CMPU 240 · Theory of Computation

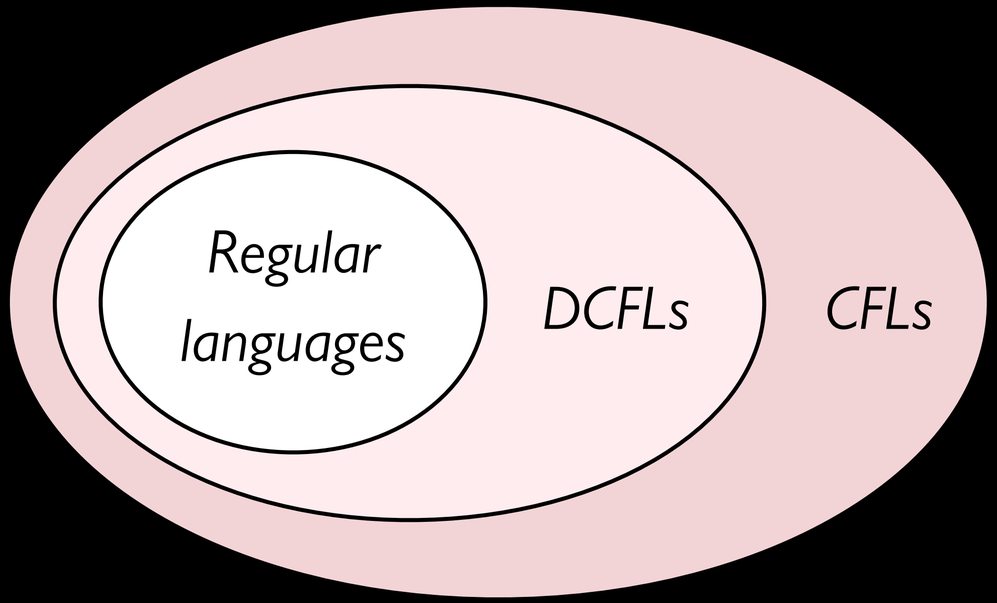
Turing Machines

14 April 2026

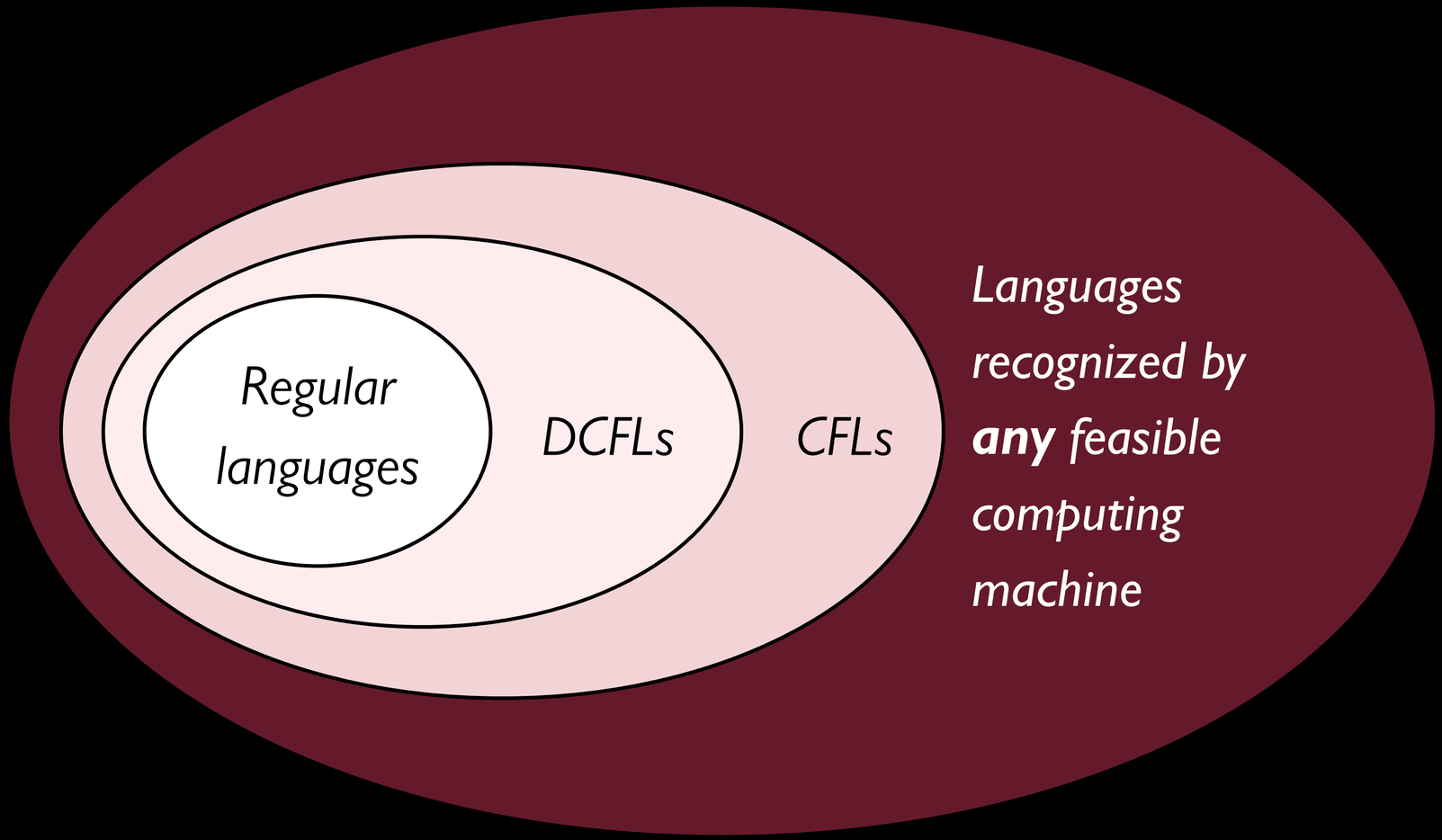
What problems can we solve with a computer?

Finite automata represent computers with bounded memory. They recognize the *regular languages*.

Pushdown automata represent computers with a limited form of unbounded memory. They recognize the *context-free languages*.



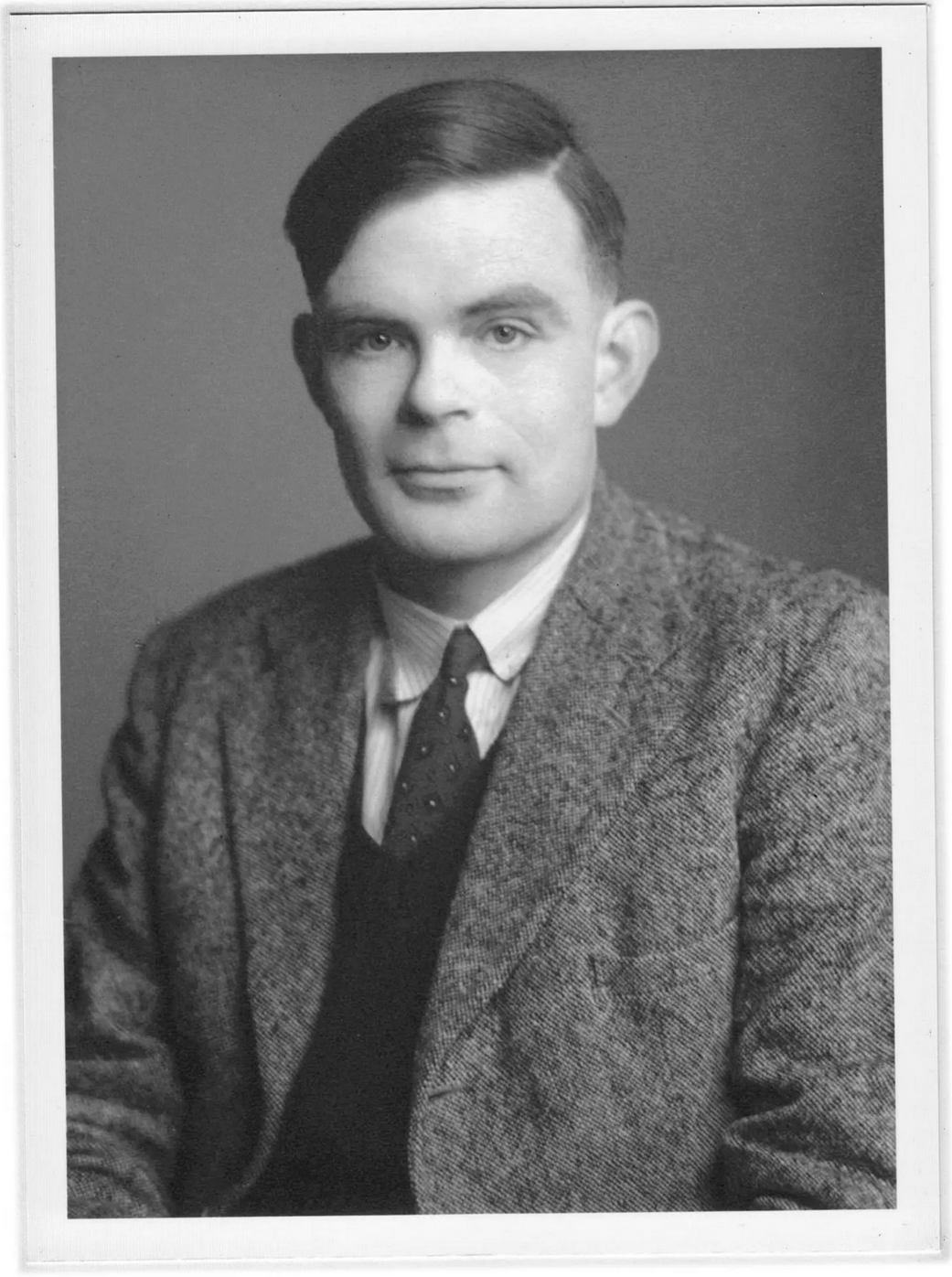
All languages


Regular languages
DCFLs
CFLs
Languages recognized by **any** feasible computing machine
All languages

To talk about what languages could be recognized by "any feasible computing machine", we need our final, most powerful model of computation.

A brief history lesson

Entscheidungsproblem = Decision problem



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM. A CORRECTION

By A. M. TURING.

In a paper entitled "On computable numbers, with an application to the Entscheidungsproblem"* the author gave a proof of the insolubility of the Entscheidungsproblem of the "engere Funktionenkalkül". This proof contained some formal errors† which will be corrected here: there are also some other statements in the same paper which should be modified, although they are not actually false as they stand.

The expression for Inst $\{q_i S_j S_k L q_l\}$ on p. 260 of the paper quoted should read

$$(x, y, x', y')\{\left(R_{S_j}(x, y) \,\&\, I(x, y) \,\&\, K_{q_i}(x) \,\&\, F(x, x') \,\&\, F(y', y)\right)$$

$$\rightarrow \left(I(x', y') \,\&\, R_{S_k}(x', y) \,\&\, K_{q_l}(x') \,\&\, F(y', z) \vee \left[\left(R_{S_0}(x, z) \rightarrow R_{S_0}(x', z)\right)\right.\right.$$

$$\left.\left. \,\&\, \left(R_{S_1}(x, z) \rightarrow R_{S_1}(x', z)\right) \,\&\, \ldots \,\&\, \left(R_{S_M}(x, z) \rightarrow R_{S_M}(x', z)\right)\right]\right)\},$$

At the same time, *Alonzo Church* published similar ideas and results – the λ calculus.

However, the Turing model has become the standard in theoretical computer science

Turing’s idea

$$
\begin{array}{r}
271828182845 90 \\
+314159265358 97 \\
\hline
\end{array}
$$

$$
\begin{array}{r}
2718281828459\mathbf{0} \\
+3141592653589\mathbf{7} \\
\hline
\mathbf{7}
\end{array}
$$

$$
\begin{array}{r}
{}^{1} \\
2718281828459\,0 \\
+3141592653589\,7 \\
\hline
8\,7
\end{array}
$$

```
           1 1
 2718281828459 0
+3141592653589 7
----------------
             487
```

```
           1 1 1
  27182818284590
+31415926535897
----------------
            0487
```

```
    1 1 1 1
  2718281828 4590
+3141592653 5897
---------------
  2 0487
```

```
          1 1 1 1
  271828182845 90
+ 314159265358 97
-----------------
          820487
```

```
      1    1 1 1 1
  2718281828459 0
+3141592653589 7
-----------------
  4820487
```

```
      1     1 1 1 1
  2718281828459 0
+3141592653589 7
---------------
  4482048 7
```

```
    1   1   1 1 1 1
  2718281828459 0
+ 3141592653589 7
-----------------
  7448204 87
```

```
    1   1   1 1 1 1
  2718281828459 0
+3141592653589 7
---------------
 8744820487
```

					1		1		1	1	1	1		
	2	7	1	8	2	8	1	8	2	8	4	5	9	0
+	3	1	4	1	5	9	2	6	5	3	5	8	9	7
				9	8	7	4	4	8	2	0	4	8	7

```
     1 1 1111
 27182818284590
+31415926535897
---------------
   598744820487
```

```
    1   1   1 1 1 1
  271828182845 90
+314159265358 97
---------------
 859874482048 7
```

```
    1   1   1 1 1 1
  271828182845 90
+3141592653589 7
---------------
 5859874482048 7
```

$$
\begin{array}{r}
\,{}^{1}\,\,{}^{1}\,\,{}^{1}\,{}^{1}\,{}^{1}\,{}^{1} \\
271828182845 90 \\
+\,314159265358 97 \\
\hline
585987448204 87
\end{array}
$$

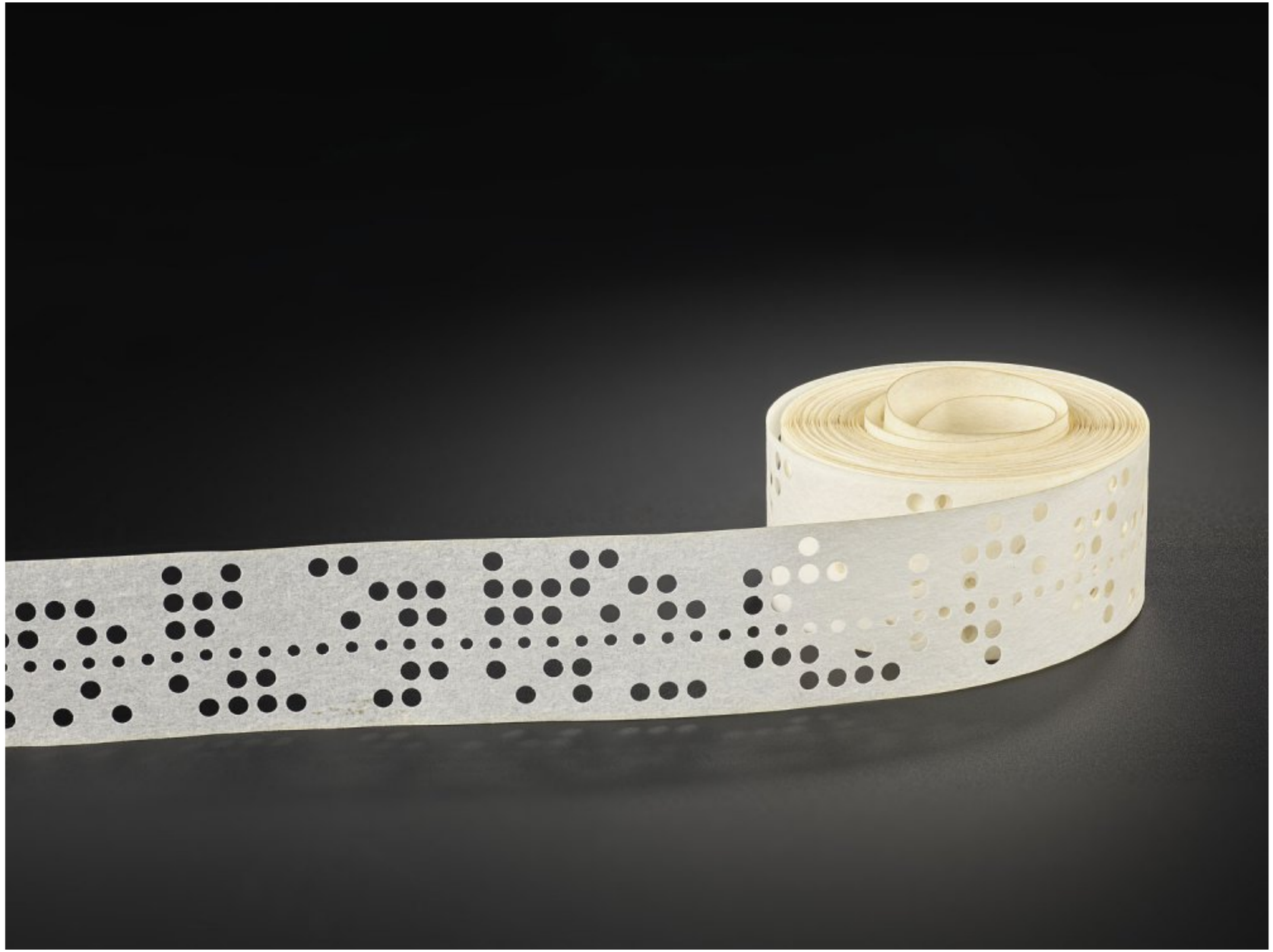
Key idea:

You might need huge amounts of scratch space to perform a calculation…

But at each point in the calculation you only need access to a small amount of that scratch space!

To provide his machines extra memory, Turing gave them access to an *infinite tape*, subdivided into *tape cells*.

A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.



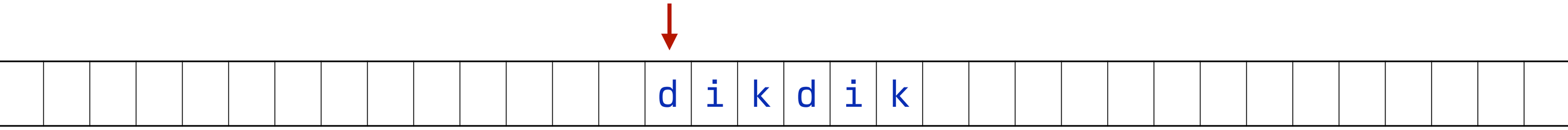
Punched paper computer tape (1955–65)

At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.



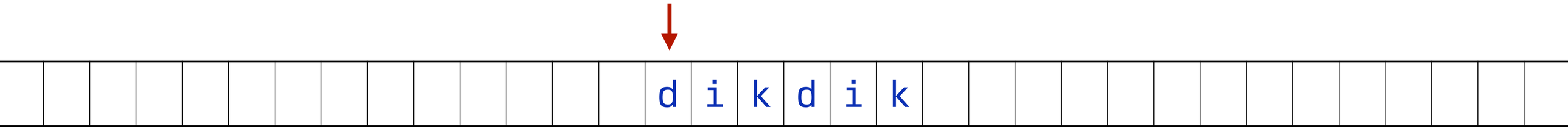
At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

A dikdik

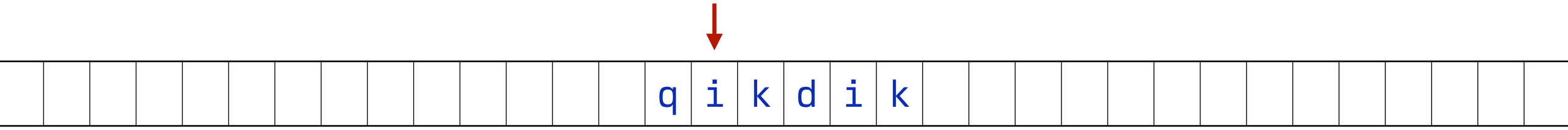


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

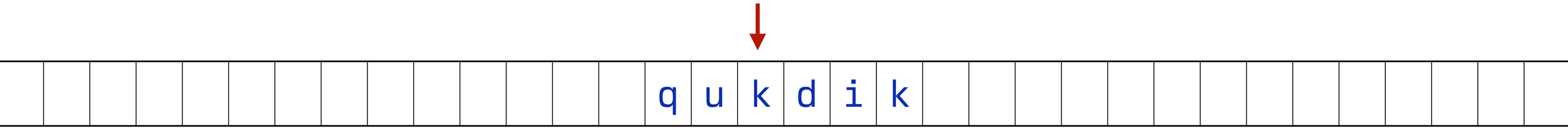


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

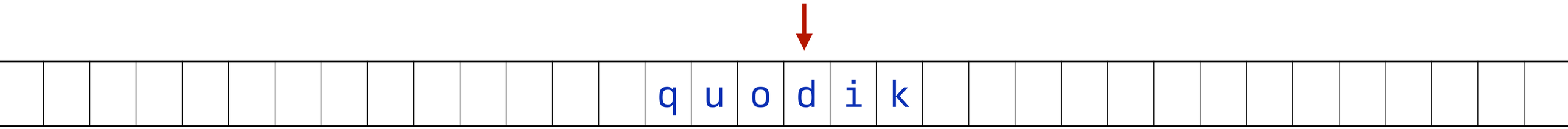


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

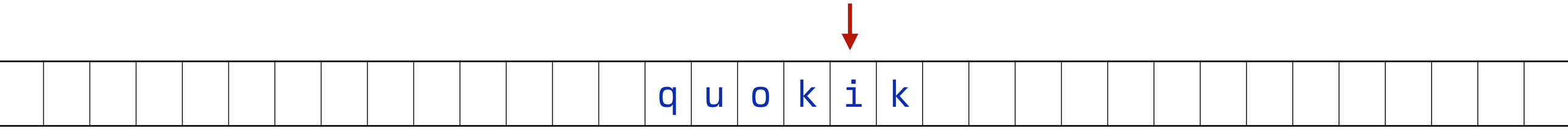


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

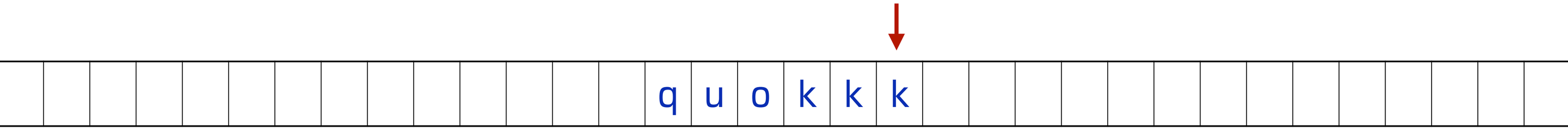


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

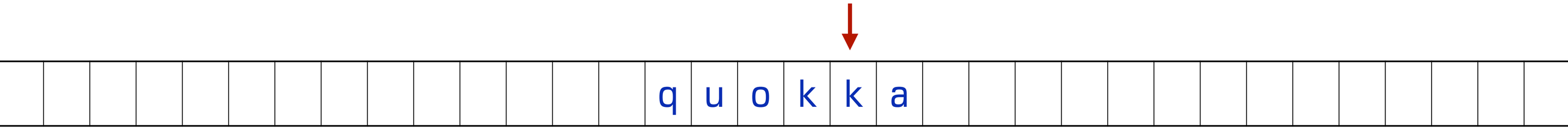


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.



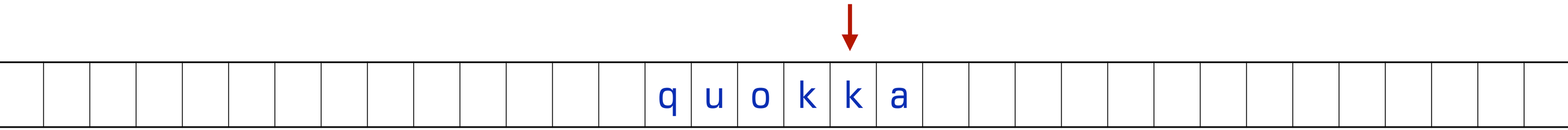
At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

A quokka

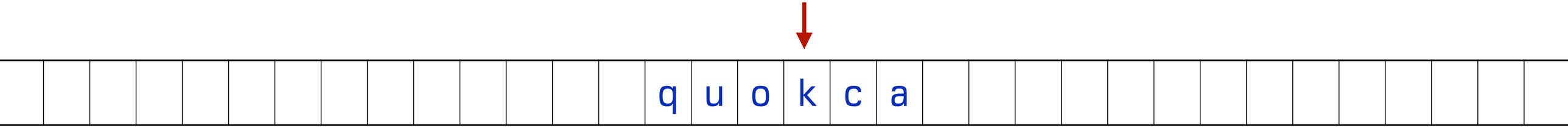


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

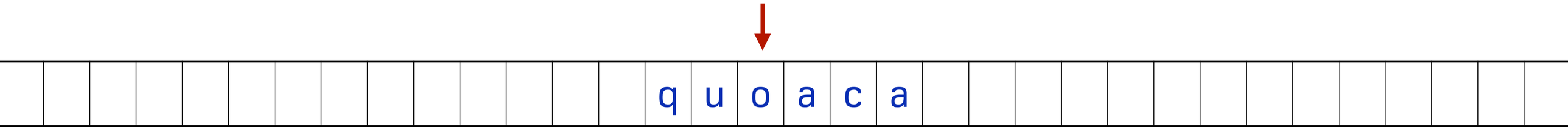


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

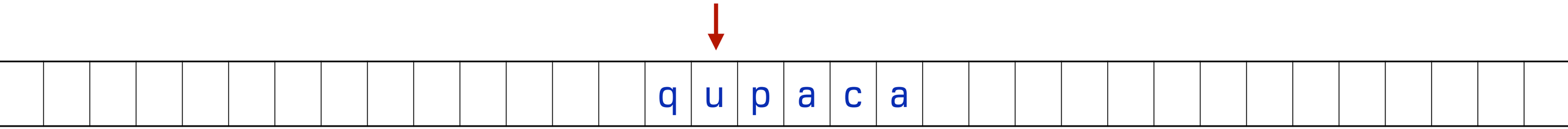


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

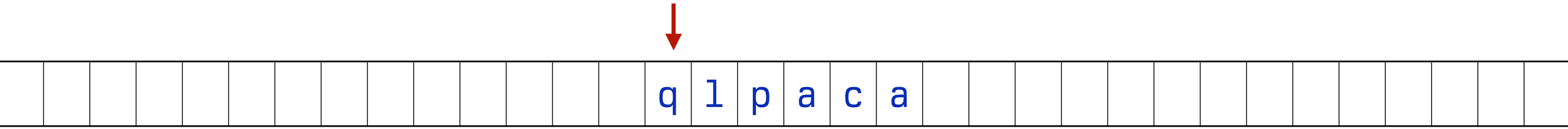


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.

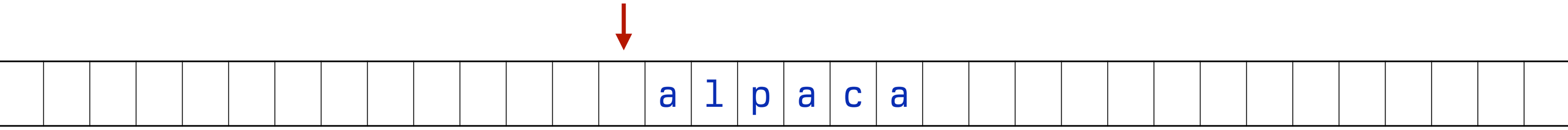


At every step, the Turing machine can

read the cell under the tape head,

change which symbol was written under the tape head, and

move its tape head to the left or to the right.



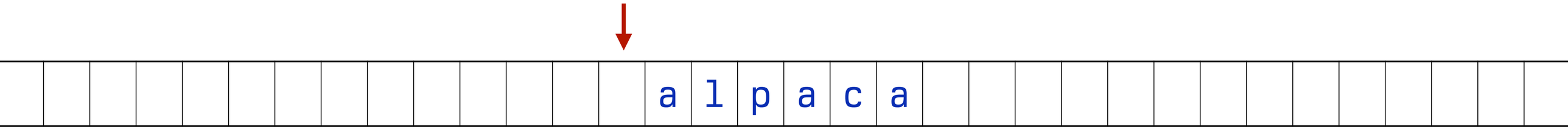
At every step, the Turing machine can

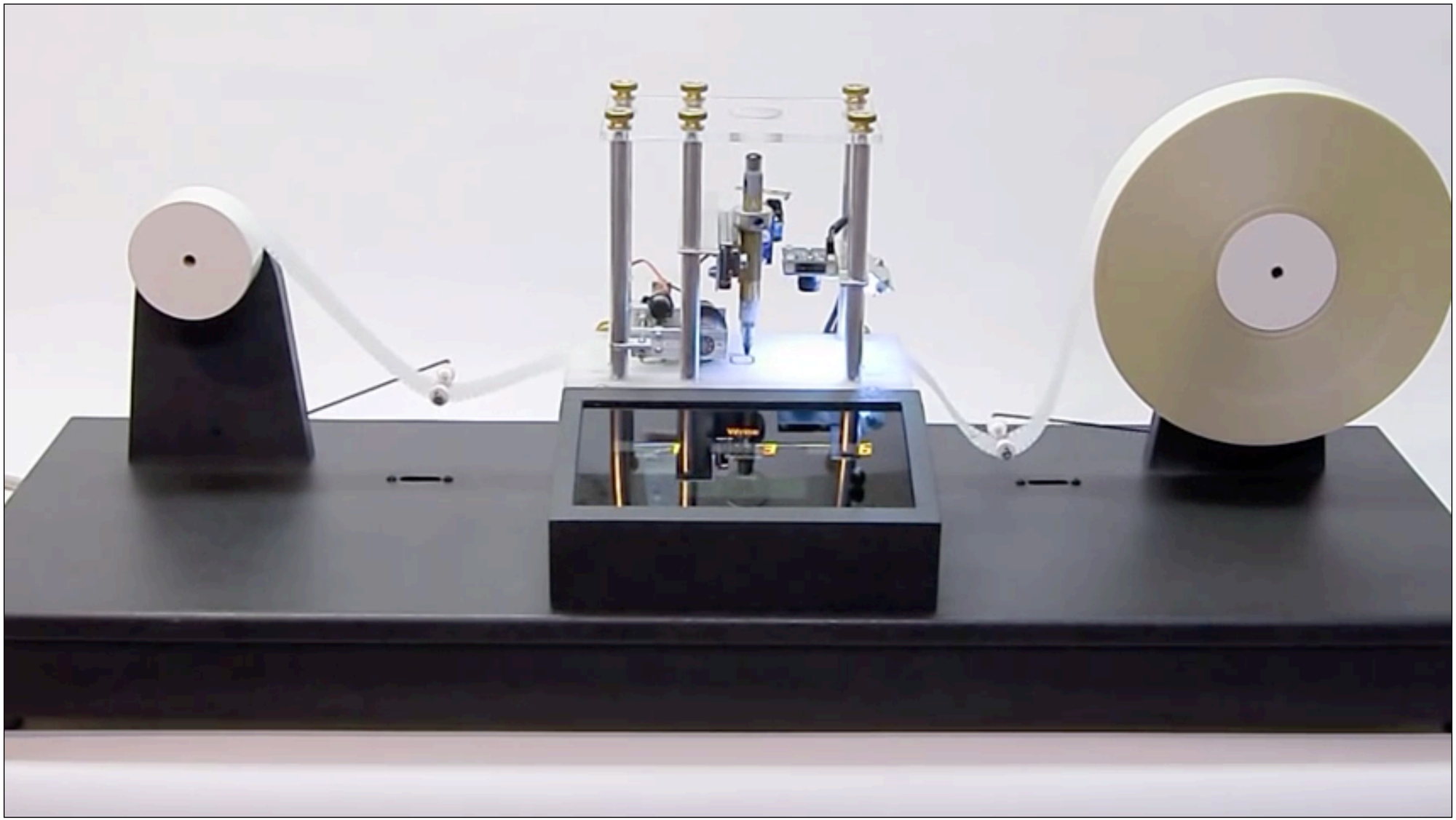
read the cell under the tape head,

change which symbol was written under the tape head, and

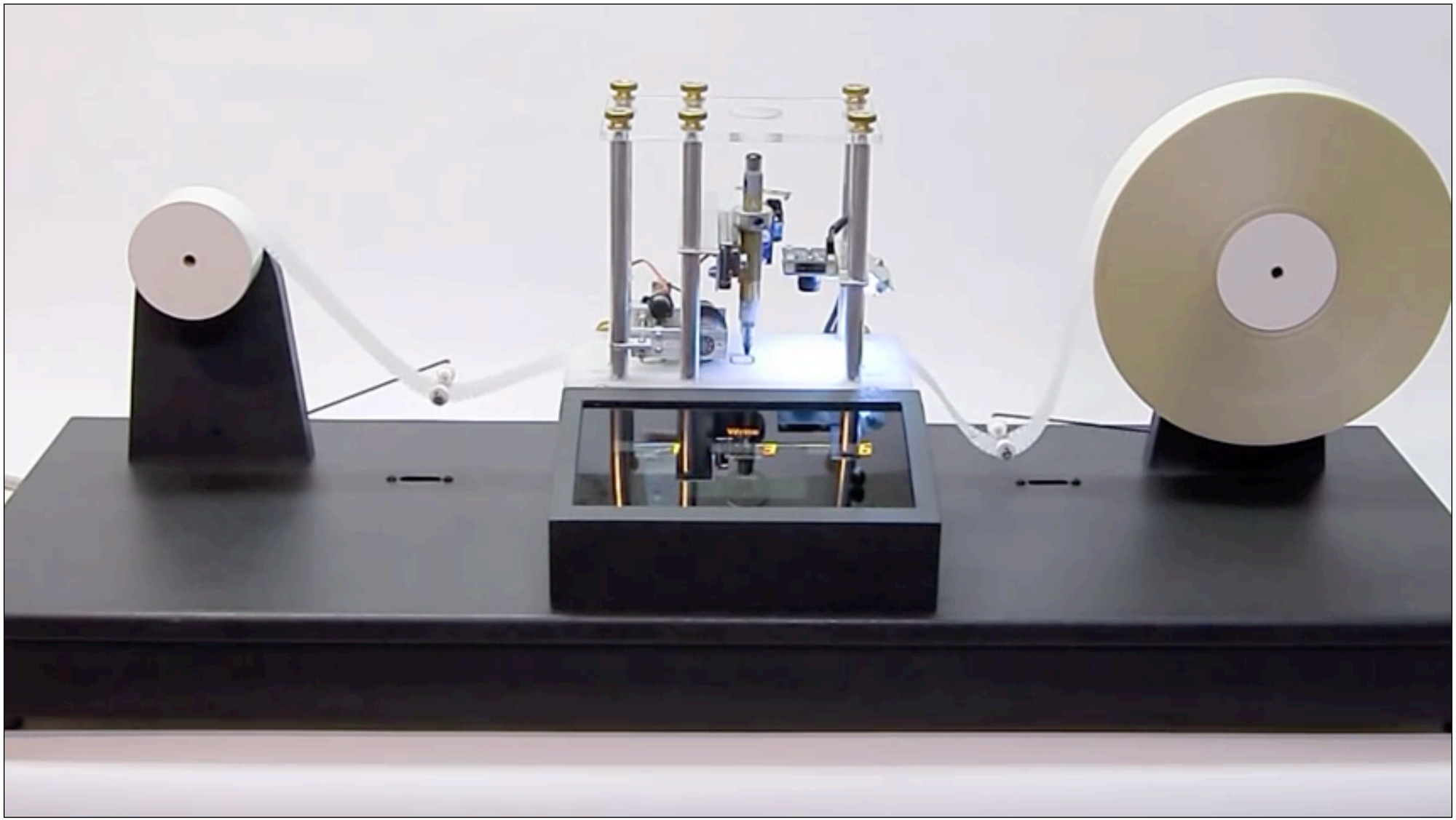
move its tape head to the left or to the right.

An alpaca

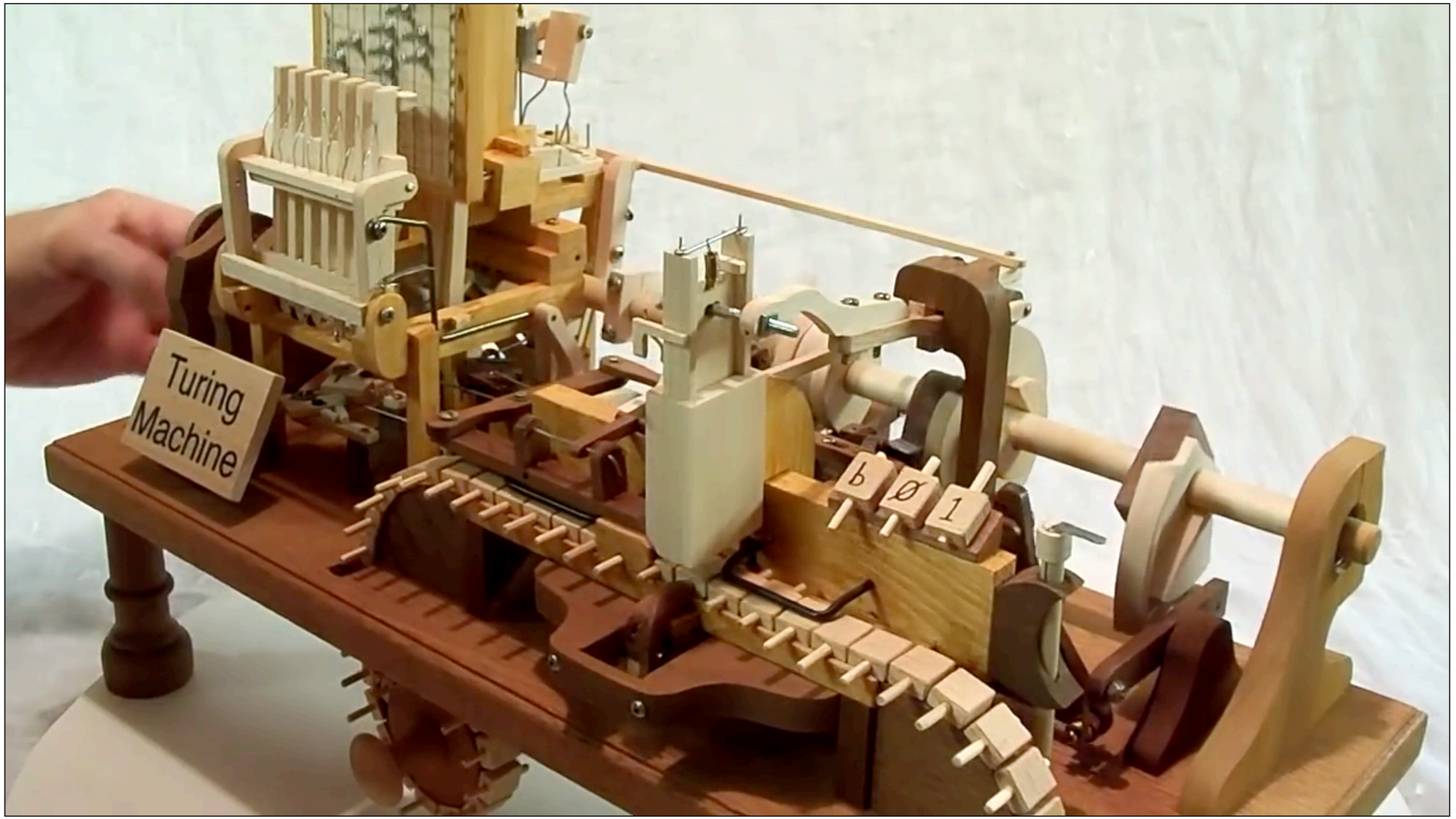




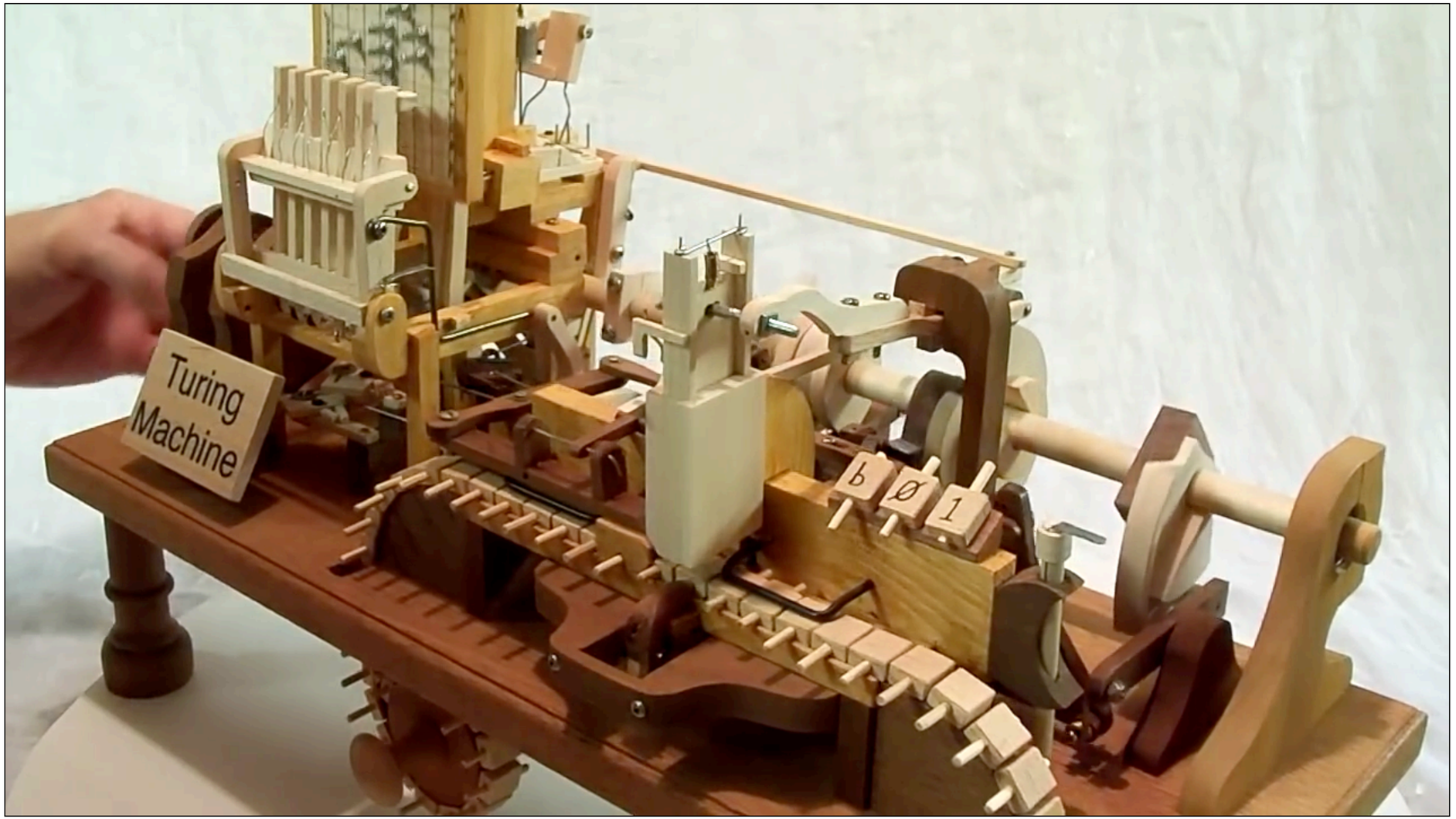
youtu.be/E3keLeMwfHY



youtu.be/E3keLeMwfHY

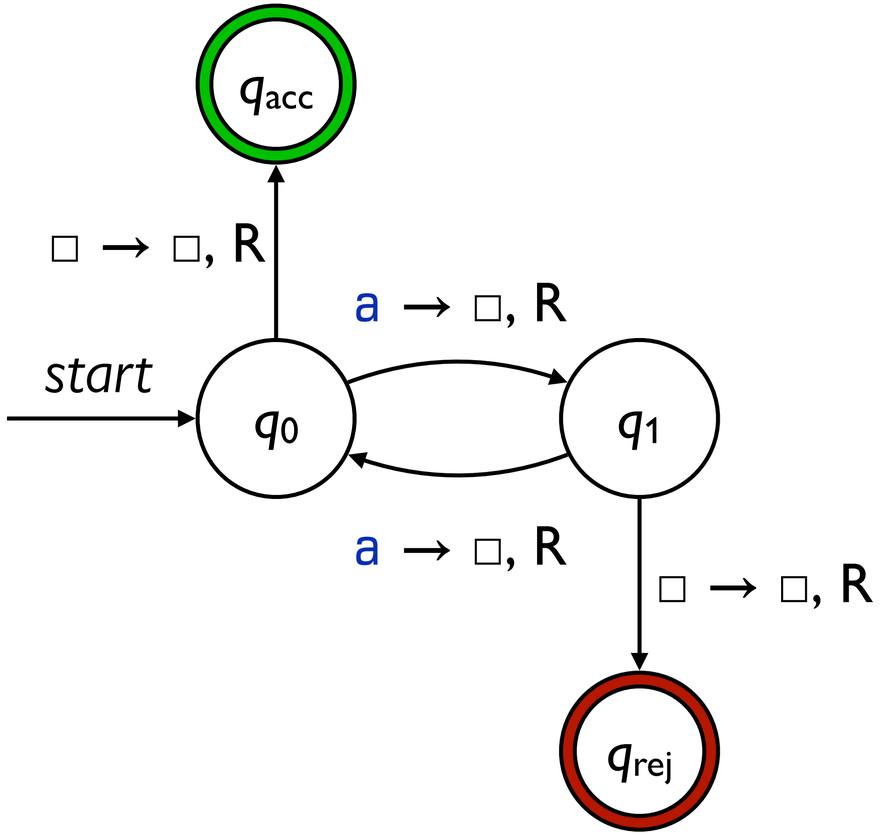


youtu.be/vo8izCKHiF0

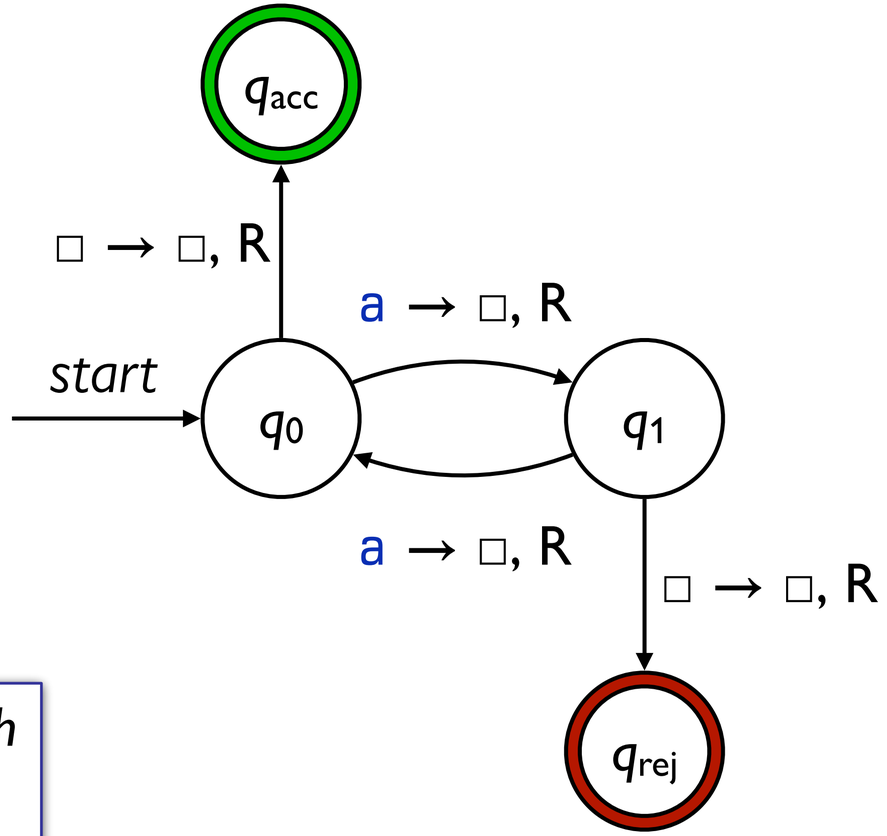


youtu.be/vo8izCKHiF0

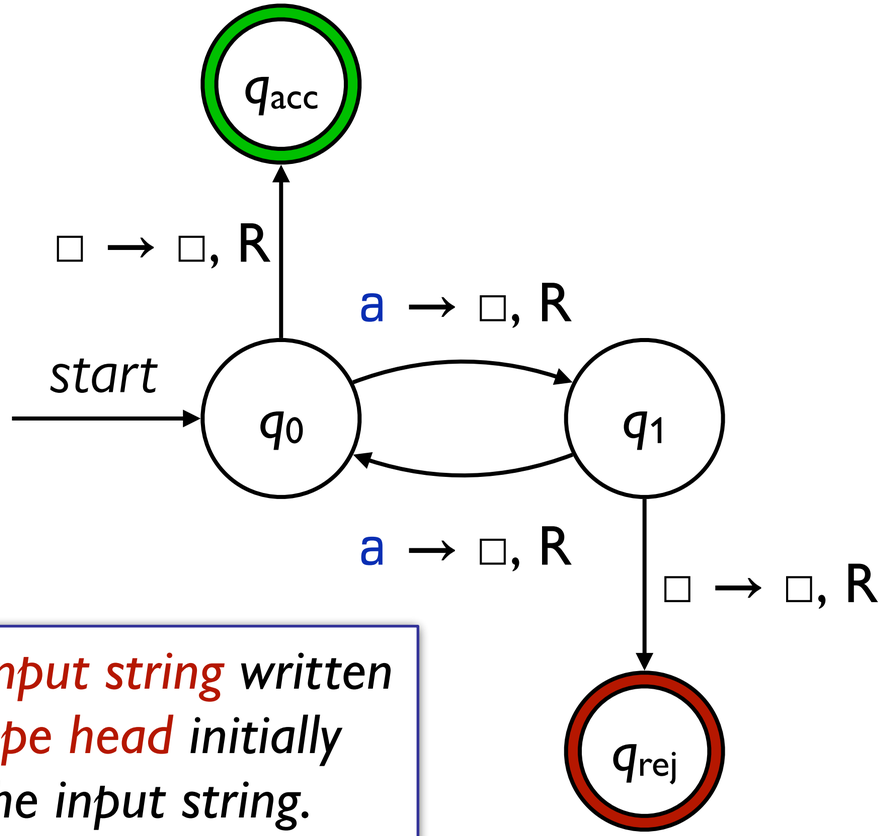
A simple Turing machine



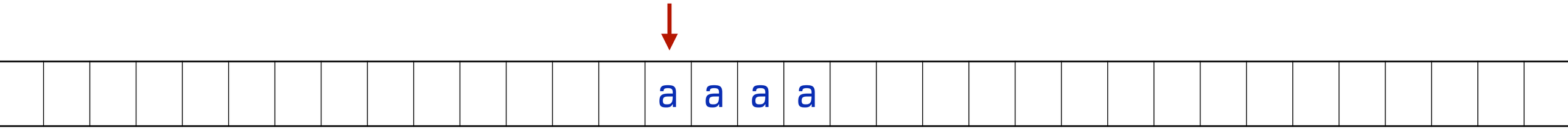
This is the Turing machine's finite-state control. It issues commands that drive the operation of the machine.



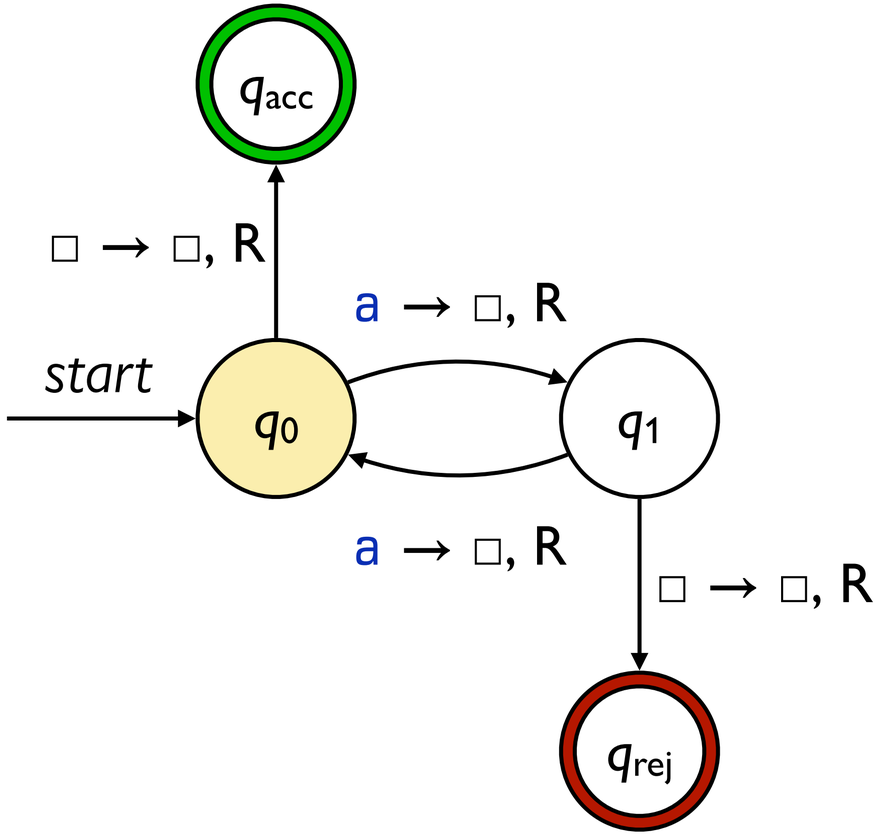
This is the TM's infinite tape. Each tape cell holds a tape symbol. Initially all tape symbols are blank.

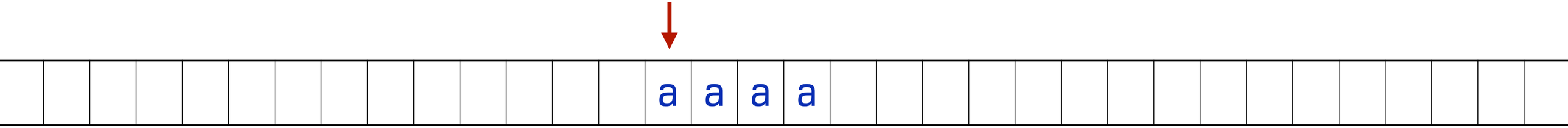


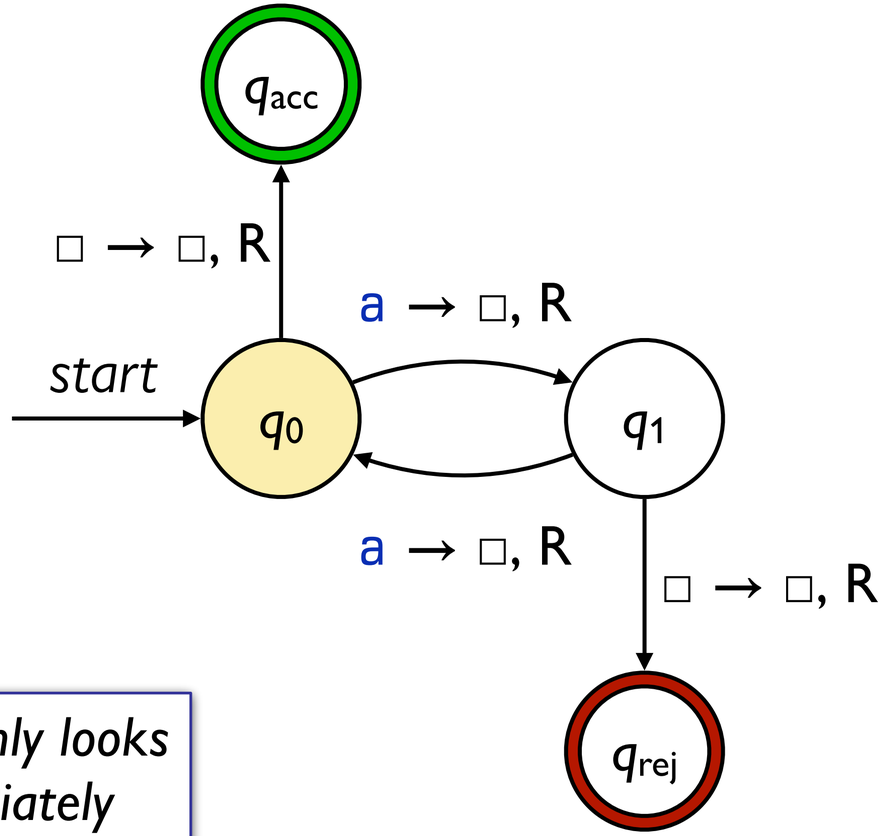
The machine is started with the input string written somewhere on the tape. The tape head initially points to the first symbol of the input string.



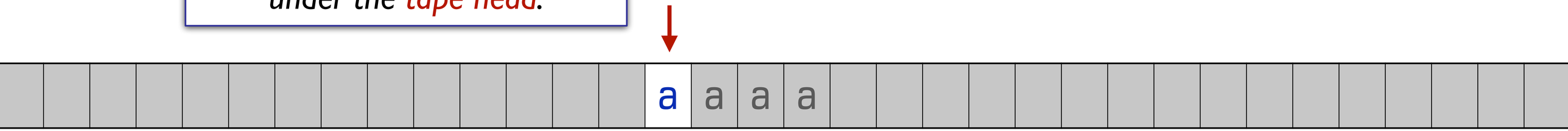
Like other automata, TMs begin execution in their start state

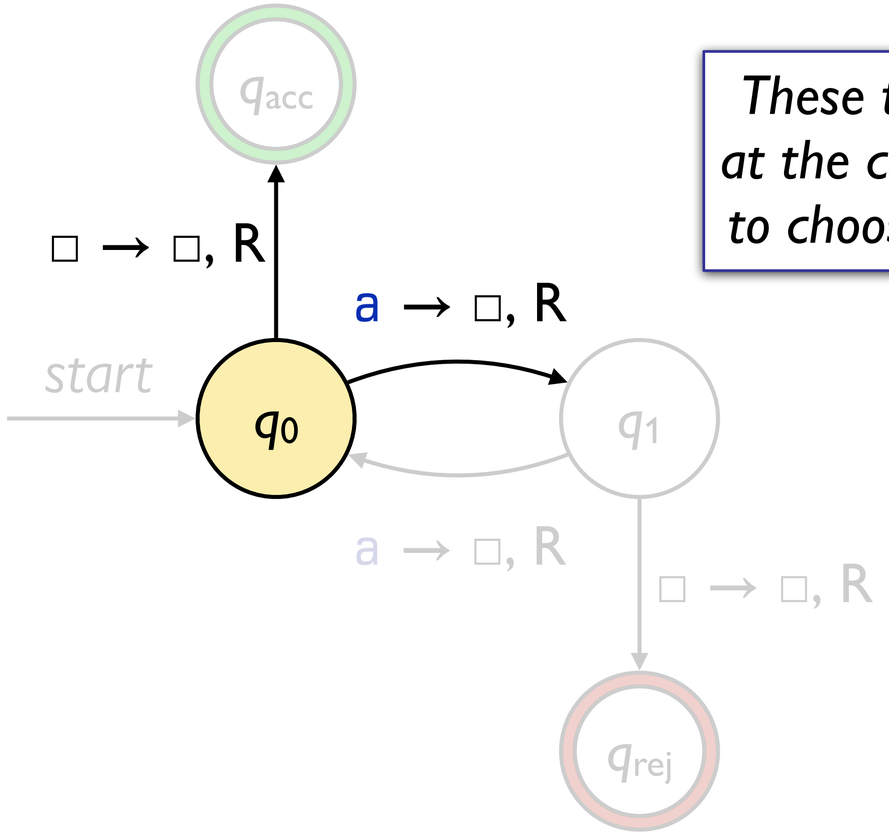




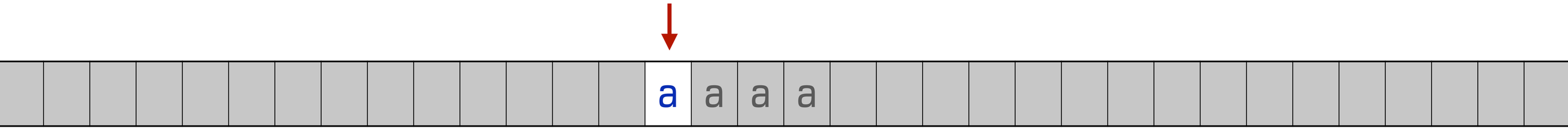


At each step, the TM only looks at the symbol immediately under the tape head.

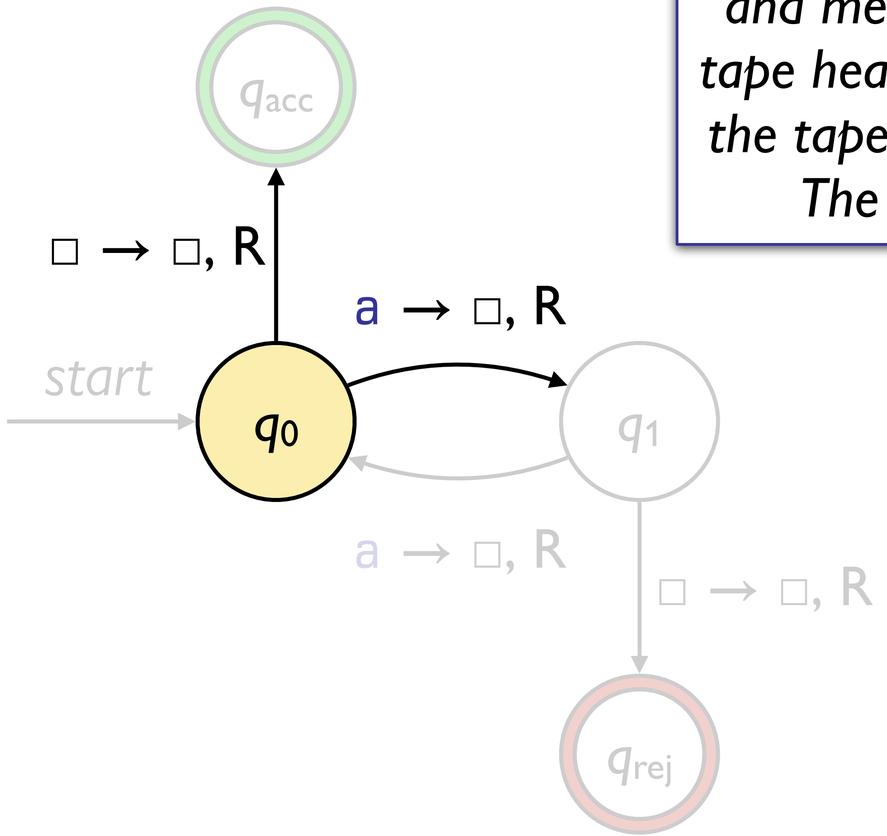


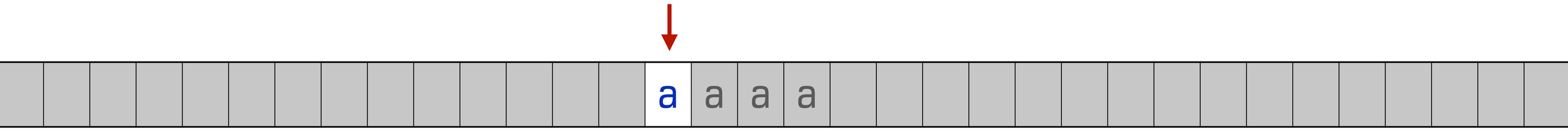


These two transitions originate at the current state. We're going to choose one of them to follow.

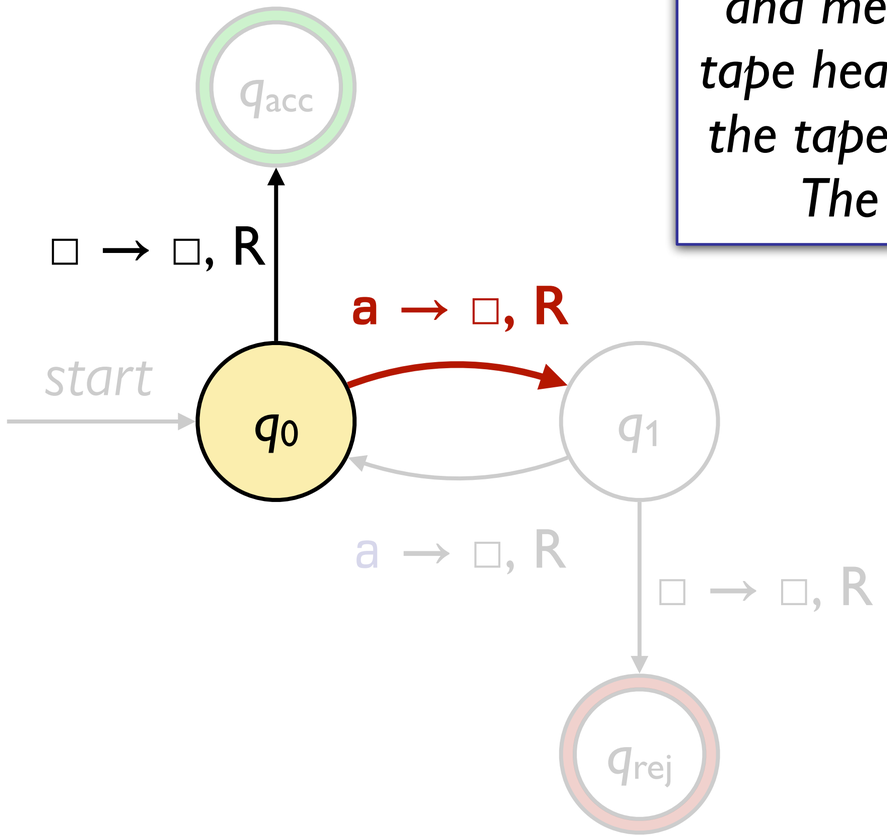


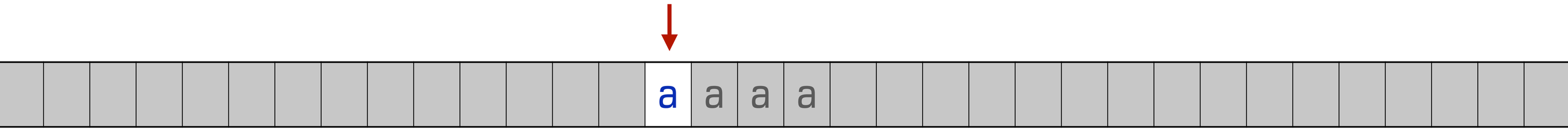
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.



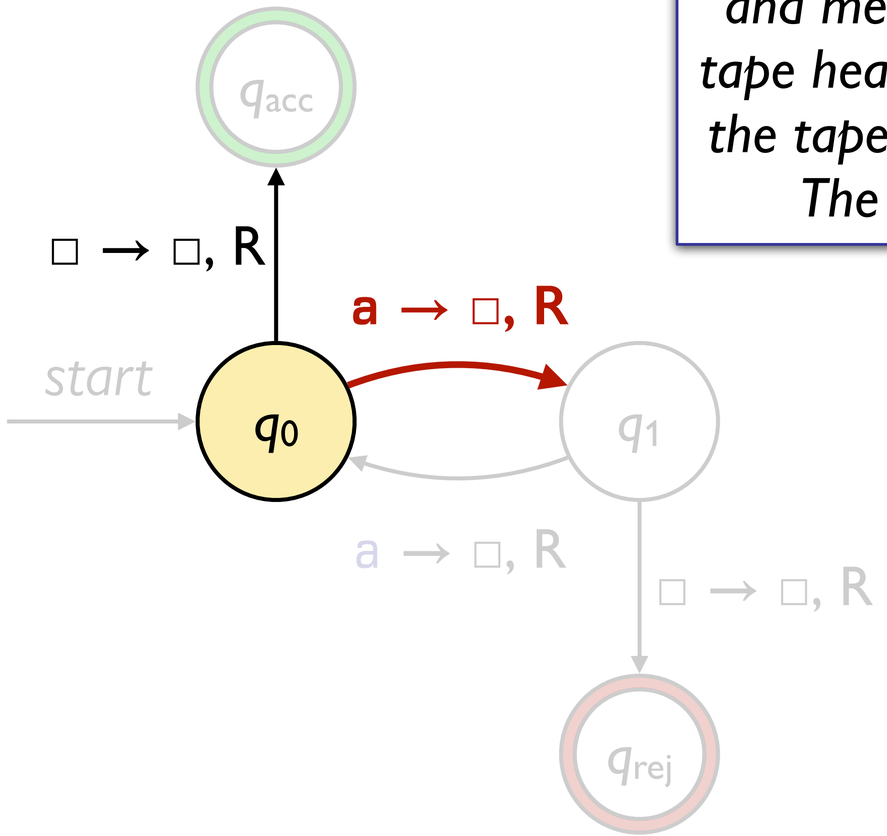


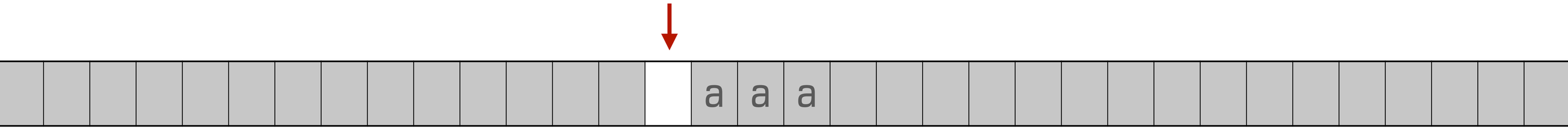
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.



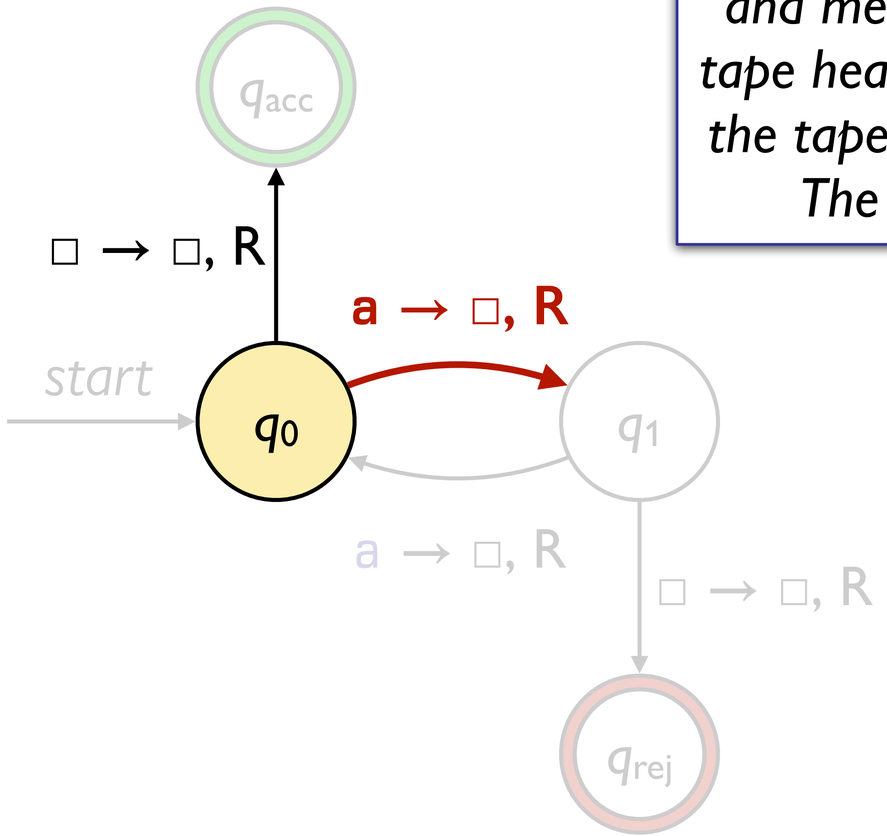


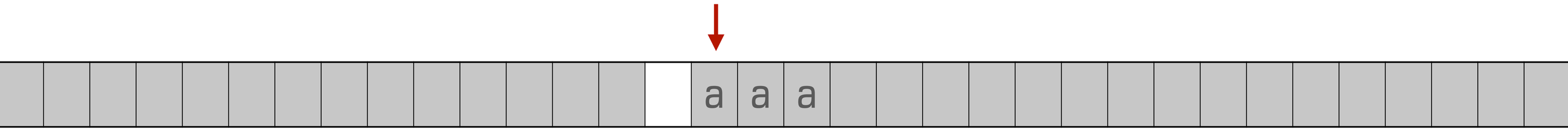
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.



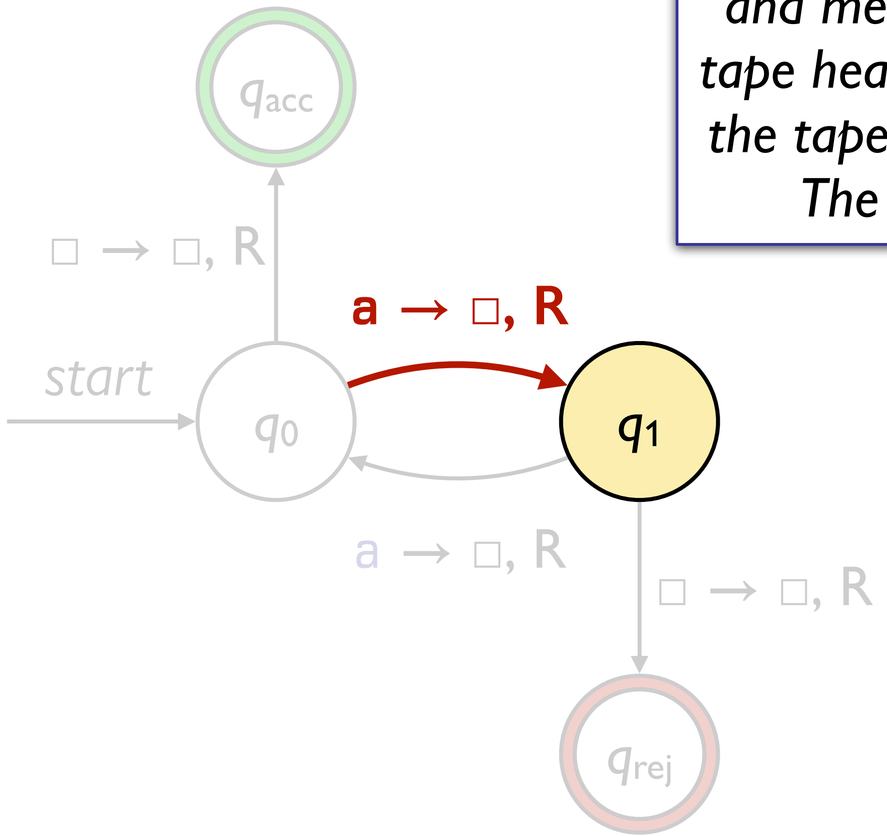


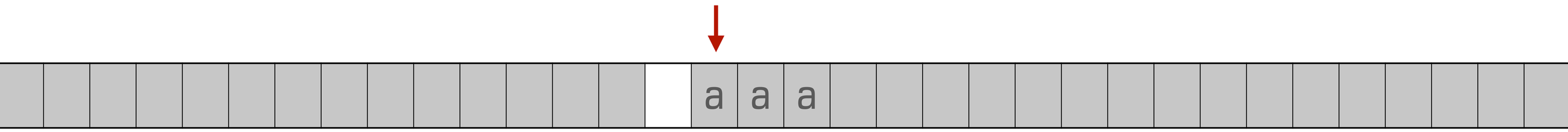
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.

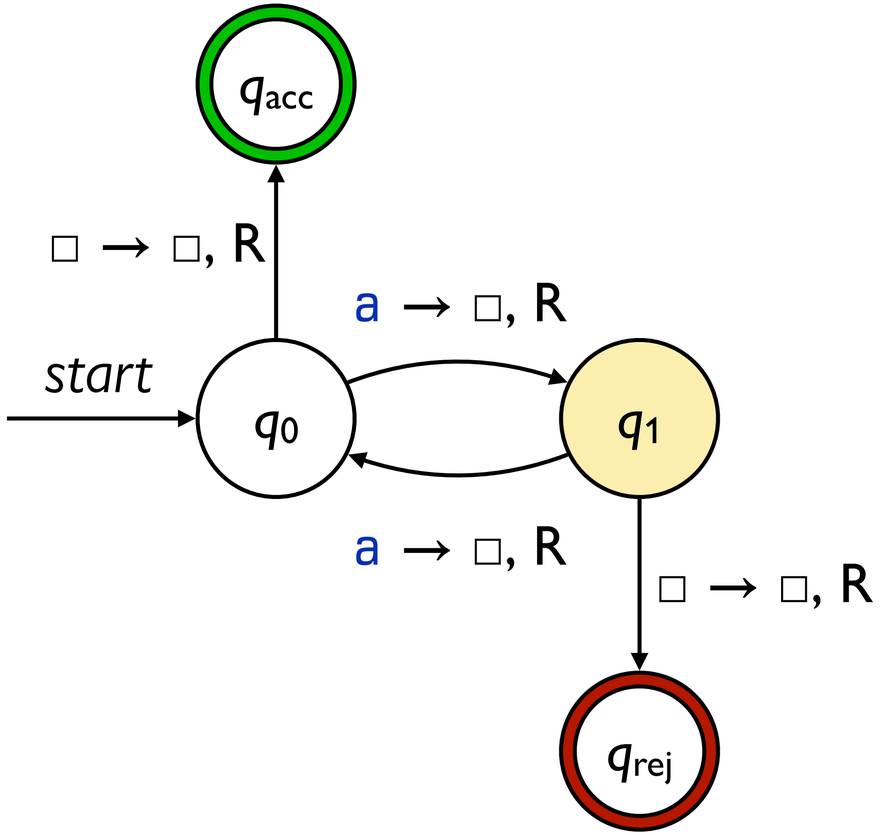


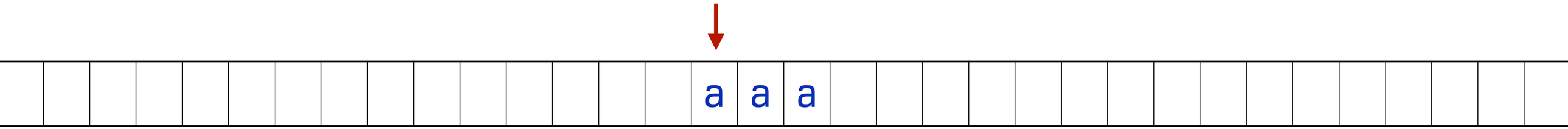


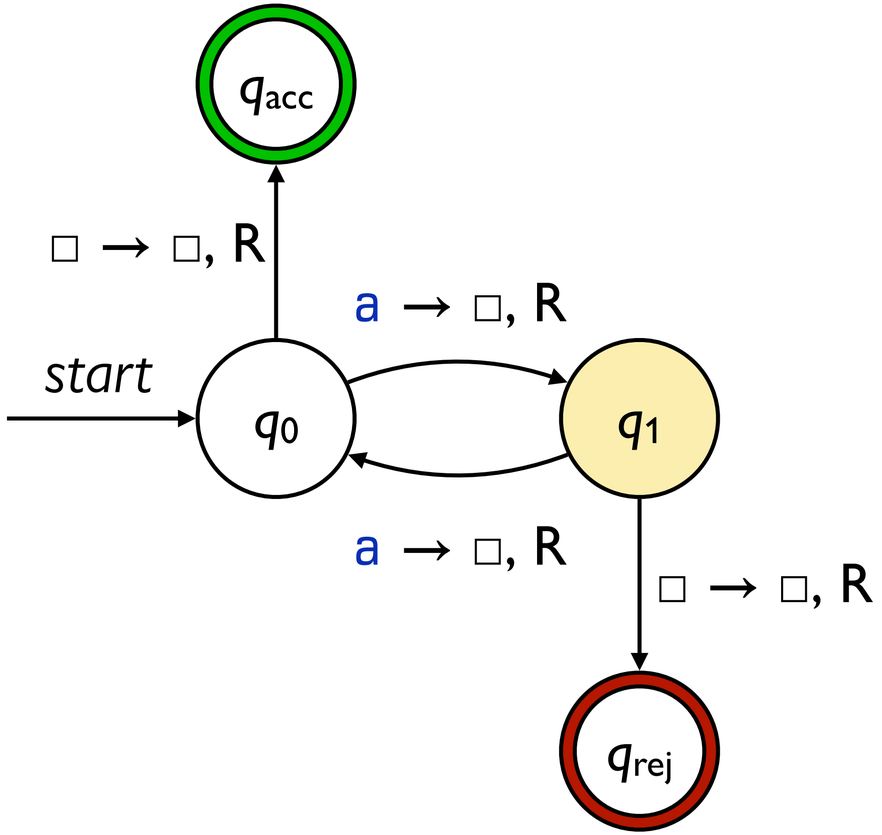
Each transition has the form
*⟨**read**⟩ → ⟨**write**⟩, ⟨**direction**⟩*
*and means "if symbol ⟨**read**⟩ is under the tape head, replace it with ⟨**write**⟩ and move the tape head in ⟨**direction**⟩ (left or right)".*
The □ symbol denotes a blank cell.

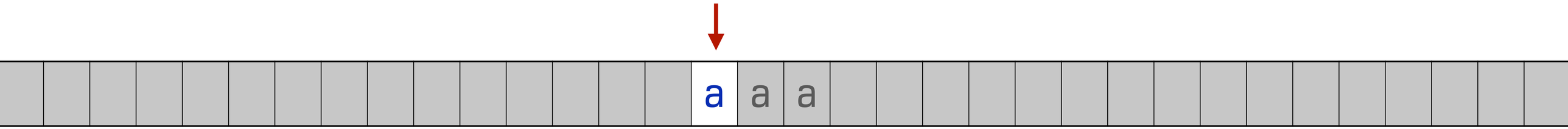


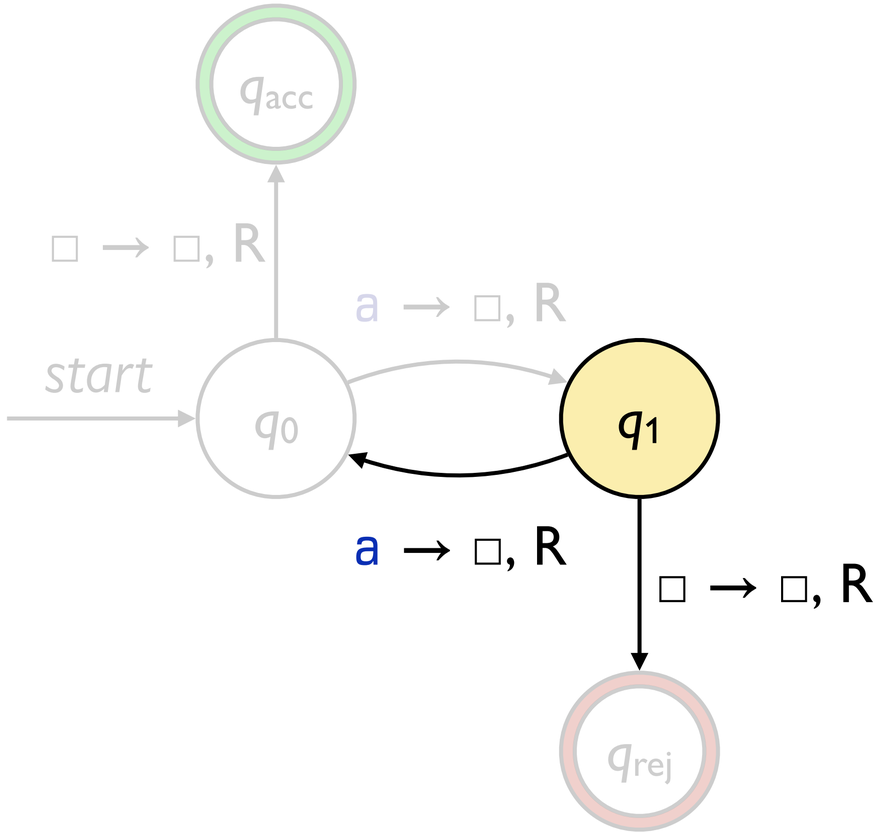


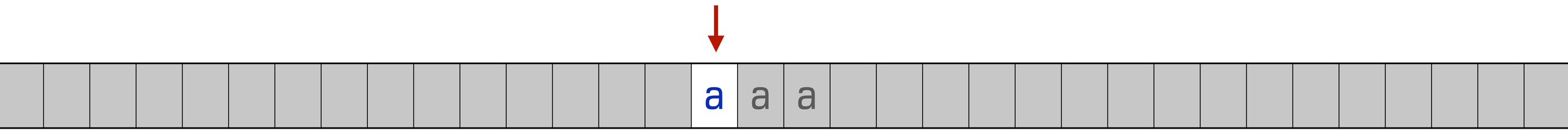

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

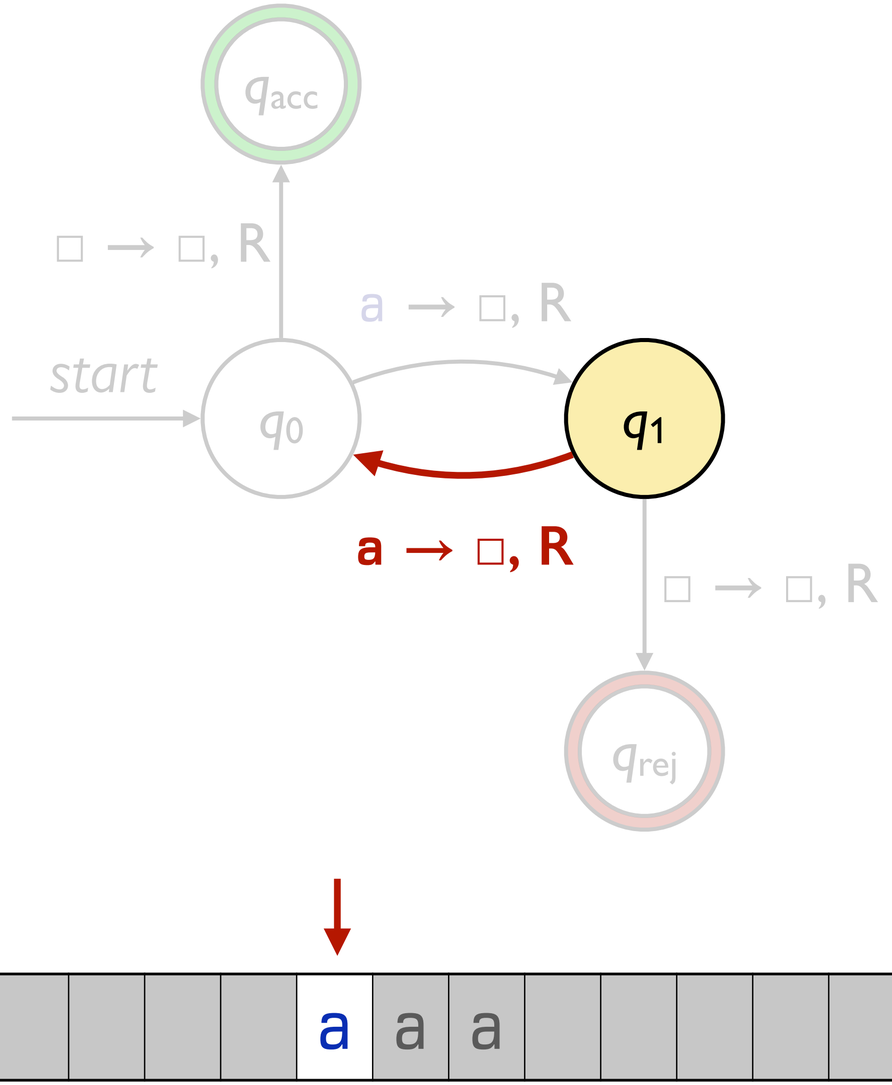

a a a

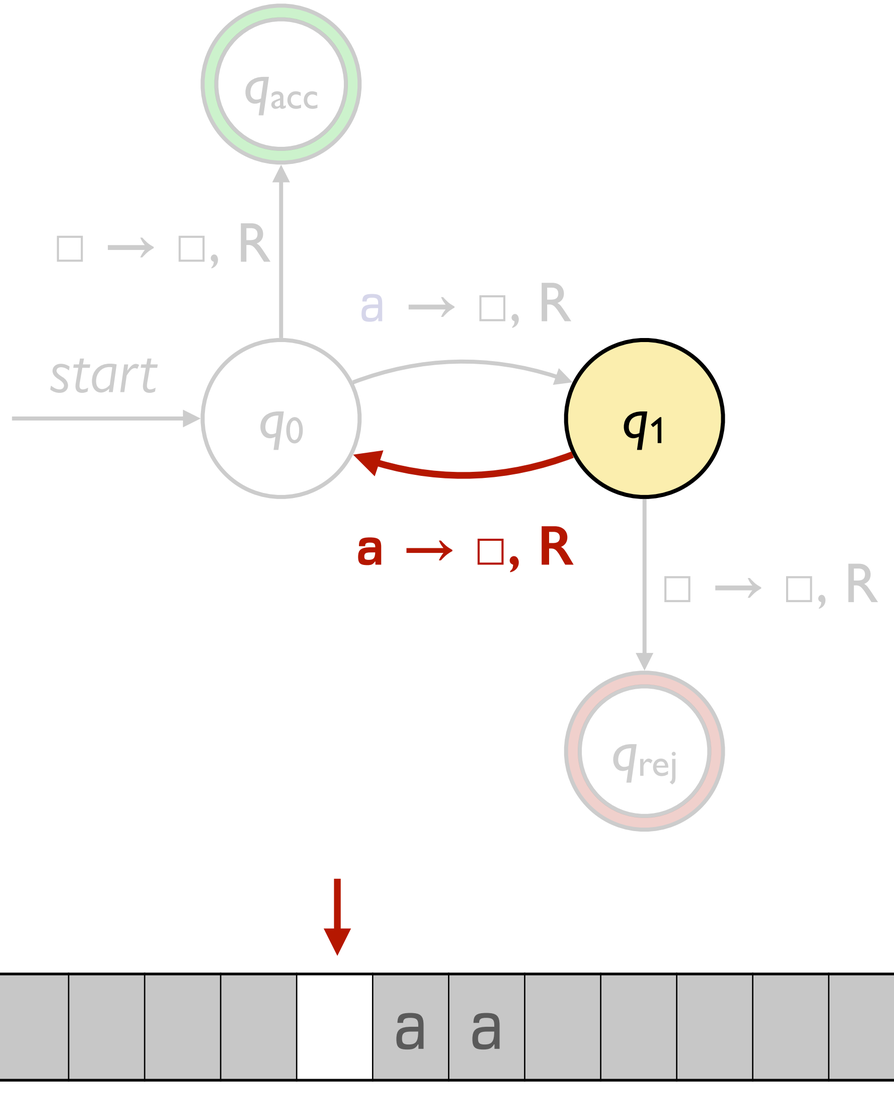

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

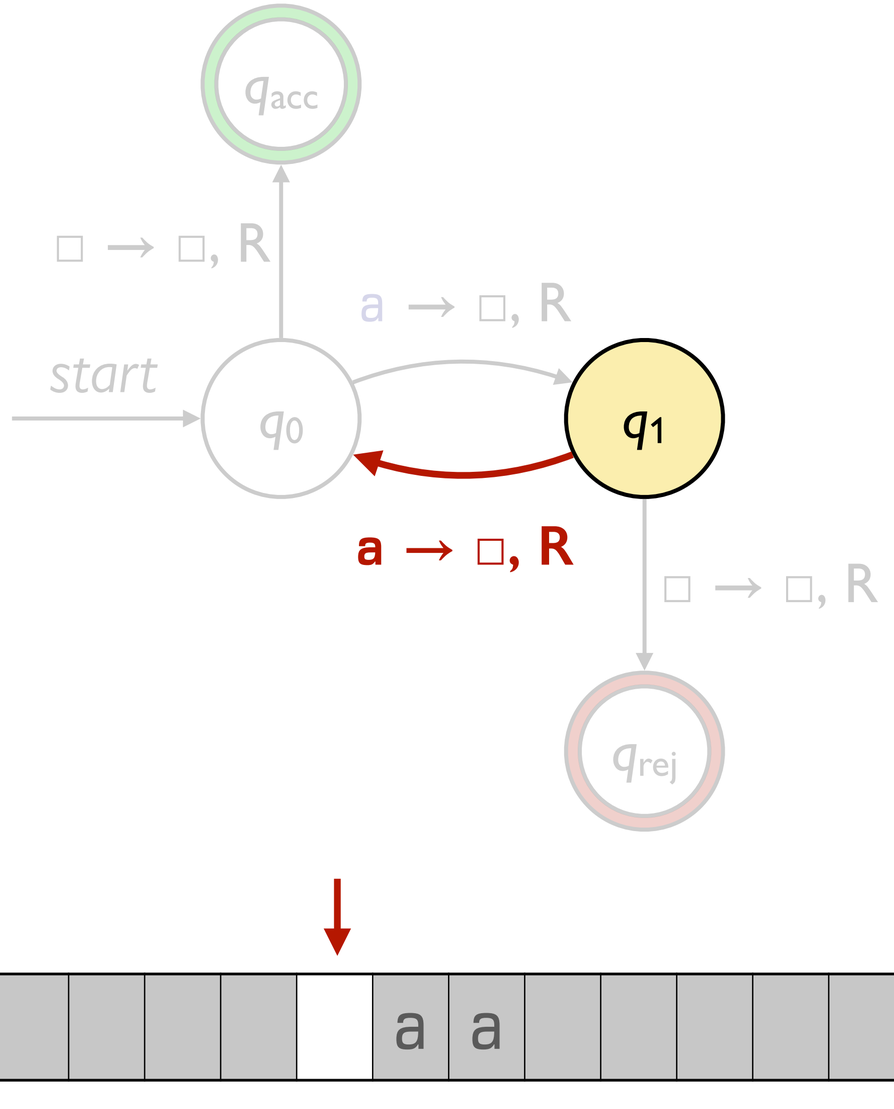

a
a
a

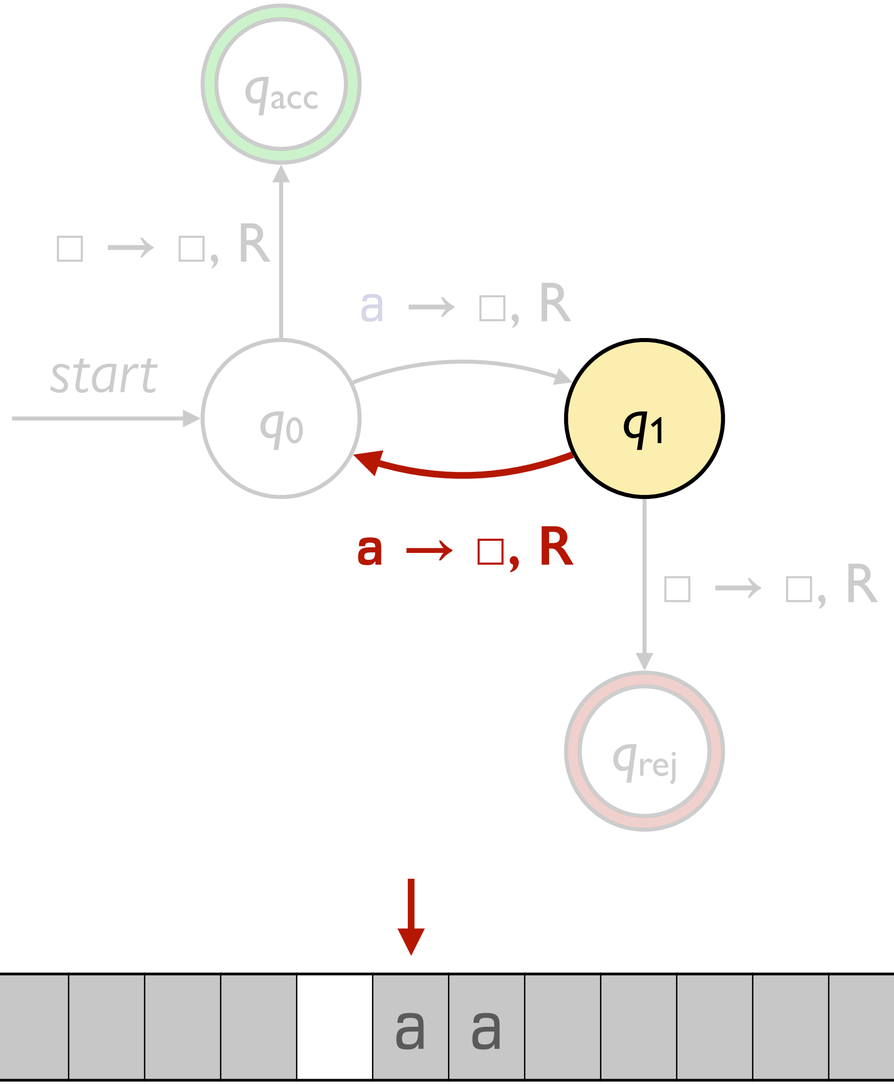

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

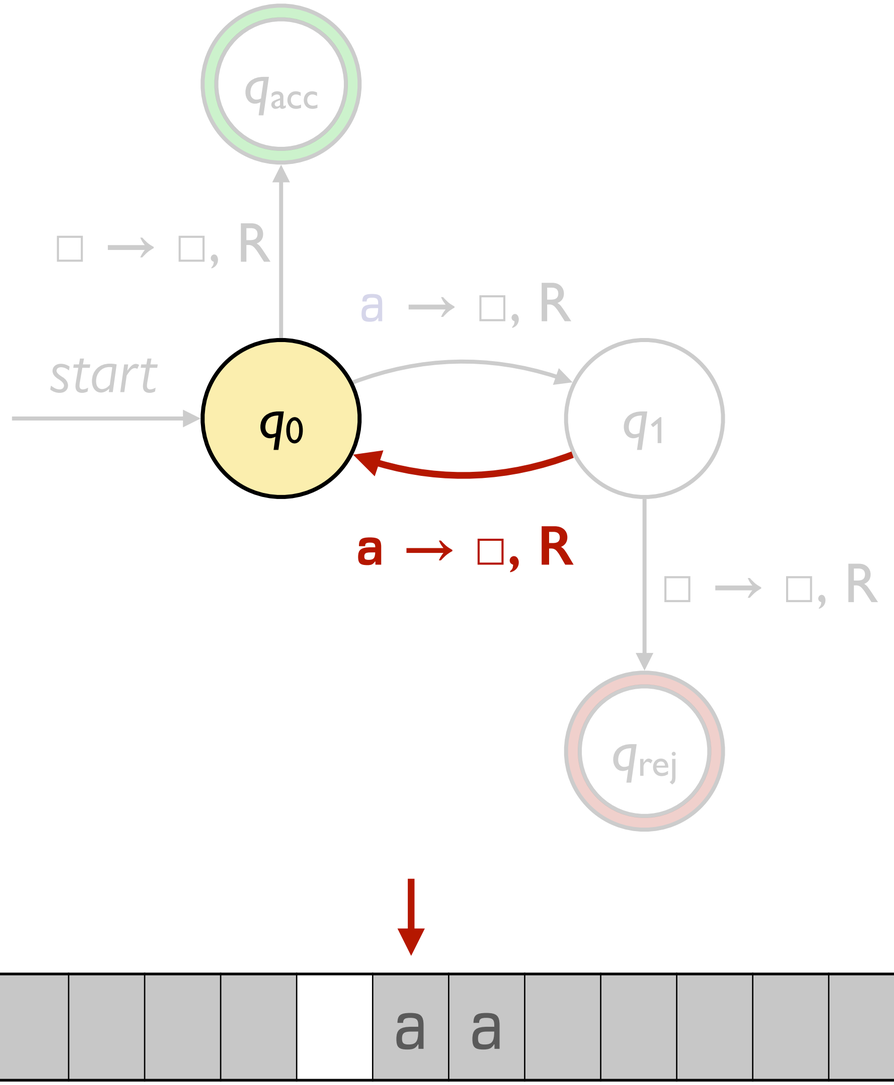

a a a

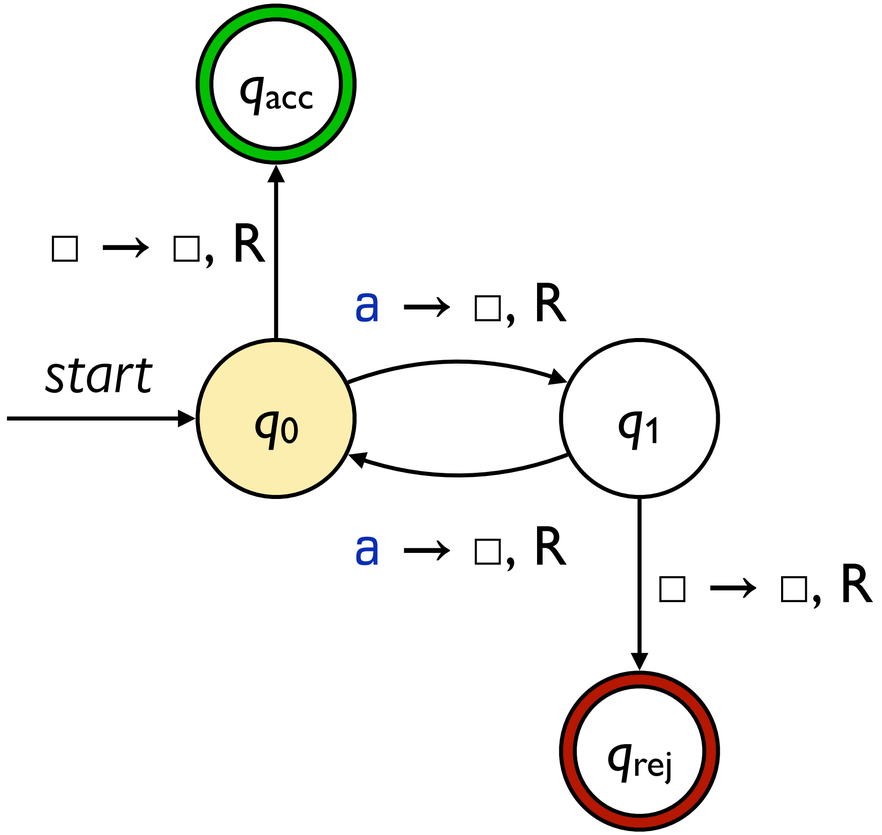

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}
a a a

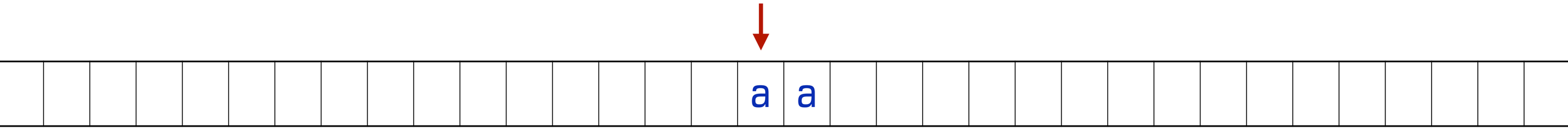

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}
a
a

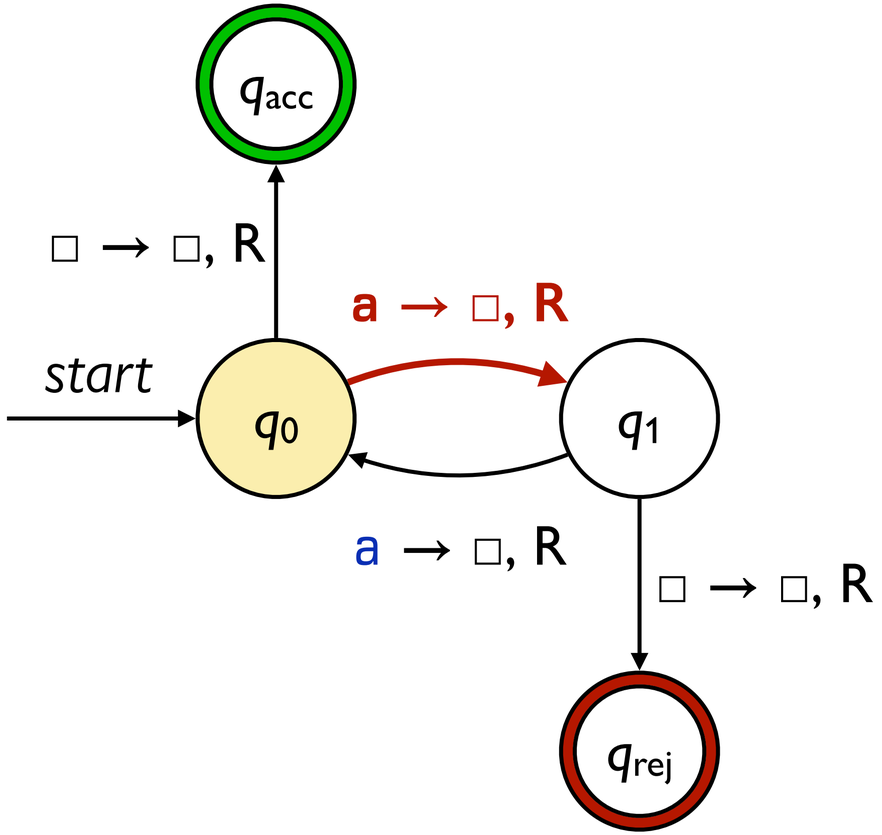

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}
a
a

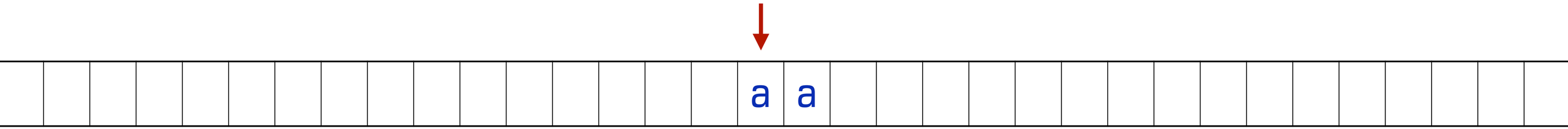

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}
a
a

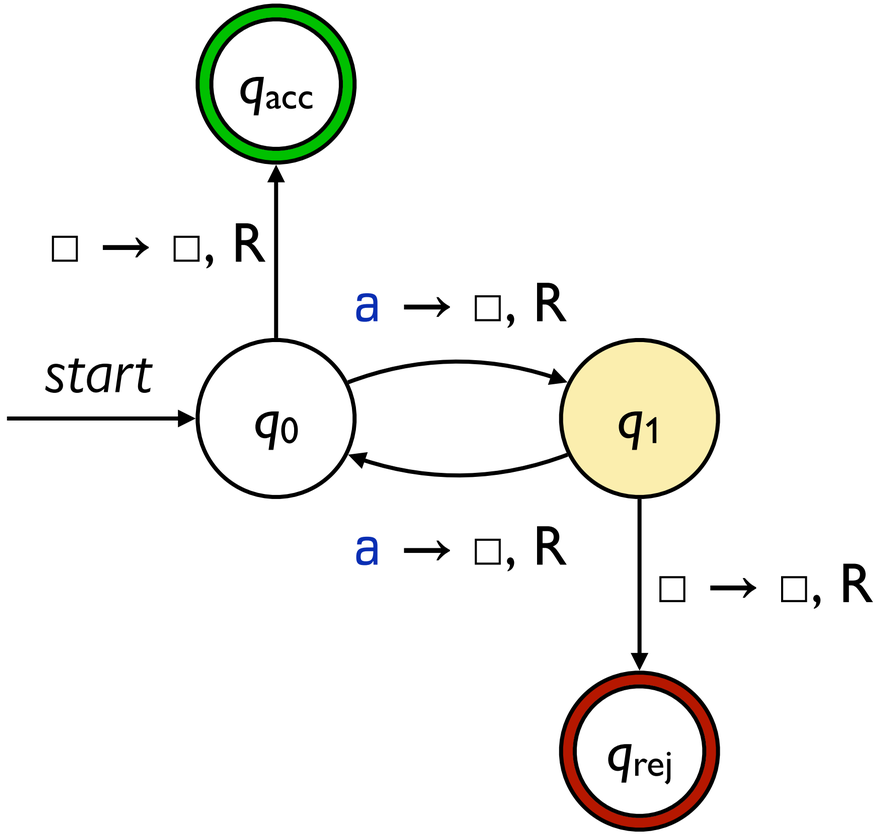

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}
a
a

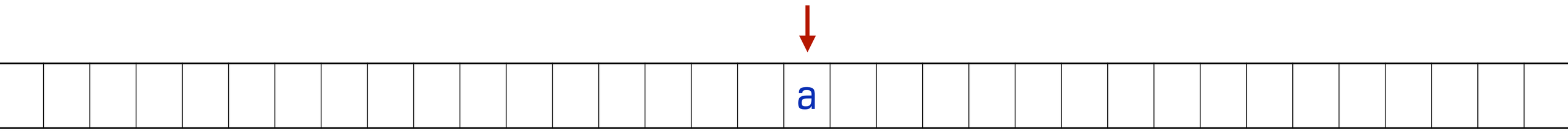

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

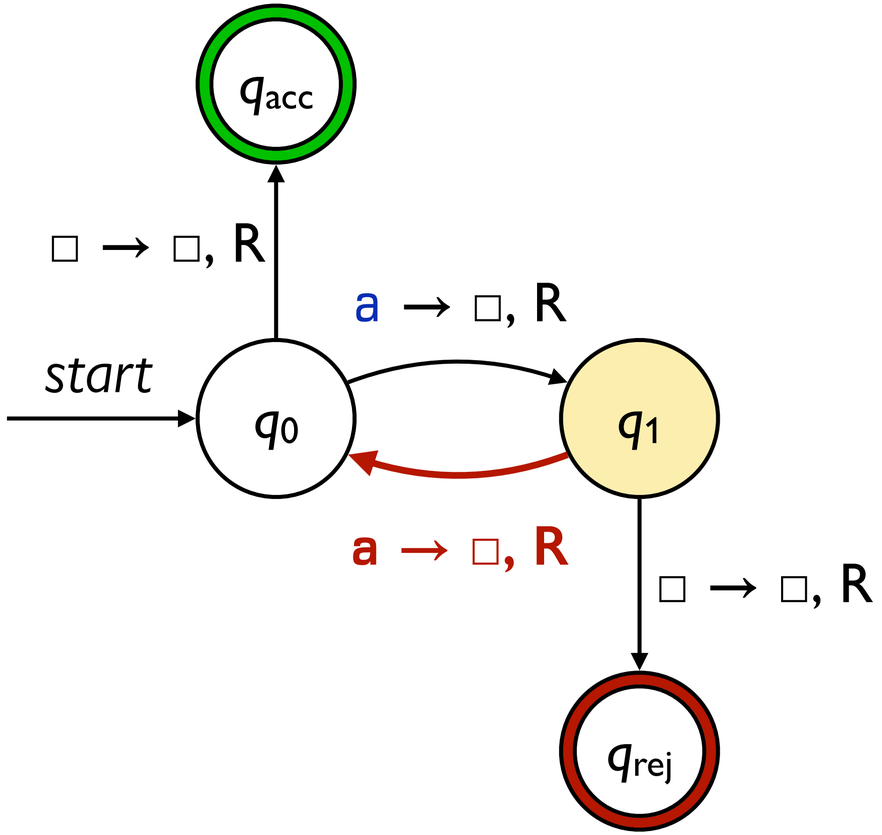

a a

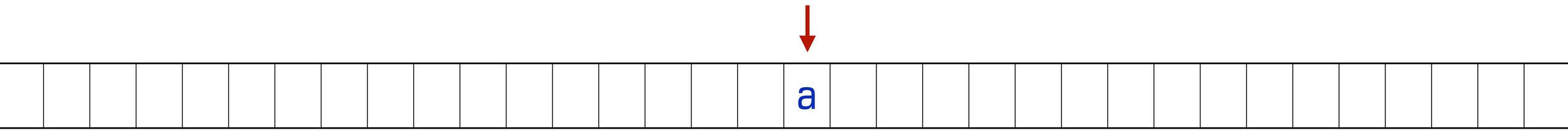

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

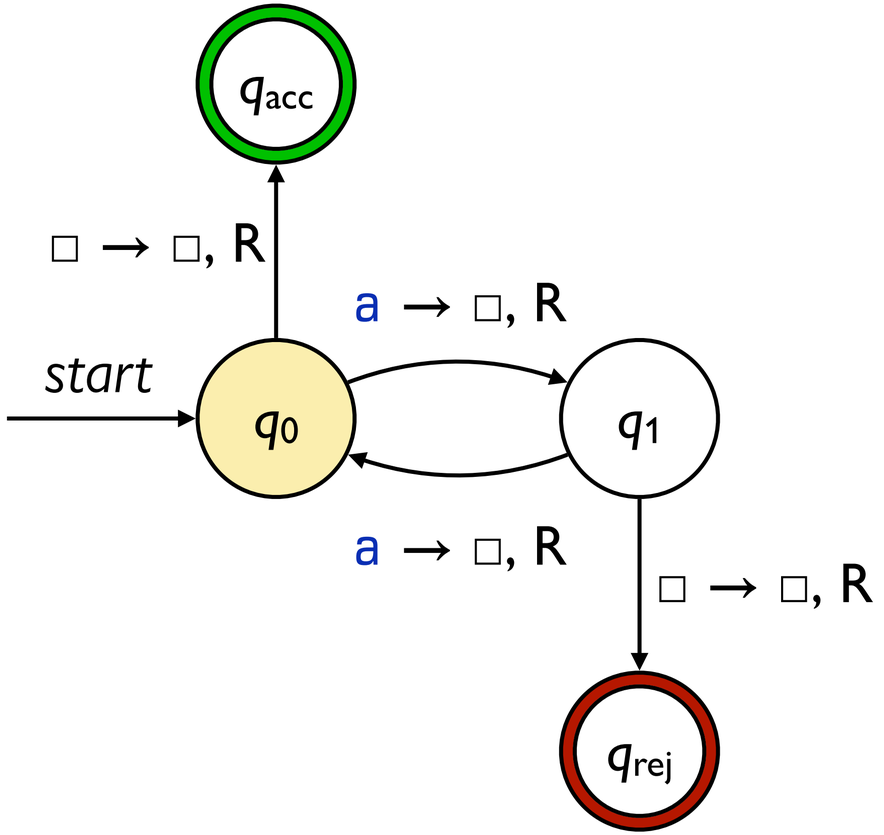

a a


q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

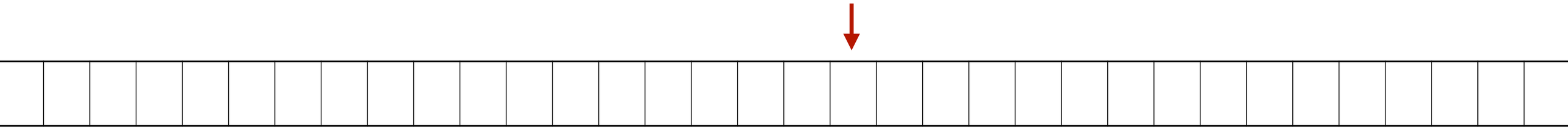

a

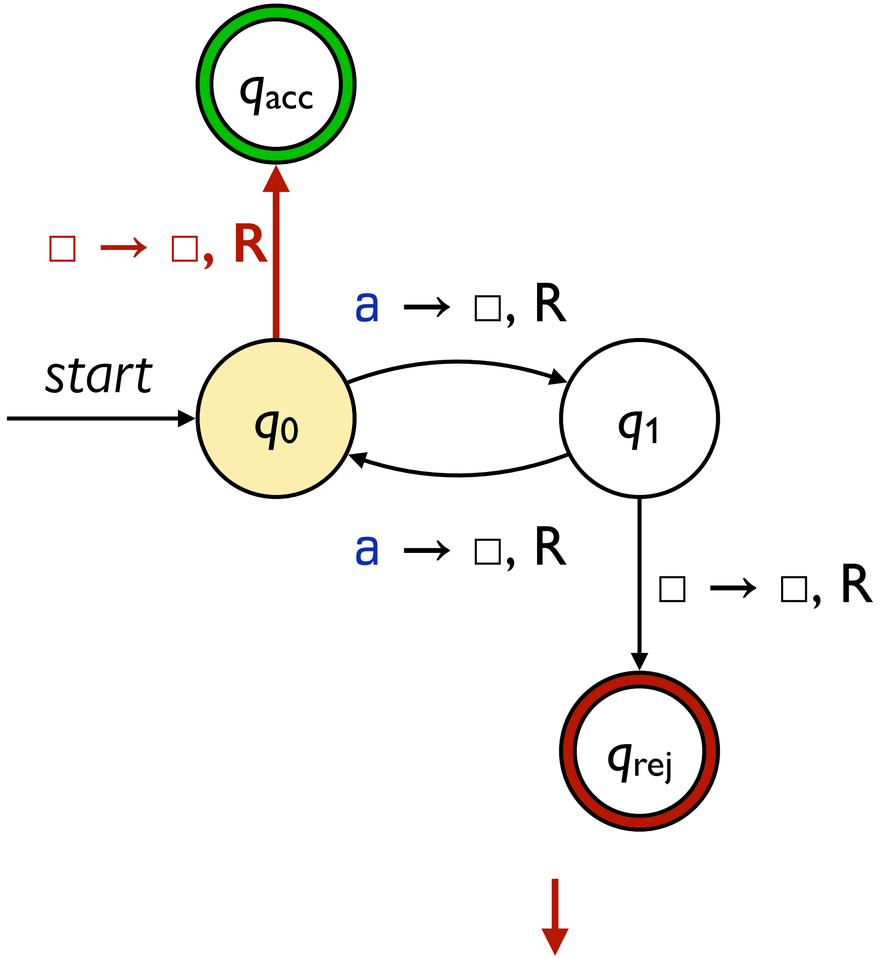

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}


a

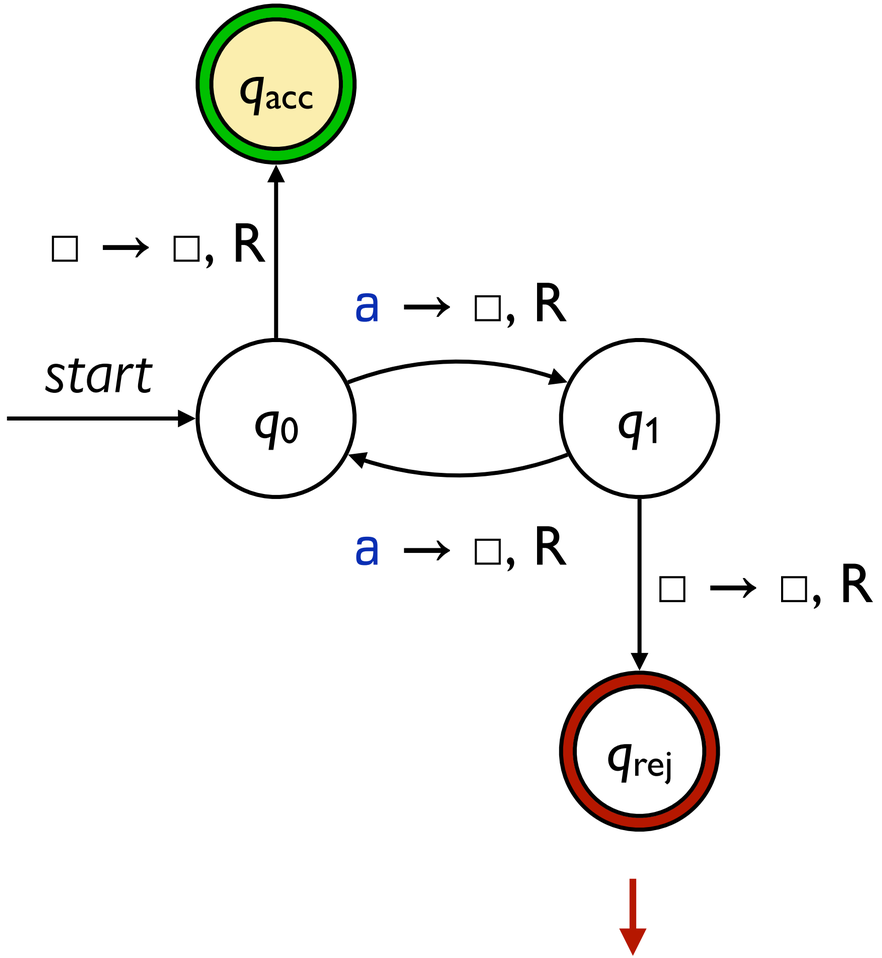


Unlike a DFA or NFA, a TM doesn't stop after reading all the input characters. We keep running until the machine explicitly says to stop.

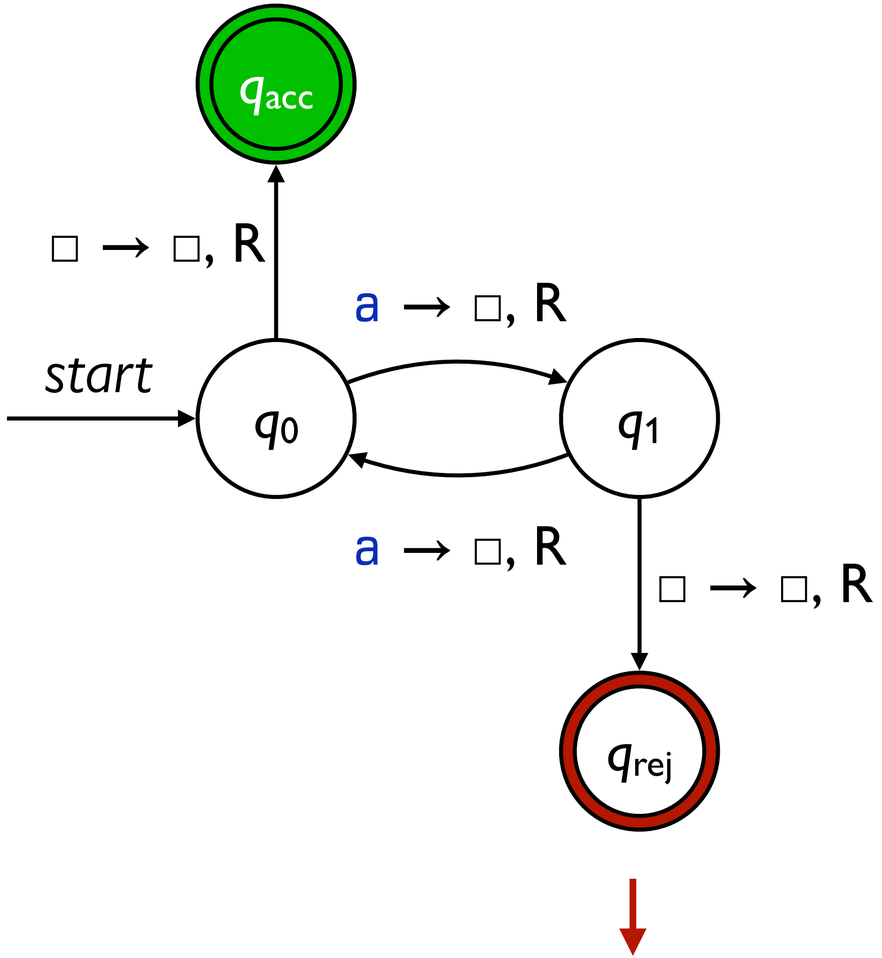


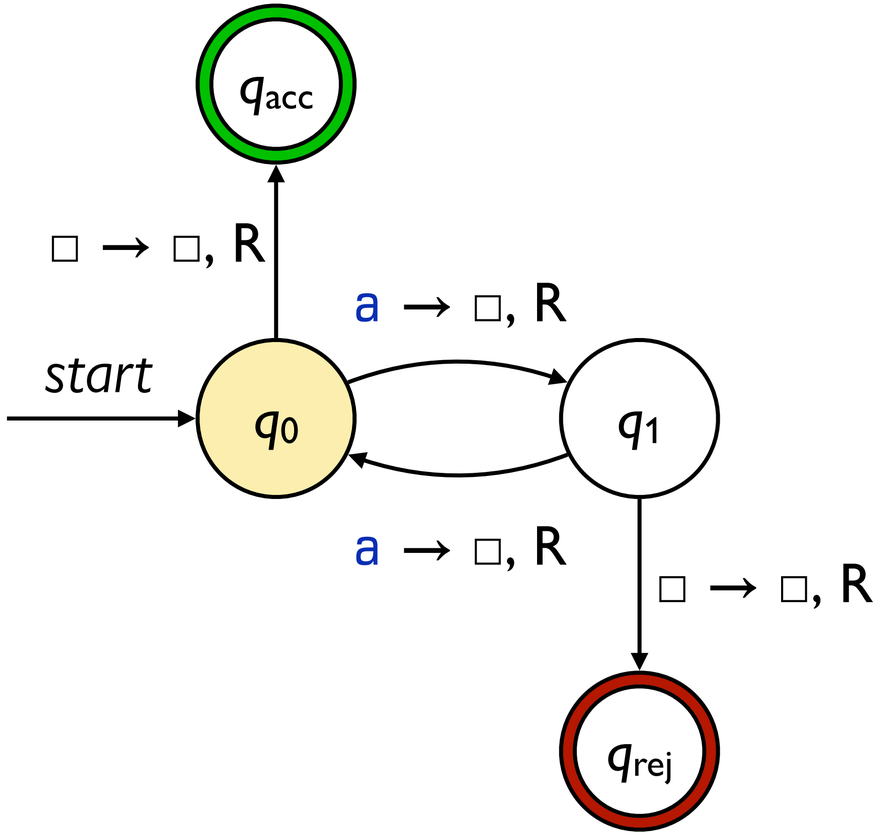

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

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

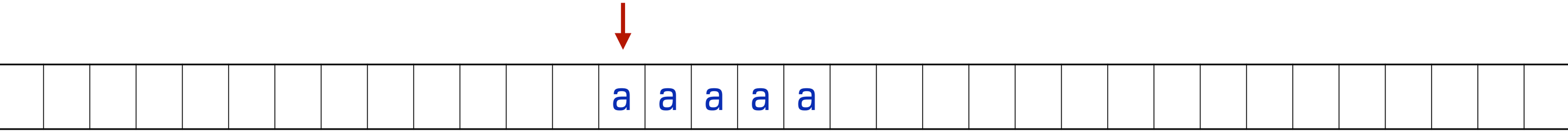


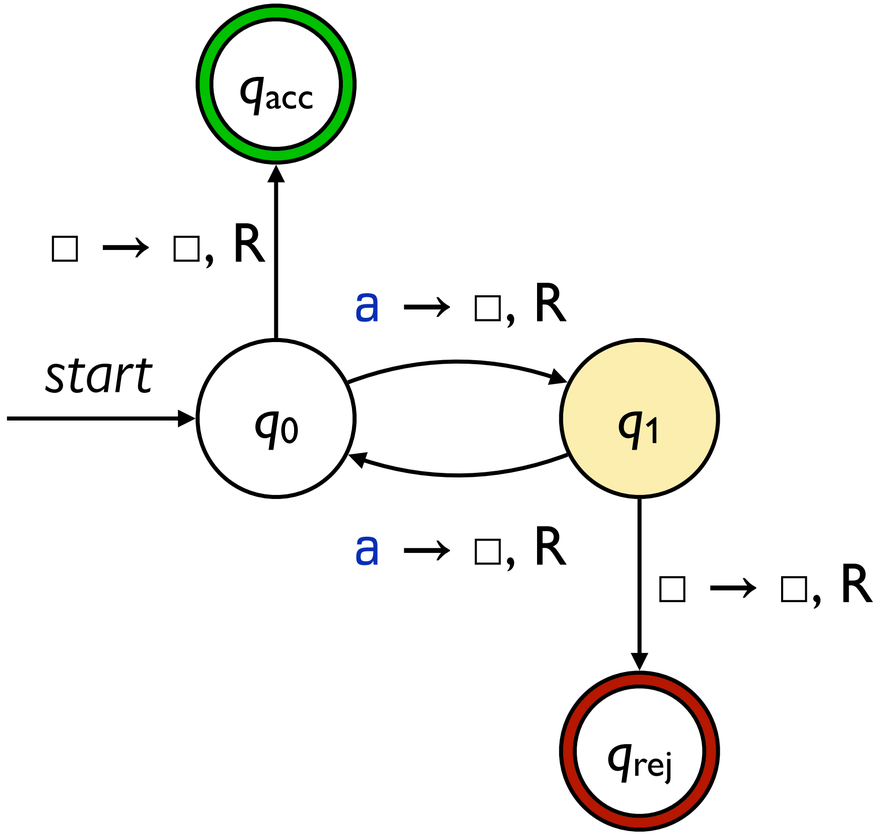
This special state is an accept state. When a TM enters an accept state, it immediately stops running and accepts whatever the original input string was – in this case, aaaa.

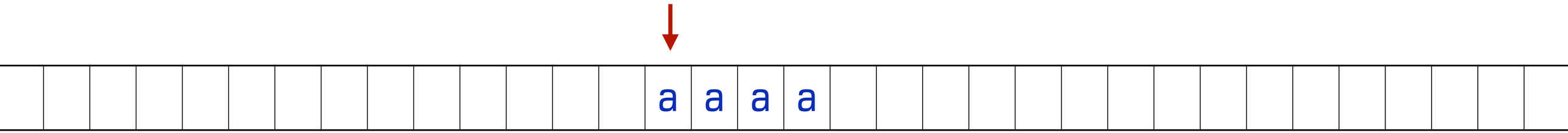


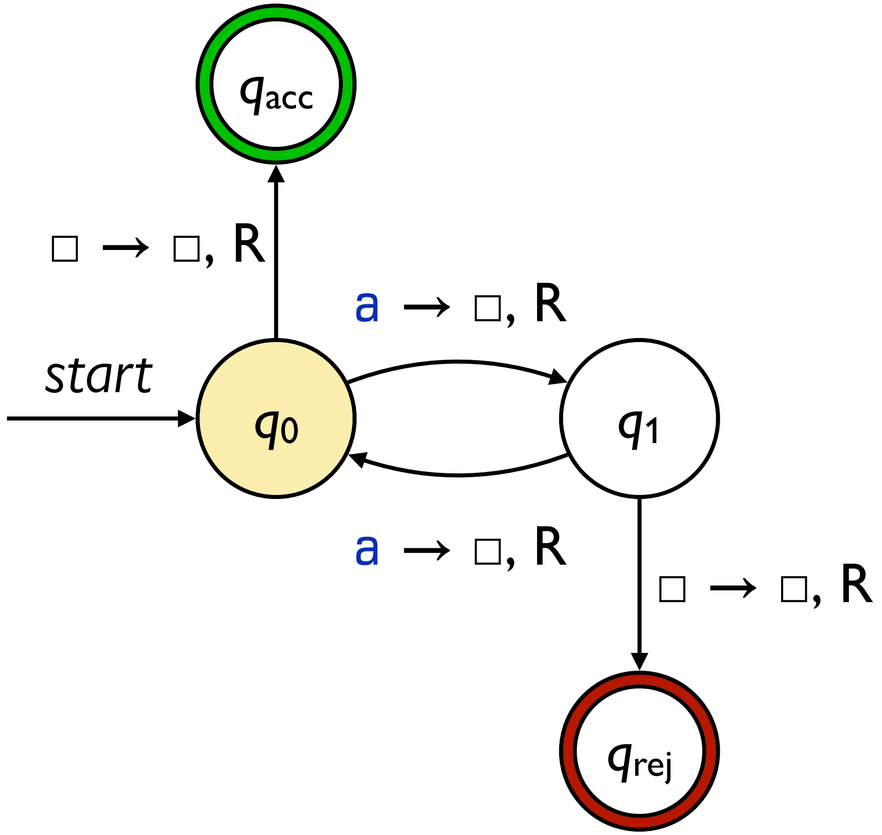


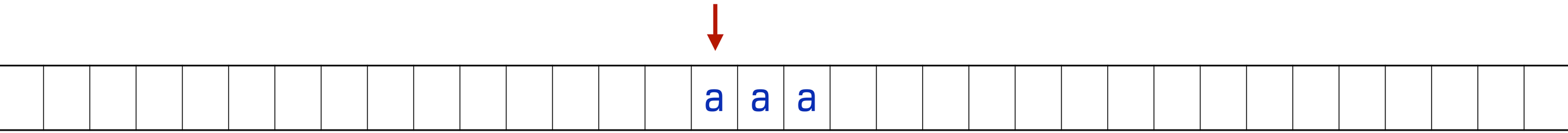
Would the TM accept this string?

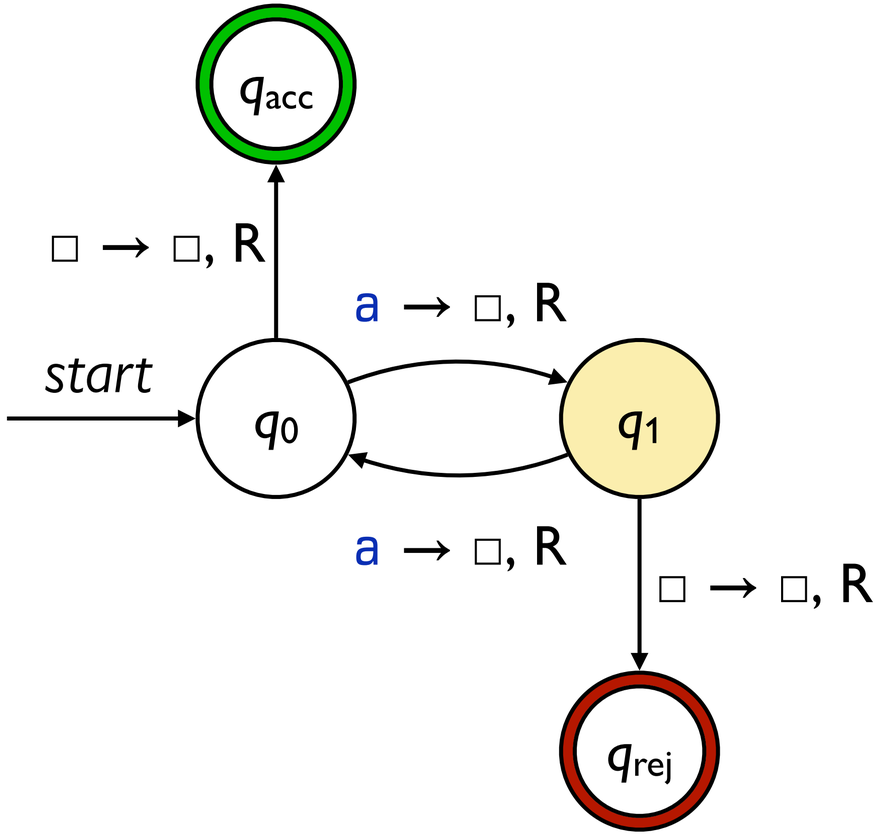


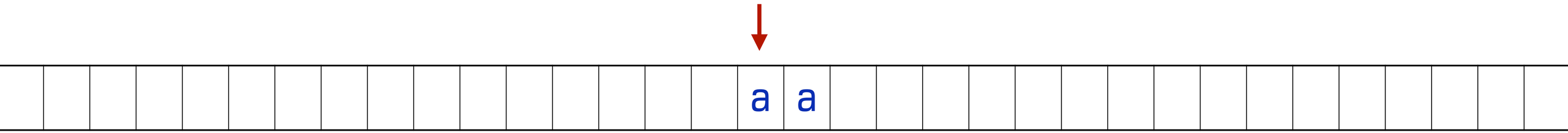

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

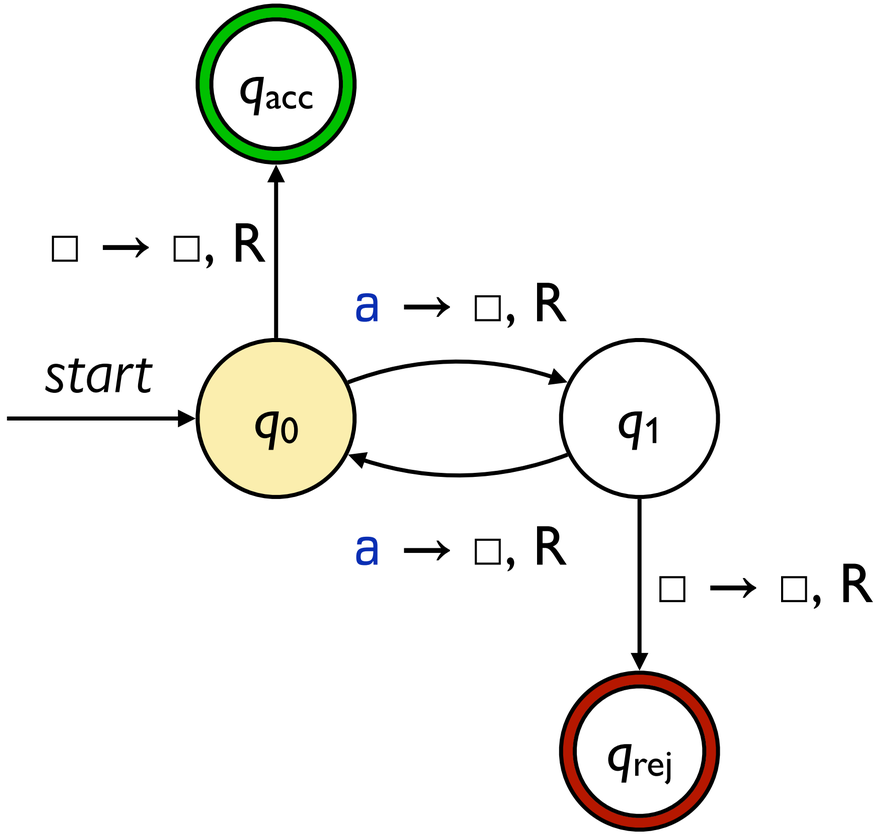

a a a a

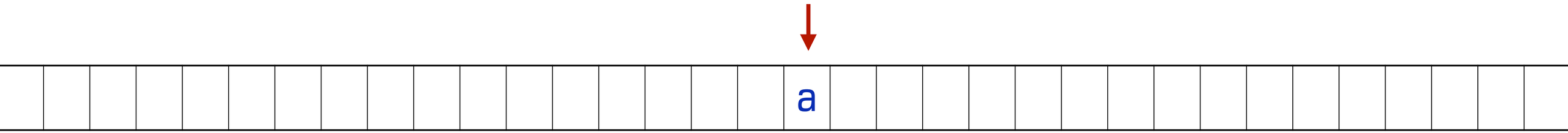

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}

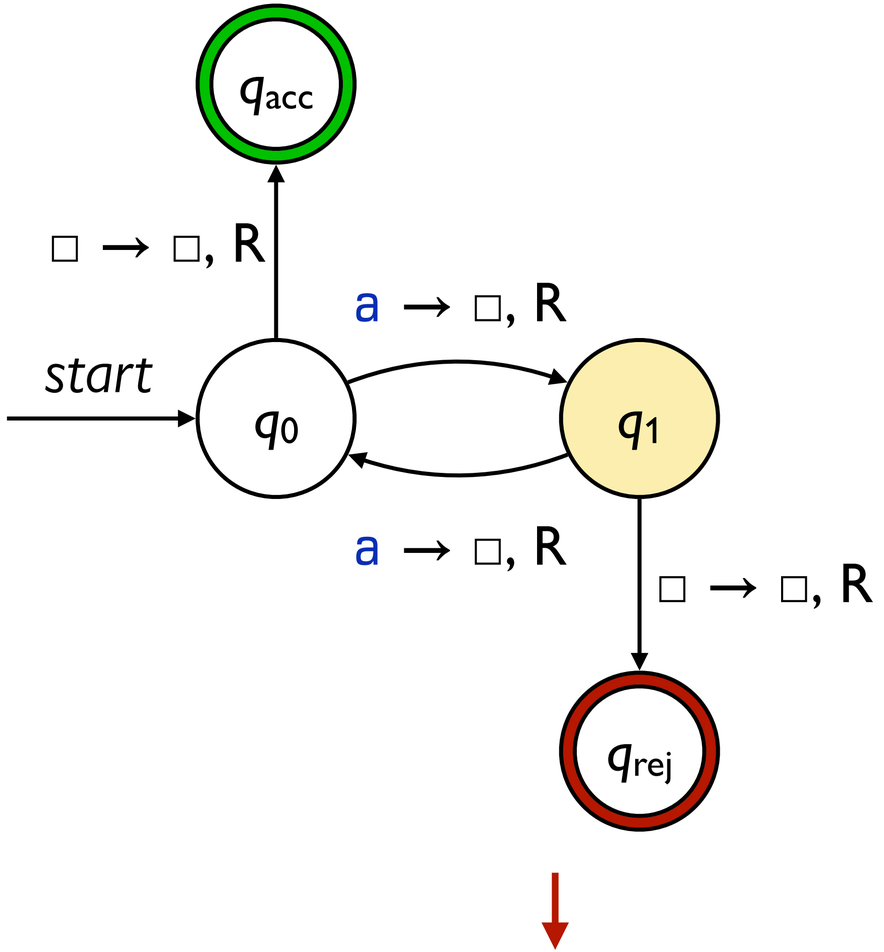

a a a

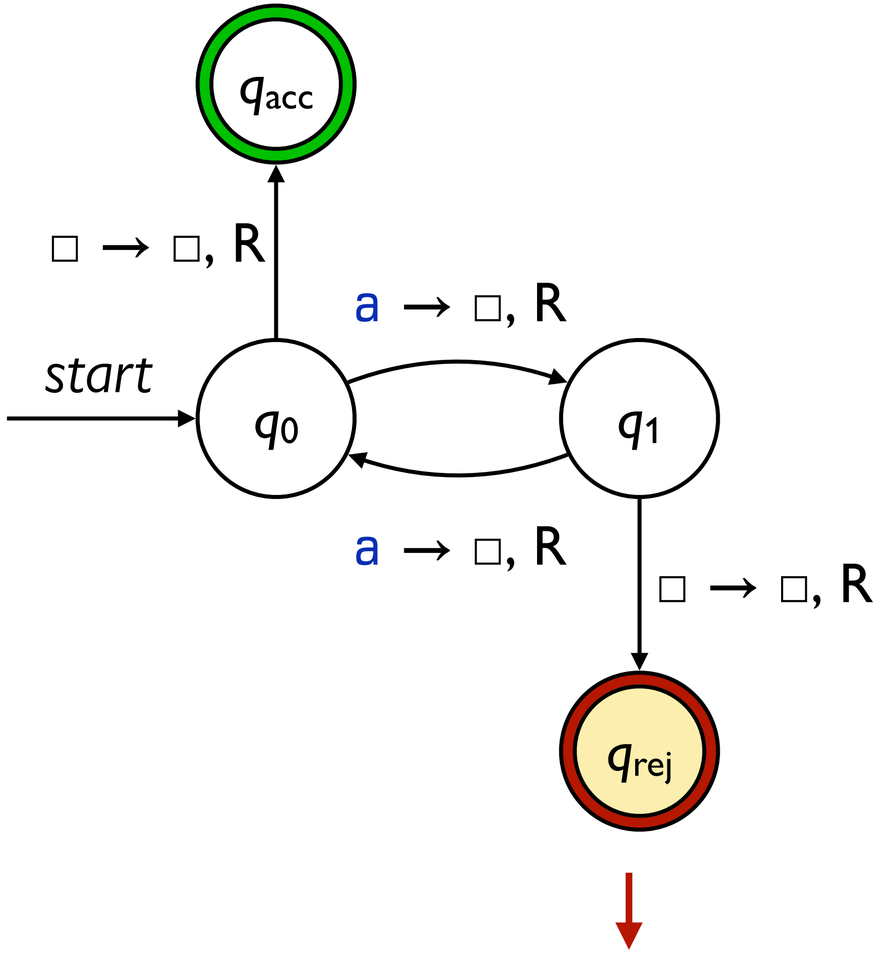

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}


a a

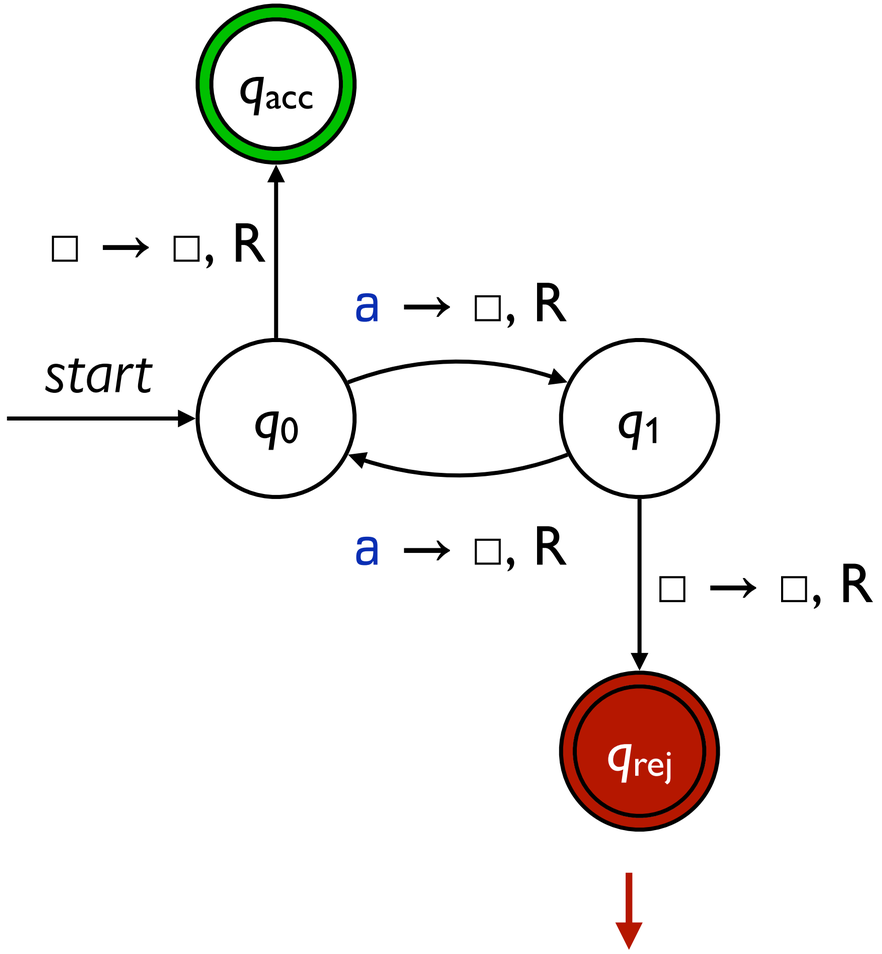

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}


a

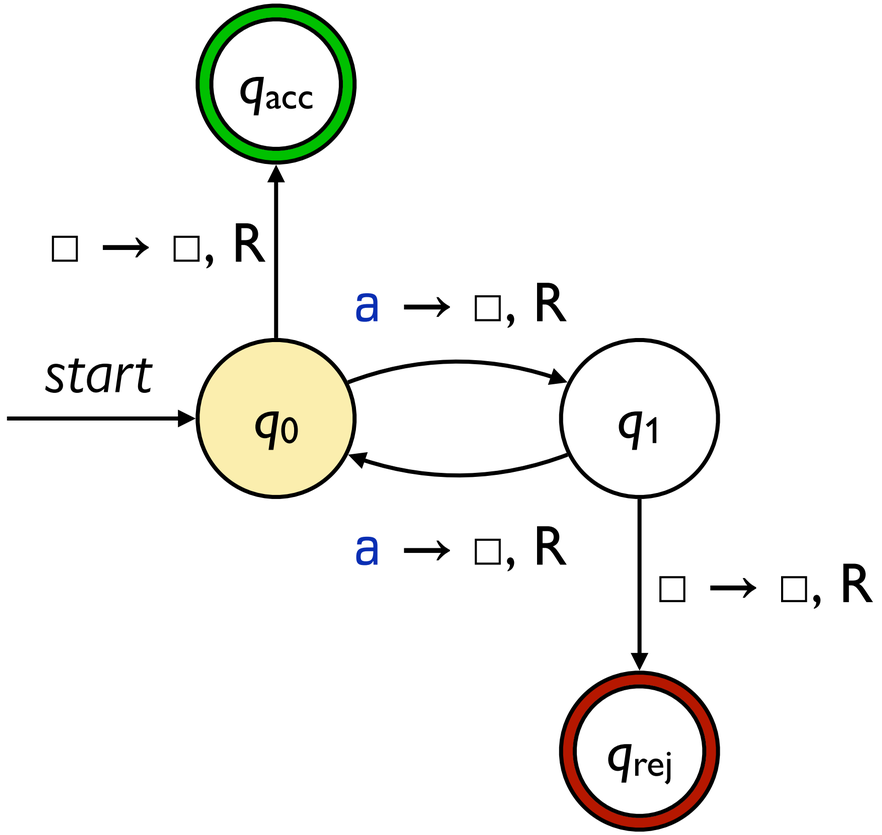

q_{acc}
□ → □, R
a → □, R
start
q_0
q_1
a → □, R
□ → □, R
q_{rej}



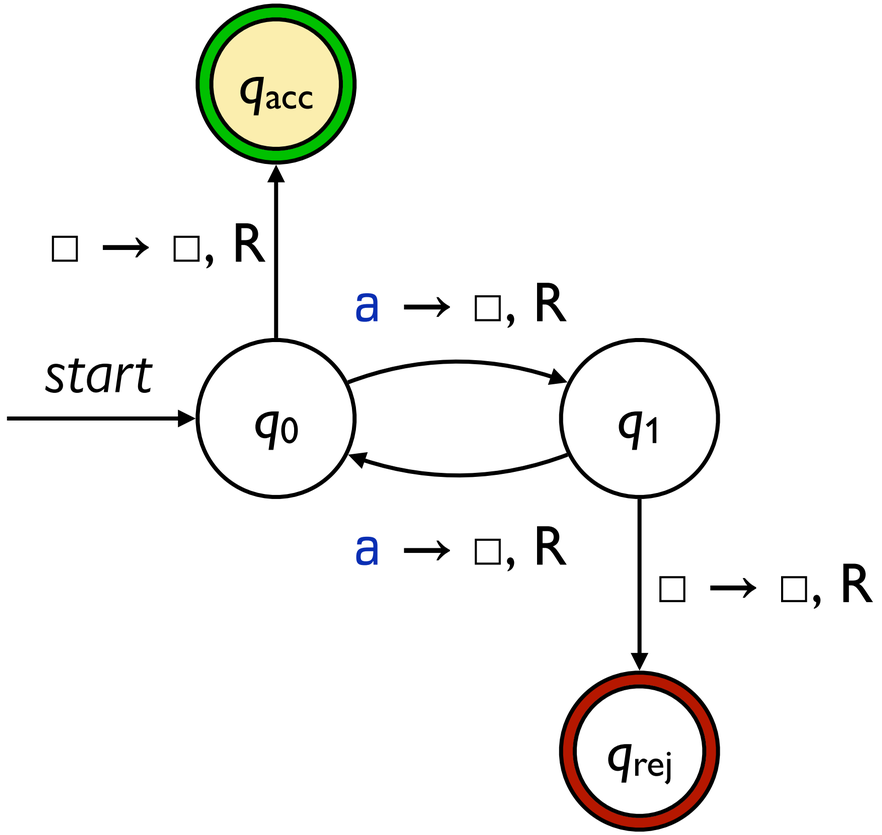
This special state is a reject state. When a TM enters a reject state, it immediately stop running and rejects whatever the original input string was – in this case, aaaaa.



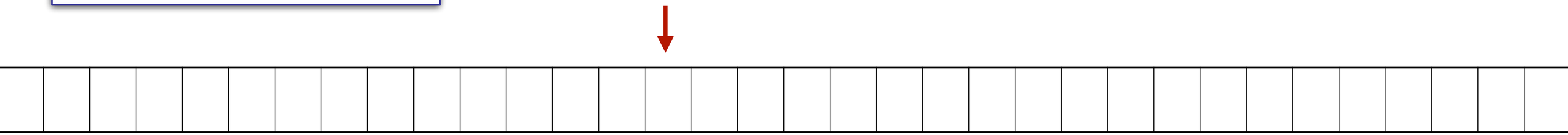
This special state is a reject state. When a TM enters a reject state, it immediately stop running and rejects whatever the original input string was – in this case, aaaaa.

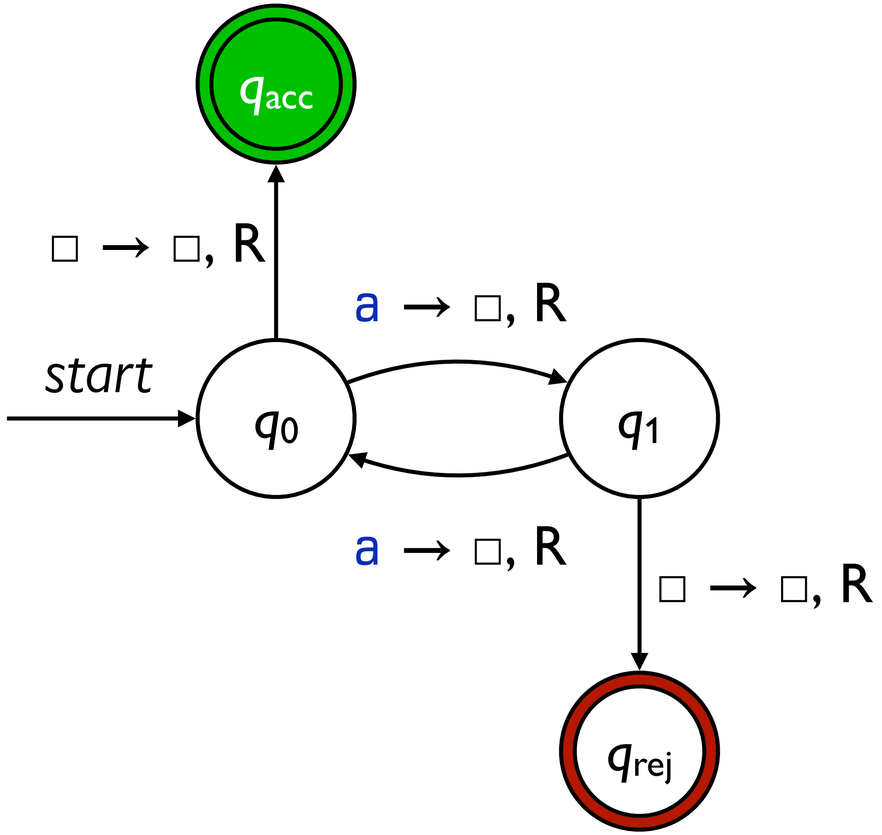


If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.

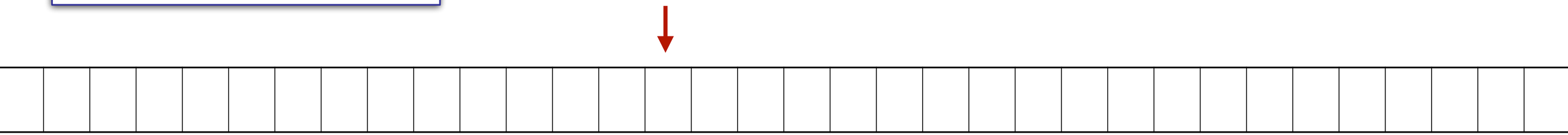


If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.





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.



Defining a Turing machine

A Turing machine is $(Q, \Sigma, \Gamma, \delta, q_o, q_{acc}, q_{rej})$, where

Q is a finite set of ***states***,

A Turing machine is $(Q, \Sigma, \Gamma, \delta, q_o, q_{acc}, q_{rej})$, where

Q is a finite set of *states*,

Σ is a finite *input alphabet*,

Γ is a finite *tape alphabet*, with $\Sigma \subset \Gamma$,

The tape alphabet Γ can contain any number of symbols but always includes at least one blank symbol, $\square \notin \Sigma$.

A Turing machine is $(Q, \Sigma, \Gamma, \delta, q_o, q_{acc}, q_{rej})$, where

Q is a finite set of *states*,

Σ is a finite *input alphabet*,

Γ is a finite *tape alphabet*, with $\Sigma \subset \Gamma$,

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the *transition function,*

At each step, the Turing machine
Writes a symbol to the tape cell under the tape head,
Changes state, and
Moves the tape head left or right.

A Turing machine is $(Q, \Sigma, \Gamma, \delta, q_o, q_{acc}, q_{rej})$, where

Q is a finite set of *states*,

Σ is a finite *input alphabet*,

Γ is a finite *tape alphabet*, with $\Sigma \subset \Gamma$,

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the *transition function,*

$q_o \in Q$ is the *start state*,

$q_{acc} \in Q$ is the *accept state*, and

$q_{rej} \in Q$, $q_{rej} \neq q_{acc}$ is the *reject state*.

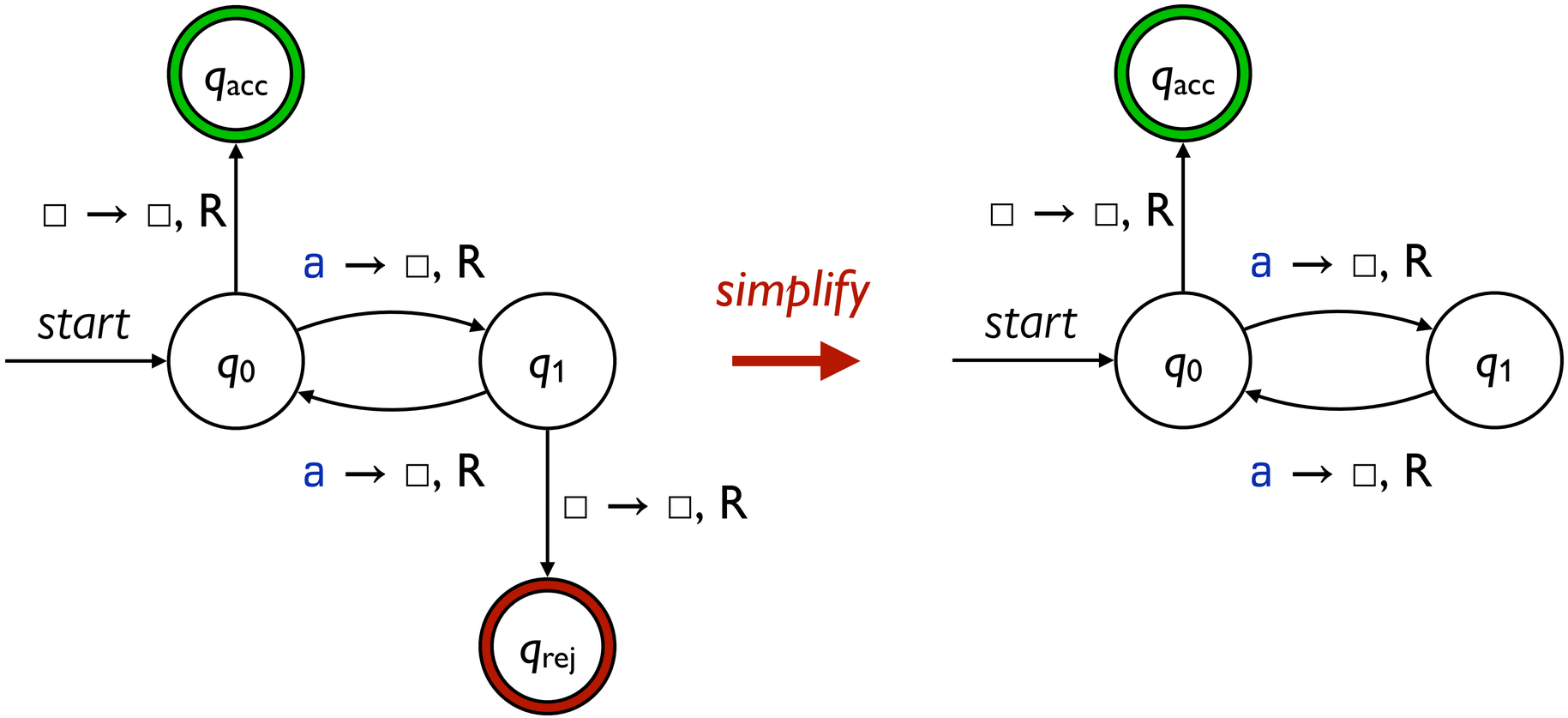
Unlike the automata we've seen before, Turing machines can revisit characters from the input, and the machine doesn't need to read the entire input.

Turing machines decide when – and if! – they will accept or reject their input.

Turing machines are *deterministic*.

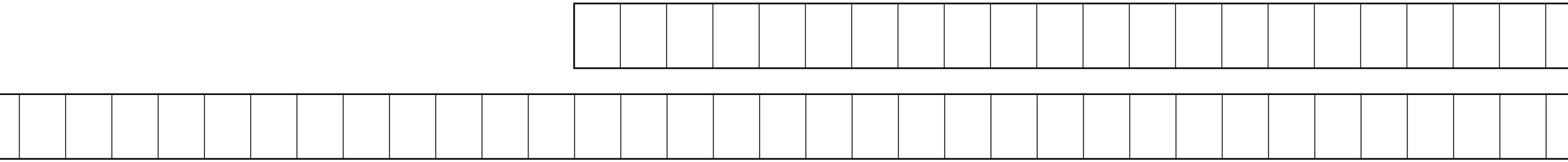
For every combination of a (non-accepting, non-rejecting) state q and a tape symbol $a \in \Gamma$, there must be exactly one transition defined for that combination of q and a.

Any transitions that are missing implicitly go straight to a rejecting state. We'll use this later to simplify our designs.

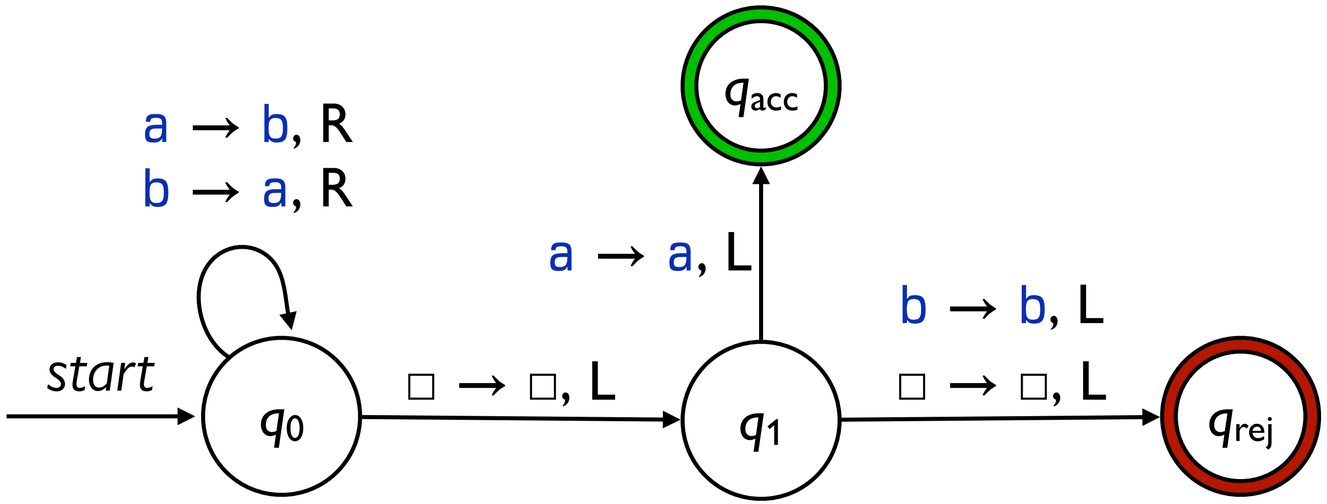


Warning: The textbook treats the tape of a Turing machine as only being infinite to the right.

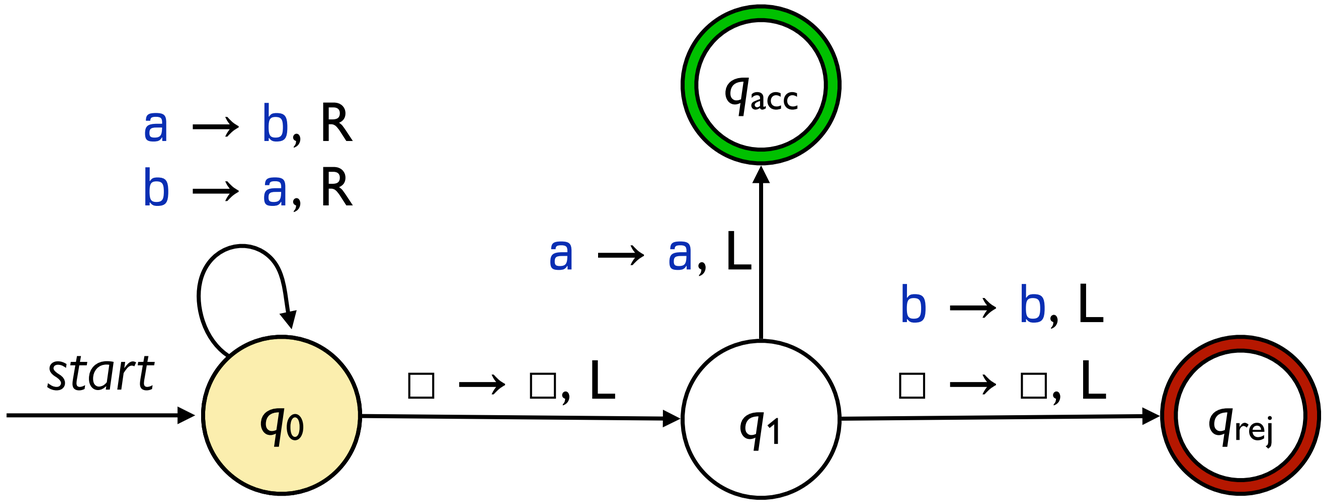
This is no less powerful than a tape that's infinite in both directions, but it's more annoying to use.

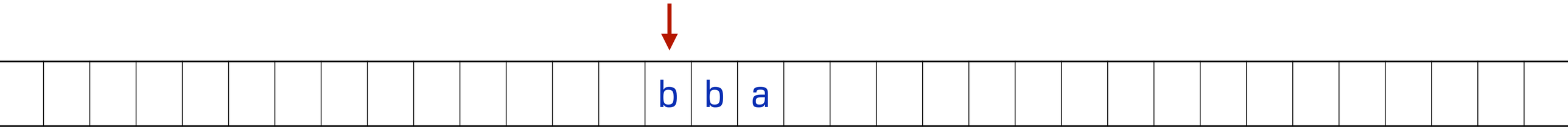


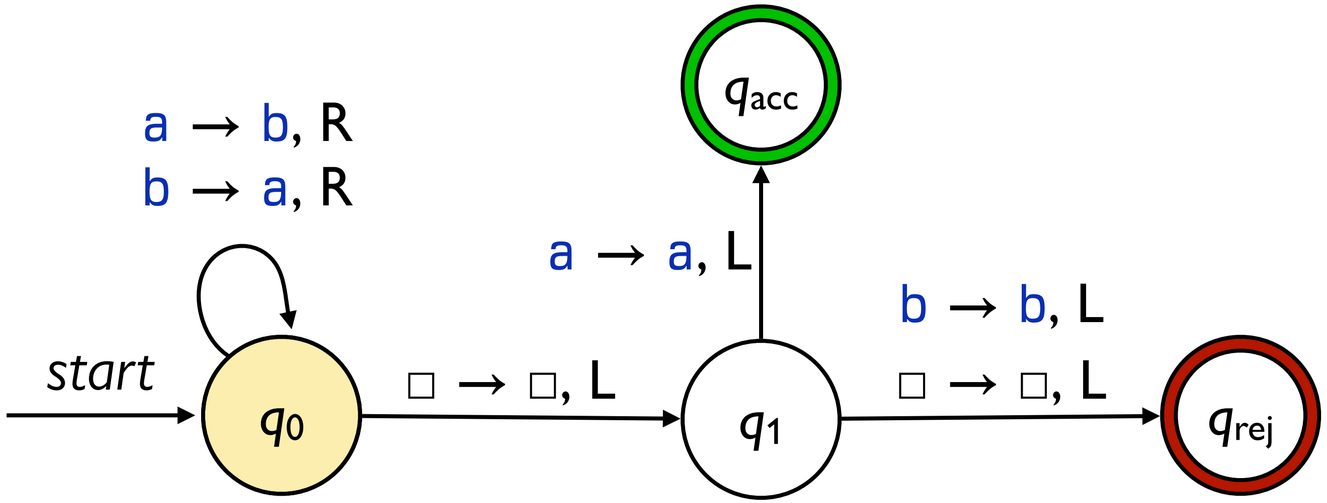
The language of a Turing machine

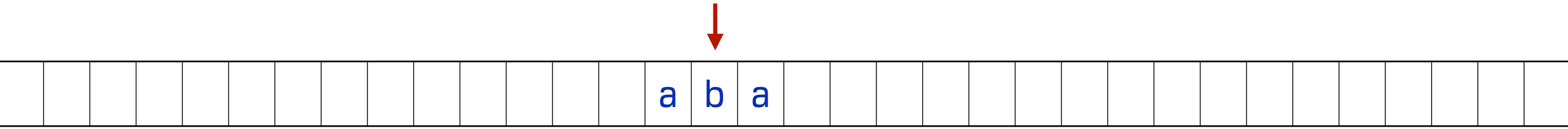


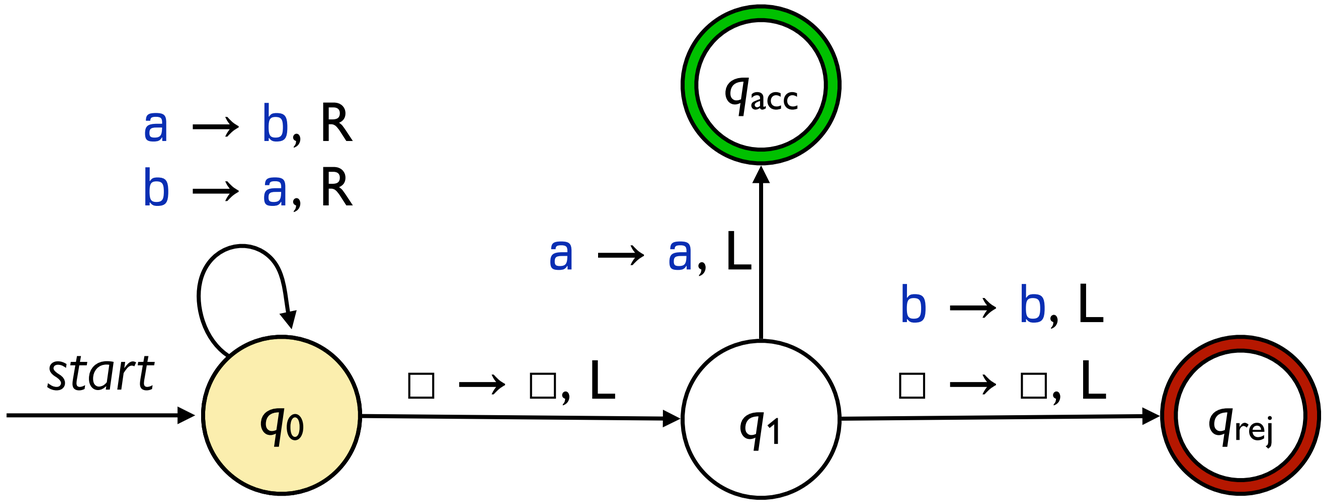
Run the TM shown above on the input string bba.
What will the tape look like when the TM finishes running?

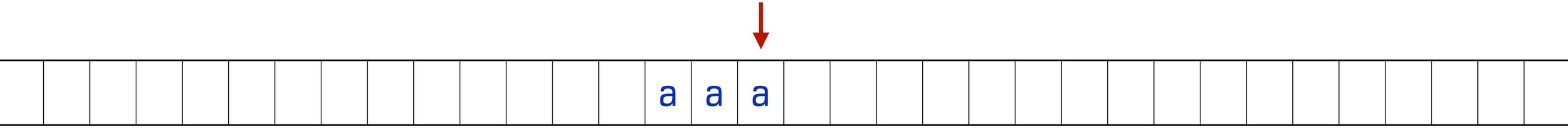

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

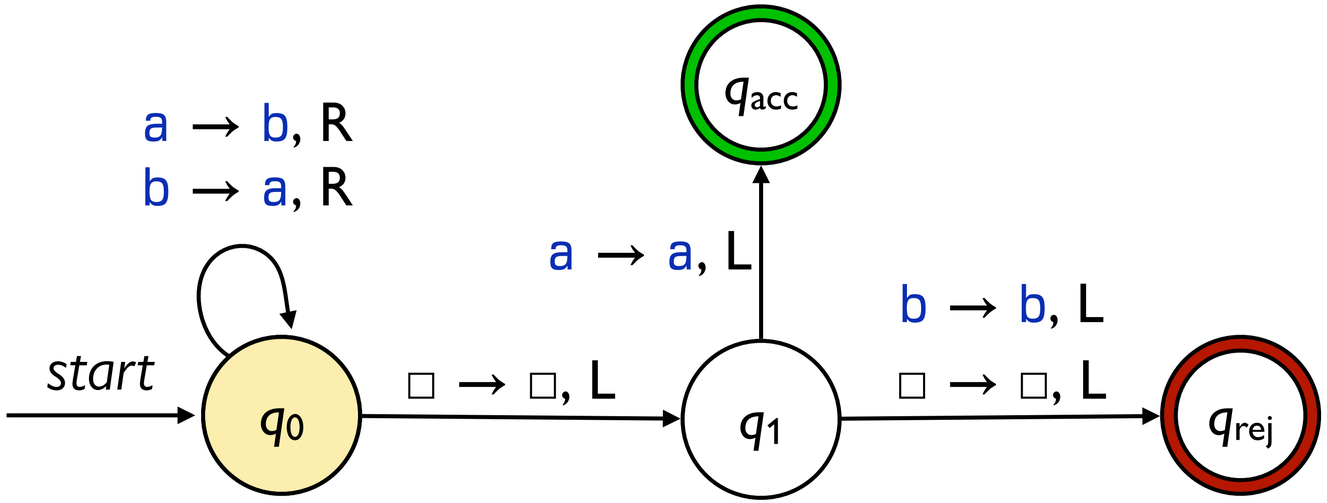

b b a

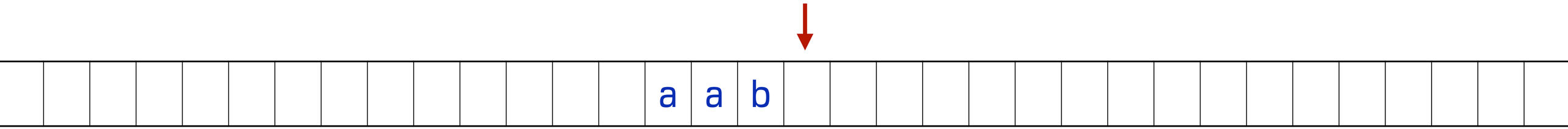

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

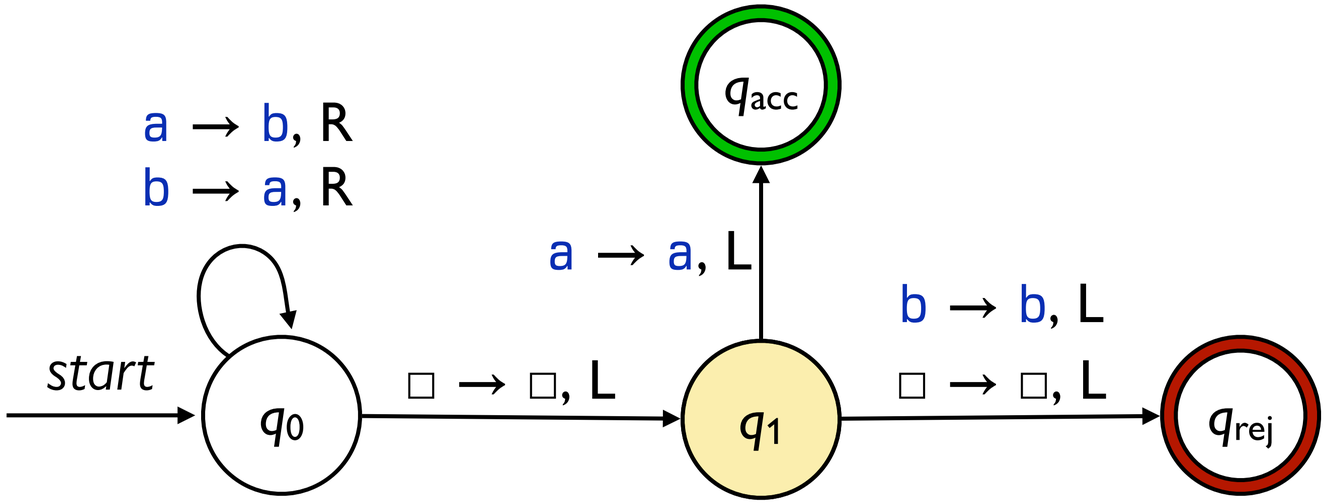

a
b
a

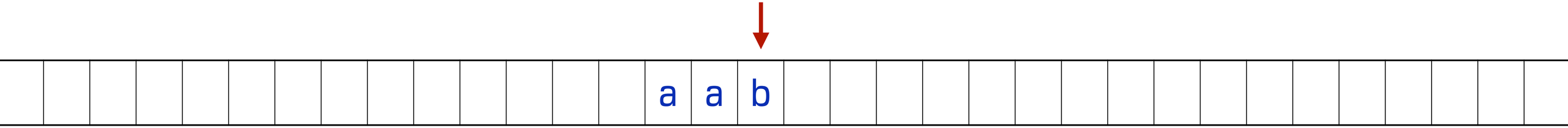

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

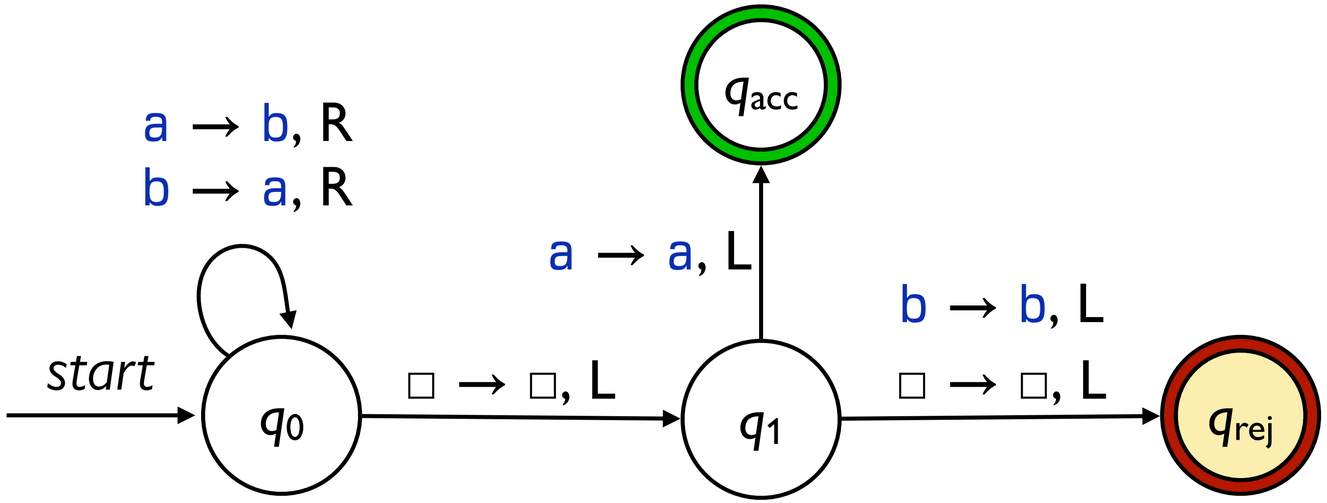

a a a

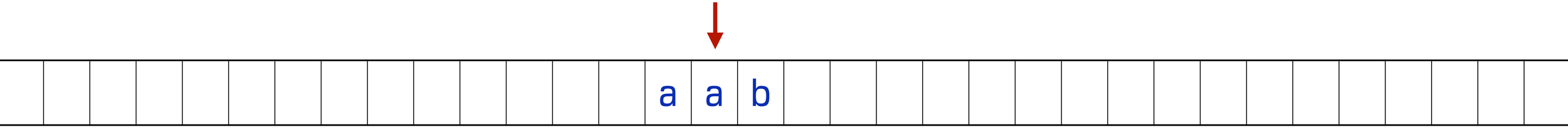

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

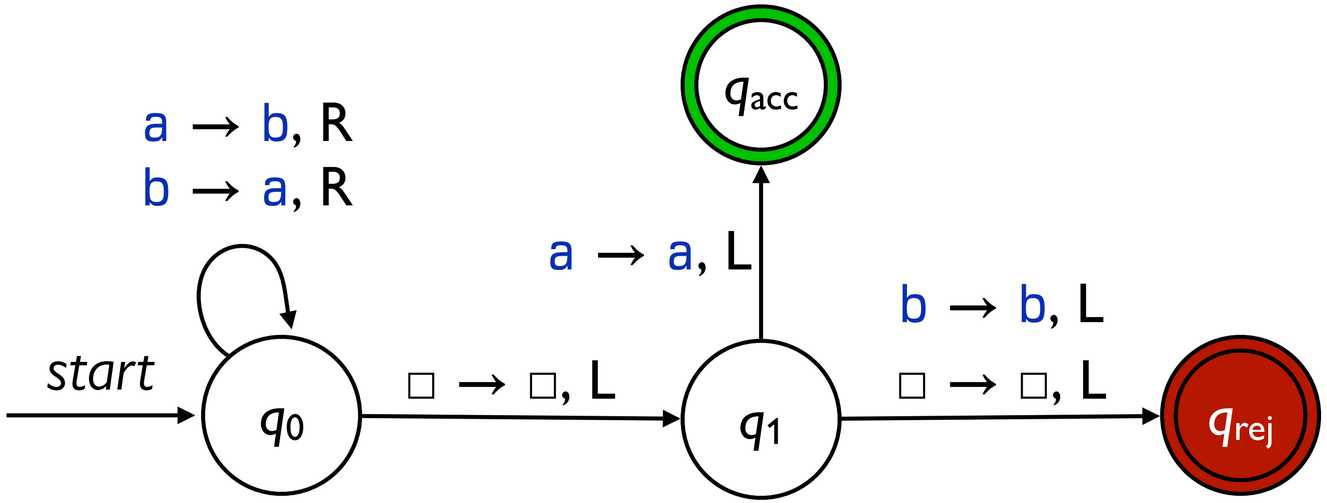

a
a
b

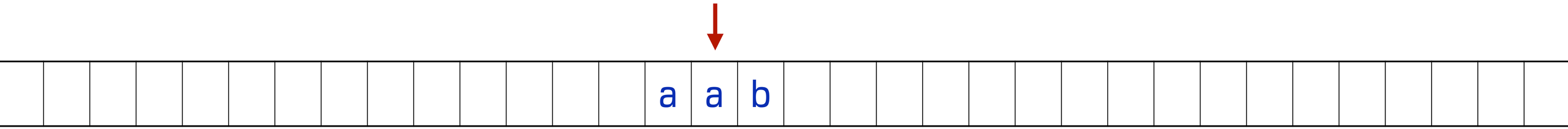

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

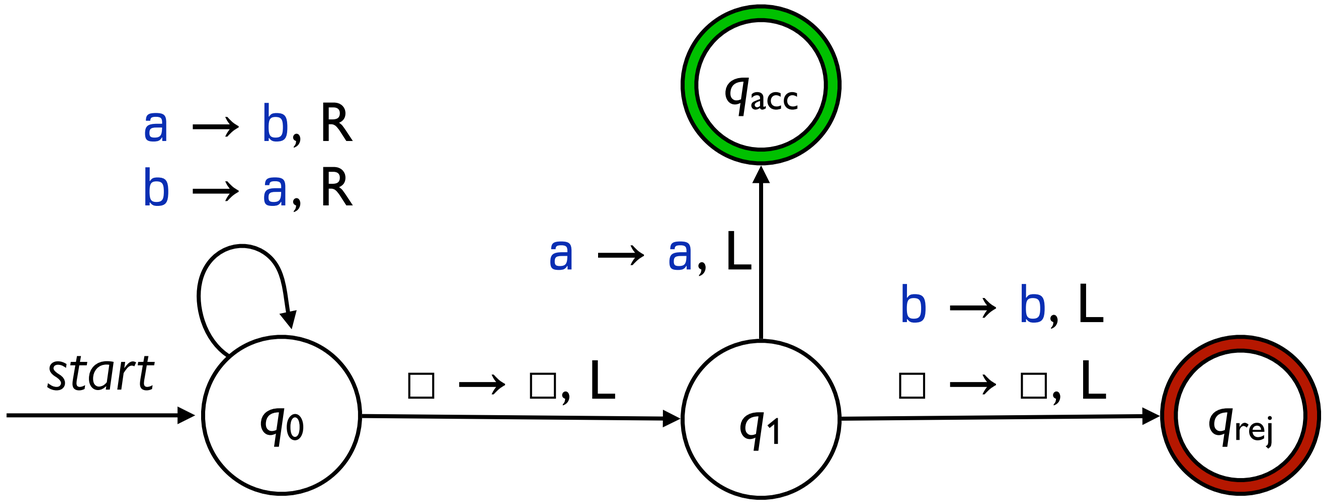

a a b


q_{acc}
a → b, R
b → a, R
a → a, L
b → b, L
start
□ → □, L
□ → □, L
q_0
q_1
q_{rej}


a a b


q_{acc}
a → b, R
b → a, R
a → a, L
b → b, L
start
□ → □, L
□ → □, L
q_0
q_1
q_{rej}

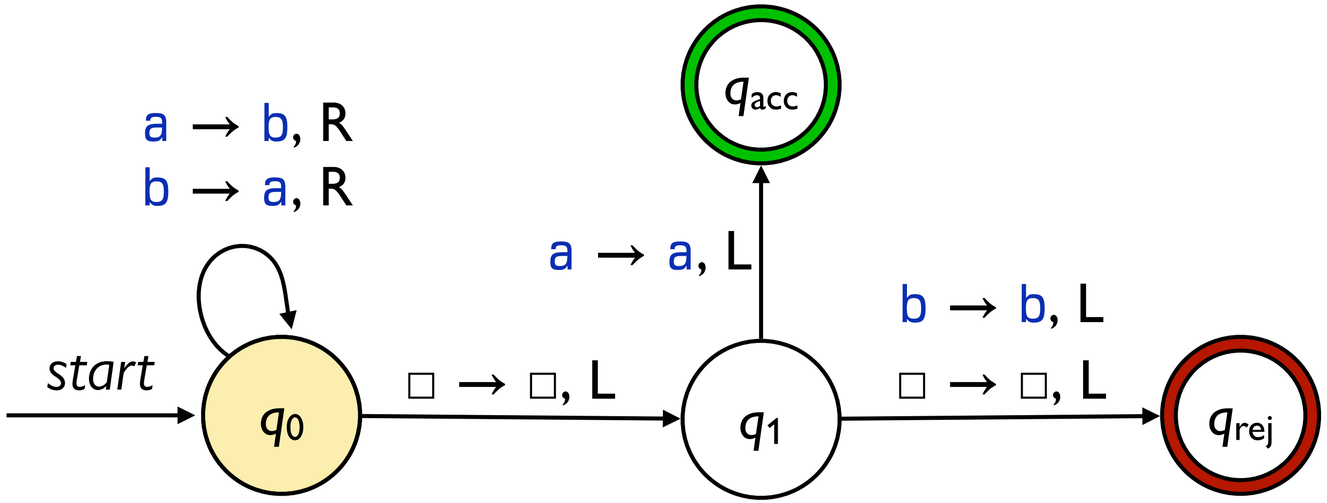

a a b

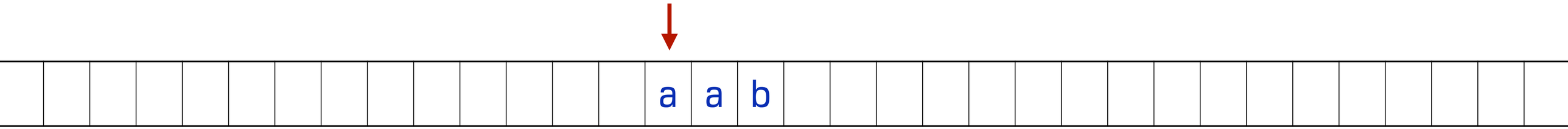


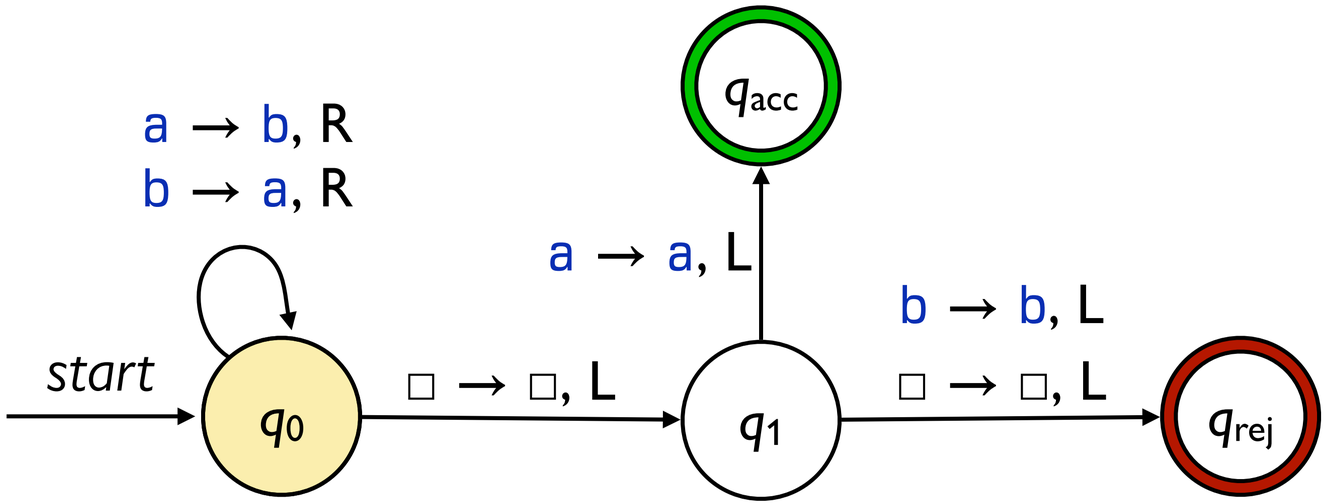
If M is a Turing machine with input alphabet Σ, then the language of M, denoted L(M), is the set

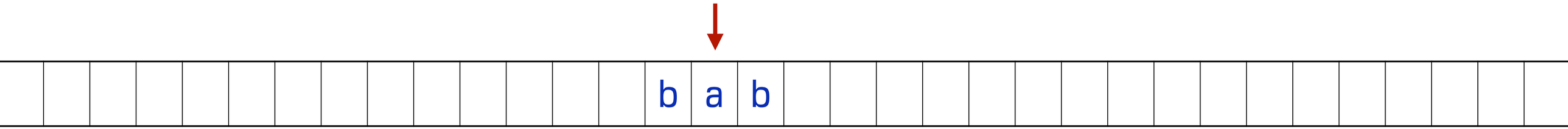
$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

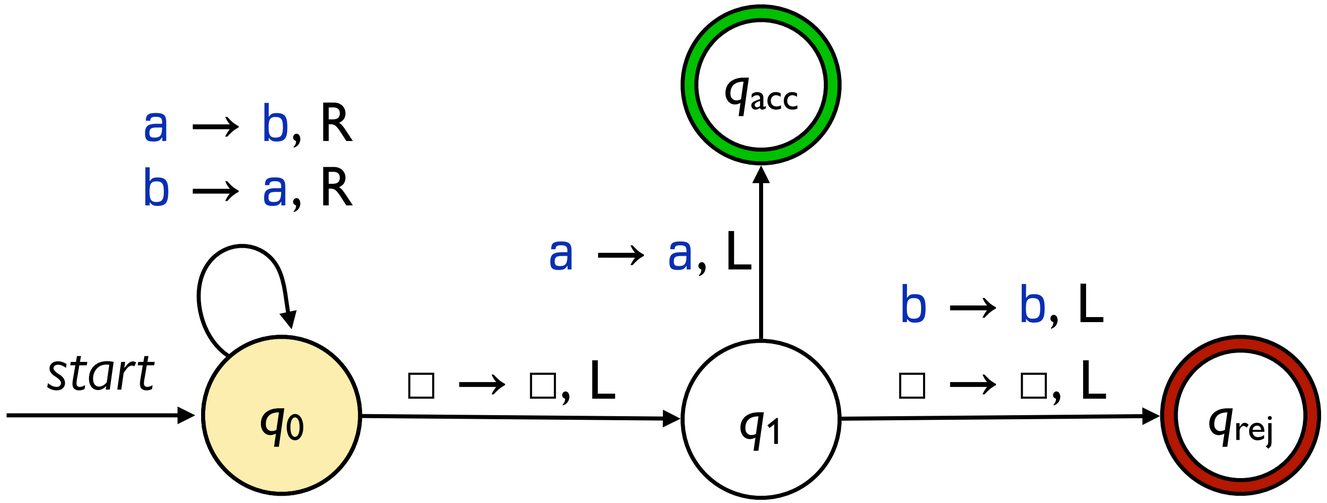
What is L(M), where M is the above TM?

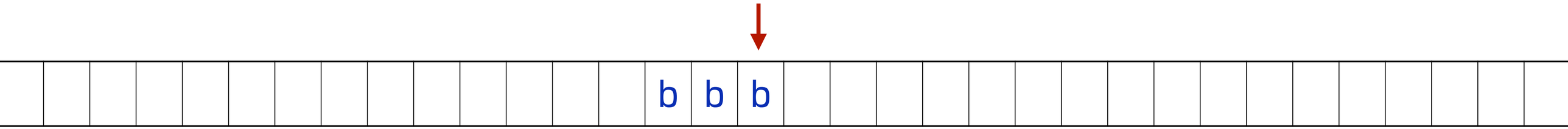

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

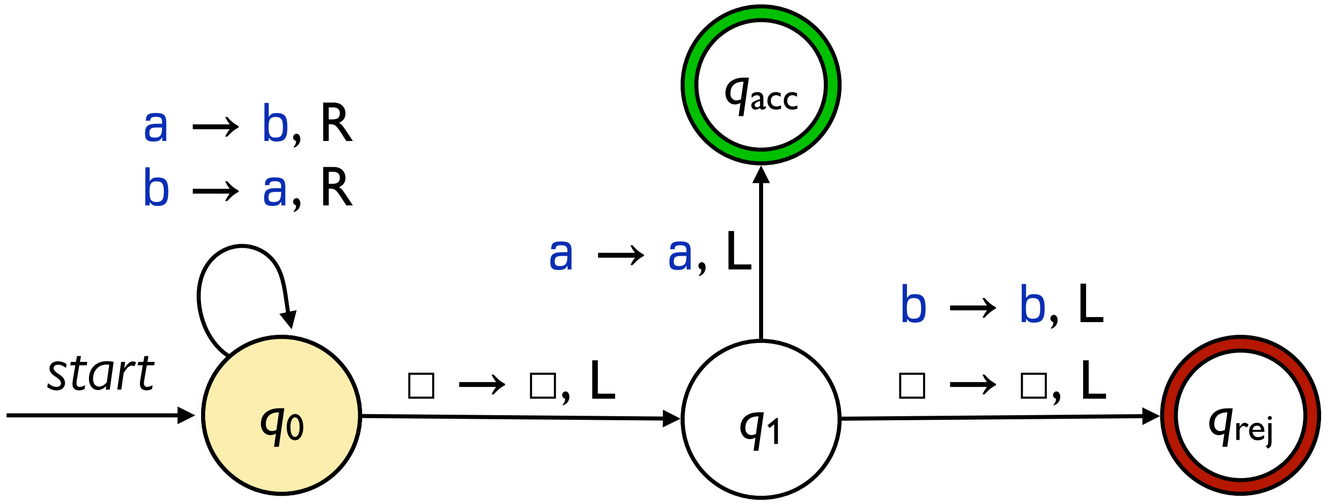

a a b

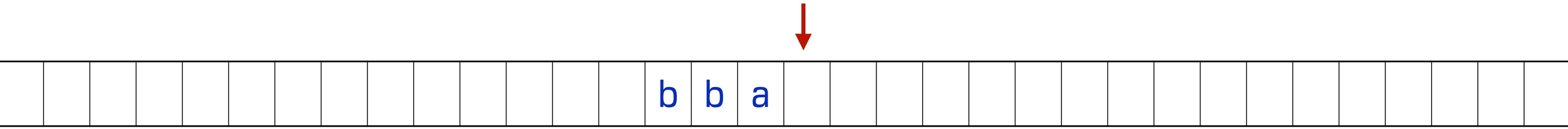

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

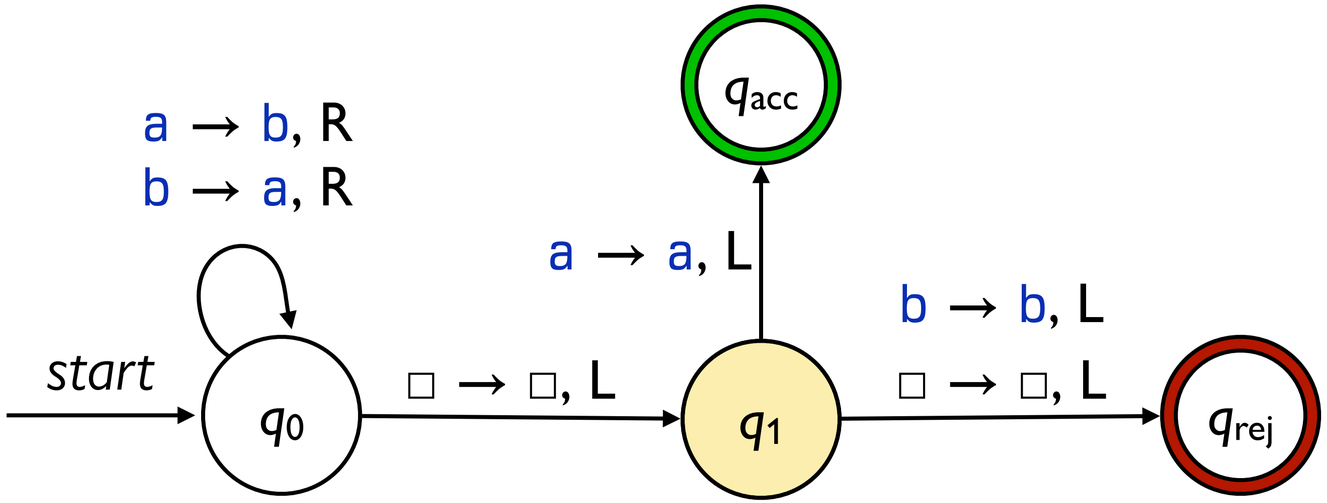

b
a
b

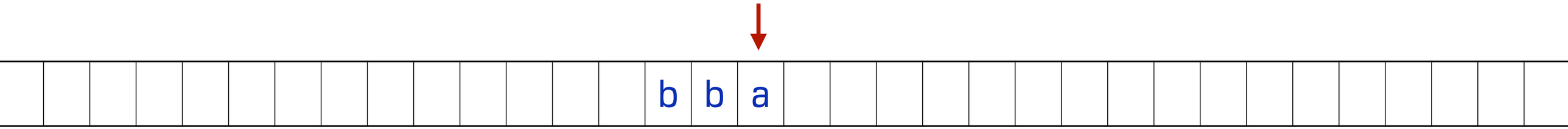

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

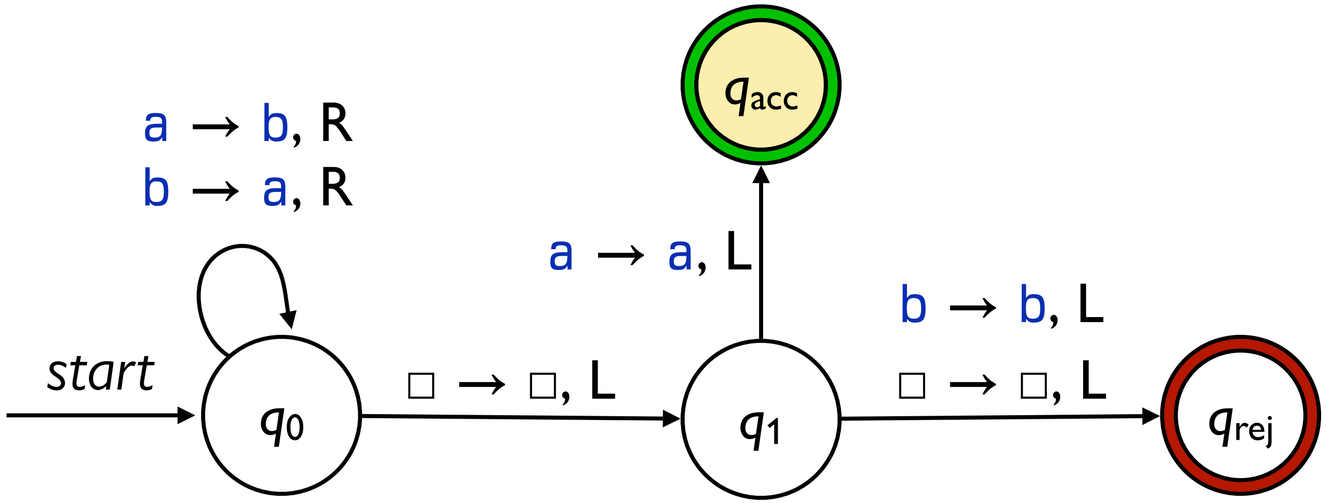

b b b

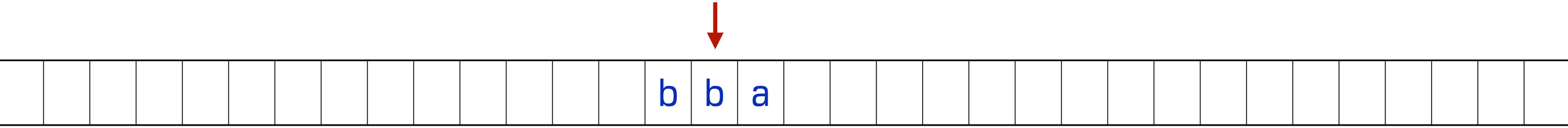

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej

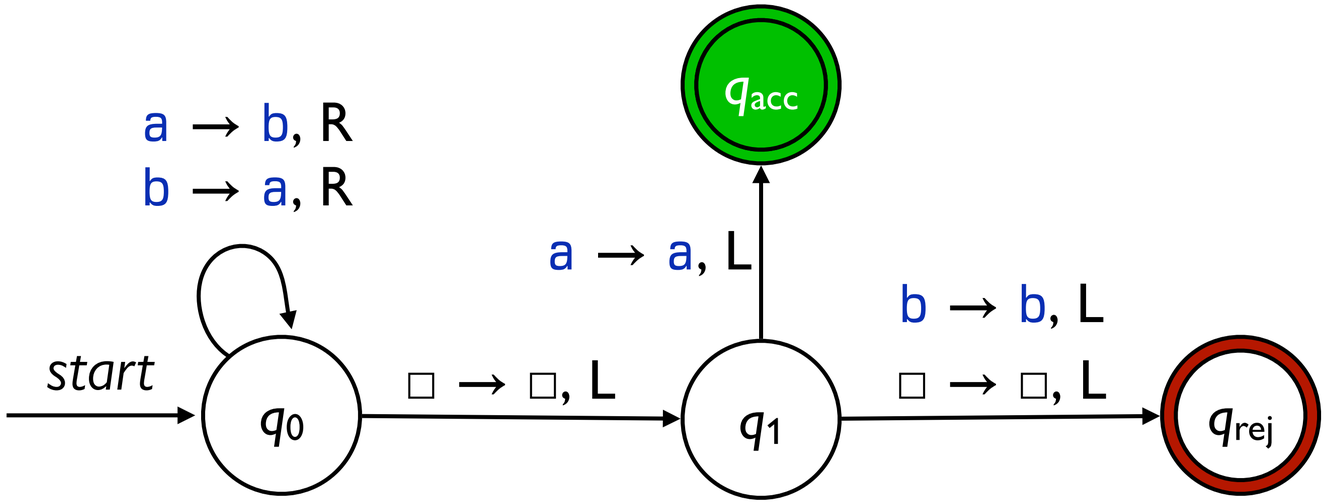

b
b
a


a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej


b
b
a

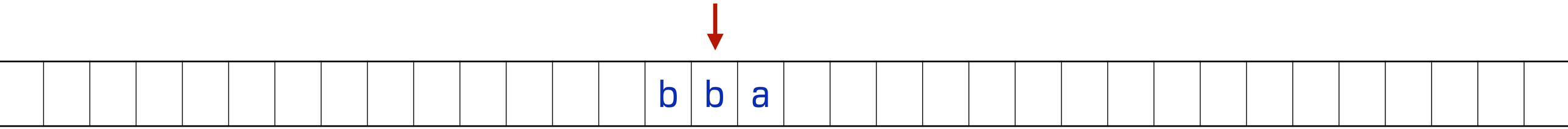

q_{acc}
a → b, R
b → a, R
a → a, L
b → b, L
start
□ → □, L
□ → □, L
q_0
q_1
q_{rej}


b b a



Although the tape ends with bba *written on it, the original input string was* aab*. This shows that the TM accepts* aab*, not* bba*.*

So $L(M) = \{w \in \{a, b\}^* \mid w$ ends in b$\}$



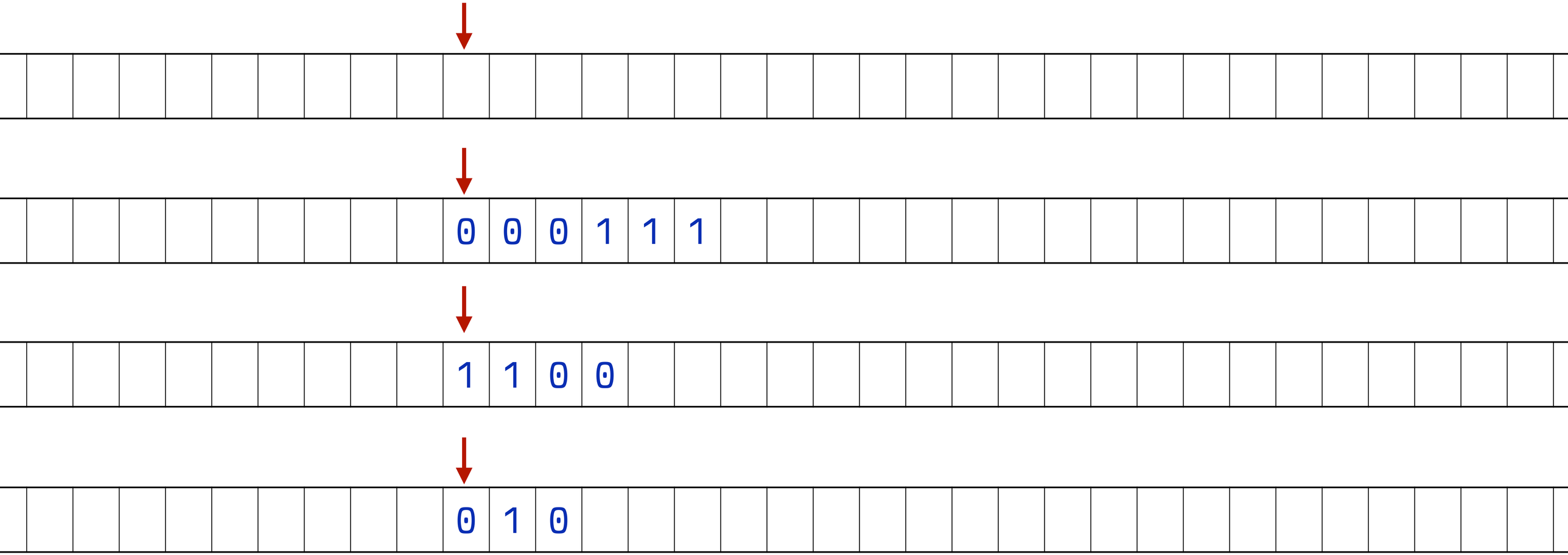
Designing Turing machines

Let $\Sigma = \{\texttt{0}, \texttt{1}\}$ and consider the language $L = \{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}_0\}$.

We know that L is context-free.

How could we build a Turing machine for it?

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



A recursive approach

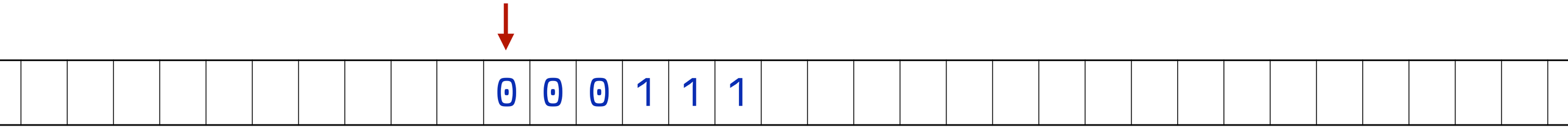
The string `ε` is in *L*.

The string `0`*w*`1` is in *L* iff *w* is in *L*.

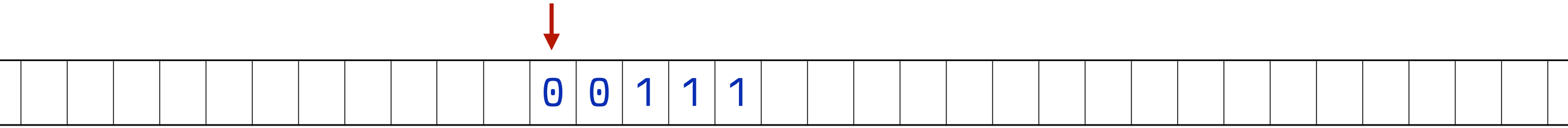
Any string starting with `1` is not in *L*.

Any string ending with `0` is not in *L*.

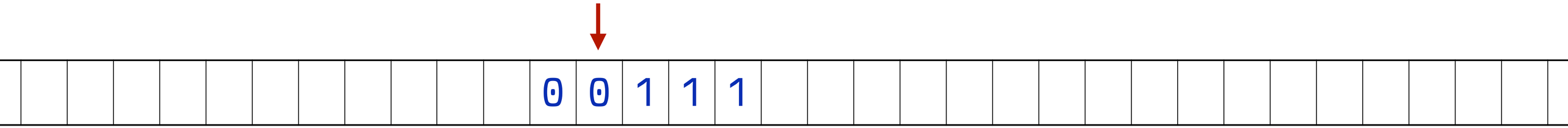
A sketch of the TM



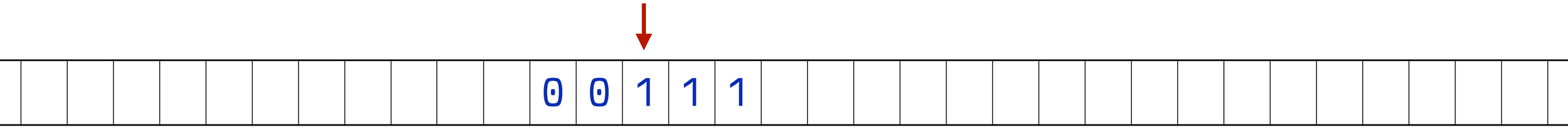
A sketch of the TM



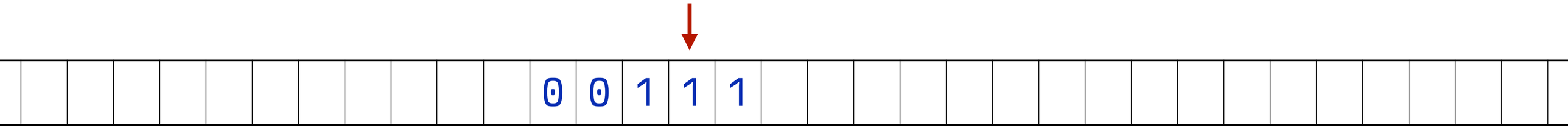
A sketch of the TM



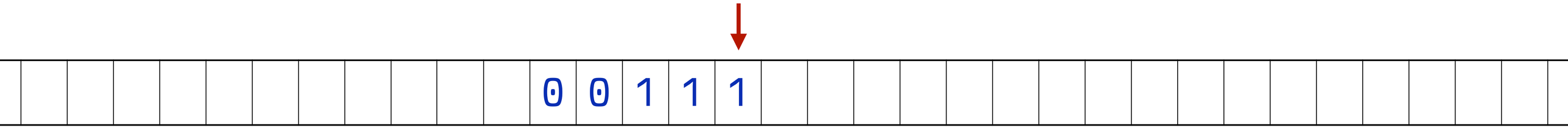
A sketch of the TM



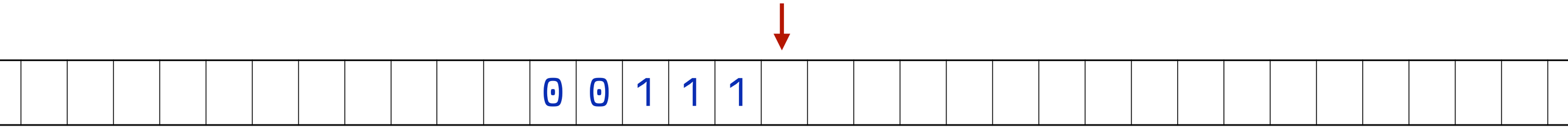
A sketch of the TM



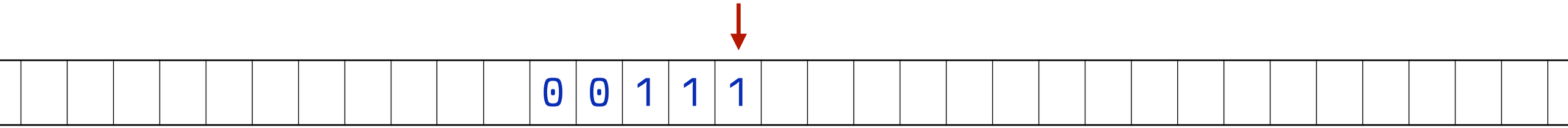
A sketch of the TM



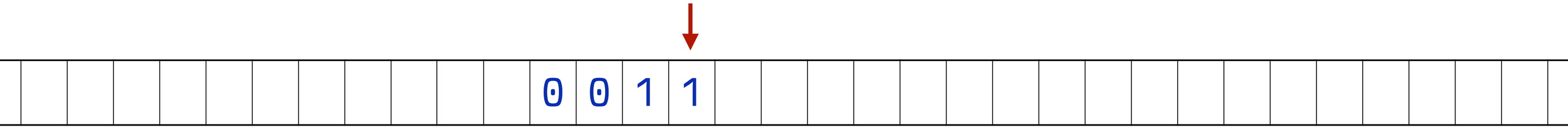
A sketch of the TM



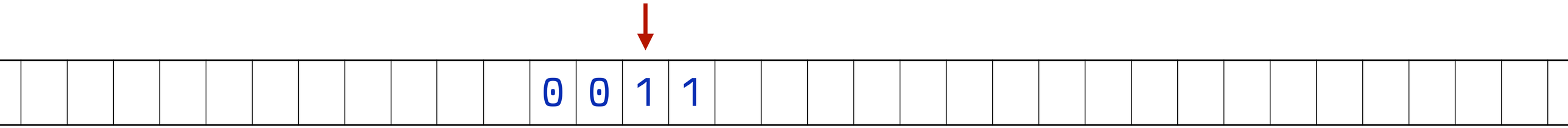
A sketch of the TM



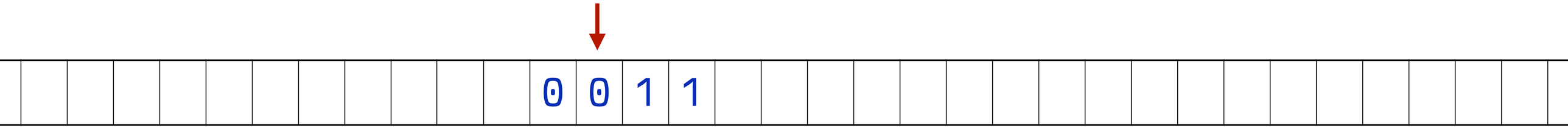
A sketch of the TM



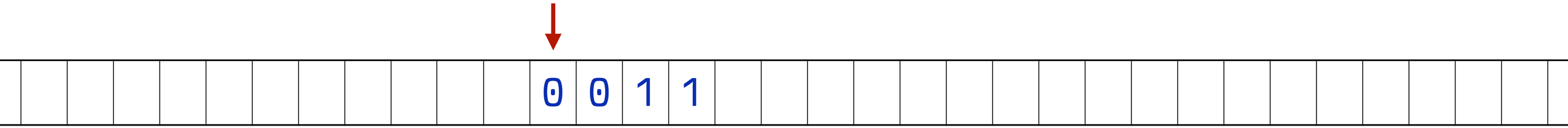
A sketch of the TM



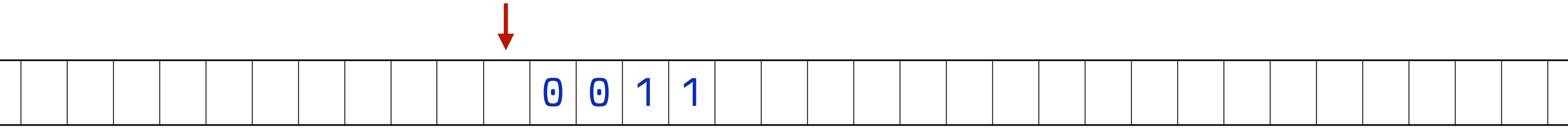
A sketch of the TM



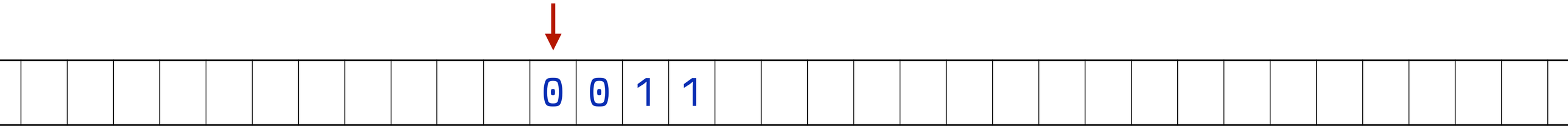
A sketch of the TM



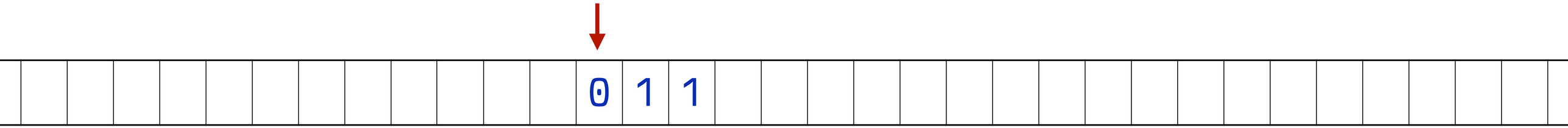
A sketch of the TM



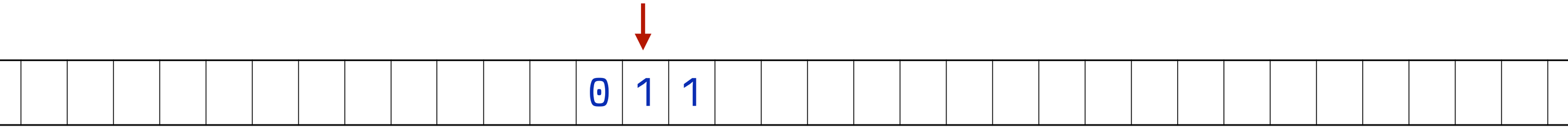
A sketch of the TM



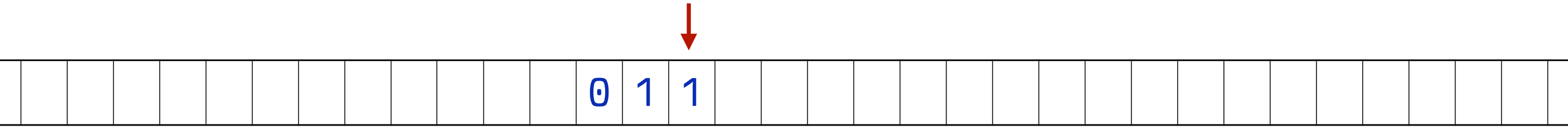
A sketch of the TM



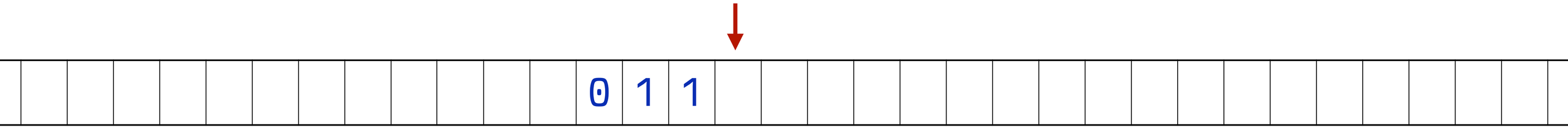
A sketch of the TM



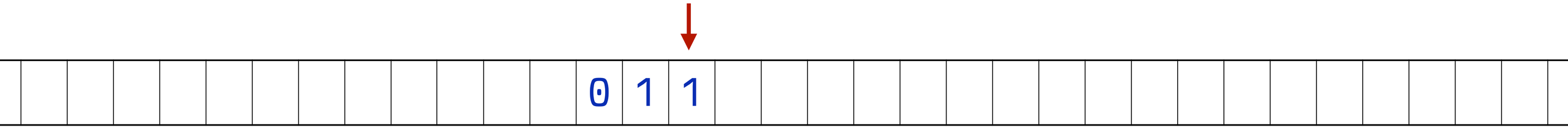
A sketch of the TM



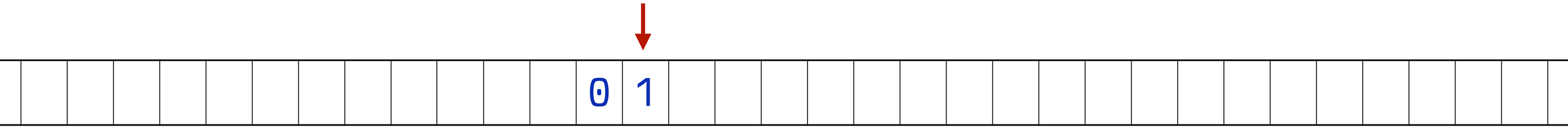
A sketch of the TM



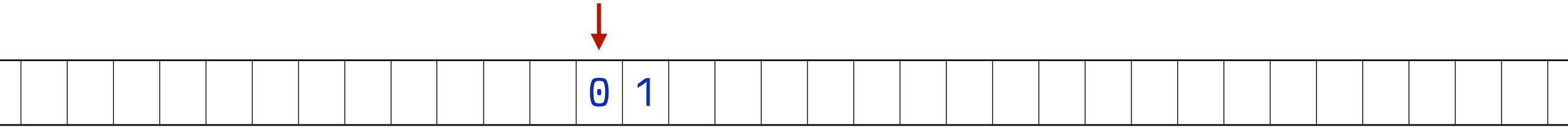
A sketch of the TM



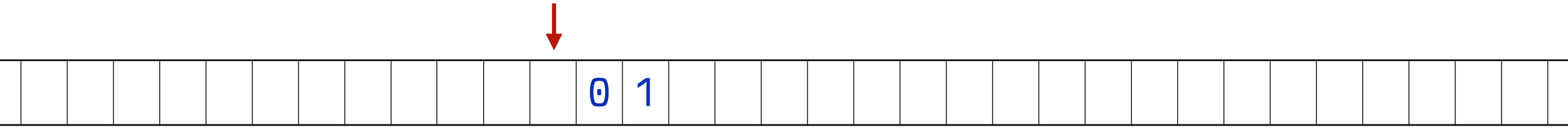
A sketch of the TM



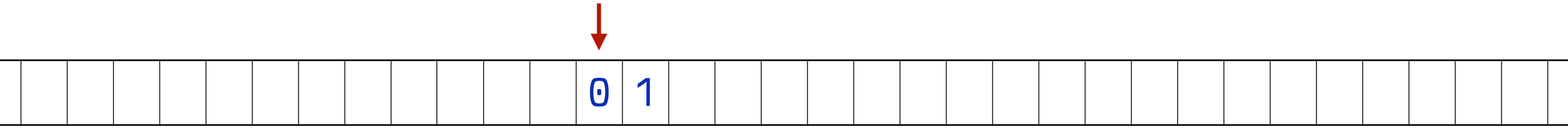
A sketch of the TM



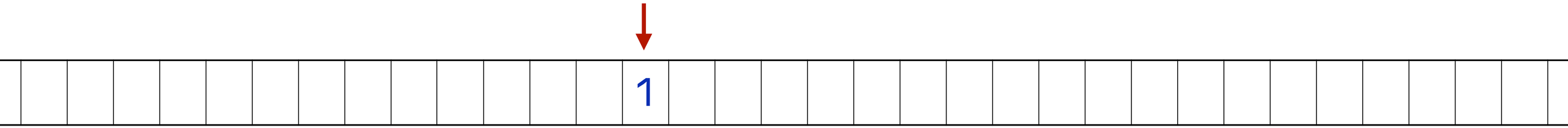
A sketch of the TM



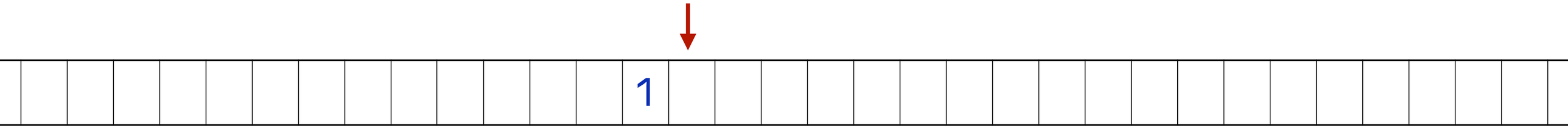
A sketch of the TM



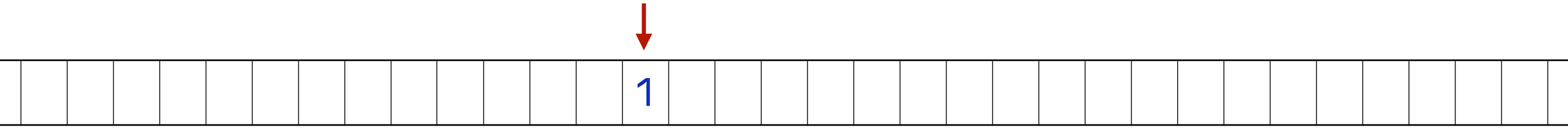
A sketch of the TM



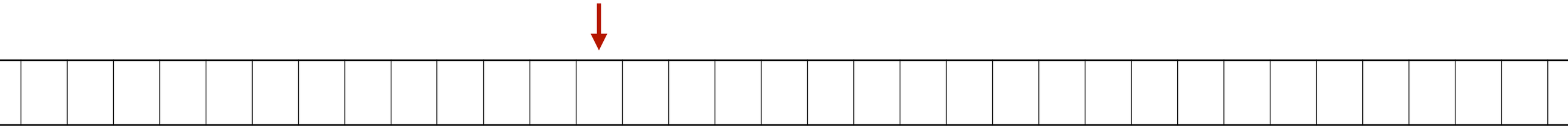
A sketch of the TM



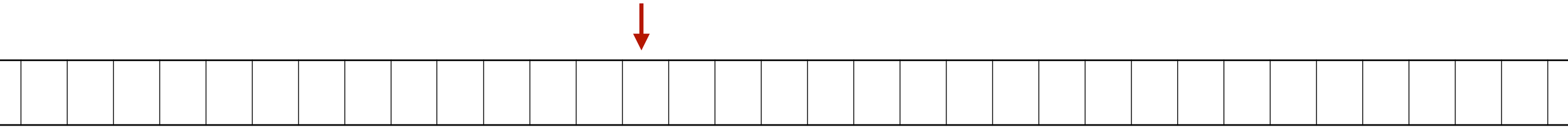
A sketch of the TM

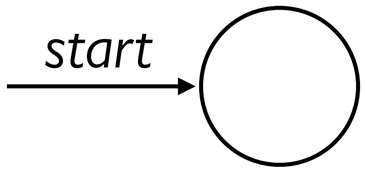


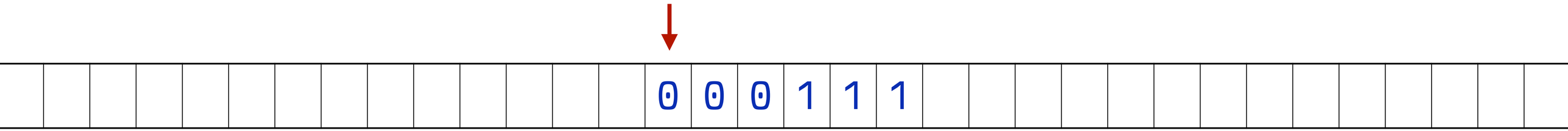
A sketch of the TM

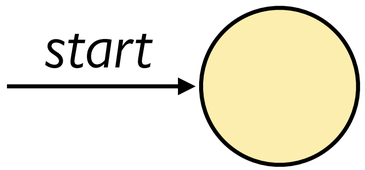


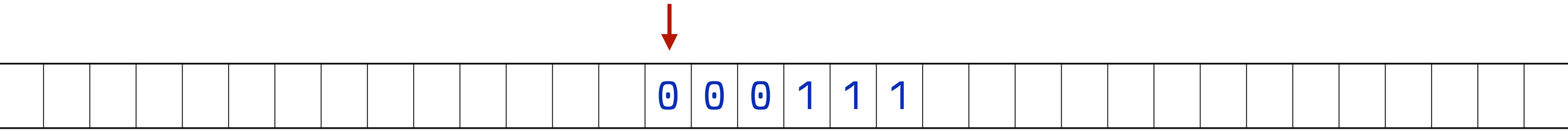
A sketch of the TM

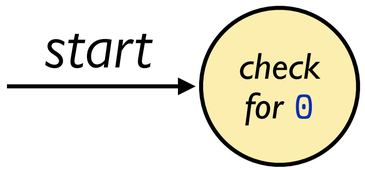


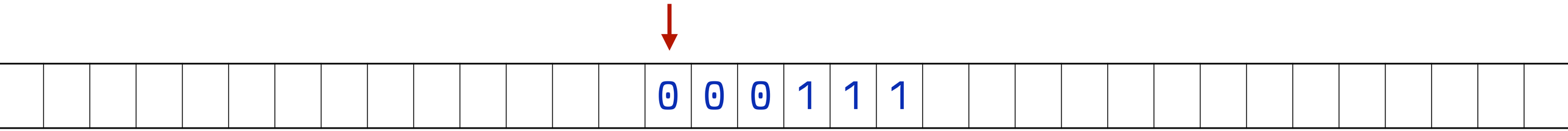

start

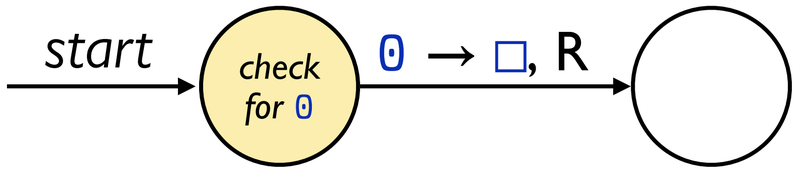

0 0 0 1 1 1

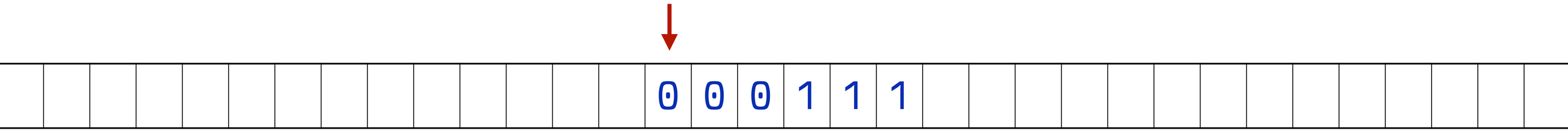

start

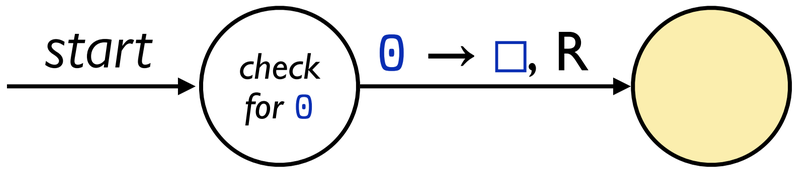

0 0 0 1 1 1

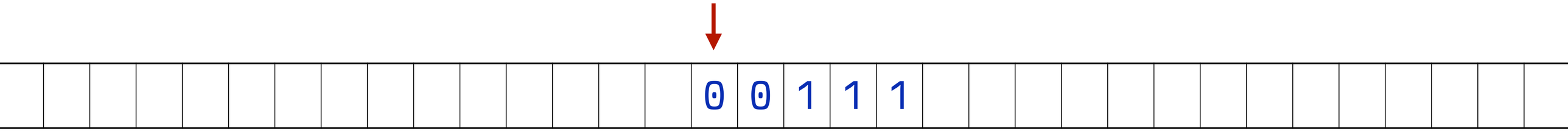

start
check
for 0

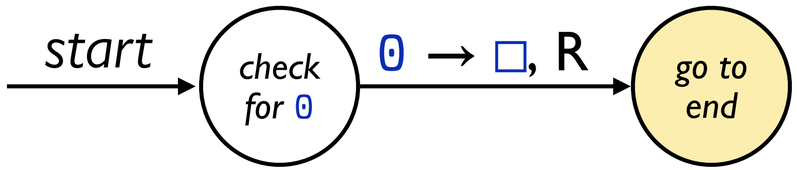

0 0 0 1 1 1

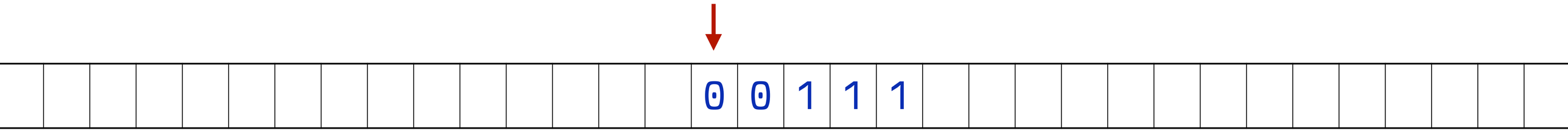

start
check for 0
0 → □, R

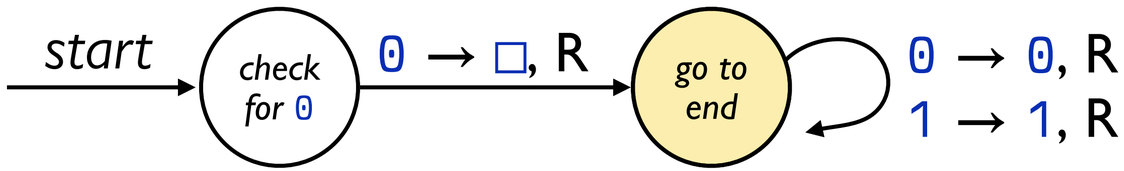

0 0 0 1 1 1

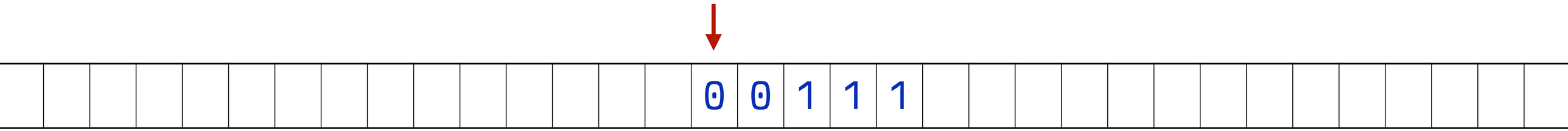

start
check for 0
0 → □, R

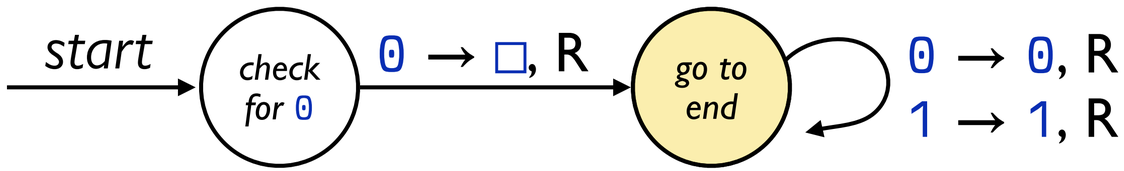

0 0 1 1 1

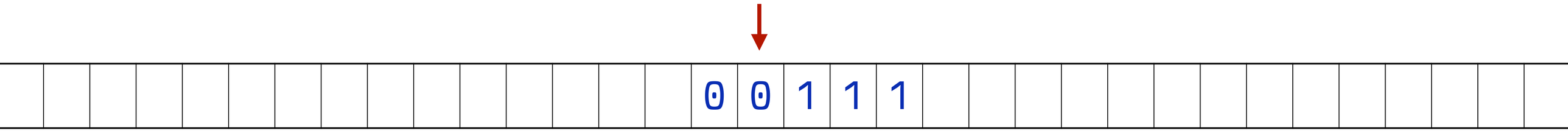

start
check for 0
0 → □, R
go to end

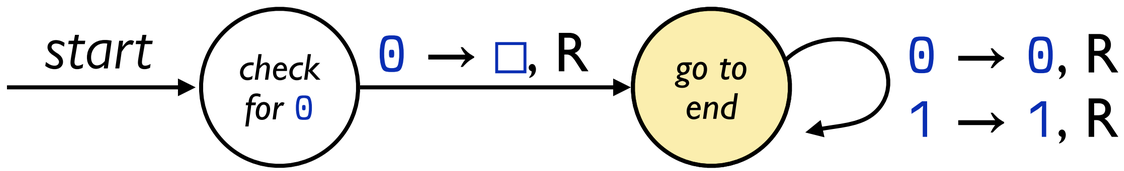

0 0 1 1 1

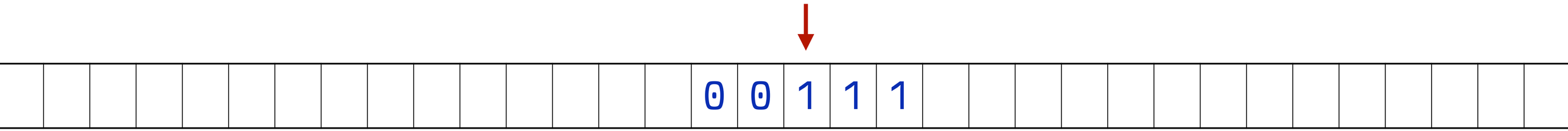

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

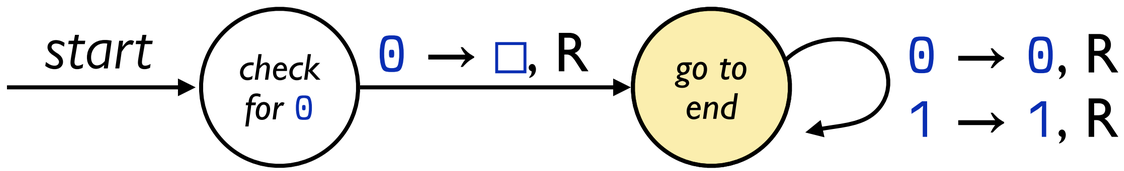

0 0 1 1 1

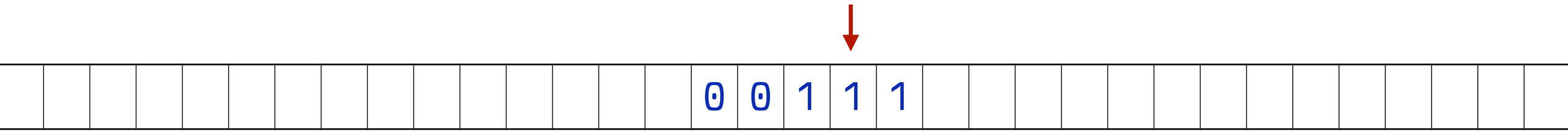

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

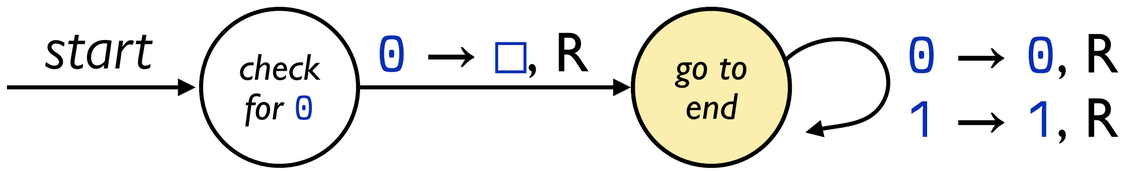

0 0 1 1 1

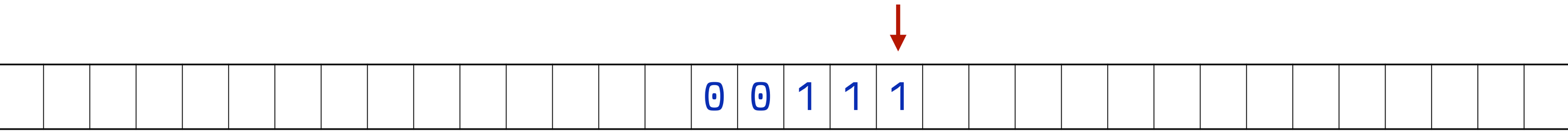

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

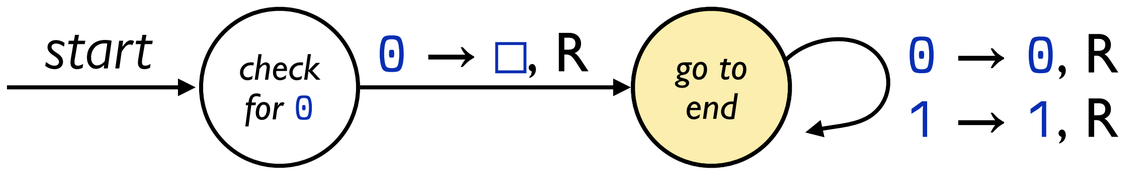

0 0 1 1 1

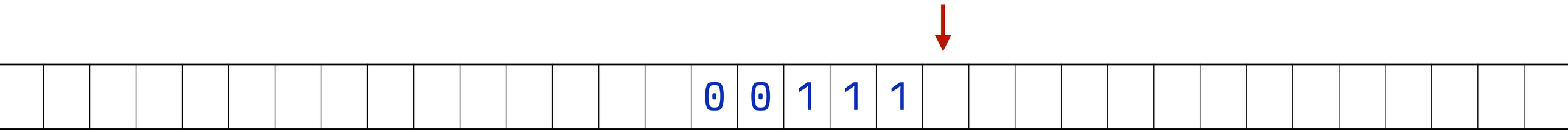

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

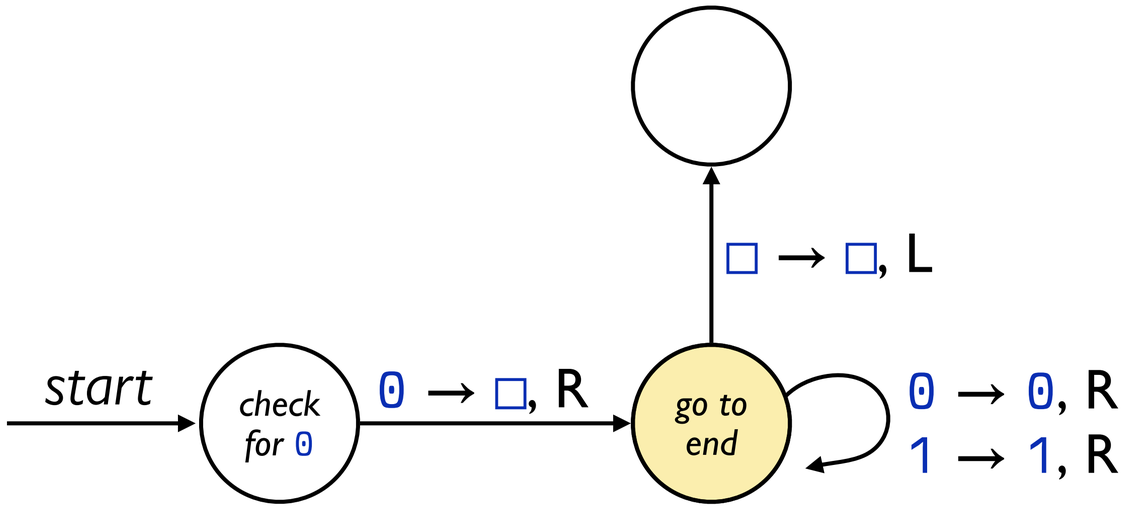

0 0 1 1 1

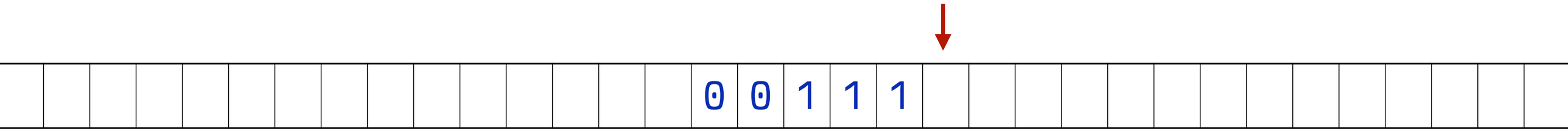

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

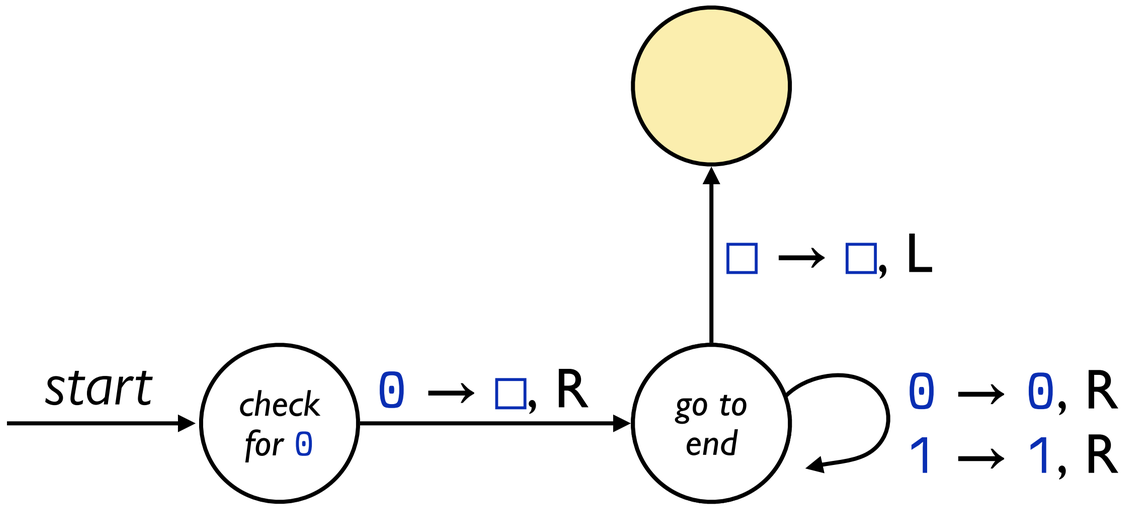

0 0 1 1 1

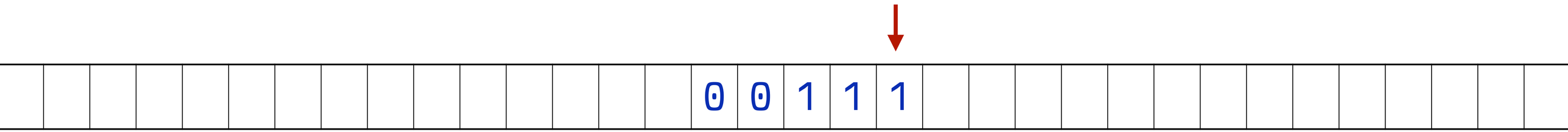

start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

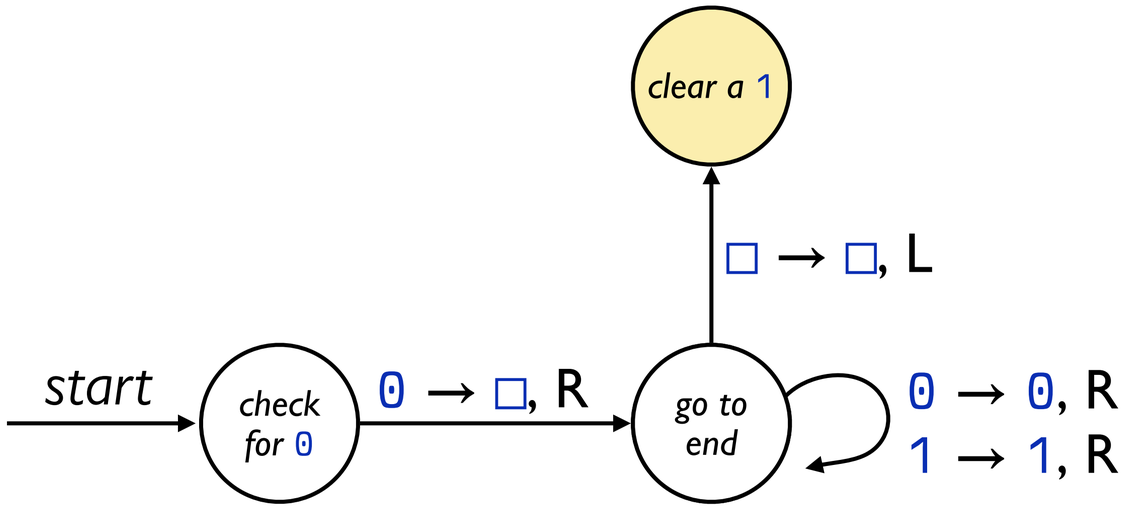

0 0 1 1 1

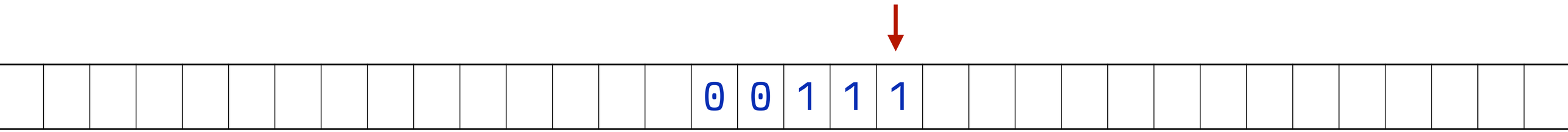

start
check for 0
0 → □, R
go to end
□ → □, L
0 → 0, R
1 → 1, R

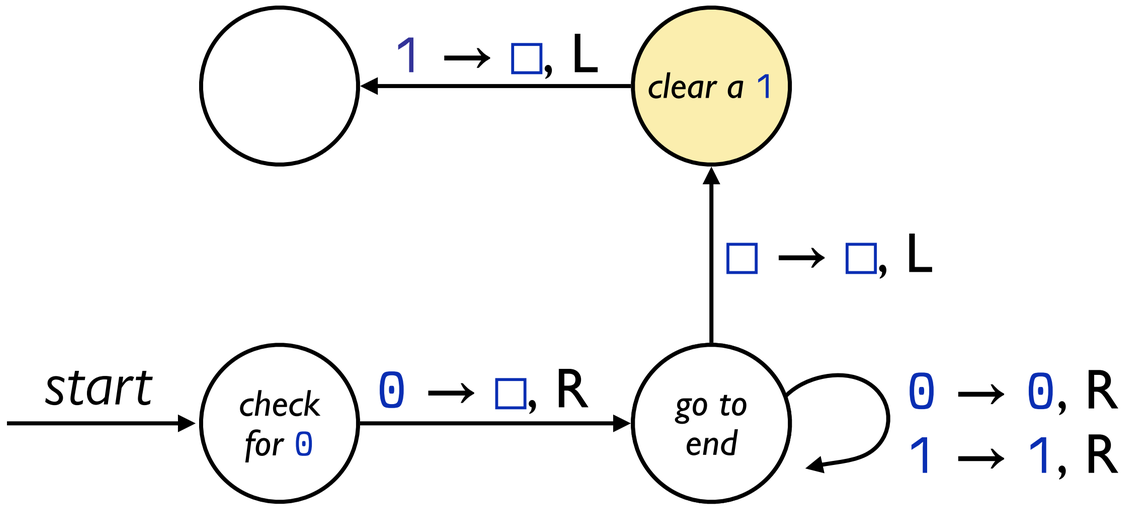

0 0 1 1 1

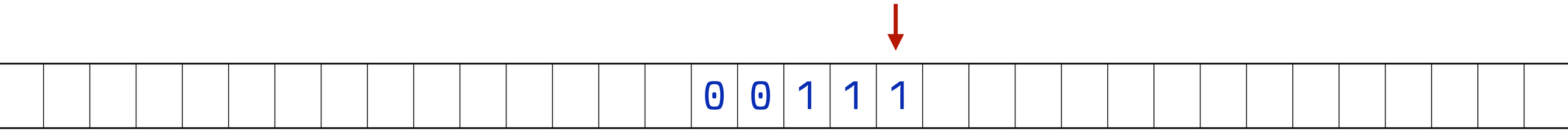

start
check for 0
0 → □, R
go to end
□ → □, L
0 → 0, R
1 → 1, R

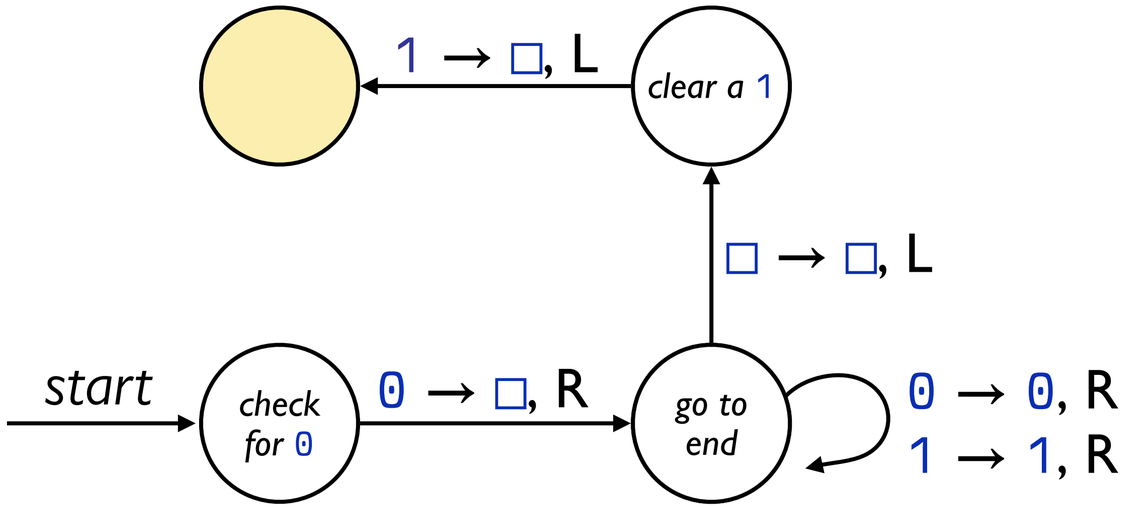

0 0 1 1 1

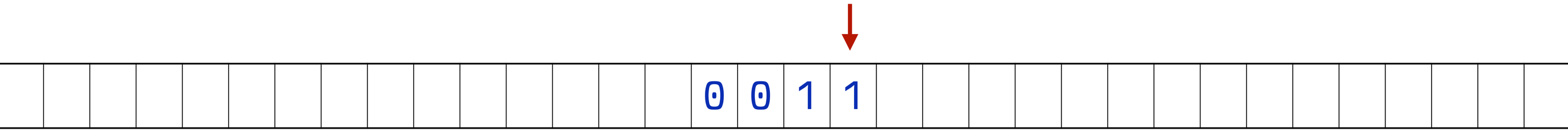

clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

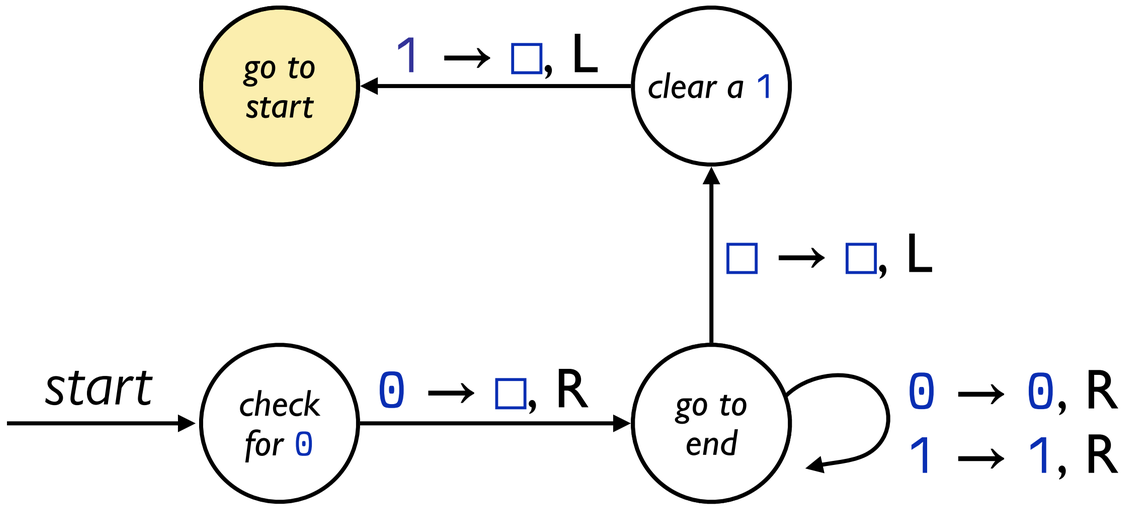

0 0 1 1 1

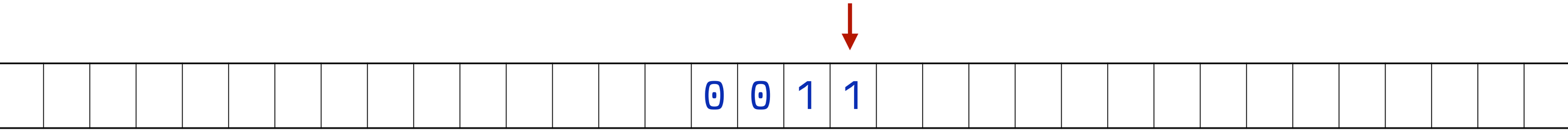

1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

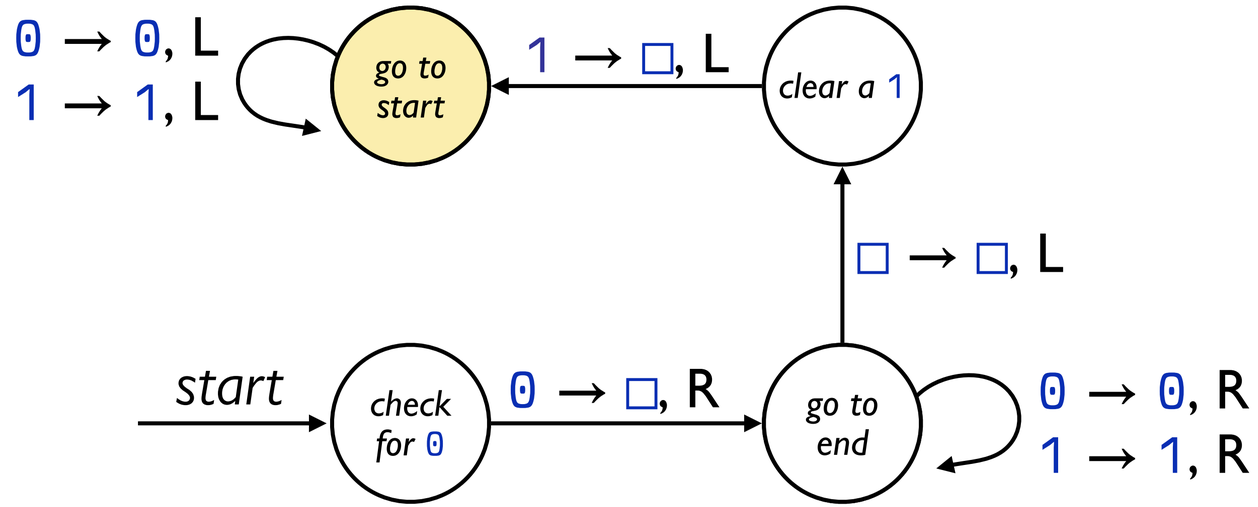

0 0 1 1 1

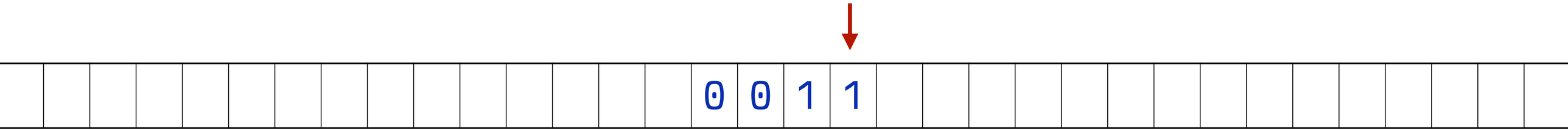

1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

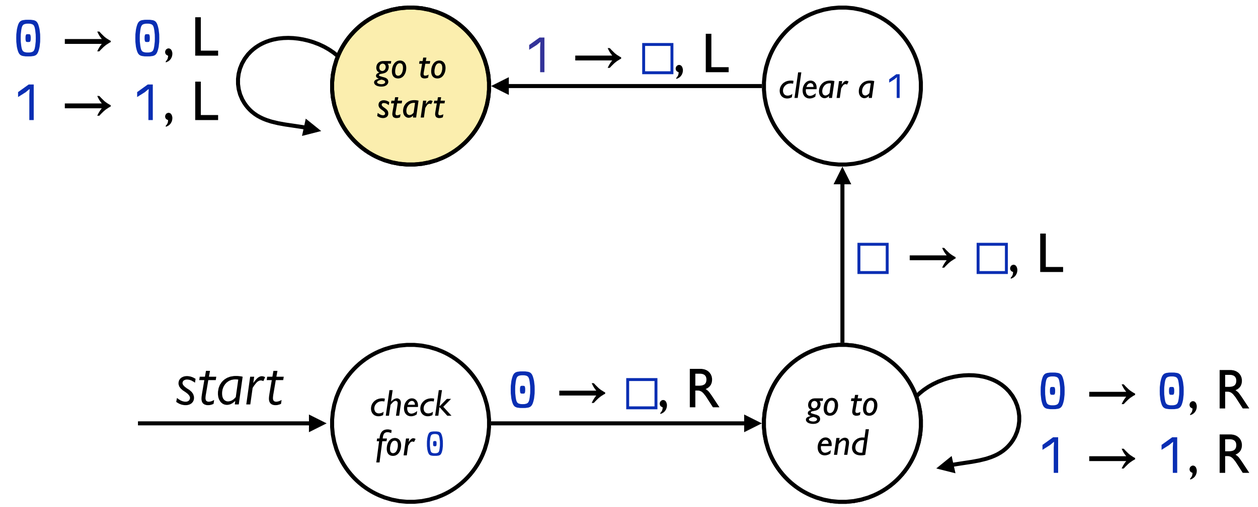

0 0 1 1

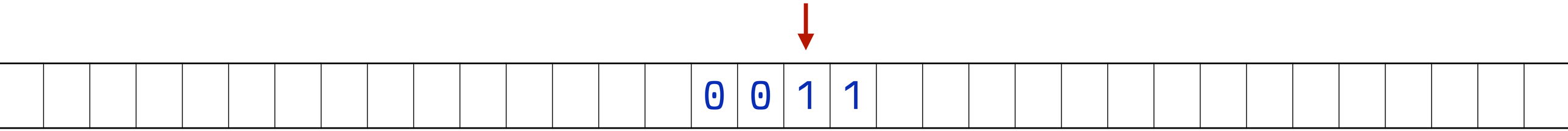

go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

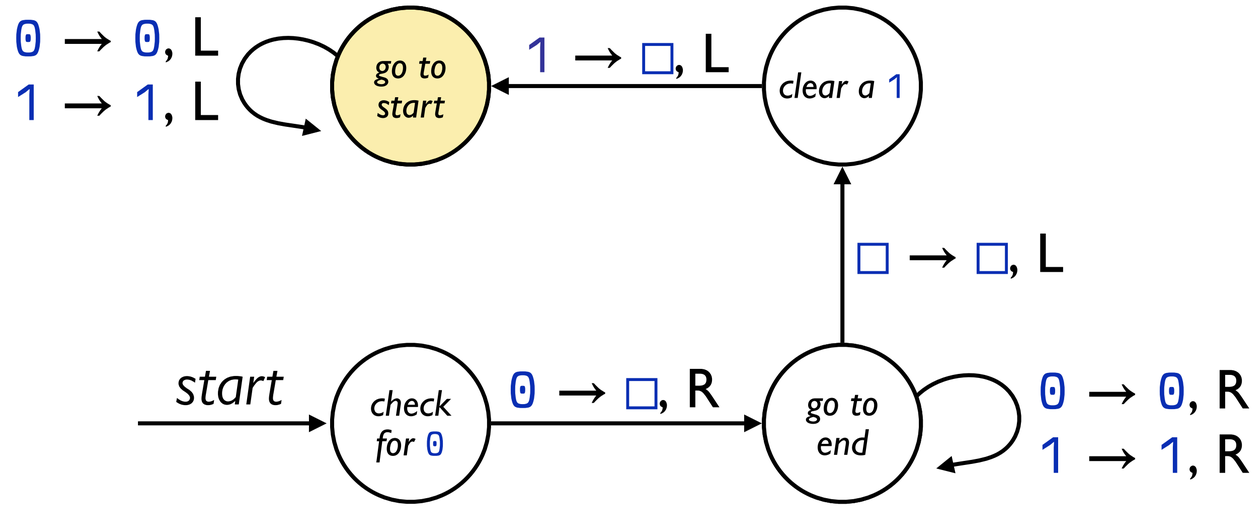

0 0 1 1

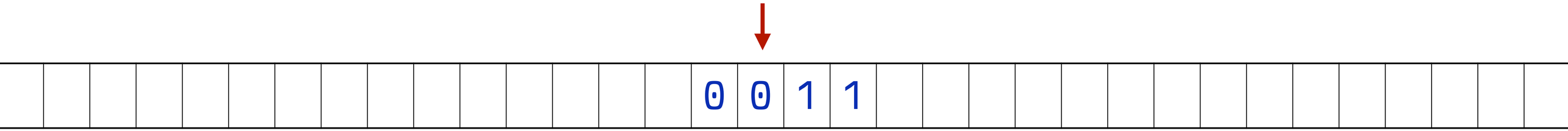

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

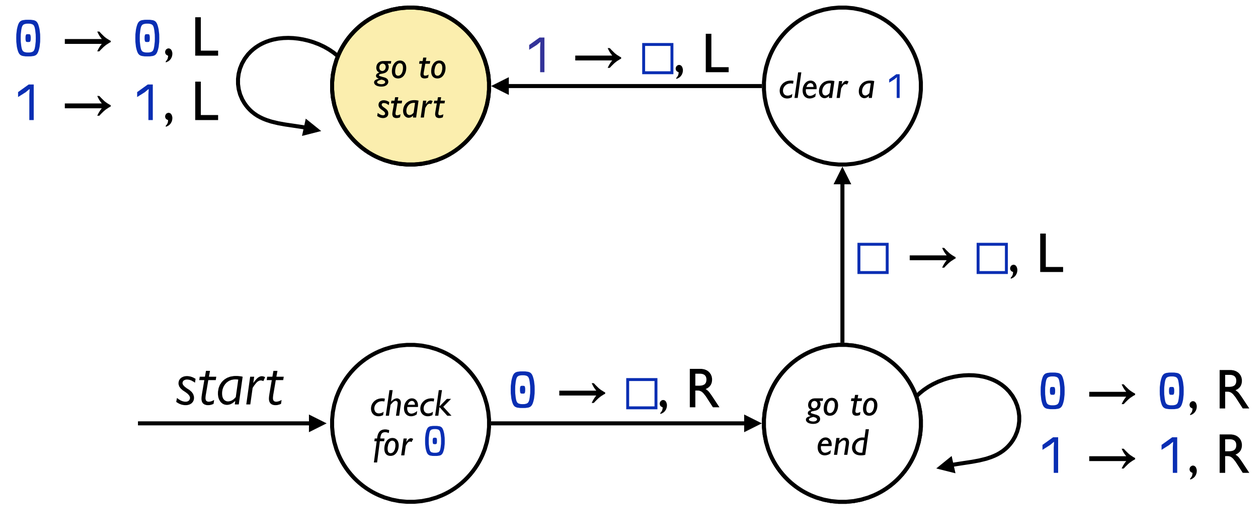

0 0 1 1

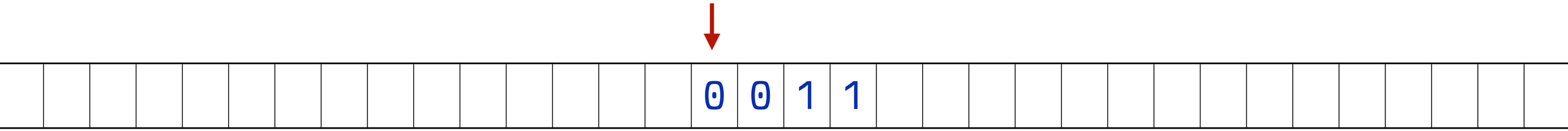

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

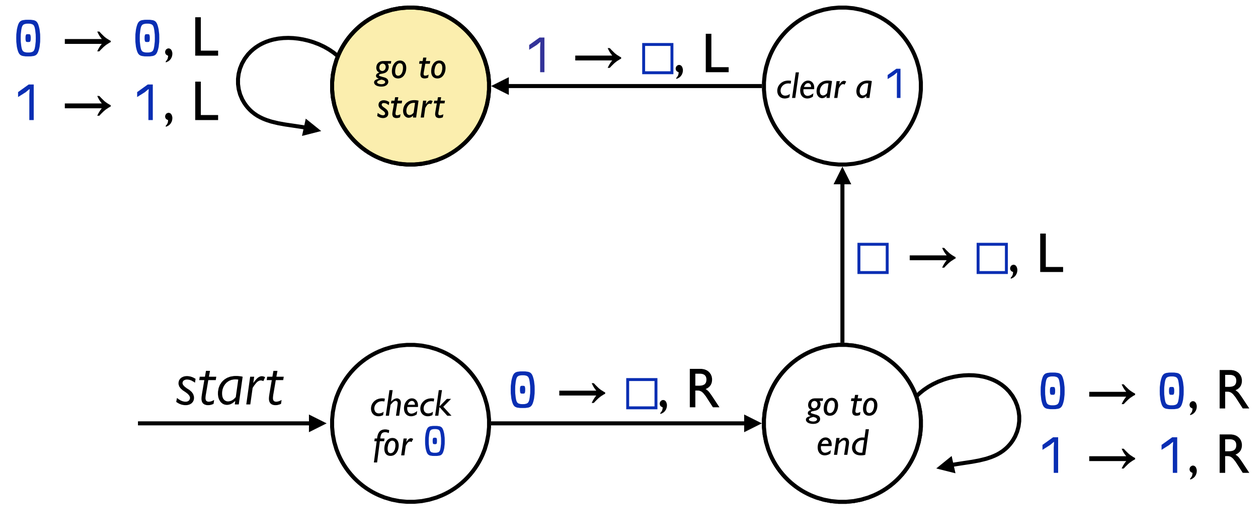

0 0 1 1

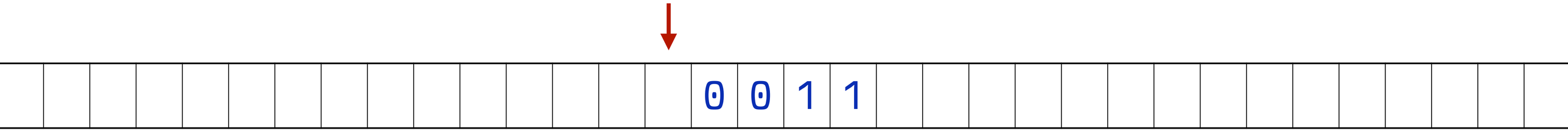

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

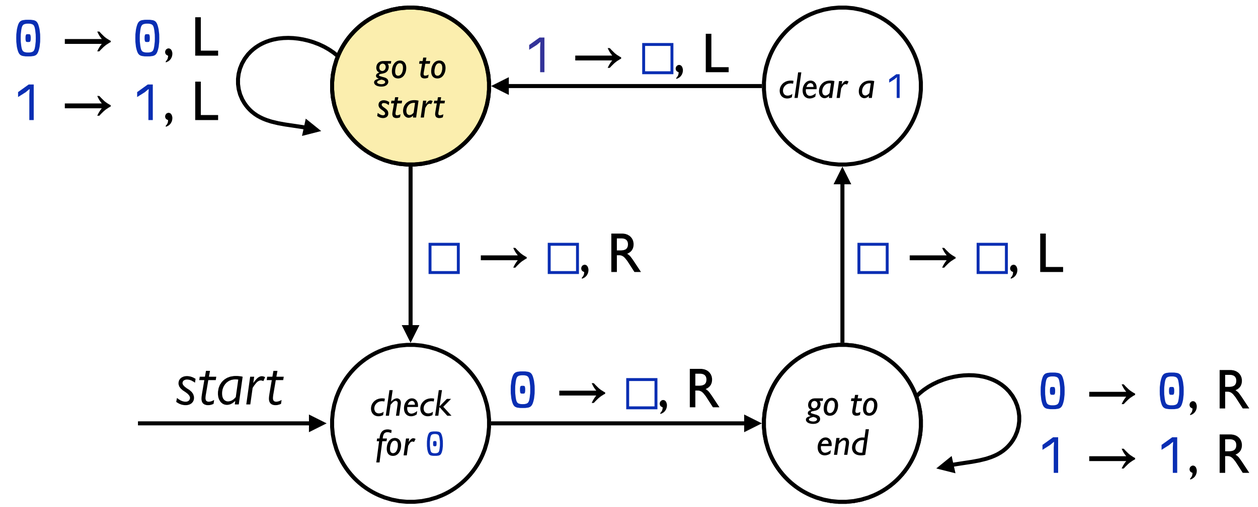

0 0 1 1

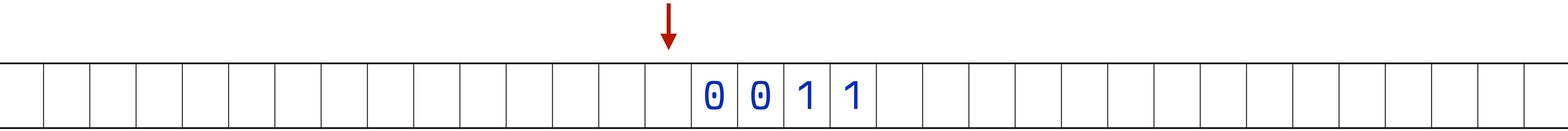

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

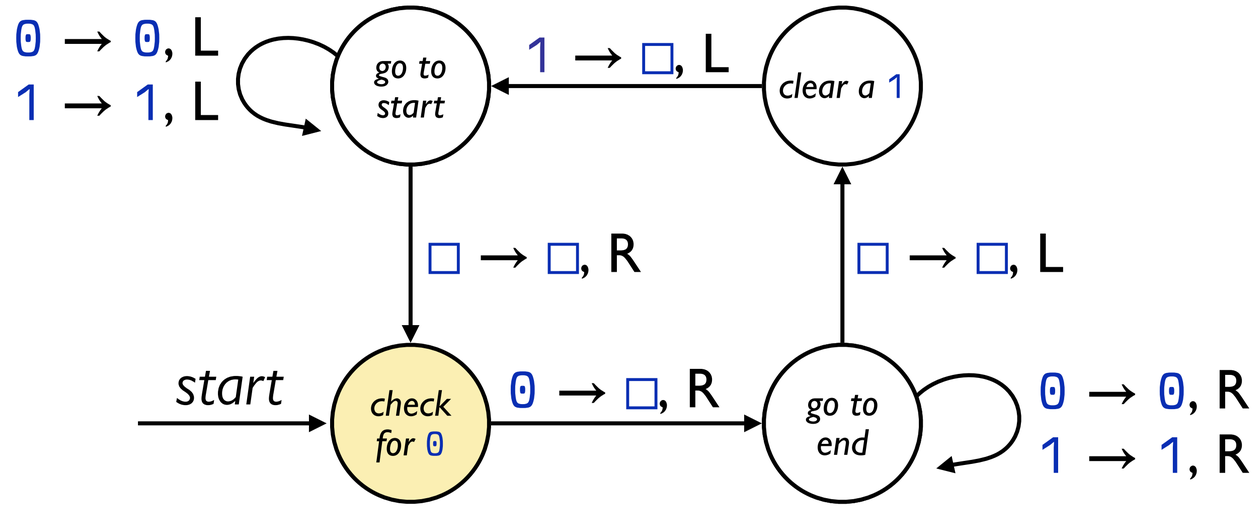

0 0 1 1

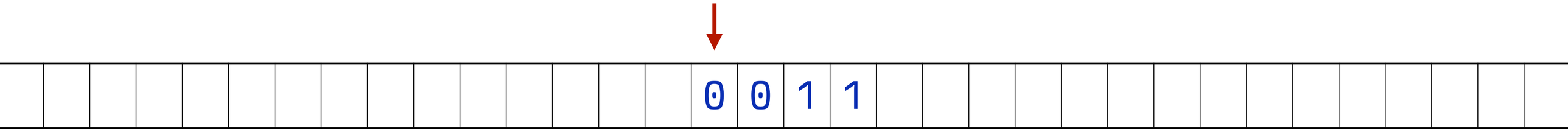

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

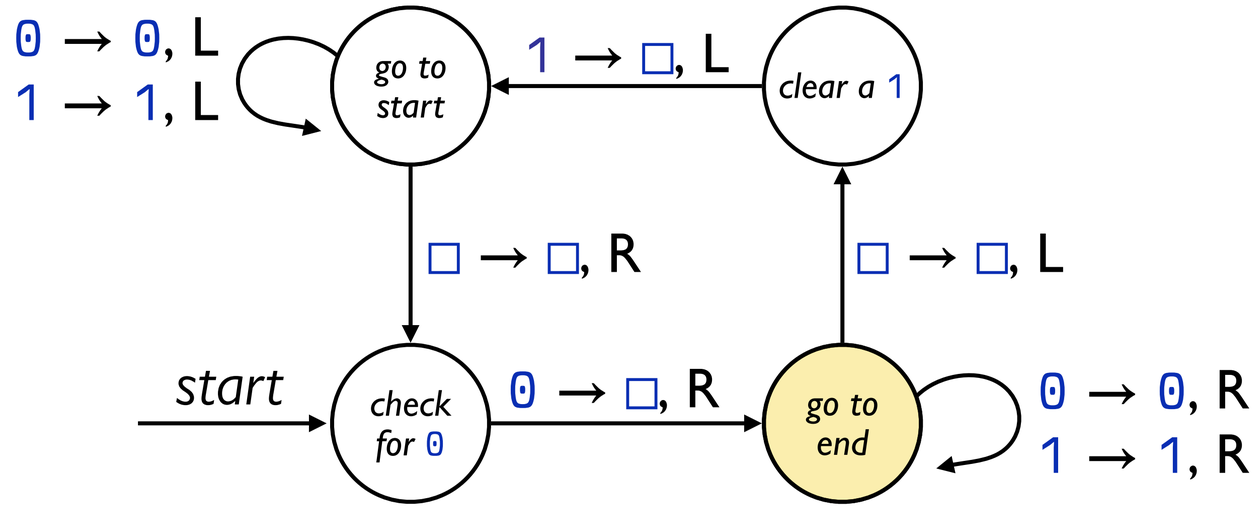

0 0 1 1

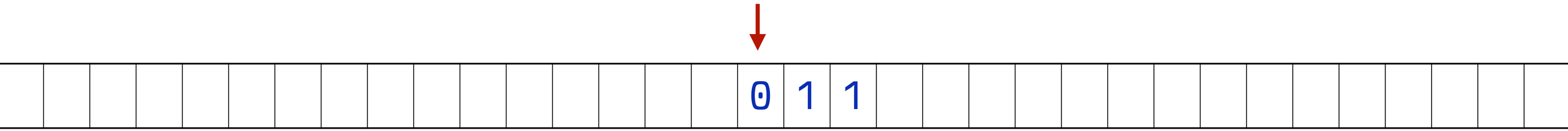

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

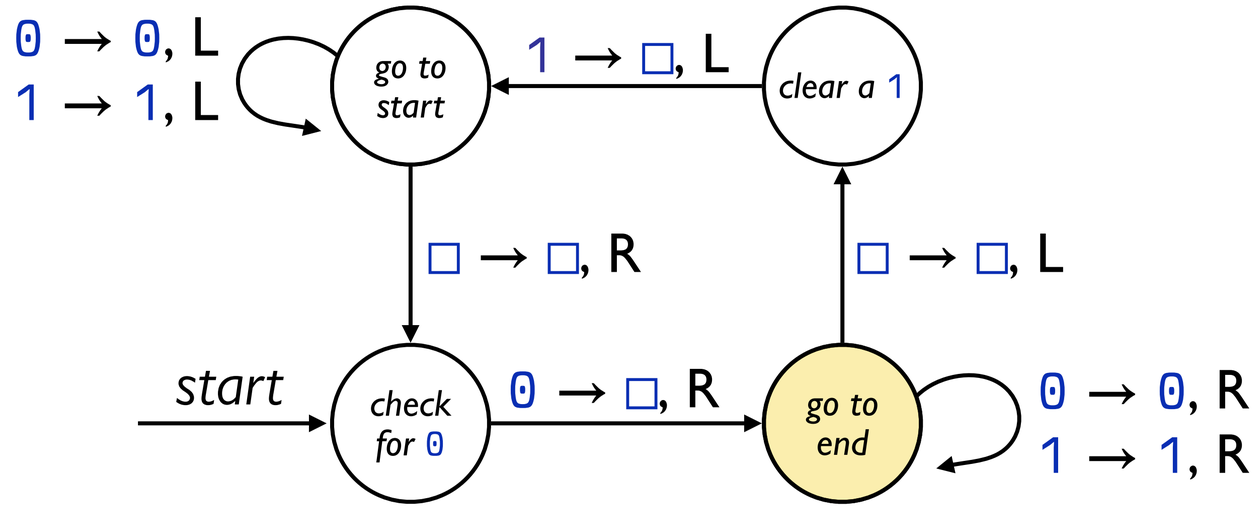

0 0 1 1

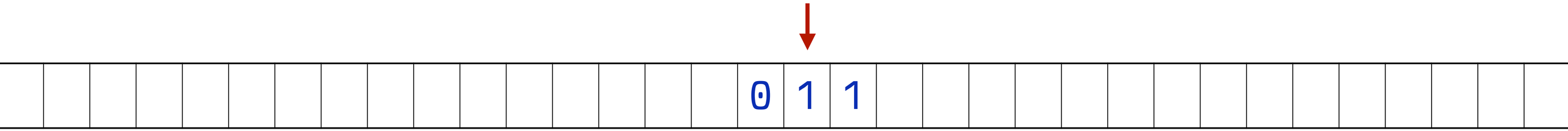

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

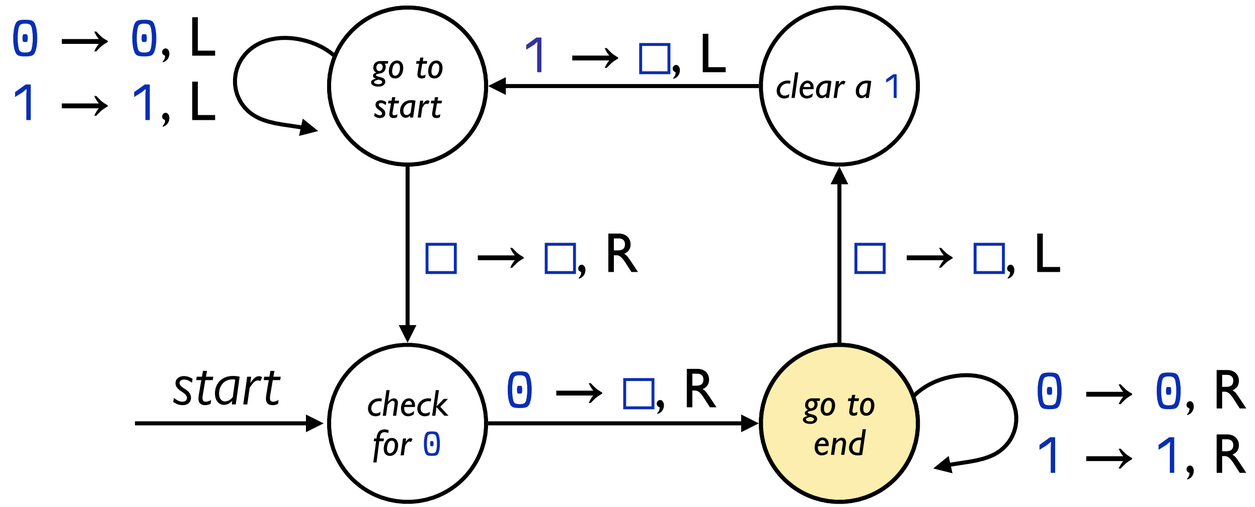

0 0 1 1

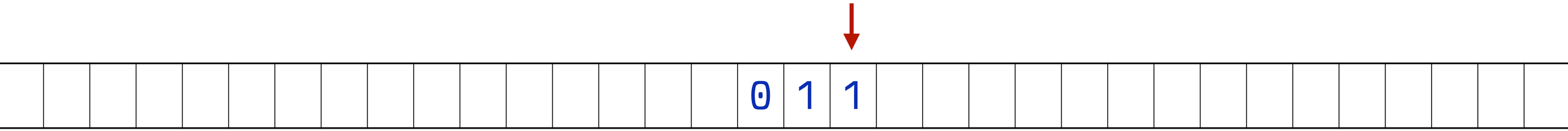

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

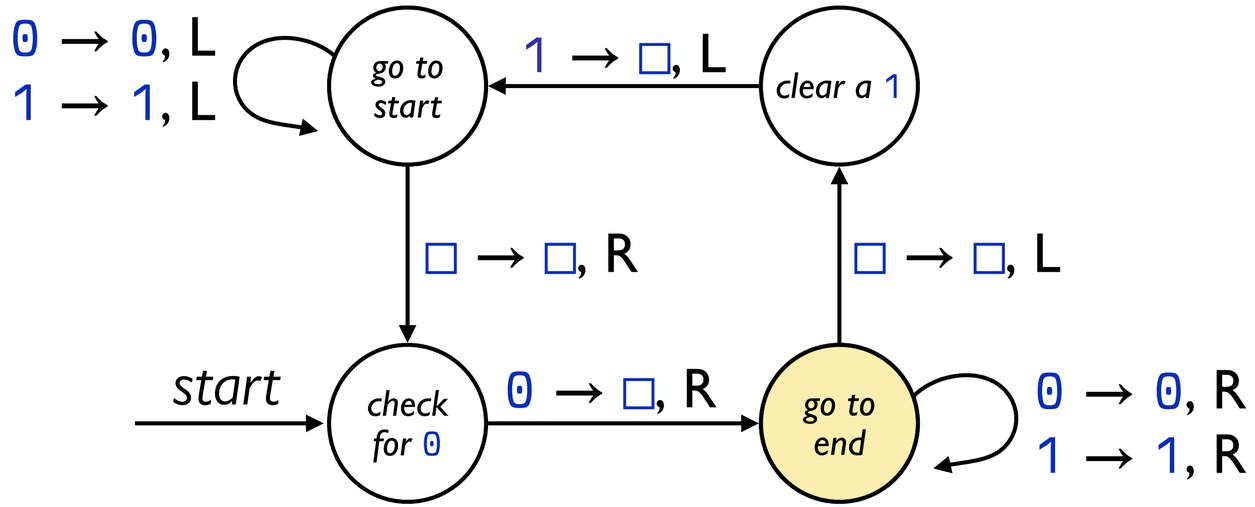

0 1 1

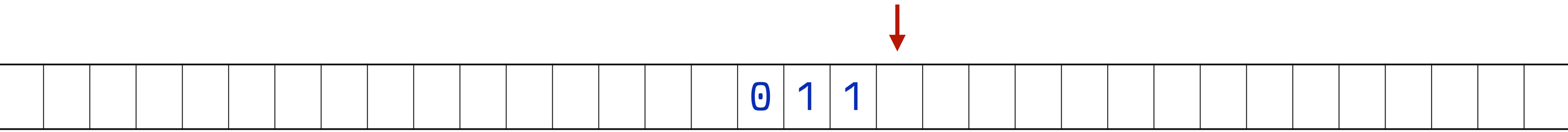

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

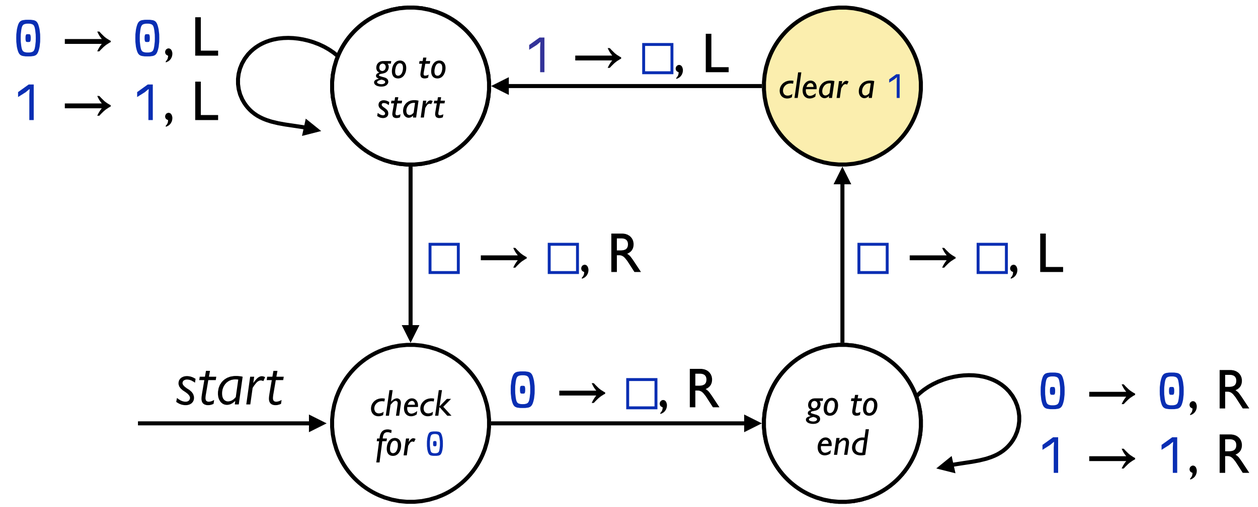

0 1 1

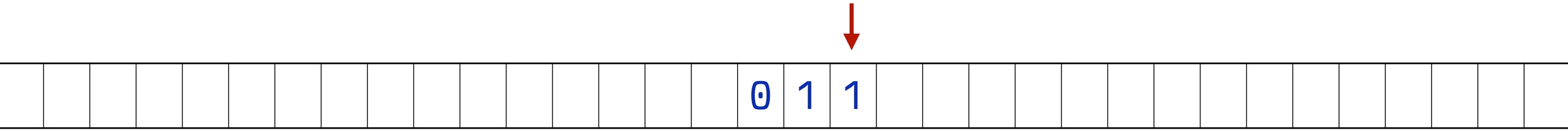

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

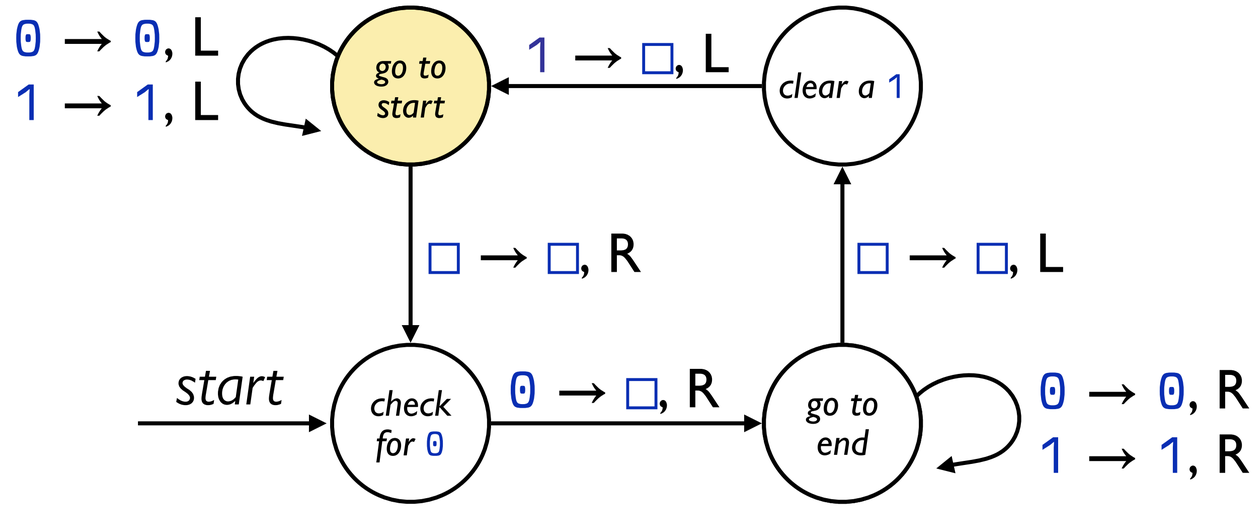

0 1 1

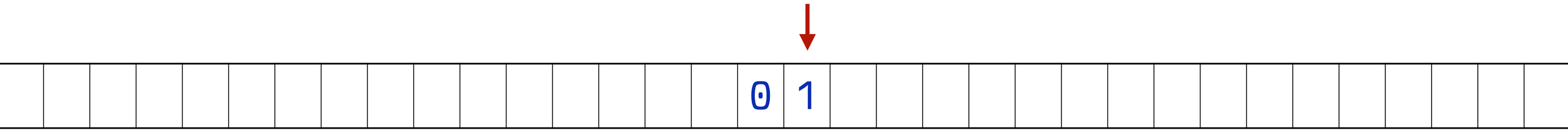

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

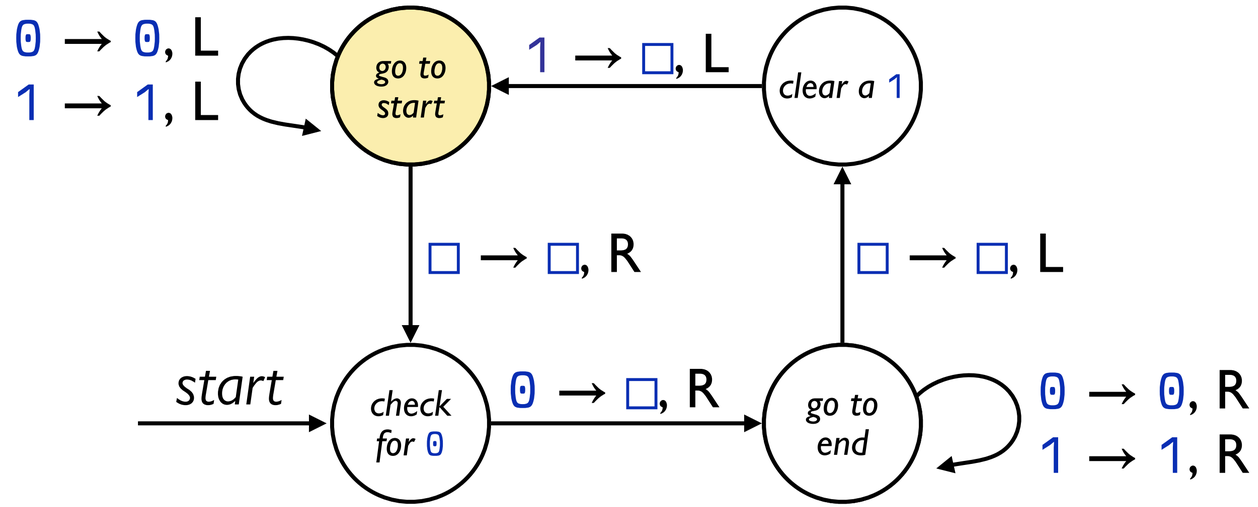

0 1 1

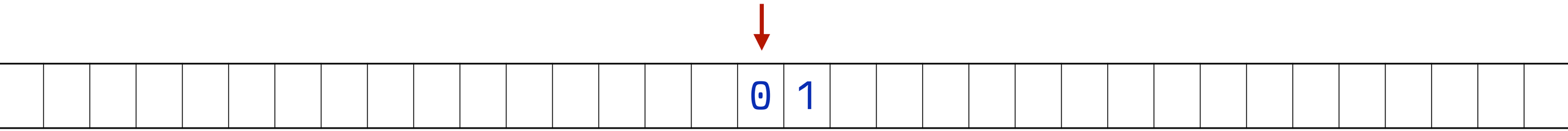

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

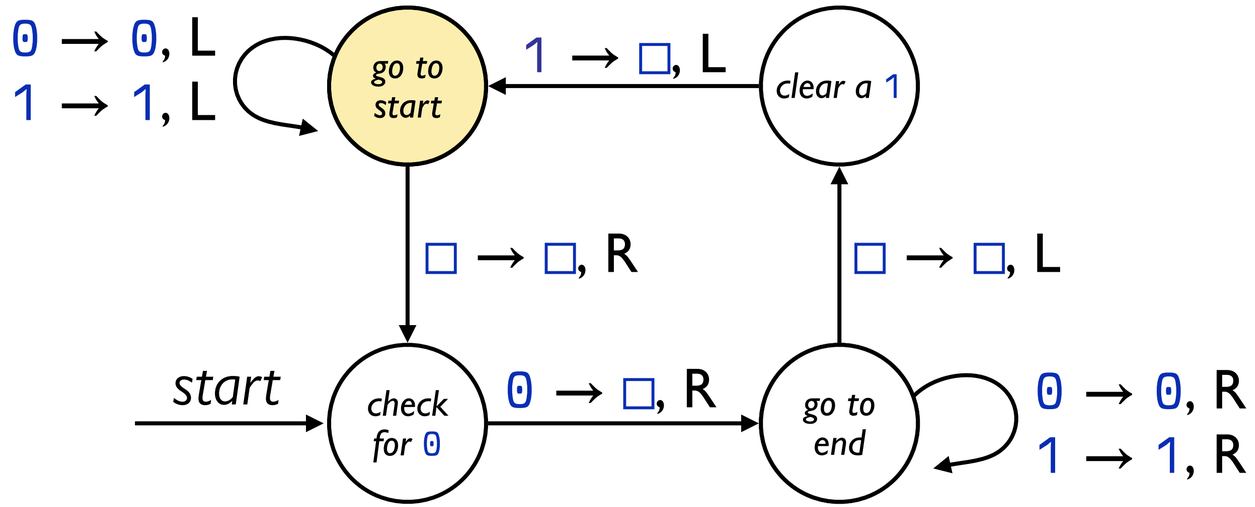

0 1 1

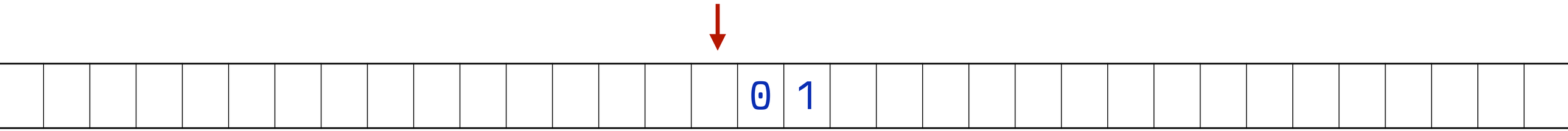

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

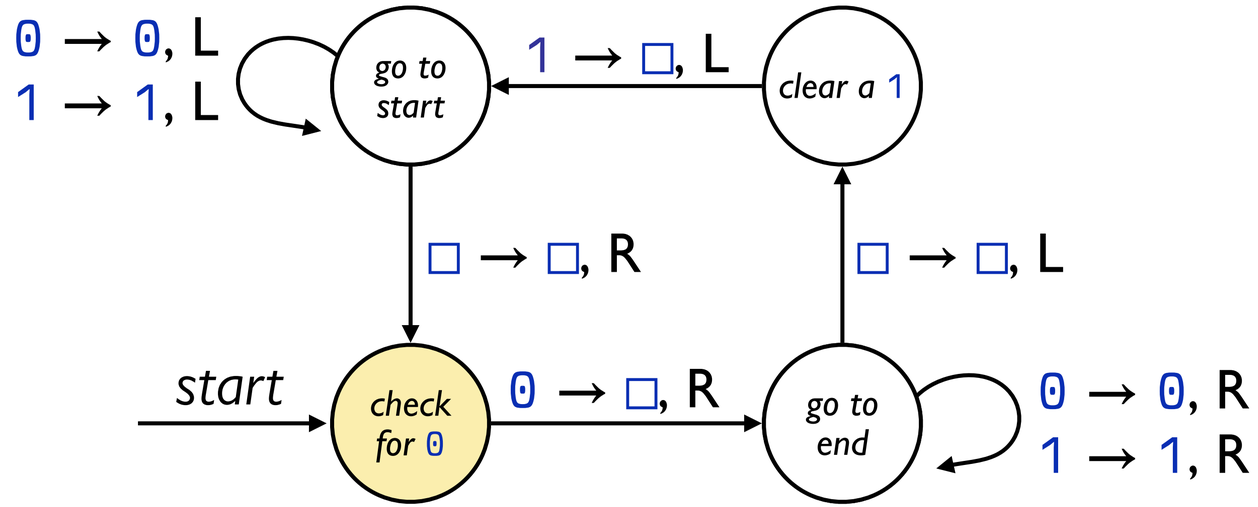

0
1

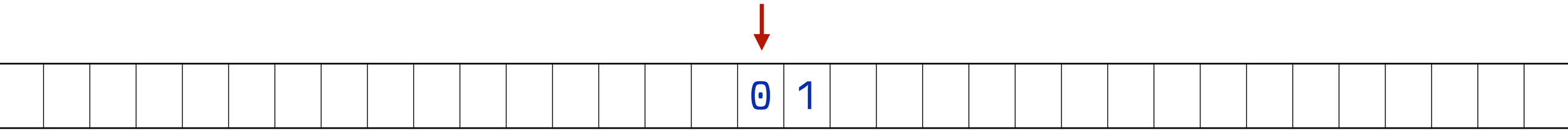

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

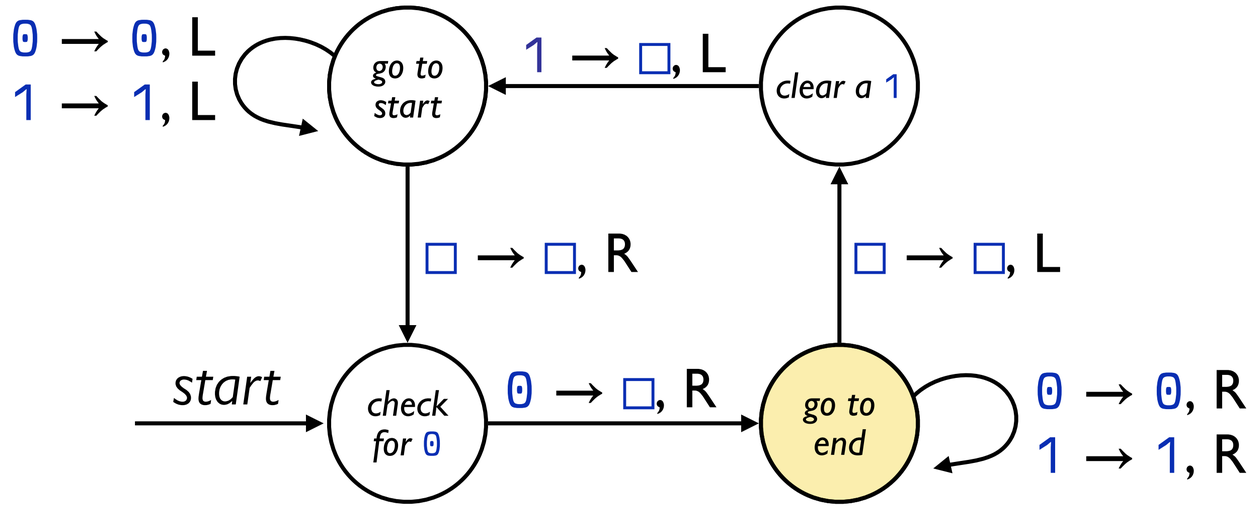

0
1

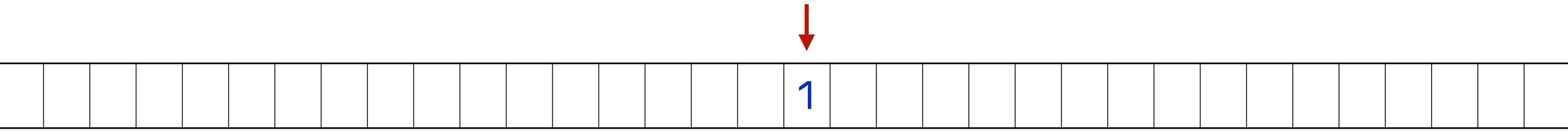

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

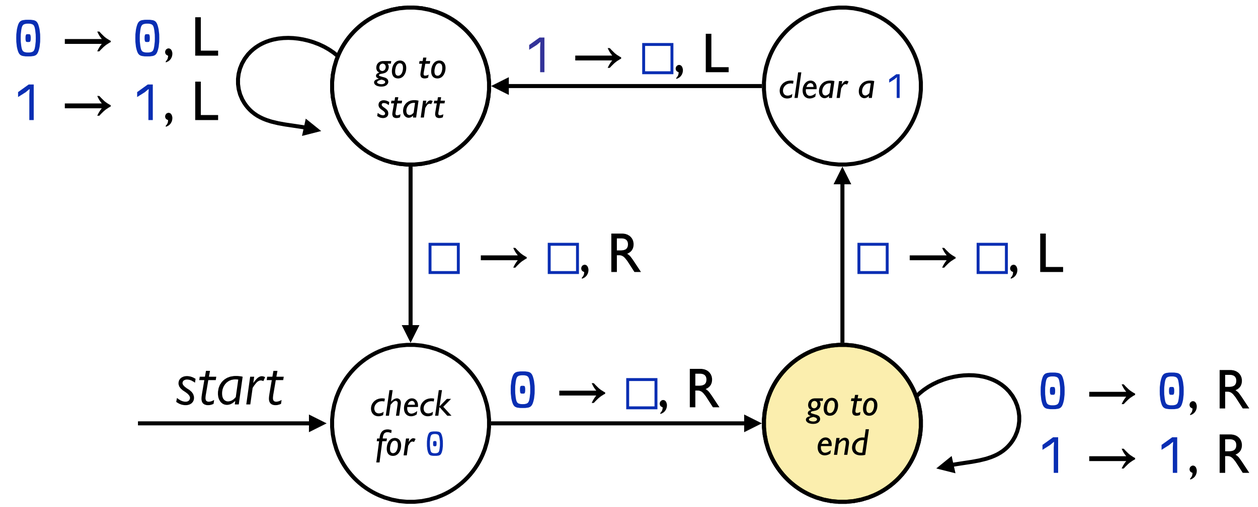

0
1

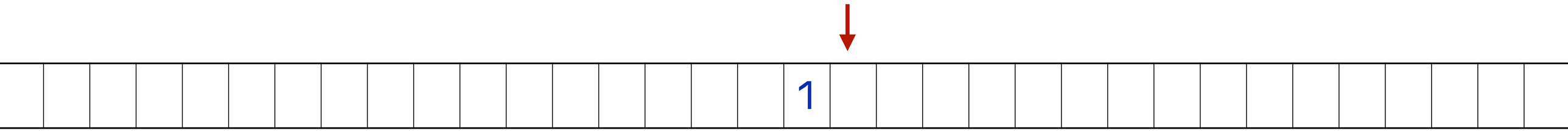

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

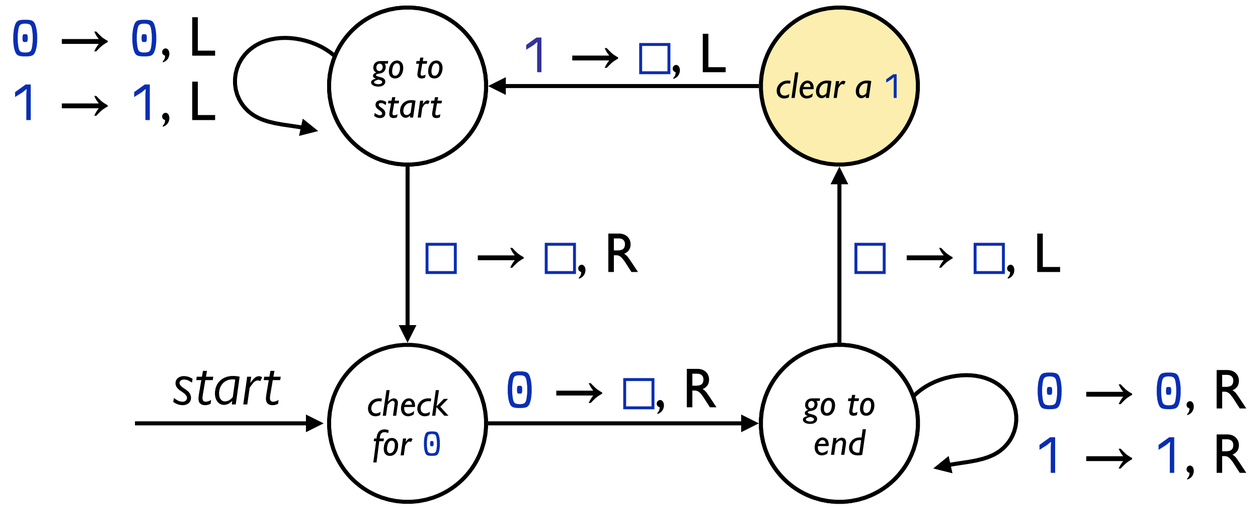

0
1

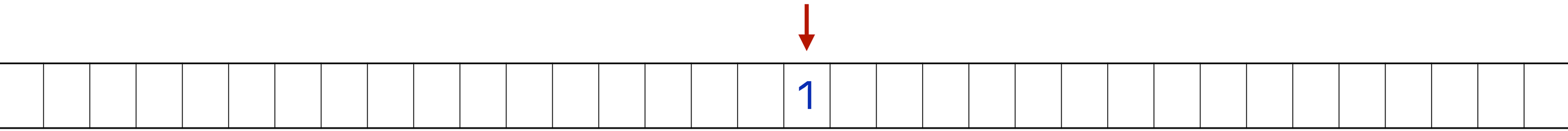

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

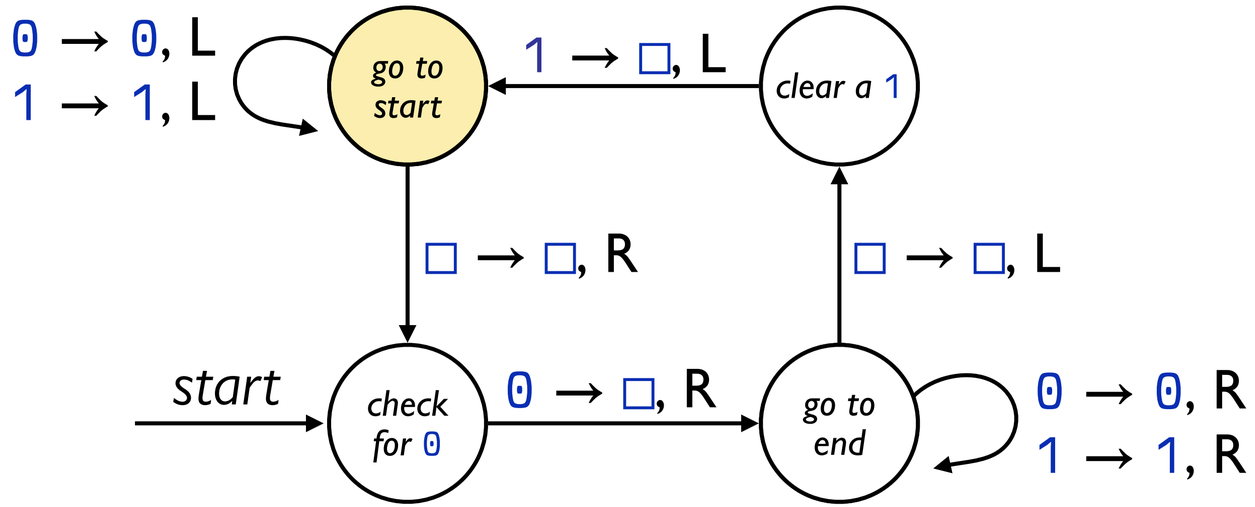

1

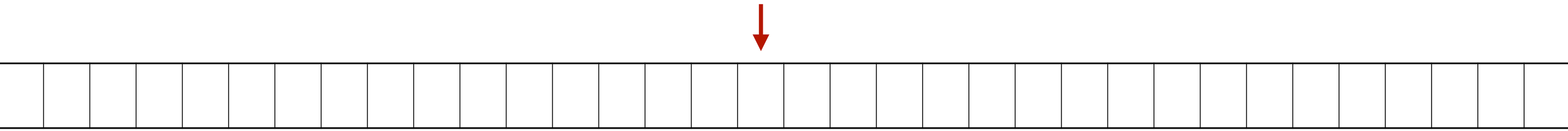

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

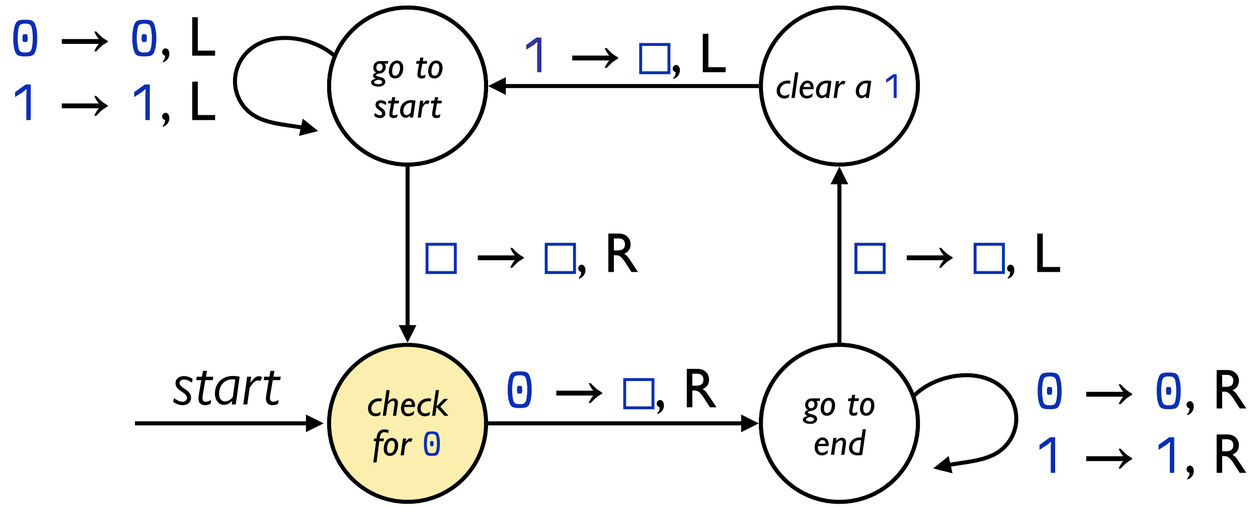

1

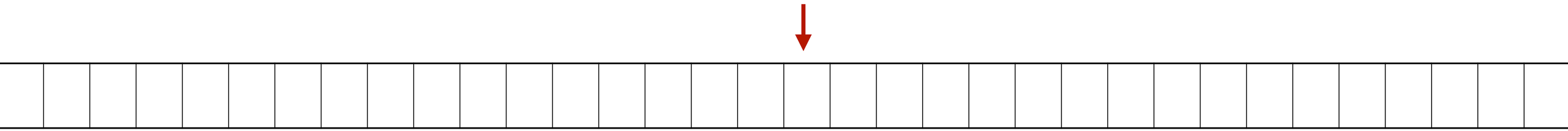

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

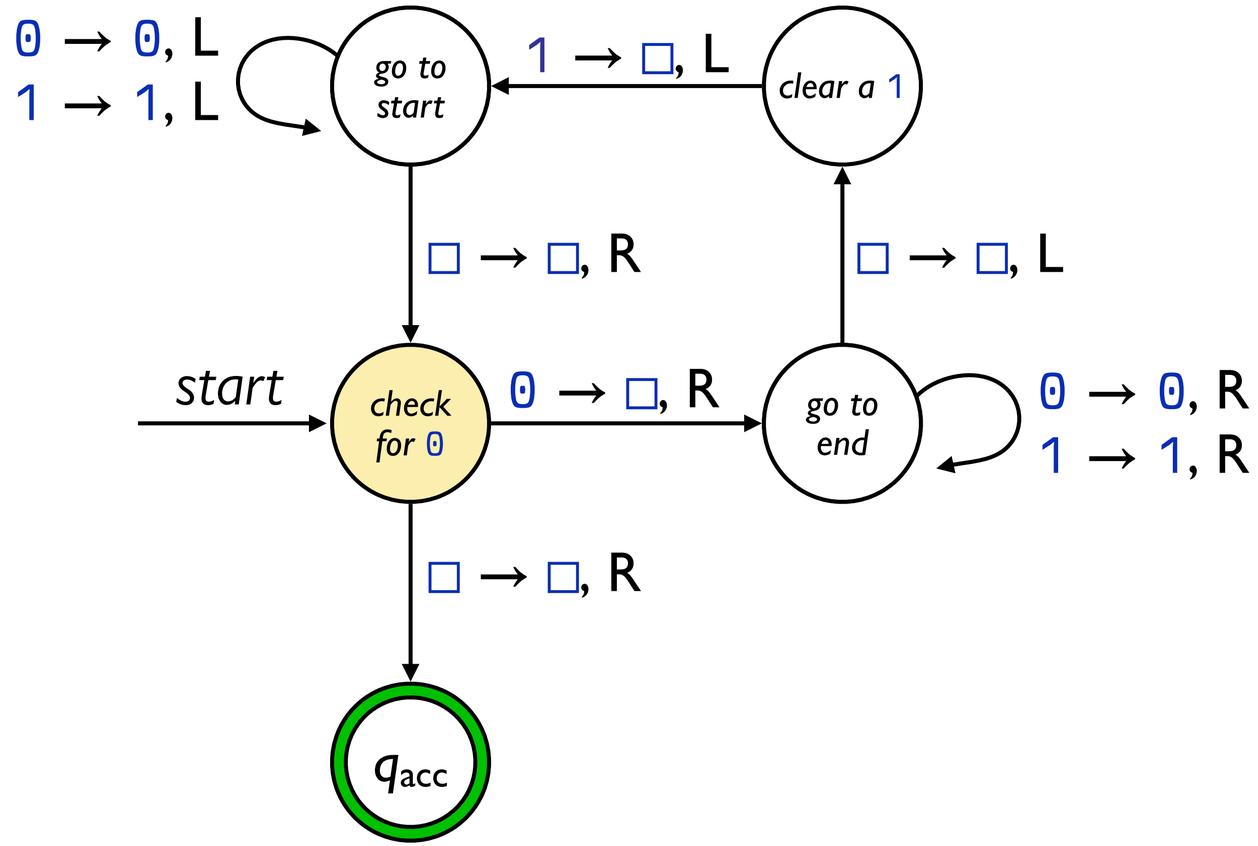

1

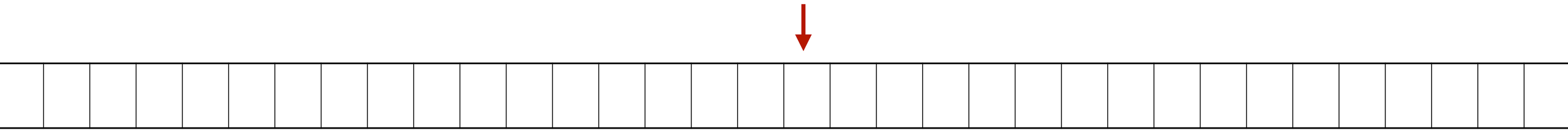

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

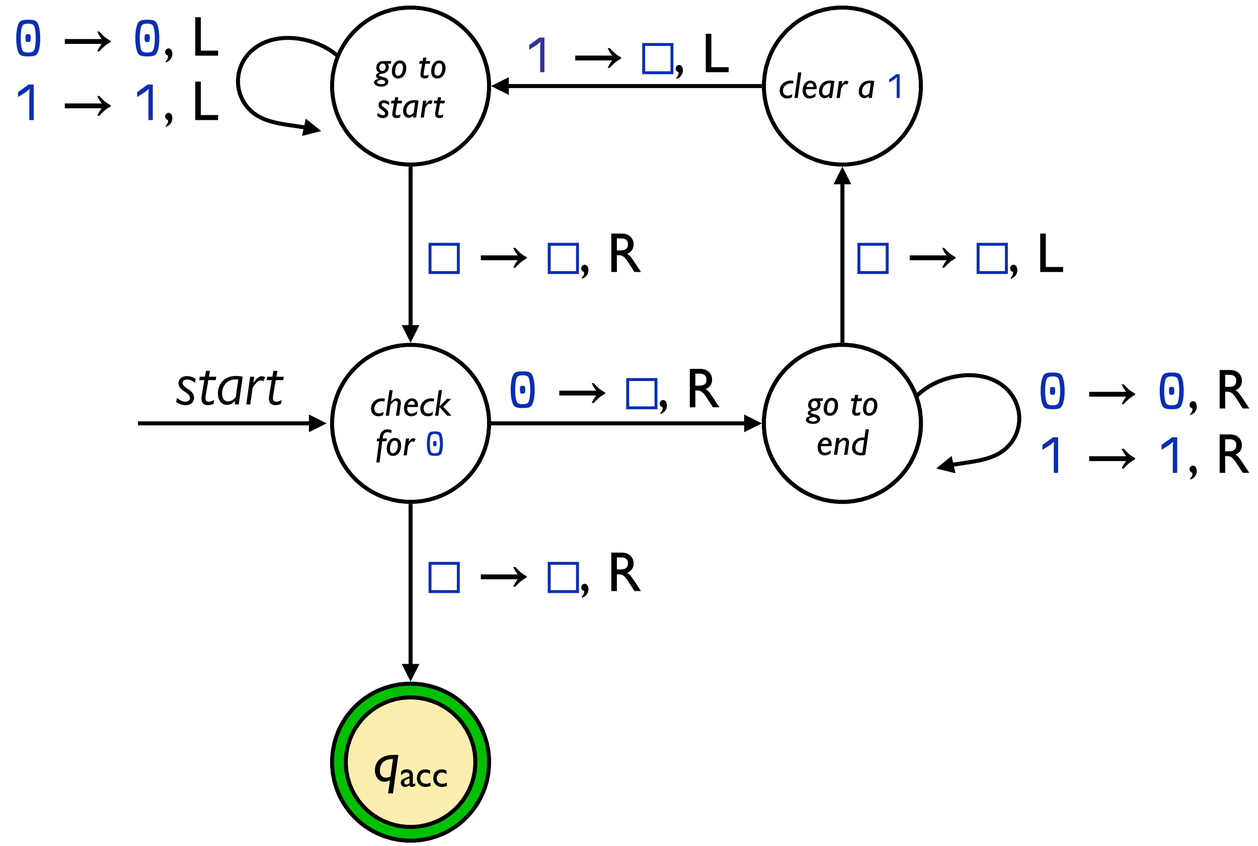


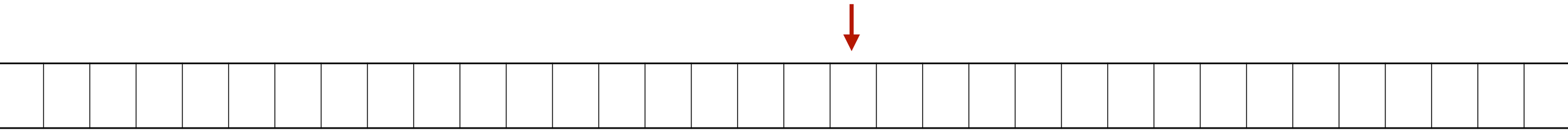

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R

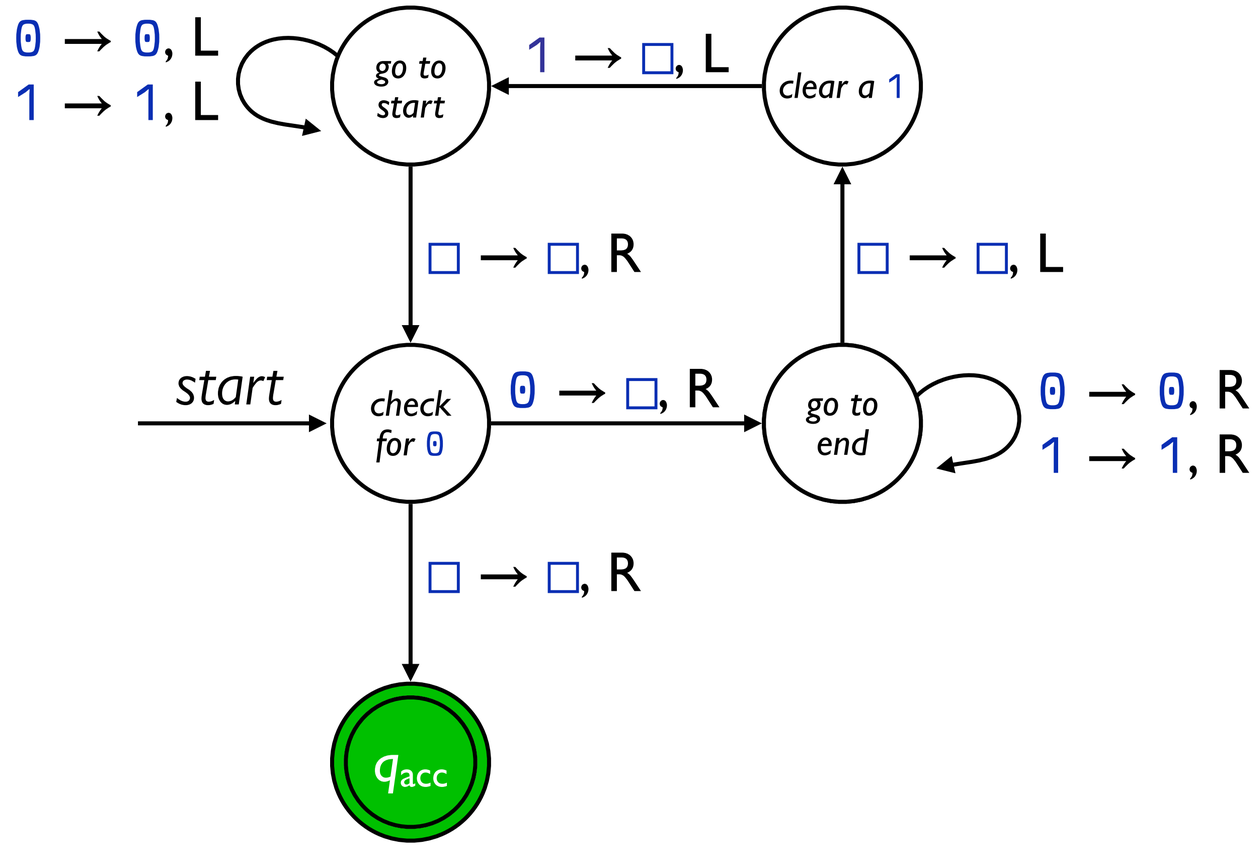


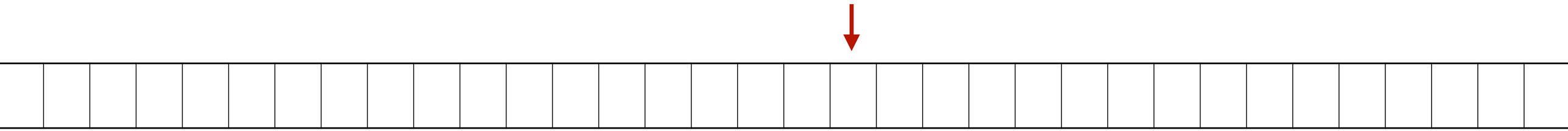

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}

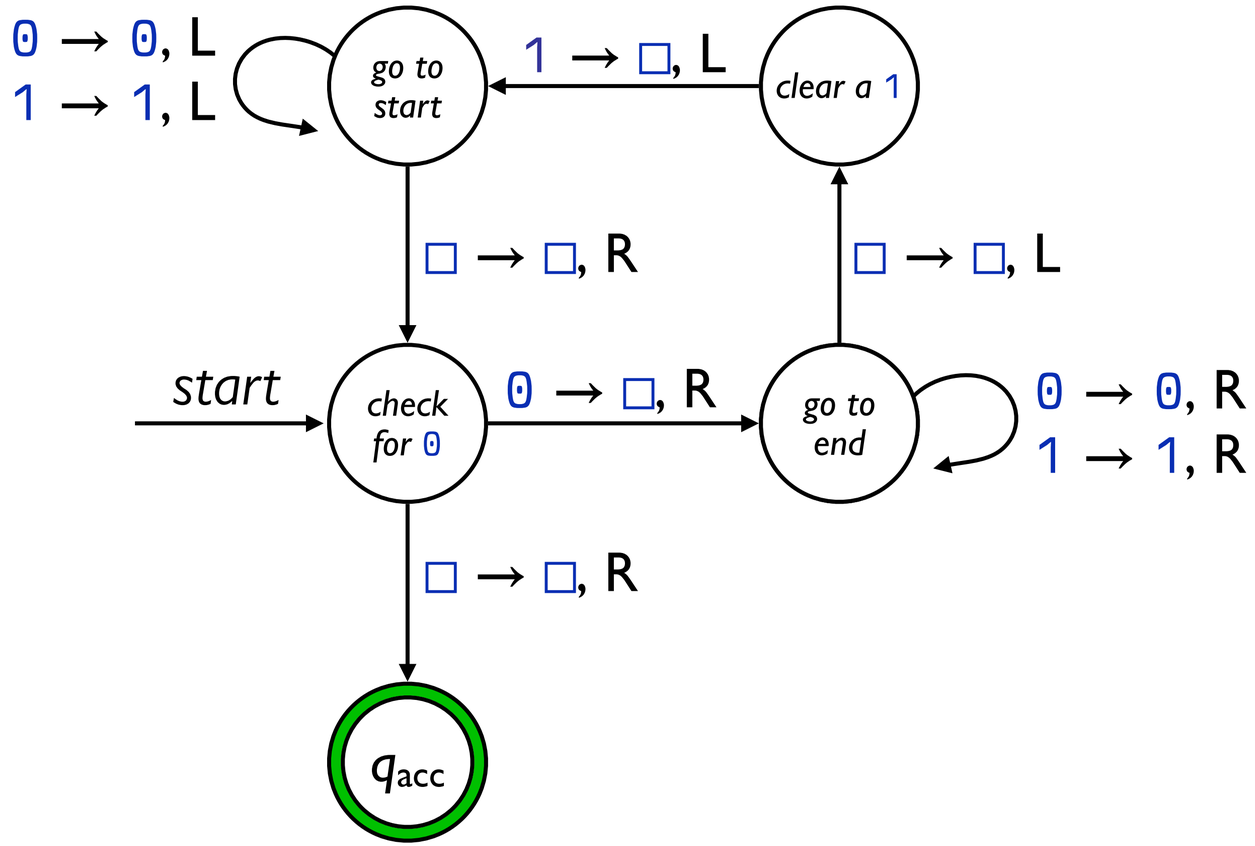


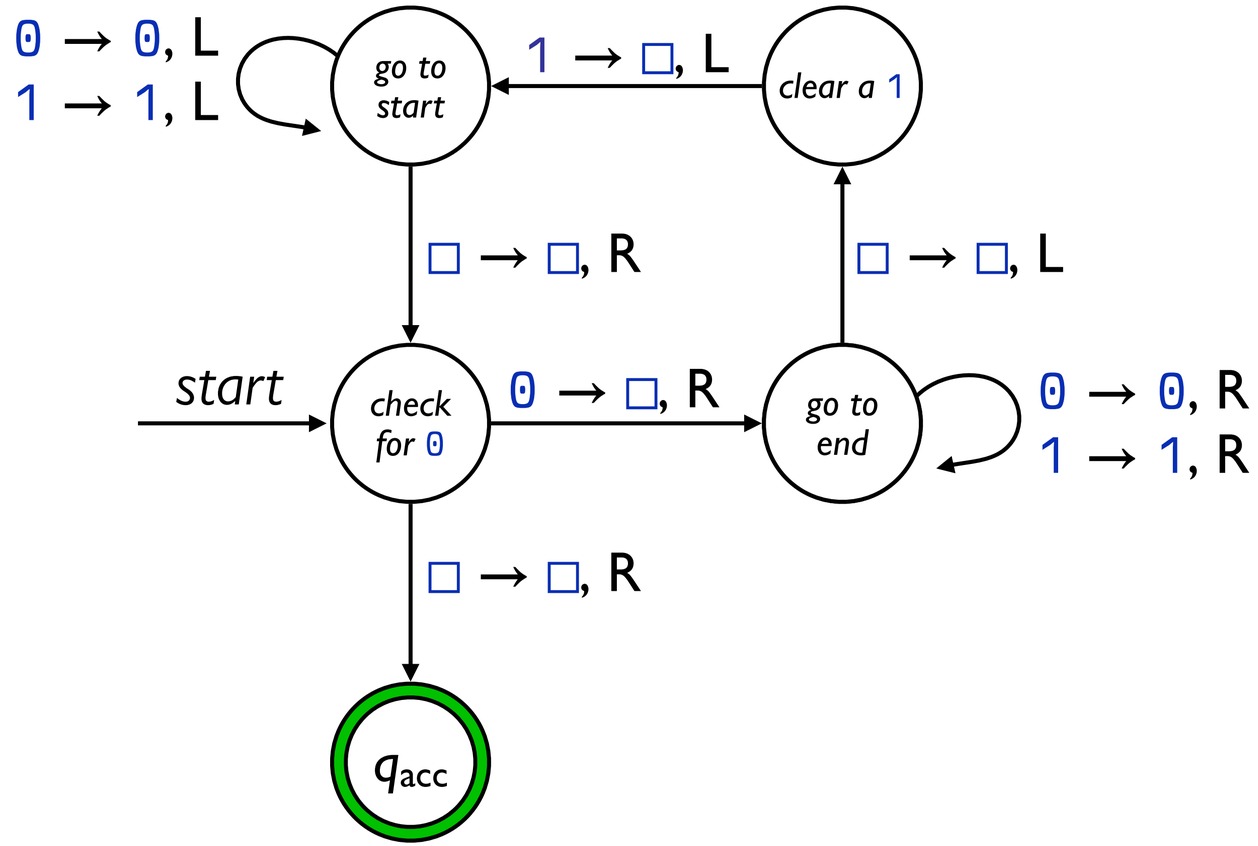

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

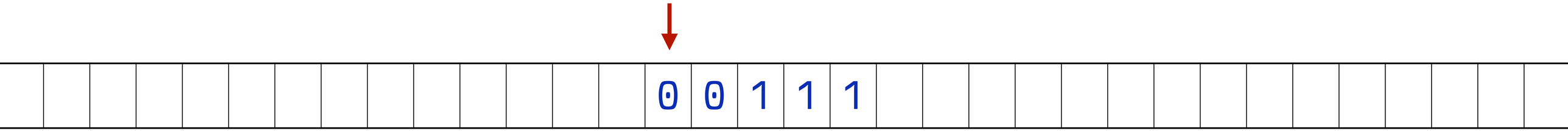


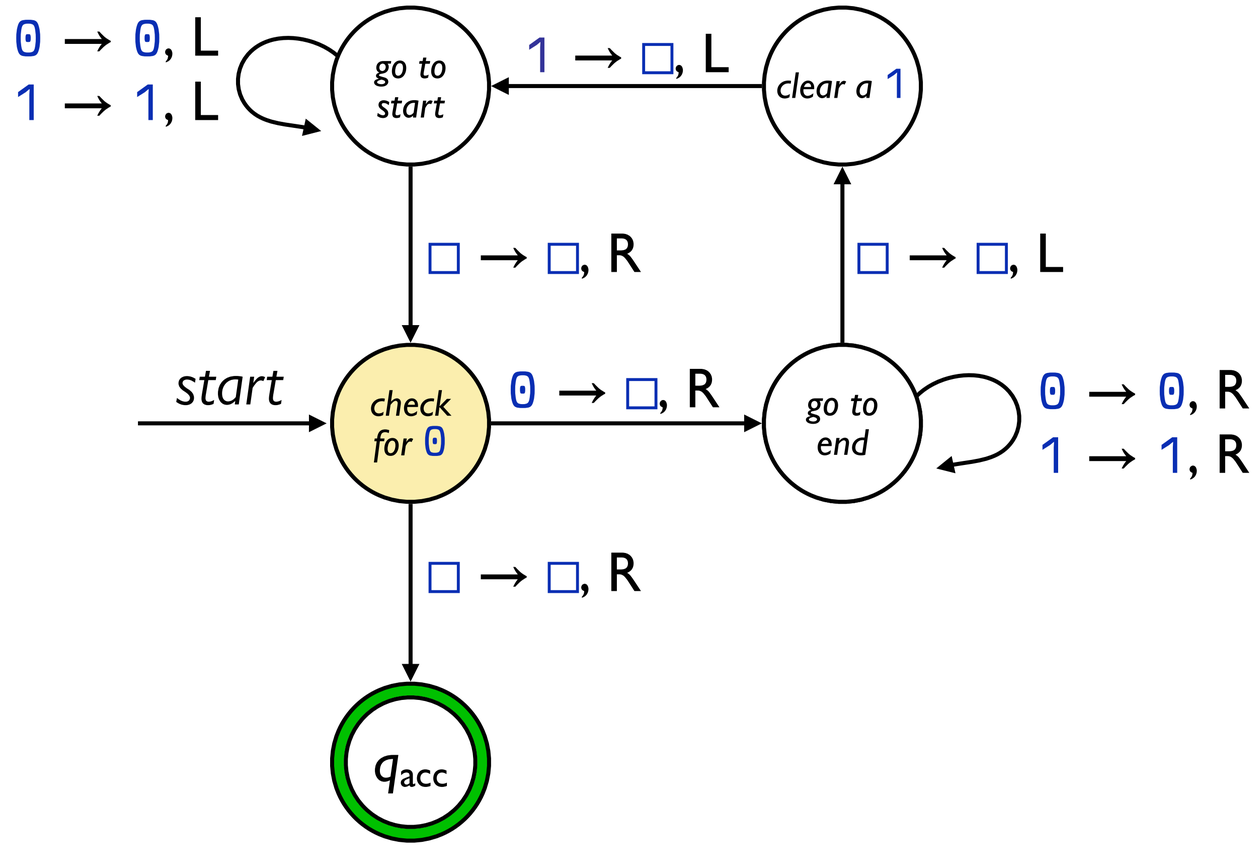

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

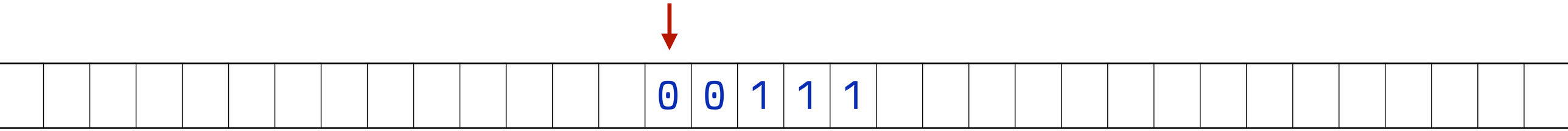


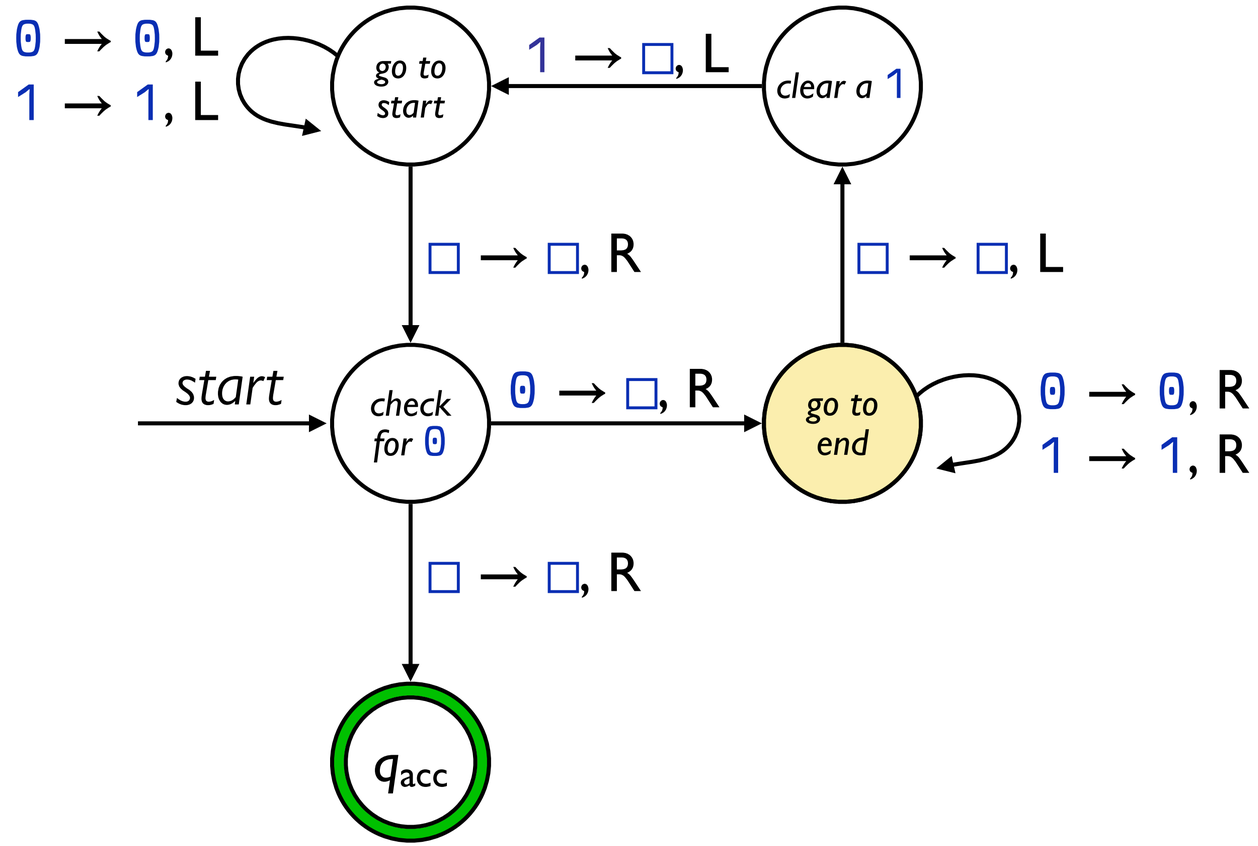

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

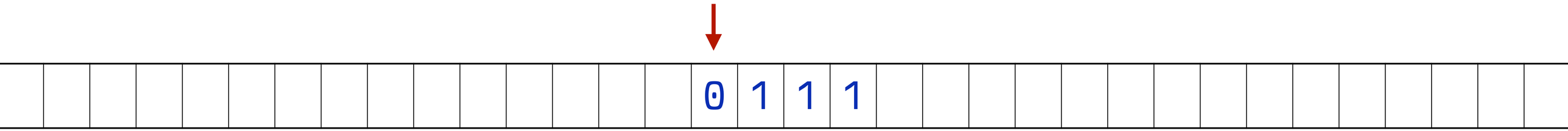

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

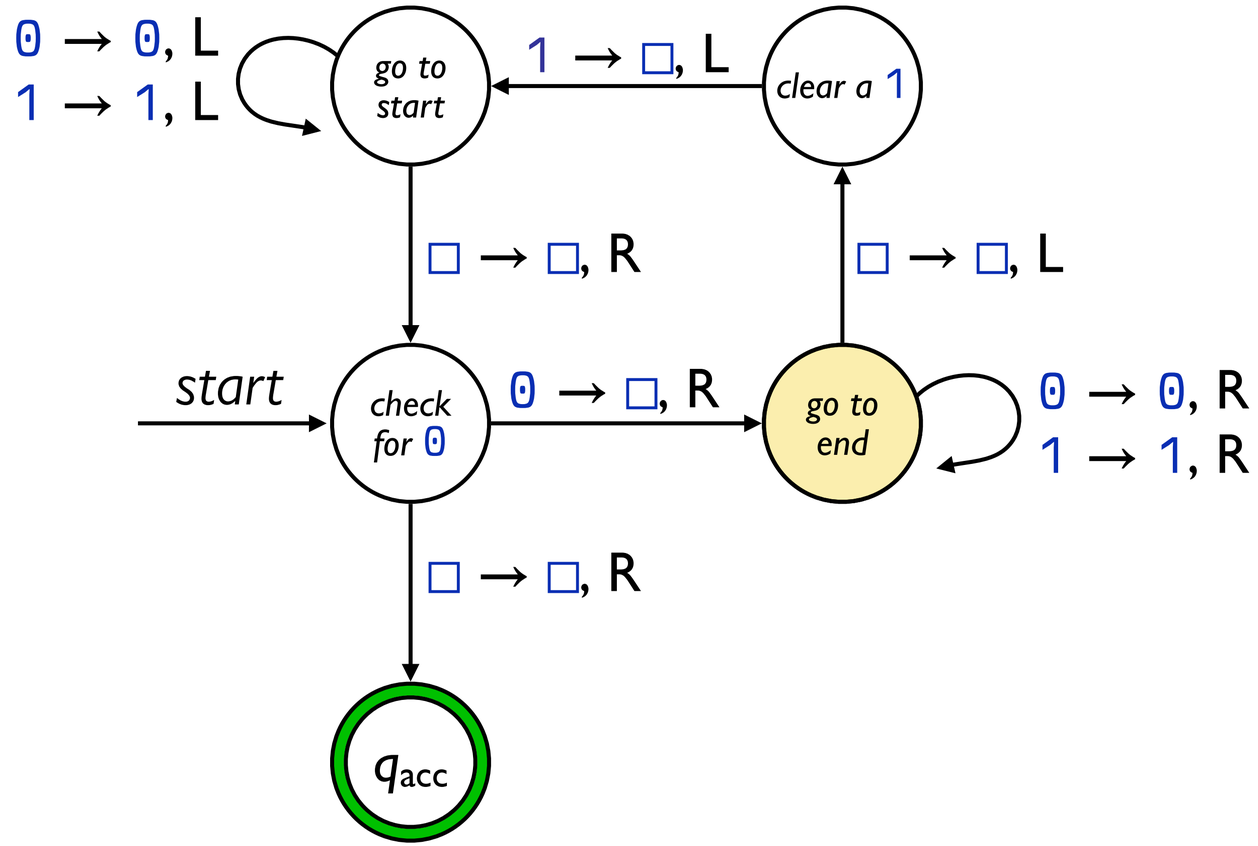

0 0 1 1 1

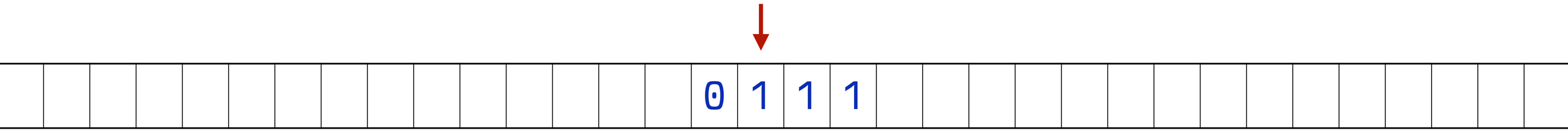

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

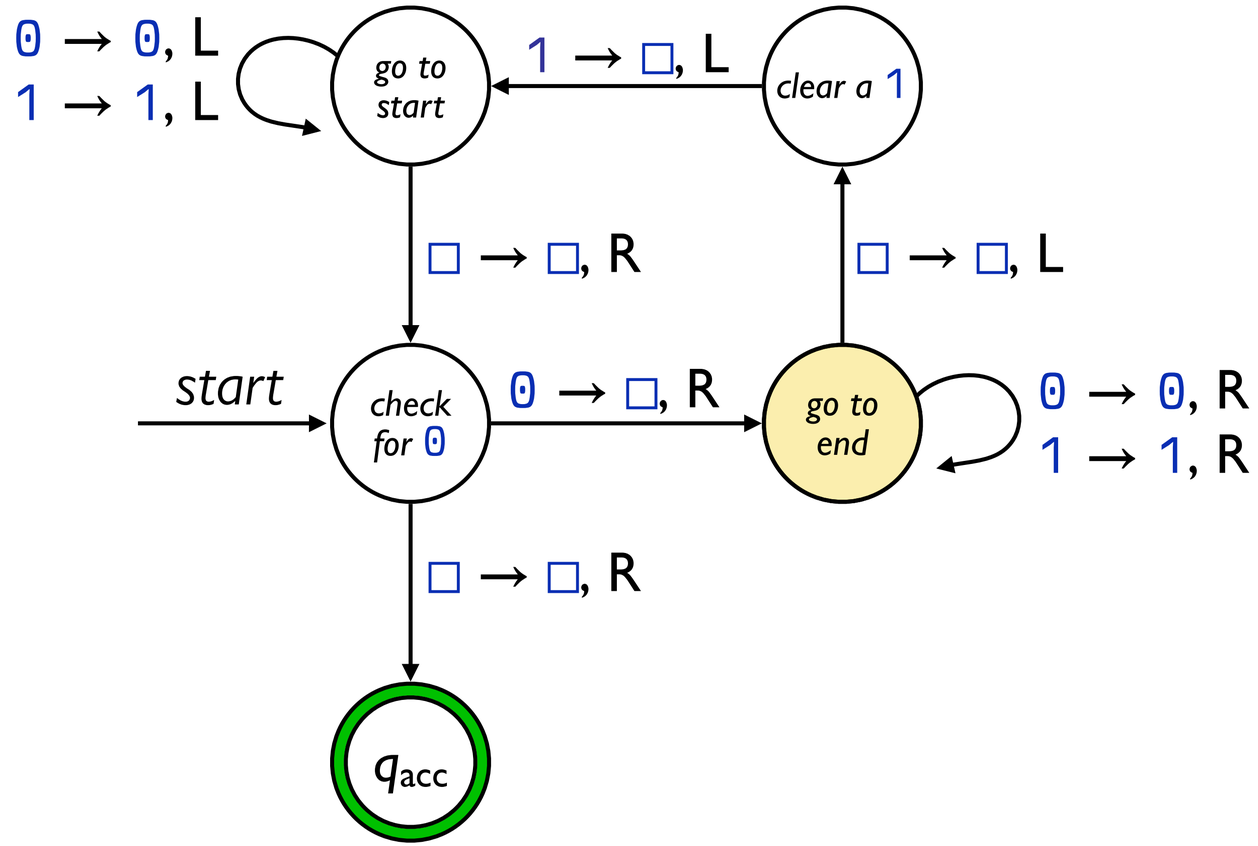

0 0 1 1 1

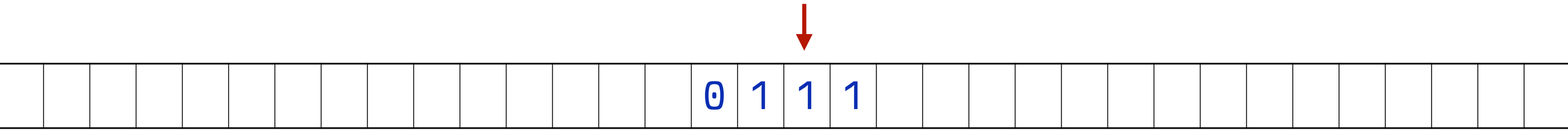

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}

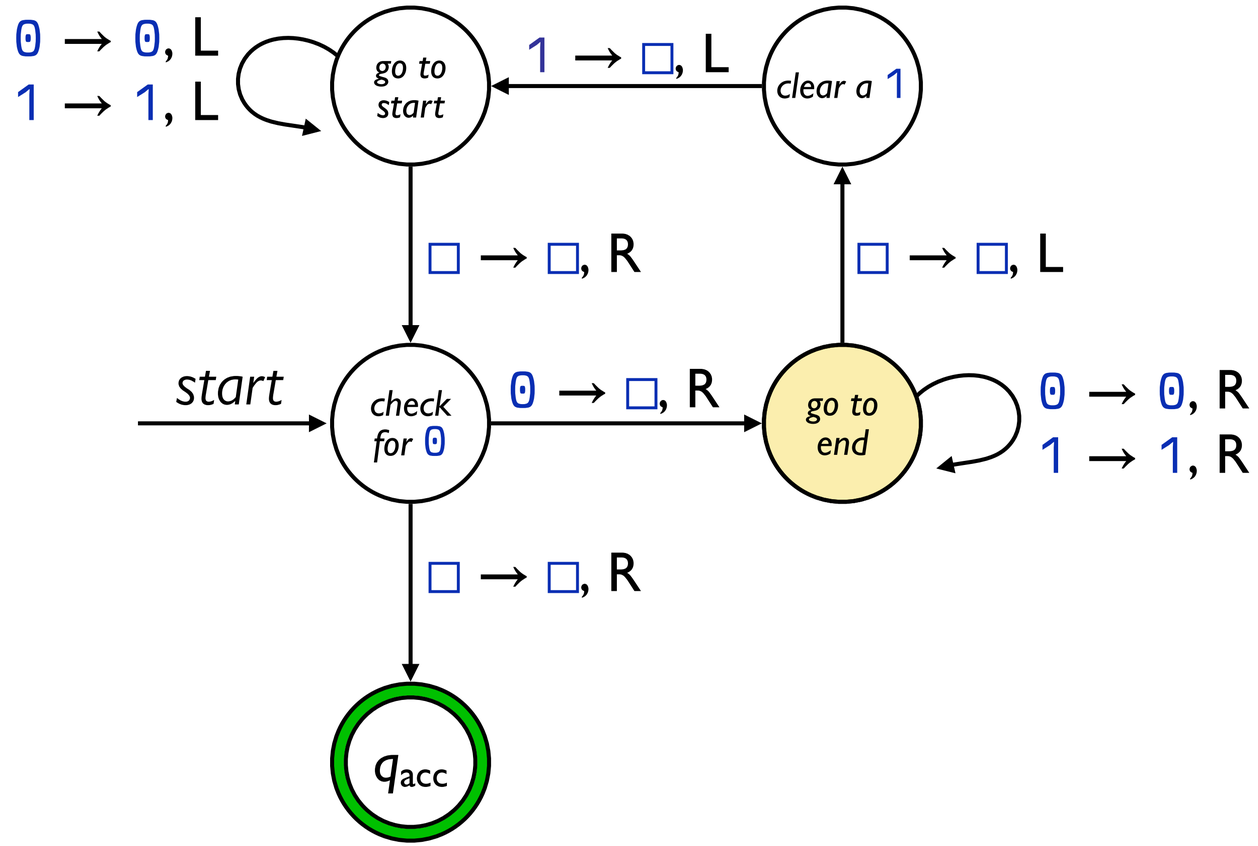

0 1 1 1

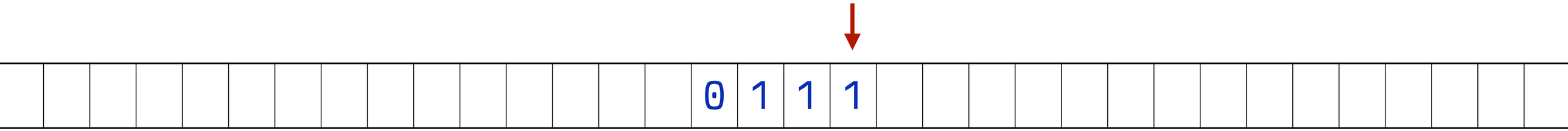

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

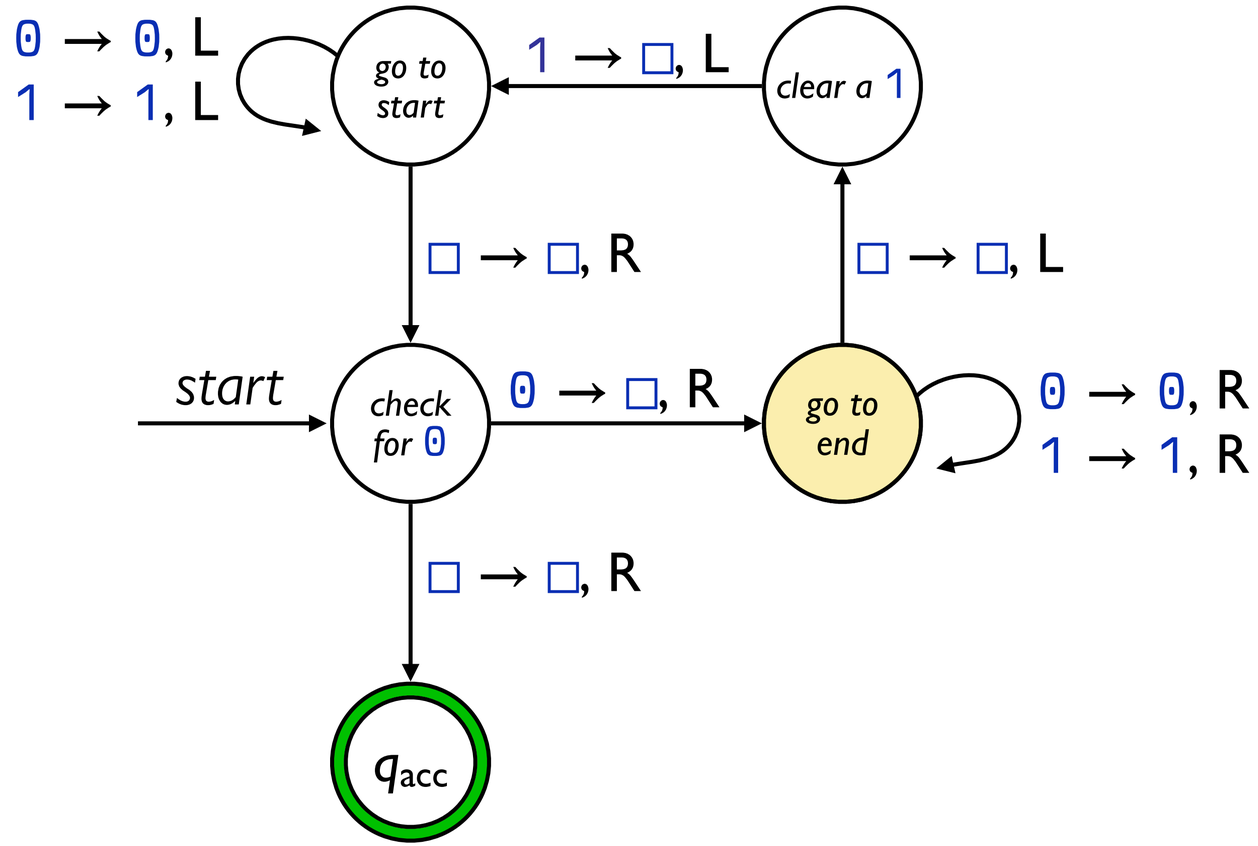

0 1 1 1

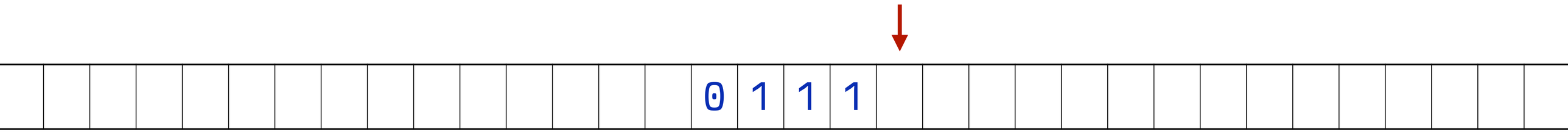

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

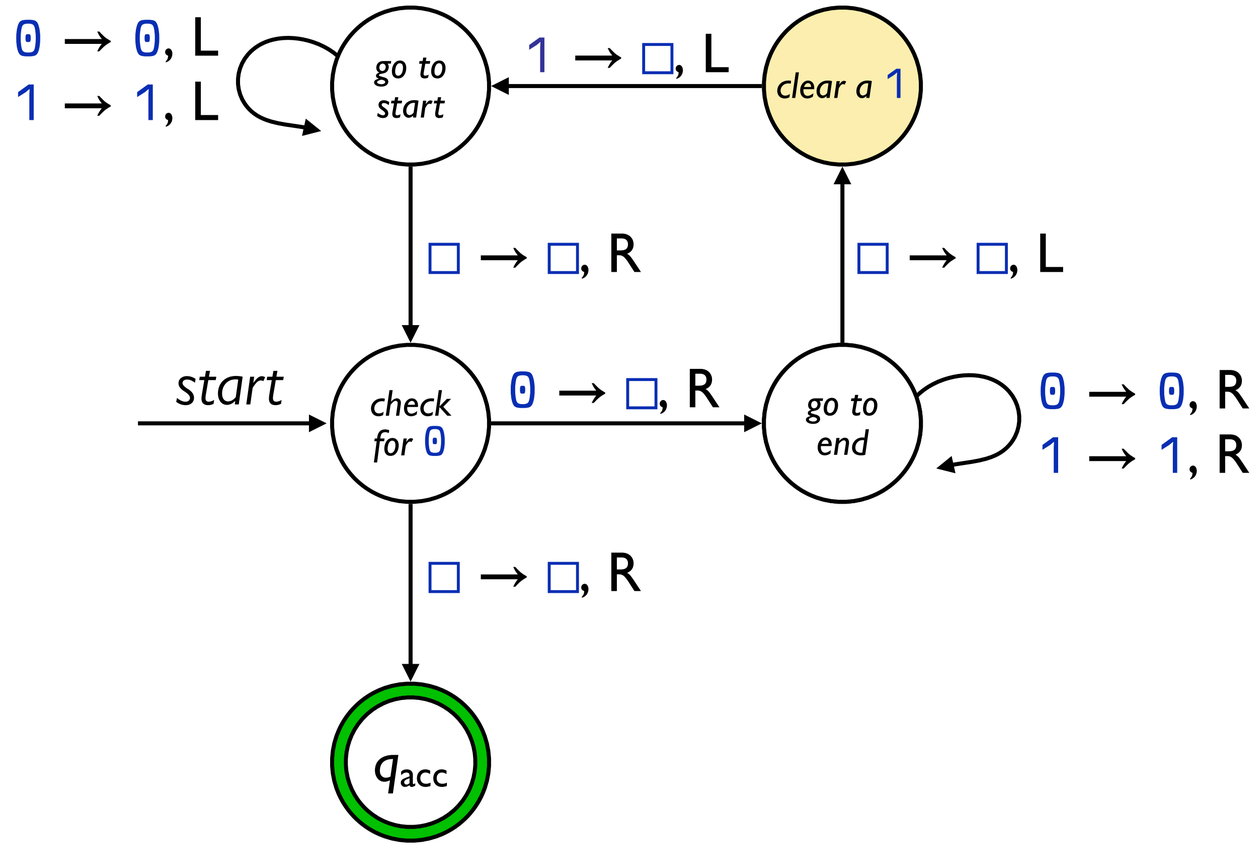

0 1 1 1

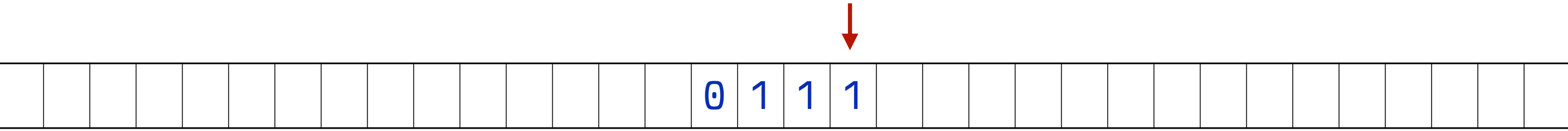

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

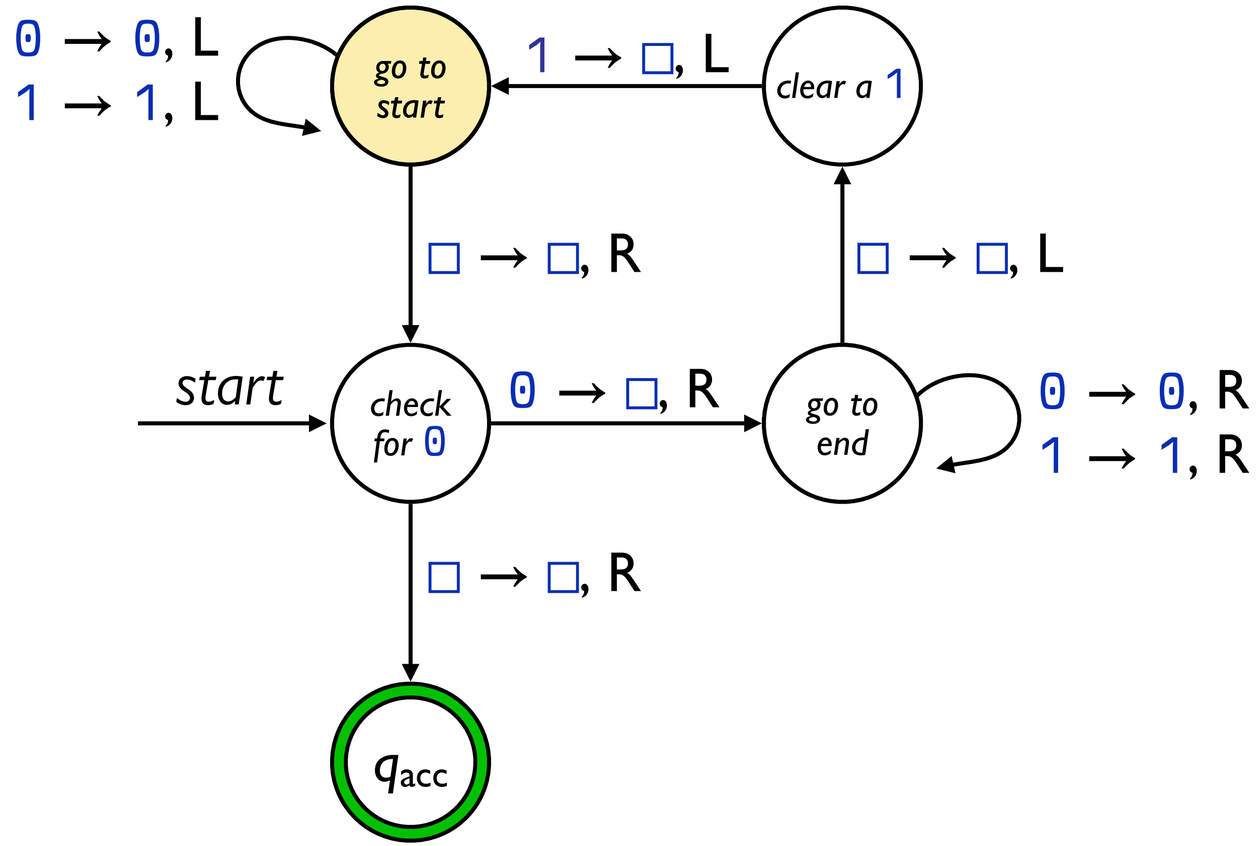

0 1 1 1

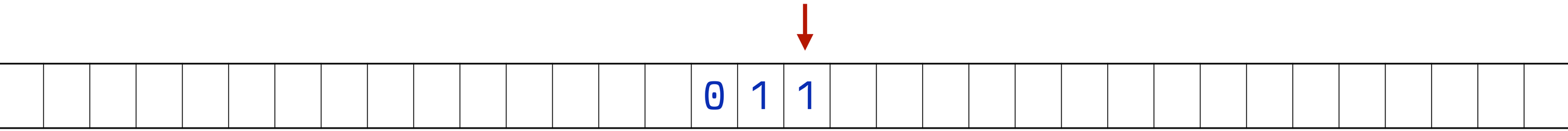

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

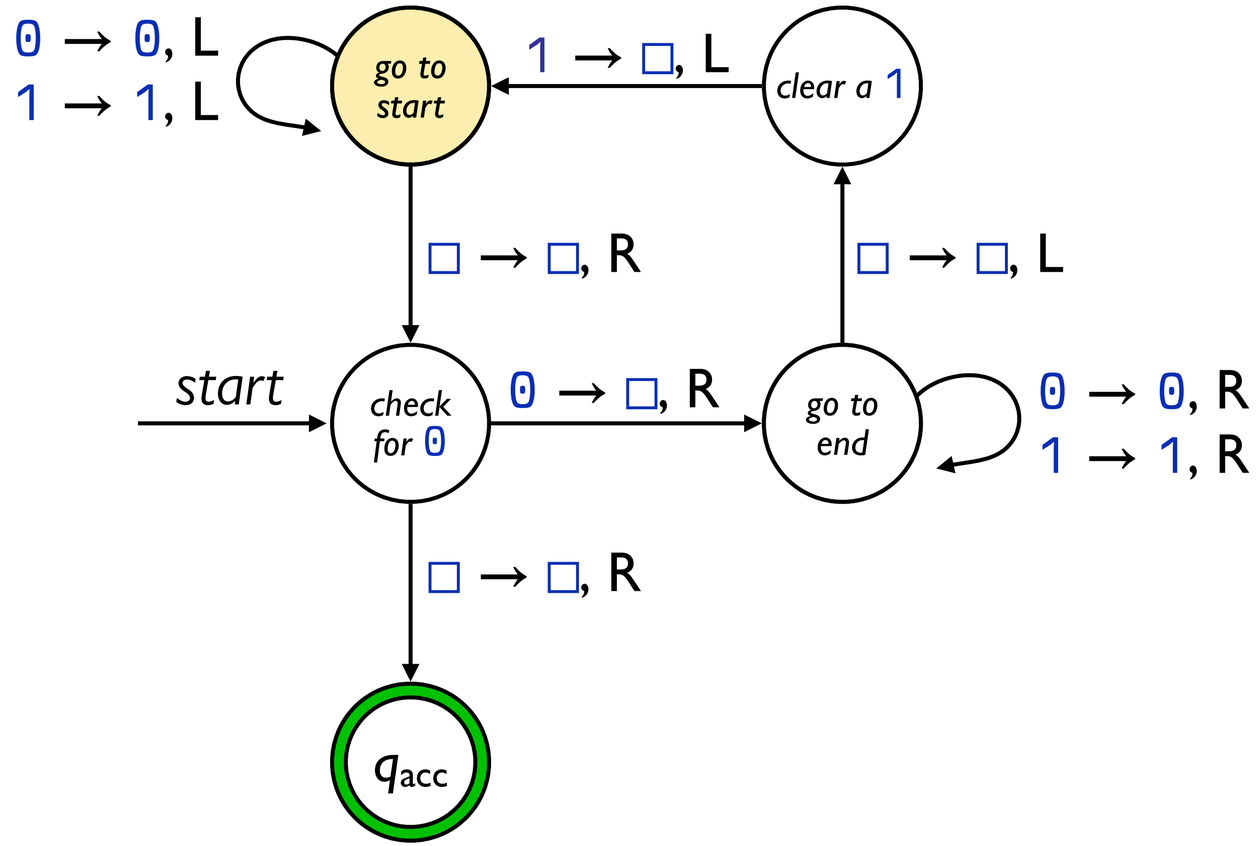

0 1 1 1

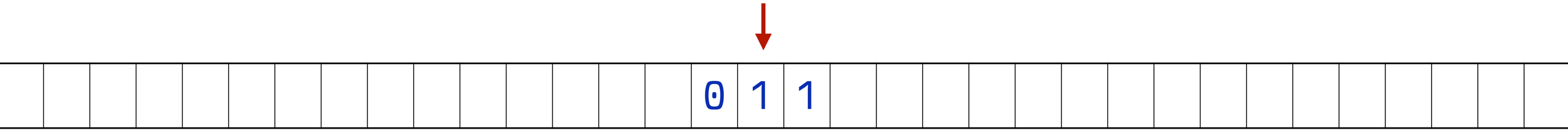

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

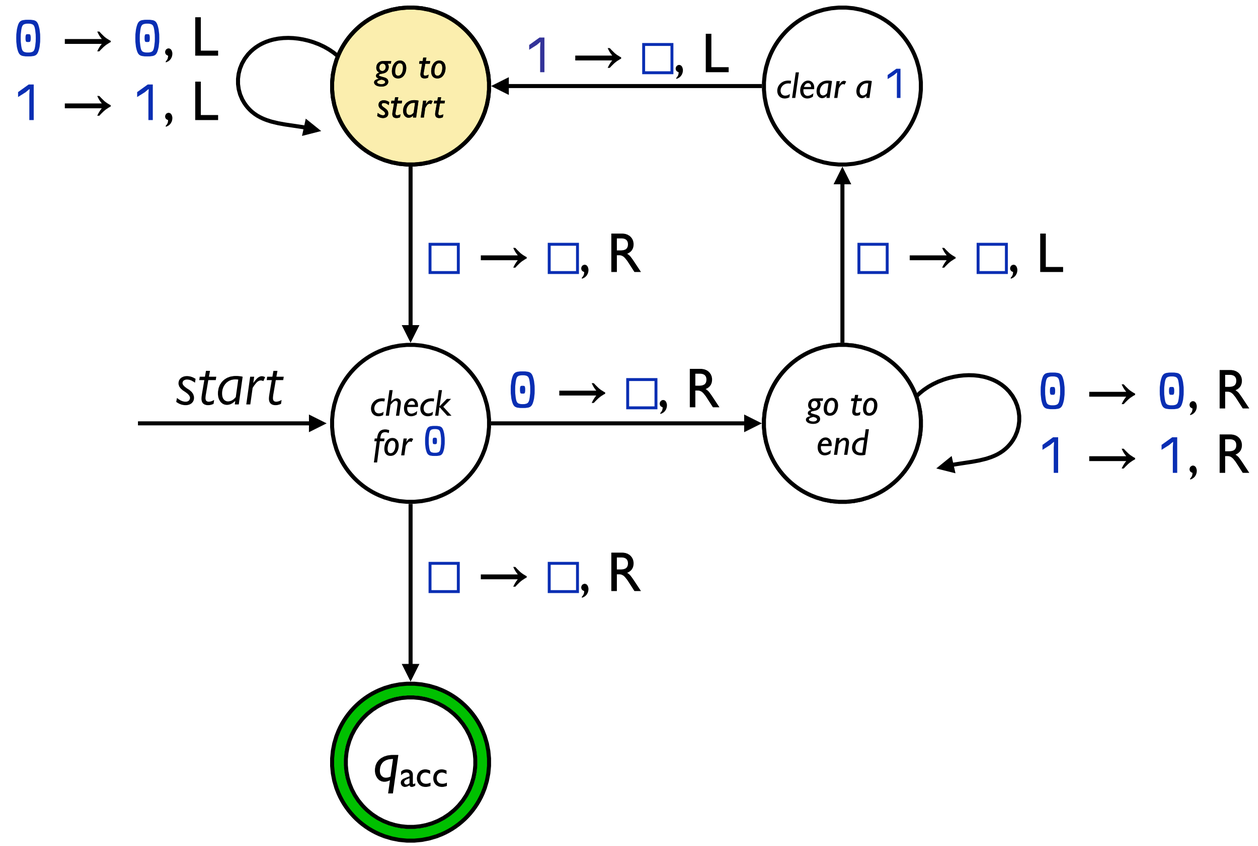

0 1 1 1

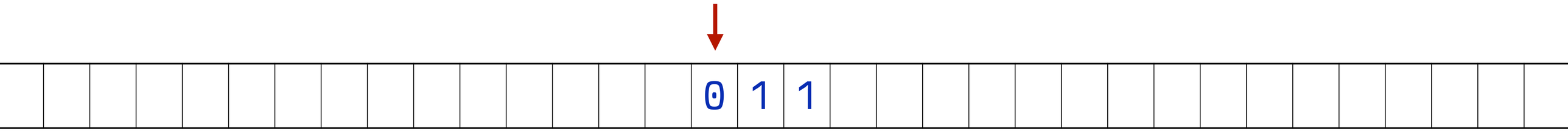

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

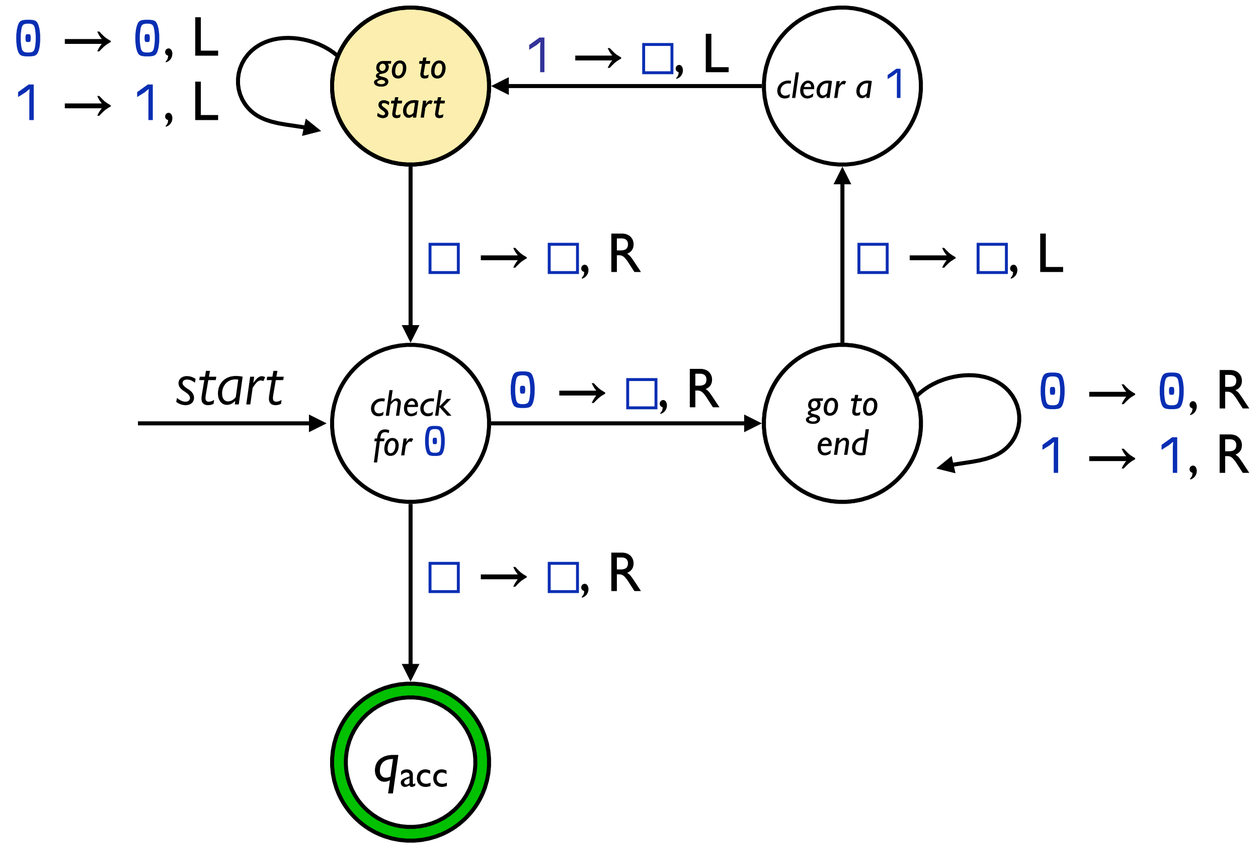

0 1 1

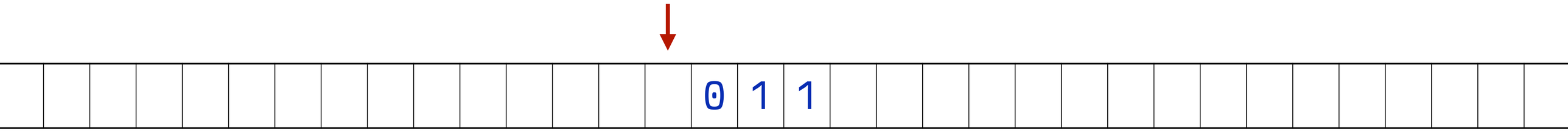

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

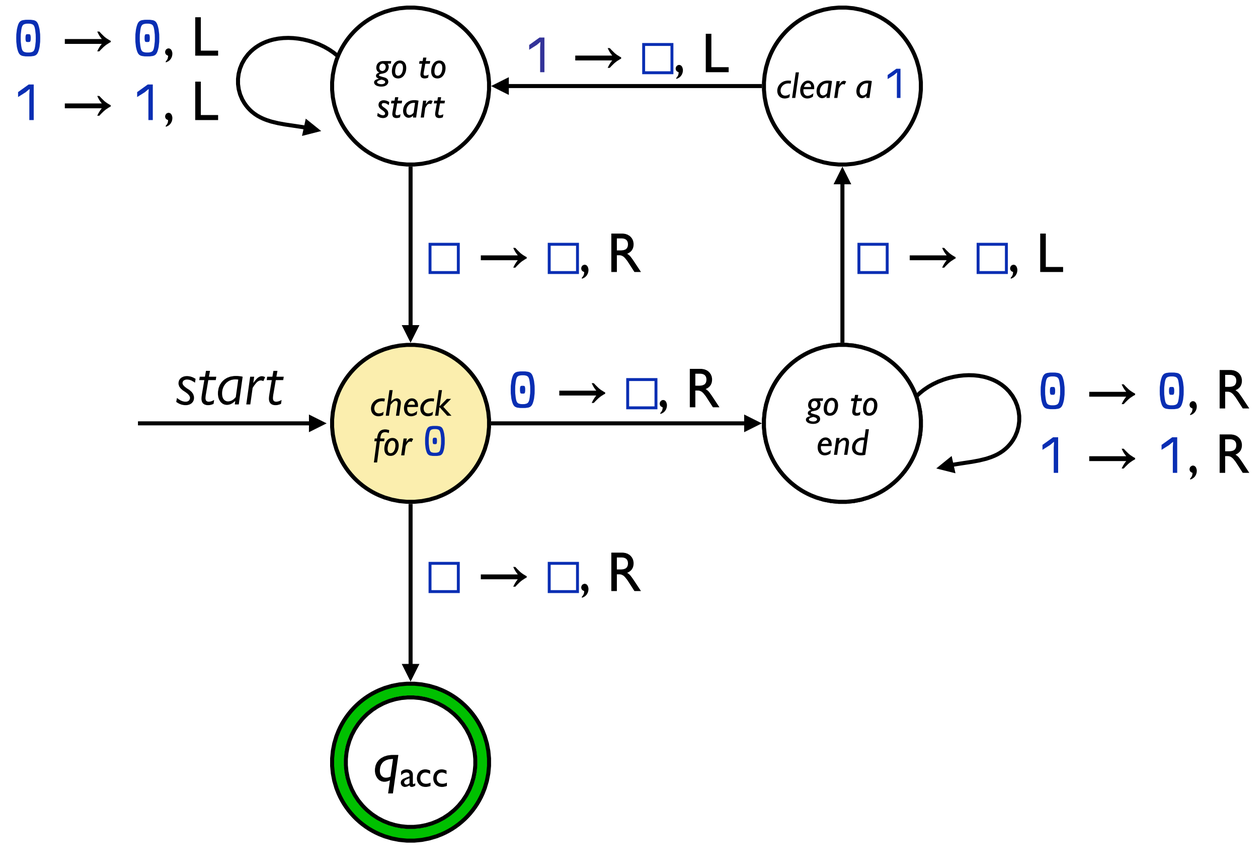

0 1 1

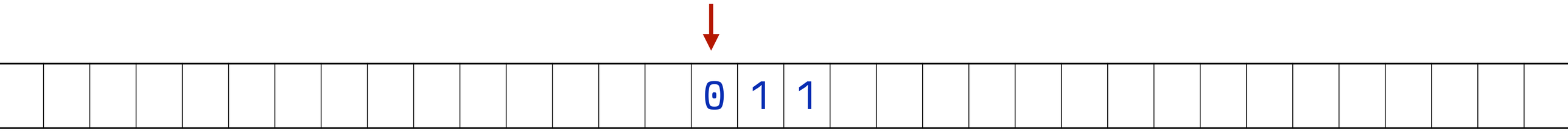

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

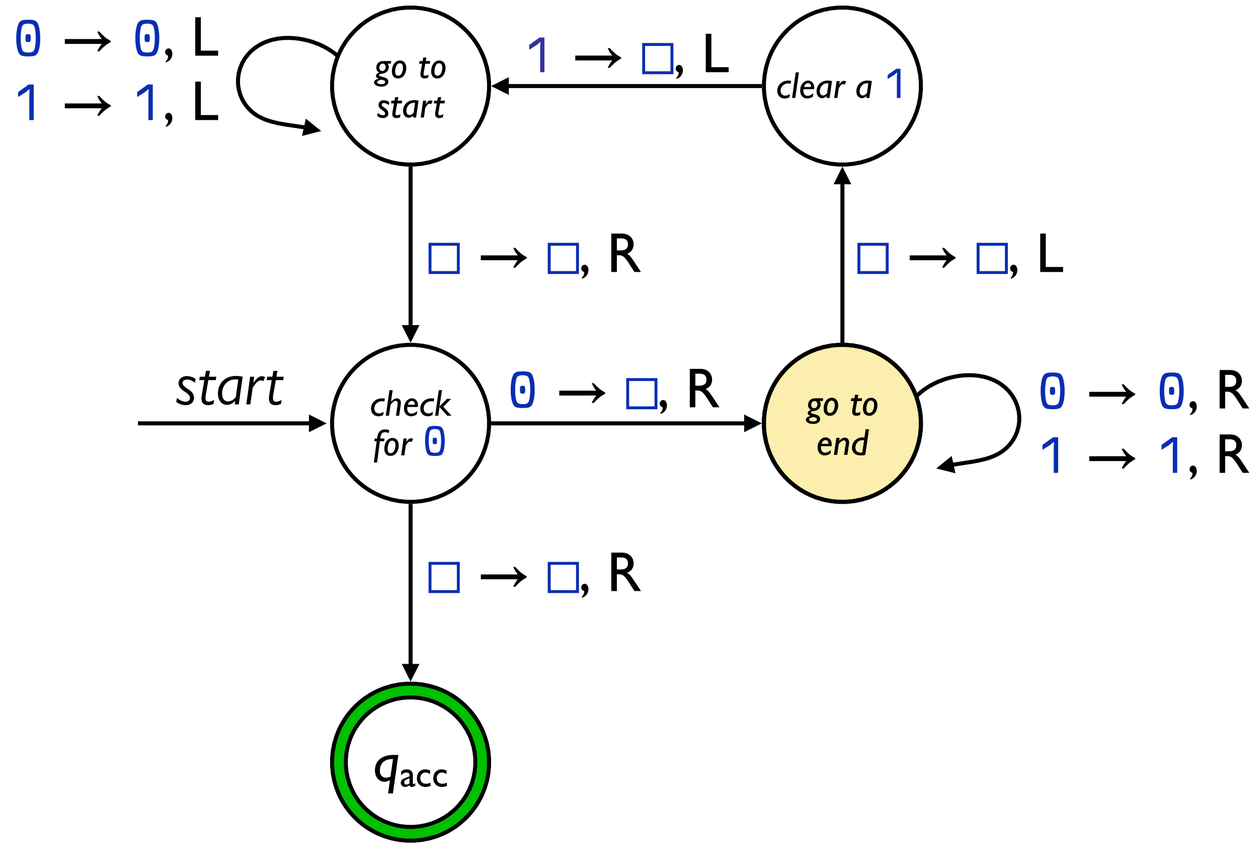

0 1 1

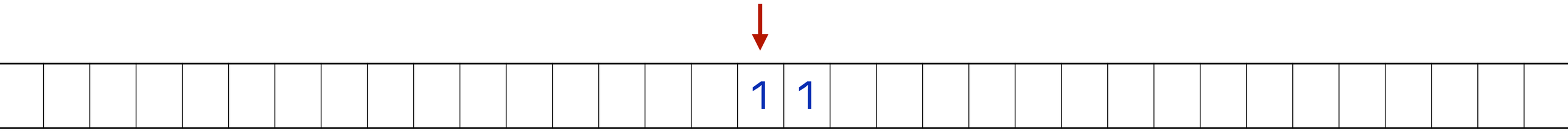

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

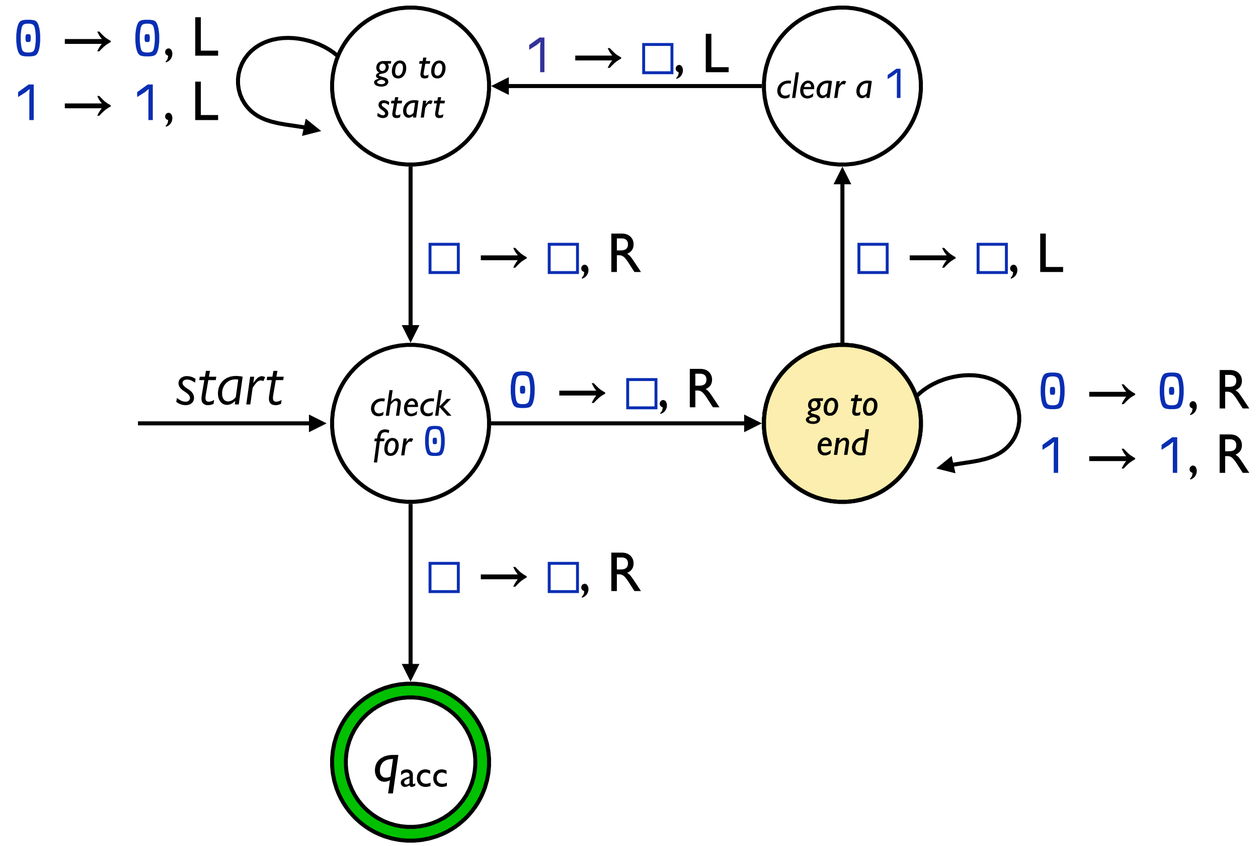

0 1 1

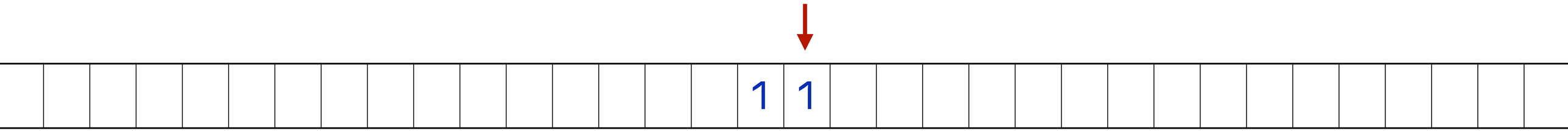

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

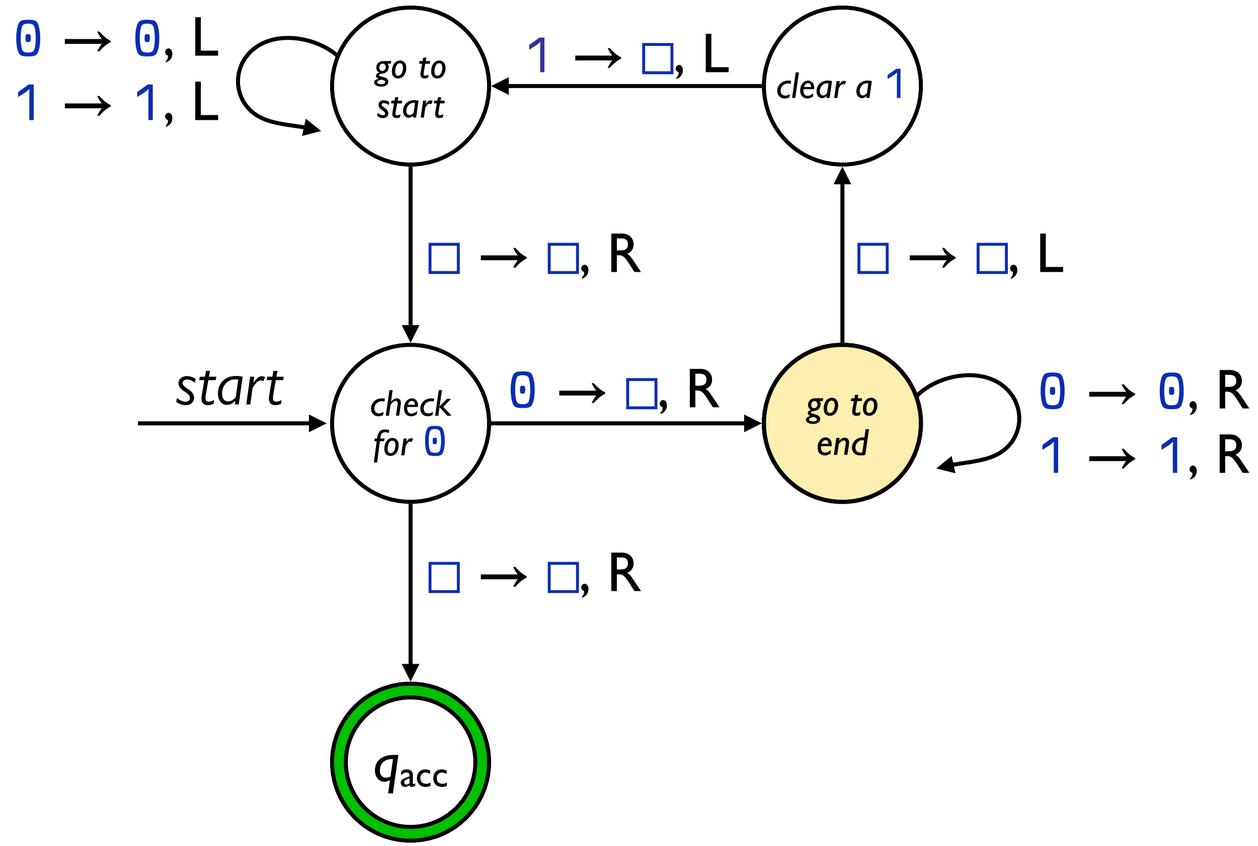

0 1 1

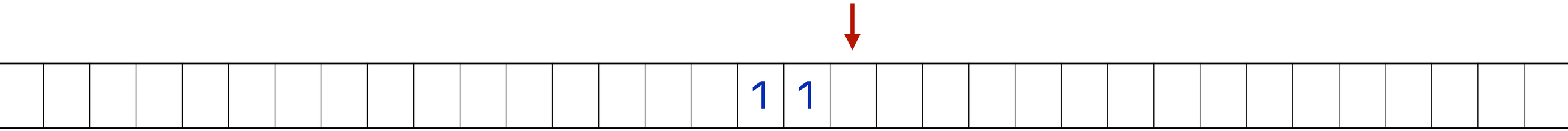

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

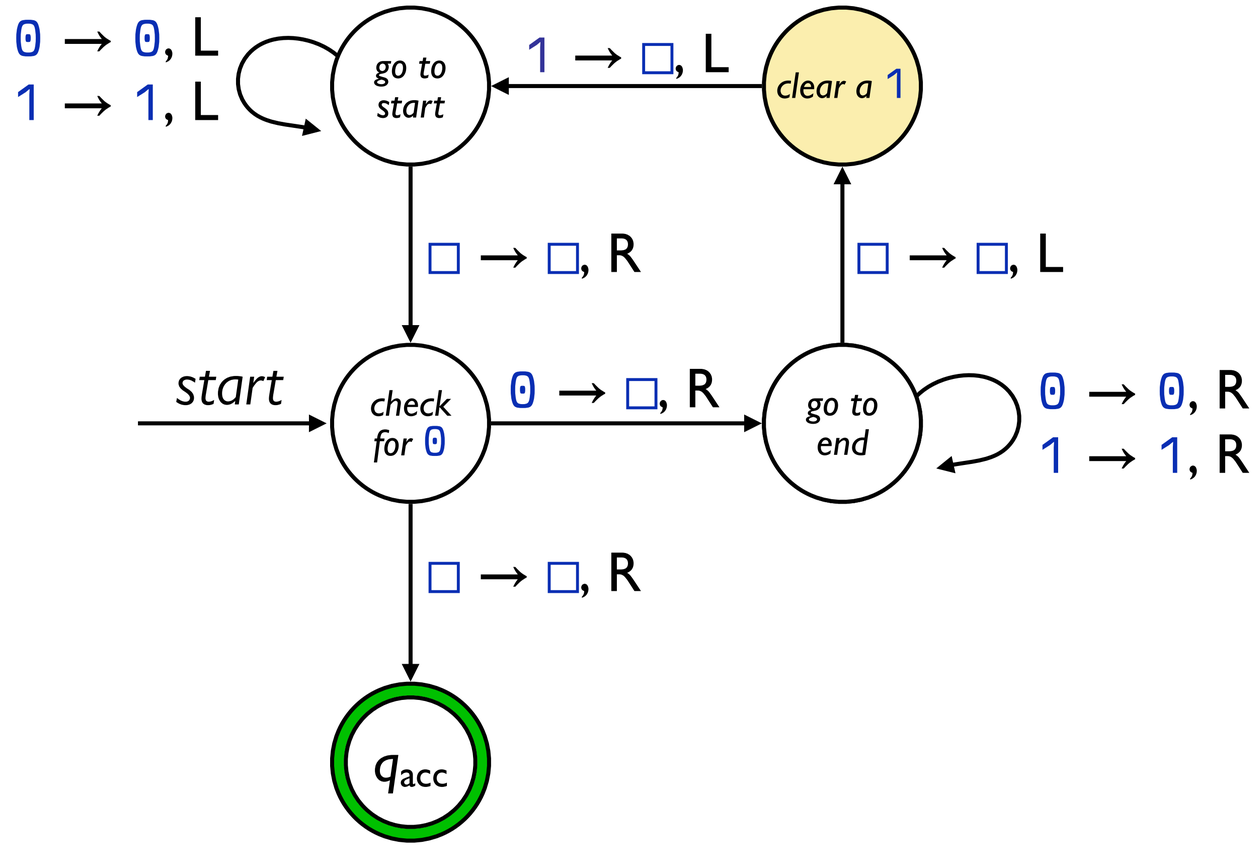

1 1

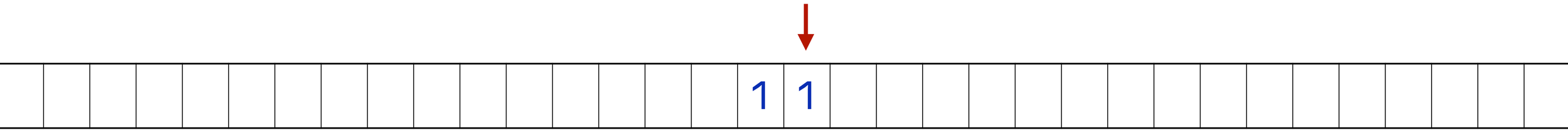

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

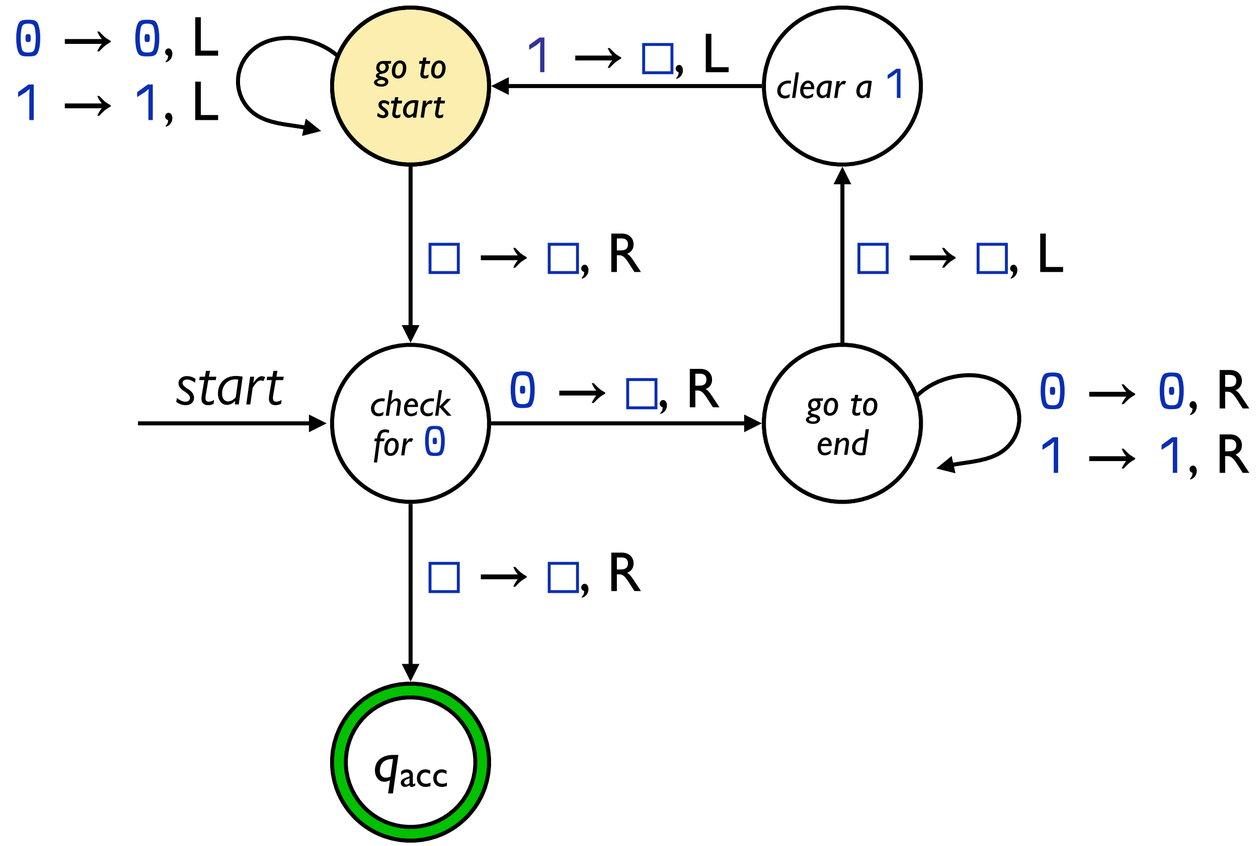

1 1

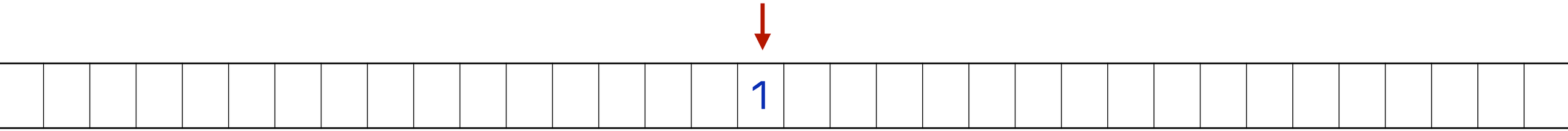

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

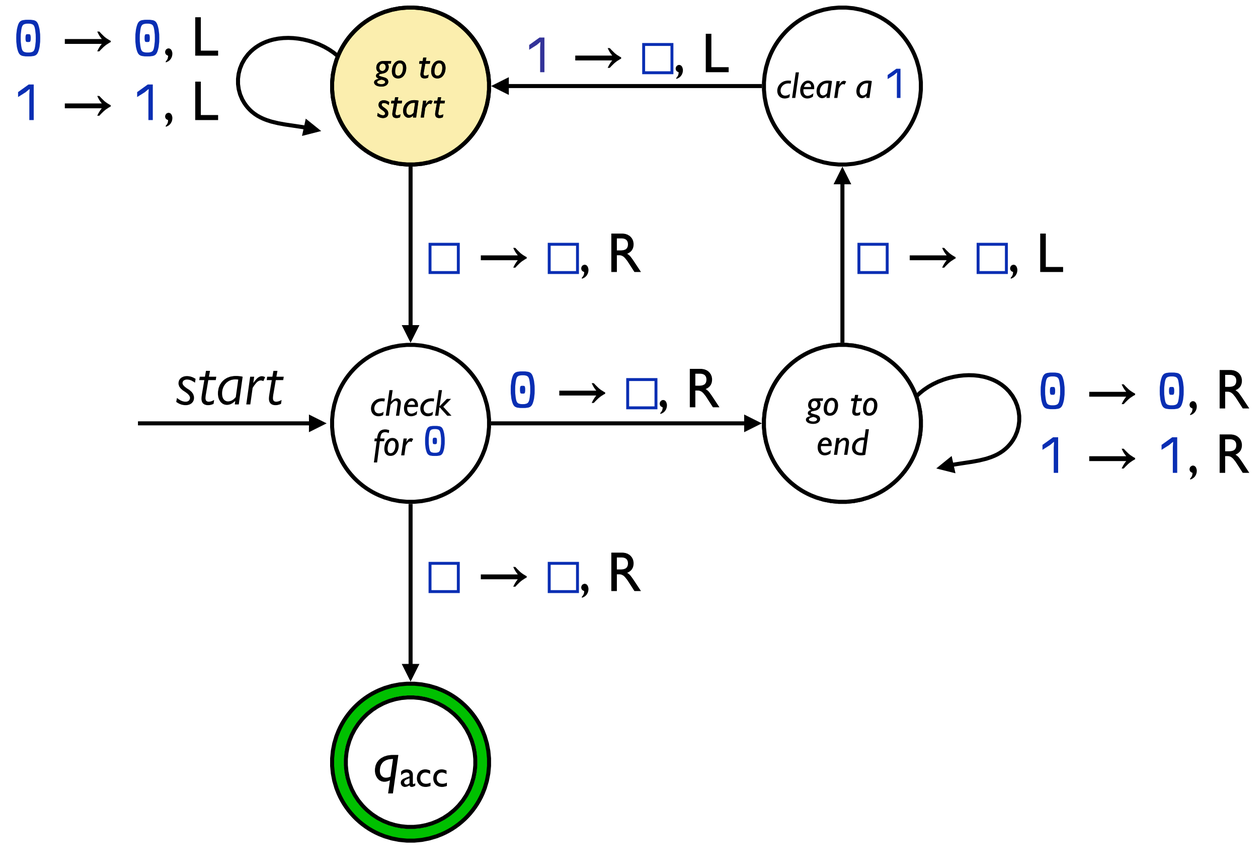

1 1

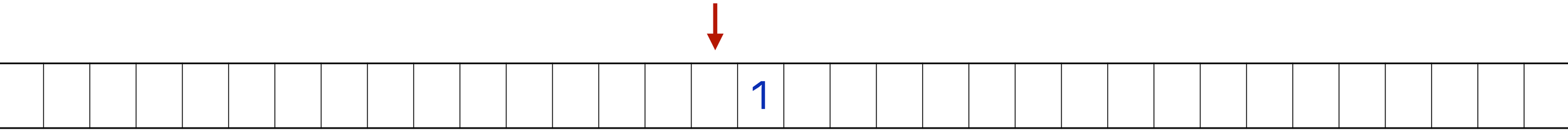

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

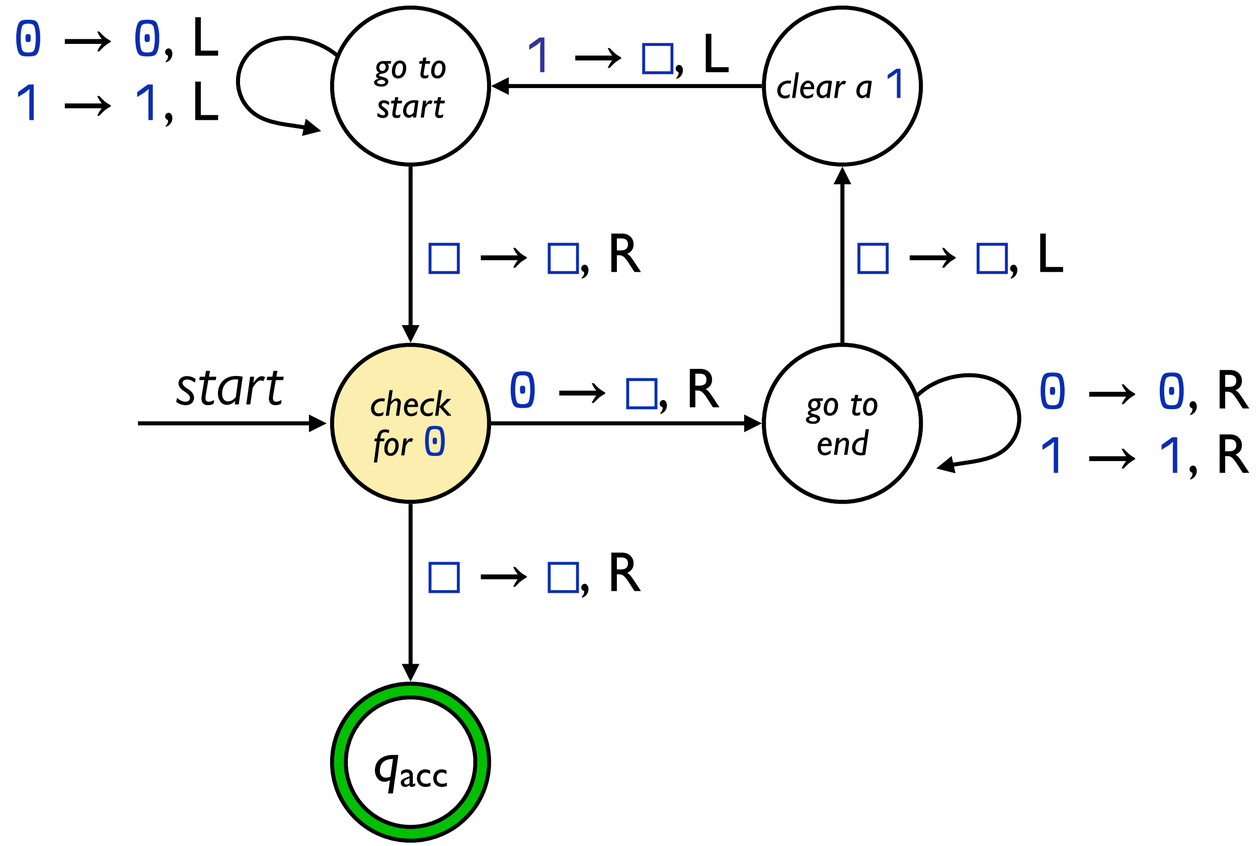

1 1

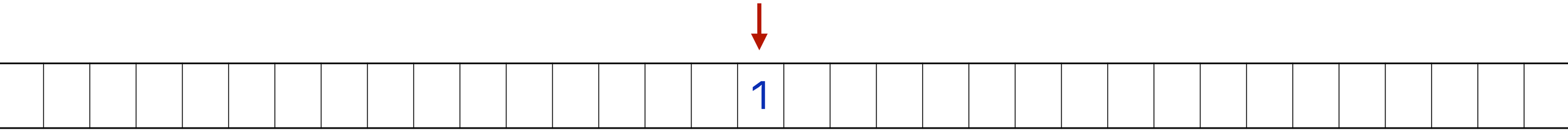

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

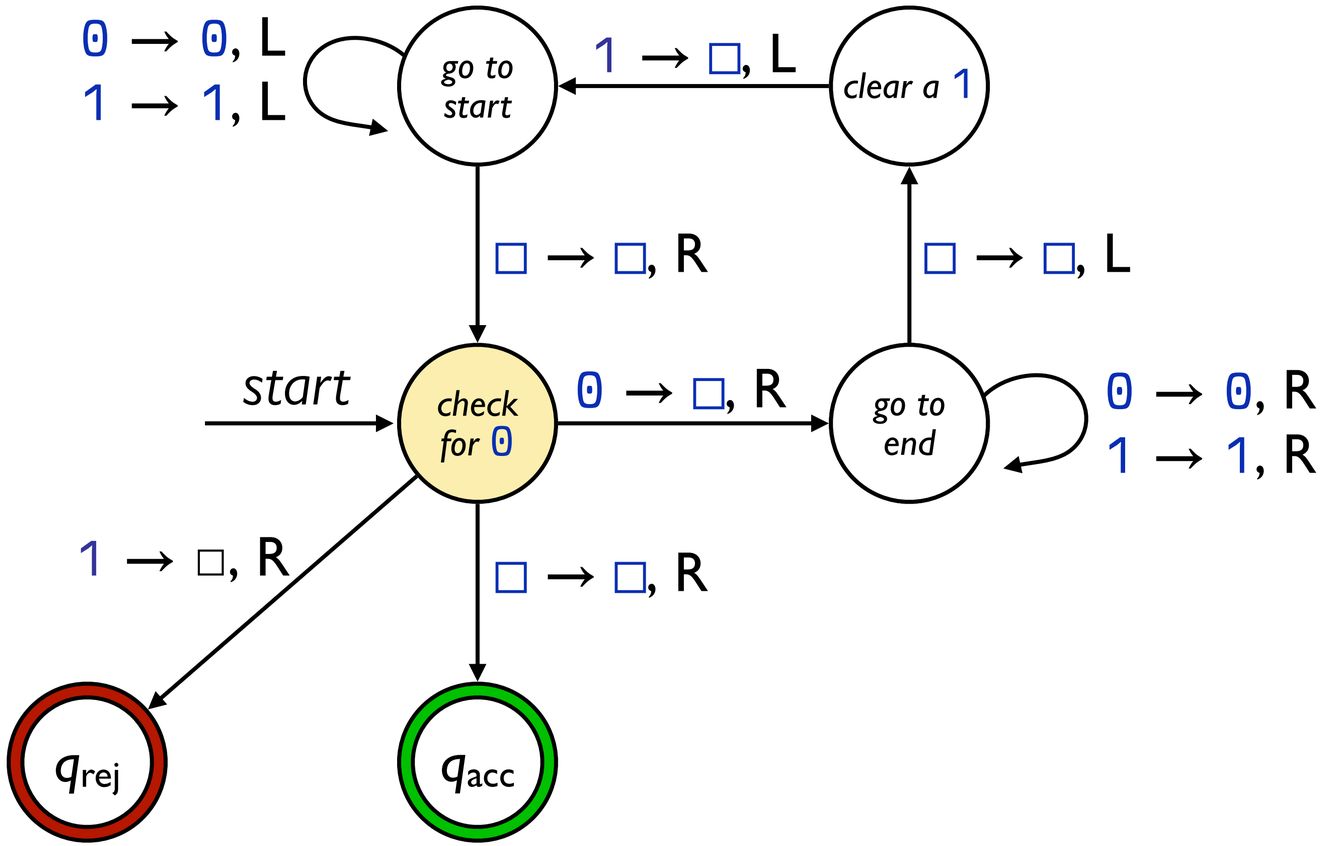

1

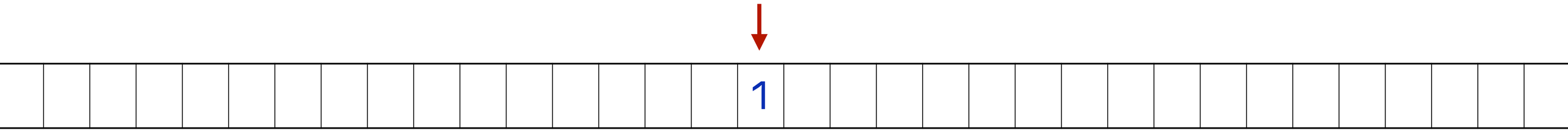

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

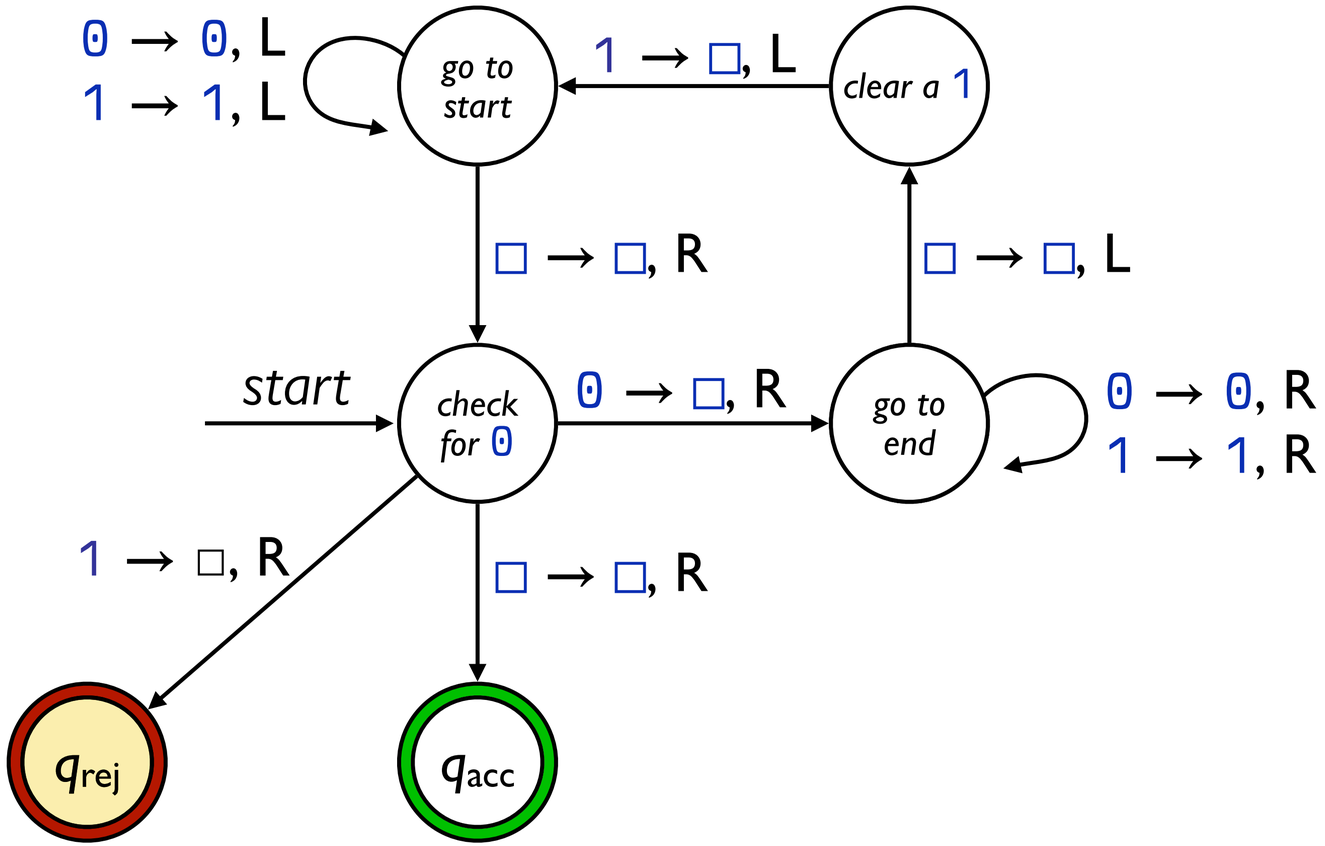

1

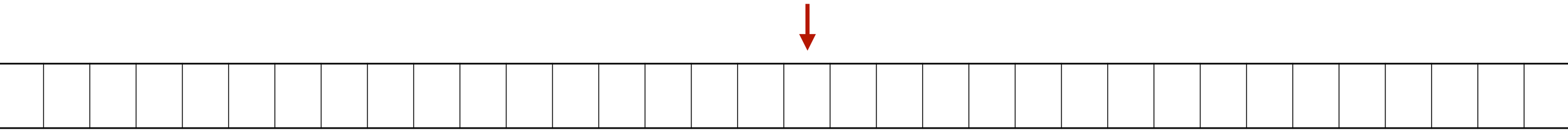

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
□ → □, R
qacc

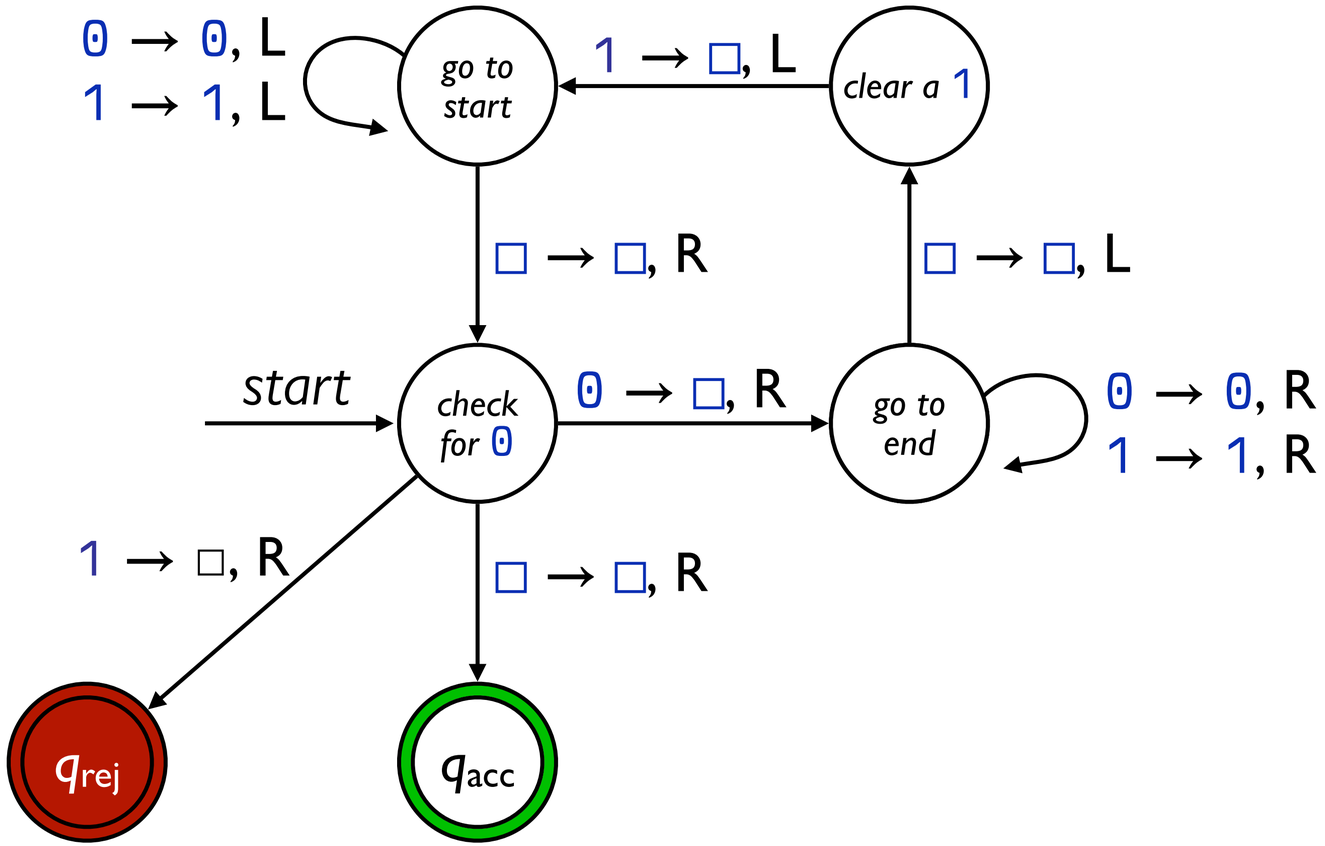

1

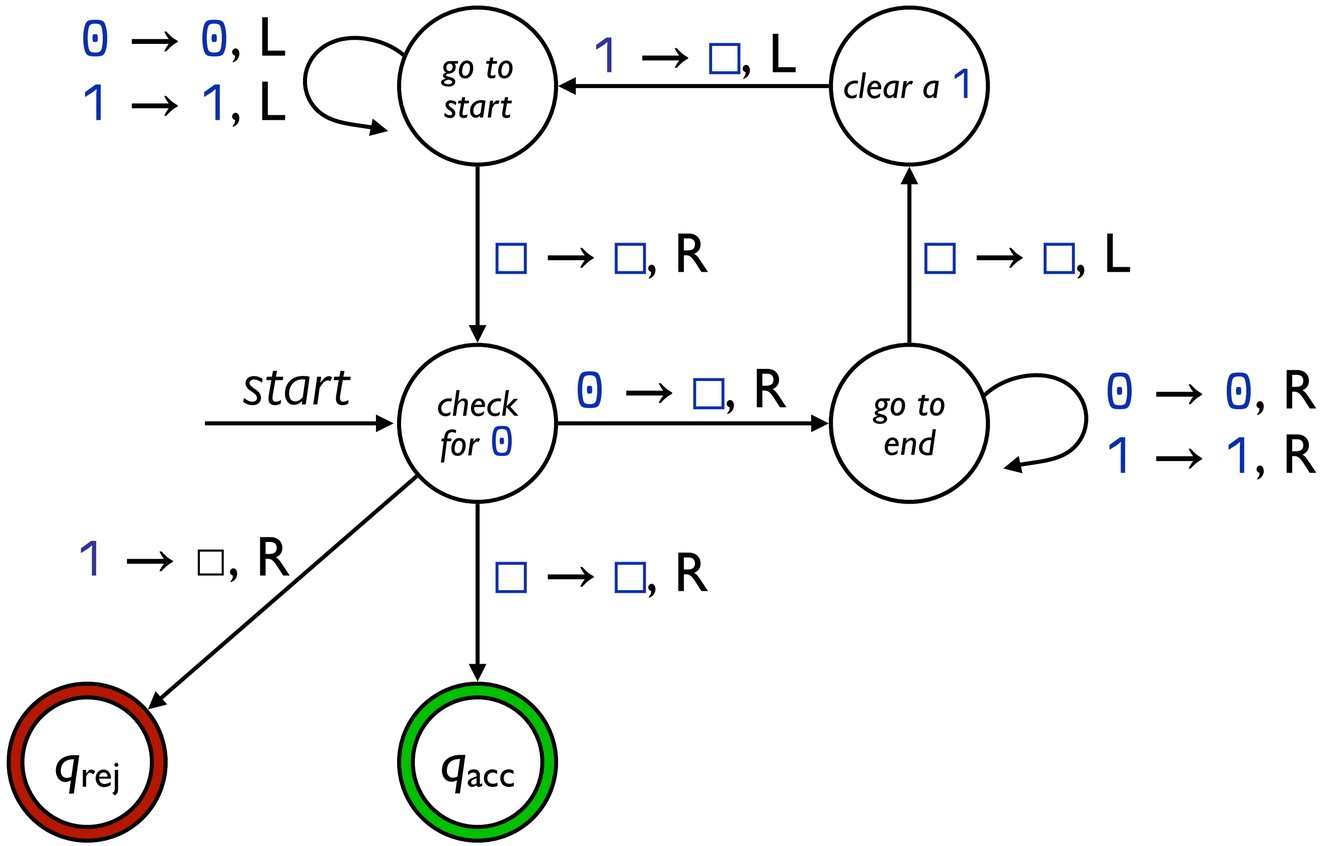

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

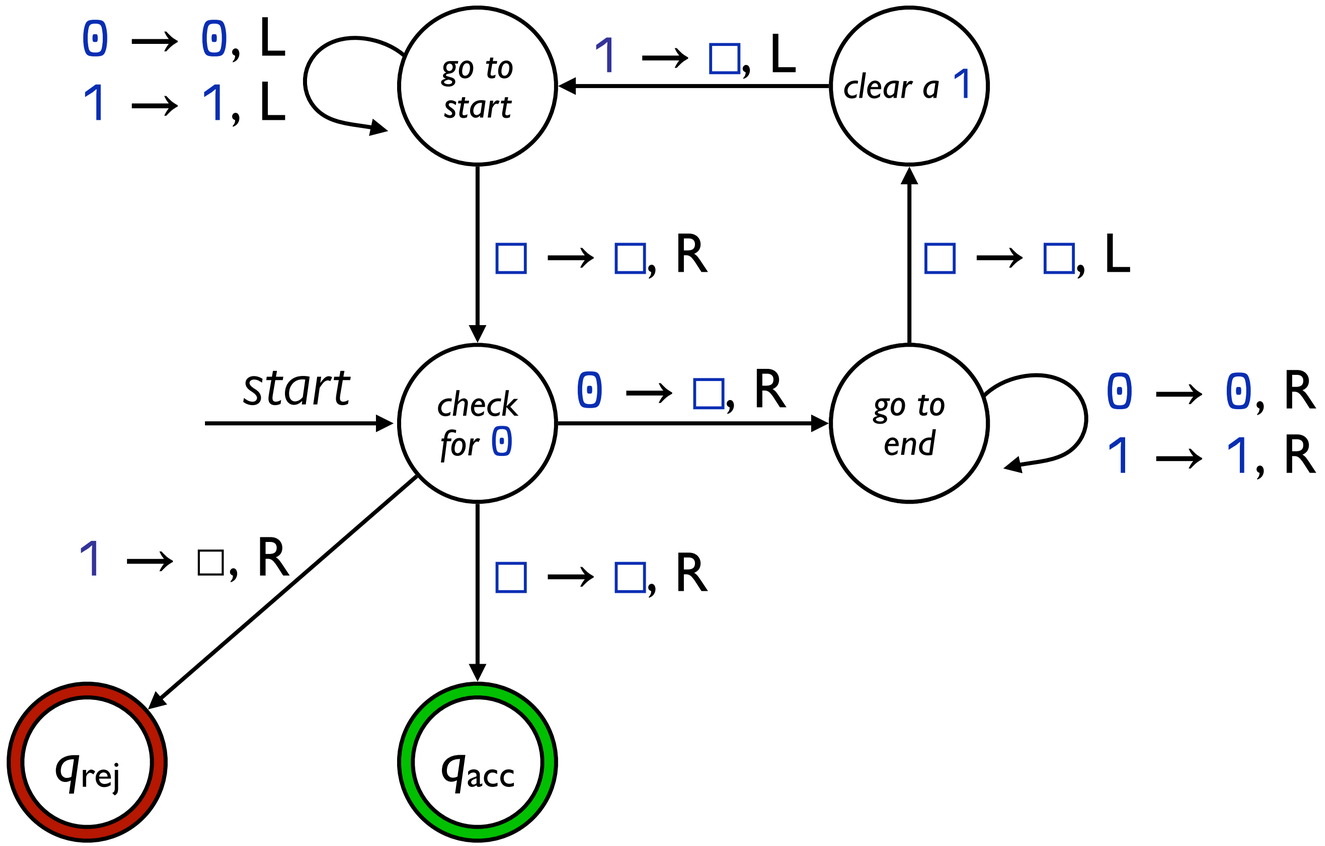

1

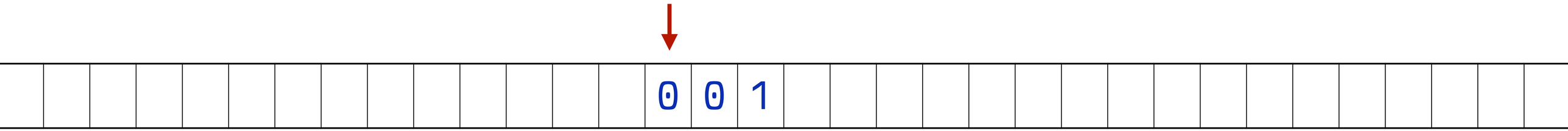

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

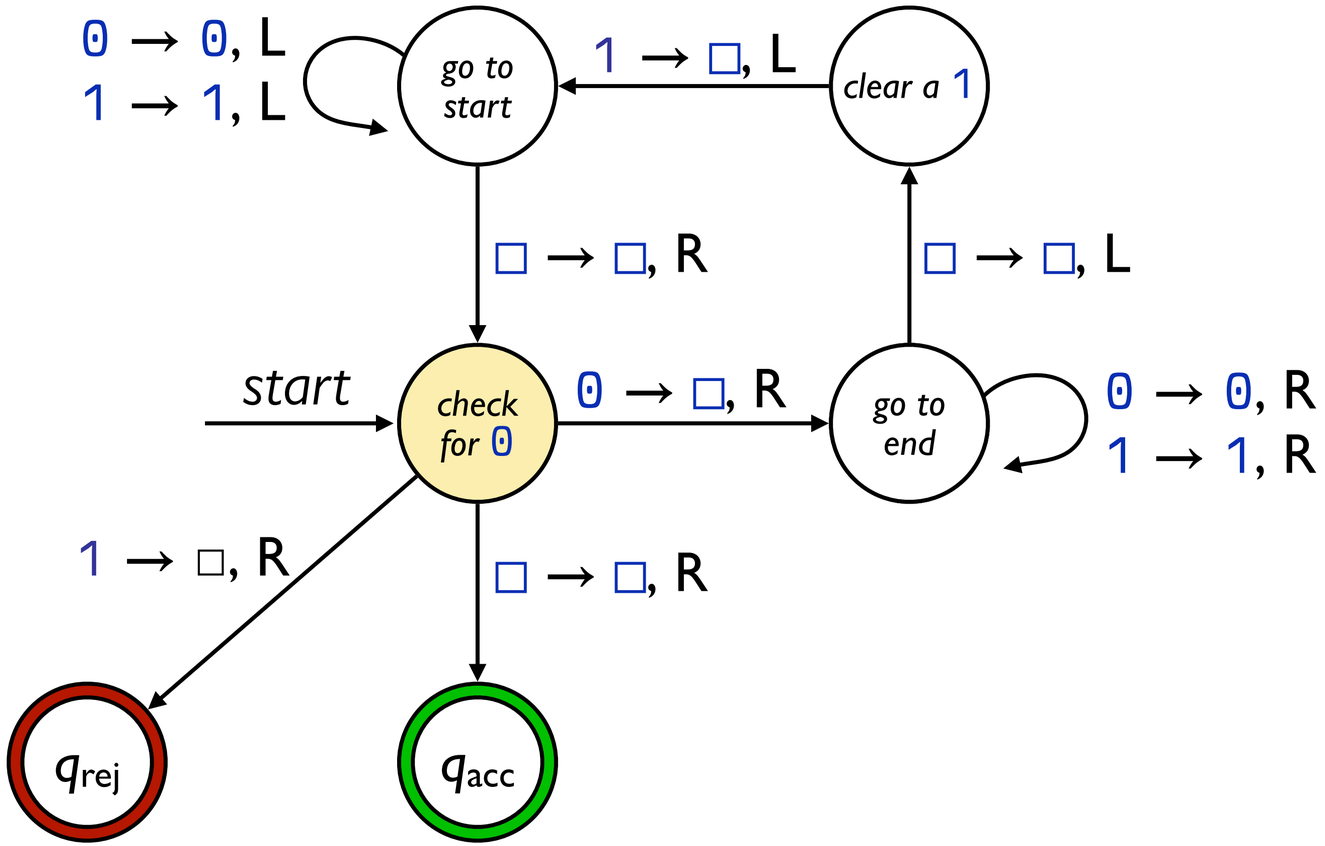


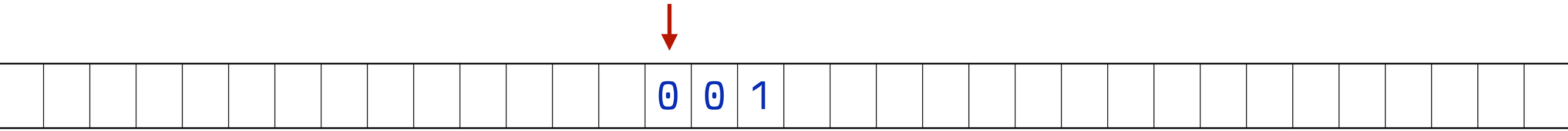

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

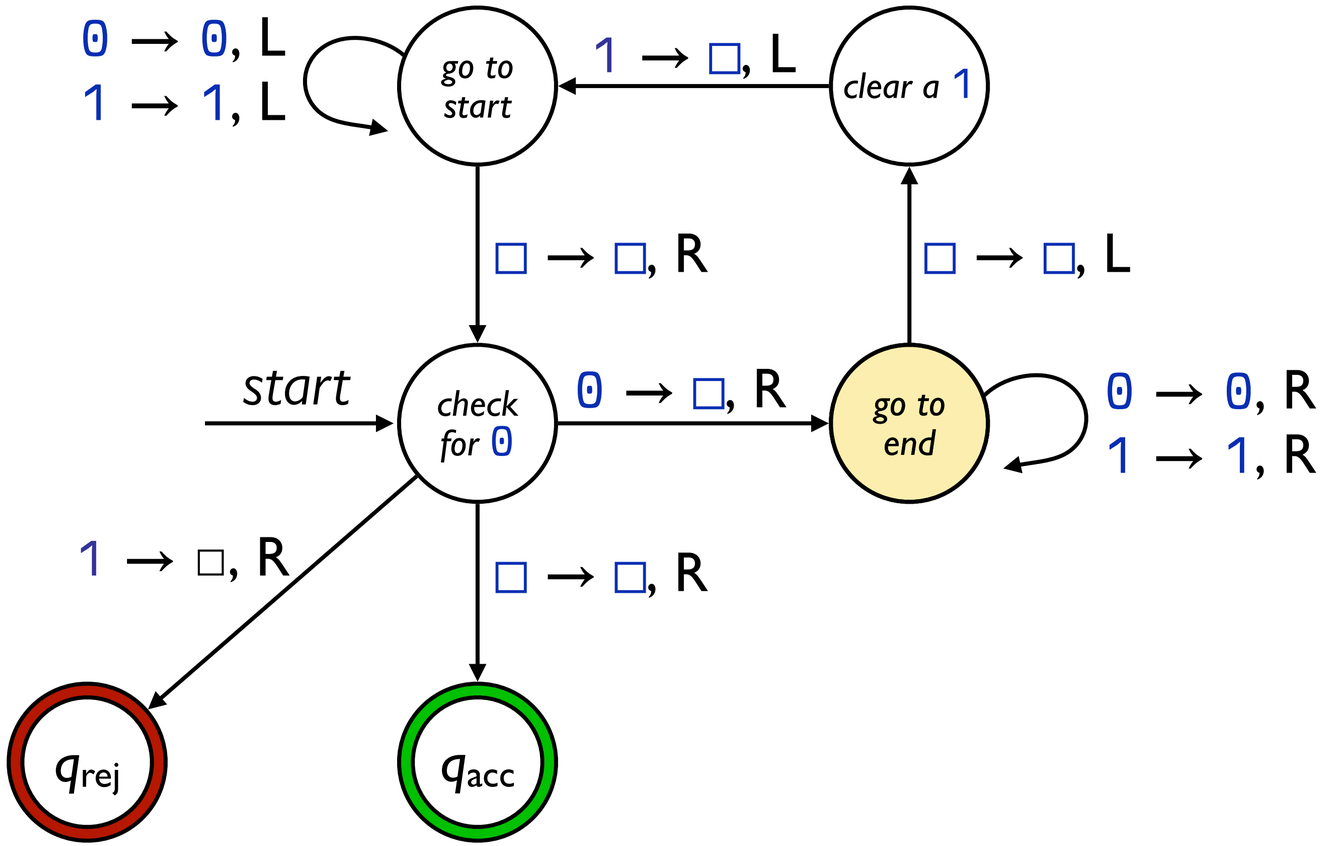

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

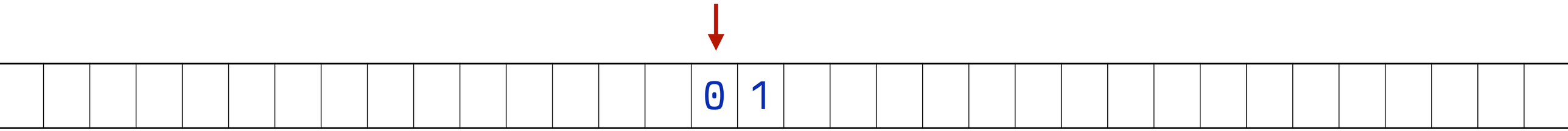

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

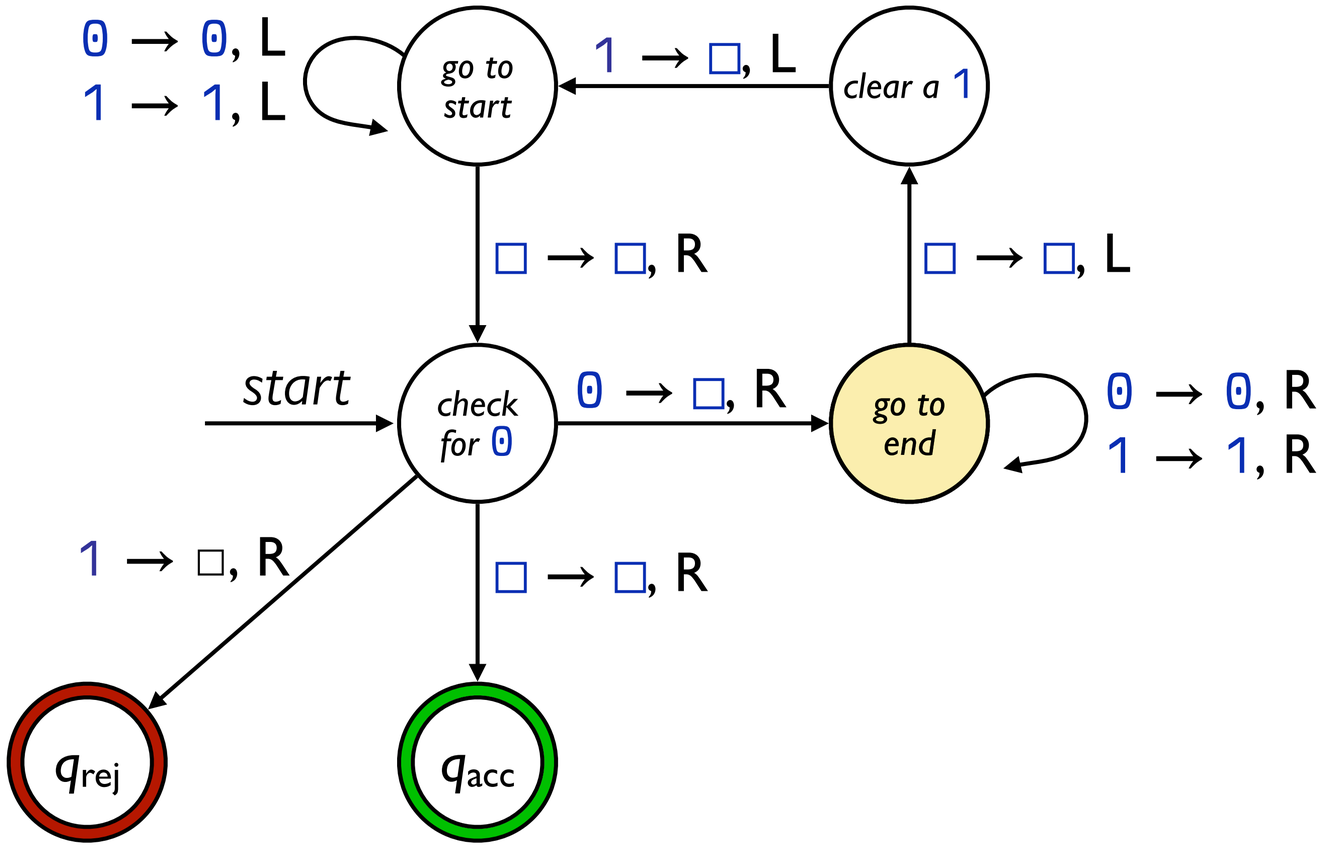

0 0 1

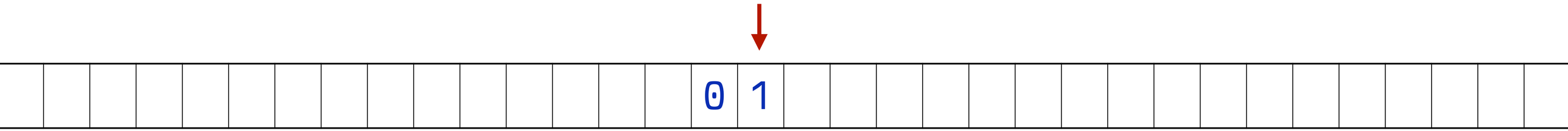

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

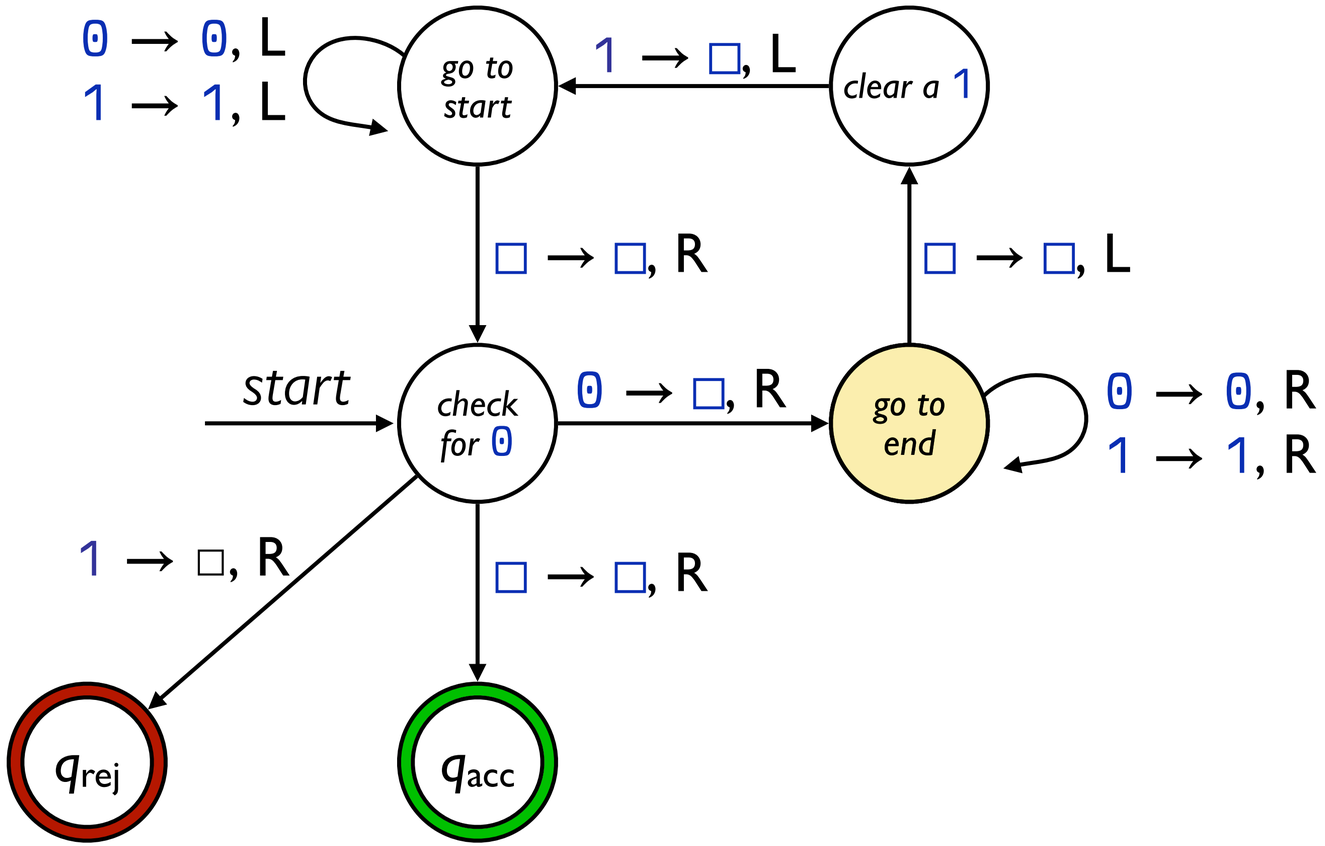

0 0 1

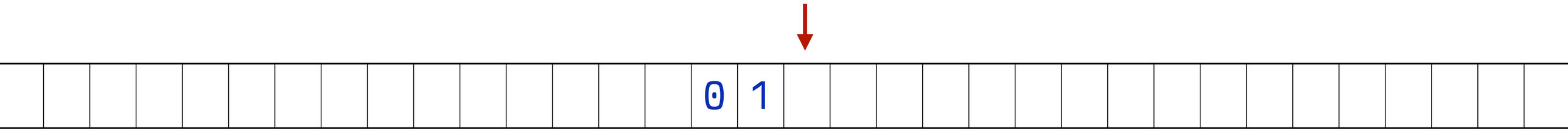

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

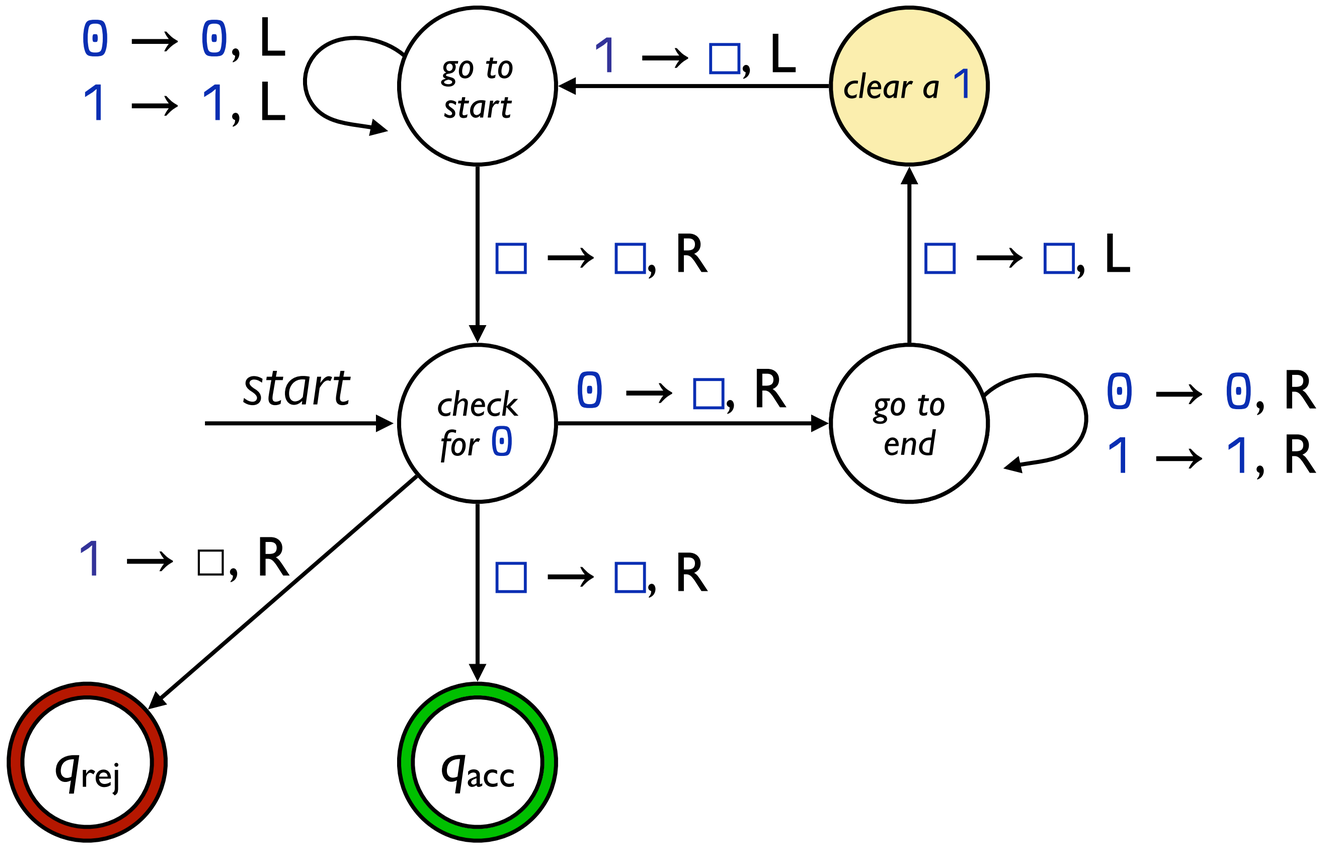

0 1

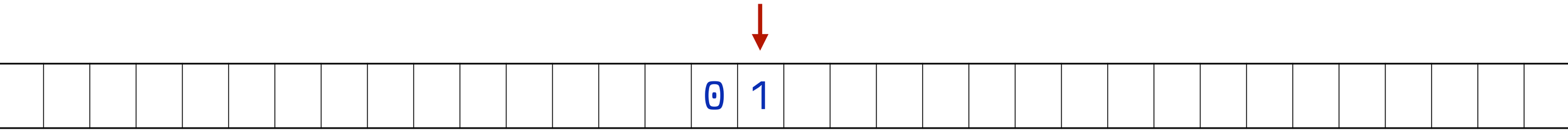

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

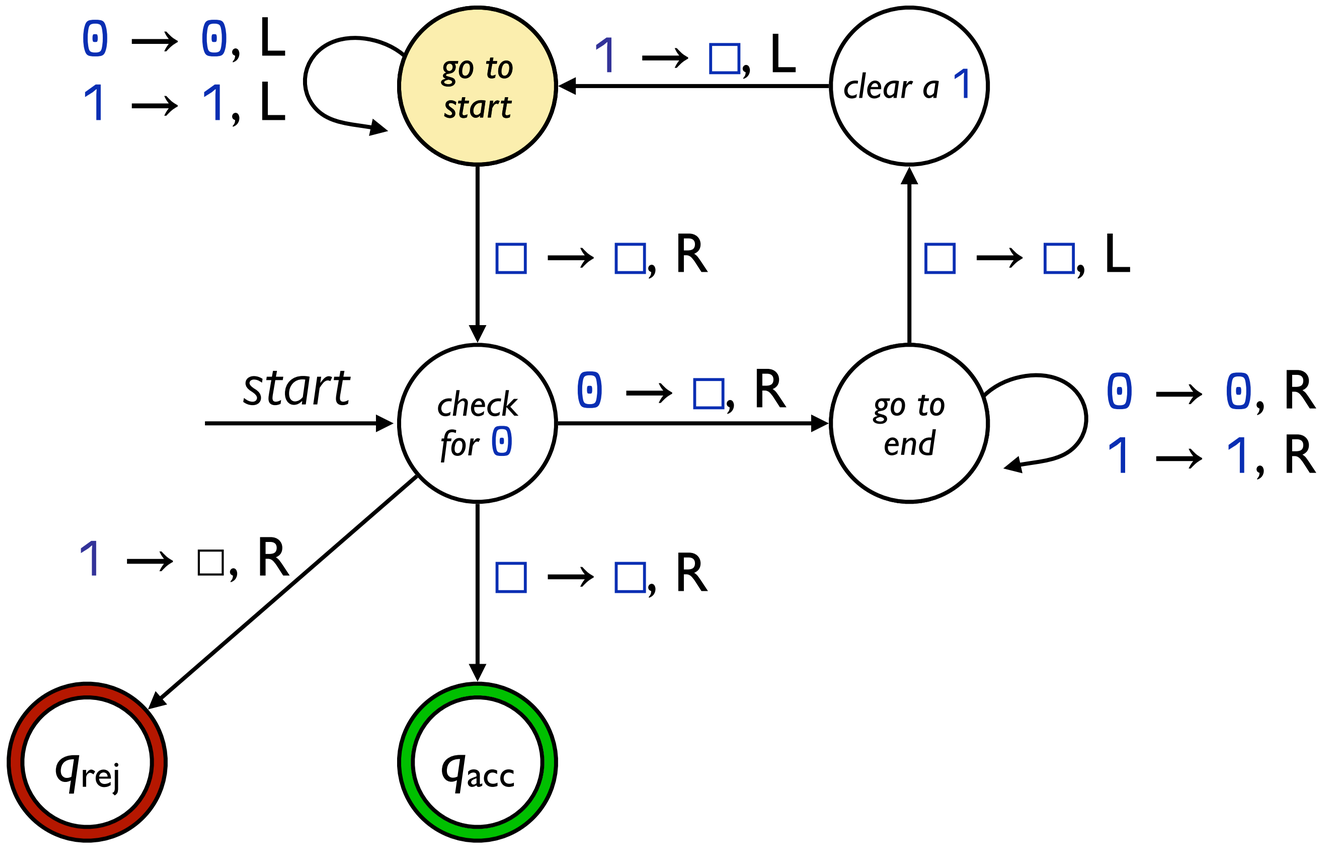

0 1

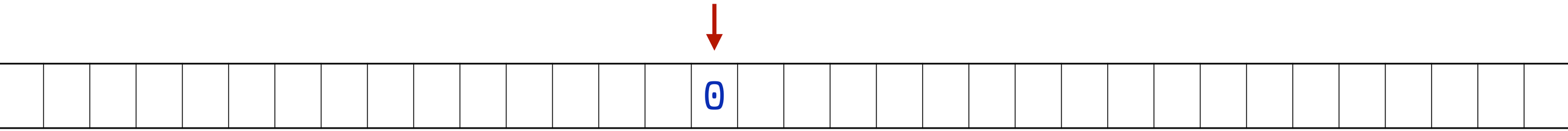

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

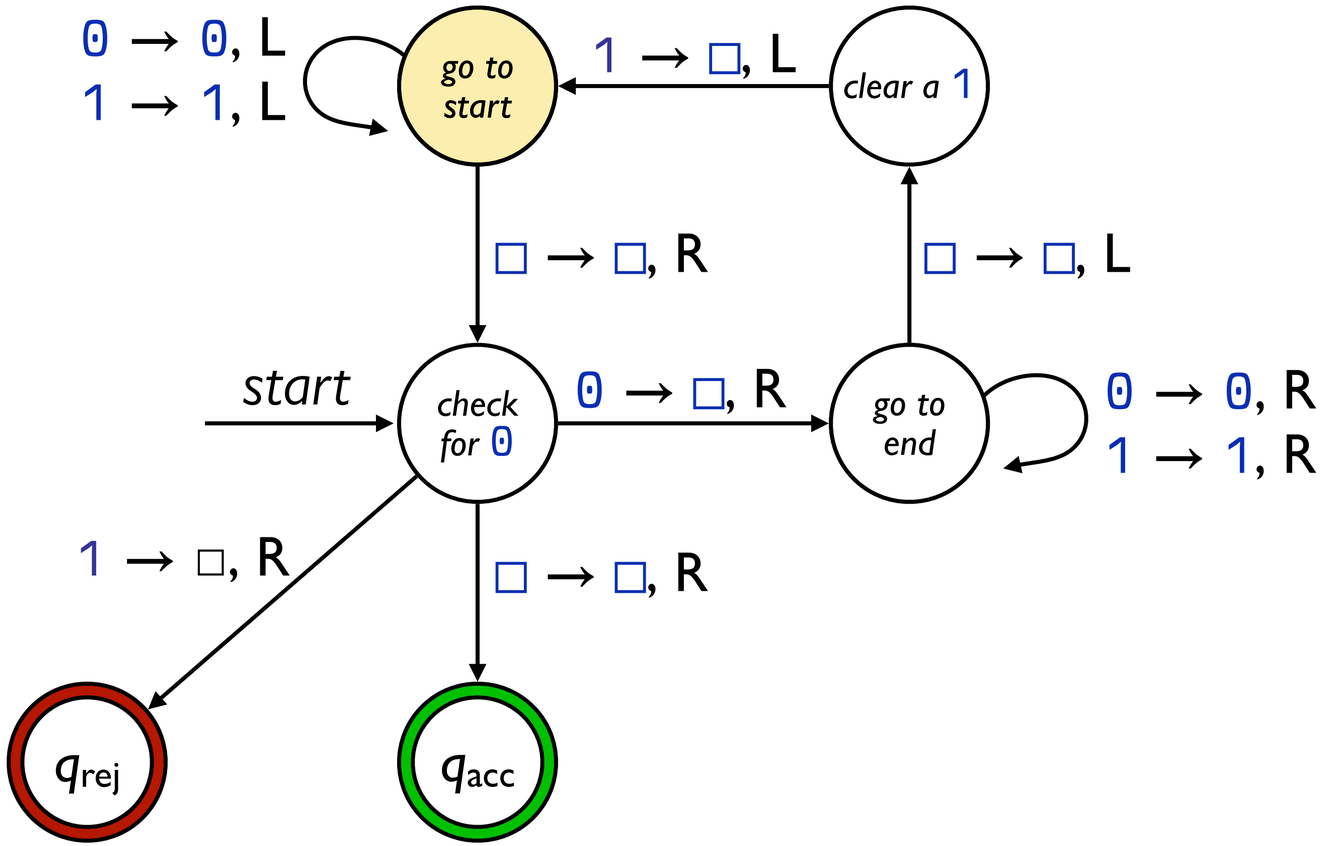

0 1

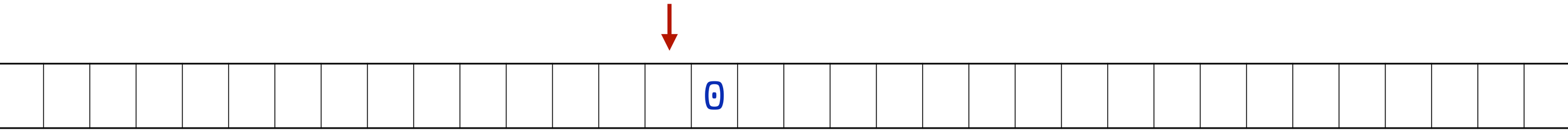

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

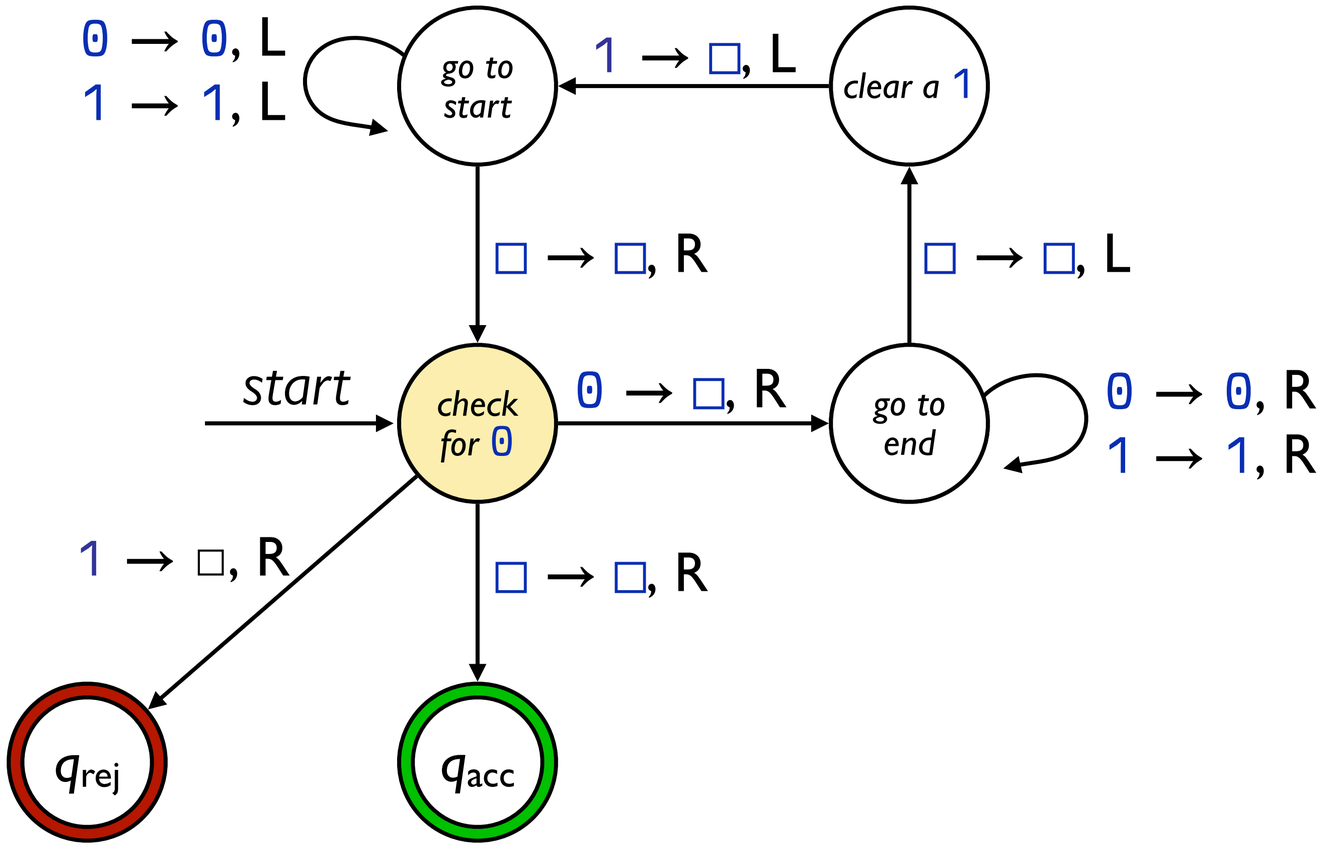

0 1

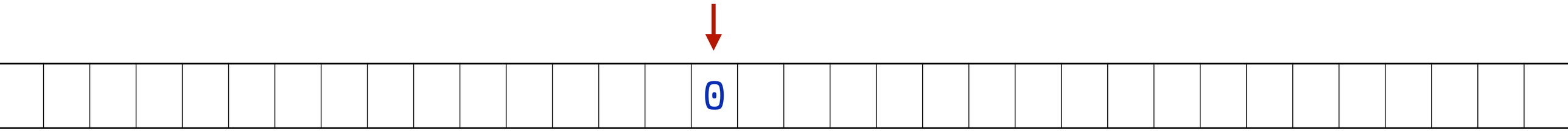

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

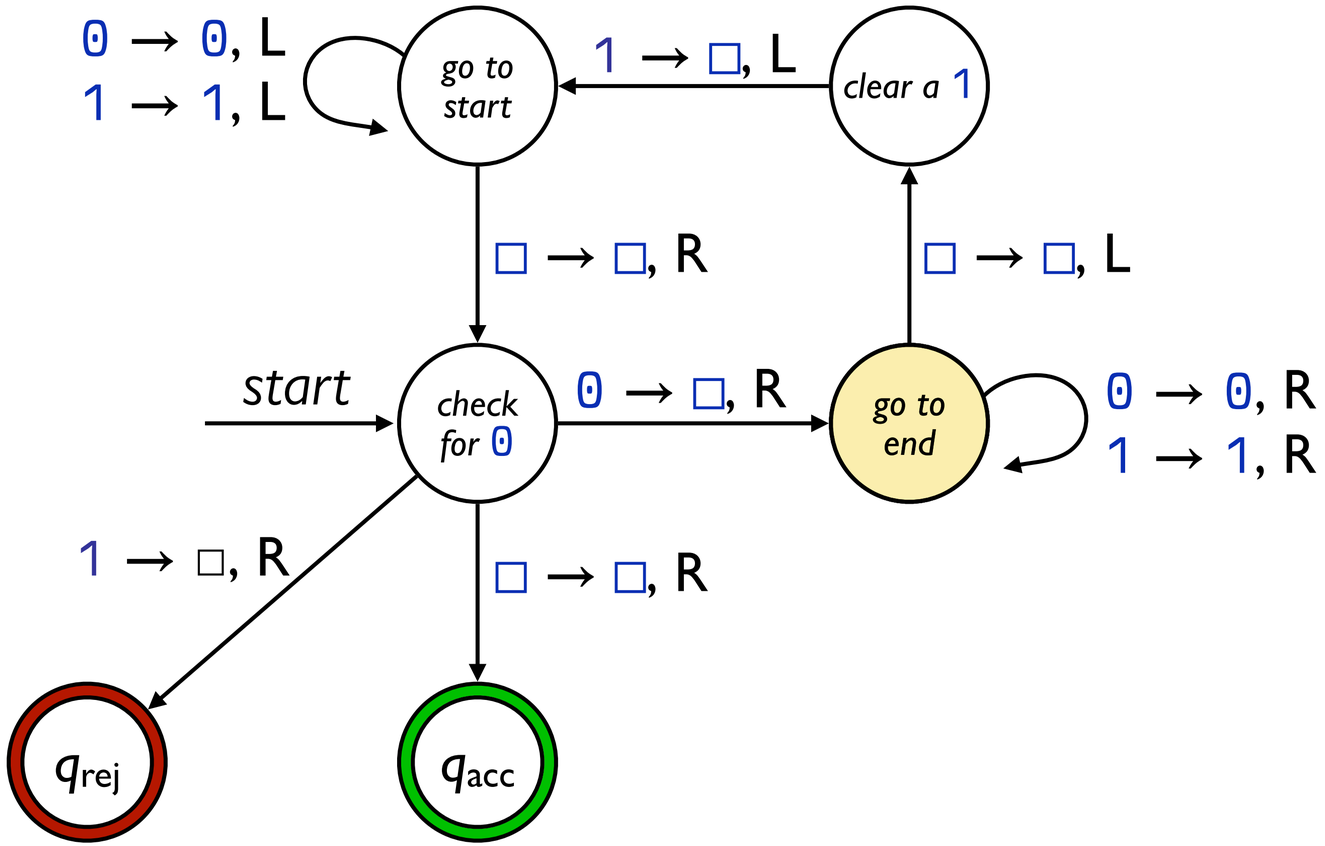

0

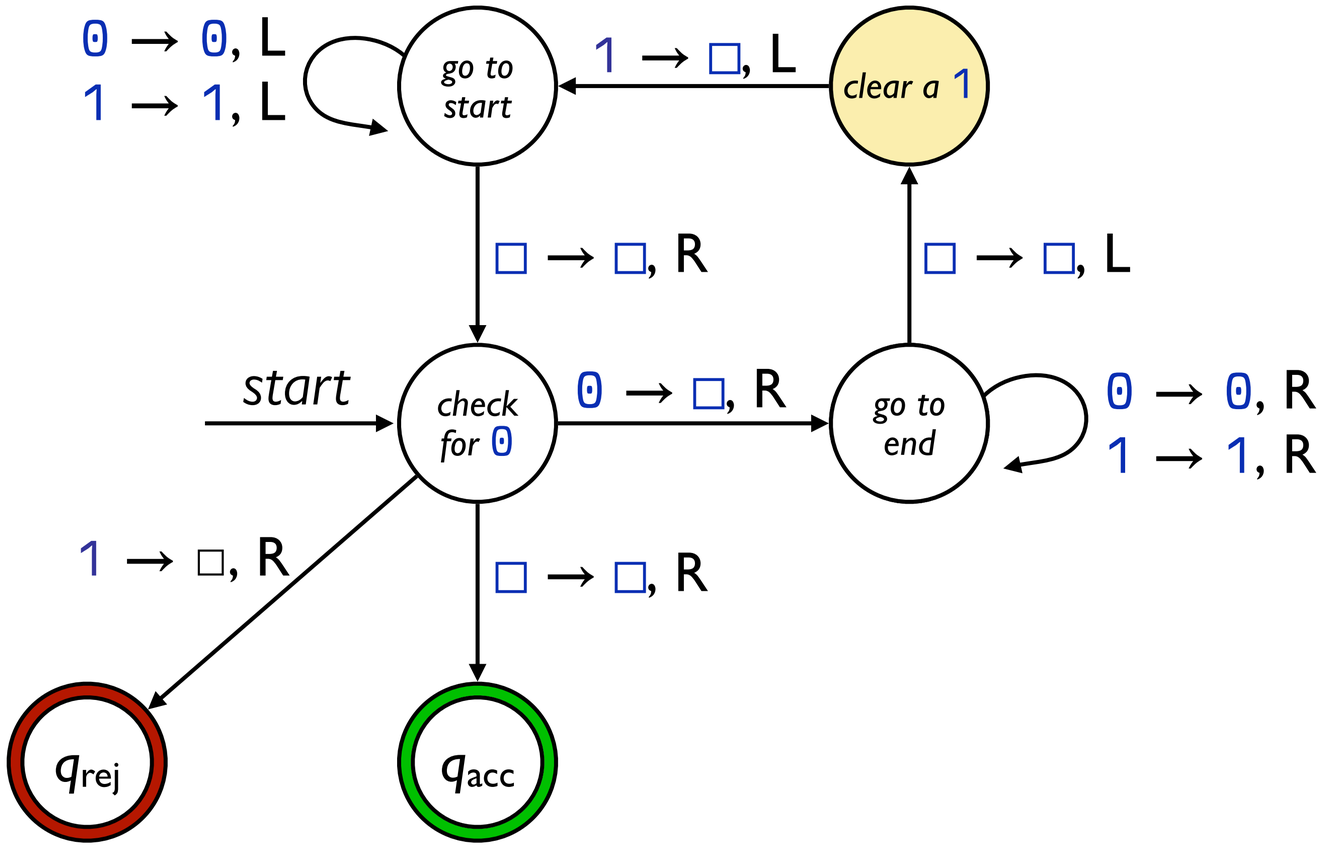

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

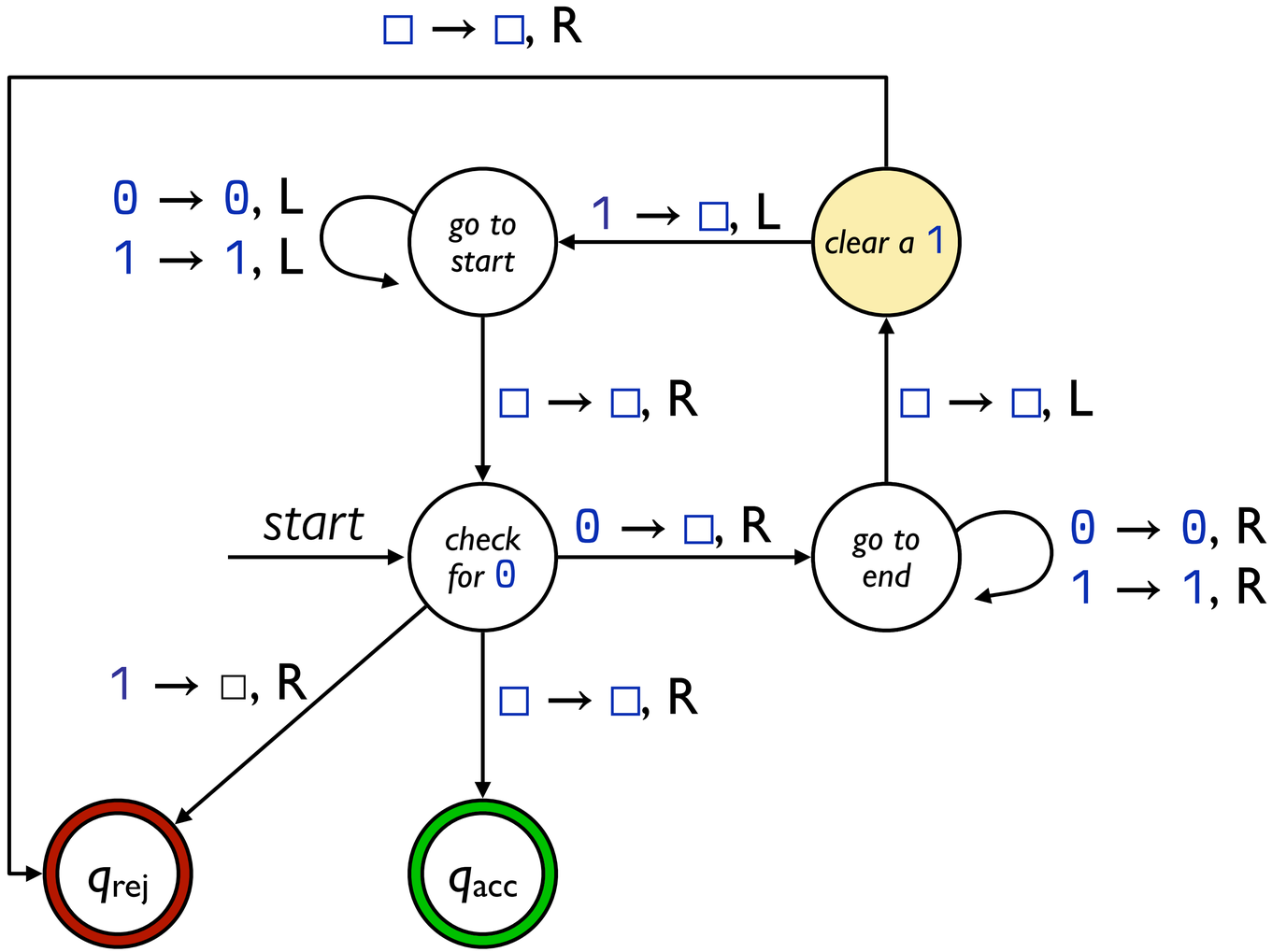

0

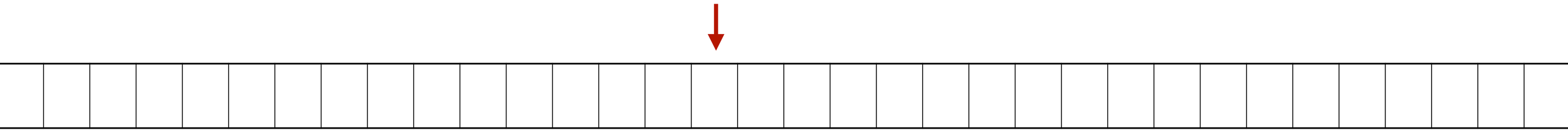

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

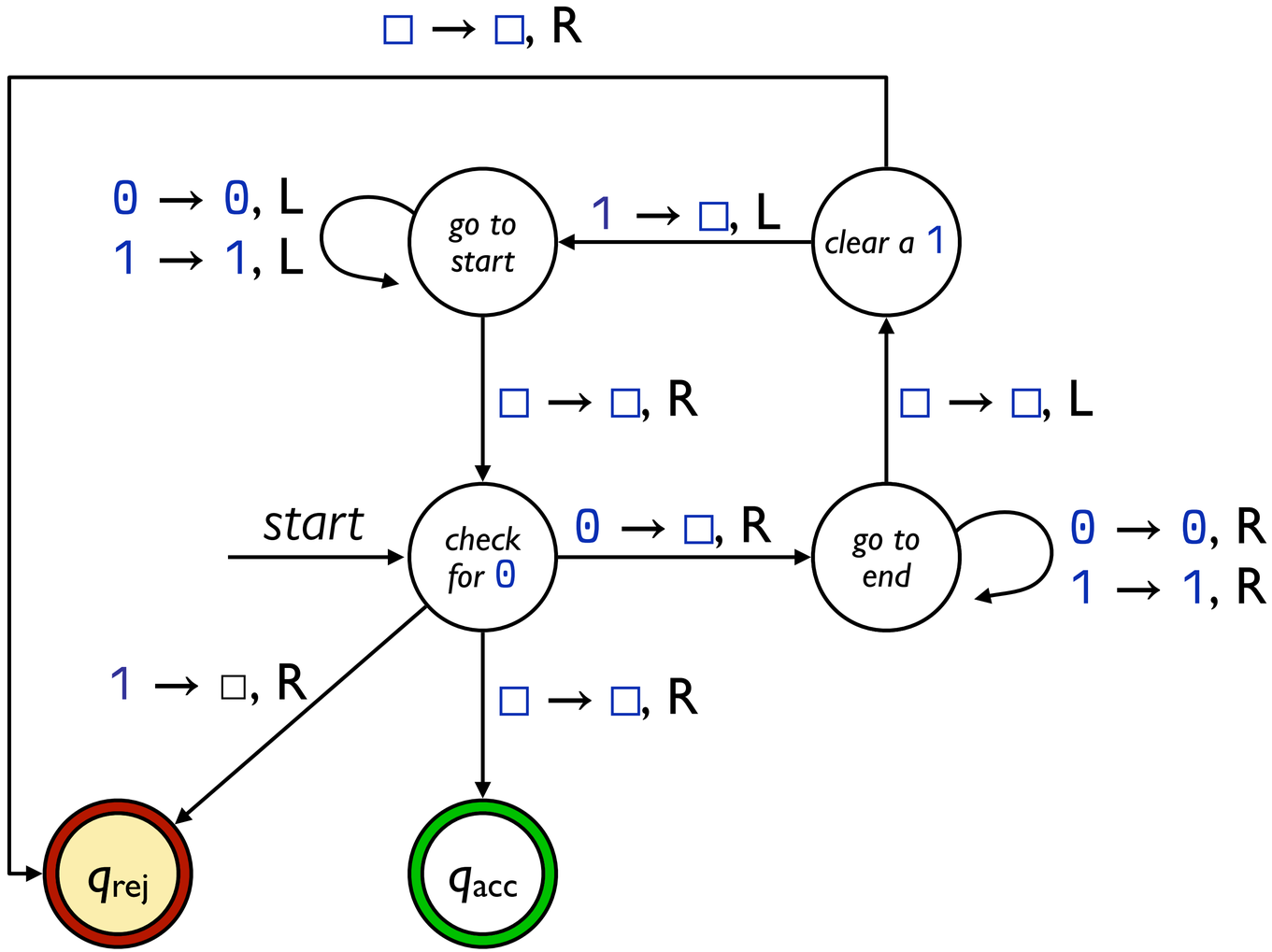

0

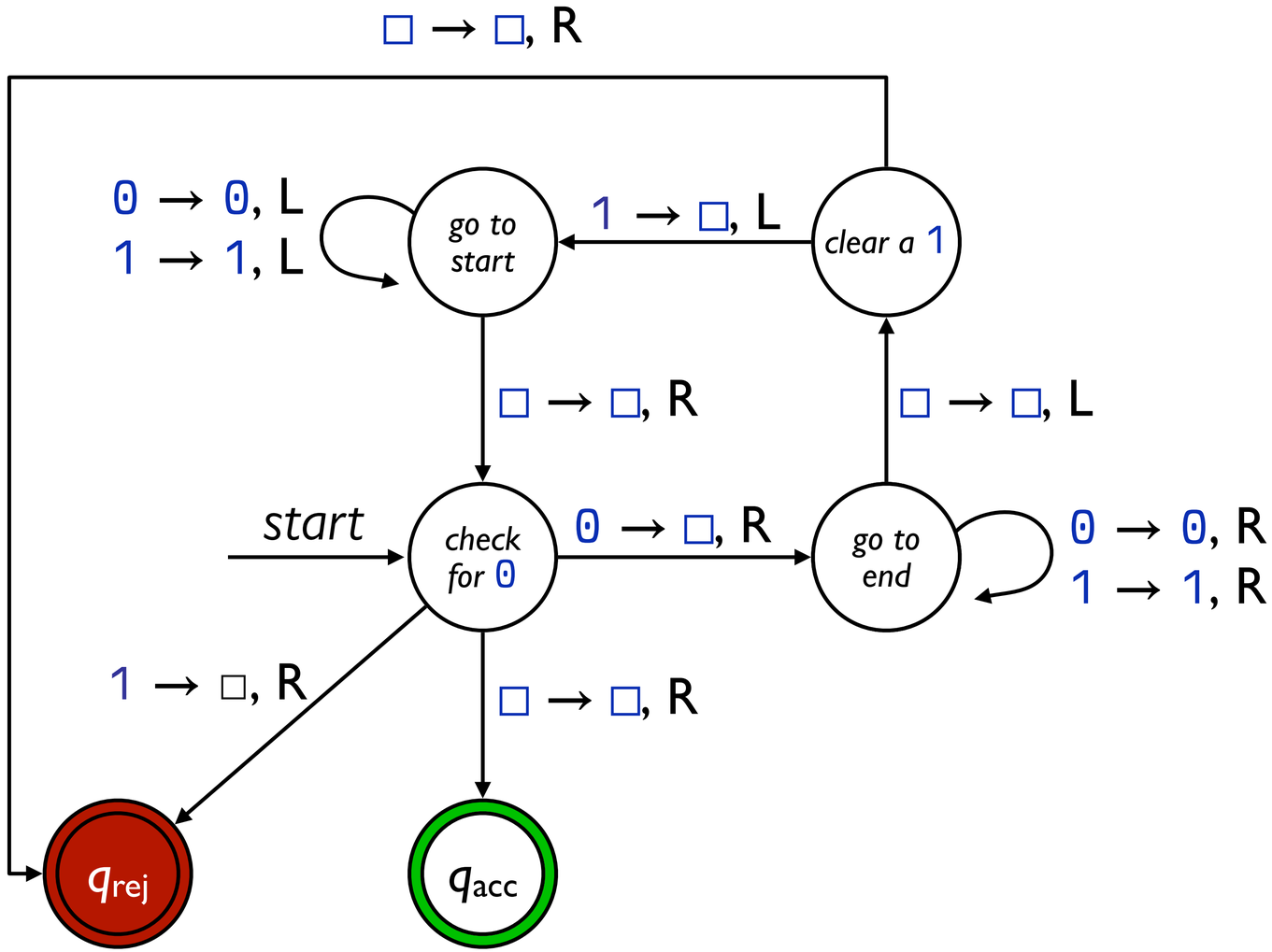

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

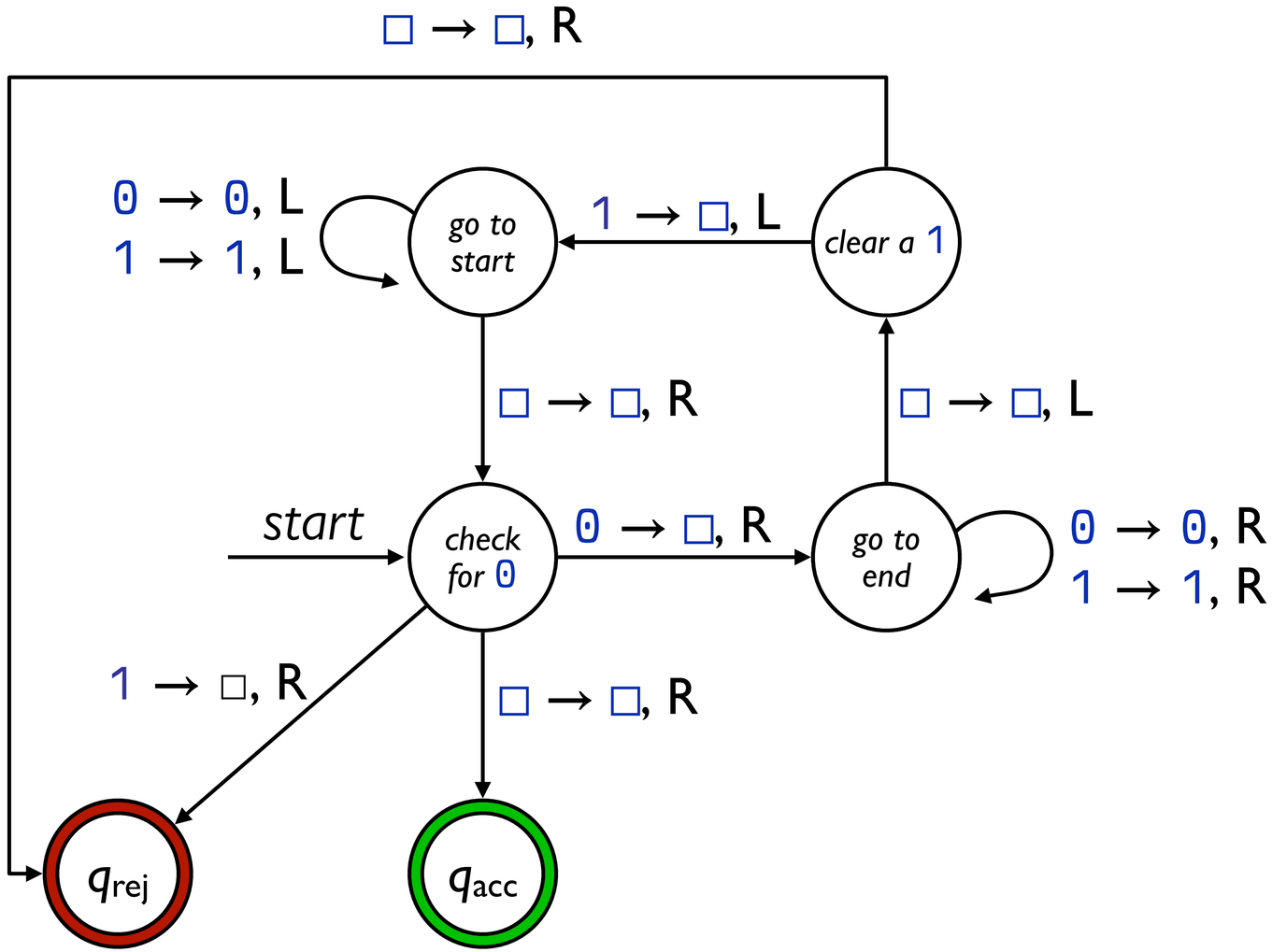

0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_{rej}
q_{acc}

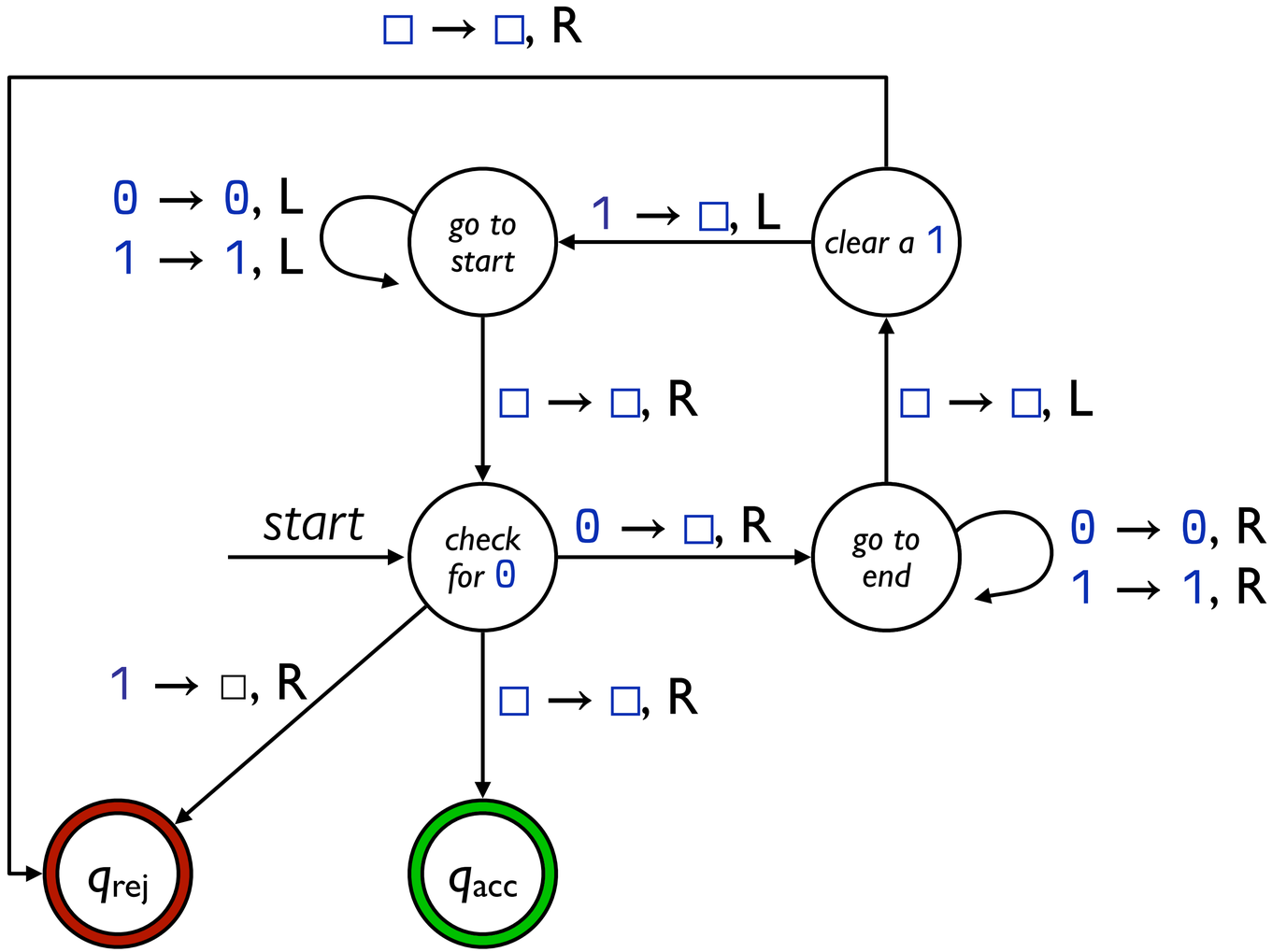

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

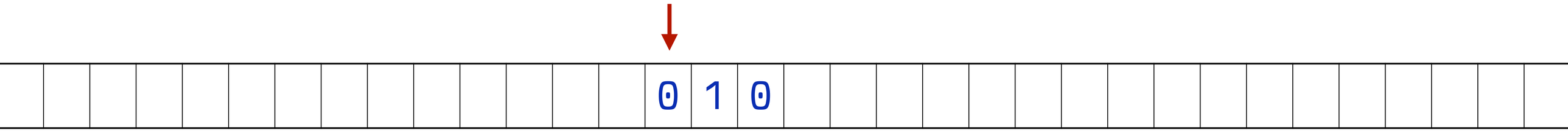


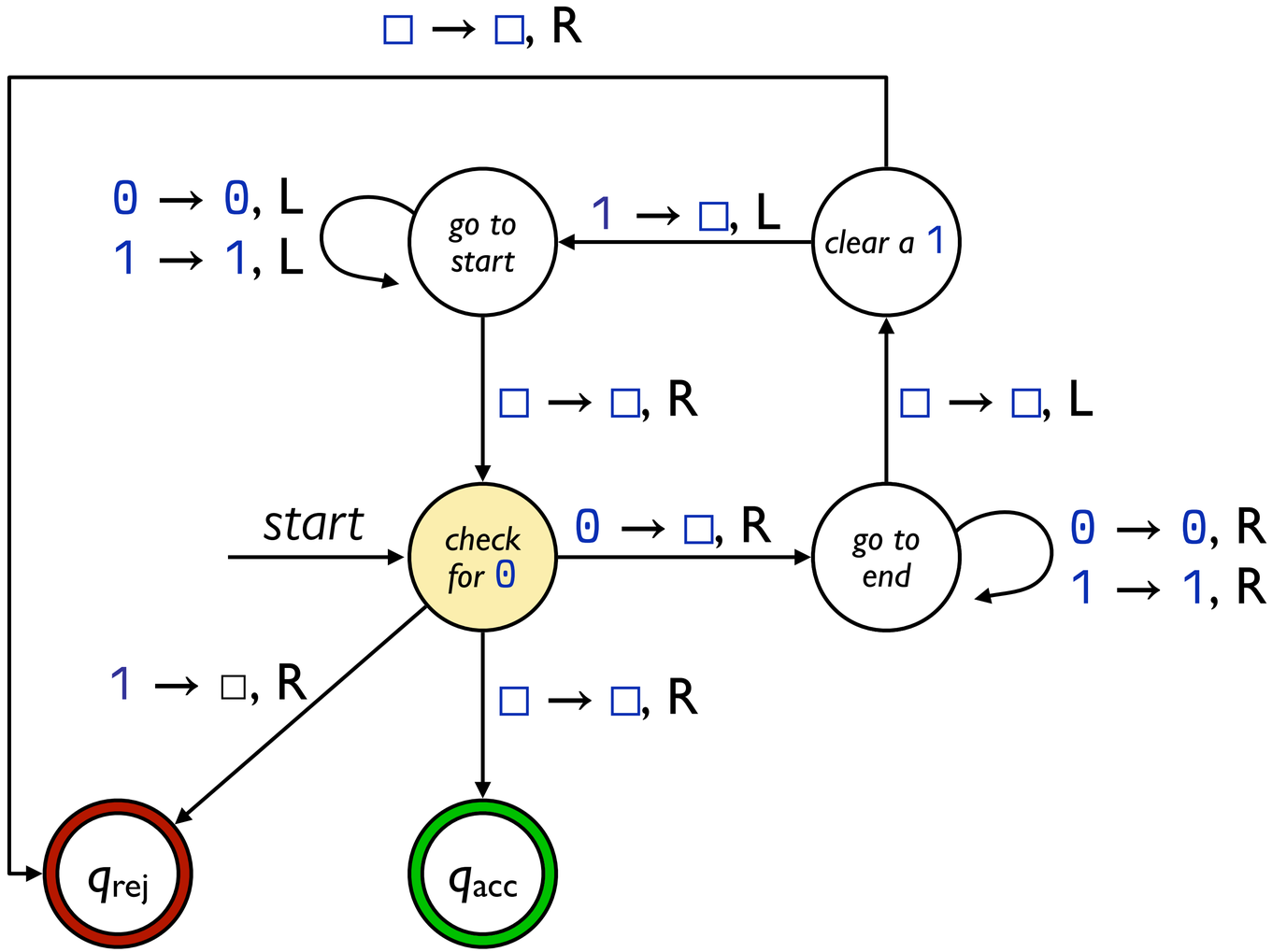

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

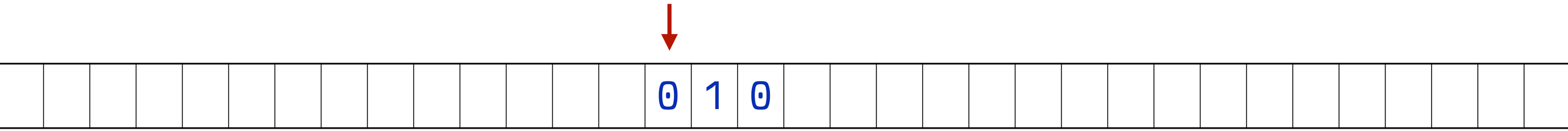

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

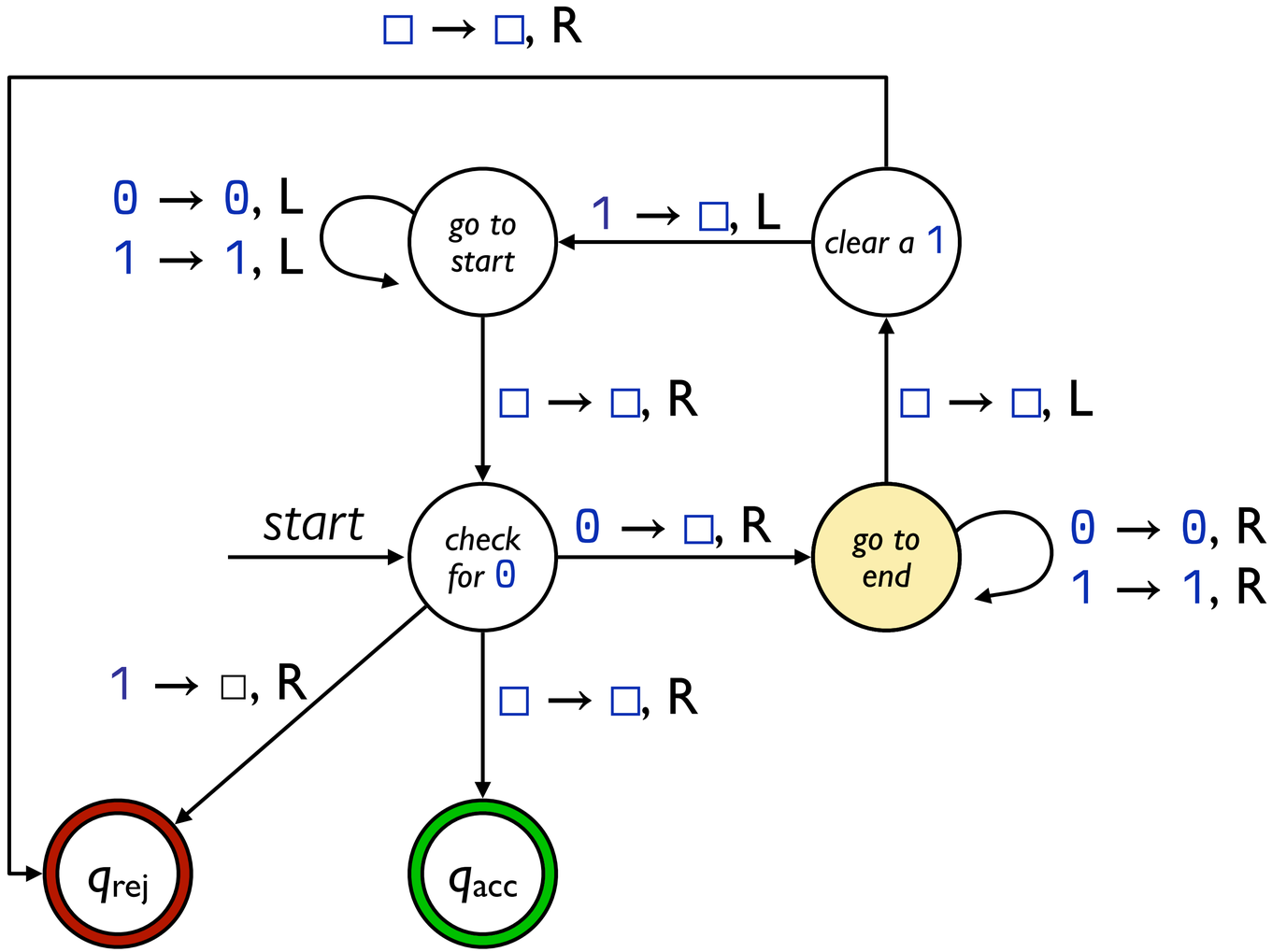

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

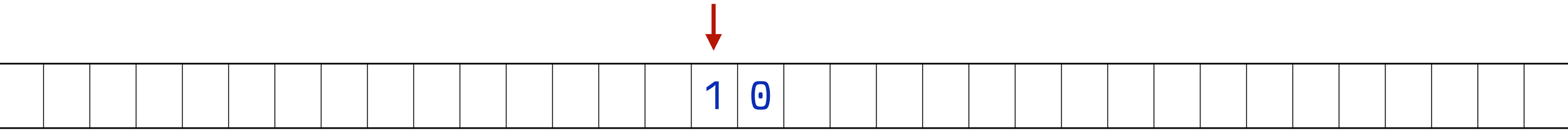

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

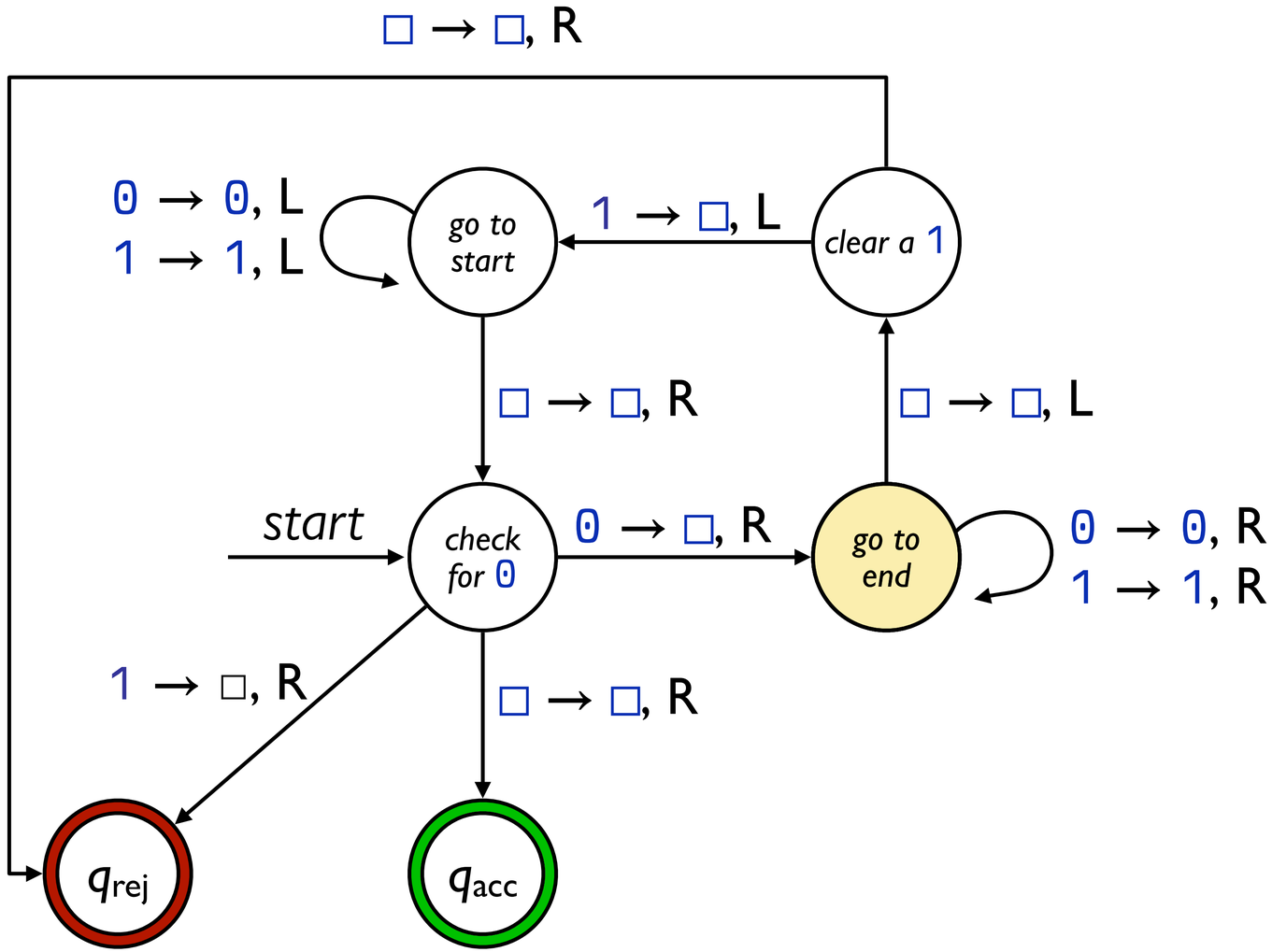

0 1 0

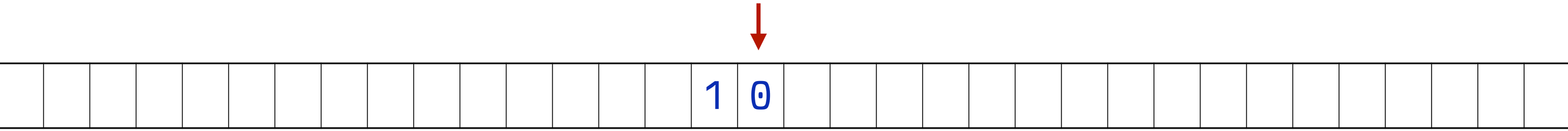

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_rej
q_acc

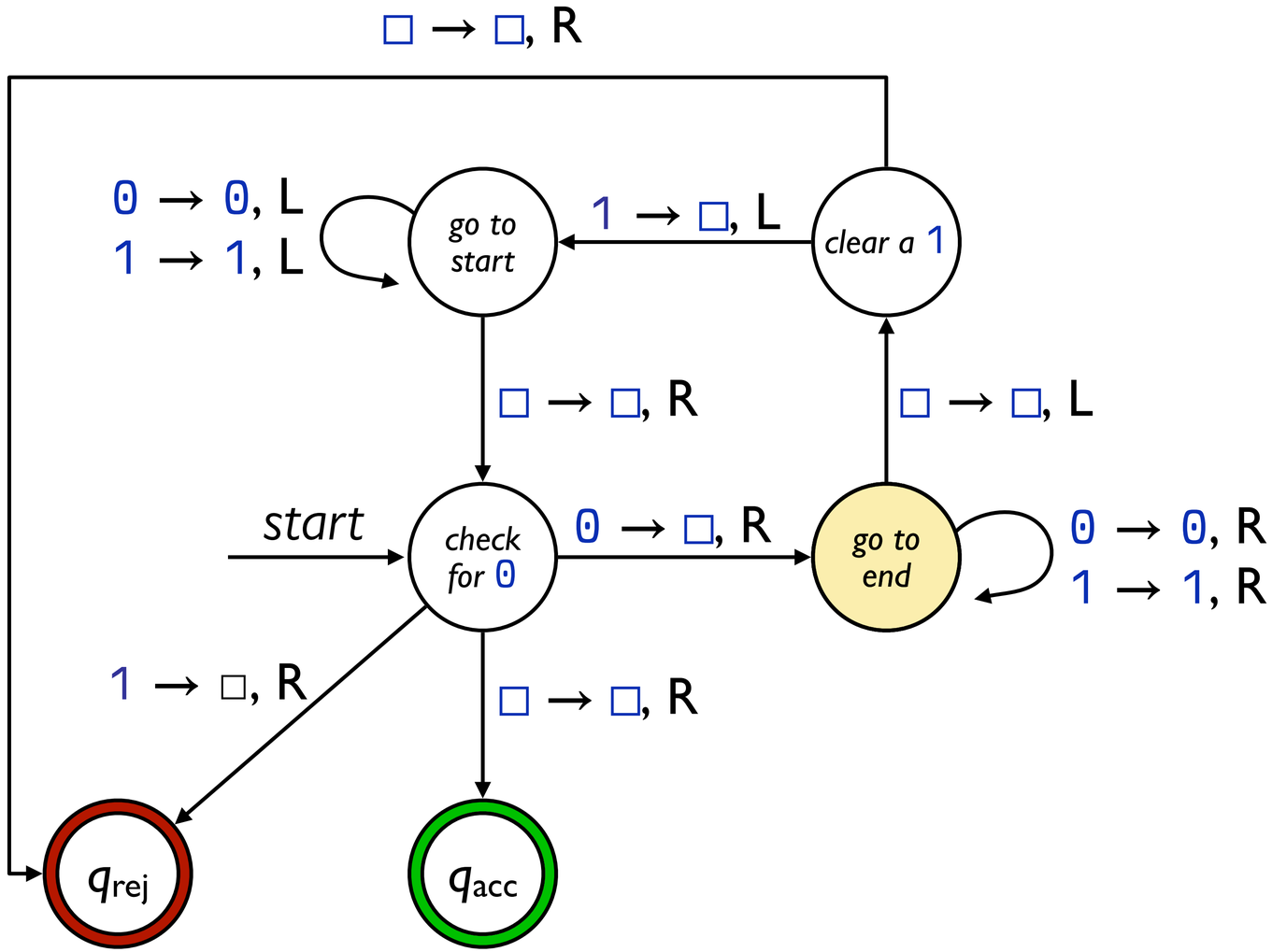

0 1 0

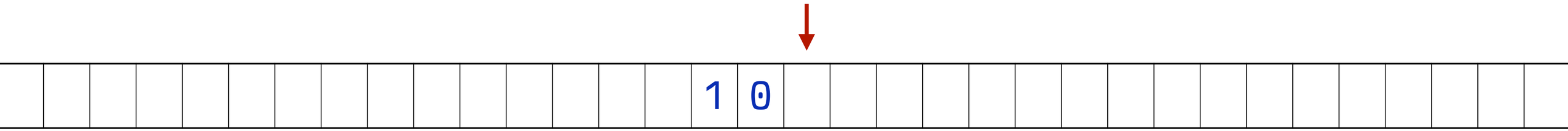

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_rej
q_acc

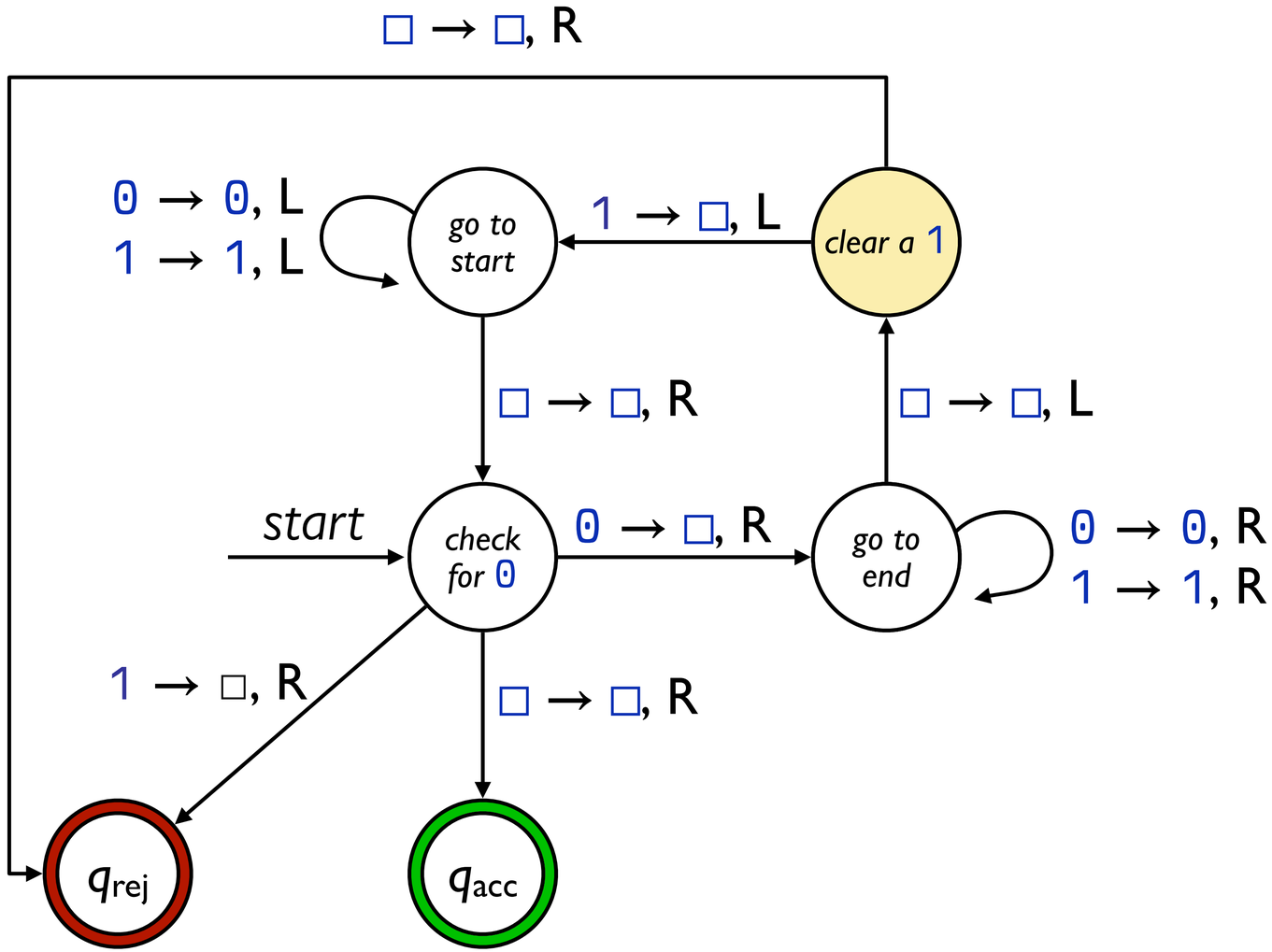

1 0

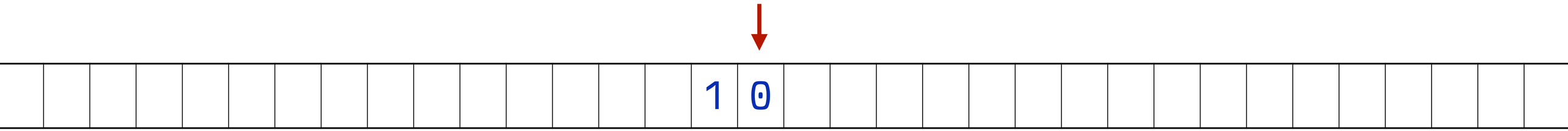

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_{rej}
q_{acc}

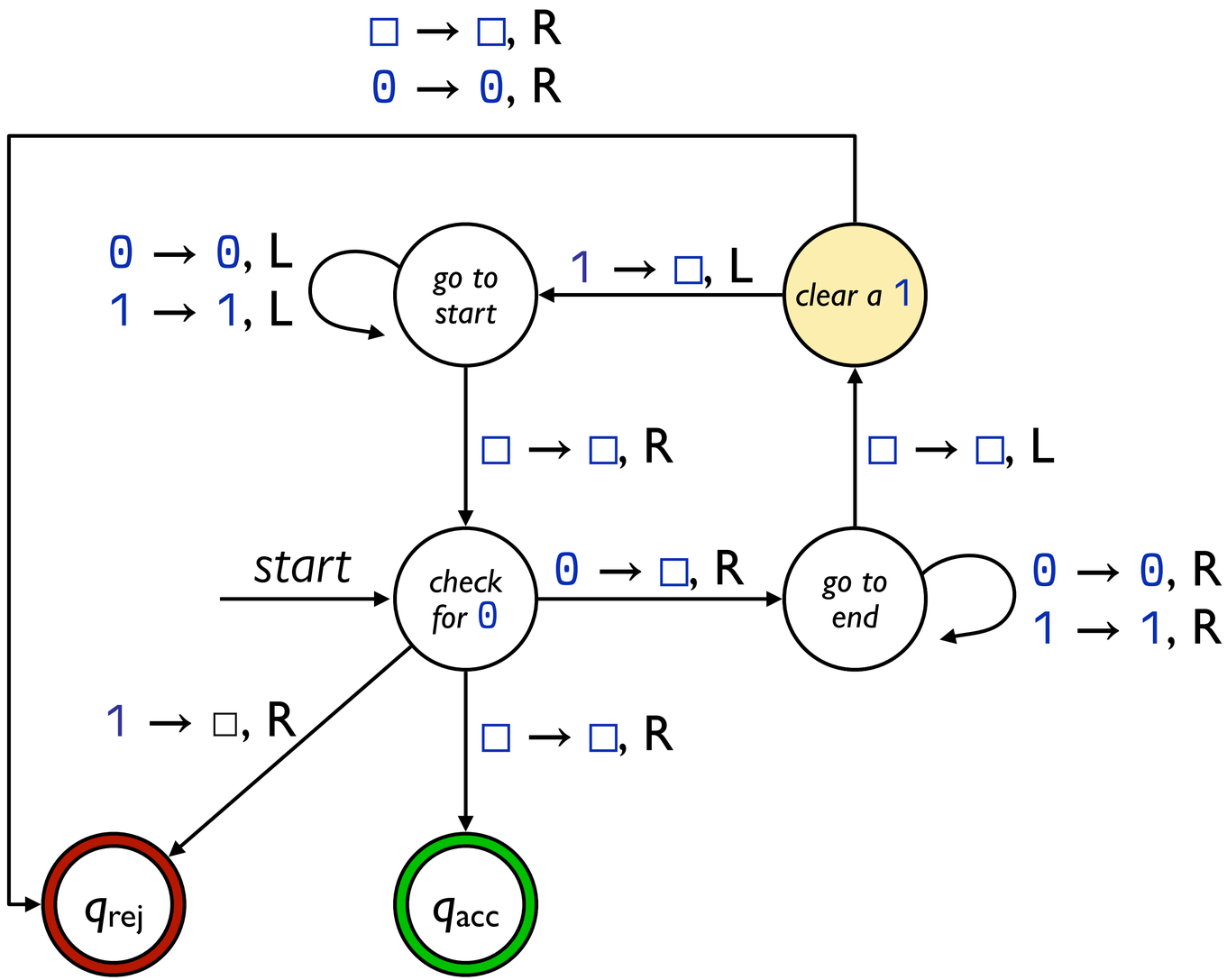

1 0

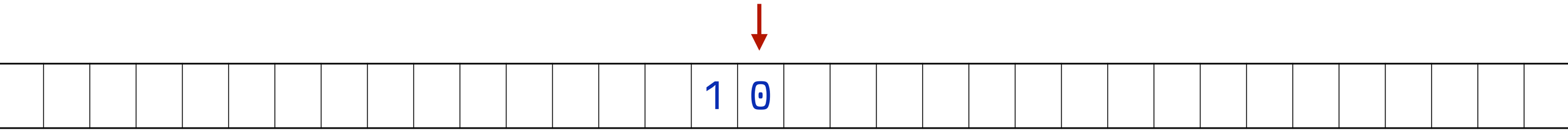

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_{rej}
q_{acc}

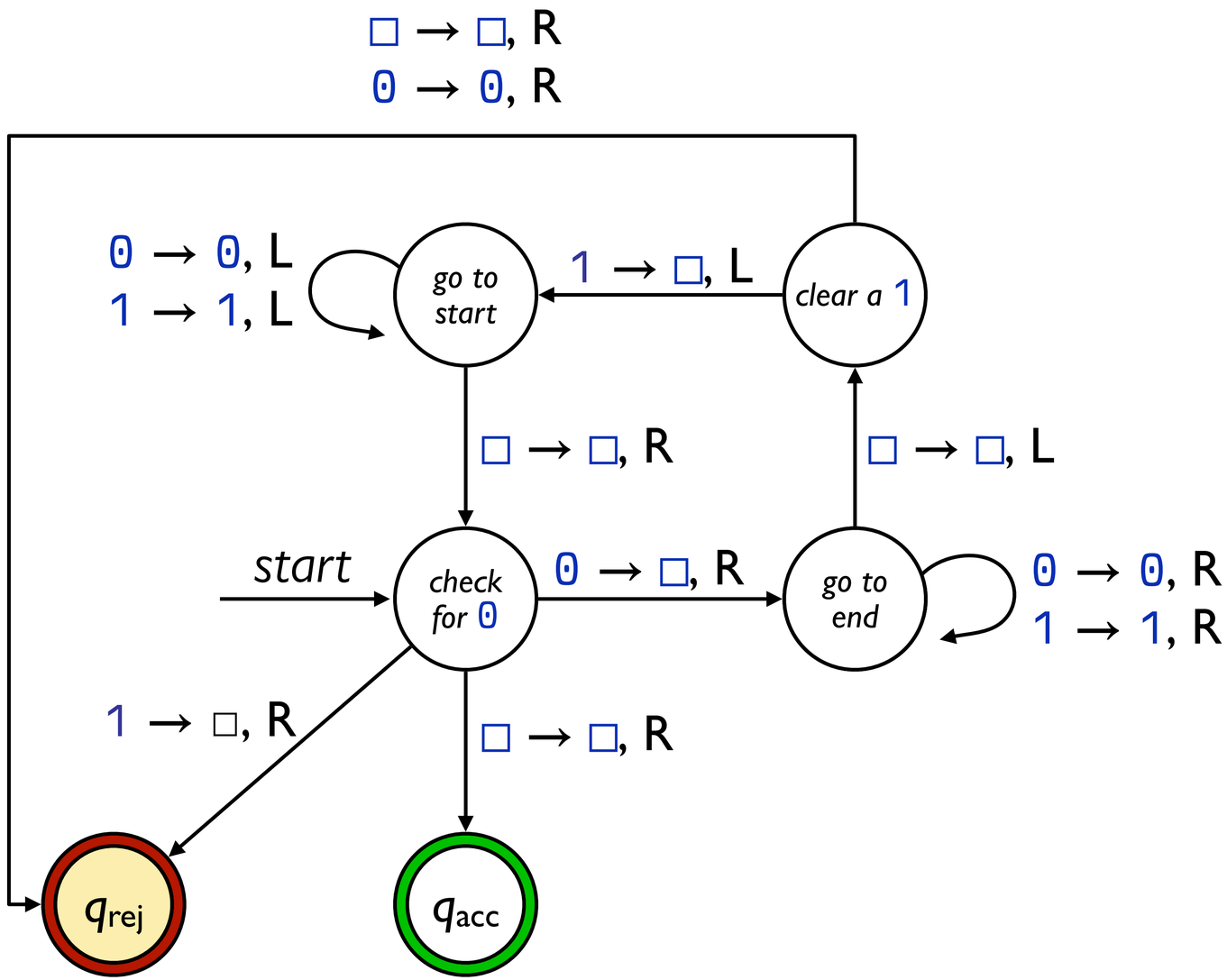

1 0

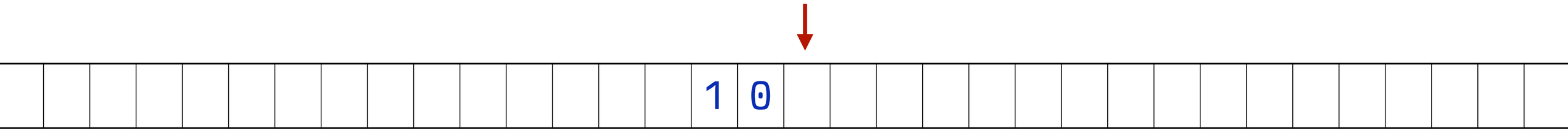

□ → □, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
q_rej
q_acc

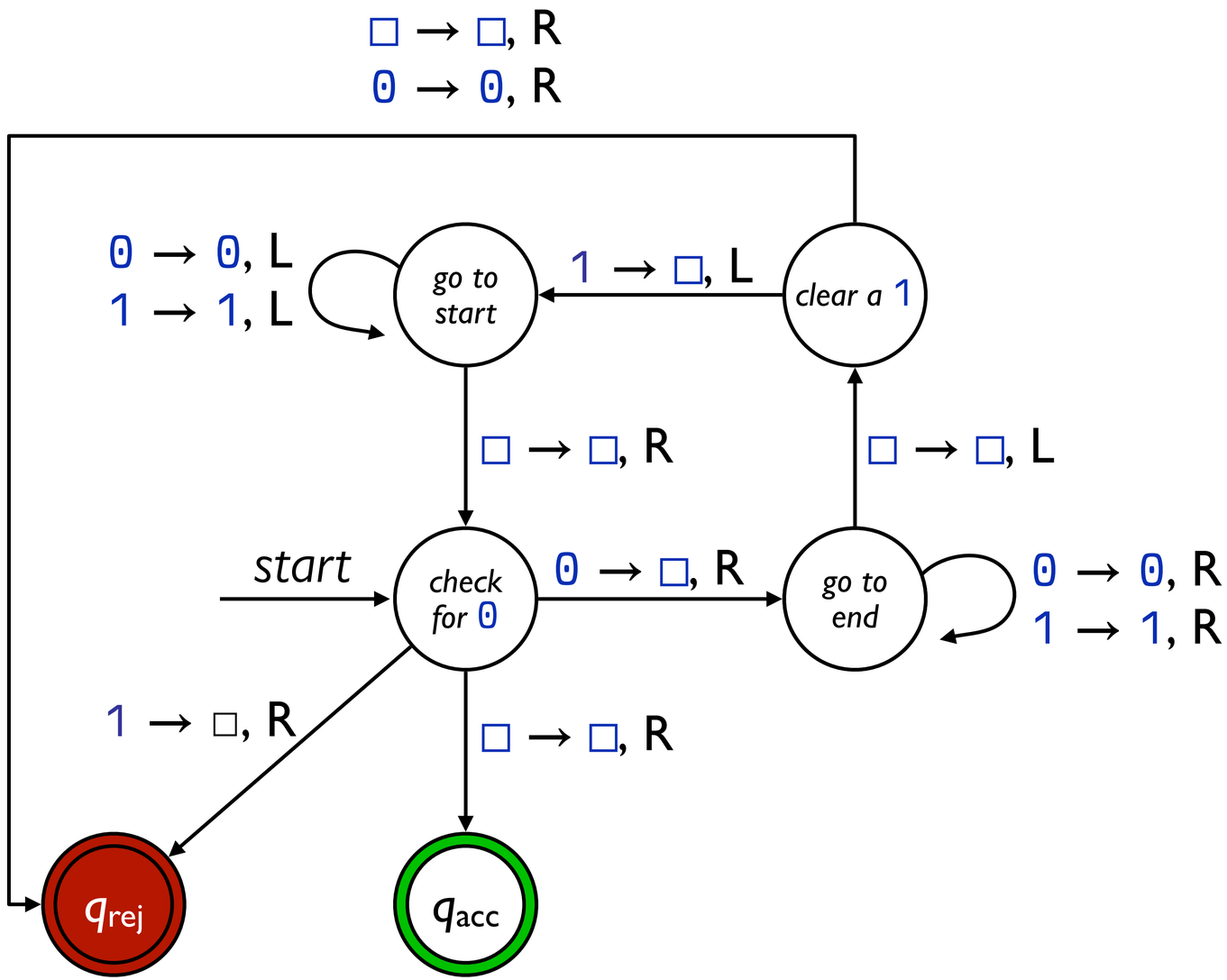

1 0

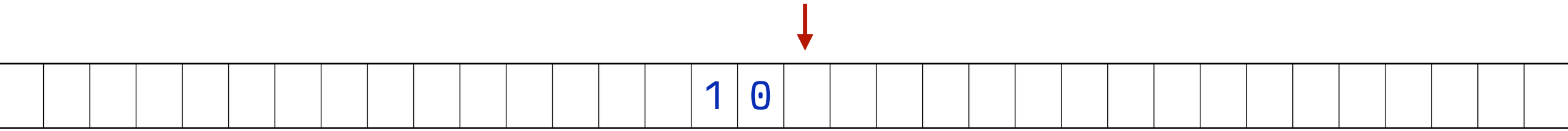

□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc

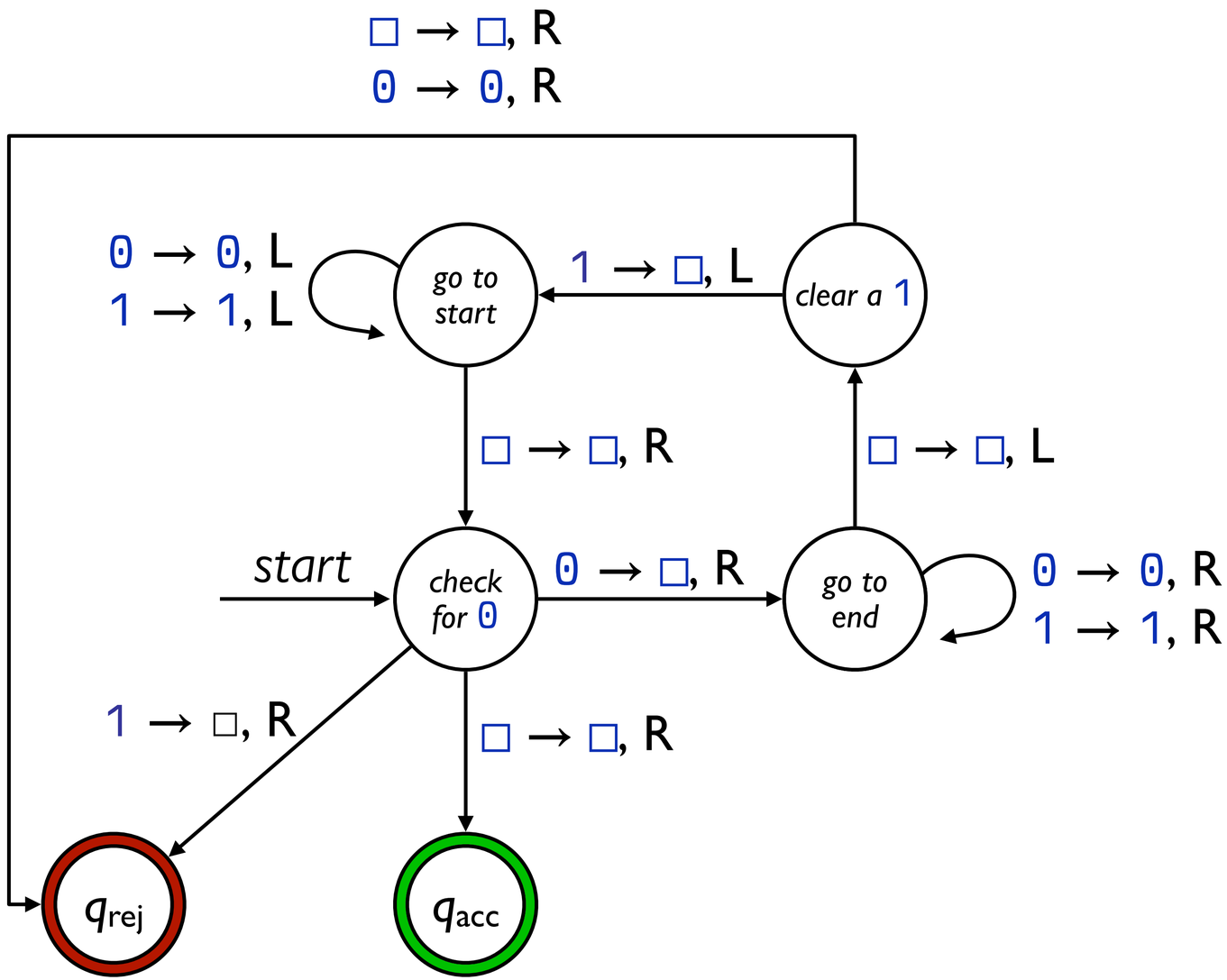

1 0

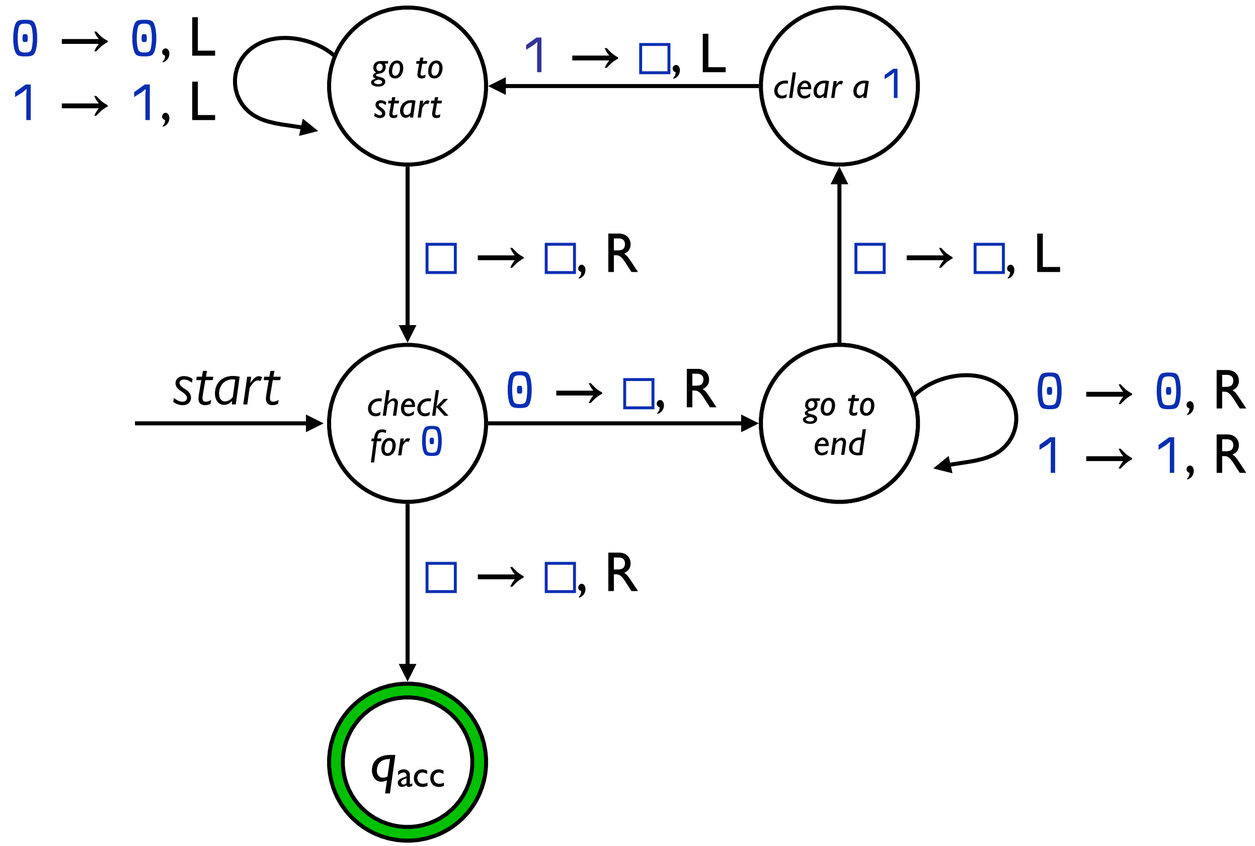

□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc


1 0


□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc


1 0


□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
go to start
1 → □, L
clear a 1
□ → □, R
□ → □, L
start
check for 0
0 → □, R
go to end
0 → 0, R
1 → 1, R
1 → □, R
□ → □, R
qrej
qacc


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

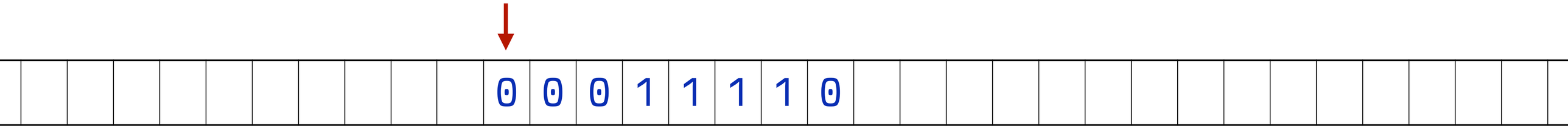
Another Turing machine design

We designed a Turing machine for $\{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}_0\}$.

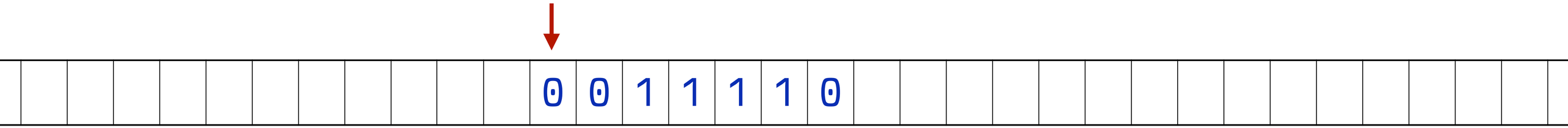
Let's consider how we could design a Turing machine for a related context-free language over $\Sigma = \{\texttt{0}, \texttt{1}\}$:

$$L = \{w \in \Sigma^* \mid n_{\texttt{0}}(w) = n_{\texttt{1}}(\text{w})\}$$

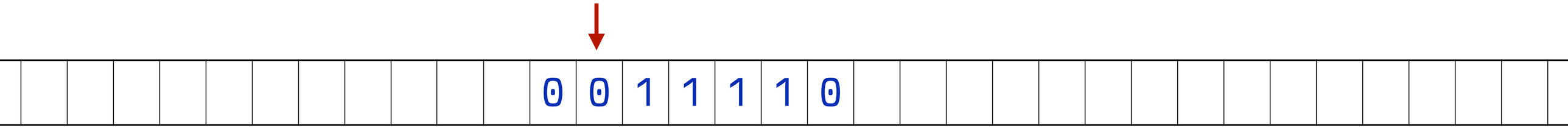
A caveat



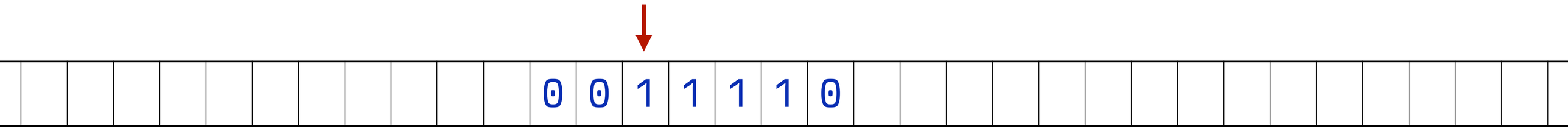
A caveat



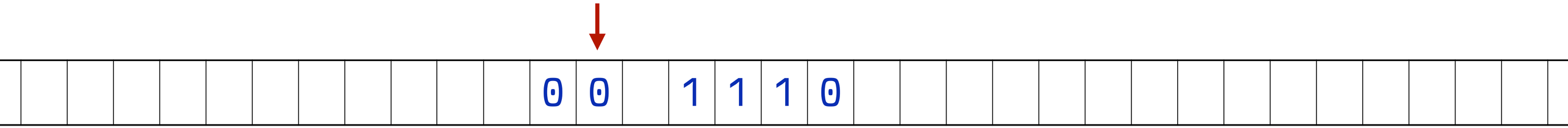
A caveat



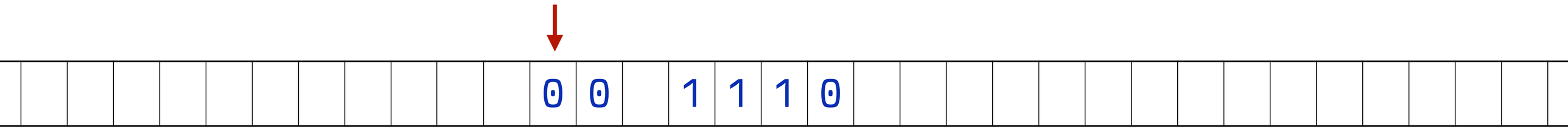
A caveat



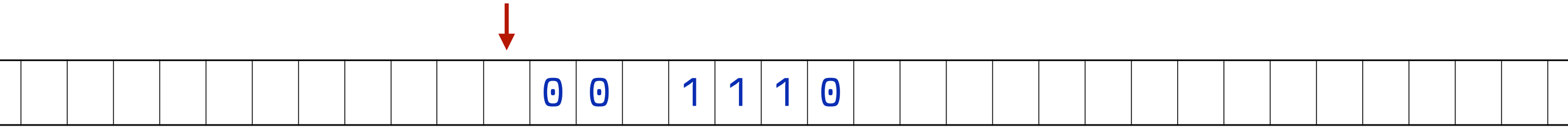
A caveat



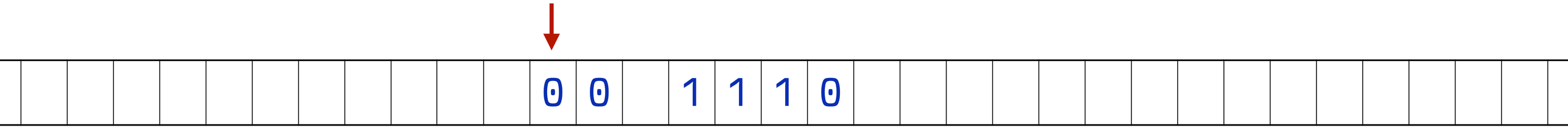
A caveat



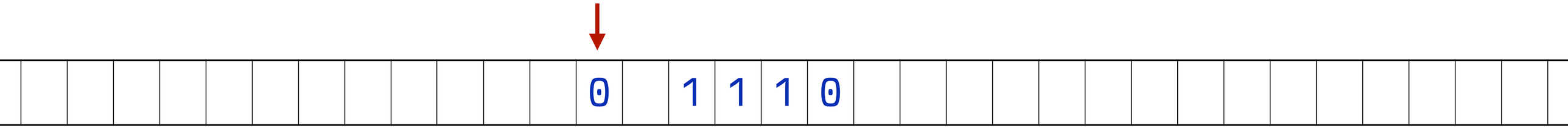
A caveat



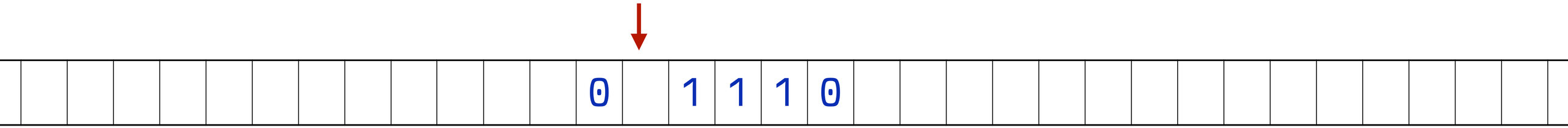
A caveat



A caveat

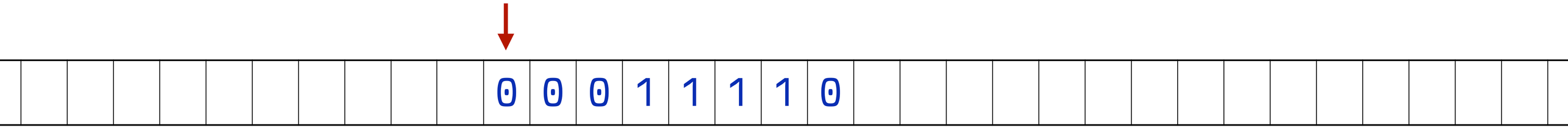


A caveat

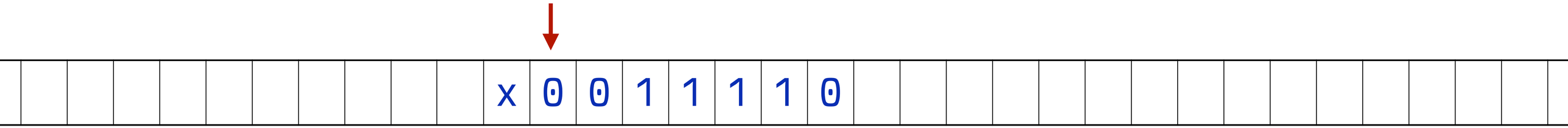


How do we know that this blank isn't one of the infinitely many blanks after our input string?

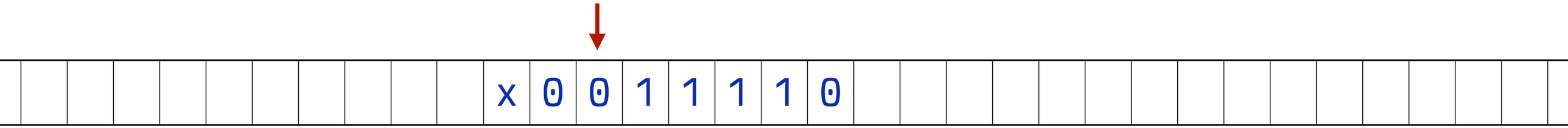
One solution



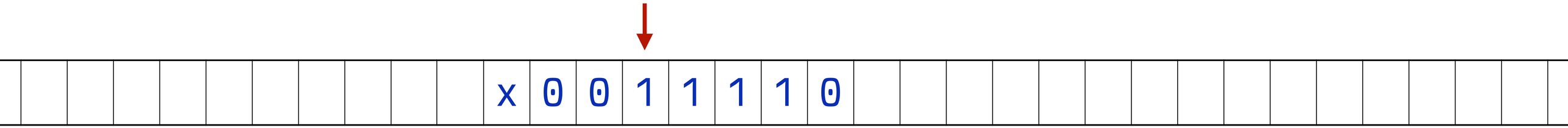
One solution



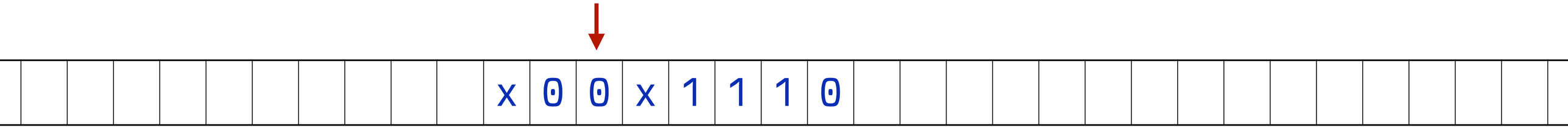
One solution



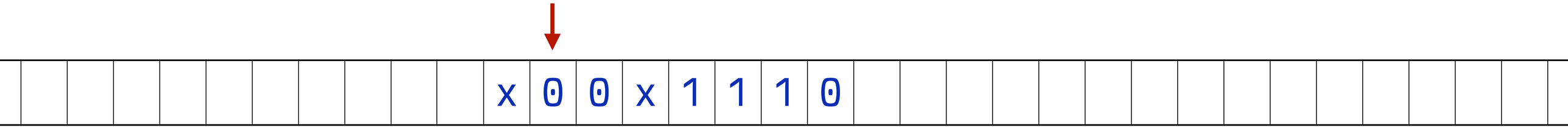
One solution



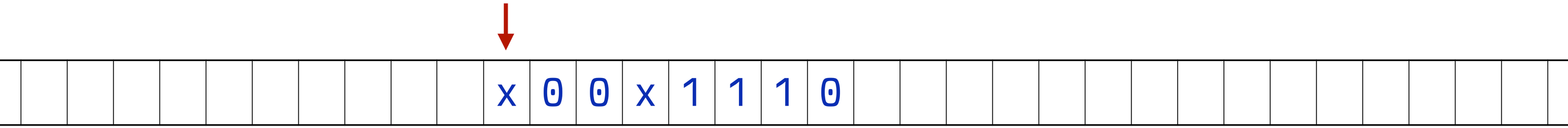
One solution



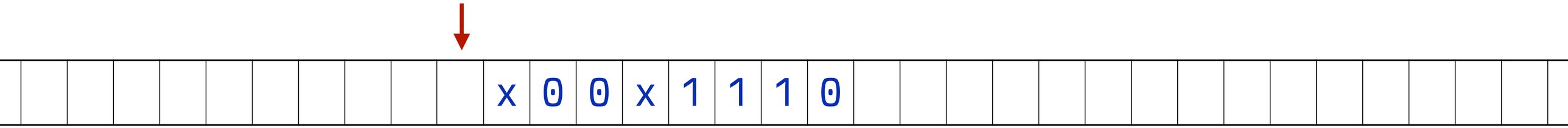
One solution



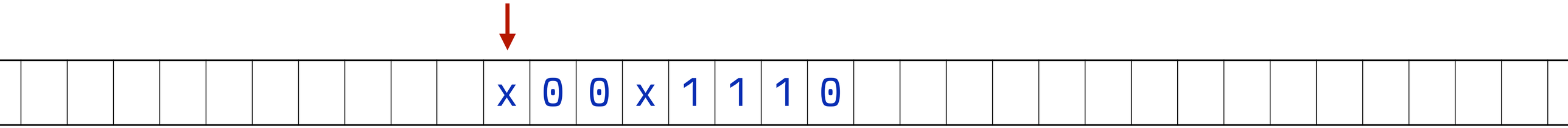
One solution



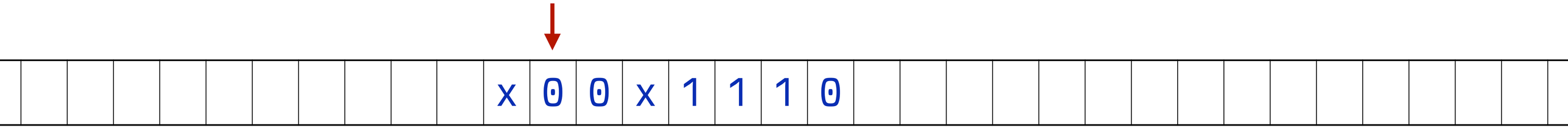
One solution



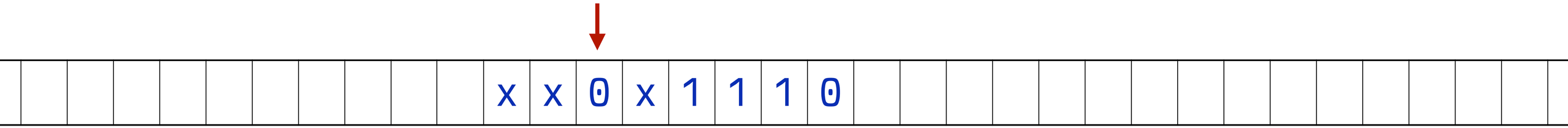
One solution



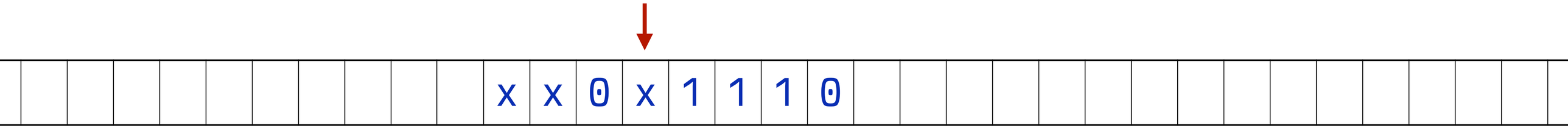
One solution



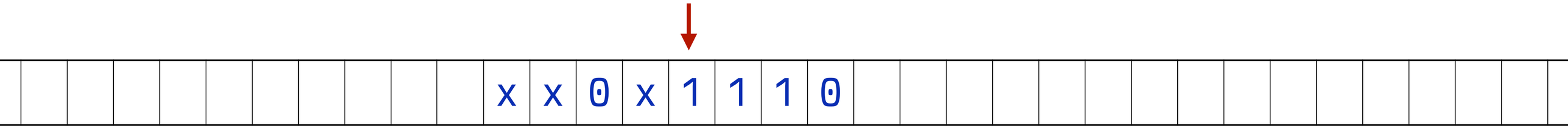
One solution



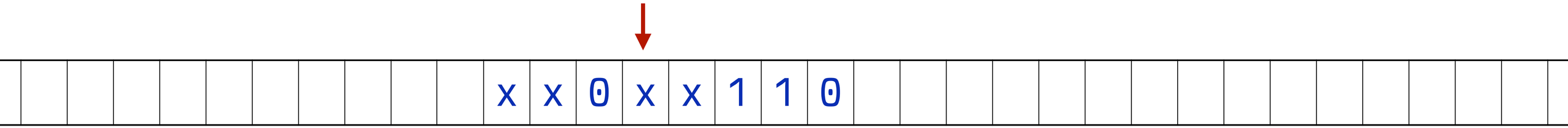
One solution



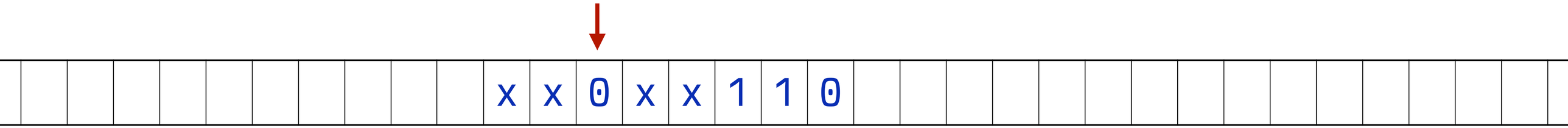
One solution



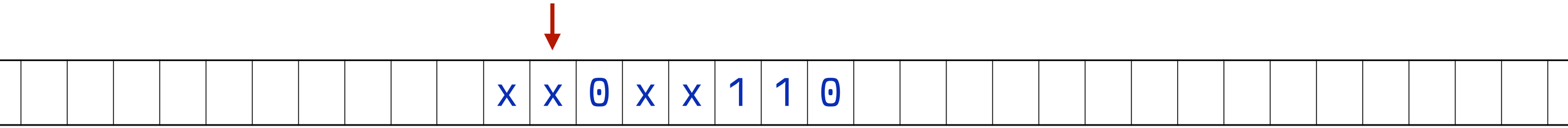
One solution



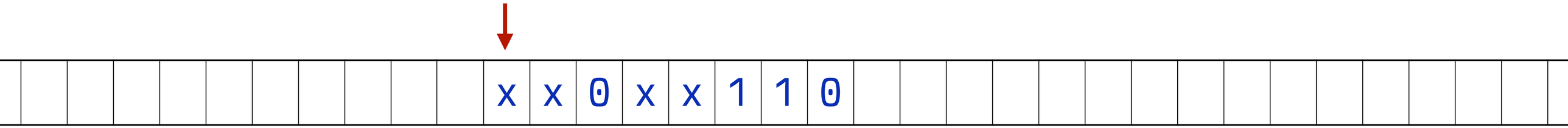
One solution



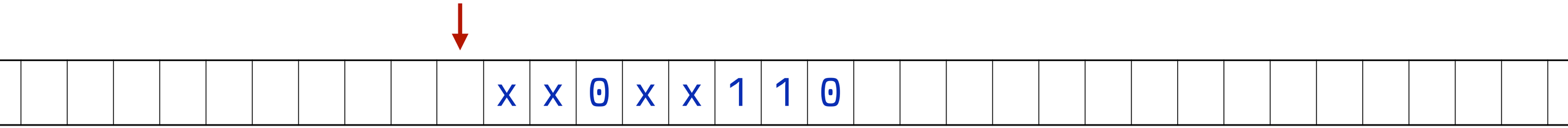
One solution



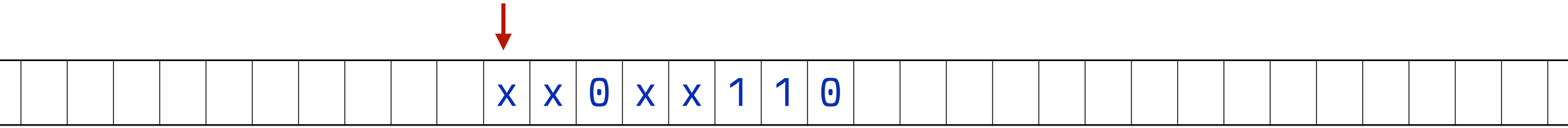
One solution



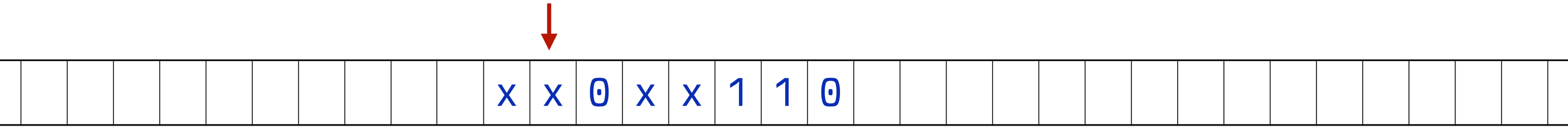
One solution



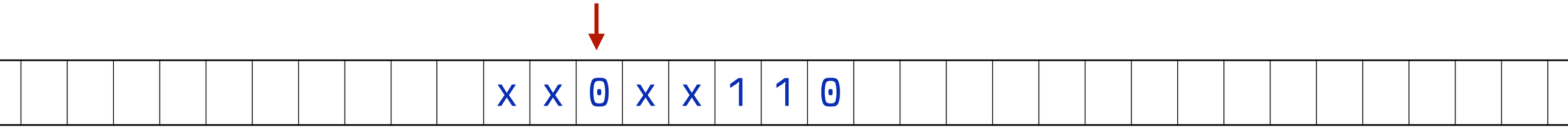
One solution



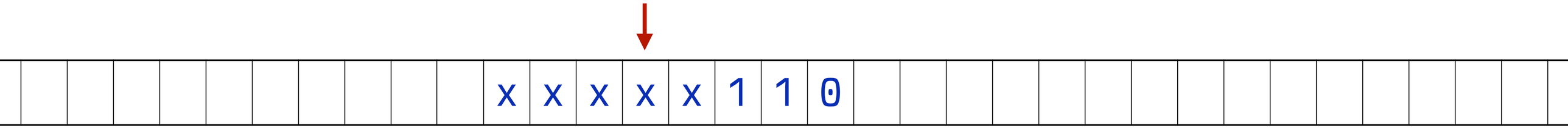
One solution



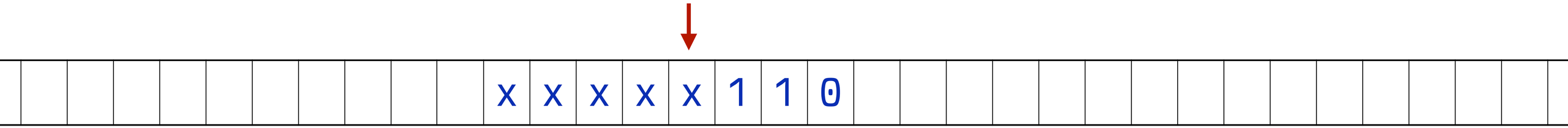
One solution



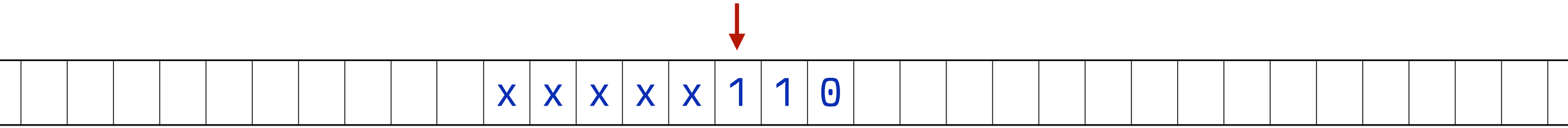
One solution



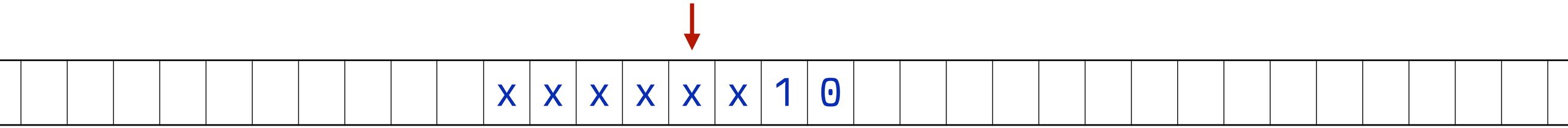
One solution



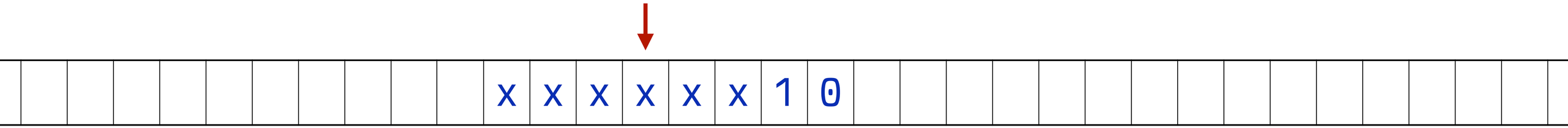
One solution



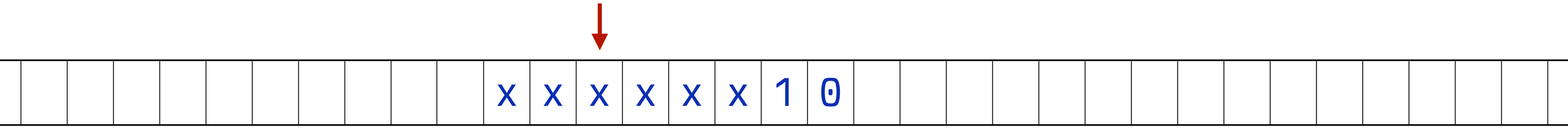
One solution



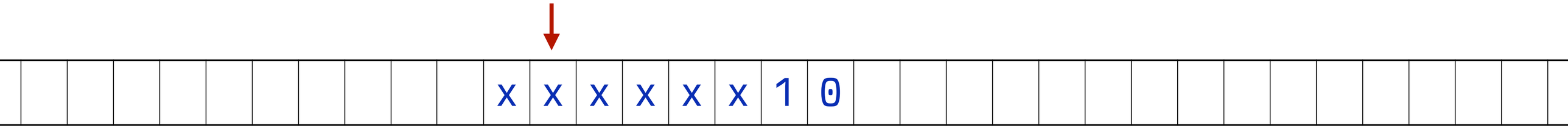
One solution



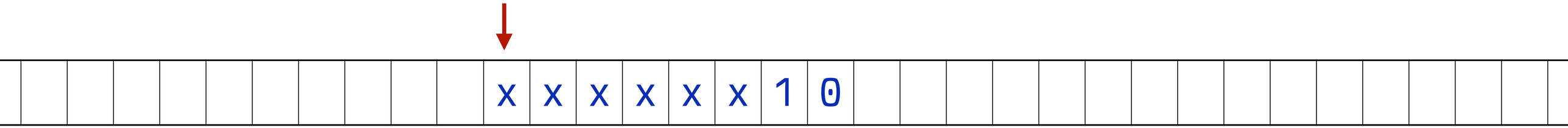
One solution



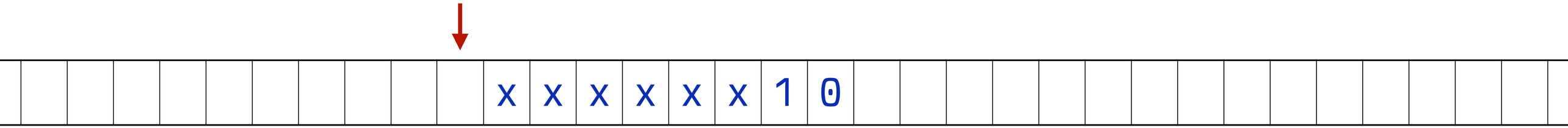
One solution



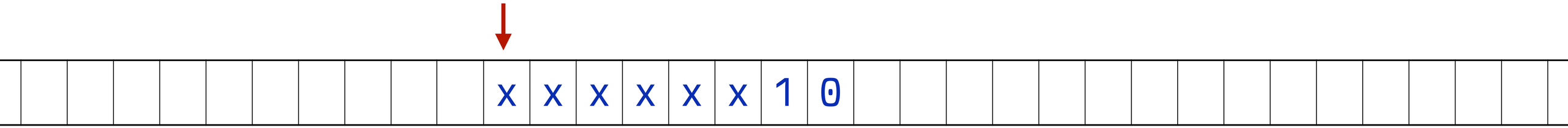
One solution



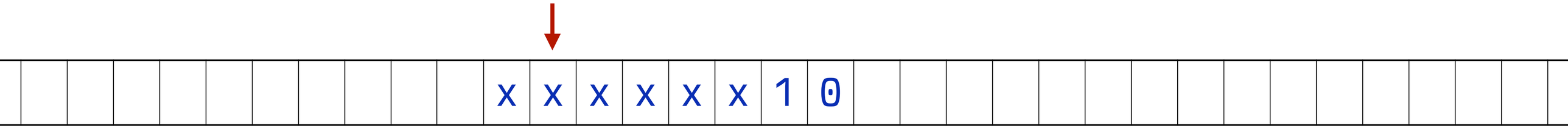
One solution



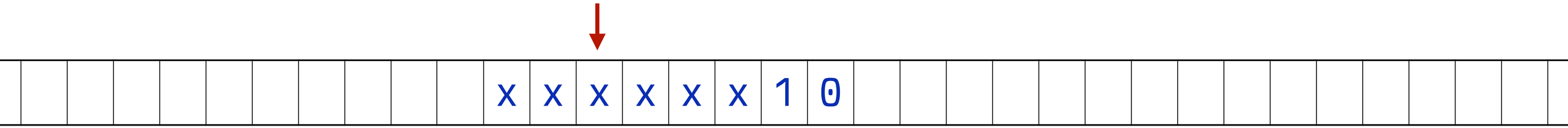
One solution



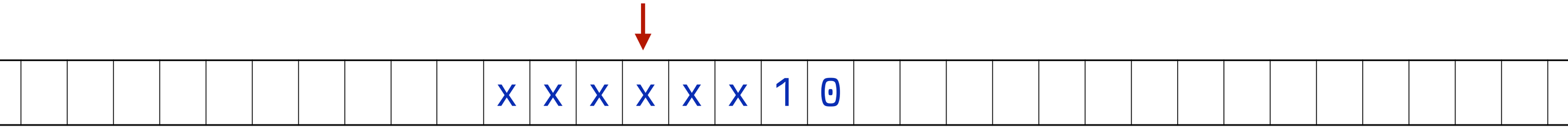
One solution



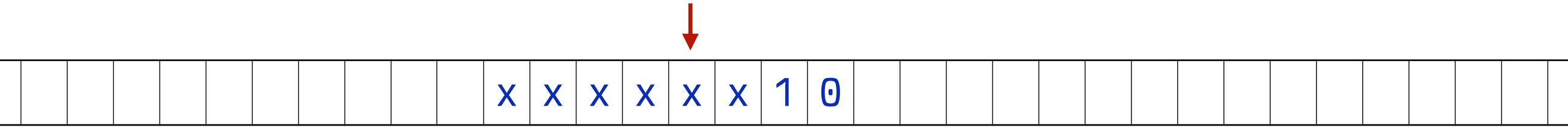
One solution



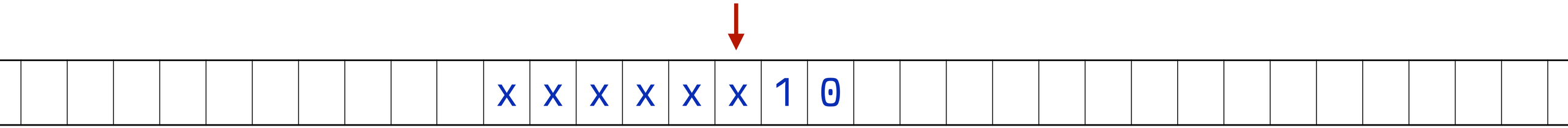
One solution



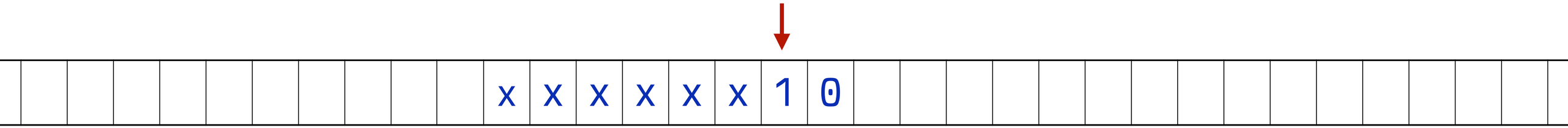
One solution



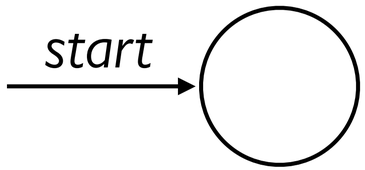
One solution

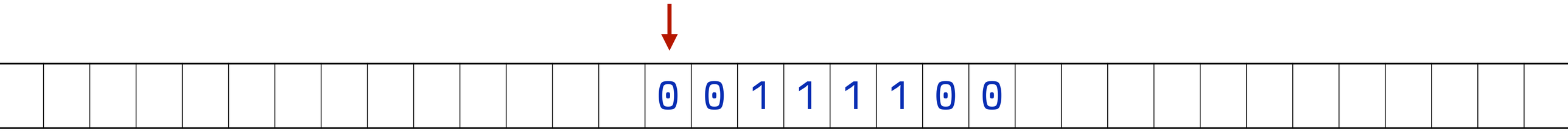


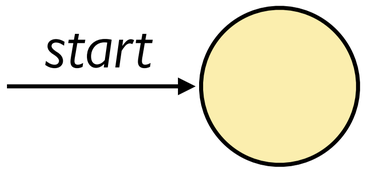
One solution

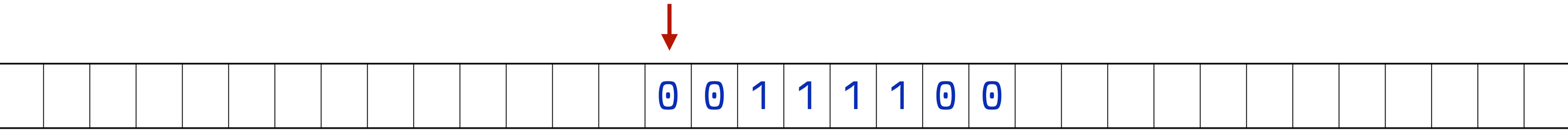


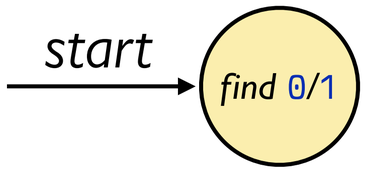
Now the first non-x character we encounter is a 1, so we want to cross it off but remember that we're looking for a matching 0

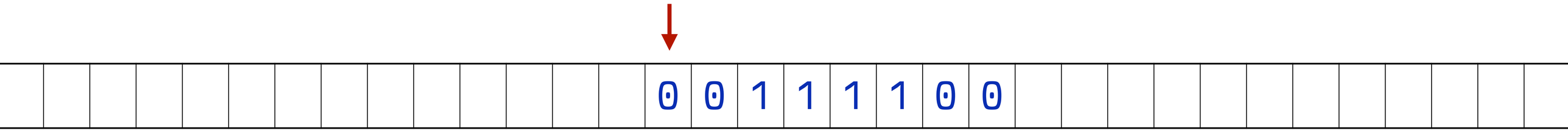

start

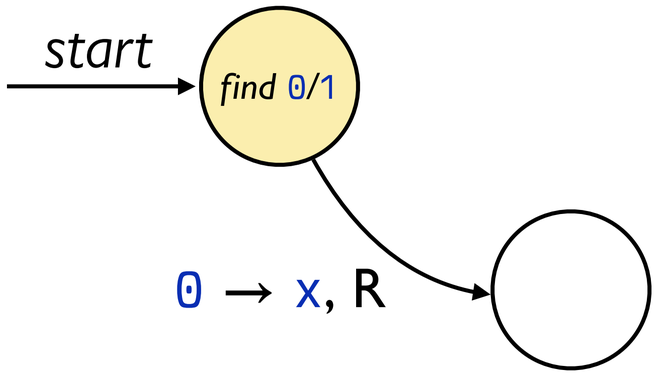

0 0 1 1 1 1 0 0

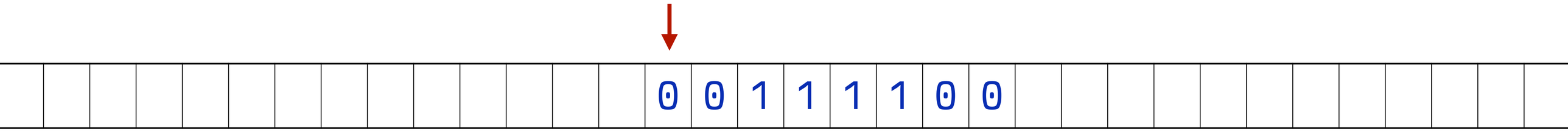

start

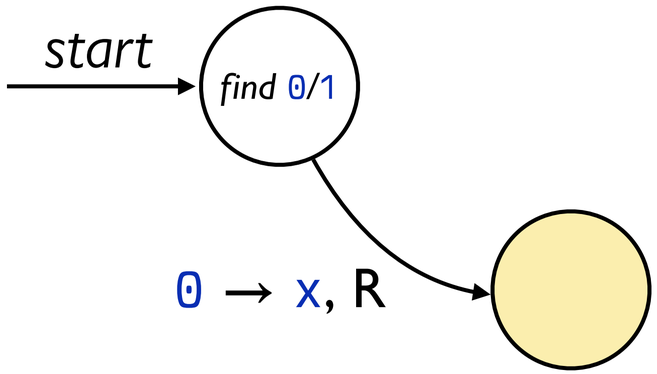

0 0 1 1 1 1 0 0

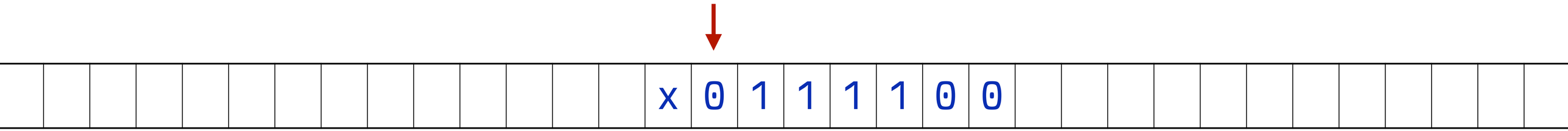

start
find 0/1

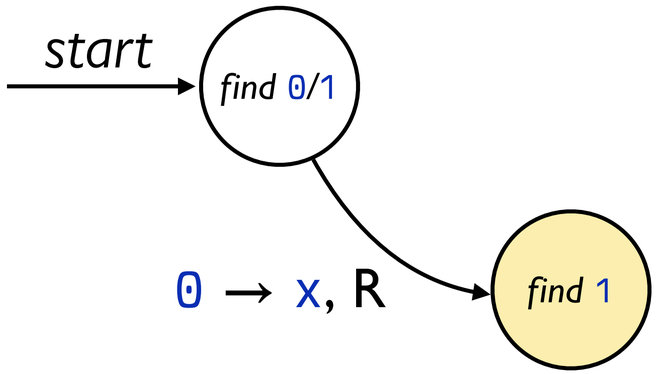

0 0 1 1 1 1 0 0

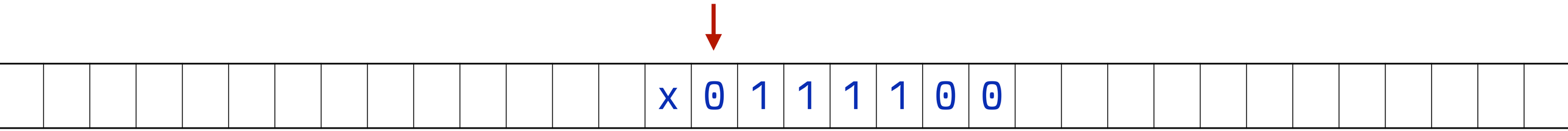

start
find 0/1
0 → x, R

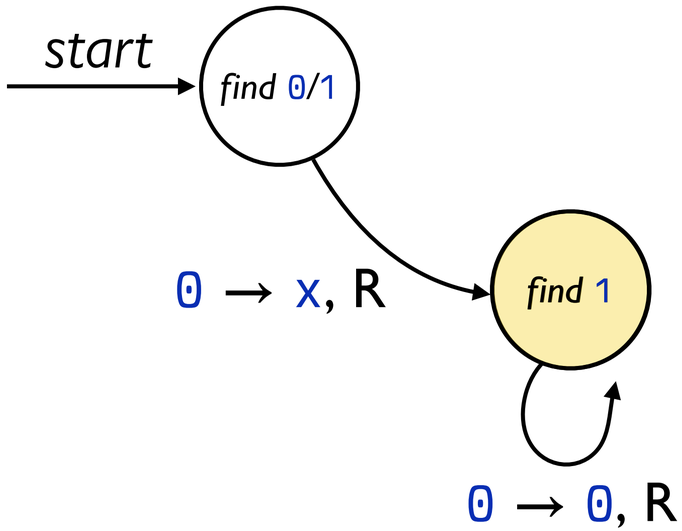

0 0 1 1 1 1 0 0

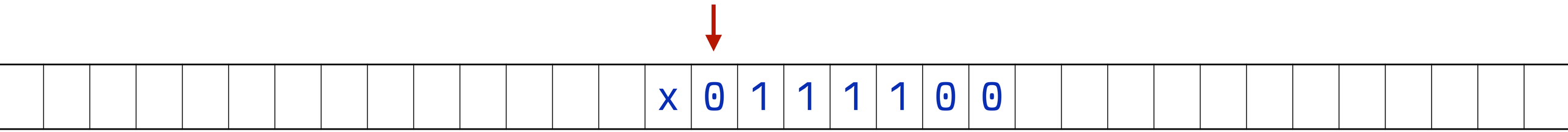

start
find 0/1
0 → x, R

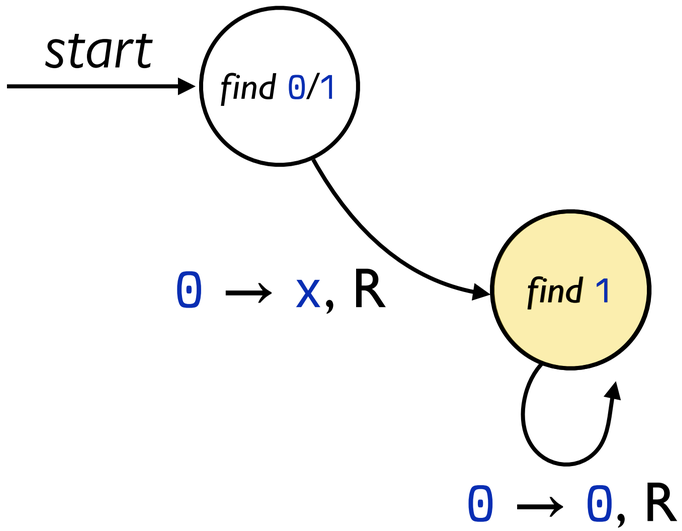

x 0 1 1 1 1 0 0

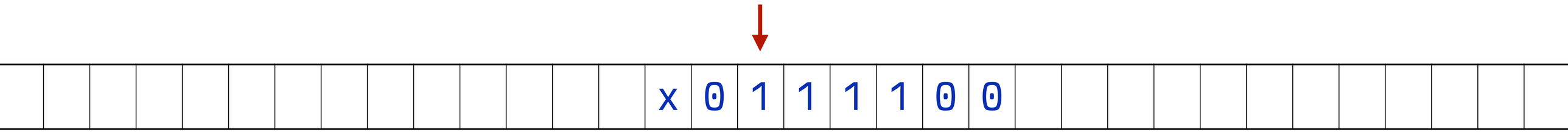

start
find 0/1
0 → x, R
find 1

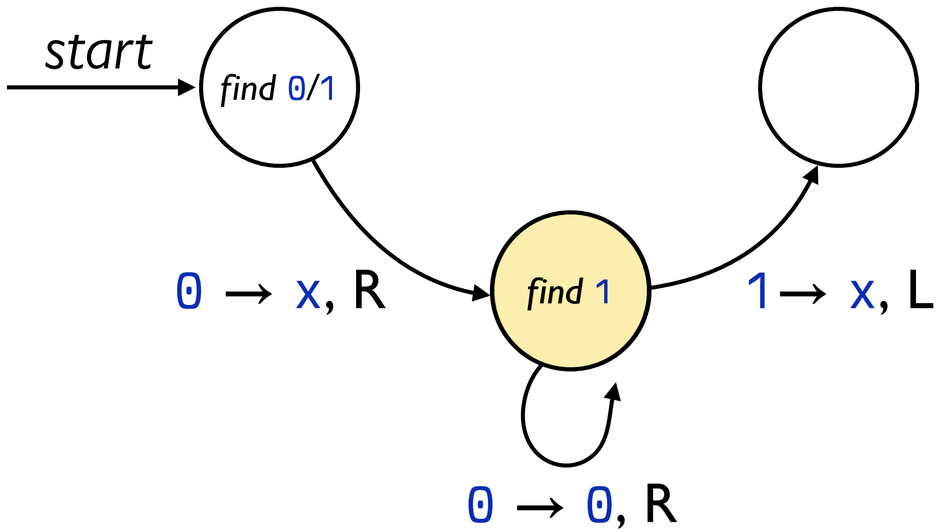

x 0 1 1 1 1 0 0

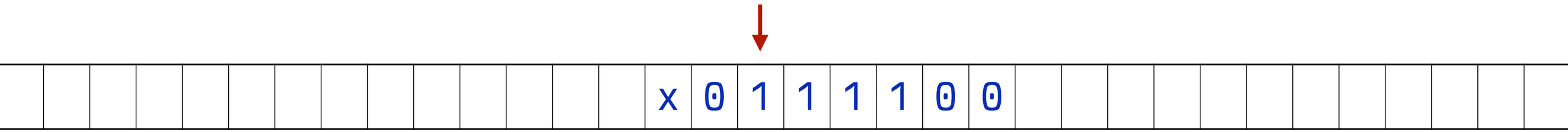

start
find 0/1
0 → x, R
find 1
0 → 0, R

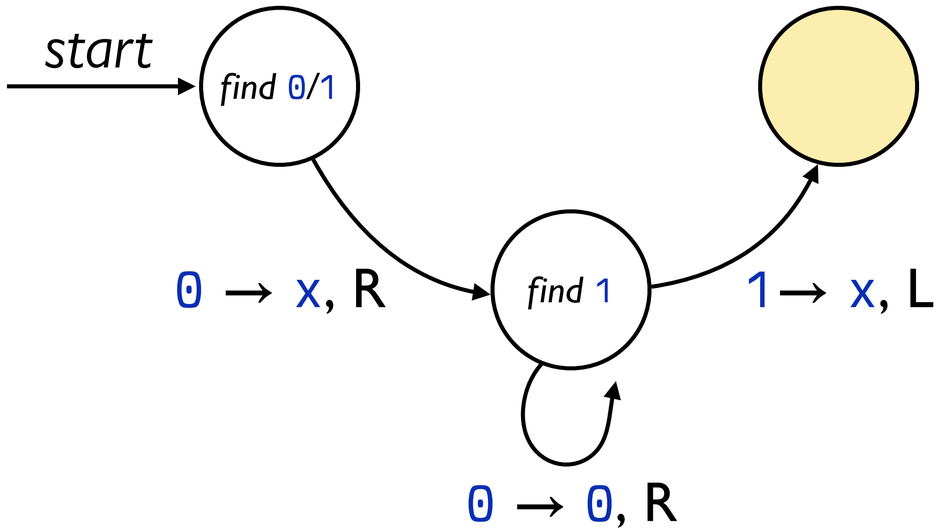

x 0 1 1 1 1 0 0

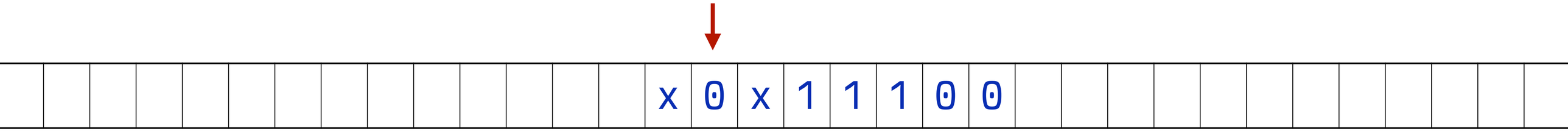

start
find 0/1
0 → x, R
find 1
0 → 0, R

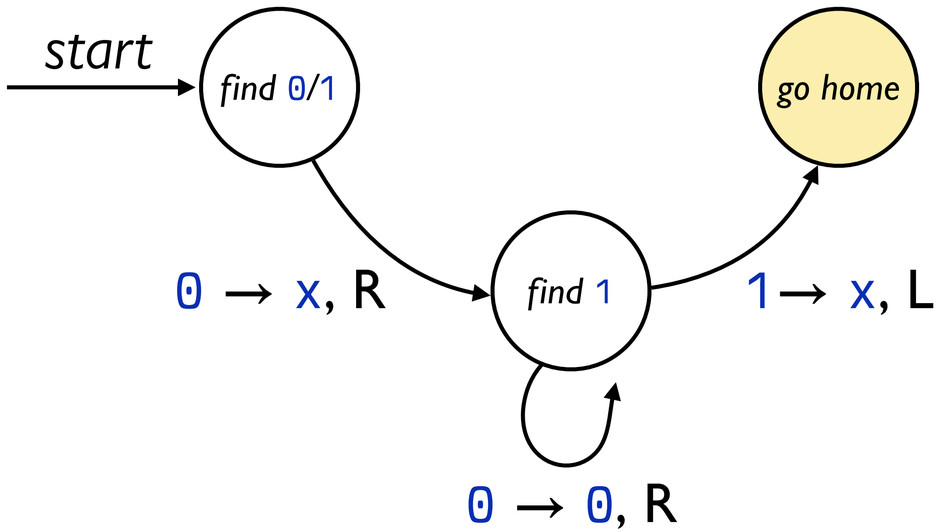

x 0 1 1 1 1 0 0

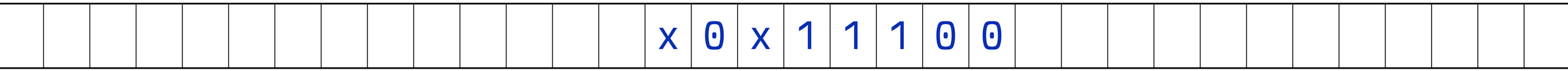

start
find 0/1
find 1
0 → x, R
1 → x, L
0 → 0, R

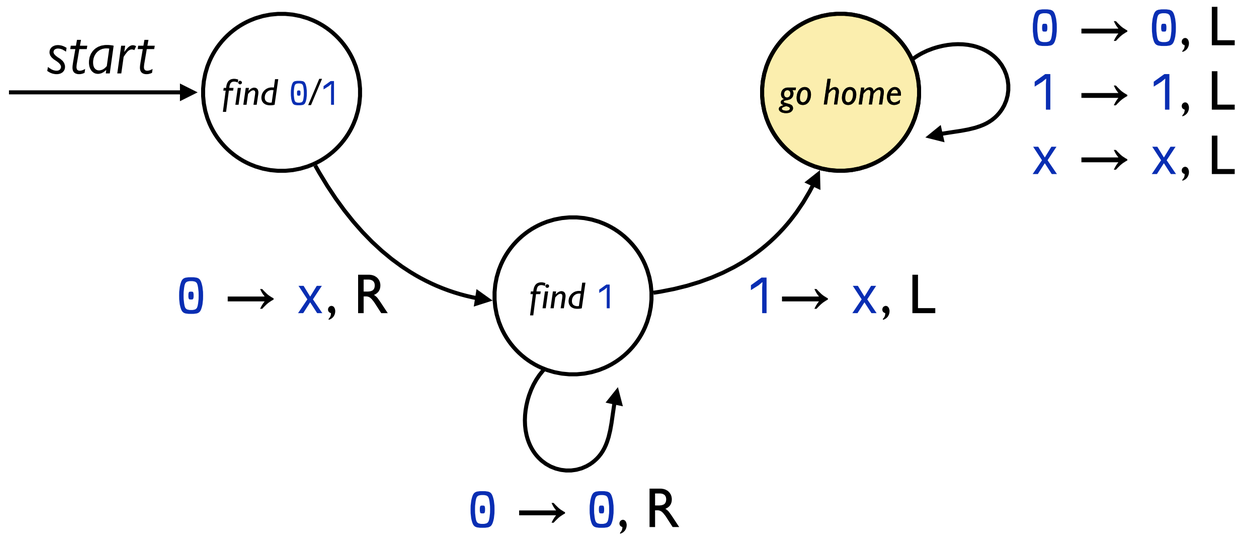

x 0 1 1 1 1 0 0

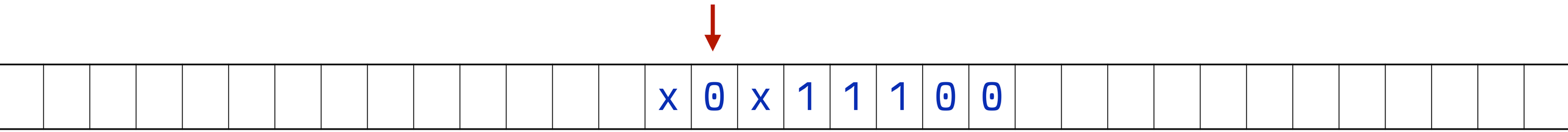

start
find 0/1
find 1
0 → x, R
1 → x, L
0 → 0, R

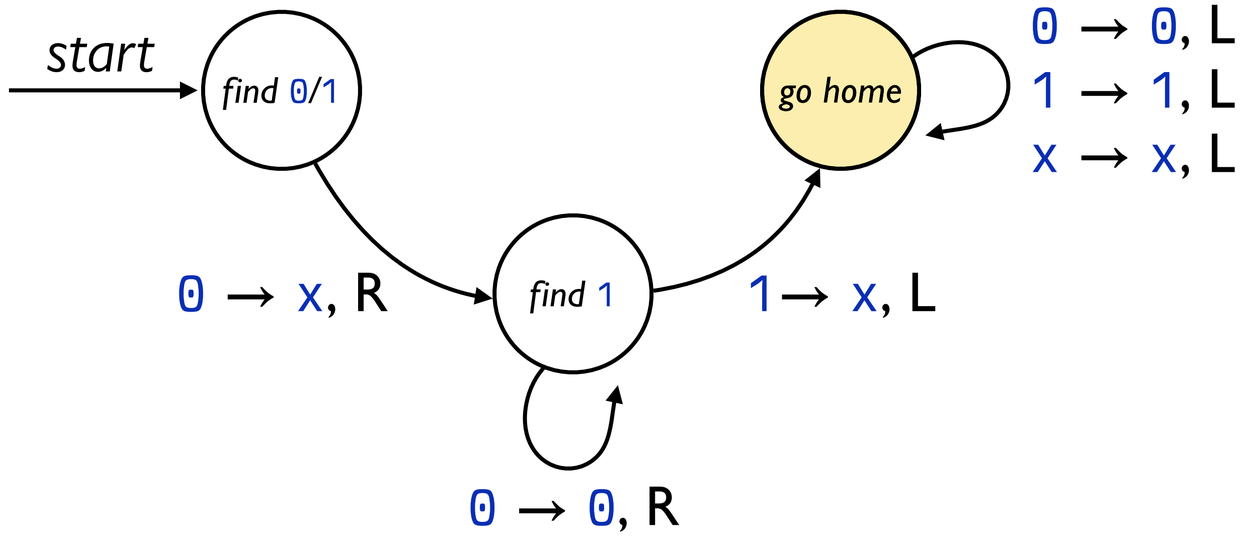

x 0 x 1 1 1 0 0

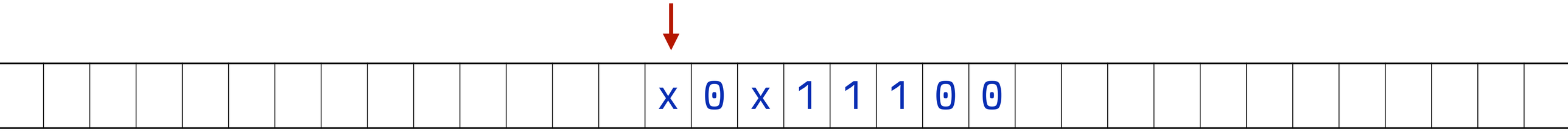

start
find 0/1
go home
find 1
0 → x, R
1 → x, L
0 → 0, R

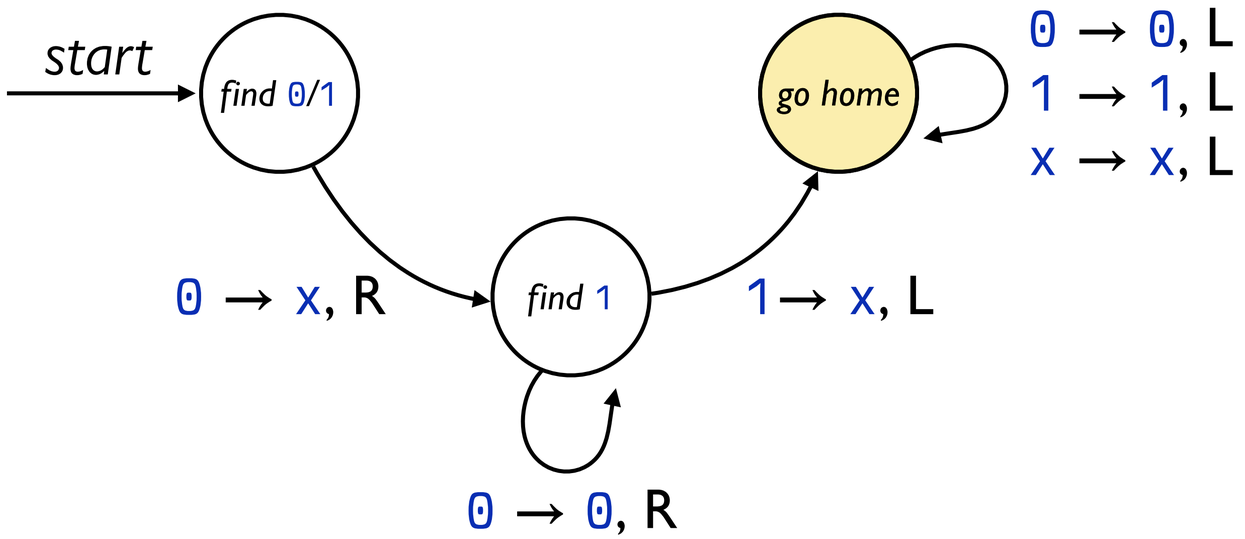

x 0 x 1 1 1 0 0

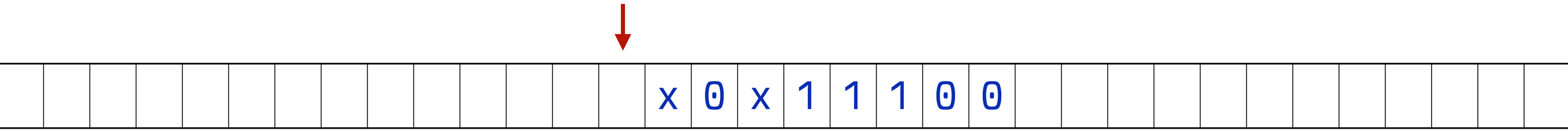

start
find 0/1
find 1
go home
0 → x, R
1 → x, L
0 → 0, R
0 → 0, L
1 → 1, L
x → x, L

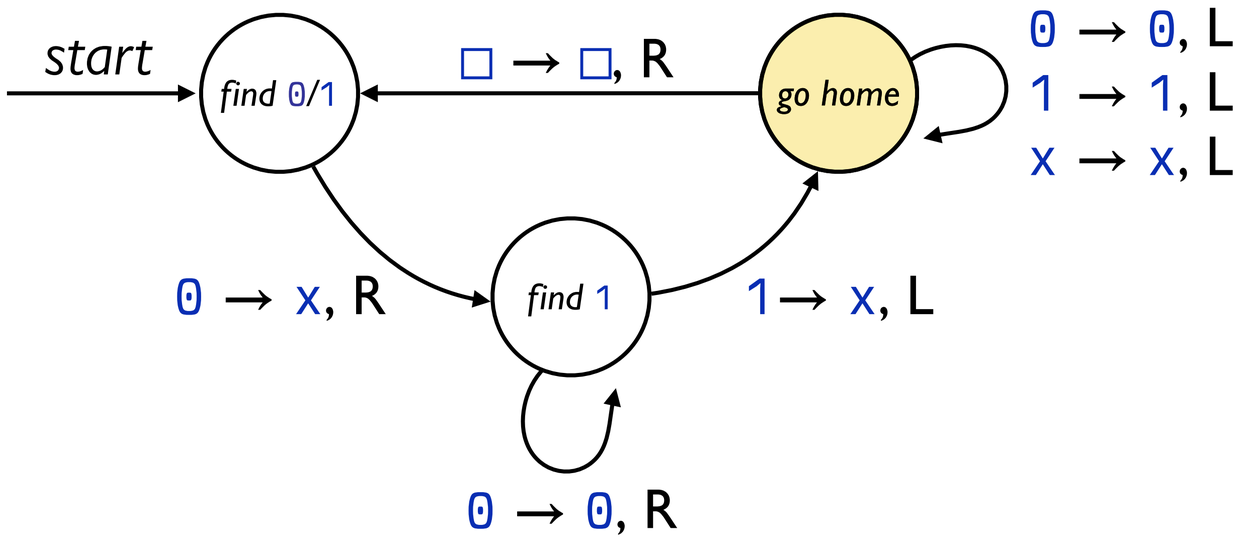

x 0 x 1 1 1 0 0

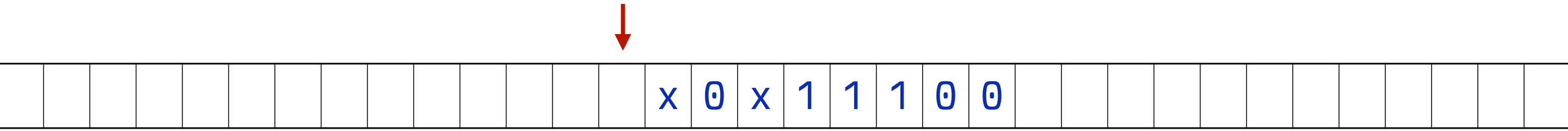

start
find 0/1
find 1
go home
0 → x, R
1 → x, L
0 → 0, R
0 → 0, L
1 → 1, L
x → x, L

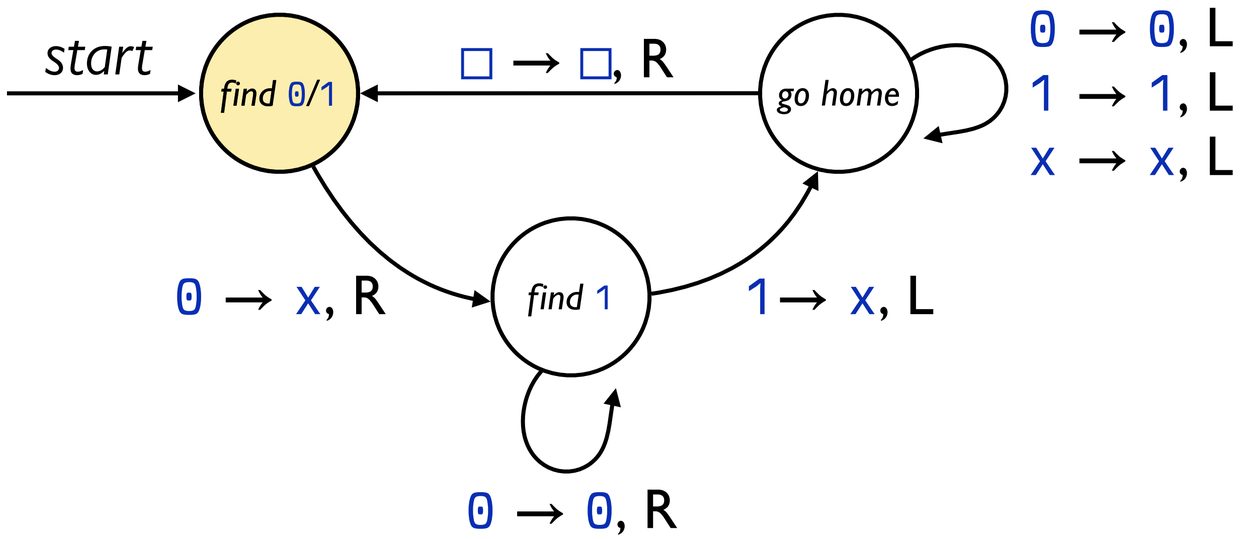

x 0 x 1 1 1 0 0

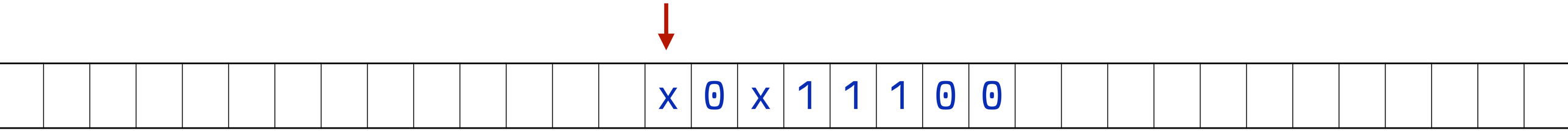

start
find 0/1
find 1
go home
0 → x, R
1 → x, L
0 → 0, R
0 → 0, L
1 → 1, L
x → x, L

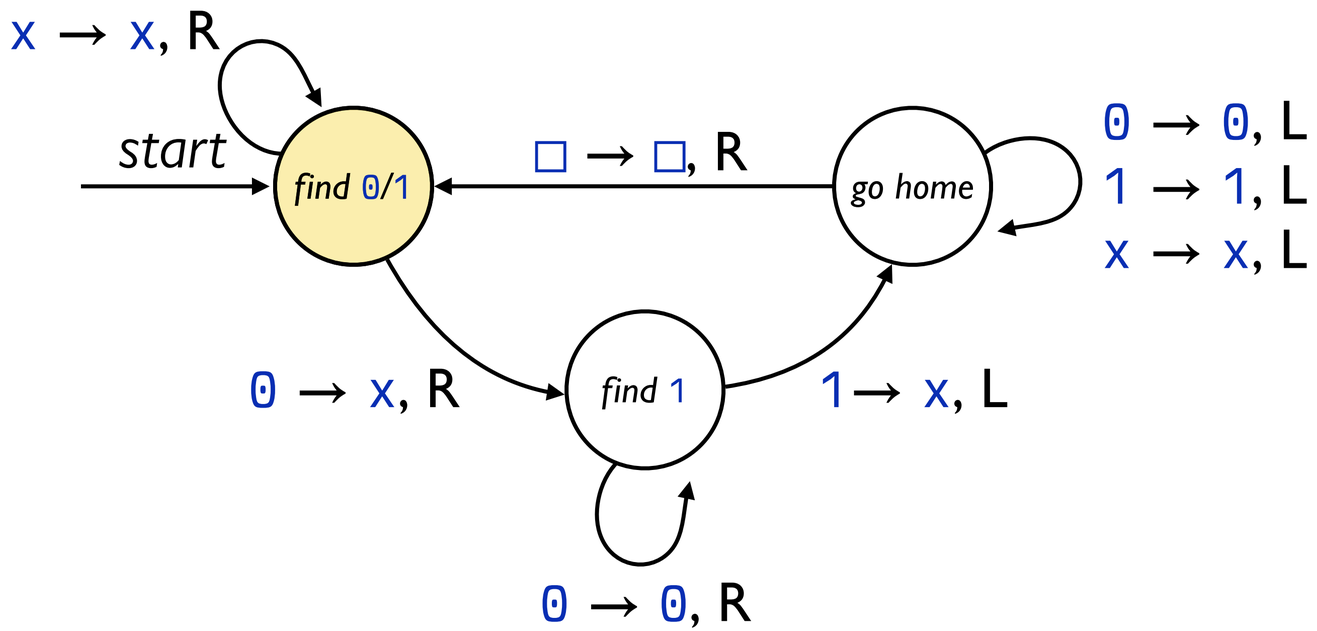

x 0 x 1 1 1 0 0

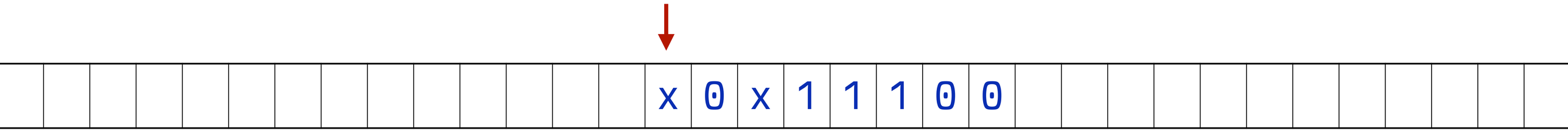

start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R

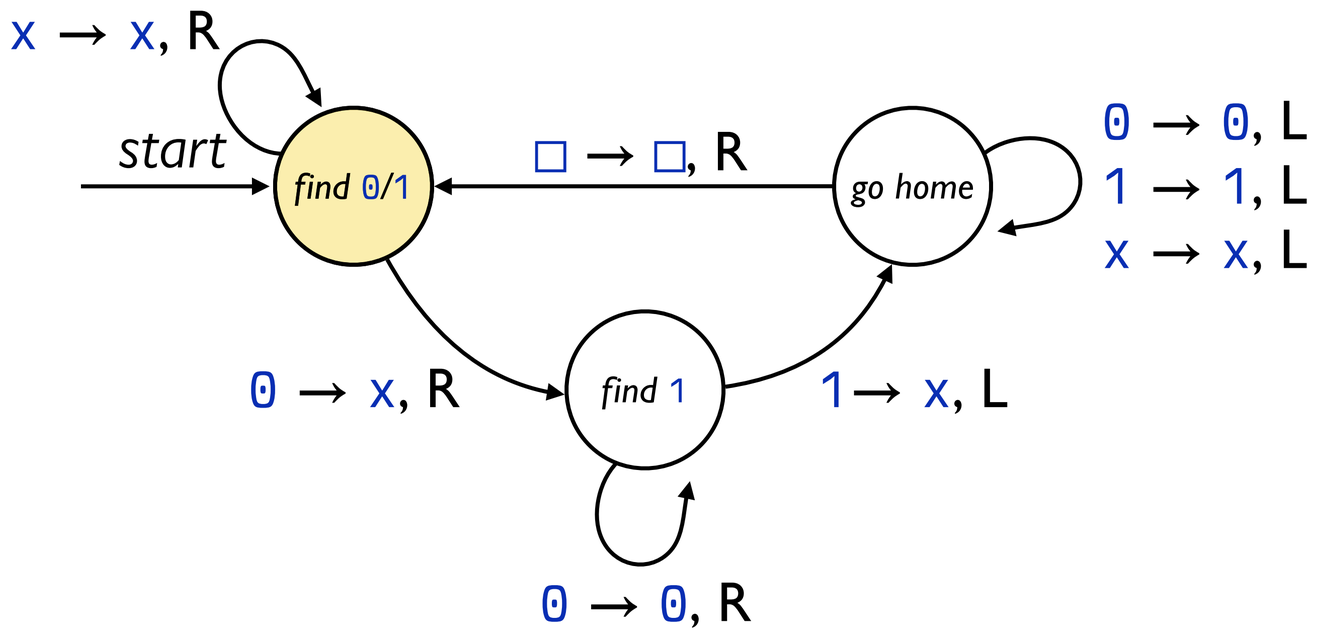

x 0 x 1 1 1 0 0

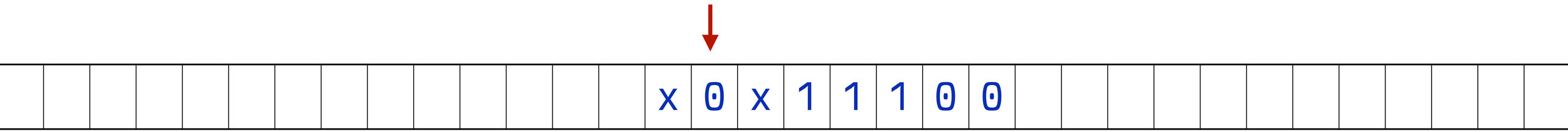

start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R

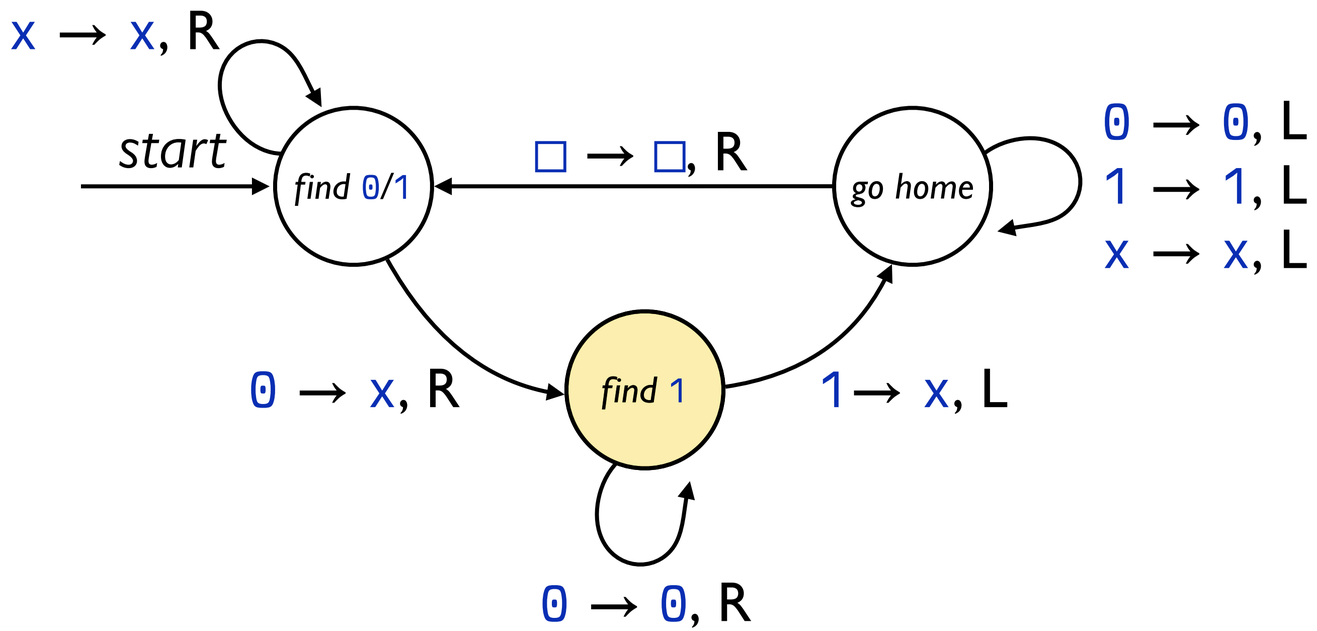

x 0 x 1 1 1 0 0

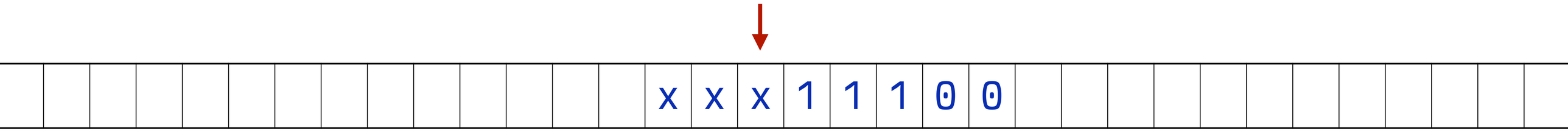

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R


x 0 x 1 1 1 0 0

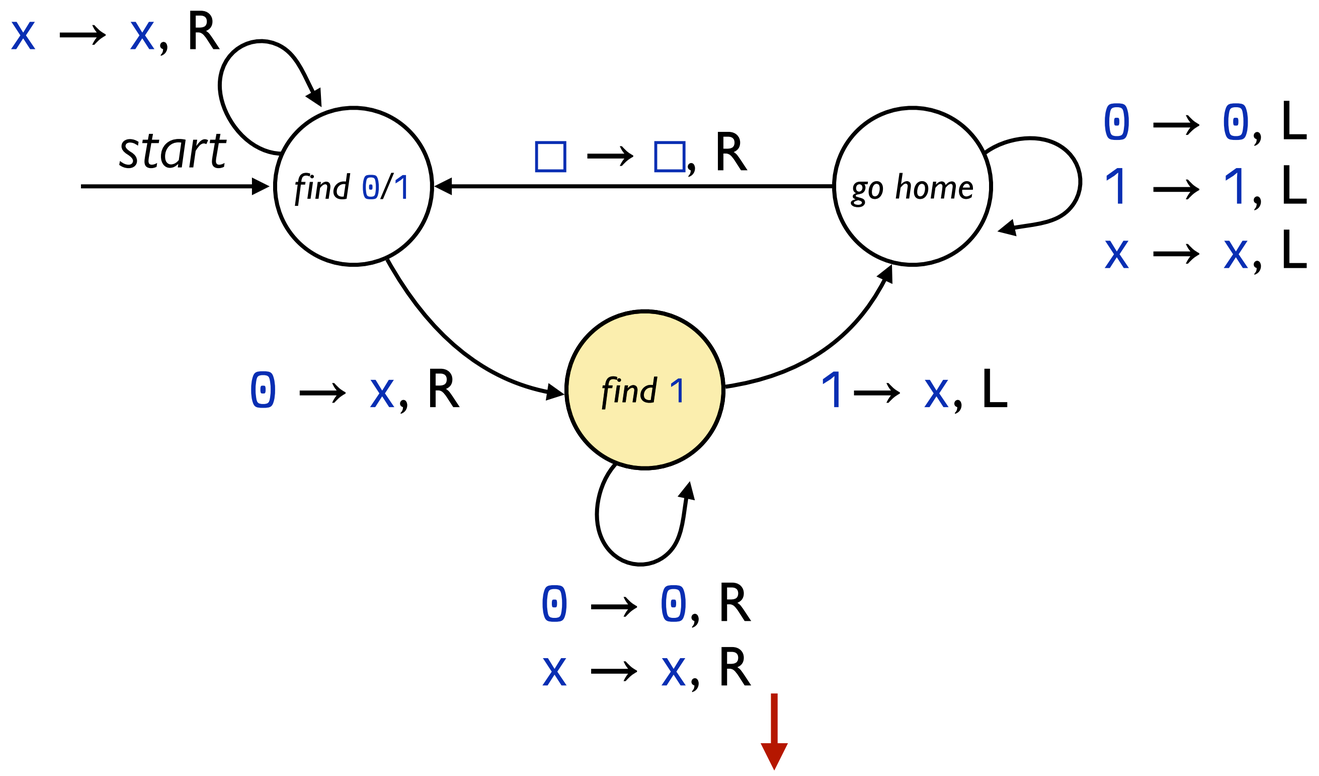

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R

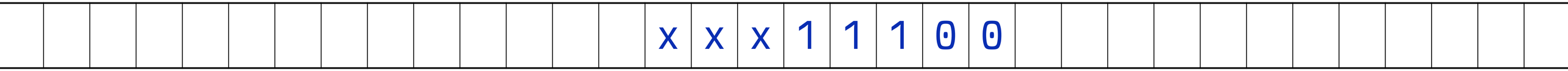

x 0 x 1 1 1 0 0

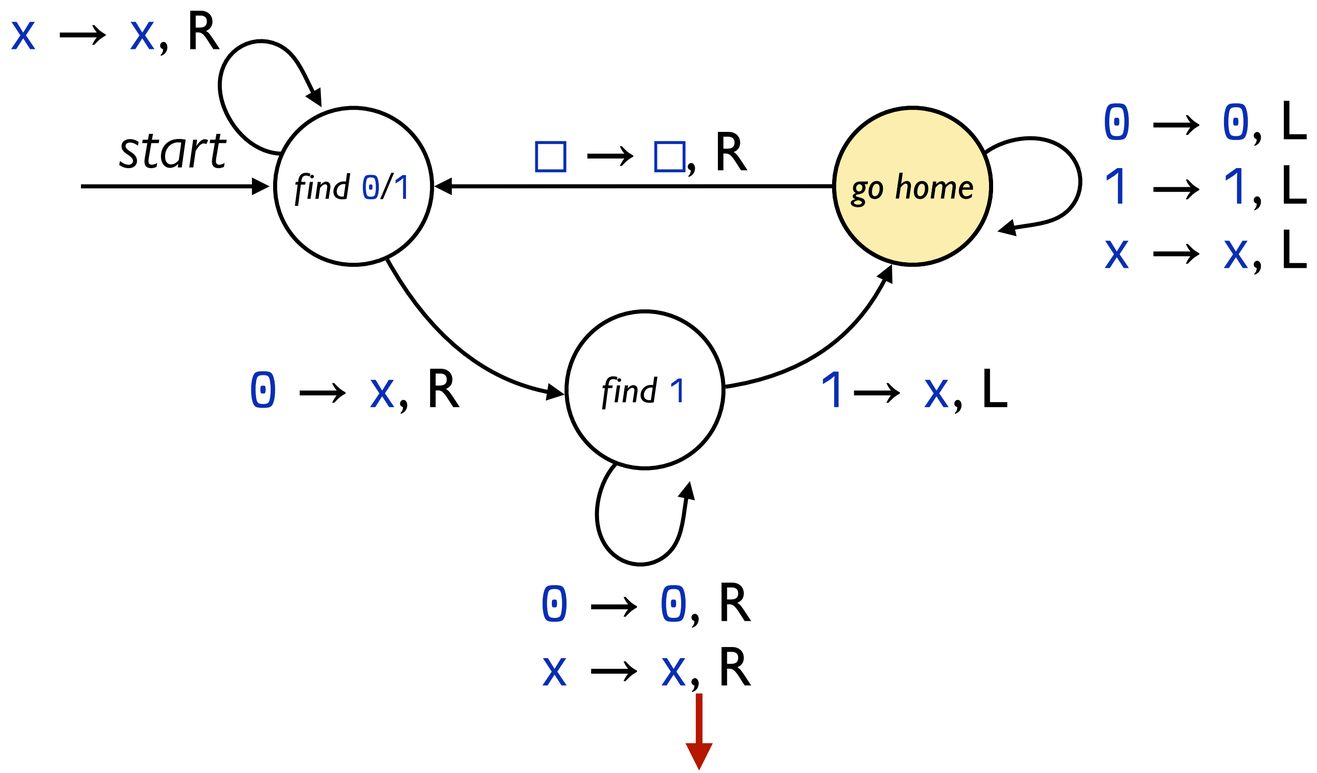

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R

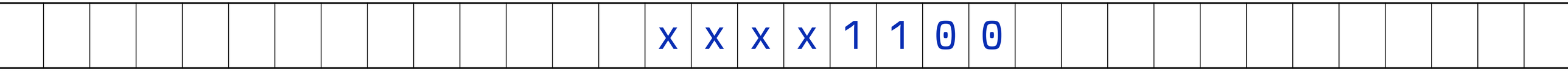

x x x 1 1 1 0 0

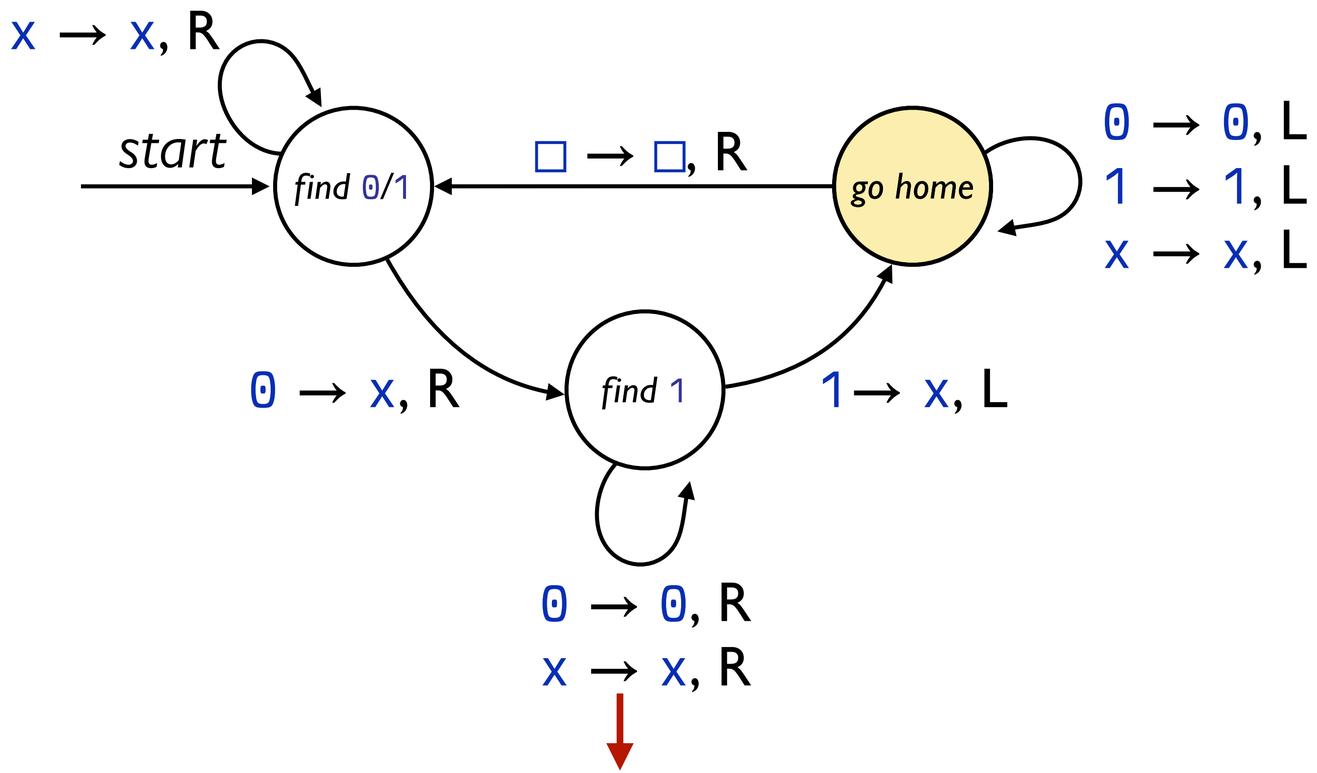

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1→ x, L
0 → 0, R
x → x, R
x x x 1 1 1 0 0

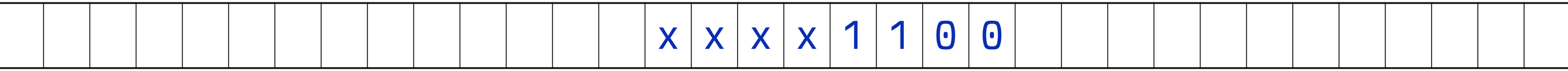

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

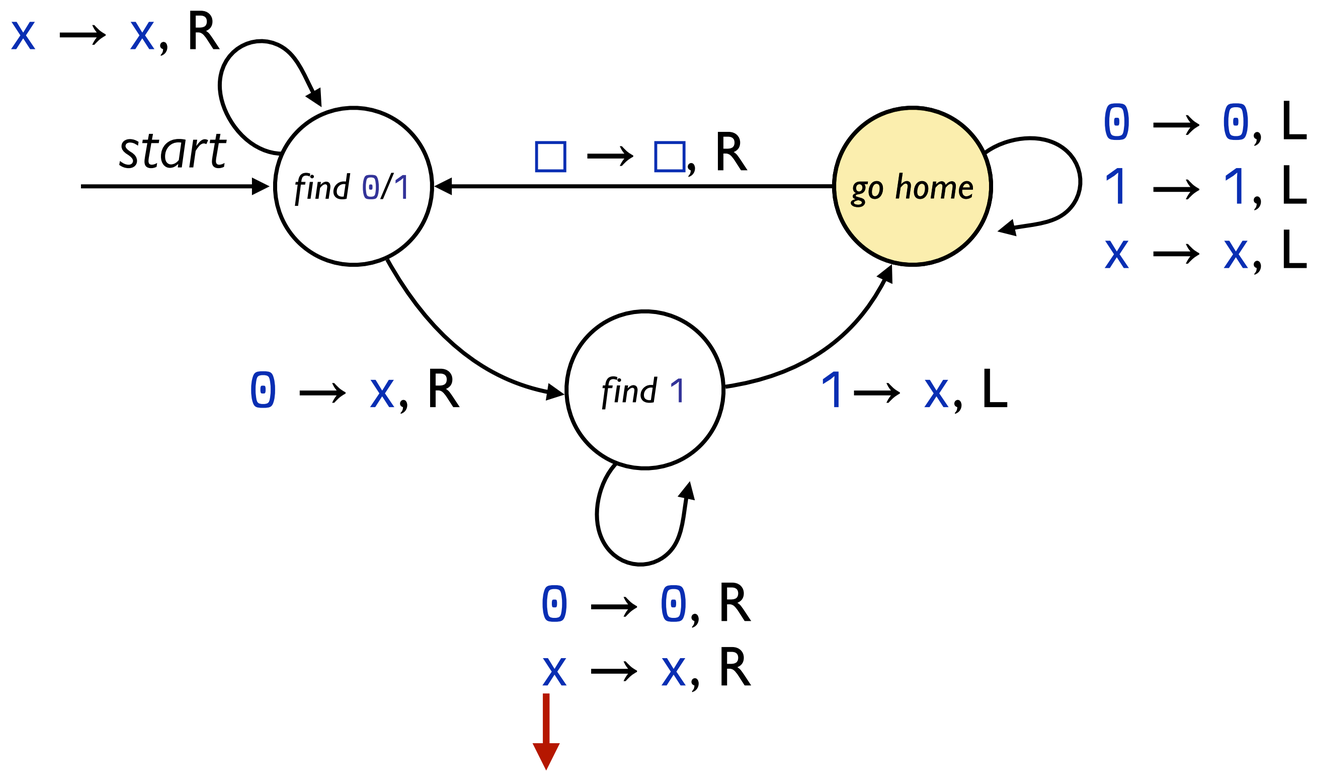

x x x 1 1 1 0 0

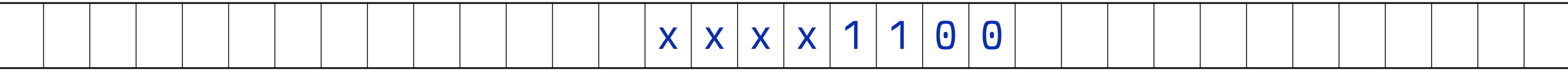

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

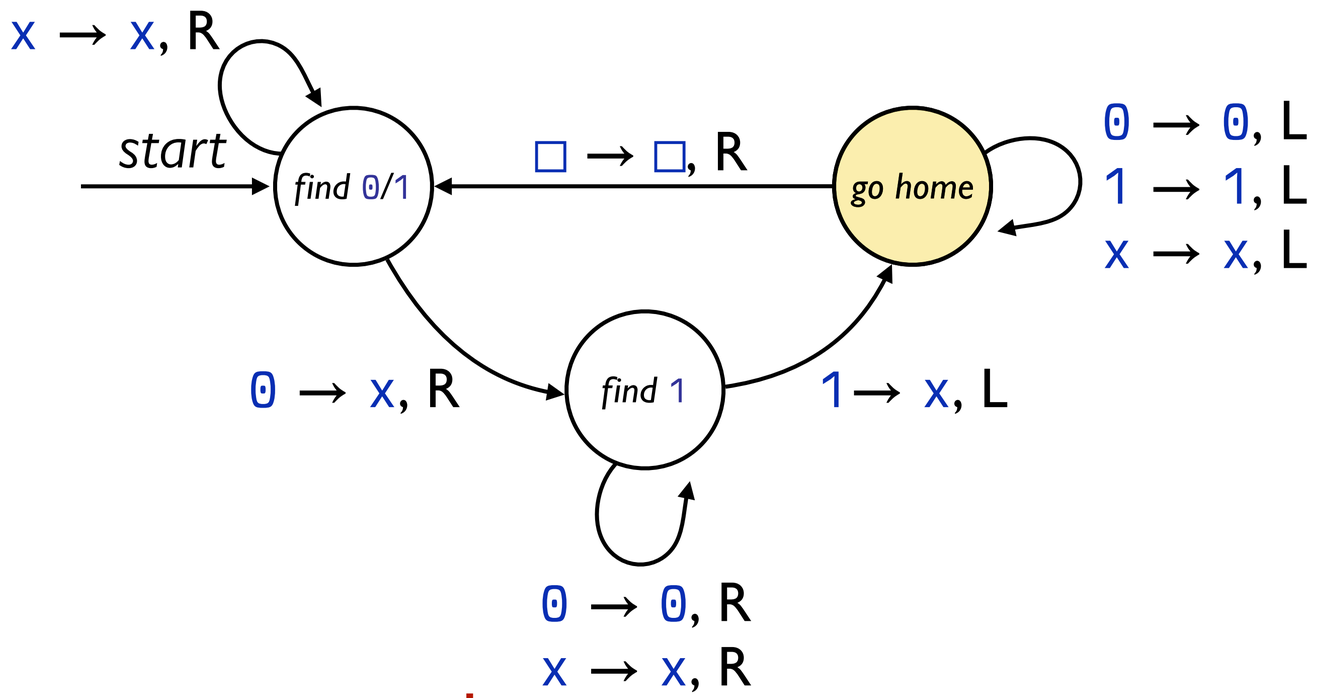

x x x x 1 1 0 0

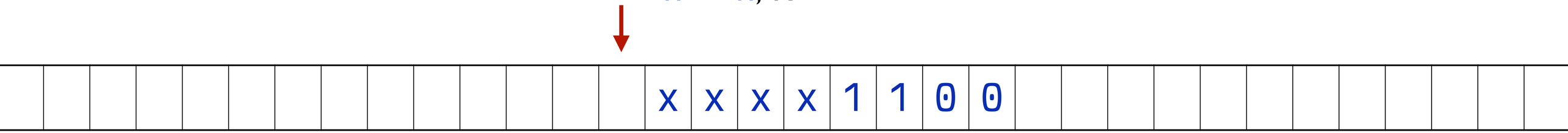

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

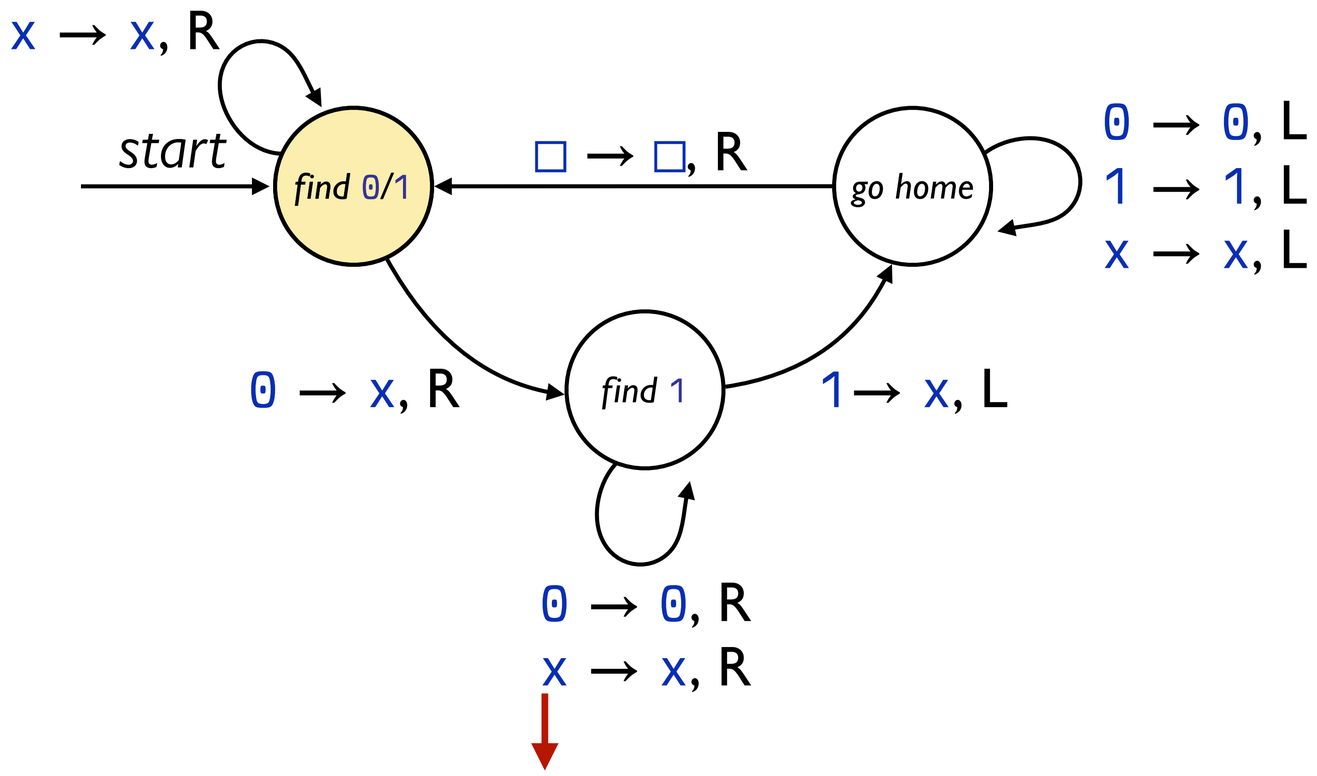

x x x x 1 1 0 0

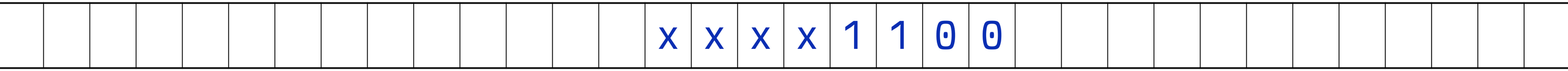

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

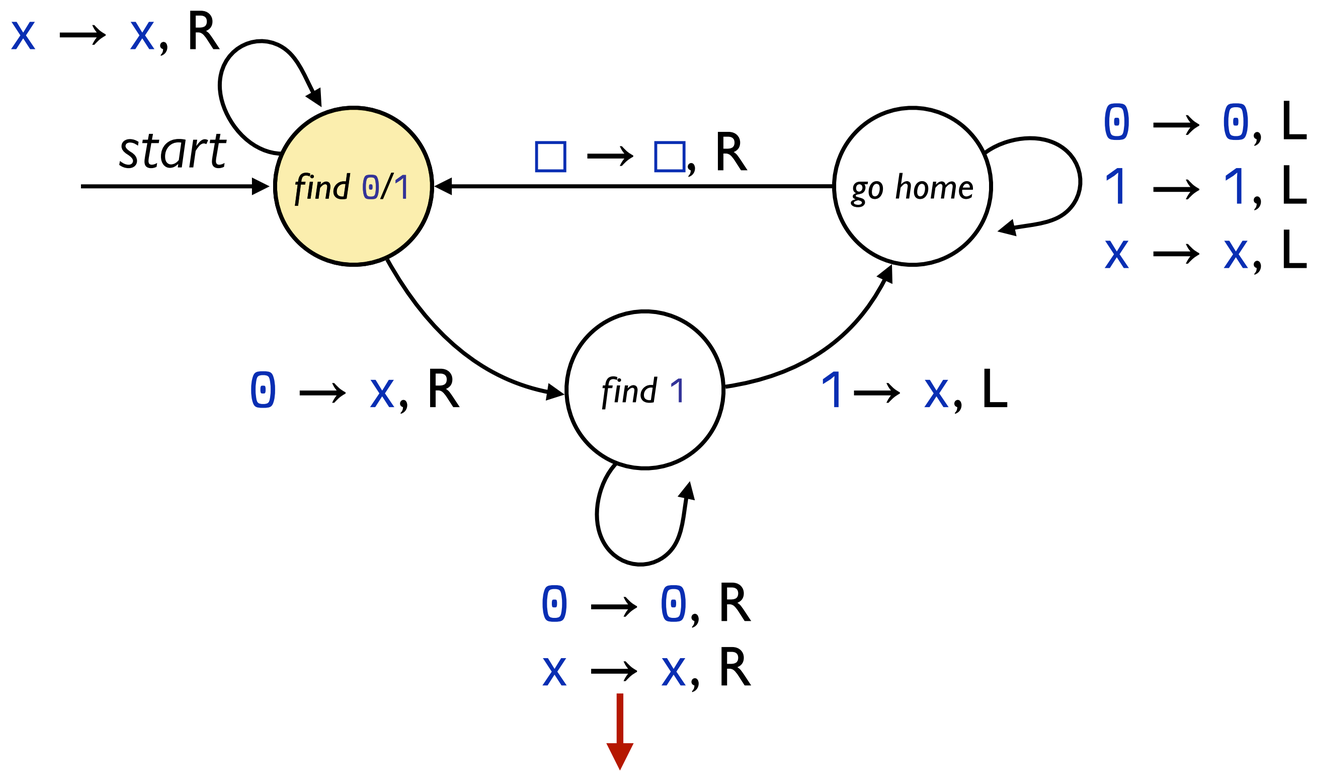

x x x x 1 1 0 0

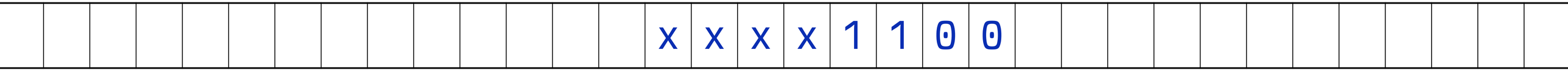

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

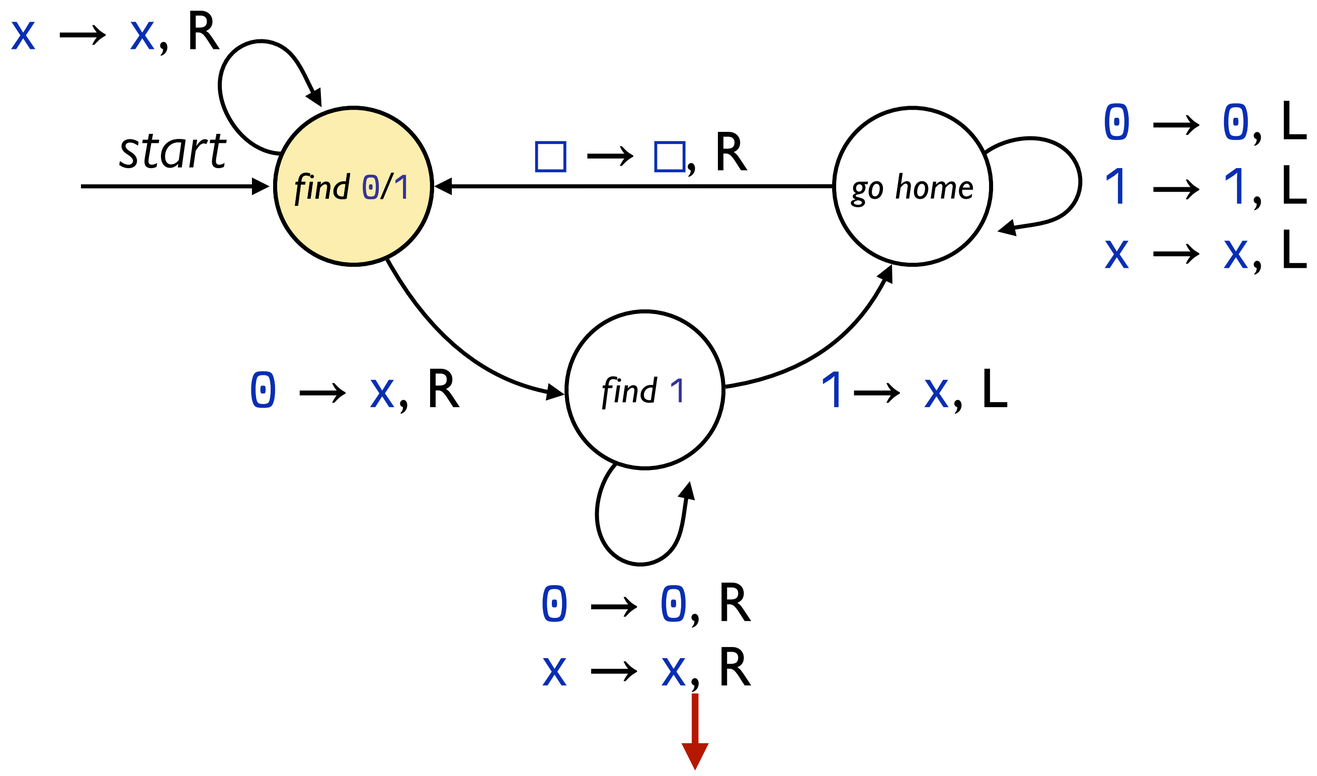

x x x x 1 1 0 0

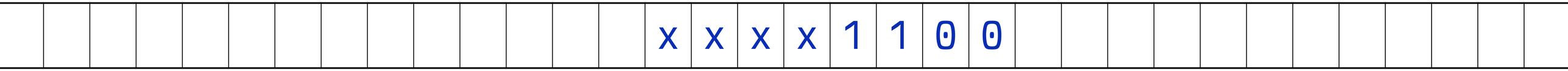

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

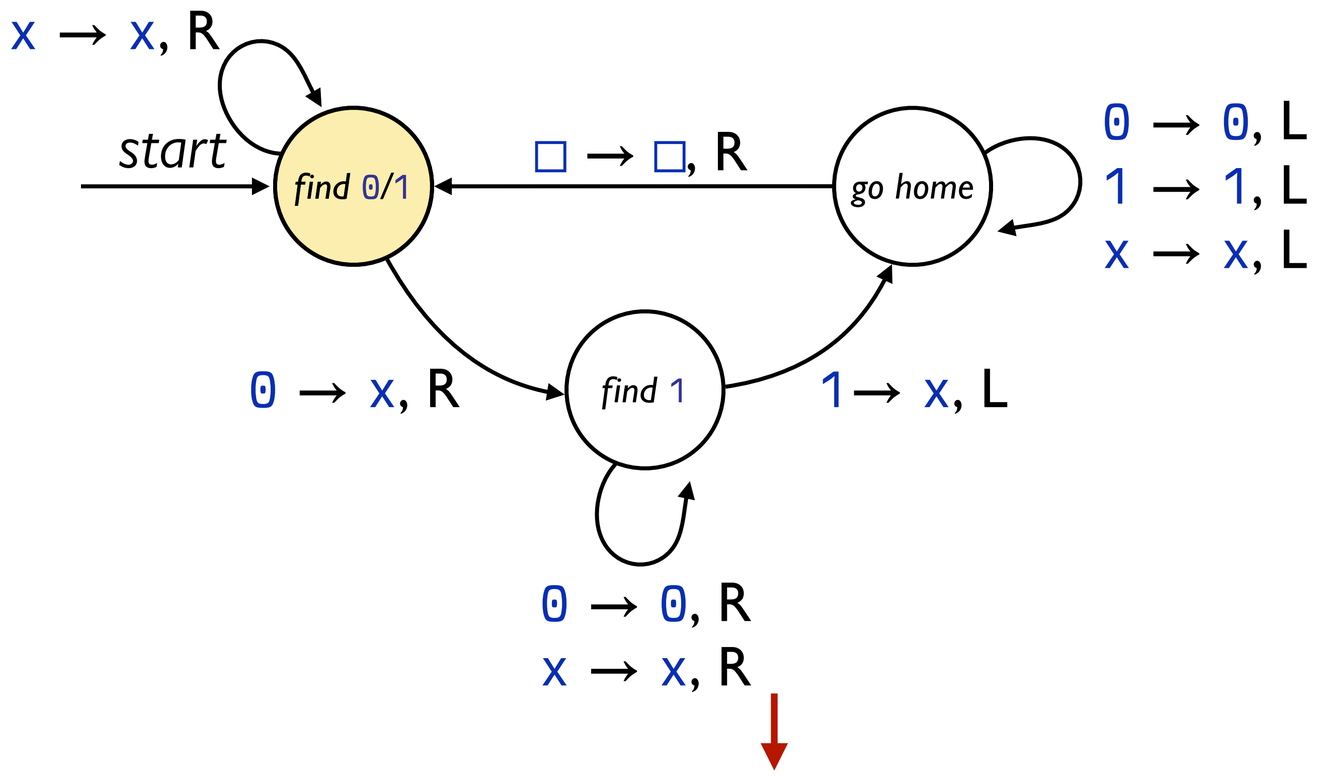

x x x x 1 1 0 0

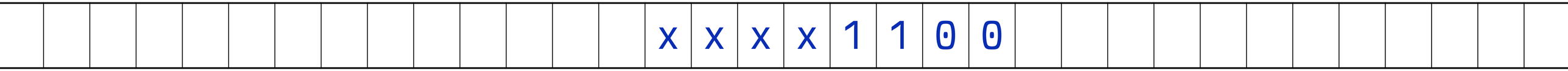

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

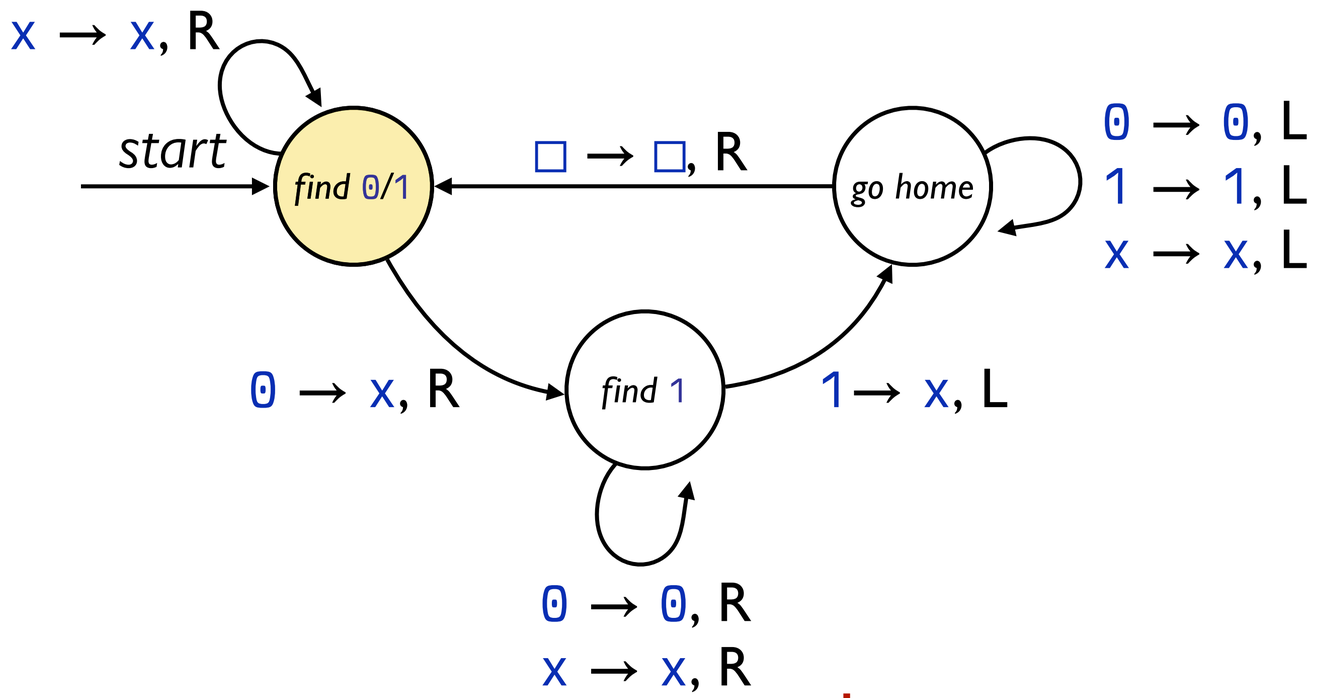

x x x x 1 1 0 0

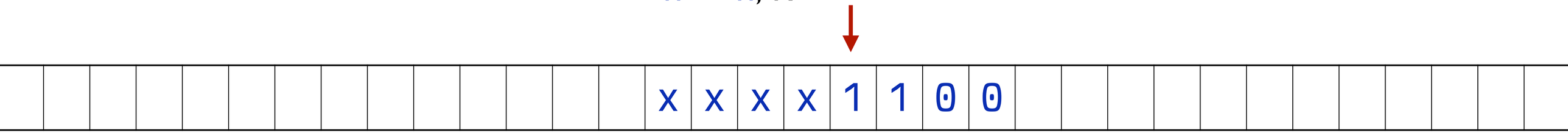

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

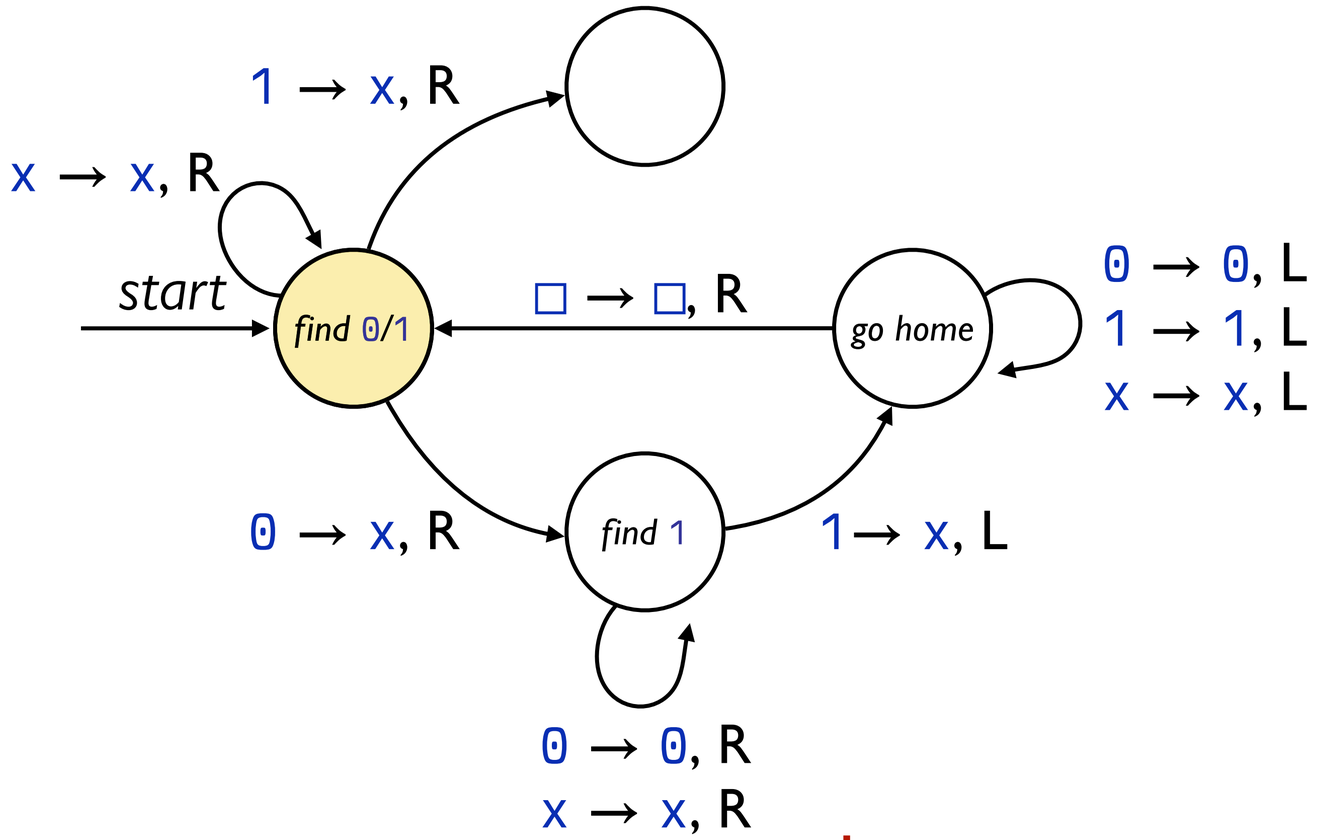

x x x x 1 1 0 0

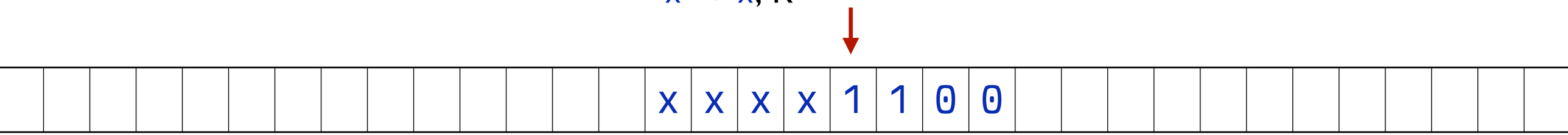

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

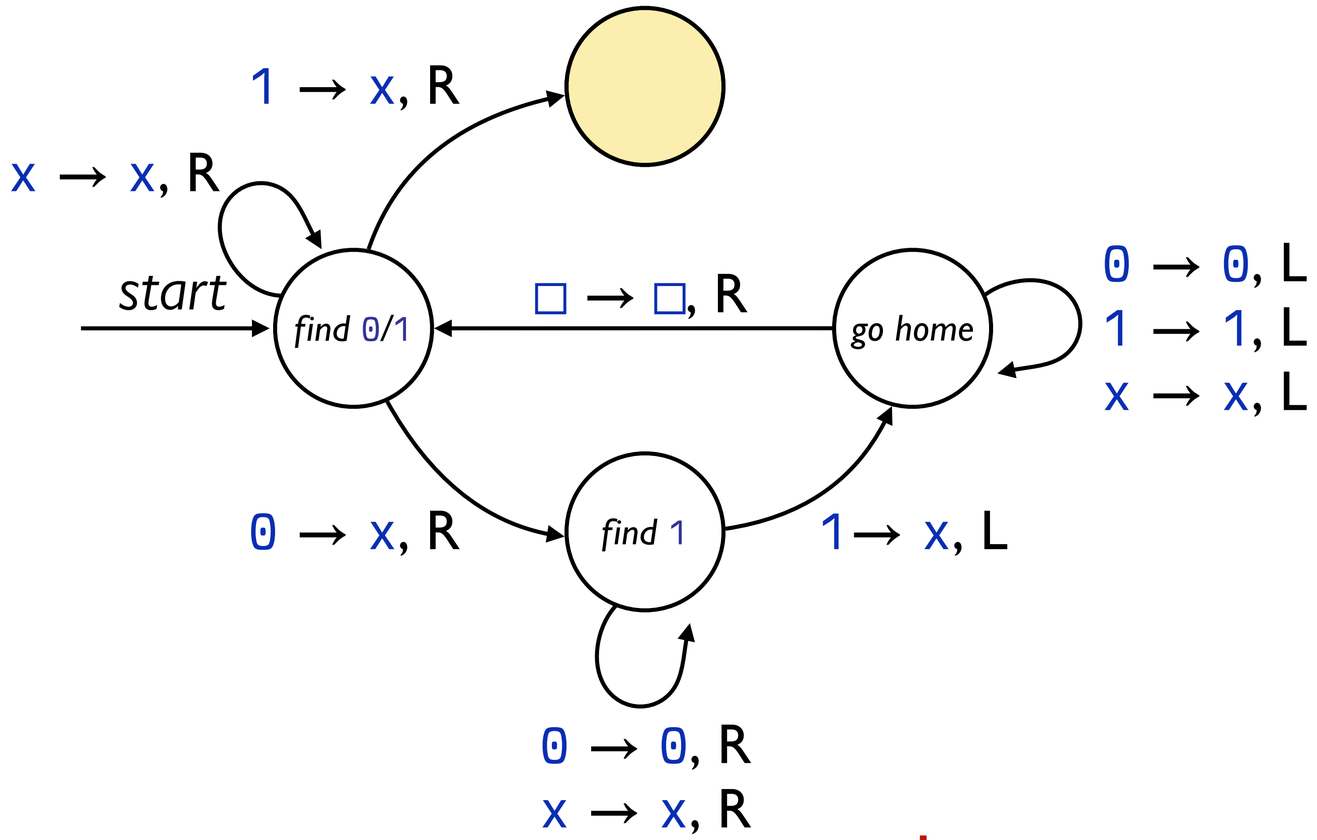

x x x x 1 1 0 0

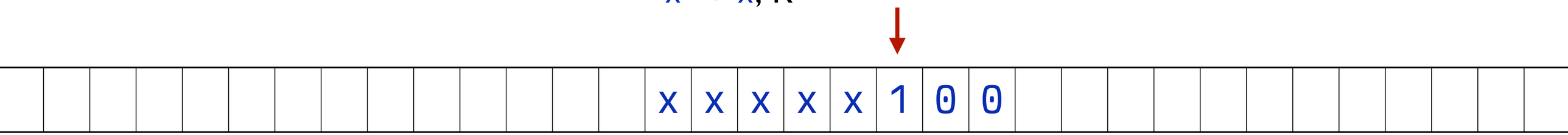

x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

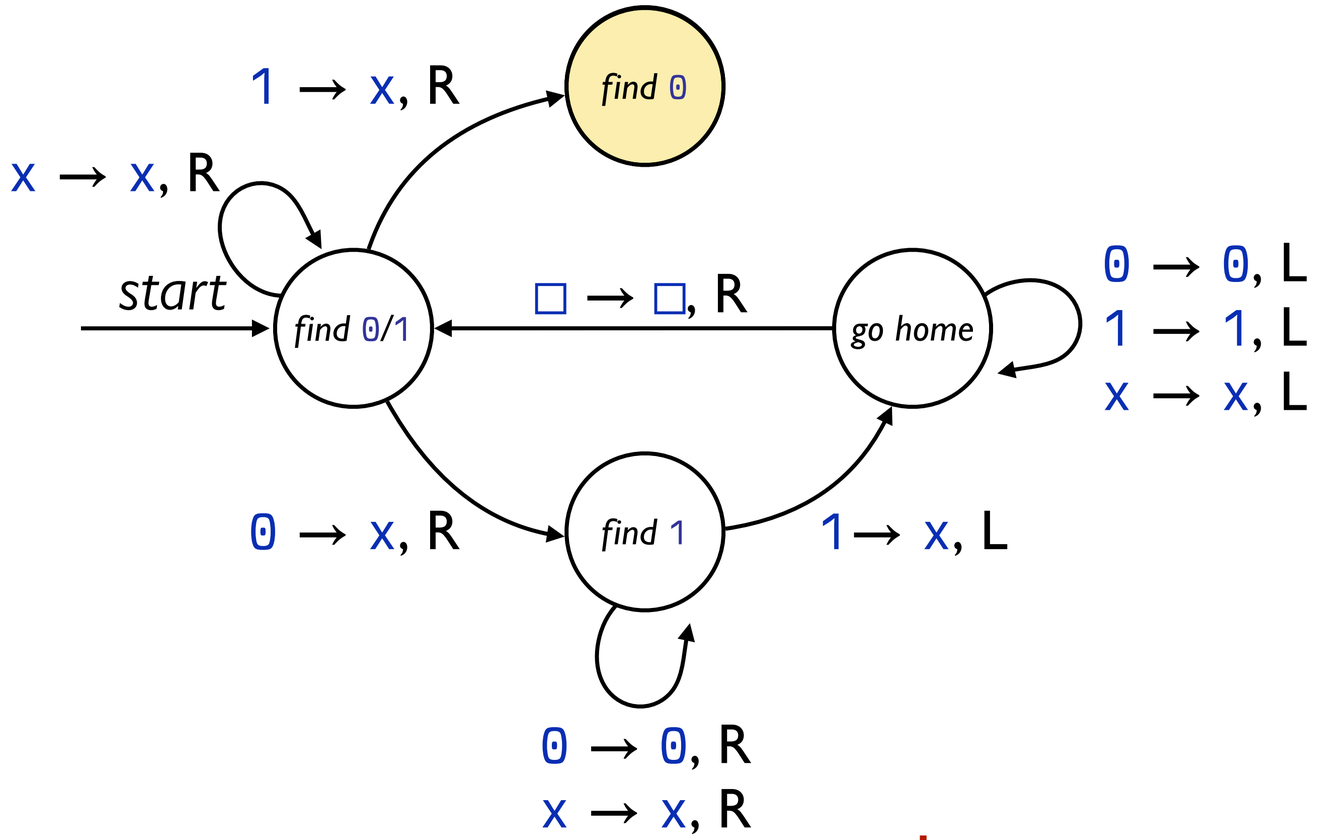

x x x x 1 1 0 0

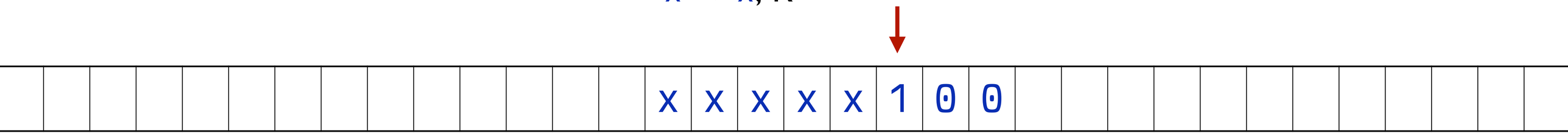

1 → x, R
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

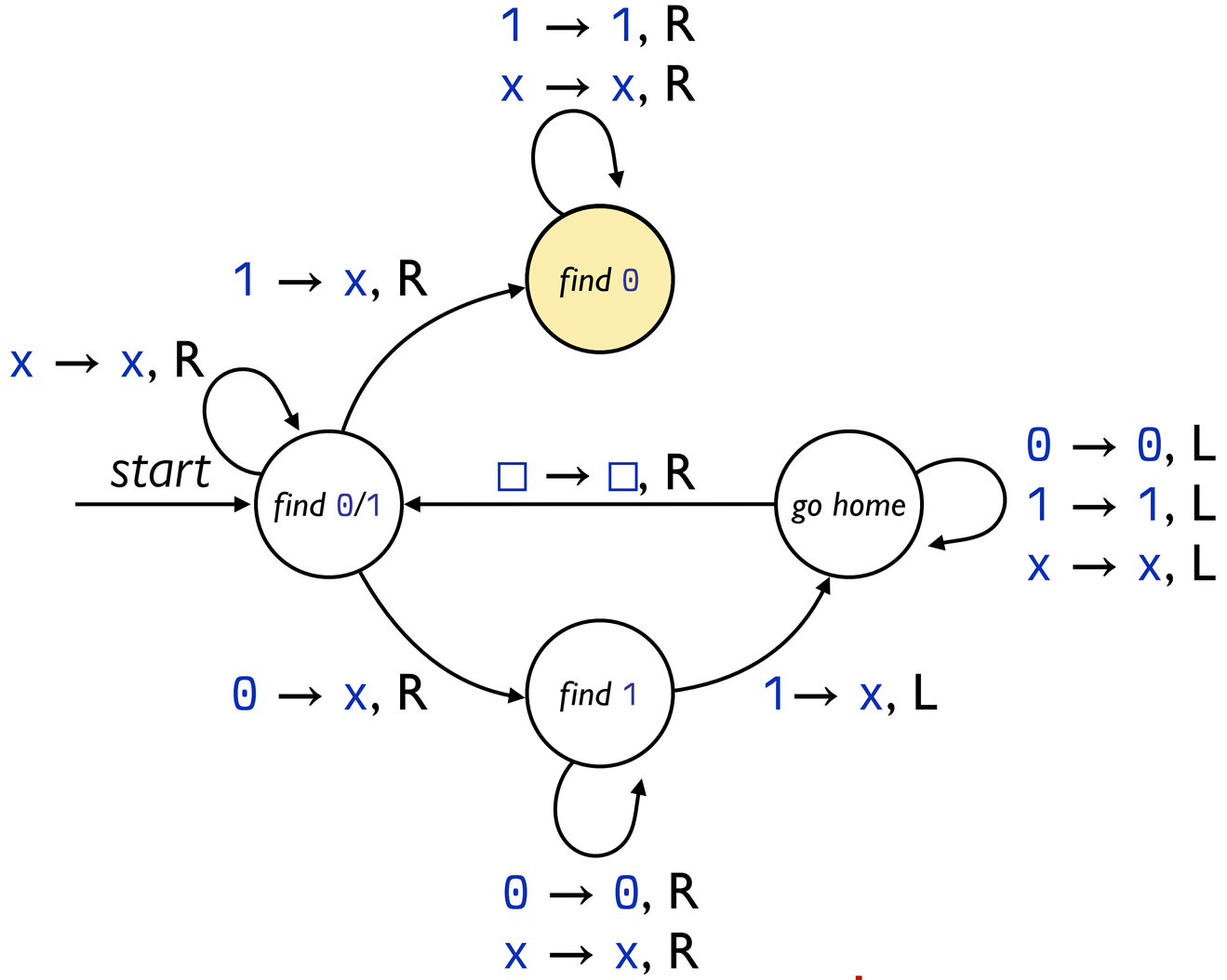

x x x x 1 1 0 0

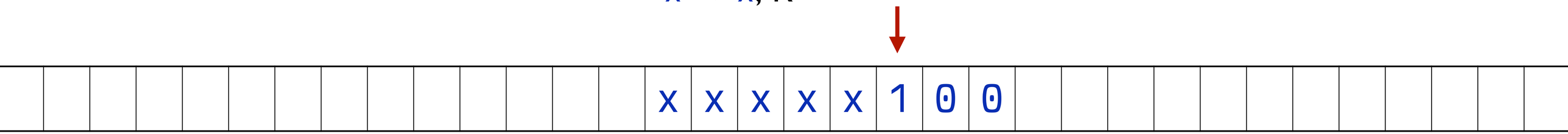

1 → x, R
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

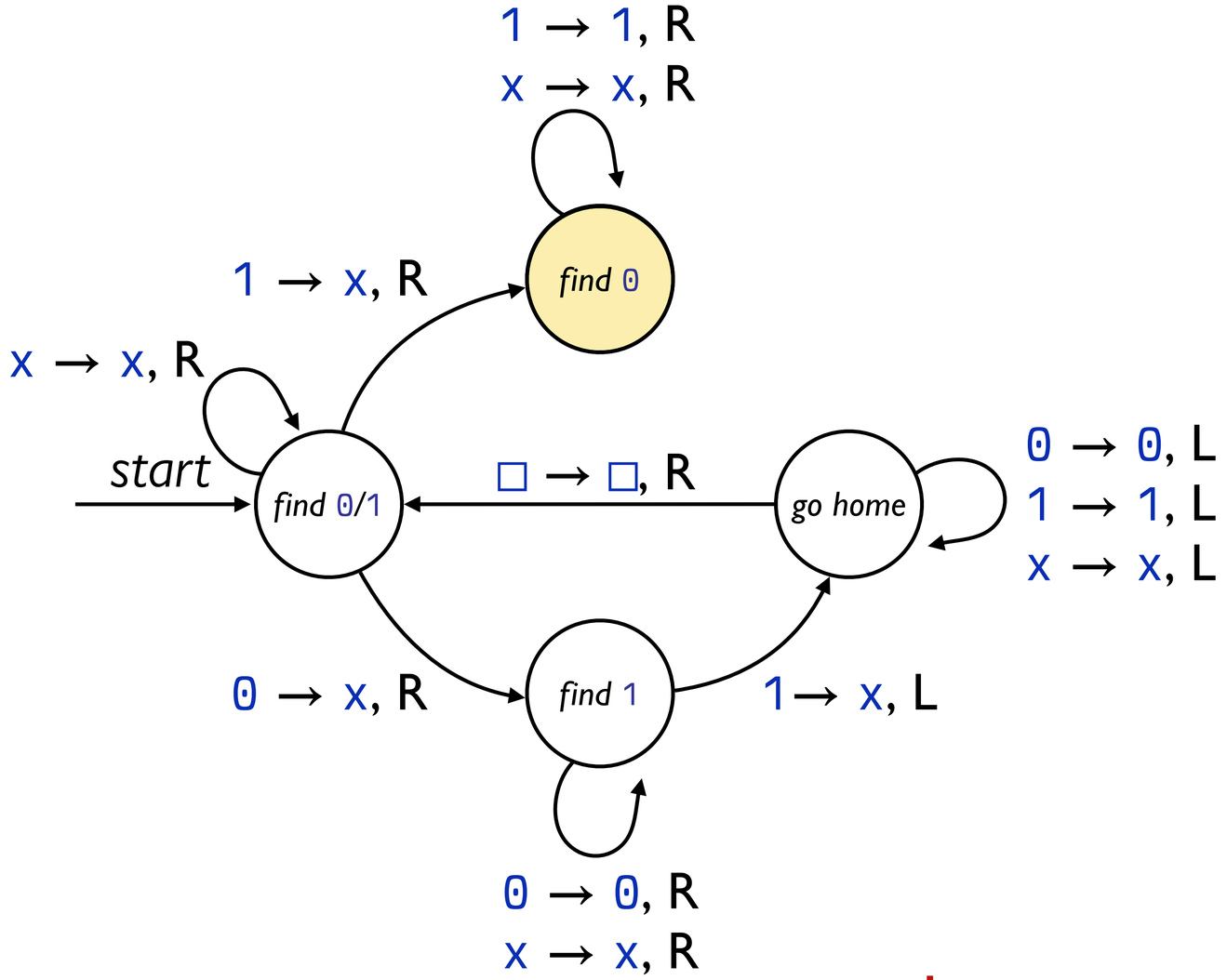

x x x x x 1 0 0

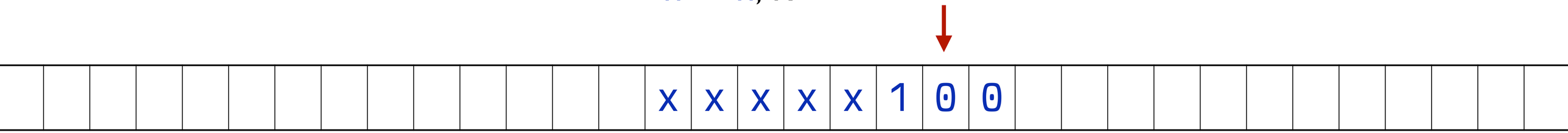

1 → x, R
find 0
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

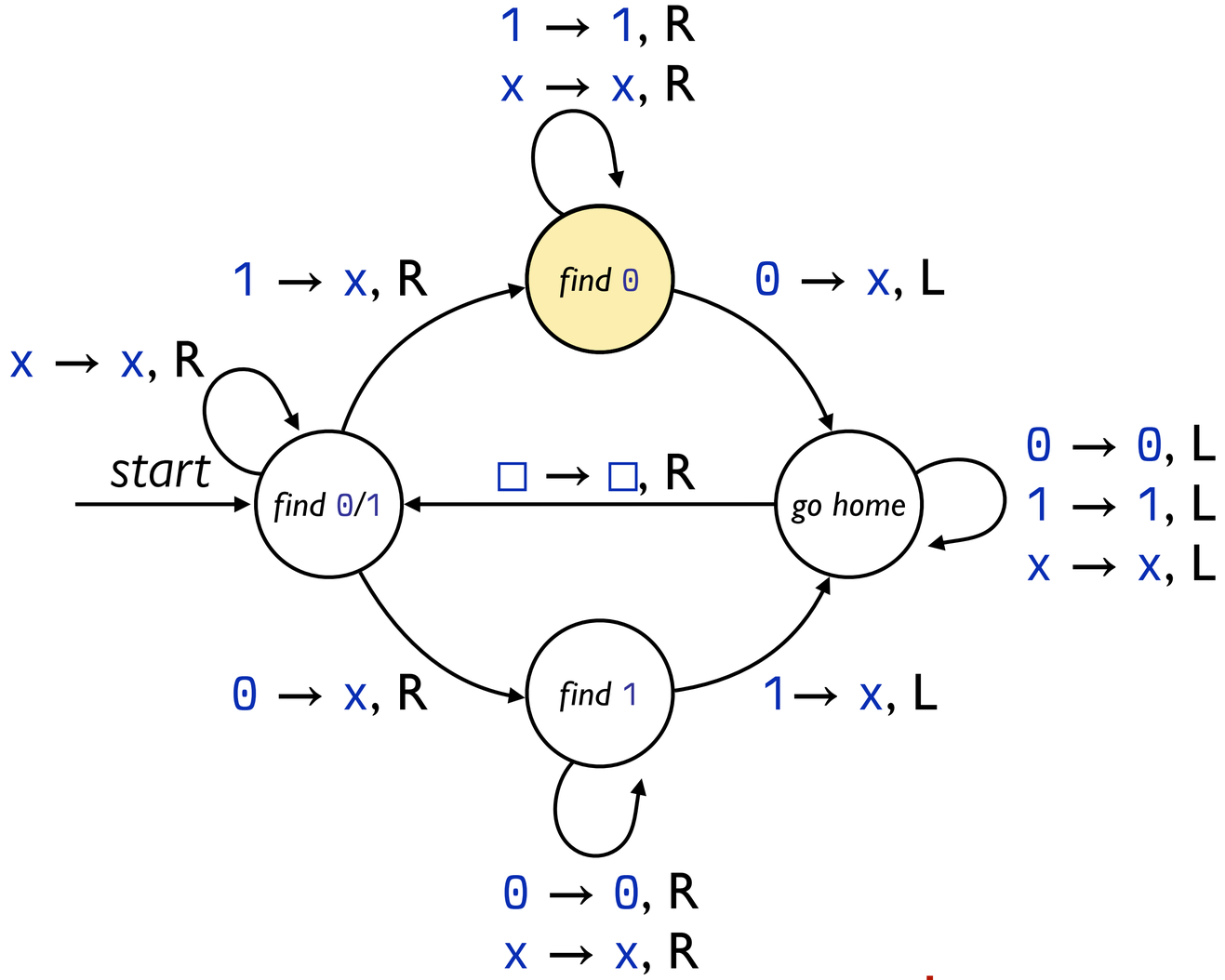

x x x x x 1 0 0

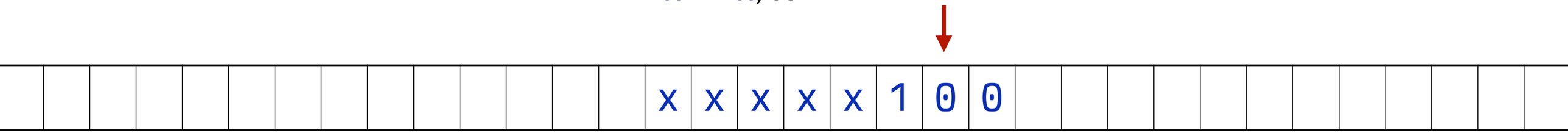

1 → 1, R
x → x, R
find 0
1 → x, R
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

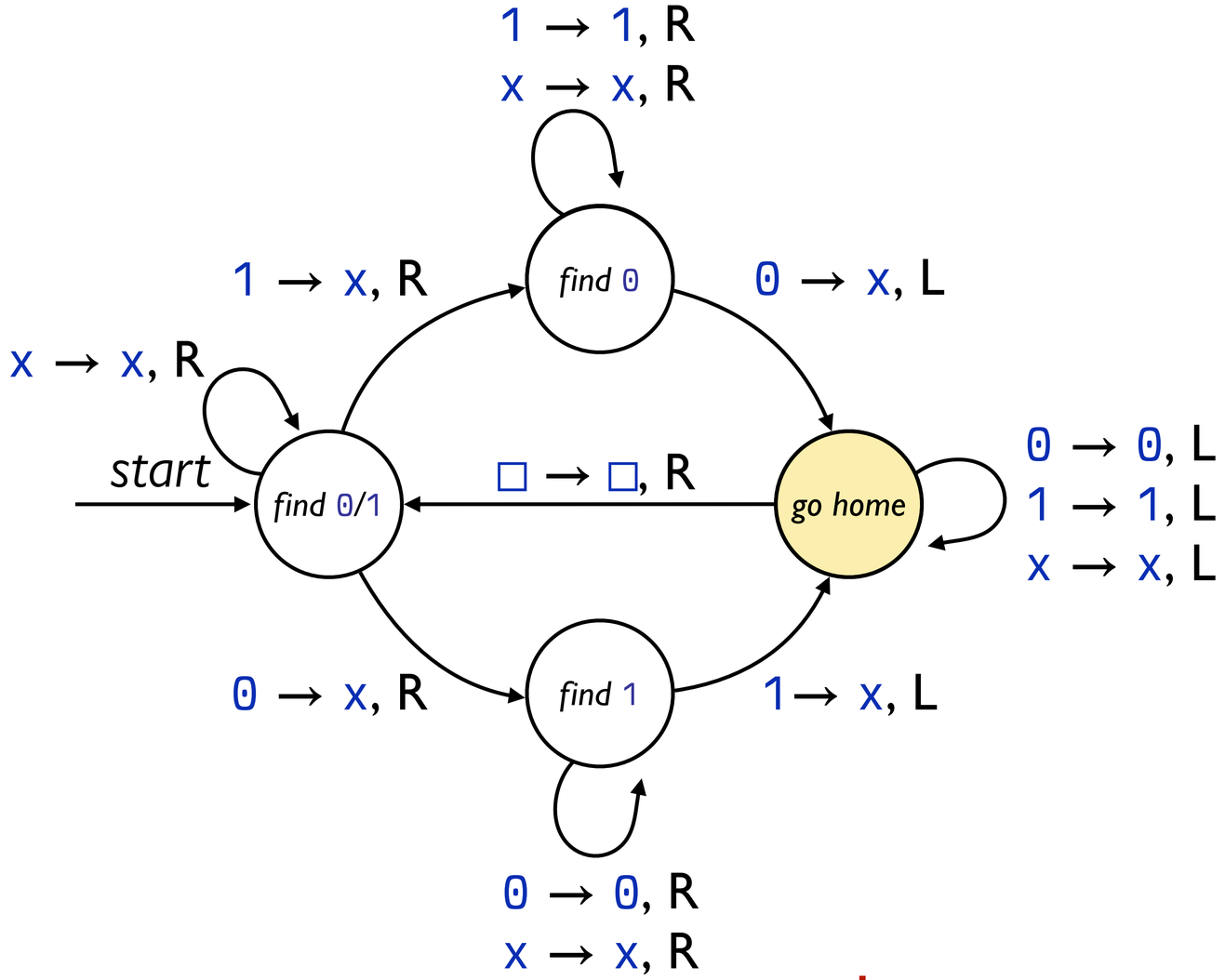

x x x x x 1 0 0

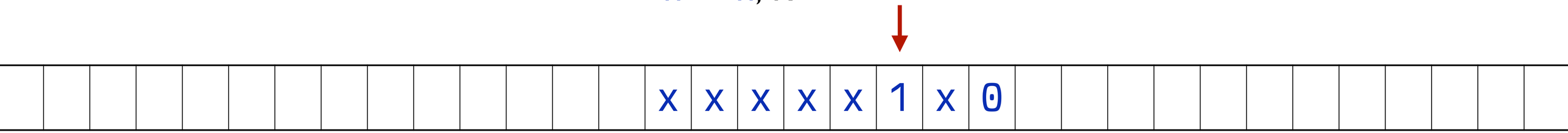

1 → 1, R
x → x, R
find 0
1 → x, R
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

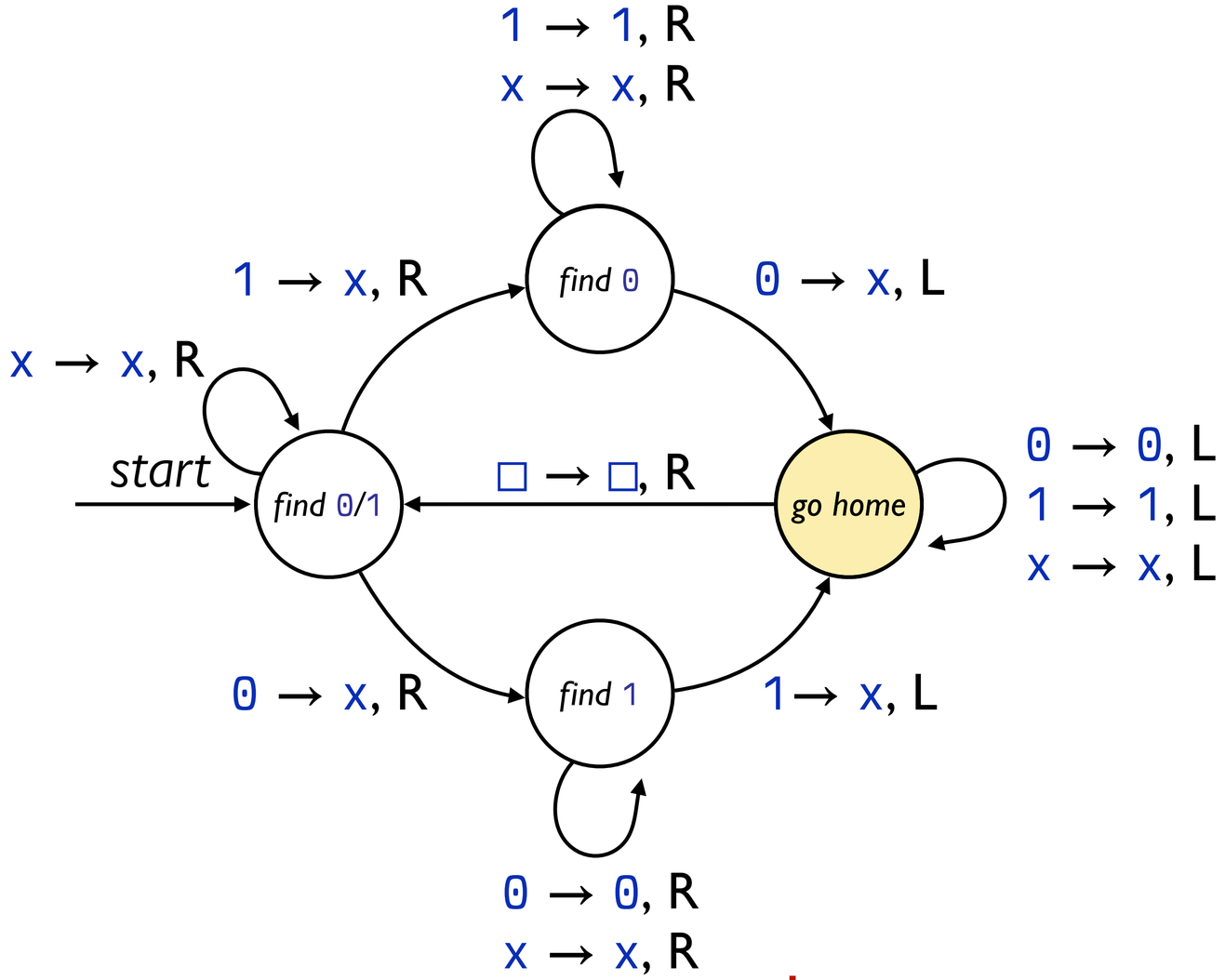

x x x x x 1 0 0

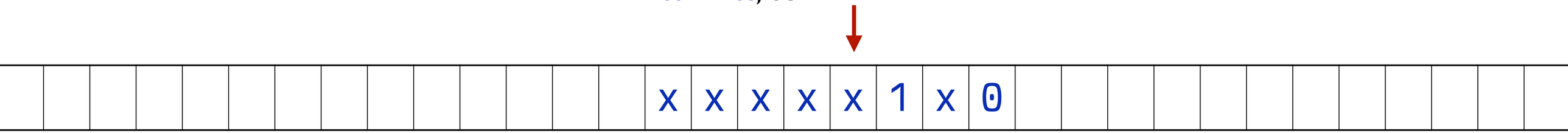

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

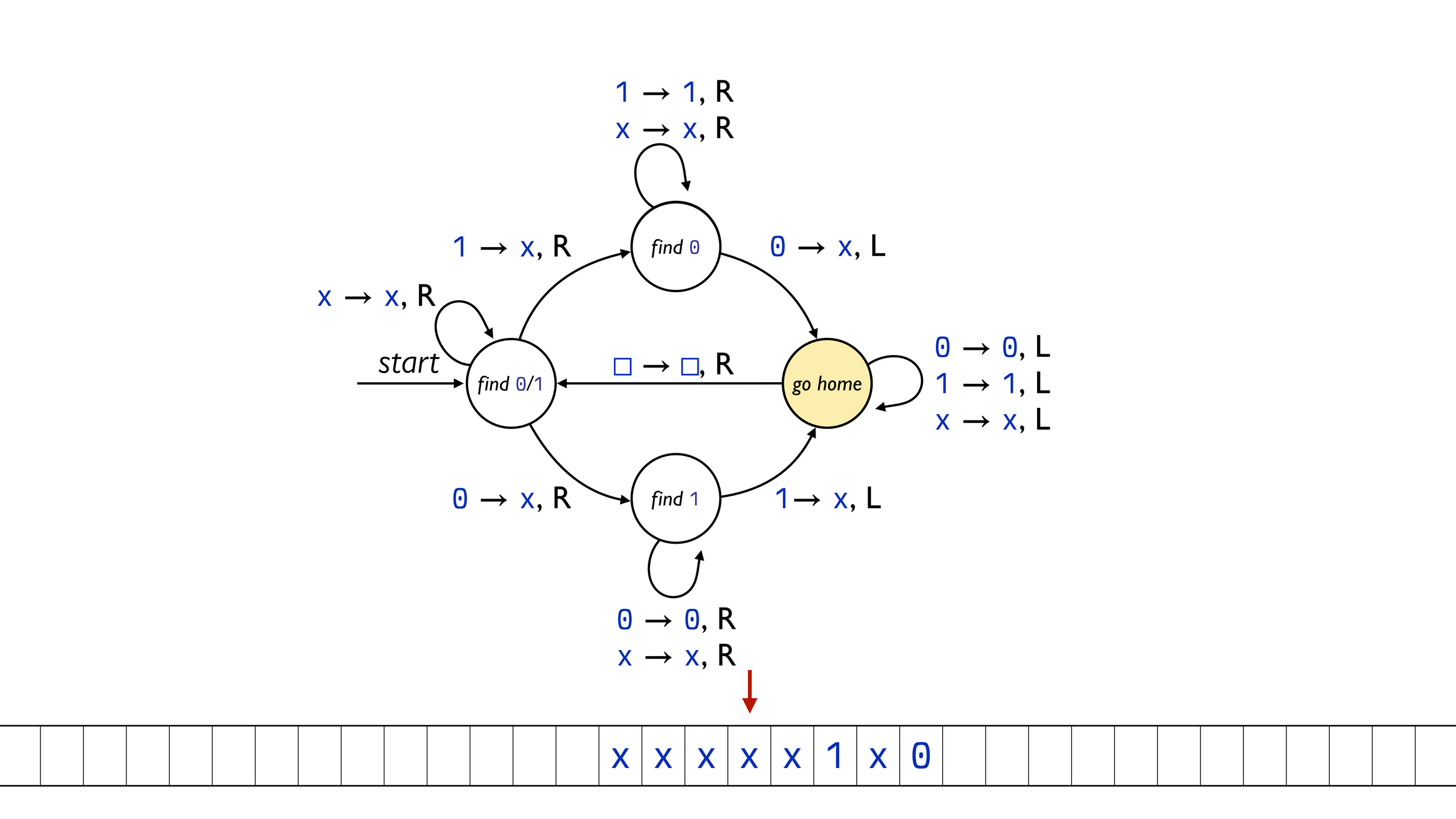

x x x x x 1 0 0

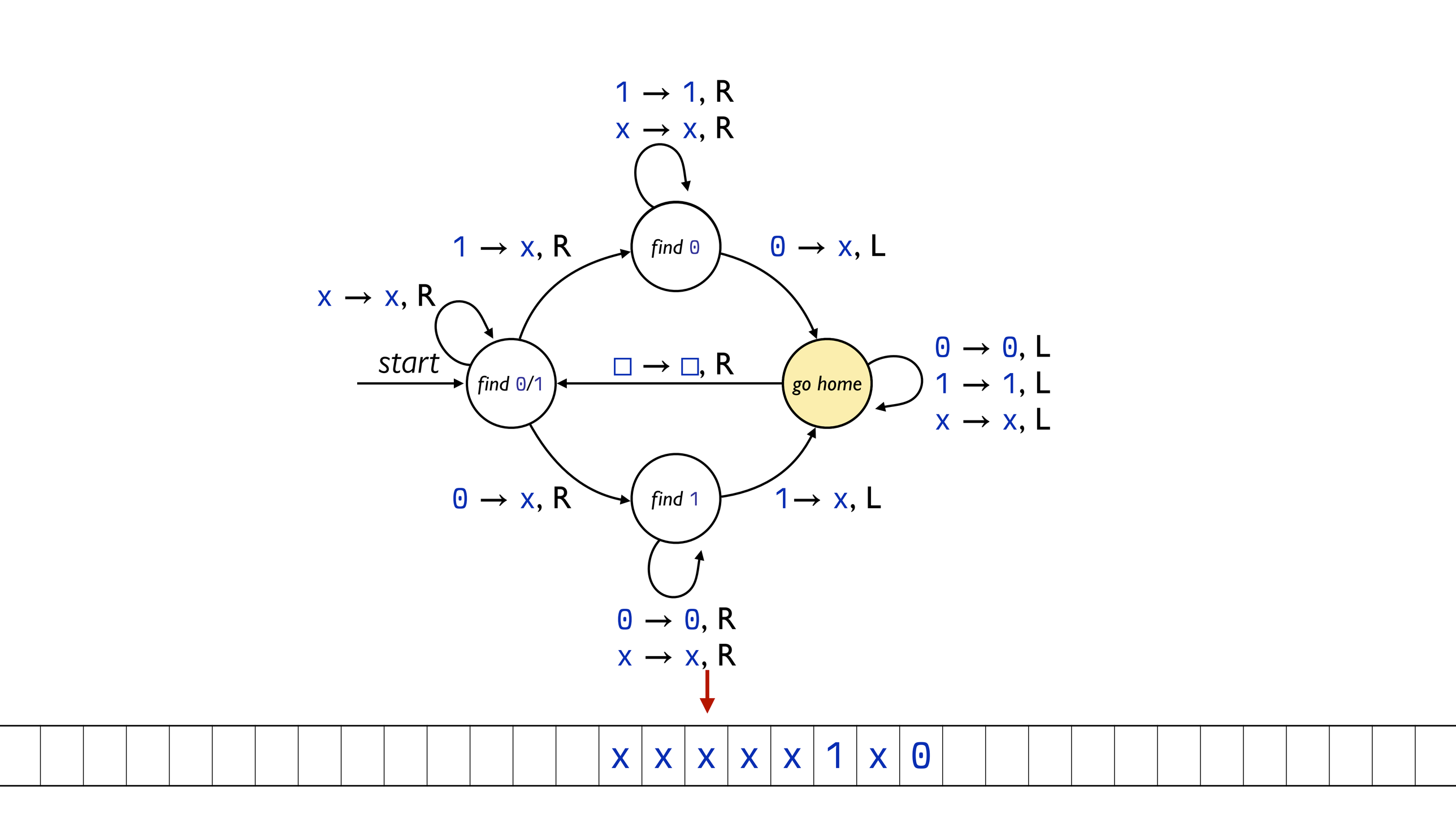

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

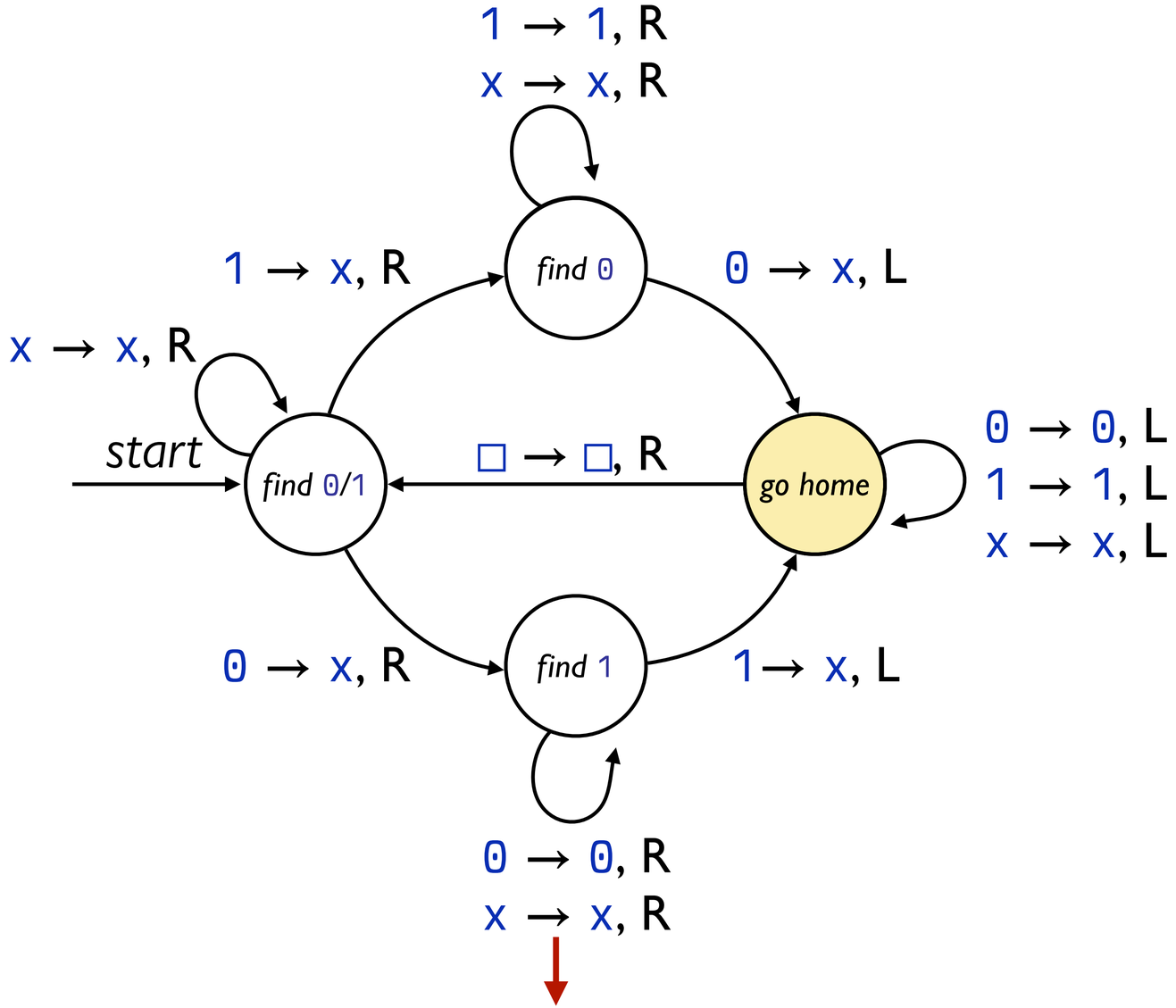

x x x x x 1 x 0

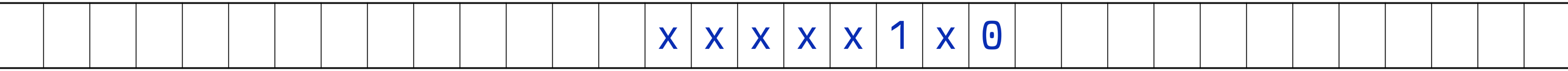

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

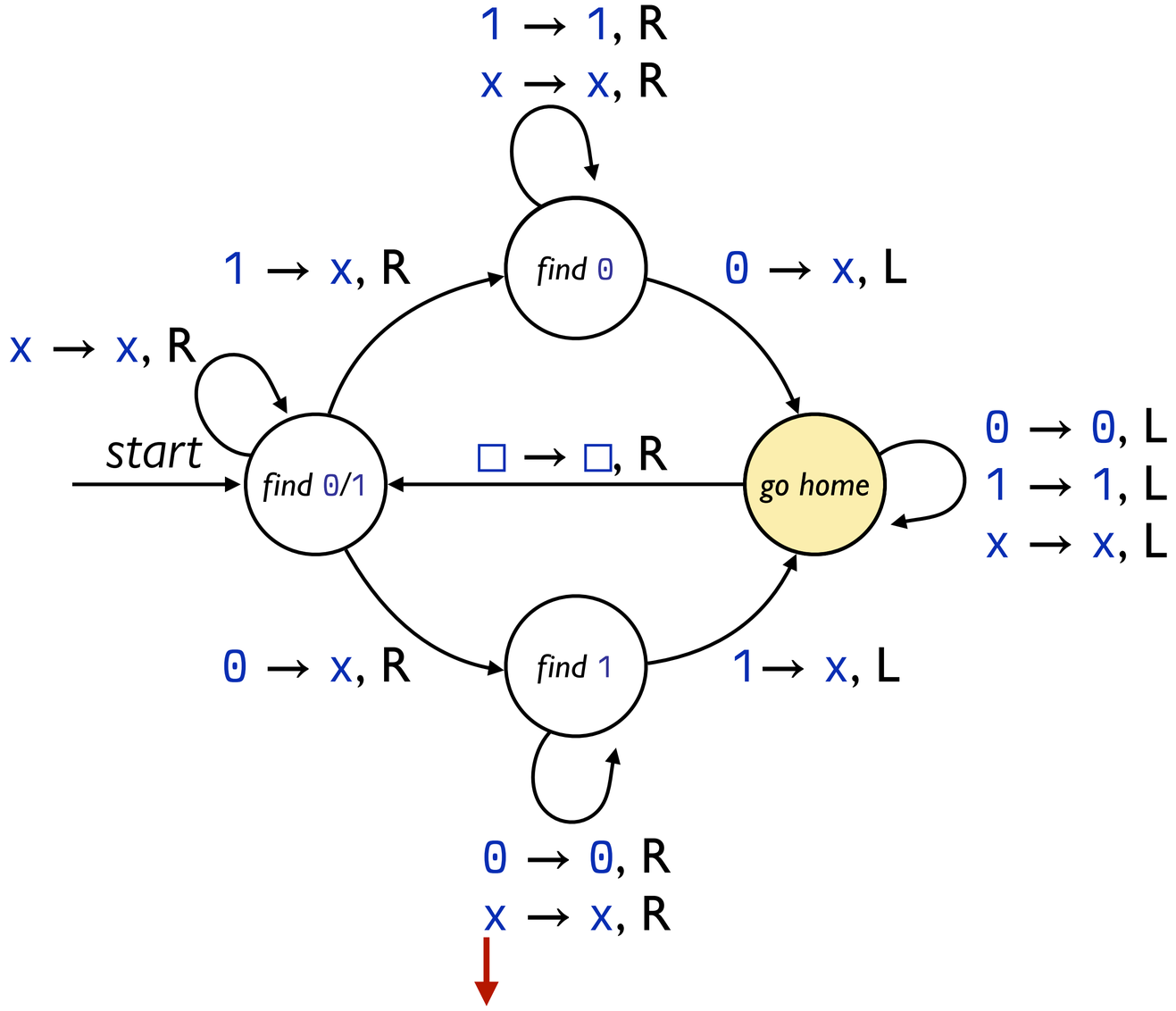

x x x x x 1 x 0

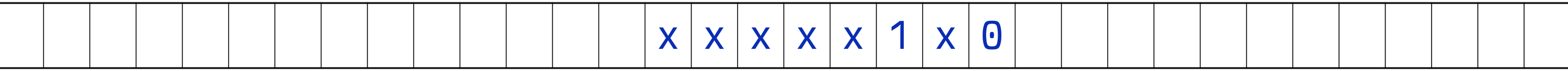

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R
x x x x x 1 x 0

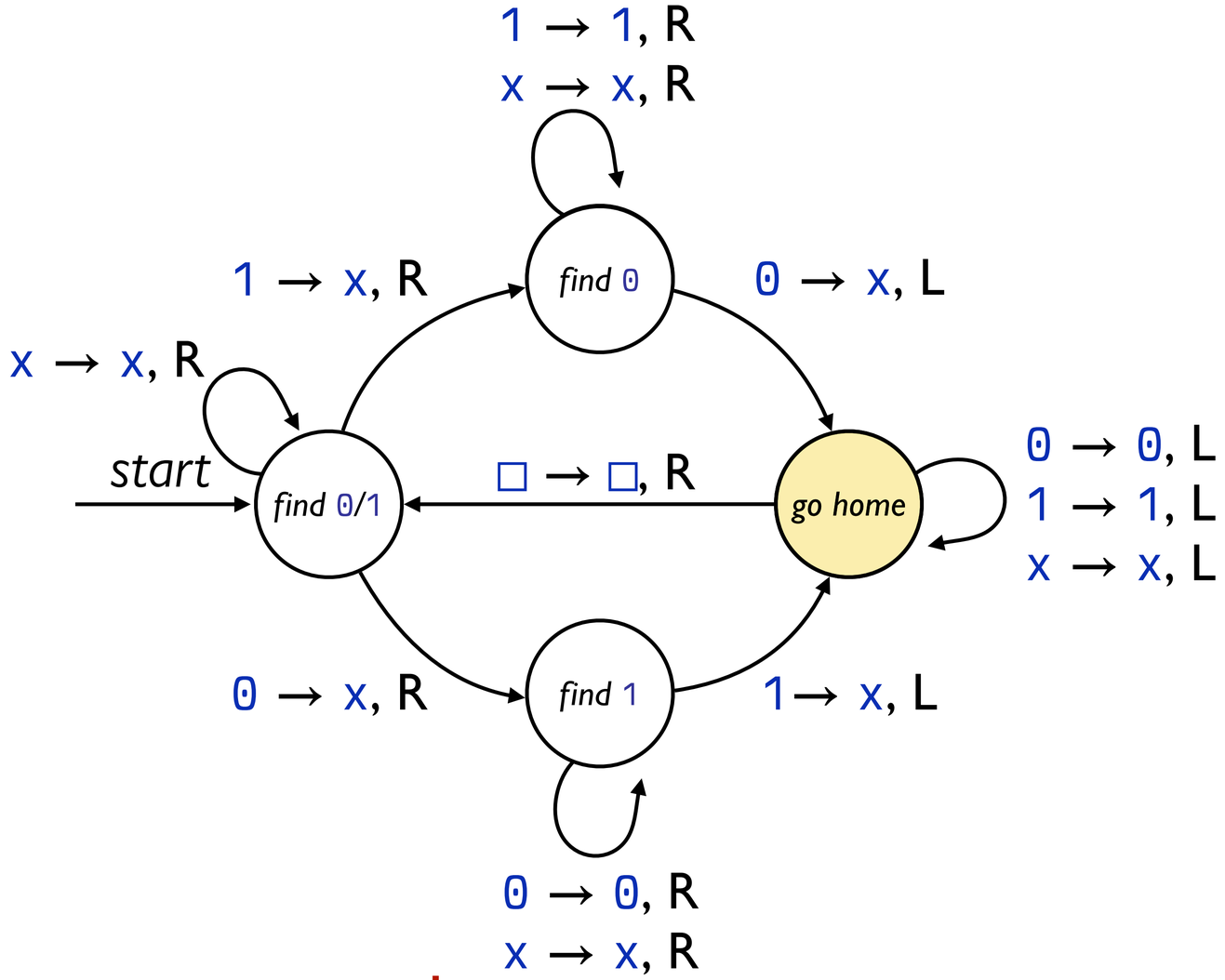

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R
x x x x x 1 x 0

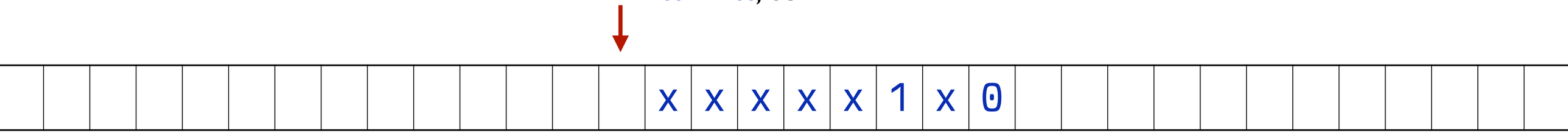

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

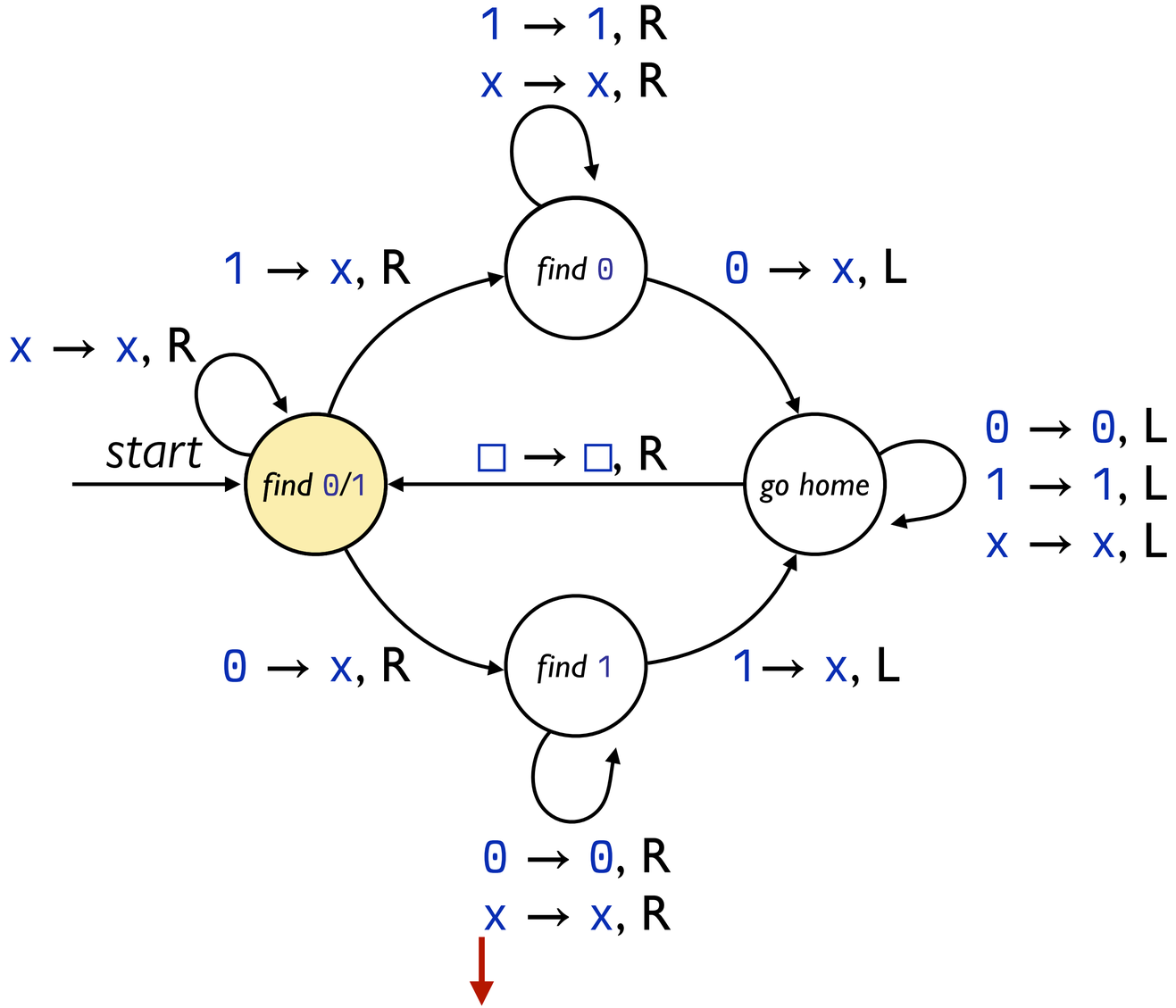

x x x x x 1 x 0

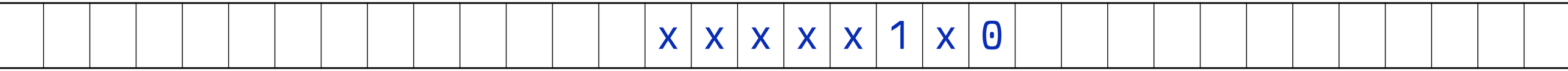

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

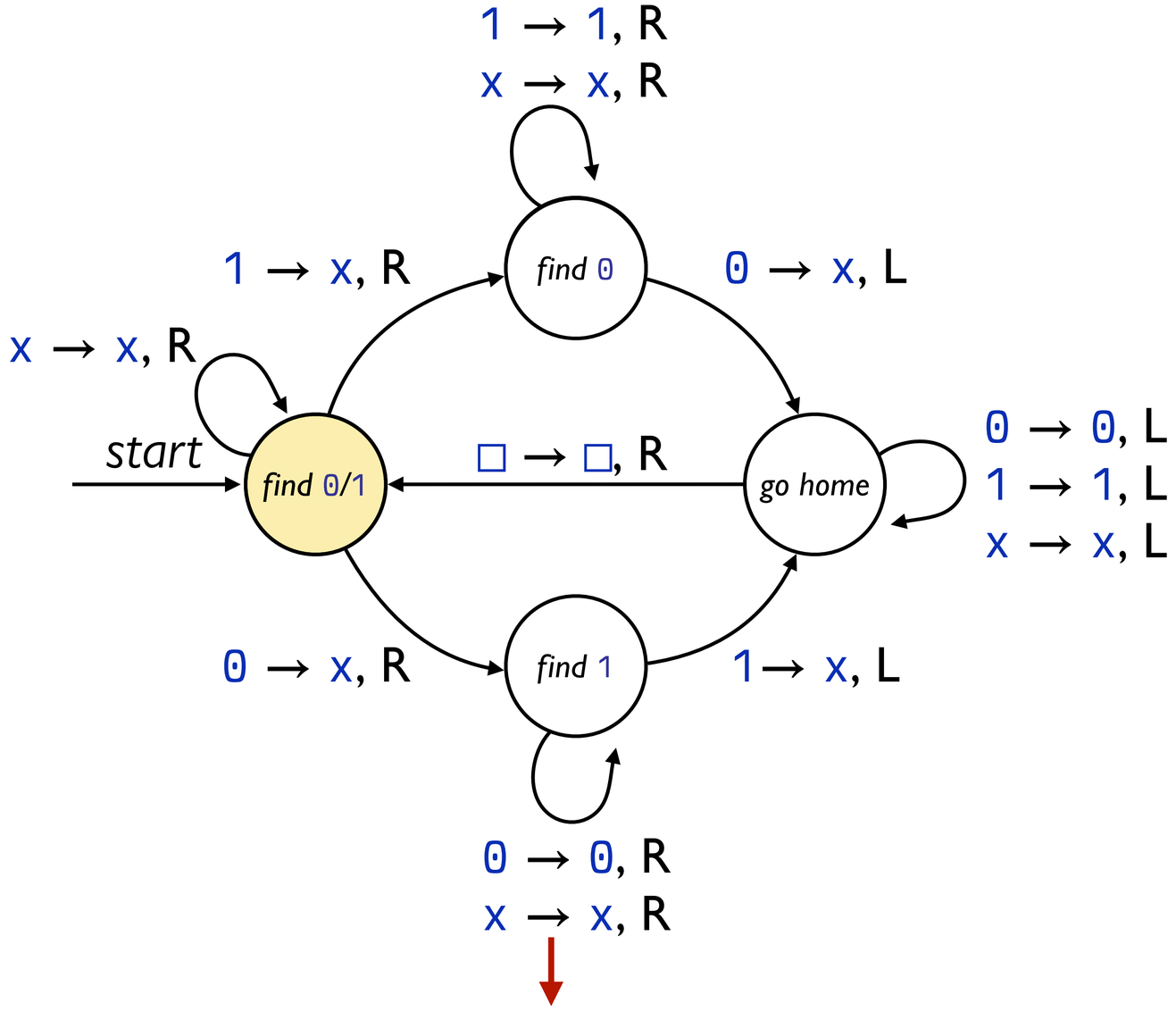

x x x x x 1 x 0

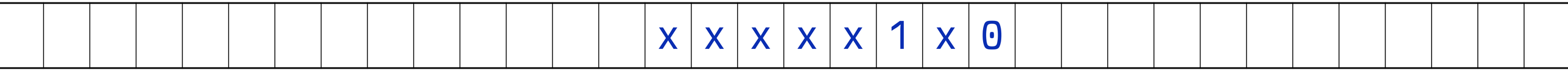

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

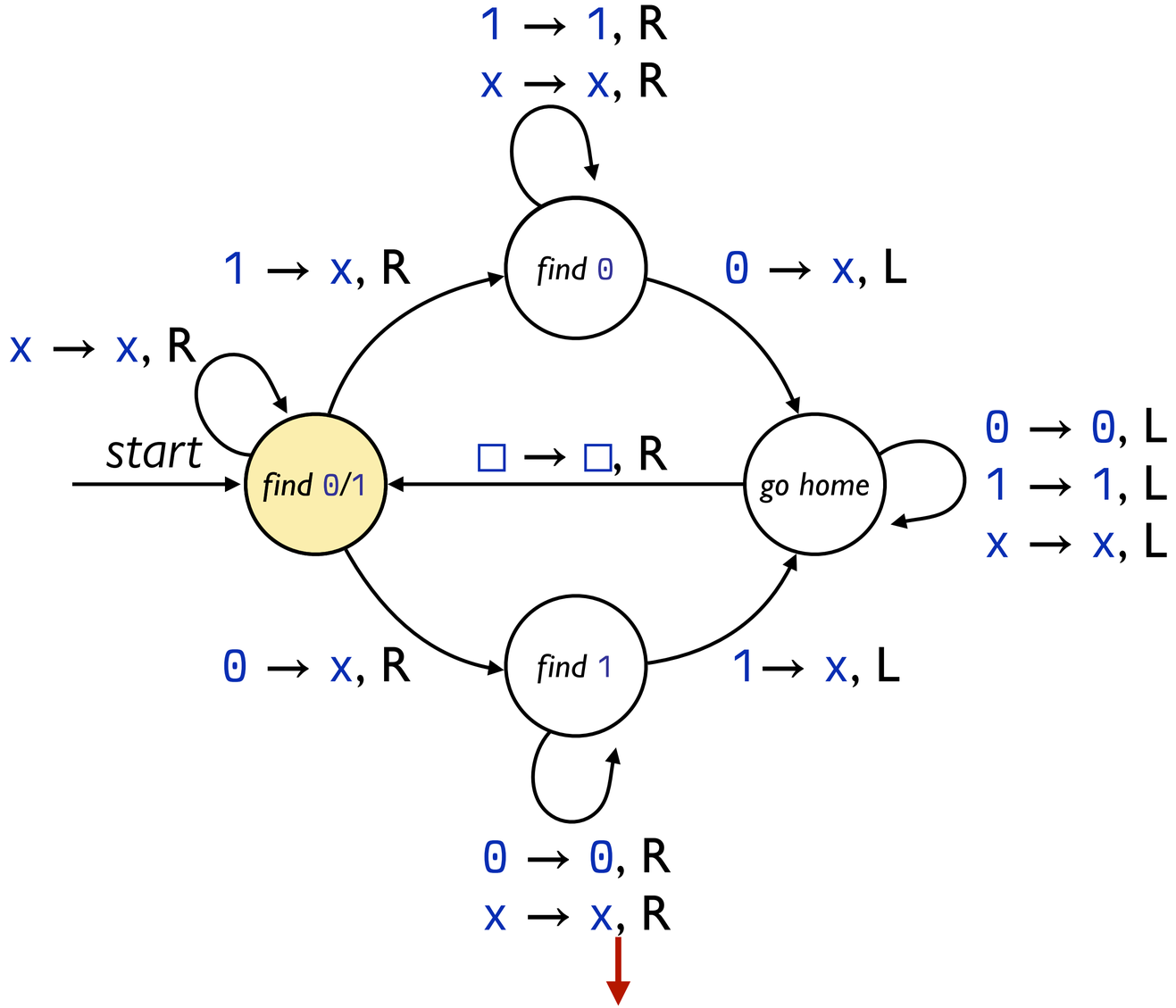

x x x x x 1 x 0

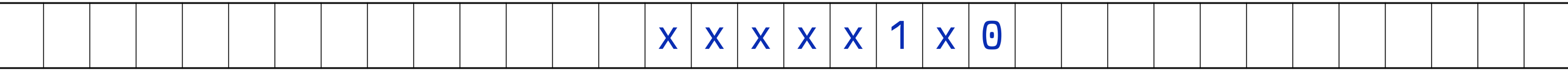

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

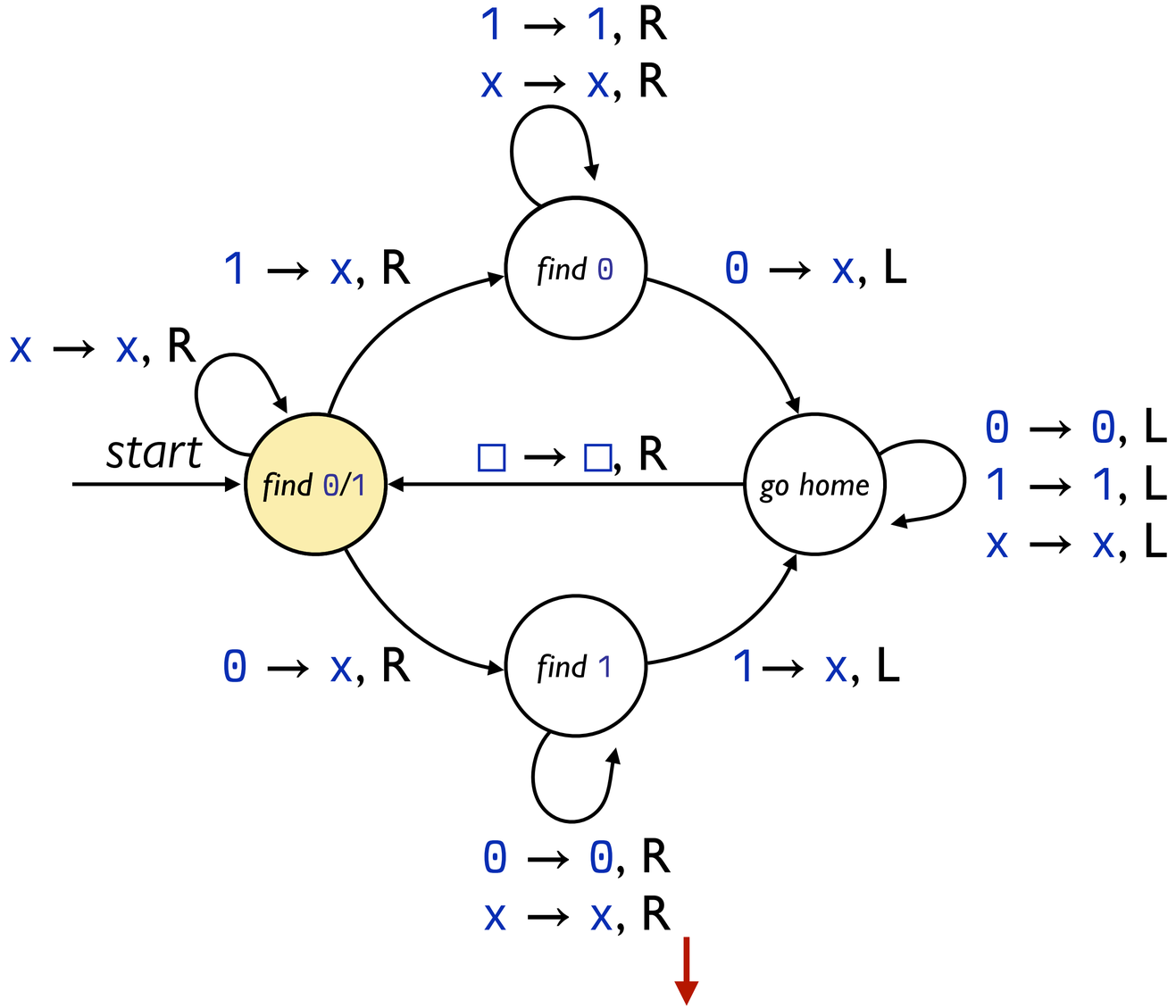

x x x x x 1 x 0

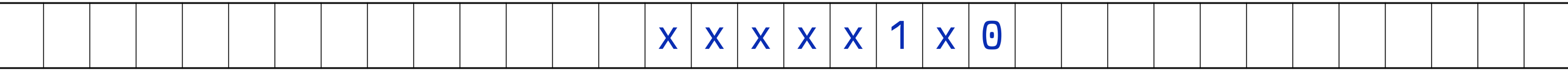

1 → 1, R
x → x, R
1 → x, R
0 → x, L
x → x, R
start
find 0/1
find 0
go home
□ → □, R
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

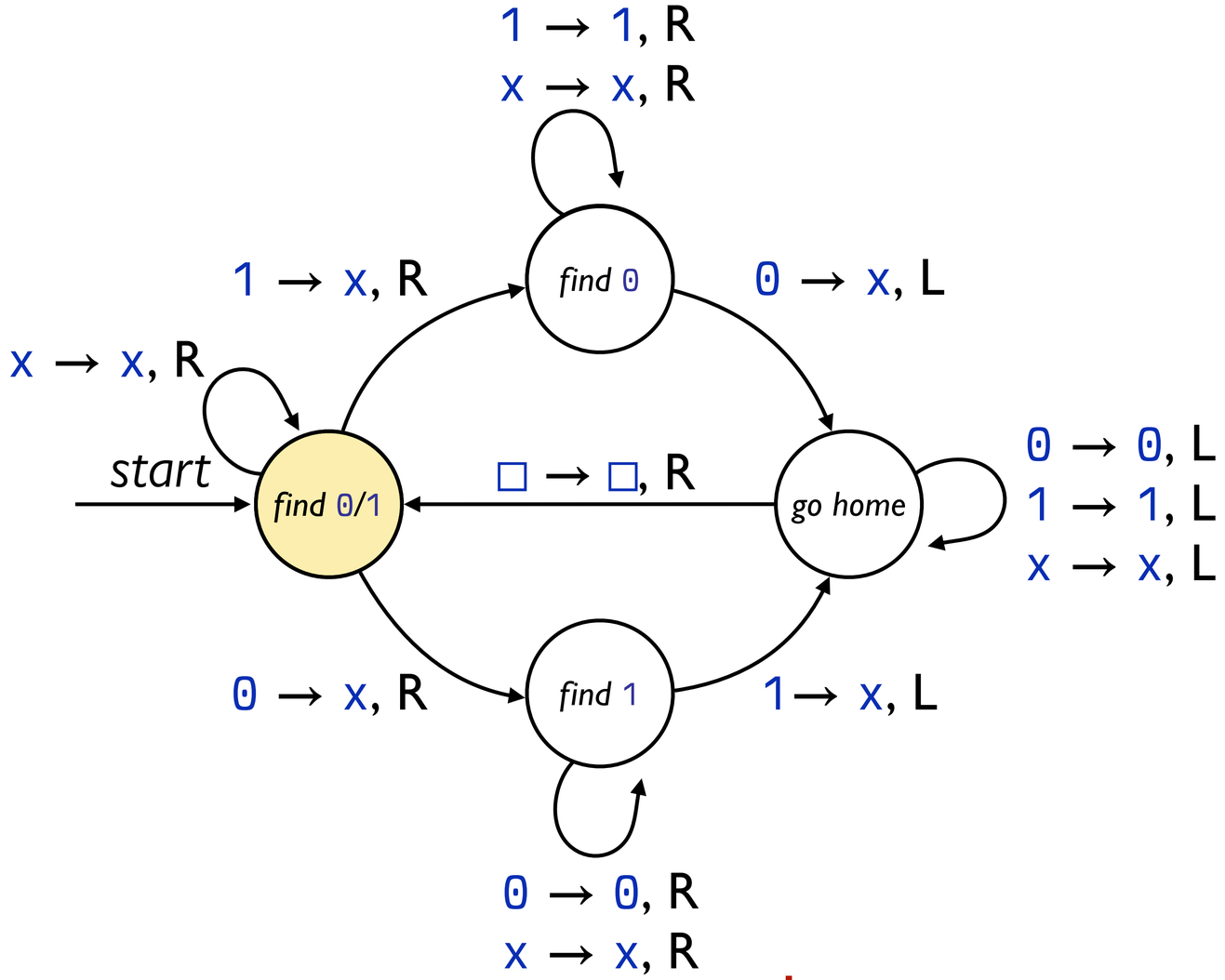

x x x x x 1 x 0

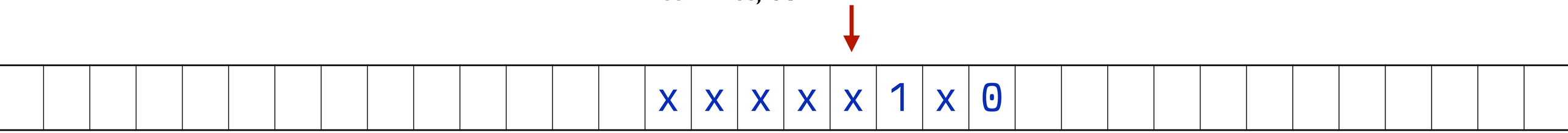

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

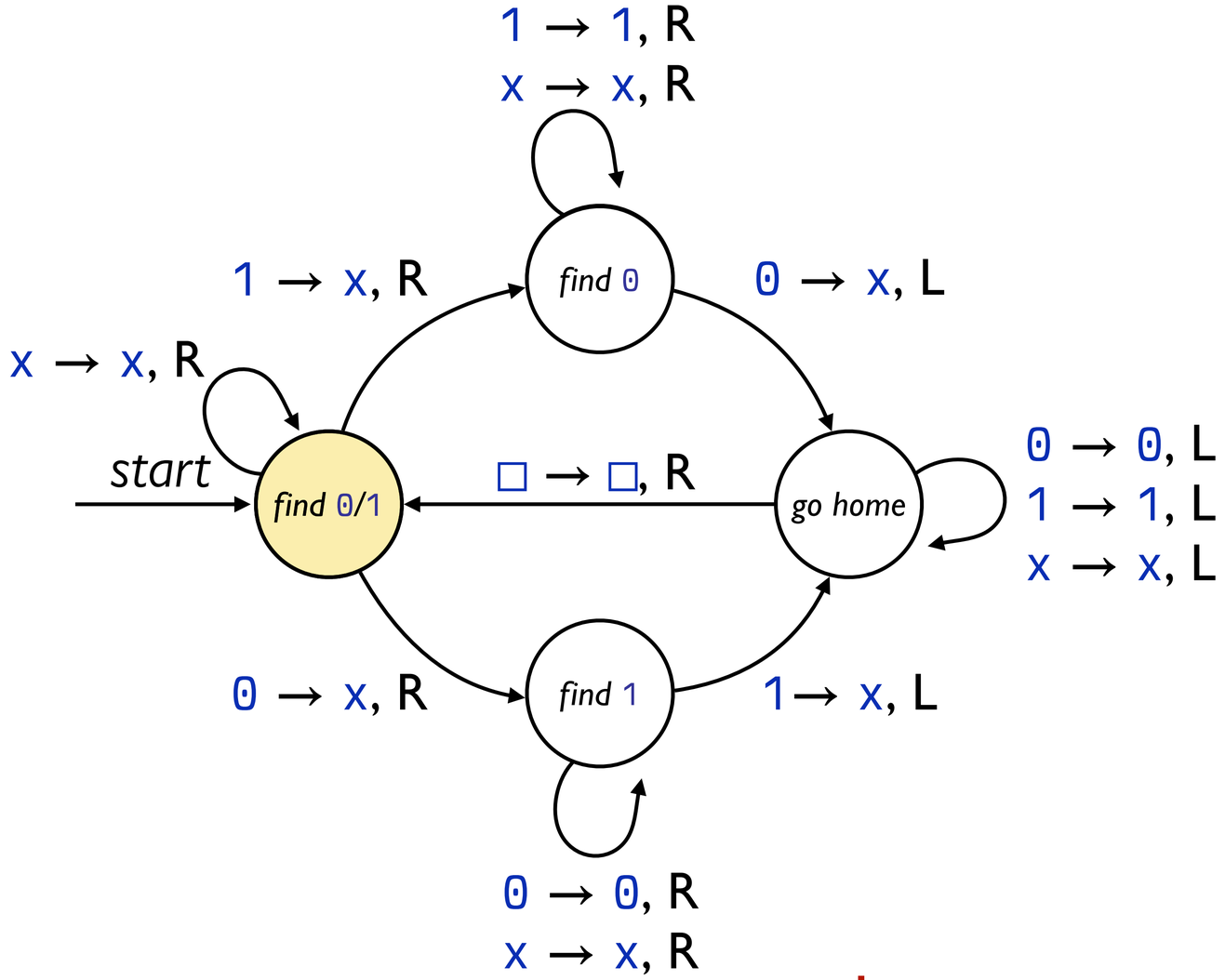

x x x x x 1 x 0

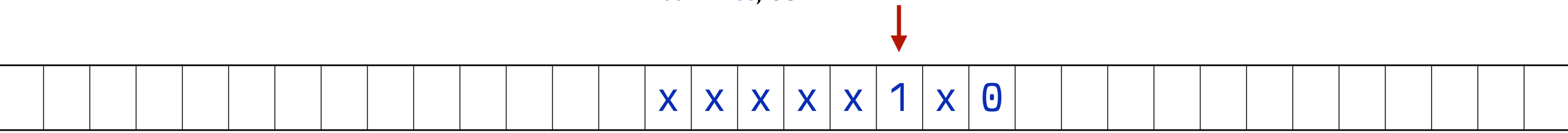

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

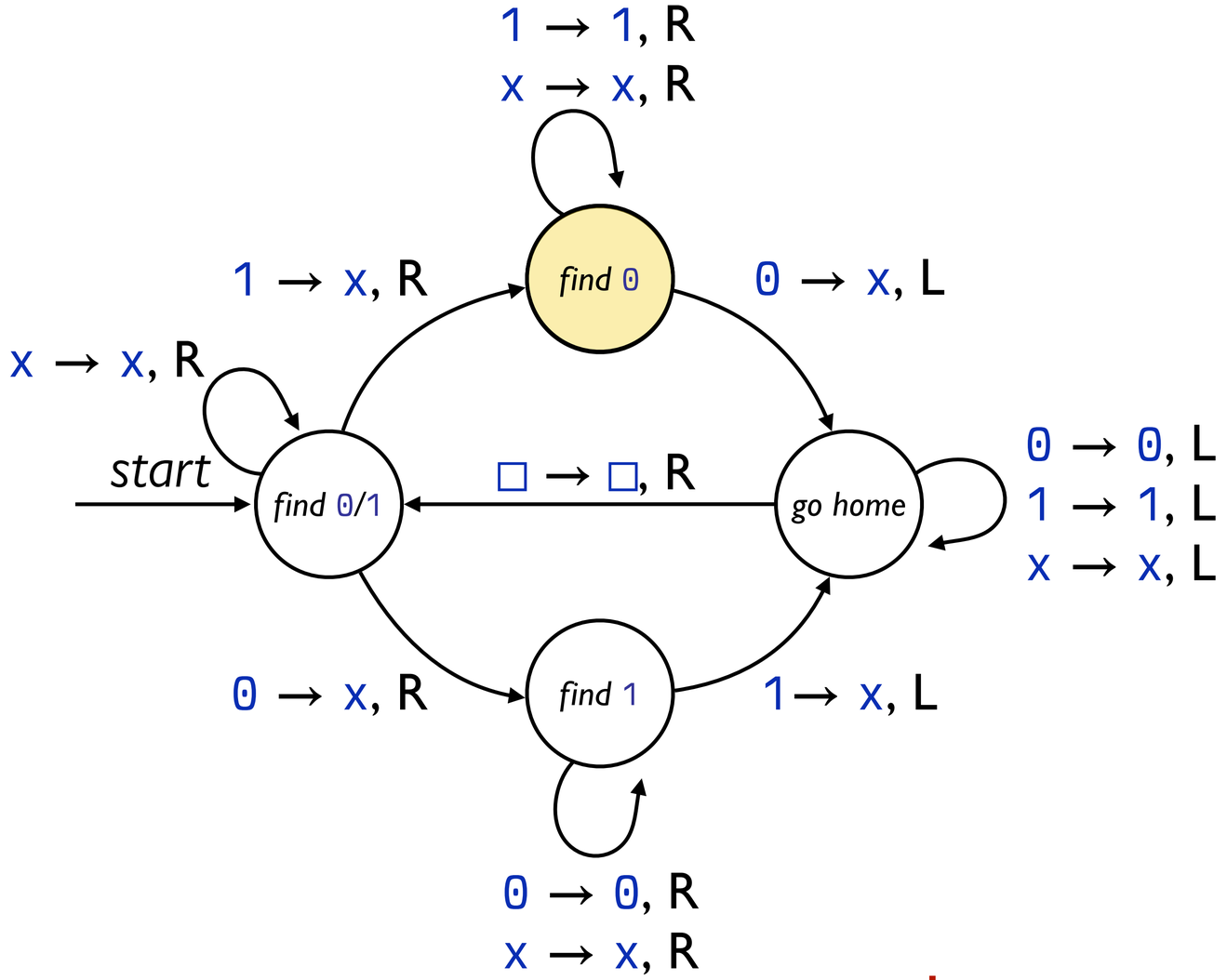

x x x x x 1 x 0

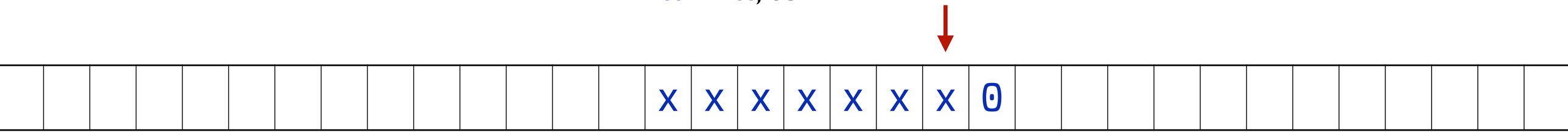

1 → 1, R
x → x, R
1 → x, R
find 0
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

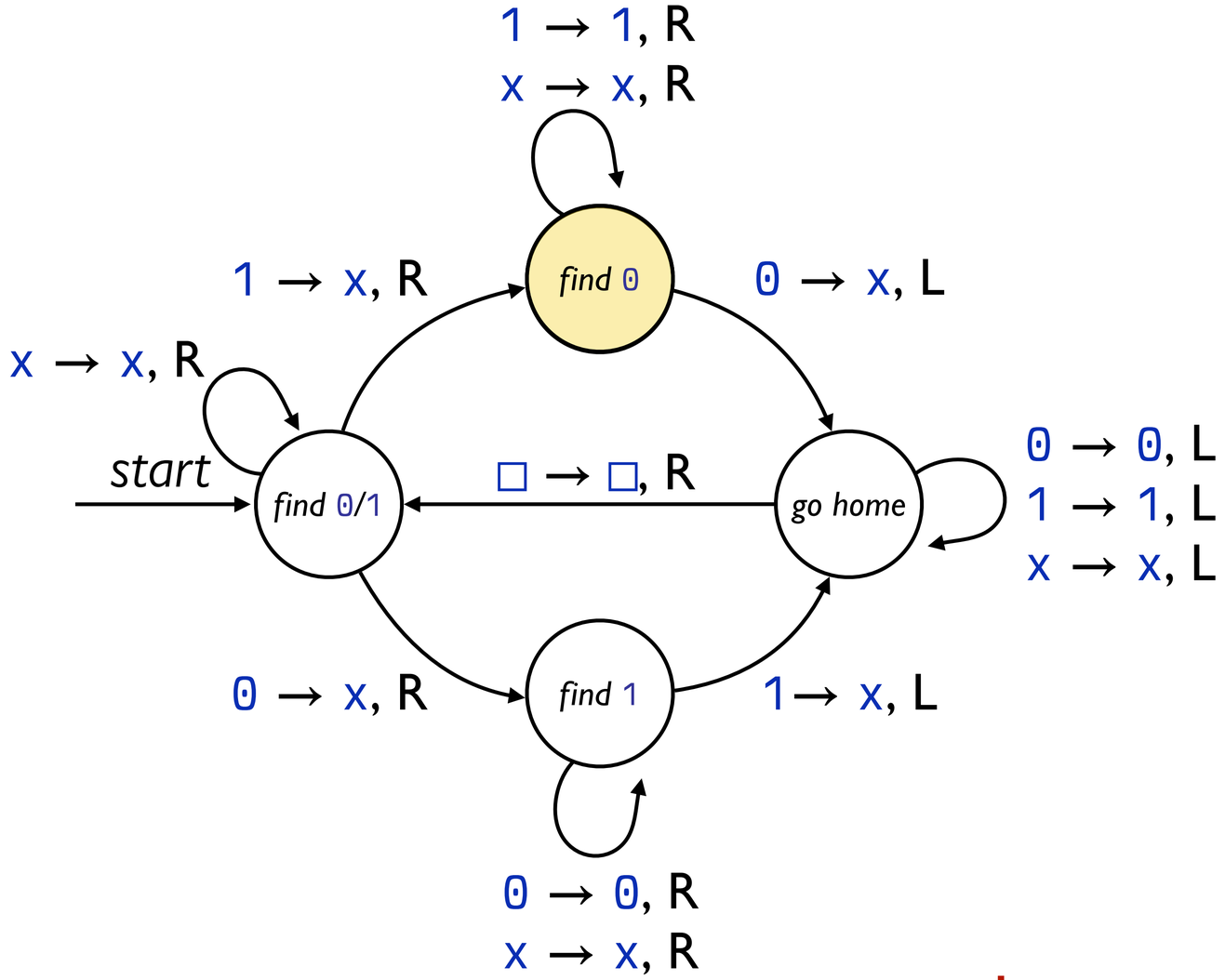

x x x x x 1 x 0

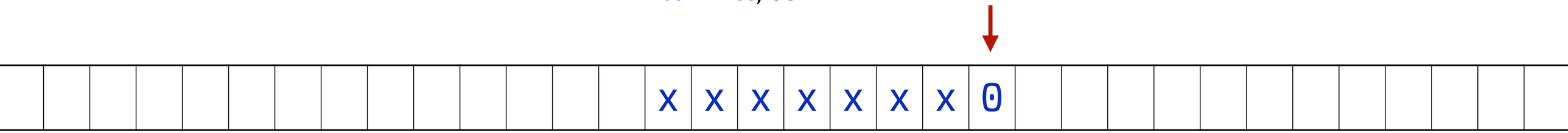

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1→ x, L
0 → 0, R
x → x, R

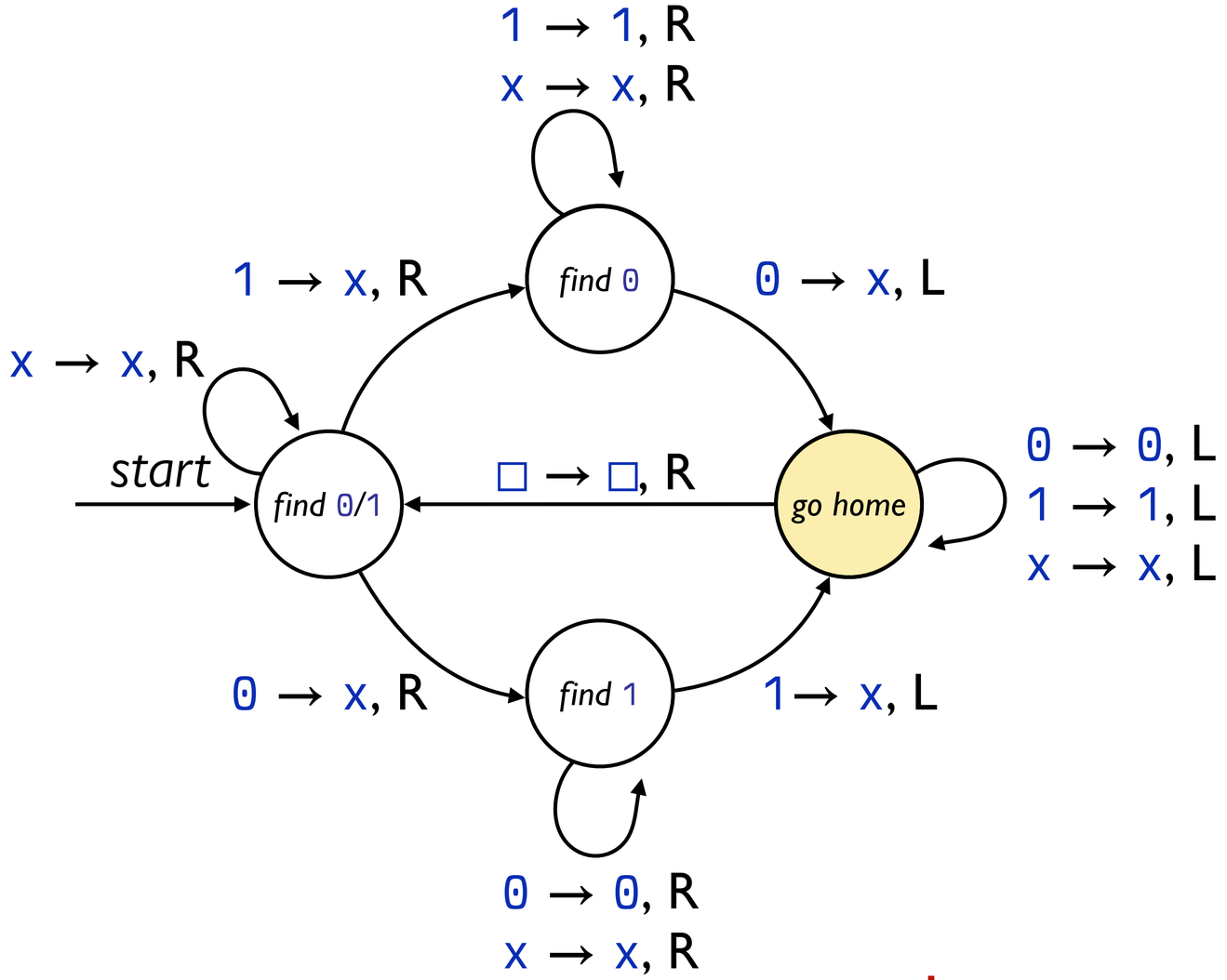

x x x x x 1 x 0

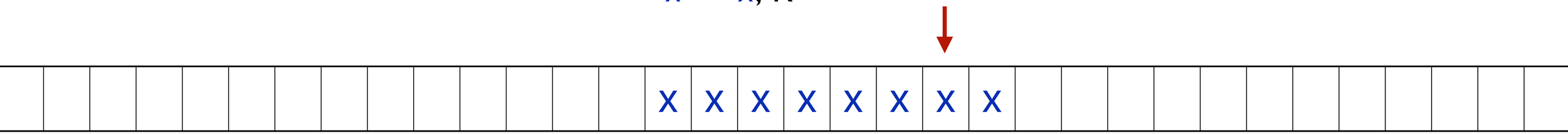

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

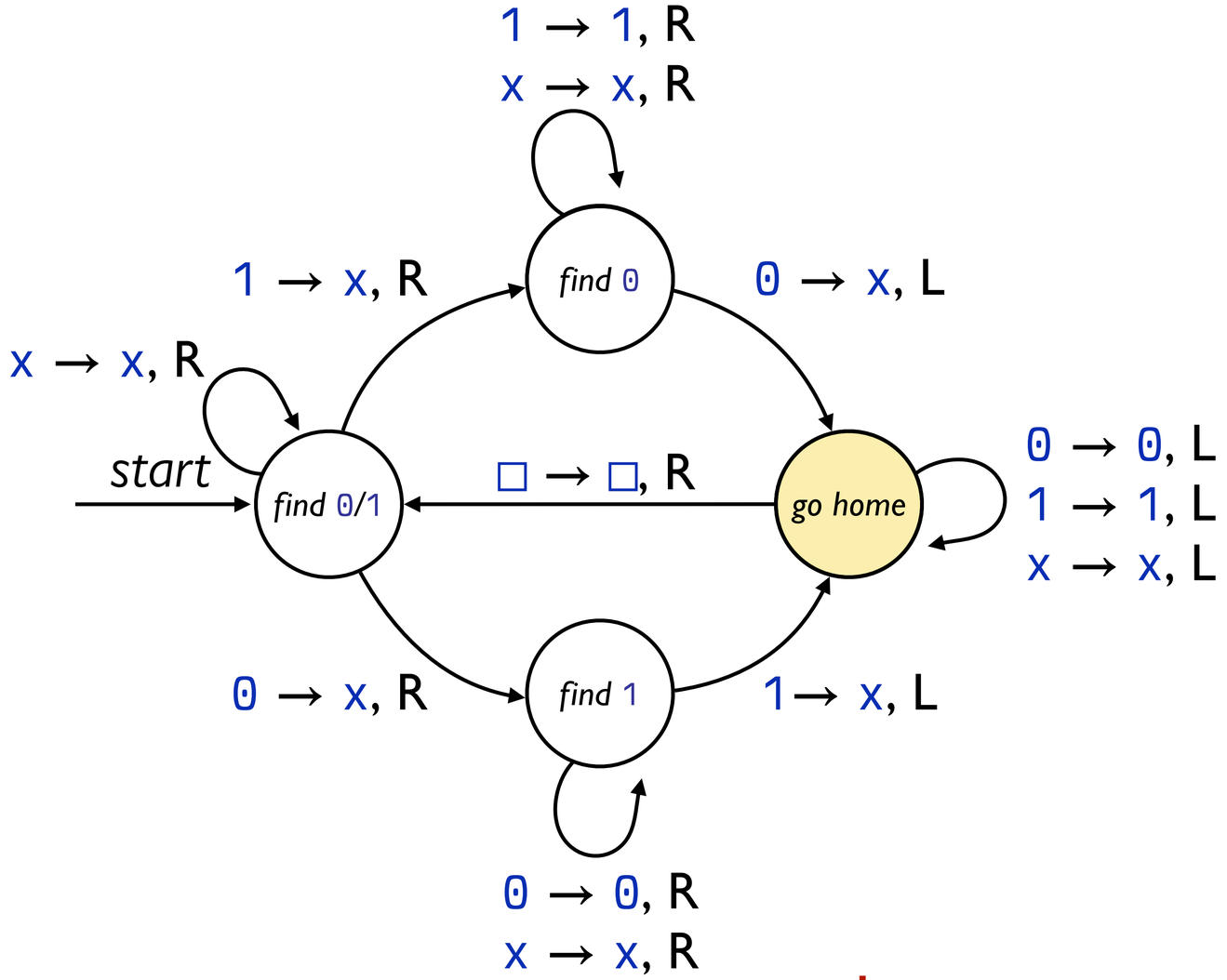

x x x x x x x 0

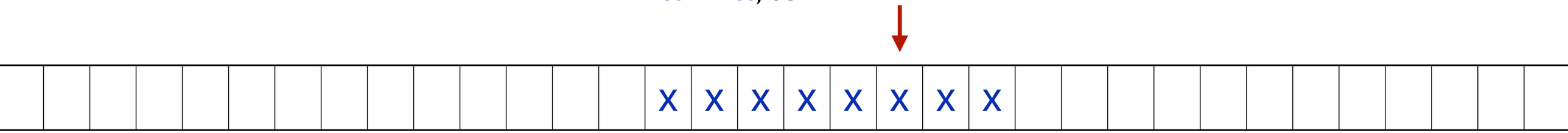

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

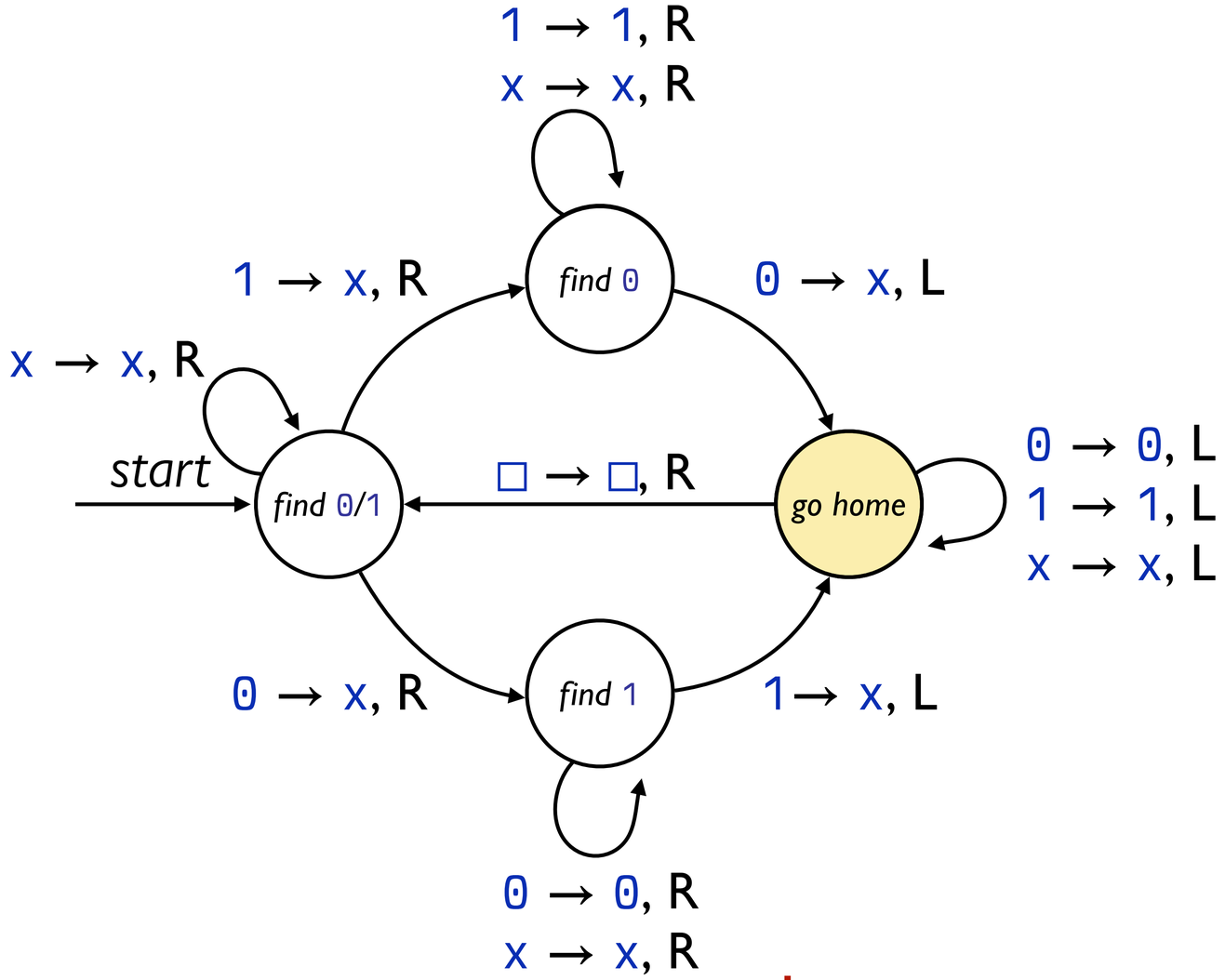

x x x x x x x 0

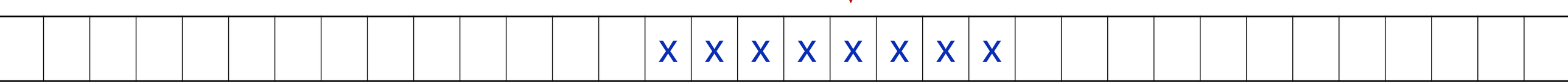

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

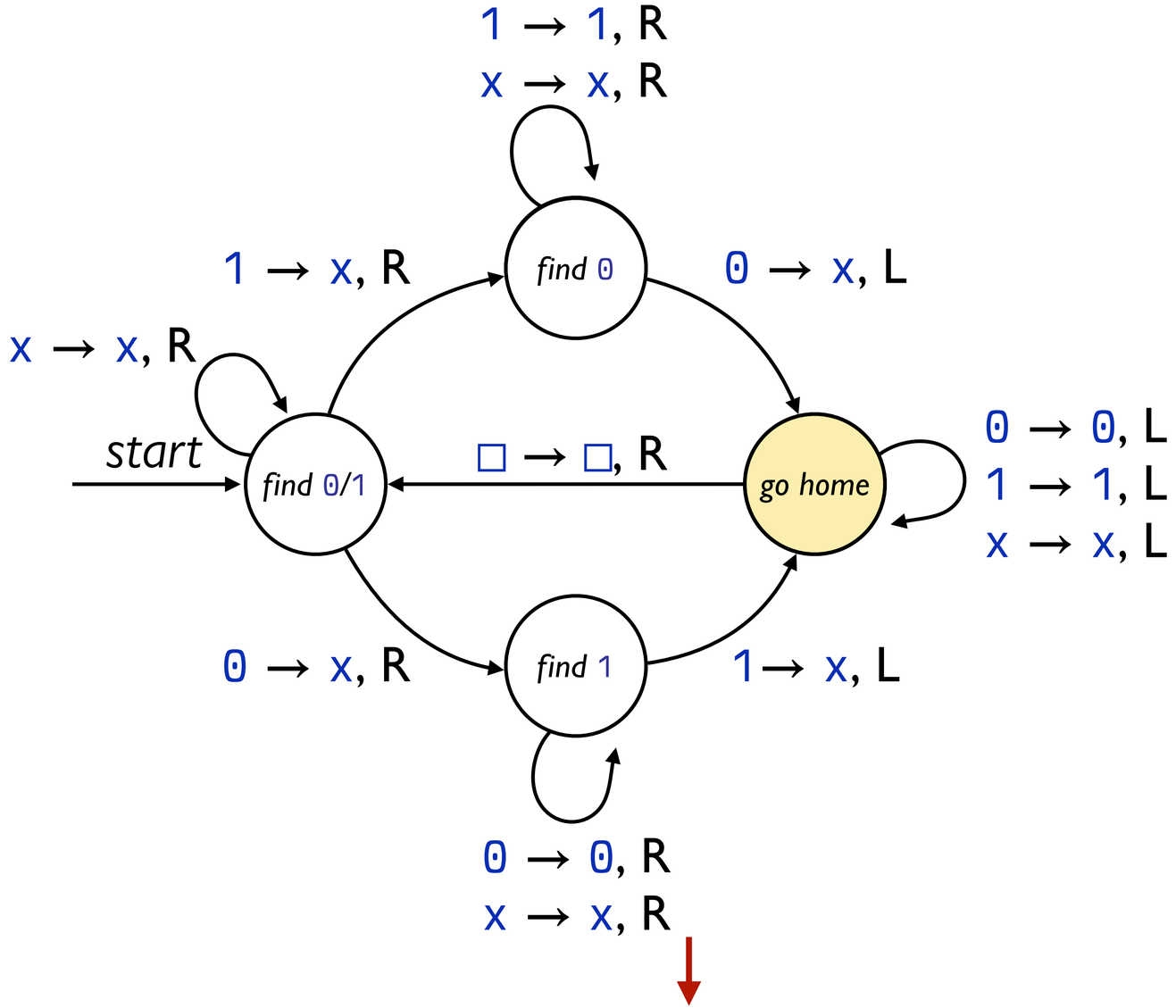

x x x x x x x x

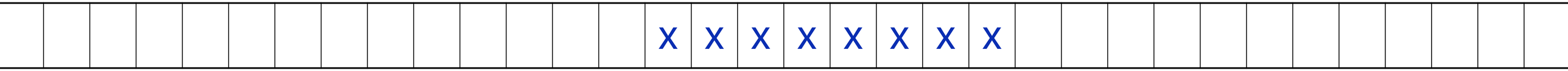

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

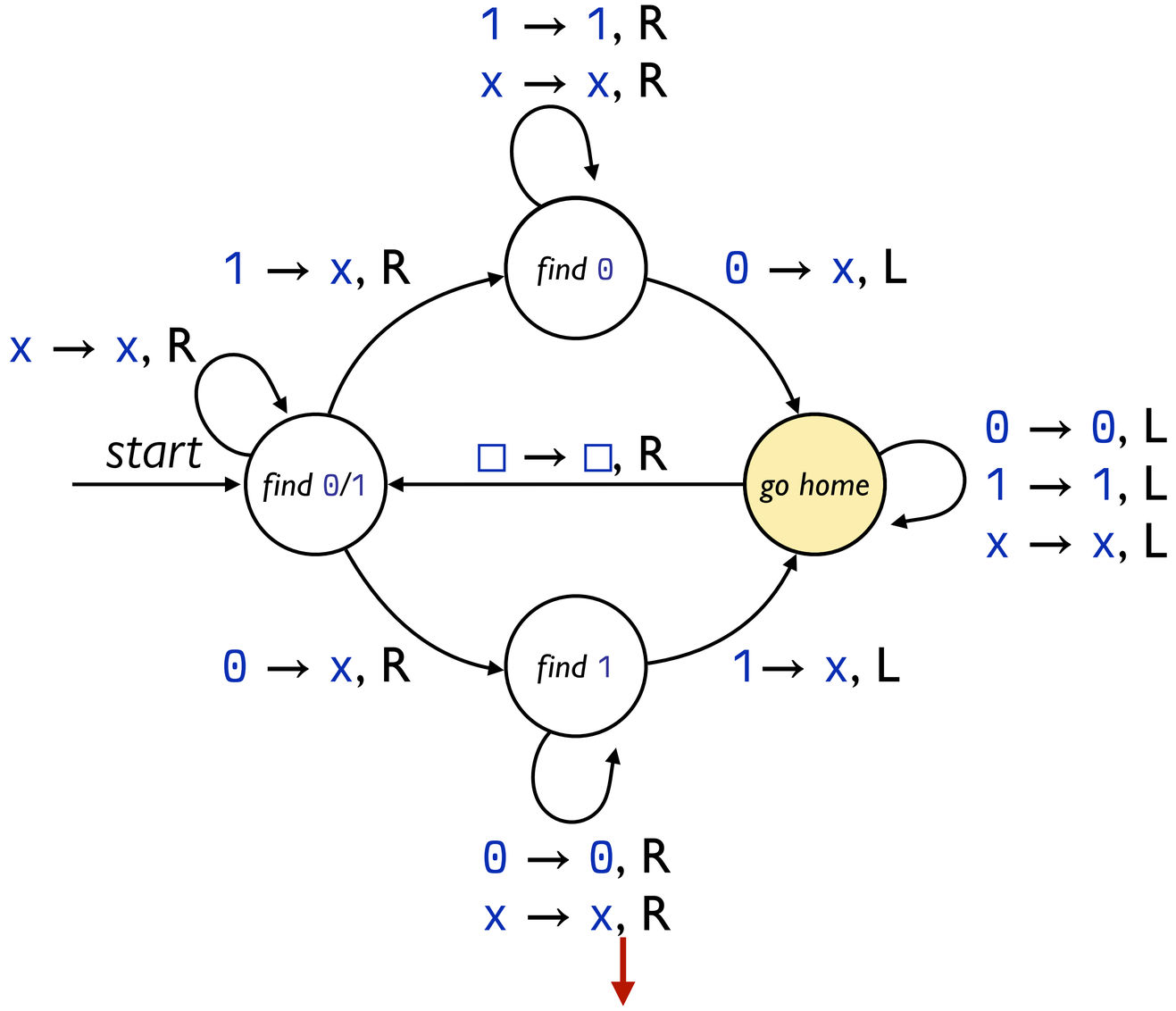

x x x x x x x x

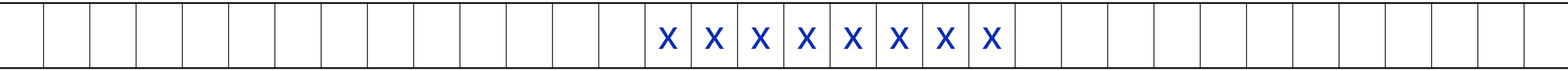

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

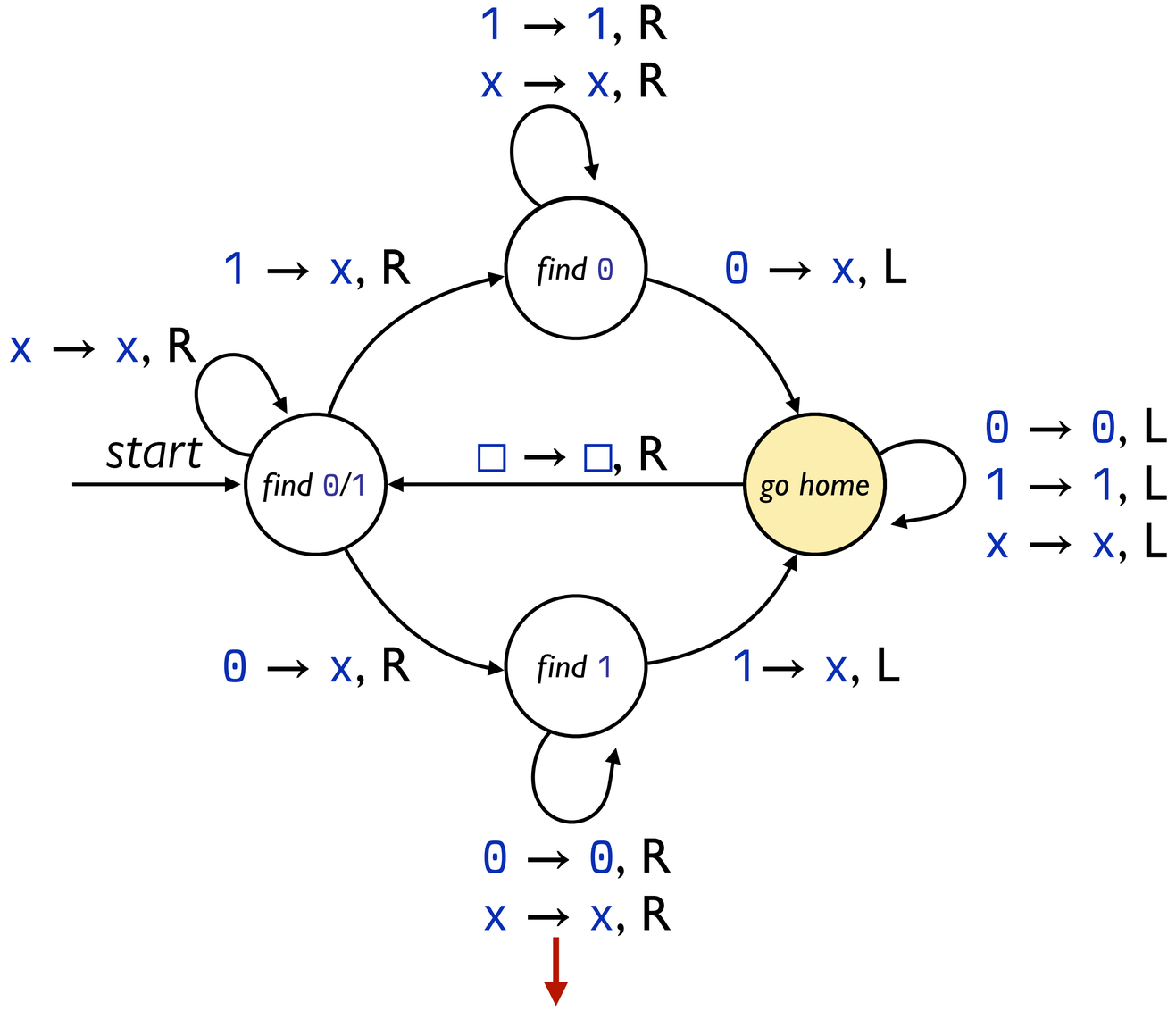

x x x x x x x x

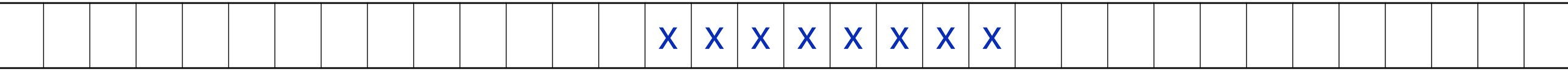

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

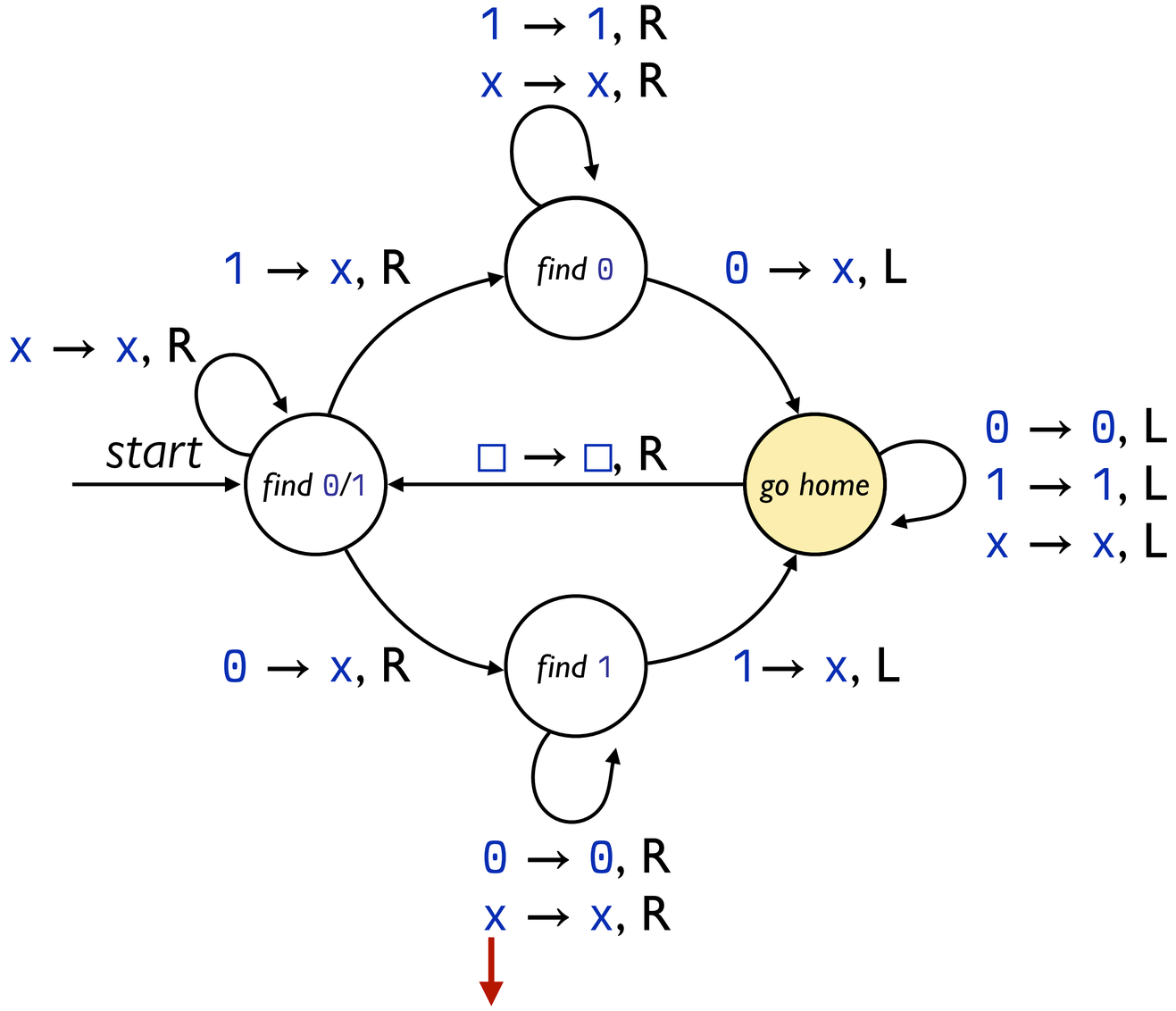

x x x x x x x x

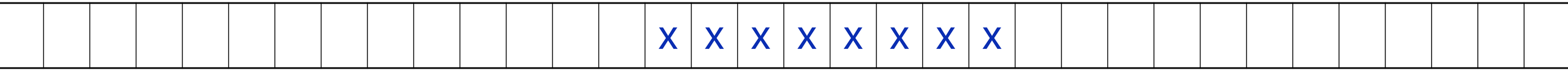

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

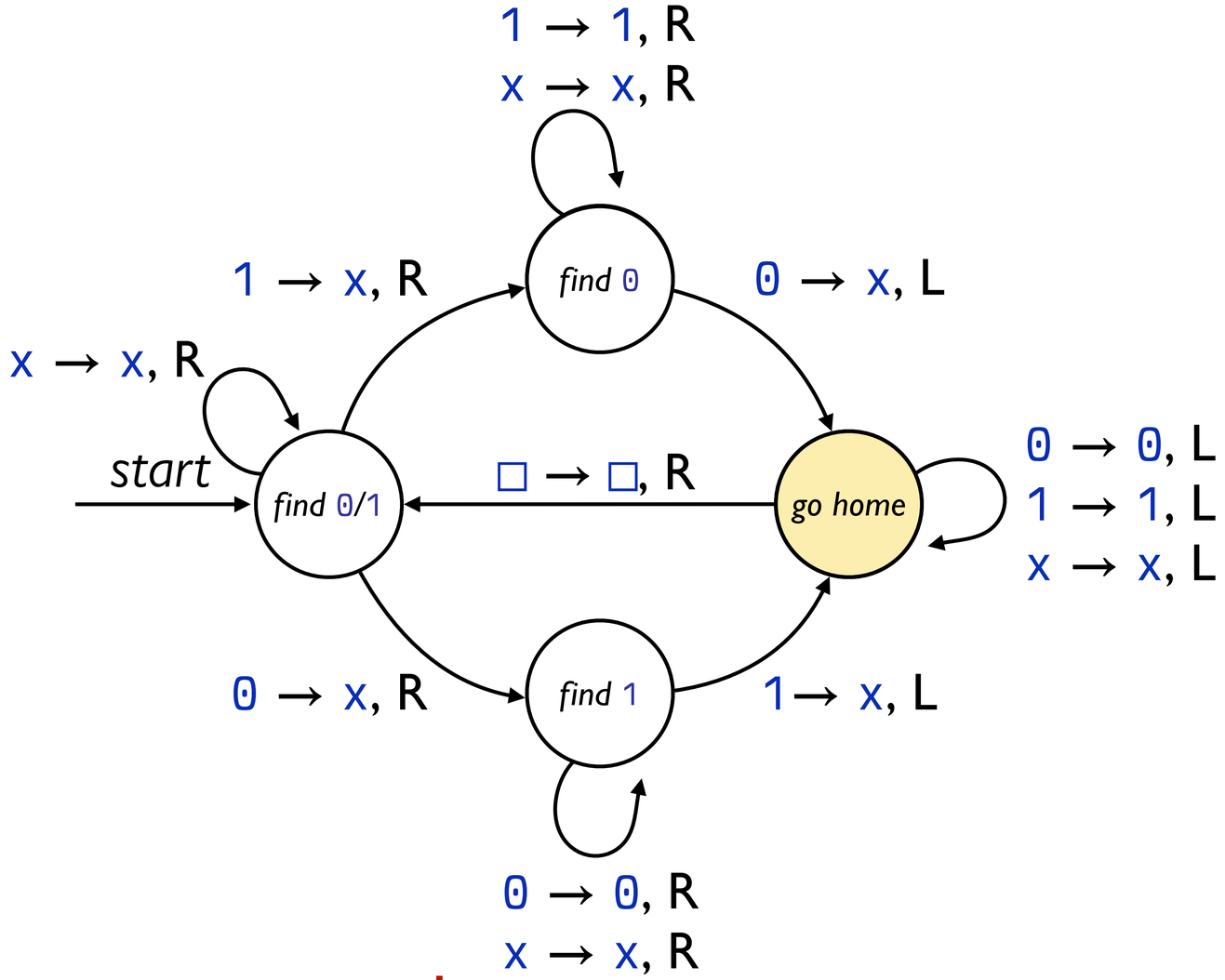

x x x x x x x x

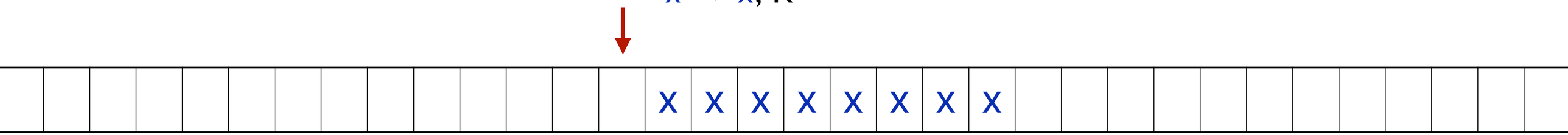

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

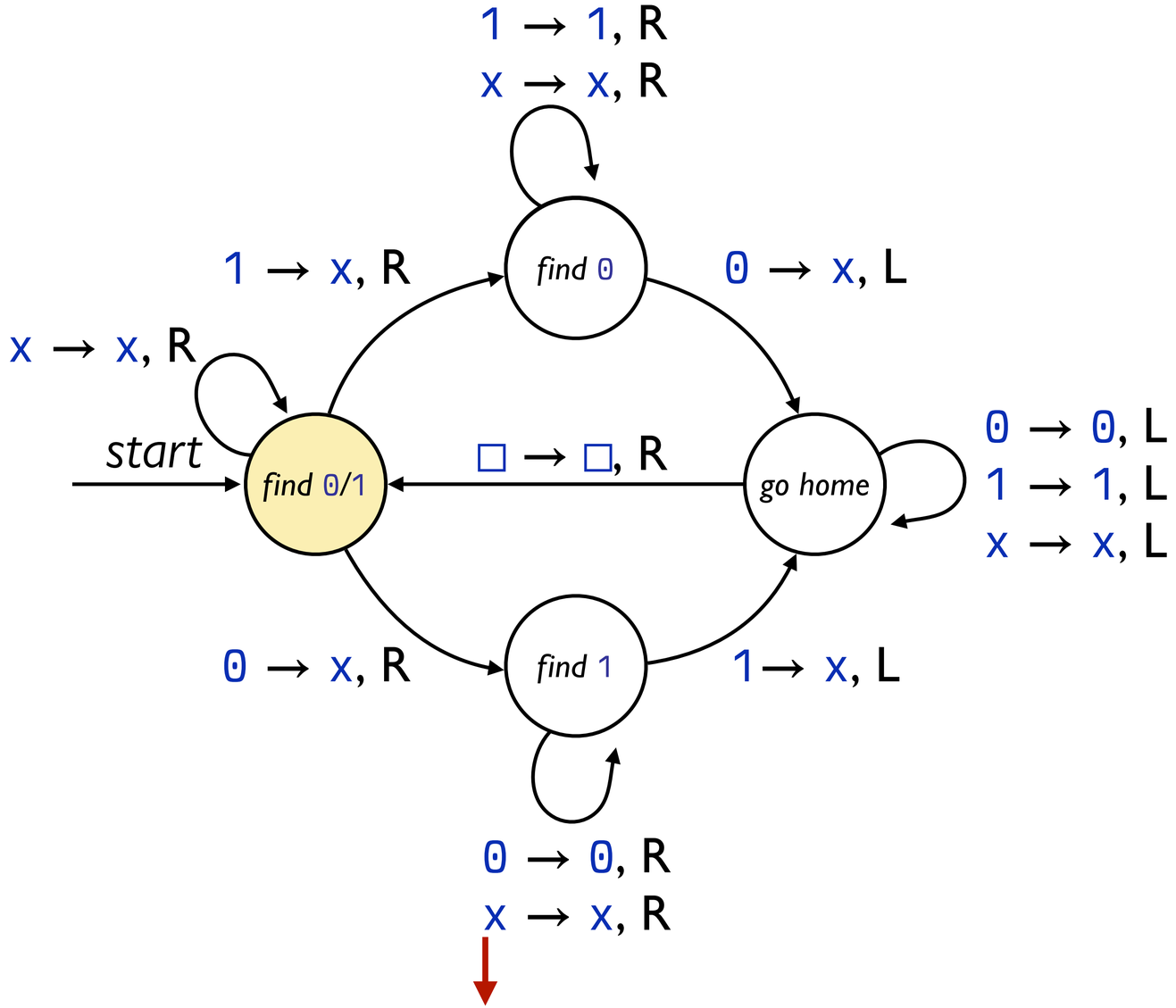

x x x x x x x x

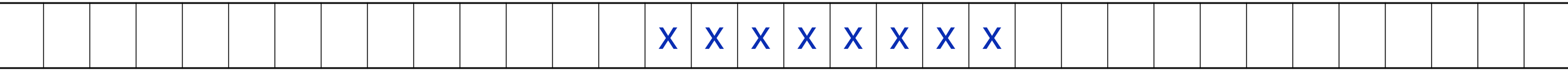

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

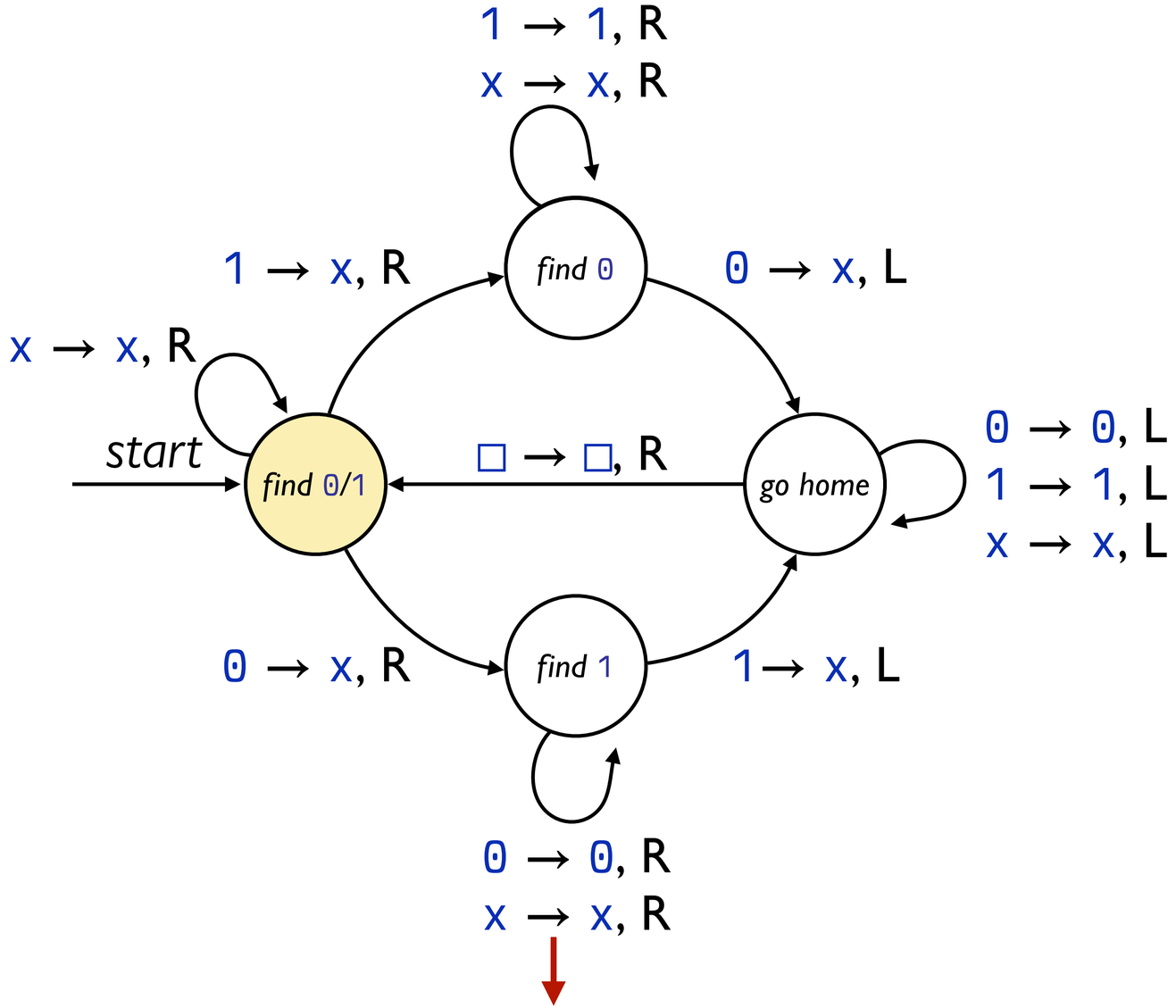

x x x x x x x x

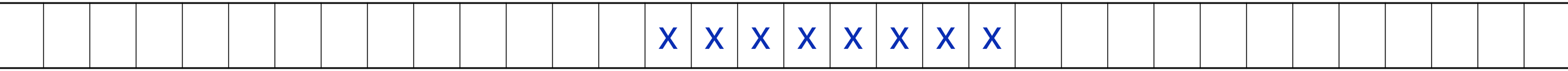

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

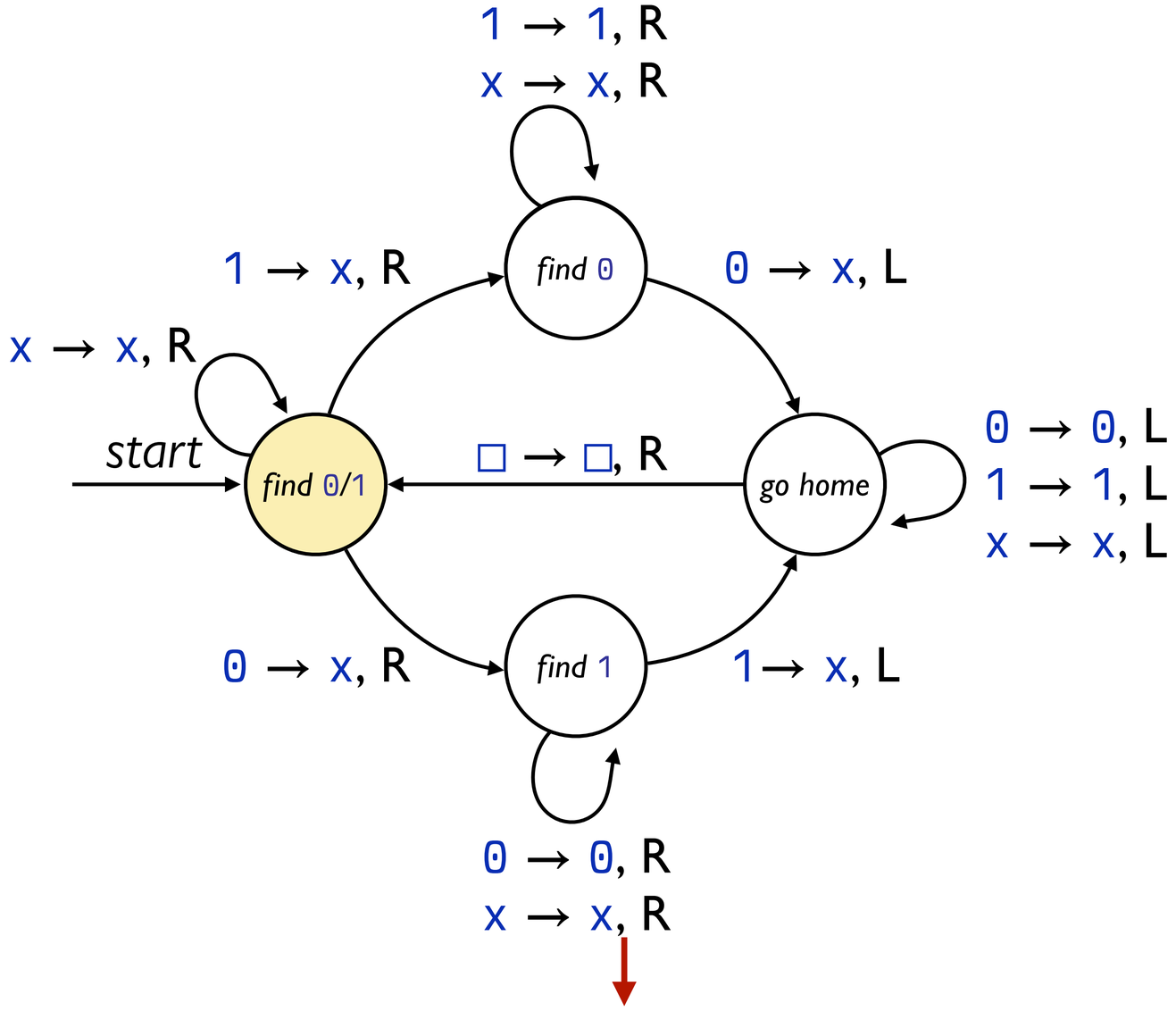

x x x x x x x x

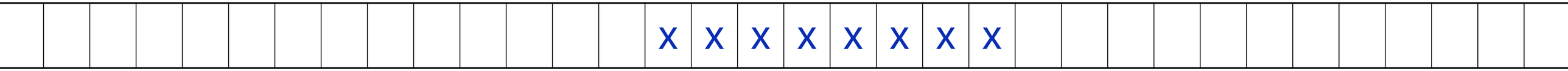

1 → 1, R
x → x, R
1 → x, R
0 → x, L
x → x, R
start
find 0/1
find 0
go home
□ → □, R
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

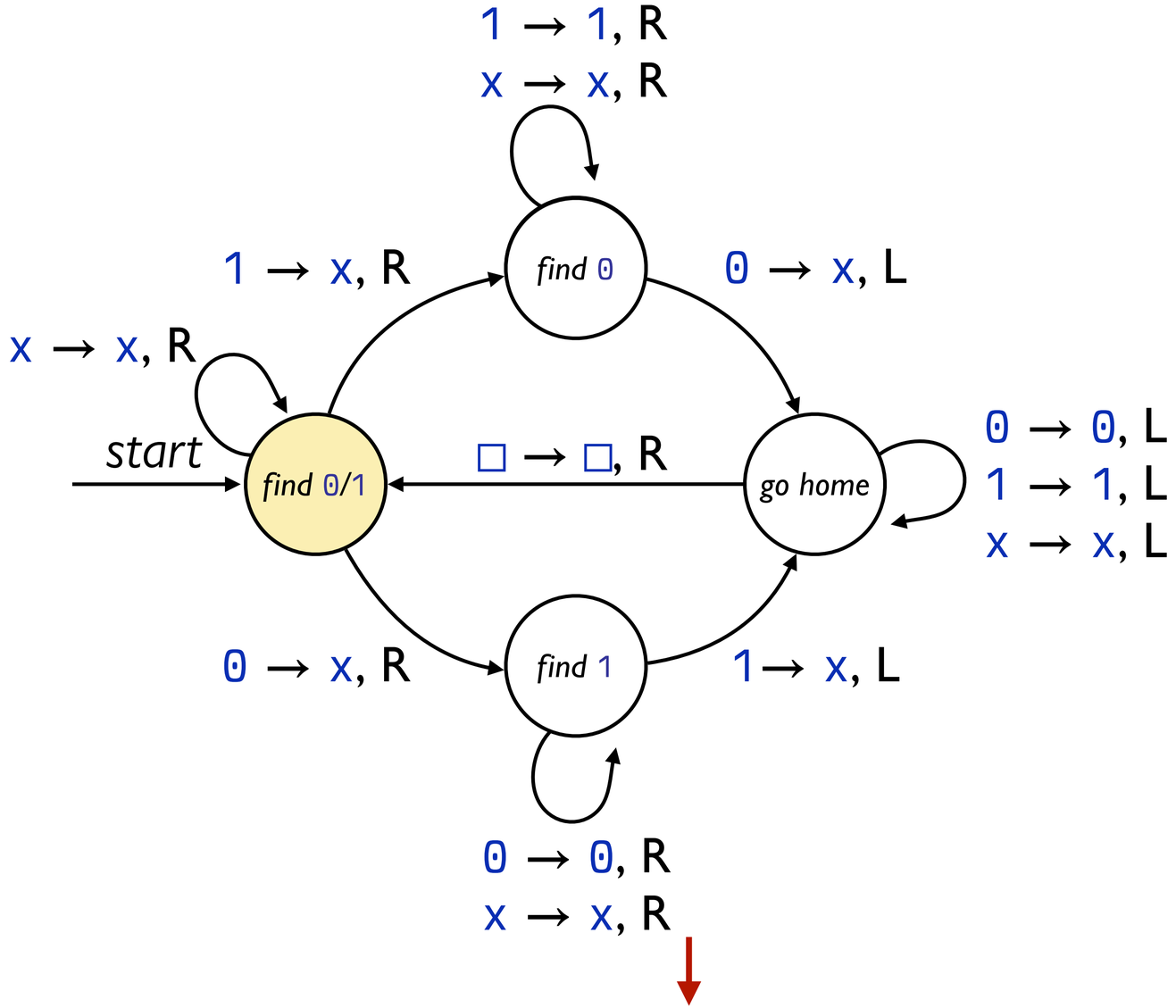

x x x x x x x x

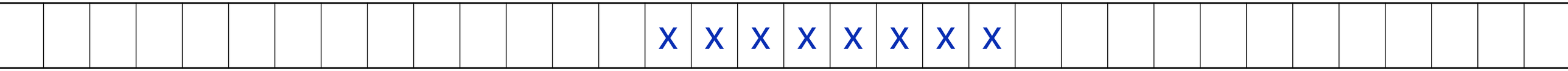

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1→ x, L
0 → 0, R
x → x, R

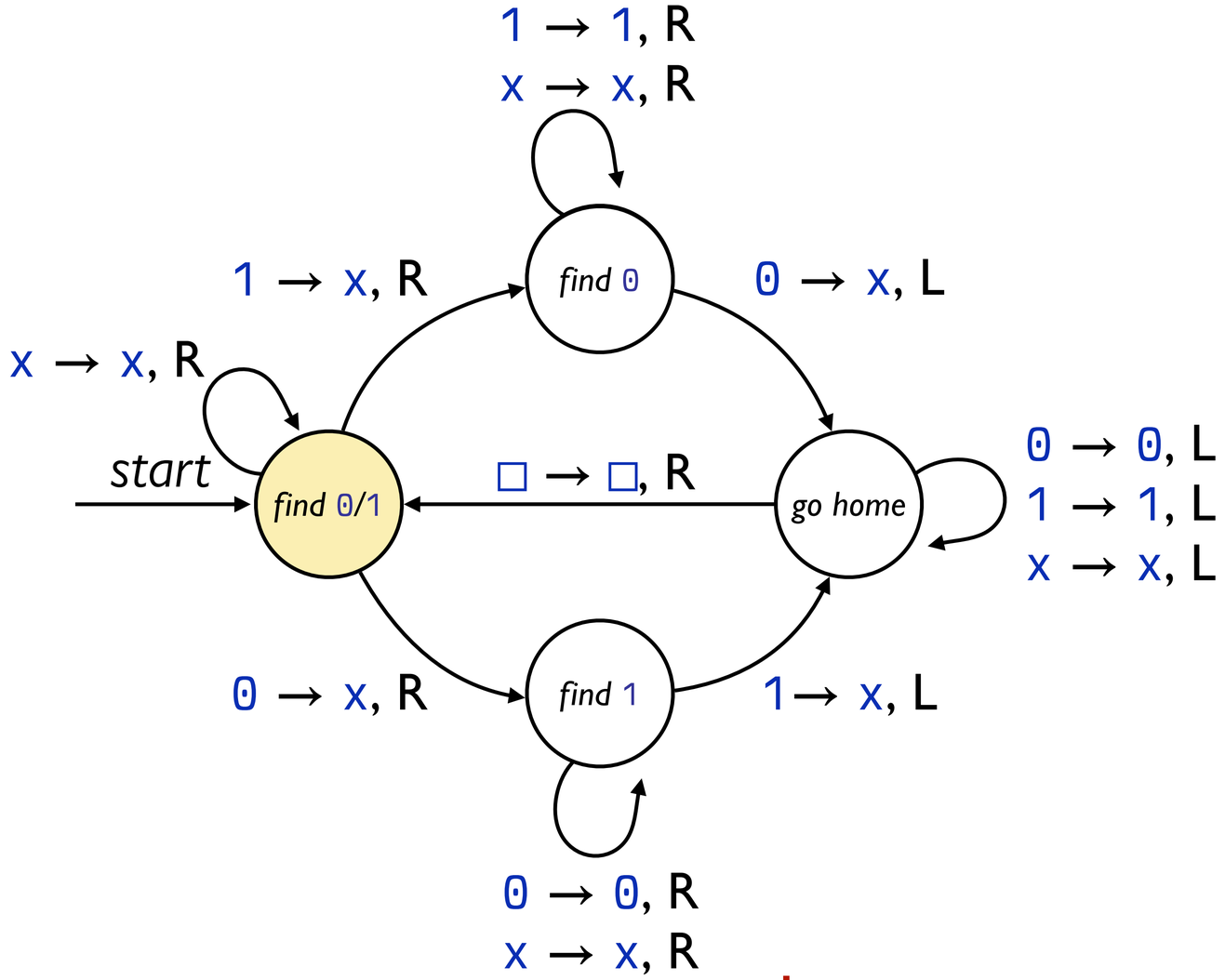

x x x x x x x x

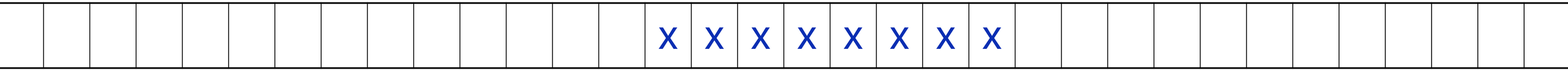

1 → 1, R
x → x, R
1 → x, R
0 → x, L
x → x, R
start
find 0/1
find 0
go home
□ → □, R
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1→ x, L
0 → 0, R
x → x, R

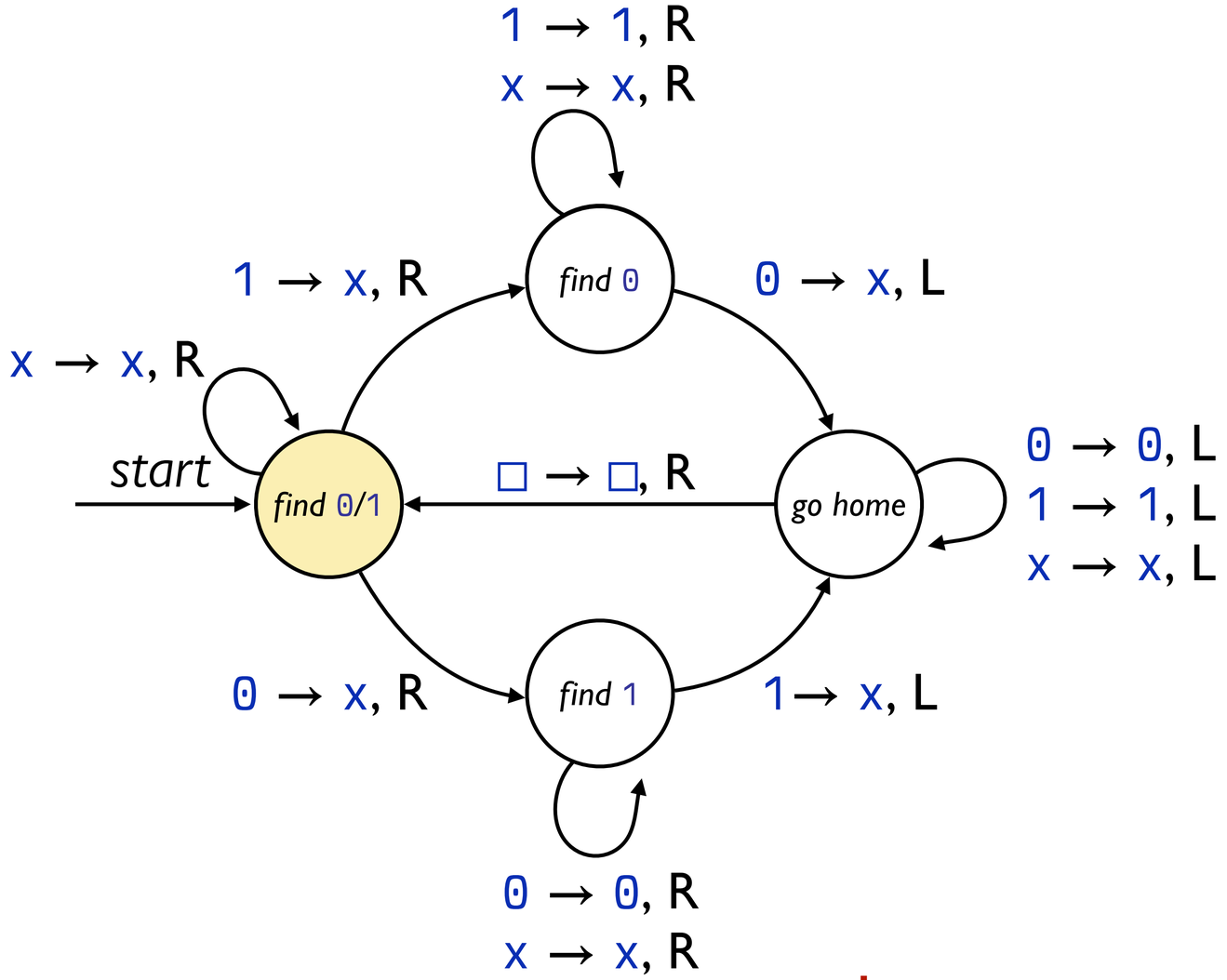

x x x x x x x x

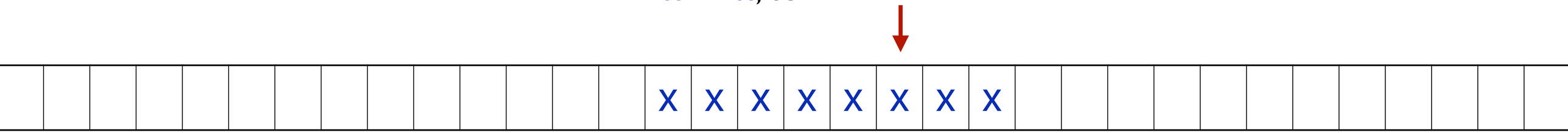

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

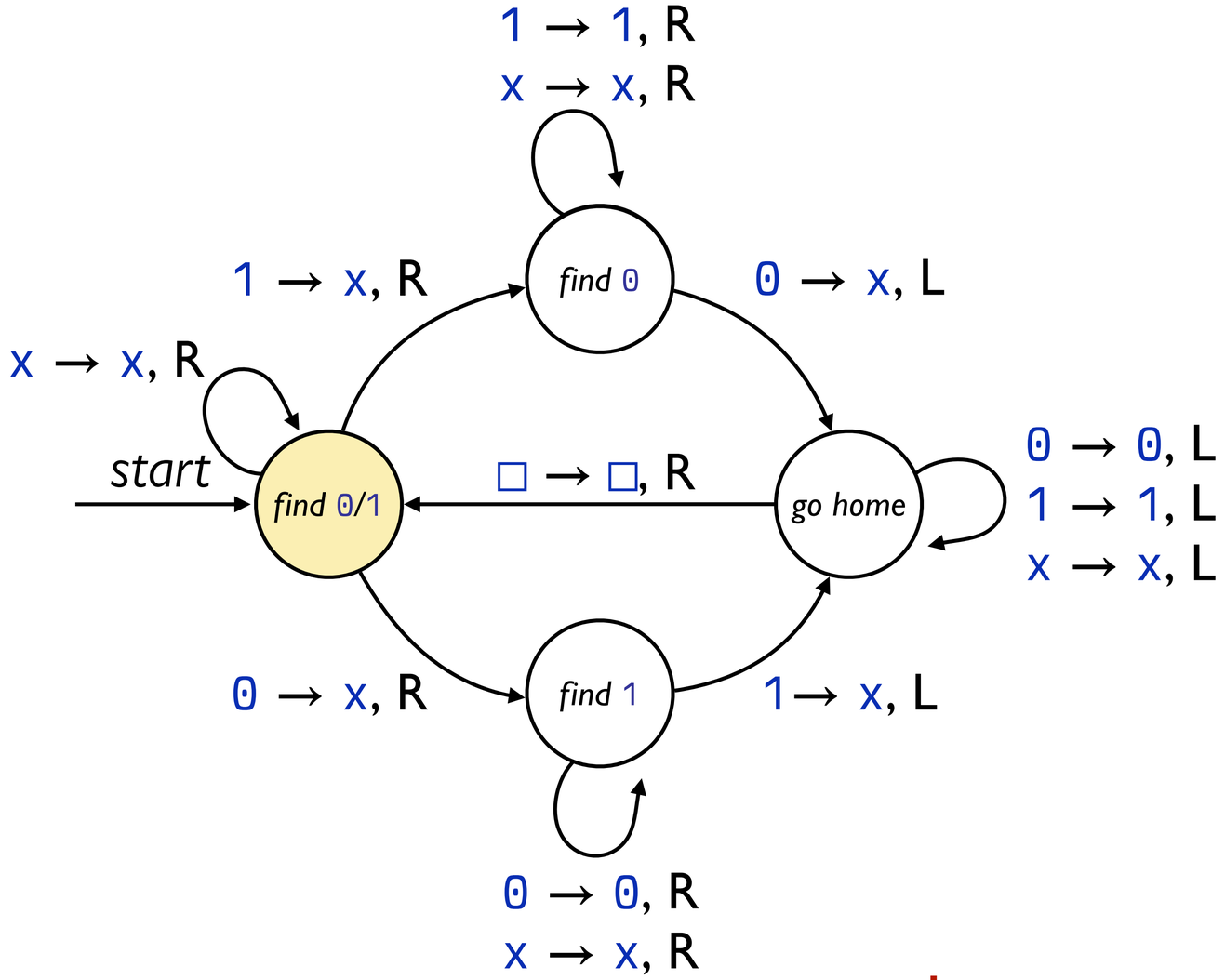

x x x x x x x x

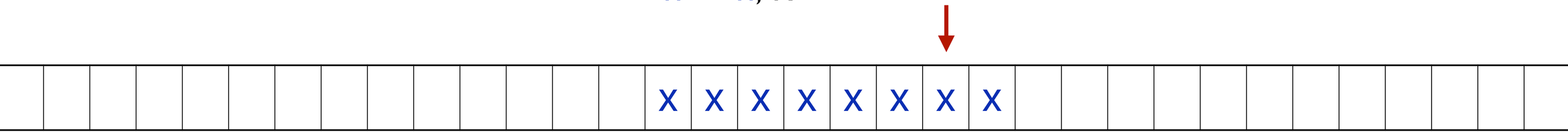

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

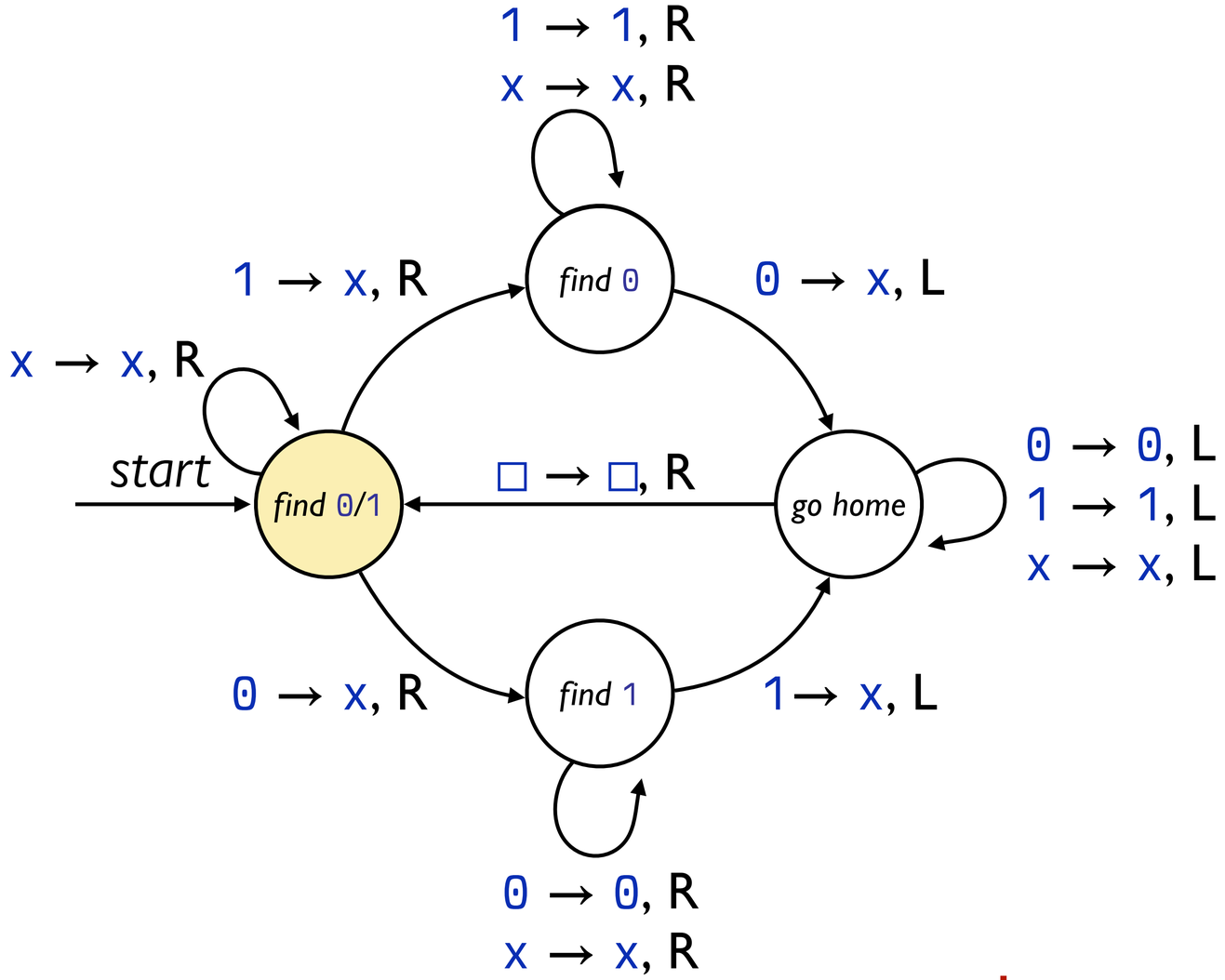

x x x x x x x x

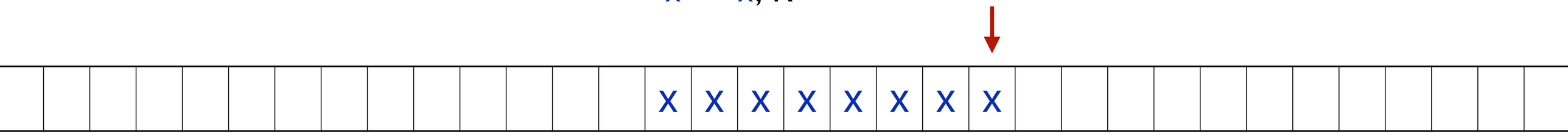

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

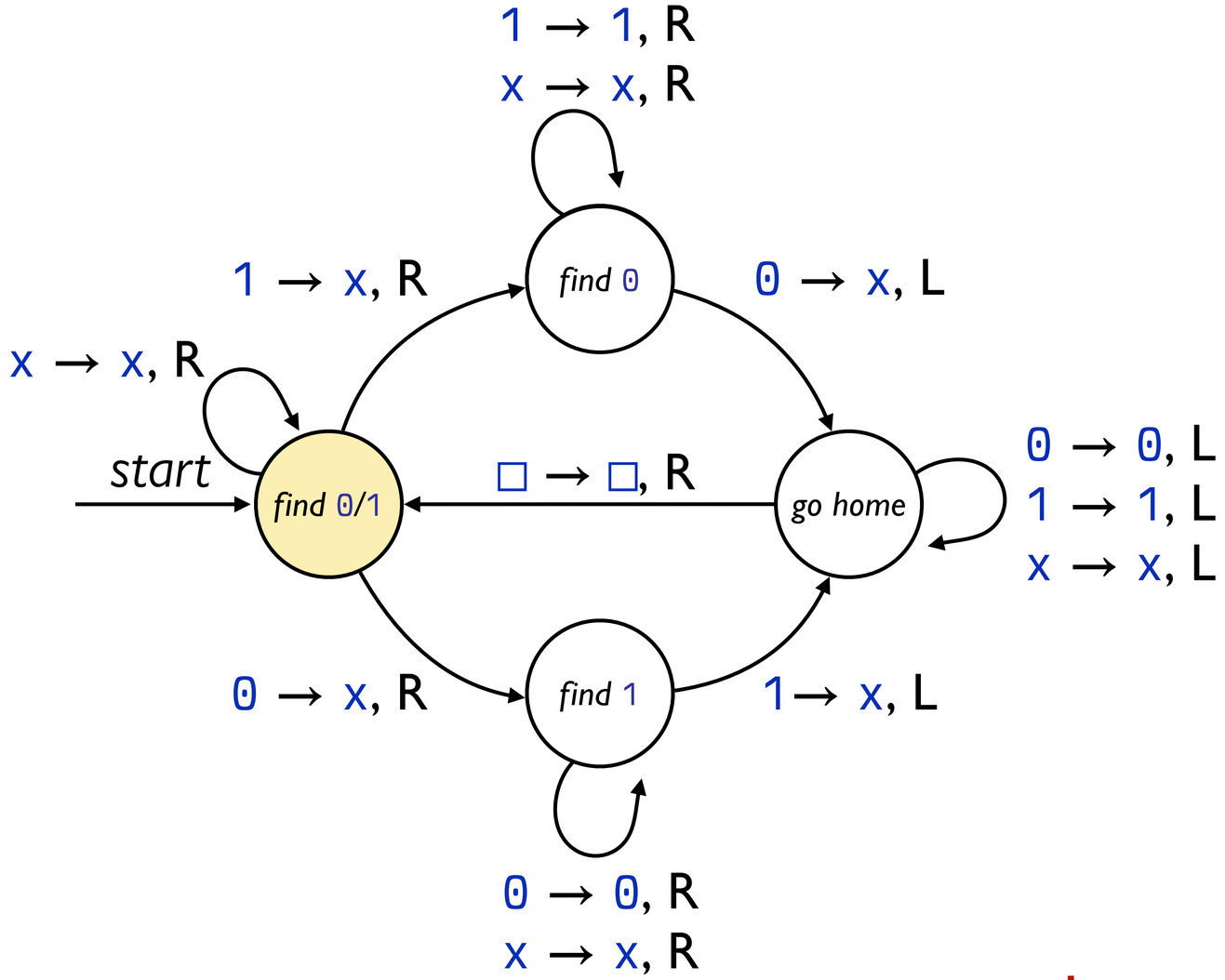

x x x x x x x x

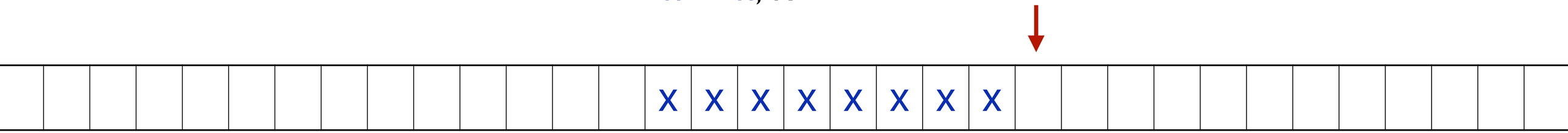

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

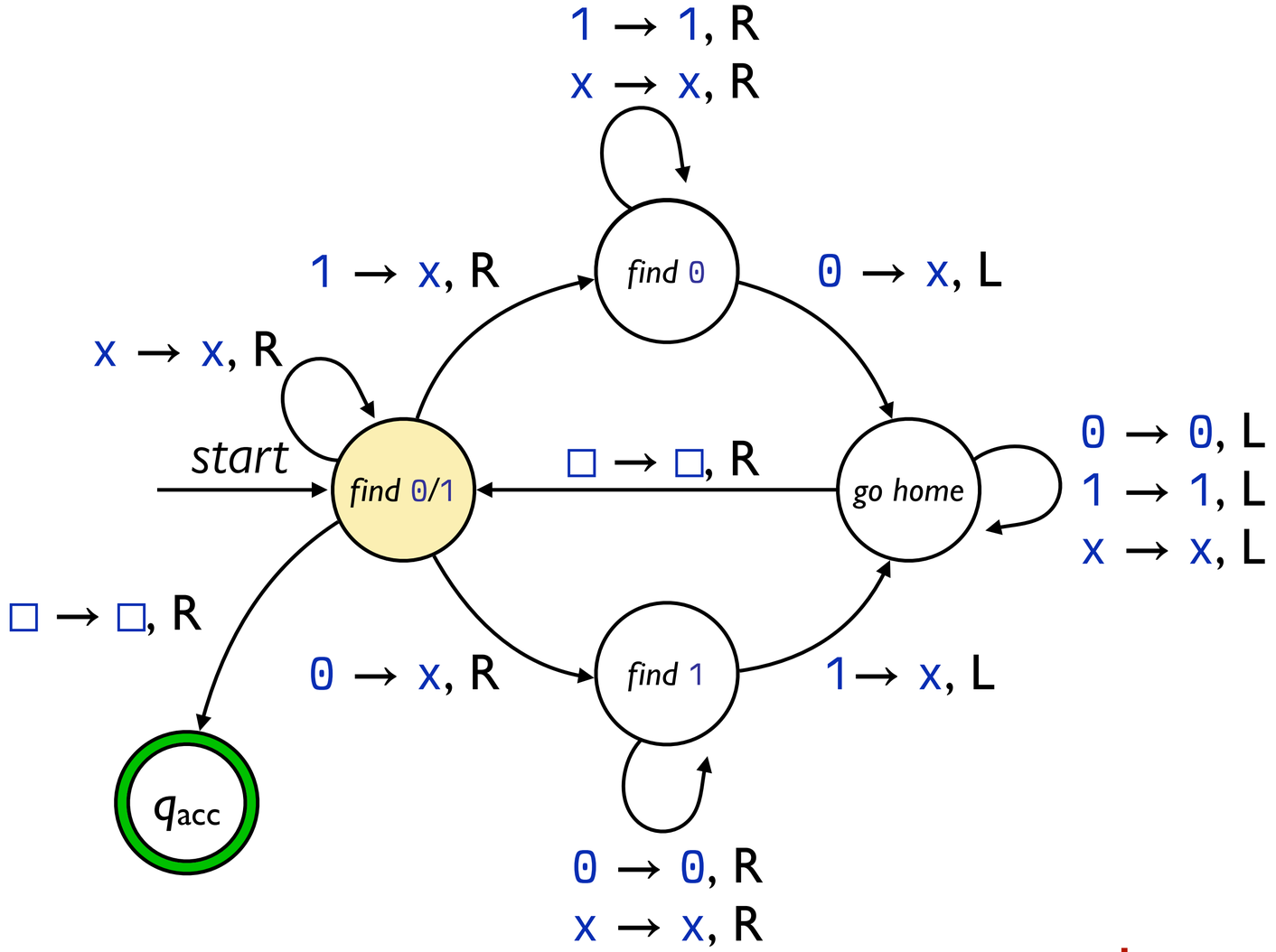

x x x x x x x x

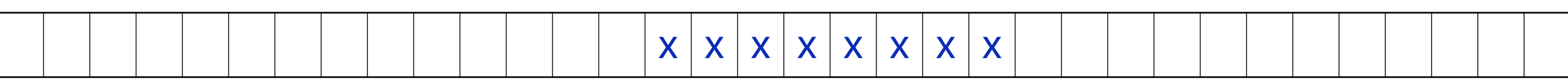

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

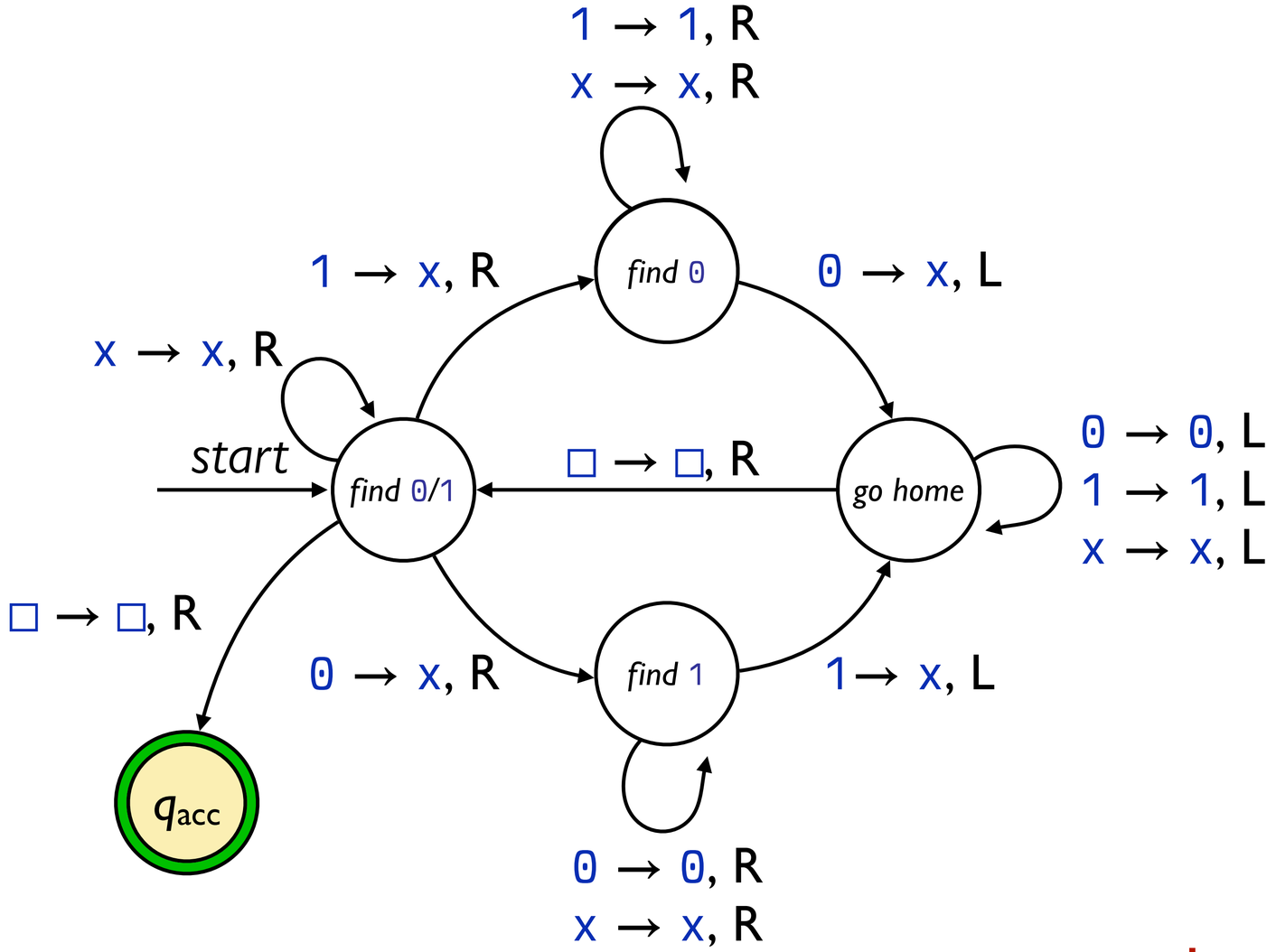

x x x x x x x x

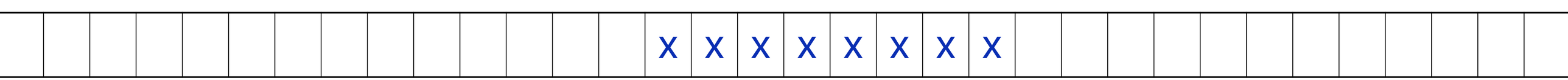

1 → 1, R
x → x, R
1 → x, R
0 → x, L
x → x, R
start
find 0/1
find 0
go home
□ → □, R
0 → 0, L
1 → 1, L
x → x, L
0 → x, R
find 1
1 → x, L
0 → 0, R
x → x, R

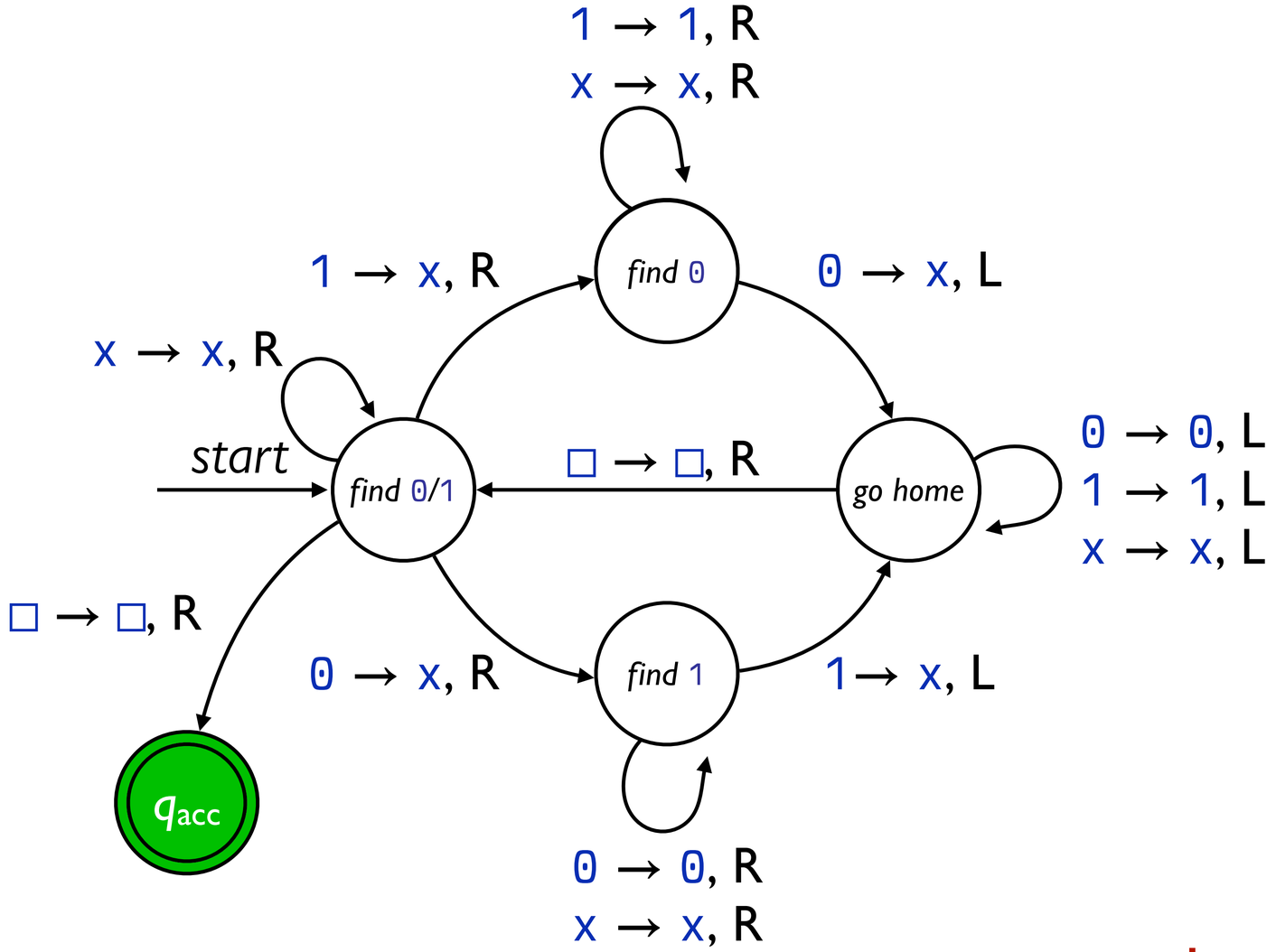

x x x x x x x x

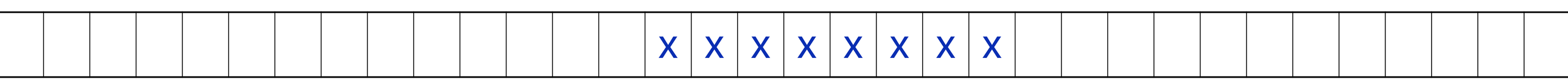

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
□ → □, R
0 → x, R
find 1
1 → x, L
qacc
0 → 0, R
x → x, R

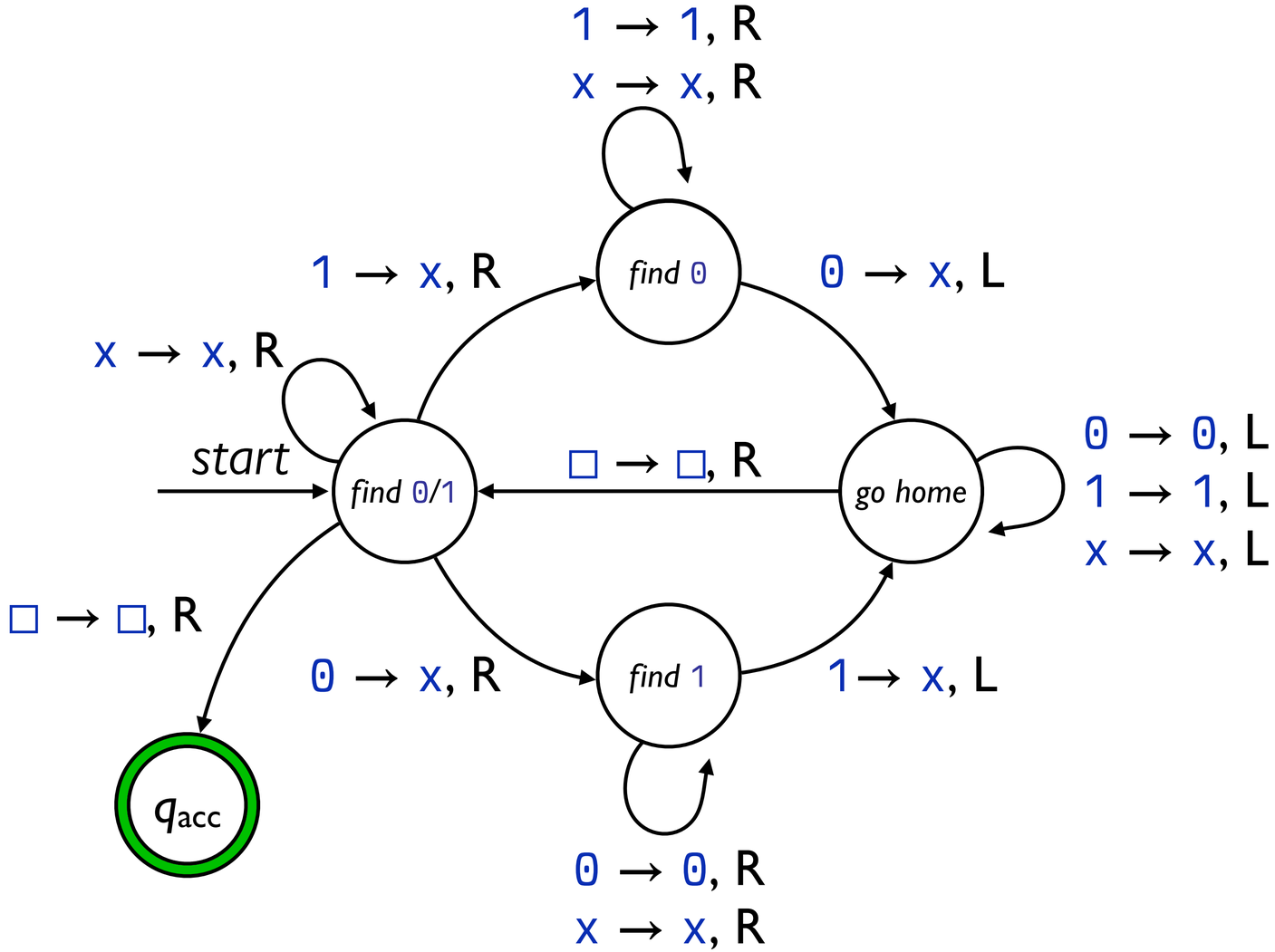

x x x x x x x x

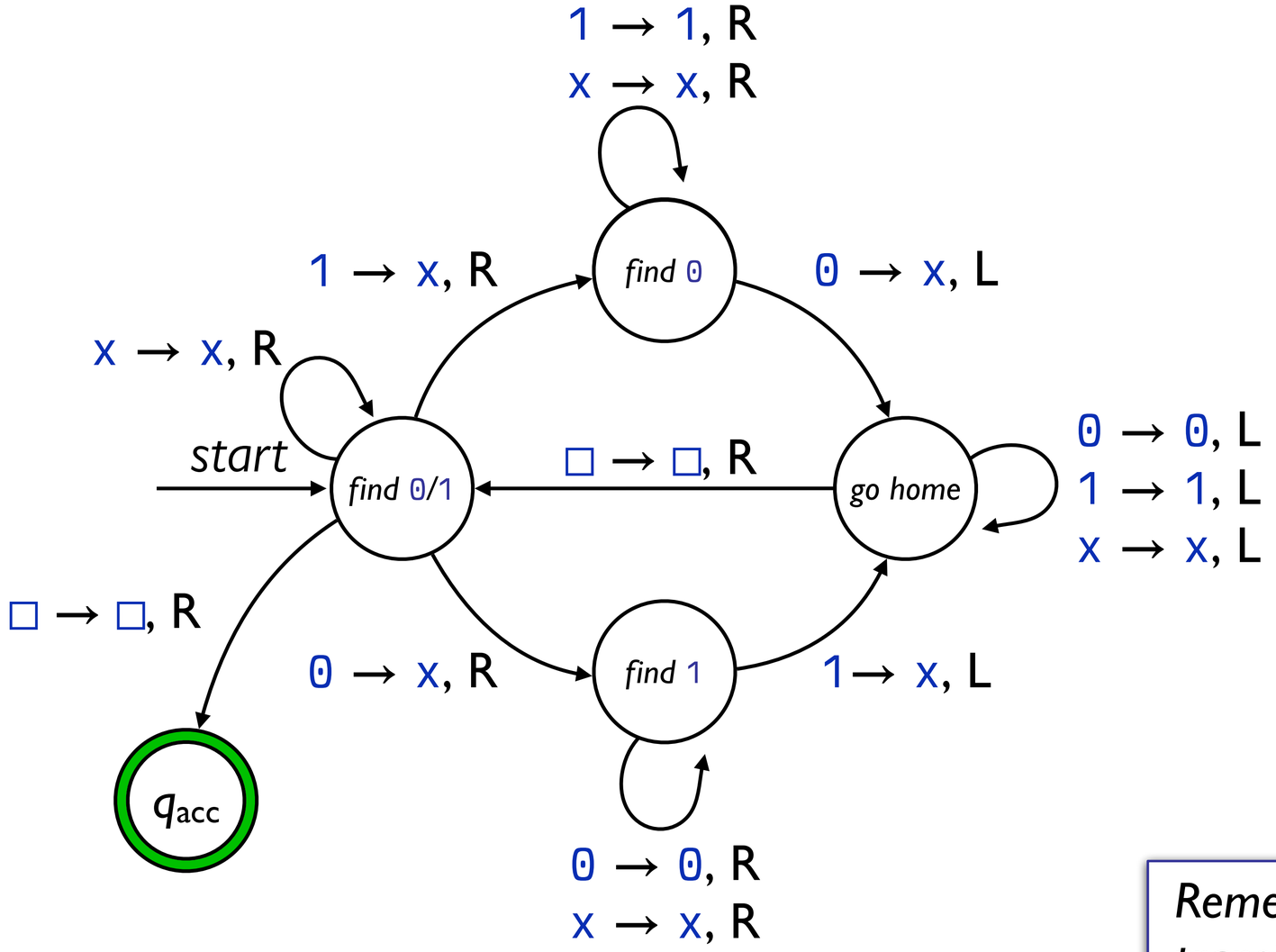

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
□ → □, R
0 → x, R
find 1
1 → x, L
qacc
0 → 0, R
x → x, R


x x x x x x x x

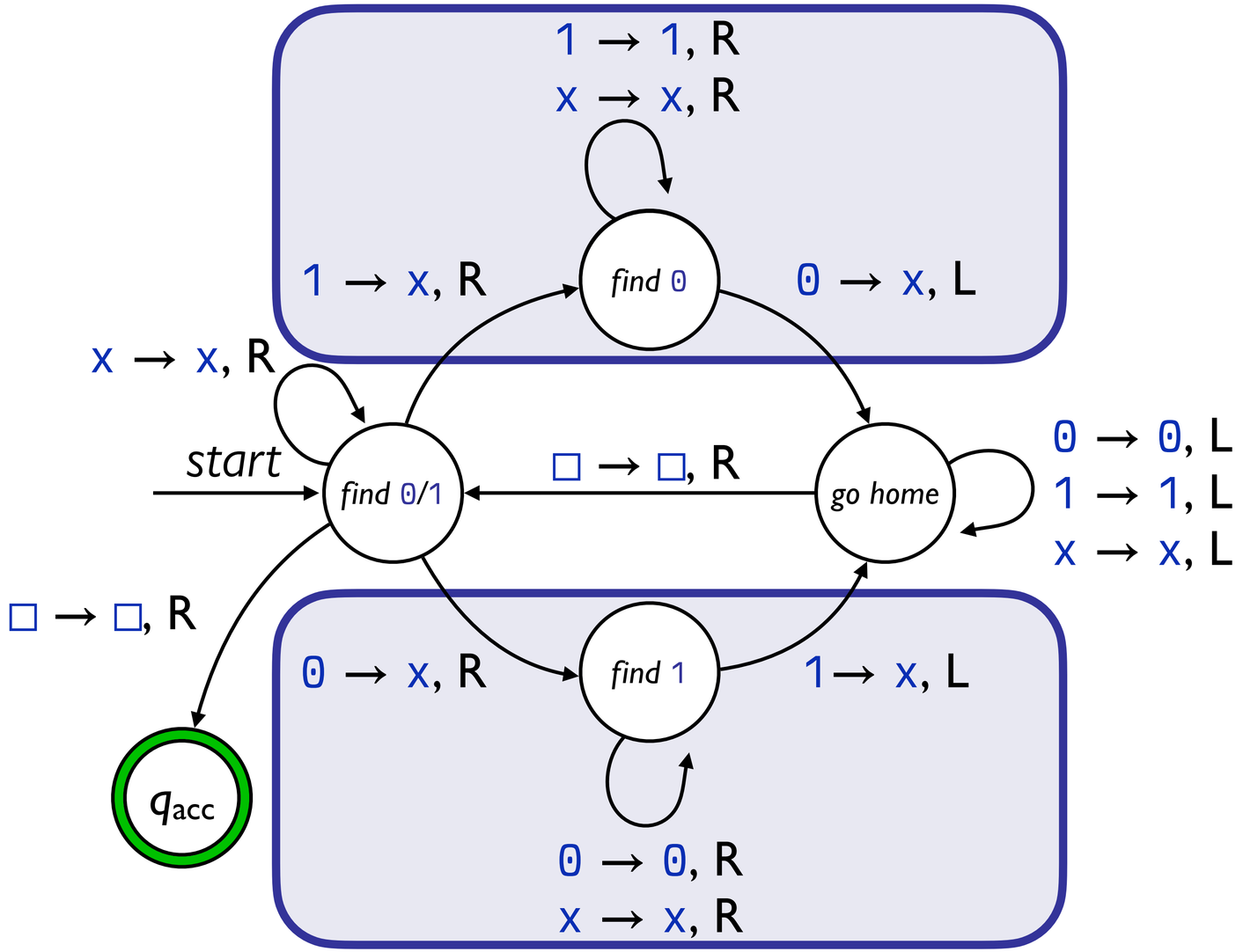

1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
□ → □, R
0 → x, R
find 1
1 → x, L
qacc
0 → 0, R
x → x, R


x x x x x x x x


1 → 1, R
x → x, R
find 0
1 → x, R
0 → x, L
x → x, R
start
find 0/1
□ → □, R
go home
0 → 0, L
1 → 1, L
x → x, L
□ → □, R
0 → x, R
find 1
1 → x, L
qacc
0 → 0, R
x → x, R



Remember that all missing transitions implicitly reject.



These parts of the Turing machine are two "cases" it handles, though the tapes they operate on are indistinguishable.

Constant storage

Sometimes a Turing machine needs to remember some additional information that can't (at least conveniently) be put on the tape.

In this case, you can use the same techniques you used in designing DFAs and introduce extra states into the Turing machine's finite-state control.

The finite-state control can only remember one of finitely many things, but that might be all you need!

One more TM for practice

Design a Turing machine for the language $\{ww^R \mid w \in \{a, b\}^*\}$

Acknowledgments

This lecture incorporates material from:

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

Nancy Ide, Vassar College

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*