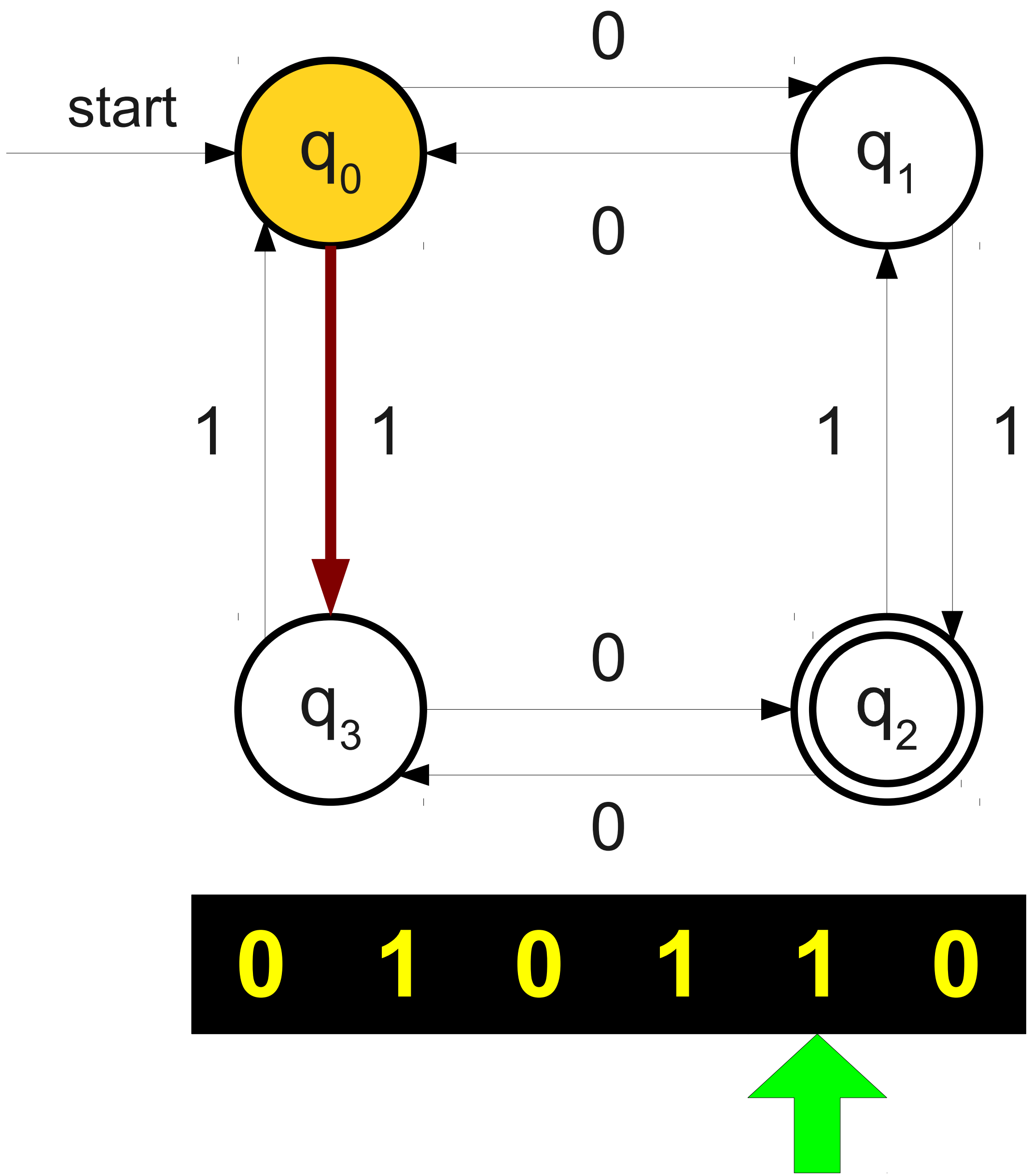
A Simple Finite Automaton



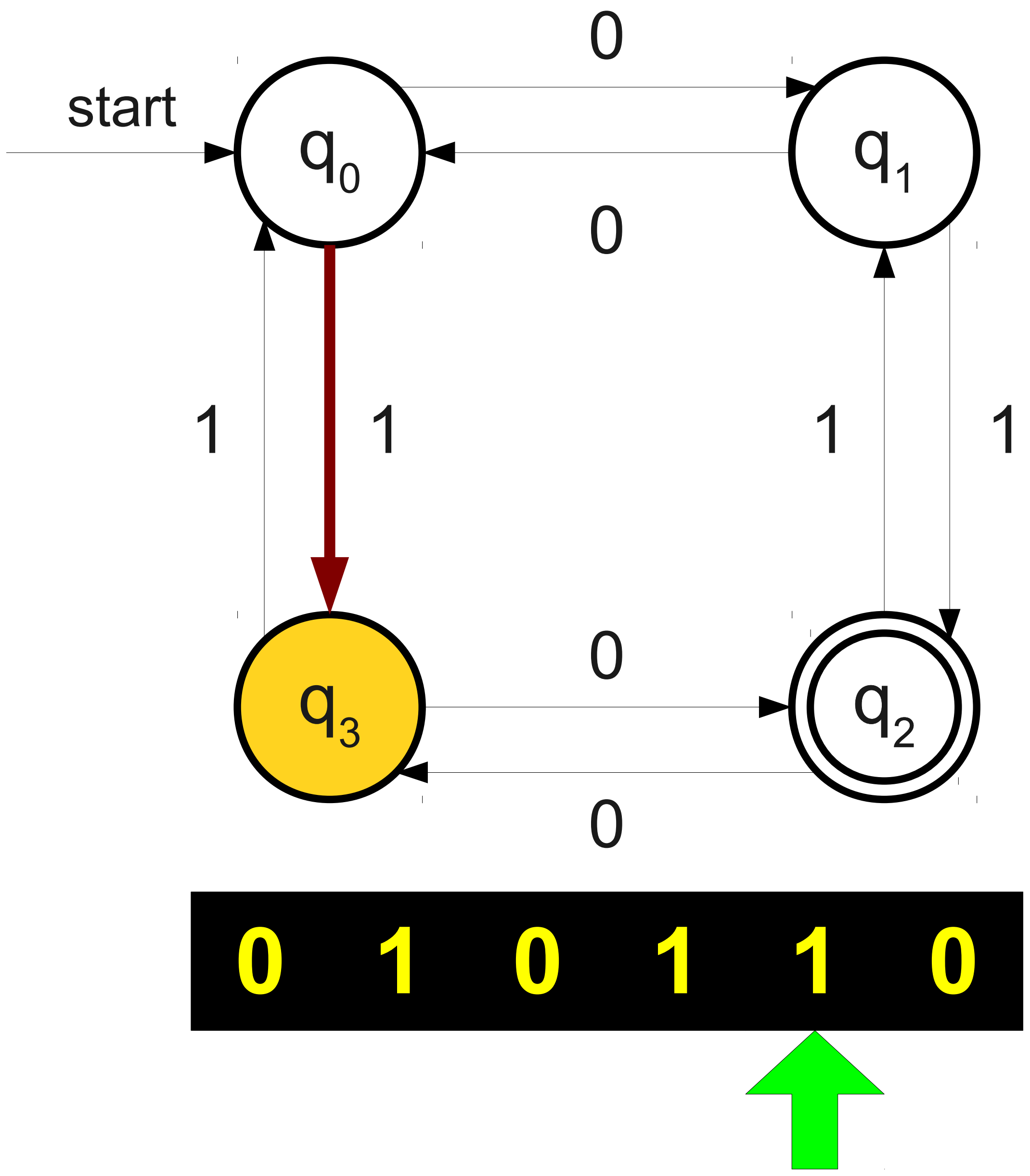
A Simple Finite Automaton



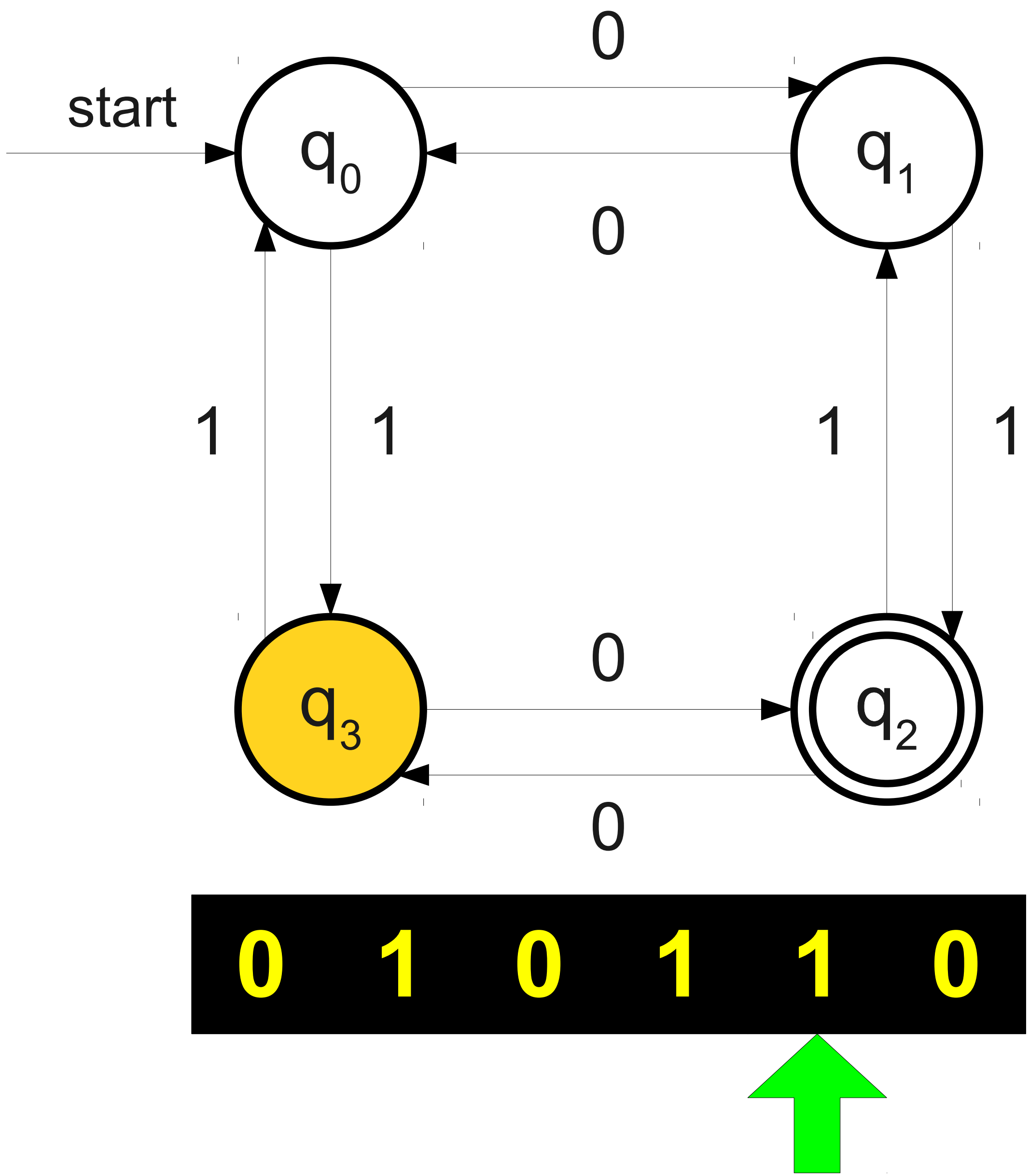
A Simple Finite Automaton



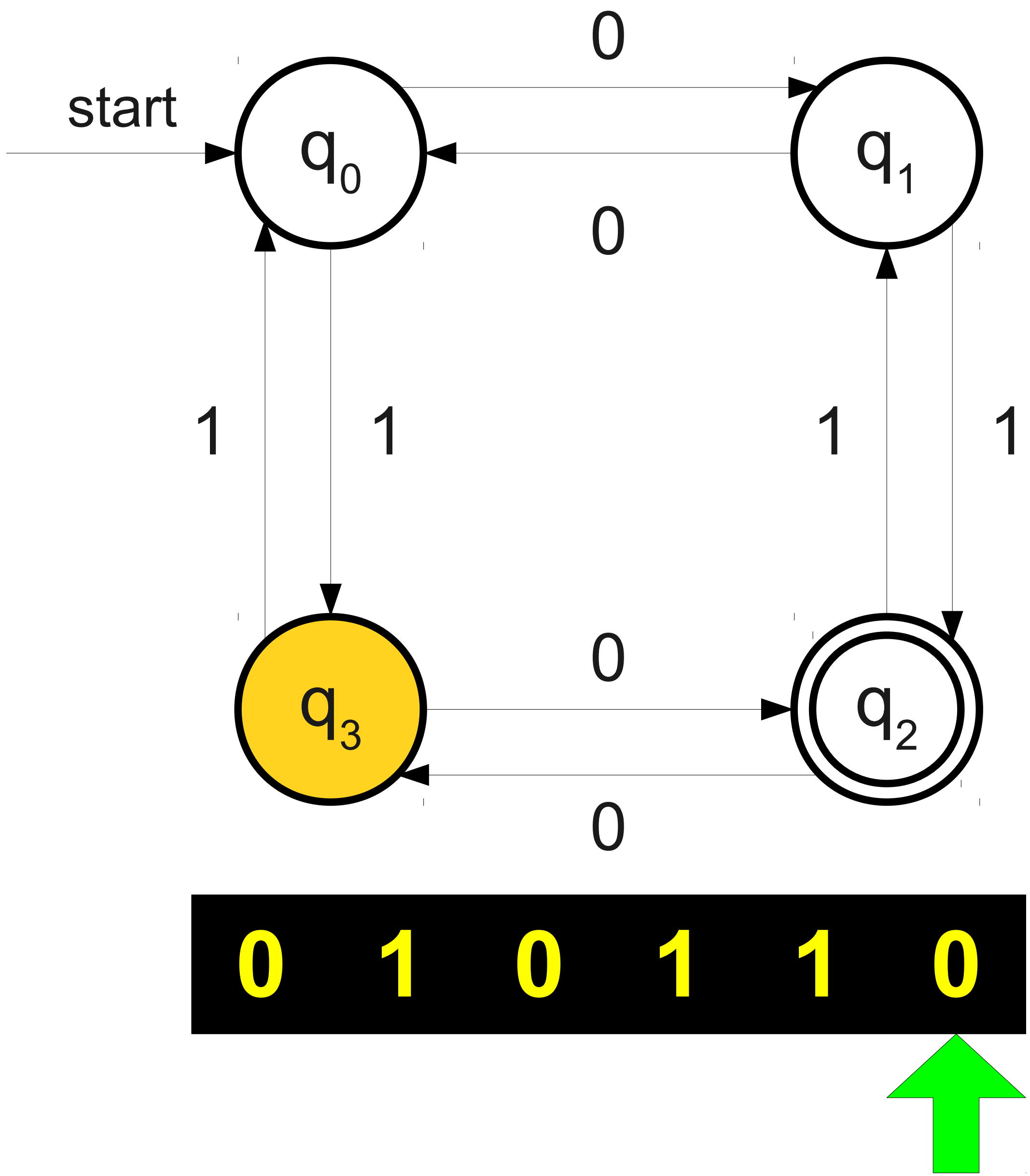
A Simple Finite Automaton



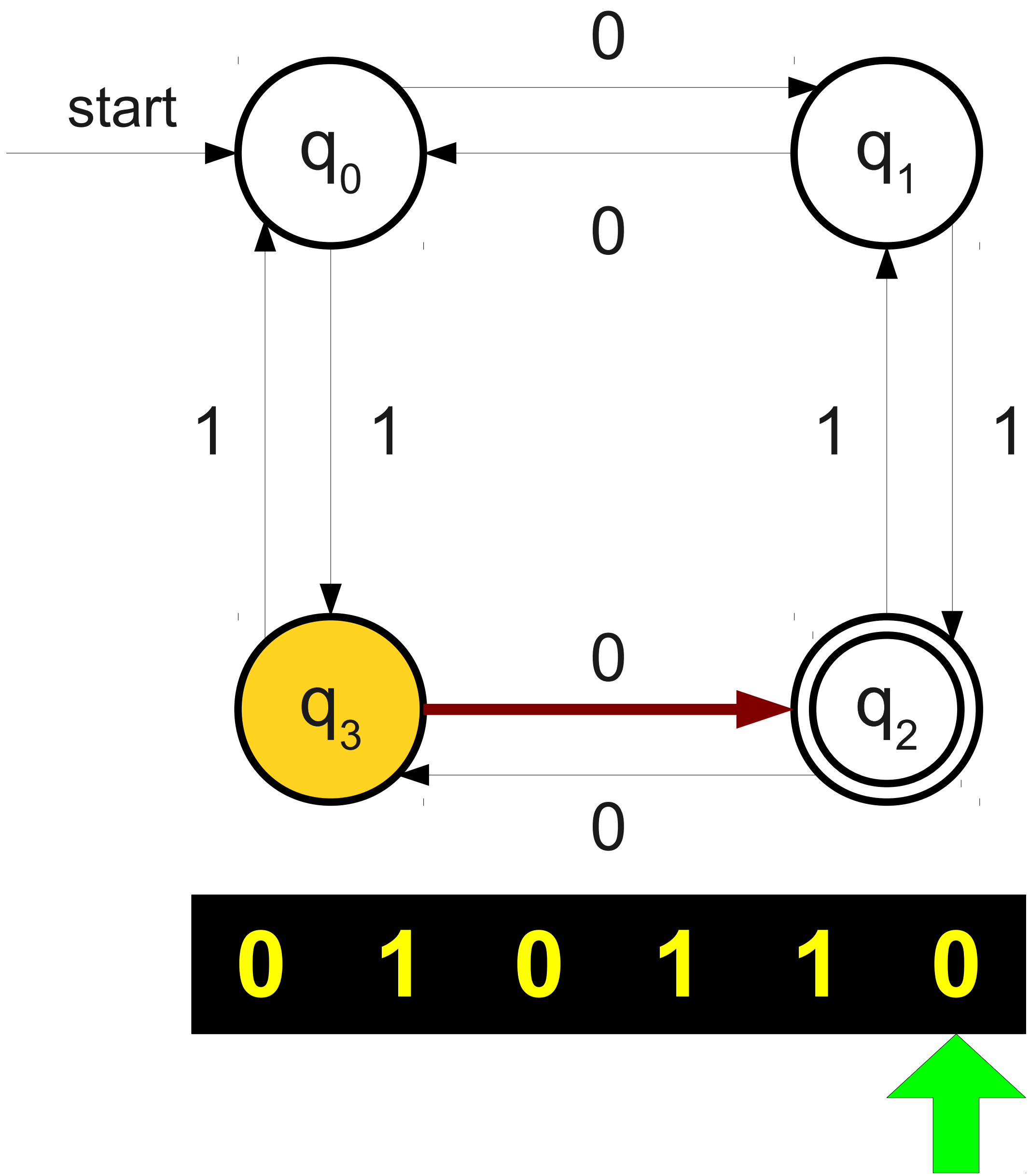
A Simple Finite Automaton



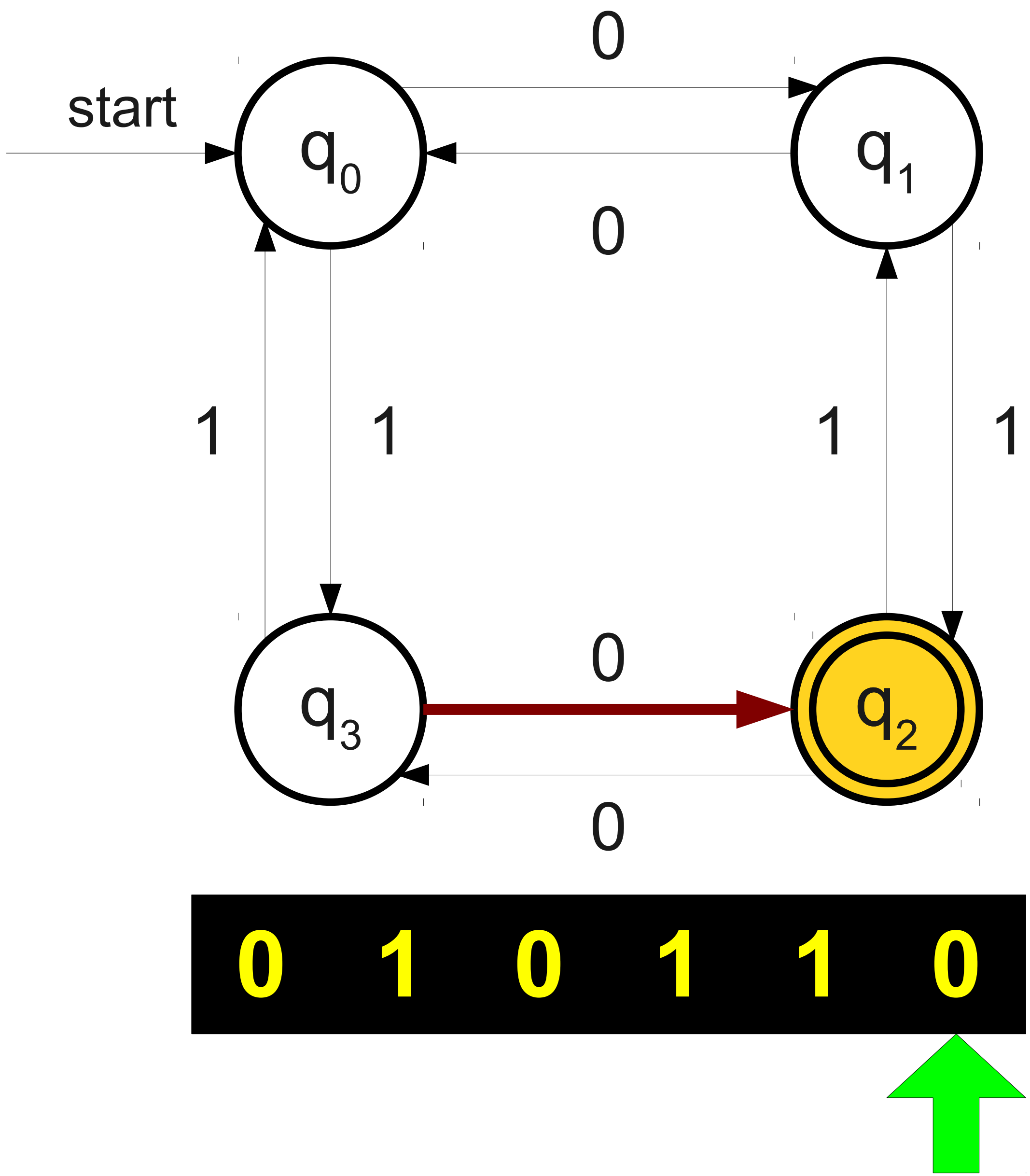
A Simple Finite Automaton



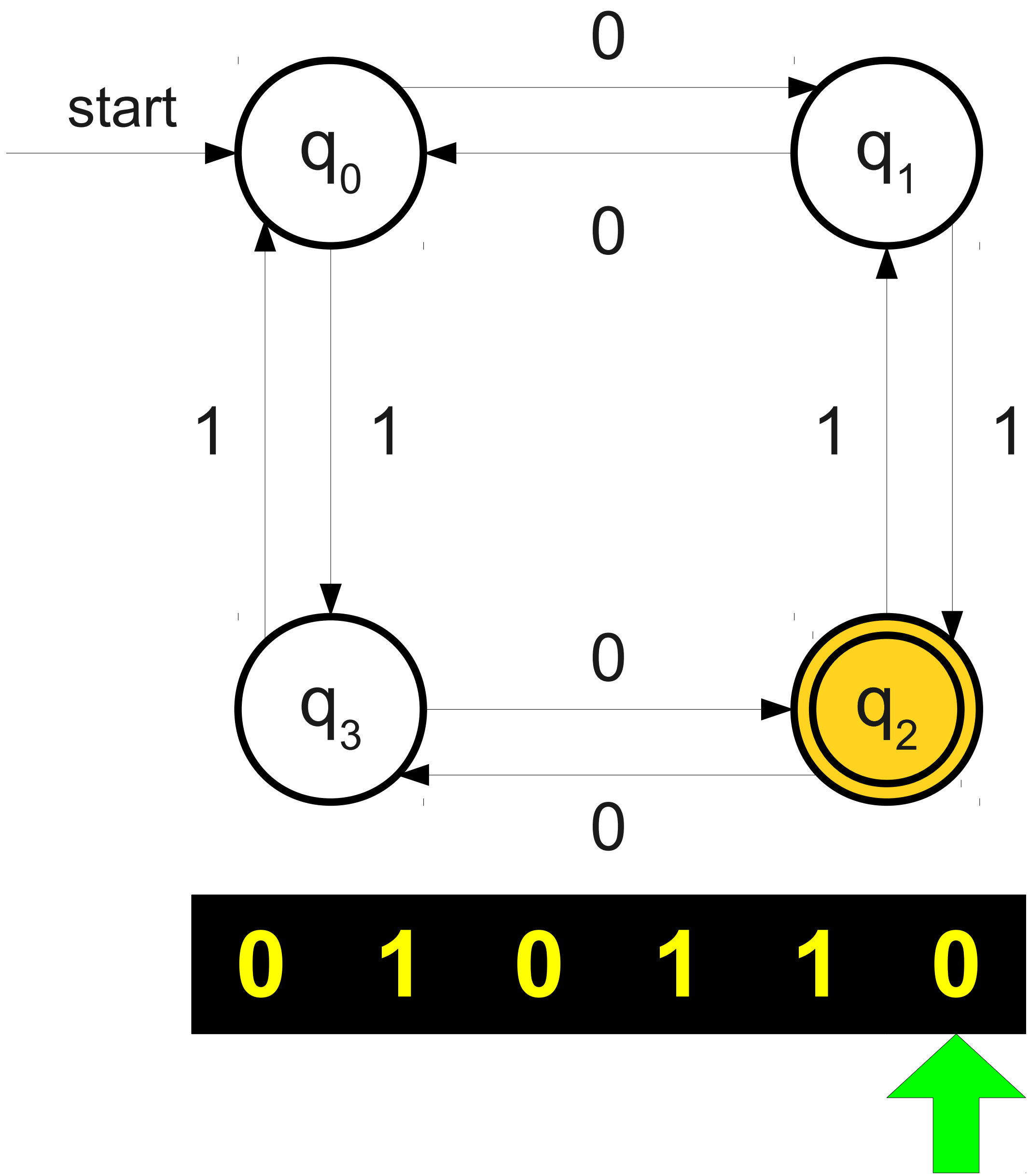
A Simple Finite Automaton



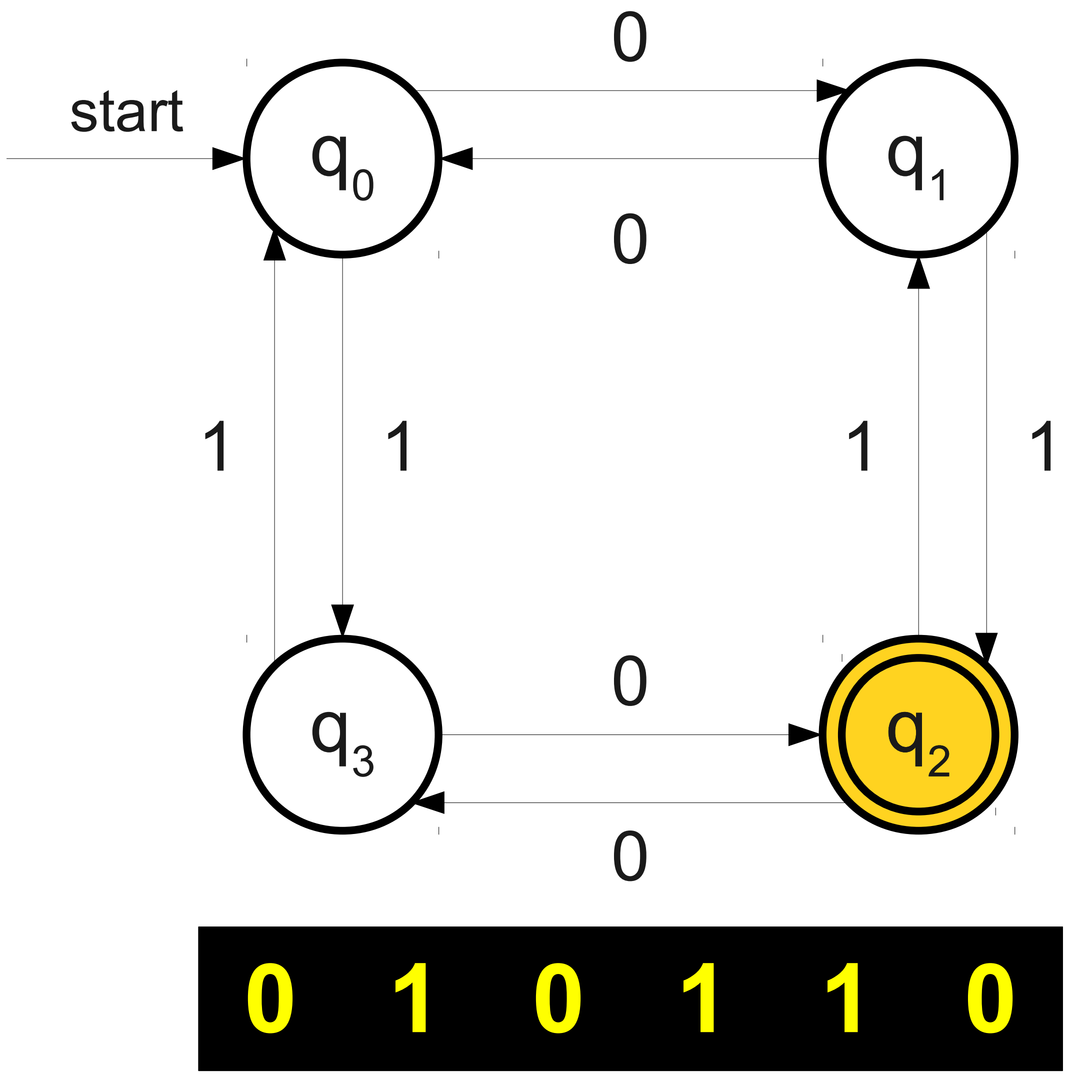
A Simple Finite Automaton



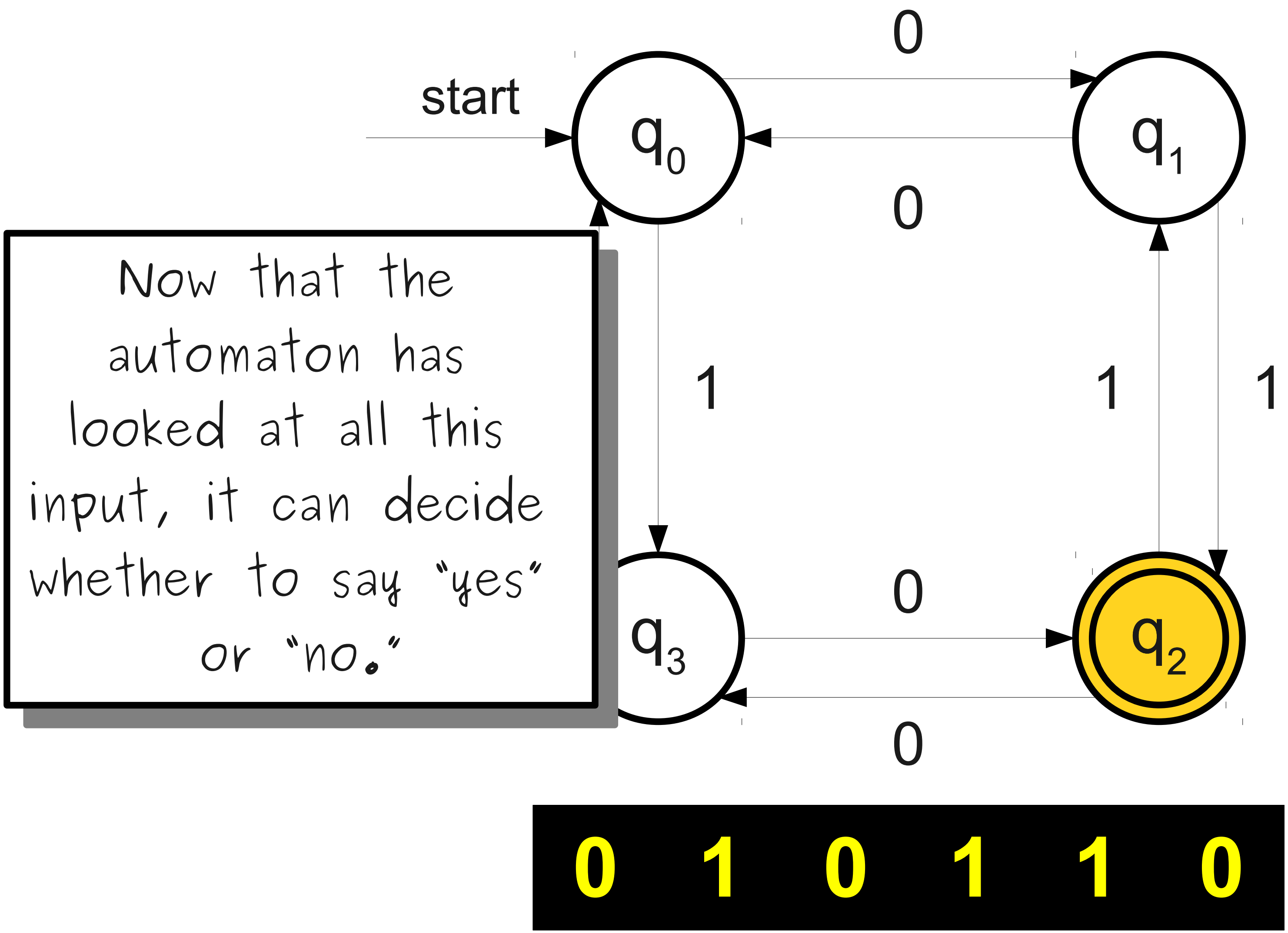
A Simple Finite Automaton



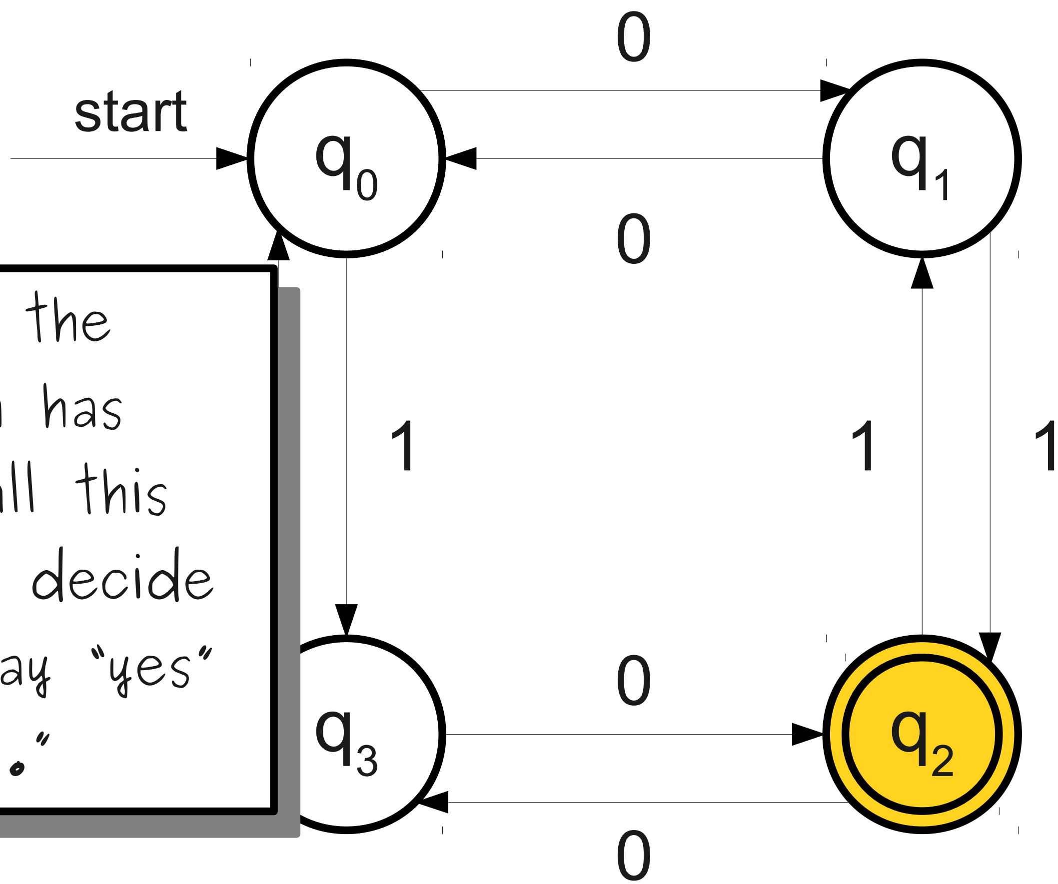
A Simple Finite Automaton



A Simple Finite Automaton



A Simple Finite Automaton

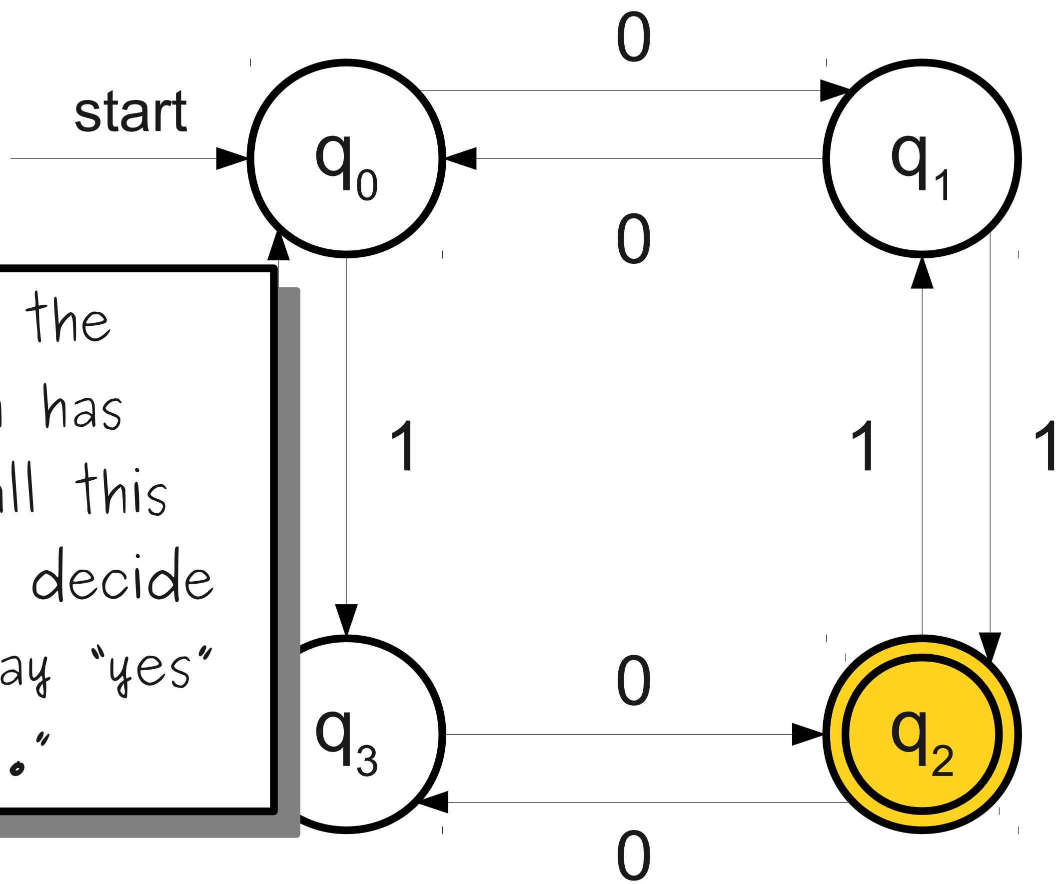


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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



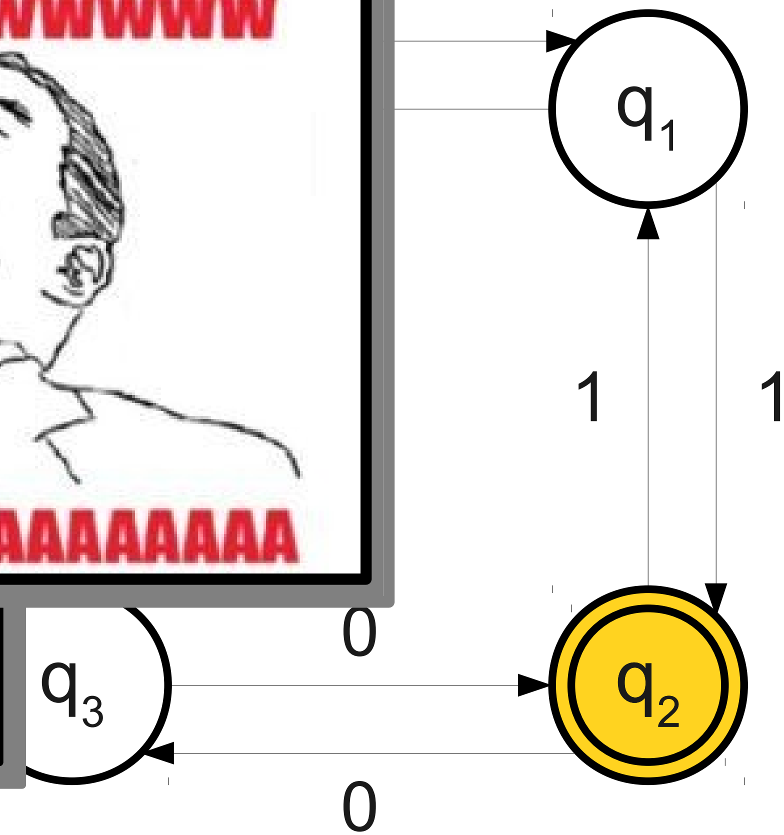
Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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

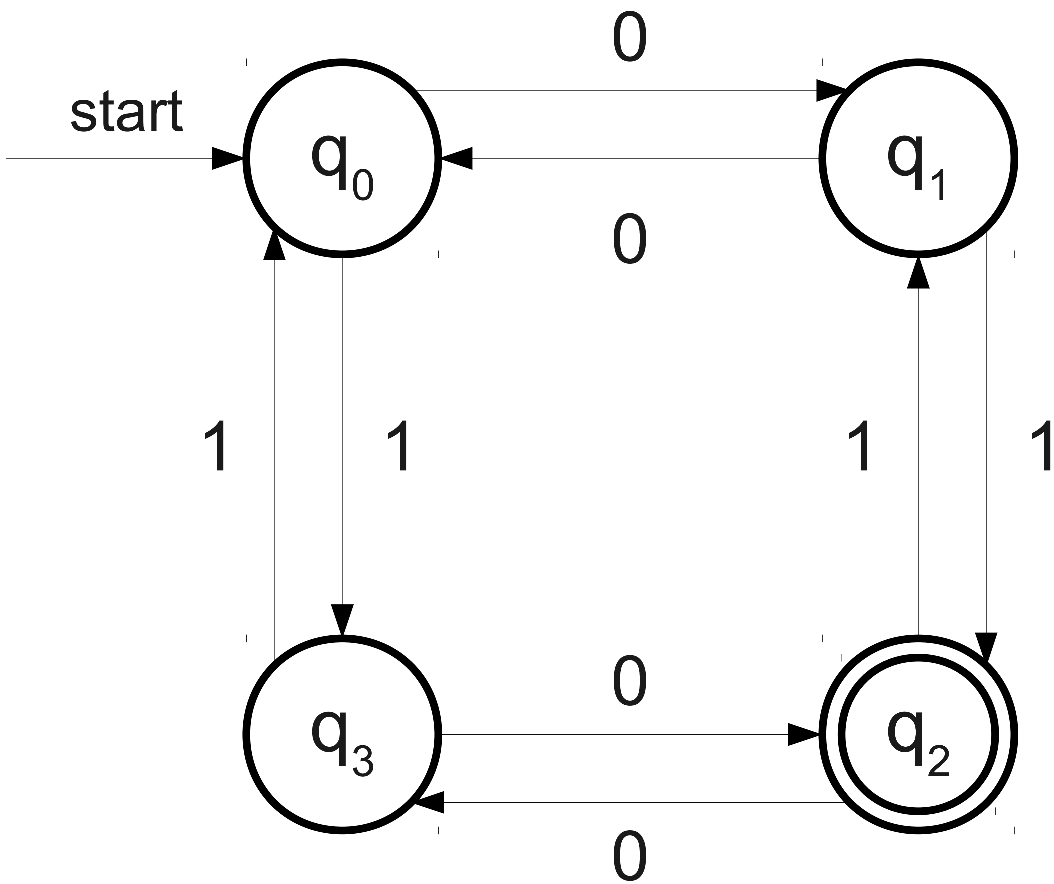
lo
inpu
whether to say "yes"
or "no."



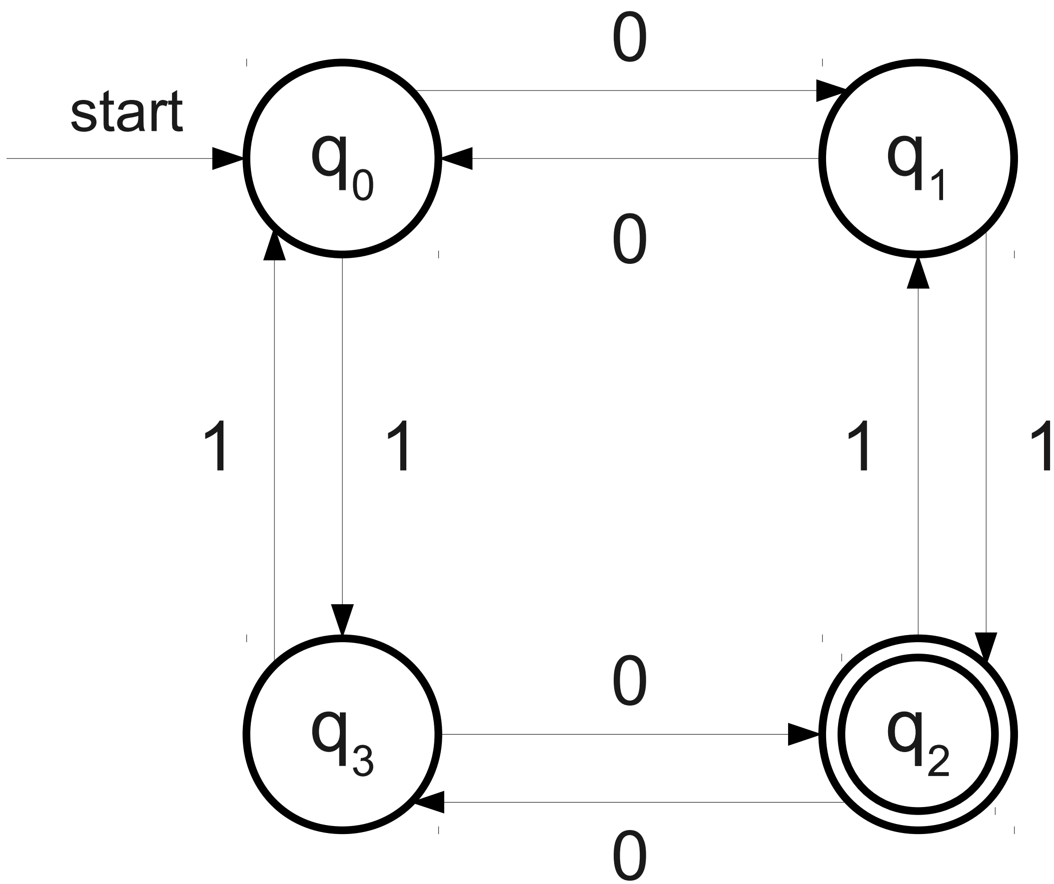
The double circle indicates that this state is an **accepting state**, so the automaton outputs "yes."

0 1 0 1 1 0

A Simple Finite Automaton

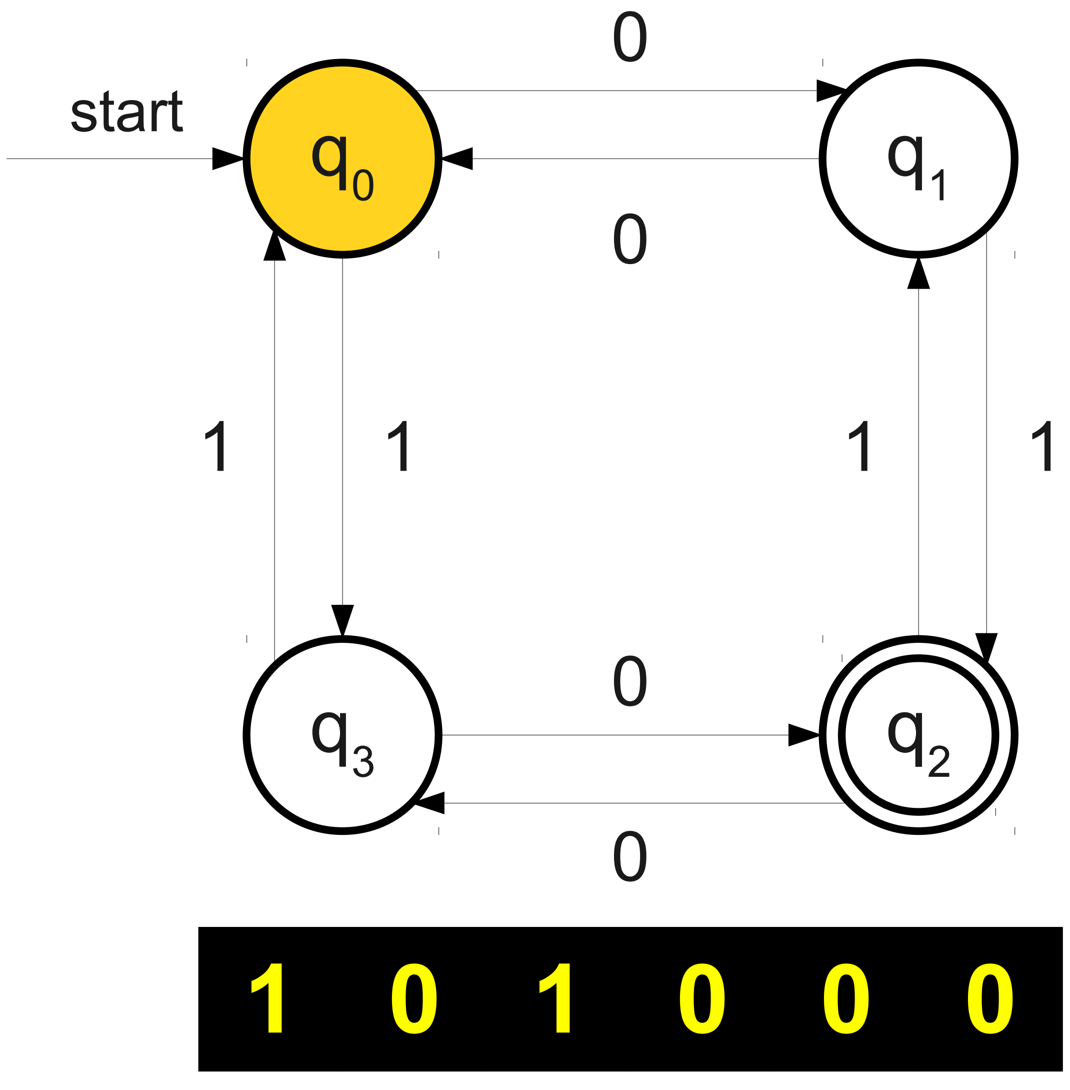


A Simple Finite Automaton

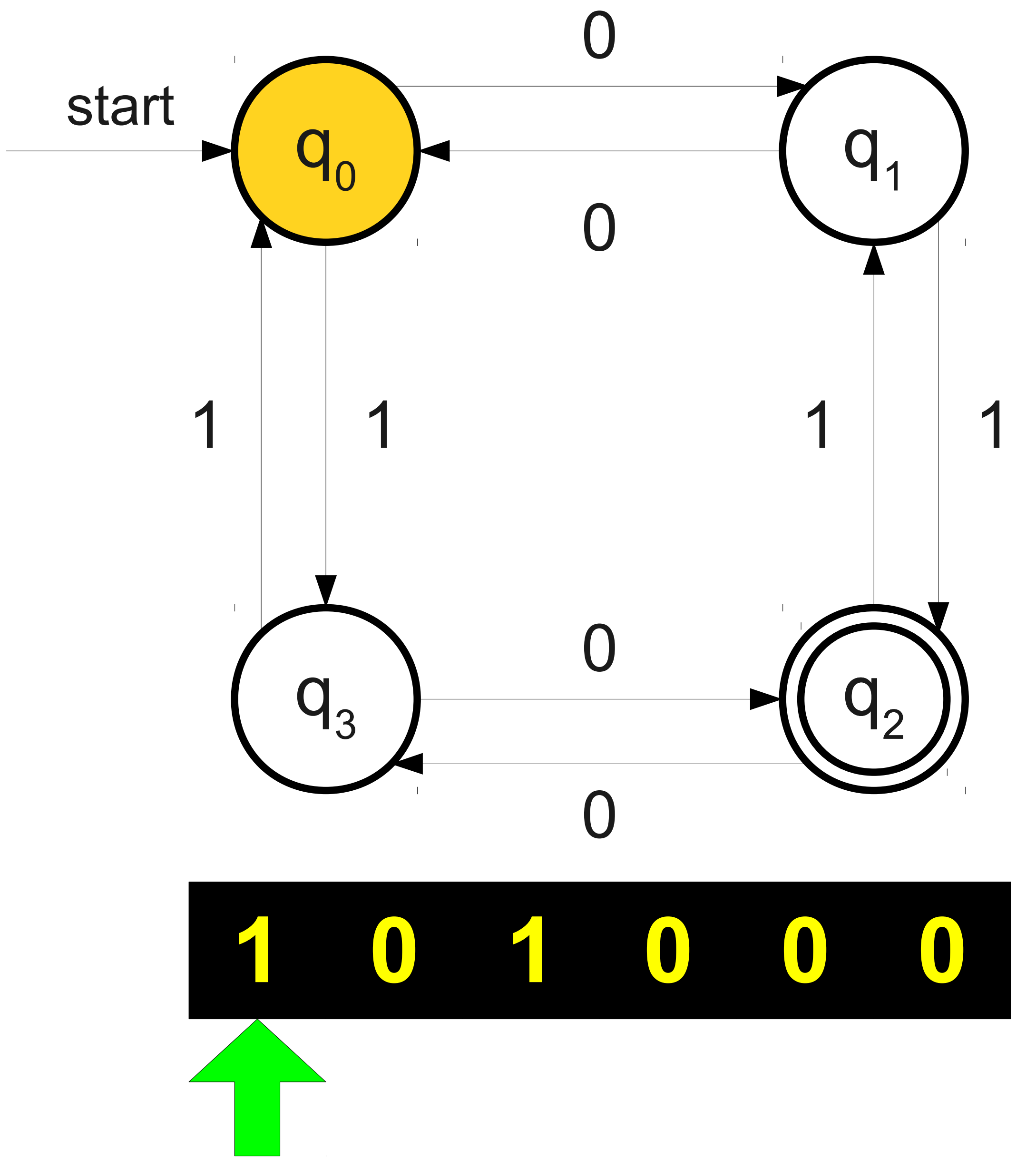


1	0	1	0	0	0

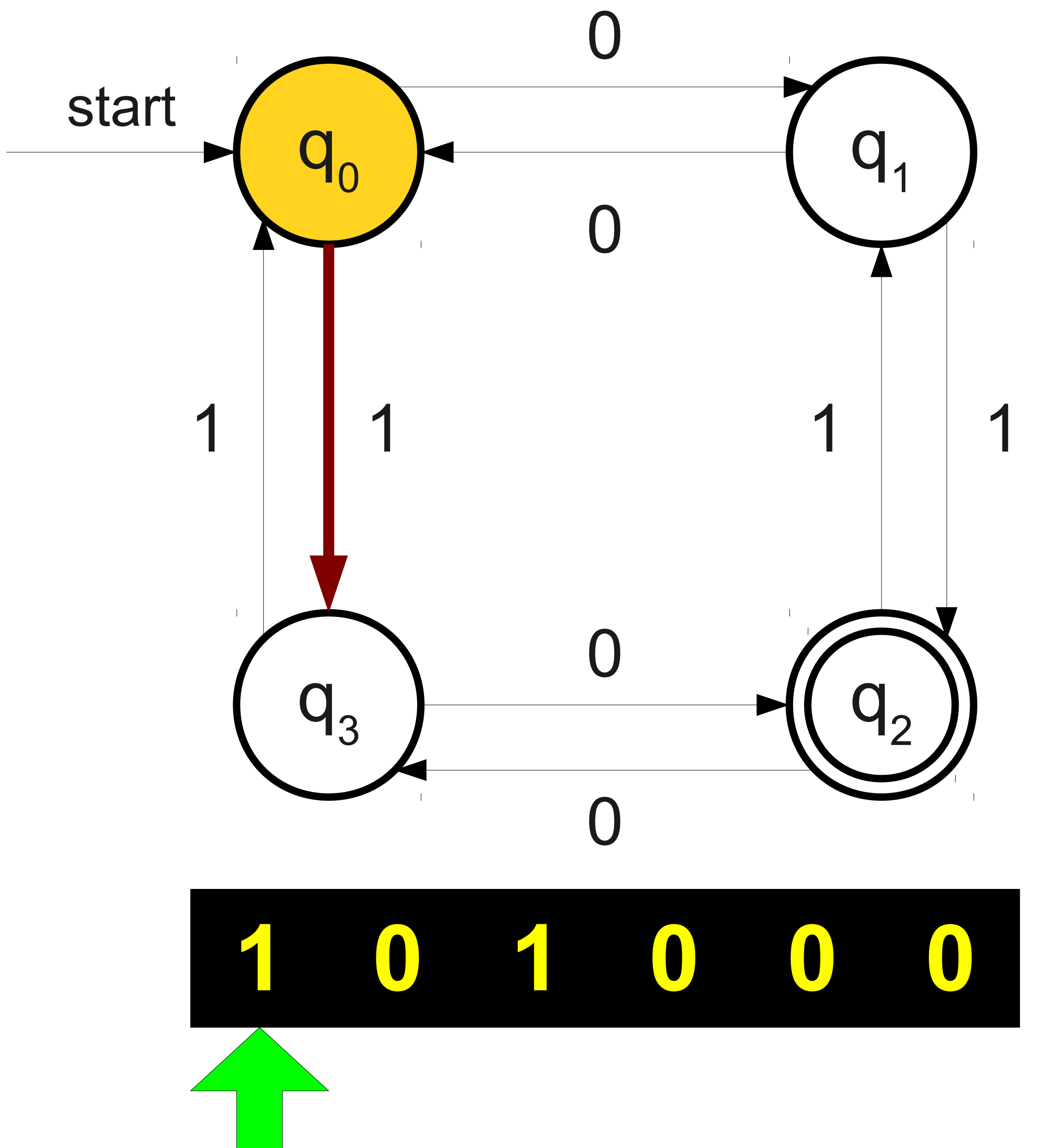
A Simple Finite Automaton



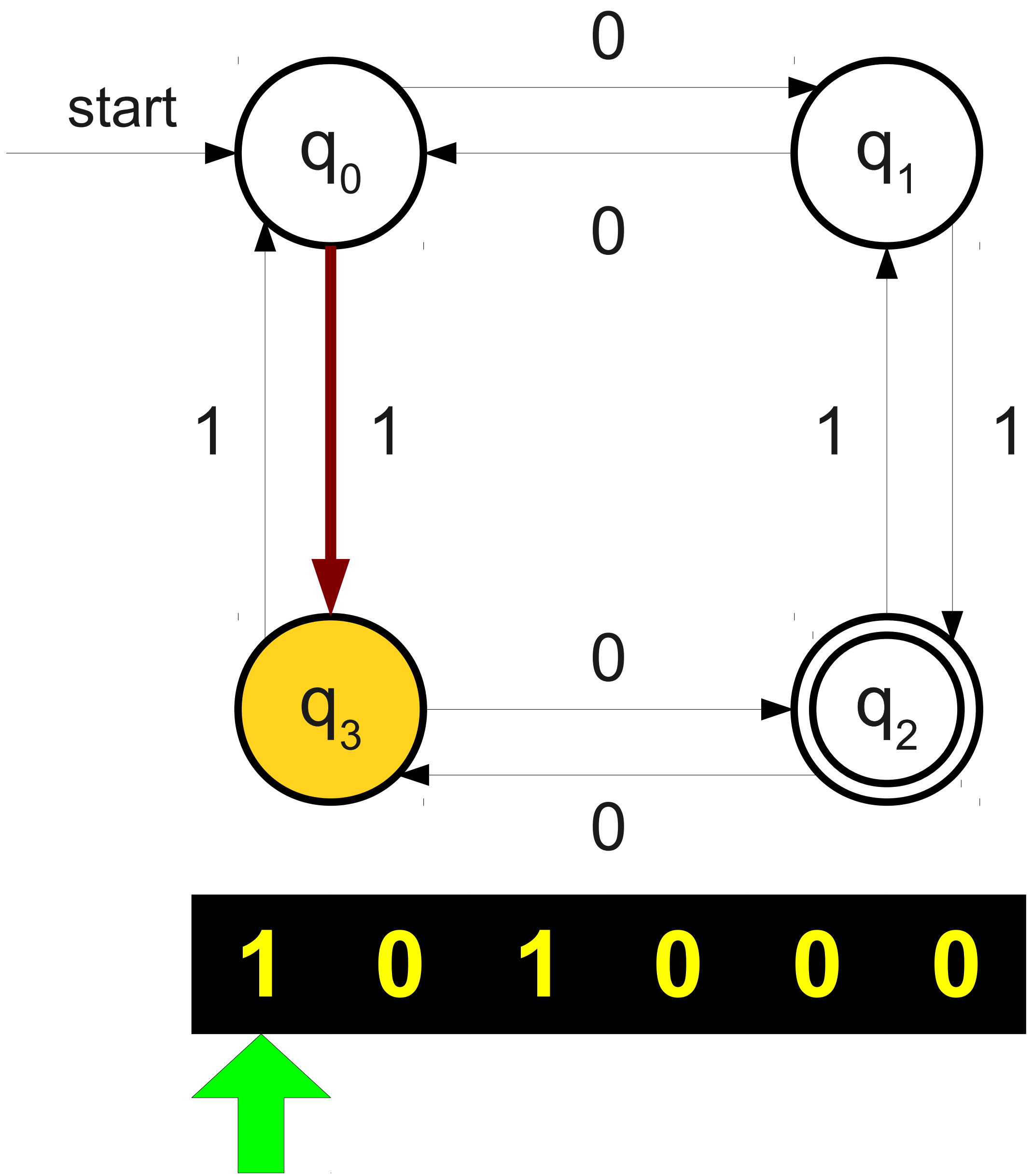
A Simple Finite Automaton



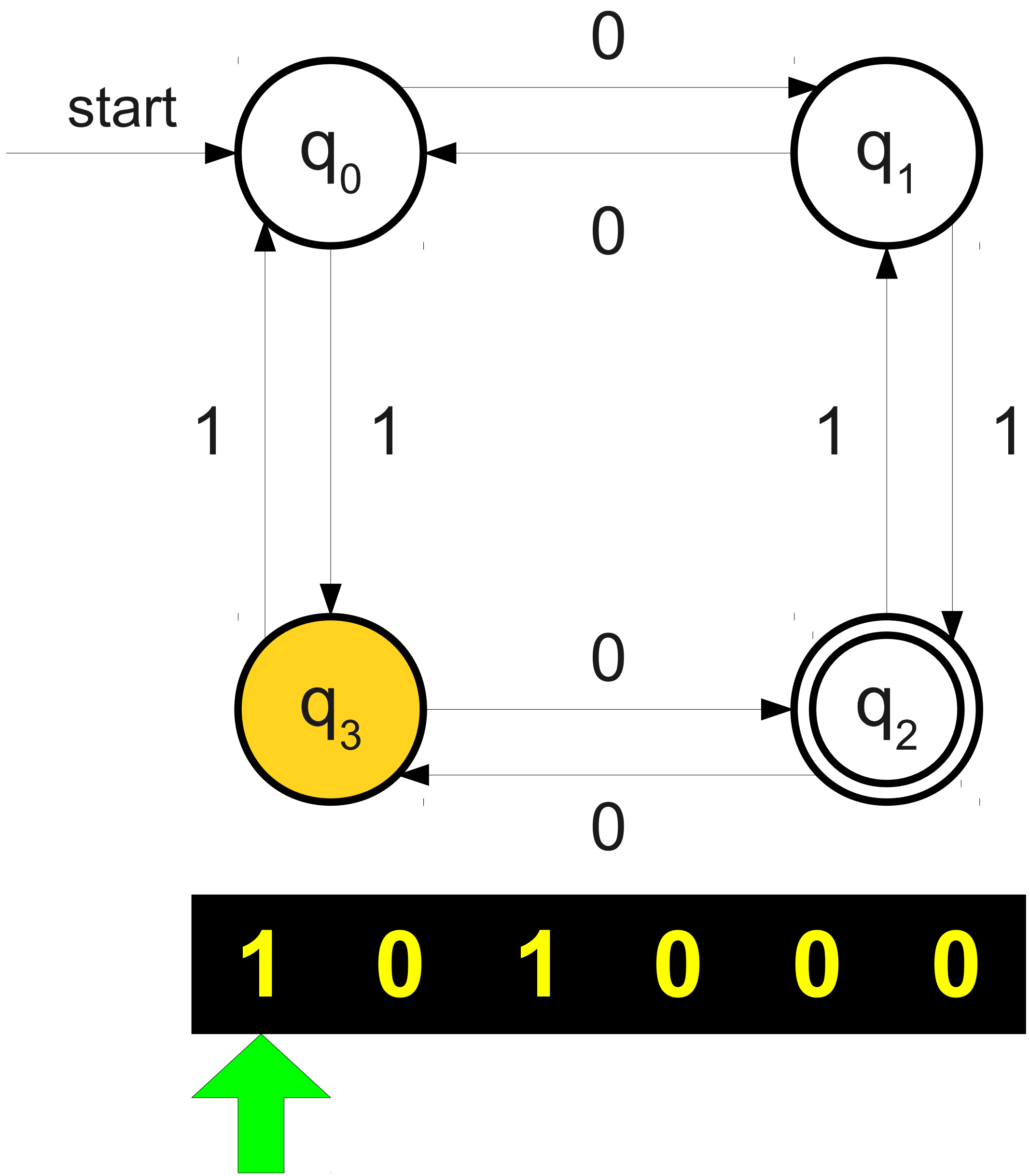
A Simple Finite Automaton



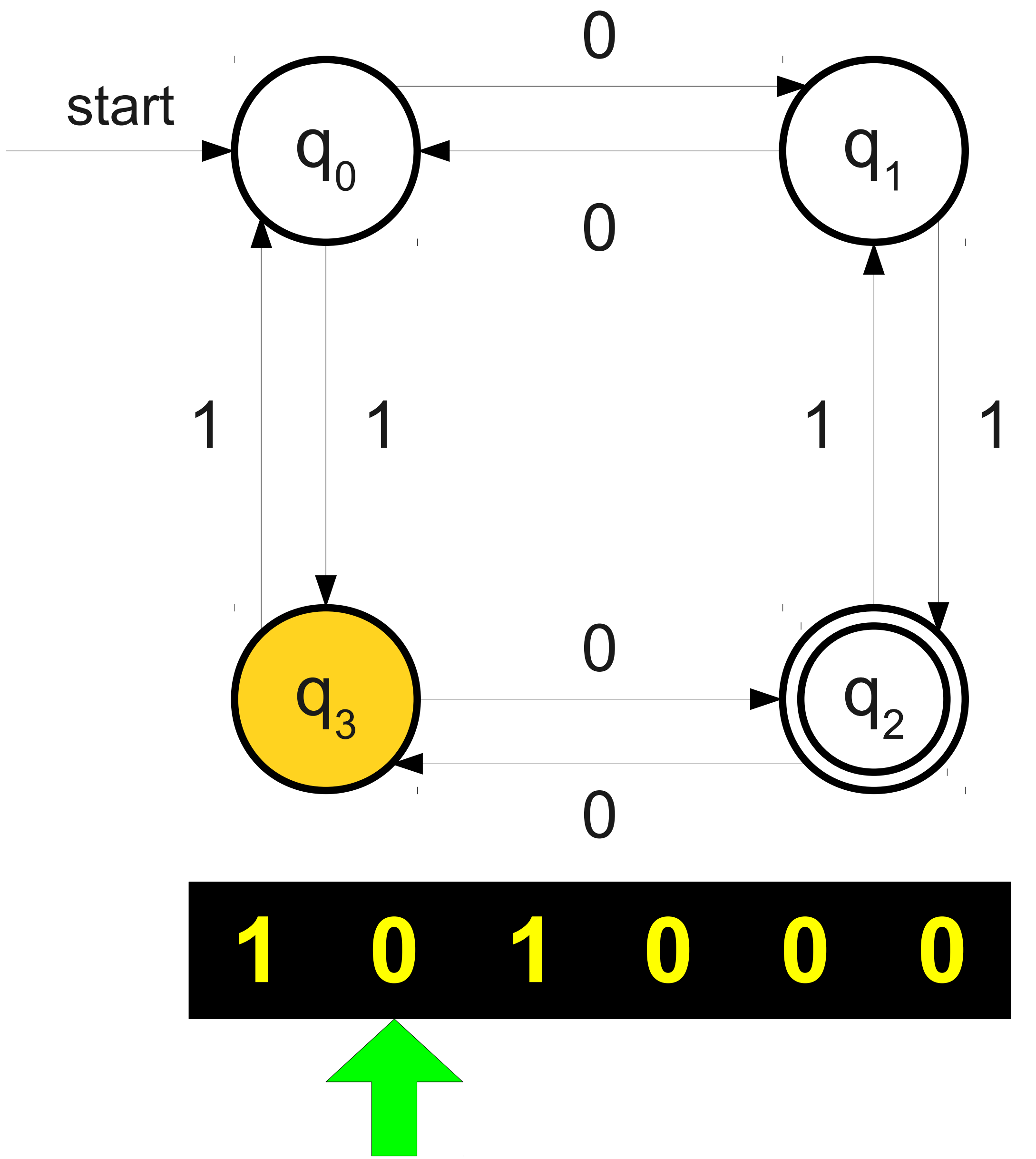
A Simple Finite Automaton



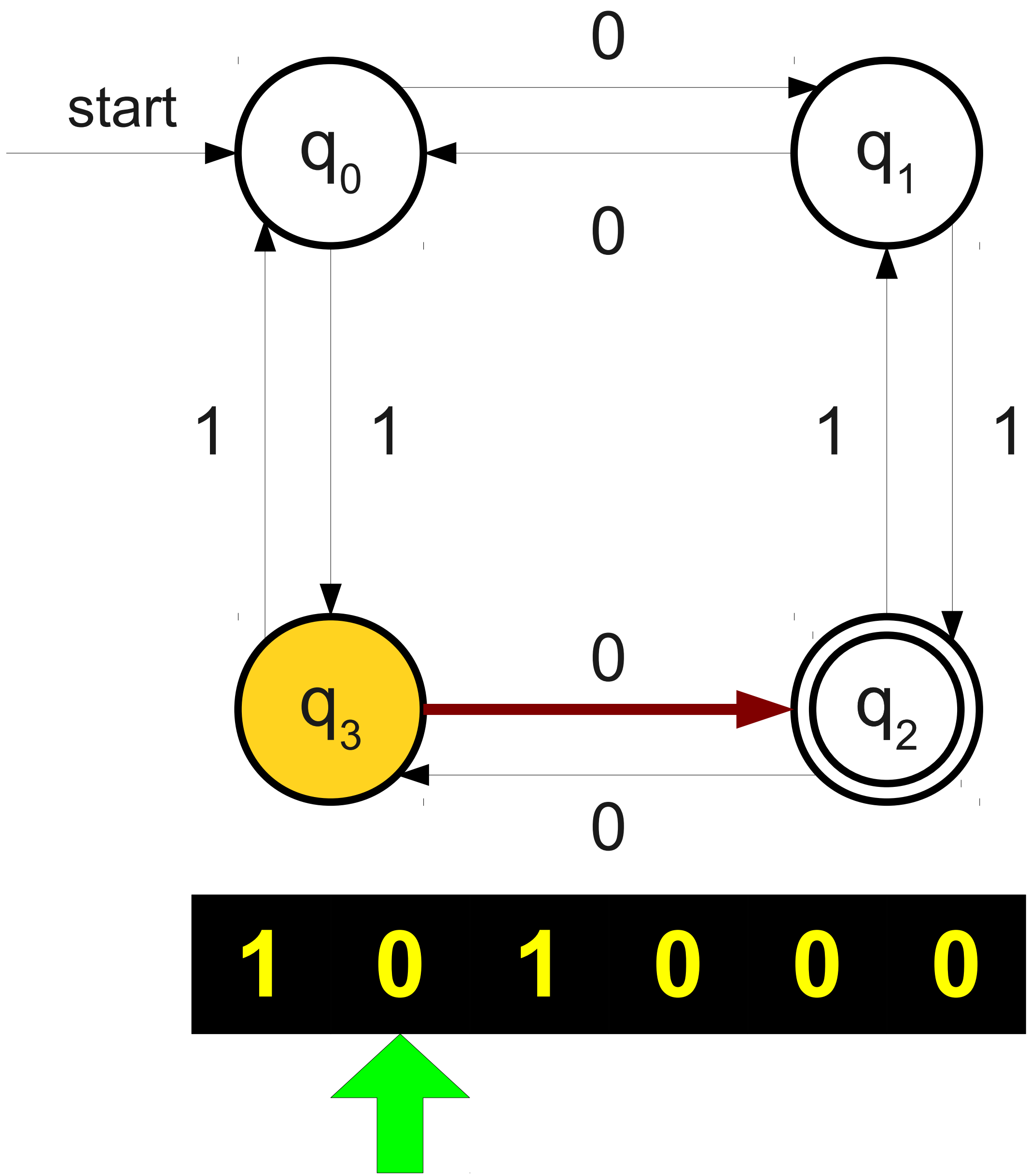
A Simple Finite Automaton



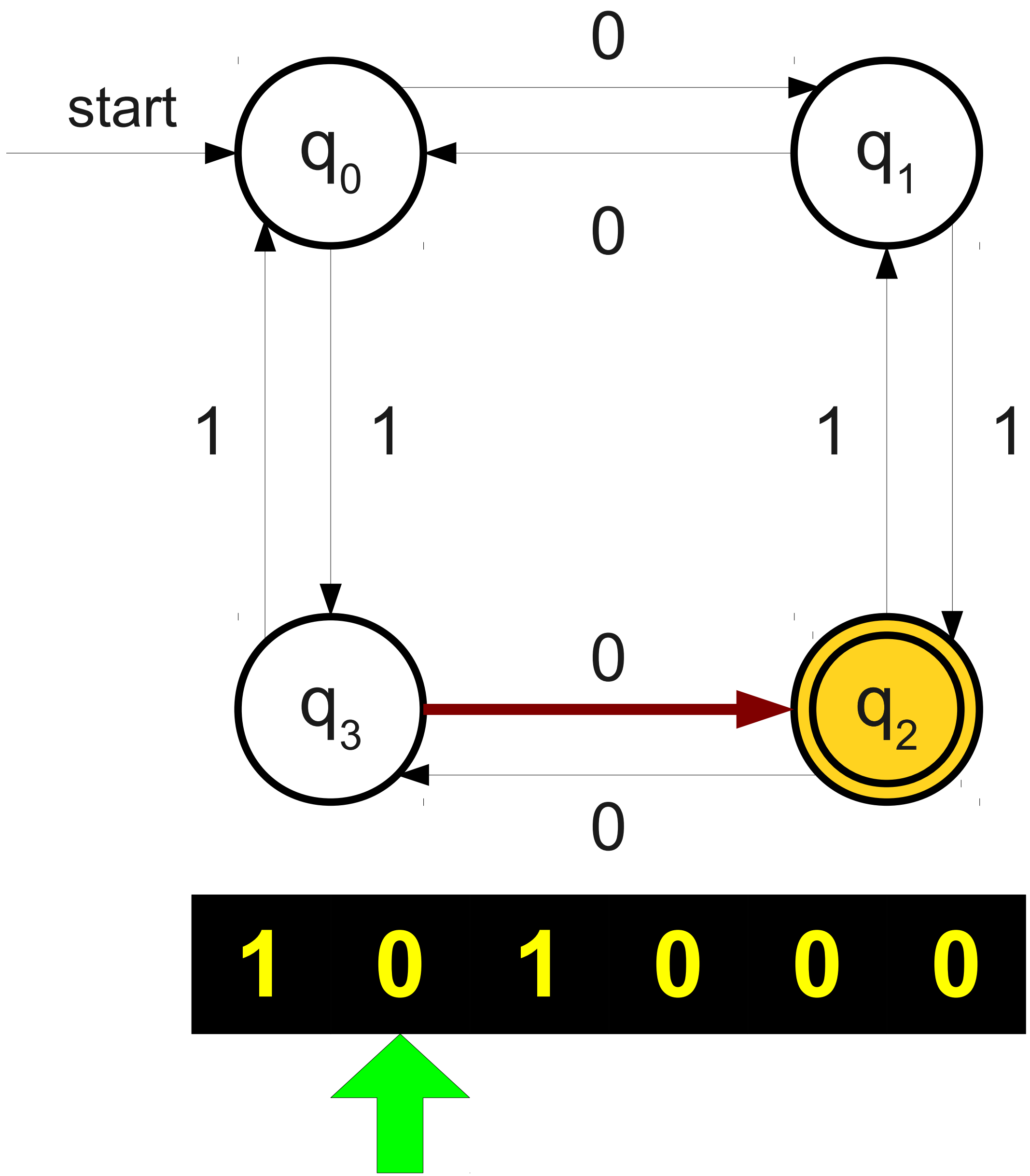
A Simple Finite Automaton



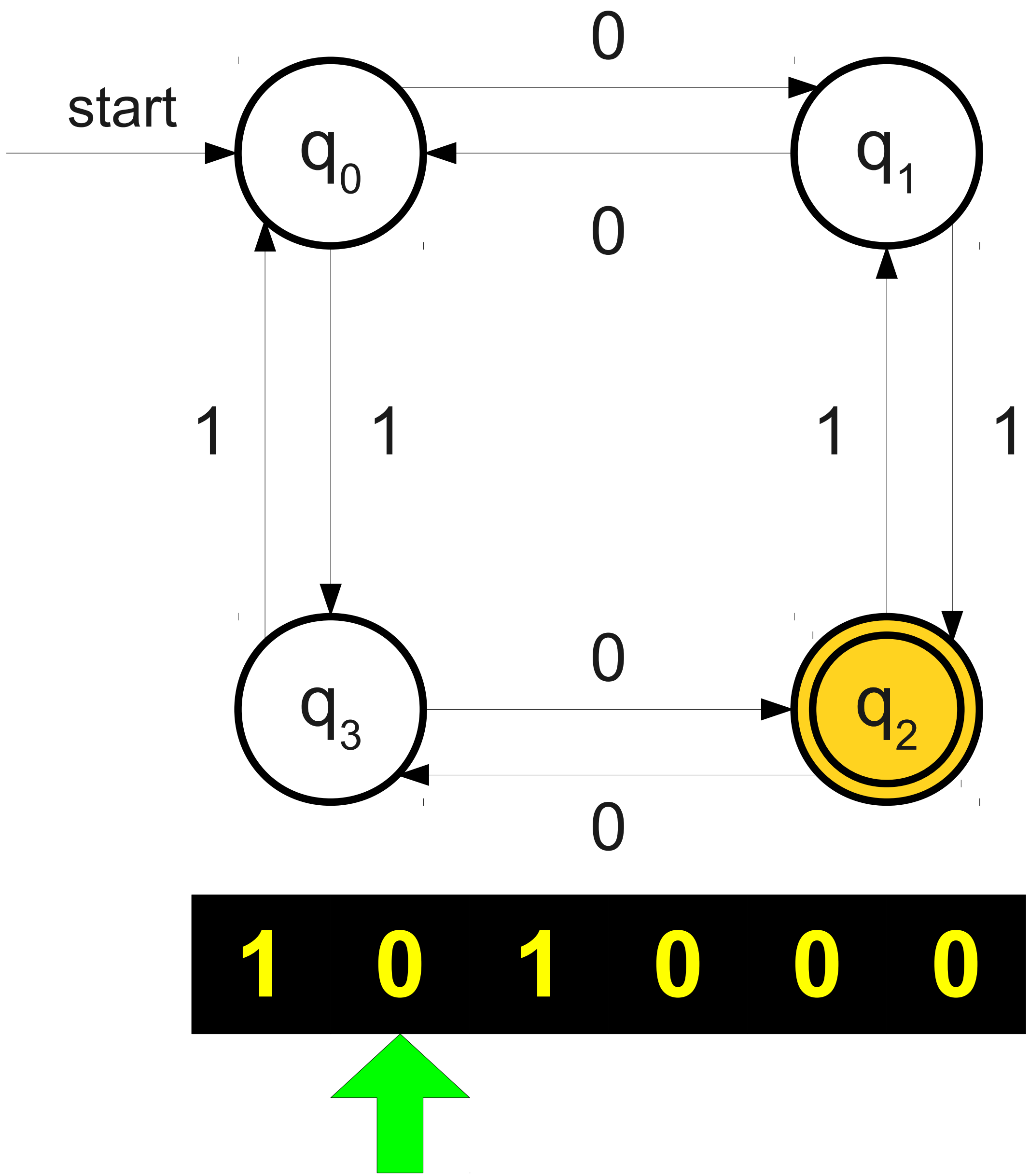
A Simple Finite Automaton



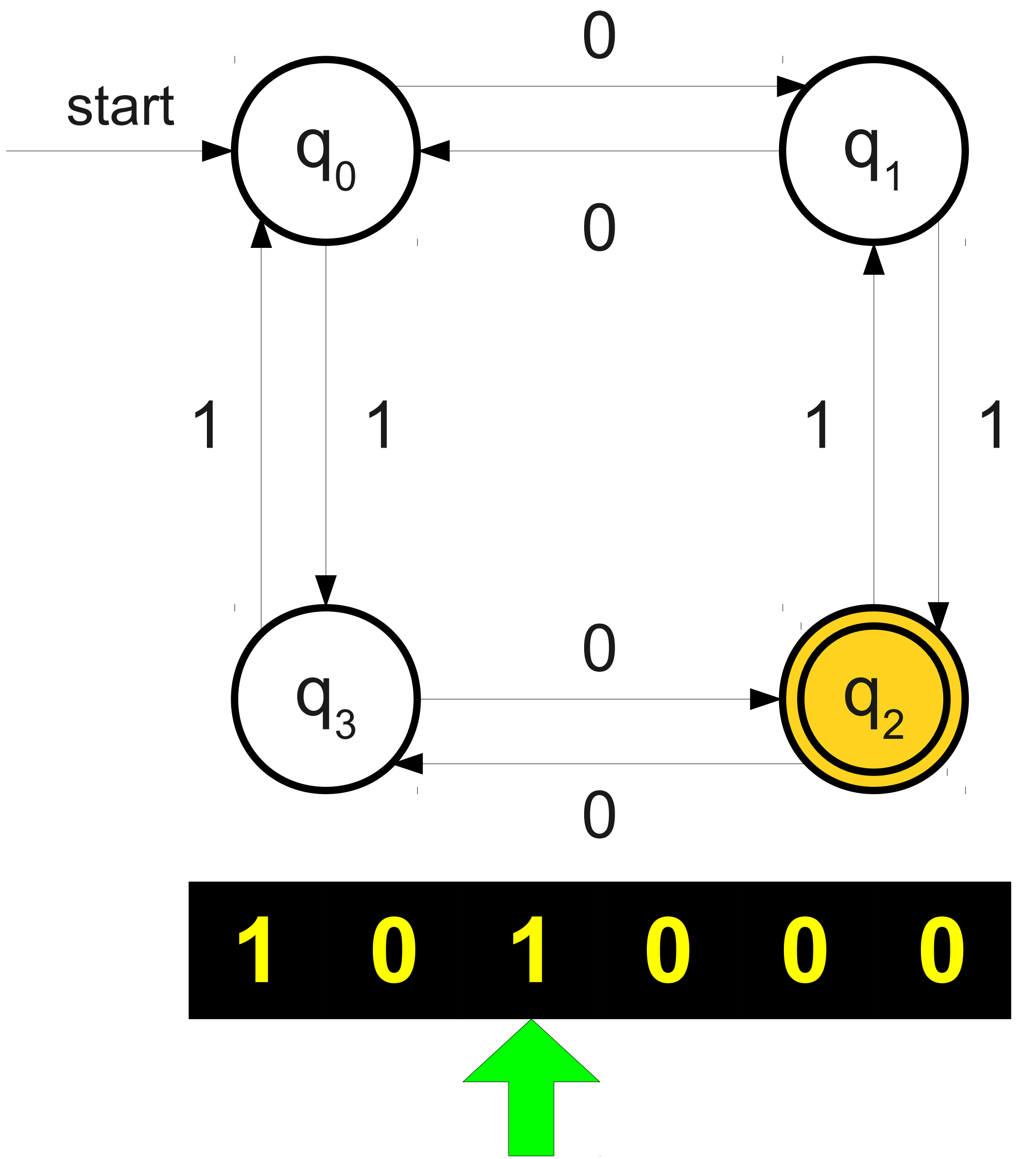
A Simple Finite Automaton



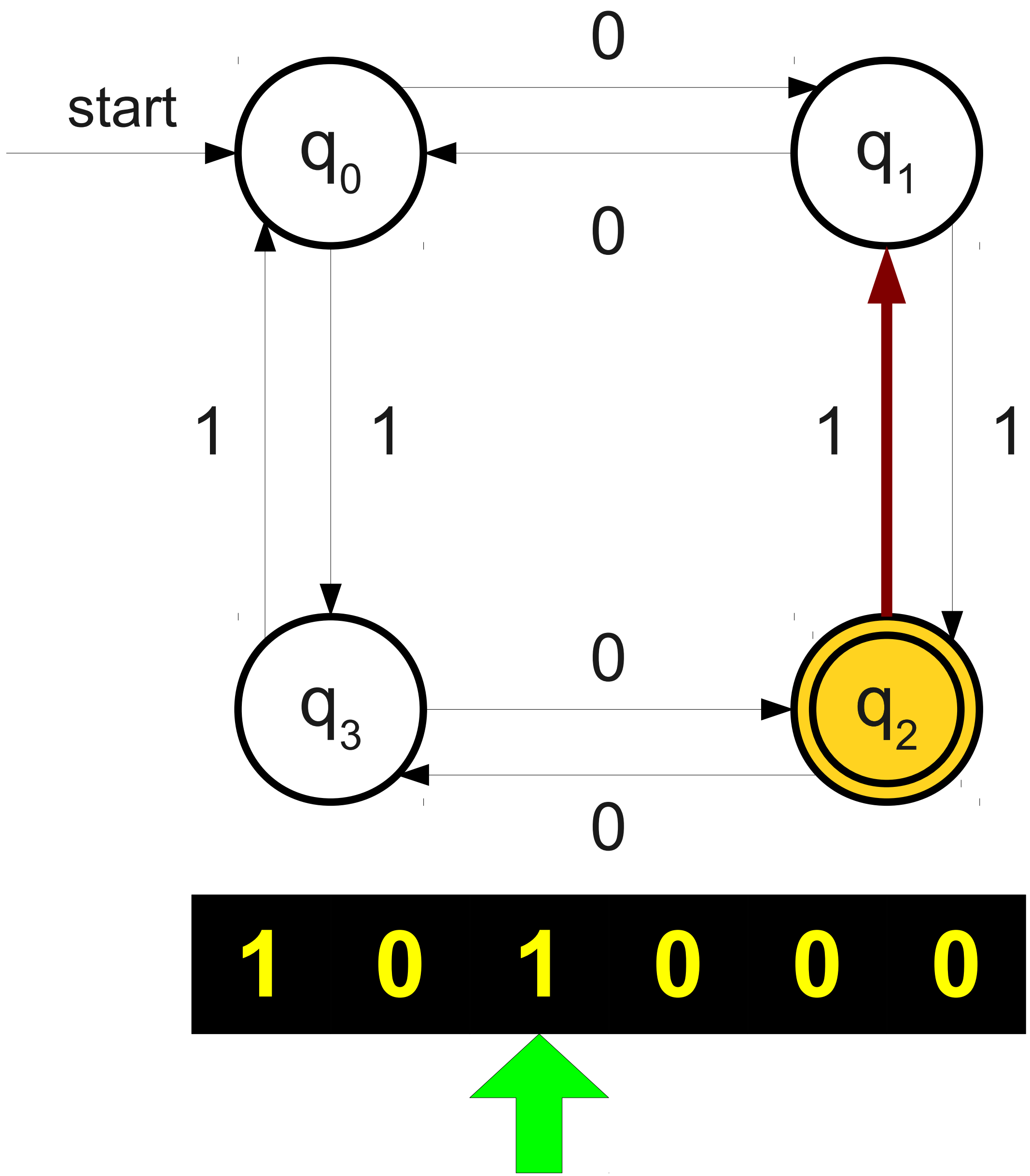
A Simple Finite Automaton



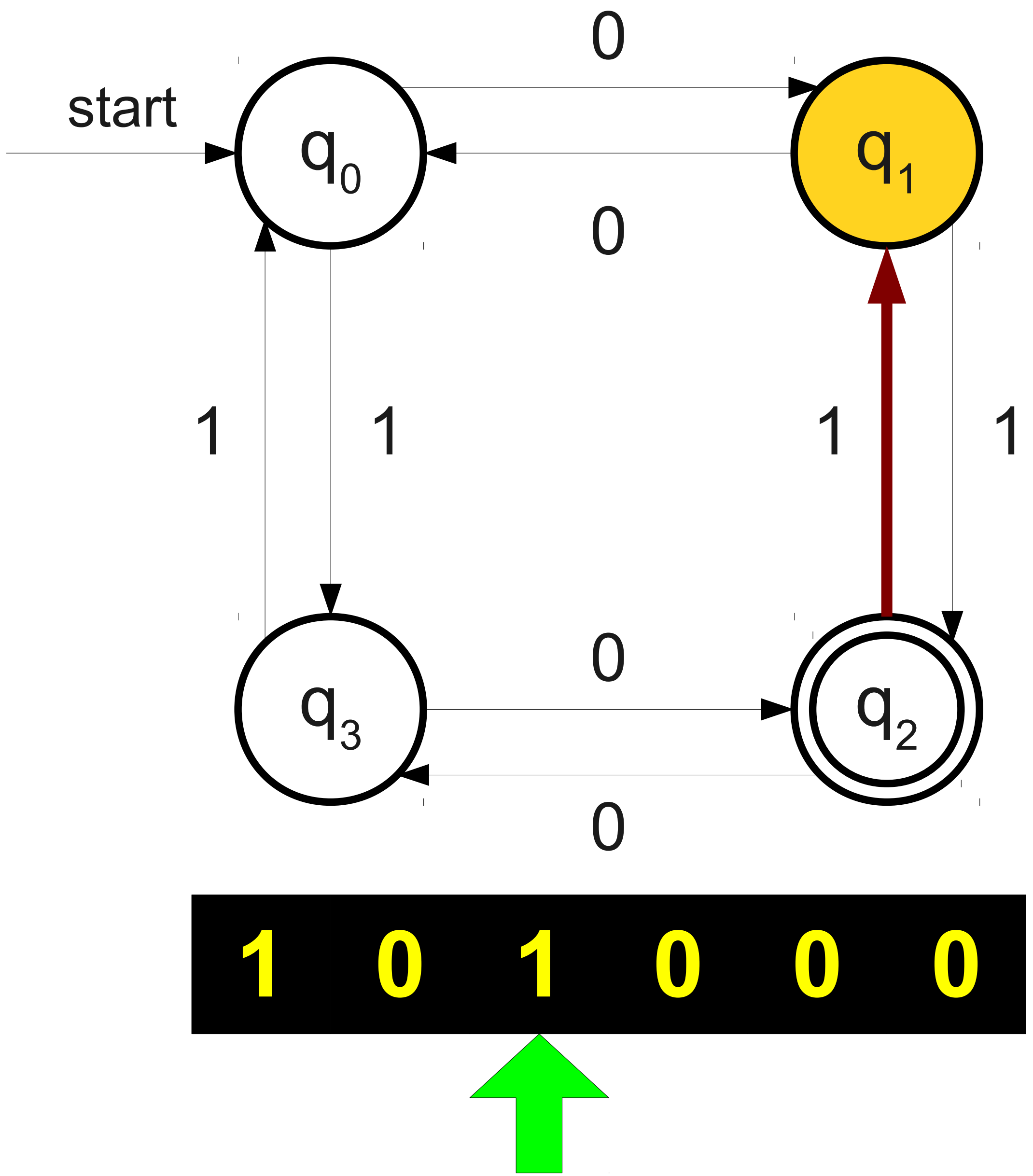
A Simple Finite Automaton



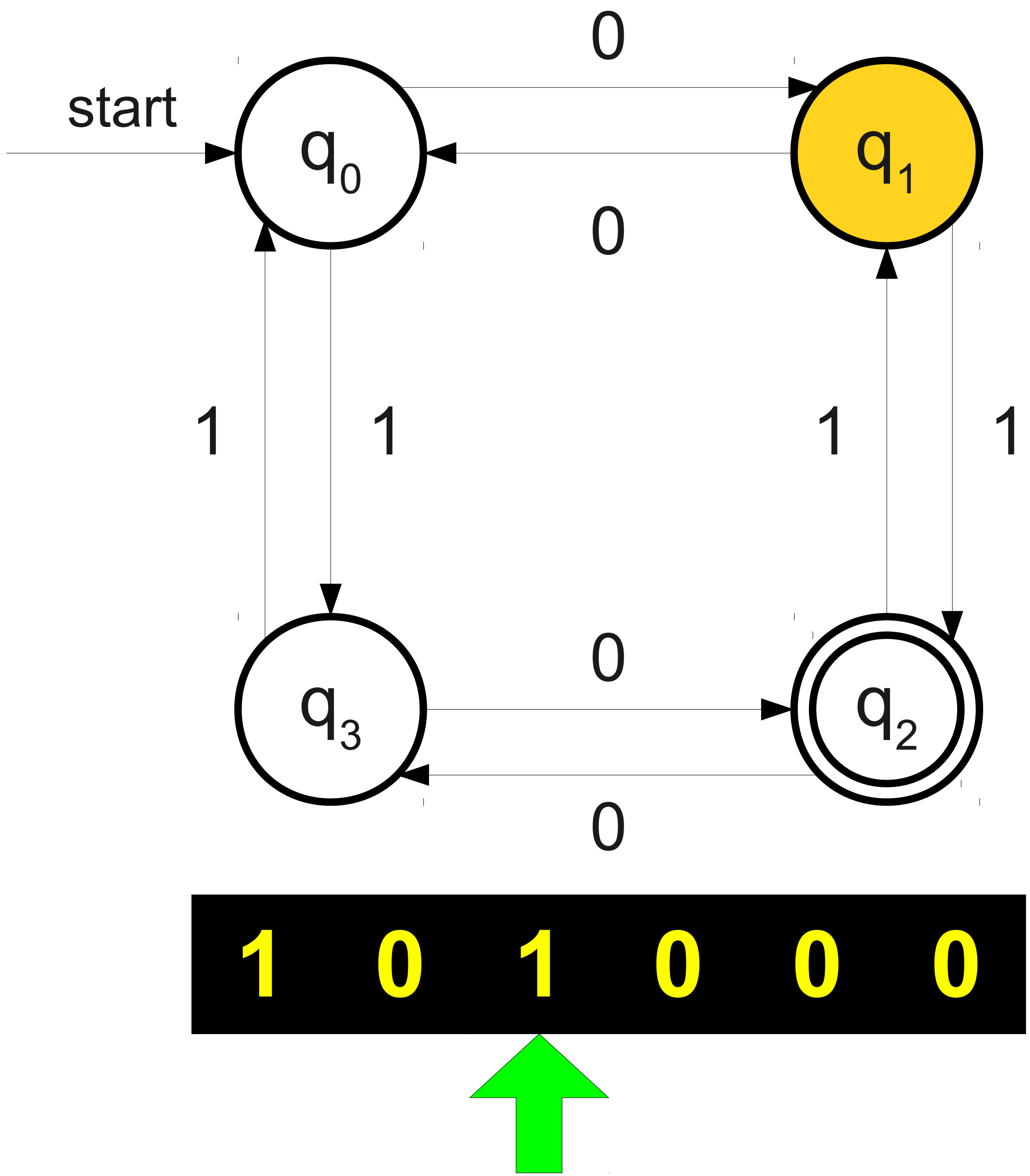
A Simple Finite Automaton



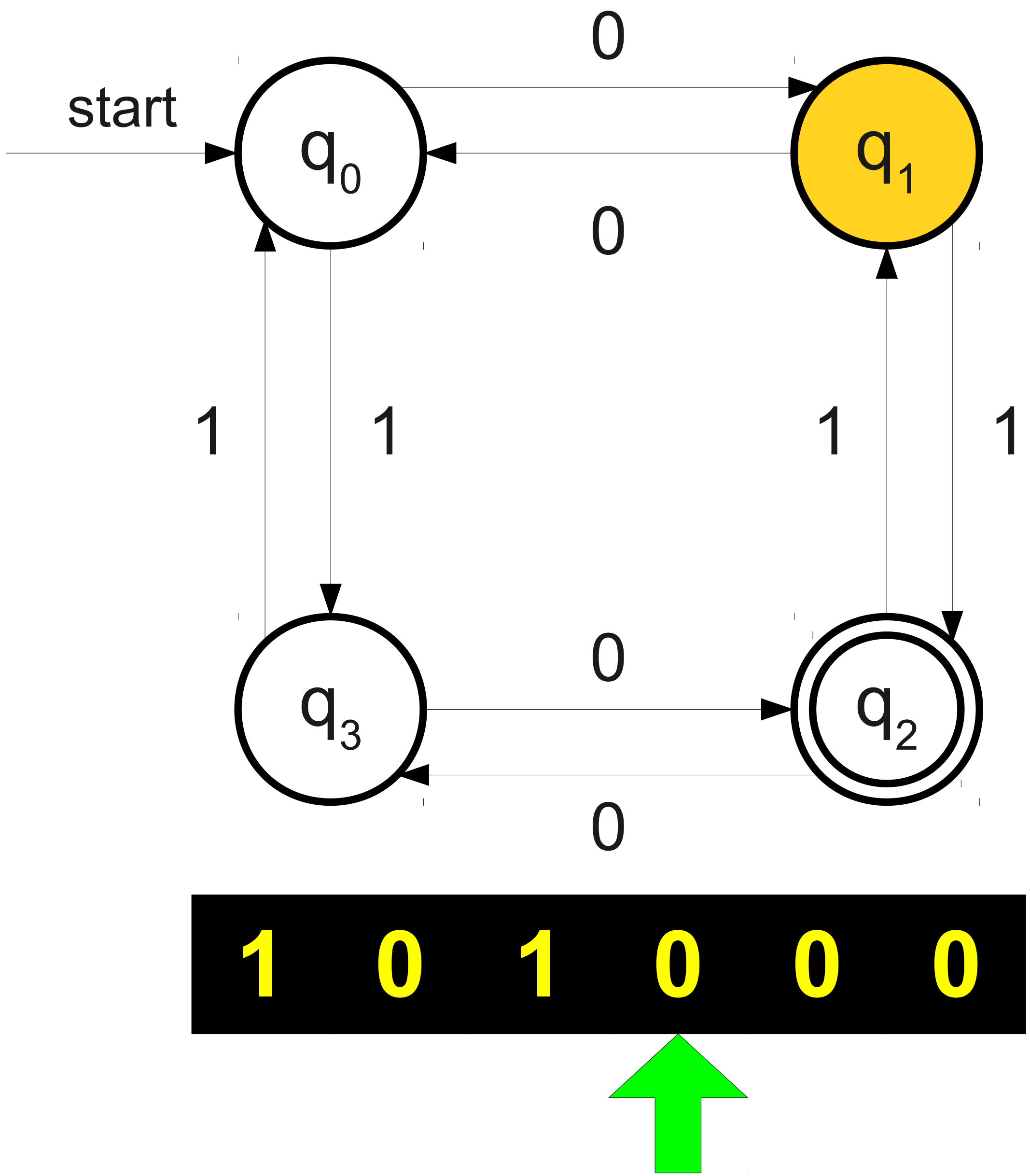
A Simple Finite Automaton



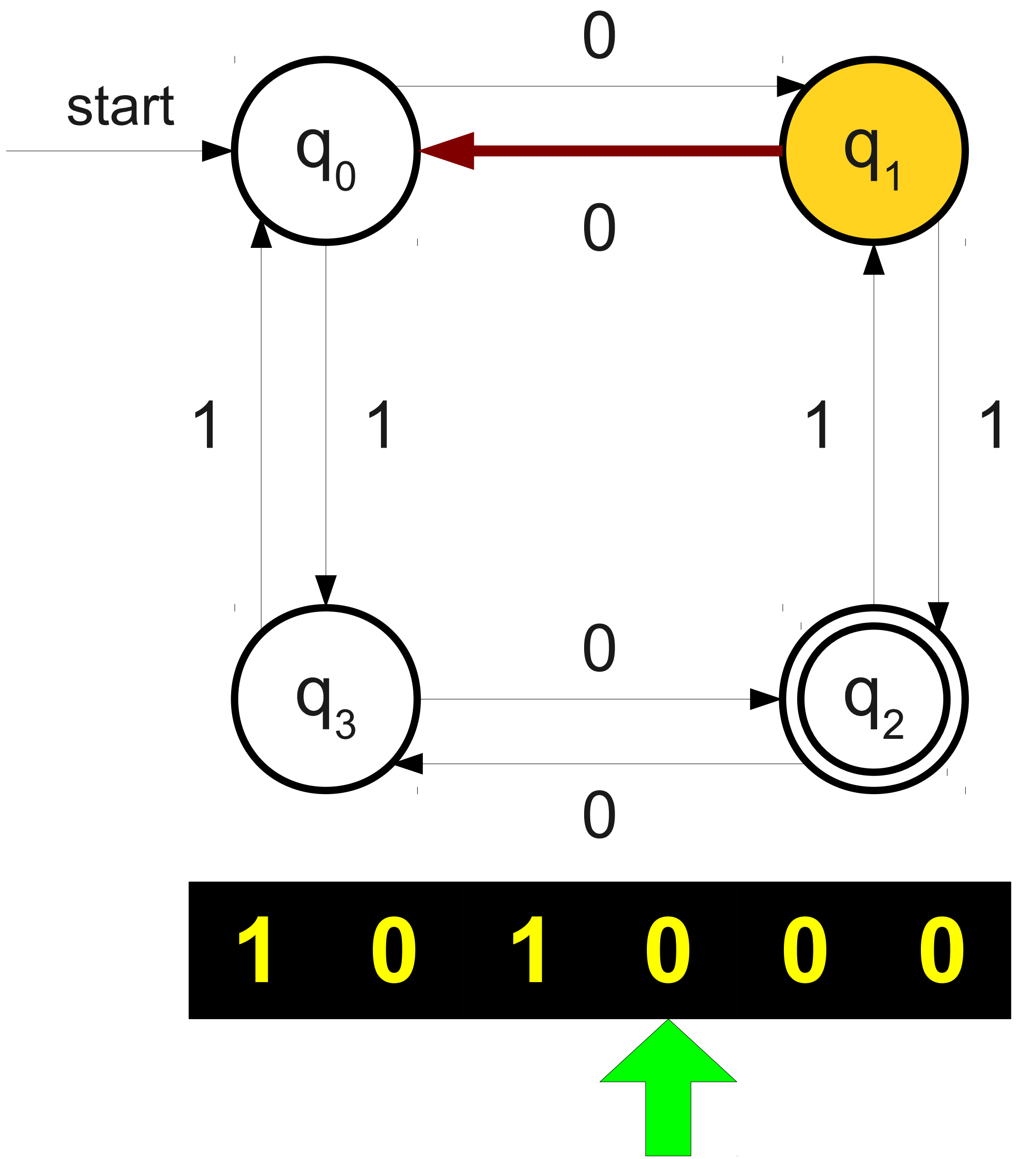
A Simple Finite Automaton



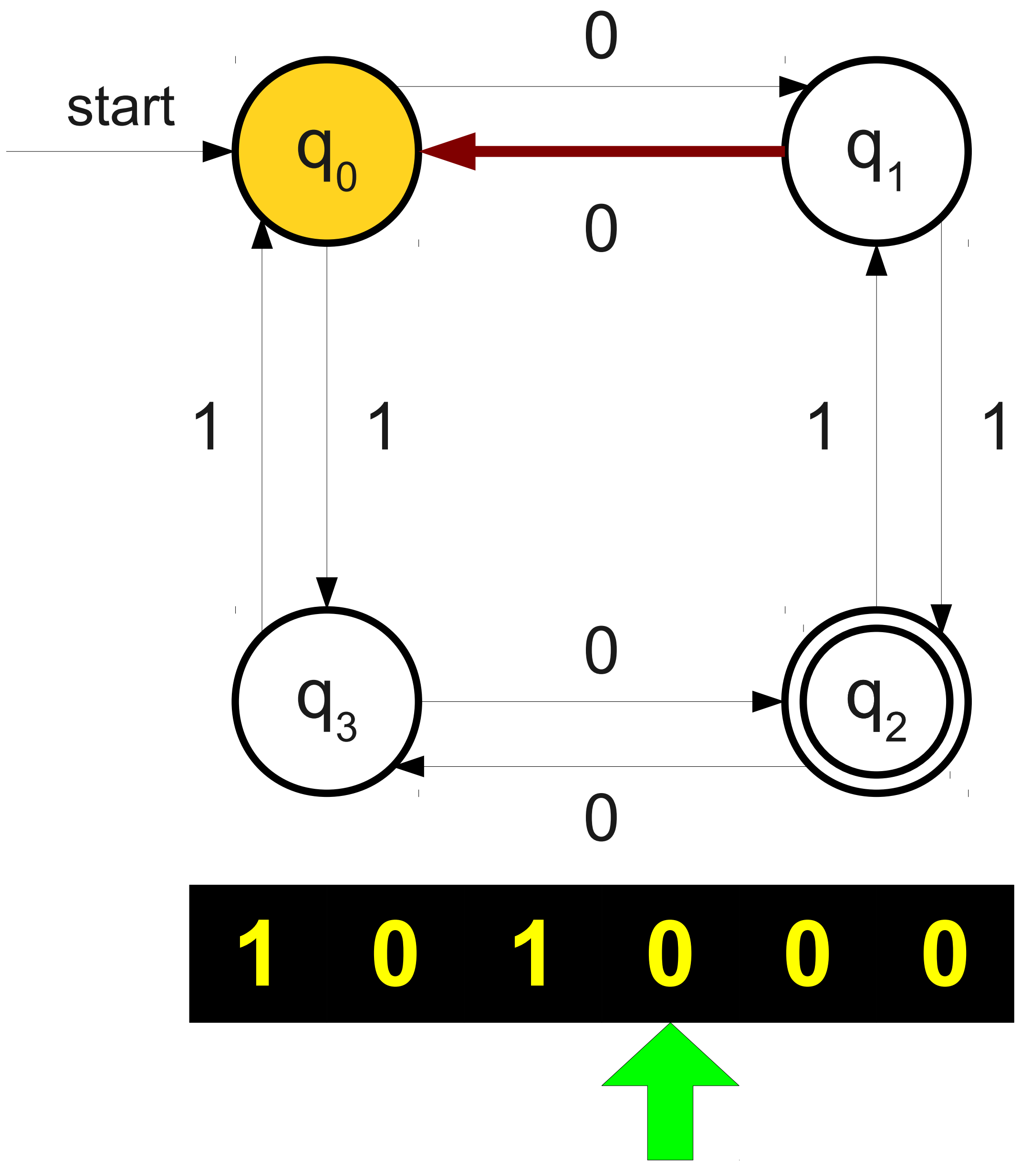
A Simple Finite Automaton



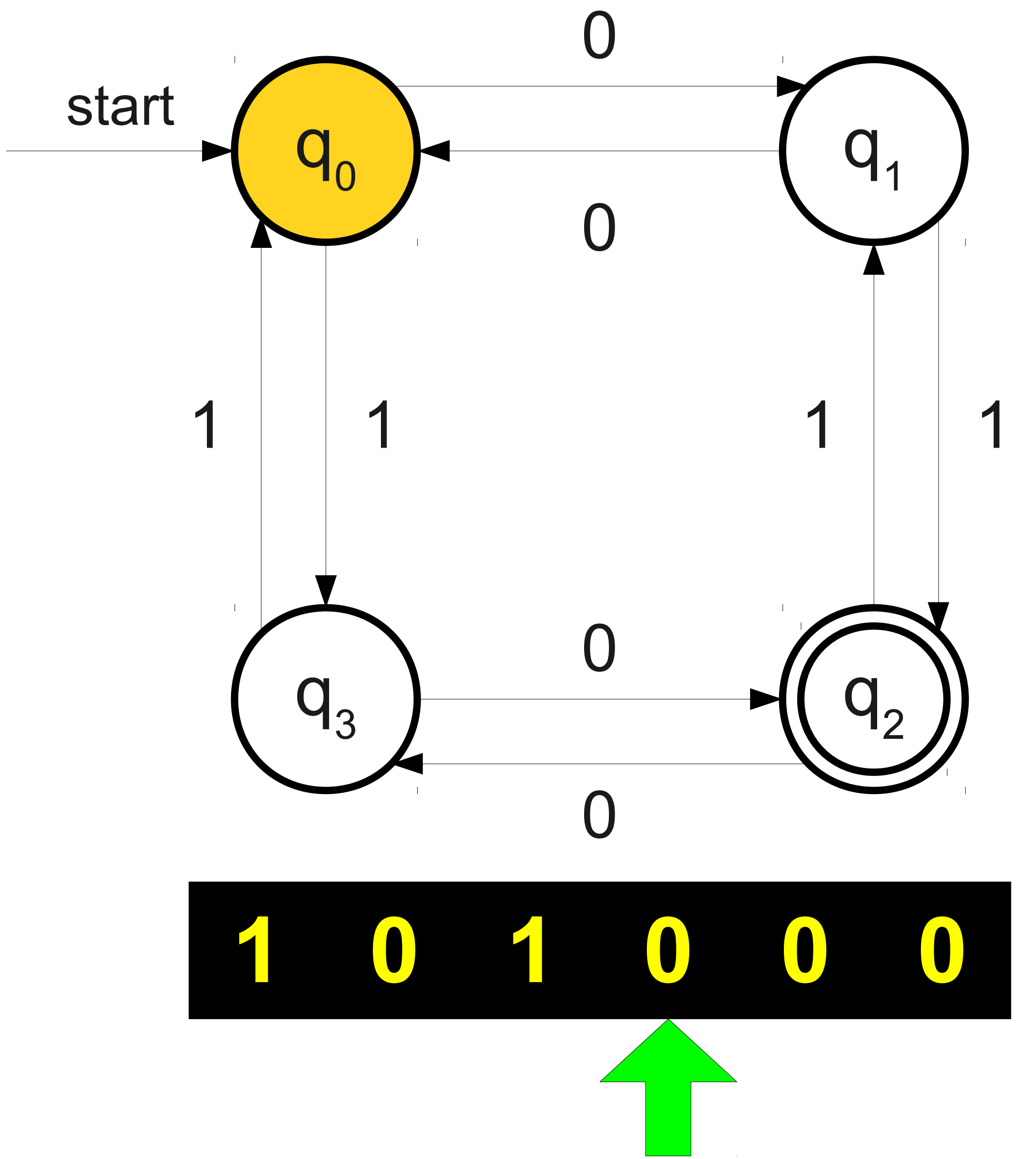
A Simple Finite Automaton



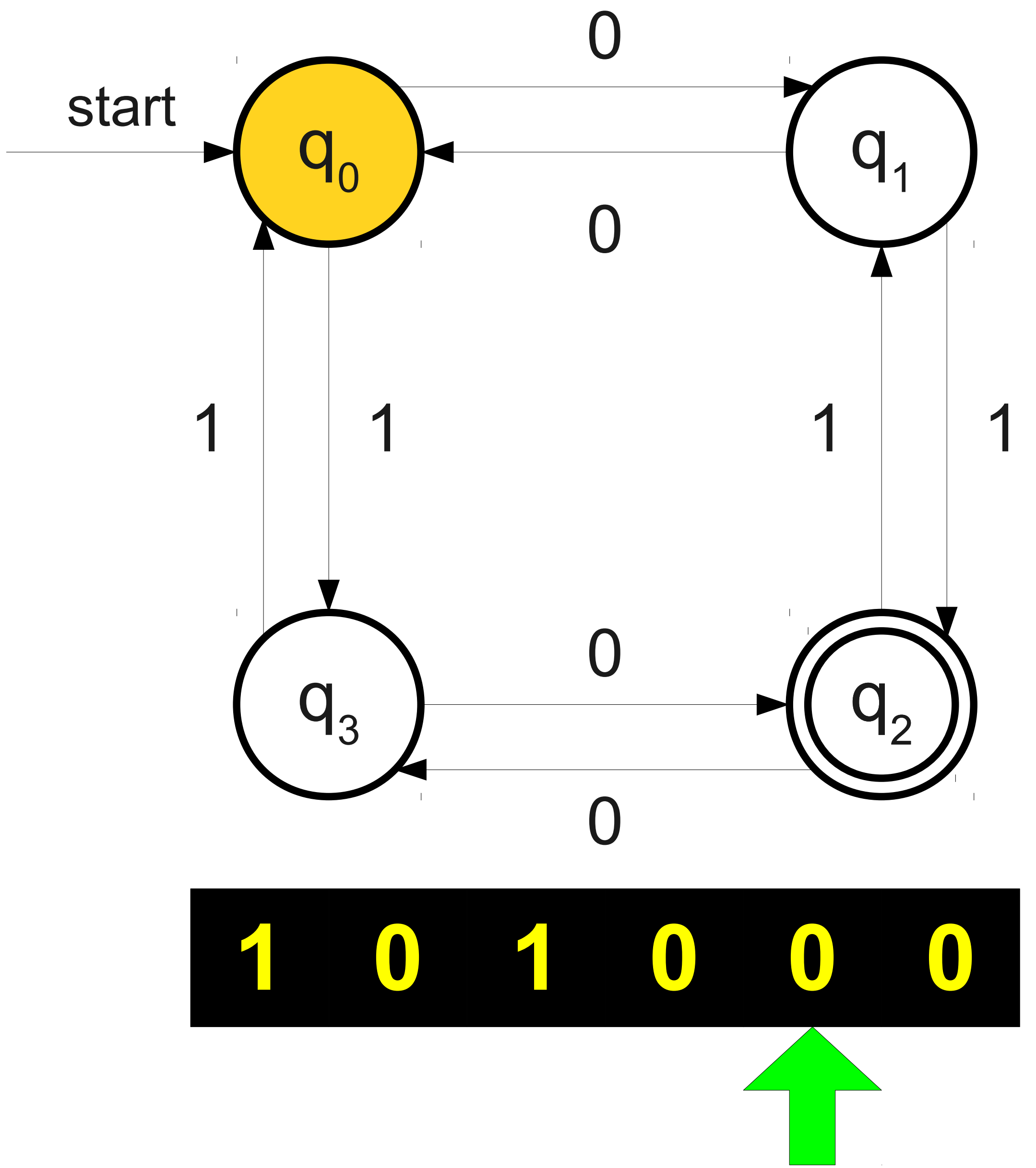
A Simple Finite Automaton



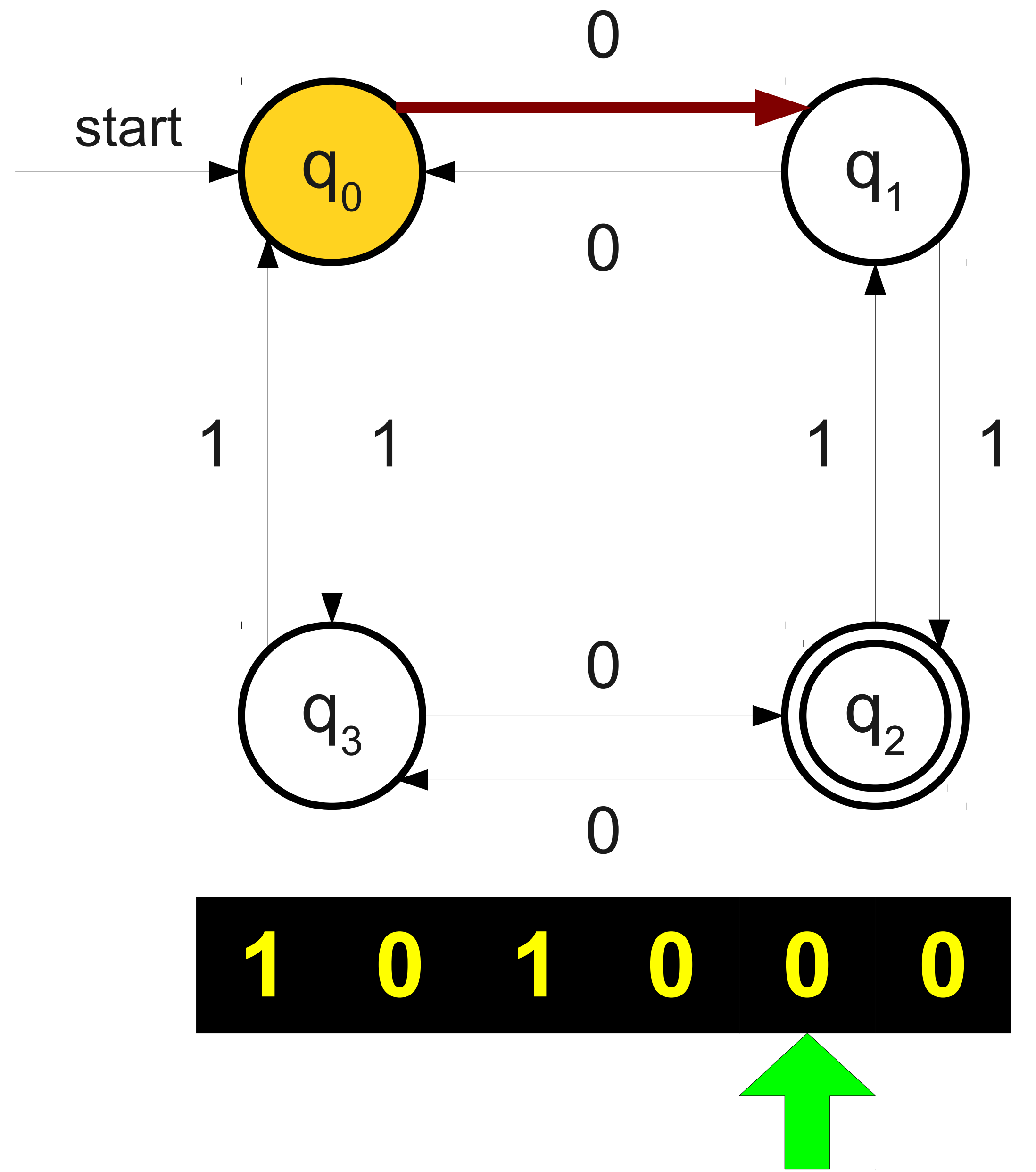
A Simple Finite Automaton



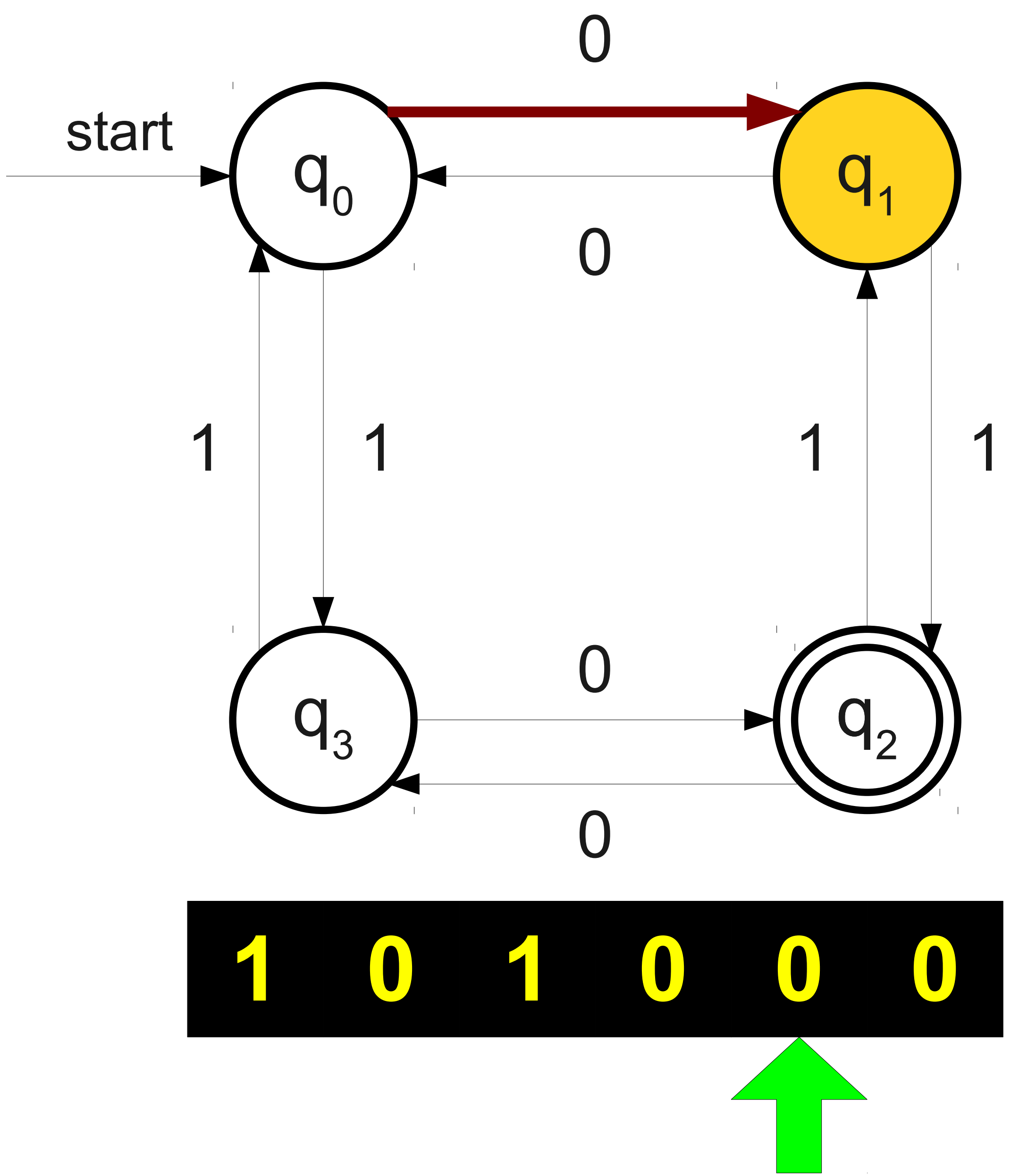
A Simple Finite Automaton



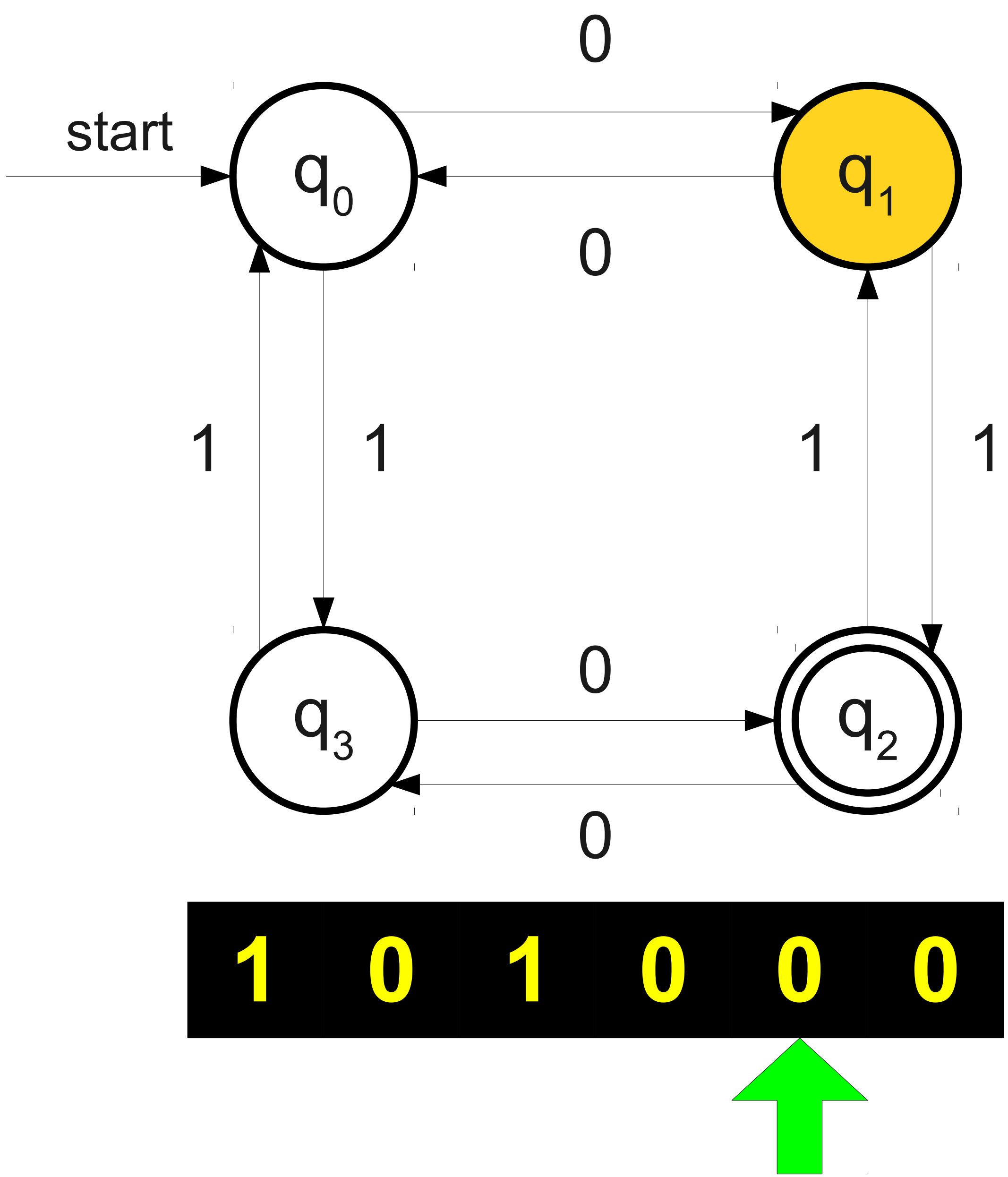
A Simple Finite Automaton



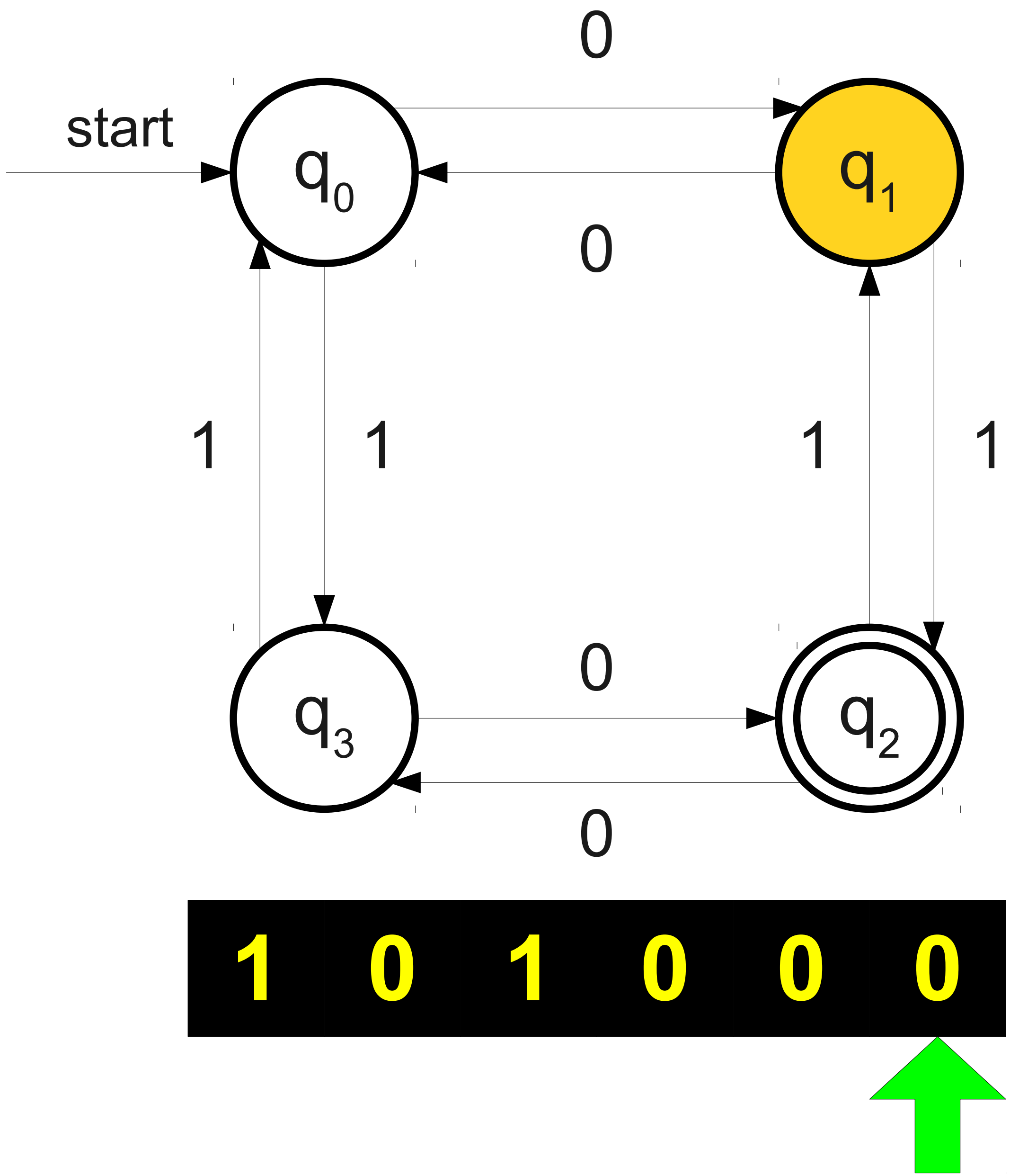
A Simple Finite Automaton



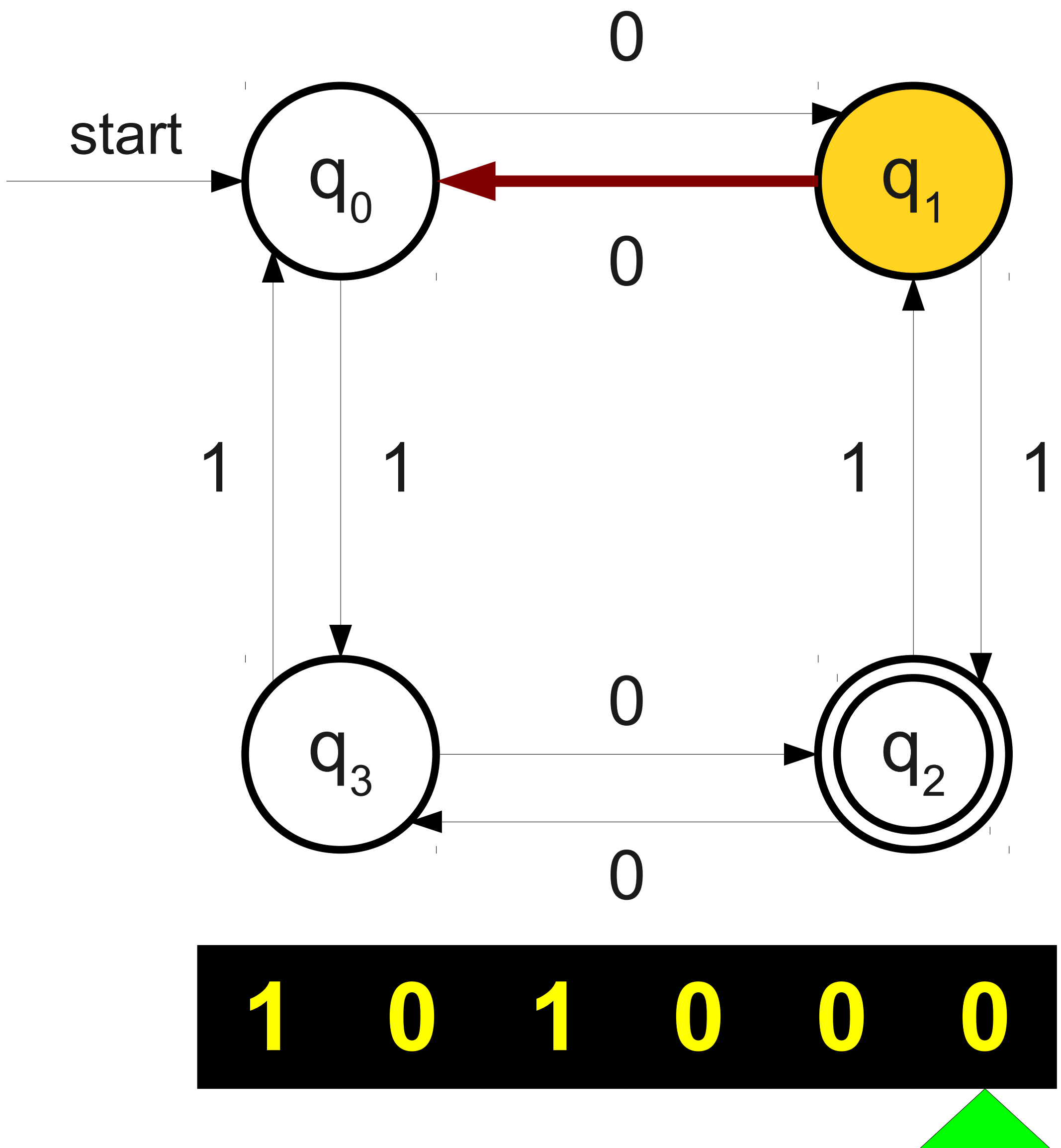
A Simple Finite Automaton



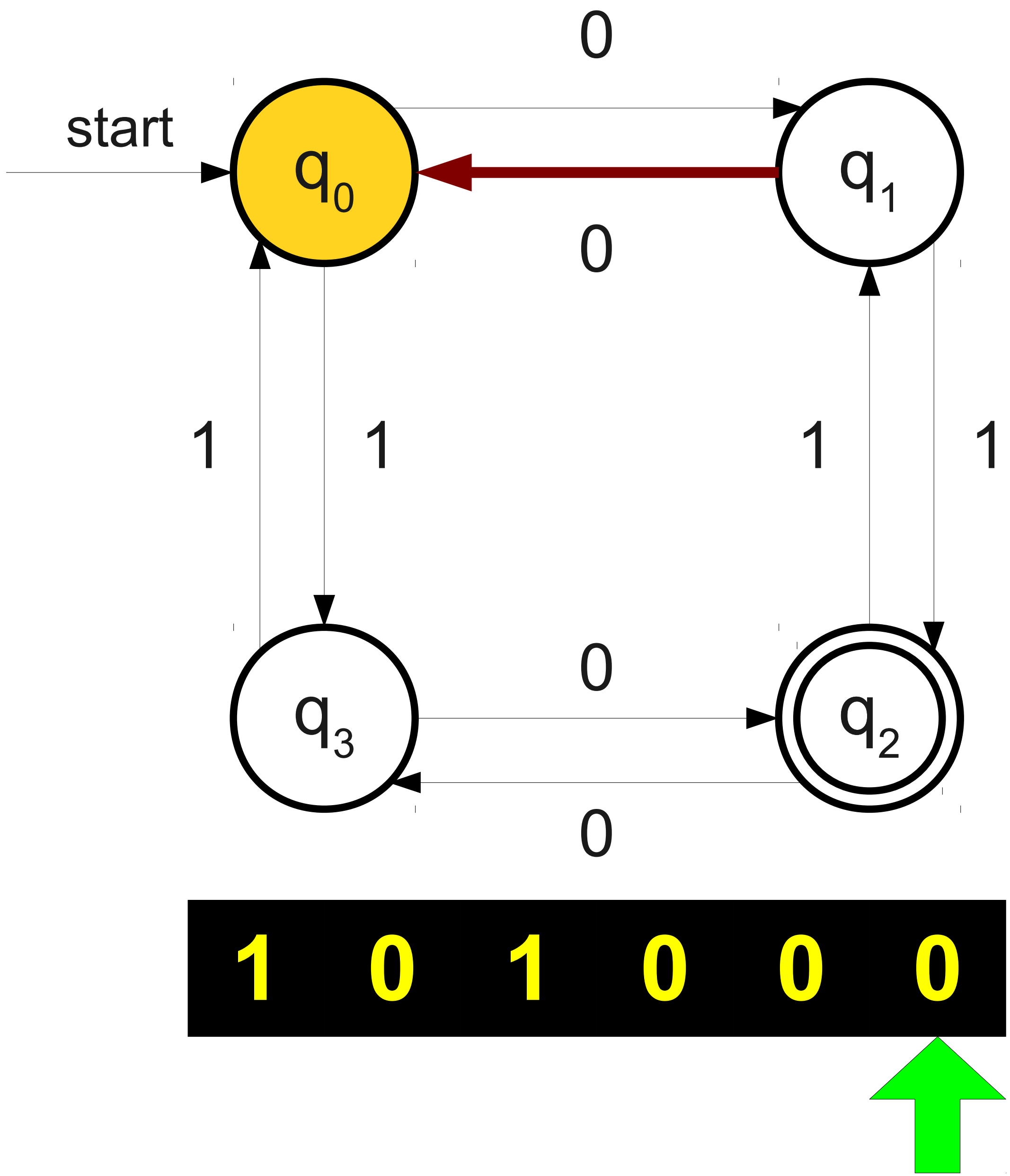
A Simple Finite Automaton



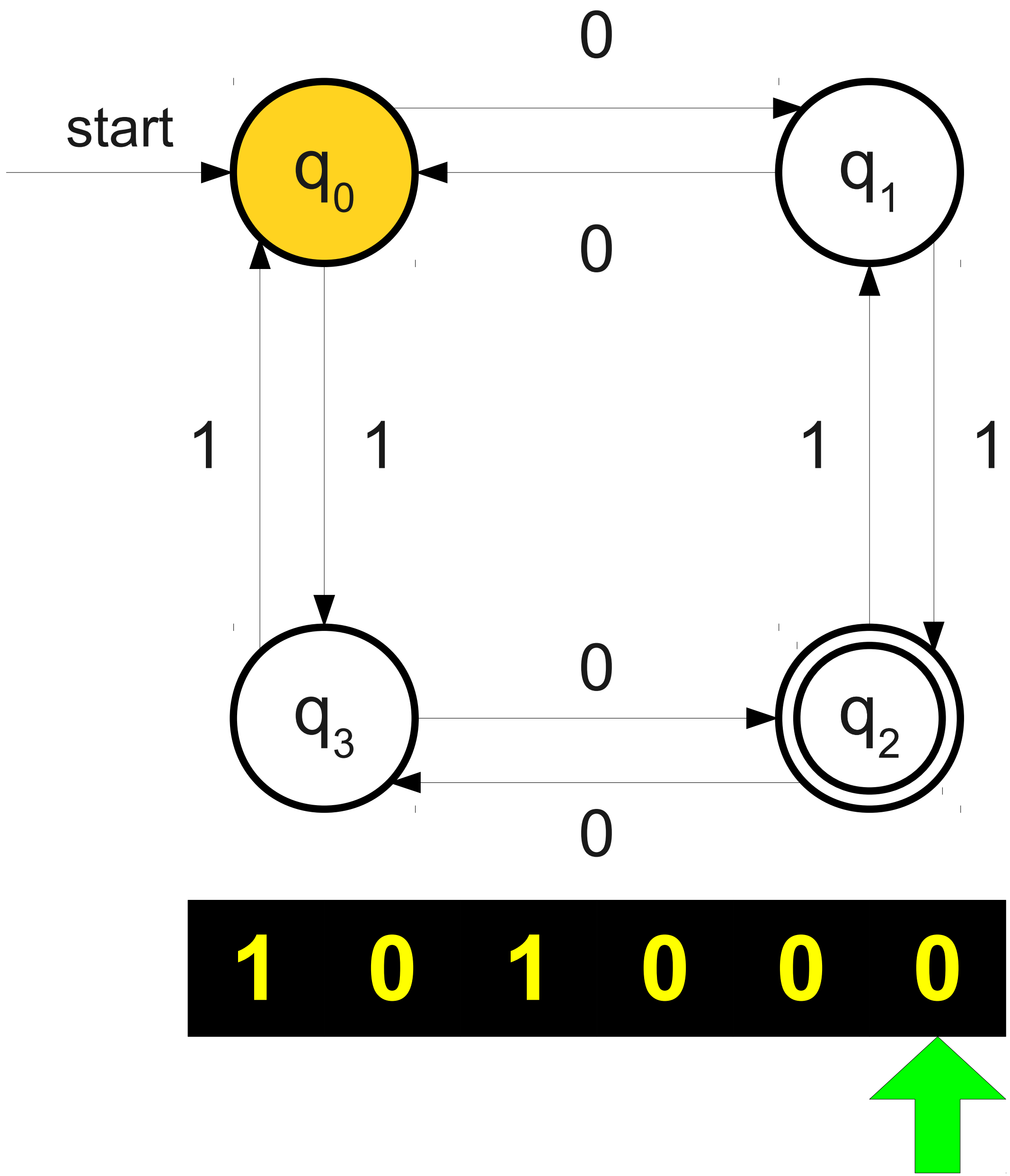
A Simple Finite Automaton



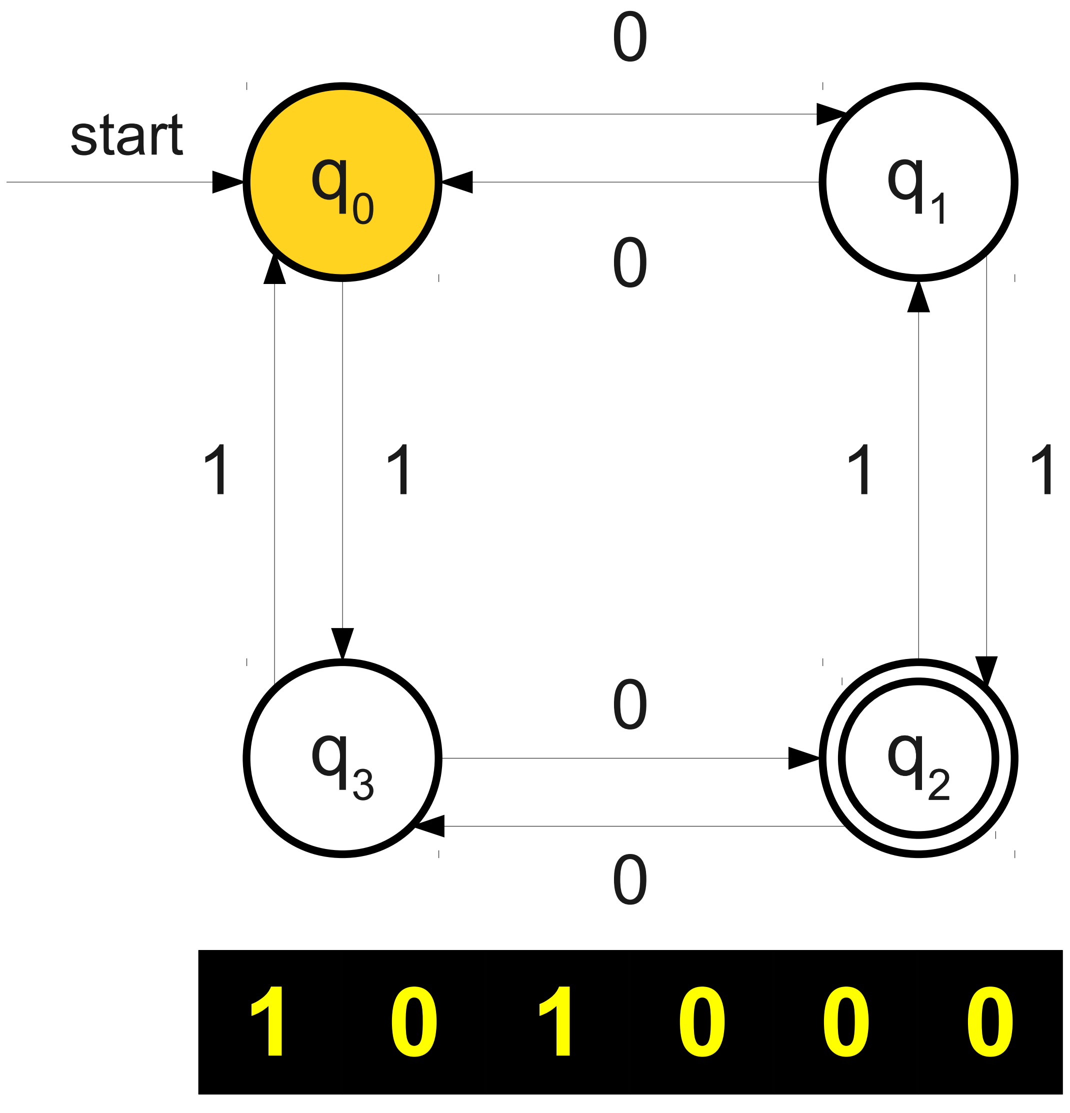
A Simple Finite Automaton



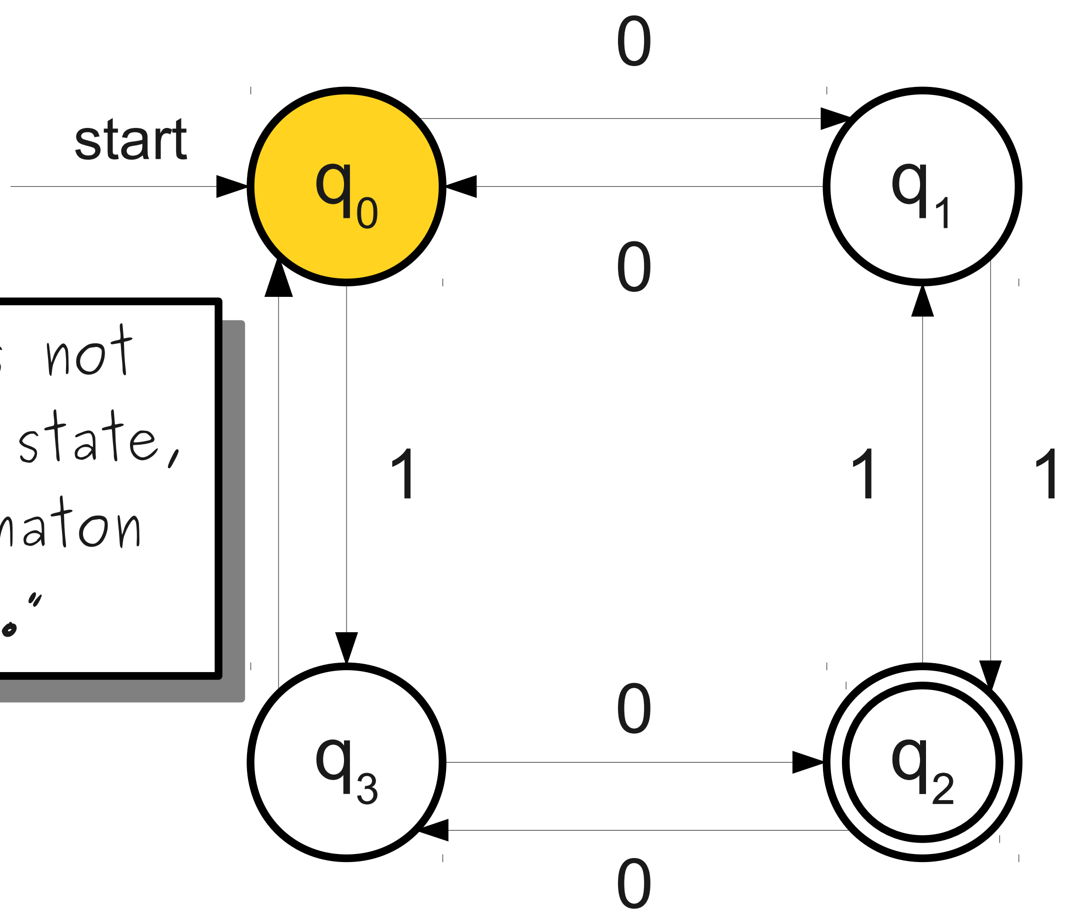
A Simple Finite Automaton



A Simple Finite Automaton



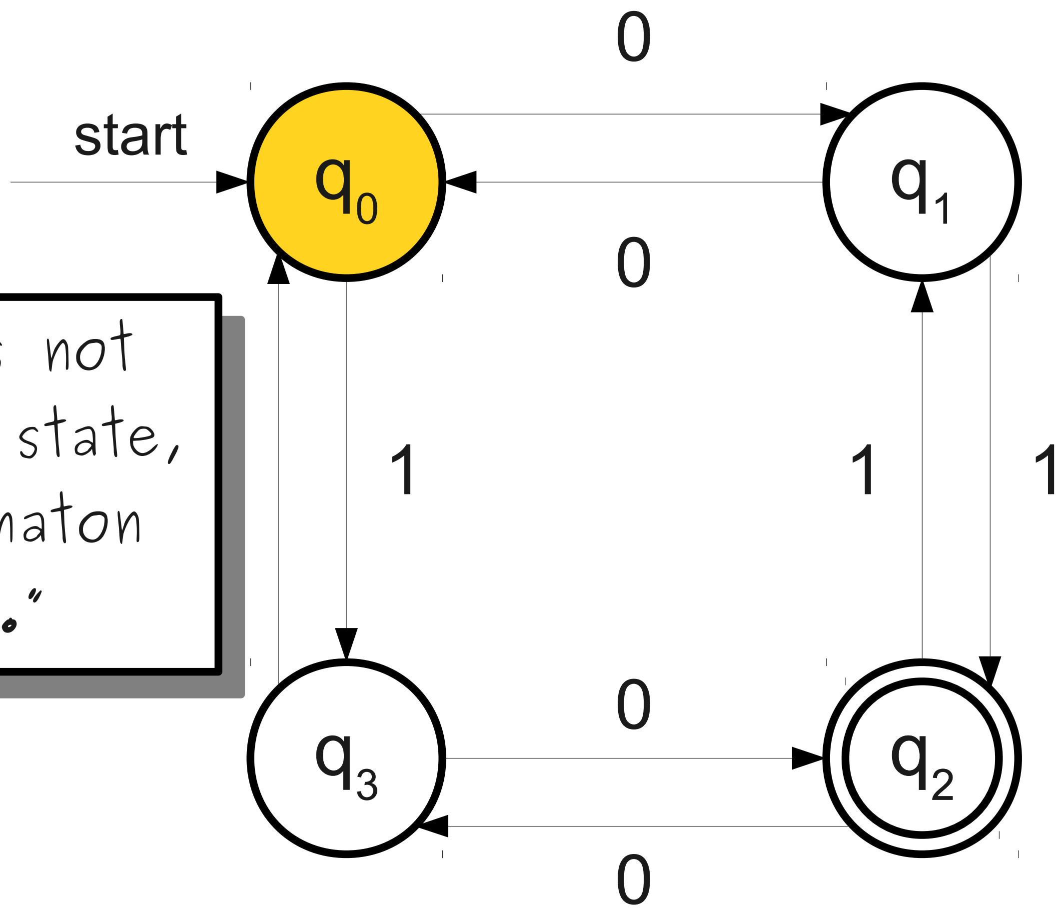
A Simple Finite Automaton



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

1 0 1 0 0 0

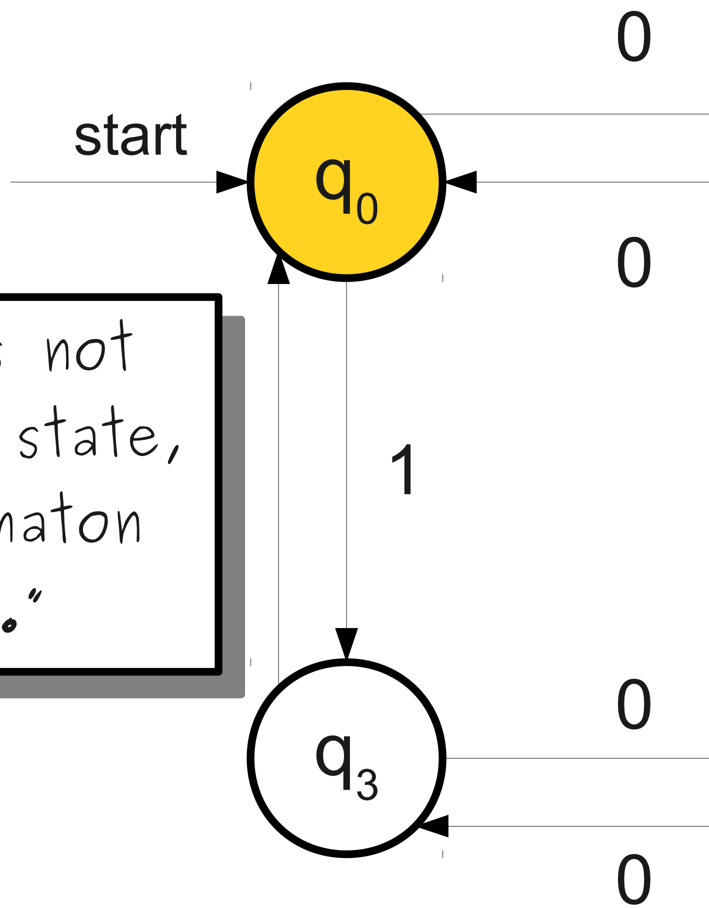
A Simple Finite Automaton



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

1 0 1 0 0 0

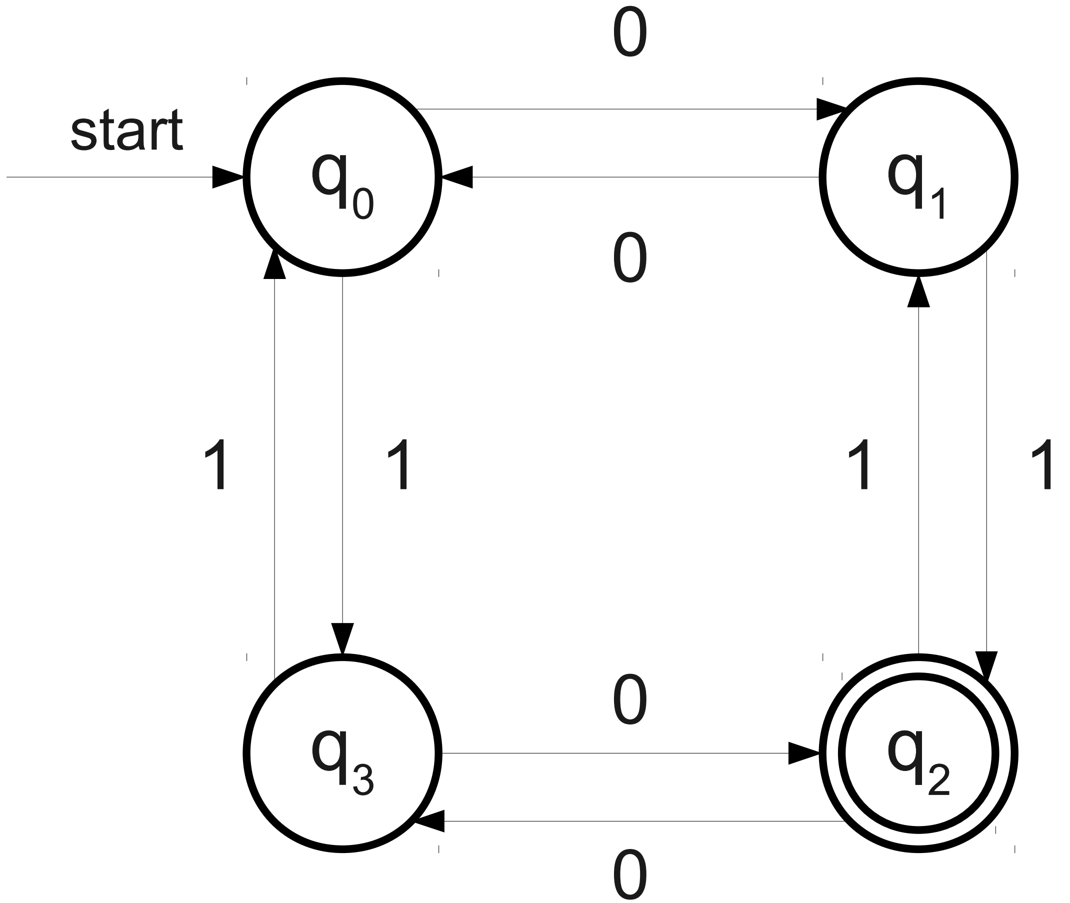
A Simple Finite Automaton



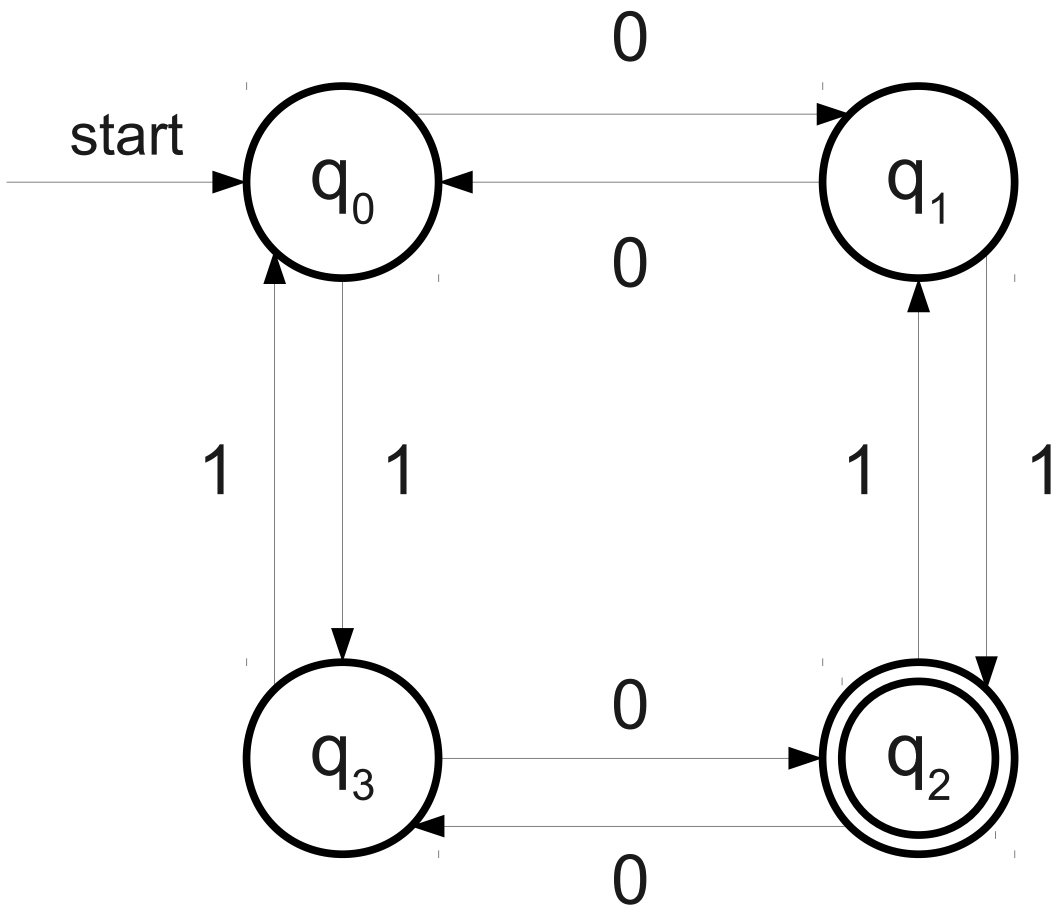
This state is not an accepting state, so the automaton says "no."

1 0 1 0 0 0

A Simple Finite Automaton



A Simple Finite Automaton



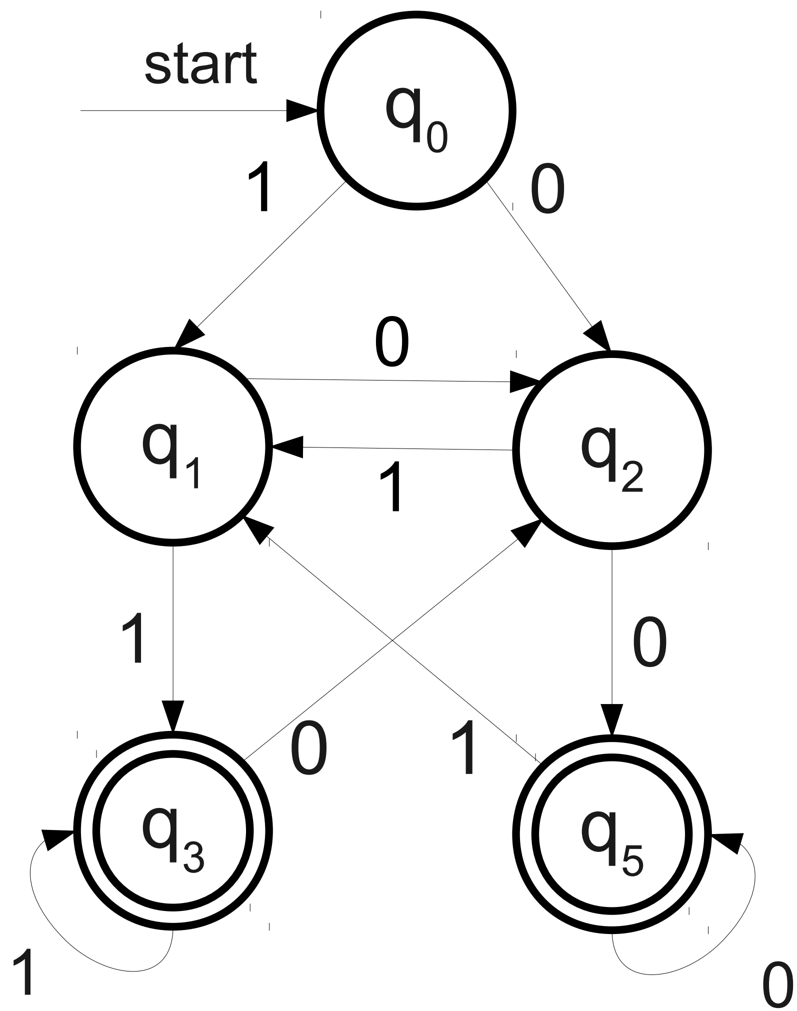
Try it yourself! Does the automaton **accept** (say yes) or **reject** (say no)?

1 1 0 1 1 1 0 0

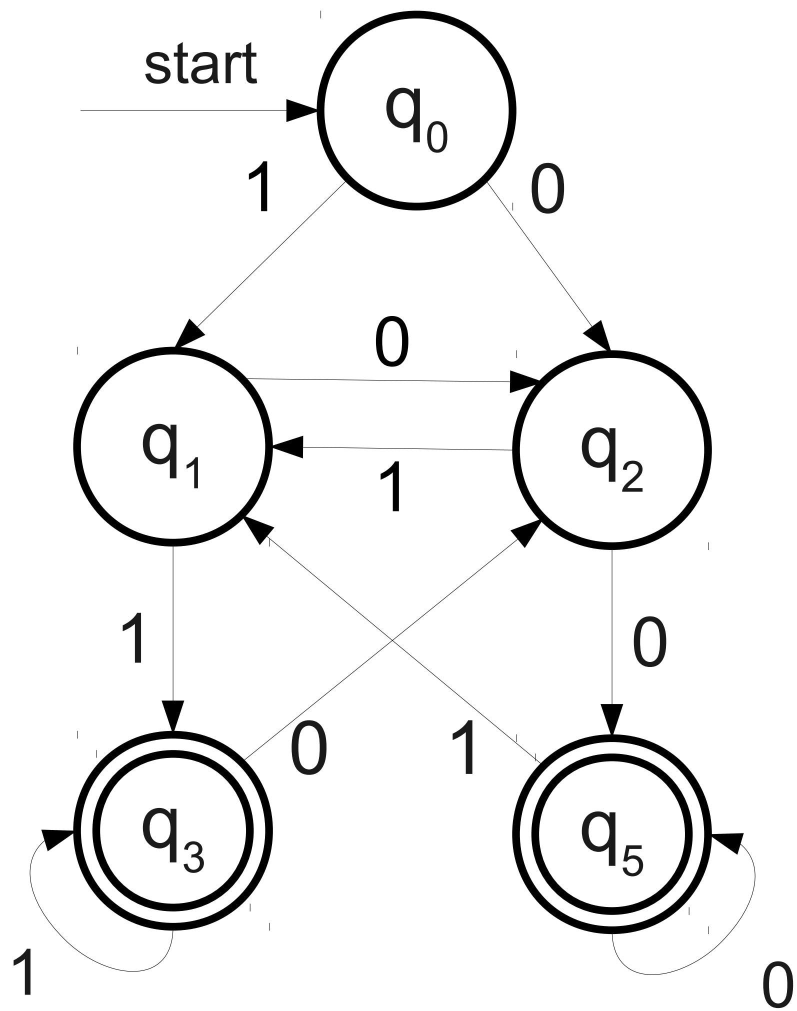
The Story So Far

- 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.

Accepting States, Revisited

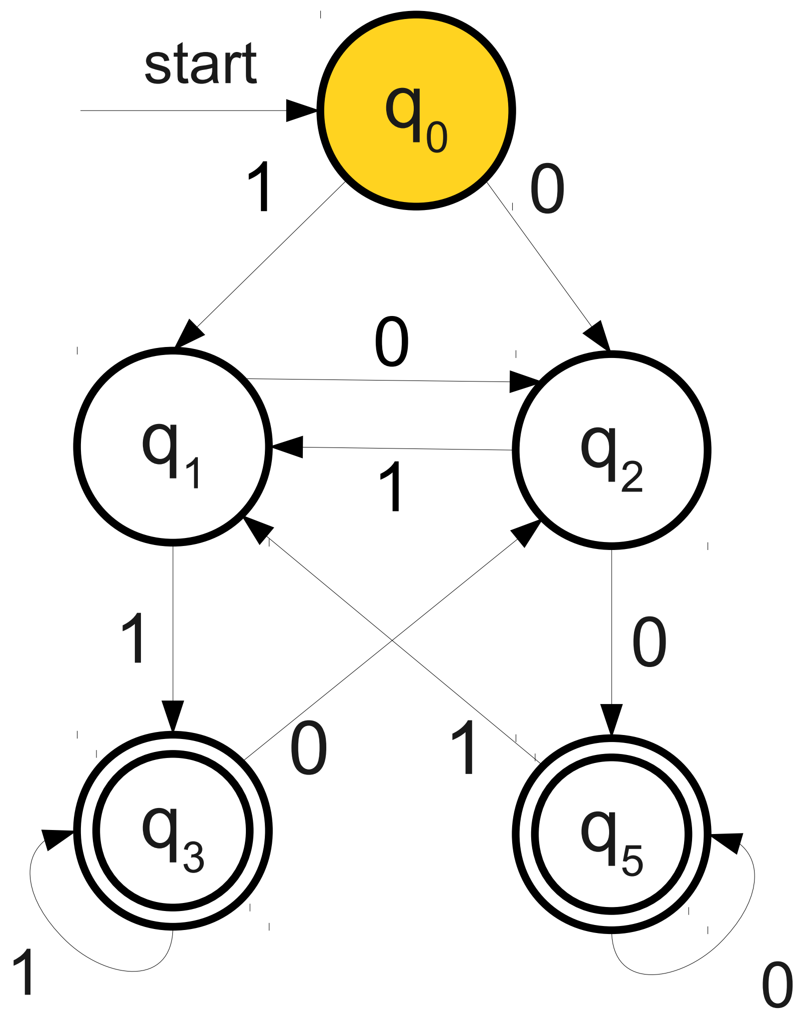


Accepting States, Revisited

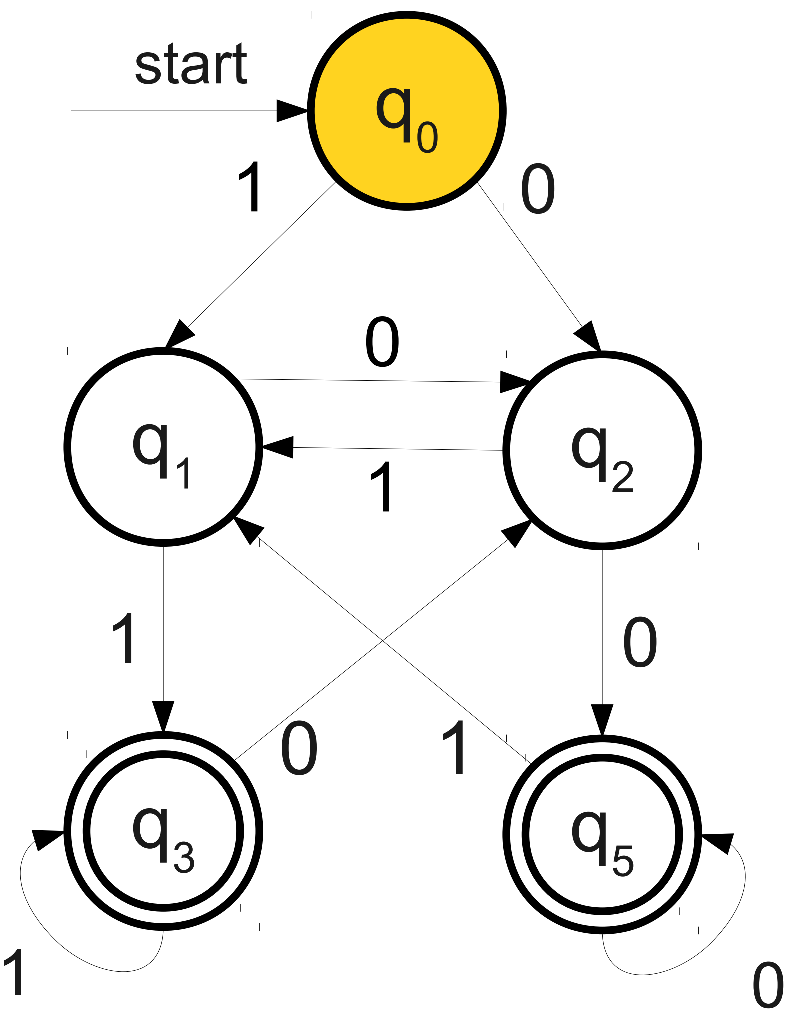


1 1 0 1

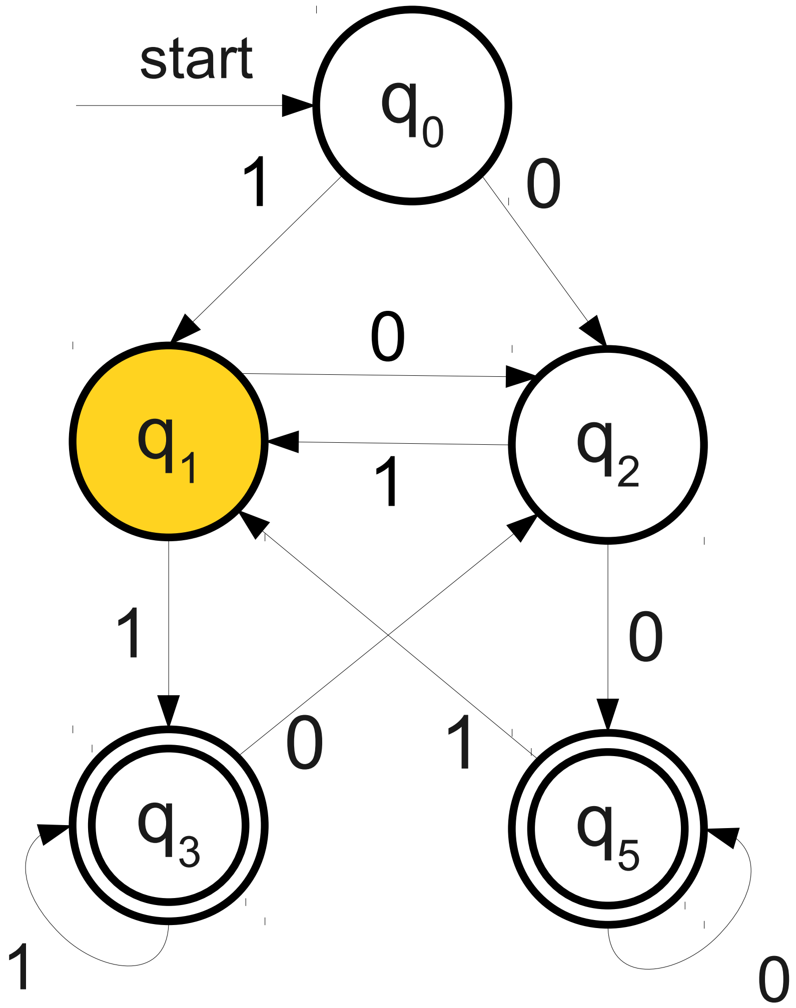
Accepting States, Revisited



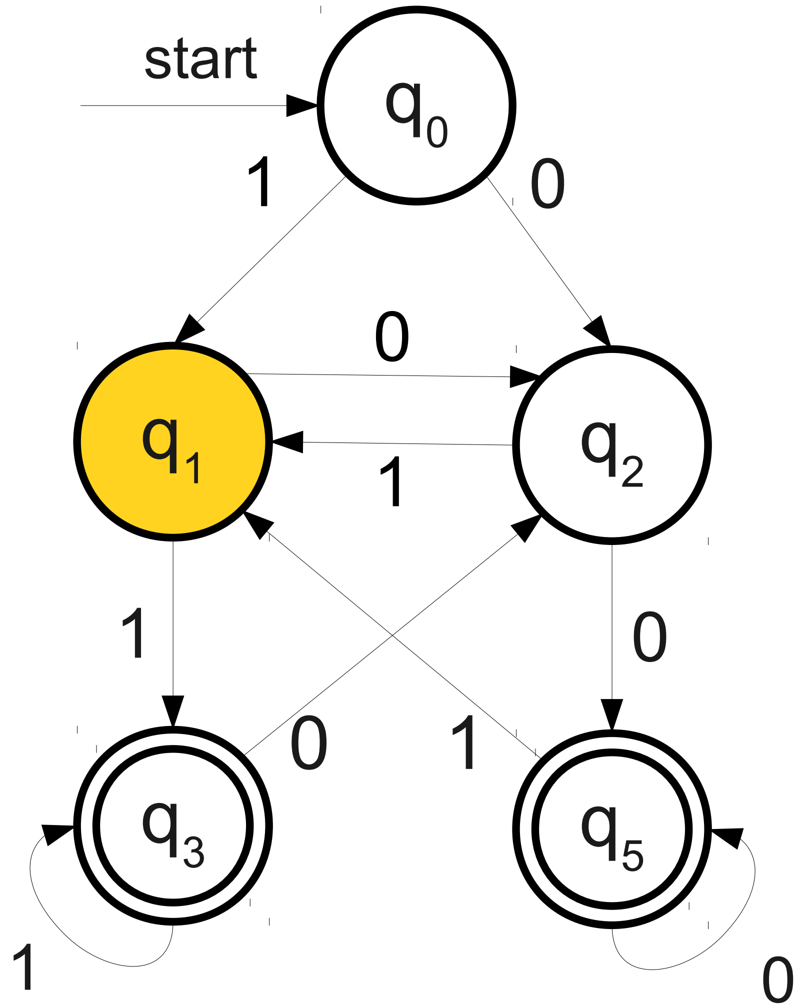
Accepting States, Revisited

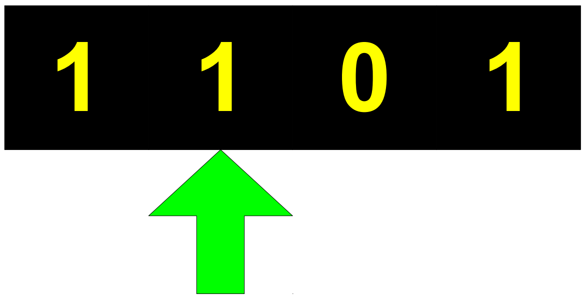


Accepting States, Revisited

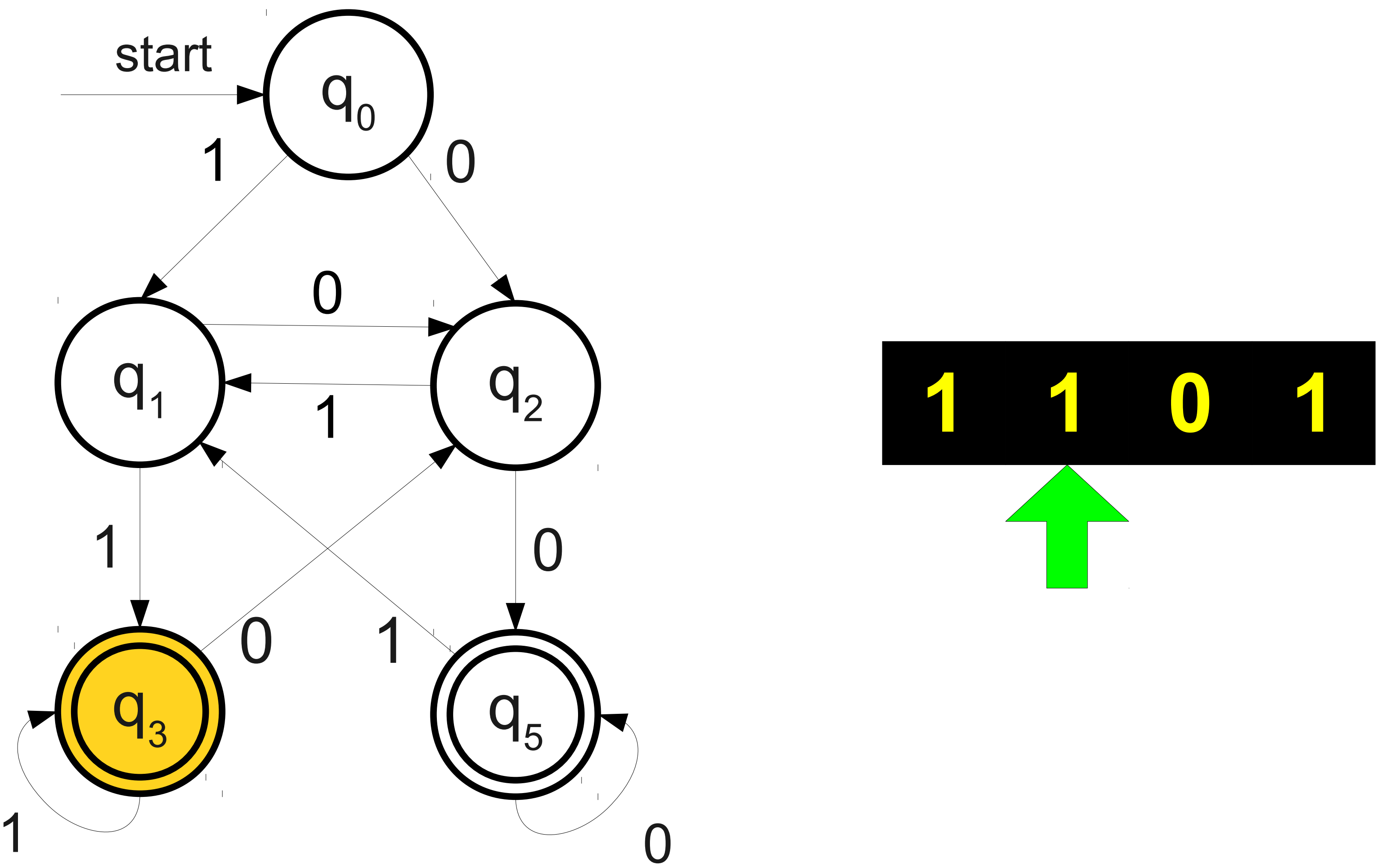


Accepting States, Revisited

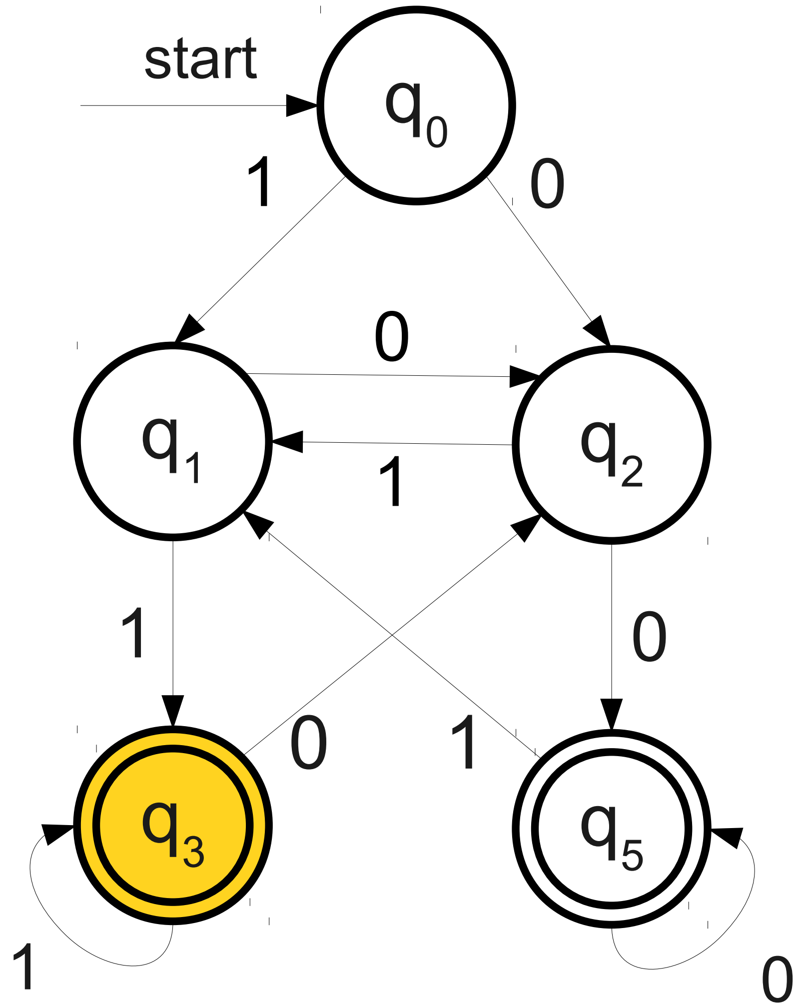


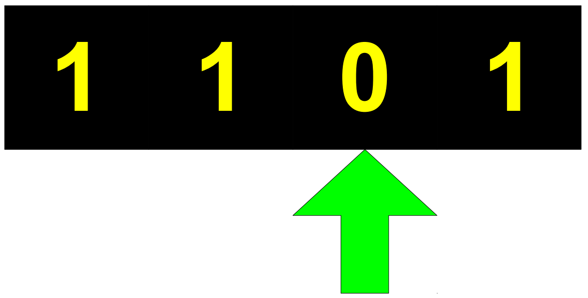


Accepting States, Revisited

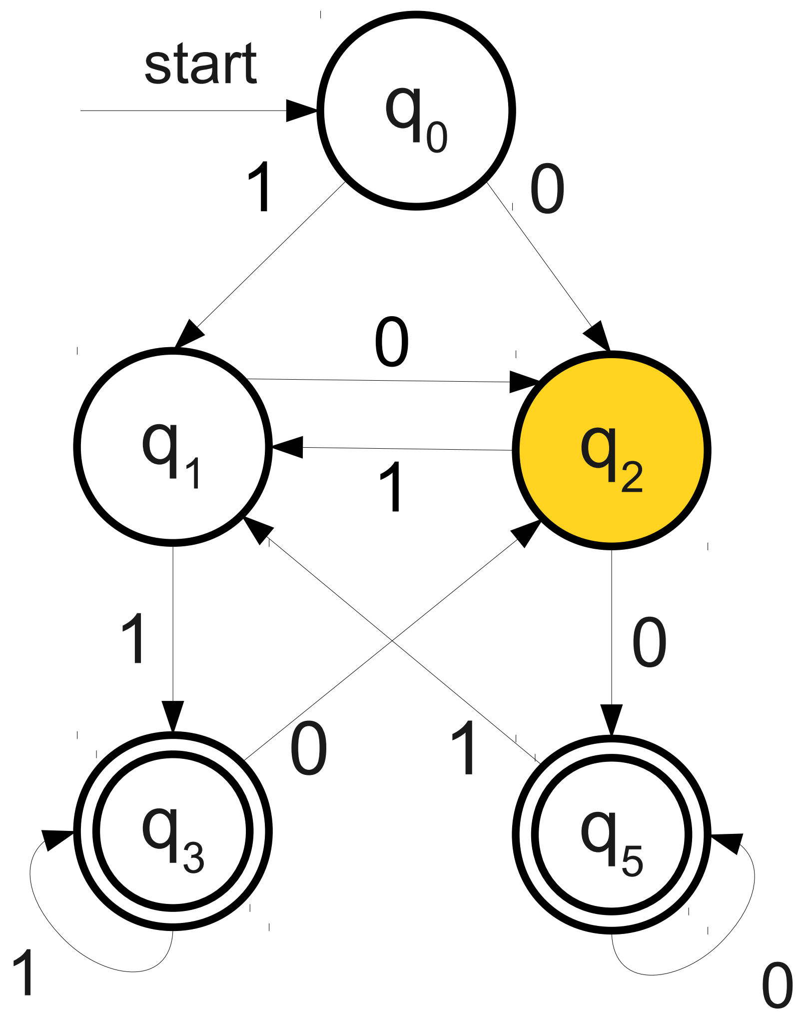


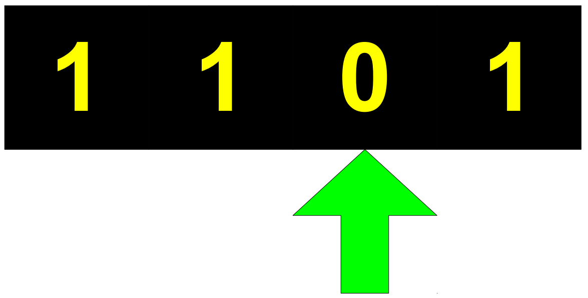
Accepting States, Revisited



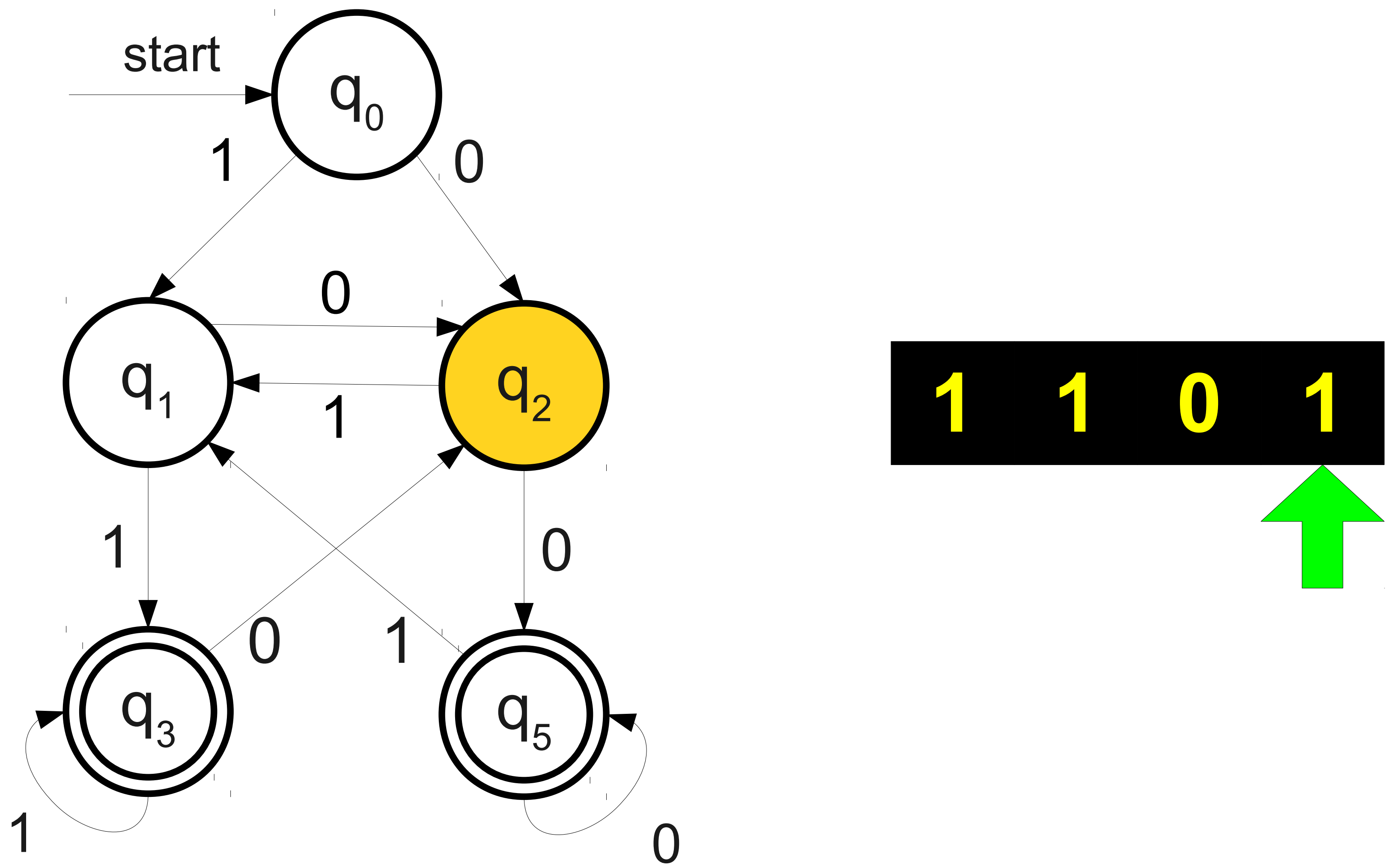


Accepting States, Revisited

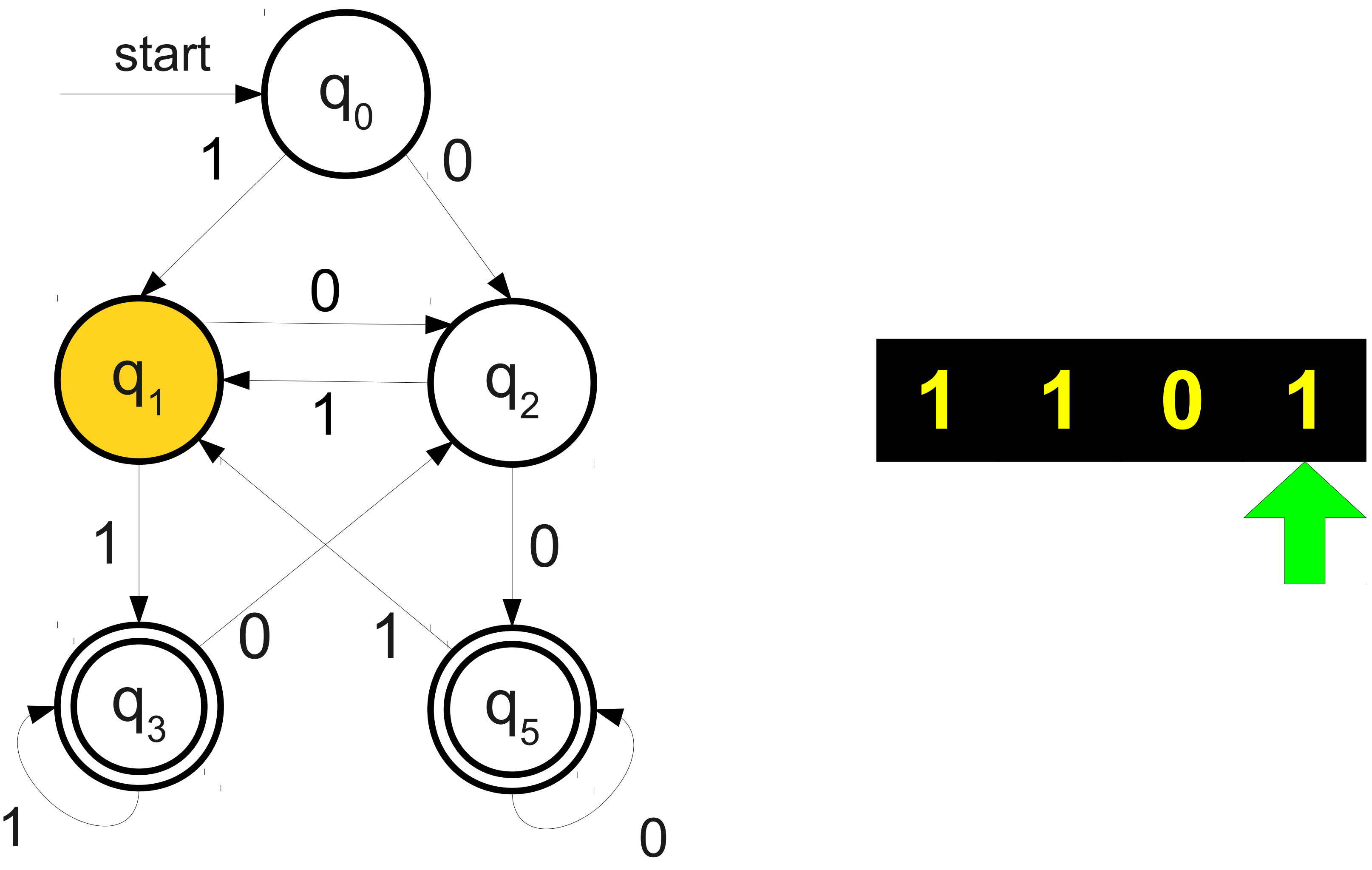




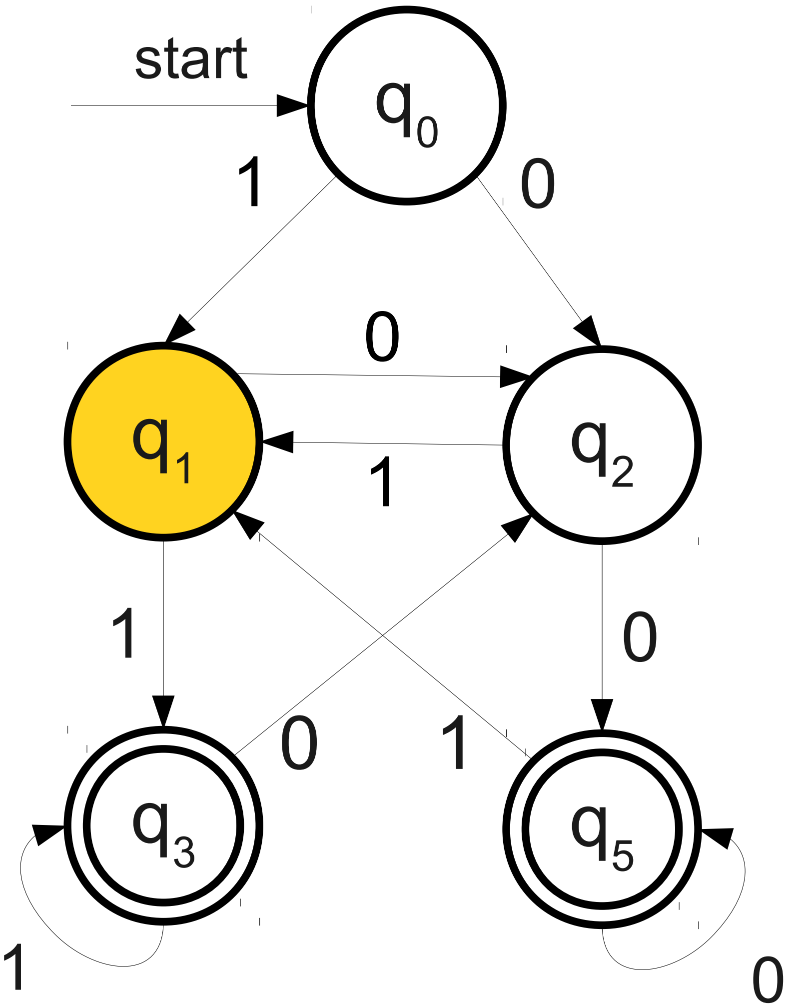
Accepting States, Revisited



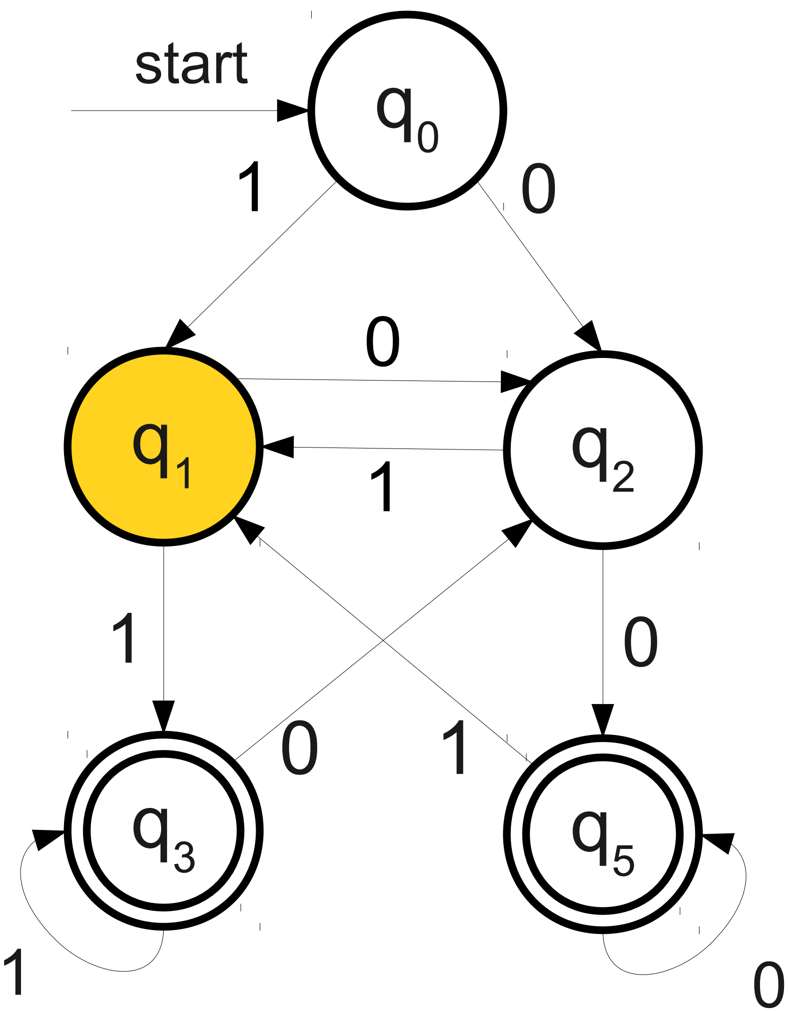
Accepting States, Revisited



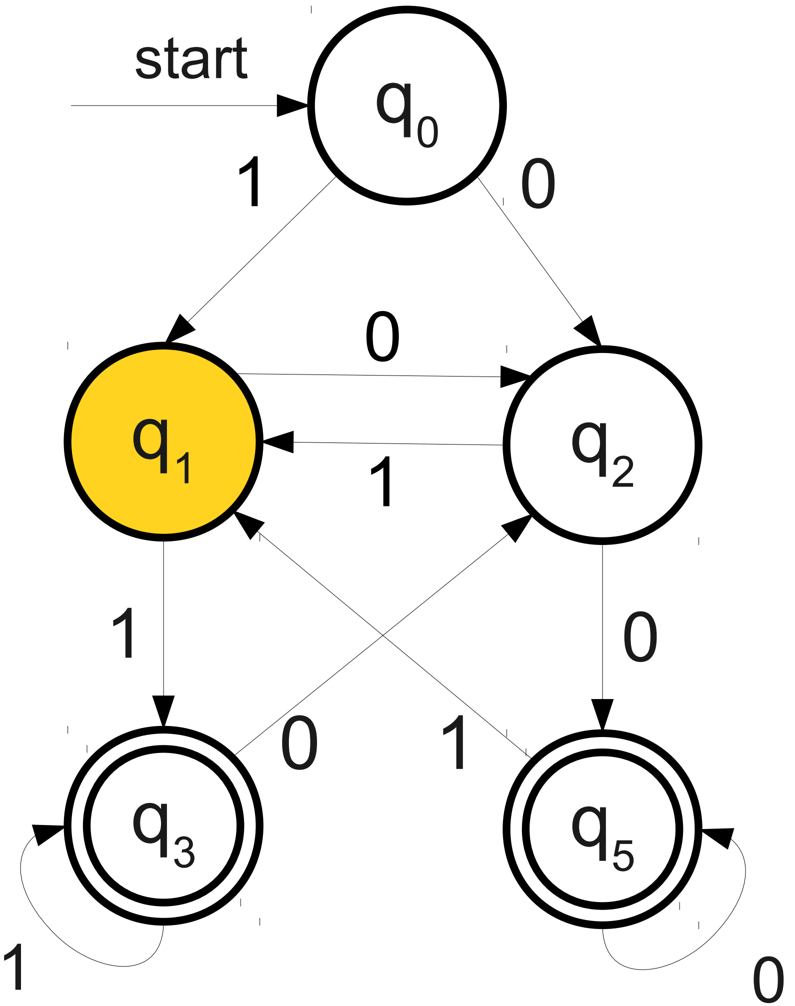
Accepting States, Revisited



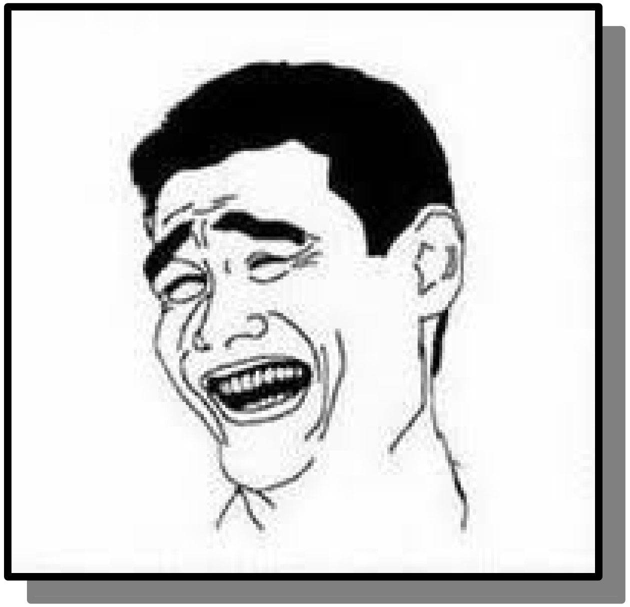
Accepting States, Revisited



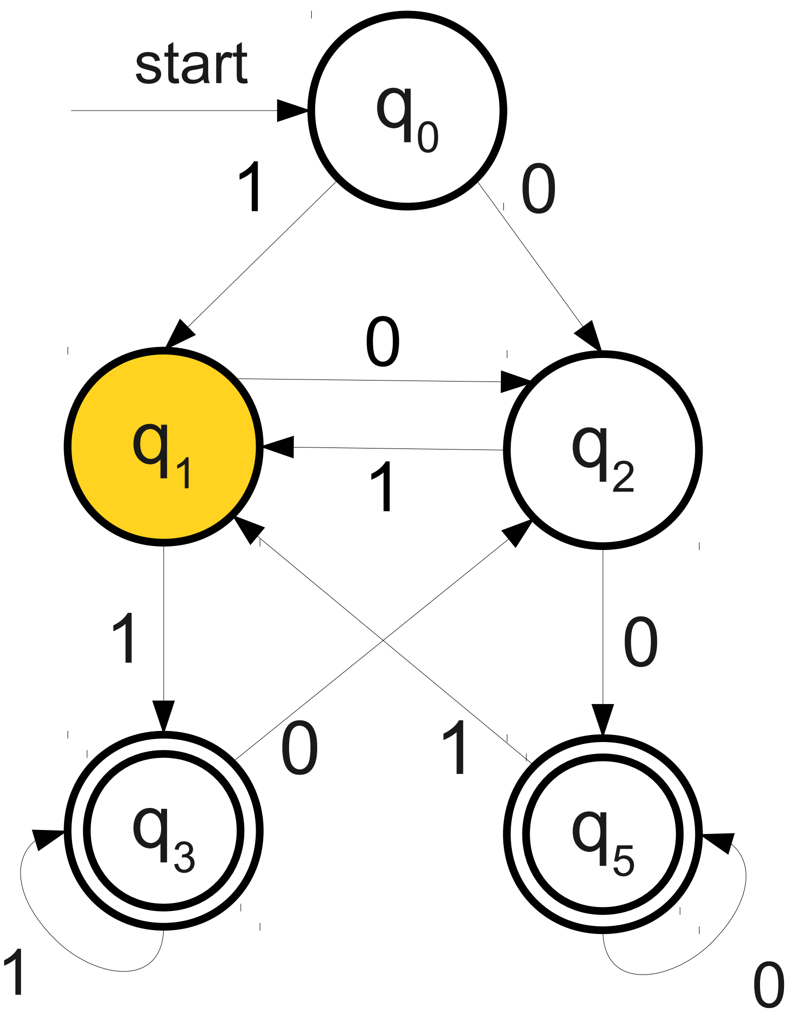
Accepting States, Revisited



1 1 0 1



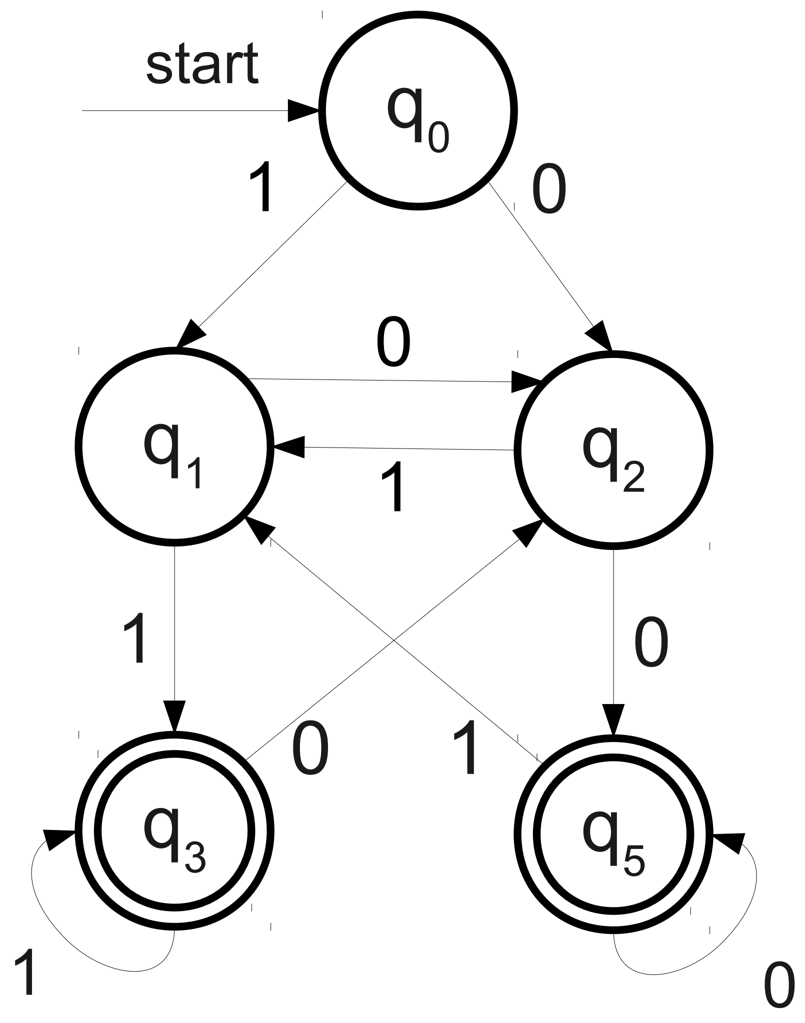
Accepting States, Revisited



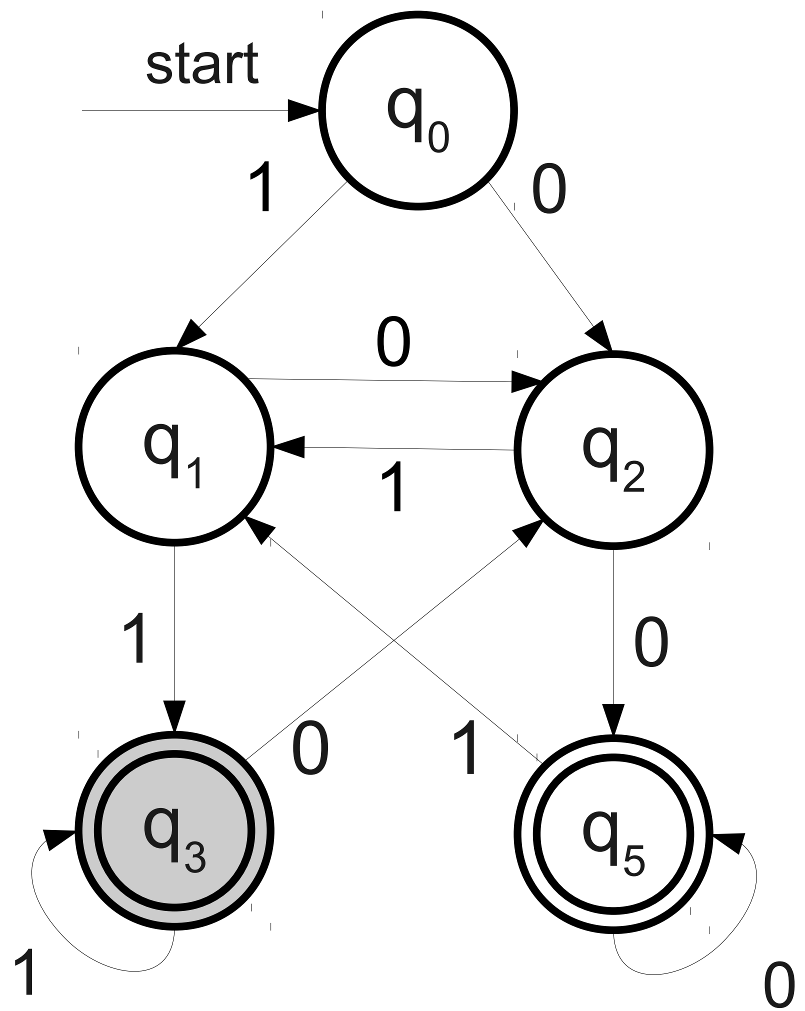
A finite automaton does **not** accept as soon as the input enters an accepting state.

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

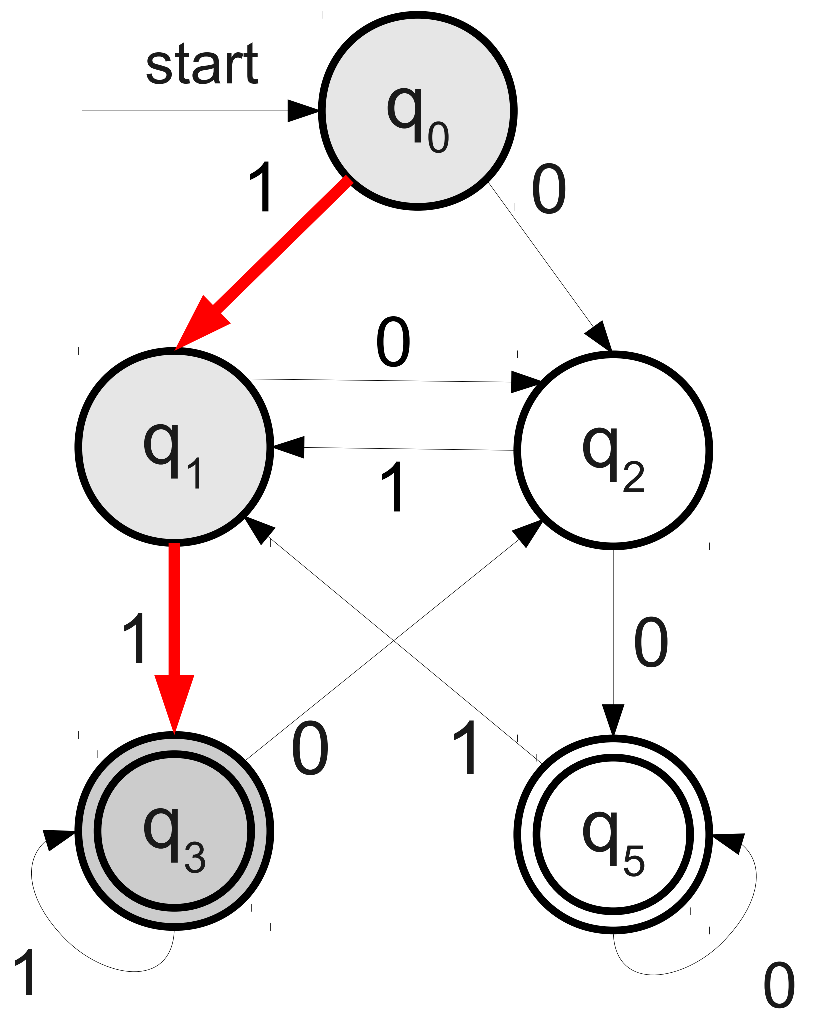
What Does This Accept?



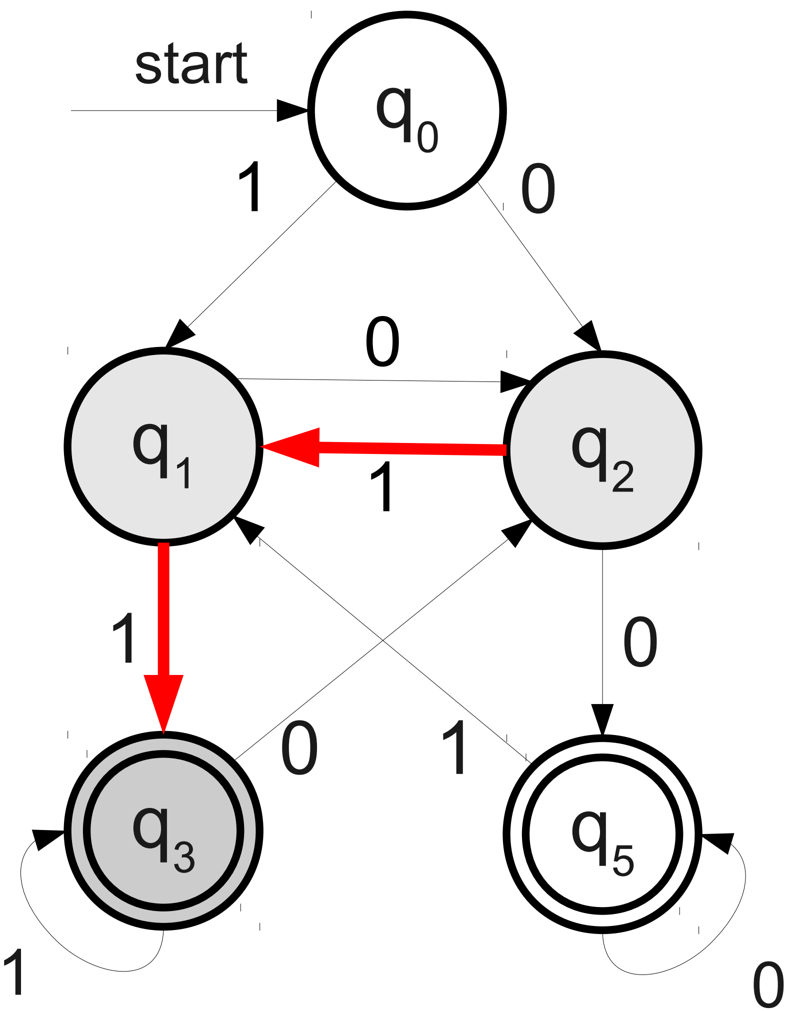
What Does This Accept?



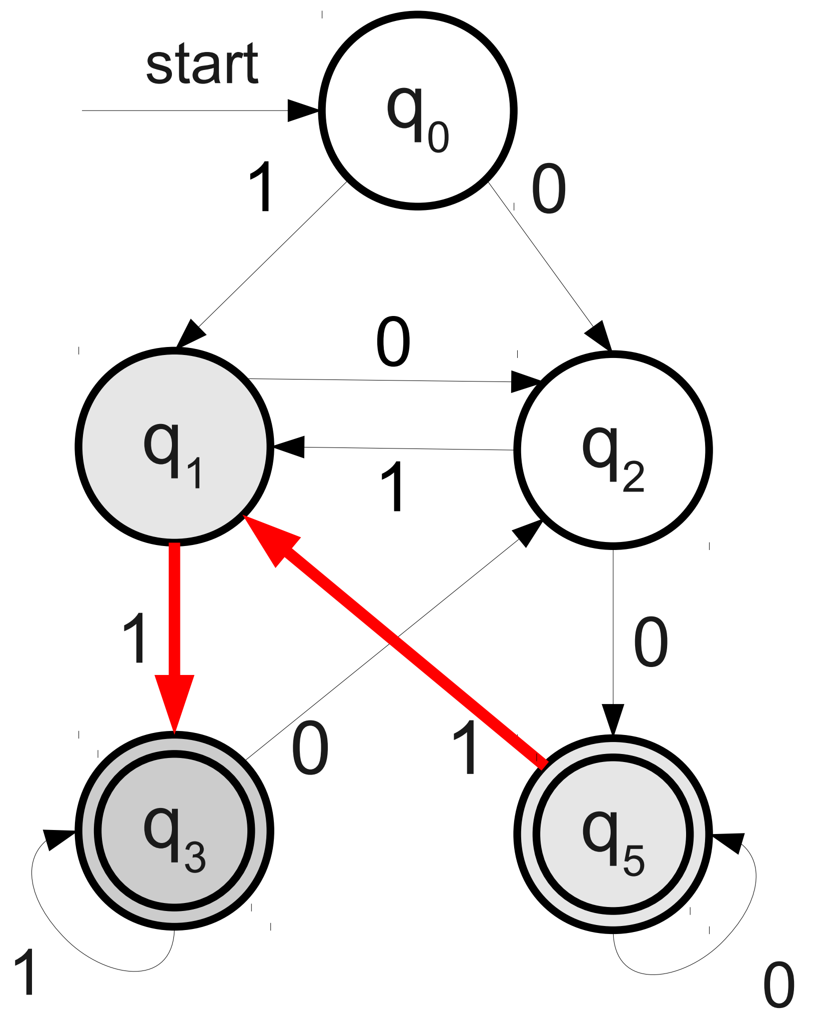
What Does This Accept?



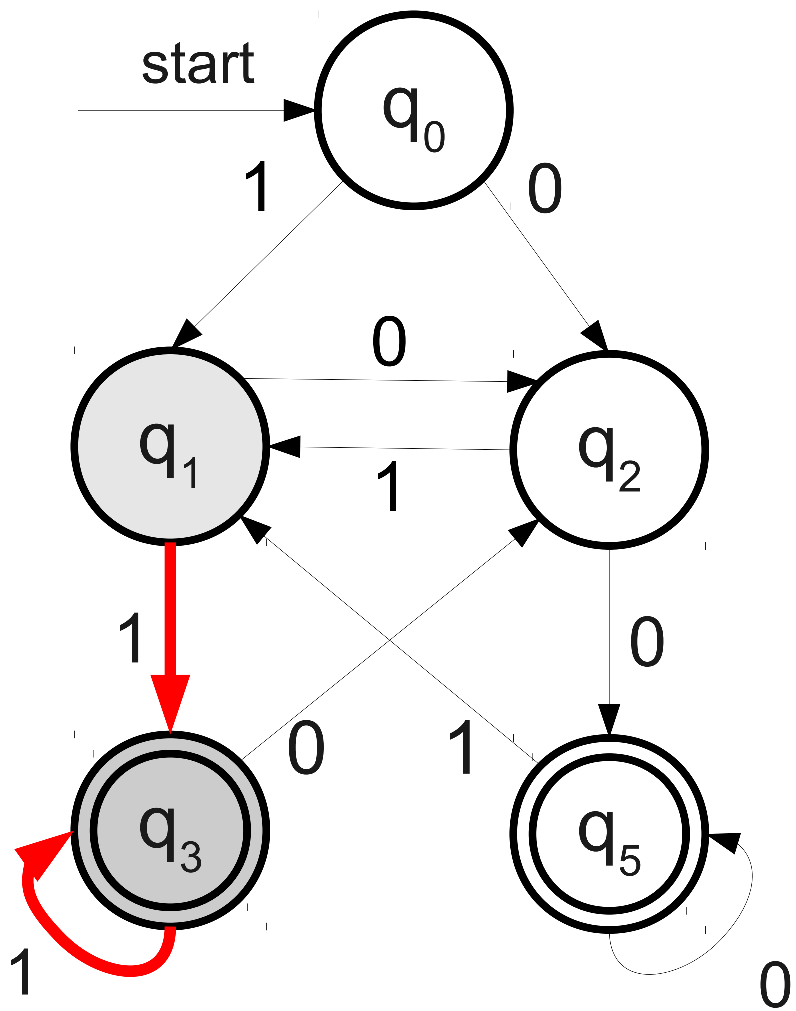
What Does This Accept?



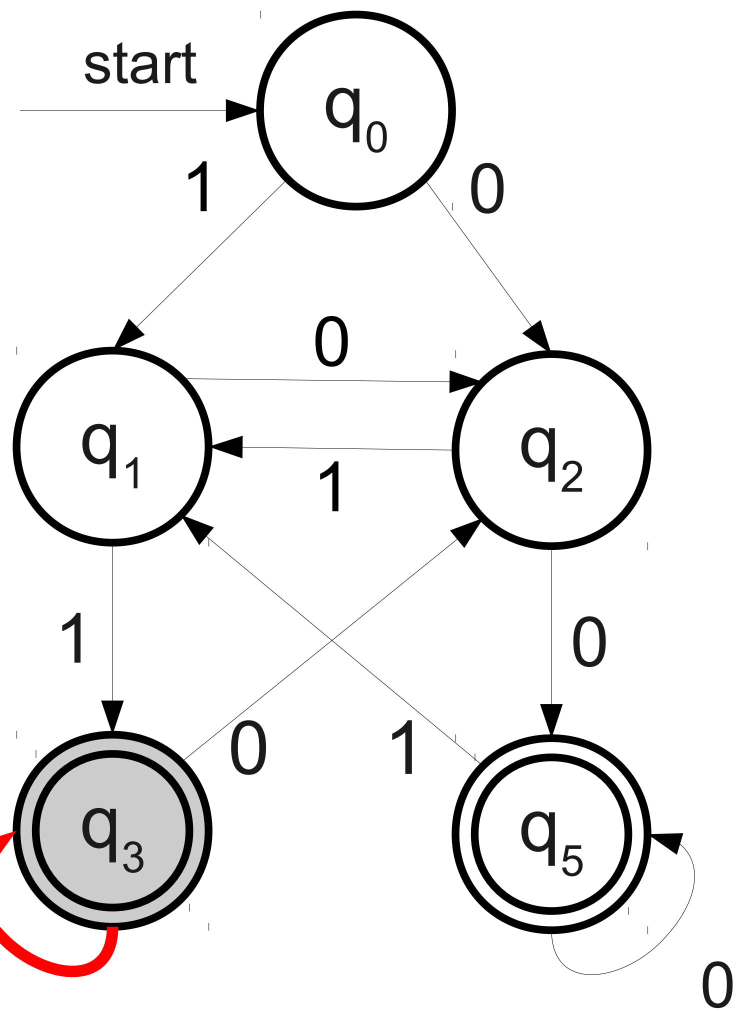
What Does This Accept?



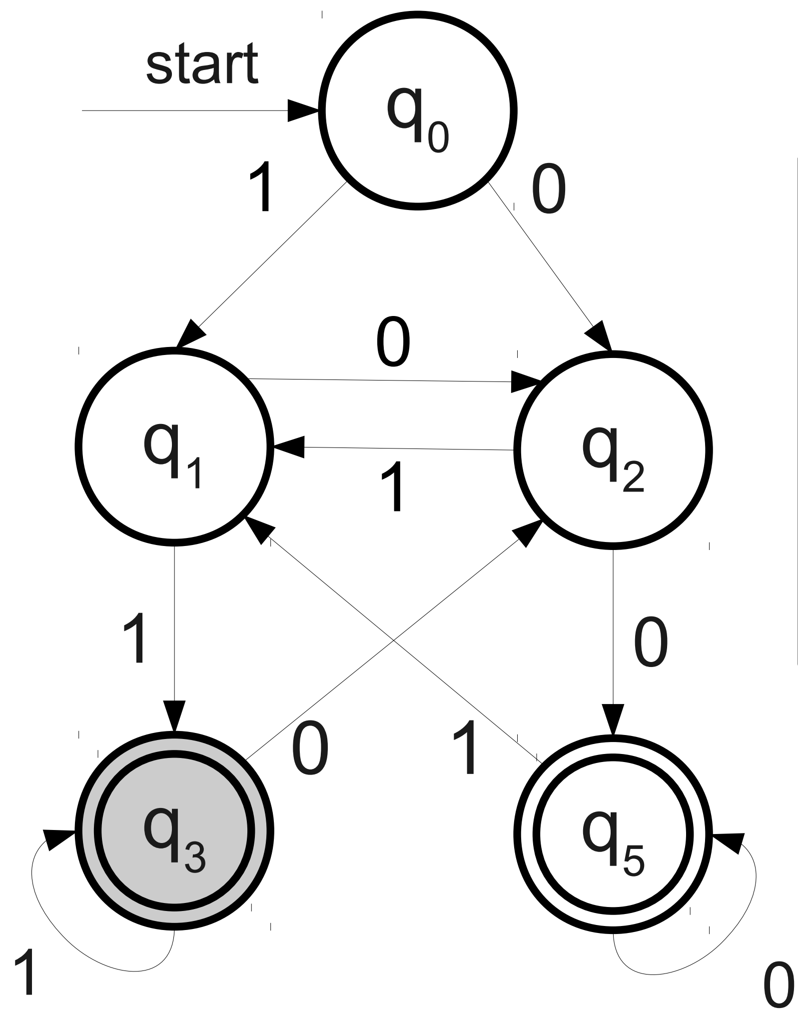
What Does This Accept?



What Does This Accept?

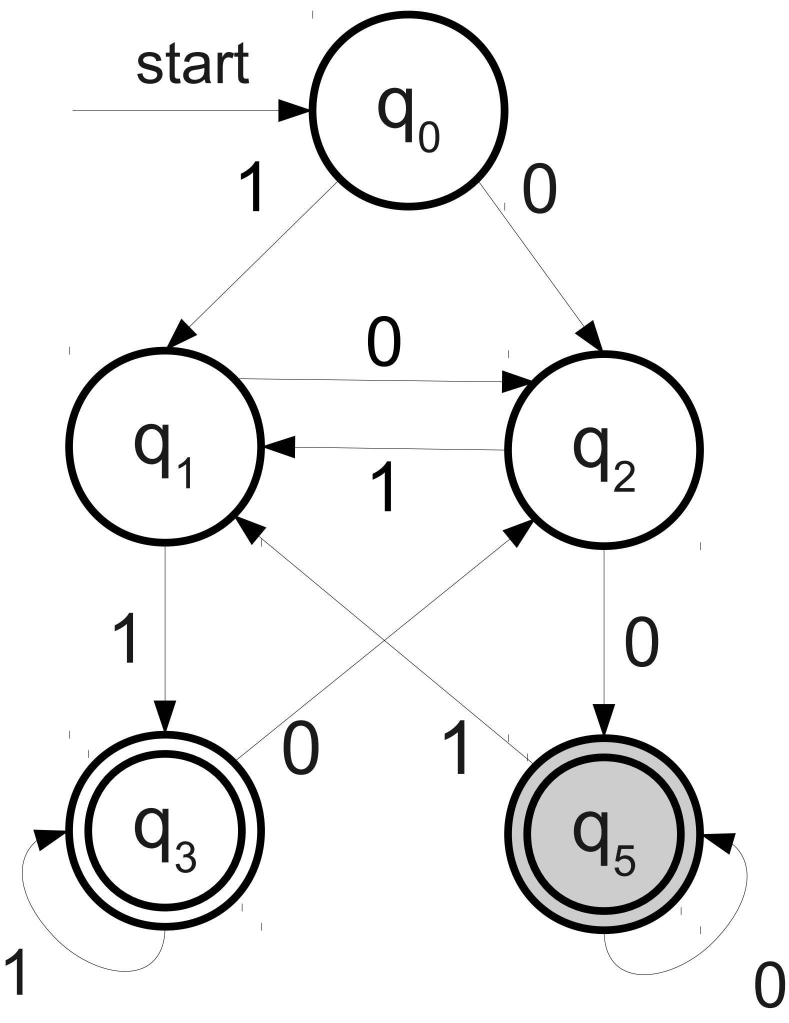


What Does This Accept?

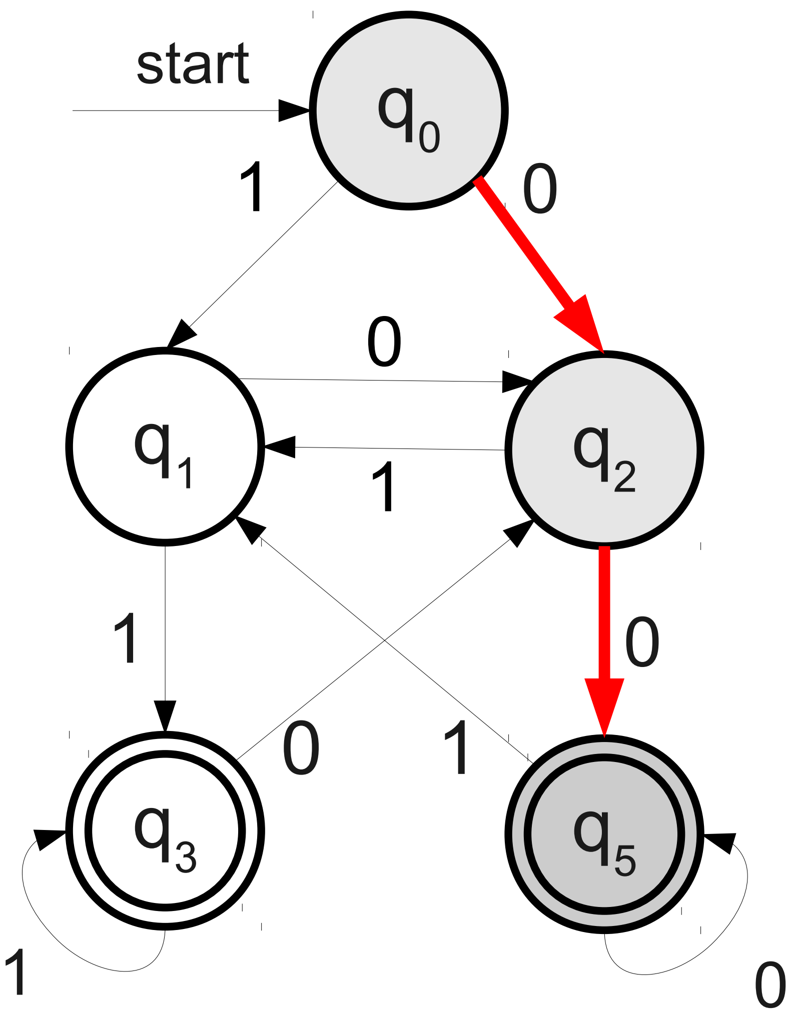


No matter where we start in the automaton, after seeing two 1's, we end up in accepting state q_3.

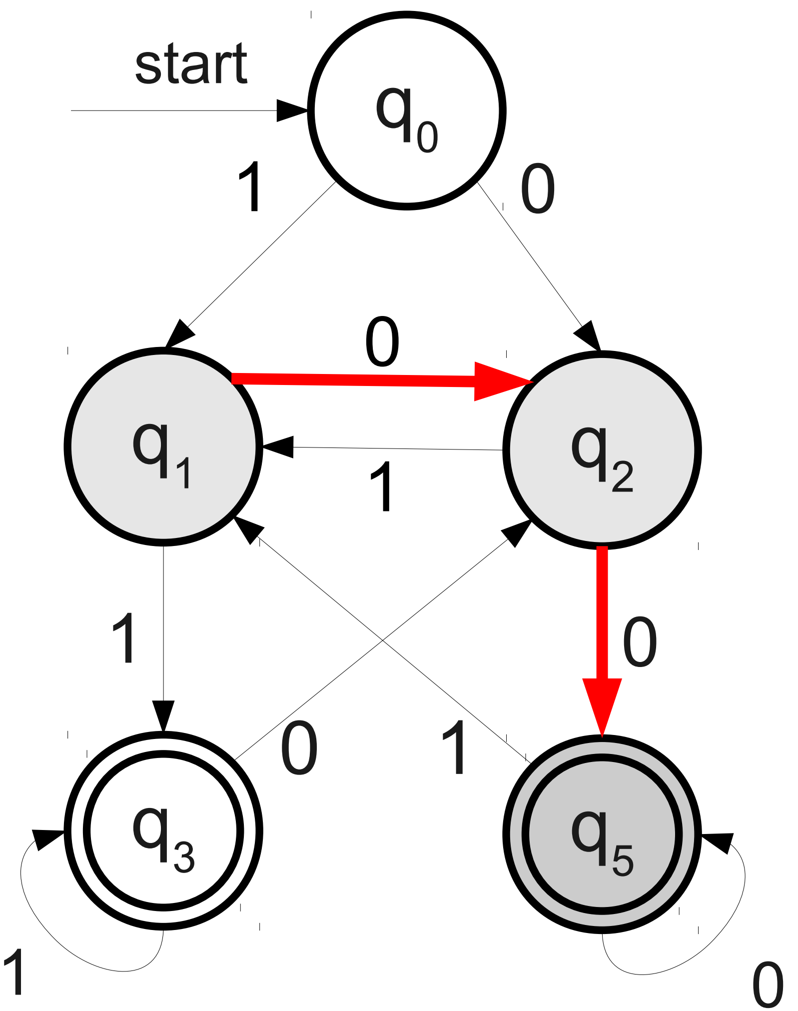
What Does This Accept?



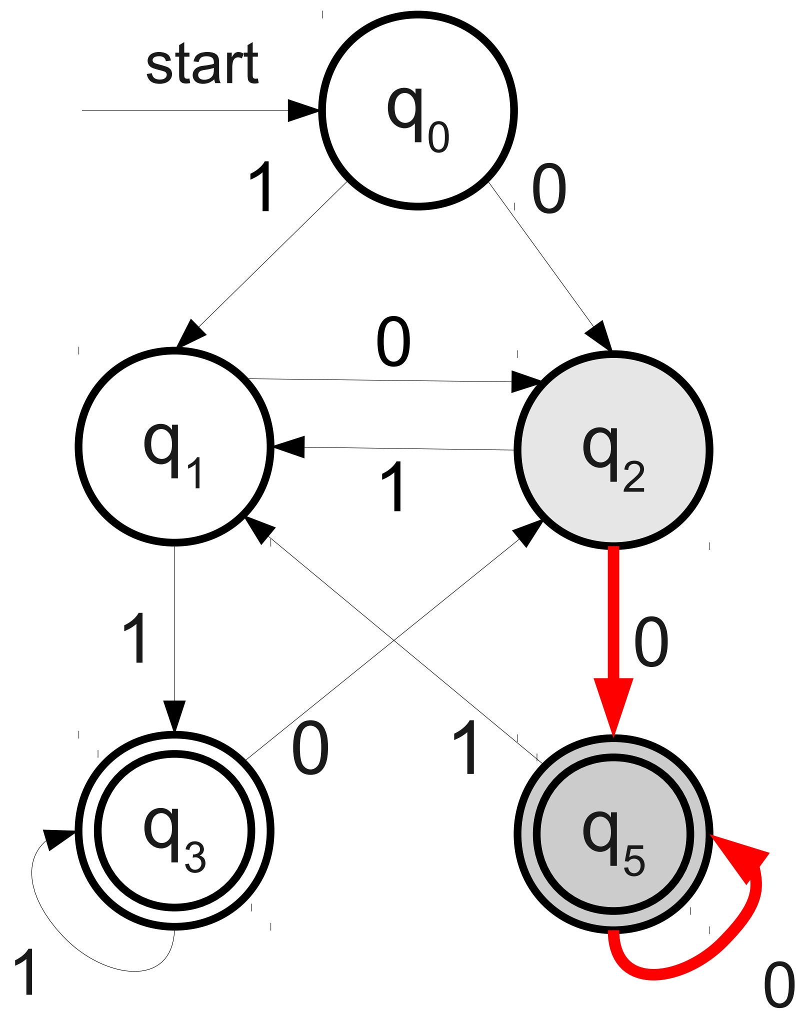
What Does This Accept?



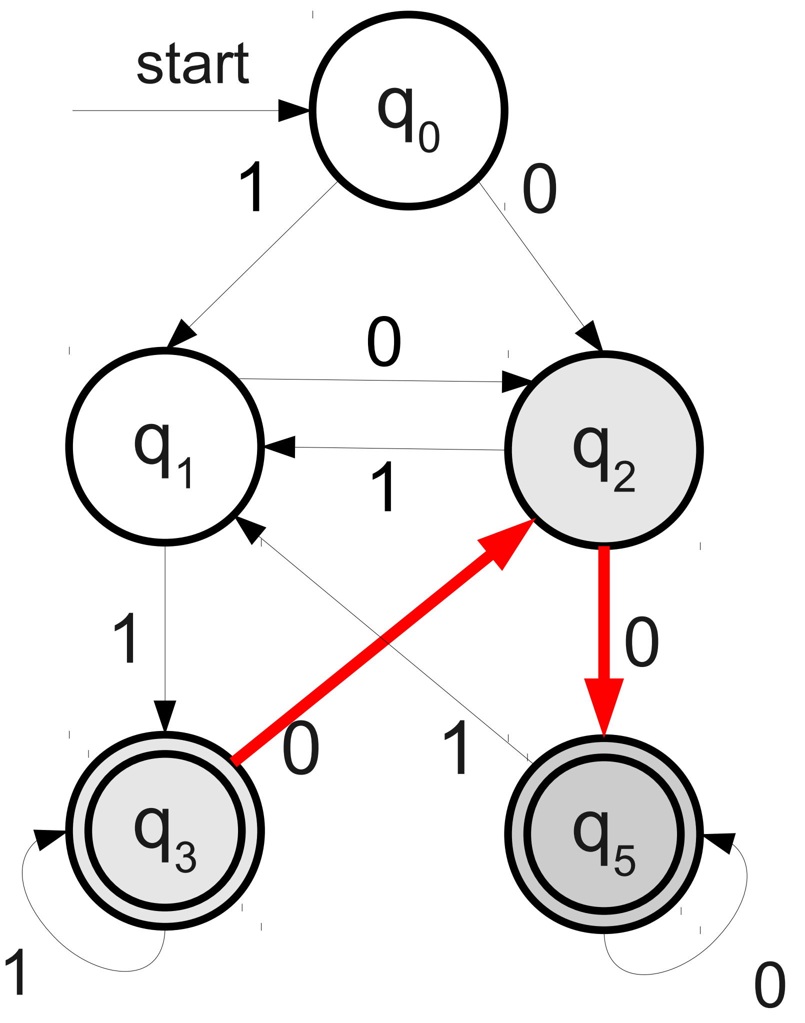
What Does This Accept?



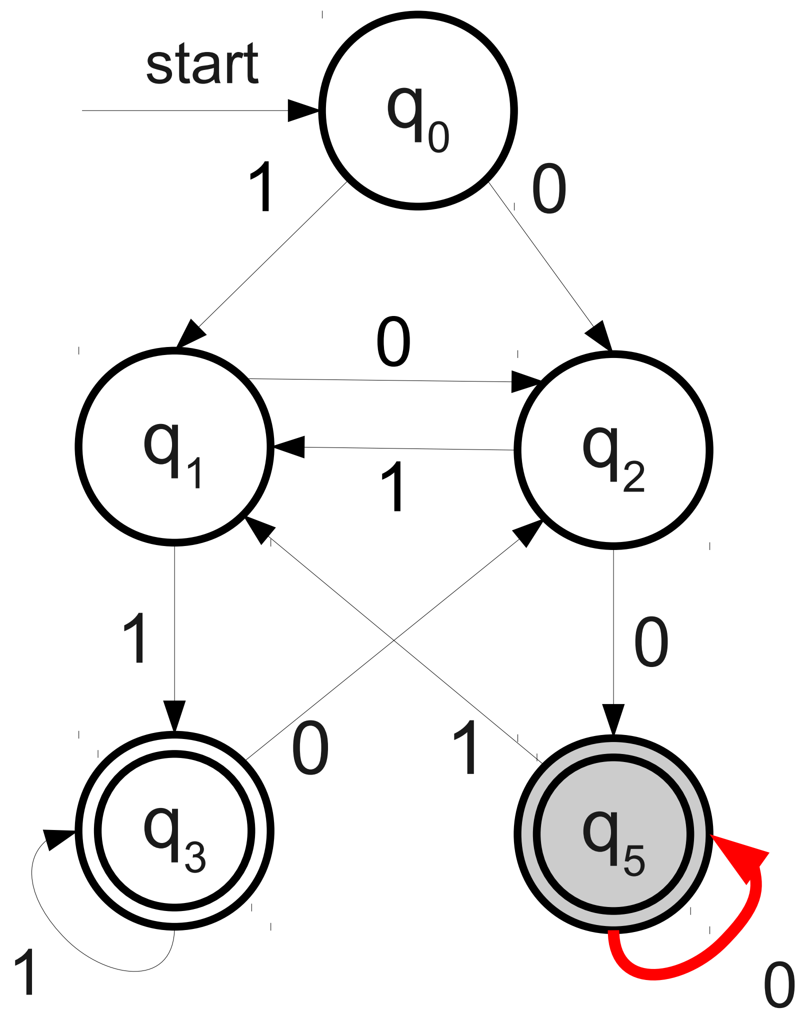
What Does This Accept?



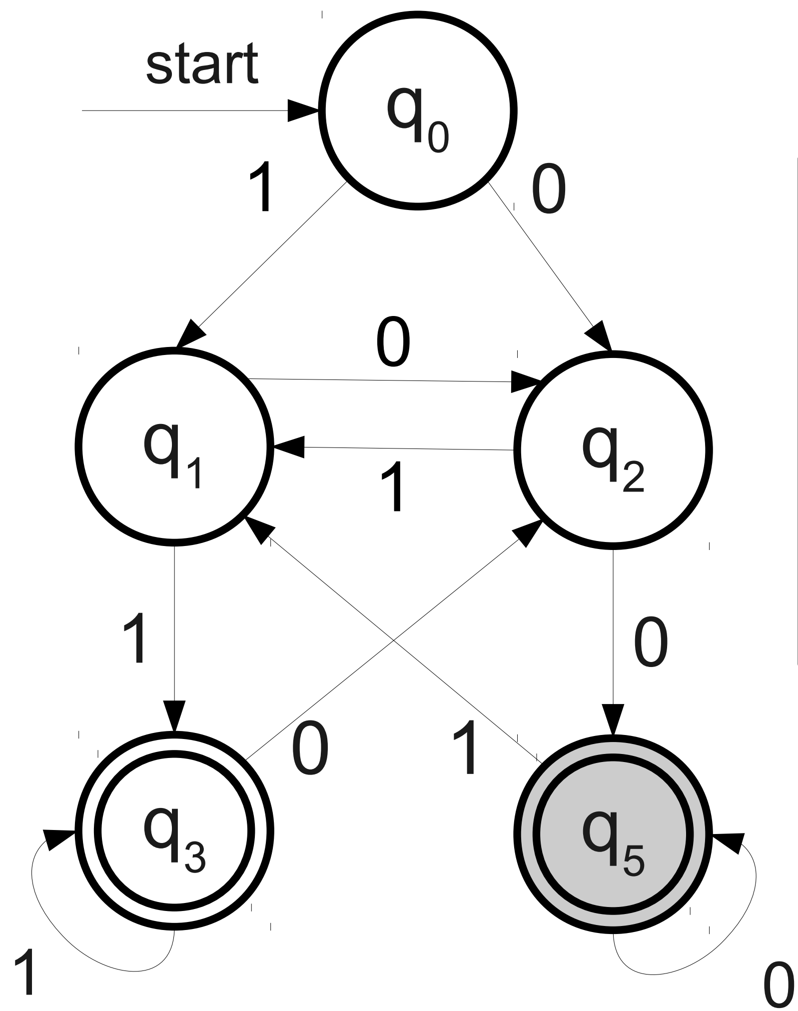
What Does This Accept?



What Does This Accept?

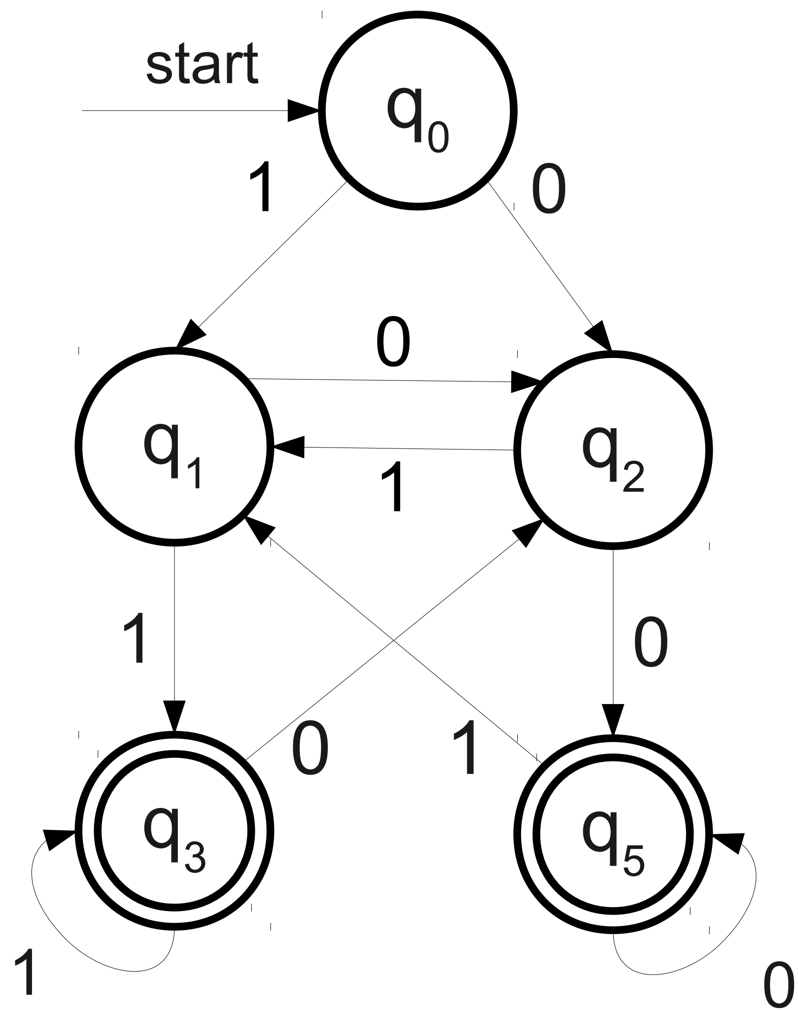


What Does This Accept?

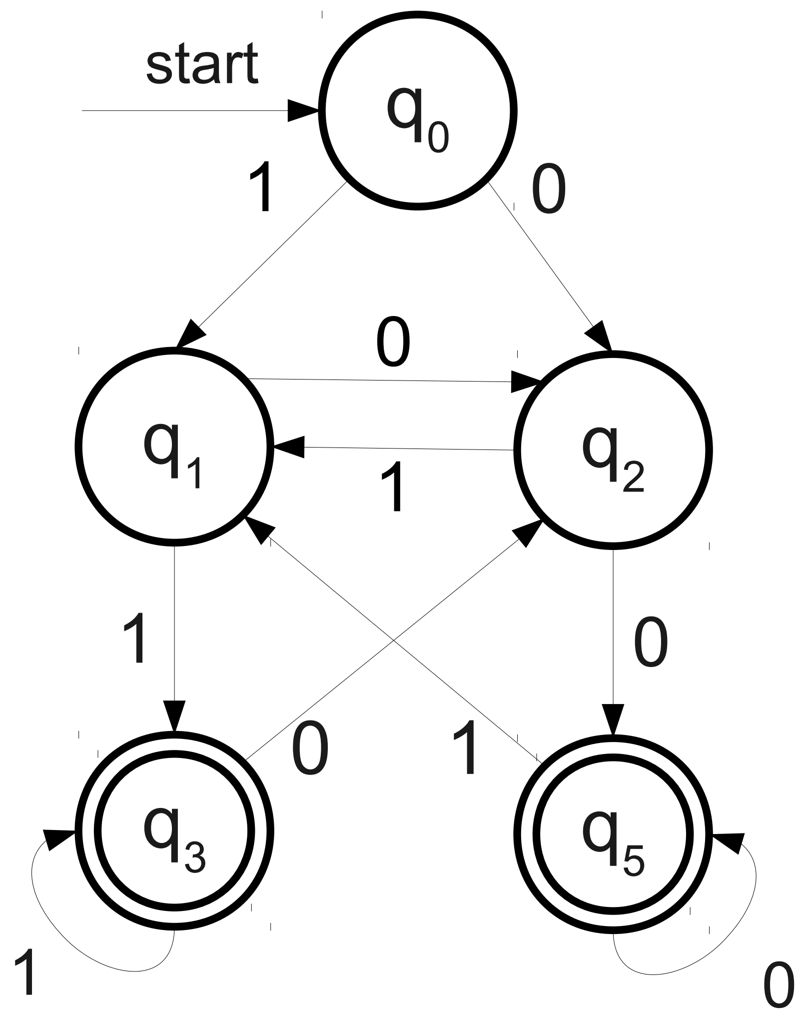


No matter where we start in the automaton, after seeing two 0's, we end up in accepting state q_5.

What Does This Accept?



What Does This Accept?



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

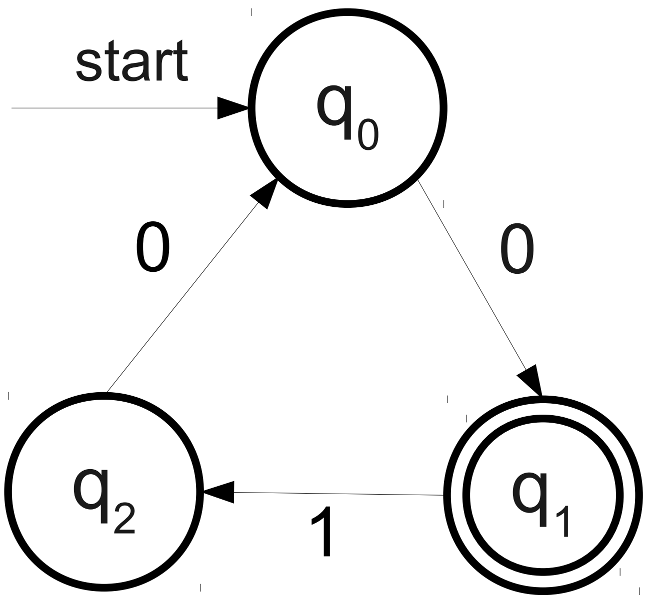
The **language of an automaton** is the set of strings that it accepts.

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

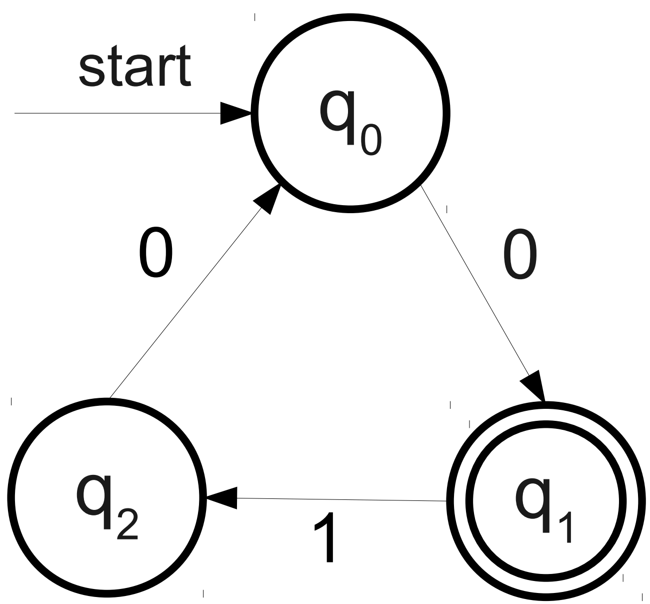
Intuitively:

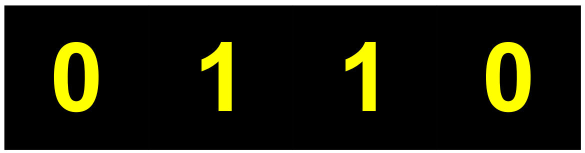
$$\mathscr{L}(A) = \{\ w \in \Sigma^* \mid A \text{ accepts } w\ \}$$

A Small Problem

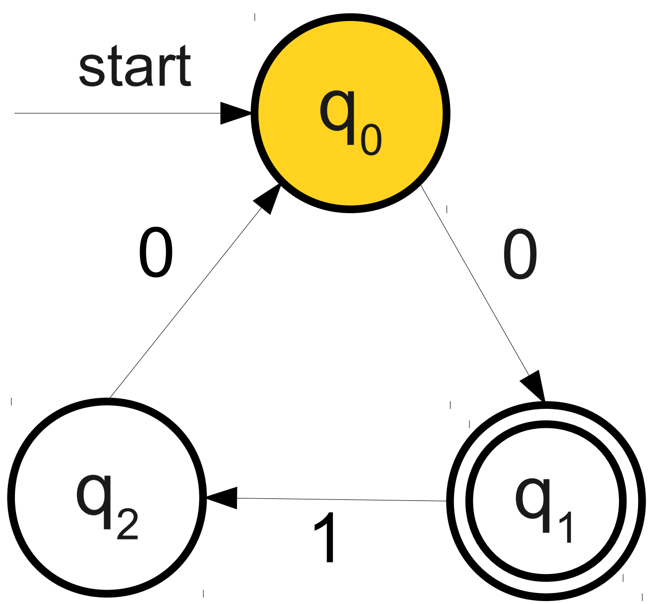


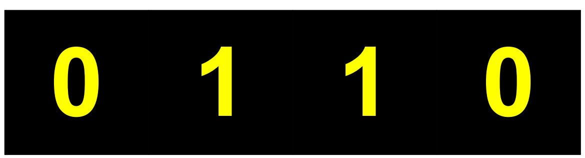
A Small Problem



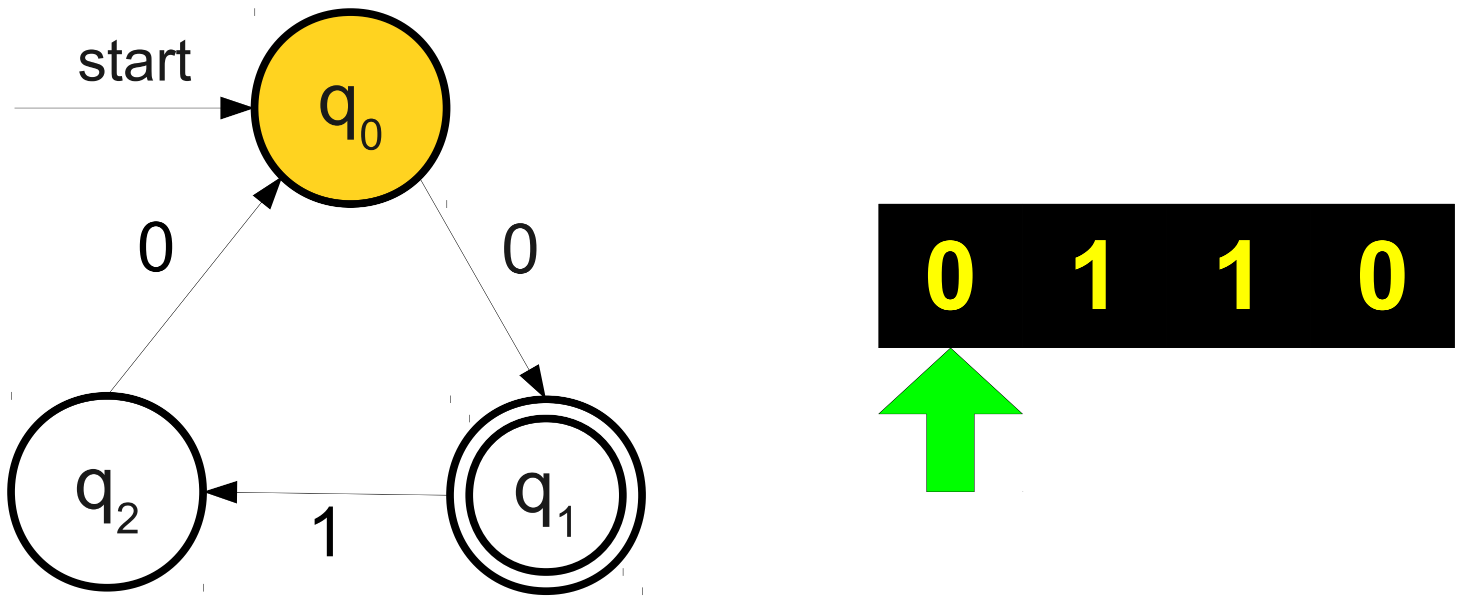


A Small Problem

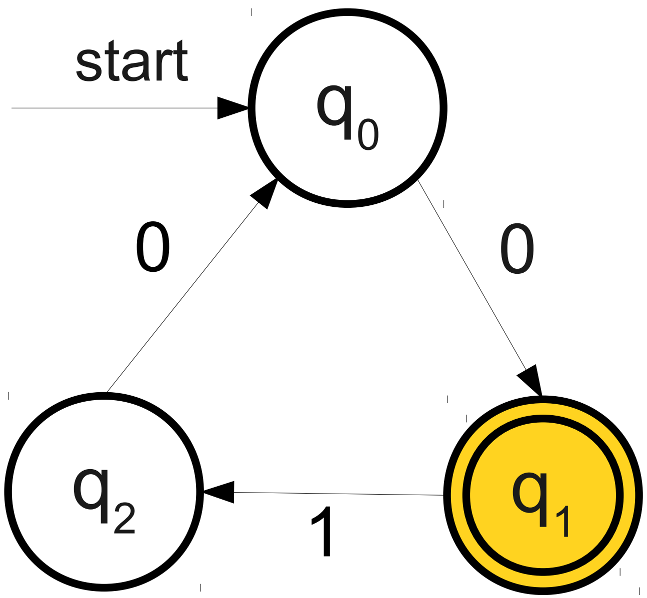




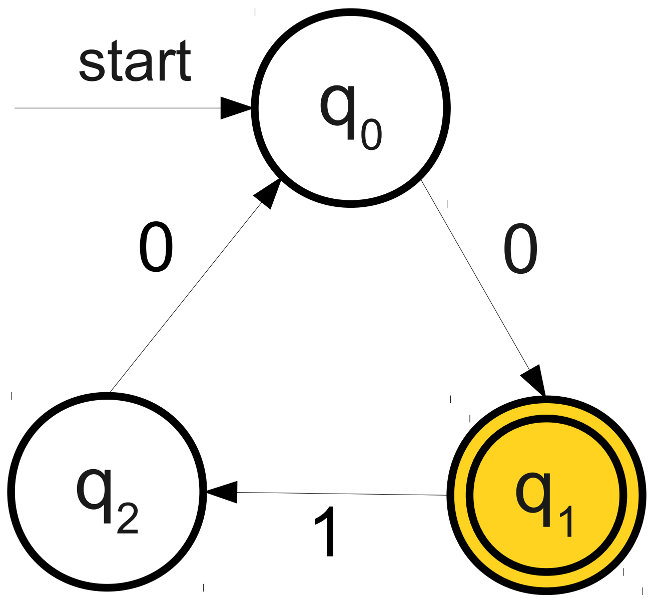
A Small Problem

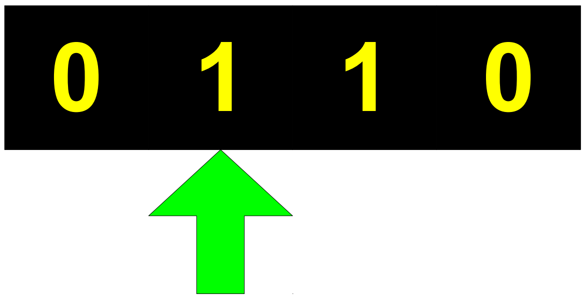


A Small Problem

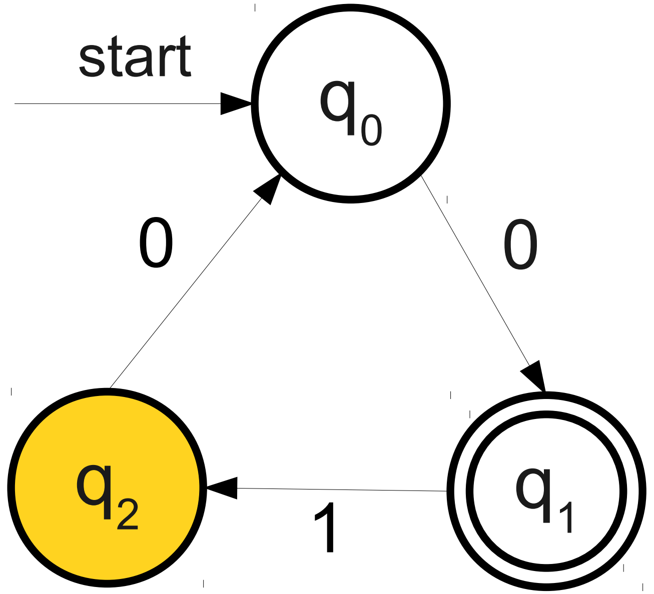


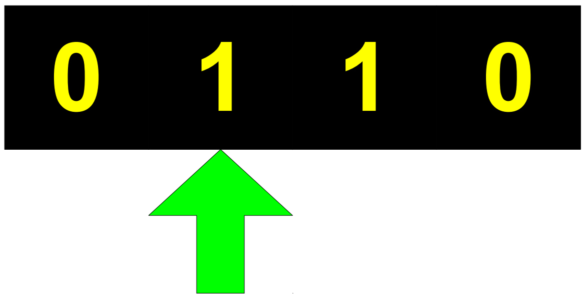
A Small Problem



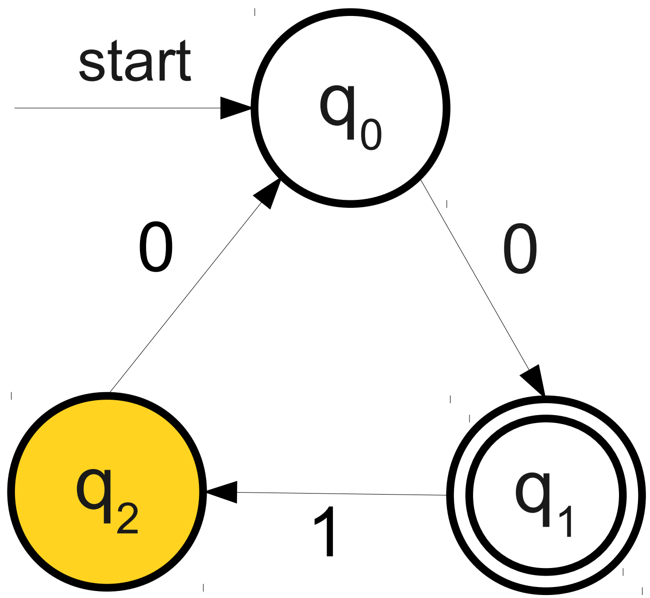


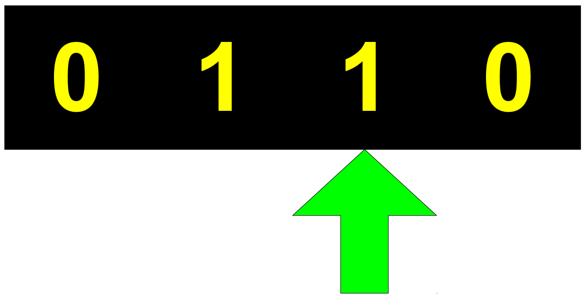
A Small Problem



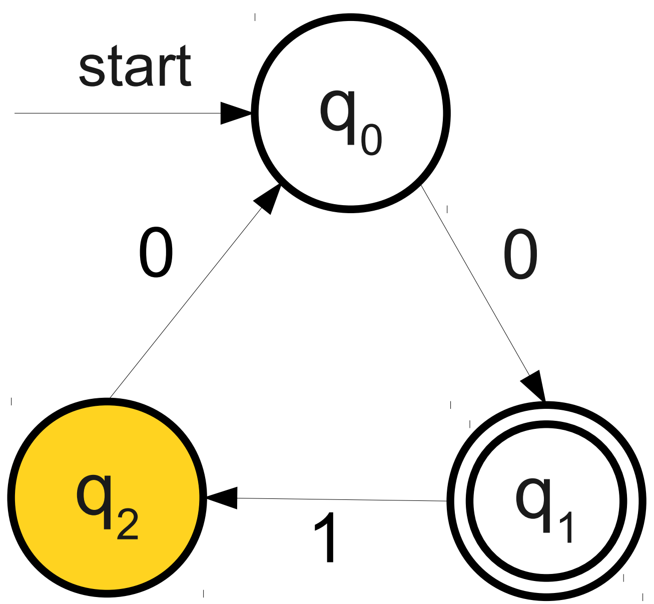


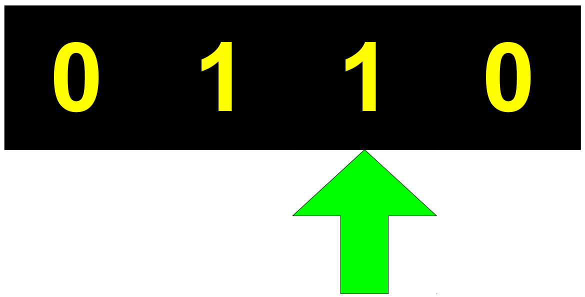
A Small Problem

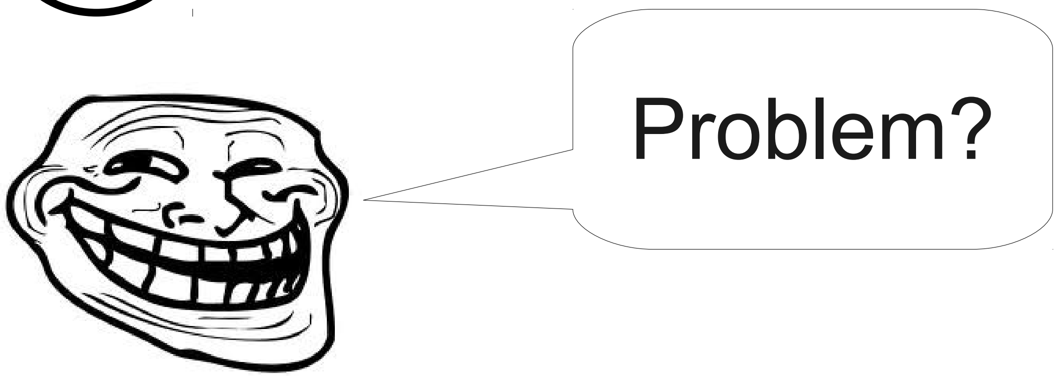




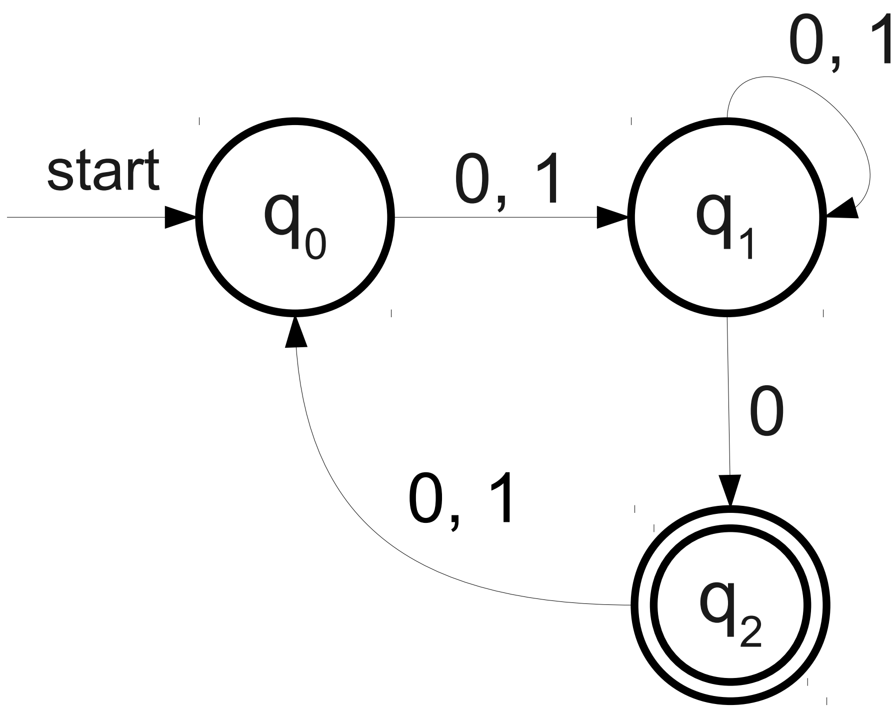
A Small Problem



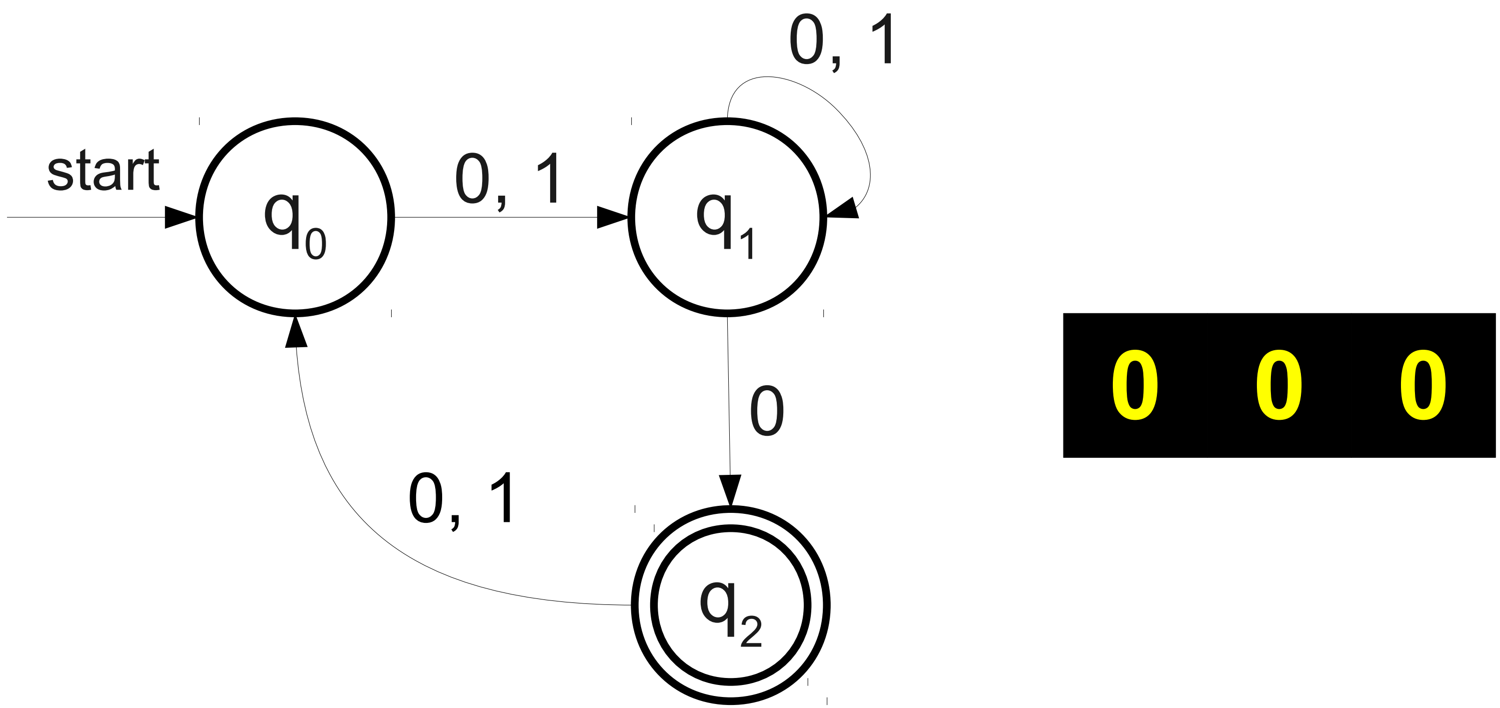




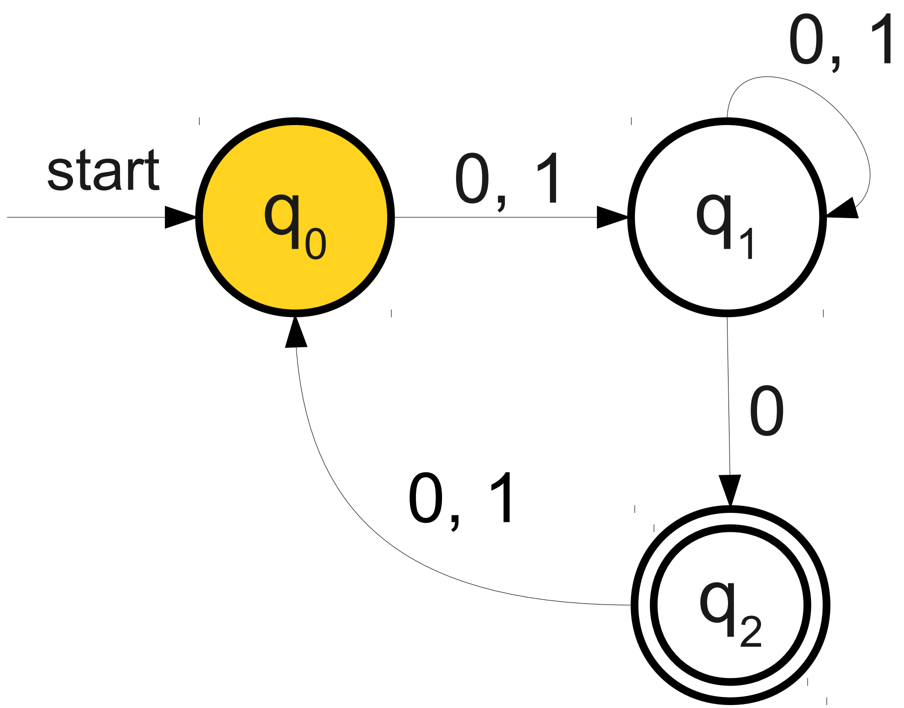
Another Small Problem

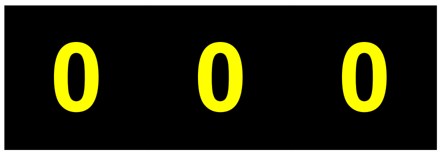


Another Small Problem

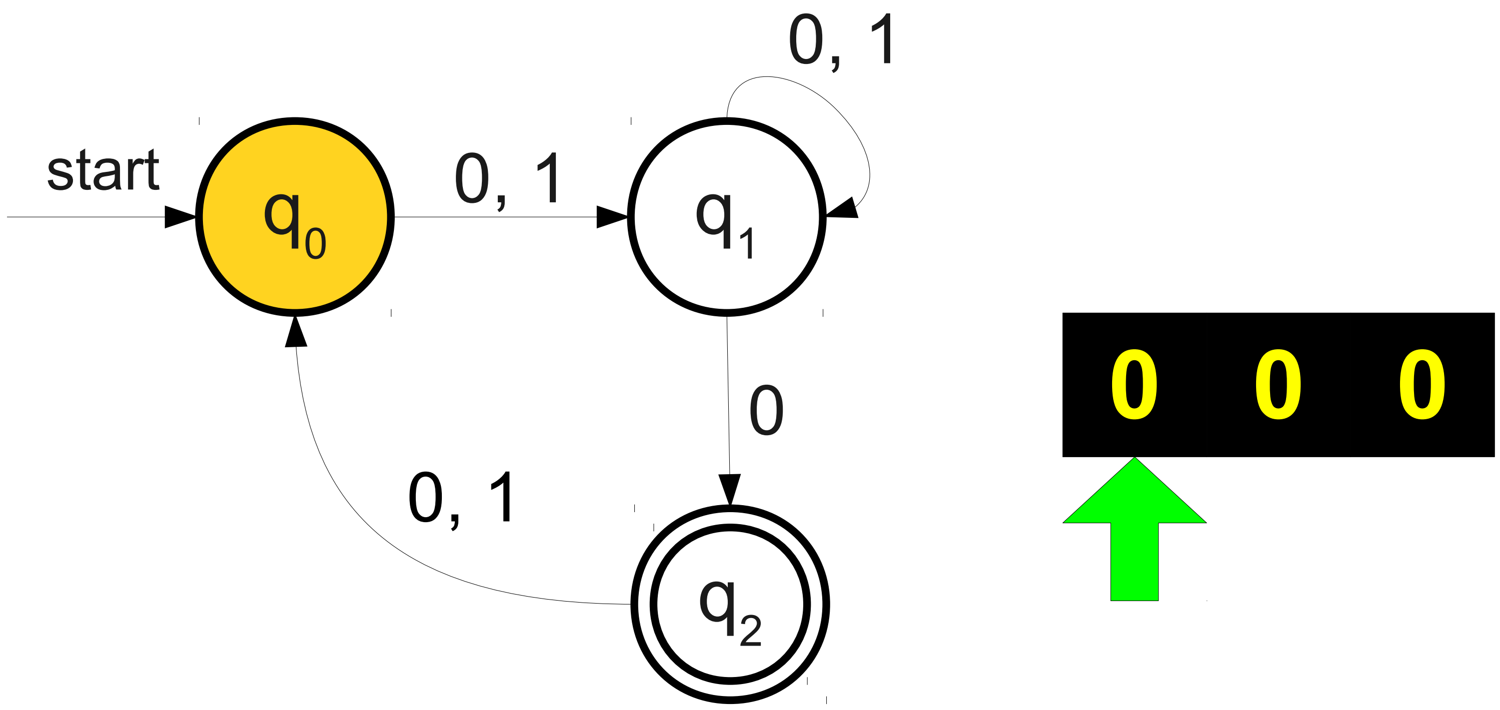


Another Small Problem

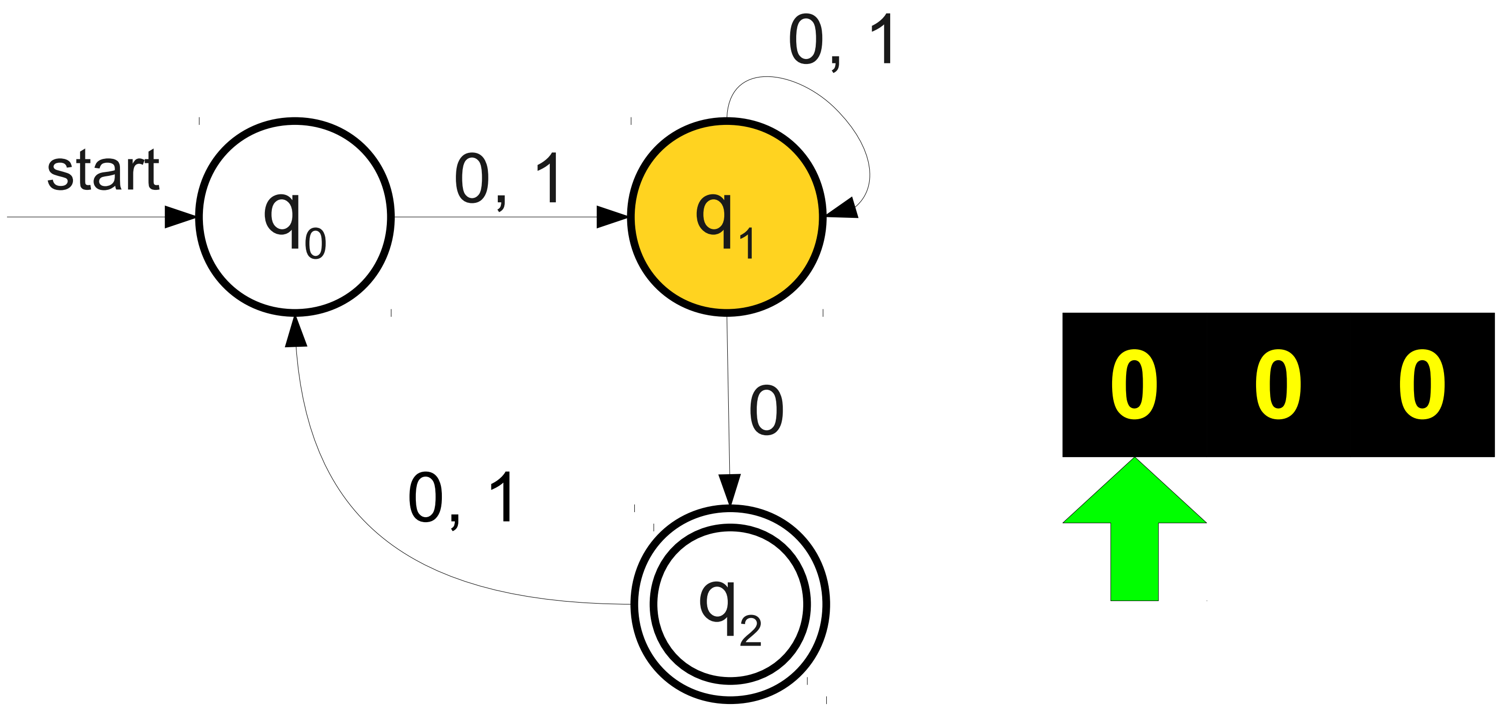




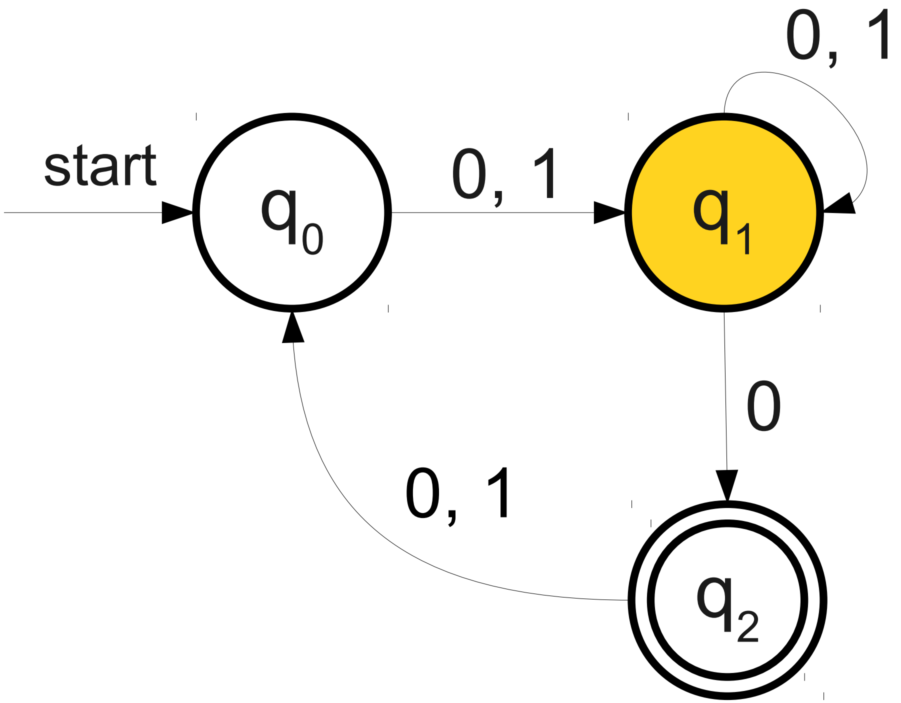
Another Small Problem

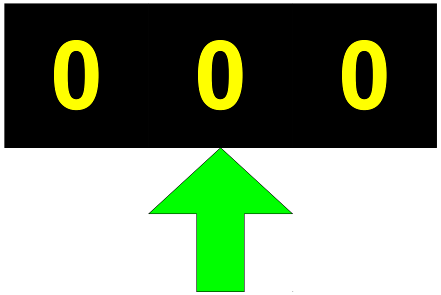


Another Small Problem

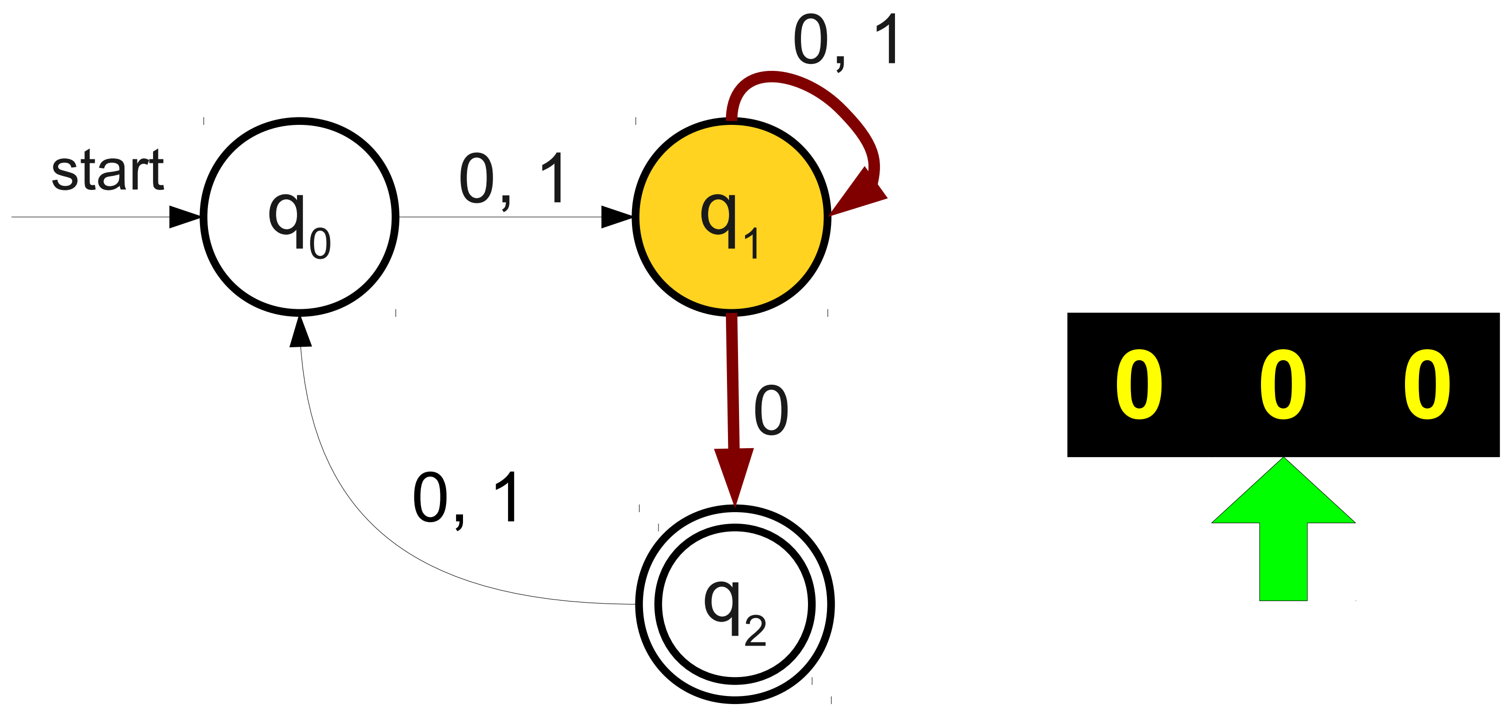


Another Small Problem

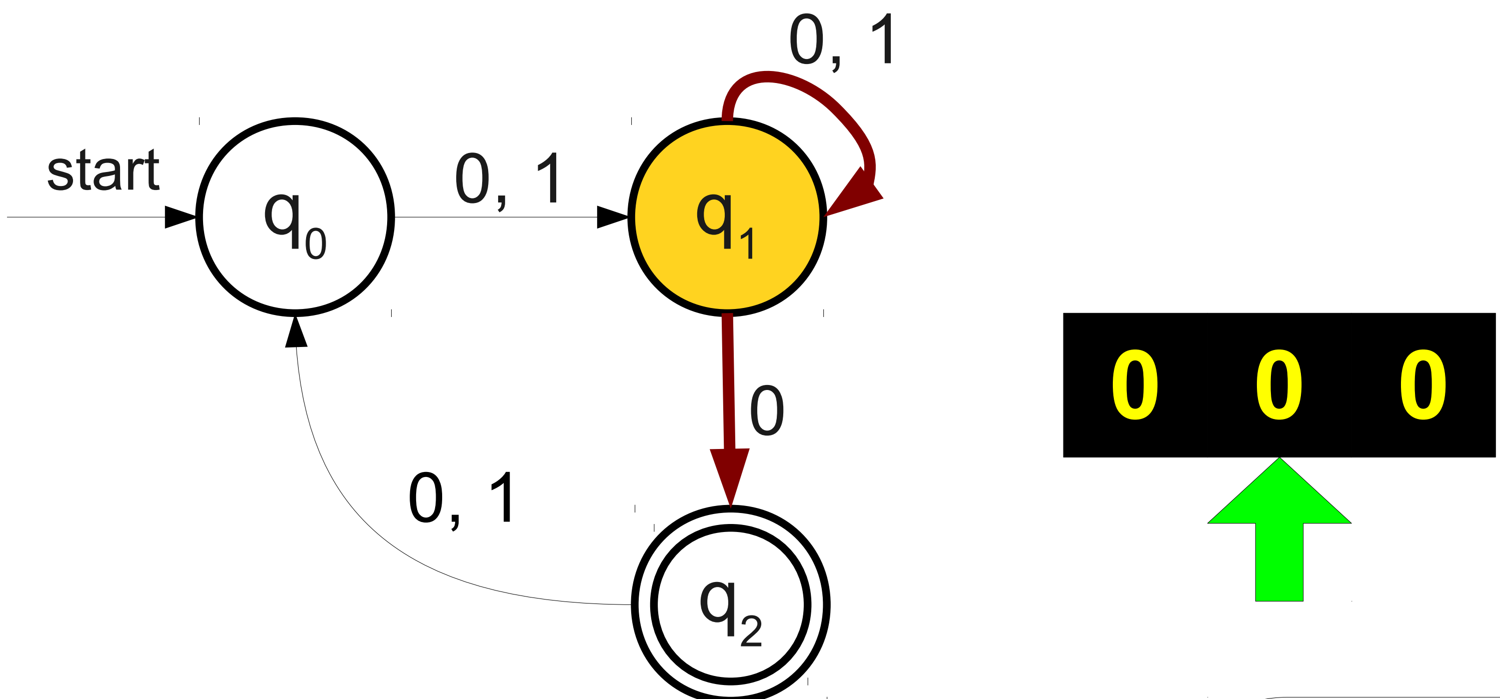


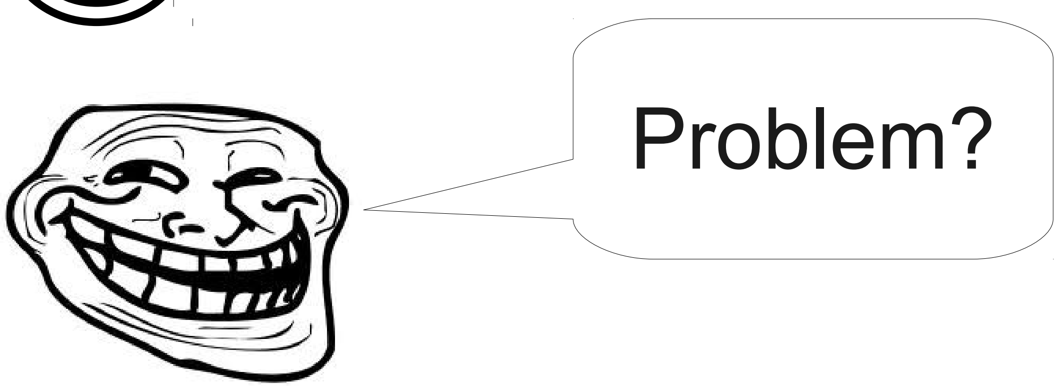


Another Small Problem



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

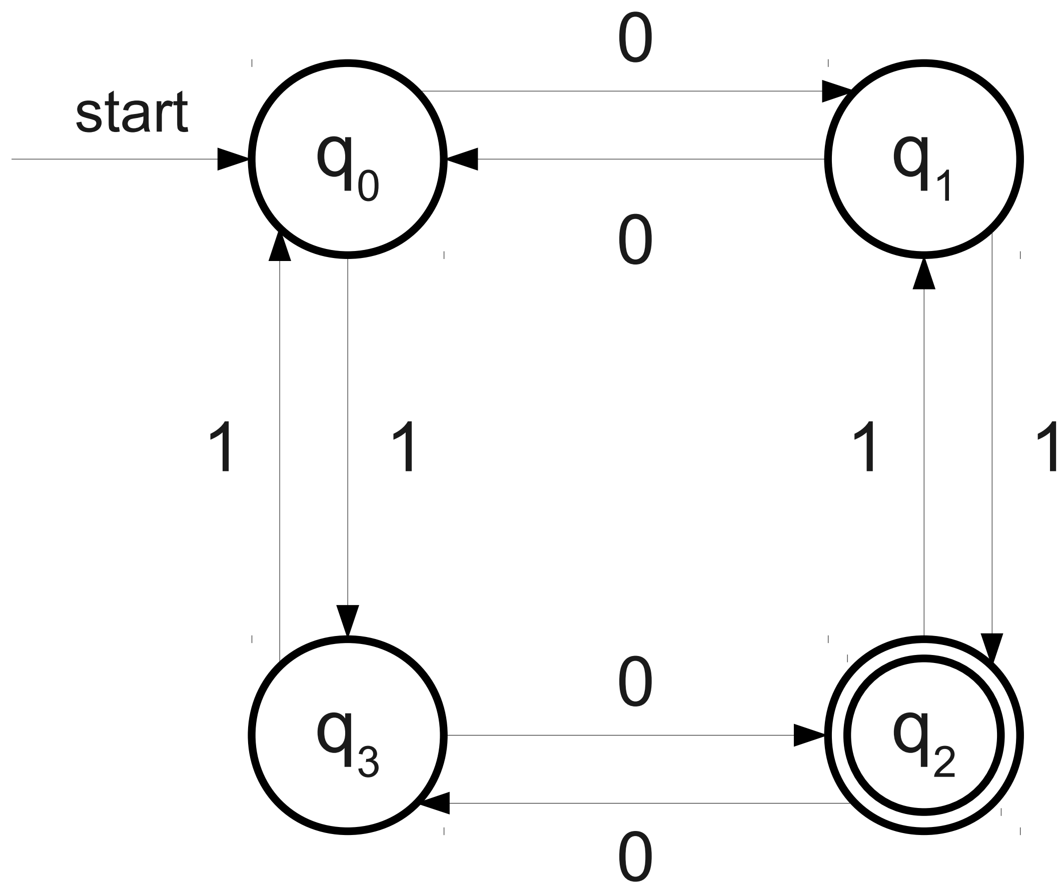
- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- DFAs are the simplest type of automaton that we will see in this course.

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 the alphabet.
  - This is the "deterministic" part of DFA.
- There is a **unique** start state.
- There may be multiple accepting states.

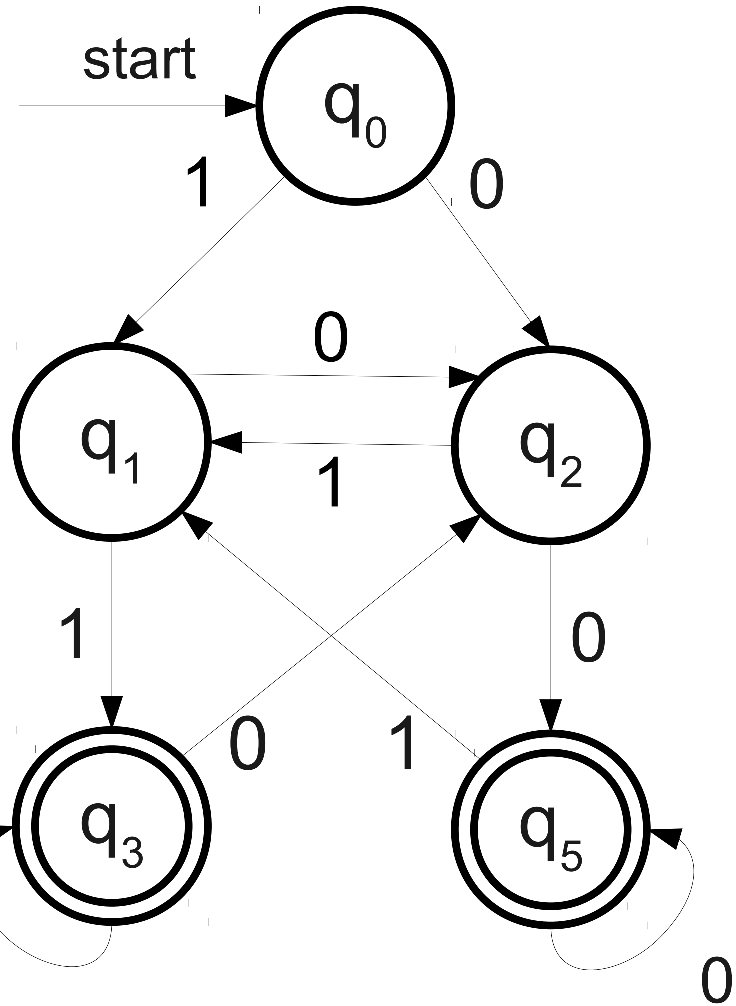
Is this a DFA?

Is this a DFA?



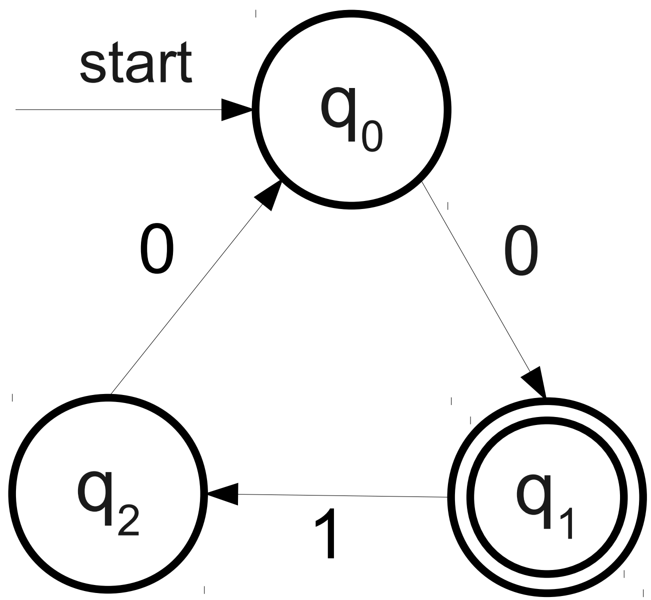
Is this a DFA?

Is this a DFA?

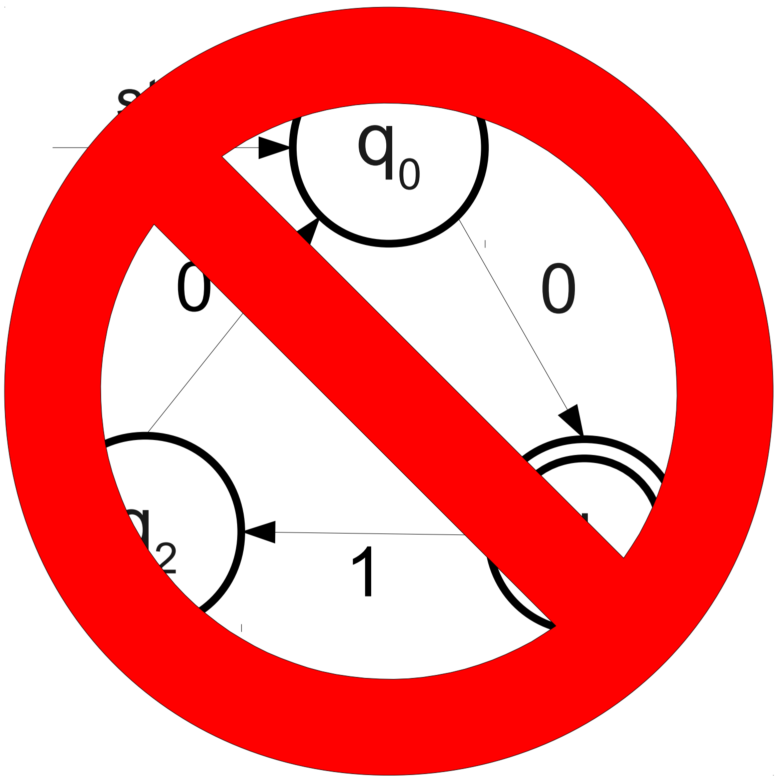


Is this a DFA?

Is this a DFA?

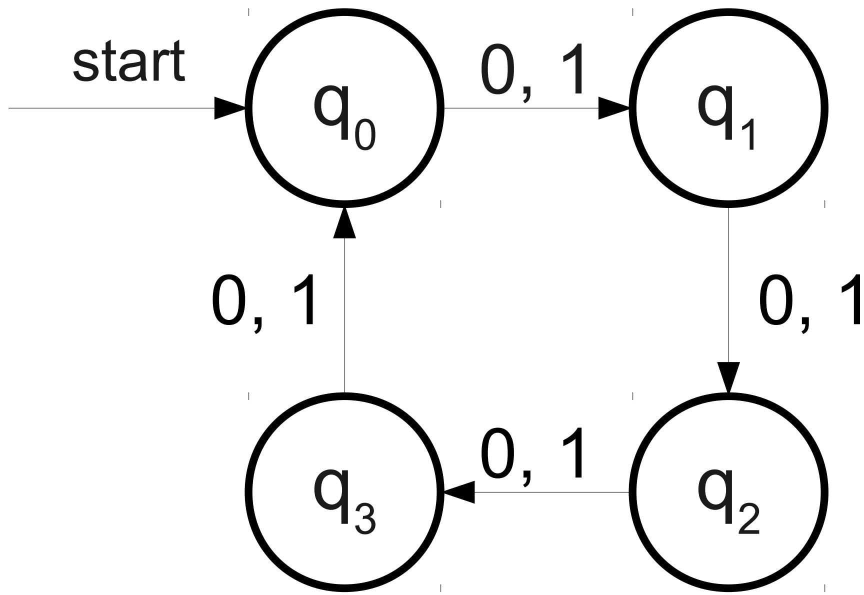


Is this a DFA?



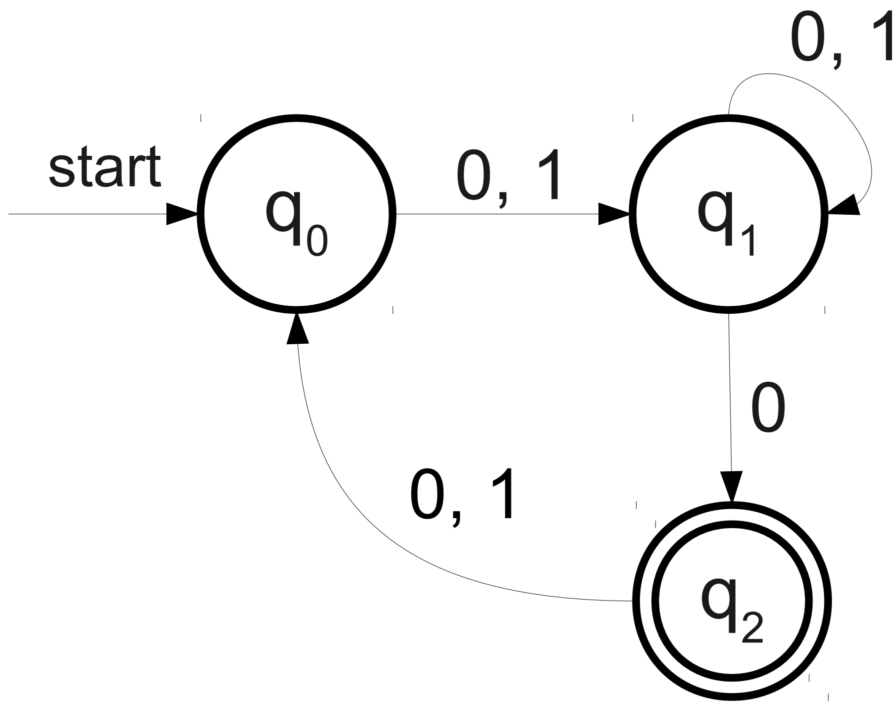
Is this a DFA?

Is this a DFA?

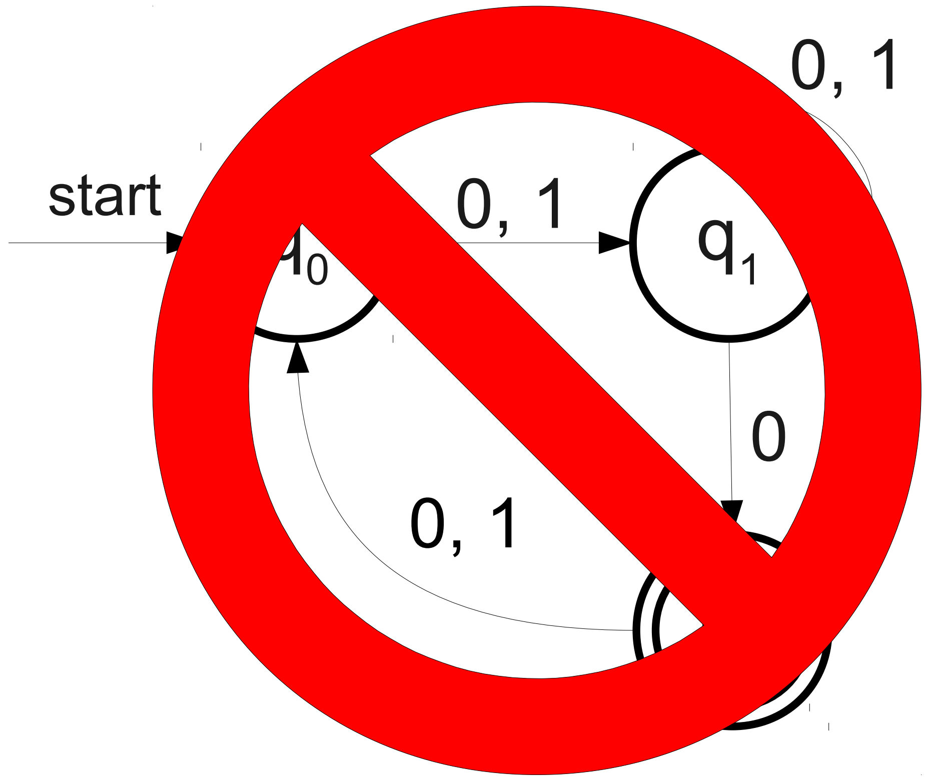


Is this a DFA?

Is this a DFA?



Is this a DFA?



Is this a DFA?

Is this a DFA?

Is this a DFA?

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

Designing DFAs

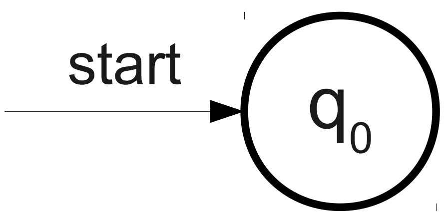
- At each point in its execution, the DFA can only remember what state it is in.
- A good way to design DFAs is to think about what information you would need to pick up where you left off.
  - Each state acts as a “memento” of what you're supposed to do next.

Recognizing Languages with DFAs

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

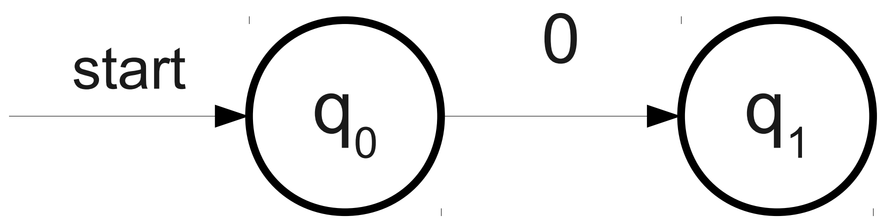
Recognizing Languages with DFAs

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



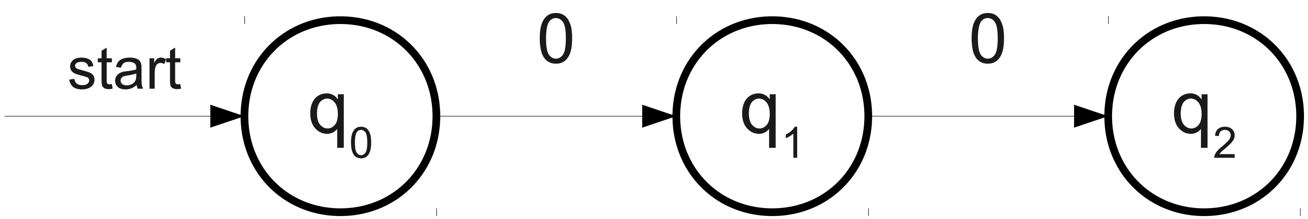
Recognizing Languages with DFAs

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



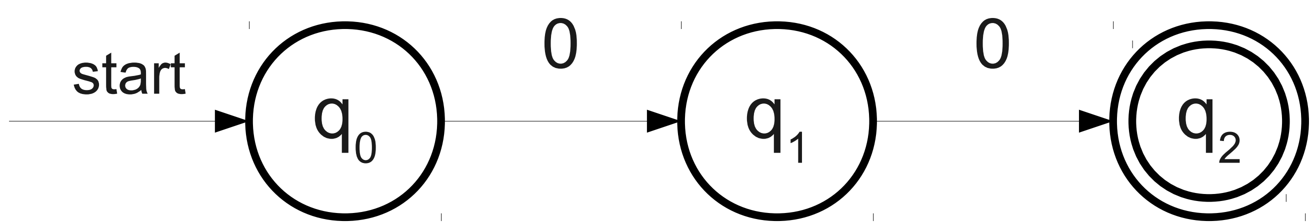
Recognizing Languages with DFAs

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



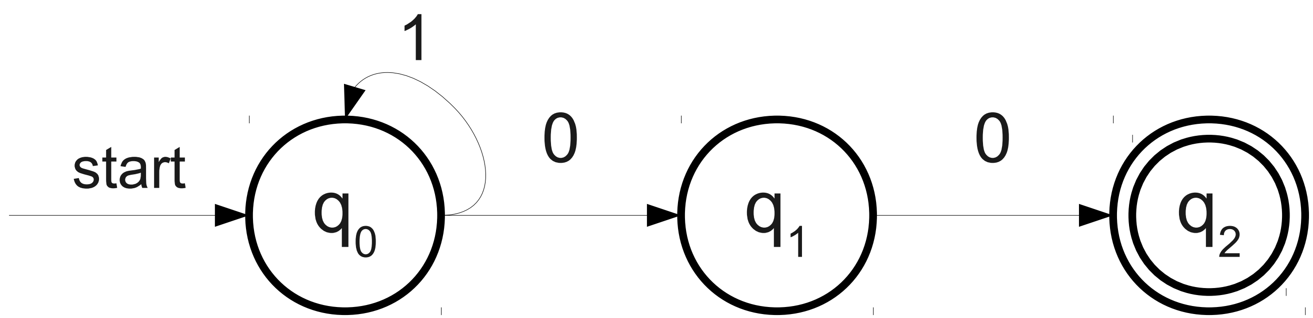
Recognizing Languages with DFAs

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



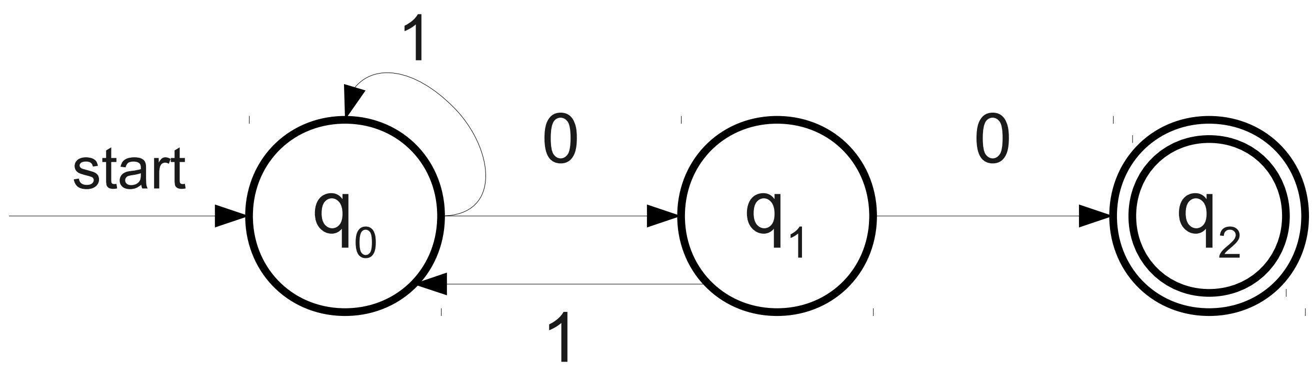
Recognizing Languages with DFAs

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



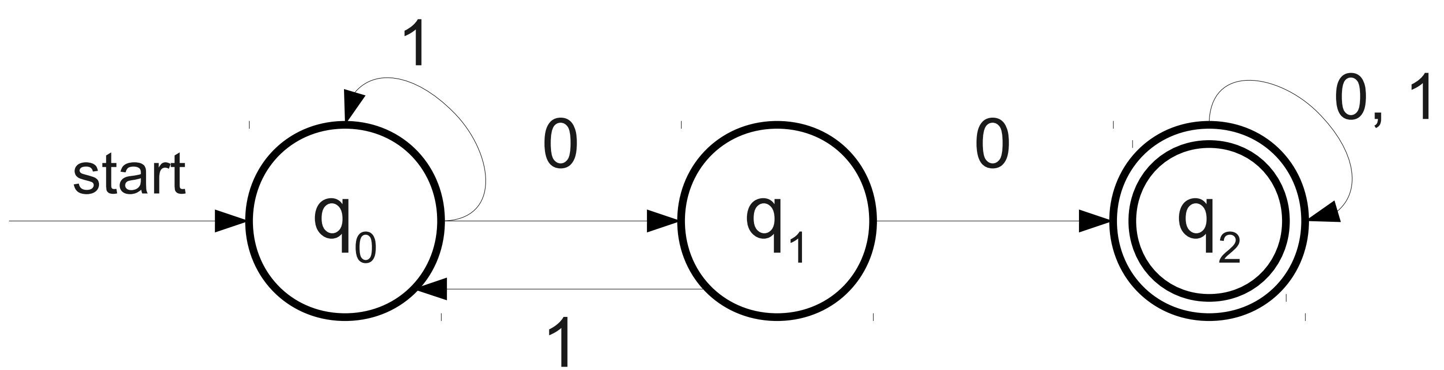
Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

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



Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid \text{all even-numbered digits of } w \text{ are } 0 \}$

Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* |$ all even-numbered digits of w are $0 \}$

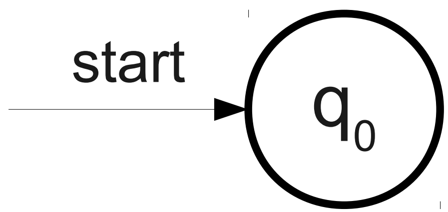
YES	**NO**
01	**1**
00**0**1	**0**0**1**
01**0**1**0**1**0**0**0**1	**0**0**0**0**1**

Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid \text{all even-numbered digits of } w \text{ are } 0 \}$

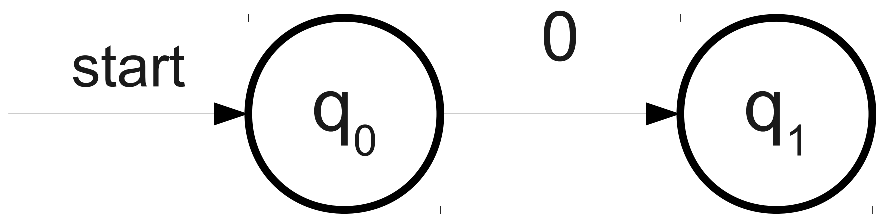
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid \text{all even-numbered digits of } w \text{ are } 0 \}$



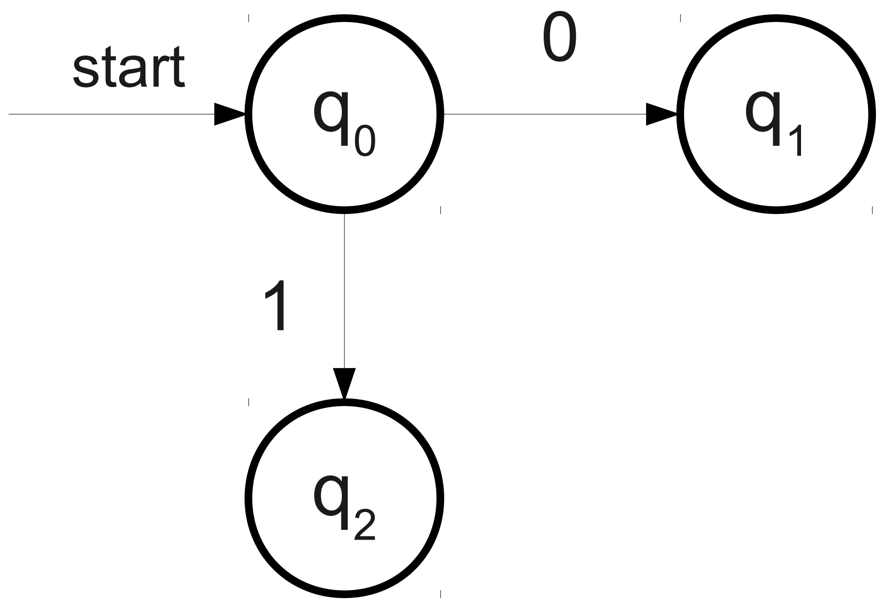
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid$ all even-numbered digits of w are 0 $\}$



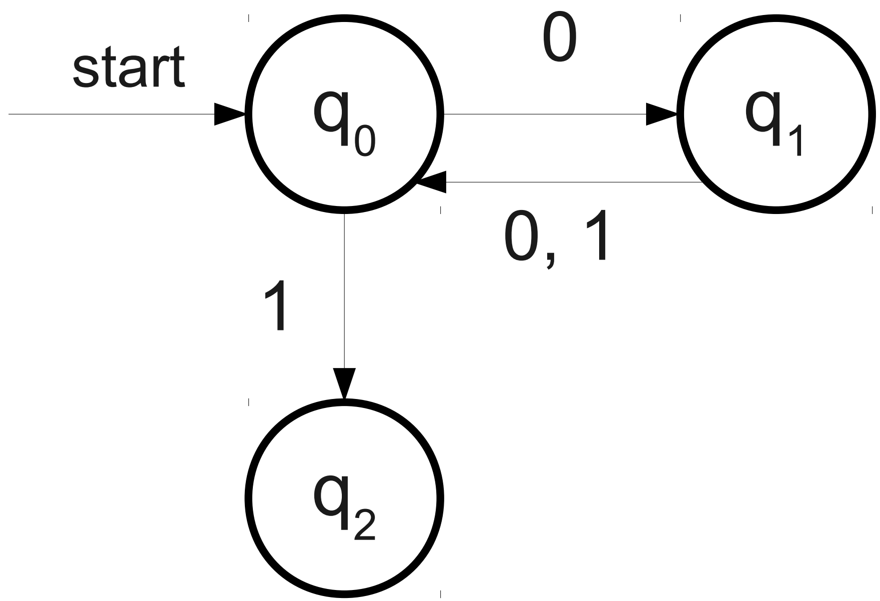
Recognizing Languages with DFAs

$L = \{\ w \in \{0, 1\}^*|$ all even-numbered digits of w are 0 $\}$



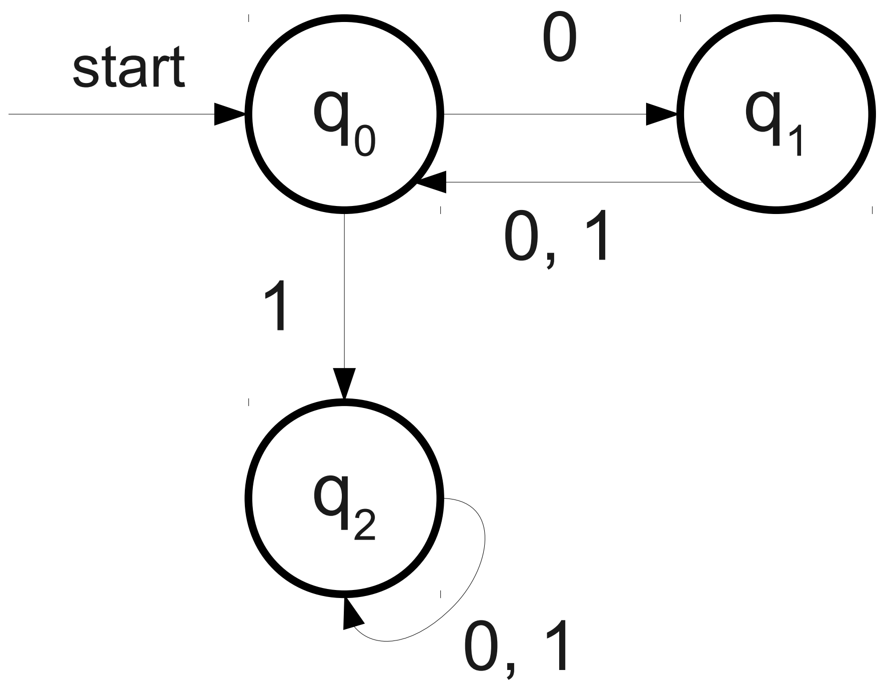
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid \text{all even-numbered digits of } w \text{ are } 0 \}$



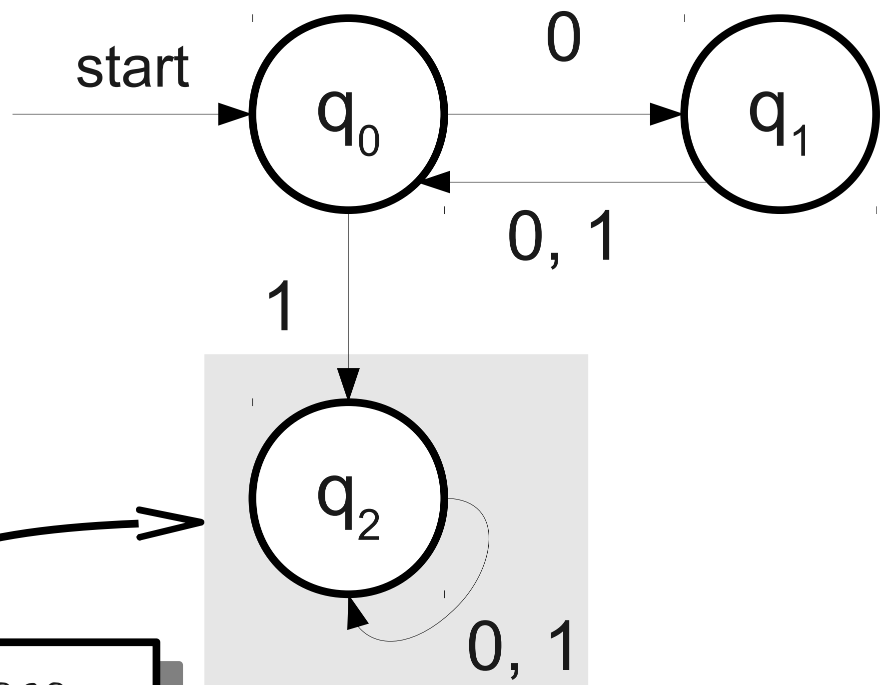
Recognizing Languages with DFAs

$L = \{\ w \in \{0, 1\}^*|$ all even-numbered digits of w are 0 $\}$



Recognizing Languages with DFAs

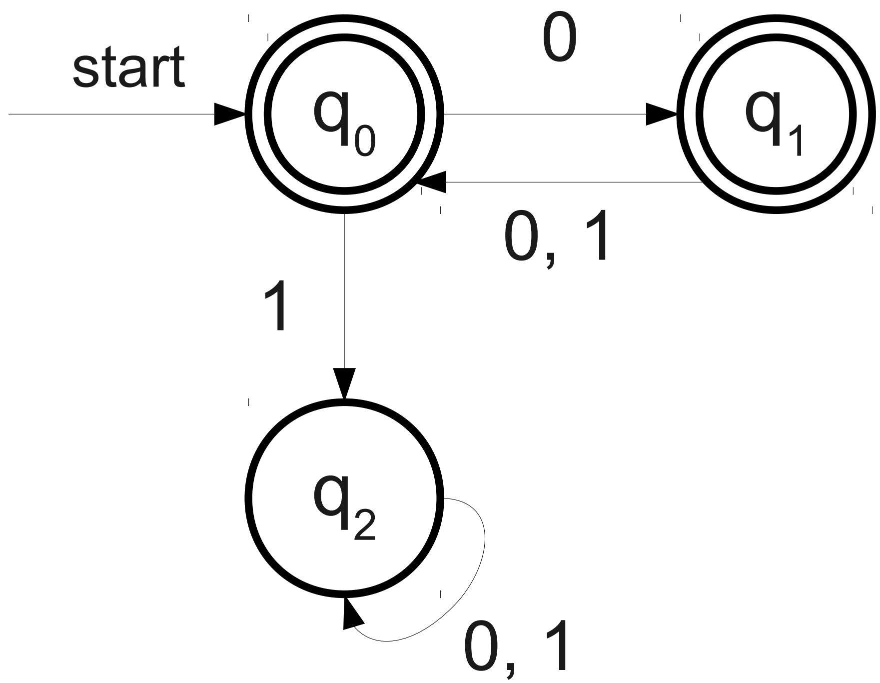
$L = \{ w \in \{0, 1\}^* |$ all even-numbered digits of w are 0 $\}$



States like these are called **dead states**.

Recognizing Languages with DFAs

$L = \{ \ w \in \{0, 1\}^* | \text{ all even-numbered digits of } w \text{ are } 0 \ \}$



More Elaborate DFAs

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

Suppose the alphabet is

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

Try designing a DFA for comments!

Some test cases:

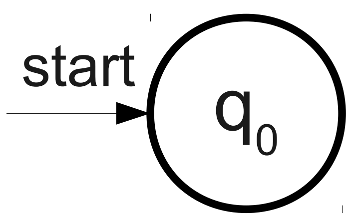
ACCEPTED	REJECTED
`/*a*/`	`/**`
`/**/`	`/**/a`
`/***/`	`aaa/**/`
`/*aaa*aaa*/`	`/*/`

More Elaborate DFAs

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

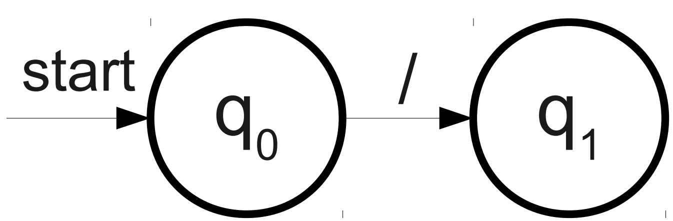
More Elaborate DFAs

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



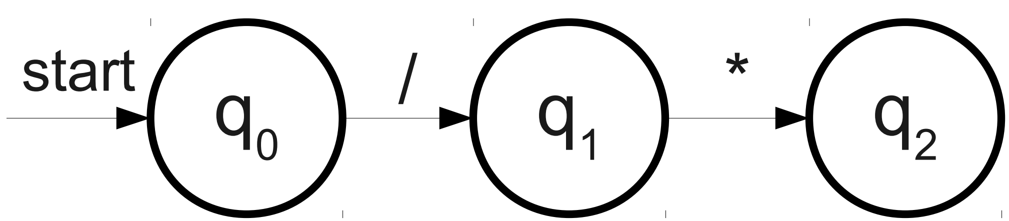
More Elaborate DFAs

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



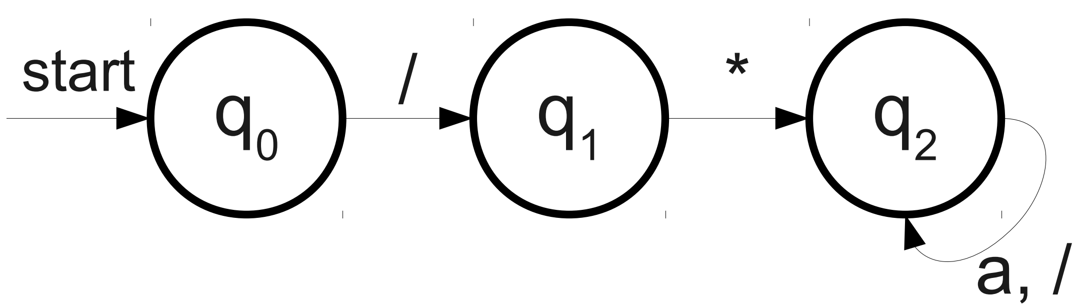
More Elaborate DFAs

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



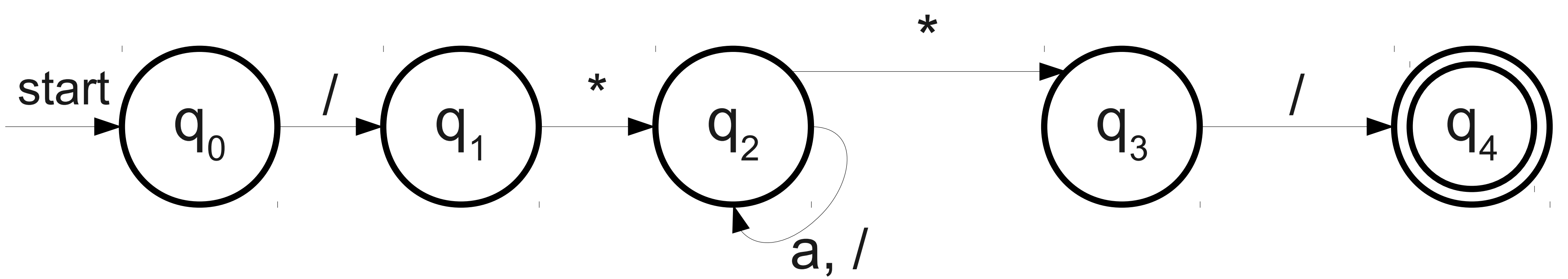
More Elaborate DFAs

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



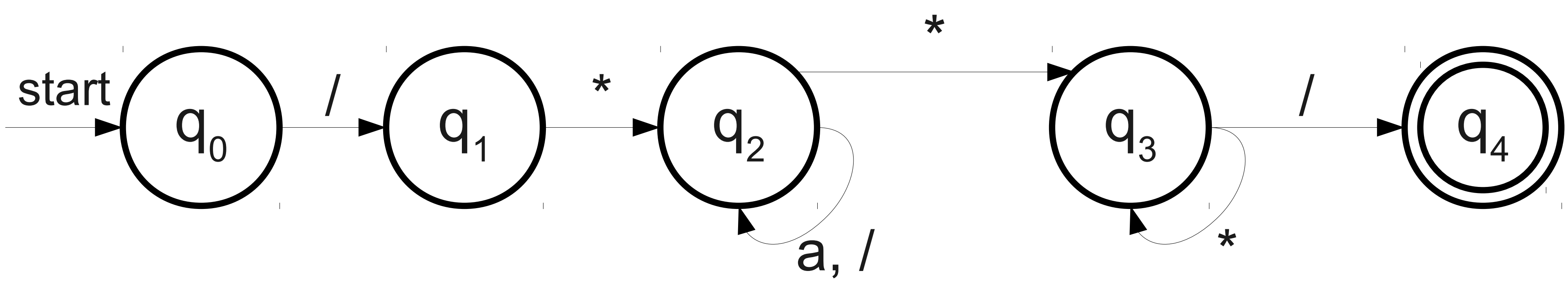
More Elaborate DFAs

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



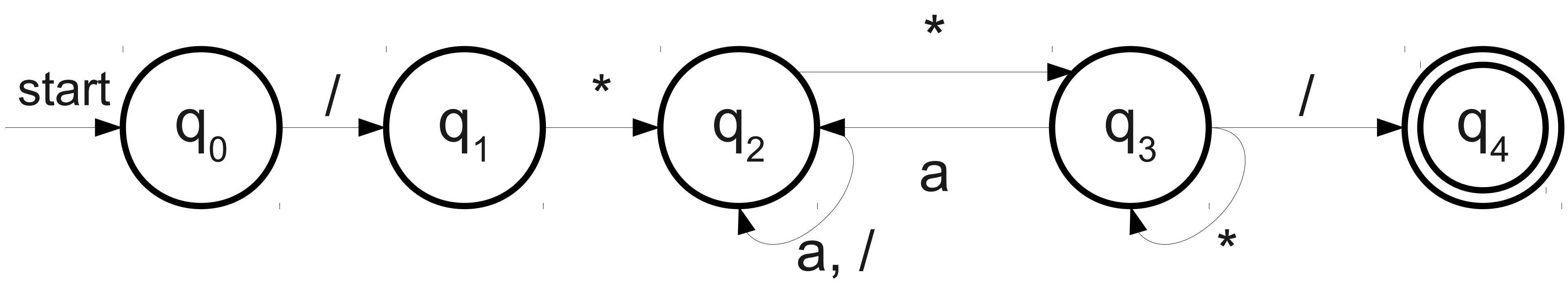
More Elaborate DFAs

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



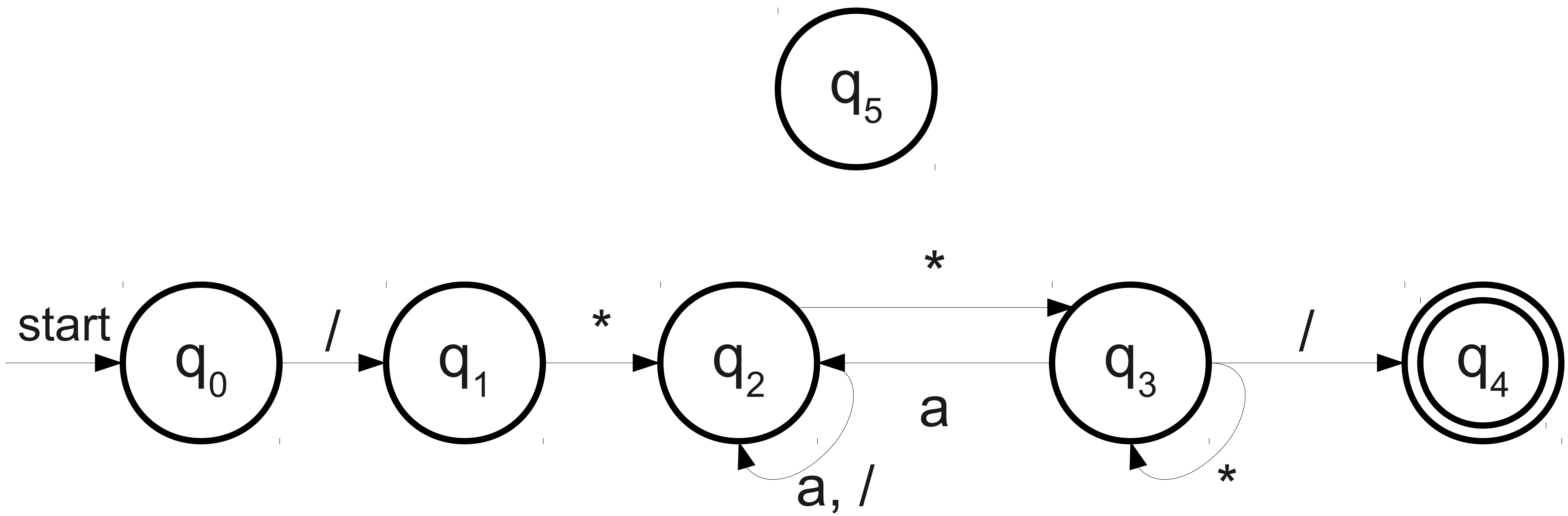
More Elaborate DFAs

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



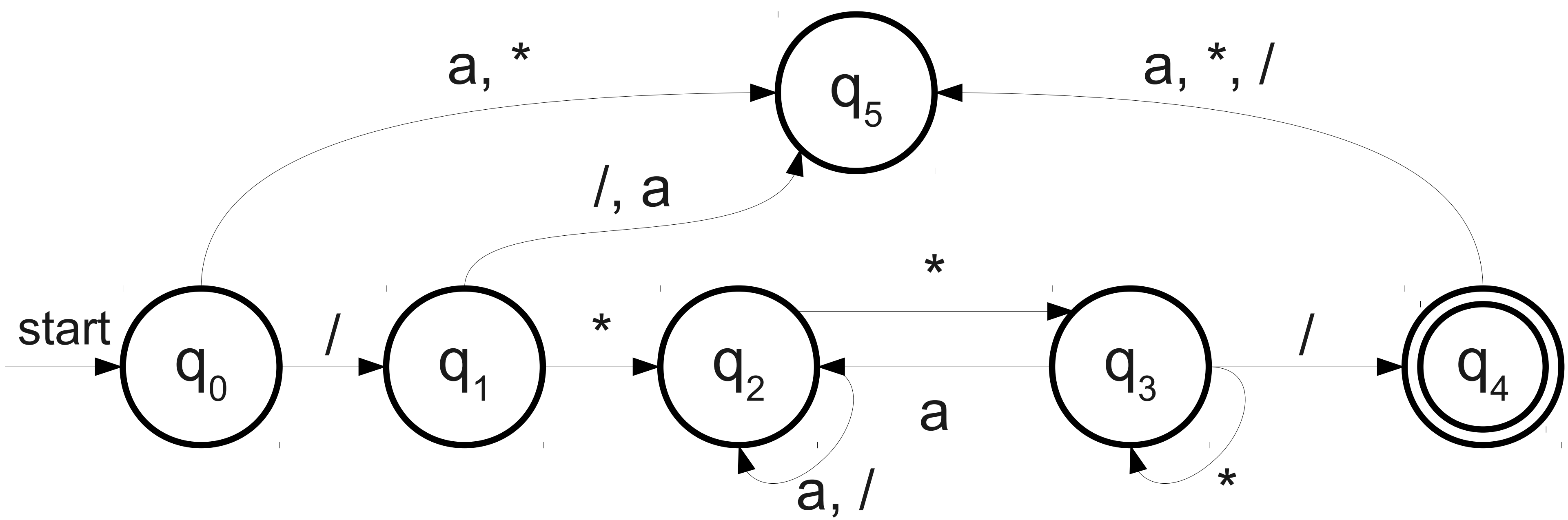
More Elaborate DFAs

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



More Elaborate DFAs

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



More Elaborate DFAs

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