CMPU 240 · Theory of Computation

# Languages and Automata

27 January 2026

Reminder: You can ask questions on Ed (including logistic questions).

The first assignment will come out on Thursday and be due on Tuesday.

I'll talk more about assignments on Thursday, but I've posted the Guide to Assignments on the course website, which gives information you can look at.

## Where are we?

What problems can we solve with a computer?

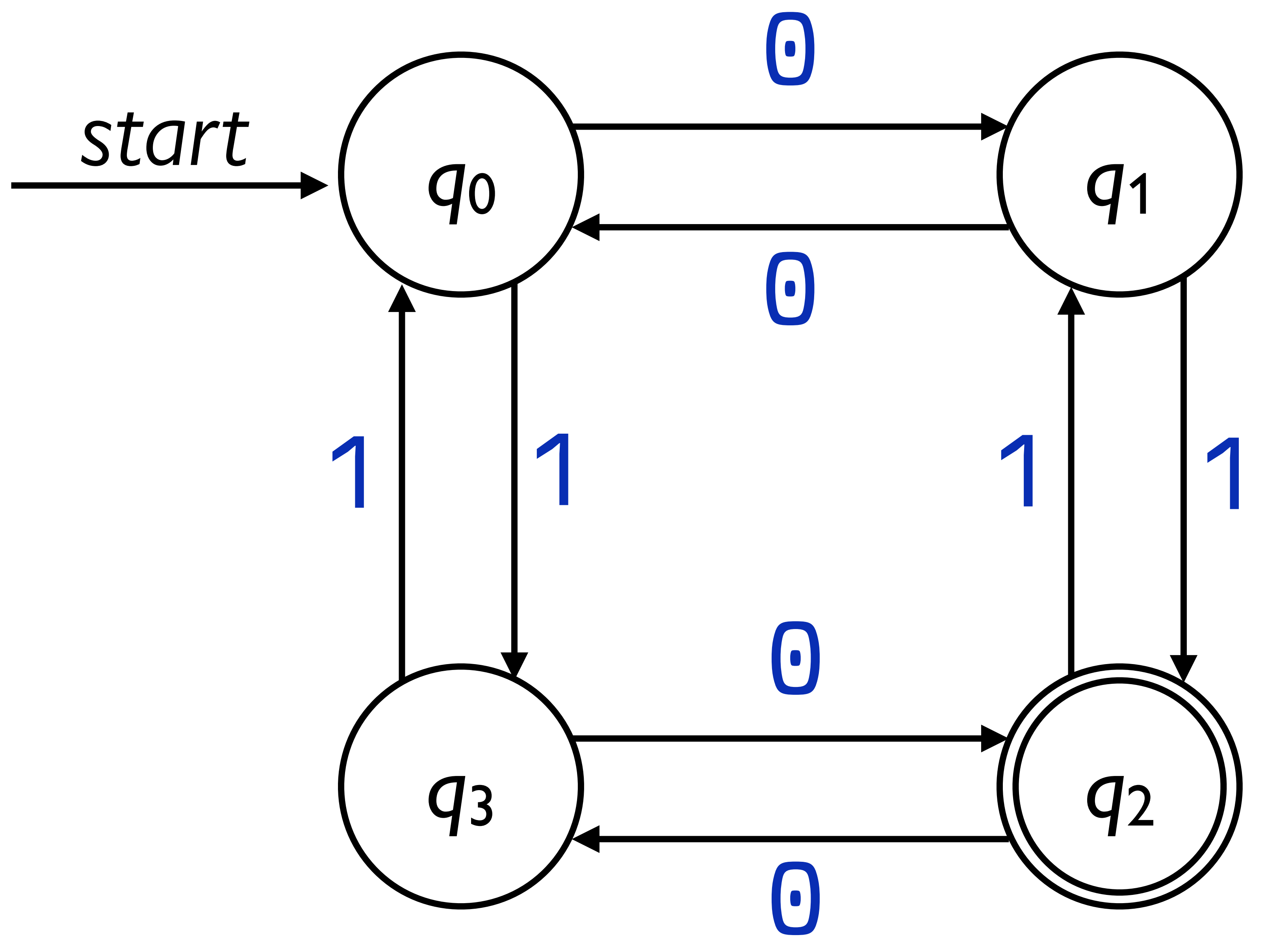
What problems can we solve with a computer?

*What kind of computer?*

## Enter automata

An *automaton* is a mathematical model of a computing device.

It's an abstraction of a real computer, like how graphs are abstractions of social networks, transportation grids, etc.


start
q0
q1
q2
q3
0
0
0
0
1
1
1
1

$$\begin{tikzcd}[column sep=large]
{} \arrow[r, "start"] & \text{0¢} \arrow[r, "nickel"] \arrow[rr, bend left, "dime"] \arrow[rrrrr, bend right, "quarter"'] & \text{5¢} \arrow[r, "nickel"] \arrow[rr, bend left, "dime"] \arrow[rrrr, bend left=60, "quarter"] & \text{10¢} \arrow[r, "nickel"] \arrow[rr, bend left, "dime"] \arrow[rrr, bend left=50, "quarter"] & \text{15¢} \arrow[r, "nickel"] \arrow[rr, bend left, "{dime, quarter}"] & \text{20¢} \arrow[r, "{nickel, dime, quarter}"] & \text{25¢} \arrow[loop below, "{nickel, dime, quarter}"]
\end{tikzcd}$$

## Language theory

What problems can we solve with a computer?

*What's a "problem"?*

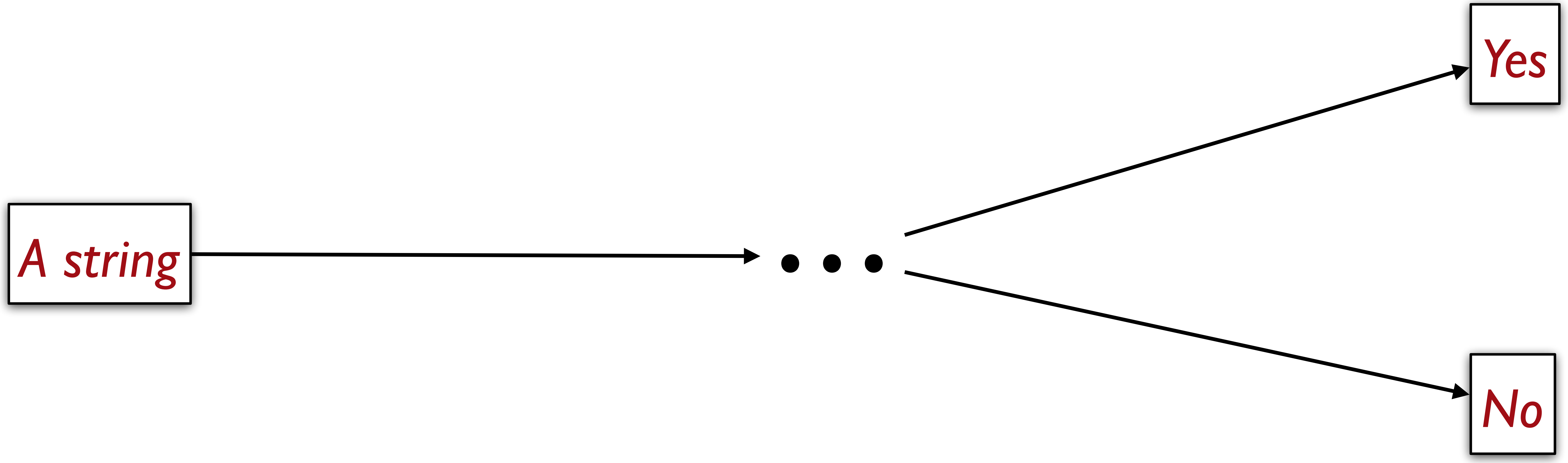
Before we can talk about what problems we can solve, we need a formal definition of a "problem".

We want a definition that

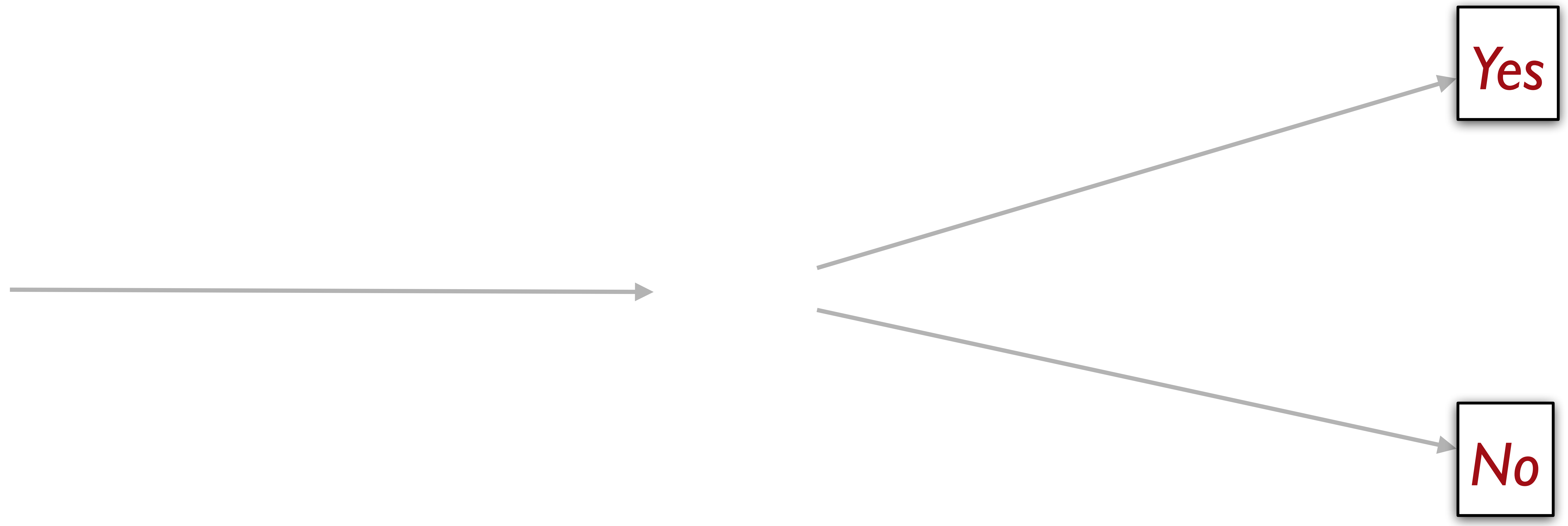
corresponds to the problems we want to solve,

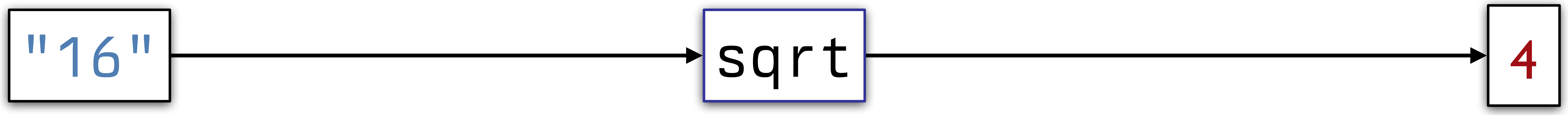
captures a large class of problems, and

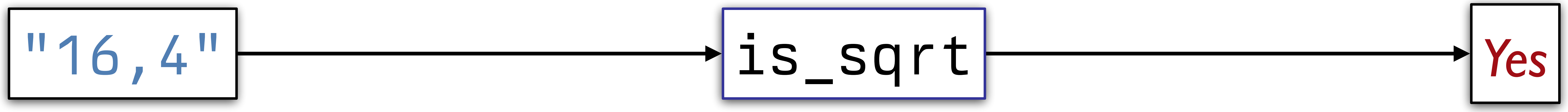
is mathematically simple to reason about


A string
• • •
Yes
No

"0100111010101010..."


Yes
No


"16"
sqrt
4


"16,4"
is_sqrt
Yes

$$\begin{tikzcd}\text{sqrt}(16) = 4 \arrow[d] \\ \text{is\_sqrt}(16, 4) = \text{Yes}\end{tikzcd}$$

is_sqrt(16, 3) = No

$$\begin{tikzcd}f(x) = y \arrow[d] \\ f'(x, y) = \text{Yes}\end{tikzcd}$$

Virtually all computational problems can be recast as *language recognition problems*.

For example, the problem of determining whether an integer is prime:

> *Problem*: Is 97 prime?
>
> *Recast*: Is the string 97 in the language of all primes, {2, 3, 5, 7, 13, …}?

## Formalizing things

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

$\emptyset$

$\{0\}$

$\{0, 1\}$

$\mathbb{N}$

$\mathbb{N}_0$

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

| | |
|---|---|
| $\varnothing$ | 0 objects – the ***empty set*** |
| $\{0\}$ | |
| $\{0, 1\}$ | |
| $\mathbb{N}$ | |
| $\mathbb{N}_0$ | |

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

| | |
|---|---|
| $\varnothing$ | 0 objects – the ***empty set*** |
| $\{0\}$ | 1 object – the set containing the number 0 |
| $\{0, 1\}$ | |
| $\mathbb{N}$ | |
| $\mathbb{N}_0$ | |

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

| | |
|---|---|
| $\varnothing$ | 0 objects – the ***empty set*** |
| $\{0\}$ | 1 object – the set containing the number 0 |
| $\{0, 1\}$ | 2 objects – the set containing the numbers 0 and 1 |
| $\mathbb{N}$ | |
| $\mathbb{N}_0$ | |

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

|  |  |
|---|---|
| $\varnothing$ | 0 objects – the ***empty set*** |
| $\{0\}$ | 1 object – the set containing the number 0 |
| $\{0, 1\}$ | 2 objects – the set containing the numbers 0 and 1 |
| $\mathbb{N}$ | infinite objects – the set of all natural numbers |
| $\mathbb{N}_0$ |  |

Recall that a ***set*** is an unordered collection of zero or more distinct objects of any type, e.g.,

| | |
|---|---|
| $\varnothing$ | 0 objects – the ***empty set*** |
| $\{0\}$ | 1 object – the set containing the number 0 |
| $\{0, 1\}$ | 2 objects – the set containing the numbers 0 and 1 |
| $\mathbb{N}$ | infinite objects – the set of all natural numbers |
| $\mathbb{N}_0$ | infinite objects – the set of all natural numbers including 0 |

An *alphabet*, denoted Σ, is a finite, non-empty set of symbols called *characters*, e.g.,

*Binary*: Σ = {`0`, `1`}

*ASCII*: Σ = {`a`, `b`, `c`, …, `0`, `1`, …, `!`, `@`, `#`, …}

A *string* over an alphabet Σ is a finite sequence of characters drawn from Σ.

For example, if Σ = {a, b}, then

    a

and

    abbaba

are strings over Σ.

A ***string*** over an alphabet Σ is a $\boxed{\text{finite}}$ sequence of characters drawn from Σ.

*This is important! You cannot have a string of infinite length!*

For example, if Σ = {a, b}, then

    a

and

    abbaba

are strings over Σ.

In a programming language, a string is written with quotation marks, e.g., `"sleepy otters"`.

In a programming language, a string is written with quotation marks, e.g., `"sleepy otters"`.

In language theory, we omit the quotation marks and use visible characters for any spaces, e.g., `sleepy␣otters`.

If you’re programming, what’s `""`?

If you're programming, what's `""`?

It's a string of length 0, called the *empty string*.

If you're programming, what's `""`?

It's a string of length 0, called the *empty string*.

Because we don't use quotation marks in theory, we write it as $\varepsilon$ (epsilon).

A set of strings is called a *language*.

We say that $L$ is a *language over* $\Sigma$ if it is a set of strings over the alphabet $\Sigma$.

For a set to be a language, it can't have any elements except for strings.

$\varnothing$

$\{\varepsilon\}$

{`kitty`}

{`kitty`, `cat`}

For a set to be a language, it can't have any elements except for strings.

$\varnothing$ — 0 strings – the *empty language*

$\{\varepsilon\}$

{`kitty`}

{`kitty`, `cat`}

For a set to be a language, it can’t have any elements except for strings.

| | |
|---|---|
| $\varnothing$ | 0 strings – the *empty language* |
| $\{\varepsilon\}$ | 1 string – the language containing the empty string |
| {`kitty`} | |
| {`kitty`, `cat`} | |

For a set to be a language, it can't have any elements except for strings.

| | |
|---|---|
| $\varnothing$ | 0 strings – the *empty language* |
| {ε} | 1 string – the language containing the empty string |
| {`kitty`} | 1 string – the language containing the string `kitty` |
| {`kitty`, `cat`} | |

For a set to be a language, it can't have any elements except for strings.

| | |
|---|---|
| $\varnothing$ | 0 strings – the *empty language* |
| $\{\varepsilon\}$ | 1 string – the language containing the empty string |
| {`kitty`} | 1 string – the language containing the string `kitty` |
| {`kitty`, `cat`} | 2 strings – the language containing the strings `kitty` and `cat` |

A language can be *finite* – it might contain a fixed number of strings, even if that number is very large!

Or a language can be *infinite* – it might contain an unbounded number of strings!

For example, the language of palindromes over $\Sigma = \{a, b, c\}$ is the infinite set

$$\{\varepsilon, a, b, c, aa, bb, cc, aaa, aba, aca, bab, \ldots\}$$

The language of all strings composed from characters in $\Sigma$ is denoted $\Sigma^*$.

So, formally, we can say that $L$ is a language over $\Sigma$ iff $L \subseteq \Sigma^*$.

## Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😟

## Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😟

- $\in$ is the element-of relation.

## Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😟

$\in$ is the element-of relation.

$\varepsilon$ is the empty string.

## Mathematical lookalikes

We now have ∈, ε, Σ, and $\Sigma^*$ 😟

∈ is the element-of relation.

ε is the empty string.

Σ is an alphabet.

## Mathematical lookalikes

We now have $\in$, ε, Σ, and $\Sigma^*$ 😟

$\in$ is the element-of relation.

ε is the empty string.

Σ is an alphabet.

$\Sigma^*$ means “all strings that can be made from characters in Σ”.

## Mathematical lookalikes

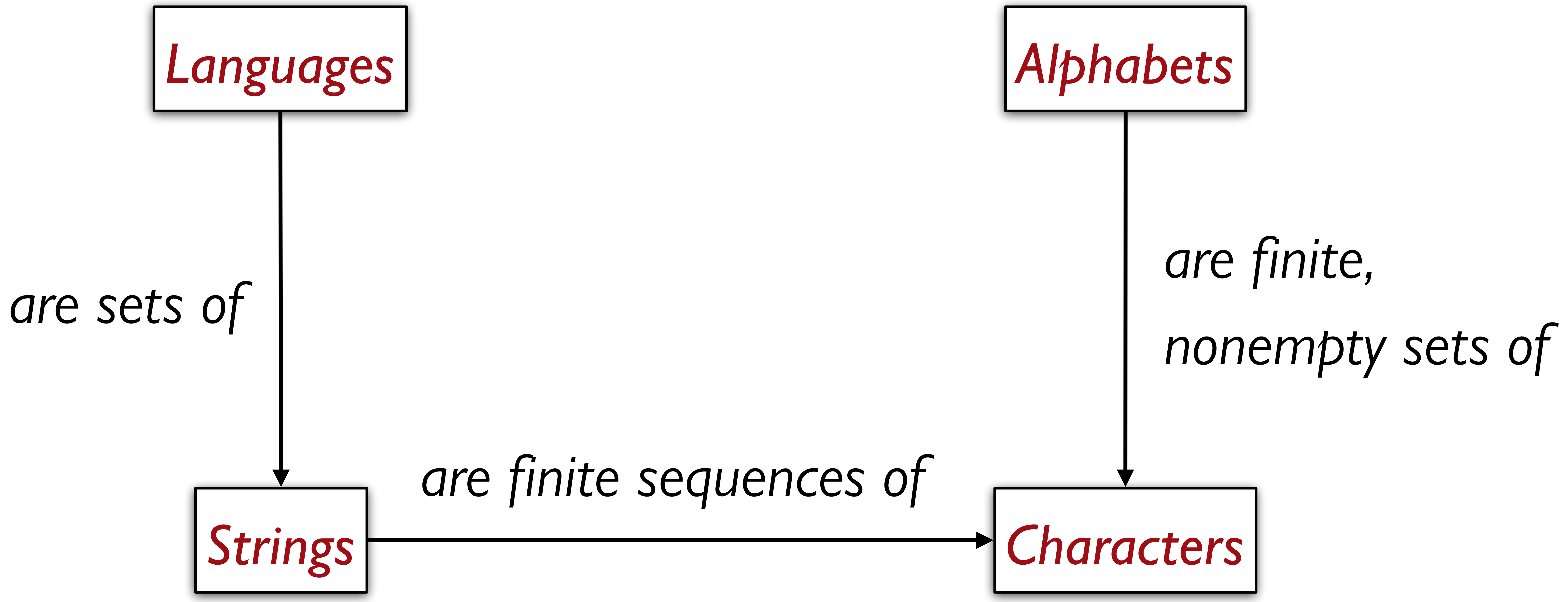
We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😟

- $\in$ is the element-of relation.
- $\varepsilon$ is the empty string.
- $\Sigma$ is an alphabet.
- $\Sigma^*$ means "all strings that can be made from characters in $\Sigma$".

This means we can write things like

- We have $\varepsilon \in \Sigma^*$, but $\varepsilon \notin \Sigma$

which is true!

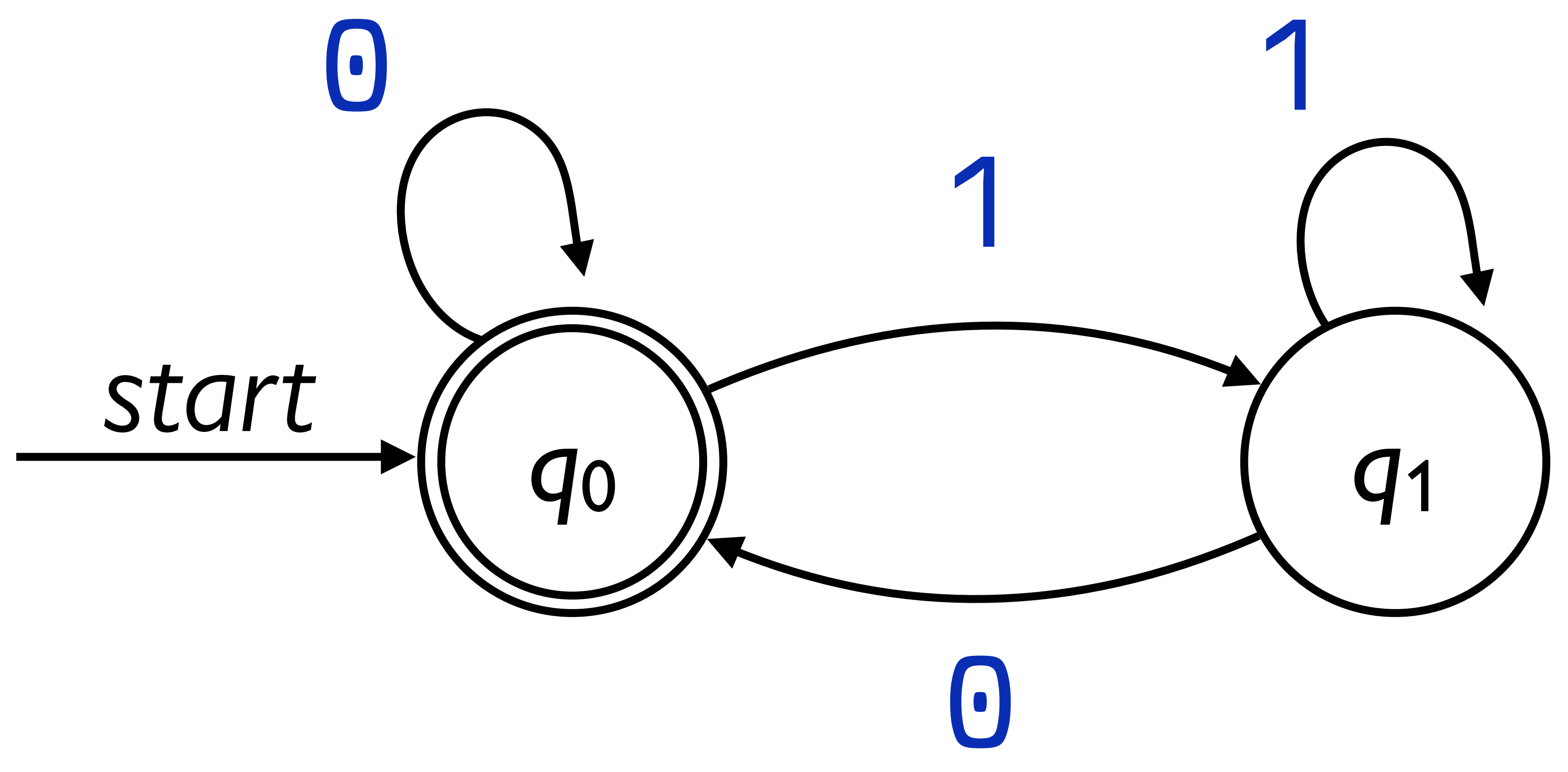

Languages
are sets of
Strings
are finite sequences of
Characters
Alphabets
are finite,
nonempty sets of

The *language of a finite automaton* is the set of strings that it accepts, i.e., strings that label paths that go from the start state to some accept state.

If $M$ is an automaton that processes characters from the alphabet $\Sigma$, then its language is defined as

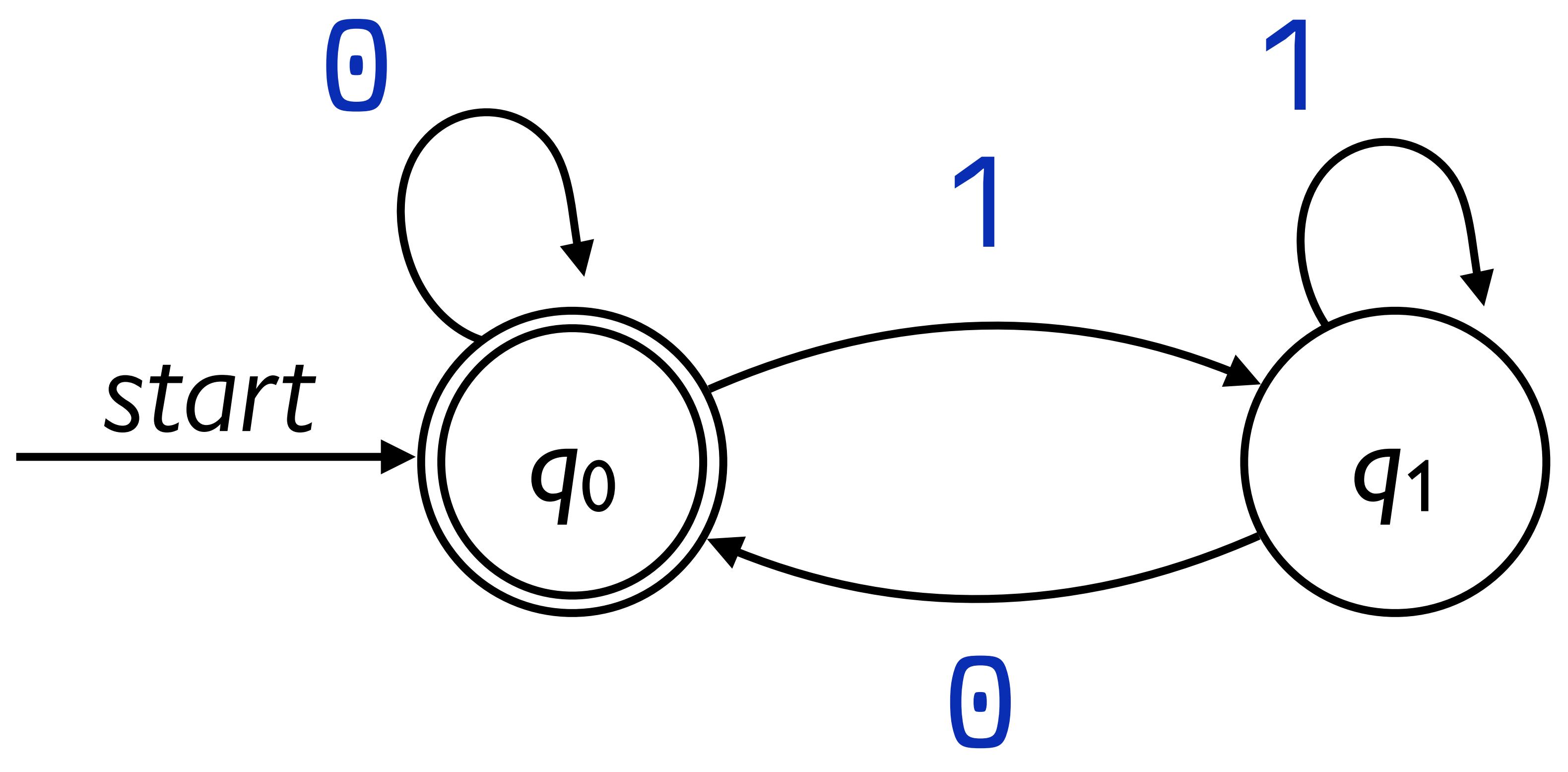
$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

## Exercise



$L(M) = \{\ ?\ \}$

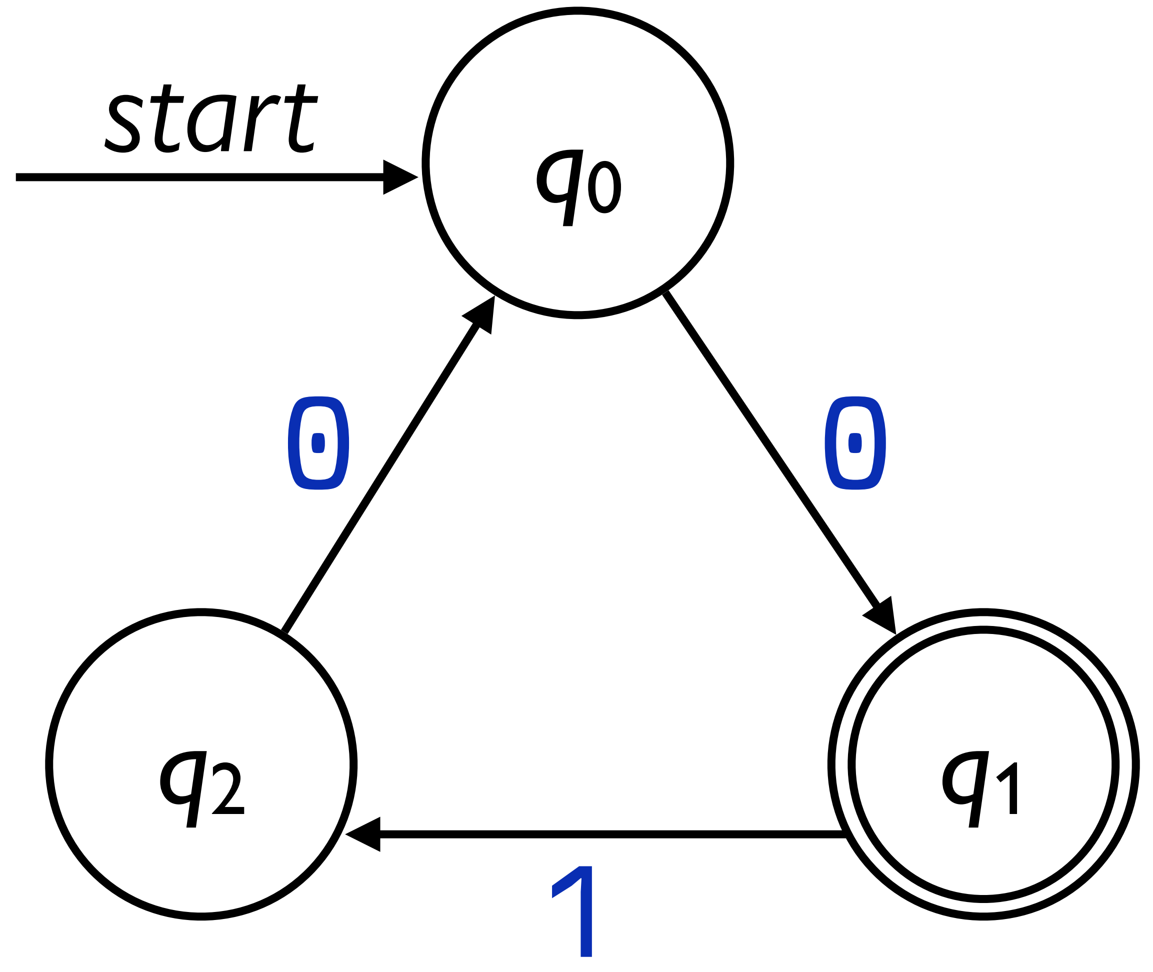
## Exercise



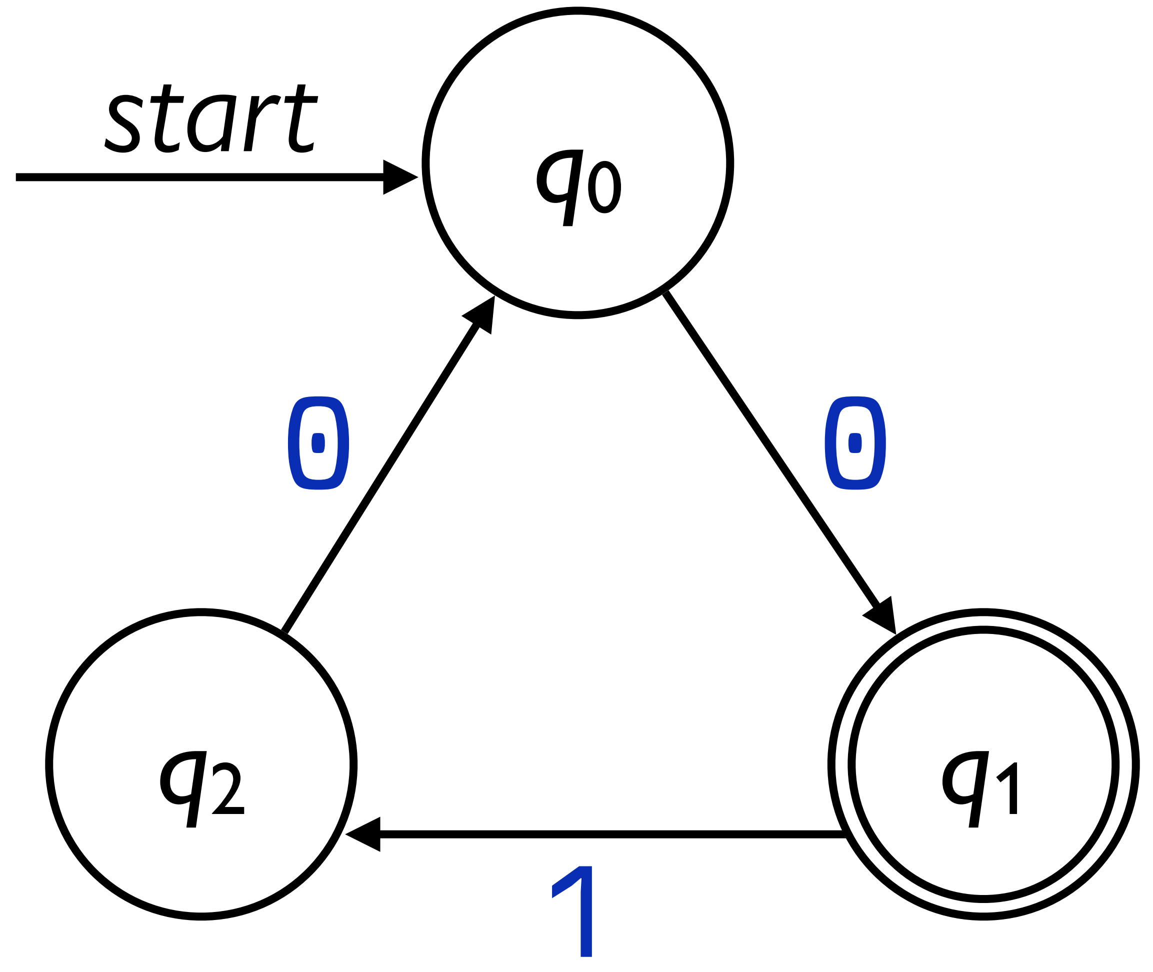
$L(M) = \{w \mid w \in \{0, 1\}^*$ and $w$ does not have suffix $1\}$

## Finite automata, revisited

## A small problem

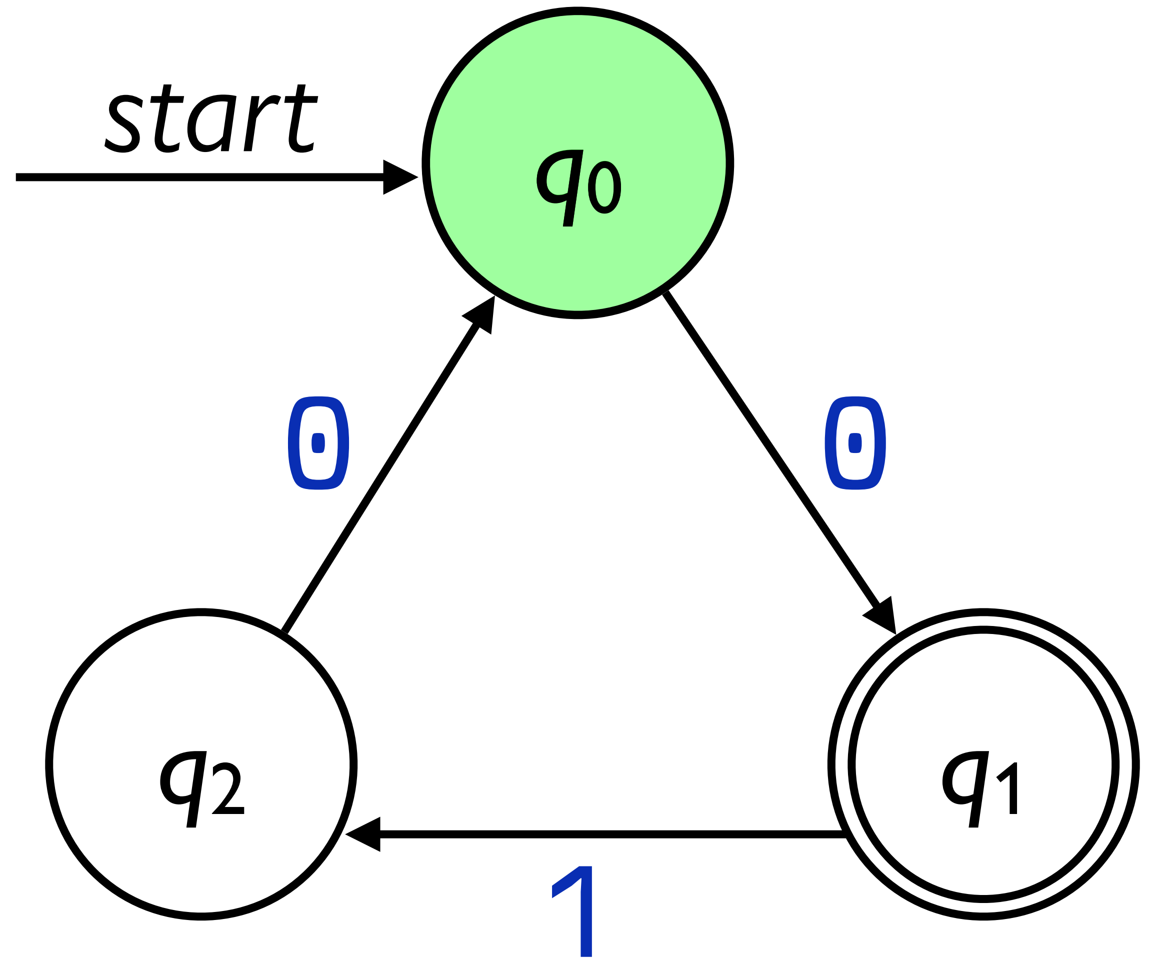


## A small problem



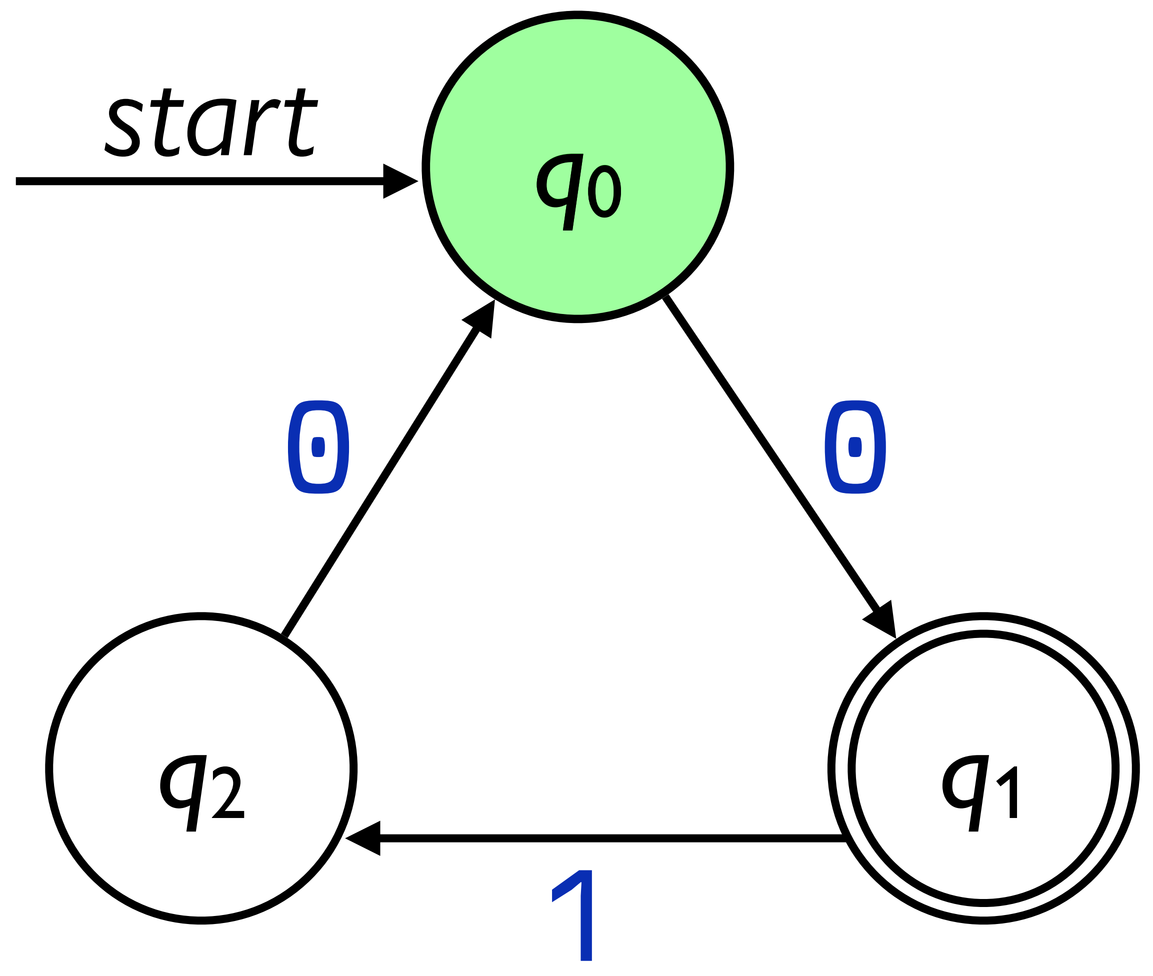
```
0 1 1 0
```

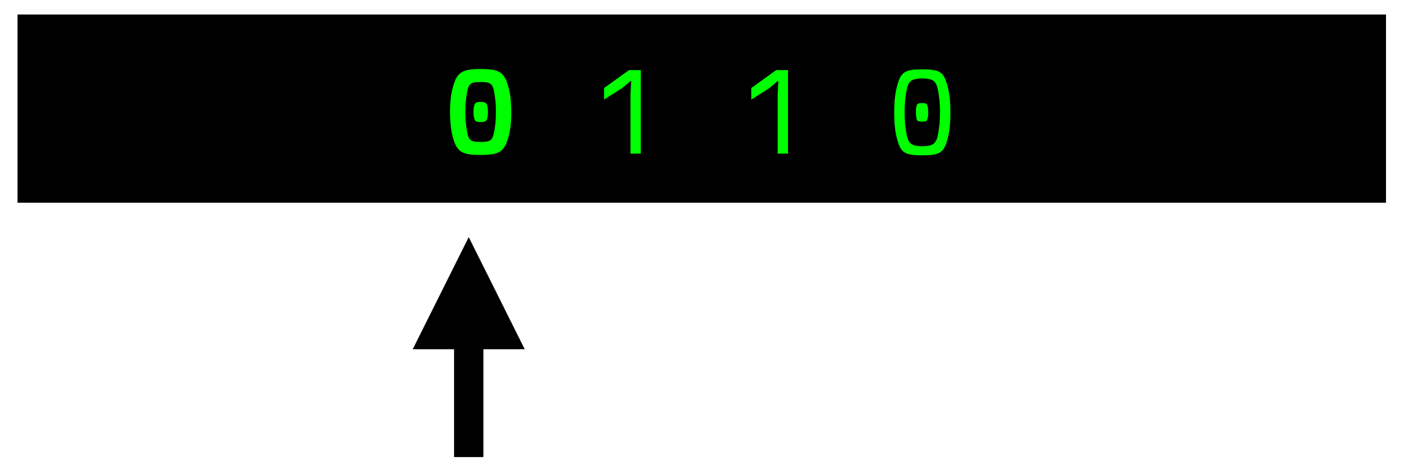
## A small problem



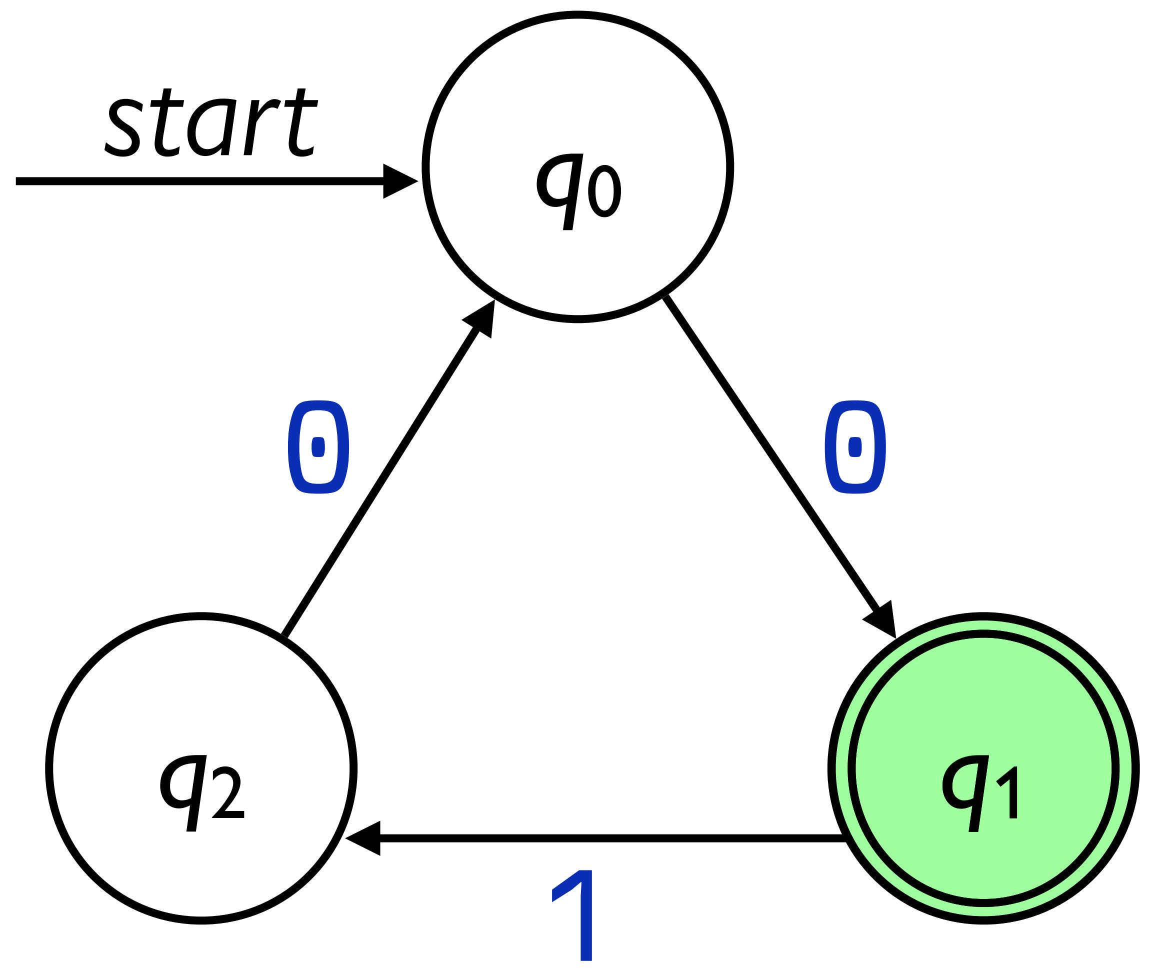
`0 1 1 0`

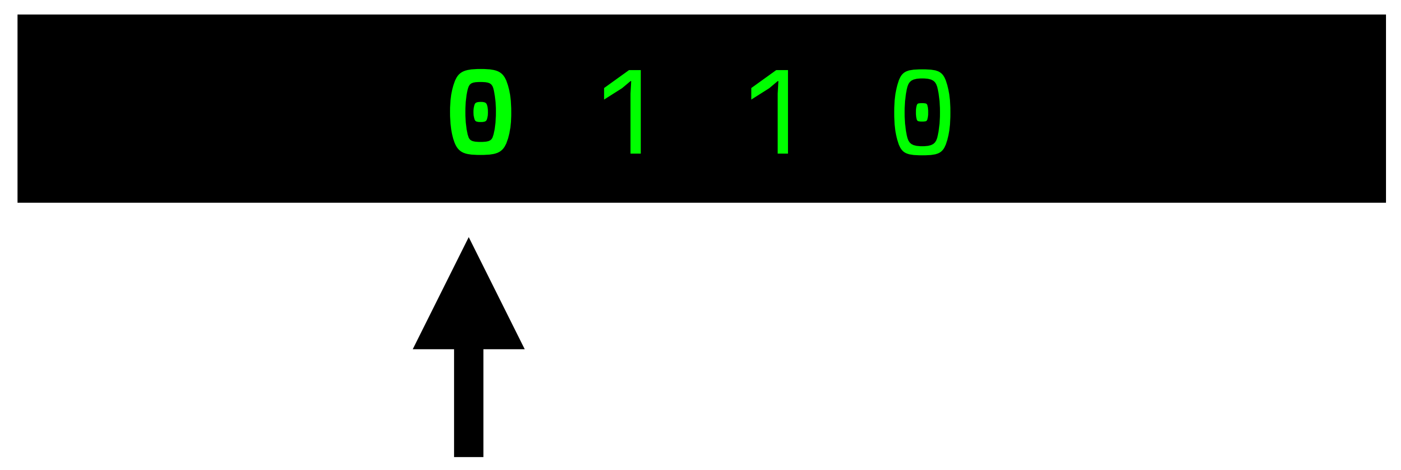
## A small problem



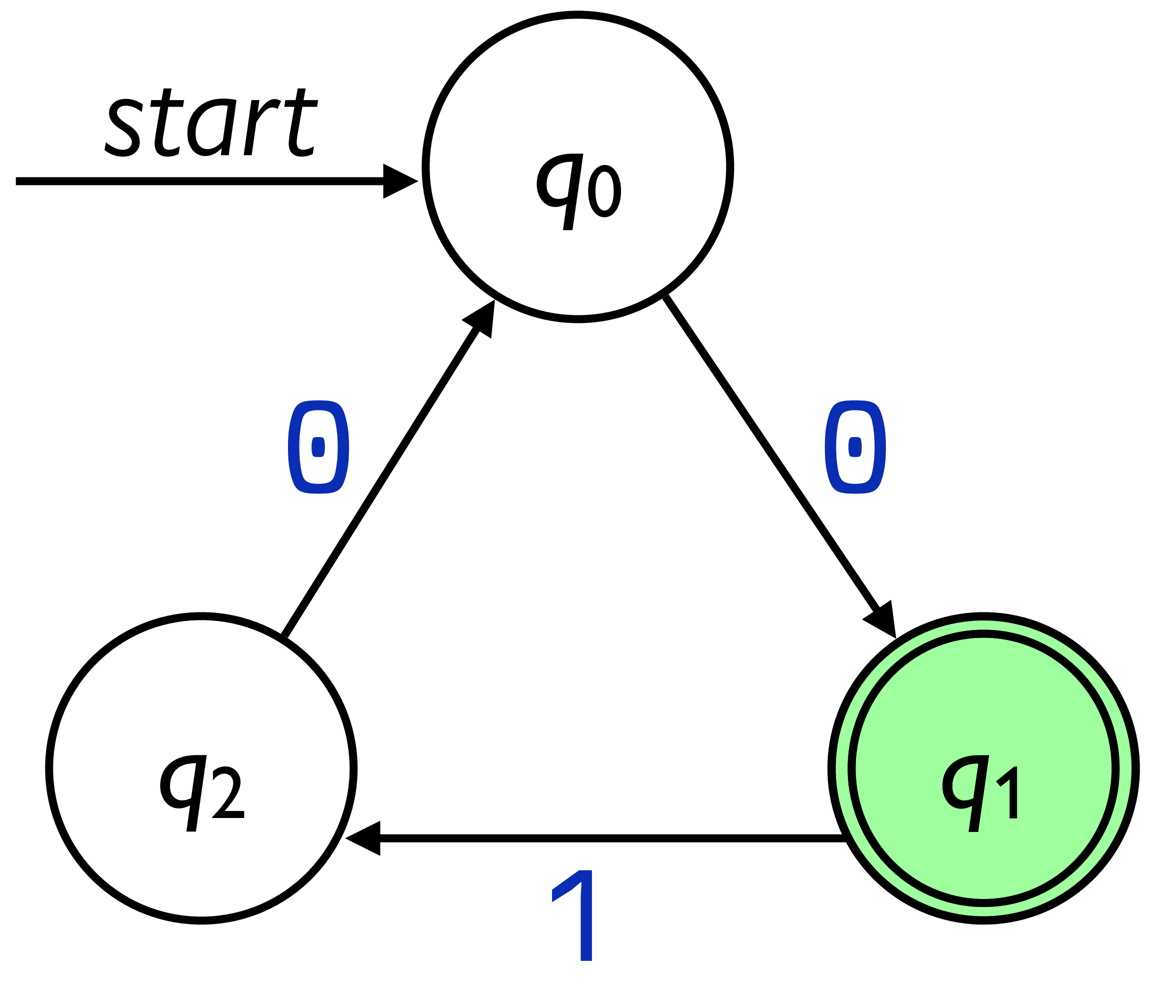


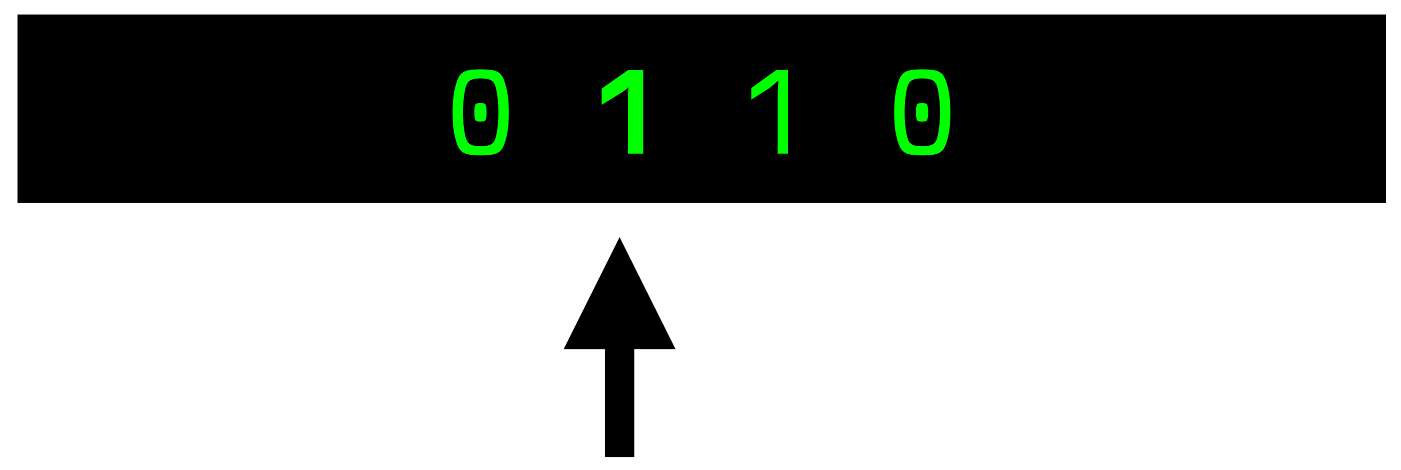
## A small problem



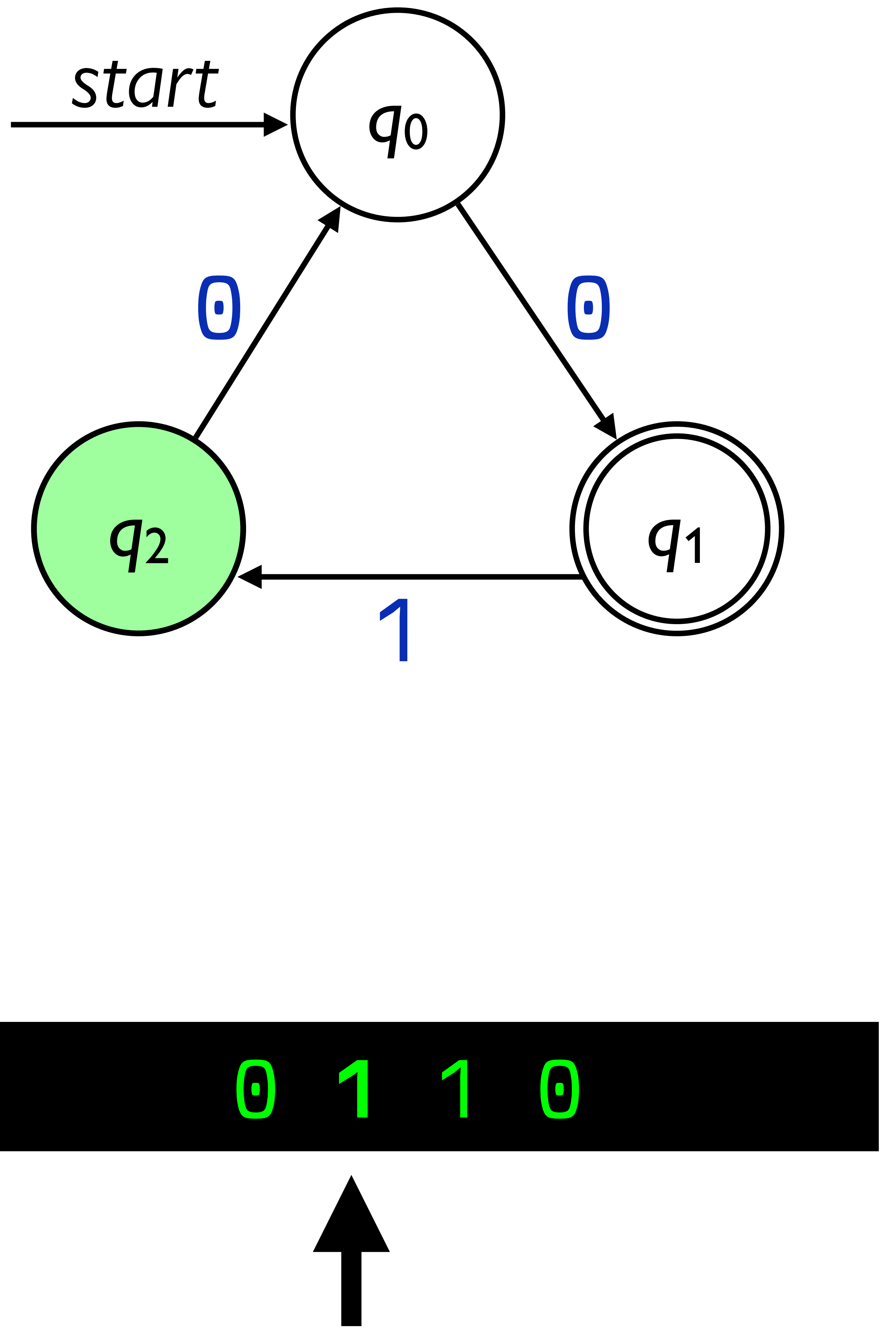


## A small problem

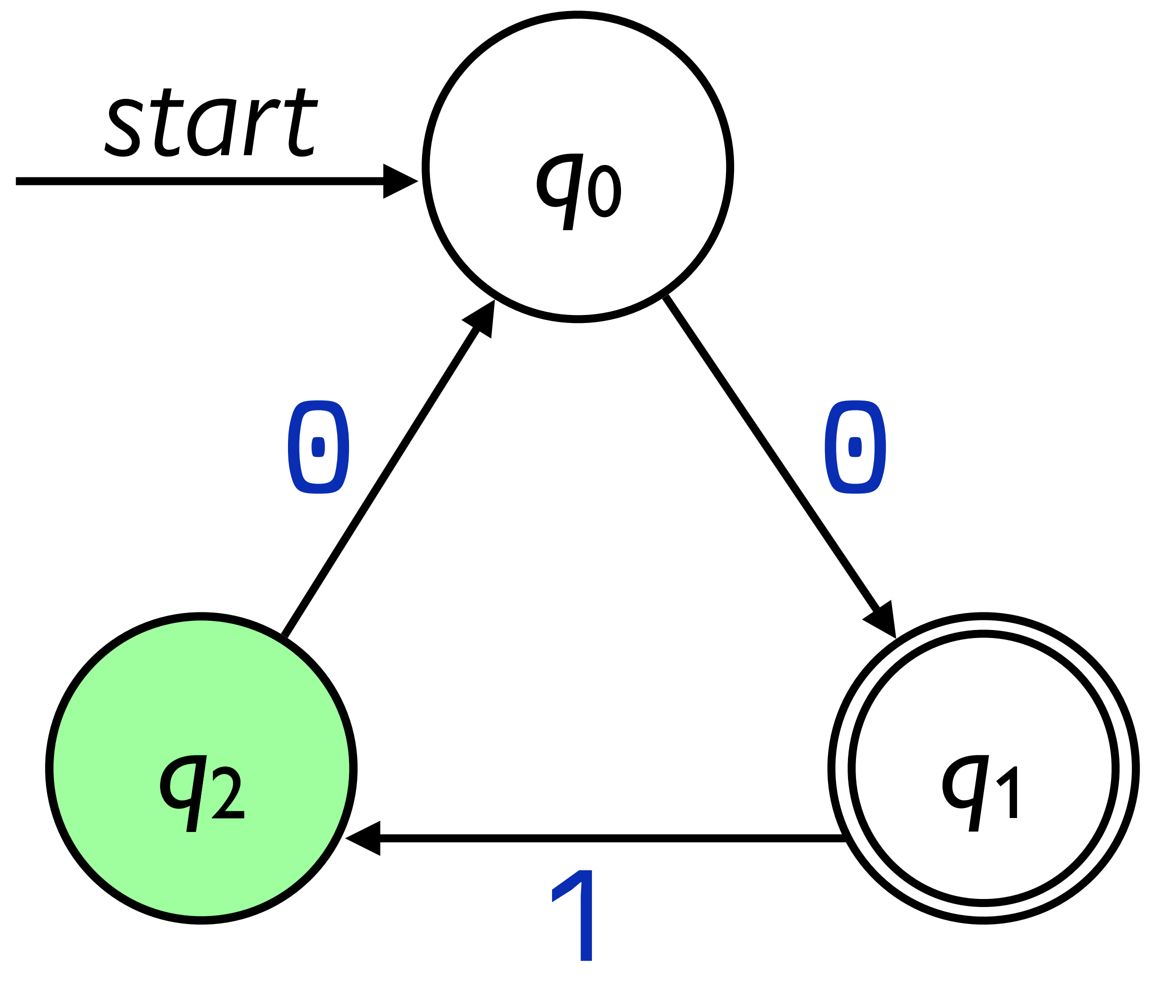


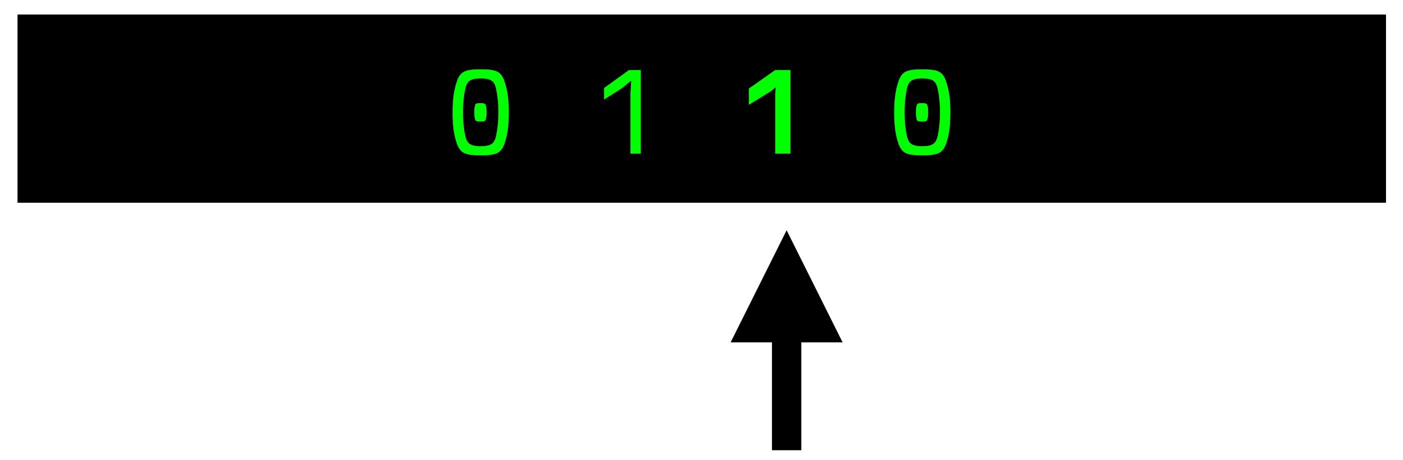


## A small problem

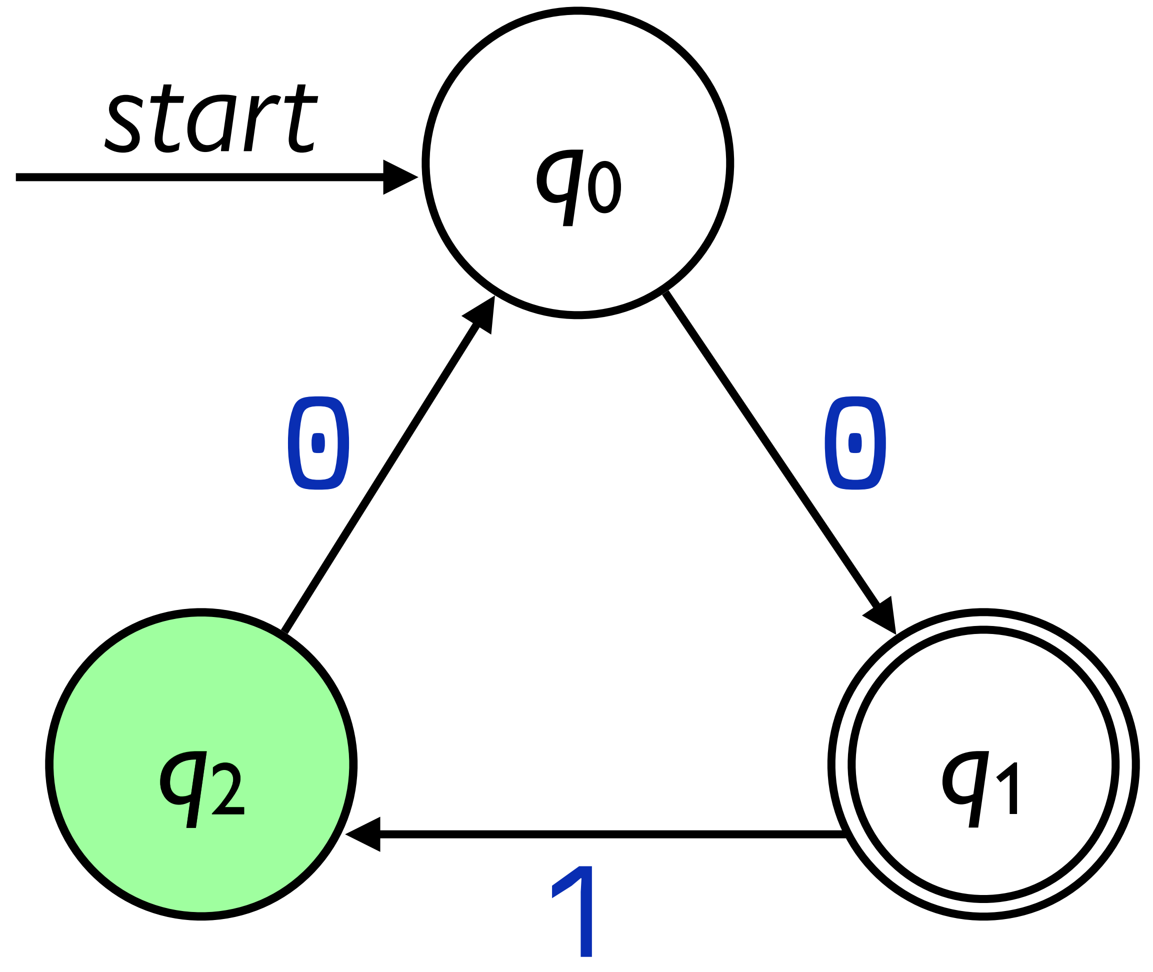


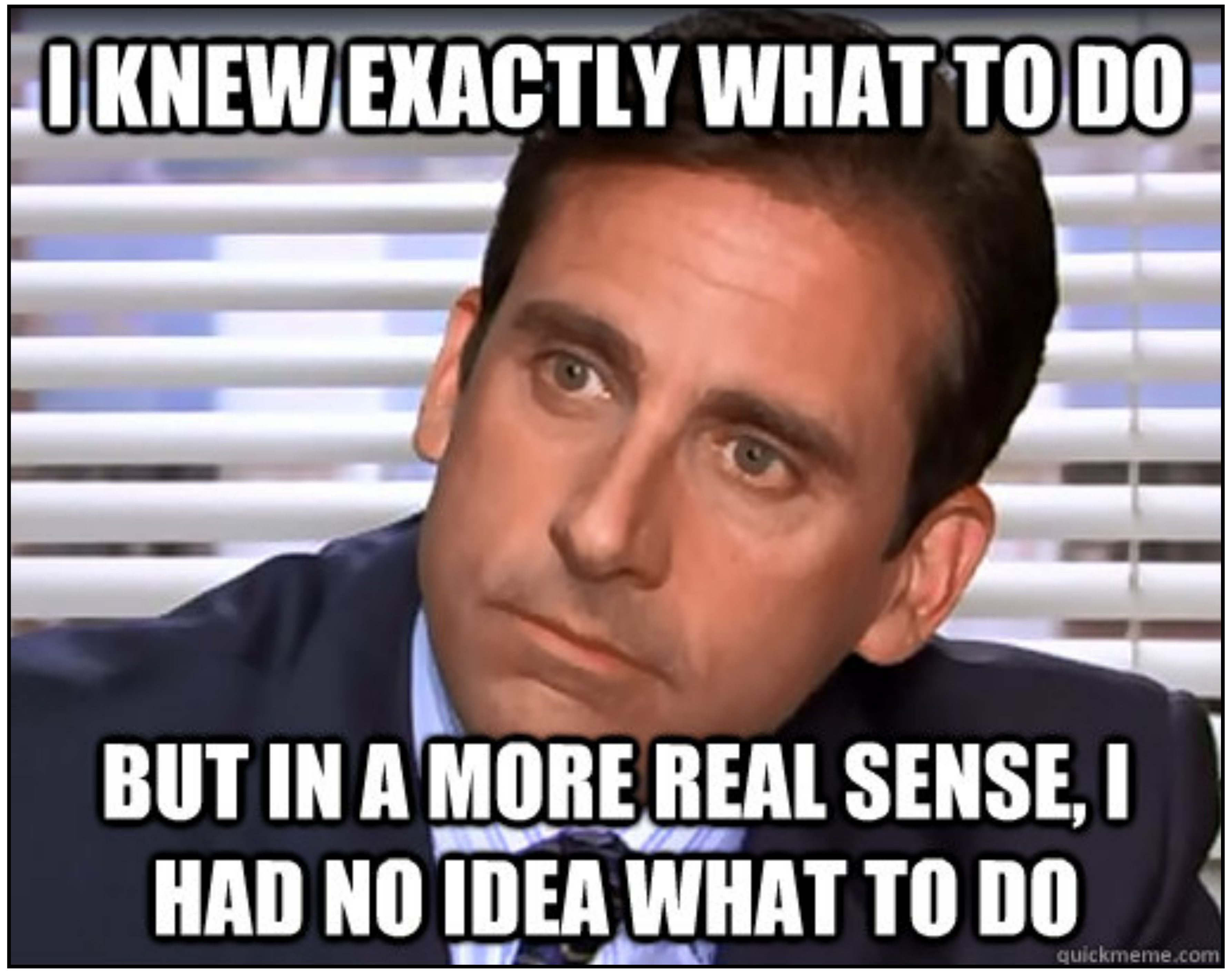
## A small problem





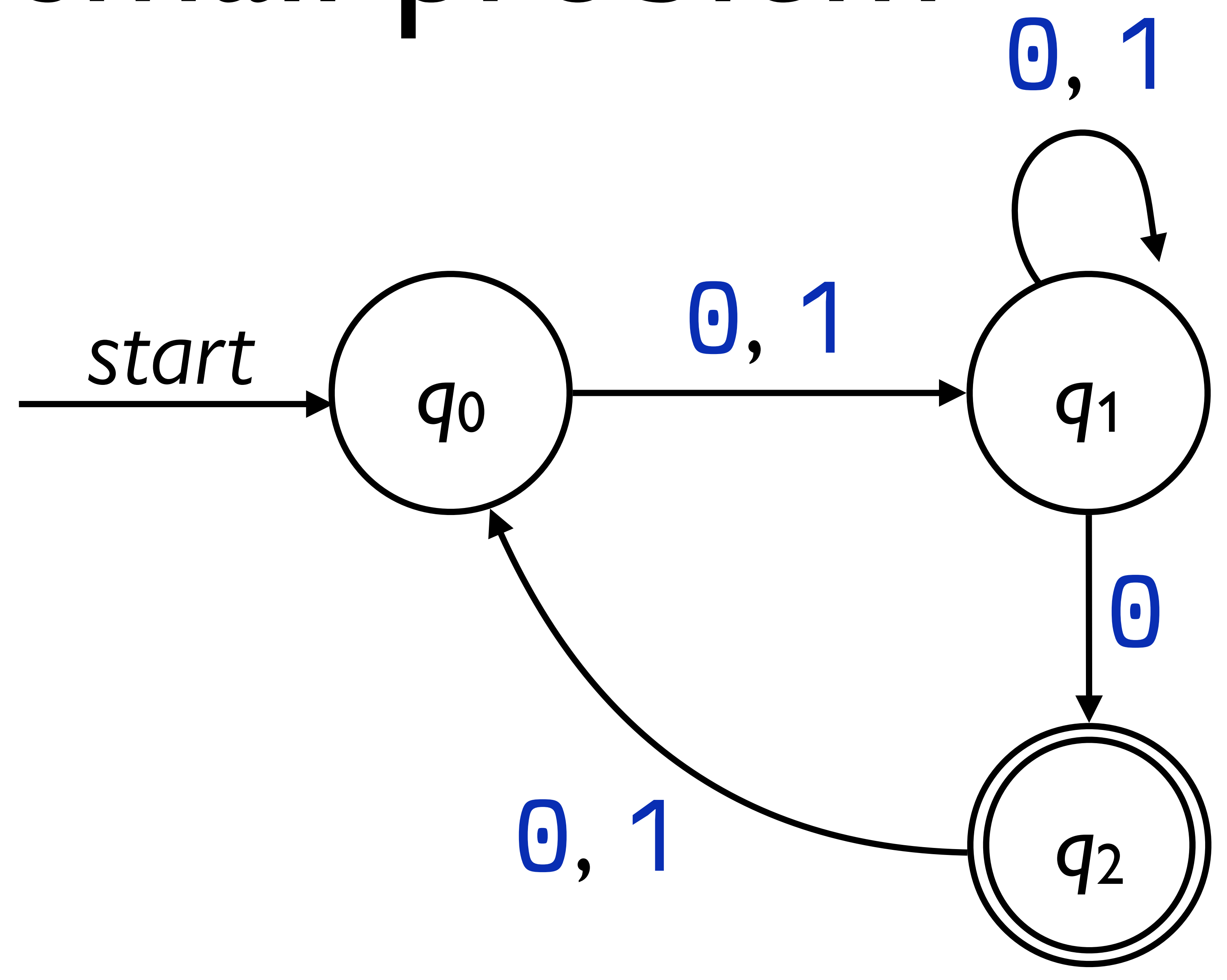
# A small problem



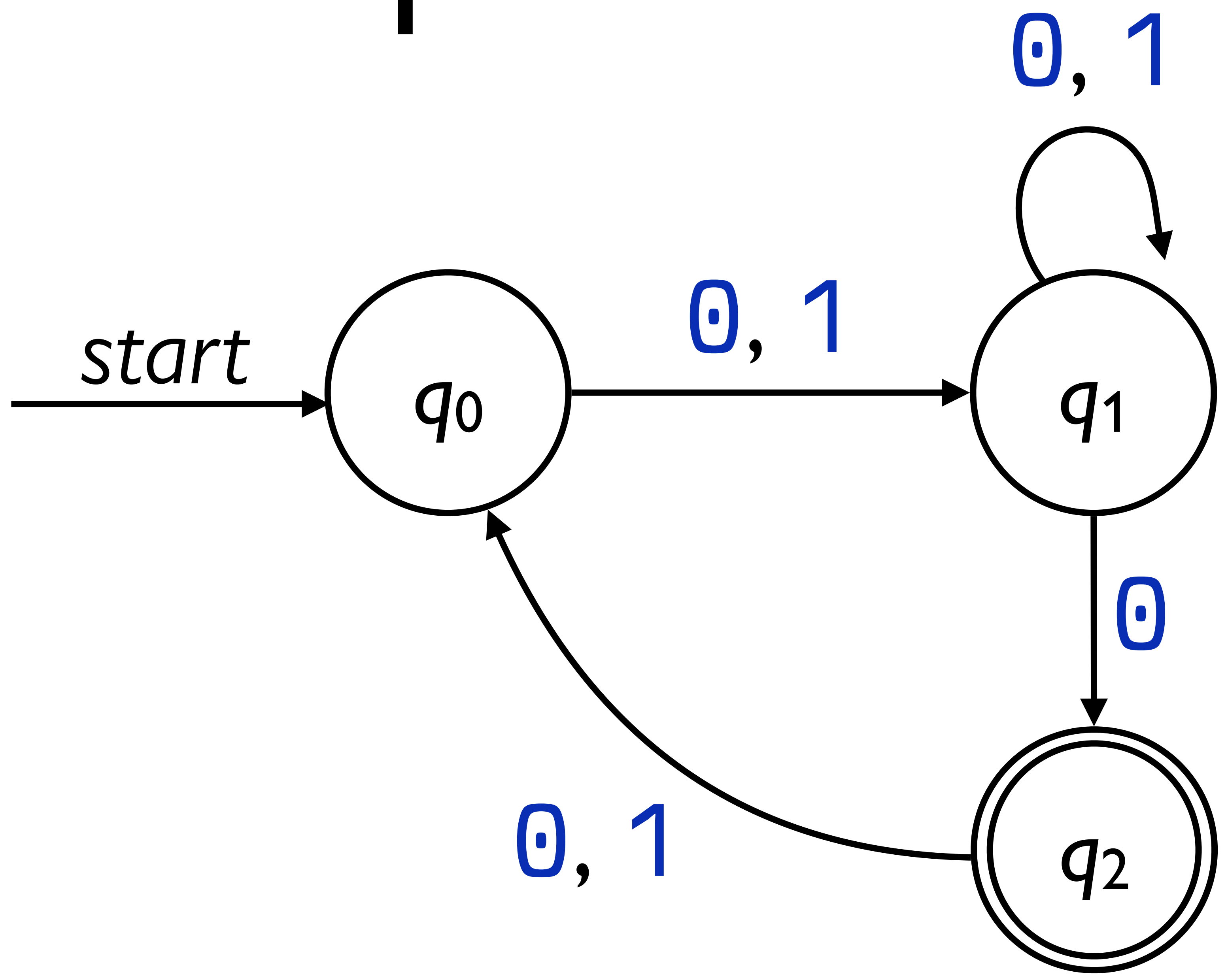

I KNEW EXACTLY WHAT TO DO
BUT IN A MORE REAL SENSE, I HAD NO IDEA WHAT TO DO
quickmeme.com

```
0 1 1 0
```

## Another small problem

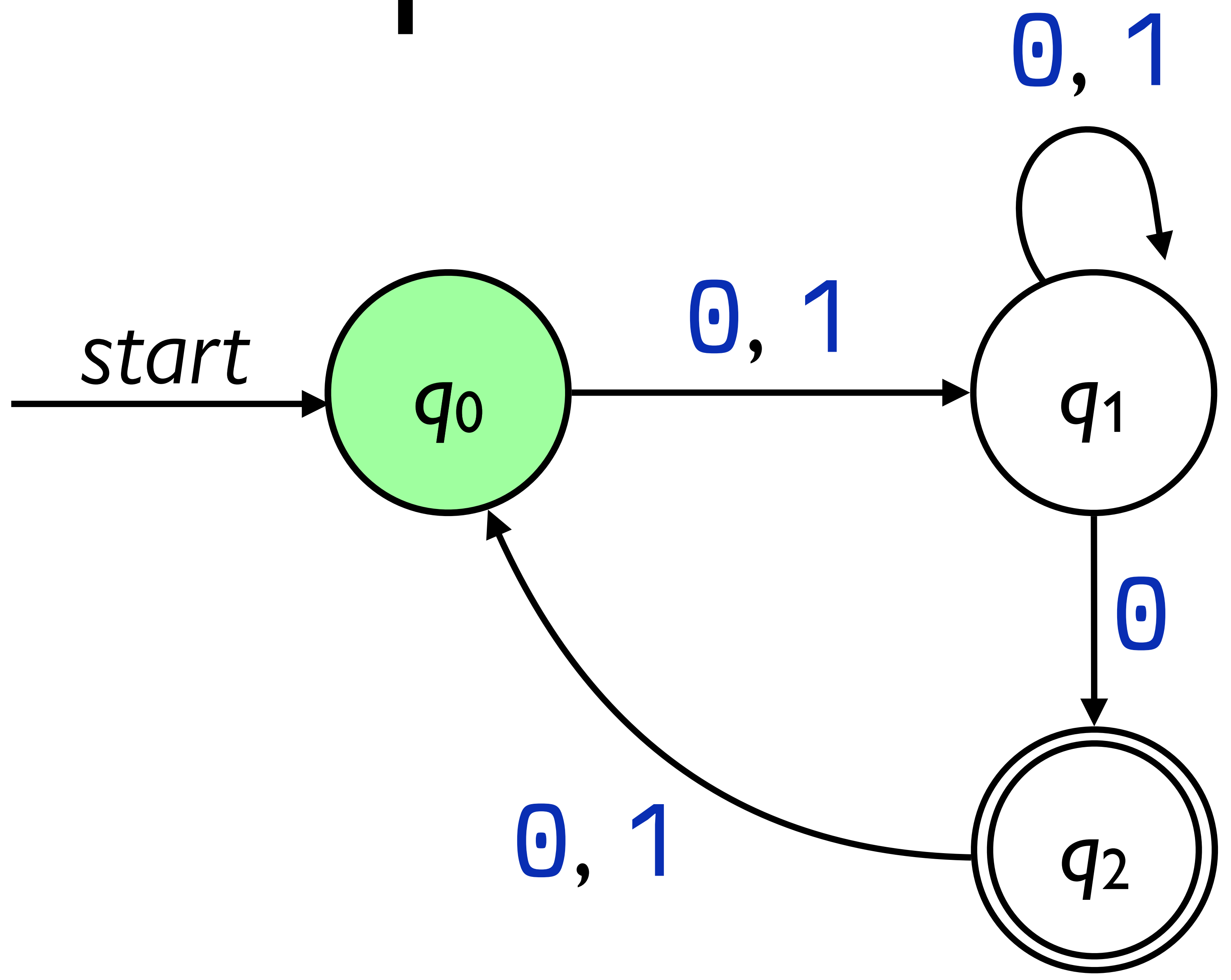


## Another small problem



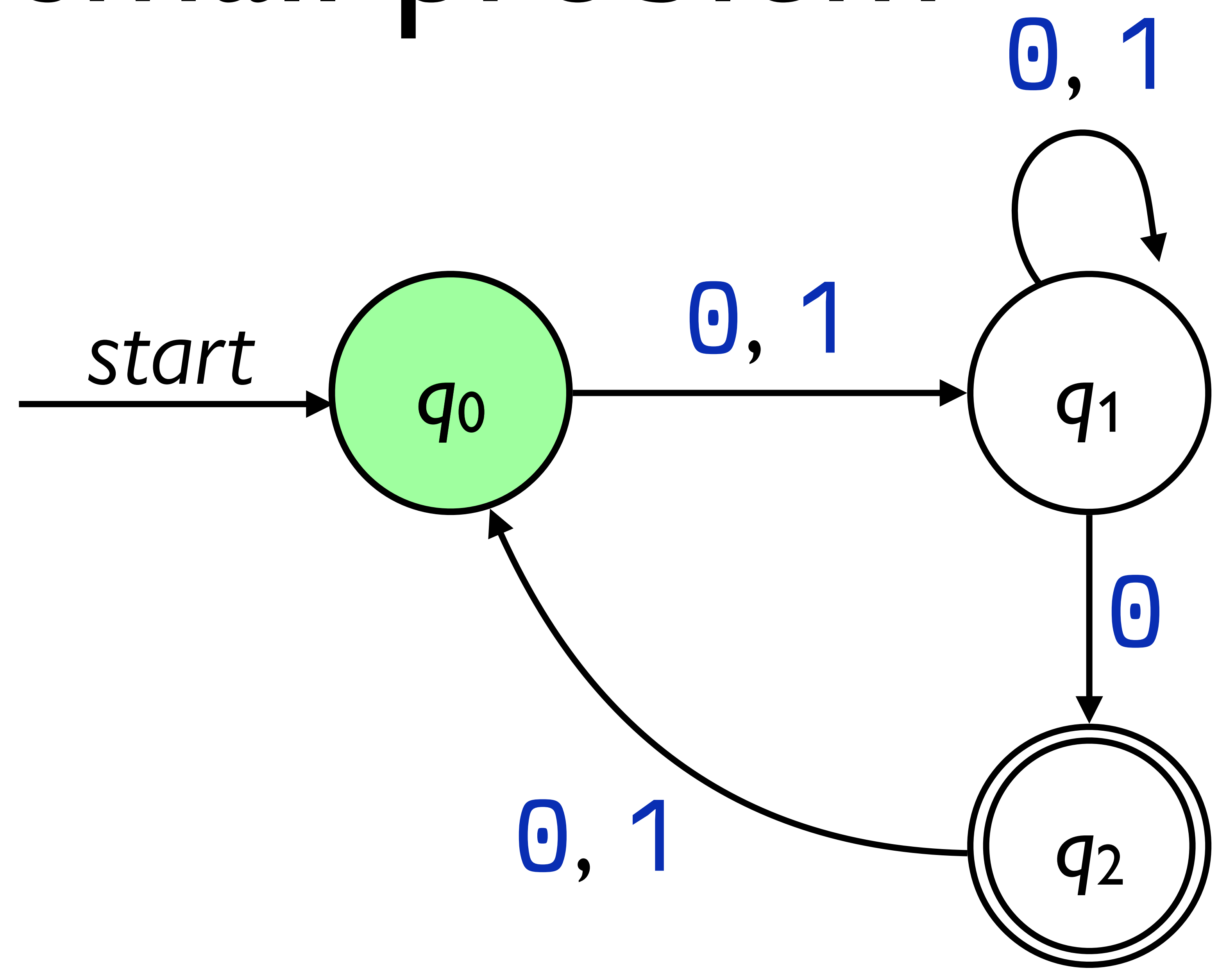
```
0 0 0
```

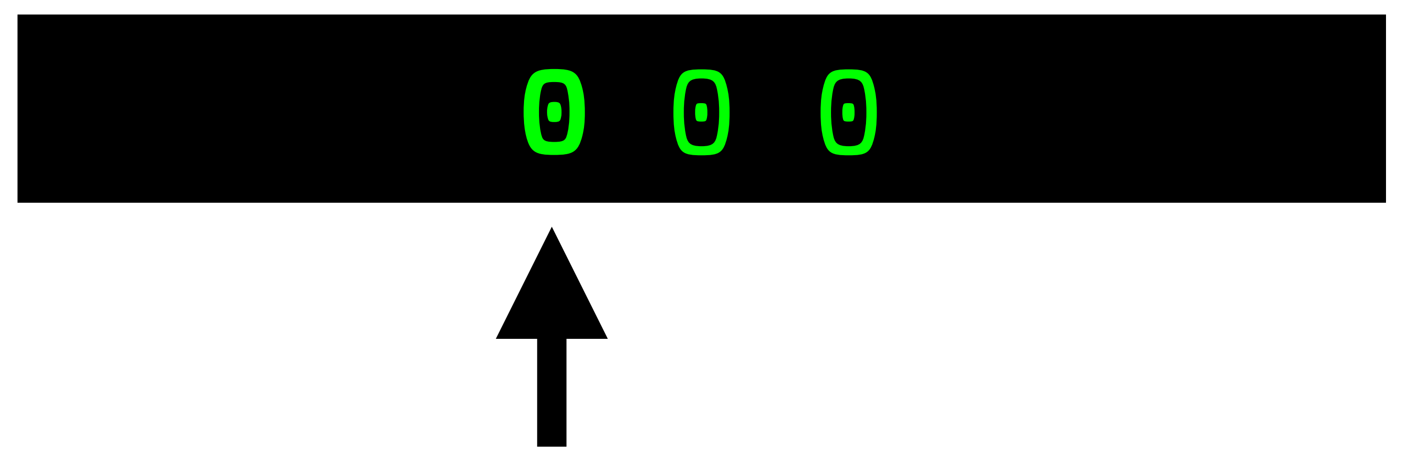
## Another small problem



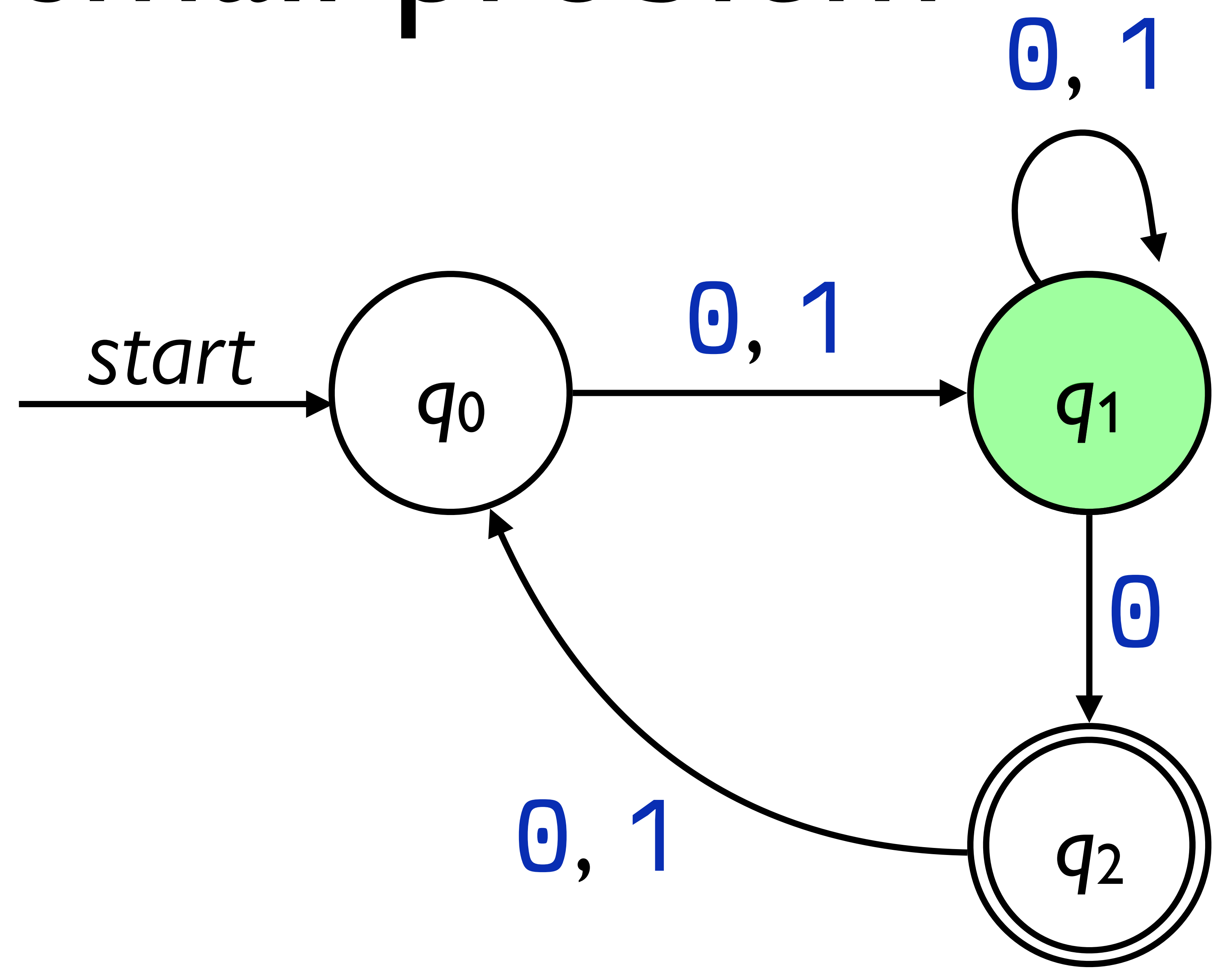
```
0 0 0
```

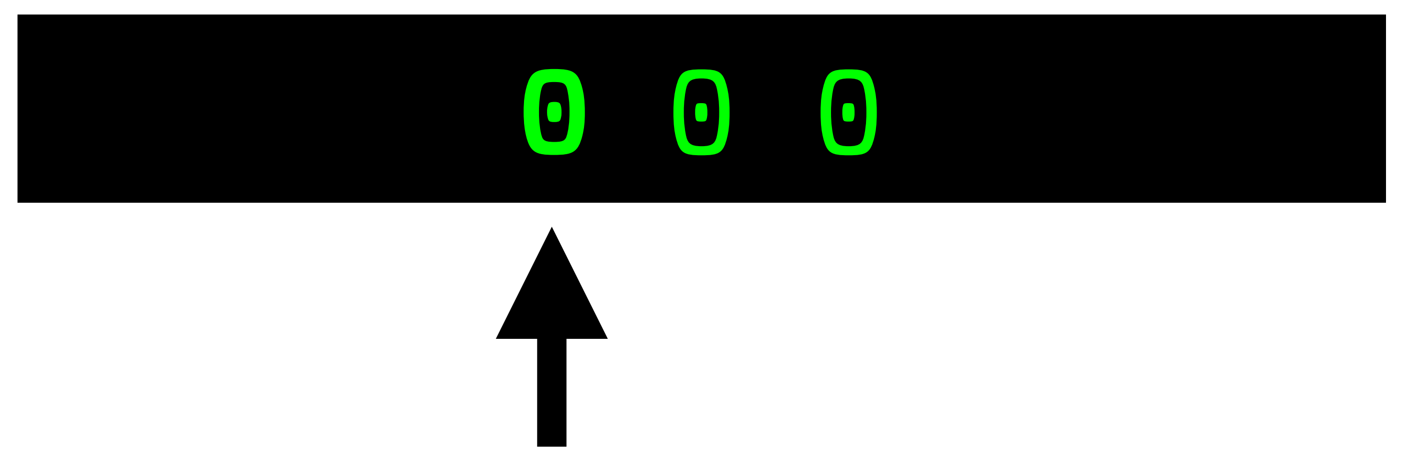
## Another small problem



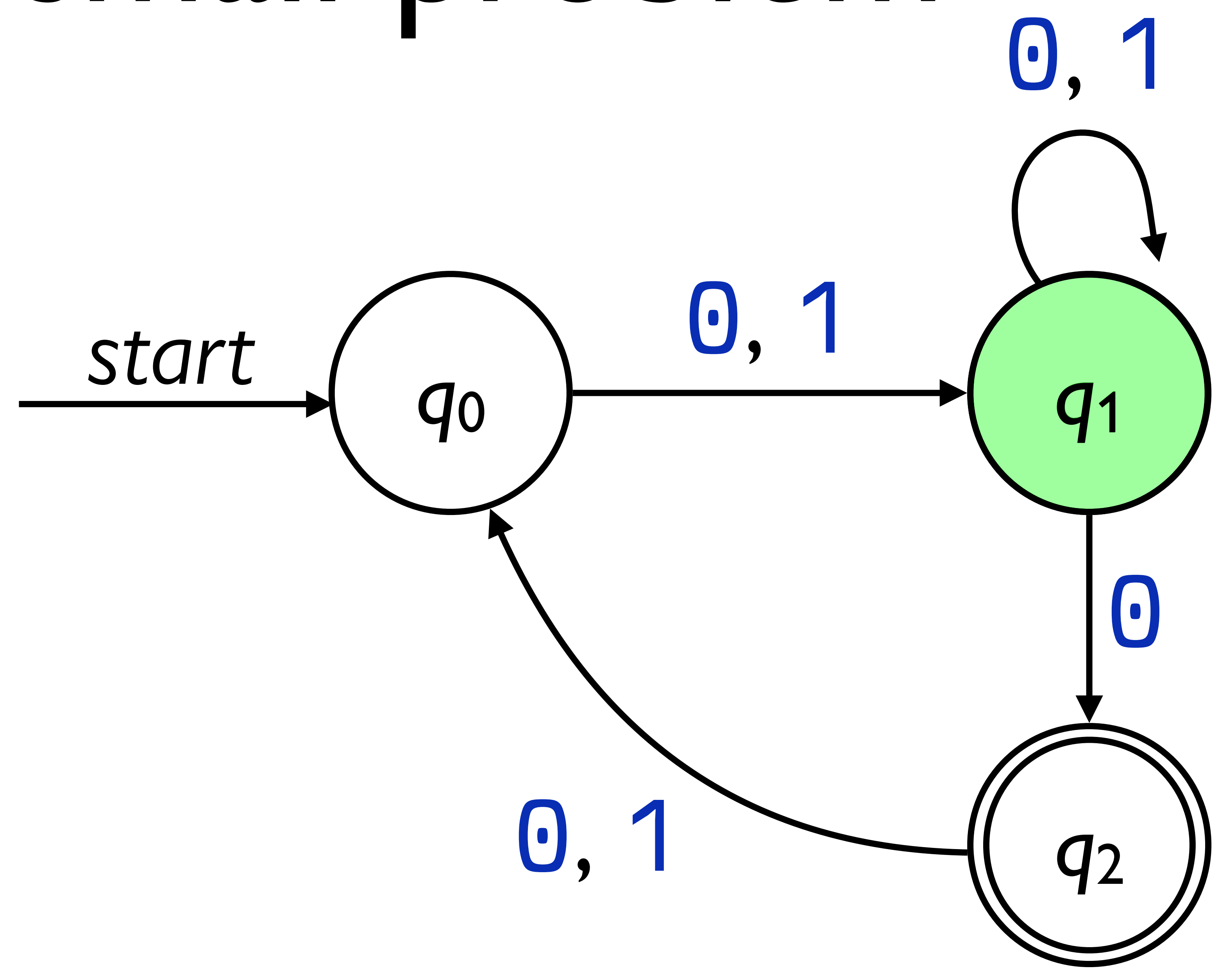


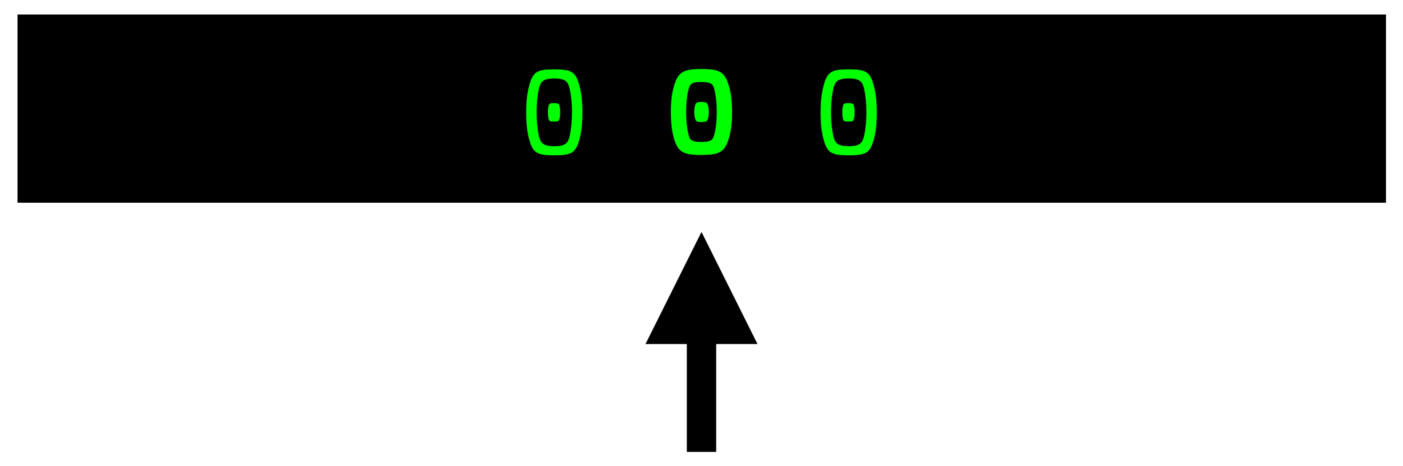
## Another small problem



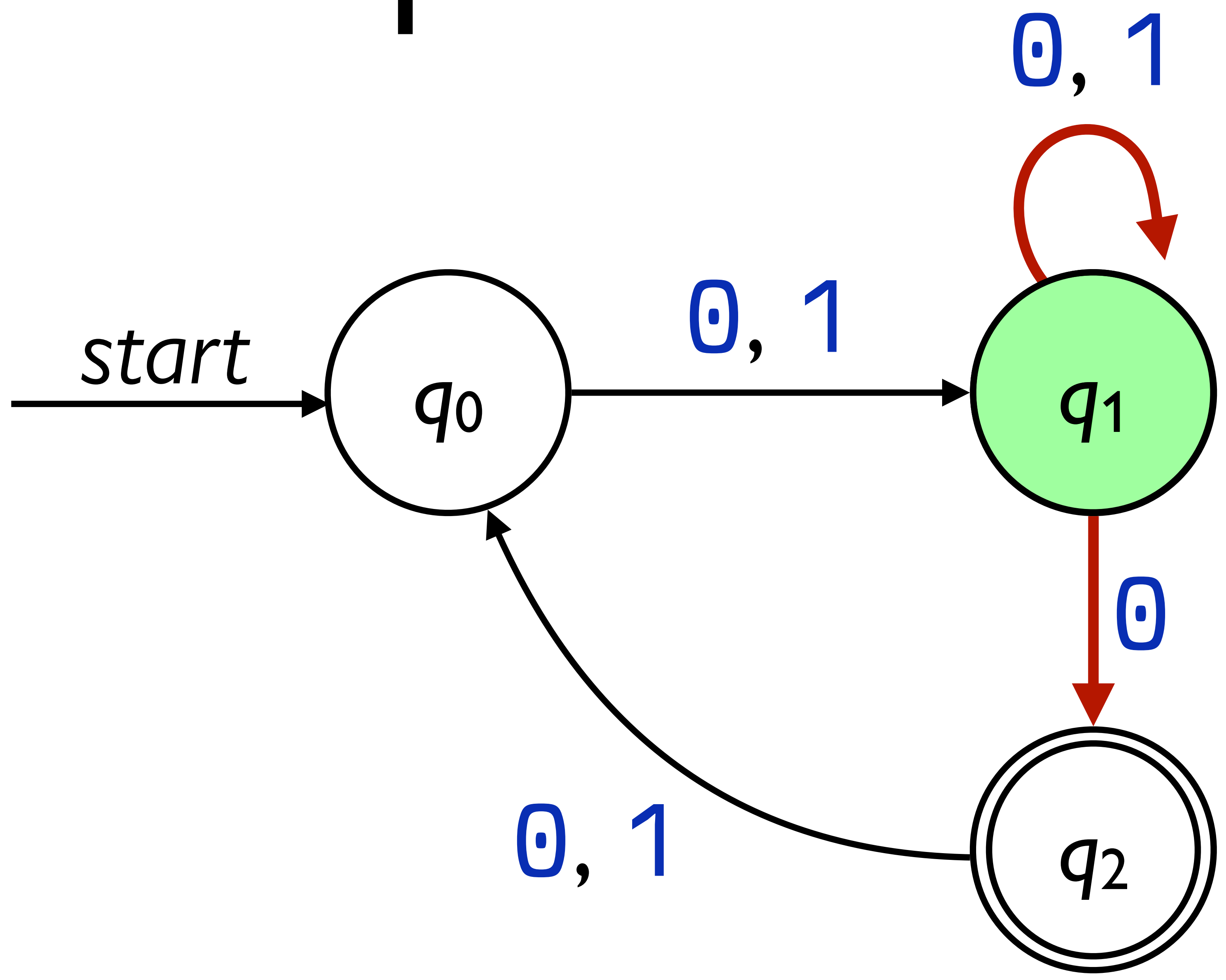


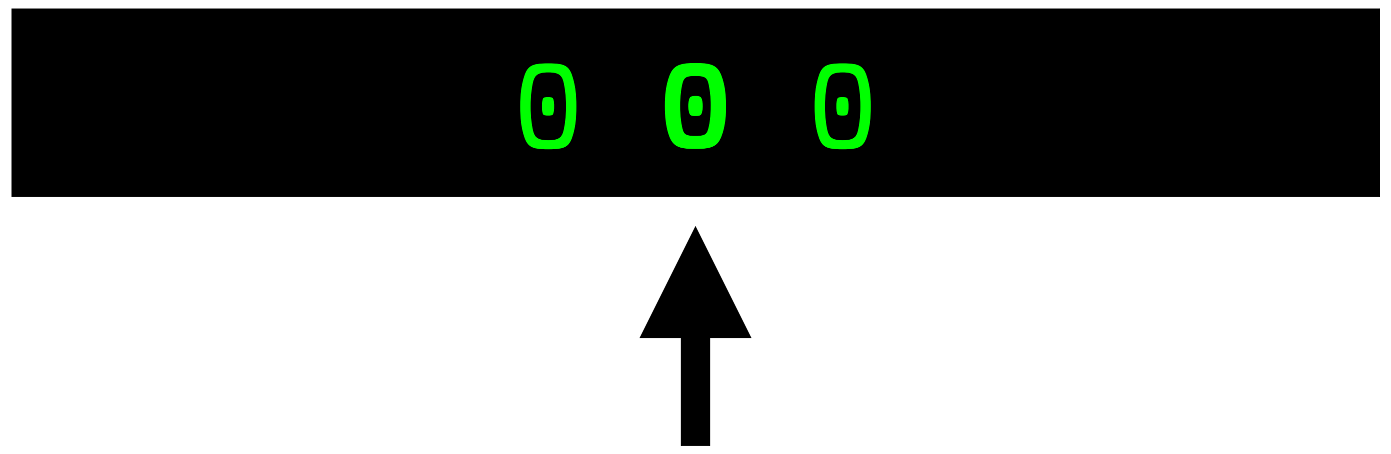
## Another small problem



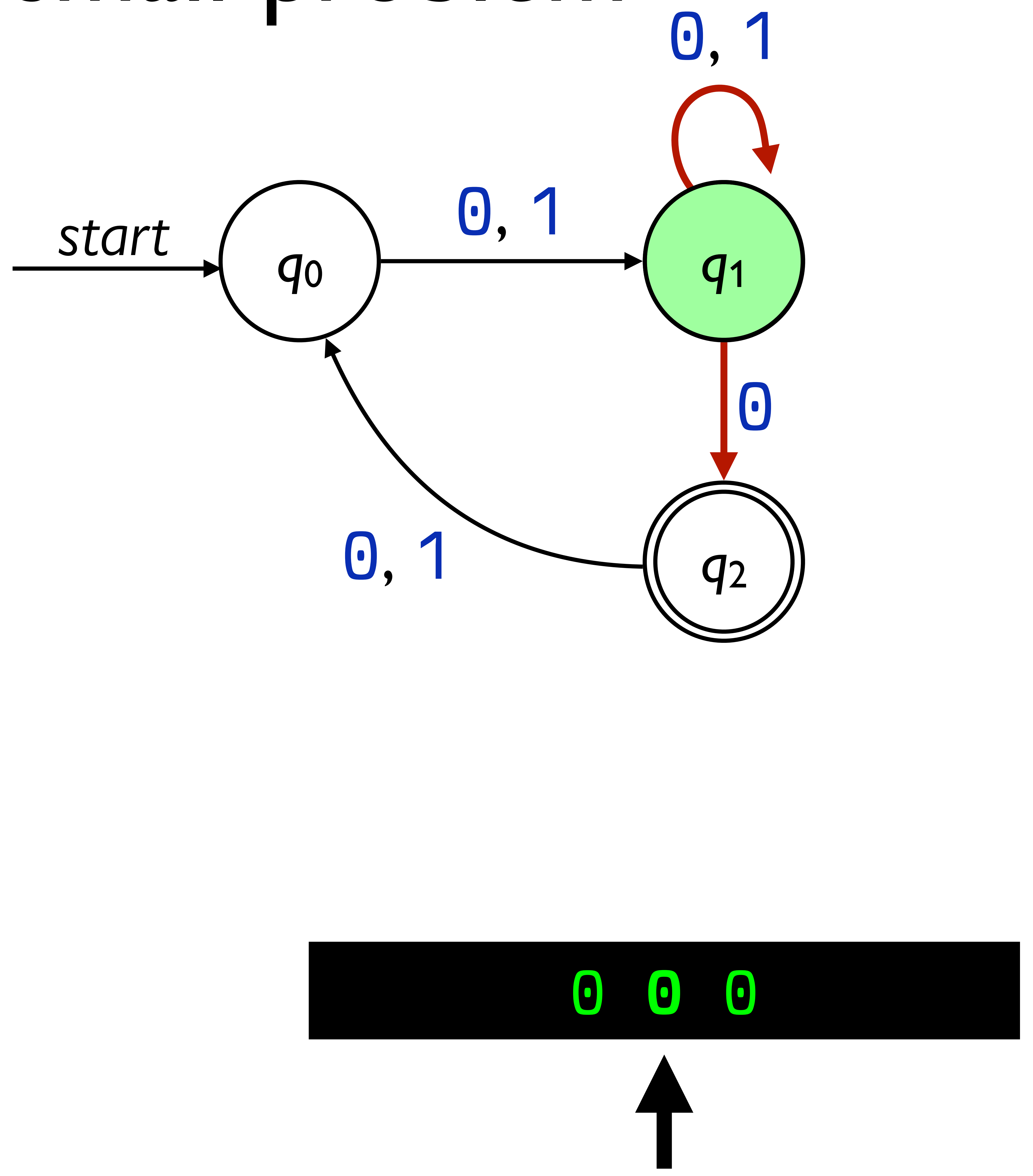


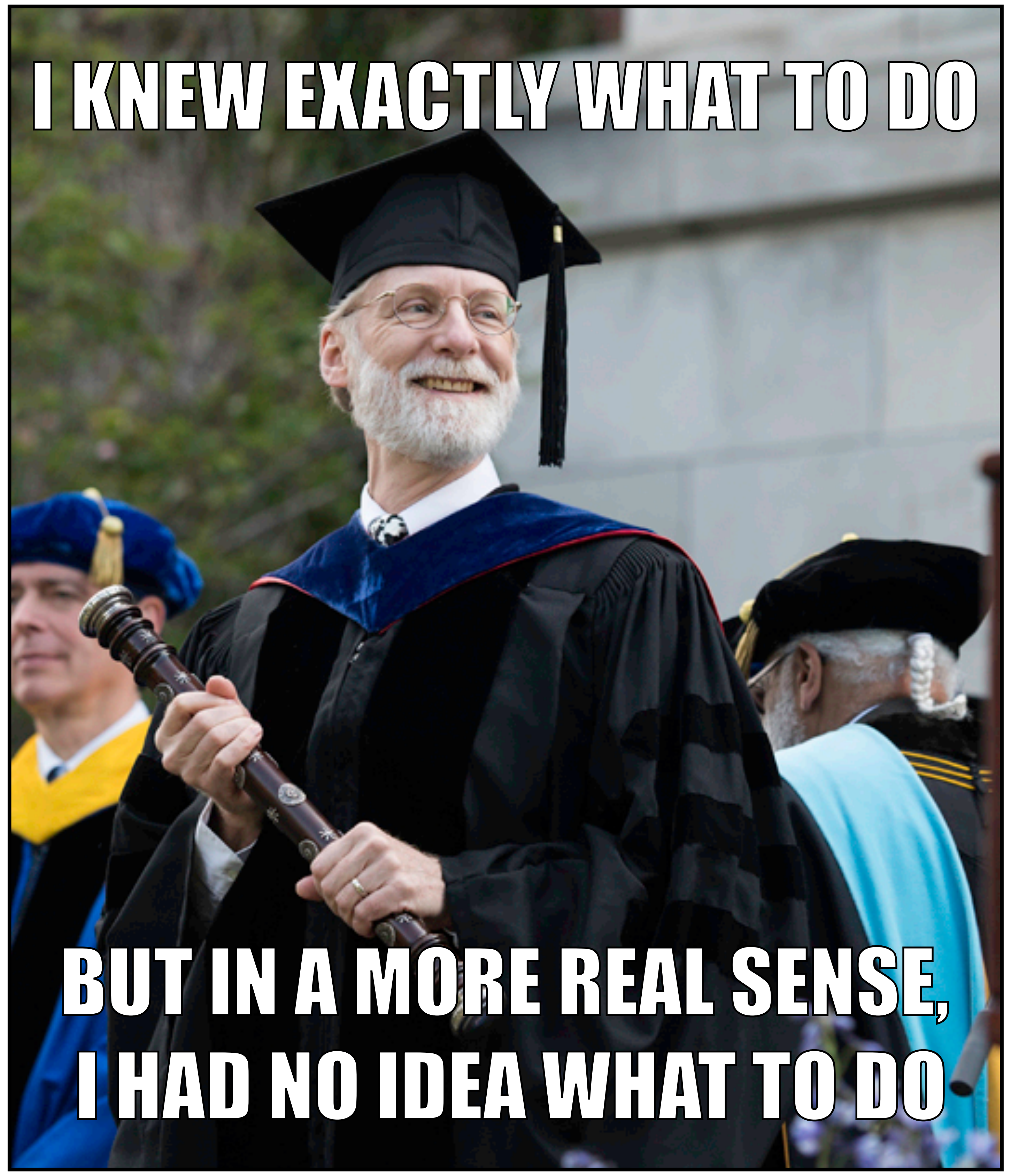
## Another small problem





## Another small problem




I KNEW EXACTLY WHAT TO DO
BUT IN A MORE REAL SENSE, I HAD NO IDEA WHAT TO DO

## 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.

What happens if there is *no* transition out of a state on some input?

What is there are *multiple* transitions out of a state on some input?

A *deterministic finite automaton* (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.

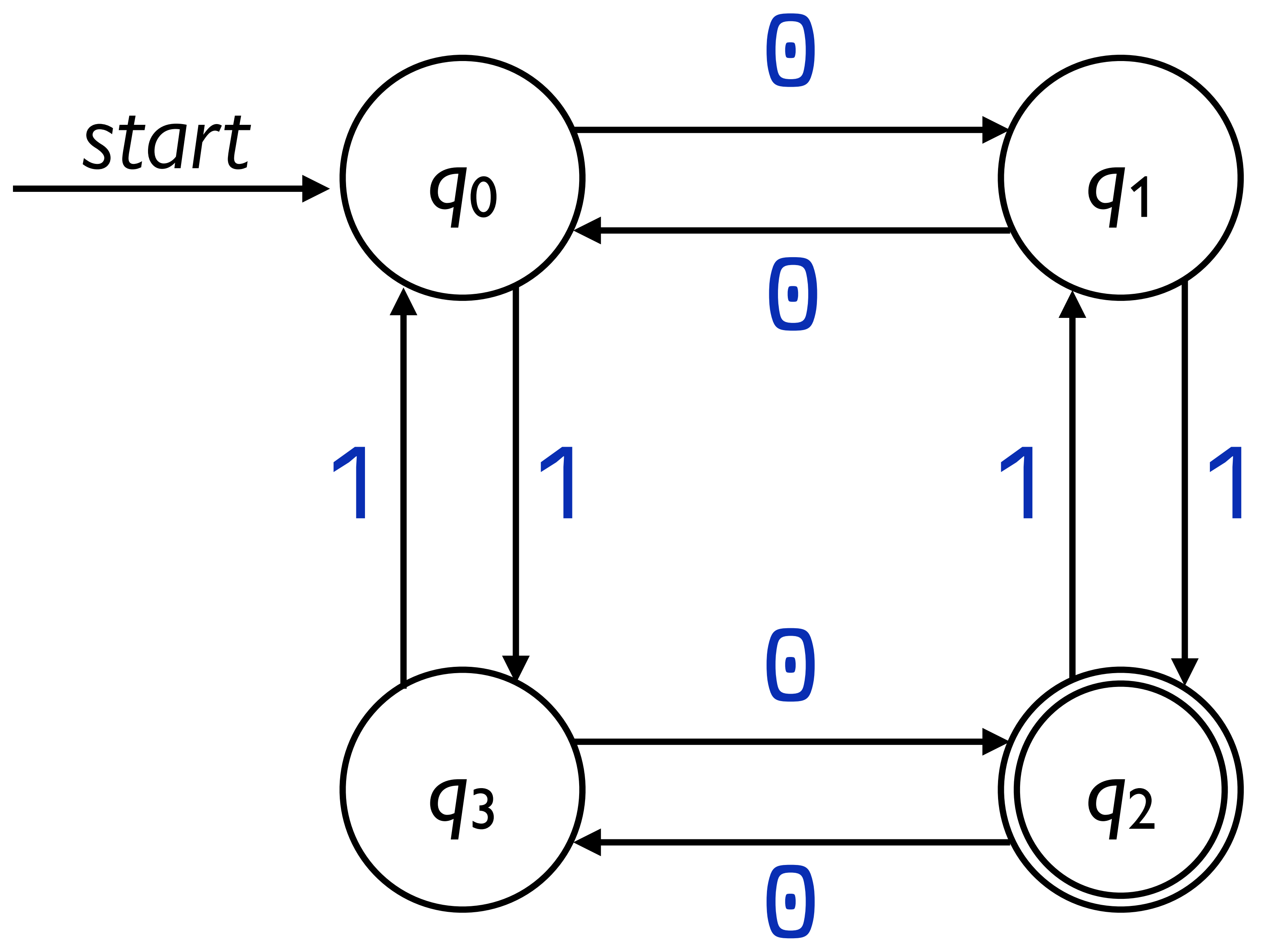
This is the “deterministic” part!

There is a unique start state.

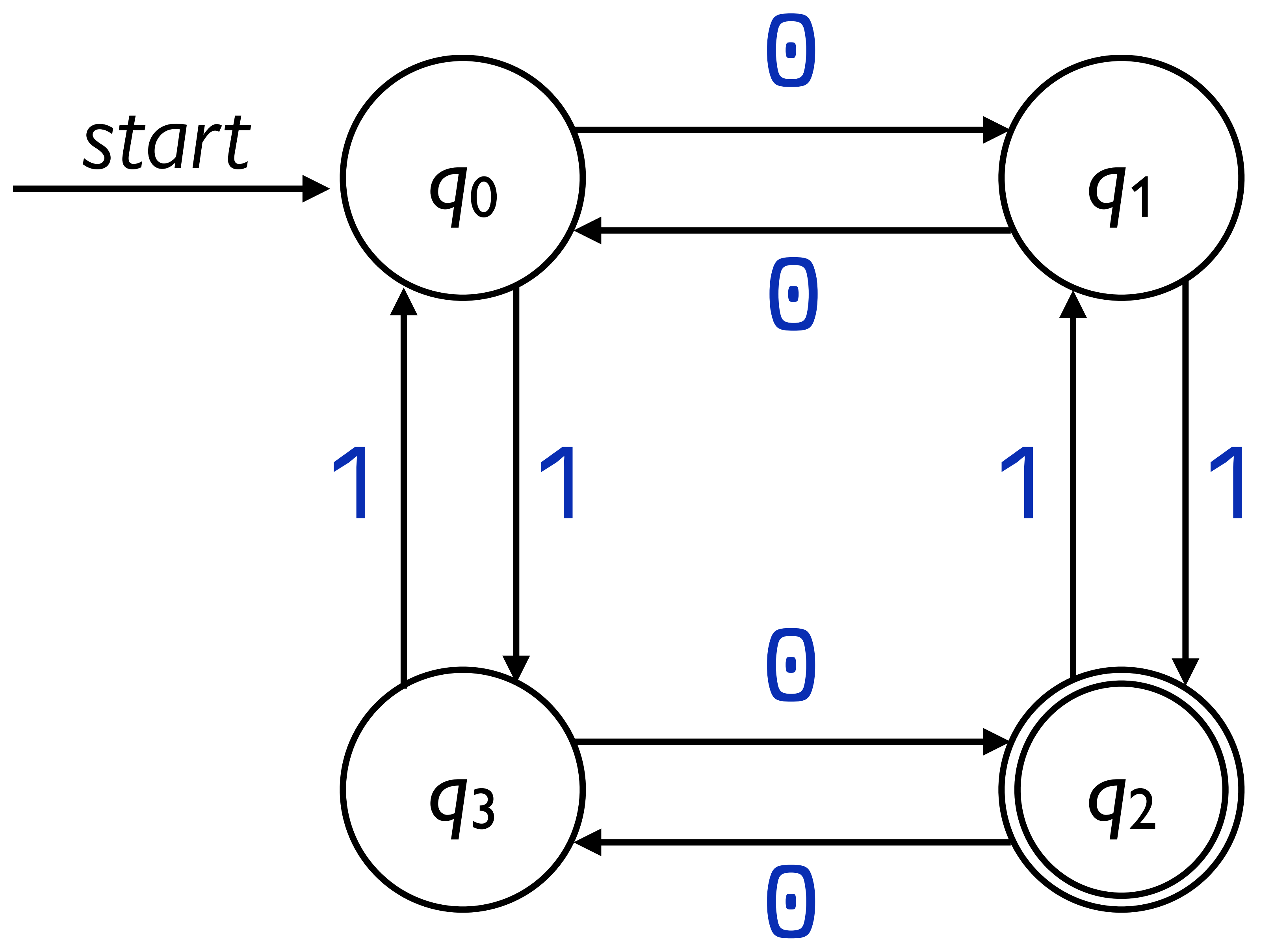
There are zero or more accept states.

## Is this a DFA over {`0`, `1`}?

## Is this a DFA over {`0`, `1`}?



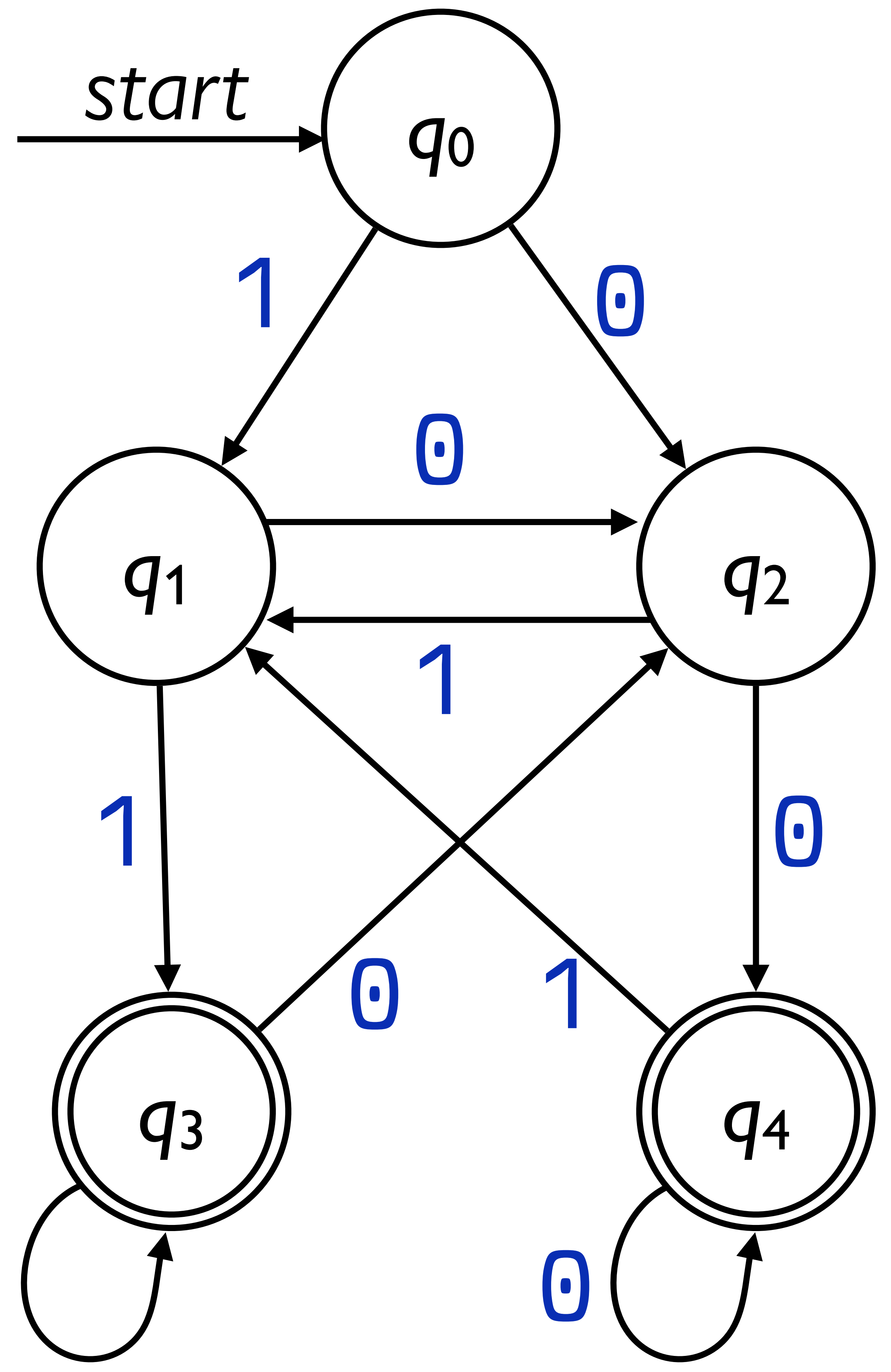
## Is this a DFA over {0, 1}?



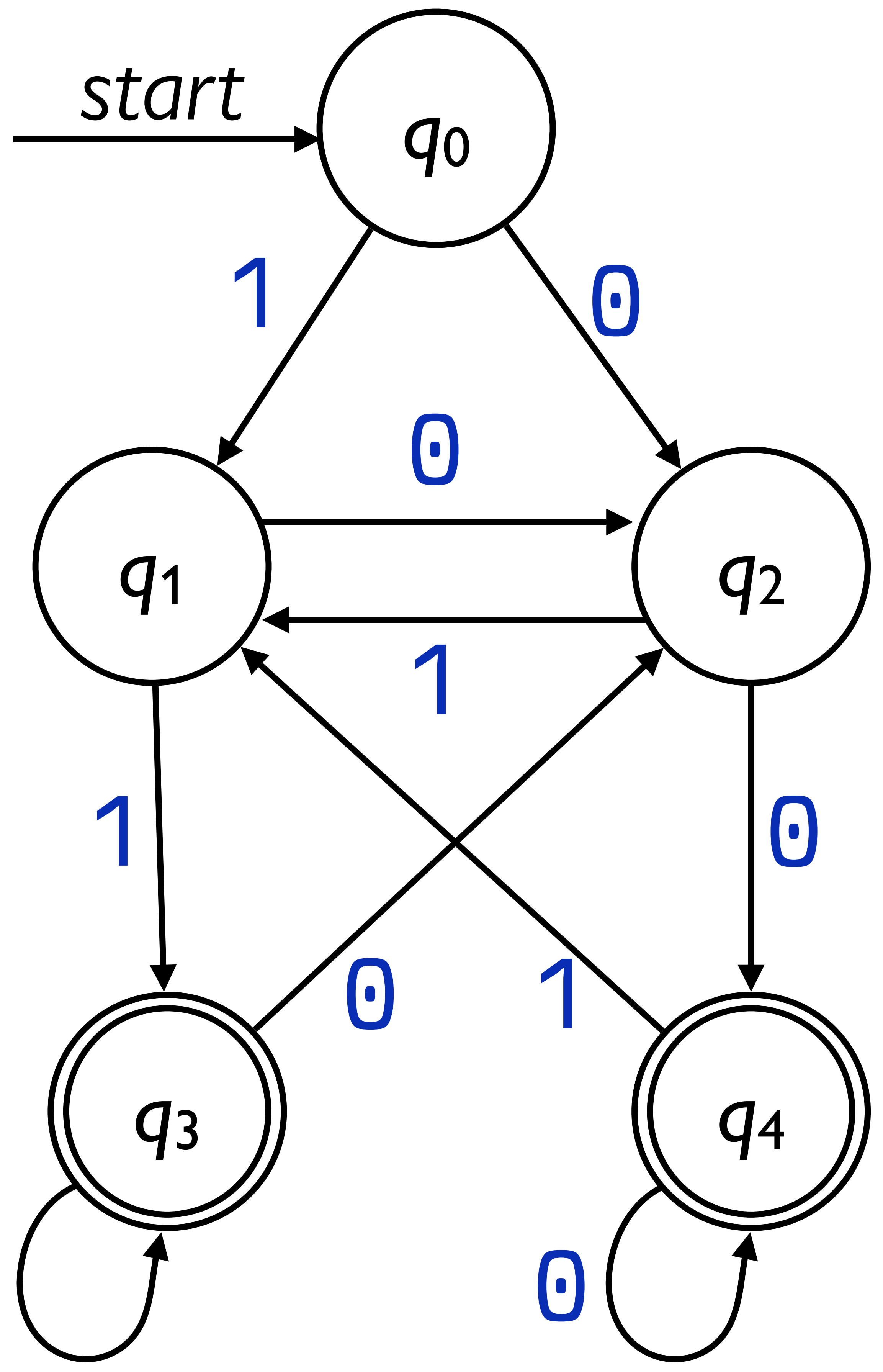
✓

## Is this a DFA over {`0`, `1`}?

## Is this a DFA over {`0`, `1`}?

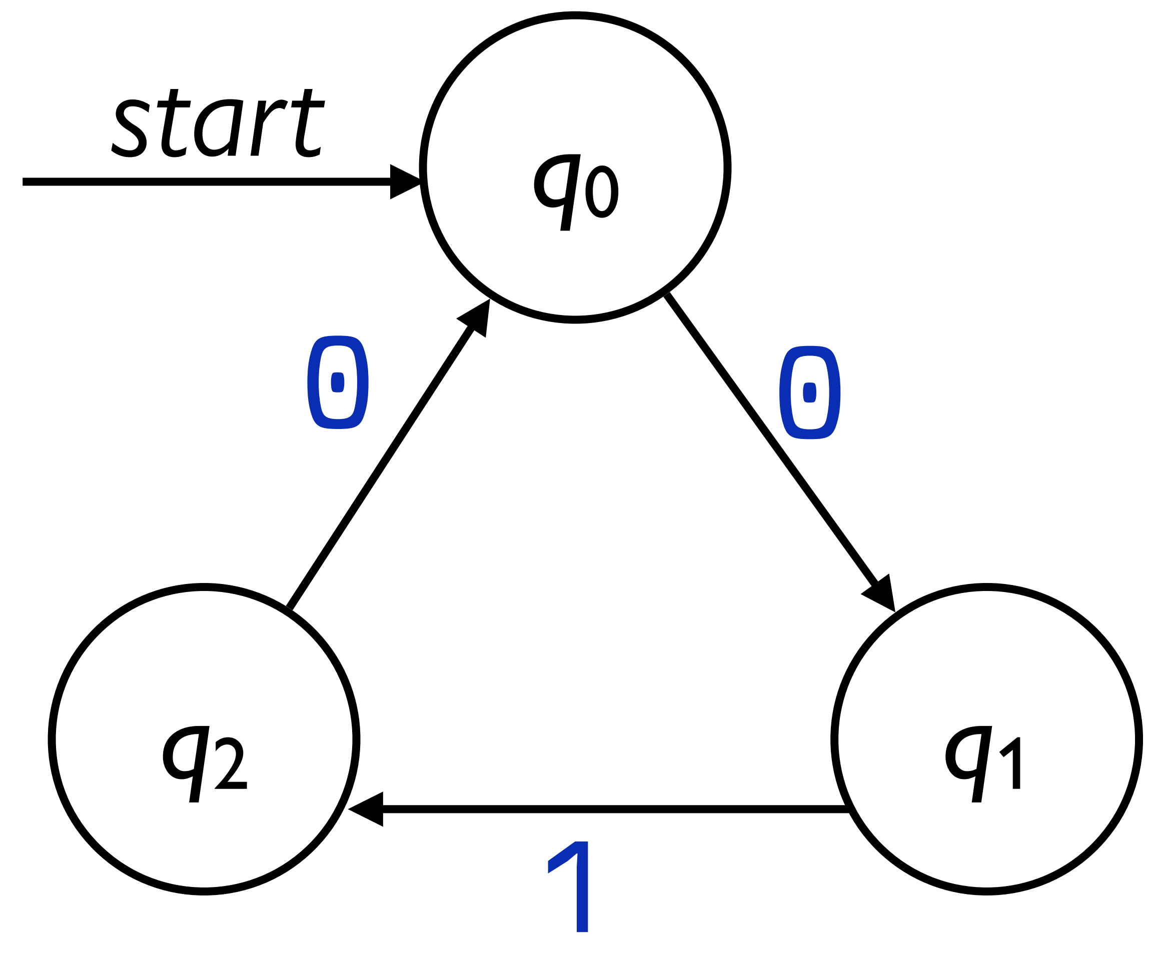


## Is this a DFA over {0, 1}?

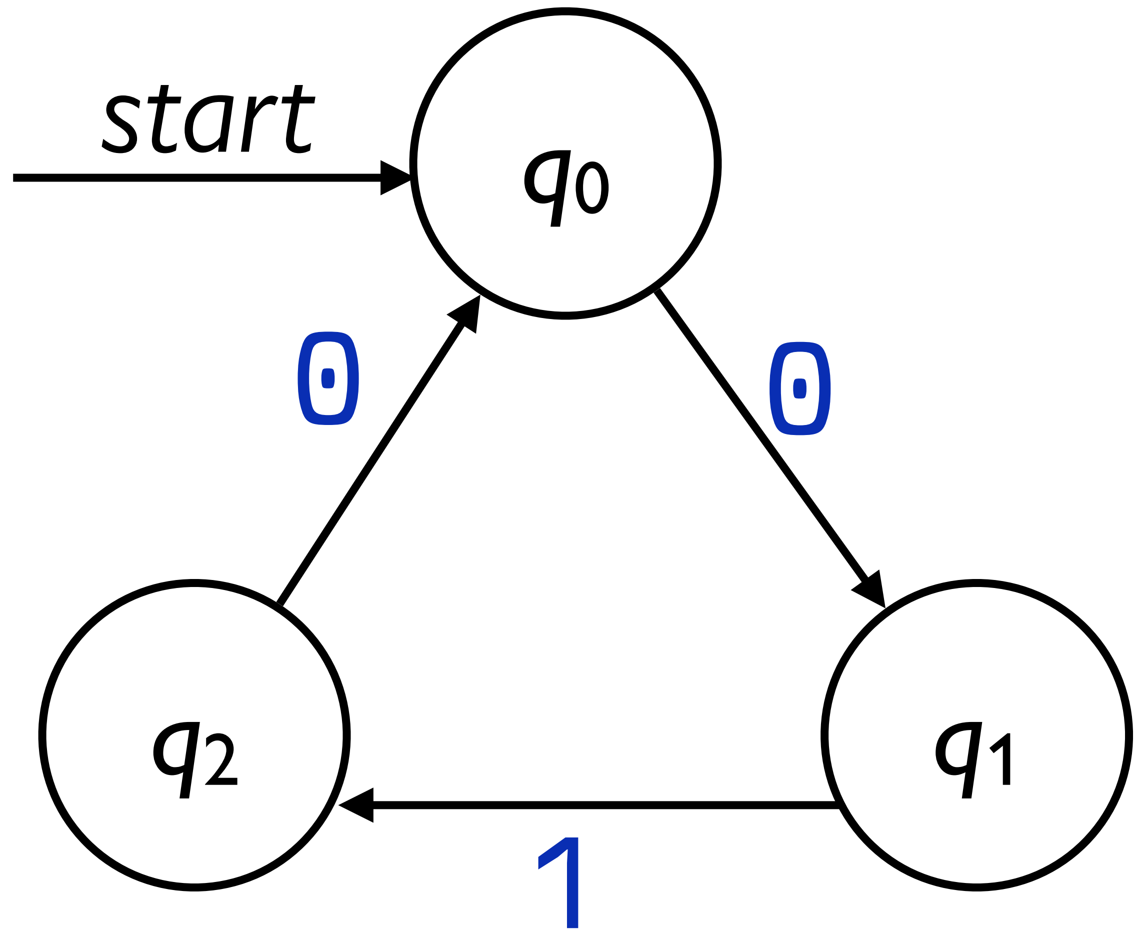


✓

## Is this a DFA over {`0`, `1`}?



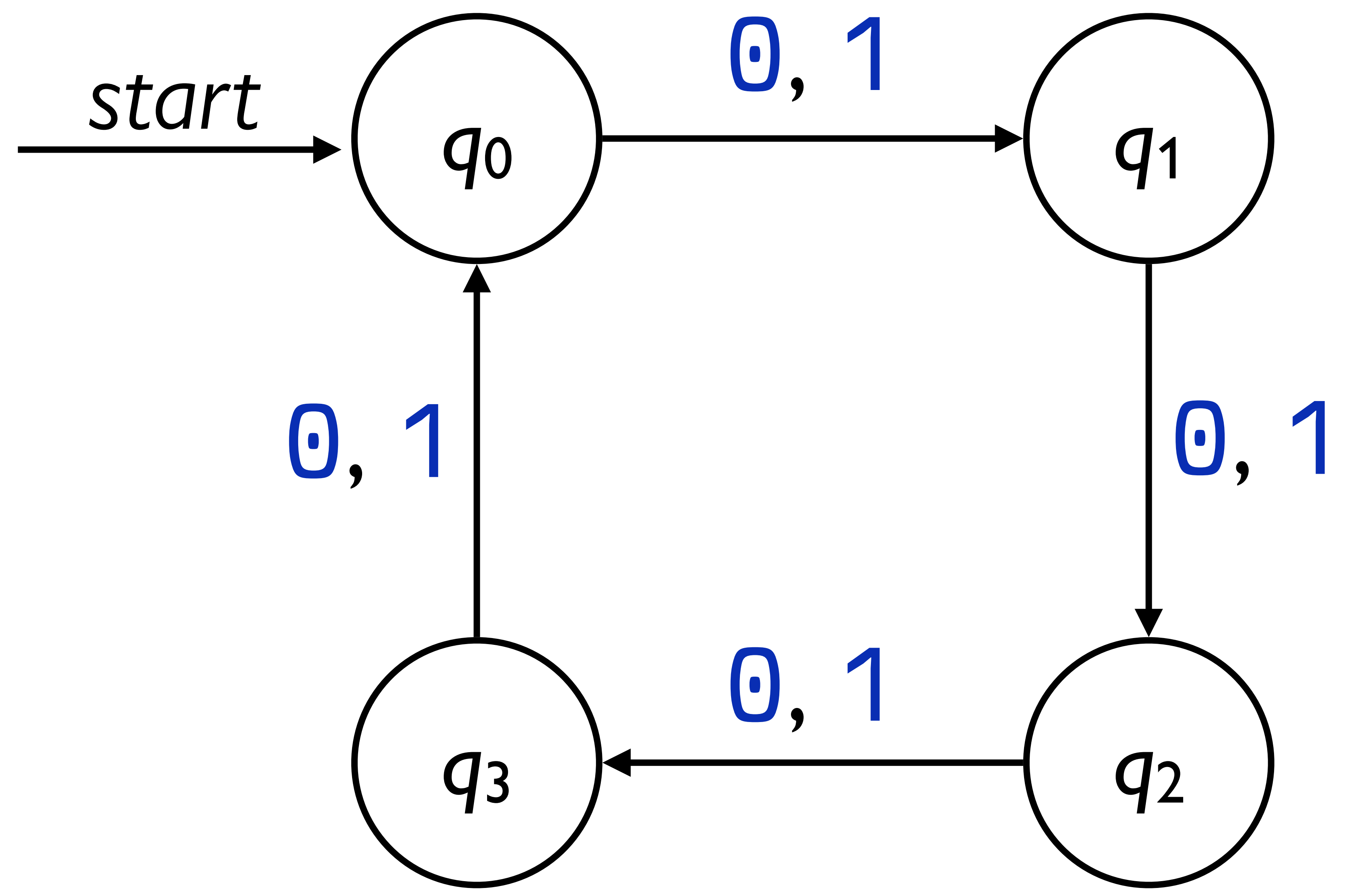
## Is this a DFA over {`0`, `1`}?



✗

## Is this a DFA over {`0`, `1`}?

## Is this a DFA over {`0`, `1`}?



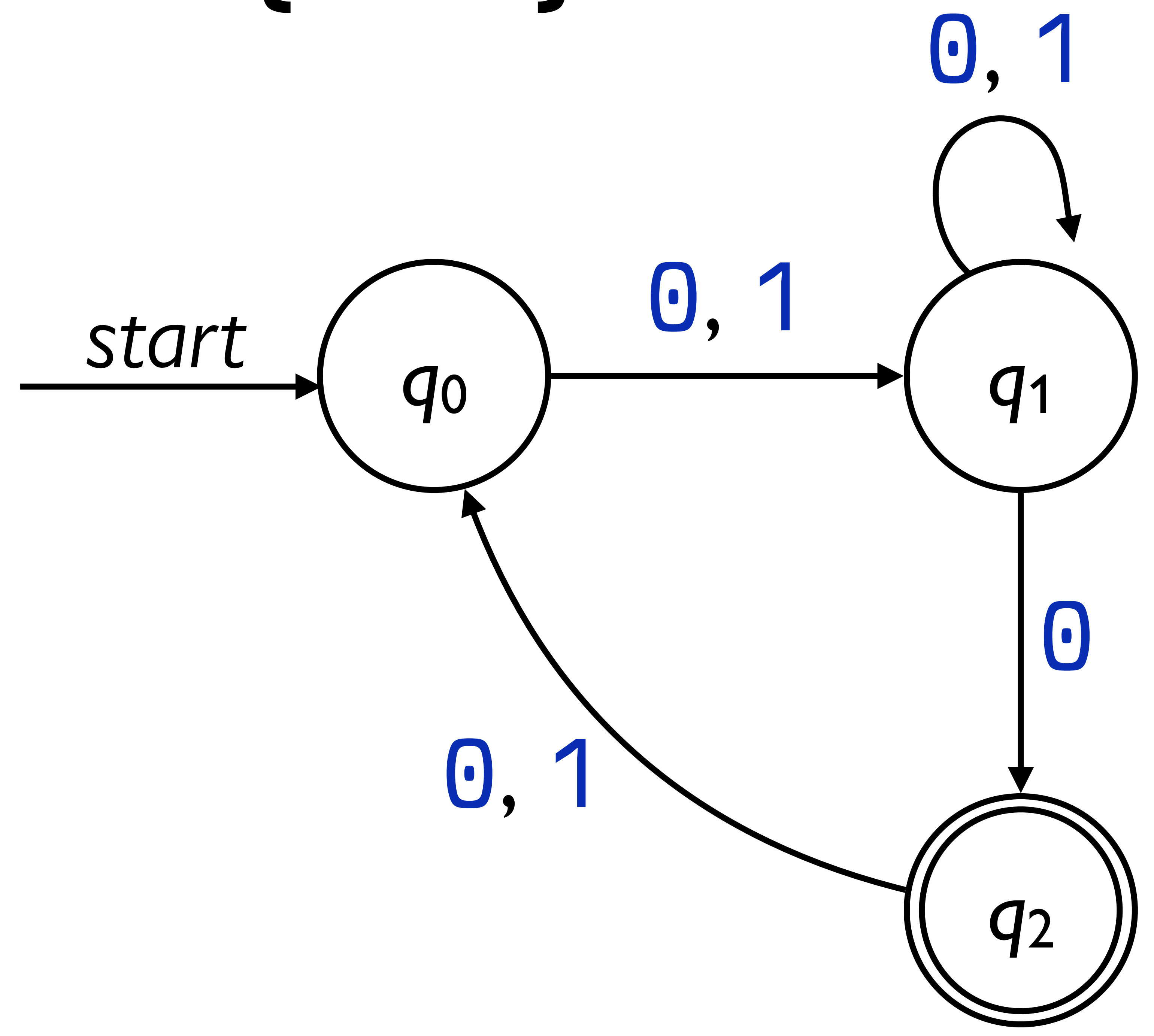
## Is this a DFA over {0, 1}?

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

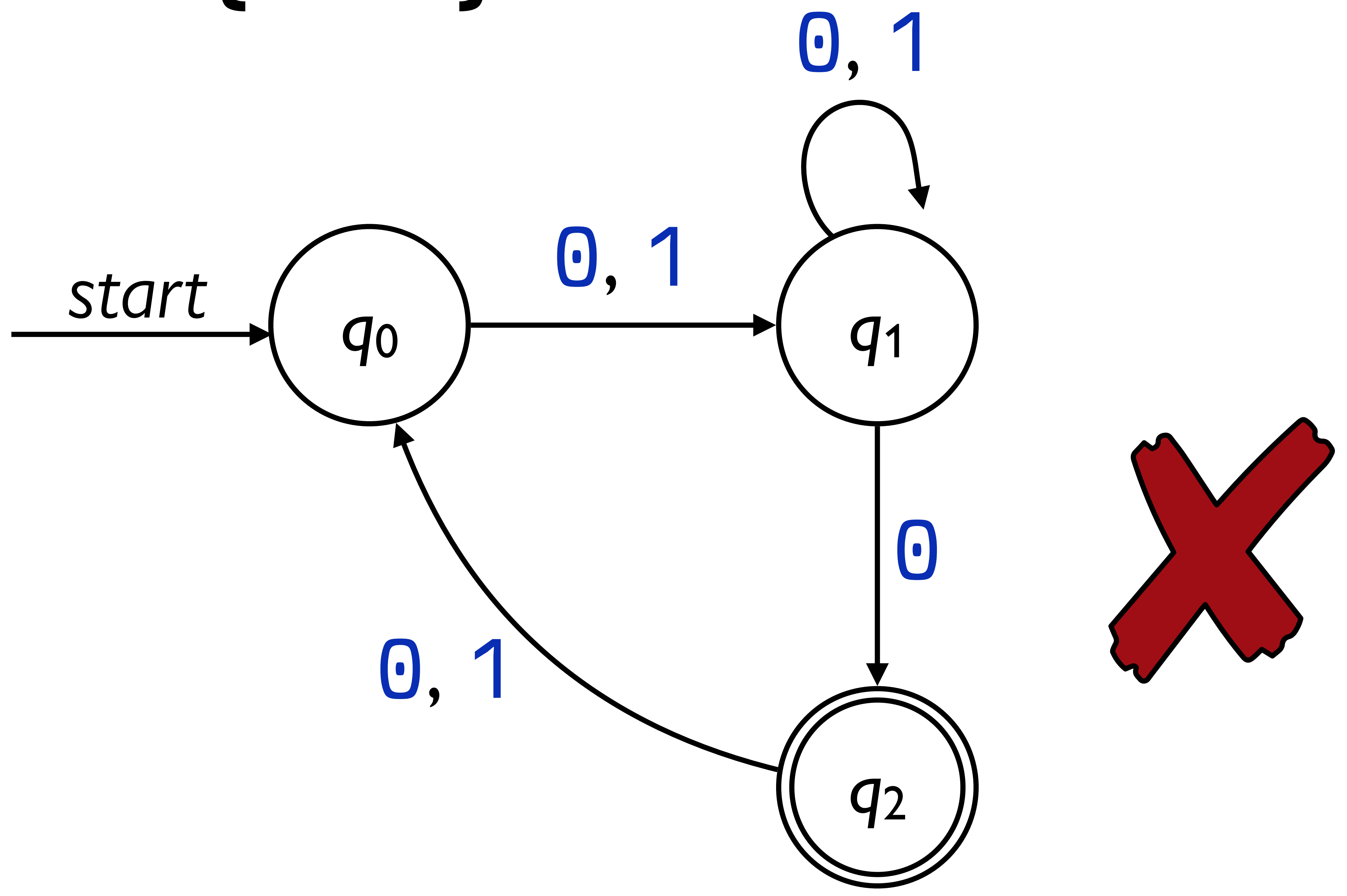
✓

## Is this a DFA over {`0`, `1`}?

## Is this a DFA over {0, 1}?



## Is this a DFA over {**0**, **1**}?



## Exercise

Design a finite automaton to recognize decimal numbers.

## Acknowledgments

This lecture incorporates material from:

Phyllis Frankl, New York University

W. Daniel Hillis, *The Pattern on the Stone*

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*

Jeffrey Ullman, Stanford University