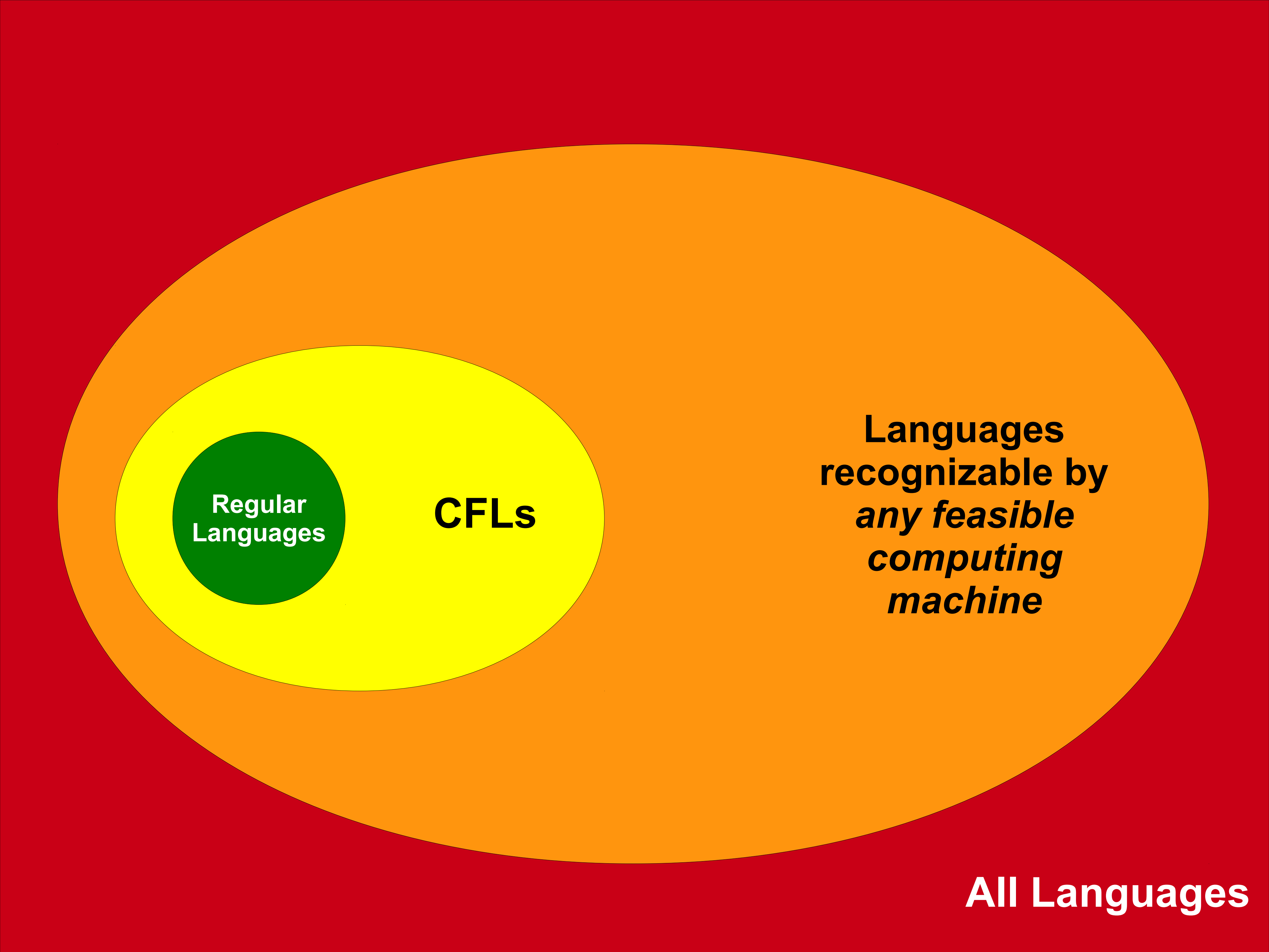
Turing Machines

Part One

What problems can we solve with a computer?


Regular Languages
CFLs
Languages recognizable by *any feasible computing machine*
All Languages

That same drawing, to scale.

The Problem

- Finite automata accept precisely the regular languages.
- We may need unbounded memory to recognize context-free languages.
  - e.g. $\{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ requires unbounded counting.
- How do we build an automaton with finitely many states but unbounded memory?

A Brief History Lesson

<ahem>

Technology has solved all of mankind’s problems! No more wars or sad ever!

Hilbert’s Vision

- 1900: International Congress of Mathematicians meeting in Paris
- Proposes 23 unsolved problems as the agenda for the coming years
- An important theme is not simply proving more theorems, but achieving *automation* of theorem-proving, even theorem generation.
- Humanity lives in leisure while all Truth flows effortlessly into our hands on a ticker tape!

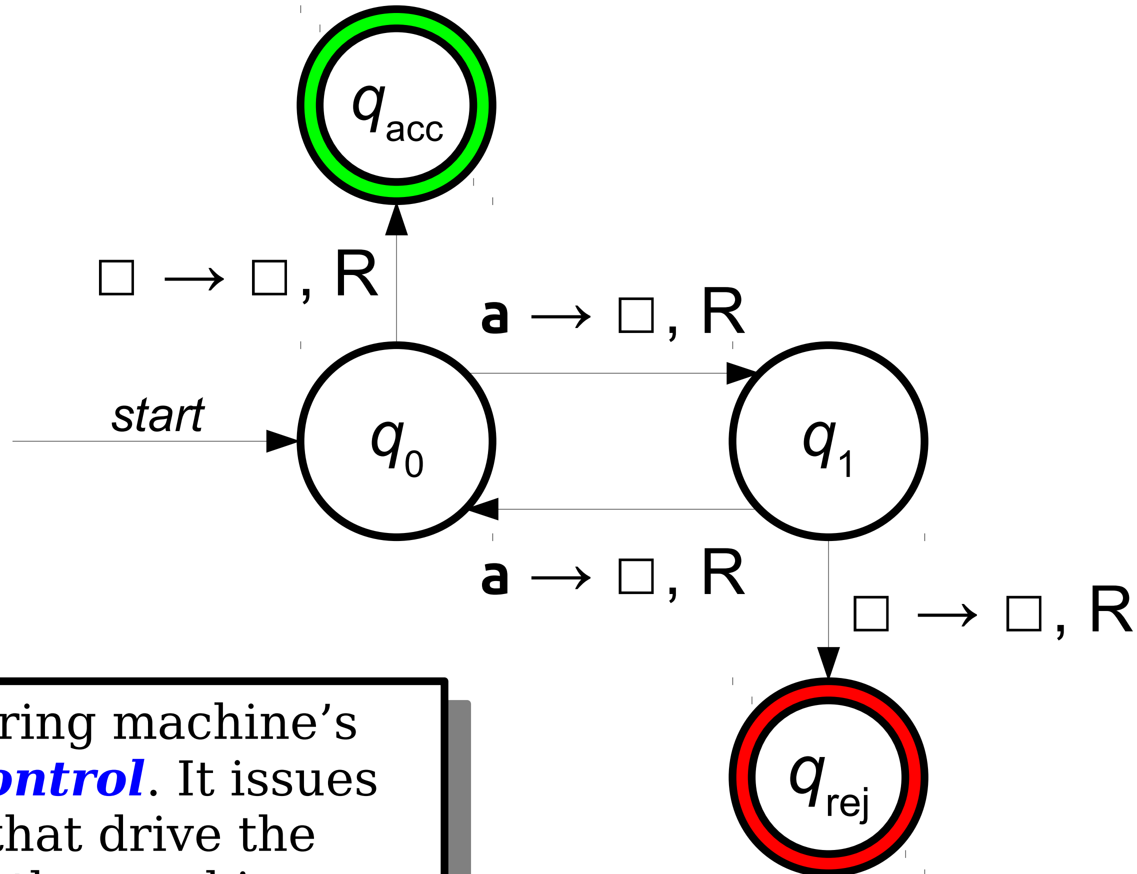
“No one shall expel us from the Paradise that Cantor has created!” -David Hilbert

Hi there, Reality!

- Hilbert’s agenda is both a spectacular success and a spectacular failure
- Inspires some of the most impactful theoretical work in mathematical history, human history
- Brings us heroes like **Alan Turing** and **Kurt Gödel**!
- These incredible results consist of utterly demolishing all the pillars of Hilbert’s vision of automated knowledge creation, within just a few years

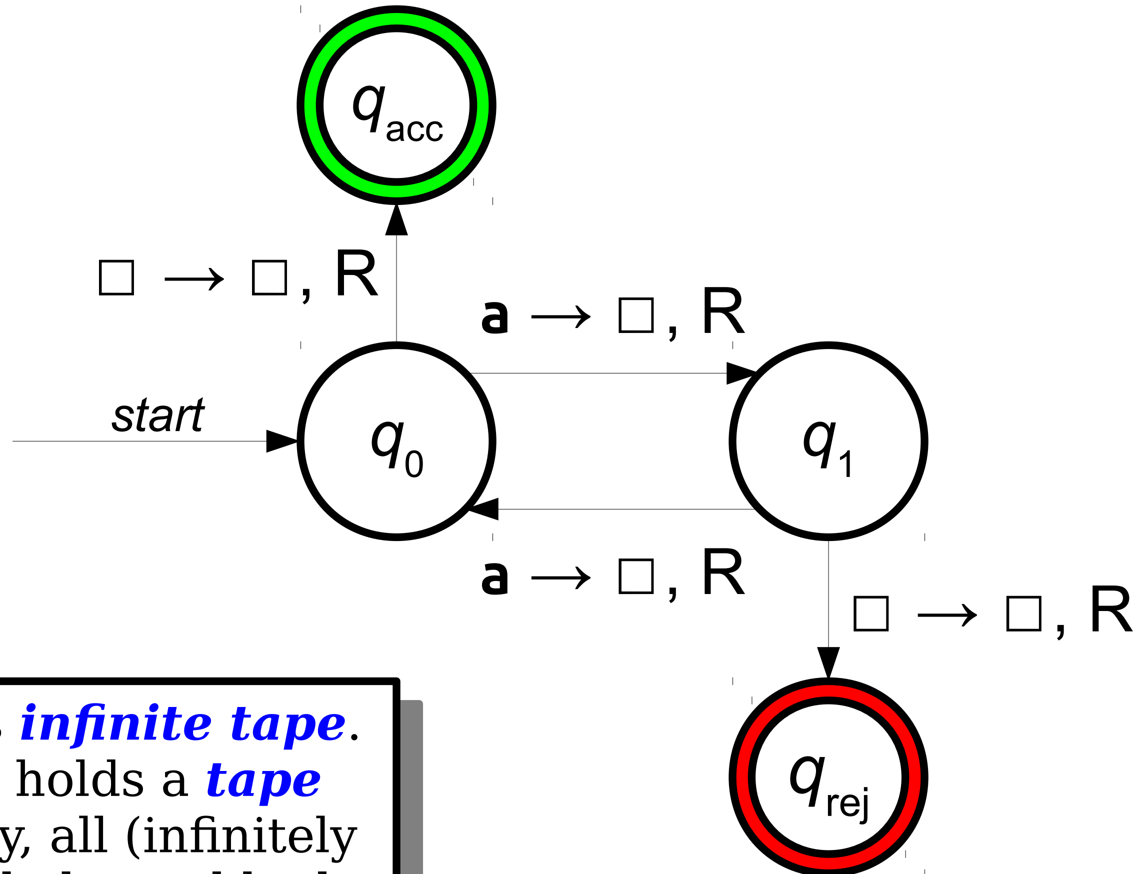
“No one shall expel us from the Paradise that Cantor has created!” -David Hilbert

A Simple Turing Machine

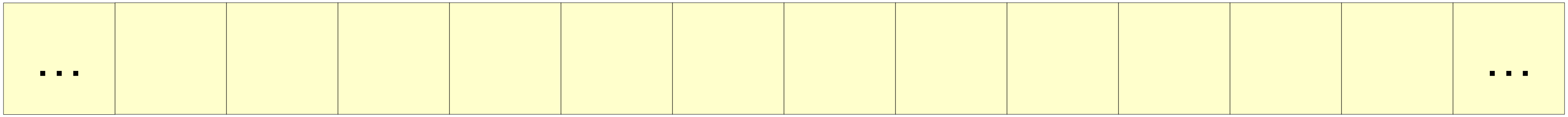


This is the Turing machine's ***finite state control***. It issues commands that drive the operation of the machine.

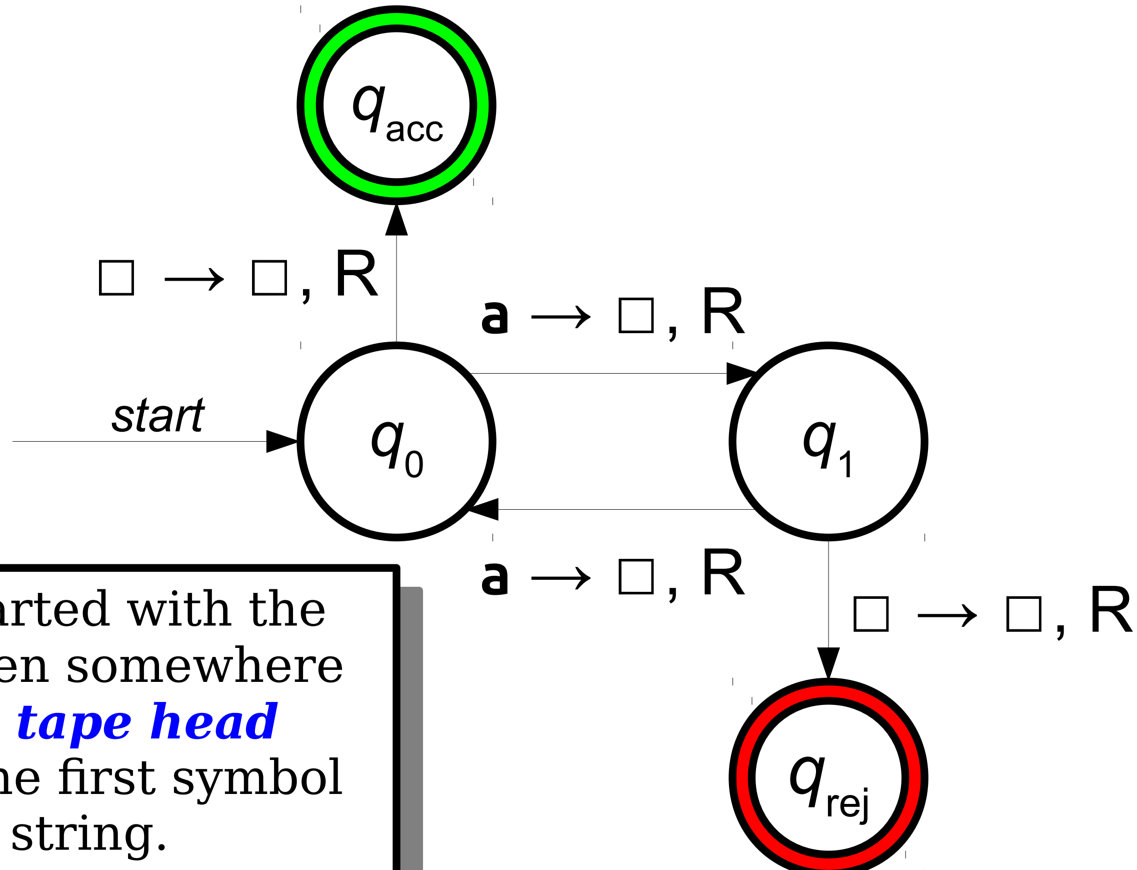
A Simple Turing Machine



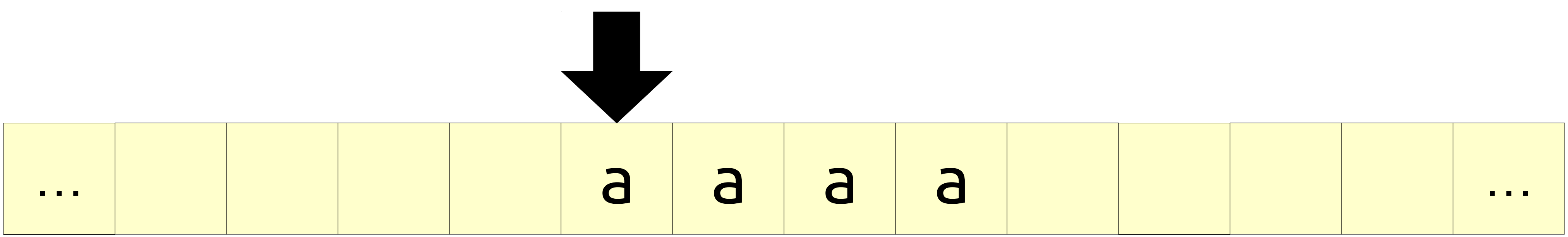
This is the TM's ***infinite tape***. Each tape cell holds a ***tape symbol***. Initially, all (infinitely many) tape symbols are blank.



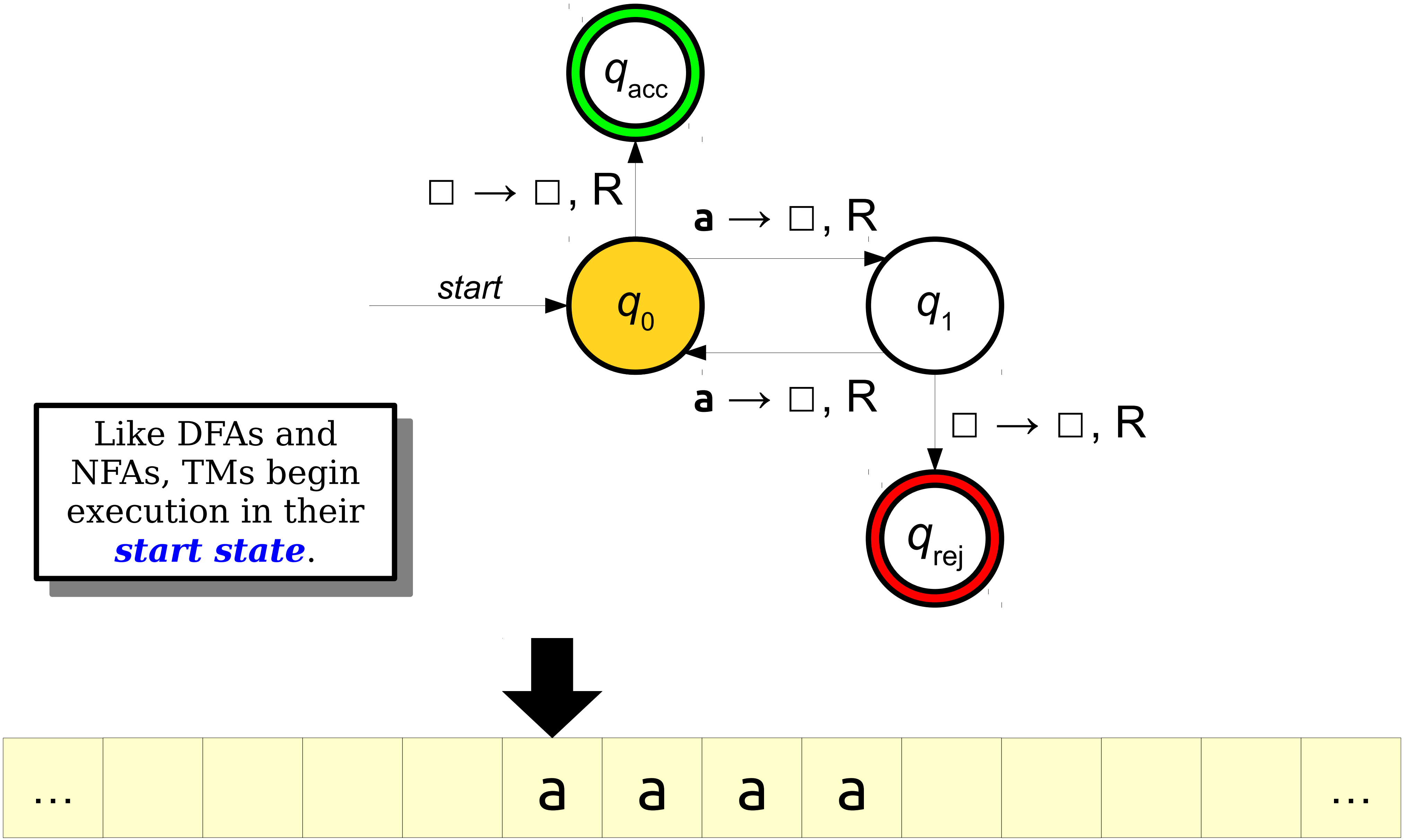
A Simple Turing Machine



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

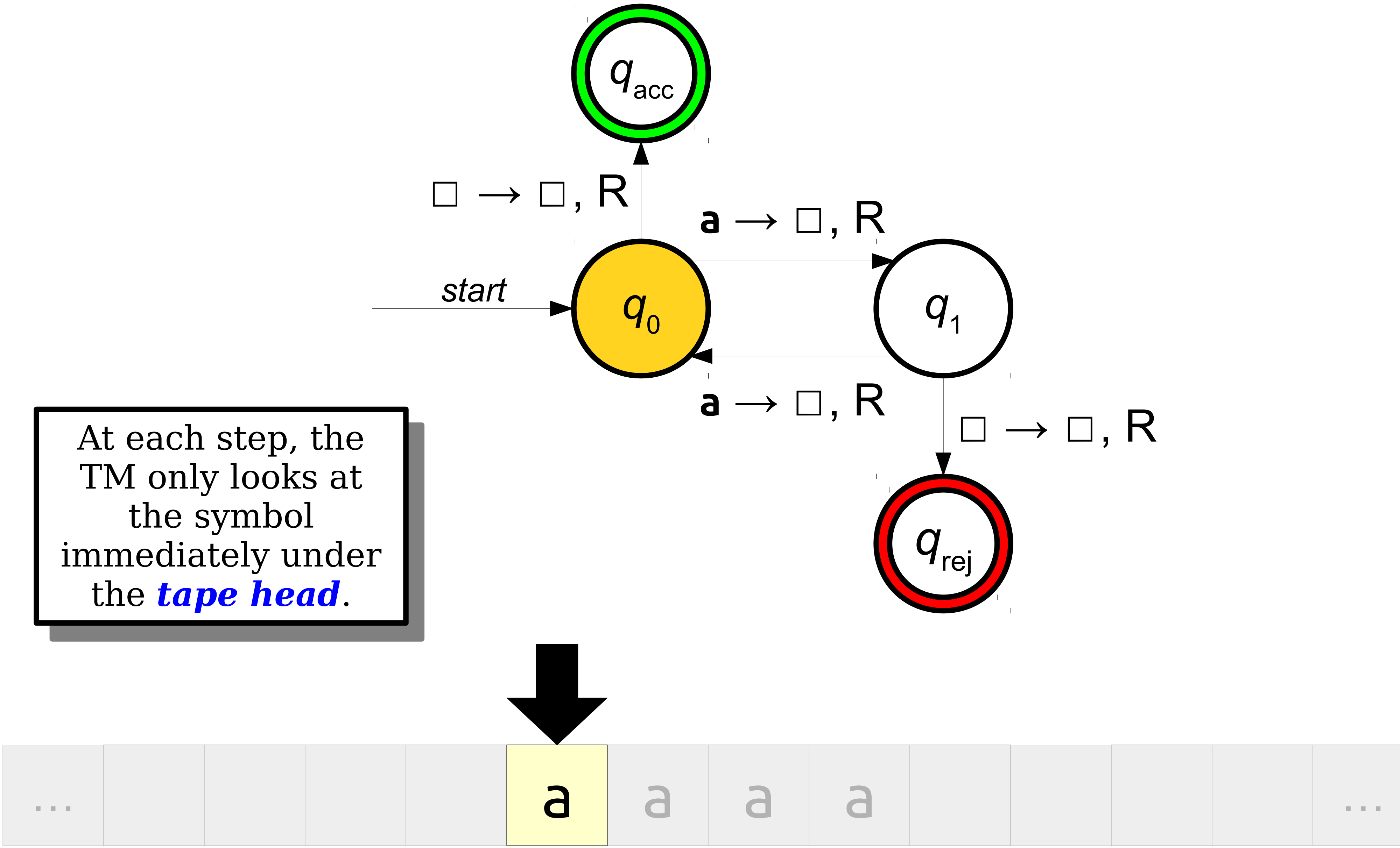


A Simple Turing Machine



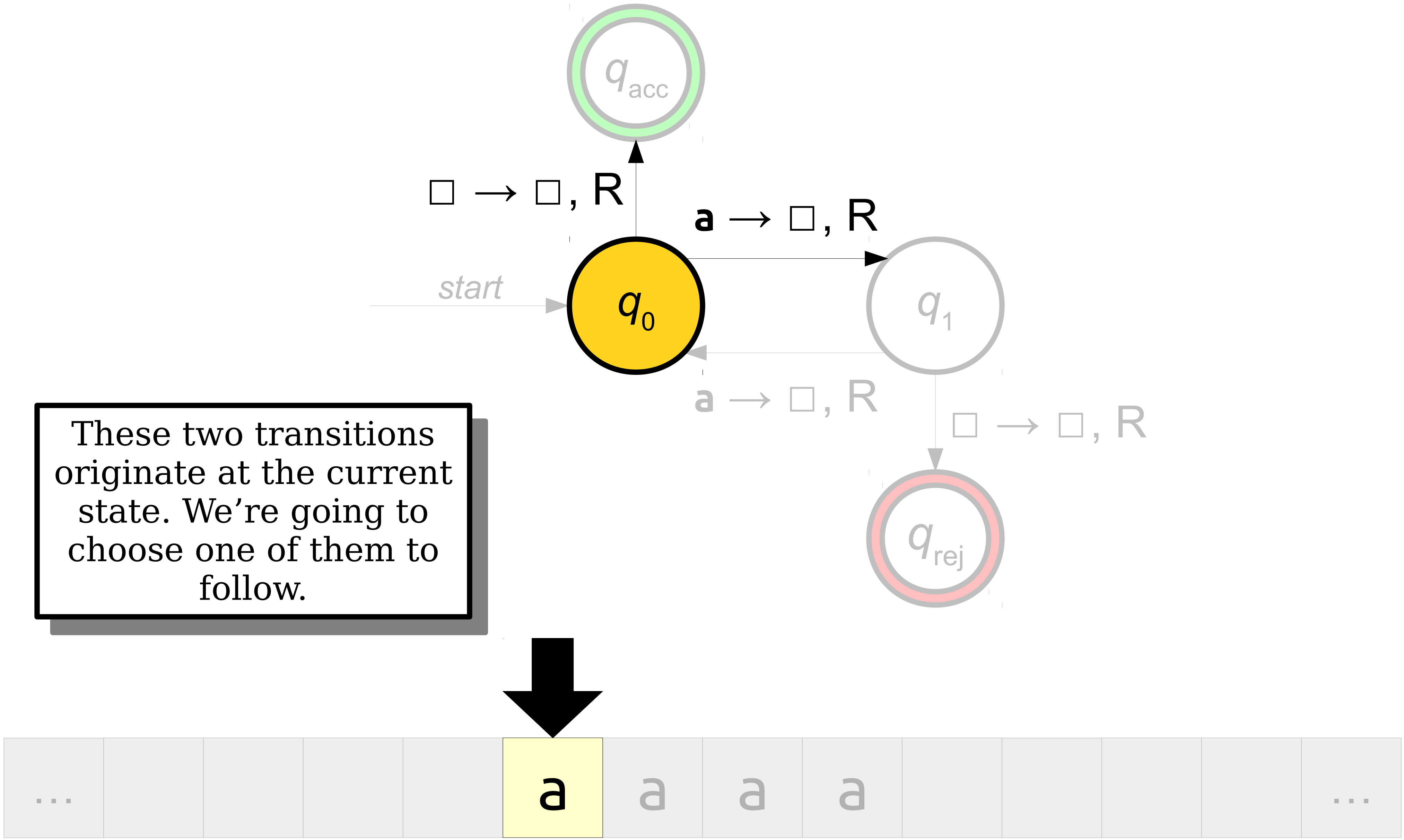
Like DFAs and NFAs, TMs begin execution in their ***start state***.

A Simple Turing Machine



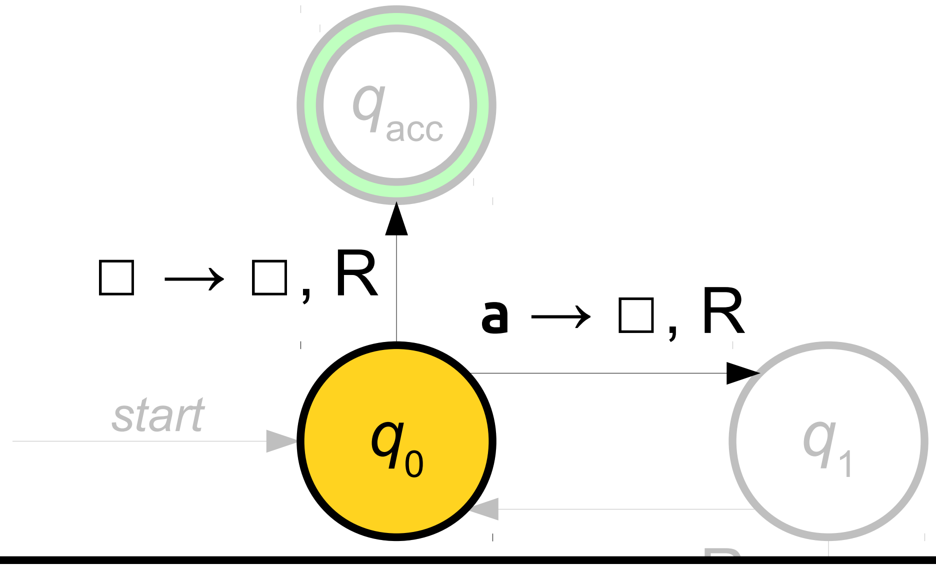
At each step, the TM only looks at the symbol immediately under the ***tape head***.

A Simple Turing Machine



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

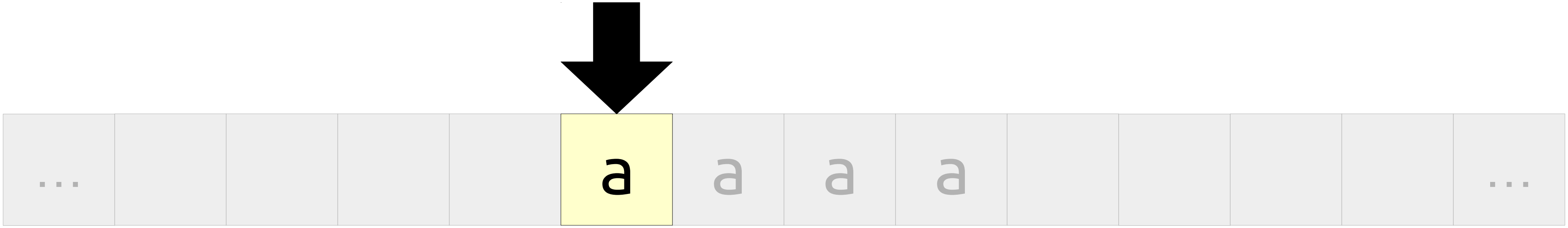
A Simple Turing Machine



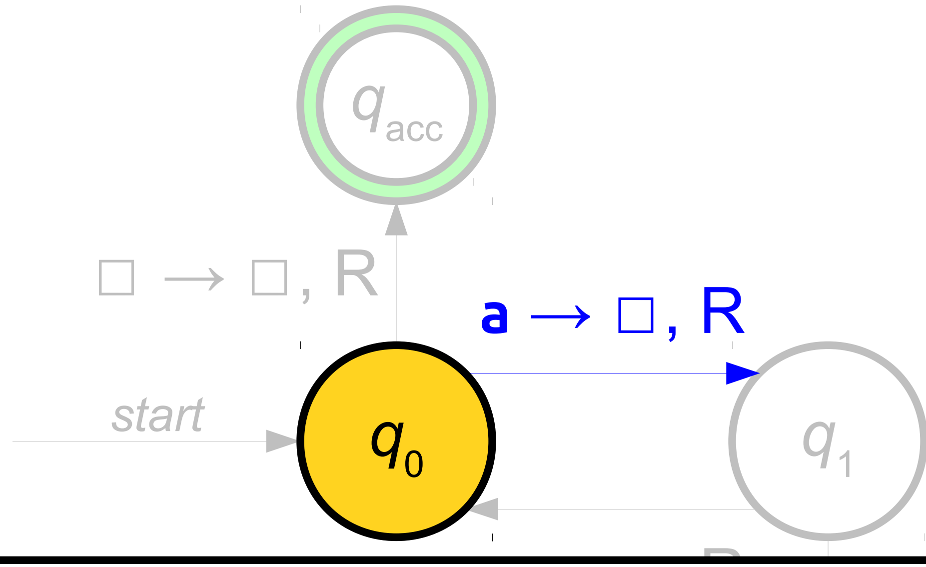
Each transition has the form

read* → *write*, *dir

and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.



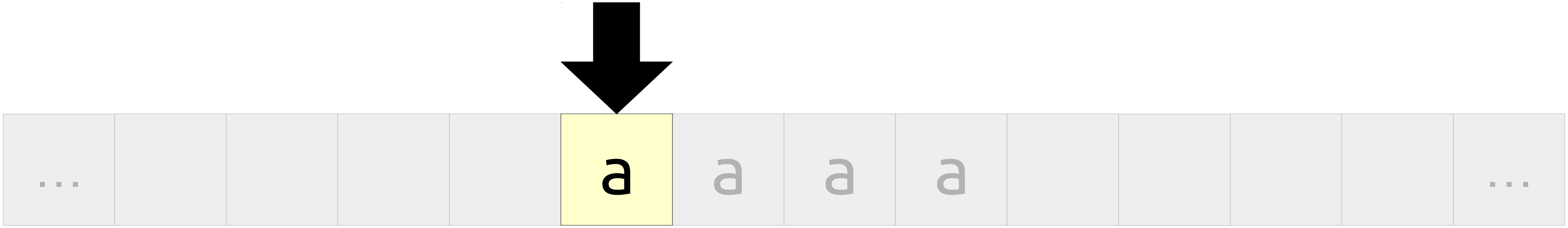
A Simple Turing Machine



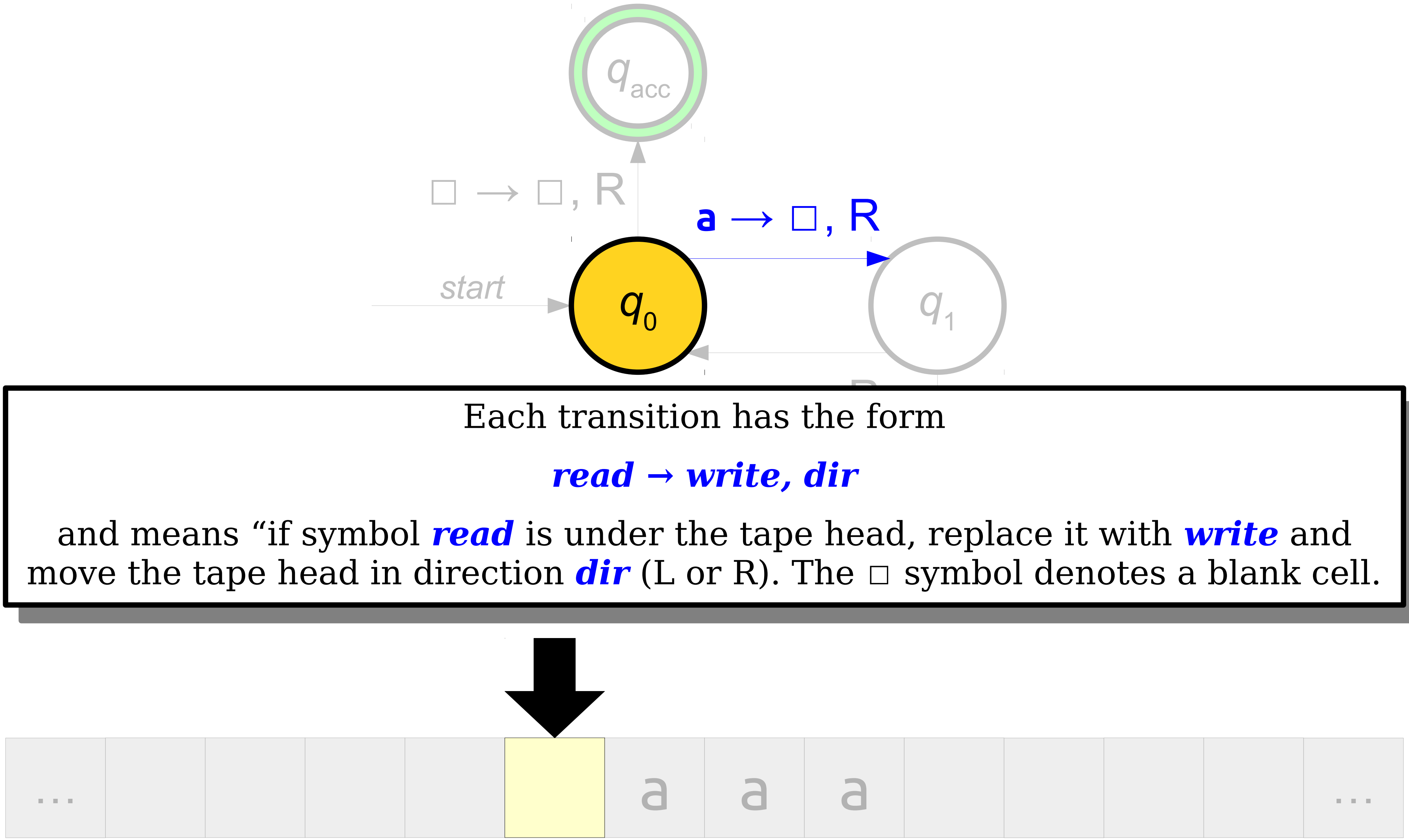
Each transition has the form

read* → *write*, *dir

and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.



A Simple Turing Machine

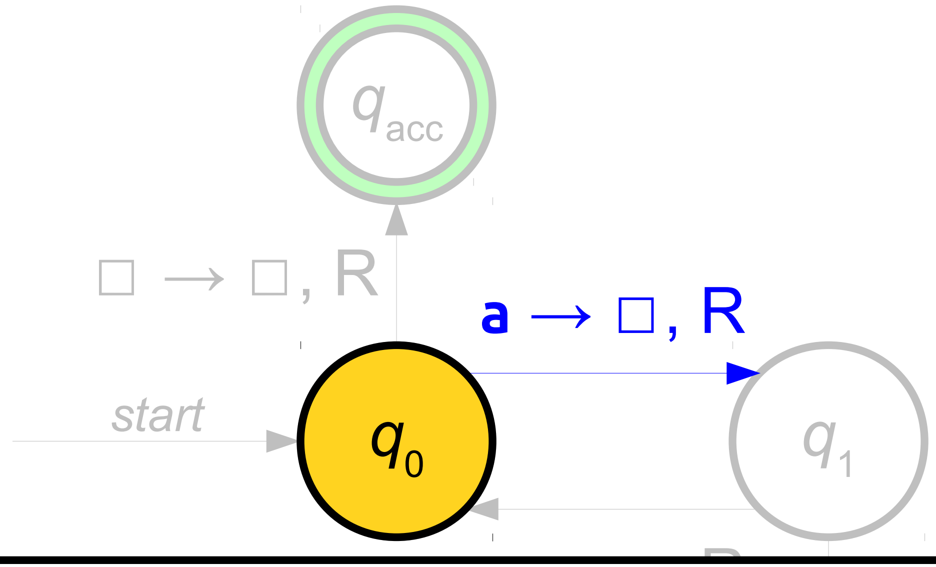


Each transition has the form

read* → *write*, *dir

and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.

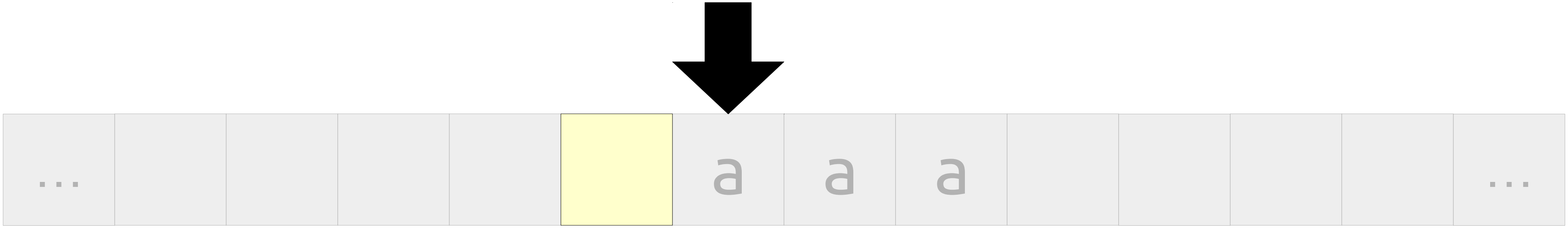
A Simple Turing Machine



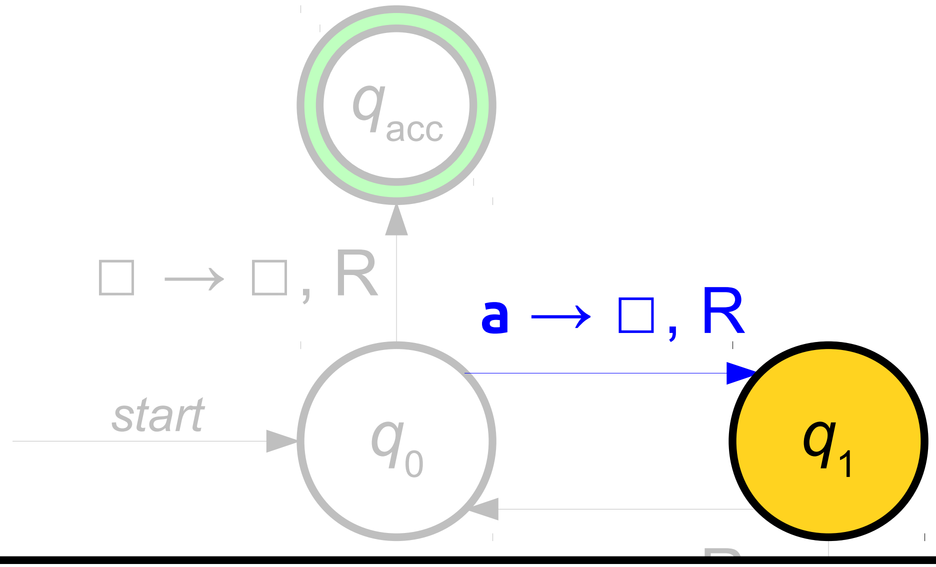
Each transition has the form

read → write, dir

and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.



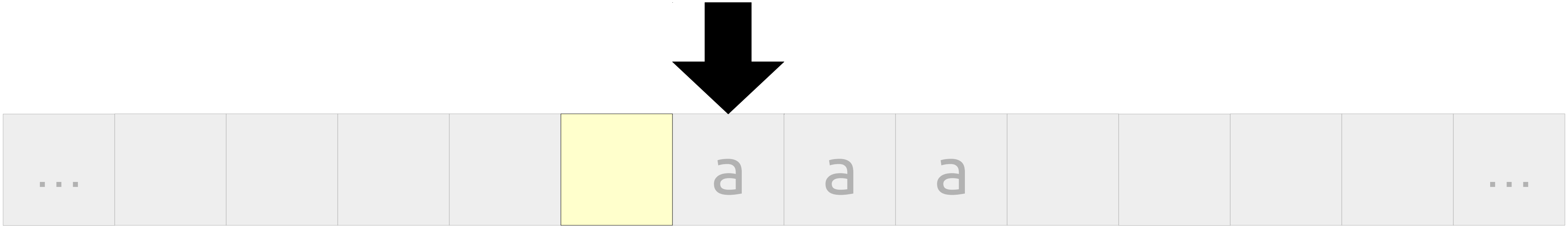
A Simple Turing Machine



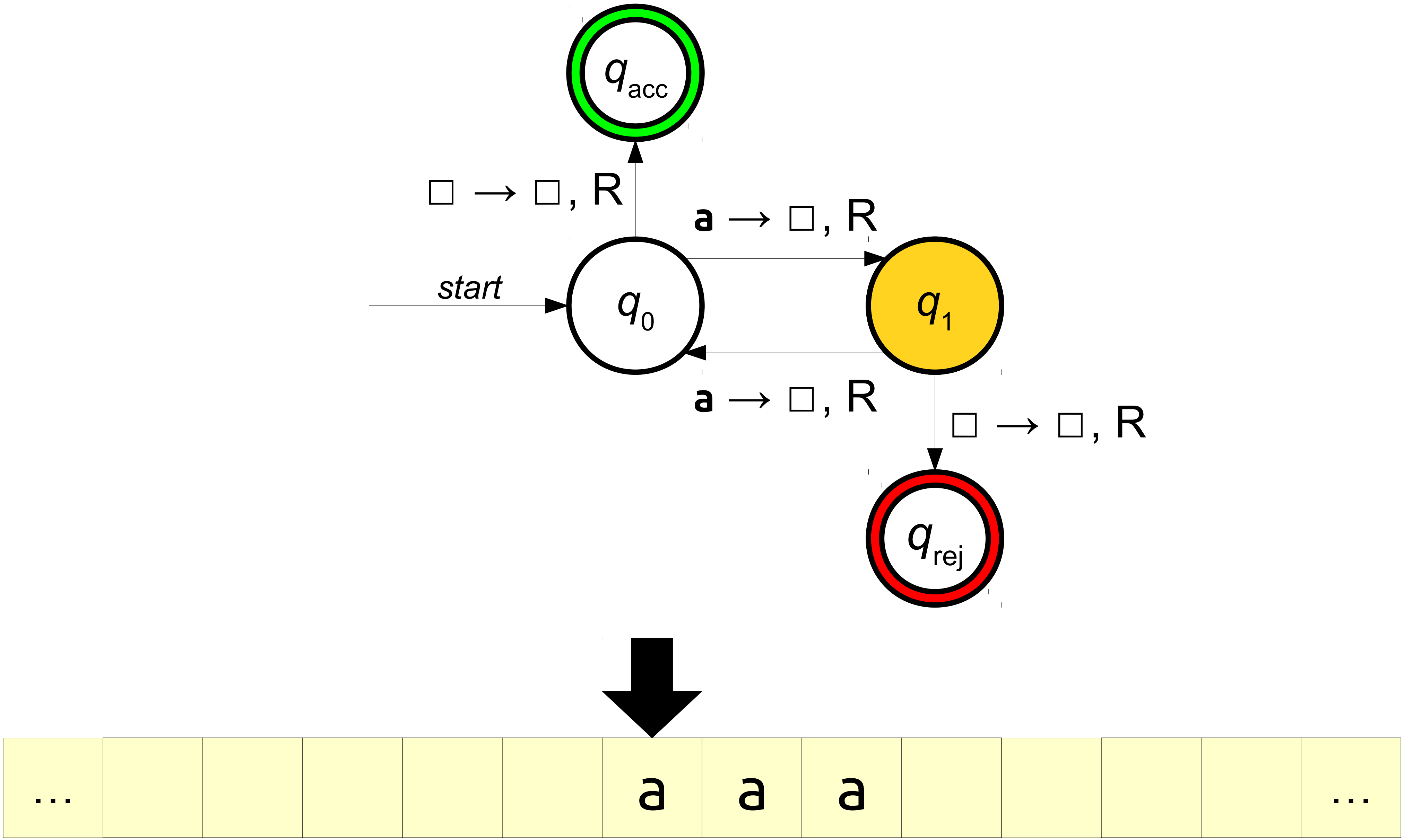
Each transition has the form

read* → *write*, *dir

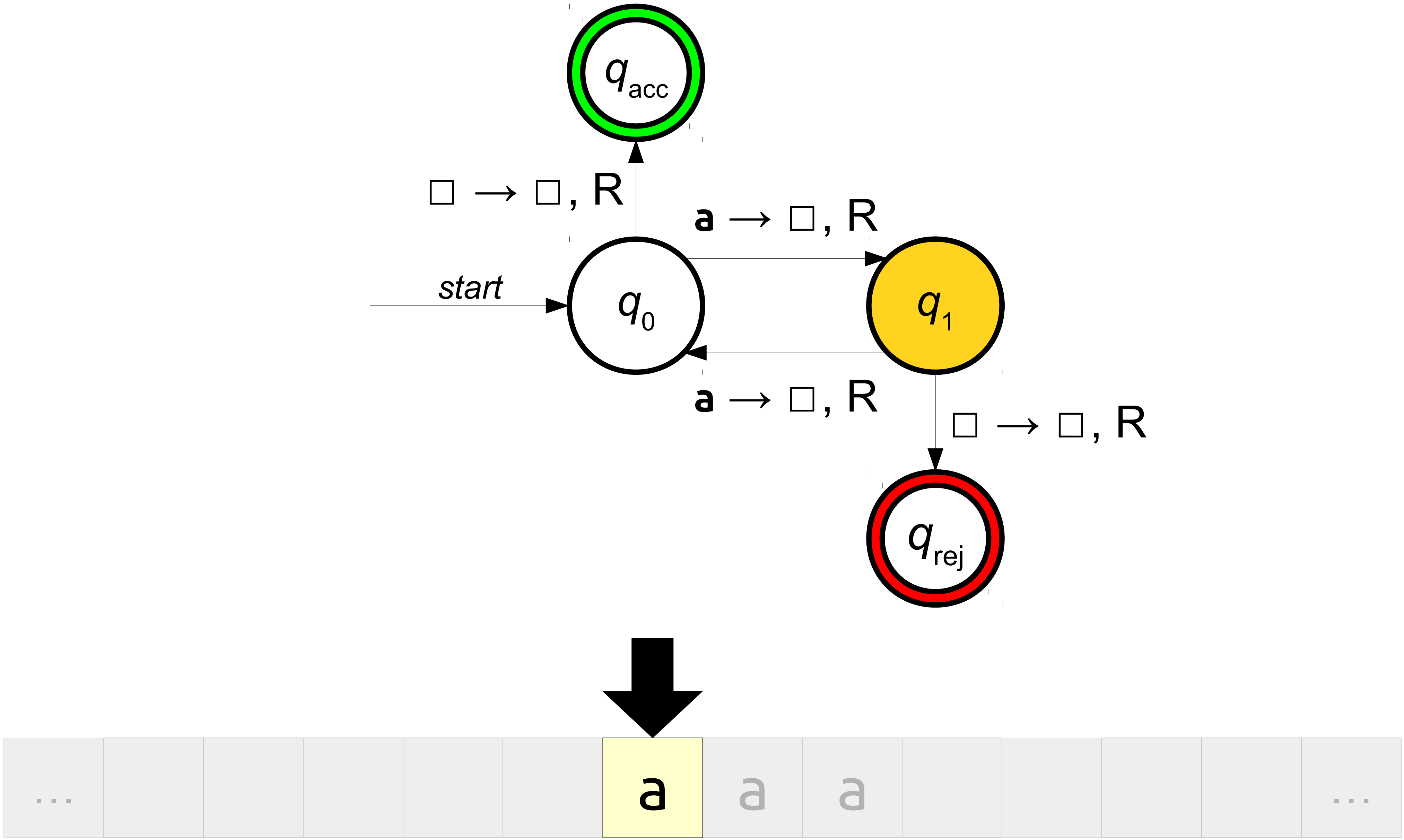
and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.



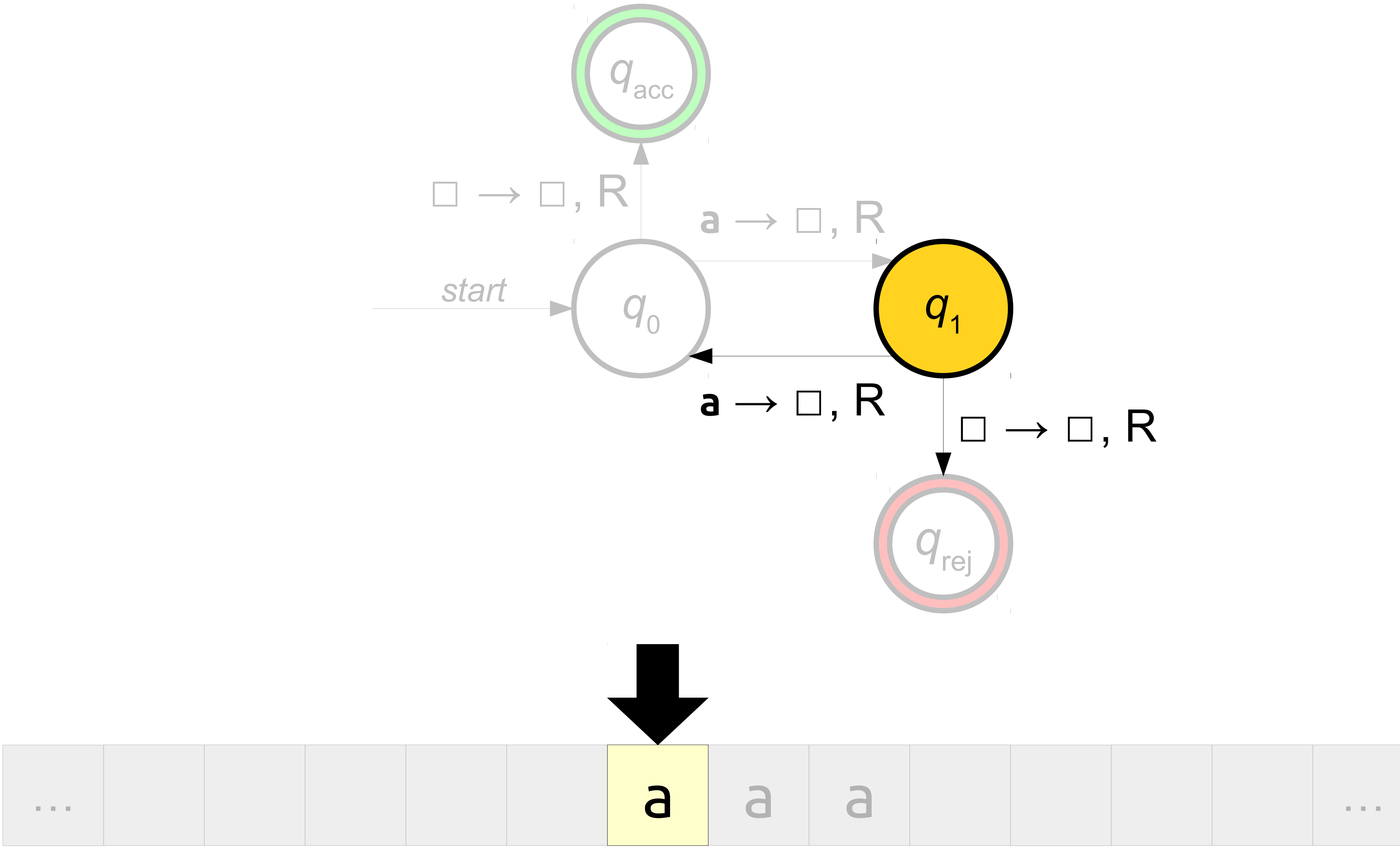
A Simple Turing Machine



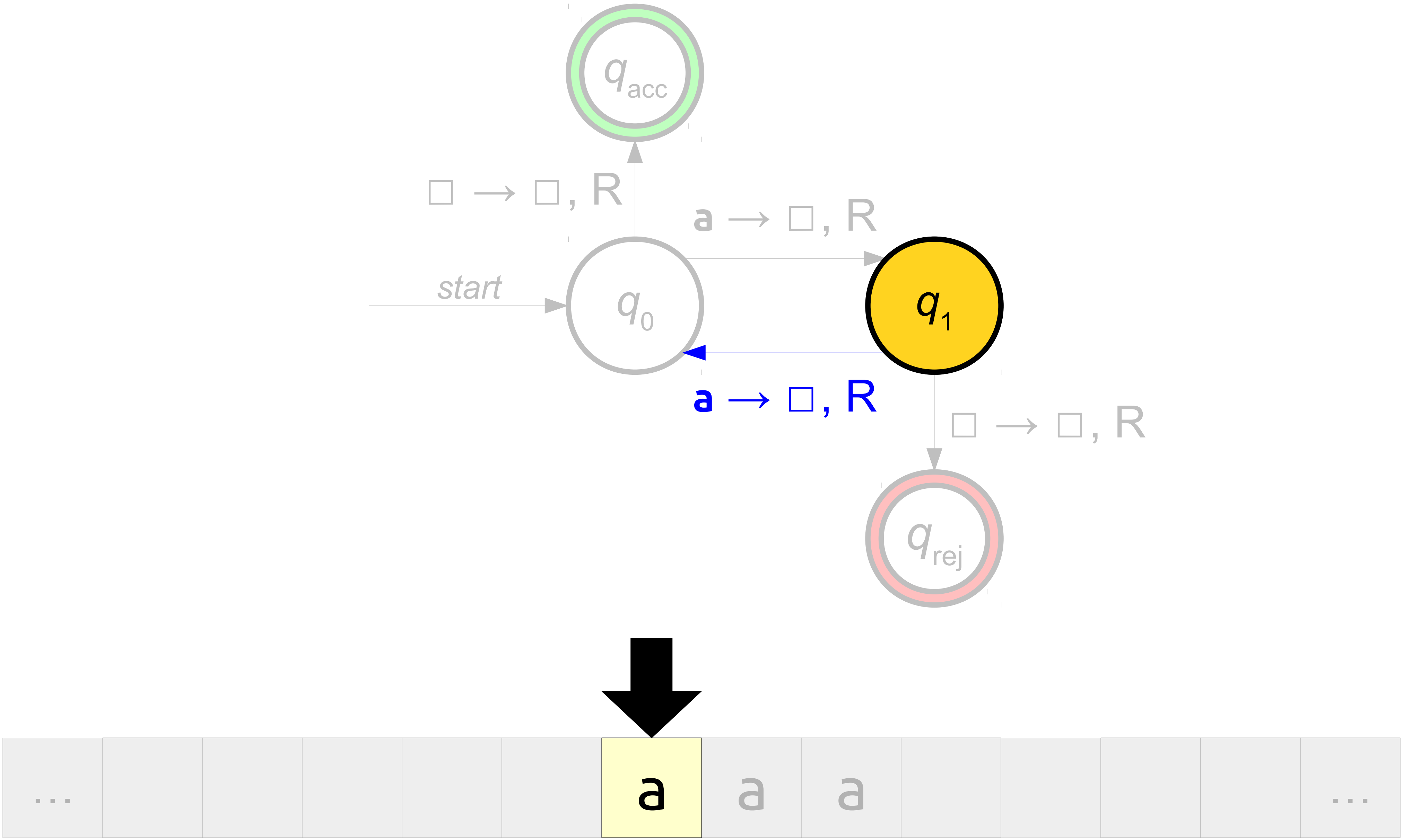
A Simple Turing Machine



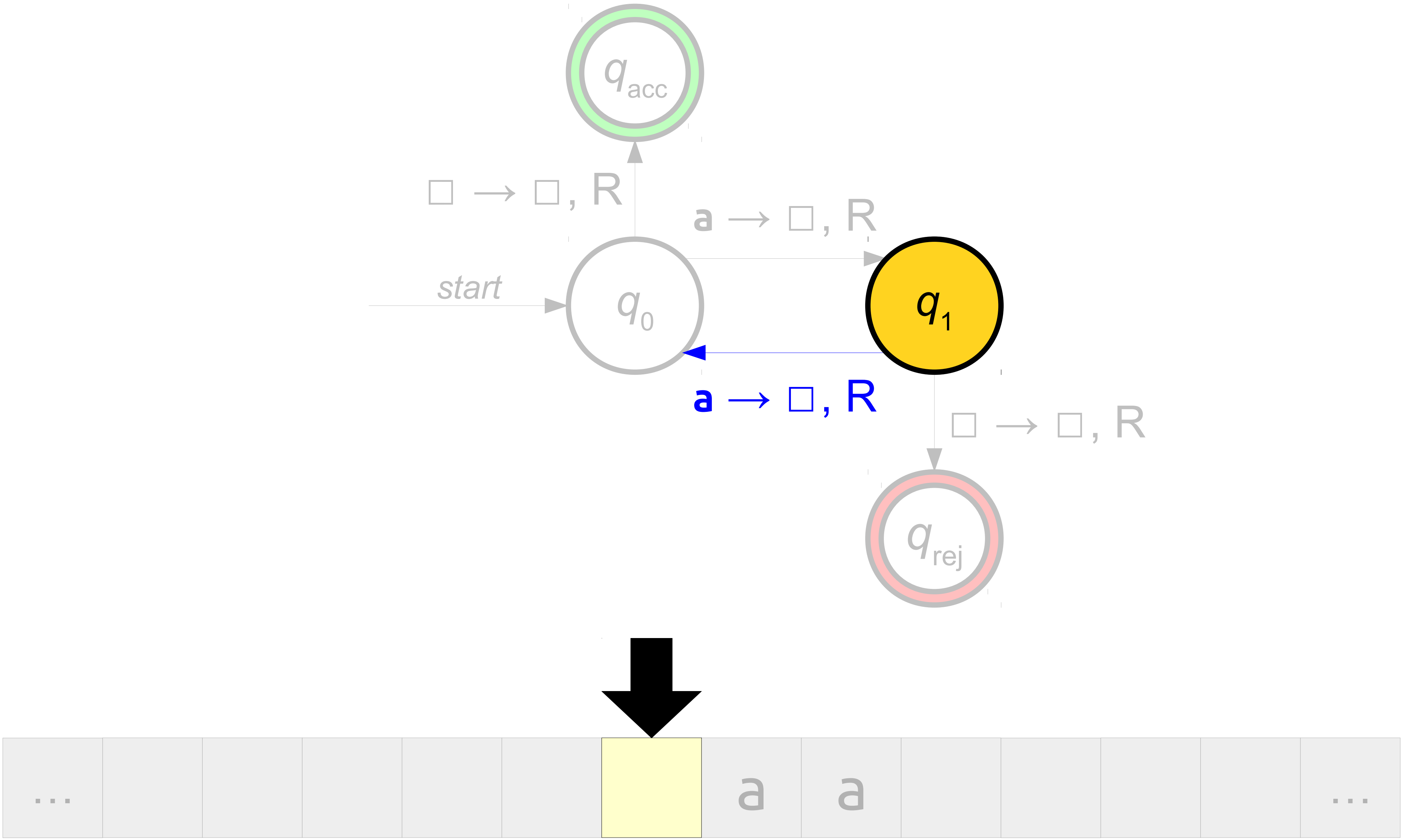
A Simple Turing Machine



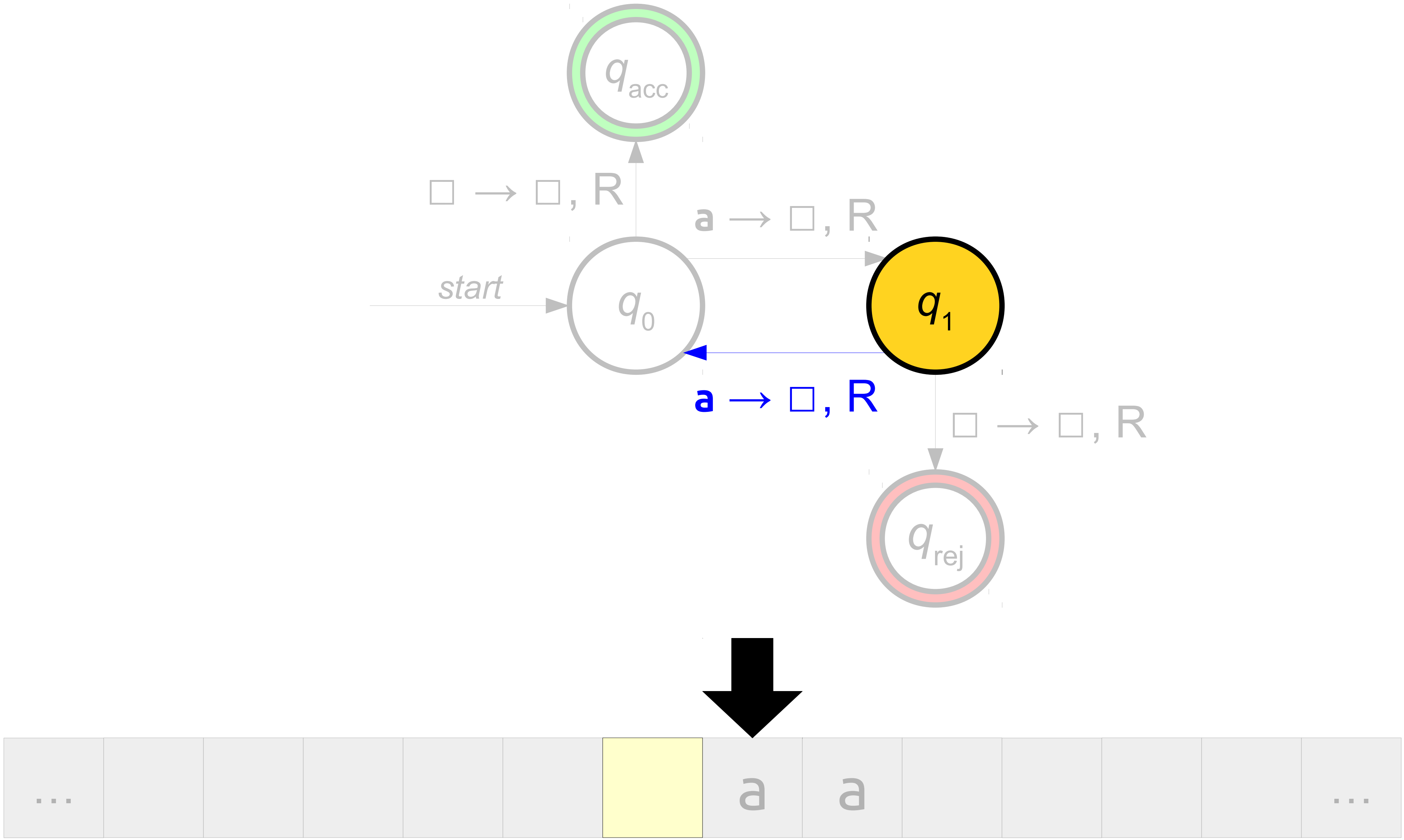
A Simple Turing Machine



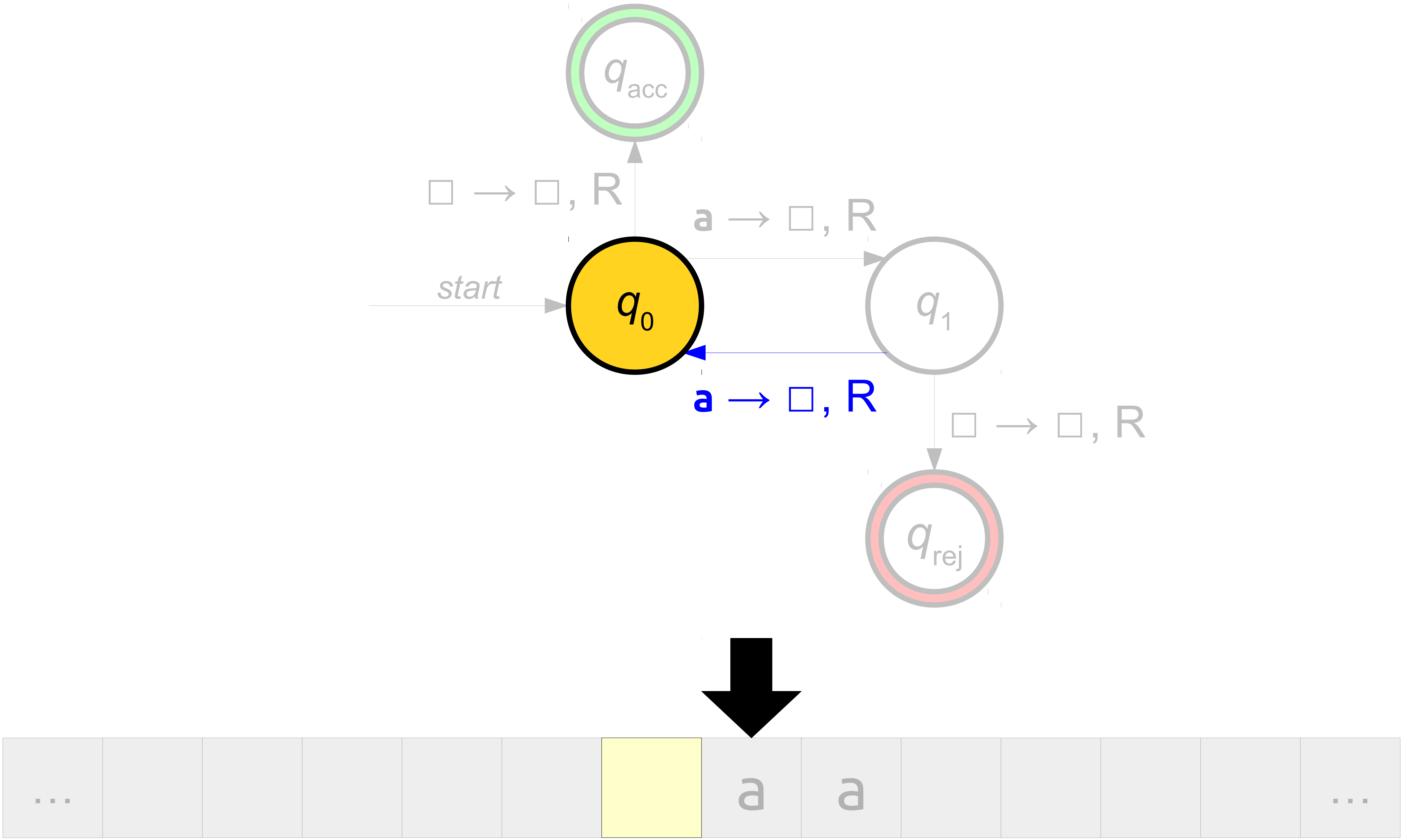
A Simple Turing Machine



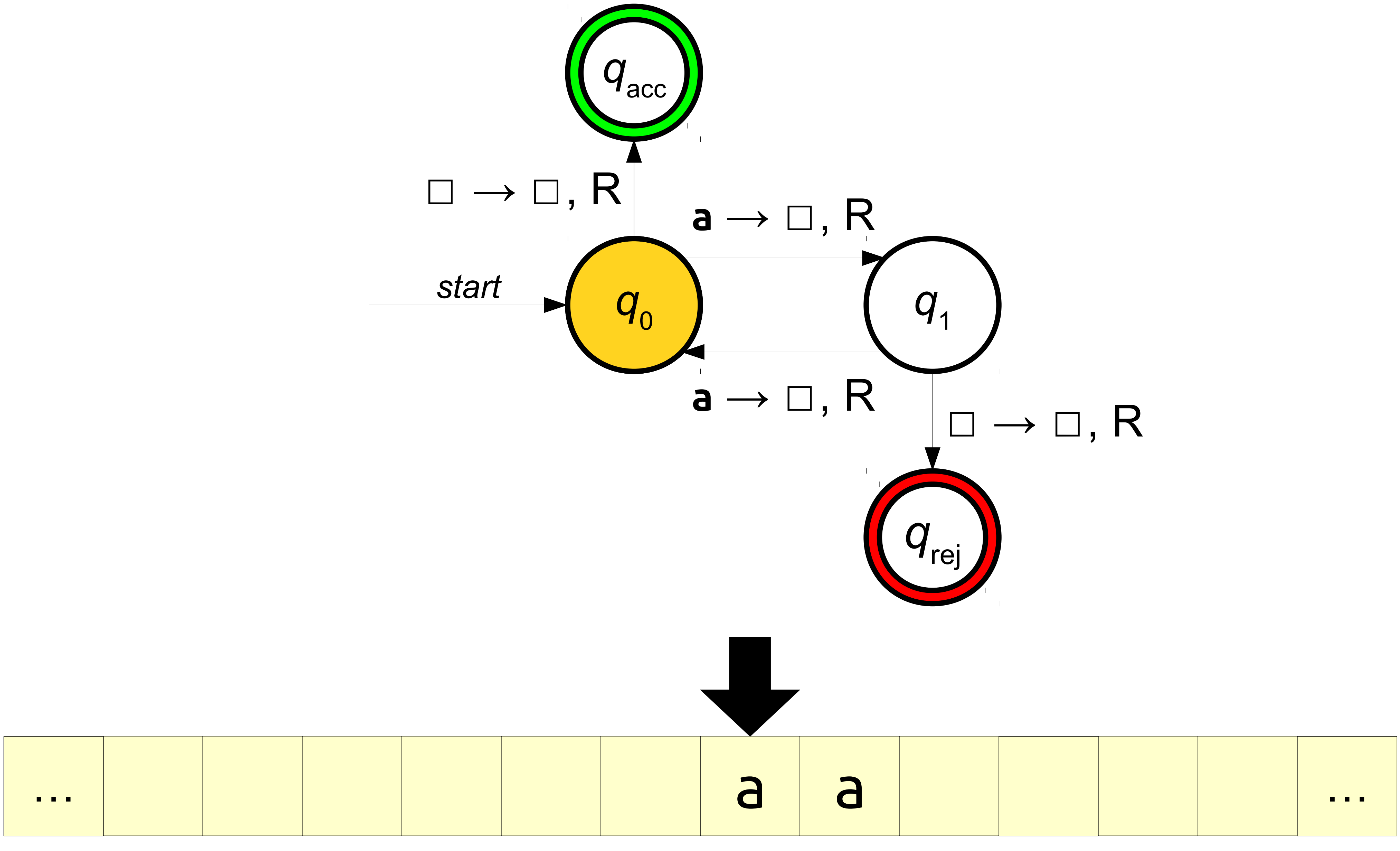
A Simple Turing Machine



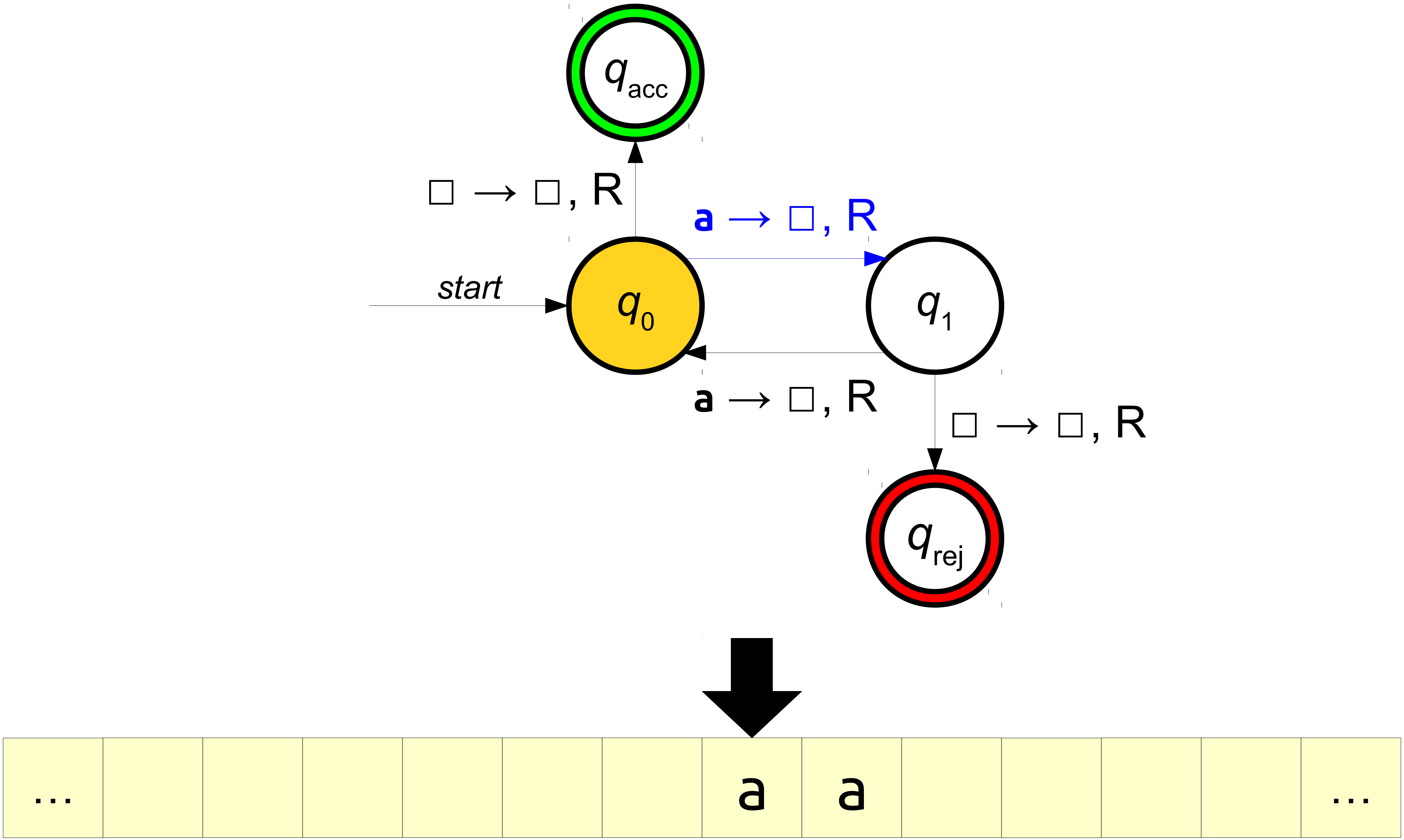
A Simple Turing Machine



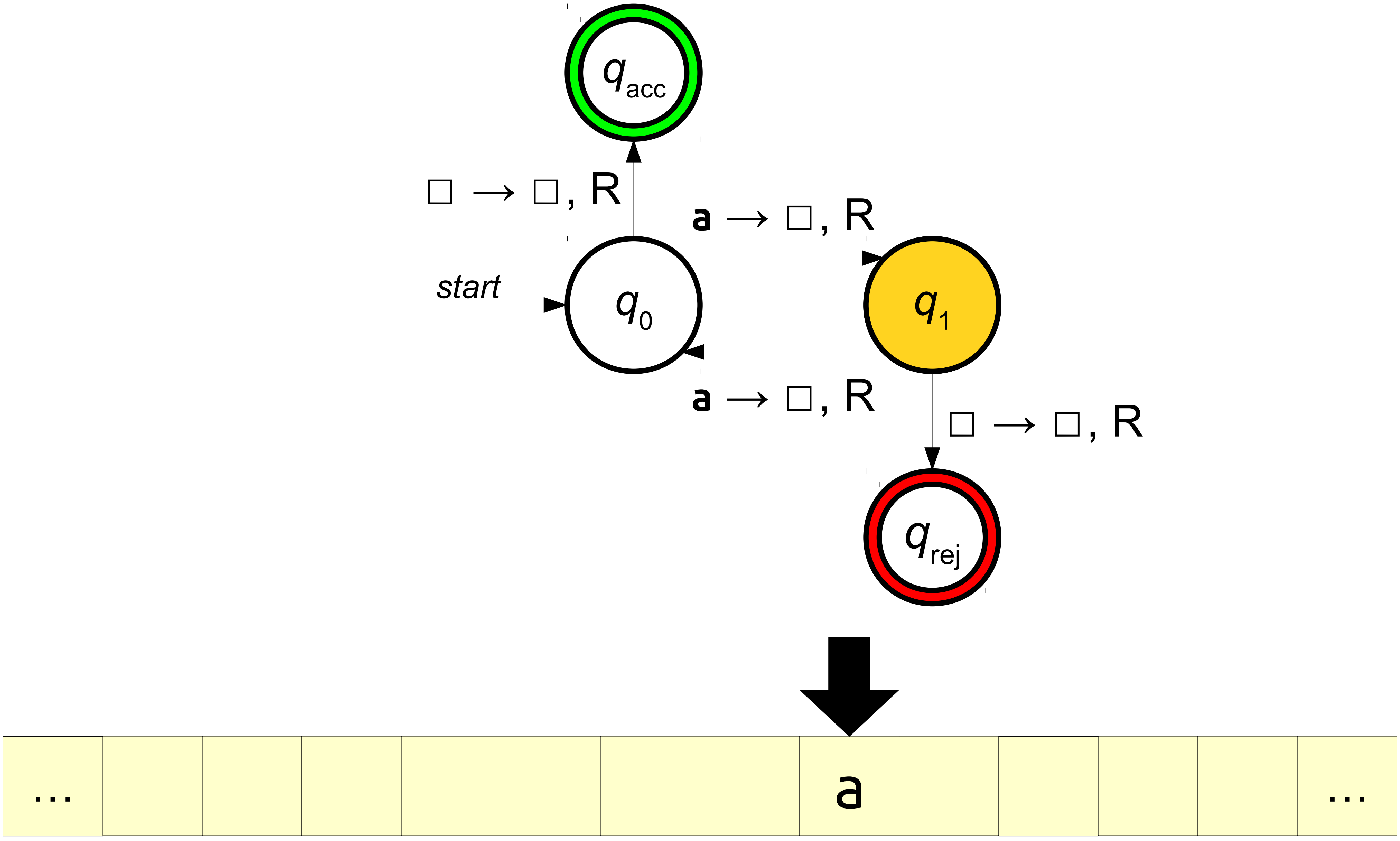
A Simple Turing Machine



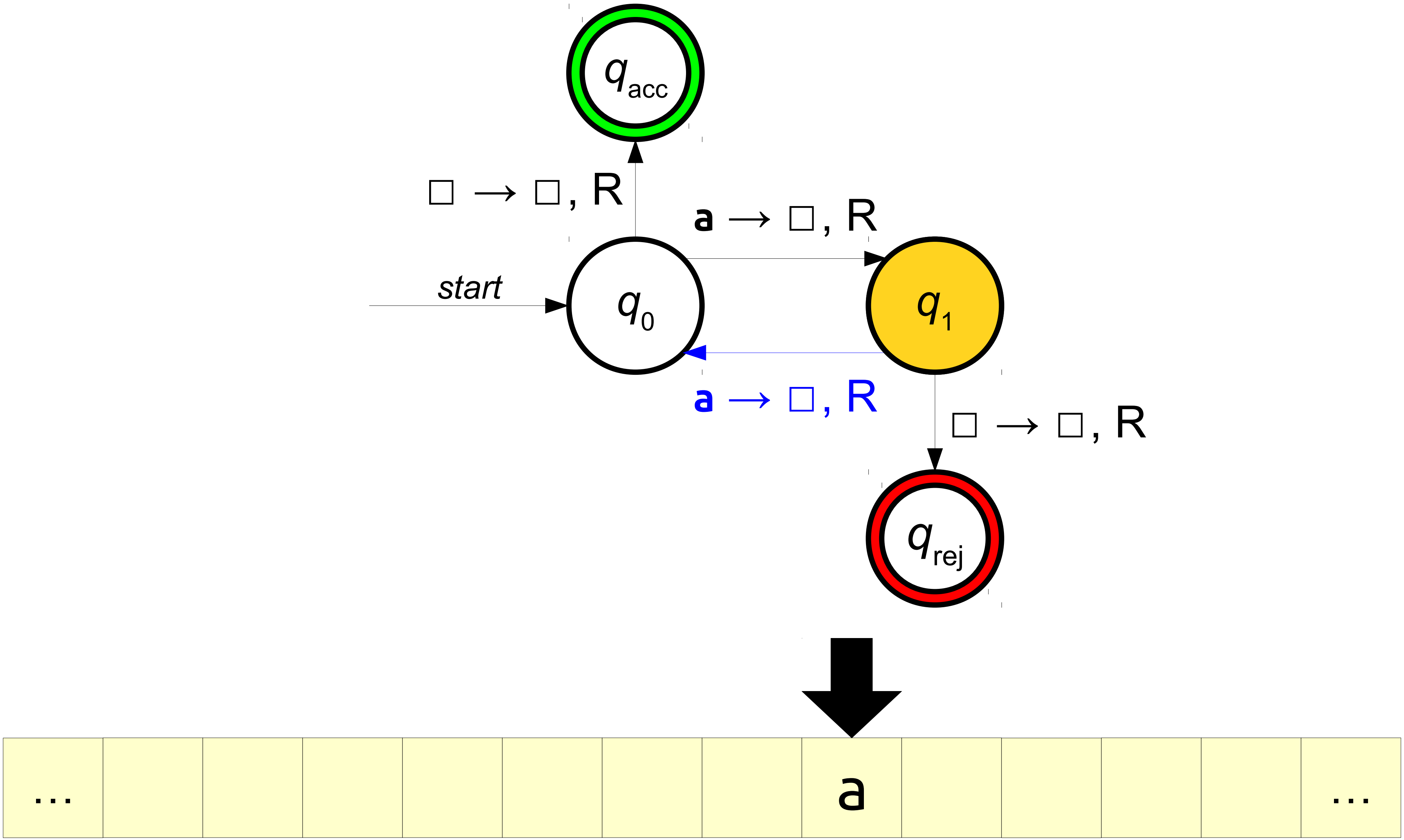
A Simple Turing Machine



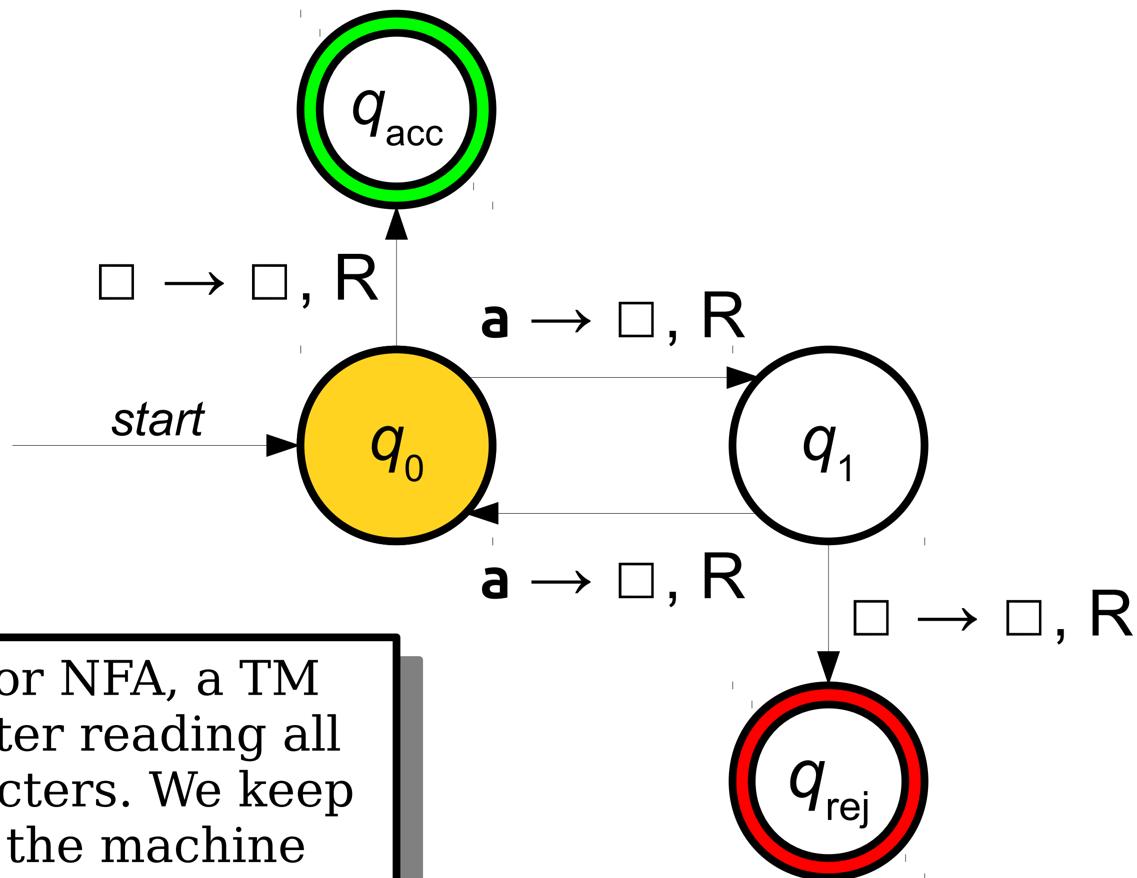
A Simple Turing Machine



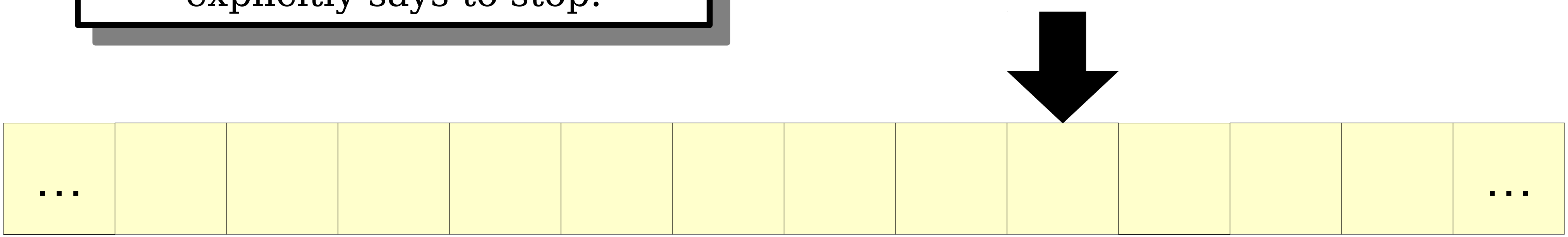
A Simple Turing Machine



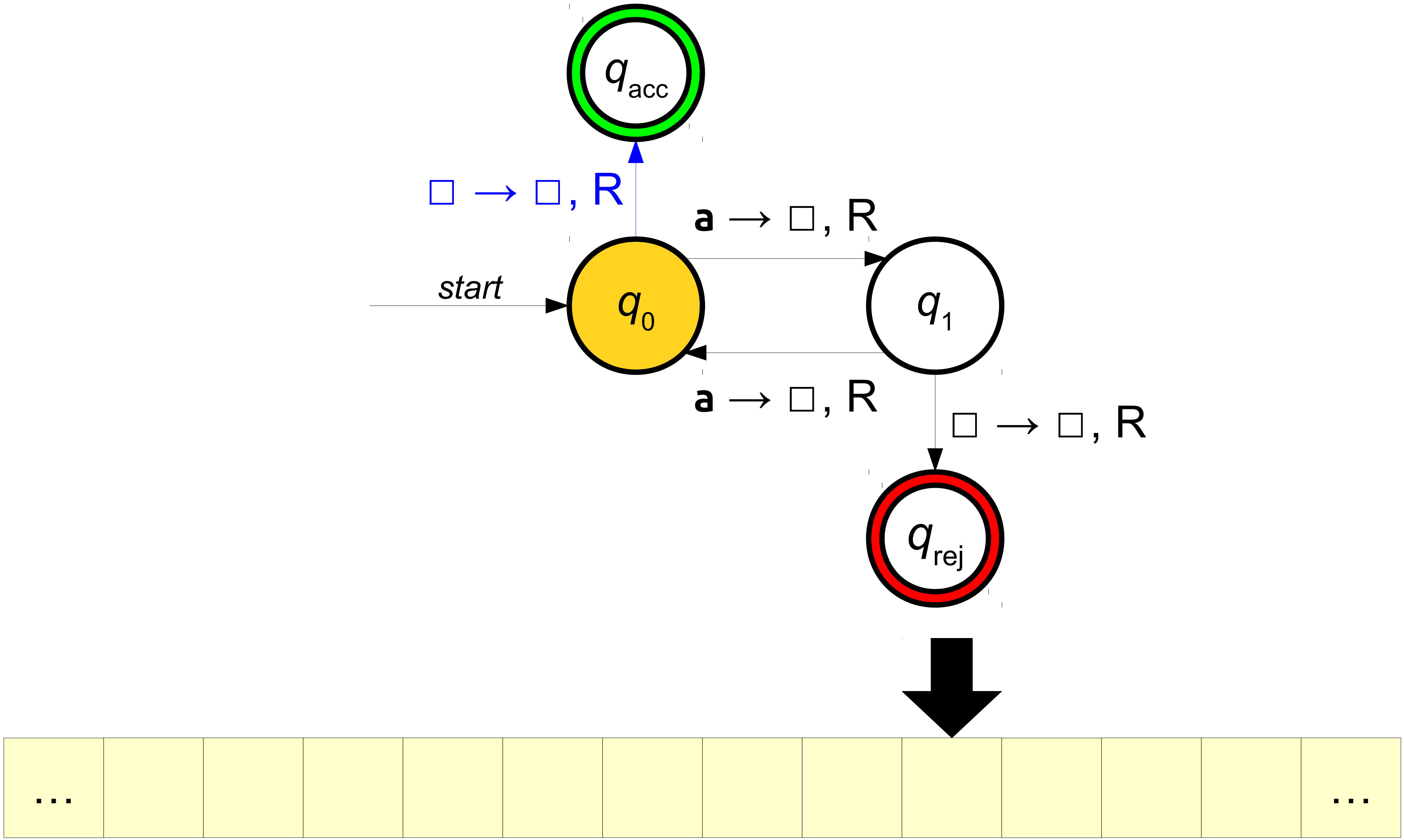
A Simple Turing Machine



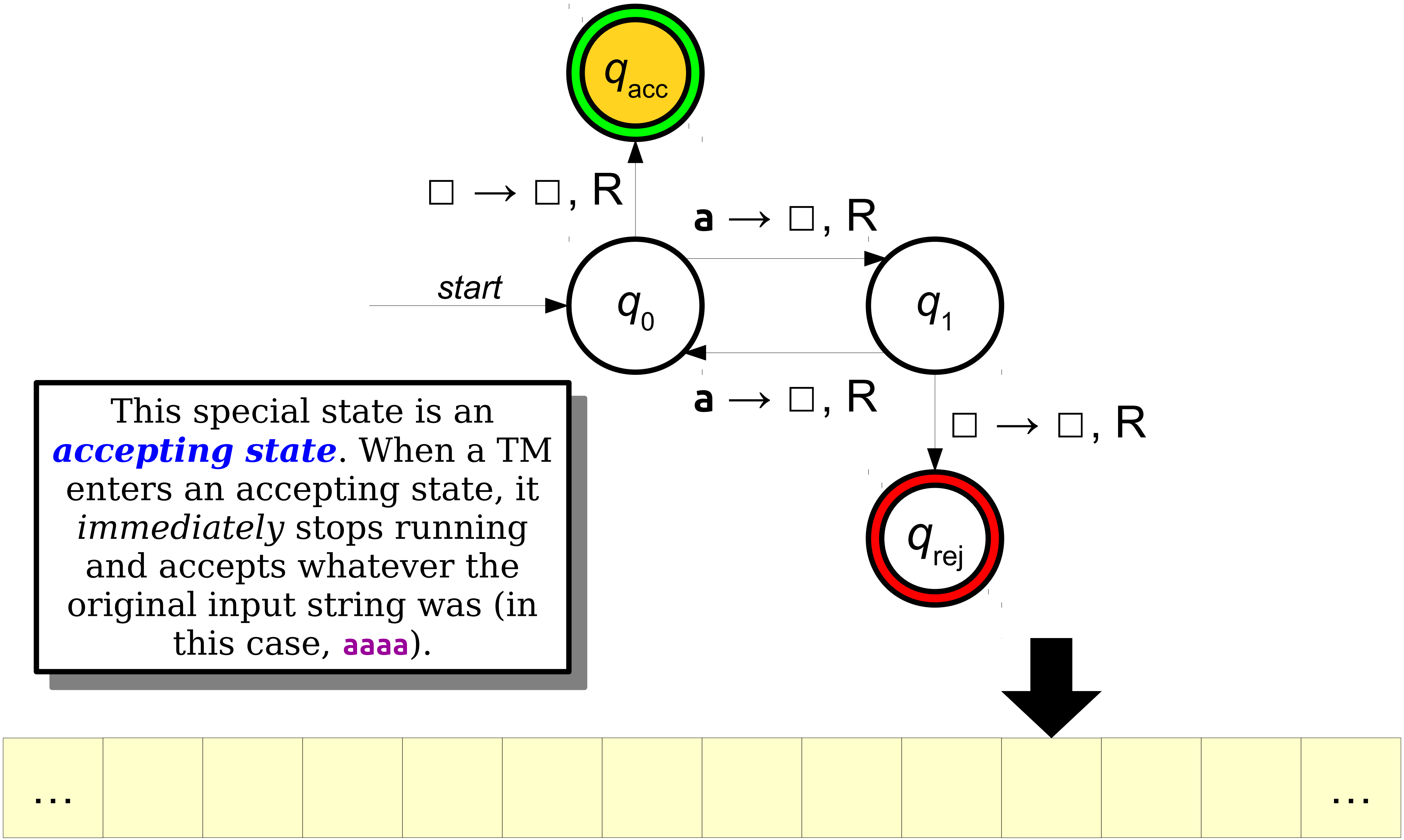
Unlike a DFA or NFA, a TM doesn't stop after reading all the input characters. We keep running until the machine explicitly says to stop.



A Simple Turing Machine

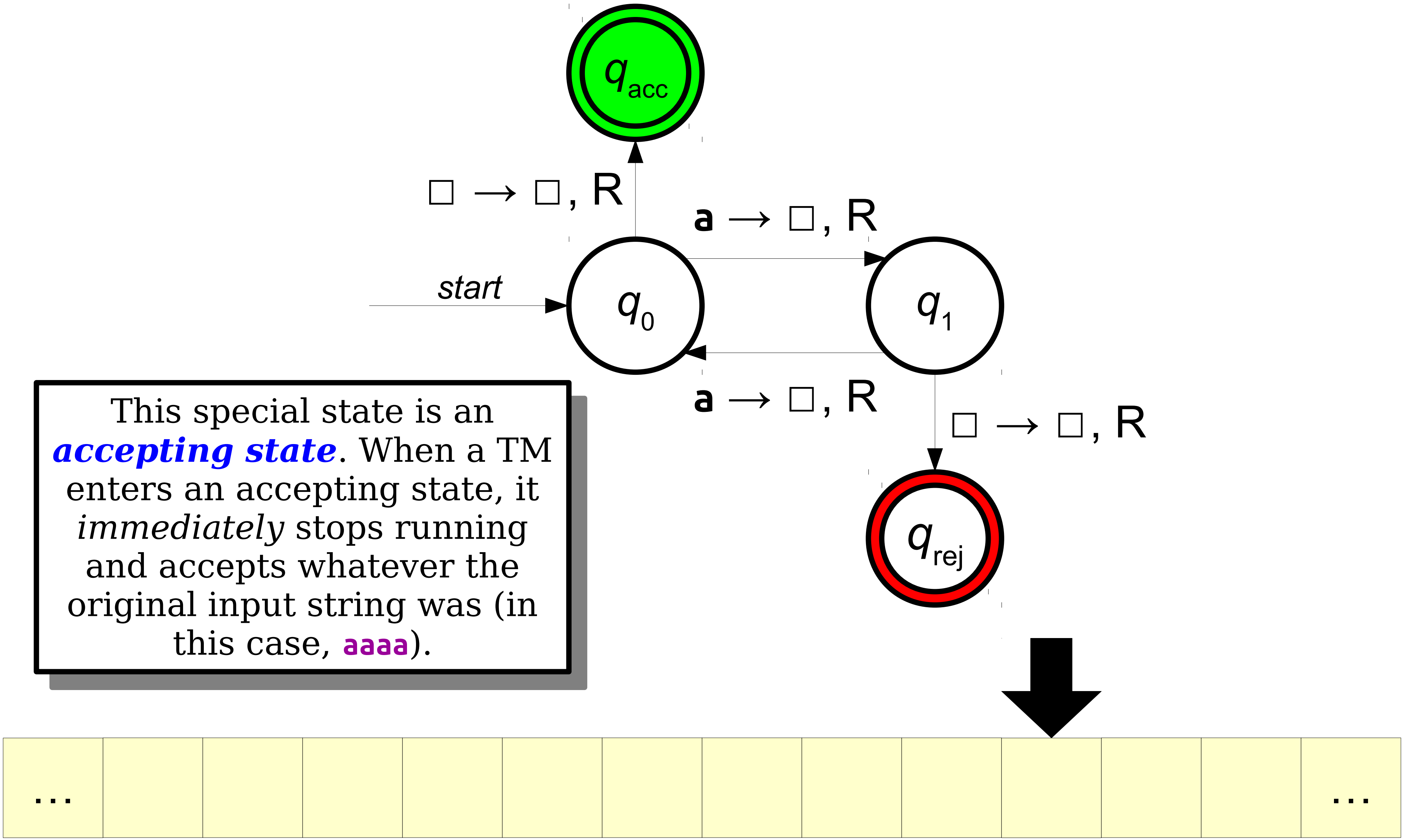


A Simple Turing Machine



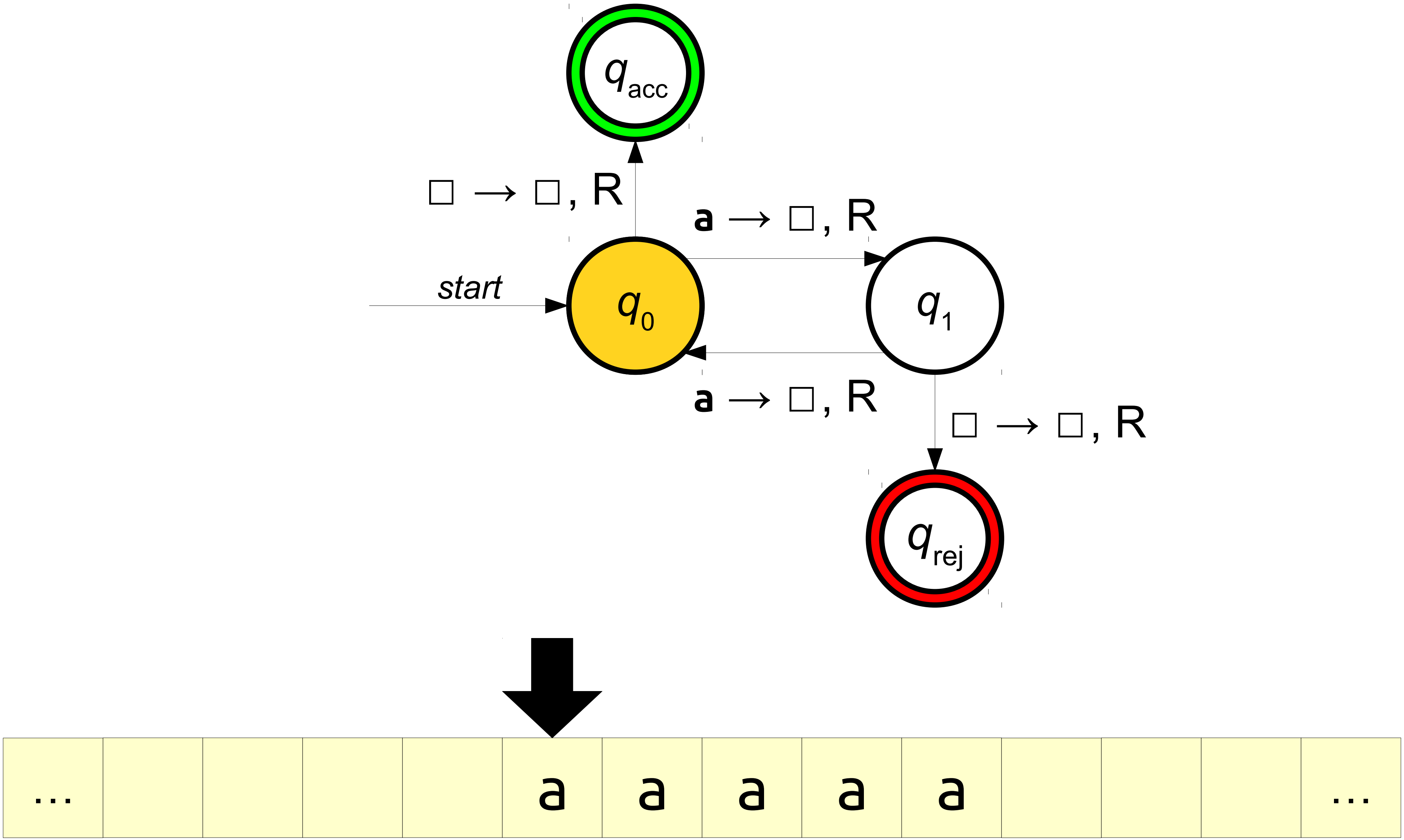
This special state is an ***accepting state***. When a TM enters an accepting state, it *immediately* stops running and accepts whatever the original input string was (in this case, **aaaa**).

A Simple Turing Machine

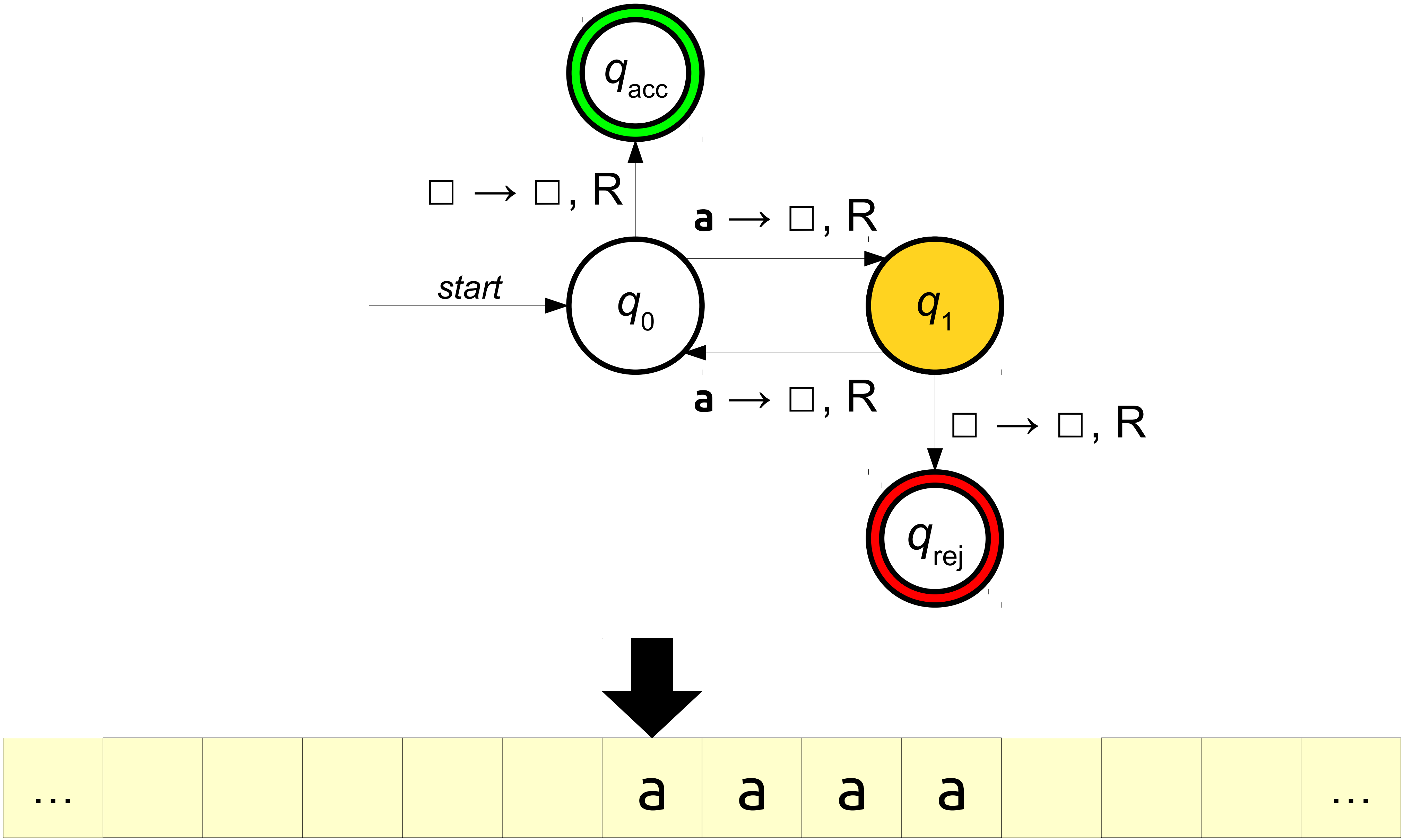


This special state is an ***accepting state***. When a TM enters an accepting state, it *immediately* stops running and accepts whatever the original input string was (in this case, **aaaa**).

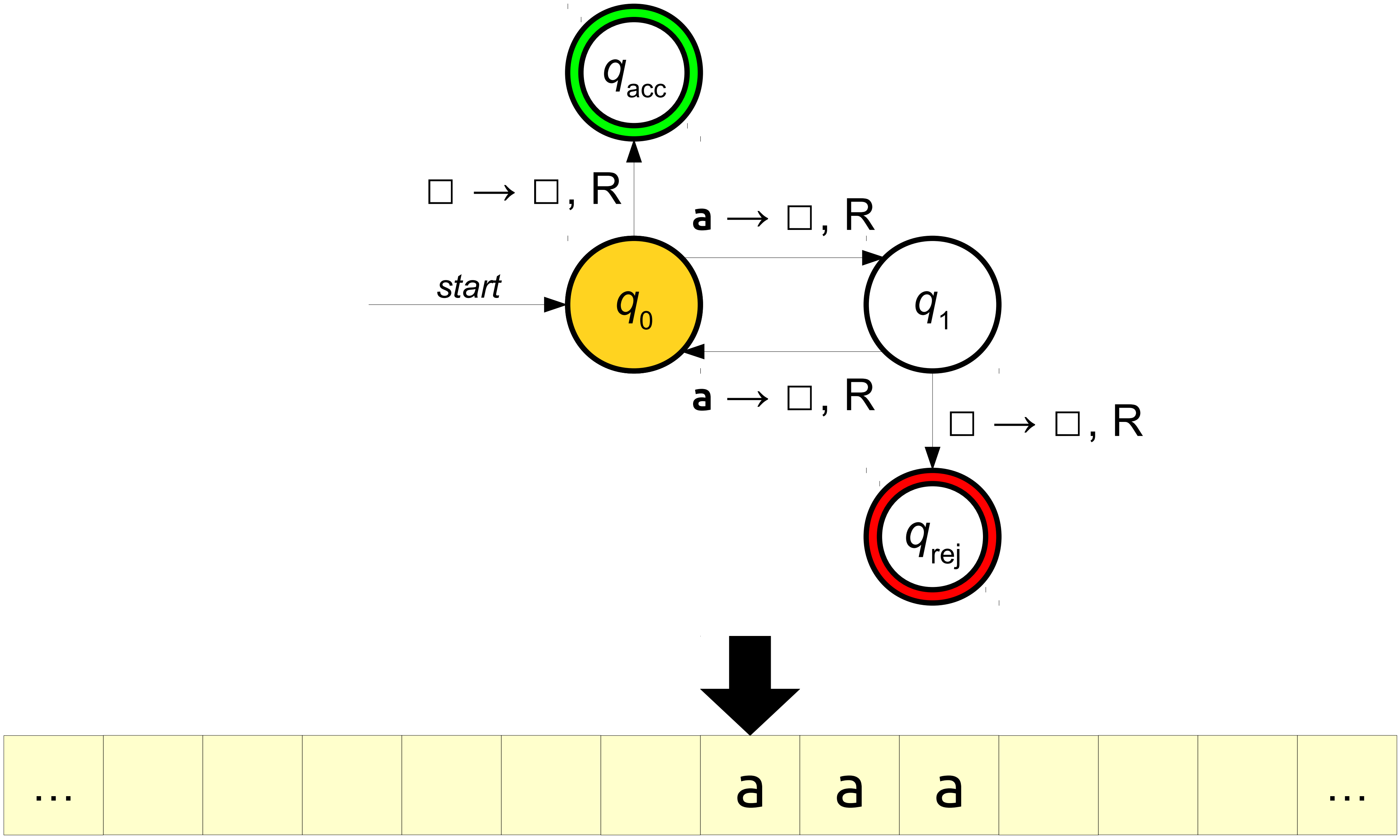
A Simple Turing Machine



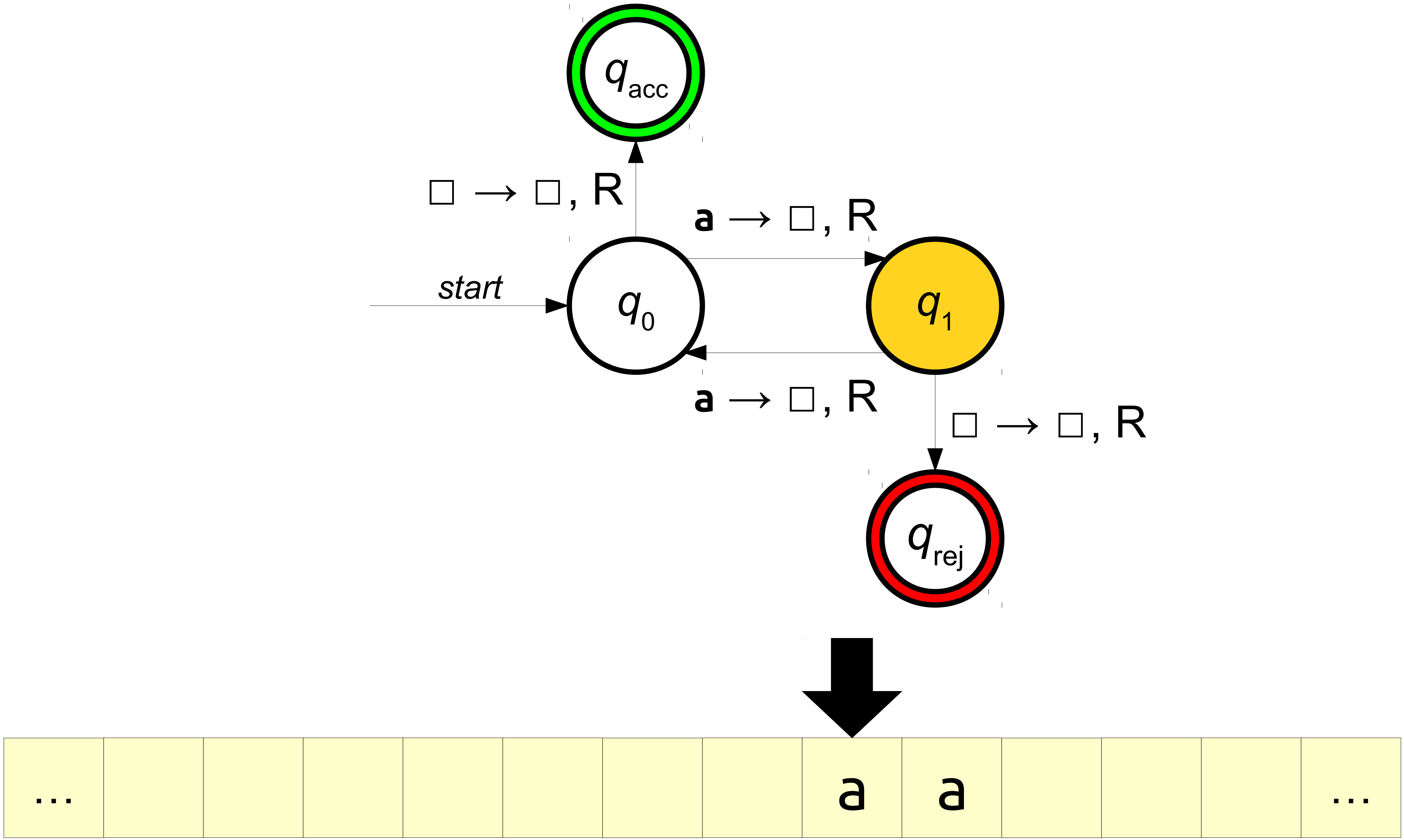
A Simple Turing Machine



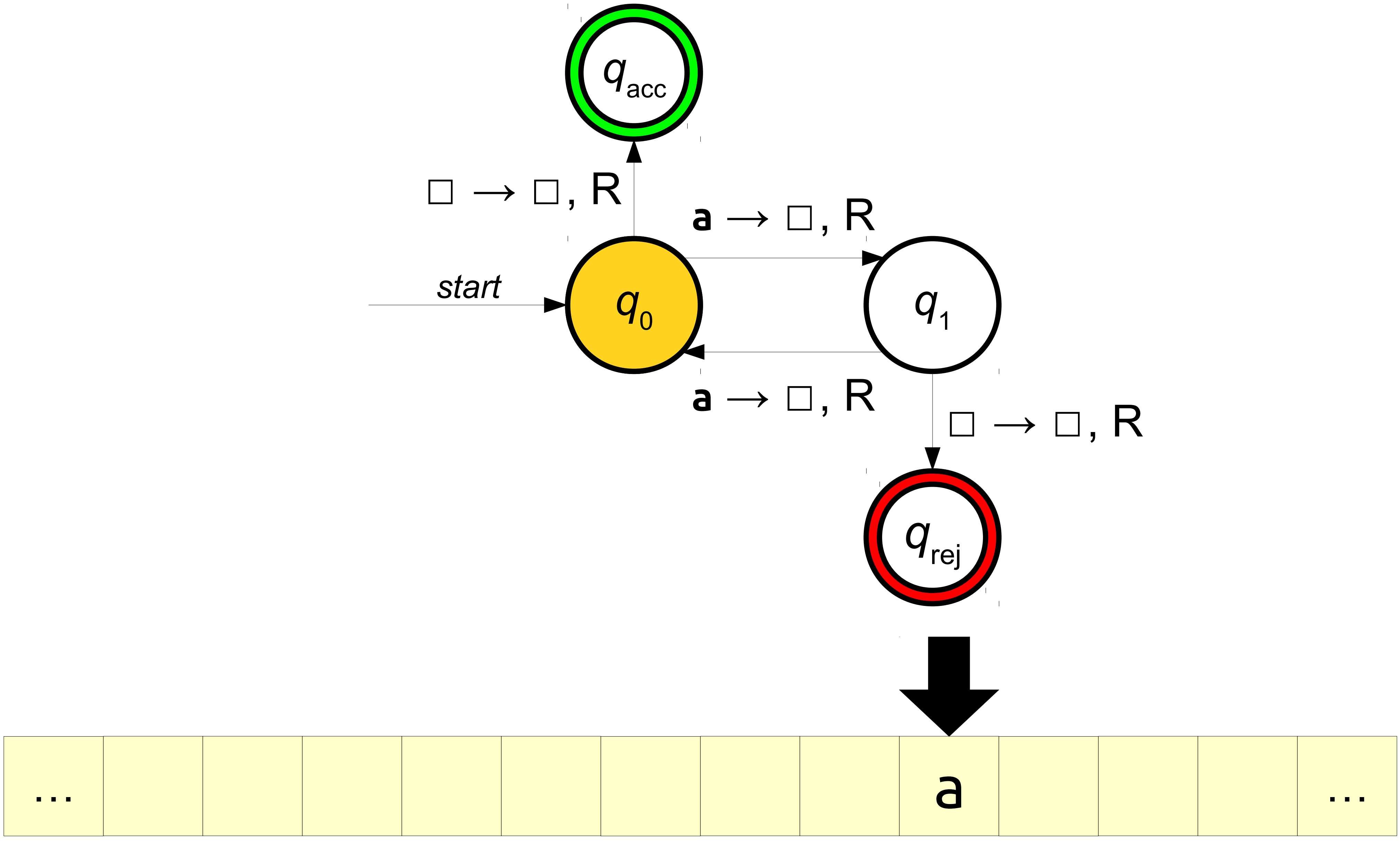
A Simple Turing Machine



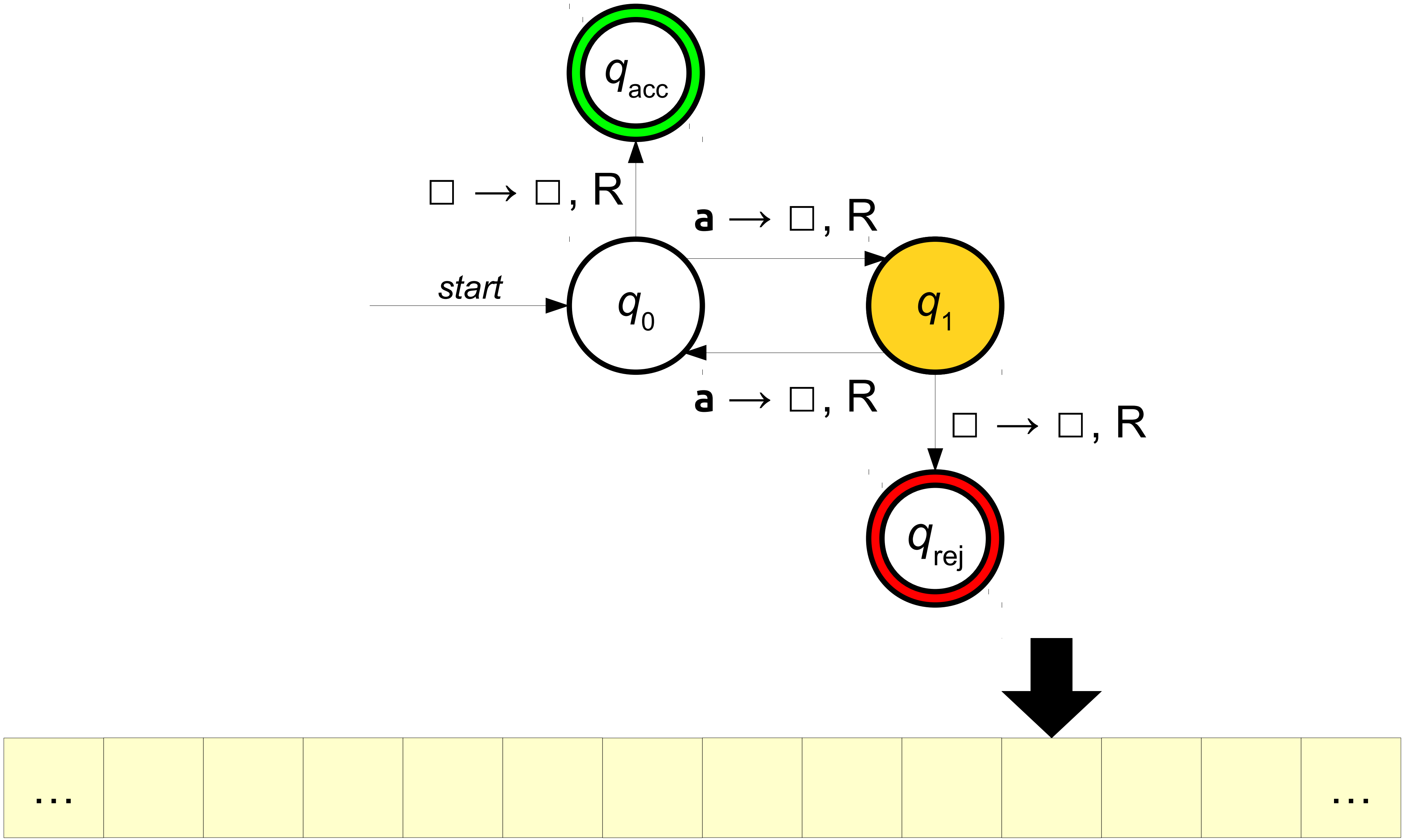
A Simple Turing Machine



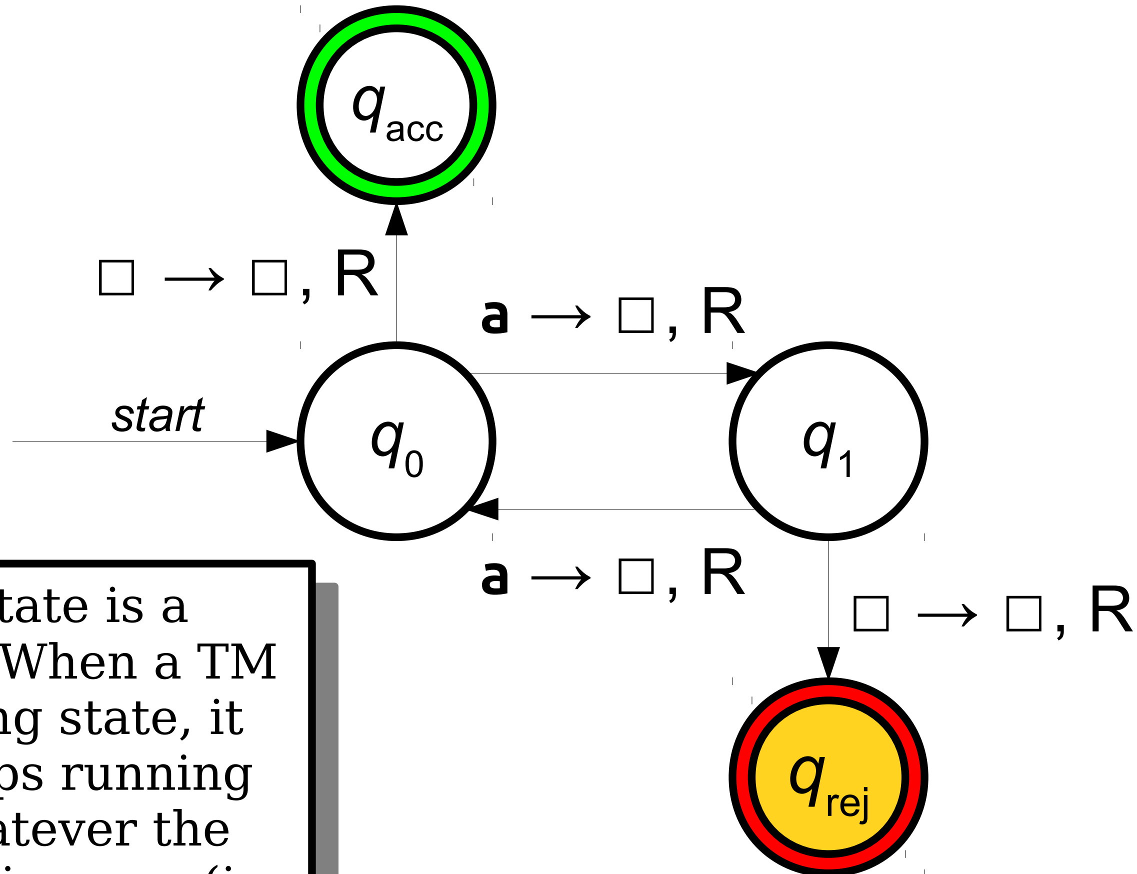
A Simple Turing Machine



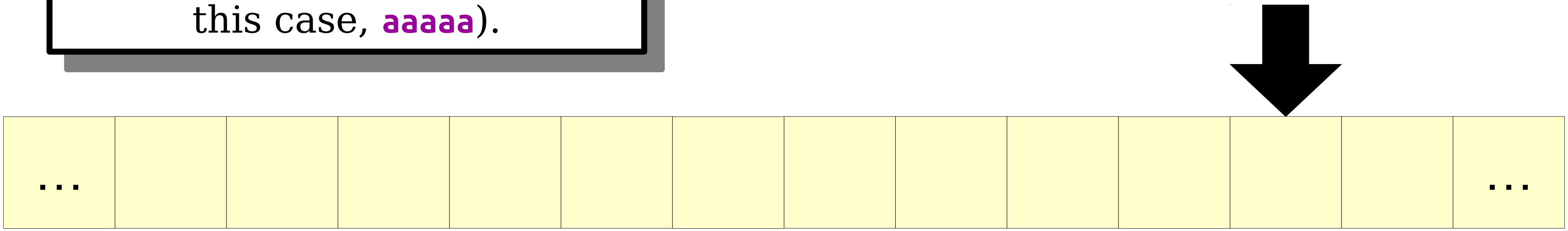
A Simple Turing Machine



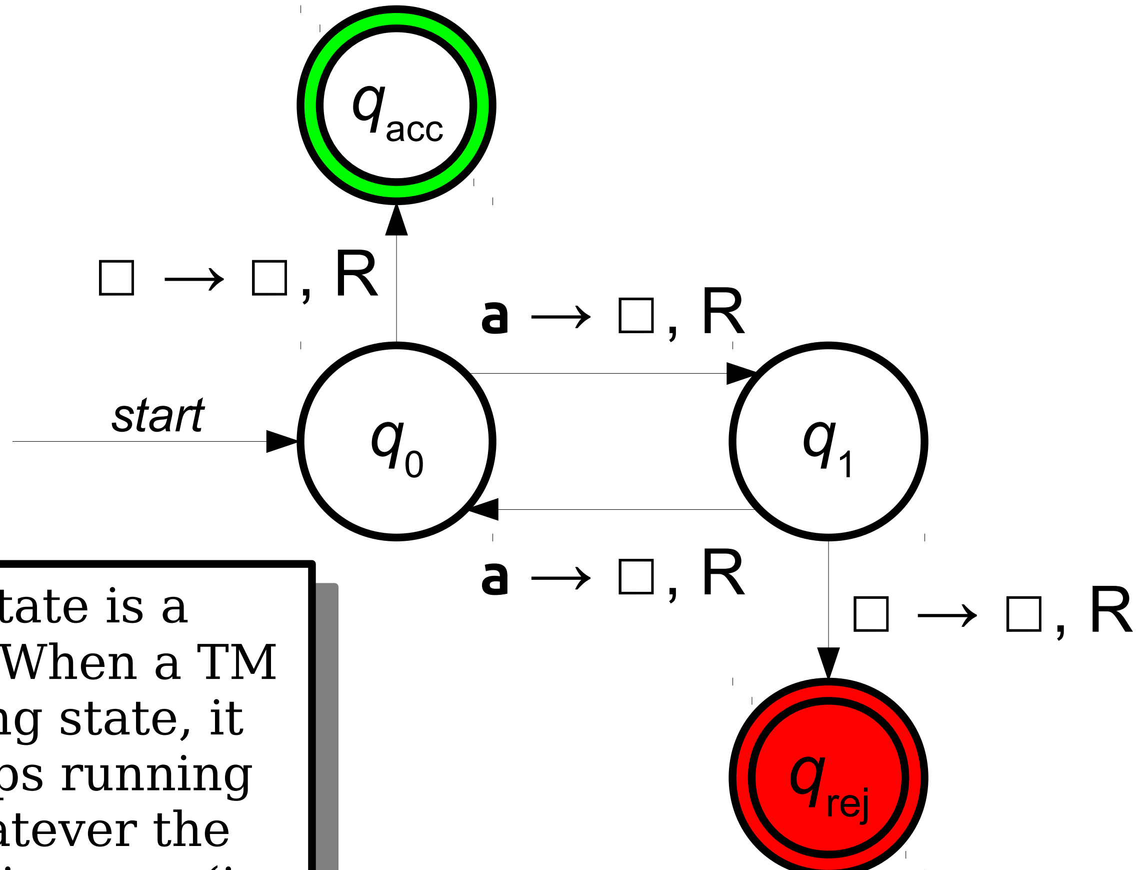
A Simple Turing Machine



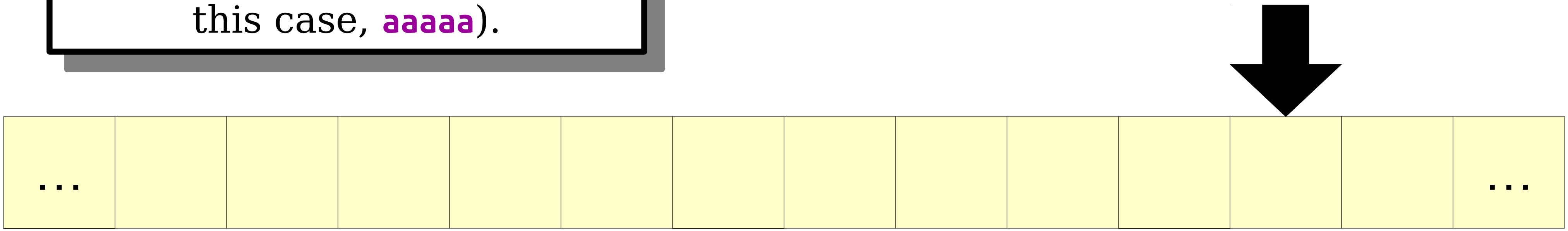
This special state is a ***rejecting state***. When a TM enters a rejecting state, it *immediately* stops running and rejects whatever the original input string was (in this case, **aaaaa**).



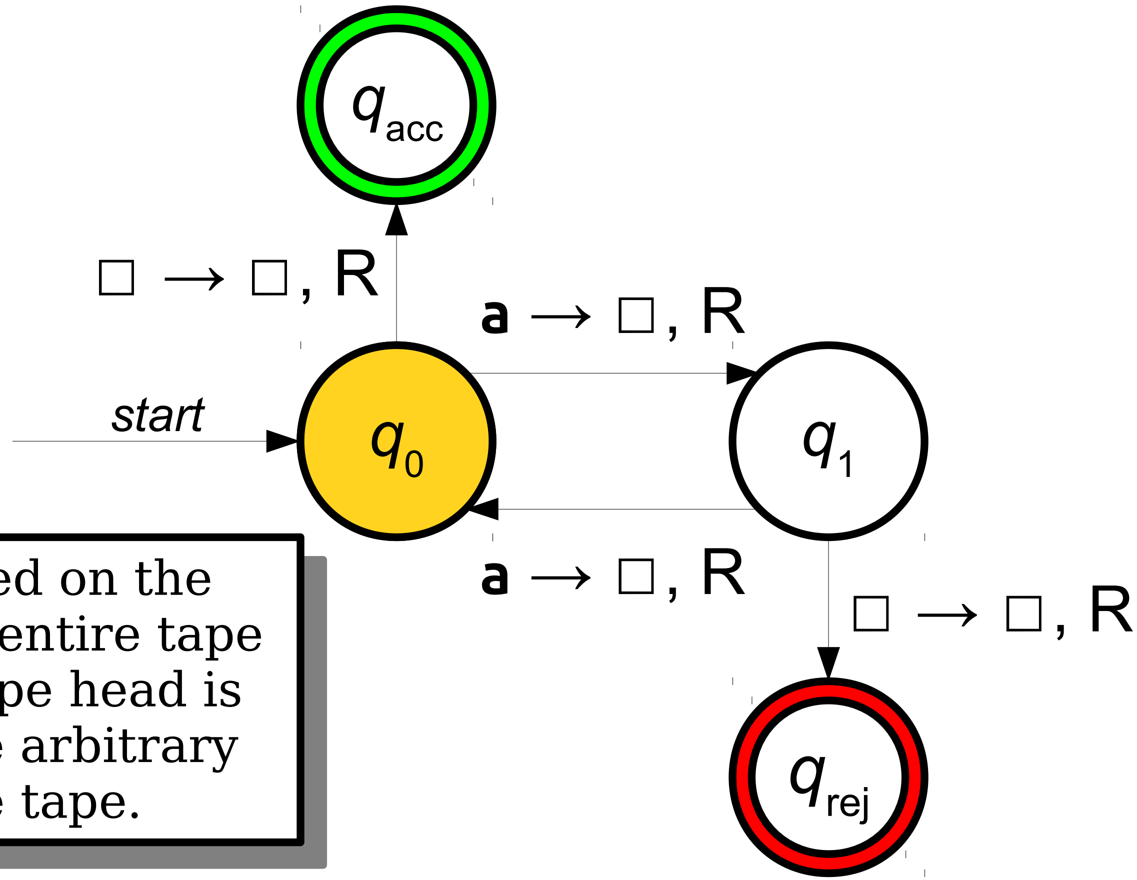
A Simple Turing Machine



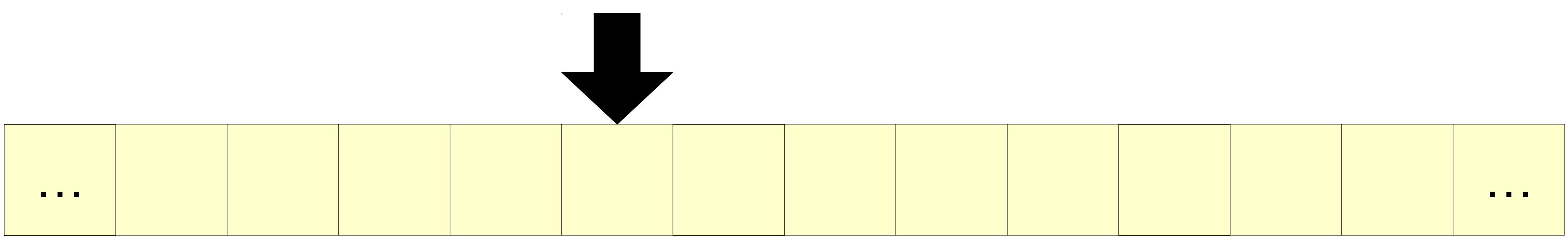
This special state is a ***rejecting state***. When a TM enters a rejecting state, it *immediately* stops running and rejects whatever the original input string was (in this case, **aaaaa**).



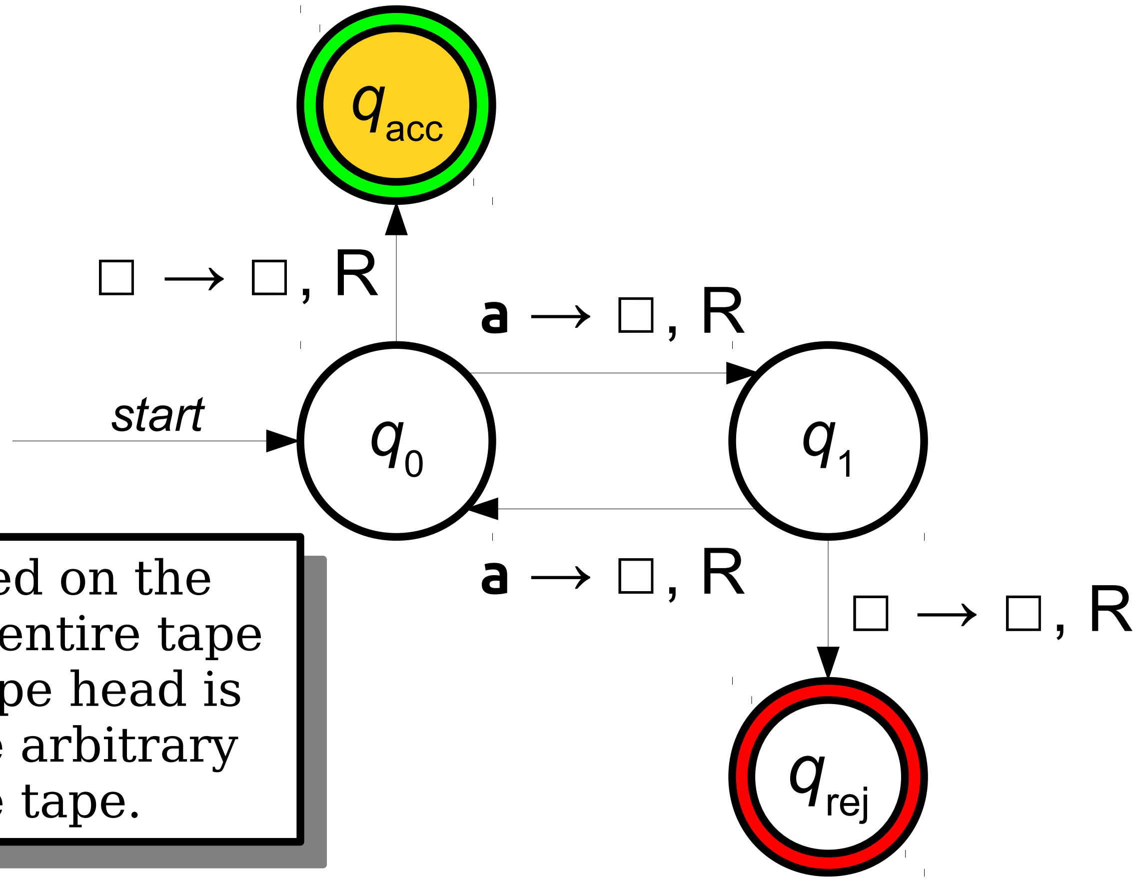
A Simple Turing Machine



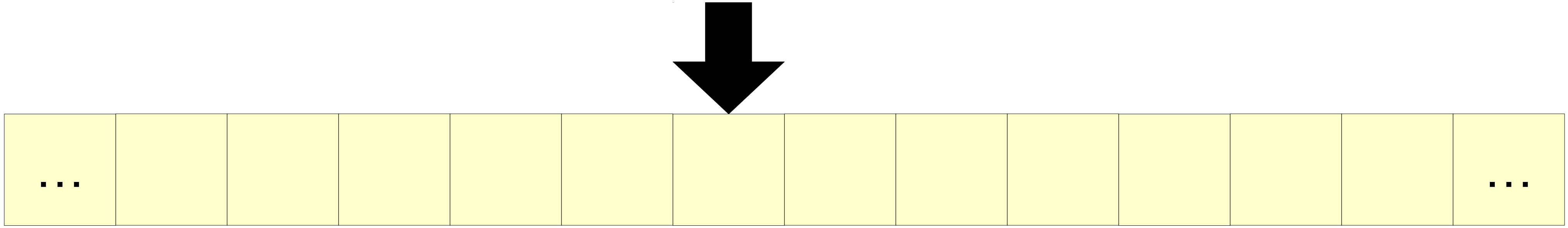
If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.



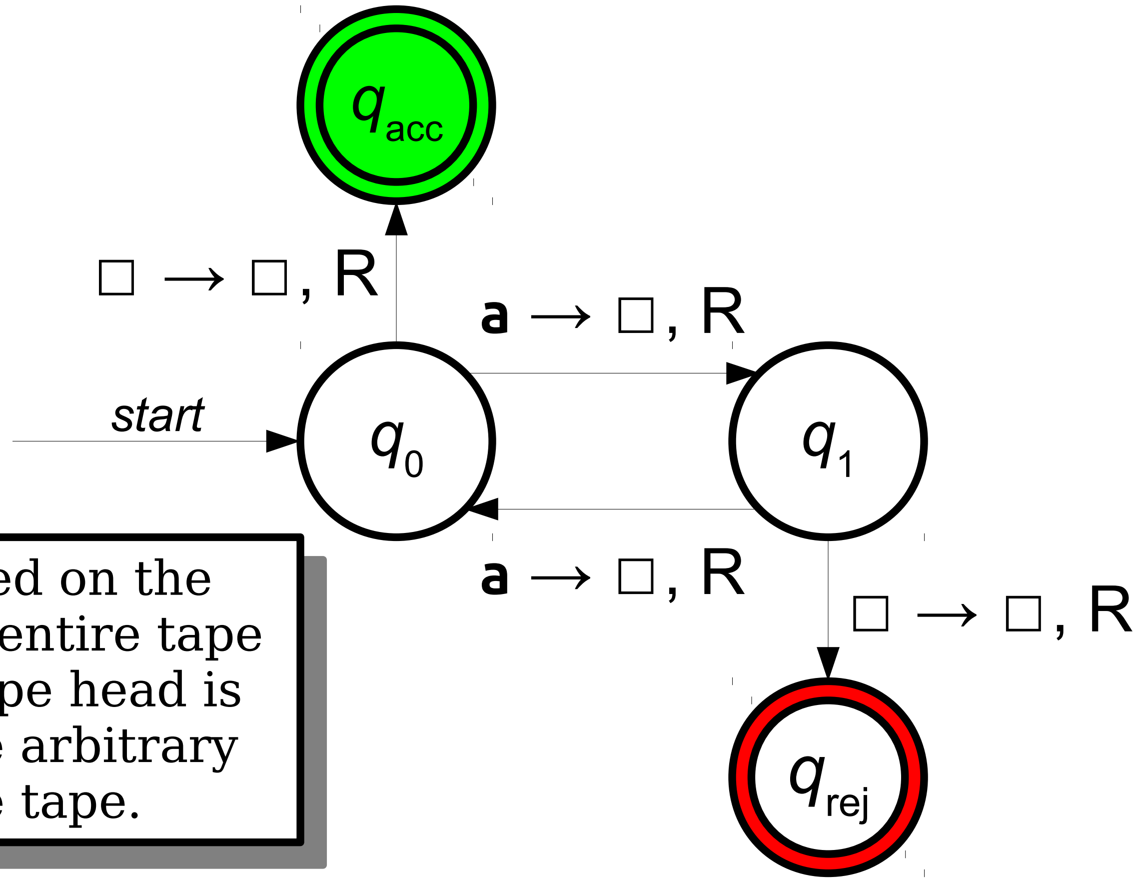
A Simple Turing Machine



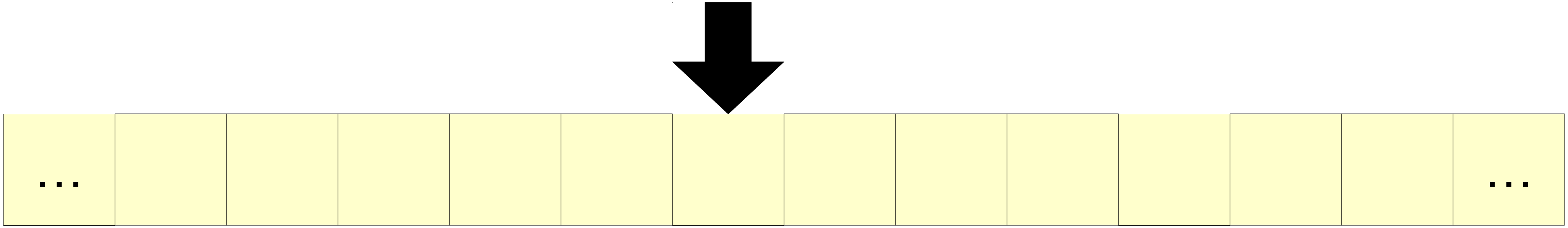
If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.



A Simple Turing Machine



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



The Turing Machine

- A Turing machine consists of three parts:
  - A ***finite-state control*** that issues commands,
  - an ***infinite tape*** for input and scratch space, and
  - a ***tape head*** that can read and write a single tape cell.
- At each step, the Turing machine
  - writes a symbol to the tape cell under the tape head,
  - changes state, and
  - moves the tape head to the left or to the right.

Input and Tape Alphabets

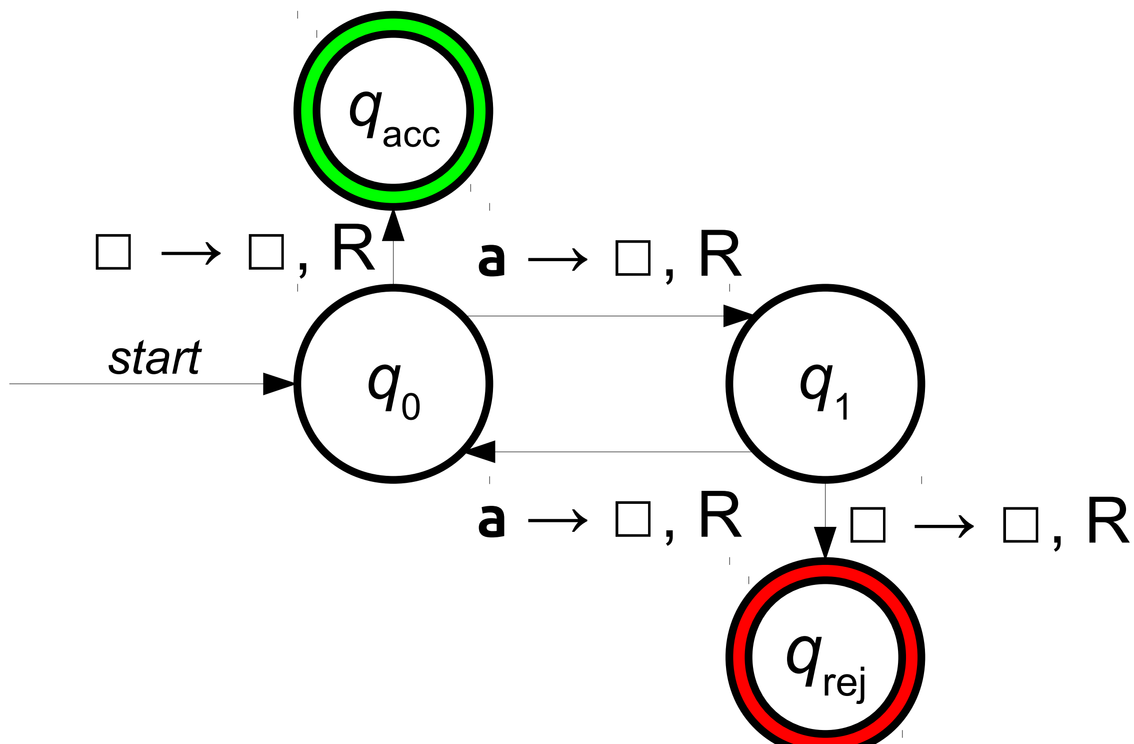
- A Turing machine has two alphabets:
  - An ***input alphabet*** Σ. All input strings are written in the input alphabet.
  - A ***tape alphabet*** Γ, where $\Sigma \subsetneq \Gamma$. The tape alphabet contains all symbols that can be written onto the tape.
- The tape alphabet Γ can contain any number of symbols, but always contains at least one ***blank symbol***, denoted $\square$. You are guaranteed $\square \notin \Sigma$.
- At startup, the Turing machine begins with an infinite tape of $\square$ symbols with the input written at some location. The tape head is positioned at the start of the input.

Accepting and Rejecting States

- Unlike DFAs, Turing machines do not stop processing the input when they finish reading it.
- Turing machines decide when (and if!) they will accept or reject their input.
- Turing machines can enter infinite loops and never accept or reject; more on that later…

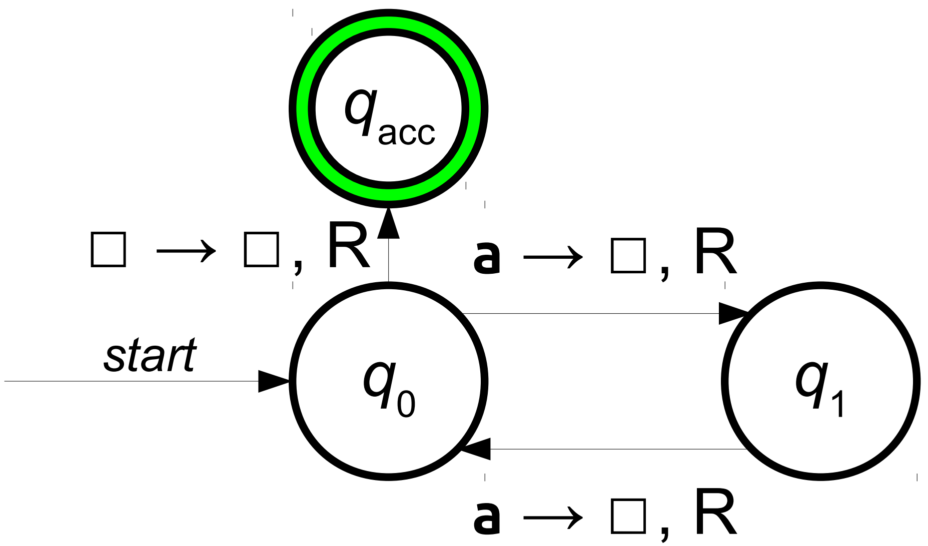
Determinism

- 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.

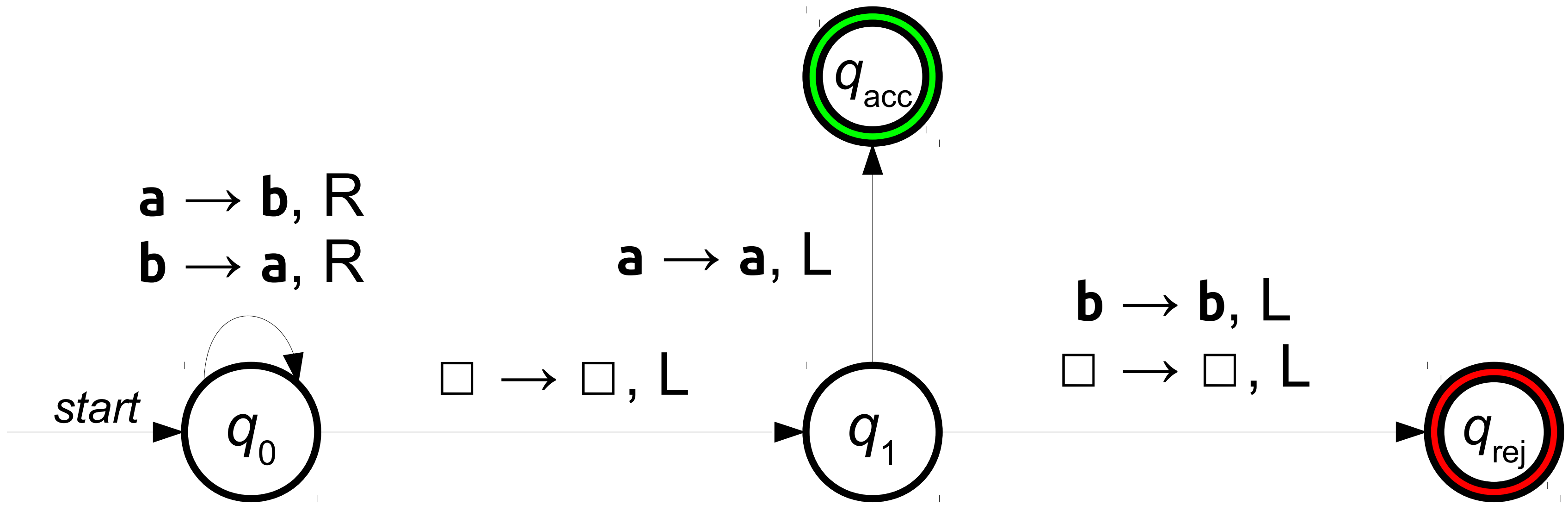


Determinism

- 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.

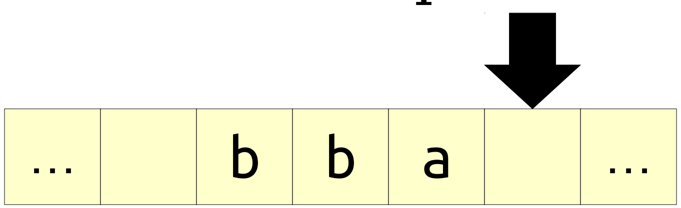


This machine is exactly the same as the previous one.

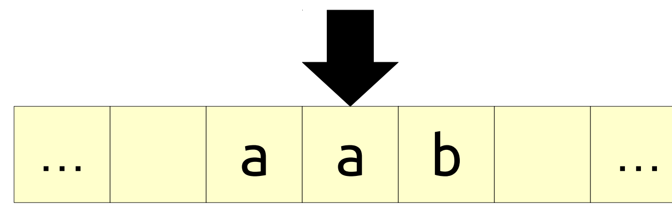


Run the TM shown above on the input string **bba**.
What will the tape look like when the TM finishes running?

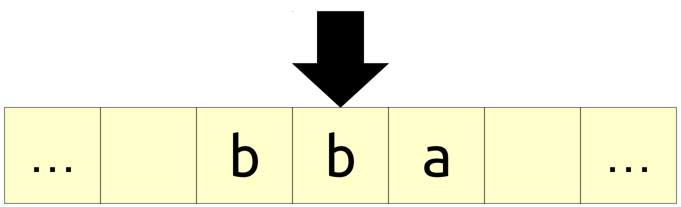
A.



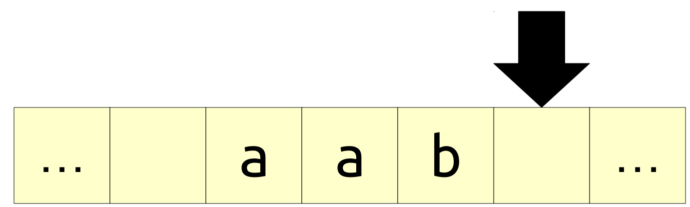
B.



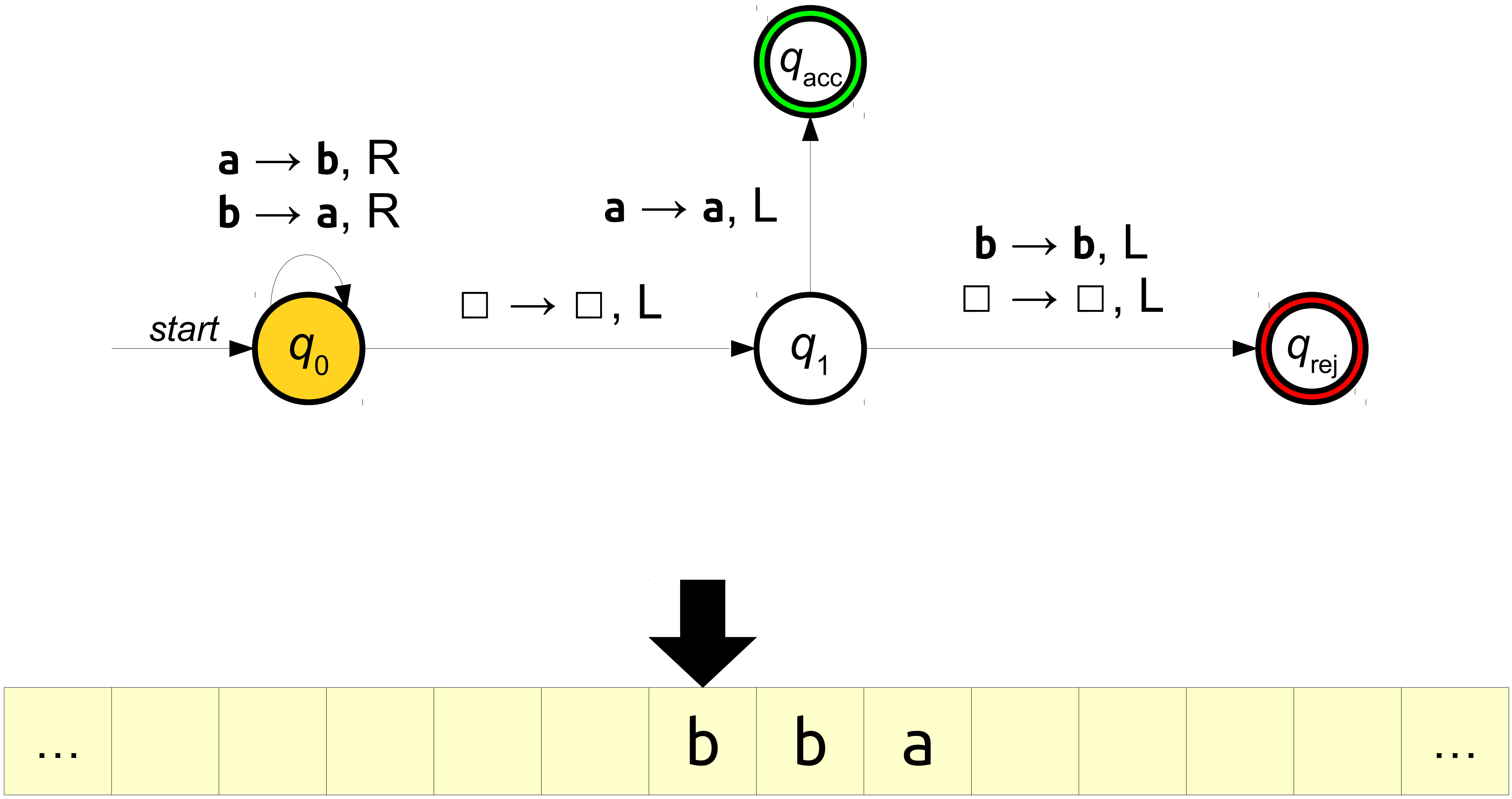
C.



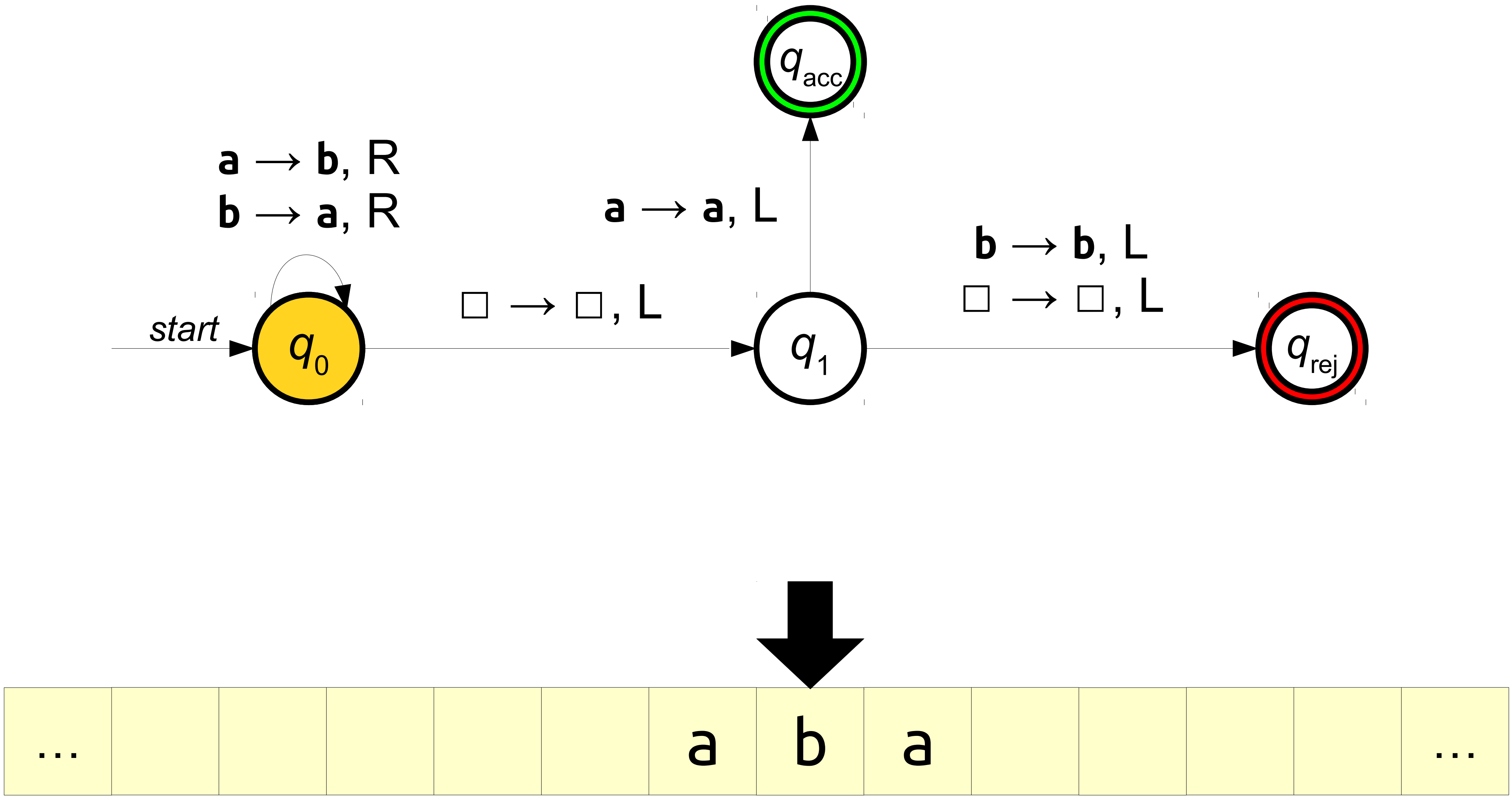
D.

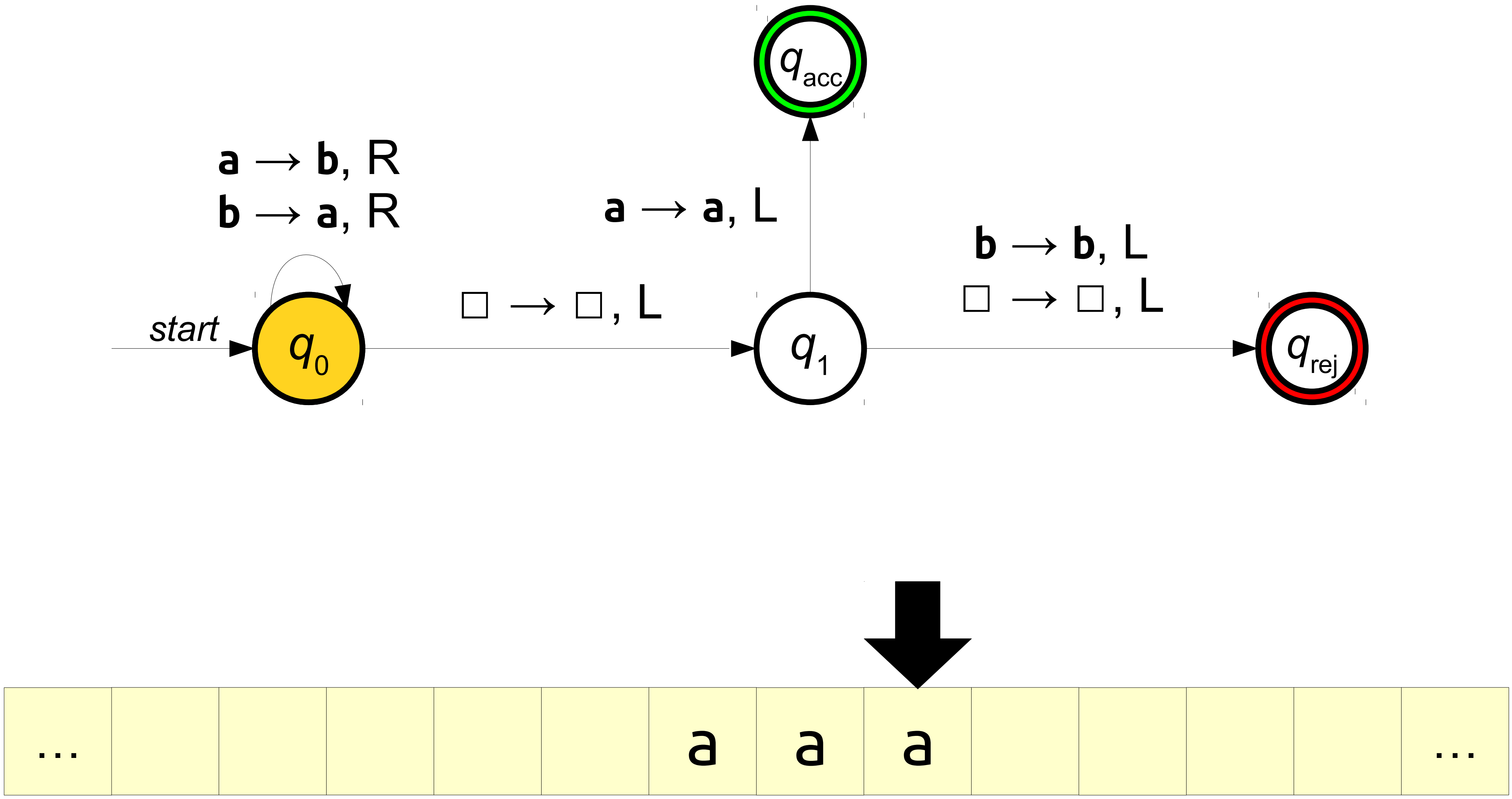


E. None of these, or two or more of these.

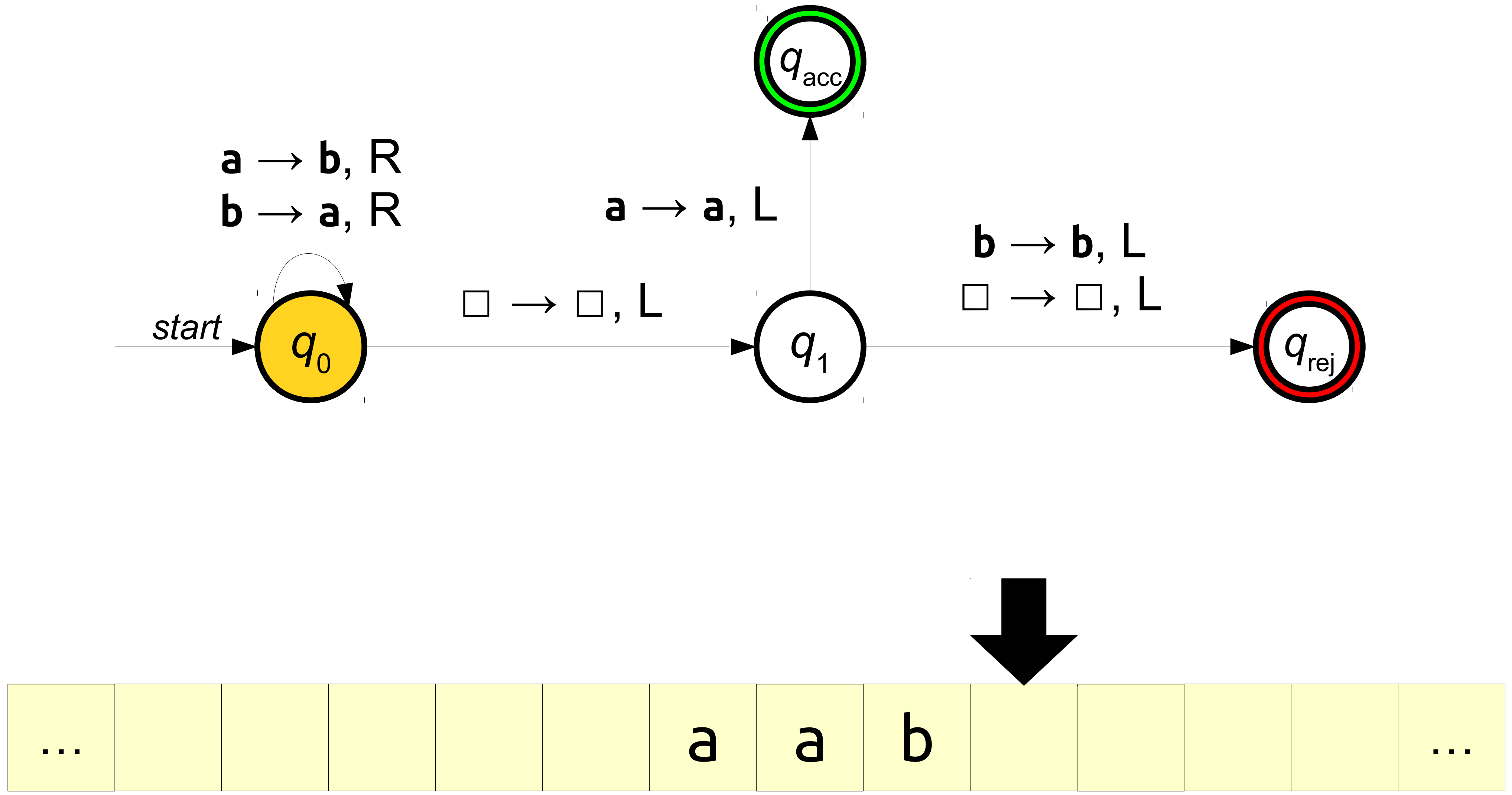


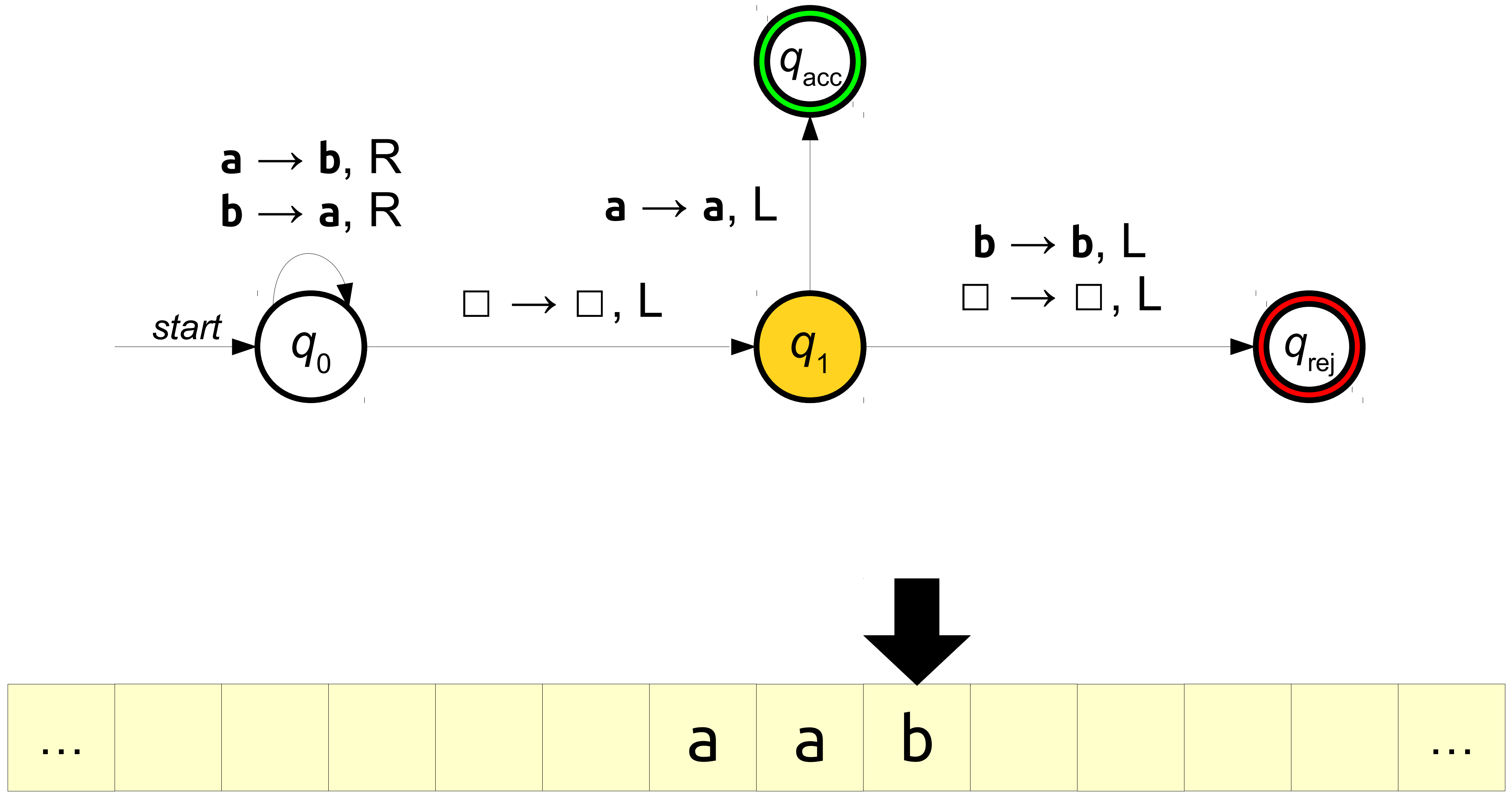
a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
b
b
a
…


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

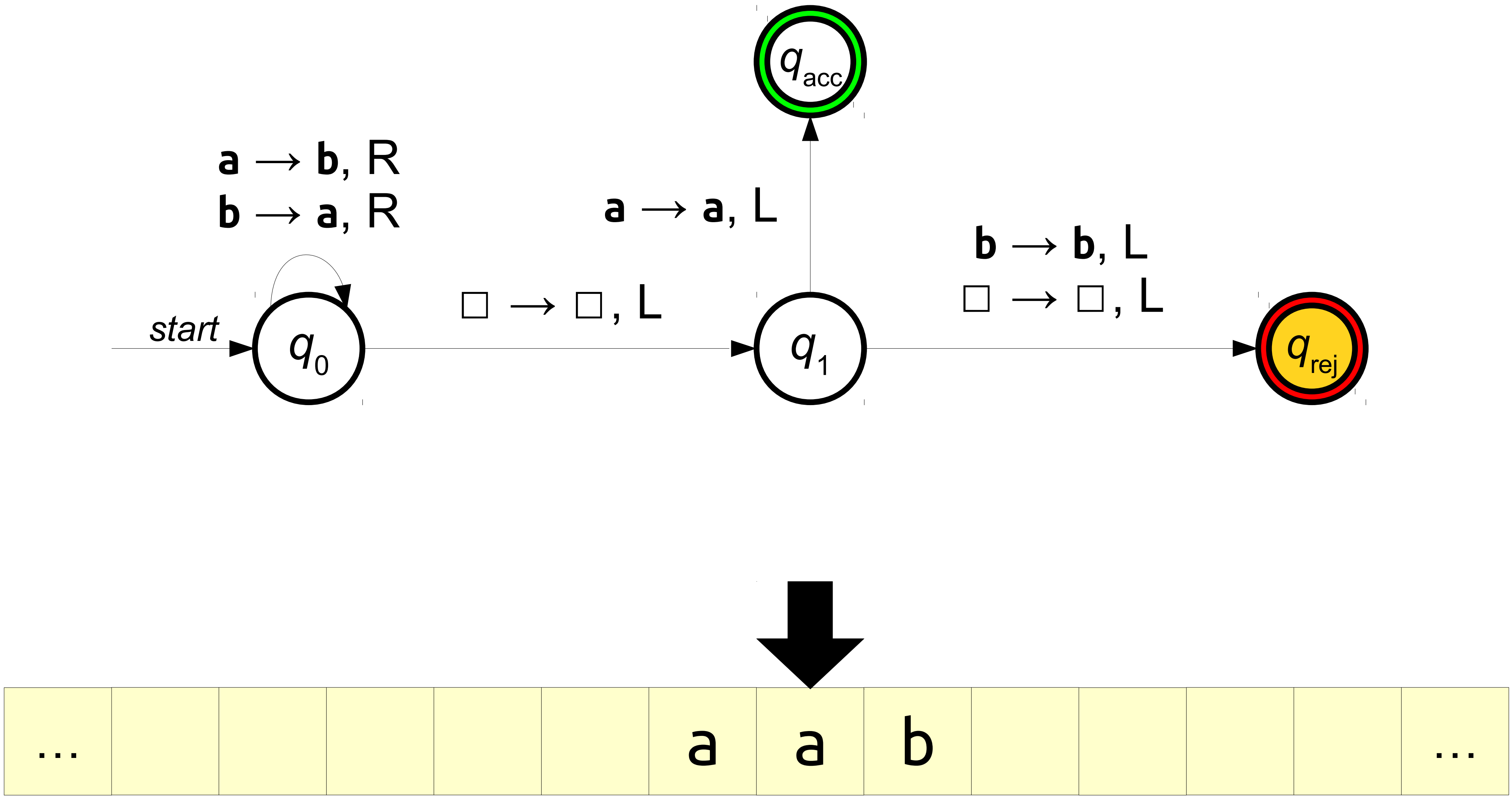


a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
a
a
a
…

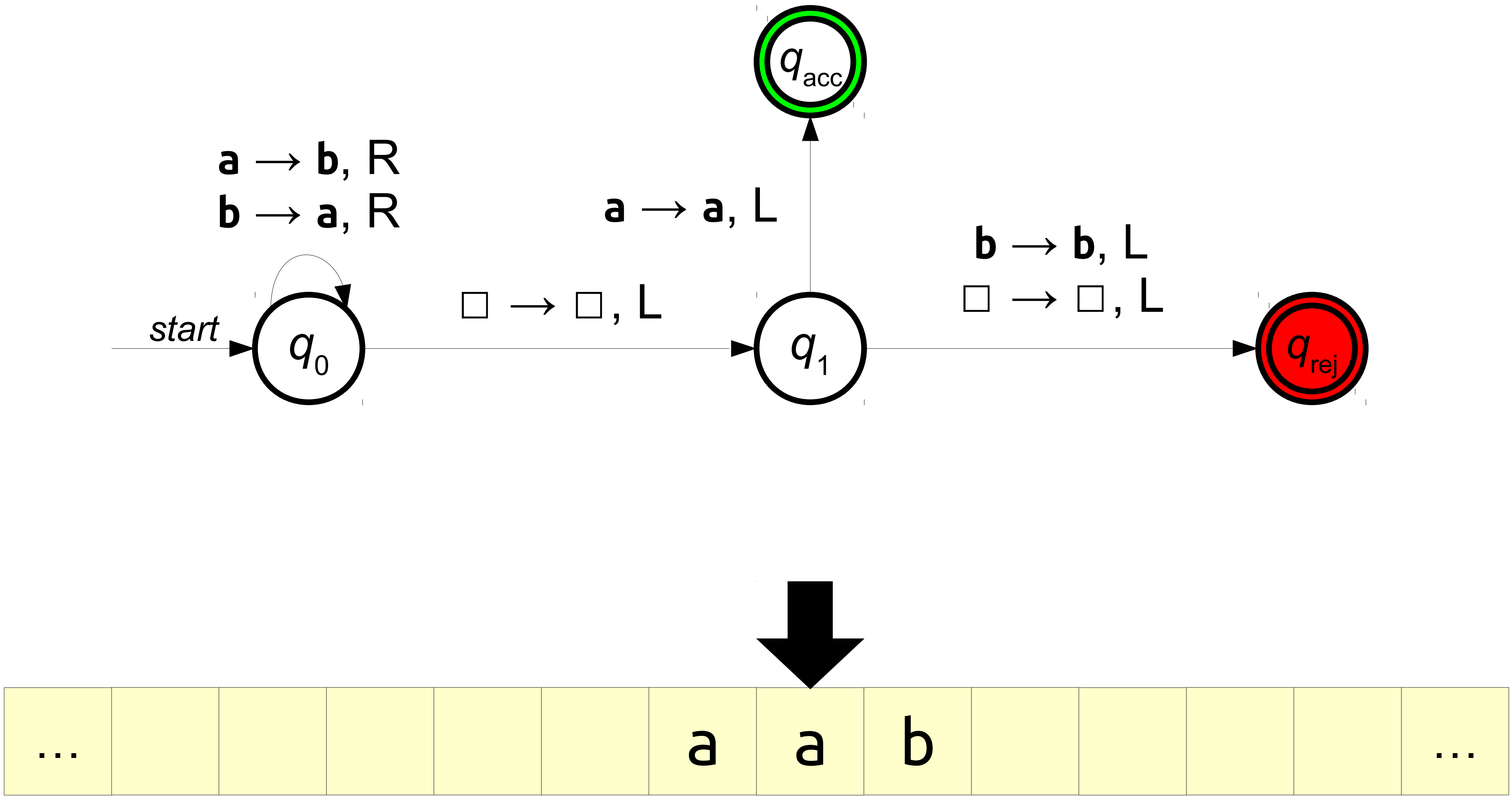

a → b, R
b → a, R
start
q0
□ → □, L
a → a, L
qacc
q1
b → b, L
□ → □, L
qrej
…
a
a
b
…

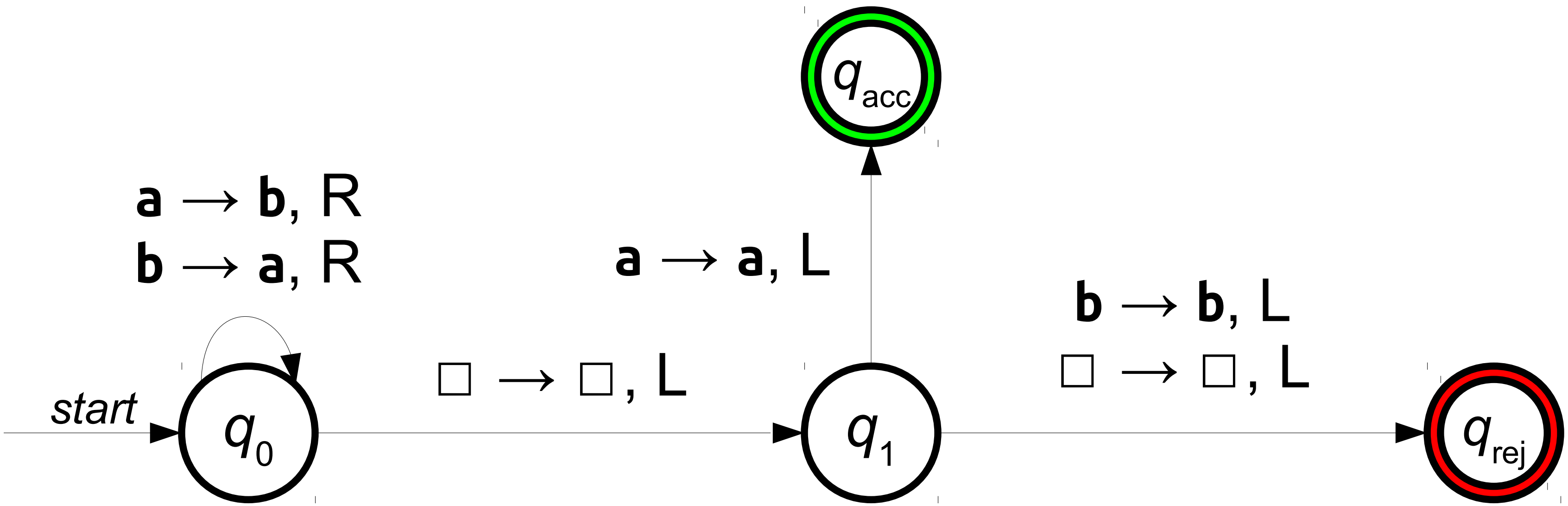


a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
a
a
b
…



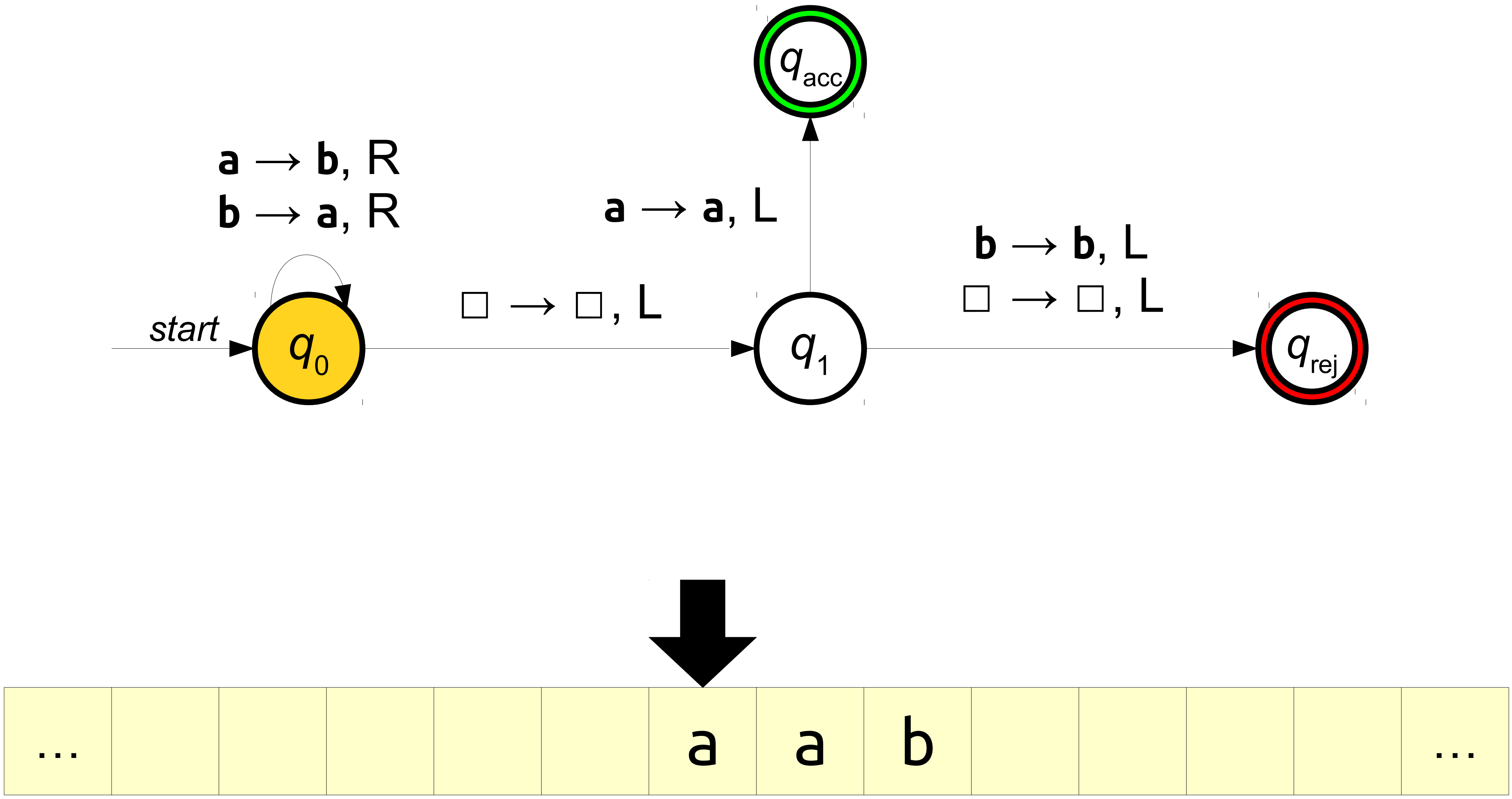
q_{acc}
a → b, R
b → a, R
a → a, L
b → b, L
□ → □, L
□ → □, L
start
q_0
q_1
q_{rej}
…
a
a
b
…

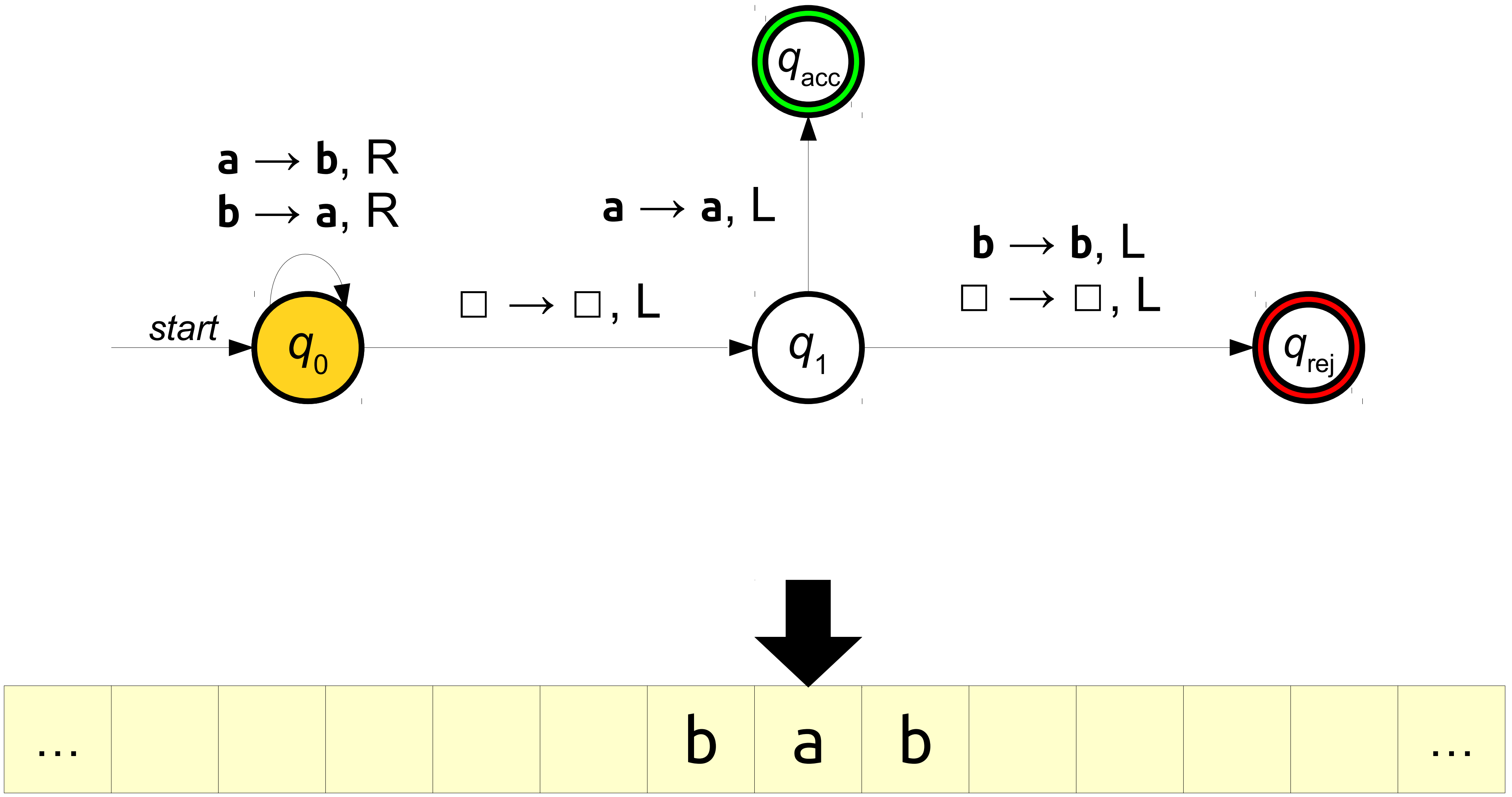

a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej
…
a
a
b
…



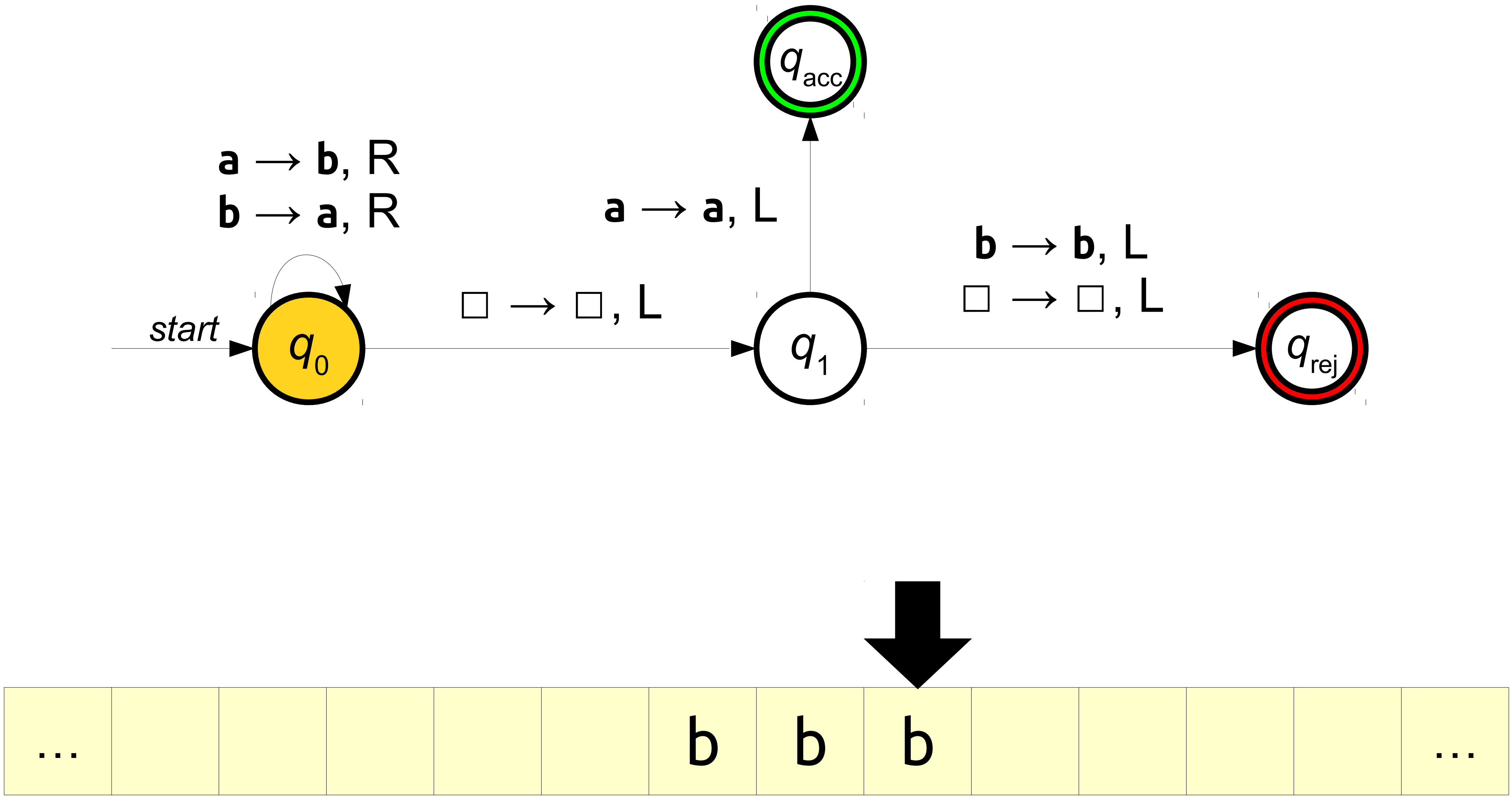
If M is a Turing machine with input alphabet Σ, then the ***language of M***, denoted $\mathscr{L}(M)$, is the set

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

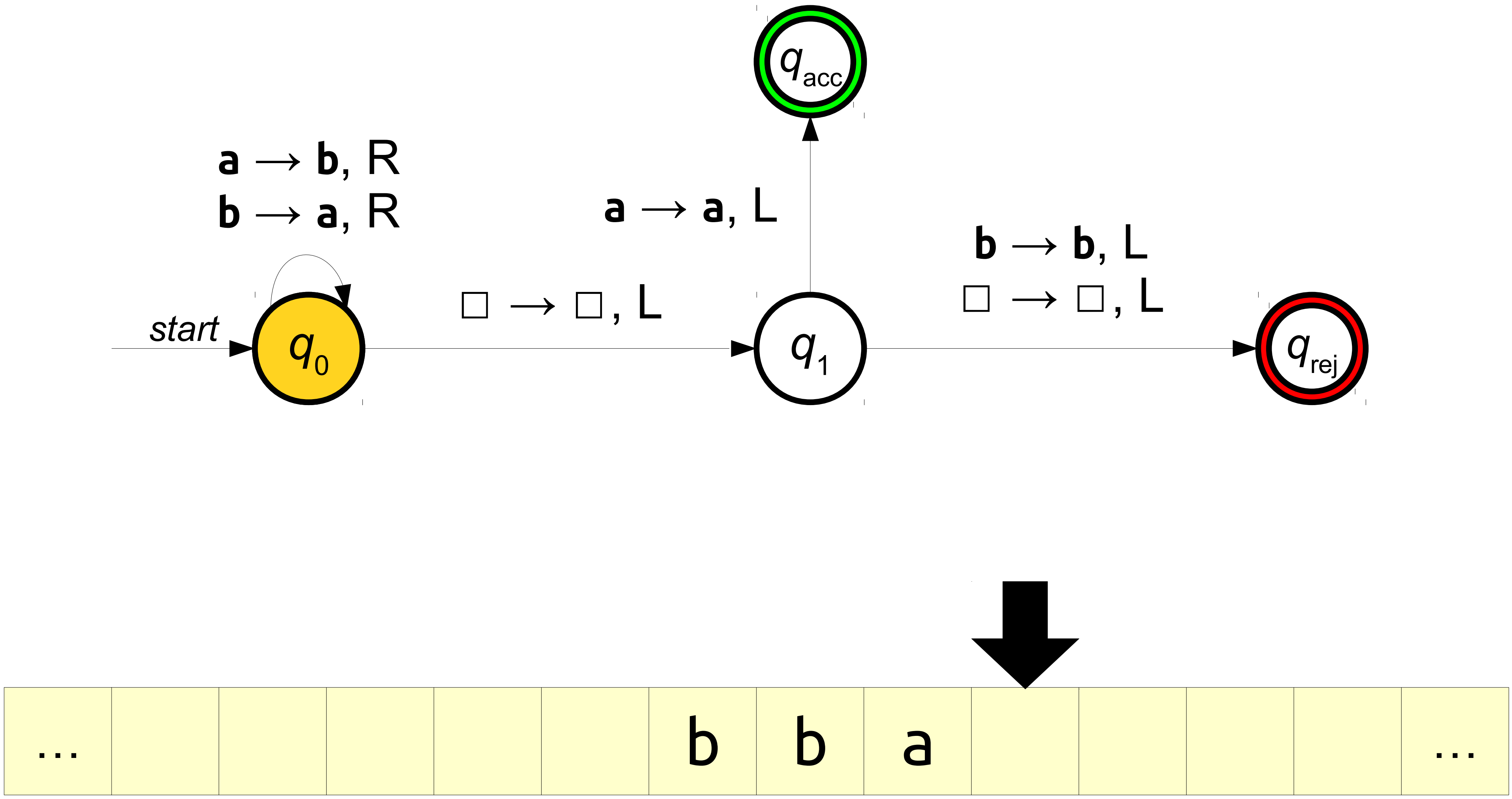

a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
a
a
b
…



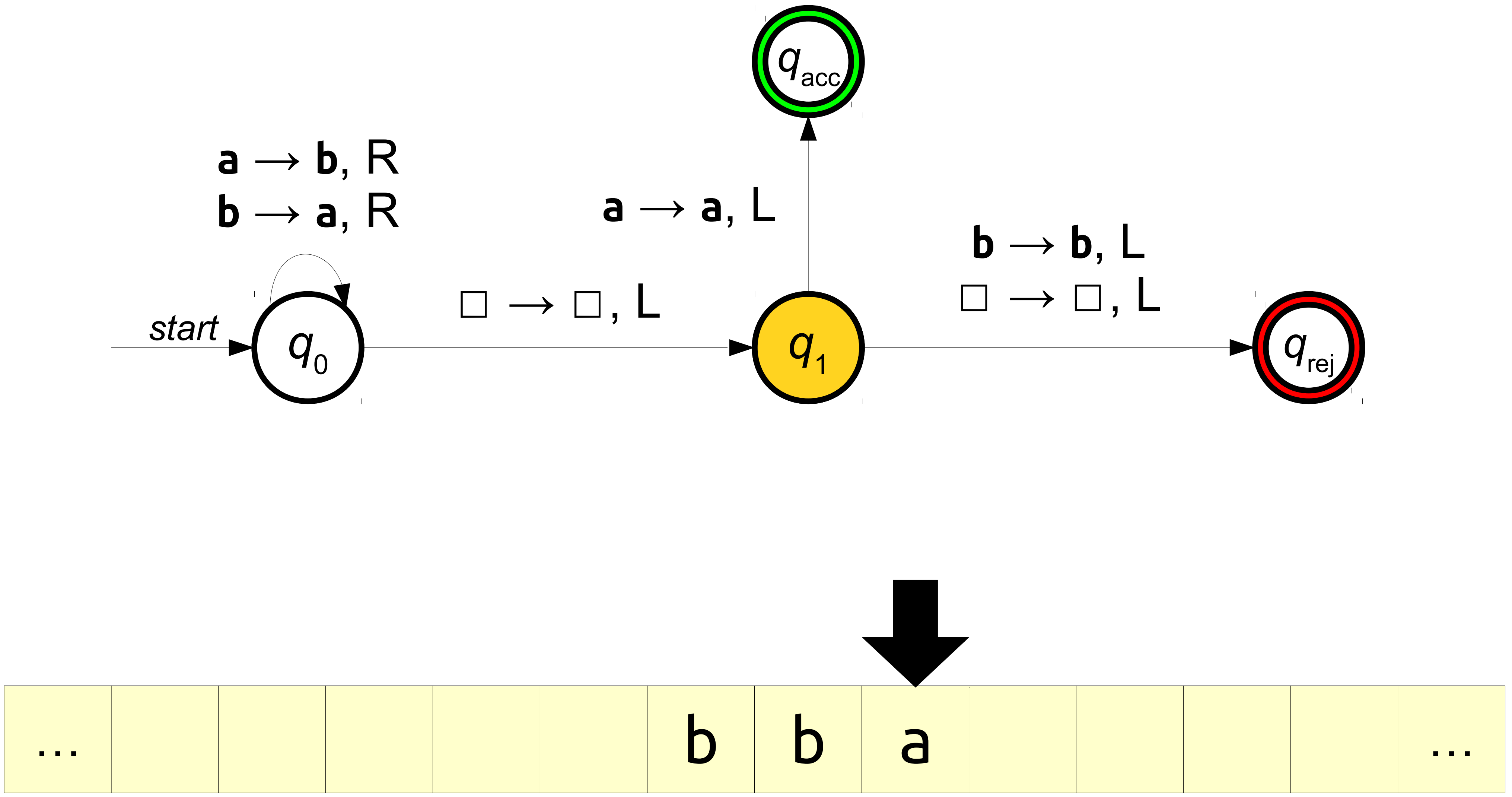
a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
b
a
b
…



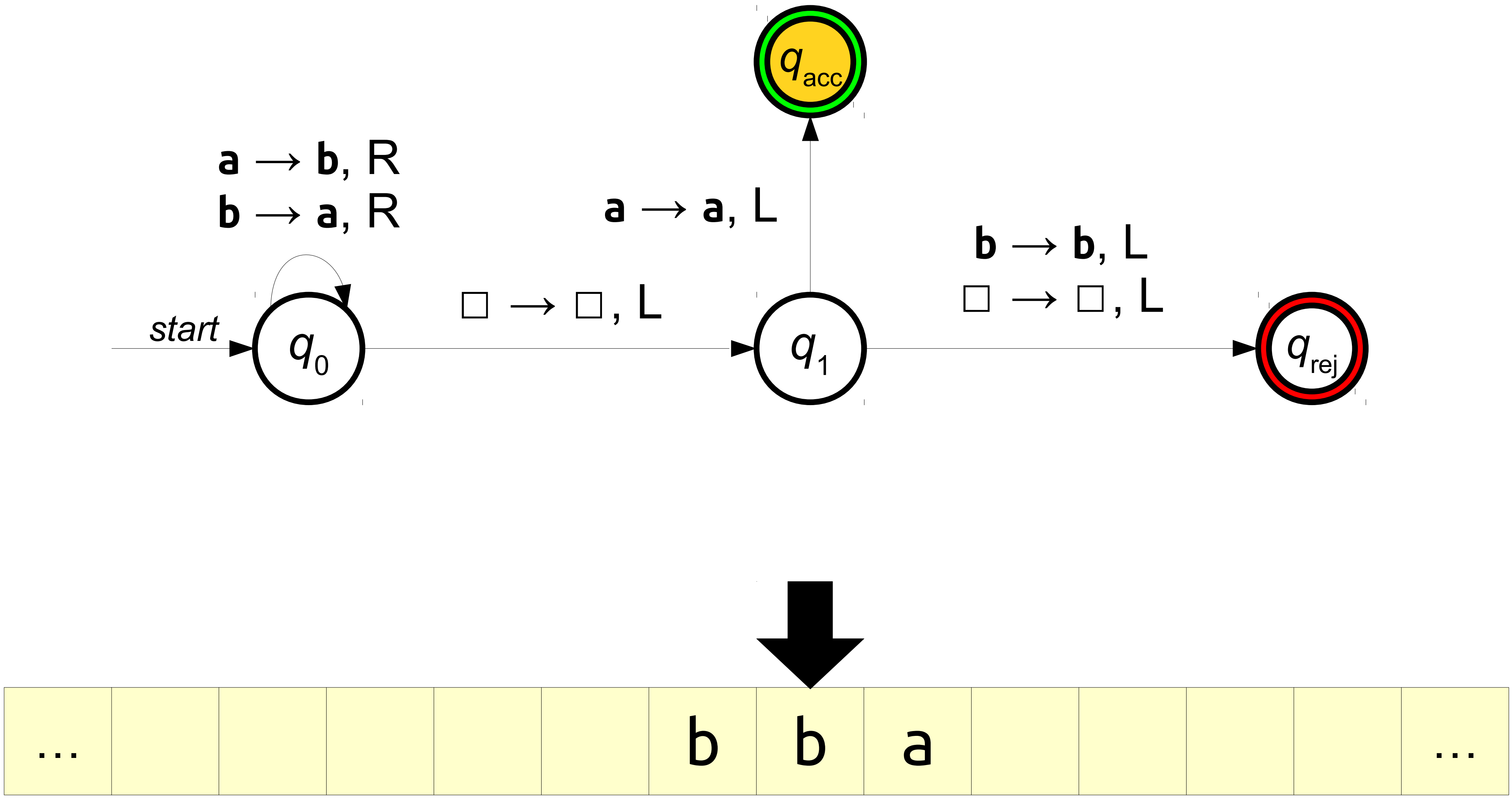
a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
b
b
b
…

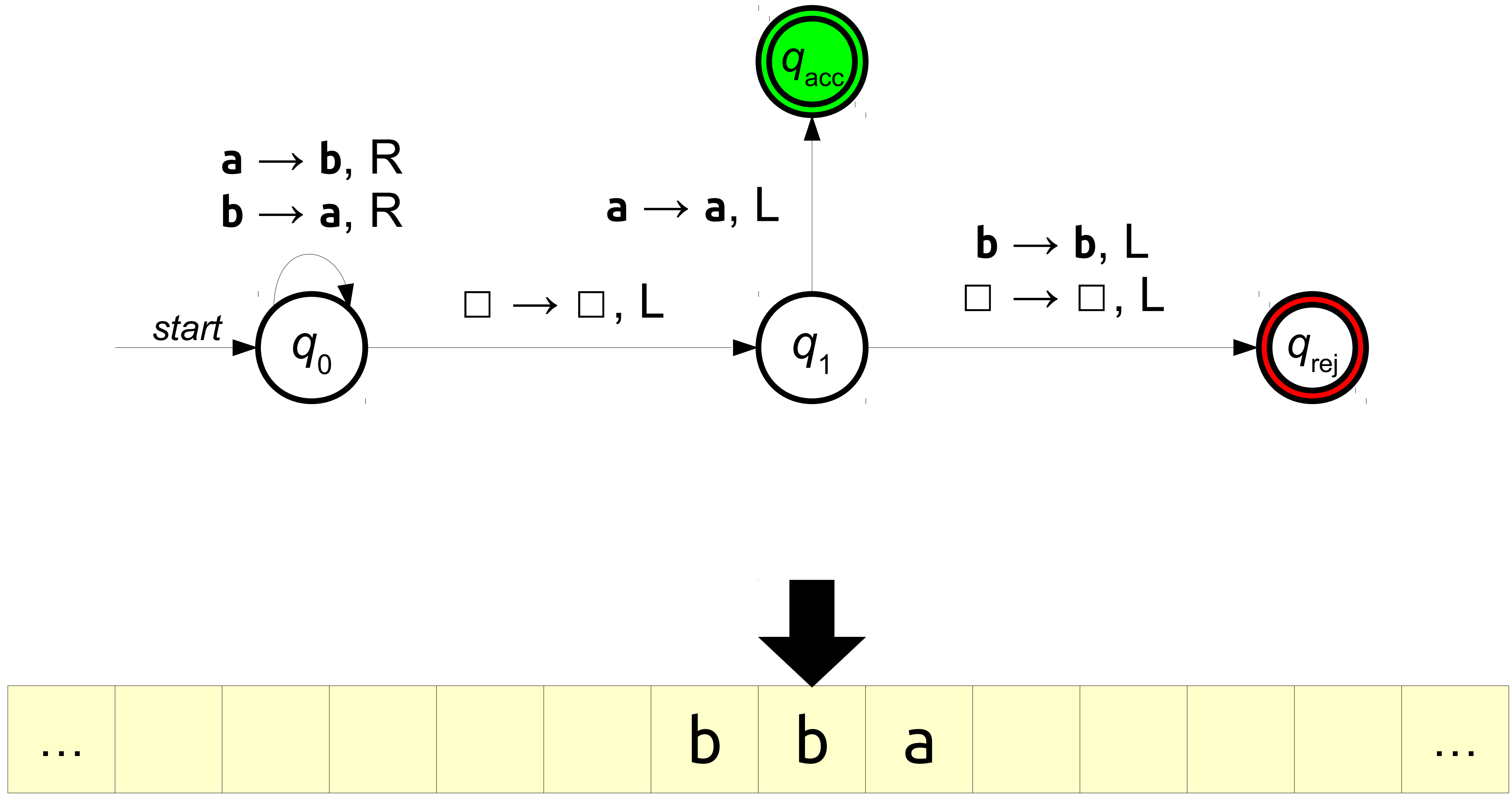


a → b, R
b → a, R
start
q_0
□ → □, L
q_1
a → a, L
q_{acc}
b → b, L
□ → □, L
q_{rej}
…
b
b
a
…

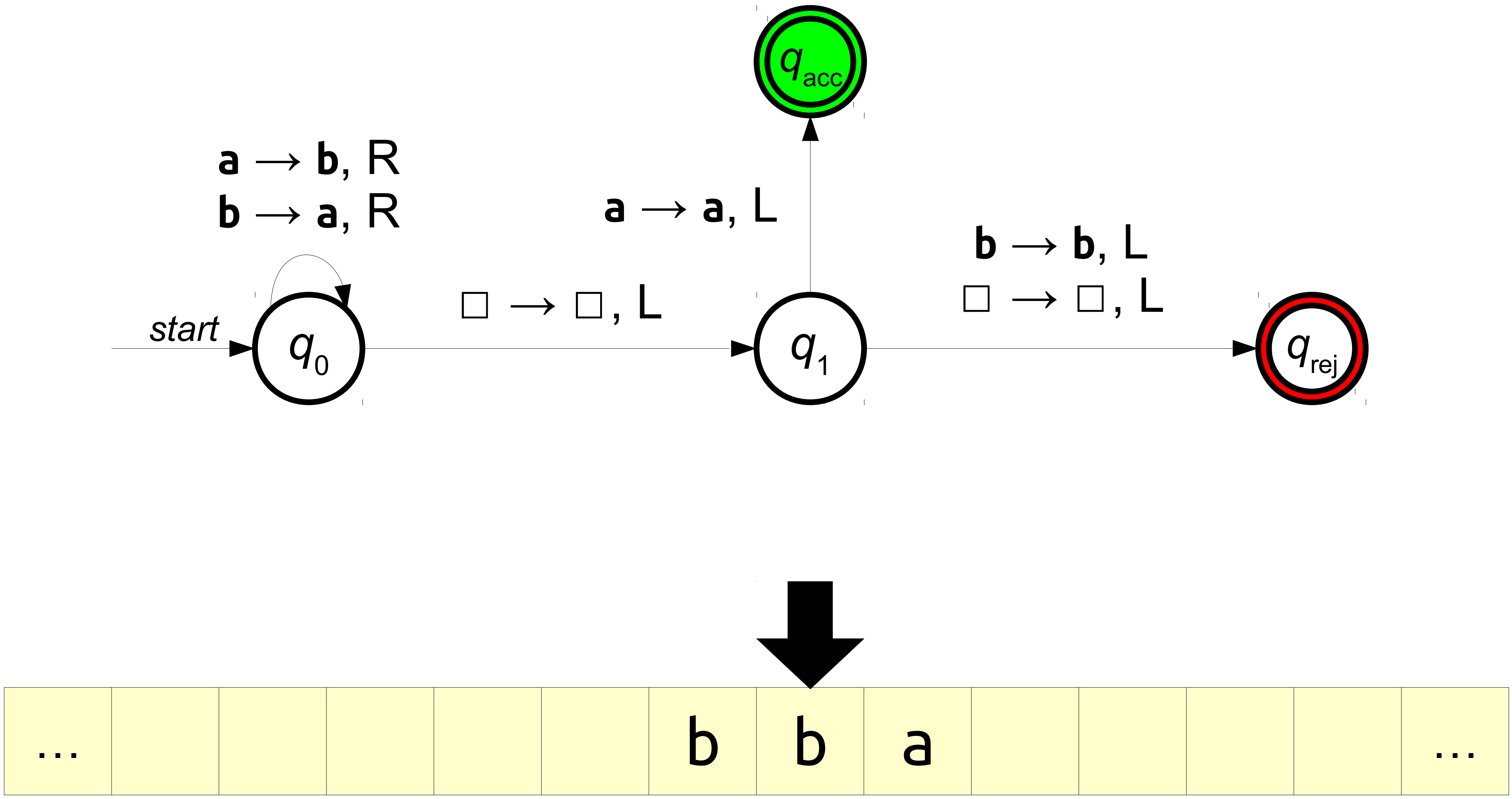


a → b, R
b → a, R
start
q_0
□ → □, L
a → a, L
q_{acc}
q_1
b → b, L
□ → □, L
q_{rej}
…
b
b
a
…


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



a → b, R
b → a, R
start
q0
□ → □, L
q1
a → a, L
qacc
b → b, L
□ → □, L
qrej
…
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 $\mathcal{L}(M) = \{\ w \in \{$**a**, **b**$\}^* \mid w$ ends in **b** $\}$

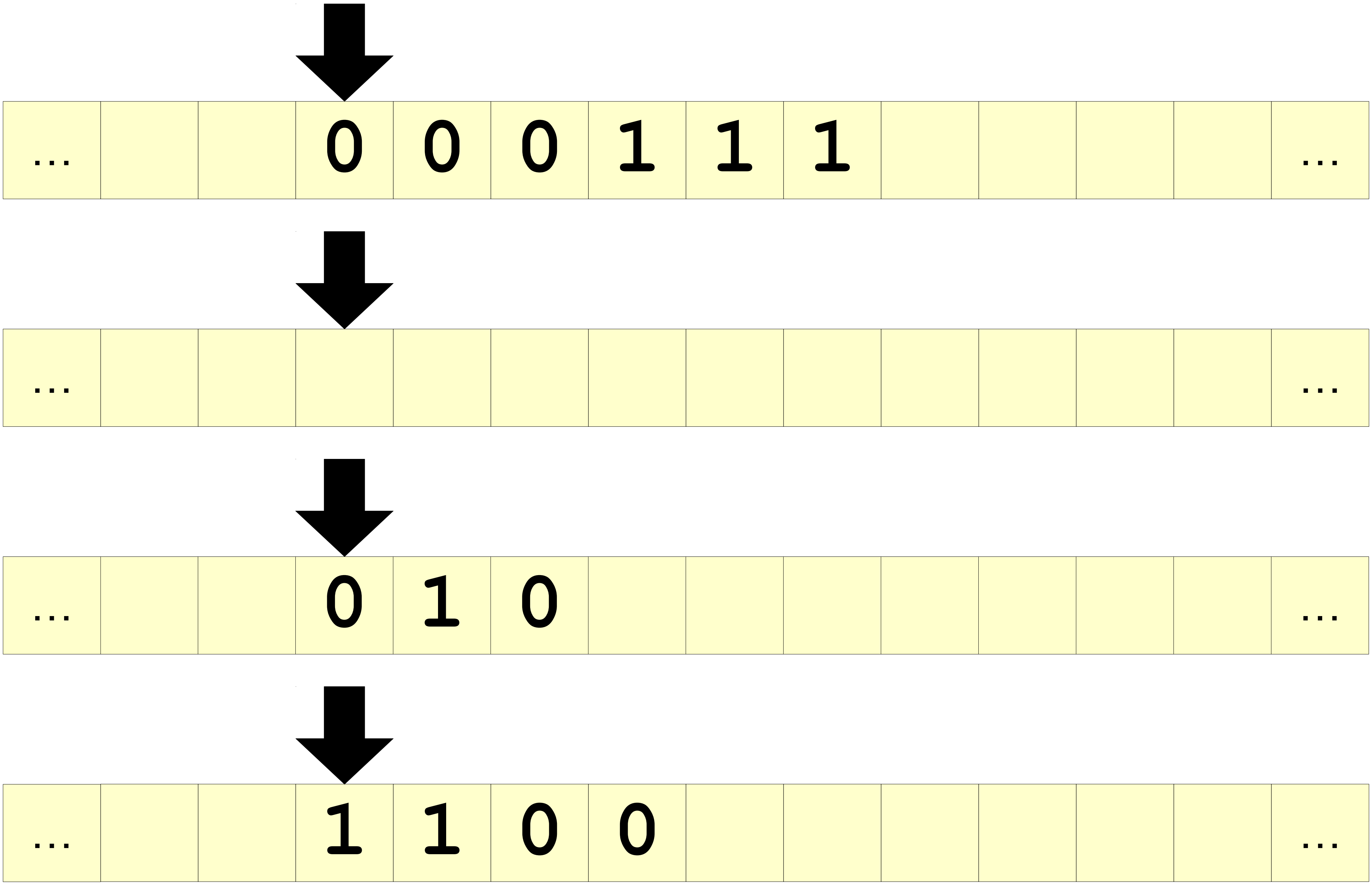
Designing Turing Machines

- Despite their simplicity, Turing machines are very powerful computing devices.
- Today's lecture explores how to design Turing machines for various languages.

Designing Turing Machines

- Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and consider the language $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$.
- We know that L is context-free.
- How might we build a Turing machine for it?

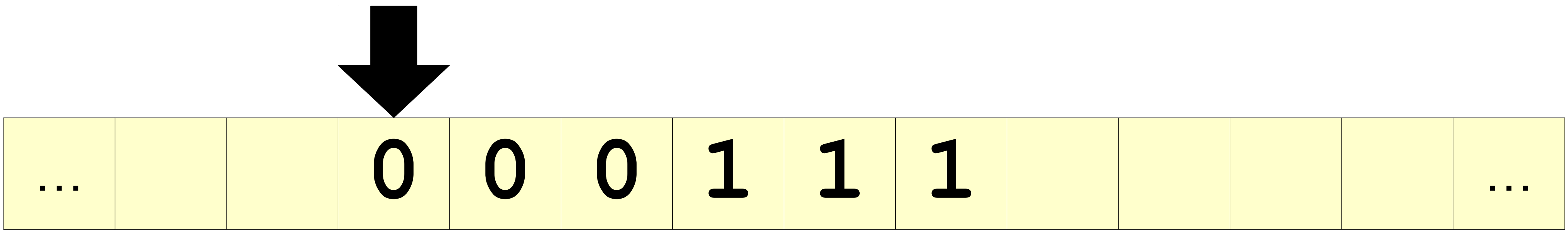
$$L = \{ \mathbf{0}^n \mathbf{1}^n \mid n \in \mathbb{N} \}$$



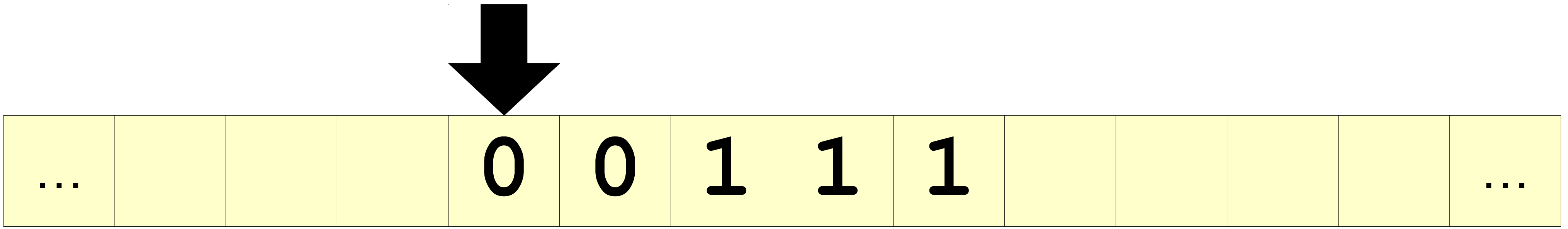
A Recursive Approach

- 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.

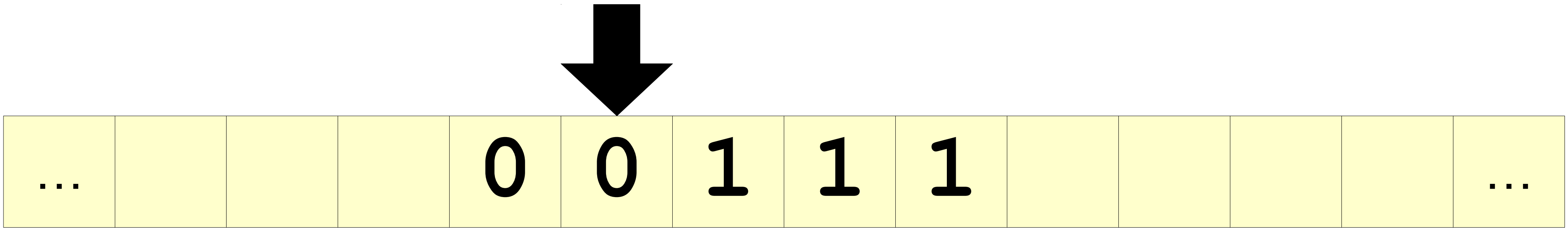
A Sketch of the TM



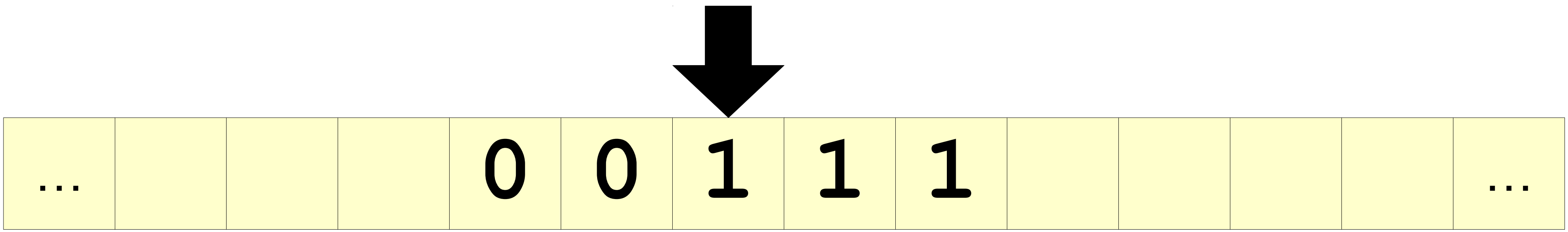
A Sketch of the TM



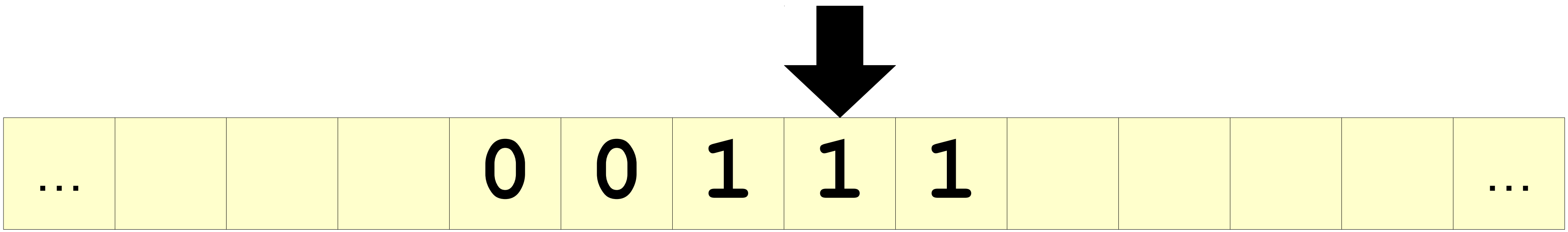
A Sketch of the TM



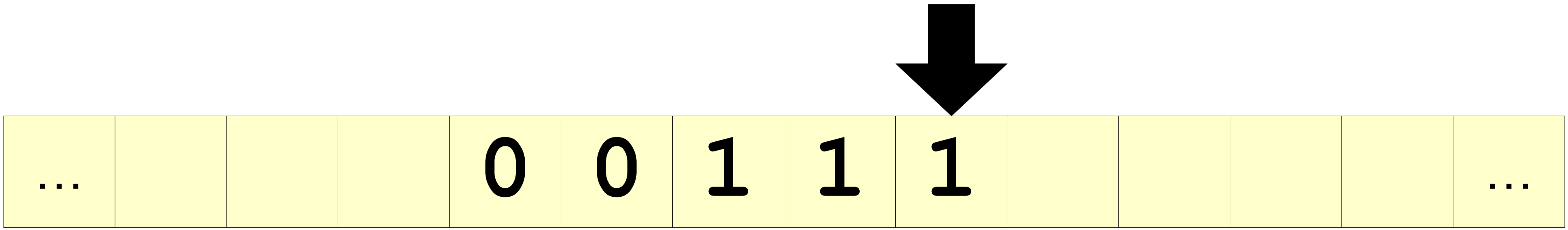
A Sketch of the TM



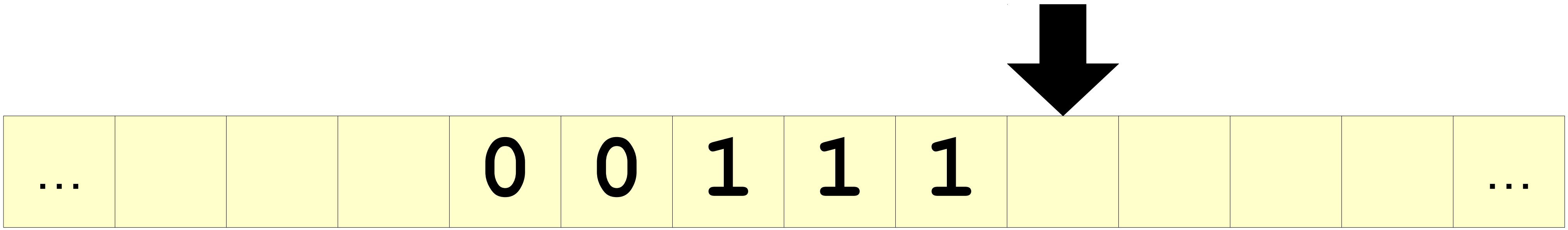
A Sketch of the TM



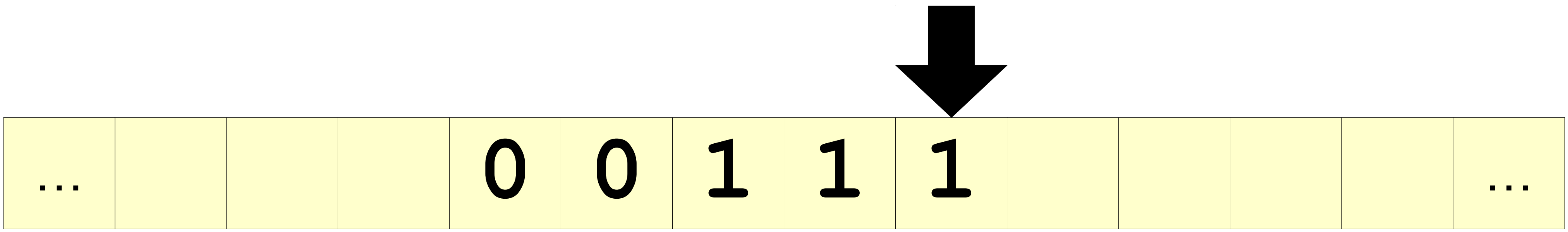
A Sketch of the TM



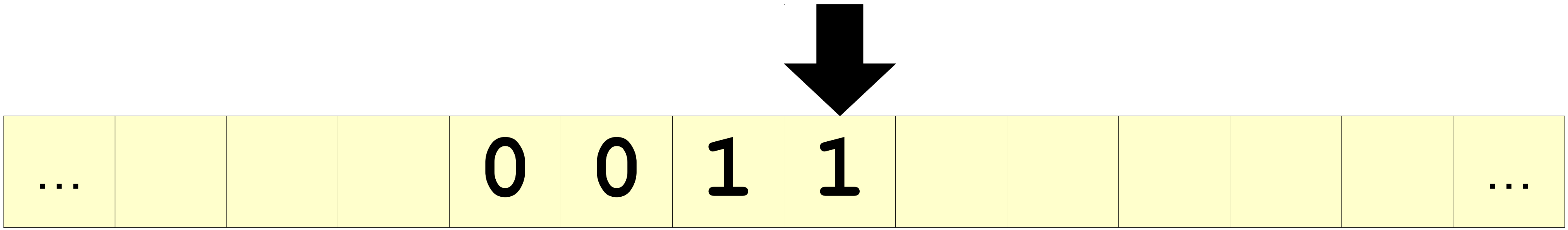
A Sketch of the TM



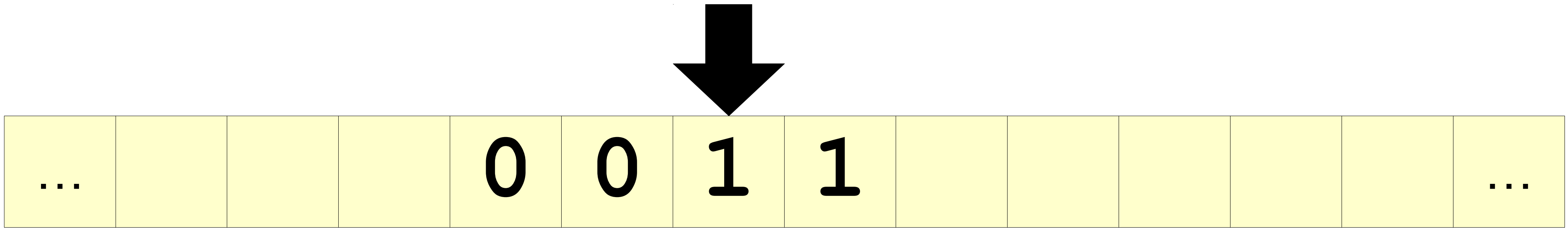
A Sketch of the TM



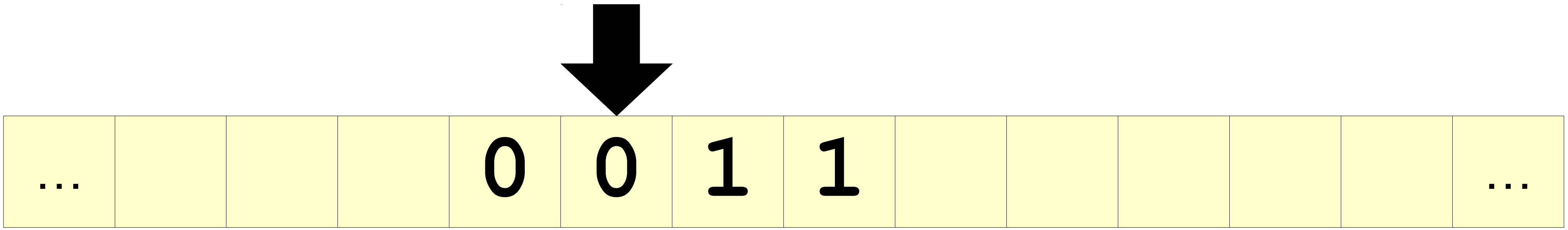
A Sketch of the TM



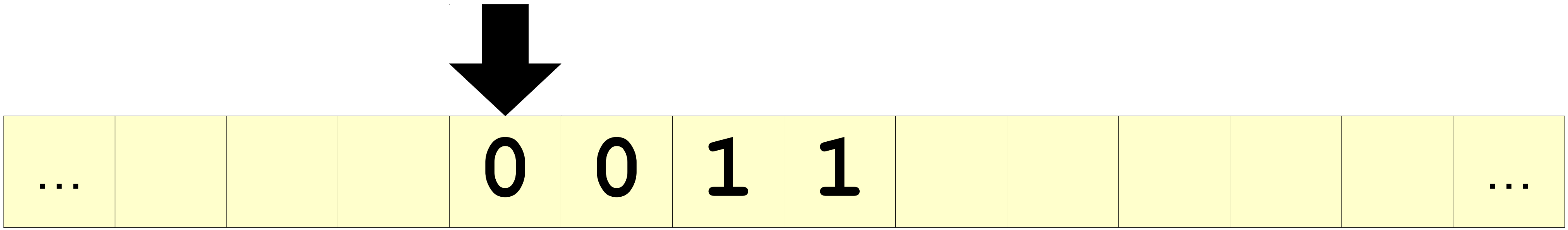
A Sketch of the TM



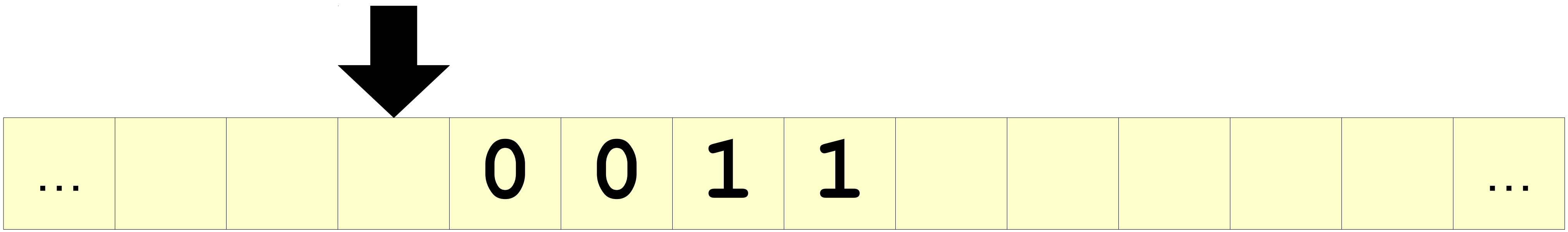
A Sketch of the TM



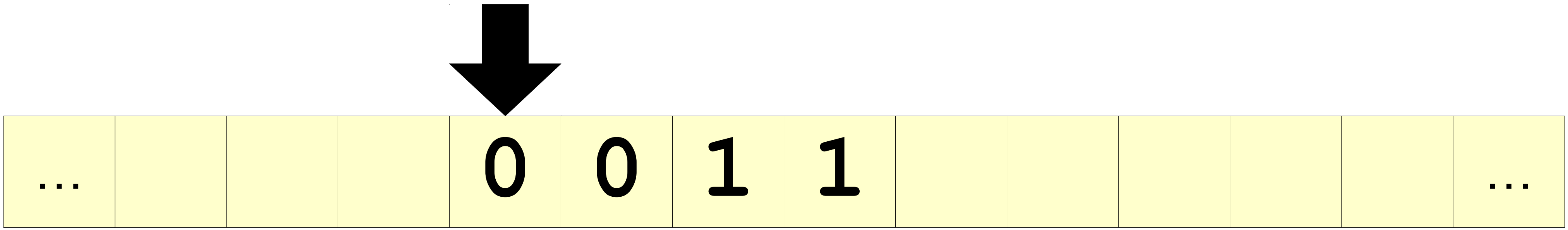
A Sketch of the TM



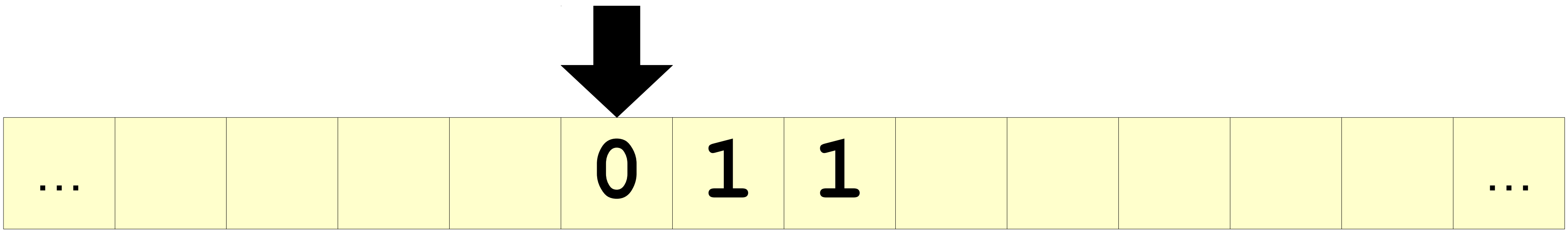
A Sketch of the TM



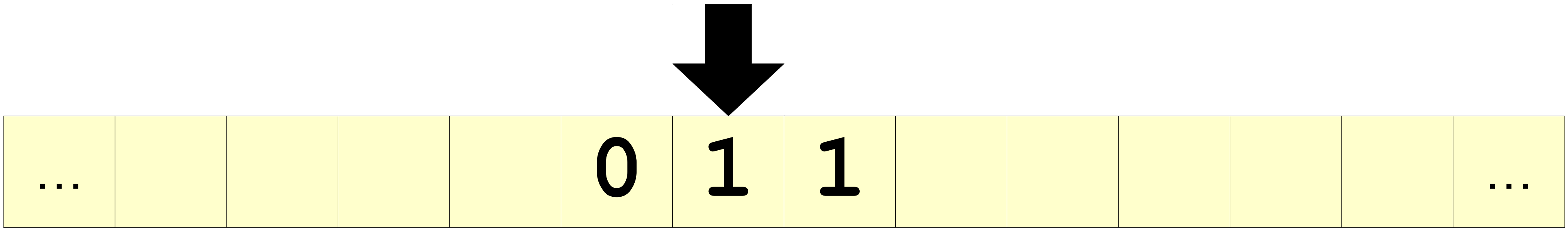
A Sketch of the TM



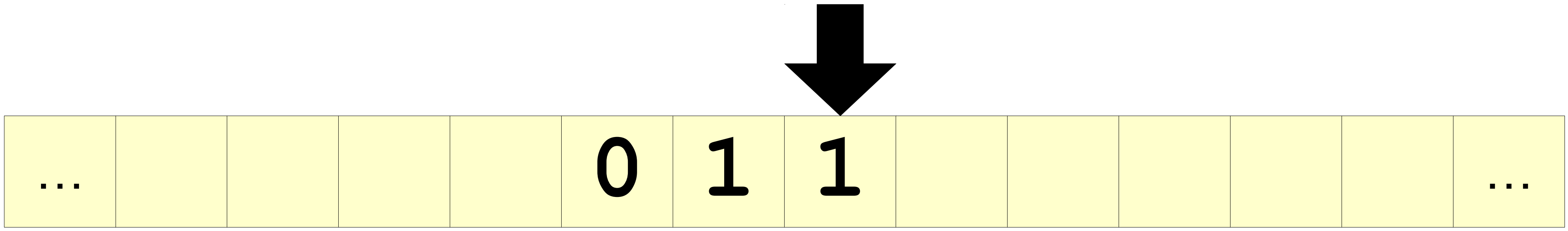
A Sketch of the TM



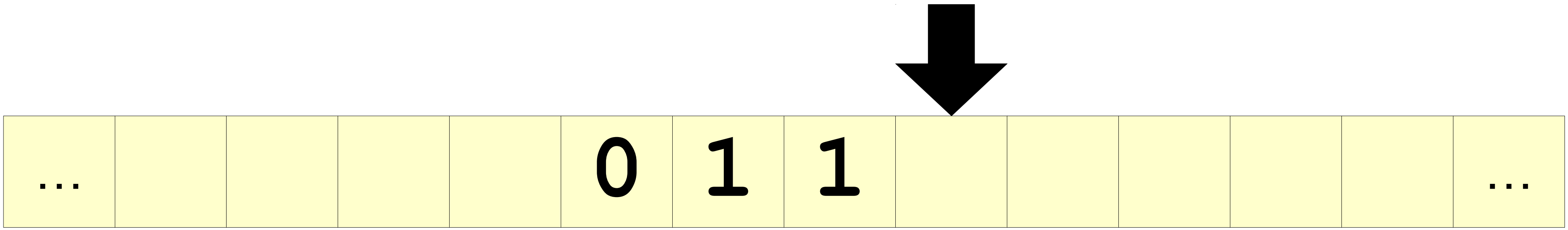
A Sketch of the TM



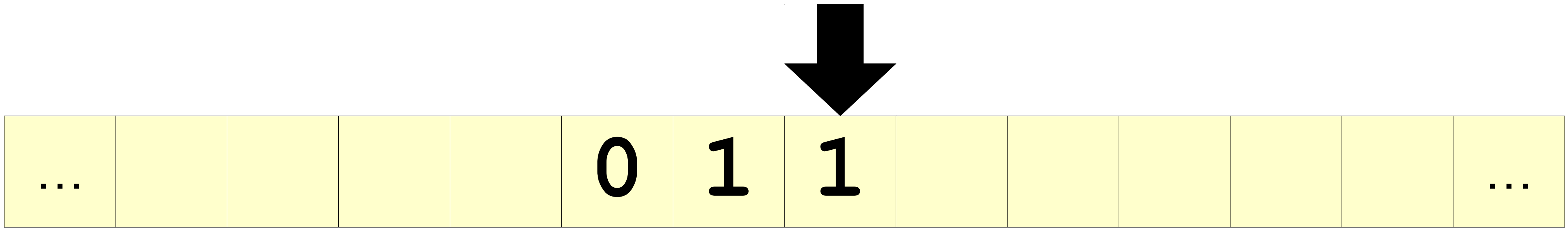
A Sketch of the TM



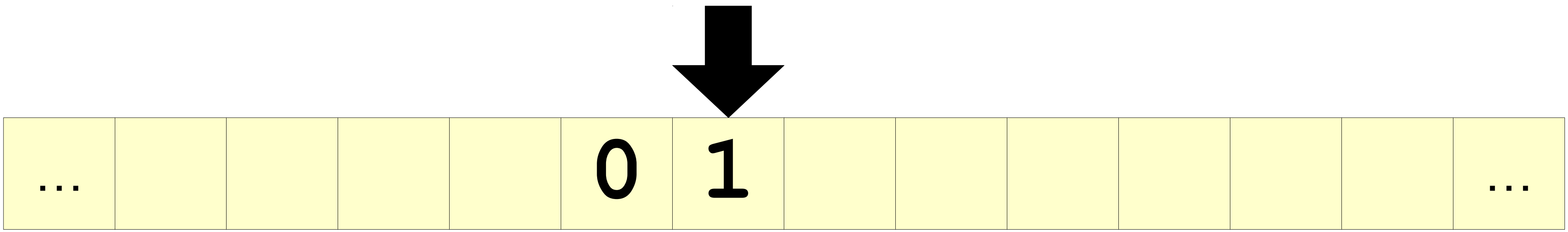
A Sketch of the TM



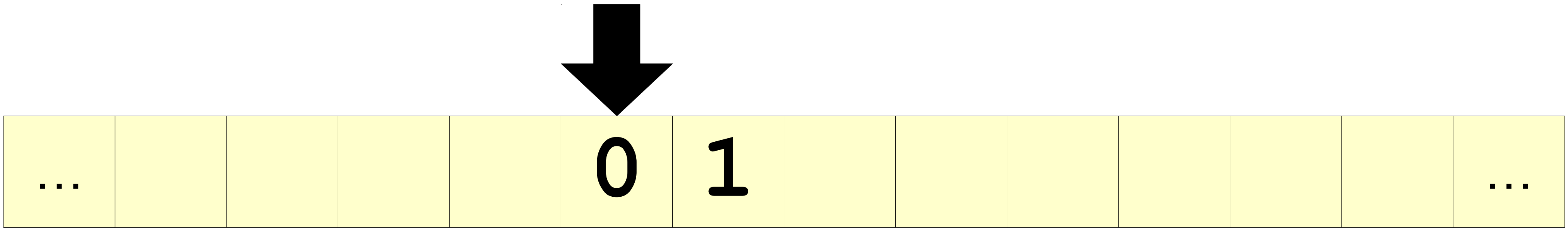
A Sketch of the TM



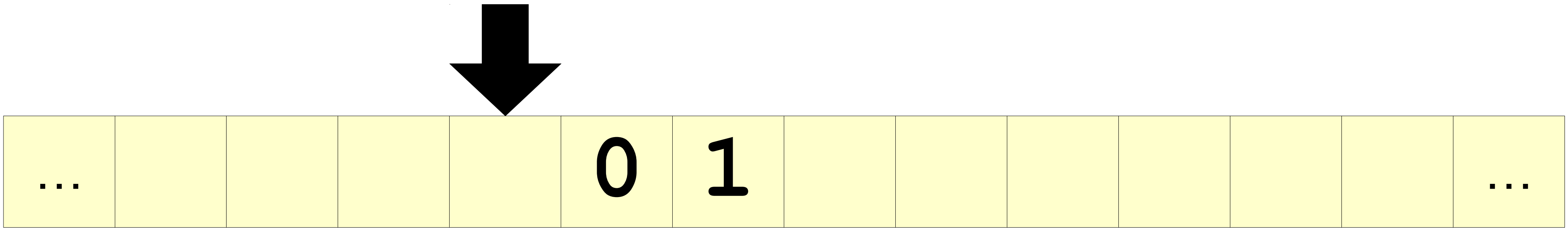
A Sketch of the TM



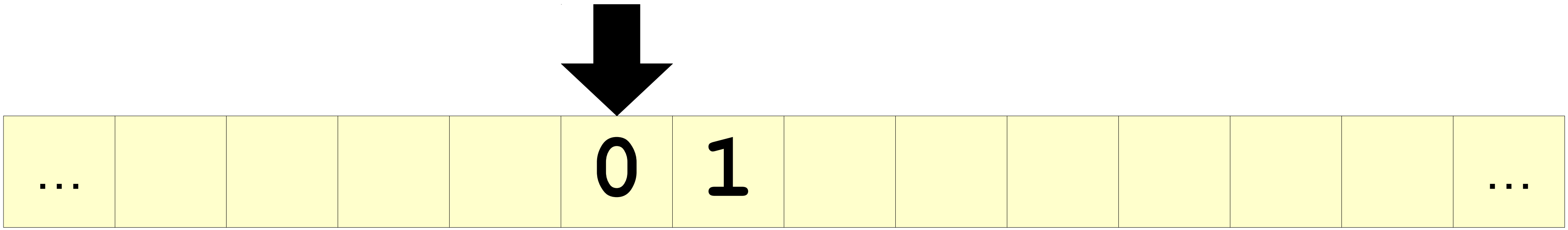
A Sketch of the TM



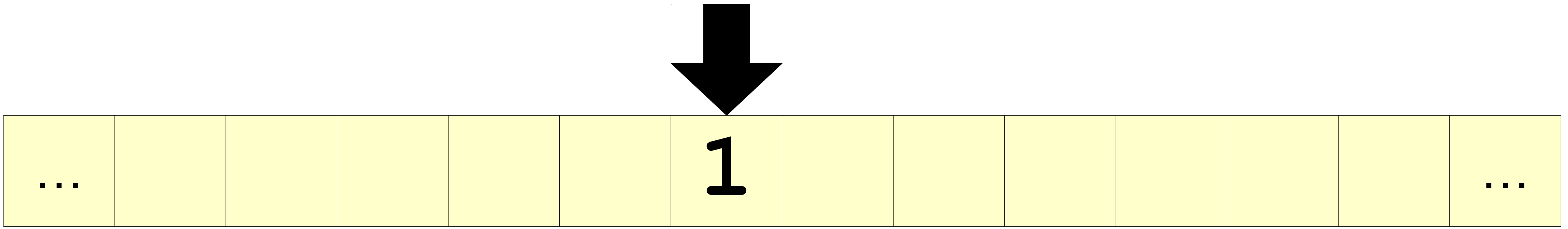
A Sketch of the TM



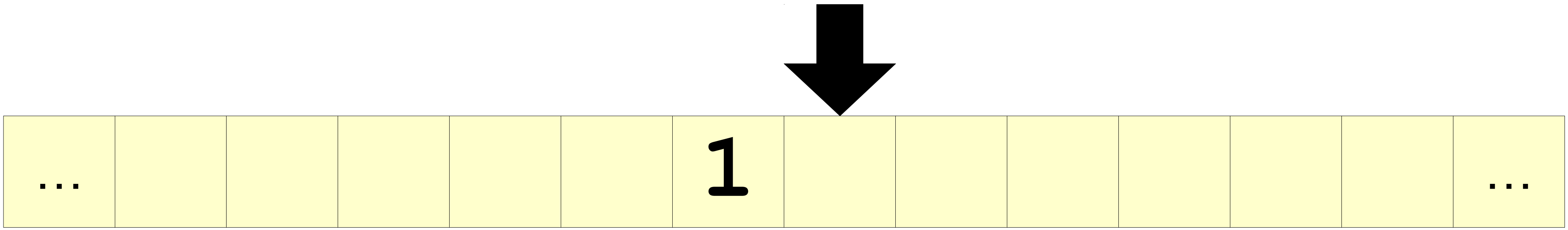
A Sketch of the TM



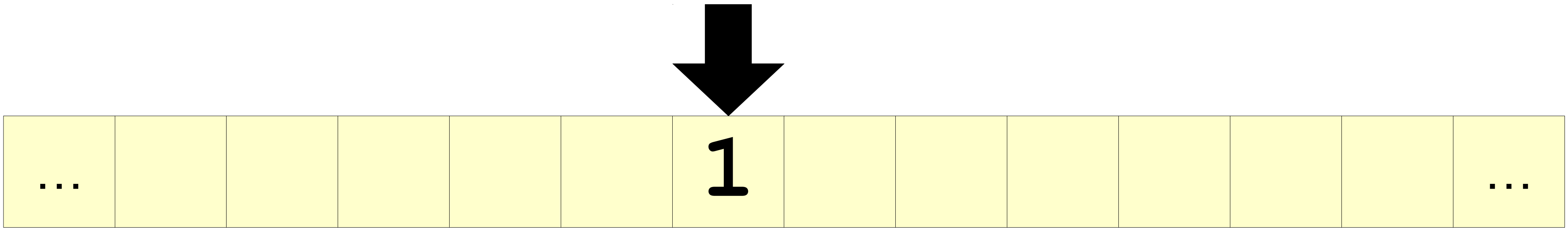
A Sketch of the TM



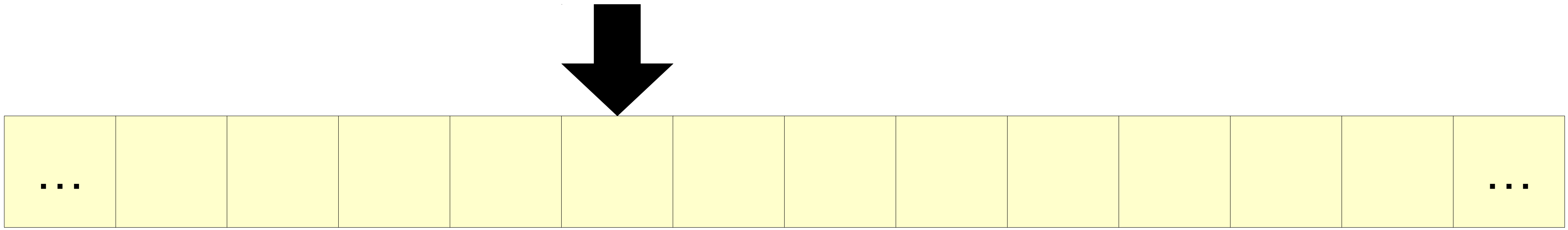
A Sketch of the TM



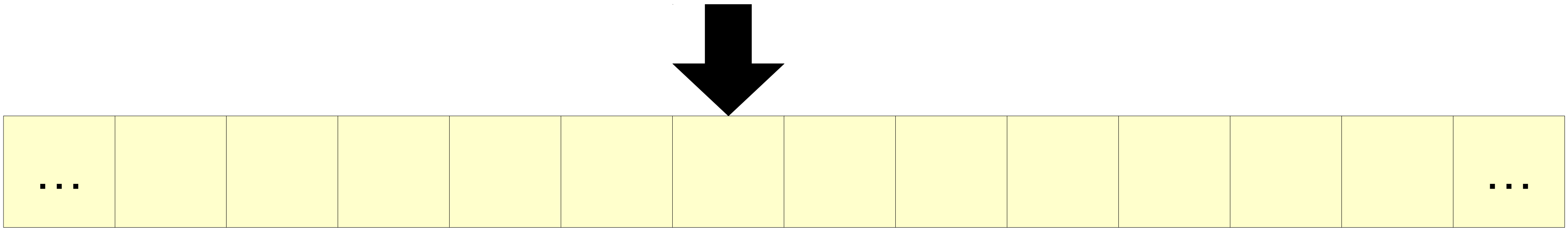
A Sketch of the TM

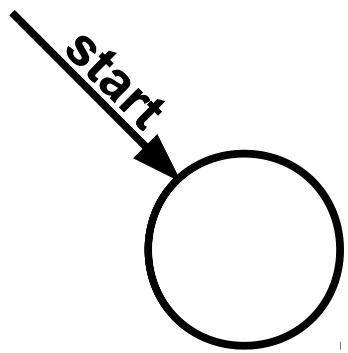


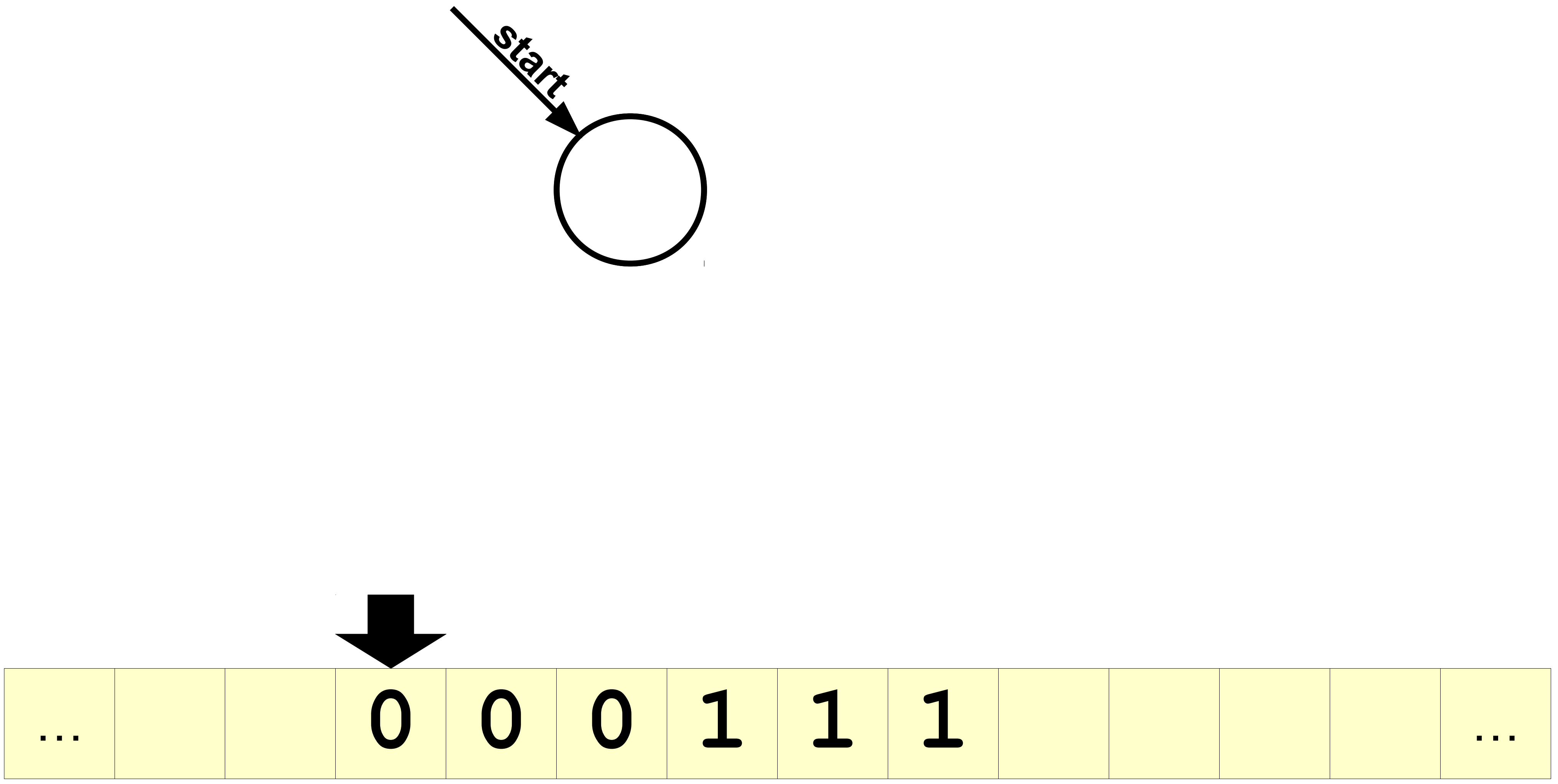
A Sketch of the TM



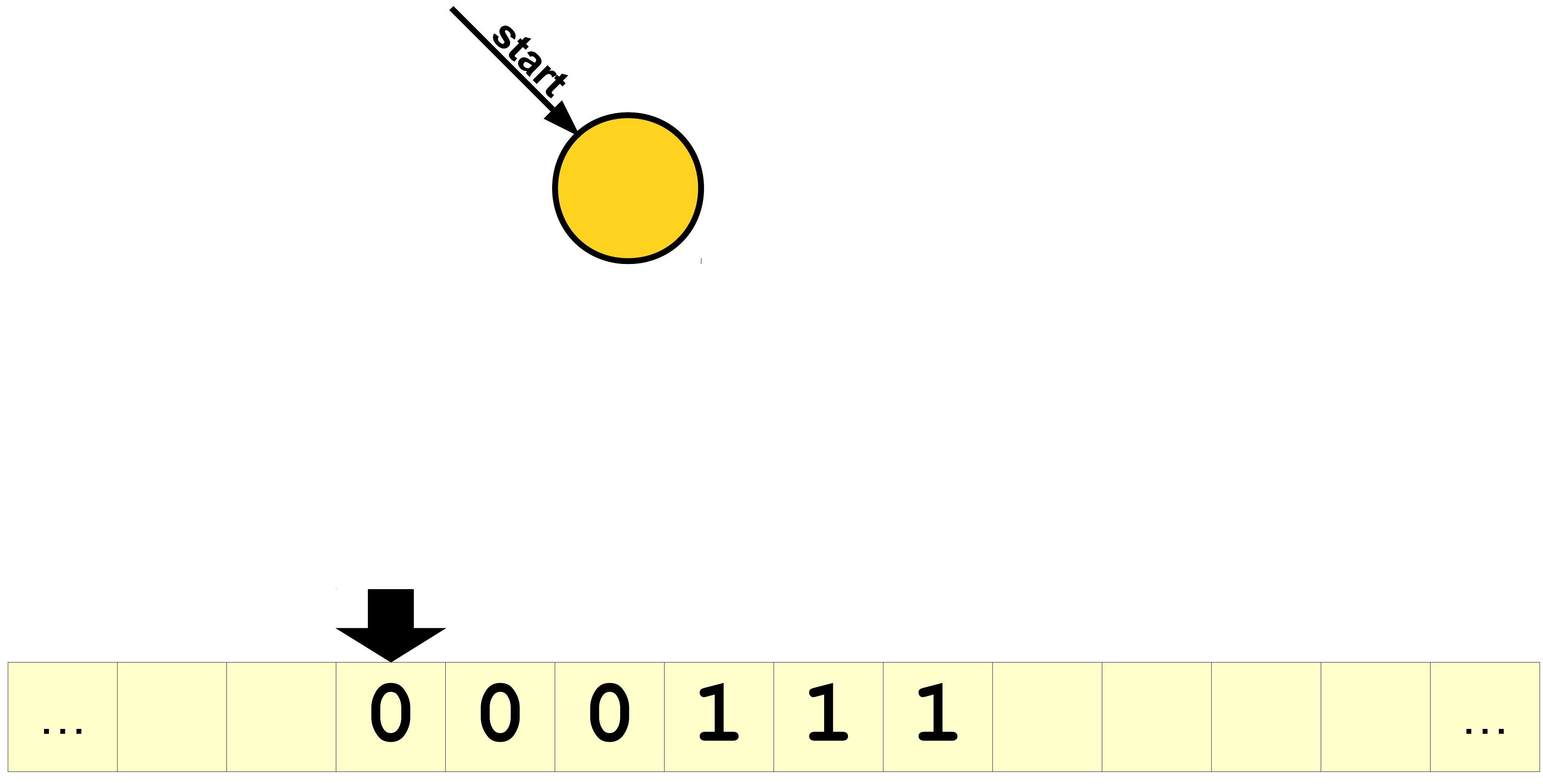
A Sketch of the TM

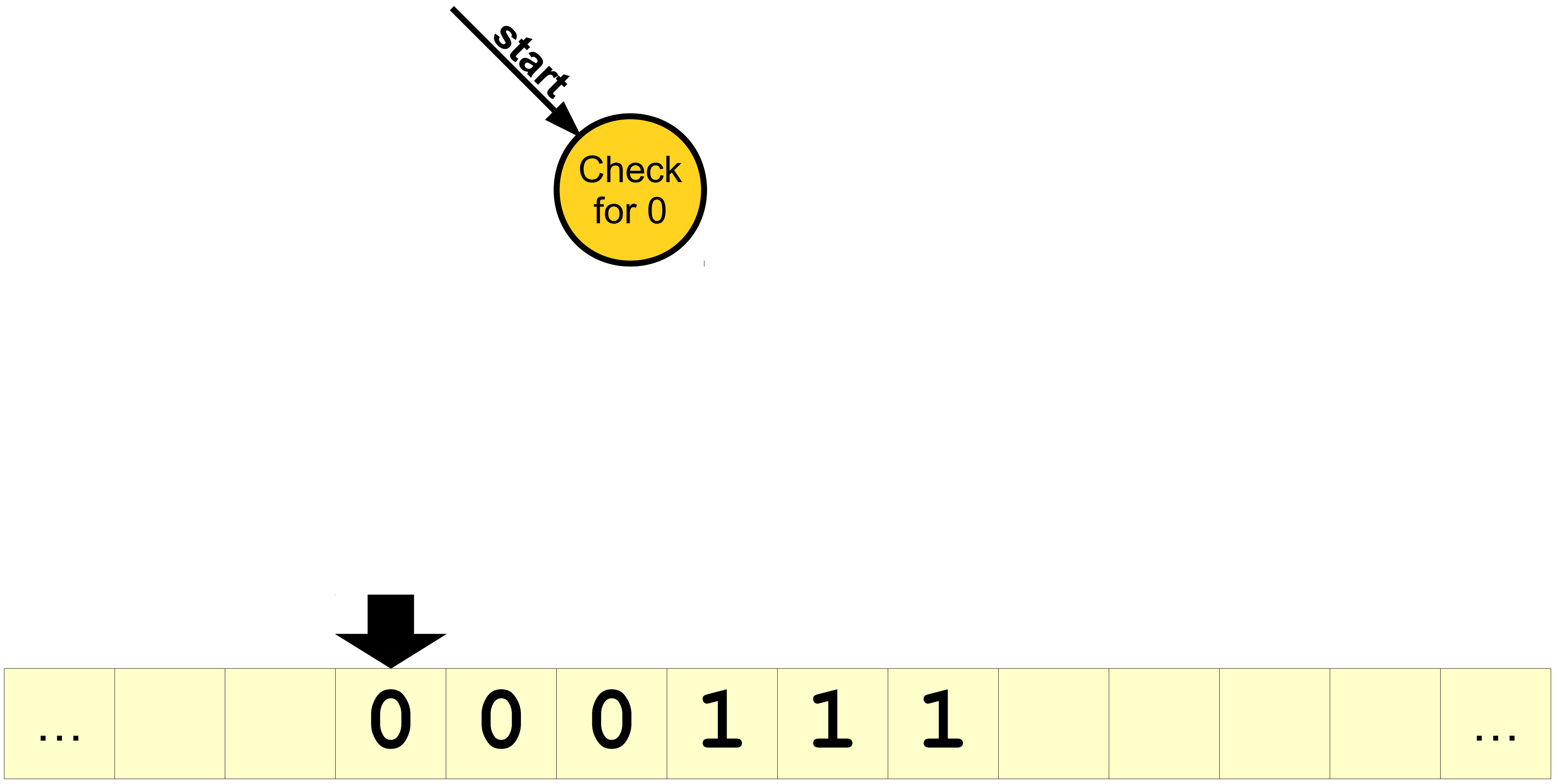


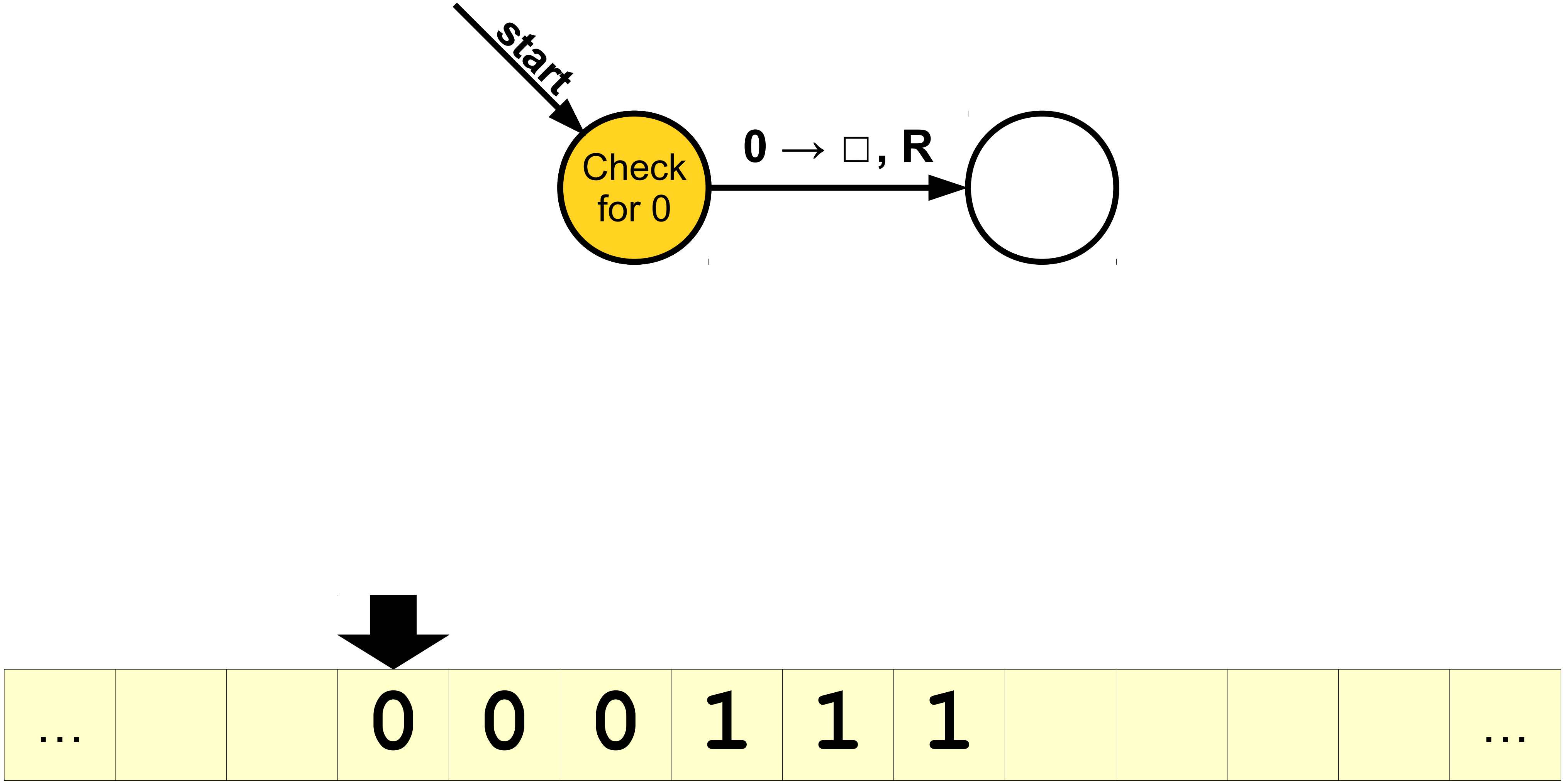

start

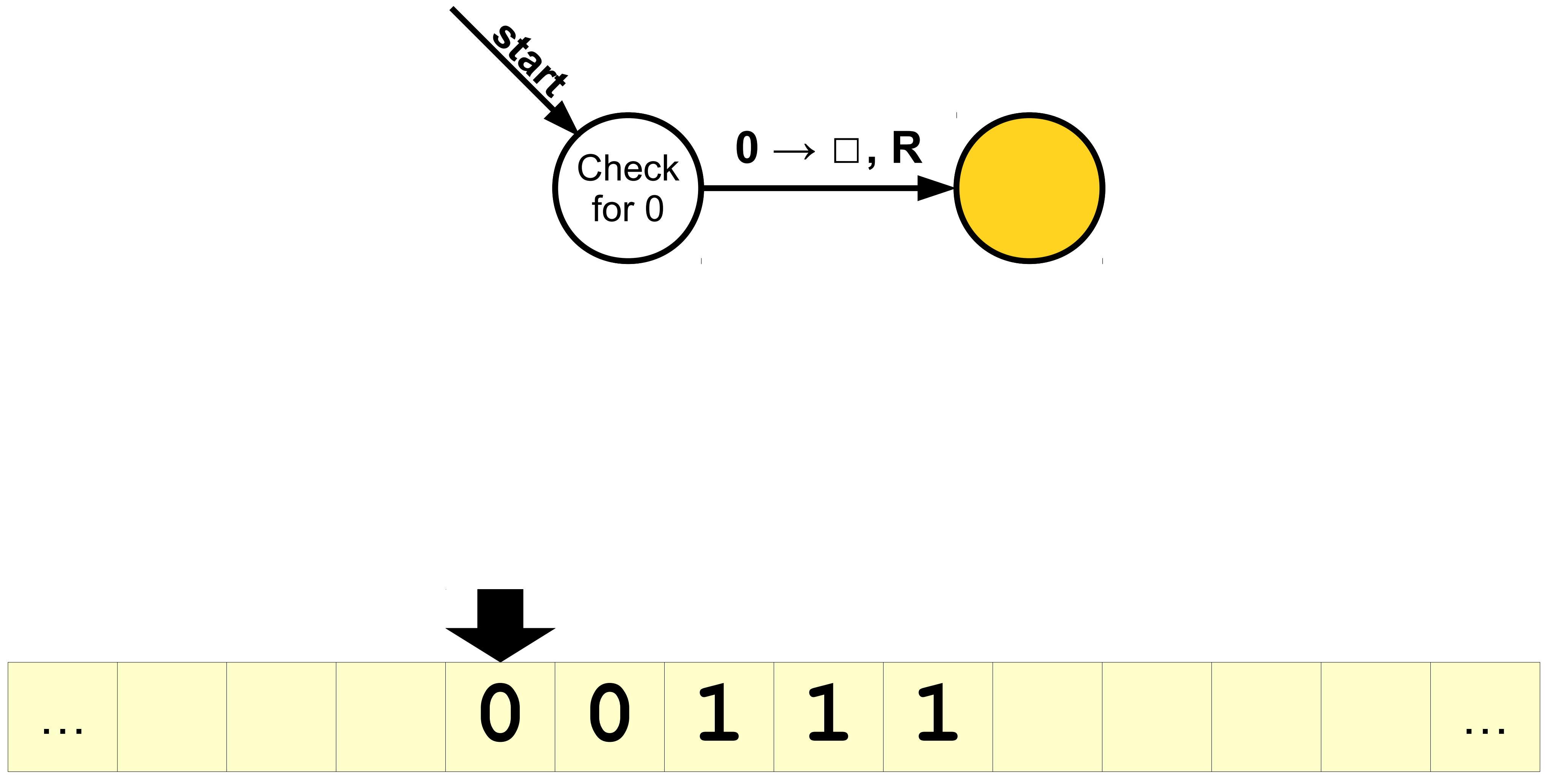


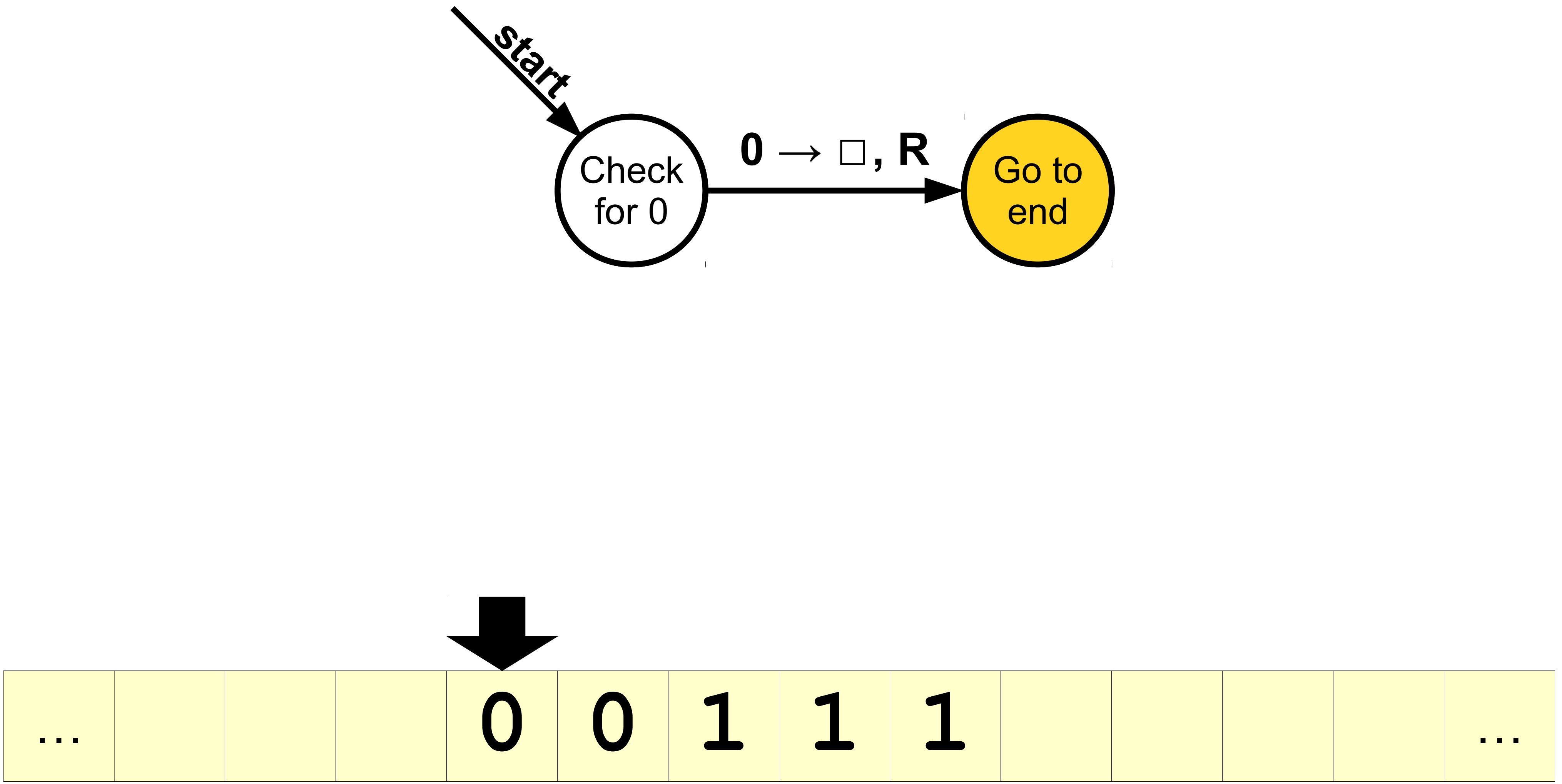
start
…
0
0
0
1
1
1
…

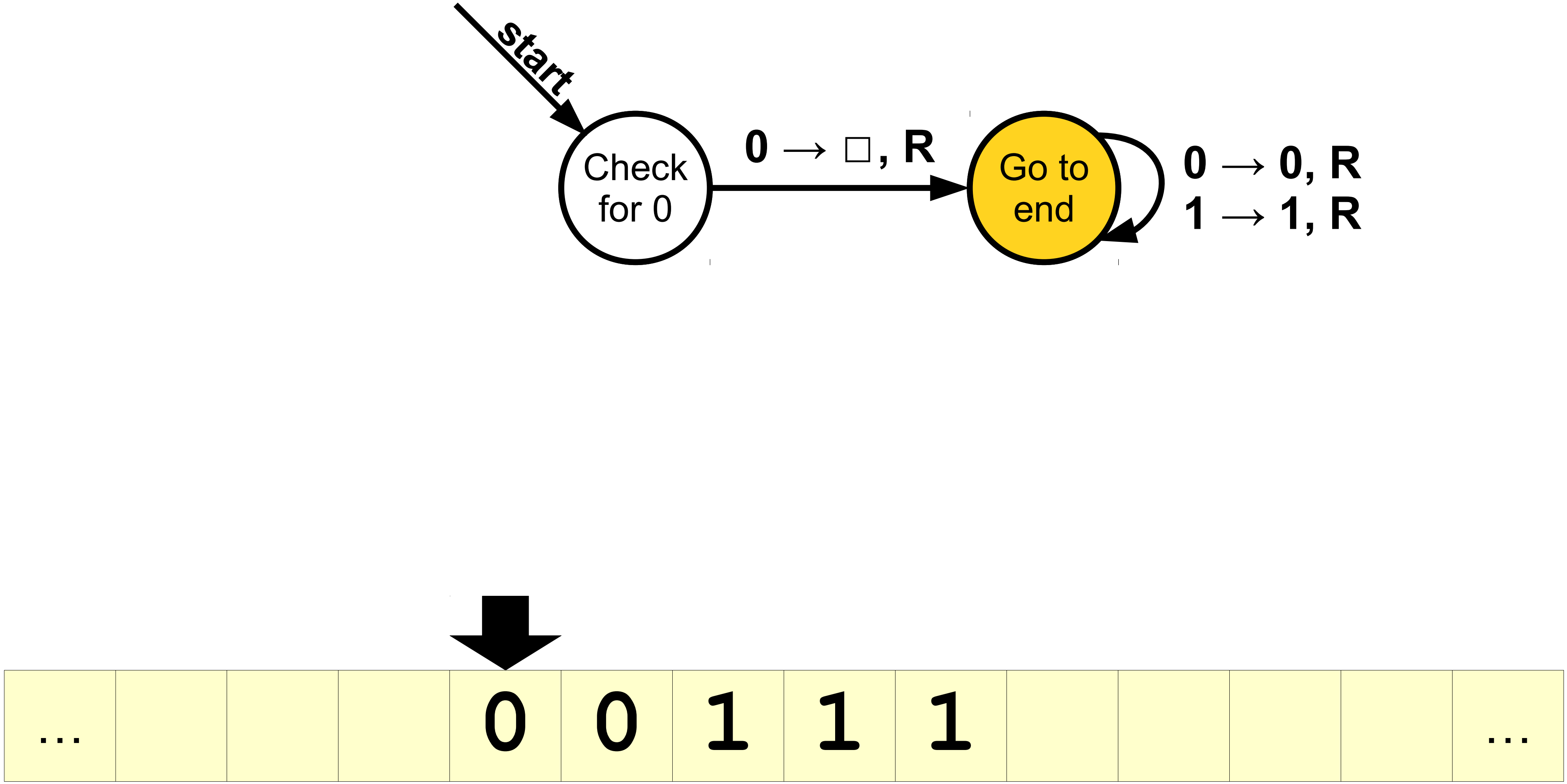

start
…
0
0
0
1
1
1
…


start
Check for 0
...
0
0
0
1
1
1
...

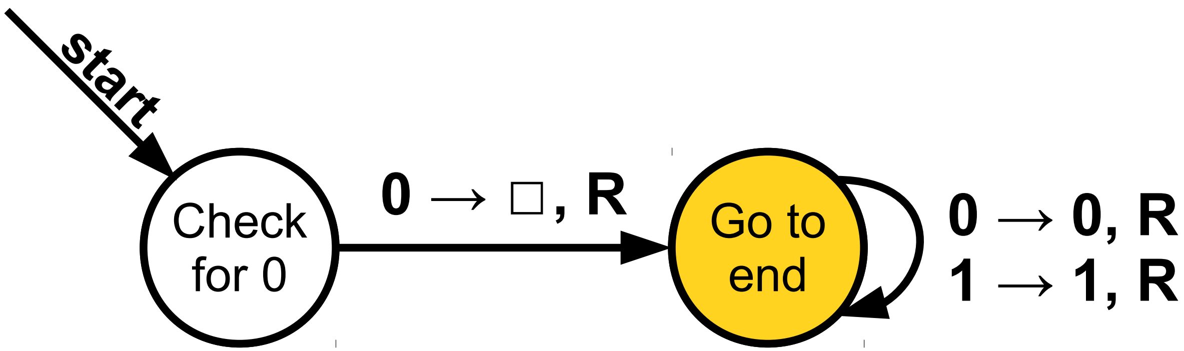

start
Check for 0
0 → □, R
…
0
0
0
1
1
1
…


start
Check for 0
0 → □, R
…
0
0
1
1
1
…

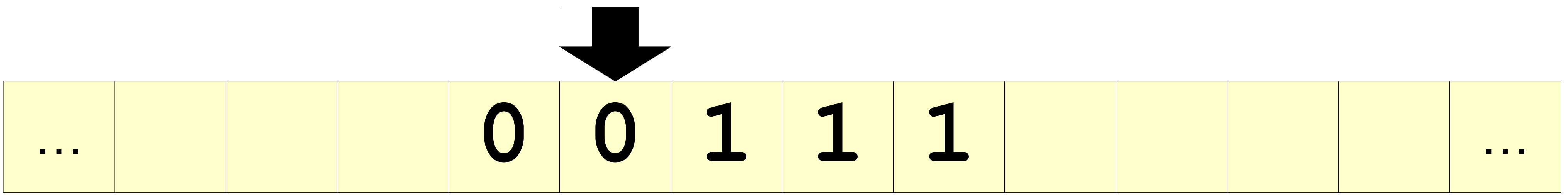

start
Check for 0
0 → □, R
Go to end
...
0
0
1
1
1
...



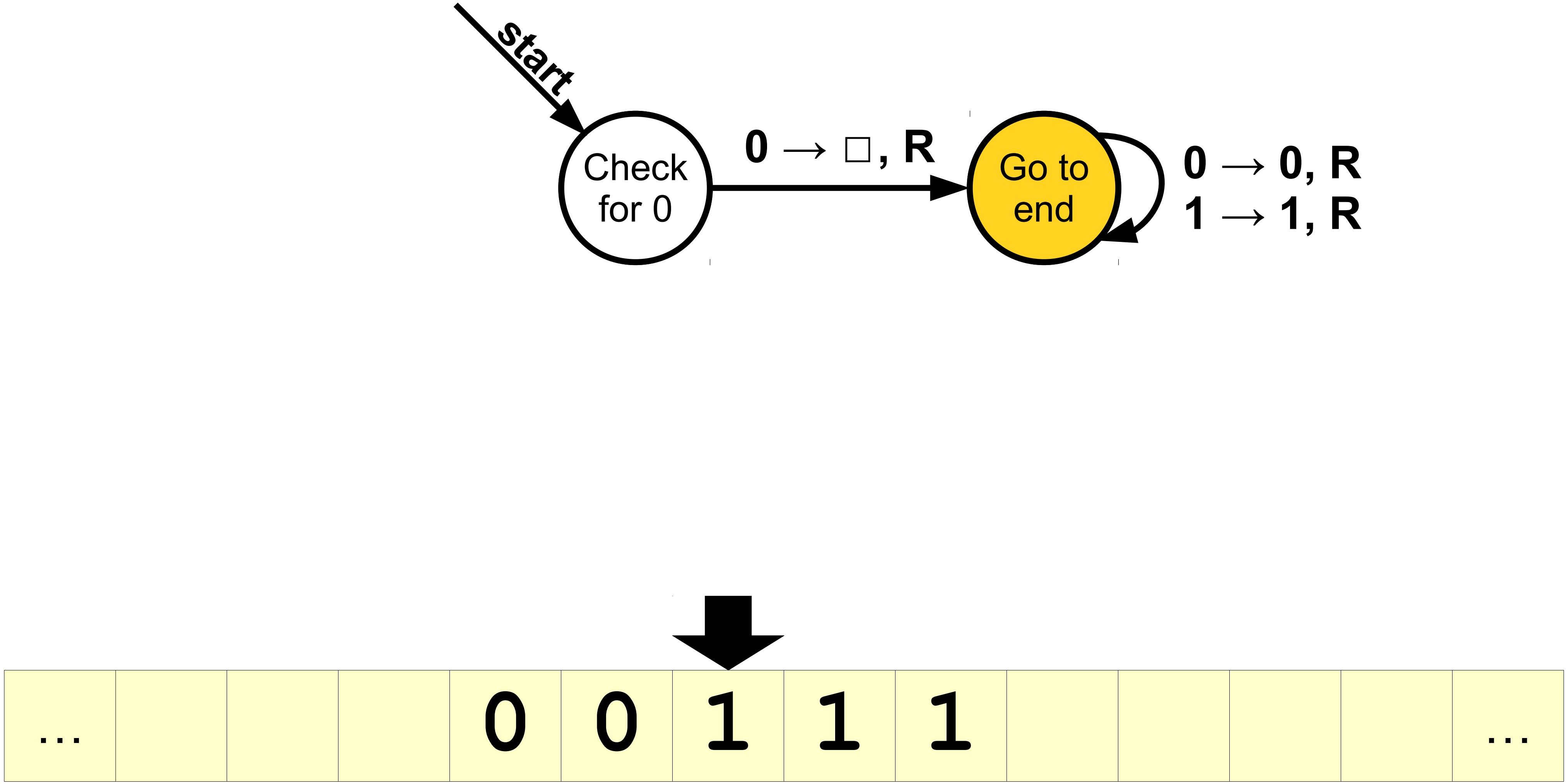
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
1
…

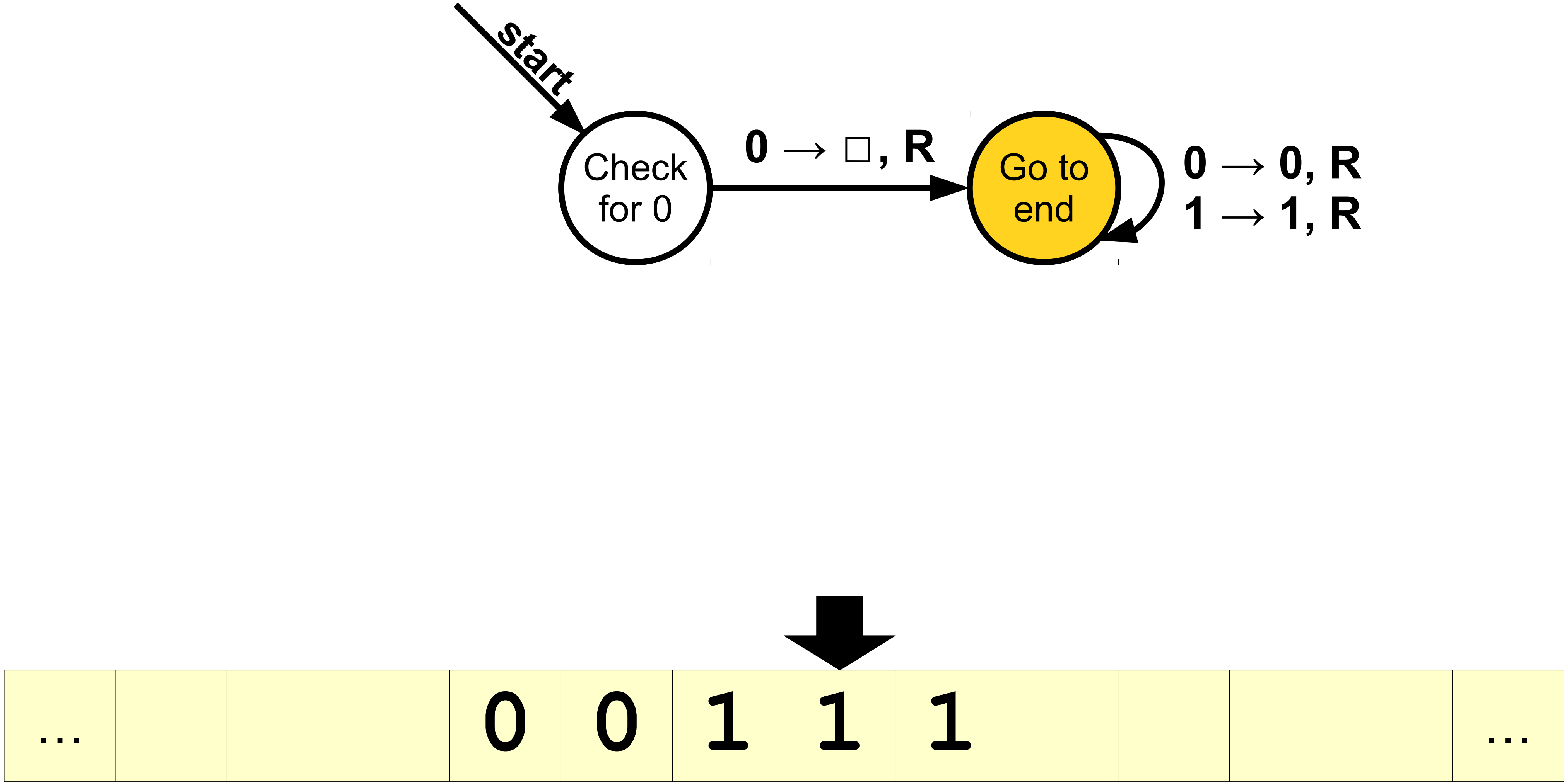


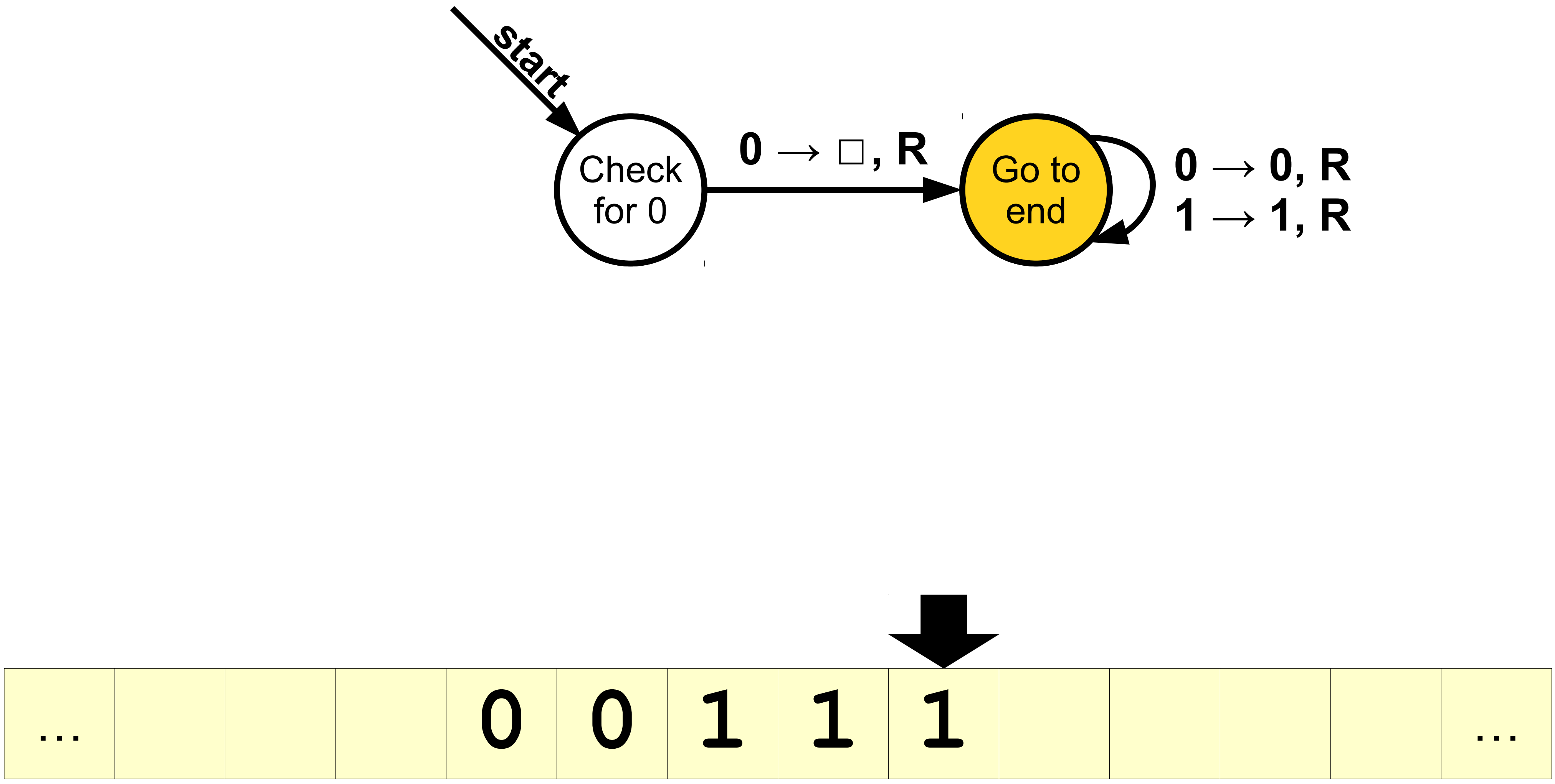
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R



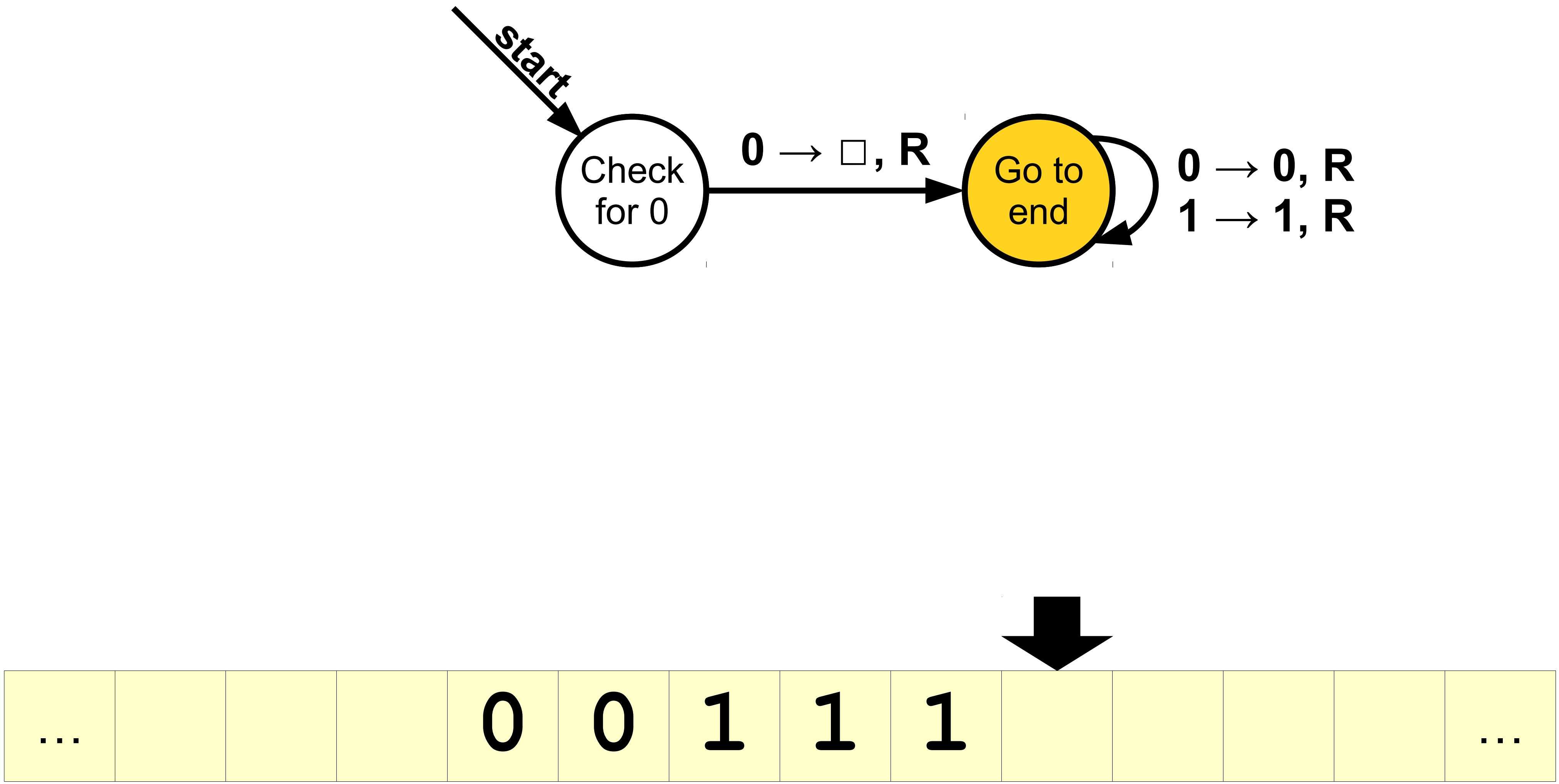
…
0
0
1
1
1
…

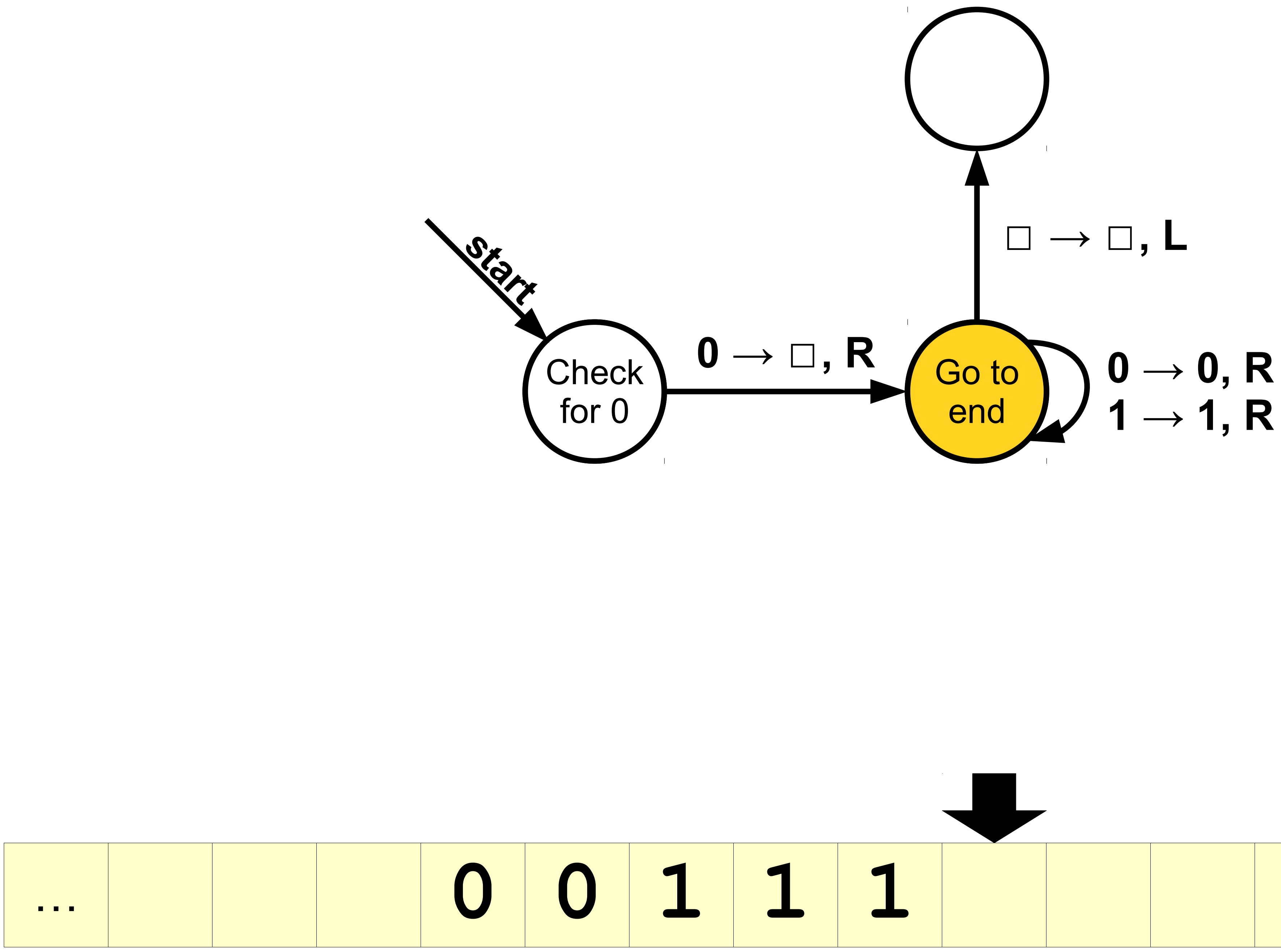

start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
1
…

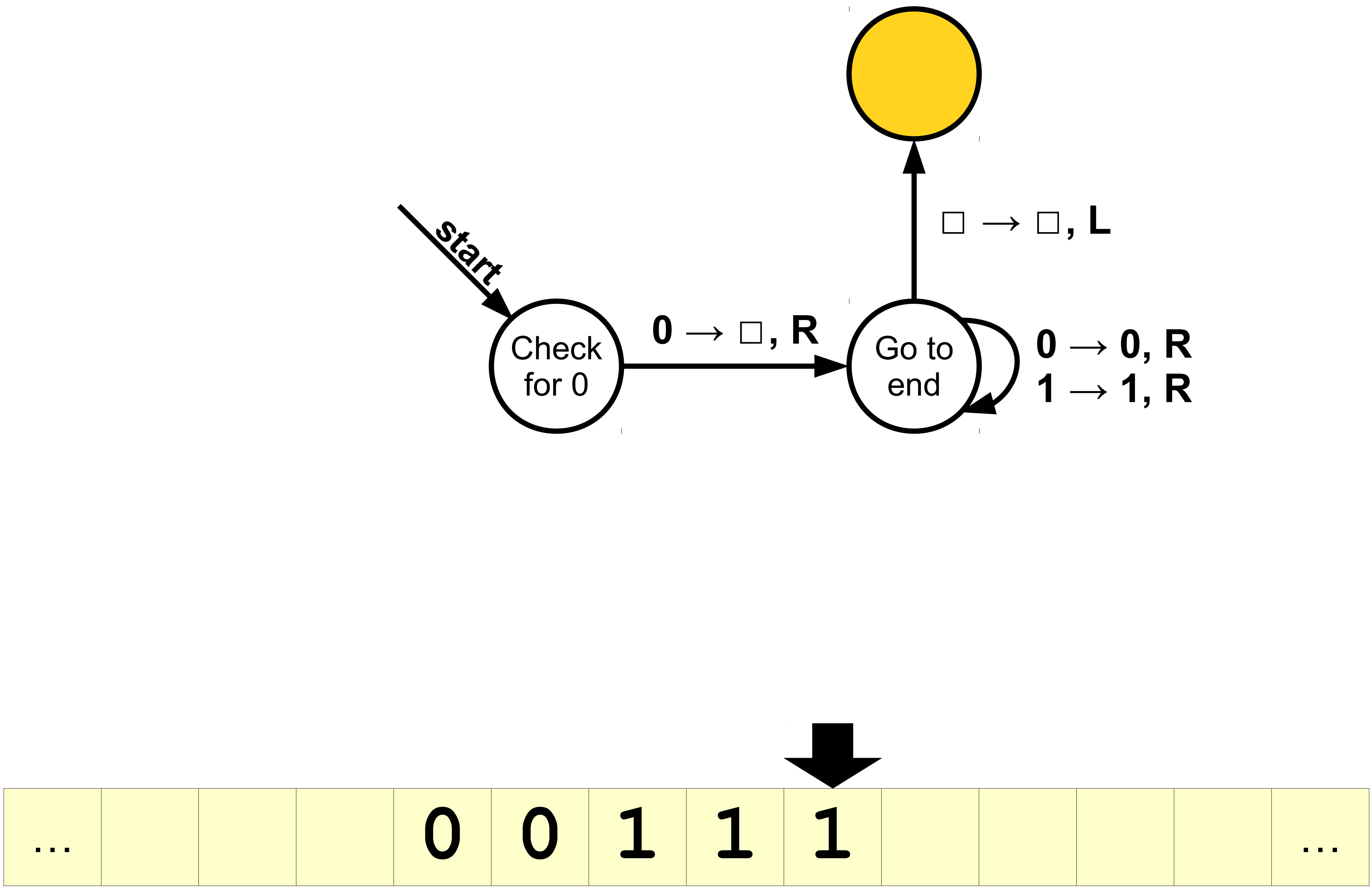

start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
1
…



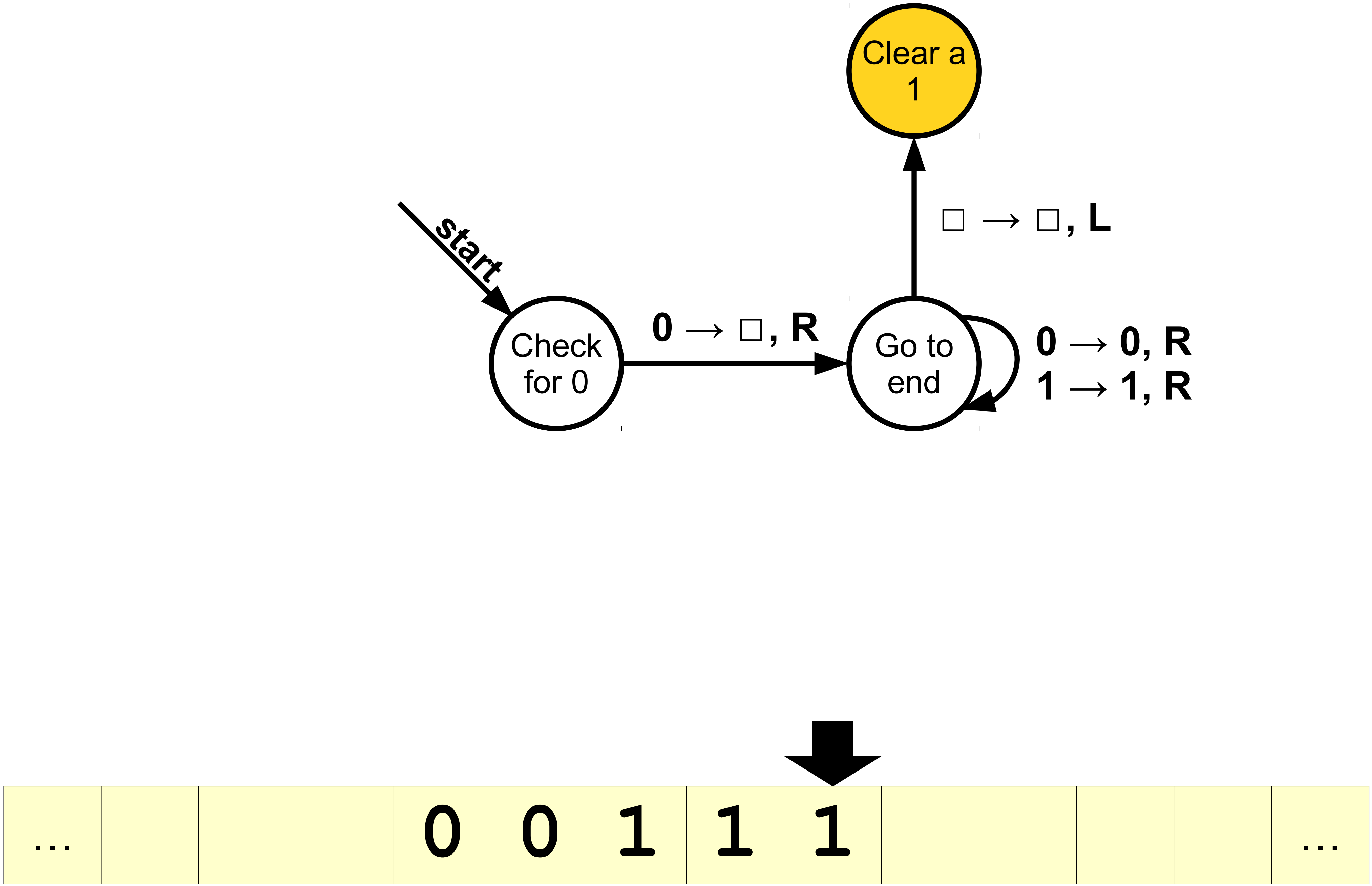
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
1
…


start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
0
0
1
1
1
...

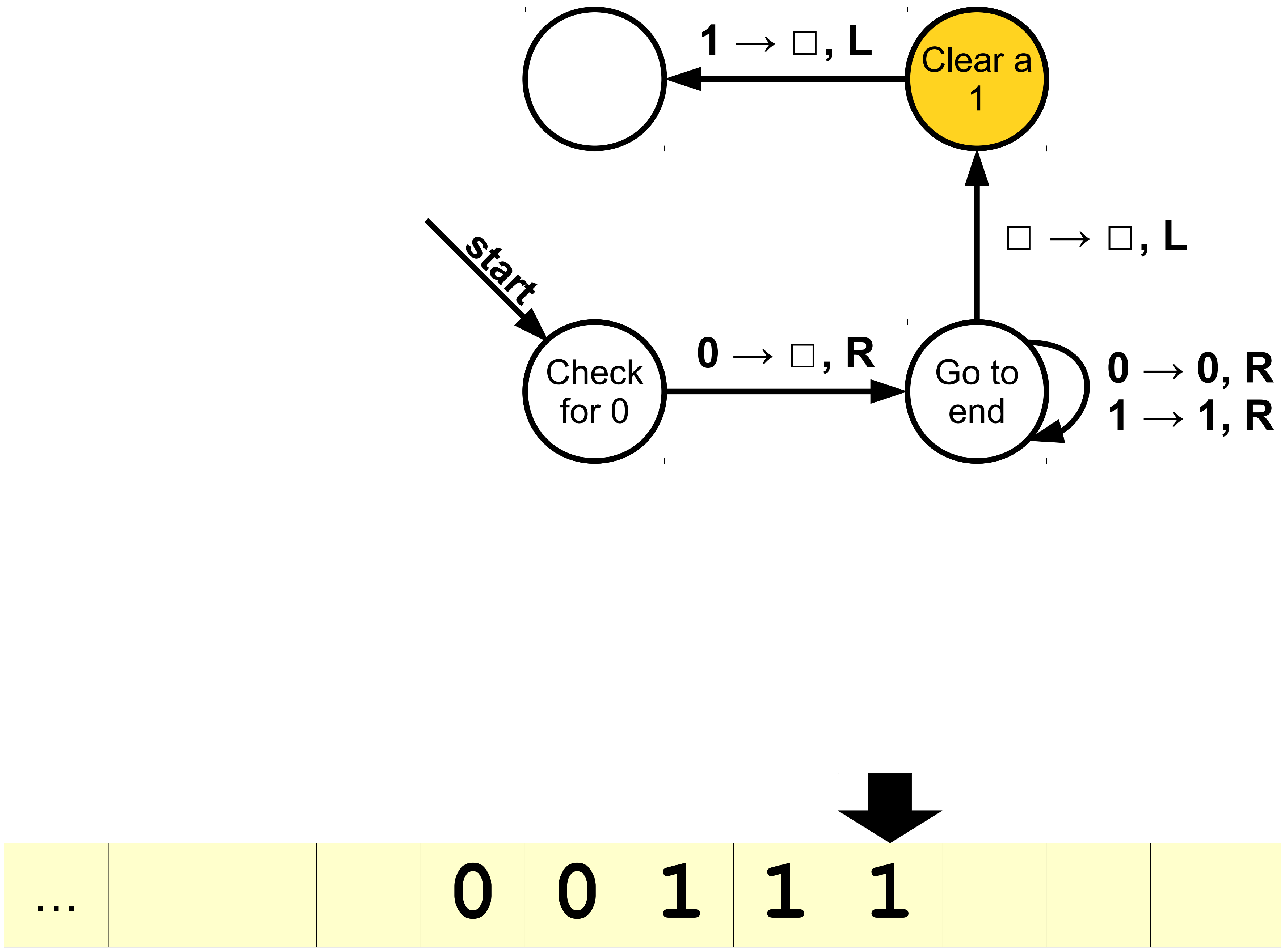

start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, L
…
0
0
1
1
1
…

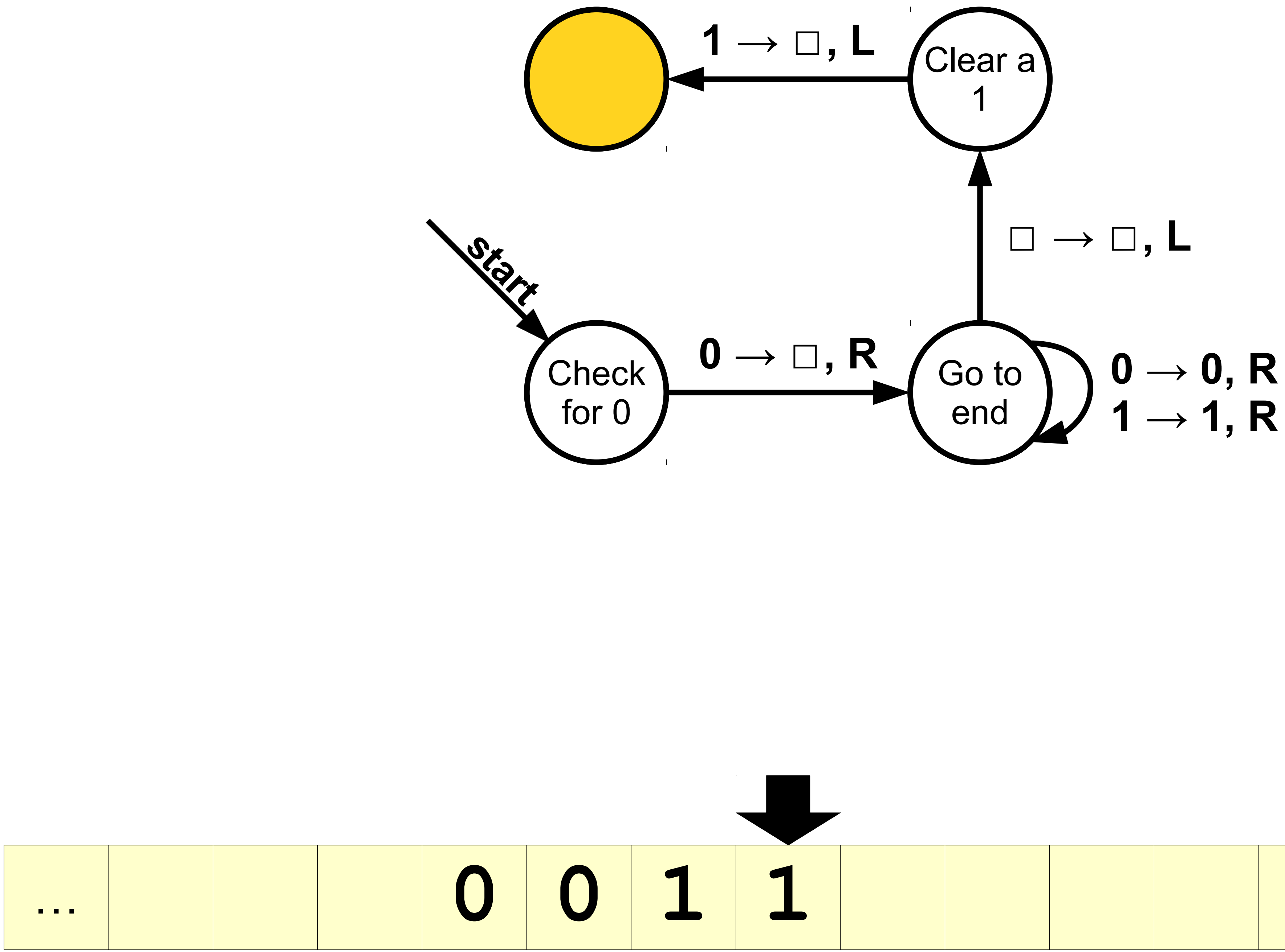


start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, L
… 0 0 1 1 1 …

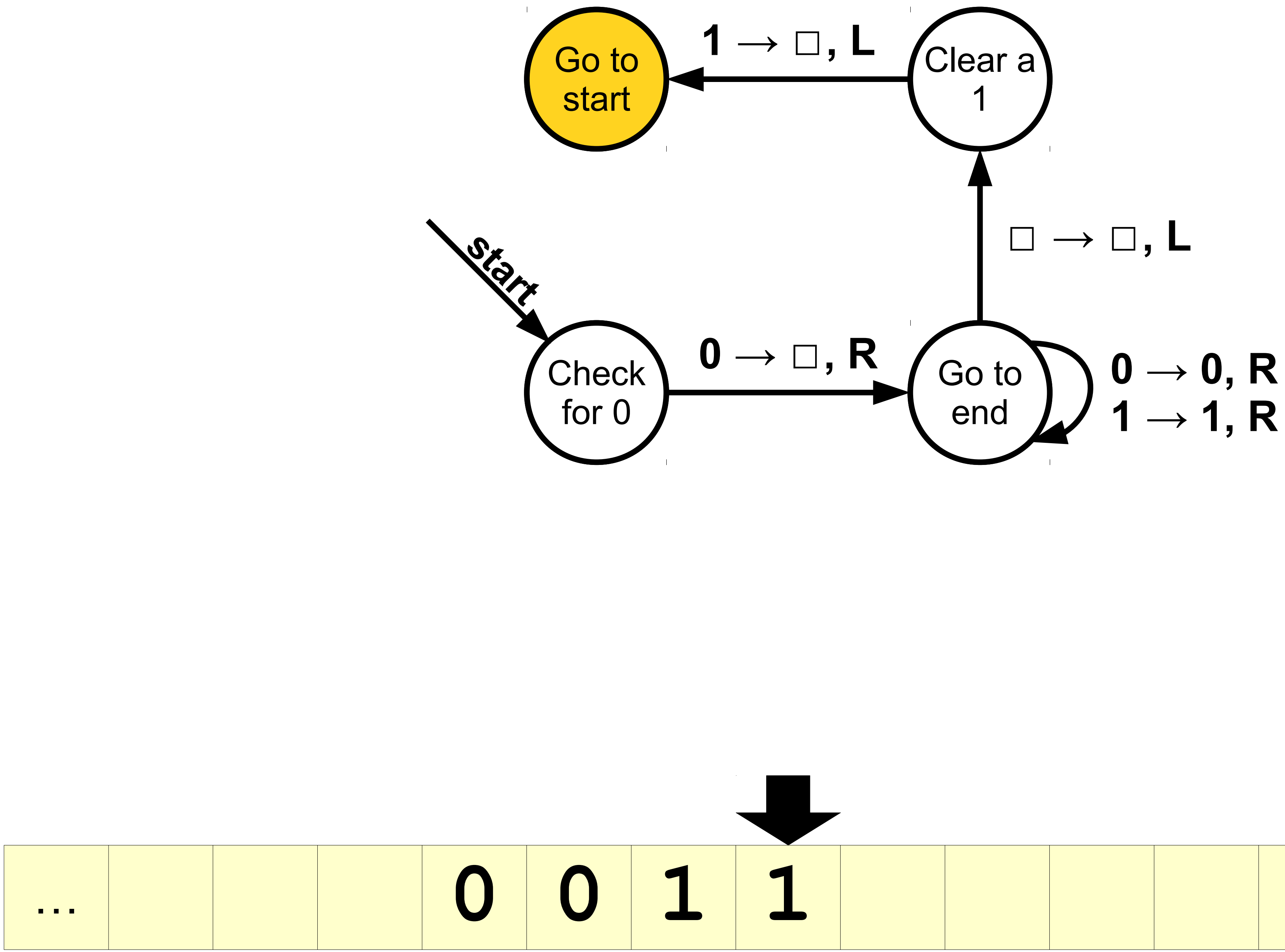


start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, L
Clear a 1
...
0
0
1
1
1
...

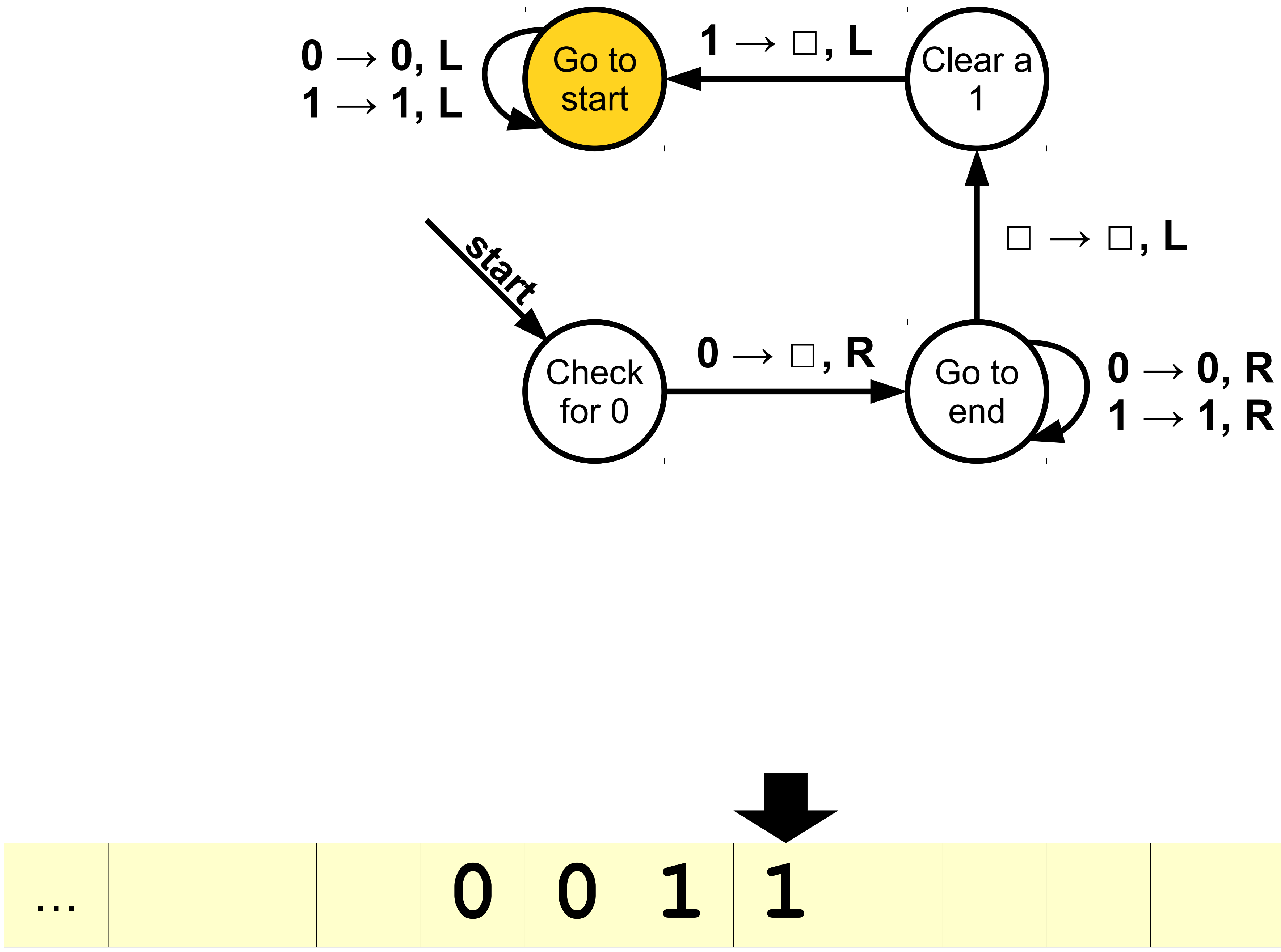

1 → □, L
Clear a 1
start
□ → □, L
0 → □, R
Check for 0
Go to end
0 → 0, R
1 → 1, R
...
0
0
1
1
1
...



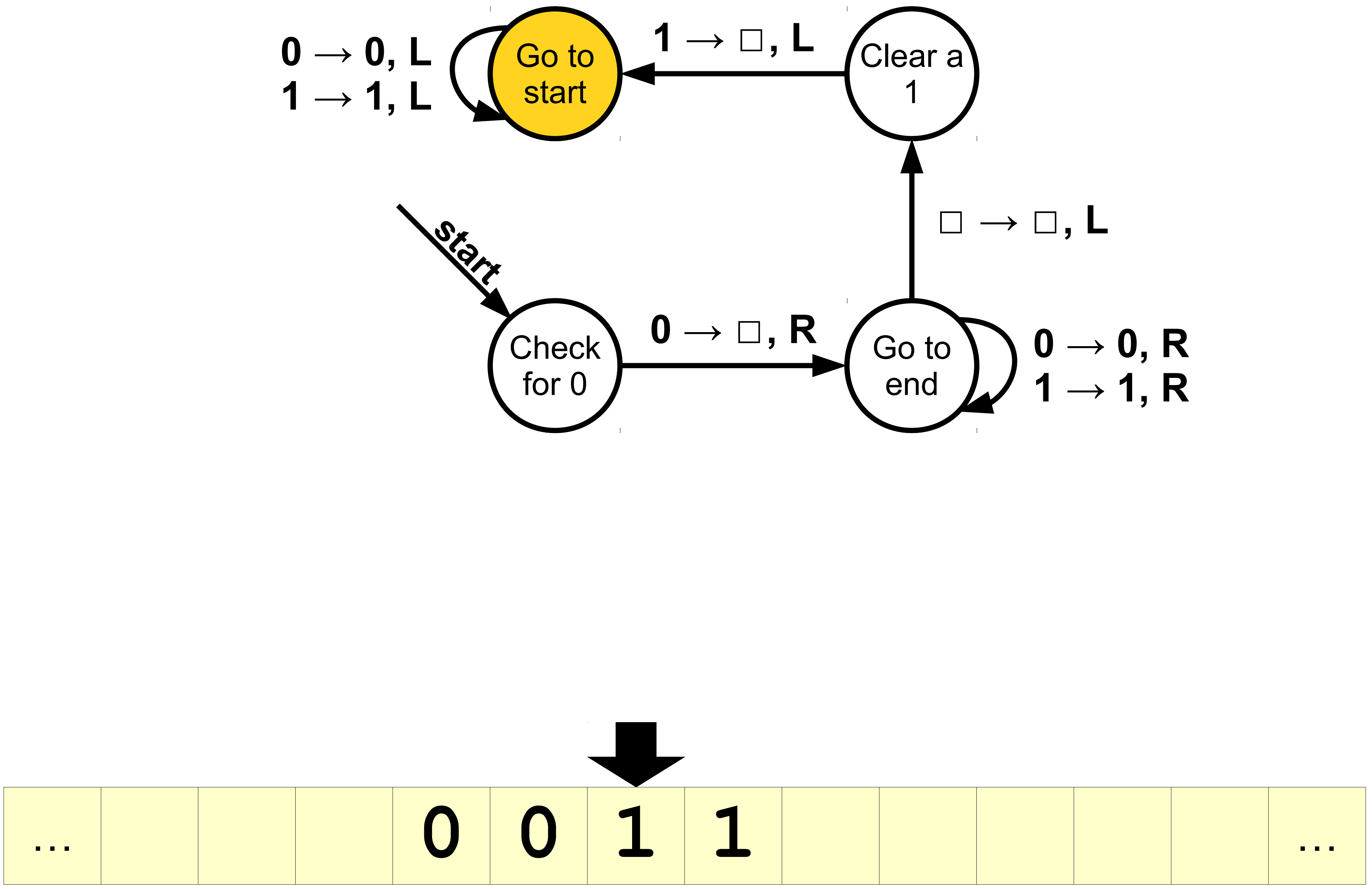
1 → □, L
Clear a 1
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
…

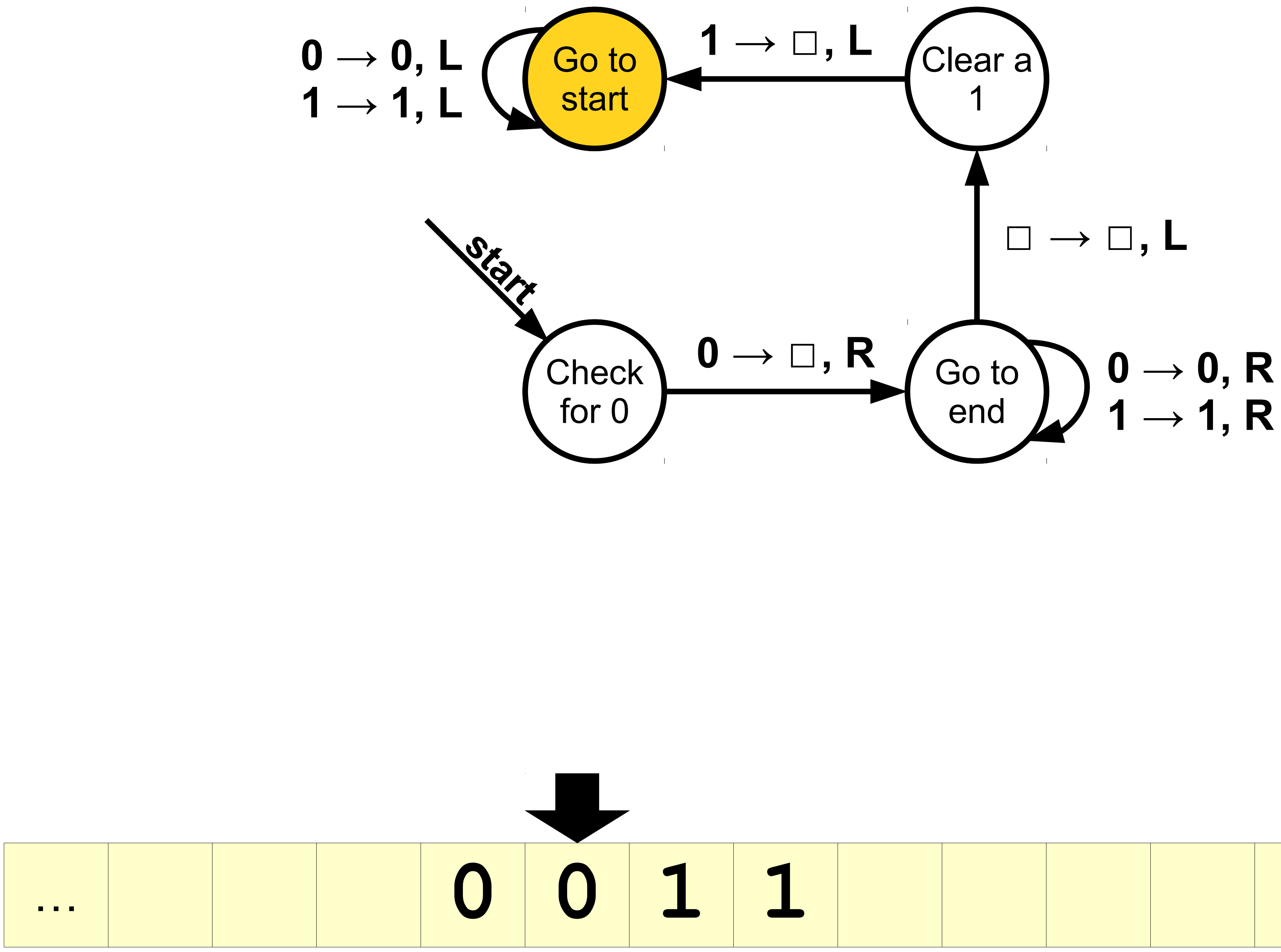


Go to start
1 → □, L
Clear a 1
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
… 0 0 1 1 …

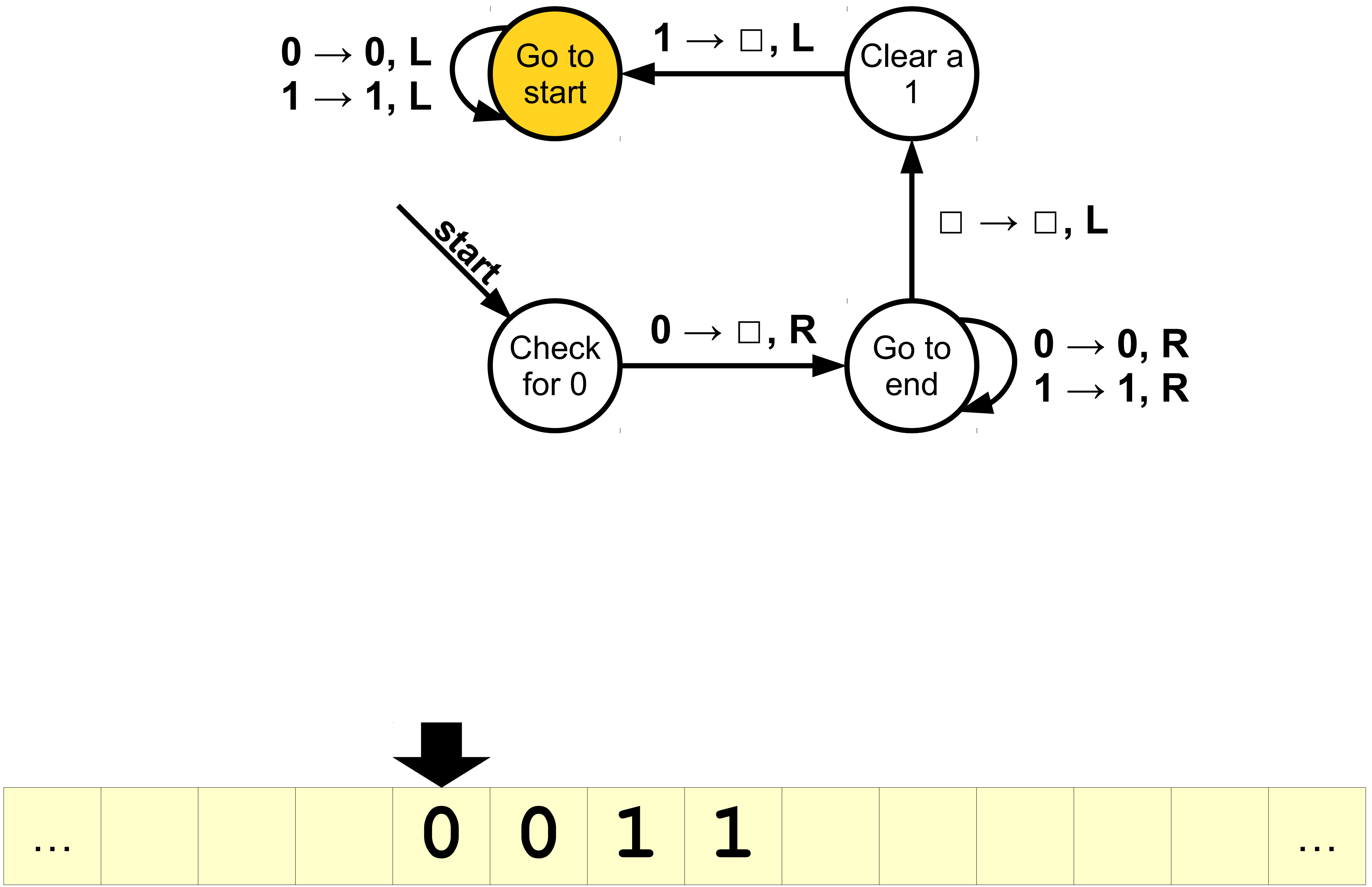


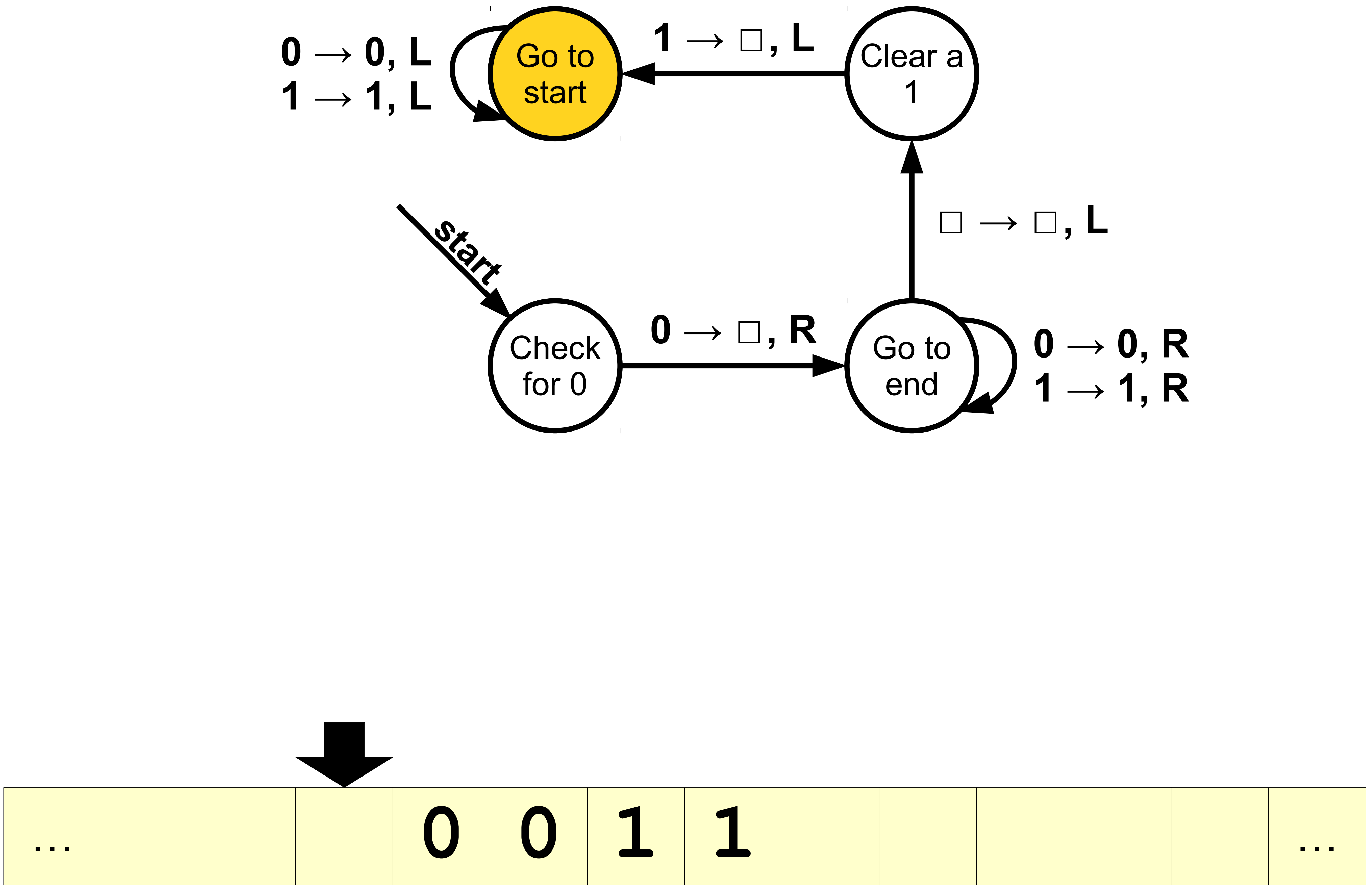
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
0
1
1
…

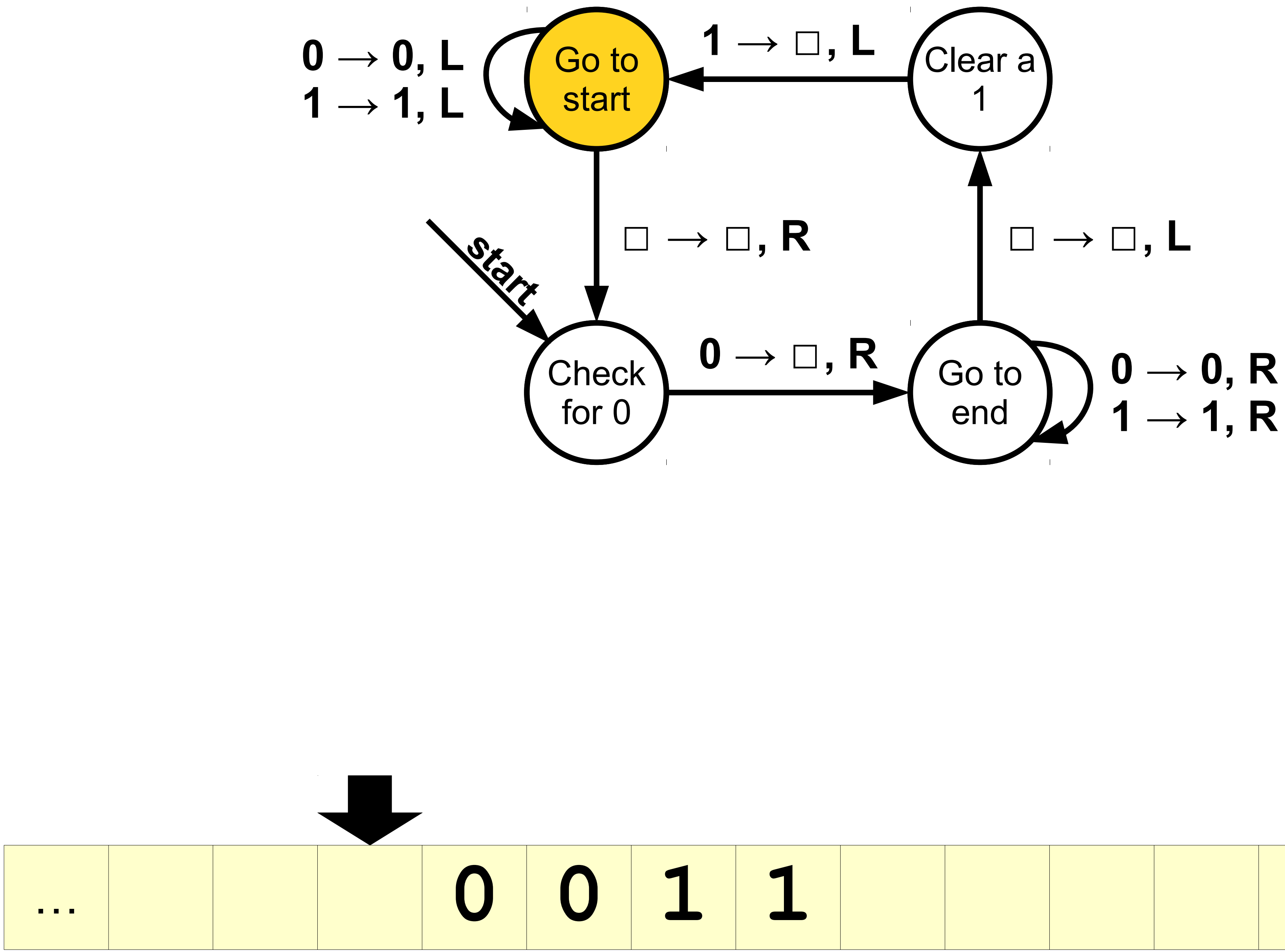

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
…



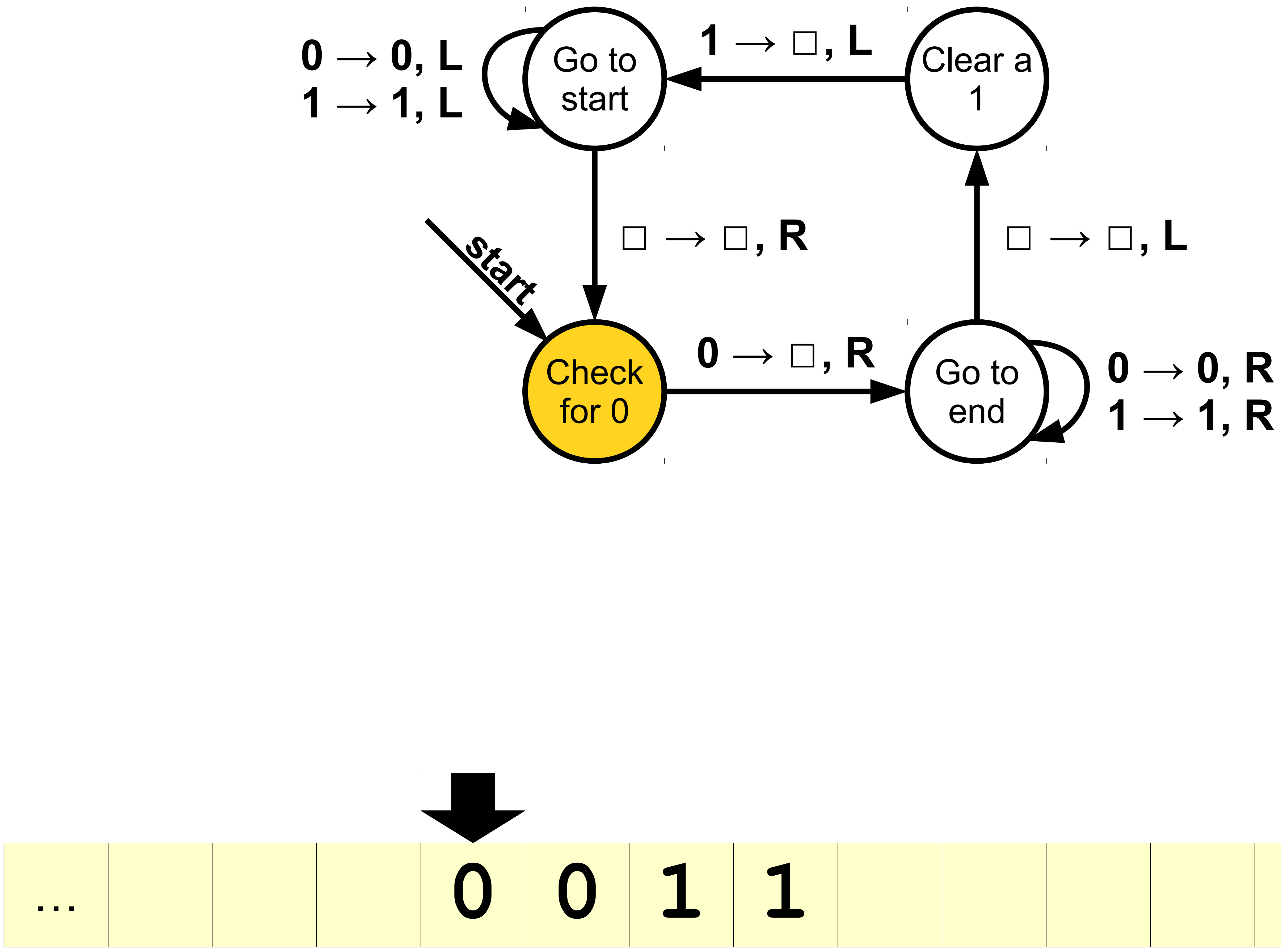
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
0 0 1 1
...


Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
0
1
1
…

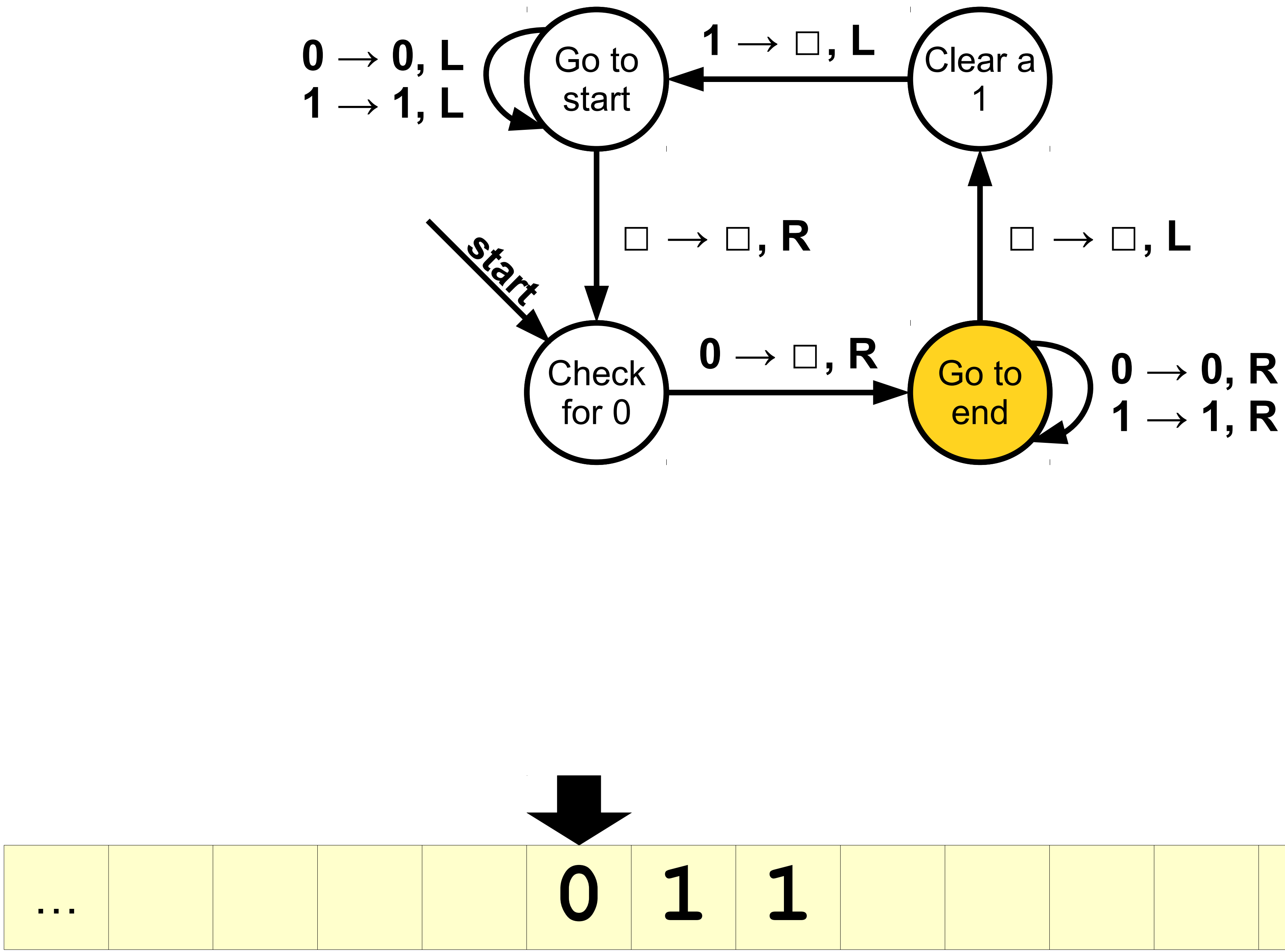

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
…



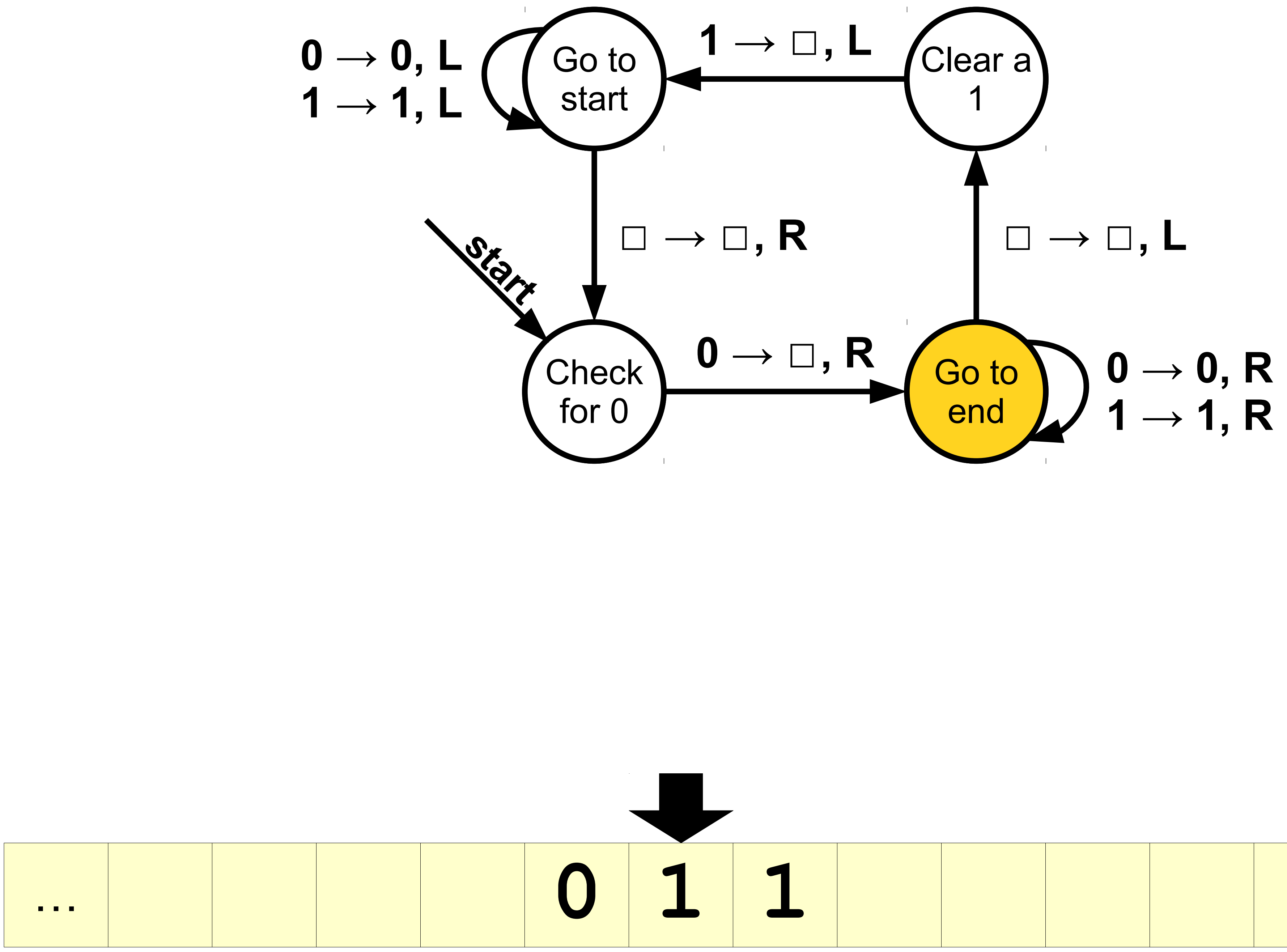
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
0
1
1
…

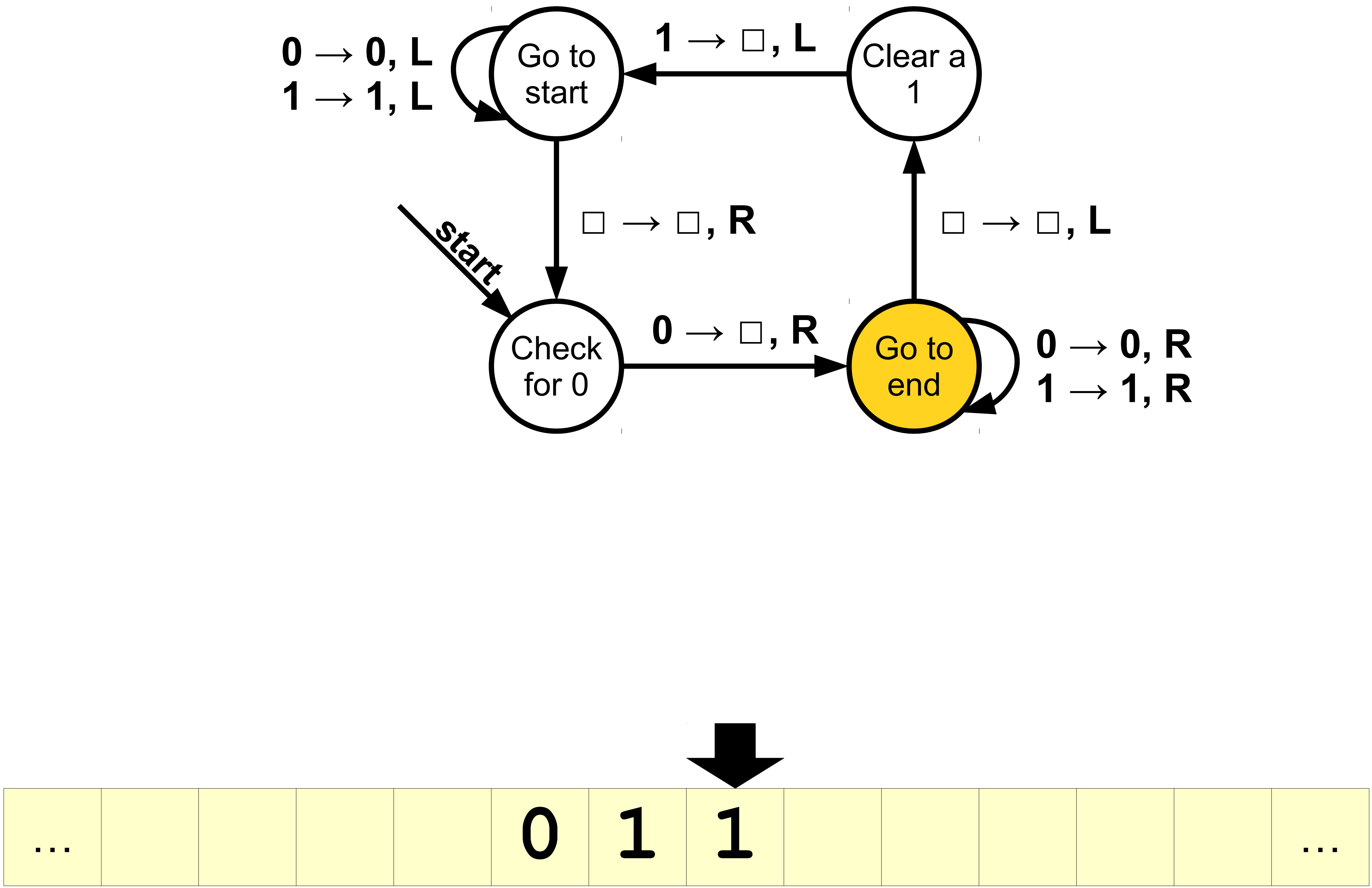


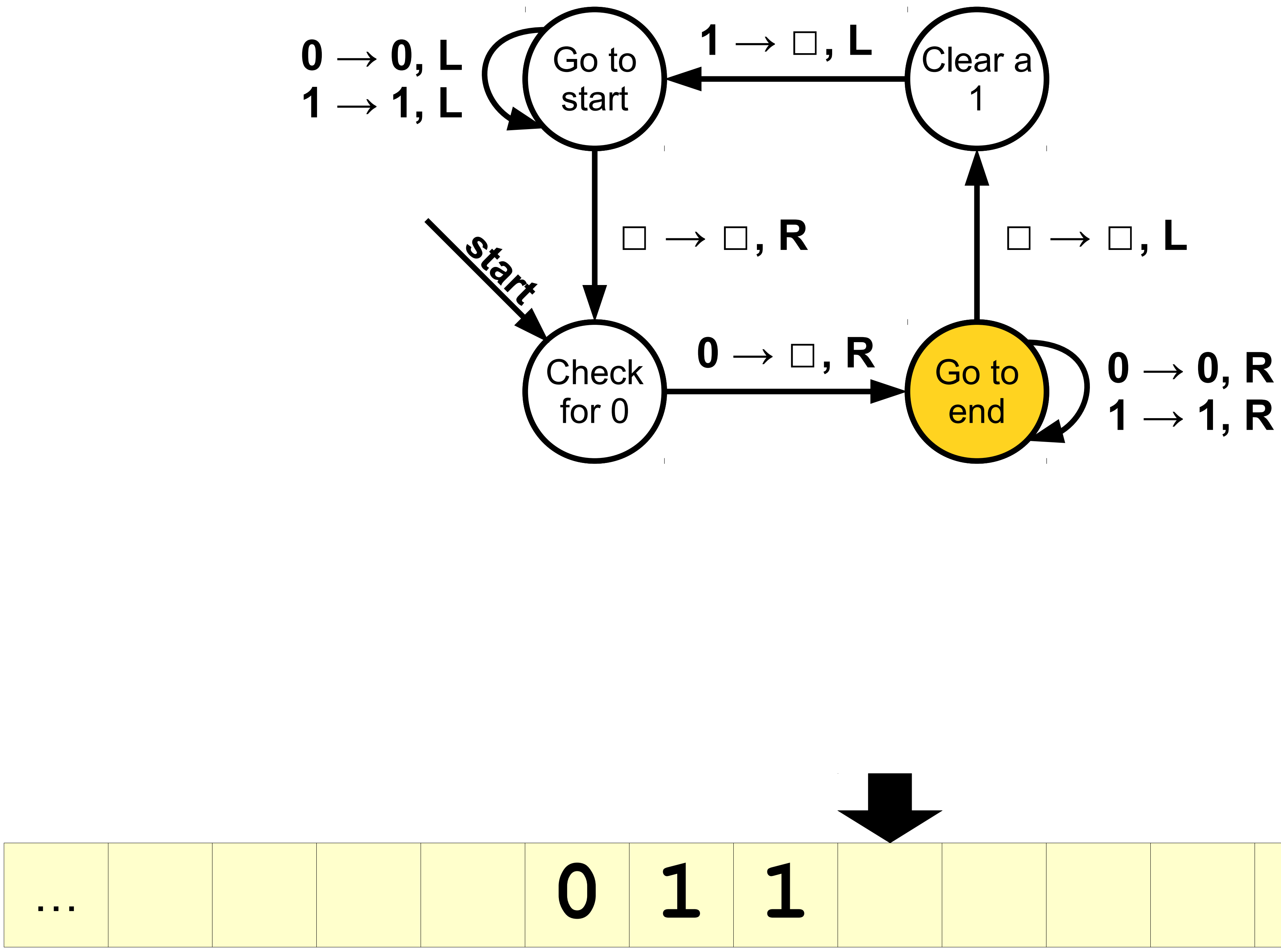
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
0
1
1
…

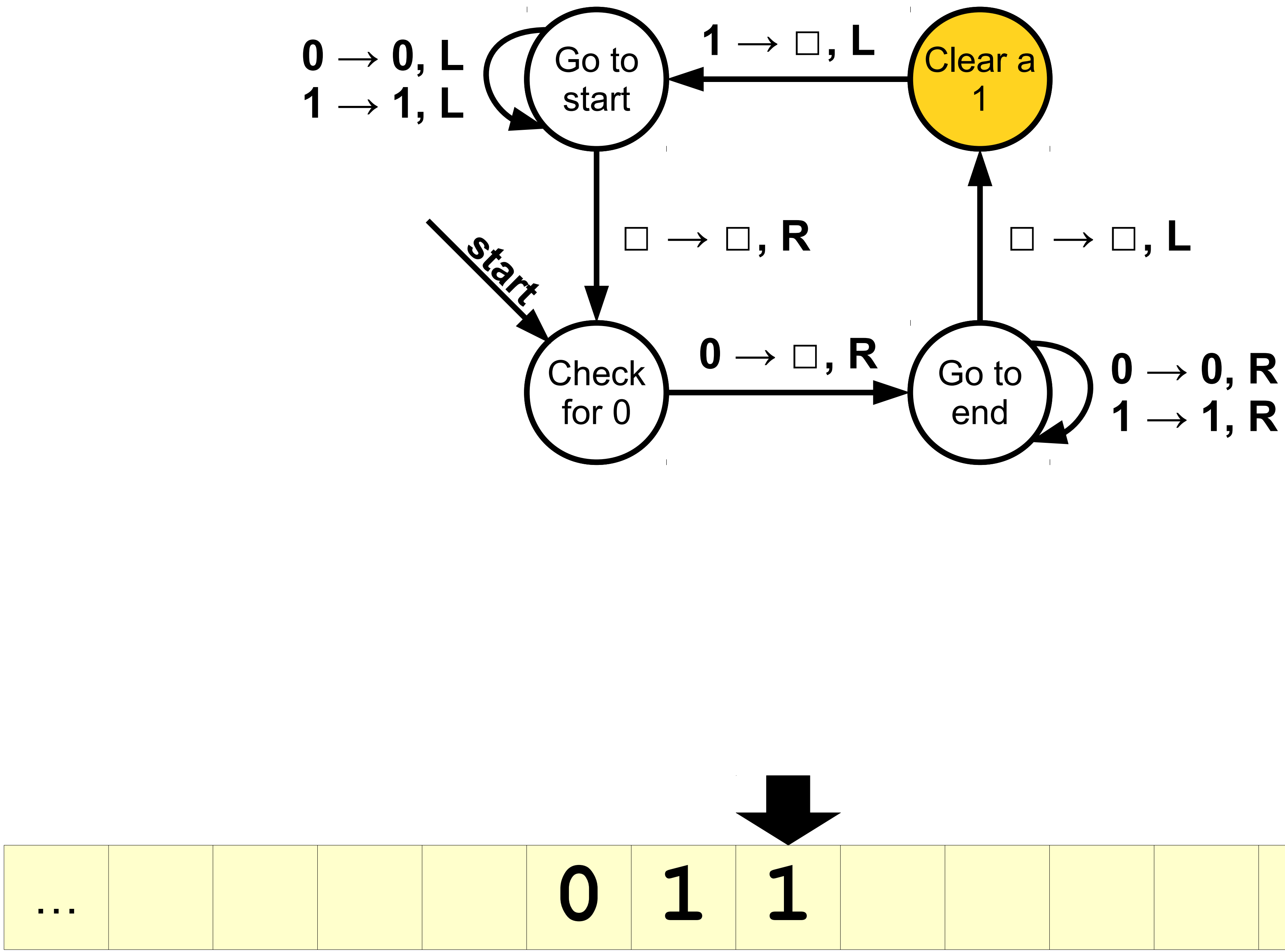


Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
1
1
…

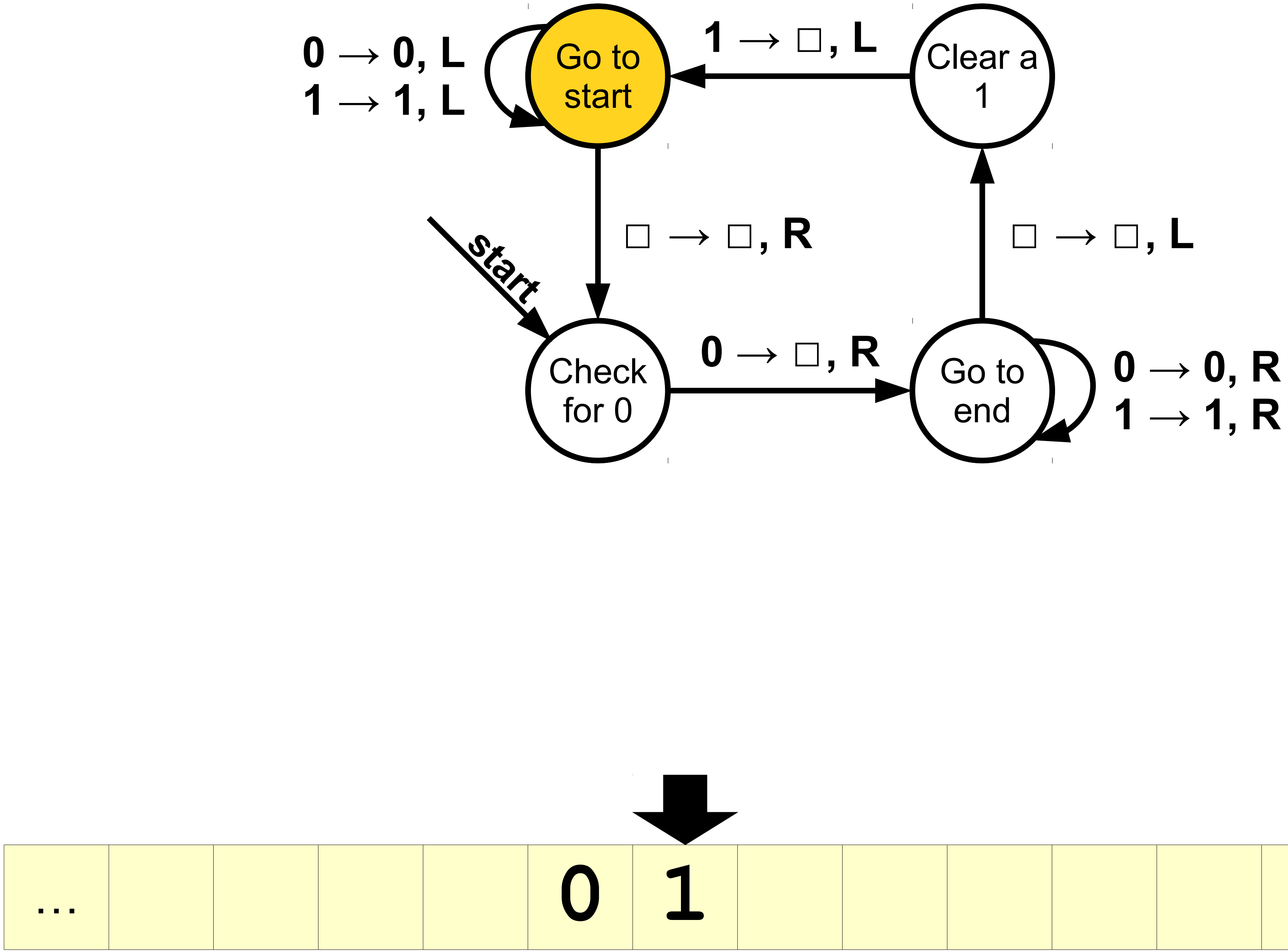

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
0
1
1
...

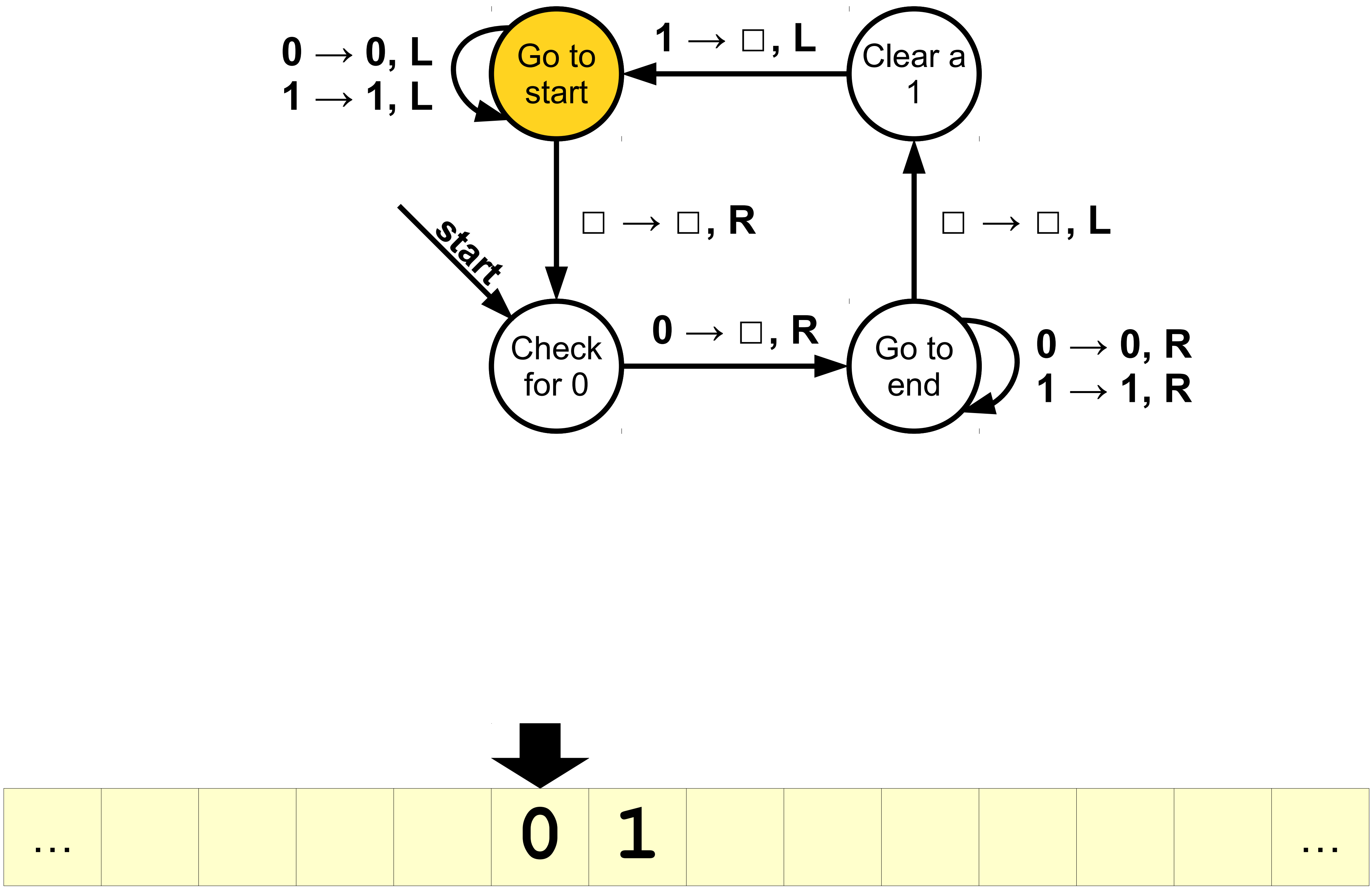

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
0
1
1
...


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
0
1
1
...

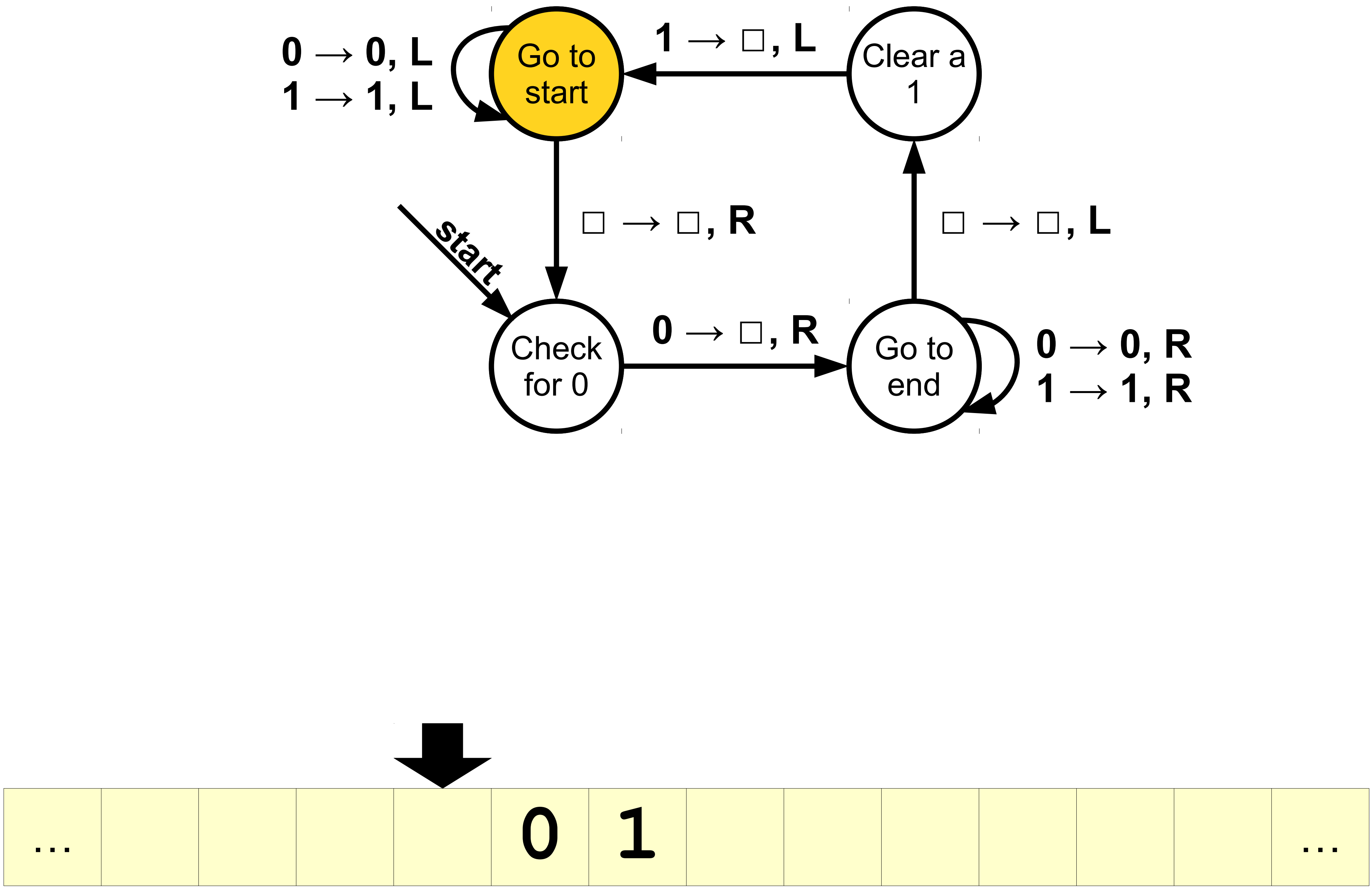


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
1
1
…

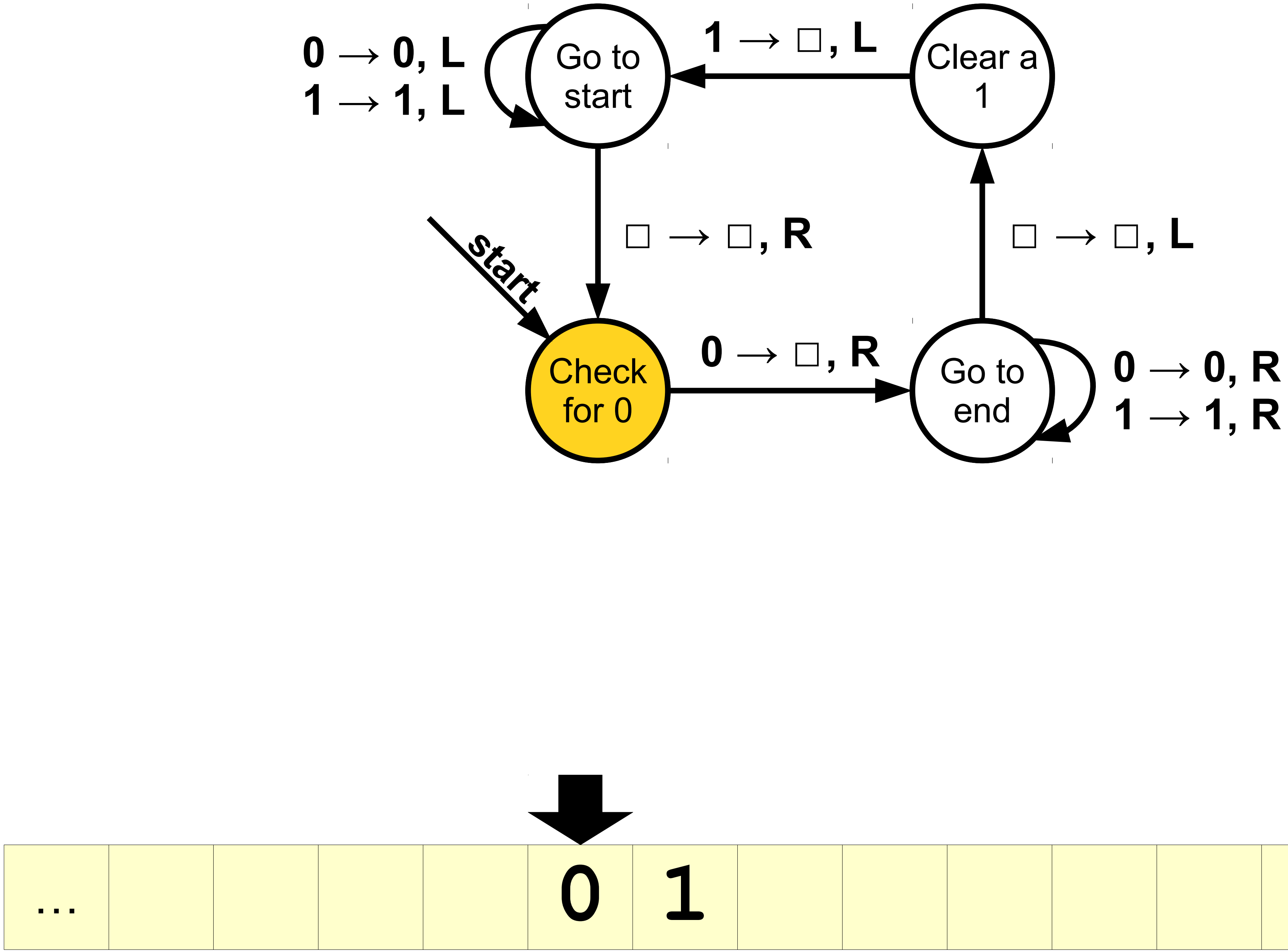

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
1
…



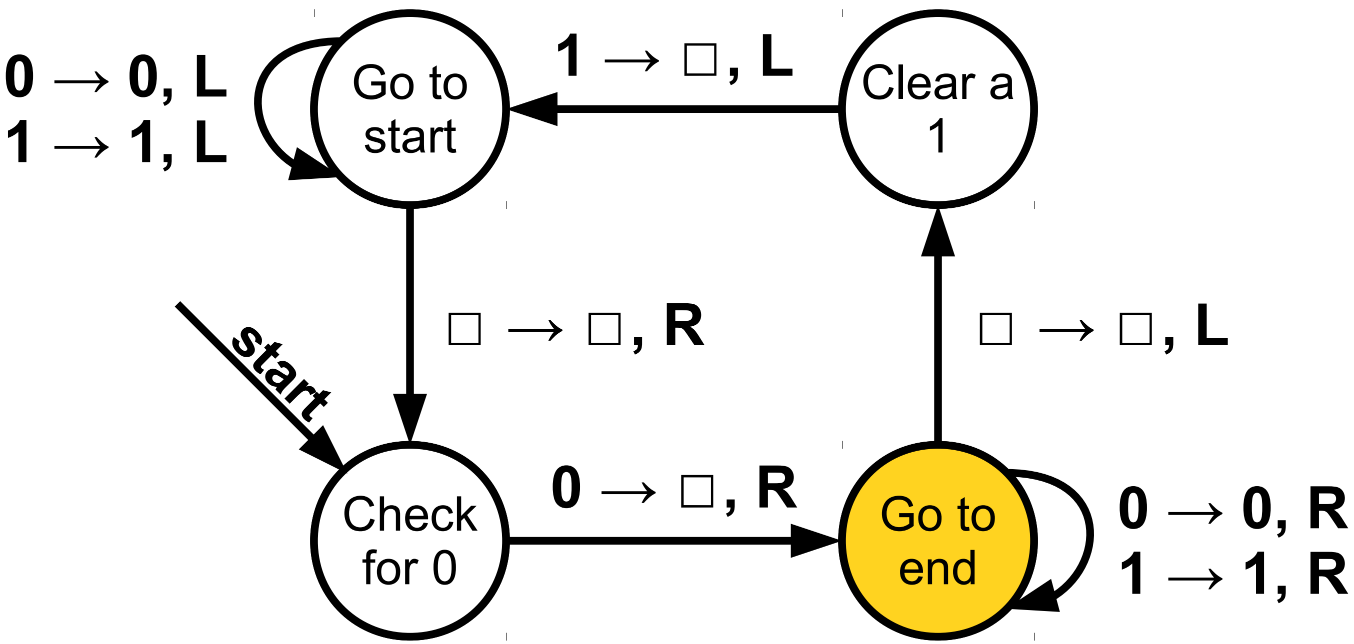
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
1
…

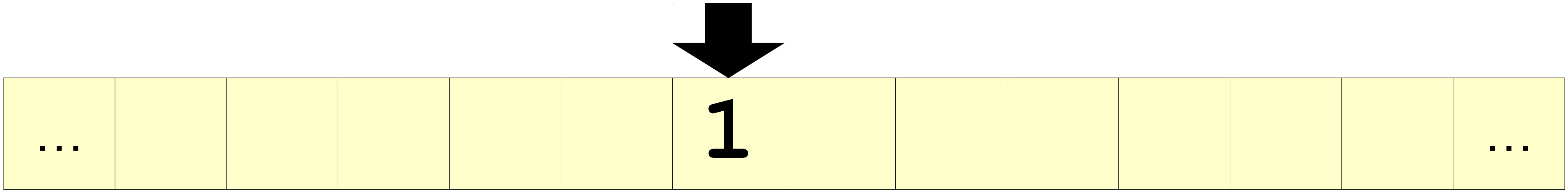


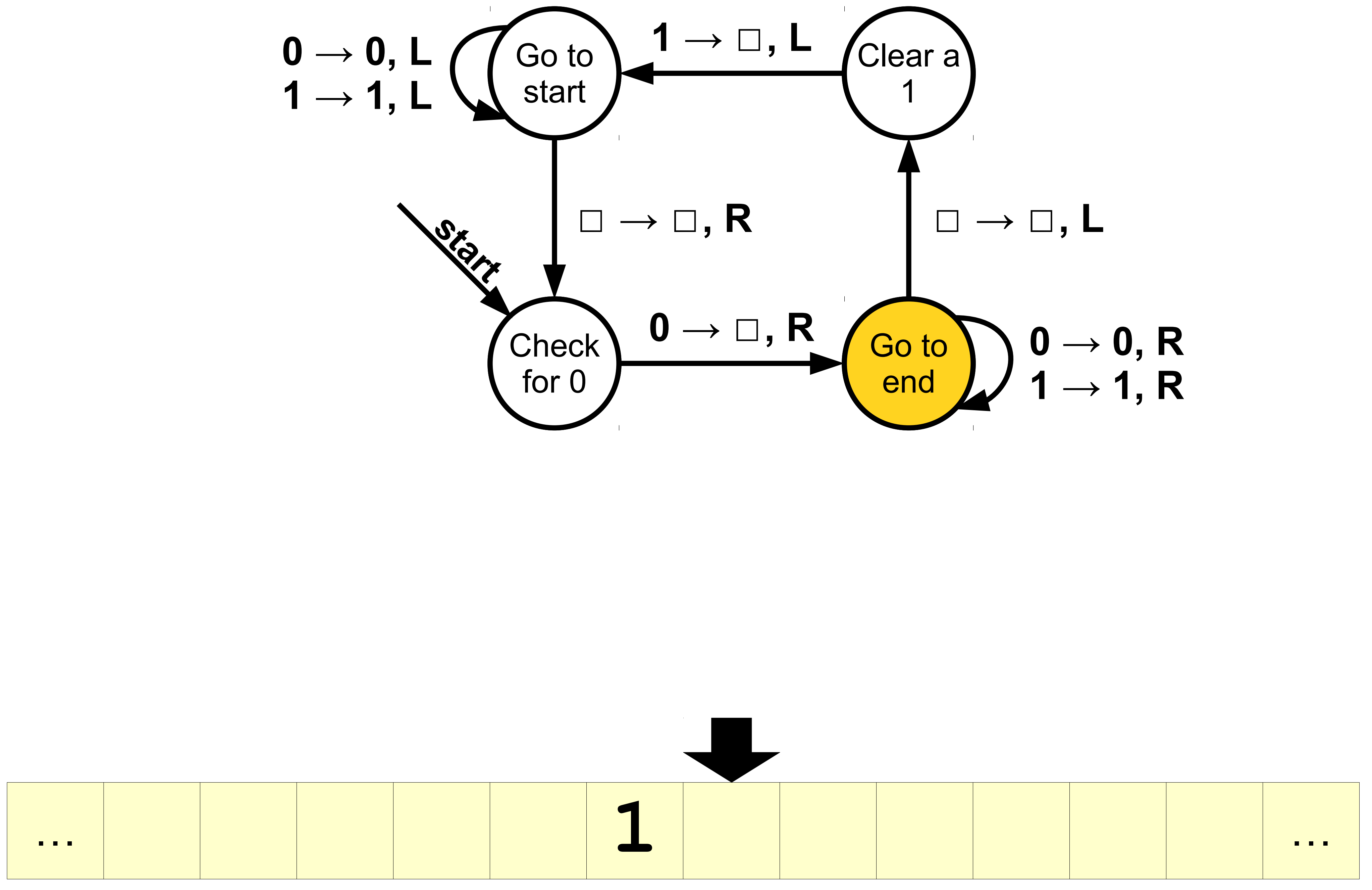
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
0
1
…



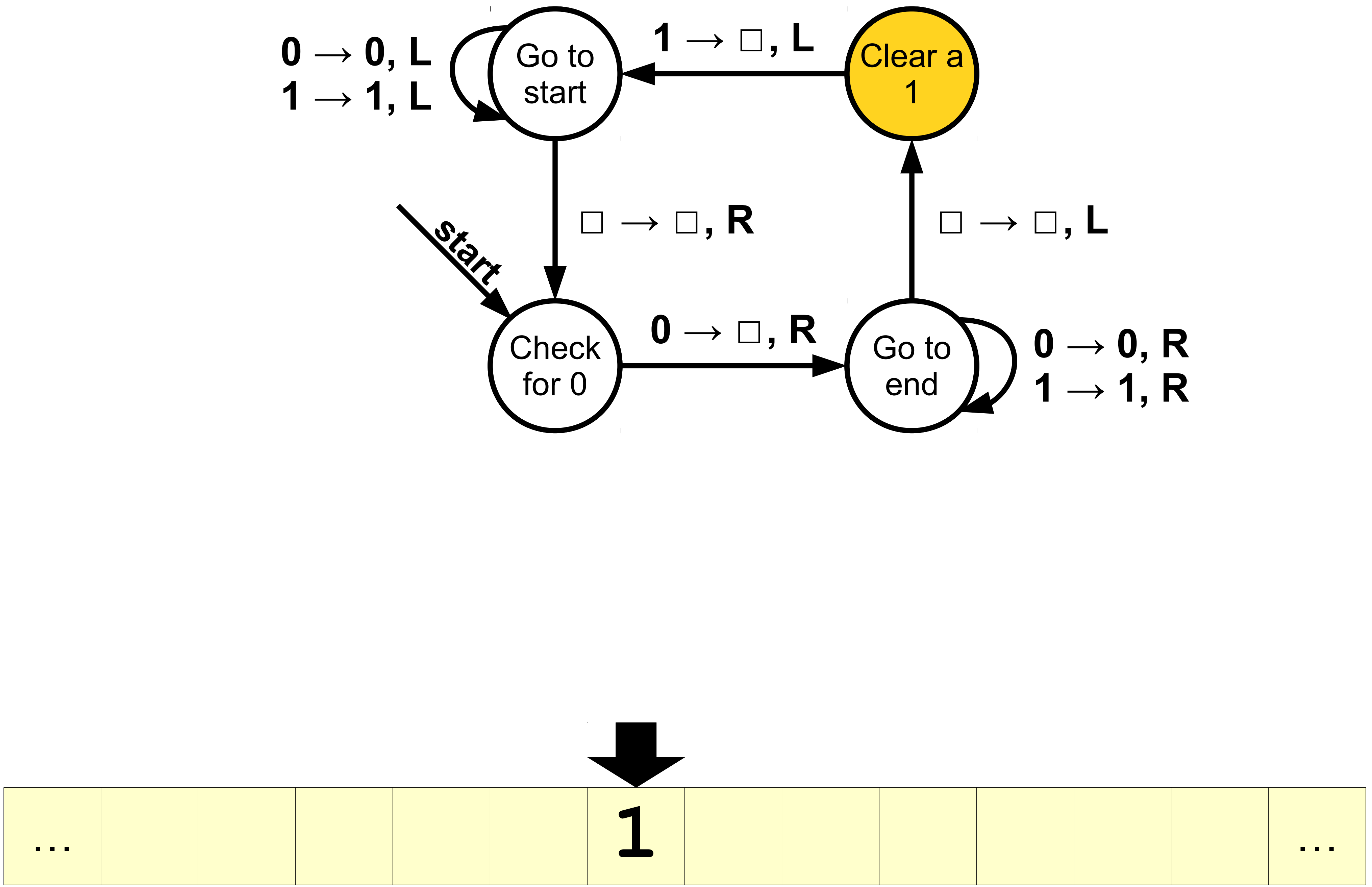
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
…
0
1
…


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R

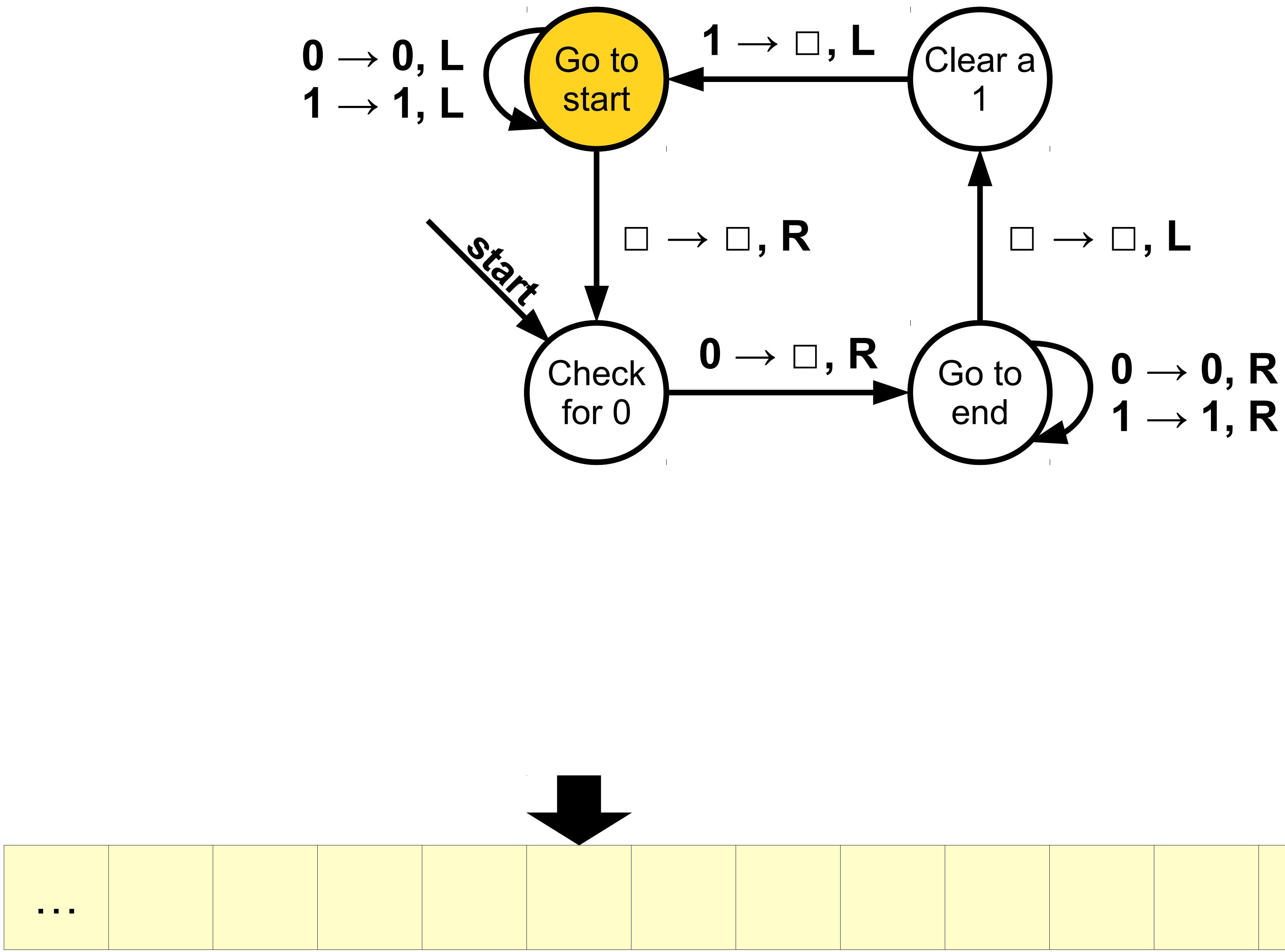

…
1
…



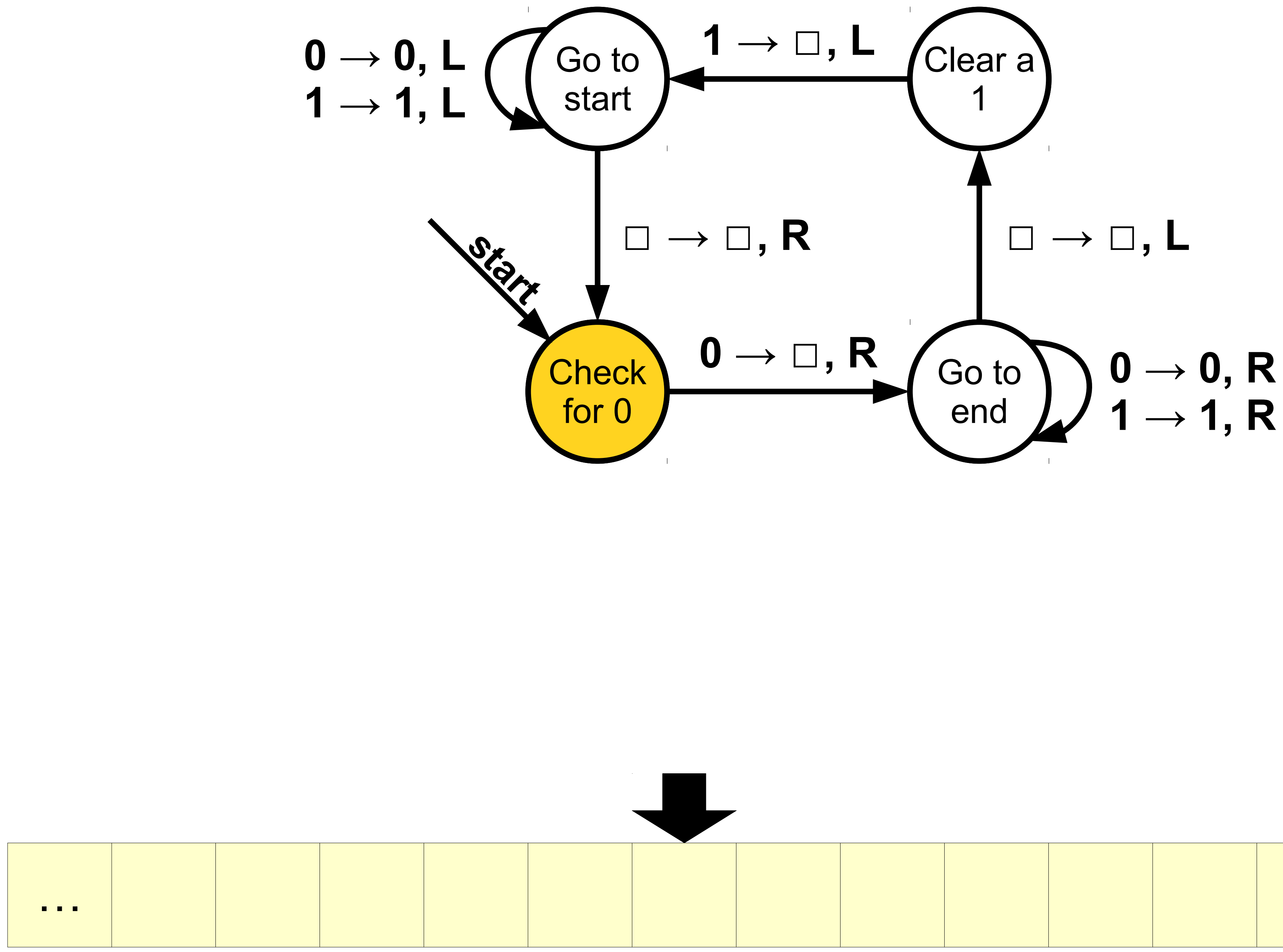
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
1
…



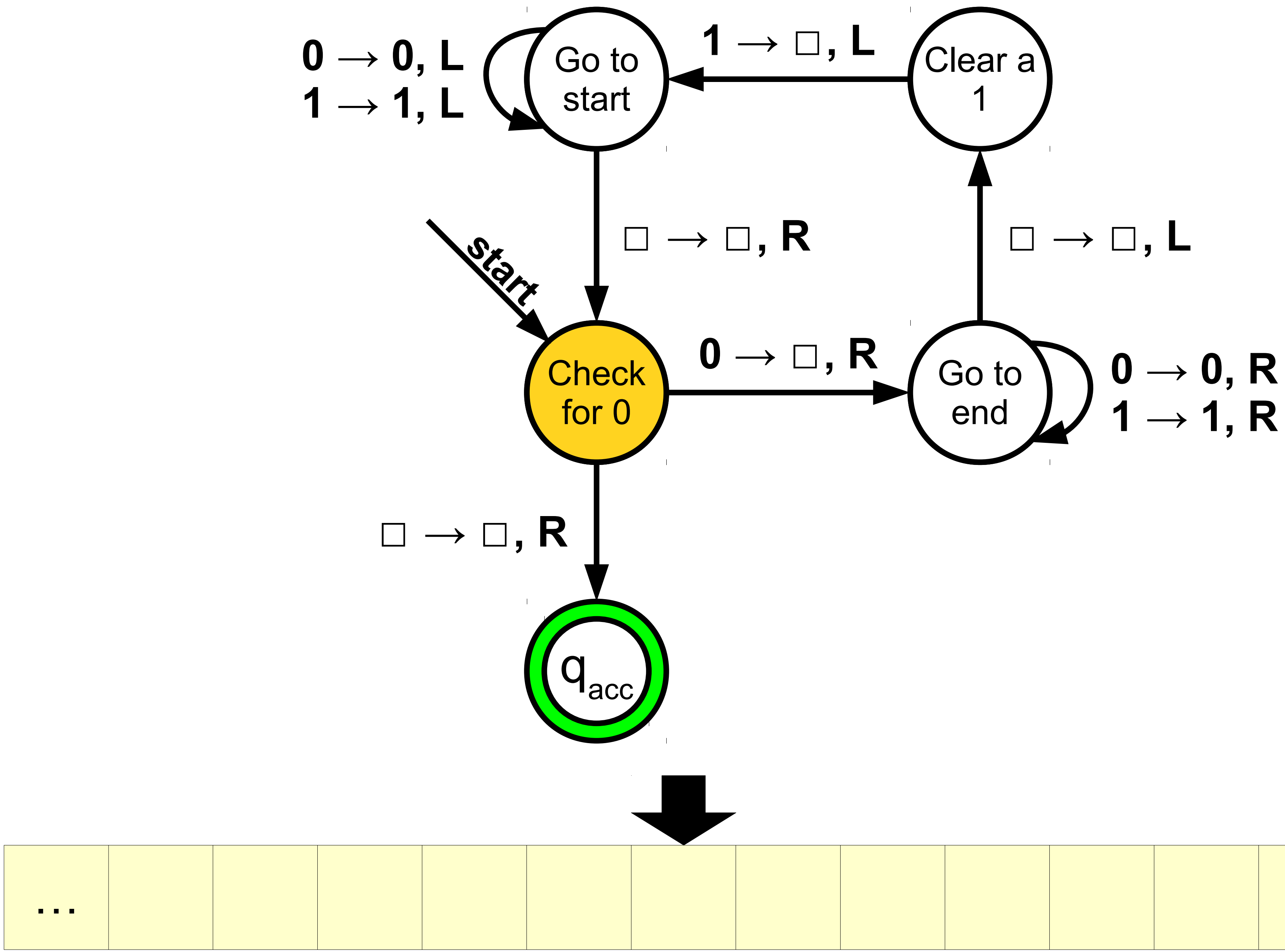
Go to start
Clear a 1
Check for 0
Go to end
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
…
1
…



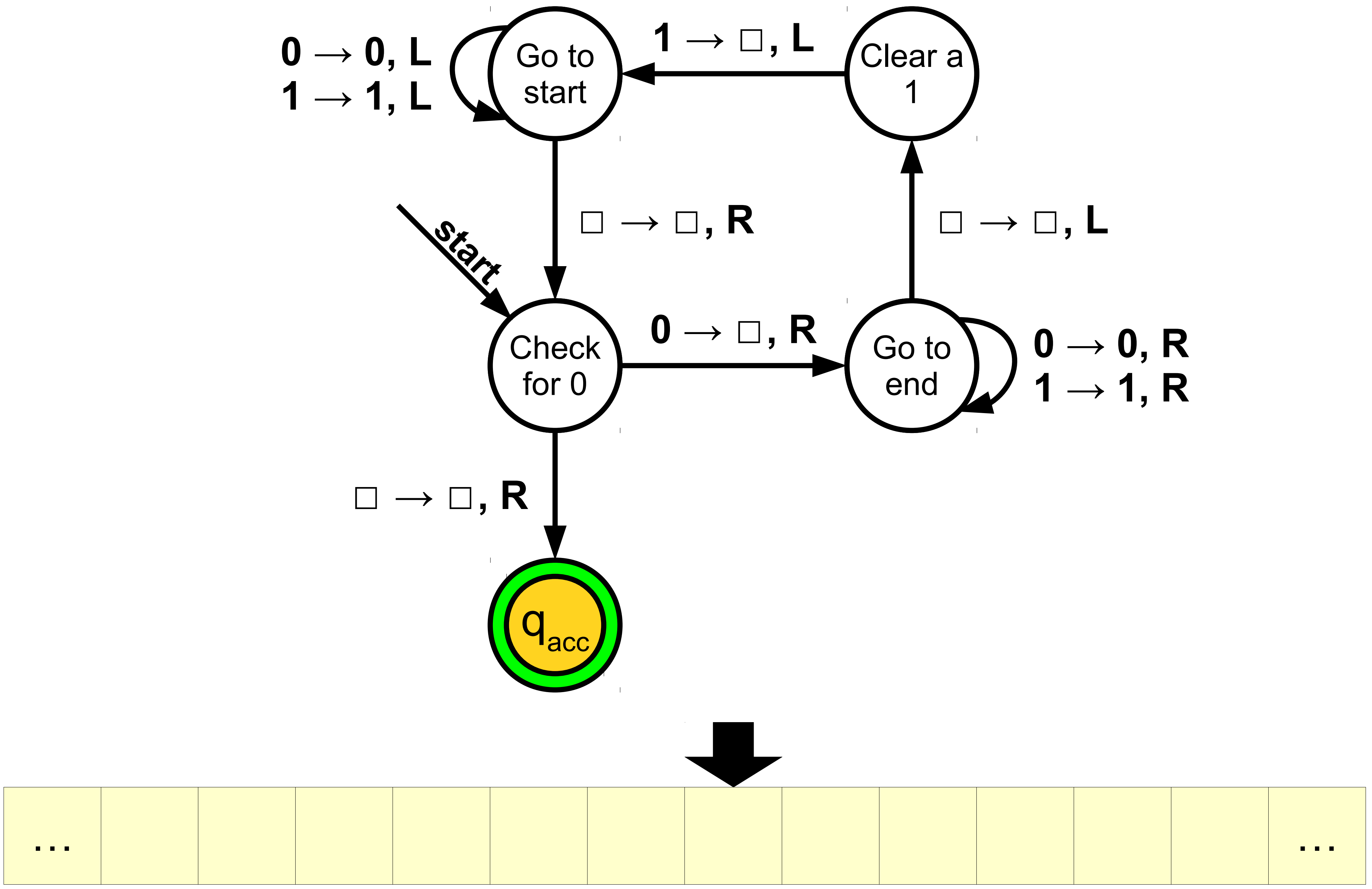
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
...



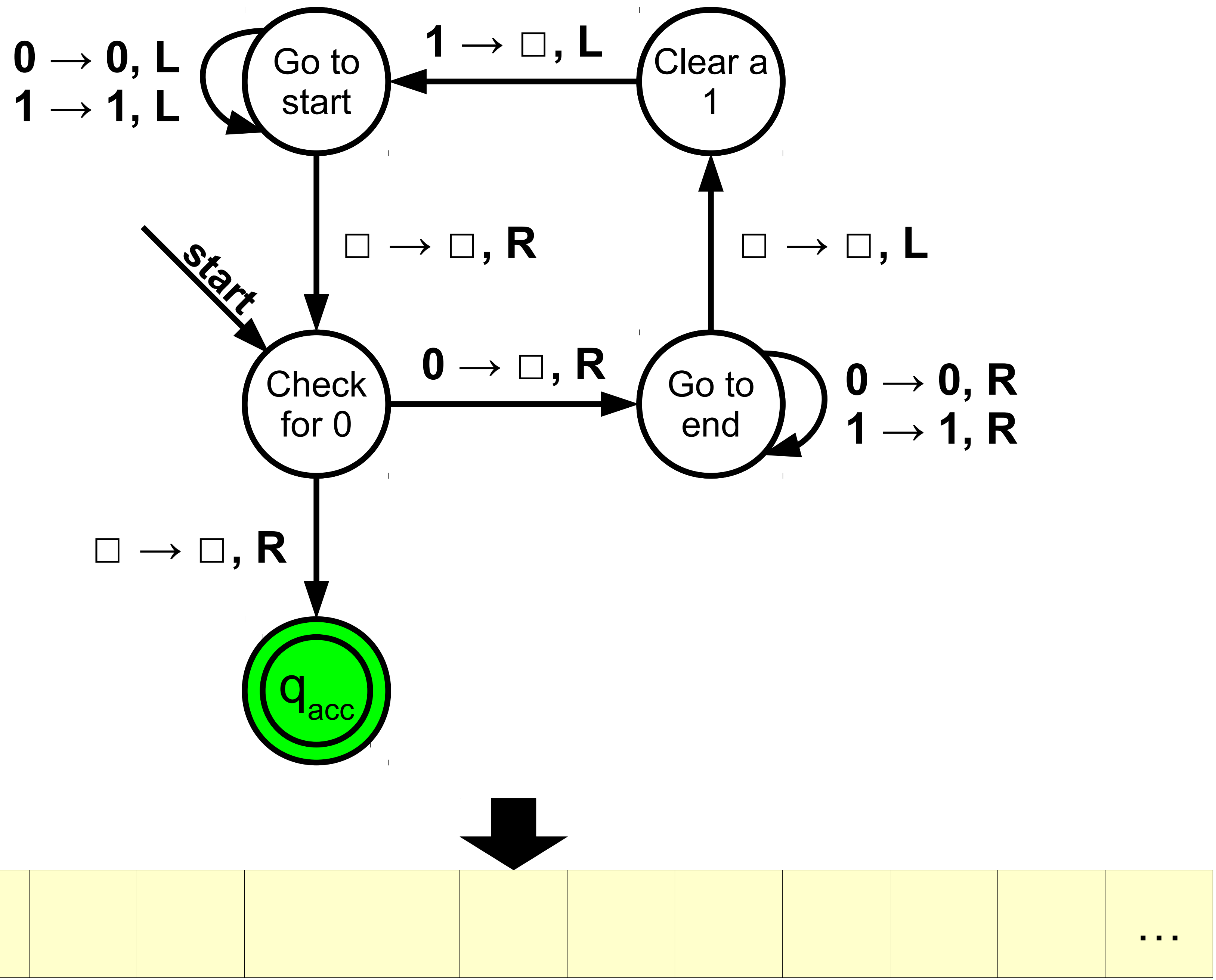
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
...
...

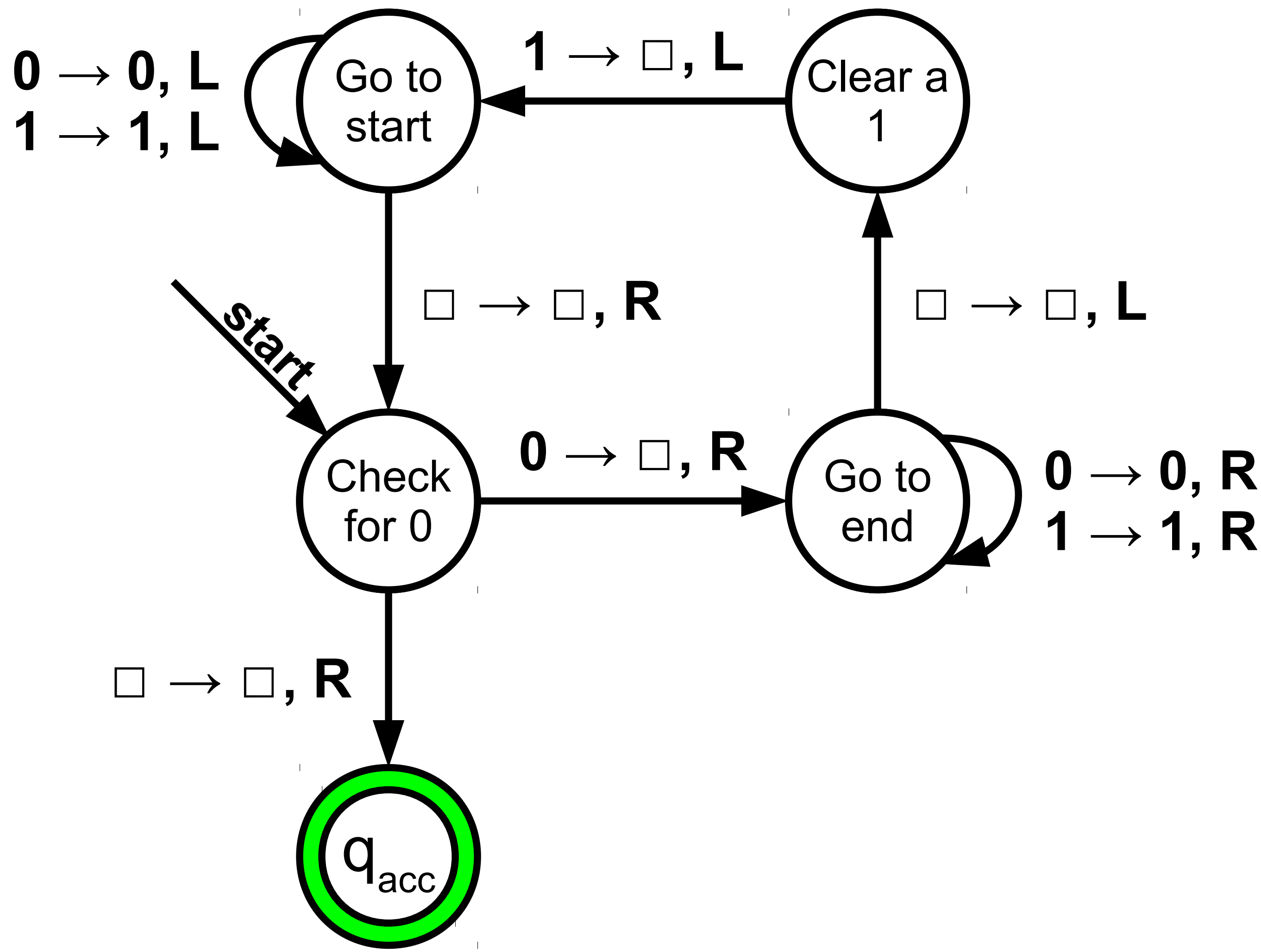


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
...
...

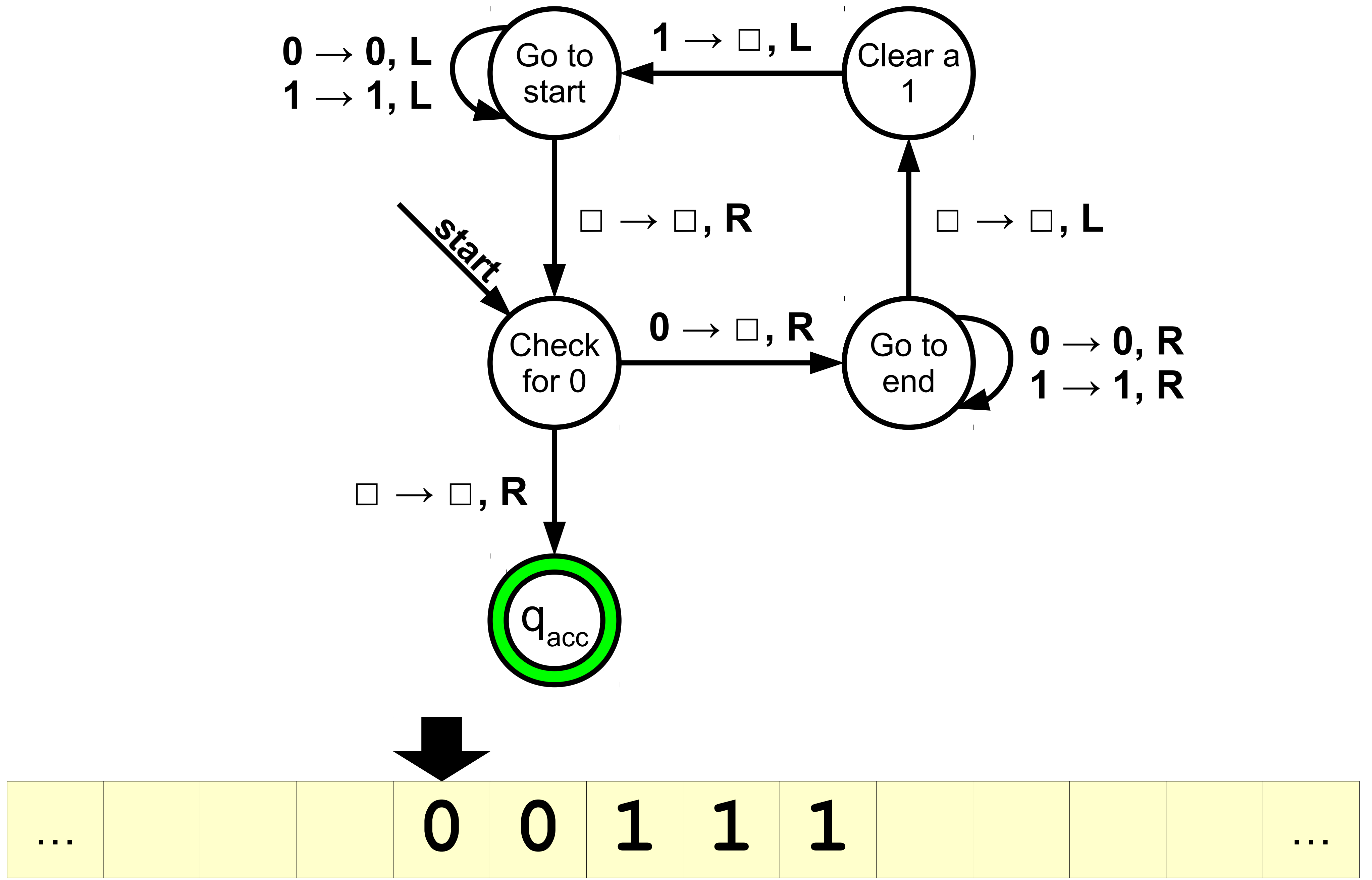


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
...
...

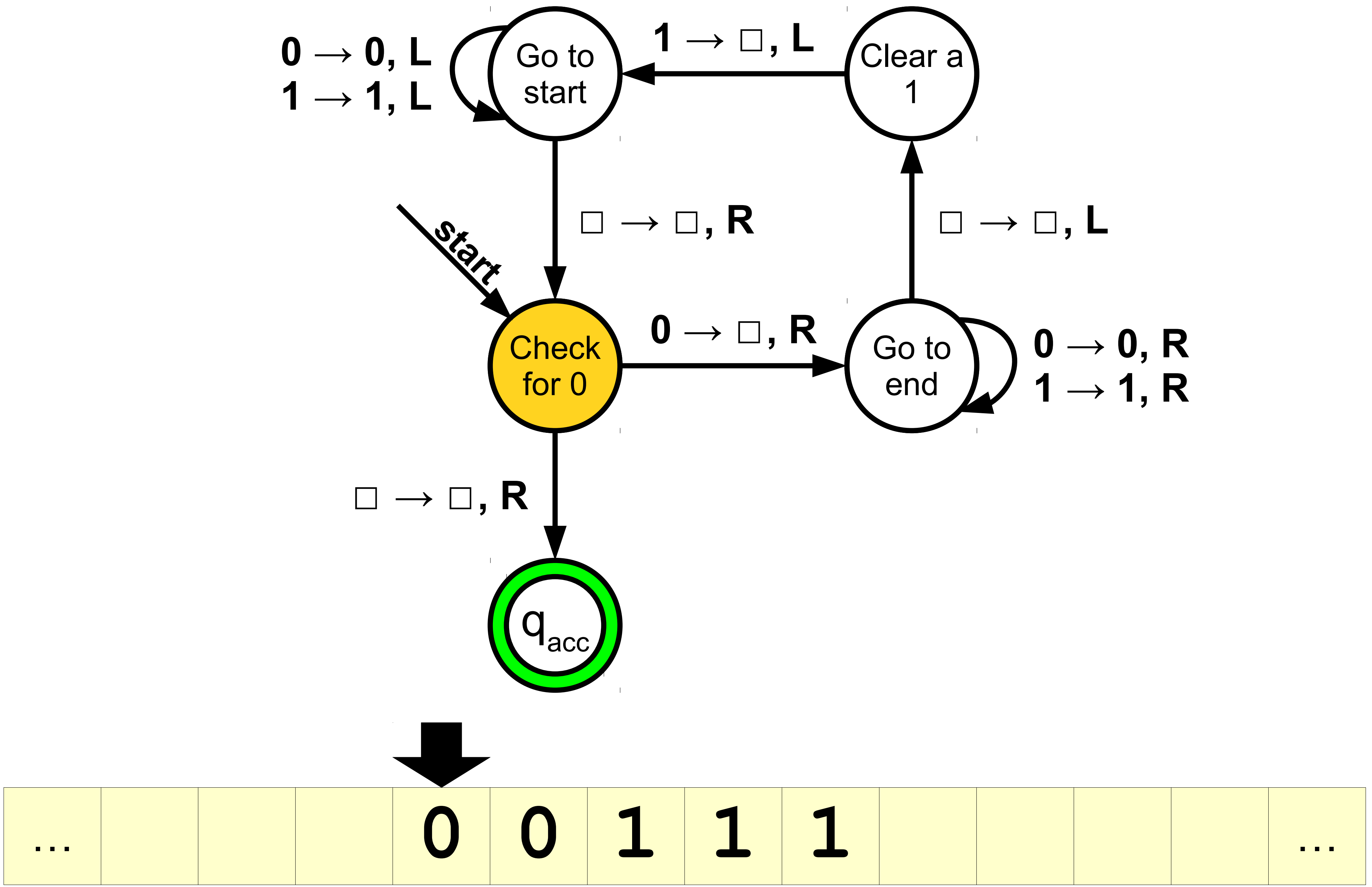

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
...
...



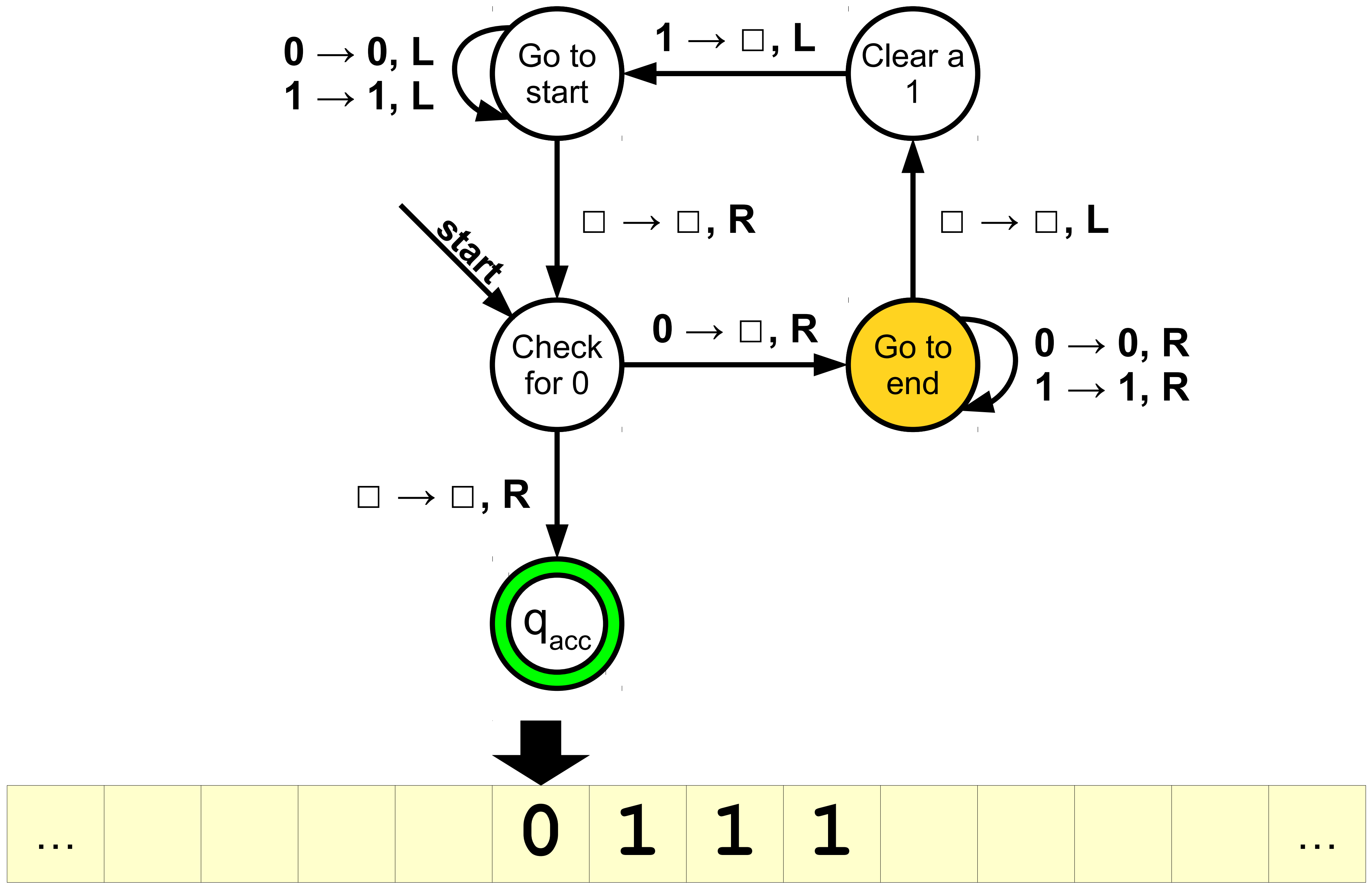
Go to start
Clear a 1
Check for 0
Go to end
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R



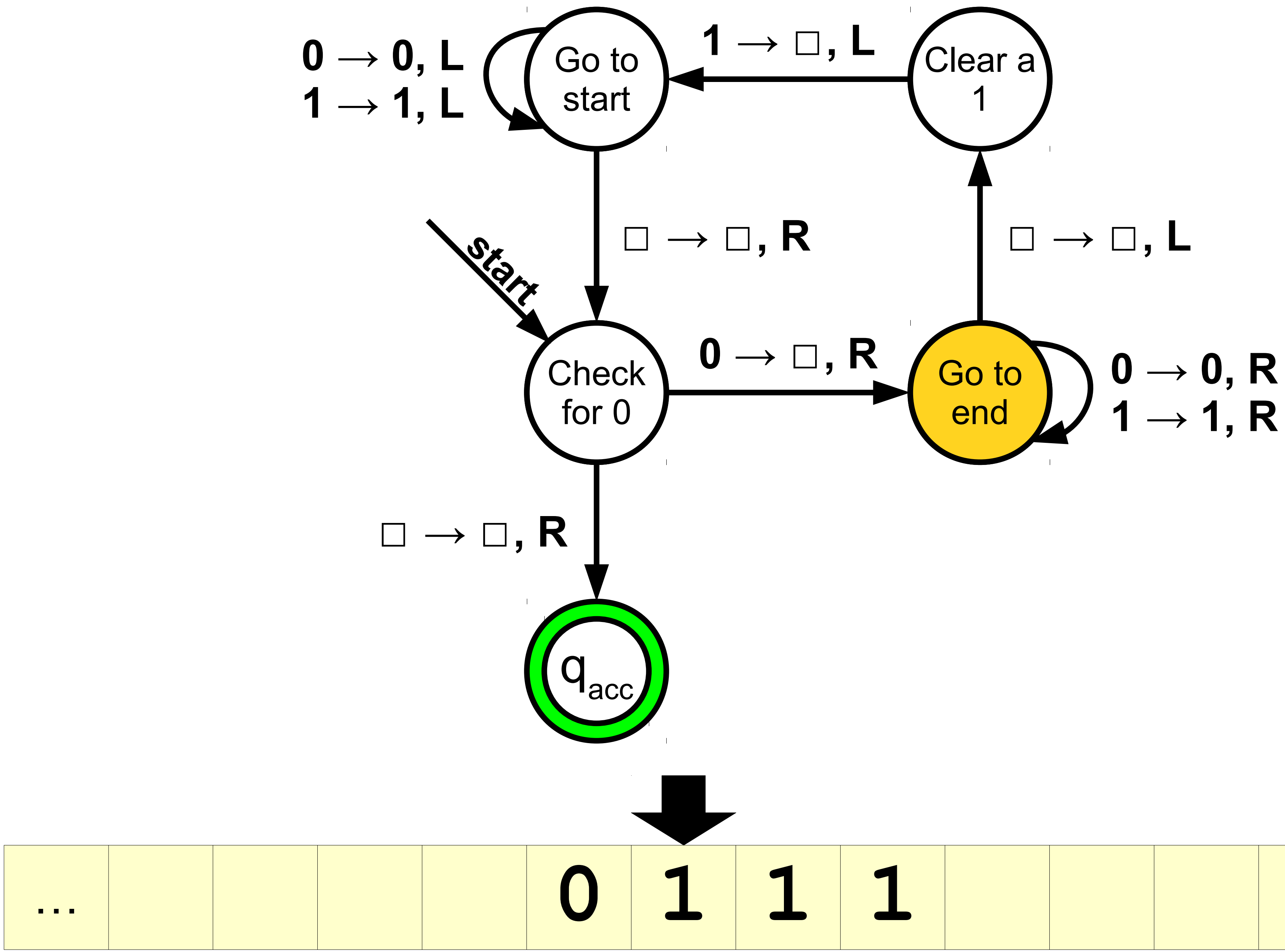
Go to start
Clear a 1
Check for 0
Go to end
q_acc
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
0
0
1
1
1
…



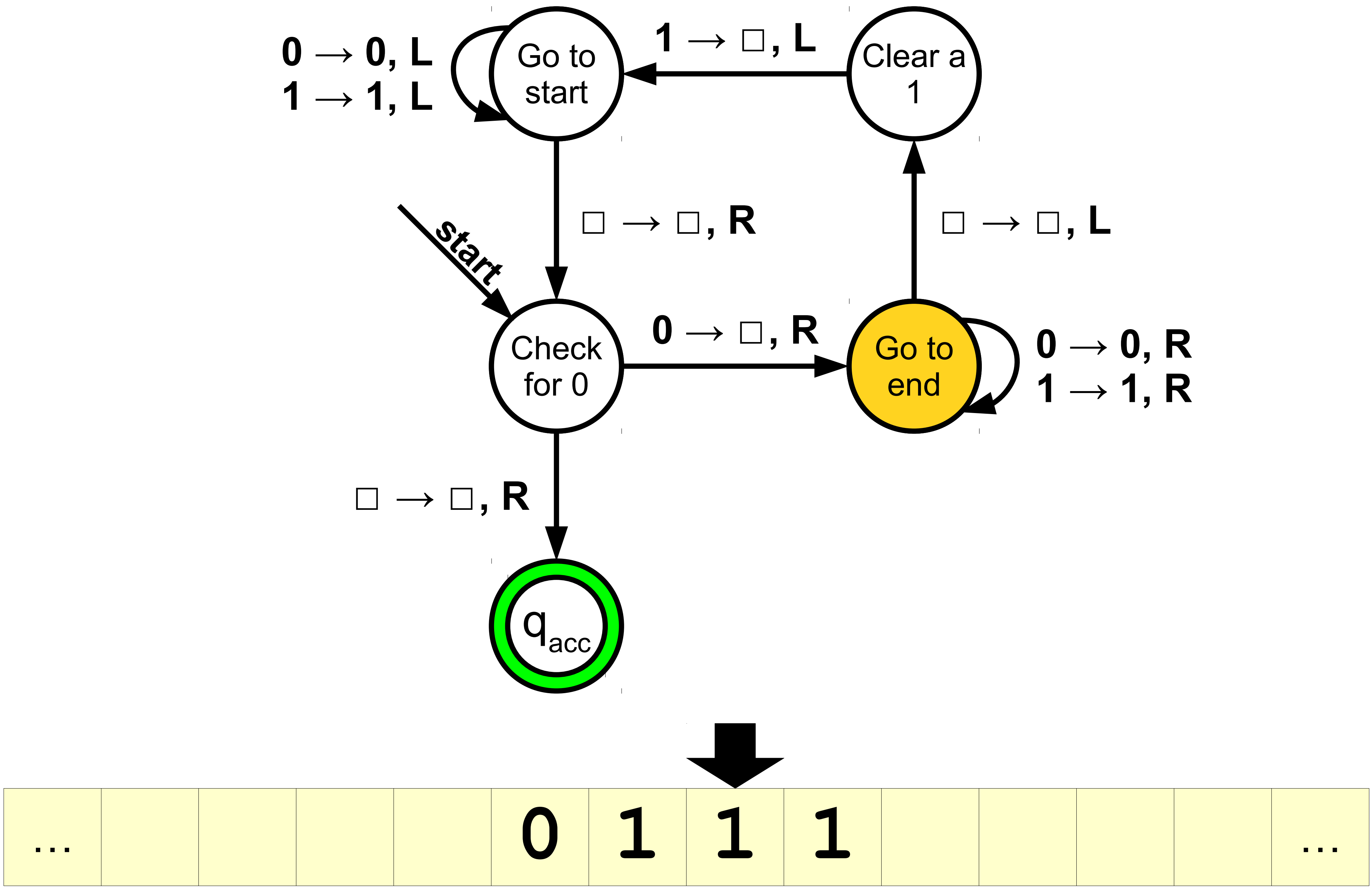
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q$_{acc}$
… 0 0 1 1 1 …

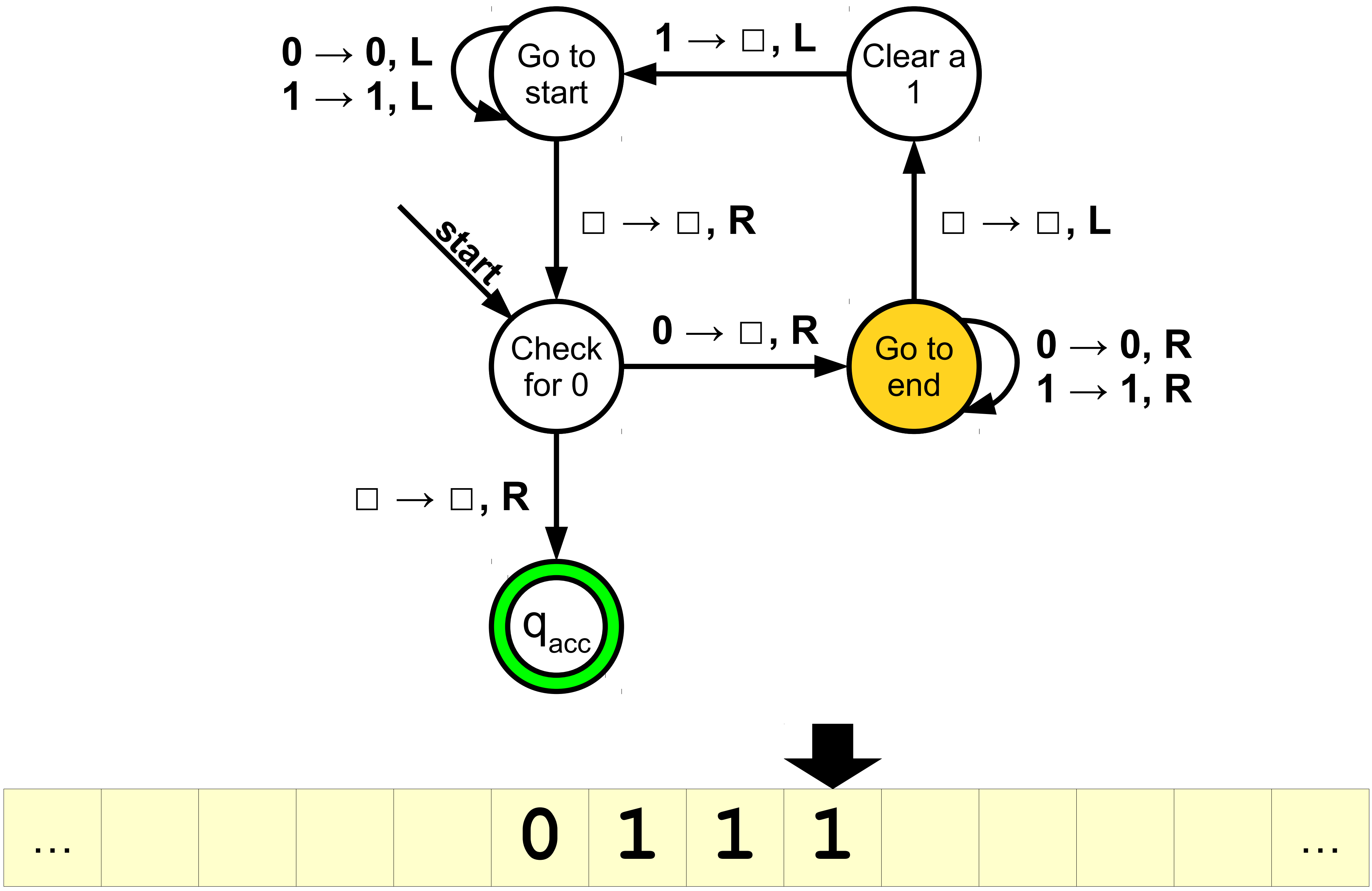


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
1
1
…

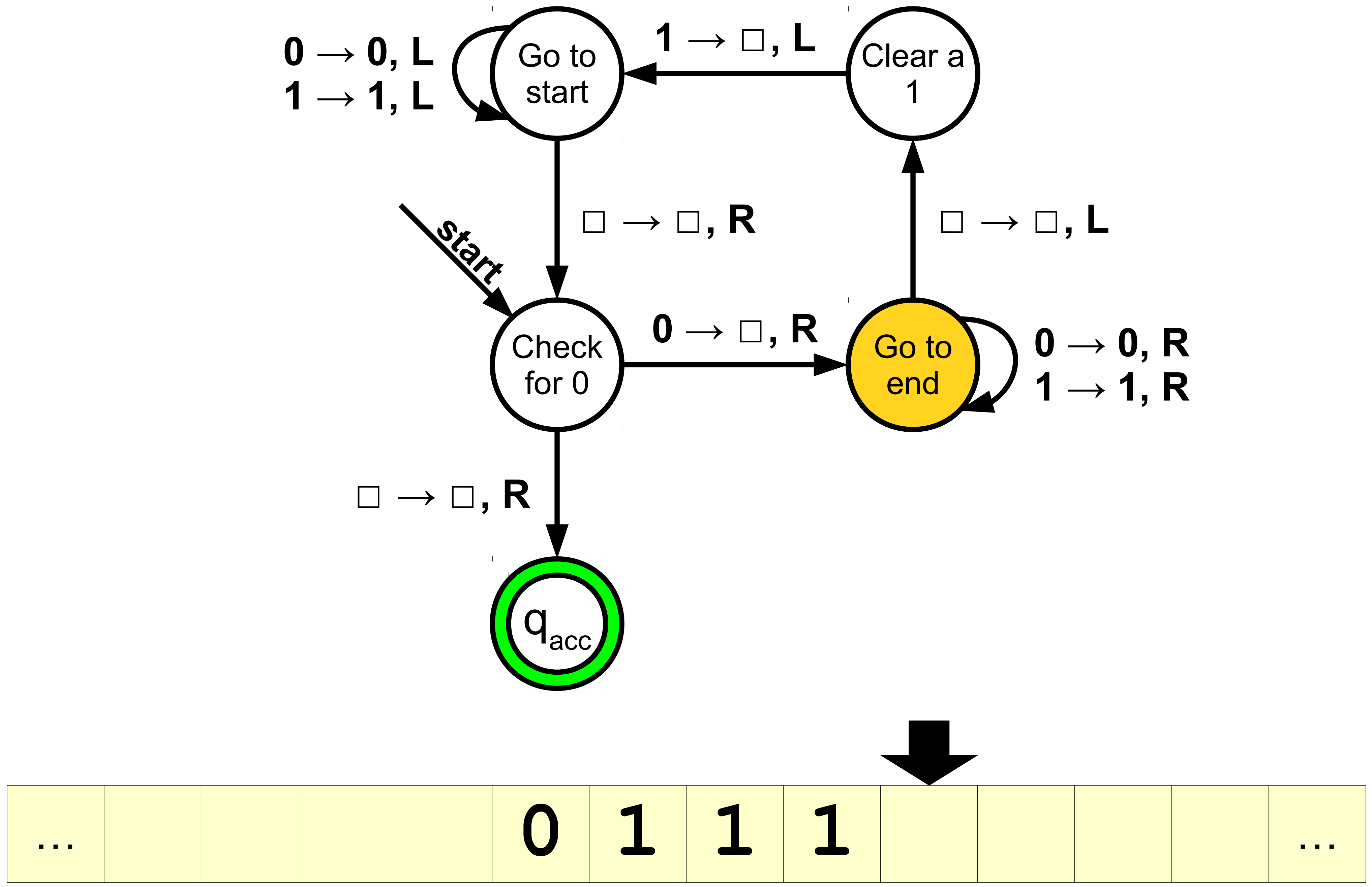


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
1
1
…

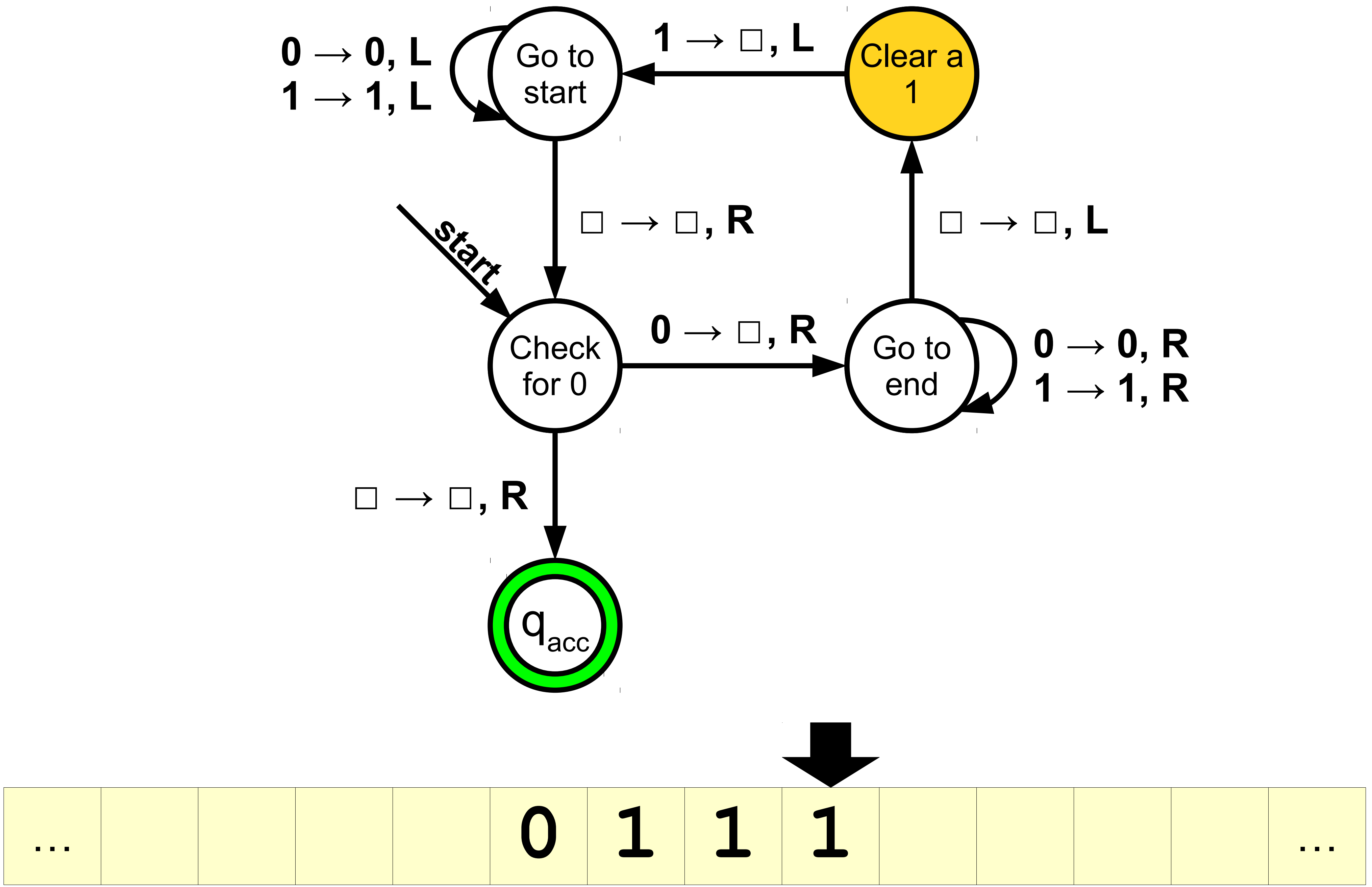

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
1
1
1
…

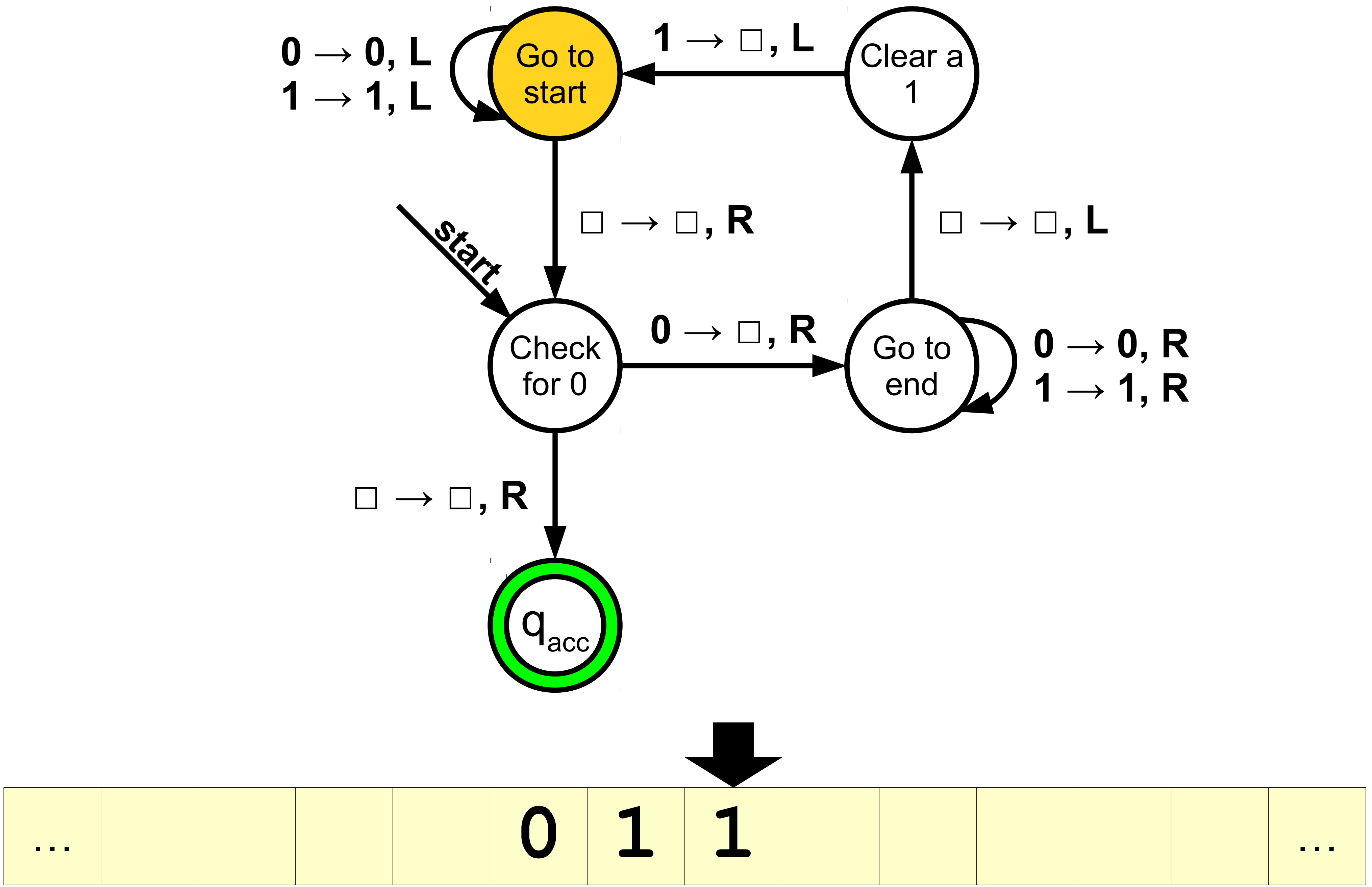


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
1
1
1
…

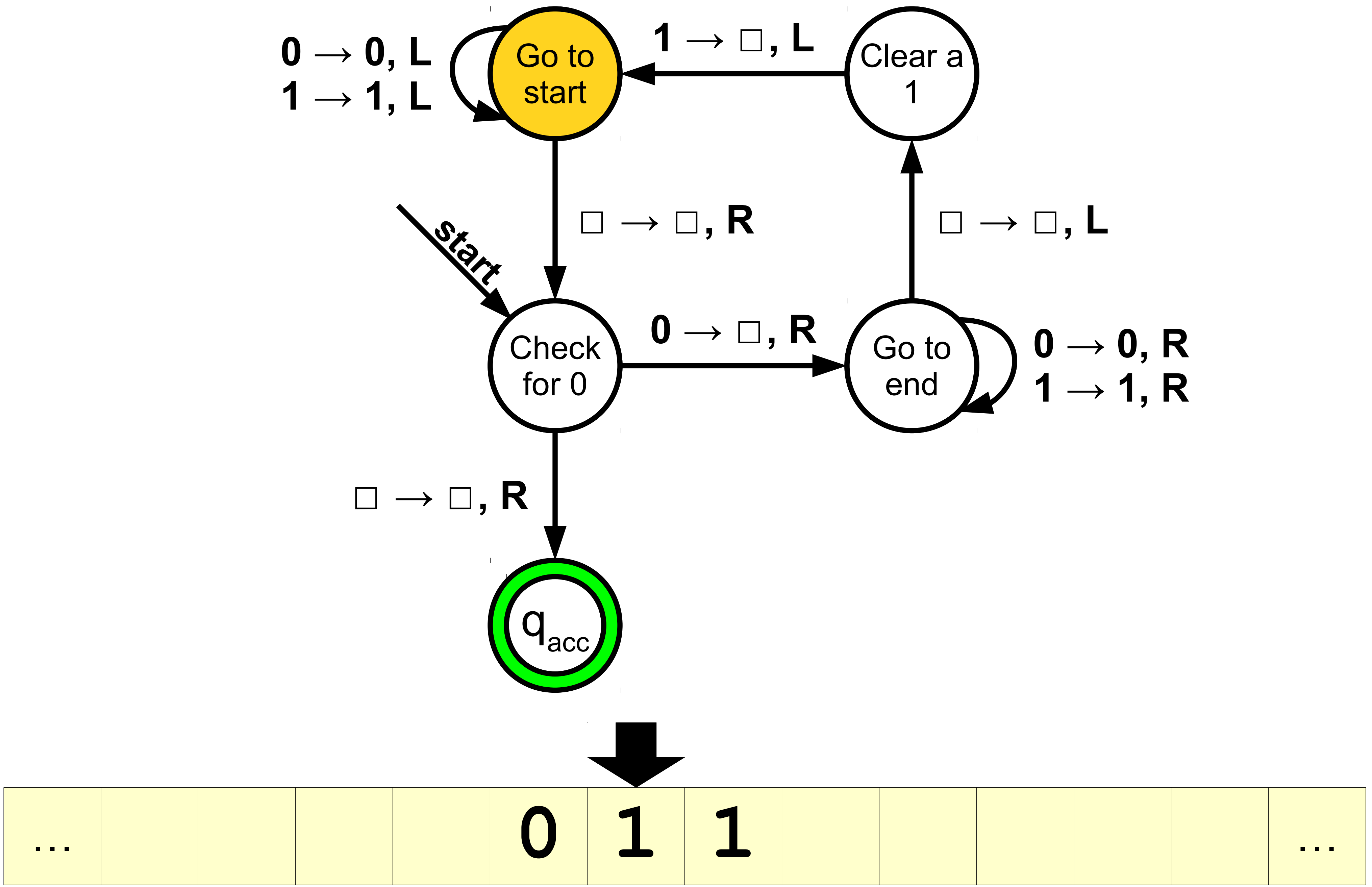


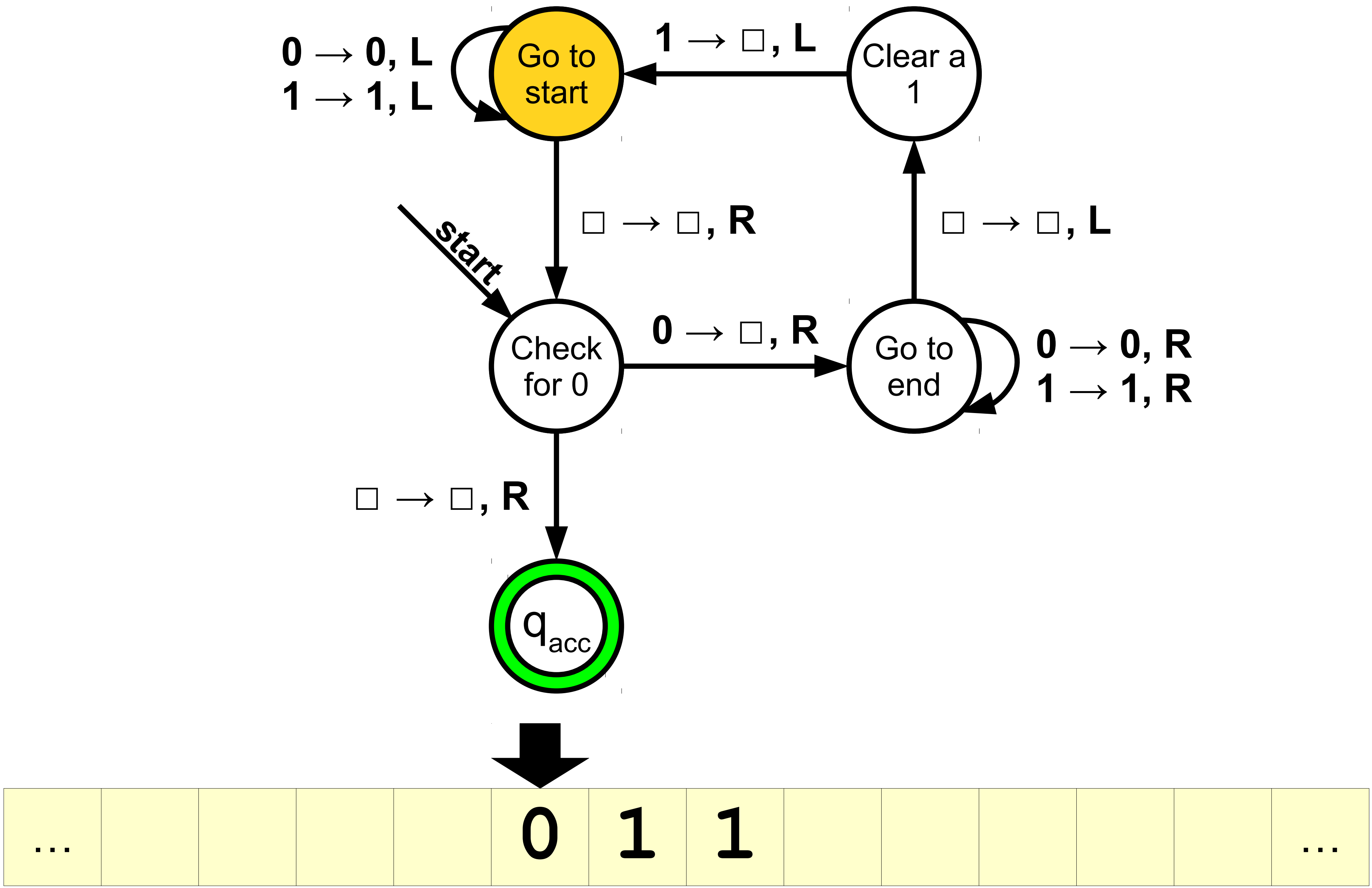
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
0 → □, R
Check for 0
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
1
1
…

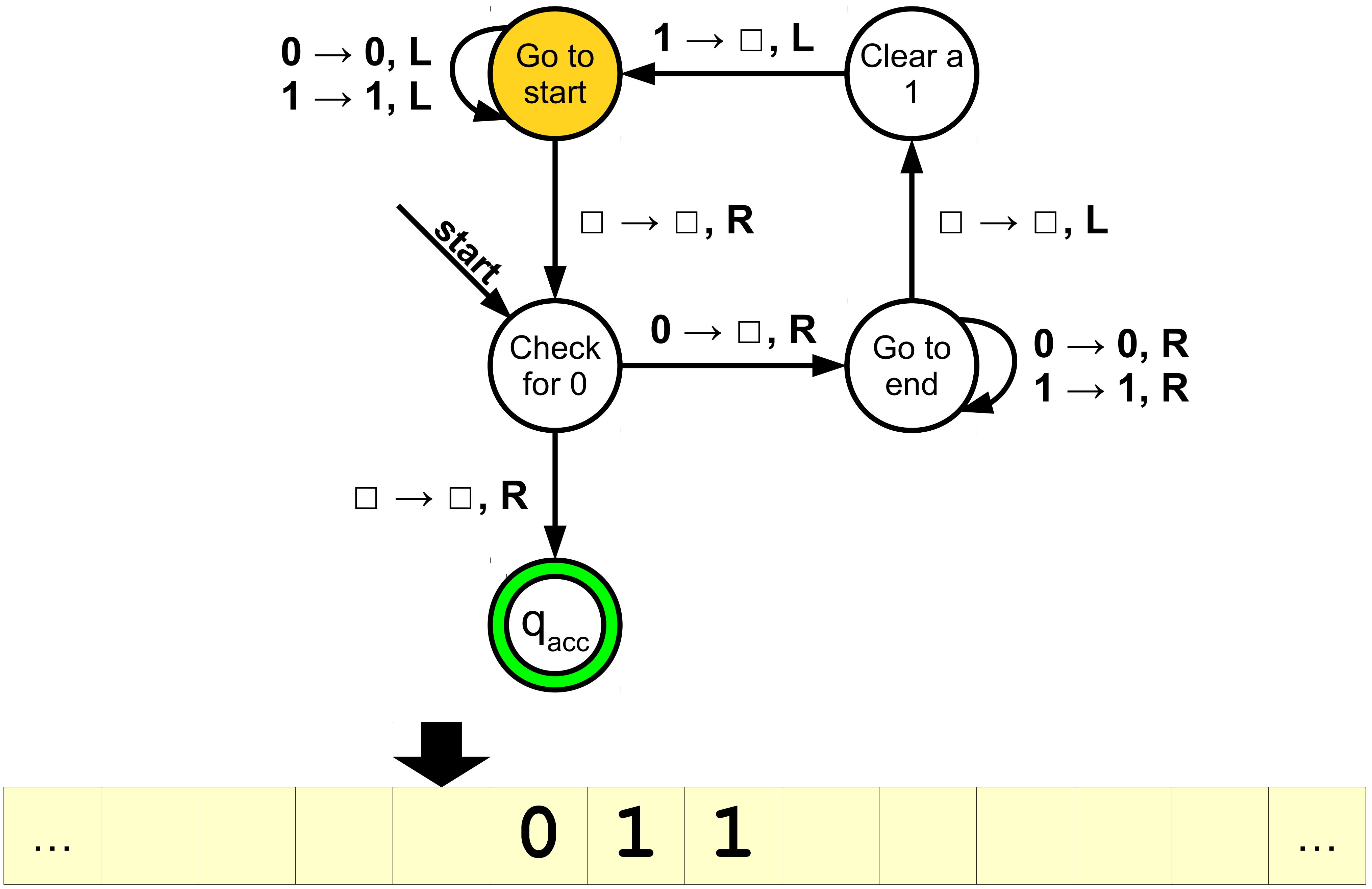

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
1
1
…



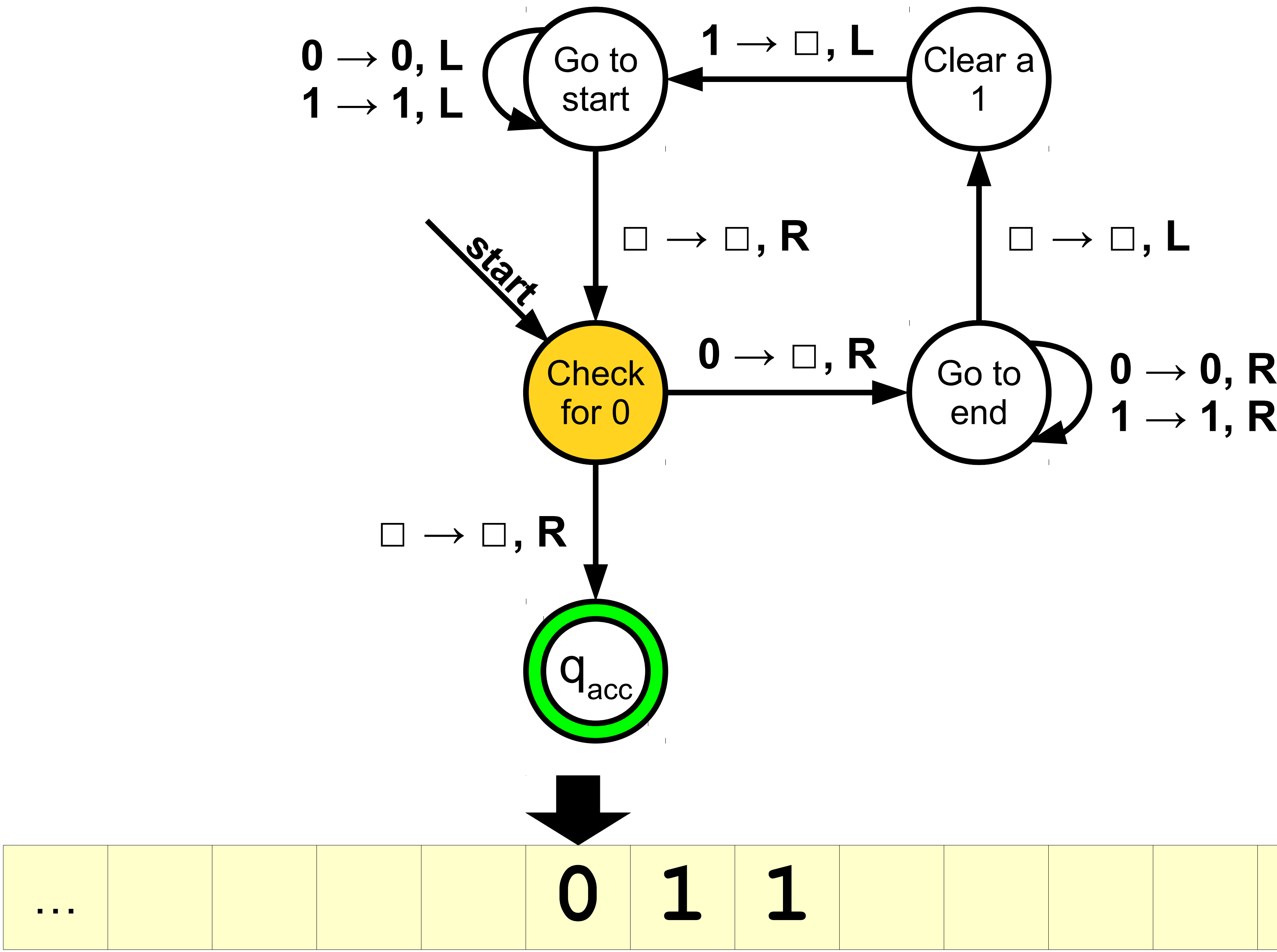
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
1
1
…

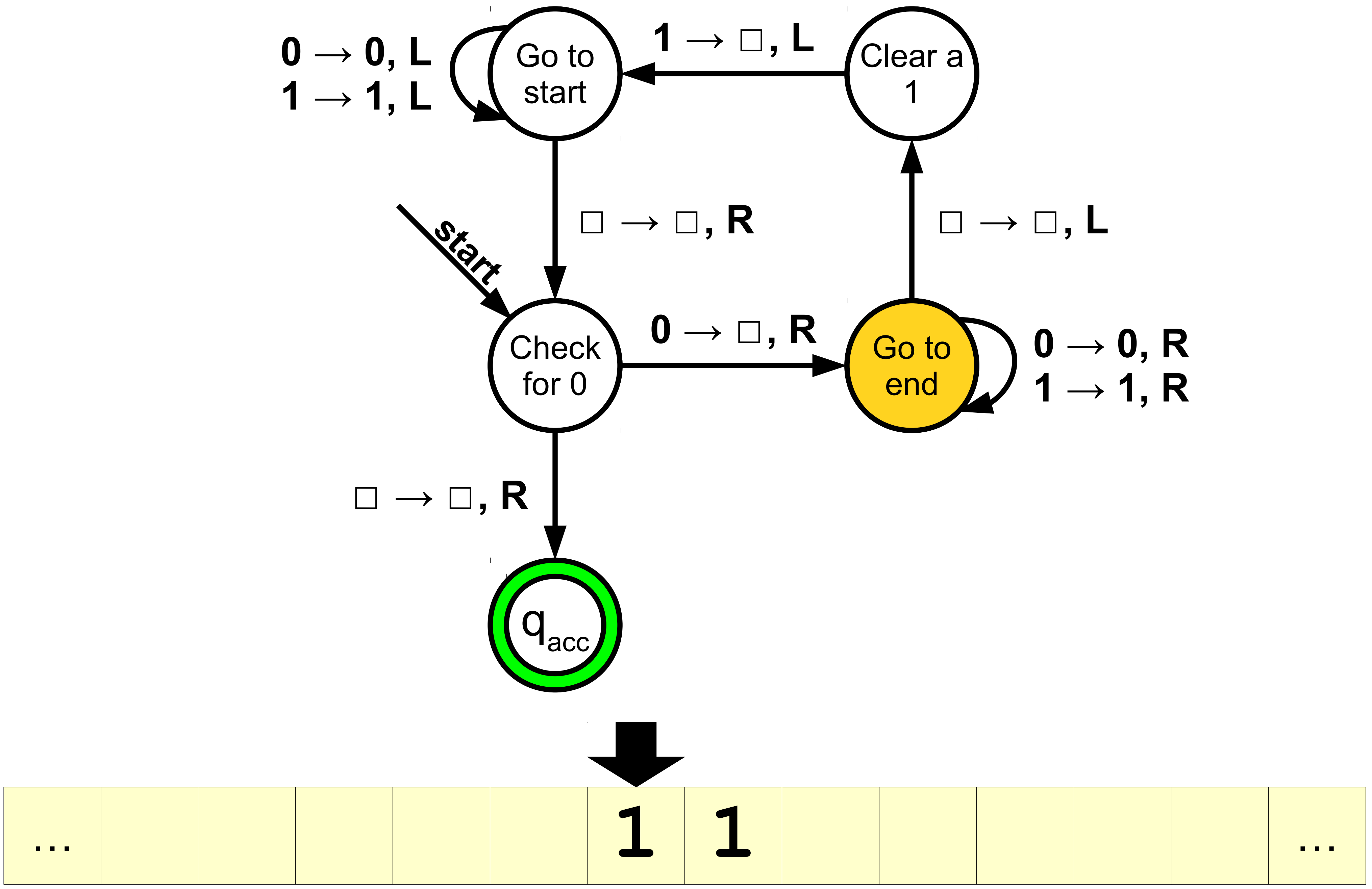

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
...
0
1
1
...


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
1
1
…

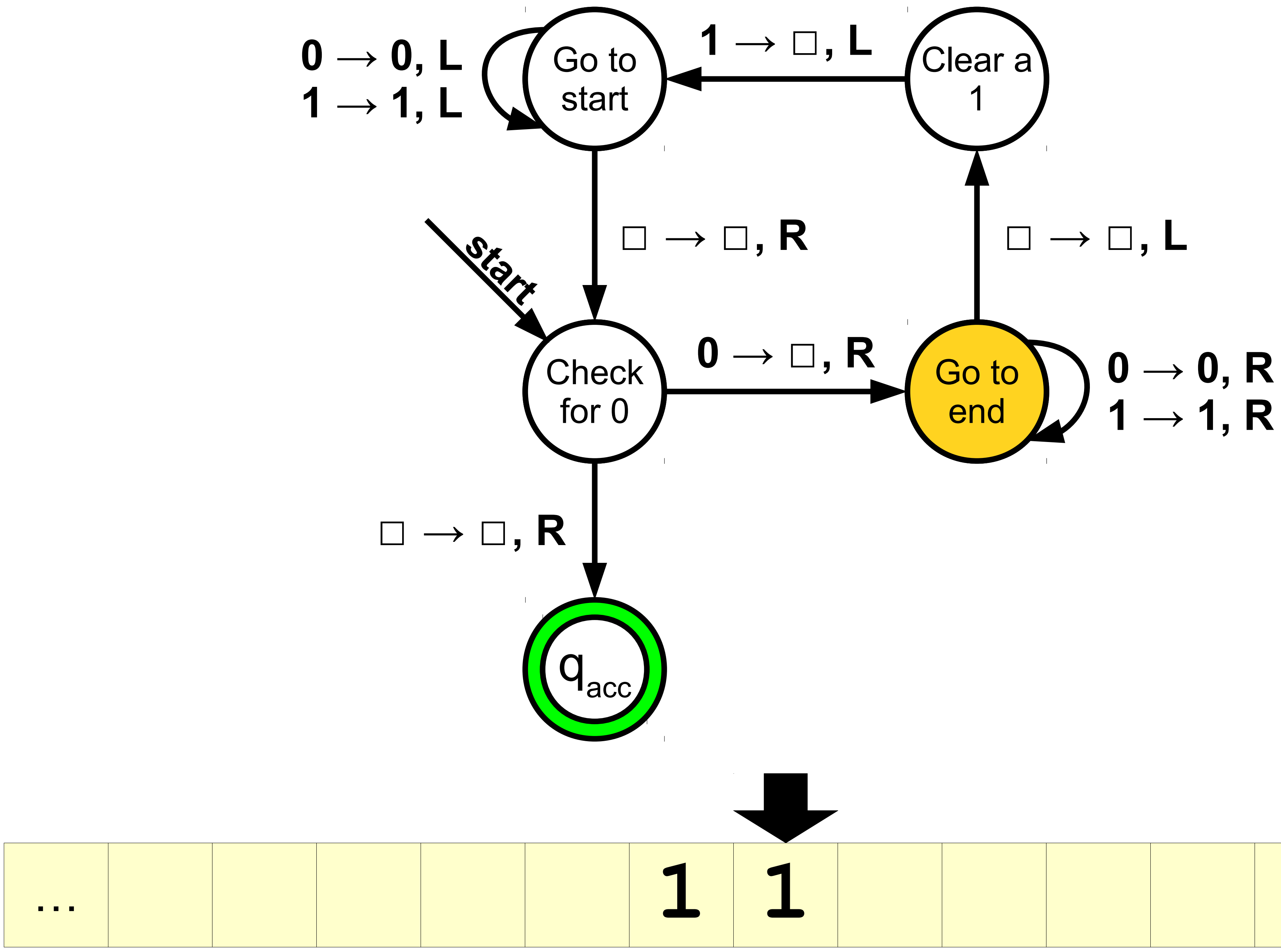


Go to start
Clear a 1
Check for 0
Go to end
q_acc
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
0
1
1
…

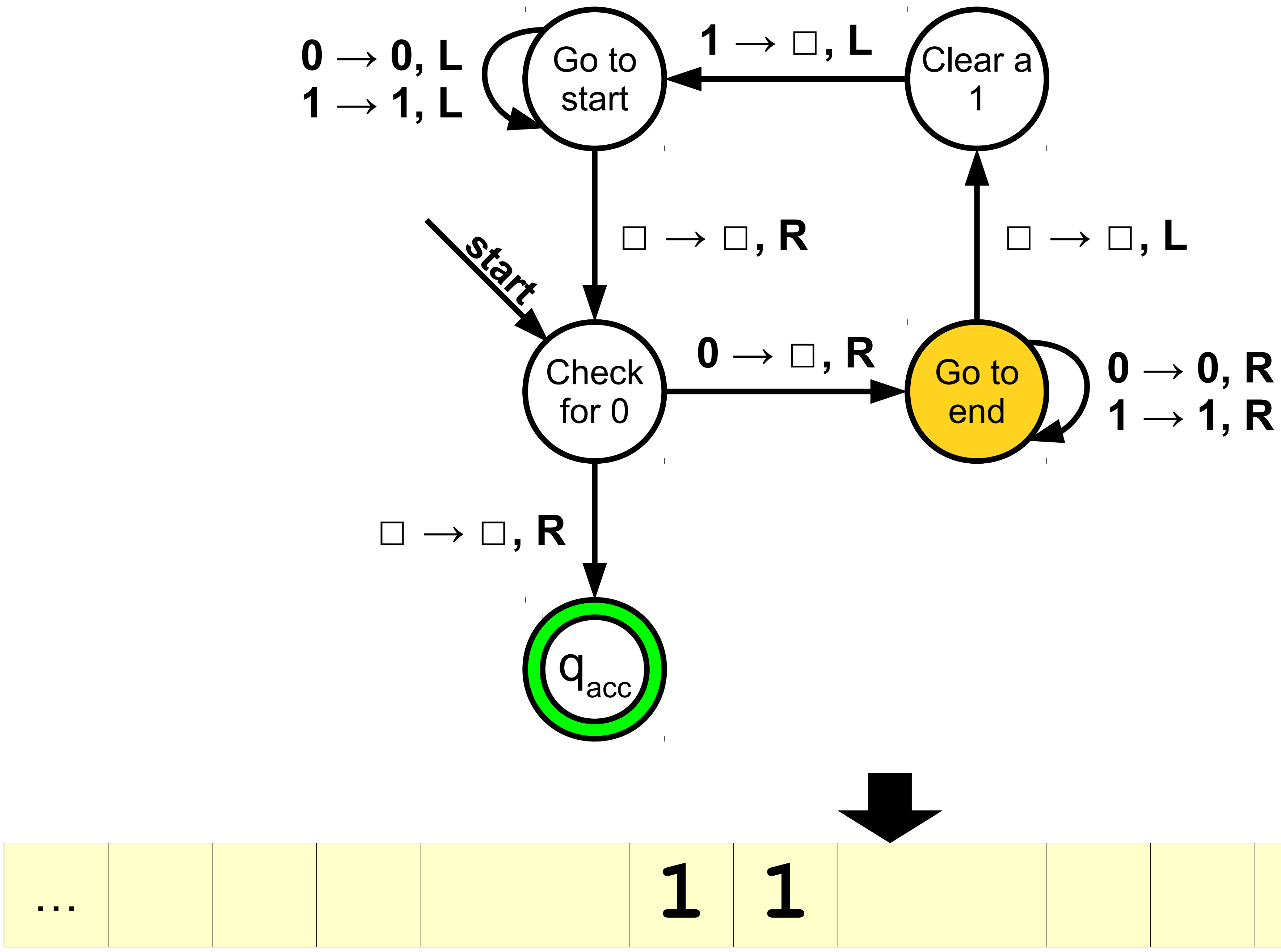

Go to start
Clear a 1
1 → □, L
0 → 0, L
1 → 1, L
start
□ → □, R
□ → □, L
Check for 0
Go to end
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
1
1
…



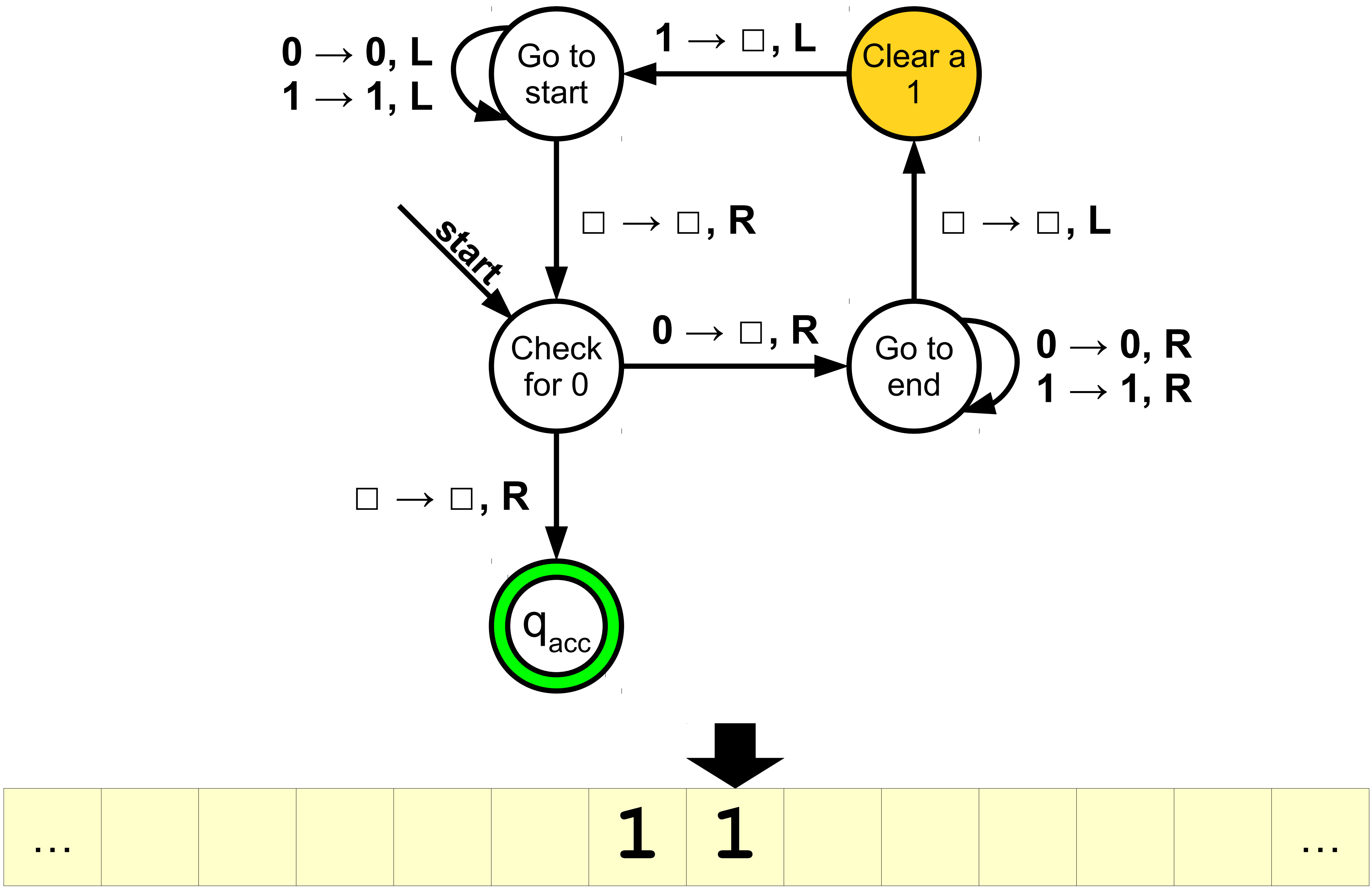
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1 1
…

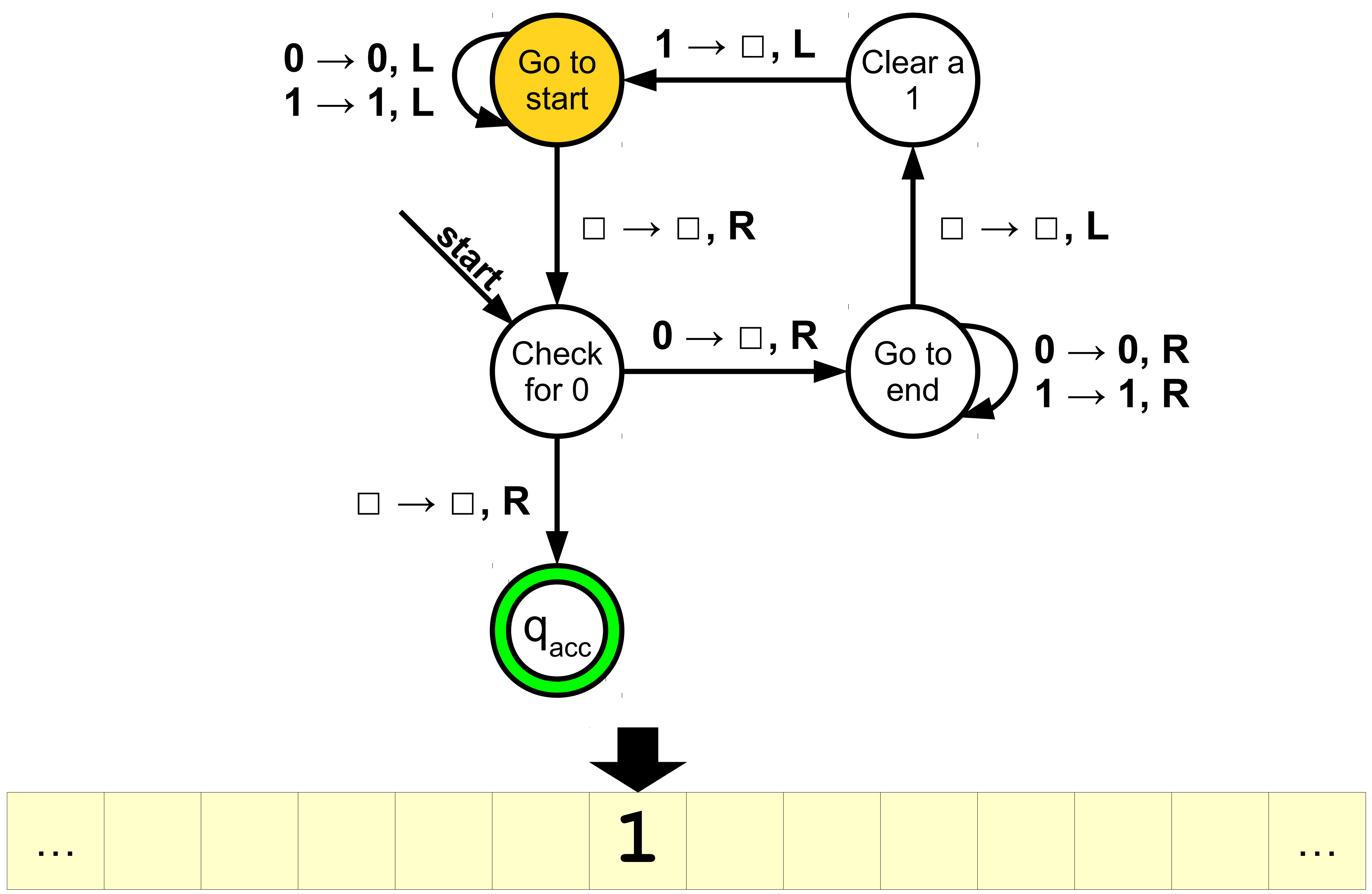


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
1
…

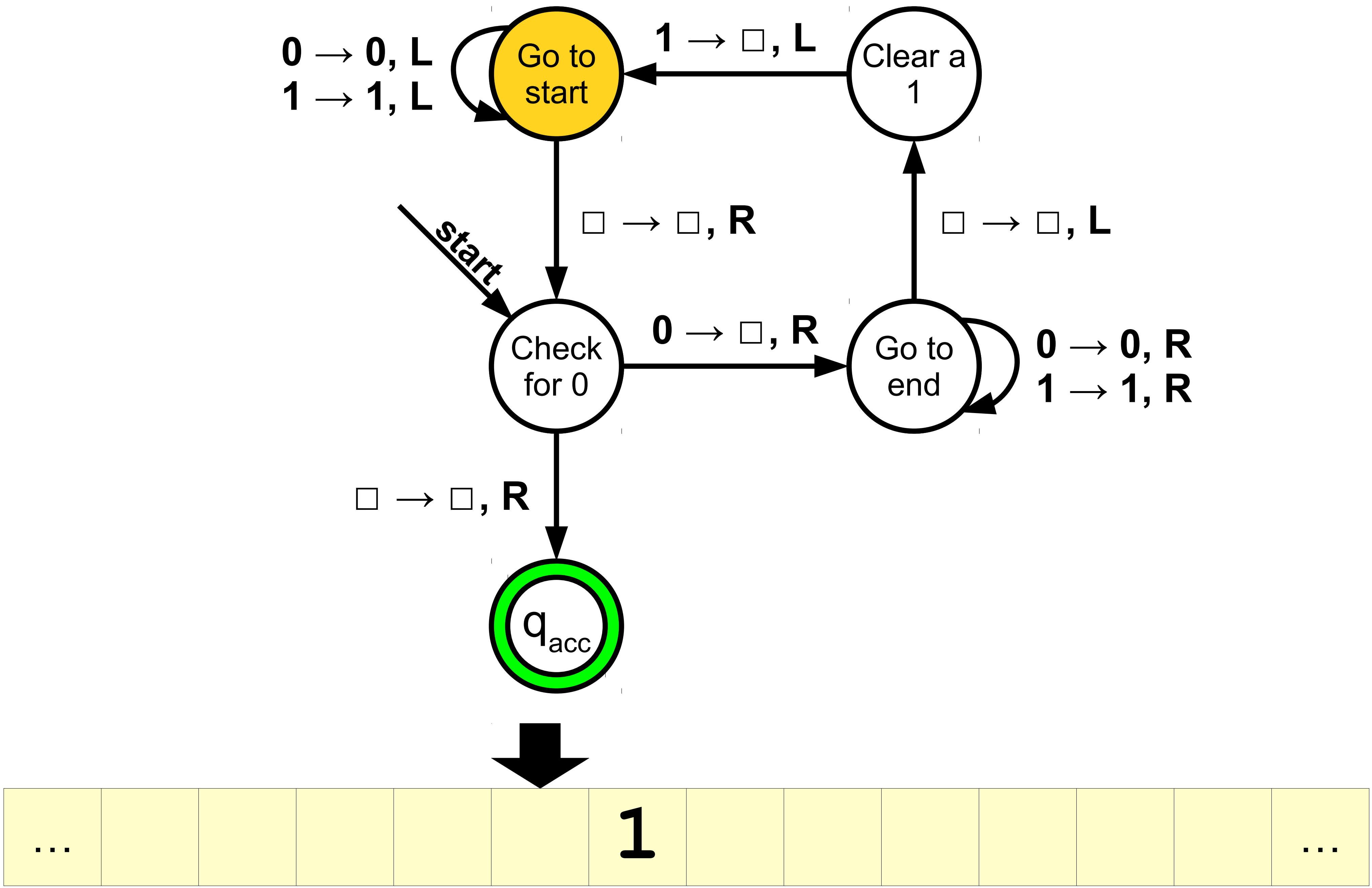


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
1
…

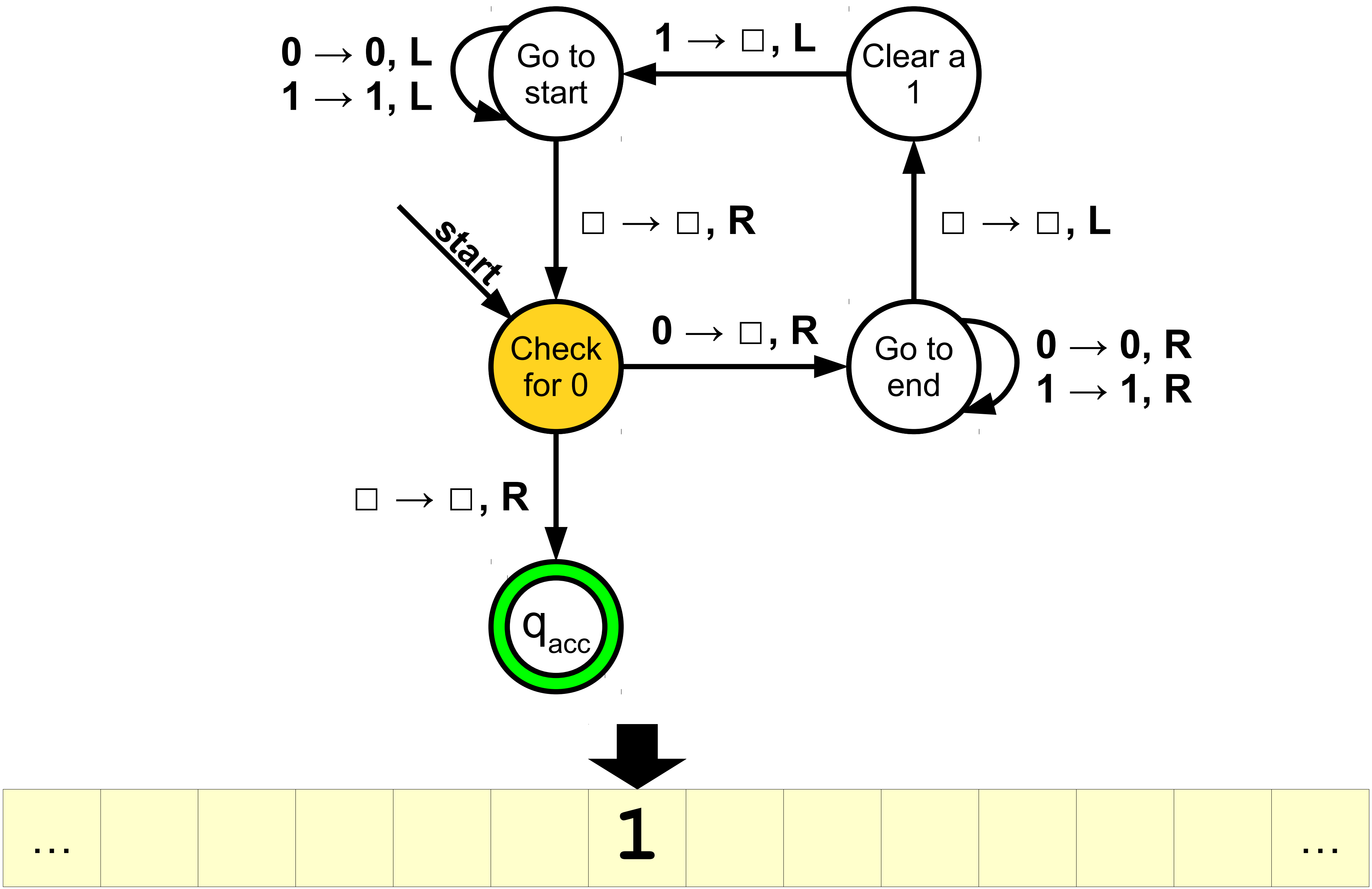

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
1
…



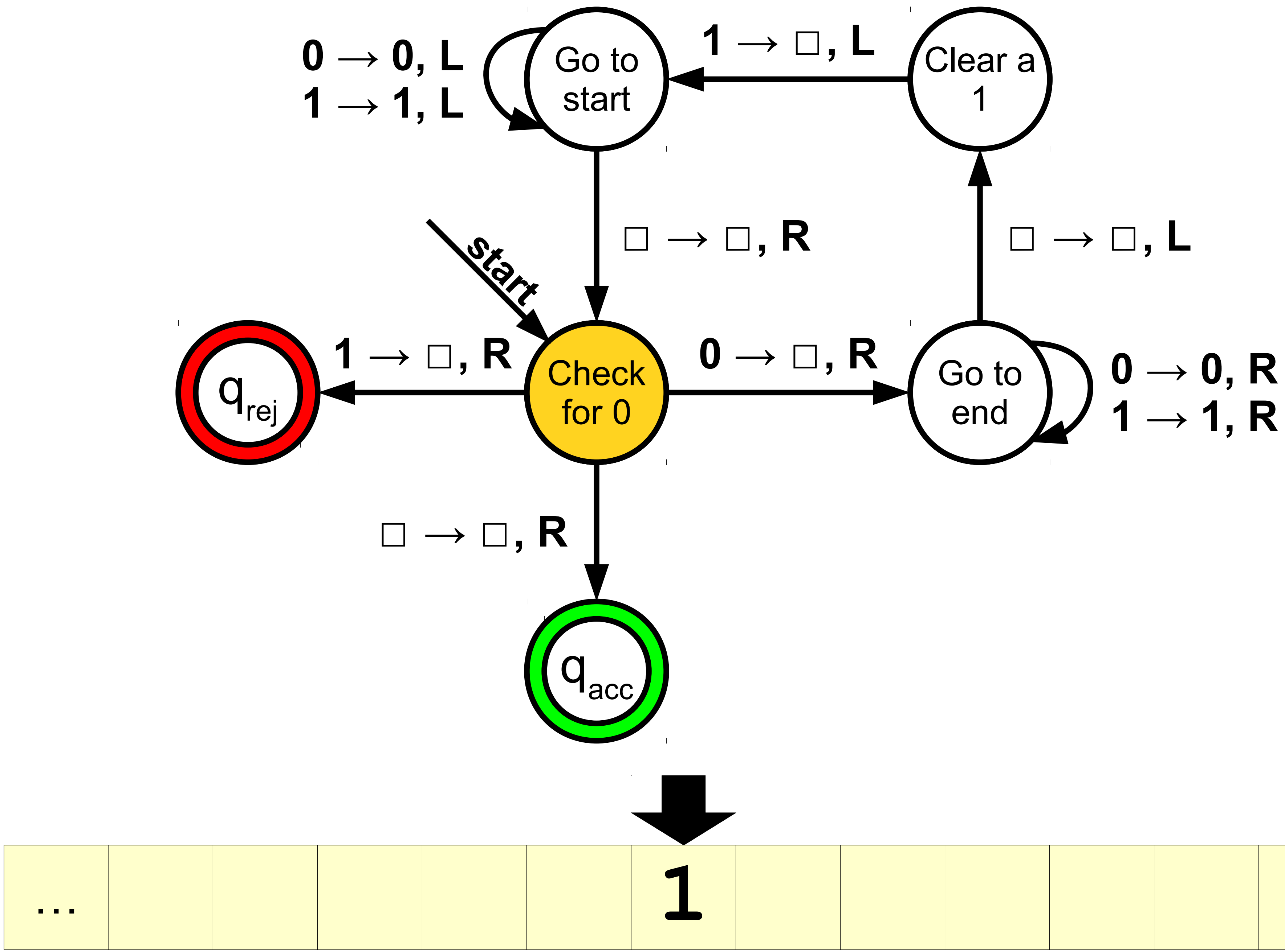
Go to start
Clear a 1
Check for 0
Go to end
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
1
…



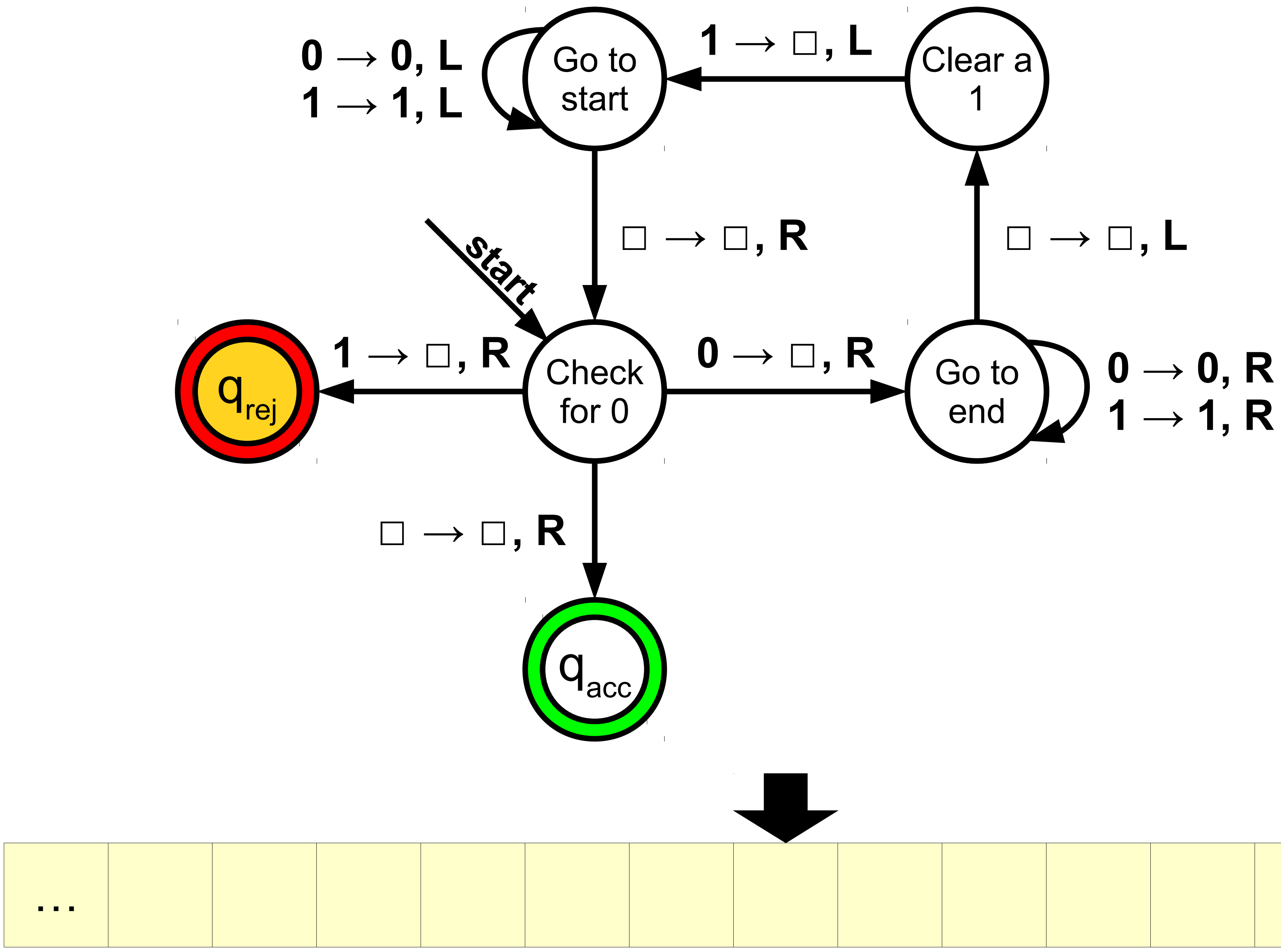
Go to start
Clear a 1
Check for 0
Go to end
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
1
…



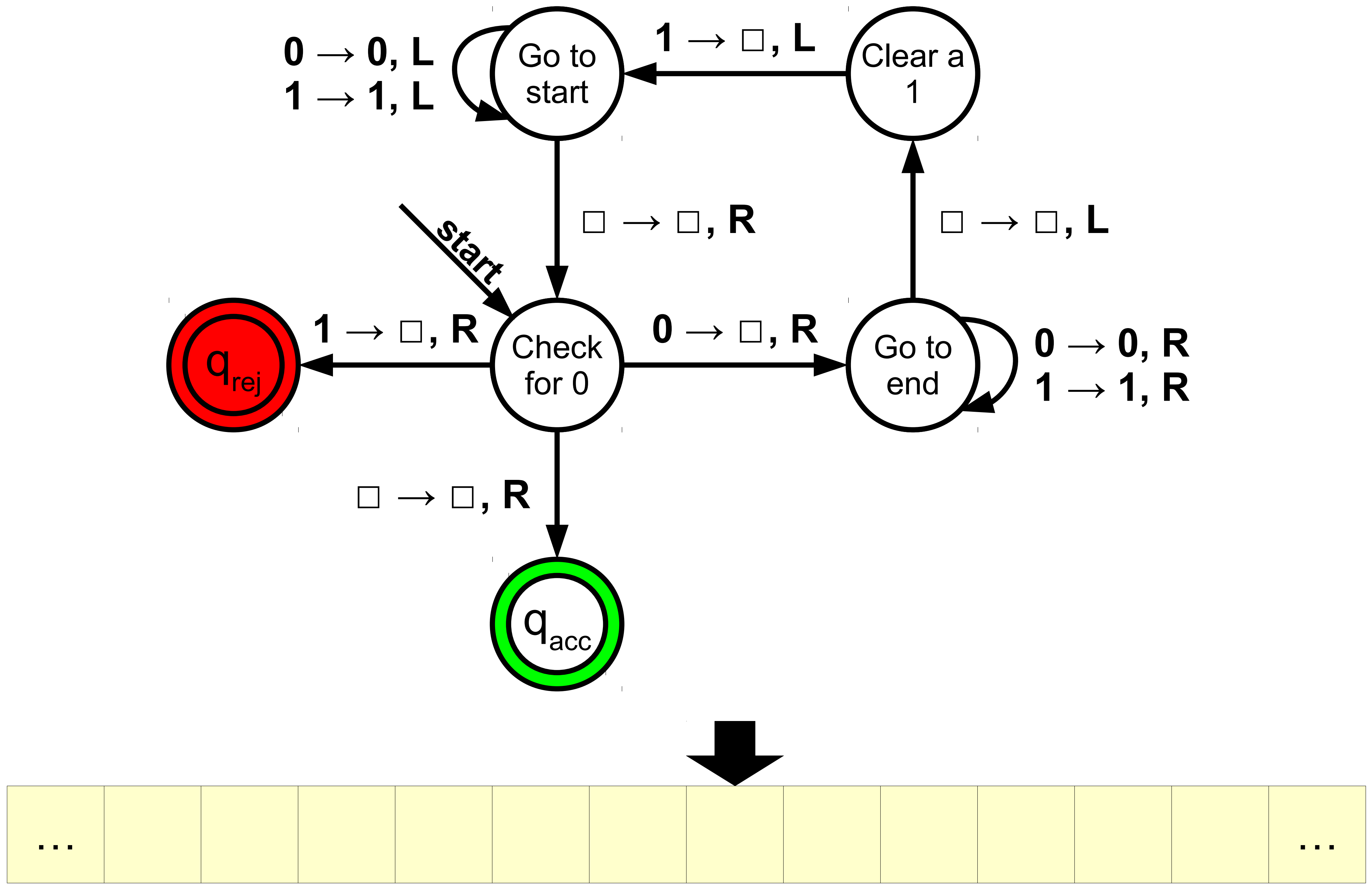
Go to start
Clear a 1
Check for 0
Go to end
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
1
…



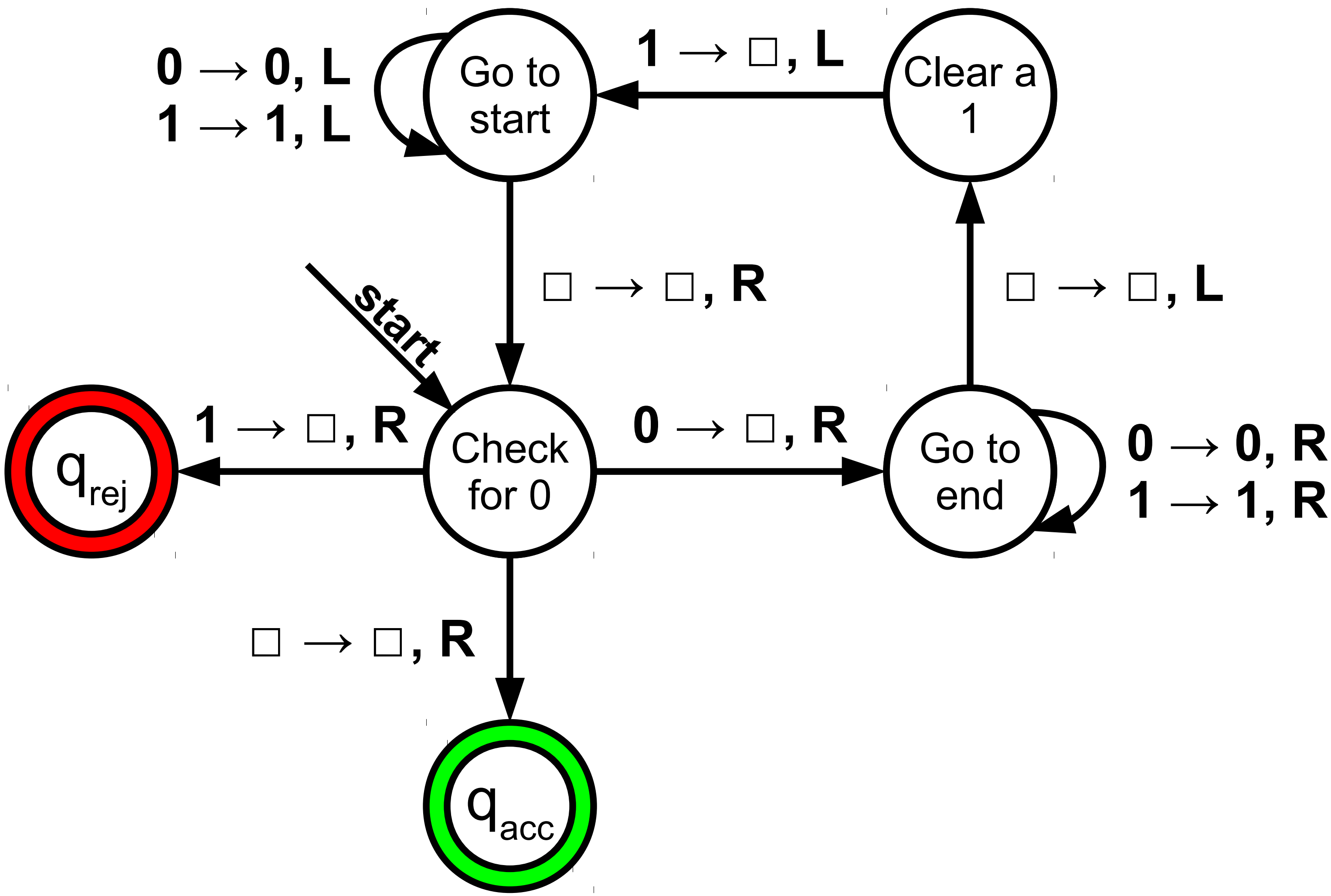
Go to start
Clear a 1
Check for 0
Go to end
start
q_{rej}
q_{acc}
$0 \to 0, L$
$1 \to 1, L$
$1 \to \square, L$
$\square \to \square, R$
$\square \to \square, L$
$1 \to \square, R$
$0 \to \square, R$
$0 \to 0, R$
$1 \to 1, R$
$\square \to \square, R$
…
1
…



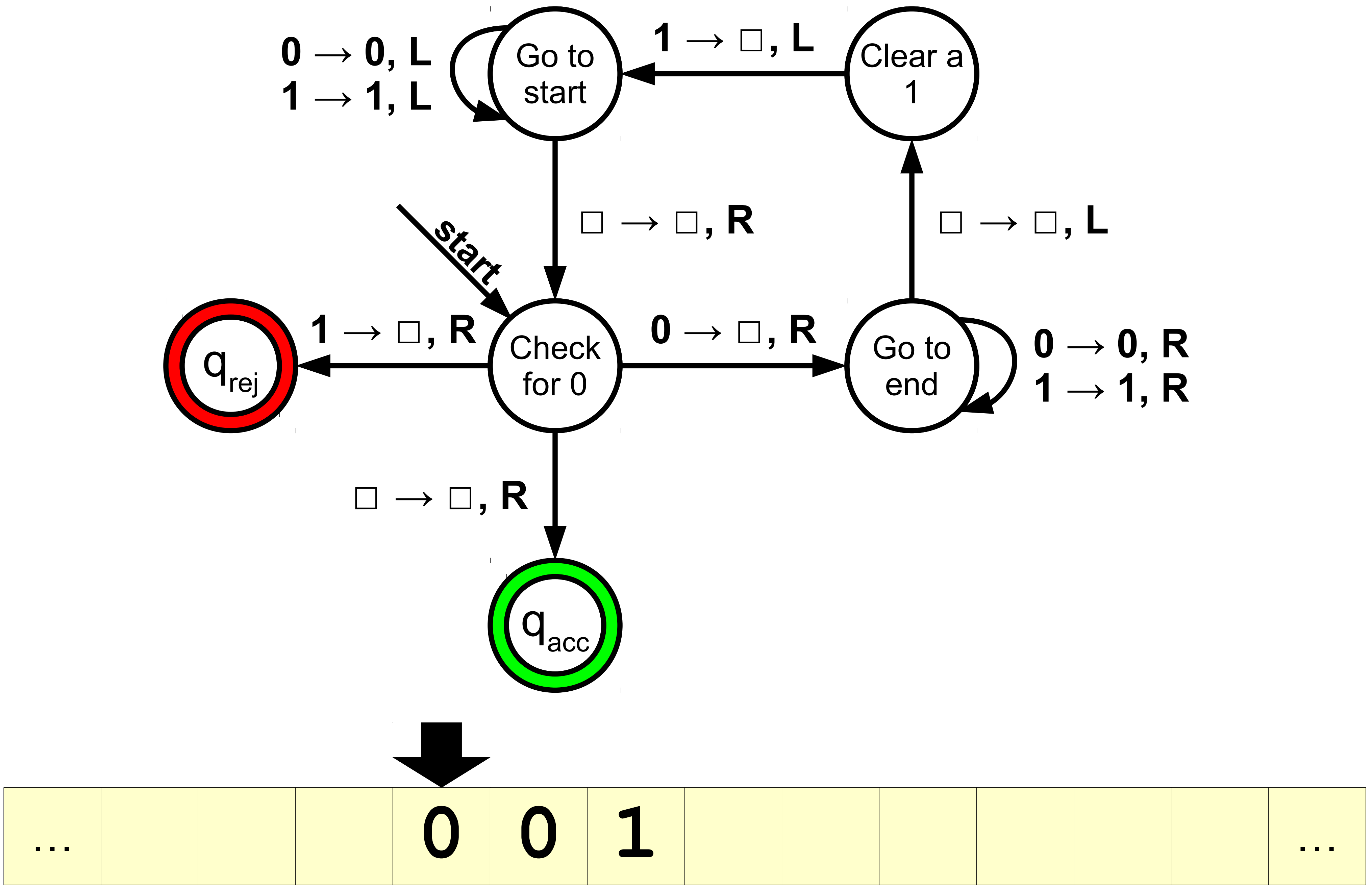
Go to start
Clear a 1
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
start
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_{rej}
□ → □, R
q_{acc}
…
…



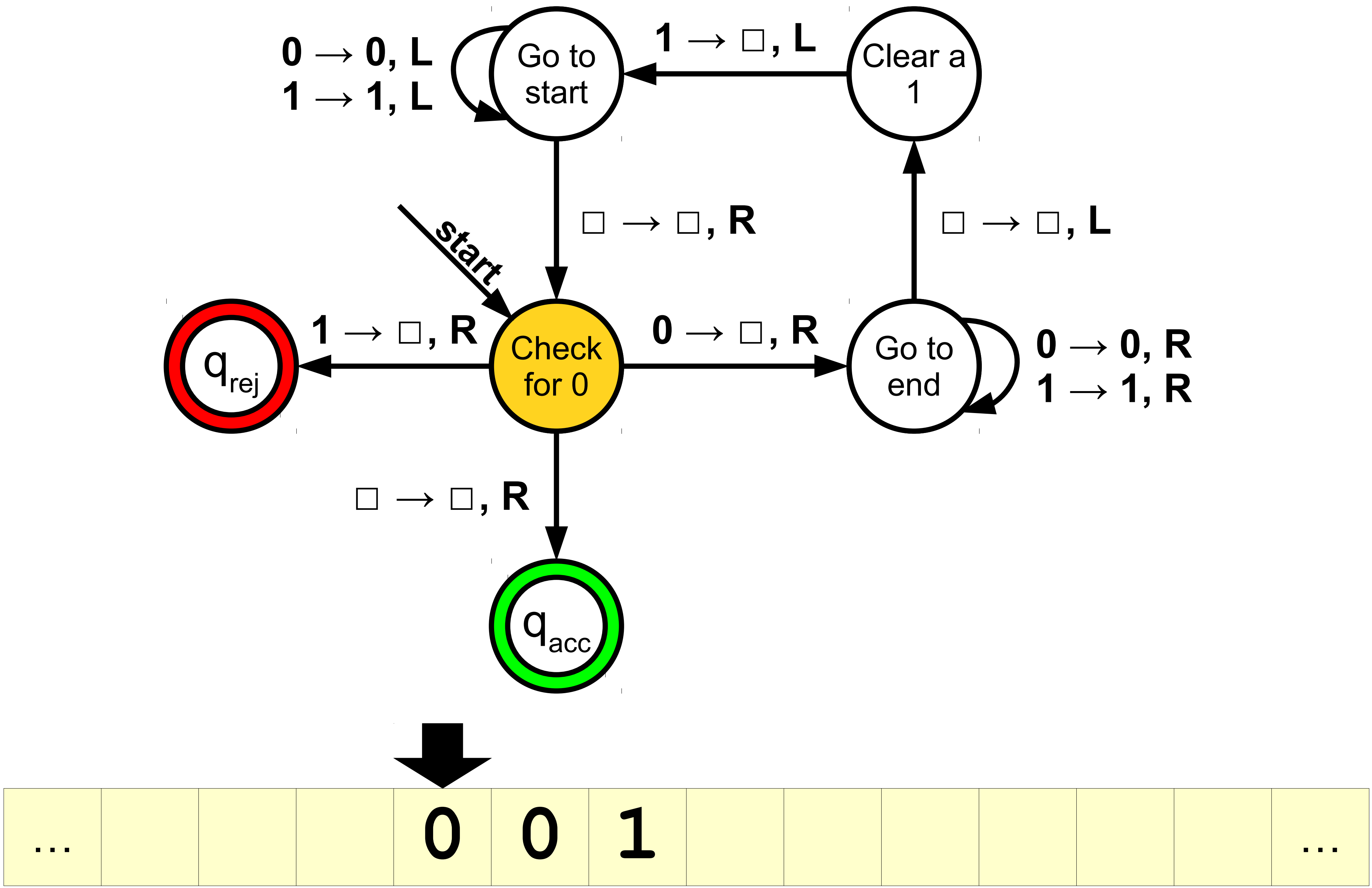
Go to start
Clear a 1
Check for 0
Go to end
q_rej
q_acc
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
1 → □, R
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
…

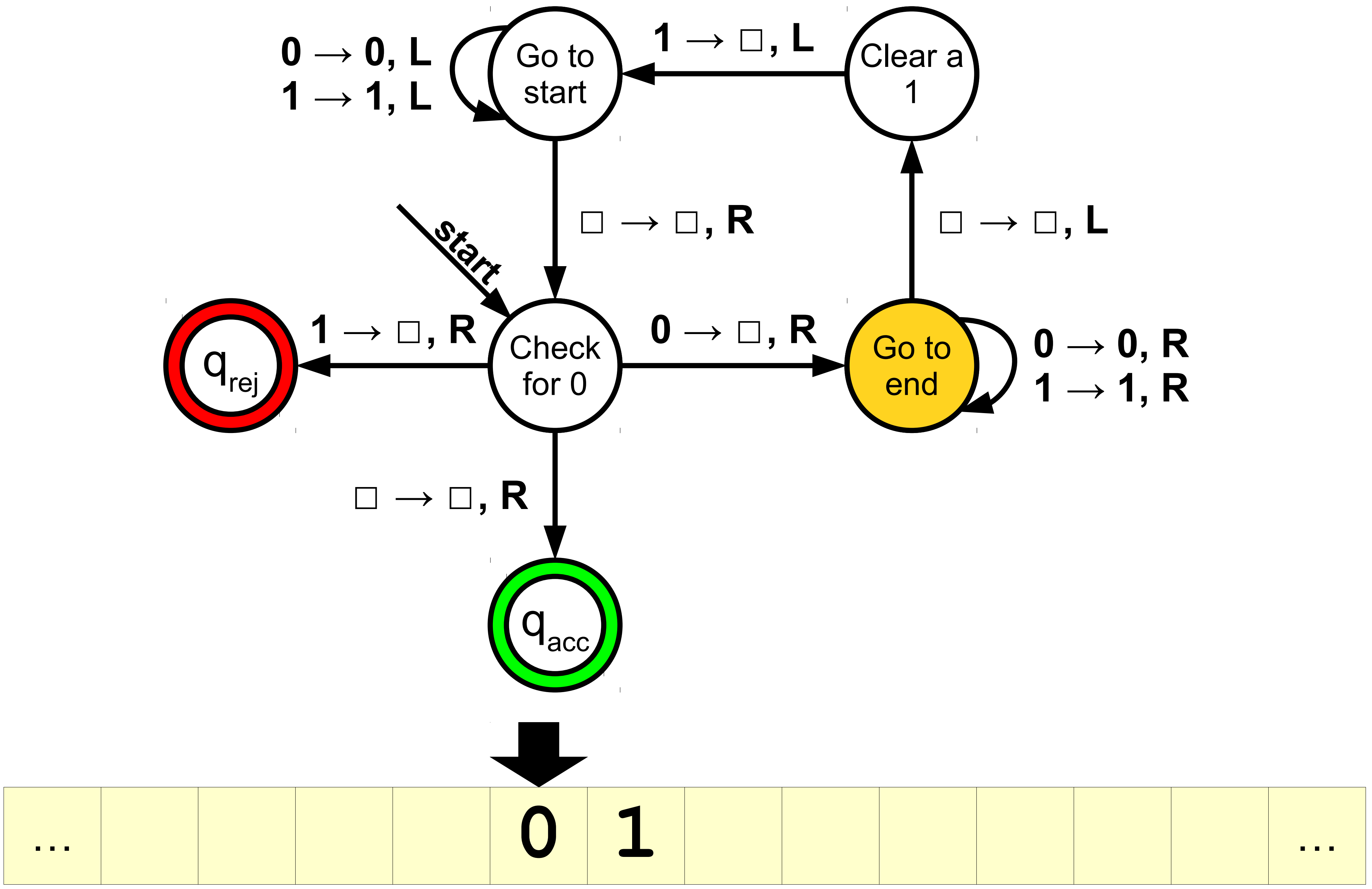


Go to start
Clear a 1
Check for 0
Go to end
q_{rej}
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
1 → □, R
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R

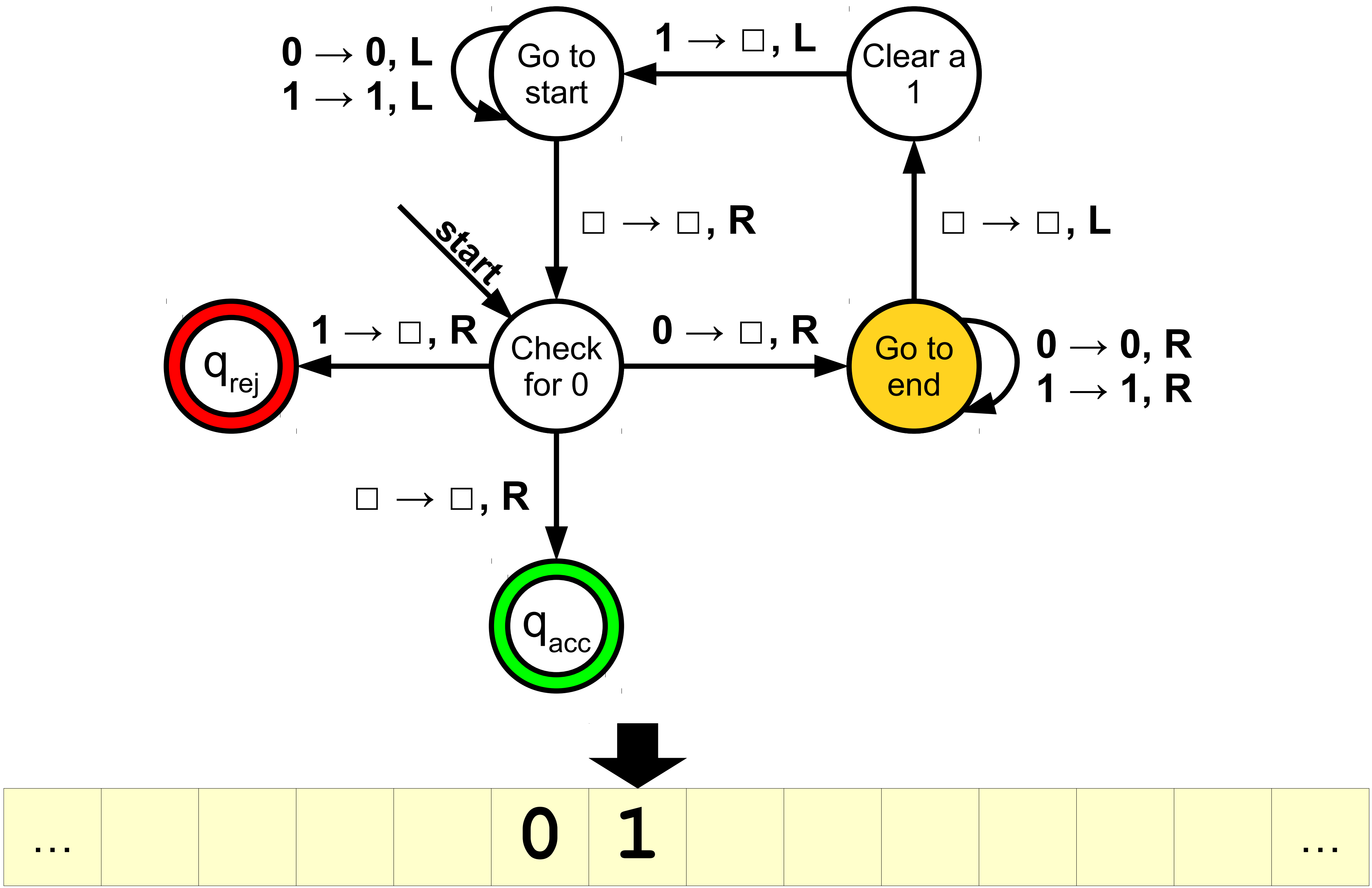


0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
q_rej
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
… 0 0 1 …

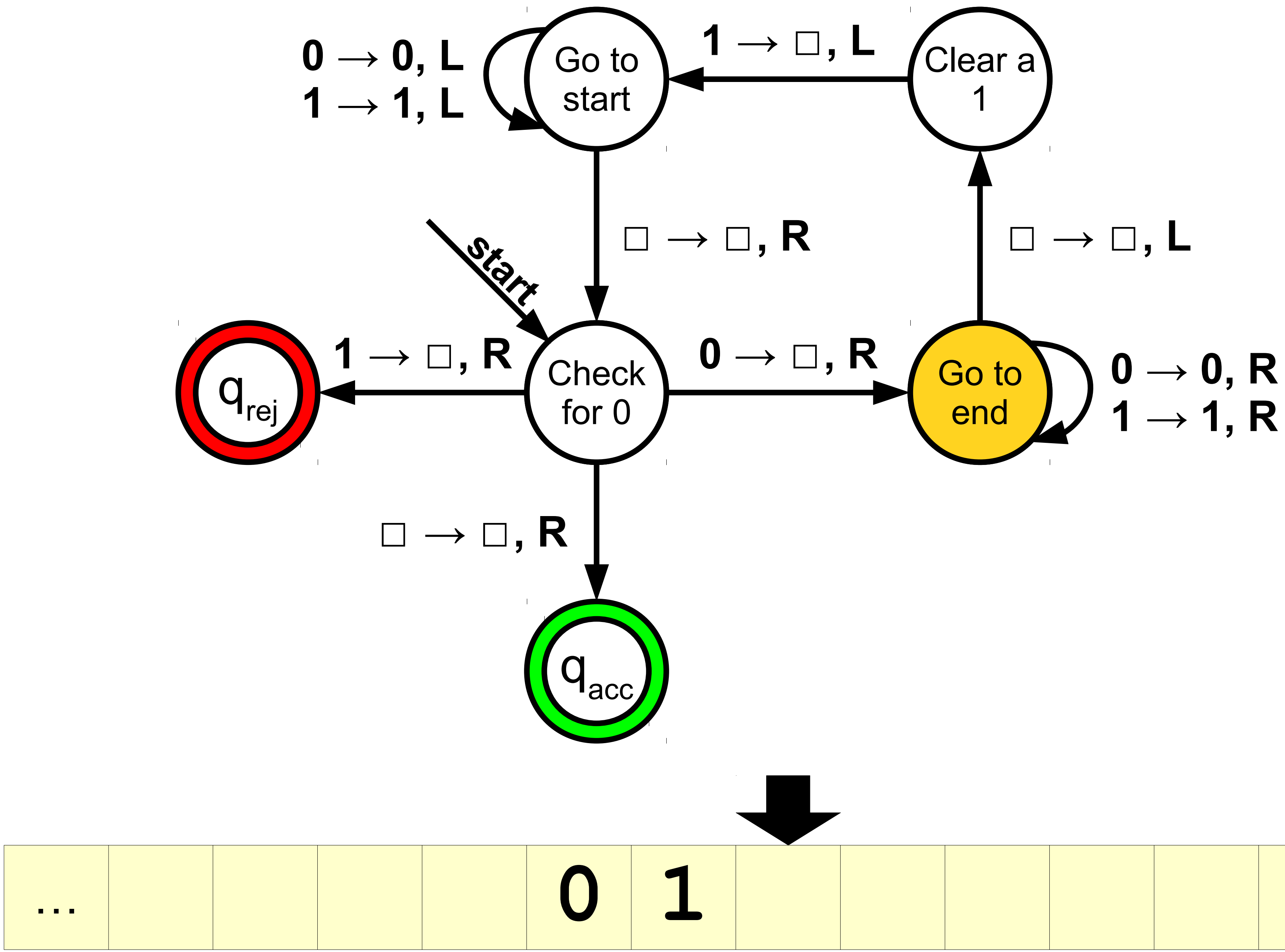

0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
...
0
0
1
...



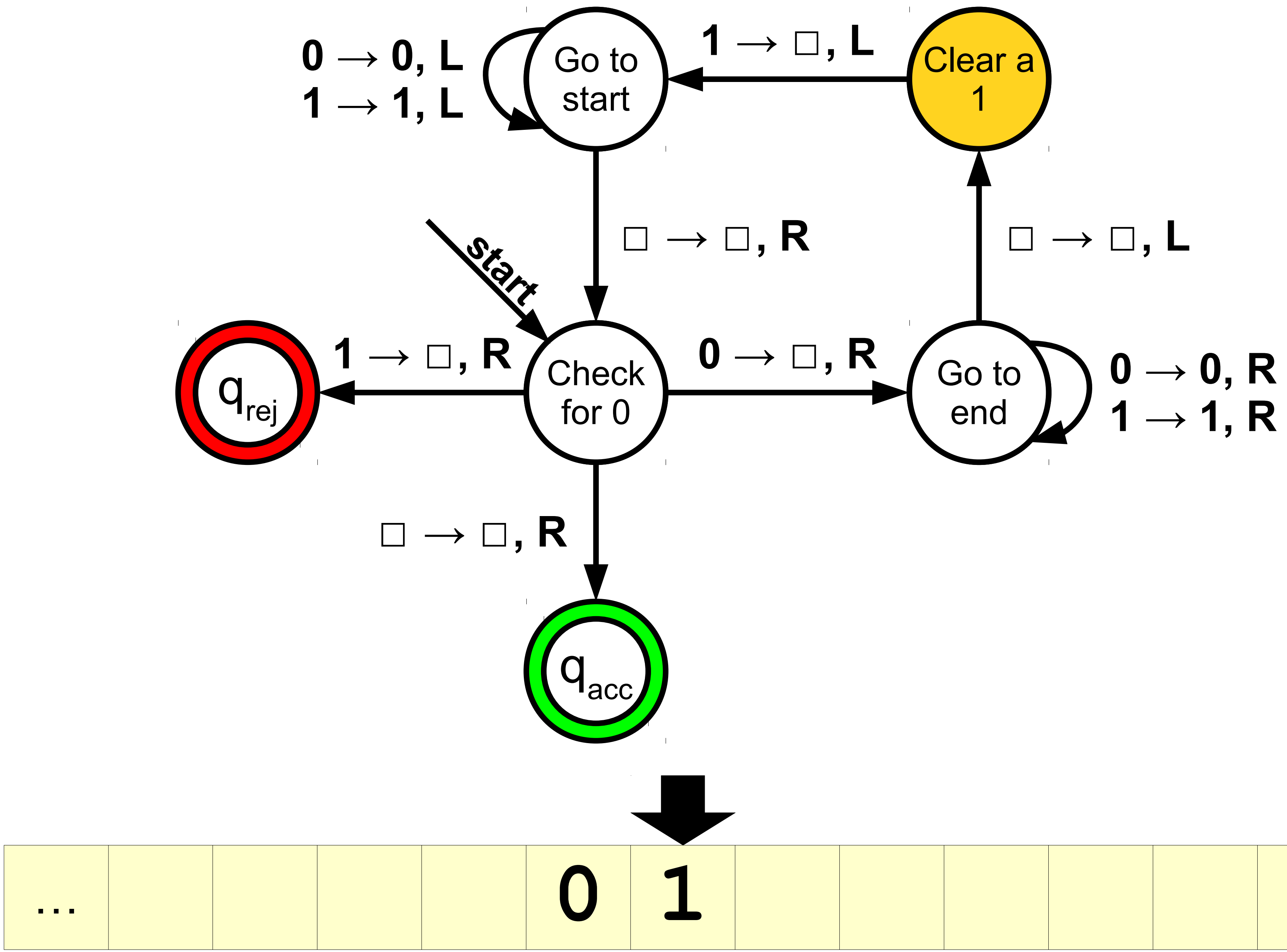
Go to start
Clear a 1
Check for 0
Go to end
q_rej
q_acc
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
1 → □, R
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
0
1
…



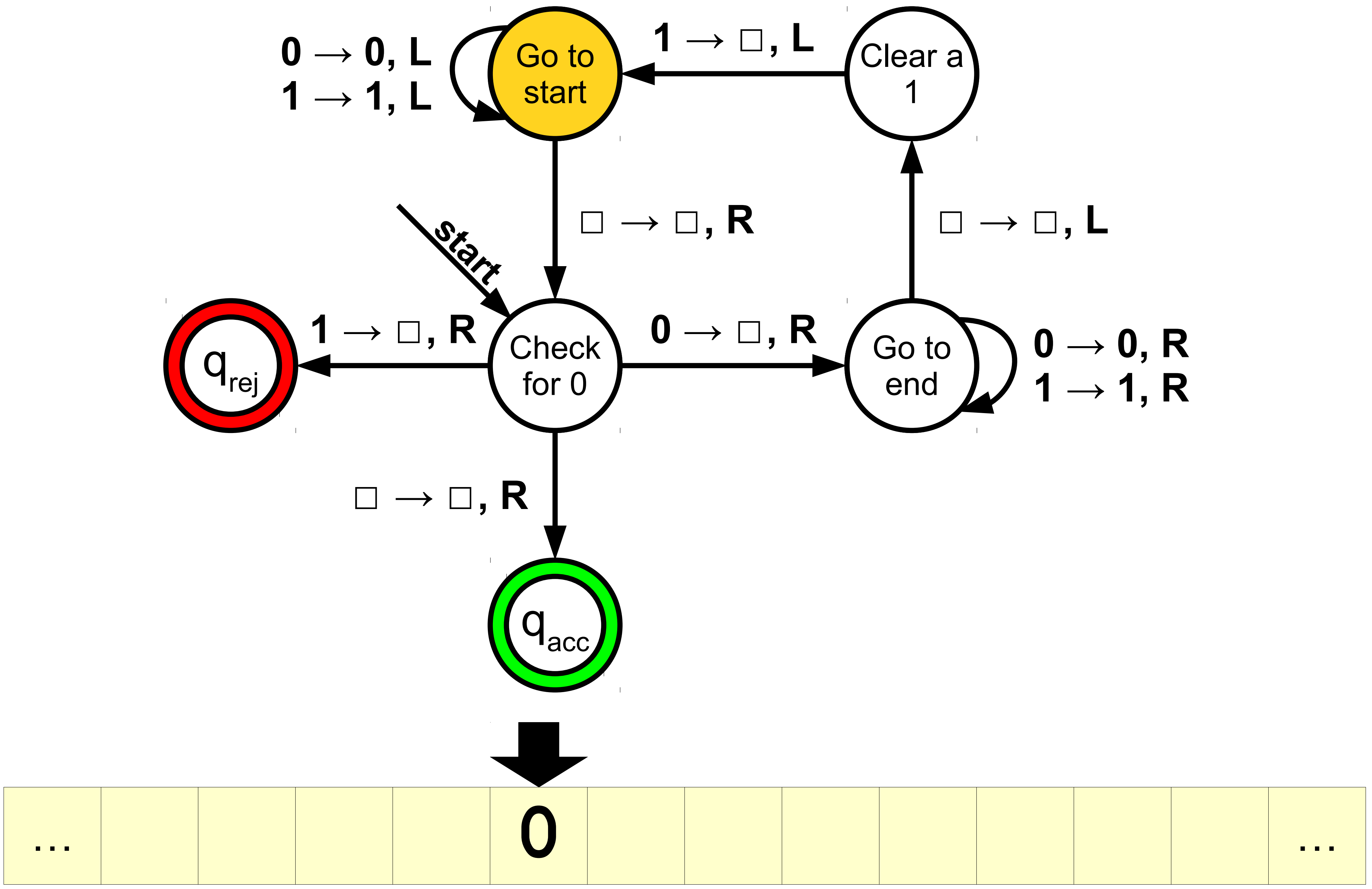
Go to start
Clear a 1
Check for 0
Go to end
q_{rej}
q_{acc}
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
1 → □, R
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R
…
0
1
…



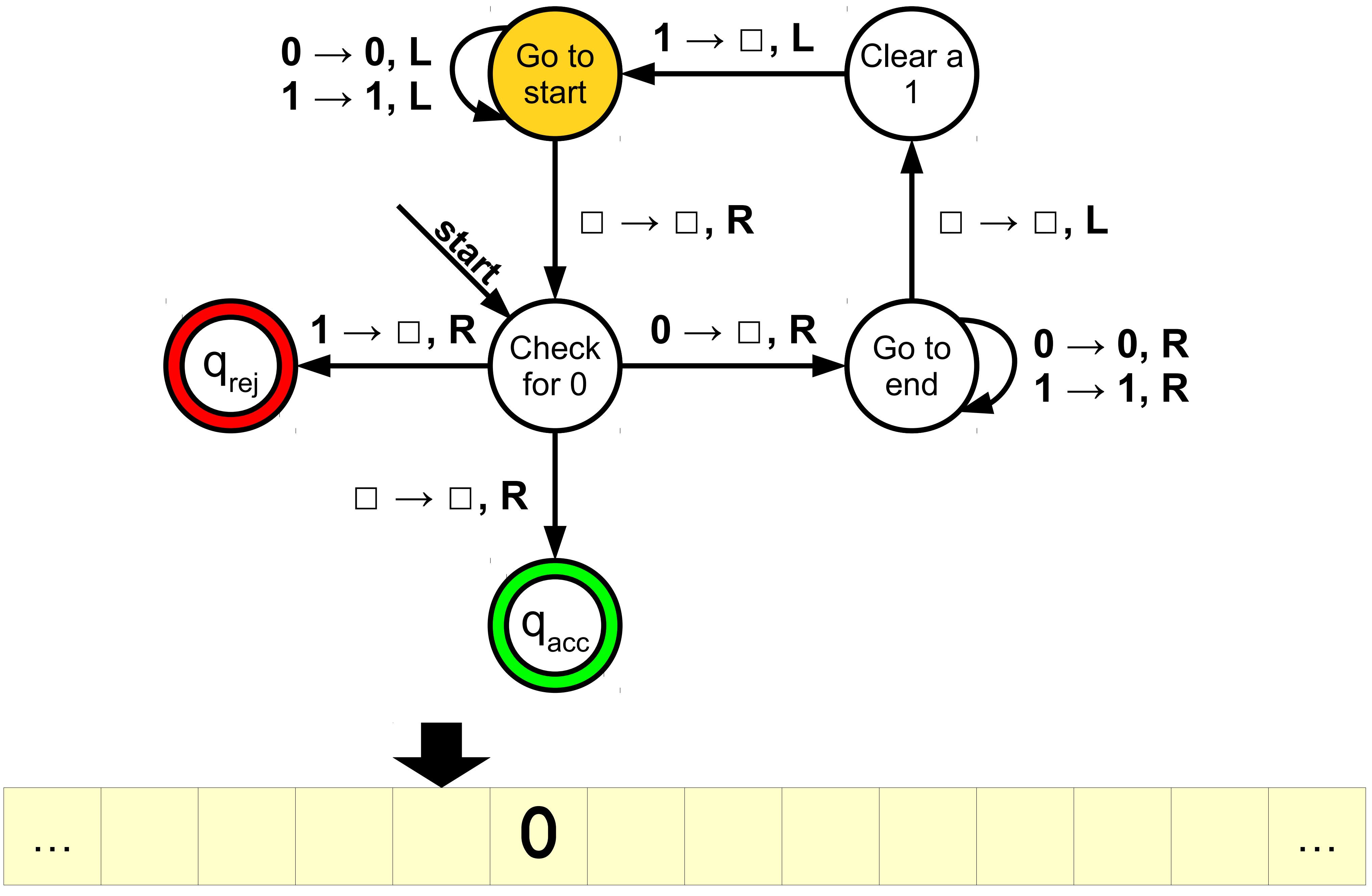
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
…



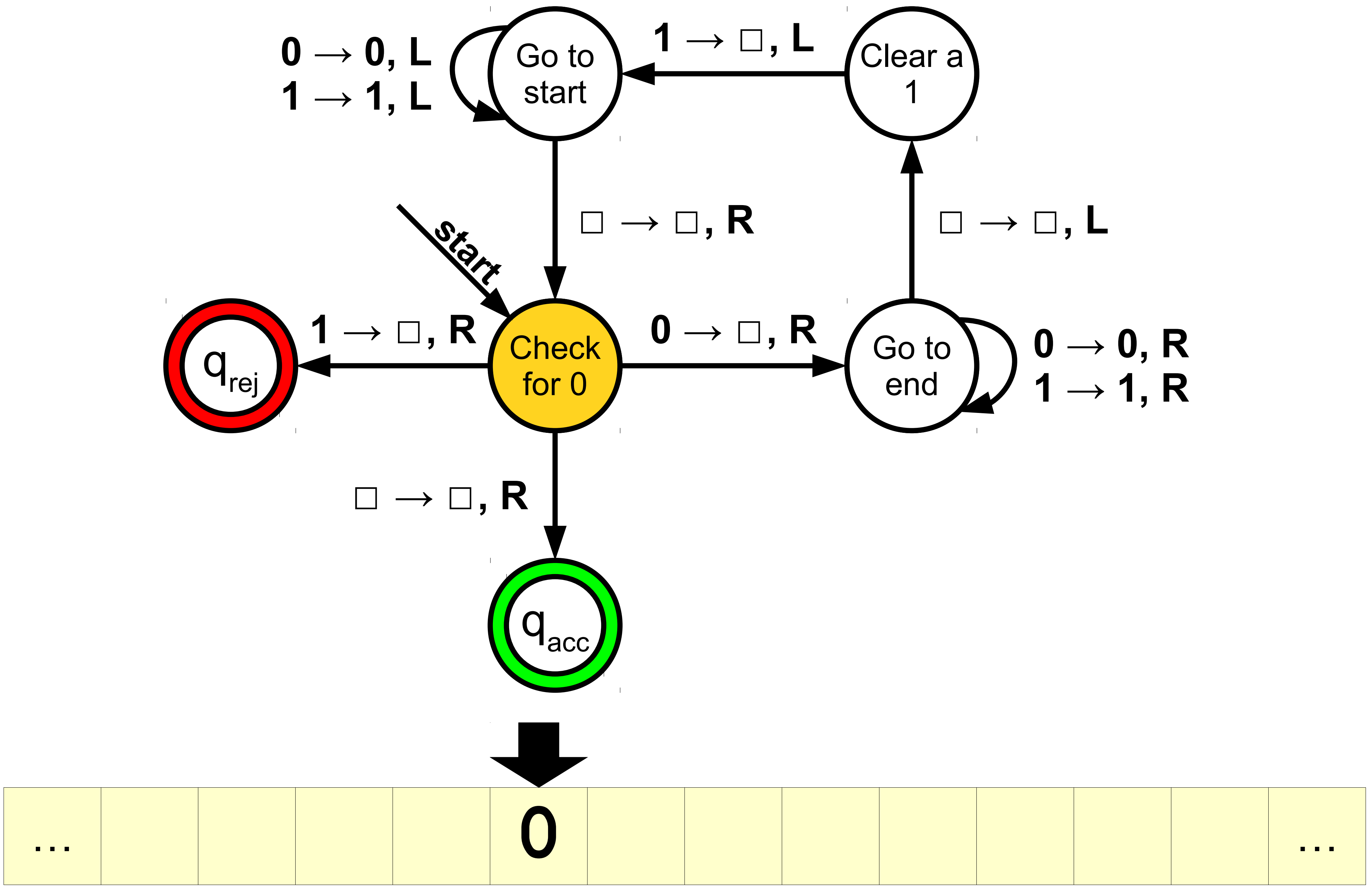
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
q_rej
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
…



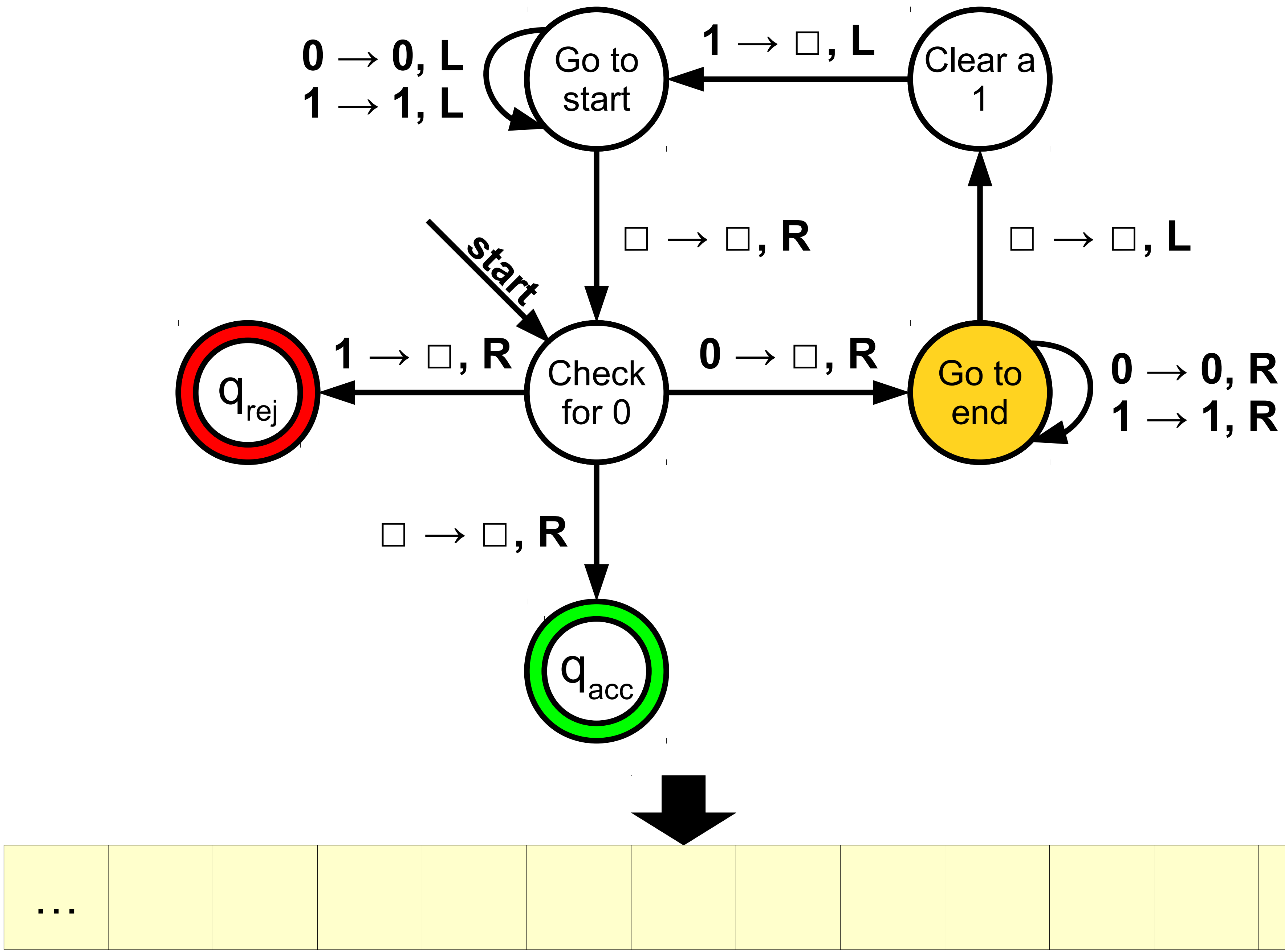
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
q_{rej}
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
0
…



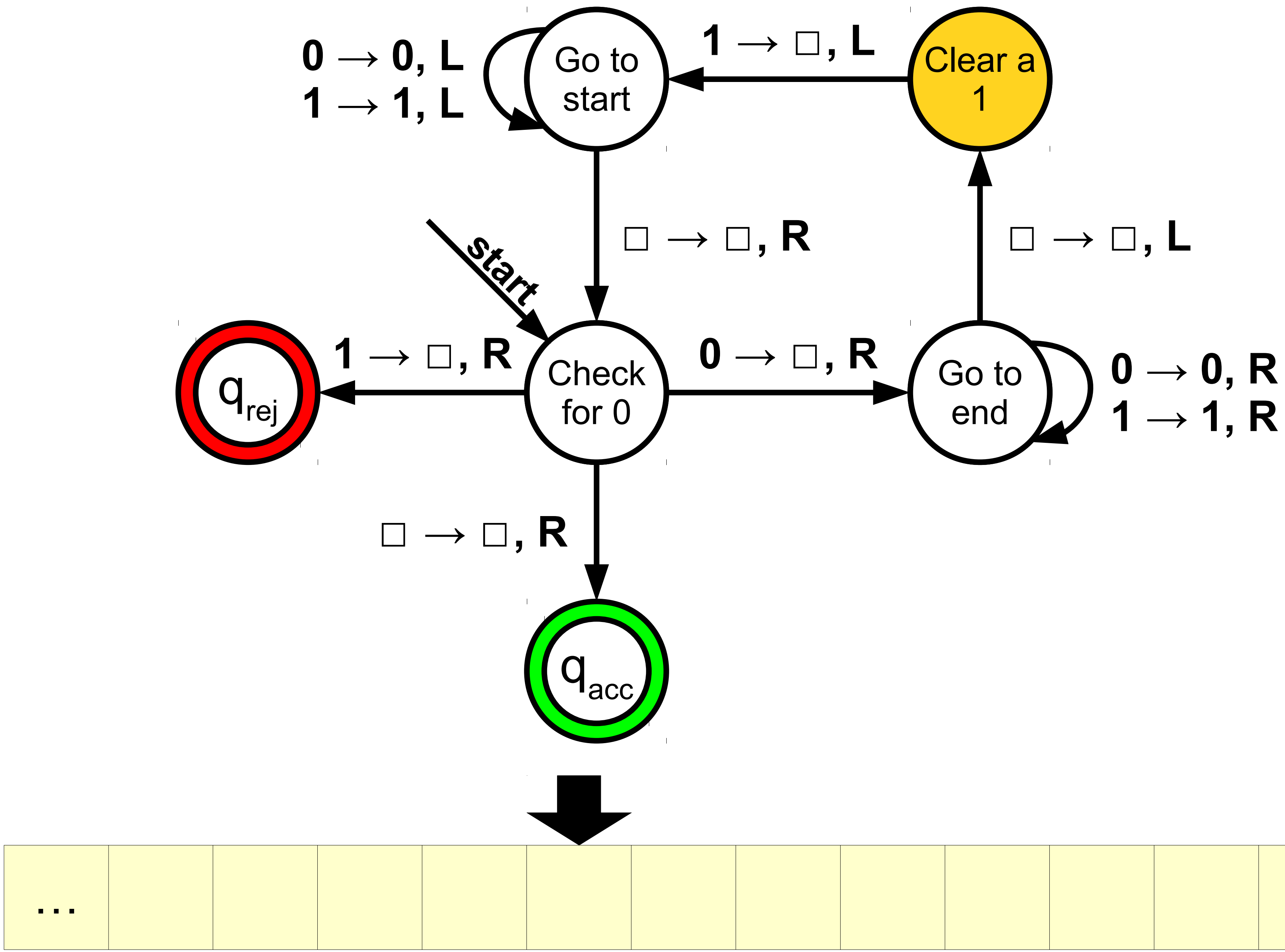
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_{rej}
□ → □, R
q_{acc}
…
0
…



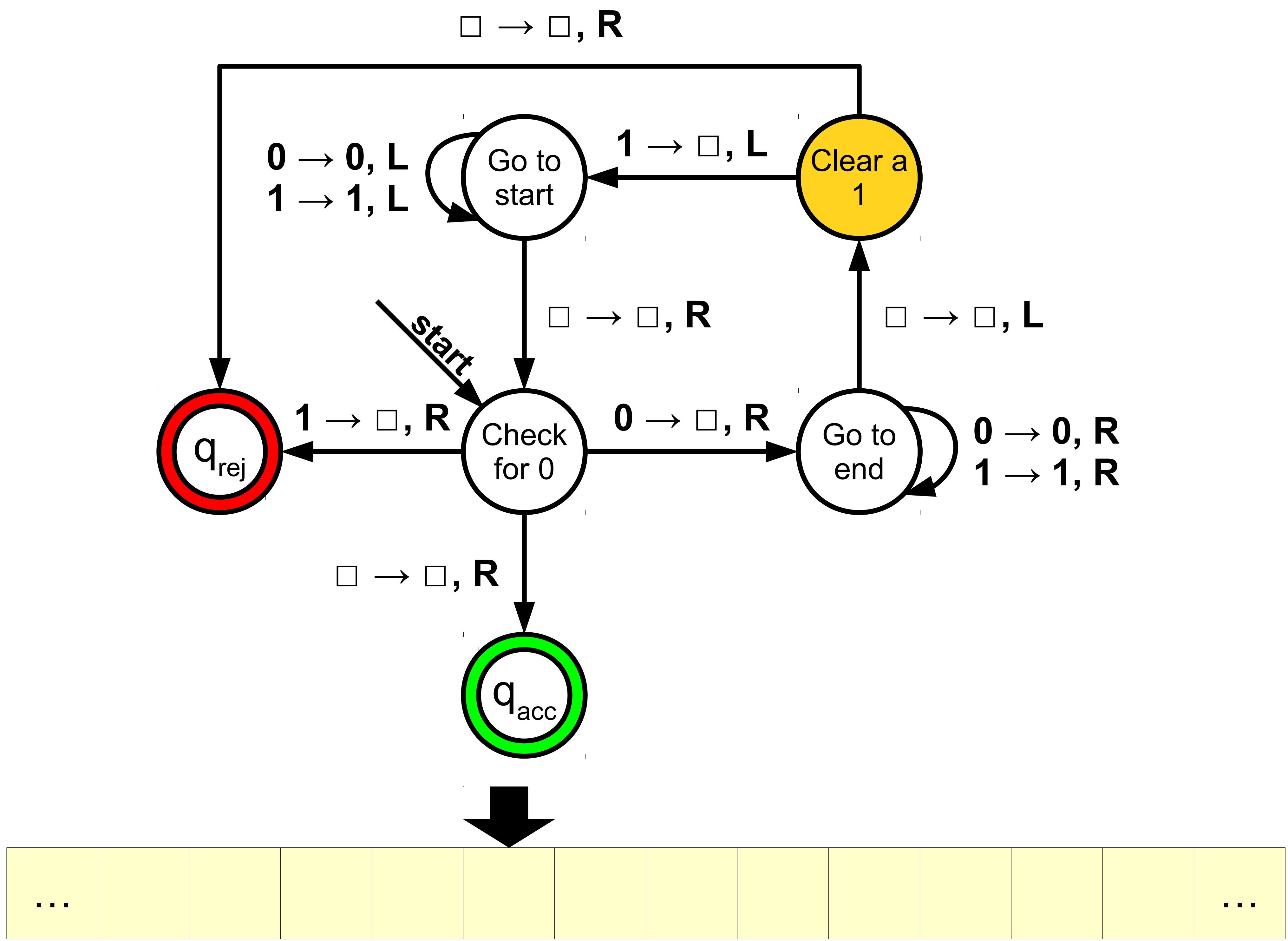
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
…



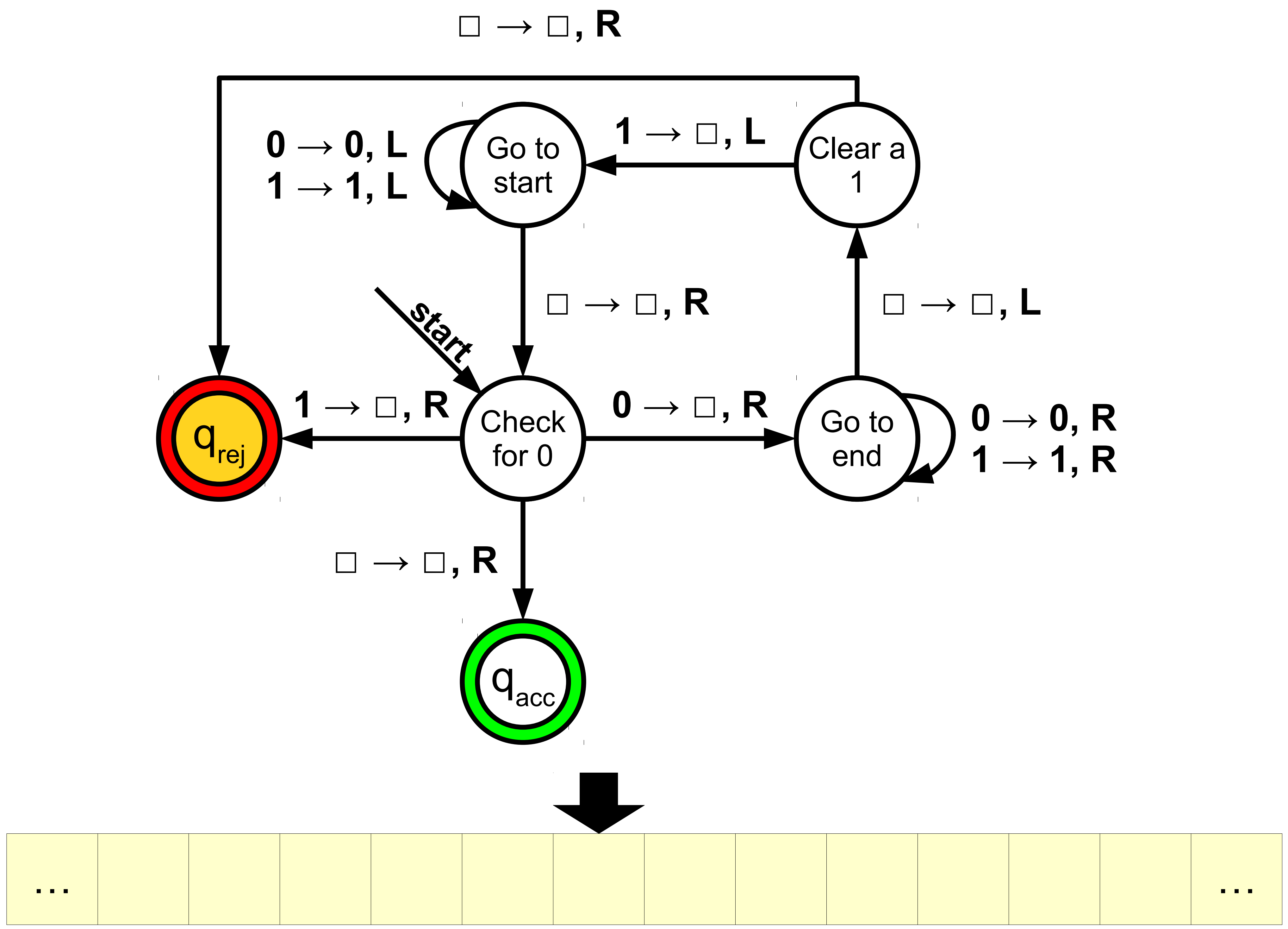
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_rej
□ → □, R
q_acc
…
…



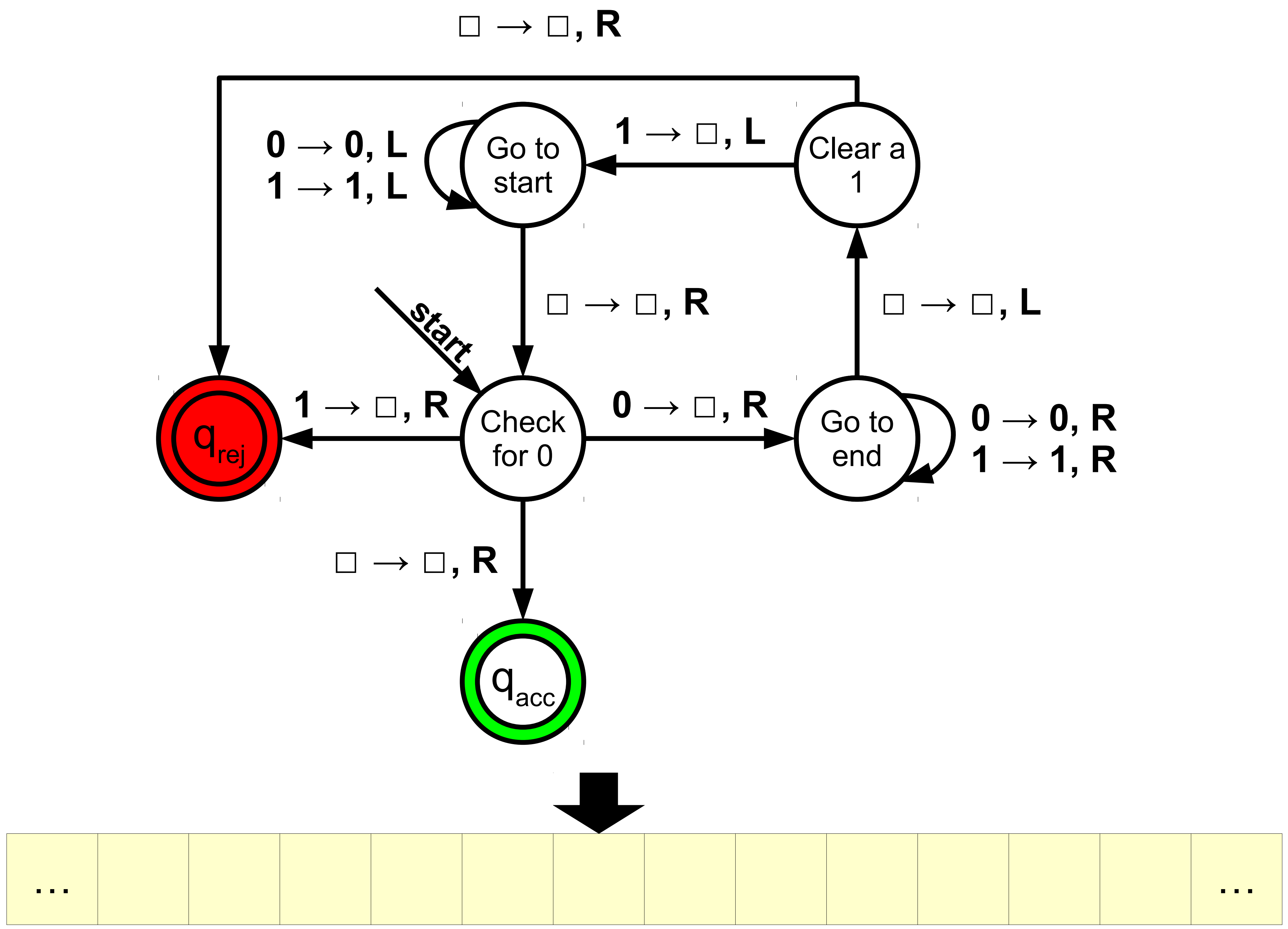
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
q_rej
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
…

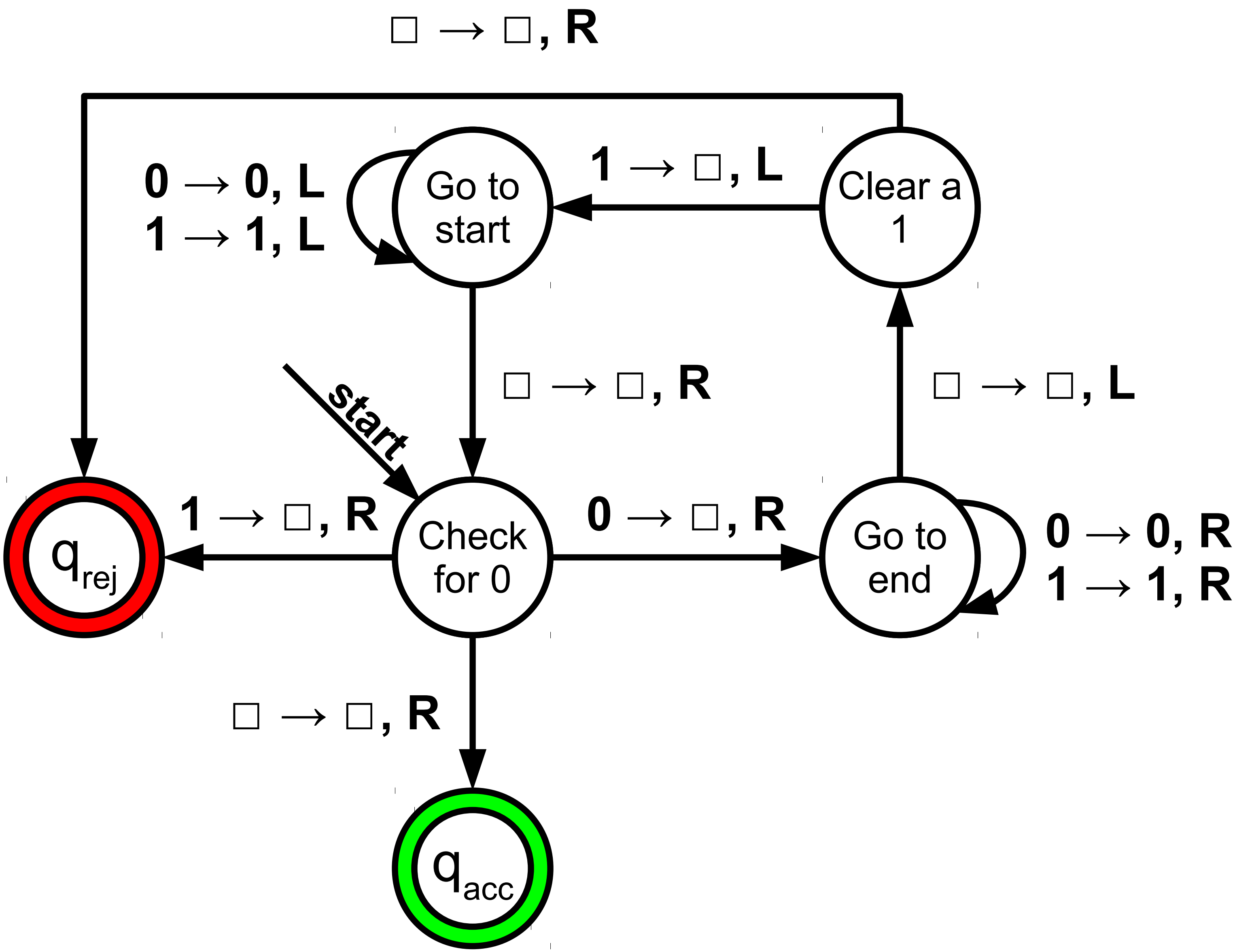


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
...
...

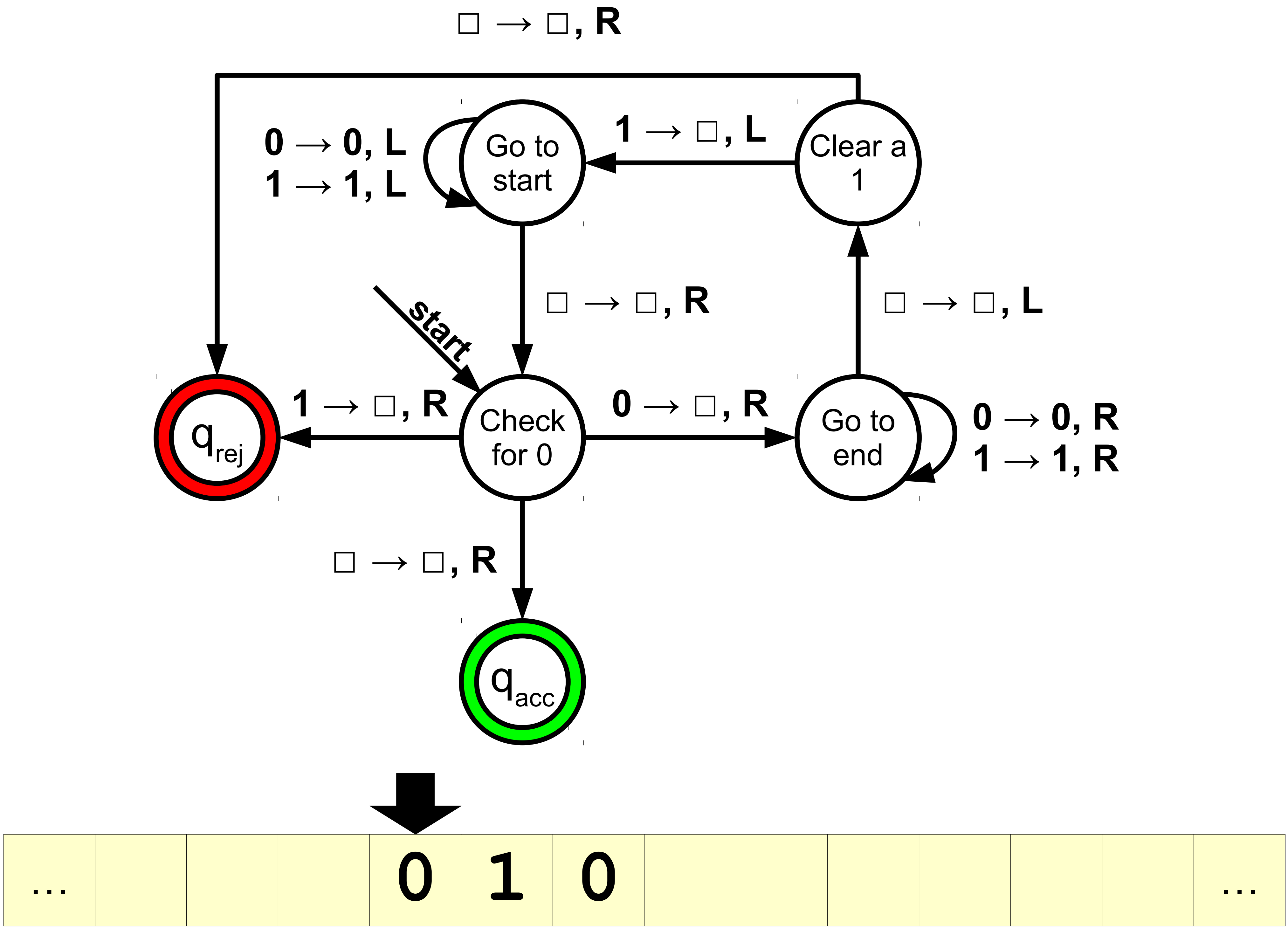


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
q_{rej}
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
…

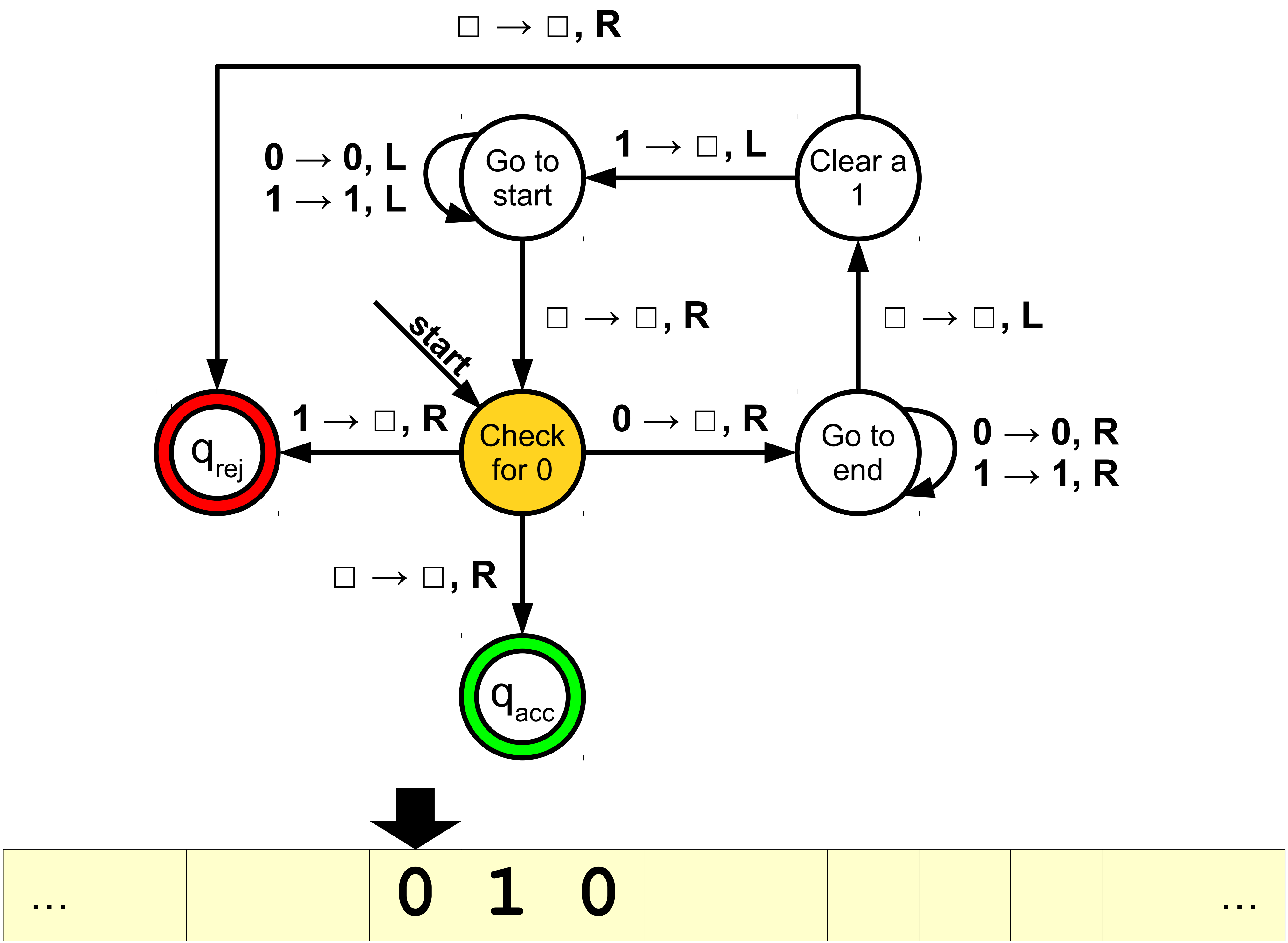

□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
...
...



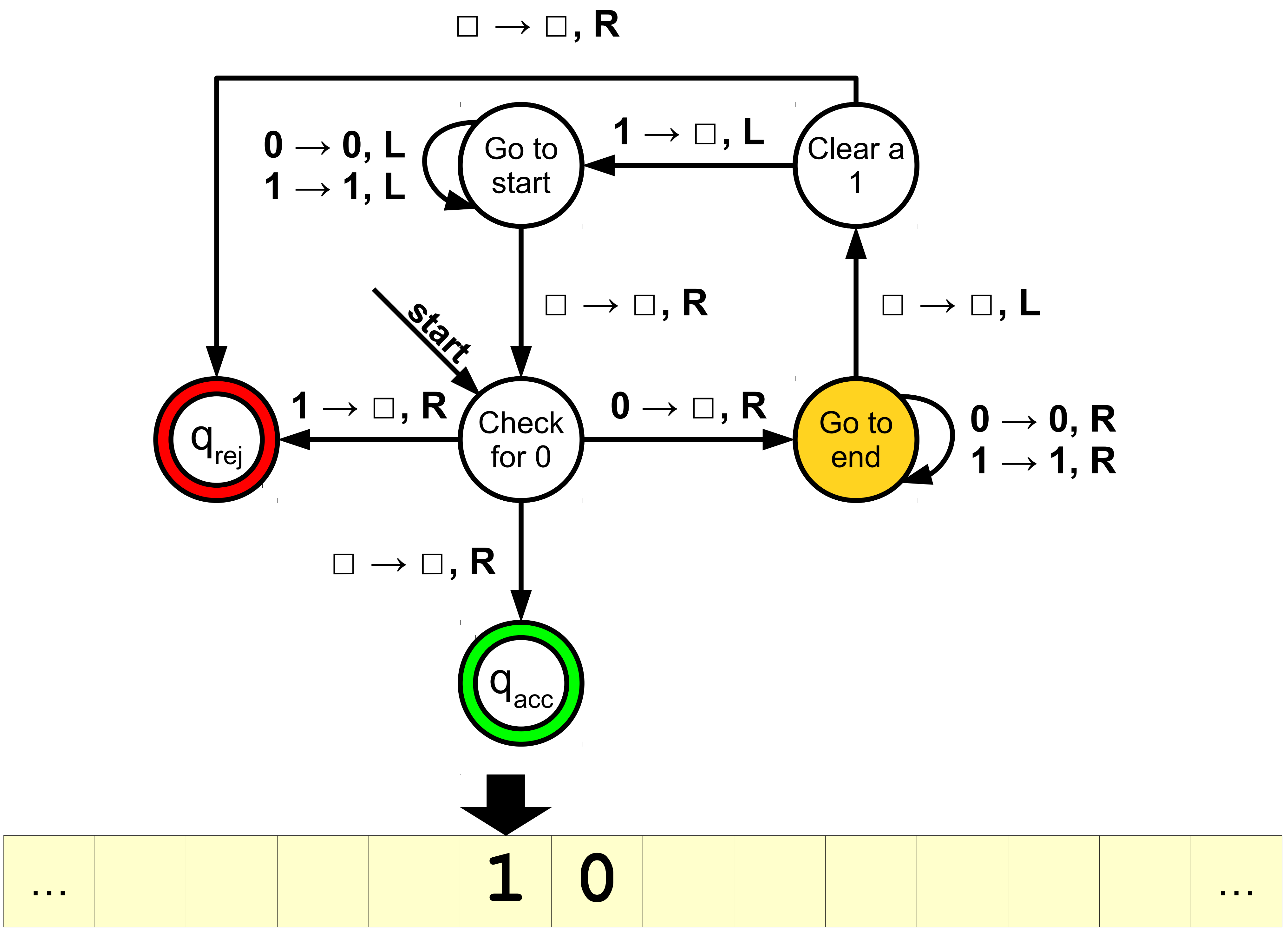
□ → □, R
Go to start
Clear a 1
0 → 0, L
1 → 1, L
1 → □, L
start
□ → □, R
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc

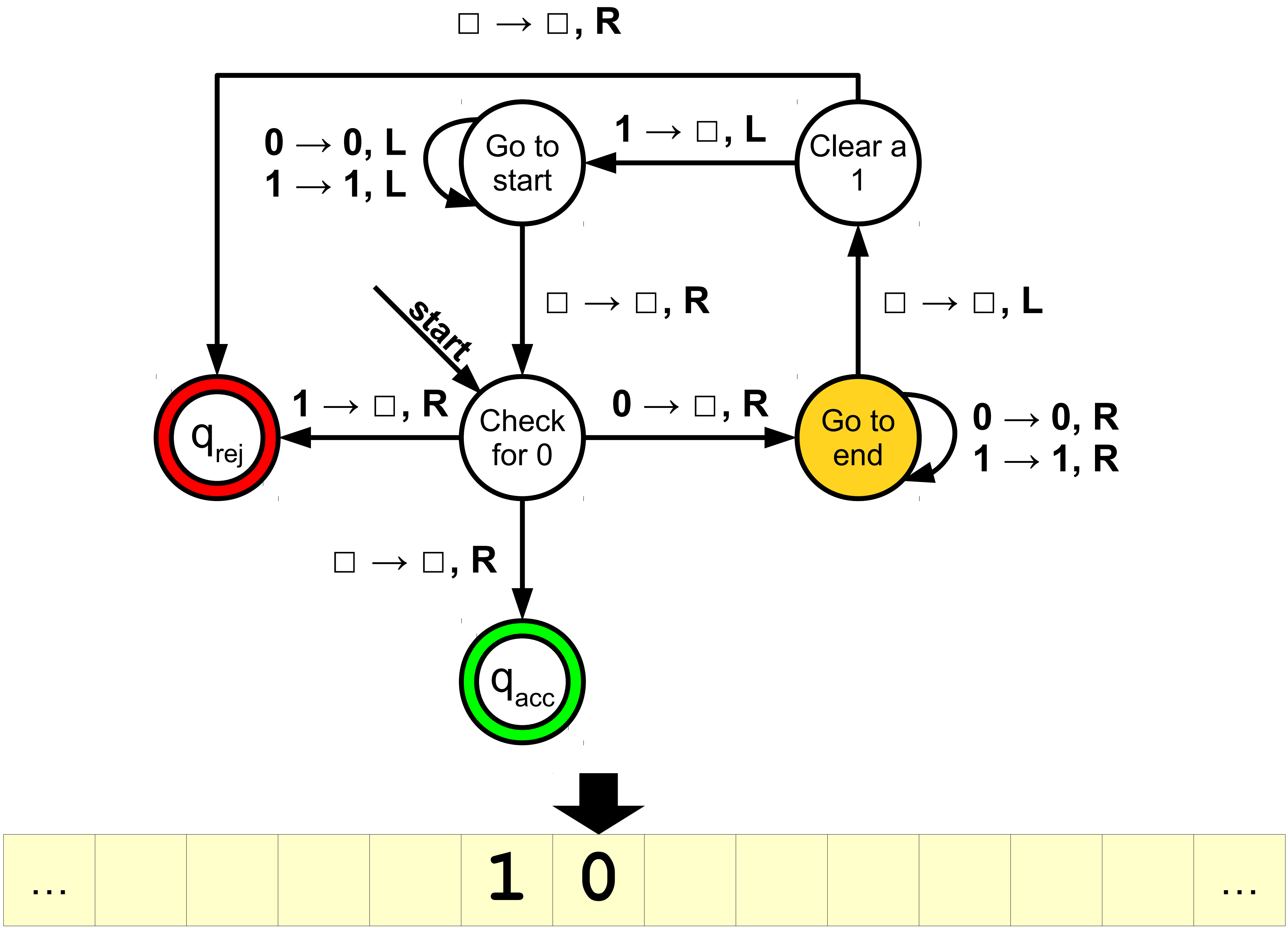


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
0
1
0
…

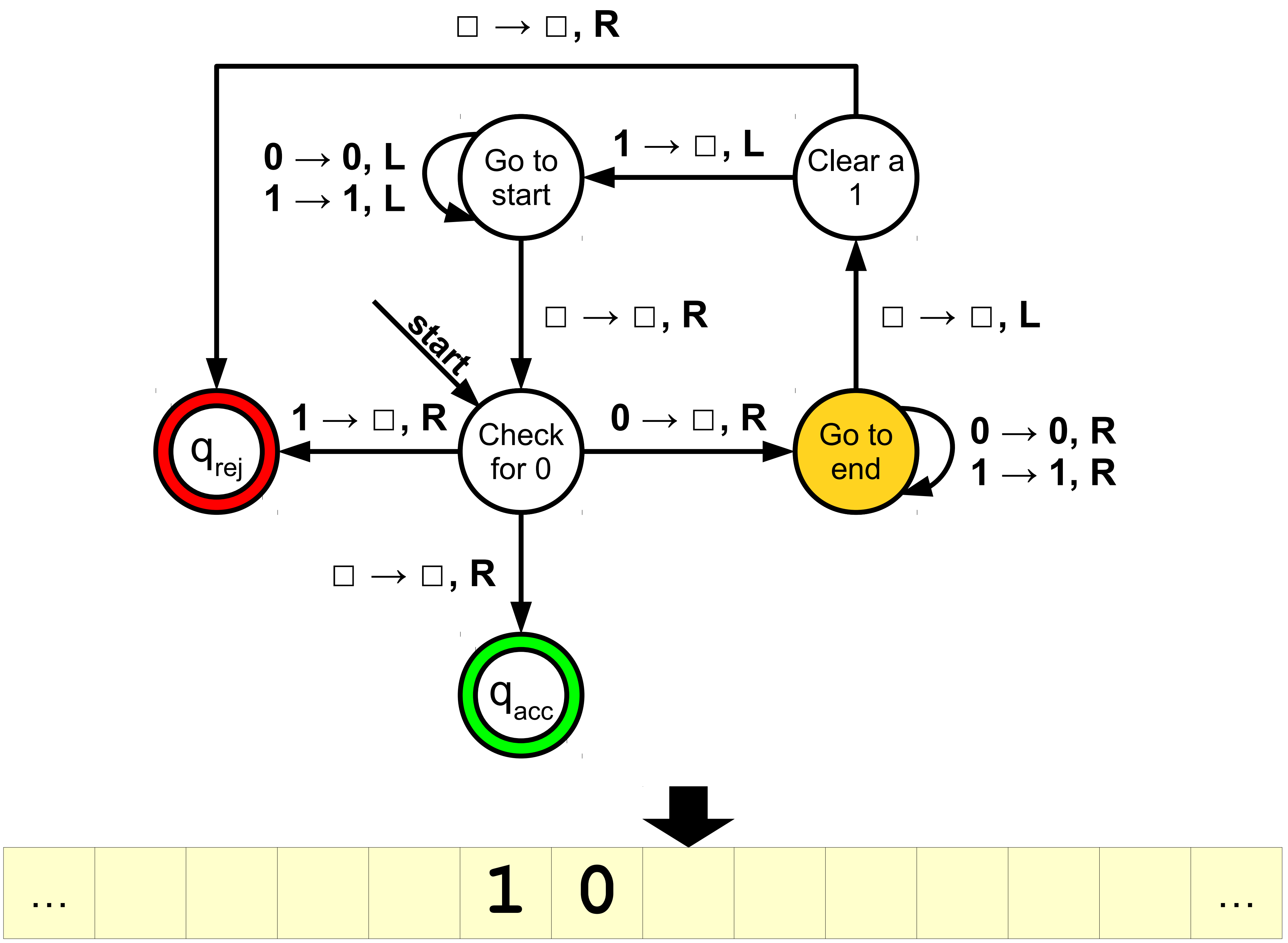


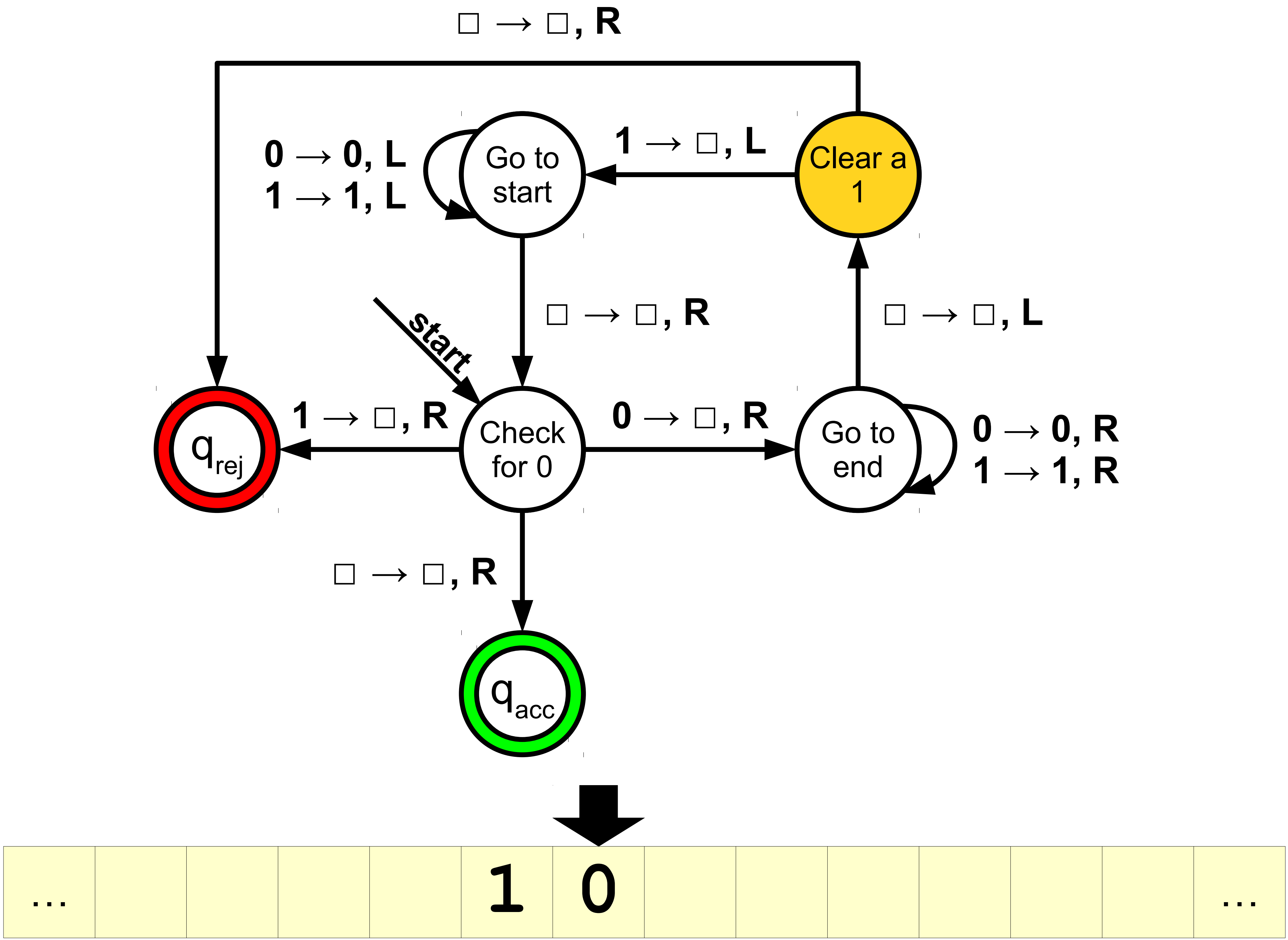
☐ → ☐, R
0 → 0, L
1 → 1, L
Go to start
1 → ☐, L
Clear a 1
☐ → ☐, R
☐ → ☐, L
start
1 → ☐, R
q_rej
Check for 0
0 → ☐, R
Go to end
0 → 0, R
1 → 1, R
☐ → ☐, R
q_acc
…
0
1
0
…


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_rej
□ → □, R
q_acc
… 1 0 …

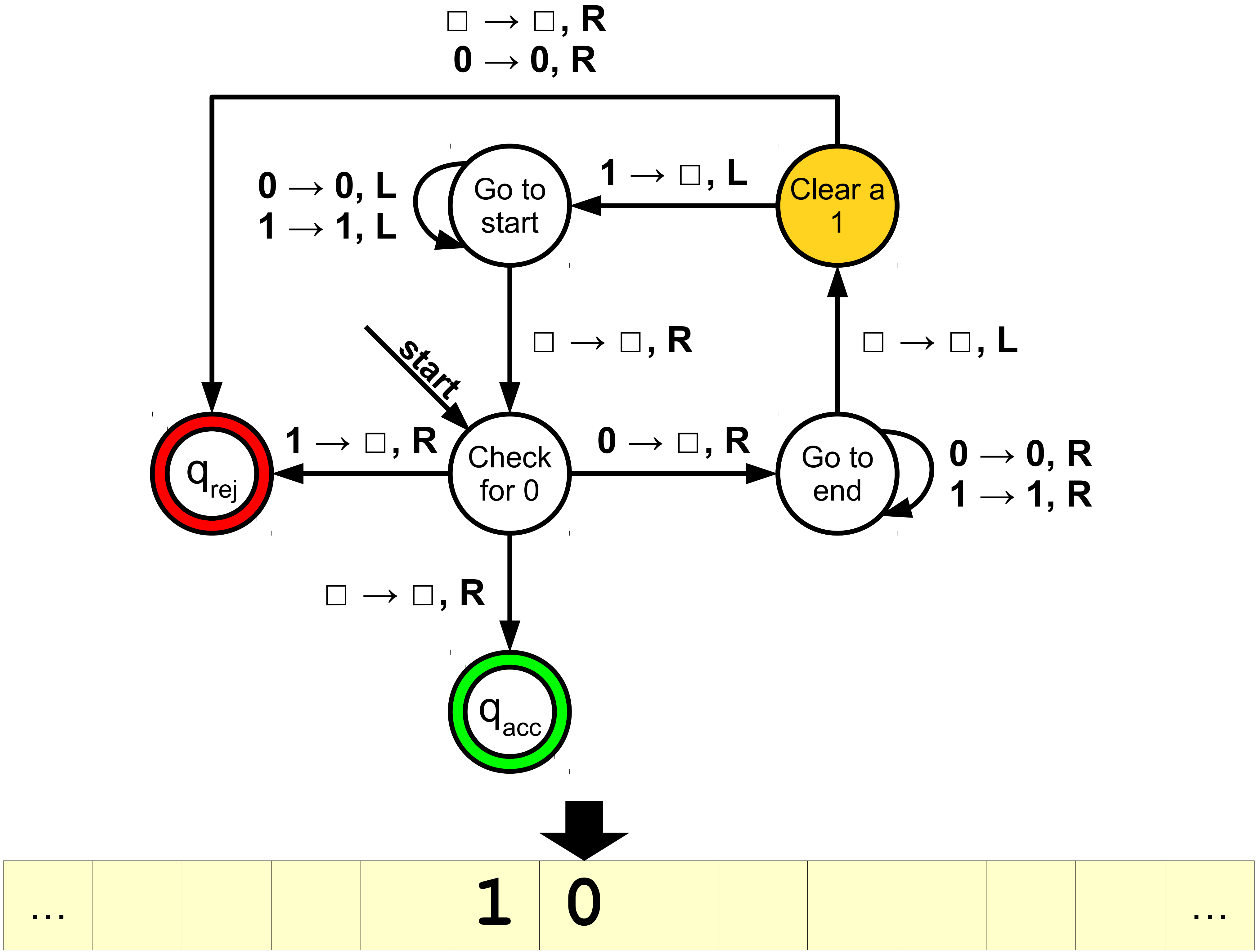


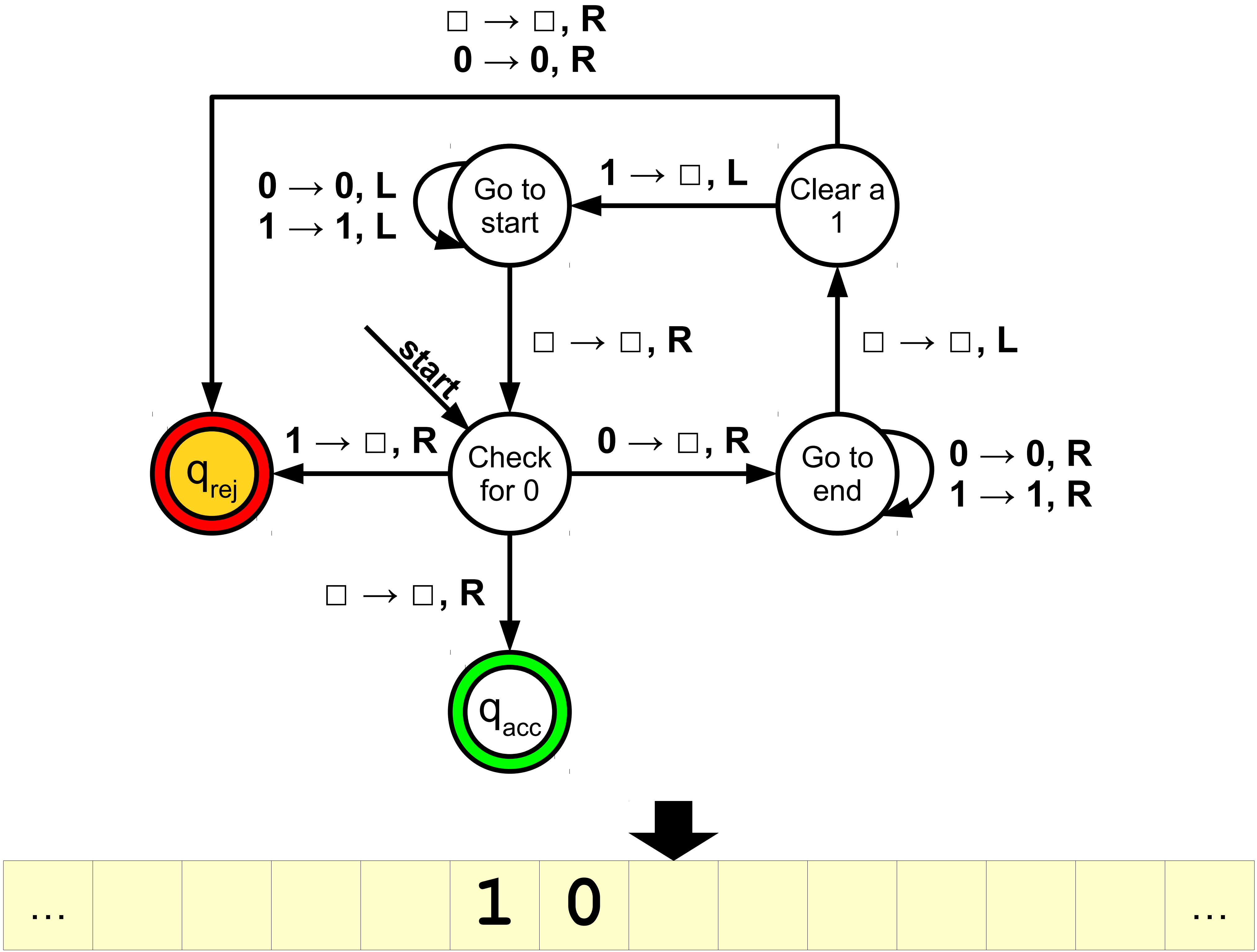
□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
0
…


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
q_{rej}
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}
…
1
0
…

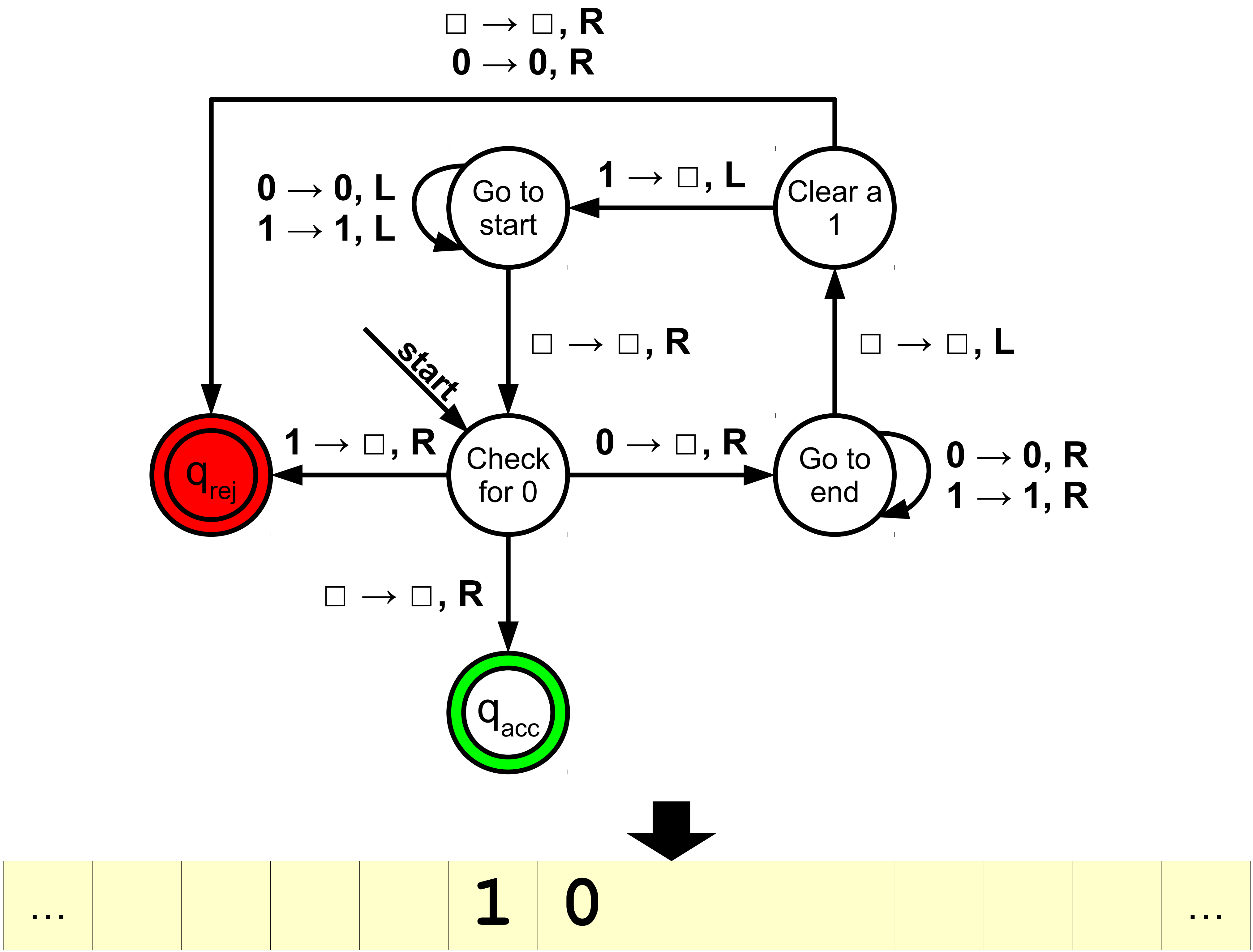


□ → □, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
… 1 0 …

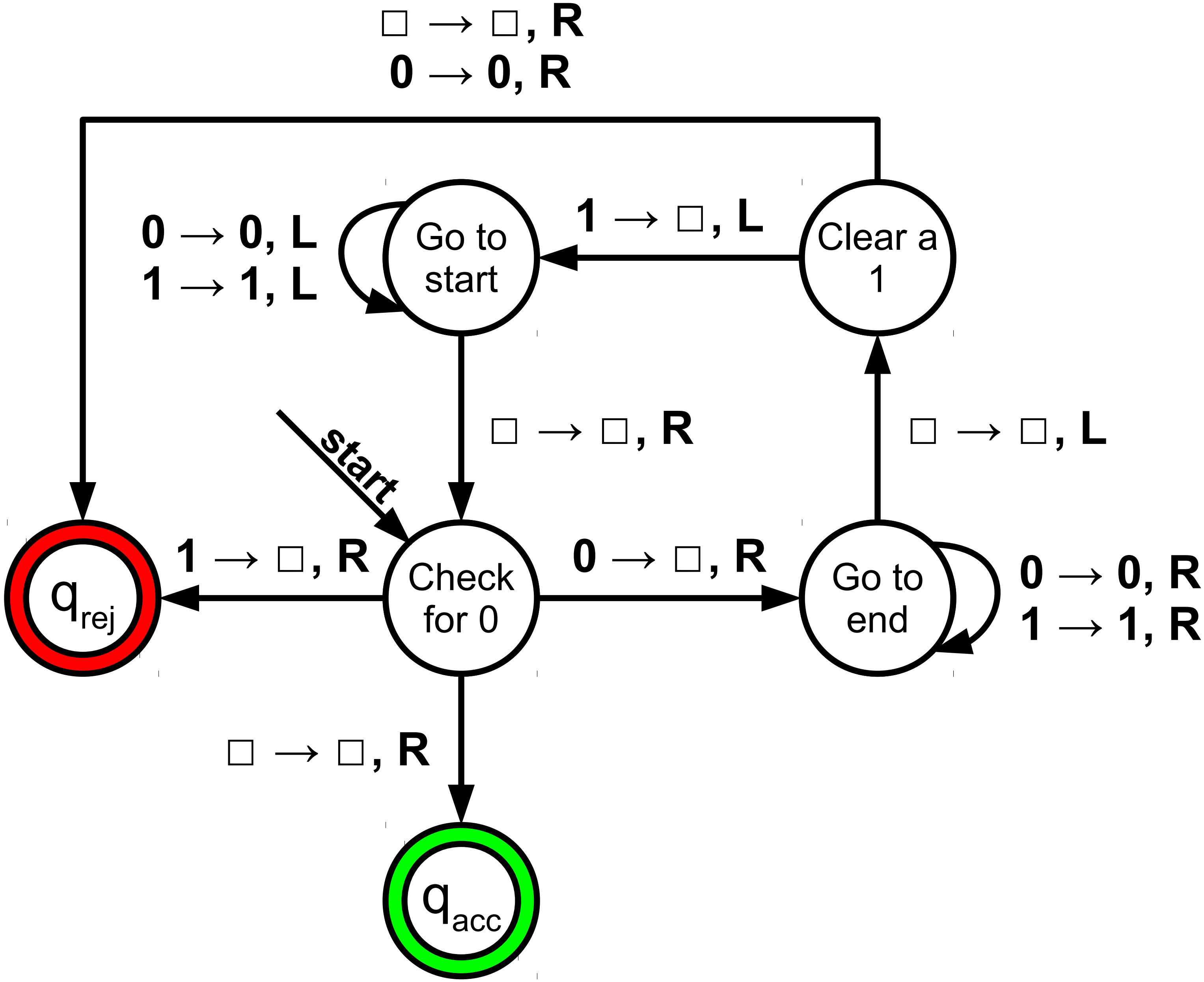

□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
0
…



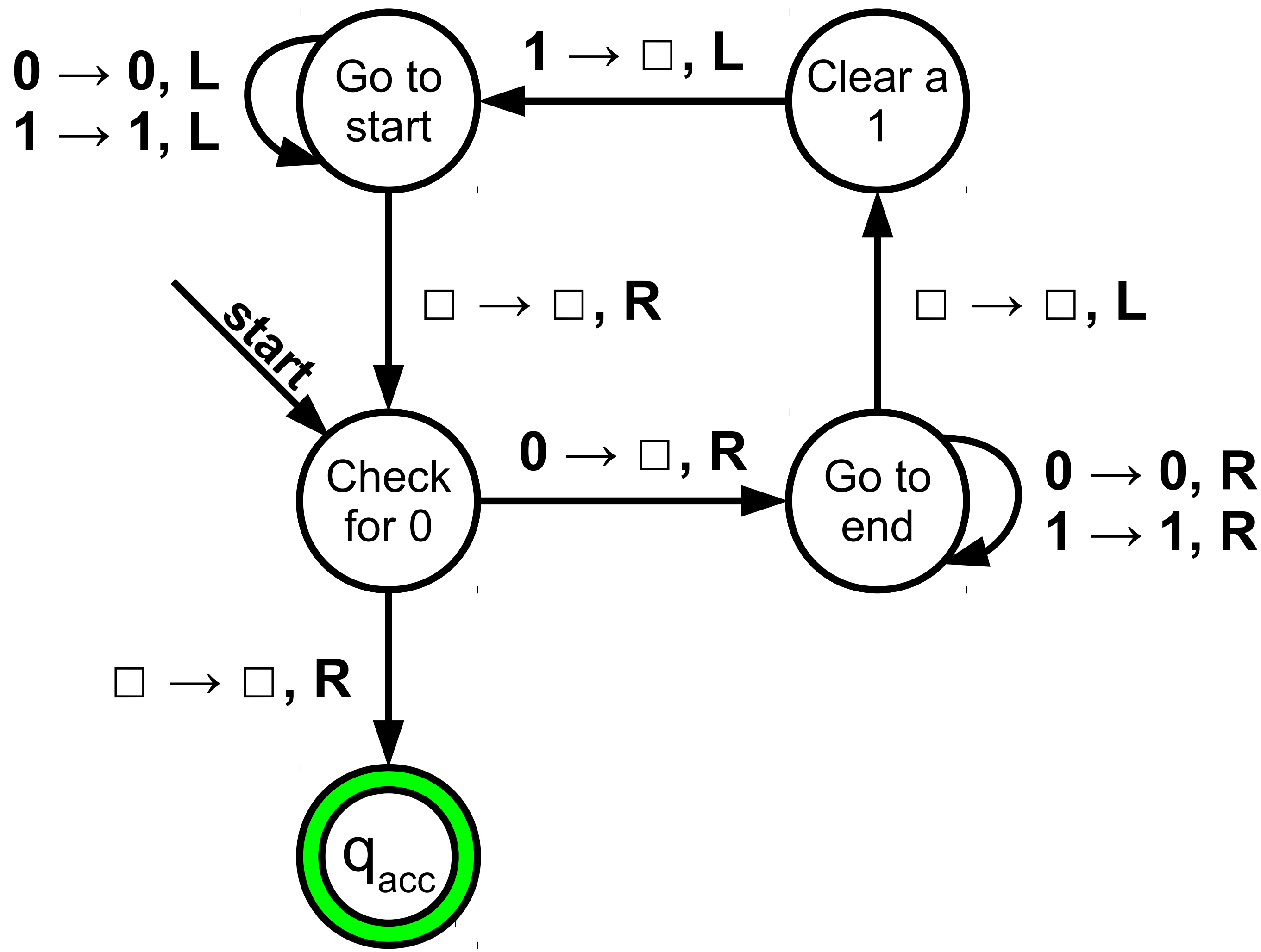
□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
q_rej
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
0
…



□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
1 → □, R
0 → □, R
q_rej
Check for 0
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc
…
1
0
…



□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
q_{rej}
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_{acc}



Go to start
Clear a 1
Check for 0
Go to end
q_acc
start
0 → 0, L
1 → 1, L
1 → □, L
□ → □, R
□ → □, L
0 → □, R
0 → 0, R
1 → 1, R
□ → □, R

Time-Out for Announcements!

Problem Sets

- Problem Set Five was due at 3:00PM today.
- Problem Set Six goes out today. It's due next Friday at 3:00PM.
- Looking ahead, the final assignment in this course, Problem Set Seven, will go out next Friday and be due on the last day of class.
  - No late days or late submissions allowed on this one, so PS6 is the last assignment you may use your late days on.
  - This last assignment will be shorter and will also serve as a review of all the topics we've covered in the course.

Final Exam

- As a reminder, the final exam for CS103 will be on Friday August 16th from 7-10 PM.
- We will release some practice final exams early next week.
- We're planning to have a sit-down practice final on Wednesday August 14th right after class from 5:30-8:30 PM. Location TBD.
  - The mock exam from this practice session will also be released online if you can't make it in person.

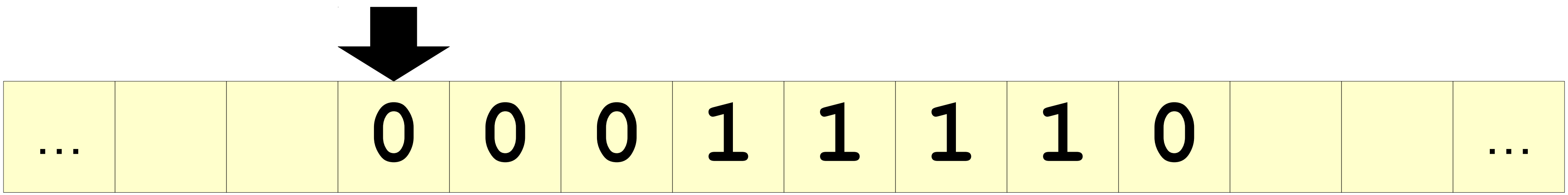
Another TM Design

- We've designed a TM for $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$.
- Consider this language over $\Sigma = \{\mathtt{0}, \mathtt{1}\}$:

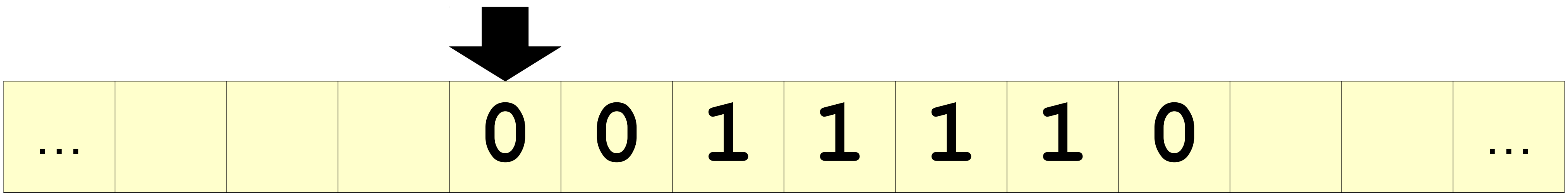
$$L = \{ w \in \Sigma^* \mid w \text{ has the same number of } \mathtt{0}\text{s and } \mathtt{1}\text{s} \}$$

- This language is also not regular, but it is context-free.
- How might we design a TM for it?

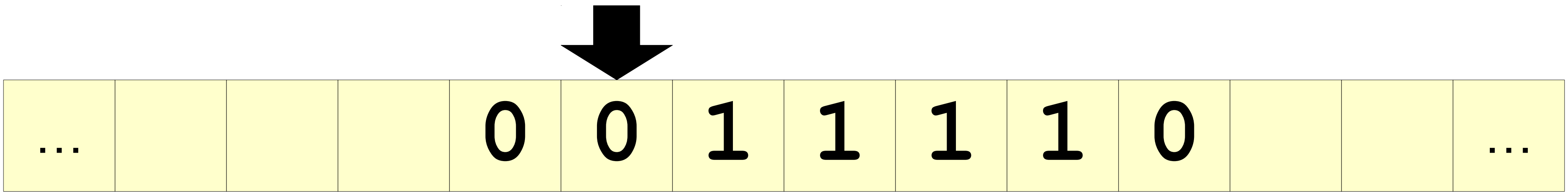
A Caveat



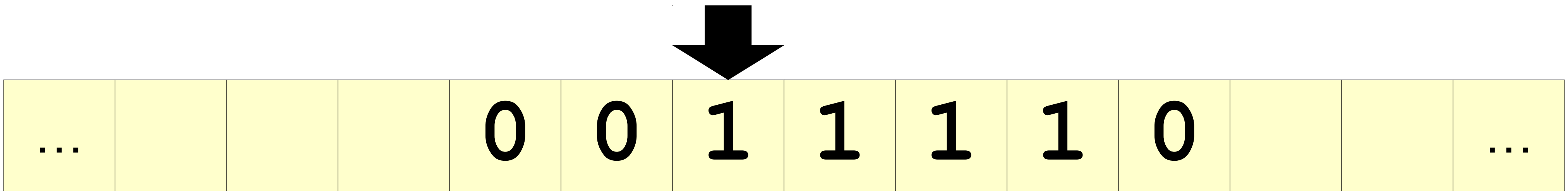
A Caveat



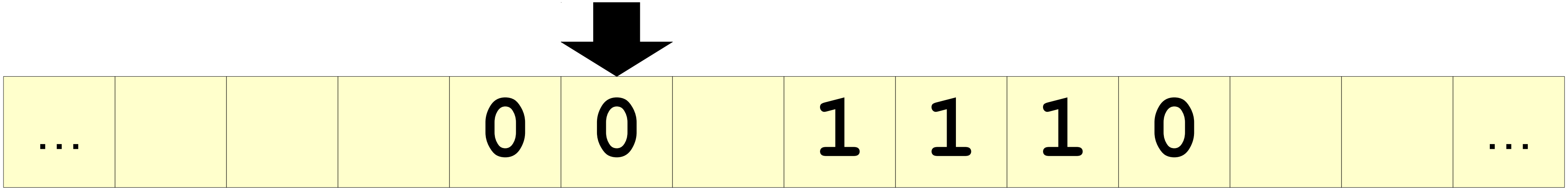
A Caveat



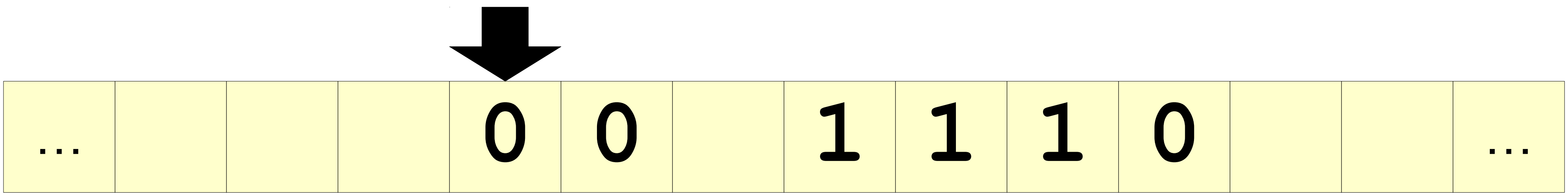
A Caveat



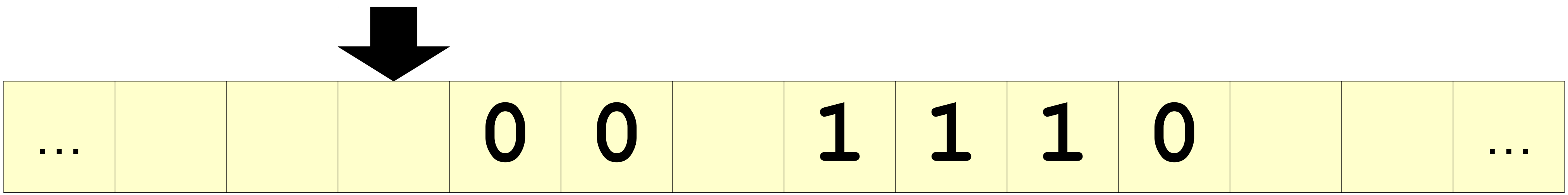
A Caveat



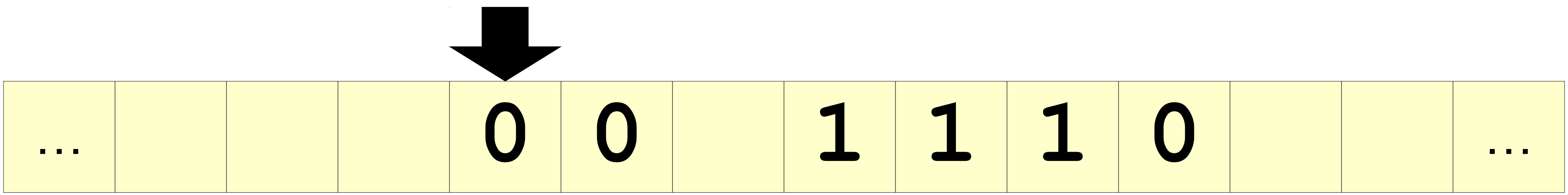
A Caveat



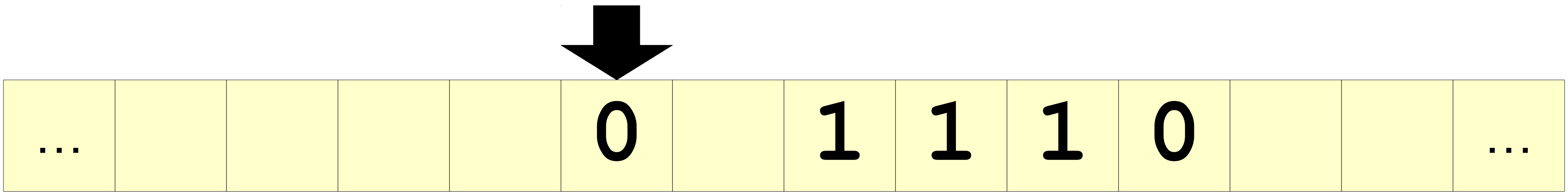
A Caveat



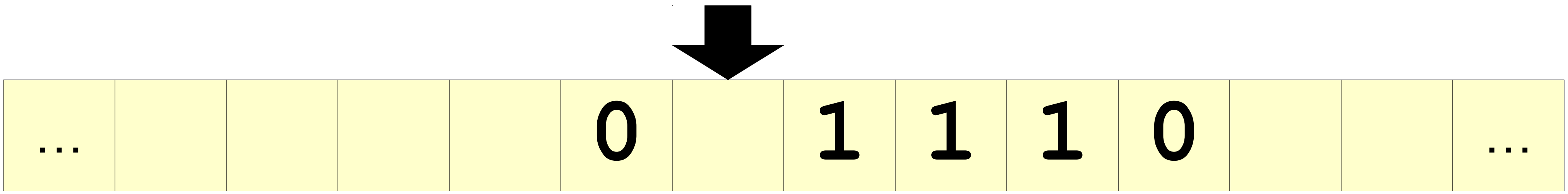
A Caveat



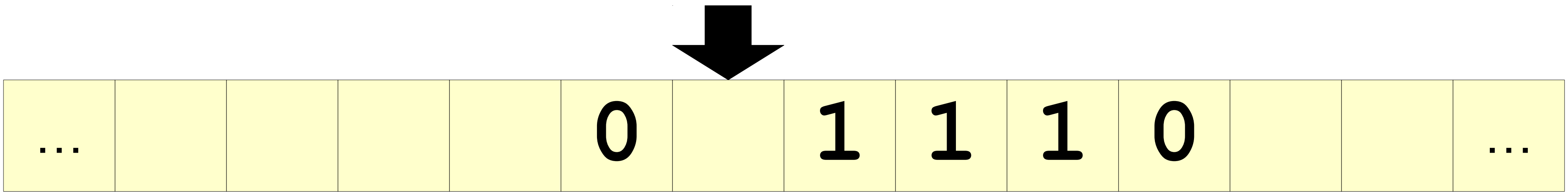
A Caveat



A Caveat

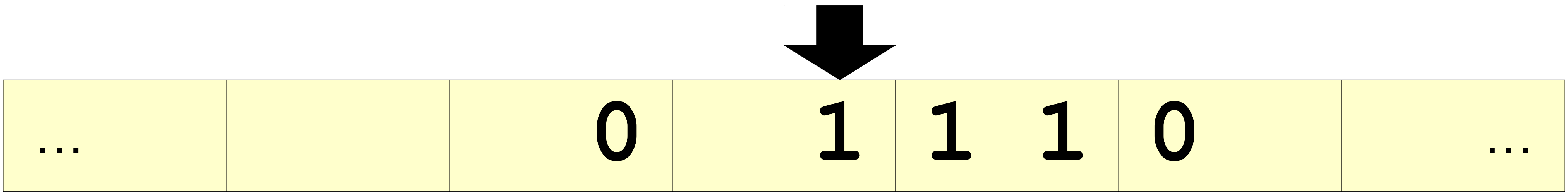


A Caveat

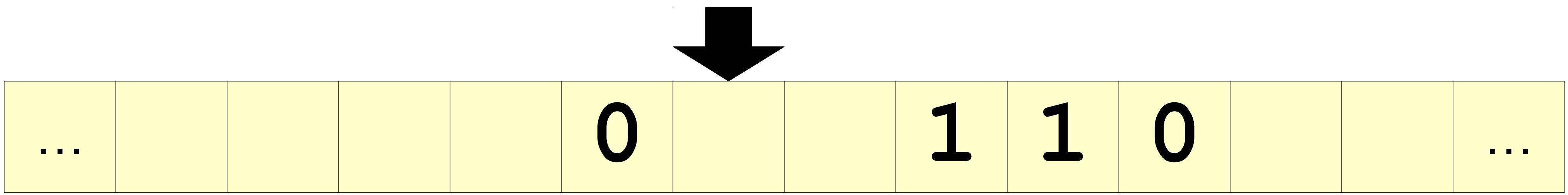


How do we know that this blank isn't one of the infinitely many blanks after our input string?

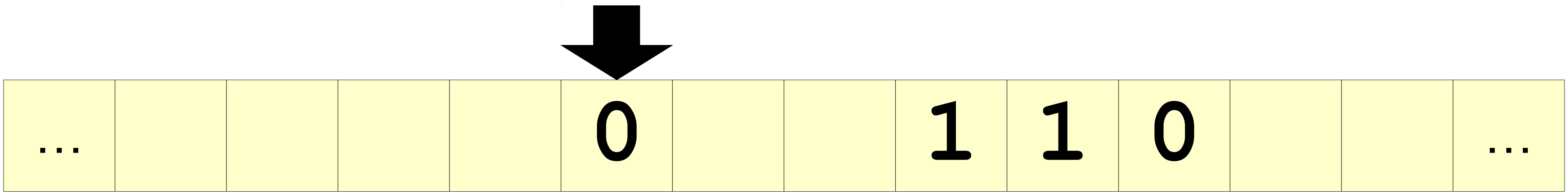
A Caveat



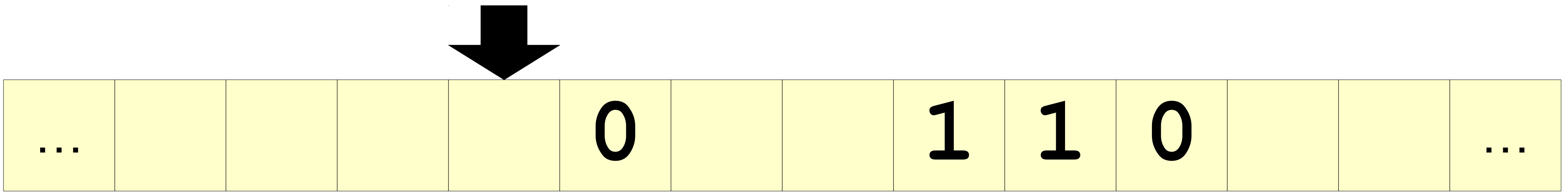
A Caveat



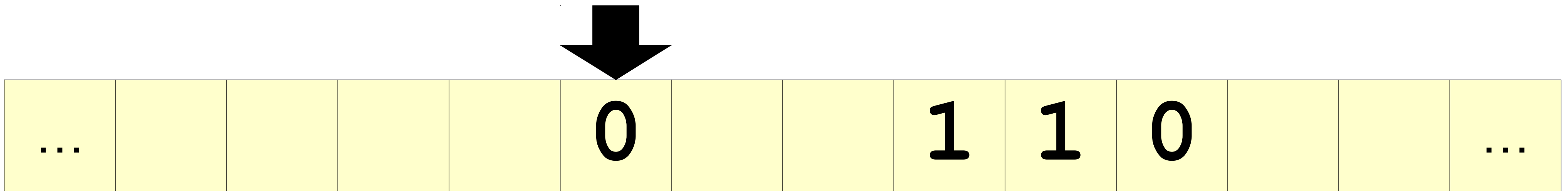
A Caveat



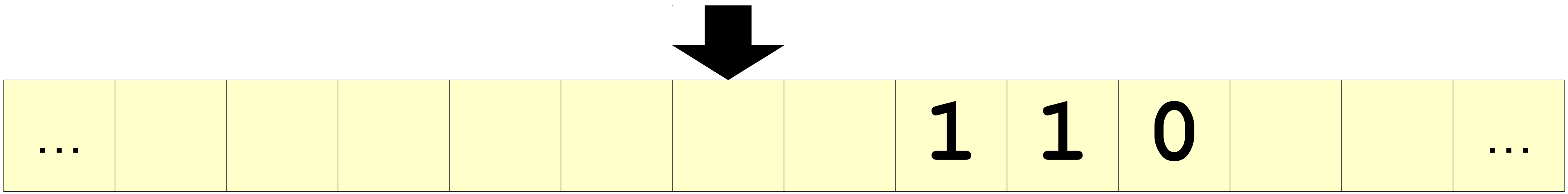
A Caveat



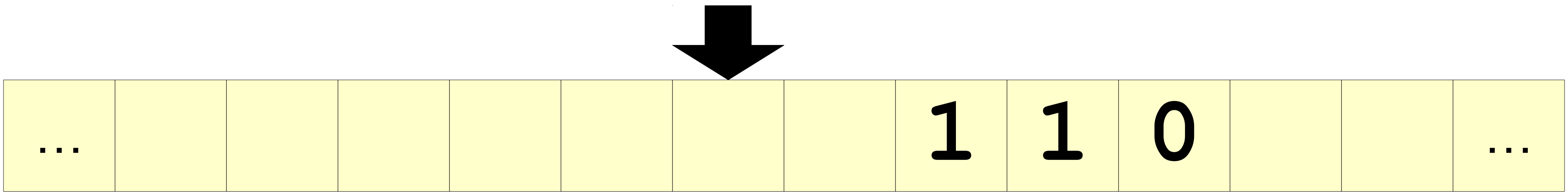
A Caveat



A Caveat

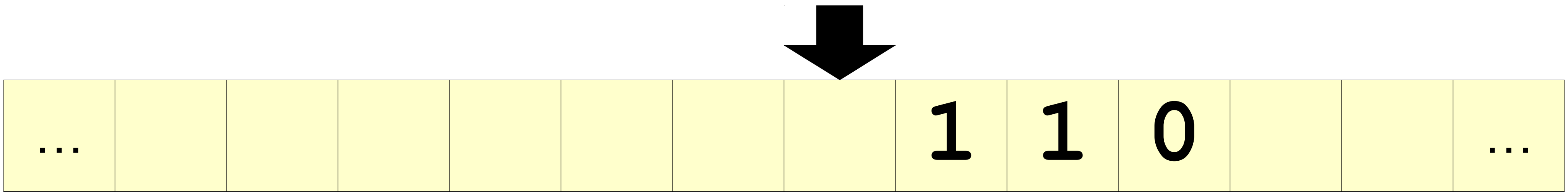


A Caveat

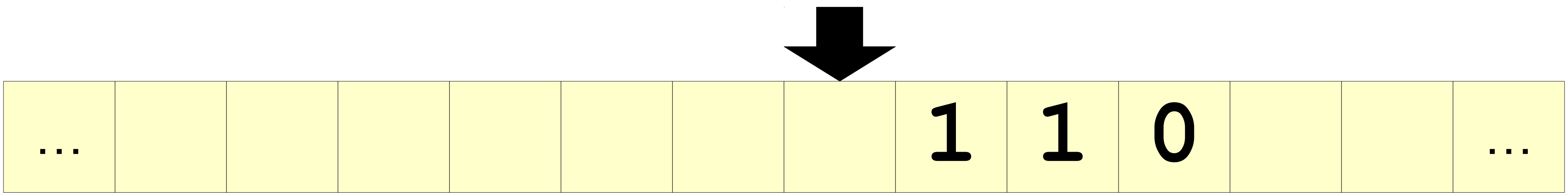


How do we know that this blank isn't one of the infinitely many blanks after our input string?

A Caveat

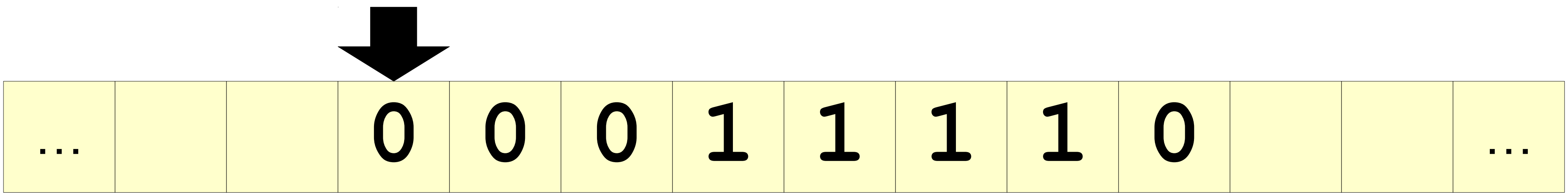


A Caveat

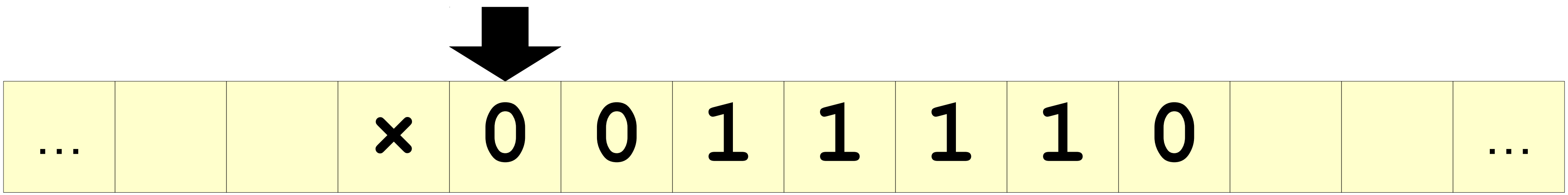


How do we know that this blank isn't one of the infinitely many blanks after our input string?

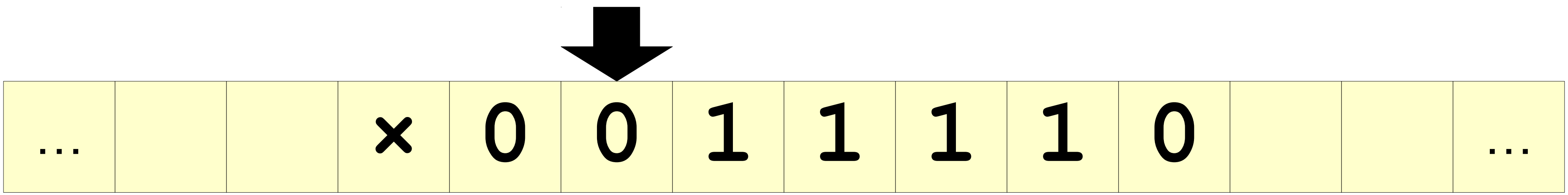
One Solution



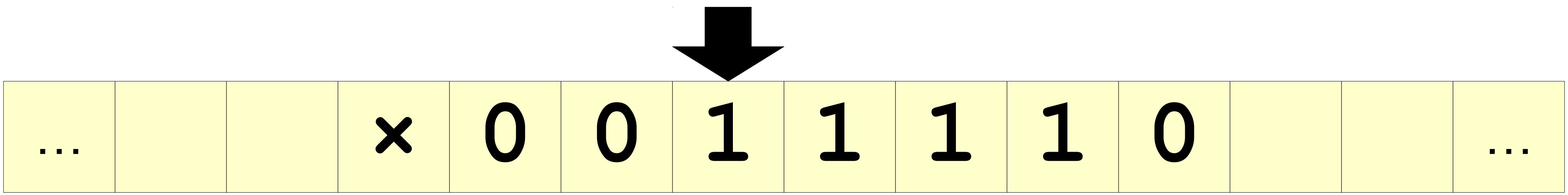
One Solution



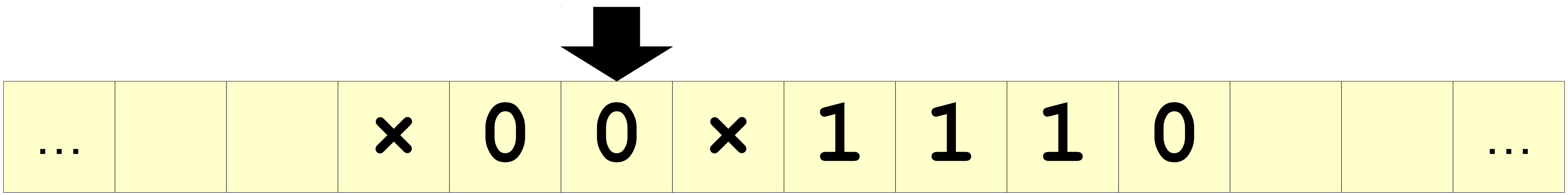
One Solution



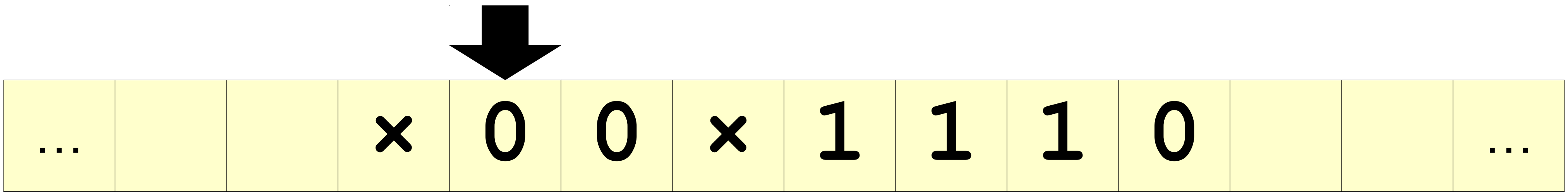
One Solution



One Solution



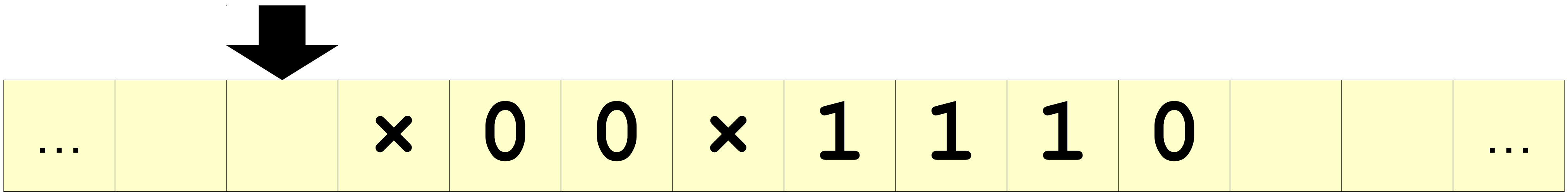
One Solution



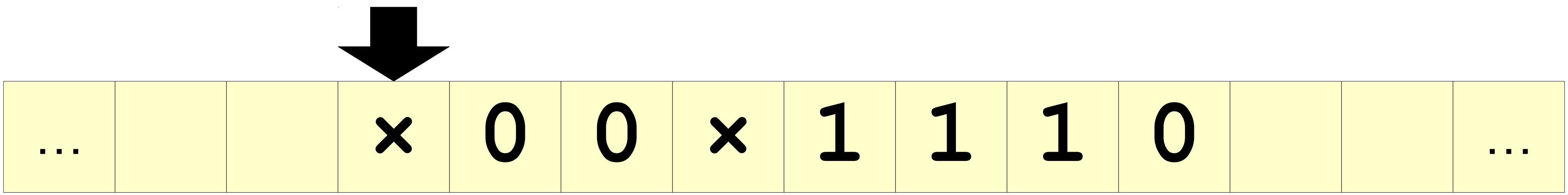
One Solution

…			×	0	0	×	1	1	1	0			…

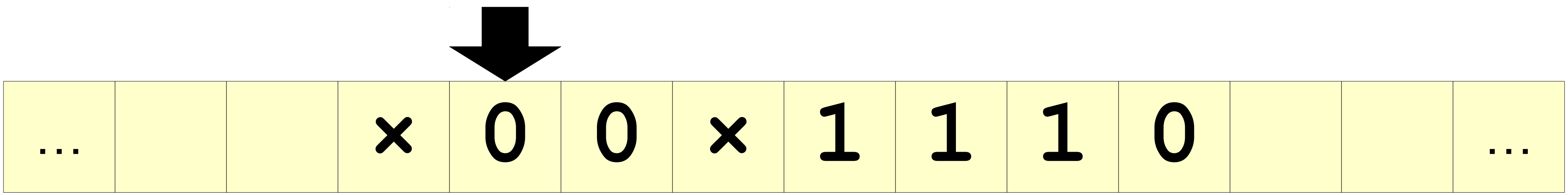
One Solution



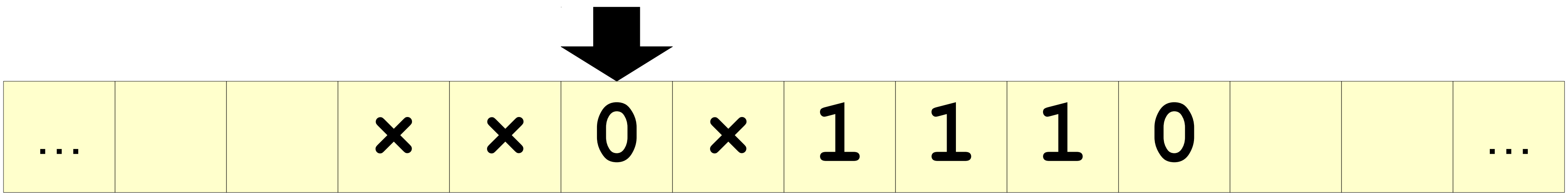
One Solution



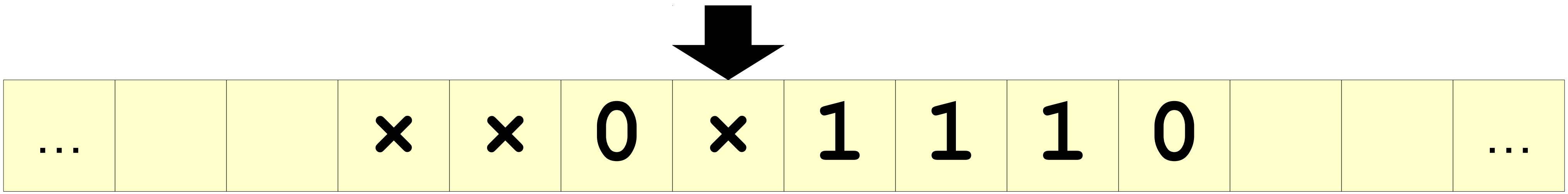
One Solution



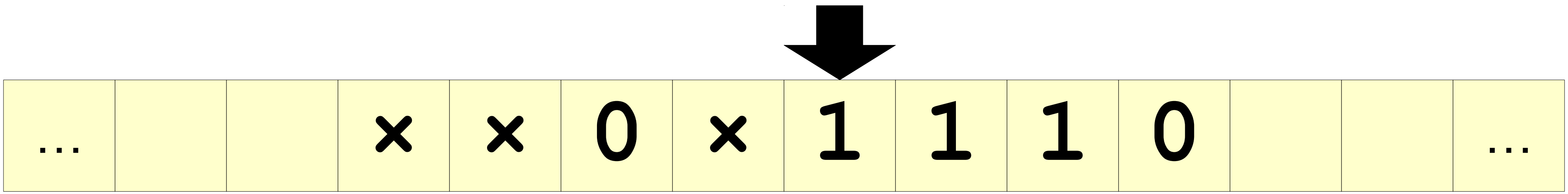
One Solution



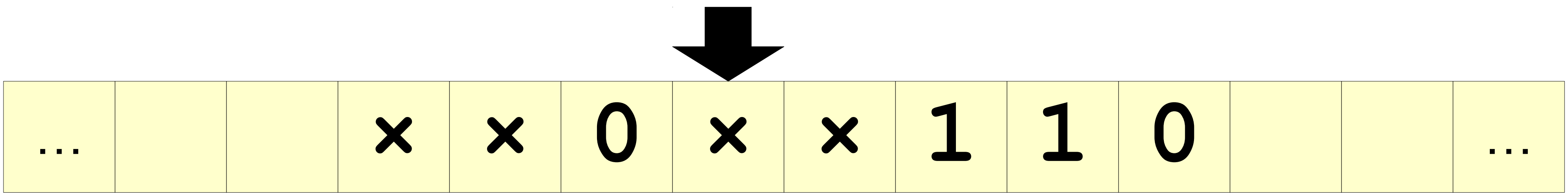
One Solution



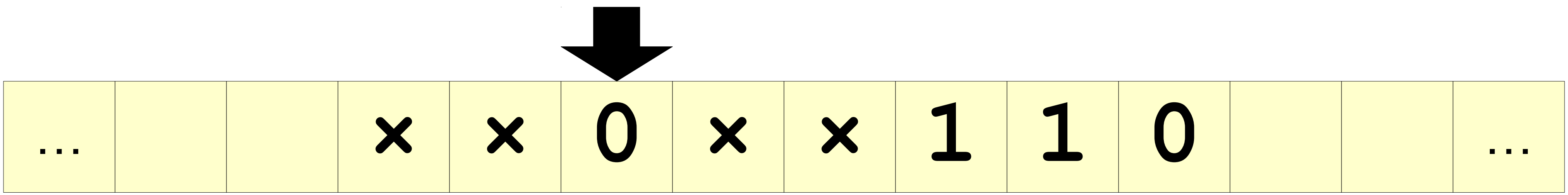
One Solution



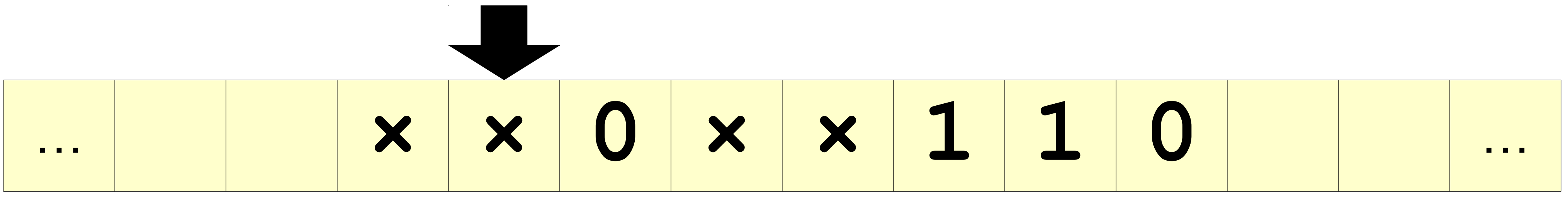
One Solution



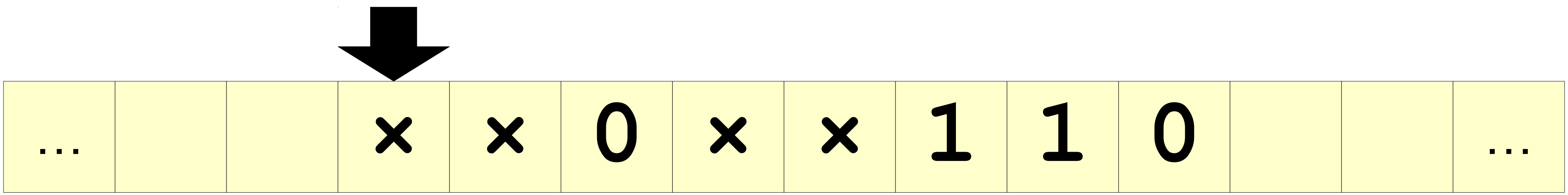
One Solution



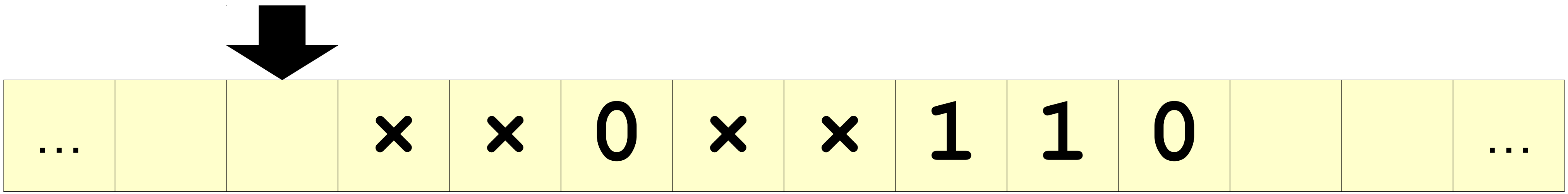
One Solution



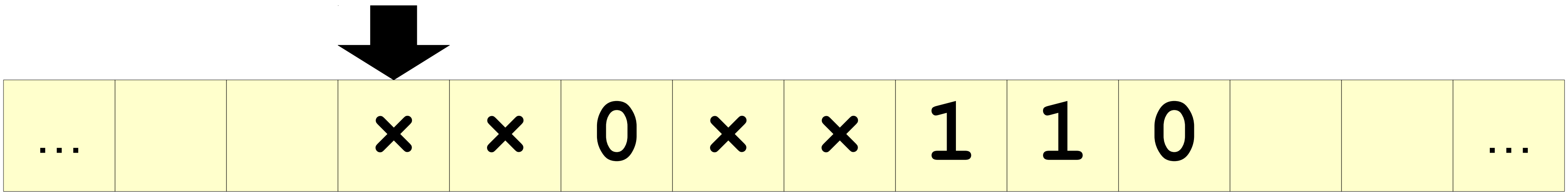
One Solution



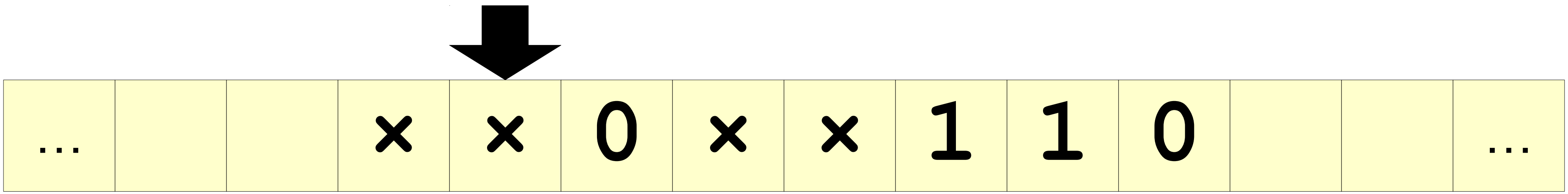
One Solution



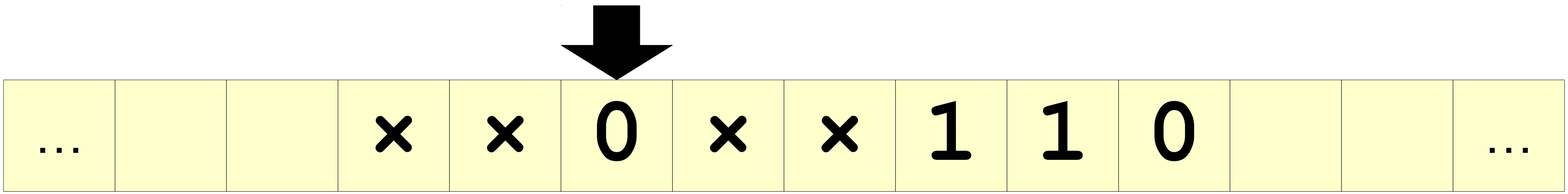
One Solution



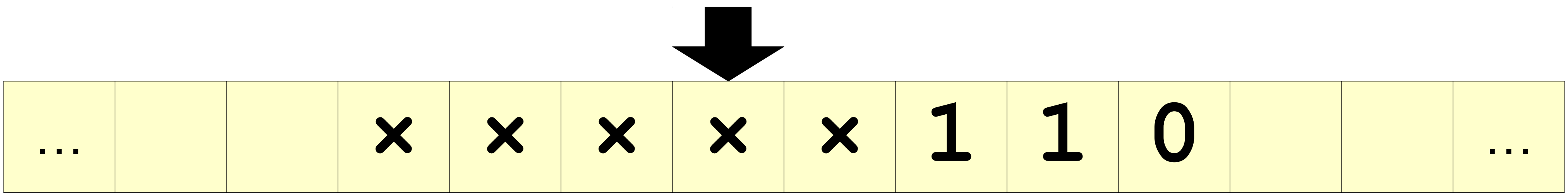
One Solution



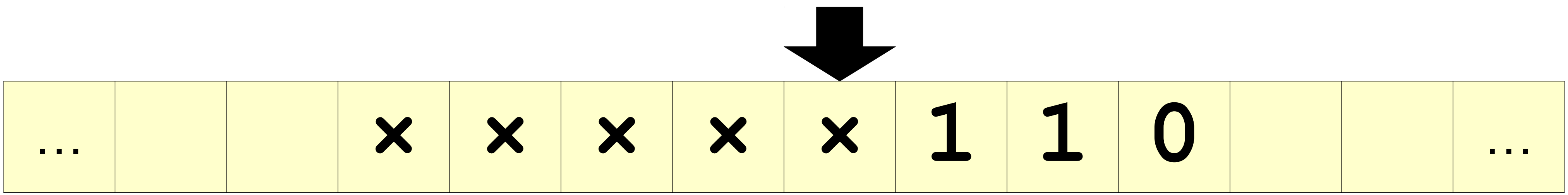
One Solution



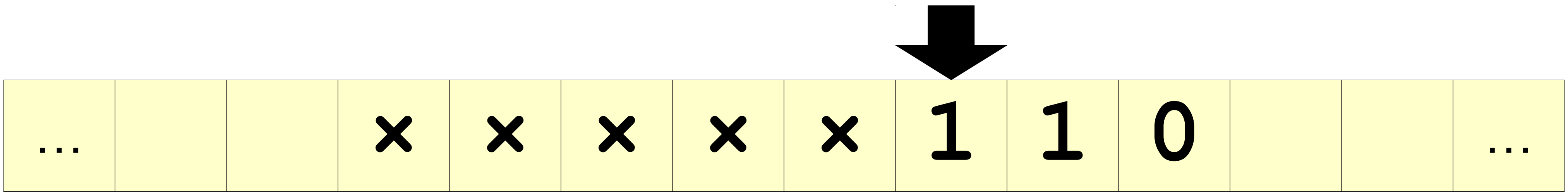
One Solution



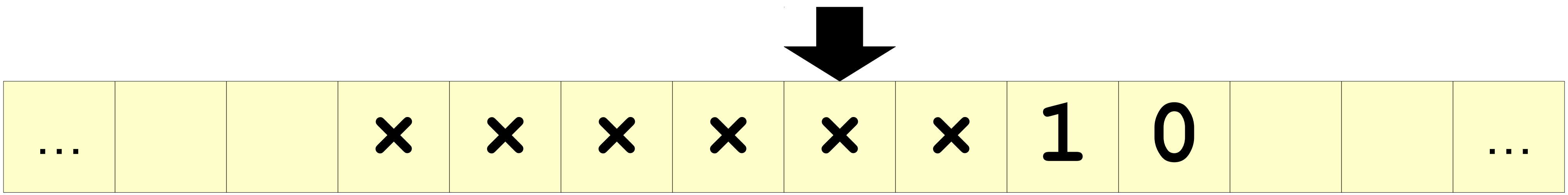
One Solution



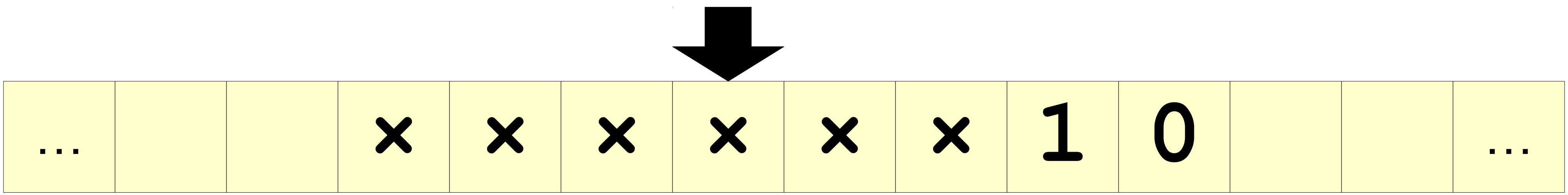
One Solution

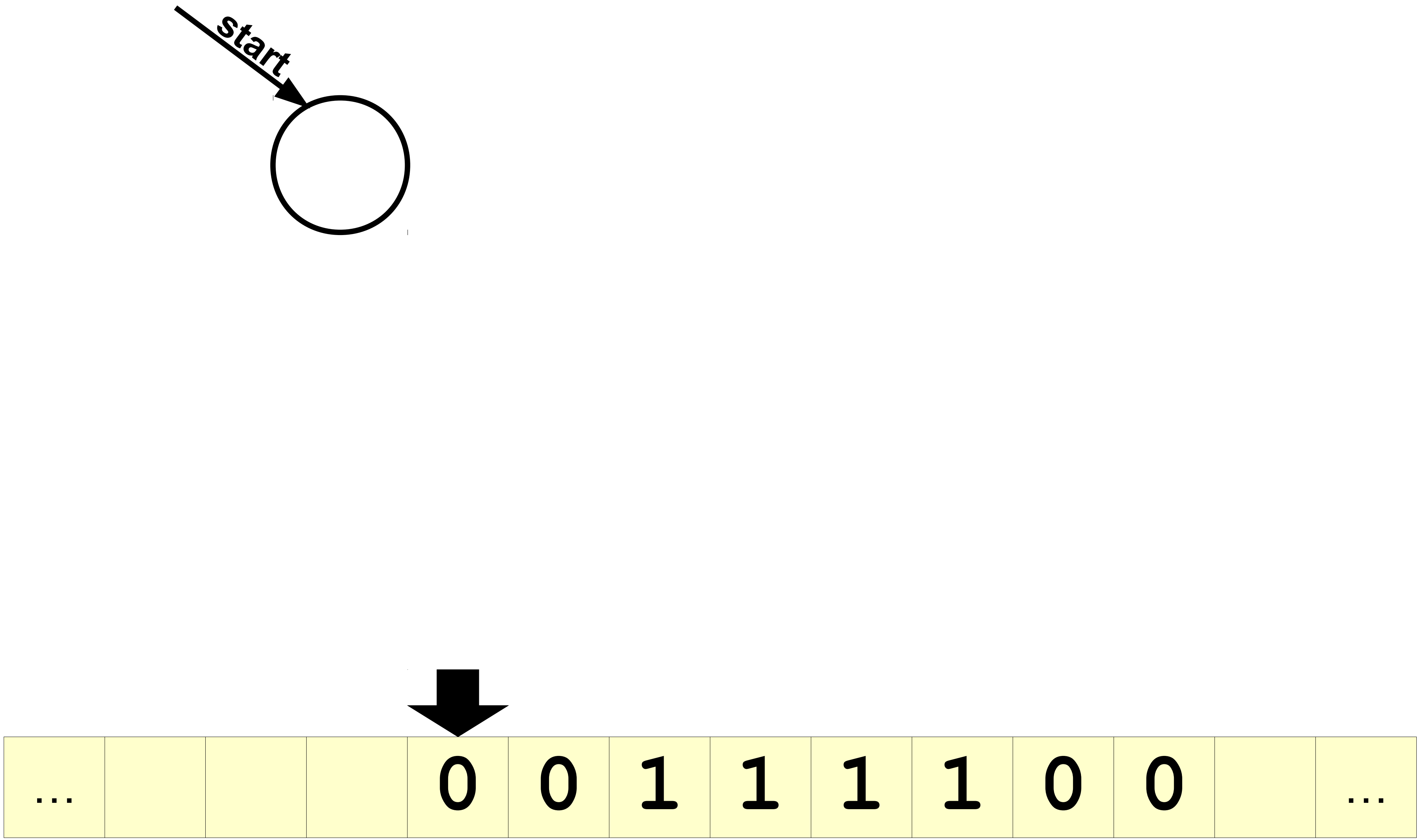


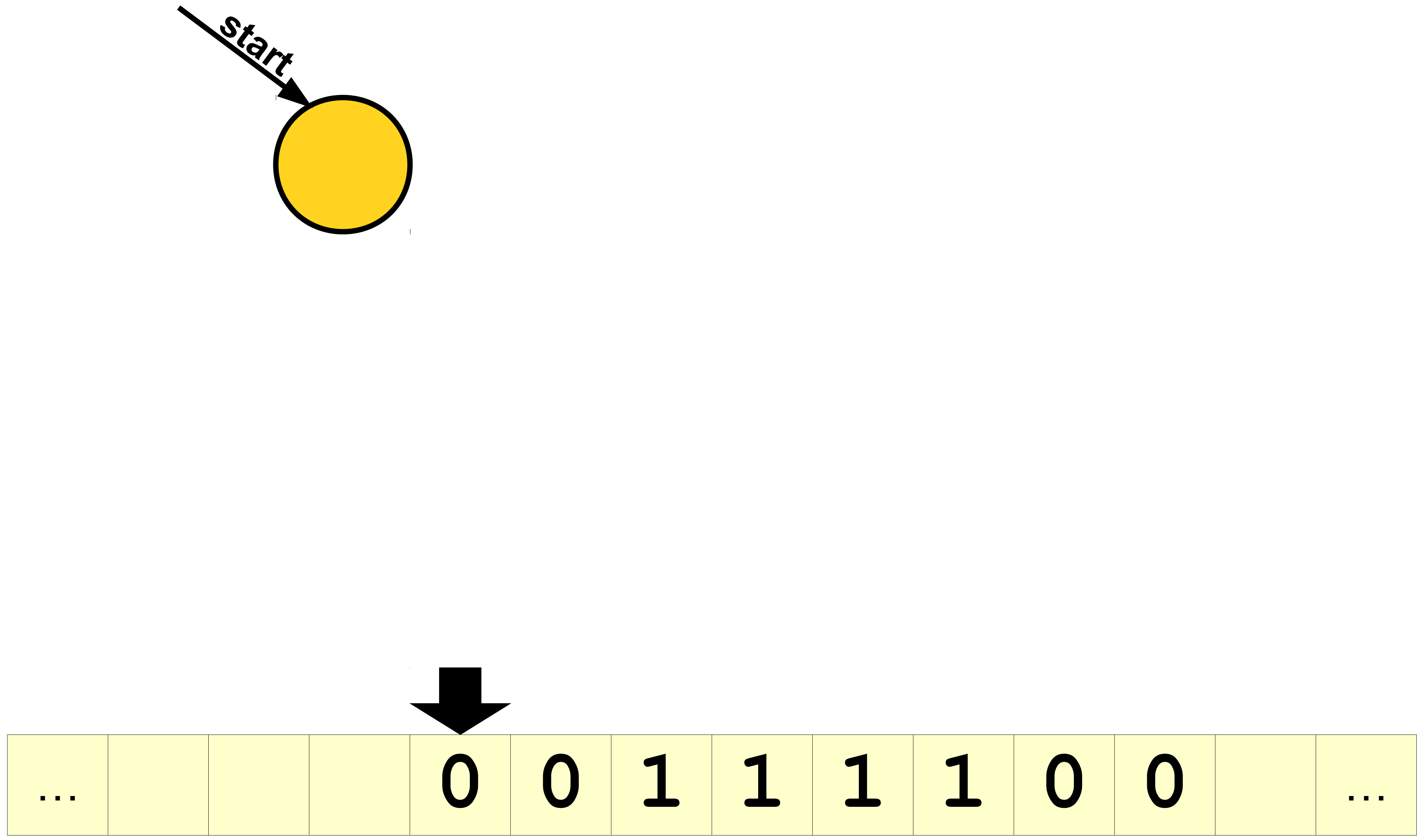
One Solution



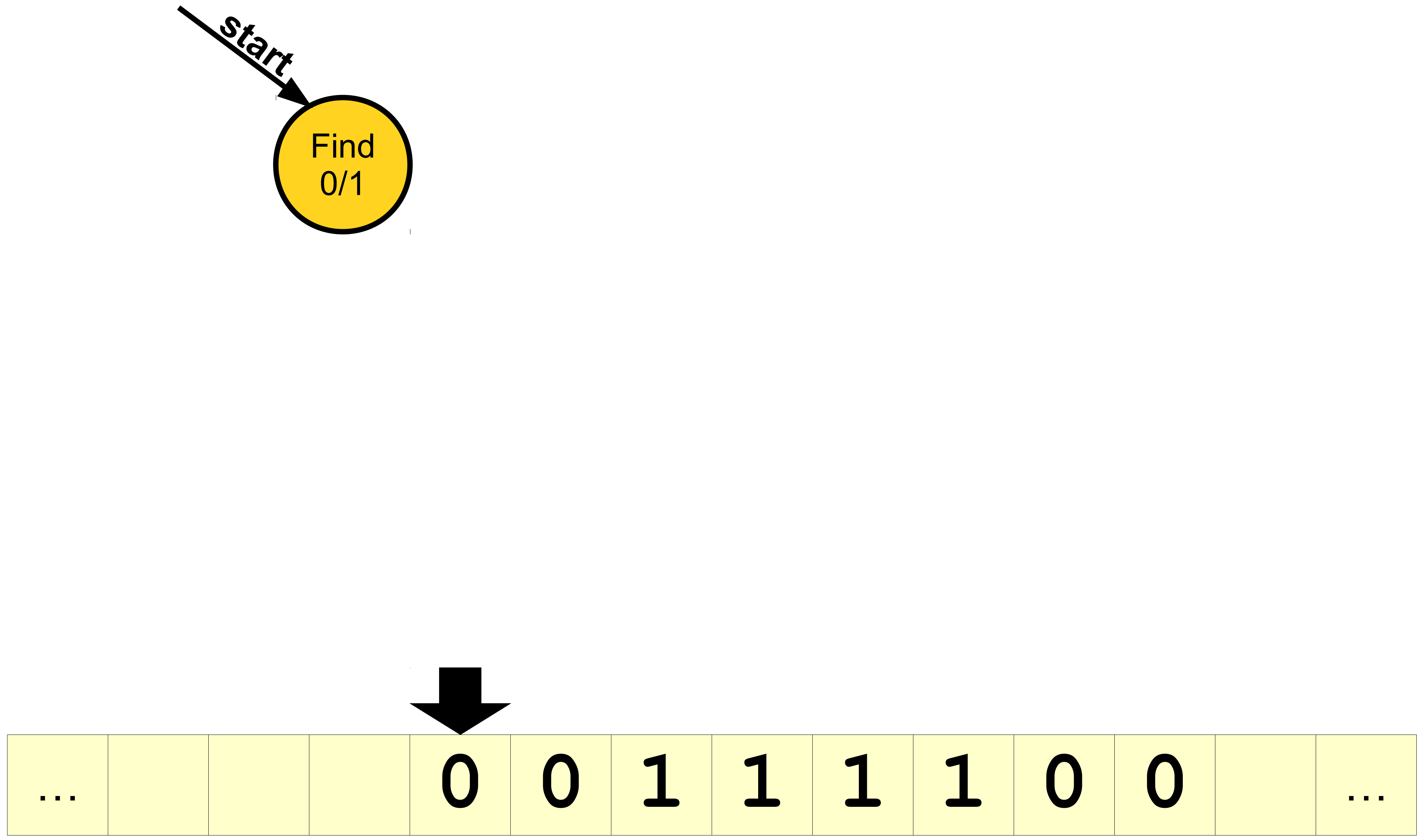
One Solution

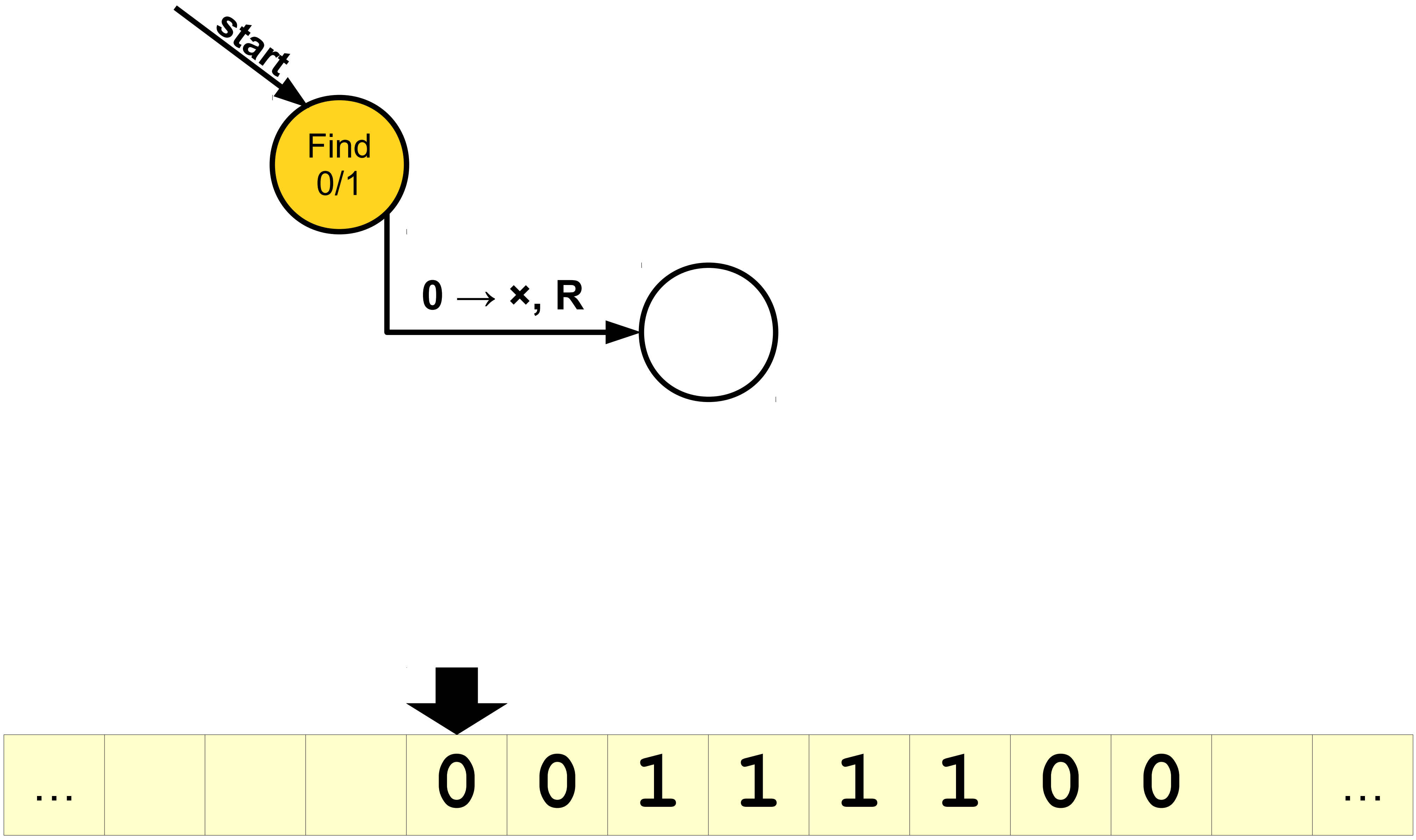


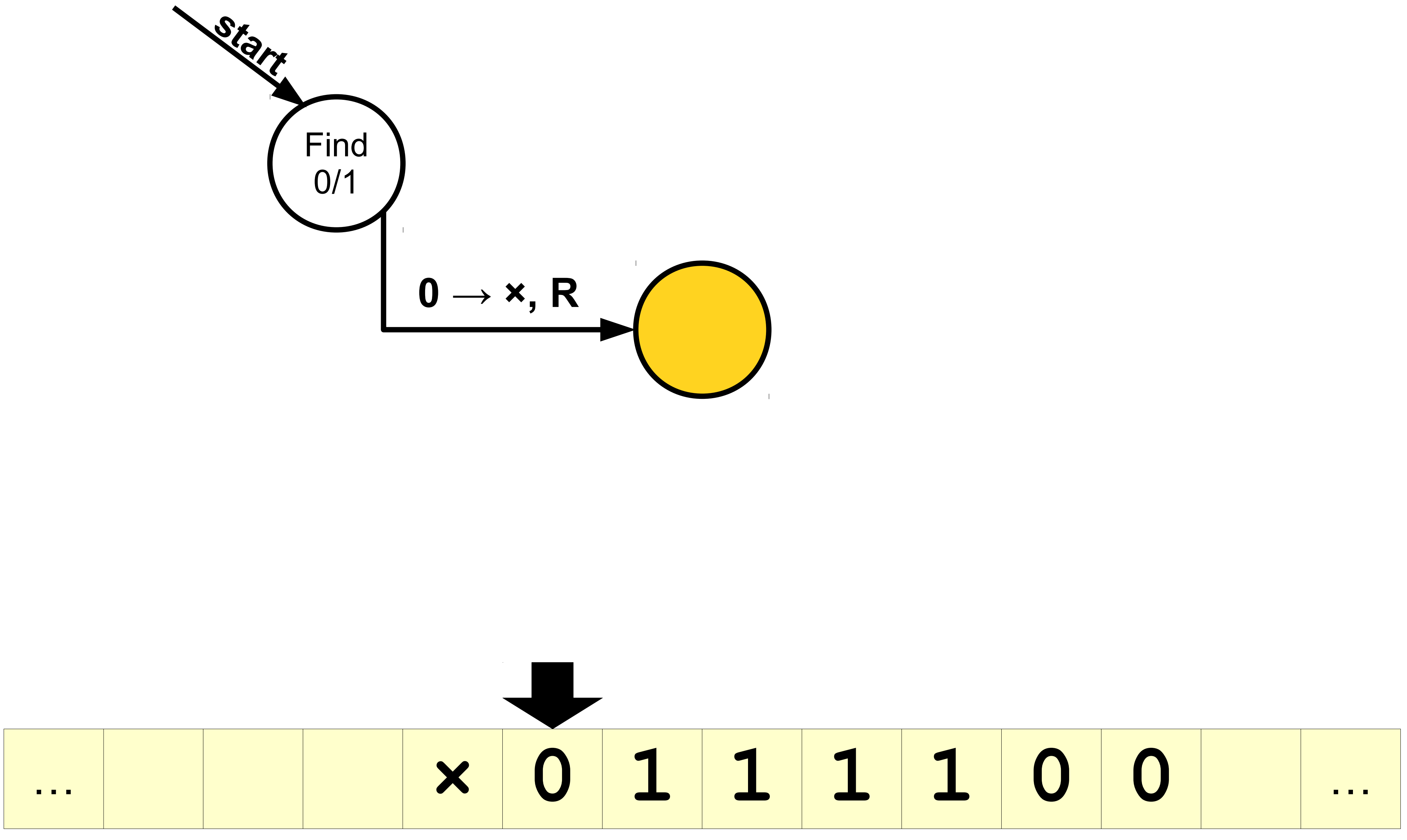

start
…
0
0
1
1
1
1
0
0
…

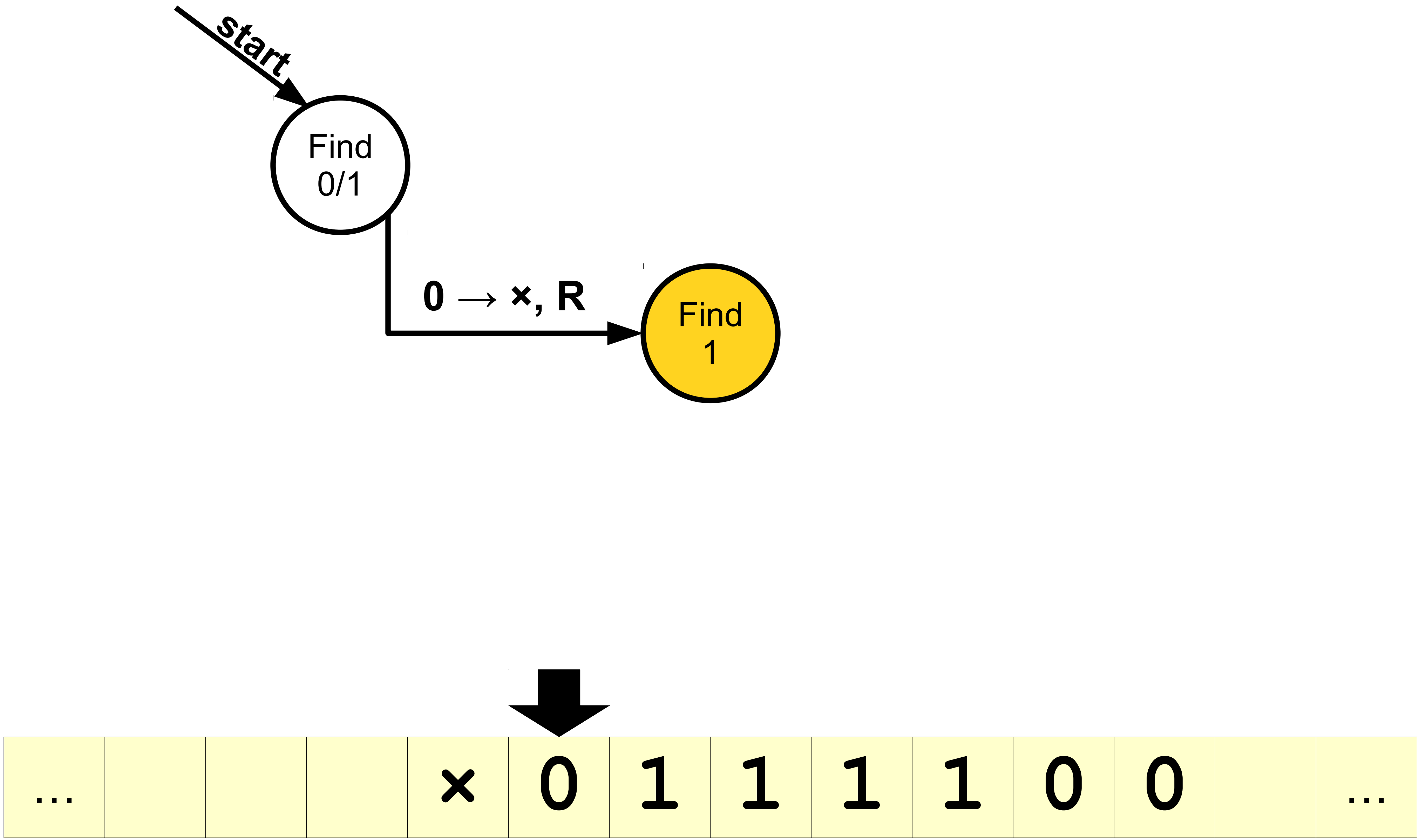


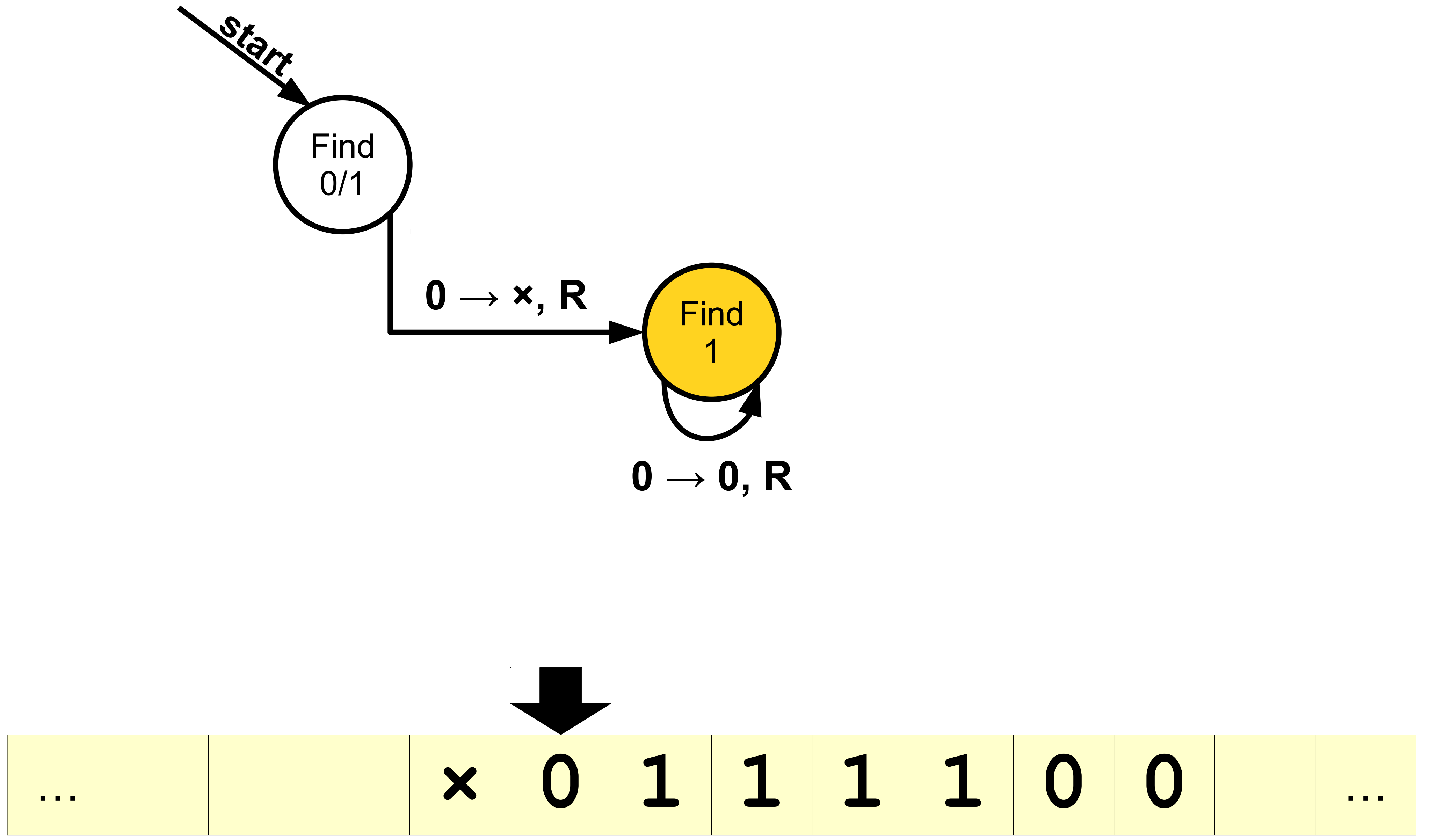
start
...
0
0
1
1
1
1
0
0
...

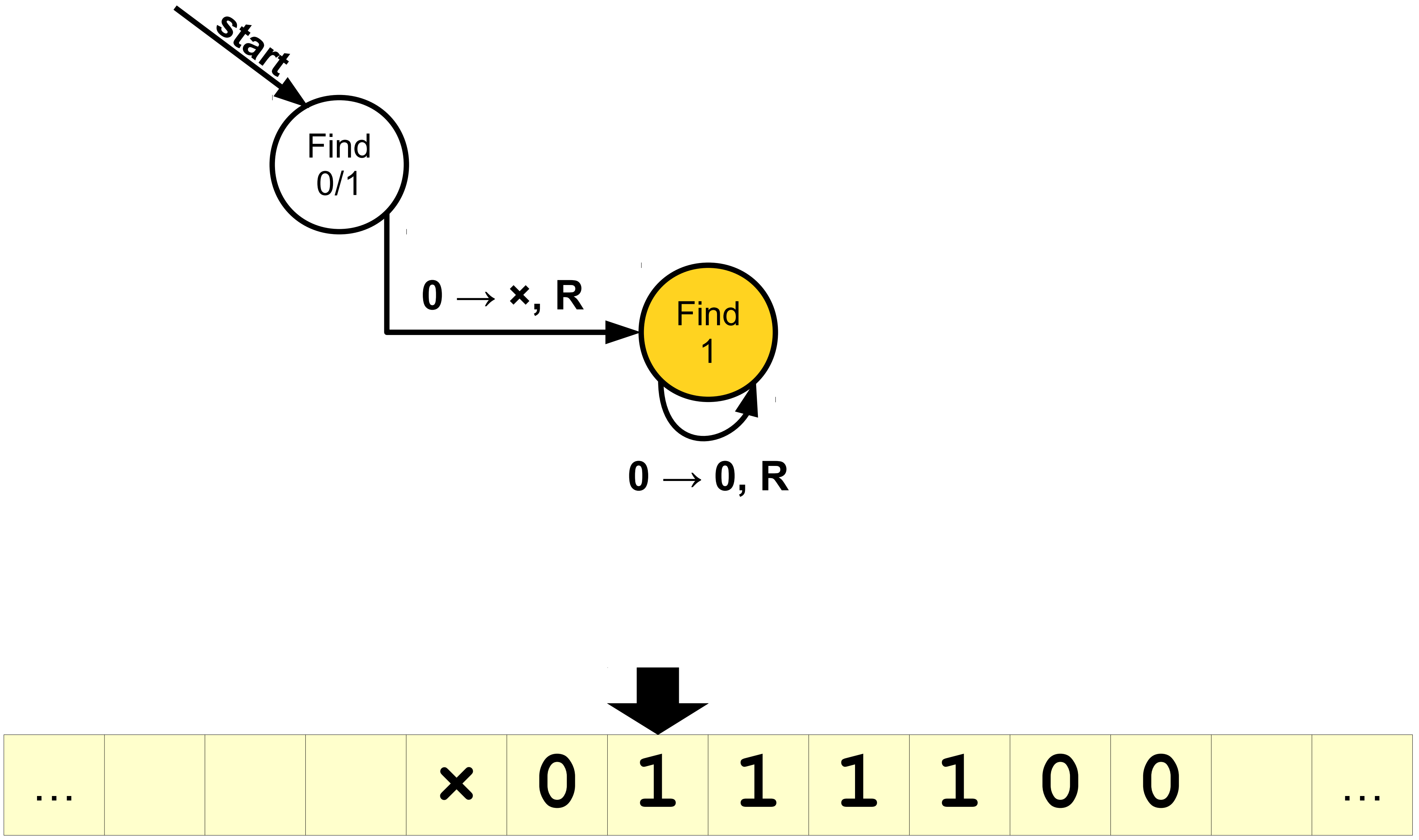

start
Find
0/1
...
0 0 1 1 1 1 0 0
...

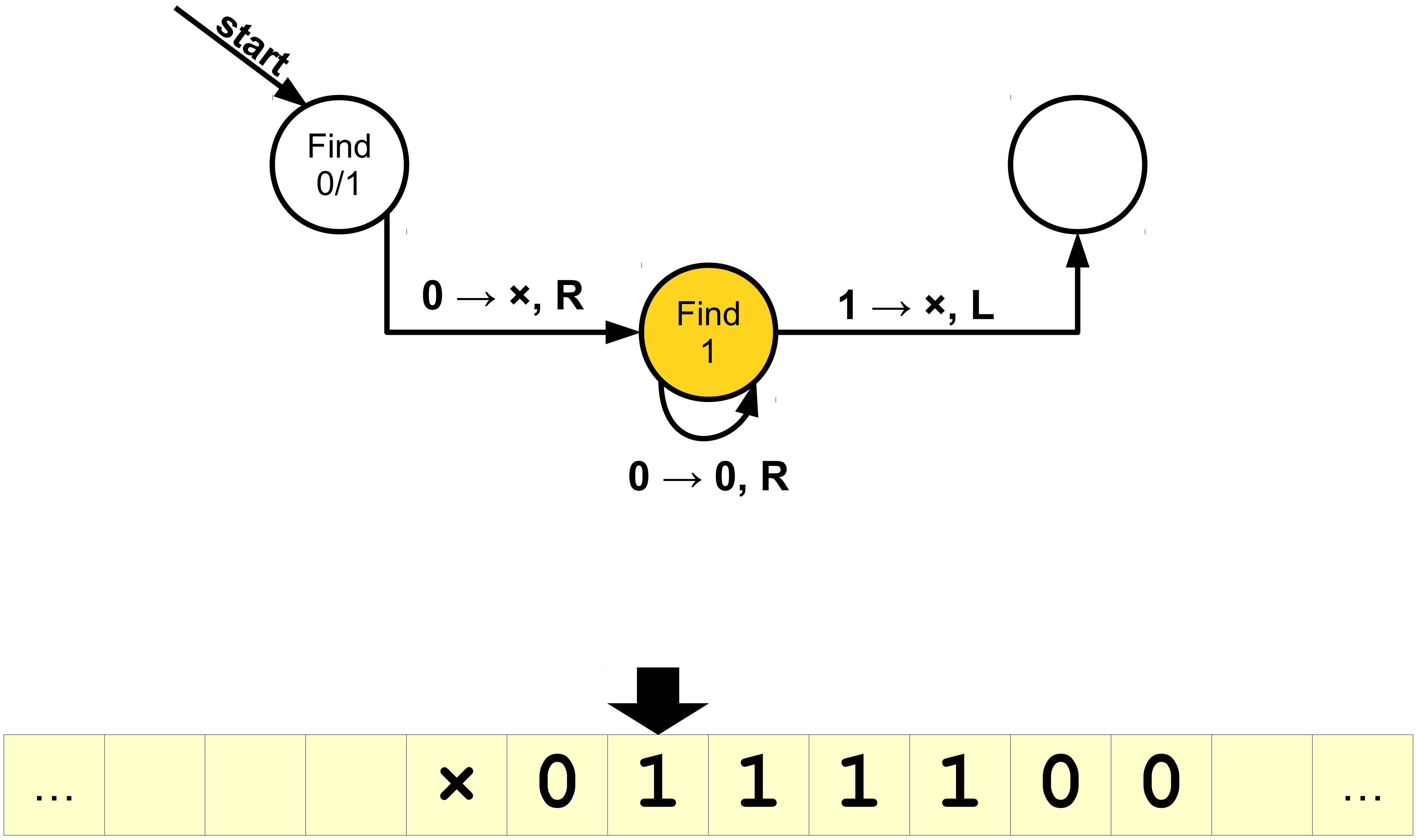

start
Find 0/1
0 → ×, R
...
0
0
1
1
1
1
0
0
...

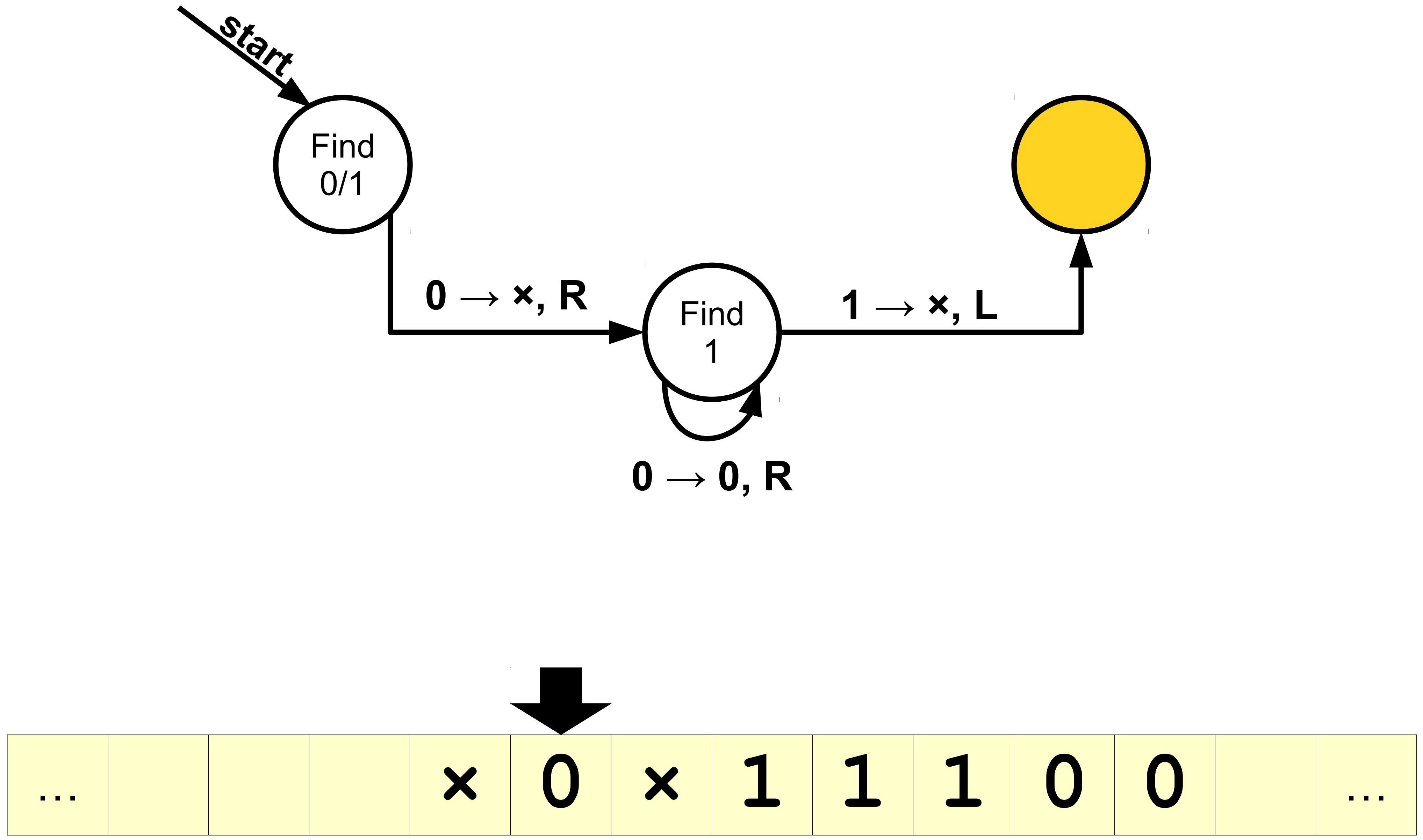

start
Find
0/1
0 → ×, R
…
×
0
1
1
1
1
0
0
…


start
Find 0/1
0 → ×, R
Find 1
...
×
0
1
1
1
1
0
0
...

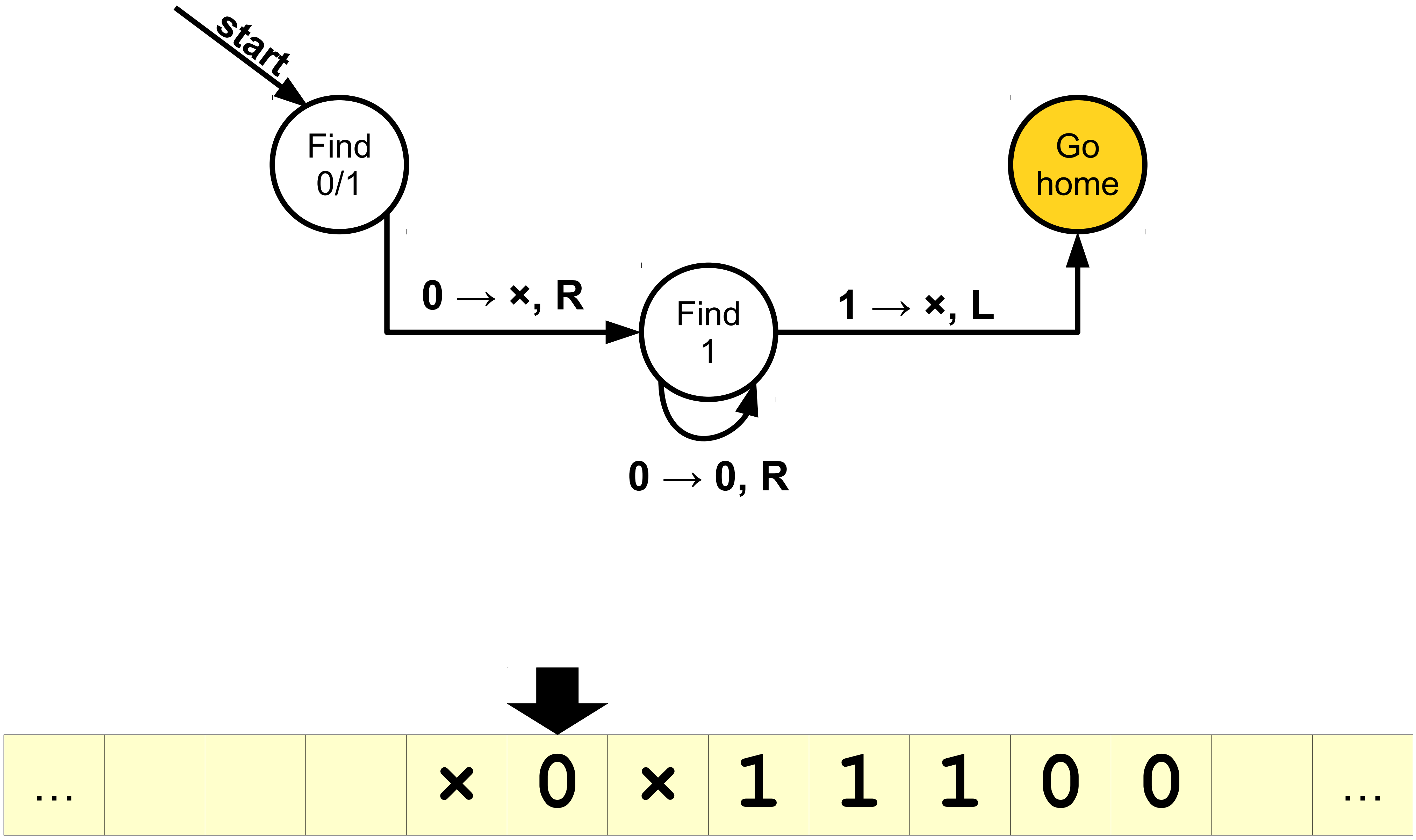

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
…
×
0
1
1
1
1
0
0
…

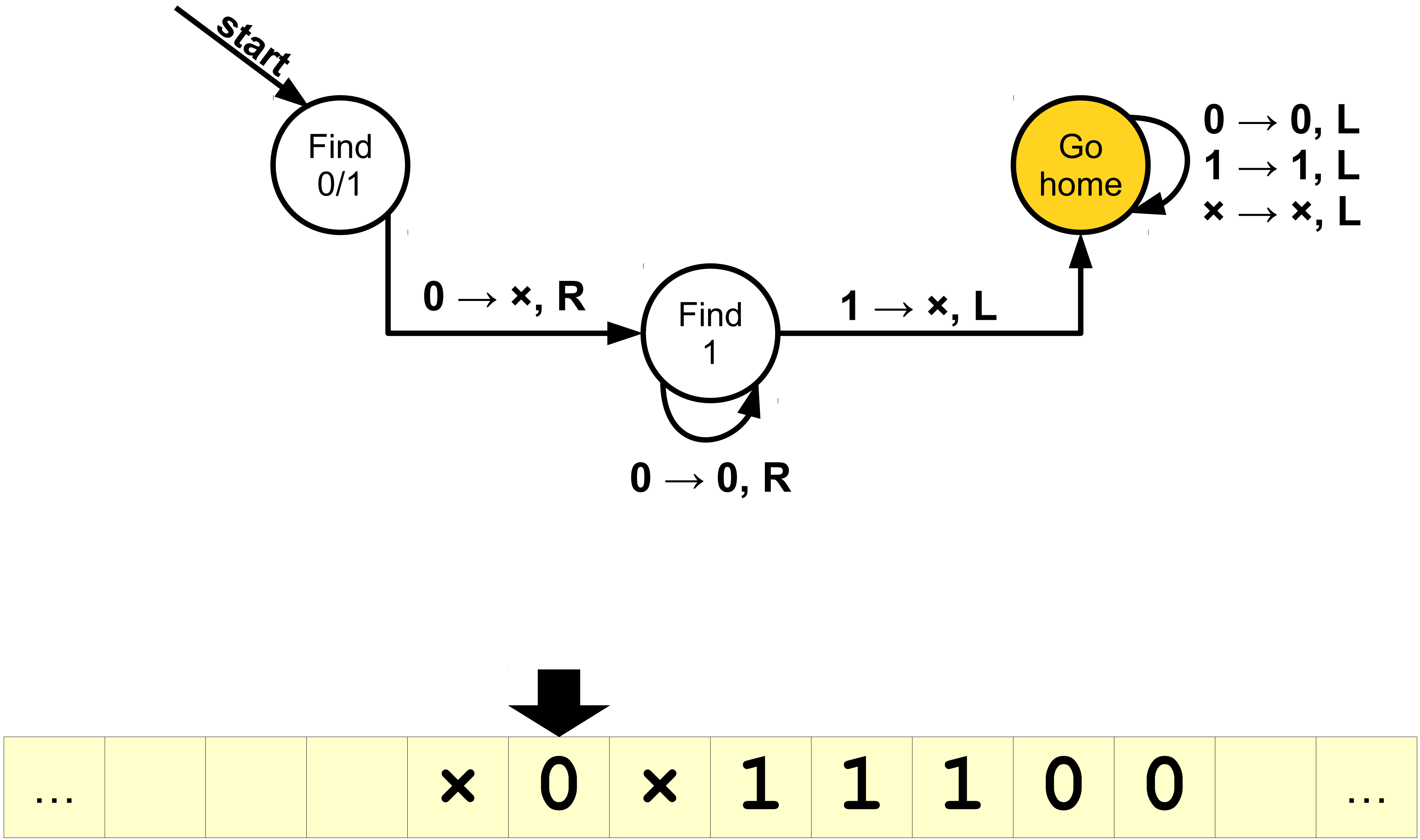

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
…
×
0
1
1
1
1
0
0
…

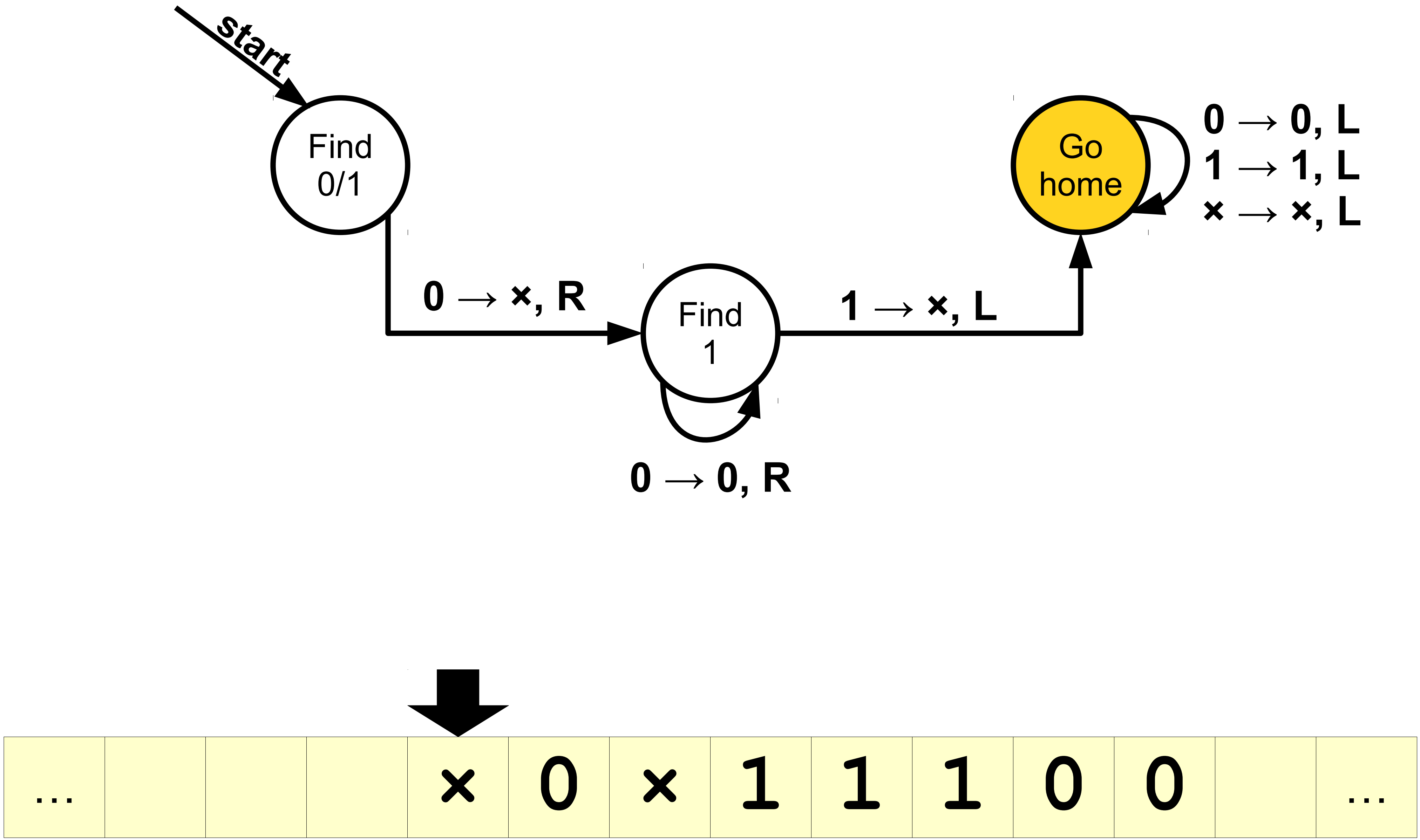

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
… × 0 1 1 1 1 0 0 …

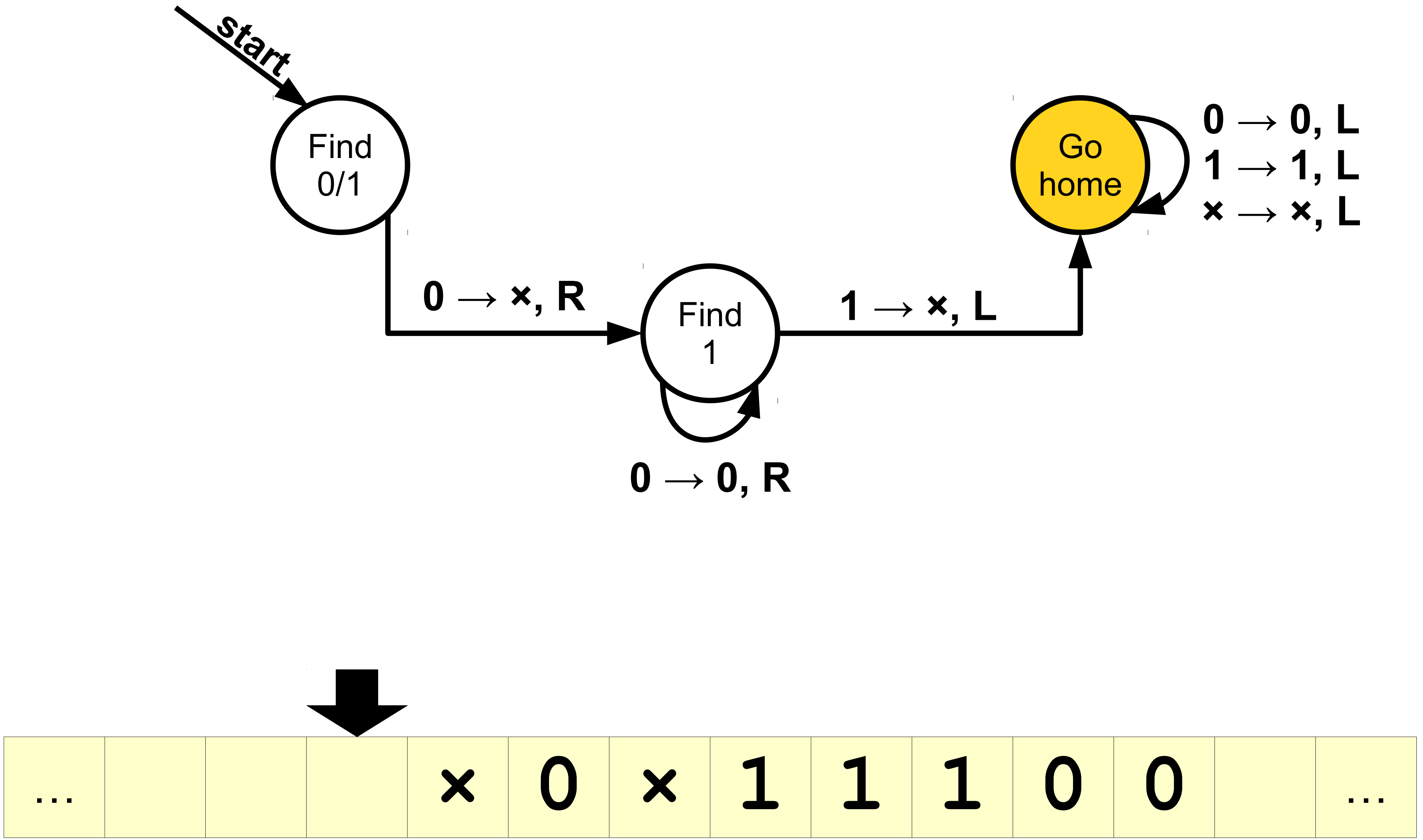


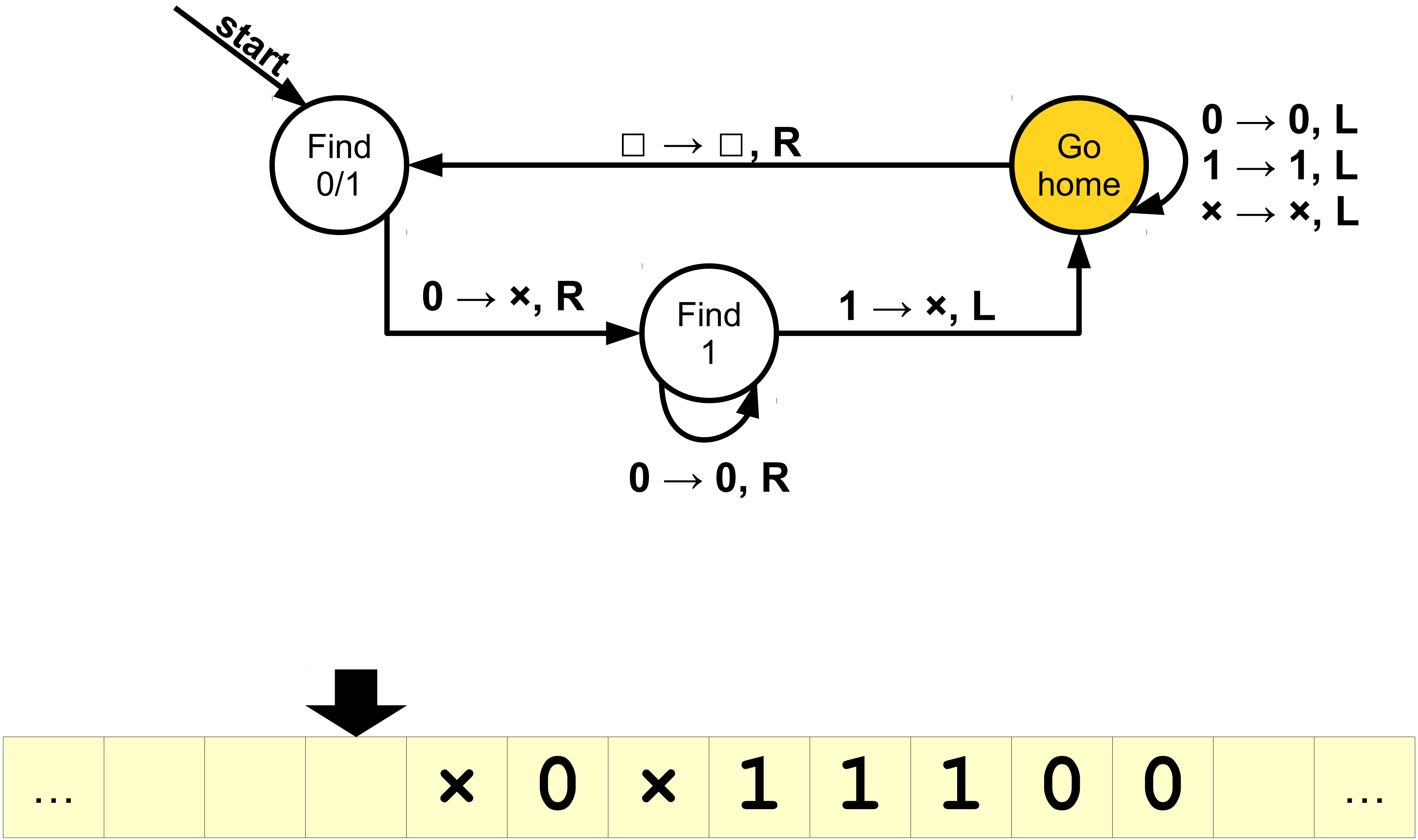
start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
…
×
0
×
1
1
1
0
0
…


start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
Go home
…
×
0
×
1
1
1
0
0
…

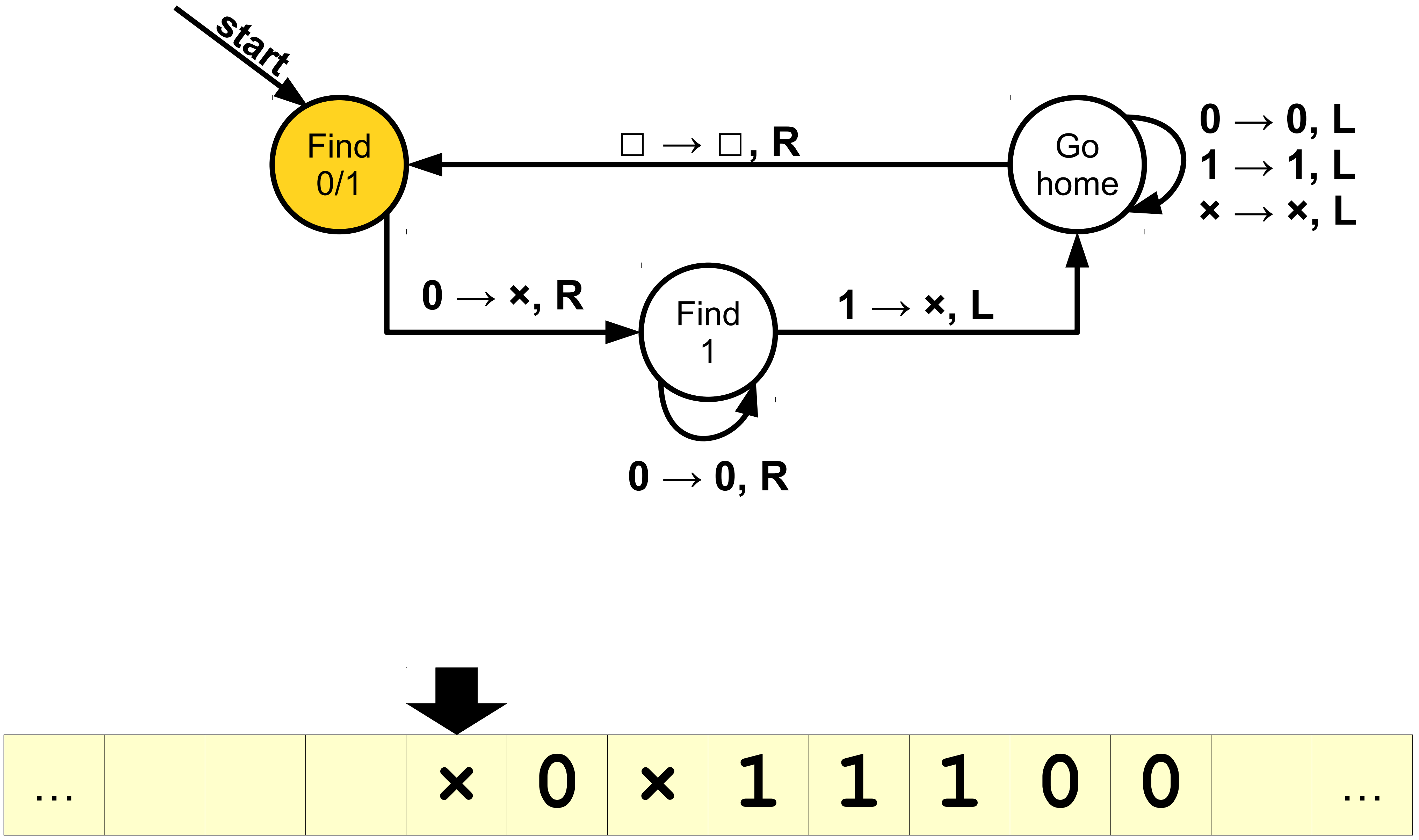

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
… × 0 × 1 1 1 0 0 …

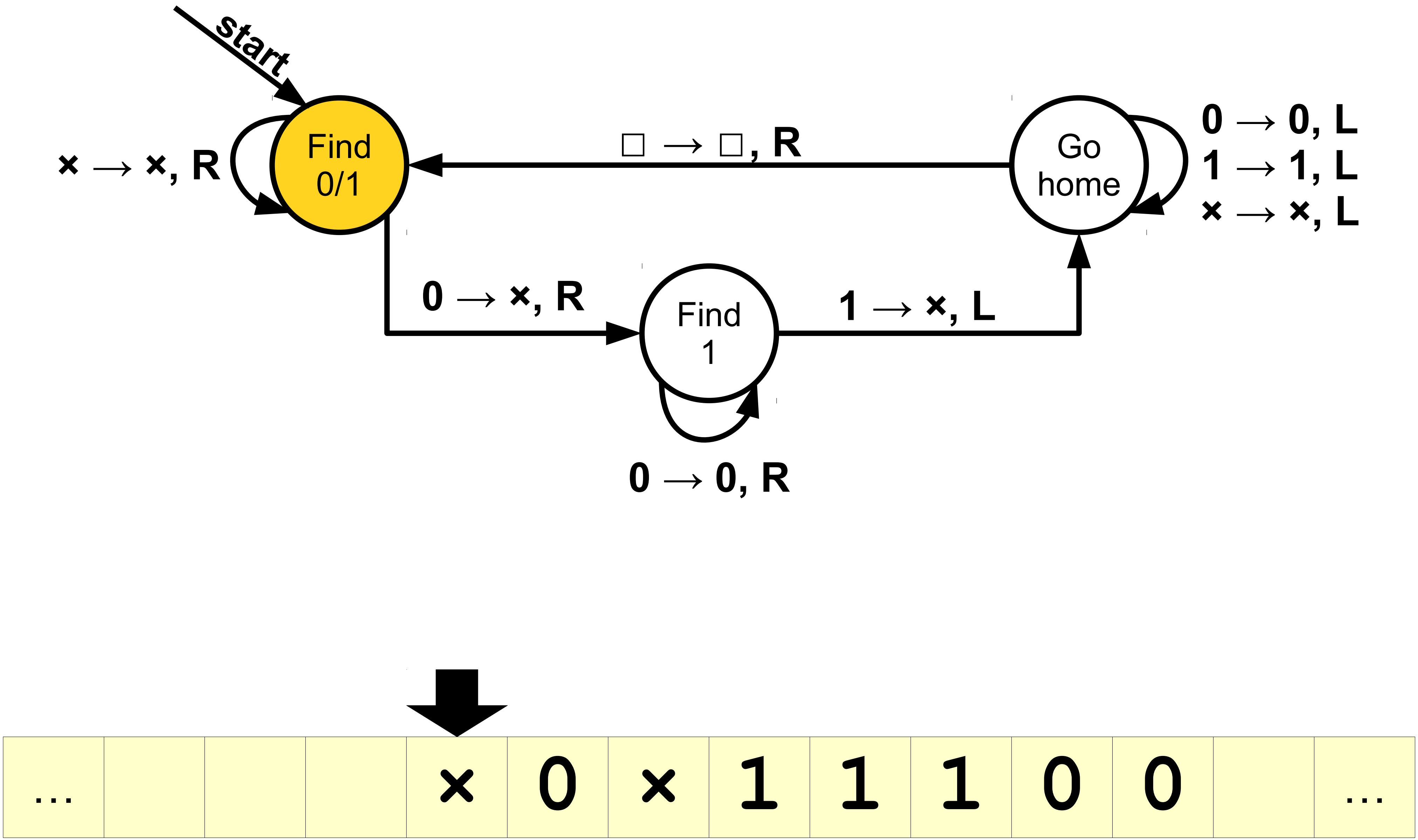

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
…
×
0
×
1
1
1
0
0
…

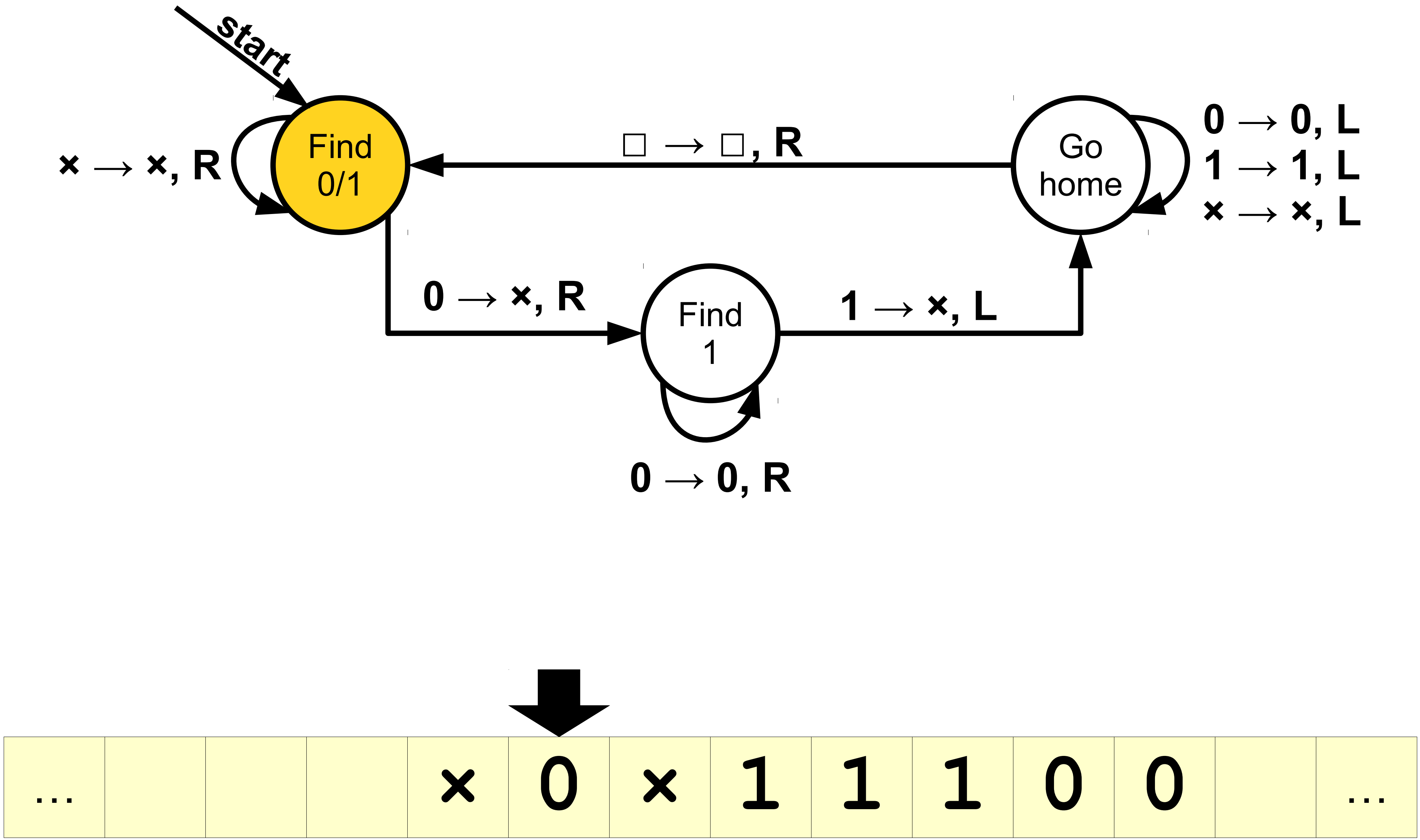

start
Find 0/1
0 → ×, R
Find 1
0 → 0, R
1 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
…
×
0
×
1
1
1
0
0
…

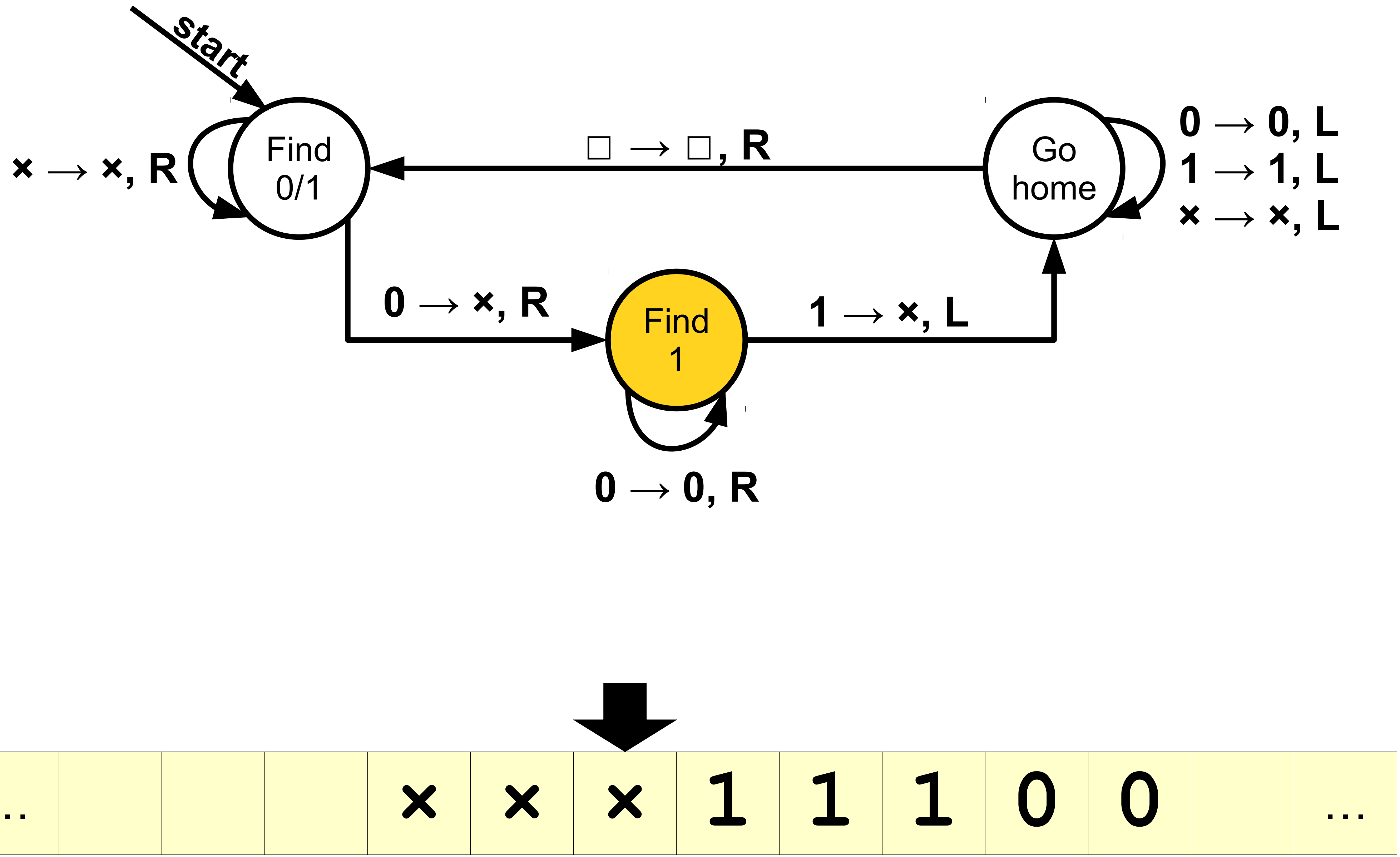


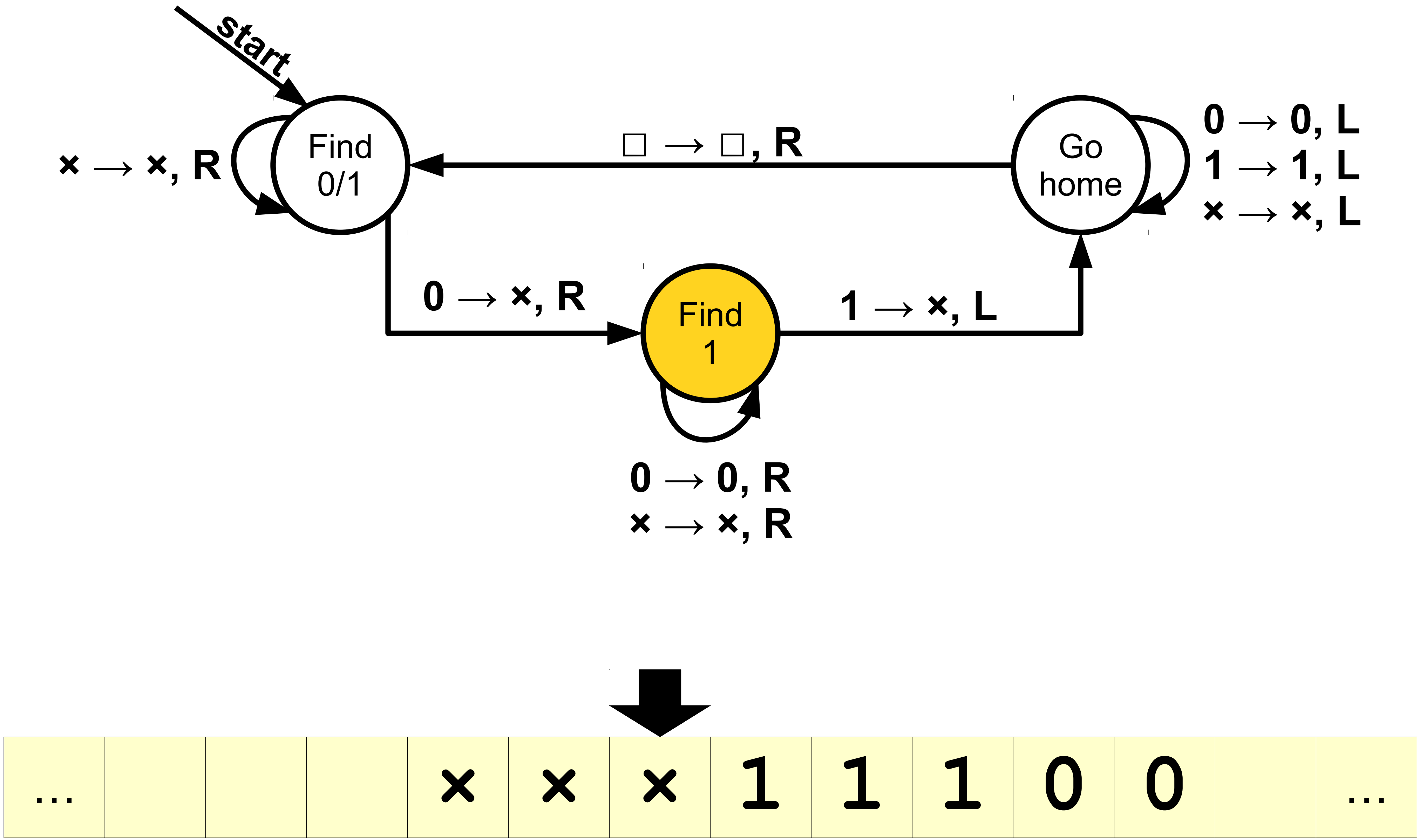
start
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
… × 0 × 1 1 1 0 0 …

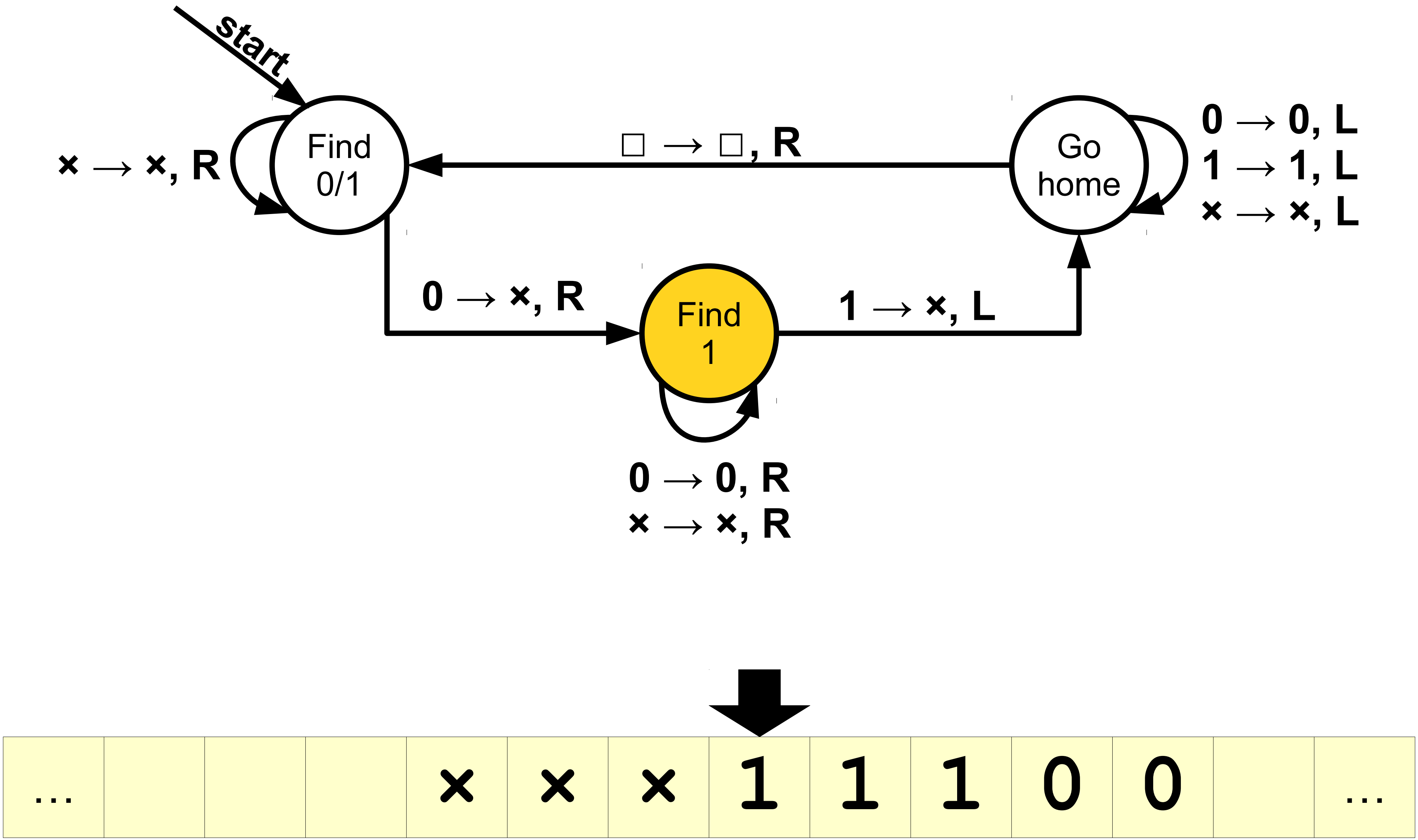

start
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
…
×
0
×
1
1
1
0
0
…


start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
… × 0 × 1 1 1 0 0 …

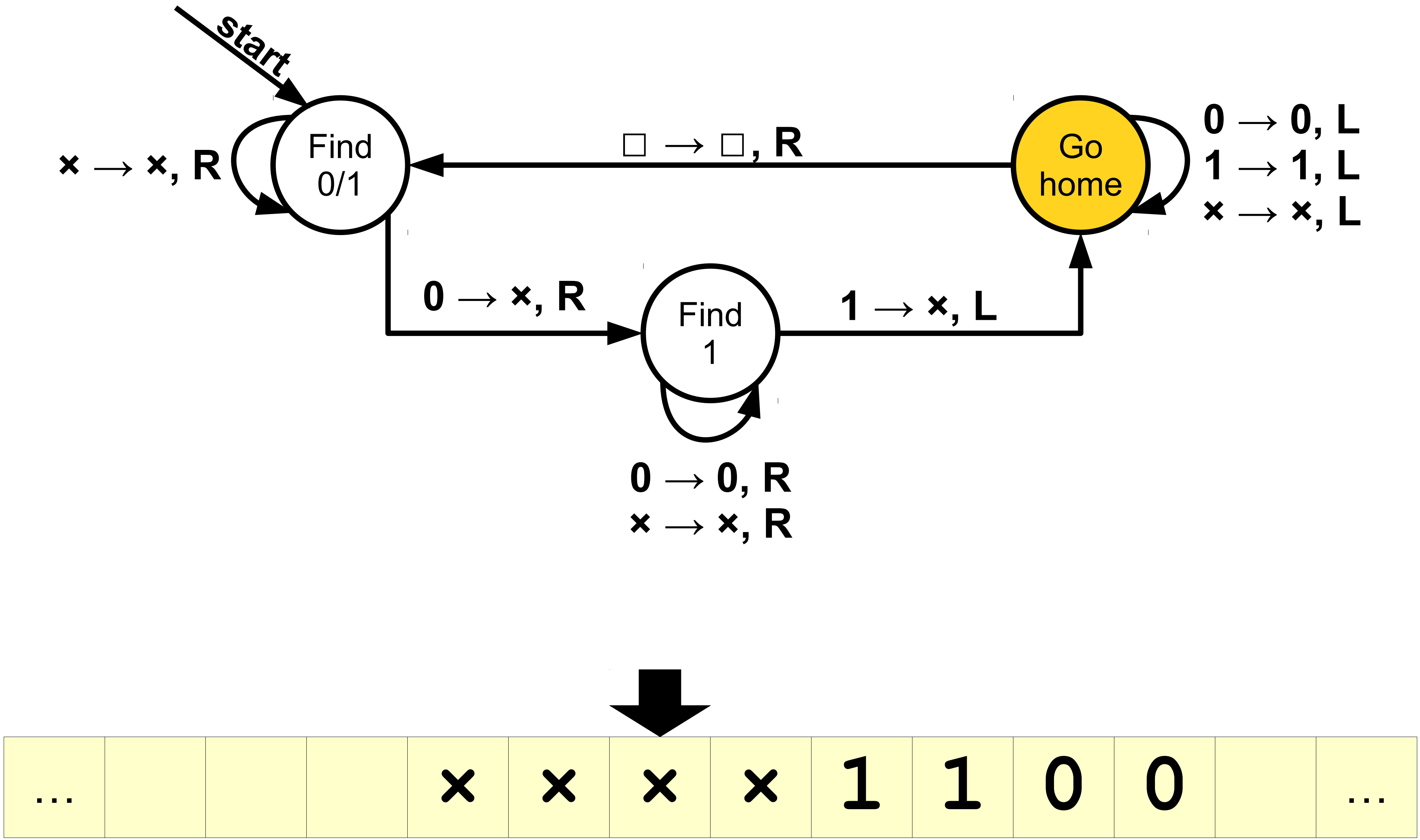

start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
… × 0 × 1 1 1 0 0 …


start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
… × × × 1 1 1 0 0 …

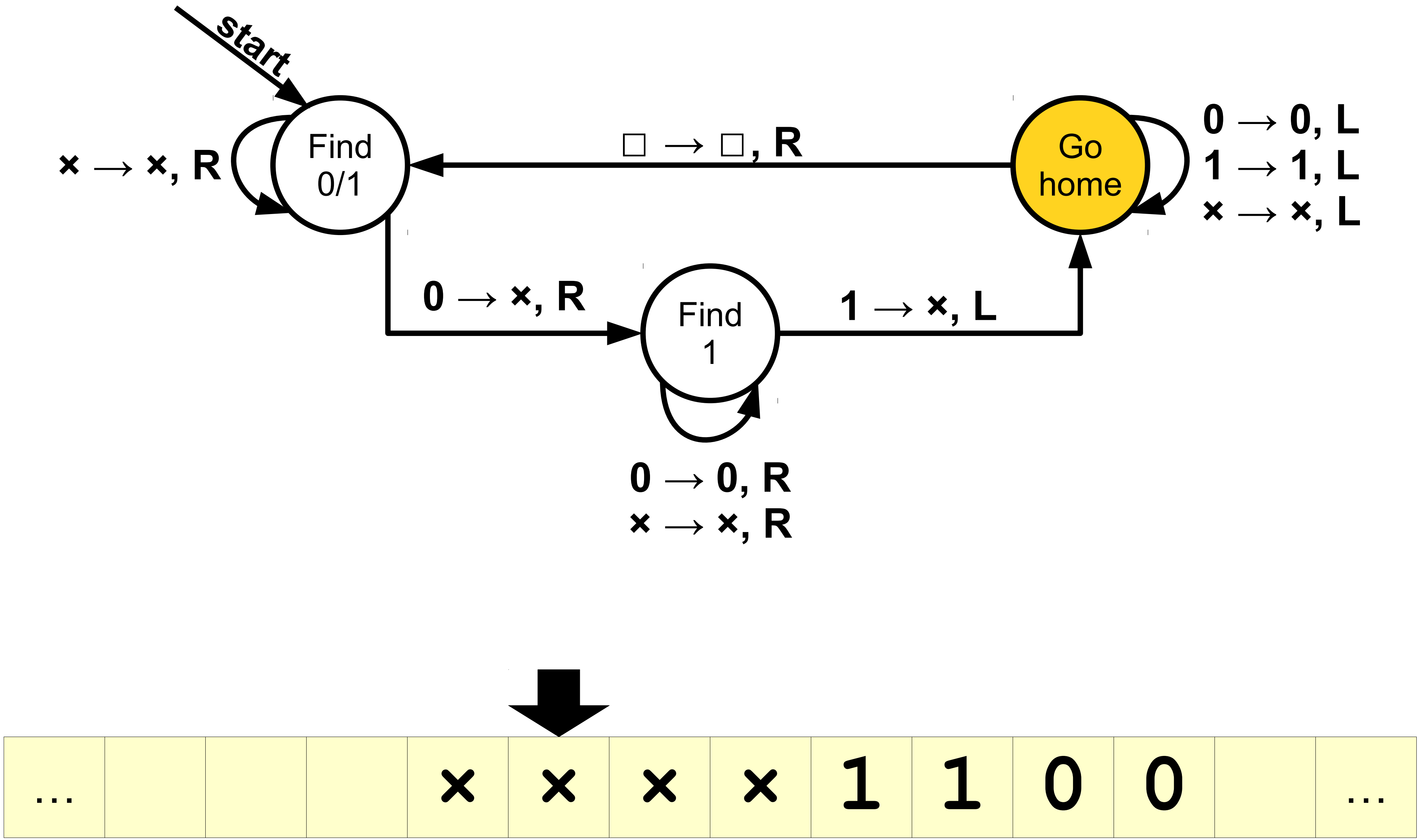

start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × 1 1 1 0 0 …



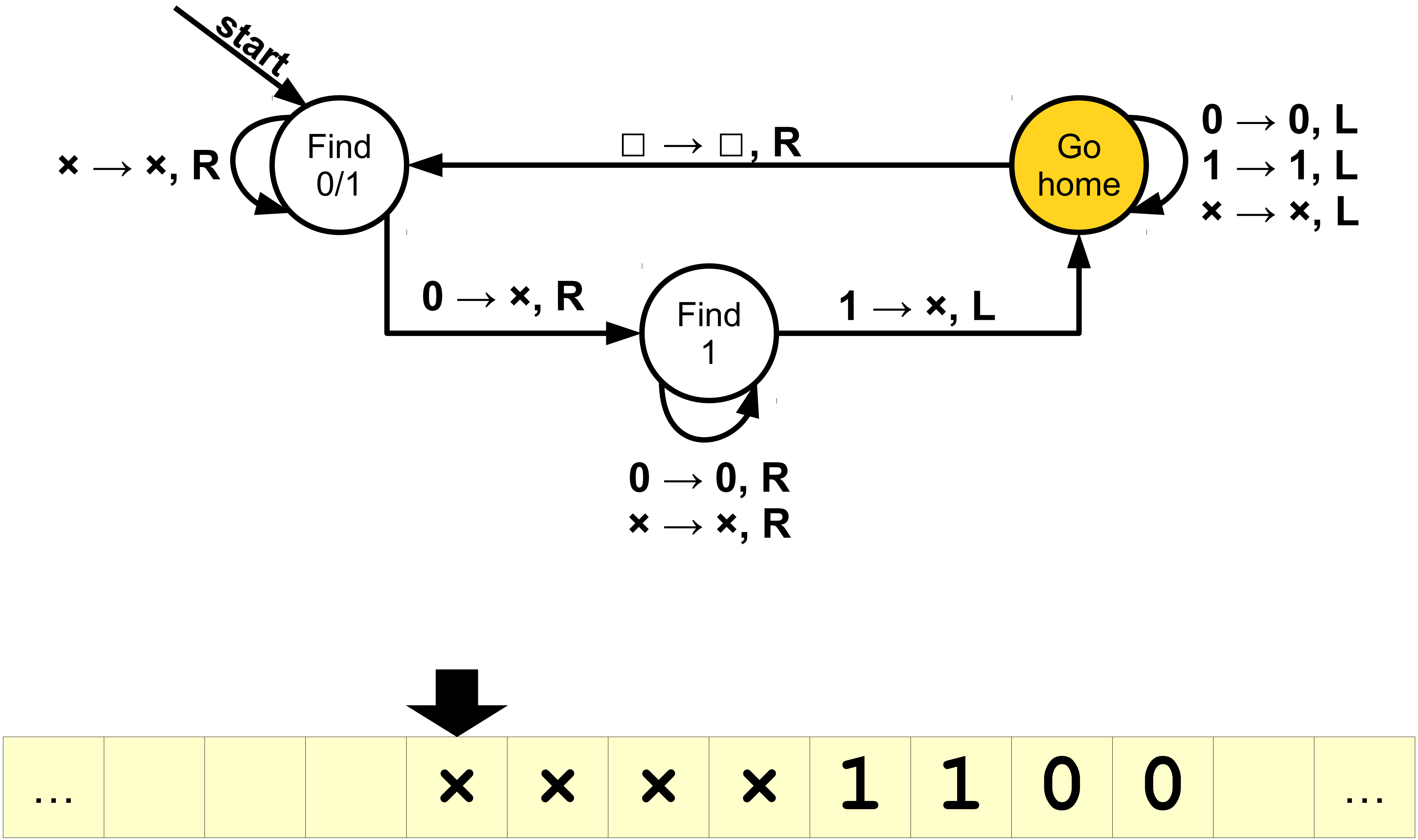
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × 1 1 1 0 0 …

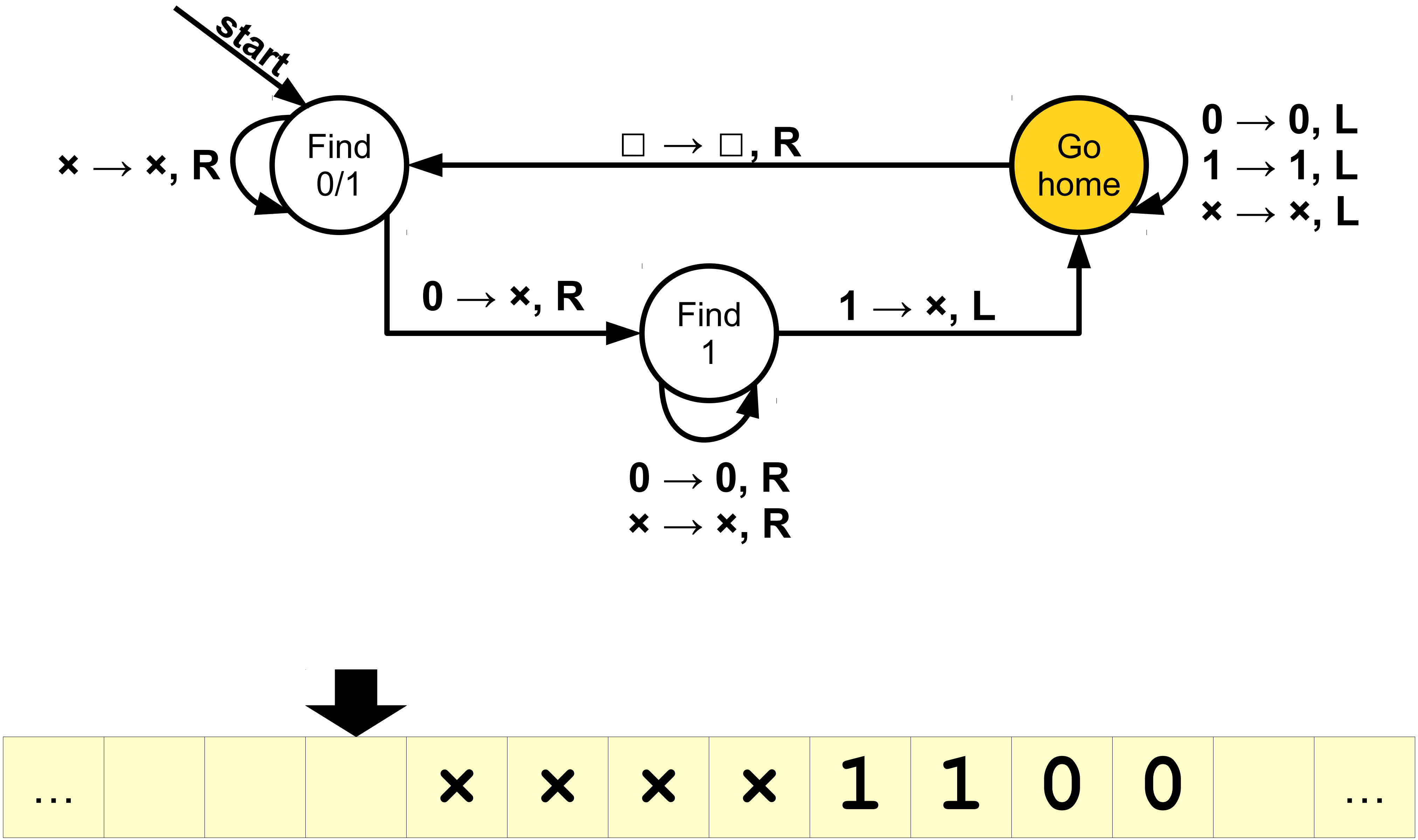


start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
1
1
0
0
…

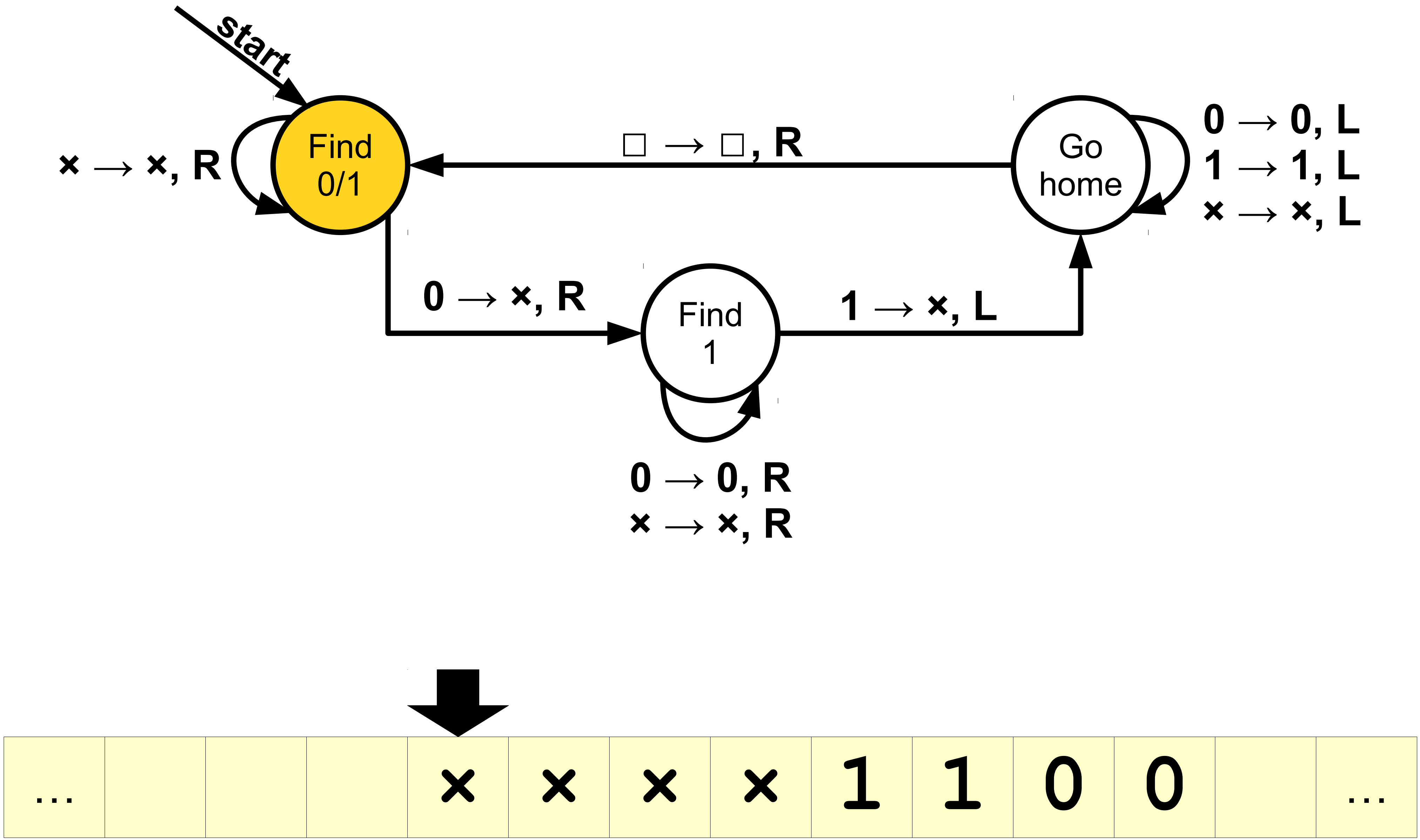


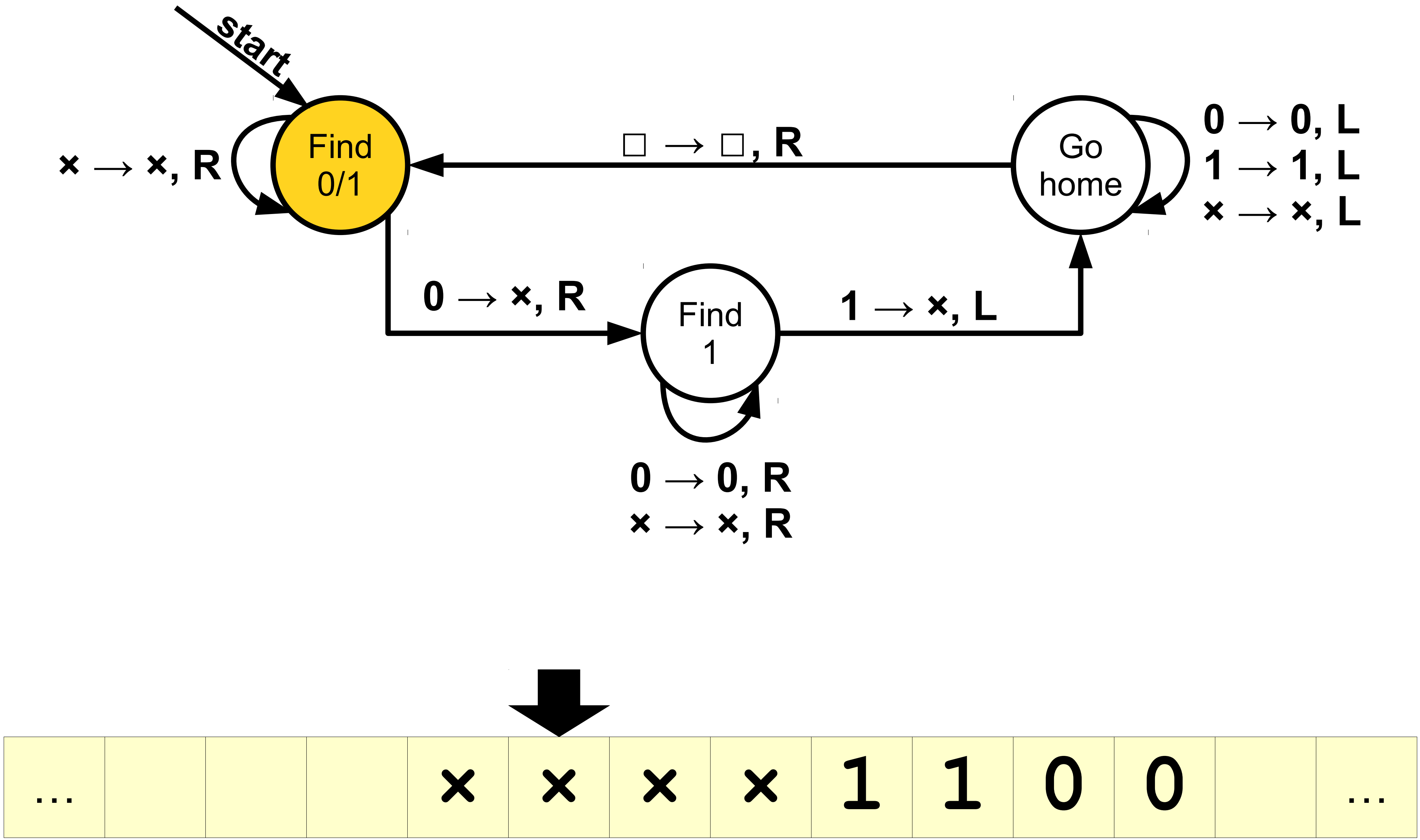
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × 1 1 0 0 …


start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
1
1
0
0
…

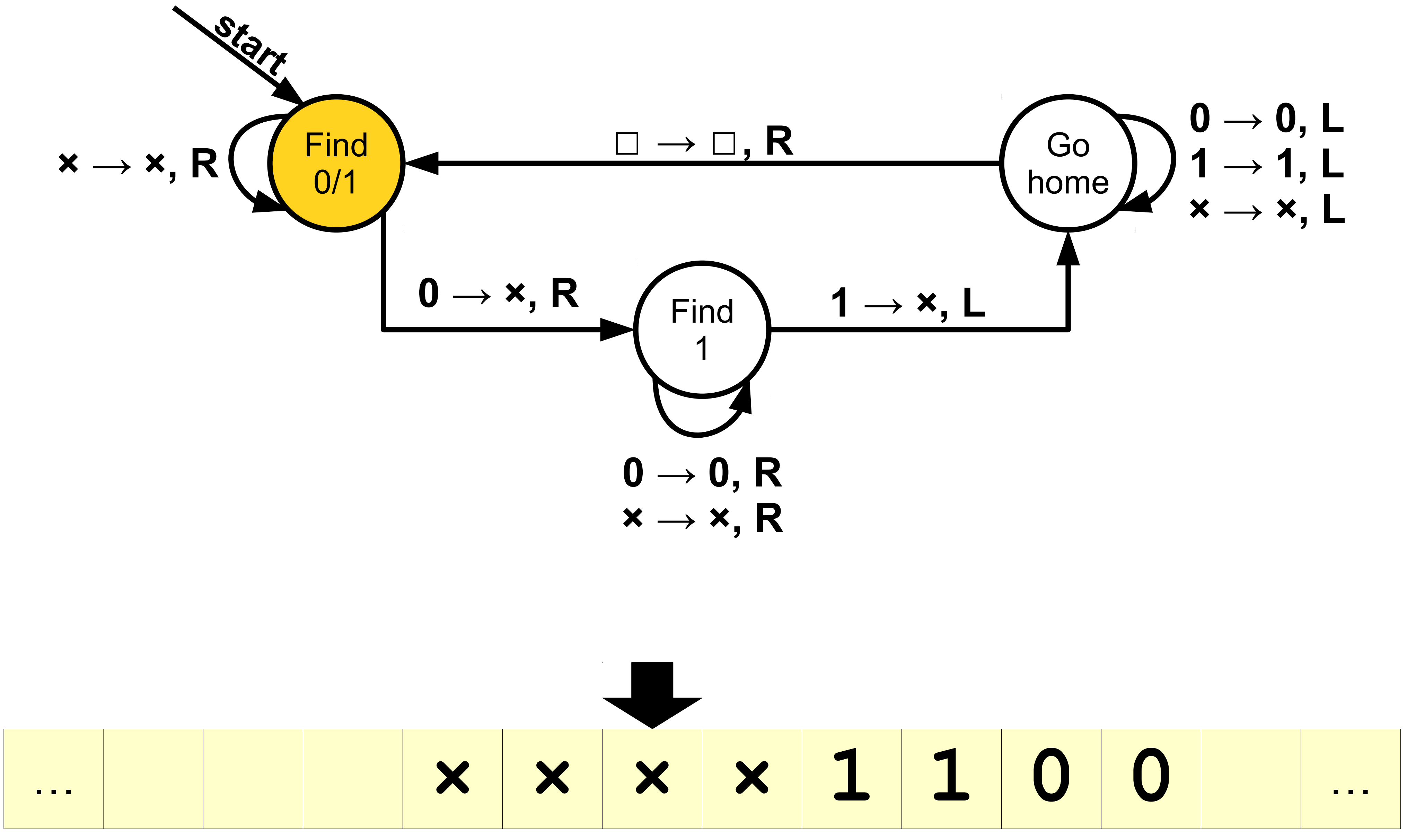


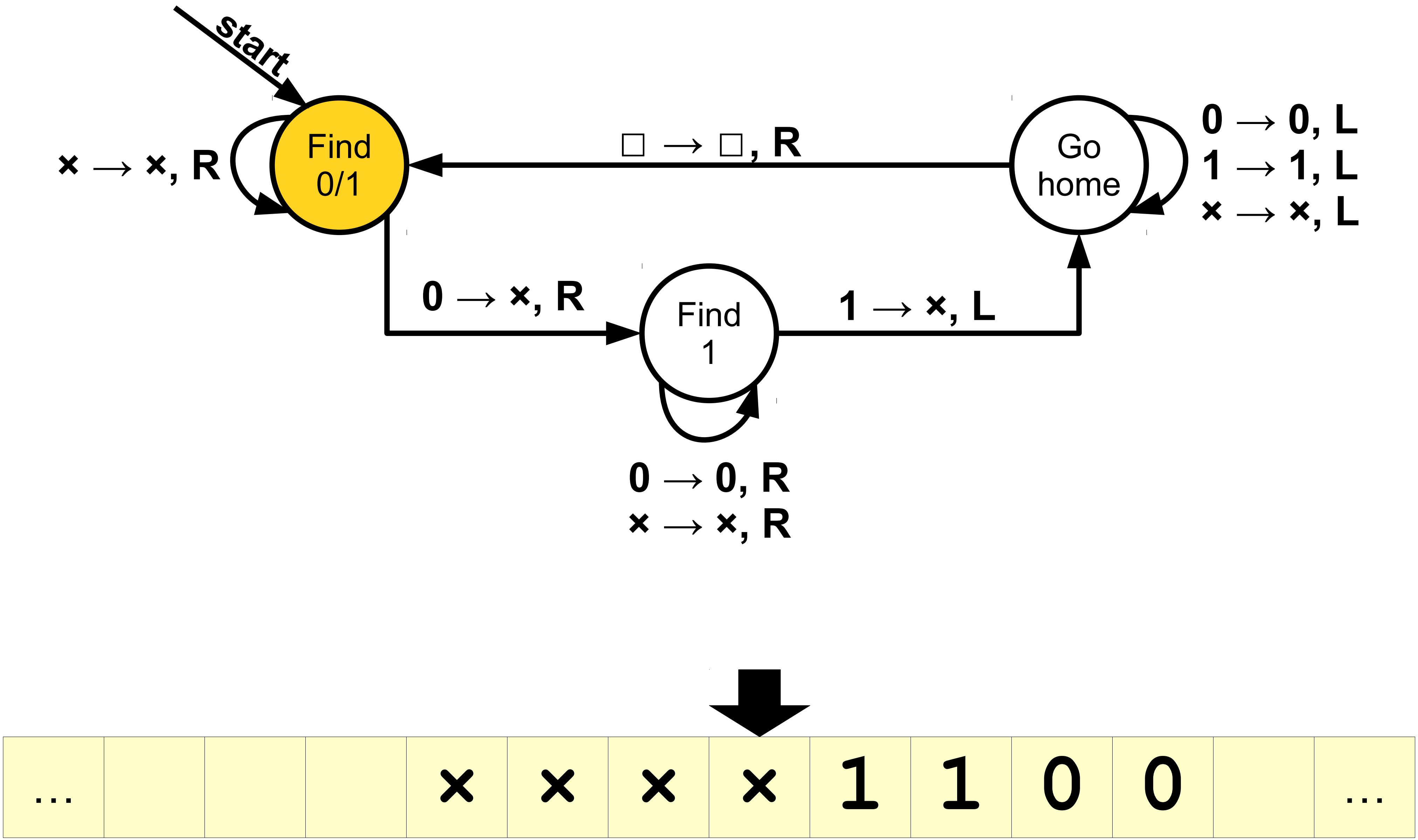
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
1
1
0
0
…


start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × 1 1 0 0 …

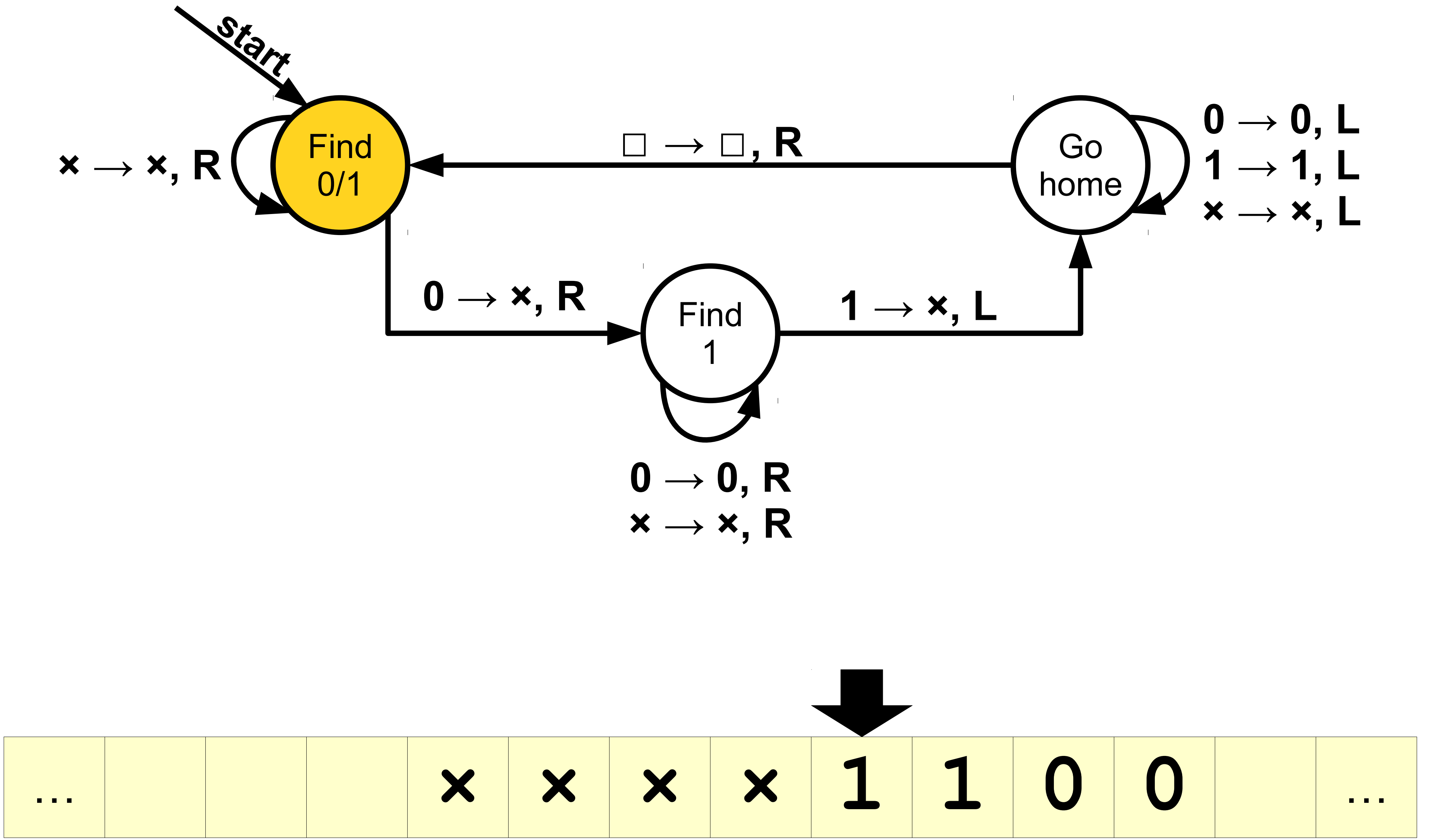


start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
1
1
0
0
…

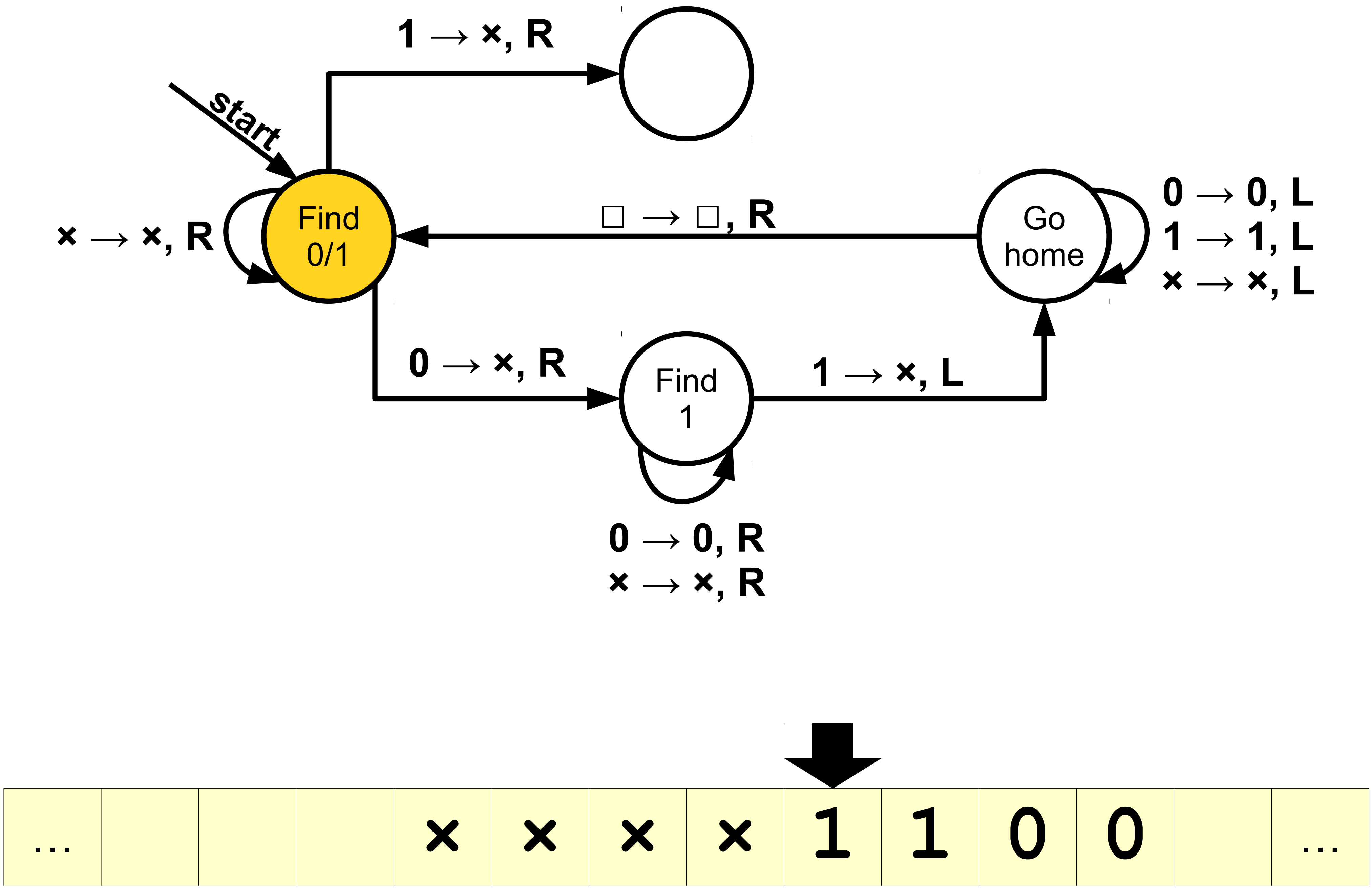

start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
1
1
0
0
…

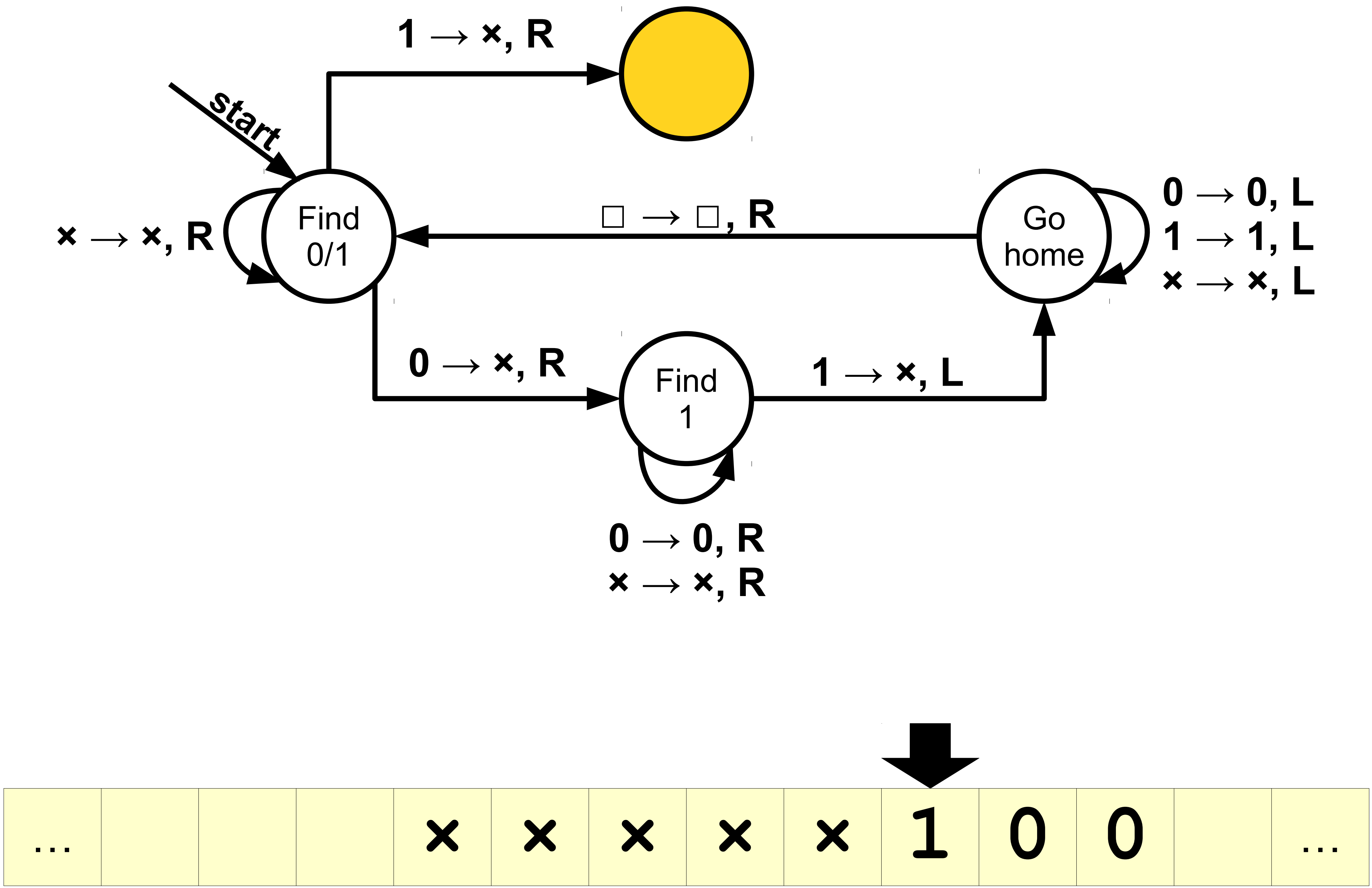


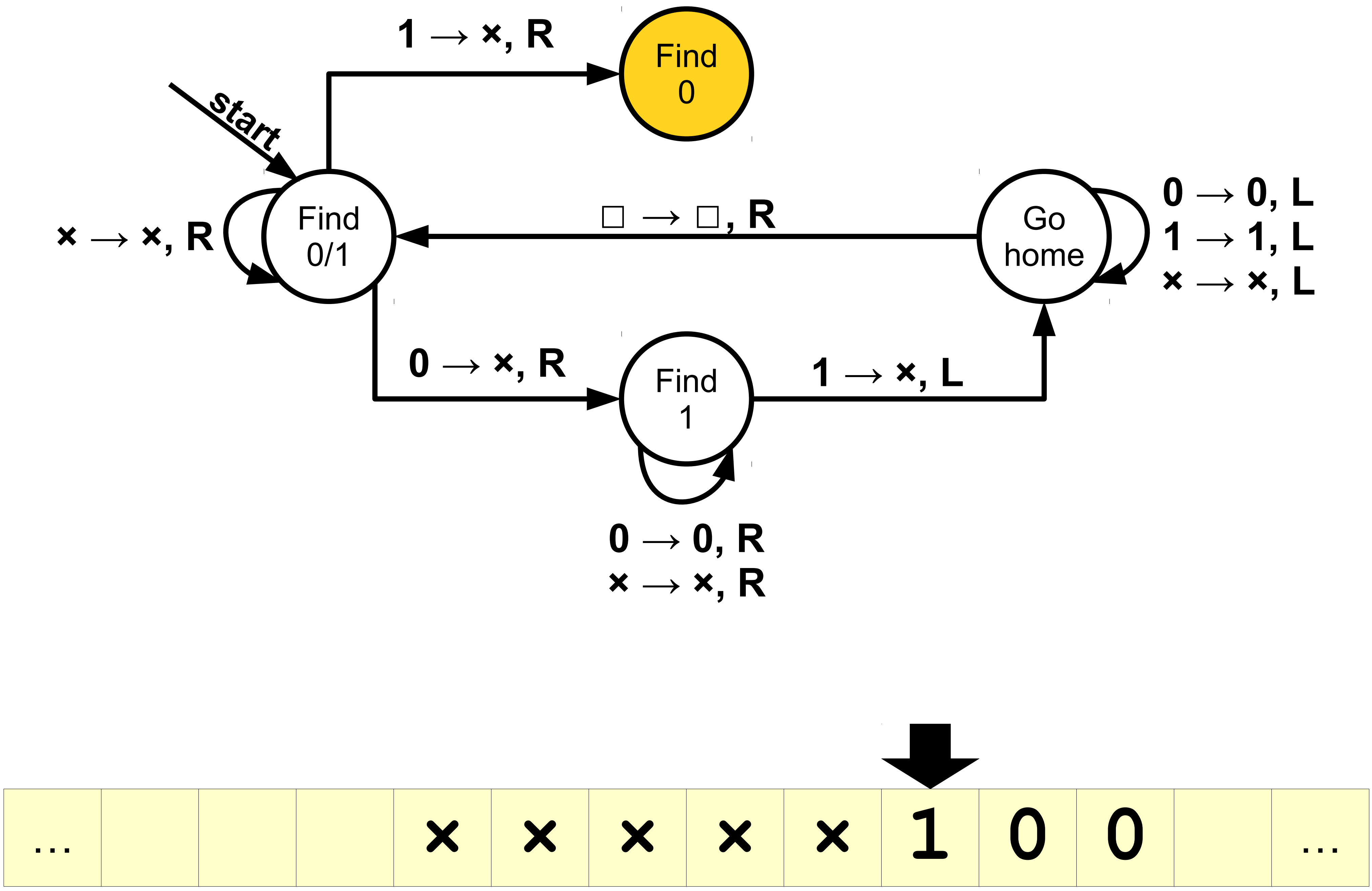
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × 1 1 0 0 …



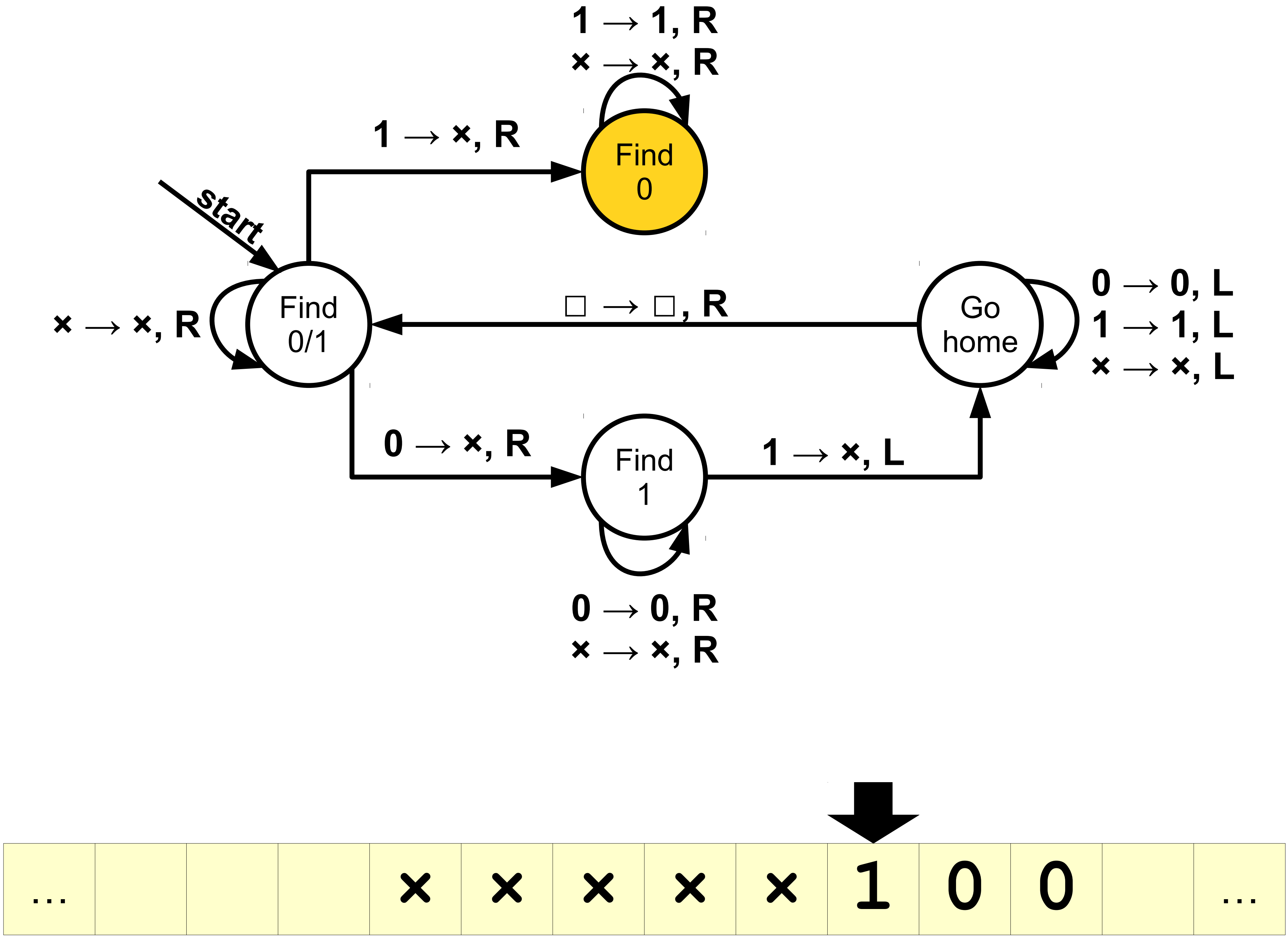
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × 1 1 0 0 …

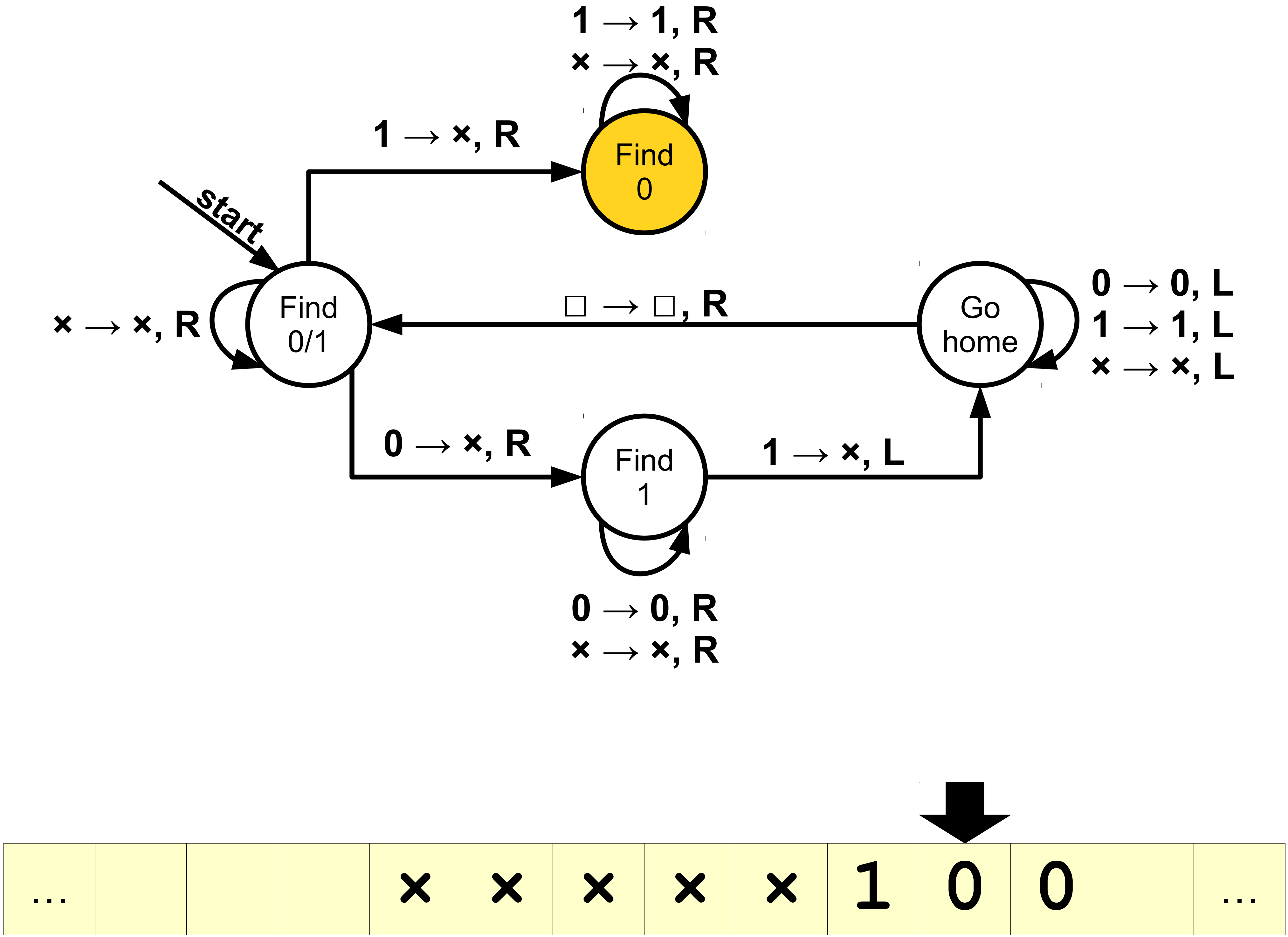

1 → ×, R
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
× × × × 1 1 0 0
…


1 → ×, R
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × × 1 0 0 …

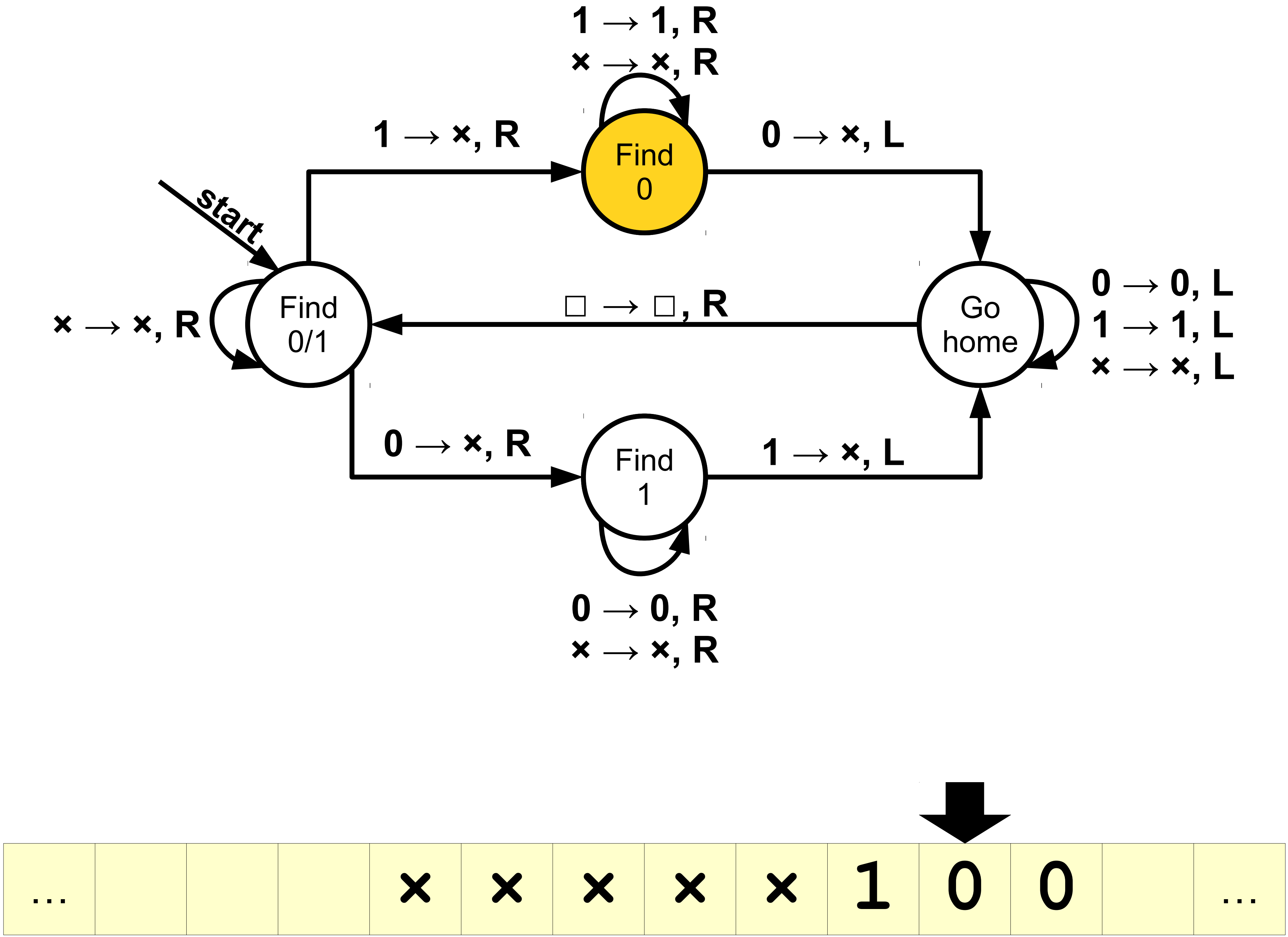


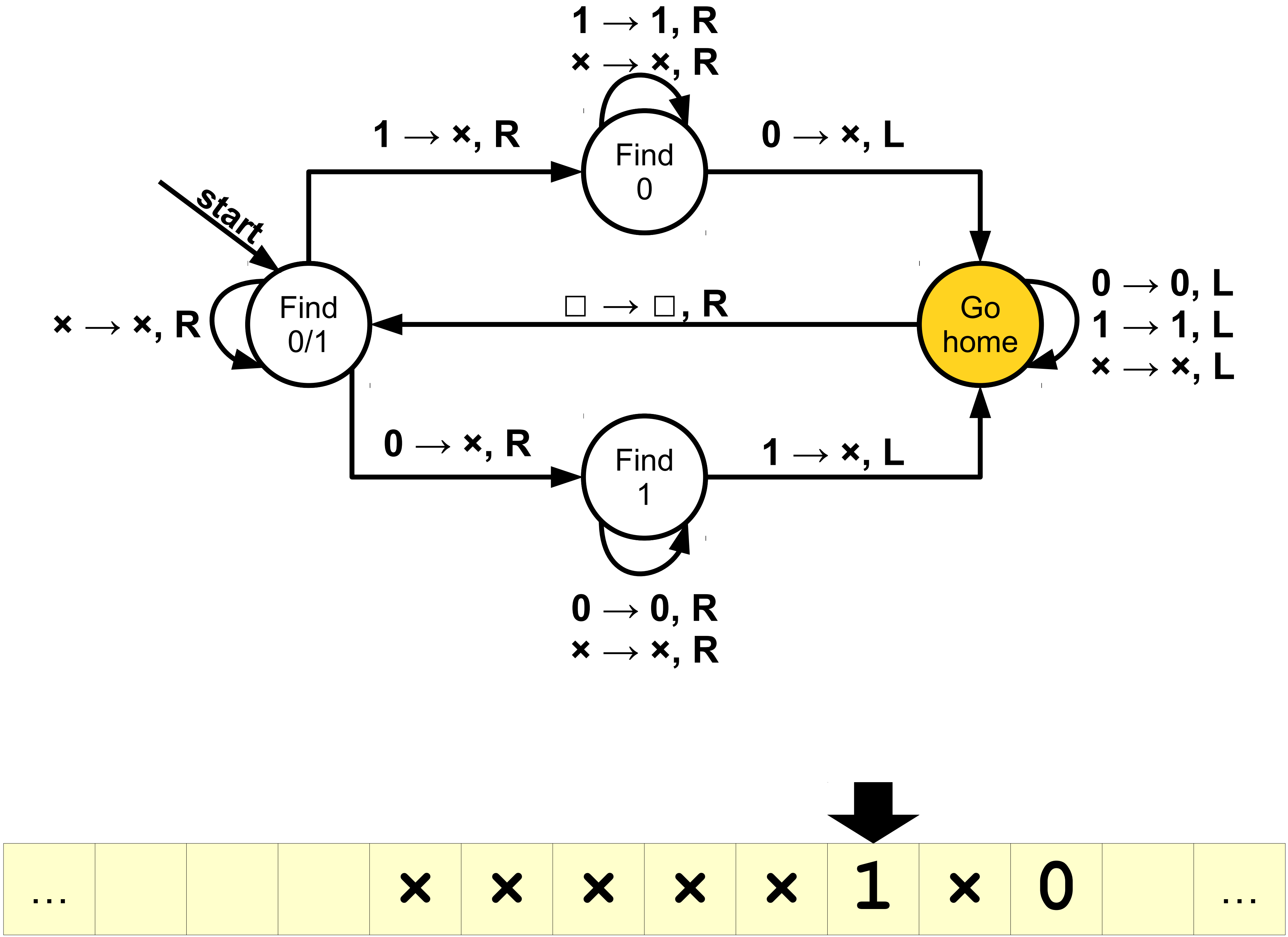
start
Find 0/1
× → ×, R
1 → ×, R
Find 0
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
… × × × × × 1 0 0 …


1 → 1, R
× → ×, R
1 → ×, R
Find
0
start
Find
0/1
× → ×, R
□ → □, R
Go
home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find
1
1 → ×, L
0 → 0, R
× → ×, R
...
×
×
×
×
×
1
0
0
...

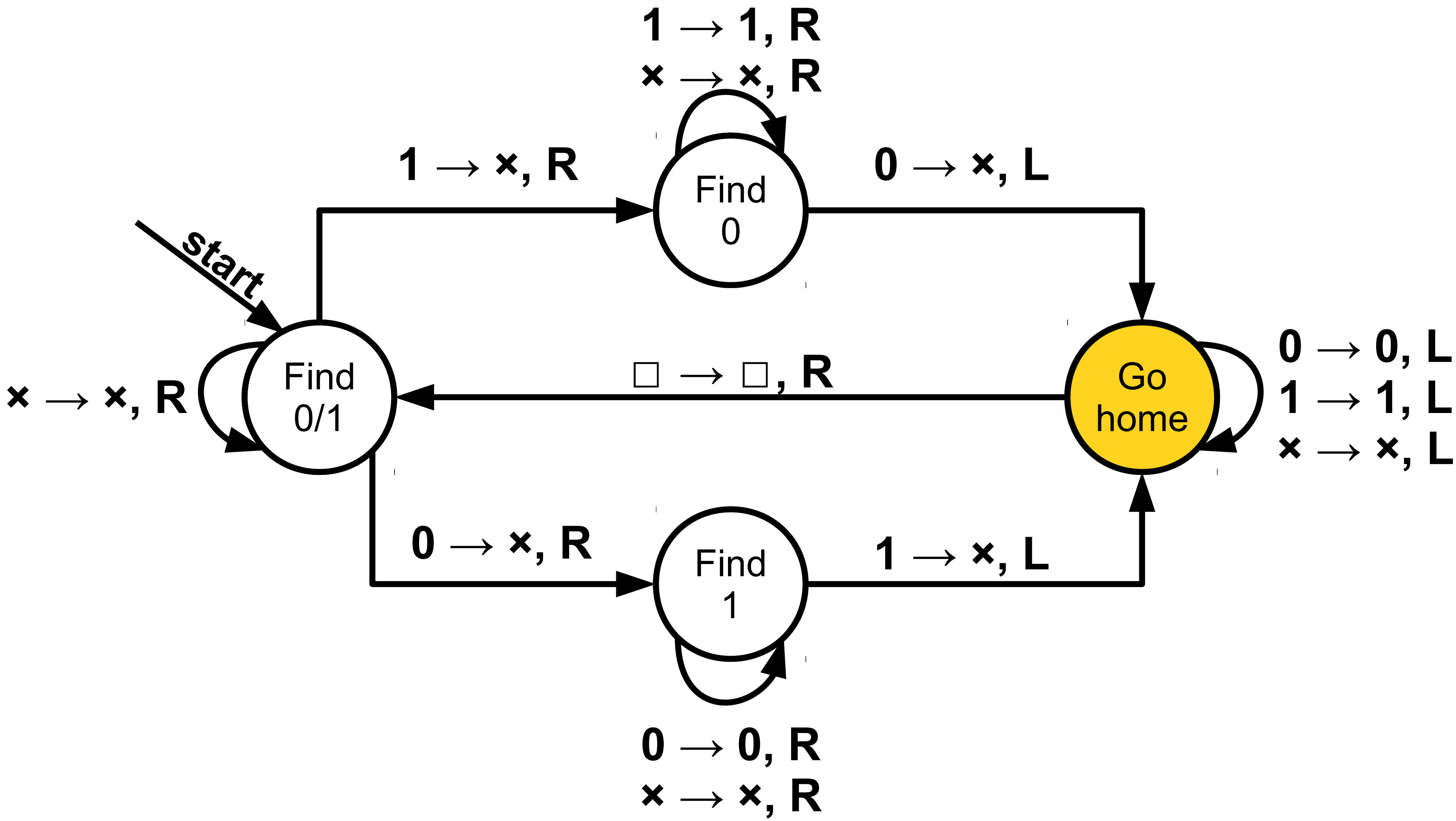


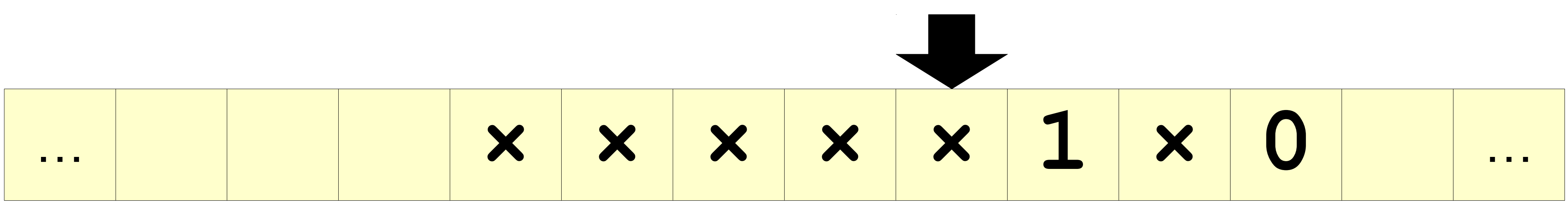
1 → 1, R
× → ×, R
1 → ×, R
Find 0
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
× × × × × 1 0 0
…

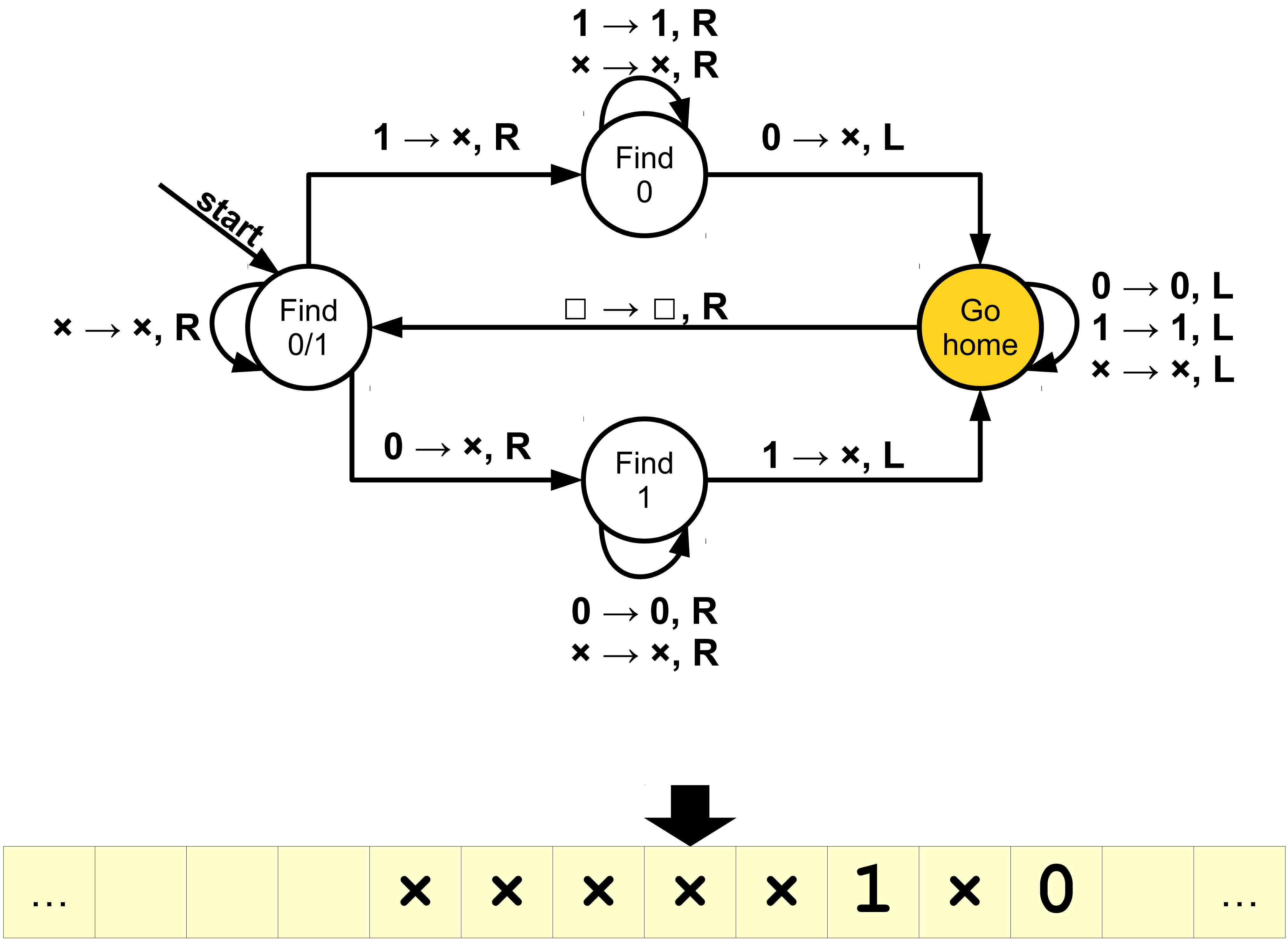

1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
0
0
…



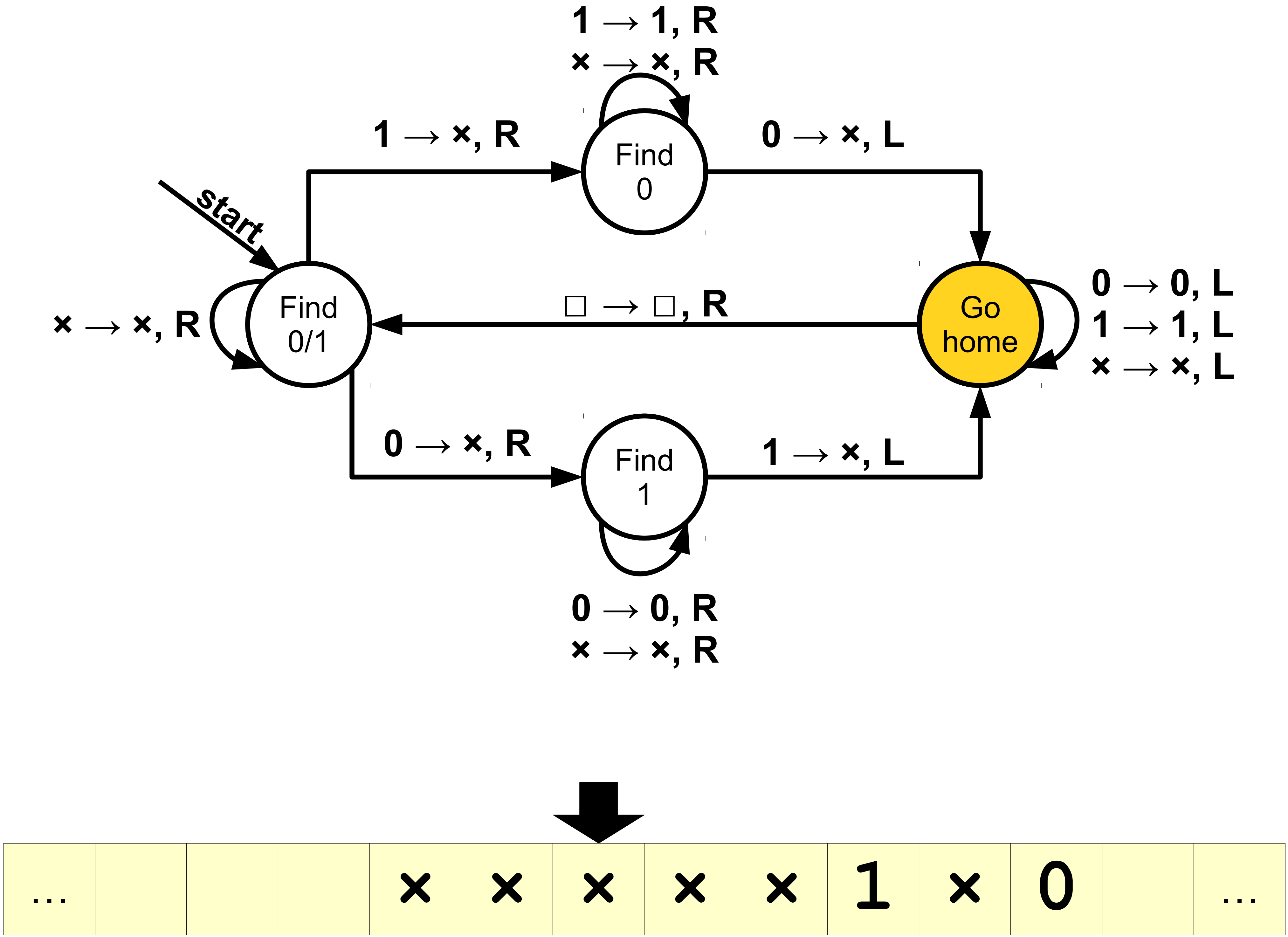
start
Find 0/1
Find 0
Find 1
Go home
1 → ×, R
0 → ×, R
0 → ×, L
1 → ×, L
□ → □, R
× → ×, R
1 → 1, R
× → ×, R
0 → 0, R
× → ×, R
0 → 0, L
1 → 1, L
× → ×, L
… × × × × × 1 × 0 …

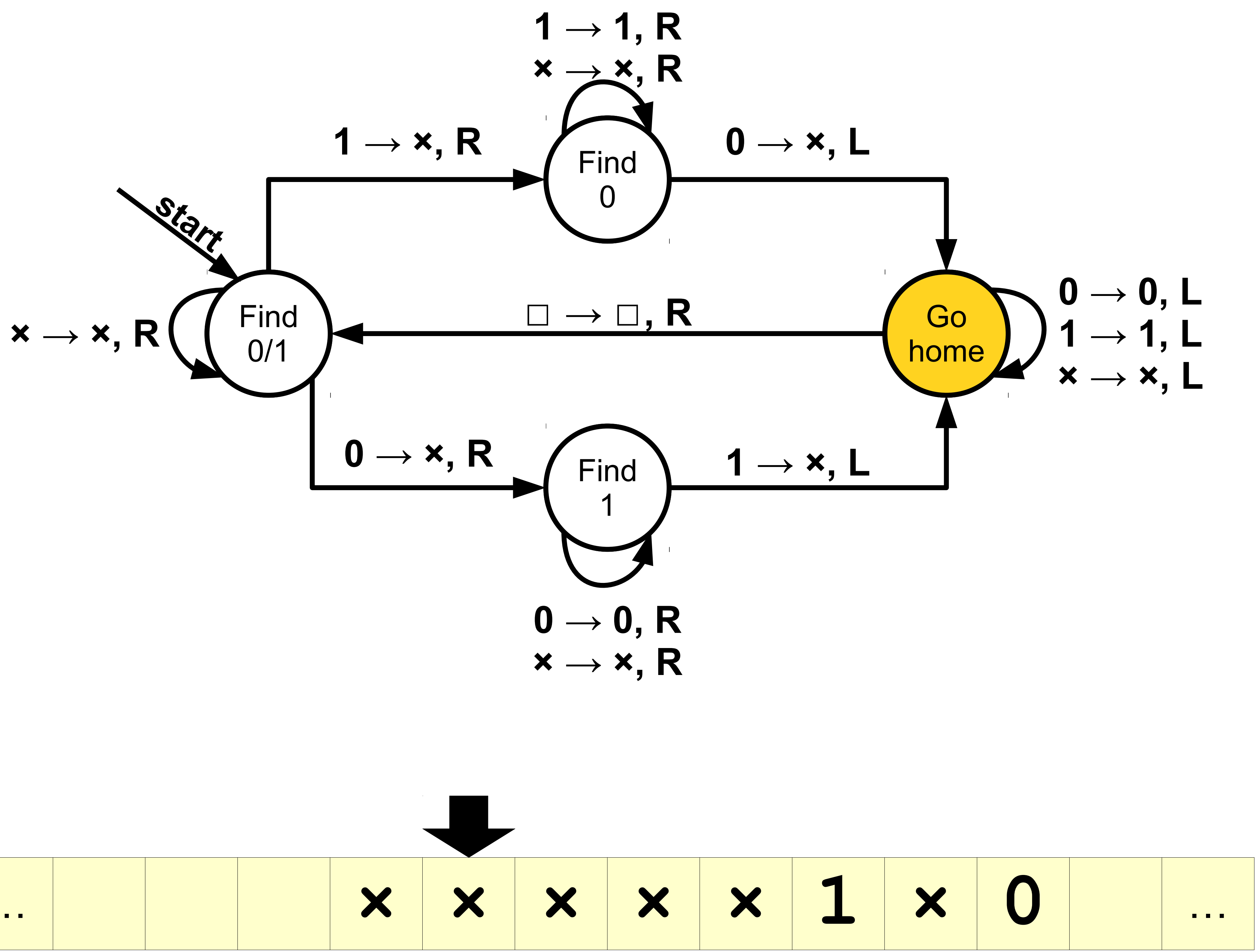

start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L


…
×
×
×
×
×
1
×
0
…

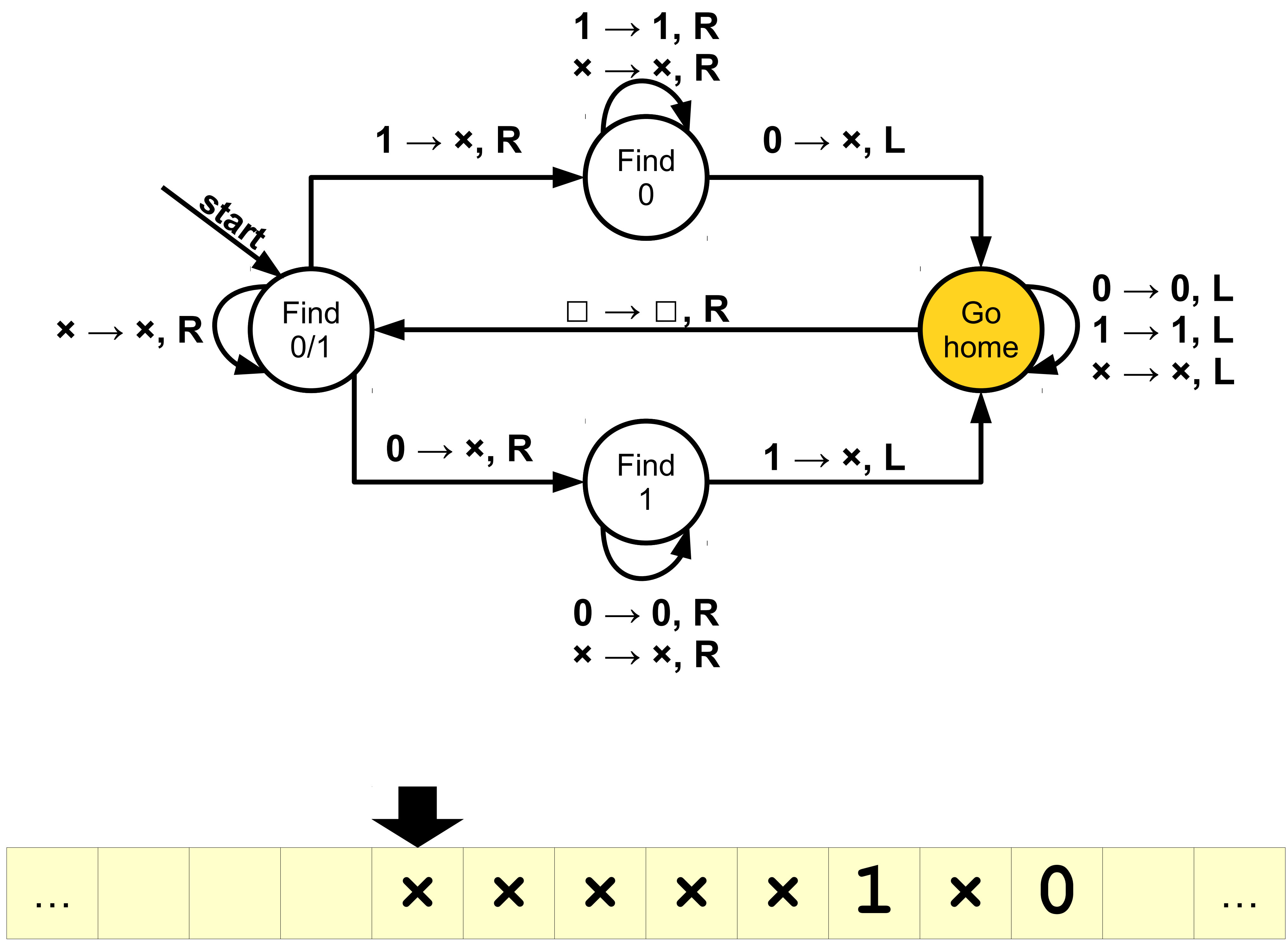


start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
…
× × × × × 1 × 0
…

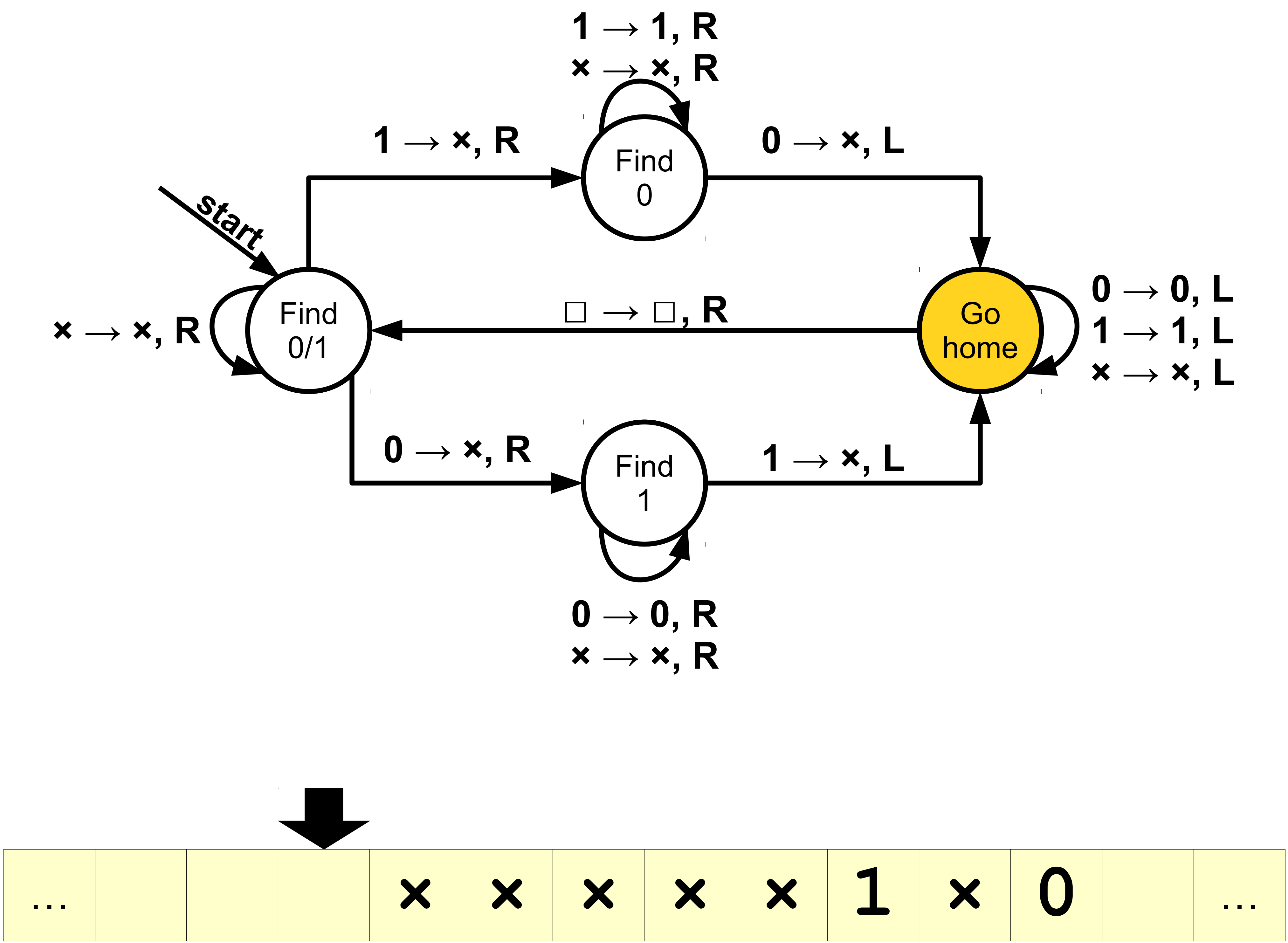

1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…

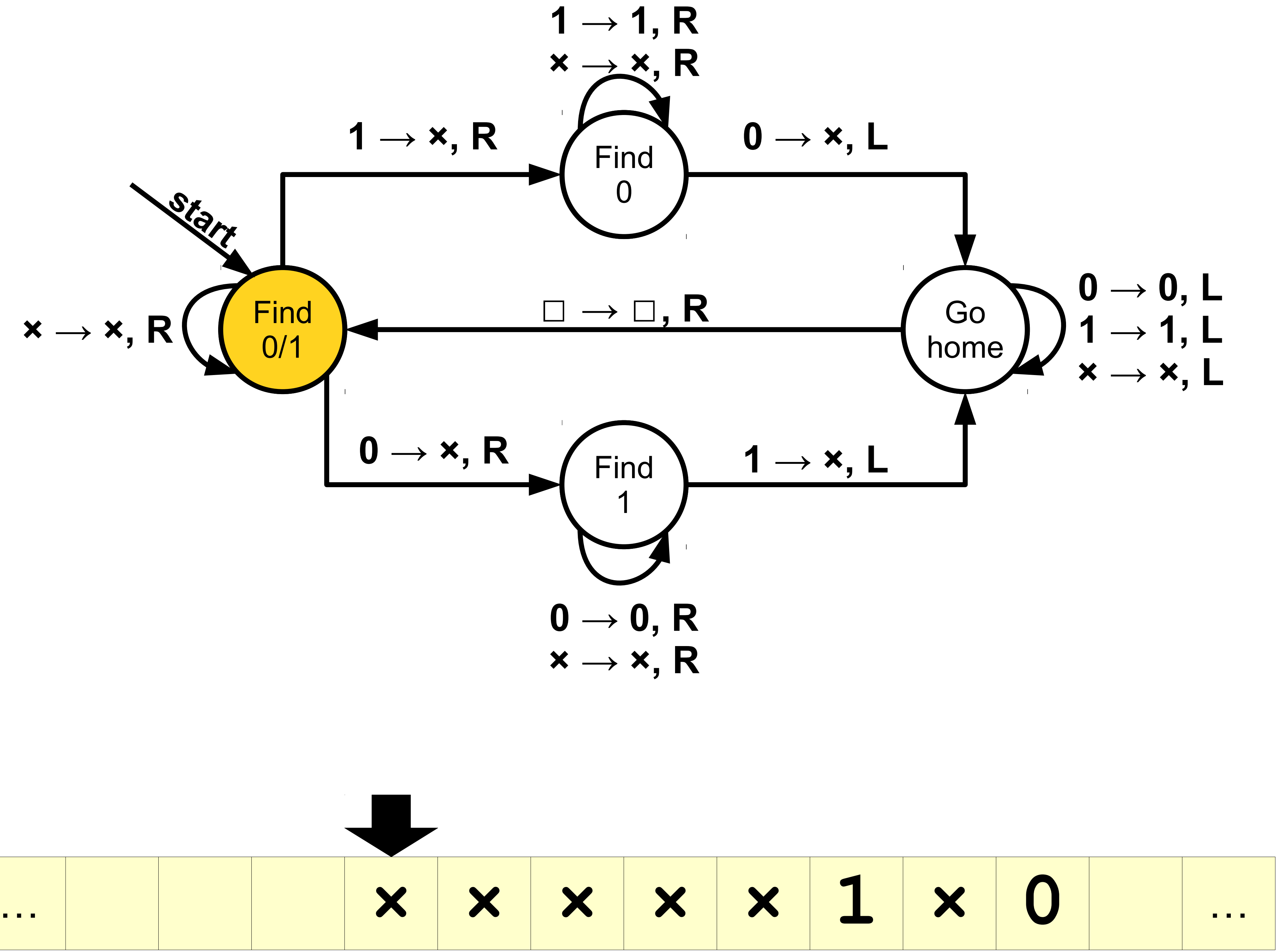


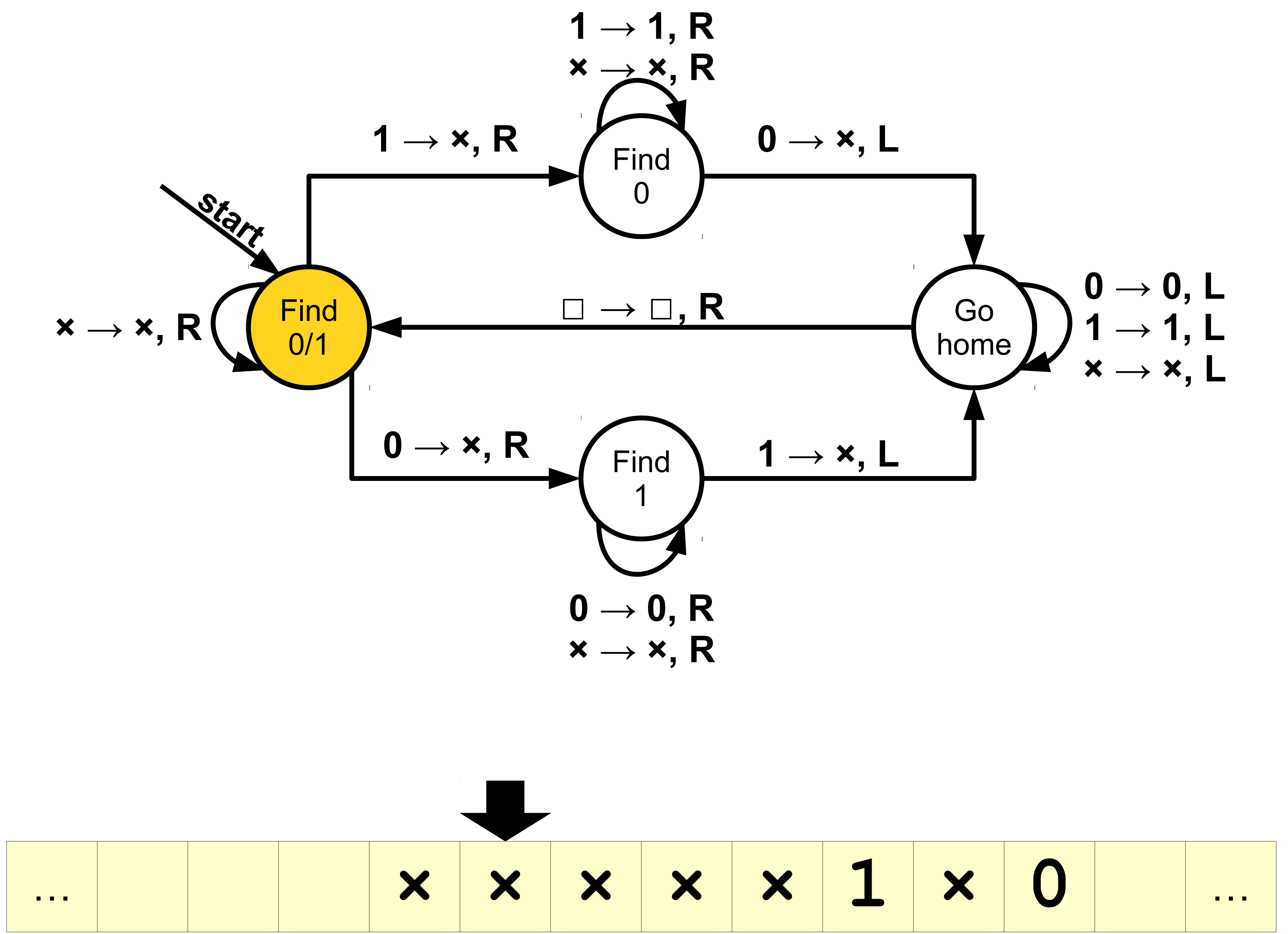
start
Find 0/1
× → ×, R
1 → ×, R
0 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Find 1
0 → 0, R
× → ×, R
1 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
… × × × × × 1 × 0 …

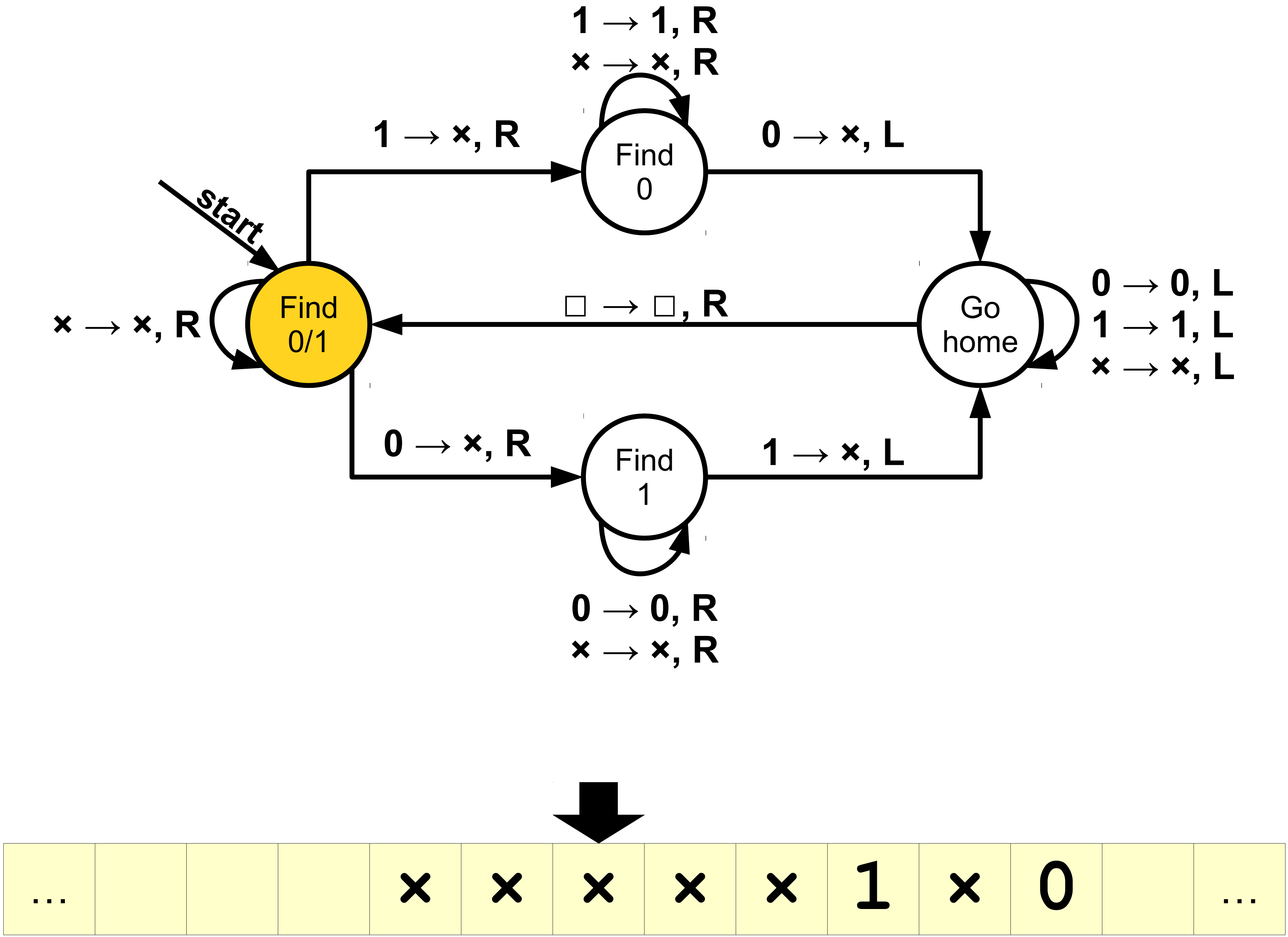


1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
Find 1
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…

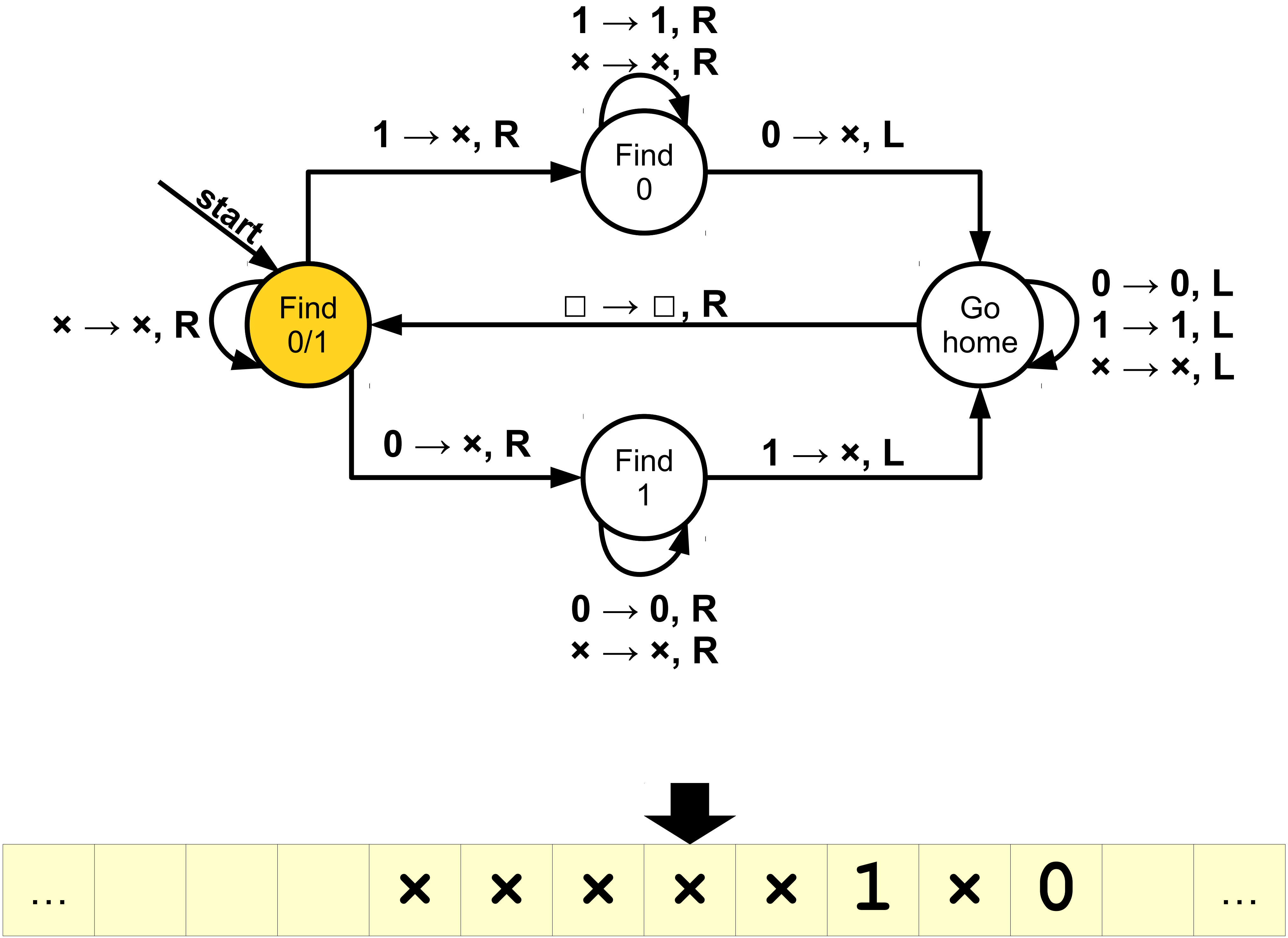

1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…


start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
… × × × × × 1 × 0 …

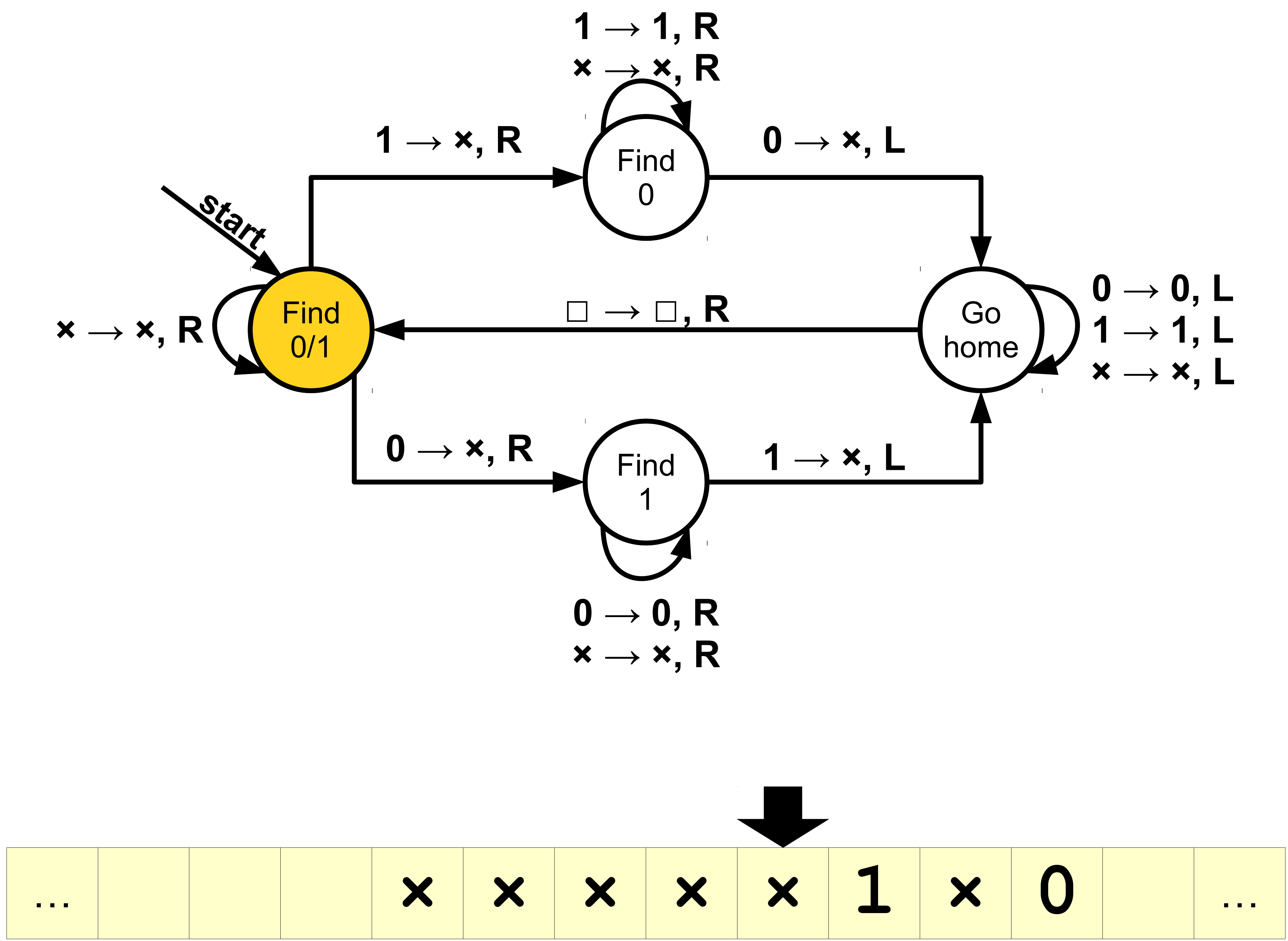

1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…



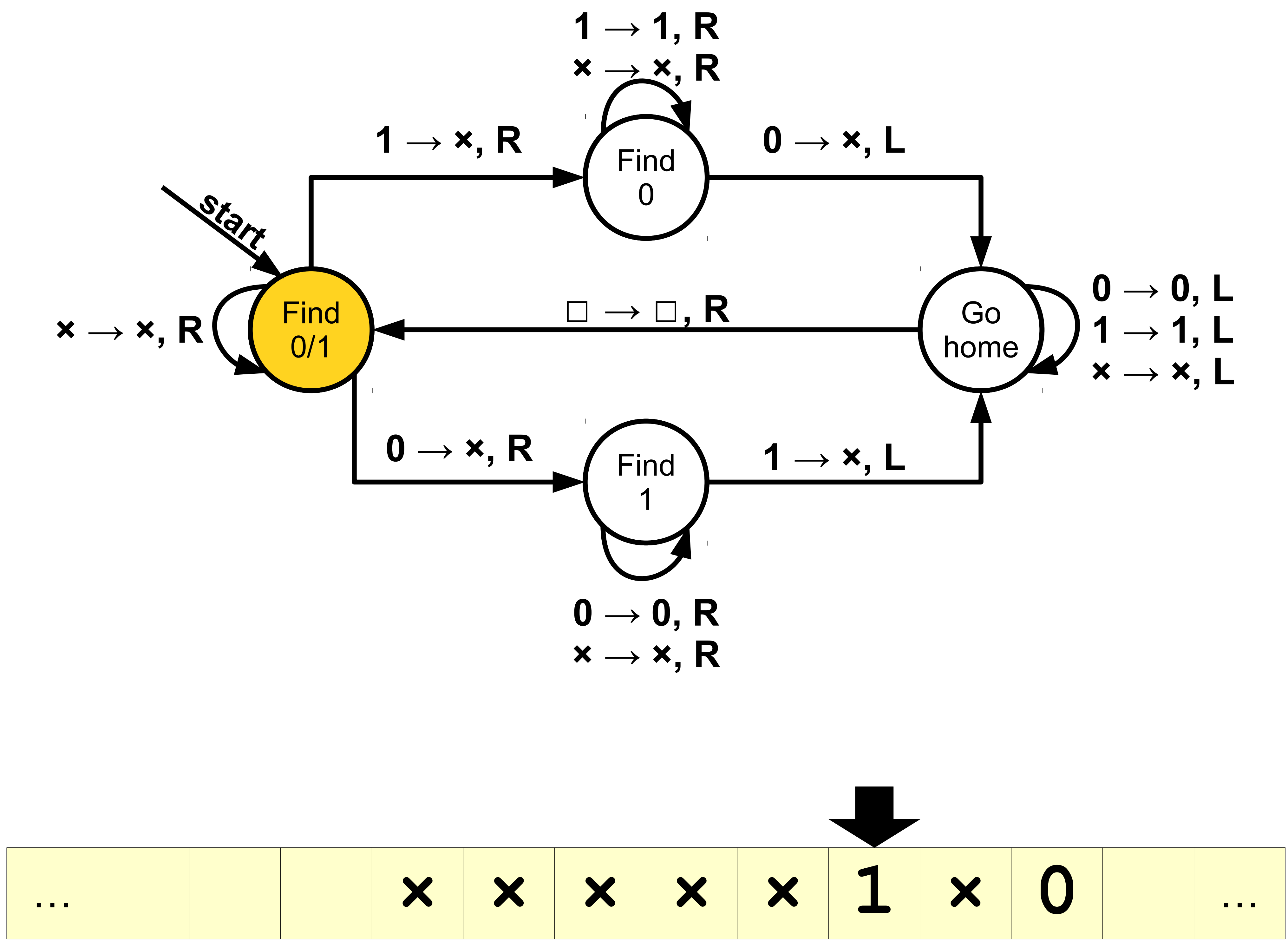
1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…



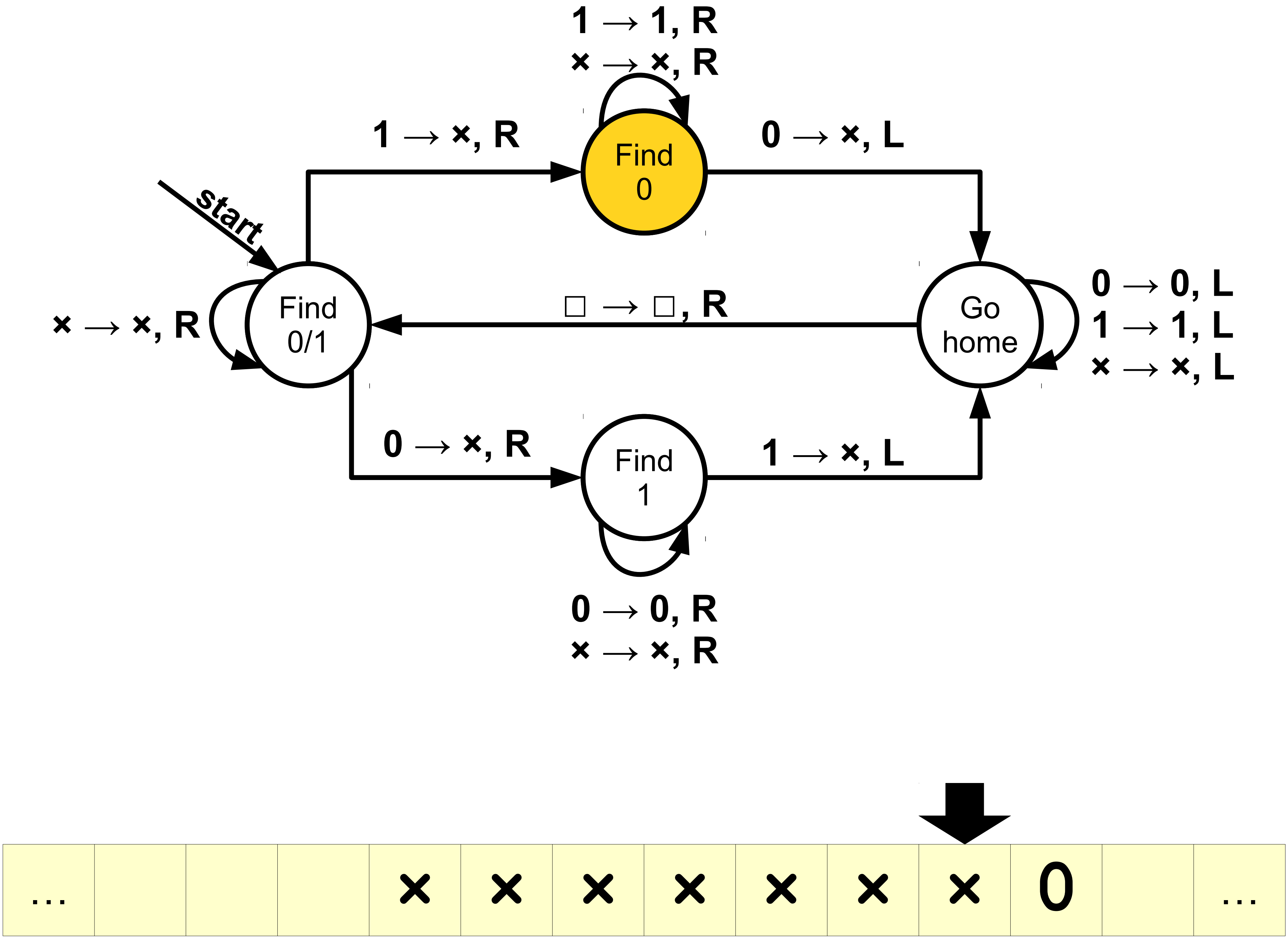
1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
… × × × × × 1 × 0 …



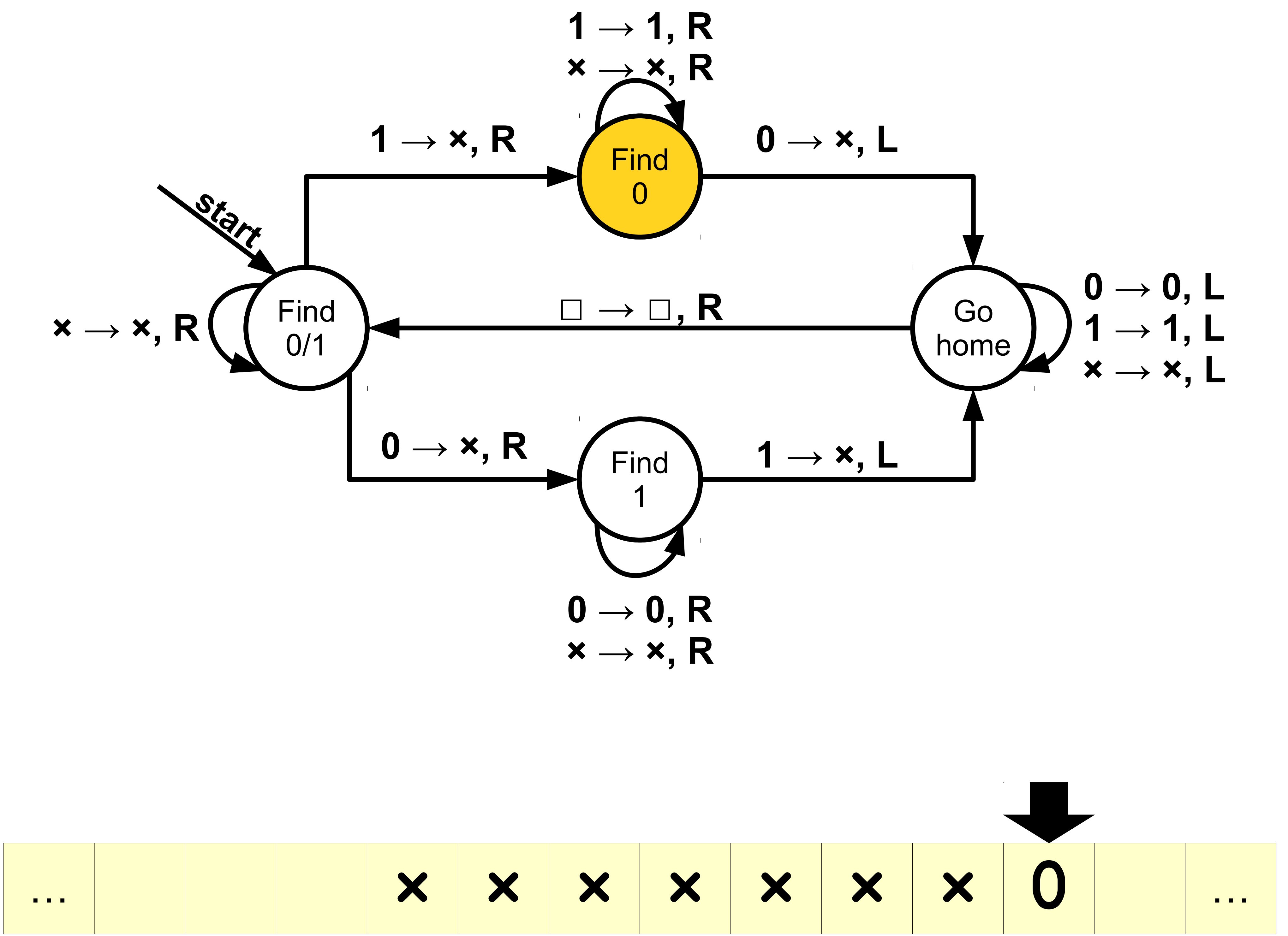
start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
… × × × × × 1 × 0 …



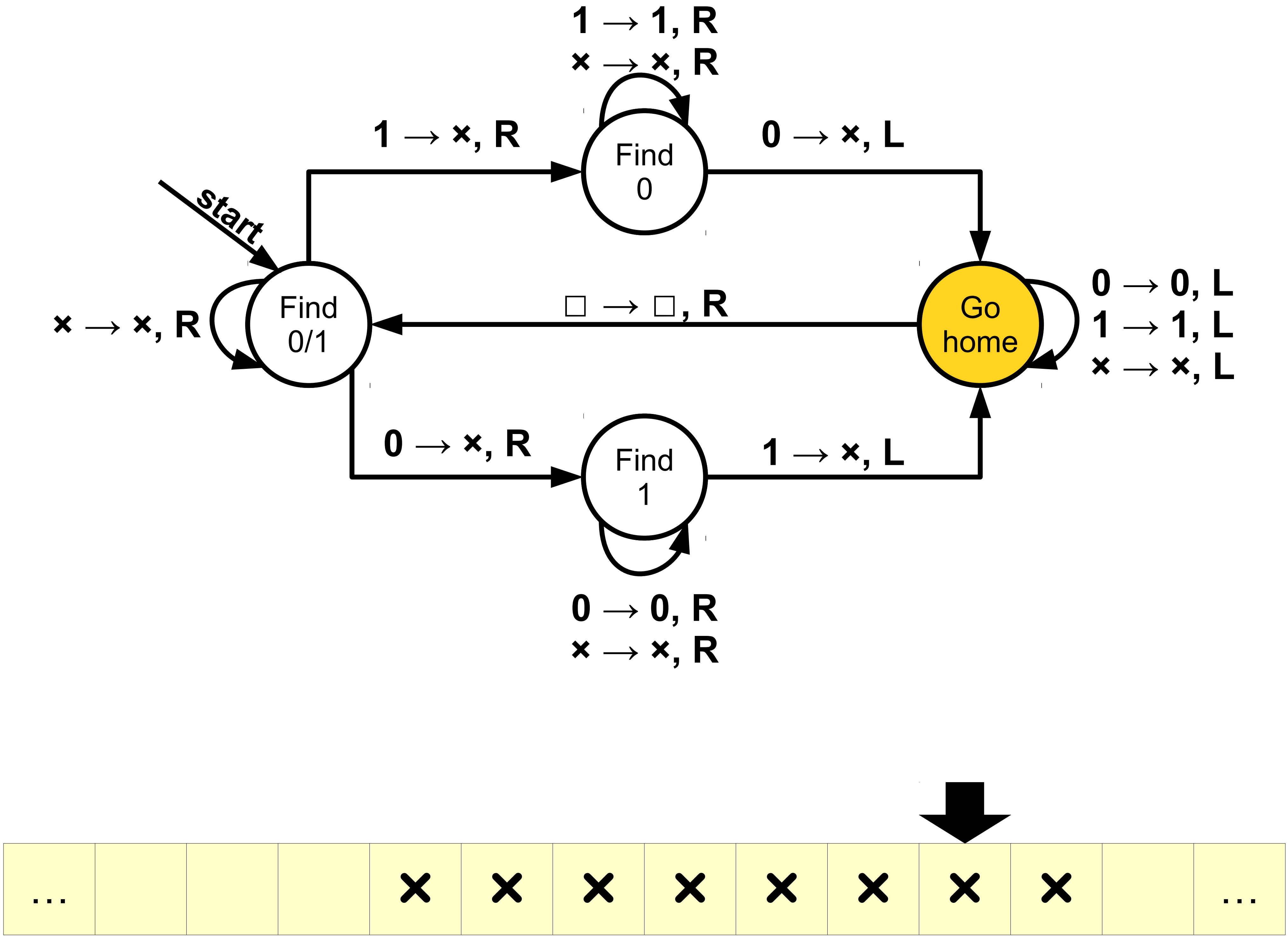
start
Find 0/1
Find 0
Find 1
Go home
1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
× → ×, R
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
1
×
0
…

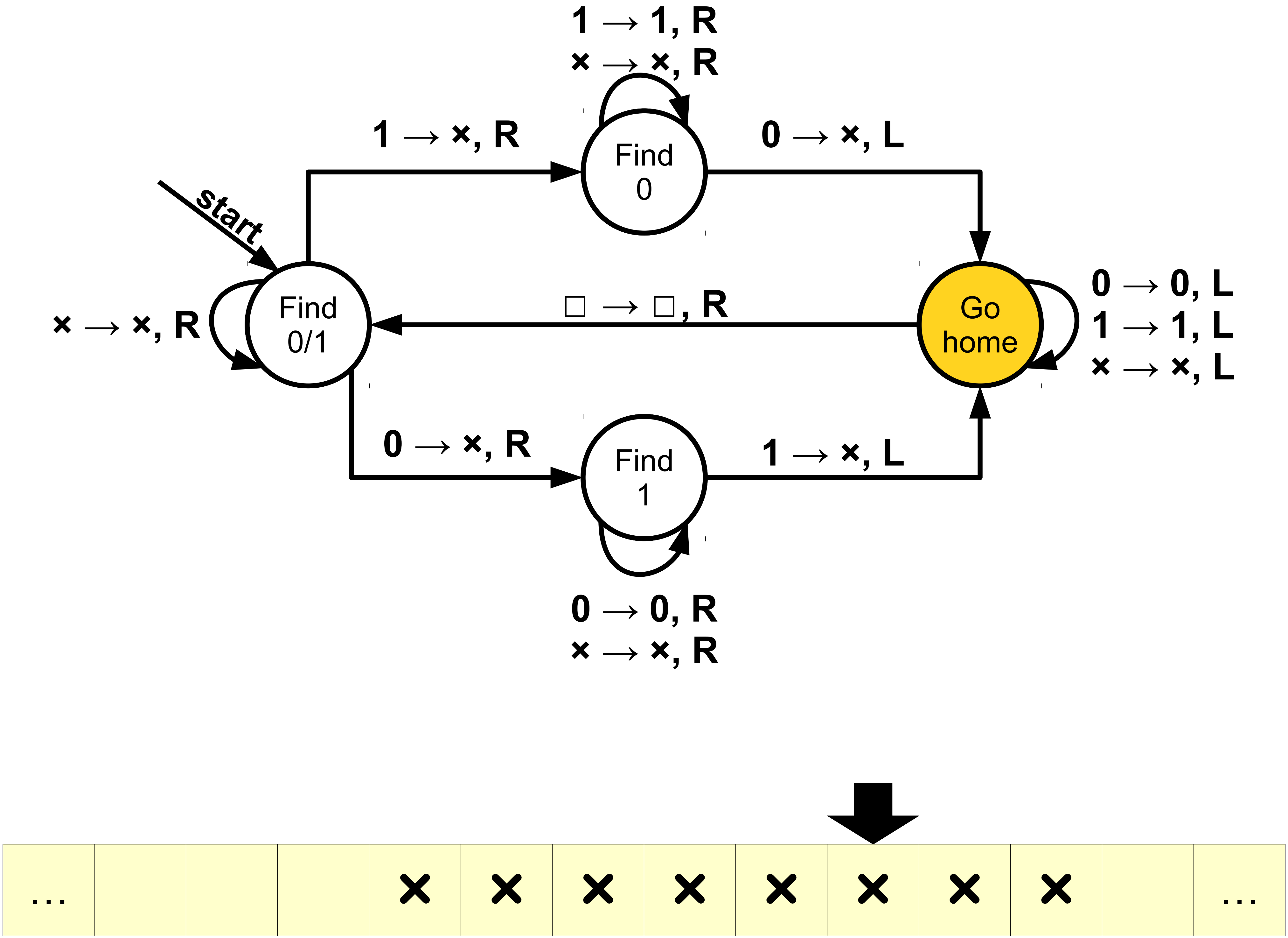


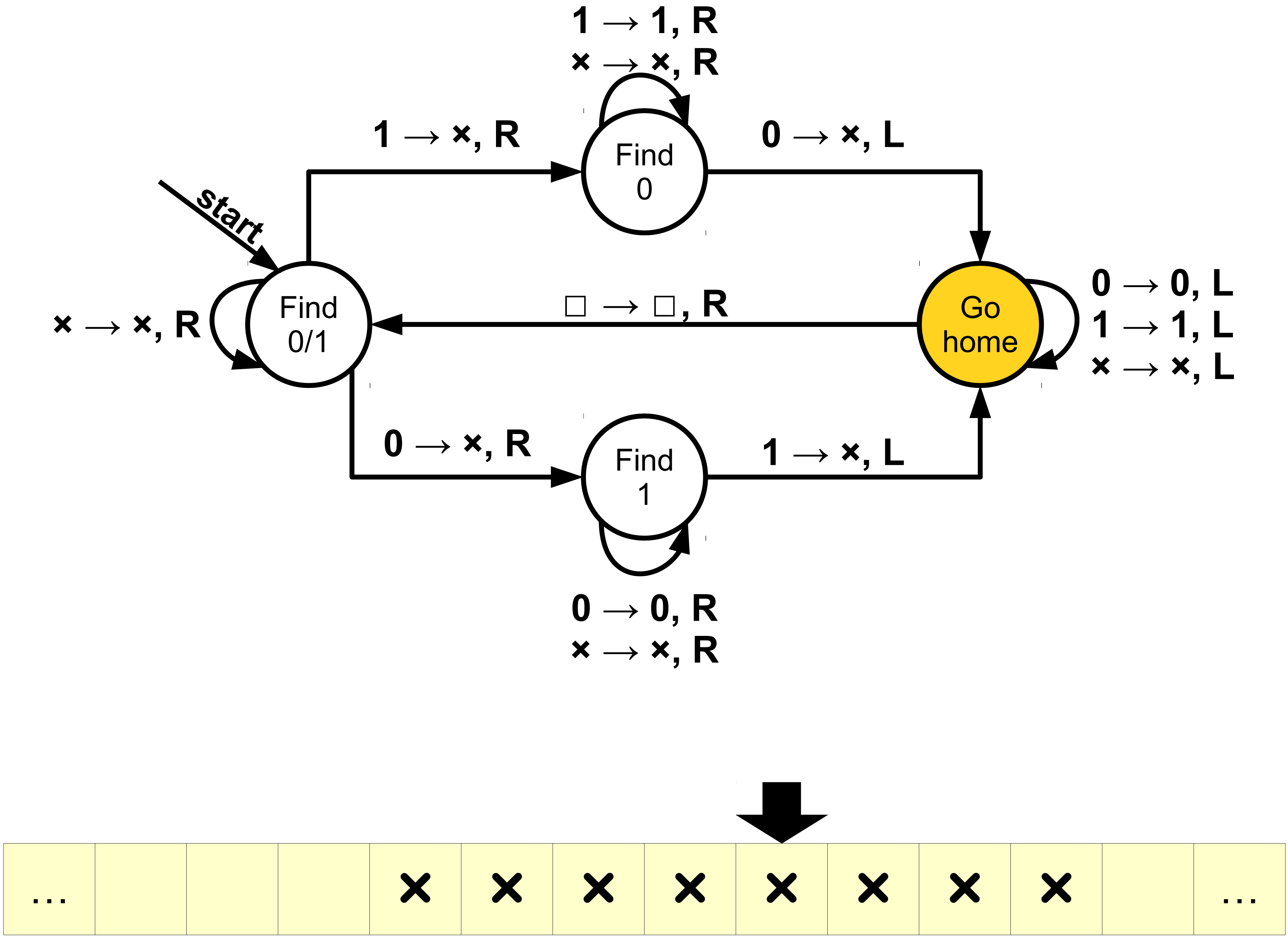
start
Find 0/1
Find 0
Find 1
Go home
1 → ×, R
0 → ×, R
0 → ×, L
1 → ×, L
□ → □, R
× → ×, R
1 → 1, R
× → ×, R
0 → 0, R
× → ×, R
0 → 0, L
1 → 1, L
× → ×, L
…
× × × × × × × 0
…



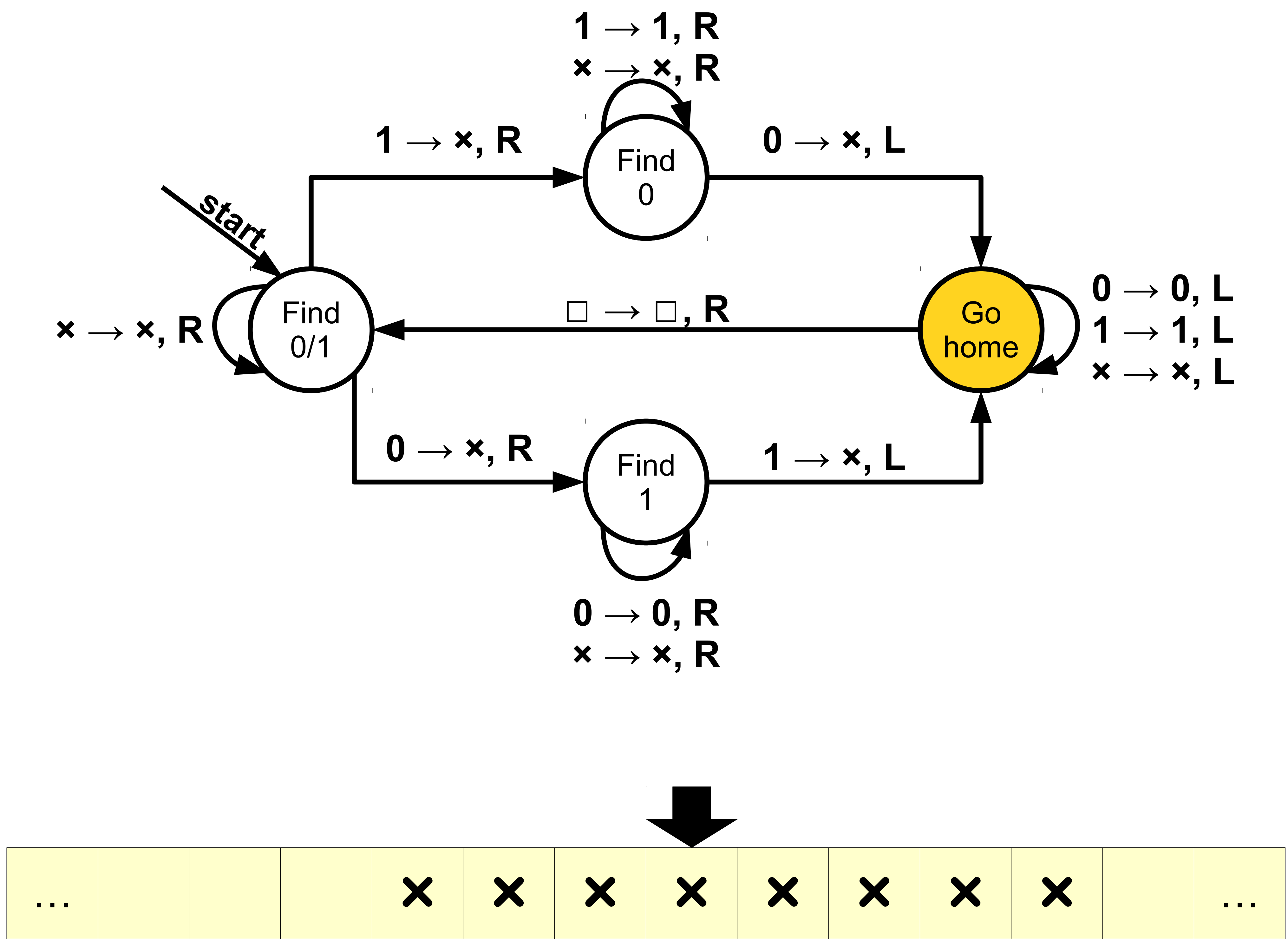
start
Find 0/1
Find 0
Find 1
Go home
1 → ×, R
1 → 1, R
× → ×, R
0 → ×, L
× → ×, R
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
0 → 0, R
× → ×, R
…
× × × × × × × 0
…


1 → 1, R
× → ×, R
1 → ×, R
Find
0
0 → ×, L
start
× → ×, R
Find
0/1
□ → □, R
Go
home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find
1
1 → ×, L
0 → 0, R
× → ×, R
...
×
×
×
×
×
×
×
×
...

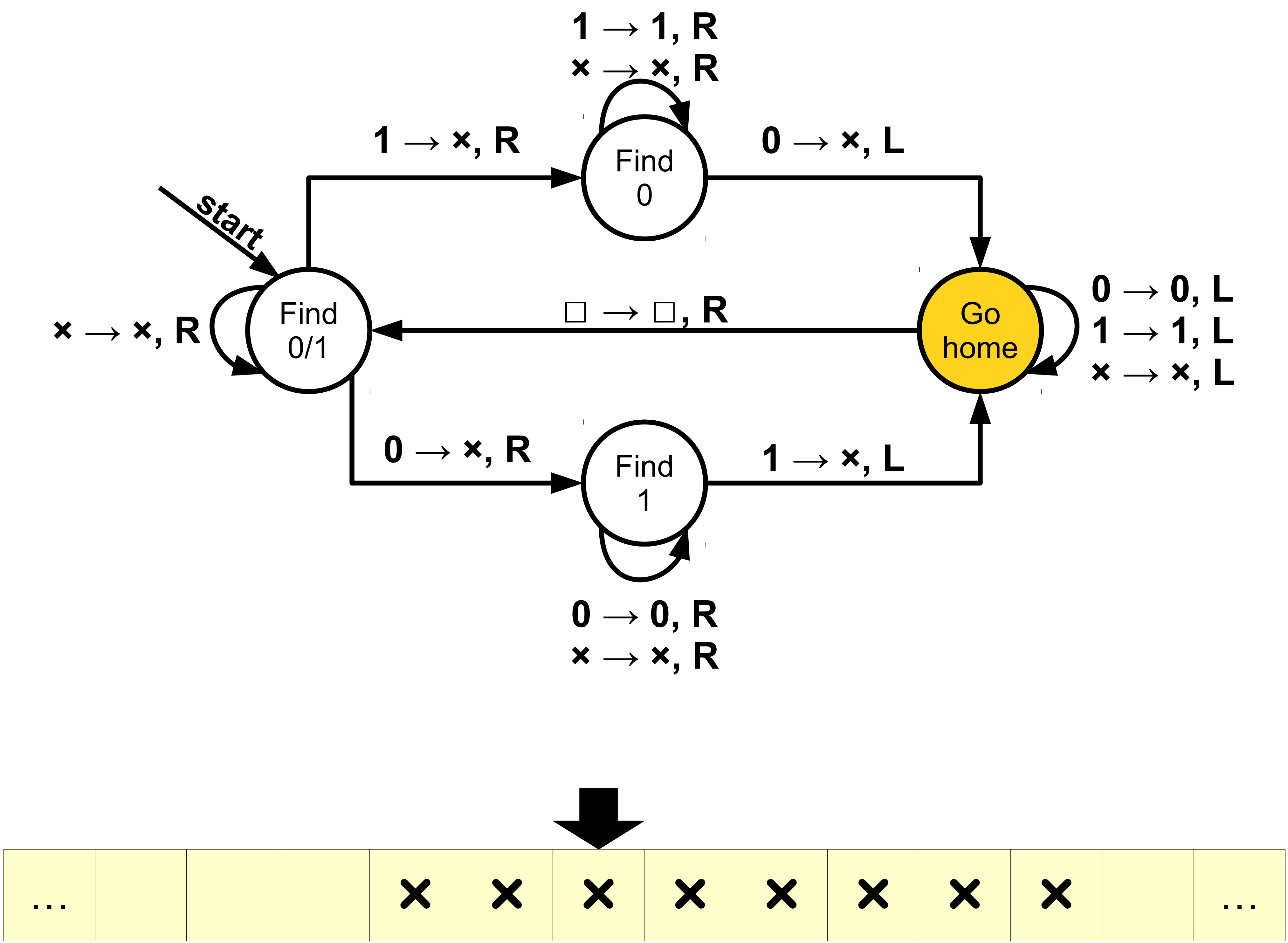

1 → 1, R
× → ×, R
1 → ×, R
Find
0
0 → ×, L
start
× → ×, R
Find
0/1
□ → □, R
Go
home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find
1
1 → ×, L
0 → 0, R
× → ×, R
...
×
×
×
×
×
×
×
×
...

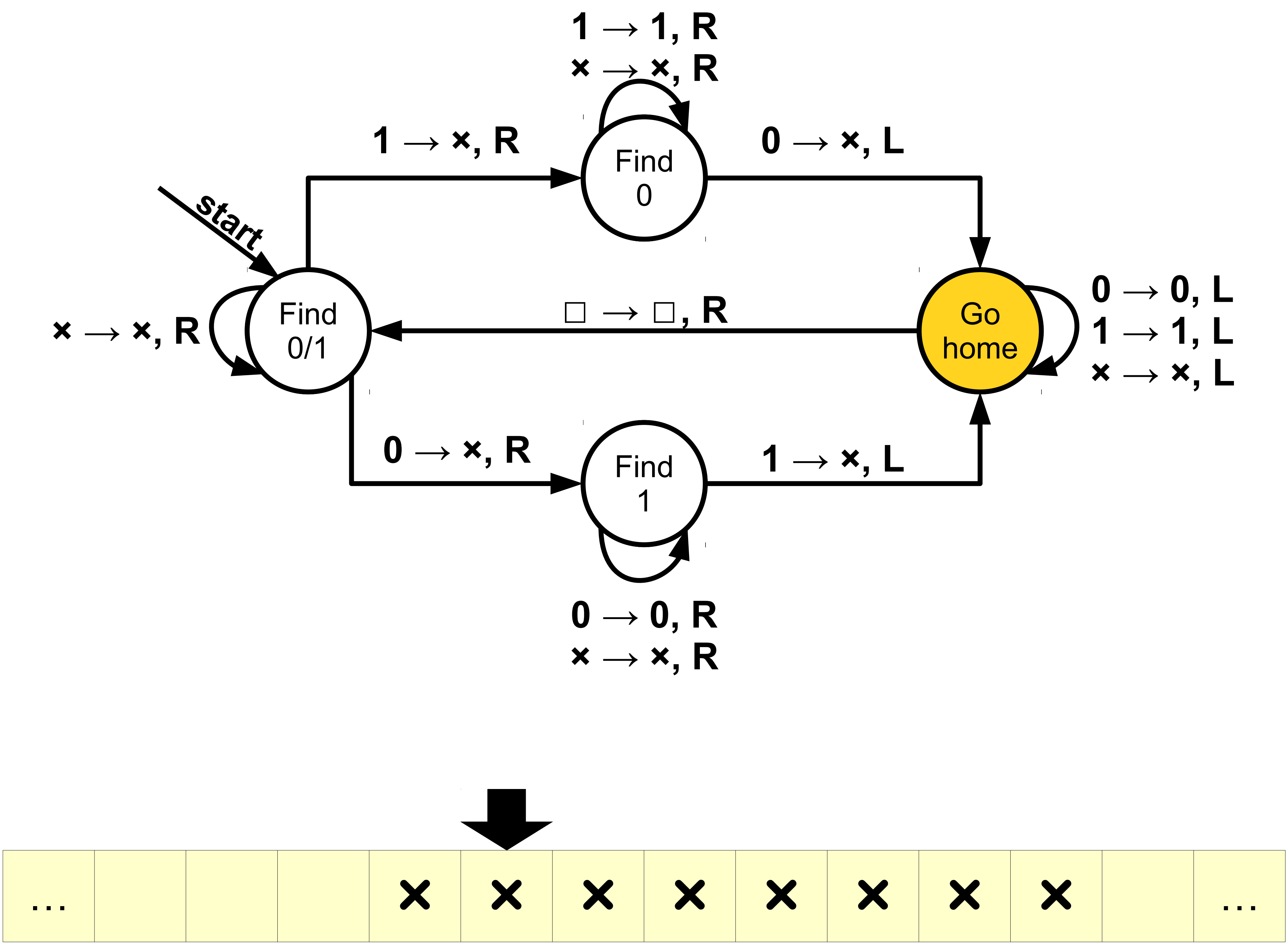


start
Find 0/1
Find 0
Find 1
Go home
× → ×, R
1 → ×, R
0 → ×, R
1 → 1, R
× → ×, R
0 → ×, L
0 → 0, R
× → ×, R
1 → ×, L
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
…
× × × × × × × ×
…

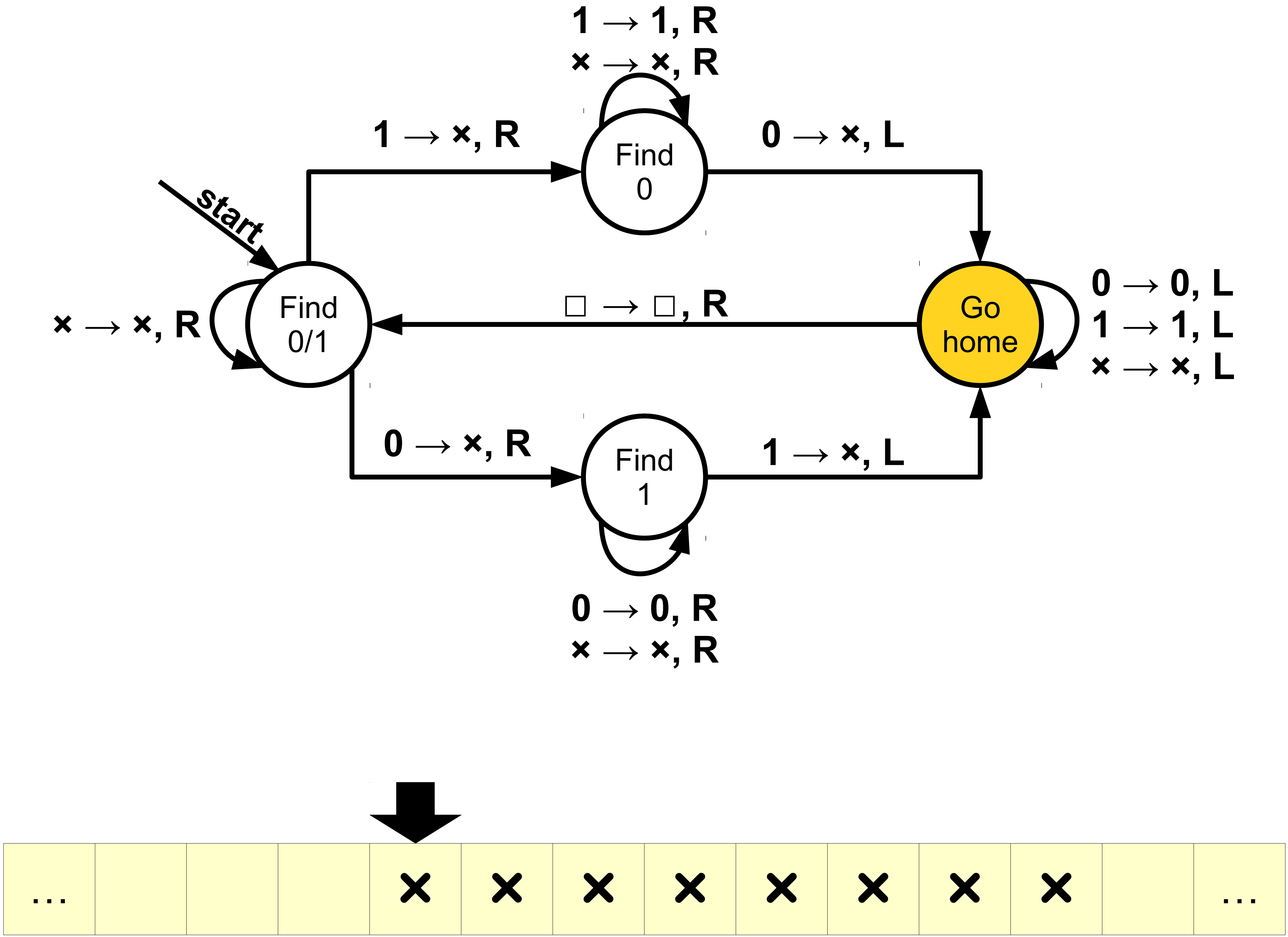


1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
× × × × × × × ×
…

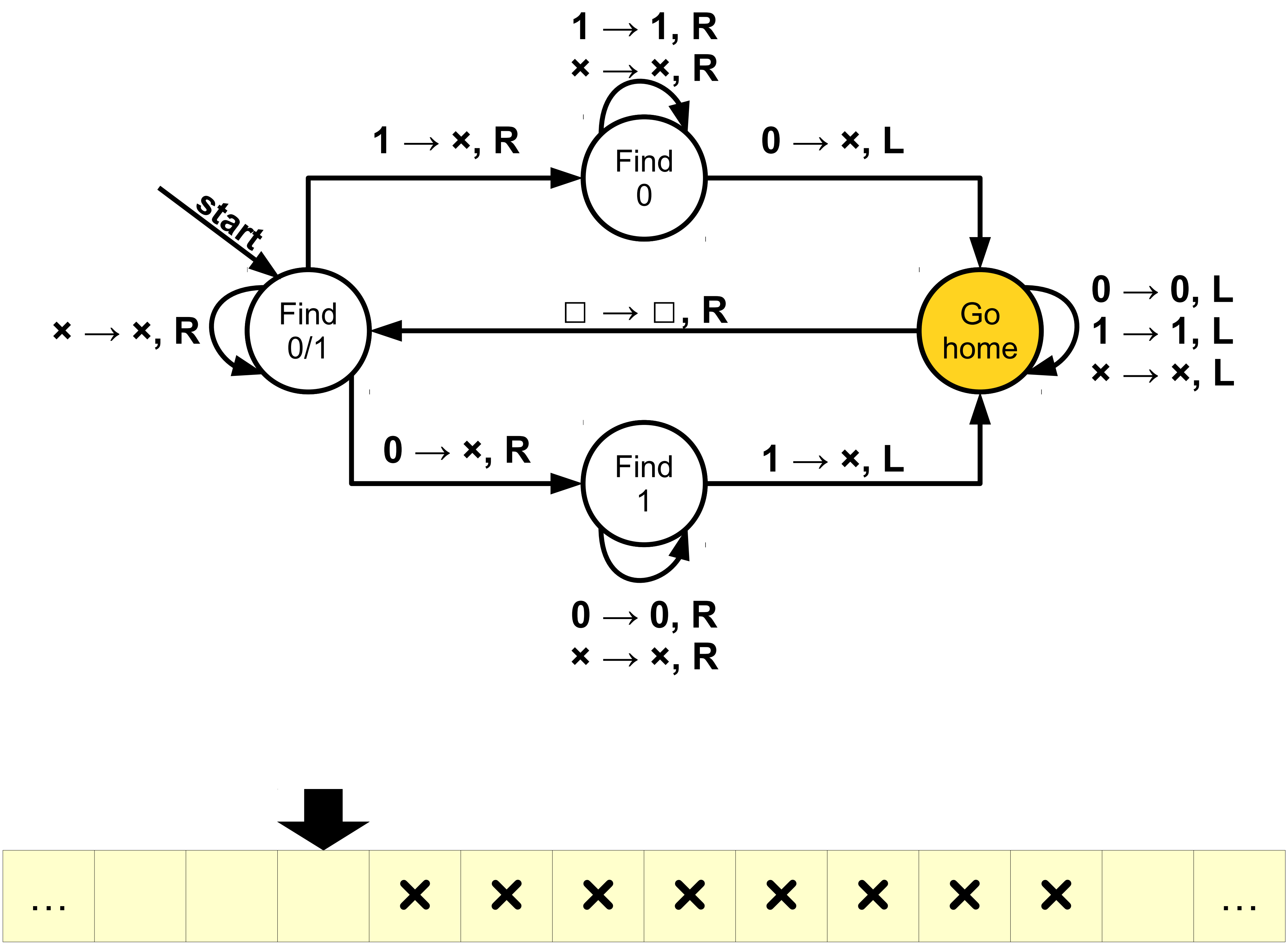

start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
...
×
×
×
×
×
×
×
×
...

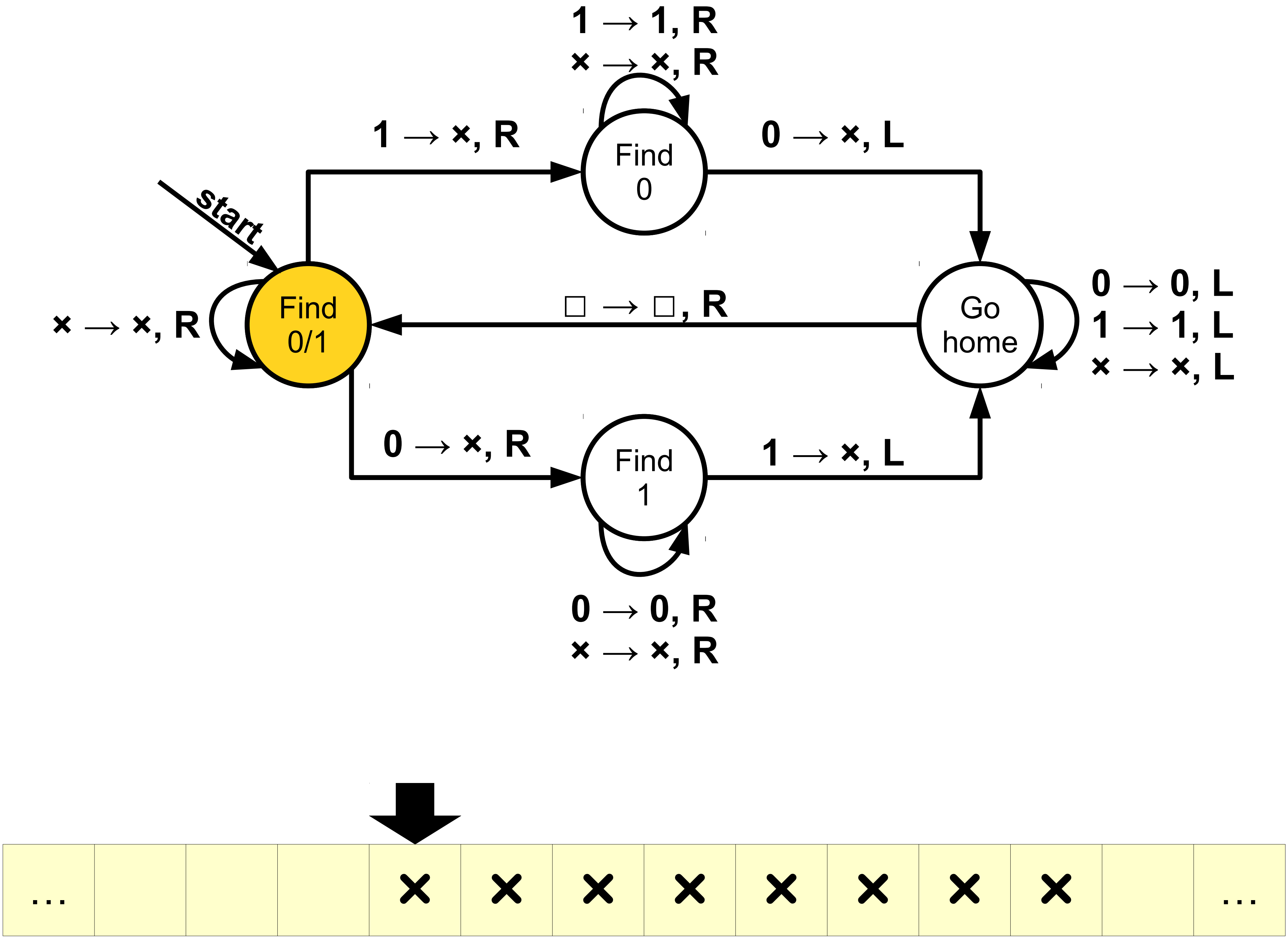


1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0/1
Find 0
Go home
Find 1
× → ×, R
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
0 → 0, R
× → ×, R
...
× × × × × × × ×
...

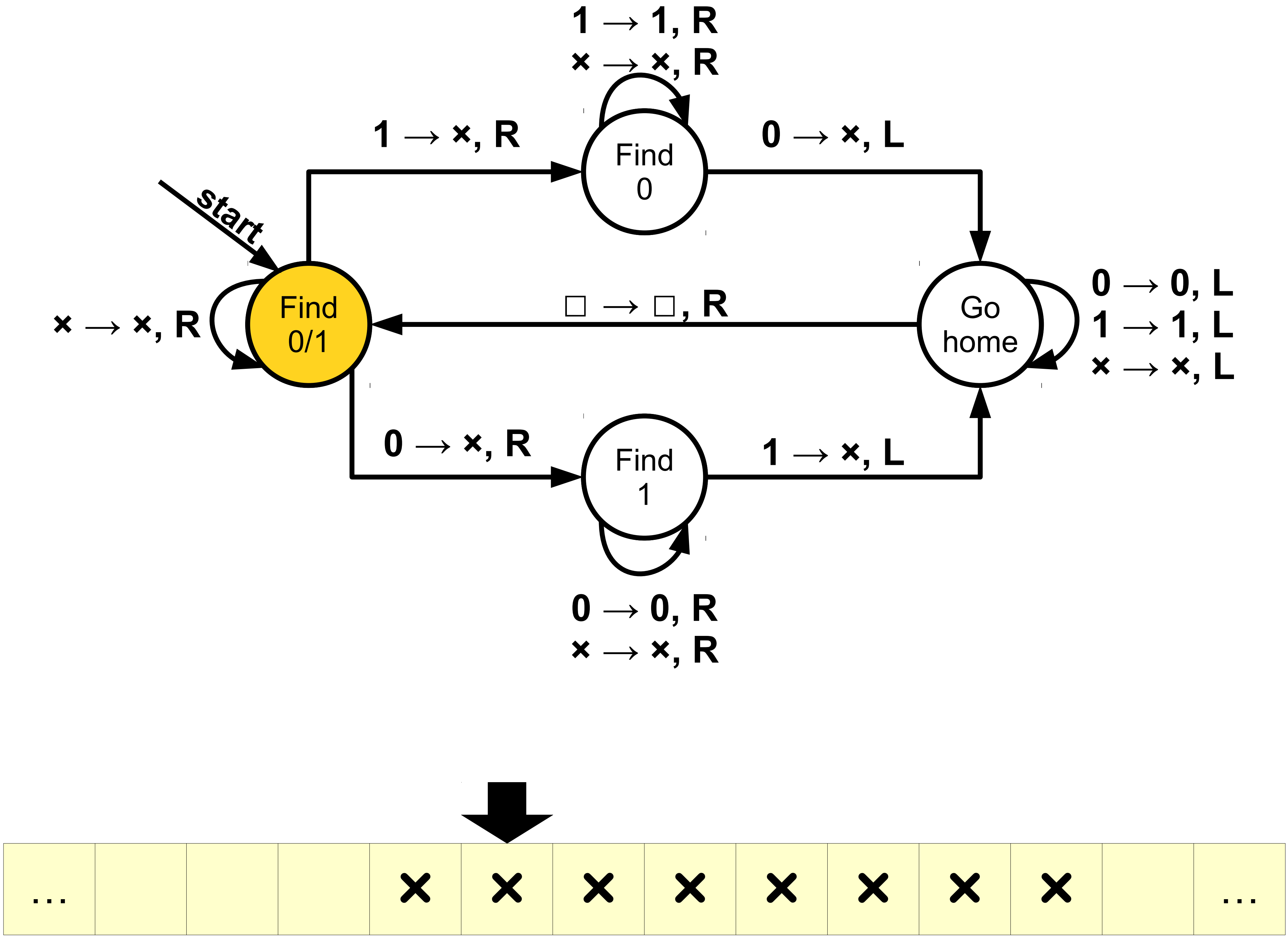


start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
… × × × × × × × × …

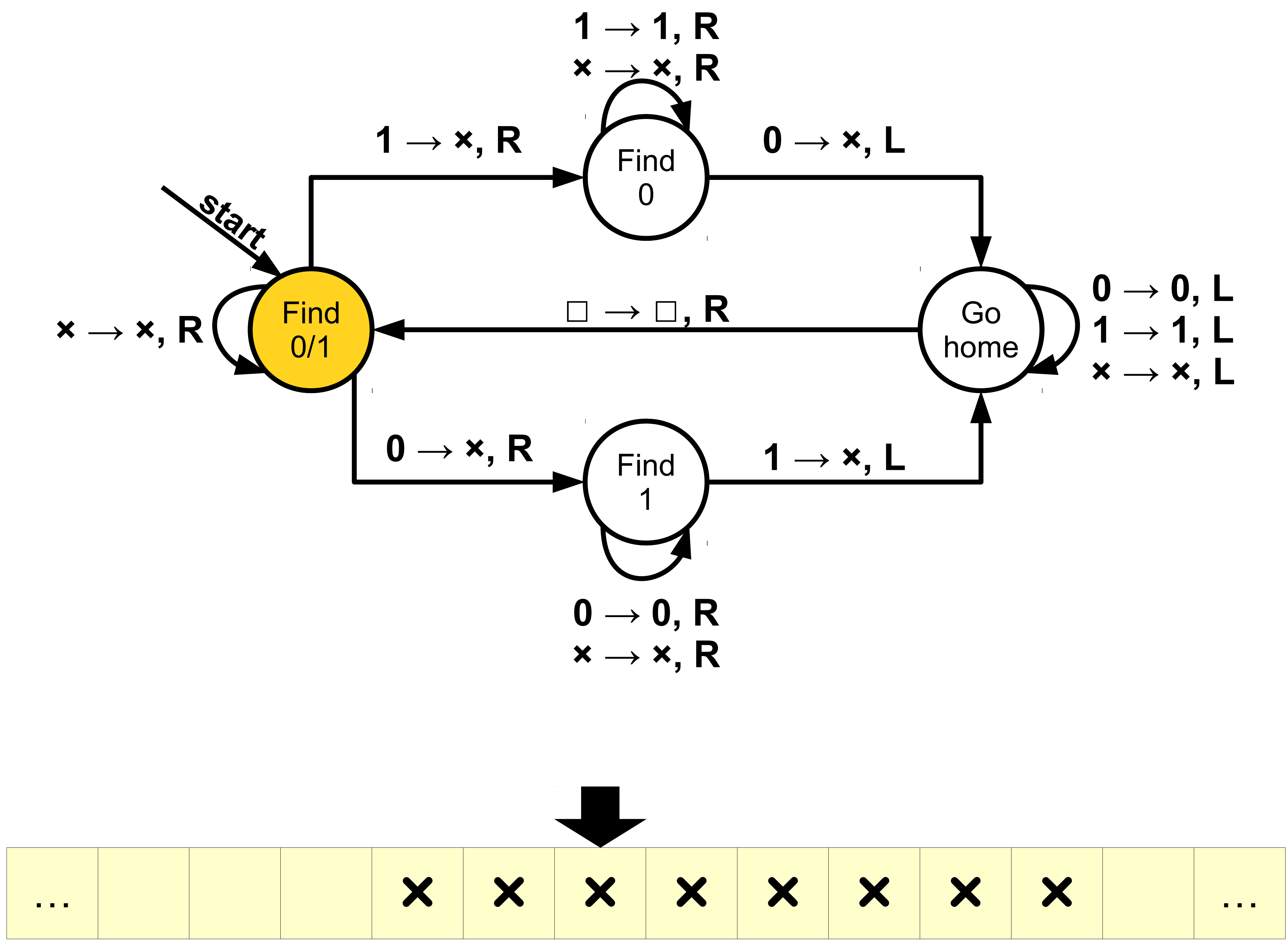

1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…



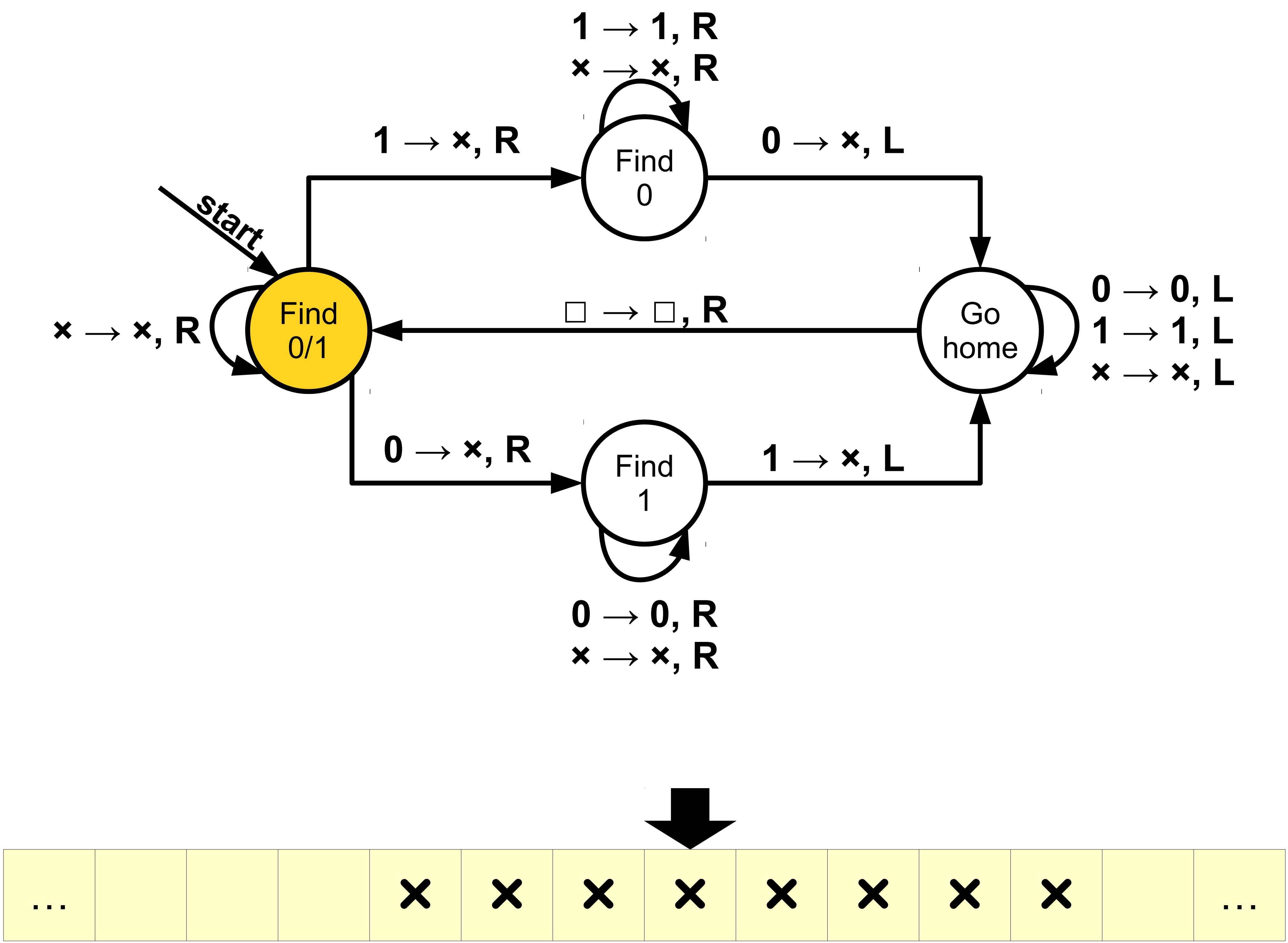
start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
…
× × × × × × × ×
…



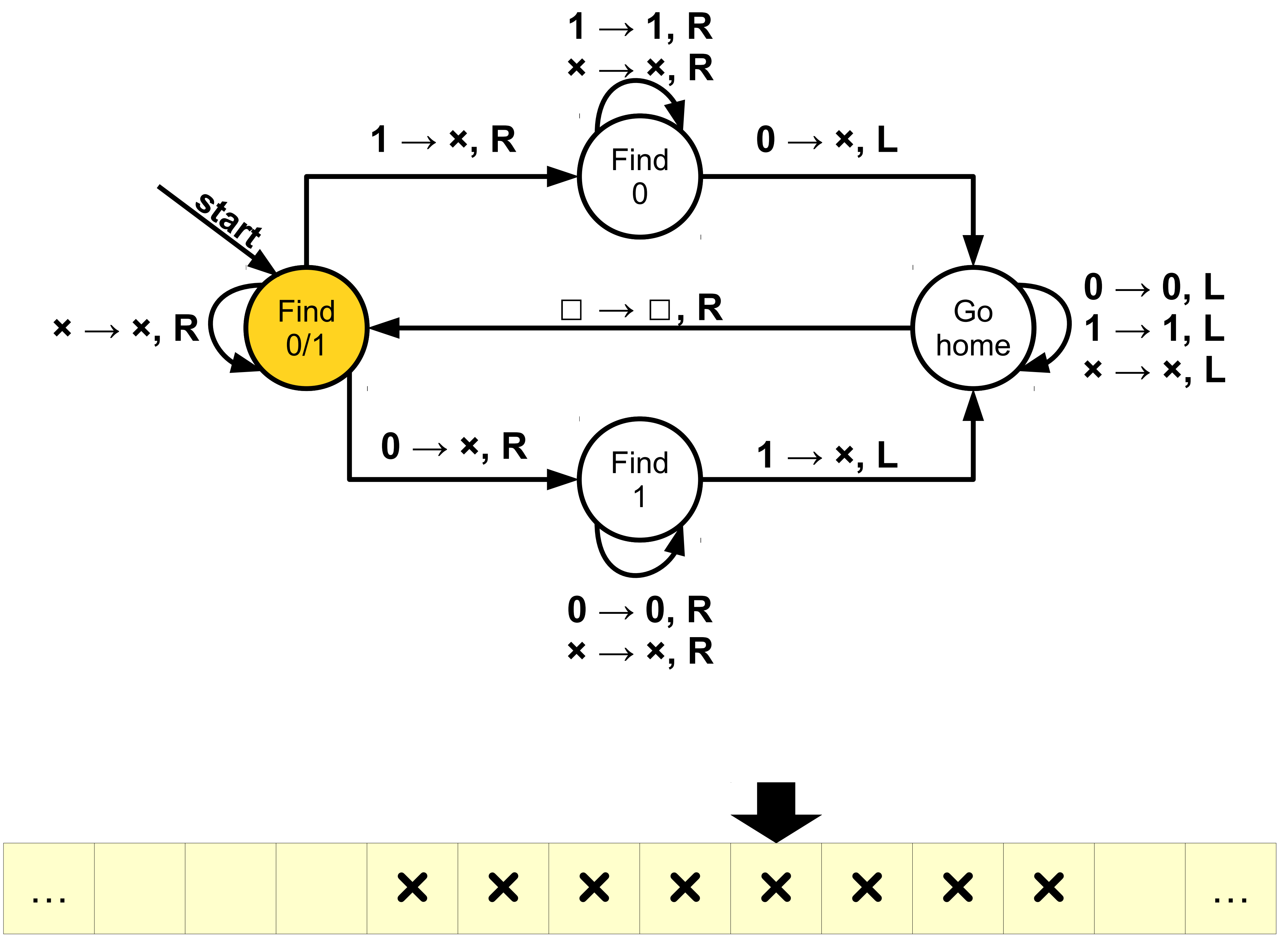
1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…

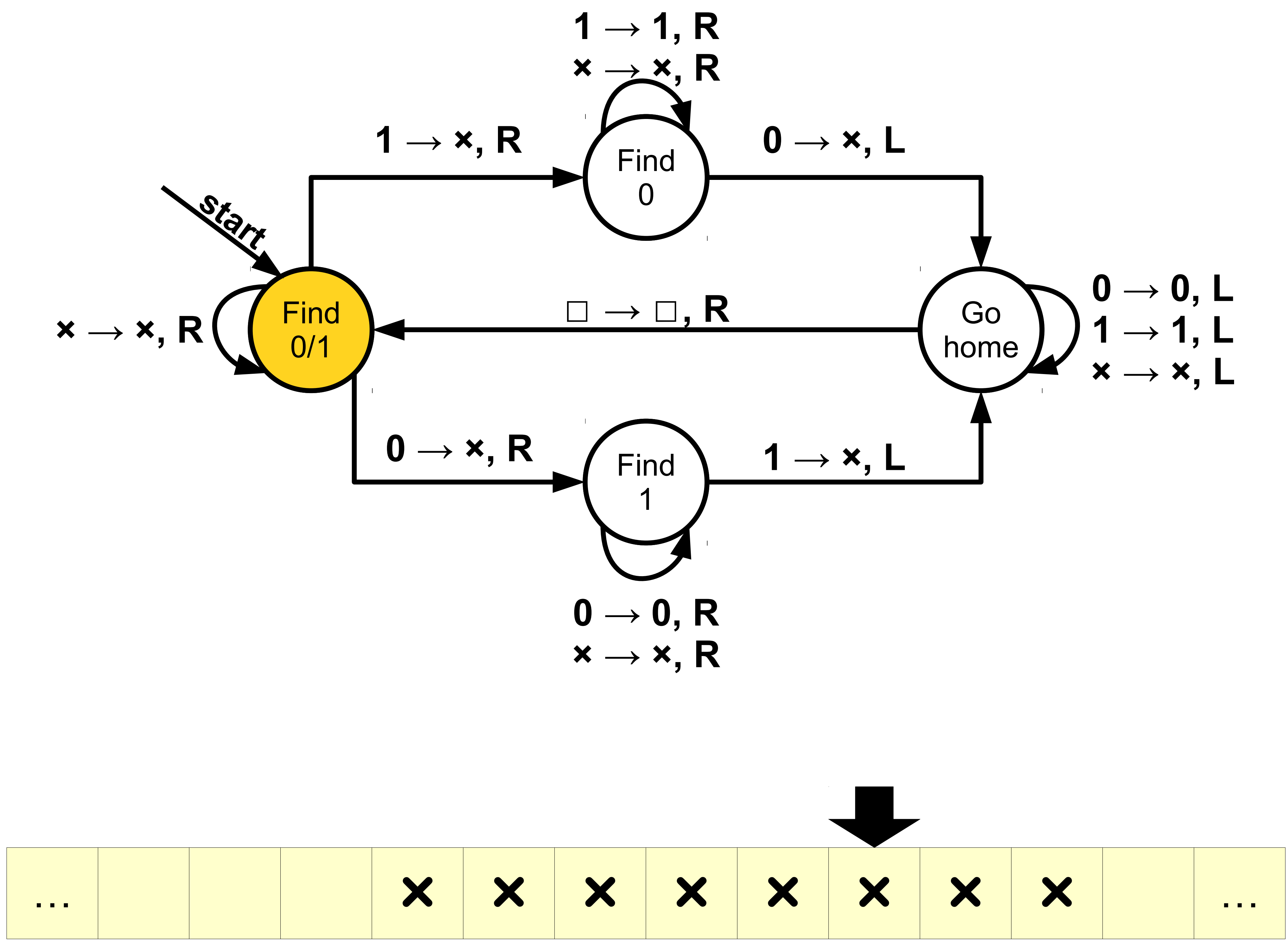


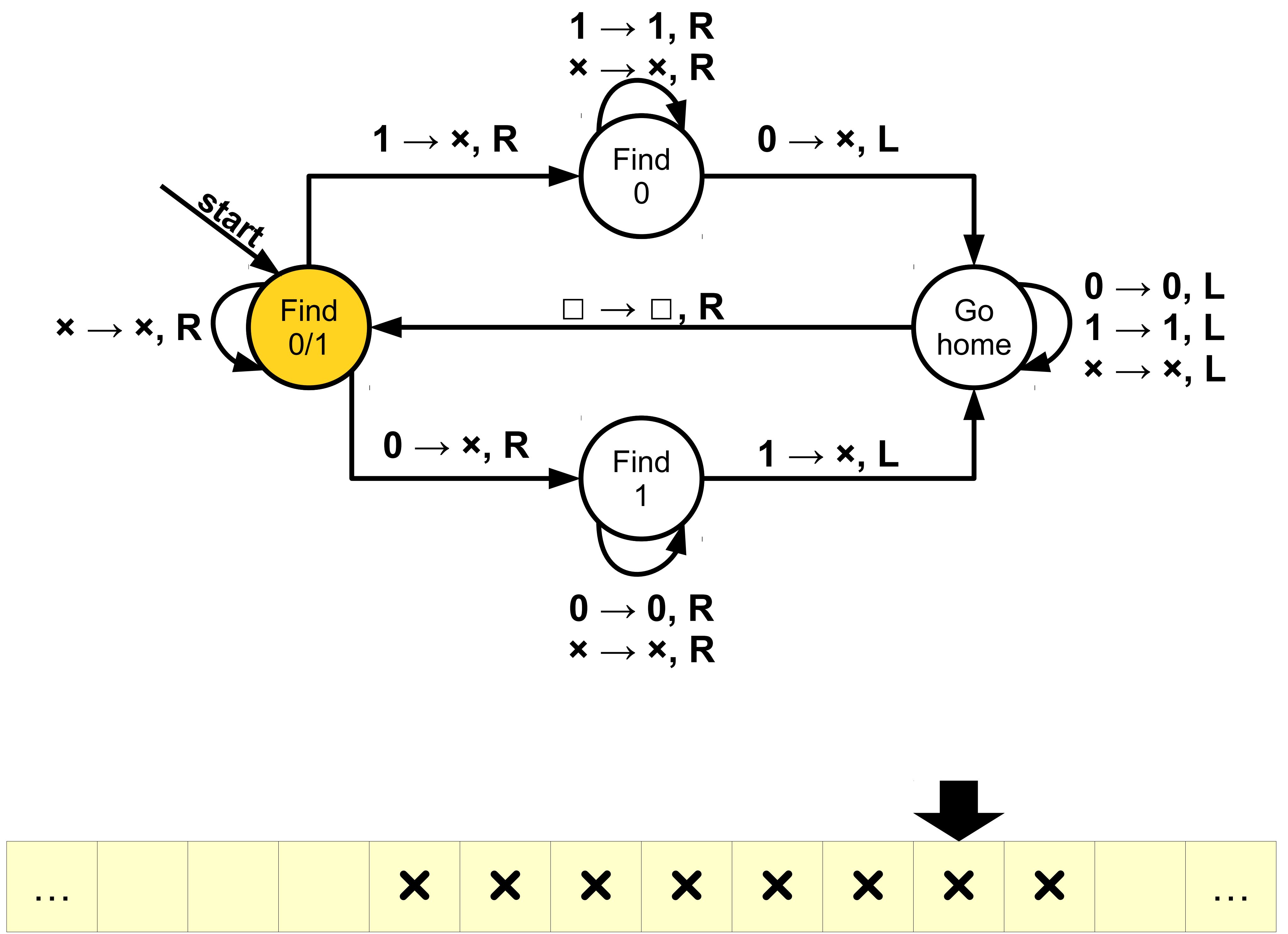
1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…



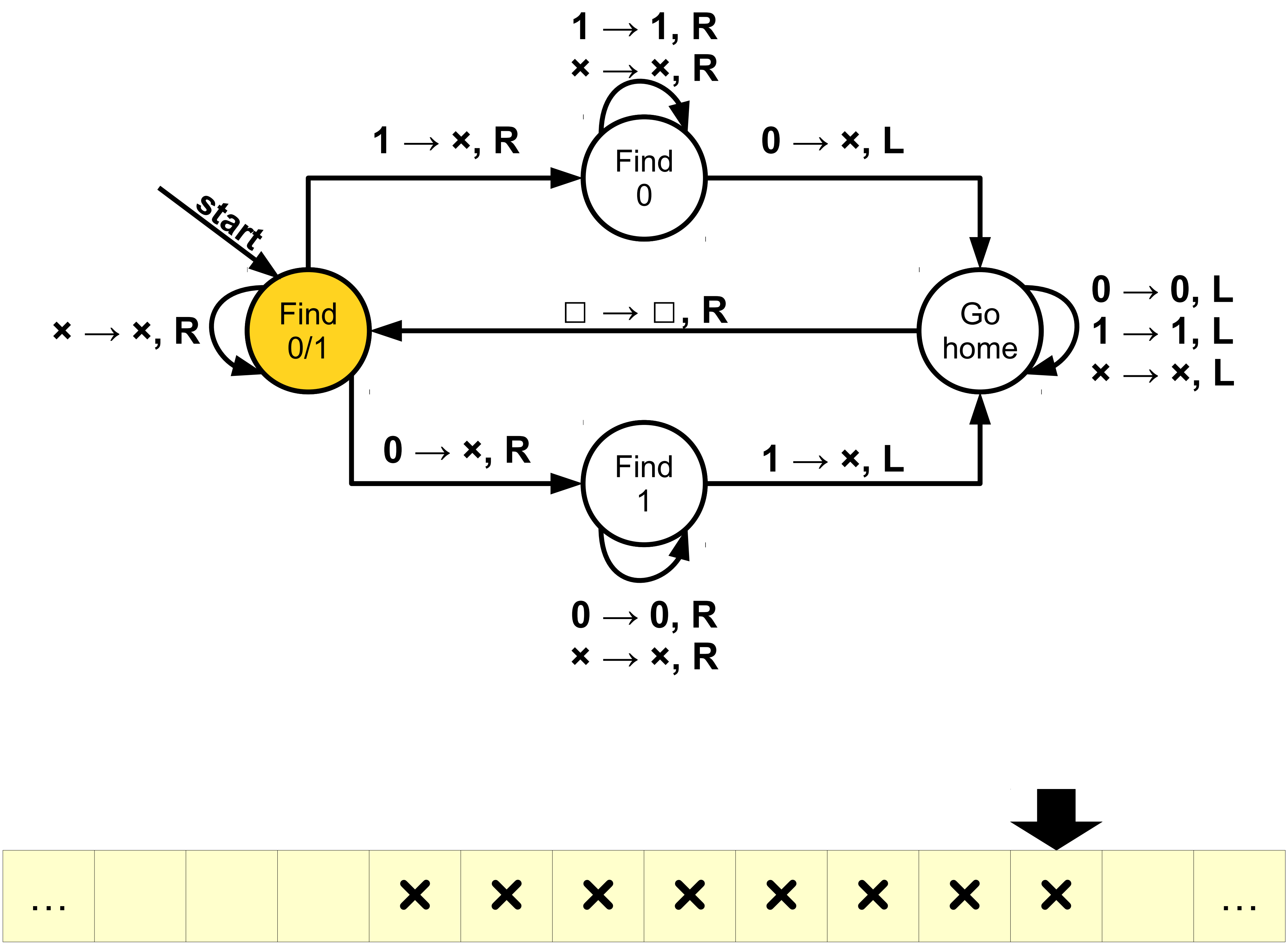
1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
Find 0
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
Find 1
0 → 0, R
× → ×, R
…
× × × × × × × ×
…


1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…

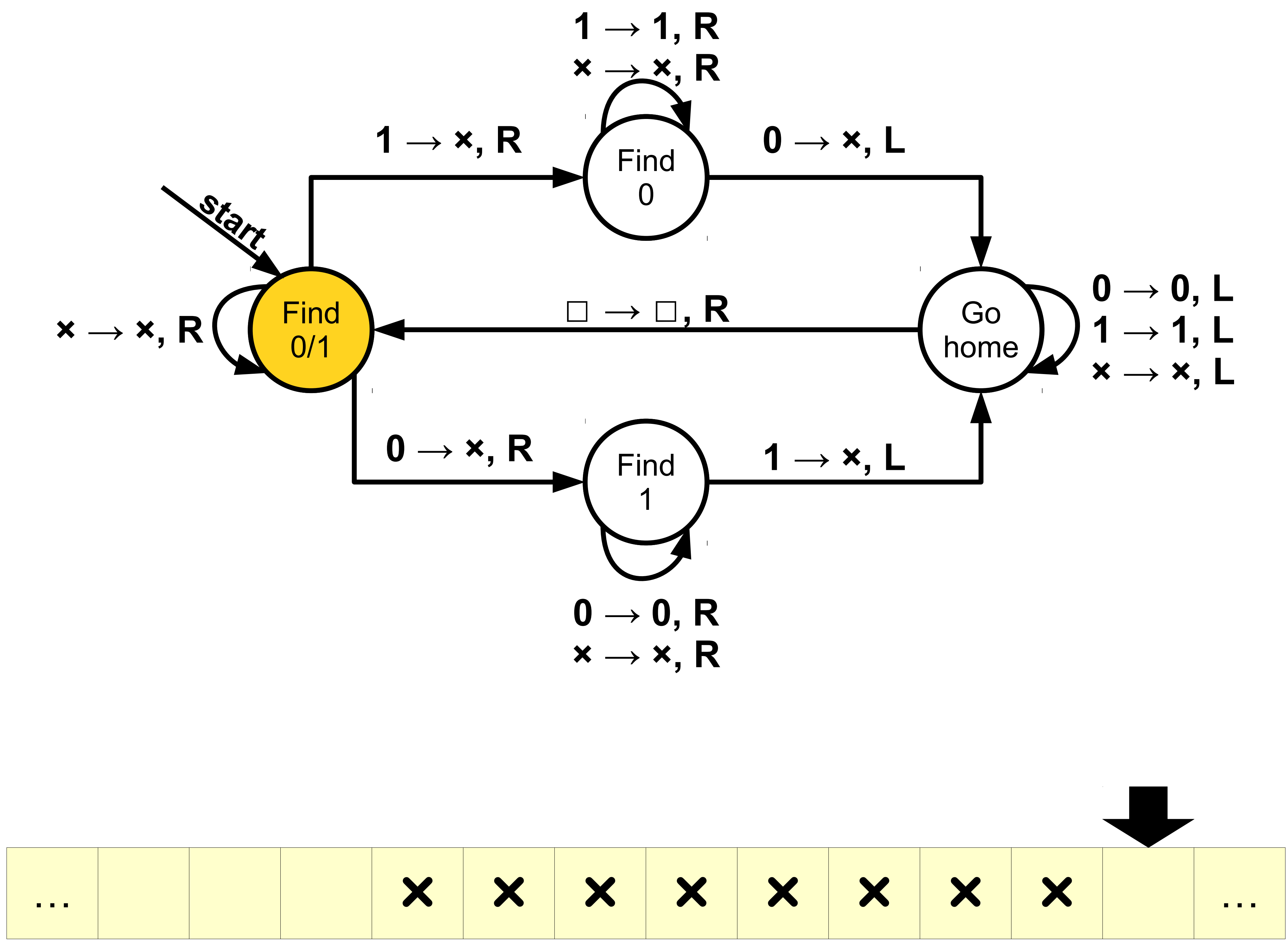

start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
…
× × × × × × × ×
…

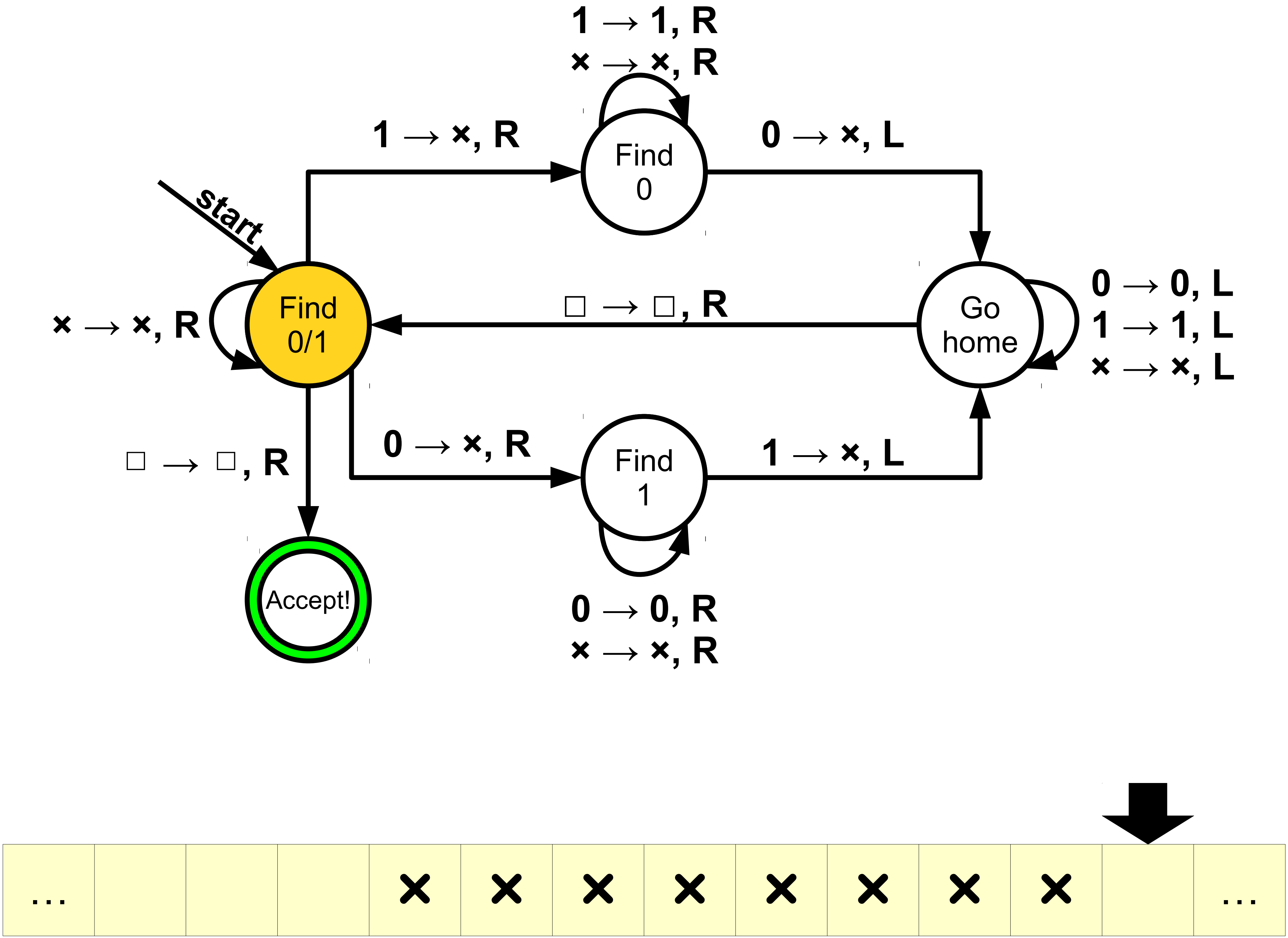


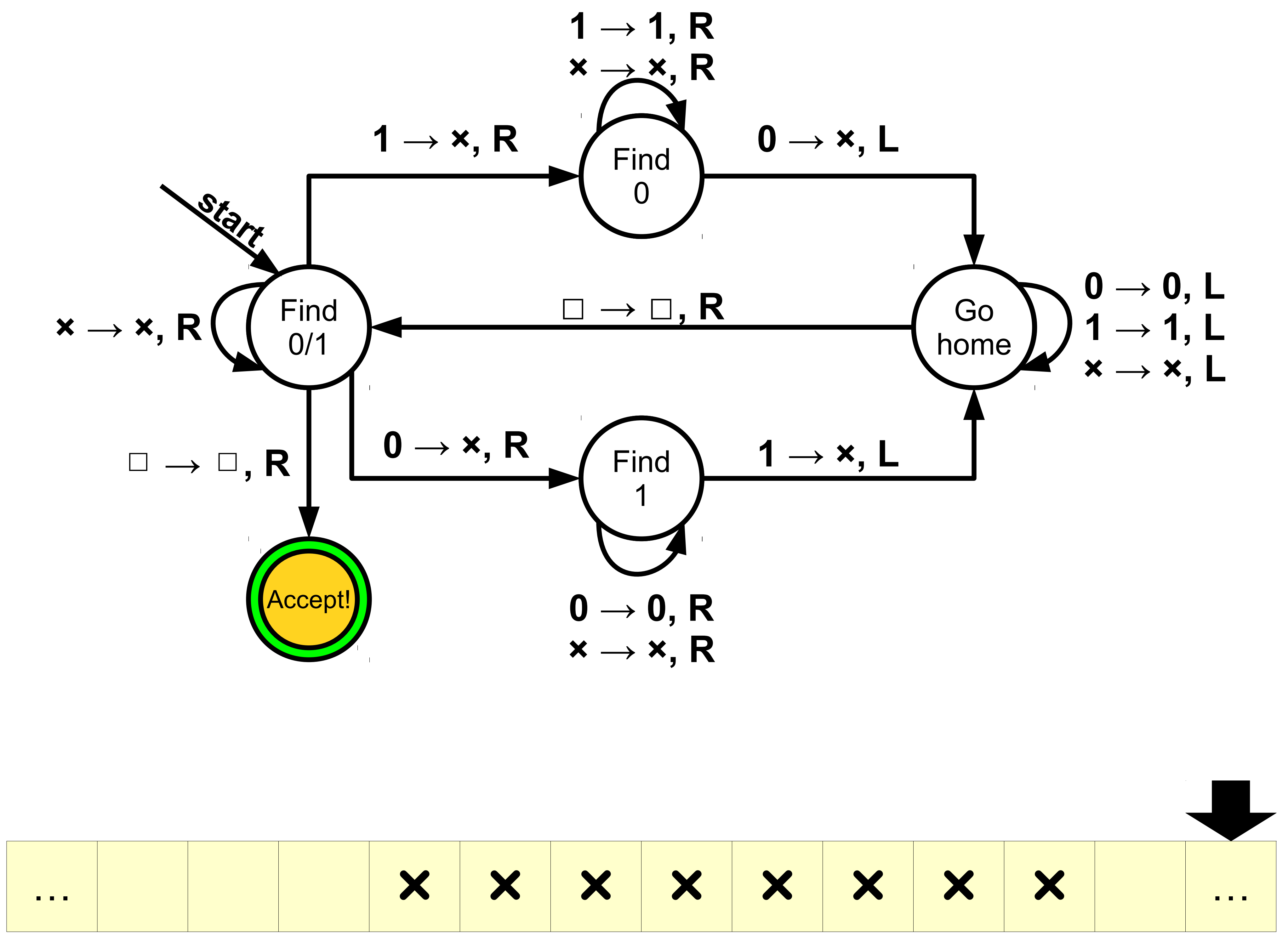
1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…



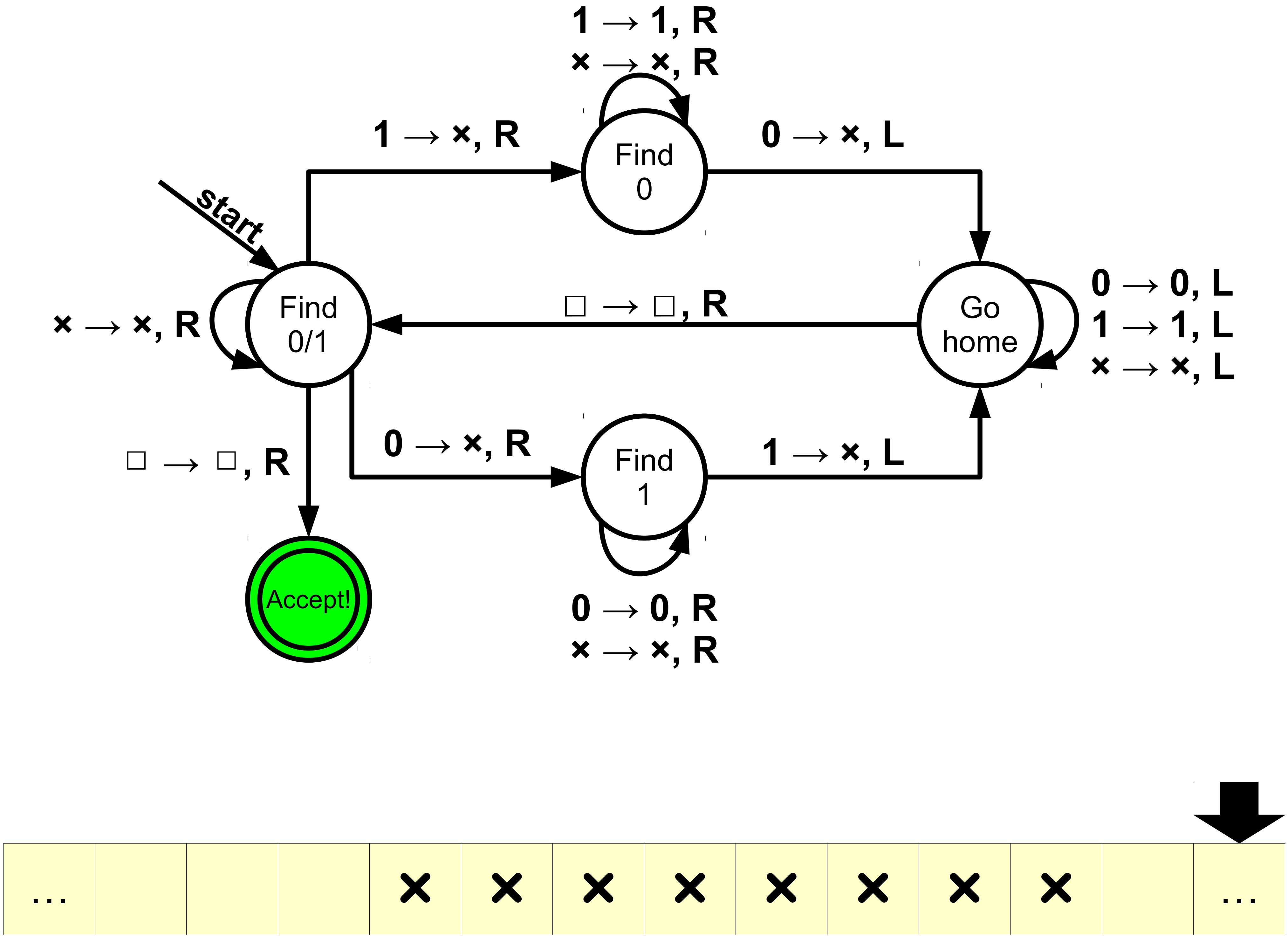
Find 0/1
Find 0
Find 1
Go home
start
1 → ×, R
0 → ×, R
× → ×, R
1 → 1, R
× → ×, R
0 → 0, R
× → ×, R
0 → ×, L
1 → ×, L
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
…
× × × × × × × ×
…


1 → 1, R
× → ×, R
1 → ×, R
0 → ×, L
start
Find 0
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…

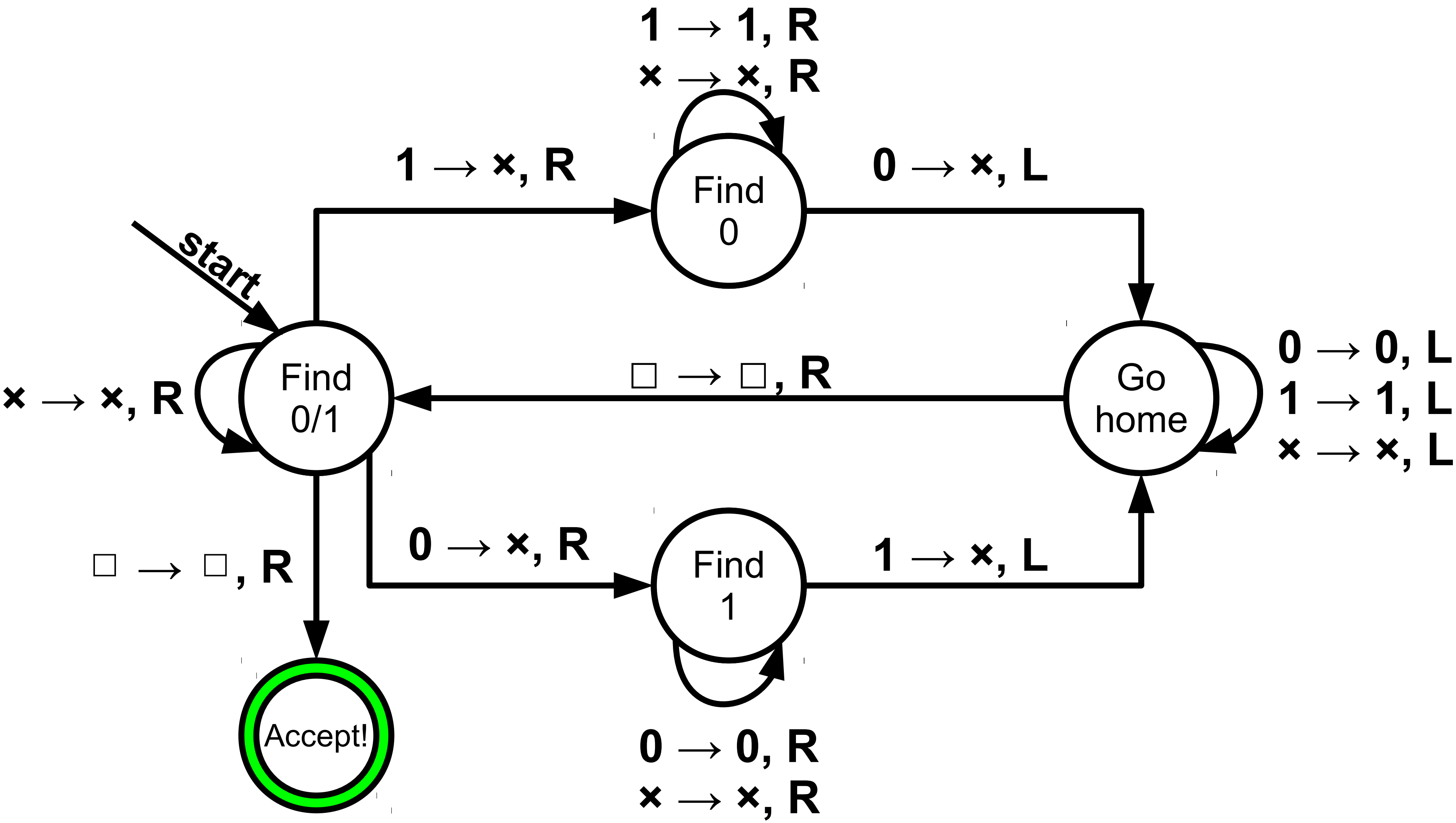

start
Find 0/1
Find 0
Find 1
Go home
Accept!
1 → ×, R
1 → 1, R
× → ×, R
0 → ×, L
× → ×, R
□ → □, R
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
1 → ×, L
□ → □, R
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…



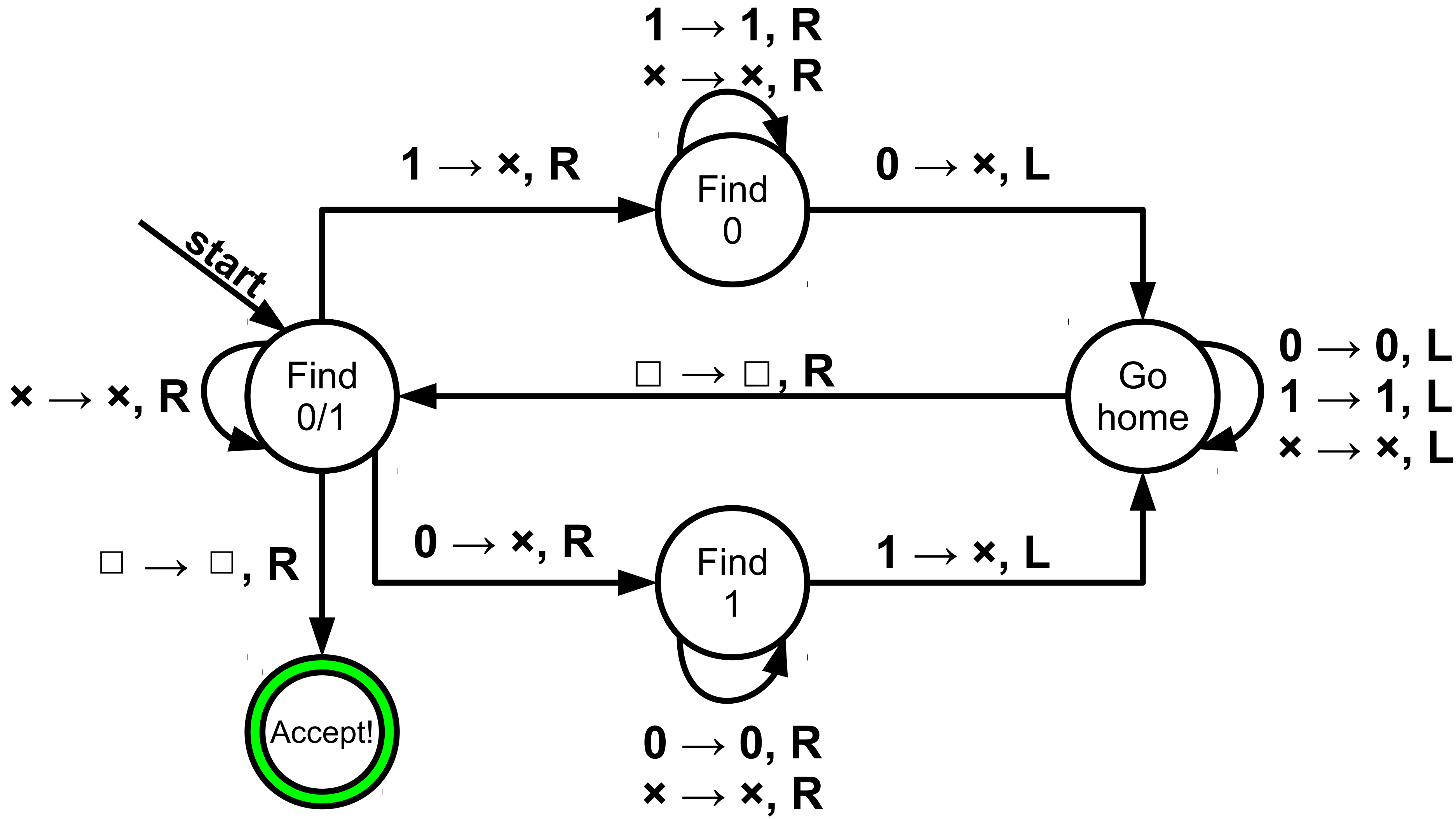
1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
□ → □, R
Accept!
0 → 0, R
× → ×, R
…
×
×
×
×
×
×
×
×
…



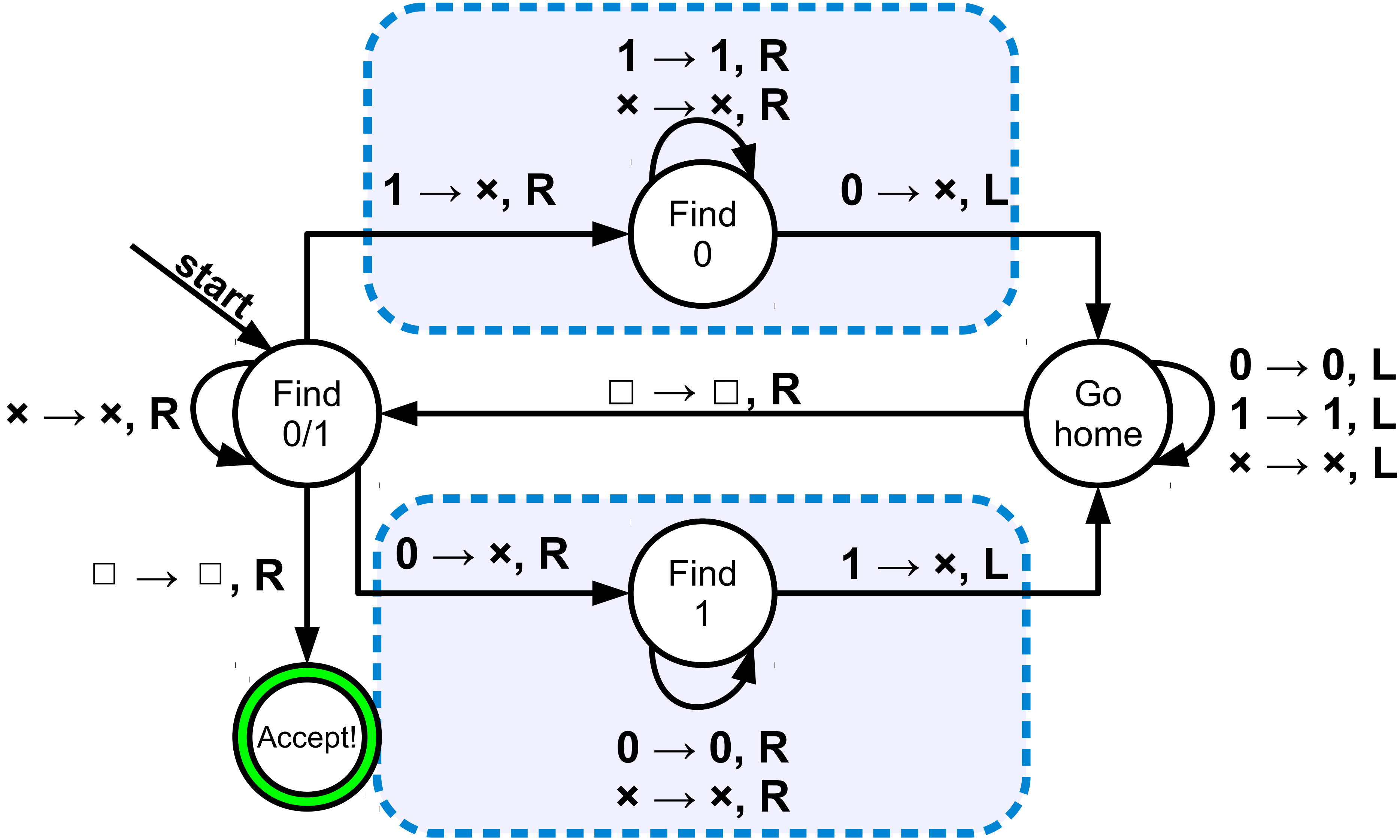
start
Find 0/1
× → ×, R
1 → ×, R
Find 0
1 → 1, R
× → ×, R
0 → ×, L
Go home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find 1
0 → 0, R
× → ×, R
1 → ×, L
□ → □, R
Accept!
…
× × × × × × × ×
…



start
Find 0/1
Find 0
Find 1
Go home
Accept!
1 → ×, R
0 → ×, R
1 → 1, R
× → ×, R
0 → ×, L
0 → 0, R
× → ×, R
1 → ×, L
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
× → ×, R
□ → □, R



Remember that all missing transitions implicitly reject.



1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
× → ×, R
Find 0/1
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
□ → □, R
Accept!
0 → 0, R
× → ×, R

Constant Storage

- Sometimes, a TM needs to remember some additional information that can't be put on the tape.
- In this case, you can use similar techniques from DFAs and introduce extra states into the TM's finite-state control.
- The finite-state control can only remember one of finitely many things, but that might be all that you need!

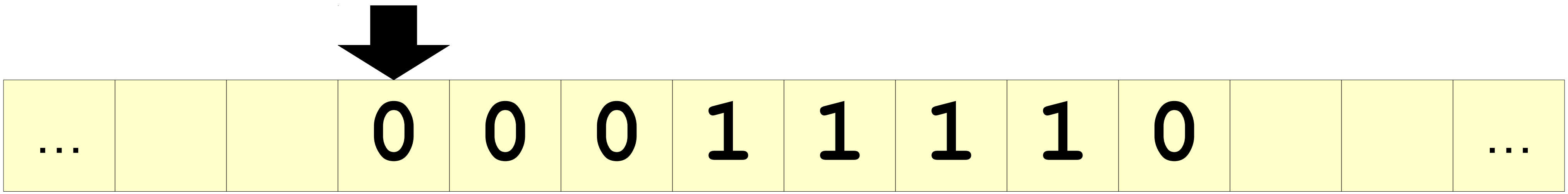
Another TM Design

- We just designed a TM for this language over $\Sigma = \{\texttt{0}, \texttt{1}\}$:

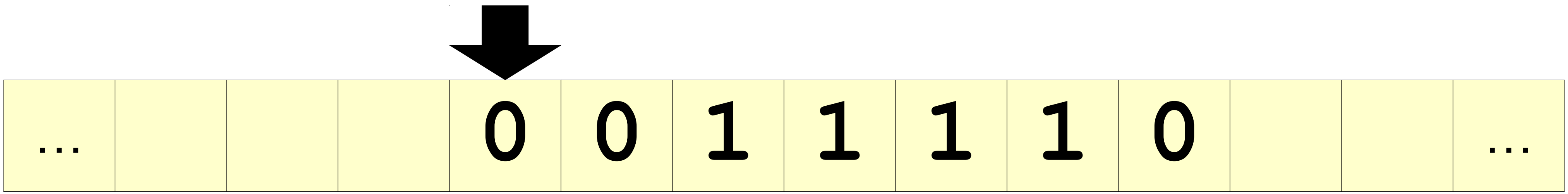
$$L = \{\ w \in \Sigma^* \mid w \text{ has the same number of } \texttt{0}\text{s and } \texttt{1}\text{s}\ \}$$

- Let's do a quick review of how it worked.

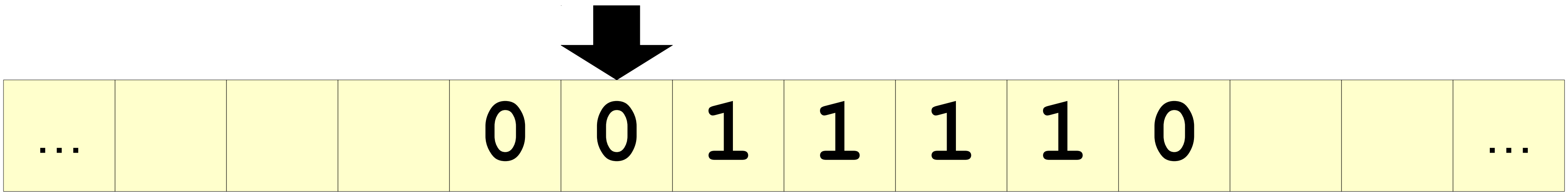
A Leap of Faith



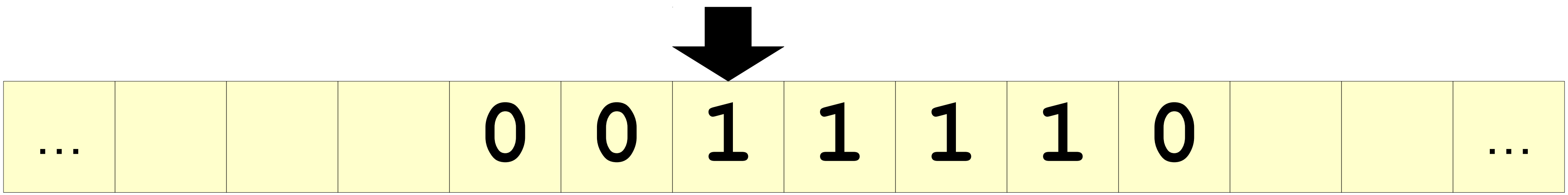
A Leap of Faith



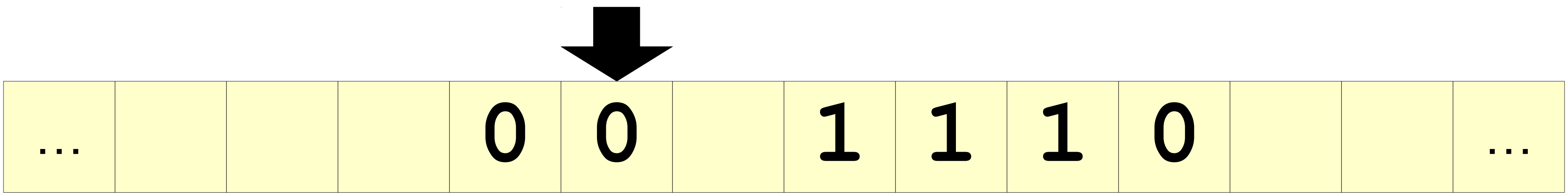
A Leap of Faith



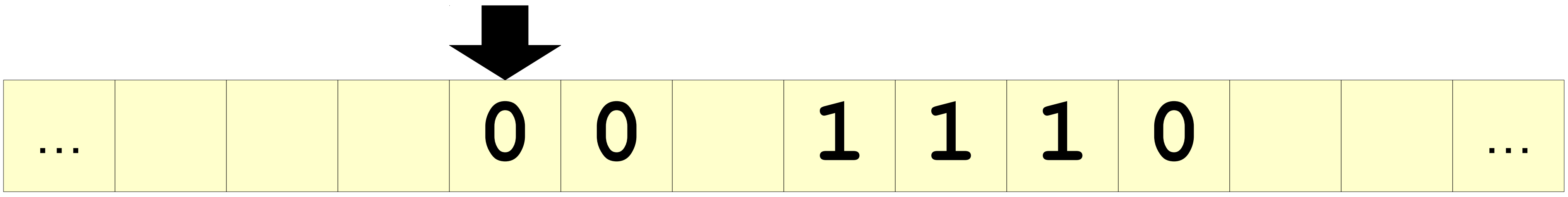
A Leap of Faith



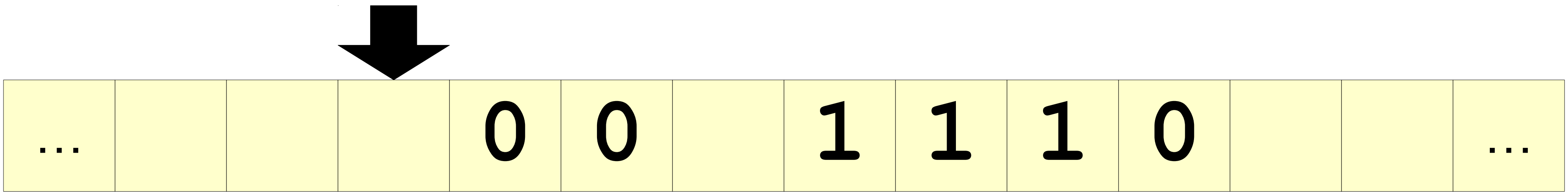
A Leap of Faith



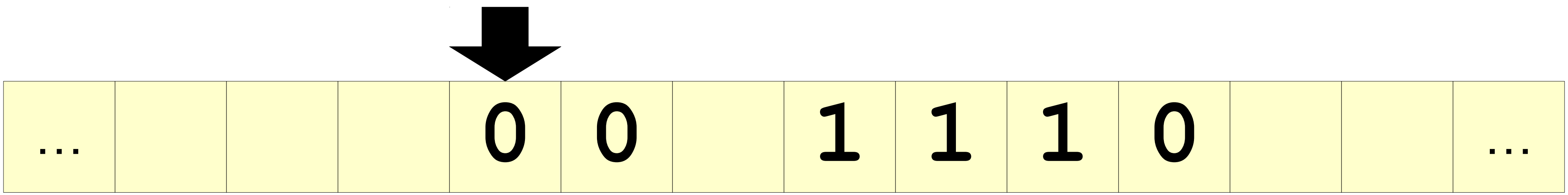
A Leap of Faith



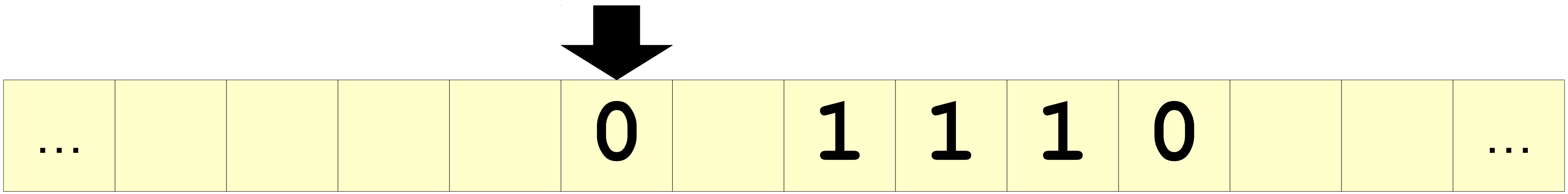
A Leap of Faith



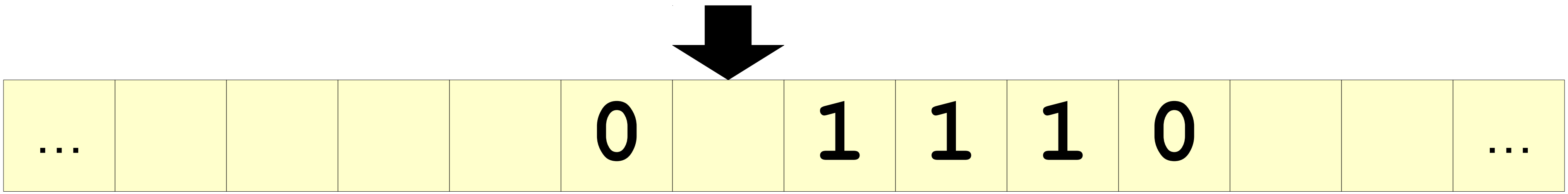
A Leap of Faith



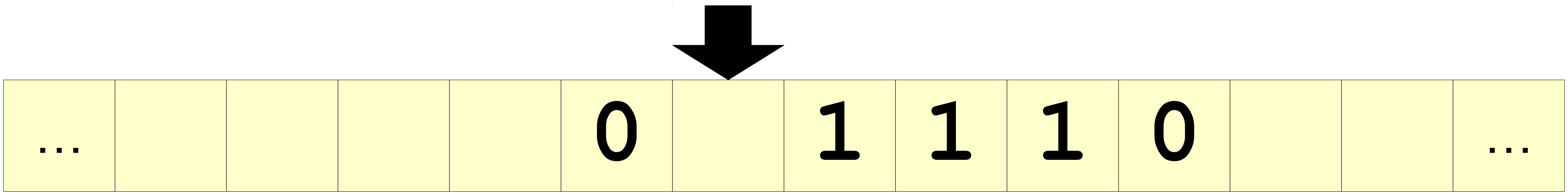
A Leap of Faith



A Leap of Faith

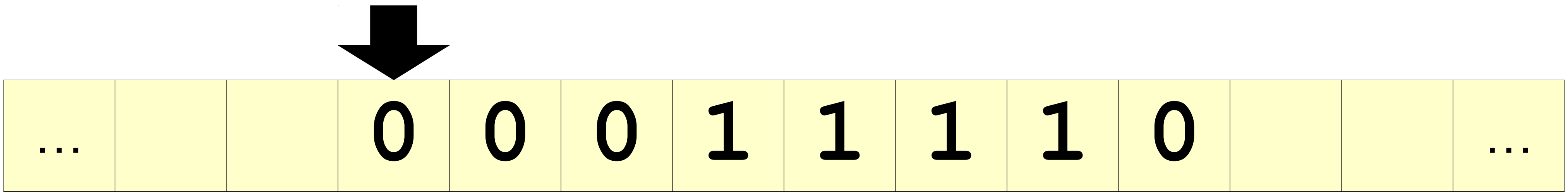


A Leap of Faith

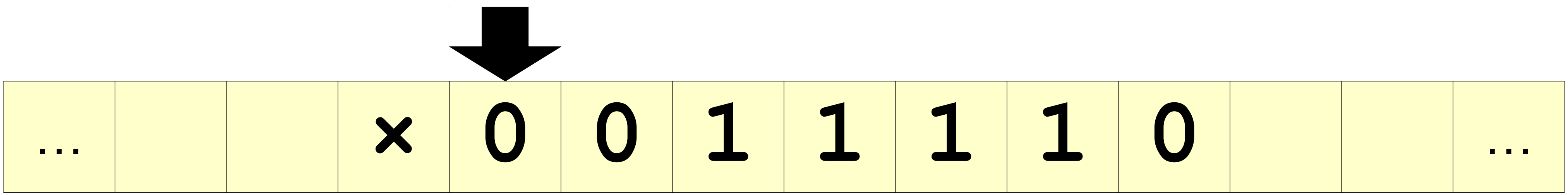


How do we know that this blank isn't one of the infinitely many blanks after our input string?

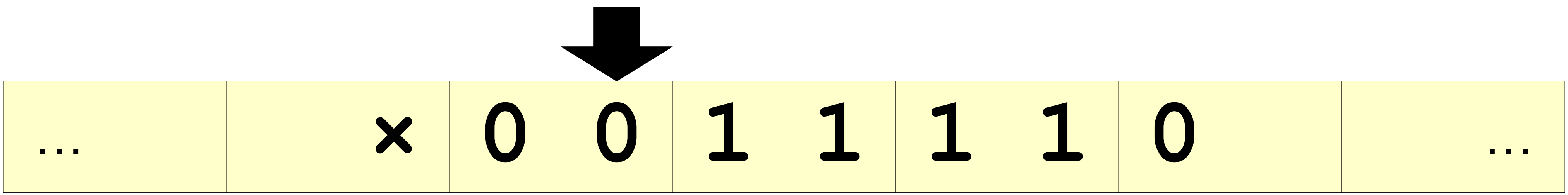
The Solution



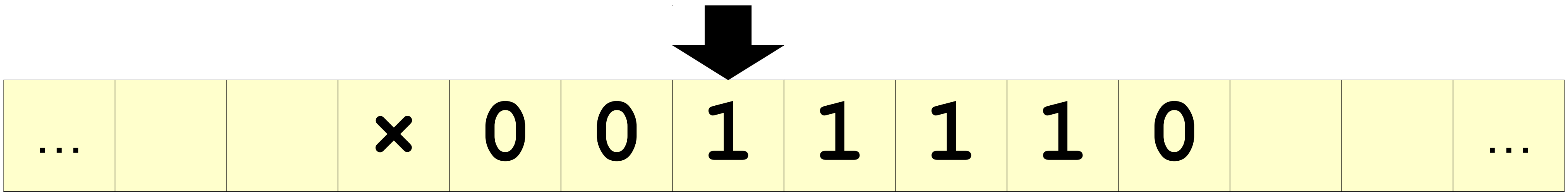
The Solution



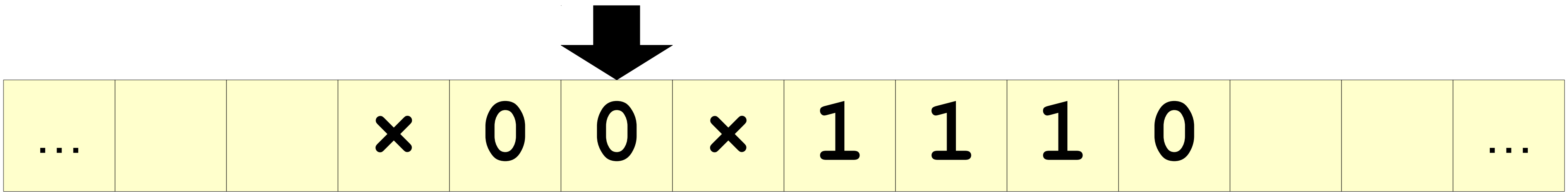
The Solution



The Solution



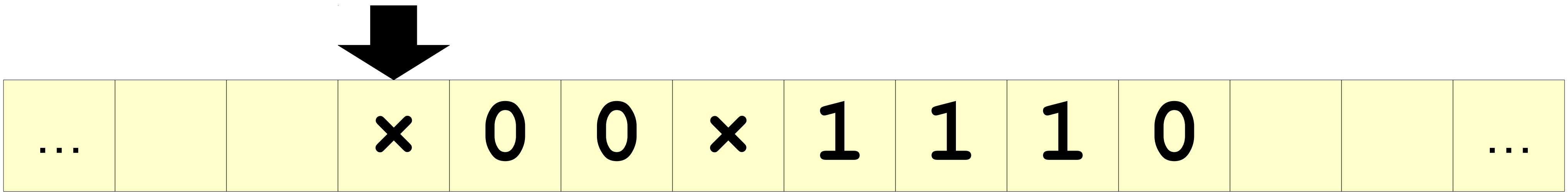
The Solution



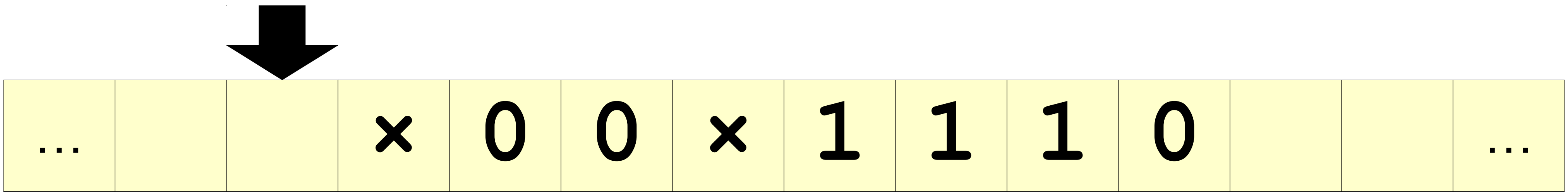
The Solution

…			×	0	0	×	1	1	1	0			…

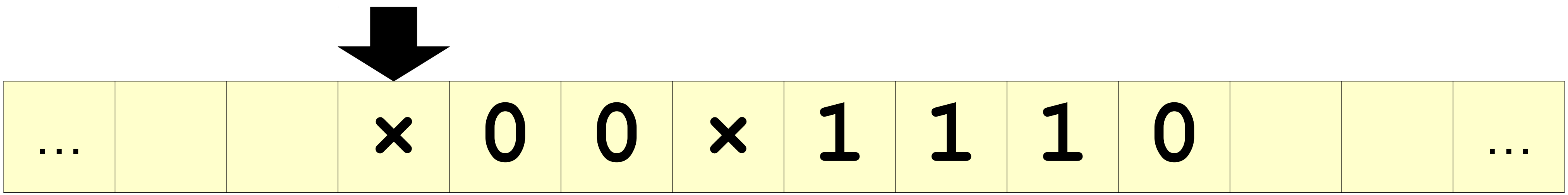
The Solution



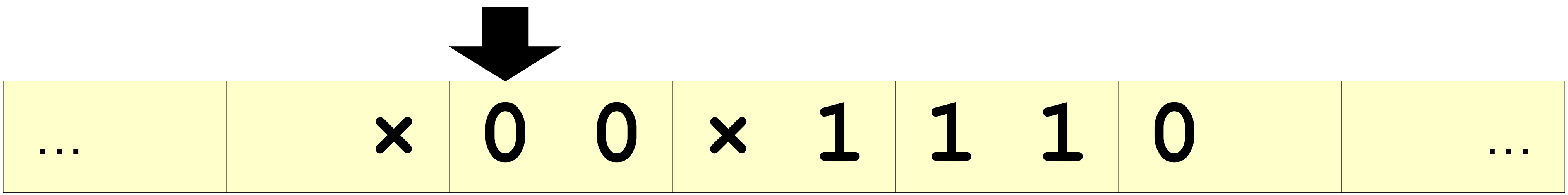
The Solution



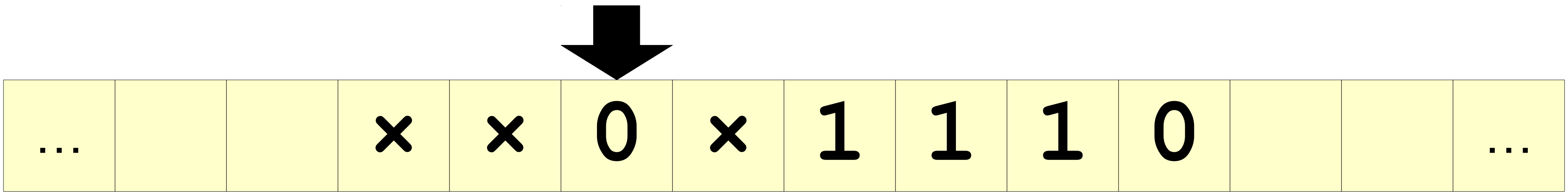
The Solution



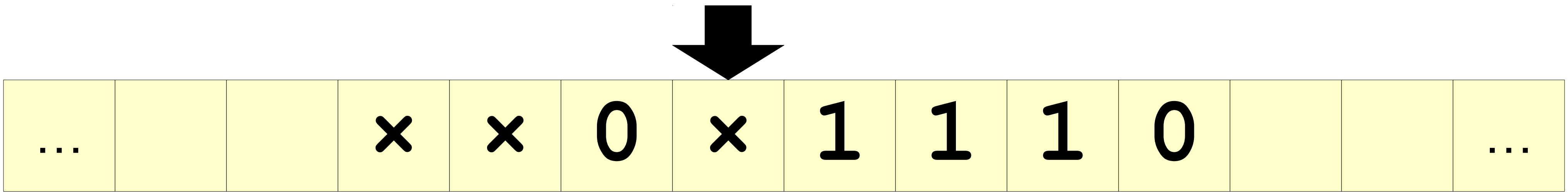
The Solution



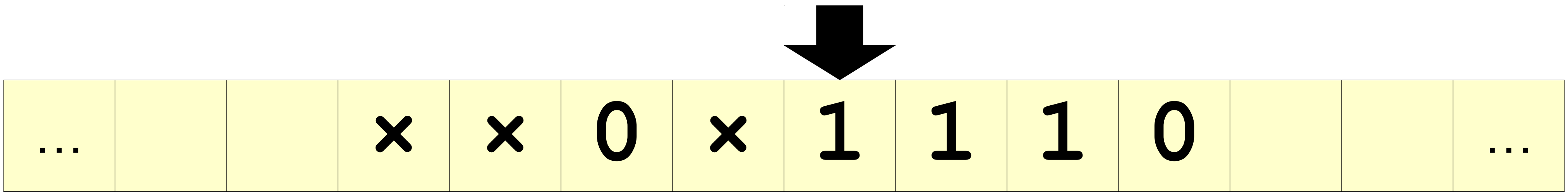
The Solution



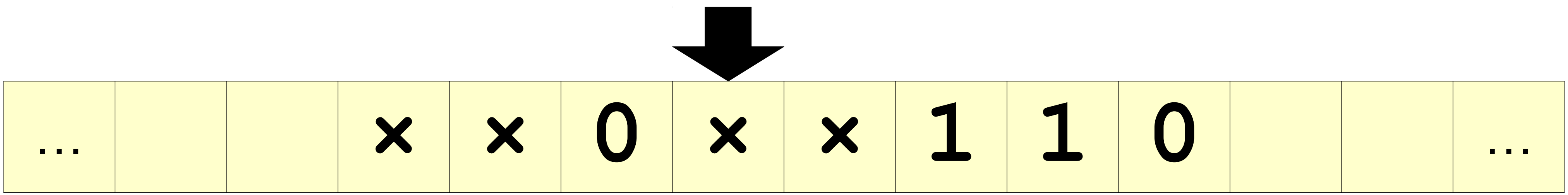
The Solution



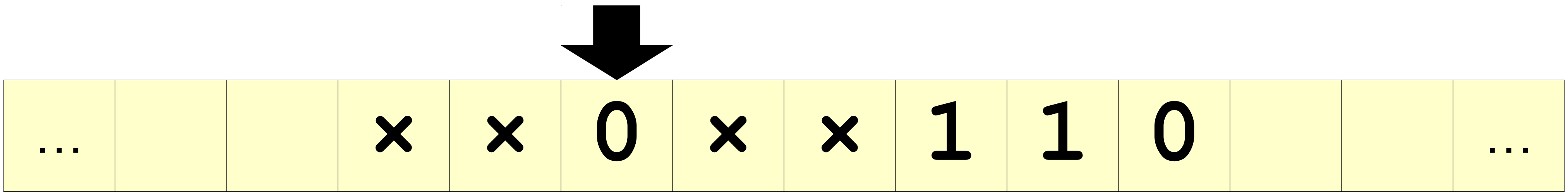
The Solution



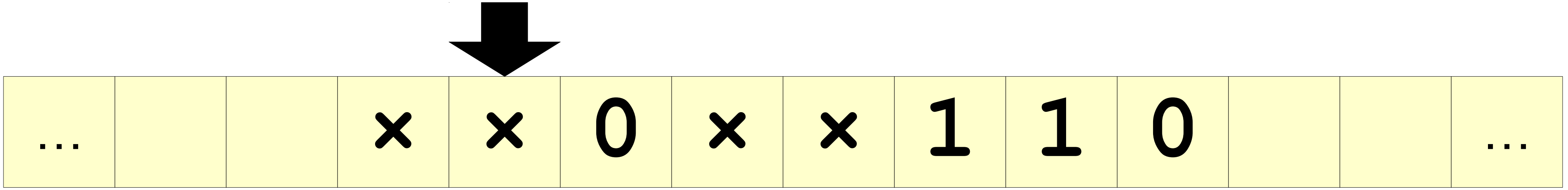
The Solution



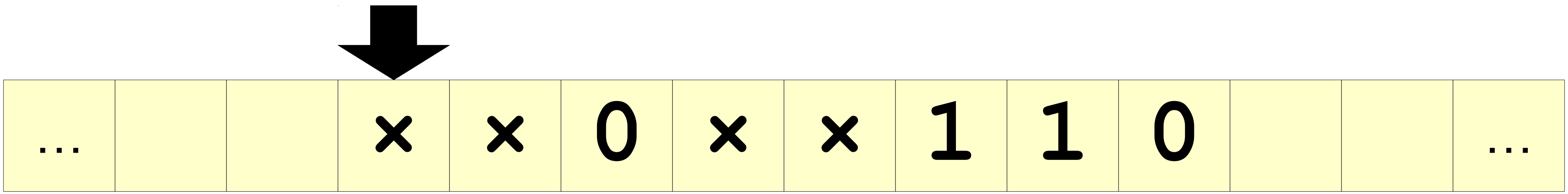
The Solution



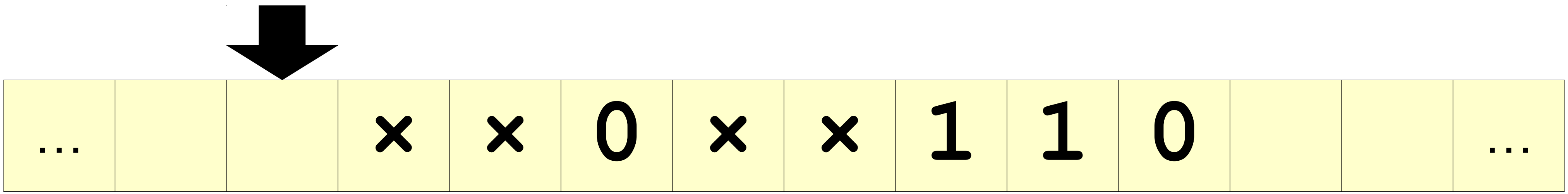
The Solution



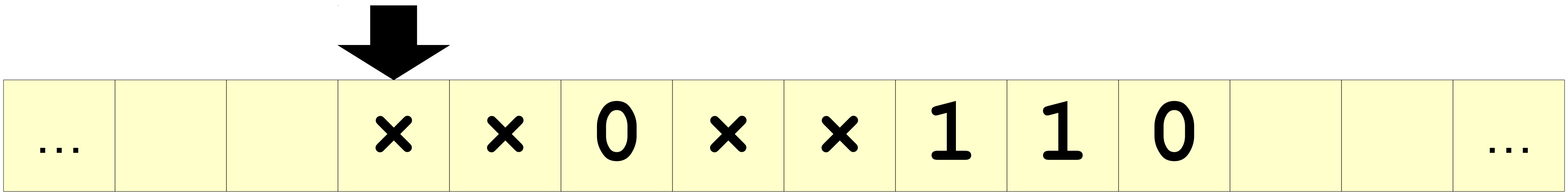
The Solution



The Solution

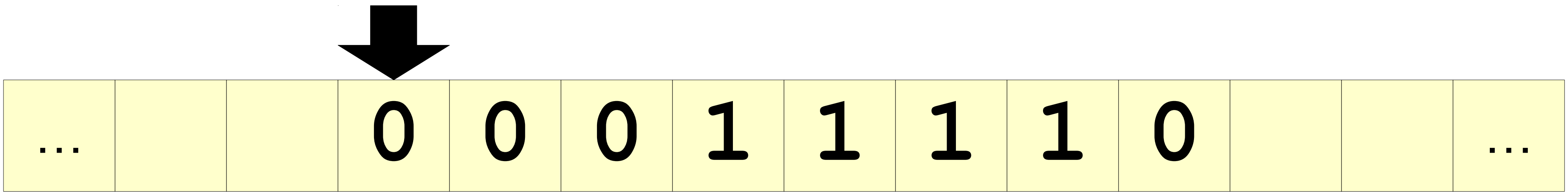


The Solution

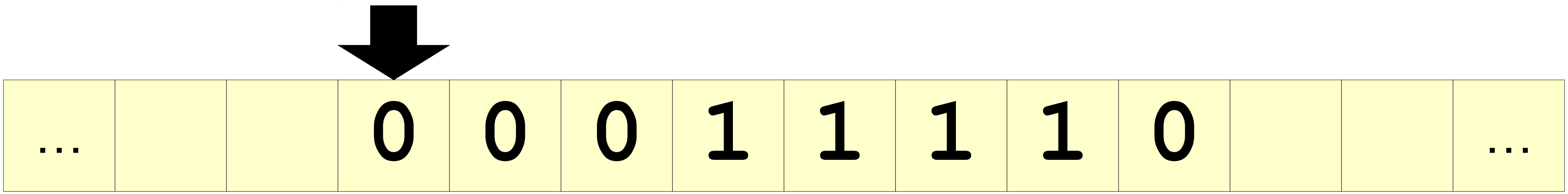


A Different Idea

A Different Strategy

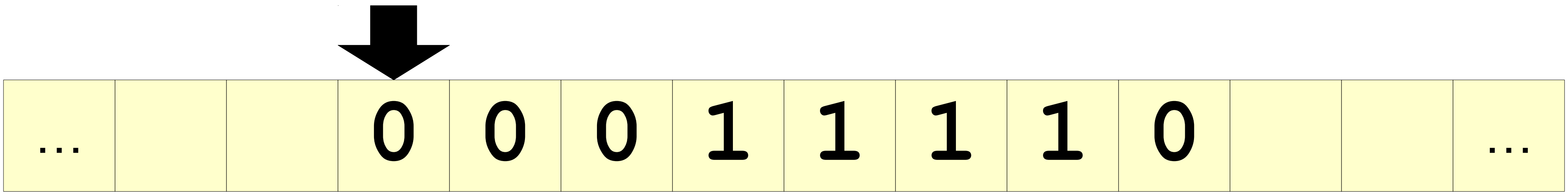


A Different Strategy



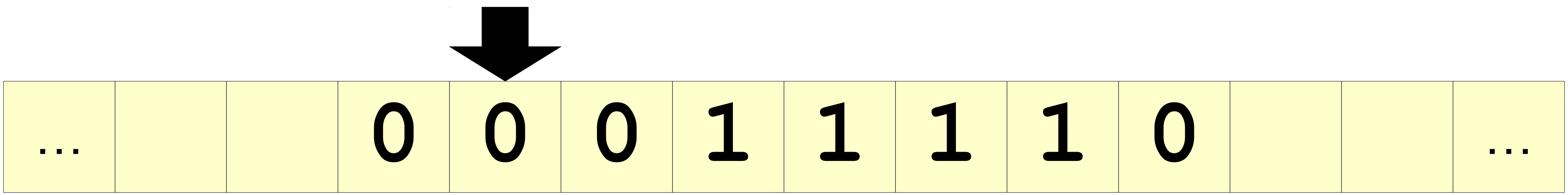
Could we sort the characters of this string?

A Different Strategy



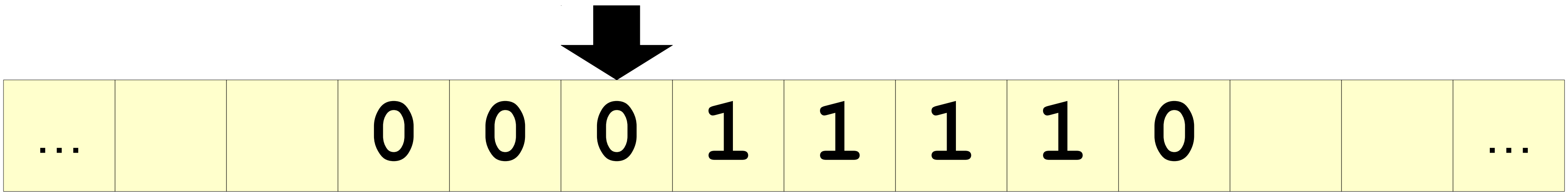
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



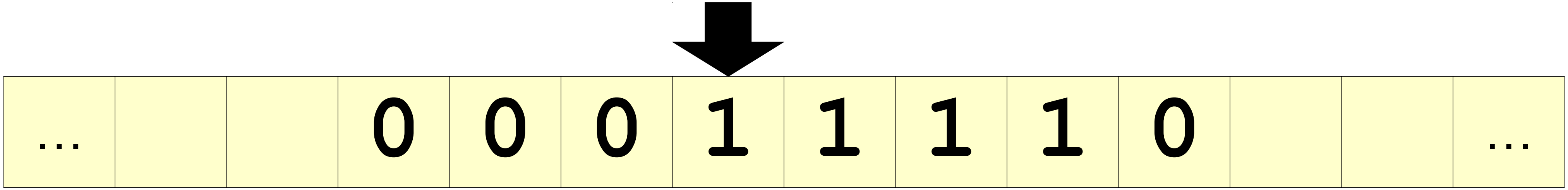
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



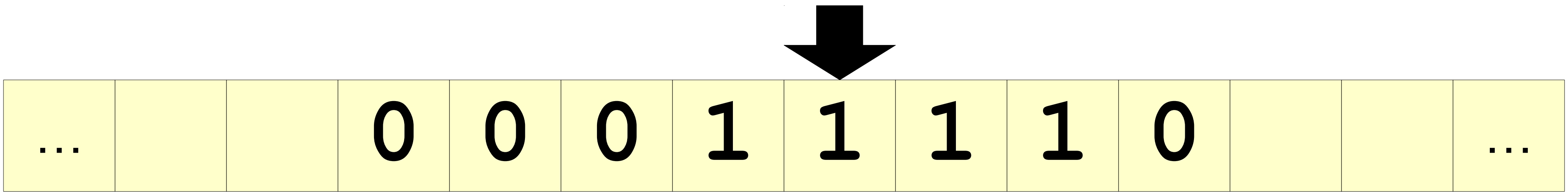
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



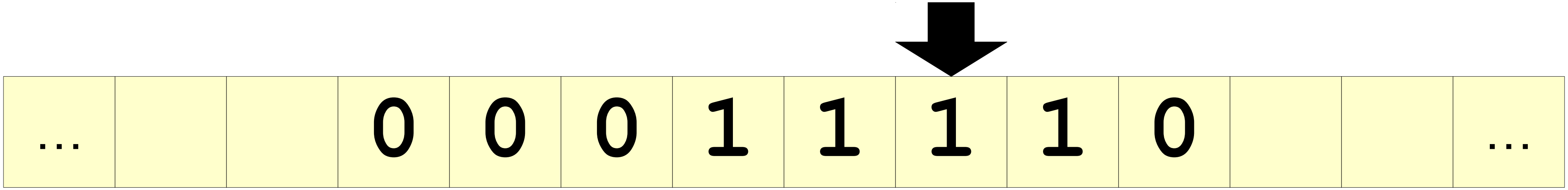
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



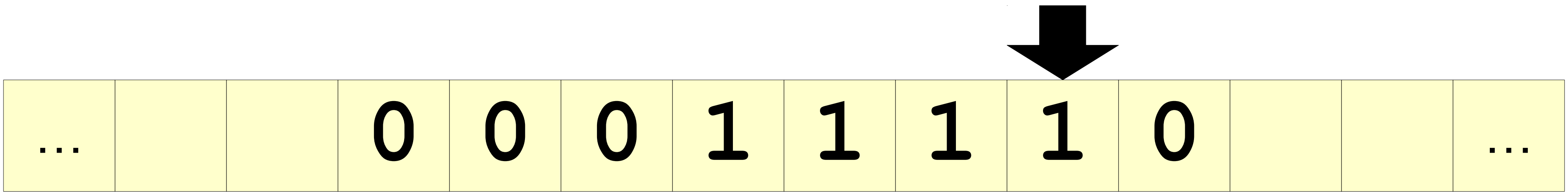
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



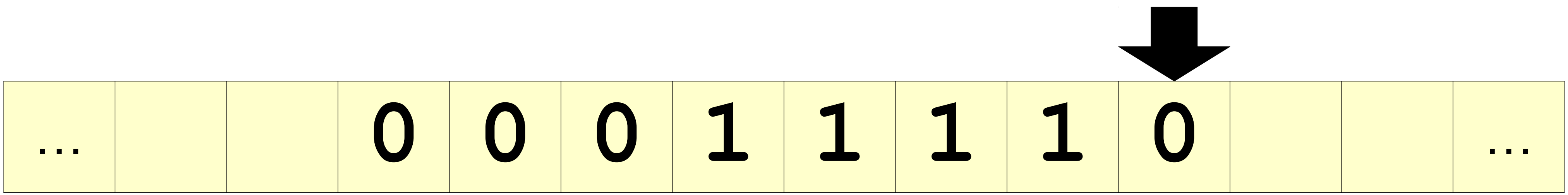
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



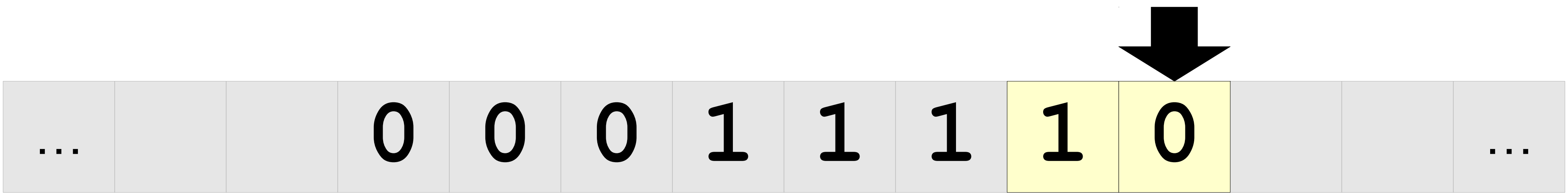
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



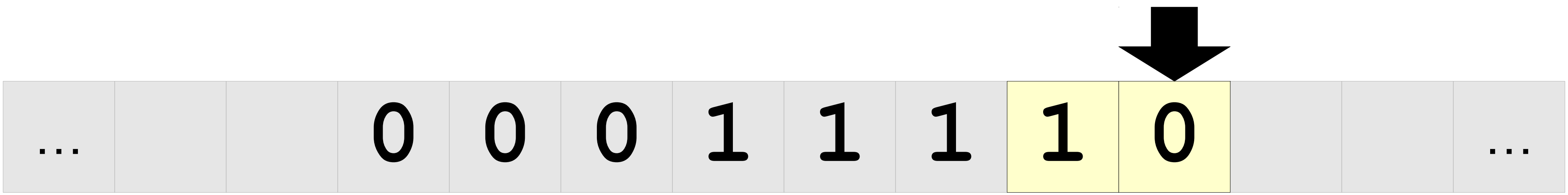
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



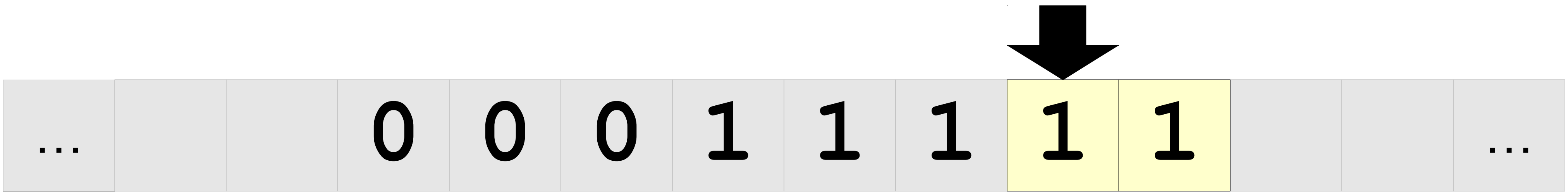
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



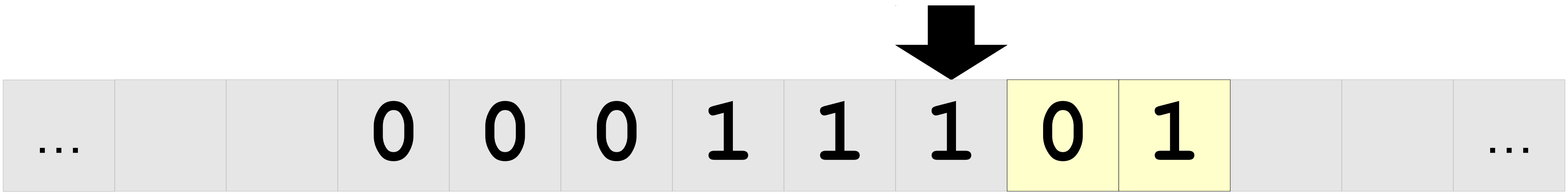
Observation 2: A string of 0s and 1s is <u>not</u> sorted if it contains 10 as a substring.

A Different Strategy



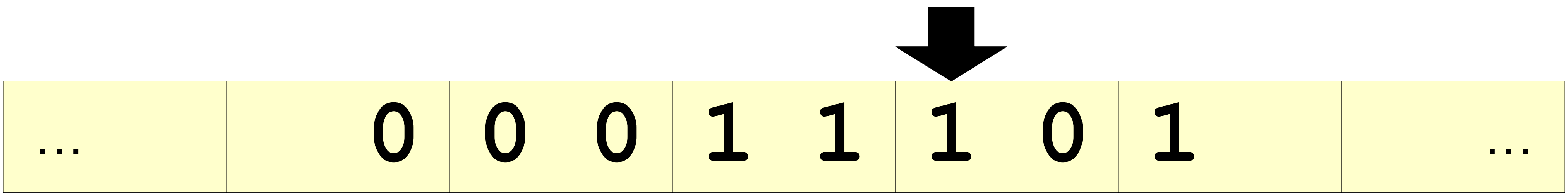
Observation 2: A string of 0s and 1s is <u>not</u> sorted if it contains 10 as a substring.

A Different Strategy



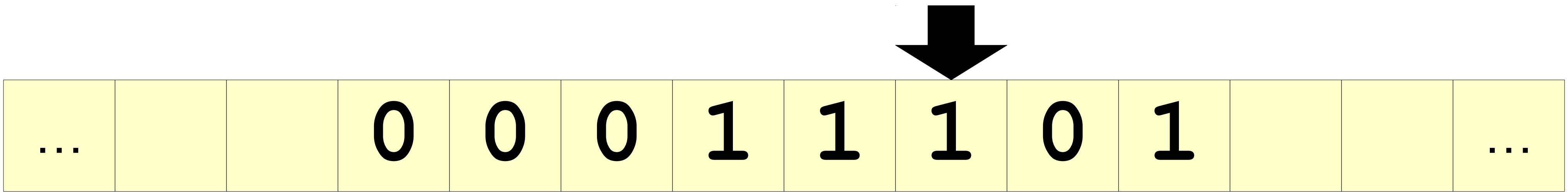
Observation 2: A string of 0s and 1s is <u>not</u> sorted if it contains 10 as a substring.

A Different Strategy



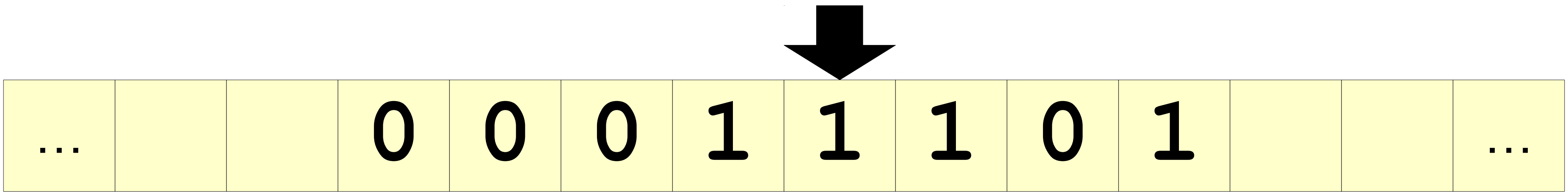
Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.

A Different Strategy



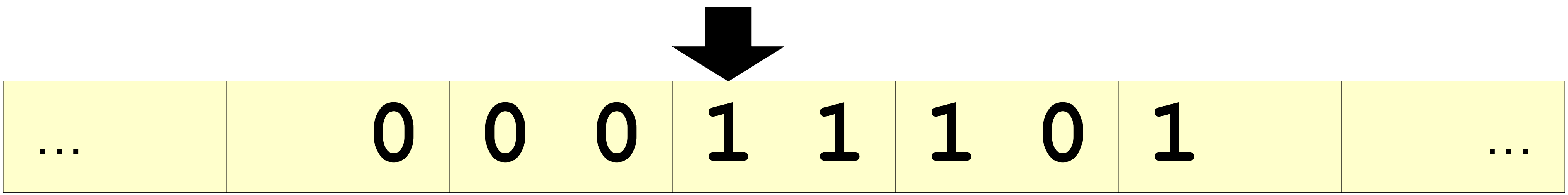
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



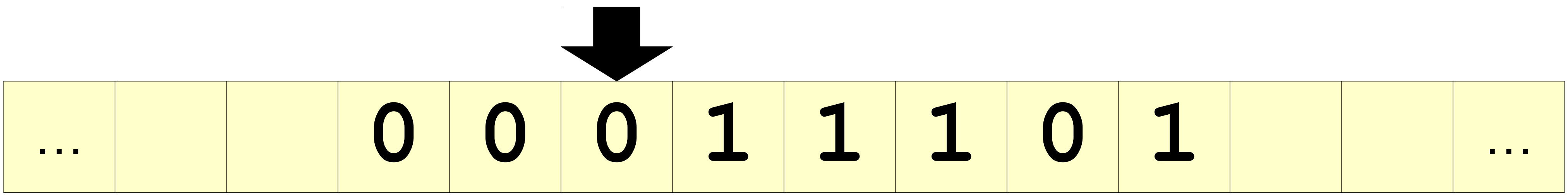
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



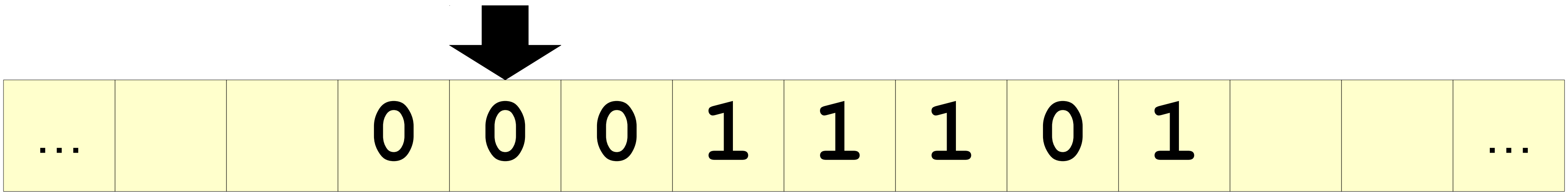
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



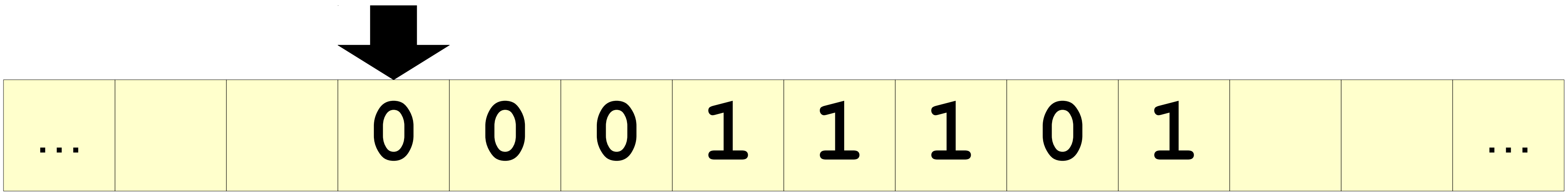
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



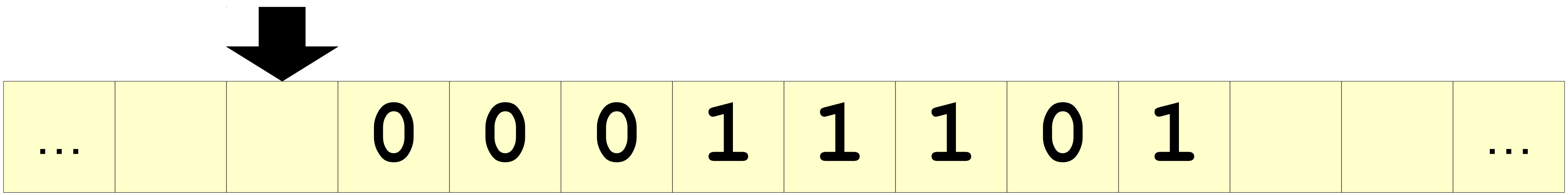
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



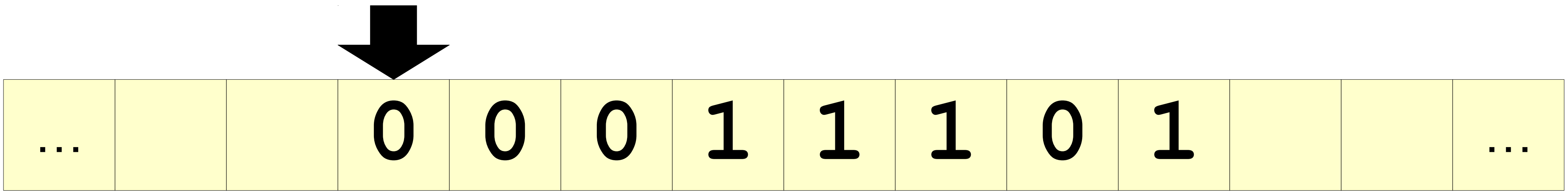
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



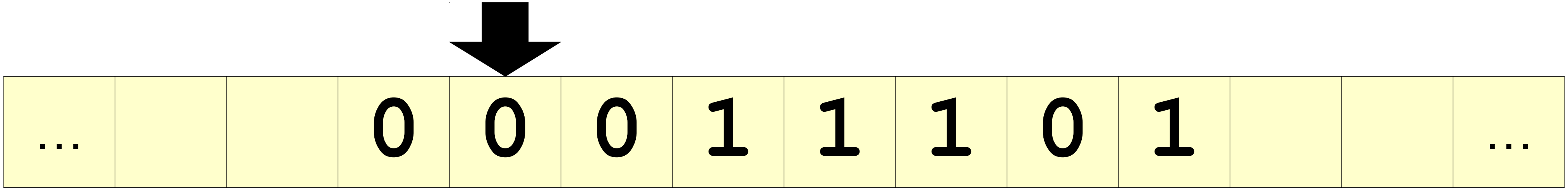
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



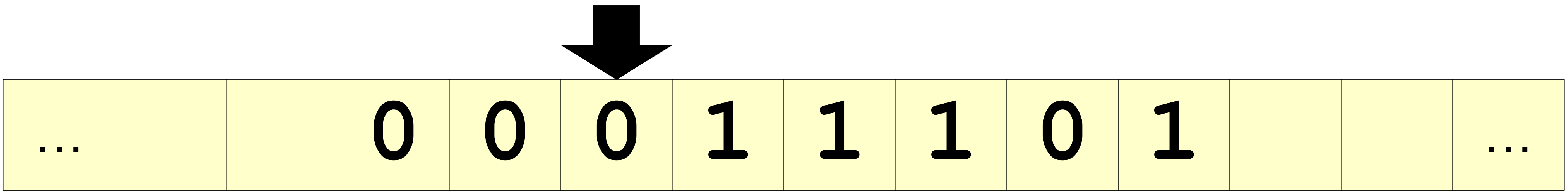
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



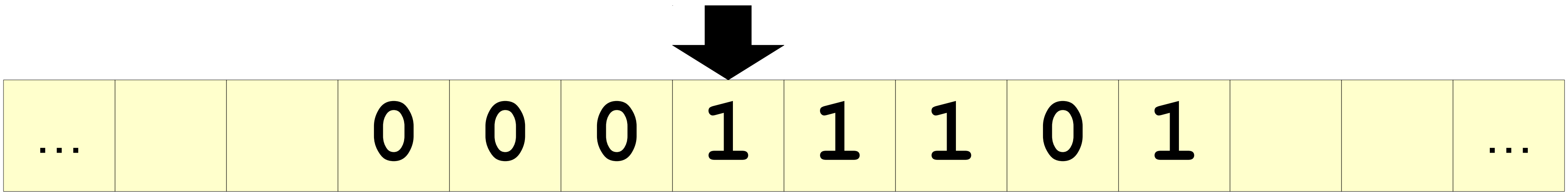
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



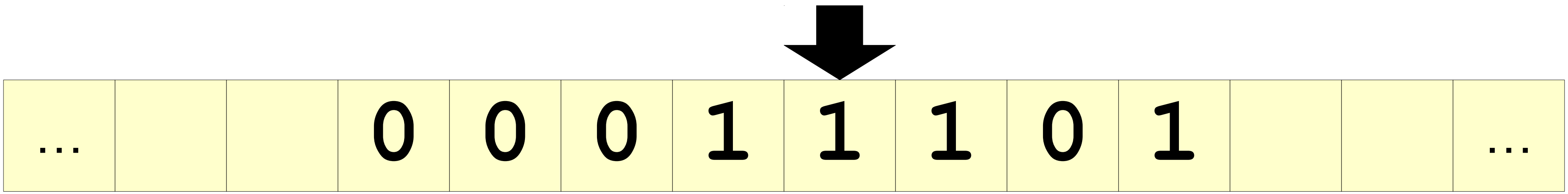
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



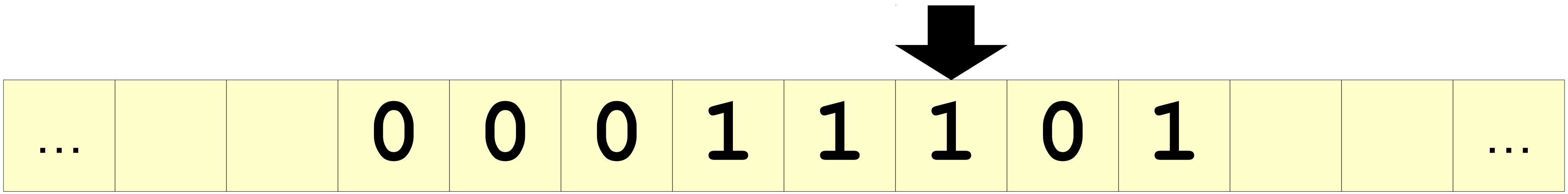
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



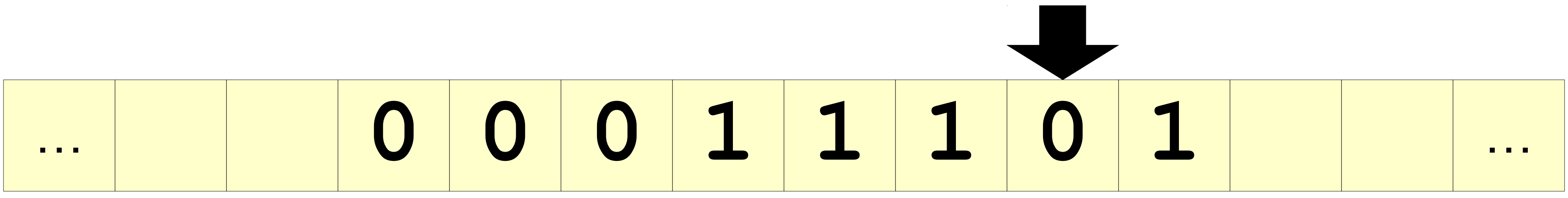
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



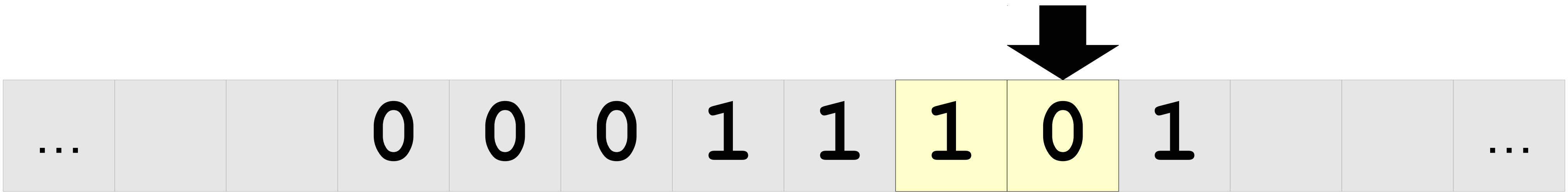
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



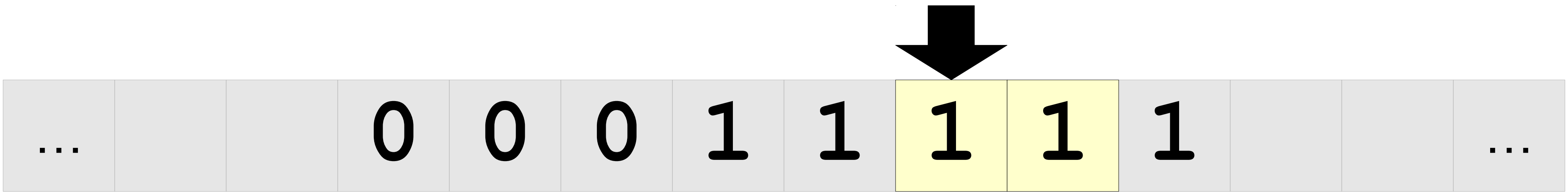
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



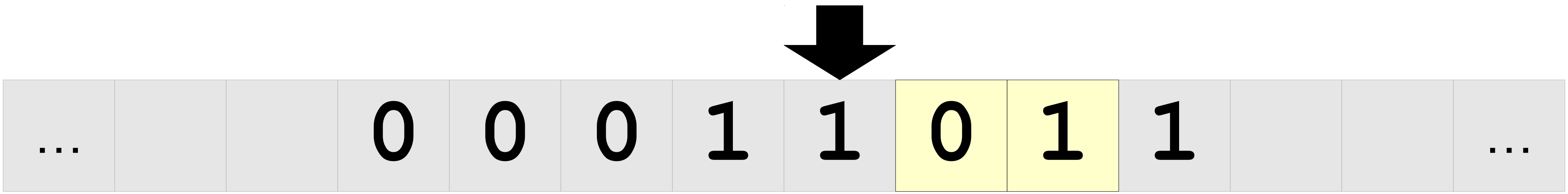
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



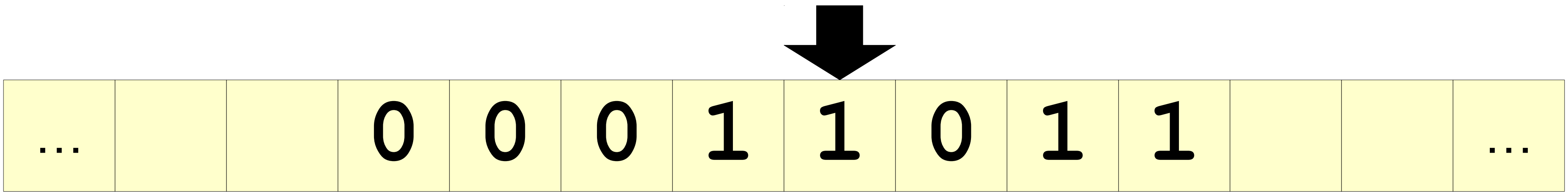
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



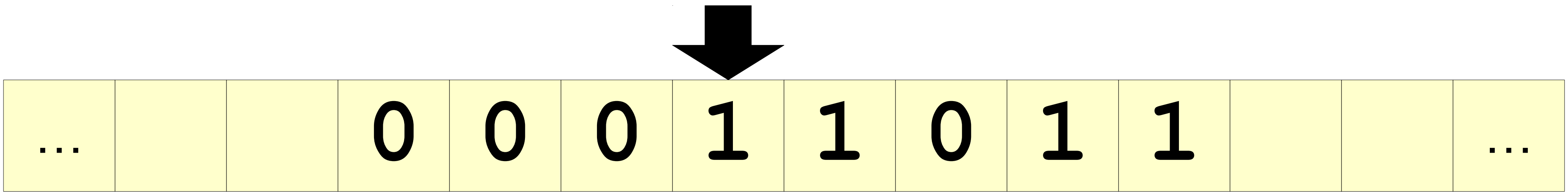
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



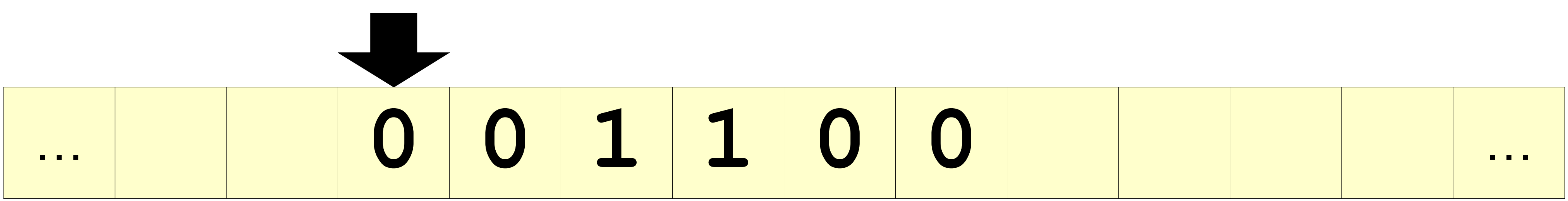
Idea: Repeatedly find a copy of 10 and replace it with 01.

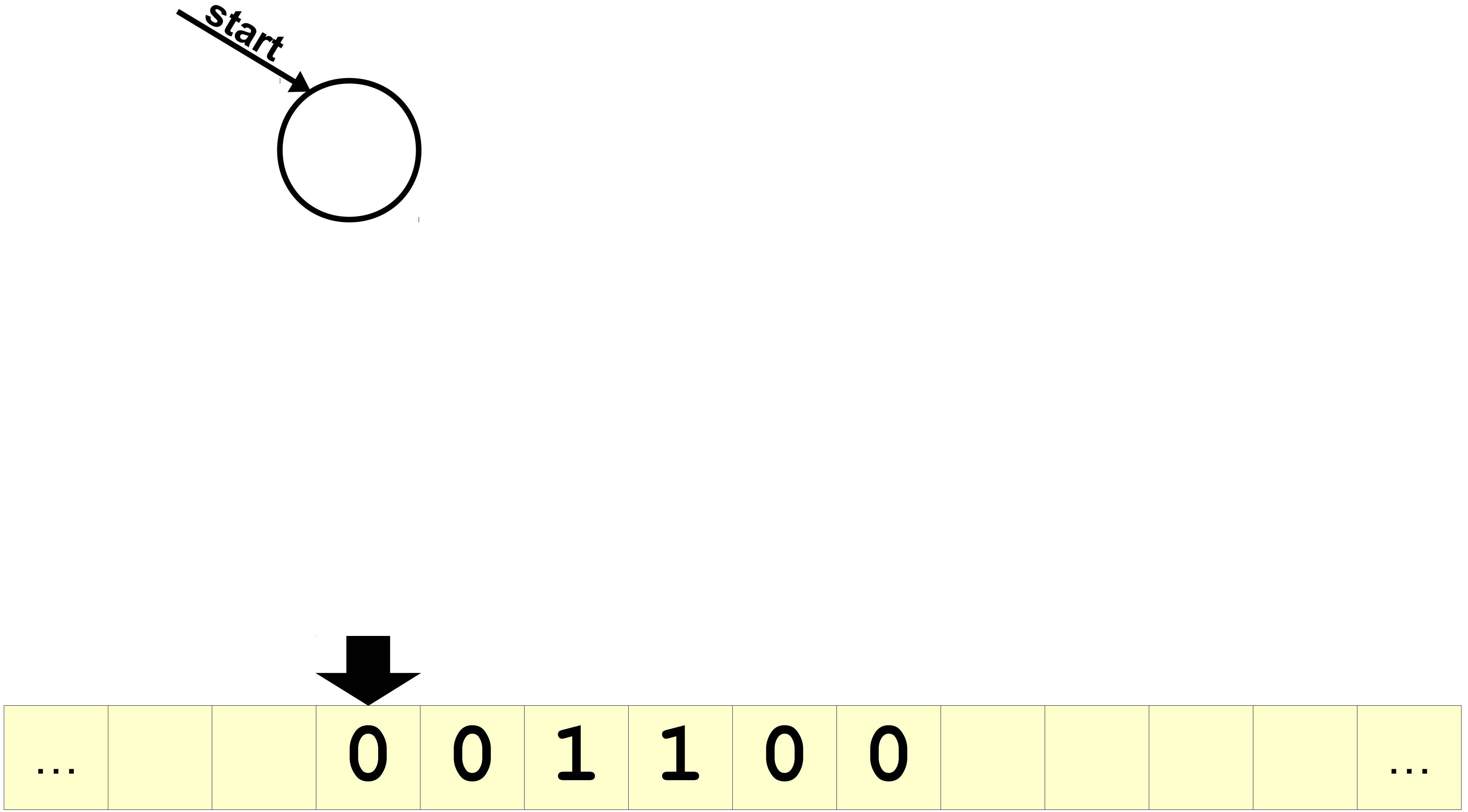
A Different Strategy

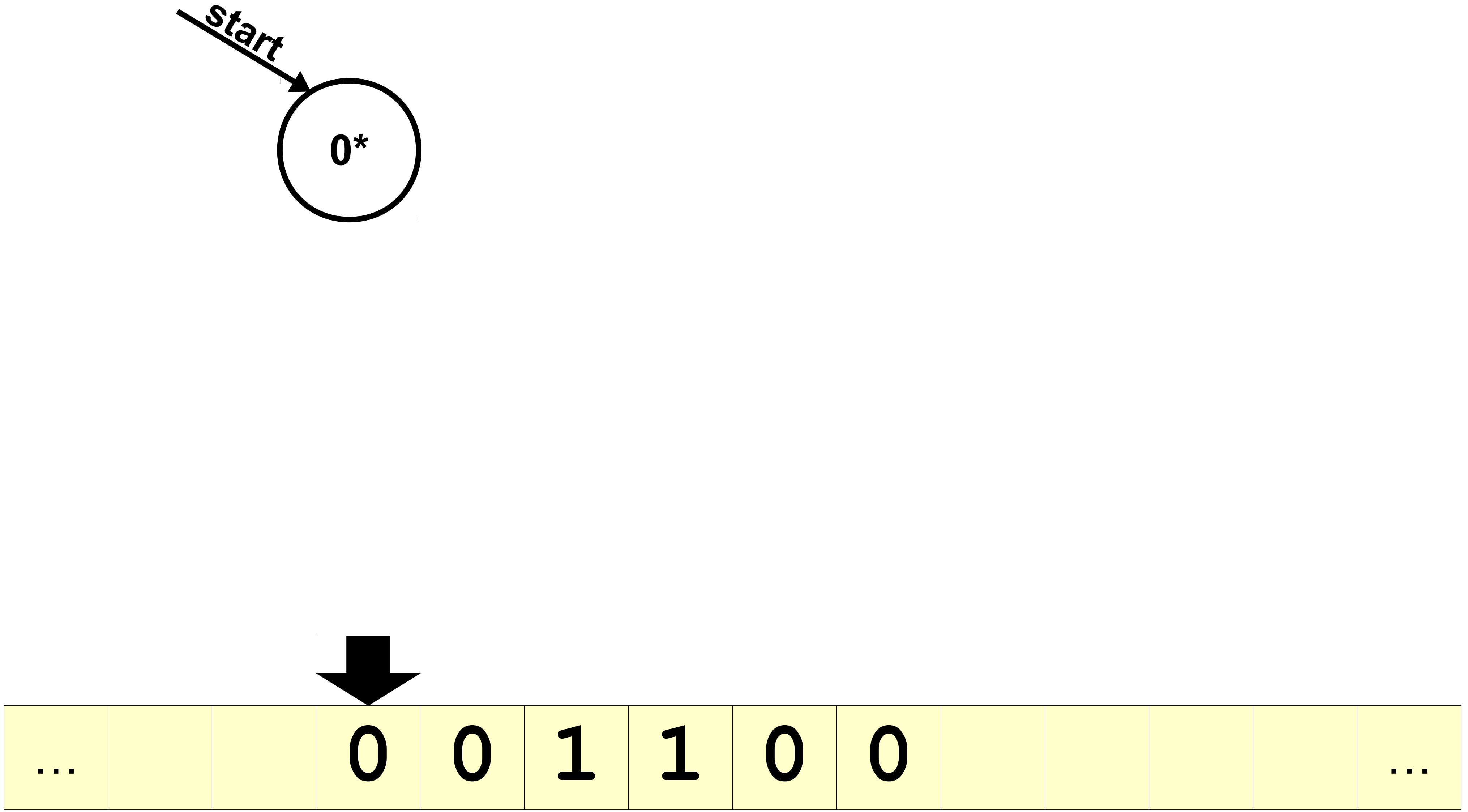


Idea: Repeatedly find a copy of 10 and replace it with 01.

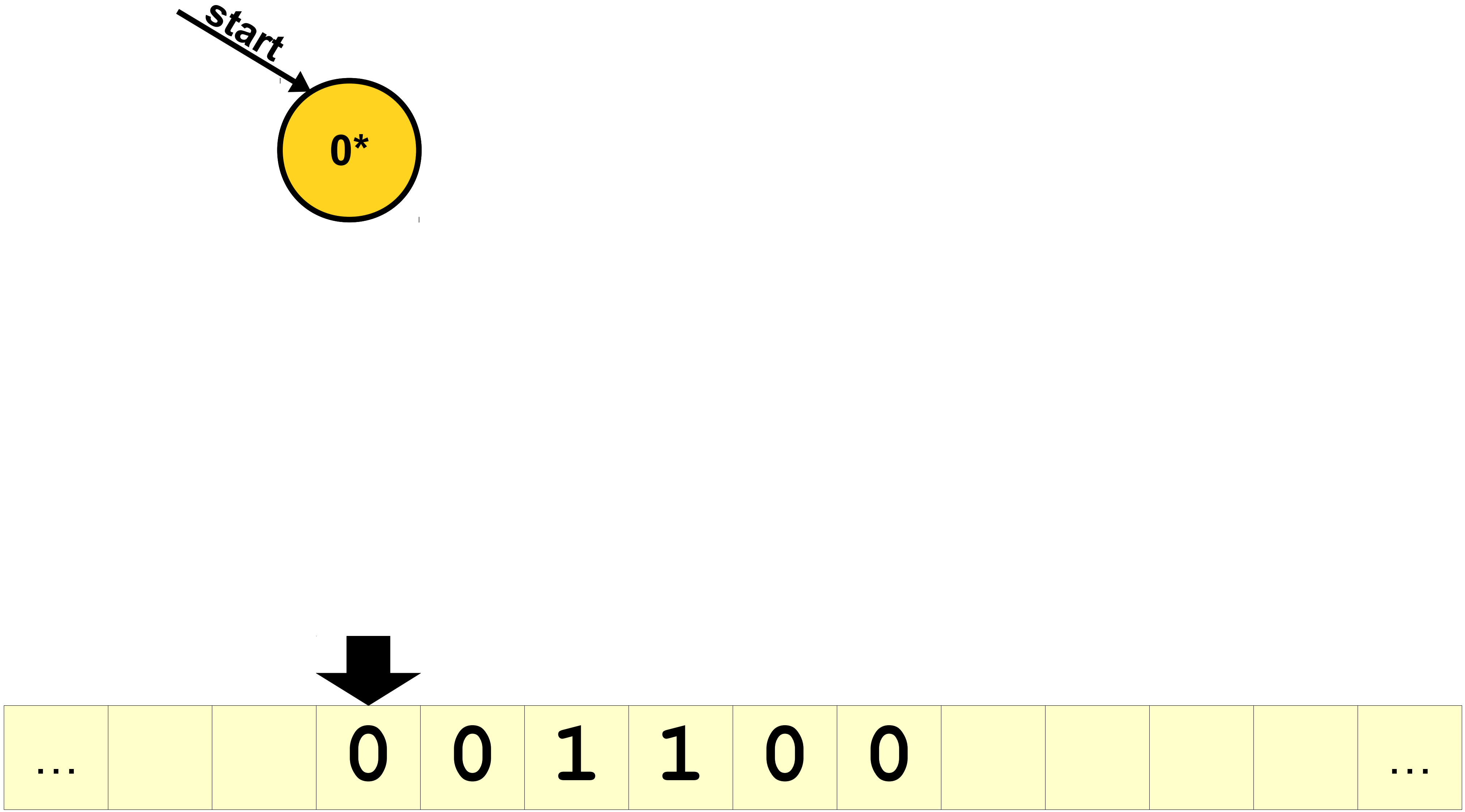
Let's Build It!

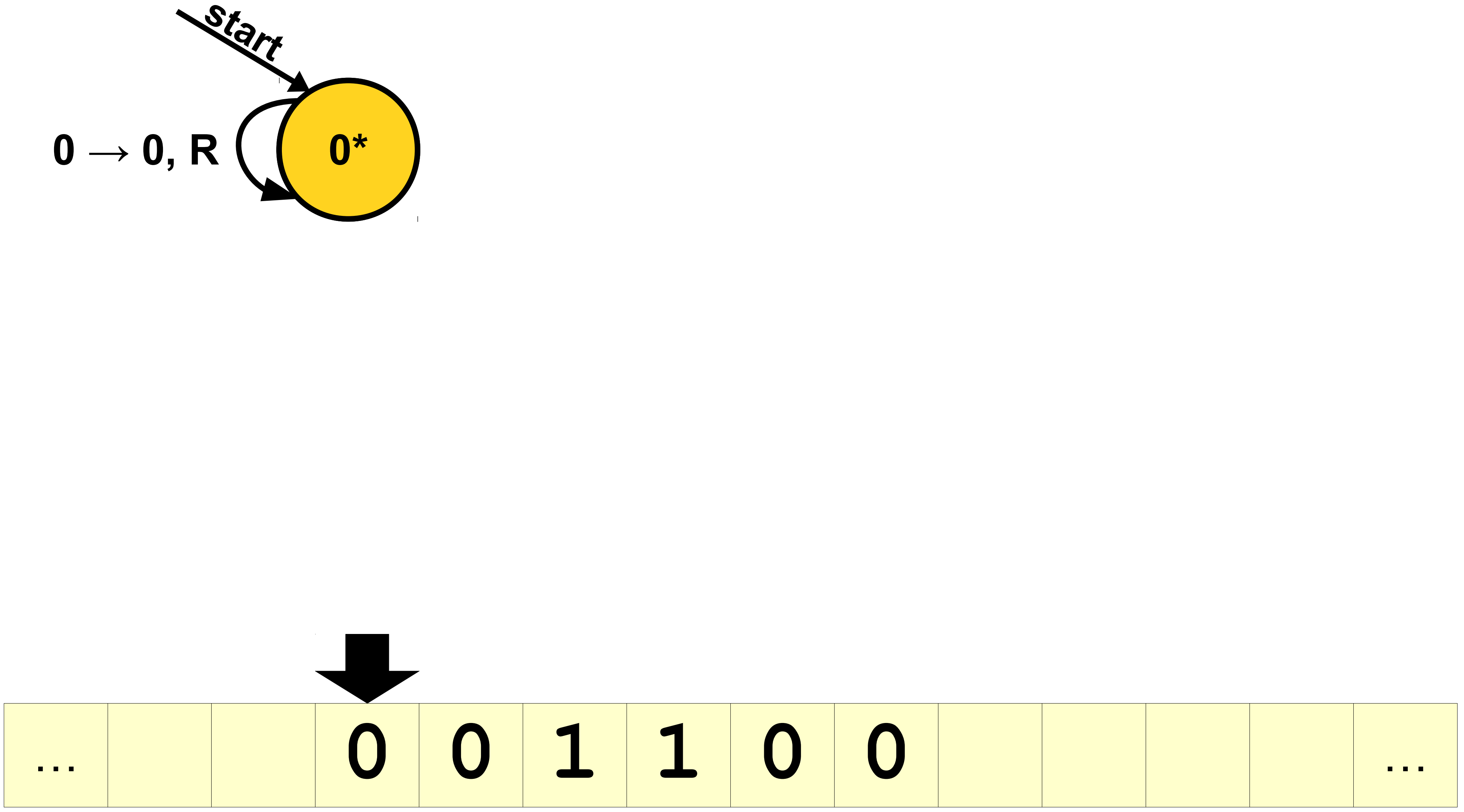

…
0
0
1
1
0
0
…

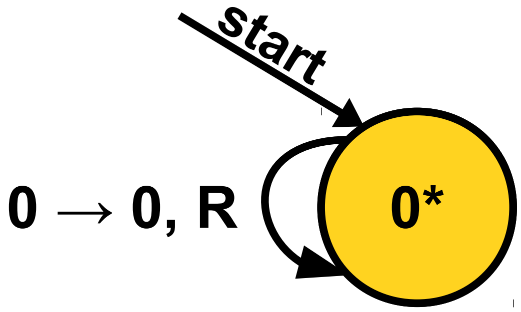

start
…
0
0
1
1
0
0
…

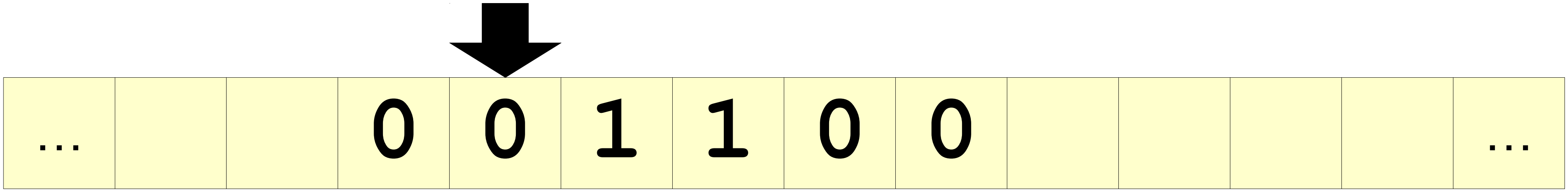


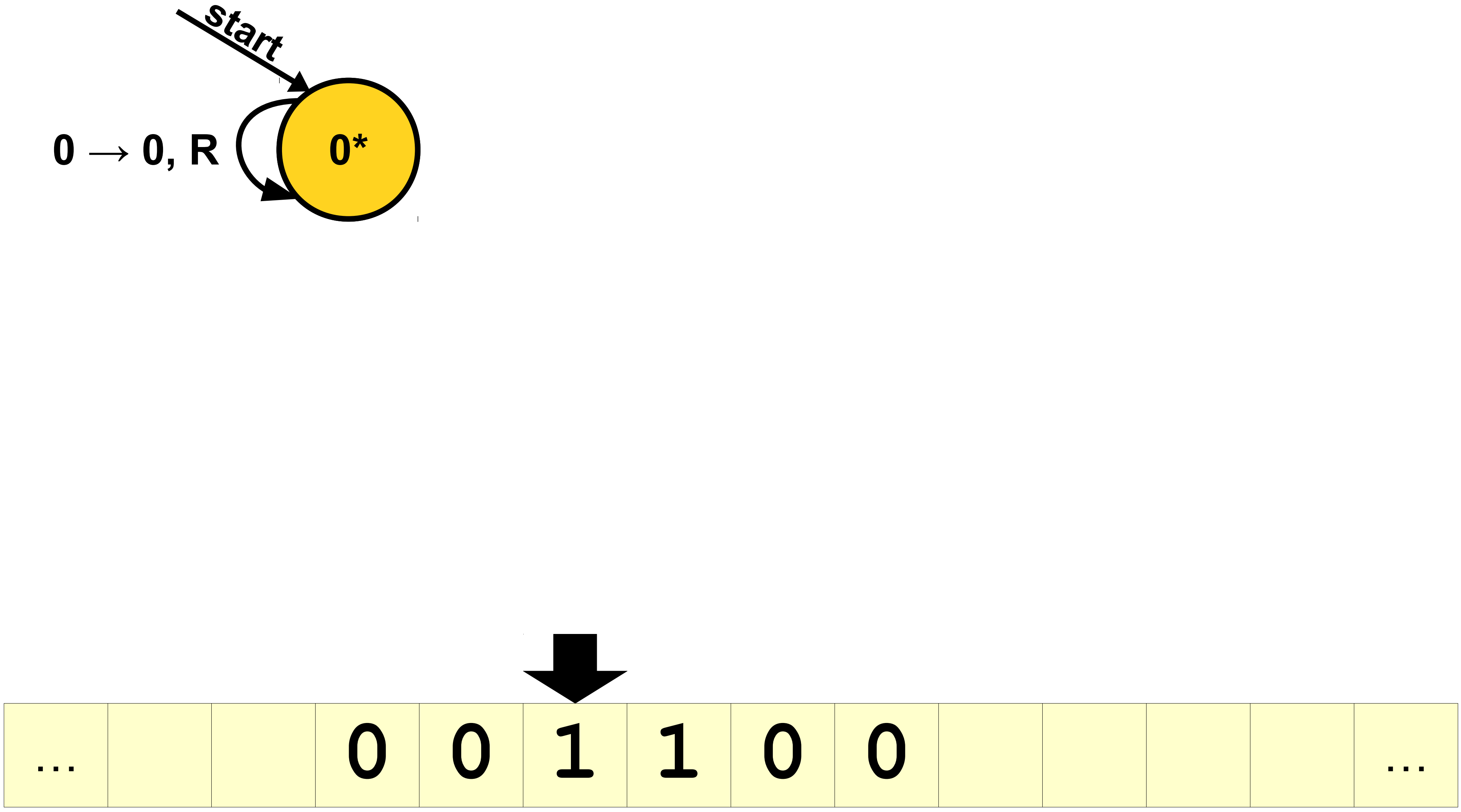
start
0*
...
0
0
1
1
0
0
...

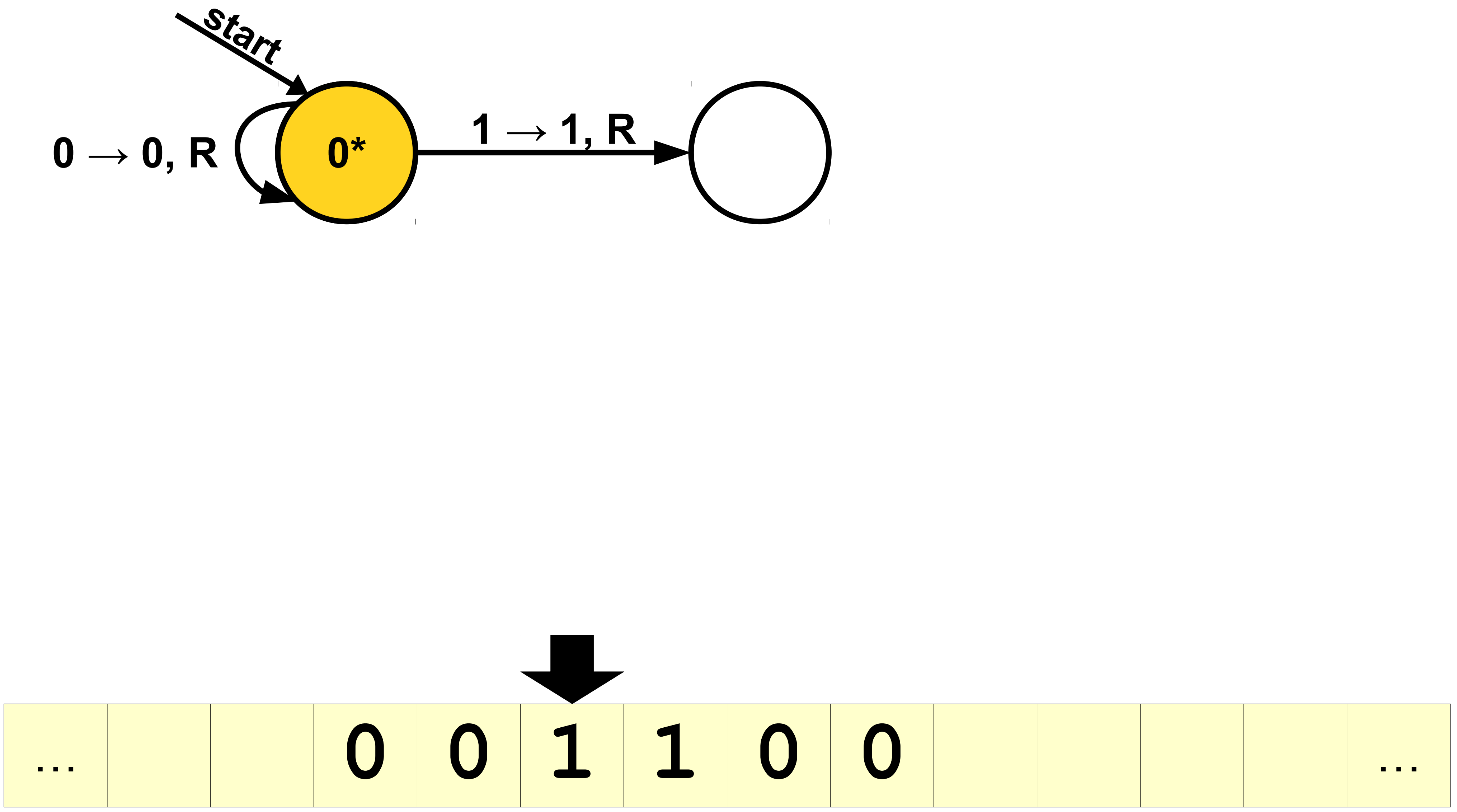

start
0*
...
0
0
1
1
0
0
...

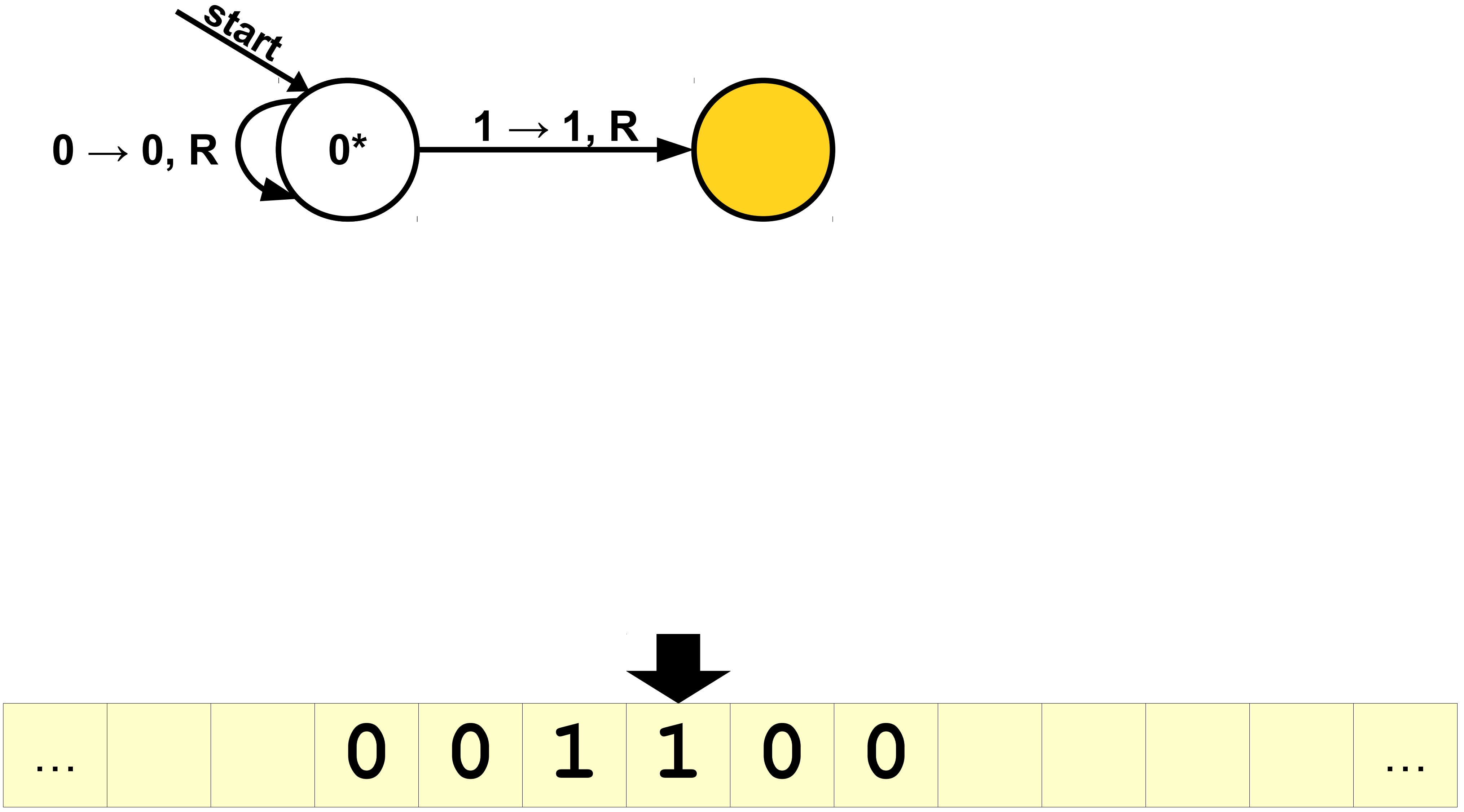

start
0 → 0, R
0*
...
0 0 1 1 0 0
...


start
0*
0 → 0, R

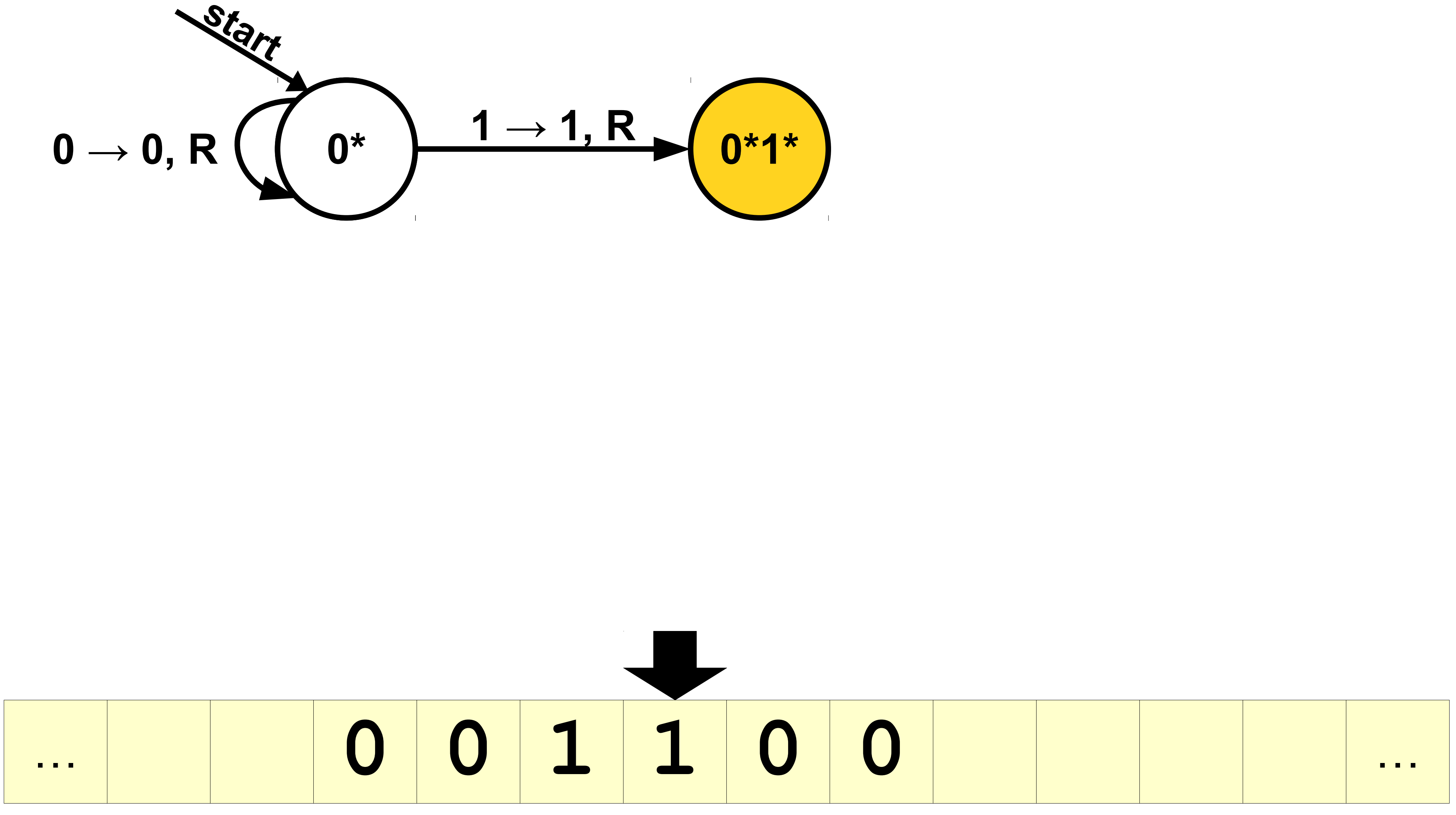

...
0
0
1
1
0
0
...


start
0 → 0, R
0*
...
0 0 1 1 0 0
...

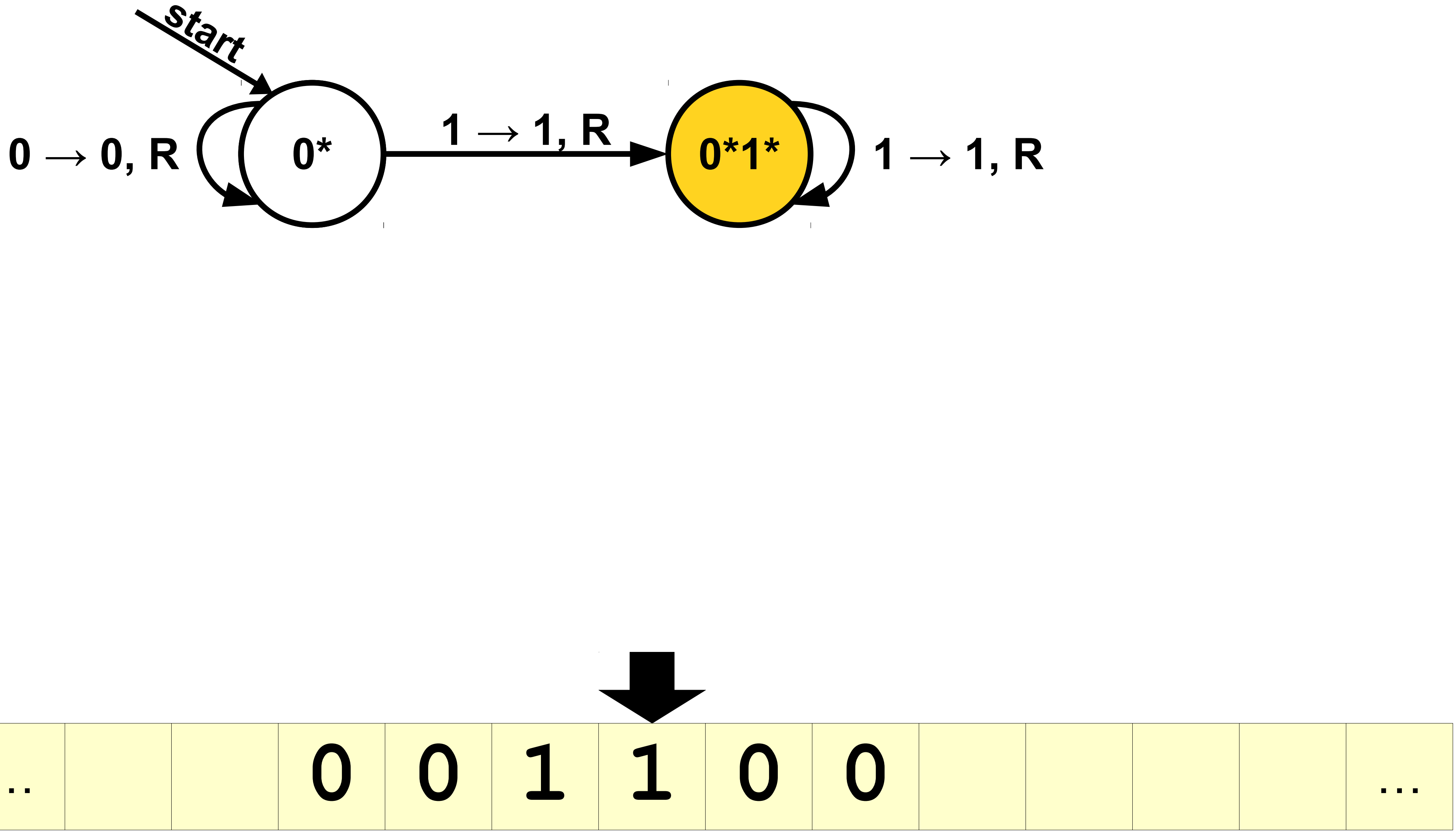

start
0 → 0, R
0*
1 → 1, R
…
0
0
1
1
0
0
…

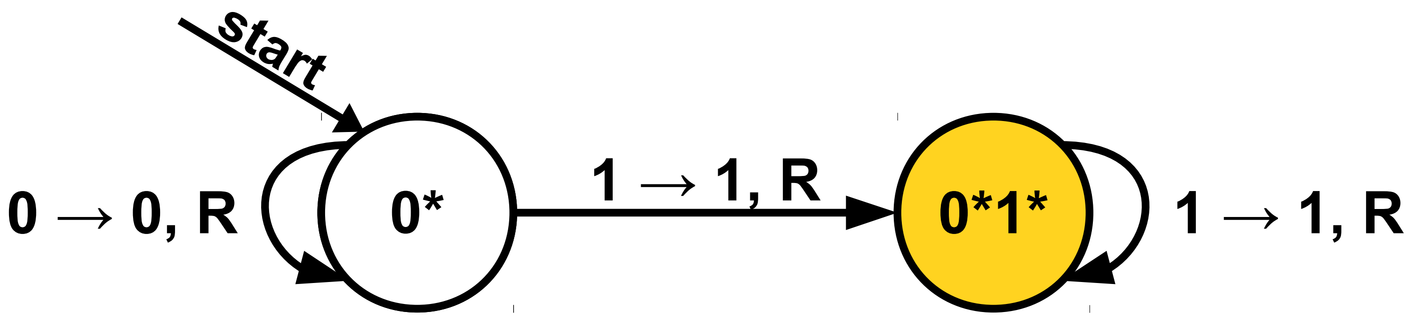


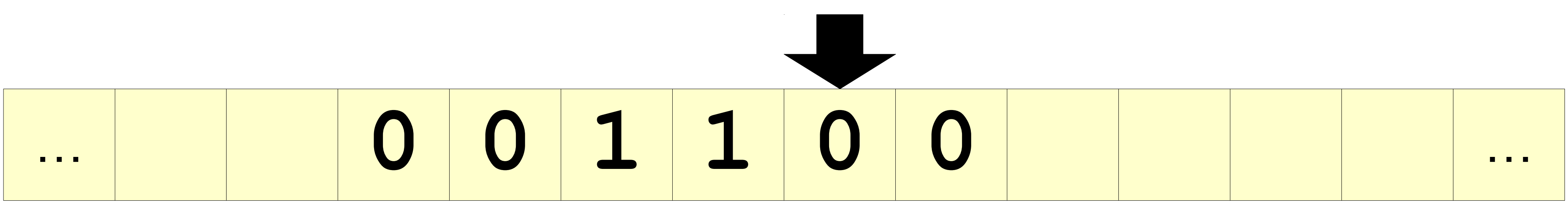
start
0 → 0, R
0*
1 → 1, R
…
0
0
1
1
0
0
…

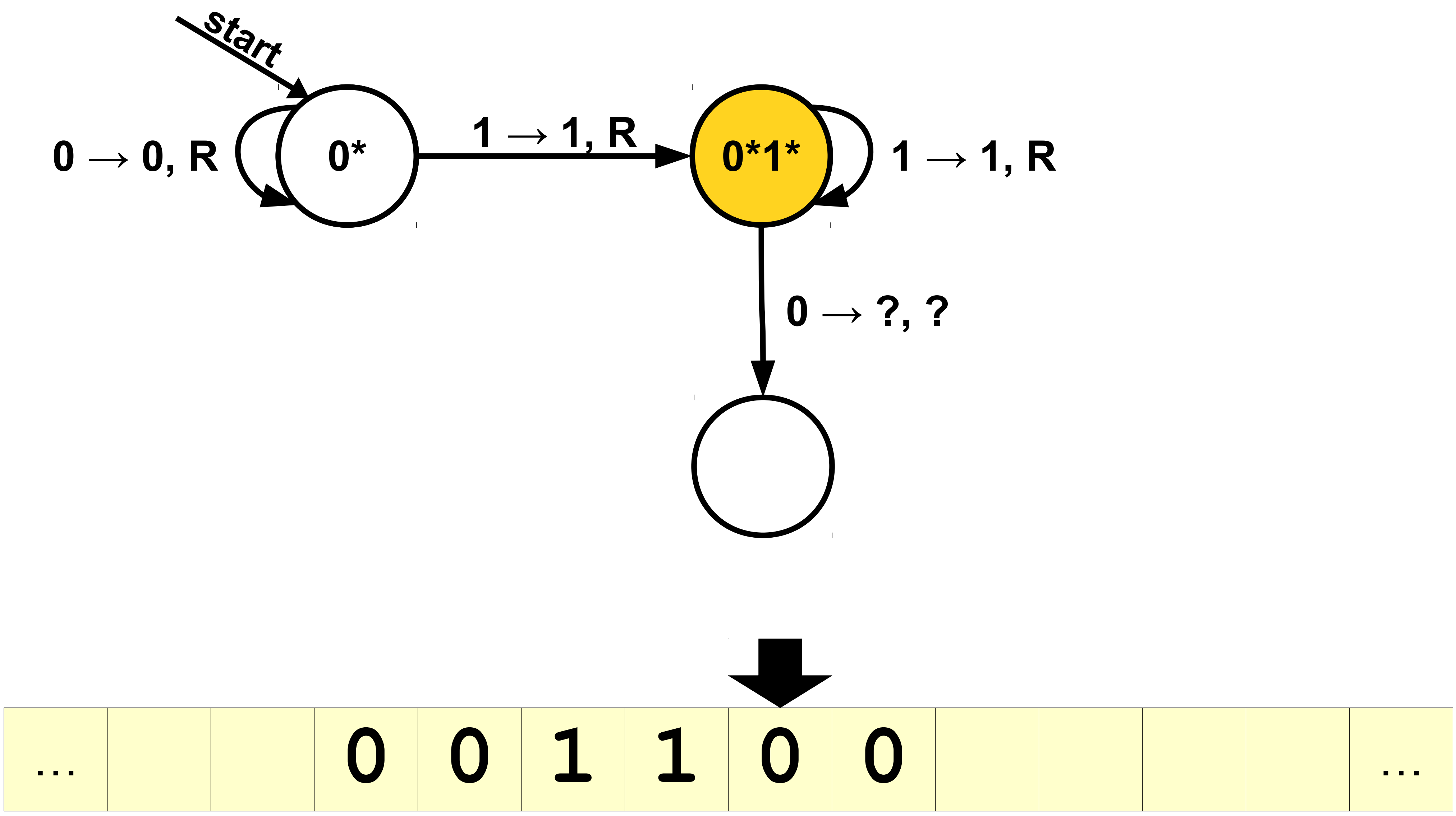


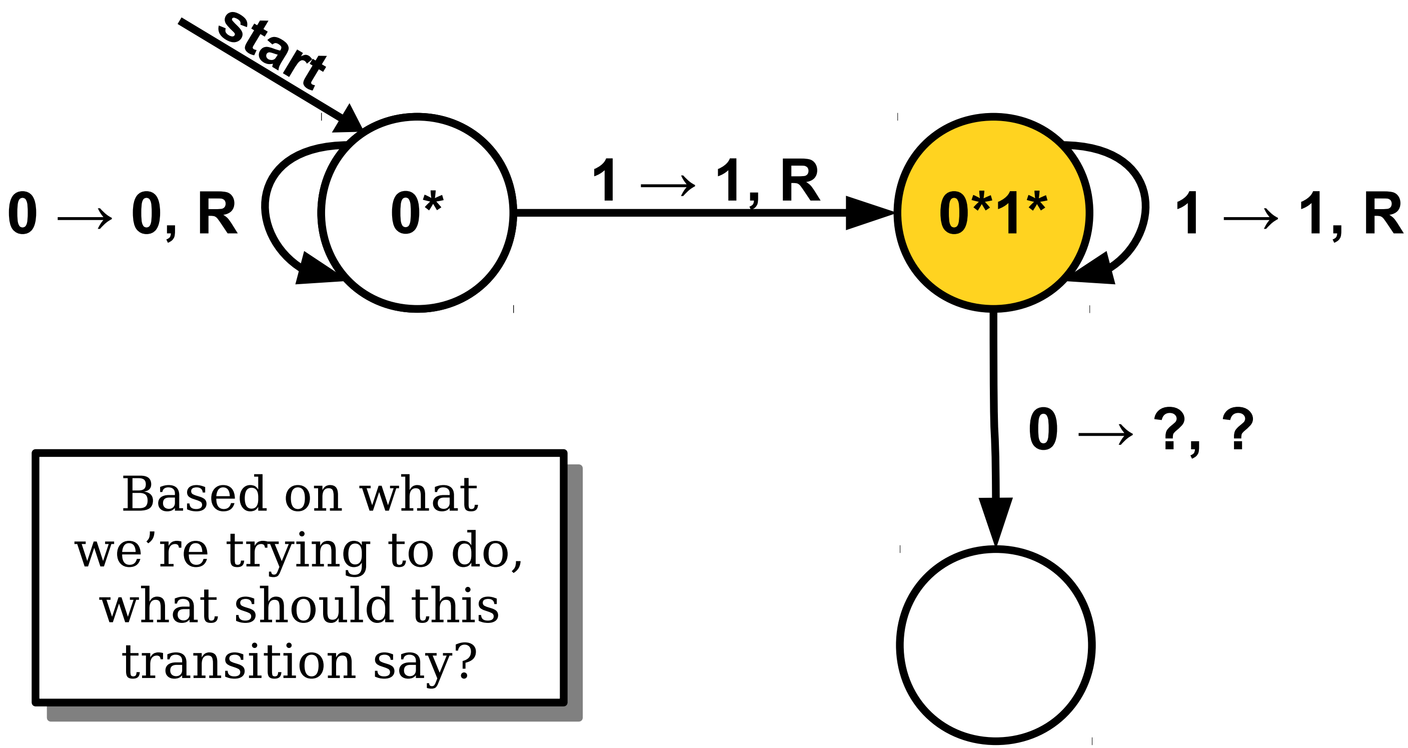
start
0 → 0, R
0*
1 → 1, R
0*1*
…
0
0
1
1
0
0
…


start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
…
0 0 1 1 0 0
…

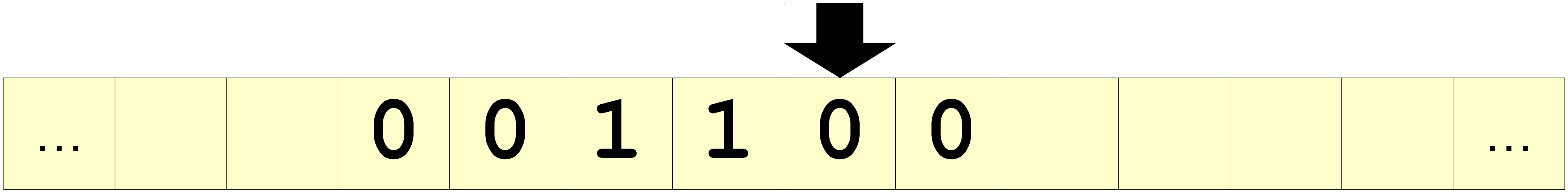

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R

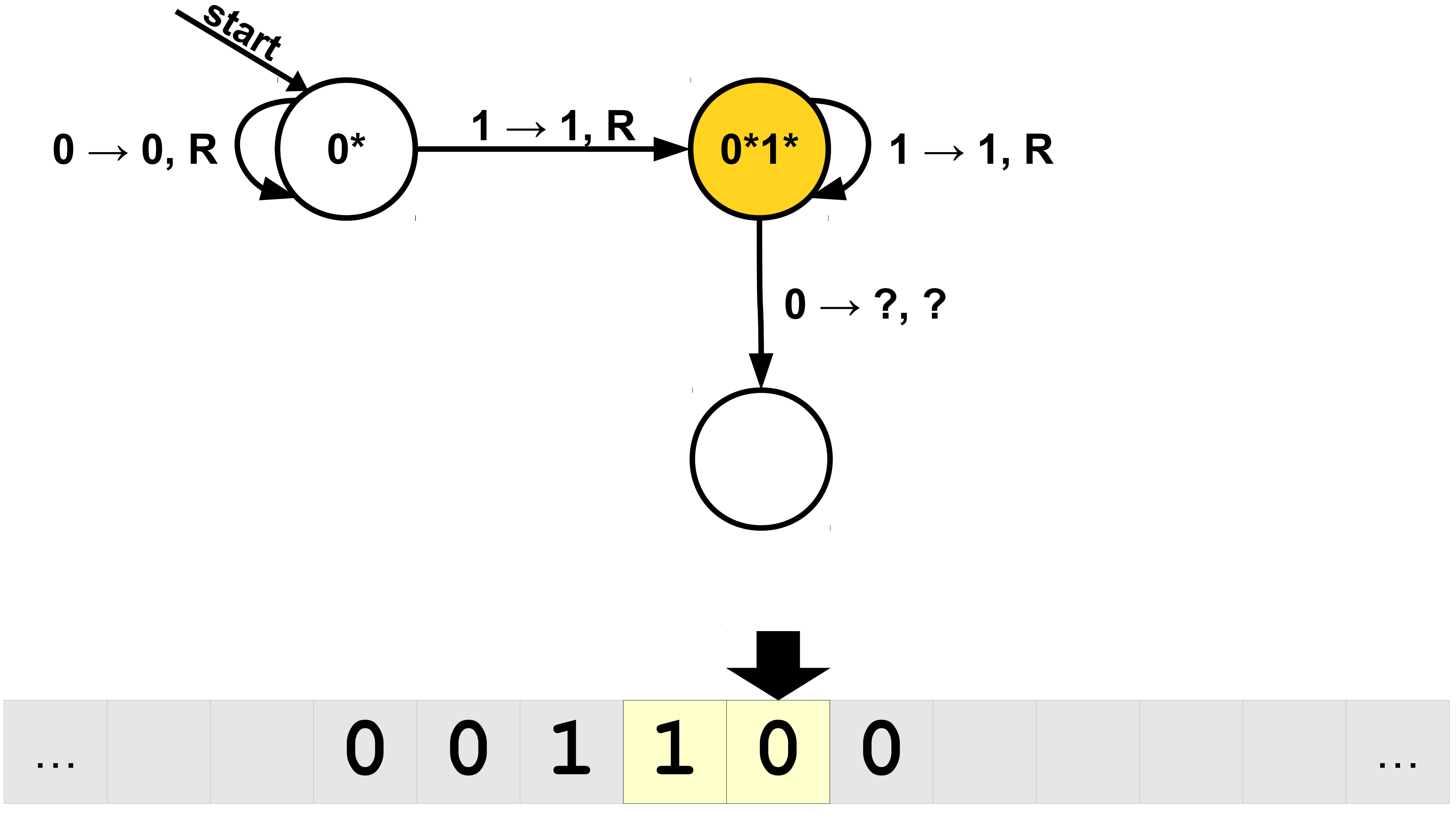

…
0
0
1
1
0
0
…

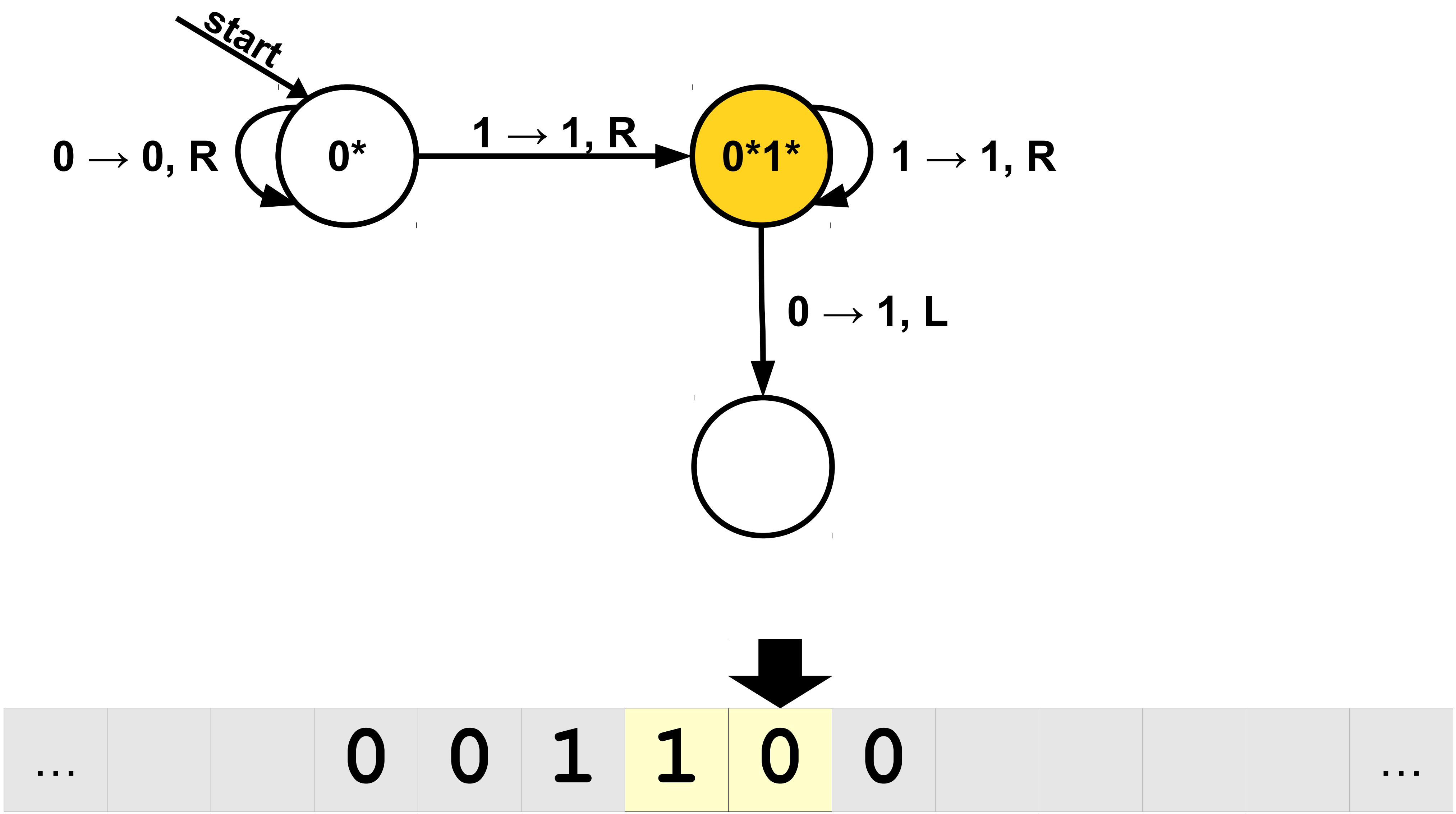

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → ?, ?
…
0
0
1
1
0
0
…



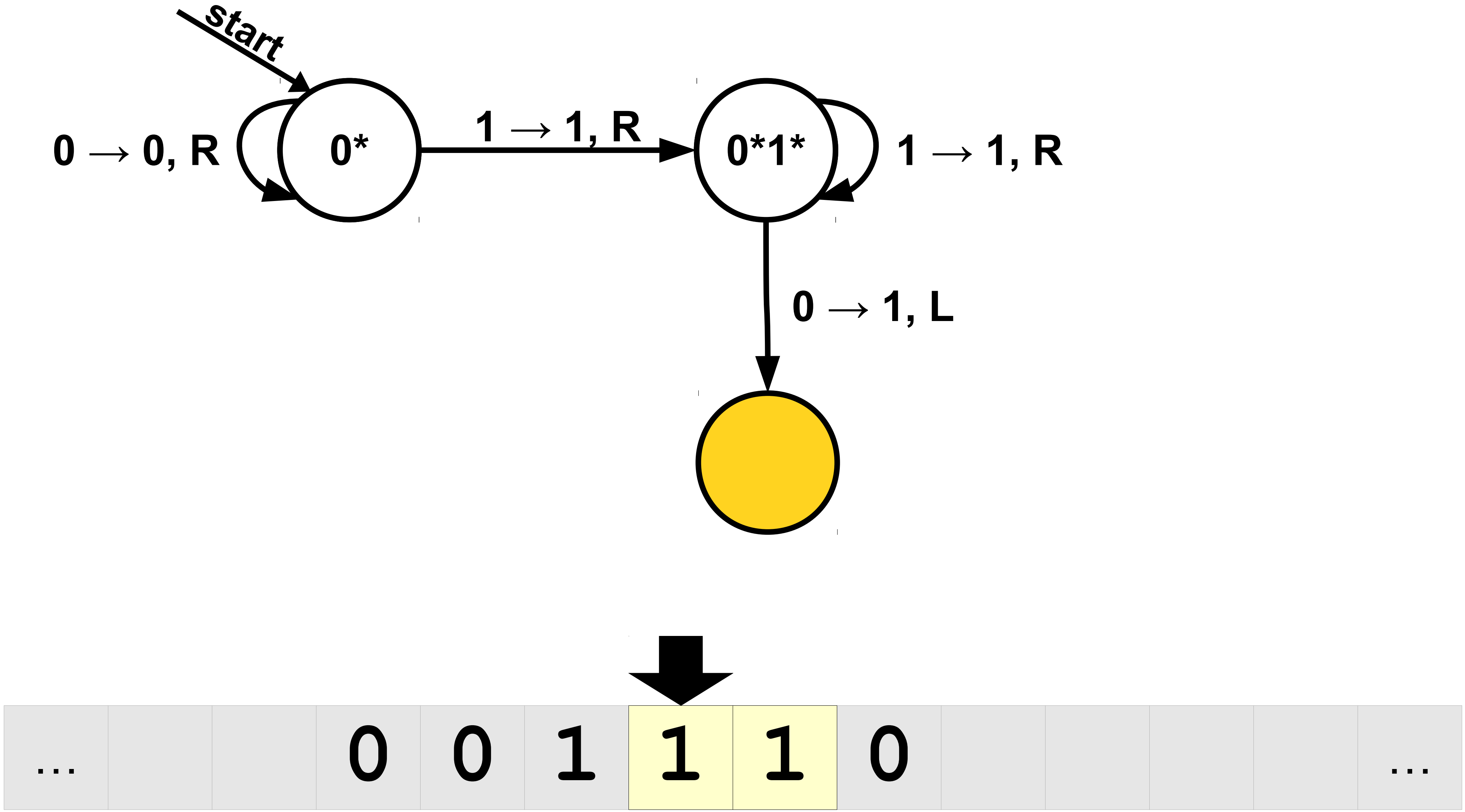
Based on what we're trying to do, what should this transition say?

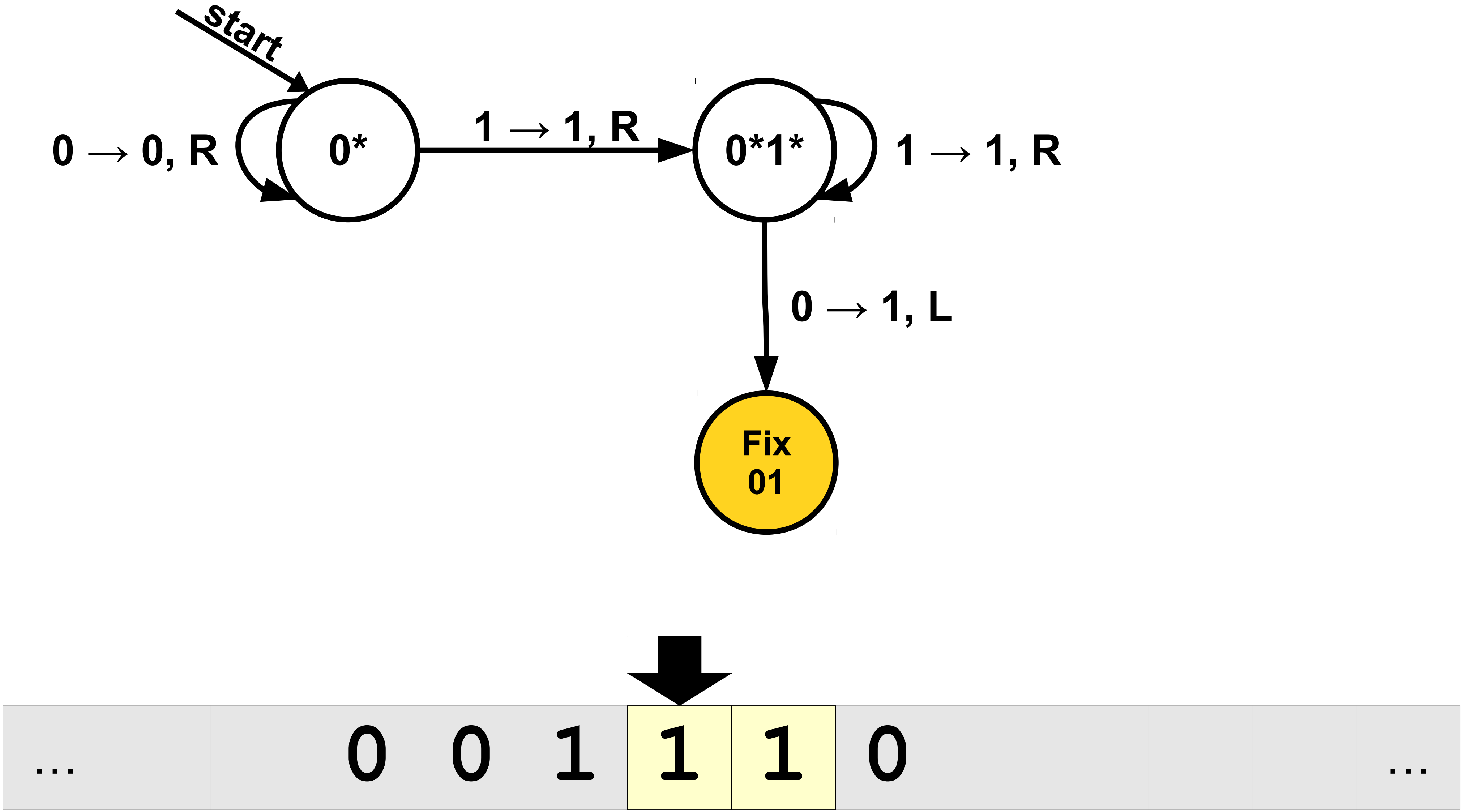


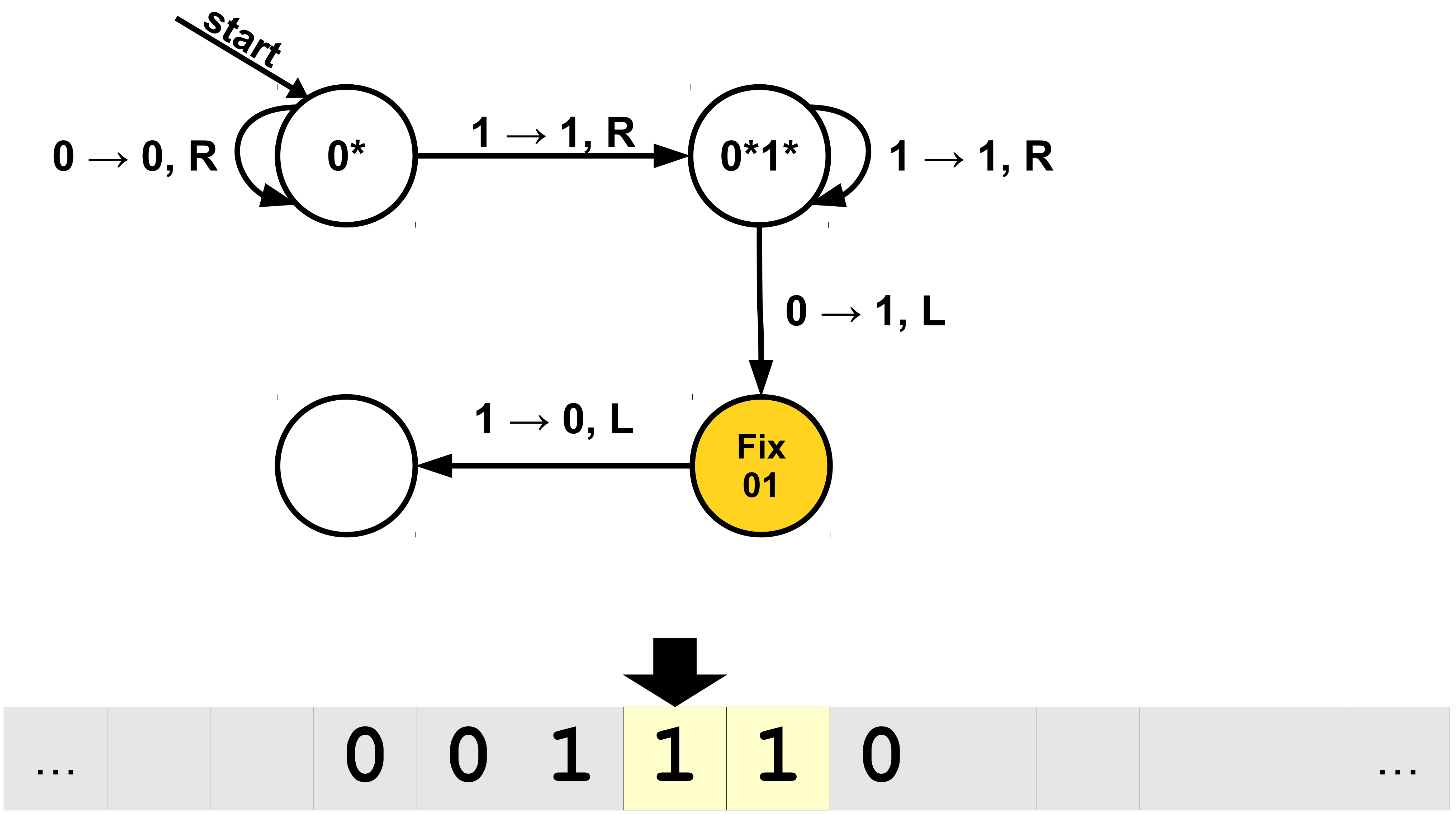

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → ?, ?
... 0 0 1 1 0 0 ...



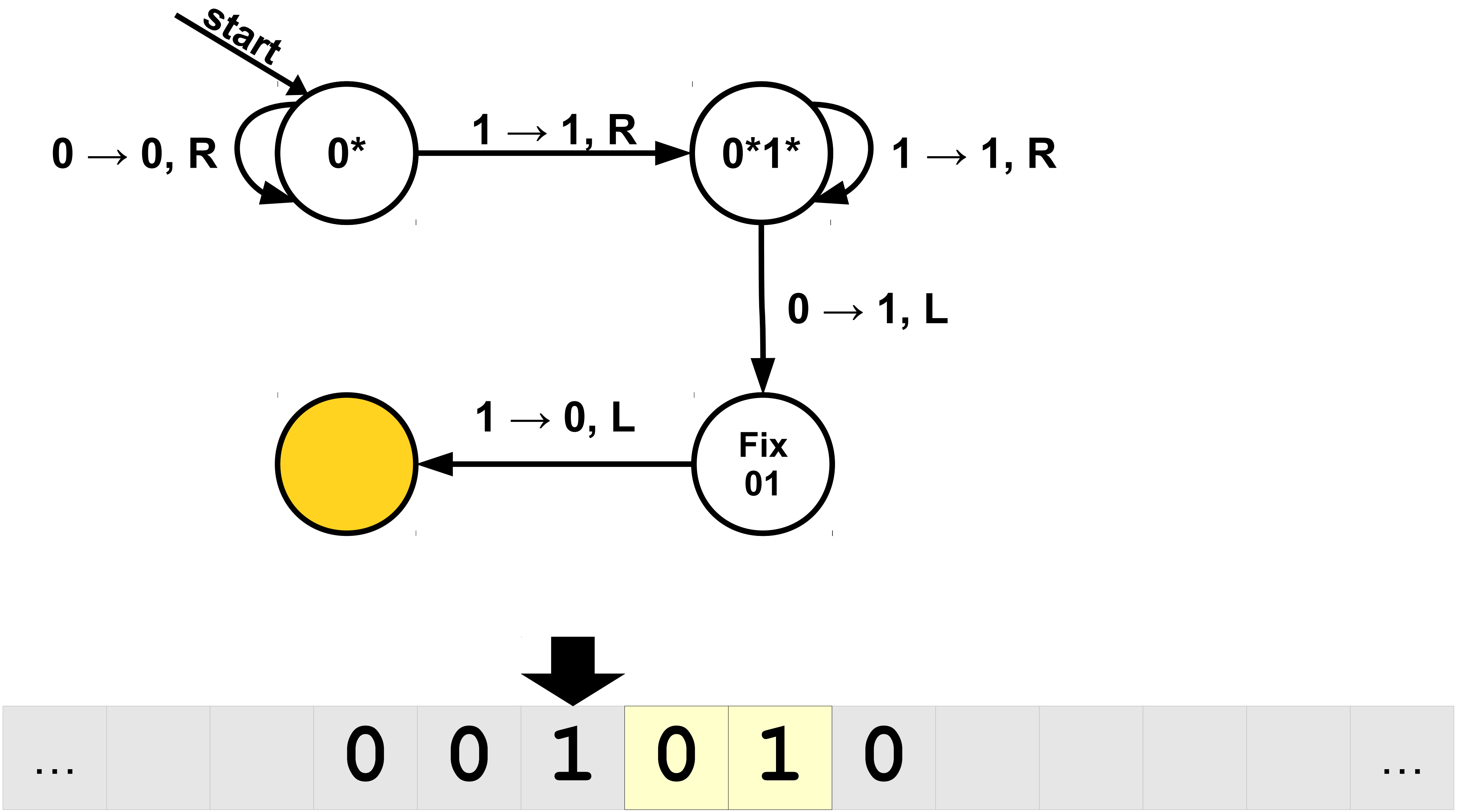
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
...
0
0
1
1
0
0
...

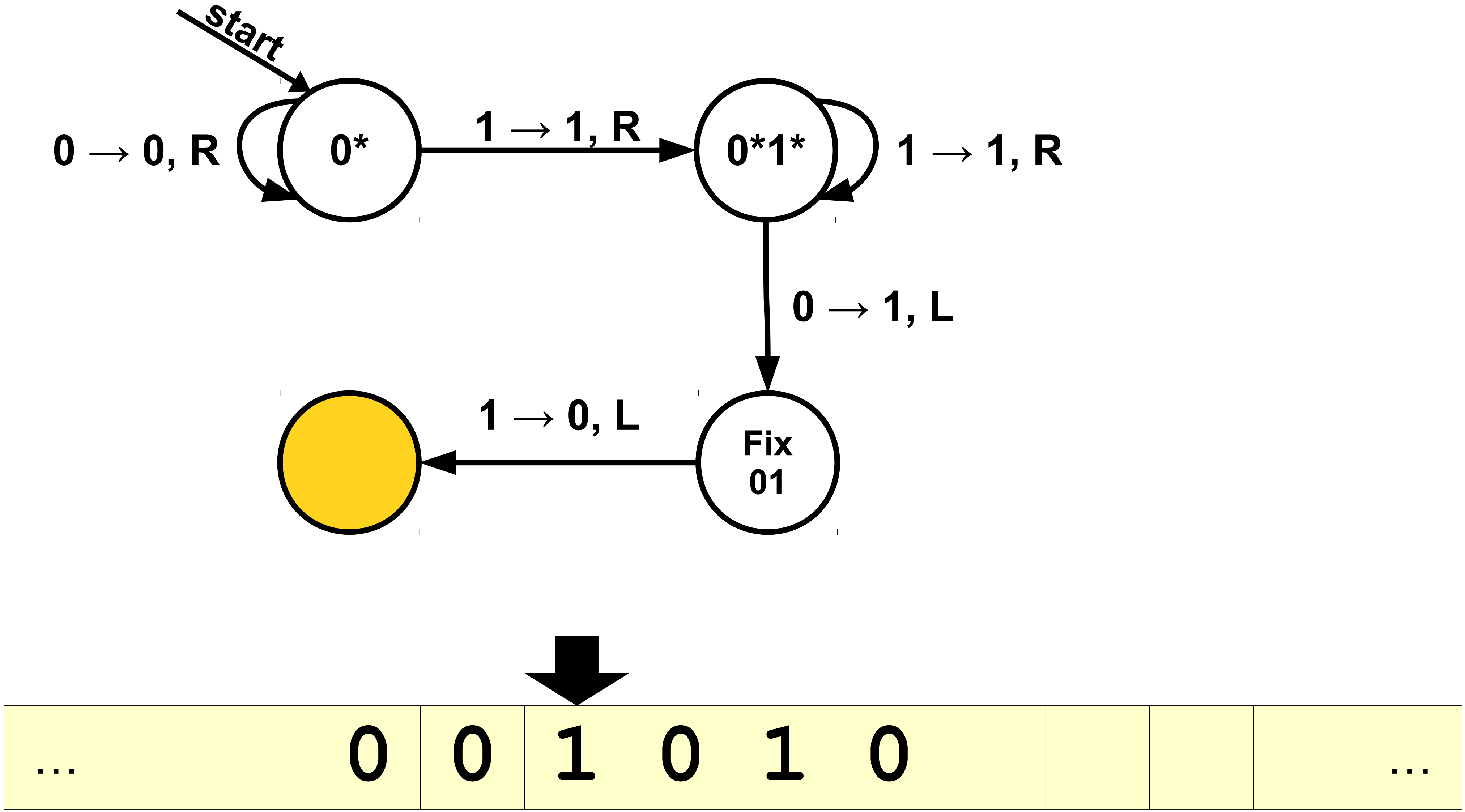

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
…
0
0
1
1
1
0
…

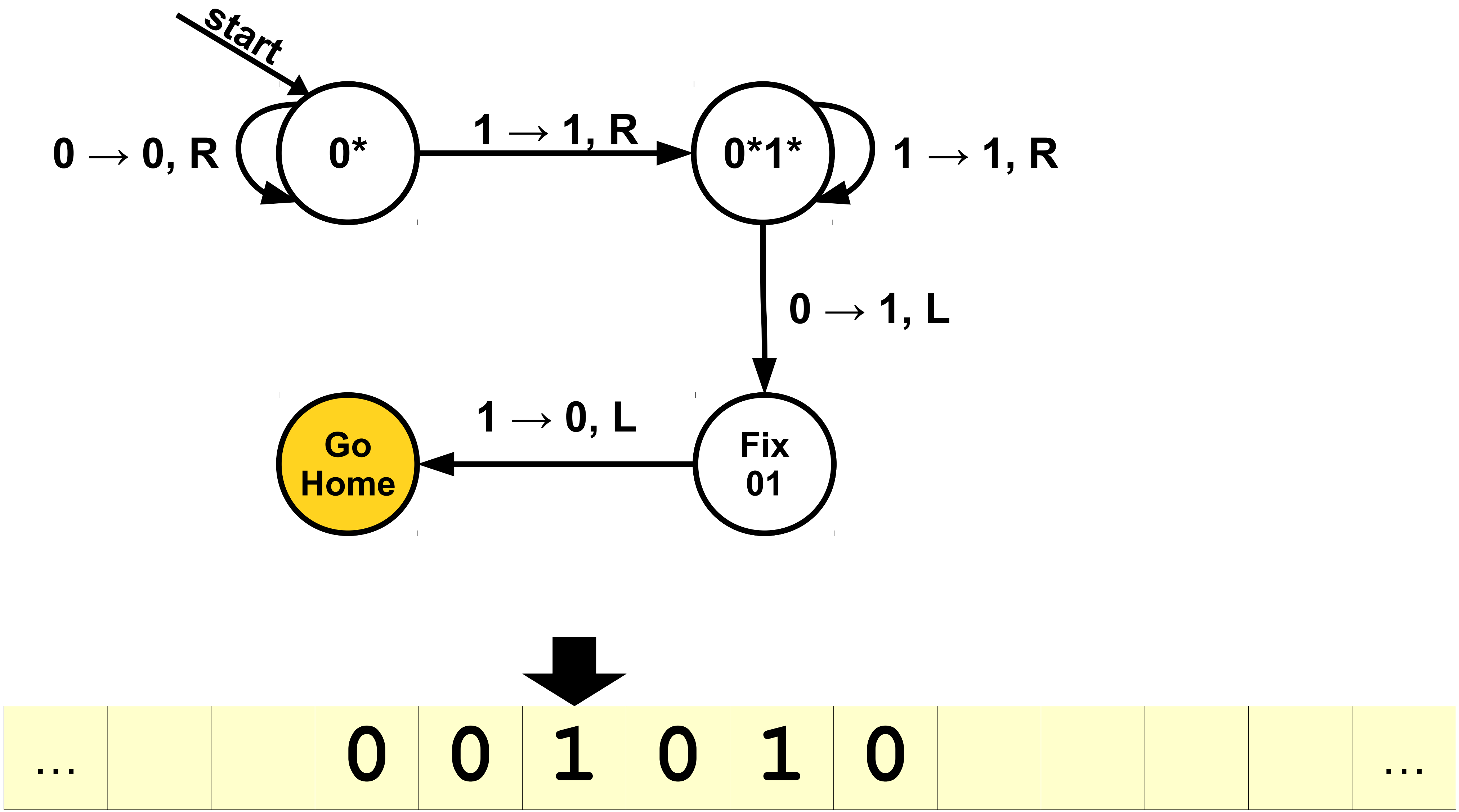

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
Fix
01
… 0 0 1 1 1 0 …

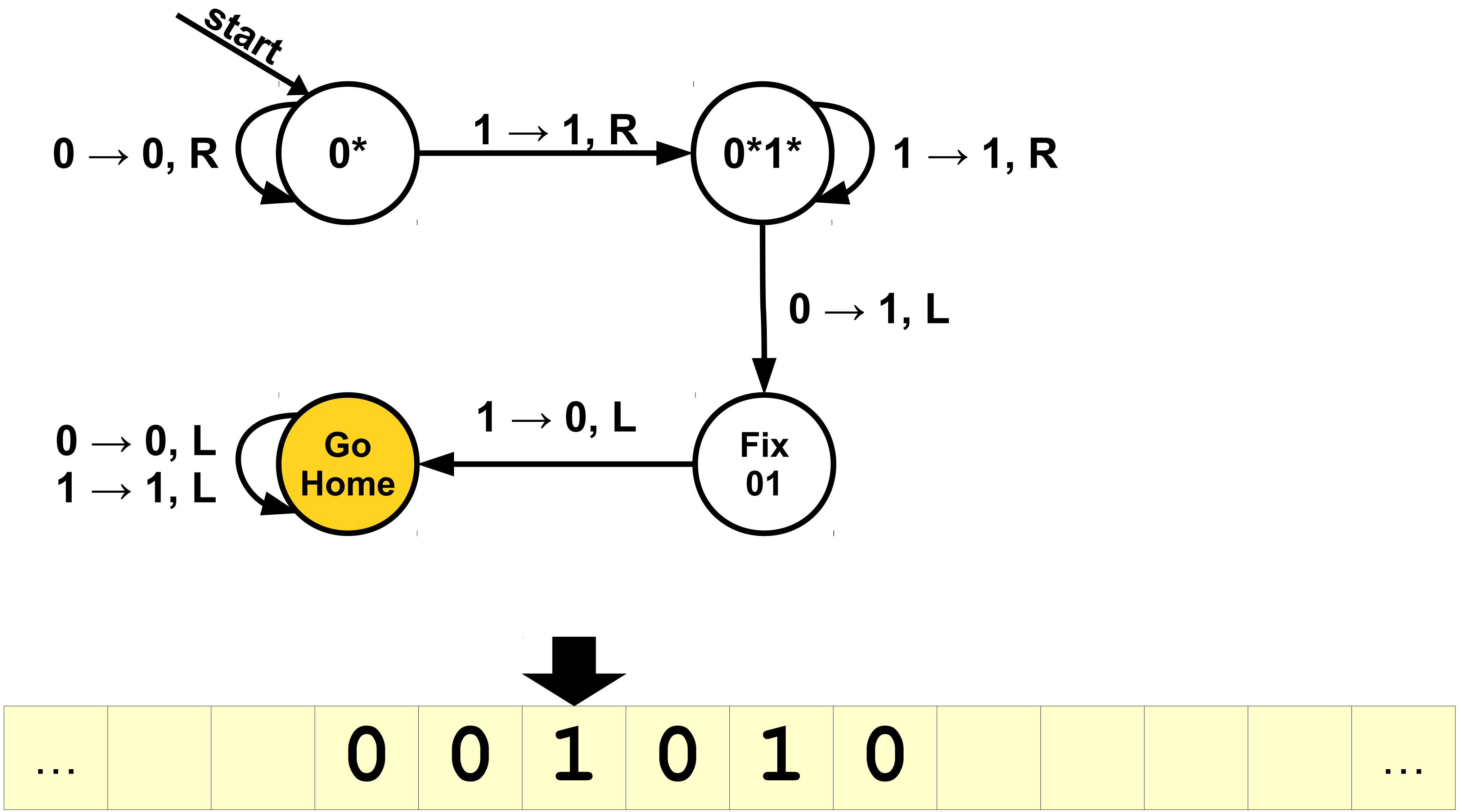


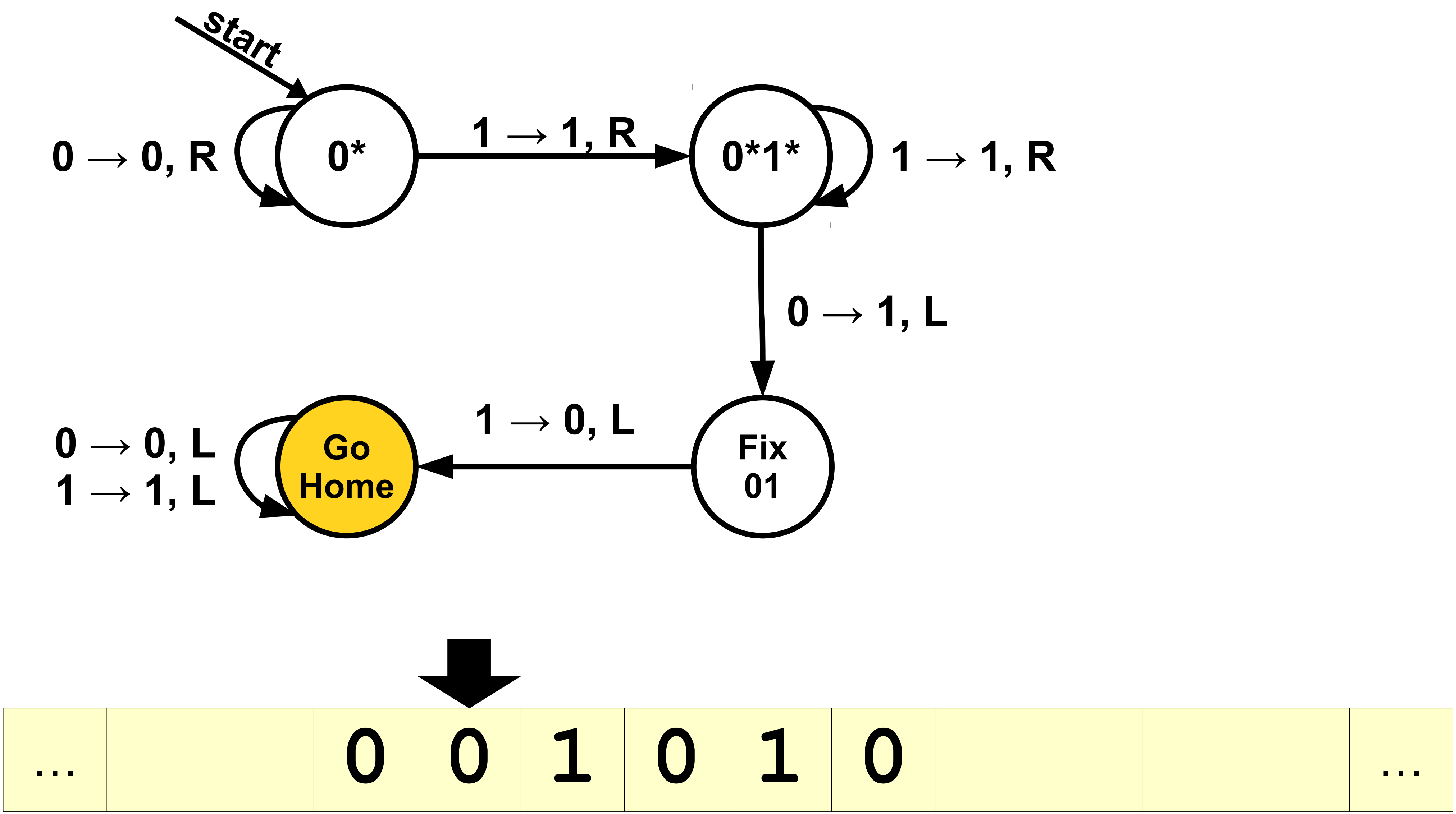
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
Fix 01
…
0
0
1
1
1
0
…

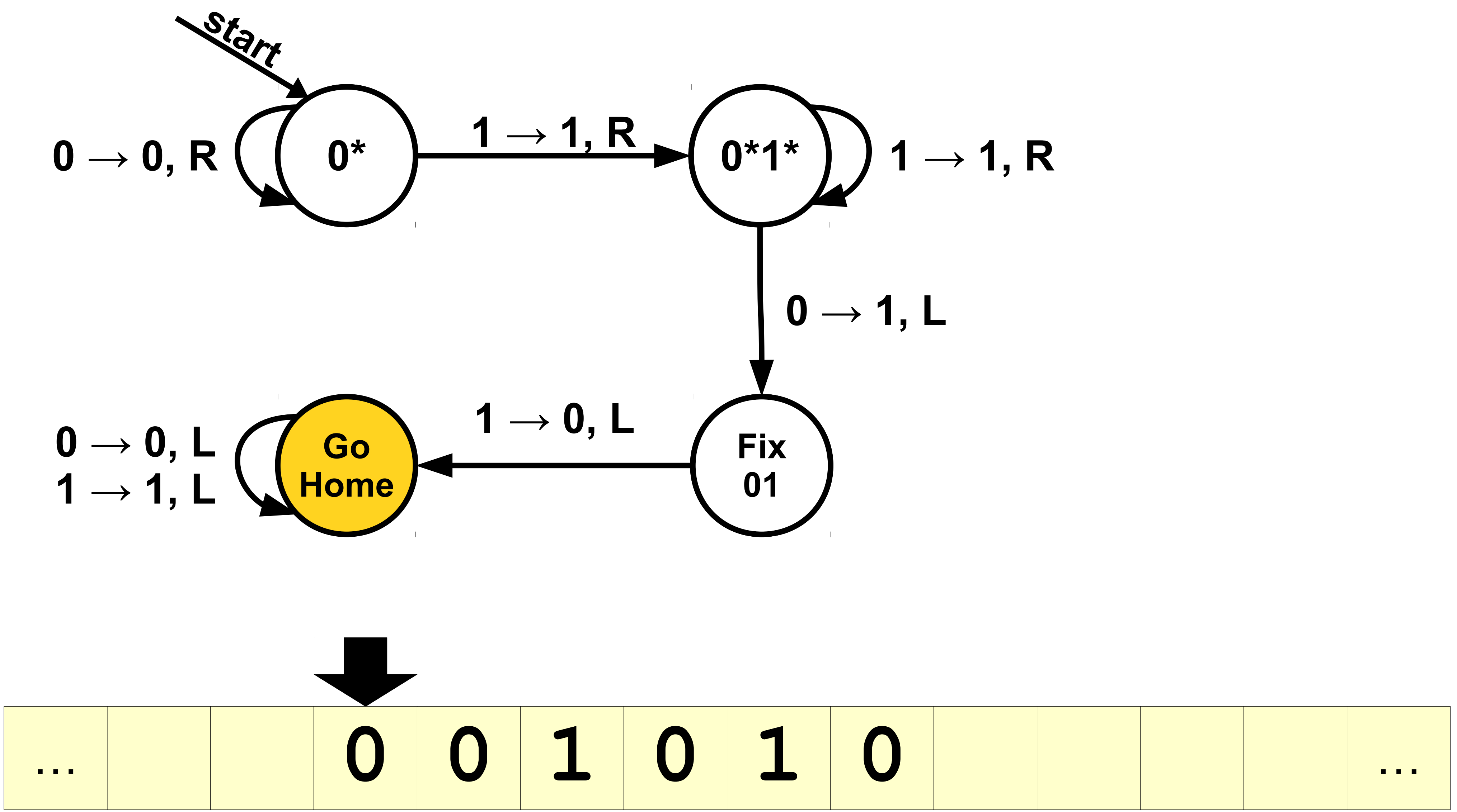

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
Fix
01
...
0
0
1
0
1
0
...


start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
1 → 0, L
Fix
01
…
0
0
1
0
1
0
…

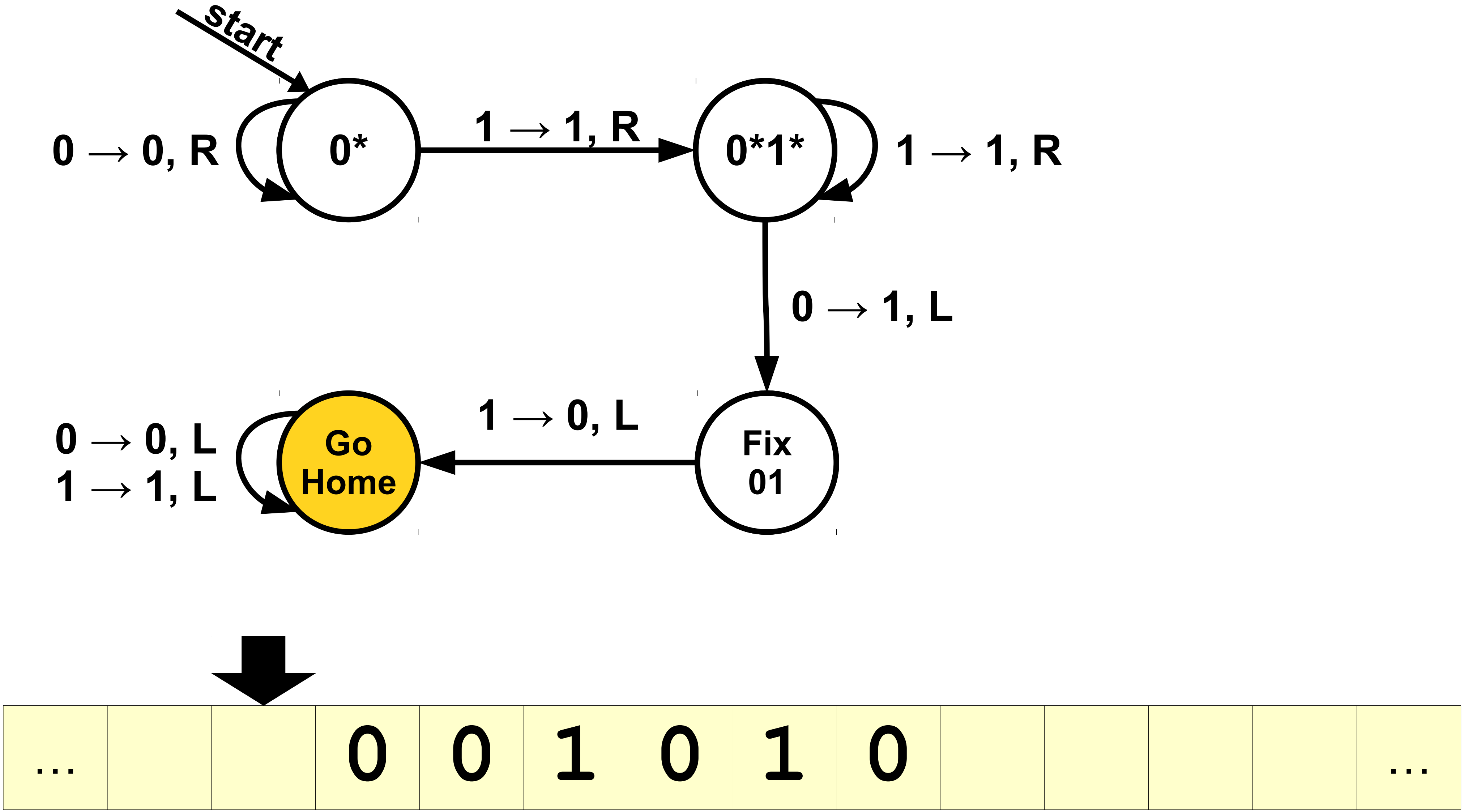

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
1
0
1
0
…


start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
… 0 0 1 0 1 0 …

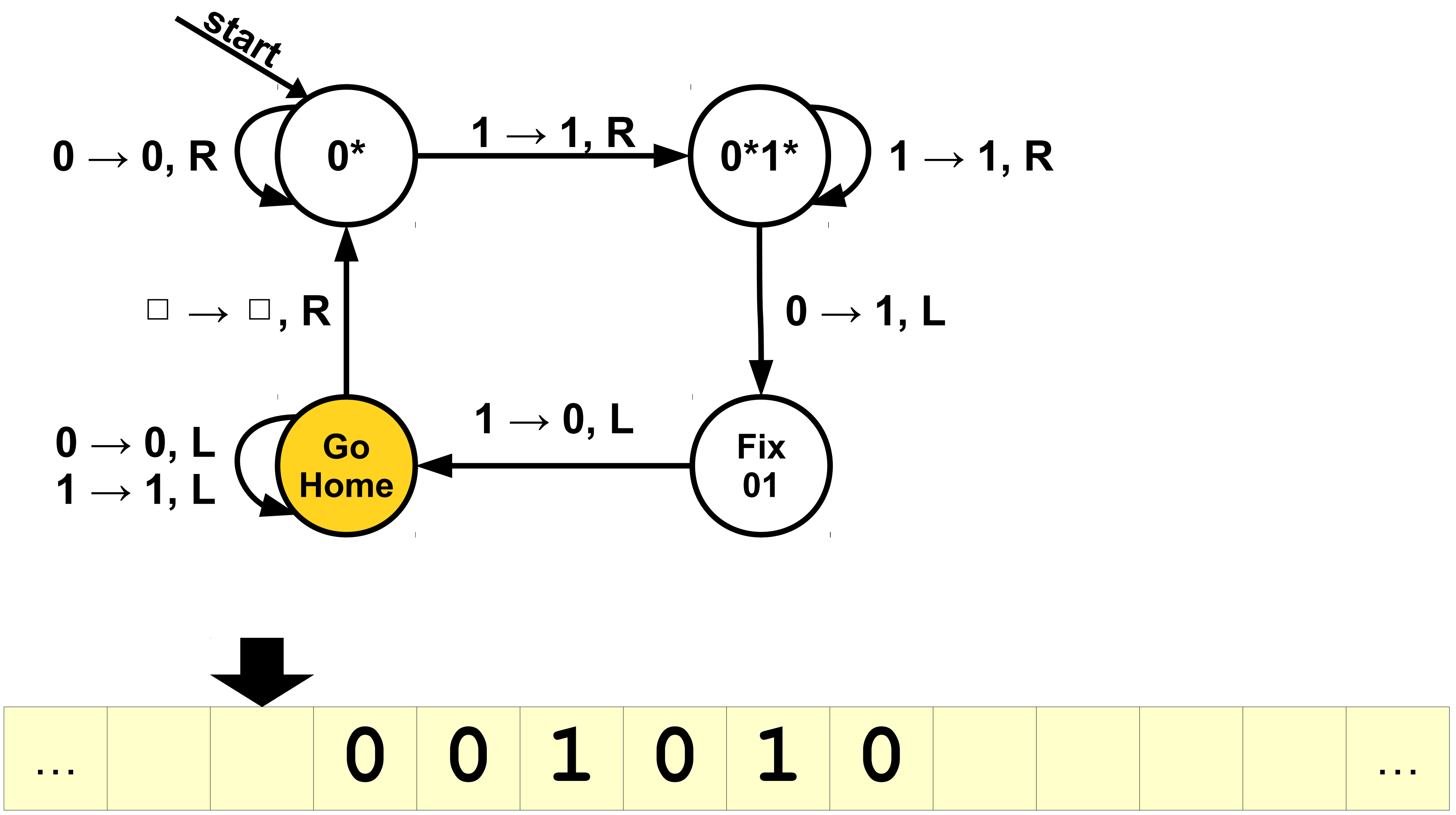

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
...
0
0
1
0
1
0
...

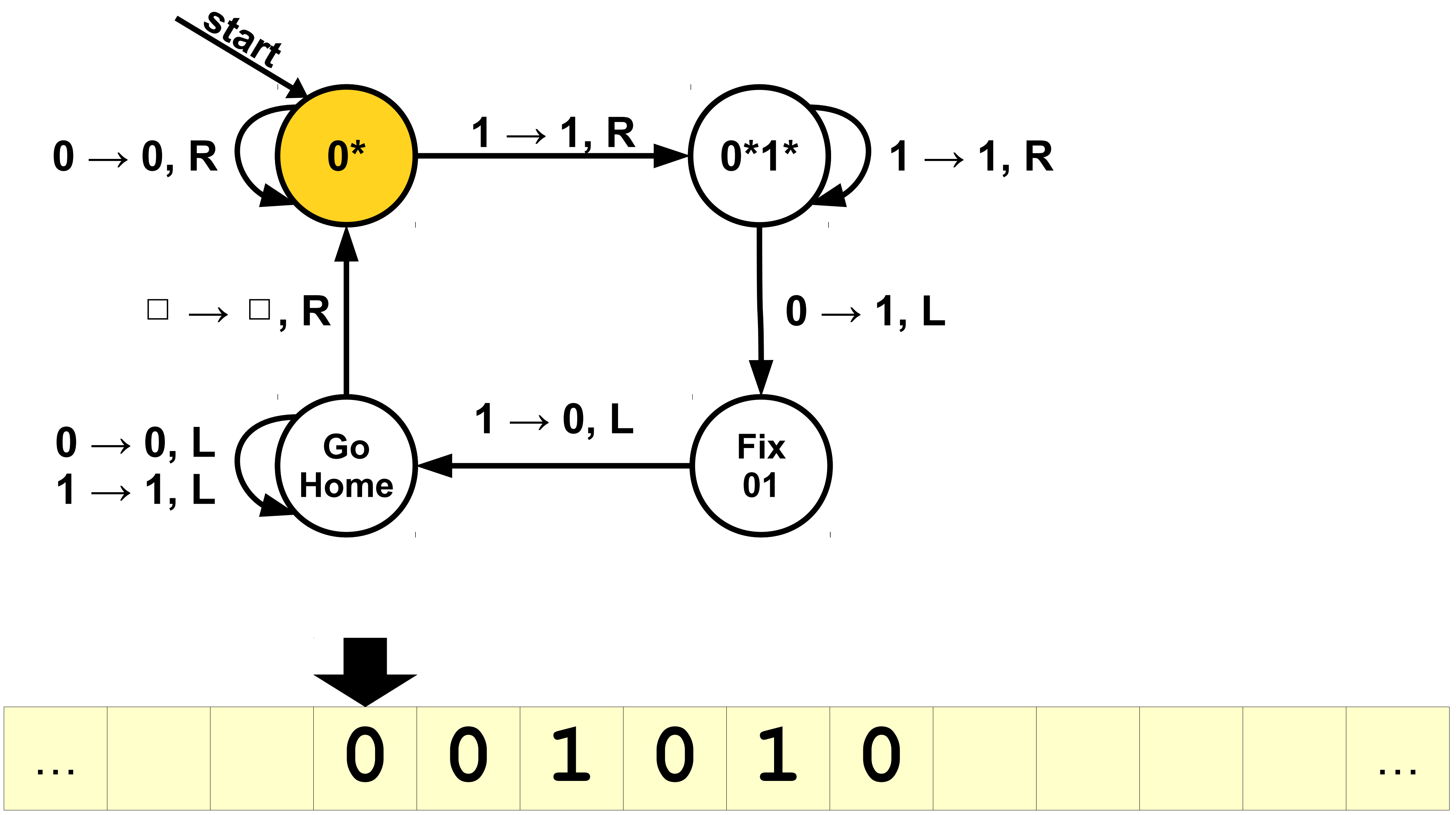


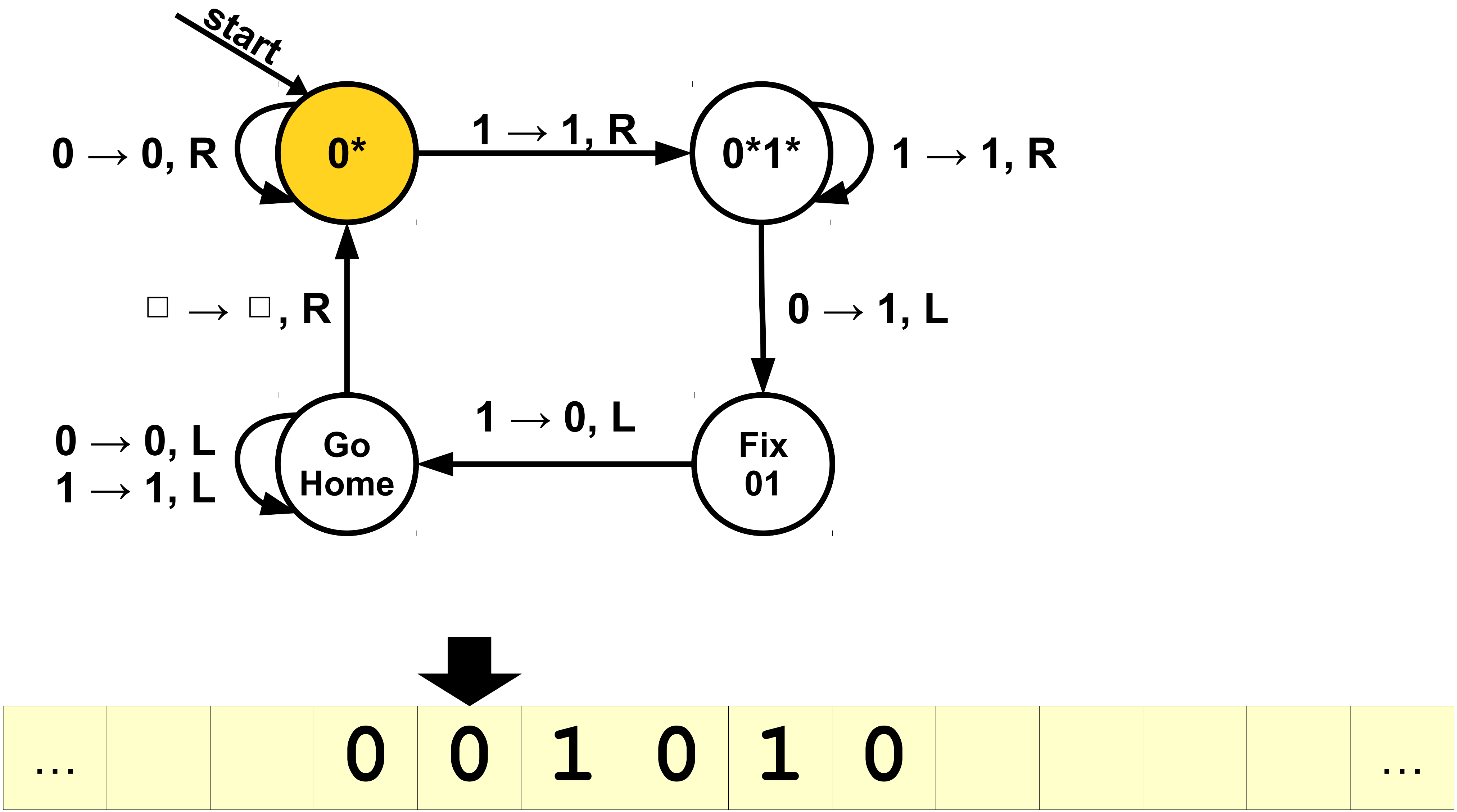
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
1
0
1
0
…



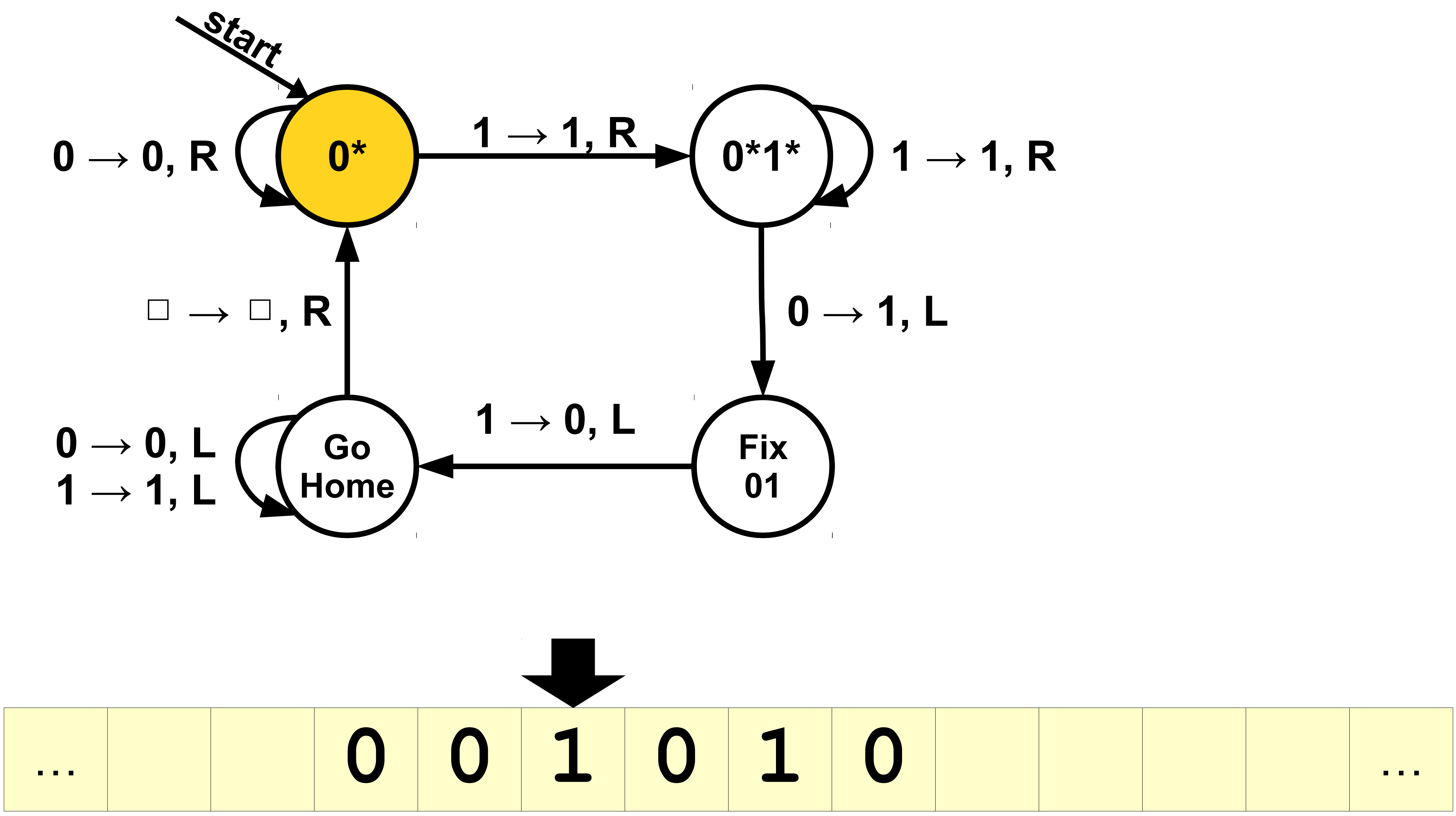
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
Fix 01
1 → 0, L
Go Home
0 → 0, L
1 → 1, L
… 0 0 1 0 1 0 …

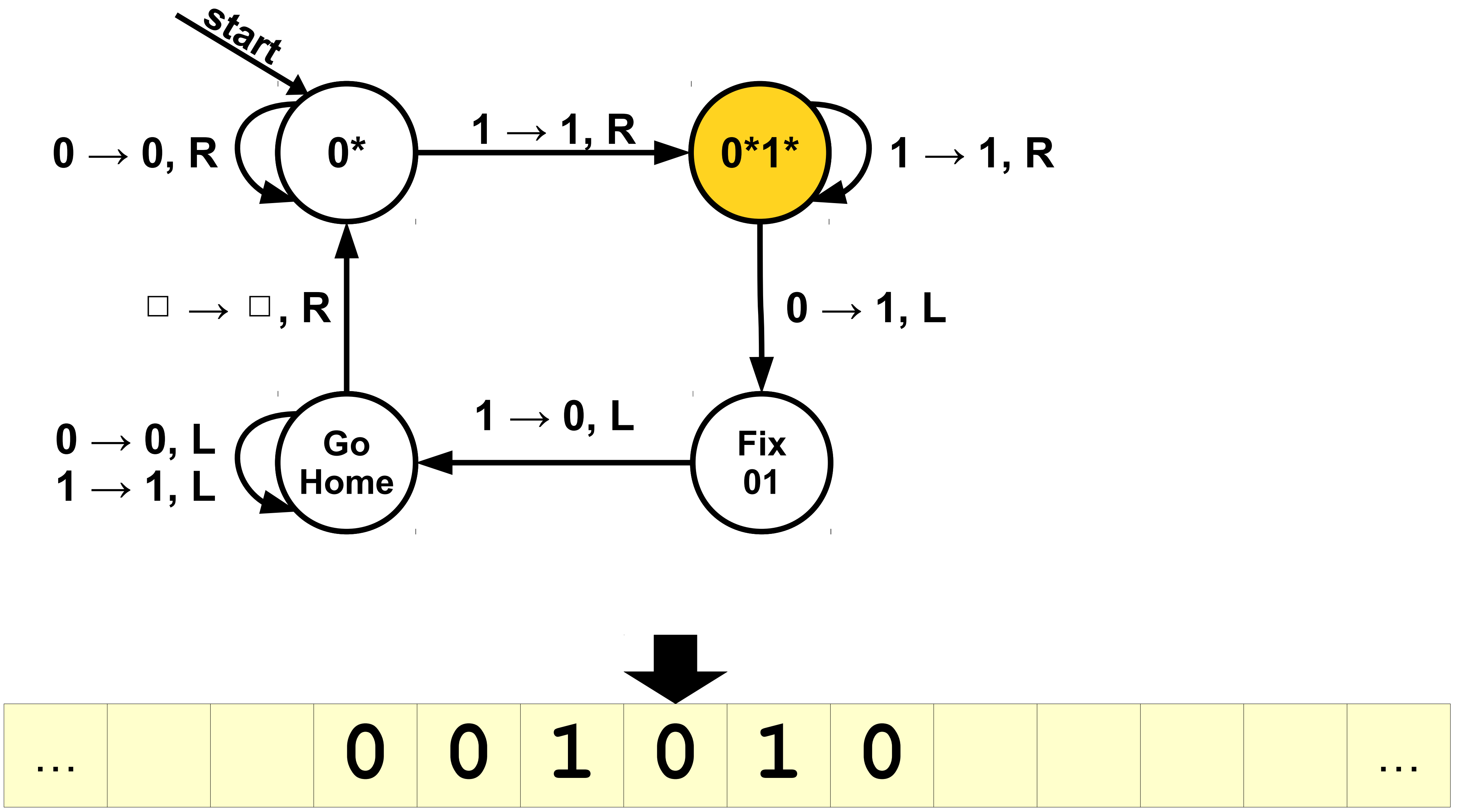

start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
1
0
1
0
…

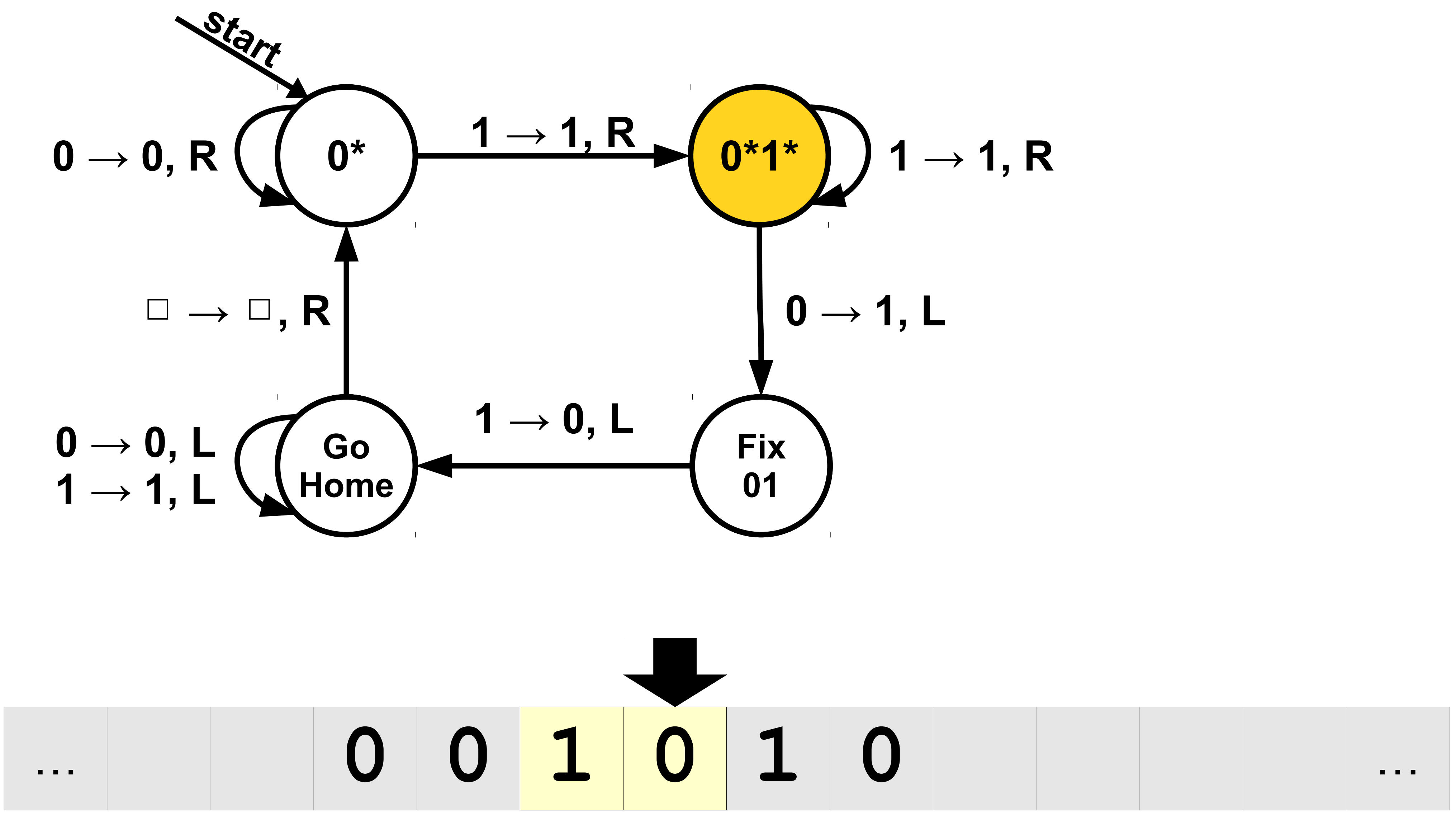

start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
1
0
1
0
…

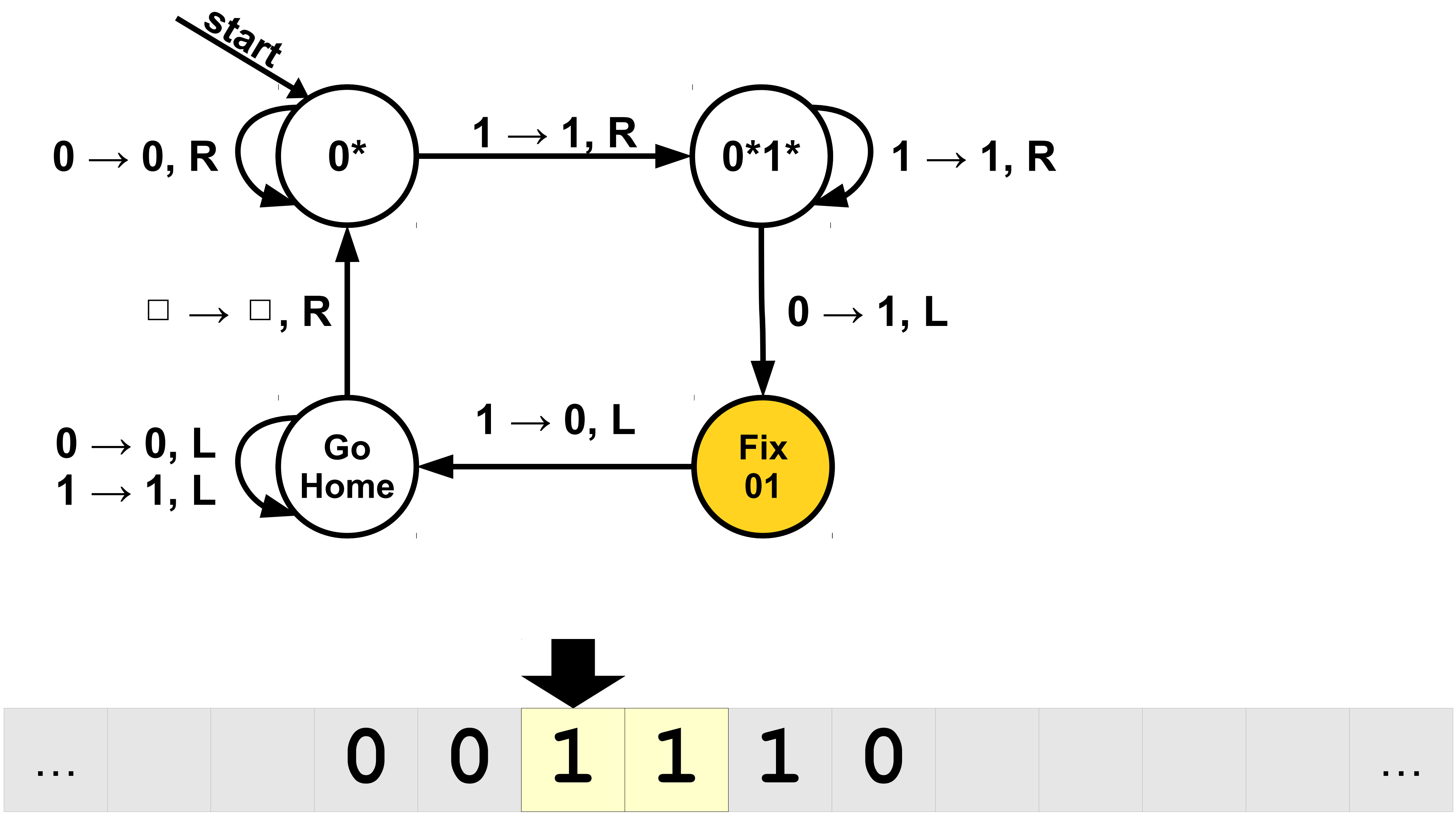


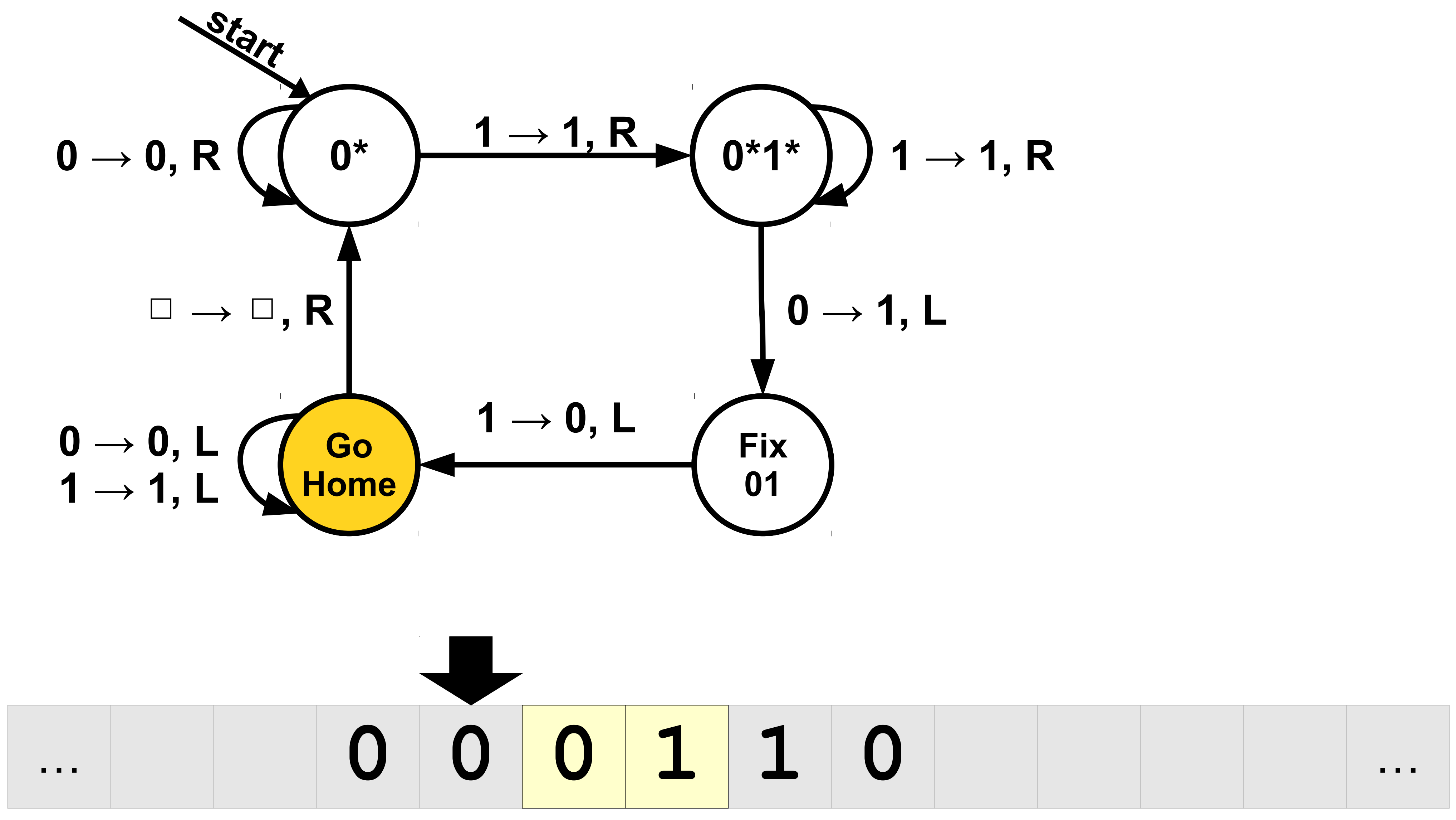
start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0 0 1 0 1 0
…

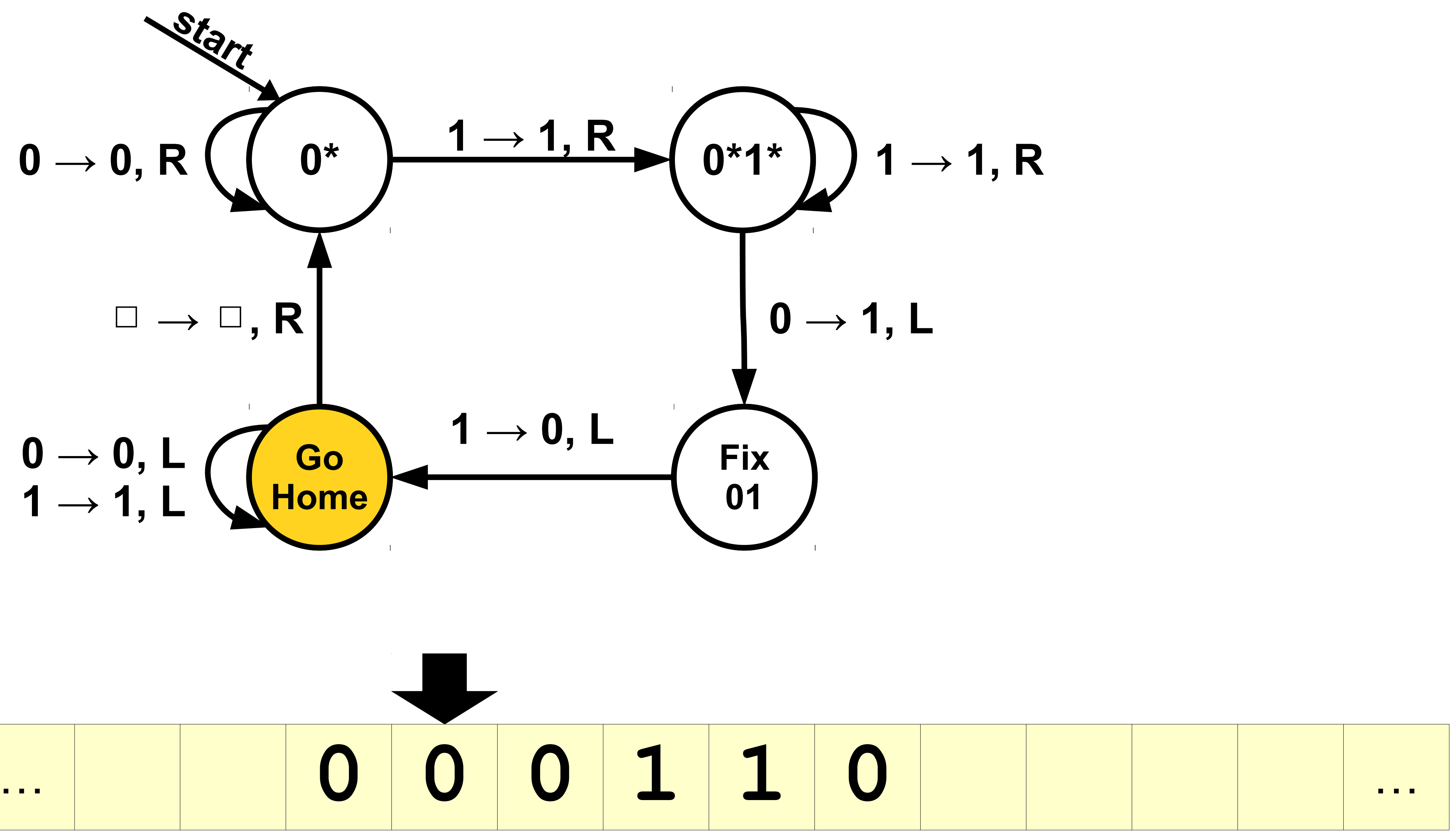

start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
Fix 01
1 → 0, L
Go Home
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
1
0
1
0
…

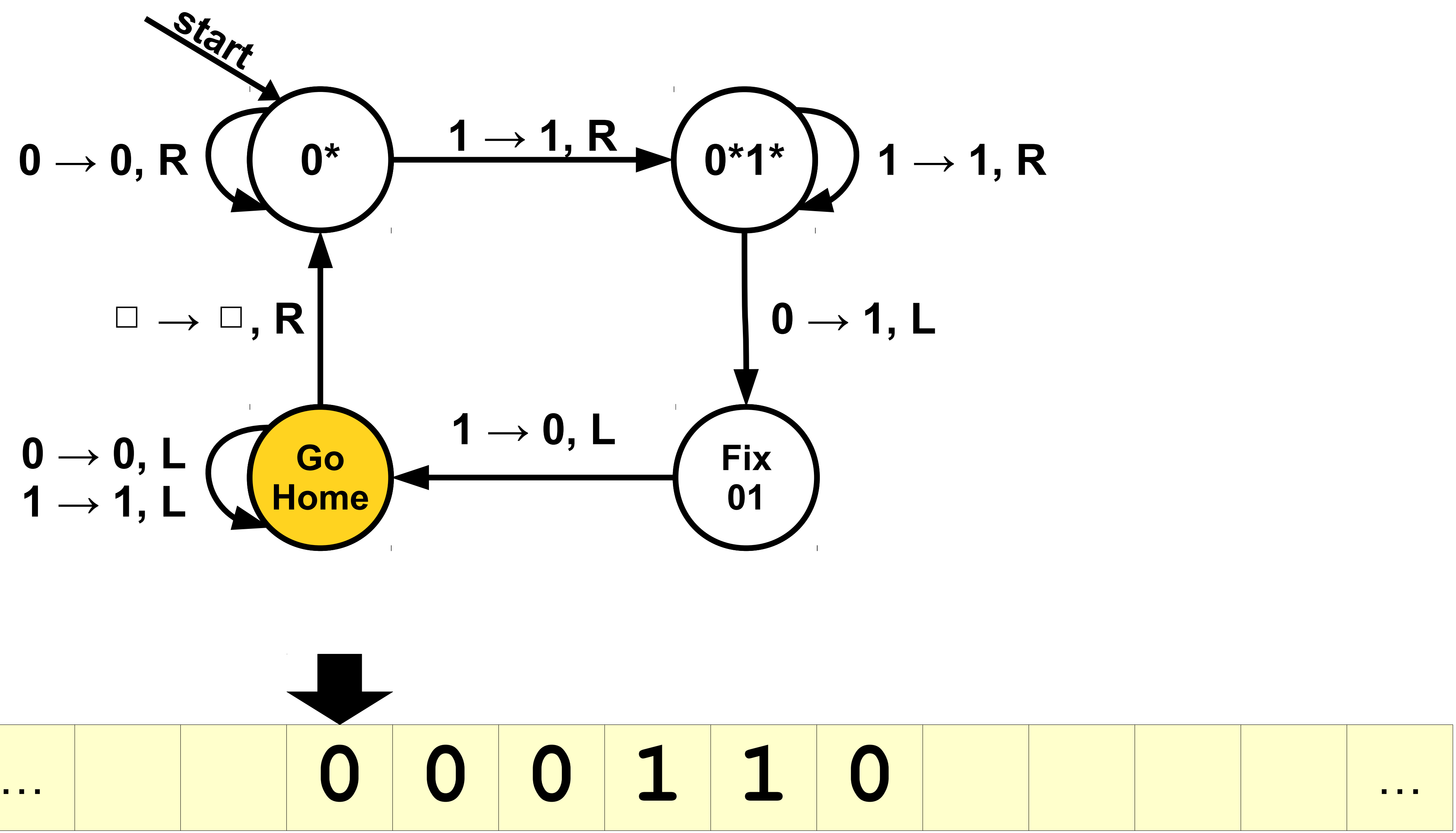

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
1
0
1
0
…

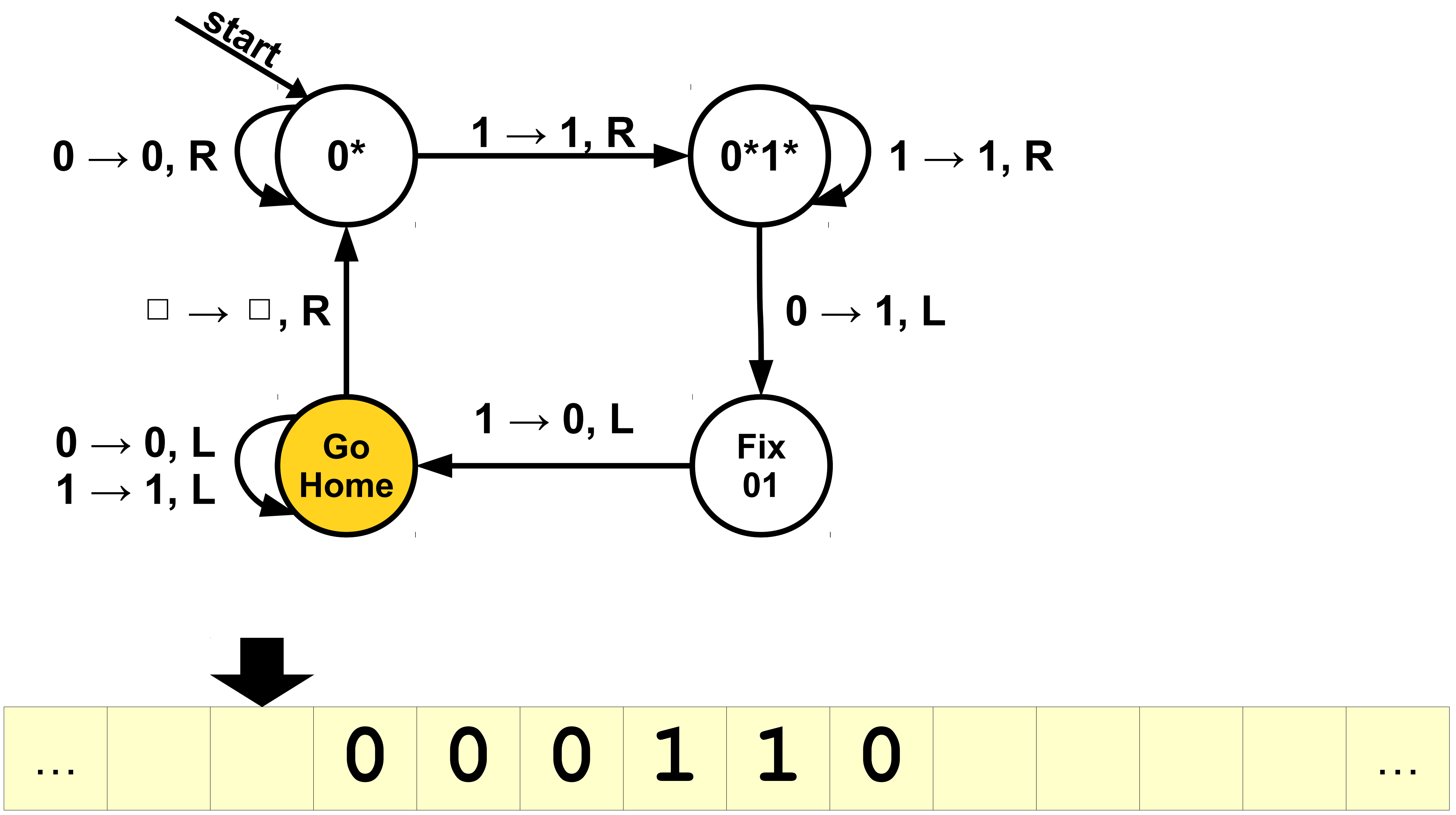

start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 1 0 1 0 …

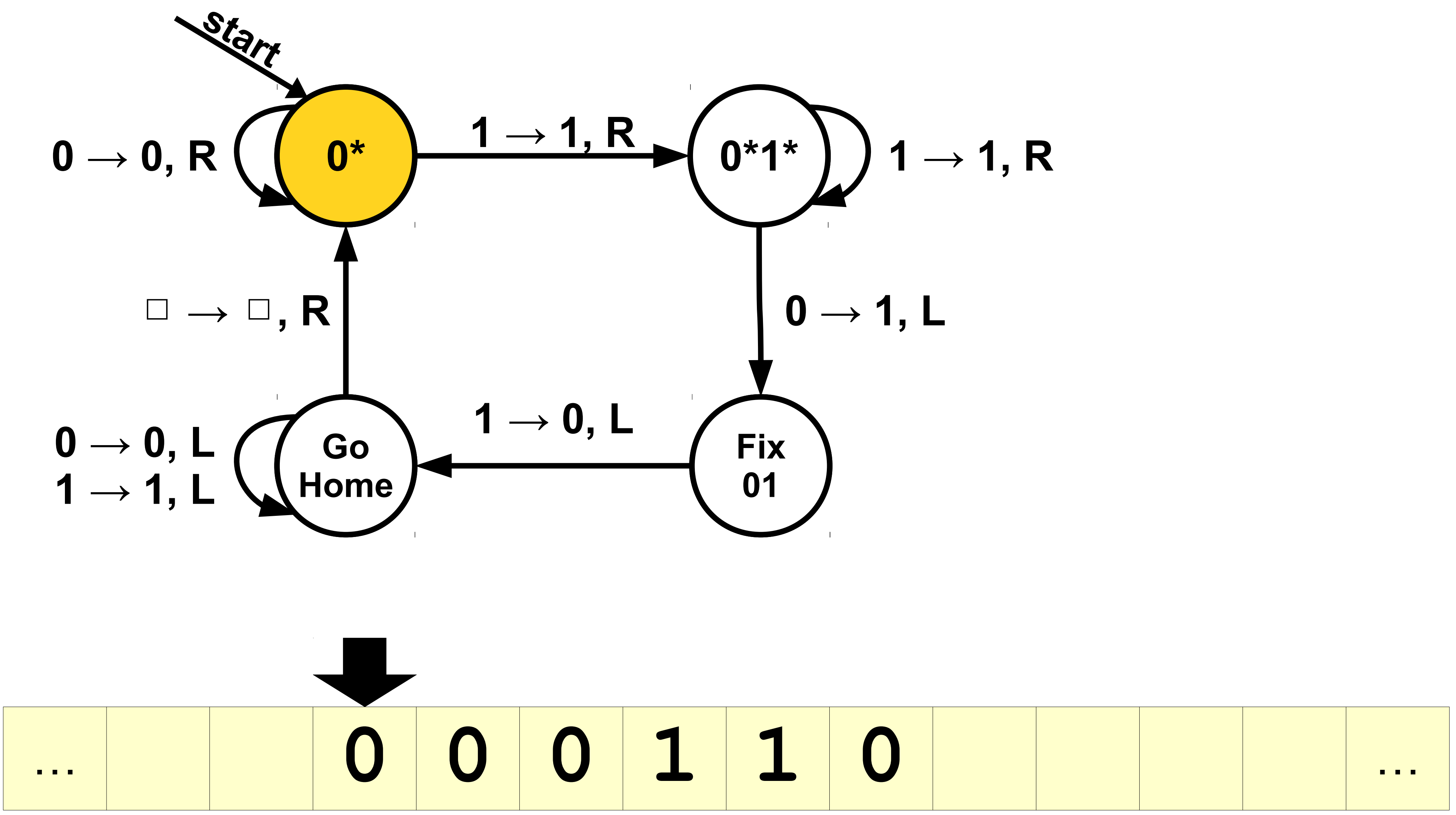

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
1
1
1
0
…

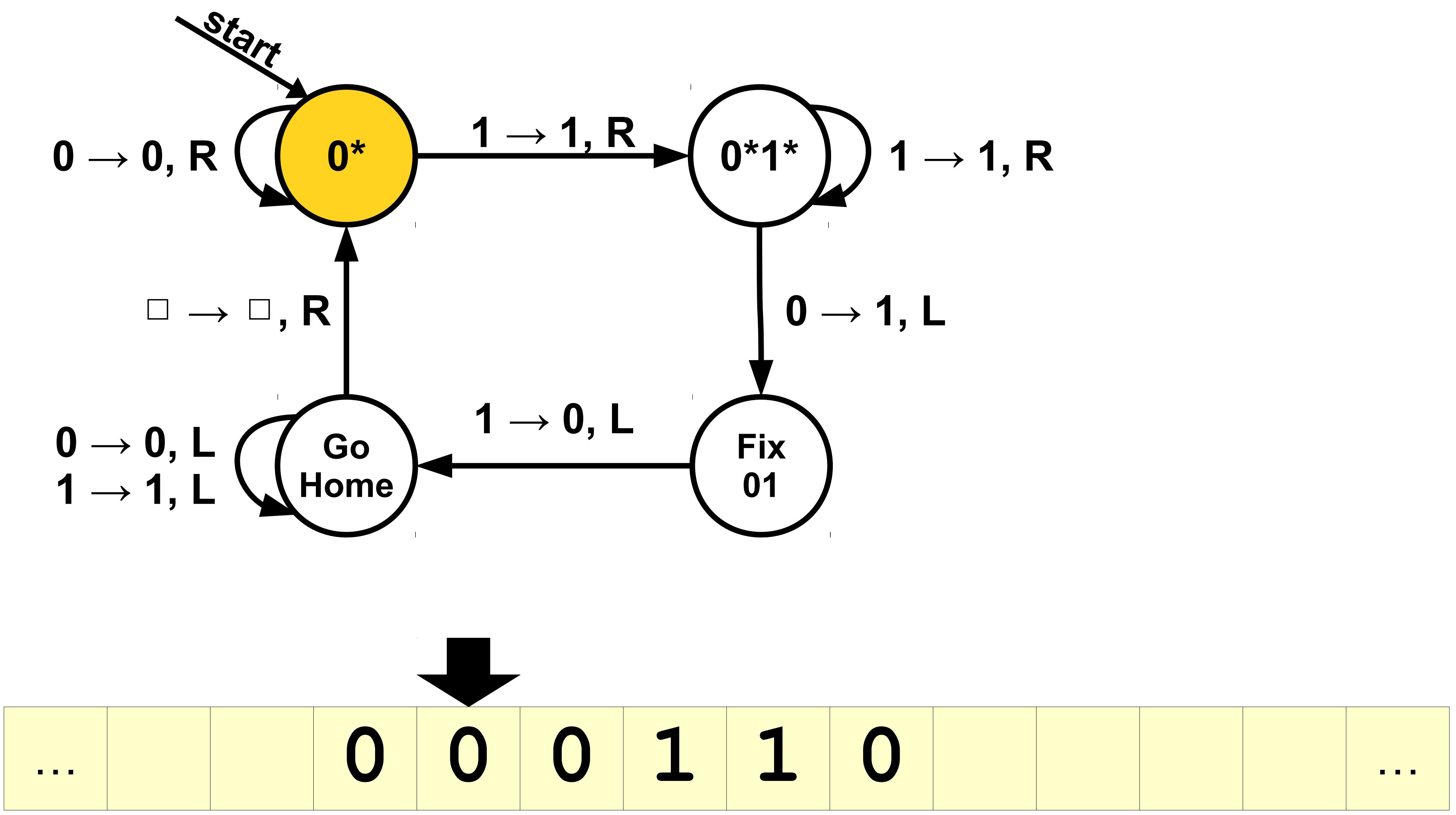

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 1 1 0 …

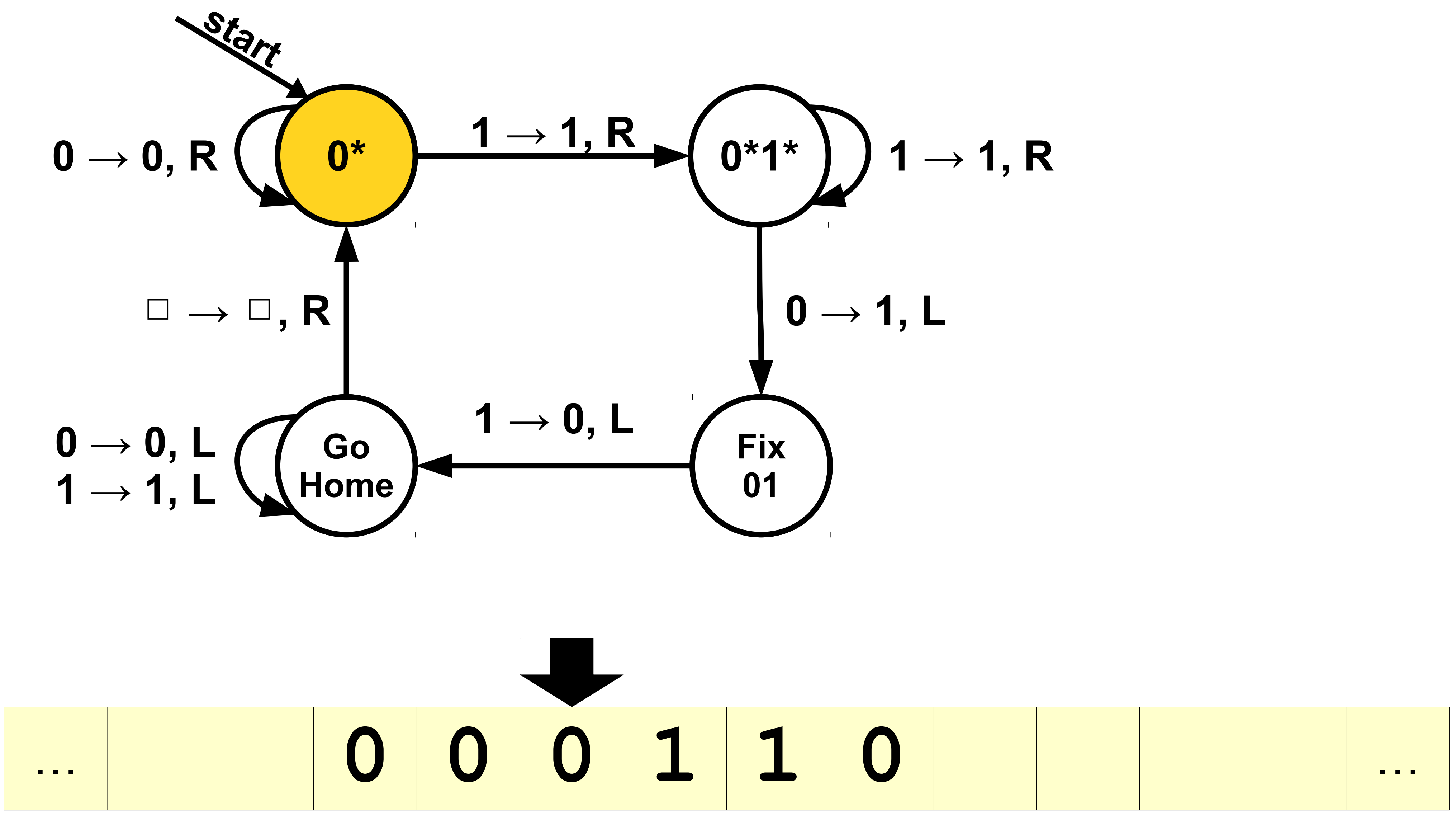

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 1 1 0 …

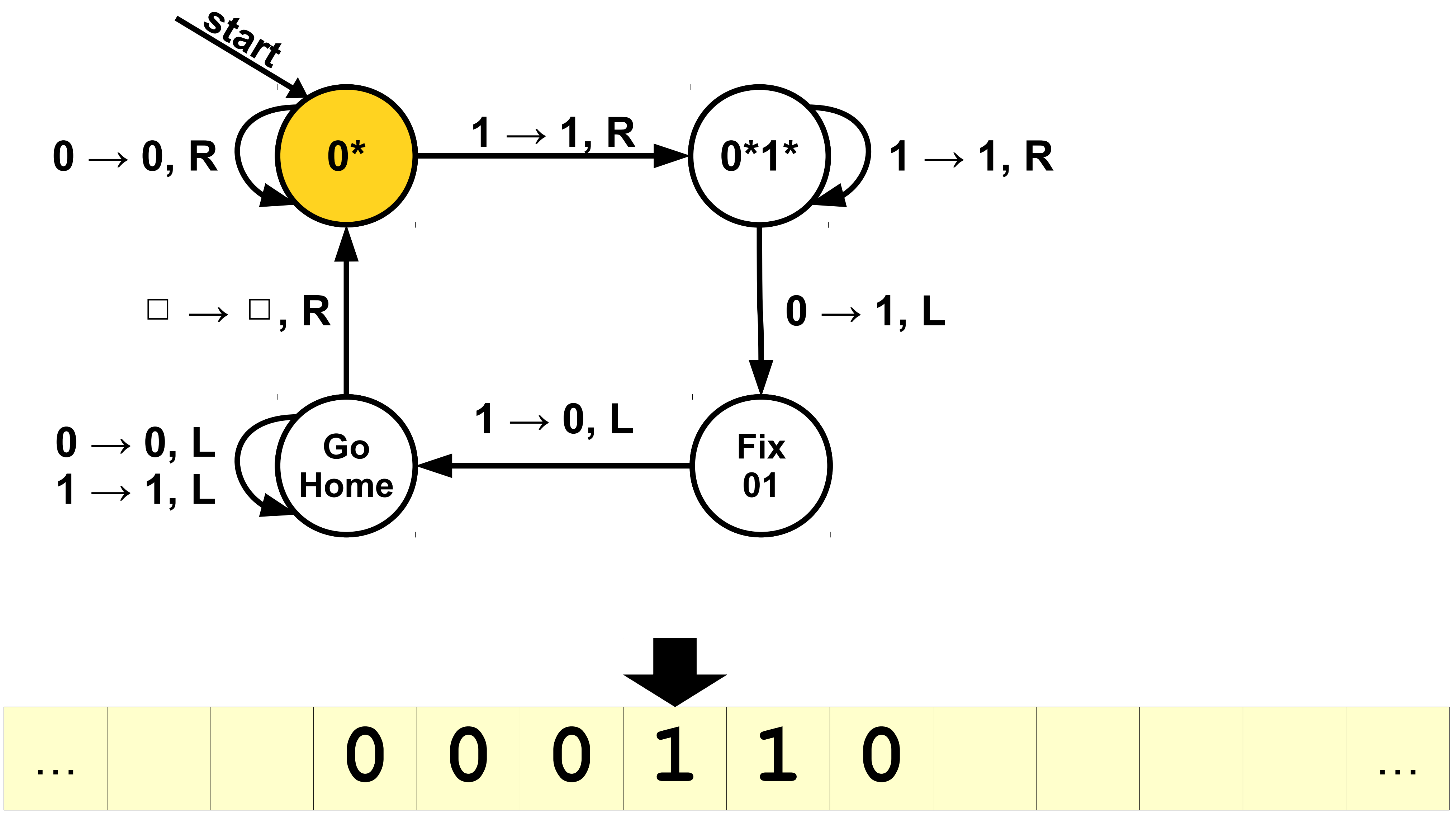

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0 0 0 1 1 0
…

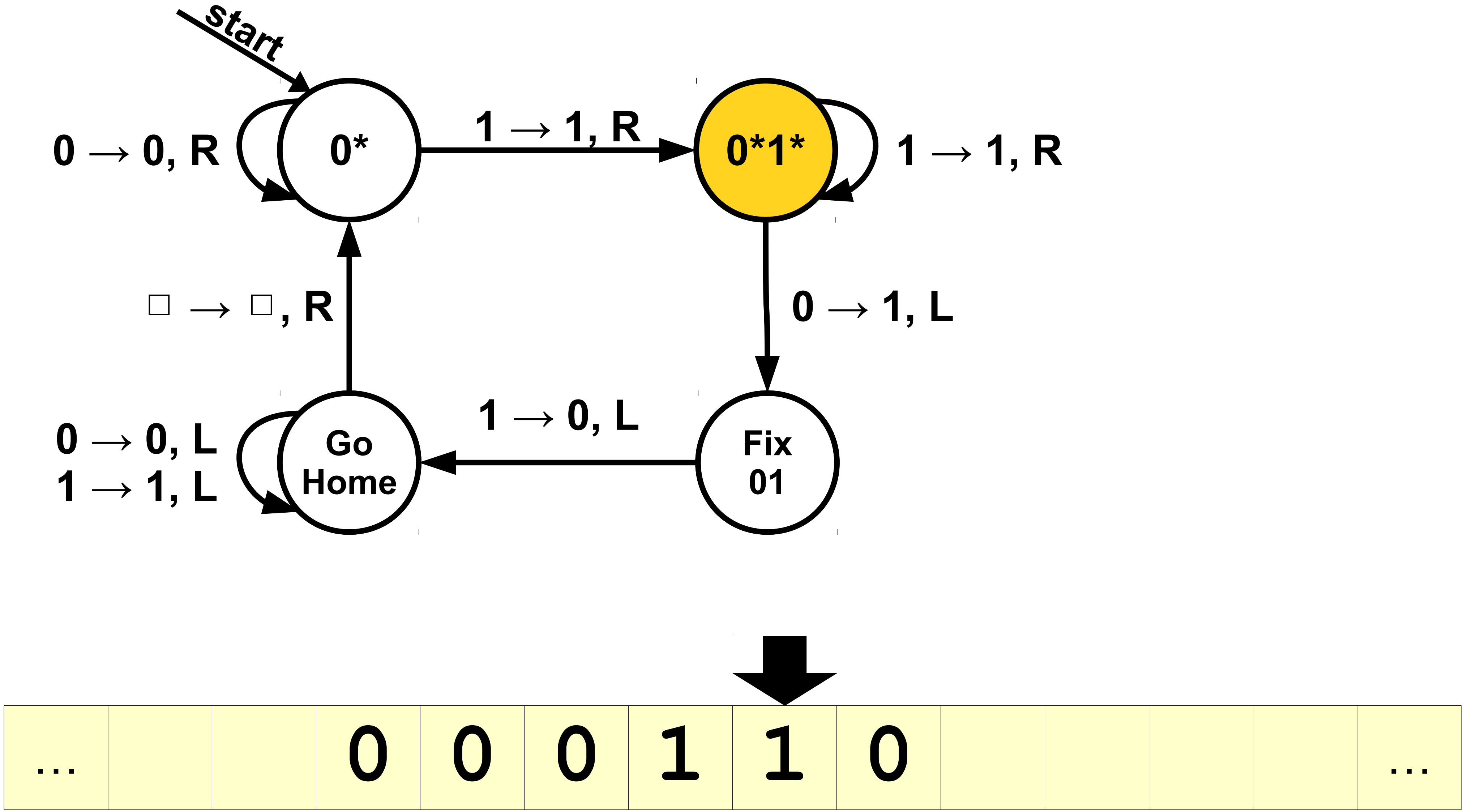

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
1
0
…


start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
...
0
0
0
1
1
0
...

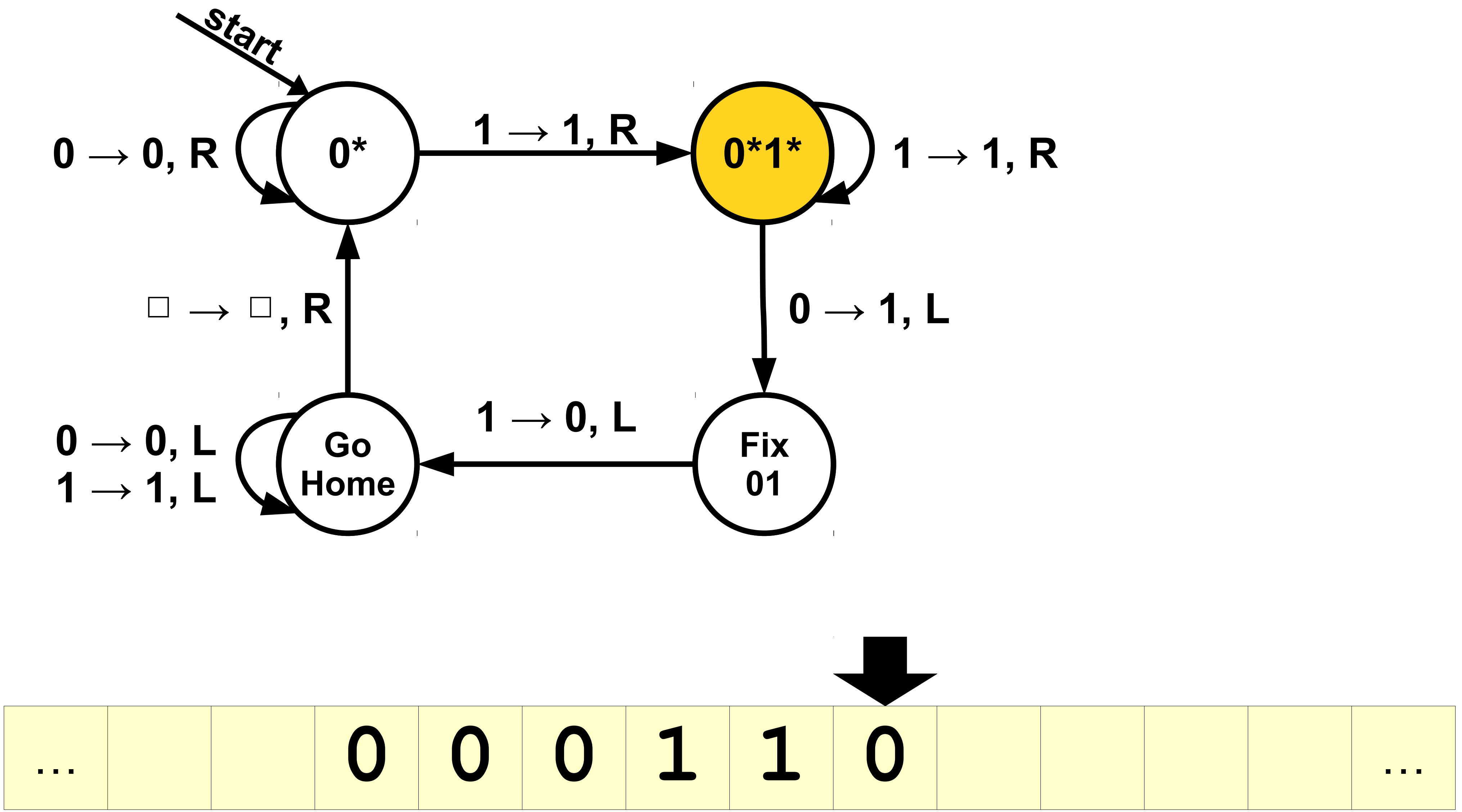

start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
1
1
0
…

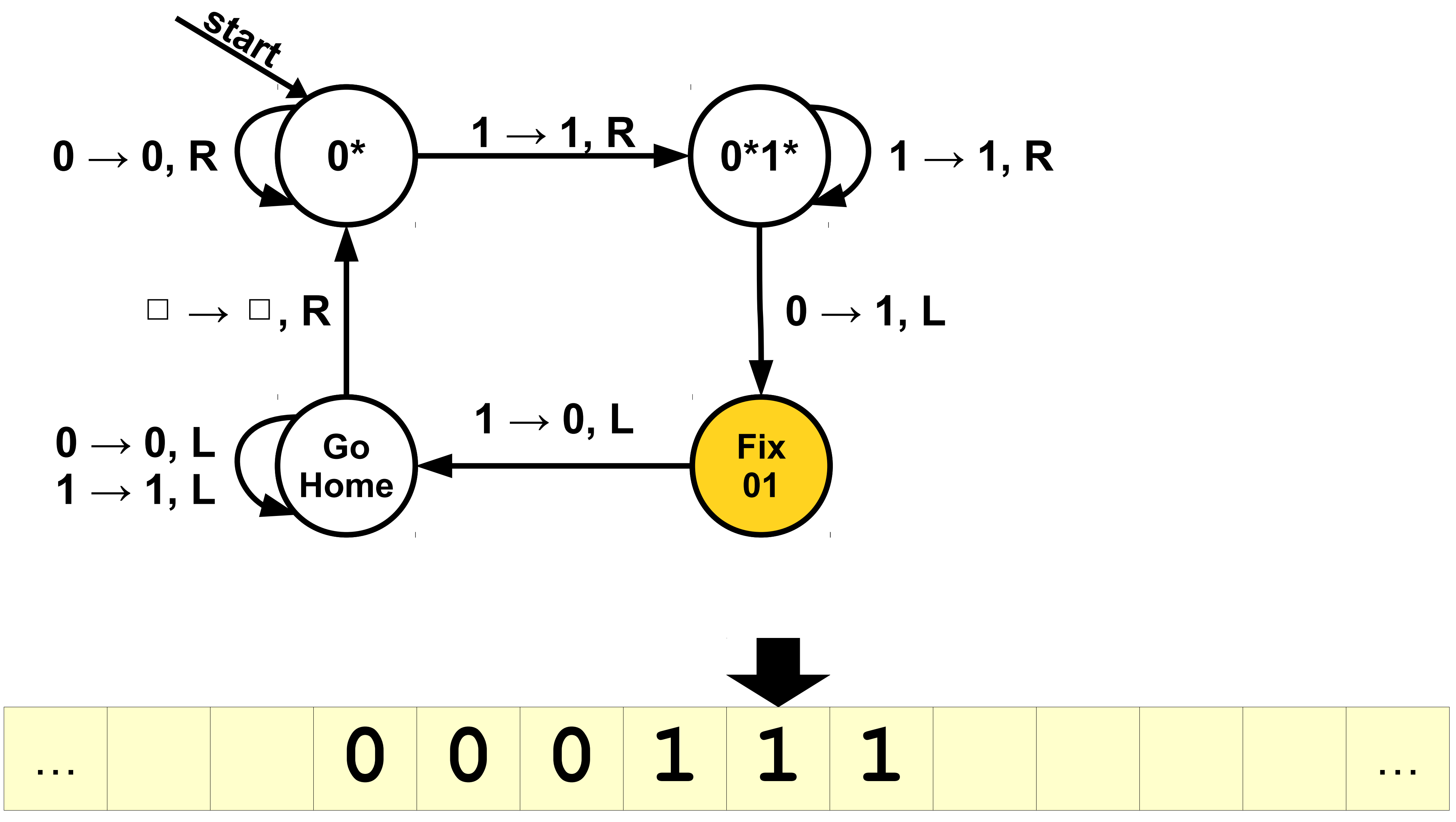

start
0*
0*1*
Go
Home
Fix
01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
1
0
…

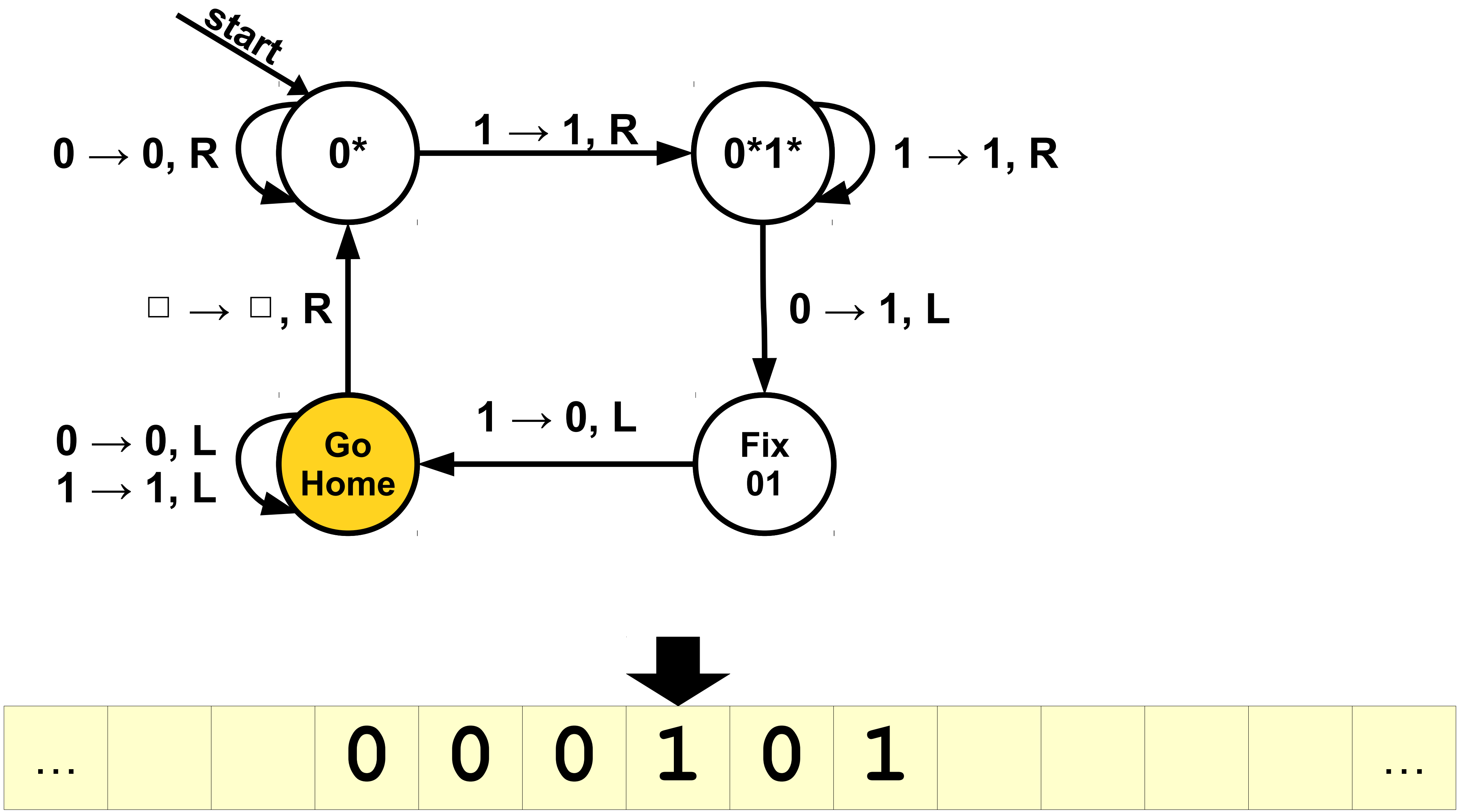

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 1 1 0 …



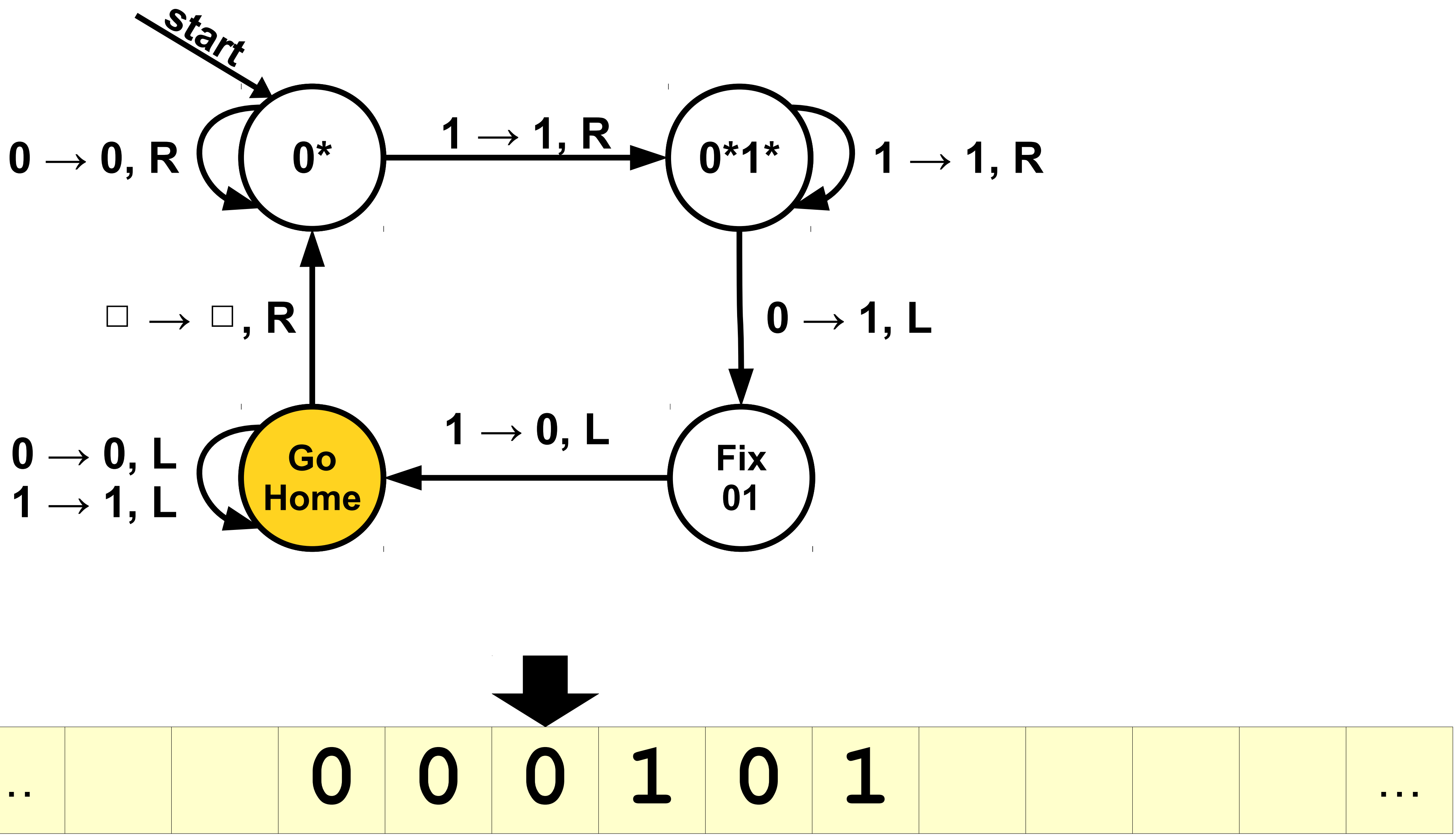
start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
☐ → ☐, R
…
0
0
0
1
1
0
…

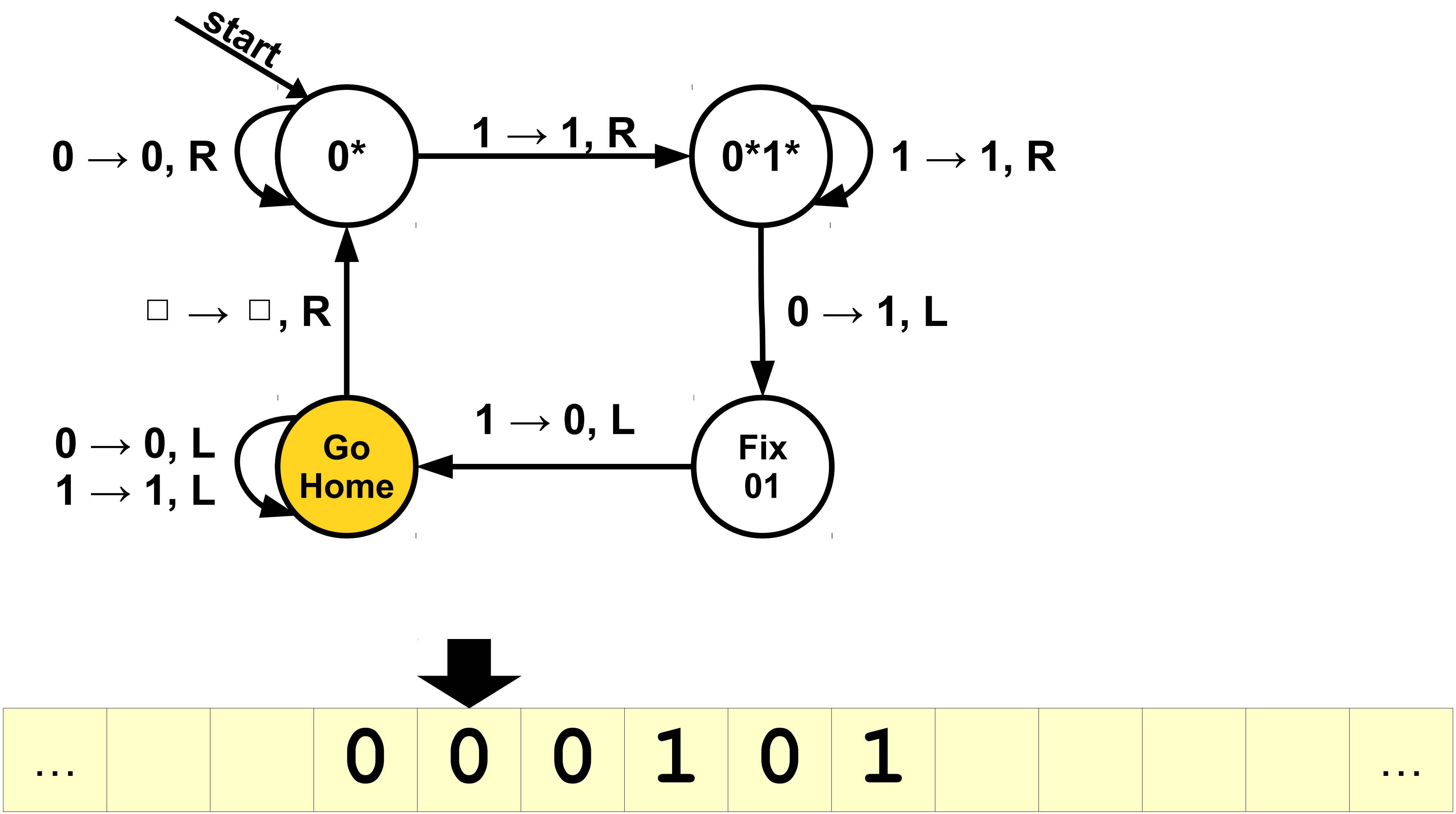

start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
1
0
…

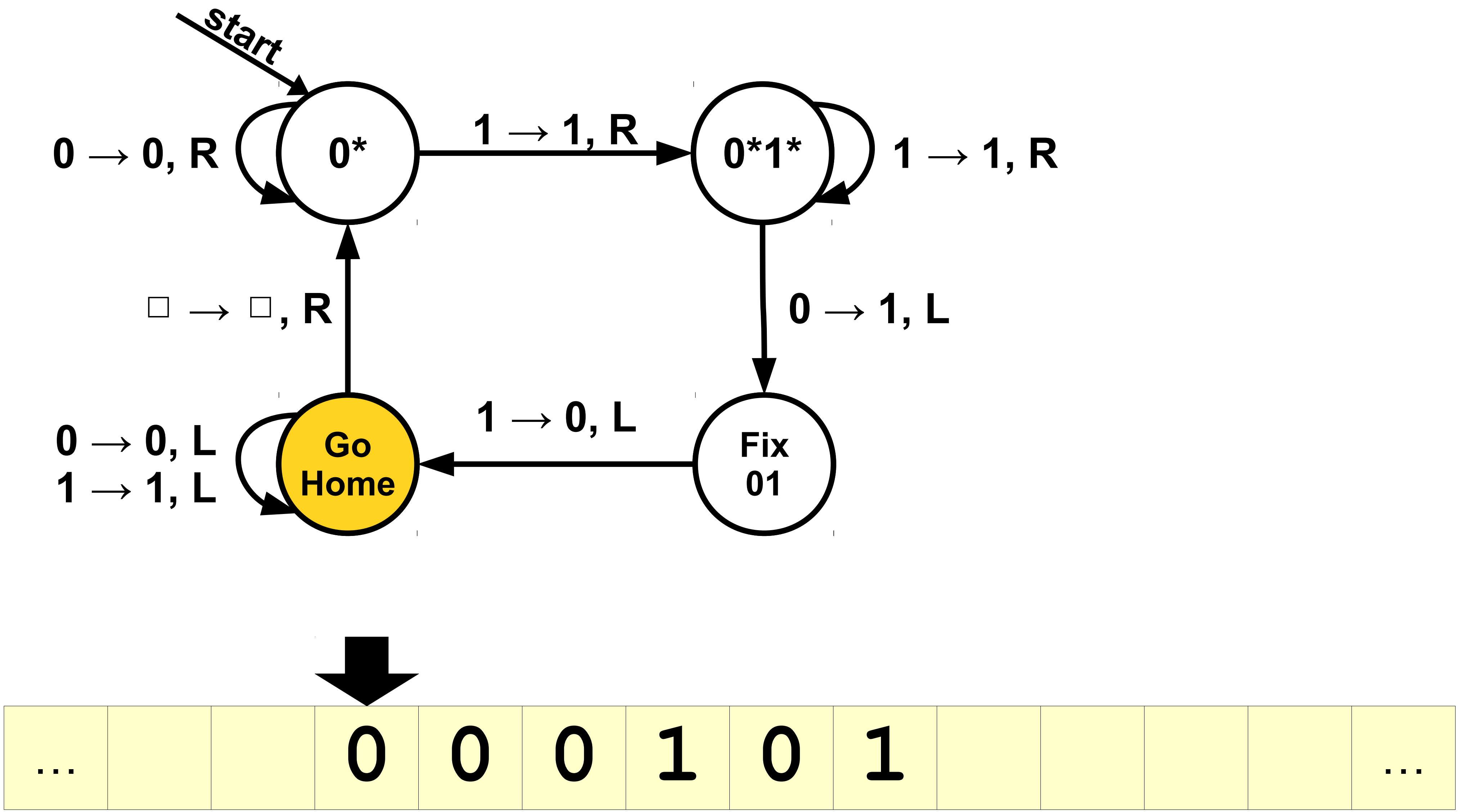

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
1
1
1
…

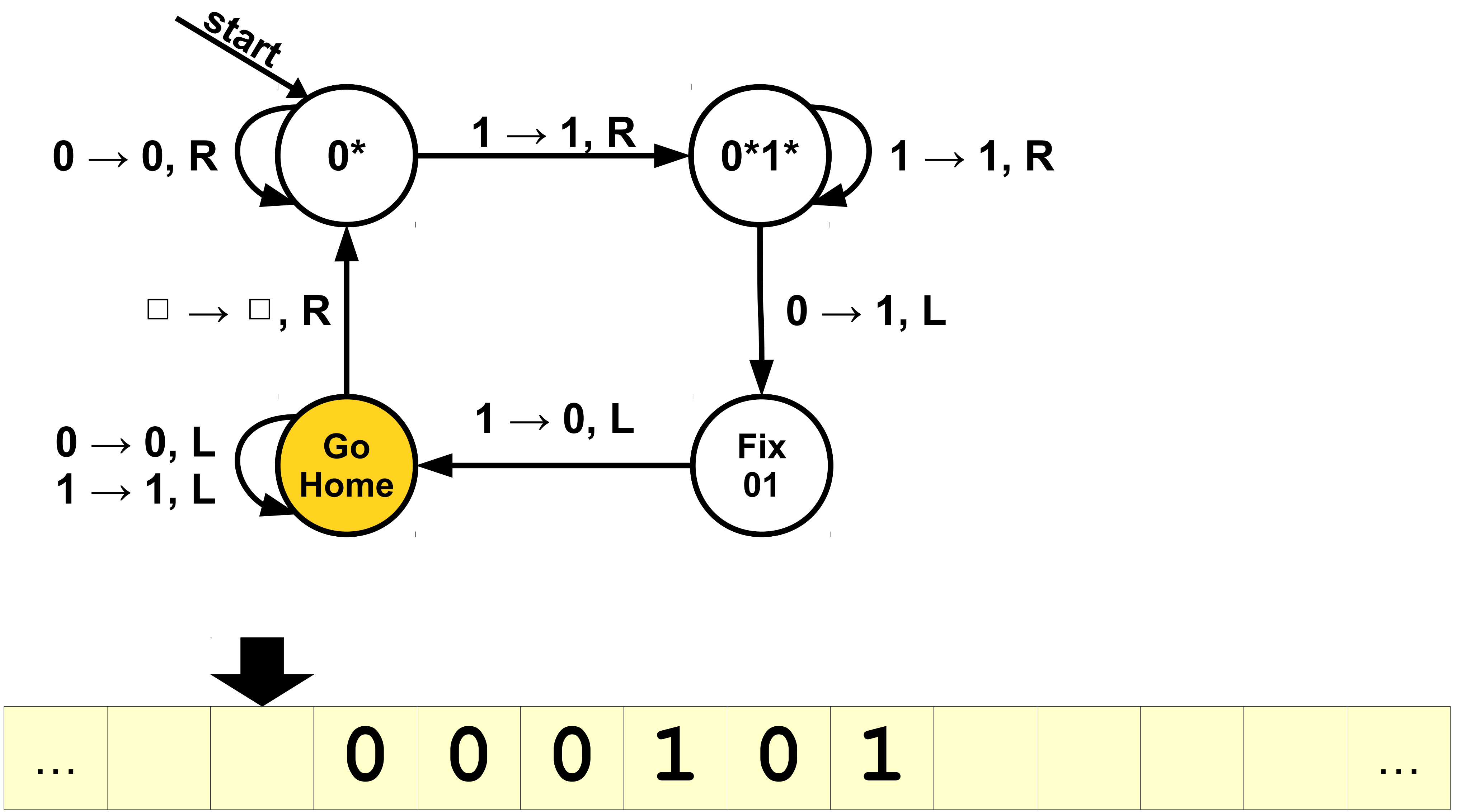


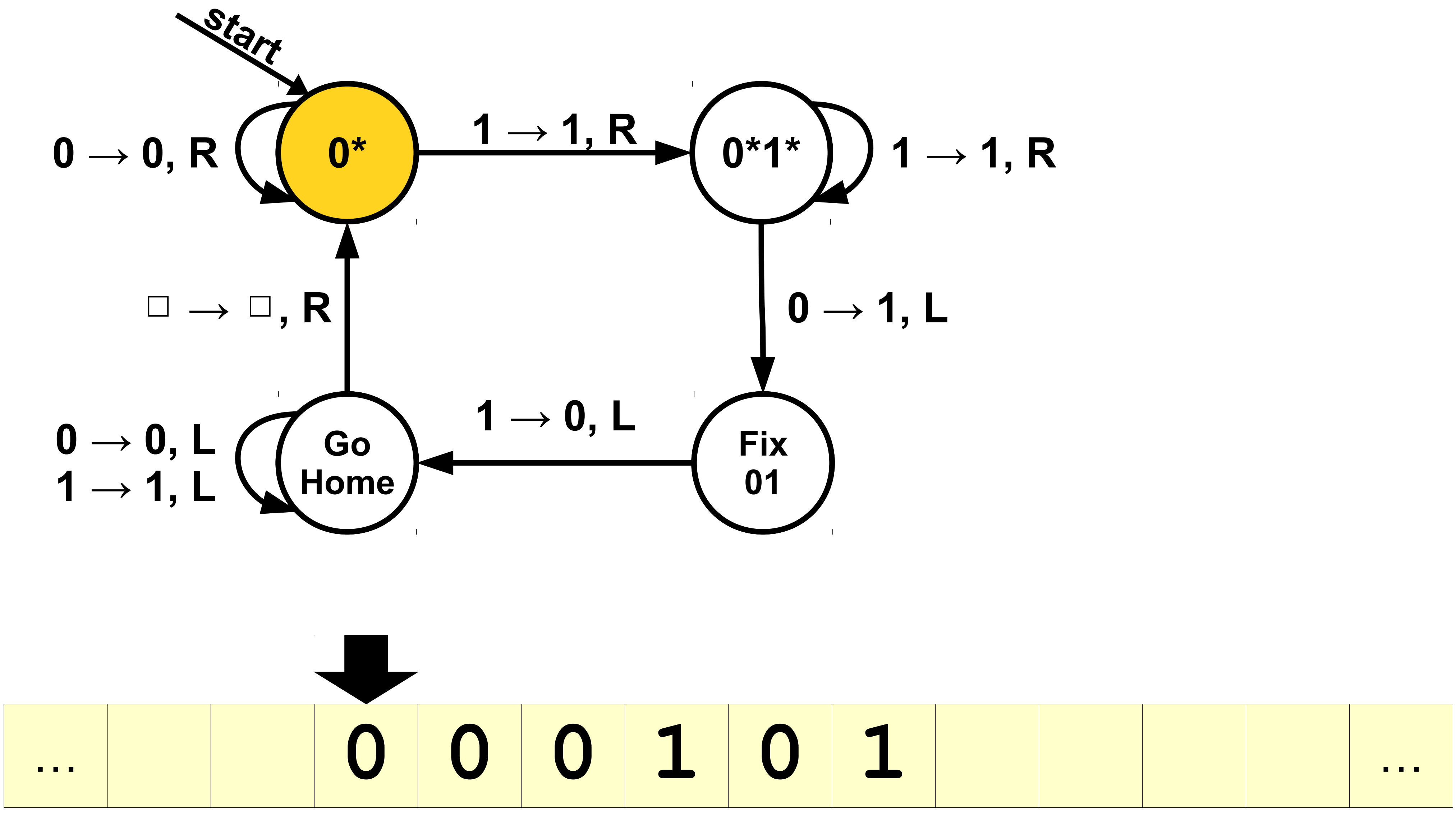
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
1
0
1
…


start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 1 0 1 …

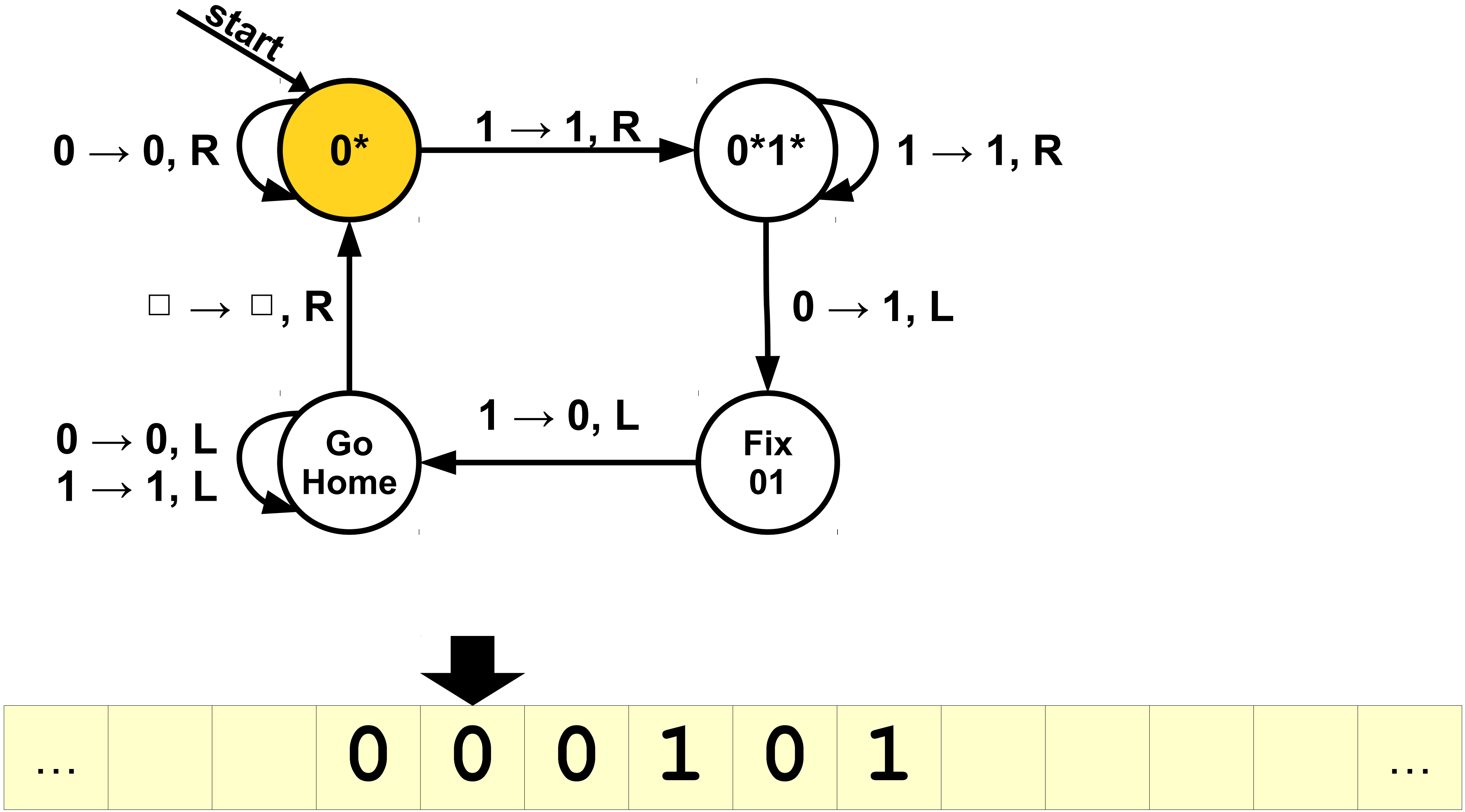

start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
… 0 0 0 1 0 1 …


start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 1 0 1 …

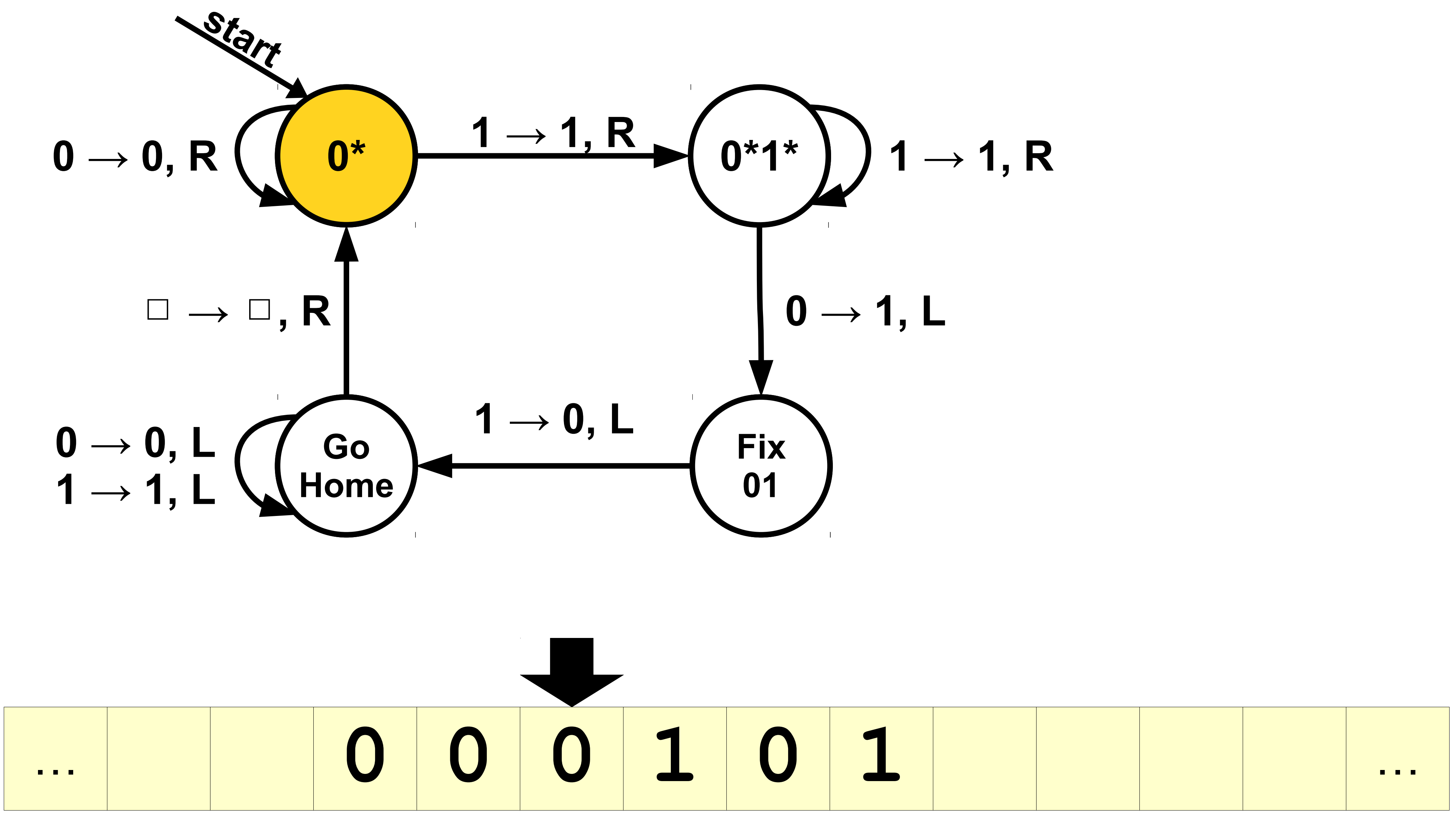

start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
...
0 0 0 1 0 1
...

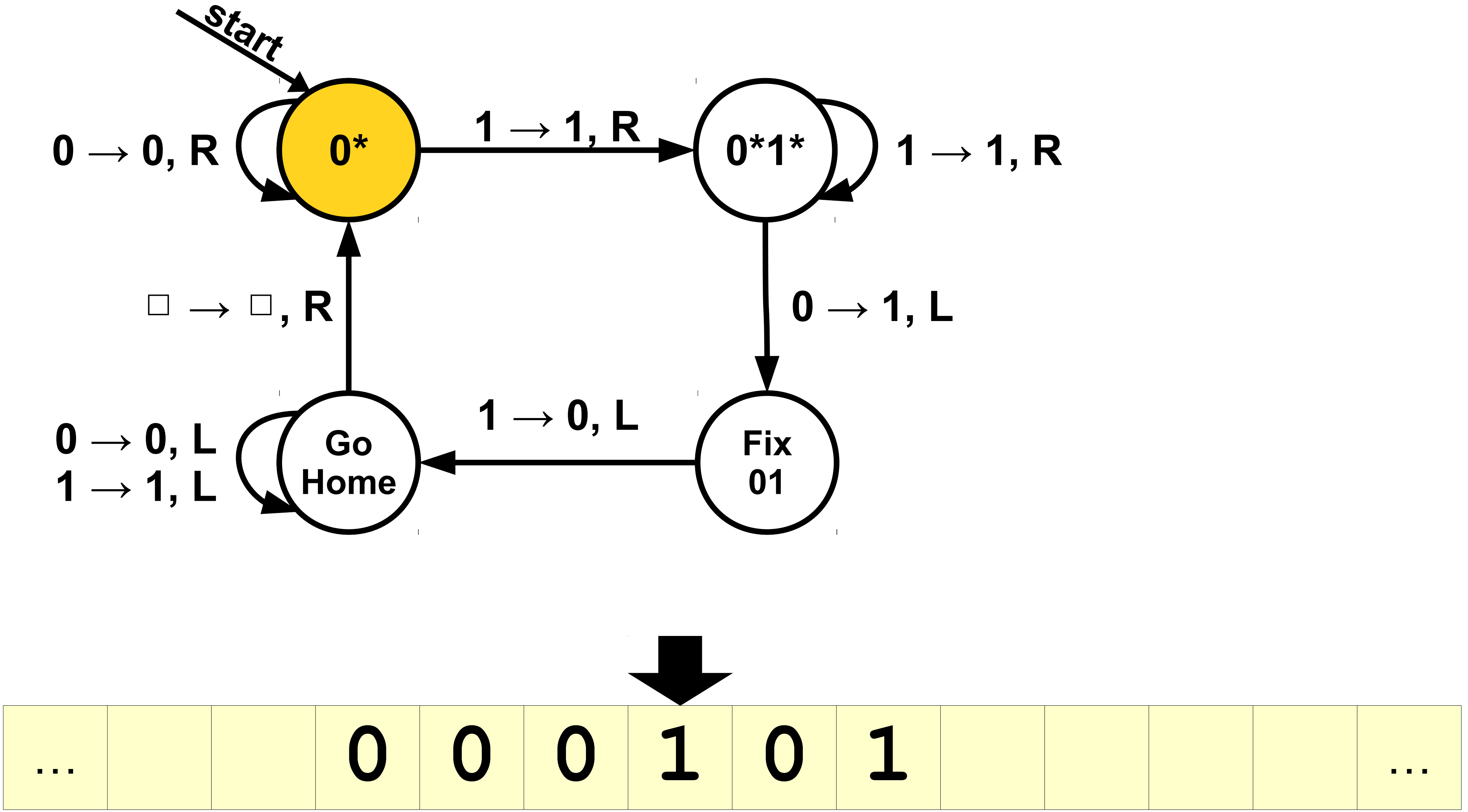


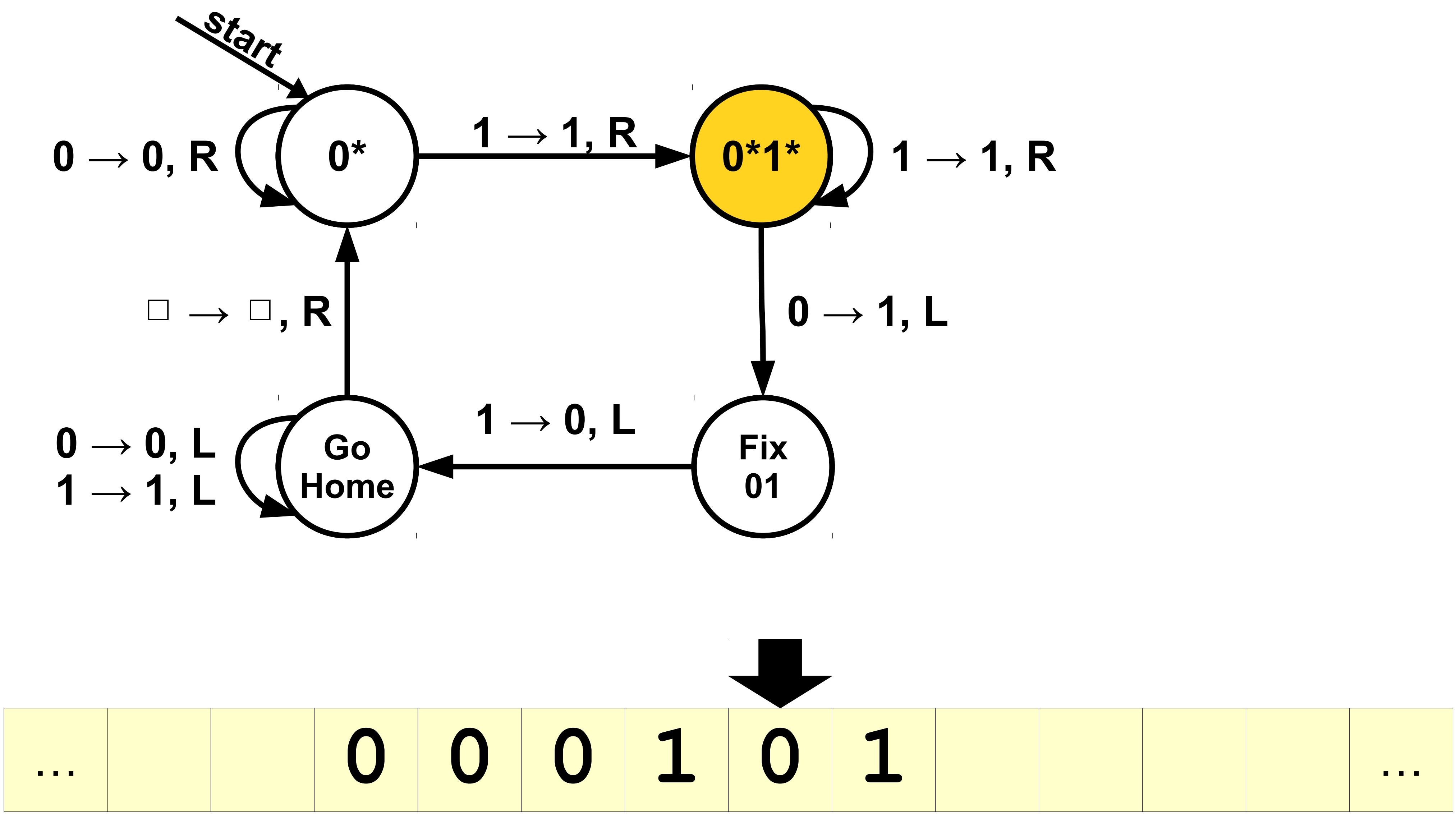
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
1
0
1
…

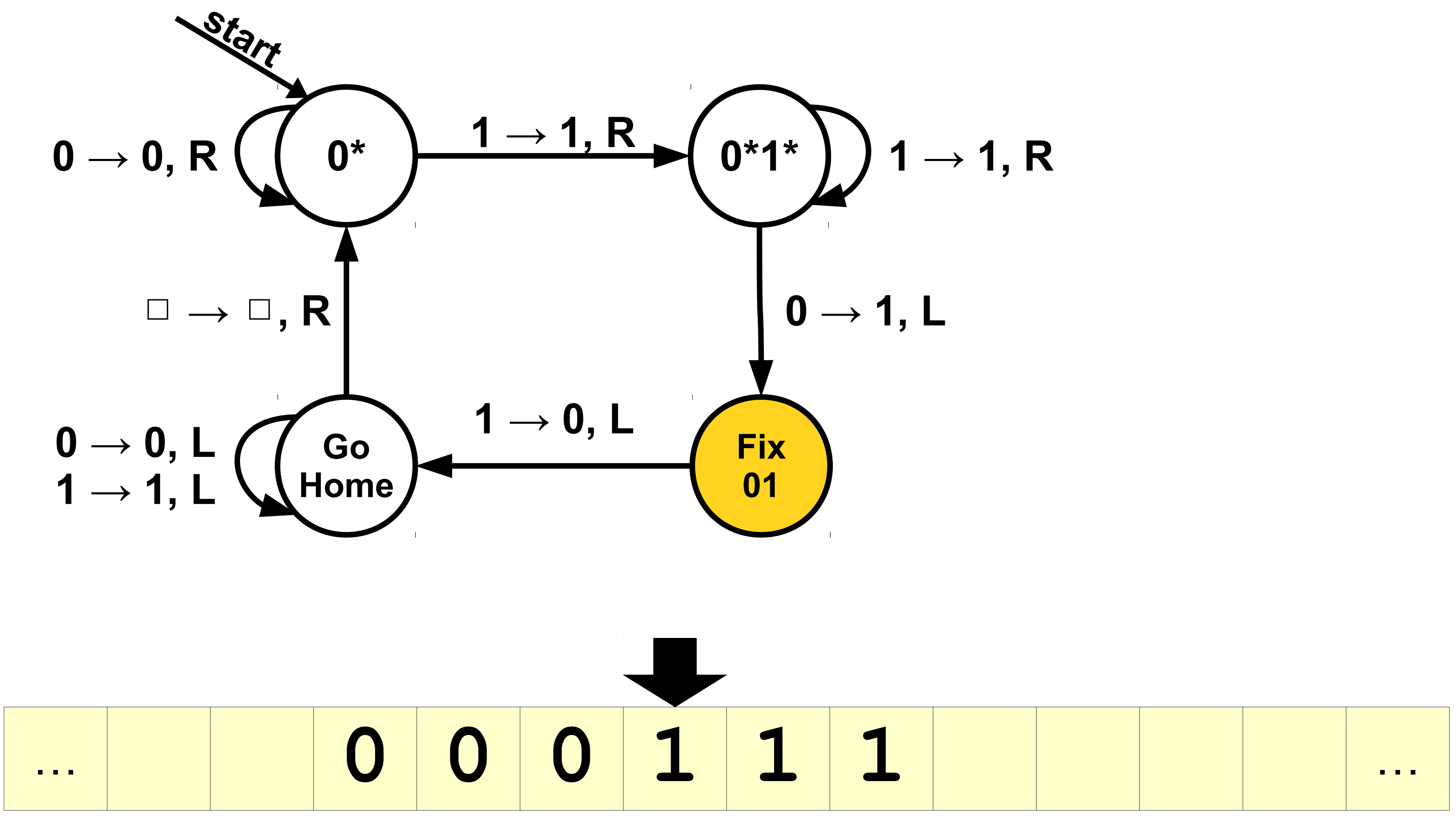


start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
0
1
…

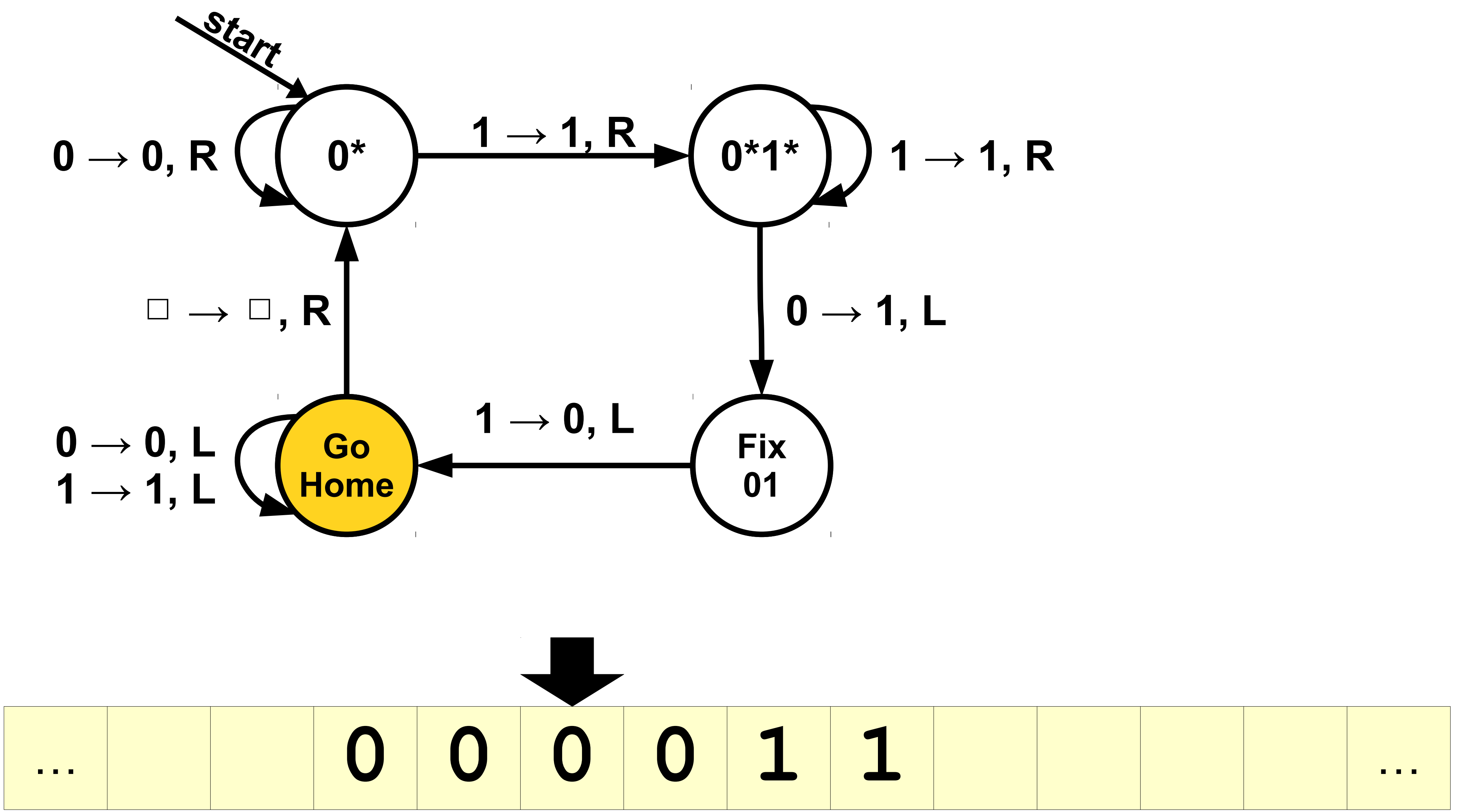

start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
0
1
…

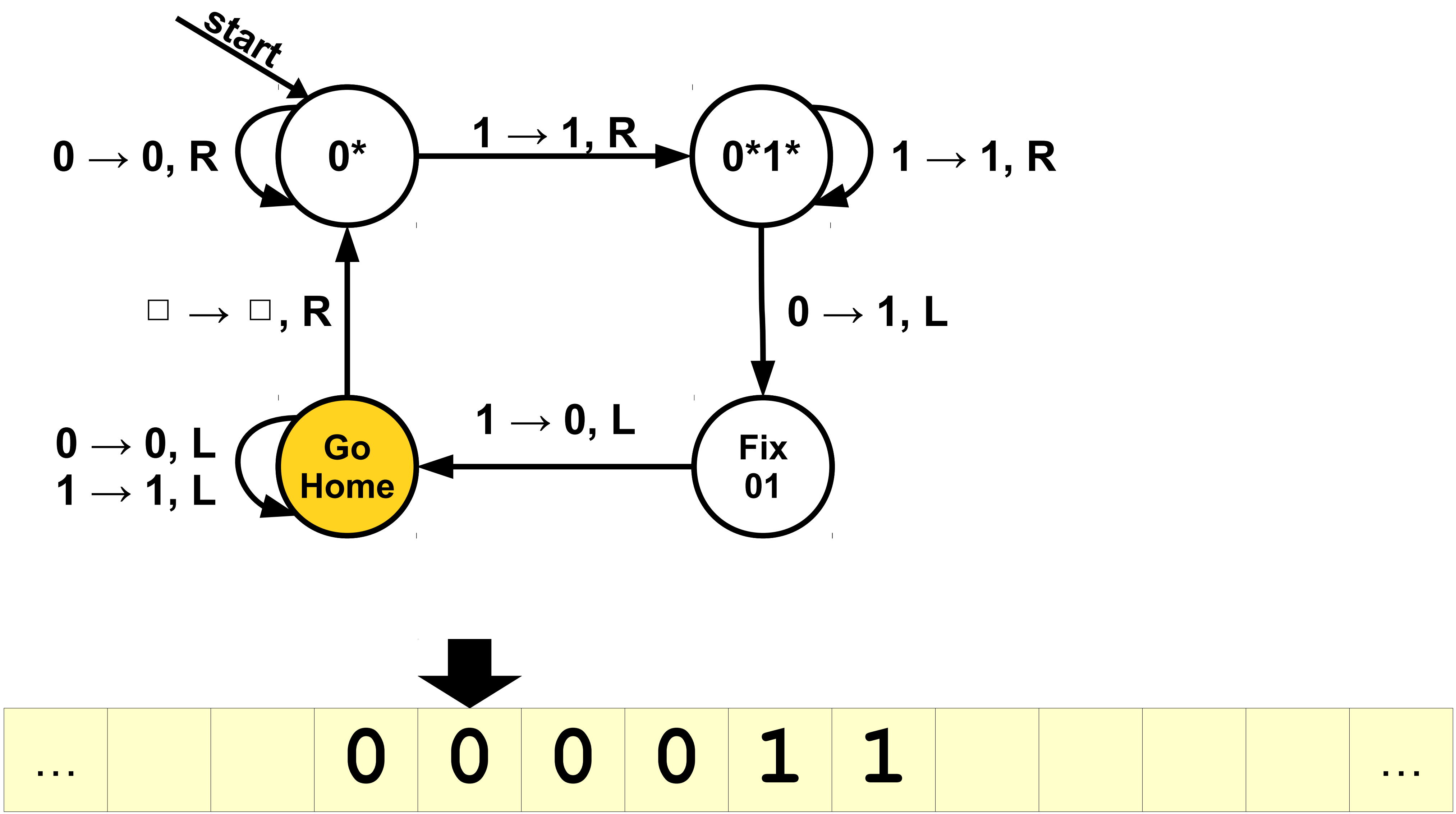

start
0*
0*1*
Fix
01
Go
Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
1
0
1
…


start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
1
0
1
…

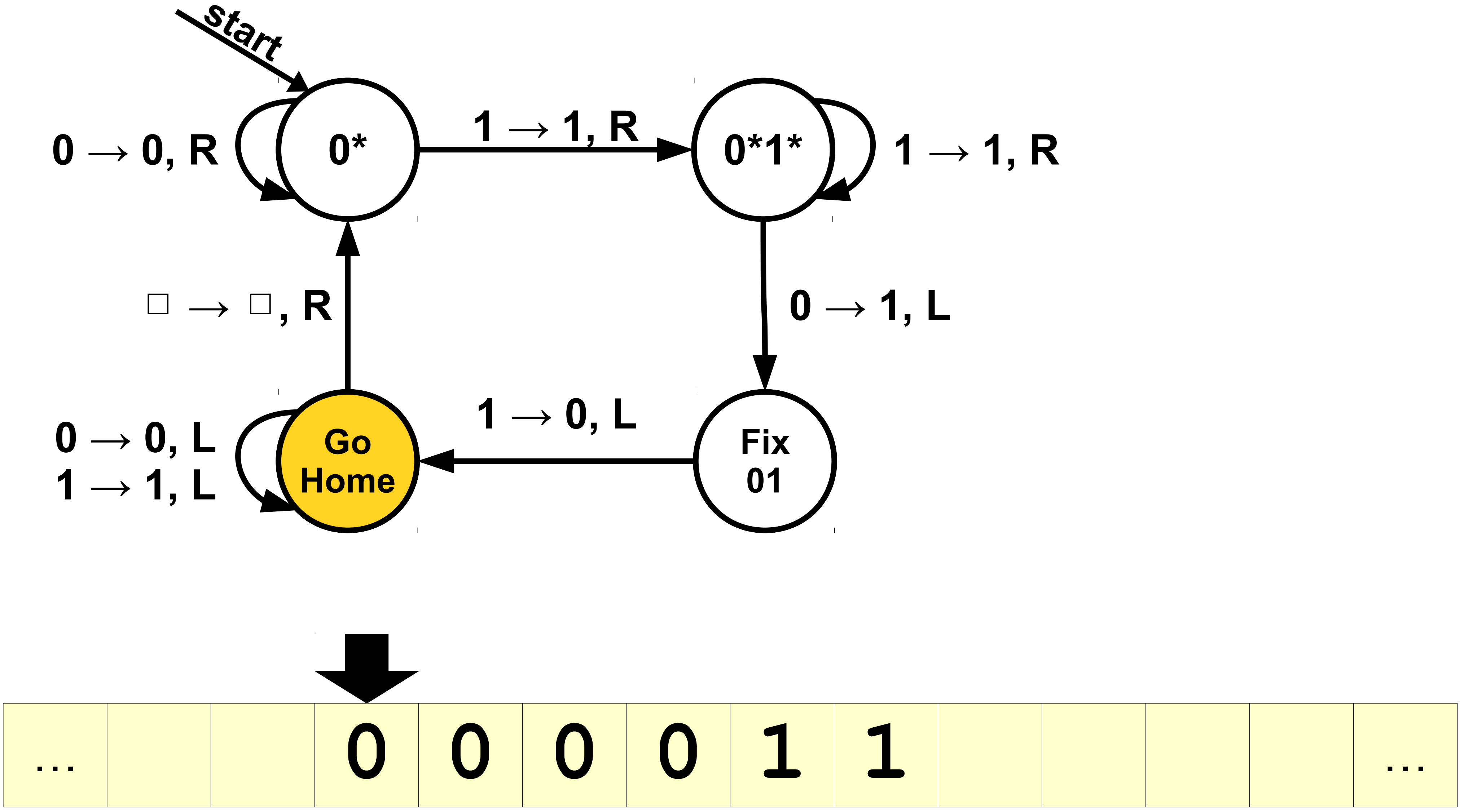


start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
1
1
1
…

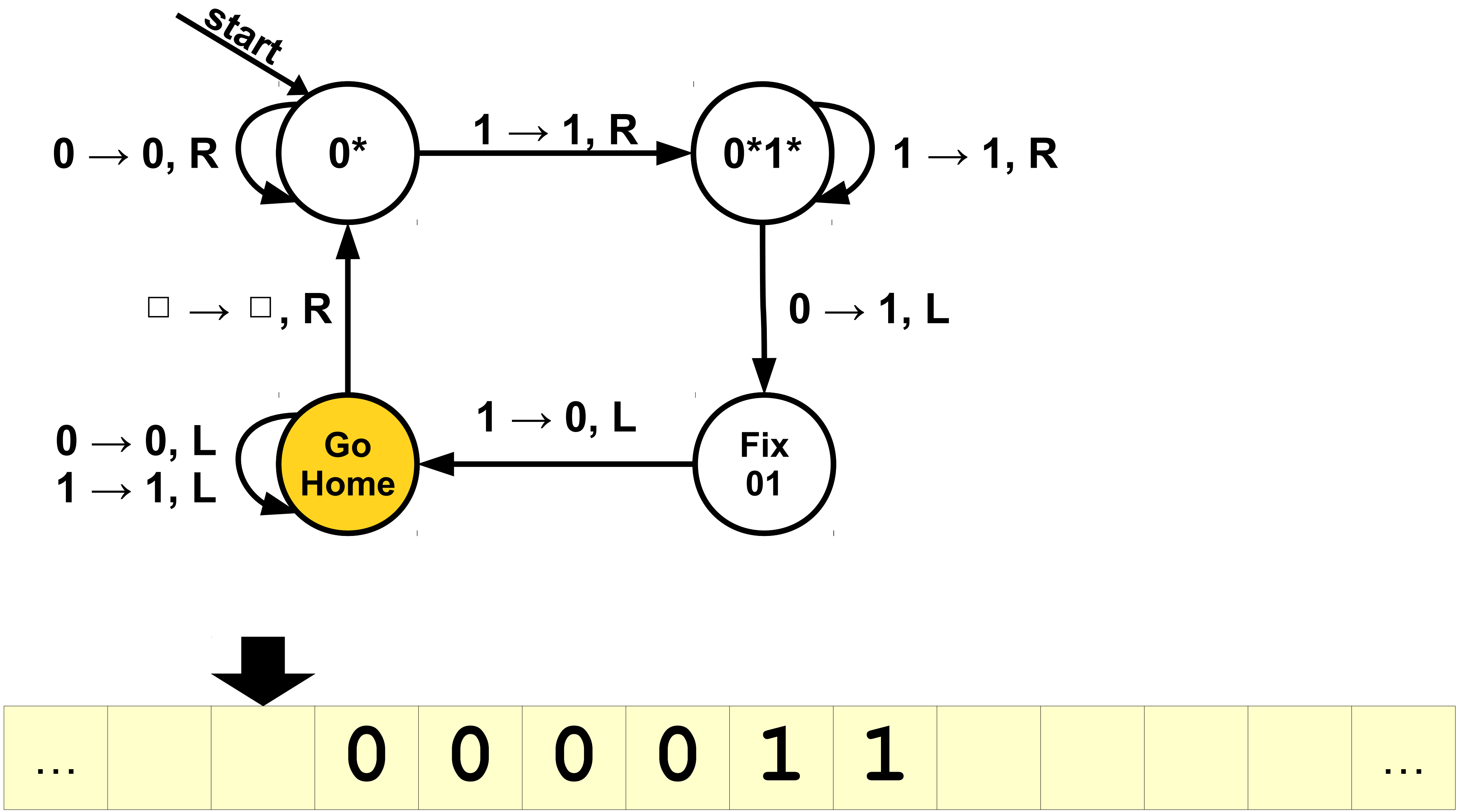

start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
...
0 0 0 0 1 1
...



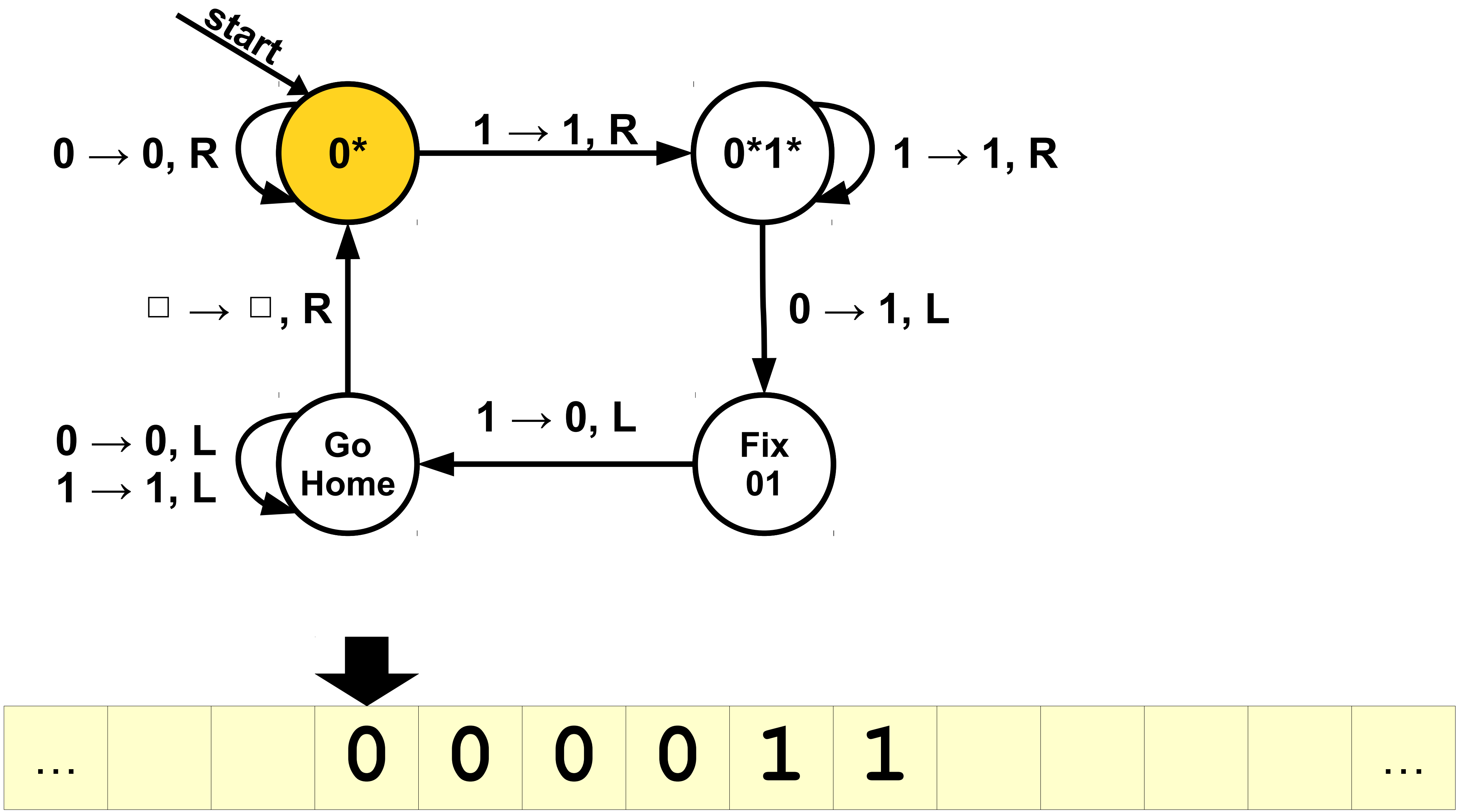
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

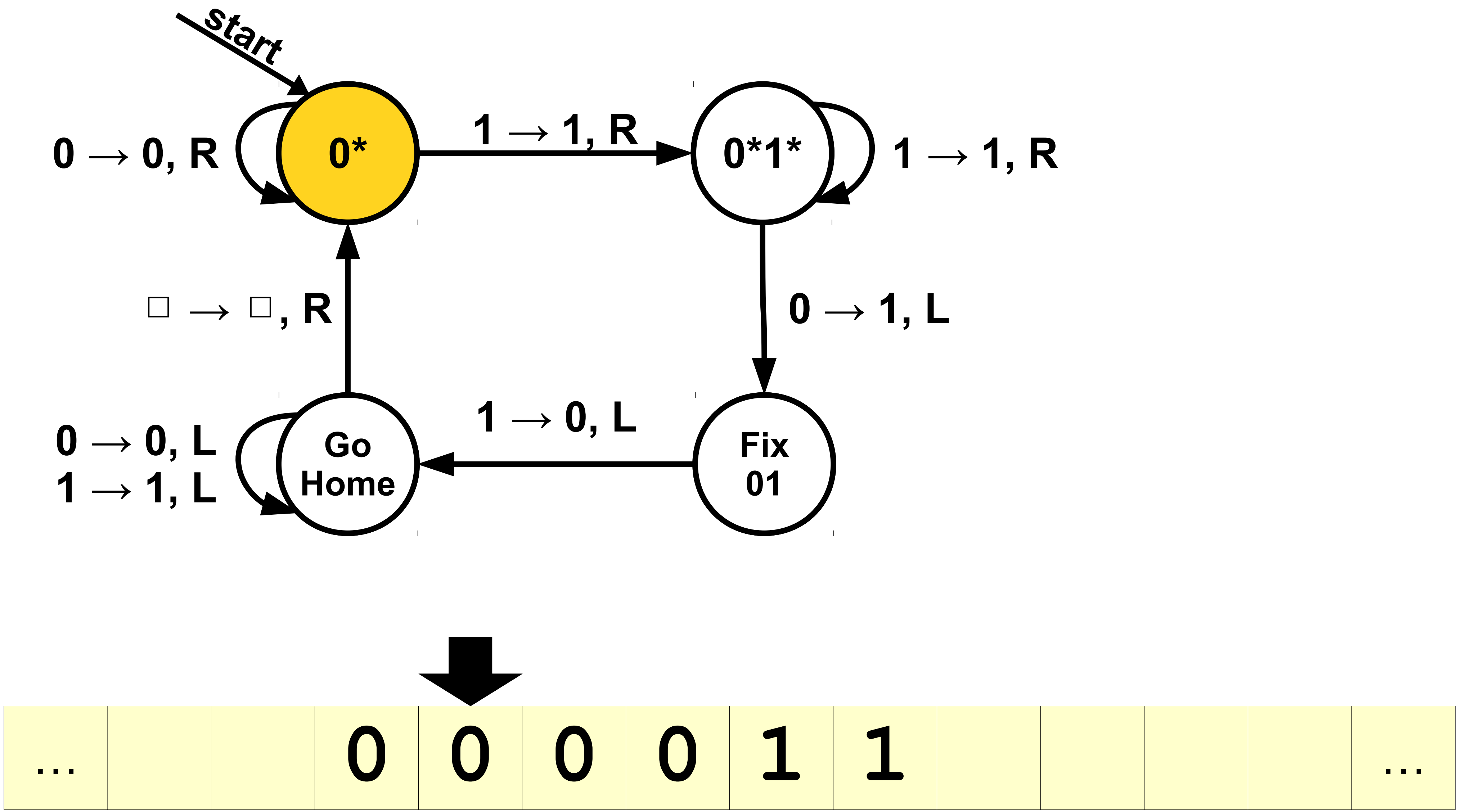


start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 0 1 1 …

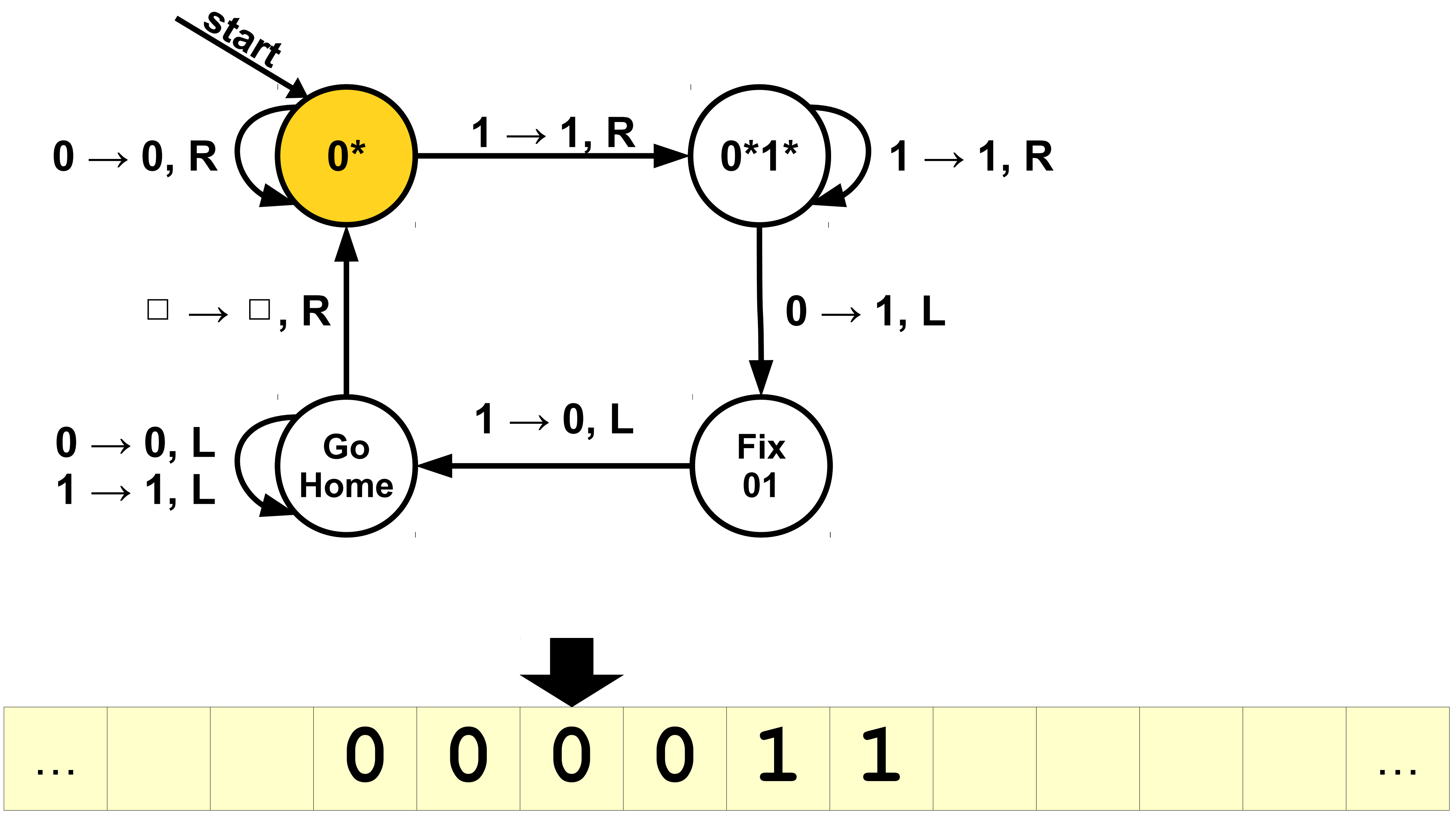


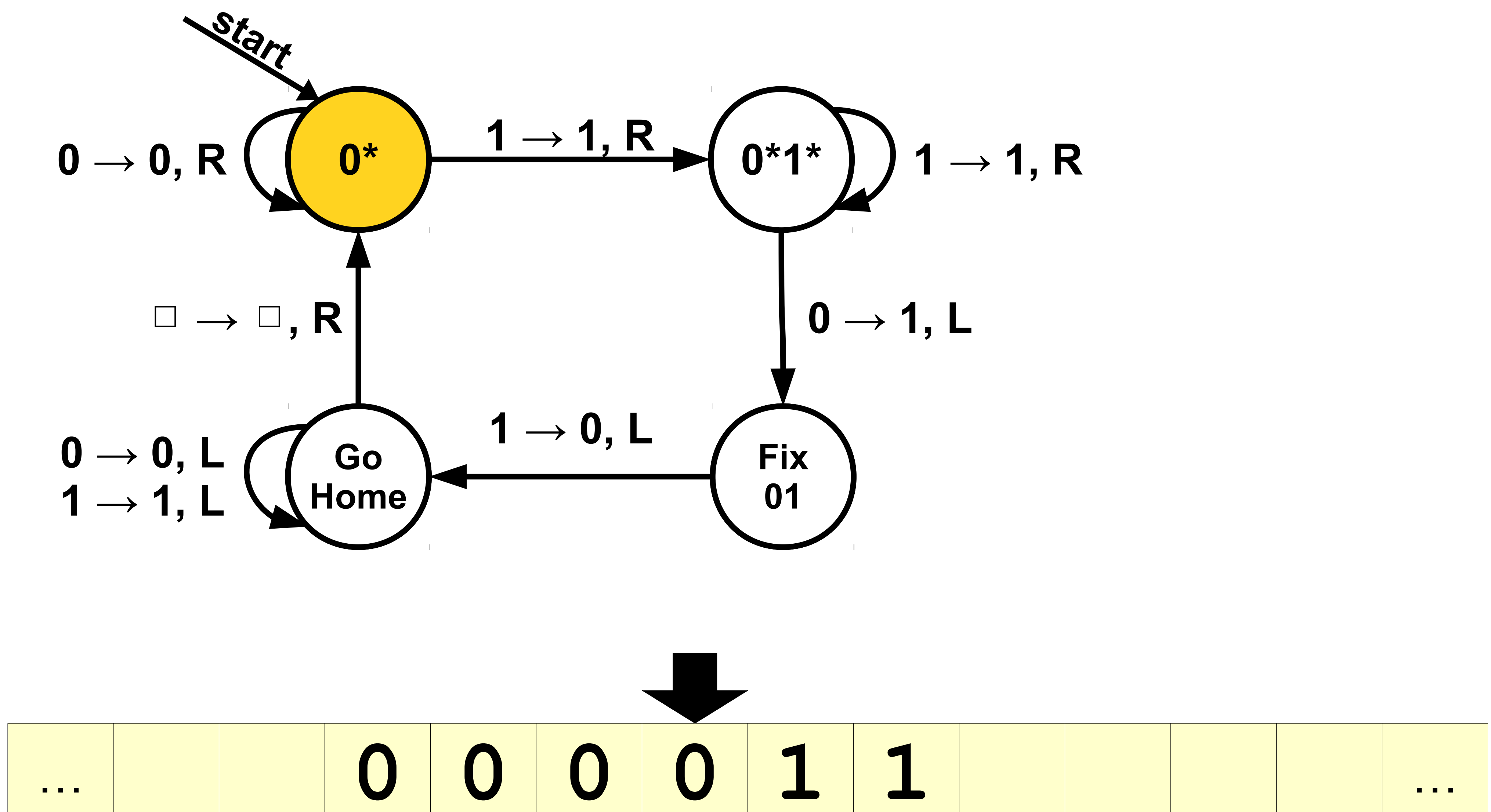
start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
0
1
1
…

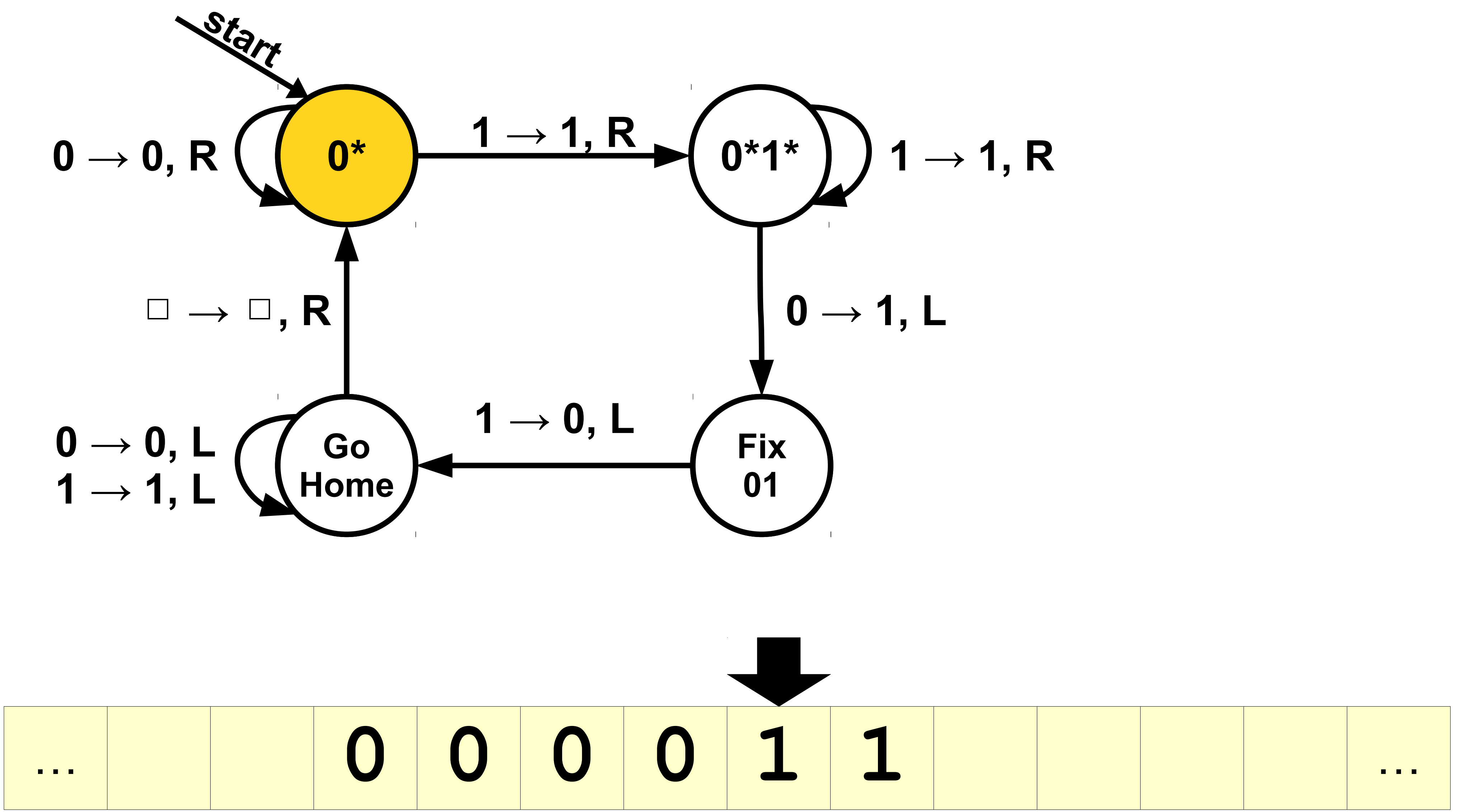

start
0*
0*1*
Go Home
Fix 01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0 0 0 0 1 1
…

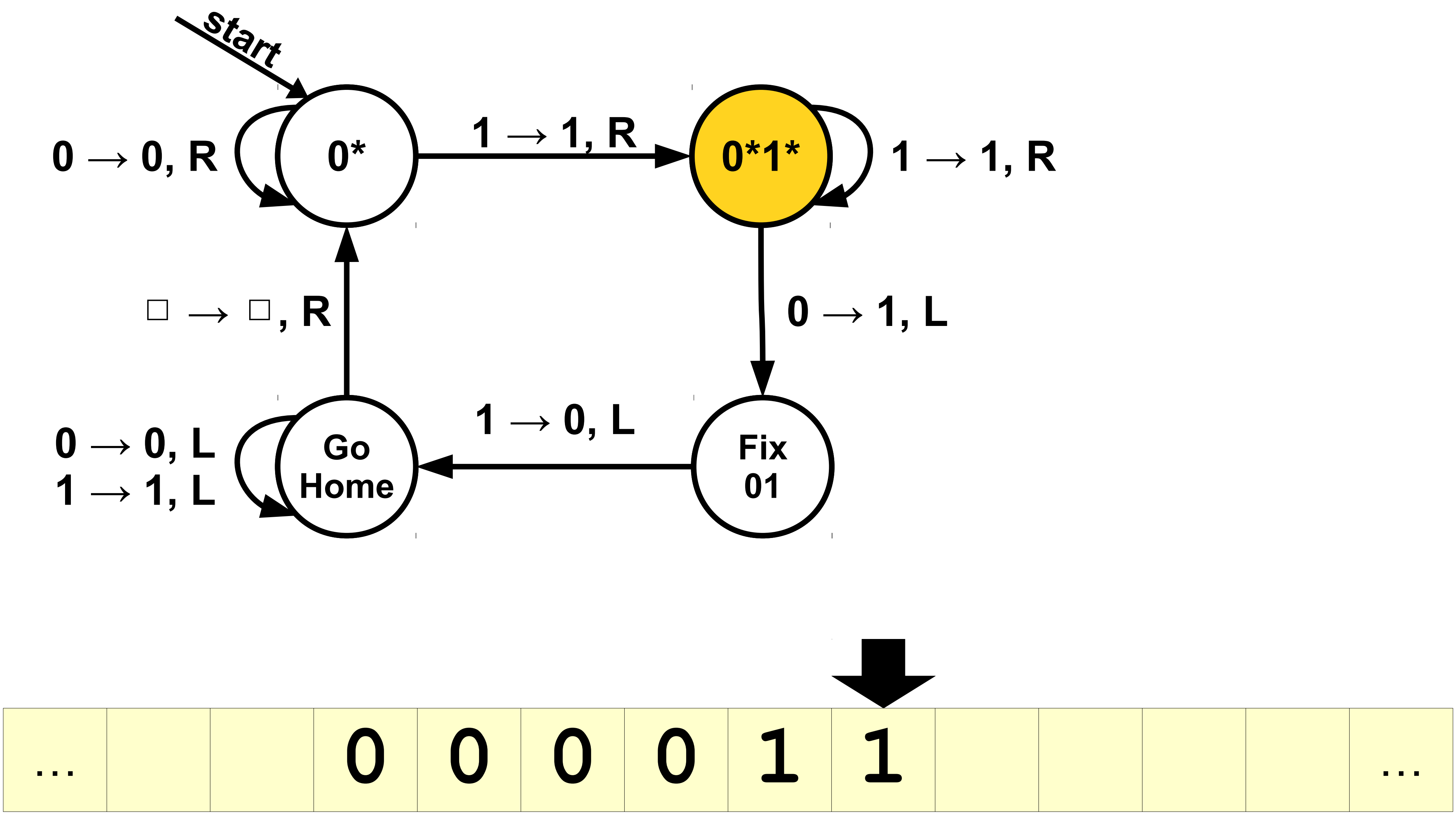


start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
… 0 0 0 0 1 1 …

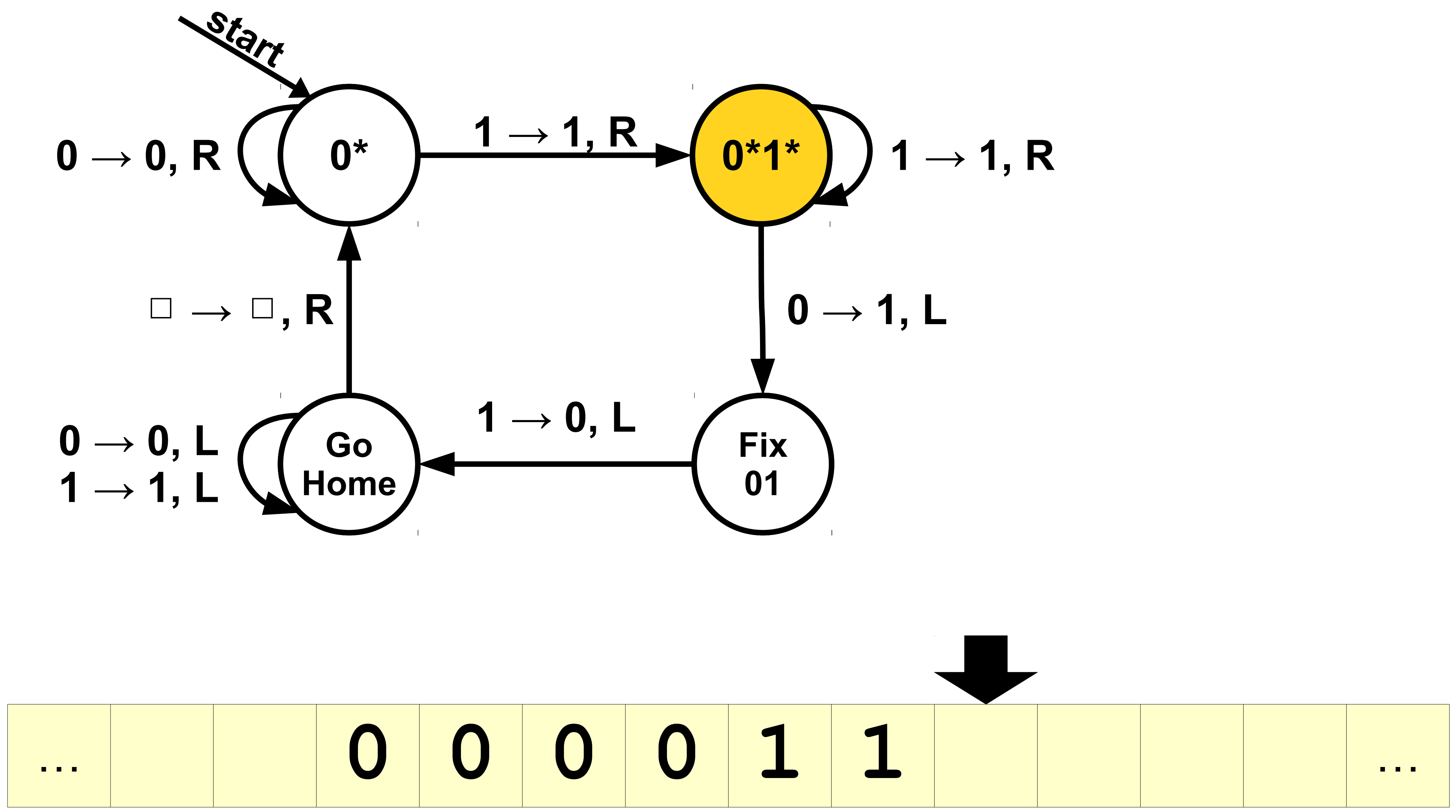

start
0*
0*1*
Go
Home
Fix
01
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
0
1
1
…


start
0*
0*1*
Fix 01
Go Home
0 → 0, R
1 → 1, R
1 → 1, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
□ → □, R
…
0
0
0
0
1
1
…


start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

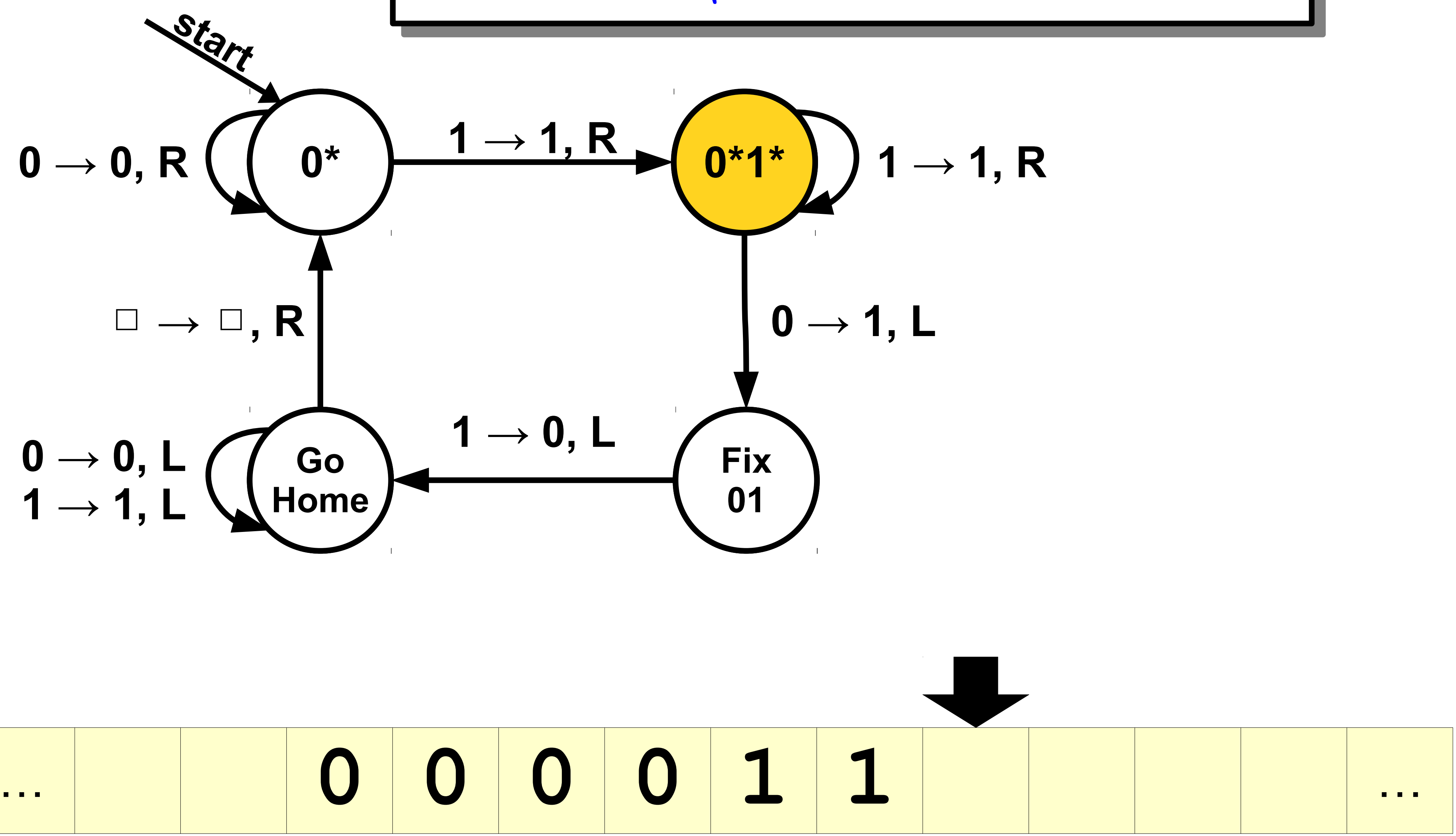


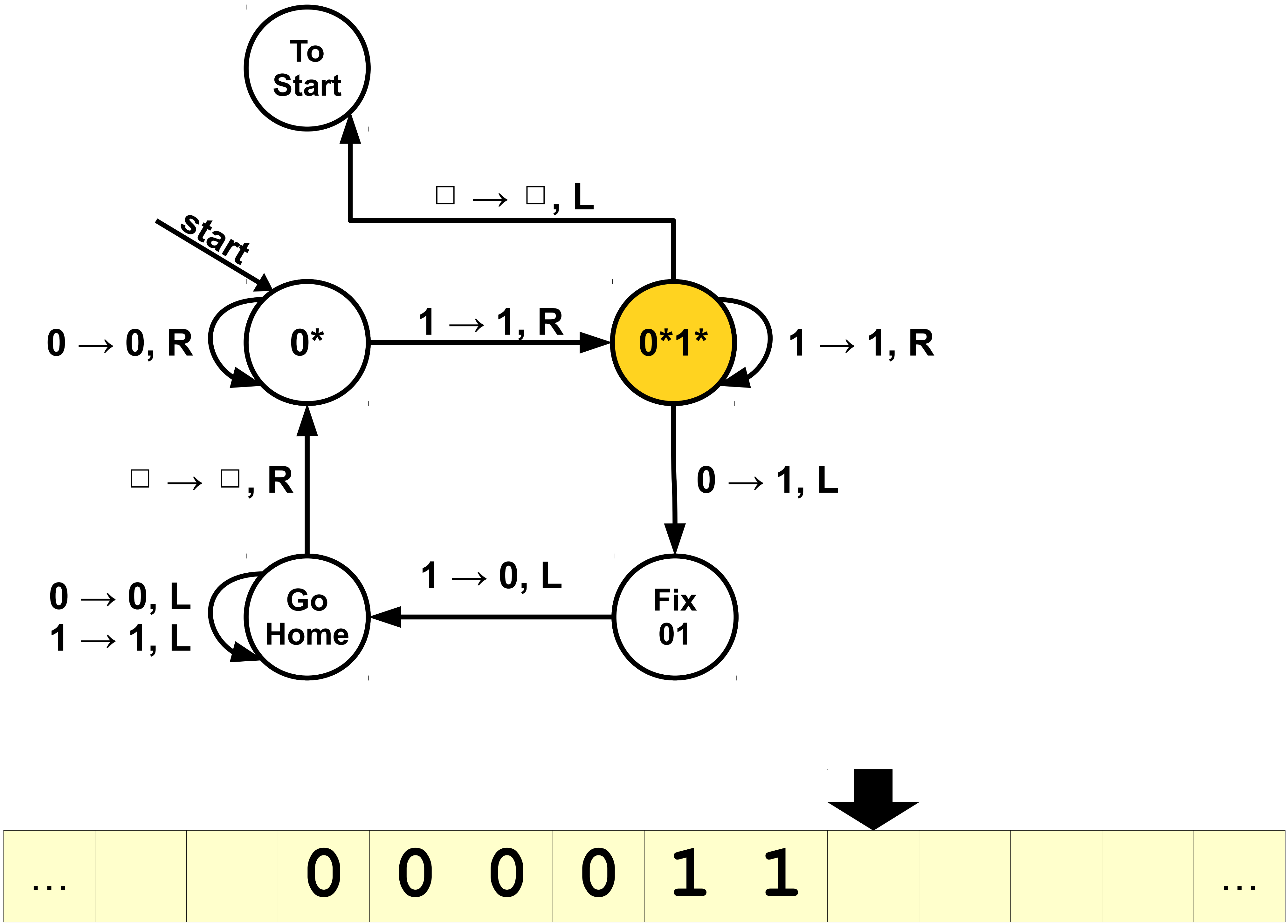
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…



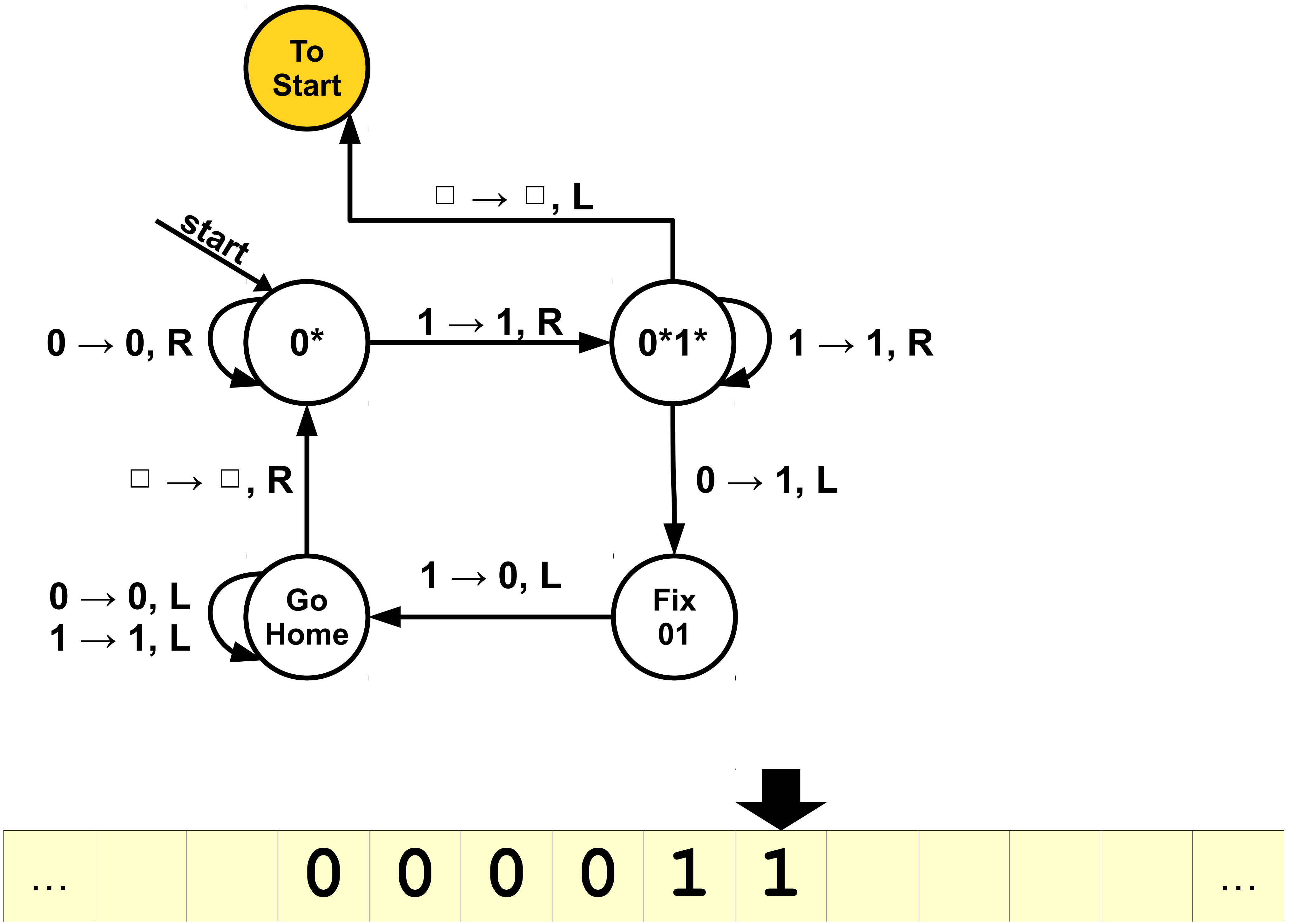
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

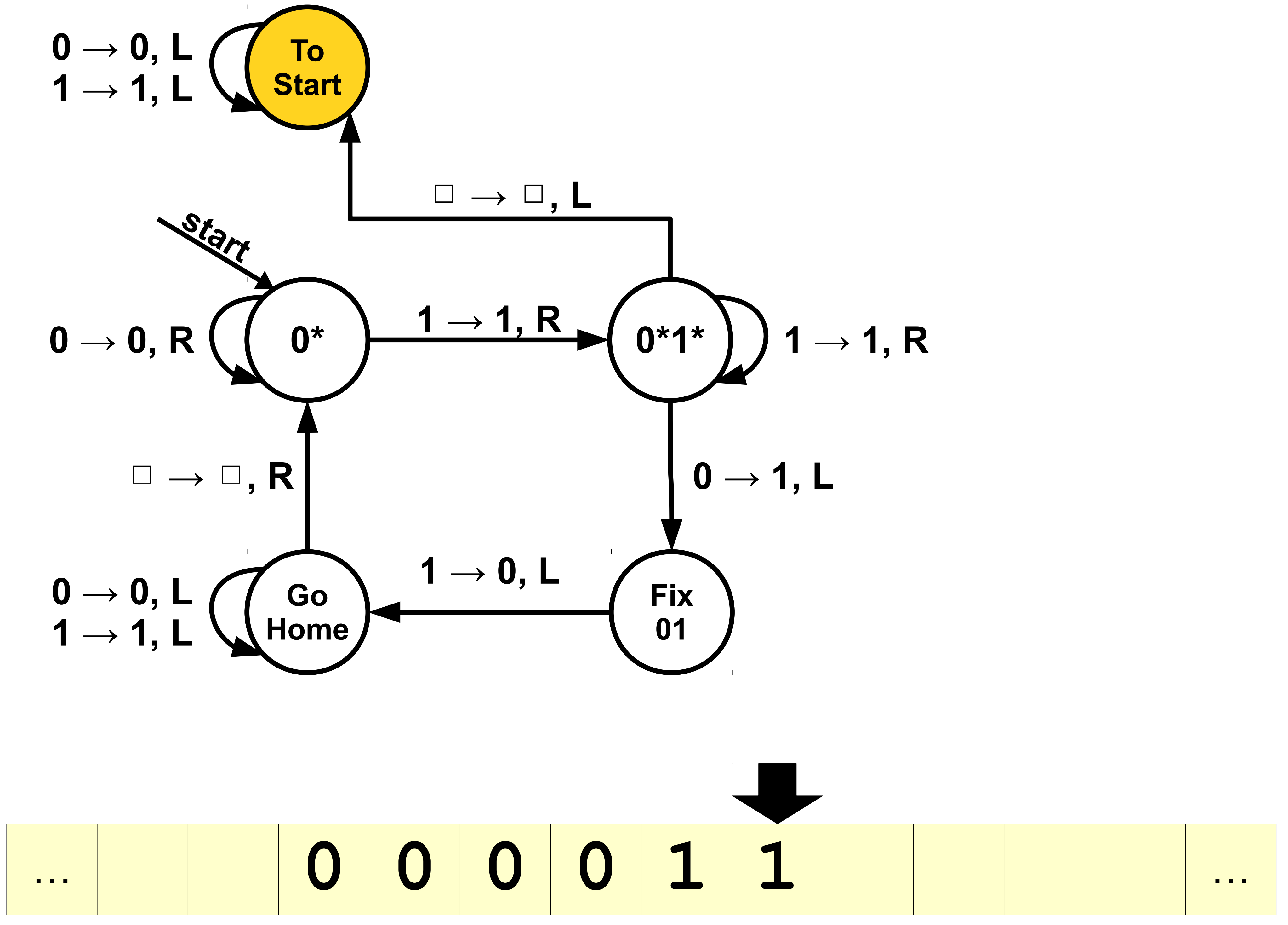
Our ultimate goal here was to sort everything so we could hand it off to the machine to check for 0^n1^n. Let's rewind the tape head back to the start.



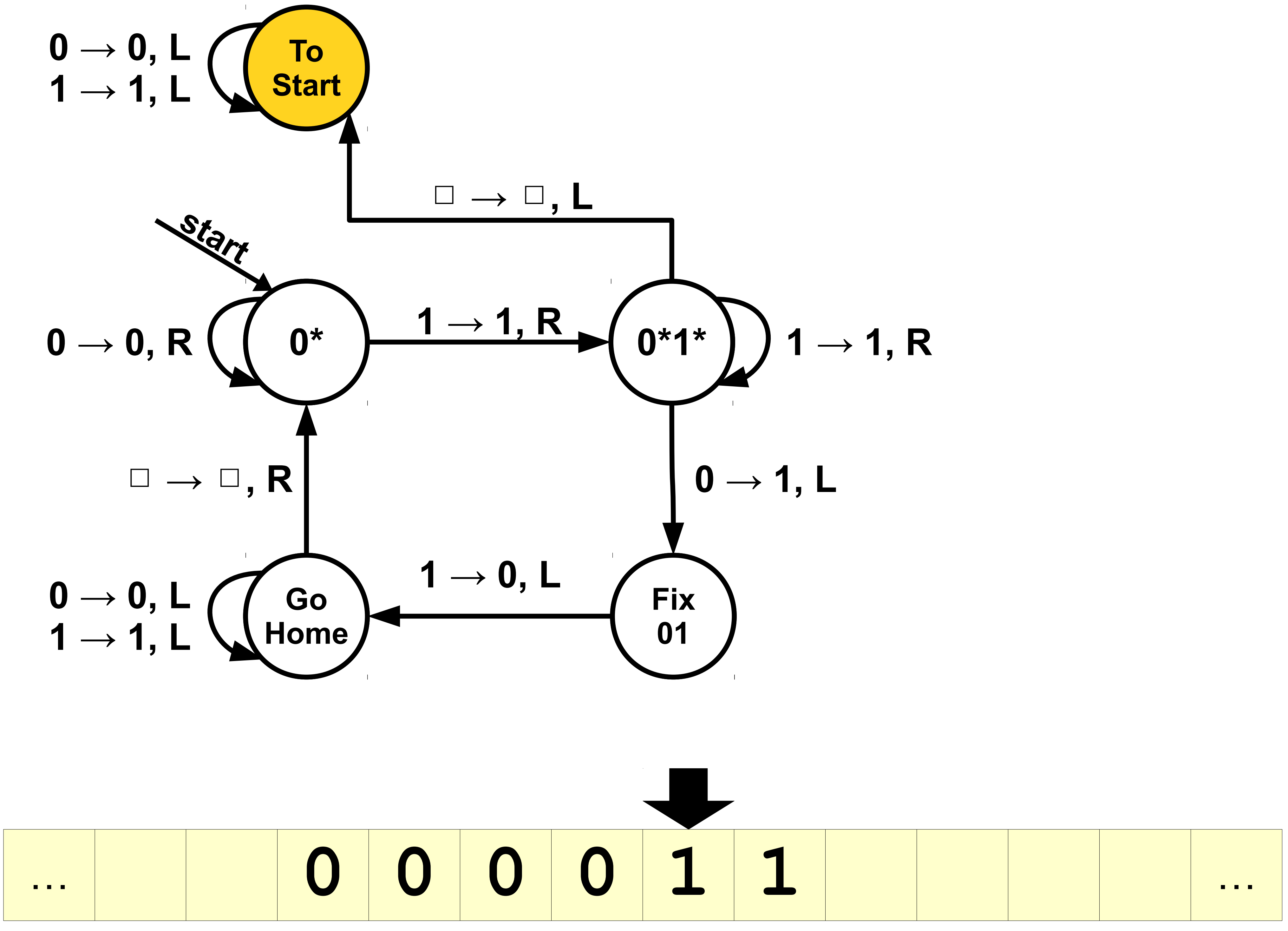


To Start
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
… 0 0 0 0 1 1 …

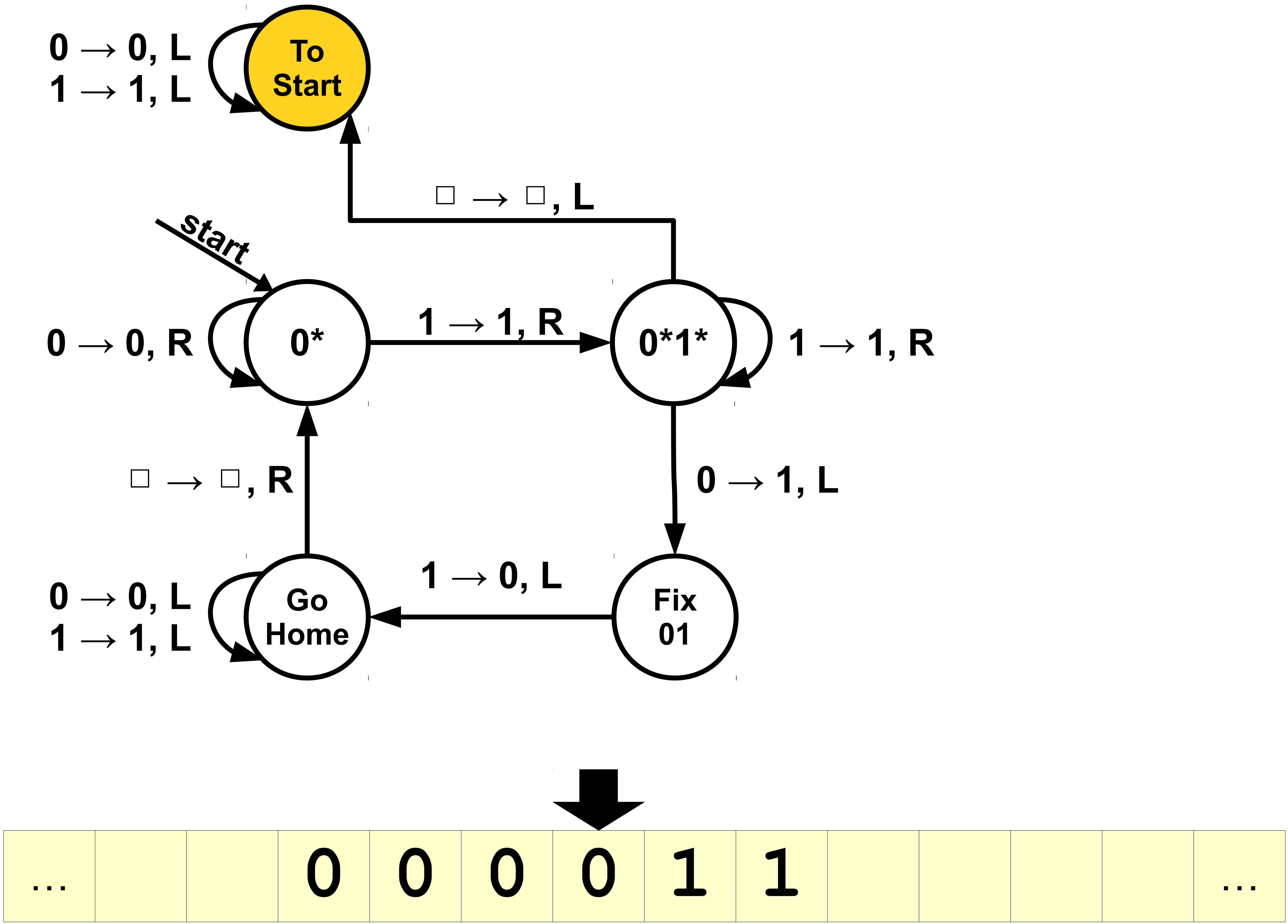

To Start
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…



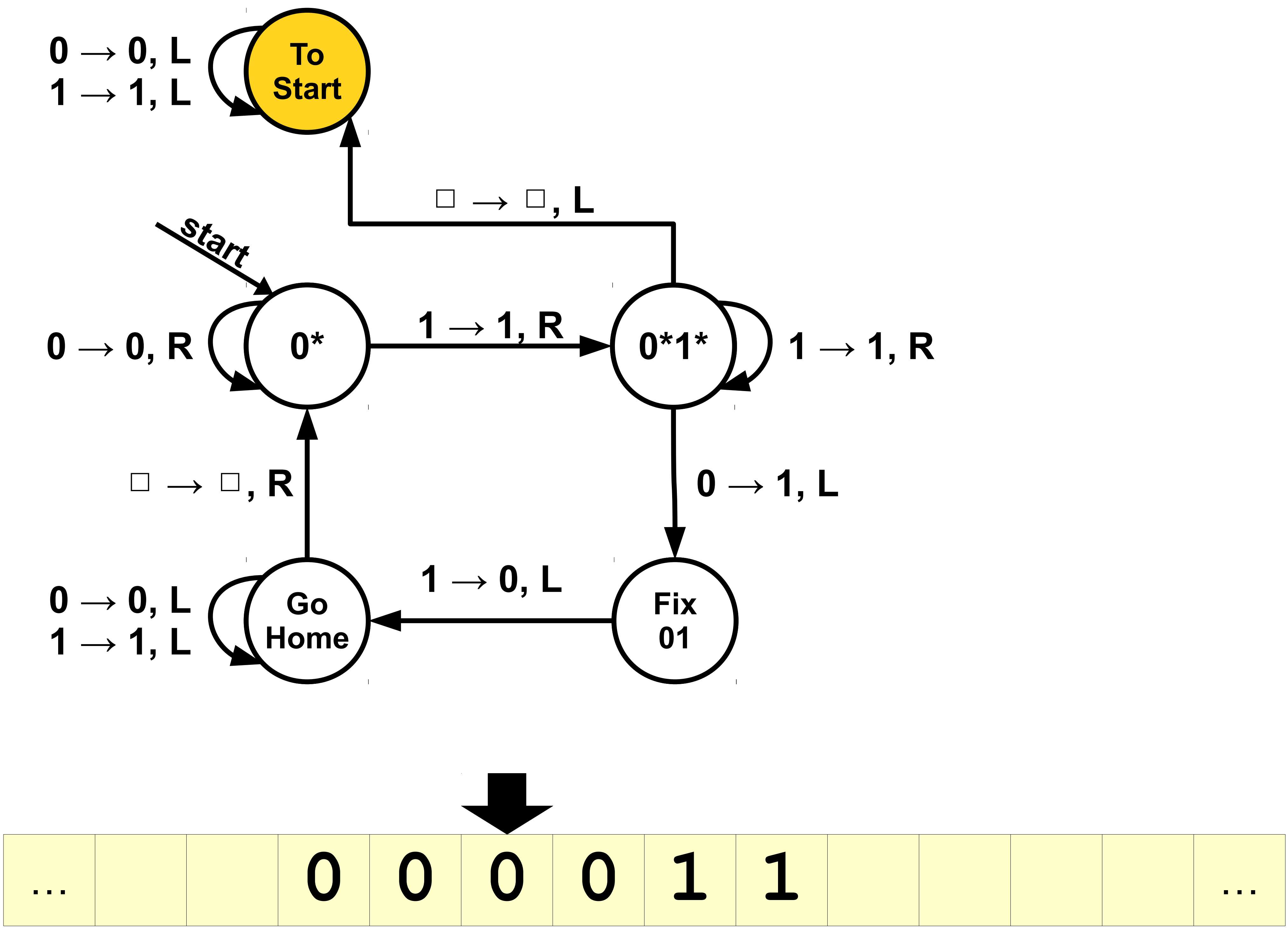
To Start
0 → 0, L
1 → 1, L
□ → □, L
start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…



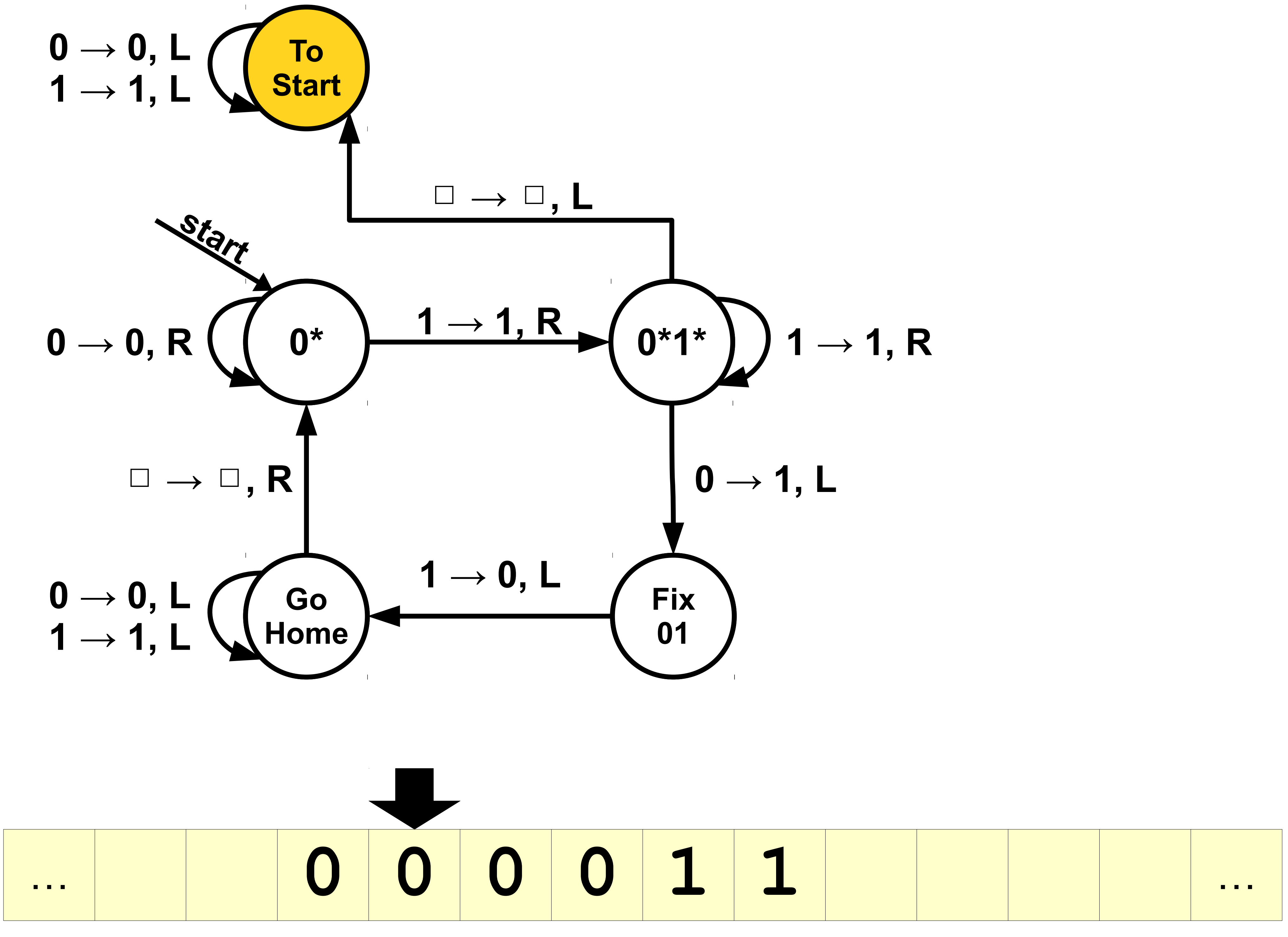
0 → 0, L
1 → 1, L
To Start
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

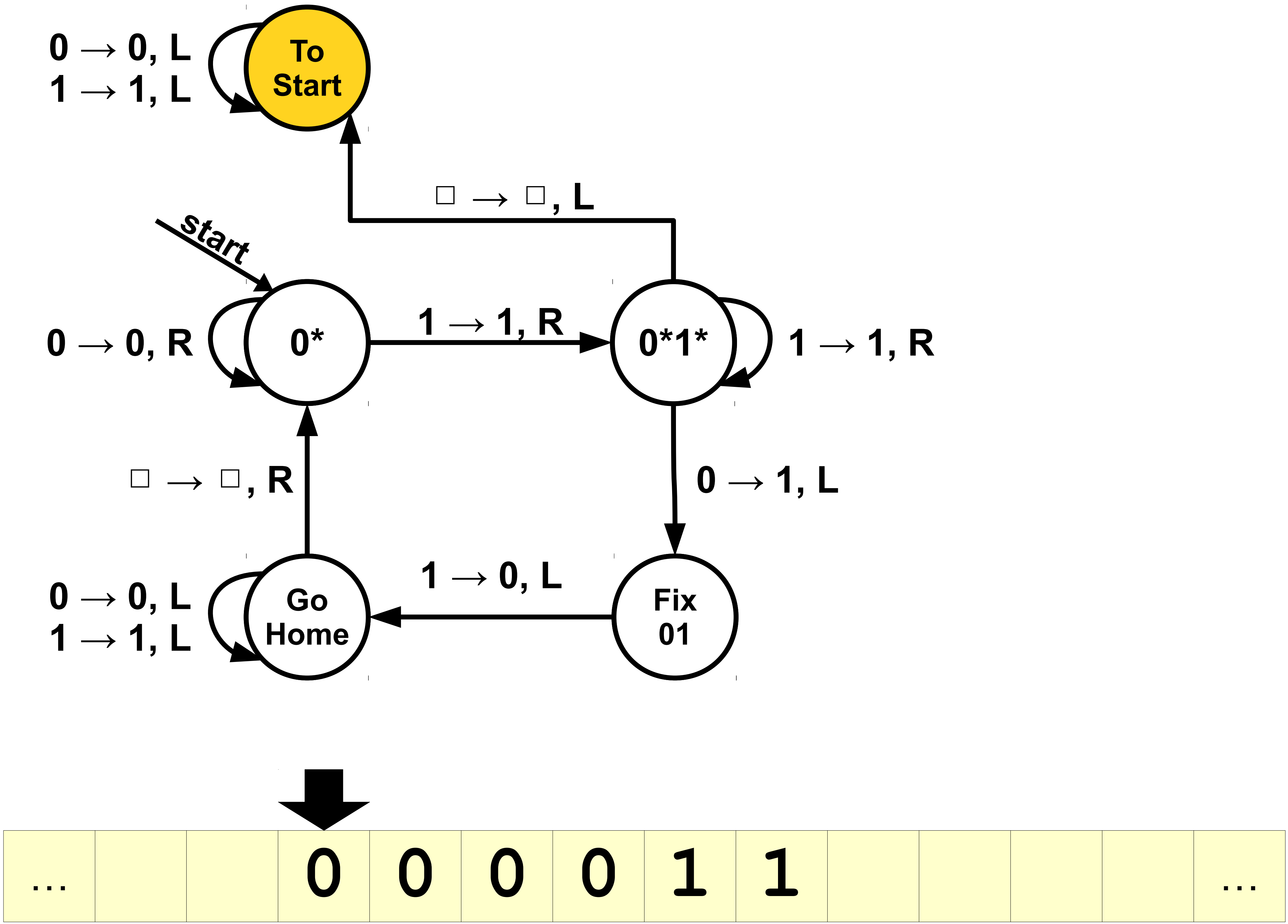


To Start
0 → 0, L
1 → 1, L
□ → □, L
start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

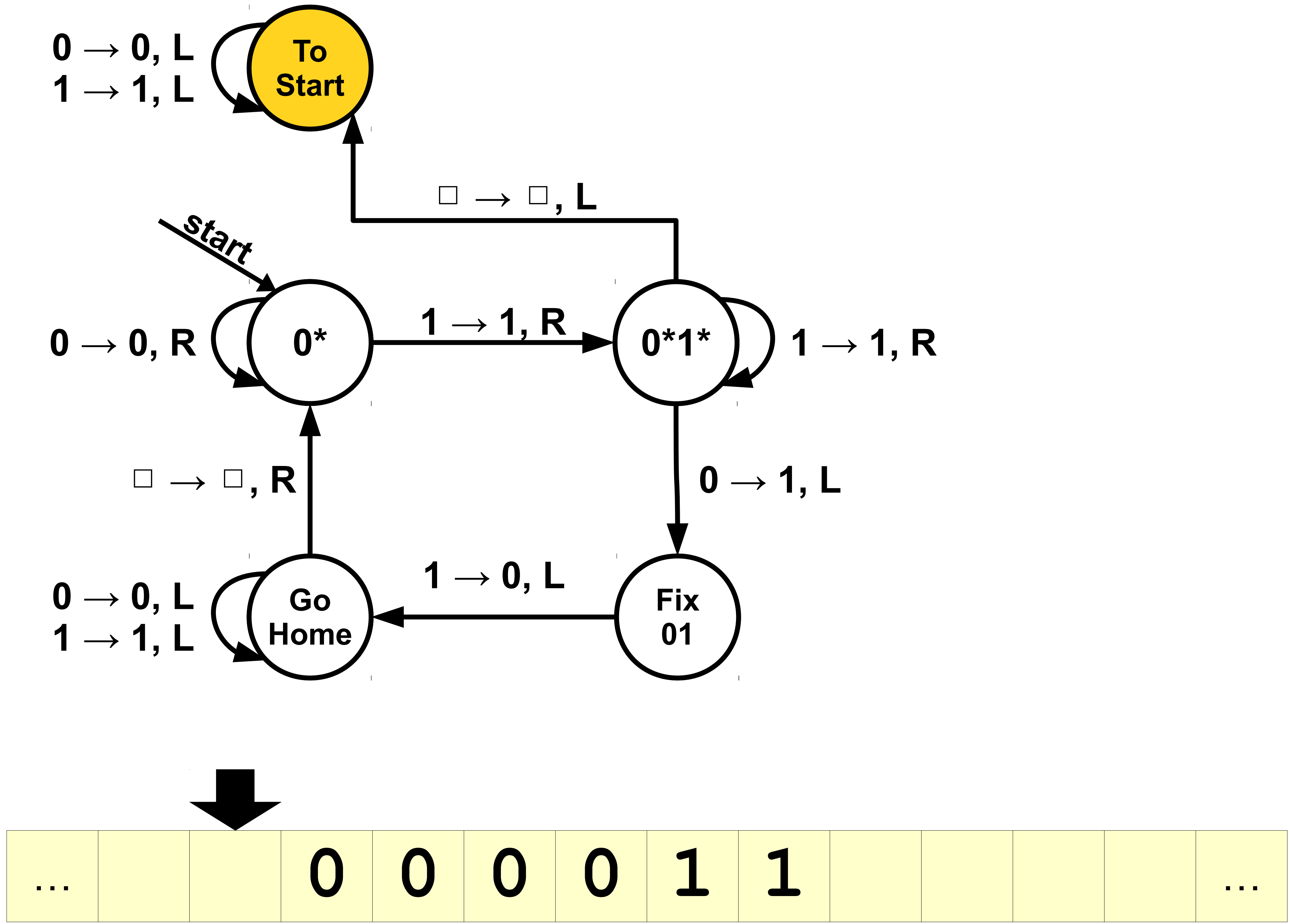


0 → 0, L
1 → 1, L
To Start
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…

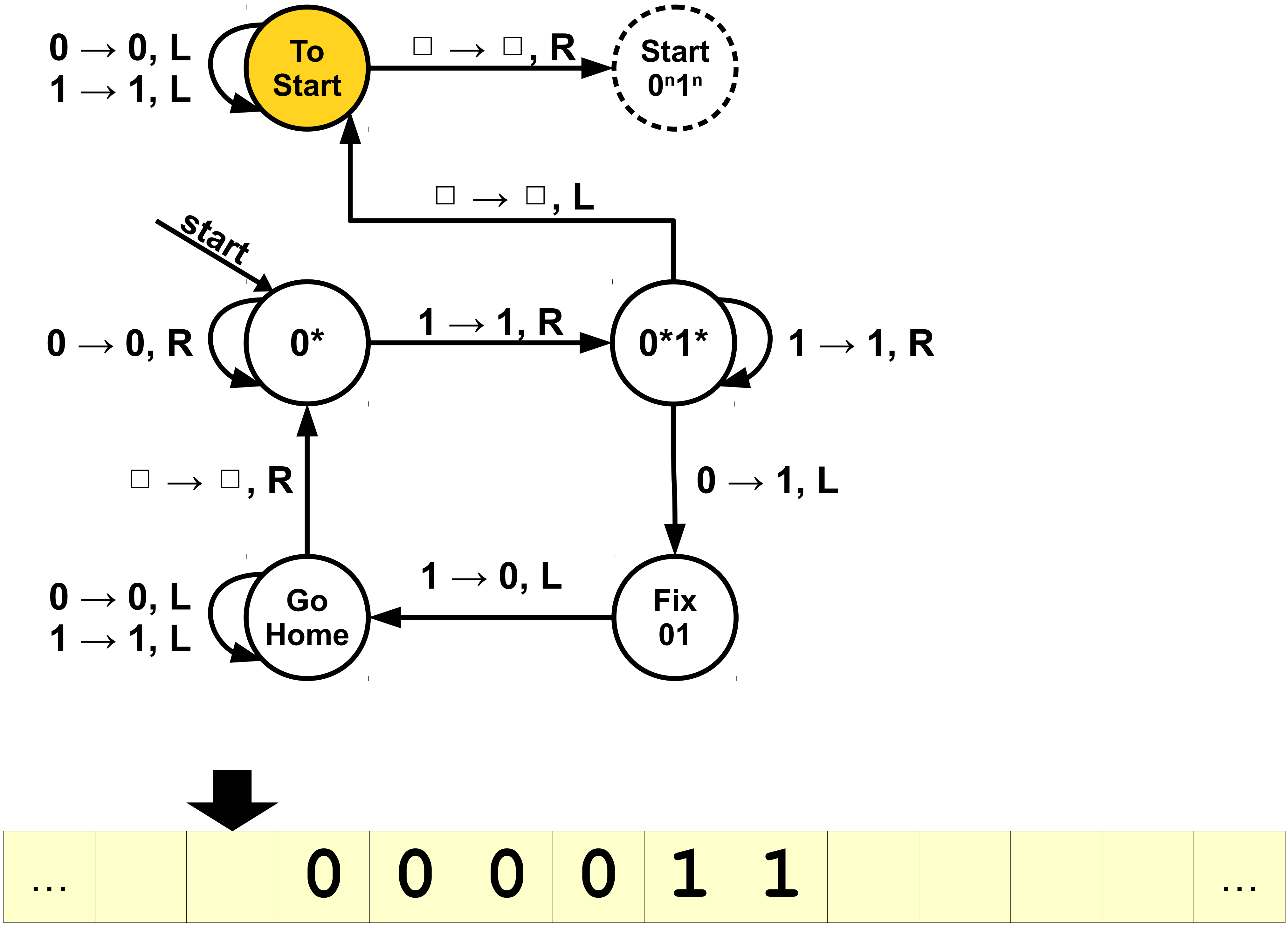

To Start
0 → 0, L
1 → 1, L
□ → □, L
start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
… 0 0 0 0 1 1 …



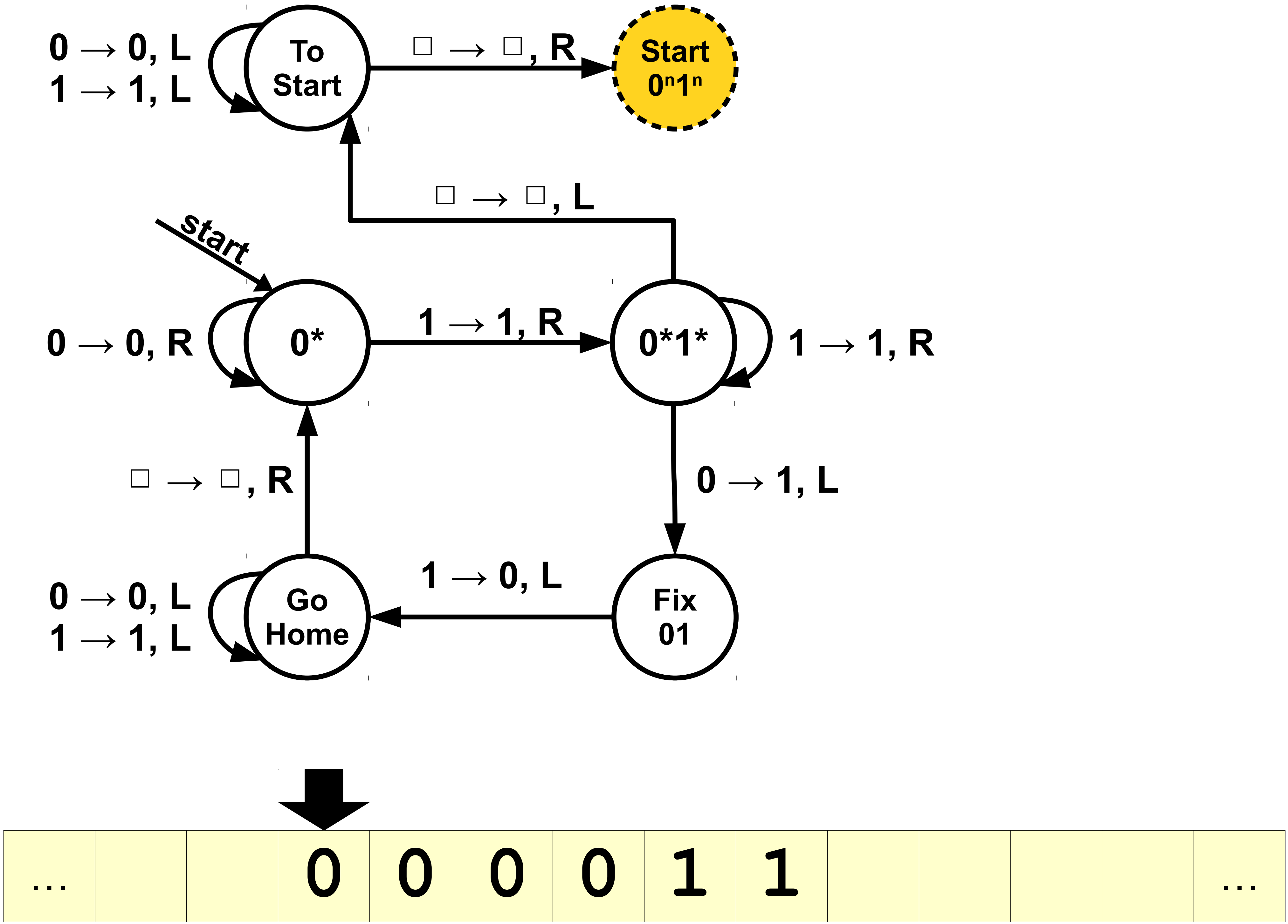
0 → 0, L
1 → 1, L
To Start
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…



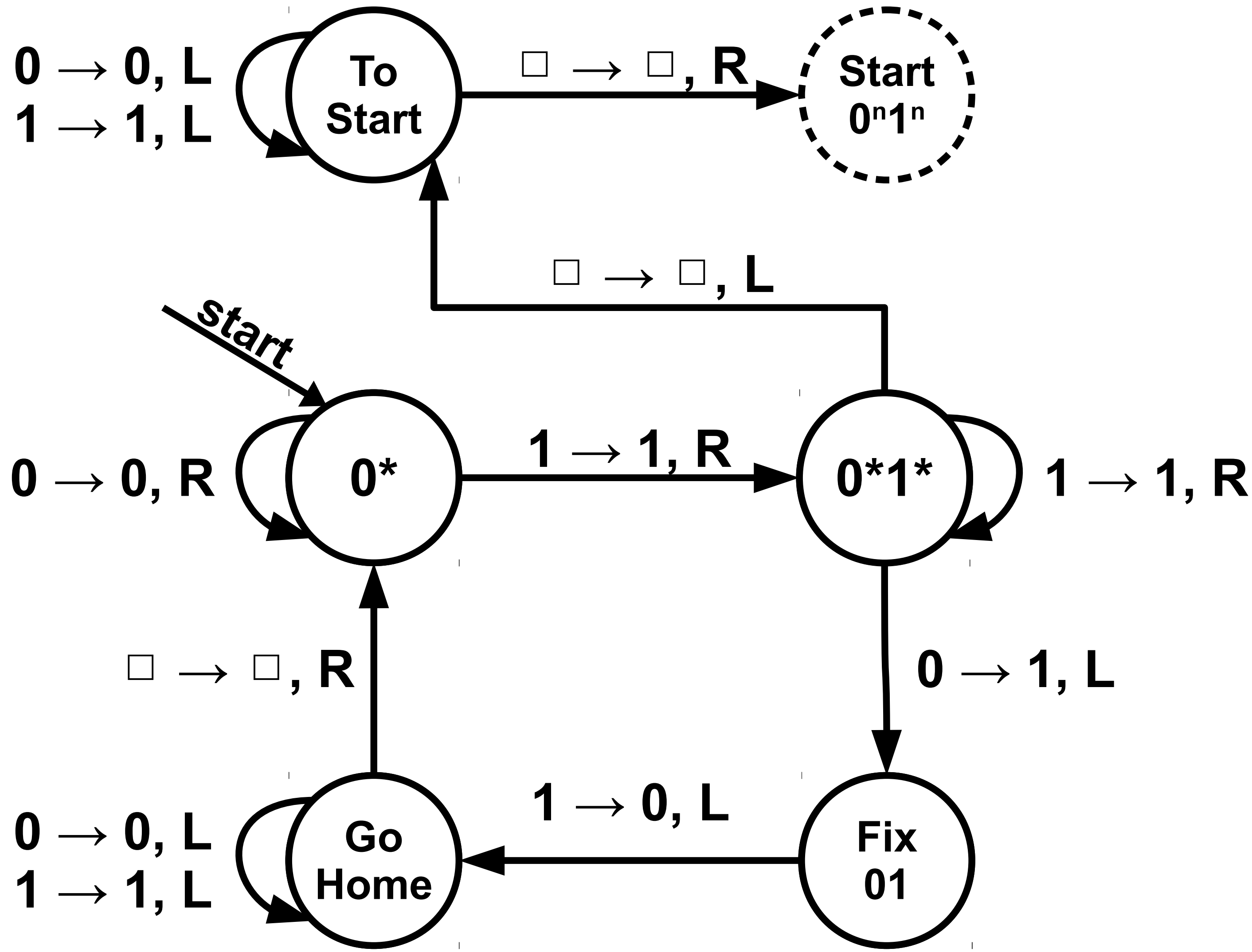
To Start
0 → 0, L
1 → 1, L
□ → □, L
start
0*
0 → 0, R
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
Go Home
0 → 0, L
1 → 1, L
1 → 0, L
Fix 01
…
0
0
0
0
1
1
…



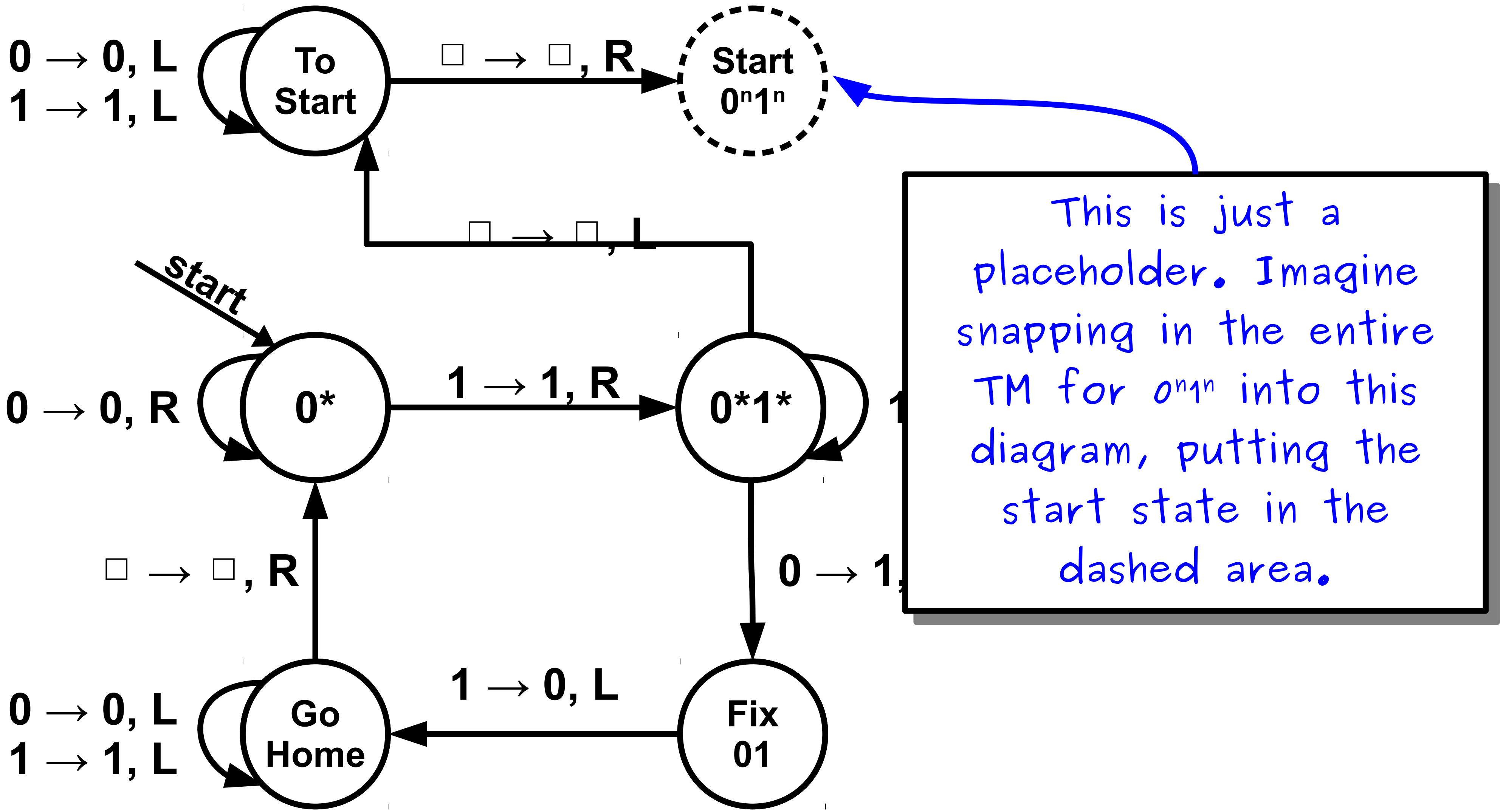
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0ⁿ1ⁿ
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
… 0 0 0 0 1 1 …

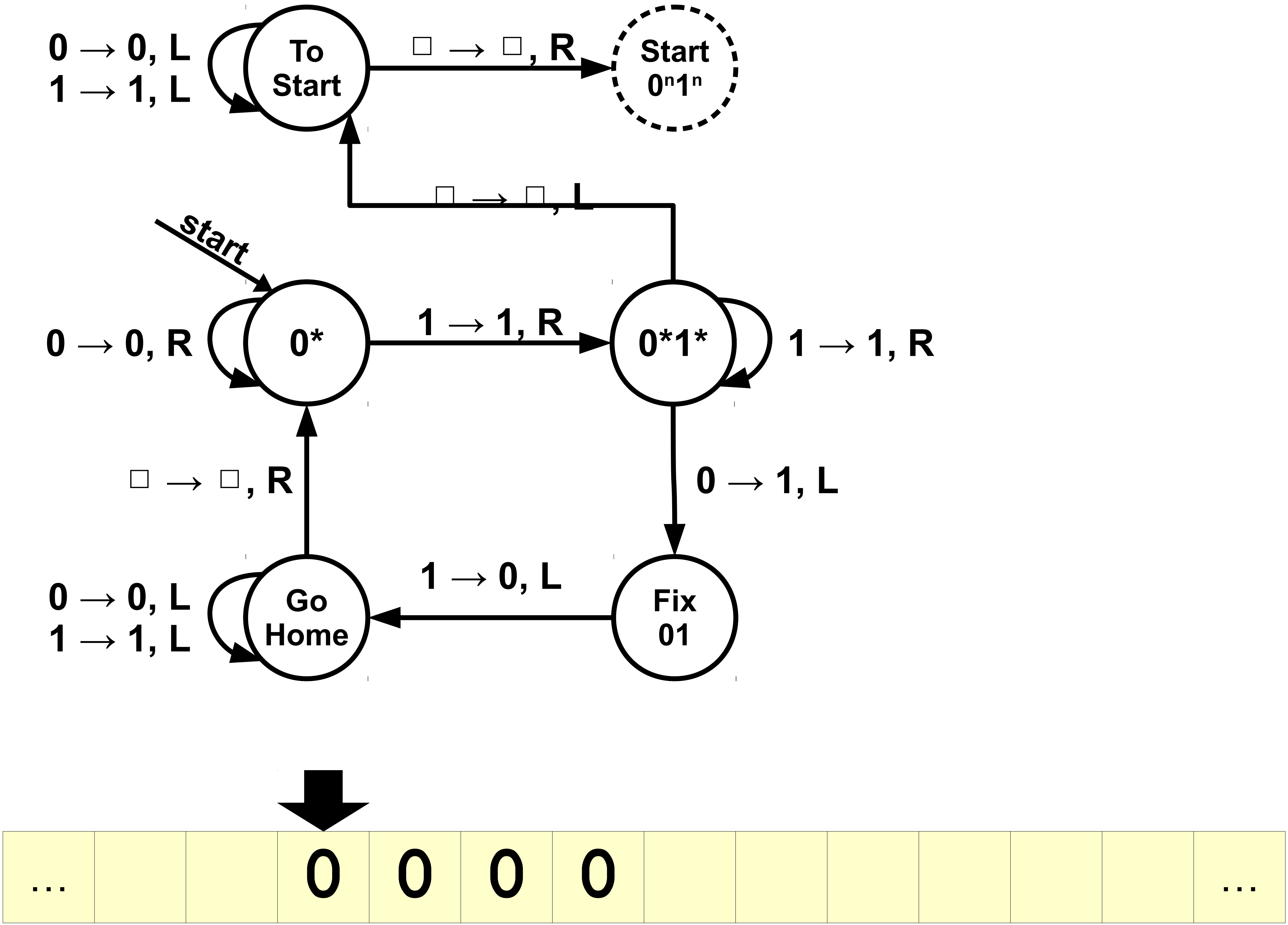


0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
Go Home
Fix 01
…
0
0
0
0
1
1
…

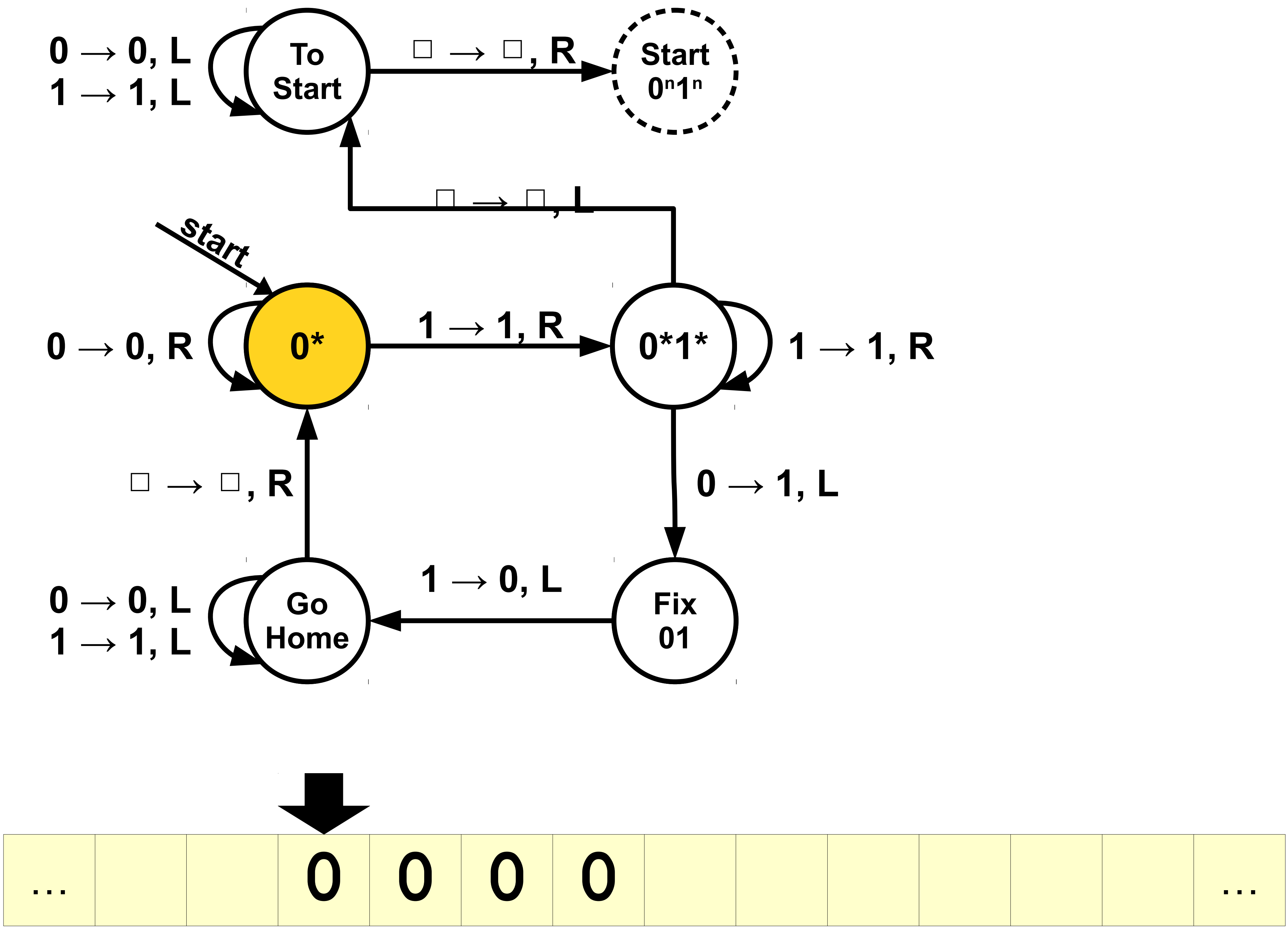


0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01

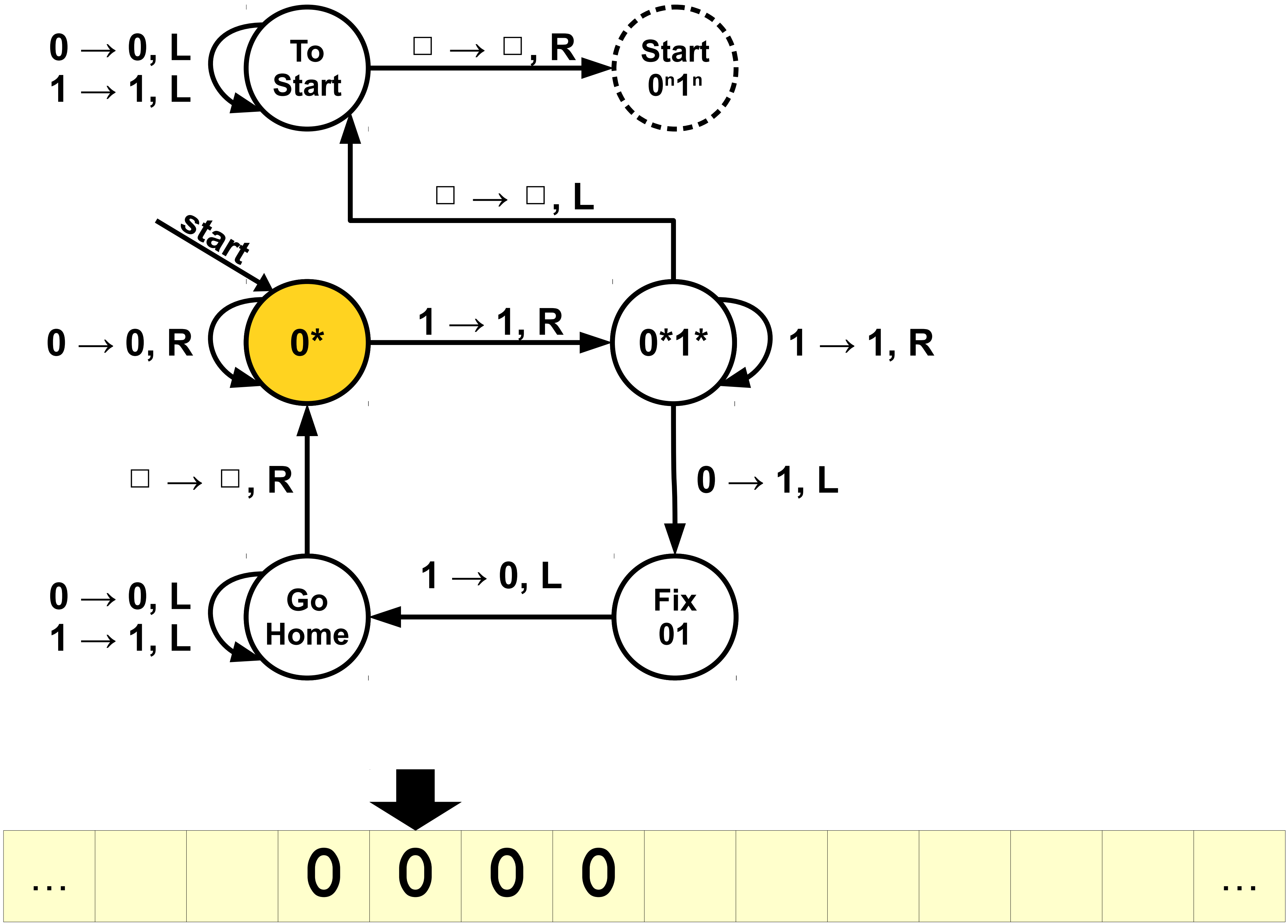

0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
□ → □, R
0 → 1,
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
This is just a placeholder. Imagine snapping in the entire TM for 0^n1^n into this diagram, putting the start state in the dashed area.



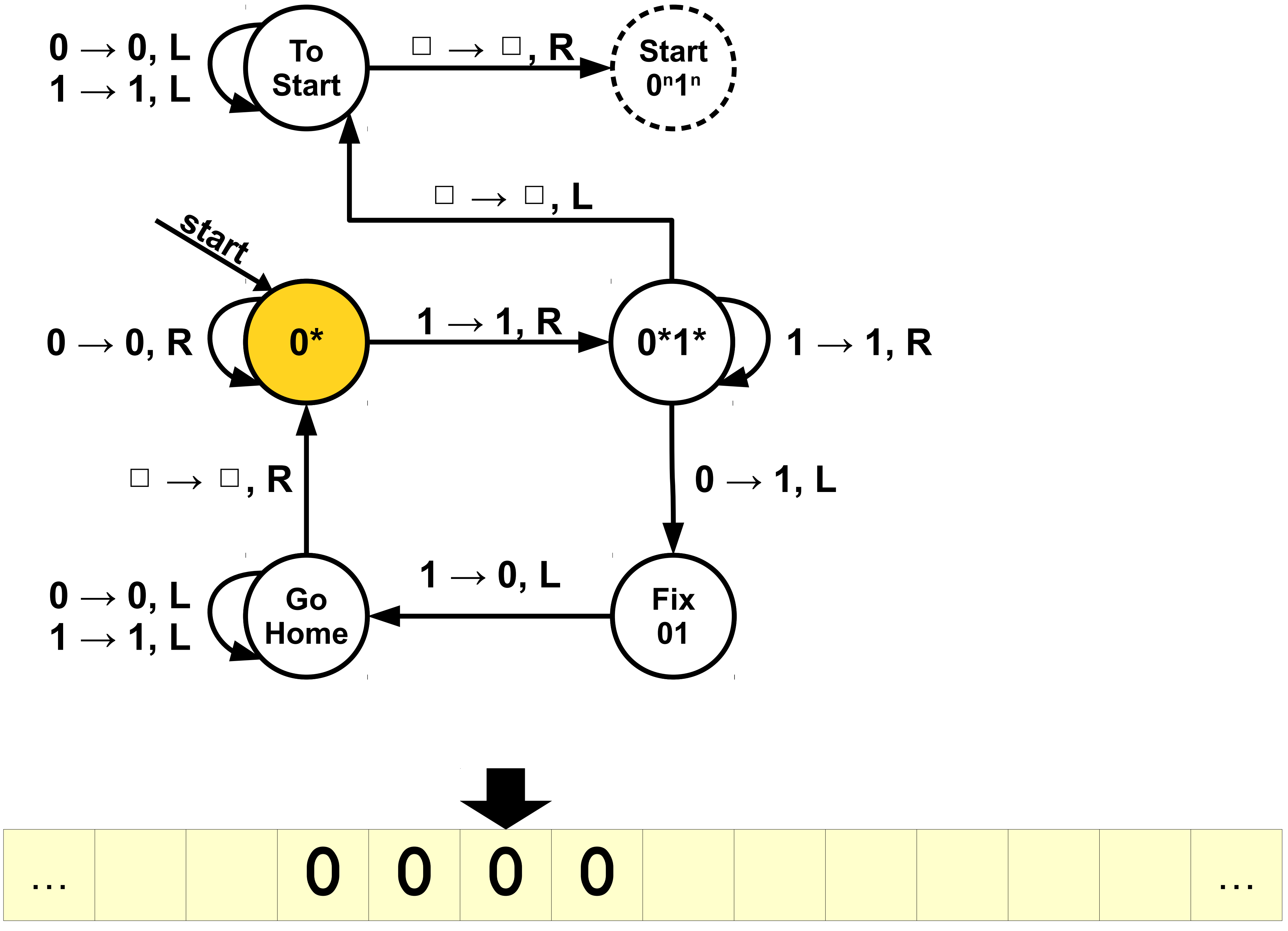
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0ⁿ1ⁿ
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
0 → 1, L
□ → □, R
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…



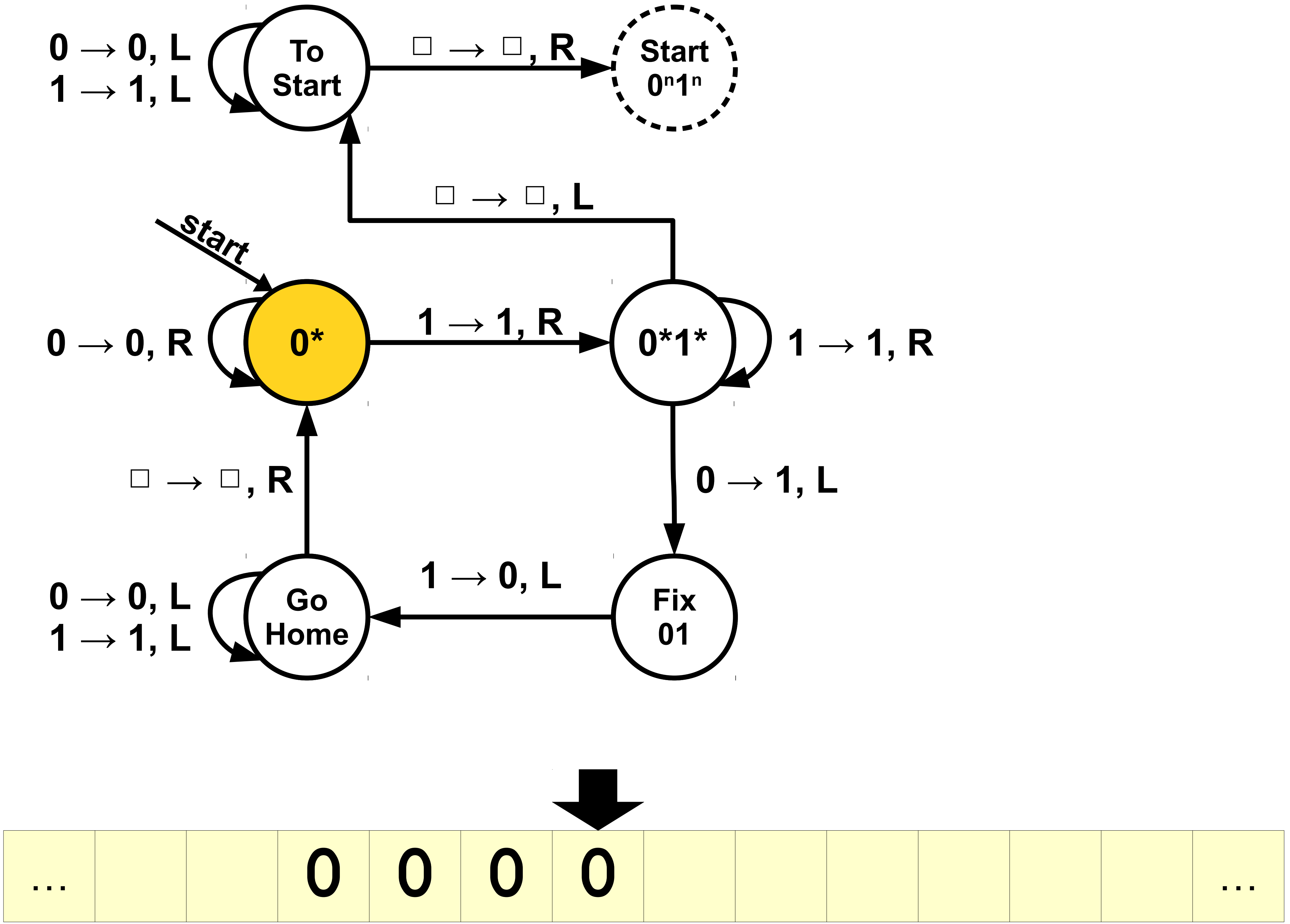
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…

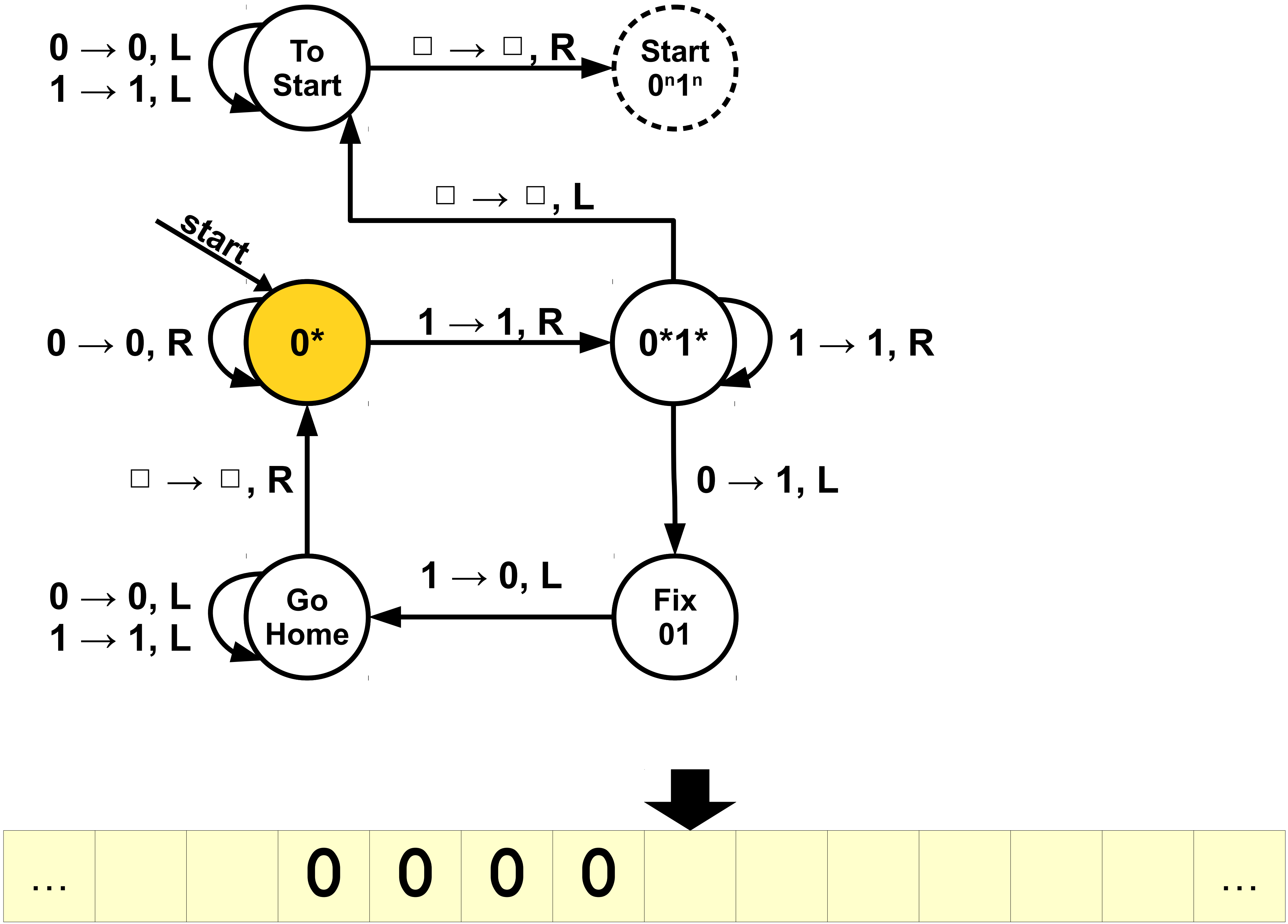


0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…

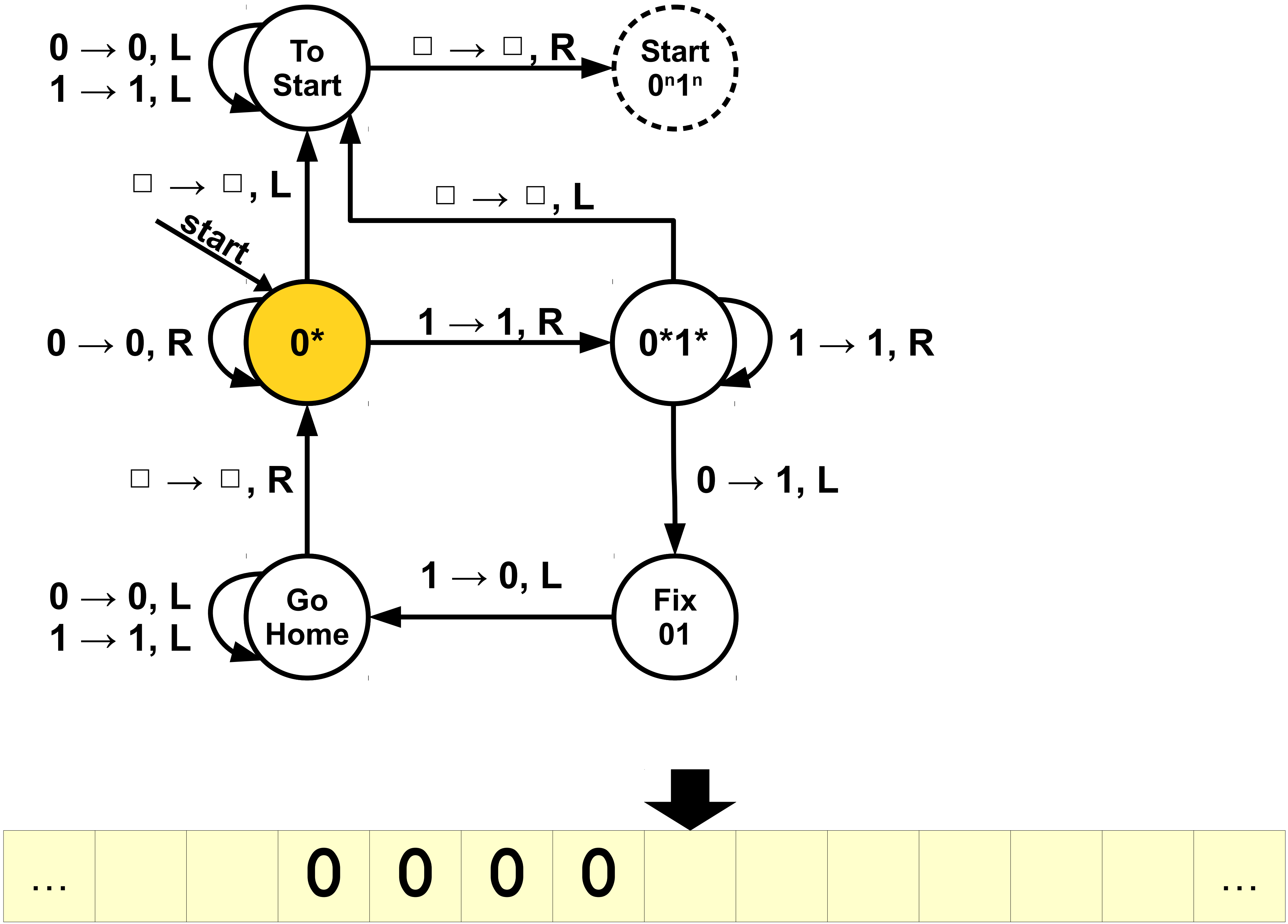


0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
… 0 0 0 0 …

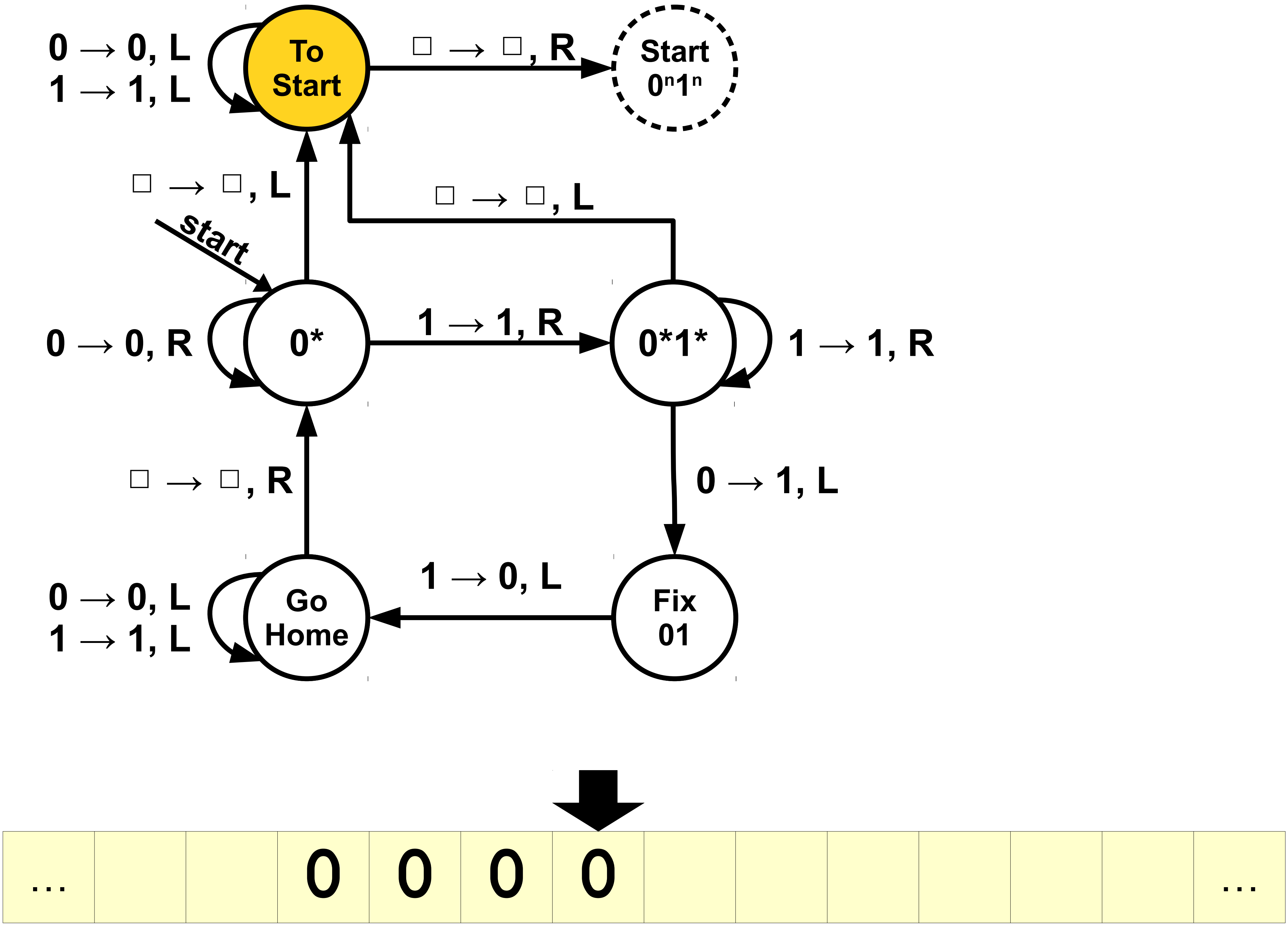

To Start
Start 0ⁿ1ⁿ
0 → 0, L
1 → 1, L
□ → □, R
□ → □, L
start
0*
0*1*
0 → 0, R
1 → 1, R
1 → 1, R
□ → □, R
0 → 1, L
Go Home
Fix 01
0 → 0, L
1 → 1, L
1 → 0, L
… 0 0 0 0 …



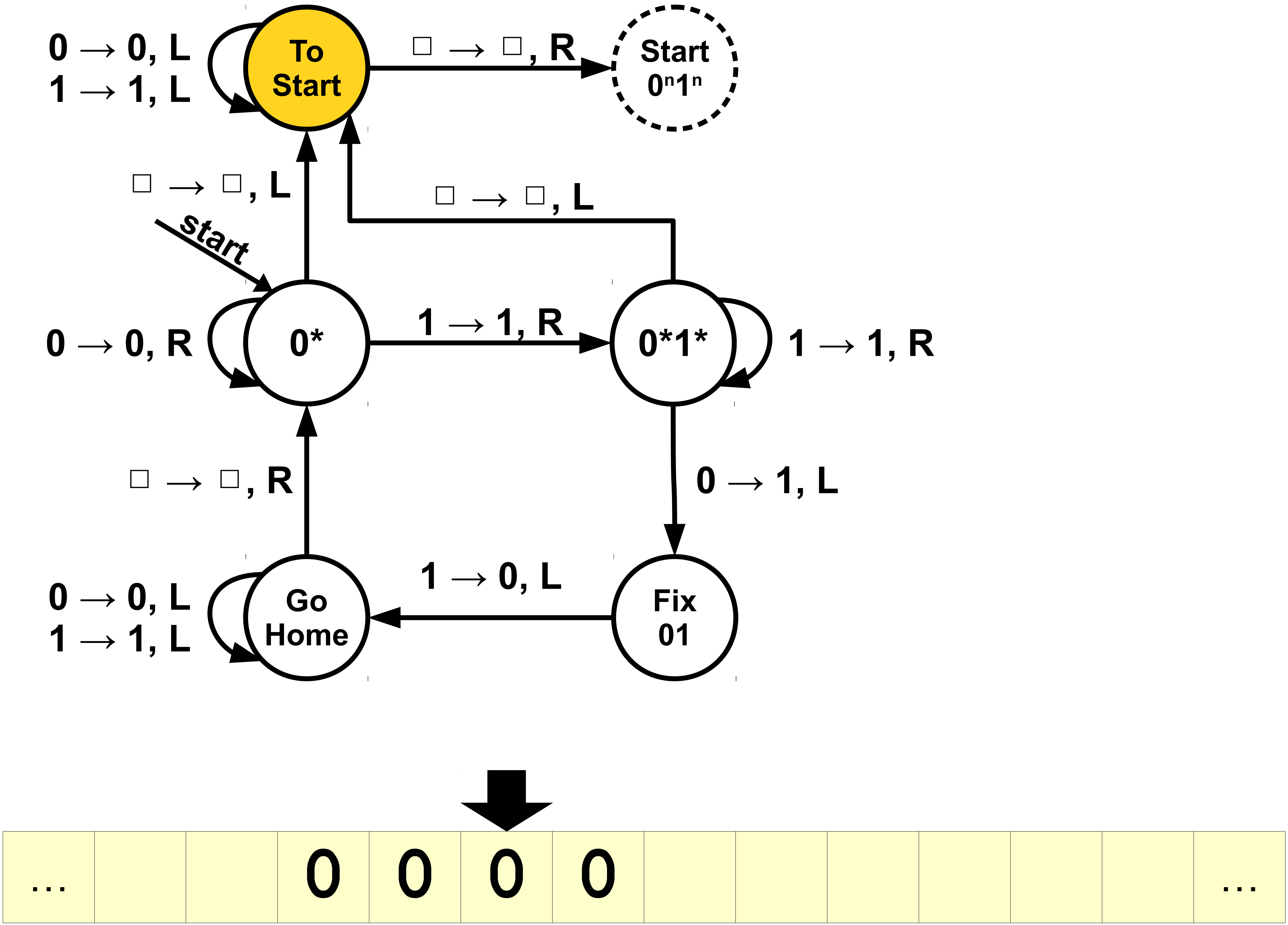
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0ⁿ1ⁿ
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…



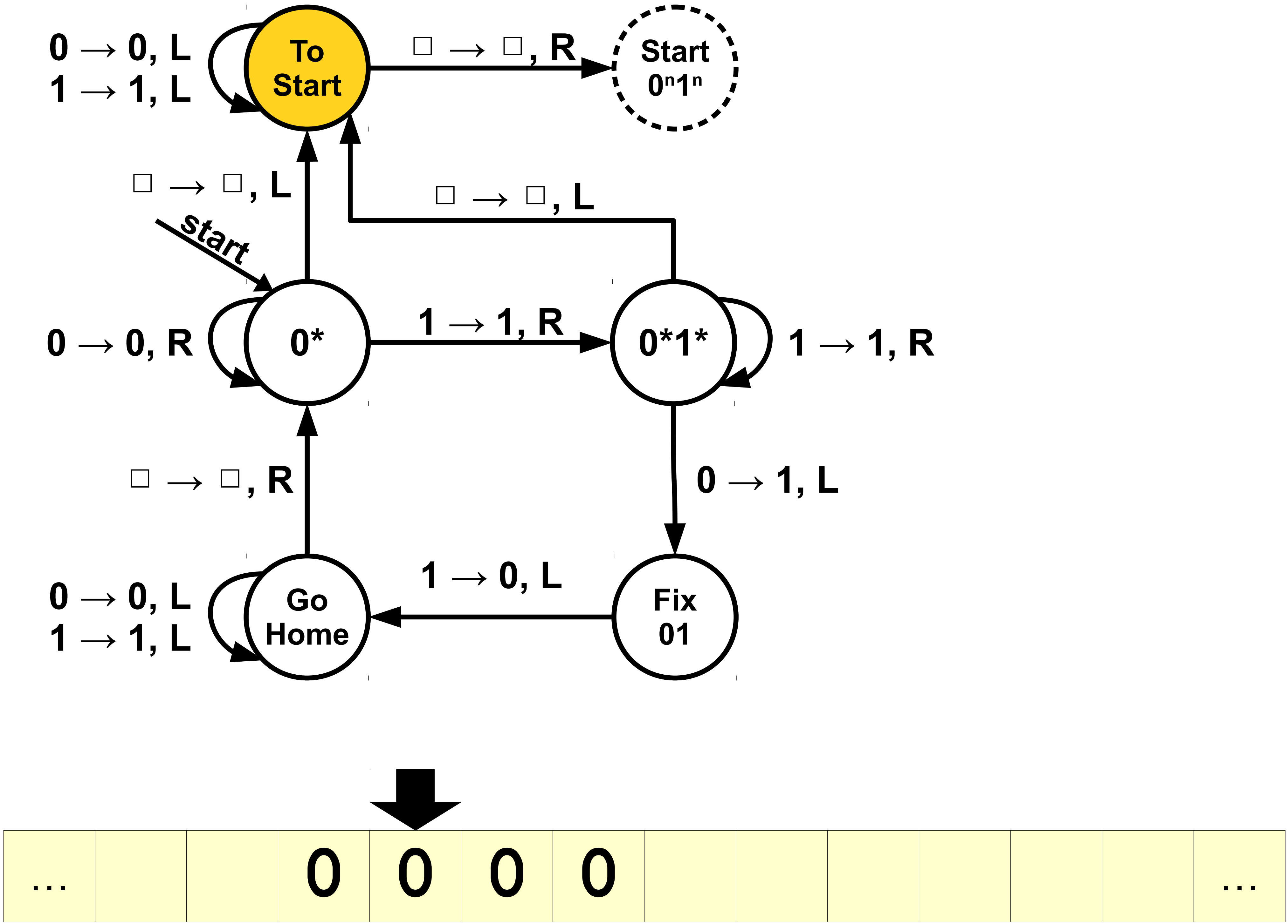
To Start
Start 0ⁿ1ⁿ
0*
0*1*
Go Home
Fix 01
start
0 → 0, L
1 → 1, L
□ → □, R
□ → □, L
□ → □, L
0 → 0, R
1 → 1, R
1 → 1, R
□ → □, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
… 0 0 0 0 …



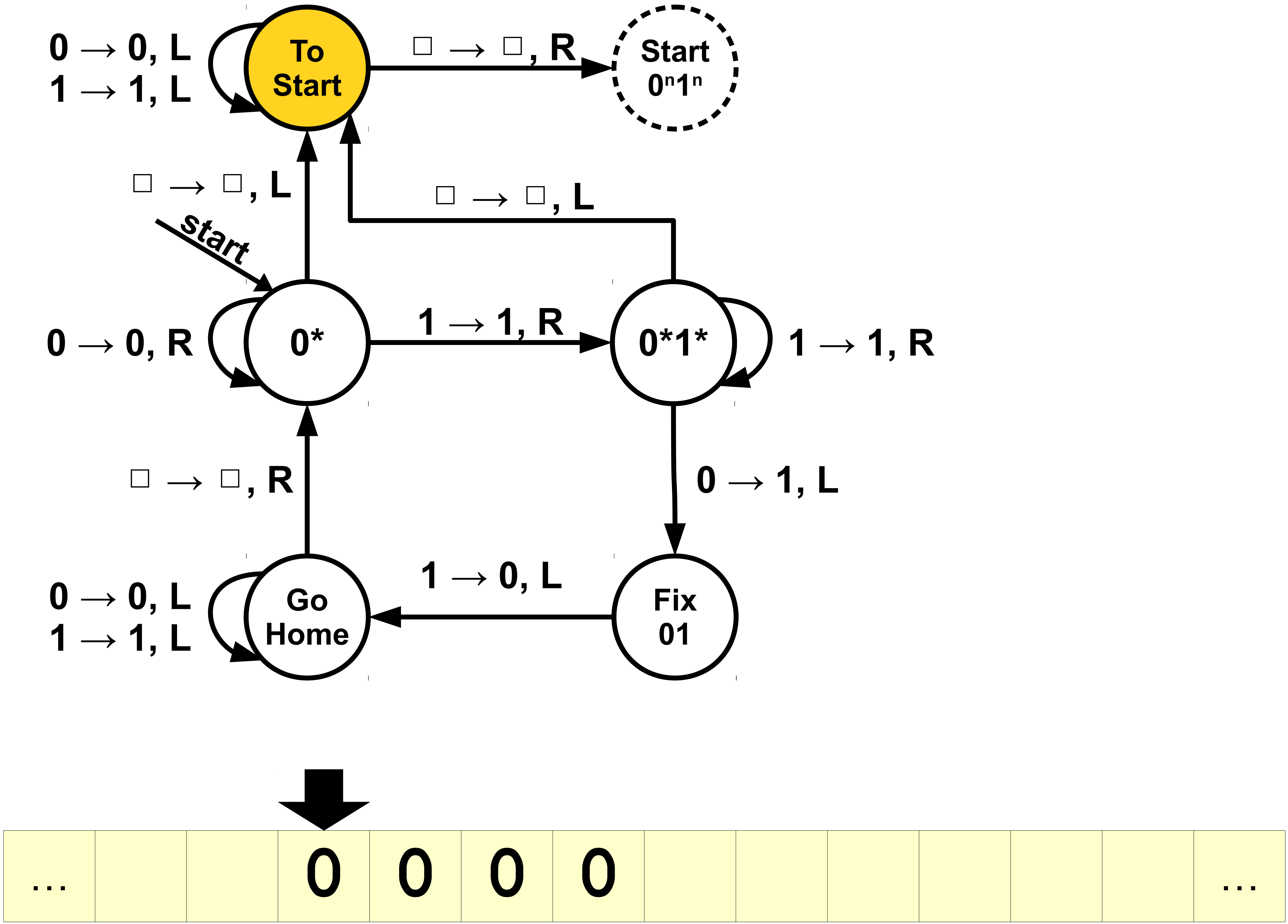
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
□ → □, L
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…



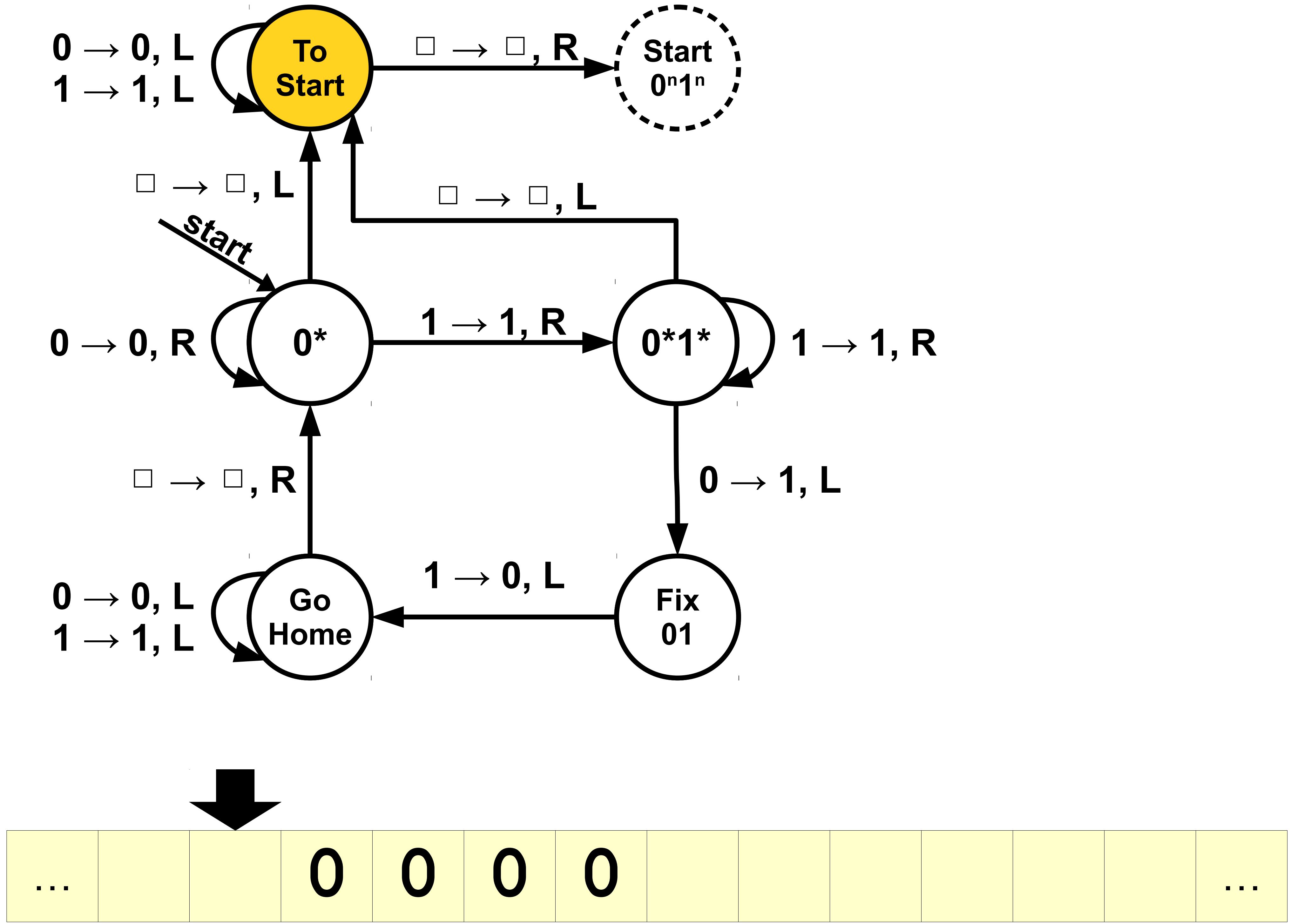
0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
□ → □, L
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…

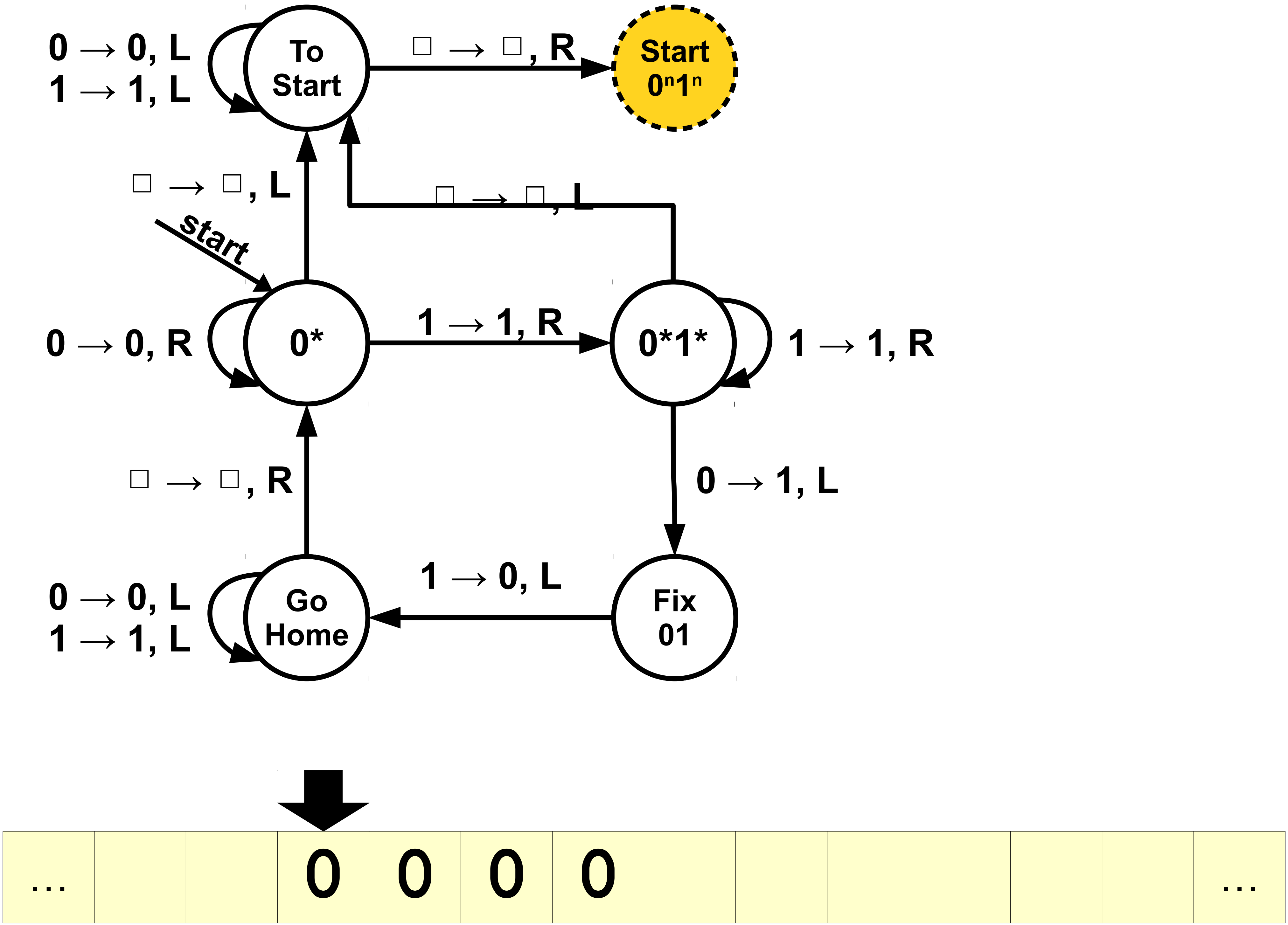


0 → 0, L
1 → 1, L
To Start
□ → □, R
Start
0ⁿ1ⁿ
□ → □, L
start
□ → □, L
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…

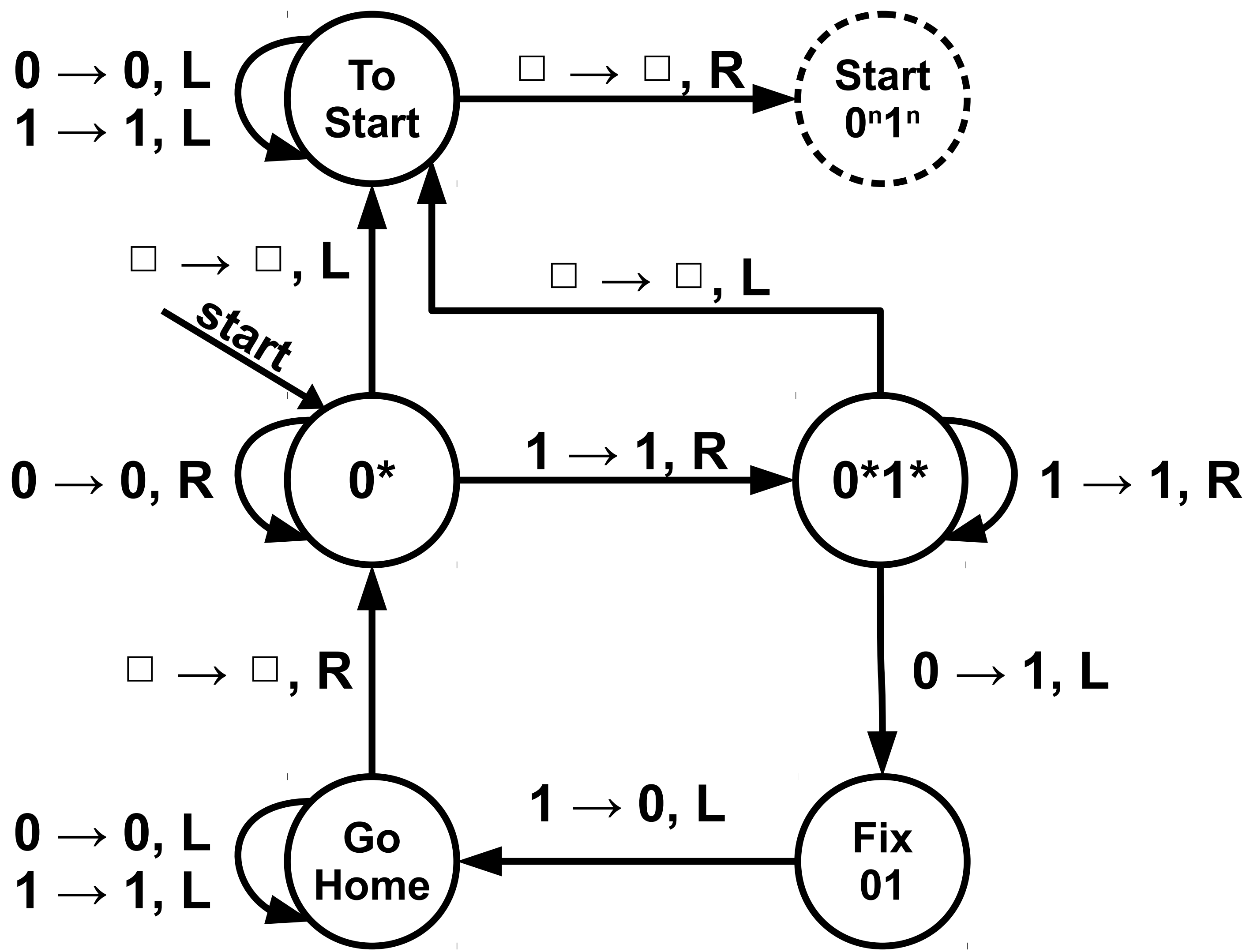


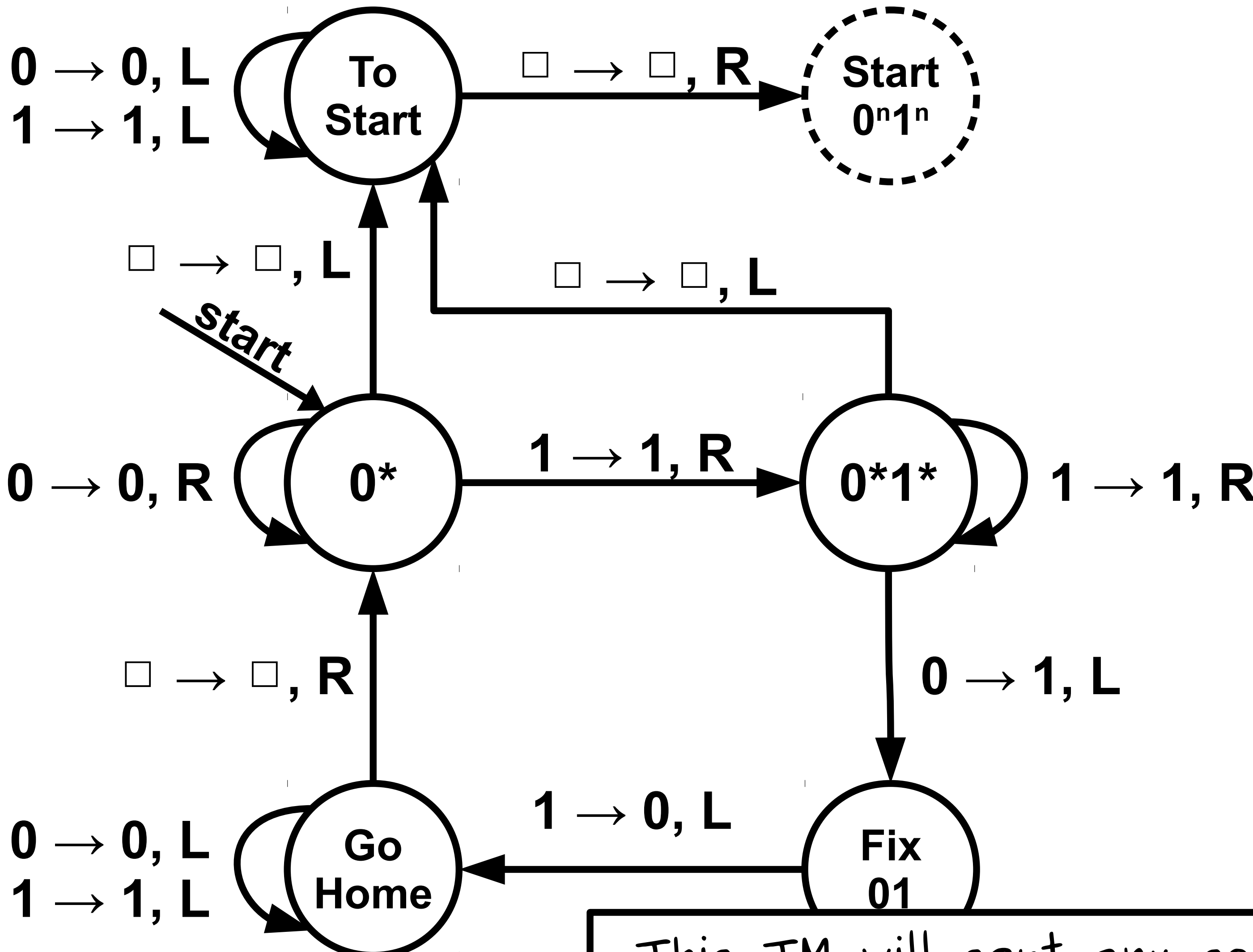
To Start
Start 0^n1^n
0*
0*1*
Go Home
Fix 01
start
$0 \to 0, L$
$1 \to 1, L$
$\square \to \square, R$
$\square \to \square, L$
$\square \to \square, L$
$0 \to 0, R$
$1 \to 1, R$
$1 \to 1, R$
$\square \to \square, R$
$0 \to 1, L$
$0 \to 0, L$
$1 \to 1, L$
$1 \to 0, L$
… 0 0 0 0 …


To Start
Start 0^n1^n
0*
0*1*
Go Home
Fix 01
start
0 → 0, L
1 → 1, L
□ → □, R
□ → □, L
□ → □, L
0 → 0, R
1 → 1, R
1 → 1, R
□ → □, R
0 → 1, L
1 → 0, L
0 → 0, L
1 → 1, L
…
0
0
0
0
…



0 → 0, L
1 → 1, L
To Start
□ → □, R
Start
0^n1^n
□ → □, L
start
□ → □, L
0 → 0, R
0*
1 → 1, R
0*1*
1 → 1, R
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
…
0
0
0
0
…


To Start
Start 0^n1^n
0*
0*1*
Go Home
Fix 01
start
$0 \rightarrow 0, L$
$1 \rightarrow 1, L$
$\square \rightarrow \square, R$
$\square \rightarrow \square, L$
$\square \rightarrow \square, L$
$0 \rightarrow 0, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$0 \rightarrow 1, L$
$\square \rightarrow \square, R$
$1 \rightarrow 0, L$
$0 \rightarrow 0, L$
$1 \rightarrow 1, L$



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

Fun problem: design a TM that sorts a string of 0s and 1s, but does so while taking way fewer steps than this machine.

TM Subroutines

- A ***TM subroutine*** is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.
- TM subroutines let us compose larger TMs out of smaller TMs, just as you'd write a larger program using lots of smaller helper functions.
- Here, we saw a TM subroutine that sorts a sequence of 0s and 1s into ascending order.

TM Subroutines

- Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.
- When we're composing multiple subroutines together – which we'll do in a bit – the idea is that we'll snap in some real state for the “done” state.

Where We Stand

- What have we seen TMs do so far?
  - Operate on numbers.
  - Sort sequences of values.
  - Break tasks down into smaller pieces.
- Here are a few other tasks TMs can do:
  - Work with base-10 numbers.
  - Increment and decrement numbers.
  - Add numbers.
- Aren't these, you know, the things computers do?

If you're curious to see how this is done, check the appendix for this lecture. You aren't required to do this, though.

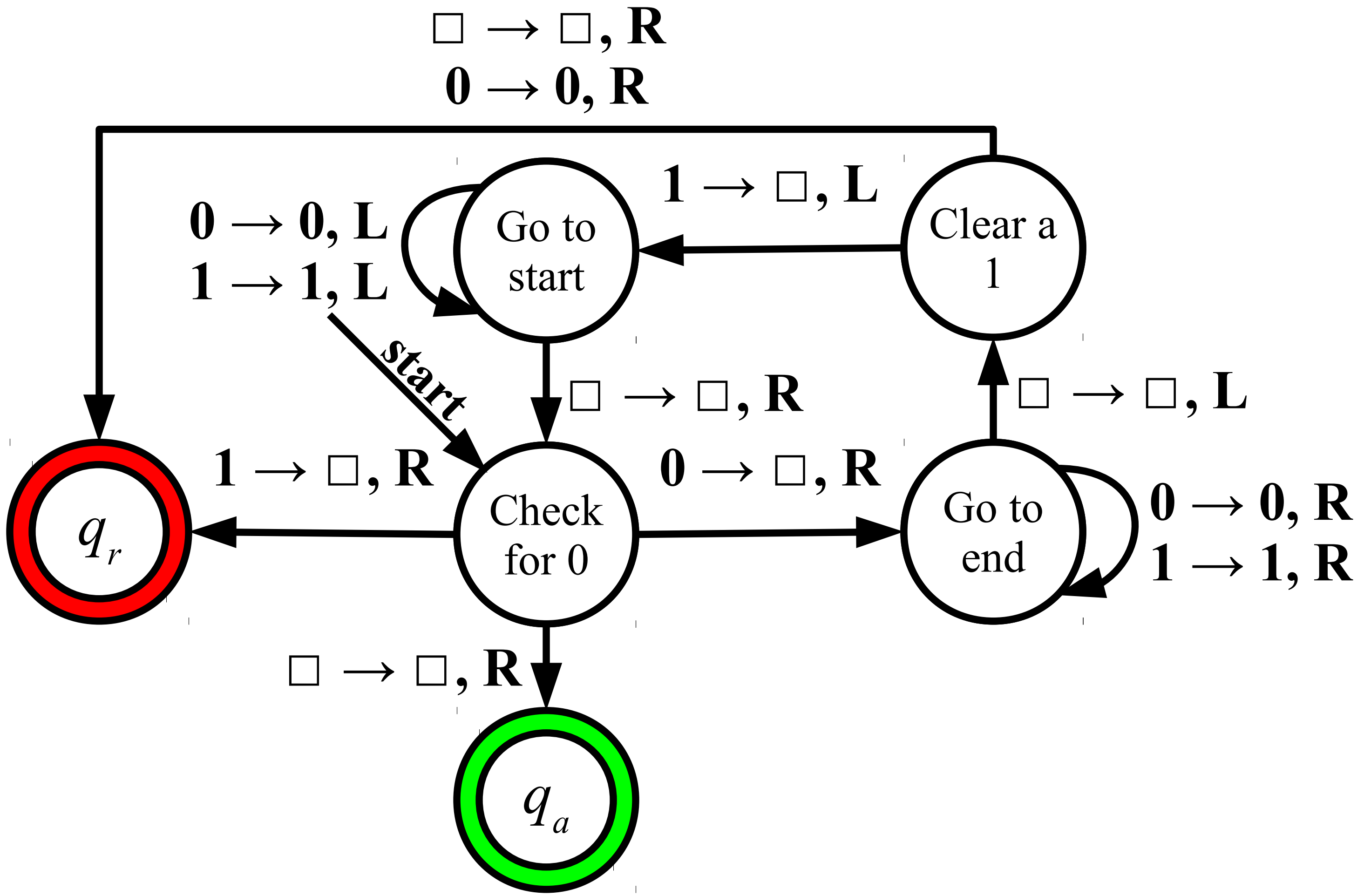
How to Turing machines compare with standard, run-of-the-mill computers?

Real and "Ideal" Computers

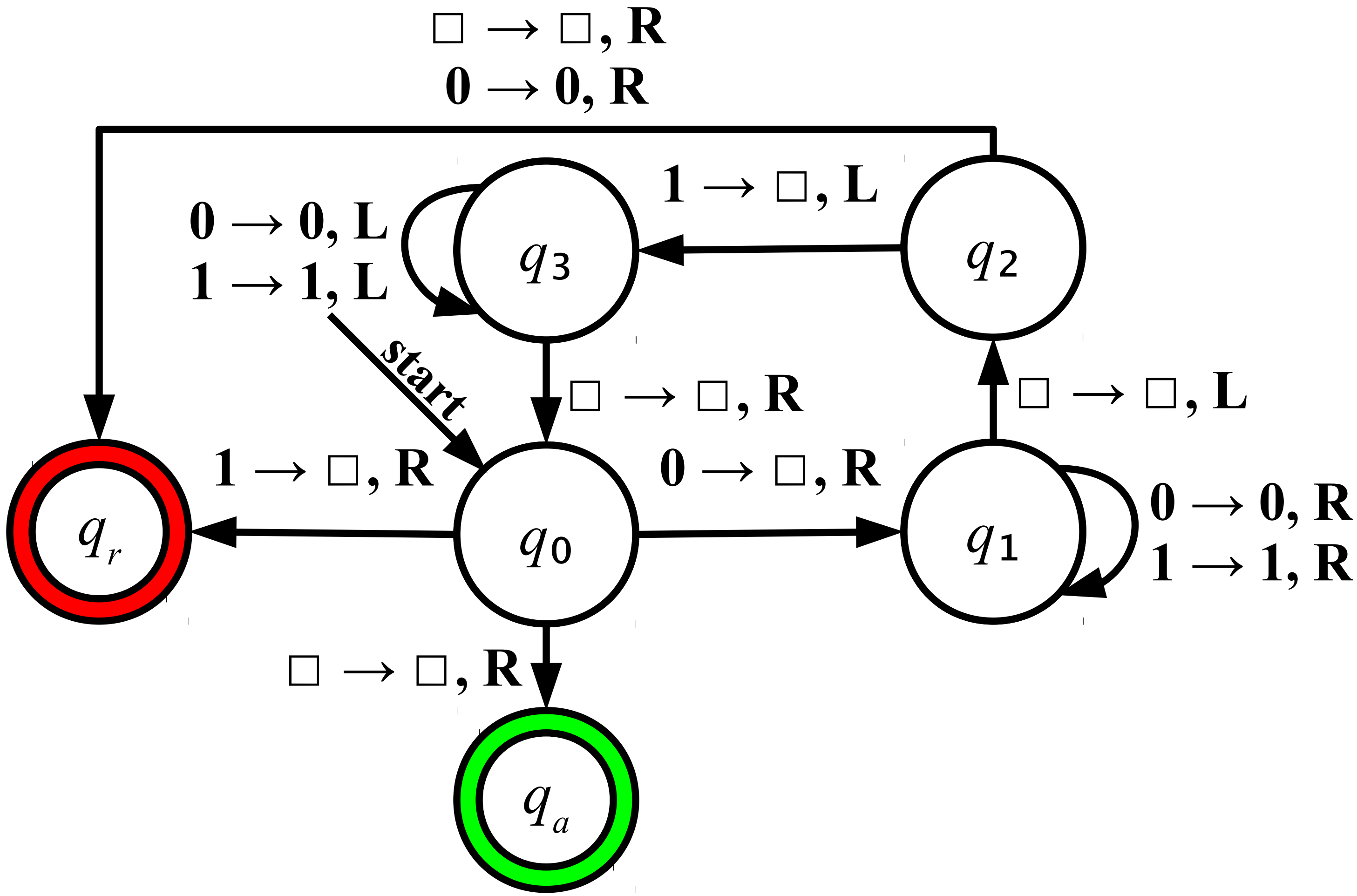
- A real computer has memory limitations: you have a finite amount of RAM, a finite amount of disk space, etc.
  - This makes them equivalent to finite automata.
- However, as computers get more and more powerful, the amount of memory available keeps increasing.
- An ***idealized computer*** is like a regular computer, but with unlimited RAM and disk space. It functions just like a regular computer, but never runs out of memory.

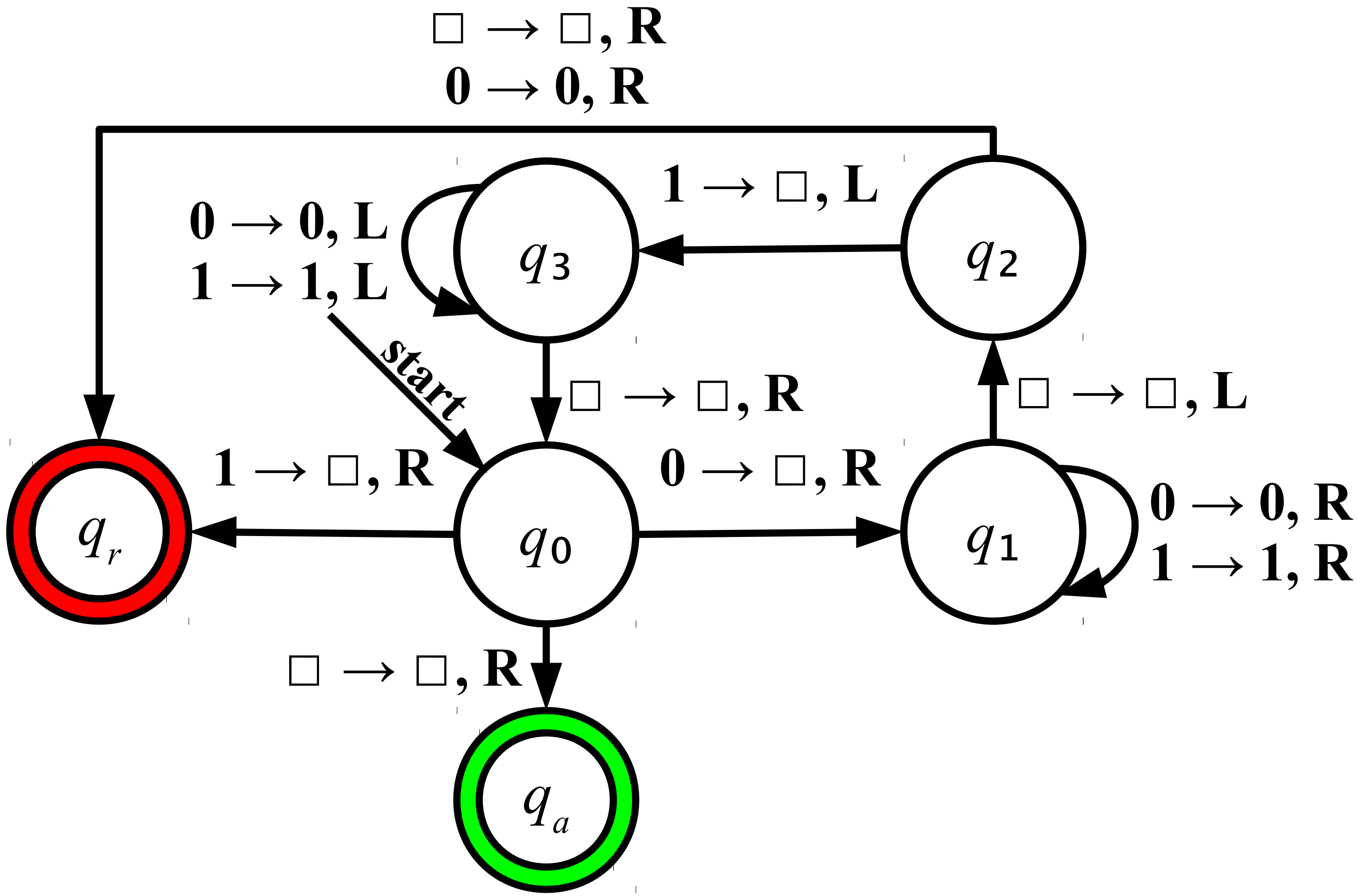
Claim 1: Idealized computers can simulate Turing machines.

"Anything that can be done with a TM can also be done with an unbounded-memory computer."

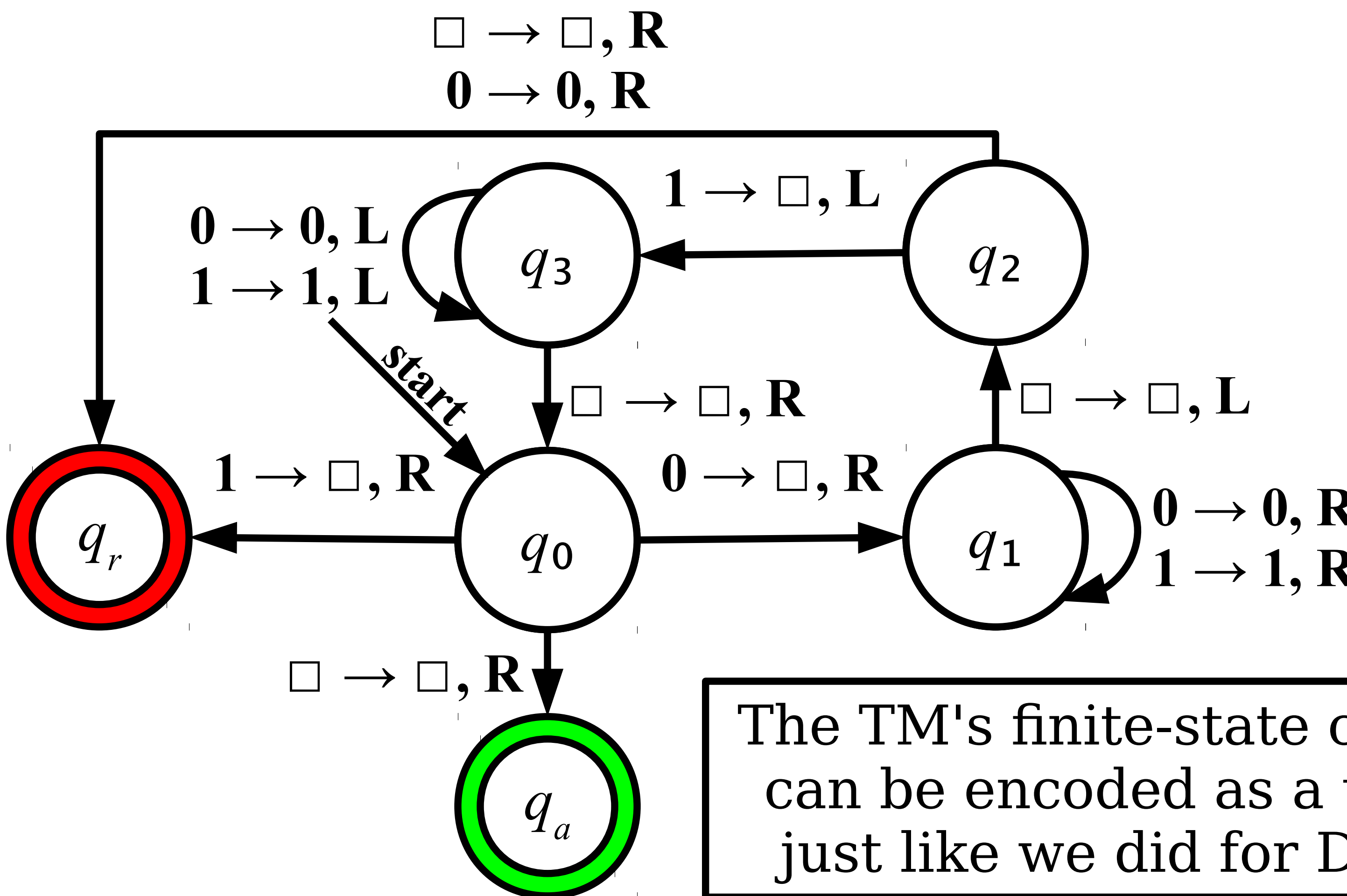


□ → □, R
0 → 0, R
1 → □, L
0 → 0, L
1 → 1, L
Go to start
Clear a 1
start
□ → □, R
□ → □, L
1 → □, R
0 → □, R
q_r
Check for 0
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_a


□ → □, R
0 → 0, R
1 → □, L
0 → 0, L
1 → 1, L
q_3
q_2
start
□ → □, R
□ → □, L
1 → □, R
0 → □, R
q_r
q_0
q_1
0 → 0, R
1 → 1, R
□ → □, R
q_a



	0			1			□		
q_0	q_1	□	R	q_r	□	R	q_a	□	R
q_1	q_1	0	R	q_1	1	R	q_2	□	L
q_2	q_r	0	R	q_3	□	L	q_r	□	R
q_3	q_3	0	L	q_3	1	L	q_0	□	R



The TM's finite-state control can be encoded as a table, just like we did for DFAs.

	0			1			□		
q_0	q_1	□	**R**	q_r	□	**R**	q_a	□	**R**
q_1	q_1	**0**	**R**	q_1	**1**	**R**	q_2	□	**L**
q_2	q_r	**0**	**R**	q_3	□	**L**	q_r	□	**R**
q_3	q_3	**0**	**L**	q_3	**1**	**L**	q_0	□	**R**

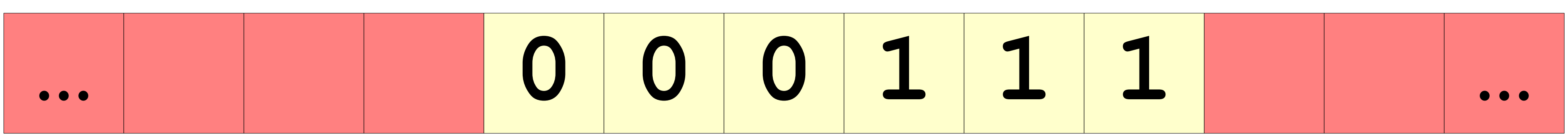
Simulating a TM

- To simulate a TM, the computer would need to be able to keep track of
  - the finite-state control,
  - the current state,
  - the position of the tape head, and
  - the tape contents.
- The tape contents are infinite, but that's because there are infinitely many blanks on both sides.
- We only need to store the “interesting” part of the tape (the parts that have been read from or written to so far.)

…				0	0	0	1	1	1			…

Simulating a TM

- To simulate a TM, the computer would need to be able to keep track of
  - the finite-state control,
  - the current state,
  - the position of the tape head, and
  - the tape contents.
- The tape contents are infinite, but that's because there are infinitely many blanks on both sides.
- We only need to store the “interesting” part of the tape (the parts that have been read from or written to so far.)



Claim 2: Turing machines can simulate idealized computers.

"Anything that can be done with an unbounded-memory computer can be done with a TM."

What We've Seen

- TMs can
  - implement loops (basically, every TM we've seen).
  - make function calls (subroutines).
  - keep track of natural numbers (written in unary or in decimal on the tape).
  - perform elementary arithmetic (equality testing, multiplication, addition, increment, decrement, etc.).
  - perform if/else tests (different transitions based on different cases).

What Else Can TMs Do?

- Maintain variables.
  - Have a dedicated part of the tape where the variables are stored.
  - We've seen this before: you can kinda sorta think of our machine for $\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$ as checking if two variables are equal.
- Maintain arrays and linked structures.
  - Divide the tape into different regions corresponding to memory locations.
  - Represent arrays and linked structures by keeping track of the ID of one of those regions.

A CS107 Perspective

- Internally, computers execute by using basic operations like
  - simple arithmetic,
  - memory reads and writes,
  - branches and jumps,
  - register operations,
  - etc.
- Each of these are simple enough that they could be simulated by a Turing machine.

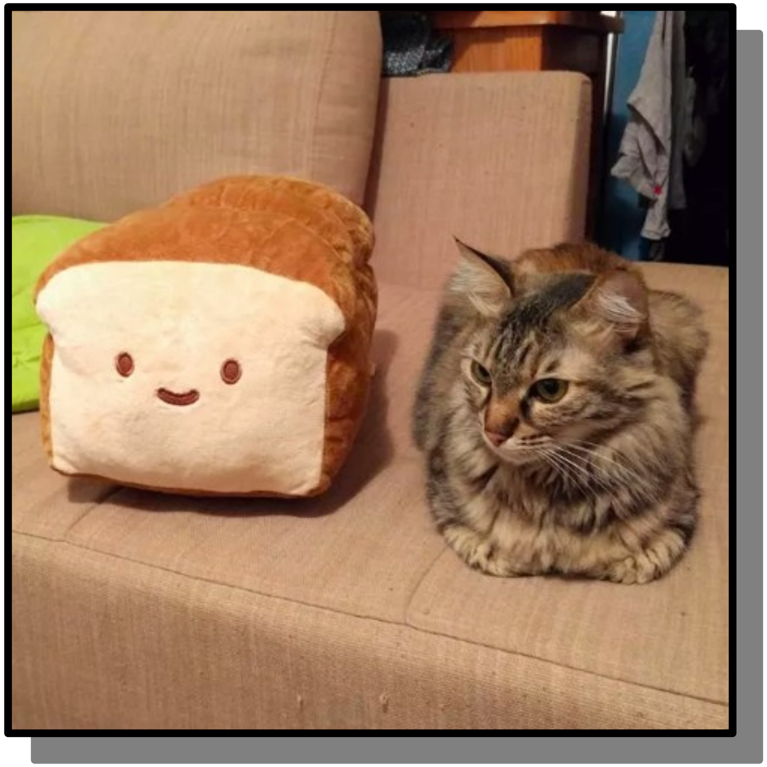
A Leap of Faith

- It may require a leap of faith, but anything you can do with a computer (excluding randomness and user input) can be performed by a Turing machine.
- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.
- We're going to take this as an article of faith in CS103. If you’re curious for more details, come talk to me after class.

Wait, You're Saying a TM Can Do…

"cat pictures?"

Sure! A picture is just a 2D array of colors, and a color can be represented as a series of numbers.

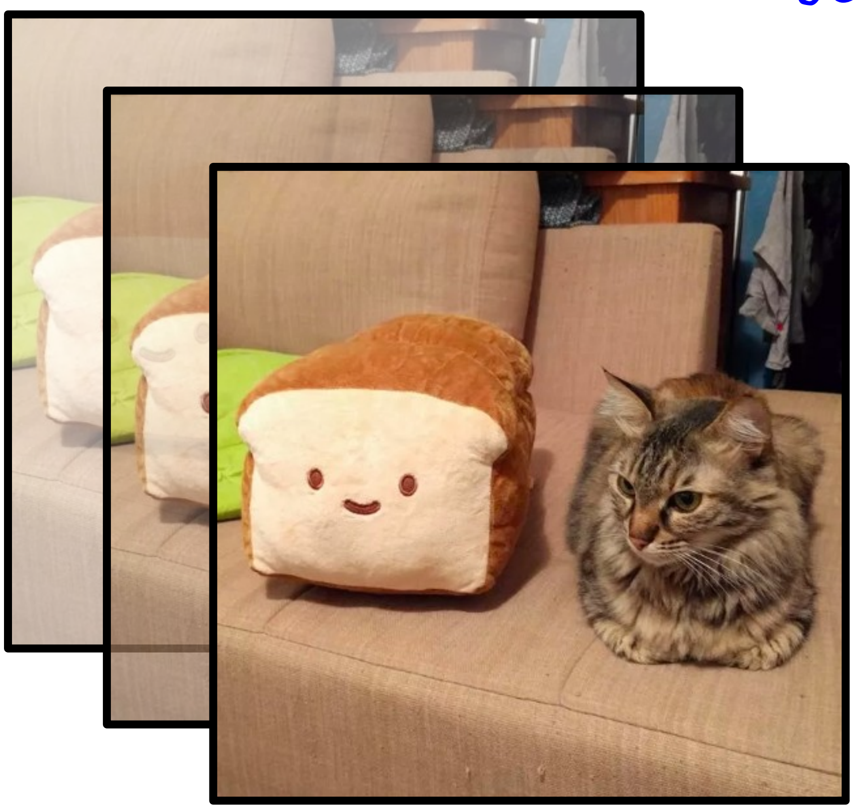


Wait, You're Saying a TM Can Do…

~~"cat pictures?"~~

"cat videos?"

If you think about it, a video is just a series of pictures!



Wait, You're Saying a TM Can Do…

"music?"

Yes! Write encodings of notes to play on the TM tape. Hook up a speaker device that reads the tape and makes sound.

"chat messages over the internet?"

Yes! View all networked computers as one gigantic machine.

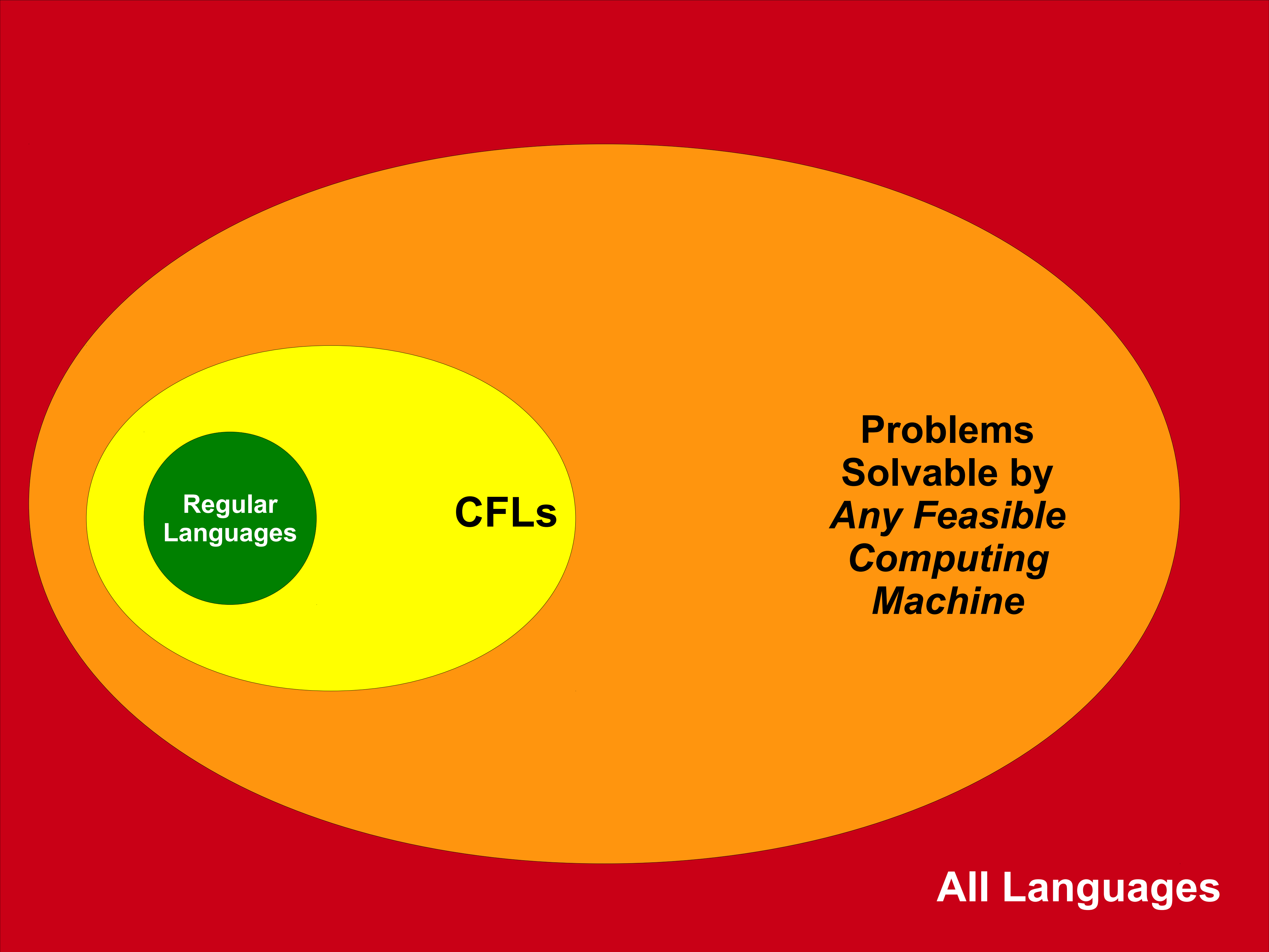
Just how powerful ***are*** Turing machines?

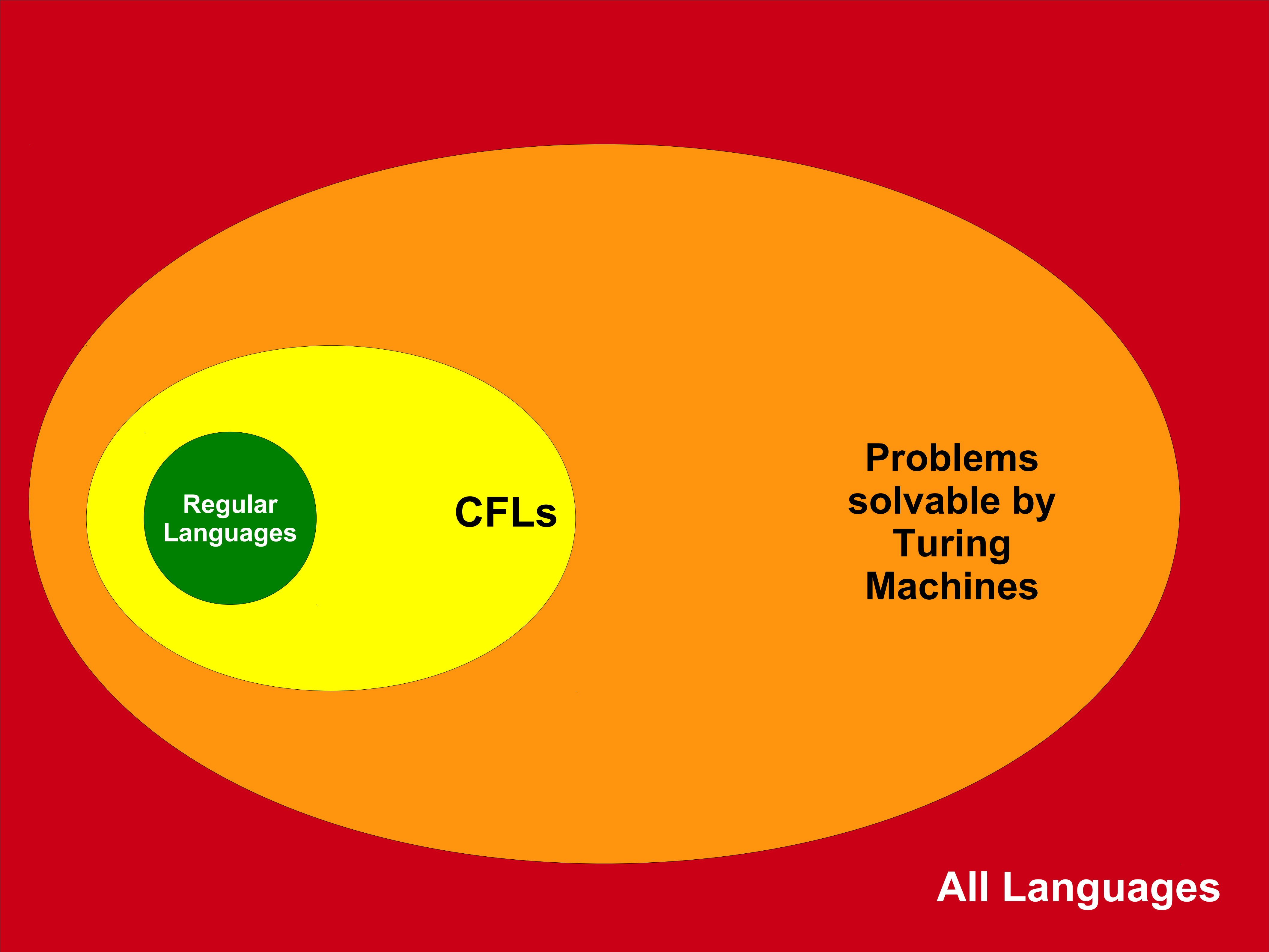
Effective Computation

- An ***effective method of computation*** is a form of computation with the following properties:
  - The computation consists of a set of steps.
  - There are fixed rules governing how one step leads to the next.
  - Any computation that yields an answer does so in finitely many steps.
  - Any computation that yields an answer always yields the correct answer.
- This is not a formal definition. Rather, it's a set of properties we expect out of a computational system.

The ***Church-Turing Thesis*** claims that

every effective method of computation is either equivalent to or weaker than a Turing machine.


Regular
Languages
CFLs
Problems
Solvable by
*Any Feasible
Computing
Machine*
All Languages


Regular Languages
CFLs
Problems solvable by Turing Machines
All Languages

TMs ≈ Computers

- Because Turing machines have the same computational powers as regular computers, we can (essentially) reason about Turing machines by reasoning about actual computer programs.
- Going forward, we're going to switch back and forth between TMs and computer programs based on whatever is most appropriate.
- In fact, our eventual proofs about the existence of impossible problems will involve a good amount of pseudocode. Stay tuned for details!

What problems can we solve with a computer?

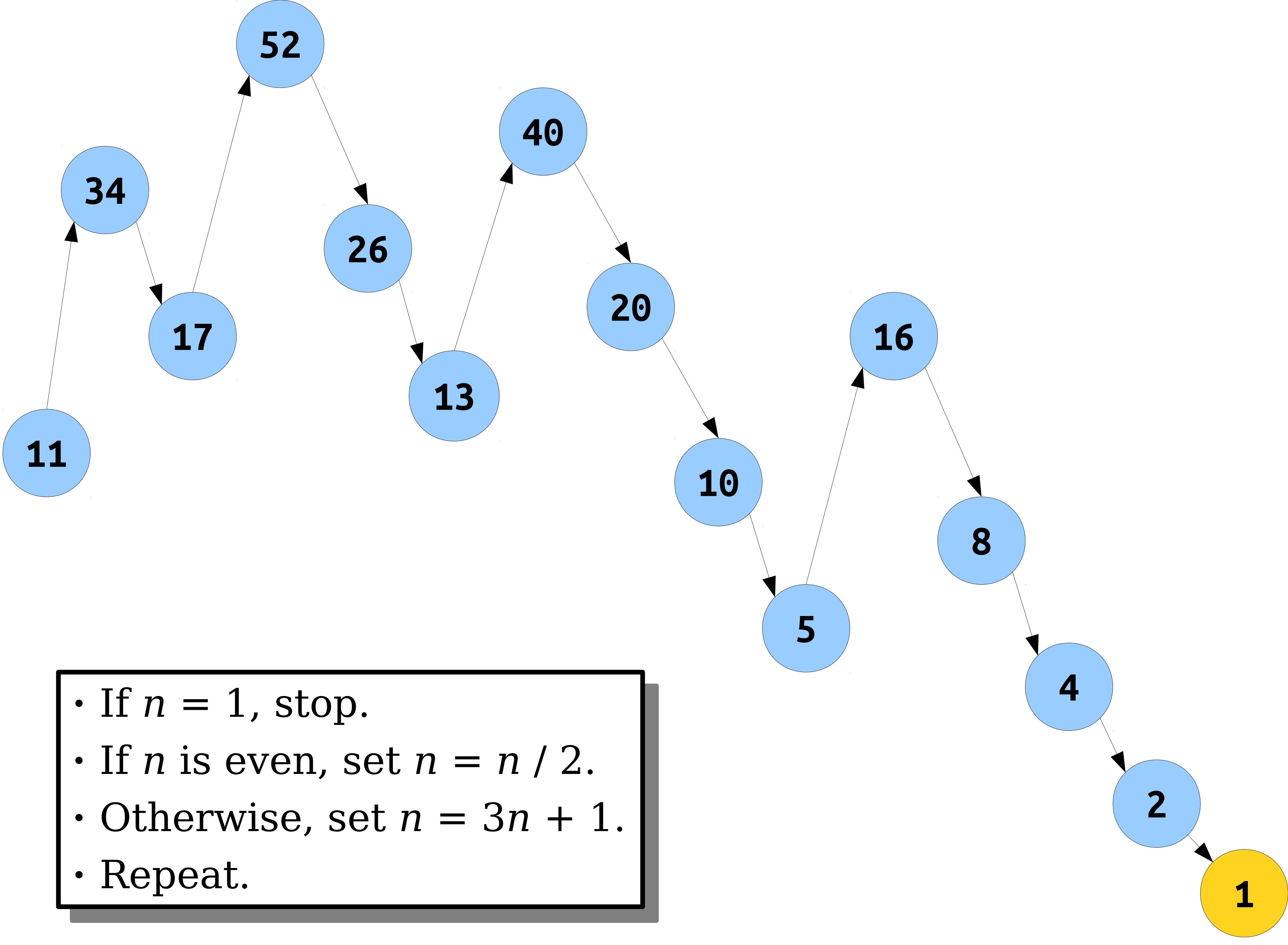
What kind
of
computer?

What problems can we solve with a computer?

What does it mean to "solve" a problem?

The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
  - If $n = 1$, you are done.
  - If n is even, set $n = n / 2$.
  - Otherwise, set $n = 3n + 1$.
  - Repeat.
- ***Question:*** Given a number n, does this process terminate?



- If $n = 1$, stop.
- If n is even, set $n = n / 2$.
- Otherwise, set $n = 3n + 1$.
- Repeat.

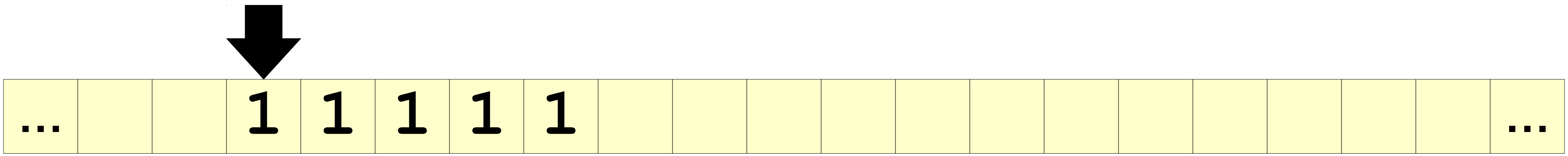
The Hailstone Sequence

- Let $\Sigma = \{\mathbf{1}\}$ and consider the language

$$L = \{ \mathbf{1}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}.$$

- Could we build a TM for L?

The Hailstone Turing Machine



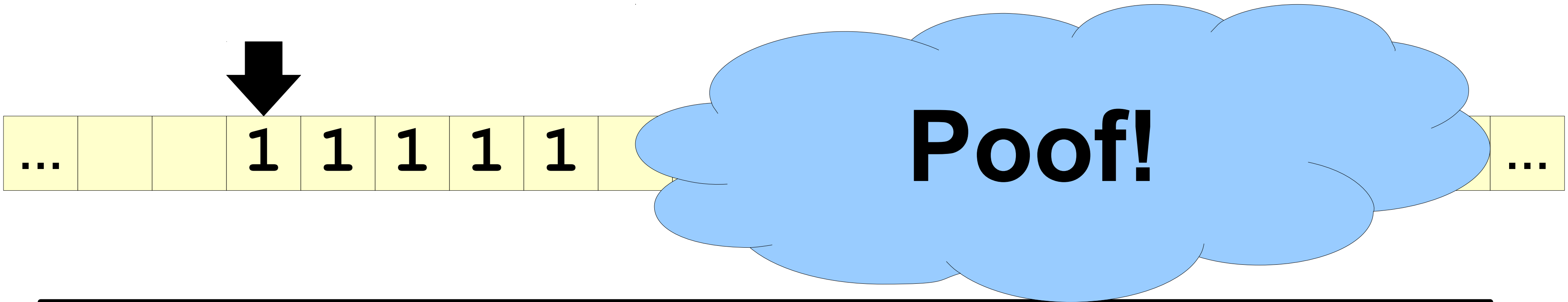
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



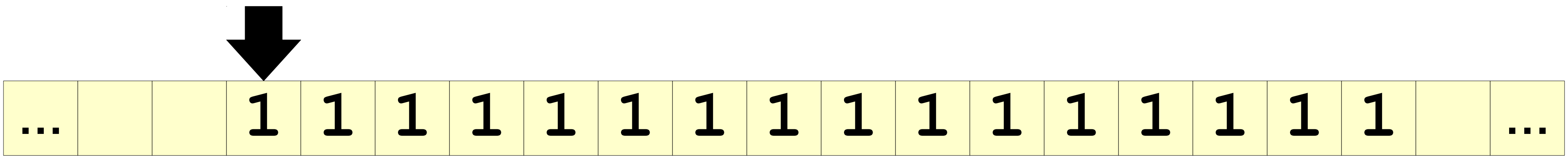
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



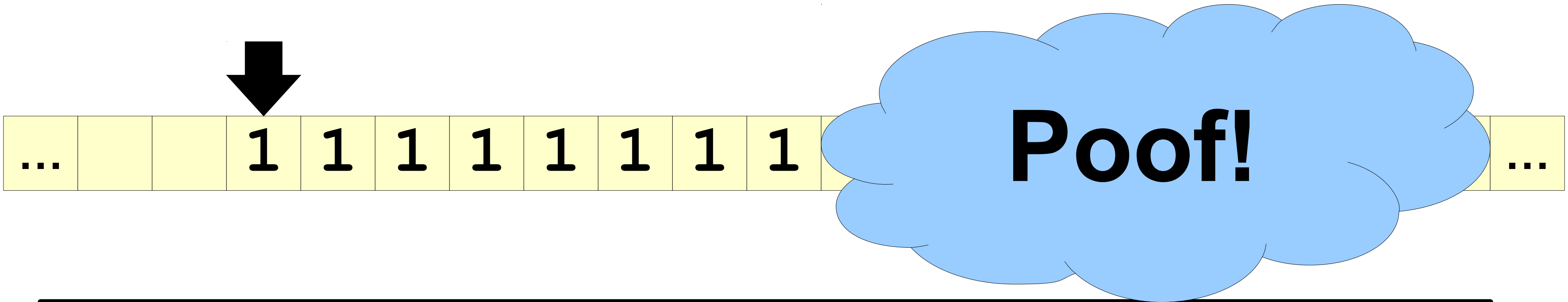
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



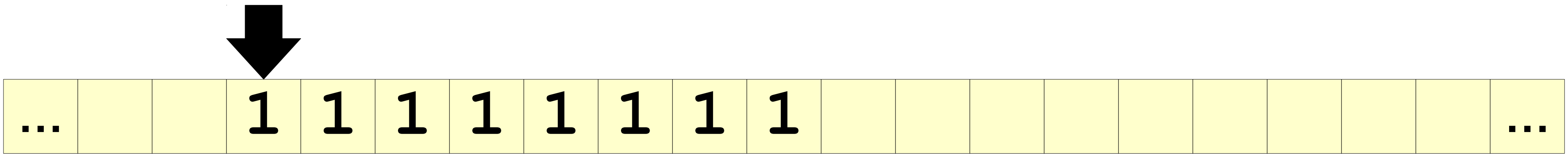
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



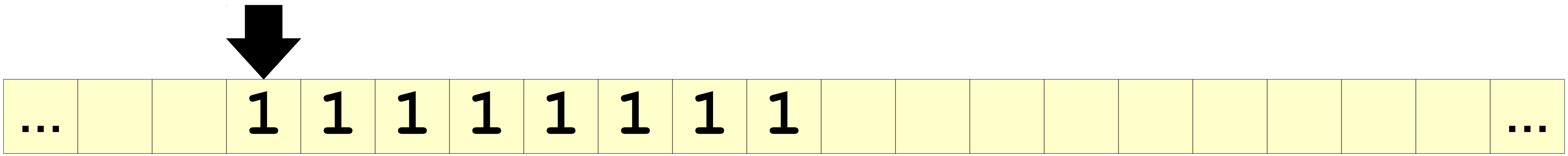
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



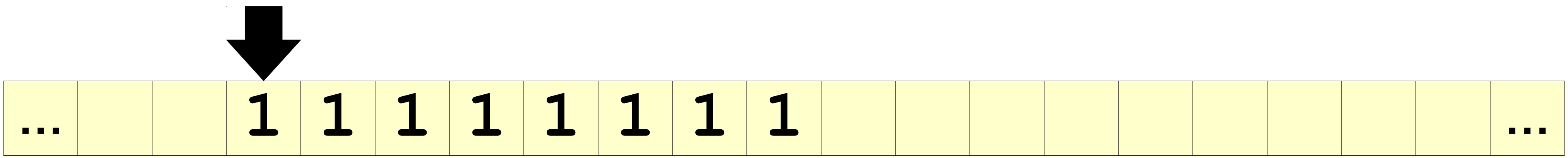
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



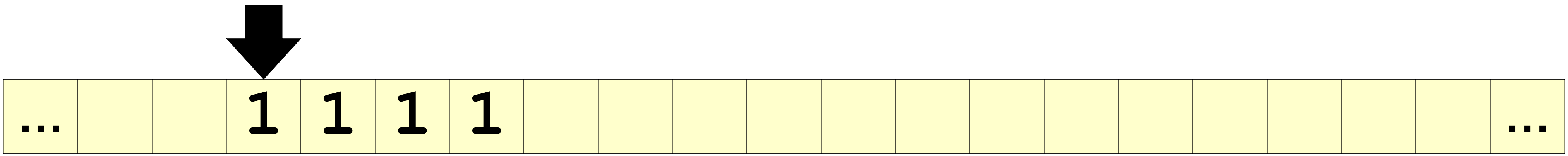
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



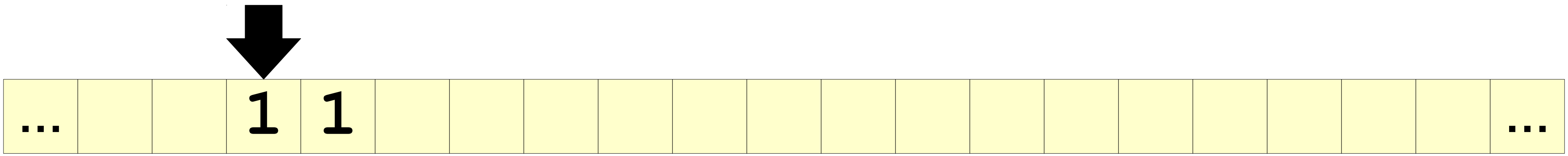
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



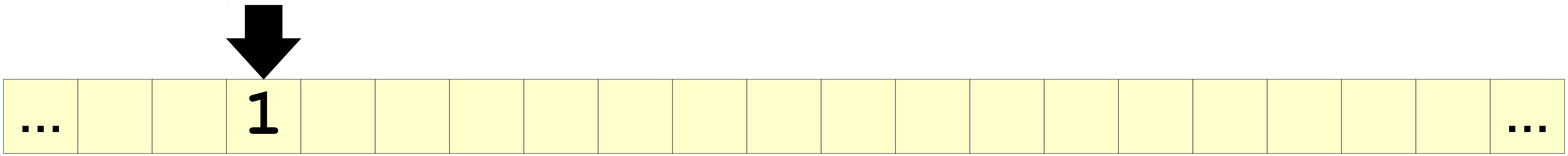
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

Does this Turing machine accept all nonempty strings?

The Collatz Conjecture

- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, no one knows whether the TM described in the previous slides will always stop running!
- The conjecture (unproven claim) that this always terminates is called the ***Collatz Conjecture***.

The Collatz Conjecture

“Mathematics may not be ready for such problems.” - Paul Erdős

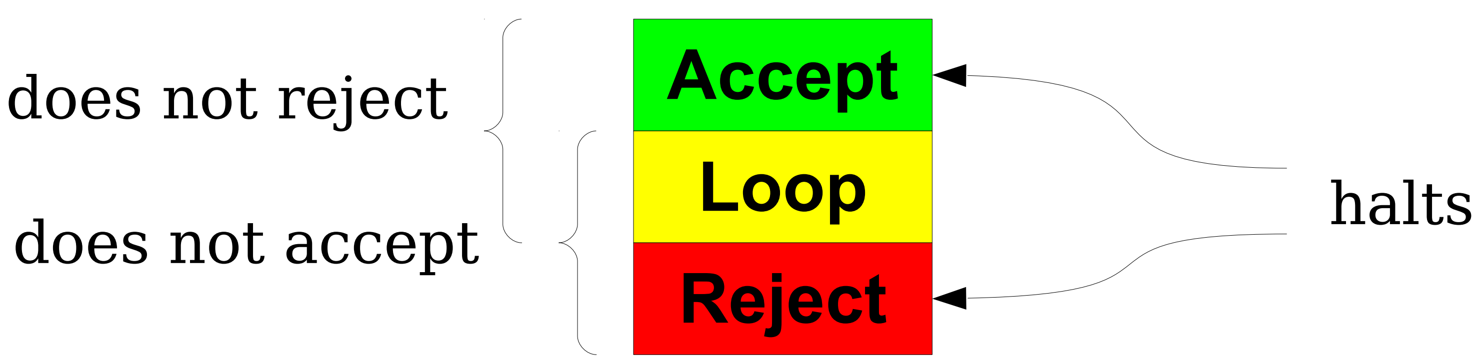
The fact that the Collatz Conjecture is unresolved is useful later on for building intuitions. Keep this in mind!

An Important Observation

- Unlike finite automata, which automatically halt after all the input is read, TMs keep running until they explicitly enter an accept or reject state.
- As a result, it's possible for a TM to run forever without accepting or rejecting.
- This leads to several important questions:
  - How do we formally define what it means to build a TM for a language?
  - What implications does this have about problem-solving?

Very Important Terminology

- Let M be a Turing machine and let w be a string.
- M ***accepts w*** if it enters an accept state when run on w.
- M ***rejects w*** if it enters a reject state when run on w.
- M ***loops infinitely on w*** (or just ***loops on w***) if when run on w it enters neither an accept nor a reject state.
- M ***does not accept w*** if it either rejects w or loops infinitely on w.
- M ***does not reject w*** w if it either accepts w or loops on w.
- M ***halts on w*** if it accepts w or rejects w.



The Language of a TM

- The language of a Turing machine M, denoted $\mathcal{L}(M)$, is the set of all strings that M accepts:

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

- For any $w \in \mathcal{L}(M)$, M accepts w.
- For any $w \notin \mathcal{L}(M)$, M does not accept w.
  - M might reject w, or it might loop on w.
- A language is called ***recognizable*** if it is the language of some TM.
- A TM M where $\mathcal{L}(M) = L$ is called a ***recognizer*** for L.
- Notation: the class **RE** is the set of all recognizable languages.

$$L \in \mathbf{RE} \quad \leftrightarrow \quad L \text{ is recognizable}$$

What do you think? Does that correspond to what you think it means to solve a problem?