CMPU 240 · Theory of Computation

Designing Turing Machines

16 April 2026

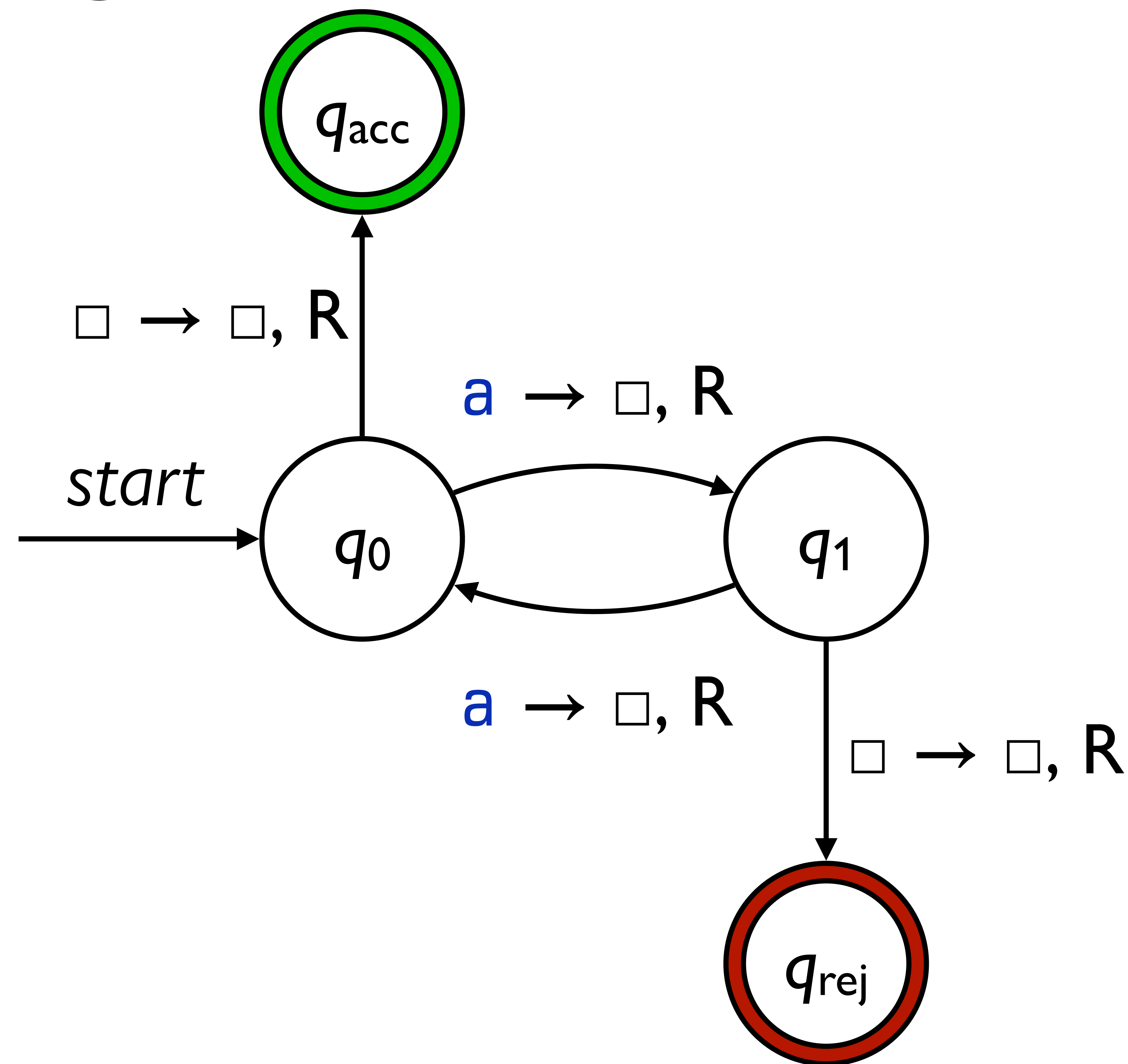
VASSAR COLLEGE A·D·1861

Assignment 8

Out today, back to our usual schedule until the end of the semester

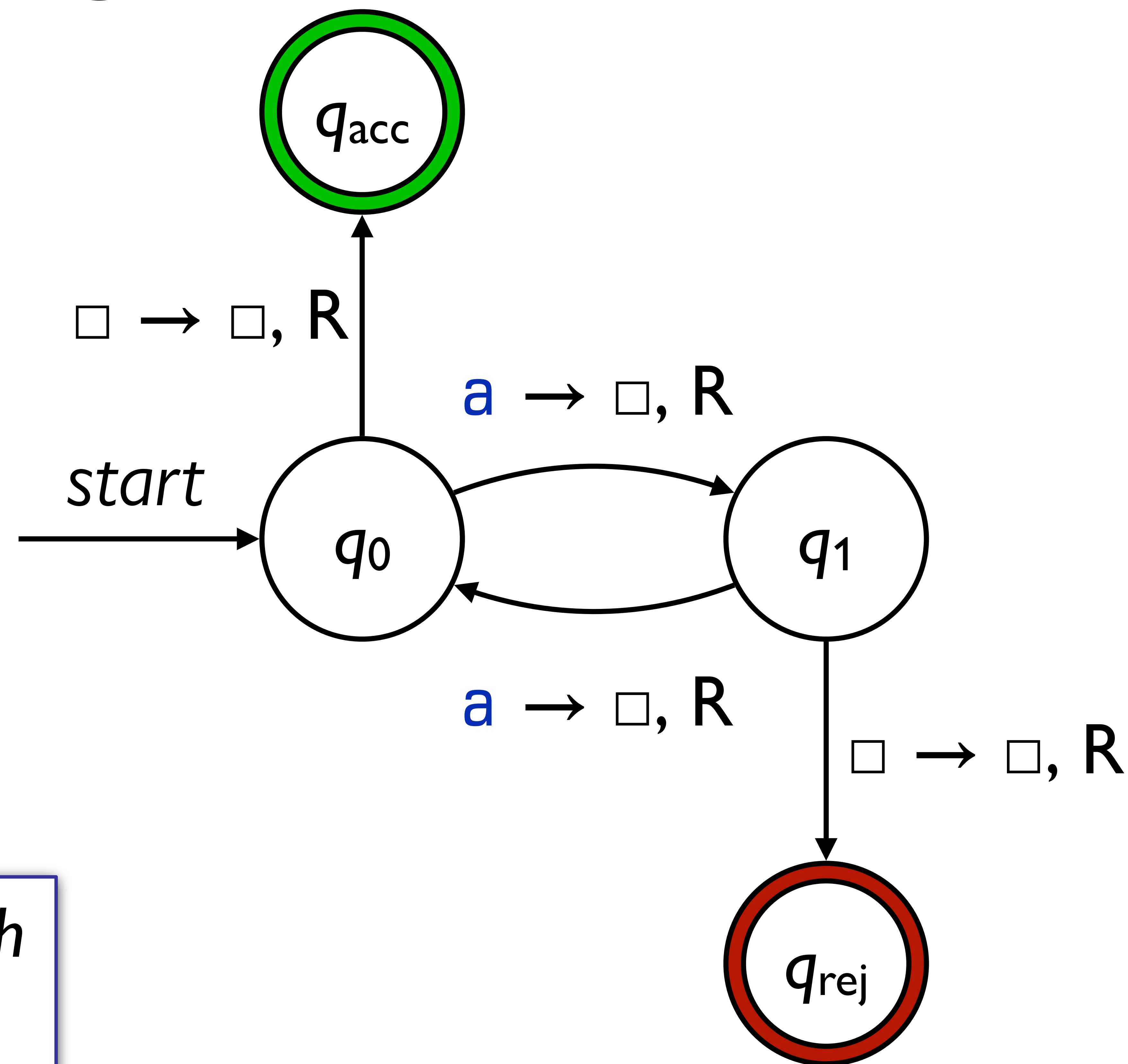
Where are we?

Our first Turing machine



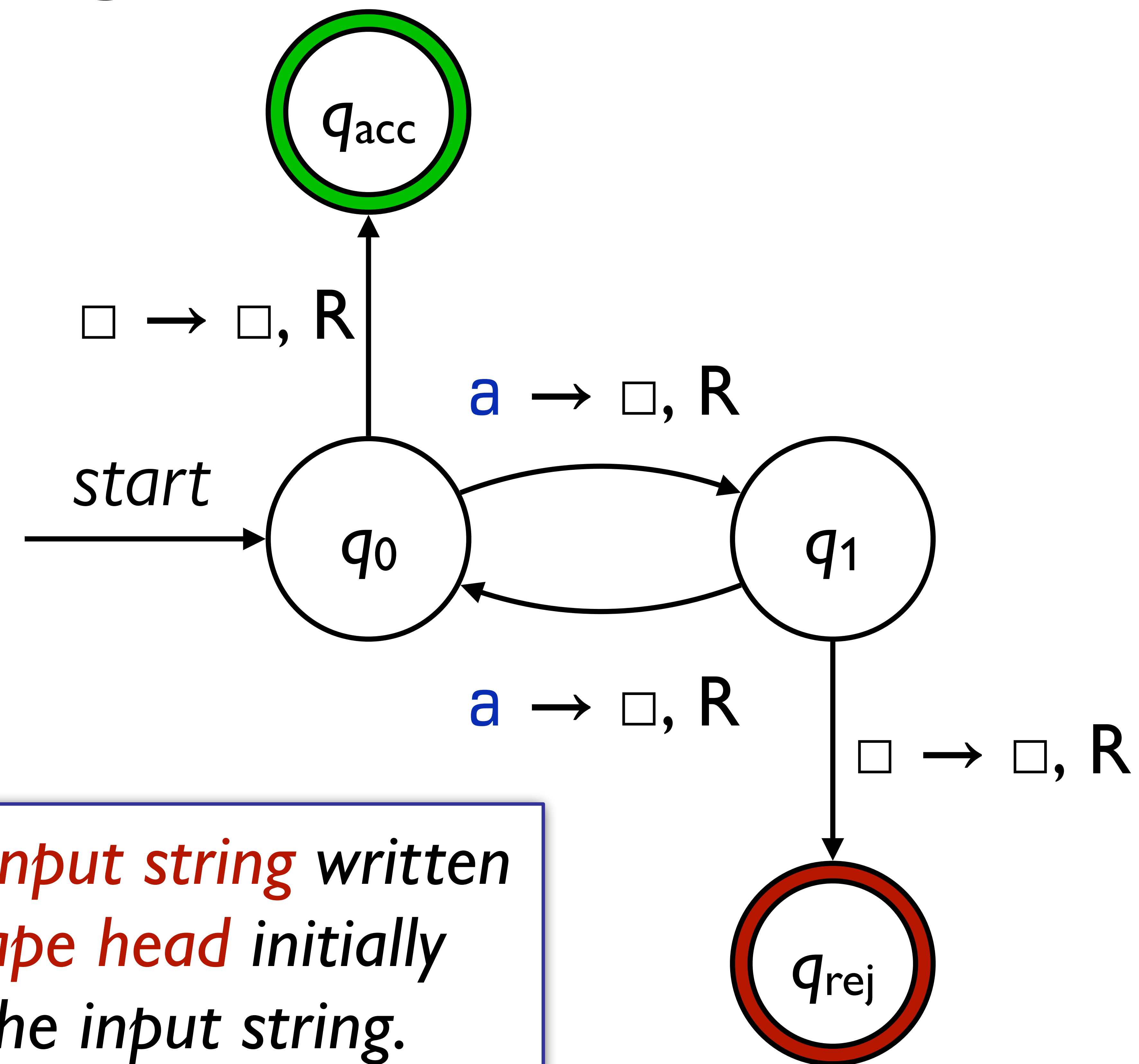
This is the Turing machine's *finite-state control*. It issues commands that drive the operation of the machine.

Our first Turing machine



This is the TM's *infinite tape*. *Each tape cell holds a* *tape symbol*. *Initially all tape symbols are blank.*

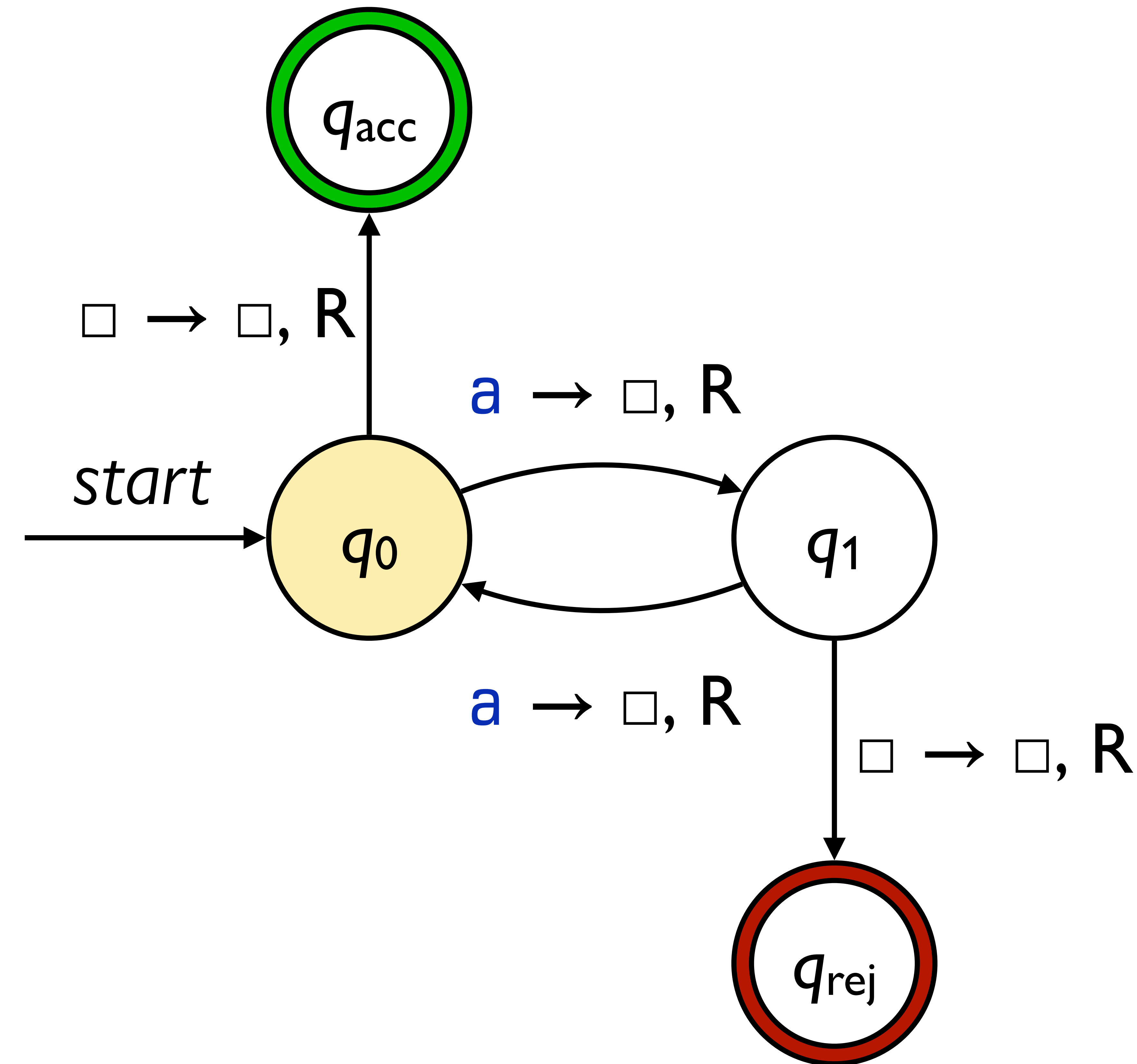
Our first Turing machine



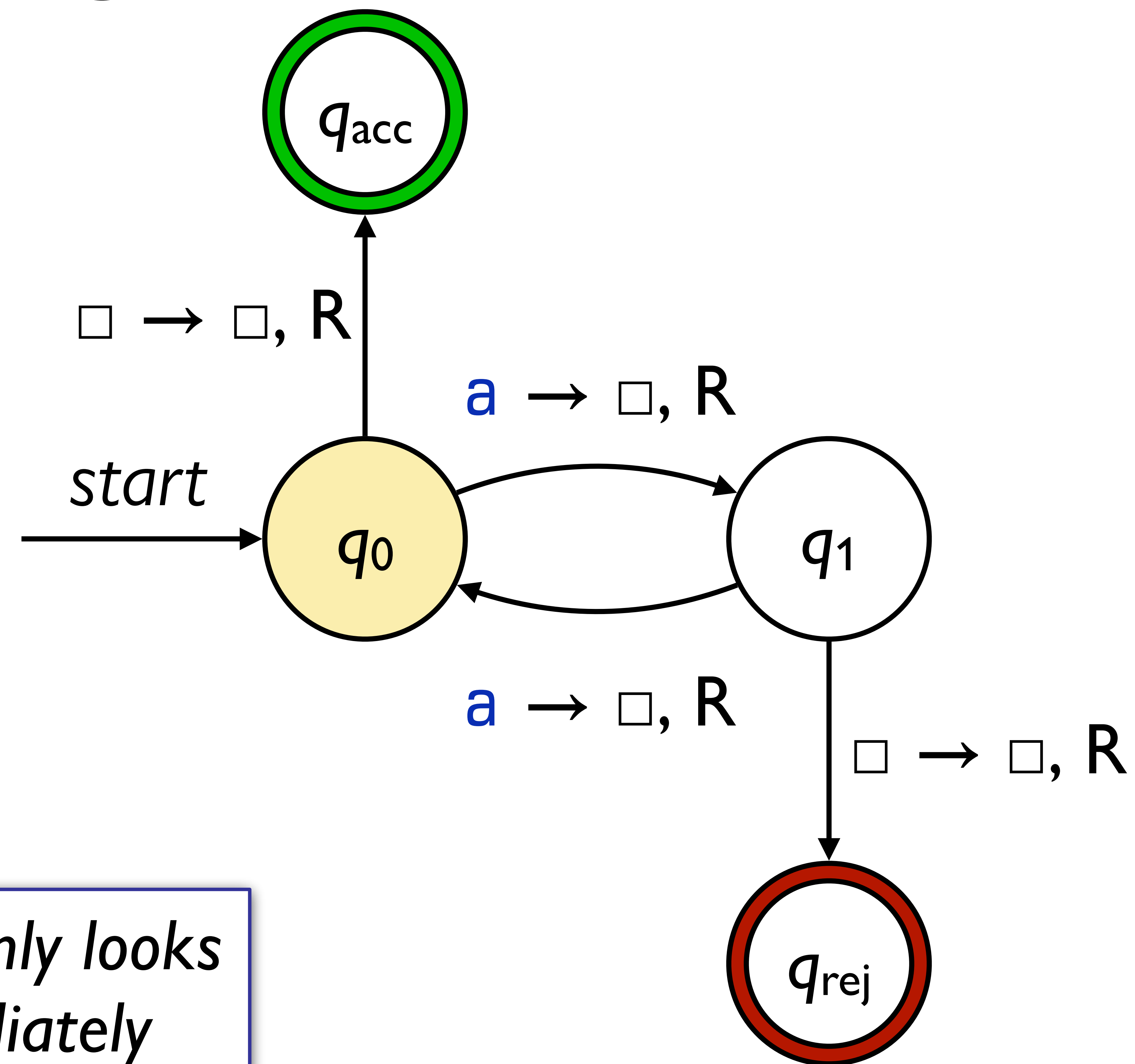
The machine is started with the input string written somewhere on the tape. The tape head initially points to the first symbol of the input string.

Our first Turing machine

Like other automata, TMs begin execution in their start state

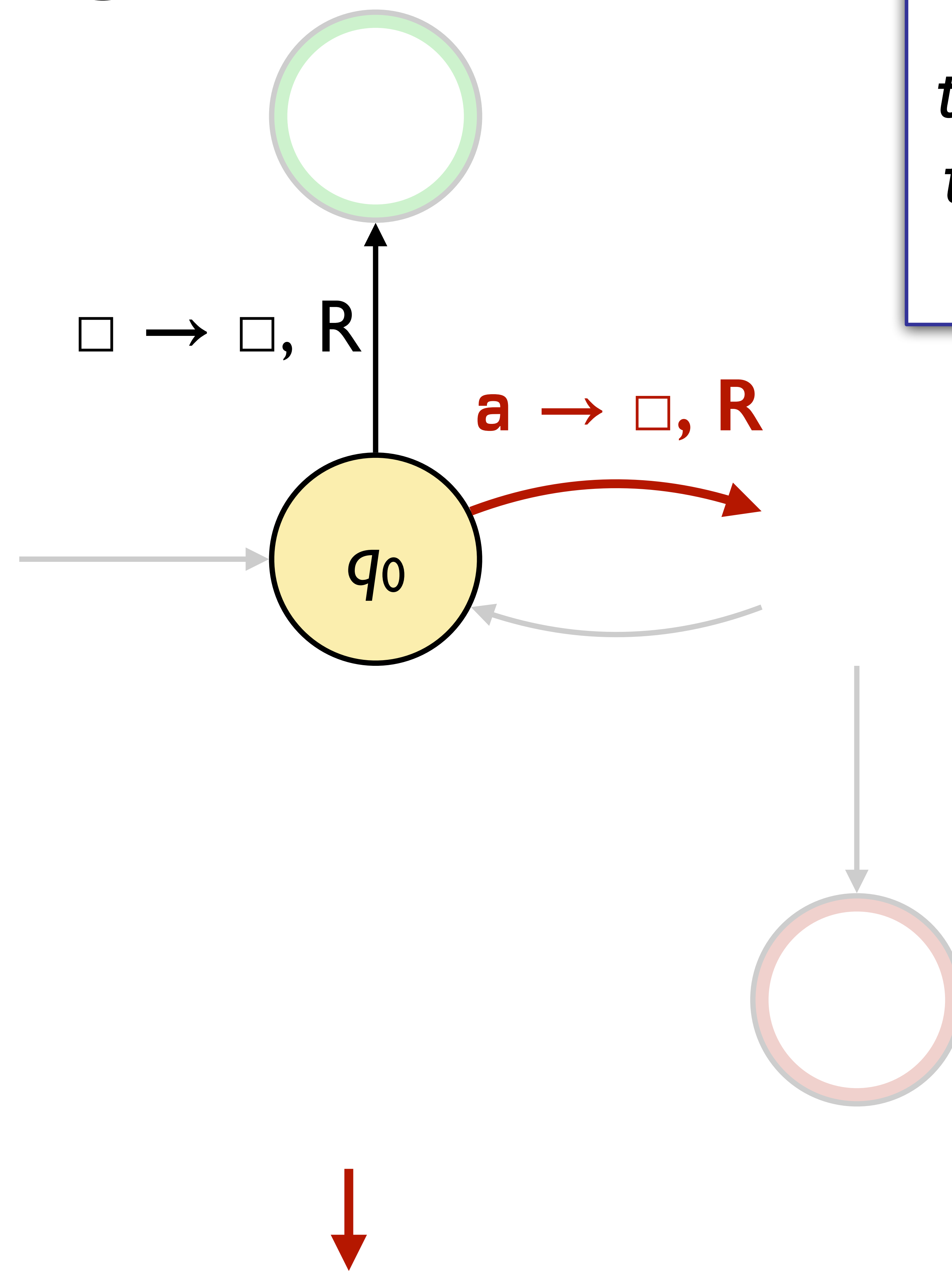


Our first Turing machine



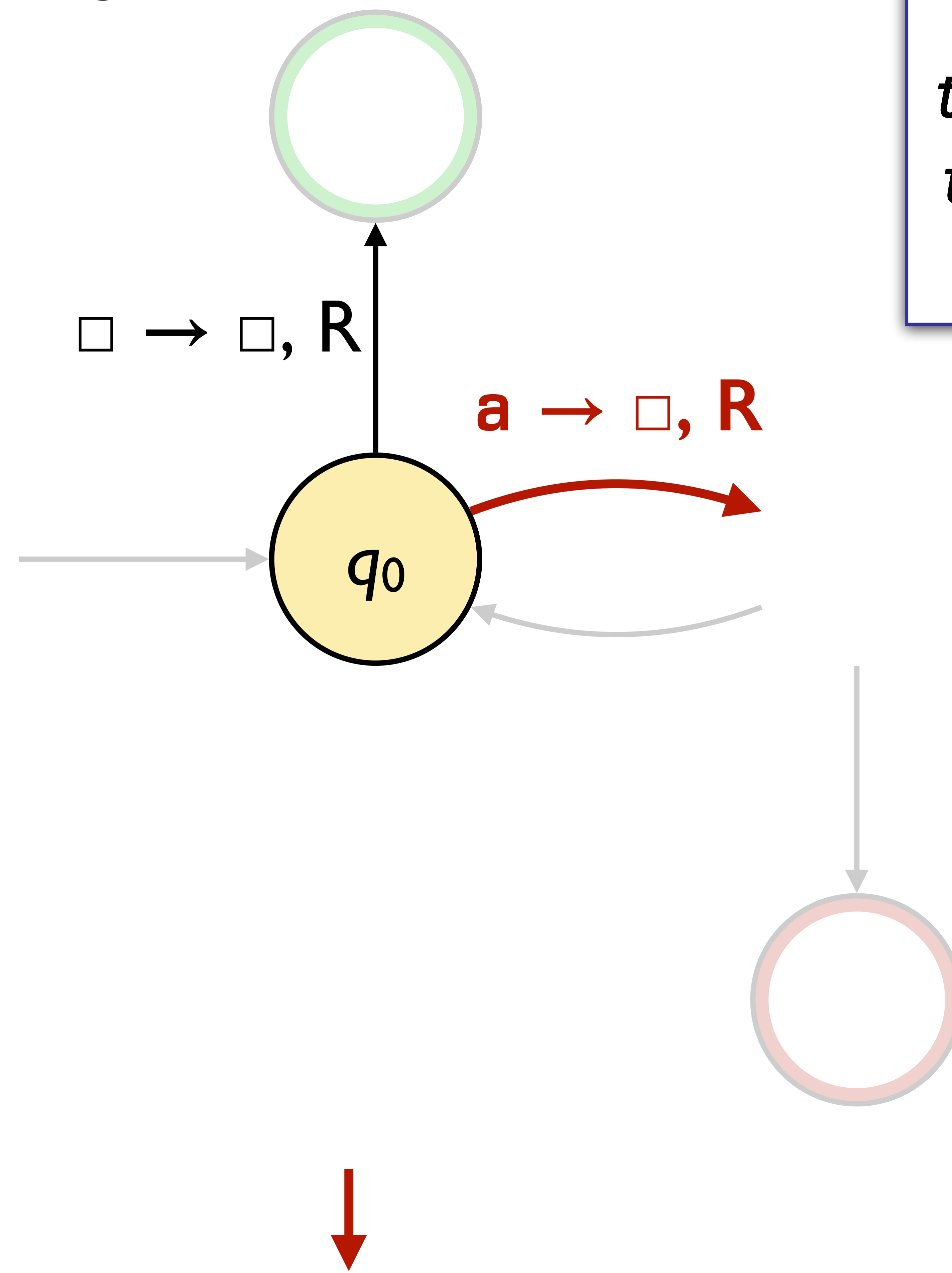
At each step, the TM only looks at the symbol immediately under the tape head.

Our first Turing machine



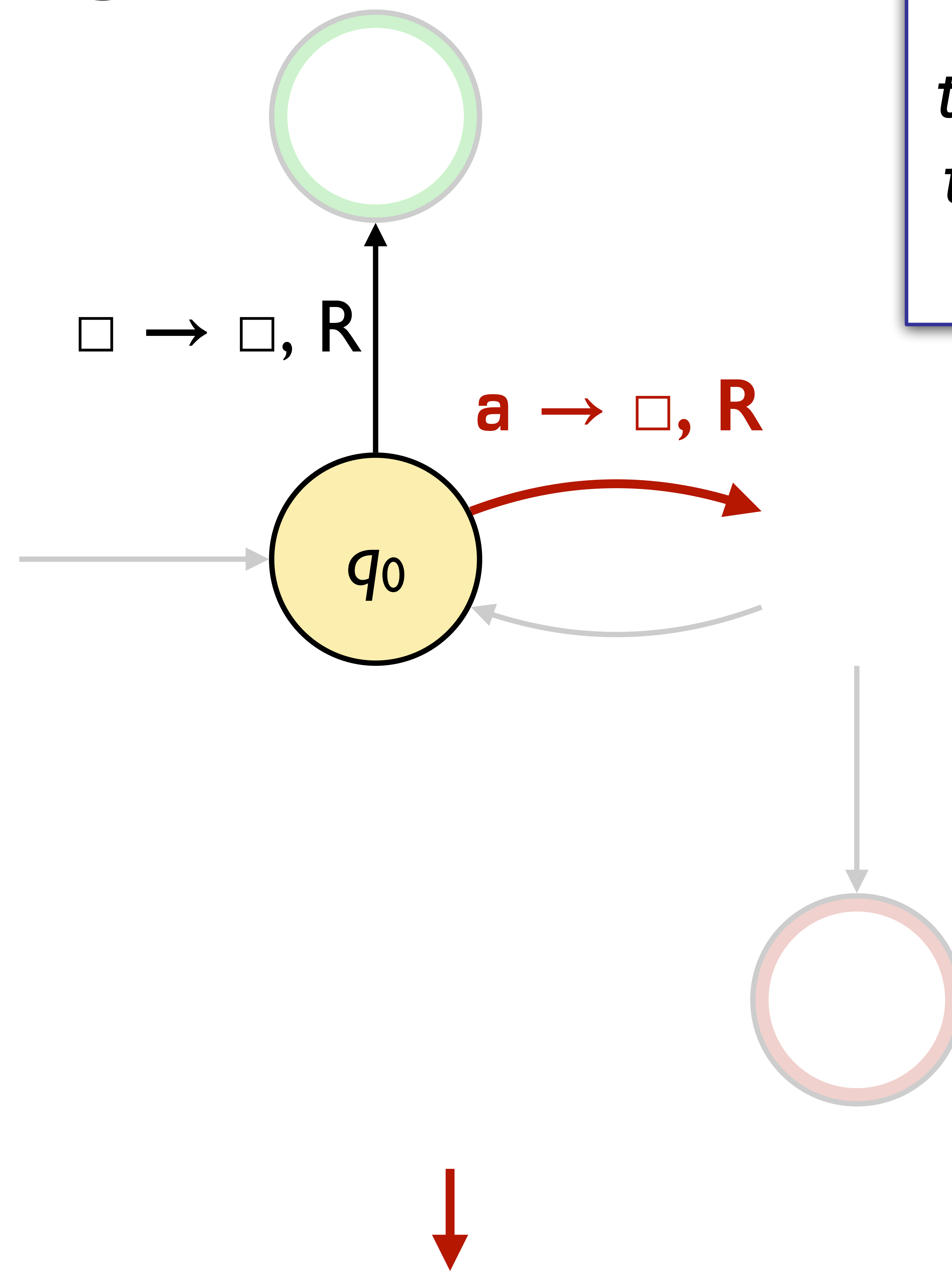
Each transition has the form
⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩
and means "if symbol ⟨**read**⟩ *is under the tape head, replace it with* ⟨**write**⟩ *and move the tape head in* ⟨**direction**⟩ *(left or right)". The □ symbol denotes a blank cell.*

Our first Turing machine



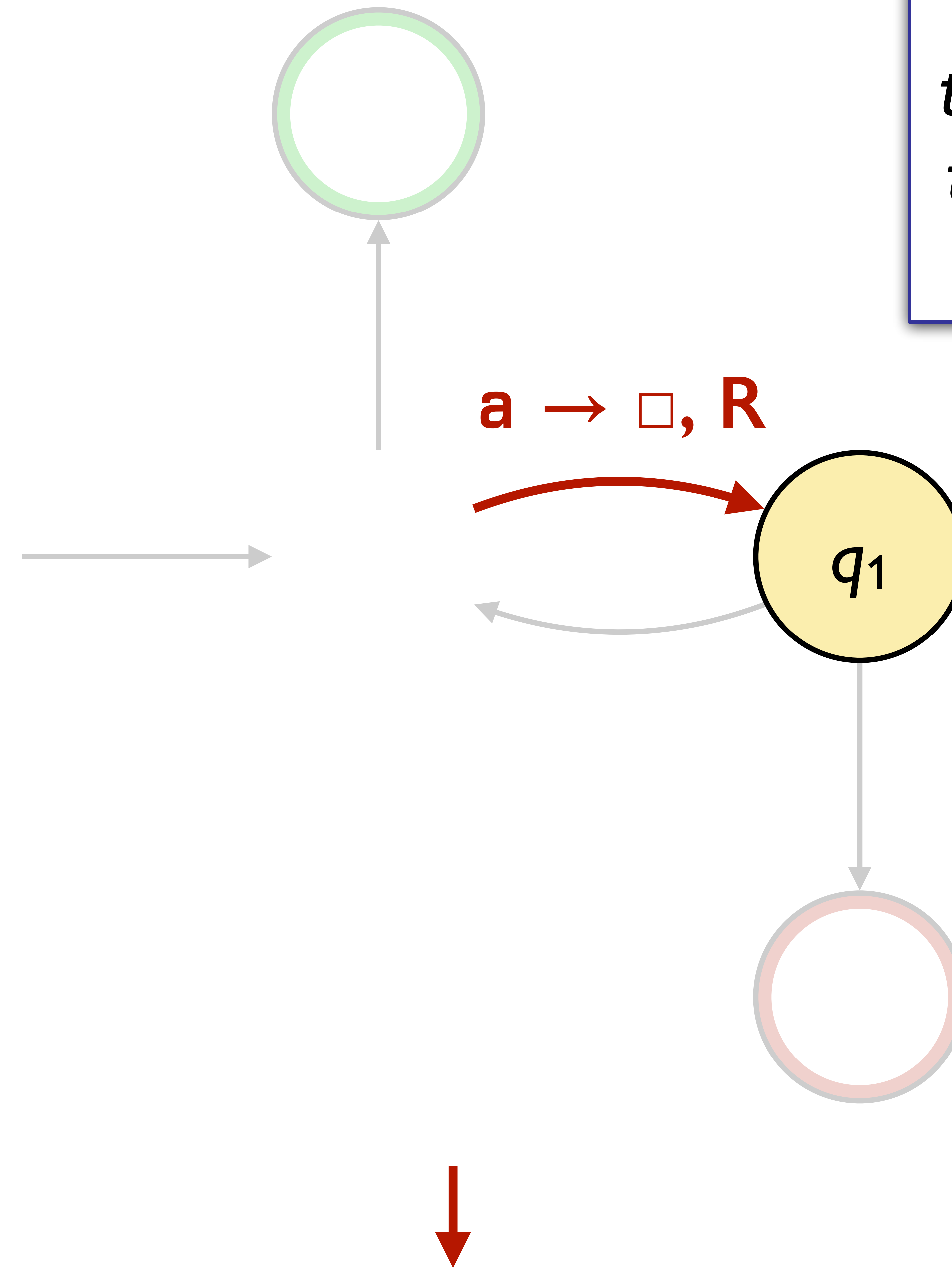
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.

Our first Turing machine



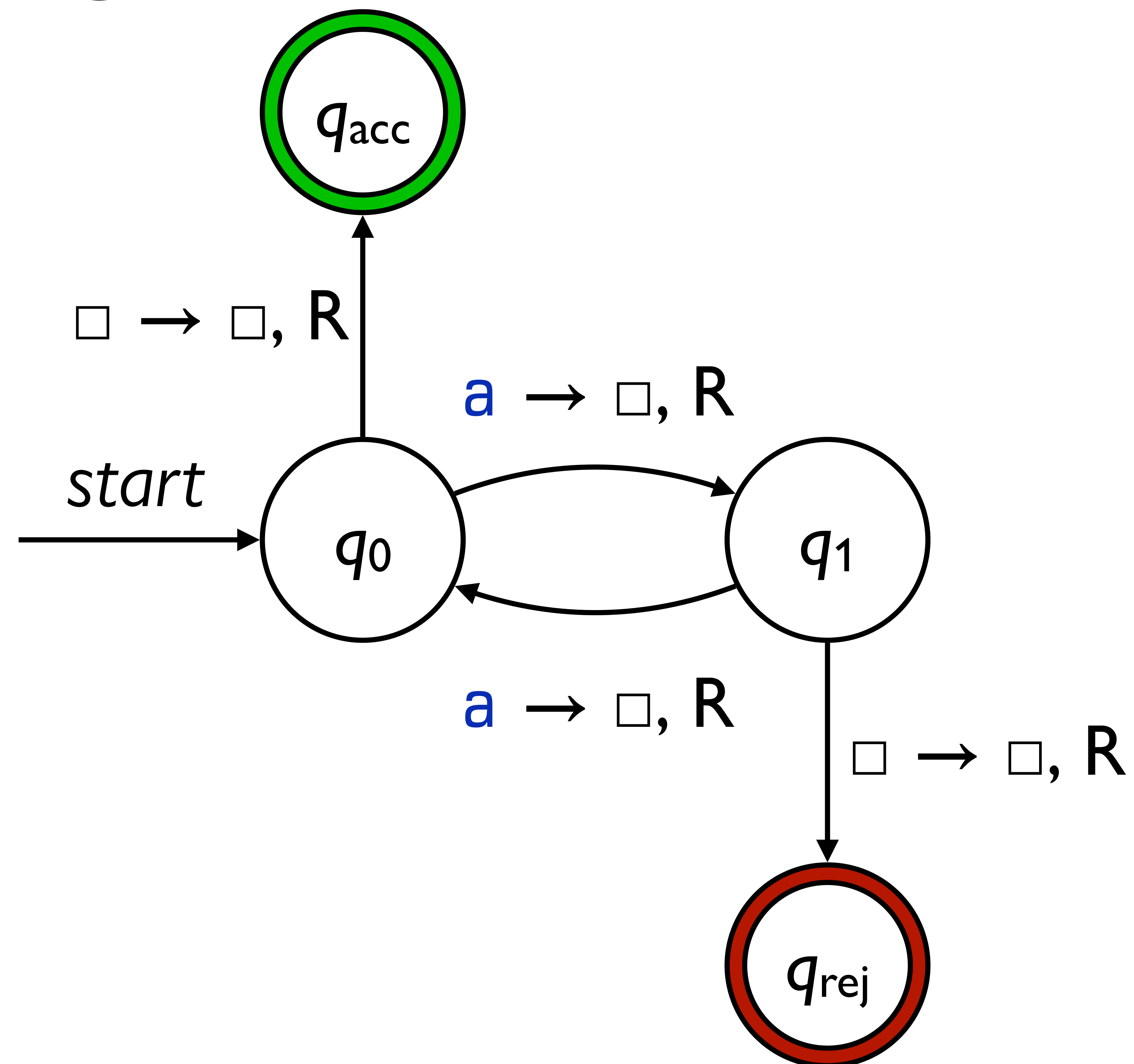
Each transition has the form
⟨read⟩ → **⟨write⟩**, **⟨direction⟩**
and means "if symbol **⟨read⟩** *is under the tape head, replace it with* **⟨write⟩** *and move the tape head in* **⟨direction⟩** *(left or right)". The □ symbol denotes a blank cell.*

Our first Turing machine



Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.

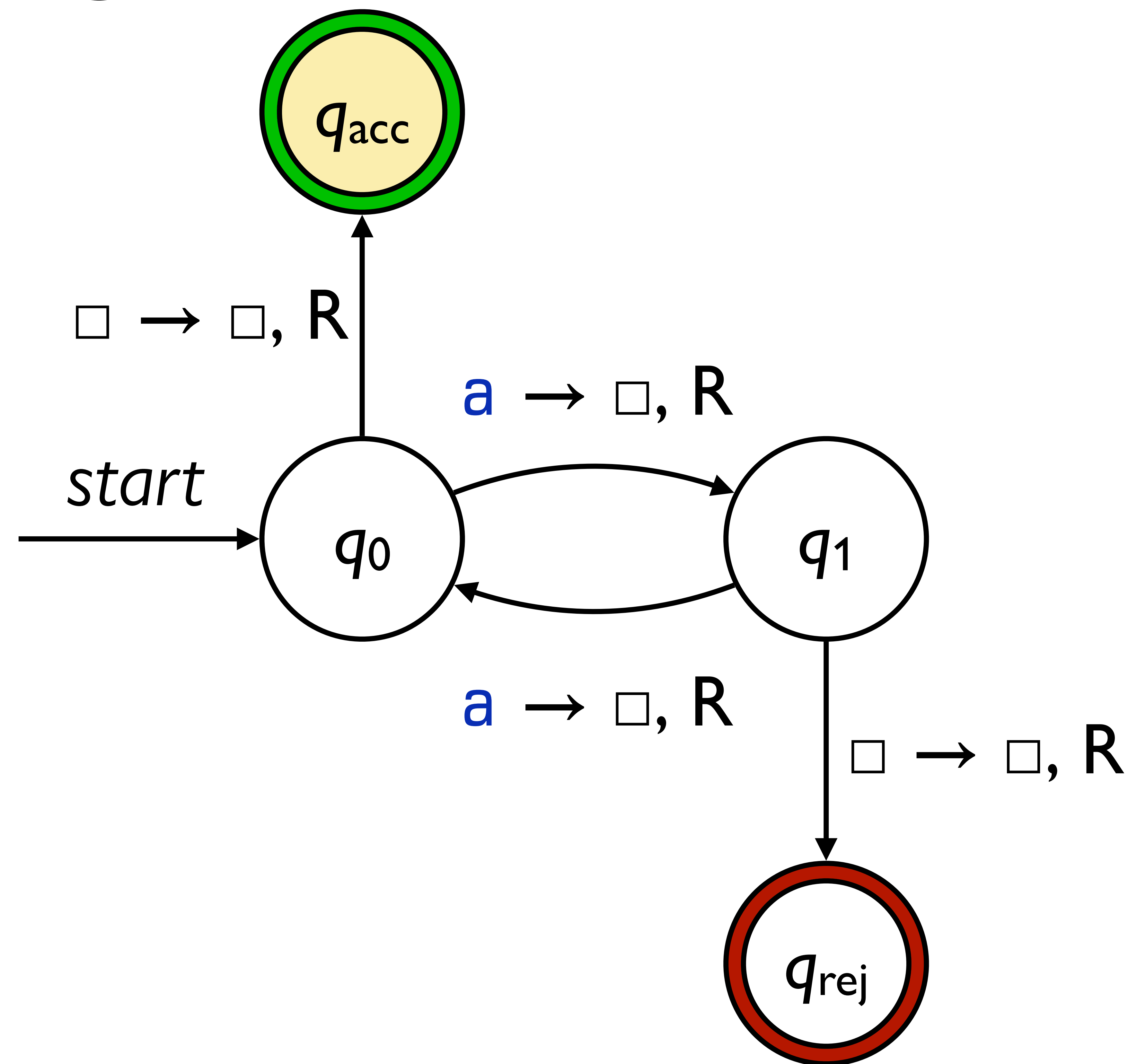
Our first Turing machine



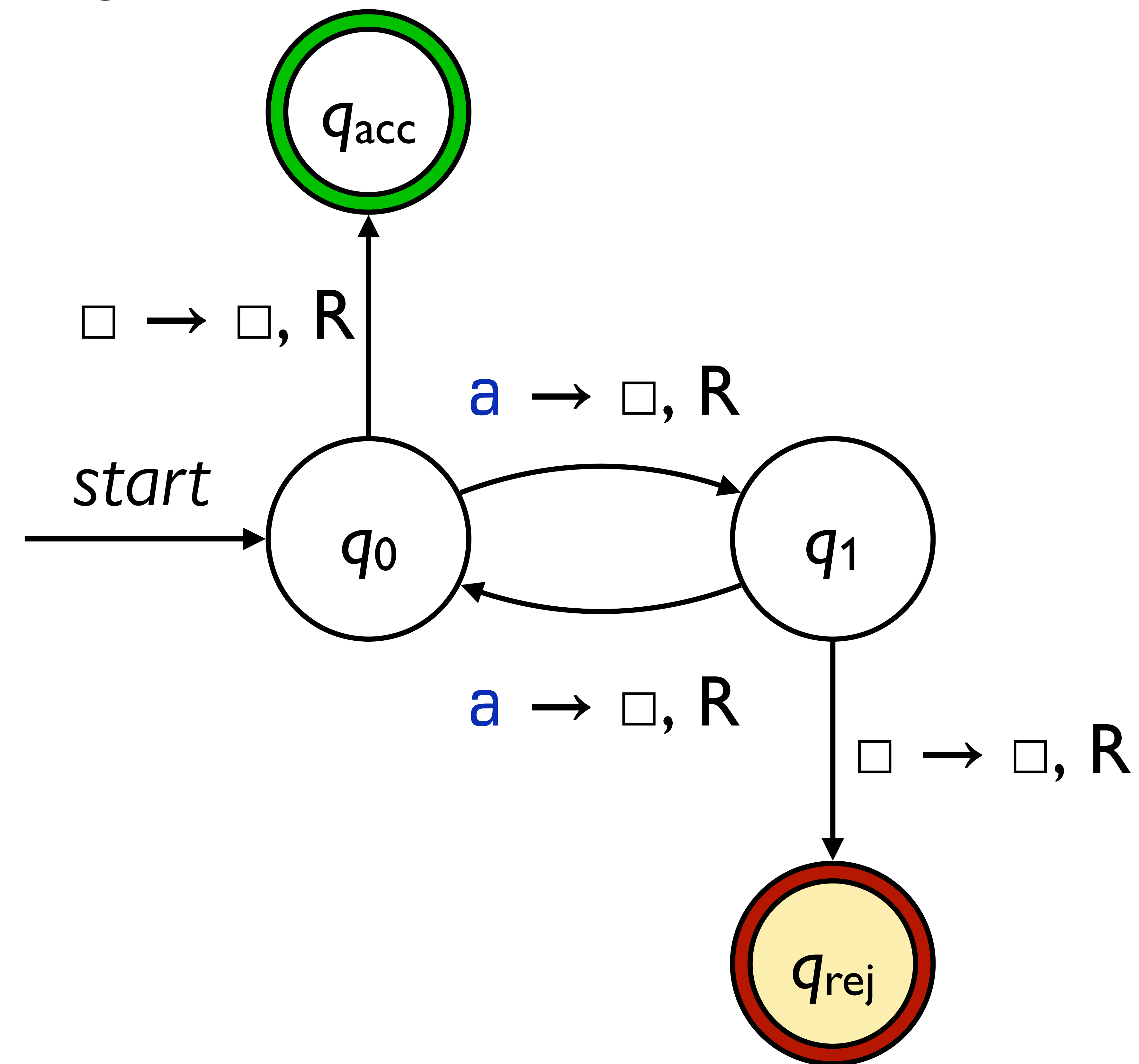
Unlike a DFA or NFA, a TM doesn't stop after reading all the input characters. We keep running until the machine explicitly says to stop.

Our first Turing machine

This special state is an accept state. When a TM enters an accepting state, it immediately stops running and accepts whatever the original input string was.

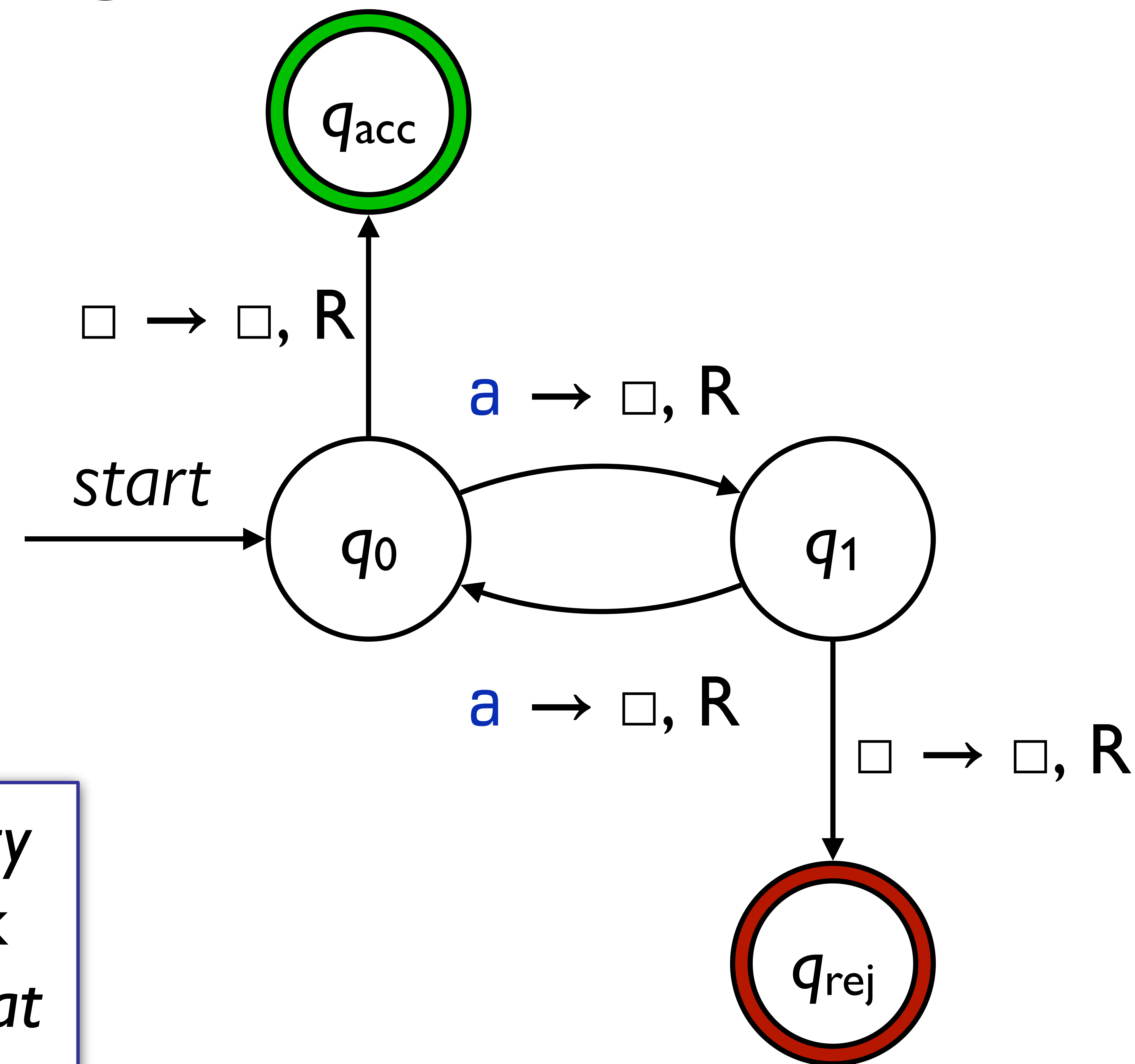


Our first Turing machine



This special state is a reject state. When a TM enters a reject state, it immediately stop running and rejects whatever the original input string was.

Our first Turing machine



If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.

All input strings are written in Σ, the *input alphabet*.

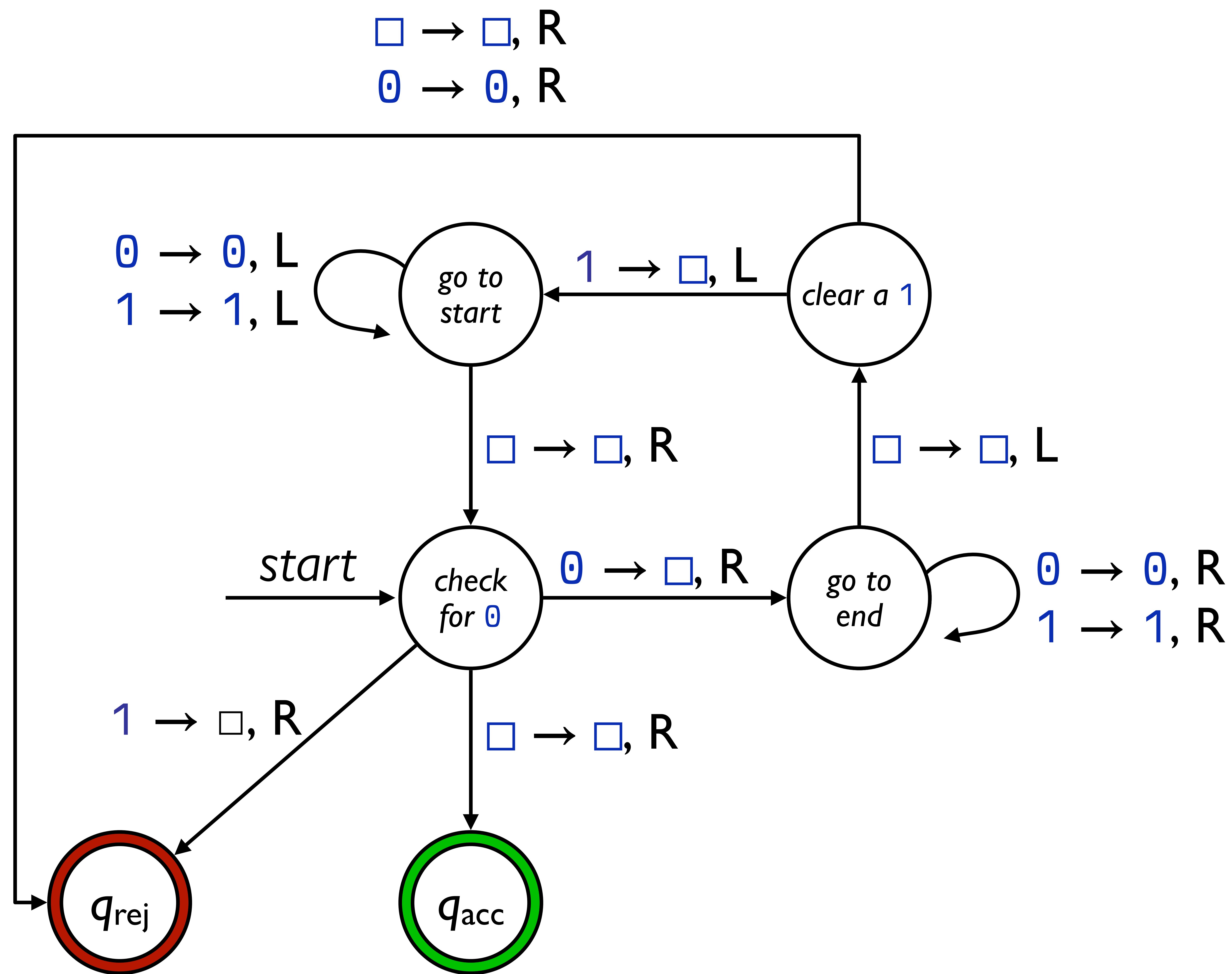
The *tape alphabet* Γ contains all symbols that can be written onto the tape.

The tape alphabet always contains the blank symbol $\square$, which is guaranteed not to be in the input alphabet.

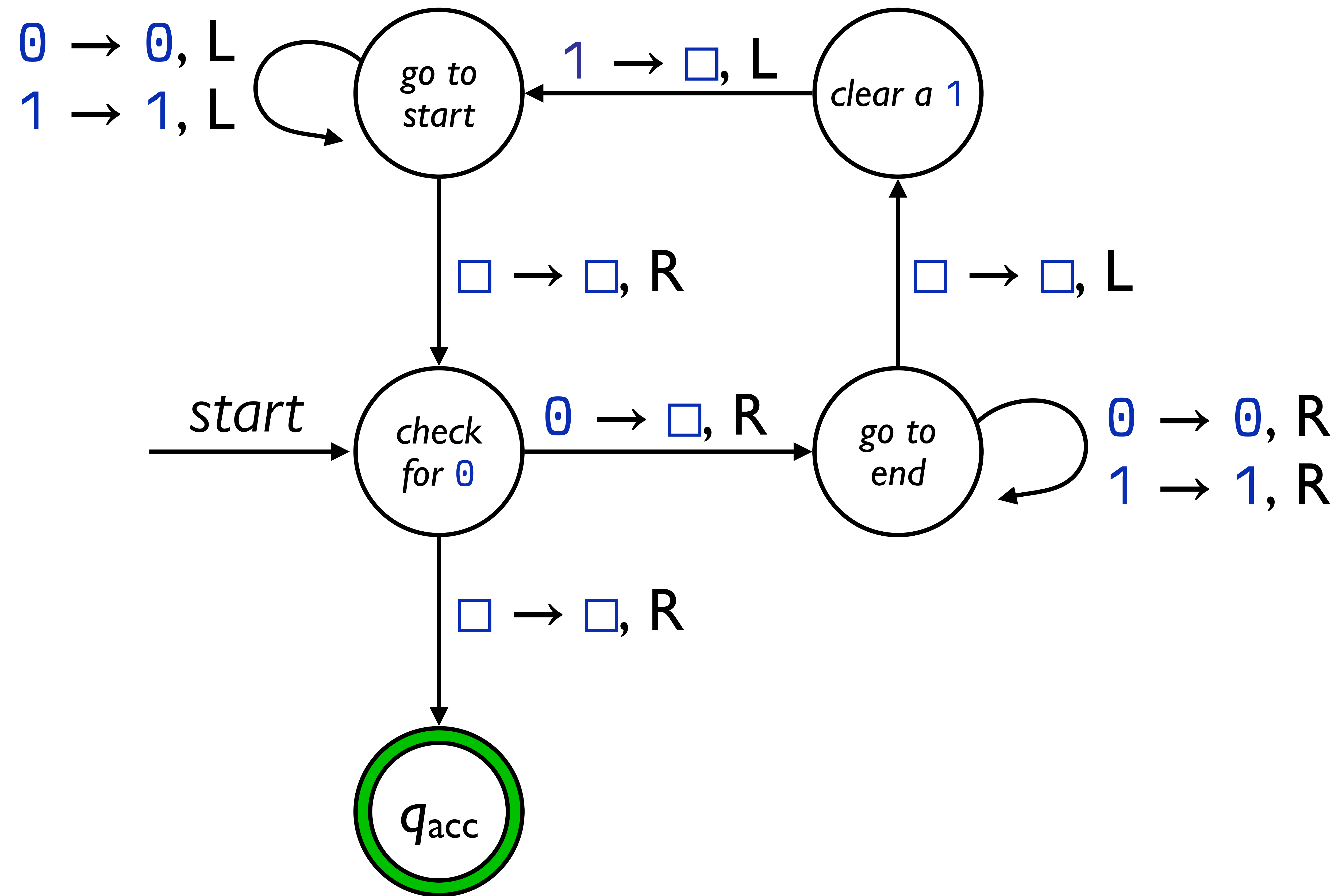
At startup, the Turing machine begins with an infinite tape of □ symbols with the input written at some location.

The tape head is positioned at the start of the input.

$$\{0^n 1^n \mid n \in \mathbb{N}_0\}$$

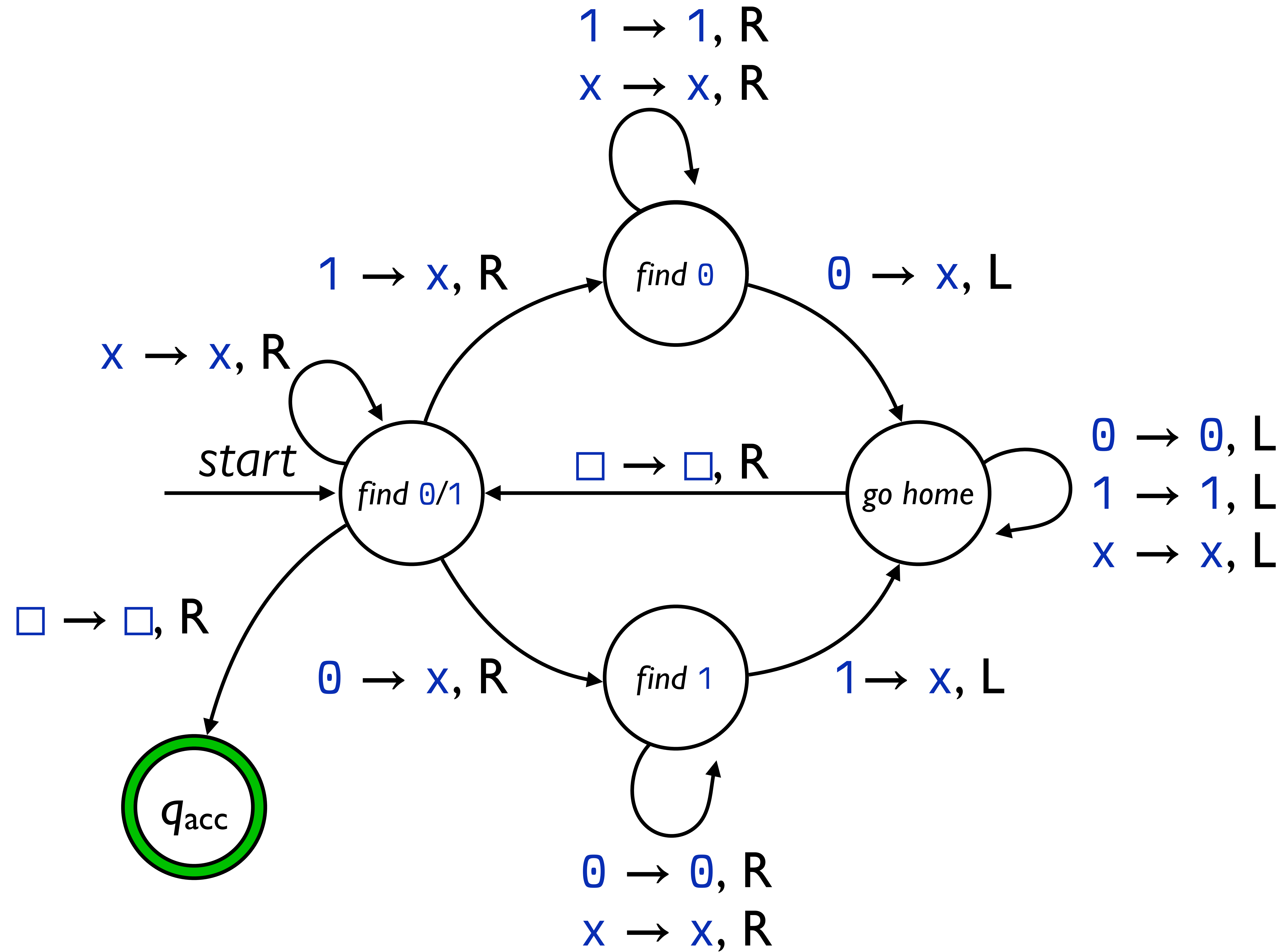


$\{0^n 1^n \mid n \in \mathbb{N}_0\}$

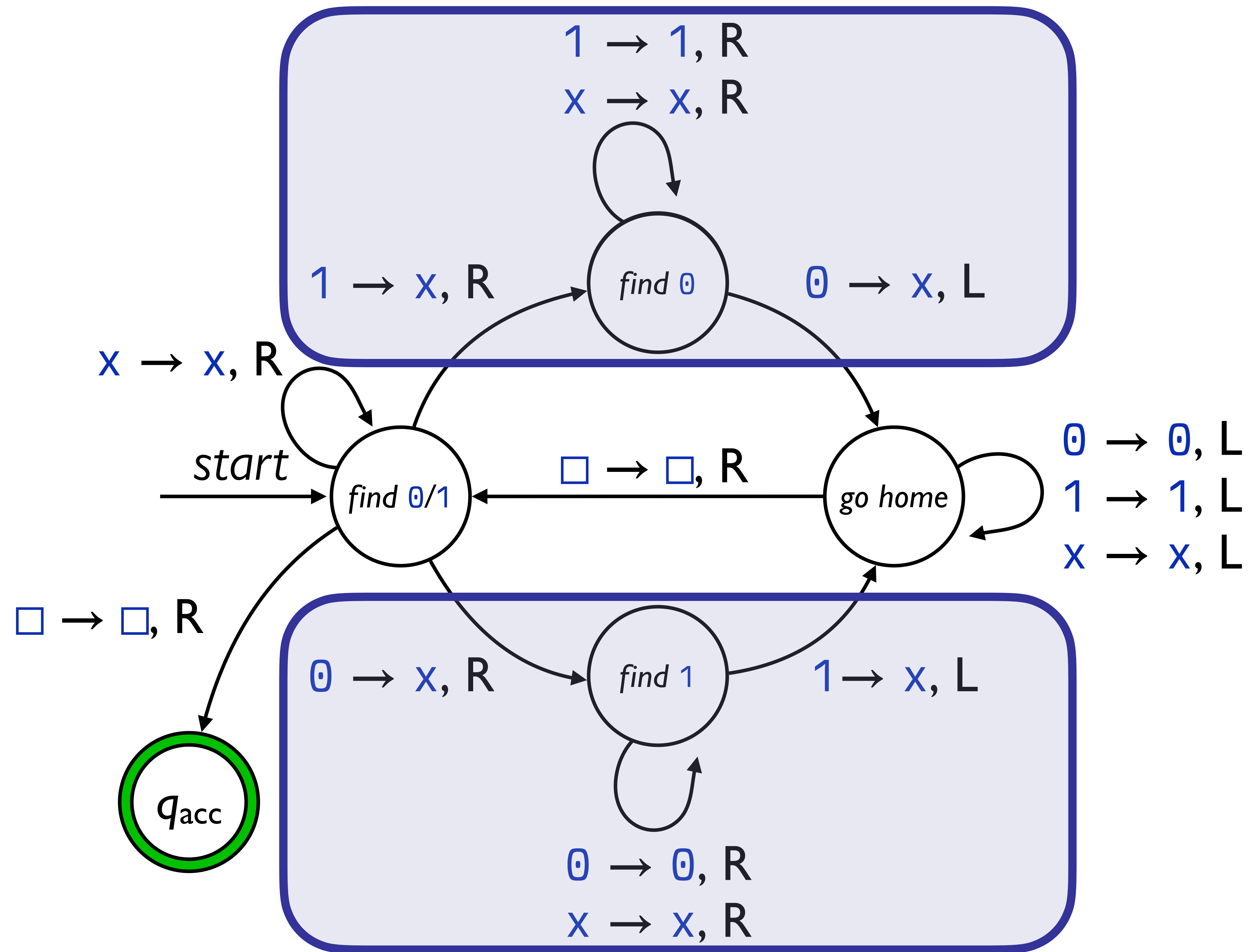


$\{0^n 1^n \mid n \in \mathbb{N}_0\}$

$$\{w \in \Sigma^* \mid n_0(w) = n_1(\mathrm{w})\}$$



$$\{w \in \Sigma^* \mid n_0(w) = n_1(w)\}$$



Two "cases" the TM handles!

$$\{w \in \Sigma^* \mid n_0(w) = n_1(w)\}$$

A Turing machine for a trickier language

Consider the following language over $\Sigma = \{\texttt{0}, \texttt{1}\}$:

$$L = \{\texttt{0}^n\texttt{1}^m \mid n, m \in \mathbb{N}_0 \text{ and } m \text{ is a multiple of } n\}$$

Is this language regular? Context-free?

Consider the following language over $\Sigma = \{\texttt{0}, \texttt{1}\}$:

$$L = \{\texttt{0}^n\texttt{1}^m \mid n, m \in \mathbb{N}_0 \text{ and } m \text{ is a multiple of } n\}$$

Is this language regular? Context-free? *Non-context-free!*

How could we design a TM for L?

An observation

We can recursively describe when one number m is a multiple of n:

If $m = 0$, then m is a multiple of n.

Otherwise, if $n = 0$, then m is not a multiple of n.

Otherwise, m is a multiple of n iff $m \geq n$ and $m - n$ is a multiple of n.

Idea: Repeatedly subtract n from m until m becomes zero (good!) or drops below zero (bad!)

One solution

										0	0	1	1	1	1	1	1															

One solution

										x	0	1	1	1	1	1	1															

One solution

										x	0	1	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1	1															

One solution

										x	0	1	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1	1															

One solution

										x	0	1	1	1	1	1	1															

One solution

										x	0	1	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1																	

One solution

										x	0	1	1	1	1	1																	

One solution

										x	0	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1																

One solution

										x	0	1	1	1	1	1																	

One solution

x	0	1	1	1	1	1

One solution

										x	0	1	1	1	1	1																	

One solution

										x	0	1	1	1	1	1																	

One solution

										x	0	1	1	1	1	1																	

One solution

										x	0	1	1	1	1	1																	

One solution

										x	x	1	1	1	1	1																	

One solution

										x	x	1	1	1	1	1																

One solution

											x	x	1	1	1	1	1																

One solution

											x	x	1	1	1	1	1																

One solution

										x	x	1	1	1	1	1																	

One solution

										x	x	1	1	1	1	1																	

One solution

										x	x	1	1	1	1	1																

One solution

										x	x	1	1	1	1																		

One solution

x	x	1	1	1	1

One solution

										x	x	1	1	1	1																		

One solution

										x	x	1	1	1	1																		

One solution

										x	x	1	1	1	1																		

One solution

										x	x	1	1	1	1																		

One solution

										x	x	1	1	1	1																		

One solution

										x	x	1	1	1	1																		

One solution

x	x	1	1	1	1

One solution

										x	x	1	1	1	1																	

One solution

										x	x	1	1	1	1																	

One solution

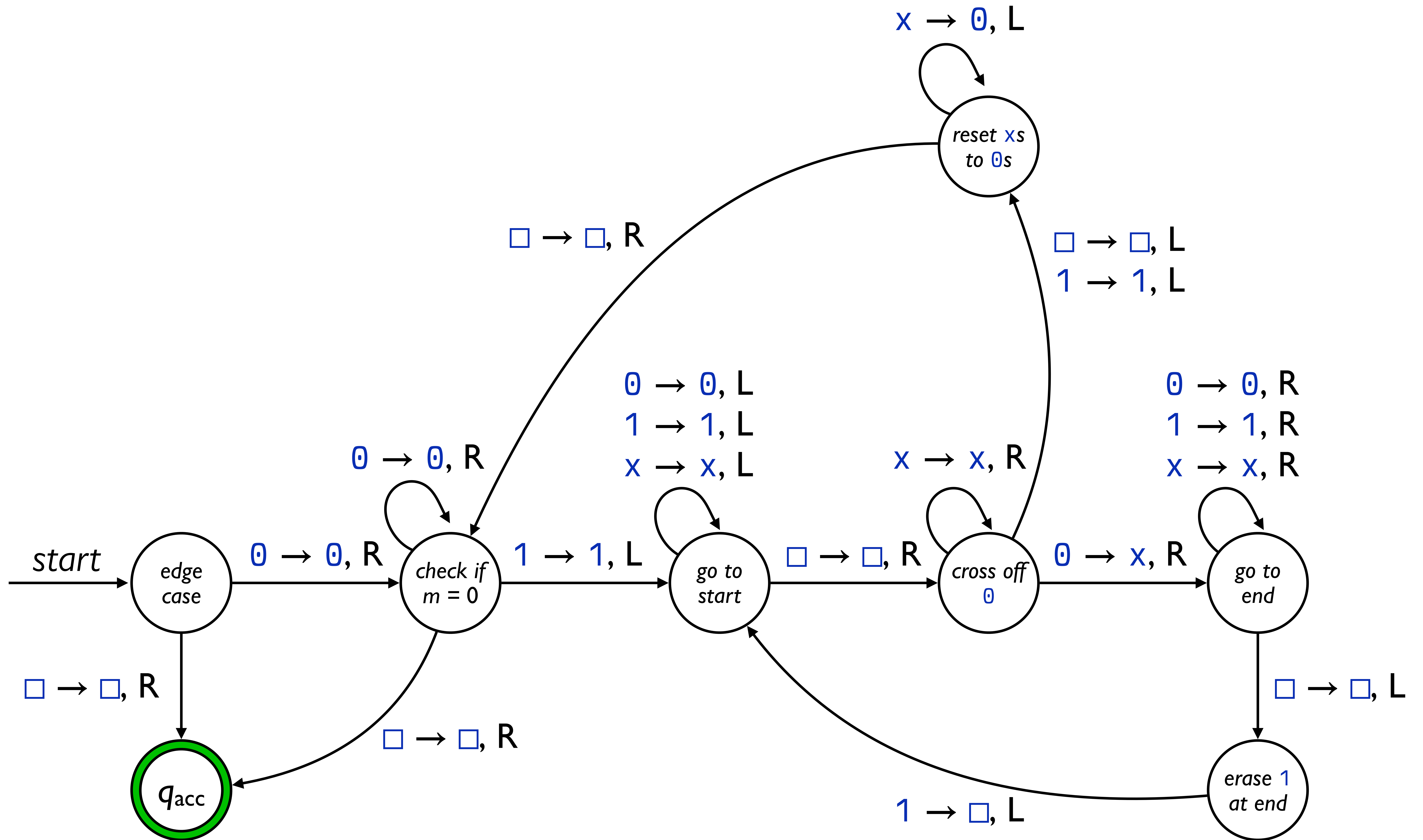
										x	0	1	1	1	1																			

One solution

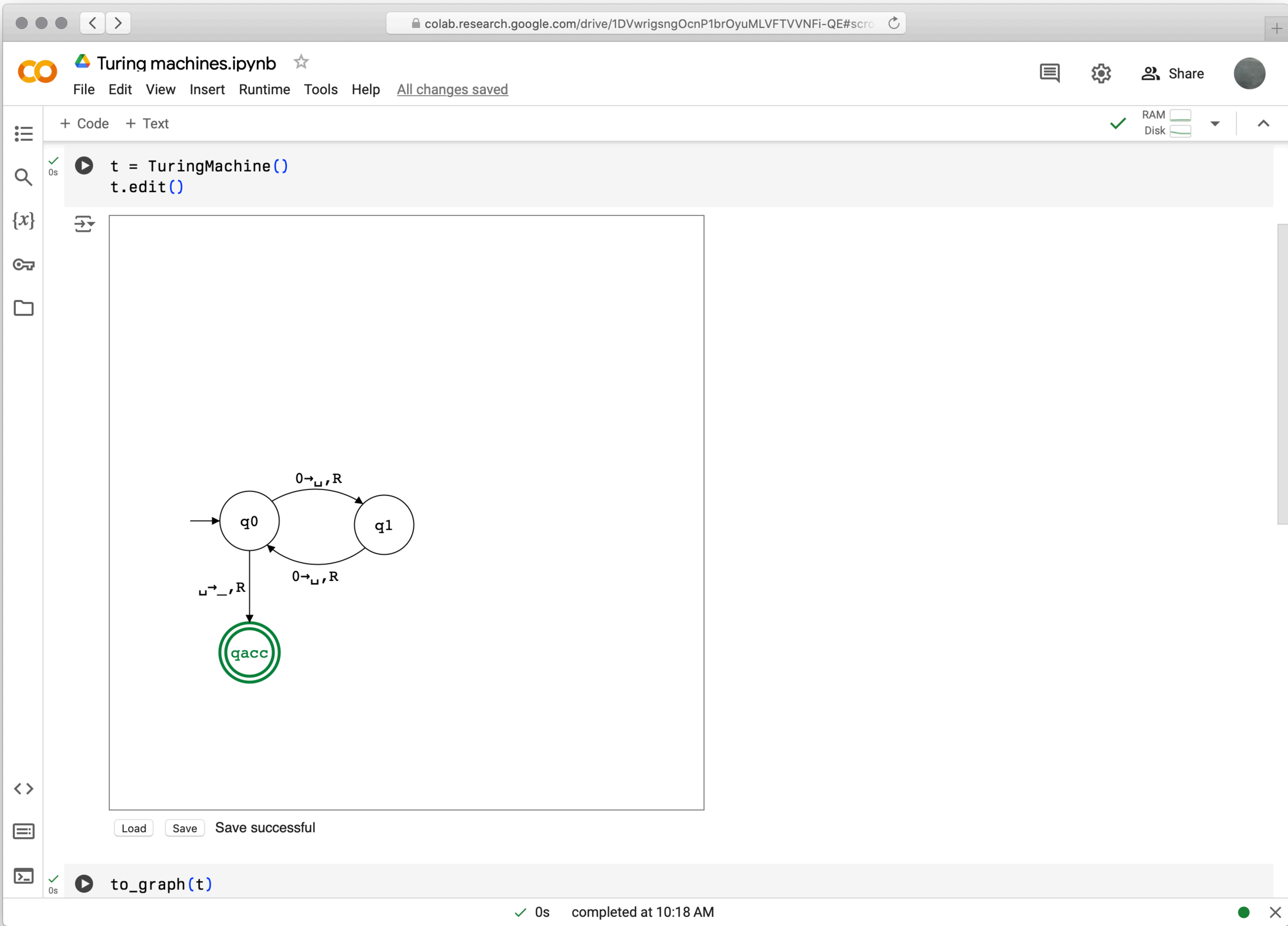
										0	0	1	1	1	1																	

One solution

										0	0	1	1	1	1																		



x → 0, L
reset xs to 0s
□ → □, R
□ → □, L
1 → 1, L
0 → 0, L
1 → 1, L
x → x, L
0 → 0, R
1 → 1, R
x → x, R
0 → 0, R
x → x, R
start
edge case
0 → 0, R
check if m = 0
1 → 1, L
go to start
□ → □, R
cross off 0
0 → x, R
go to end
□ → □, R
□ → □, R
□ → □, L
q_{acc}
erase 1 at end
1 → □, L


colab.research.google.com/drive/1DVwrigsngOcnP1brOyuMLVFTVVNFi-QE#scro
Turing machines.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
RAM
Disk
t = TuringMachine()
t.edit()
0→␣,R
q0
q1
0→␣,R
␣→_,R
qacc
Load
Save
Save successful
to_graph(t)
0s completed at 10:18 AM

Turing machine design

It can be helpful to think recursively when designing Turing machines.

It's often helpful to introduce new symbols into the tape alphabet.

Watch for edge cases that might lead to infinite loops!

Turing machine subroutines

On Tuesday, we designed Turing machines for

$$\{0^n 1^n \mid n \in \mathbb{N}_0\}$$

and

$$\{w \in \Sigma^* \mid n_0(w) = n_1(w)\}.$$

Clearly these languages are related!

So, could we use our Turing machine for the first language to recognize the second?

Could we sort the characters of this string?

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***`0*1*`***.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***`0*1*`***.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***0*1****.

Observation 1:
A string of `0`*s and* `1`*s is sorted if it matches the regex* ***`0*1*`***.

Observation 2:
A string of 0*s and* 1*s is not sorted if it contains* 10 *as a substring.*

Observation 2:
A string of `0`*s and* `1`*s is not sorted if it contains* `10` *as a substring.*

Observation 2:
A string of 0*s and* 1*s is not sorted if it contains* 10 *as a substring.*

Observation 2:
A string of `0`*s and* `1`*s is* *not* *sorted if it contains* `10` *as a substring.*

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Idea:
Repeatedly find a copy of `10` *and replace it with* `01`.

Let’s build it!

														0	0	1	1	0	0														

start

0	0	1	1	0	0

start

0*

0	0	1	1	0	0

start

0*

0 0 1 1 0 0

0 → 0, R

start

0*

0	0	1	1	0	0

0 → 0, R

start

0*

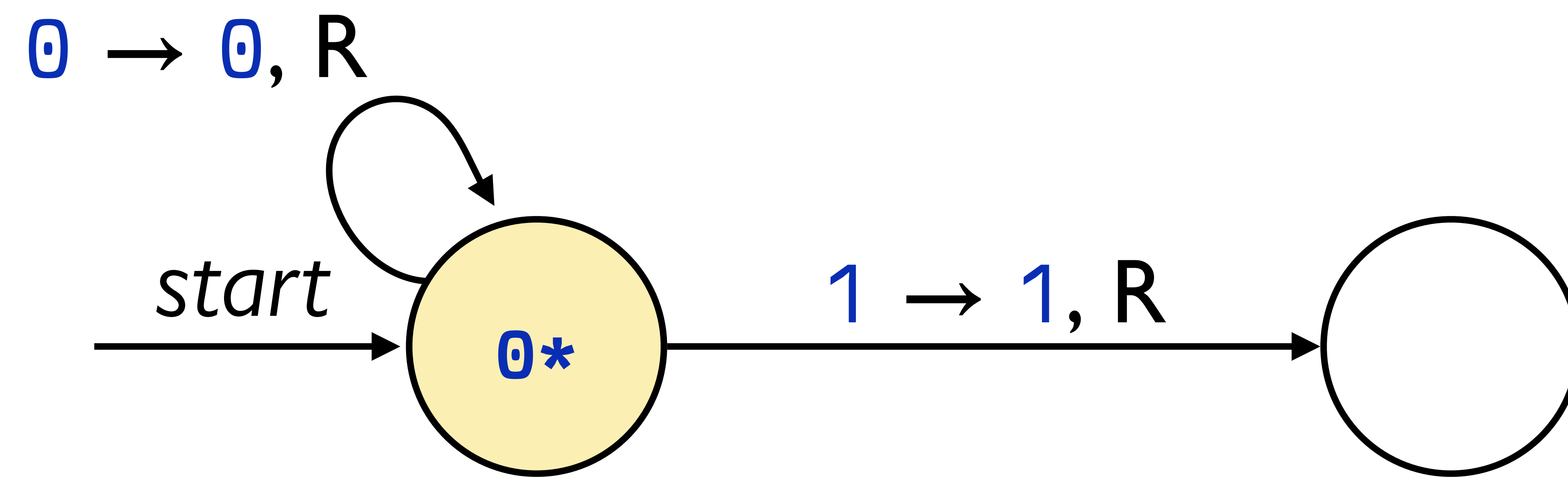
0 0 1 1 0 0

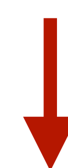
0 → 0, R

start

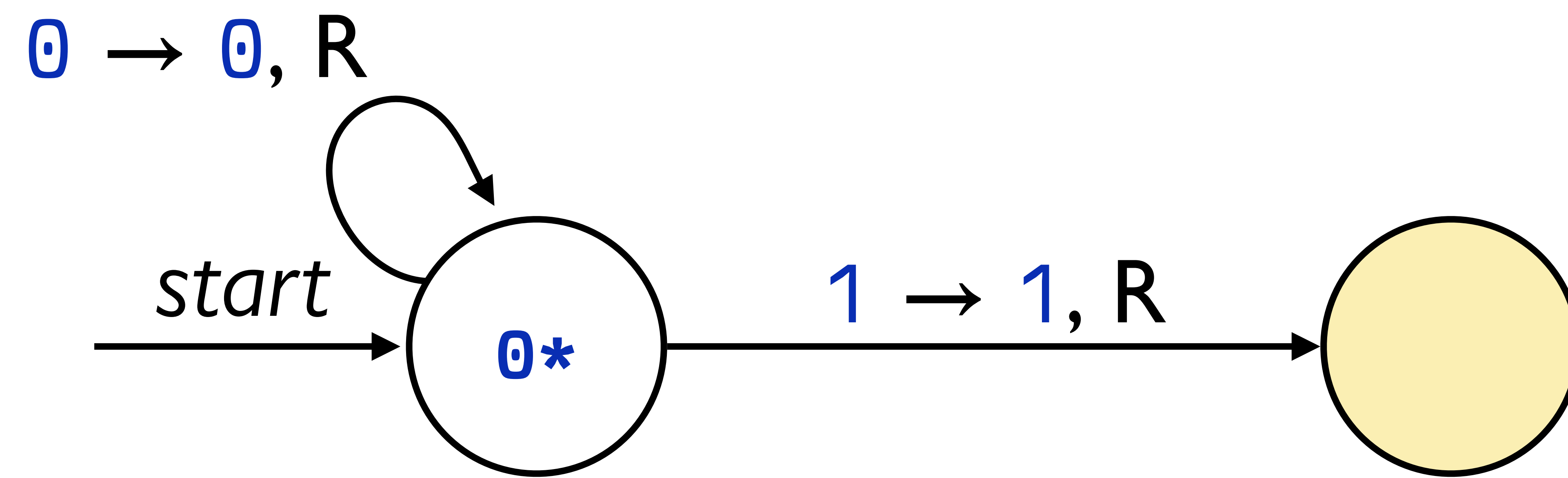
0*

0	0	1	1	0	0

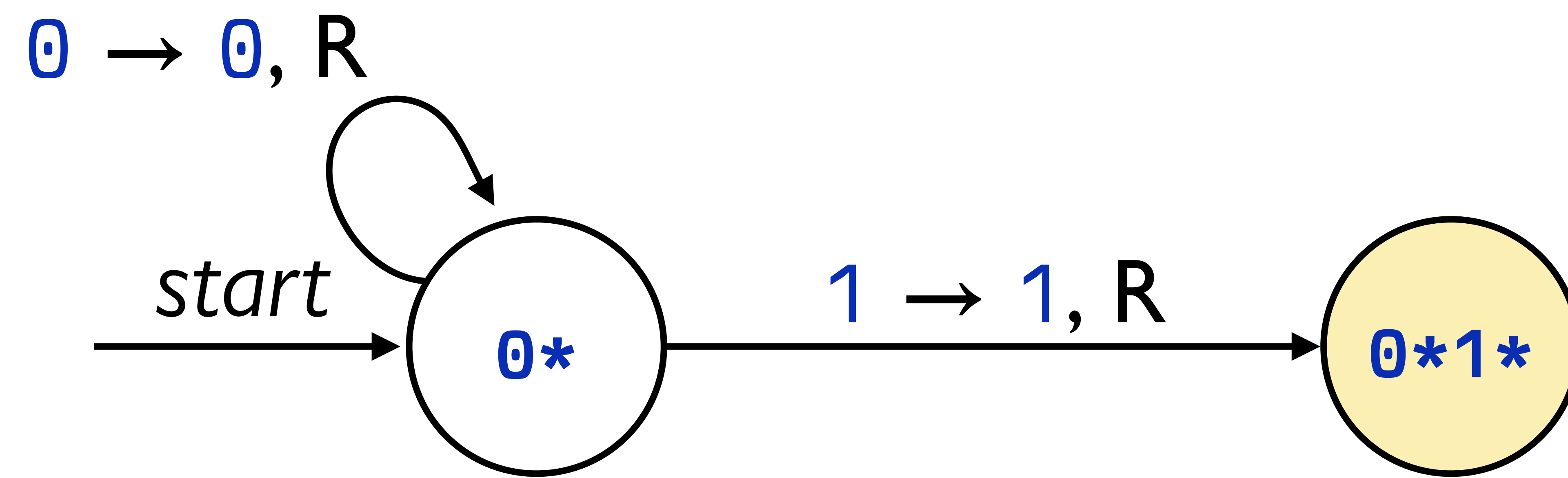

0 → 0, R
start
0*
1 → 1, R



0 0 1 1 0 0


0 → 0, R
start
0*
1 → 1, R

0 0 1 1 0 0


0 → 0, R
start
0*
1 → 1, R
0*1*

0 0 1 1 0 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

														0	0	1	1	0	0														

0 → 0, R

start

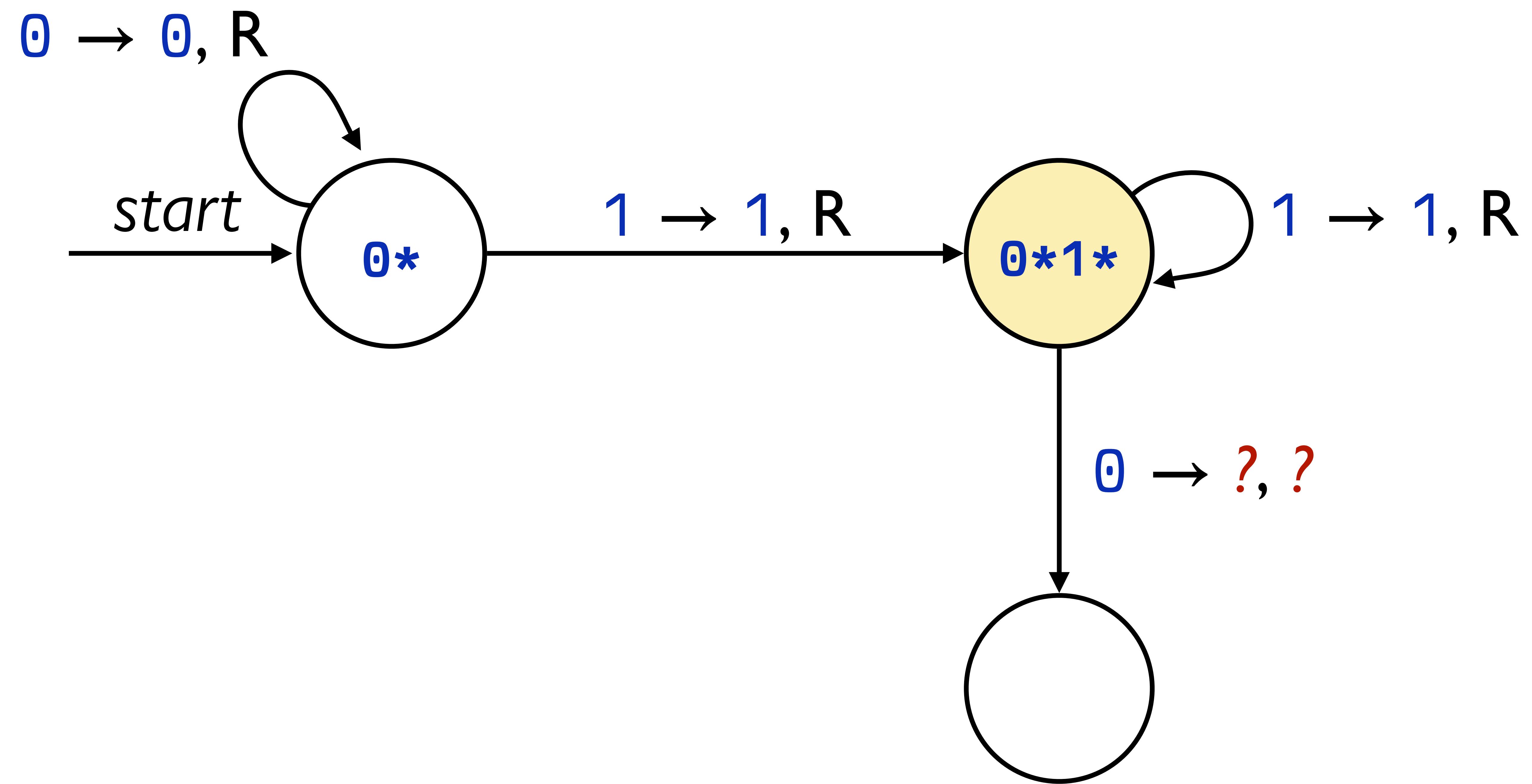
0*

1 → 1, R

0*1*

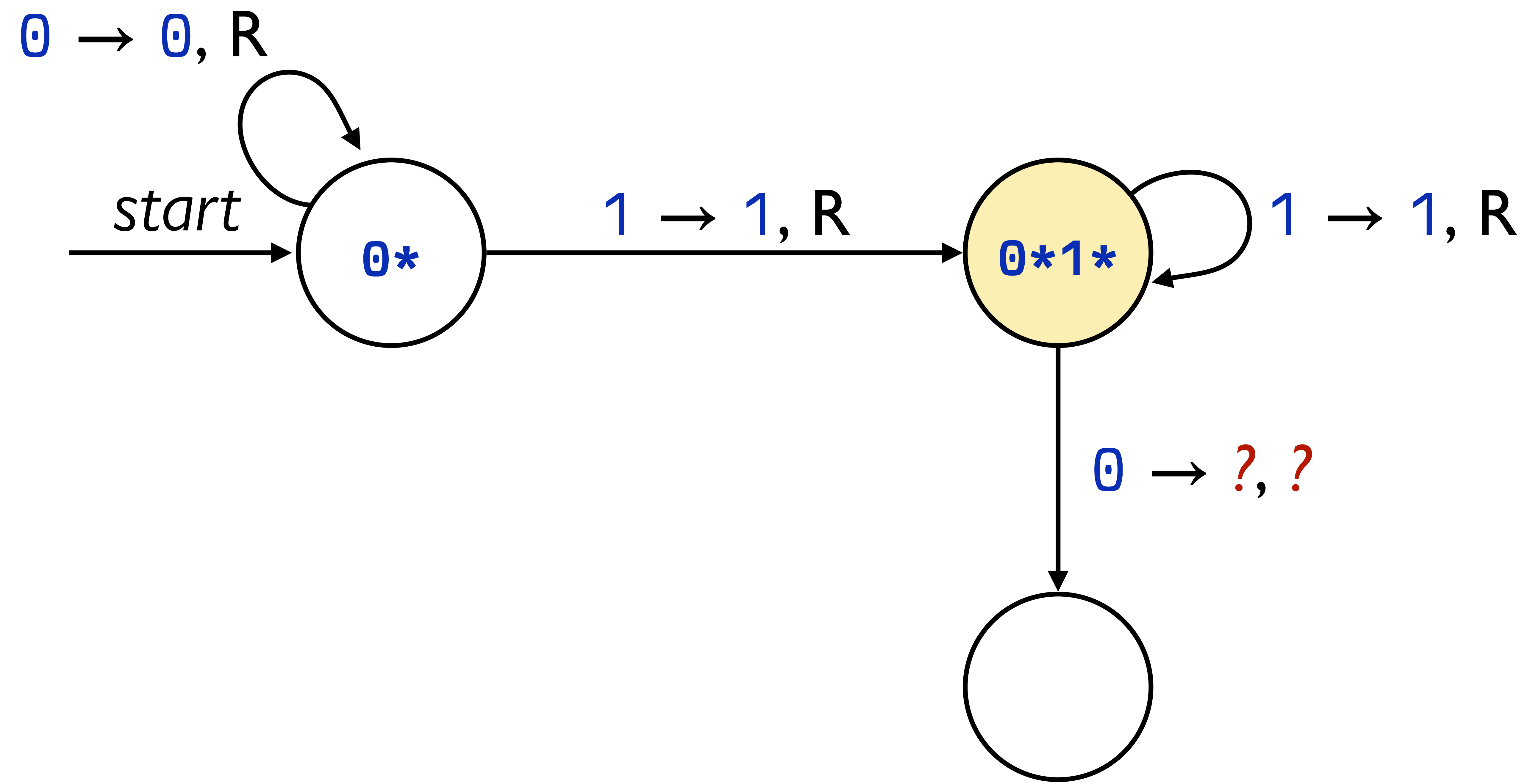
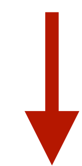
1 → 1, R

0	0	1	1	0	0

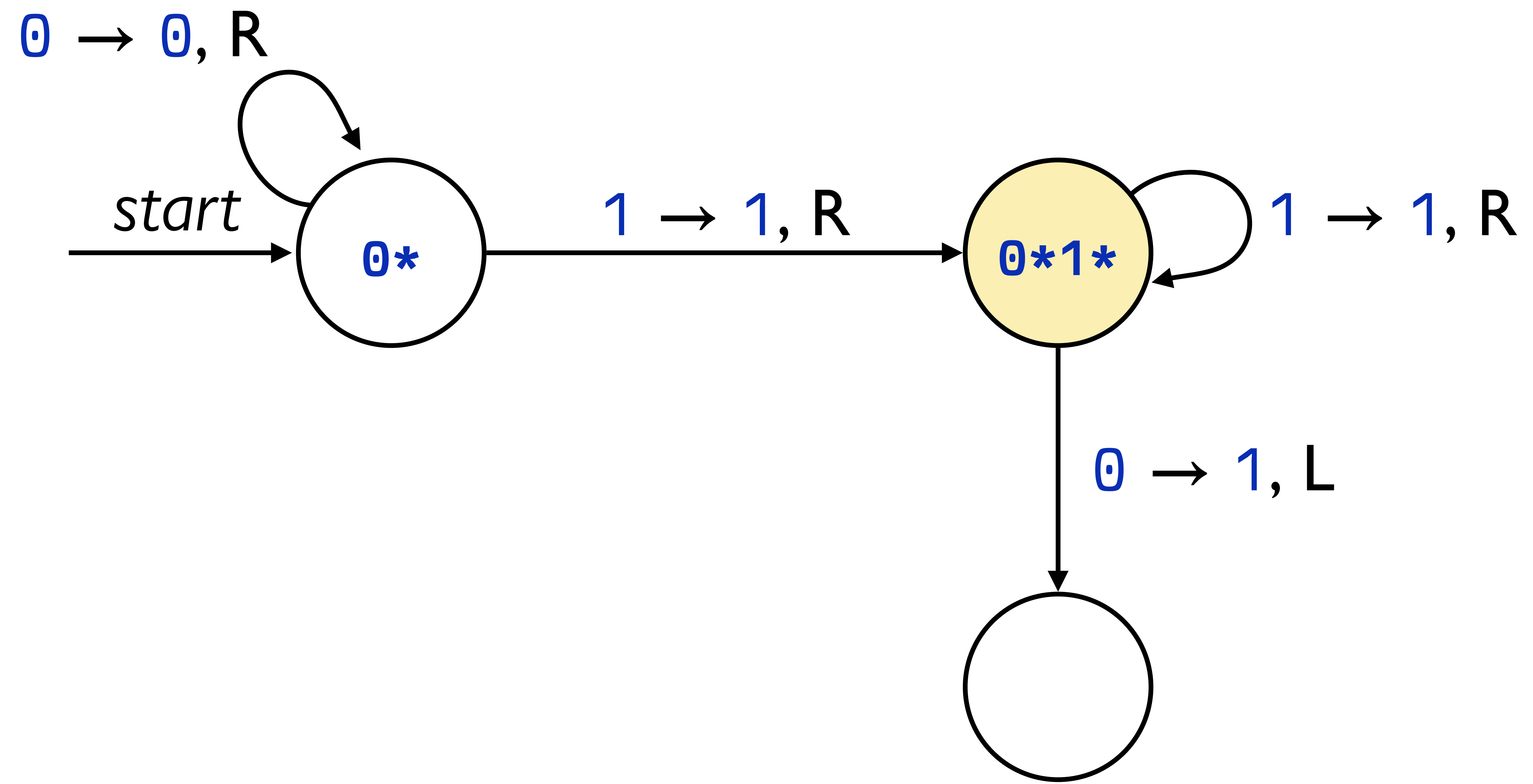


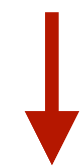
0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → ?, ?

0 0 1 1 0 0

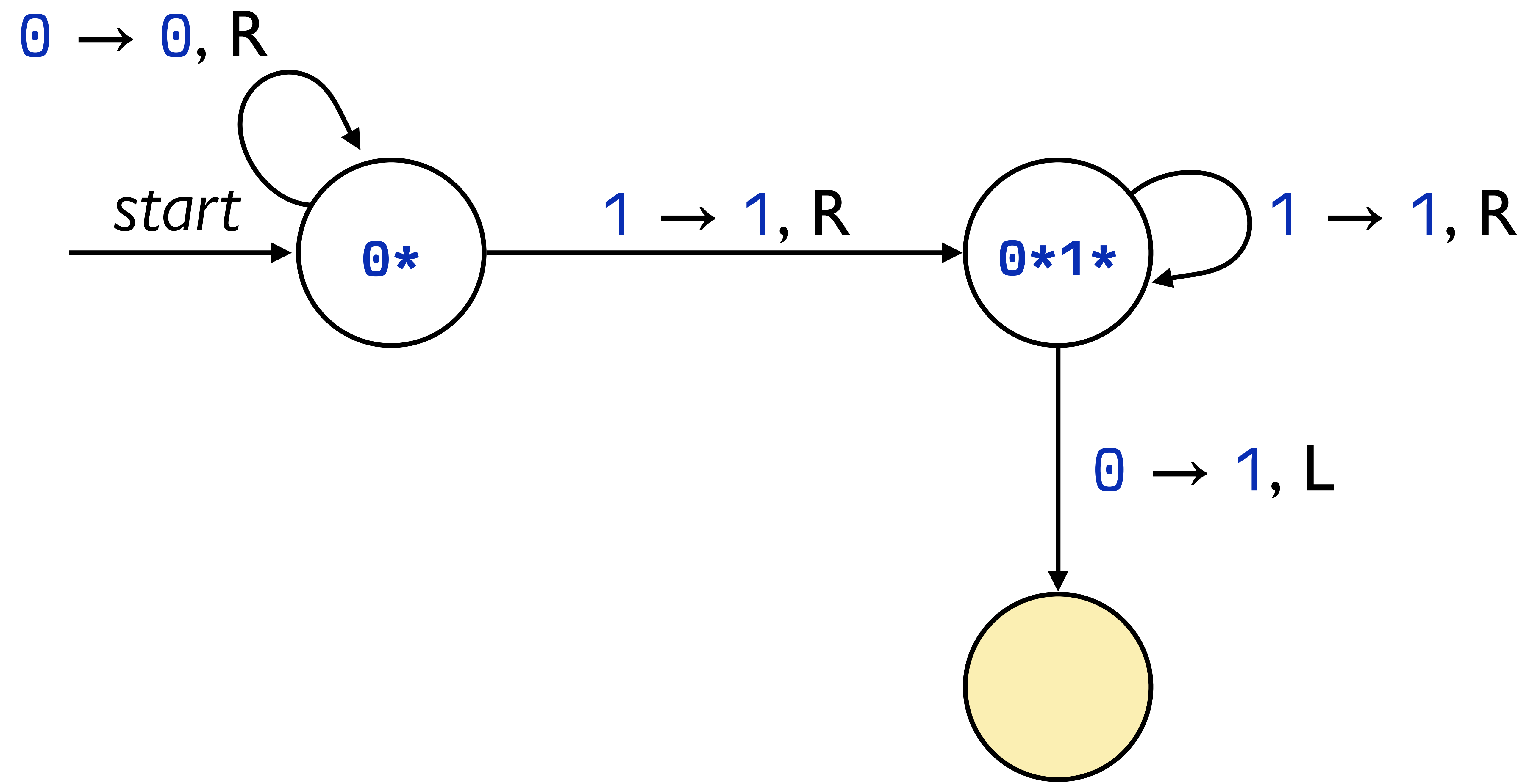

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → ?, ?


0 0 1 1 0 0

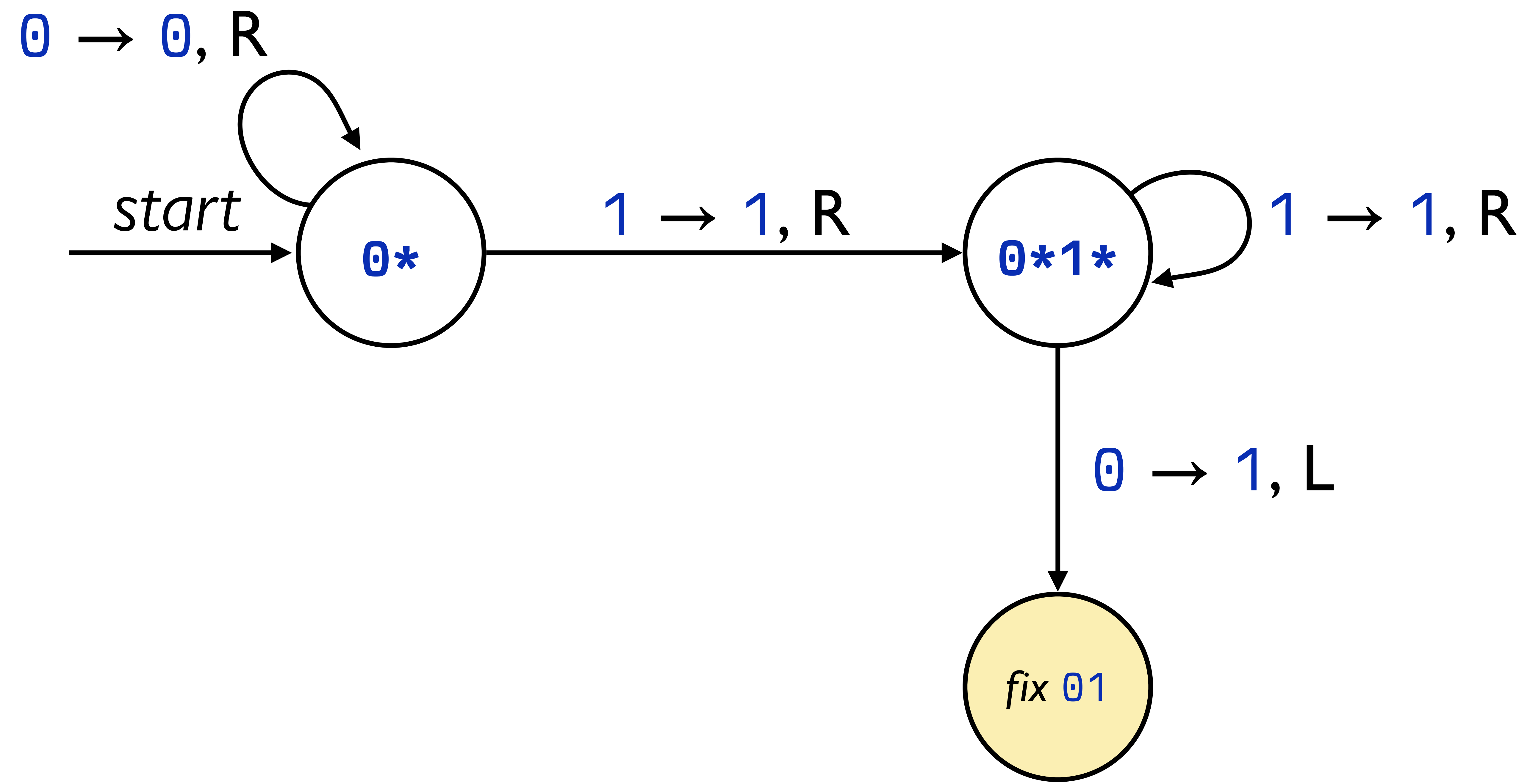

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L



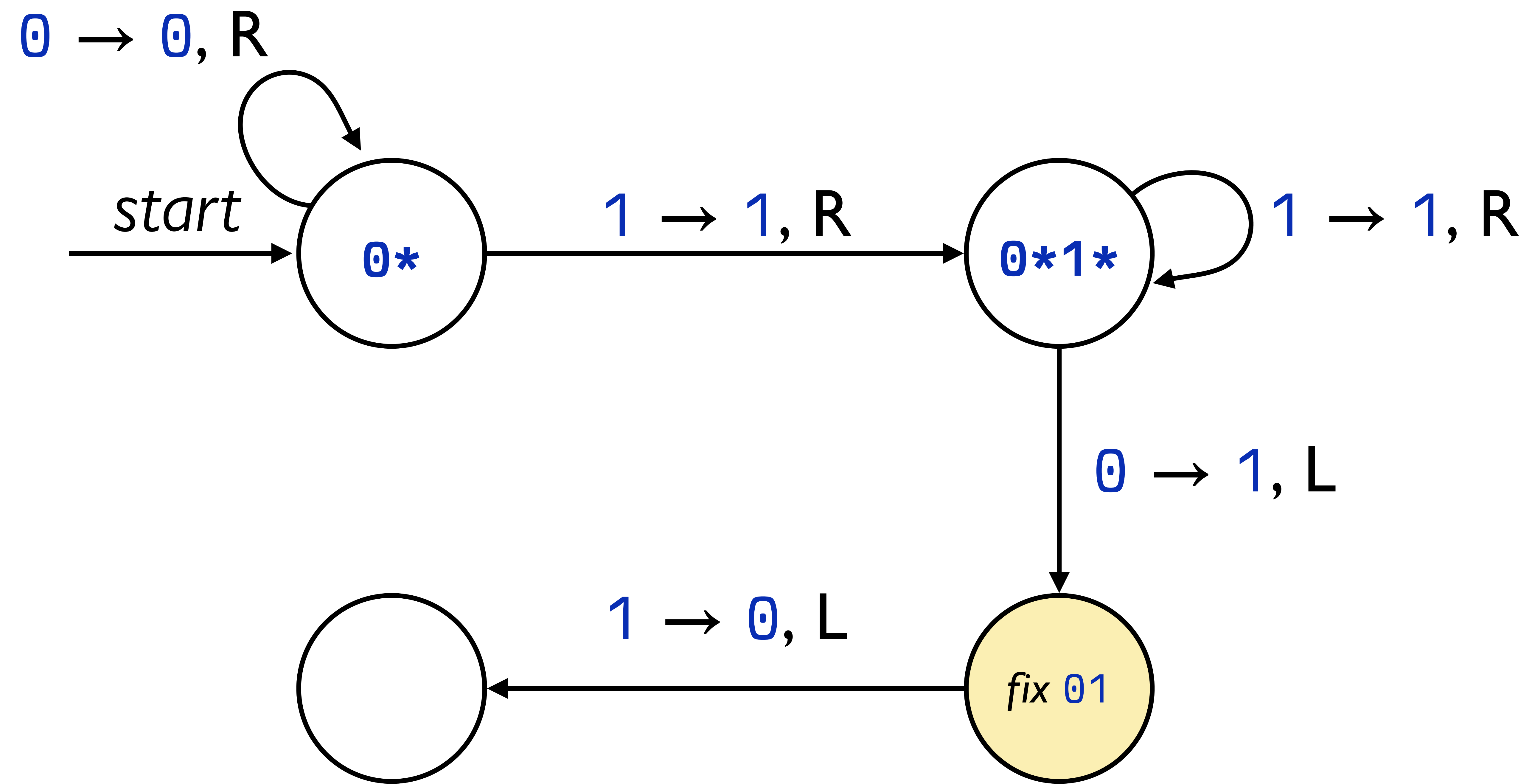
0 0 1 1 0 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L

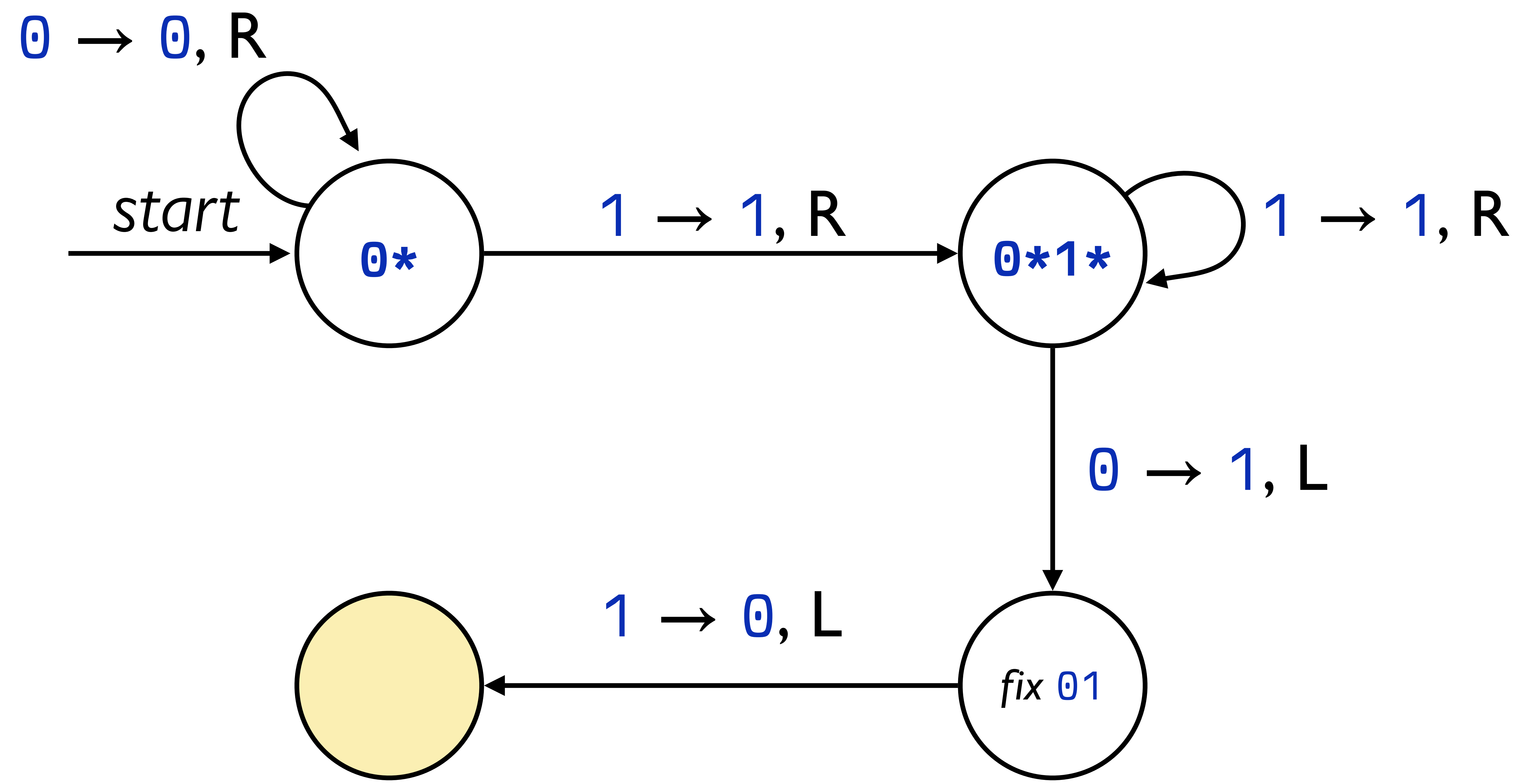
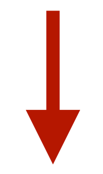
0 0 1 1 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
fix 01

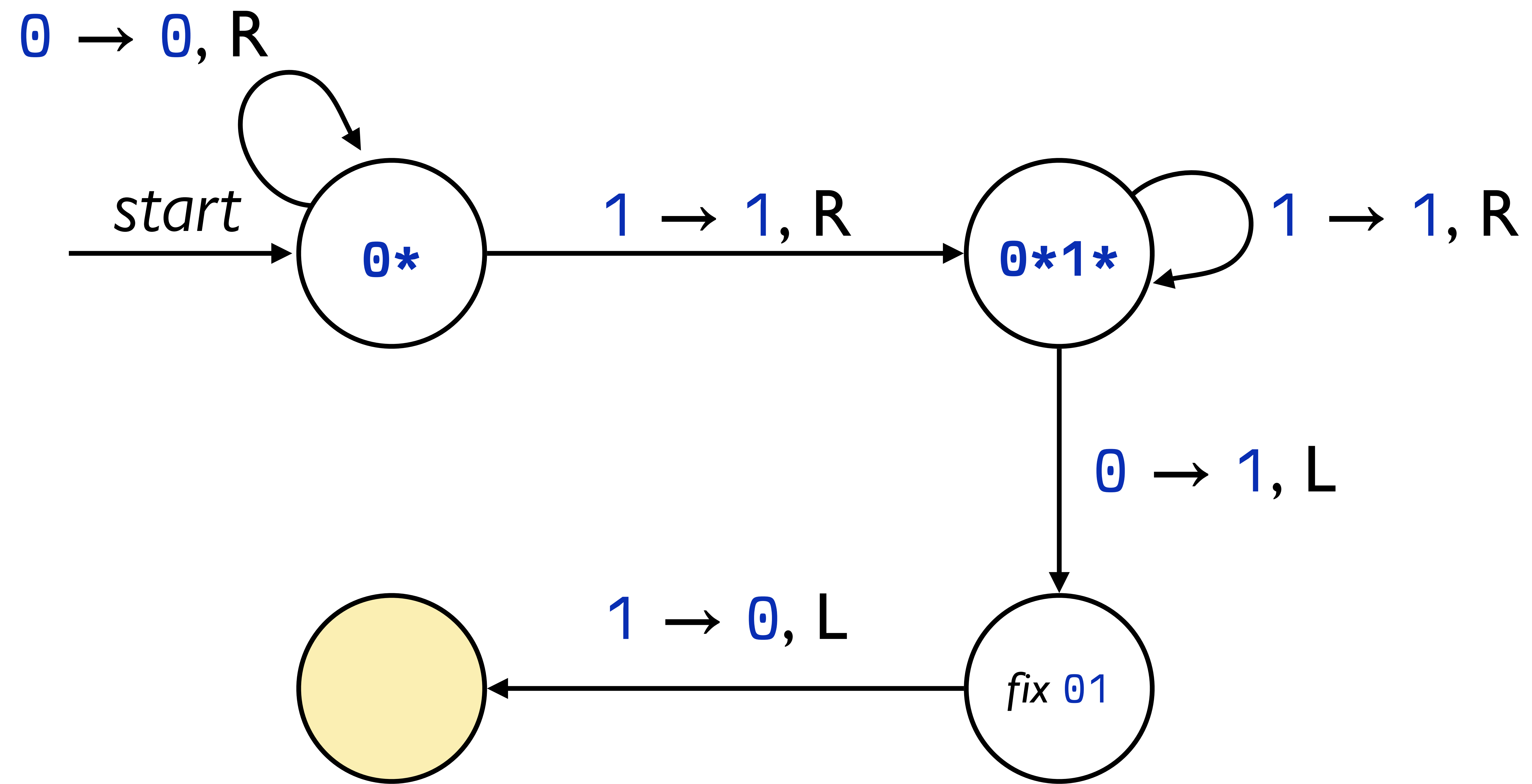
0 0 1 1 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
fix 01

0 0 1 1 1 0

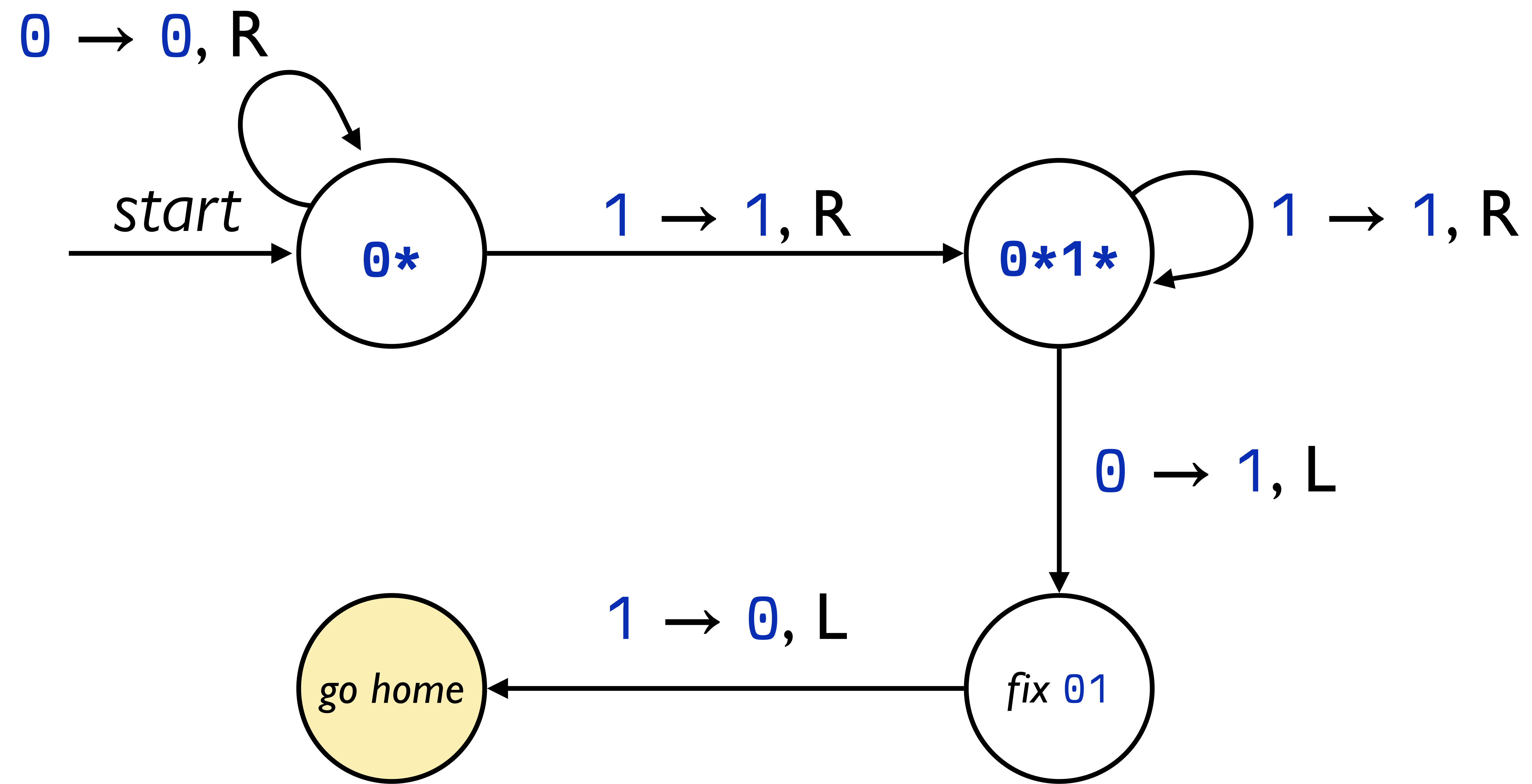

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
fix 01


0 0 1 0 1 0

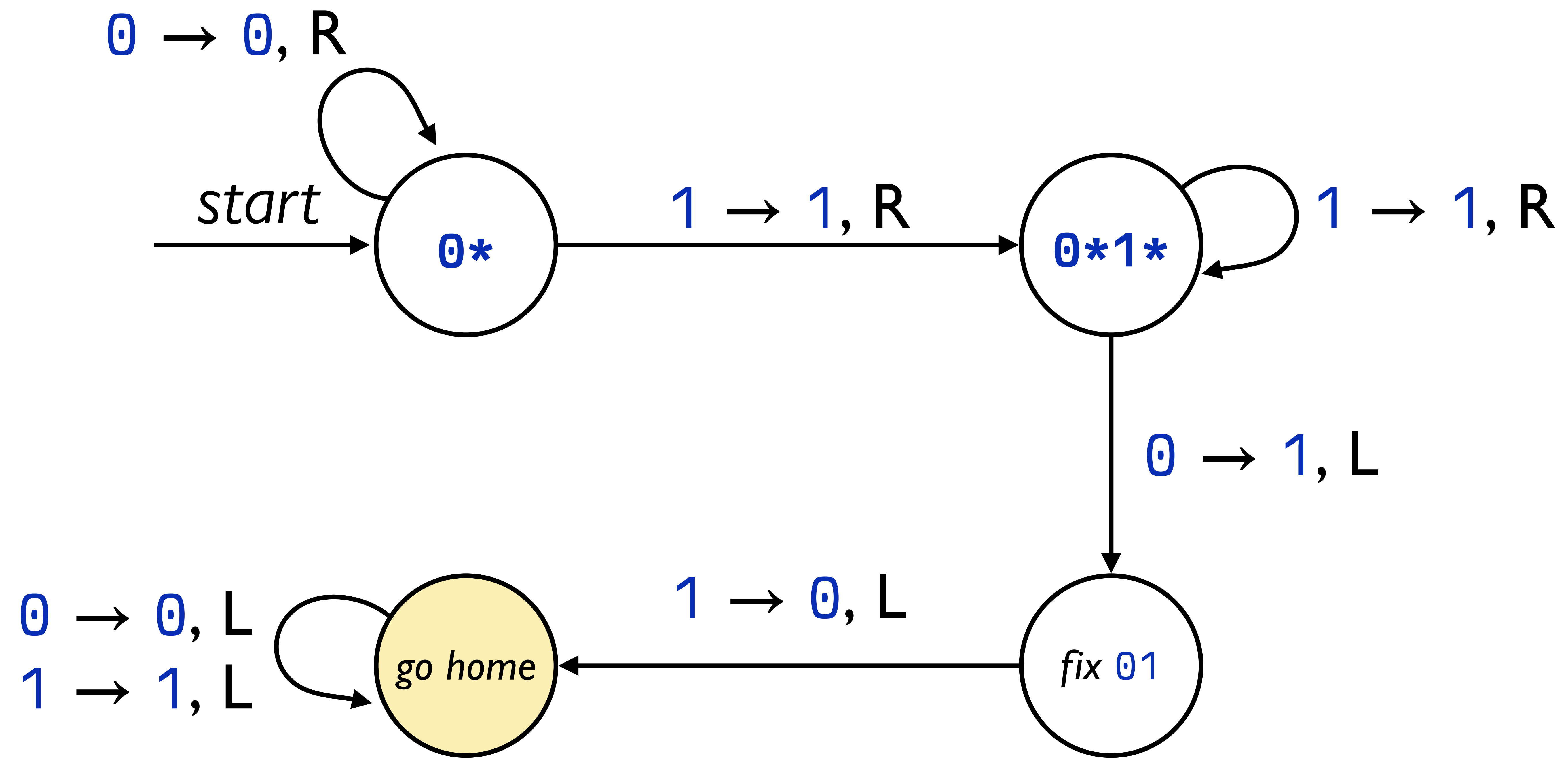


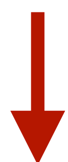
0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
fix 01

0 0 1 0 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
go home
fix 01

0 0 1 0 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 1 0 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

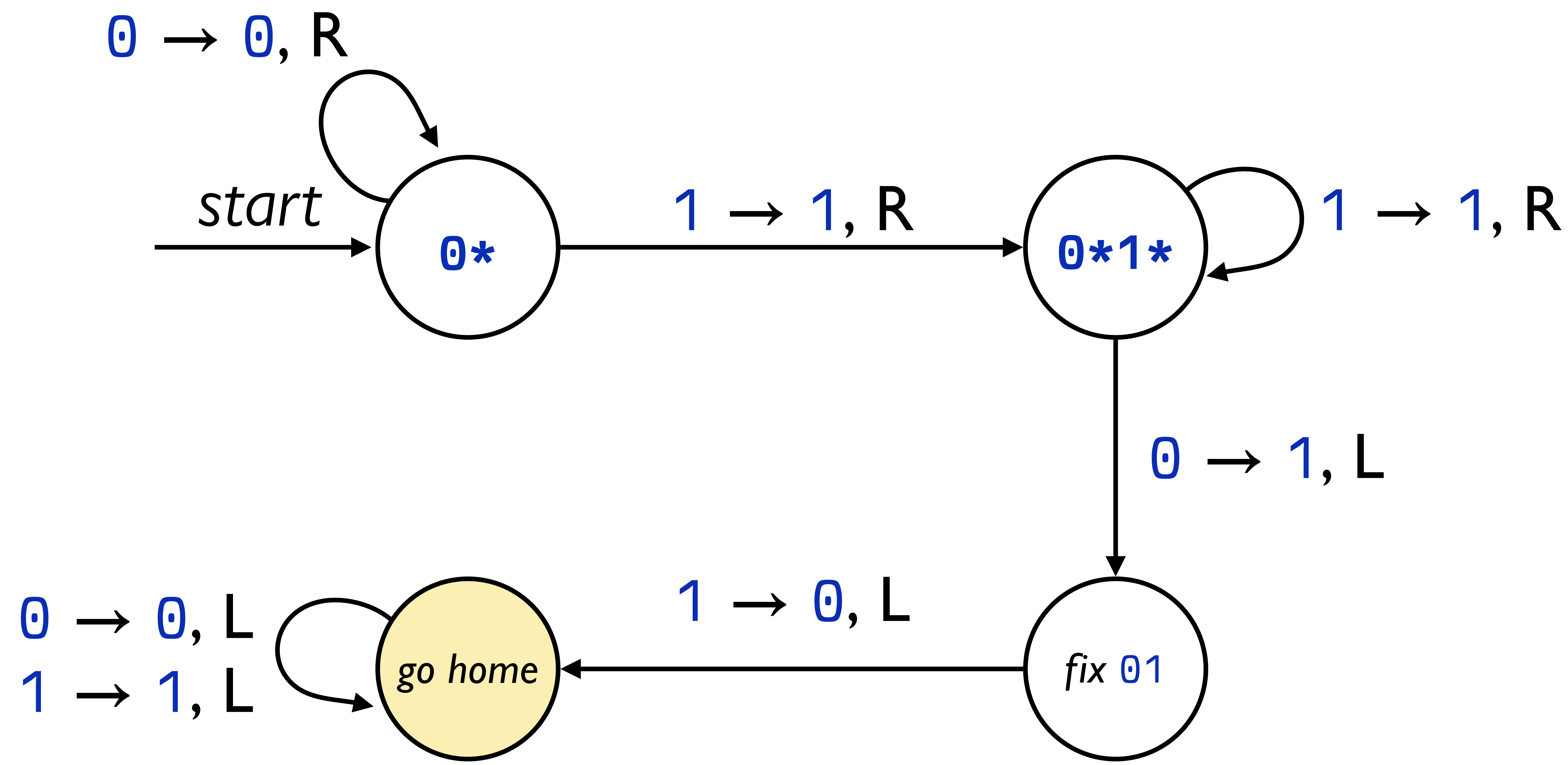
0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	1	0	1	0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 1 0 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

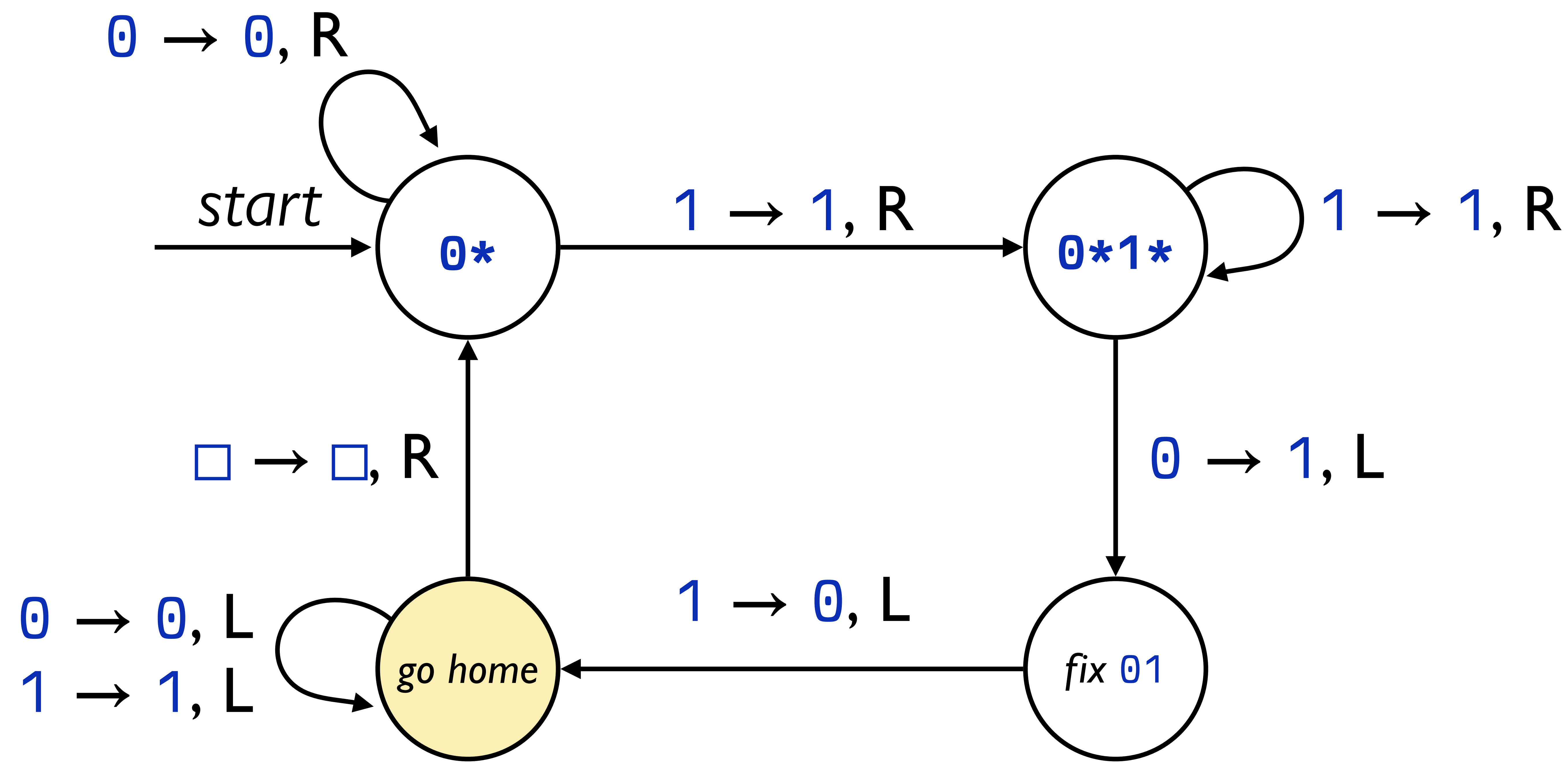
0 → 0, L
1 → 1, L

go home

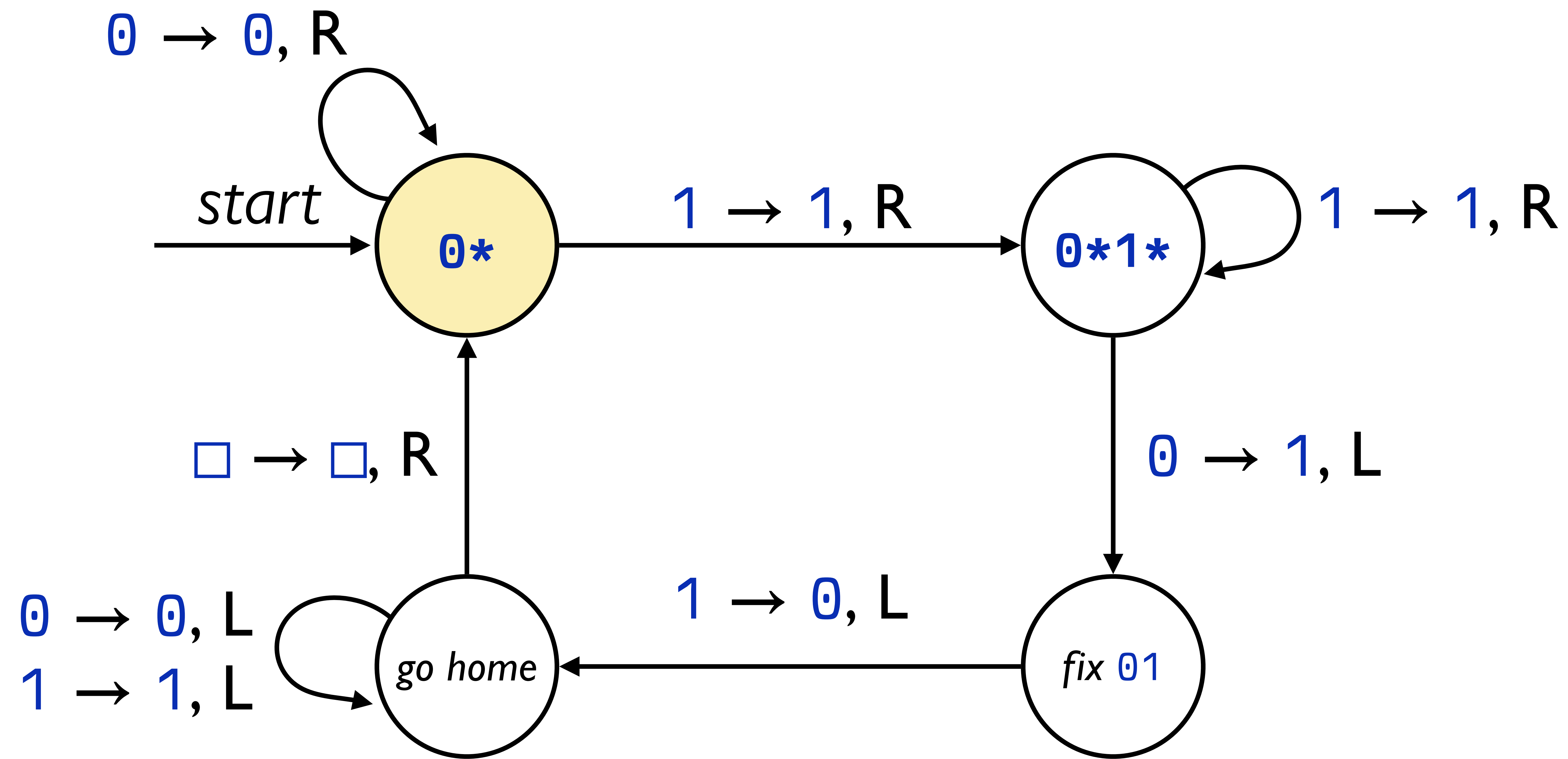
1 → 0, L

fix 01

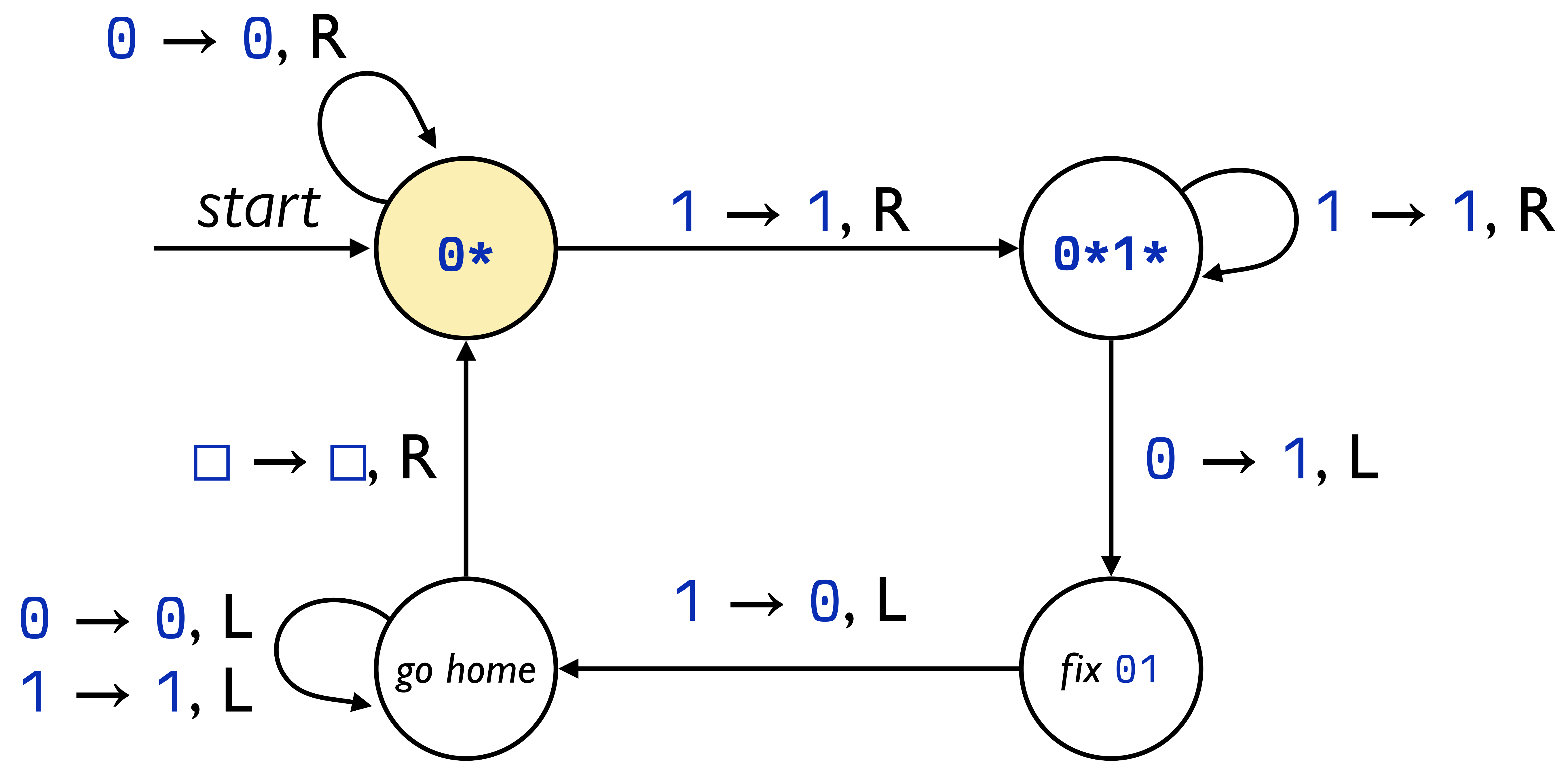
														0	0	1	0	1	0														

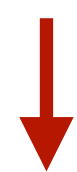

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 1 0 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 1 0 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 1 0 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	1	0	1	0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

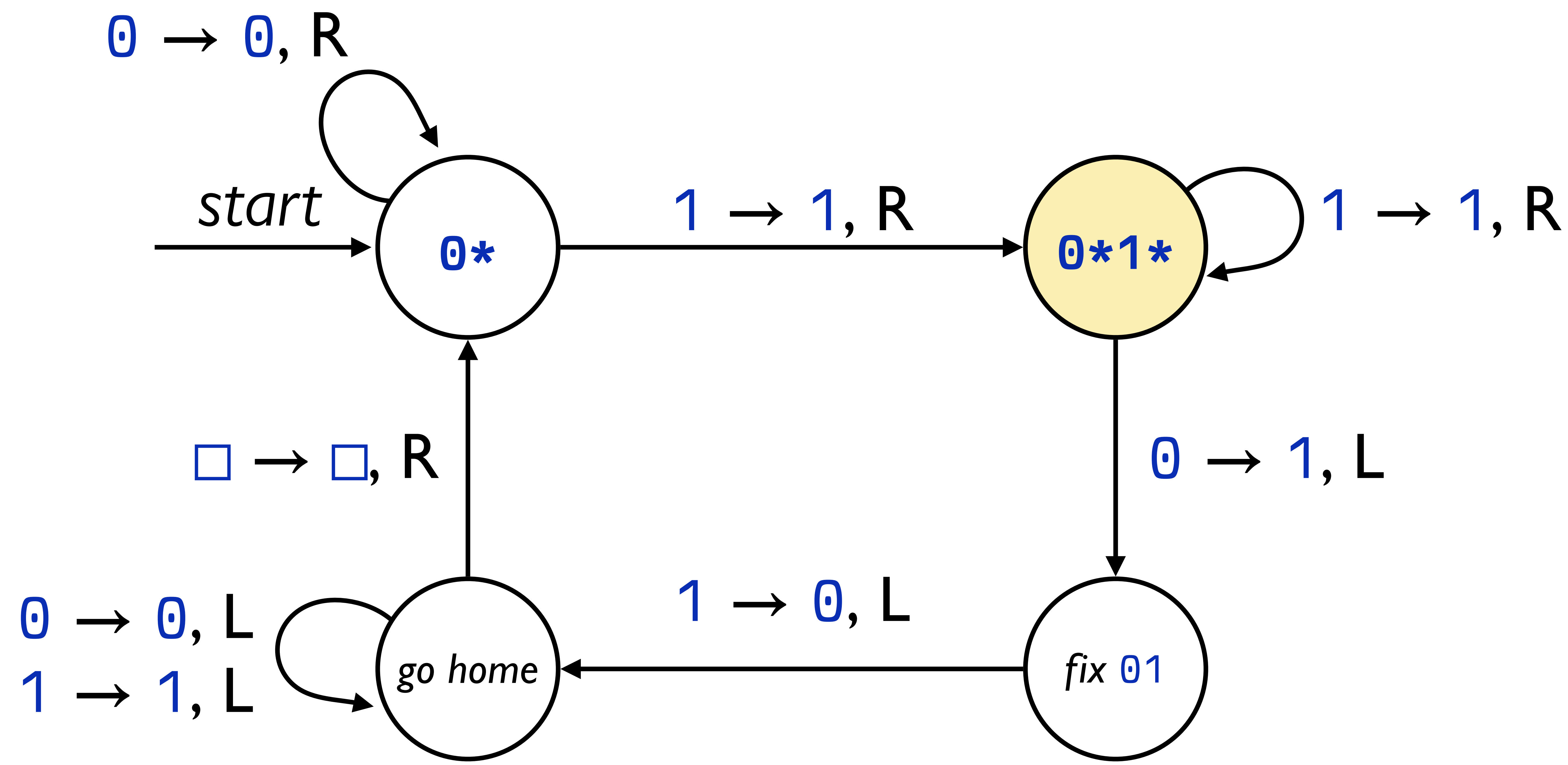
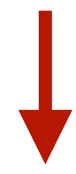
0 → 0, L
1 → 1, L

go home

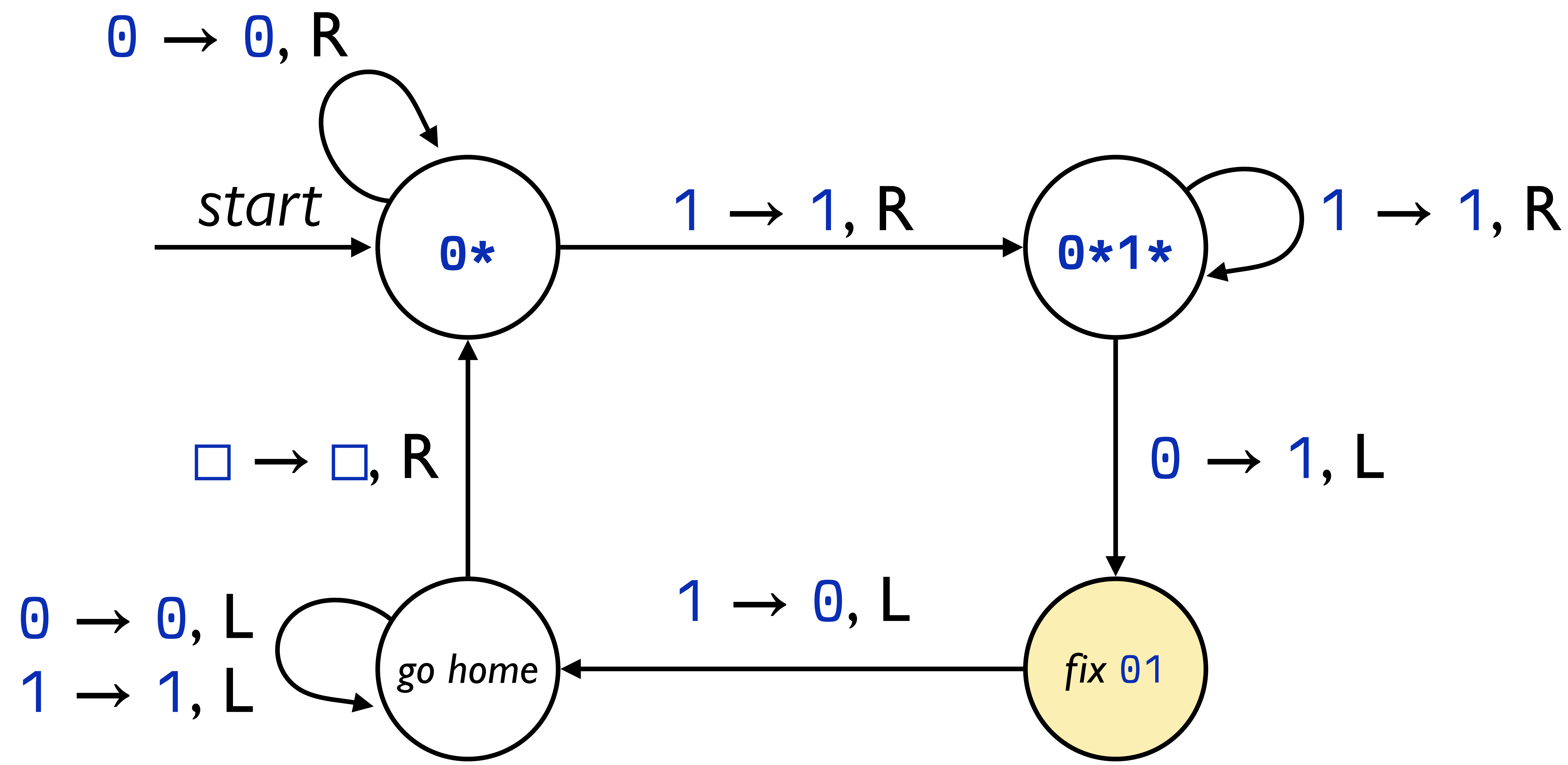
1 → 0, L

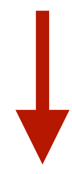
fix 01

0	0	1	0	1	0

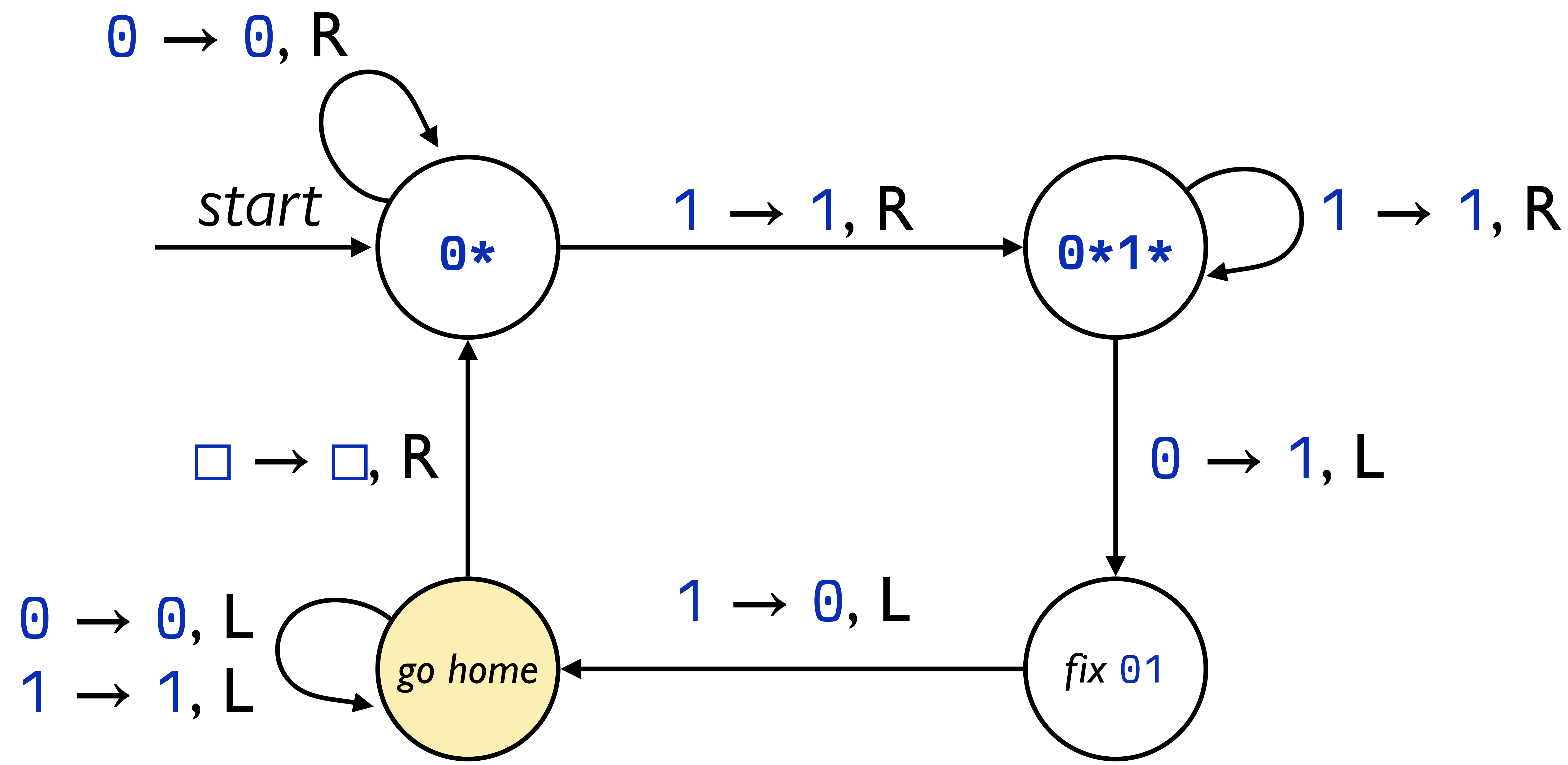

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01


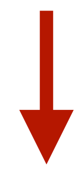
0 0 1 0 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 1 1 1 0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 0 1 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

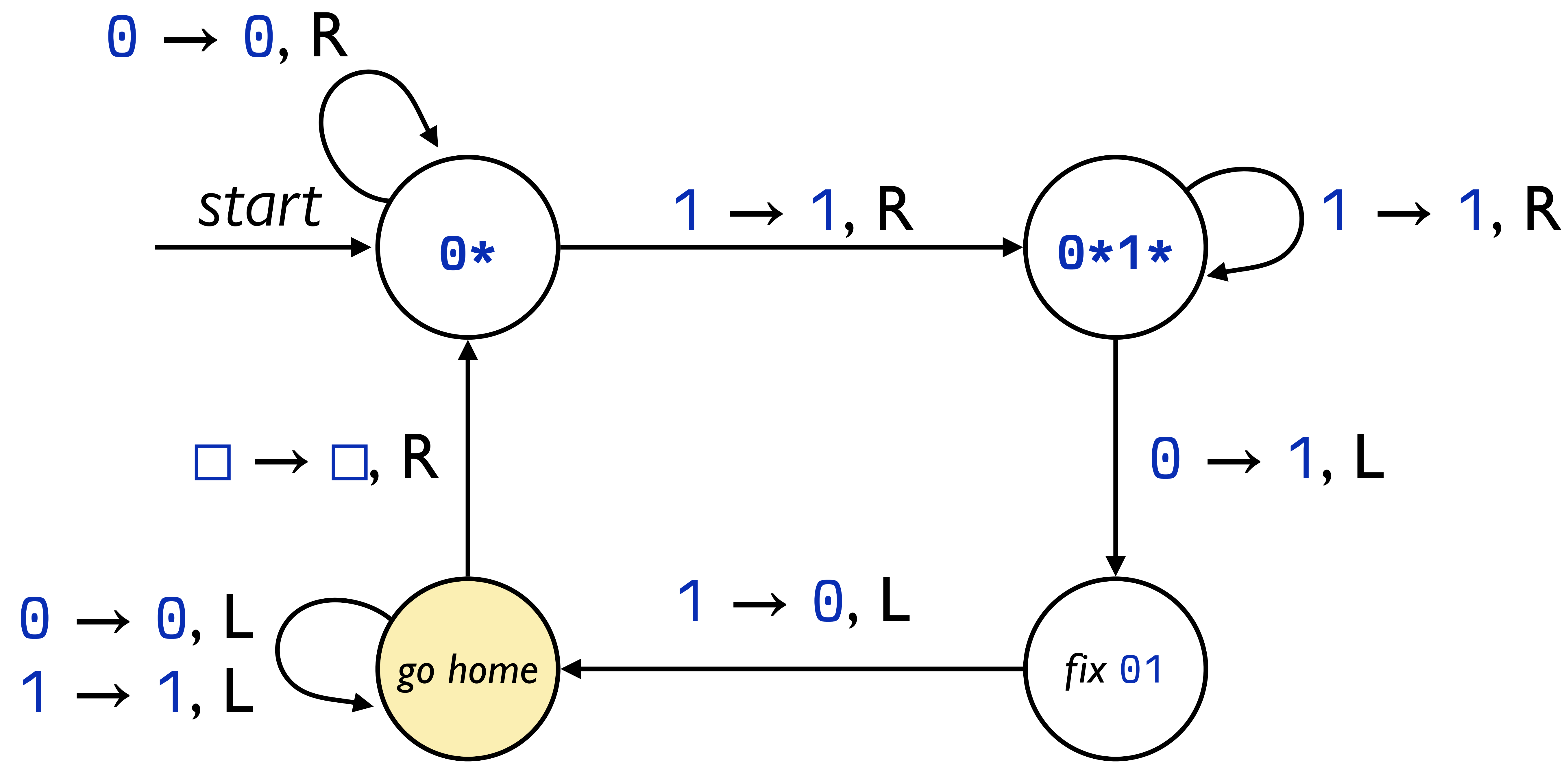
0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	1	1	0


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 0 1 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

														0	0	0	1	1	0														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	1	1	0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	1	1	0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	1	0														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

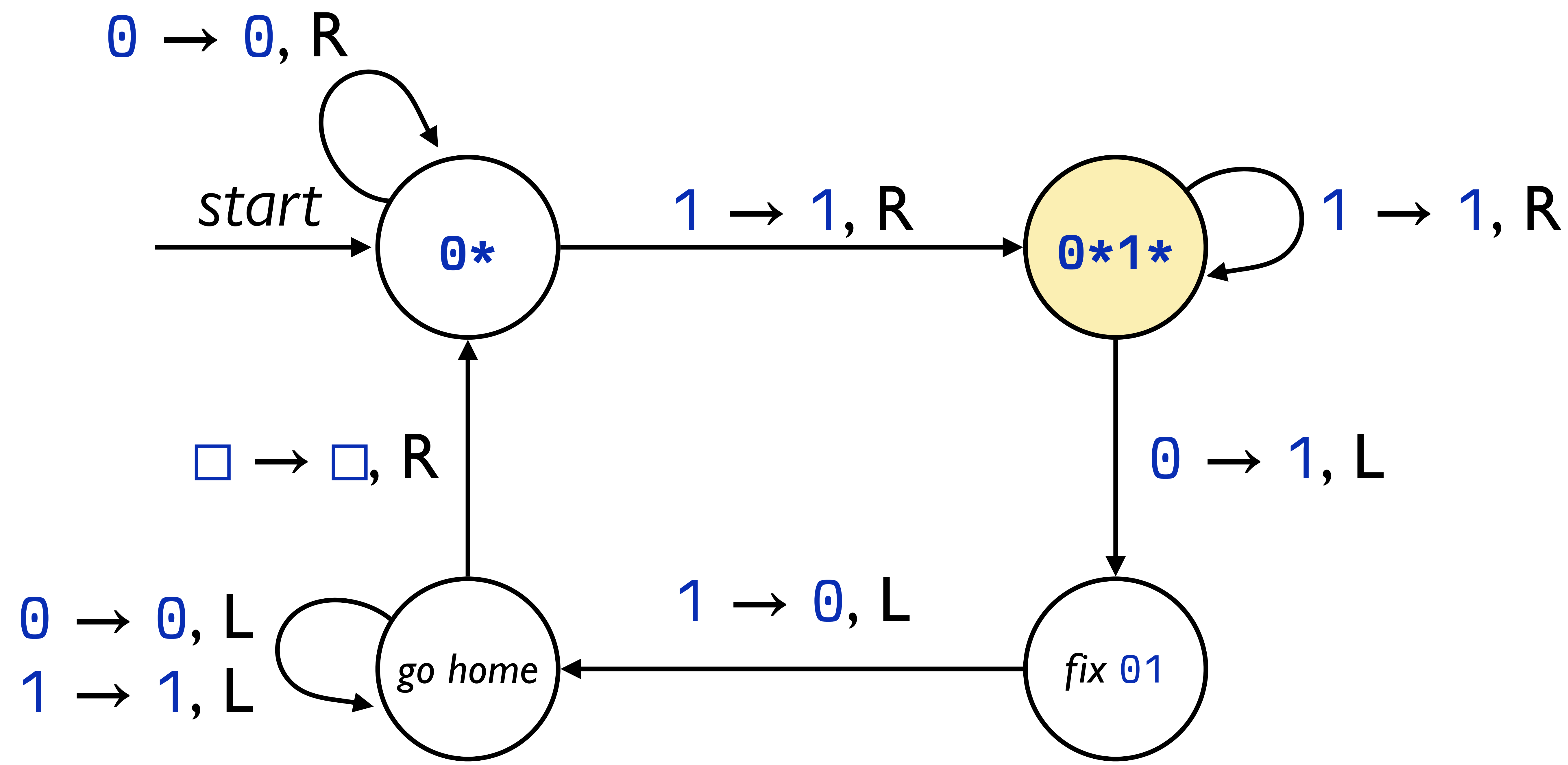
0 → 0, L
1 → 1, L

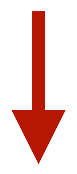
go home

1 → 0, L

fix 01

														0	0	0	1	1	0														


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
1 → 0, L
go home
fix 01



0 0 0 1 1 0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

0	0	0	1	1	0

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	1	1														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	0	1														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	0	1													

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	0	1														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	1	0	1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	1	0	1														

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01
0 0 0 1 0 1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

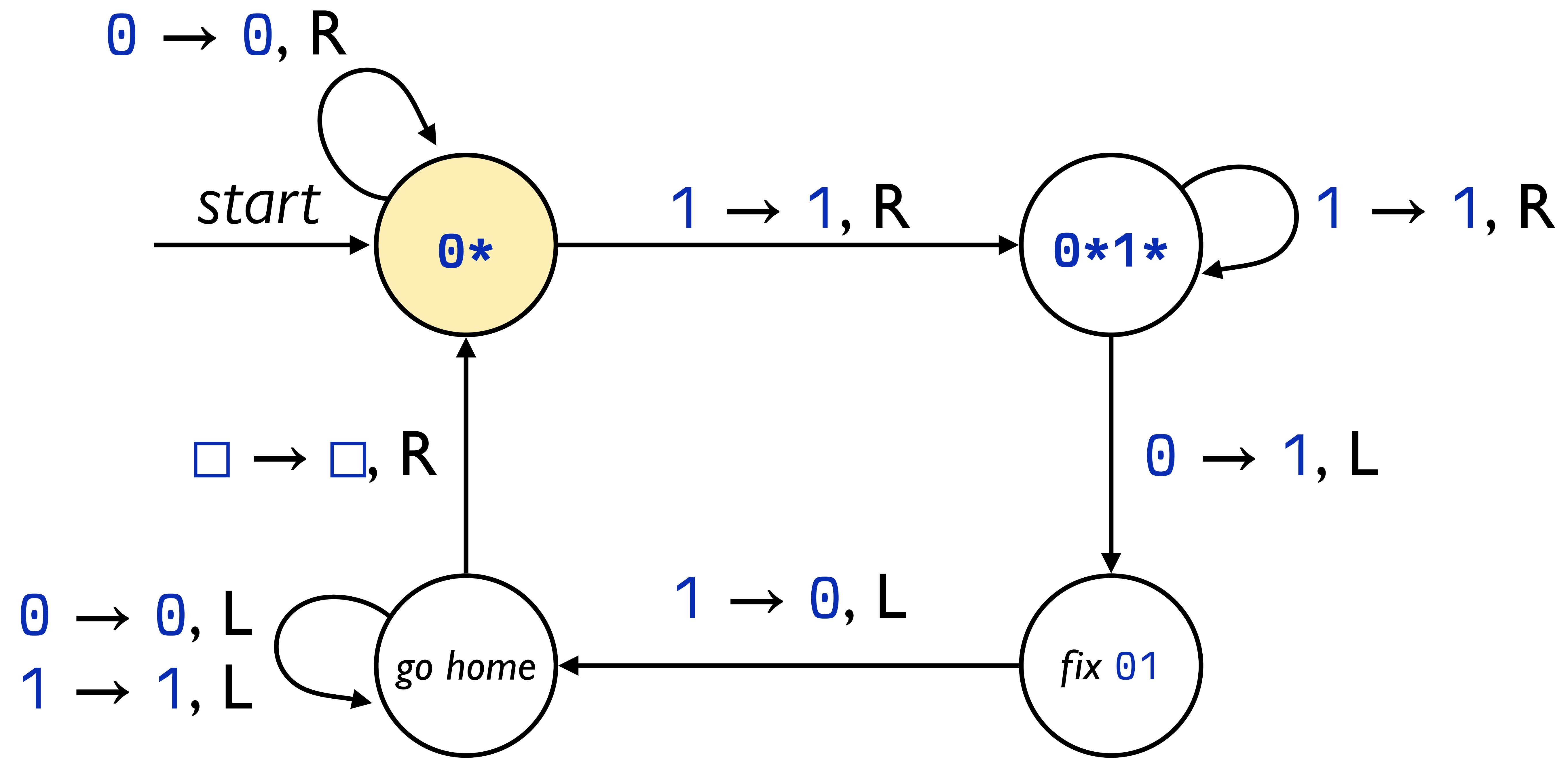
0 → 0, L
1 → 1, L

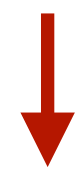
go home

1 → 0, L

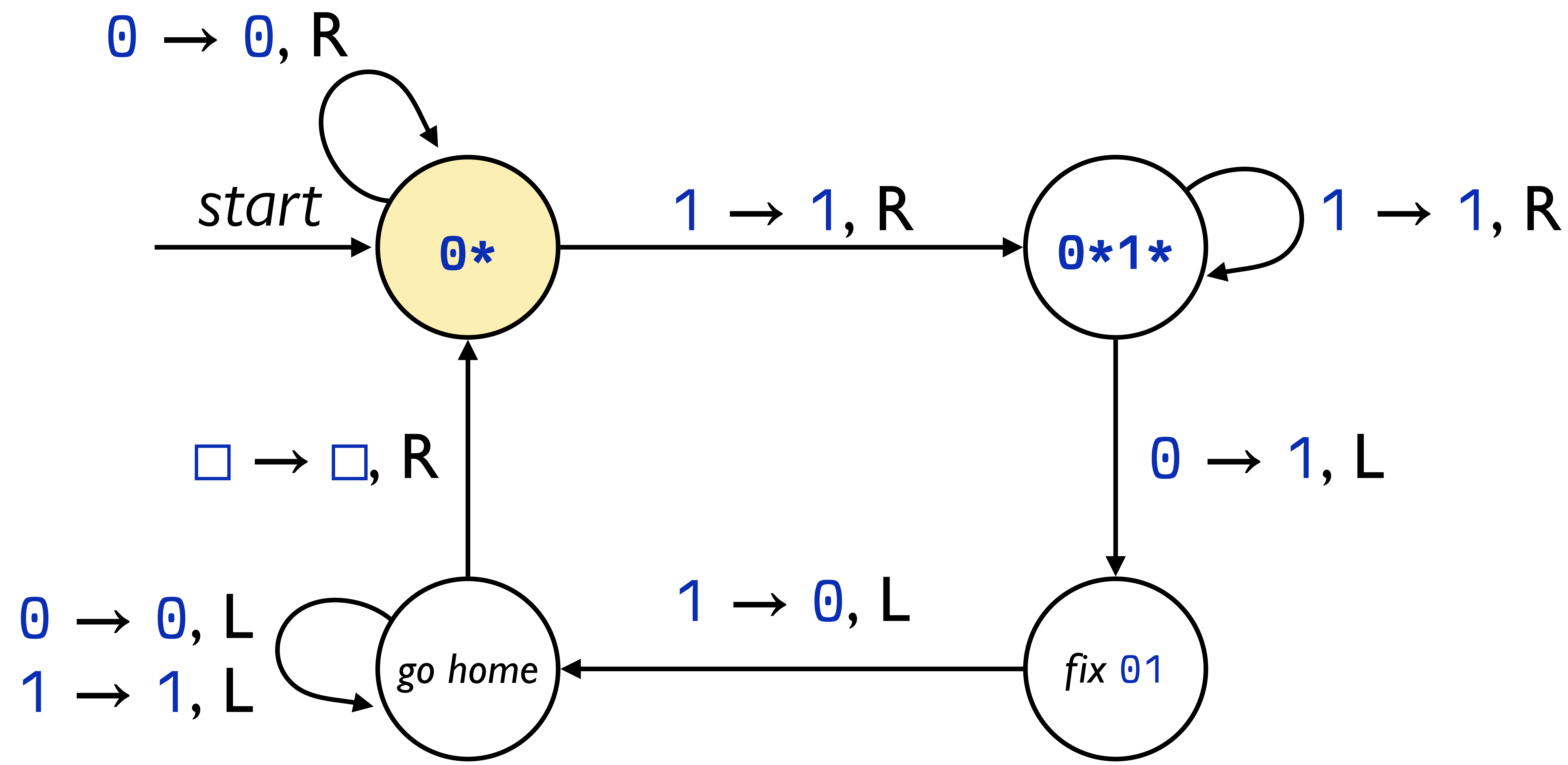
fix 01

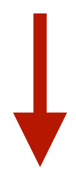
														0	0	0	1	0	1														


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 0 1 0 1


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 0 1 0 1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	1	0	1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

														0	0	0	1	1	1												

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	0	1	1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	0	1	1														

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0	0	0	0	1	1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

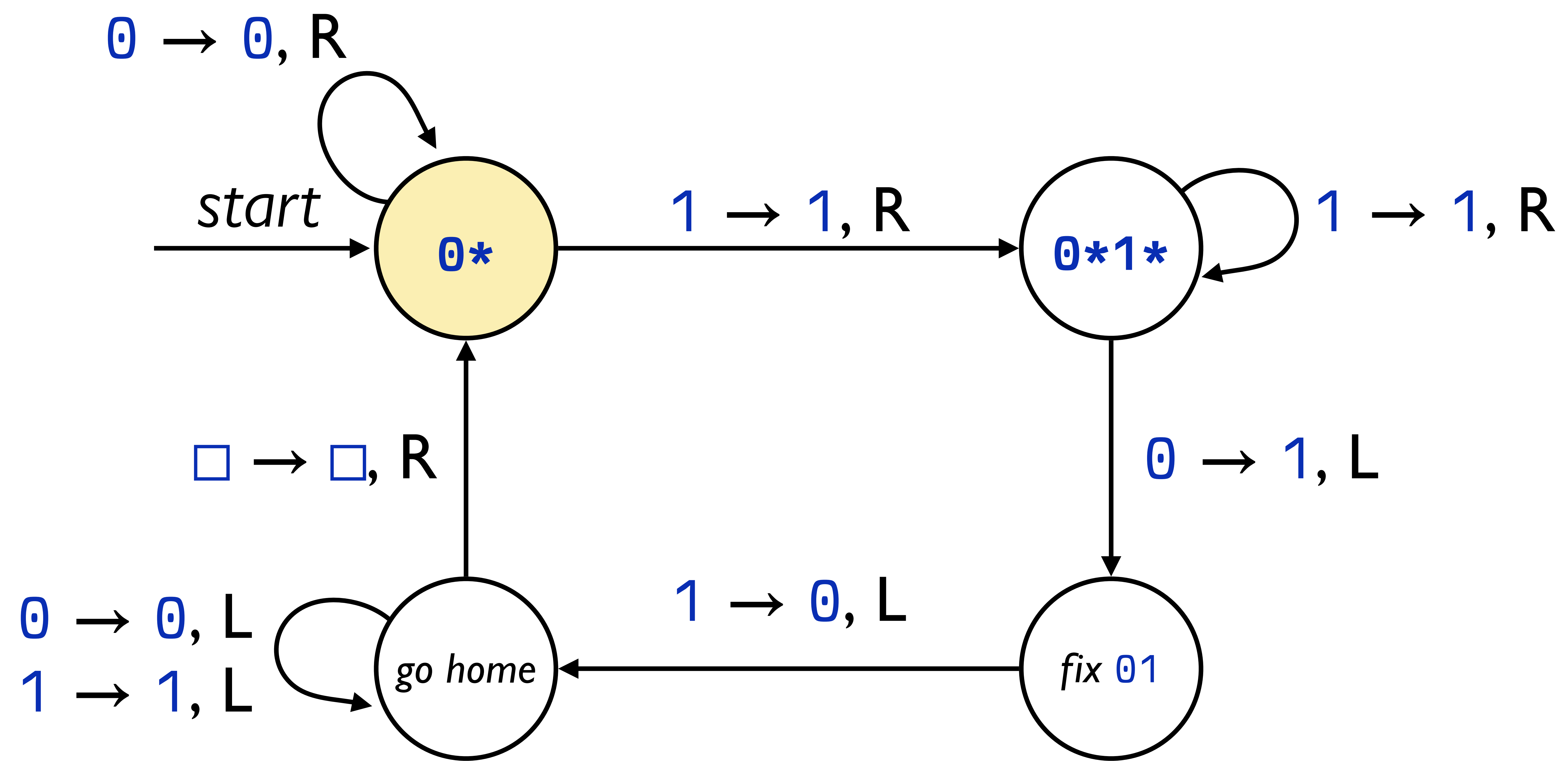
0 → 0, L
1 → 1, L

go home

1 → 0, L

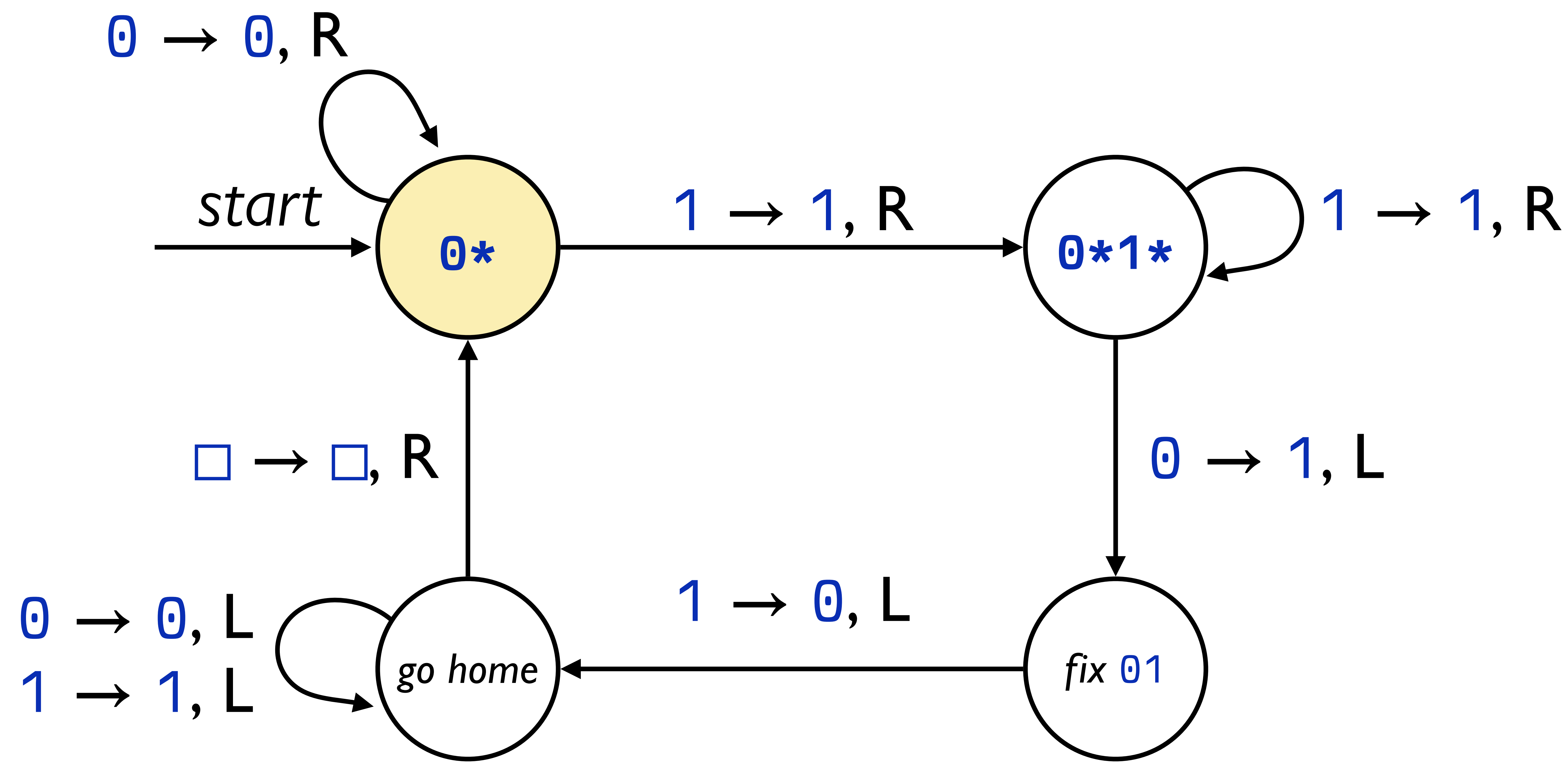
fix 01

														0	0	0	0	1	1														

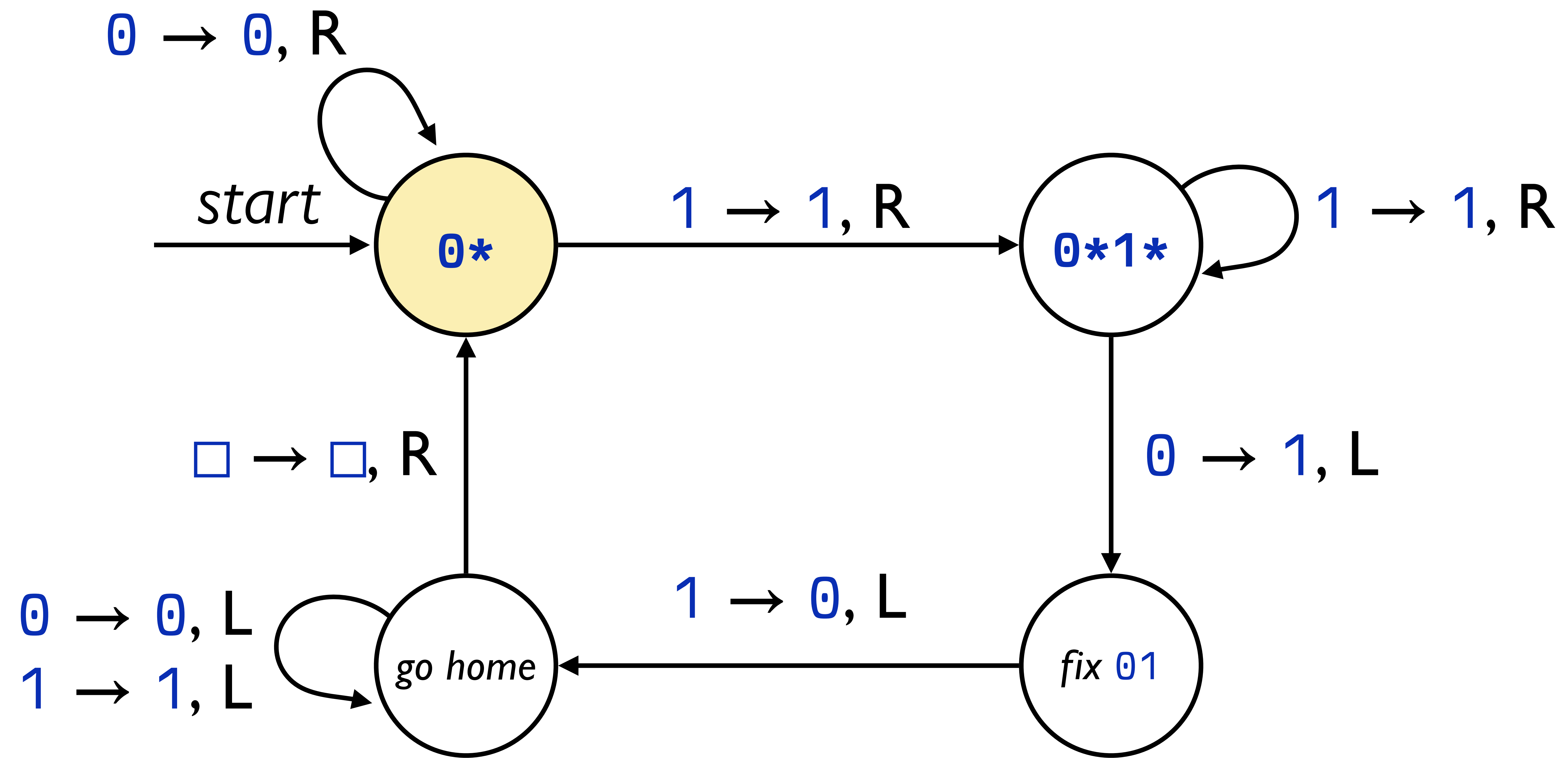

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

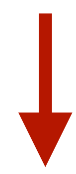
0 0 0 0 1 1

0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01
0 0 0 0 1 1


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 0 0 1 1


0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 0 0 0 1 1

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

0 → 1, L

□ → □, R

0 → 0, L
1 → 1, L

go home

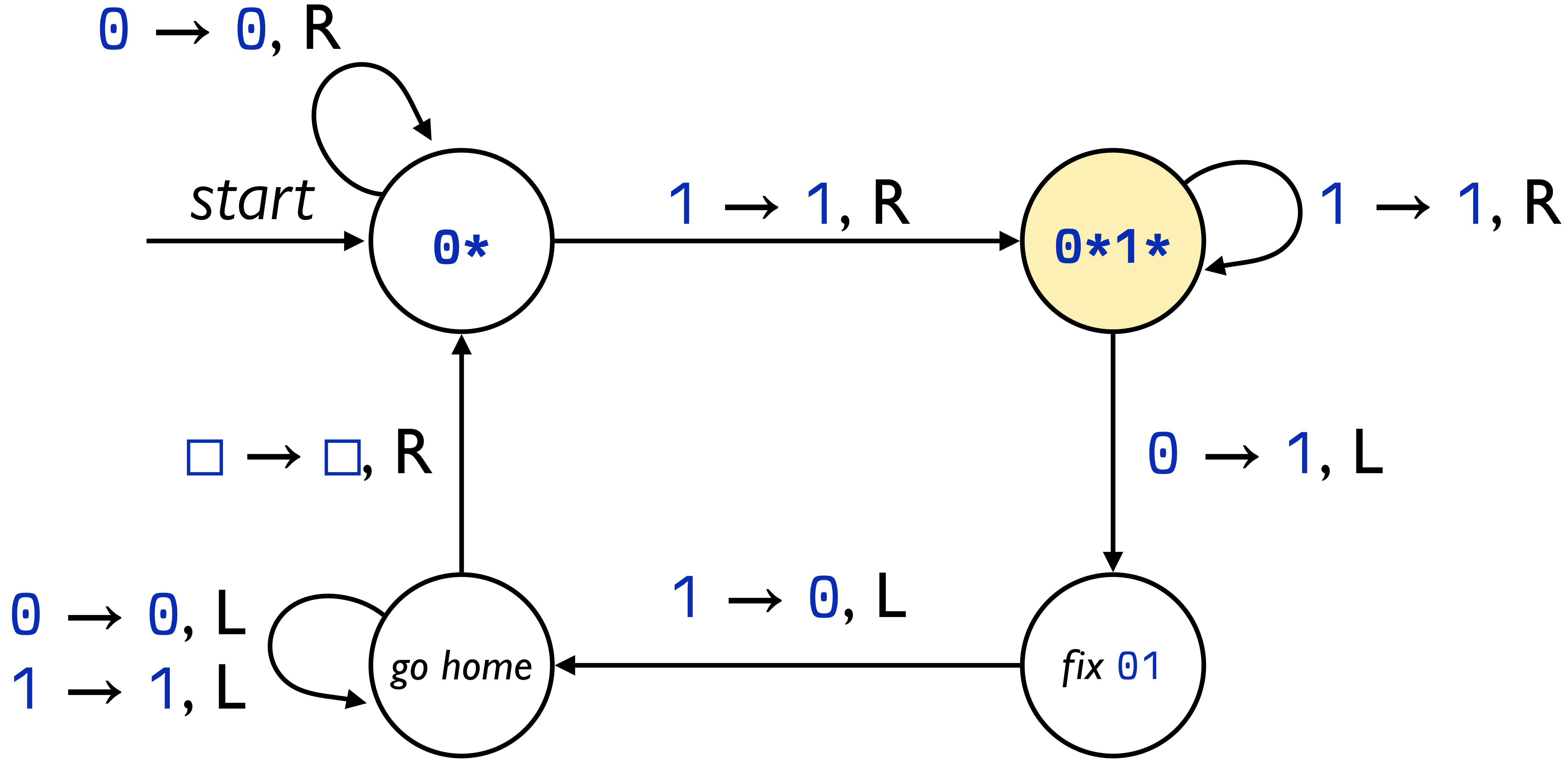
1 → 0, L

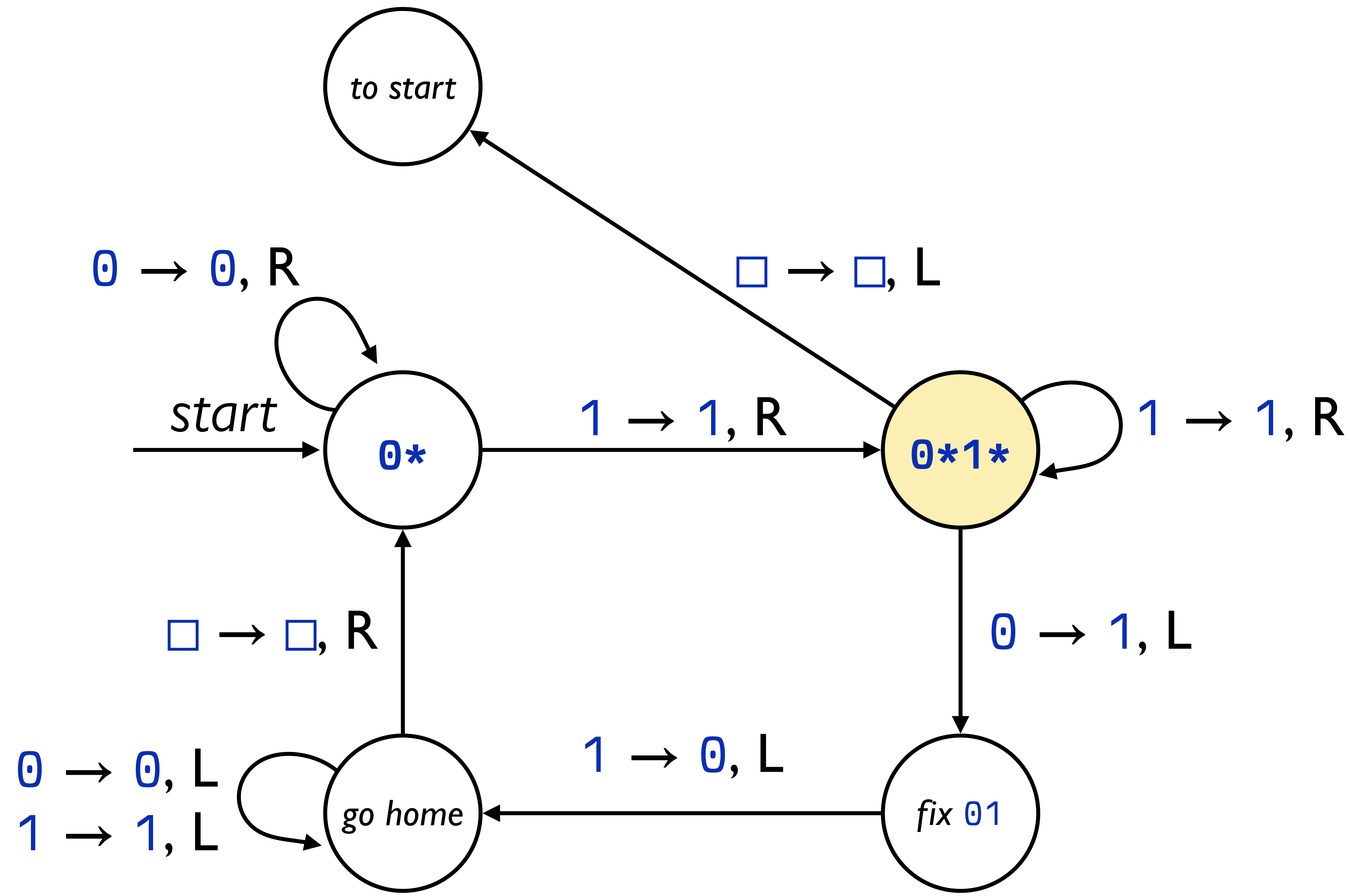
fix 01

														0	0	0	0	1	1													

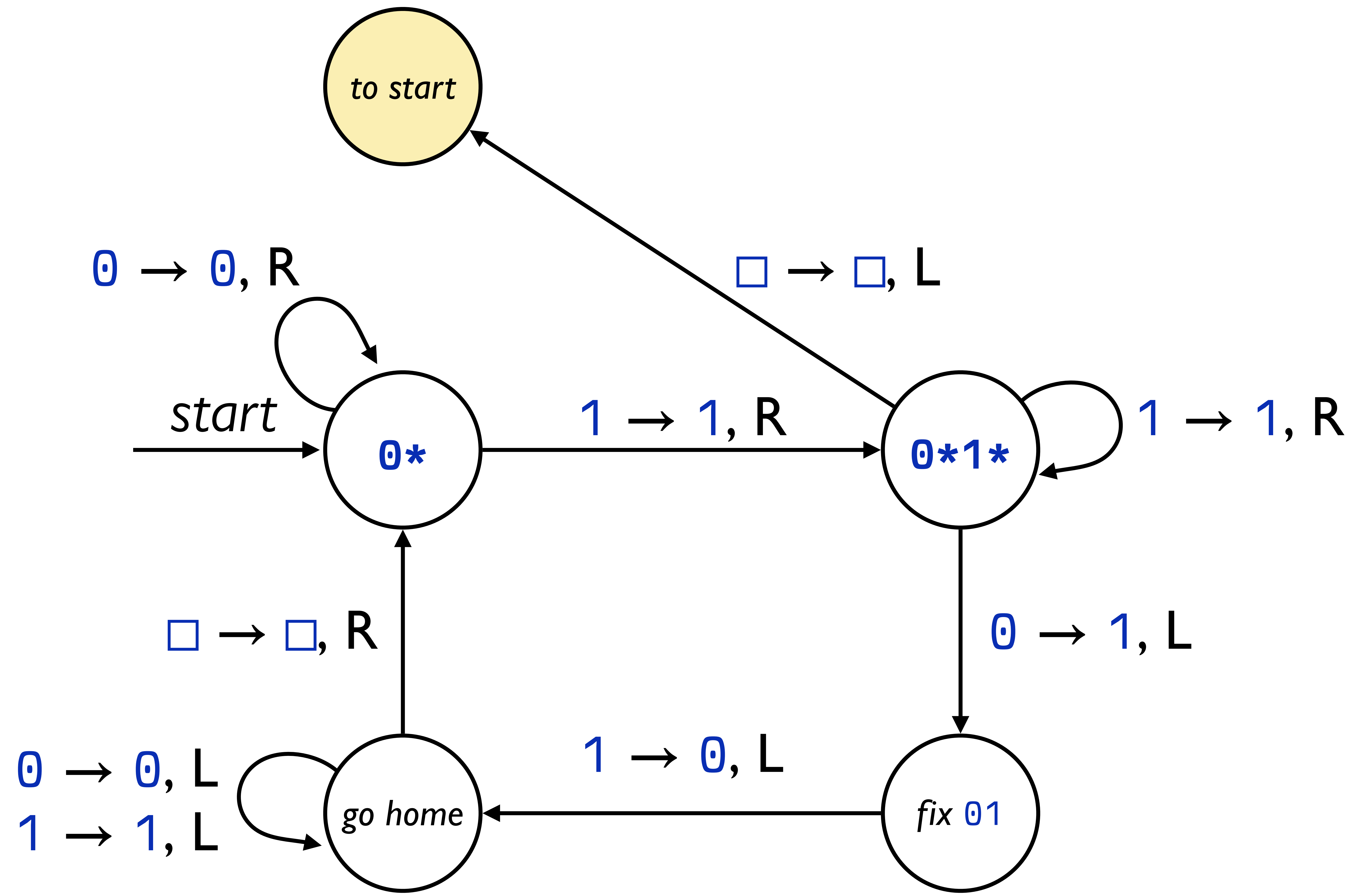
0 → 0, R
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01
0 0 0 0 1 1

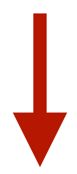
Our ultimate goal here was to sort everything so we could hand it off for the machine to check for 0^n1^n. So, let's rewind the tape head back to the start.




to start
0 → 0, R
start
0*
1 → 1, R
0*1*
□ → □, L
1 → 1, R
0 → 1, L
□ → □, R
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01

0 0 0 0 1 1


to start
0 → 0, R
□ → □, L
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
go home
fix 01



0 0 0 0 1 1

0 → 0, L
1 → 1, L

to start

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

0 → 0, L
1 → 1, L

to start

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

															0	0	0	0	1	1													

0 → 0, L
1 → 1, L

to start

0 → 0, R

start

0*

1 → 1, R

□ → □, L

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	0	1	1														

0 → 0, L
1 → 1, L

to start

□ → □, L

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

														0	0	0	0	1	1														

0 → 0, L
1 → 1, L

to start

□ → □, L

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 → 0, L
1 → 1, L

to start

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	0	1	1														

0 → 0, L
1 → 1, L

to start

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 0 0 0 1 1

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

| | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | | | | | | |

0 → 0, L
1 → 1, L

□ → □, R

to start

start
0^n1^n

□ → □, L

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

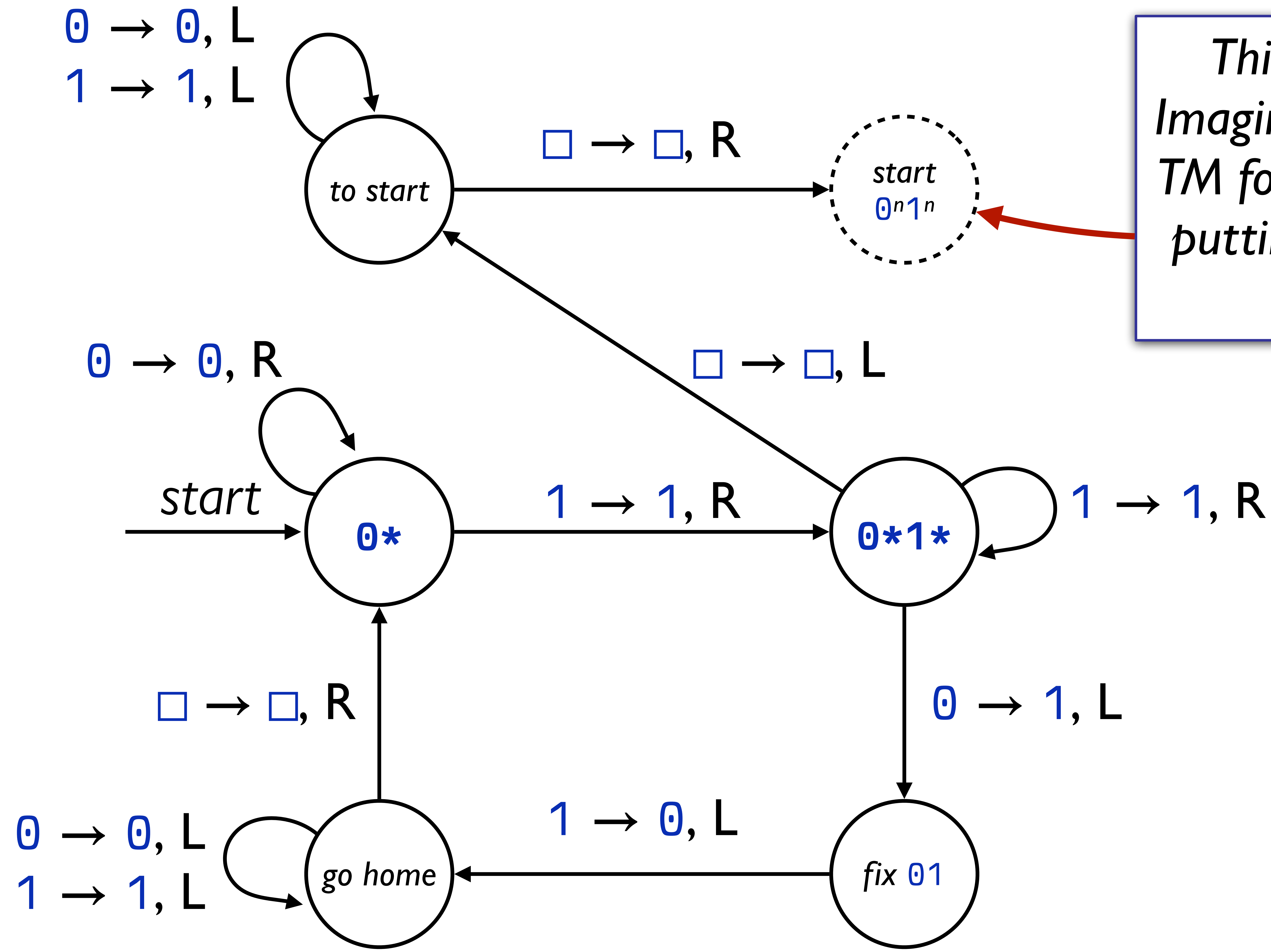
1 → 0, L

0 → 0, L
1 → 1, L

go home

fix 01

0 0 0 0 1 1



This is just a placeholder. Imagine snapping in the entire TM for 0^n1^n into this diagram, putting the start state in the dashed area.

0 → 0, L
1 → 1, L

to start

□ → □, R

start 0^n1^n

□ → □, L

0 → 0, R

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

														0	0	0	0																

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

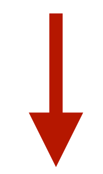
0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01



0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

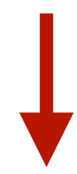
0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01



0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

start

0*

1 → 1, R

□ → □, L

0*1*

1 → 1, R

□ → □, R

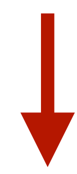
0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01



0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

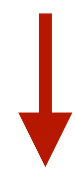
0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01



0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01

0 → 0, L
1 → 1, L

to start

□ → □, R

start
0^n1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

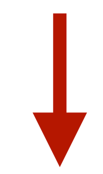
0 → 1, L

0 → 0, L
1 → 1, L

go home

1 → 0, L

fix 01



0 → 0, L
1 → 1, L

to start

□ → □, R

start 0^n1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

0 → 1, L

0 → 0, L
1 → 1, L

1 → 0, L

go home

fix 01

0 → 0, L
1 → 1, L

to start

□ → □, R

start 0^{n}1^n

0 → 0, R

□ → □, L

□ → □, L

start

0*

1 → 1, R

0*1*

1 → 1, R

□ → □, R

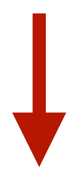
0 → 1, L

0 → 0, L
1 → 1, L

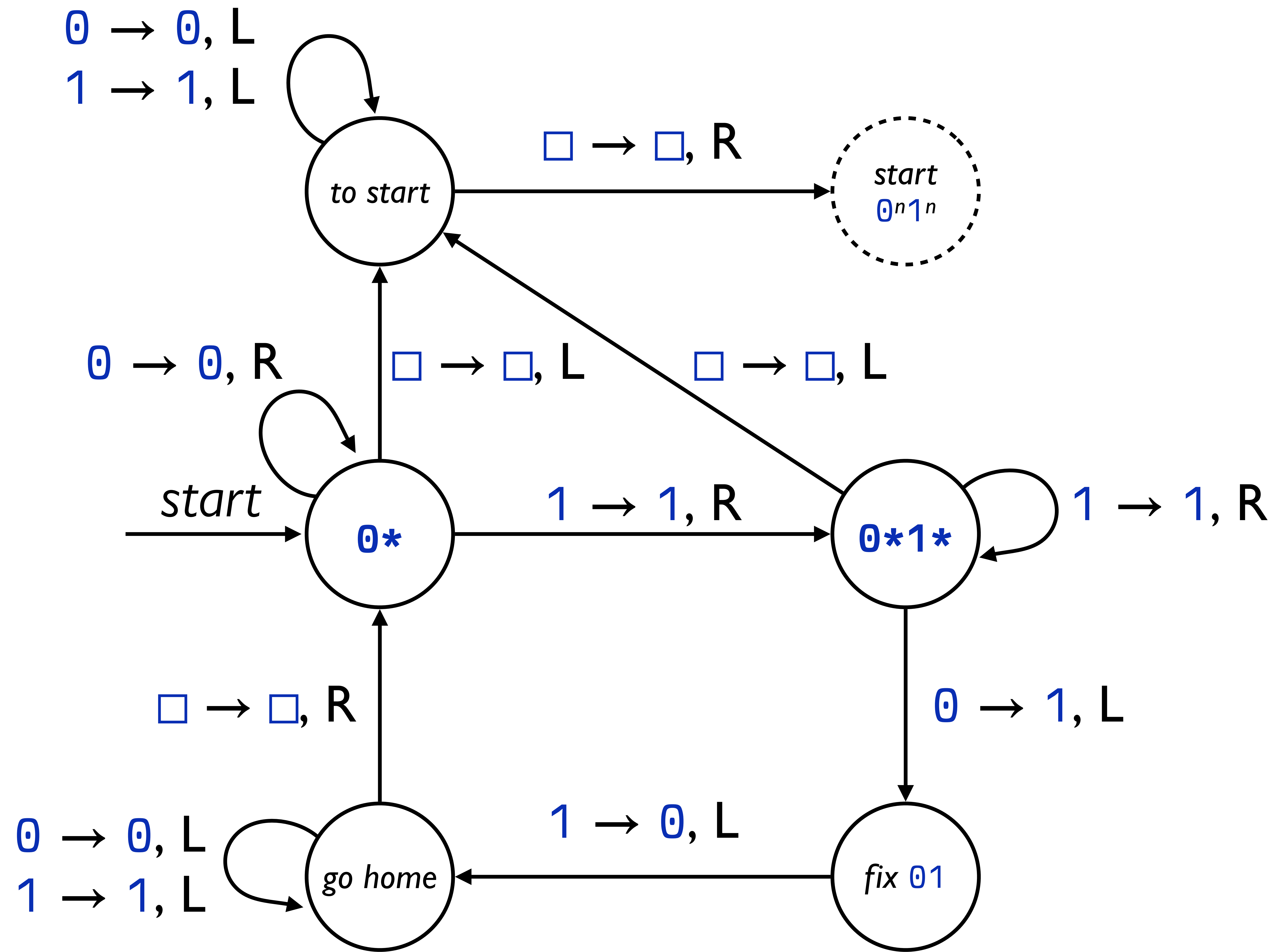
1 → 0, L

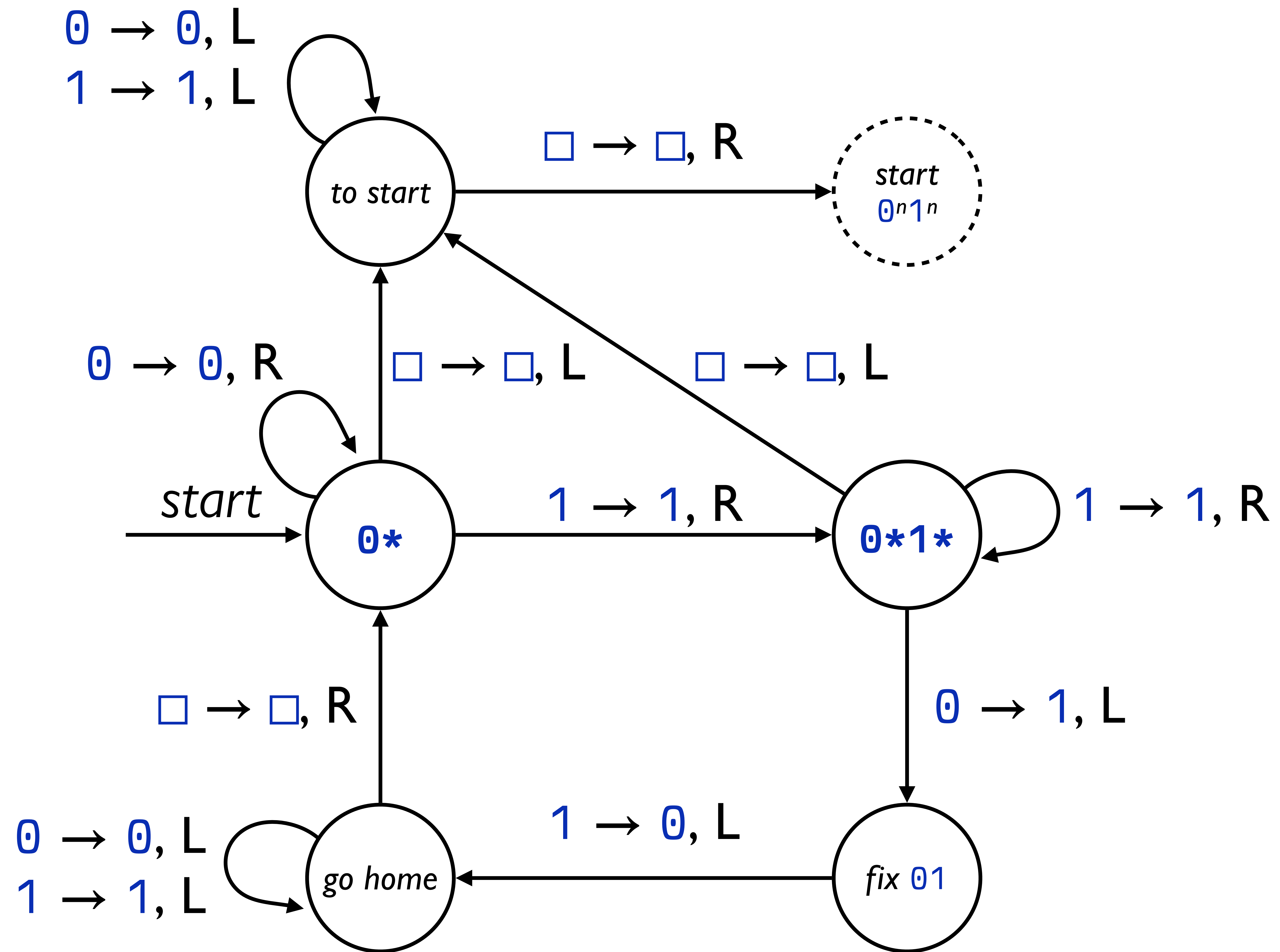
go home

fix 01

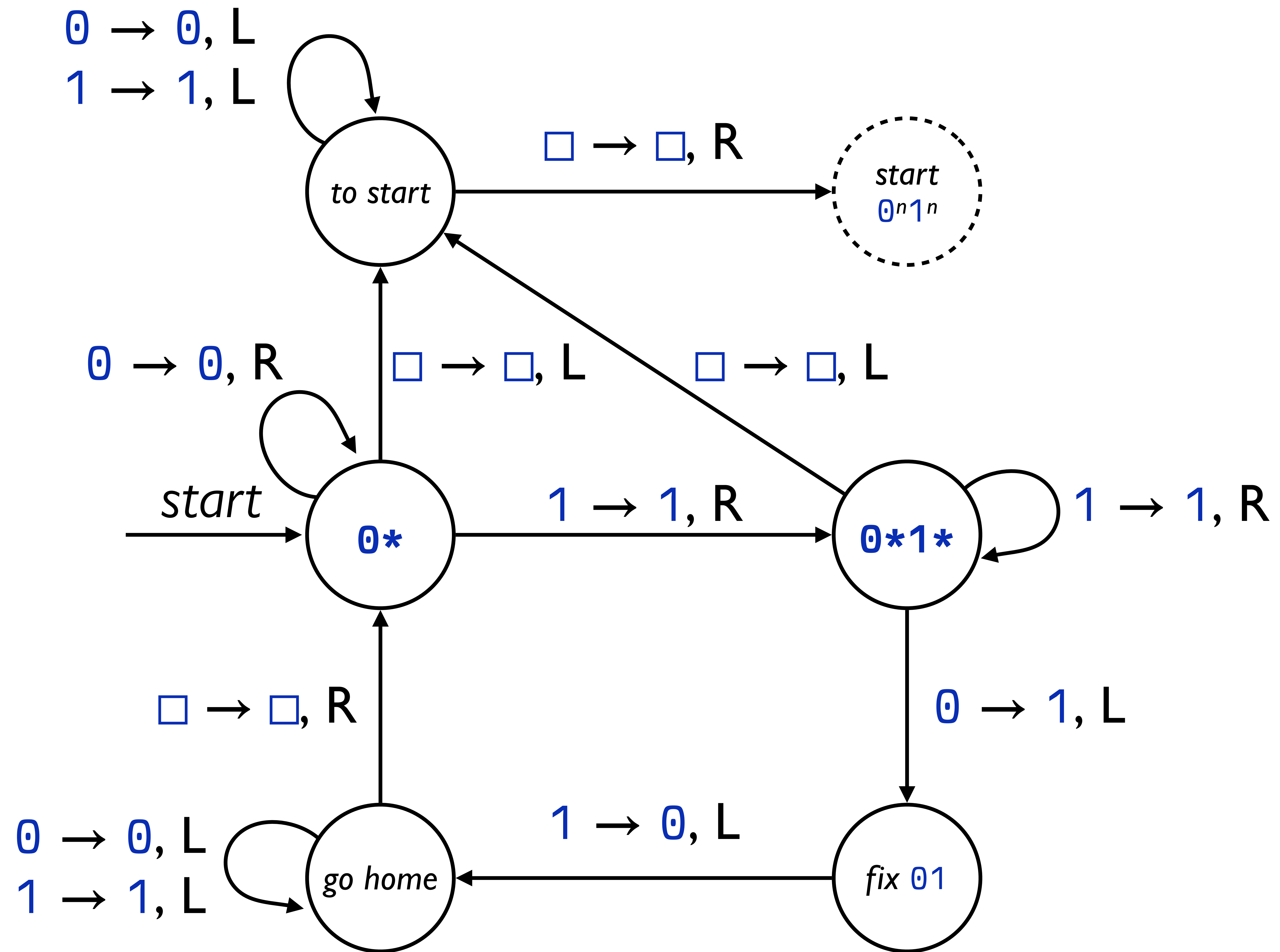


0 → 0, L
1 → 1, L
to start
□ → □, R
start
0^n1^n
0 → 0, R
□ → □, L
□ → □, L
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01
0 0 0 0

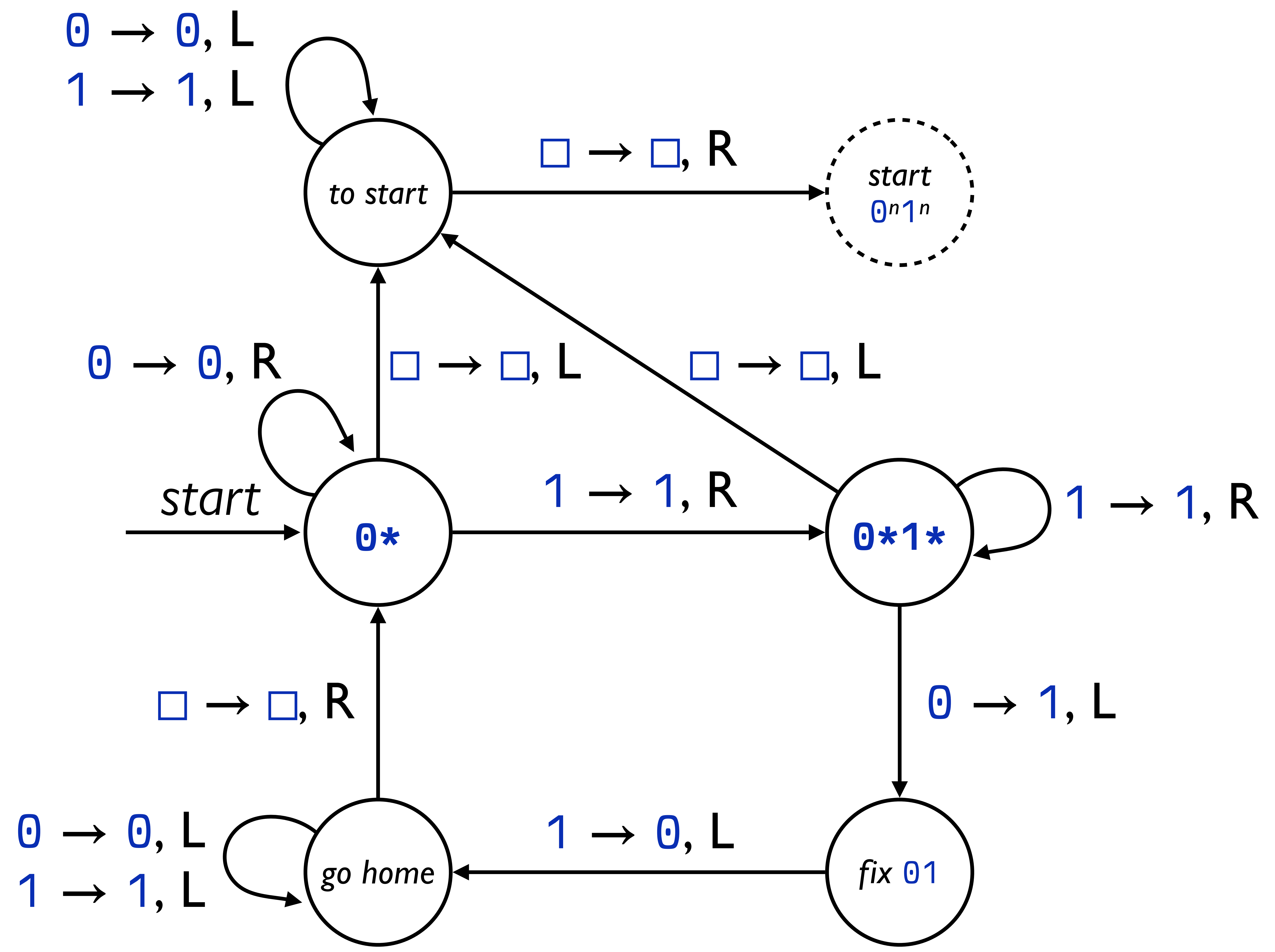

0 → 0, L
1 → 1, L
to start
□ → □, R
start
0ⁿ1ⁿ
0 → 0, R
□ → □, L
□ → □, L
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



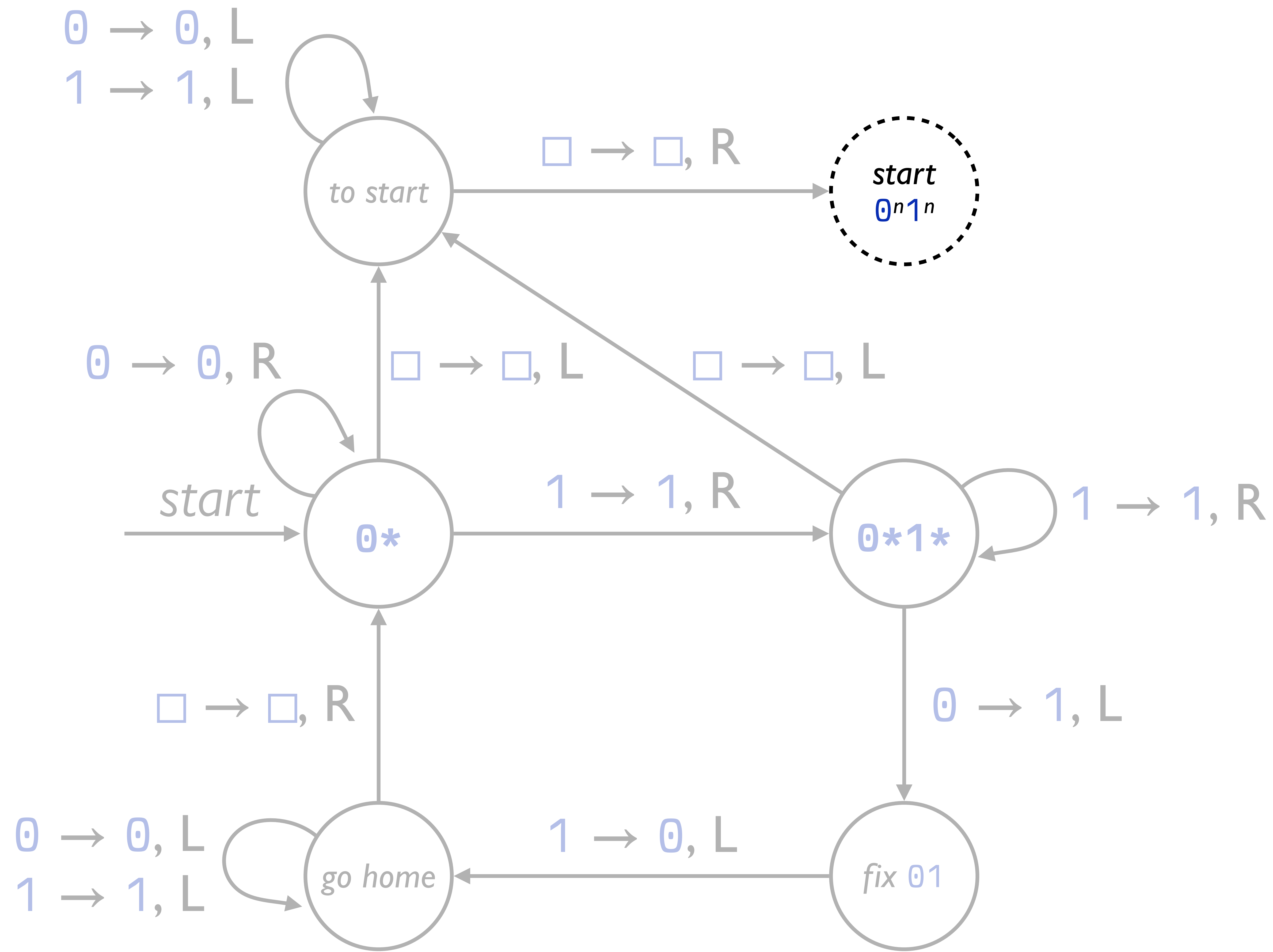
This TM will sort any sequence of 0s and 1s, though it might take a while!



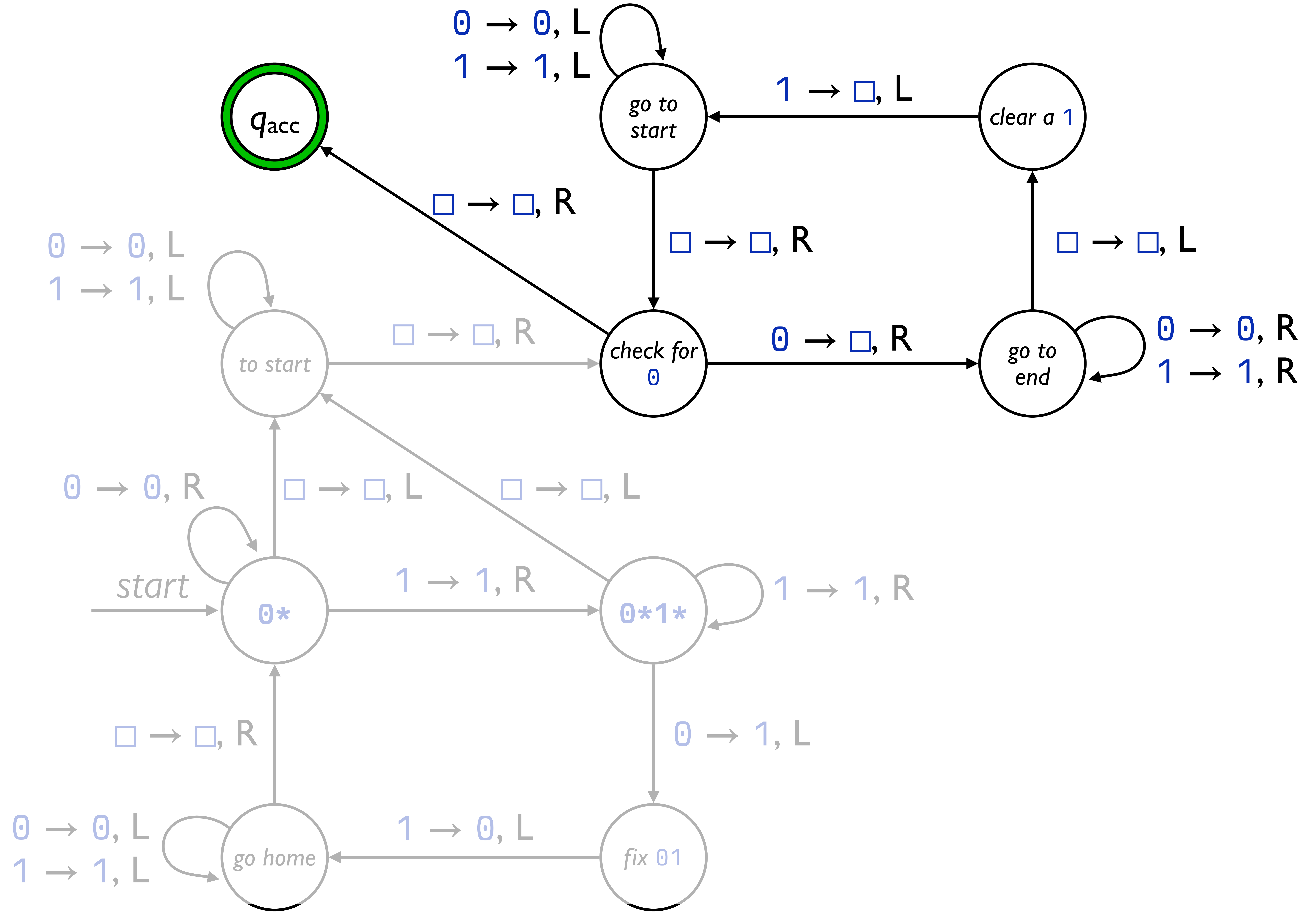
How would we assemble the full Turing machine?



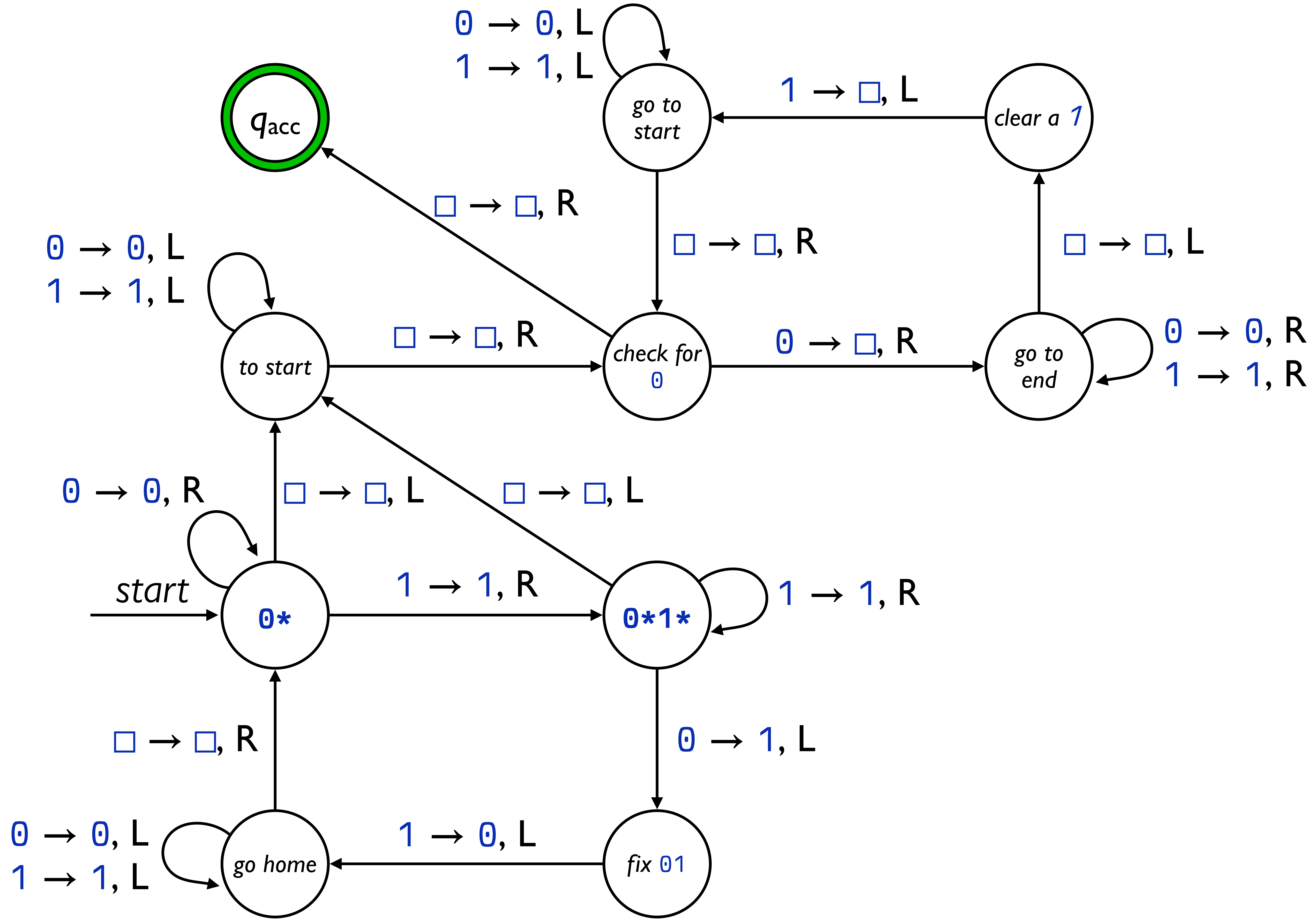
0 → 0, L
1 → 1, L
to start
□ → □, R
start
0ⁿ1ⁿ
0 → 0, R
□ → □, L
□ → □, L
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



0 → 0, L
1 → 1, L
to start
□ → □, R
start
0n1n
0 → 0, R
□ → □, L
□ → □, L
start
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
go home
1 → 0, L
fix 01



qacc
go to start
clear a 1
check for 0
go to end
to start
0*
0*1*
go home
fix 01
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
□ → □, L
□ → □, L
1 → 1, R
0 → 1, L
1 → 0, L



qacc
go to start
clear a 1
to start
check for 0
go to end
start
0*
0*1*
go home
fix 01
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, R
□ → □, L
0 → 0, L
1 → 1, L
□ → □, R
0 → □, R
0 → 0, R
1 → 1, R
0 → 0, R
□ → □, L
□ → □, L
1 → 1, R
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
1 → 0, L

A *TM subroutine* is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.

TM subroutines let us compose larger TMs out of smaller TMs, just as you'd use a variety of helper functions to write a complicated program.

Here, we saw a TM subroutine to sort a sequence of `0`s and `1`s into ascending order.

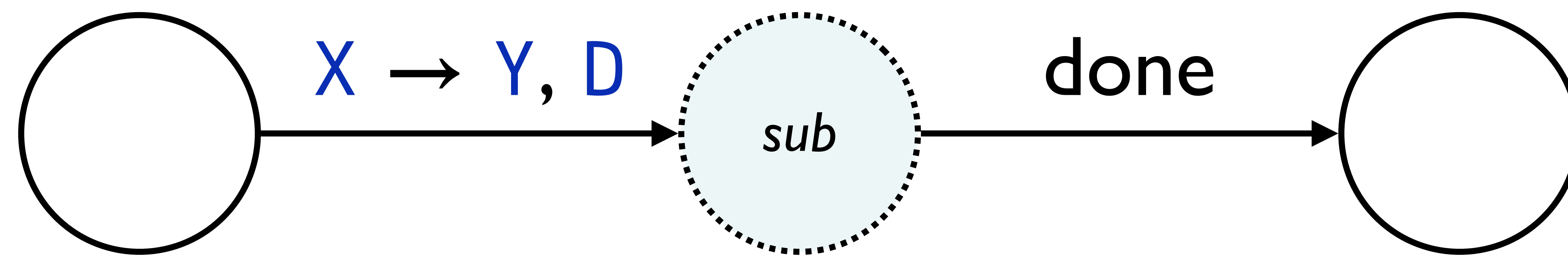
Turing machine subroutines

Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.

The idea is that you’d then replace the dashed “done” state with the next piece of the construction.

Using subroutines

Once you've built a subroutine, you can wire it into another Turing machine with something that, schematically, looks like this:



Intuitively, this corresponds to transitioning to the start state of the subroutine, then replacing the "done" state of the subroutine with the state at the end of the transition.

Turing machine subroutines

Note that subroutines are not a special part of the model for Turing machines.

Rather, they are a convenient way to *think* about building a single complex Turing machine out of simpler pieces.

Next time

The Church–Turing Thesis

Just how powerful *are* Turing machines?

R and **RE** languages

What does it mean to solve a problem?

Acknowledgments

This lecture incorporates material from:

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*