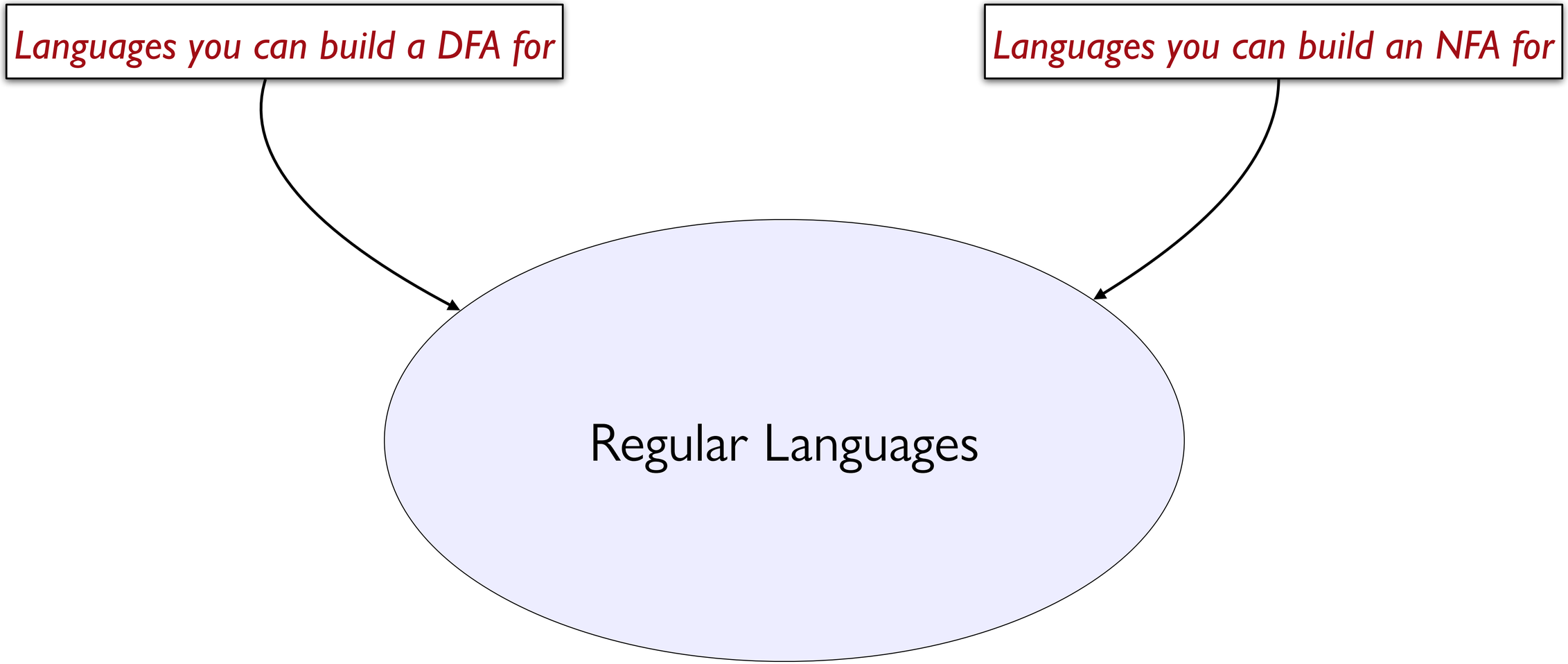
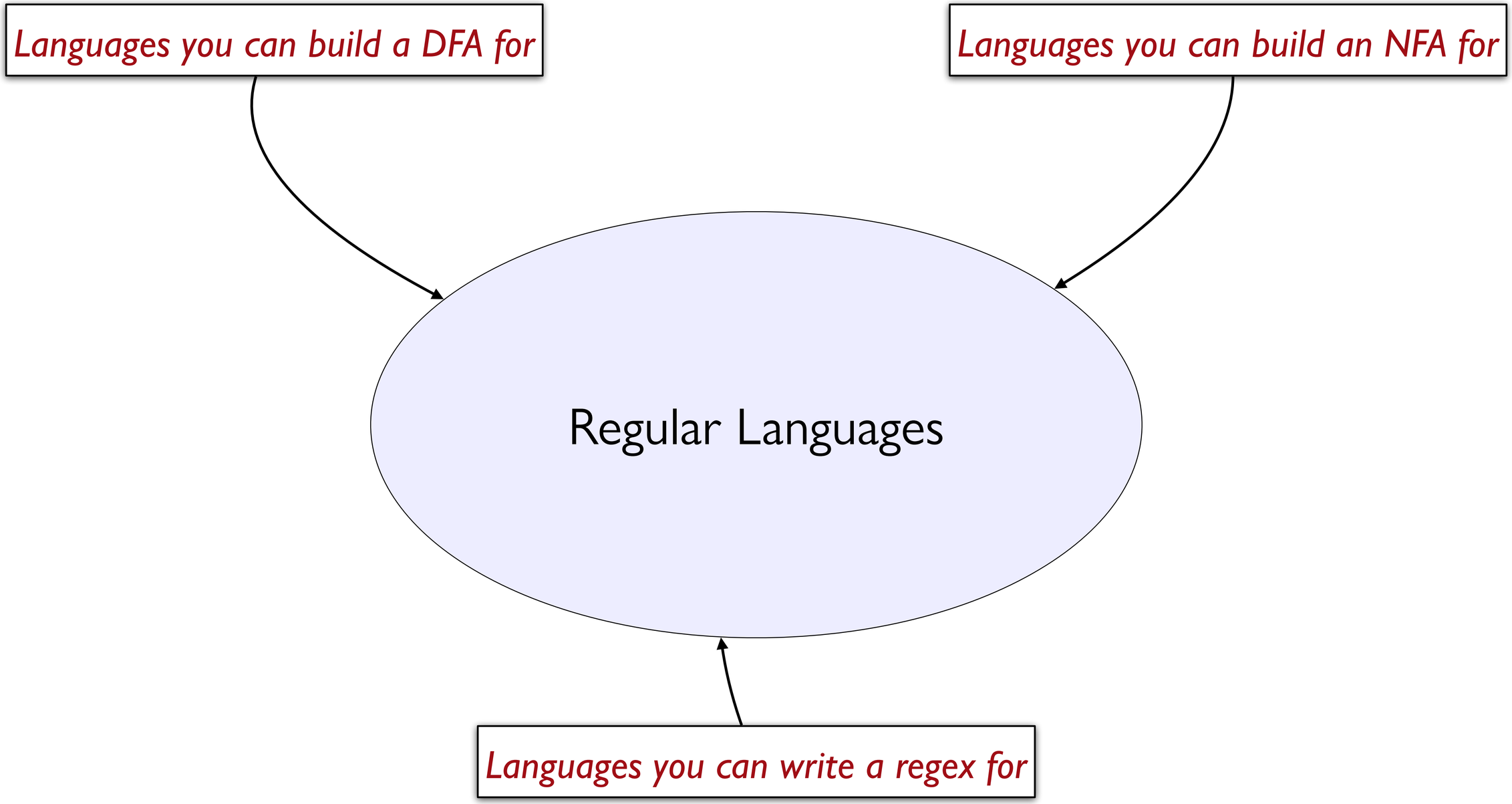
CMPU 240 · Theory of Computation

The Limits of Regular Languages

17 February 2026

Where we left off…


Languages you can build a DFA for
Languages you can build an NFA for
Regular Languages


Languages you can build a DFA for
Languages you can build an NFA for
Regular Languages
Languages you can write a regex for

There are two directions to prove:

If R is a regular expression, then $L(R)$ is a regular language.

If L is a regular language, we can write a regular expression for it.

THEOREM If R is a regular expression, then $L(R)$ is regular.

PROOF IDEA Use induction!

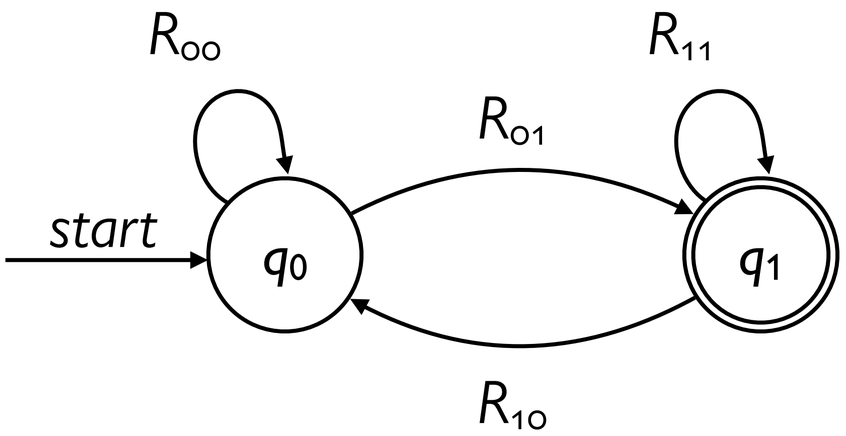
The atomic regular expressions all represent regular languages.

The combination steps represent closure properties.

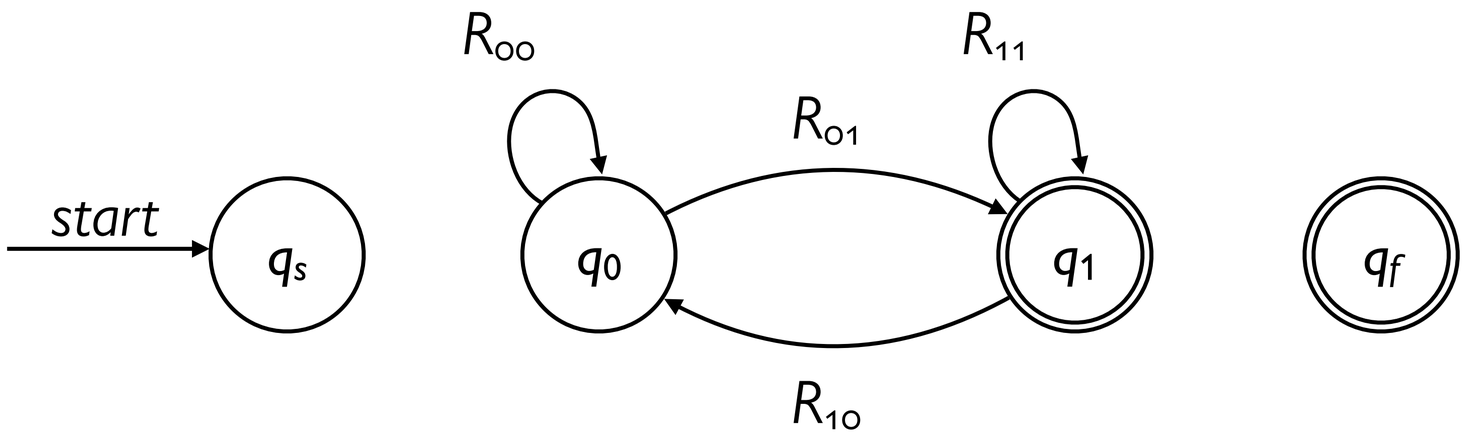
So, anything you can make from them must be regular!

THEOREM If L is a regular language, then there is a regular expression for L.

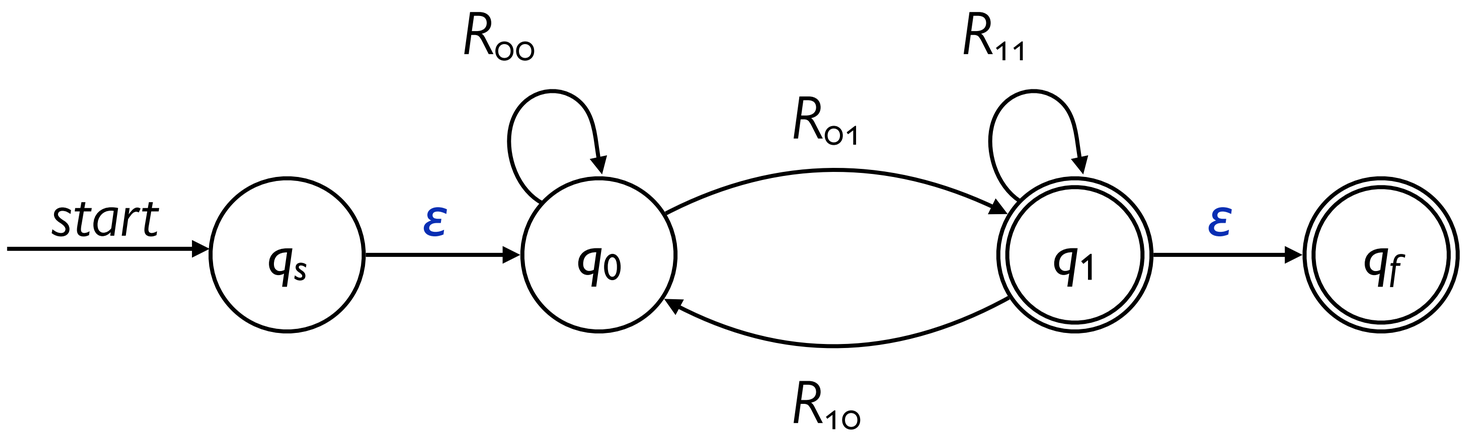
PROOF IDEA Show how to convert an arbitrary NFA into a regular expression.


R_{00}
R_{11}
R_{01}
start
q_0
q_1
R_{10}

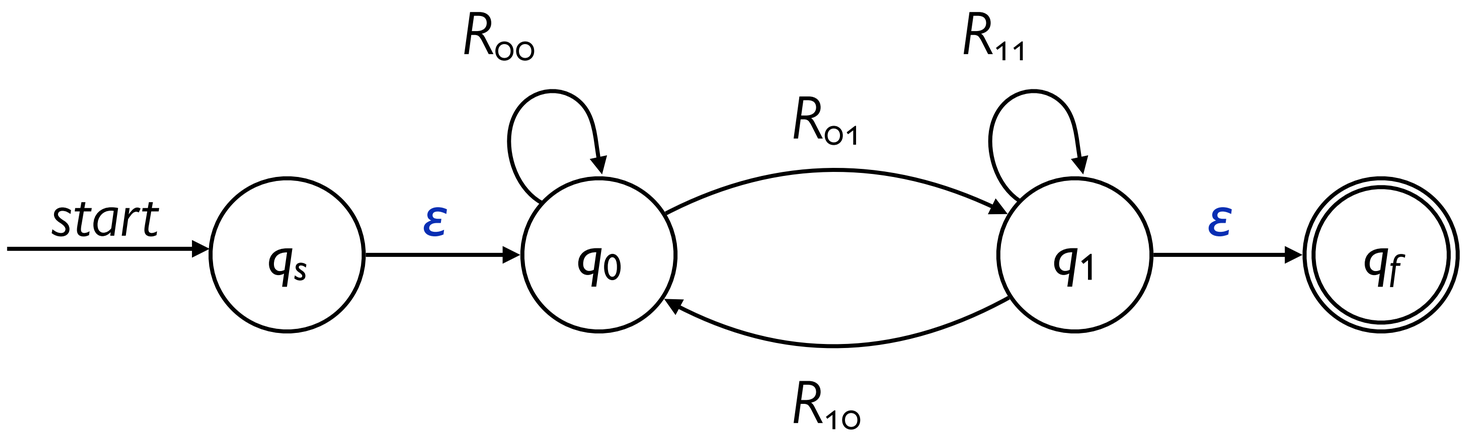
First add new start and accept states



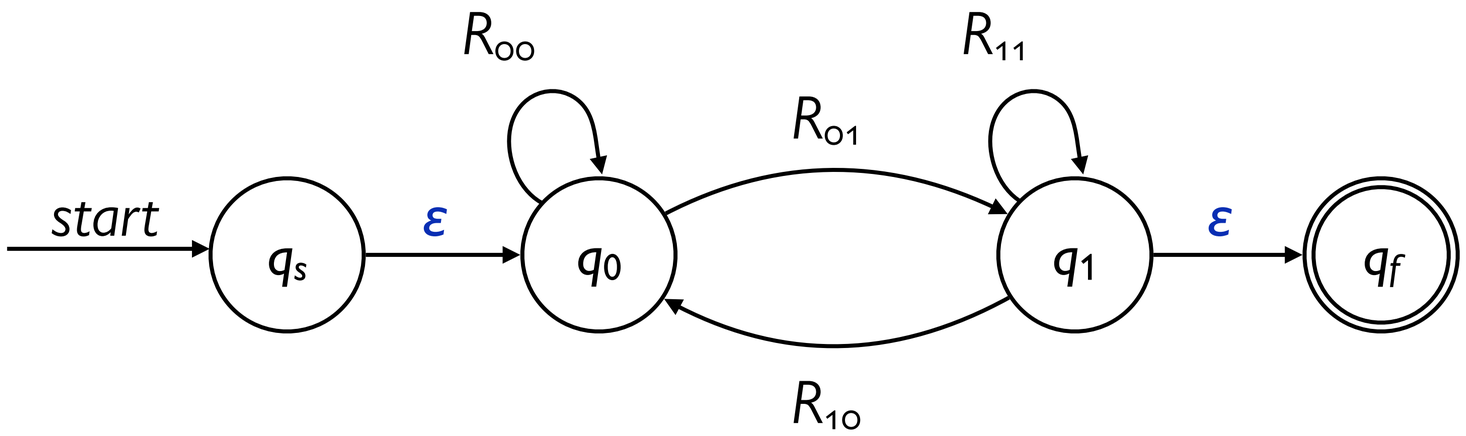
First add new start and accept states

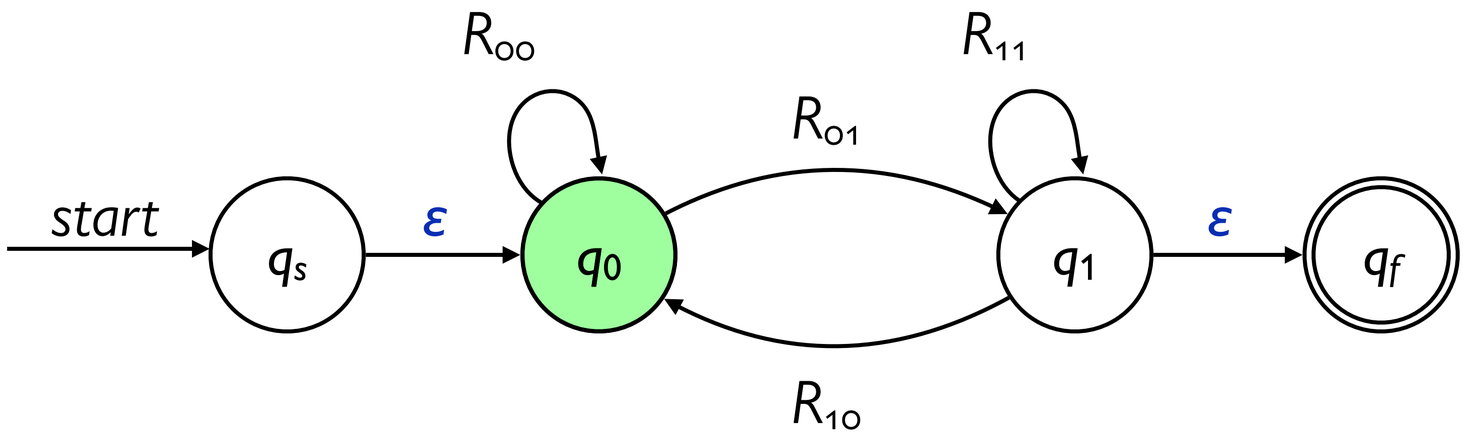


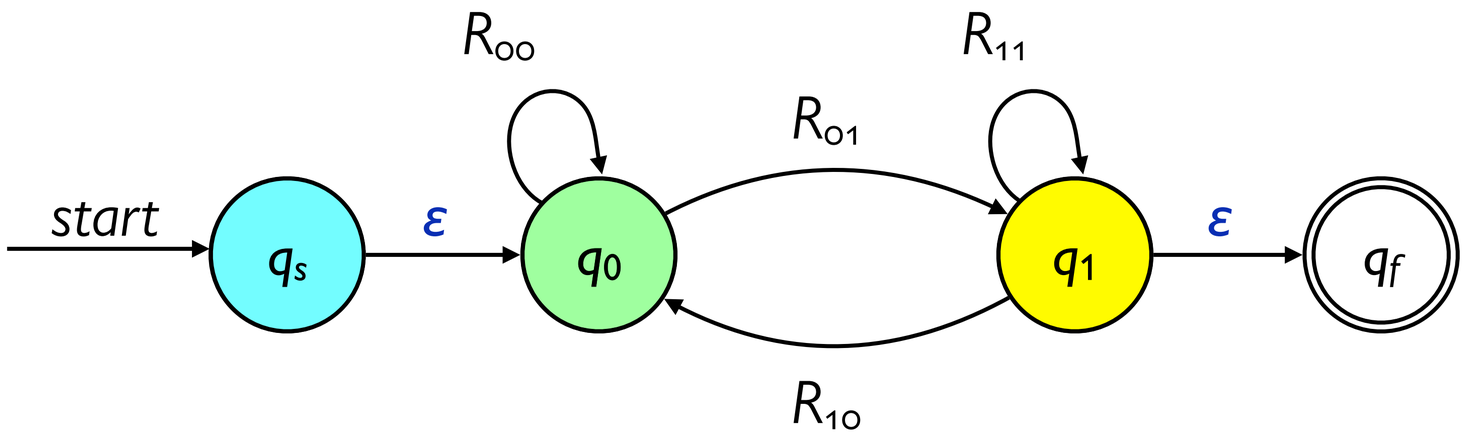
First add new start and accept states

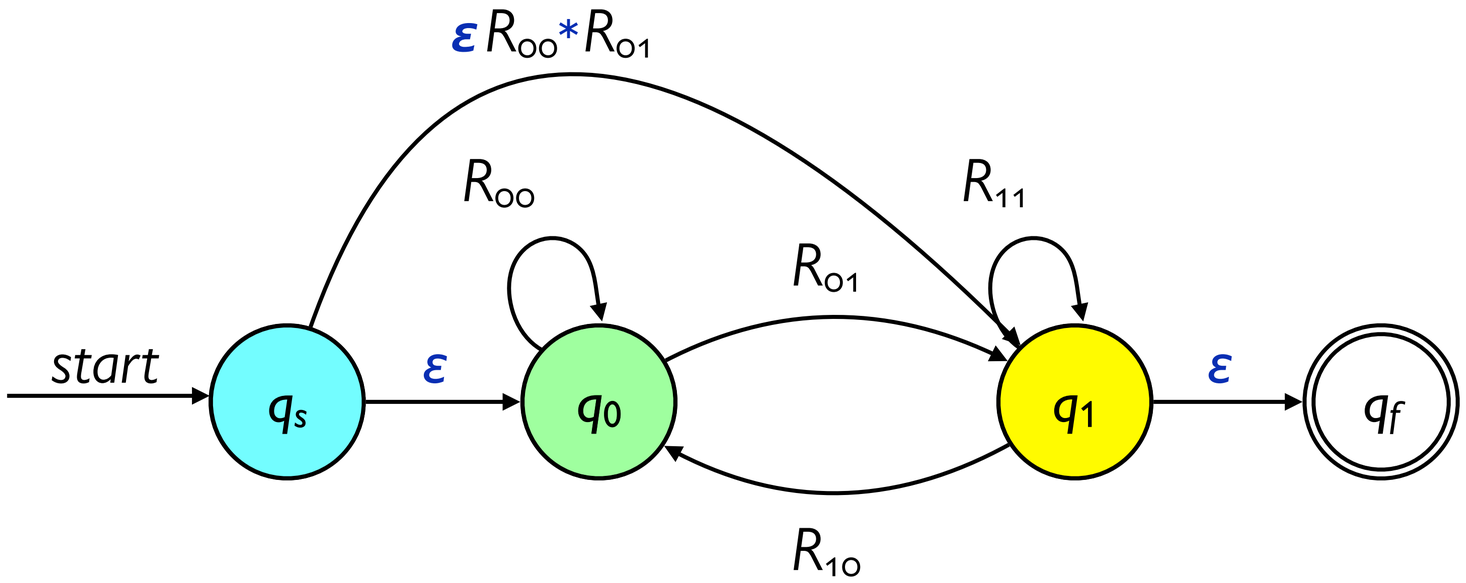


Then eliminate each of the original states, adding new transitions that capture the ways of moving through them.

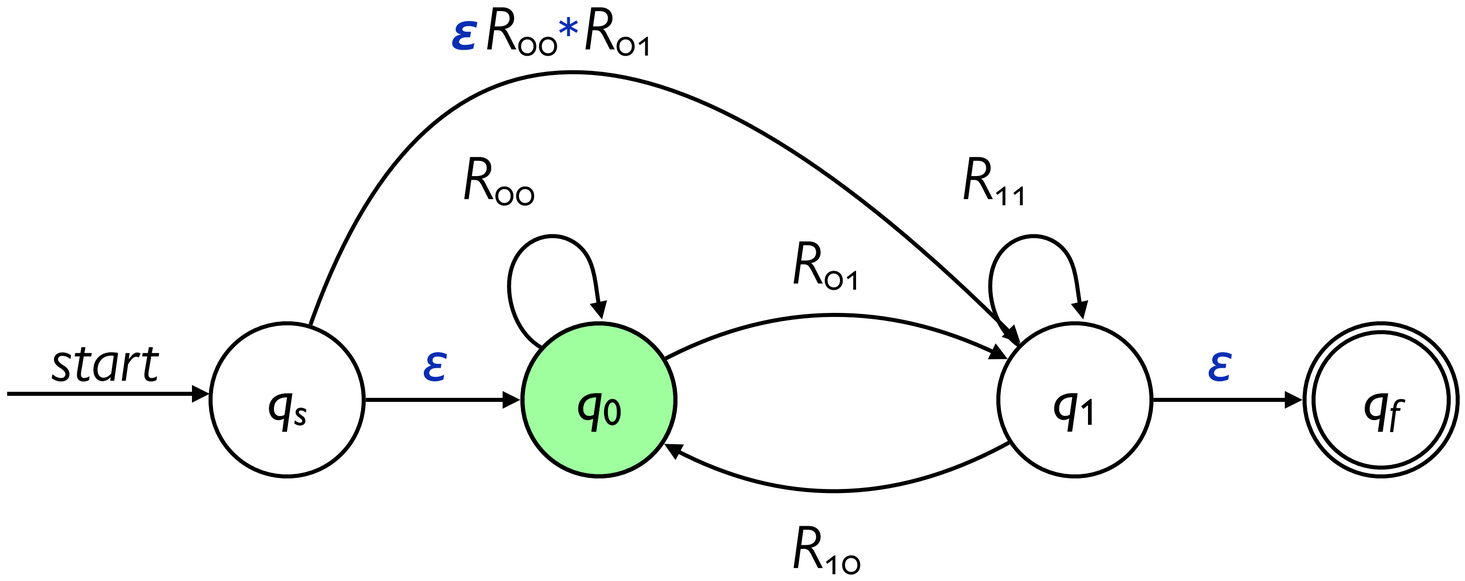


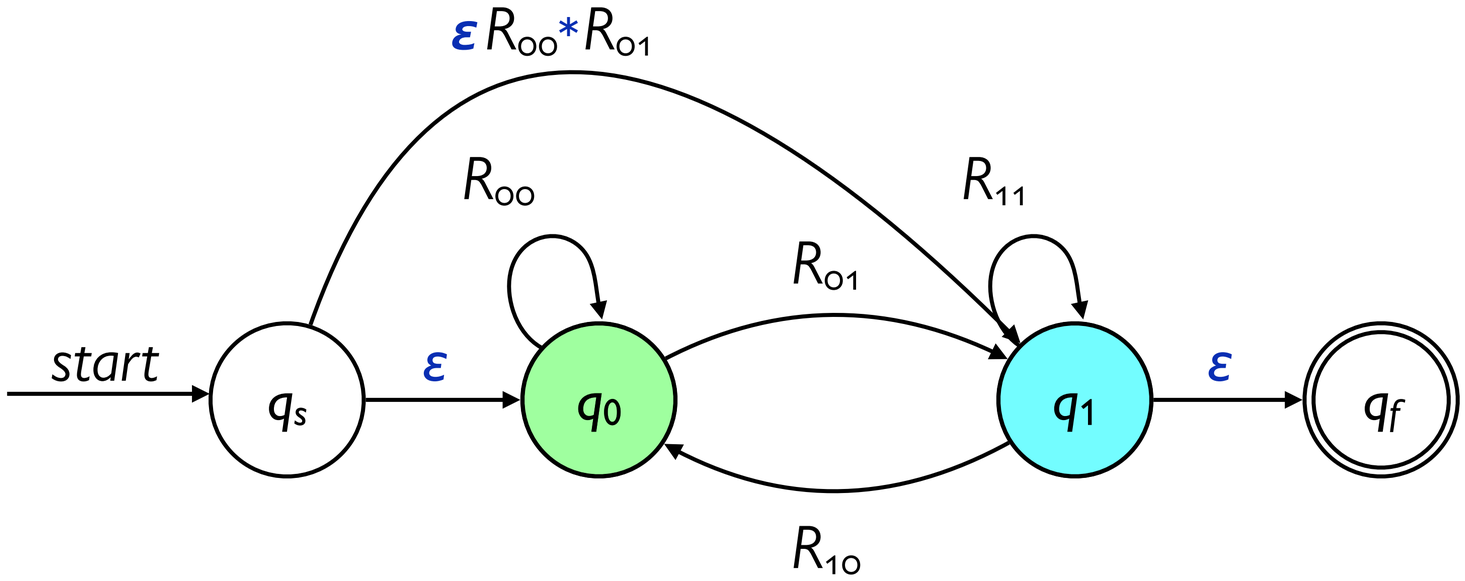

R_{00}
R_{11}
R_{01}
start
q_s
ε
q_0
q_1
ε
q_f
R_{10}

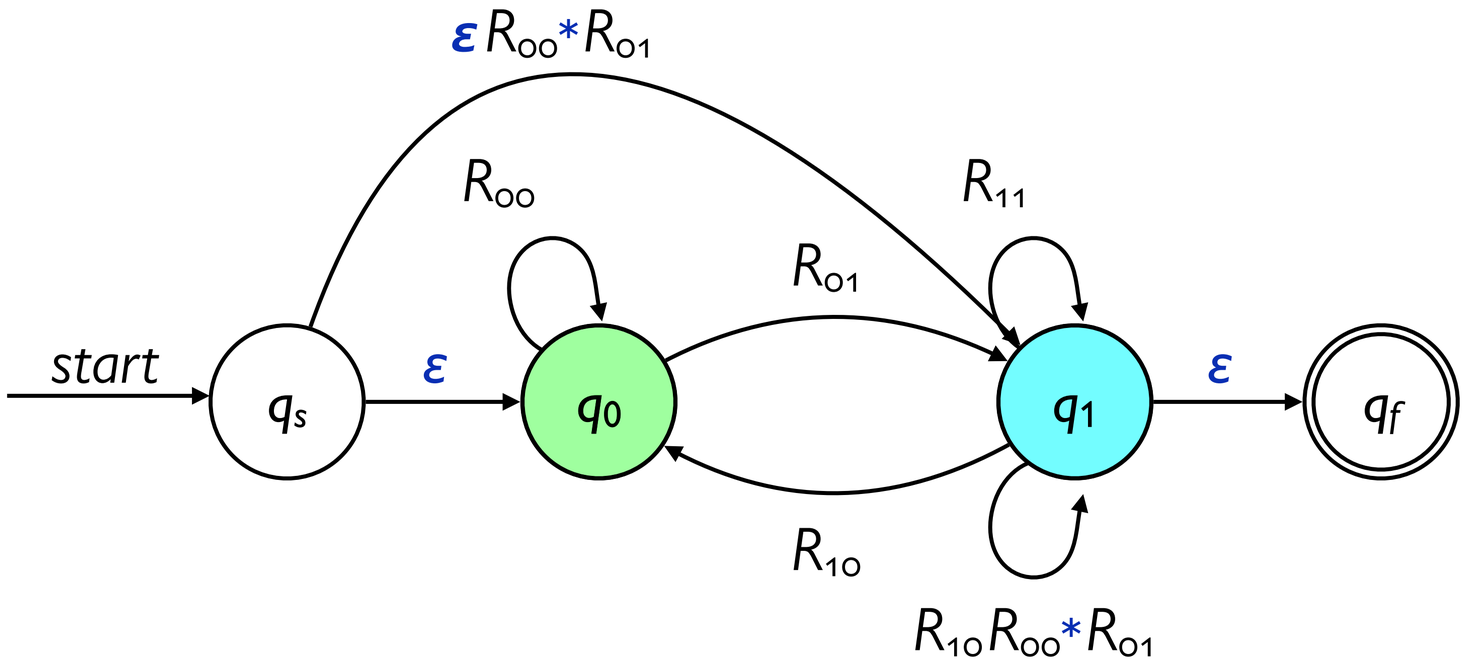

R_{00}
R_{11}
R_{01}
start
q_s
ε
q_0
q_1
ε
q_f
R_{10}

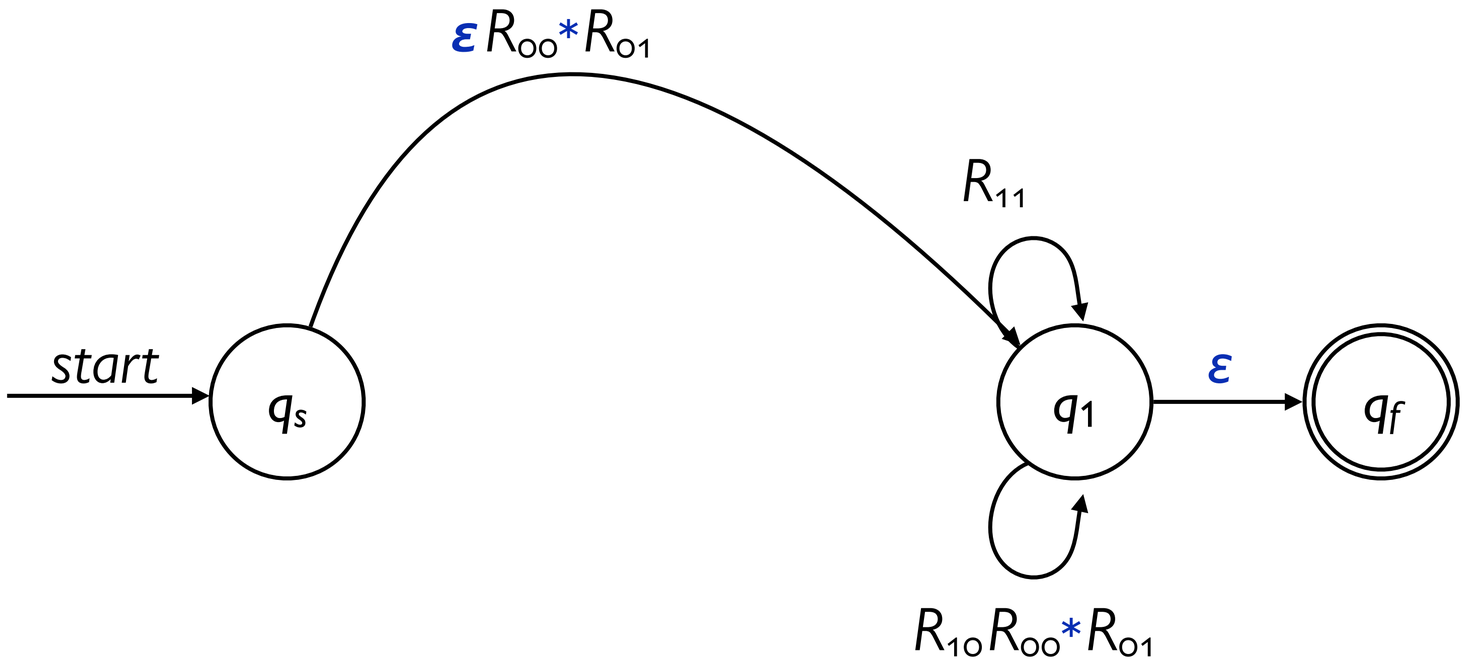


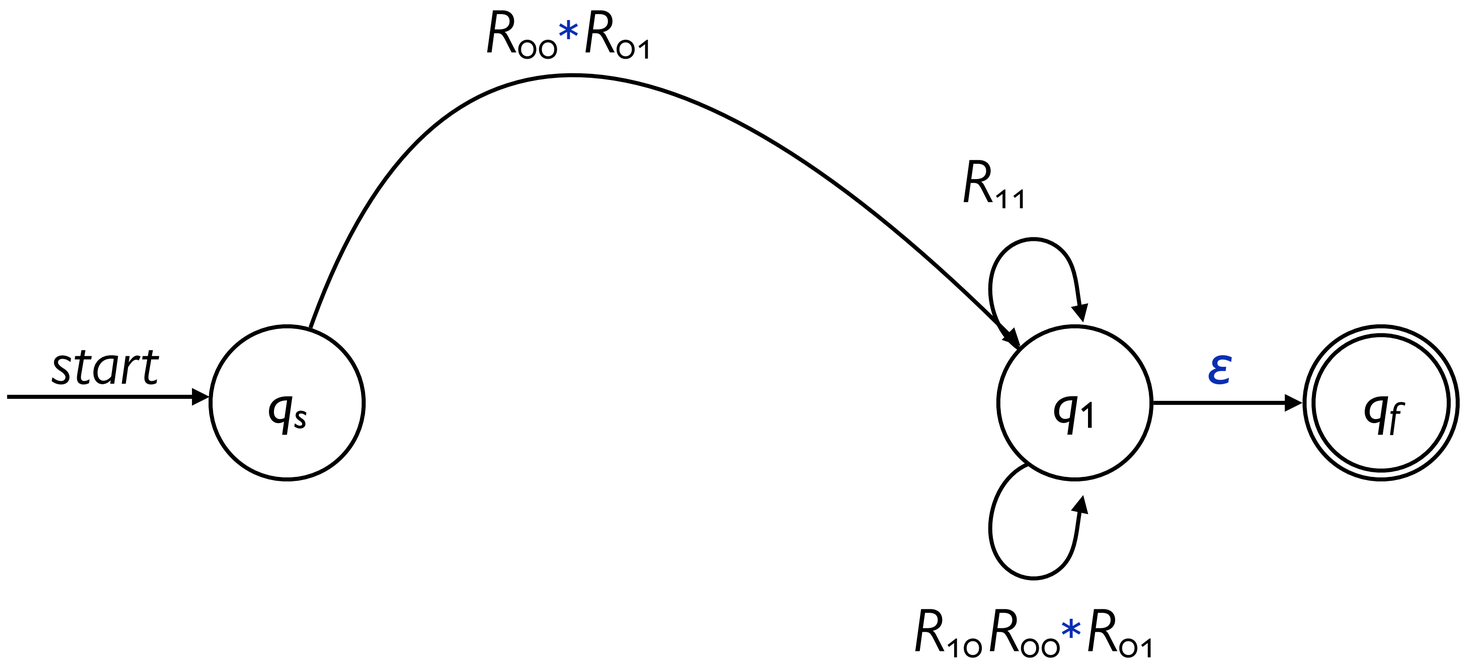
We can use concatenation and Kleene closure to skip this state.

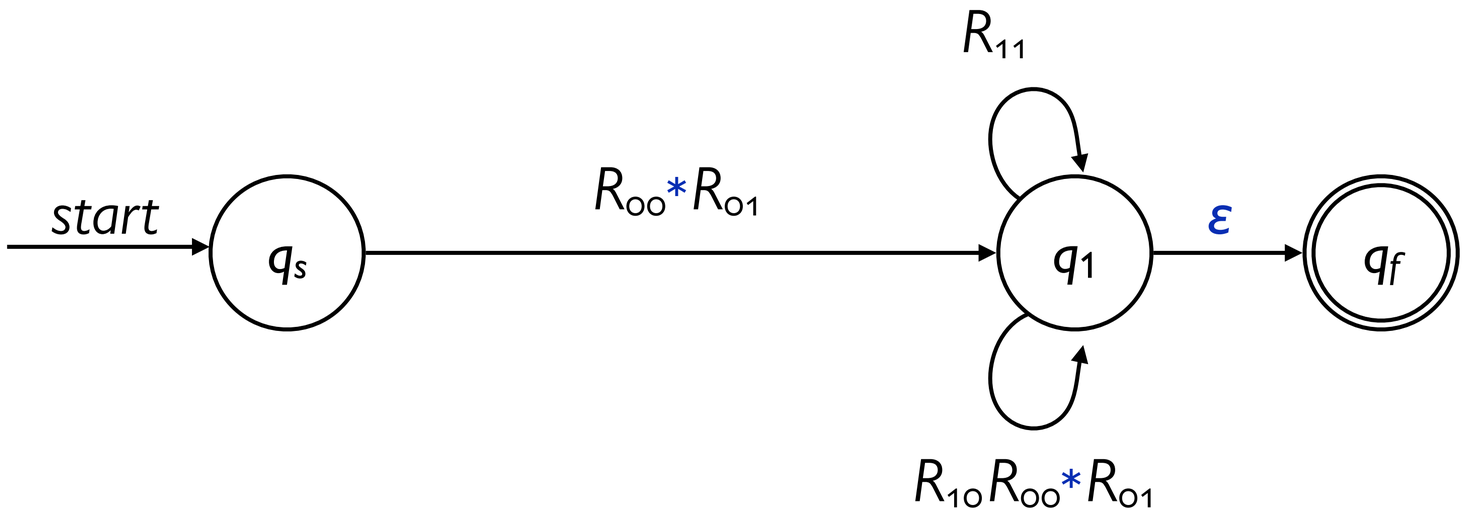

ε R_{00}*R_{01}
R_{00}
R_{11}
R_{01}
start
q_s
ε
q_0
q_1
ε
q_f
R_{10}

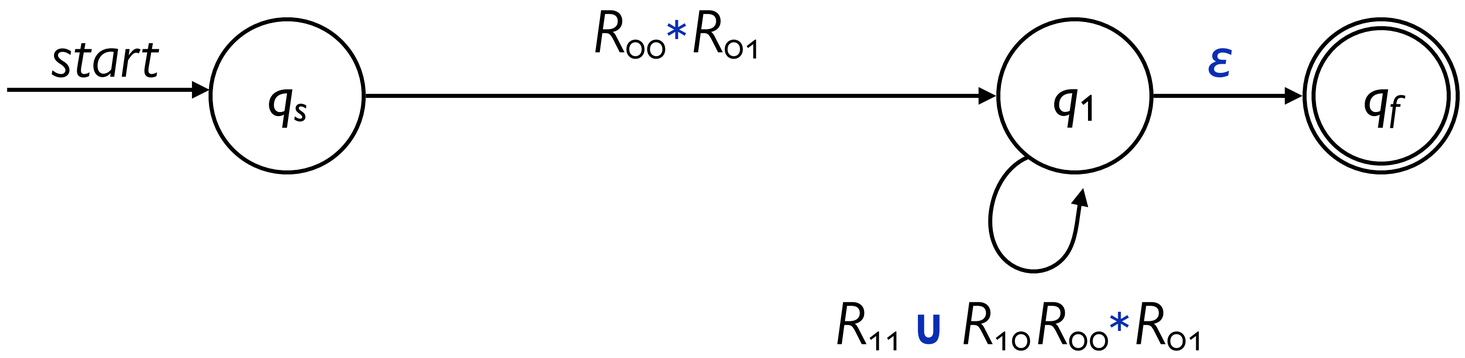

ε R_{00}*R_{01}
R_{00}
R_{11}
R_{01}
start
q_s
ε
q_0
q_1
ε
q_f
R_{10}


ε R00*R01
R00
R11
R01
start
qs
ε
q0
q1
ε
qf
R10
R10R00*R01


ε R00* R01
R11
start
qs
q1
ε
qf
R10 R00* R01

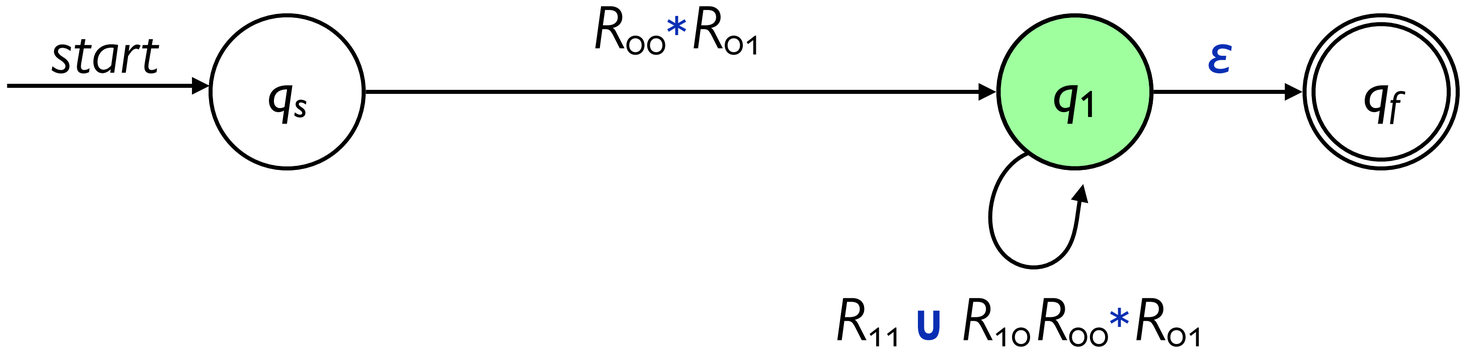

$R_{00}{}^{*}R_{01}$
R_{11}
start
q_s
q_1
ε
q_f
$R_{10}R_{00}{}^{*}R_{01}$


start
q_s
$R_{00}{}^*R_{01}$
R_{11}
q_1
ε
q_f
$R_{10}R_{00}{}^*R_{01}$

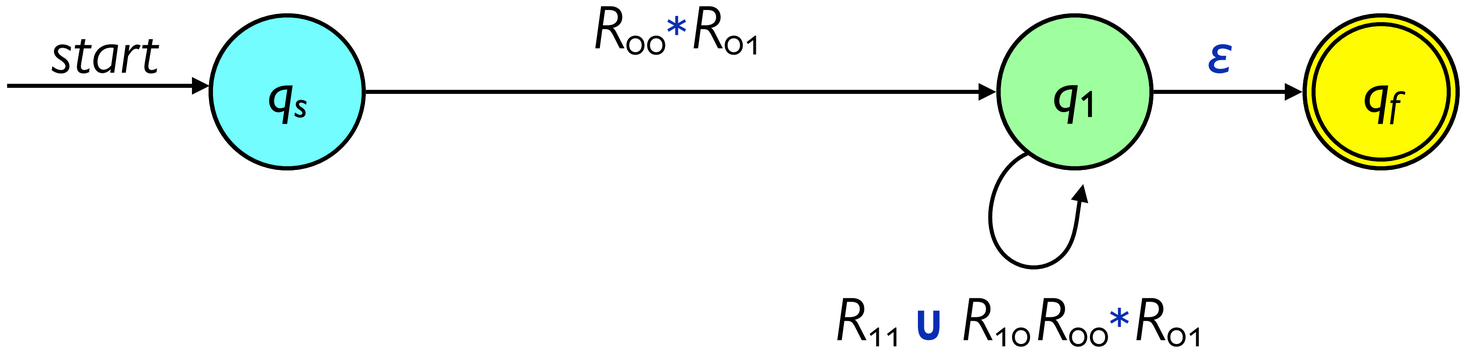


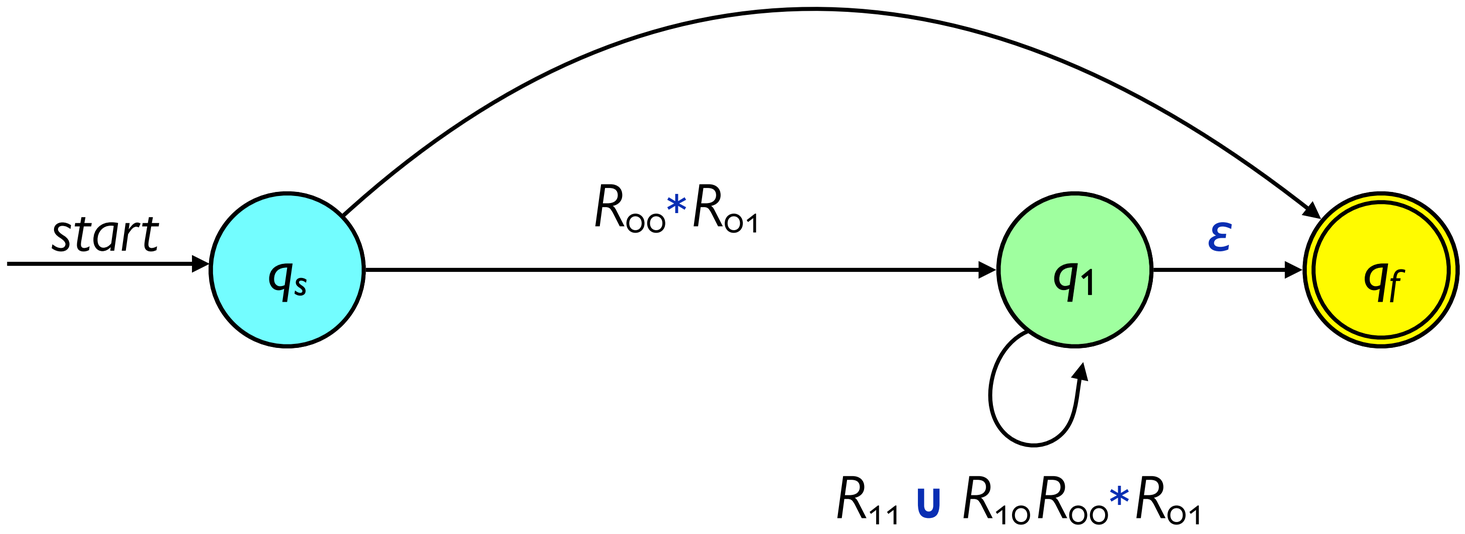
We can use union to combine these transitions.

Could we eliminate this state from the GNFA?

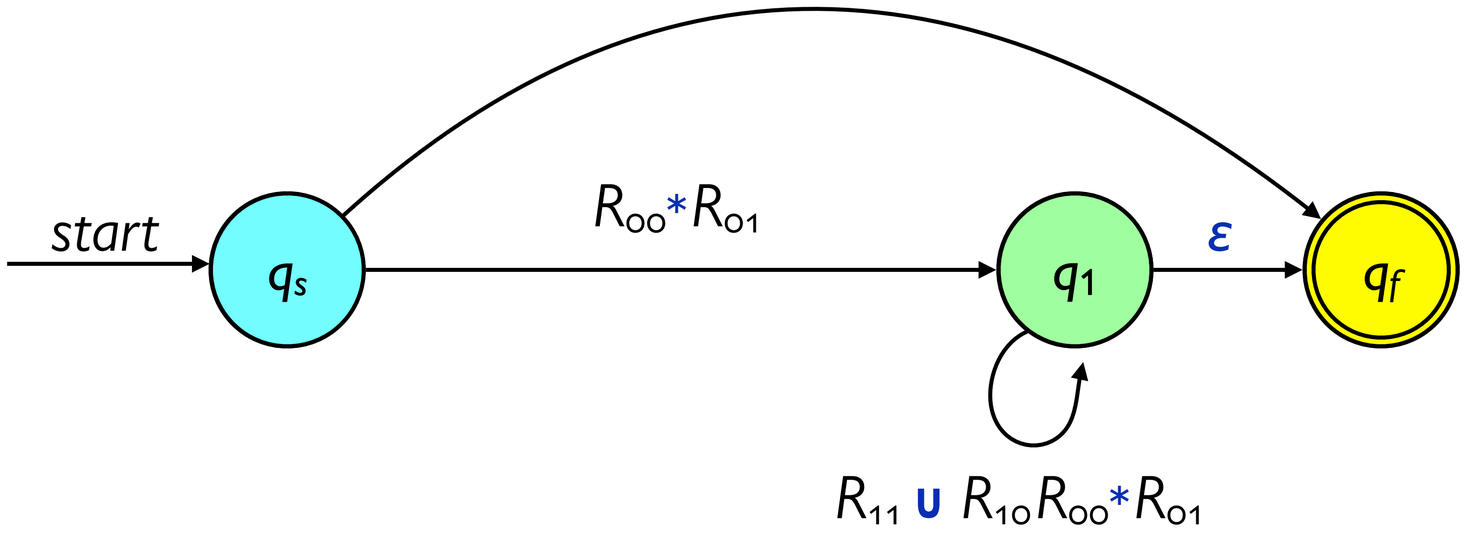


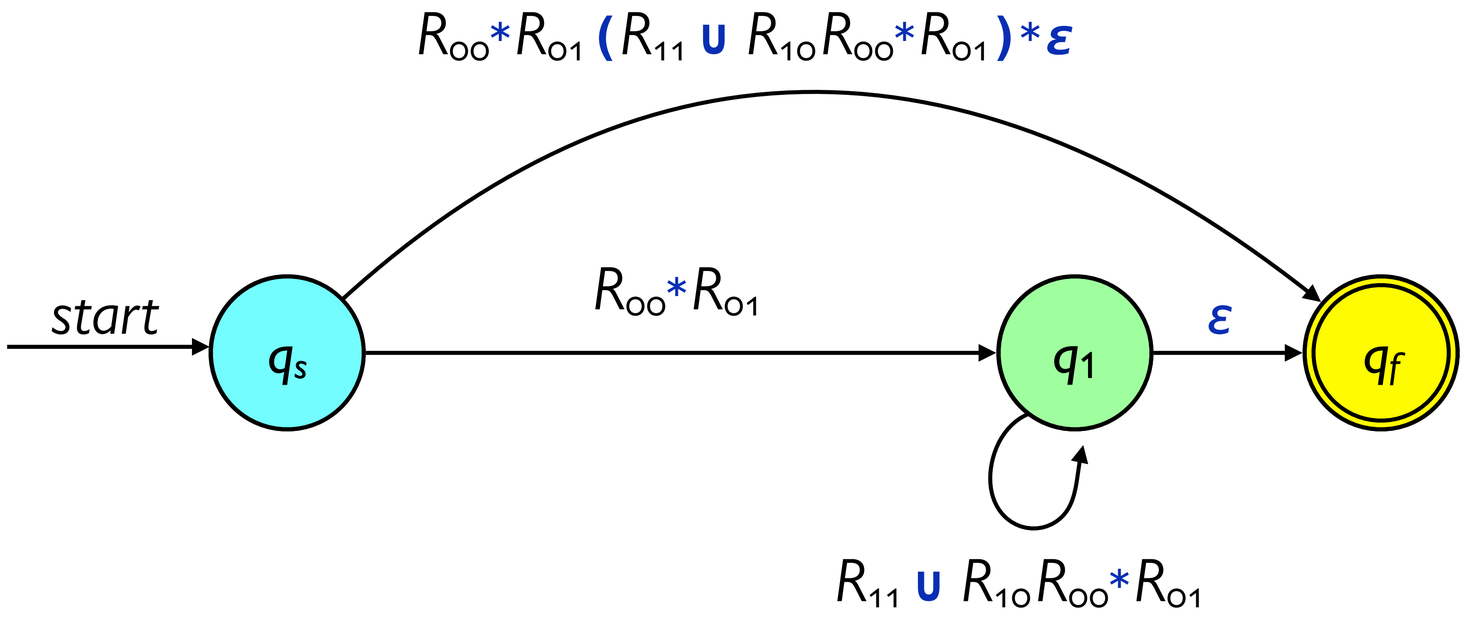
Could we eliminate this state from the GNFA?

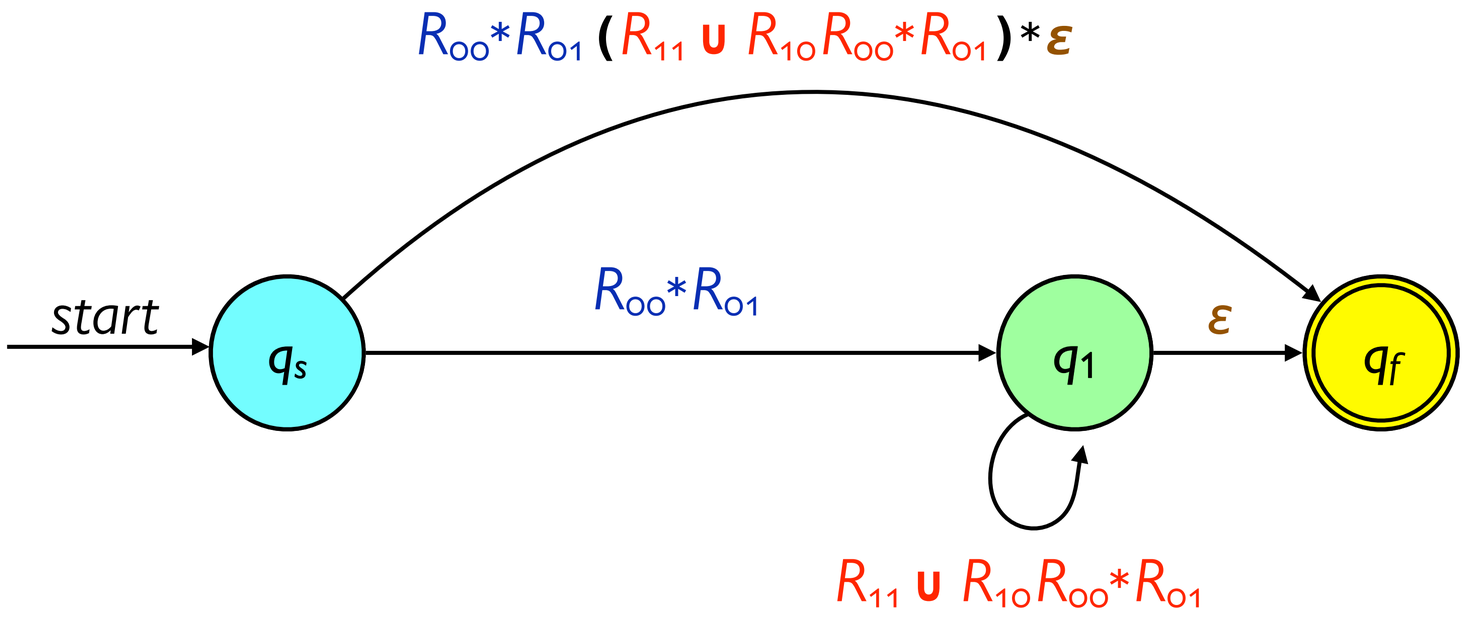


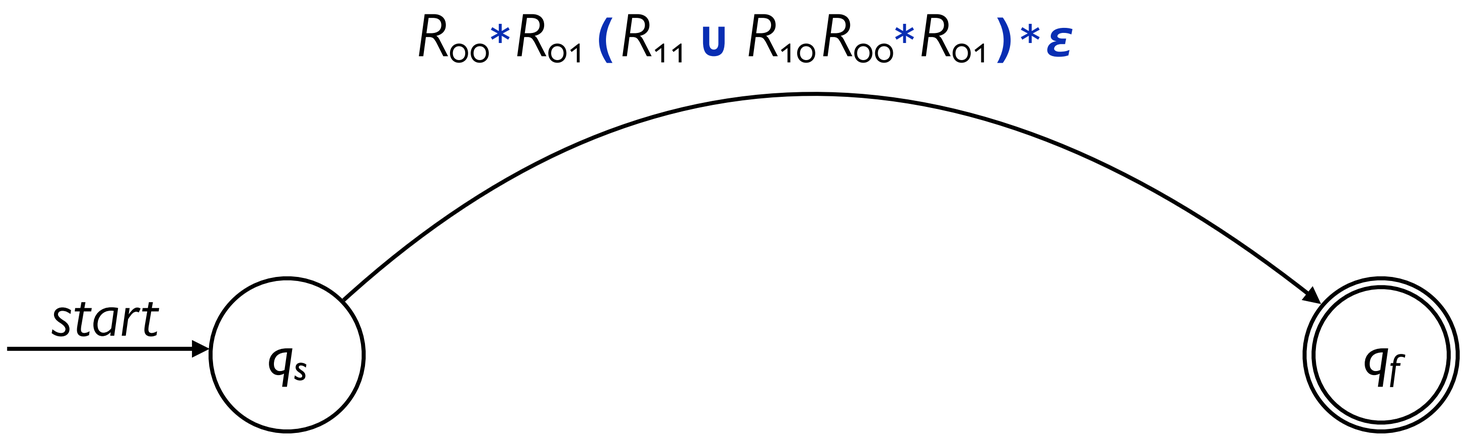

start
q_s
$R_{00}{}^*R_{01}$
q_1
ε
q_f
R_{11} u $R_{10}R_{00}{}^*R_{01}$

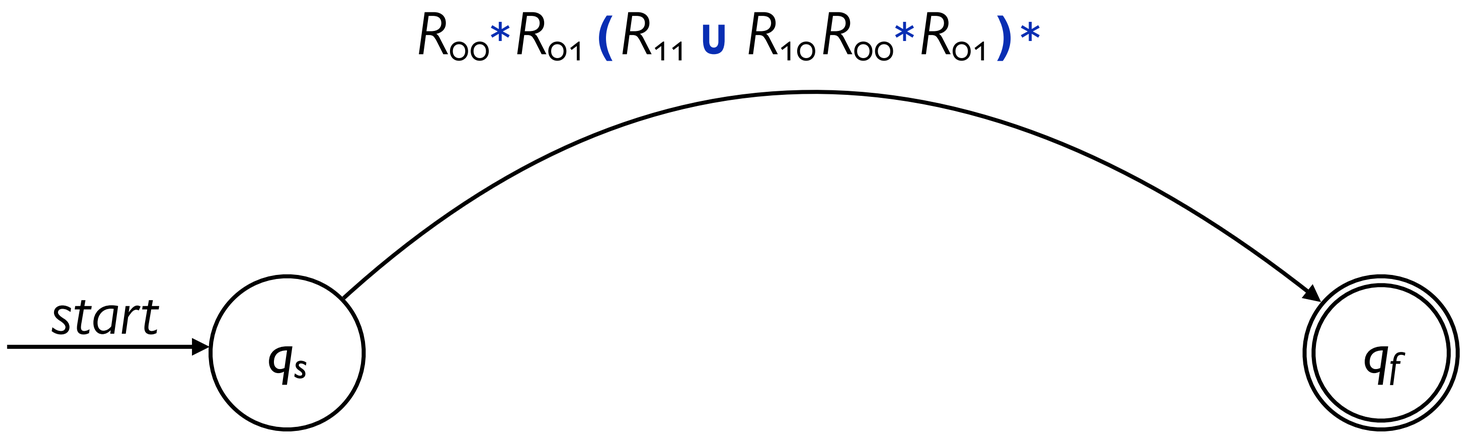
What should we put on this transition?

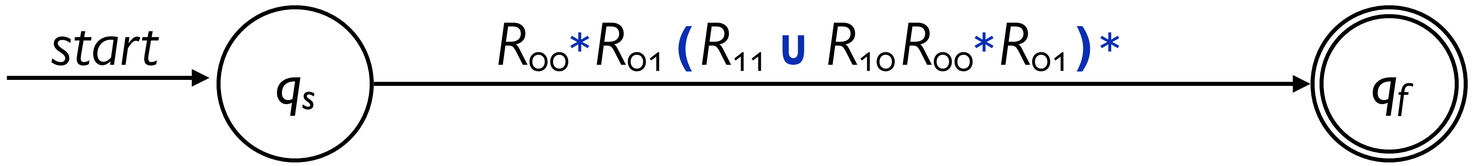



$R_{00}{}^*R_{01}\,(R_{11} \cup R_{10}R_{00}{}^*R_{01})^*\varepsilon$
start
q_s
$R_{00}{}^*R_{01}$
q_1
ε
q_f
$R_{11} \cup R_{10}R_{00}{}^*R_{01}$

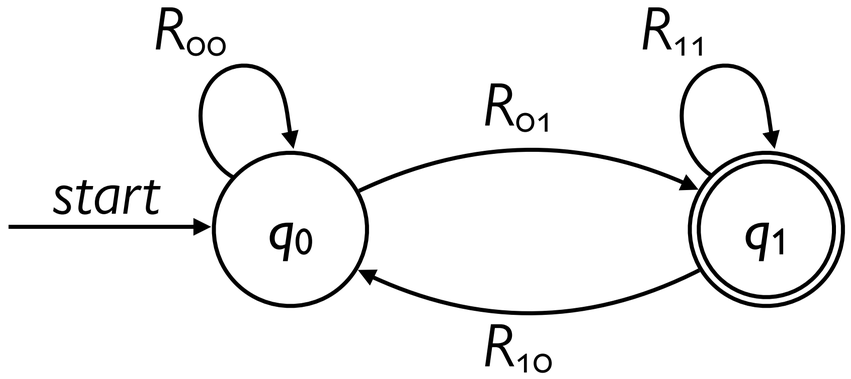

$R_{00}{}^*R_{01}\,(R_{11} \cup R_{10}R_{00}{}^*R_{01})^*\varepsilon$
start
q_s
$R_{00}{}^*R_{01}$
q_1
ε
q_f
$R_{11} \cup R_{10}R_{00}{}^*R_{01}$


$R_{00}{}^{*}R_{01}\,(R_{11} \cup R_{10}R_{00}{}^{*}R_{01})^{*}\varepsilon$
start
q_s
q_f

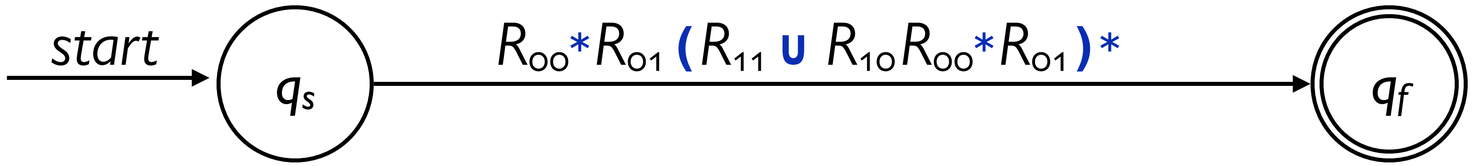

$R_{00}{}^{*}R_{01}\,(R_{11} \cup R_{10}R_{00}{}^{*}R_{01})^{*}$
start
q_s
q_f


start
q_s
$R_{00}{}^*R_{01}\,(R_{11} \cup R_{10}R_{00}{}^*R_{01})^*$
q_f

Before:

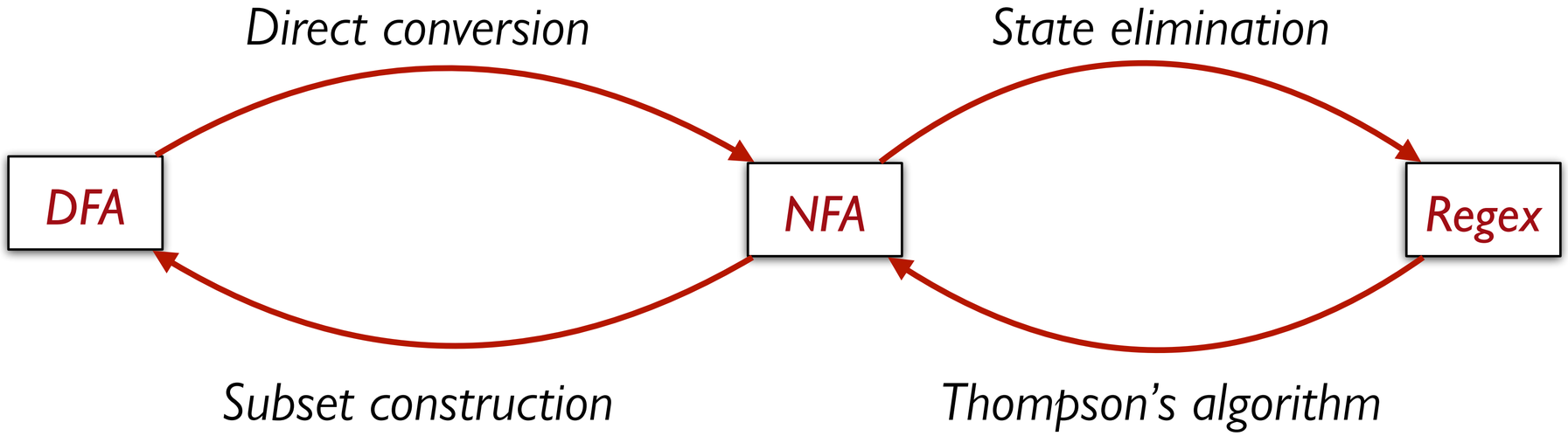


After:



What we just saw is called the *state-elimination algorithm*.

Our transformations



The following are all equivalent:

L is a regular language.

There is a DFA *D* such that $L(D) = L$.

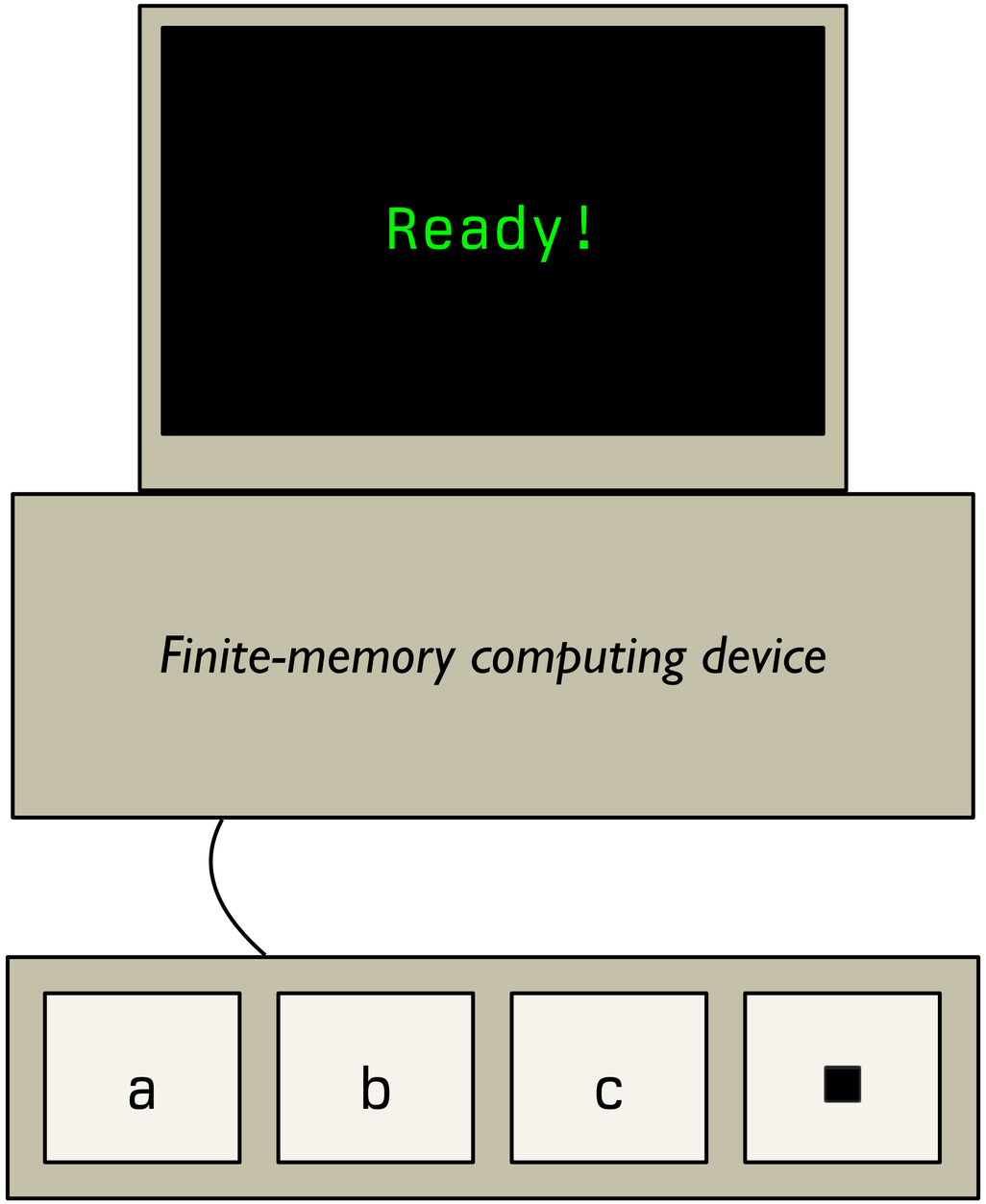
There is an NFA *N* such that $L(N) = L$.

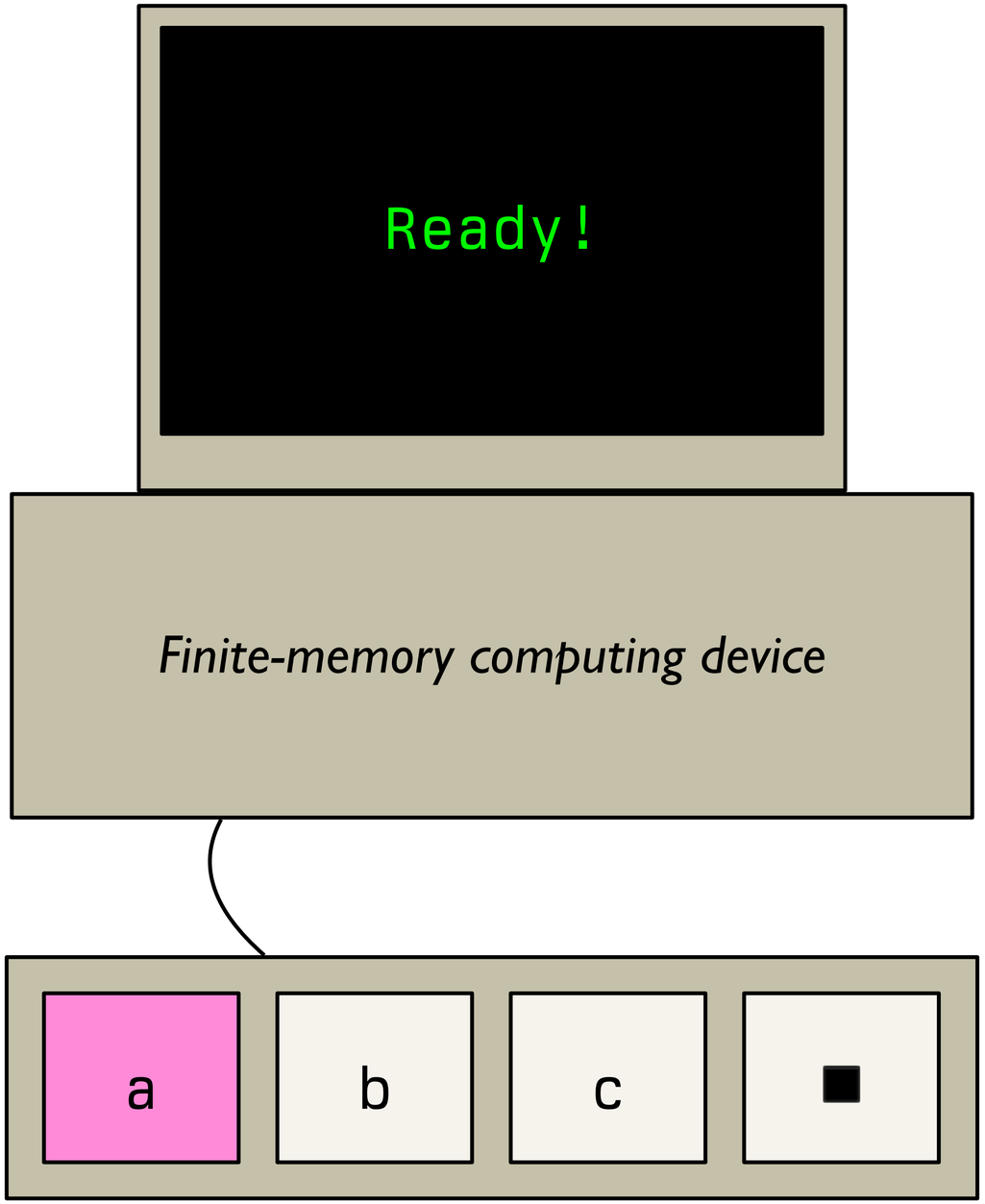
There is a regular expression *R* such that $L(R) = L$.

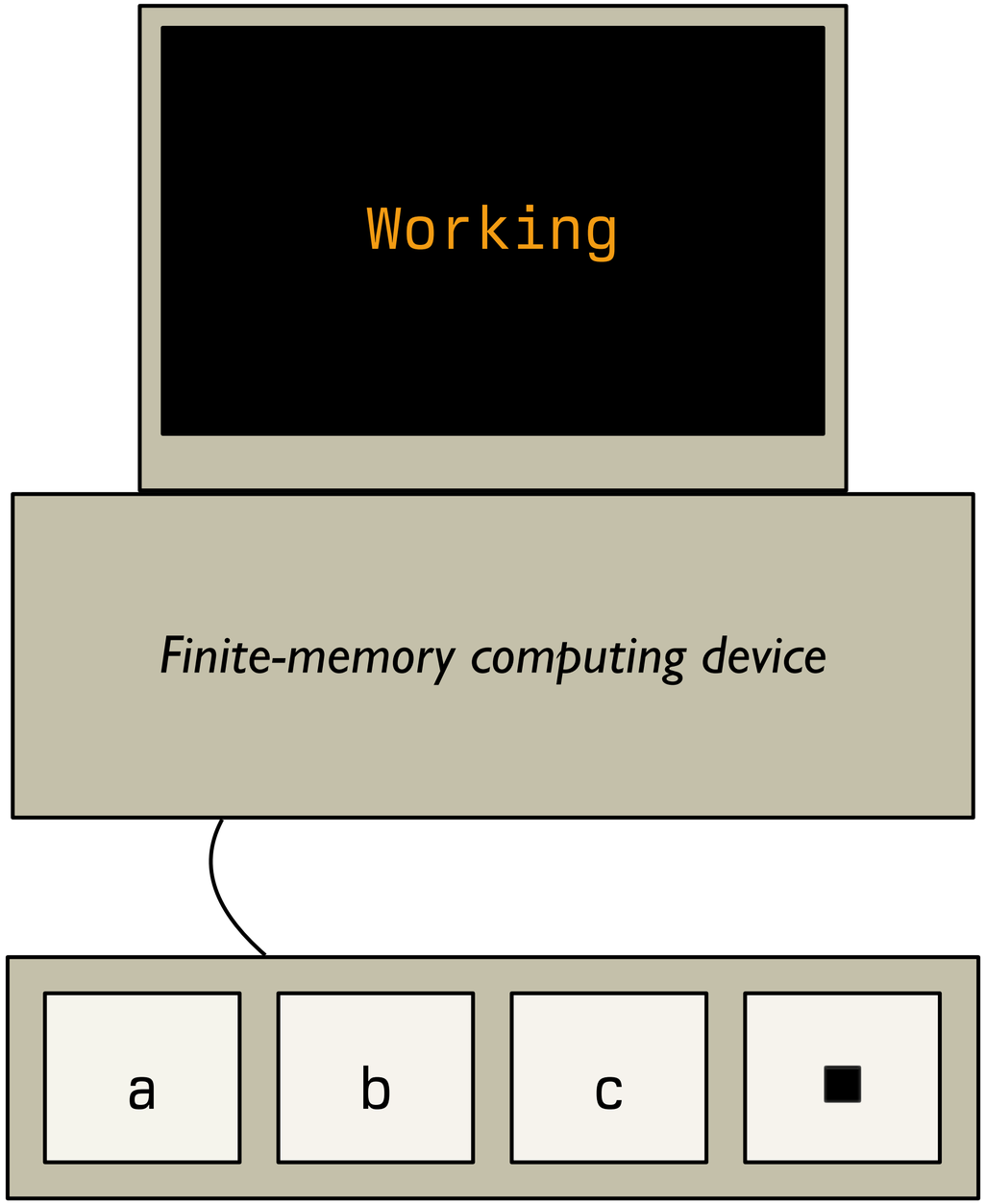
Is every language regular?

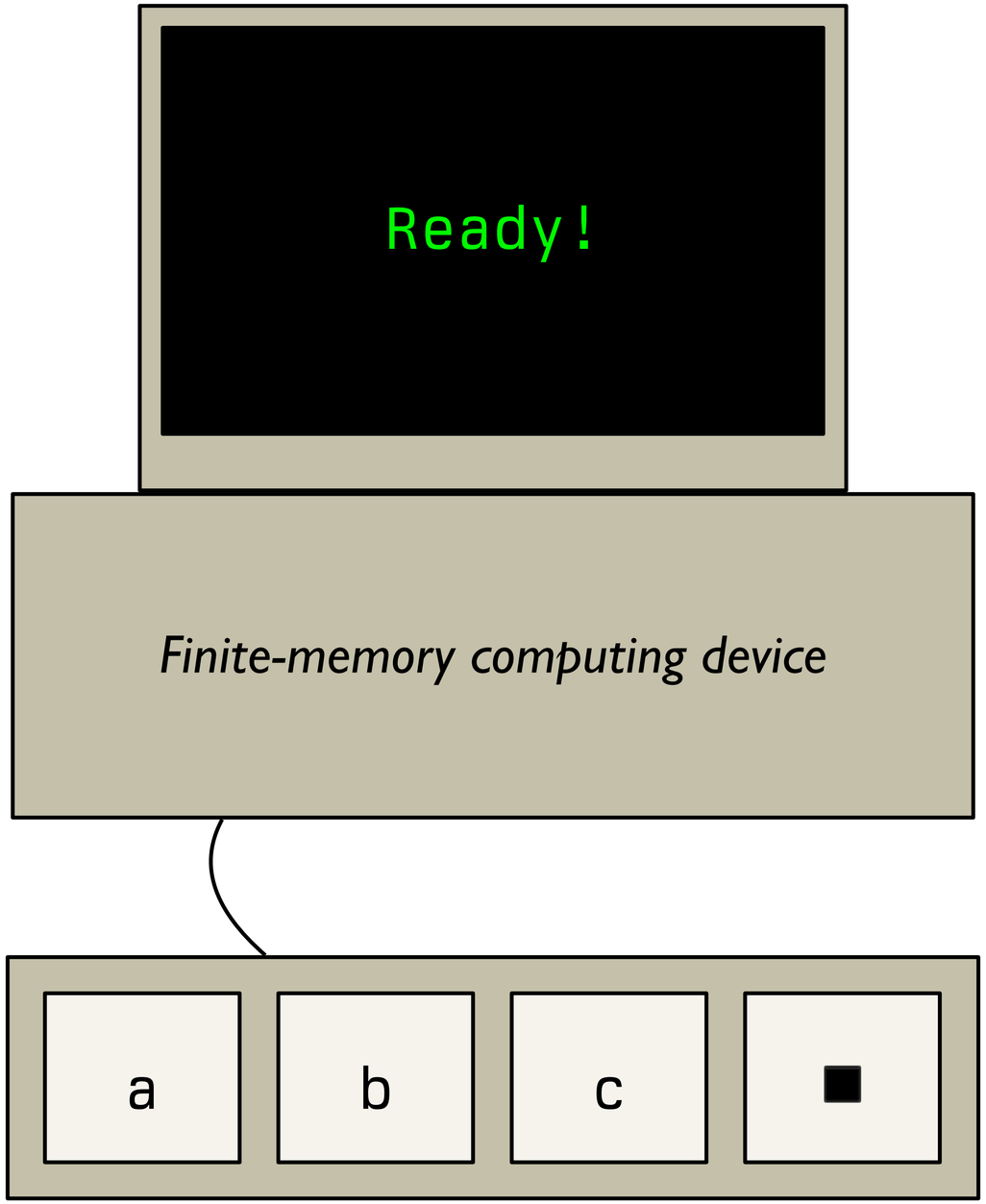
What, if anything, can't we solve with DFAs, NFAs, and regular expressions?

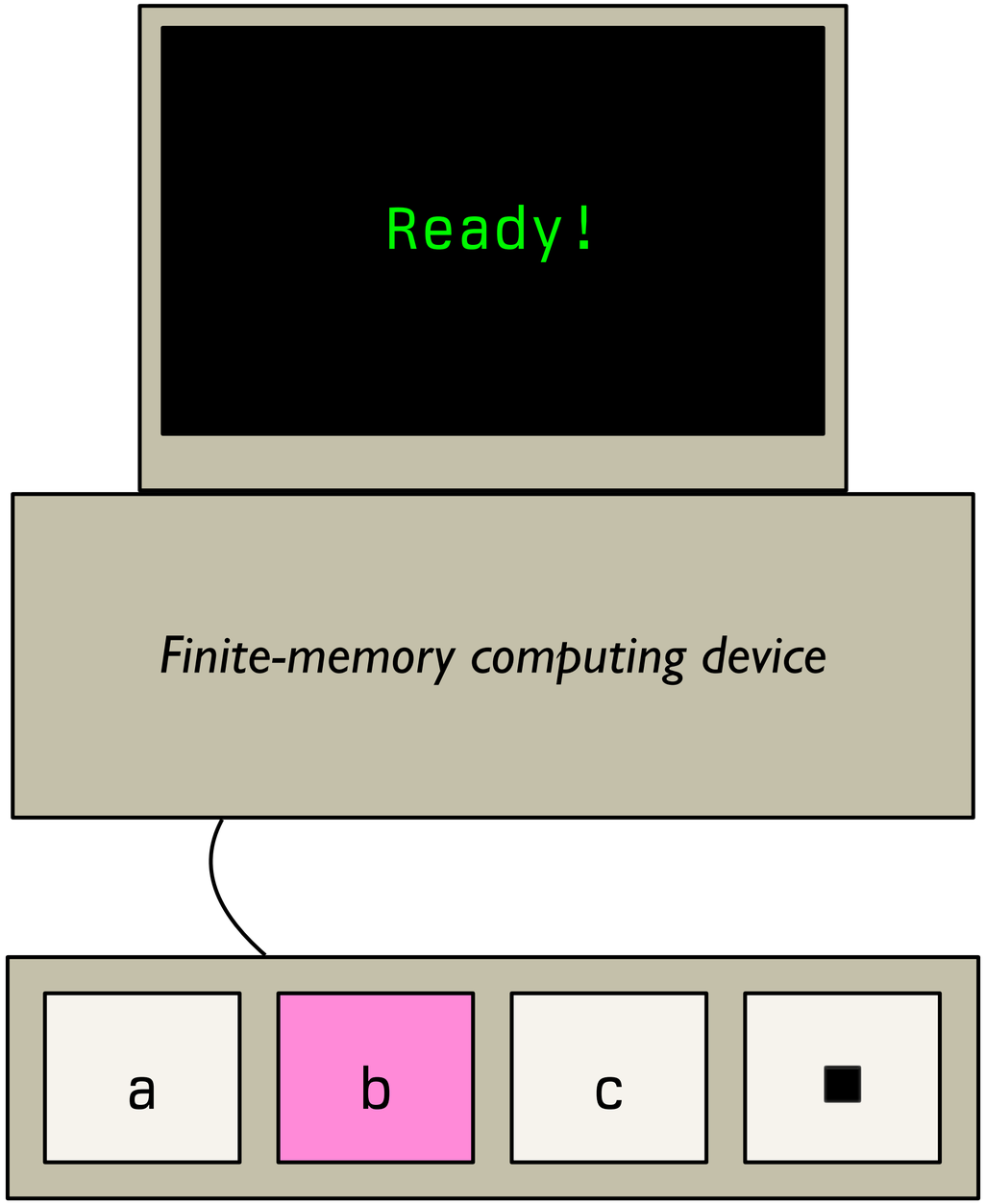
Automata and computation

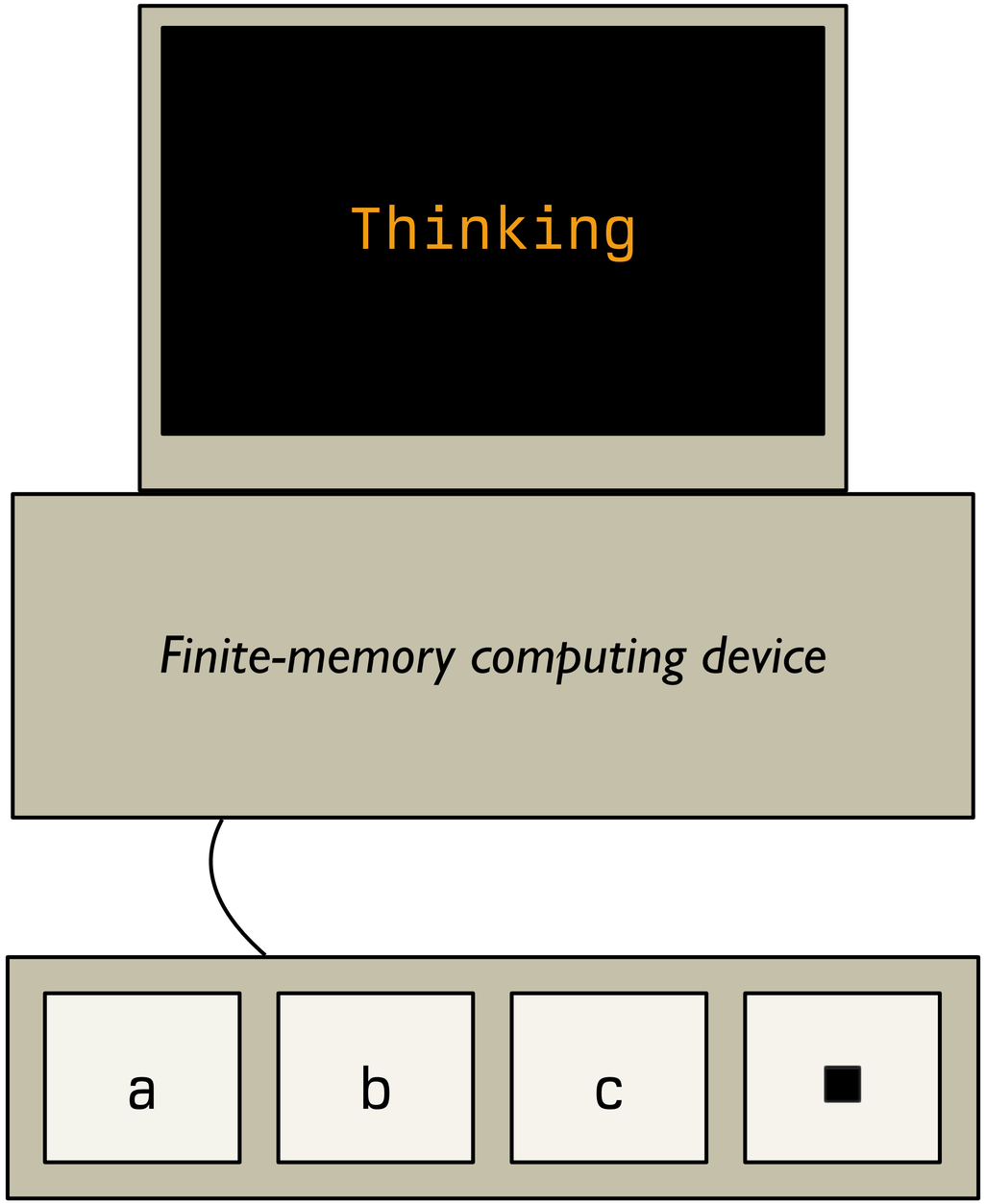

Ready!
Finite-memory computing device
a
b
c

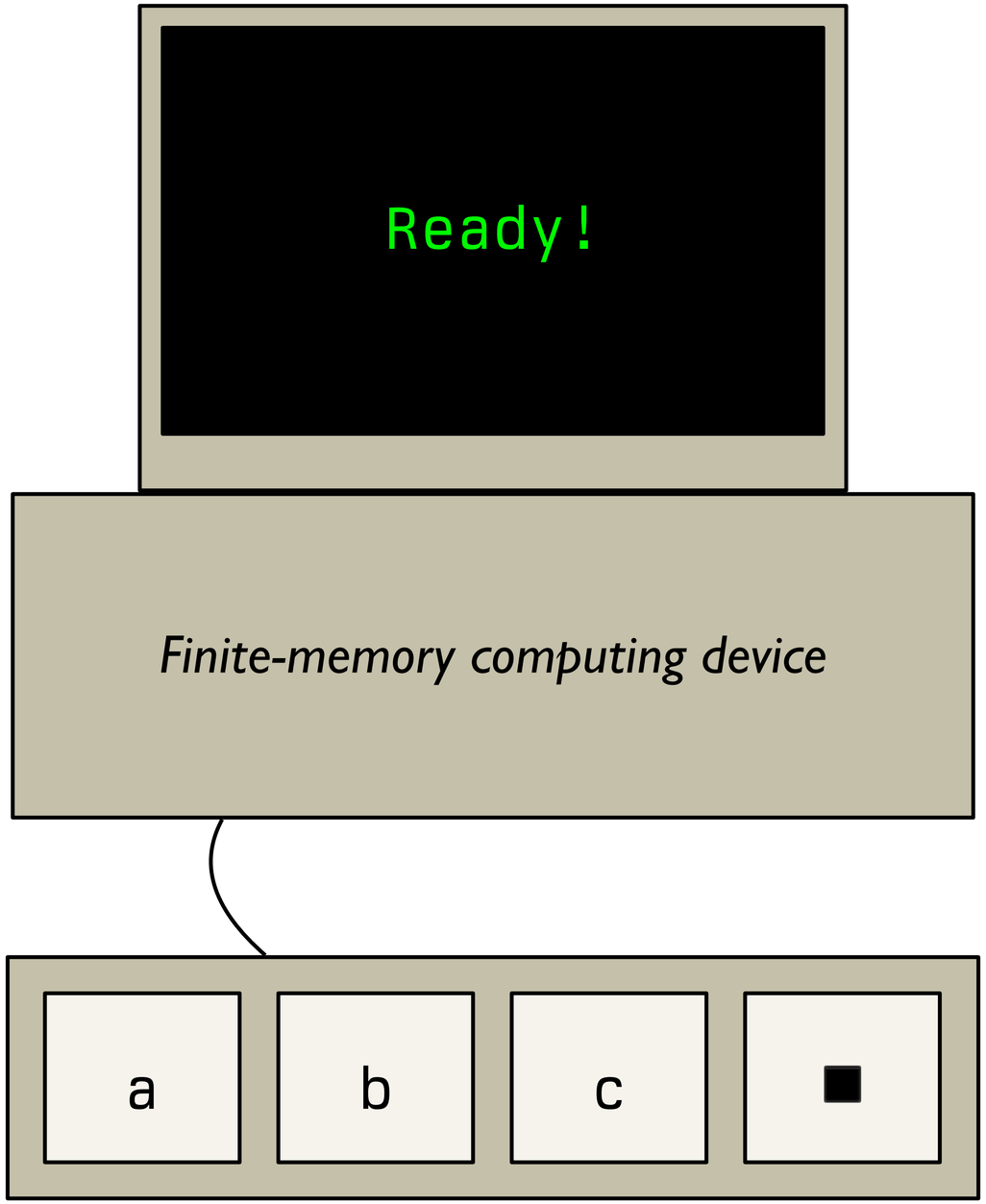

Ready!
Finite-memory computing device
a
b
c

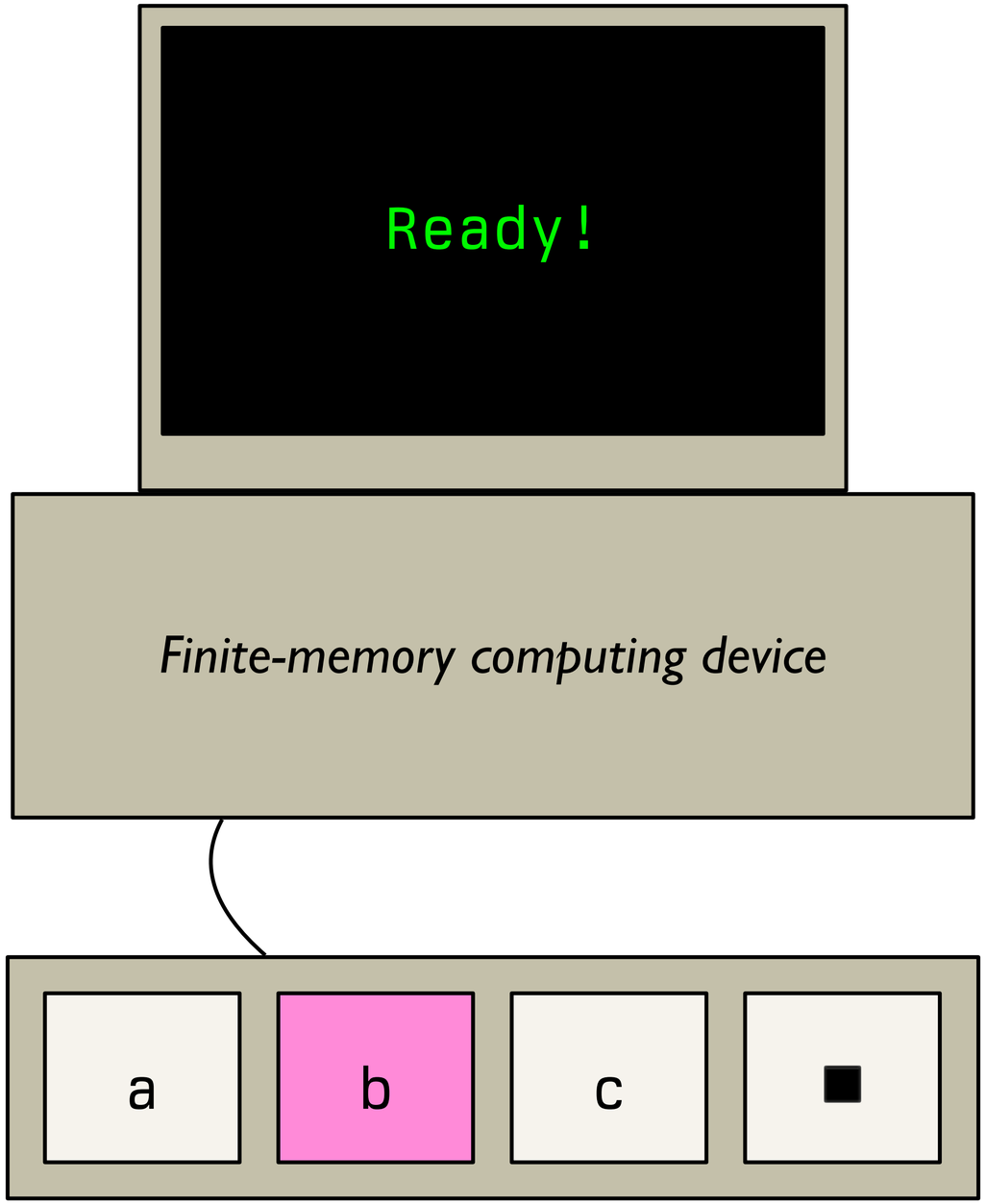

Working
Finite-memory computing device
a
b
c

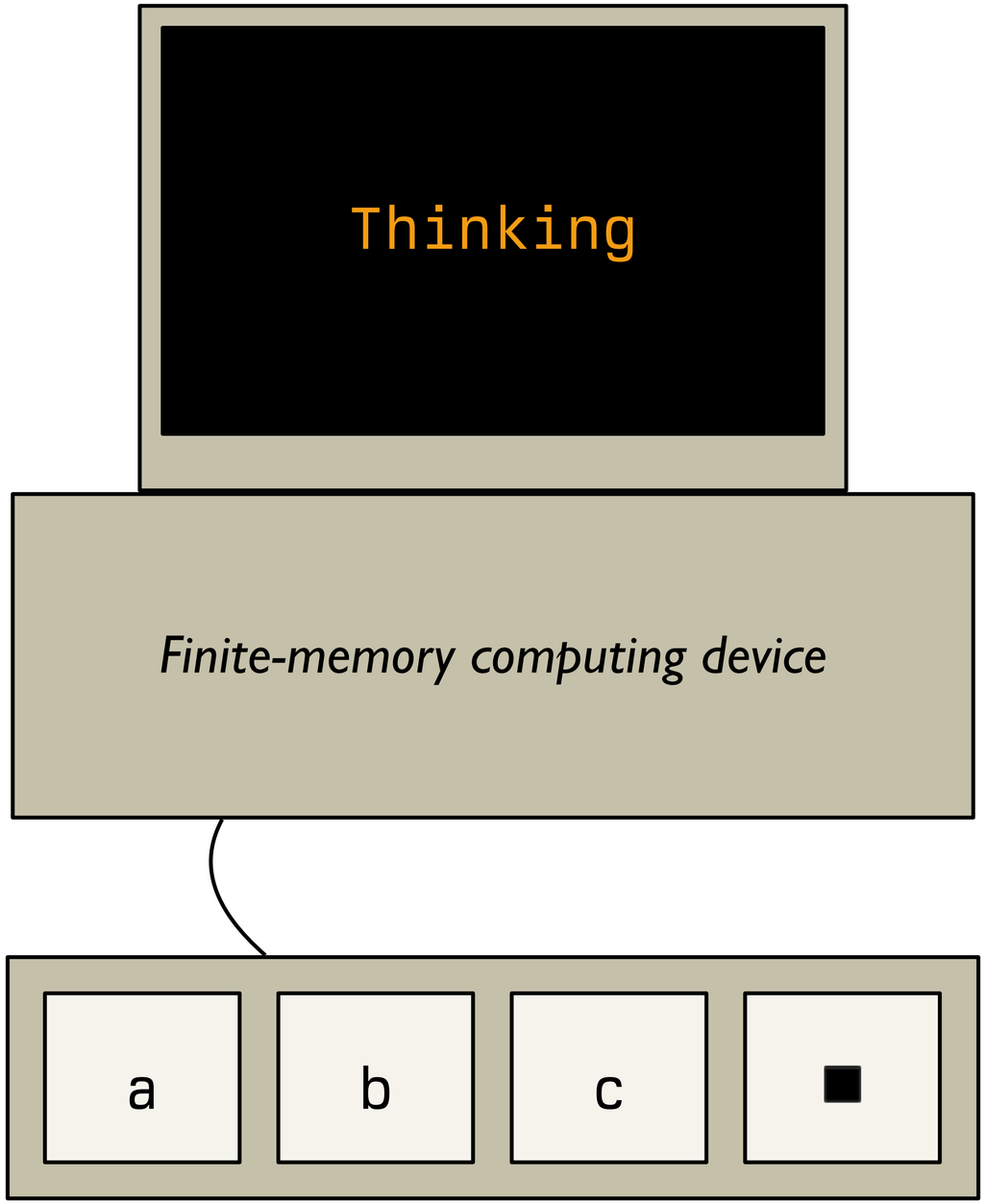

Ready!
Finite-memory computing device
a
b
c

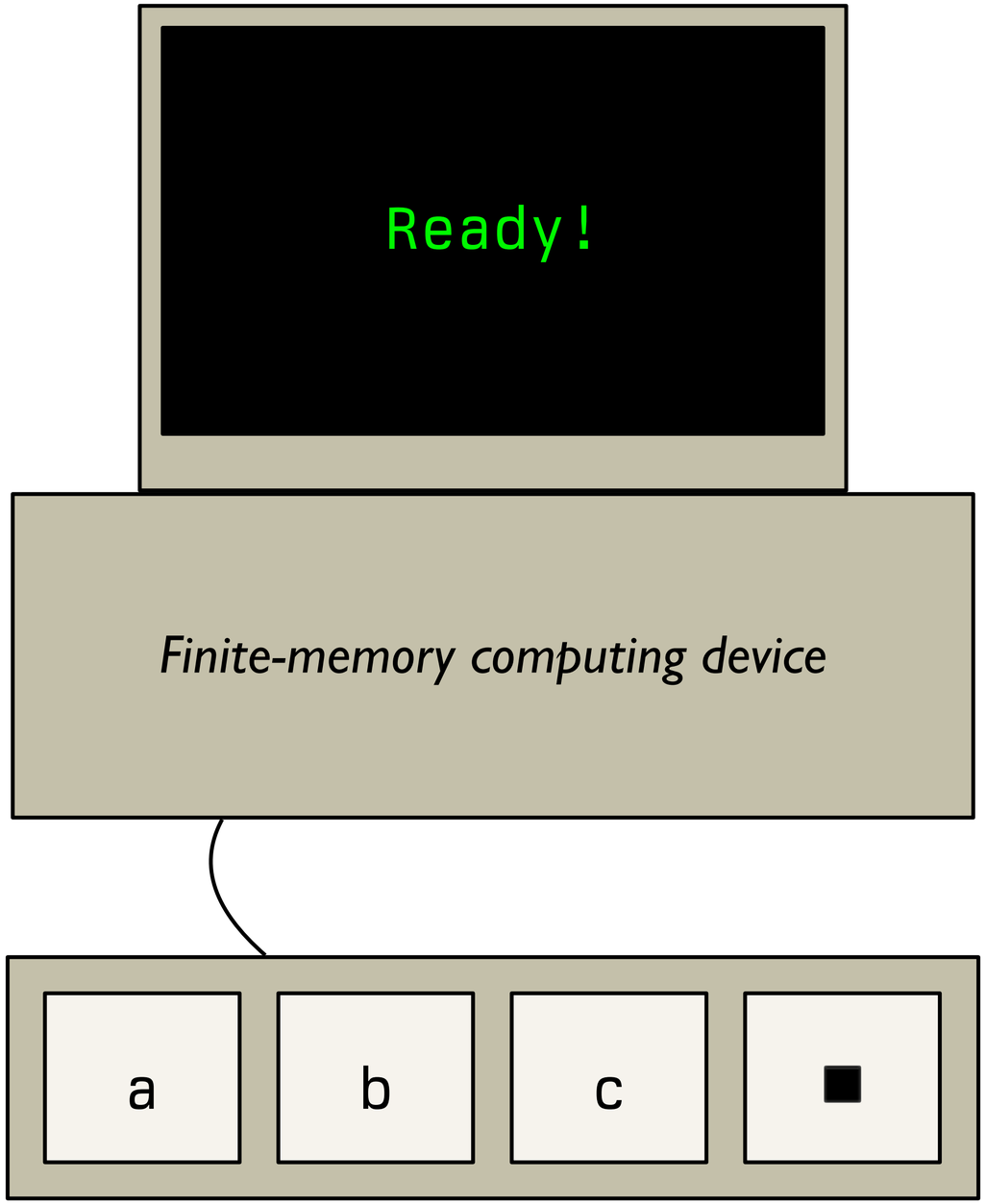

Ready!
Finite-memory computing device
a
b
c

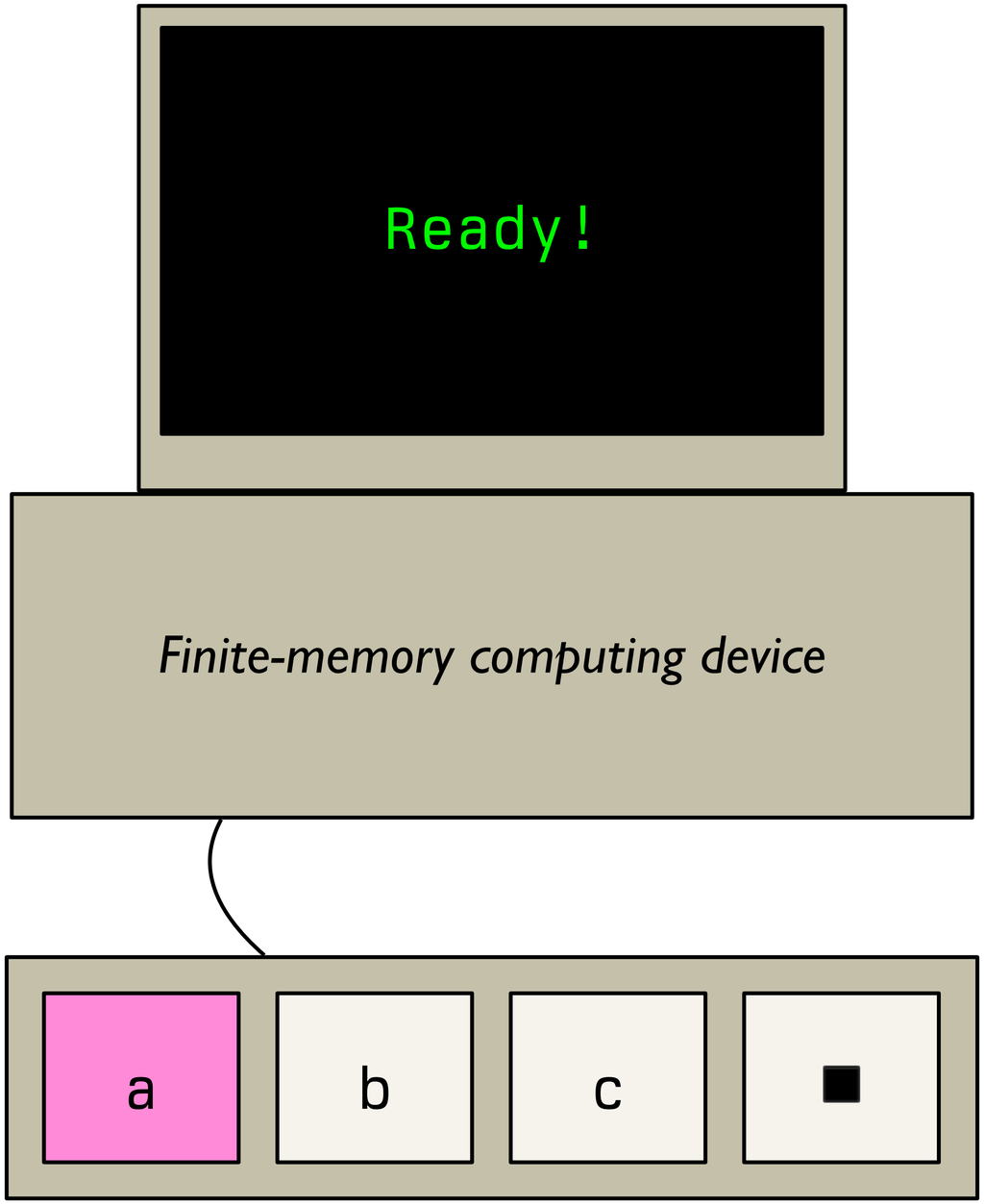

Thinking
Finite-memory computing device
a
b
c

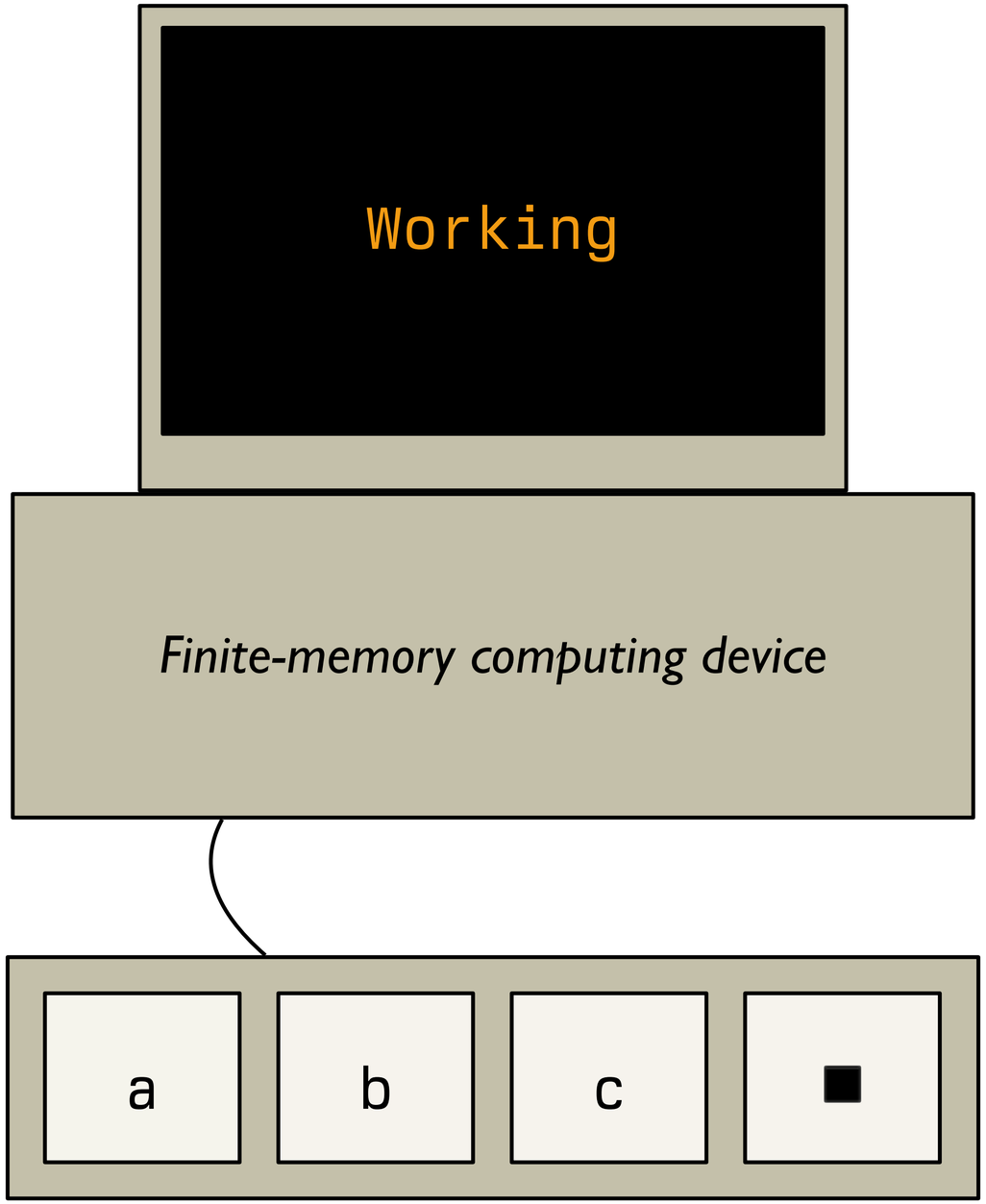

Ready!
Finite-memory computing device
a
b
c

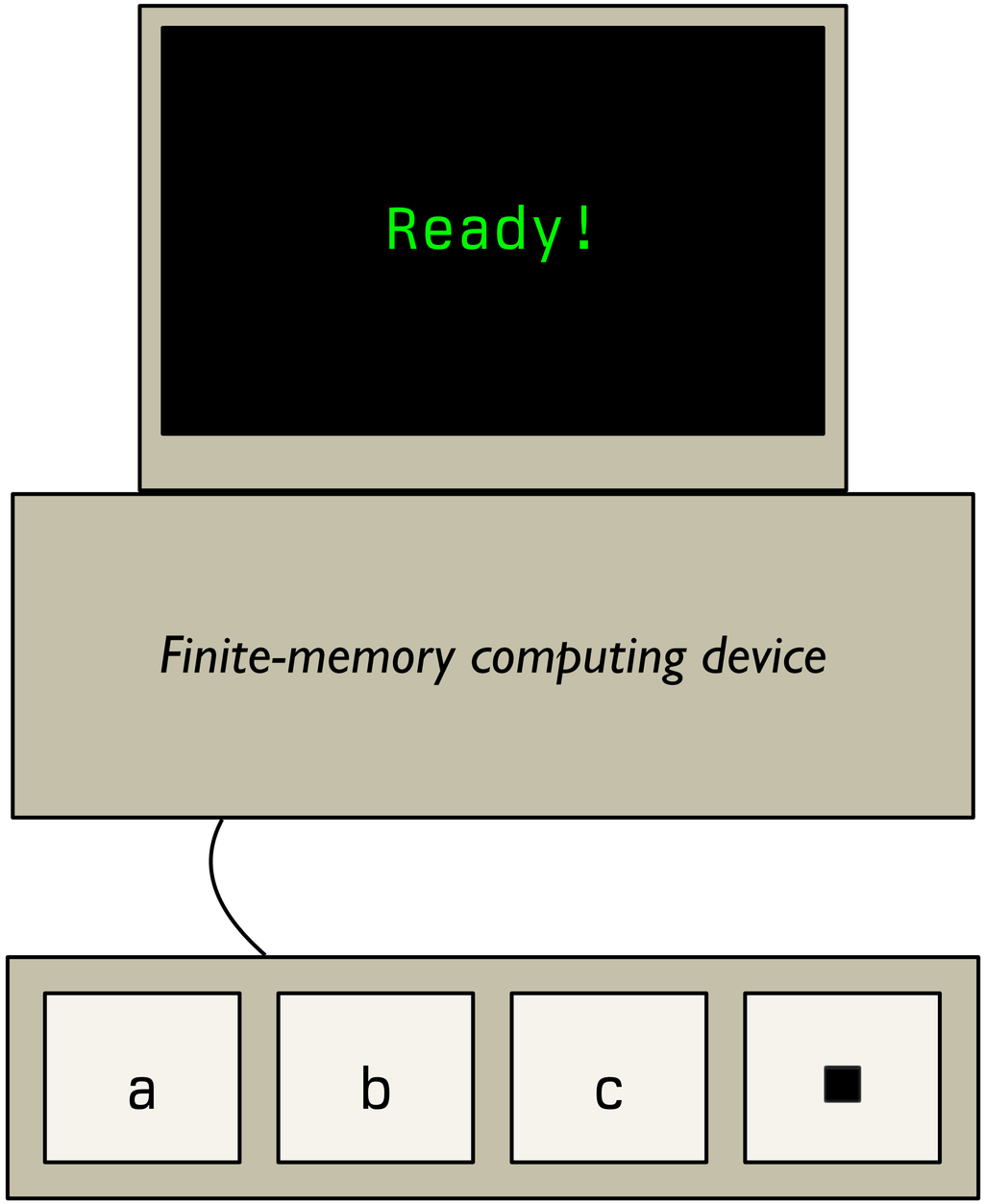

Ready!
Finite-memory computing device
a
b
c

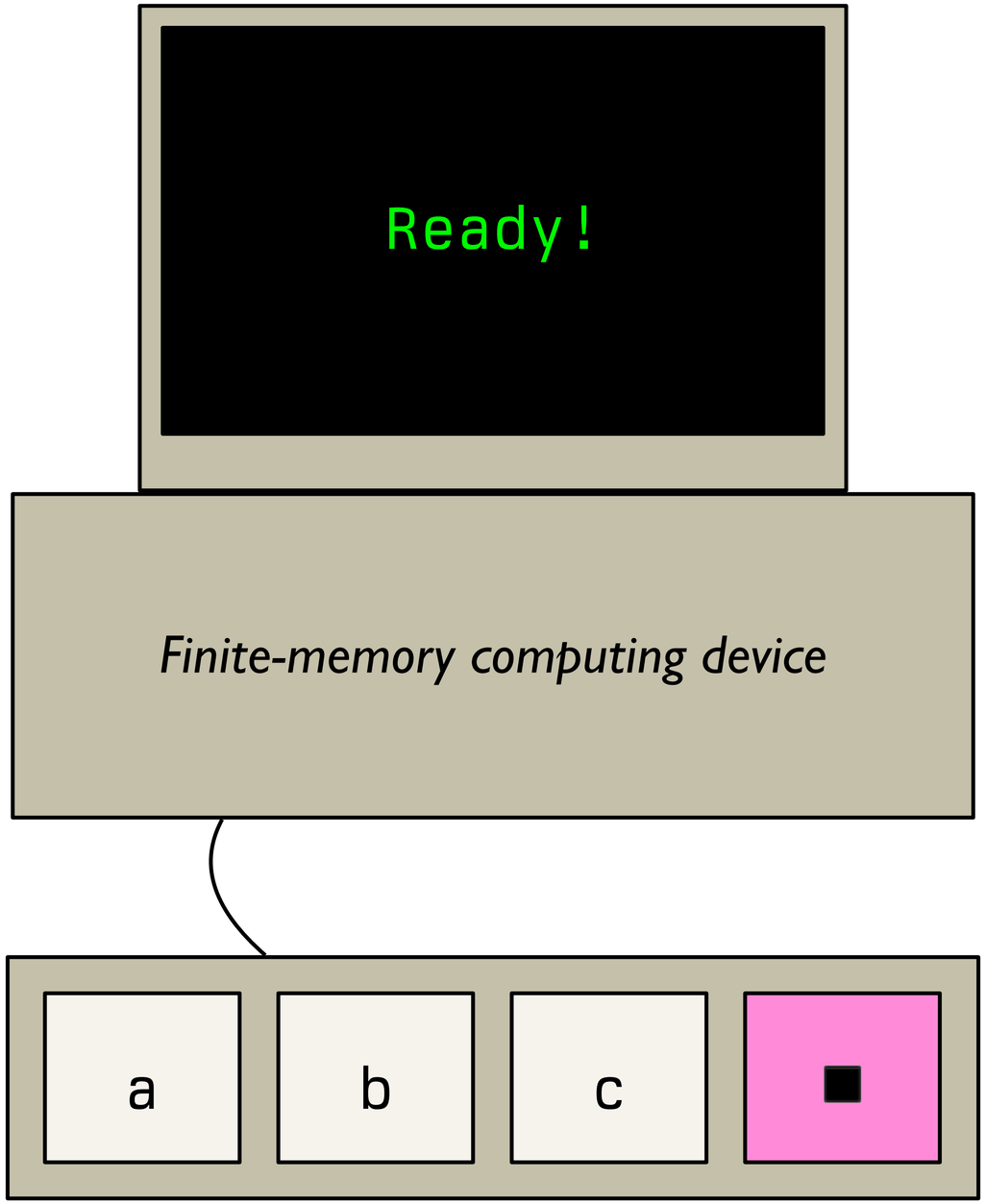

Thinking
Finite-memory computing device
a
b
c

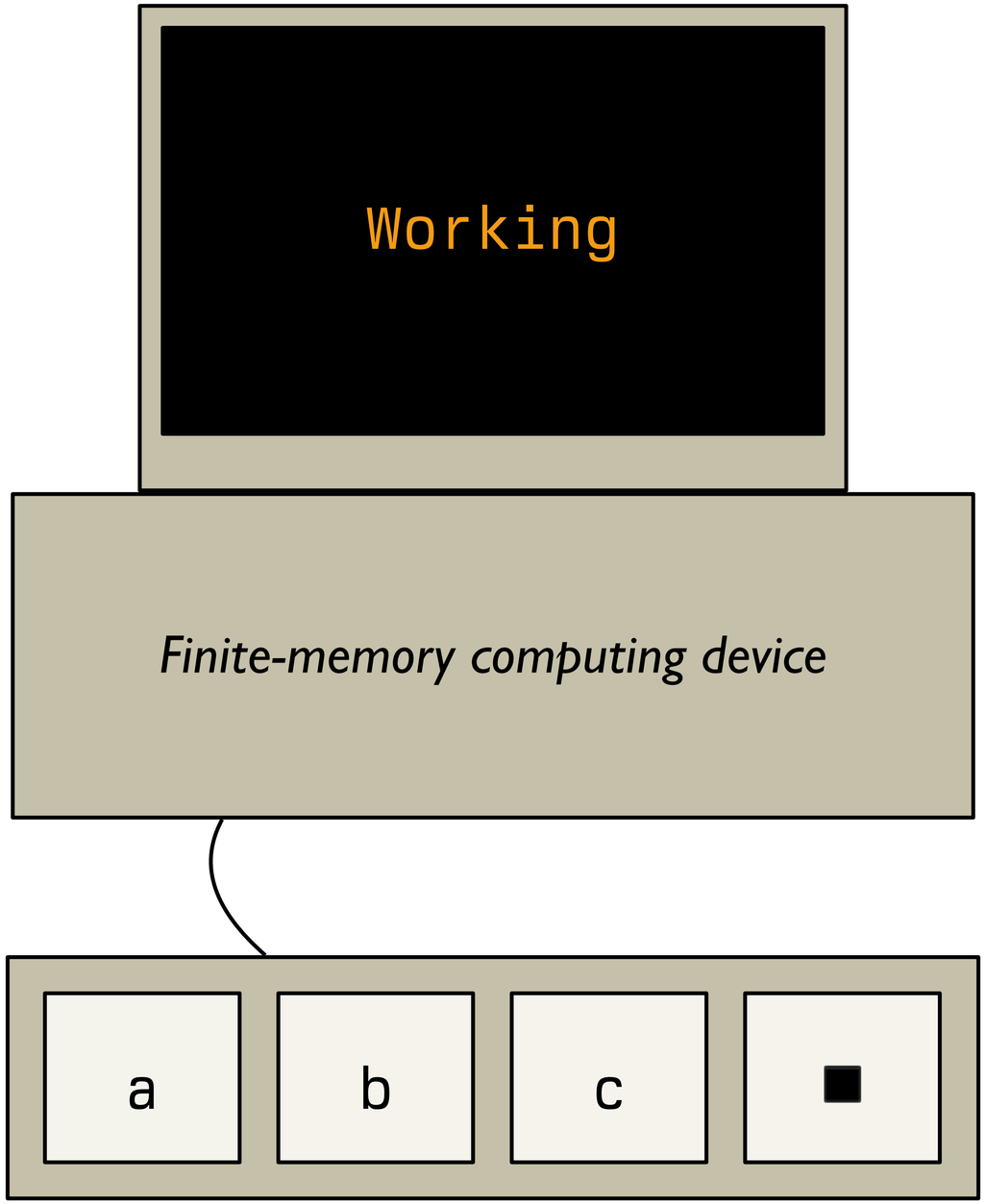

Ready!
Finite-memory computing device
a
b
c

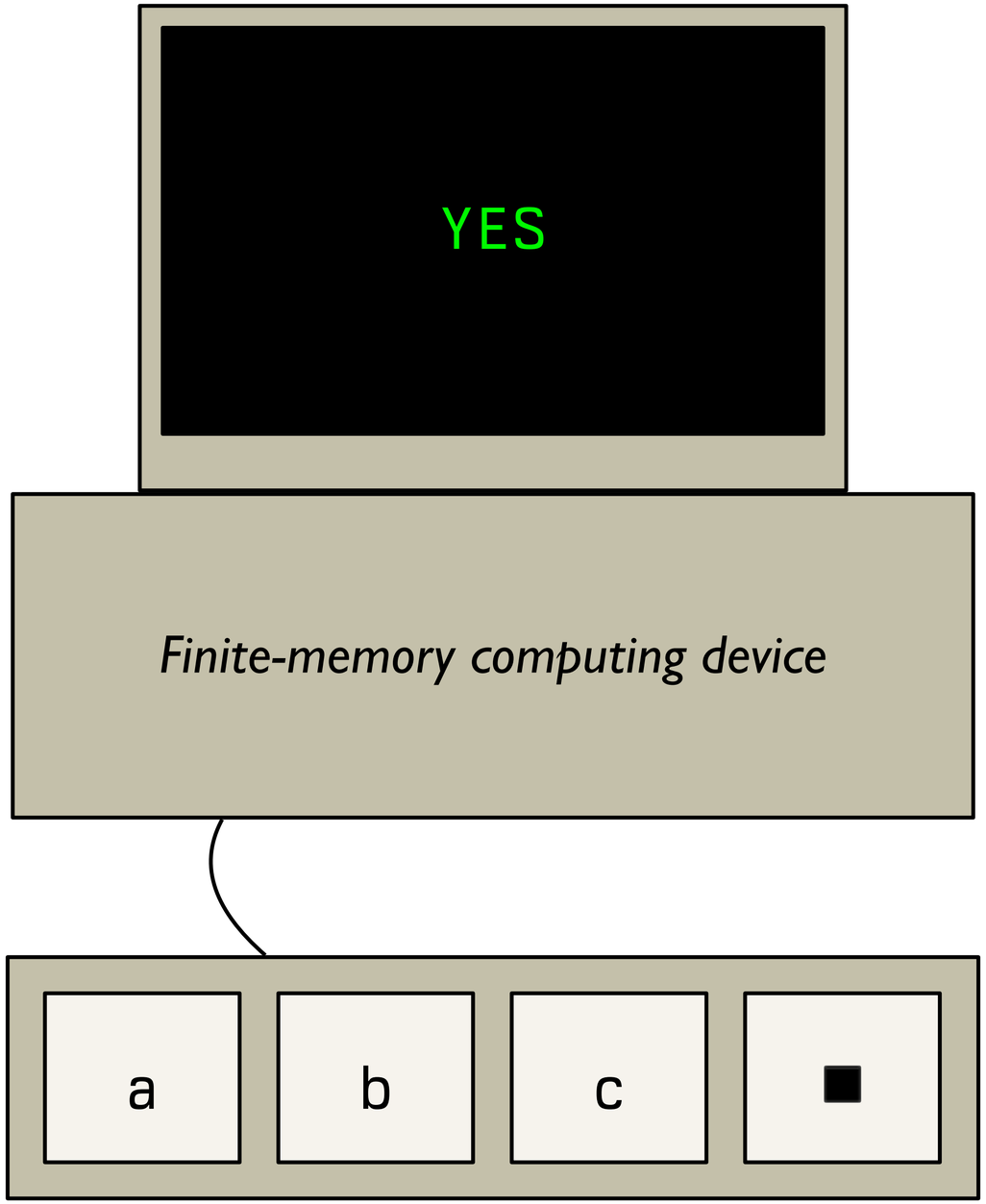

Ready!
Finite-memory computing device
a
b
c


Working
Finite-memory computing device
a
b
c


Ready!
Finite-memory computing device
a
b
c


Ready!
Finite-memory computing device
a
b
c


Working
Finite-memory computing device
a
b
c


YES
Finite-memory computing device
a
b
c

The computing device has internal workings that can be in one of finitely many possible configurations.

Each ***state*** in a DFA corresponds to some configuration of the internal workings.

After each button press, the computing device does some amount of processing then gets to a configuration where it's ready to receive more input.

Each ***transition*** abstracts away how the computation is done and just indicates what the ultimate configuration looks like.

After the user presses the "done" button, the computer outputs either YES or NO.

The ***accept*** and ***non-accept*** (rejecting) states of the machine model what happens when that button is pressed.

Regular and nonregular languages

Regular languages correspond to problems that can be solved with finite memory.

At each point in time, we only need to store one of finitely many pieces of information.

Nonregular languages correspond to problems that cannot be solved with finite memory.

Since every computer ever built has finite memory, nonregular languages correspond to problems that cannot be solved by physical computers!

To prove a language is regular, we can construct a DFA, NFA, or regular expression to recognize it.

To prove a language is not regular, we need to show that it's impossible to construct a DFA, NFA, or regular expression for it.

Claim: We can actually just prove that there's no DFA for it. Why is this?

Arguing a language is not regular

A simple language

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

E is the language of all strings of *n* **a**s followed by *n* **b**s:

$$\{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

Is this language regular?

A simple language

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

$$a^n b^n$$

Easy!

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

$$a^n b^n$$

Regular expressions can't have variables!

Easy!

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

a*b*

Maybe?

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

a*b*

Doesn't require the same number of a*s and* b*s.*

Maybe?

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

```
(ab)*
```

Surely!

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

`(ab)*`

This repeats the string ab *rather than keeping all the* as *together and all the* bs *together*

Surely!

A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

$$\varepsilon \cup ab \cup a^2b^2 \cup a^3b^3 \cup \cdots$$

Please? 🥺

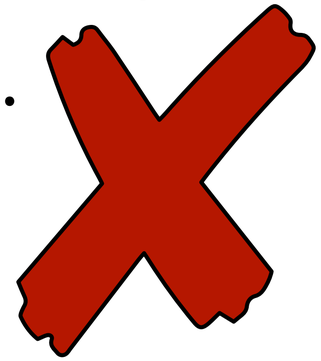
A simple language: Regular expression

Let $\Sigma = \{a, b\}$ and consider this language:

$$E = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

Could we write a regular expression for it?

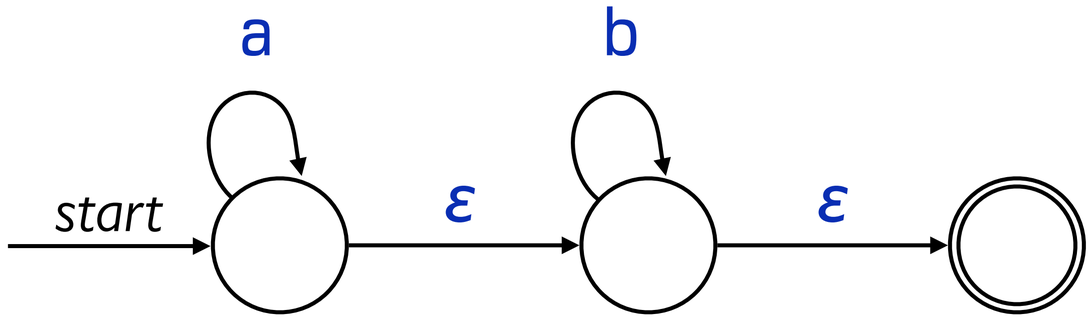
$$\varepsilon \cup ab \cup a^2b^2 \cup a^3b^3 \cup \cdots$$



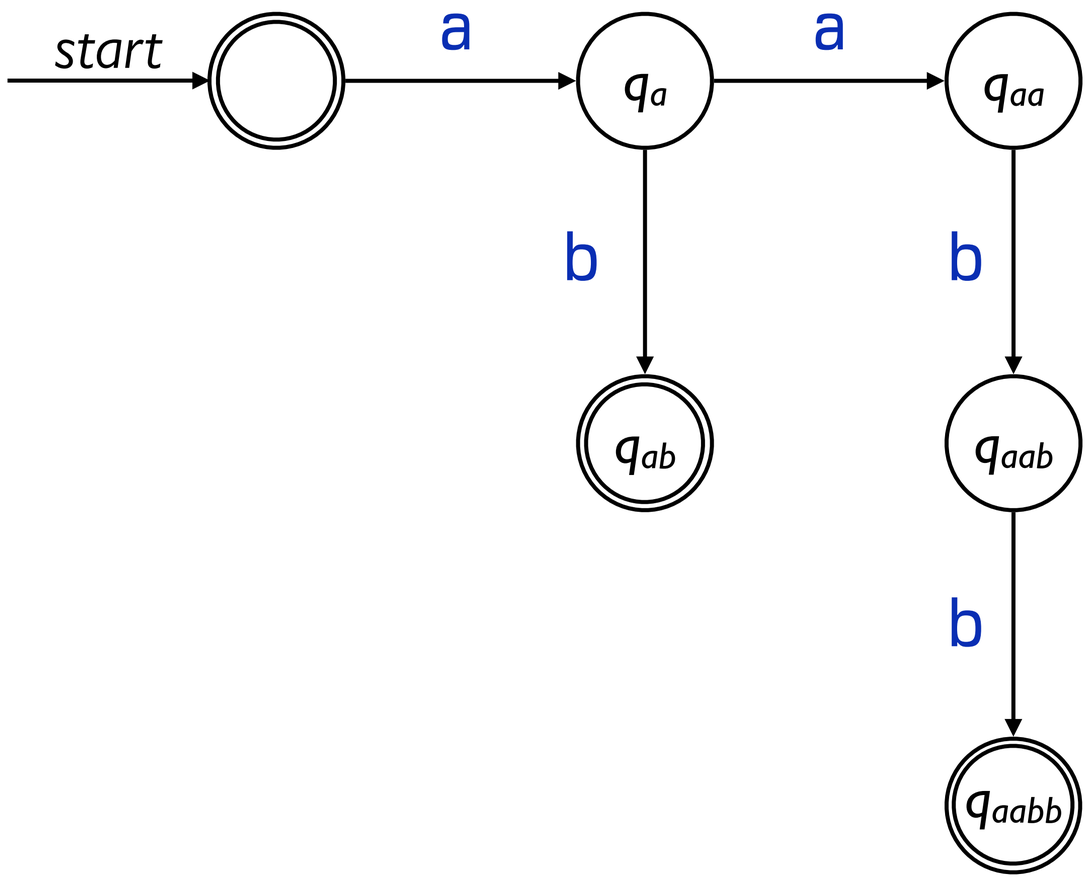
Please? 🥺

If you can't finish writing it, it's not a complete regular expression!

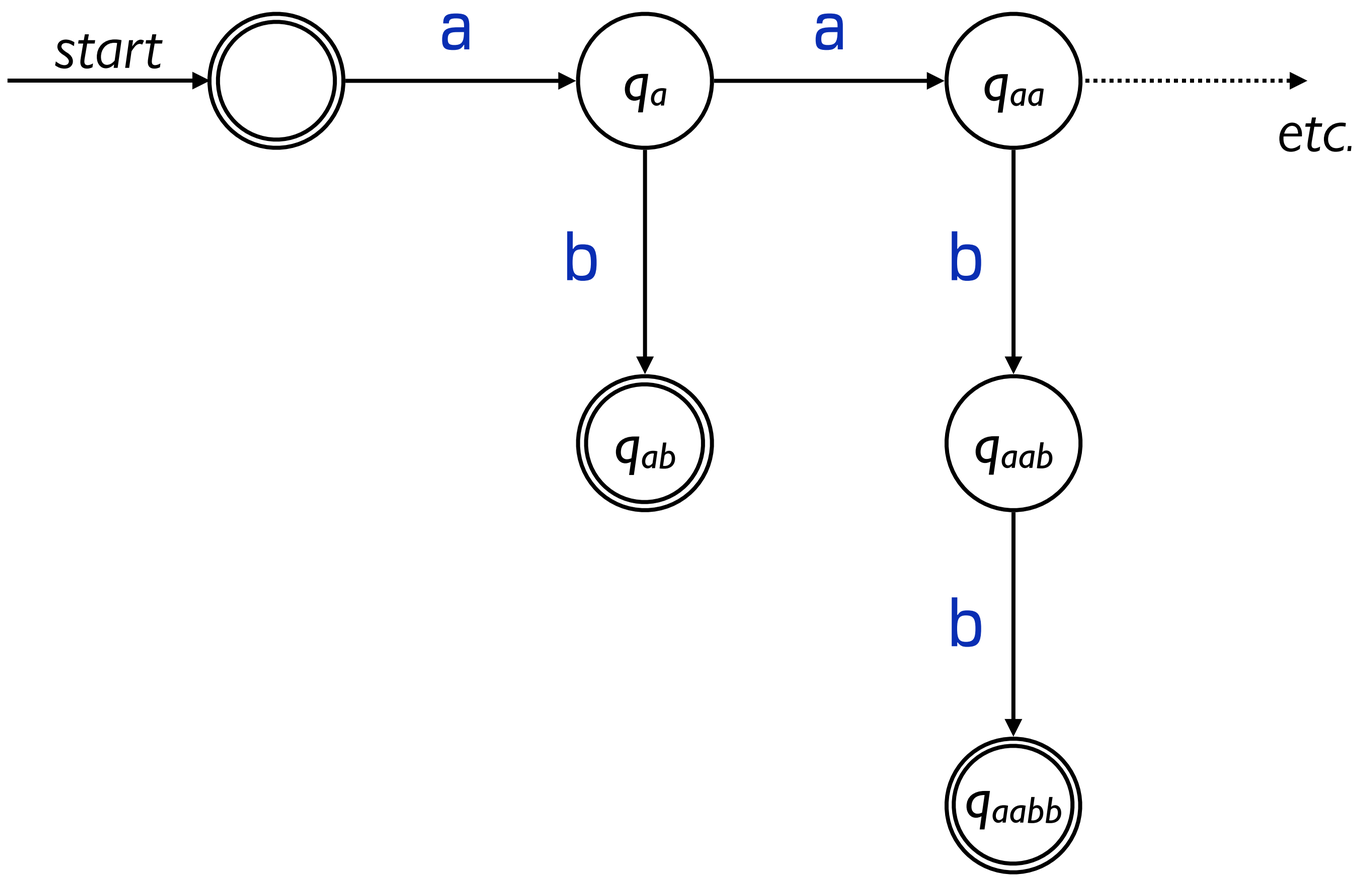
A simple language: Another attempt



A simple language: Another attempt



A simple language: Another attempt



This language is not regular!

Intuitive explanation:

Imagine a finite automaton to recognize this language.

When any DFA for L is run on any two of the strings ε, a, aa, aaa, aaaa, etc., the DFA must end in different states.

Suppose a^n and a^m end up in the same state, where $m \neq n$.

Then $a^n b^n$ and $a^m b^n$ will end up in the same state.

The DFA will either accept a string not in the language or reject a string in the language, which it shouldn't be able to do.

We can't place all these strings into different states; there are only finitely many states!

The intuition for this proof is helpful to think about.

However, actually writing one of these proofs becomes difficult for more complicated languages.

Instead, we'll take the idea of the number of states required for a DFA to recognize a language and develop a powerful proof framework.

To prove *B* is nonregular, we entertain the possibility that *B* ***is*** regular: Imagine there's a DFA *M* that recognizes *B*.

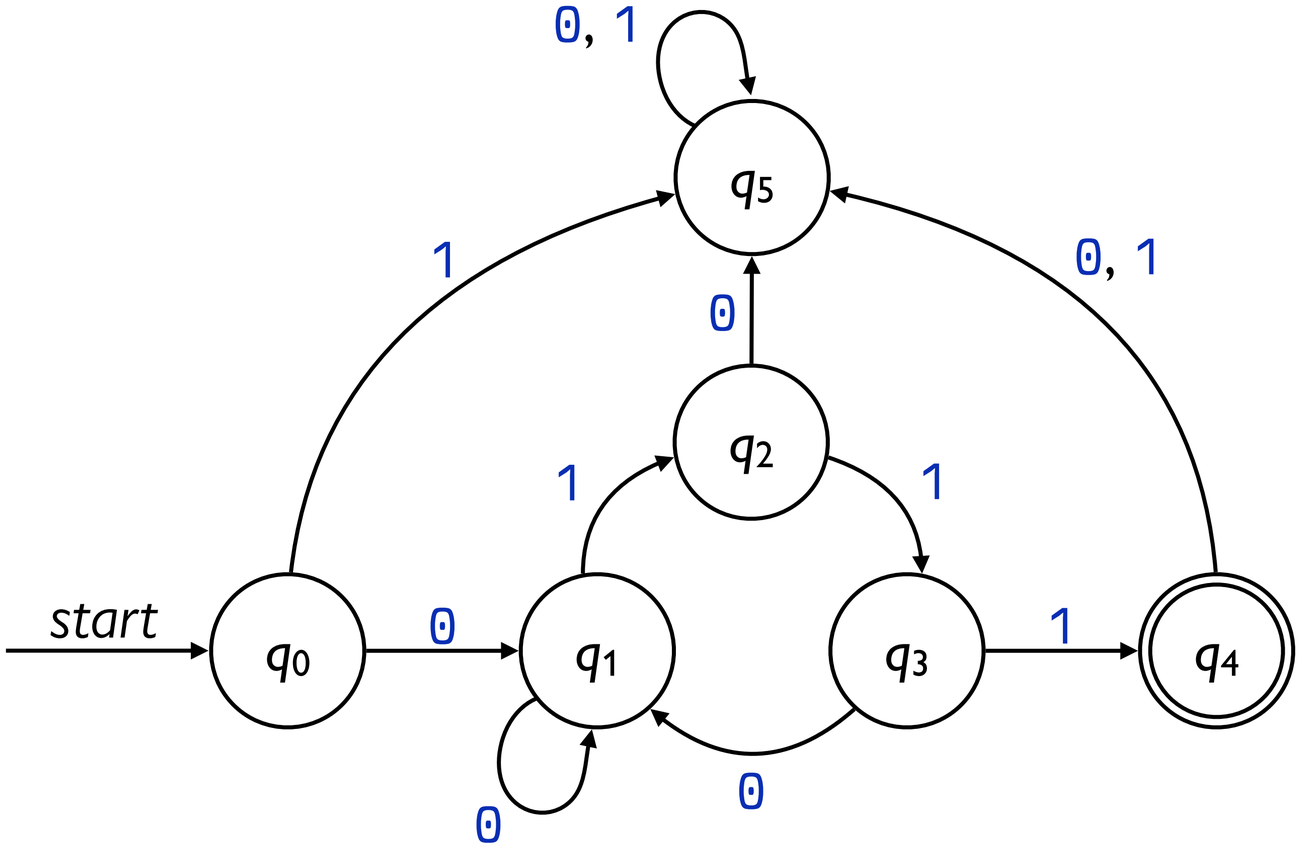
We can try to "break" that DFA by finding a string that's ***not*** in *B* but ***is*** accepted by *M*.

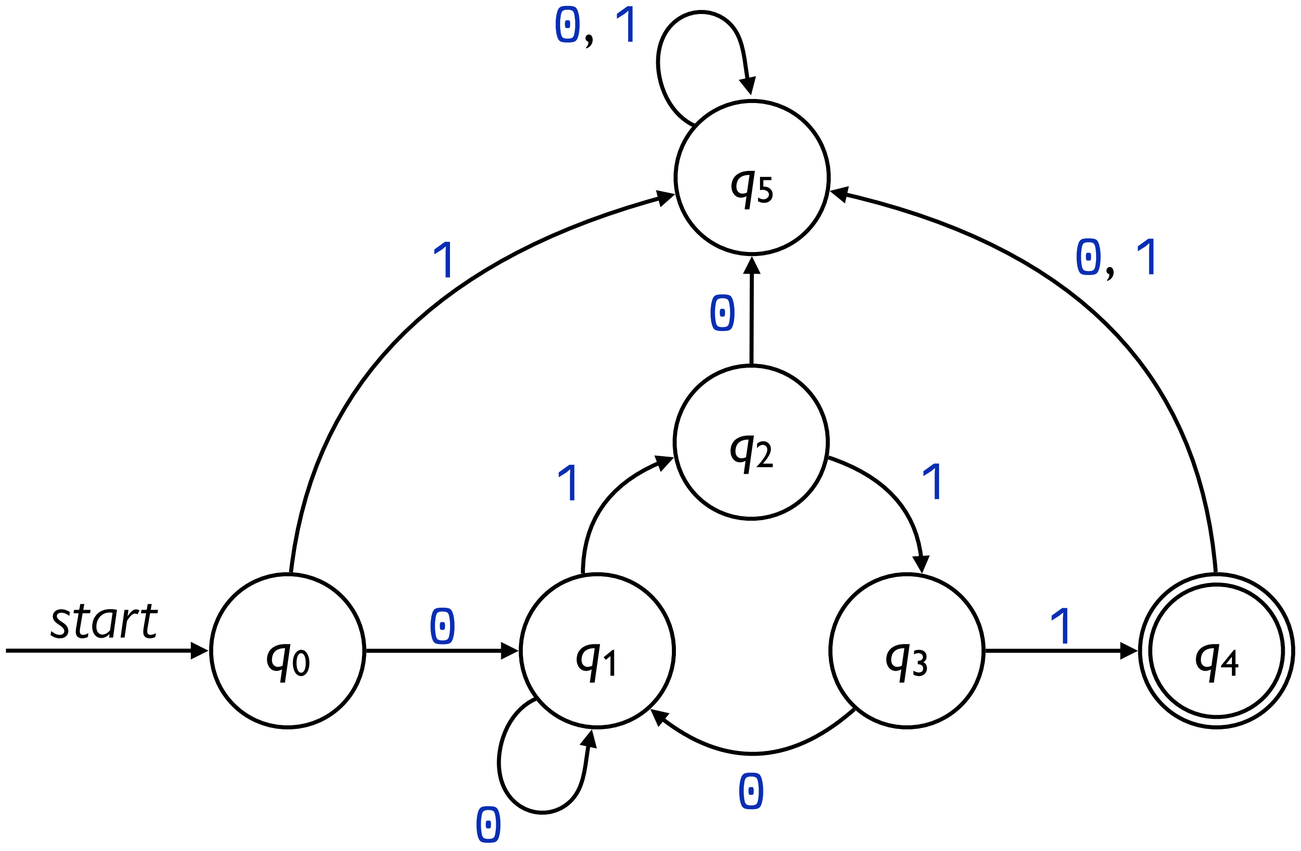
If we show that we can break ***all*** possible DFAs this way, then that means we've shown there's ***no*** DFA that recognizes *B*, and *B* must not be regular.

An important observation

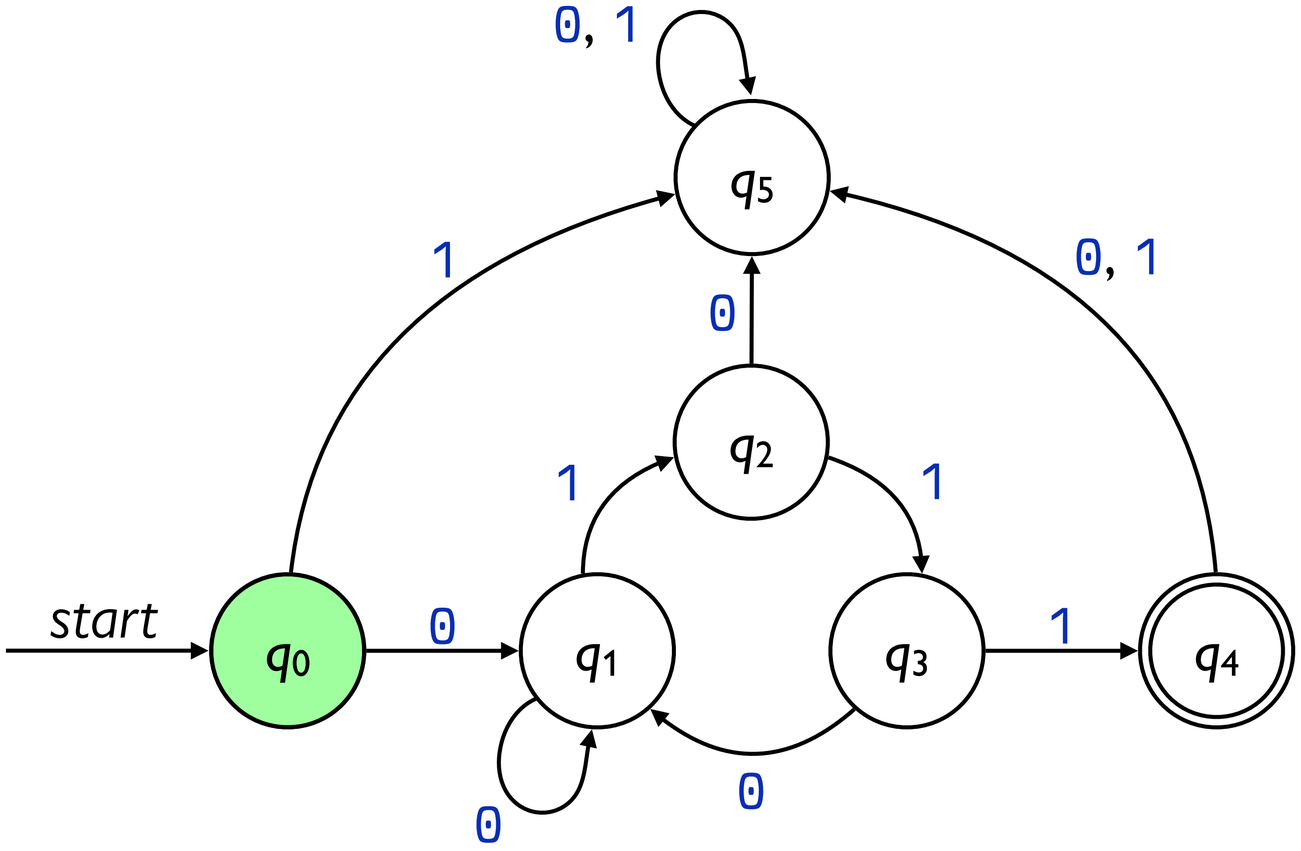
Recall from CMPU 145 the *Pigeonhole Principle*:

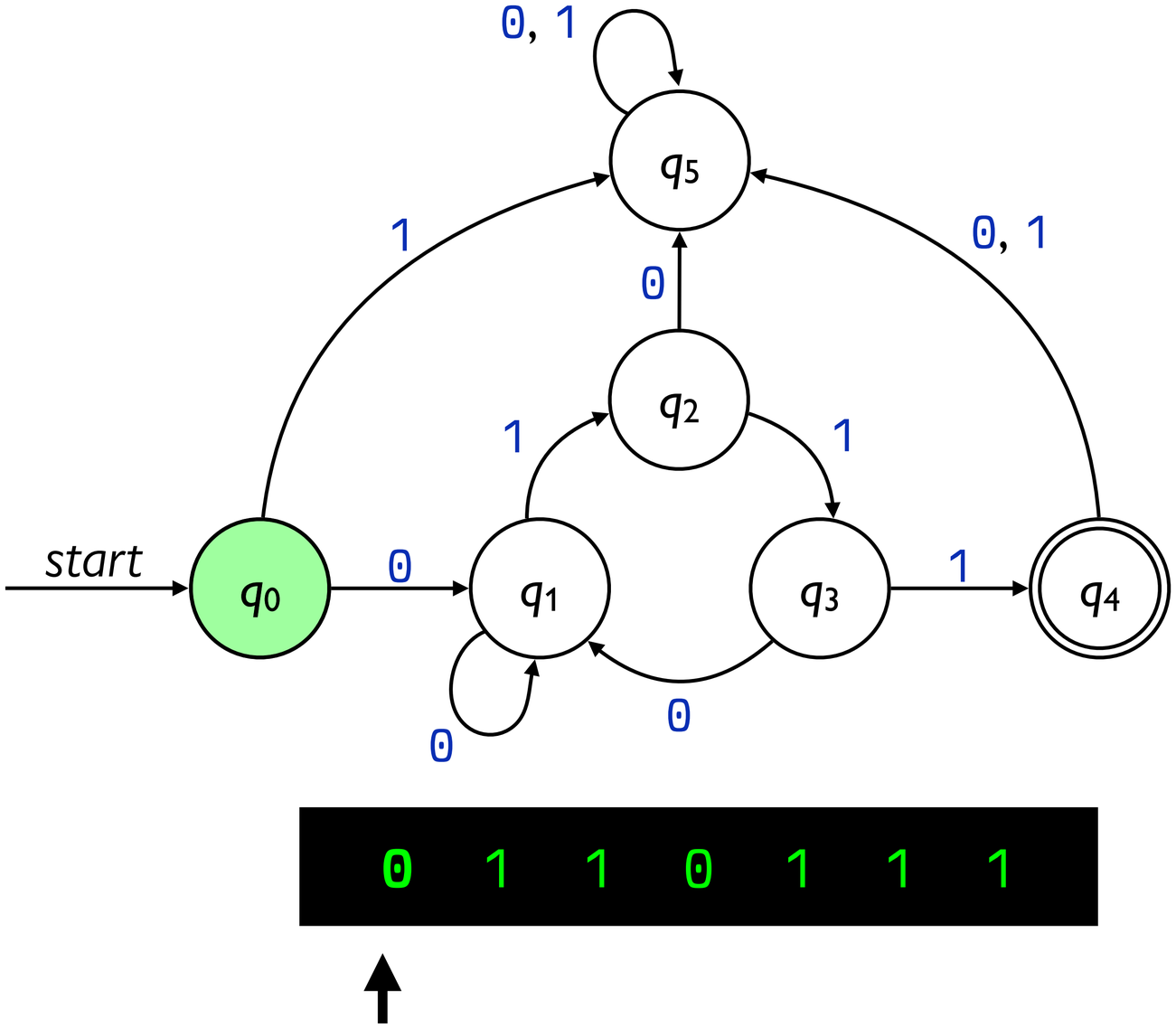
If you place n items (pigeons) in m boxes (pigeonholes), where $n > m$, then at least one of the boxes must have more than one item.

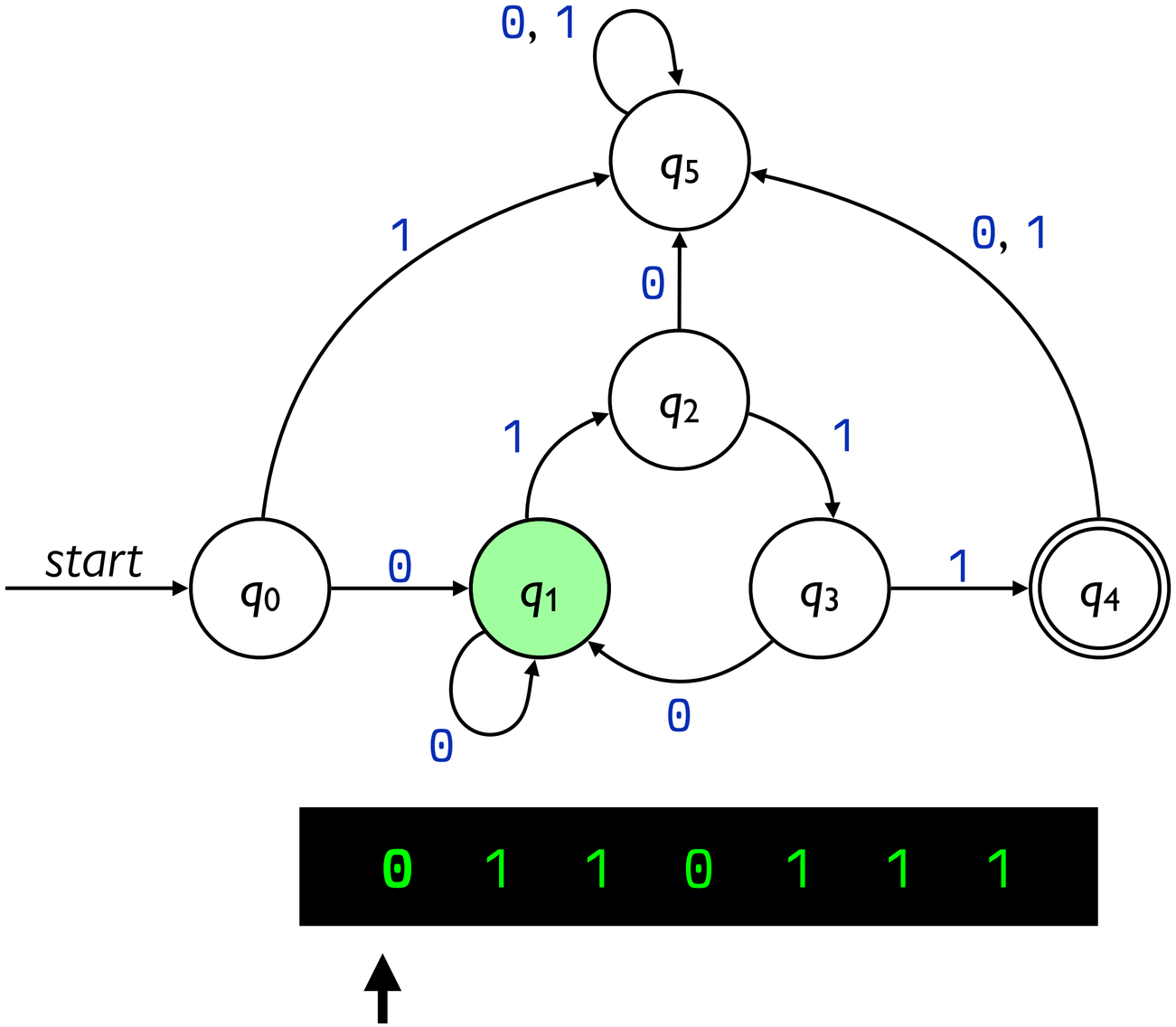

0, 1
q_5
1
0, 1
0
q_2
1
1
start
q_0
0
q_1
q_3
1
q_4
0
0

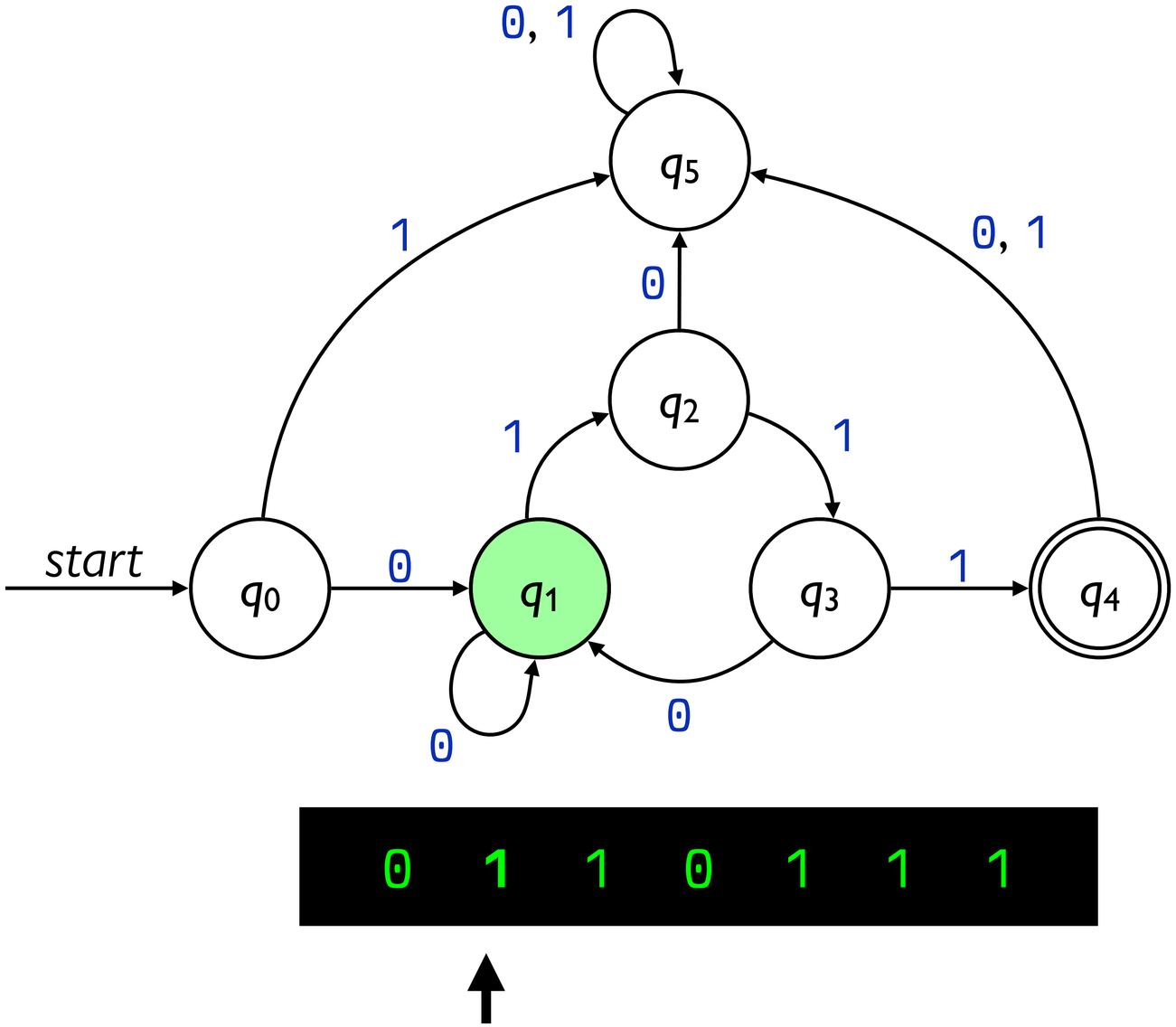


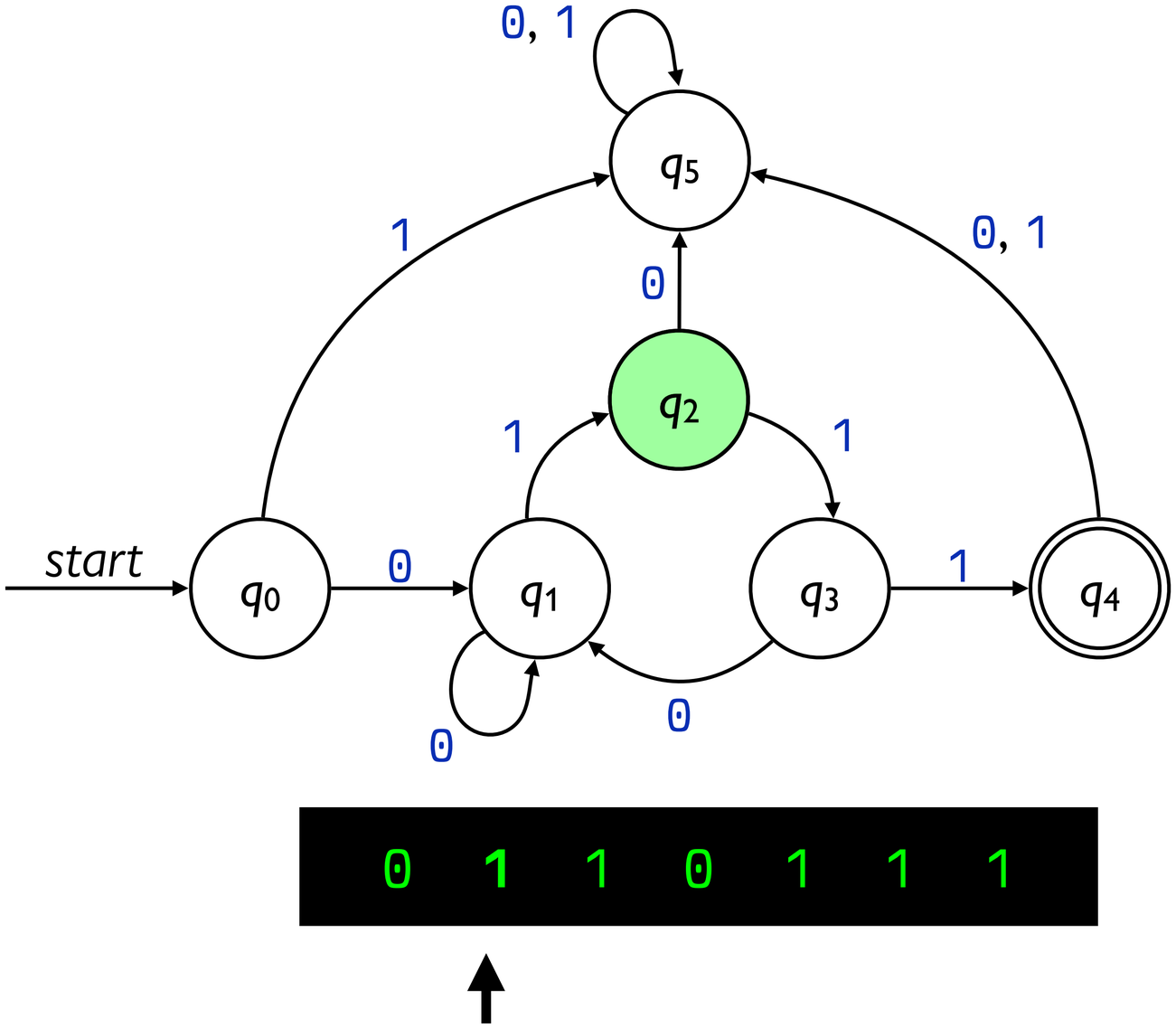
0 1 1 0 1 1 1

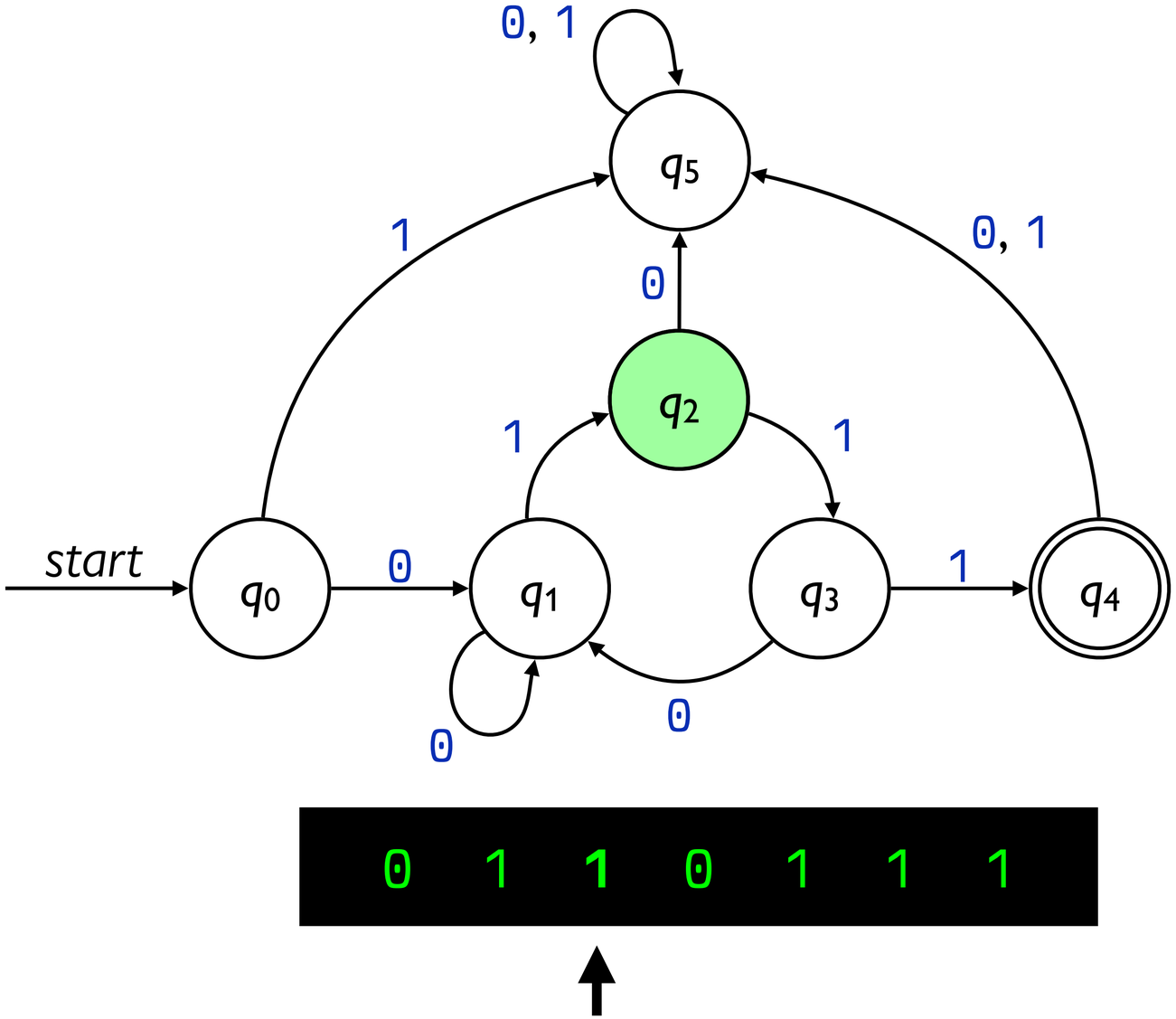

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

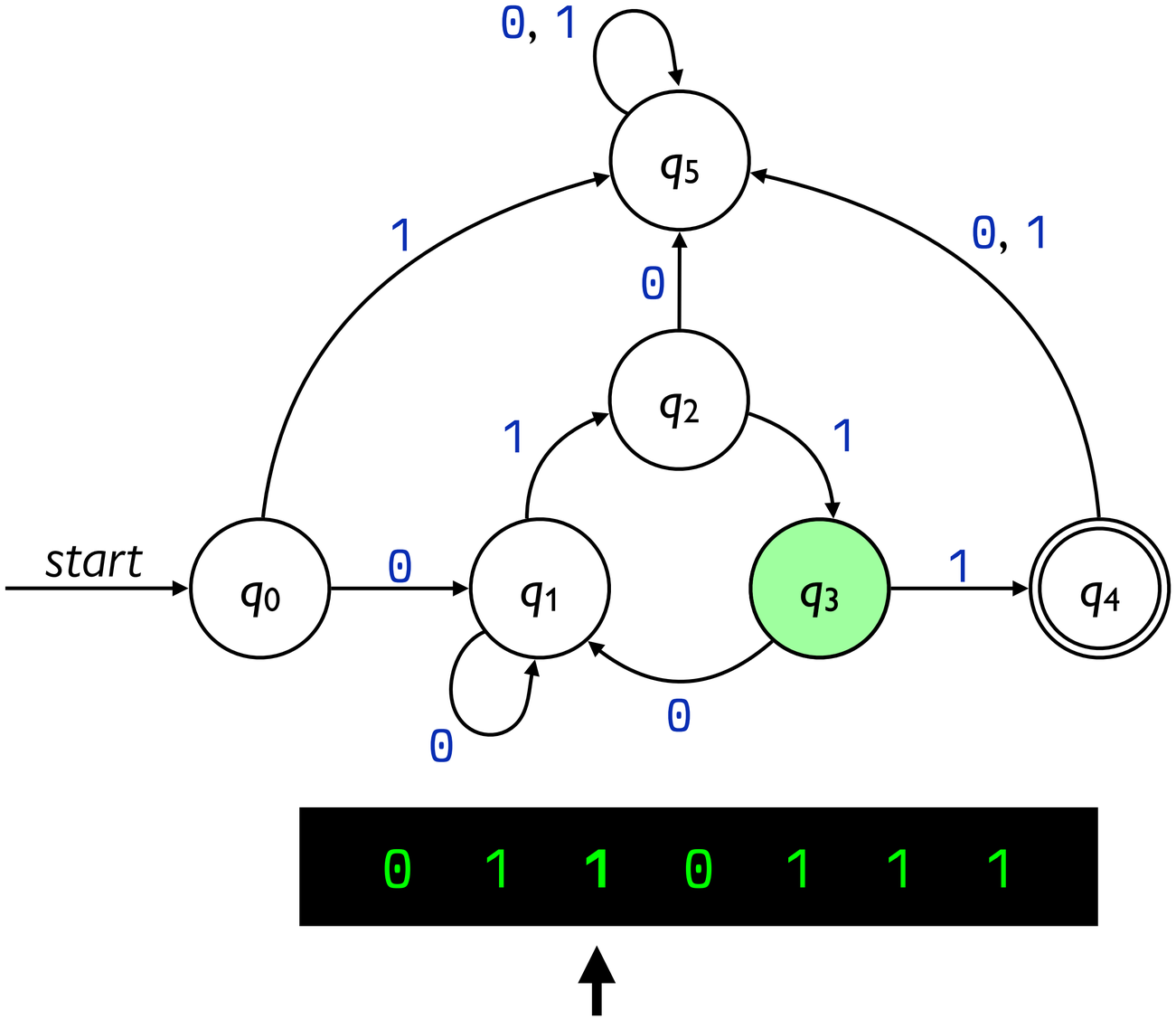

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

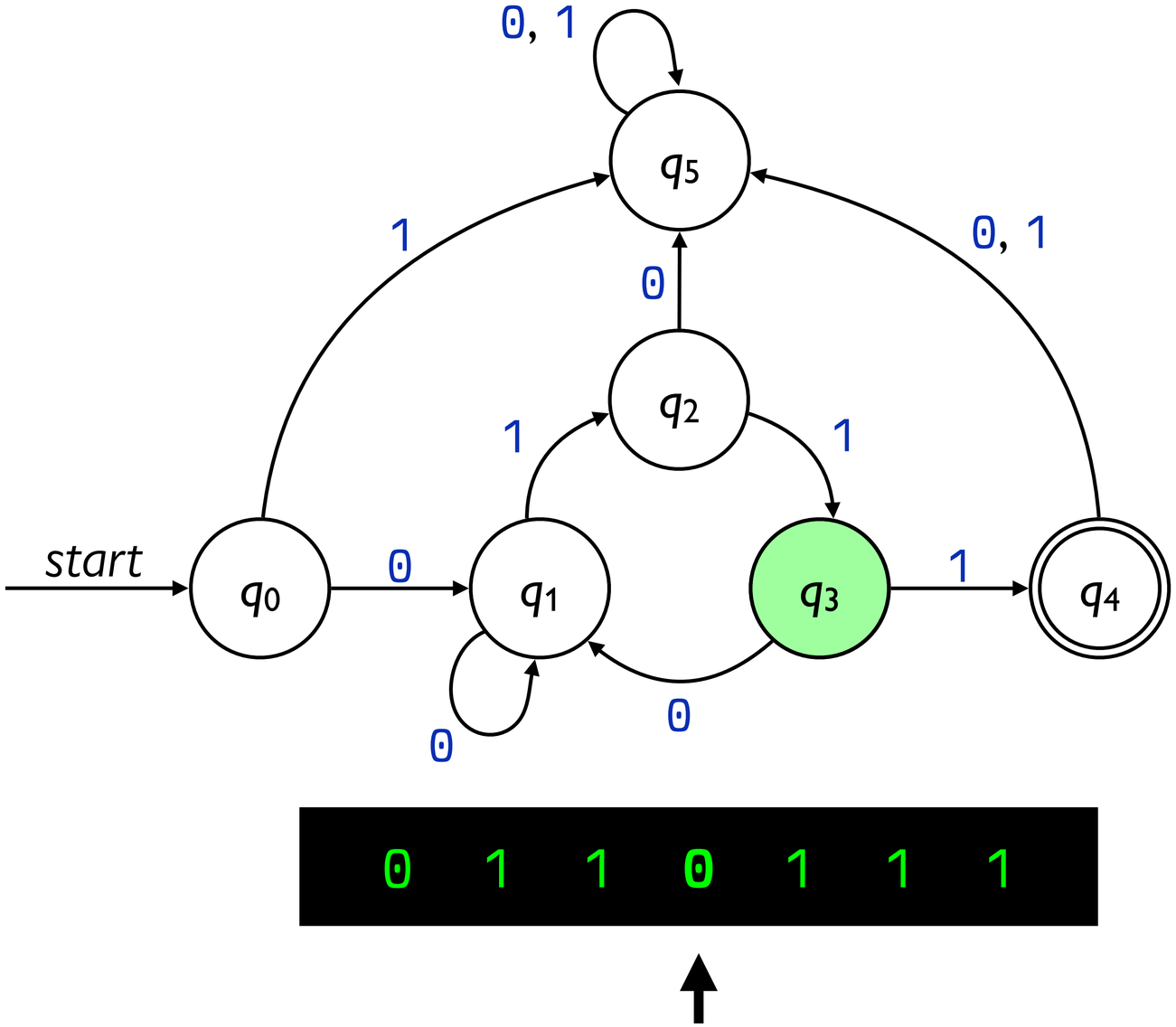

0, 1
q_5
1
0, 1
0
q_2
1
1
start
q_0
0
q_1
q_3
1
q_4
0
0
0 1 1 0 1 1 1

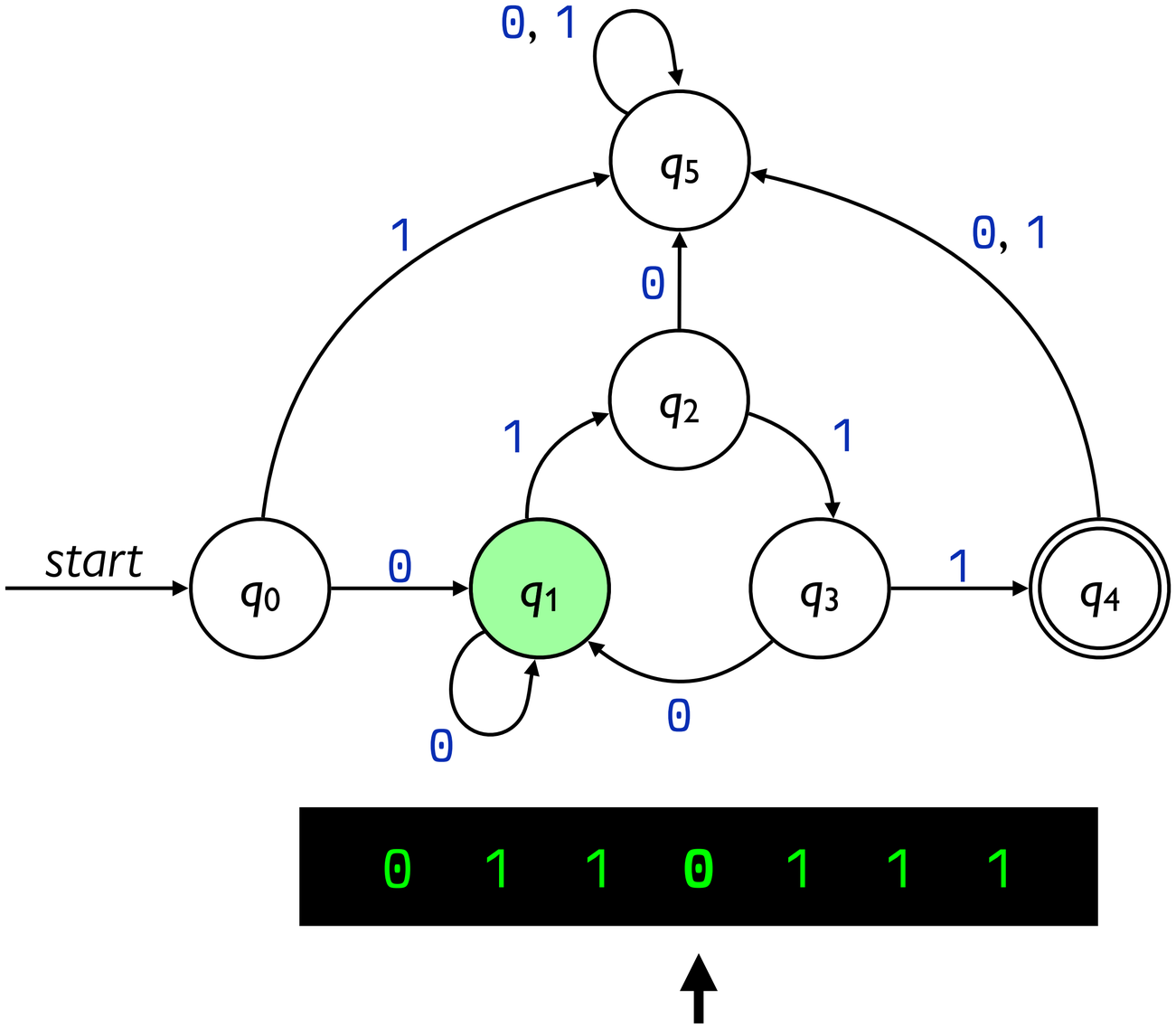

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

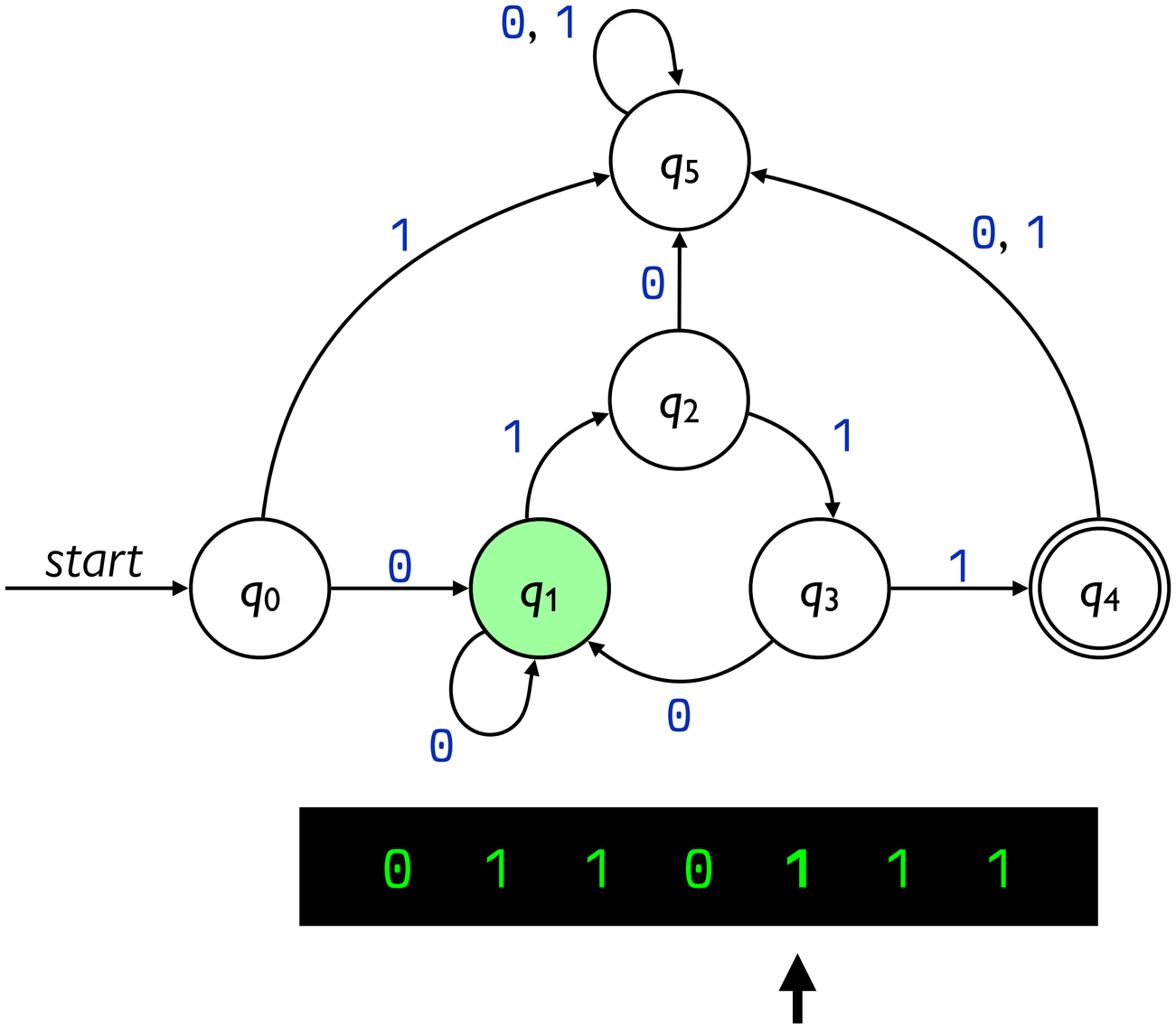

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

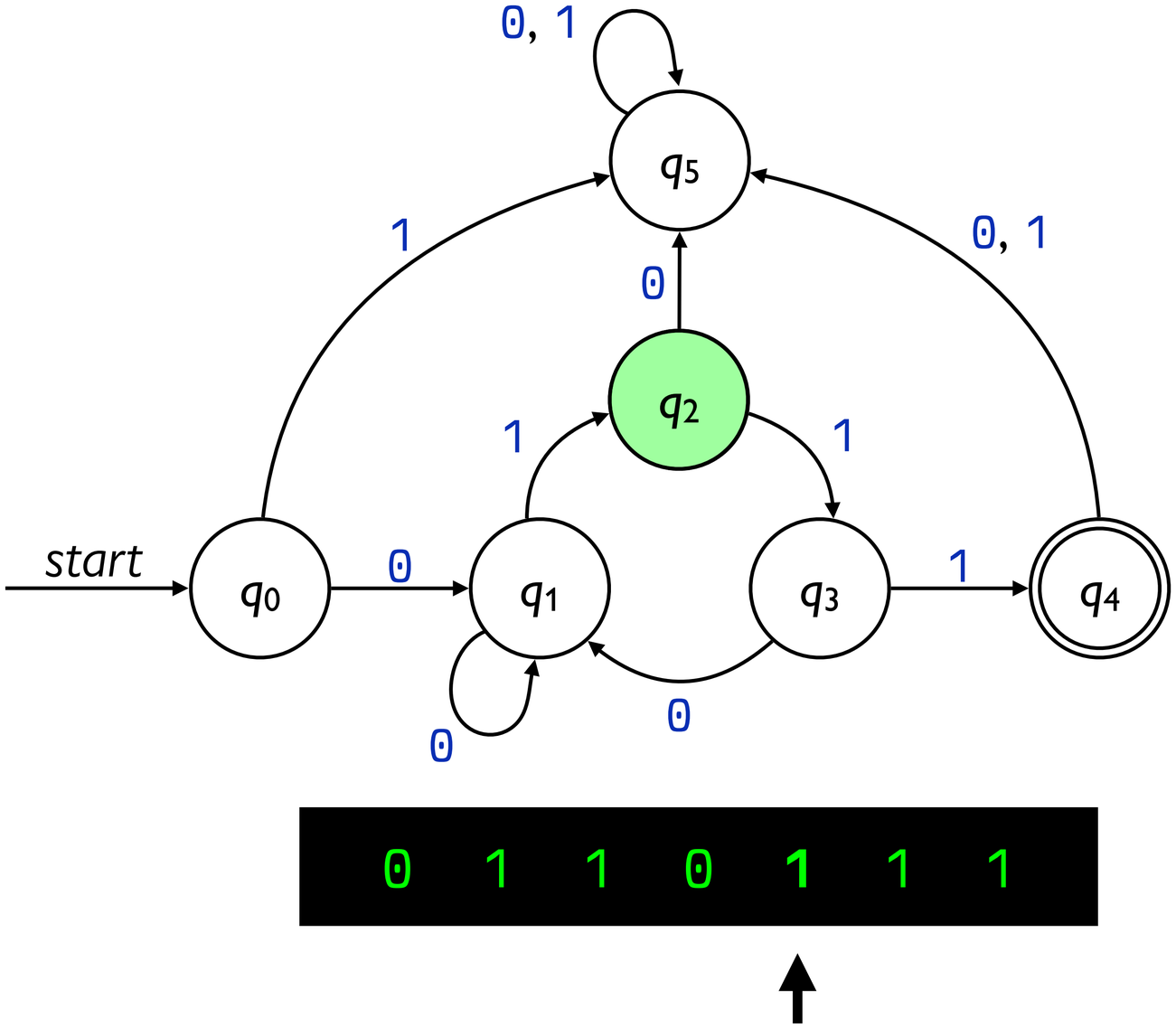

0, 1
q_5
1
0, 1
0
q_2
1
1
start
q_0
0
q_1
q_3
1
q_4
0
0
0 1 1 0 1 1 1

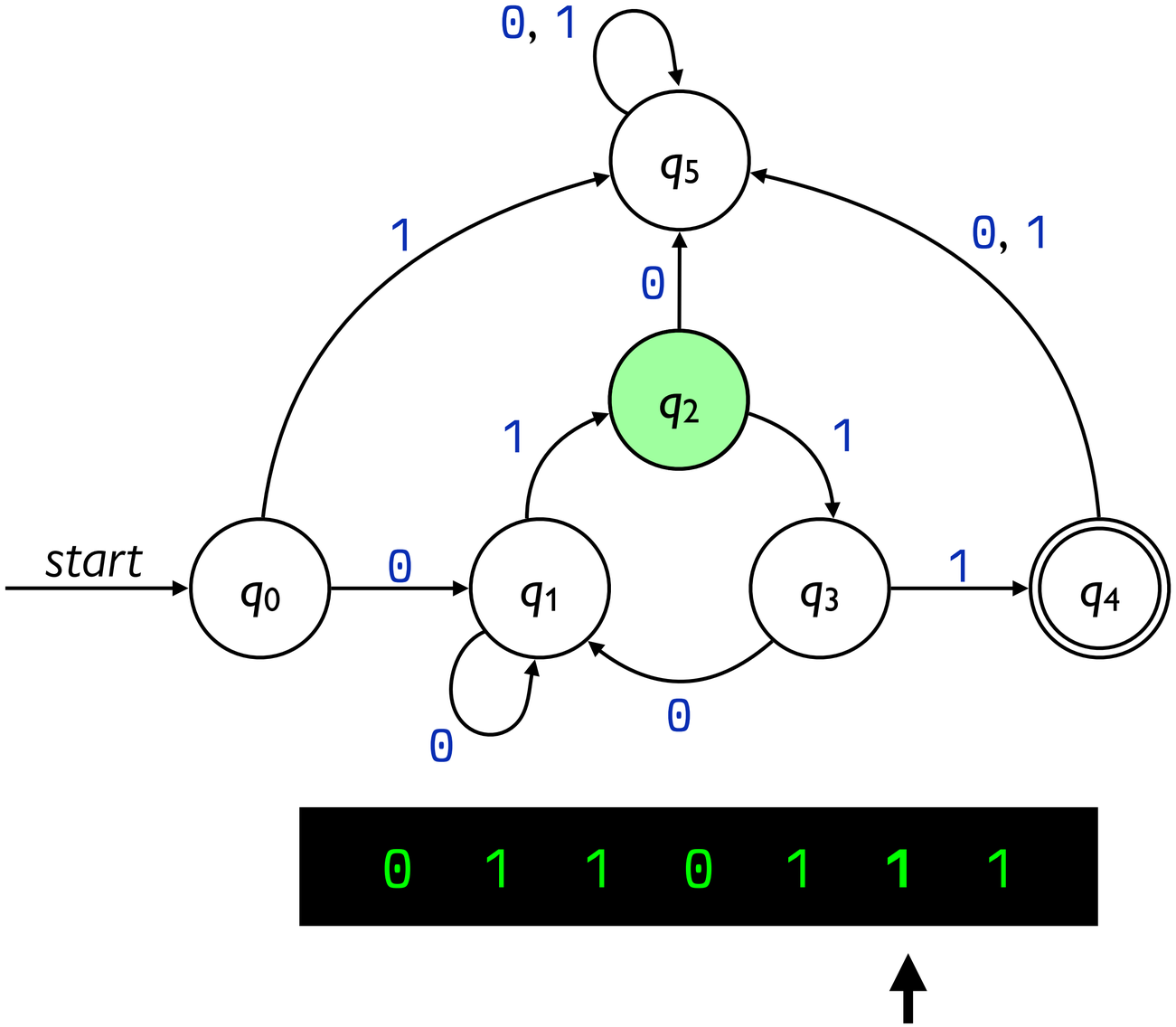

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

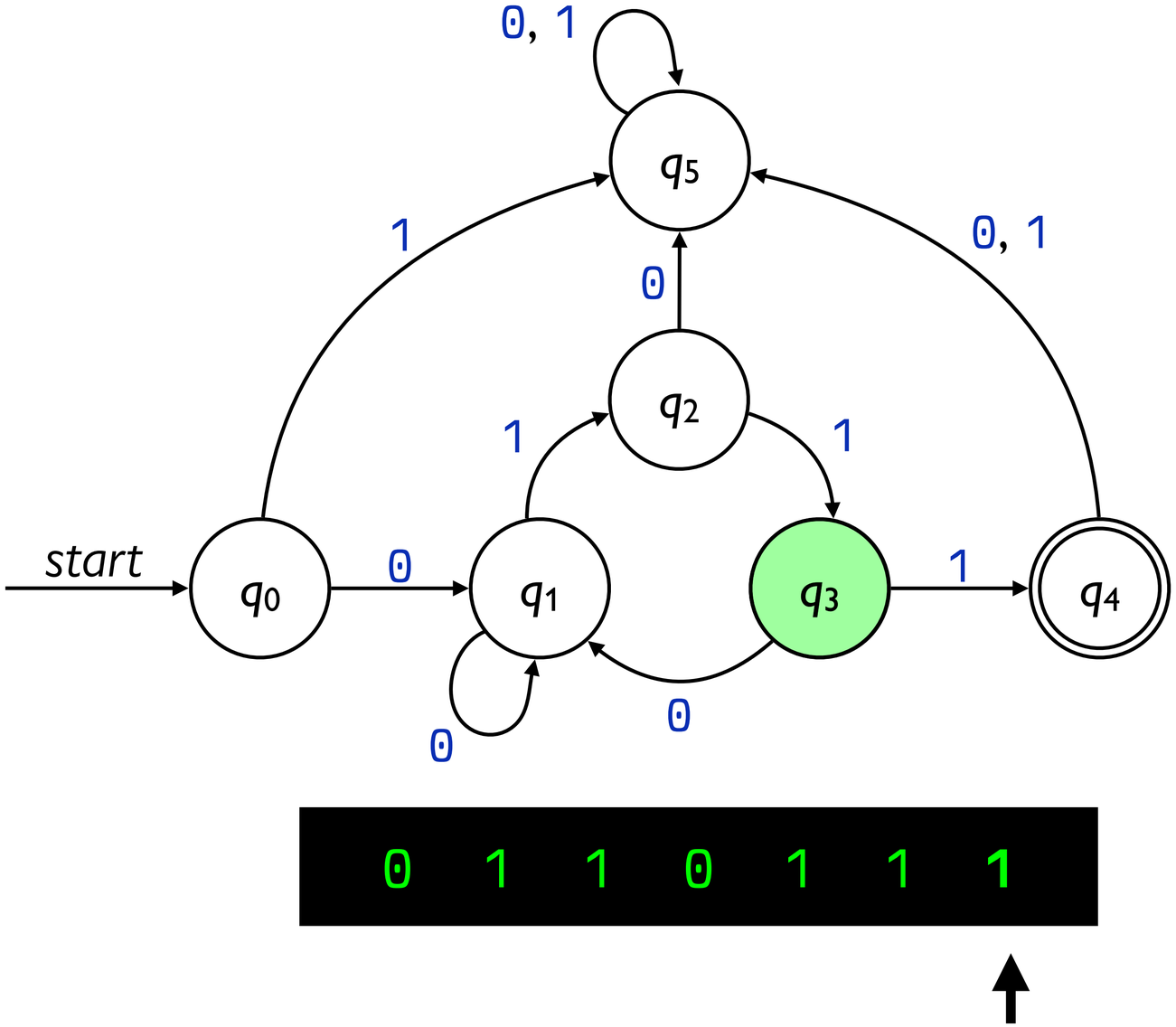

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

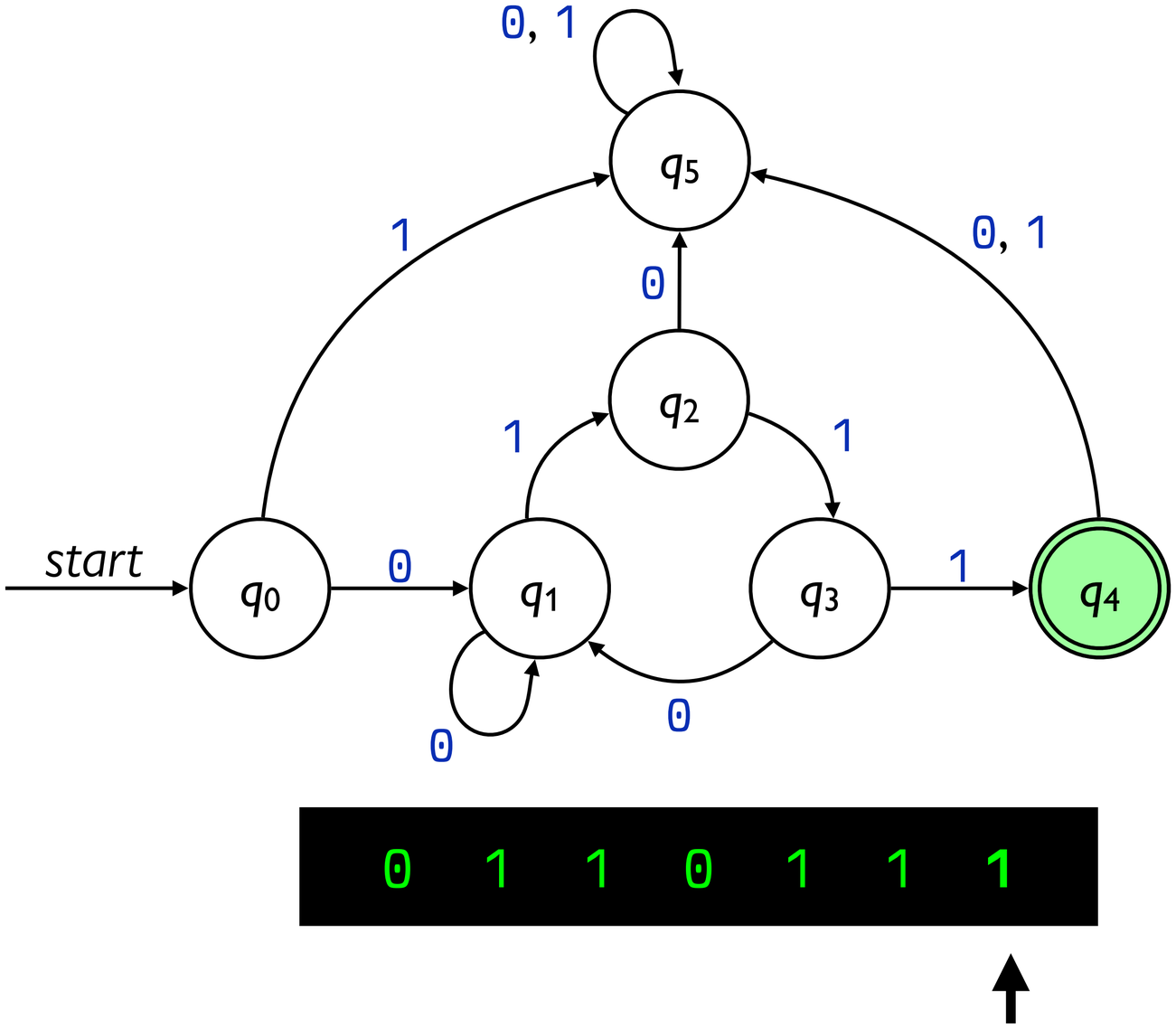

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

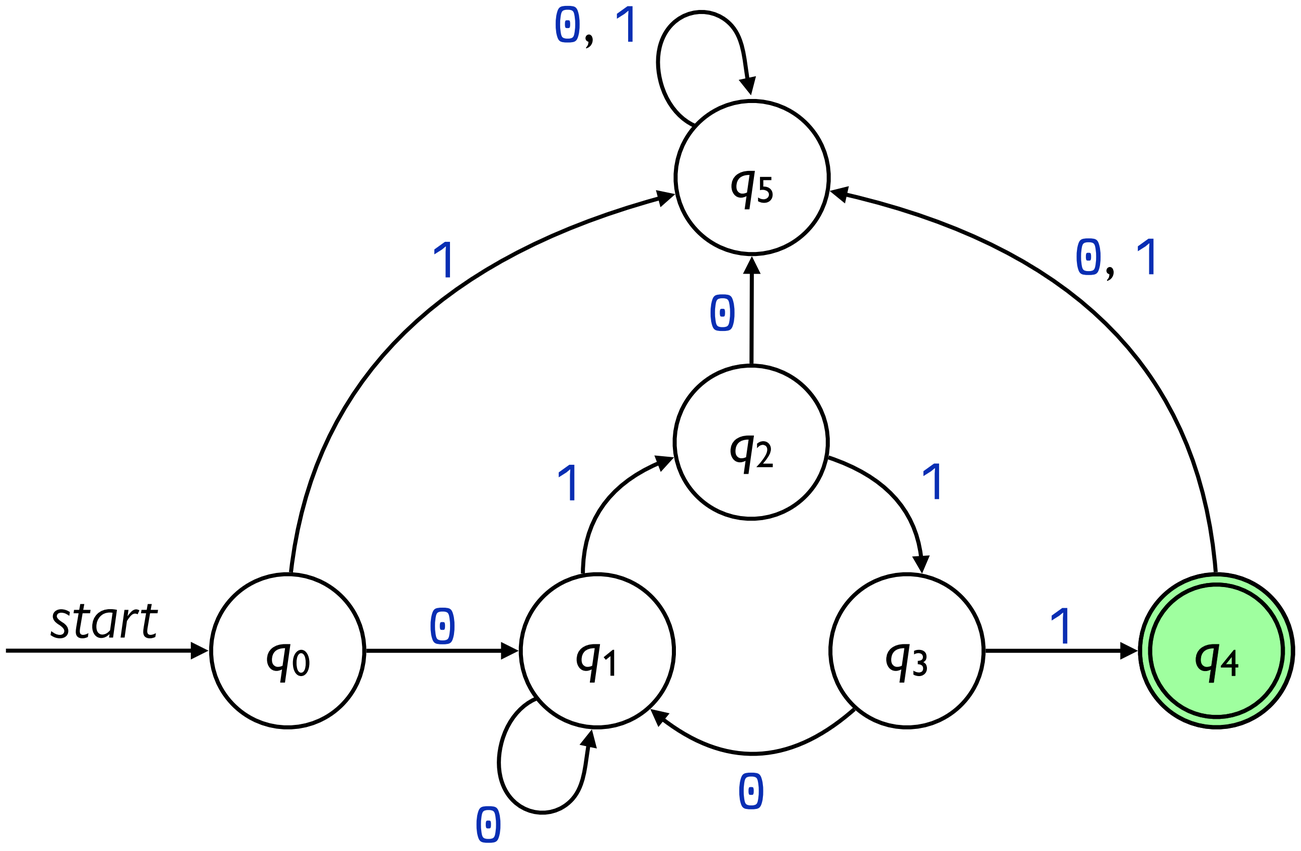

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1


0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

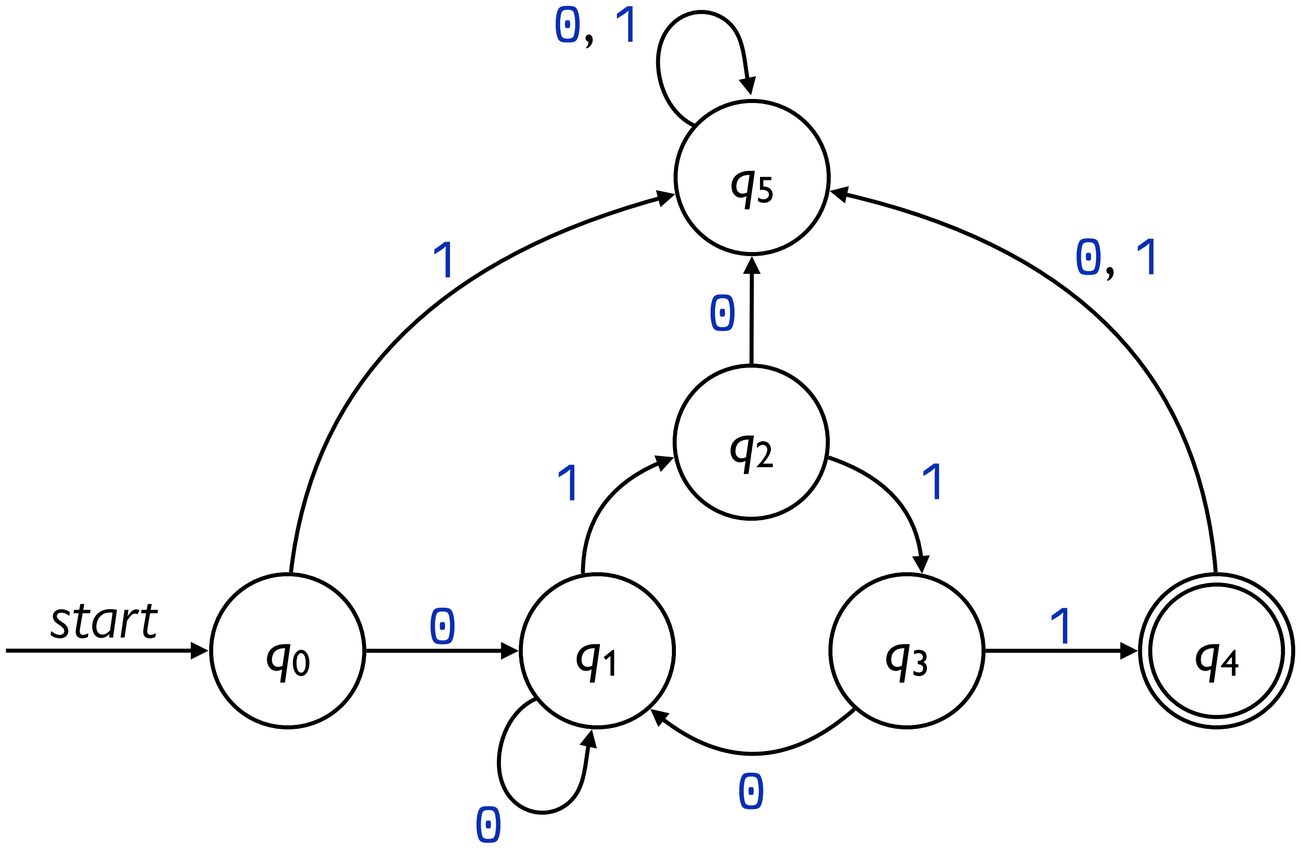

0, 1
q_5
1
0, 1
0
q_2
1
1
start
q_0
0
q_1
q_3
1
q_4
0
0
0 1 1 0 1 1 1

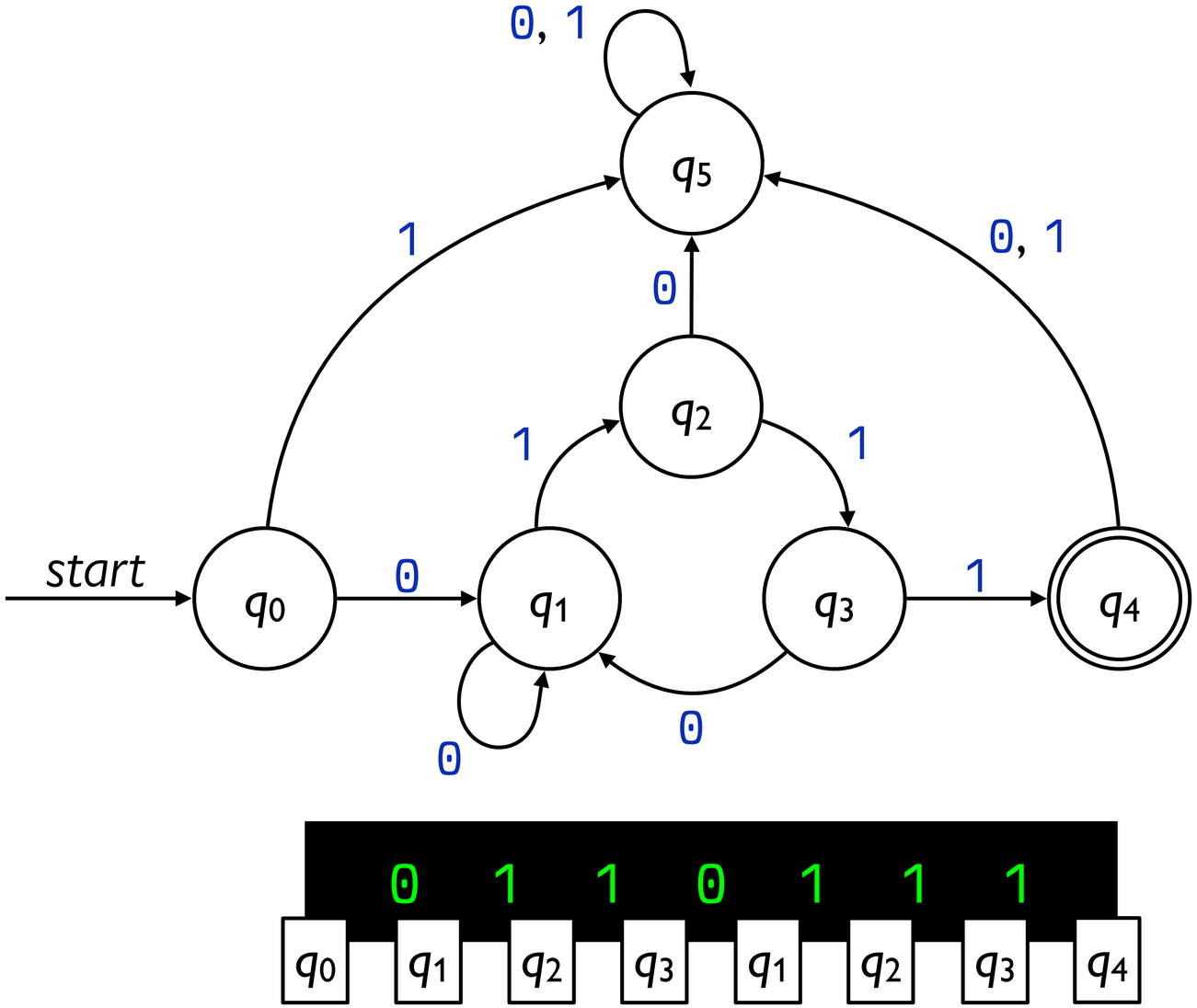

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

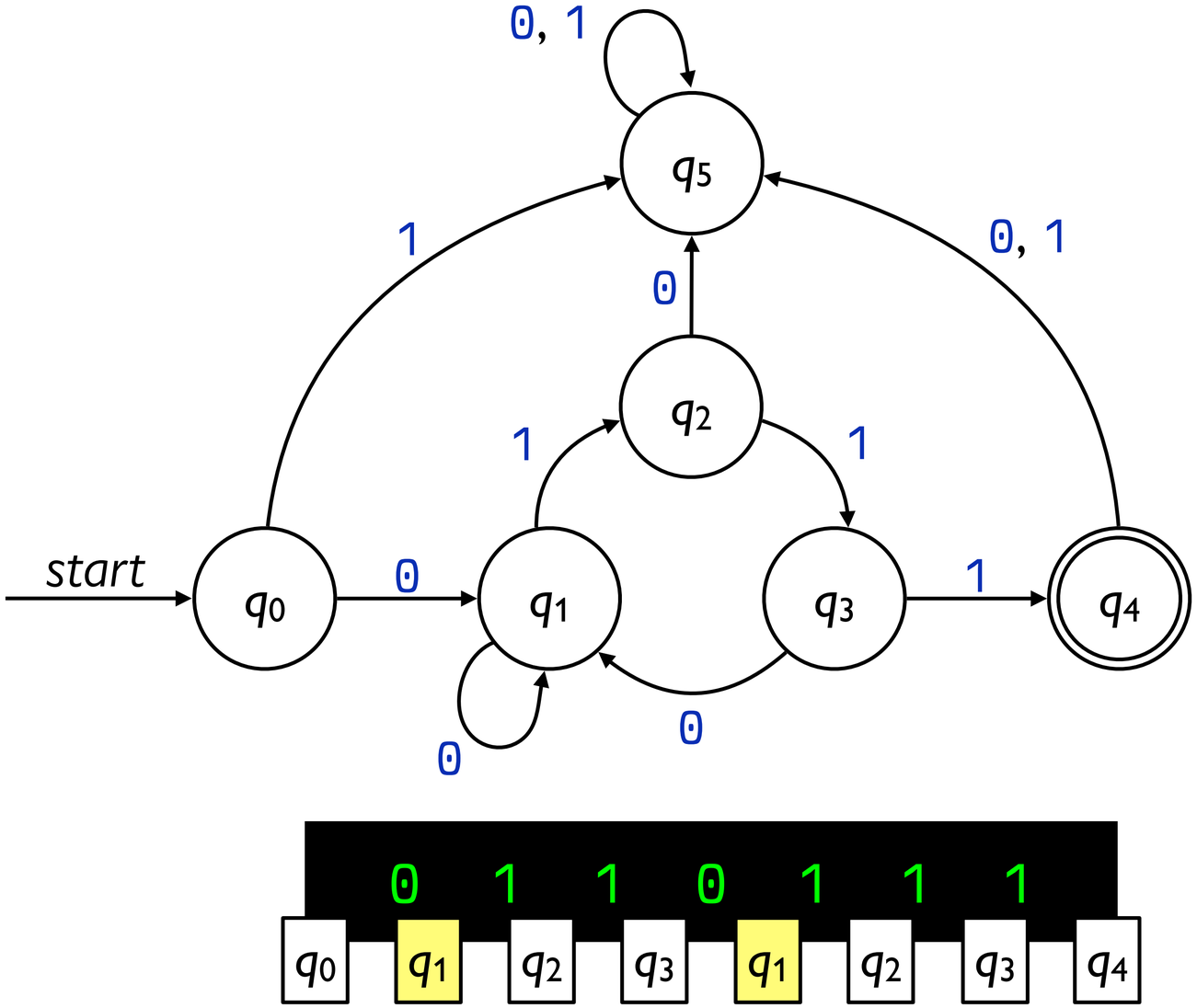

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1

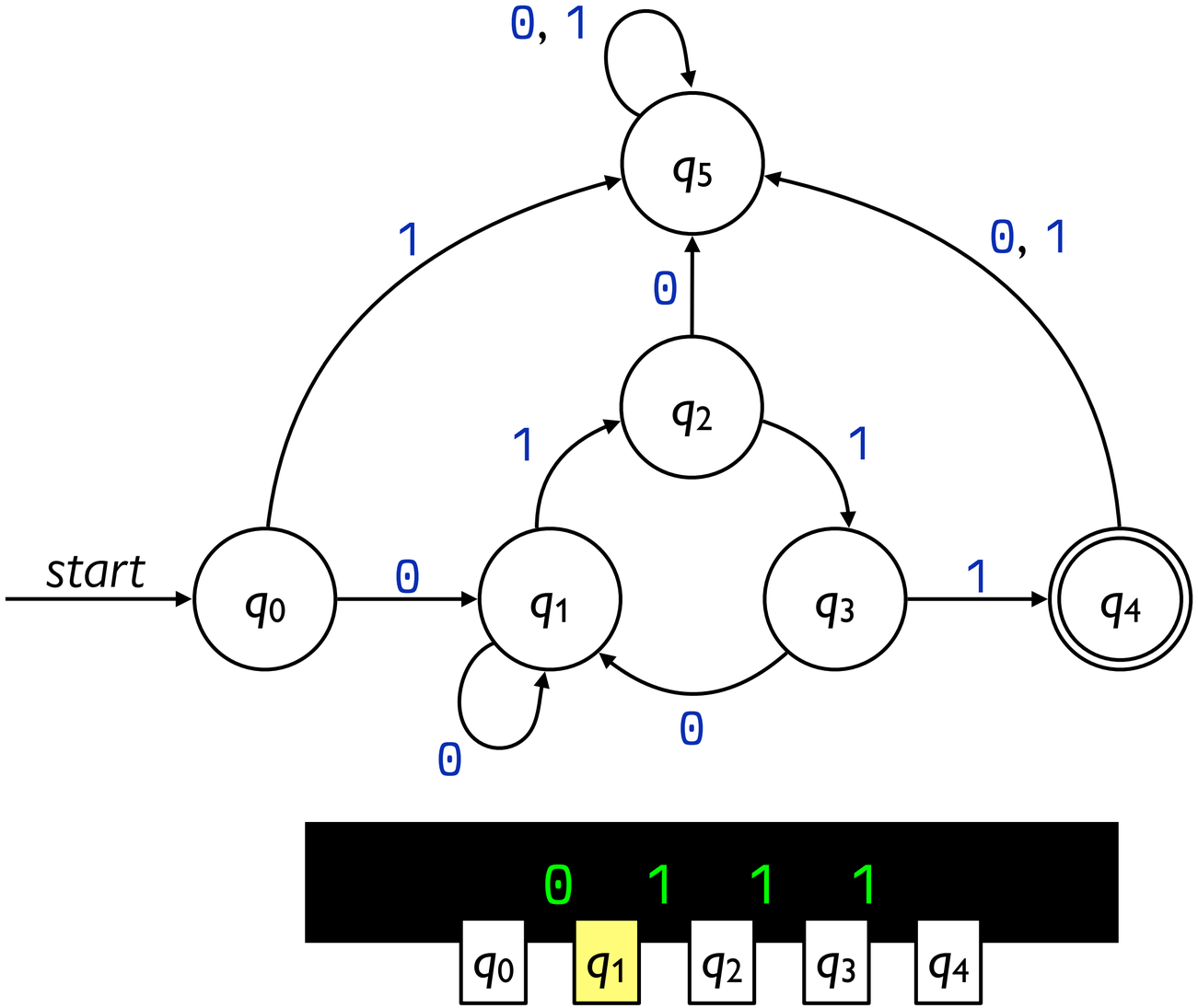


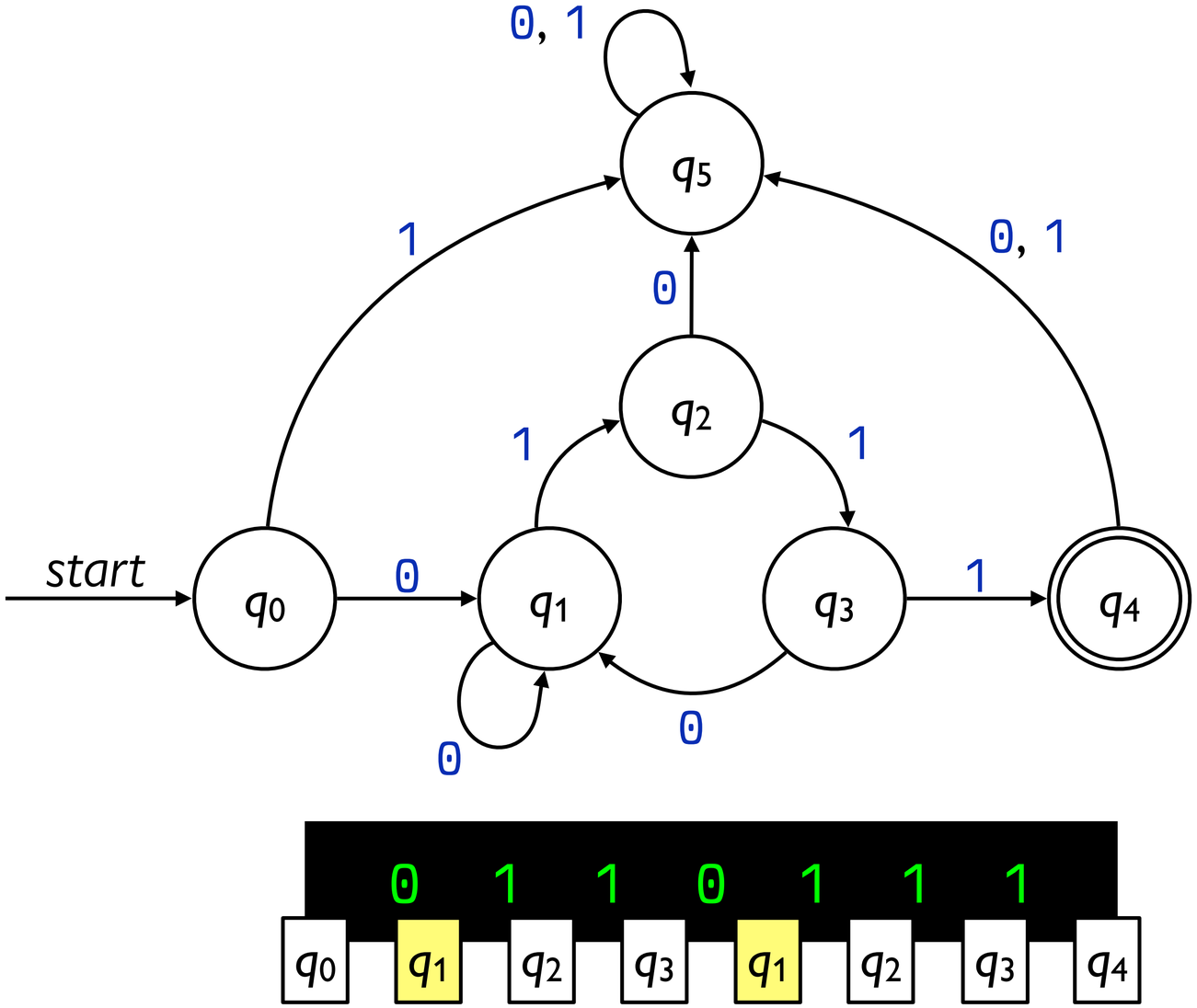
0 1 1 0 1 1 1

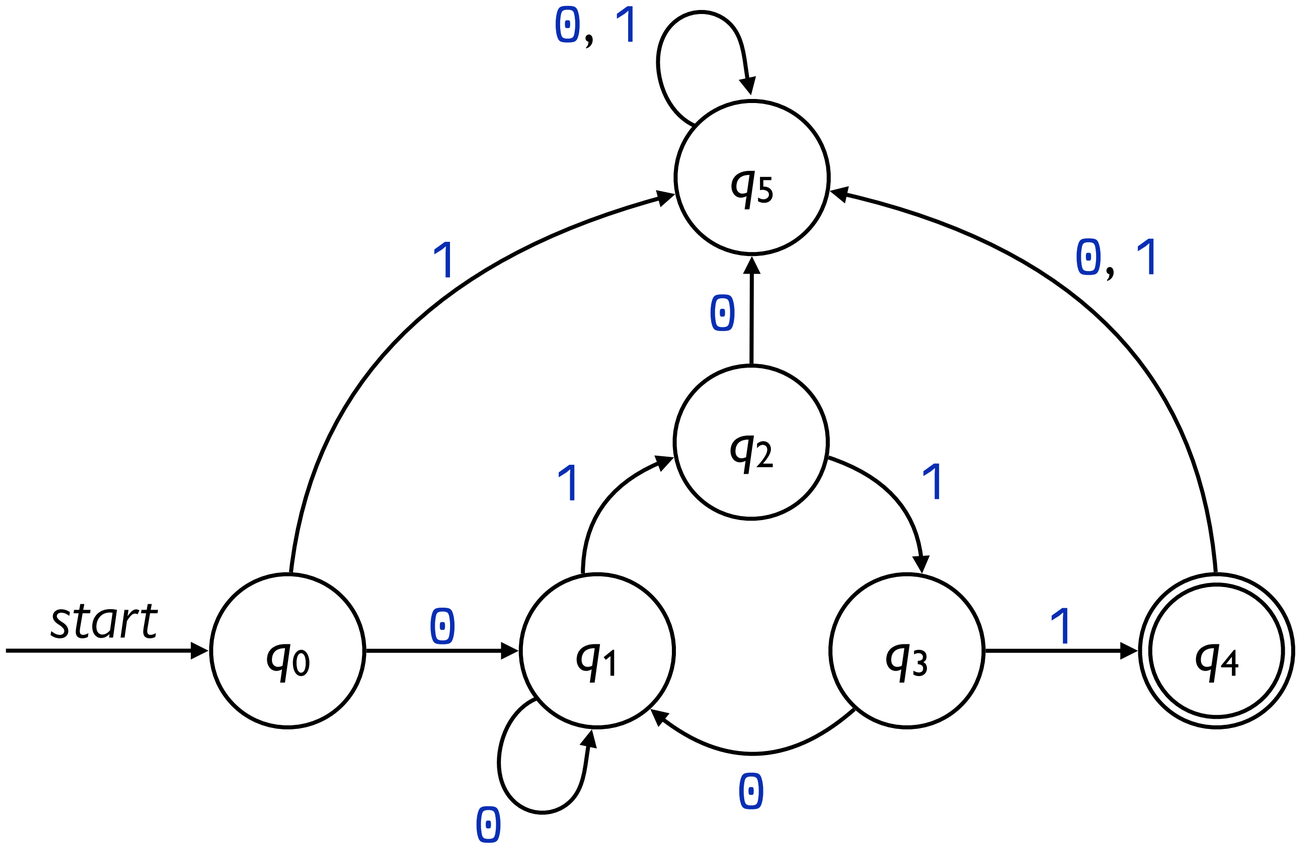

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1
0 1 2 3 4 5 6 7

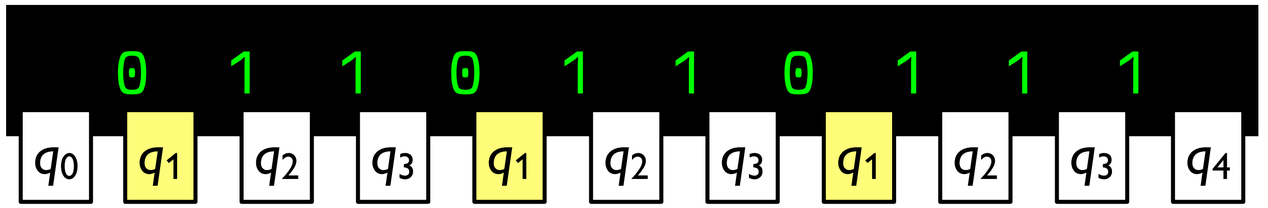

0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1
q0 q1 q2 q3 q1 q2 q3 q4


0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1
q0 q1 q2 q3 q1 q2 q3 q4


0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 1
q0
q1
q2
q3
q4


0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0
0 1 1 0 1 1 1
q0 q1 q2 q3 q1 q2 q3 q4


0, 1
q5
1
0, 1
0
q2
1
1
start
q0
0
q1
q3
1
q4
0
0


0 1 1 0 1 1 0 1 1 1
q0 q1 q2 q3 q1 q2 q3 q1 q2 q3 q4

Visiting multiple states

Let M be a DFA with p states.

Any string s accepted by M that's at least p characters long must visit some state twice within the first p characters.

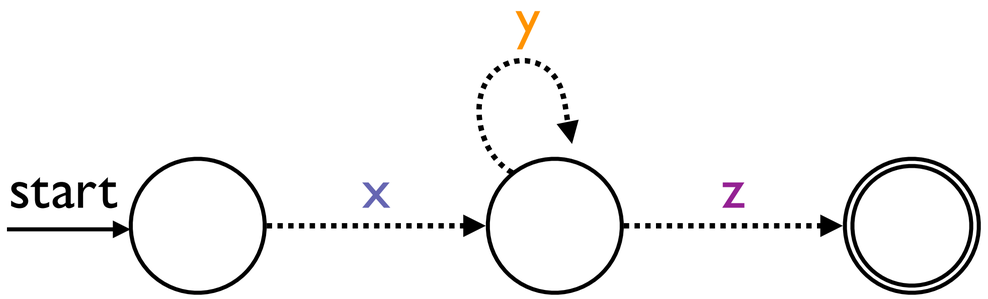
- Number of states visited is equal to $p + 1$.
- By the Pigeonhole Principle, some state is duplicated.

The substring of s between those revisited states can be removed, duplicated, tripled, etc. without changing the fact that M accepts s.

Informally

Let L be a regular language.

If we have a string $s \in L$ that is "sufficiently long", then we can split the string into three pieces and "pump" the middle:



We can write $s = xyz$ such that xy^0z, xy^1z, xy^2z, … are all in L.

Here's how we find the string that breaks *M*.

Imagine Anna and Elsa are playing a strange game:

- Elsa proposes a DFA *M* for the language.
- Anna gives Elsa a "test" string *s* that *M* is supposed to accept.
- But then she uses the information that Elsa reveals to concoct another string that breaks *M*.

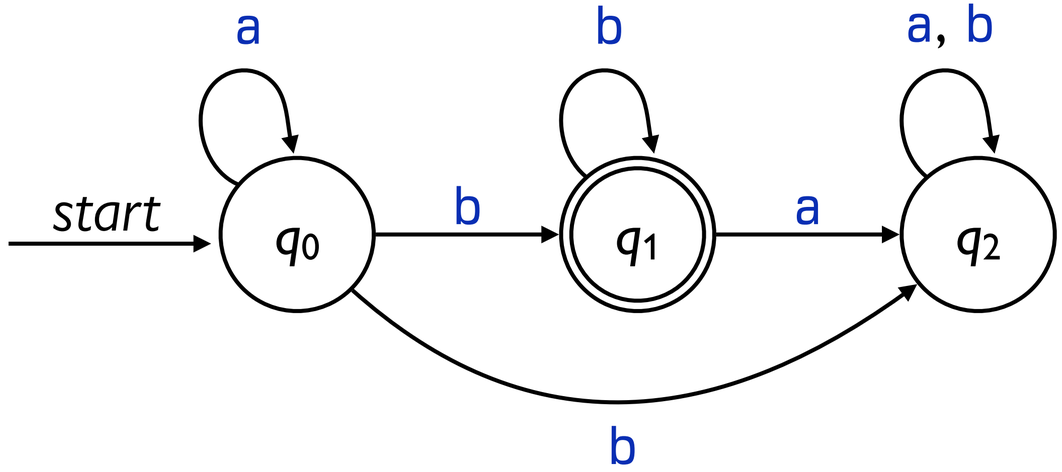
An example game dialogue

Anna: The language $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

Elsa: Oh yes, it is!

Anna: Really? Then show me a DFA that recognizes it.

Elsa: Here's one:

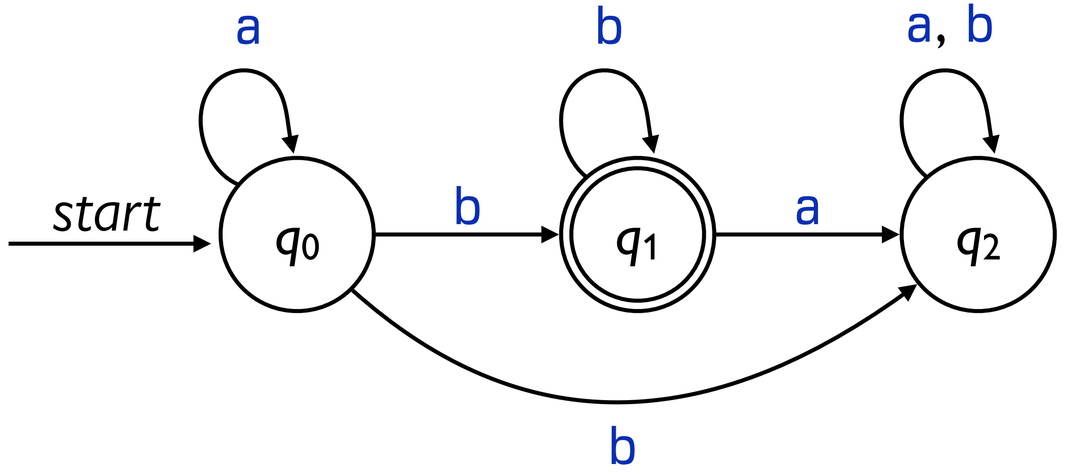


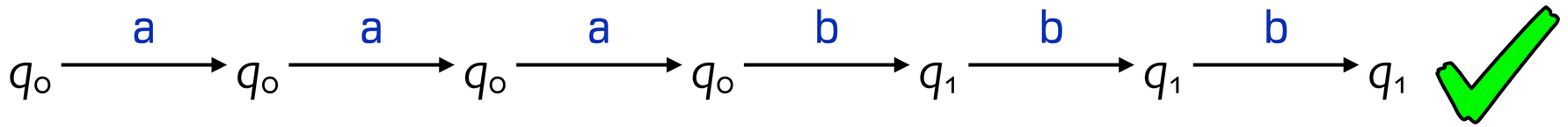
Anna: How many states does it have?

Elsa: 3.

Anna: Call that p. Does your automaton accept the string $s = a^p b^p$?

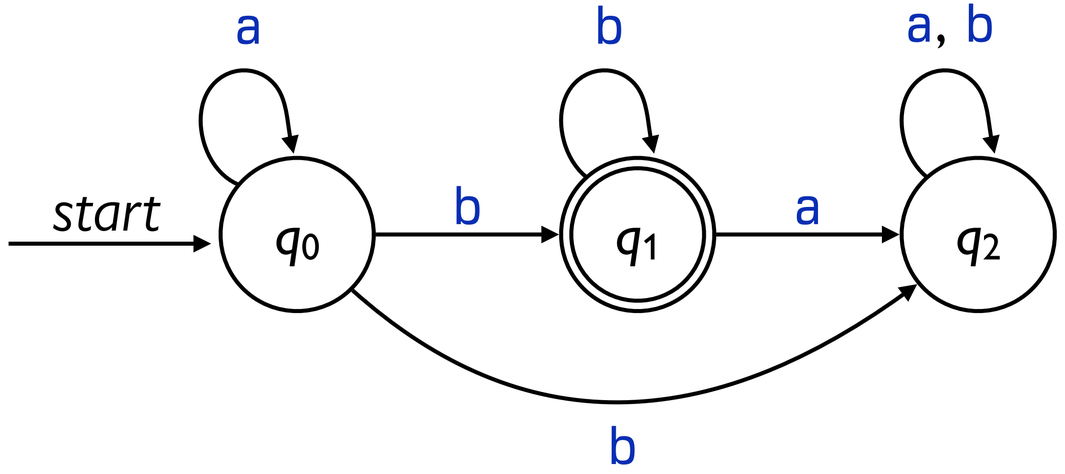
Elsa checks it:

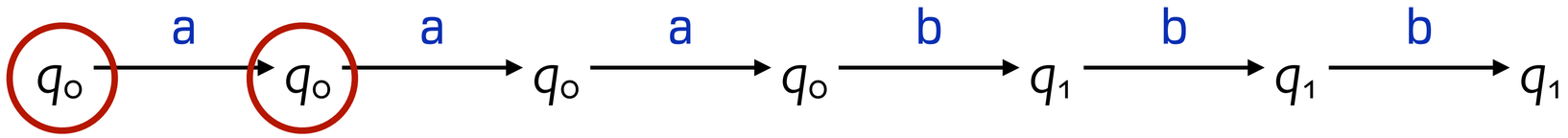




Elsa: Of course!

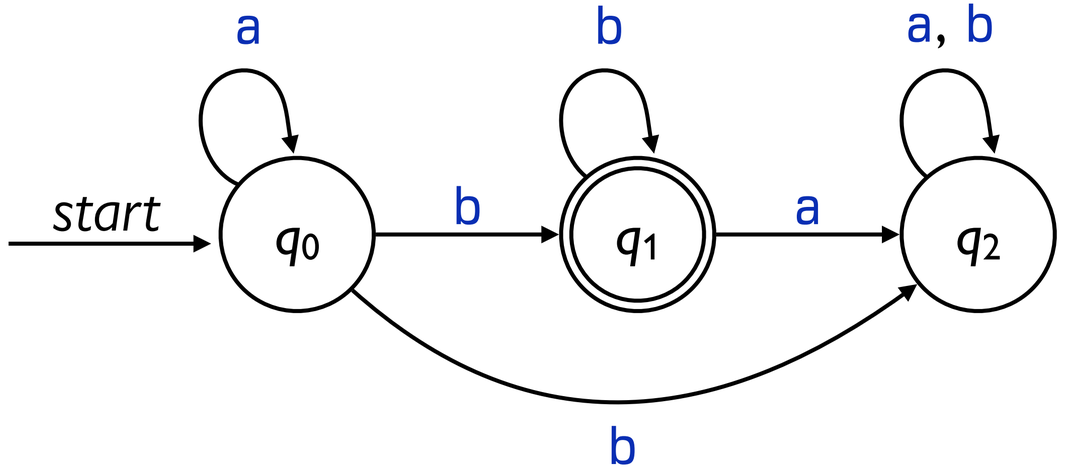
Anna: Does this run visit a state twice while reading in the first half of the string?

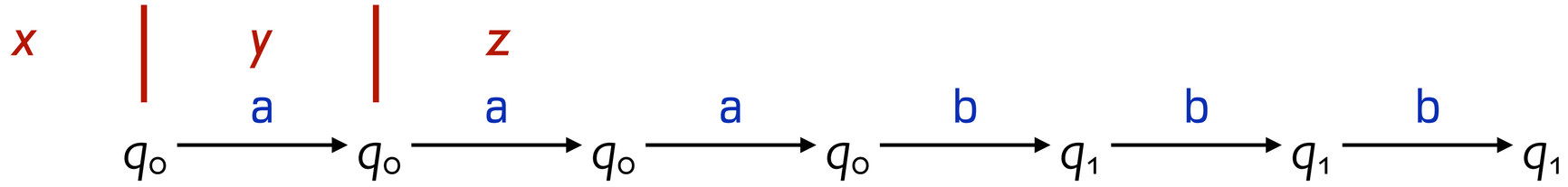




Elsa: Yes, q_0.

Anna: What are the strings that it reads up to the first visit, between the first and second visits, and after the second visit?



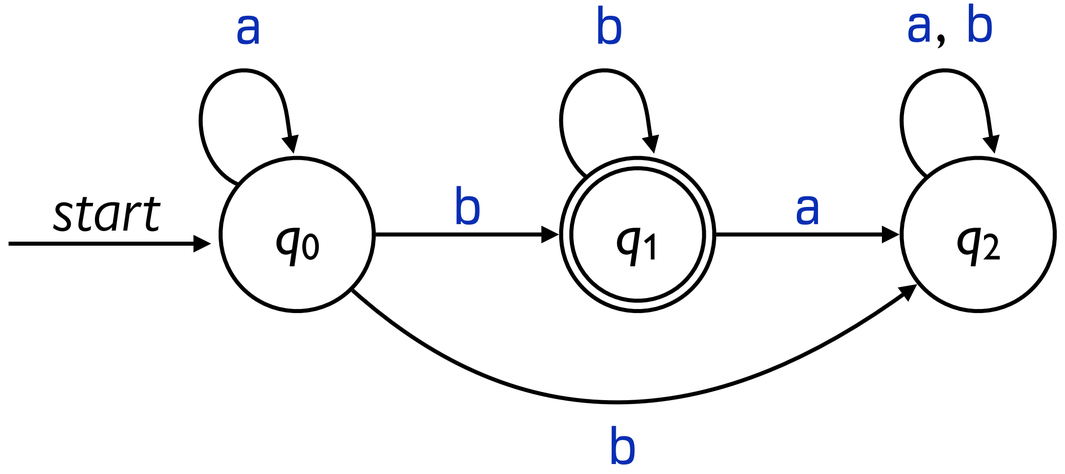


Elsa: $x = \varepsilon$, $y =$ a, $z =$ aabbb

Anna: So, does your automaton accept ***this*** string?

$s = xy^2z = \varepsilon aaabb = aaabb$

Elsa: Let's try it…



$q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1$ ✓

Elsa: Oh no! 😞 It does, but that's not in *B*. I lose!

A more general game dialogue

We want to show that Elsa will lose no matter what automaton she designs for the language *B*.

Let's write a version of it that removes Elsa's specific choices.

Anna: The language $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

Elsa: Oh yes, it is! Here, I can show you an automaton that –

Anna: I don't need to see it. Just count how many states it has.

Elsa: It has –

Anna: Don't tell me. Just call it p.

Elsa: Okay.

Anna: Does it accept the string $s = a^p b^p$?

Elsa: Yes.

Anna: On reading the first p symbols of s, your automaton goes through $(p + 1)$ configurations – the starting configuration plus one for each symbol.

Since your automaton has only p states, by the Pigeonhole Principle, it visits some state at least twice, right?

Elsa: Yes.

Anna: Could you find the substrings that it reads

—up to the first visit,

—between the first and second visits, and

—after the second visit?

Elsa: Yes, they're –

Anna: I don't need to know. Just call them *x*, *y*, and *z*.

Anna: Does your automaton also accept xy^2z?

Elsa: Yes.

Anna: But y consists only of a, so xy^2z has more a
than b. That means it isn't in the language
$B = \{a^n b^n \mid n \in \mathbb{N}_0\}$.

Anna: Does your automaton also accept xy^2z?

Elsa: Yes.

Anna: But y consists only of a's, so xy^2z has more a's than b's. That means it isn't in the language $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$.

Proof strategy:
The Pumping Lemma

The *Pumping Lemma* is a formal statement of Anna's strategy for defeating Elsa.

The Pumping Lemma for Regular Languages

∀ regular language L,

∃ positive integer p such that

∀ string $s \in L$ such that $|s| \geq p$,

∃ strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

∀ regular language L,

∃ positive integer p such that

∀ string $s \in L$ such that $|s| \geq p$,

∃ strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

∀ = *Anna*

∃ = *Elsa*

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

the "pumping length"

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

Strings longer than the pumping length must have a special property

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$ *s can be broken into three pieces,*

$|xy| \leq p$

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$ *s can be broken into three pieces,*

$|xy| \leq p$ *where the first two pieces occur at the start of the string,*

$y \neq \varepsilon$

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$ *s can be broken into three pieces,*

$|xy| \leq p$ *where the first two pieces occur at the start of the string,*

$y \neq \varepsilon$ *the middle part isn't empty, and*

$xy^iz \in L$ for all $i \in \mathbb{N}_0$

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$ *s can be broken into three pieces,*

$|xy| \leq p$ *where the first two pieces occur at the start of the string,*

$y \neq \varepsilon$ *the middle part isn't empty, and*

$xy^iz \in L$ *the middle piece can be repeated zero or more times.*

Rationale for requirements in the Pumping Lemma

$y \neq \varepsilon$

Because y is what's read on the loop, it has to consist of at least one symbol.

$|xy| \leq p$

Because xy is what you get when you only take the loop once, you need that many states.

$xy^iz \in L \text{ for all } i \in \mathbb{N}_0$

Because y can be *pumped* zero or more times.

The Pumping Lemma gets its name because the repeated string is “pumped” to get more.

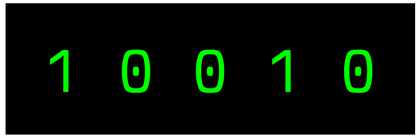
Because of the nature of finite automata, we ***can’t*** control the number of times it is pumped.

So, a regular language with strings of length ≥ p is always infinite!

A (sad) pump

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

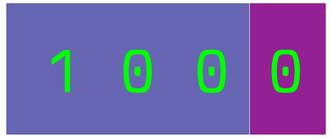


Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

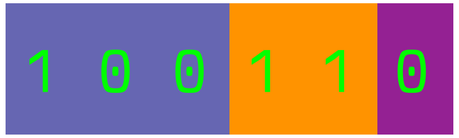


Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

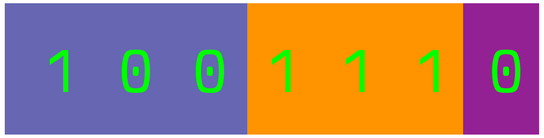
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



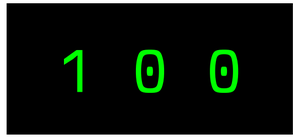
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



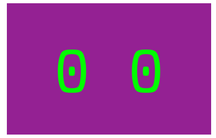
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

The first piece is just the empty string! This is perfectly fine.

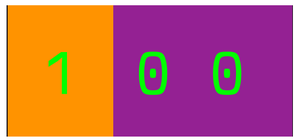
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w$ contains 00 as a substring$\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



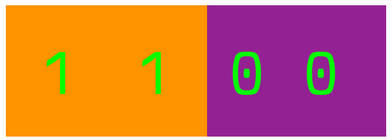
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



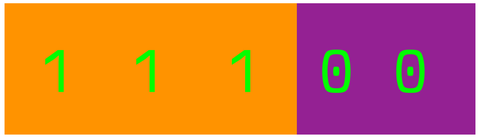
Let $\Sigma = \{0, 1\}$
and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

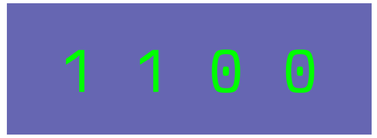
```
1 1 0 0 0 0 1
```

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".



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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

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

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

Let $\Sigma = \{0, 1\}$

and $L = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$

Let $\Sigma = \{0, 1\}$
and $L = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

Let $\Sigma = \{0, 1\}$
and $L = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$

Any string of length three or greater can be split into three pieces, the second of which can be "pumped".

*The Pumping Lemma holds for **finite** languages because the pumping length can be longer than the longest string!*

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$. Then $|s| = 2p \geq p$ and $s \in B$, so we can break this string into $s = xyz$, where $|xy| \leq p$ and $y \neq \varepsilon$, and for any $i \in \mathbb{N}_0$, the string $xy^iz \in B$.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$. Then $|s| = 2p \geq p$ and $s \in B$, so we can break this string into $s = xyz$, where $|xy| \leq p$ and $y \neq \varepsilon$, and for any $i \in \mathbb{N}_0$, the string $xy^iz \in B$.

Because $|xy| \leq p$ and $|y| > 0$, the string y has to consist only of as.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$. Then $|s| = 2p \geq p$ and $s \in B$, so we can break this string into $s = xyz$, where $|xy| \leq p$ and $y \neq \varepsilon$, and for any $i \in \mathbb{N}_0$, the string $xy^iz \in B$.

Because $|xy| \leq p$ and $|y| > 0$, the string y has to consist only of as.

So, no matter what segment of the string xy covers, pumping to the string xy^2z adds to the number of as, hence there are more as than bs.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$. Then $|s| = 2p \geq p$ and $s \in B$, so we can break this string into $s = xyz$, where $|xy| \leq p$ and $y \neq \varepsilon$, and for any $i \in \mathbb{N}_0$, the string $xy^iz \in B$.

Because $|xy| \leq p$ and $|y| > 0$, the string y has to consist only of as.

So, no matter what segment of the string xy covers, pumping to the string xy^2z adds to the number of as, hence there are more as than bs.

There is no way to segment s into xyz that can't be pumped to produce a string that isn't in the language.

THEOREM $B = \{a^n b^n \mid n \in \mathbb{N}_0\}$ is not regular.

PROOF By contradiction; assume B is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Consider the string $s = a^p b^p$. Then $|s| = 2p \geq p$ and $s \in B$, so we can break this string into $s = xyz$, where $|xy| \leq p$ and $y \neq \varepsilon$, and for any $i \in \mathbb{N}_0$, the string $xy^iz \in B$.

Because $|xy| \leq p$ and $|y| > 0$, the string y has to consist only of as.

So, no matter what segment of the string xy covers, pumping to the string xy^2z adds to the number of as, hence there are more as than bs.

There is no way to segment s into xyz that can't be pumped to produce a string that isn't in the language.

Contradiction! Therefore, B is not regular. ■

Nonregular languages

The Pumping Lemma describes a property common to all regular languages.

Any language *L* that does not have this property *cannot be regular*.

The Pumping Lemma game

You can think of a Pumping Lemma proof as a game between *you* and an *adversary*.

You win by finding a contradiction of the Pumping Lemma for the given language L.

The adversary wins if they can make a choice for which the Pumping Lemma holds for L.

The game goes as follows:

The adversary chooses a pumping length p.

You choose a string s with $|s| \geq p$ and $s \in L$.

The adversary break it into x, y, and z such that $|xy| \leq p$ and $y \neq \varepsilon$.

You choose an i such that $xy^iz \notin L$. (If you can't, you lose!)

Gameplay Magazine described the rules as "punishingly intricate".

The Pumping Lemma game

The Pumping Lemma game

Adversary	*You*
1 Maliciously choose pumping length $p > 0$.	

The Pumping Lemma game

Adversary	*You*
1 Maliciously choose pumping length $p > 0$.	**2** Cleverly choose a string $s \in L$ such that $\|s\| \geq p$.

The Pumping Lemma game

Adversary

1 Maliciously choose pumping length $p > 0$.

3 Maliciously split $s = xyz$ with $|xy| \leq p$ and $y \neq \varepsilon$.

You

2 Cleverly choose a string $s \in L$ such that $|s| \geq p$.

The Pumping Lemma game

Adversary

1 Maliciously choose pumping length $p > 0$.

3 Maliciously split $s = xyz$ with $|xy| \leq p$ and $y \neq \varepsilon$.

You

2 Cleverly choose a string $s \in L$ such that $|s| \geq p$.

4 Cleverly choose $i \in \mathbb{N}_0$ such that $xy^iz \notin L$

The Pumping Lemma game

Adversary

1 Maliciously choose pumping length $p > 0$.

3 Maliciously split $s = xyz$ with $|xy| \leq p$ and $y \neq \varepsilon$.

5 You win! 😖

You

2 Cleverly choose a string $s \in L$ such that $|s| \geq p$.

4 Cleverly choose $i \in \mathbb{N}_0$ such that $xy^iz \notin L$

What other languages can we prove are nonregular?

The *equality problem* is defined as follows: Given two strings, x and y, decide whether $x = y$.

Let $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{?}\}$.

We can encode the equality problem as a string of the form $x\texttt{?}y$.

"Is `001` equal to `110`?" would be encoded as `001?110`

"Is `11` equal to `11`?" would be encoded as `11?11`

"Is `110` equal to `110`?" would be encoded as `110?110`

Let $EQUAL = \{w\texttt{?}w \mid w \in \{\texttt{0}, \texttt{1}\}^*\}$

Is *EQUAL* a regular language?

The Pumping Lemma for Regular Languages

For all regular languages L,

there exists a positive integer p such that

for all strings $s \in L$ such that $|s| \geq p$,

there exist strings x, y, and z such that

$s = xyz$ *s can be broken into three pieces,*

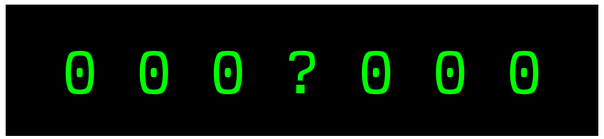
$|xy| \leq p$ *where the first two pieces occur at the start of the string,*

$y \neq \varepsilon$ *the middle part isn't empty, and*

$xy^iz \in L$ *the middle piece can be repeated zero or more times.*

Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

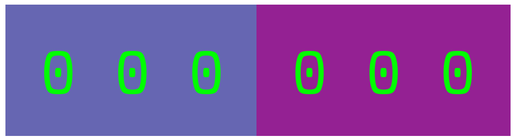


Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

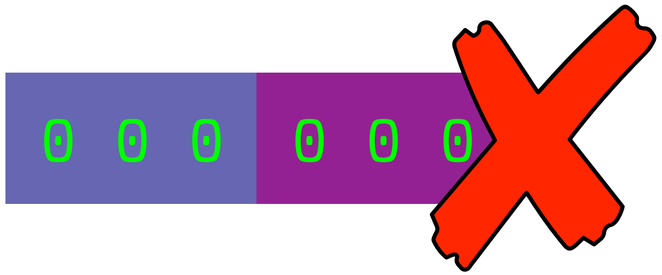
Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$



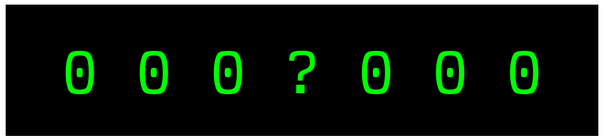
Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$



Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

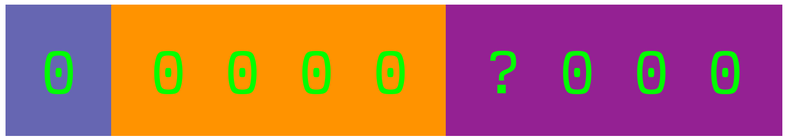


Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

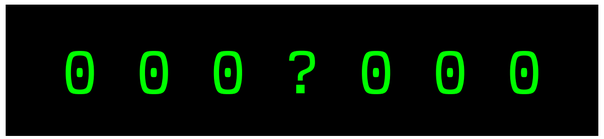


Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$

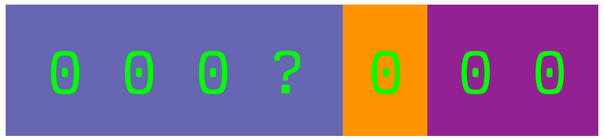
Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$



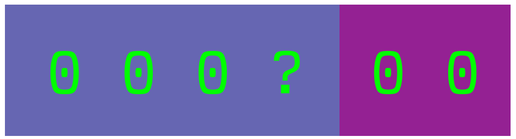
Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$



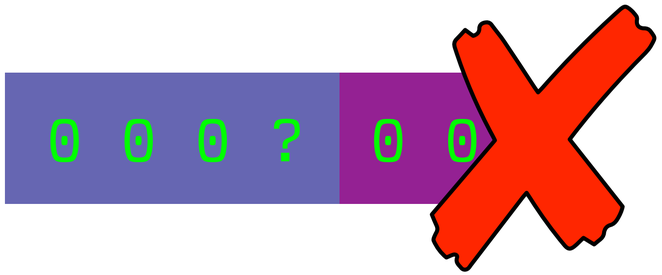
Using the Pumping Lemma

$EQUAL = \{w?w \mid w \in \{0, 1\}^*\}$



Using the Pumping Lemma

EQUAL = $\{w?w \mid w \in \{0, 1\}^*\}$



What's going on?

The Pumping Lemma says that for "sufficiently long" strings, we should be able to pump some part of the string.

We can't pump any part containing the `?` because we can't duplicate it or remove it.

We can't pump just one part of the string because then the strings on opposite sides of the `?` wouldn't match.

Can we *formally* show that *EQUAL* is not regular?

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular.

For all regular languages L,
  there exists a positive integer n such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma.

For all regular languages L,
there exists a positive integer p such that
for all strings $s \in L$ with $|s| \geq p$,
there exist strings x, y, and z such that
$s = xyz$
$|xy| \leq p$
$y \neq \varepsilon$
$xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma.

For all regular languages L,
there exists a positive integer n such that
for all strings $s \in L$ with $|s| \geq p$,
there exist strings x, y, and z such that
$s = xyz$
$|xy| \leq p$
$y \neq \varepsilon$
$xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma.

The hardest part of most Pumping Lemma proofs is choosing a string that we should be able to pump but cannot.

For **all** regular languages L,
  there exists a positive integer n such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$.

For all regular languages L,
  there exists a positive integer n such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$.

For all regular languages L,
there exists a positive integer p such that
for all strings $s \in L$ with $|s| \geq p$,
there exist strings x, y, and z such that
$s = xyz$
$|xy| \leq p$
$y \neq \varepsilon$
$xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$.

For all regular languages L,
there exists a positive integer p such that
for all strings $s \in L$ with $|s| \geq p$,
there exist strings x, y, and z such that
$s = xyz$
$|xy| \leq p$
$y \neq \varepsilon$
$xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$.

For **all** regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$.

At this point, we have some string that we should be able to split into pieces and pump. The rest of the proof shows that no matter what choice we made, the middle can't be pumped.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$. The string y must consist only of `0`s before the `?` or it would violate that $|xy| \leq p$.

For all regular languages L,
there exists a positive integer p such that
for all strings $s \in L$ with $|s| \geq p$,
there exist strings x, y, and z such that
$s = xyz$
$|xy| \leq p$
$y \neq \varepsilon$
$xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$. The string y must consist only of `0`s before the `?` or it would violate that $|xy| \leq p$. Therefore, $xy^0z = \texttt{0}^m\texttt{?}\texttt{0}^p$, where $m < p$, and is not in *EQUAL*.

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

THEOREM *EQUAL* is not regular.

PROOF By contradiction; assume that *EQUAL* is regular. Let p be the pumping length guaranteed by the Pumping Lemma. Let $s = \texttt{0}^p\texttt{?}\texttt{0}^p$. Then $s \in EQUAL$ and $|s| = 2p + 1 \geq p$. Thus by the Pumping Lemma, we can write $s = xyz$ such that $|xy| \leq p$ and $y \neq \varepsilon$ and for any $i \in \mathbb{N}_0$, $xy^iz \in EQUAL$. The string y must consist only of `0`s before the `?` or it would violate that $|xy| \leq p$. Therefore, $xy^0z = \texttt{0}^m\texttt{?}\texttt{0}^p$, where $m < p$, and is not in *EQUAL*. This contradicts the Pumping Lemma, so our assumption was wrong. Thus *EQUAL* is not regular. ■

For all regular languages L,
  there exists a positive integer p such that
    for all strings $s \in L$ with $|s| \geq p$,
      there exist strings x, y, and z such that
        $s = xyz$
        $|xy| \leq p$
        $y \neq \varepsilon$
        $xy^iz \in L$ for all $i \in \mathbb{N}_0$

Critical point

It's necessary to show there is *no segmentation* of the chosen string that won't lead to a contradiction

- This means considering *every possible* mapping of *xy* onto the first *p* symbols in the chosen string.
- We chose our string to make this easy, since every possible segmentation consists of a*s* only.
  - Pumping therefore disrupts the equivalence of the number of a*s* and b*s*.

Critical point

We only need to show that there's *one string* in the language for which the Pumping Lemma doesn't work.

For some strings in *L*, it may work perfectly well!

The Pumping Lemma mascot, the Pumping Llama

by Kimberly Do

Acknowledgments

This lecture incorporates material from:

David Chiang, University of Notre Dame

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*