

Turing Machines

Part Two

Outline for Today

- **TM Subroutines**
 - Combining multiple TMs together.
- **How Powerful are TMs?**
 - How do TMs relate back to computers?
- **The Church-Turing Thesis**
 - What is the maximum scope of computing power?
- **R and RE Languages**
 - What does it mean for a TM to solve a problem?

Recap from Last Time

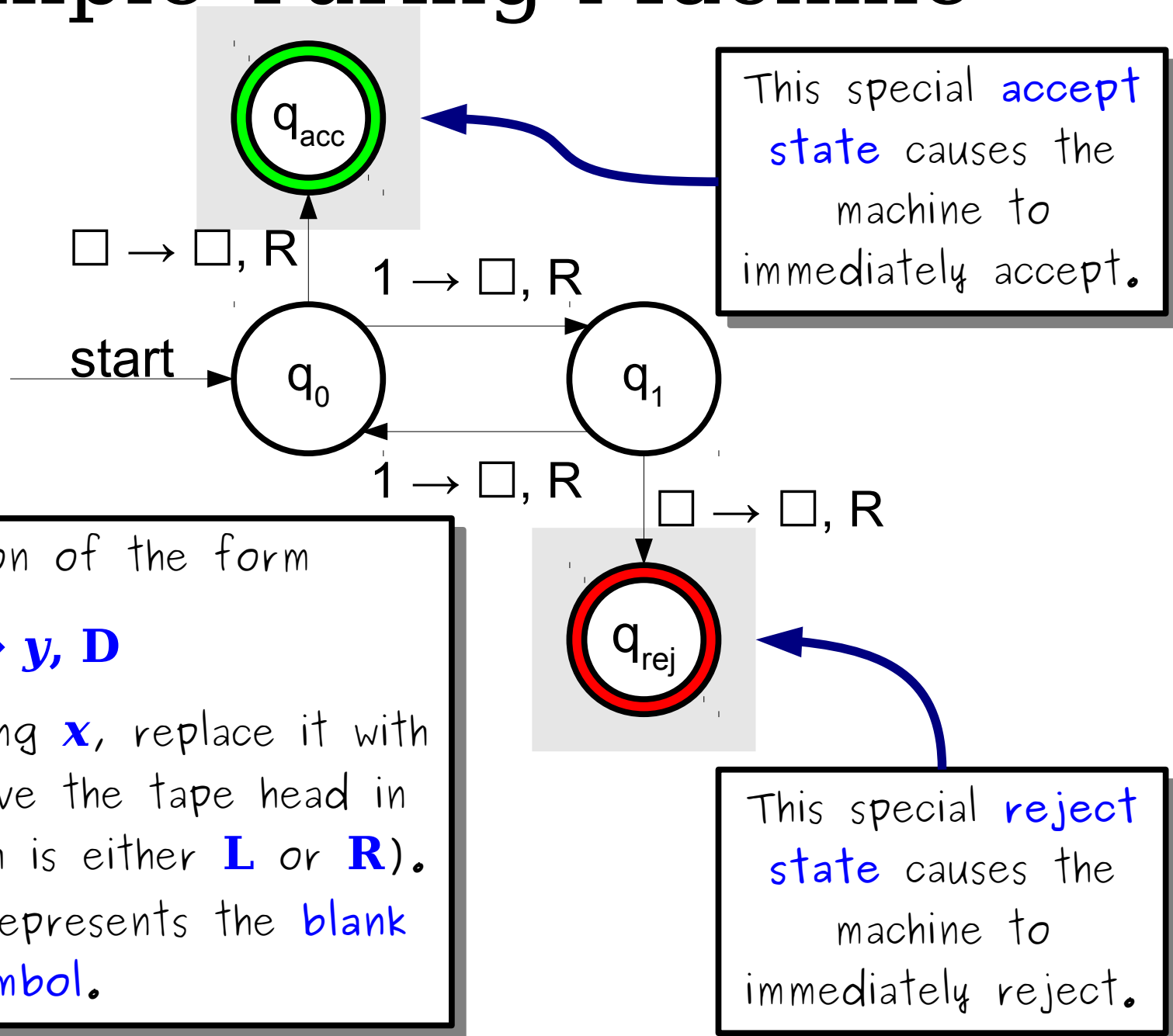
The Turing Machine

- 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 $\Sigma \subseteq \Gamma$. 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.

A Simple Turing Machine

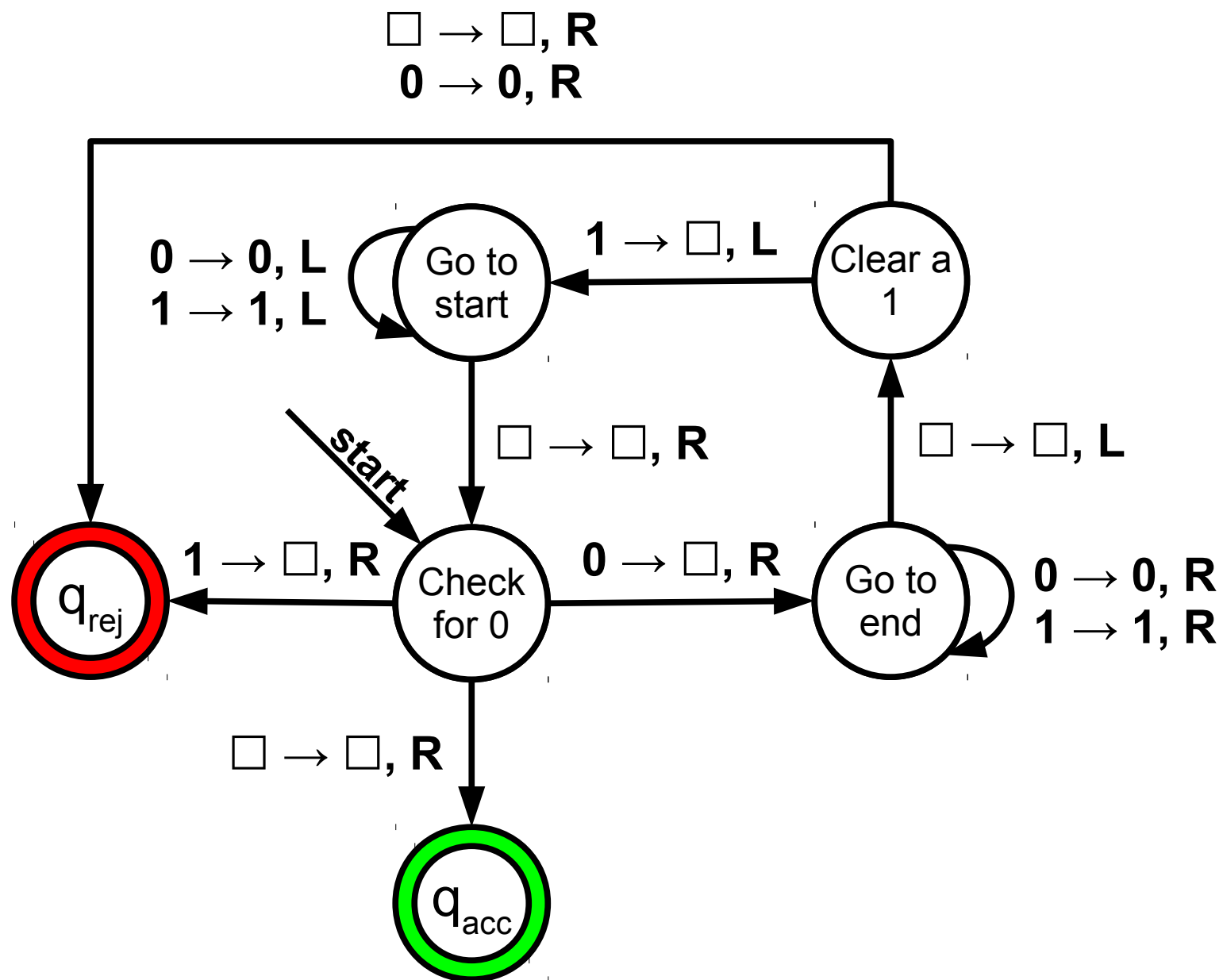


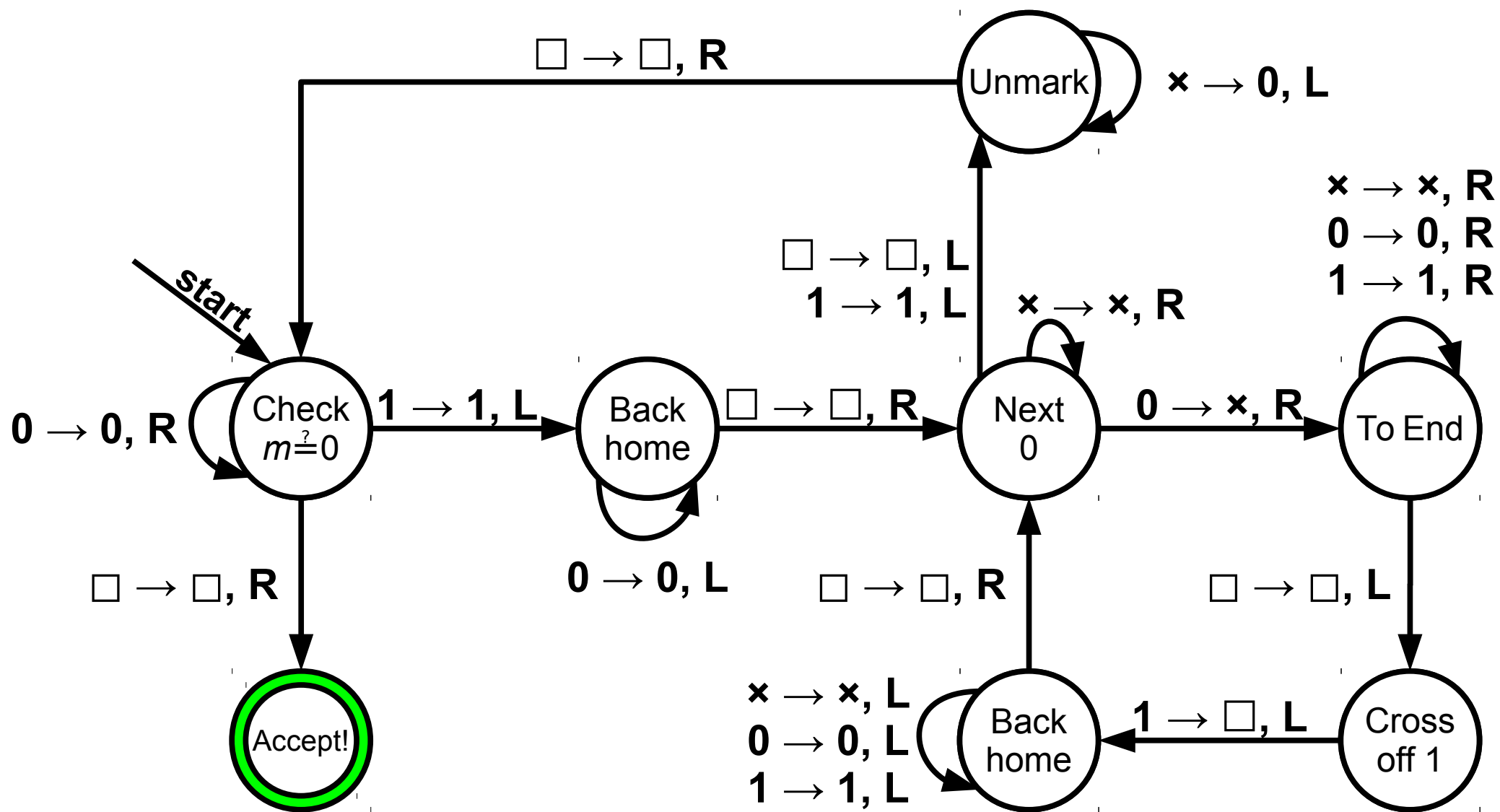
Each transition of the form

$x \rightarrow y, D$

means "upon reading **x** , replace it with symbol **y** and move the tape head in direction **D** (which is either **L** or **R**).

The symbol \square represents the **blank symbol**.





TM Subroutines

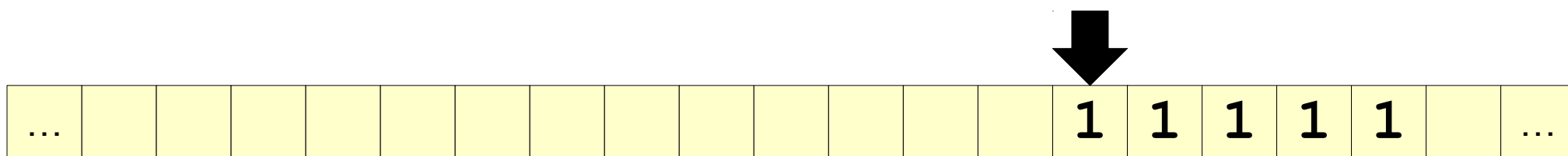
A Final TM Design

- A natural number $n \geq 2$ is called **composite** if n can be written as $p \cdot q$ for natural numbers p and q , neither of which is 1.
- Consider the following language over the alphabet $\Sigma = \{1\}$:
 - $L = \{1^n \mid n \in \mathbb{N} \text{ and } n \text{ is composite} \}$
- Is this language regular?
- How might we design a TM for this language?

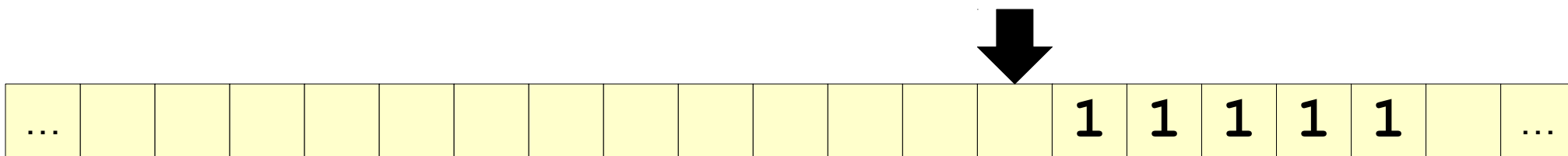
Our Approach

- A number n is composite if it can be written as $p \cdot q$ for some natural numbers p and q , neither of which is 1.
- Equivalently, n is composite if it's a multiple of some number other than 1 or n .
- We can check this by seeing if n is a multiple of 2, 3, 4, ..., or $n - 1$.
 - If so, it's composite.
 - Otherwise, it's not composite.

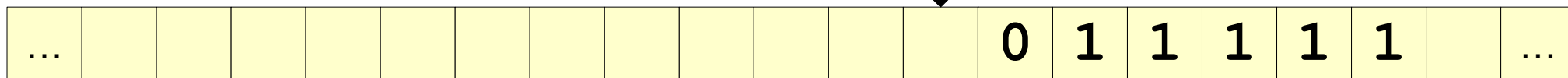
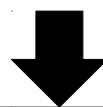
A Sketch of the TM



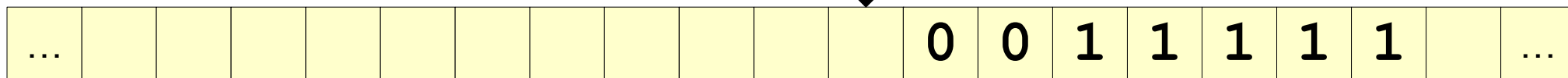
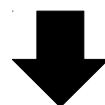
A Sketch of the TM



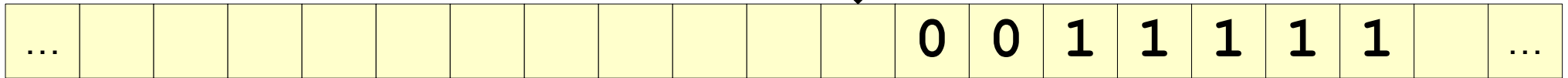
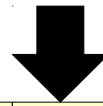
A Sketch of the TM



A Sketch of the TM

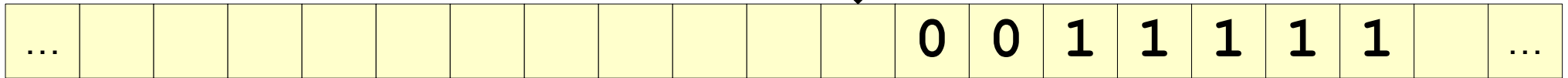
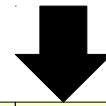


A Sketch of the TM



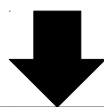
*Magically run our TM to check if the
number of 0's is the same as the
number of 1's*

A Sketch of the TM



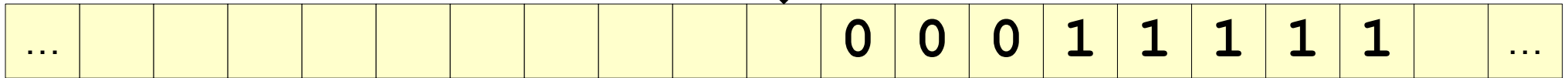
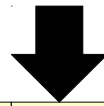
*Magically run our TM to check
if the number of 1's is a
multiple of the number of 0's*

A Sketch of the TM



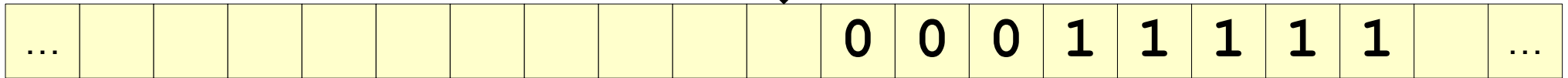
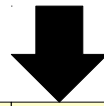
...											0	0	0	1	1	1	1	1		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	---	--	-----

A Sketch of the TM



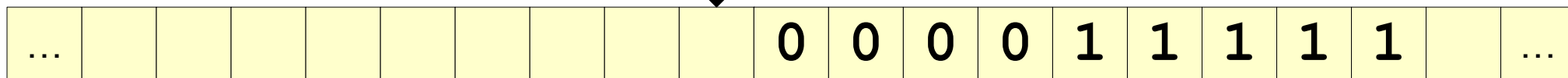
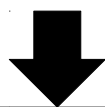
*Magically run our TM to check if the
number of 0's is the same as the
number of 1's*

A Sketch of the TM

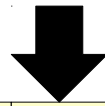


*Magically run our TM to check
if the number of 1's is a
multiple of the number of 0's*

A Sketch of the TM

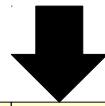


A Sketch of the TM



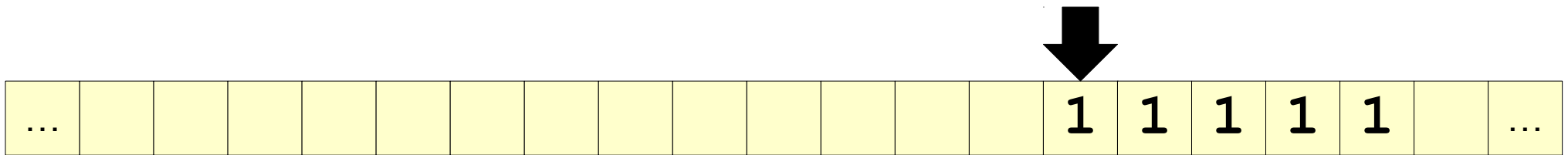
*Magically run our TM to check if the
number of 0's is the same as the
number of 1's*

A Sketch of the TM

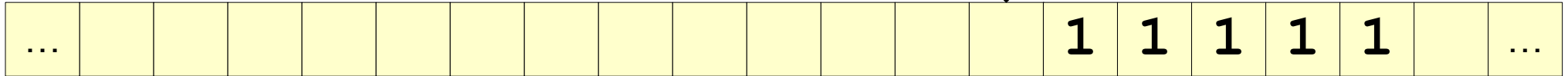
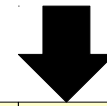


*Magically run our TM to check
if the number of 1's is a
multiple of the number of 0's*

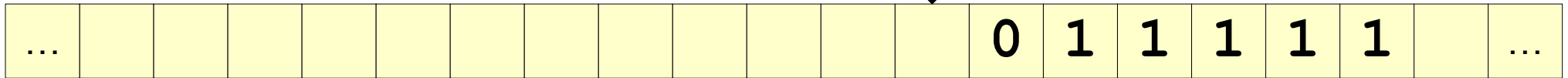
Our TM, in More Depth



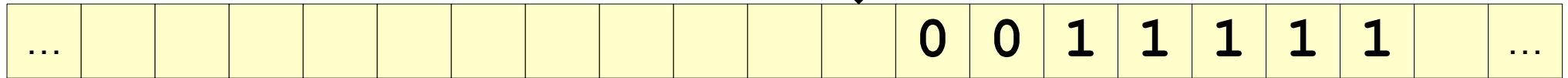
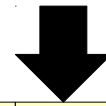
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth

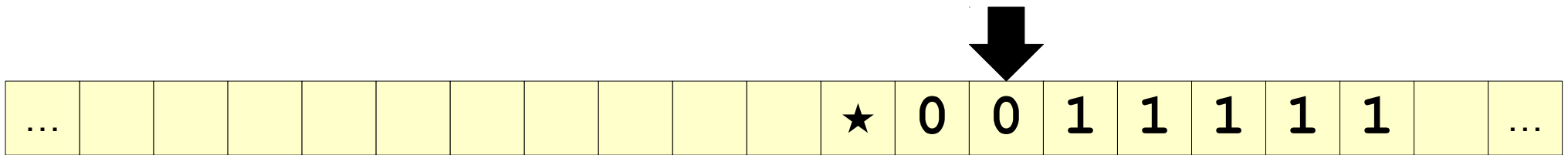


Our TM, in More Depth

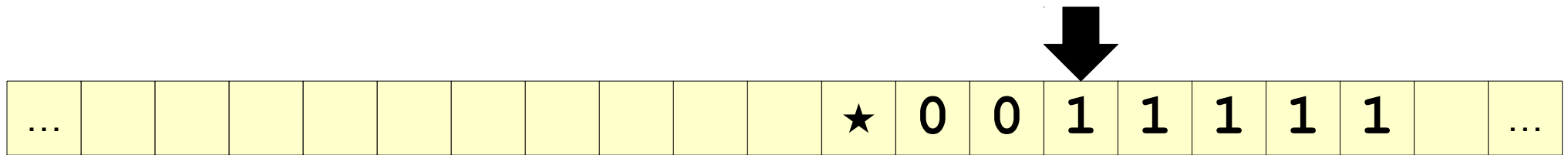


...											★	0	0	1	1	1	1	1		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	---	--	-----

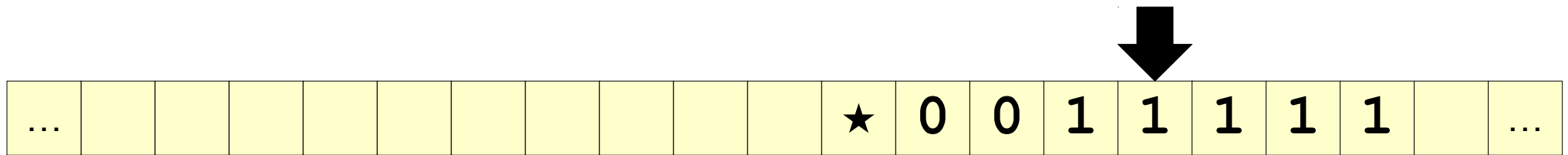
Our TM, in More Depth



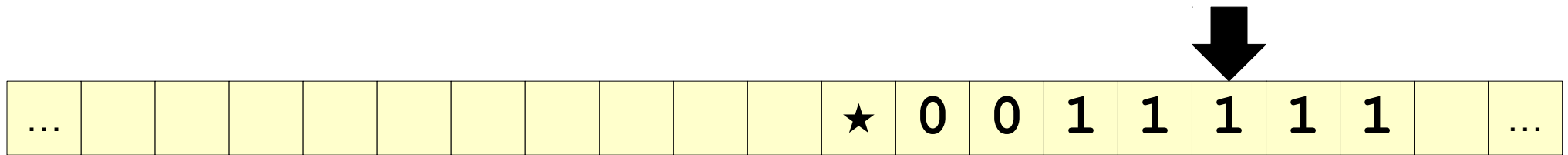
Our TM, in More Depth



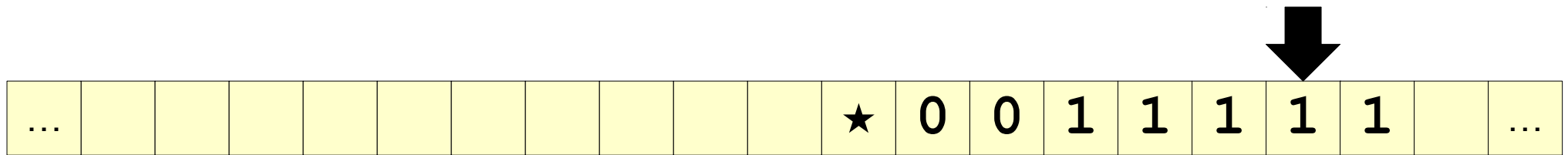
Our TM, in More Depth



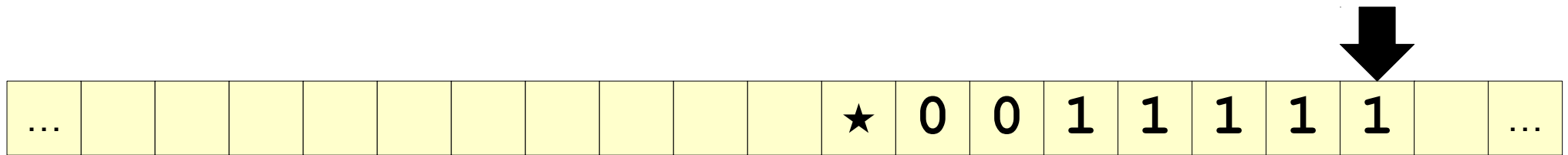
Our TM, in More Depth



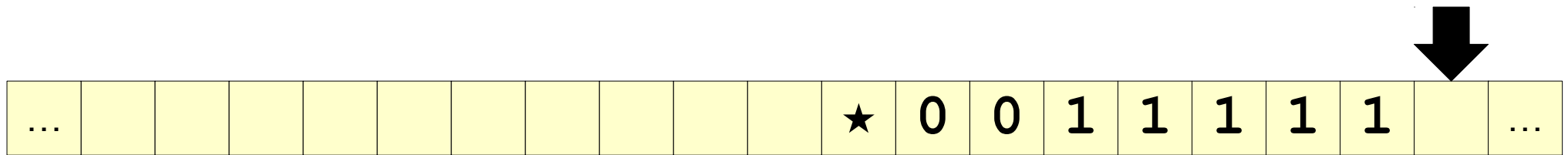
Our TM, in More Depth



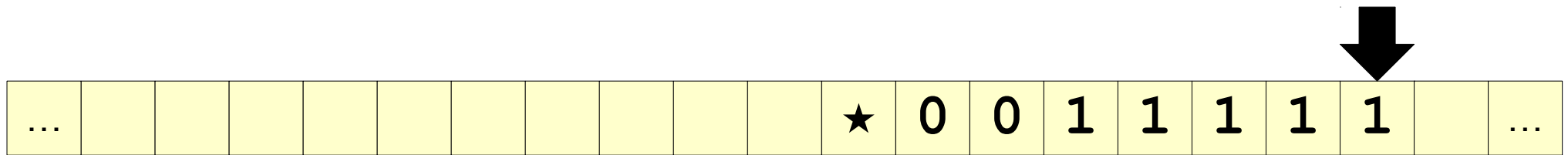
Our TM, in More Depth



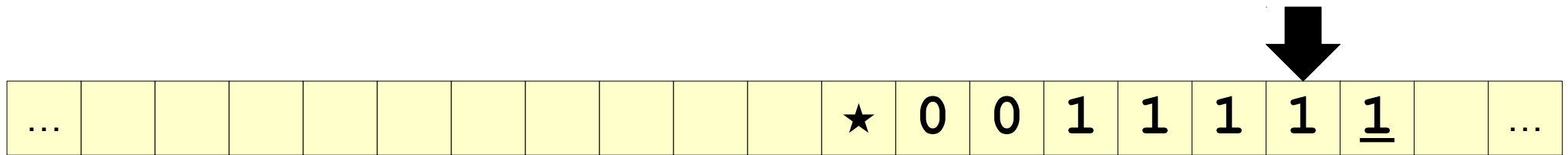
Our TM, in More Depth



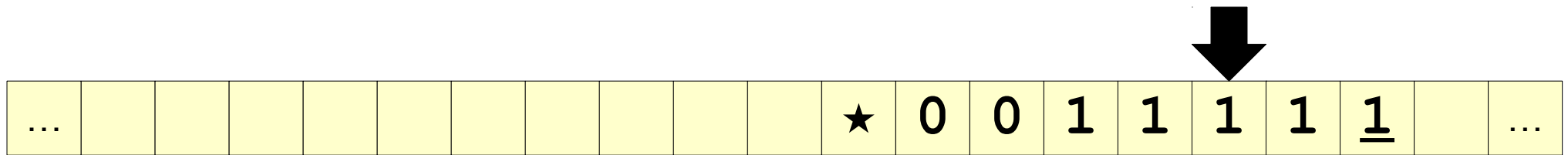
Our TM, in More Depth



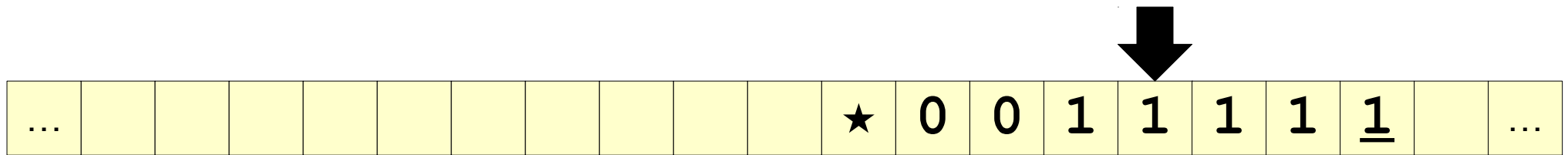
Our TM, in More Depth



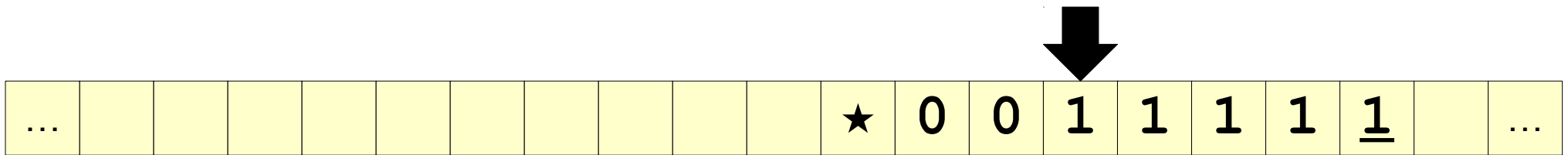
Our TM, in More Depth



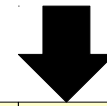
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth



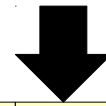
...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



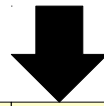
...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



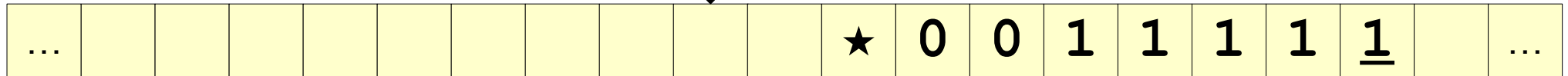
...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

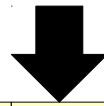


...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

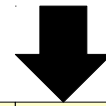


Our TM, in More Depth



...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



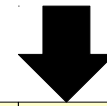
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



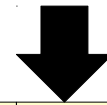
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



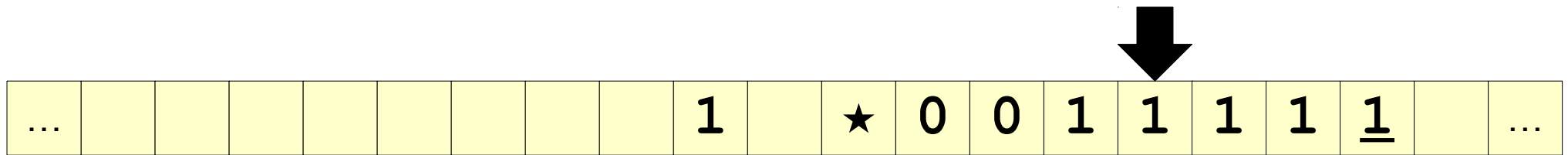
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

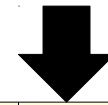


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

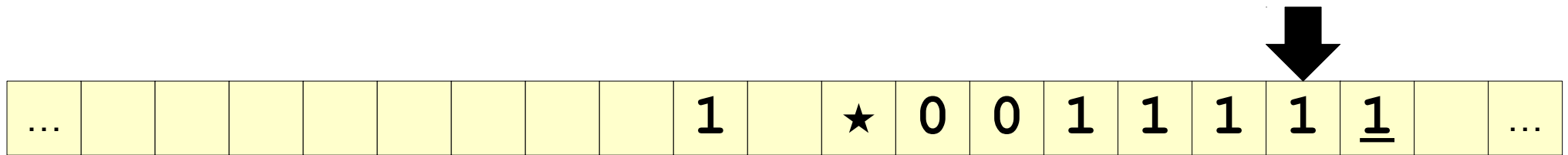


Our TM, in More Depth

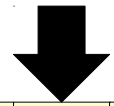


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

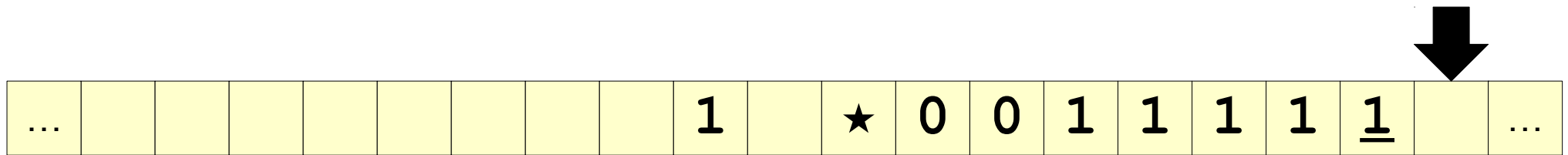


Our TM, in More Depth

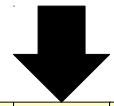


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

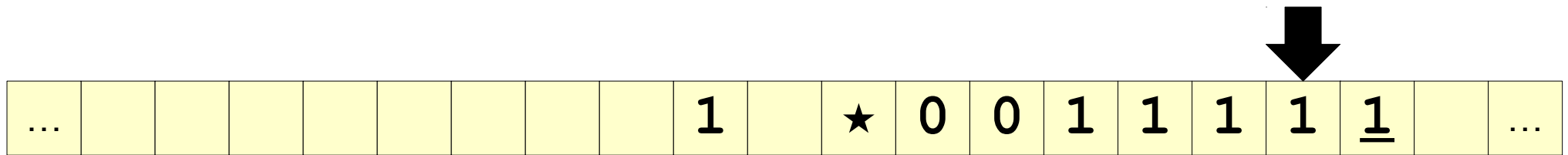


Our TM, in More Depth

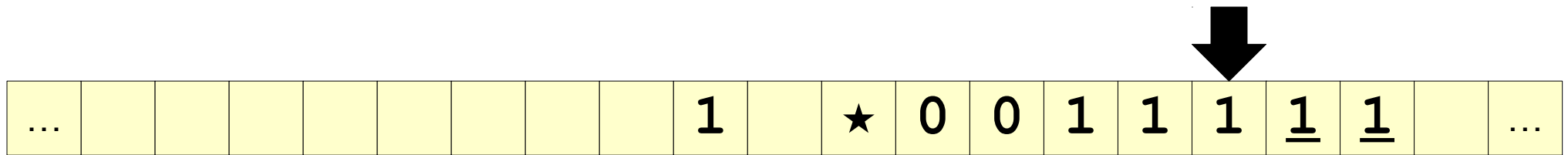


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

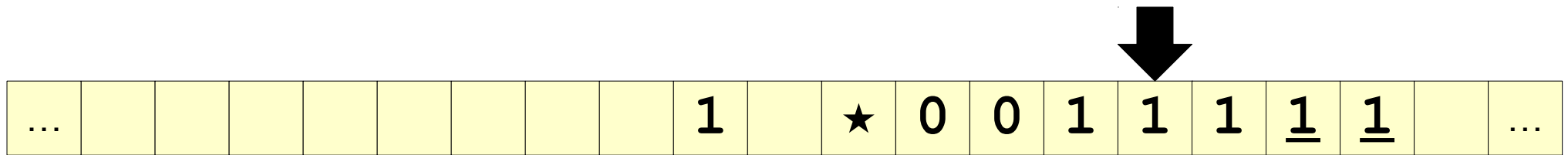
Our TM, in More Depth



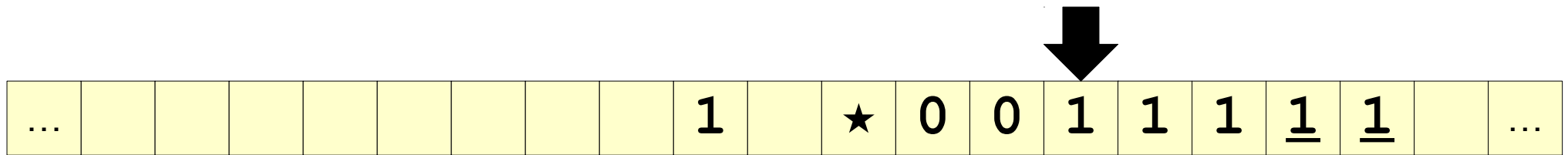
Our TM, in More Depth



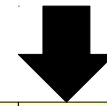
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth



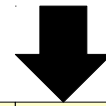
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



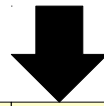
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



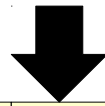
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



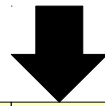
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth

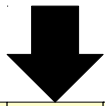


...								1	1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	---	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth

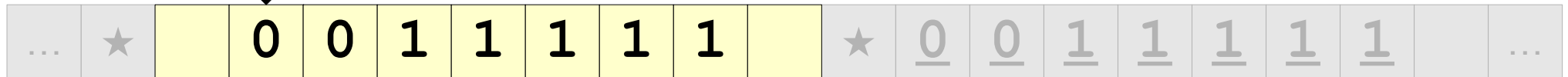
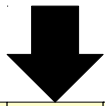


Our TM, in More Depth

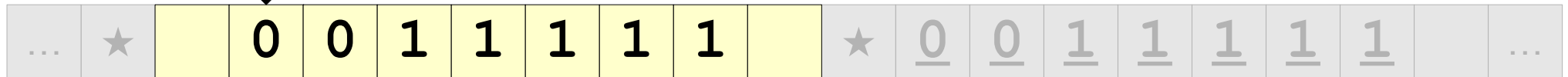
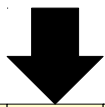


...	★		0	0	1	1	1	1	1		★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	---	---	---	---	---	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

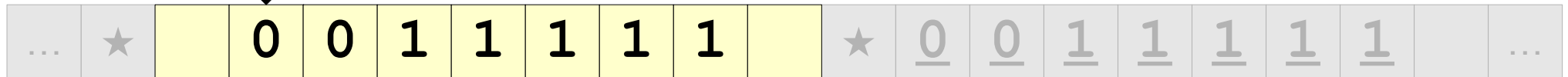
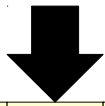


Our TM, in More Depth

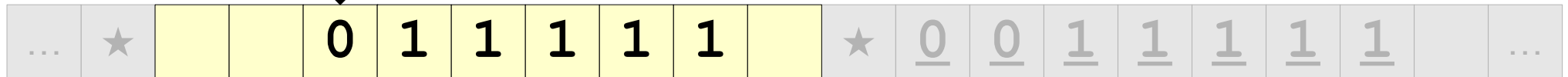


Our TM that checks whether a string is of the form 0^n1^n never uses more than one blank on either side of the string, so from its perspective it's running in a perfectly normal environment.

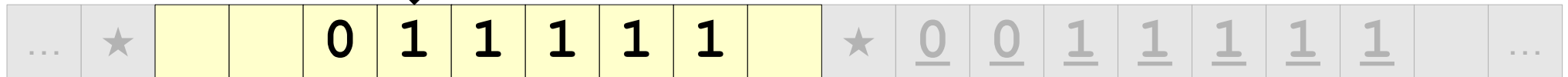
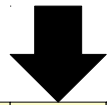
Our TM, in More Depth



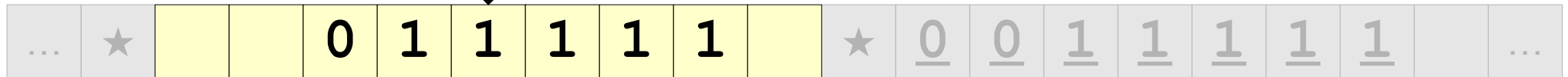
Our TM, in More Depth



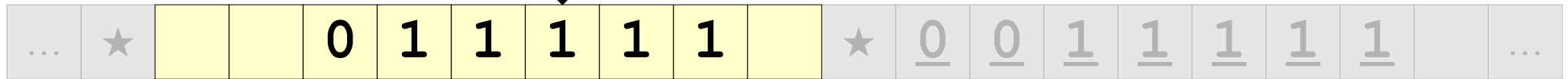
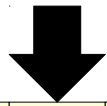
Our TM, in More Depth



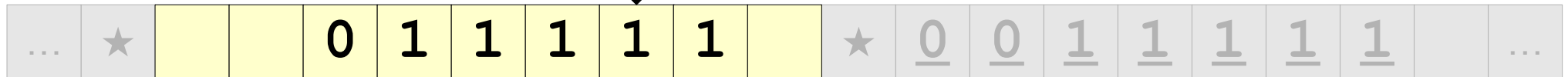
Our TM, in More Depth



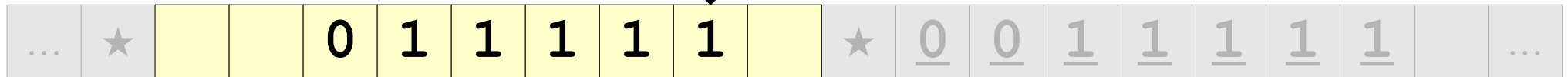
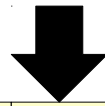
Our TM, in More Depth



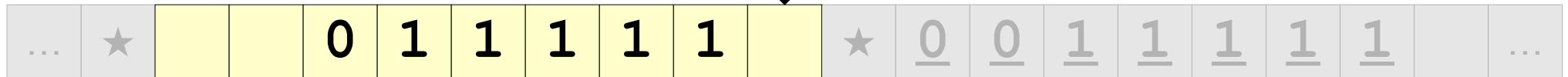
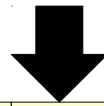
Our TM, in More Depth



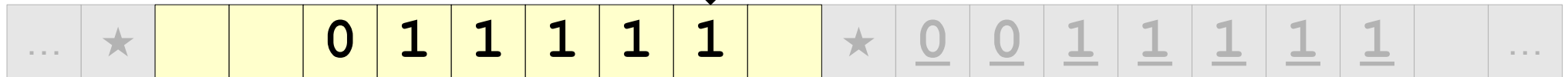
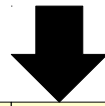
Our TM, in More Depth



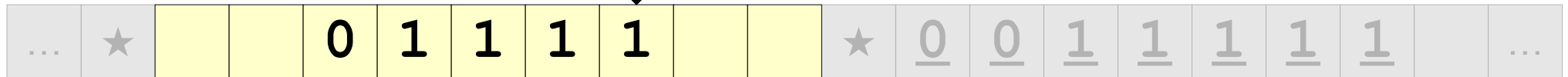
Our TM, in More Depth



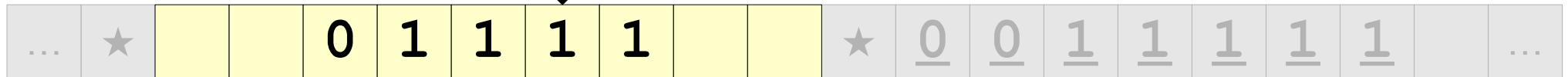
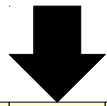
Our TM, in More Depth



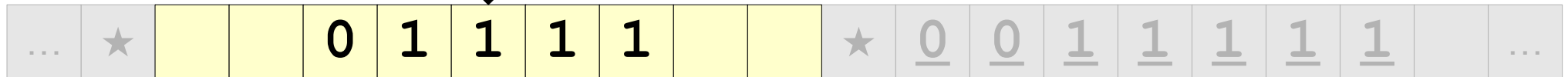
Our TM, in More Depth



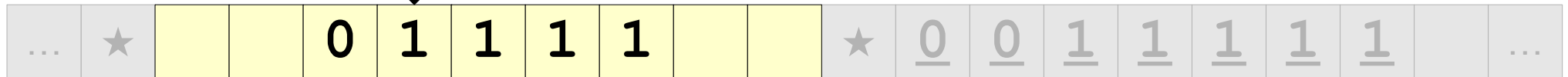
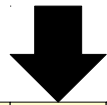
Our TM, in More Depth



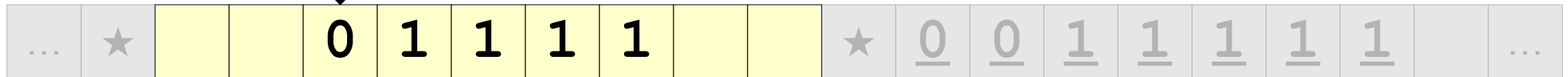
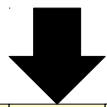
Our TM, in More Depth



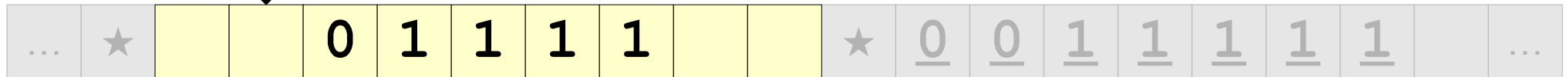
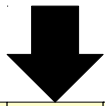
Our TM, in More Depth



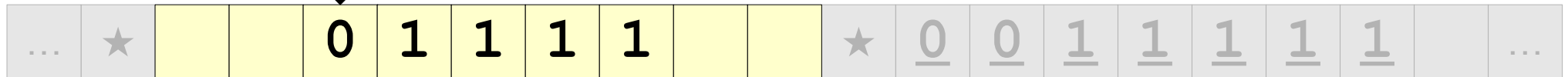
Our TM, in More Depth



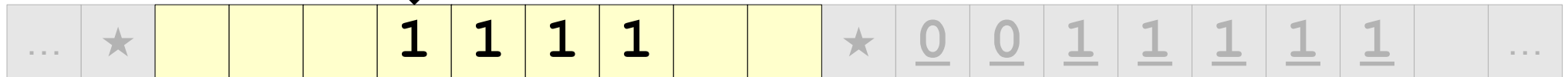
Our TM, in More Depth



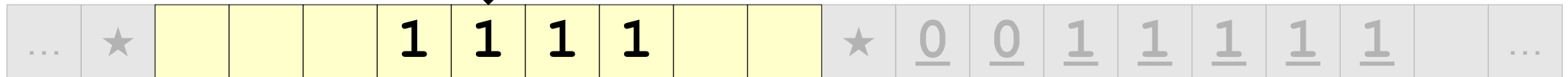
Our TM, in More Depth



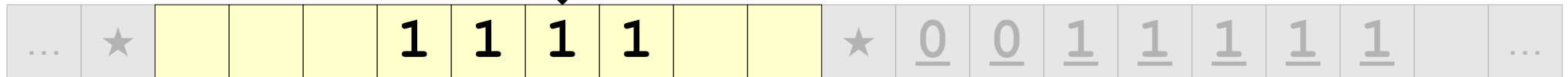
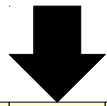
Our TM, in More Depth



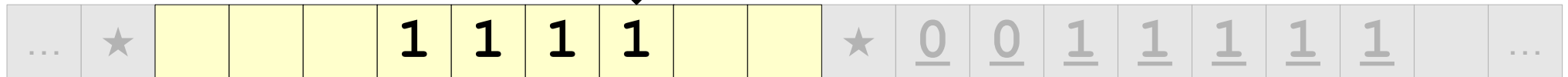
Our TM, in More Depth



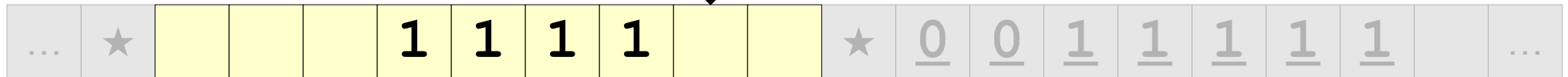
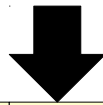
Our TM, in More Depth



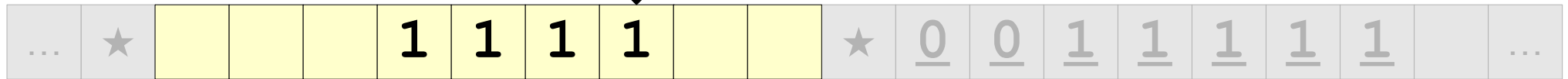
Our TM, in More Depth



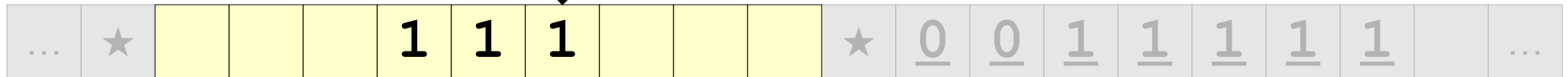
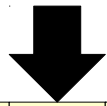
Our TM, in More Depth



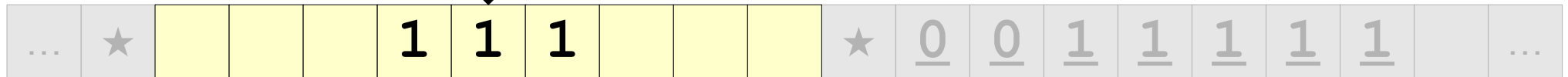
Our TM, in More Depth



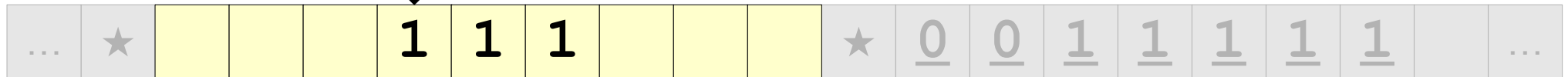
Our TM, in More Depth



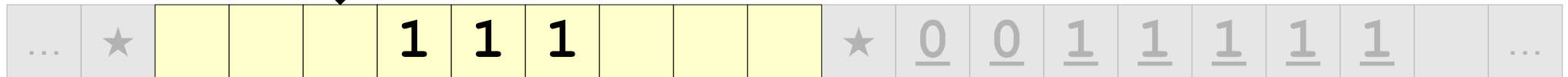
Our TM, in More Depth



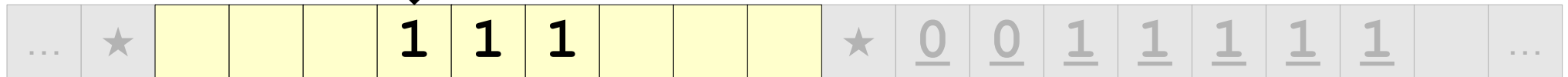
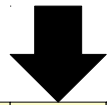
Our TM, in More Depth



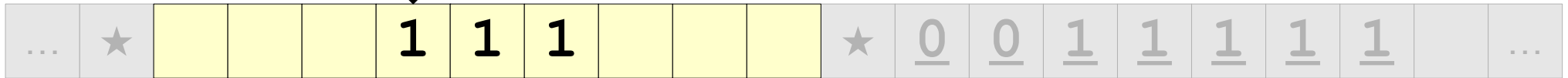
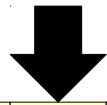
Our TM, in More Depth



Our TM, in More Depth

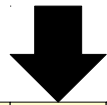


Our TM, in More Depth



At this point, our machine for checking 0^n1^n would normally reject. But what if we change the transitions to the reject state so that they now point to some other state in our overall TM?

Our TM, in More Depth



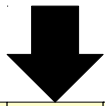
...	★				1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



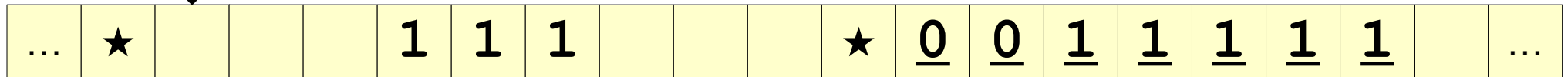
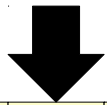
...	★				1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

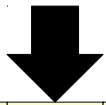


...	★				1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

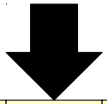


Our TM, in More Depth



...	★				1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



...					1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



...					1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



...					1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



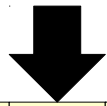
...					1	1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	---	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



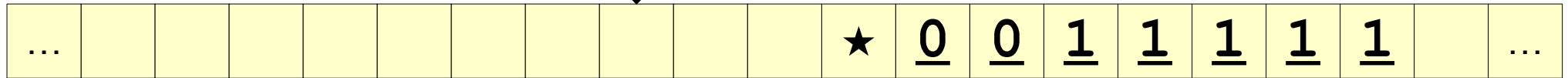
...						1	1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	---	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

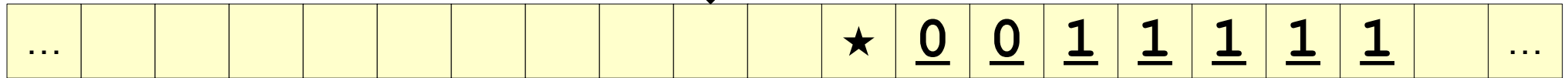
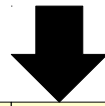


...							1				★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	---	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

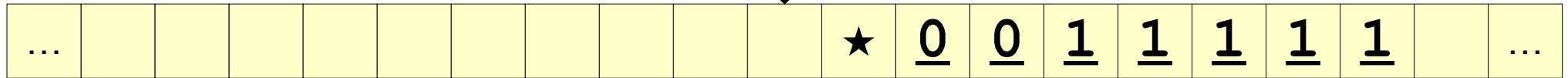
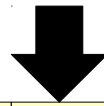
Our TM, in More Depth



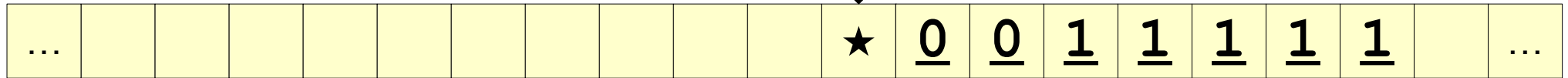
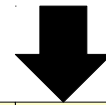
Our TM, in More Depth



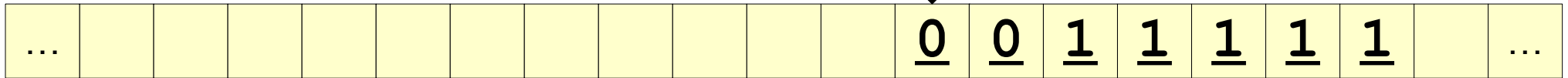
Our TM, in More Depth



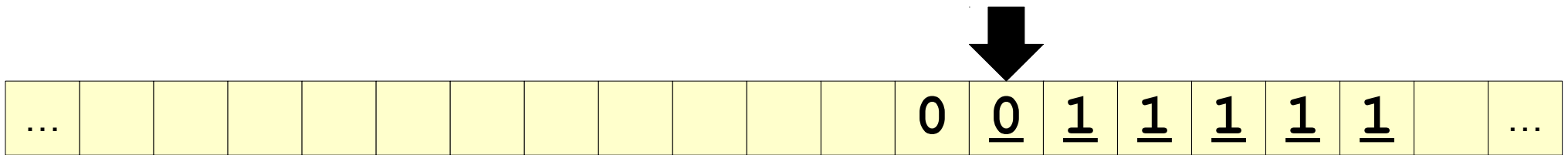
Our TM, in More Depth



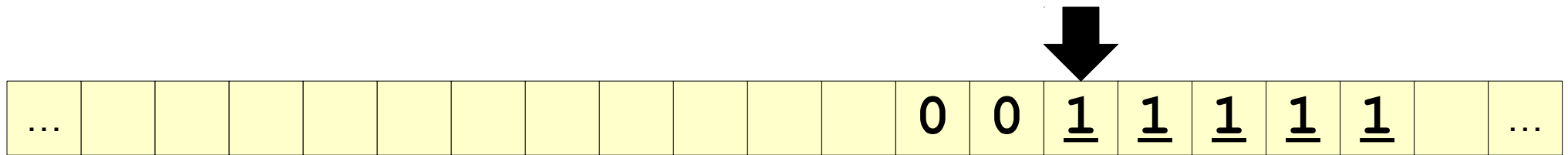
Our TM, in More Depth



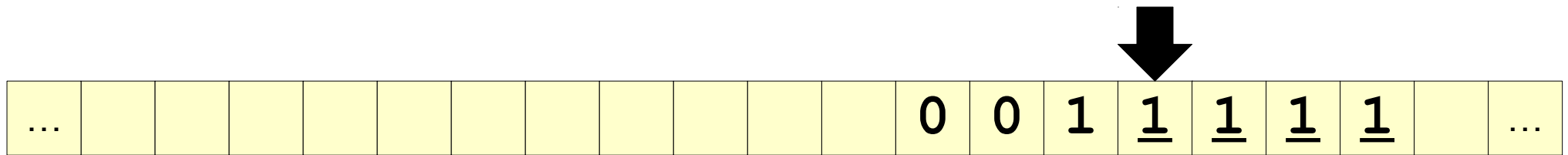
Our TM, in More Depth



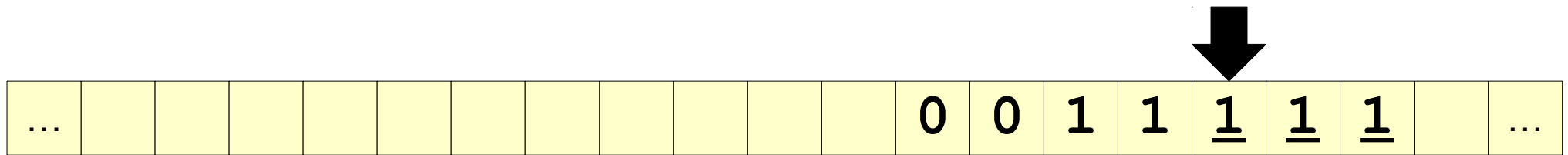
Our TM, in More Depth



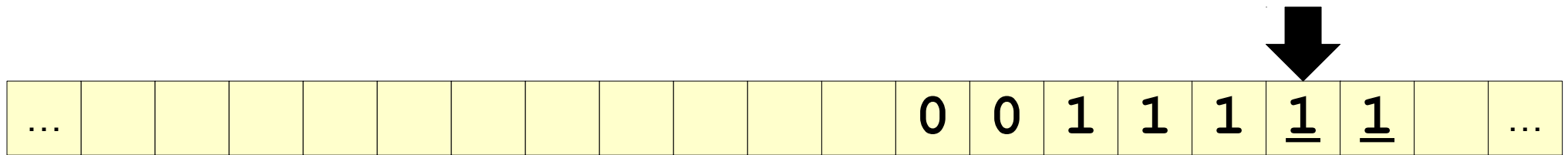
Our TM, in More Depth



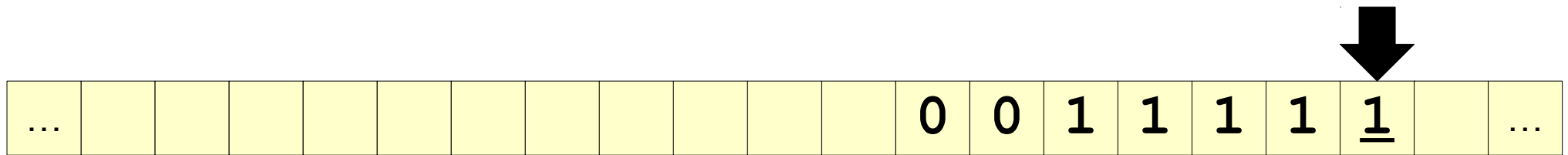
Our TM, in More Depth



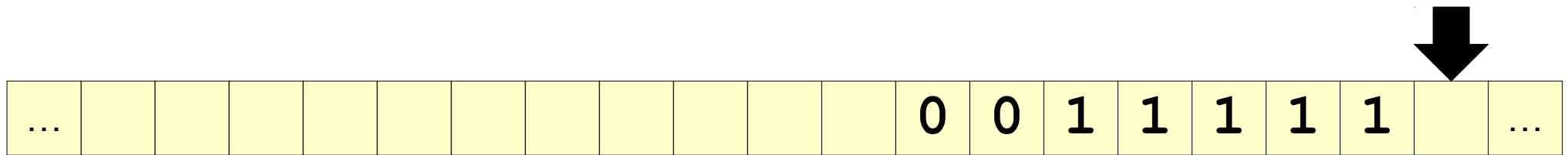
Our TM, in More Depth



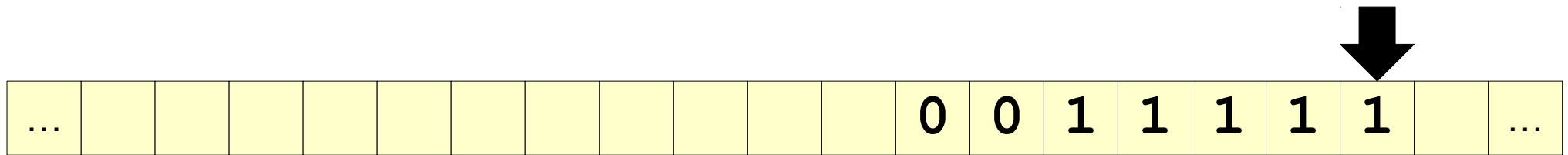
Our TM, in More Depth



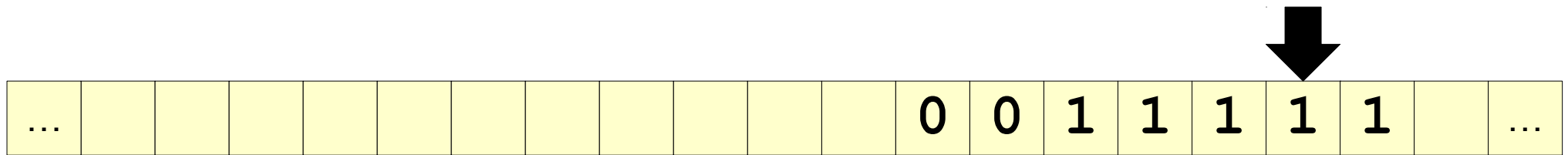
Our TM, in More Depth



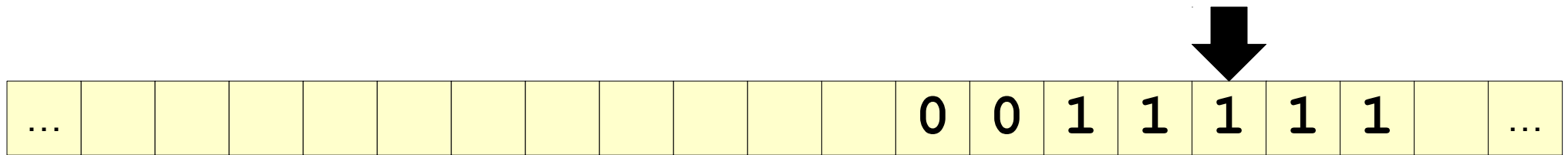
Our TM, in More Depth



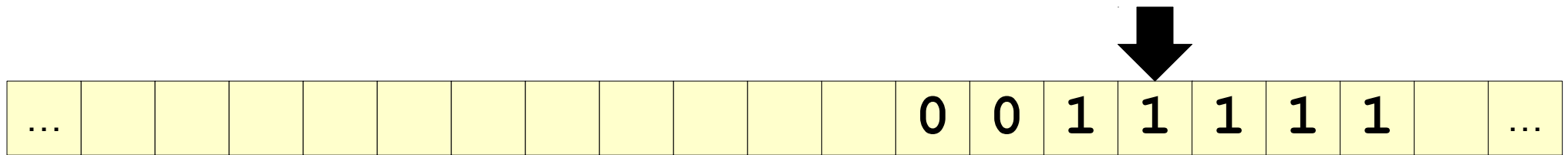
Our TM, in More Depth



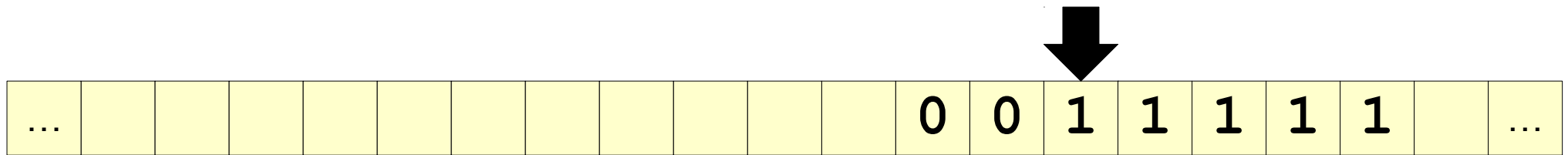
Our TM, in More Depth



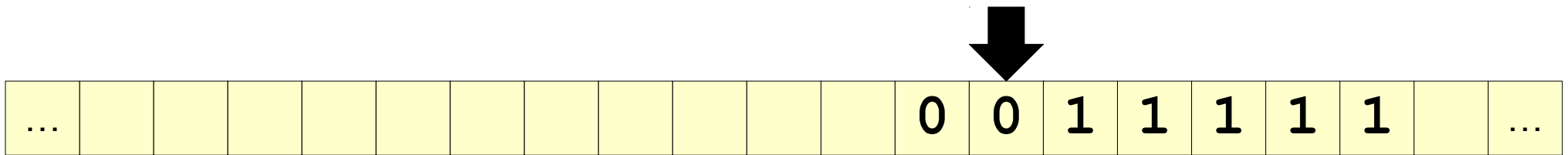
Our TM, in More Depth



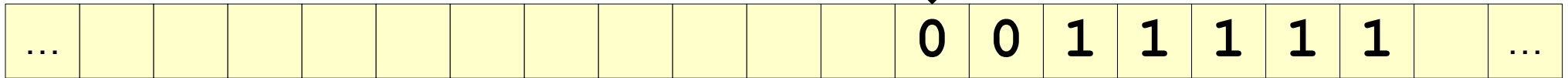
Our TM, in More Depth



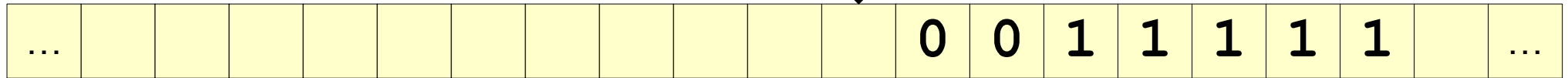
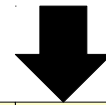
Our TM, in More Depth



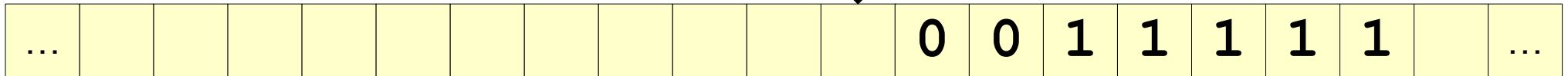
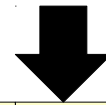
Our TM, in More Depth



Our TM, in More Depth

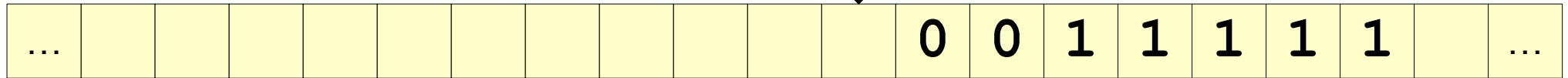
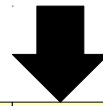


Our TM, in More Depth

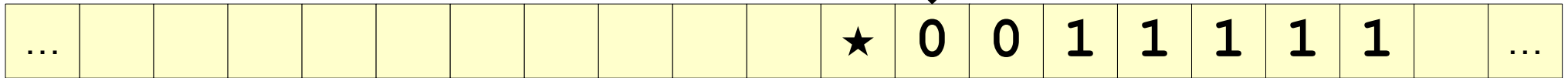
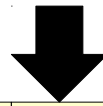


We're back in a state as if nothing happened, but now we know that the number of 0's isn't the same as the number of 1's.

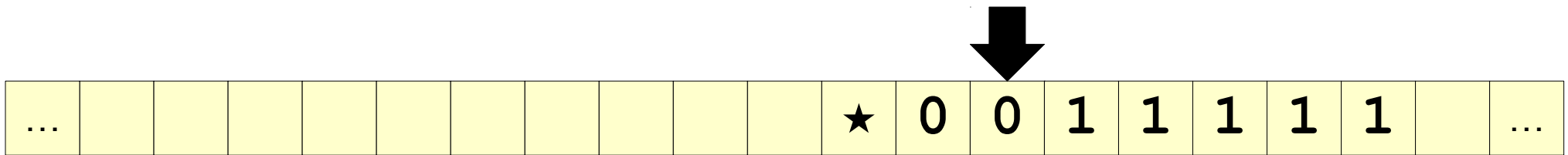
Our TM, in More Depth



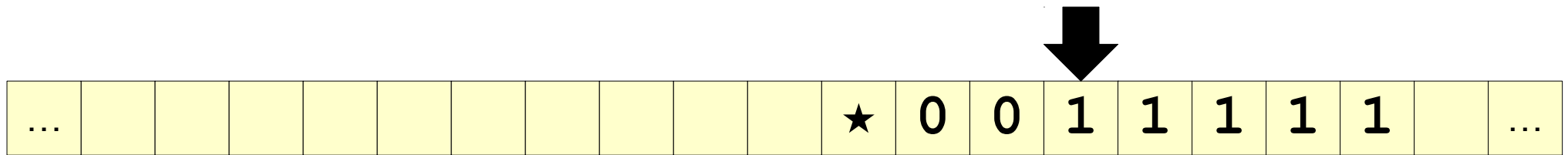
Our TM, in More Depth



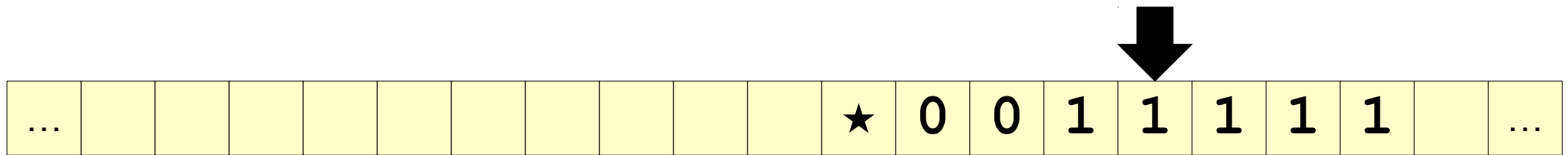
Our TM, in More Depth



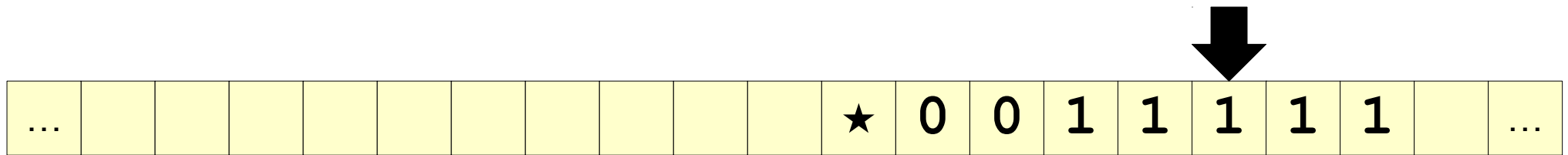
Our TM, in More Depth



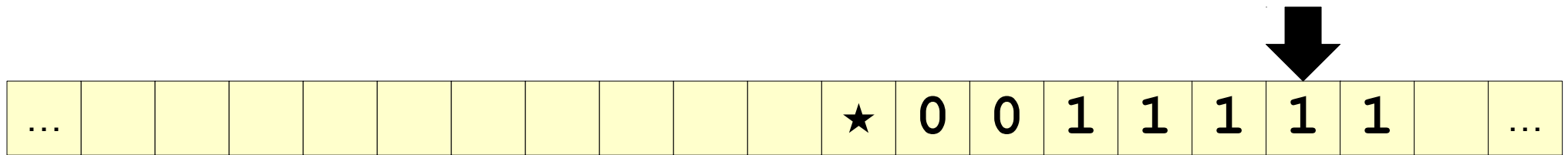
Our TM, in More Depth



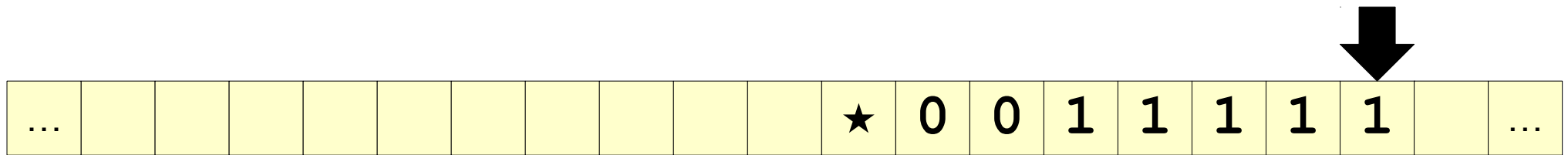
Our TM, in More Depth



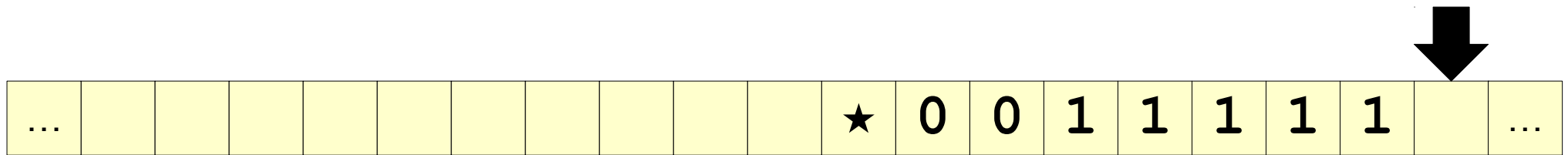
Our TM, in More Depth



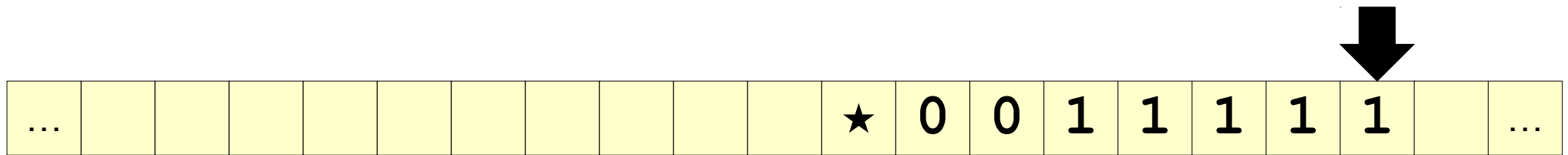
Our TM, in More Depth



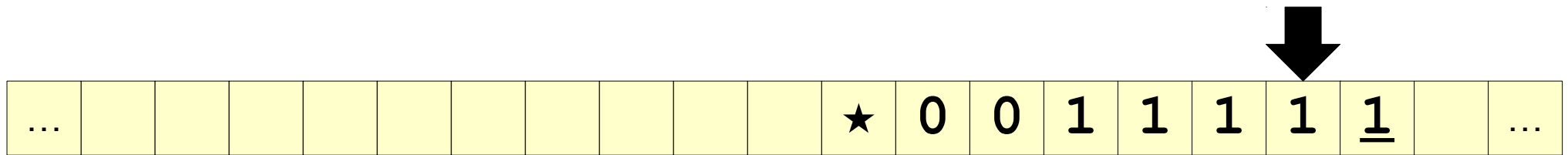
Our TM, in More Depth



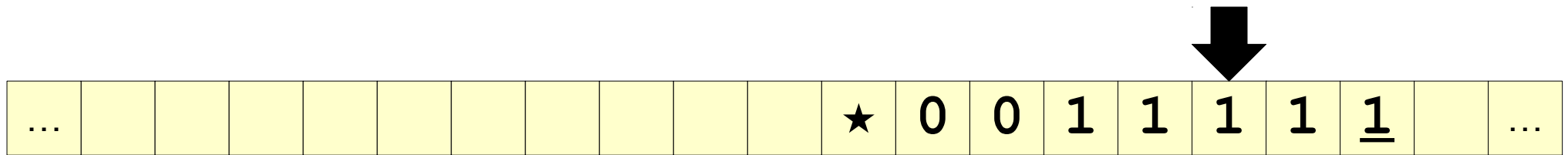
Our TM, in More Depth



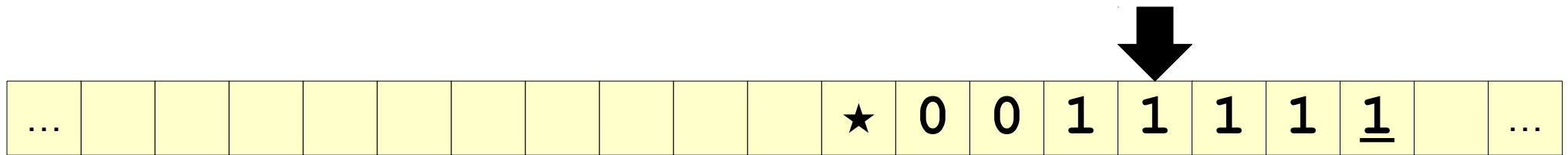
Our TM, in More Depth



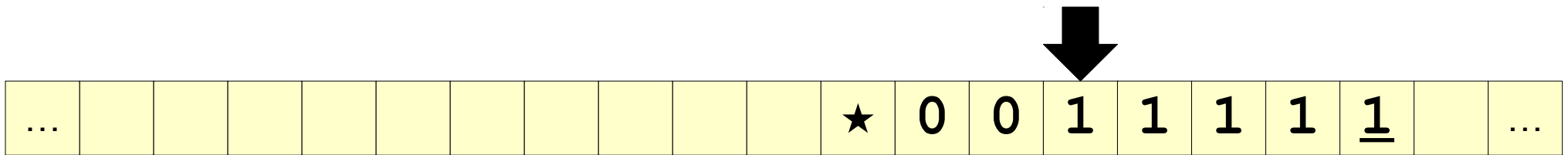
Our TM, in More Depth



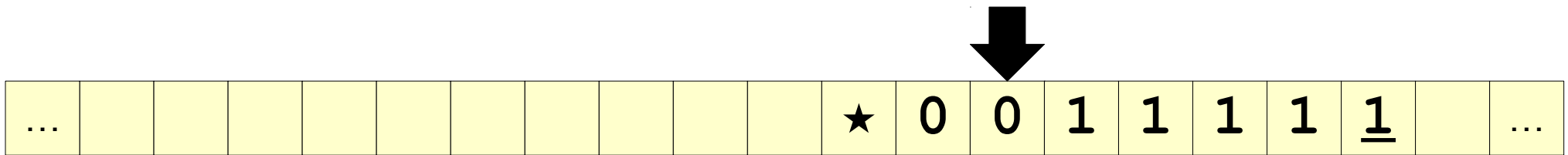
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth

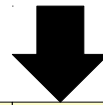


Our TM, in More Depth



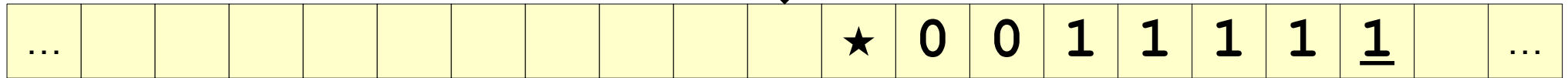
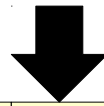
...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

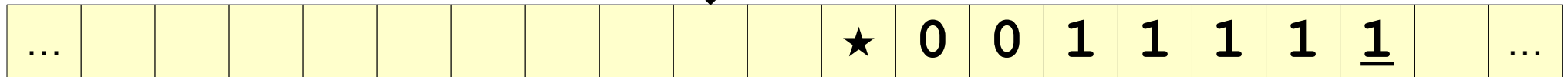
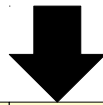


...											★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	----------	--	-----

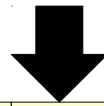
Our TM, in More Depth



Our TM, in More Depth

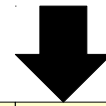


Our TM, in More Depth



...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



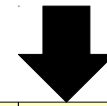
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



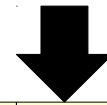
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth



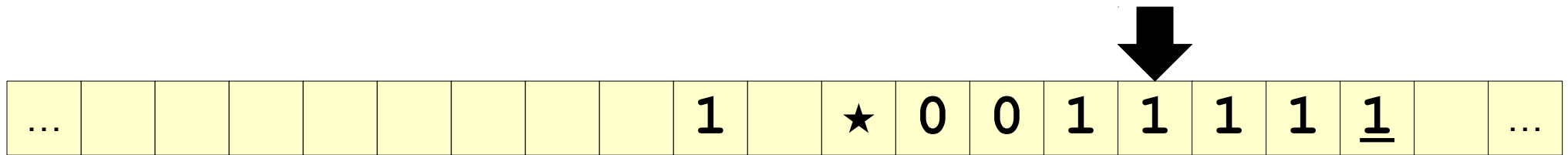
...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

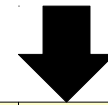


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

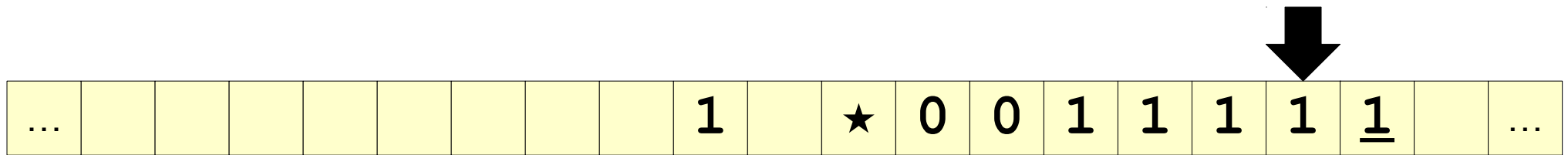


Our TM, in More Depth

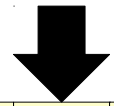


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

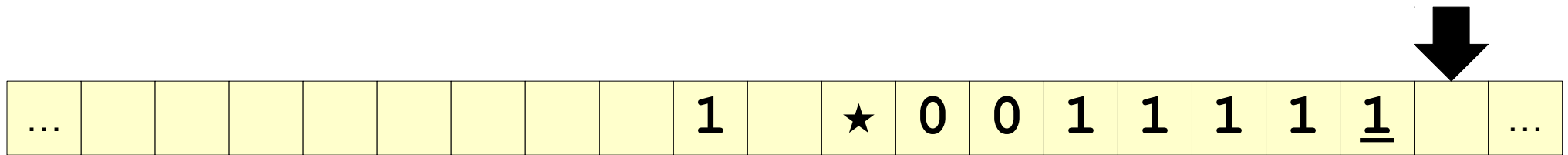


Our TM, in More Depth

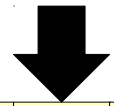


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

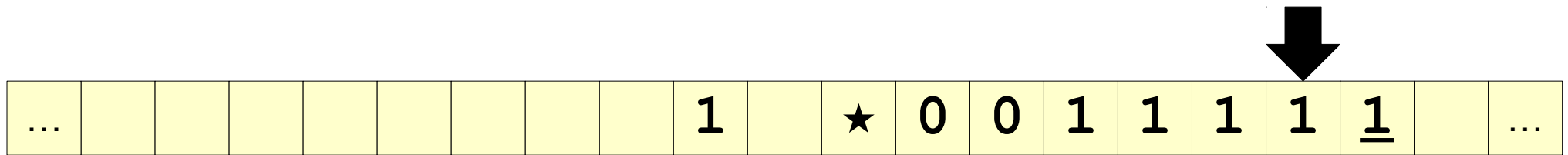


Our TM, in More Depth

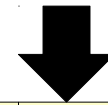


...									1		★	0	0	1	1	1	1	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	---	----------	--	-----

Our TM, in More Depth

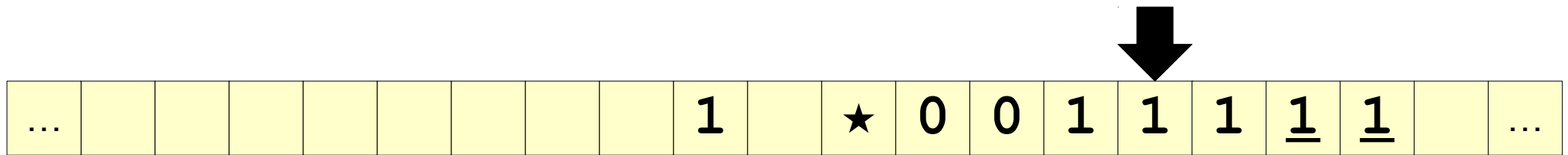


Our TM, in More Depth

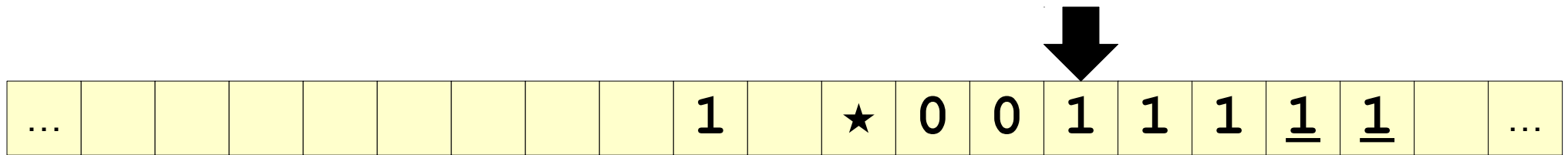


...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

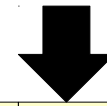
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth



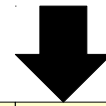
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



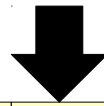
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



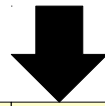
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



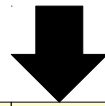
...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



...									1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	--	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth



...								1	1		★	0	0	1	1	1	<u>1</u>	<u>1</u>		...
-----	--	--	--	--	--	--	--	---	---	--	---	---	---	---	---	---	----------	----------	--	-----

Our TM, in More Depth

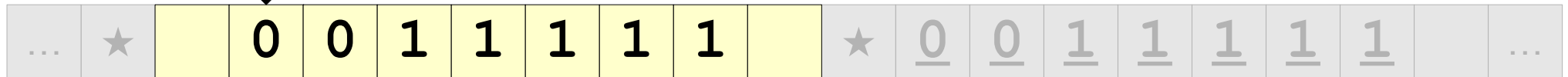
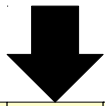


Our TM, in More Depth

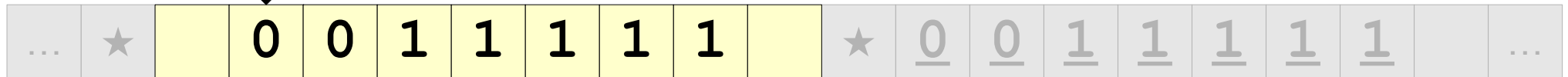
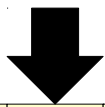


...	★		0	0	1	1	1	1	1		★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	---	---	---	---	---	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

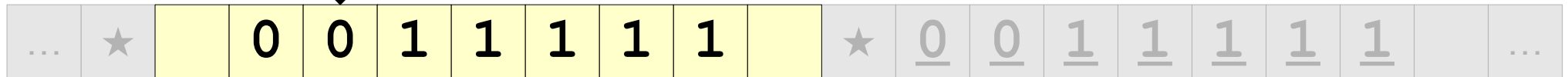


Our TM, in More Depth

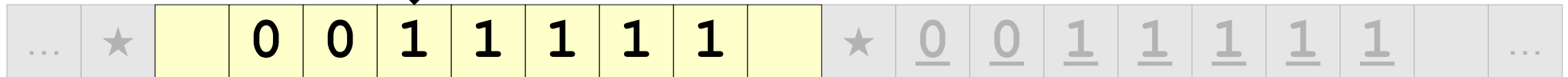
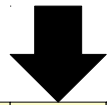


Our TM that checks whether a number is composite never walks more than one step off the input in either direction. It has no way of knowing that there's anything else on the tape.

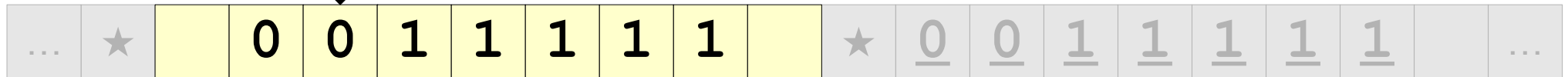
Our TM, in More Depth



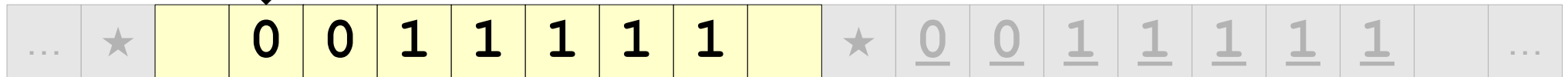
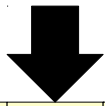
Our TM, in More Depth



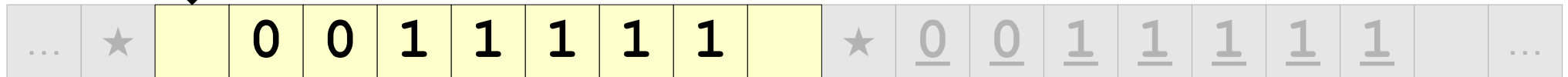
Our TM, in More Depth



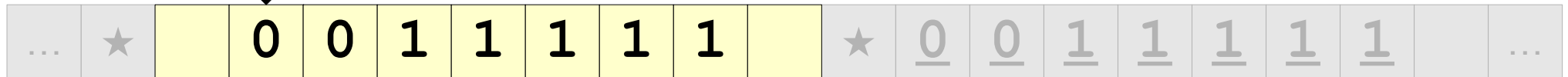
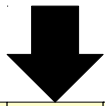
Our TM, in More Depth



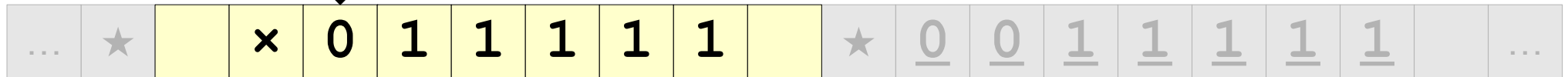
Our TM, in More Depth



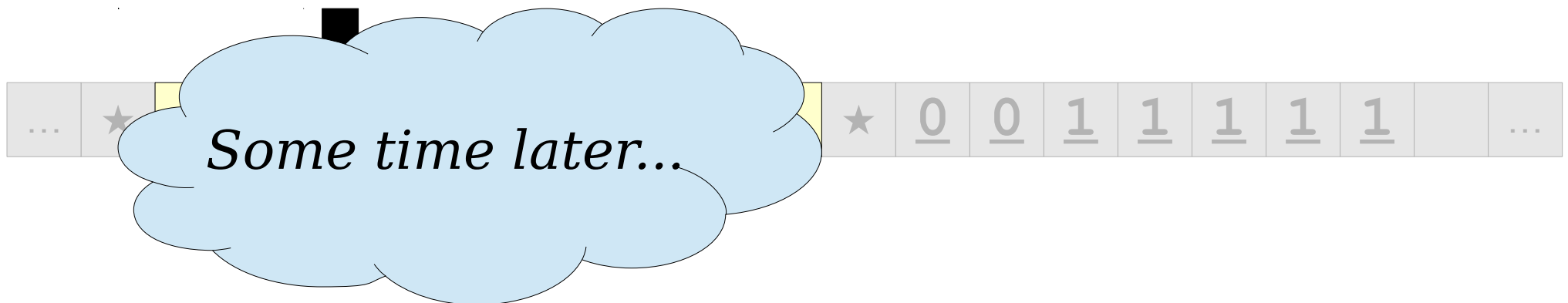
Our TM, in More Depth



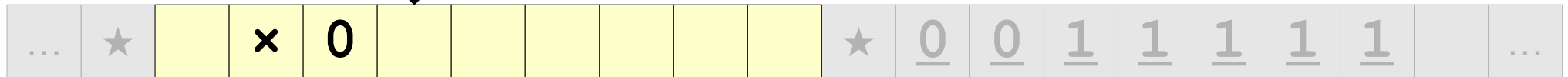
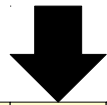
Our TM, in More Depth



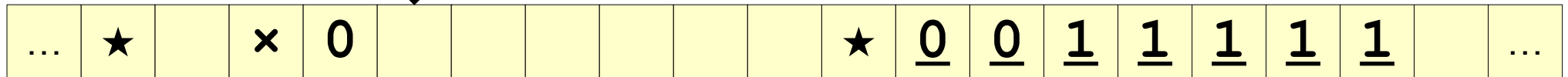
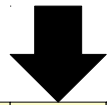
Our TM, in More Depth



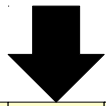
Our TM, in More Depth



Our TM, in More Depth

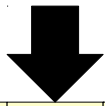


Our TM, in More Depth



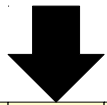
...	★		×	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



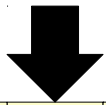
...	★		x	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



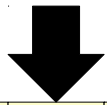
...	★		×	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



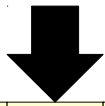
...	★		×	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	---	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



...			x	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth



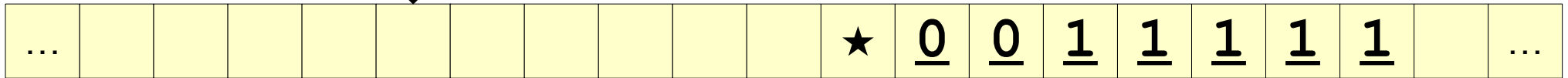
...			×	0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	---	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

Our TM, in More Depth

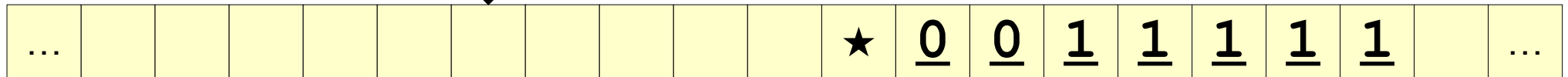


...				0							★	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>		...
-----	--	--	--	---	--	--	--	--	--	--	---	----------	----------	----------	----------	----------	----------	----------	--	-----

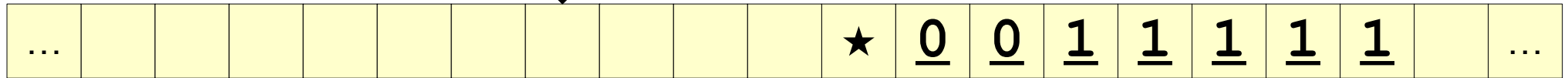
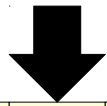
Our TM, in More Depth



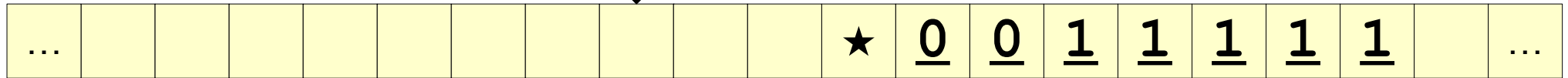
Our TM, in More Depth



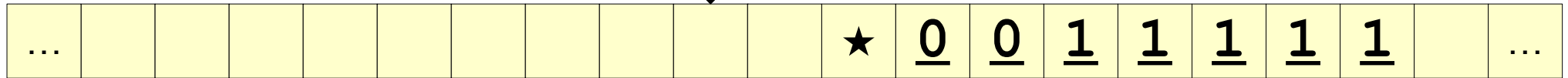
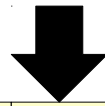
Our TM, in More Depth



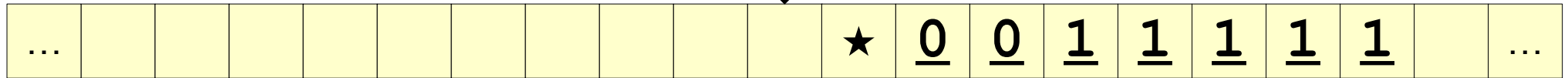
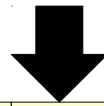
Our TM, in More Depth



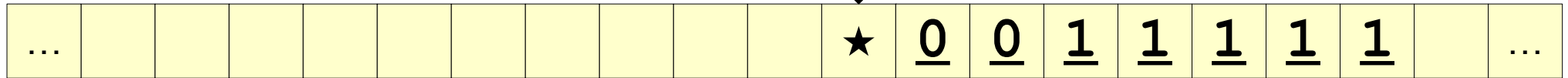
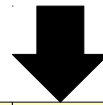
Our TM, in More Depth



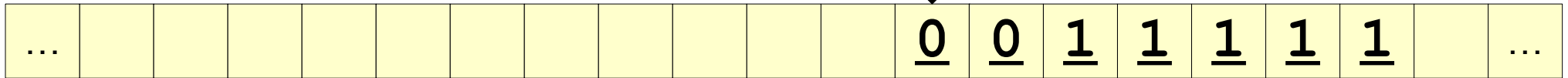
Our TM, in More Depth



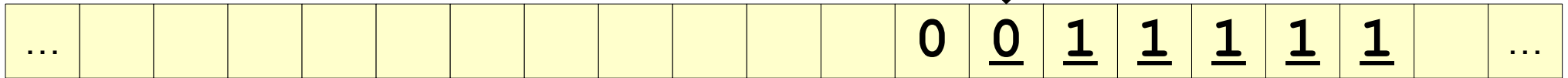
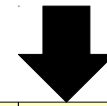
Our TM, in More Depth



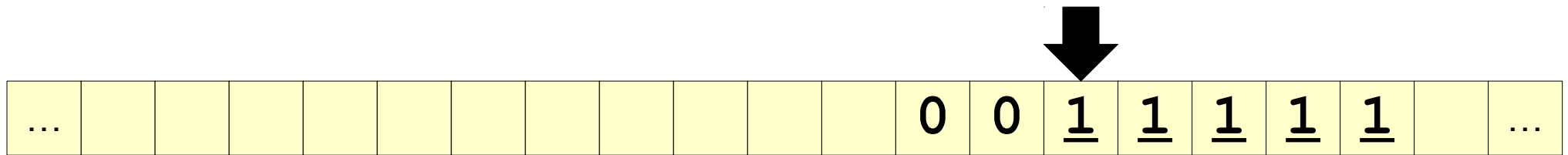
Our TM, in More Depth



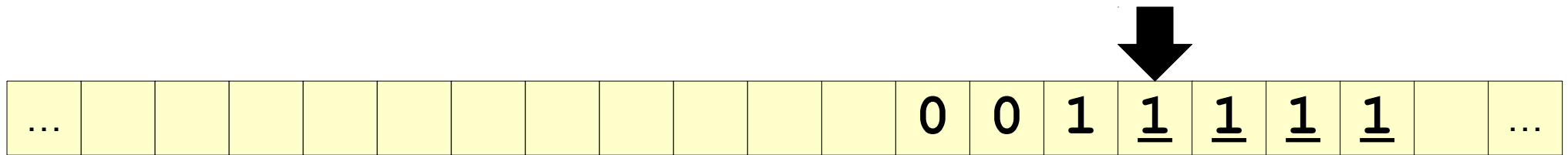
Our TM, in More Depth



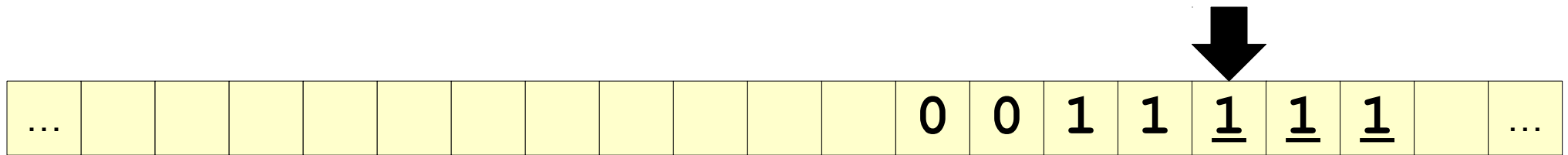
Our TM, in More Depth



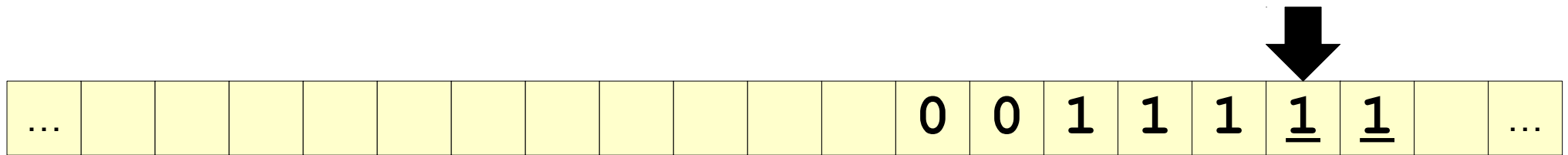
Our TM, in More Depth



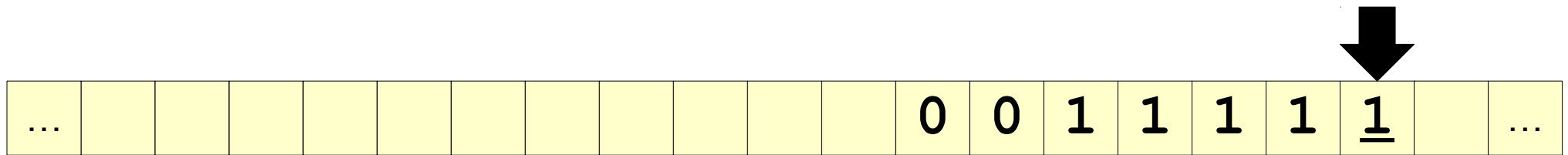
Our TM, in More Depth



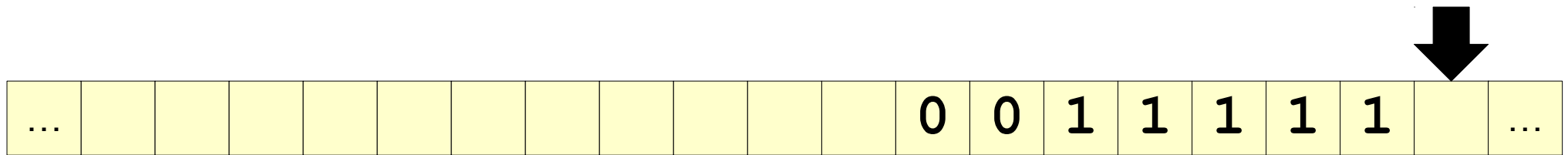
Our TM, in More Depth



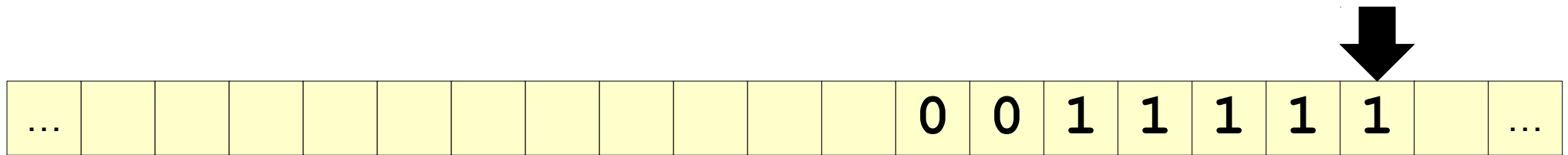
Our TM, in More Depth



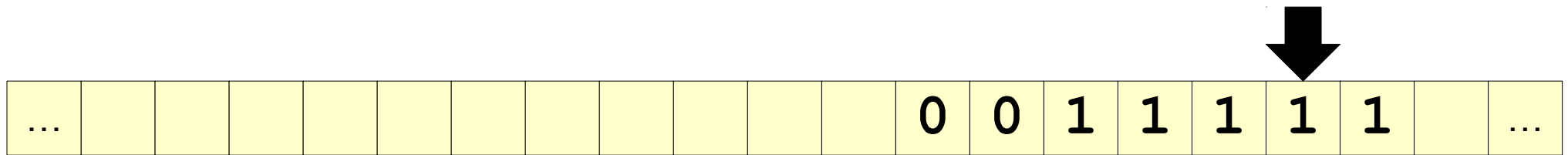
Our TM, in More Depth



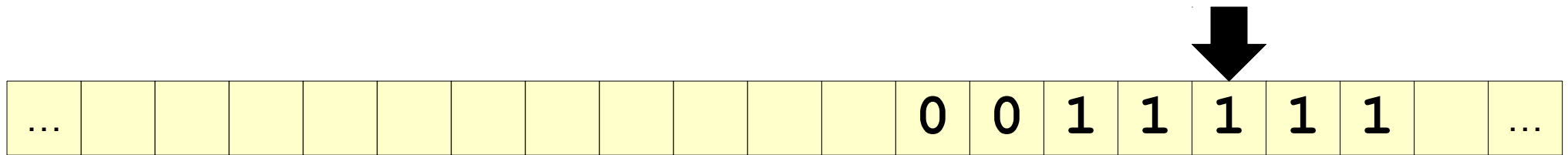
Our TM, in More Depth



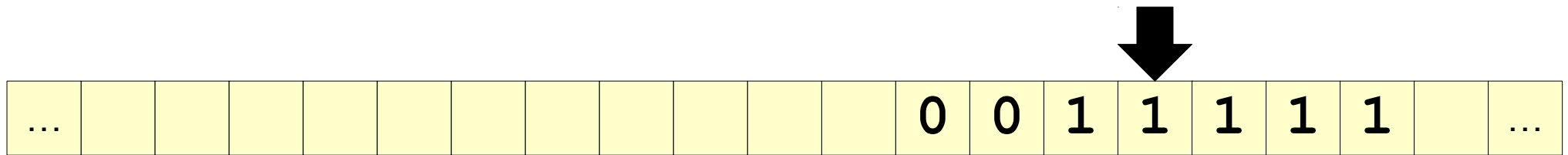
Our TM, in More Depth



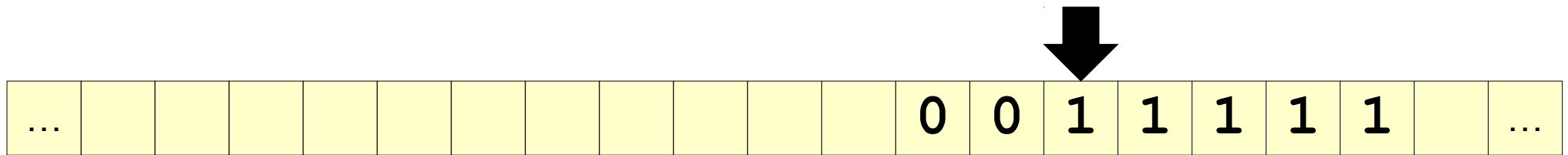
Our TM, in More Depth



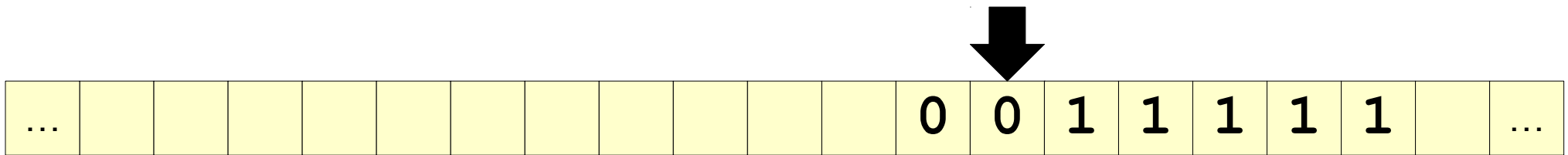
Our TM, in More Depth



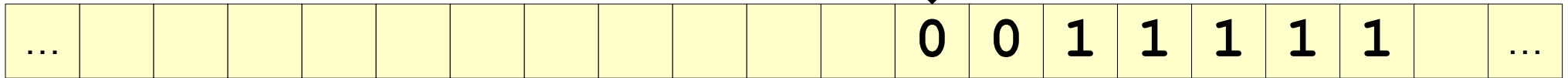
Our TM, in More Depth



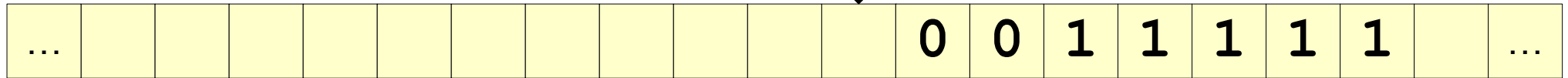
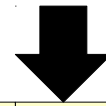
Our TM, in More Depth



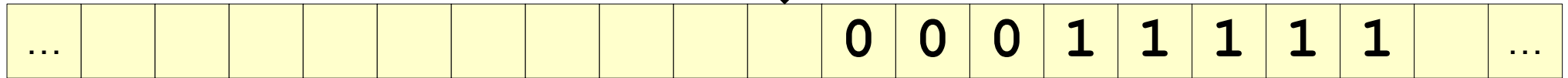
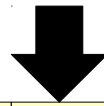
Our TM, in More Depth



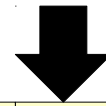
Our TM, in More Depth



Our TM, in More Depth



Our TM, in More Depth



...										★	0	0	0	1	1	1	1	1		...
-----	--	--	--	--	--	--	--	--	--	---	---	---	---	---	---	---	---	---	--	-----

The General Construction

- Check that the input isn't 1^0 or 1^1 as edge cases.
- Write two 0's in front of the input.
- Repeat the following:
 - If the number of 0's is the same as the number of 1's, reject.
 - If the number of 1's is a multiple of the number of 0's, accept.
 - Otherwise, put another 0 in front of the input and repeat.

Subroutines in TMs

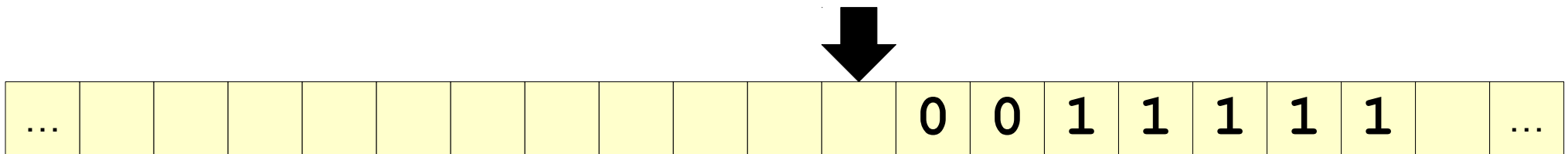
- Just as complex programs are often broken down into smaller functions and classes, complex TMs are often broken down into smaller “subroutines.”
- Each subroutine performs some task that helps in the overall task.
- The TM is then described by giving a collection of subroutines and showing how they link up.

Subroutines in TMs

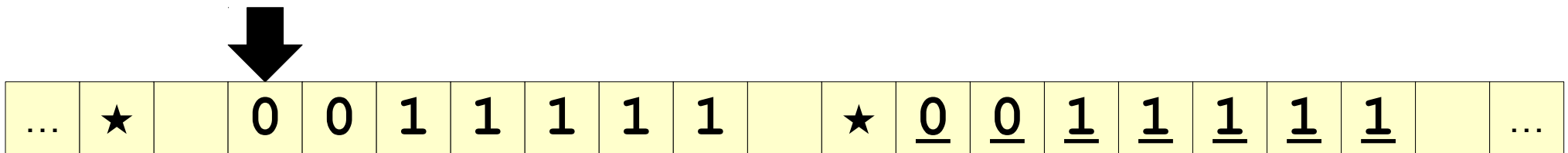
- In our TM, we need subroutines for the following:
 - Copying the TM contents to another spot on the tape.
 - Checking whether the number of 1's in that copy is a multiple of the number of 0's in that copy (we already made this!)
 - Cleaning up the tape if the answer is no.
- Let's see some of these pieces.

The “Copy” Subroutine

- This subroutine starts with the tape head at the start of a string of 0s and 1s:



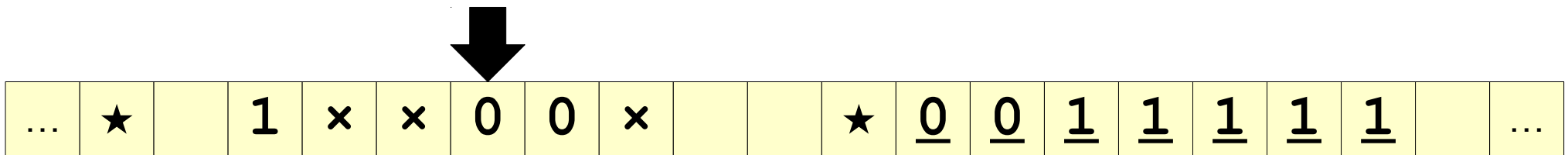
- It ends in this configuration:



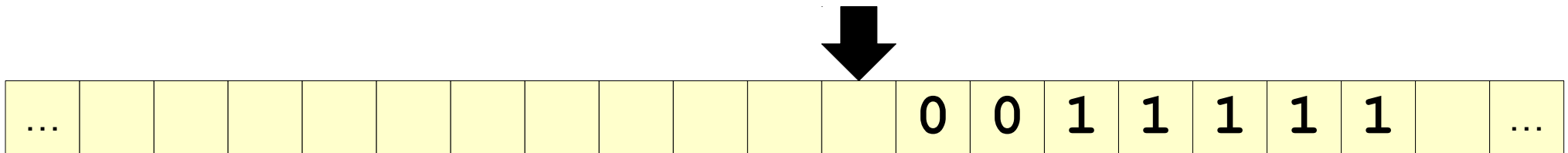
- We use the copy subroutine to let us run another TM on the current input without breaking it.

The “Cleanup” Subroutine

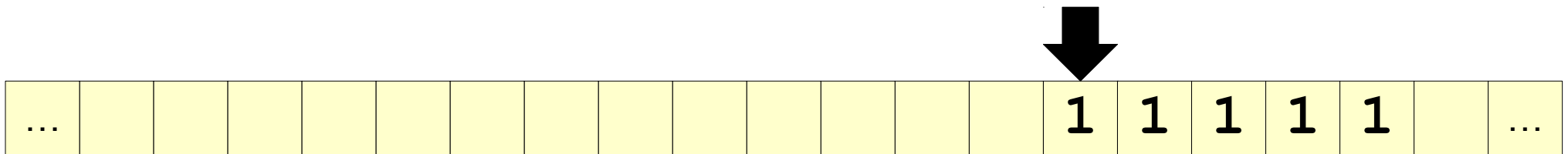
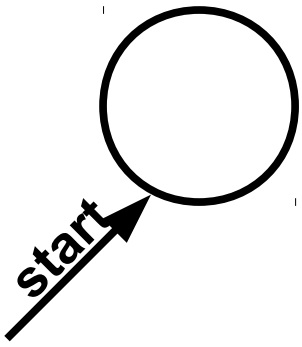
- This subroutine starts with the tape head between two ★ characters delimiting TM workspace:

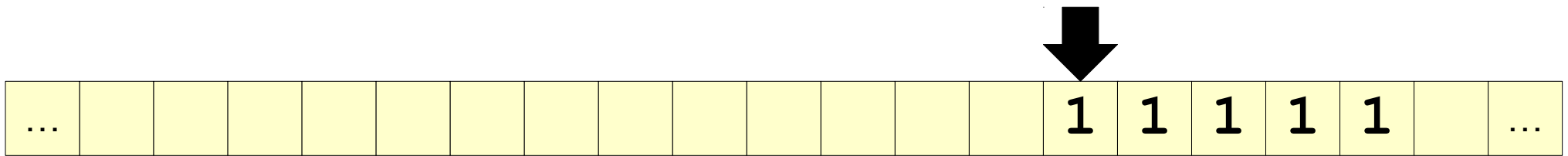
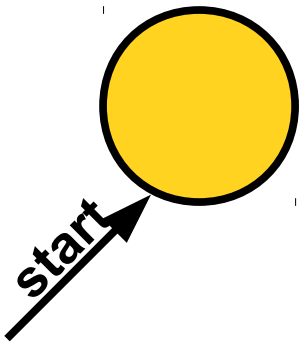


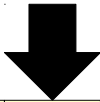
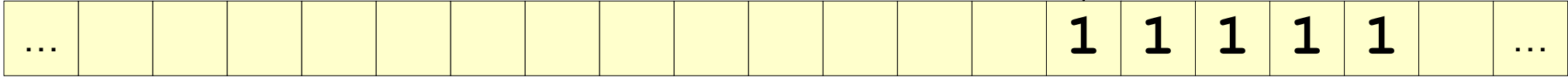
- It ends in this configuration:

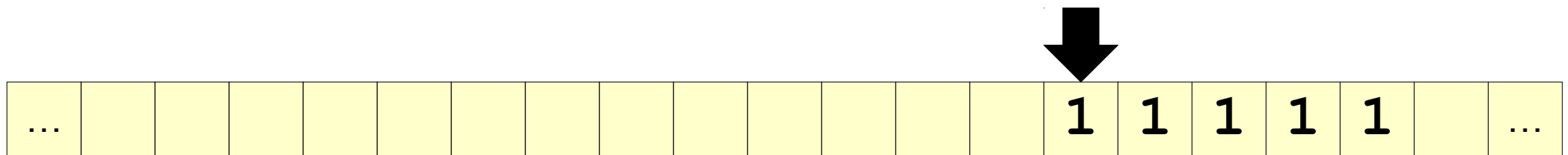
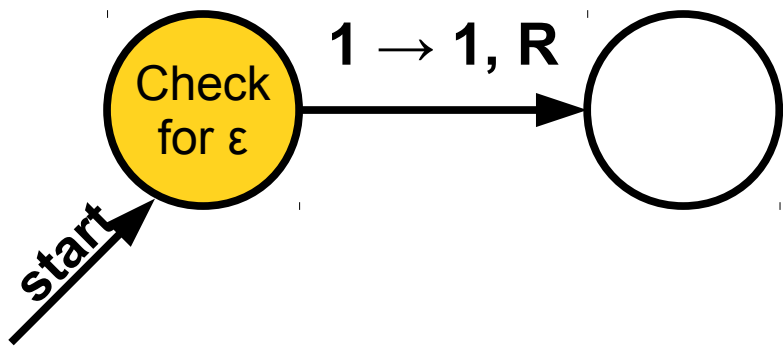


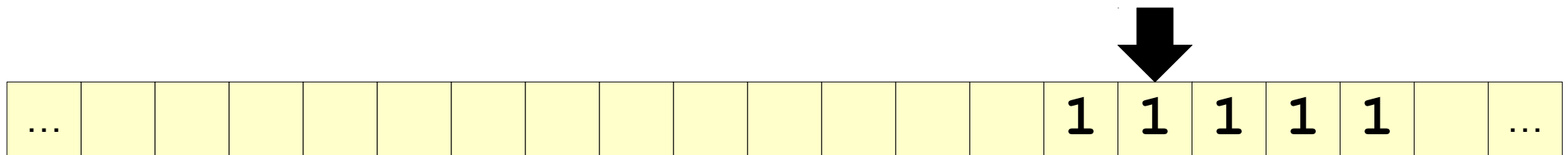
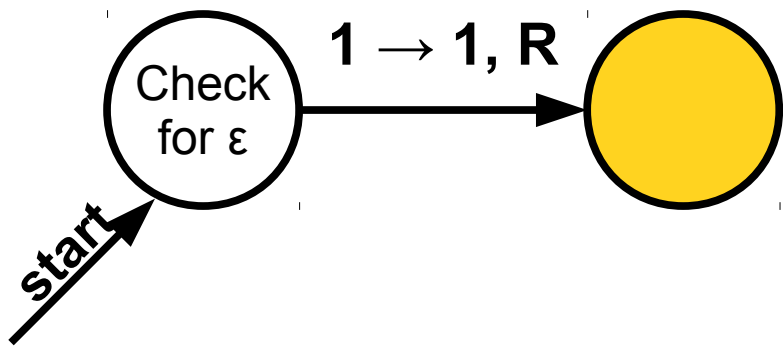
- We use the cleanup subroutine to recover from the end of running a sub-TM.

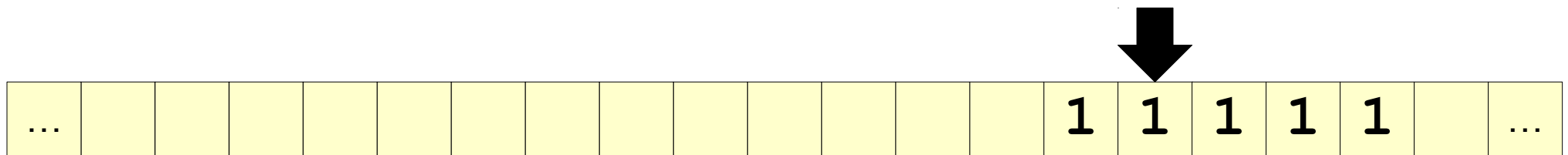
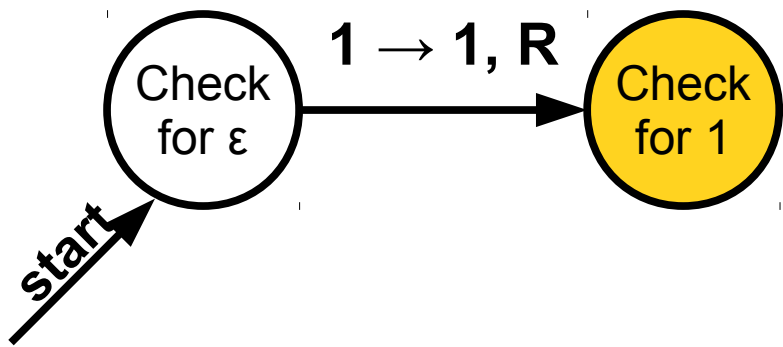


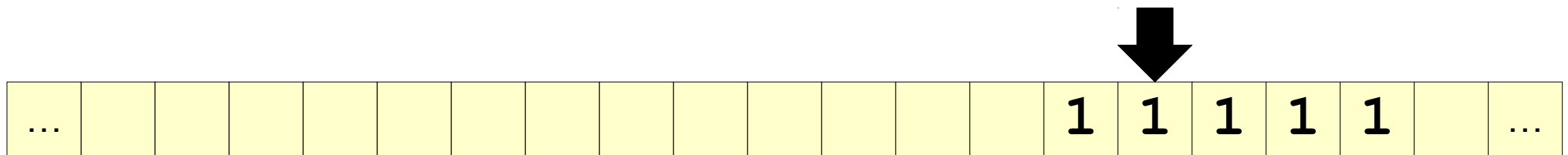
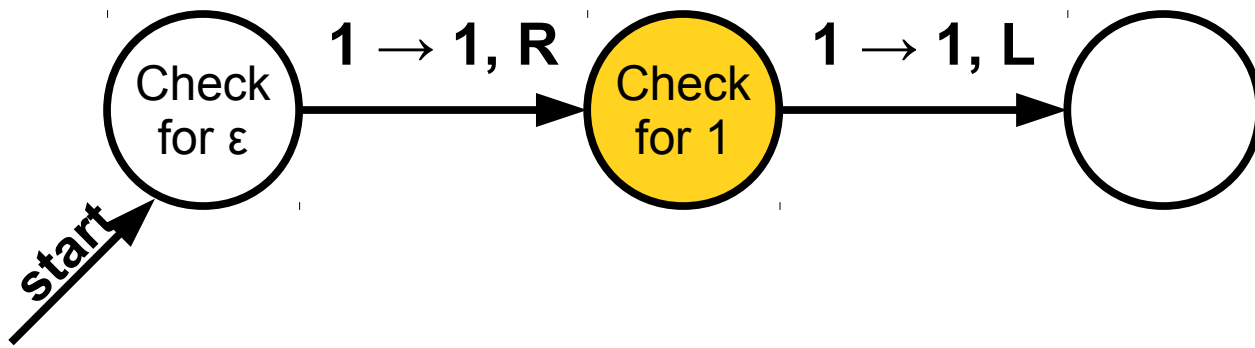


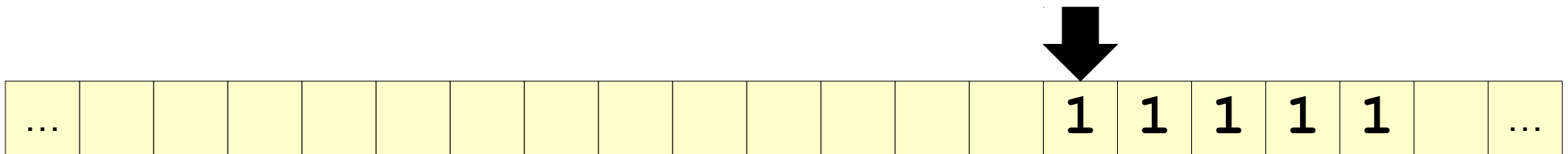
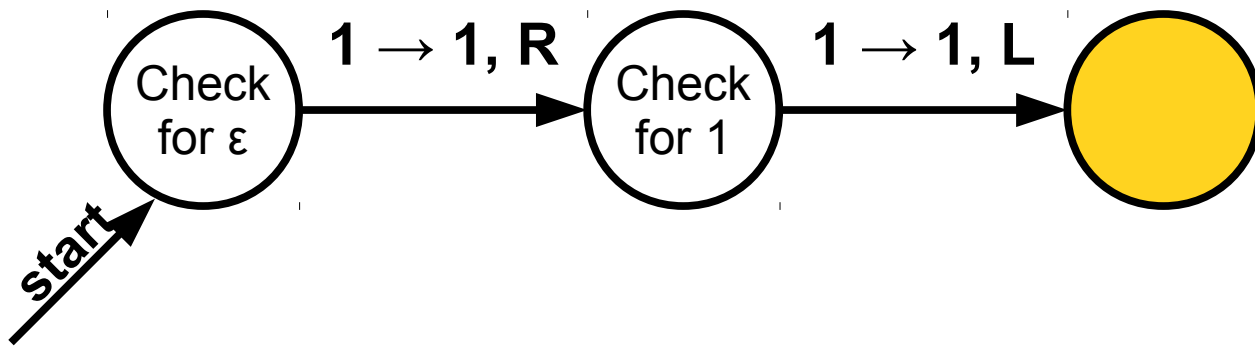


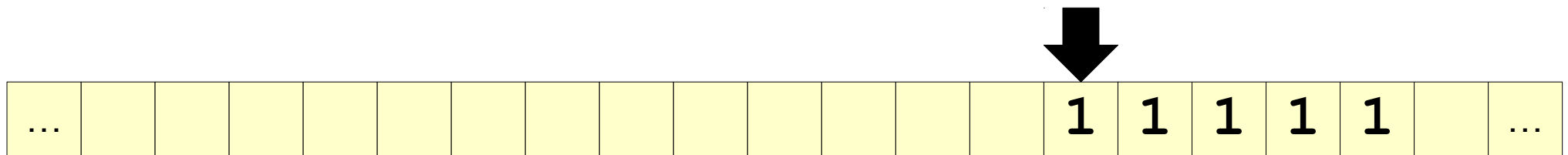
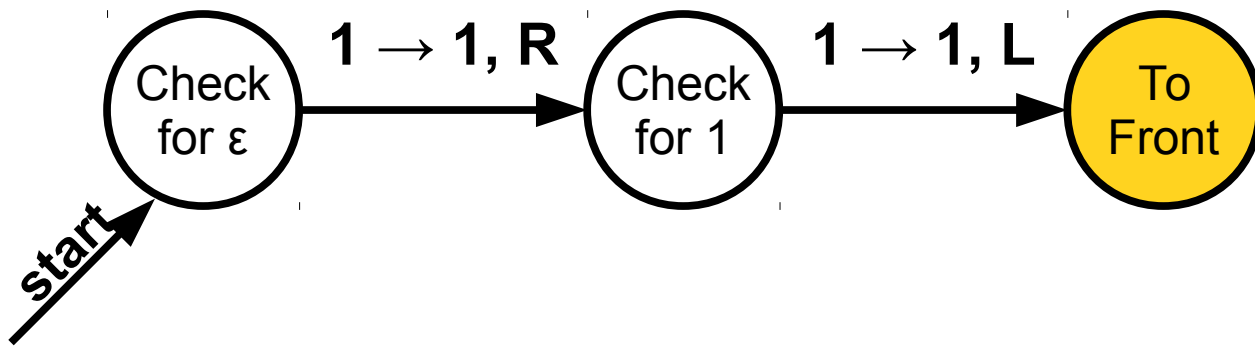


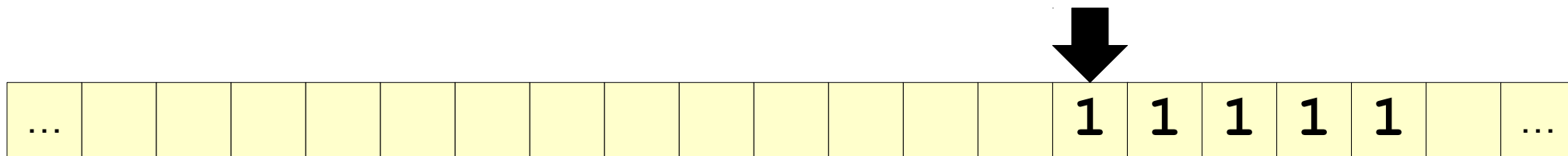
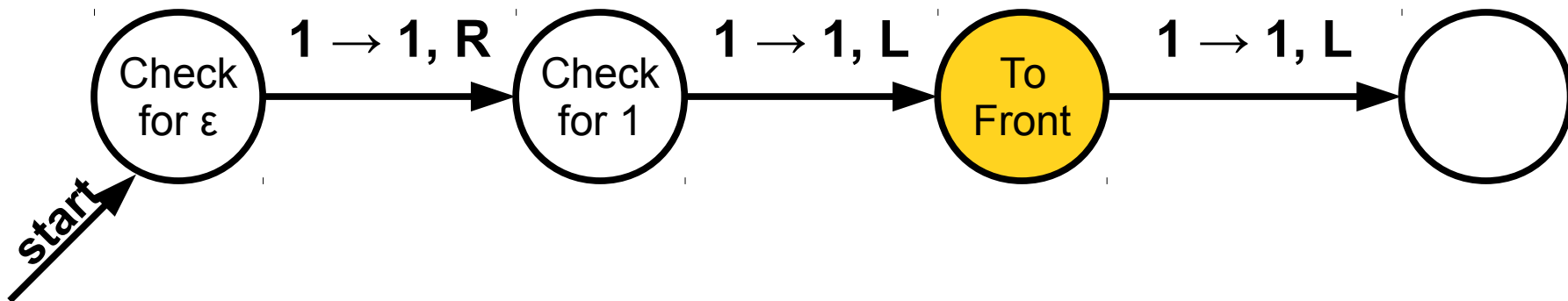


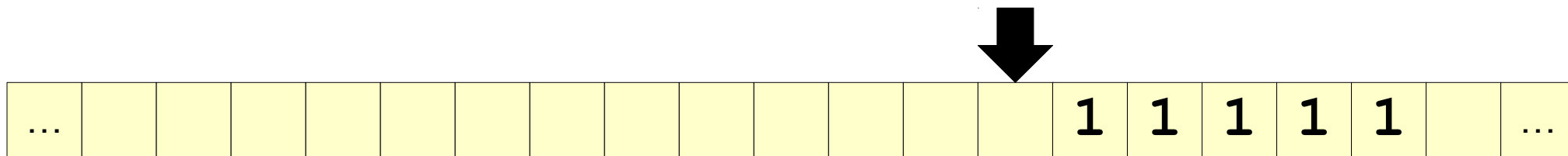
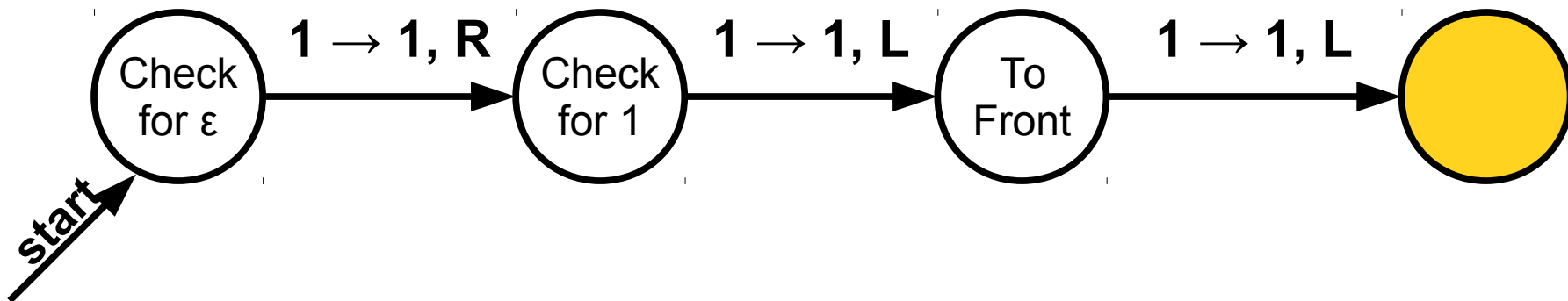


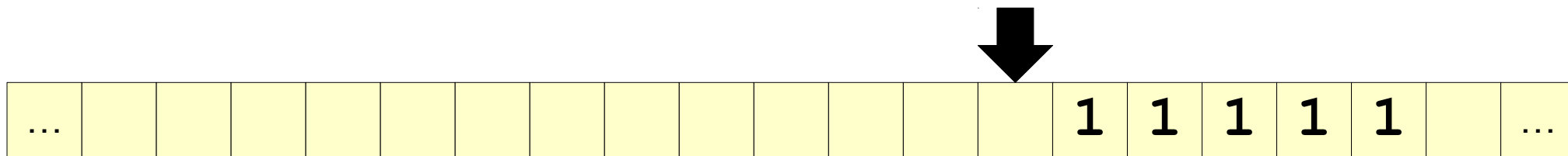
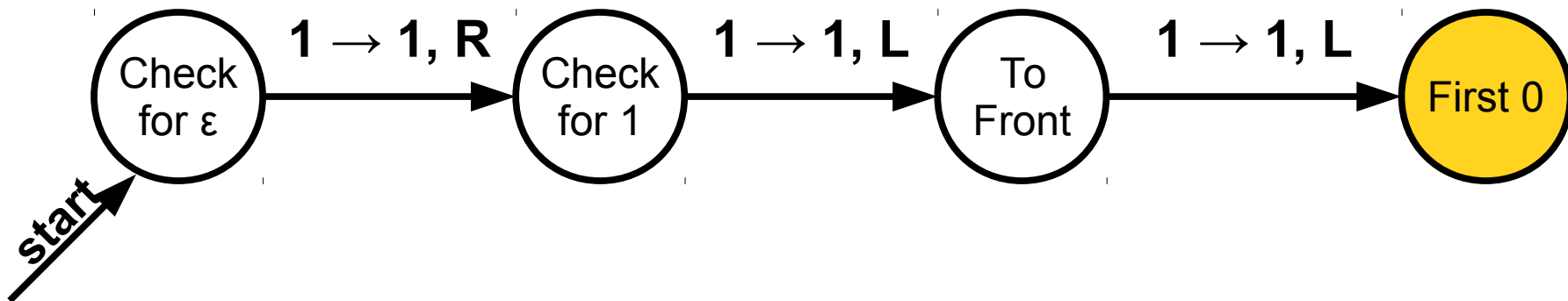


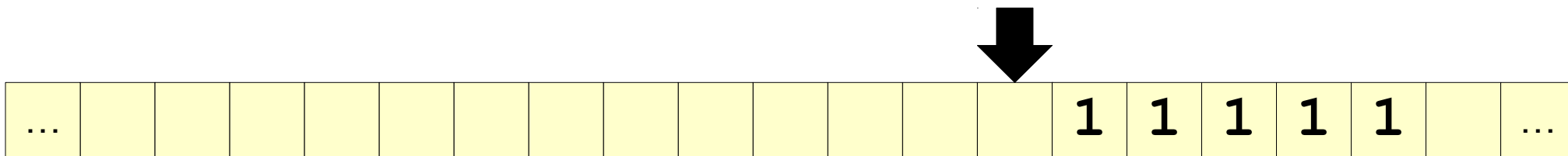
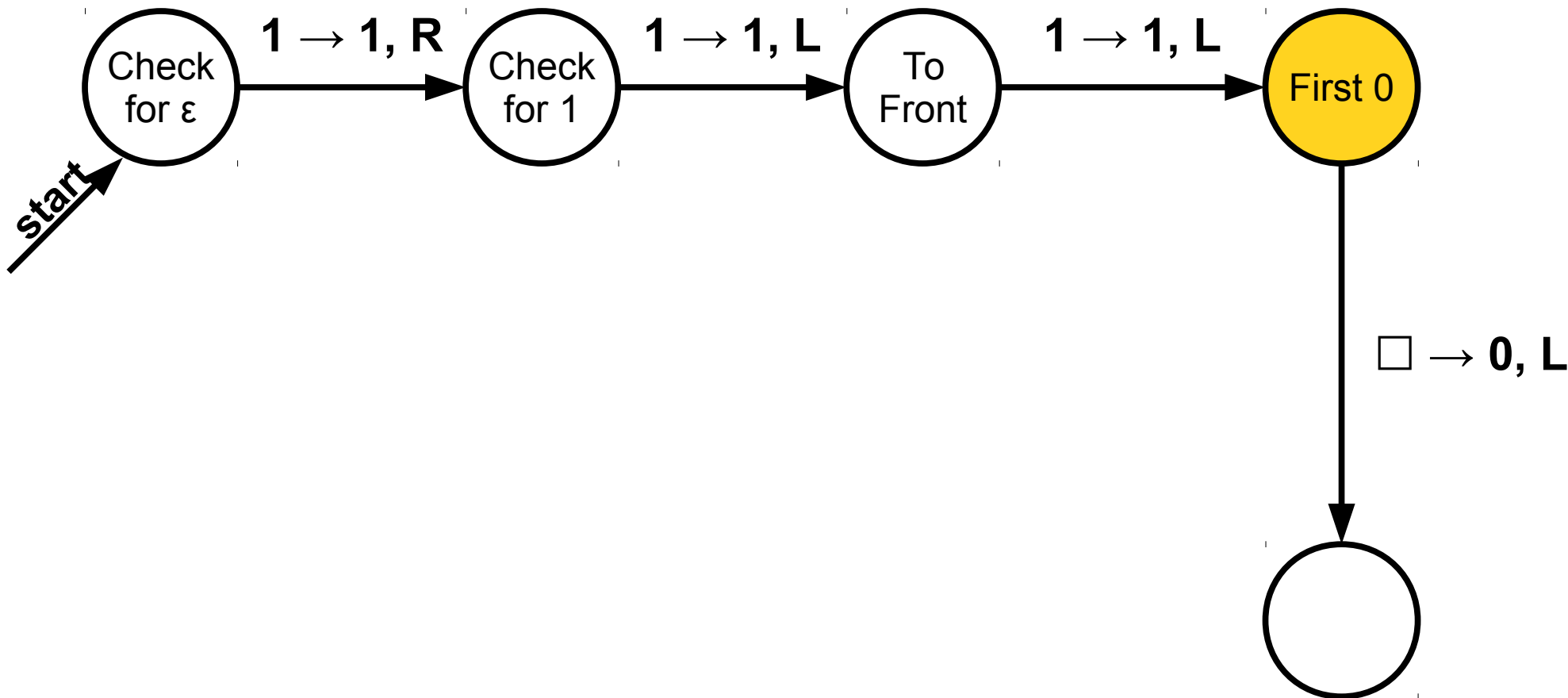


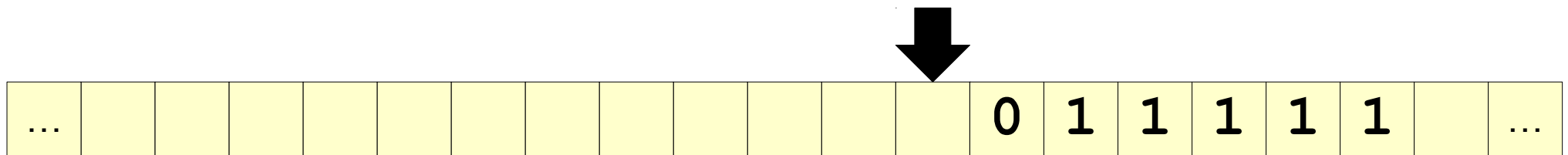
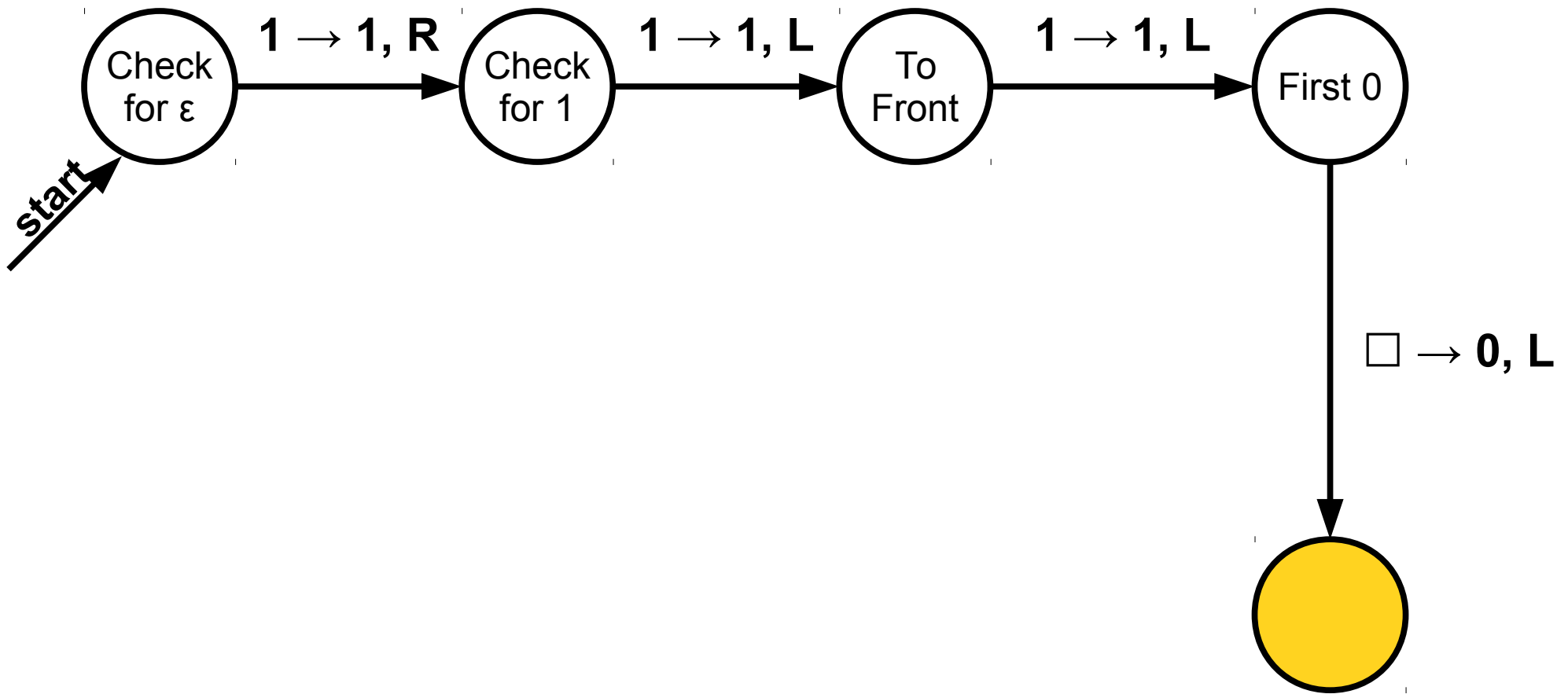


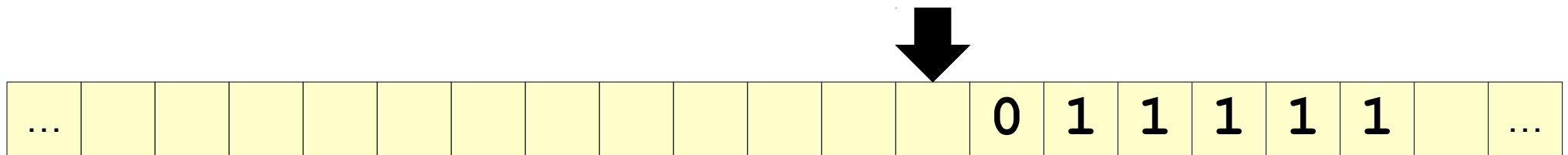
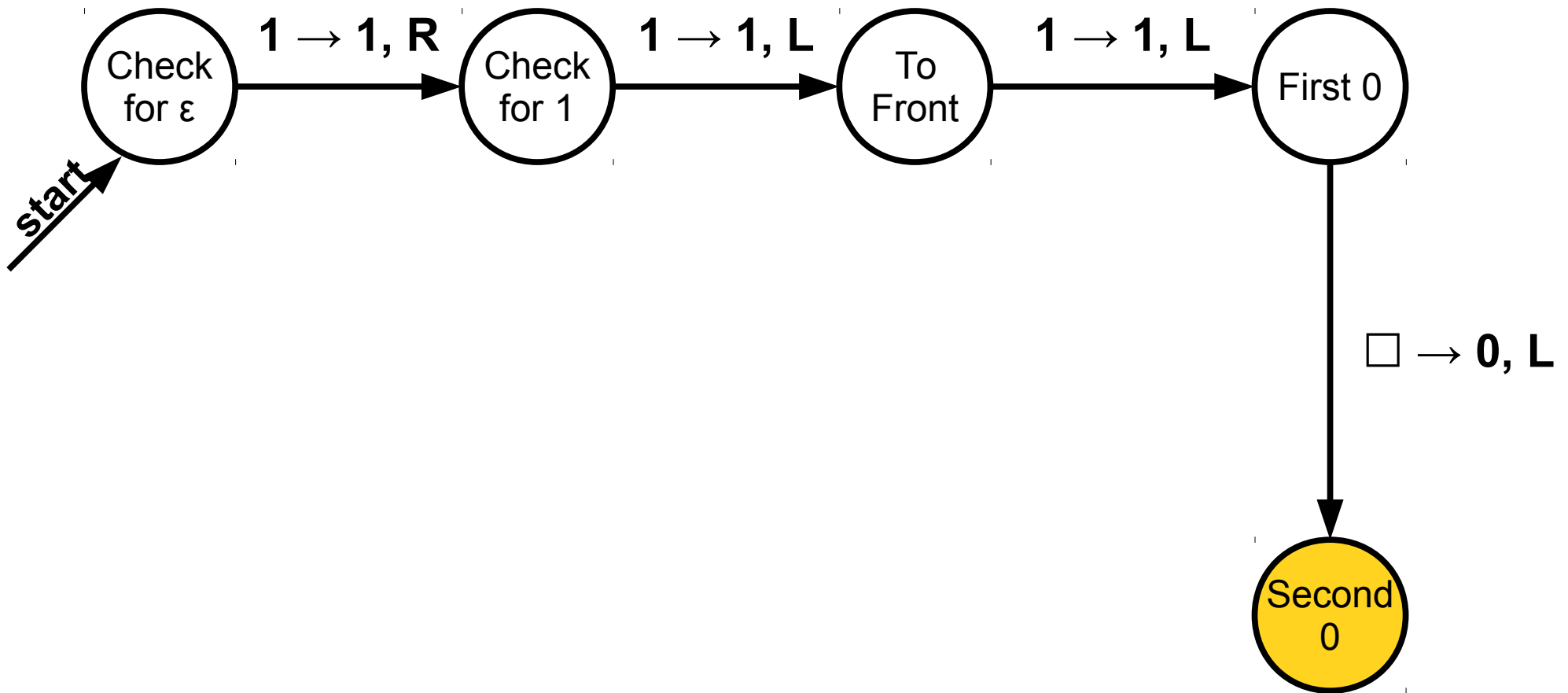


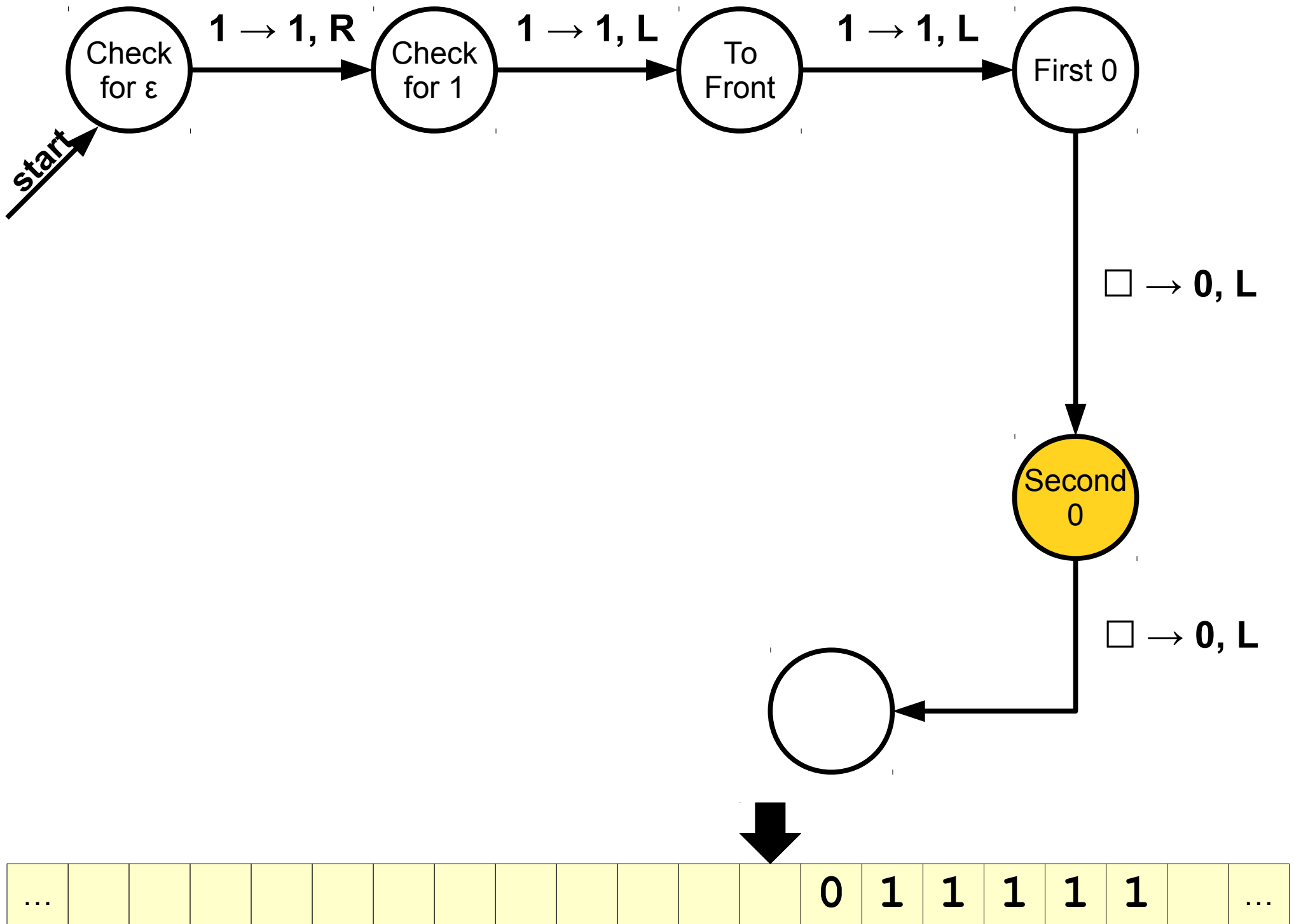


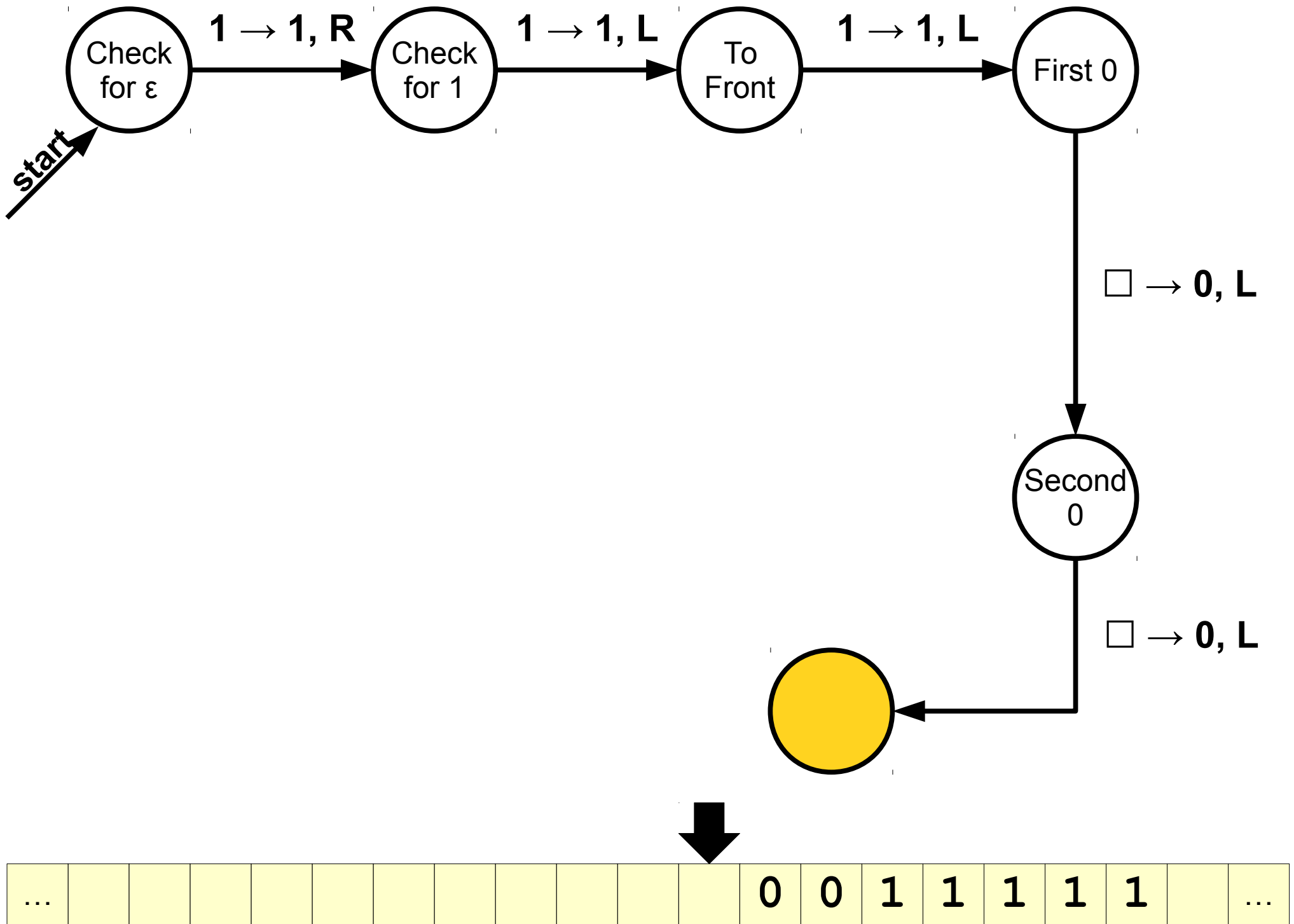


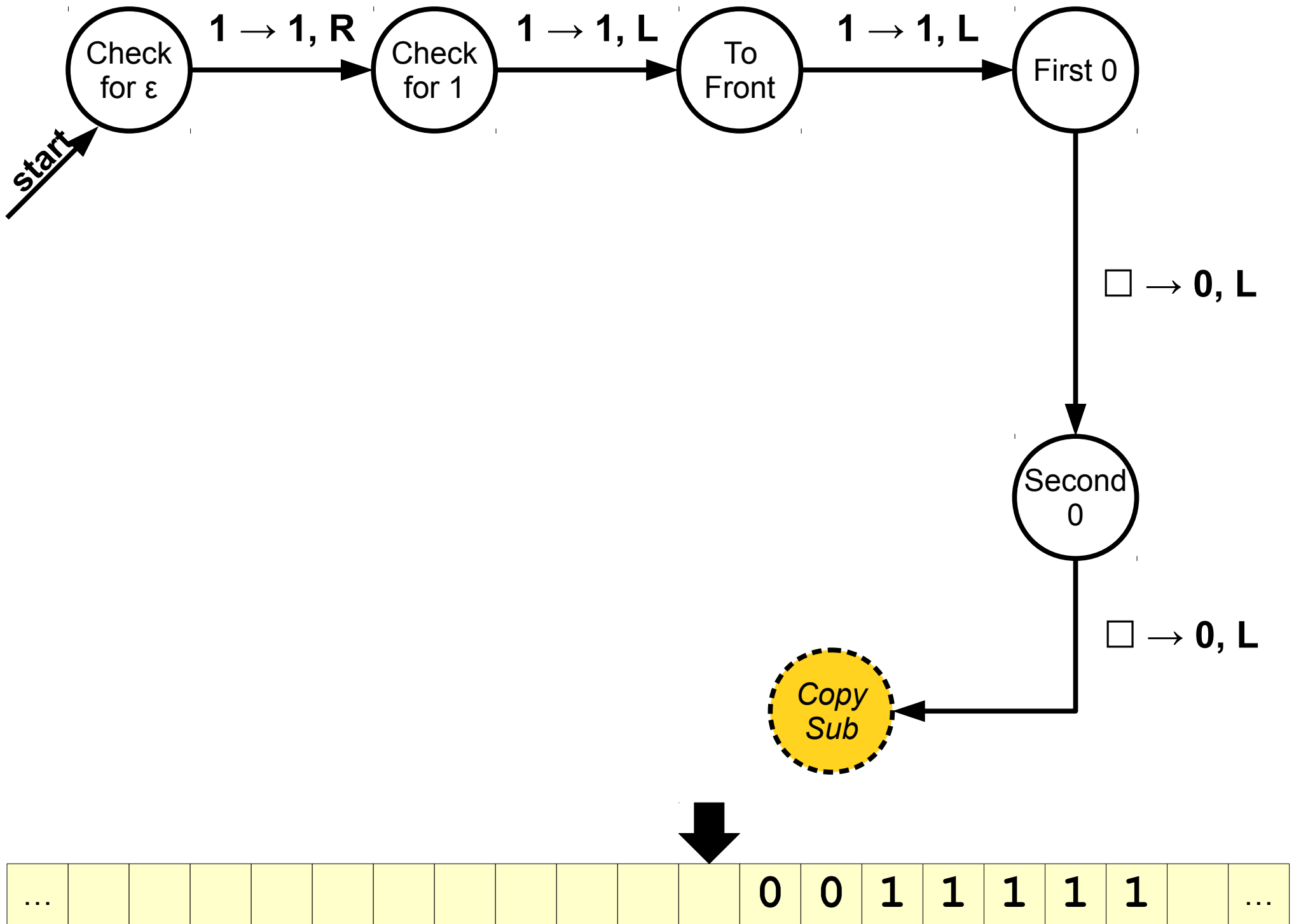


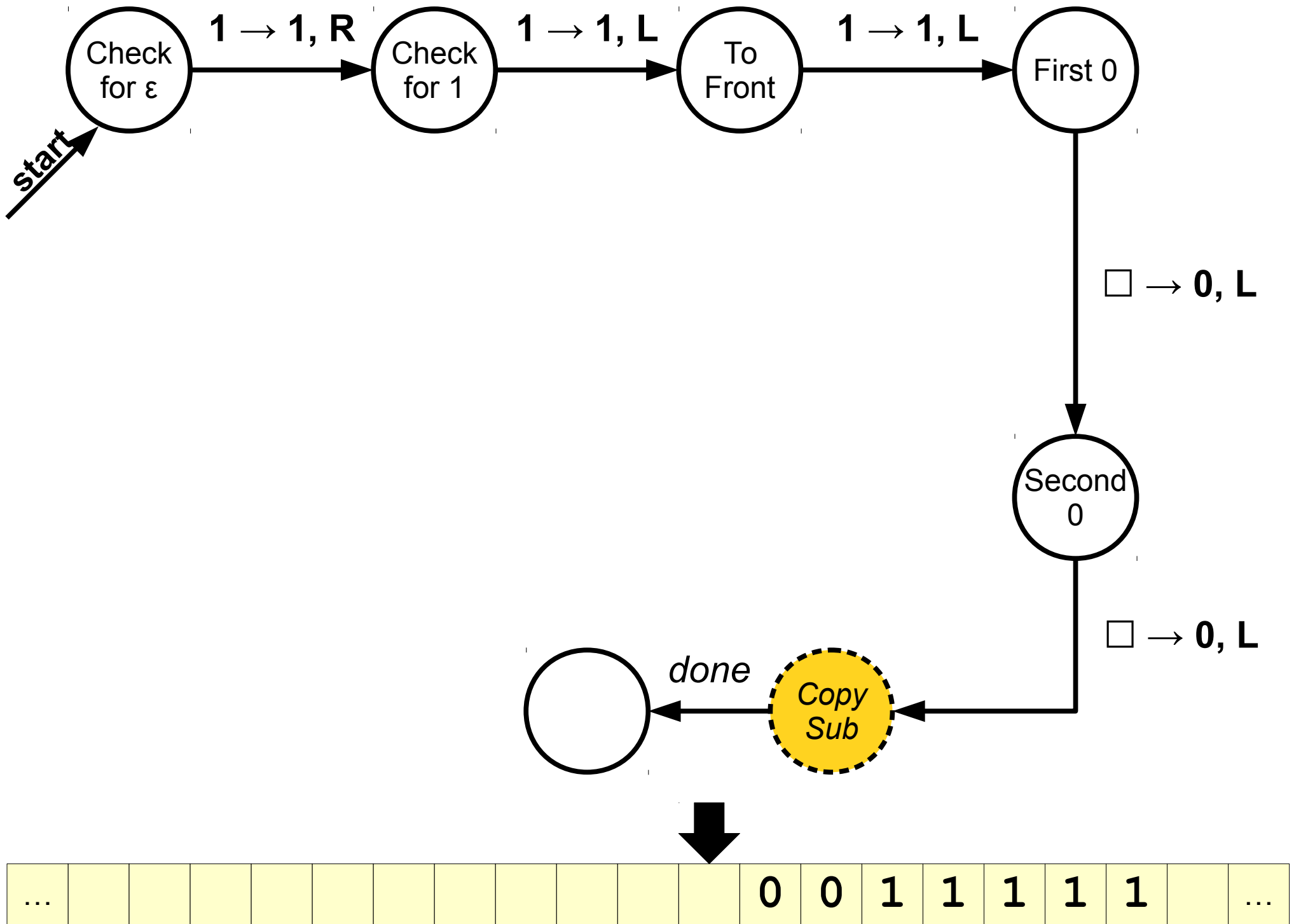


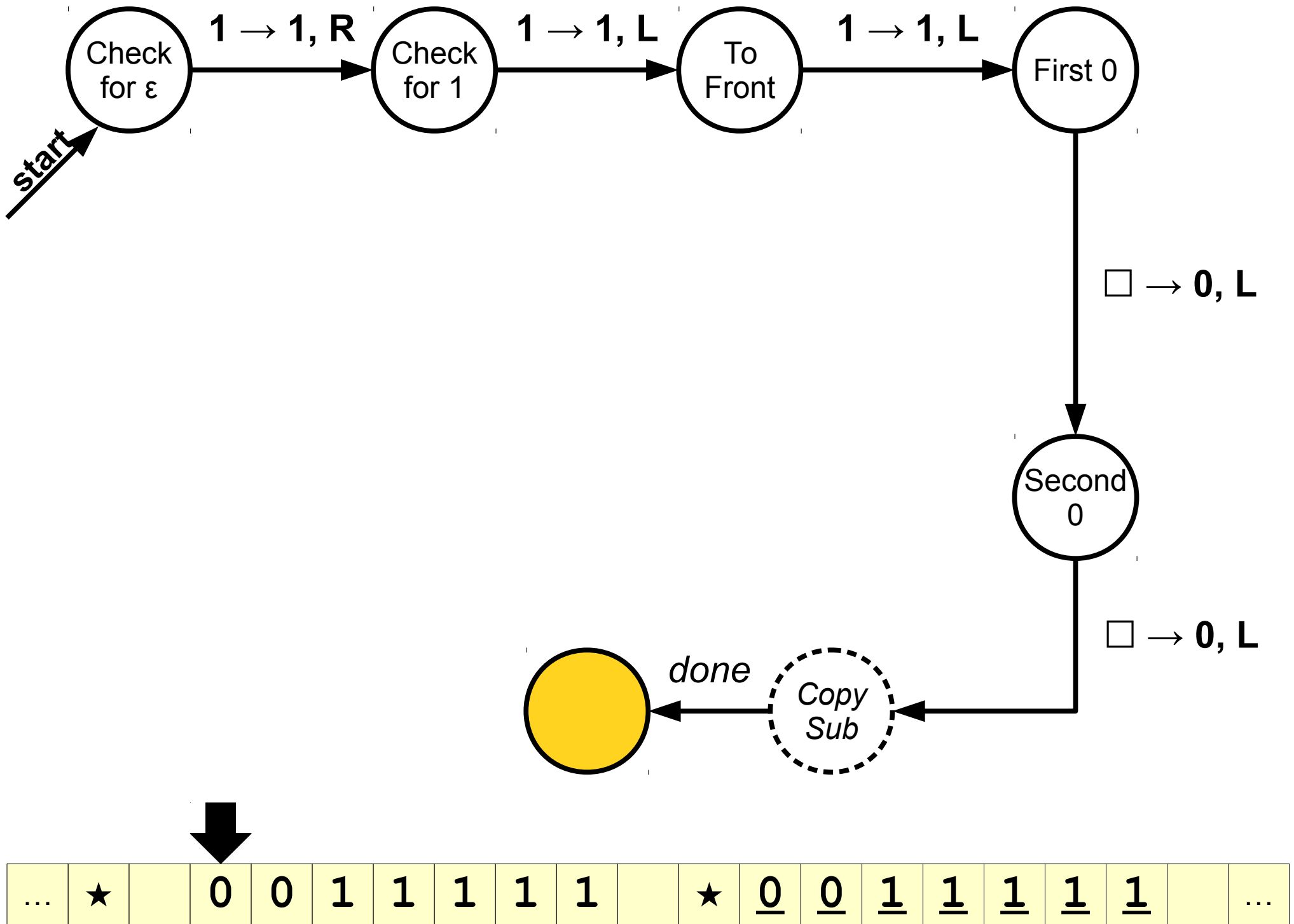


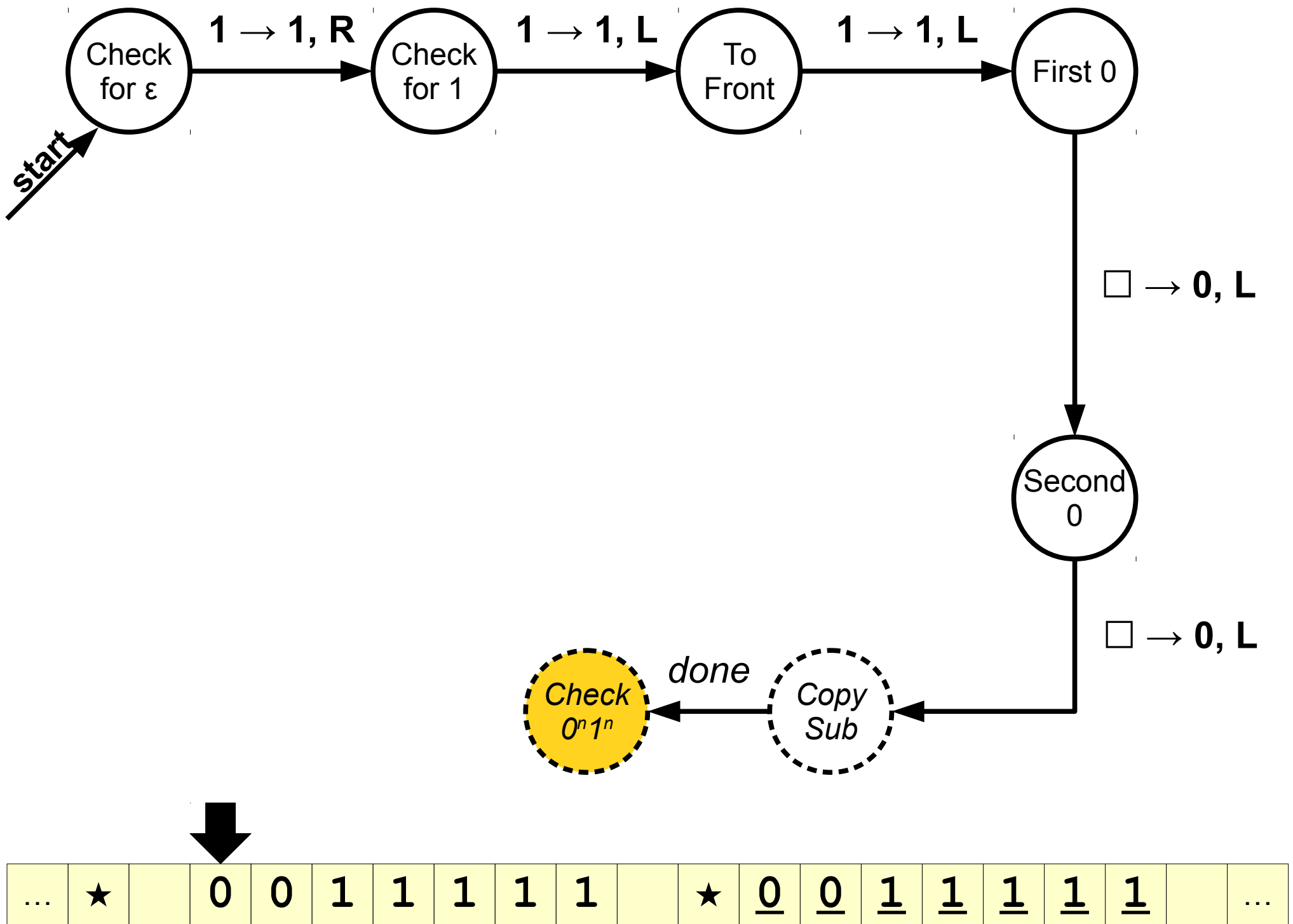


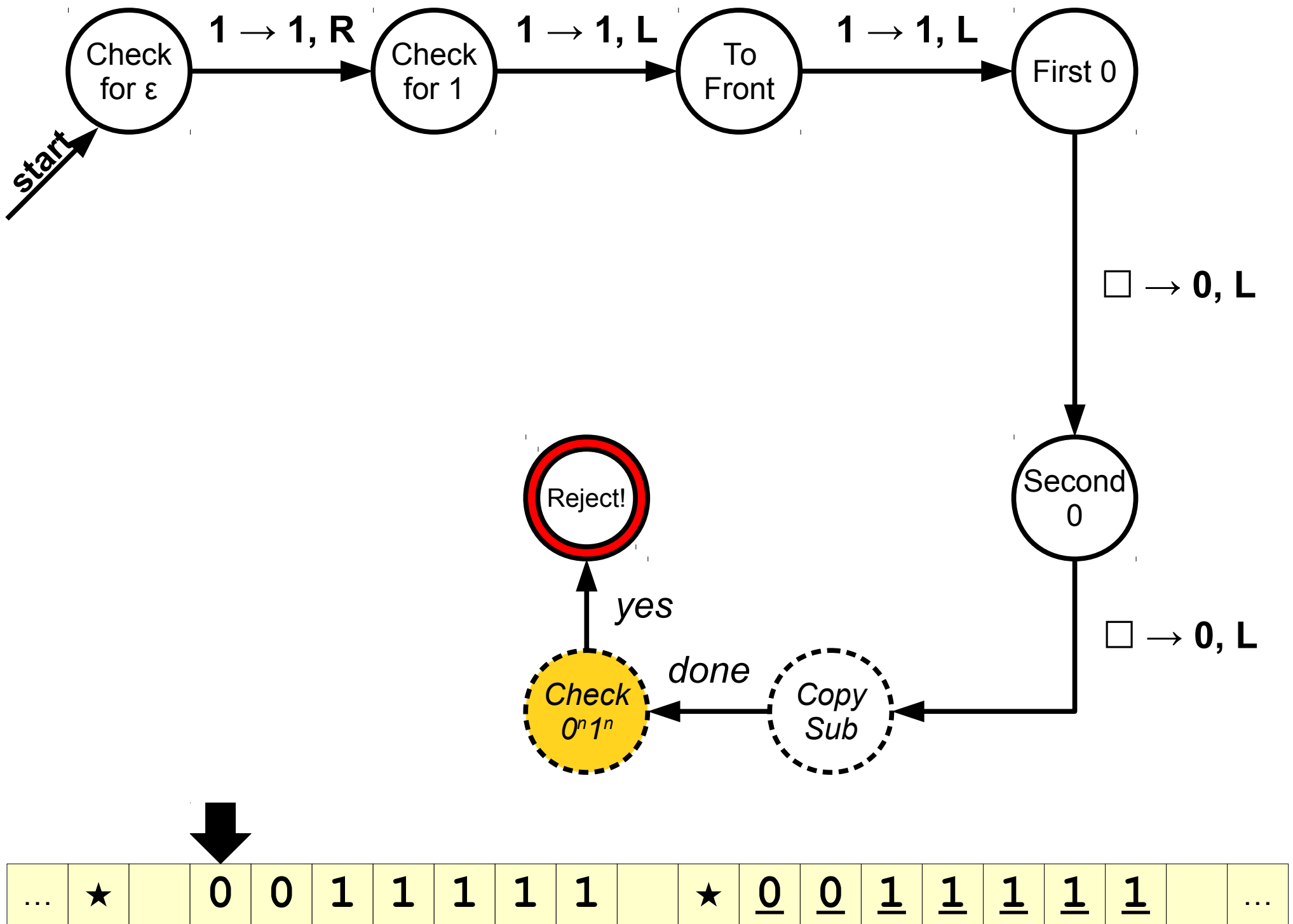


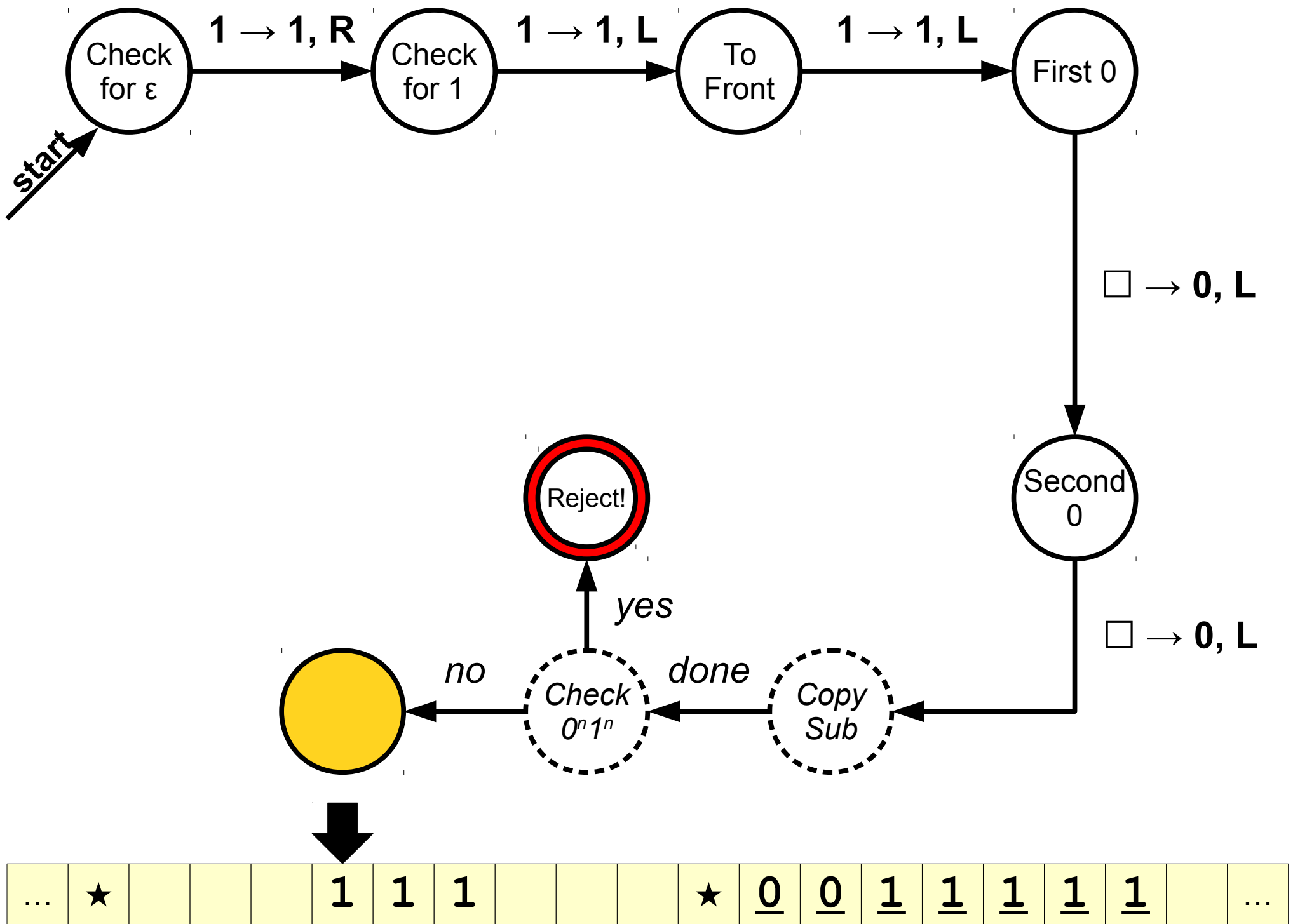


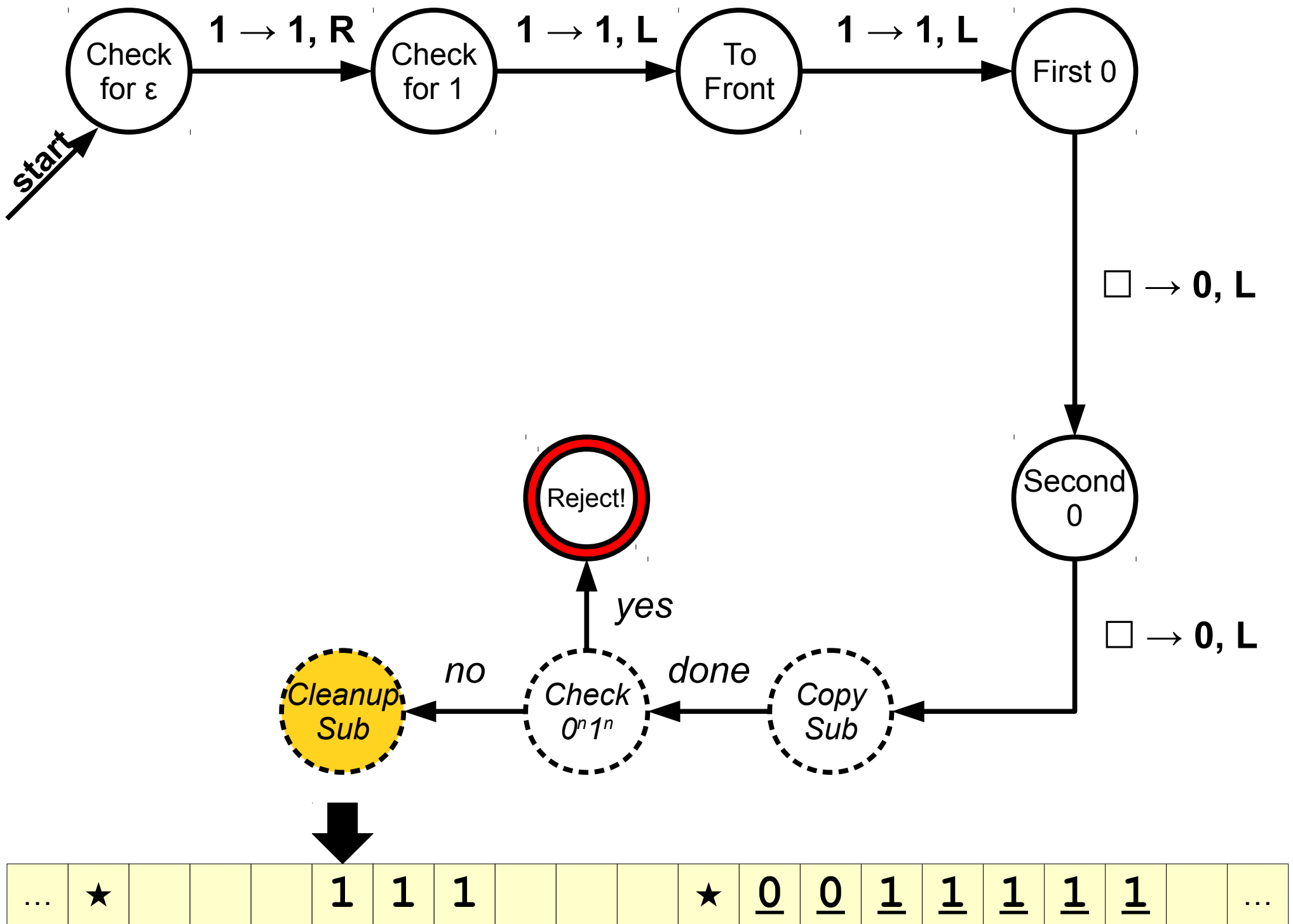


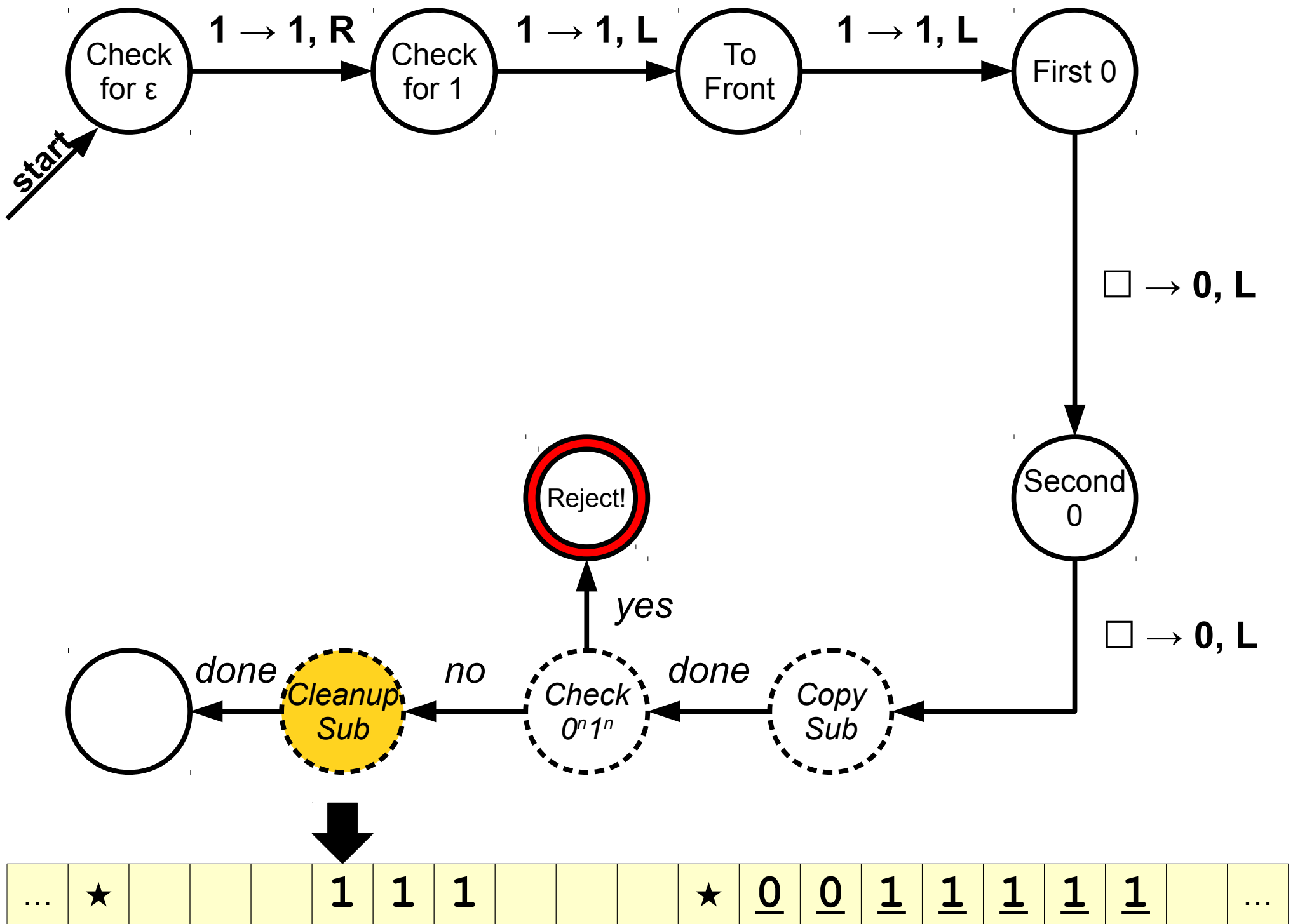


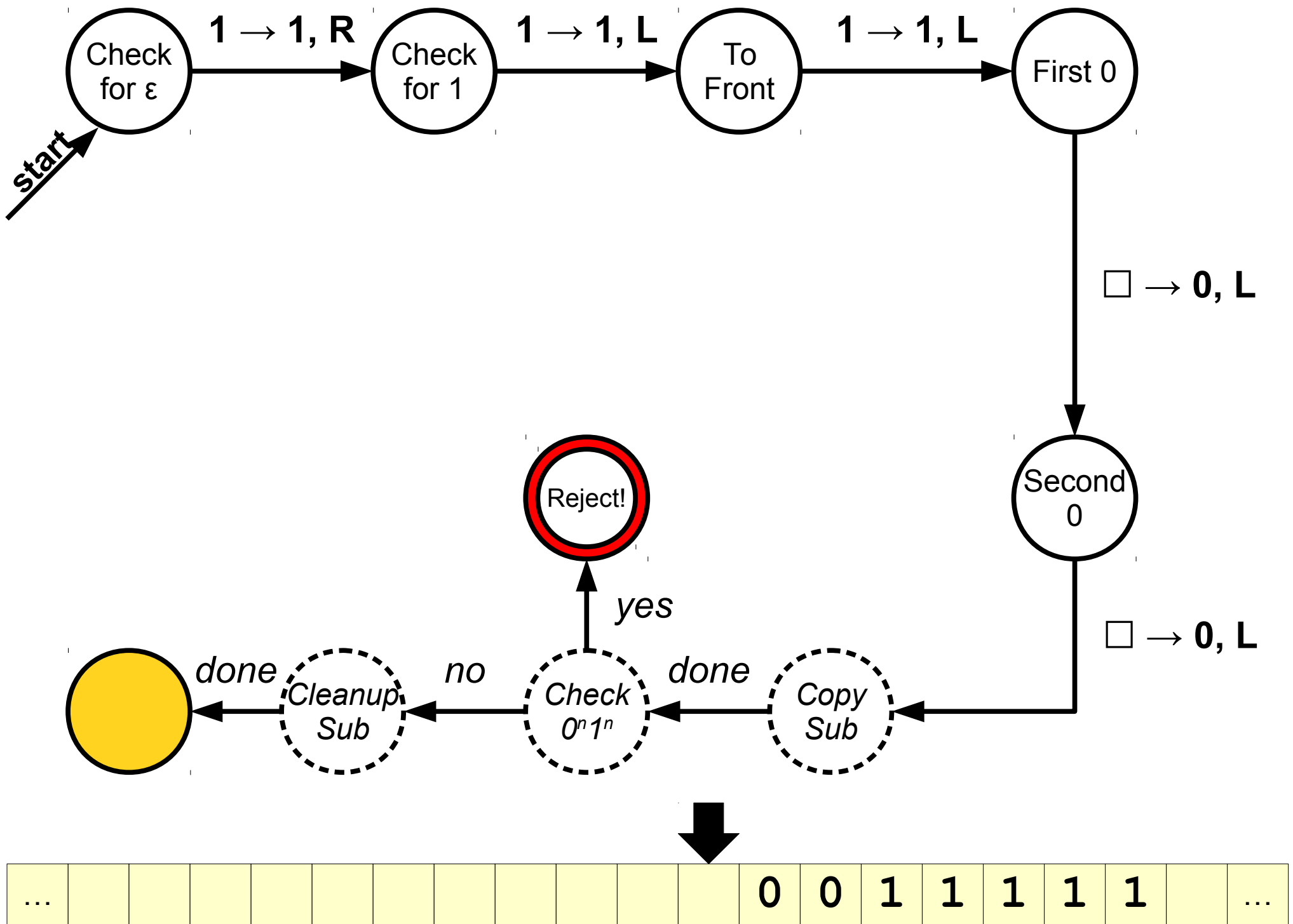


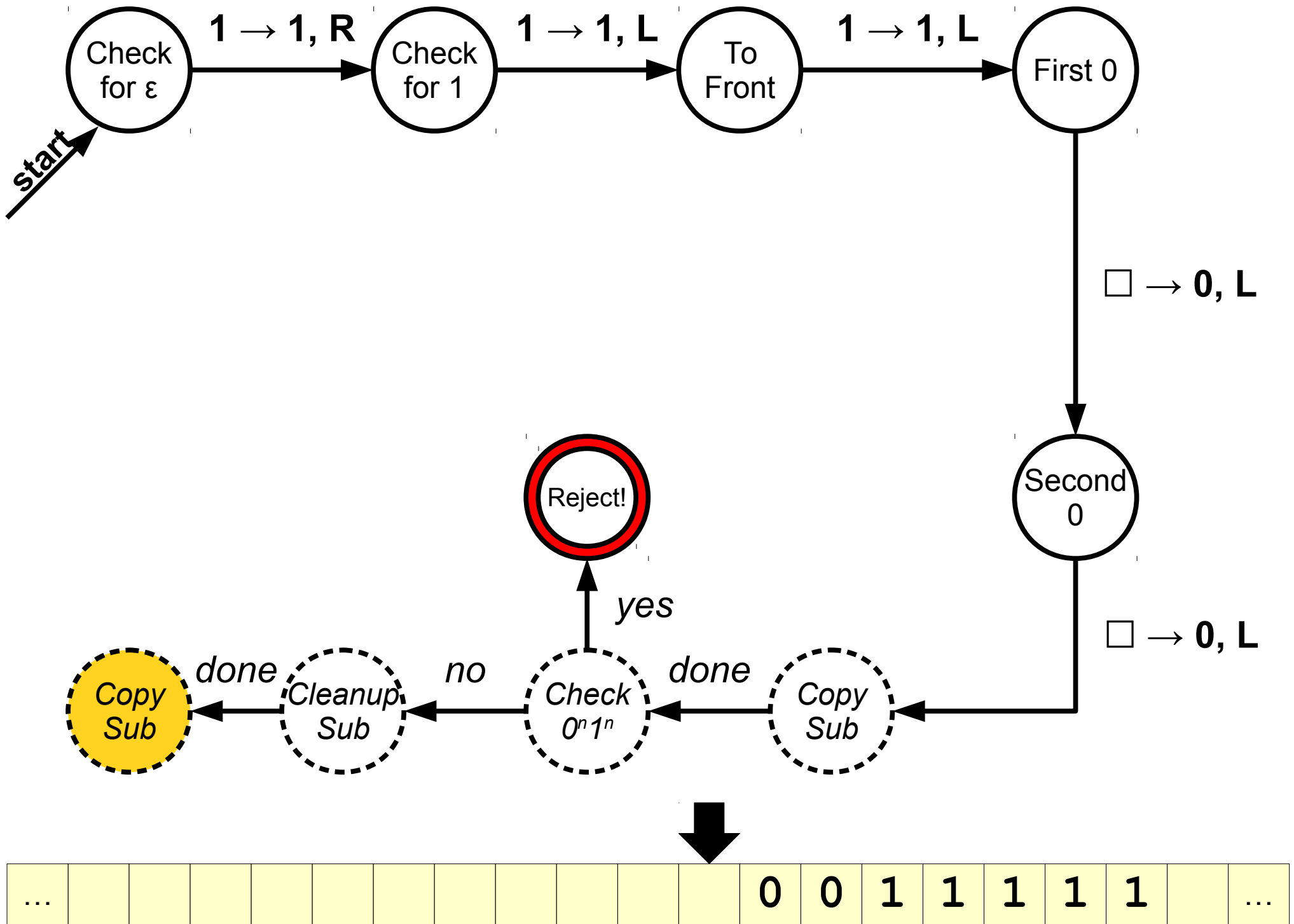


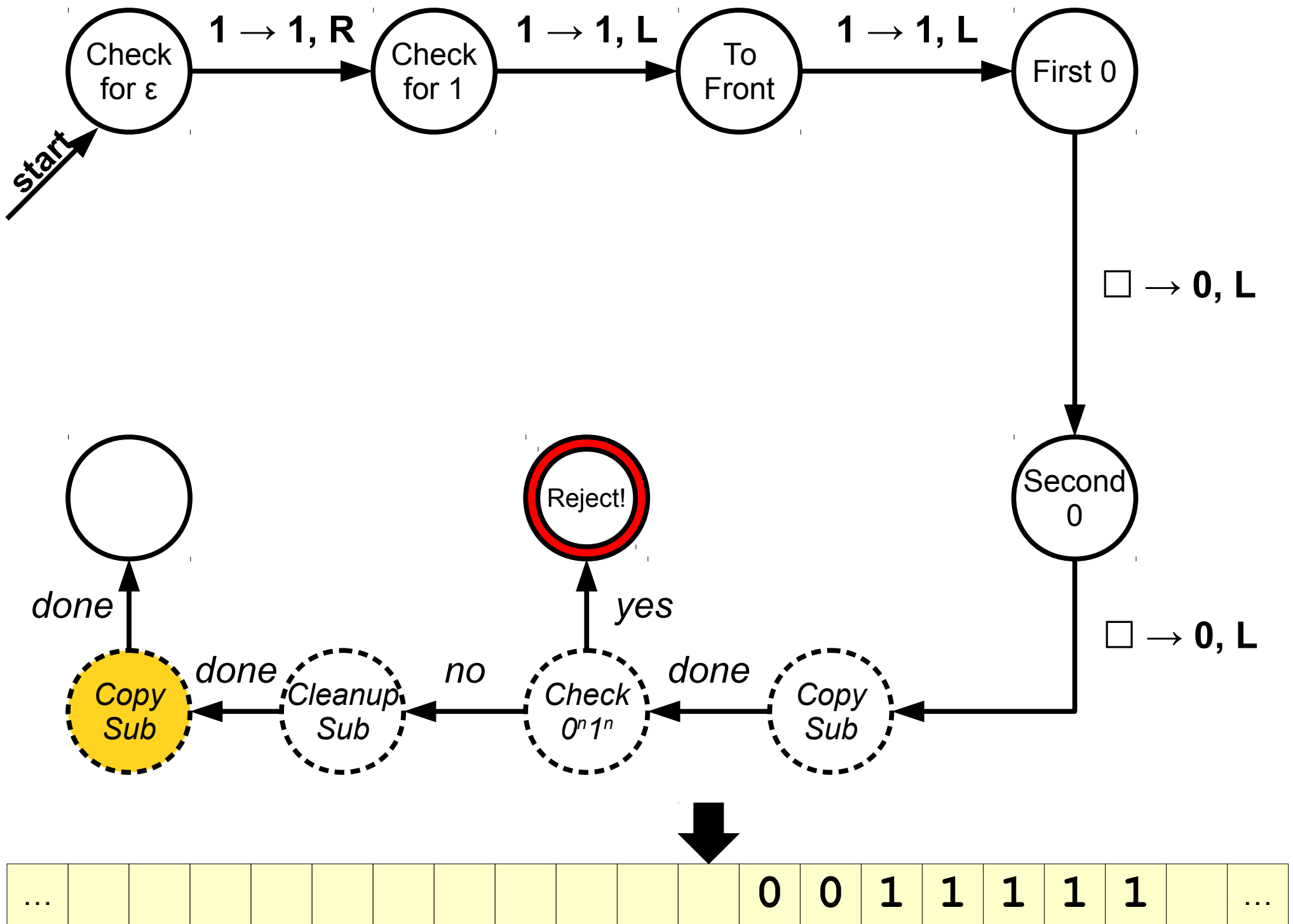


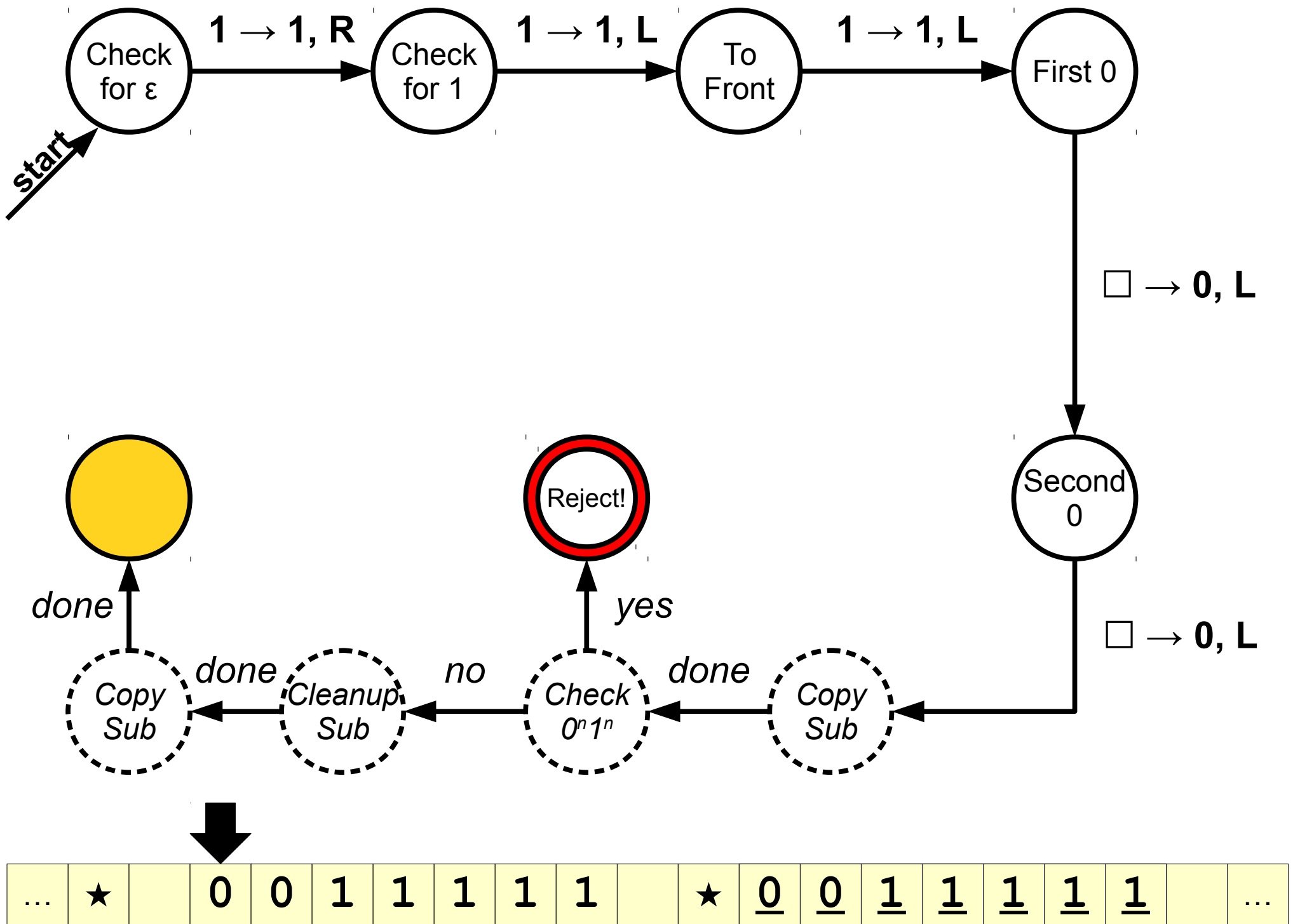


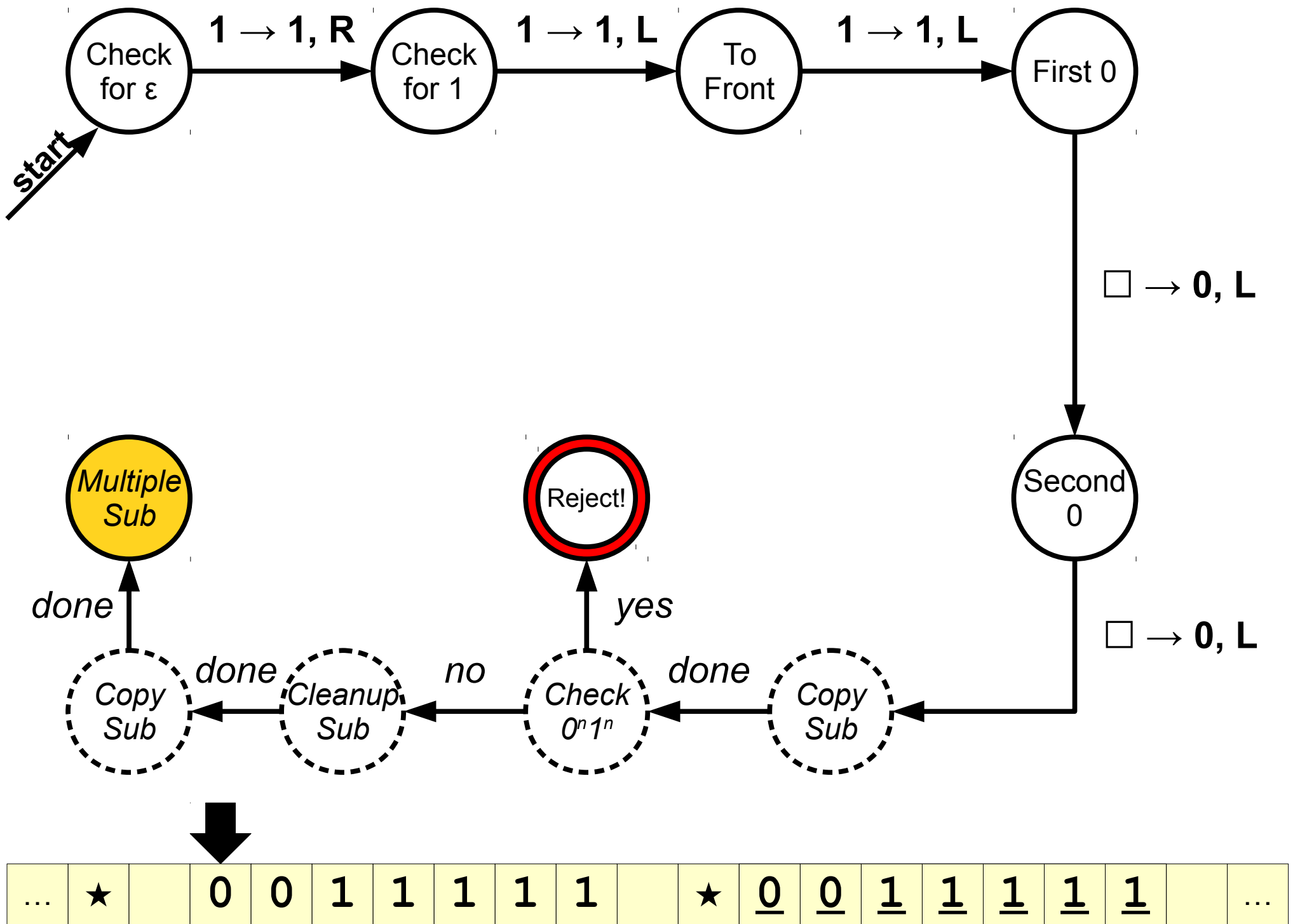


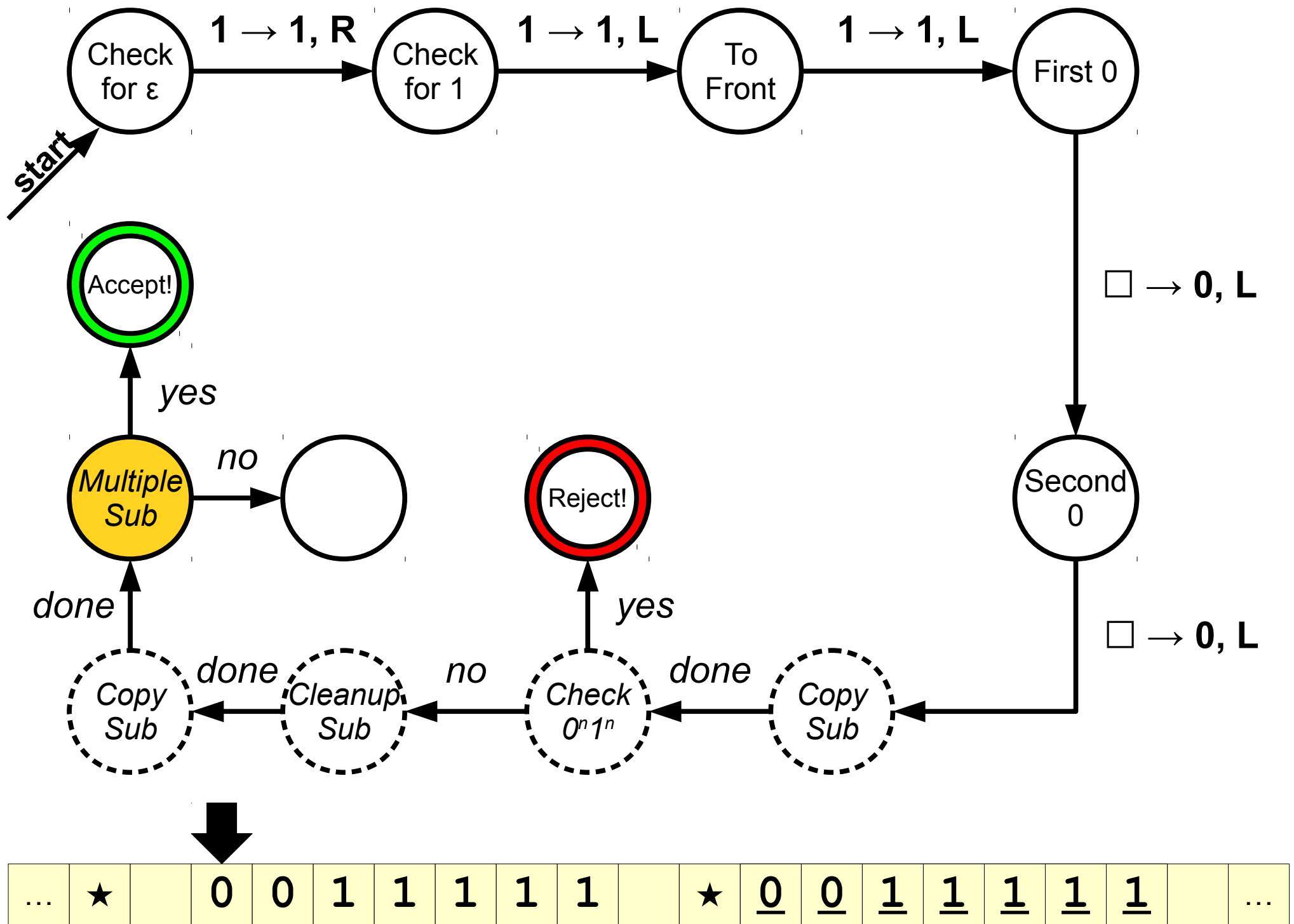


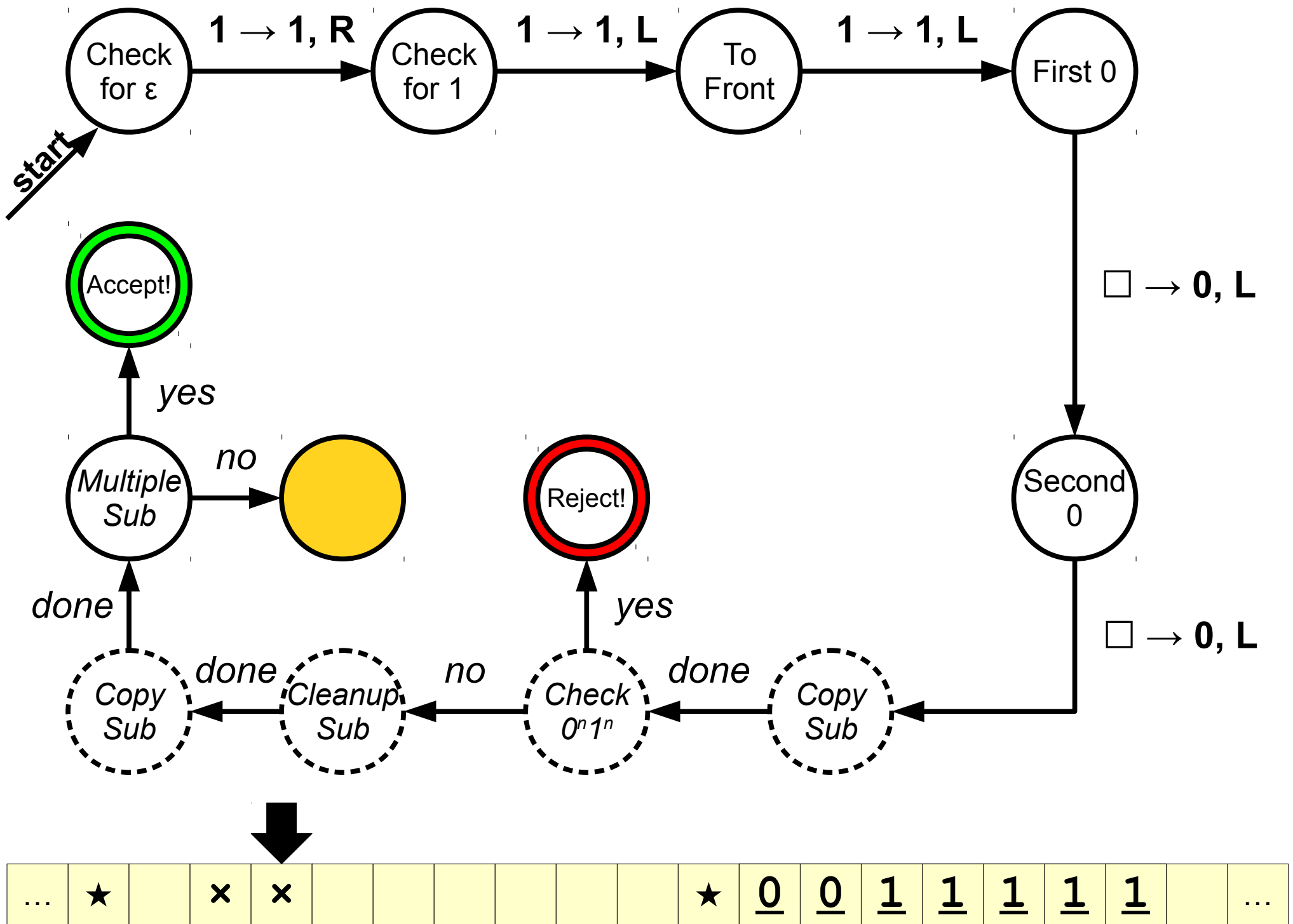


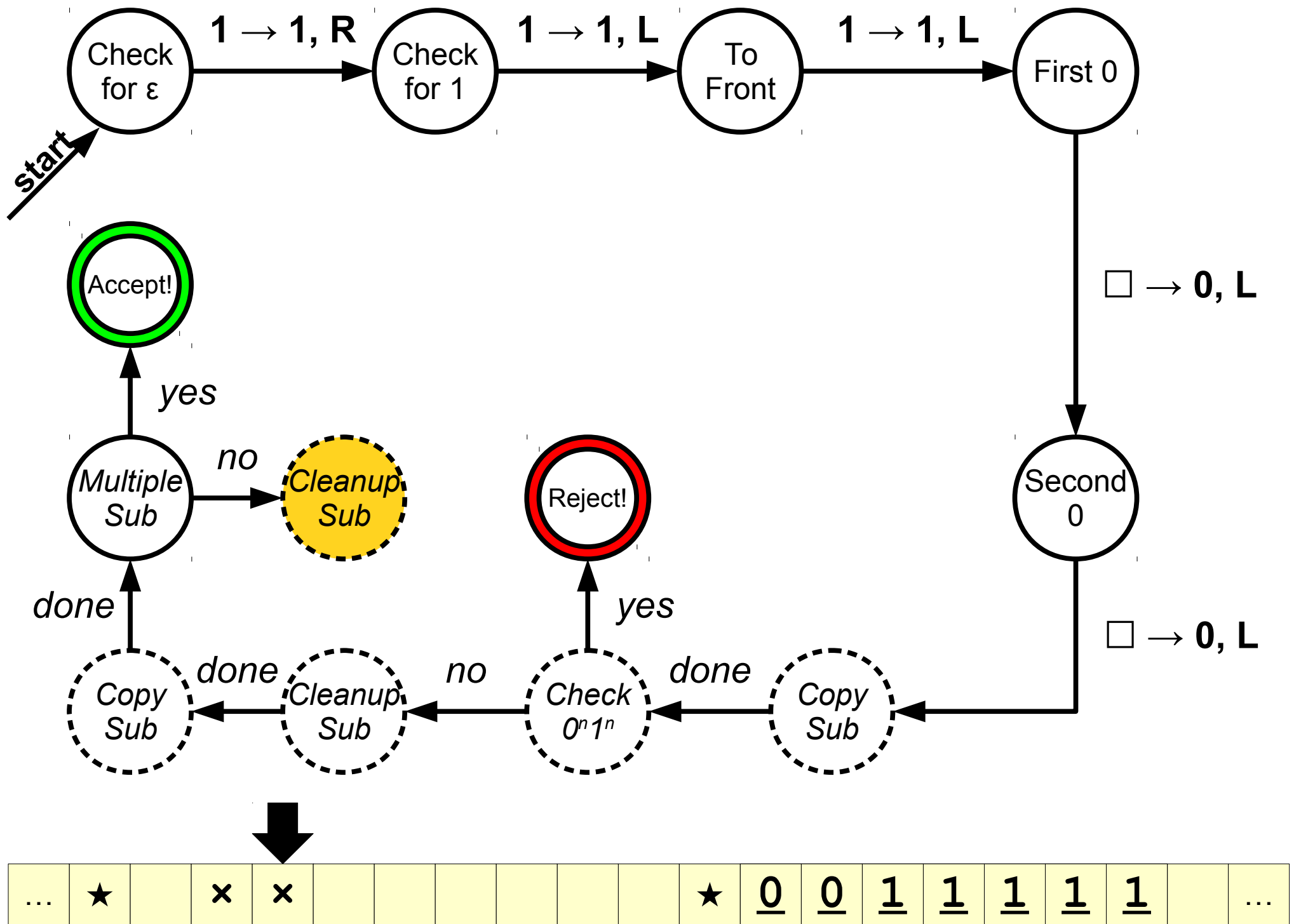


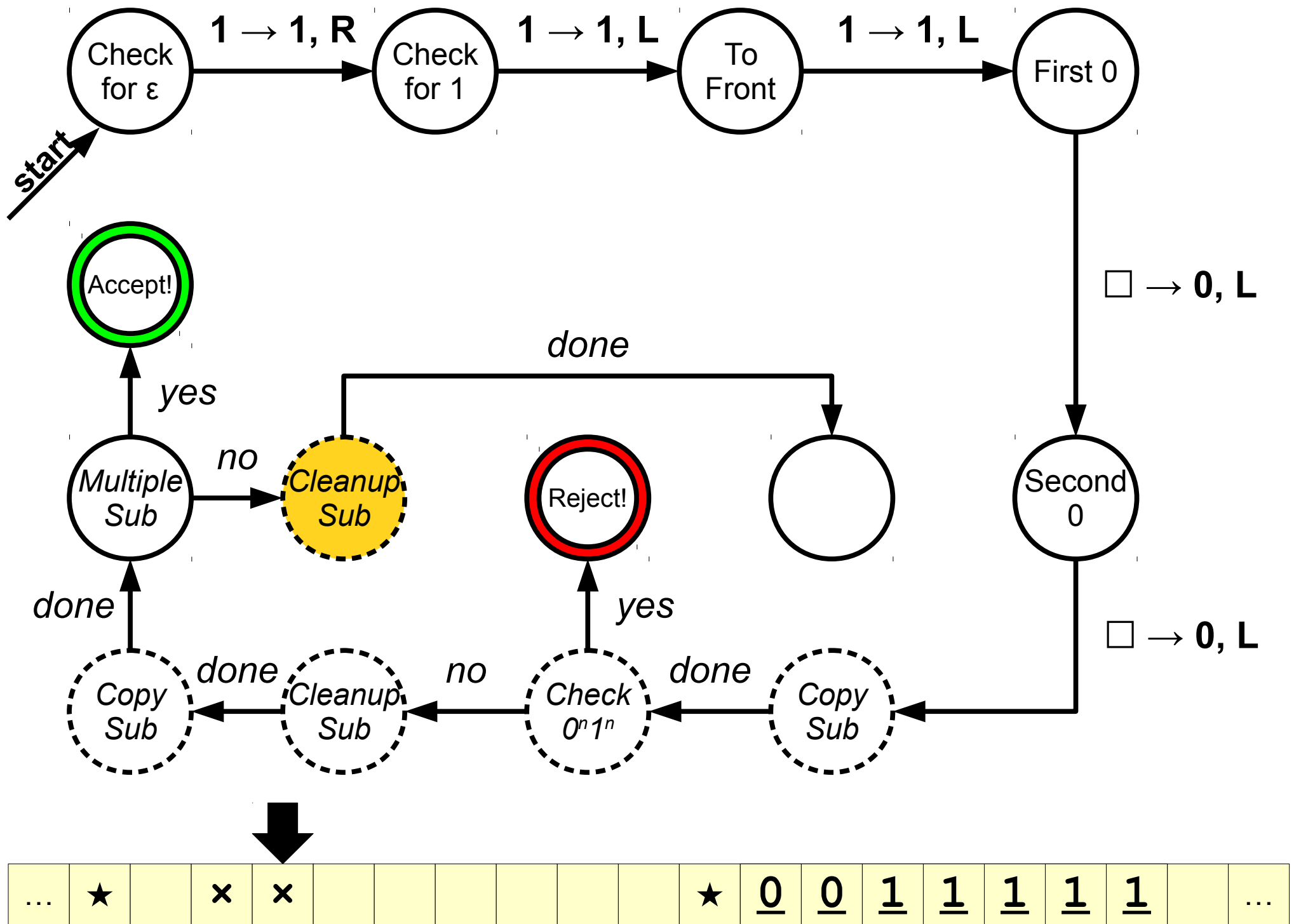


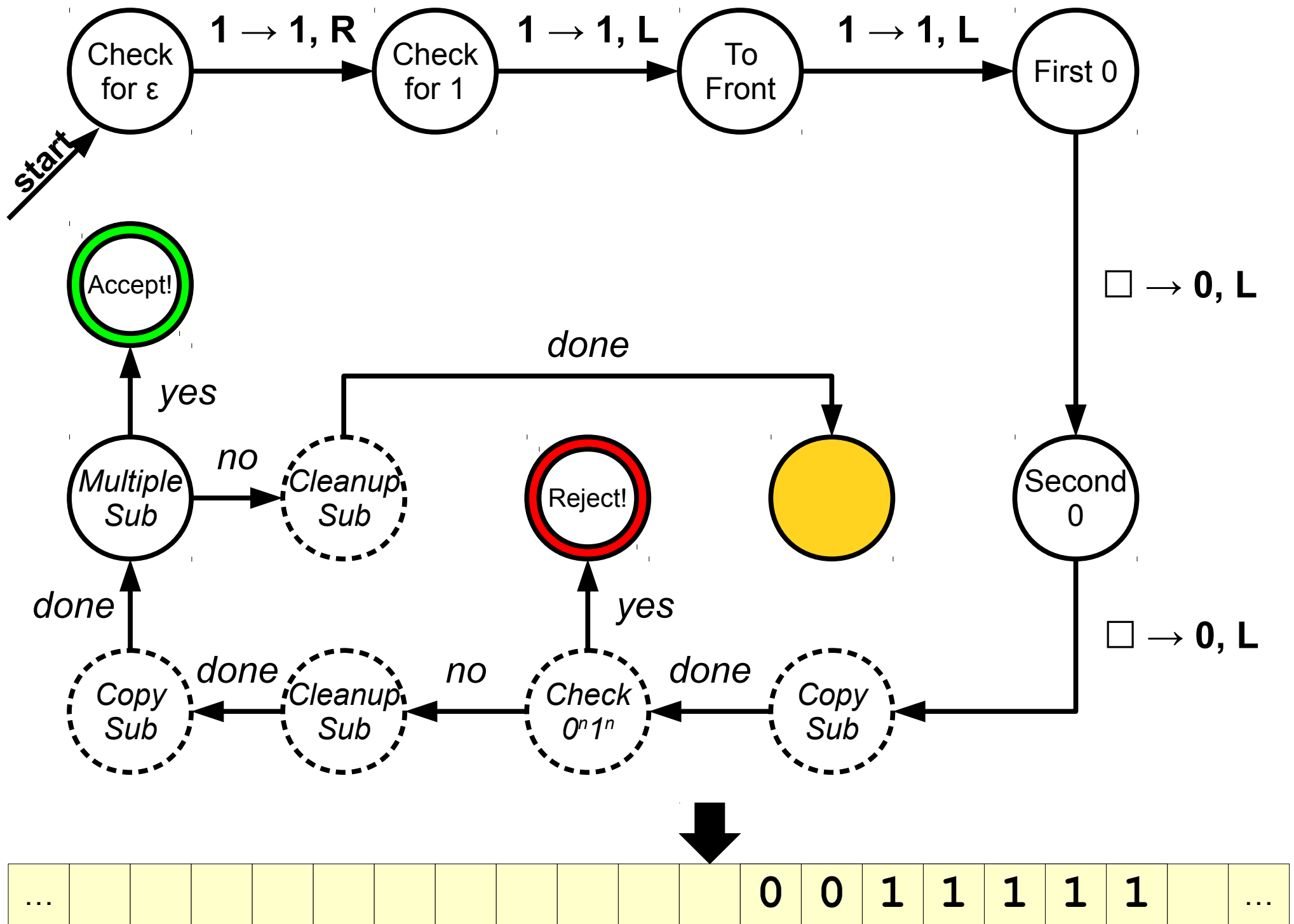


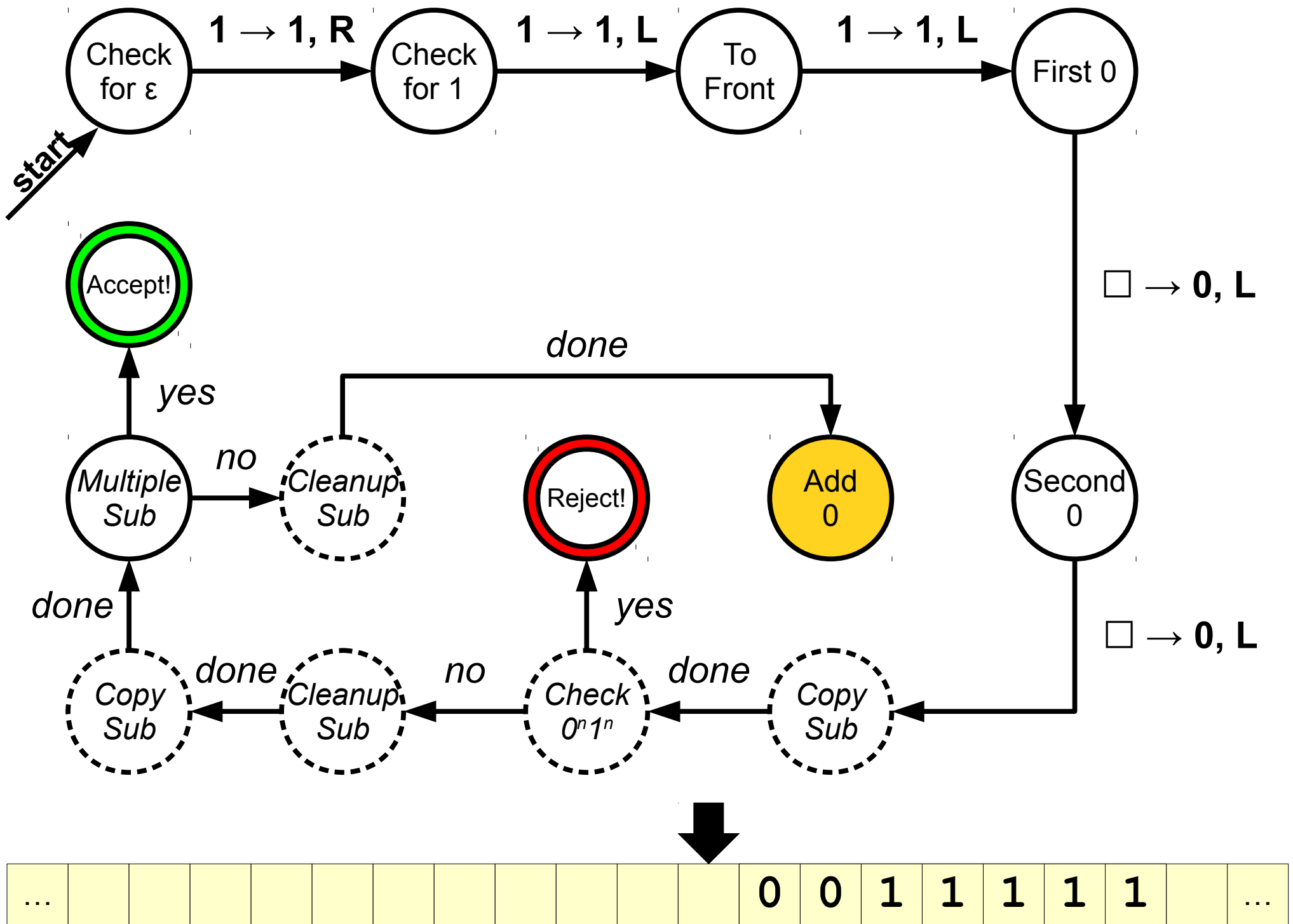


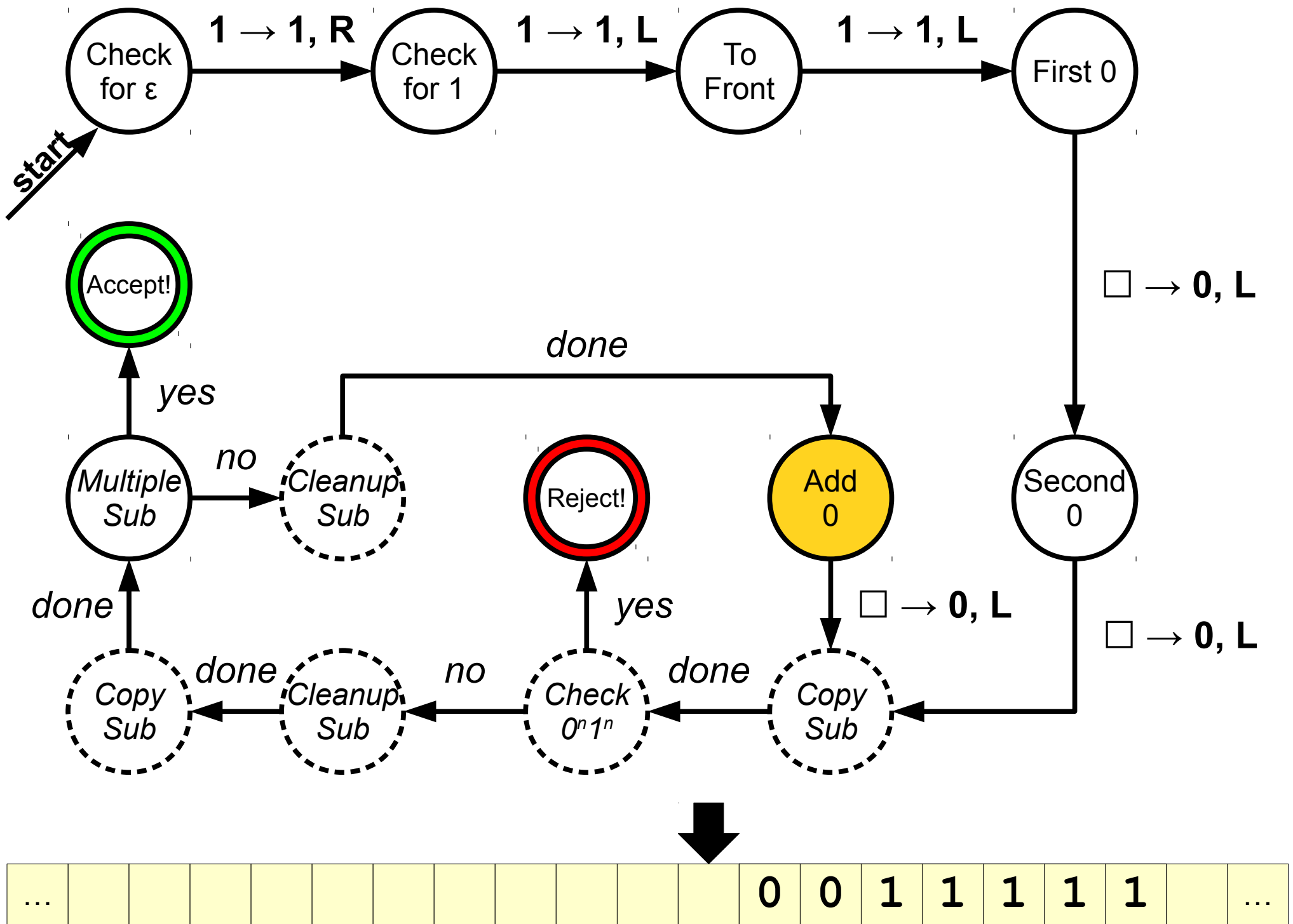


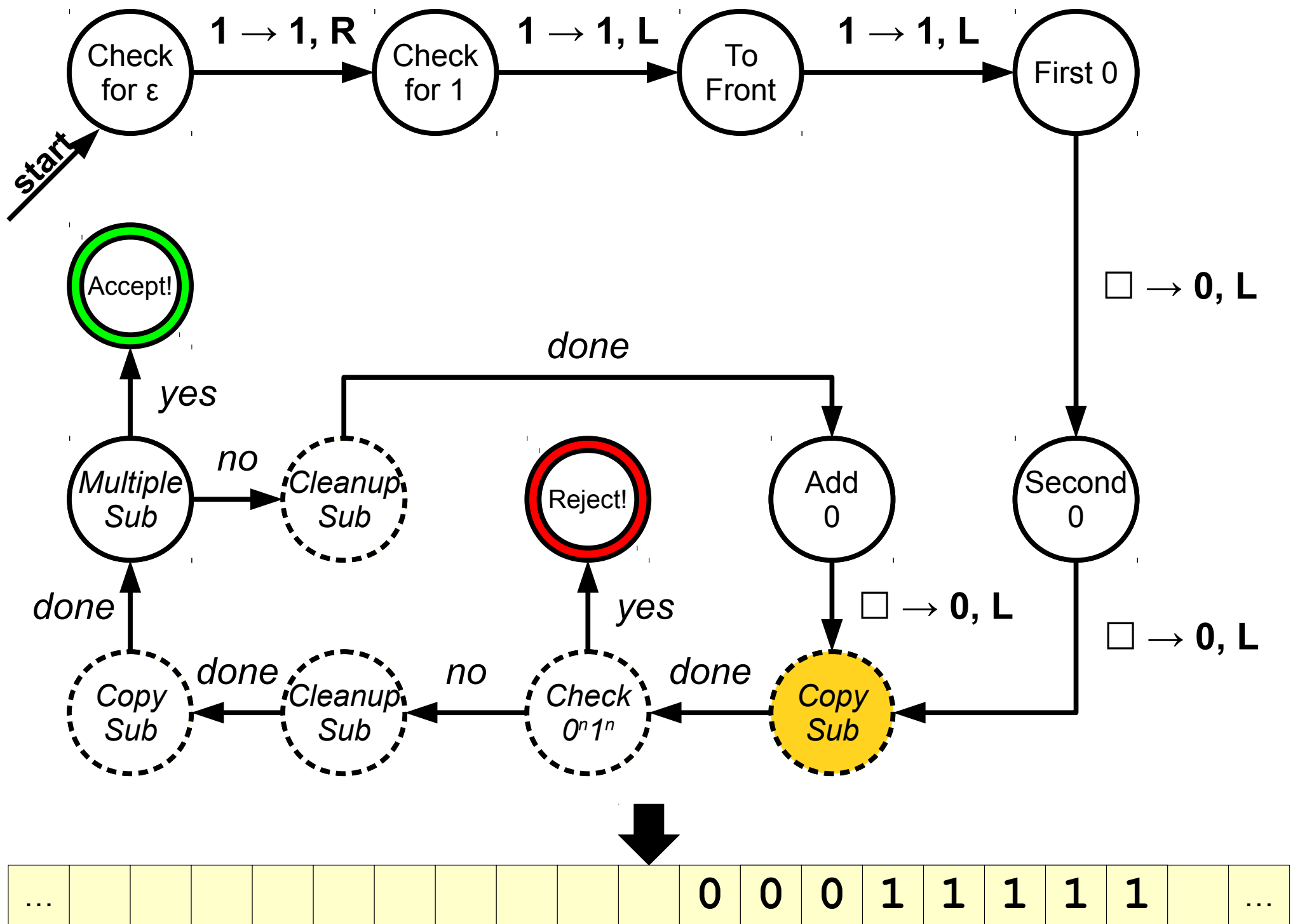


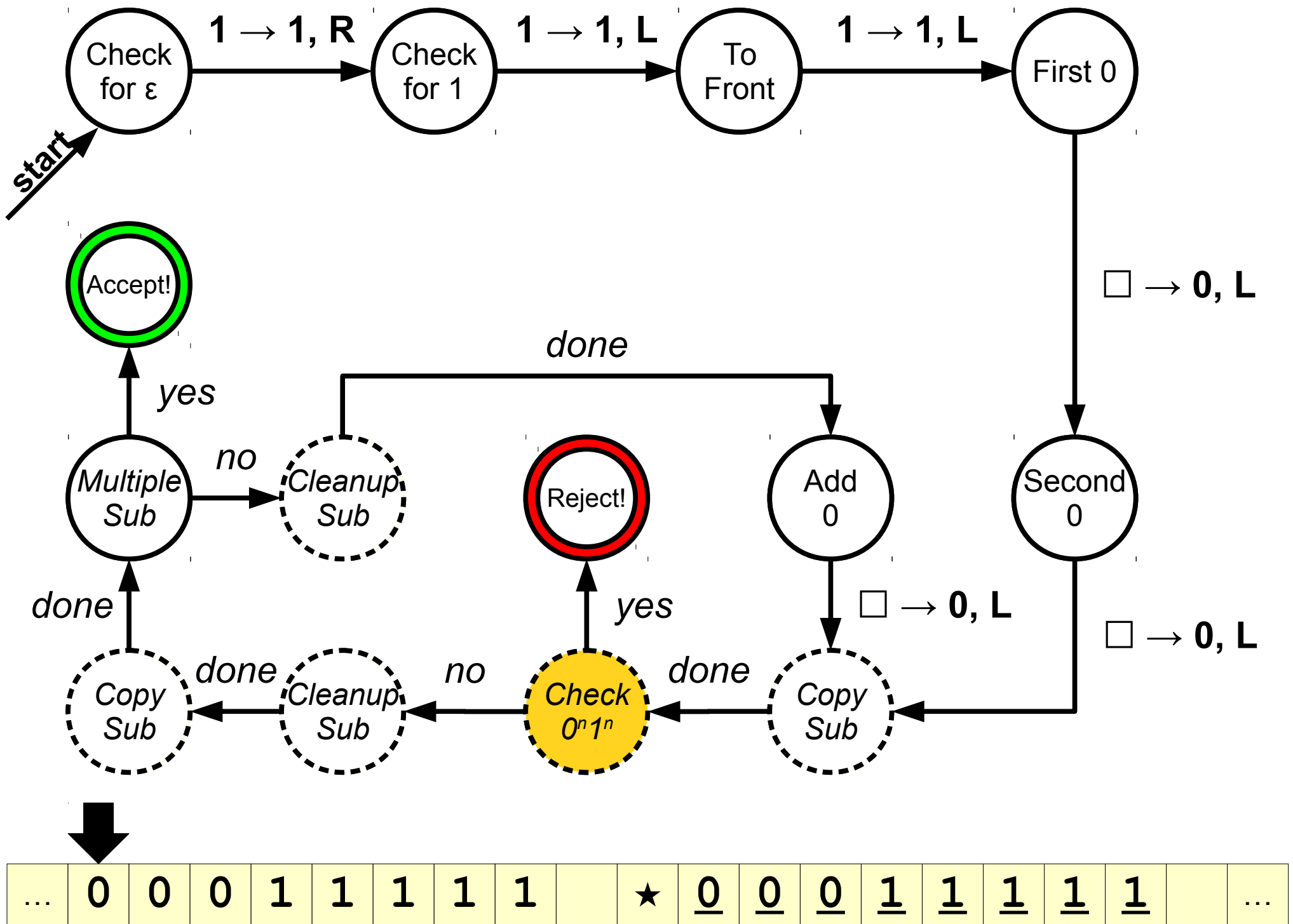


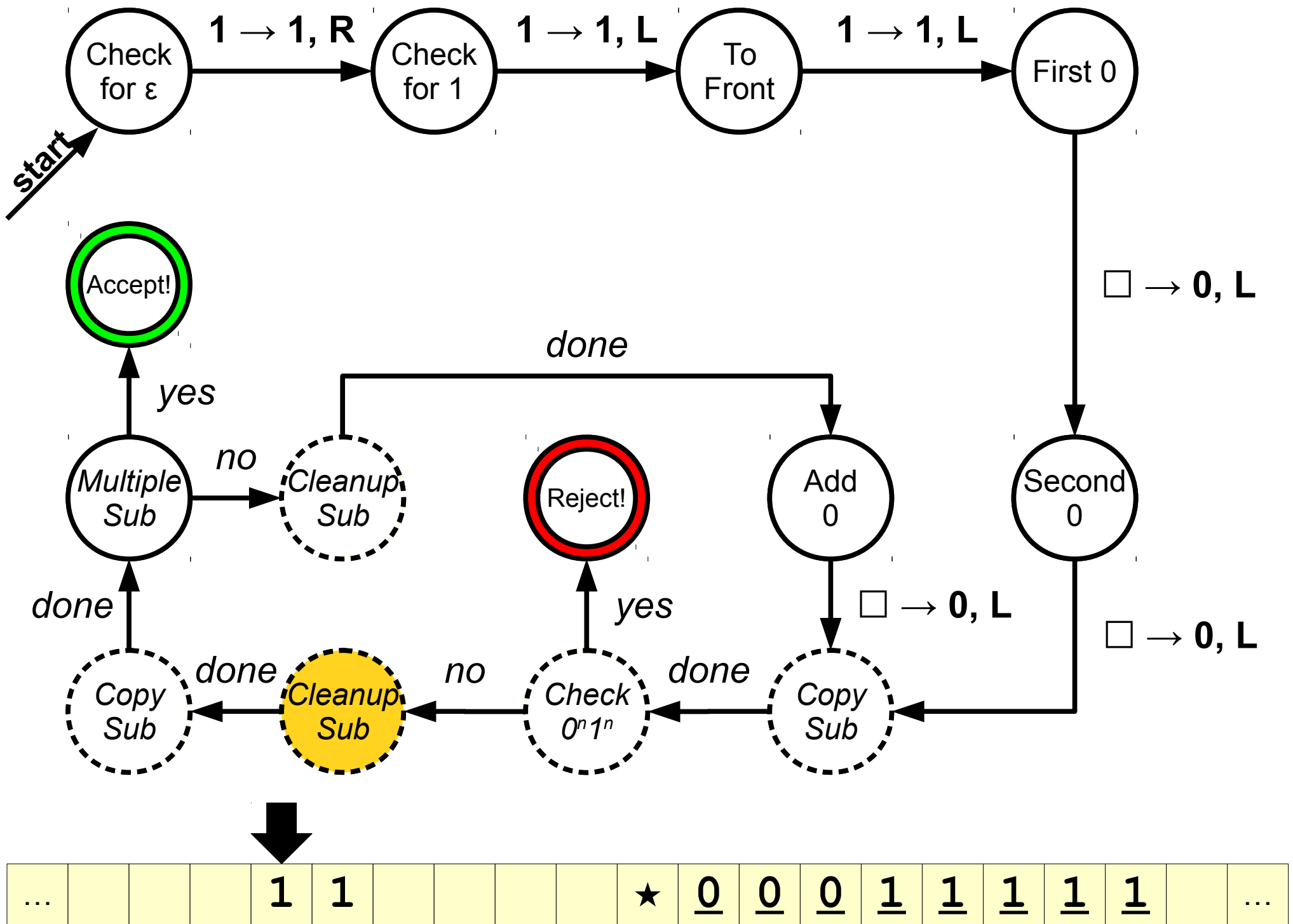


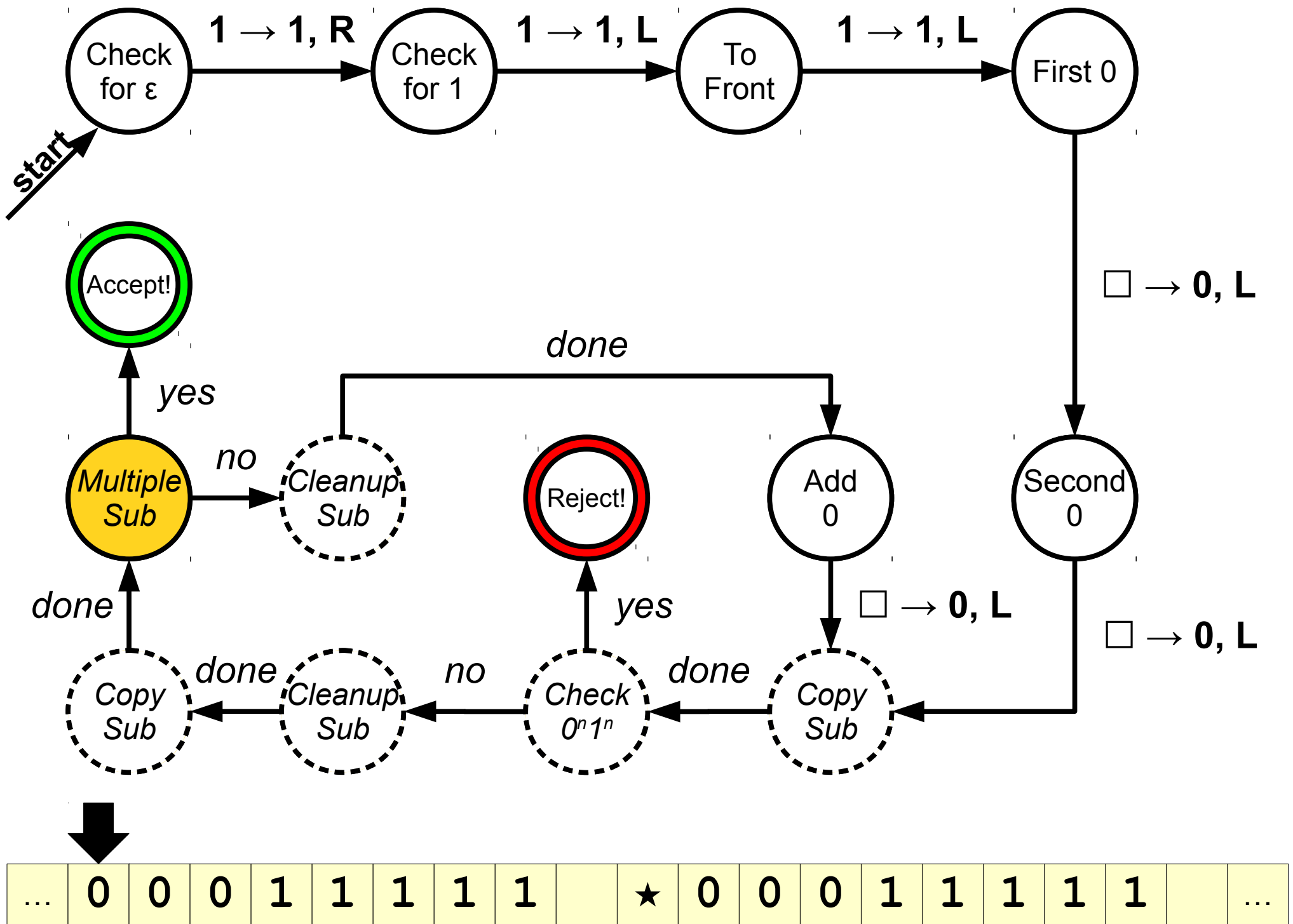


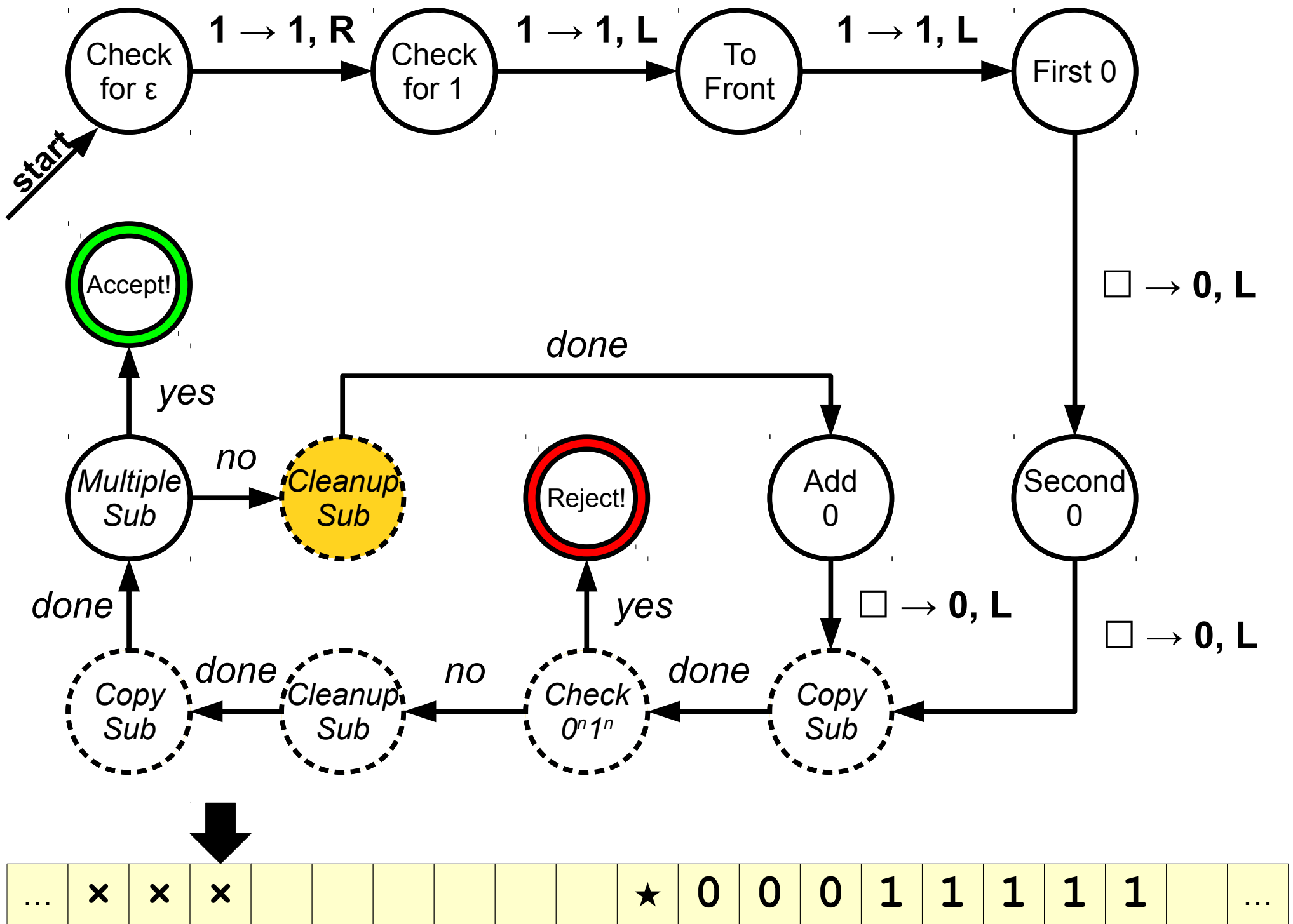


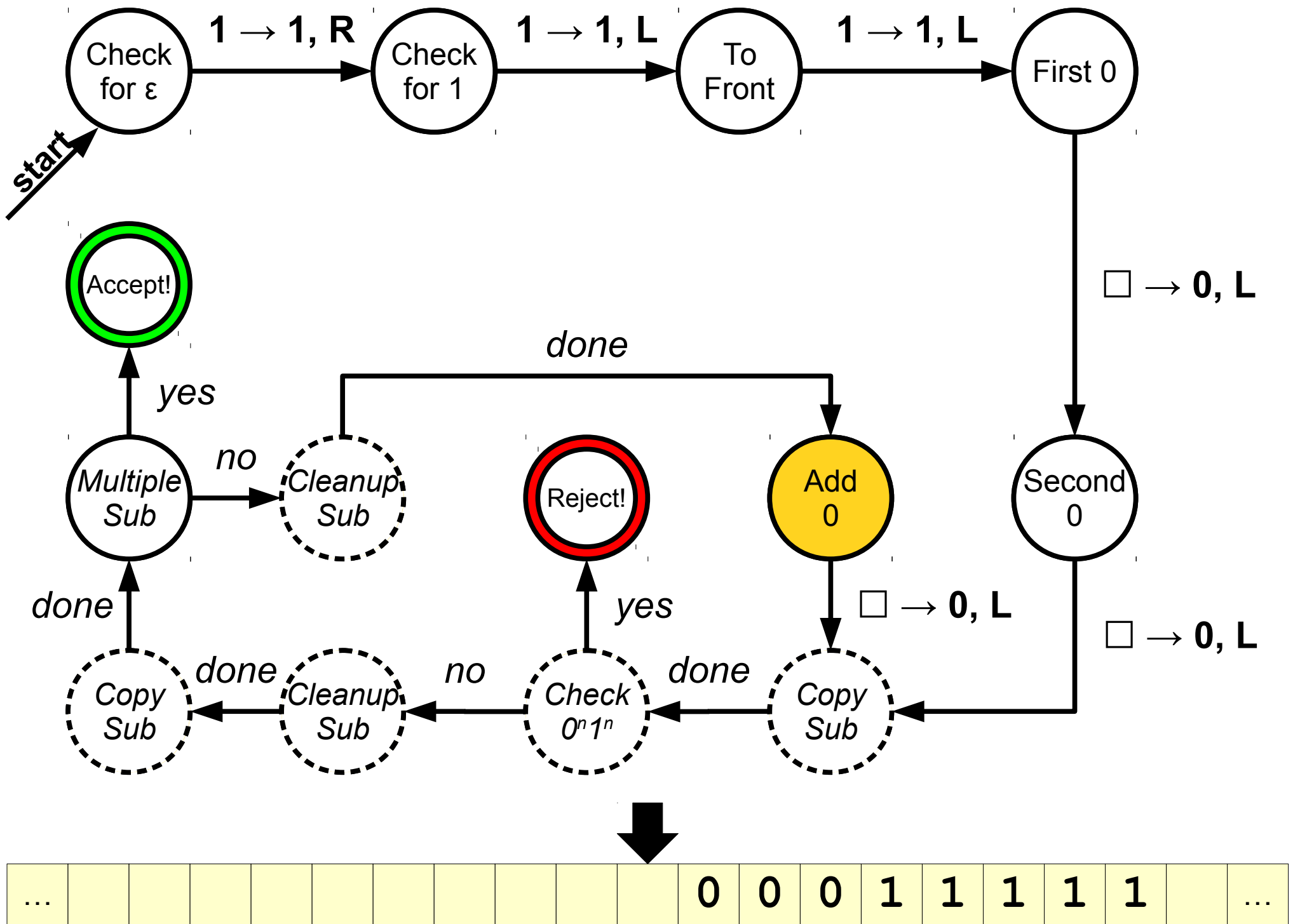


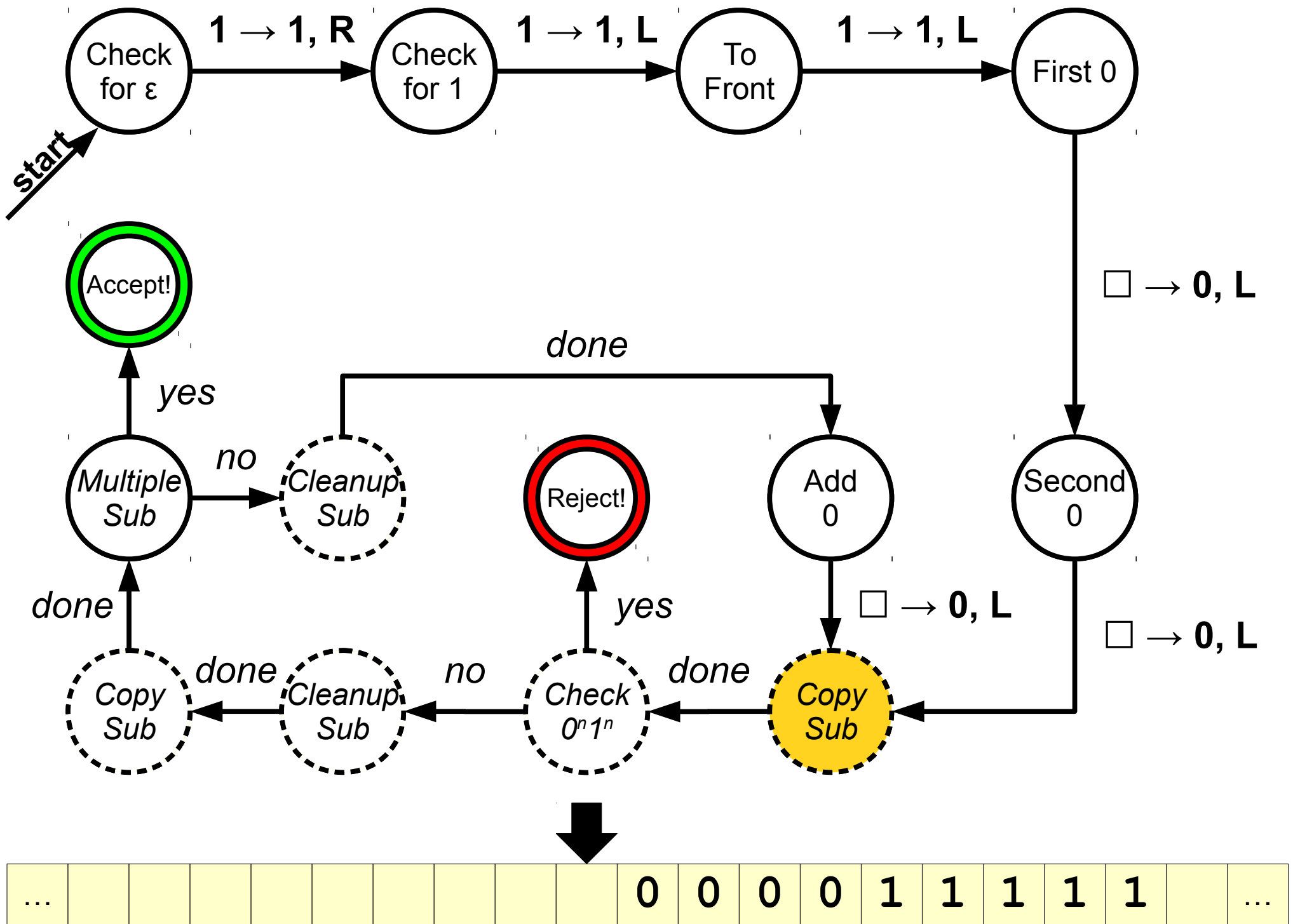










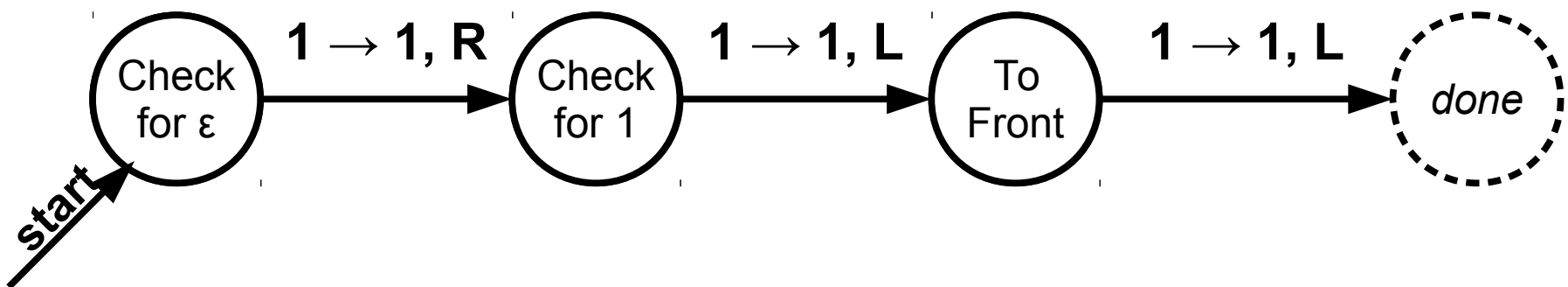


TM Subroutines

- To represent a subroutine call inside a TM, just add a state with dashed edges with the name of the subroutine.
- If the subroutine is just a “processing” subroutine, have a single exit arrow.
- If the subroutine may exit along different paths, have multiple exit arrows.

Defining a Subroutine

- If you want to (or are asked to) design a TM subroutine, design it like a normal TM with one change: have special dashed states representing the exit of the subroutine.



Time-Out for Announcements!

Midterm Logistics

- Midterm is next **Thursday, November 13** from **7PM - 10PM**, location TBA.
 - Cumulative, focusing primarily on topics from PS4 – PS6.
 - Covers material up through and including the lecture on CFGs; TMs will not be tested.
- Need to take the exam at an alternate time? Email Maesen as soon as possible.
- Practice exam next Monday, November 10 from 7PM – 10PM in Annenberg Auditorium.

More Practice Questions

- Solutions to the first set of extra practice problems released.
- We have a new set of practice problems available on the course website.
 - Focus is on regular languages, nonregular languages, and CFGs.

Some Words of Encouragement

Your Questions!

“What should I do to prepare for a software engineer internship interview when there is only one week left?”

“At my last software internship, I was the only woman in my team of ~20 engineers. In your opinion, how do you think the tech industry would benefit from having a more representative population of women (or other minorities) in software engineering?”

“Since every person has a finite number of neurons, is it possible to create a DFA that models the human brain?”


“I like Korean food too. What's your favorite dish to eat? How about to make?”

Back to CS103!

Revisiting Fundamental Questions

What problems can we solve with a computer?

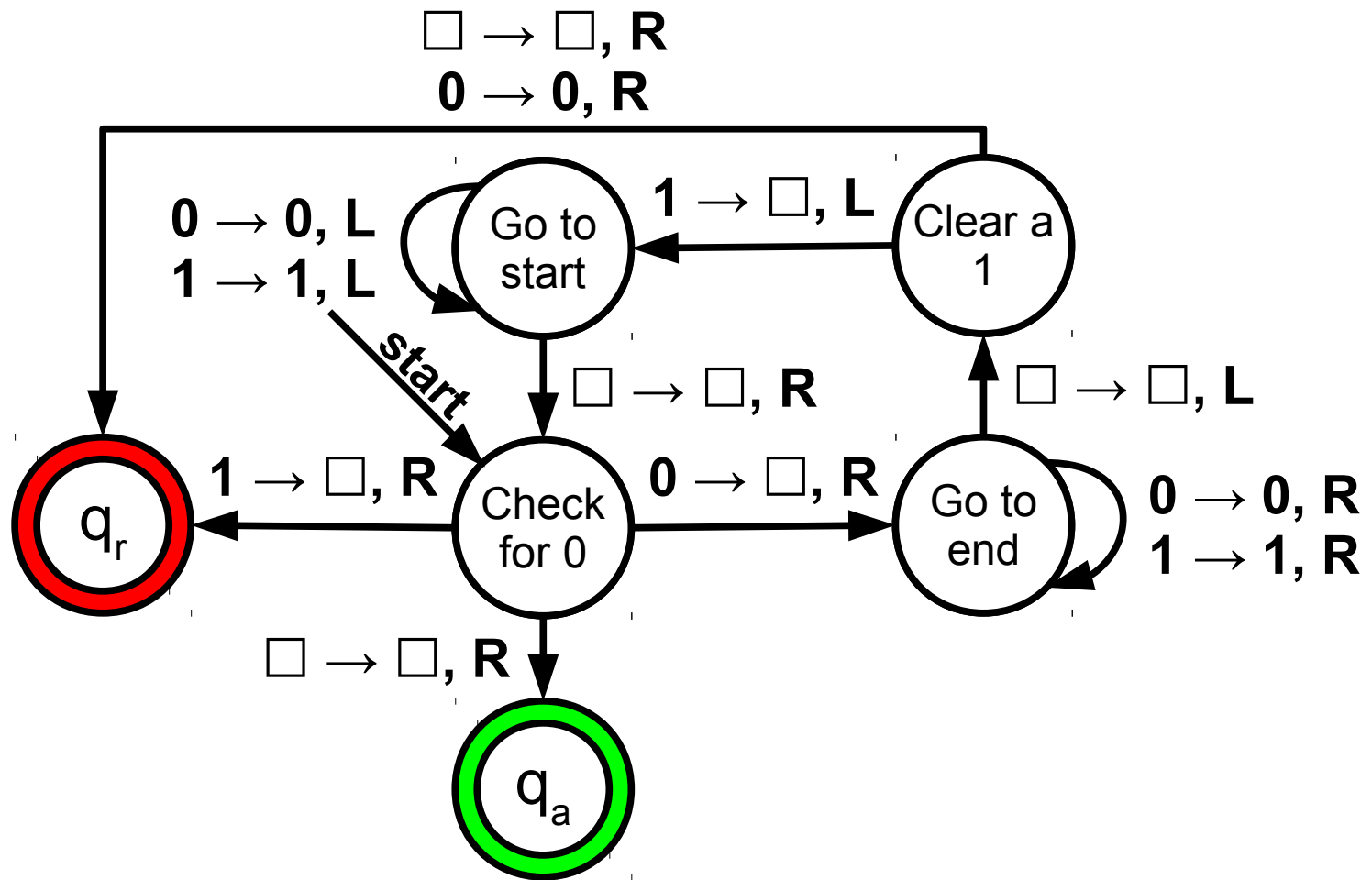
What kind of
computer?

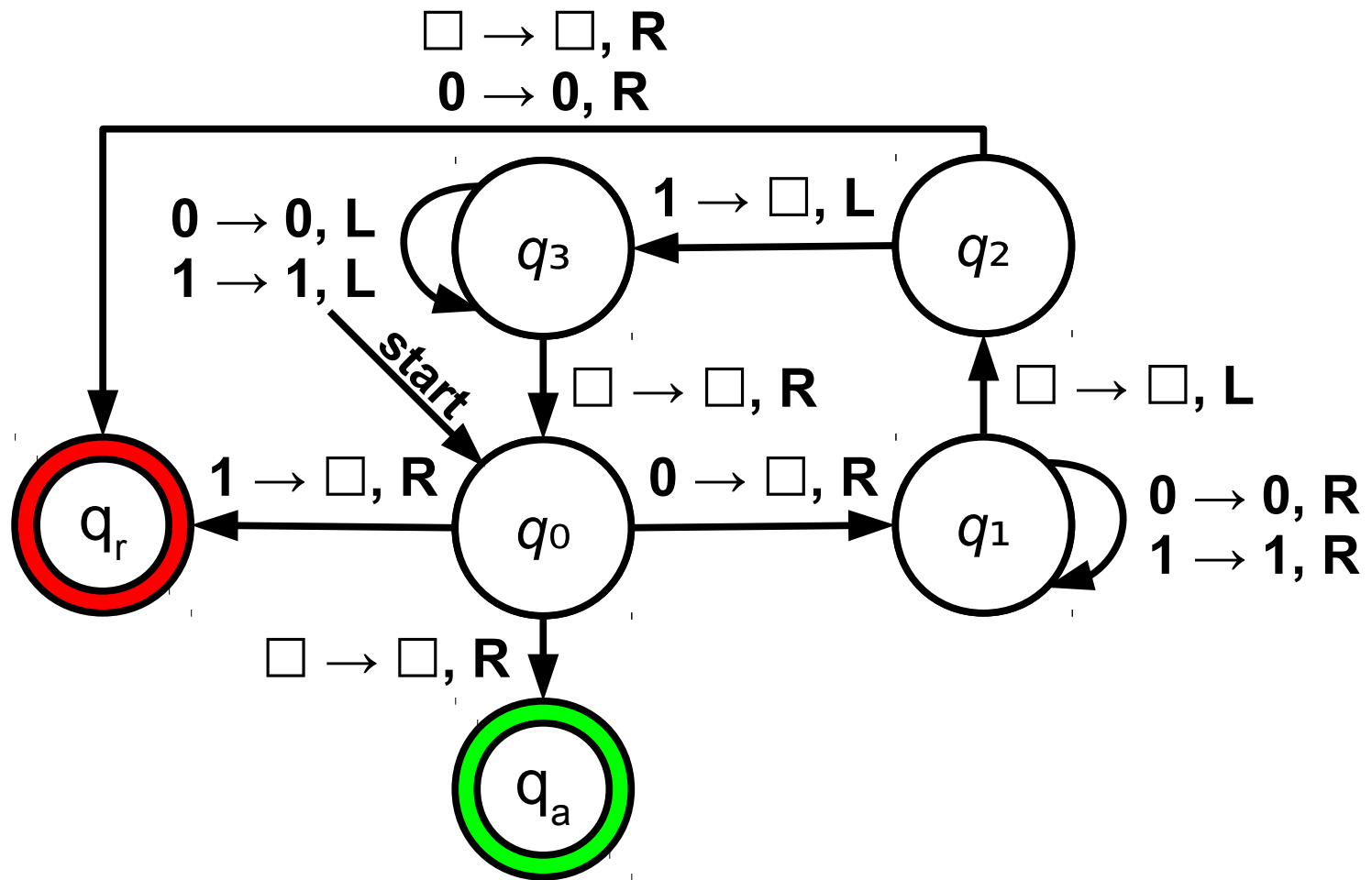


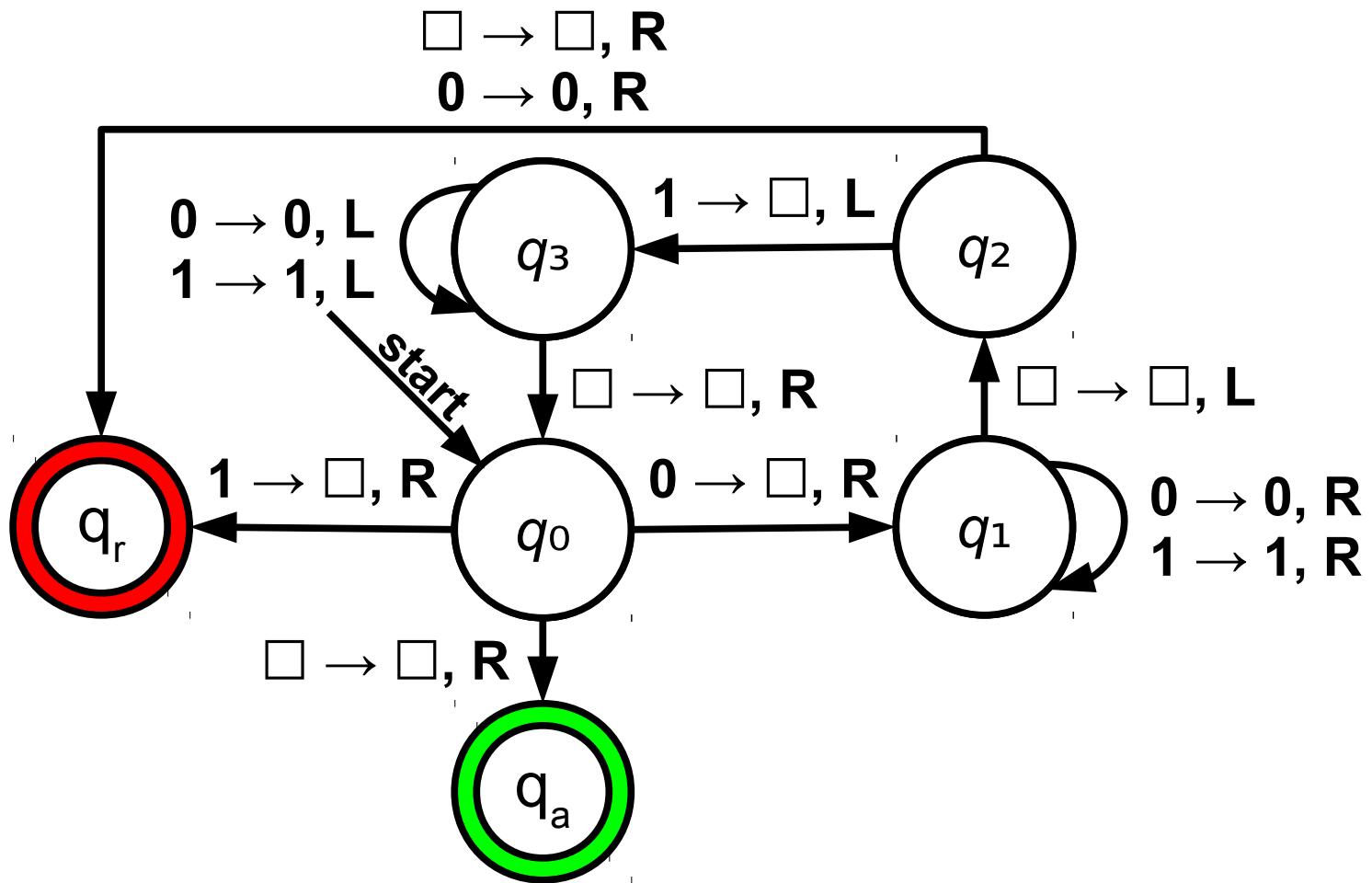
How Powerful are TMs?

- Regular languages, intuitively, are as powerful as computers with finite memory.
- TMs by themselves seem like they can do a fair number of tasks, but it's unclear specifically what they can do.
- Let's explore their expressive power.

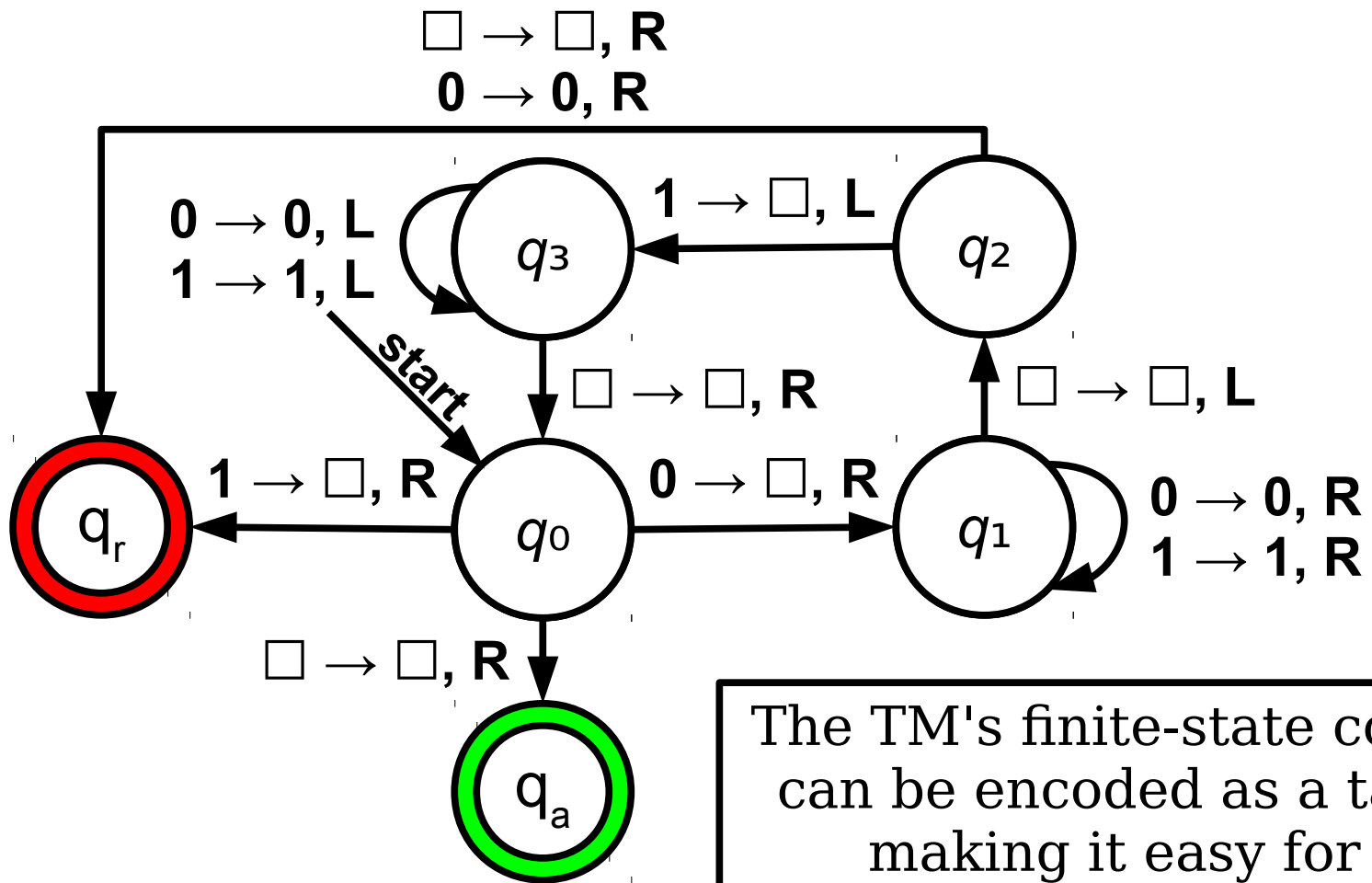
Claim 1: Computers with unbounded memory can simulate Turing machines.







		0		1		\square			
q_0	q_1	\square	R	q_r	\square	R	q_a	\square	R
q_1	q_1	0	R	q_1	1	R	q_2	\square	L
q_2	q_r	0	R	q_3	\square	L	q_r	\square	R
q_3	q_3	0	L	q_3	1	L	q_0	\square	R



The TM's finite-state control can be encoded as a table, making it easy for a computer to look up transitions information.

	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.)

Claim 2: Turing machines can simulate computers with unbounded memory.

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 on the tape).
 - perform elementary arithmetic (equality testing, multiplication, addition, division, etc.).
 - perform if/else tests (different transitions based on different cases).

What Else Can TMs Do?

- Maintain named variables.
 - Have a dedicated part of the tape where the variables are stored.
 - Each variable can be represented as a name (written one symbol at a time) followed by a value.
- Maintain arrays and linked structures.
 - Can create variables corresponding to different memory addresses.
 - Access to arrays or linked structures can be done by storing the names of the memory locations.

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 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 curious for more details, come talk to me after class.

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.*

This is not a mathematical fact – it's a
hypothesis about the nature of
computation.

Regular
Languages

CFLs

**Problems
Solvable by
*Any Feasible
Computing
Machine***

All Languages

**Regular
Languages**

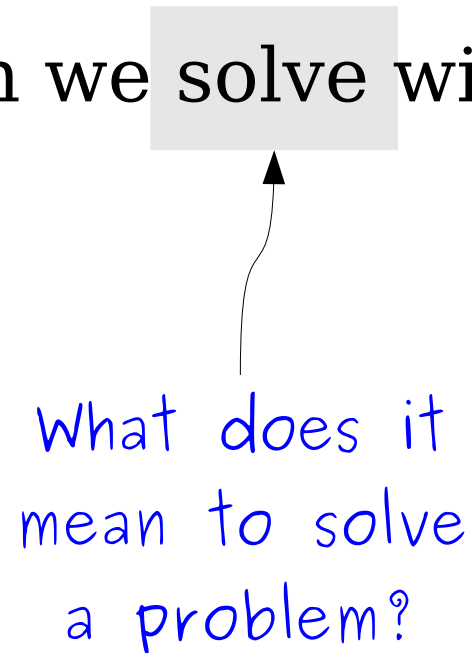
CFLs

**Languages
Recognized
by Turing
Machines**

All Languages

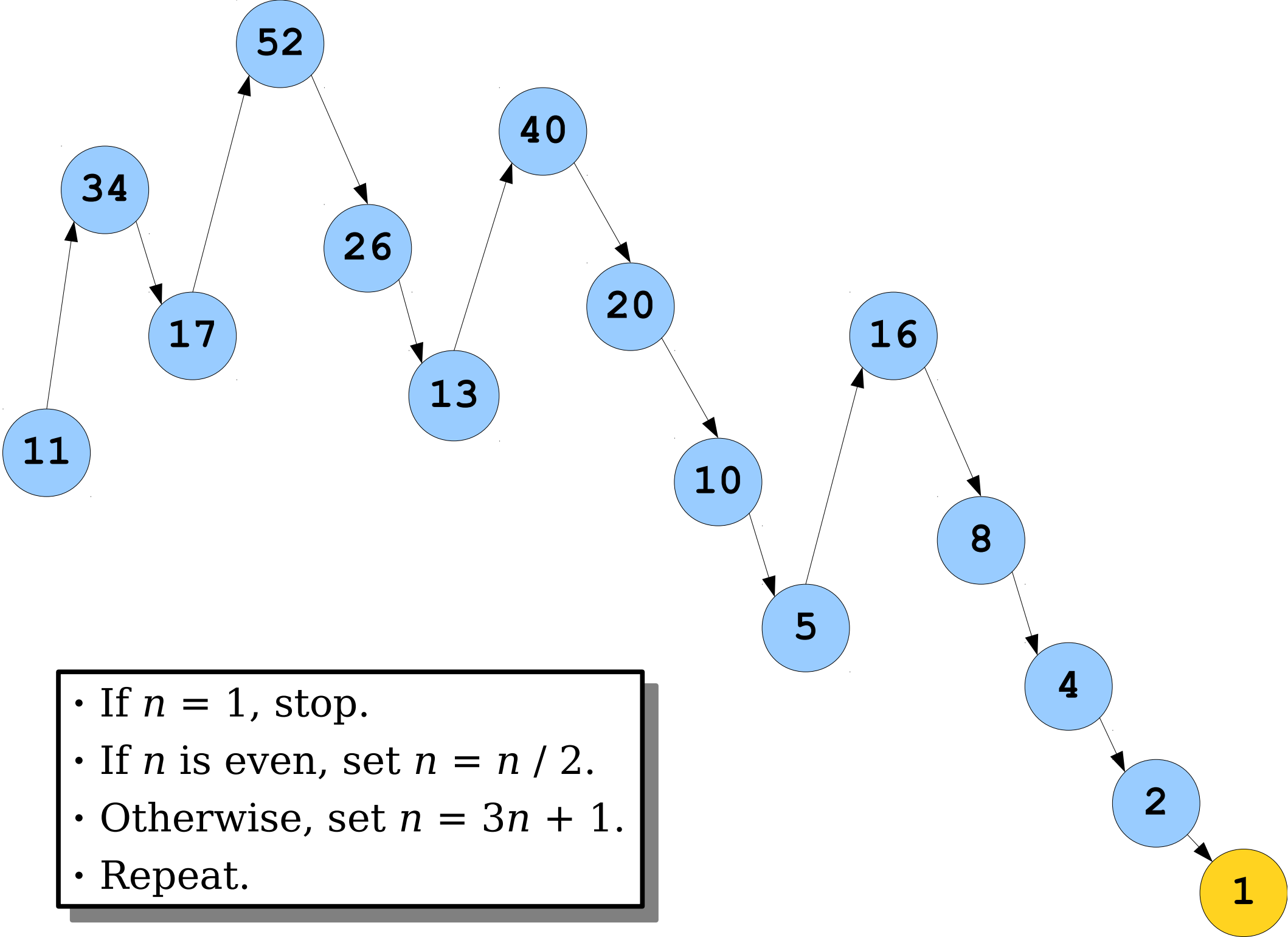
What problems can we solve with a computer?

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.

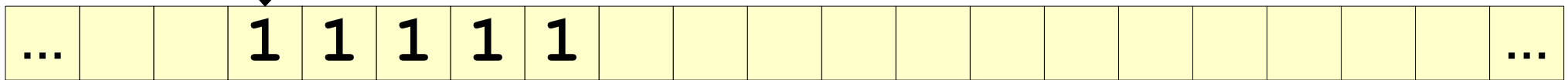
The Hailstone Sequence

- 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

- Intuitively, we can build a TM for the hailstone language as follows: the machine M does the following:
 - 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



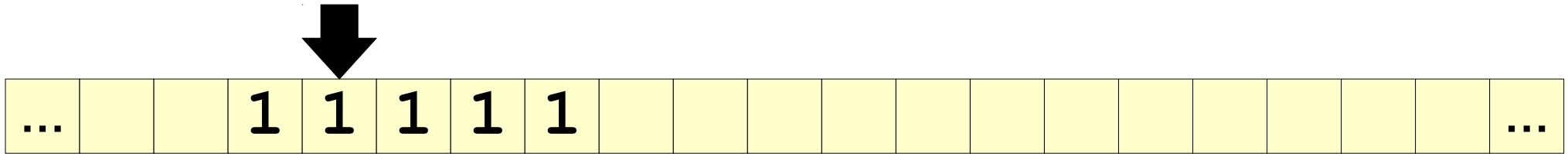
If the input is ε , reject.

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



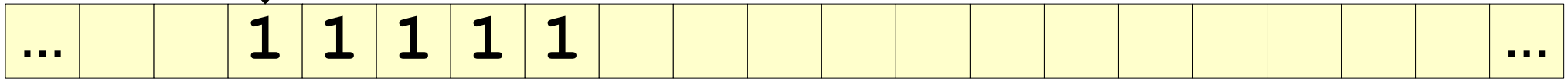
If the input is ε , reject.

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



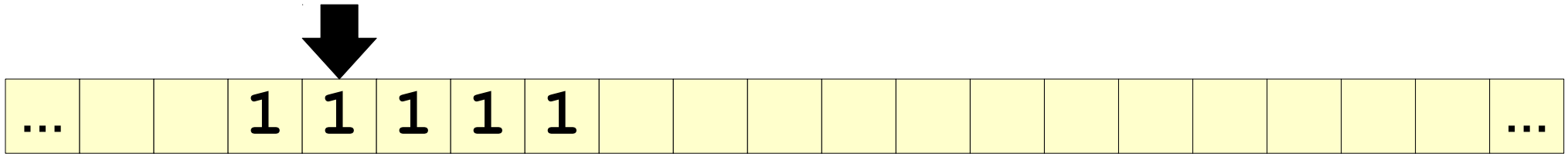
If the input is ε , reject.

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



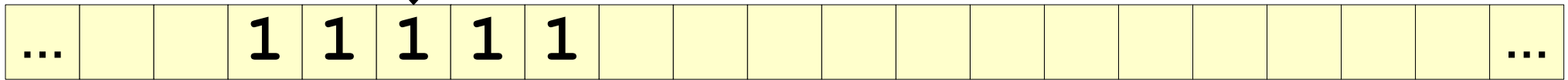
If the input is ε , reject.

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



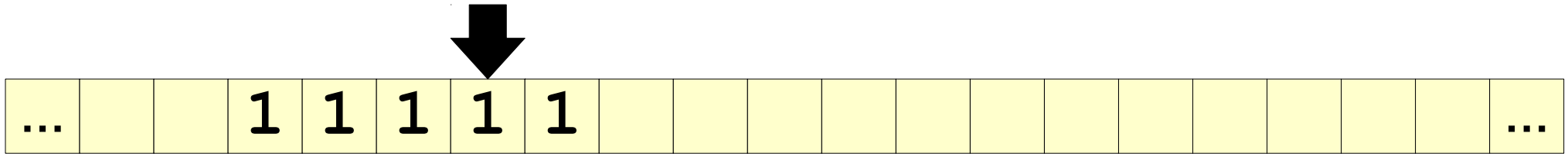
If the input is ε , reject.

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



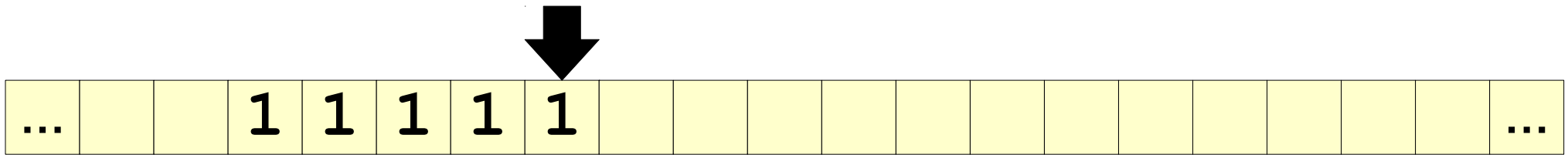
If the input is ε , reject.

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



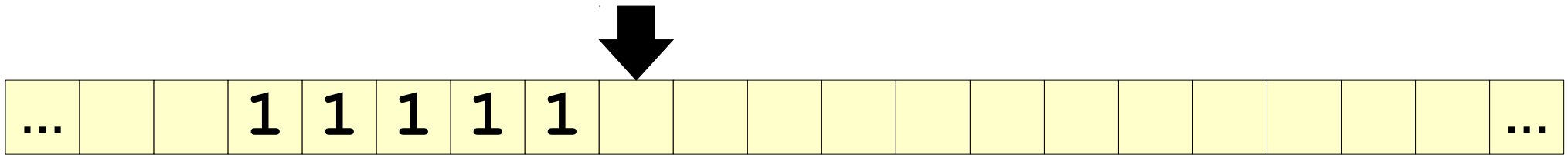
If the input is ε , reject.

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



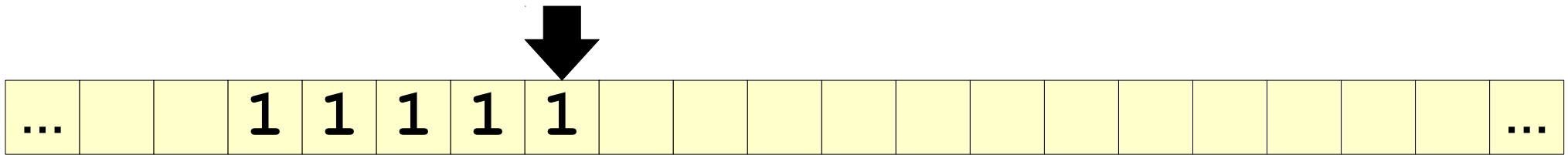
If the input is ε , reject.

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



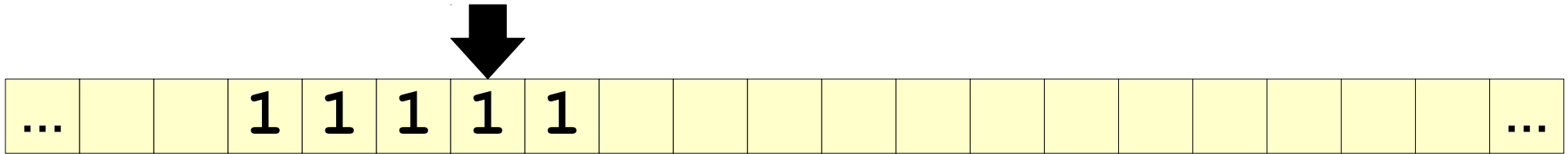
If the input is ε , reject.

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.

The Hailstone Turing Machine



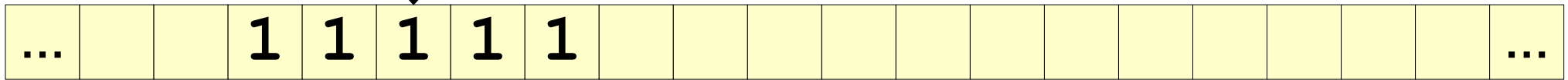
If the input is ε , reject.

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



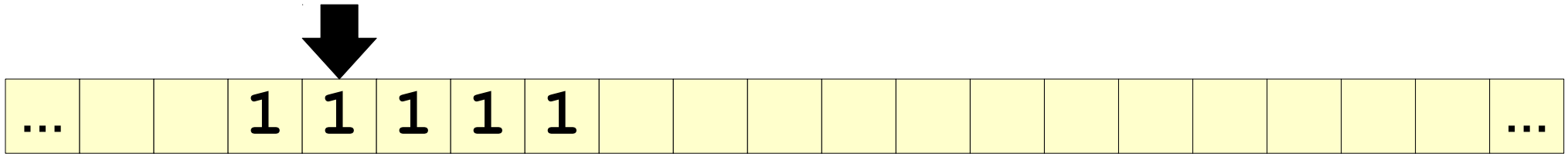
If the input is ε , reject.

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



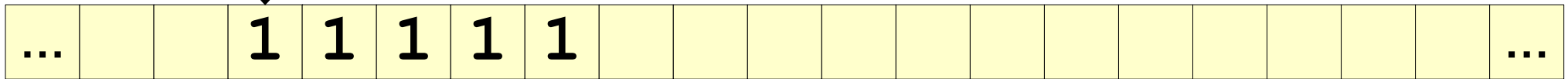
If the input is ε , reject.

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



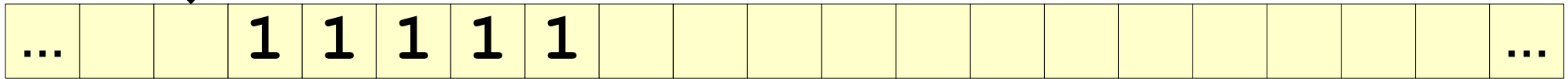
If the input is ε , reject.

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



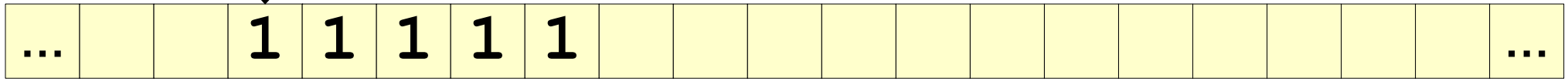
If the input is ε , reject.

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



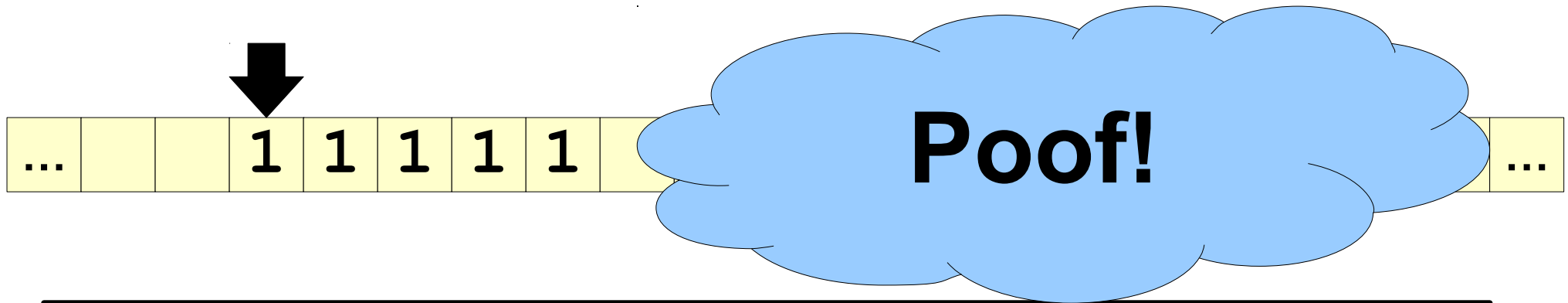
If the input is ε , reject.

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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 of the string.
- If the input has odd length of the string and append a **1**.

Accept.

Interesting problem:
Build a TM that, starting with n **1**s on its tape, ends with $3n + 1$ **1**s on its tape.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

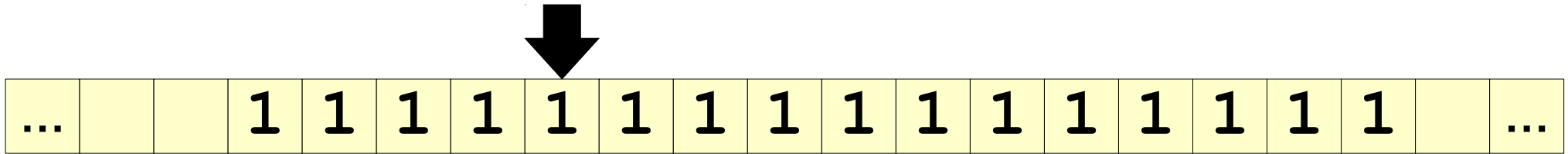
If the input is ε , reject.

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



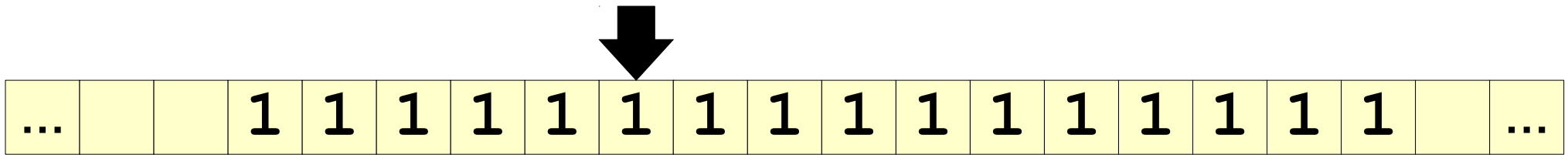
If the input is ε , reject.

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

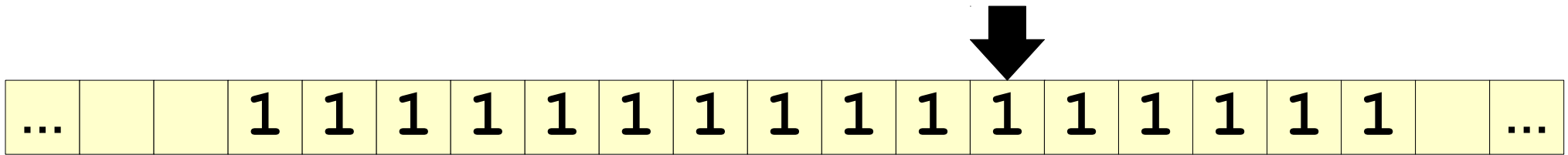
If the input is ε , reject.

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



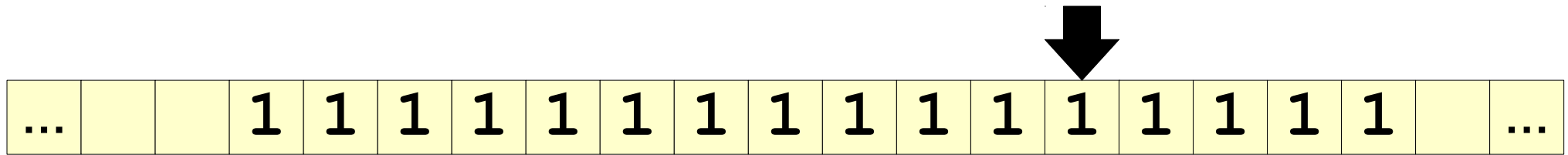
If the input is ε , reject.

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



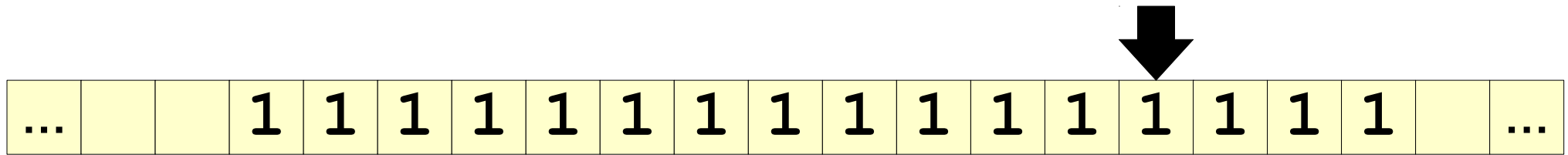
If the input is ε , reject.

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



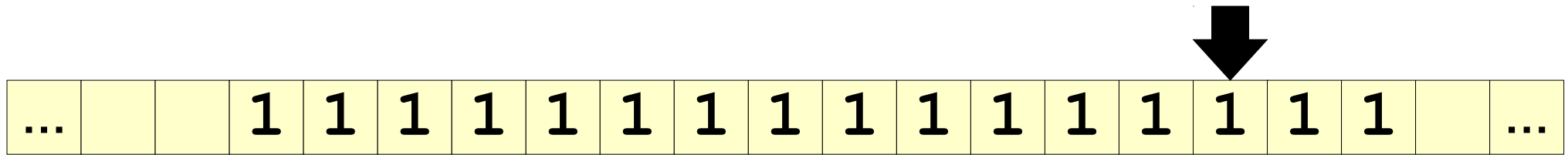
If the input is ε , reject.

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



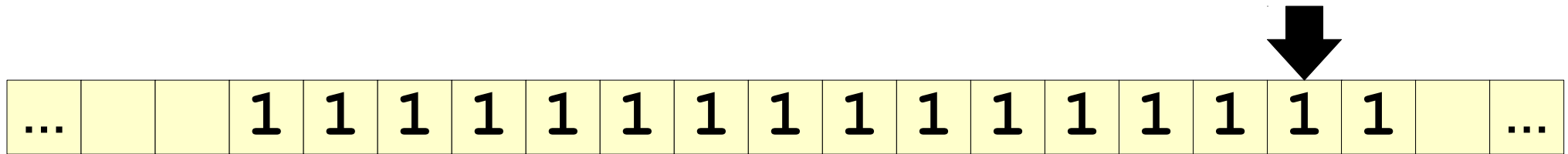
If the input is ε , reject.

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



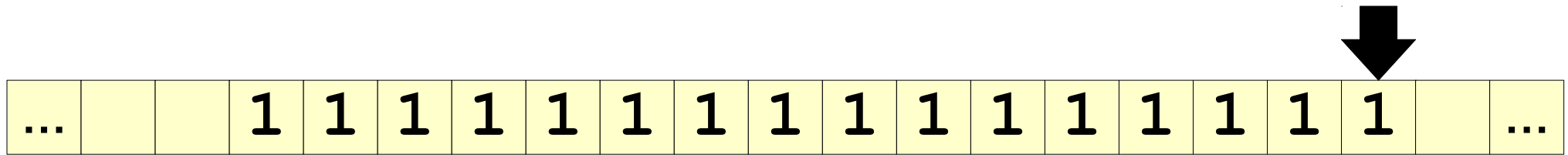
If the input is ε , reject.

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



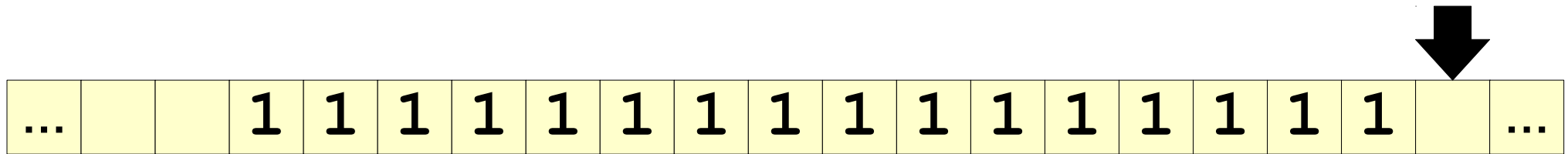
If the input is ε , reject.

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



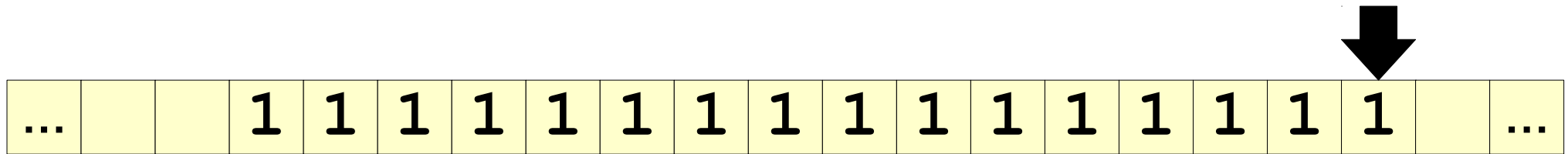
If the input is ε , reject.

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



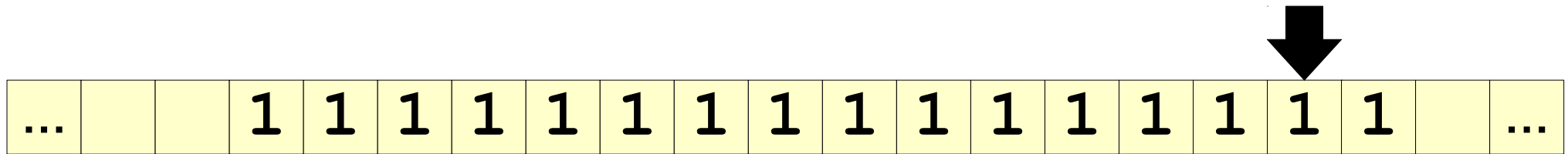
If the input is ε , reject.

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



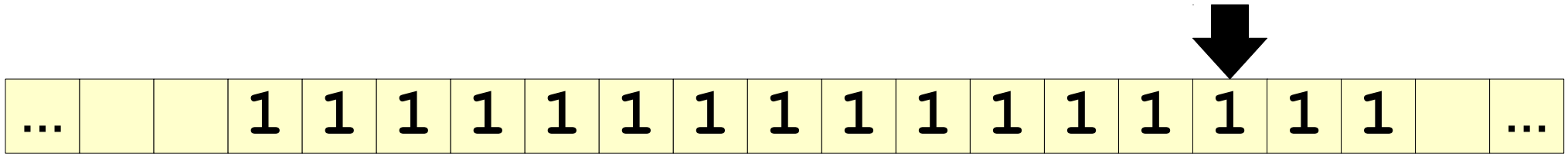
If the input is ε , reject.

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.

The Hailstone Turing Machine



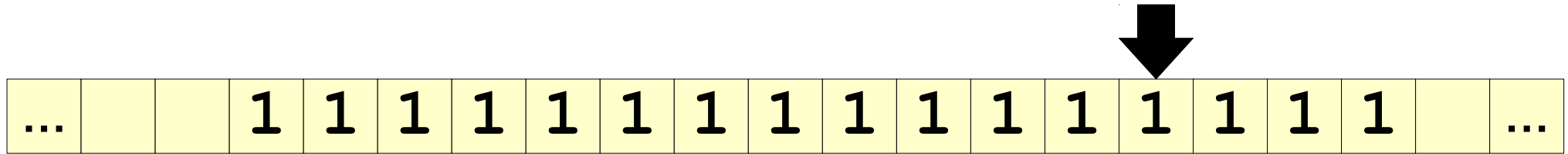
If the input is ε , reject.

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



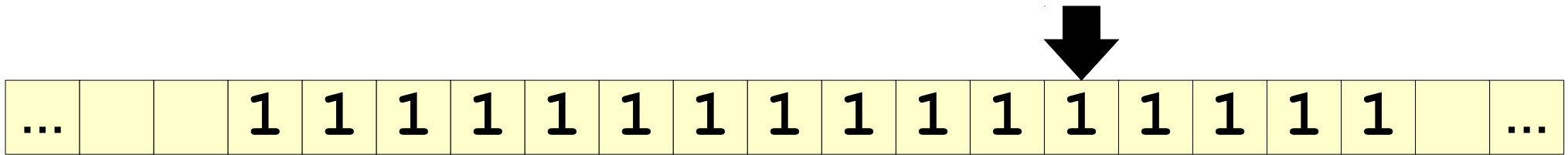
If the input is ε , reject.

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



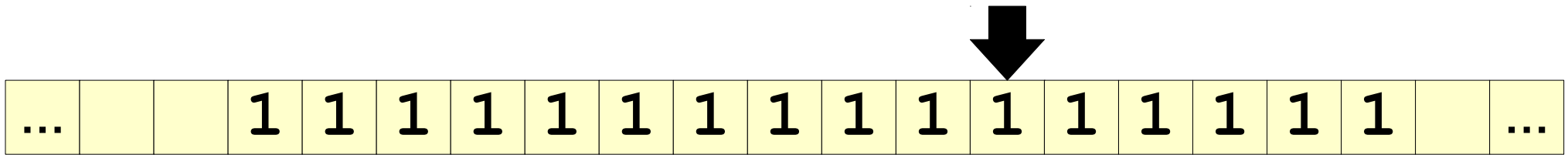
If the input is ε , reject.

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

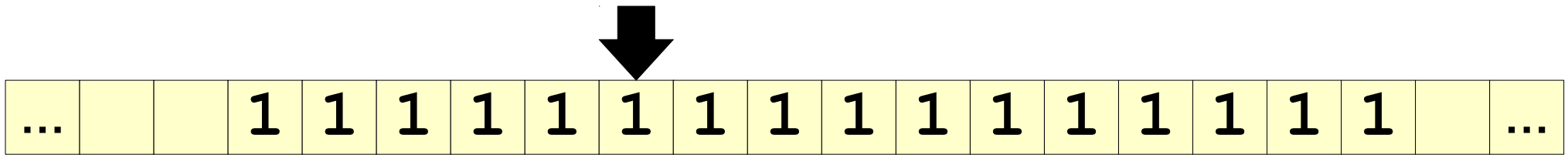
If the input is ε , reject.

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



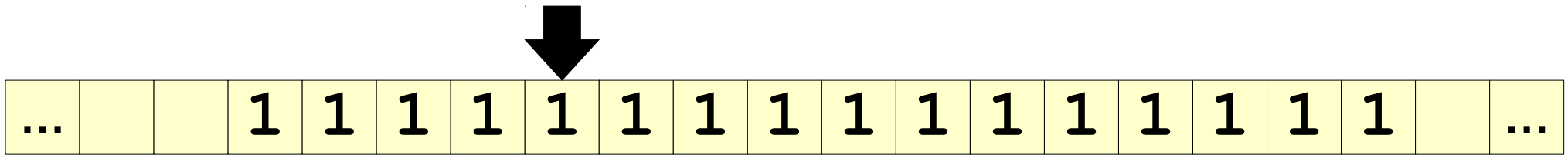
If the input is ε , reject.

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

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



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...
-----	--	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

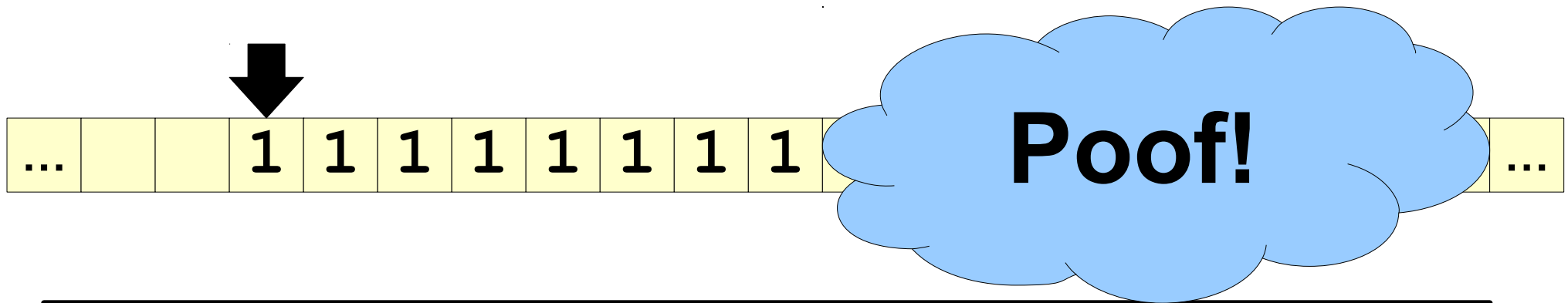
If the input is ε , reject.

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



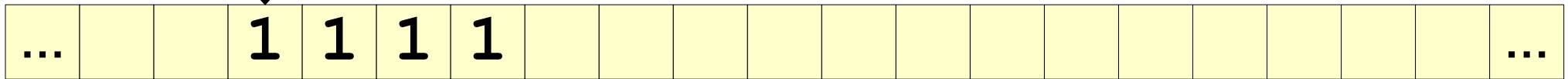
If the input is ε , reject.

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



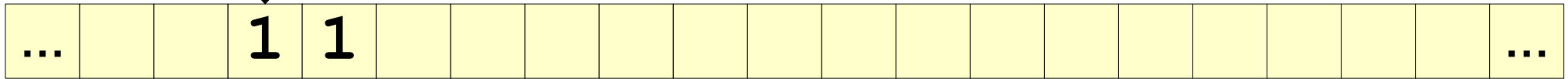
If the input is ε , reject.

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



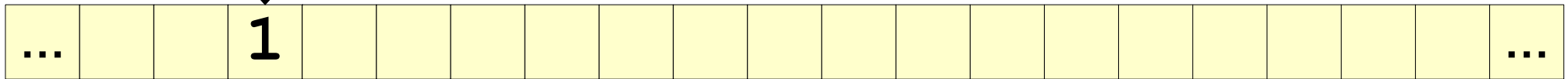
If the input is ε , reject.

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 always accept?

The Collatz Conjecture

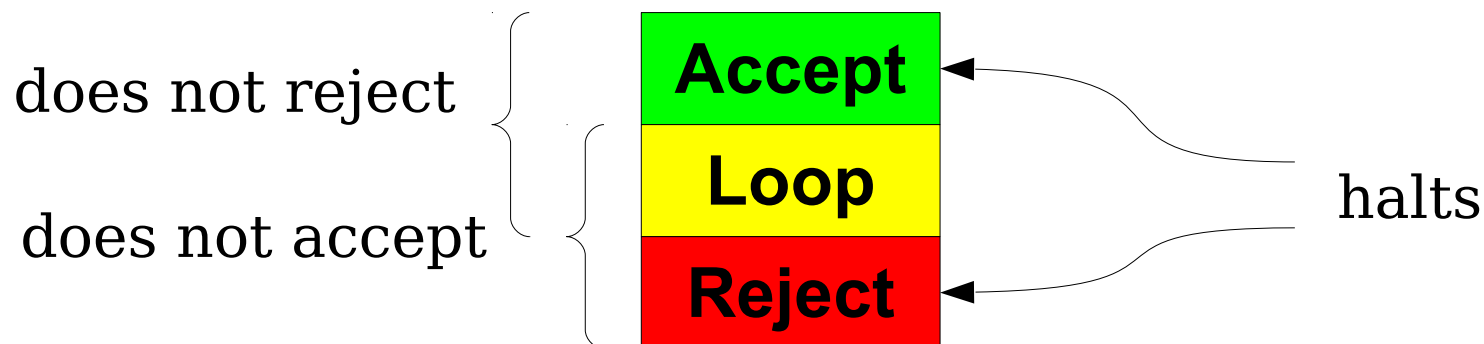
- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, ***no one knows whether the TM described in the previous slides will always stop running!***
- The conjecture (unproven claim) that this always terminates is called the ***Collatz Conjecture***.

An Important Observation

- Unlike the other automata we've seen so far, Turing machines choose for themselves whether to accept or reject.
- It is therefore 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?

Some Important Terminology

- Let M be a Turing machine.
- M **accepts** a string w if it enters the accept state when run on w .
- M **rejects** a string w if it enters the reject state when run on w .
- M **loops infinitely** (or just **loops**) on a string w if when run on w it enters neither the accept or reject state.
- M **does not accept w** if it either rejects w or loops infinitely on w .
- M **does not reject 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:

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

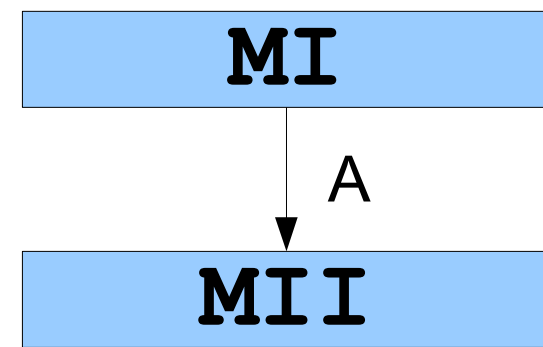
- For any $w \in \mathcal{L}(M)$, M accepts w .
- For any $w \notin \mathcal{L}(M)$, M does not accept w .
 - It might loop forever, or it might explicitly reject.
- A language is called **recognizable** if it is the language of some TM.
- Notation: the class **RE** is the set of all recognizable languages.

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

Worklist TMs

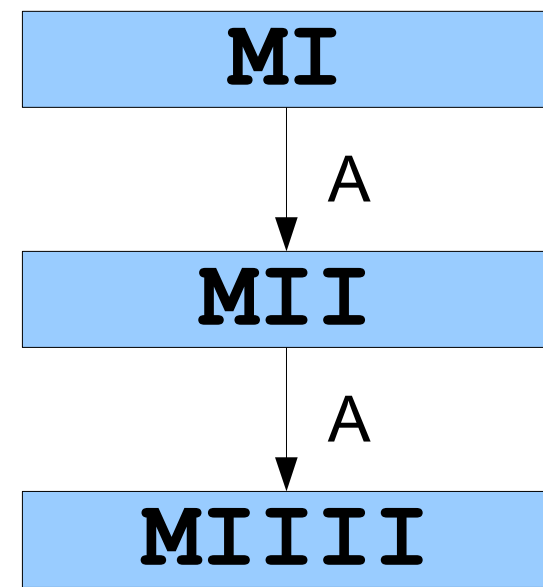
- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.

- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.

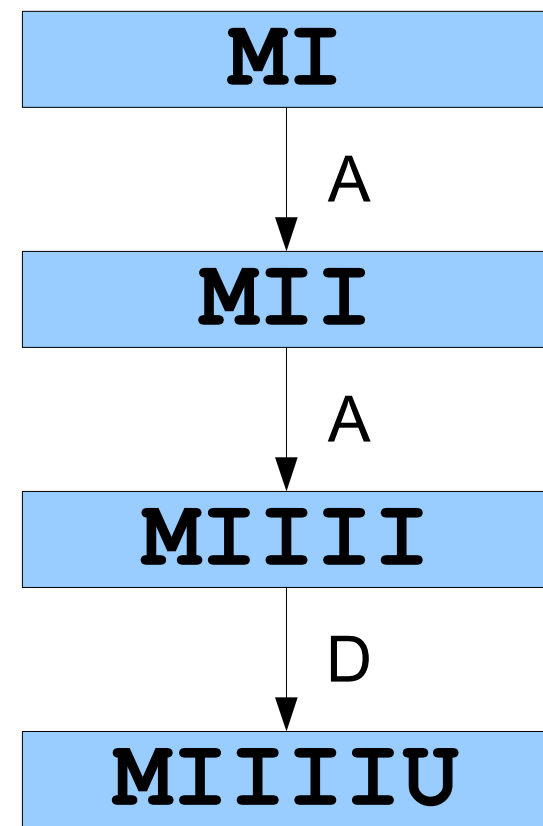


- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.

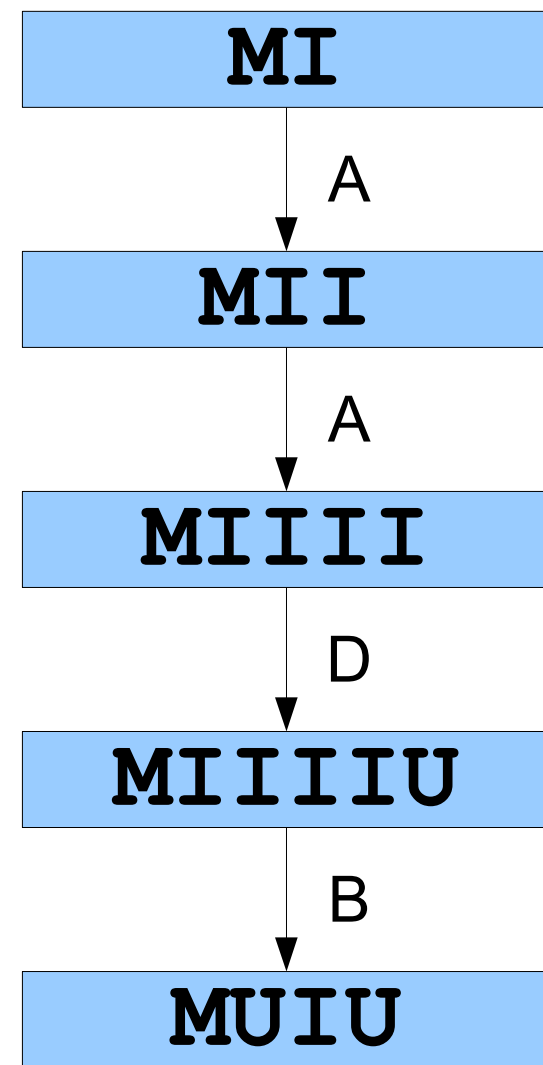
- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.



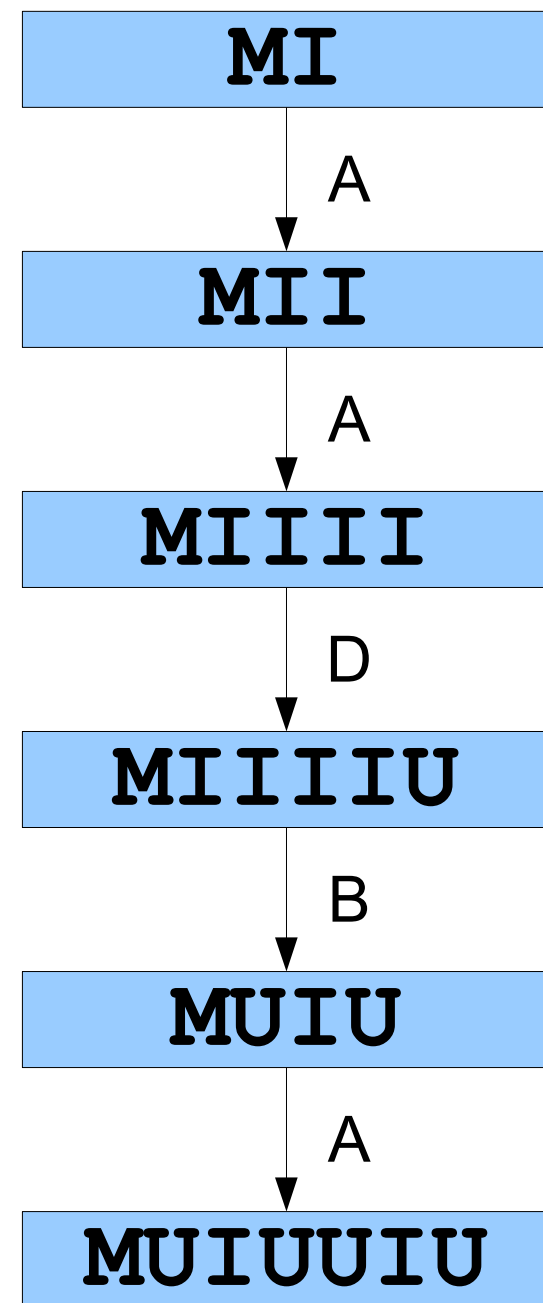
- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.



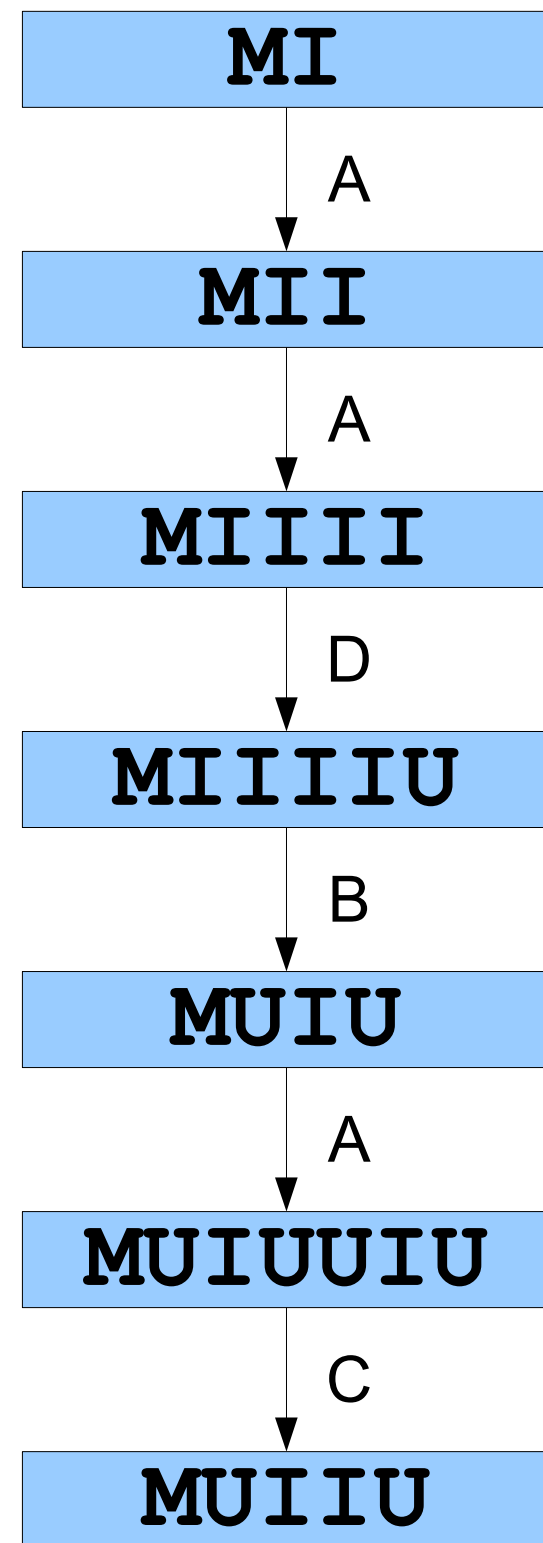
- A) Double the contents of the string after **M**.
- B) **Replace III with U.**
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.



- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) Remove **UU**
- D) Append **U** if the string ends in **I**.



- A) Double the contents of the string after **M**.
- B) Replace **III** with **U**.
- C) **Remove UU**
- D) Append **U** if the string ends in **I**.



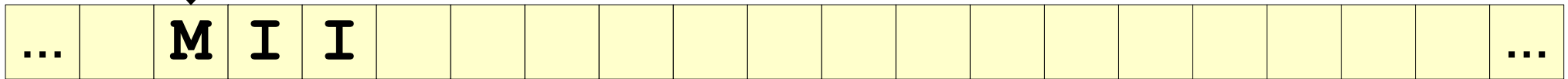
A Recognizable Language

- Let $\Sigma = \{ \mathbf{M}, \mathbf{I}, \mathbf{U} \}$ and consider the language $L = \{ w \in \Sigma^* \mid \text{Using the four provided rules, it is possible to convert } w \text{ into } \mathbf{MU} \}$
- Some strings are in this language (for example, $\mathbf{MU} \in L$, $\mathbf{MIII} \in L$, and $\mathbf{MUUU} \in L$).
- Some strings are not in this language (for example, $\mathbf{I} \notin L$, $\mathbf{MI} \notin L$, and $\mathbf{MIIU} \notin L$).
- Could we build a Turing machine for L ?

TM Design Trick: Worklists

- It is possible to design TMs that search over an infinite space using a worklist.
- Conceptually, the TM
 - Finds all possible options one step away from the original input,
 - Appends each of them to the end of the worklist,
 - Clears the current option, then
 - Grabs the next element from the worklist to process.
- ***This Turing machine is not guaranteed to halt.***

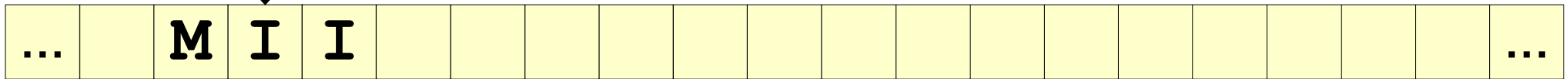
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

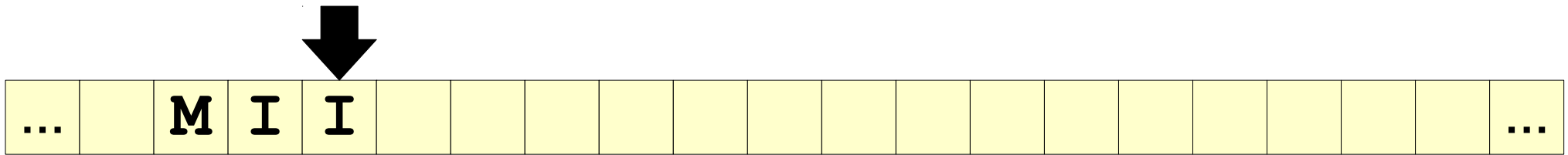
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

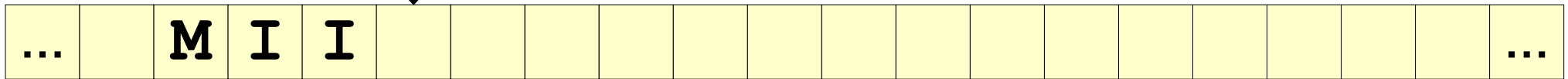
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

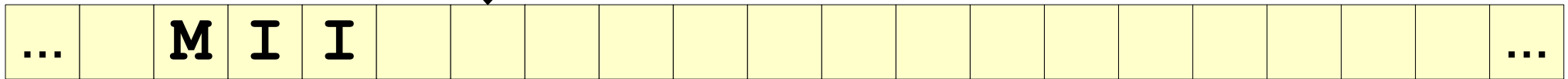
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

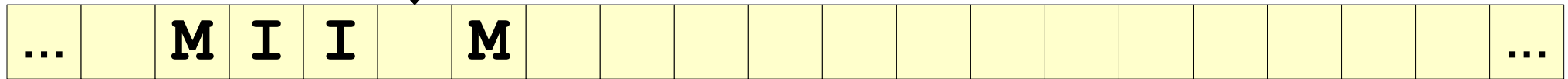
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

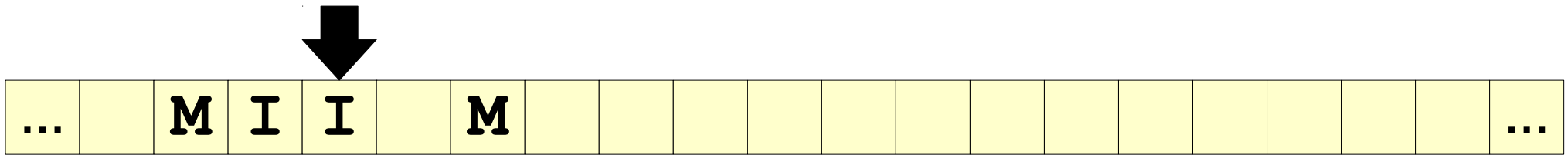
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

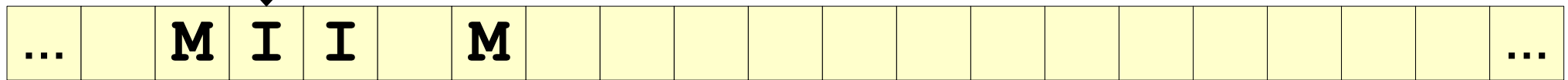
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

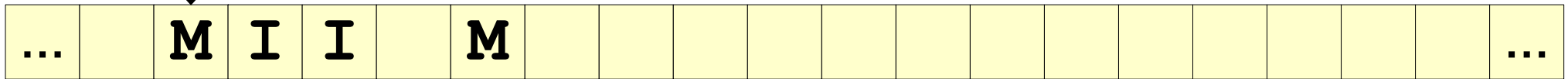
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

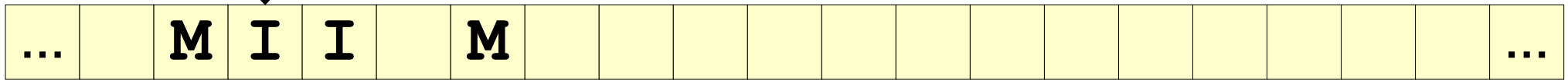
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

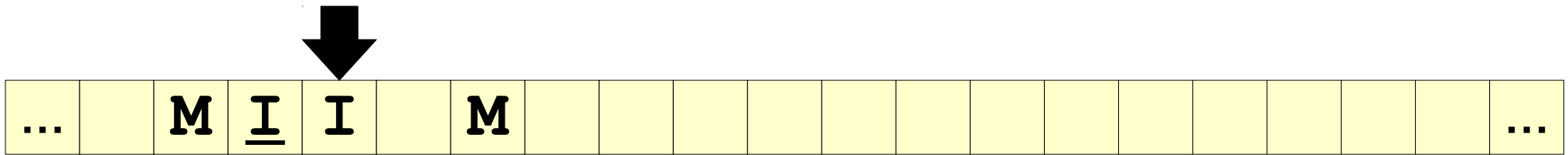
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

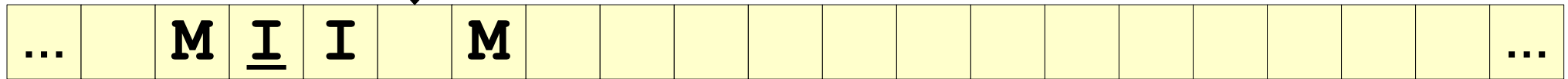
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

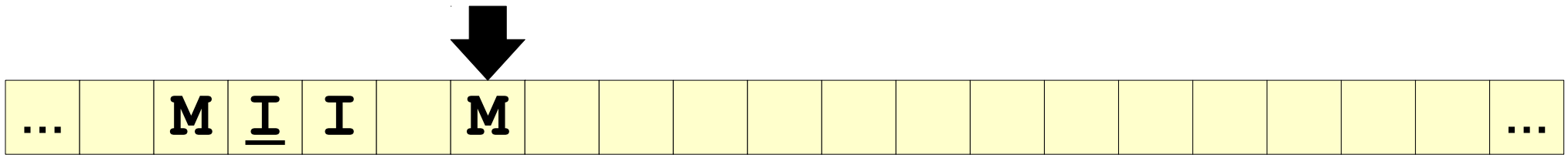
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

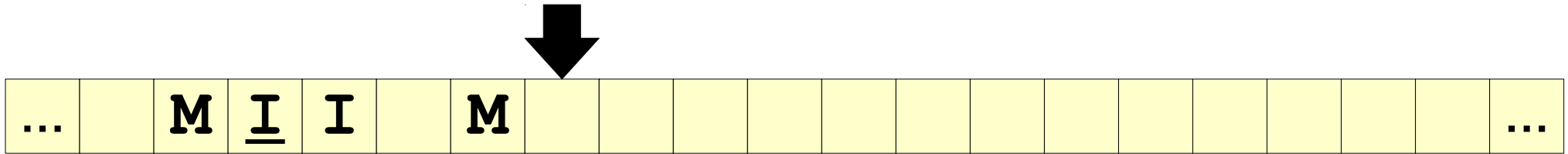
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

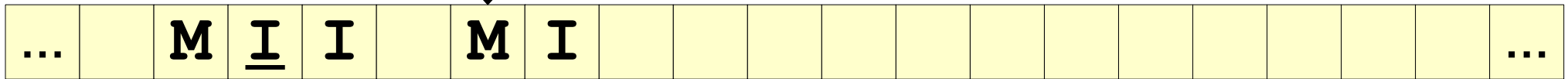
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

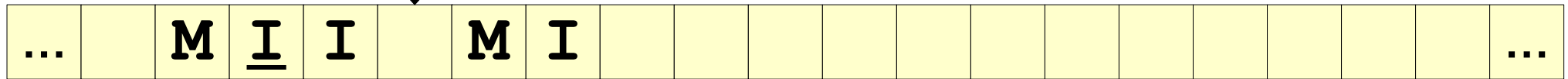
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

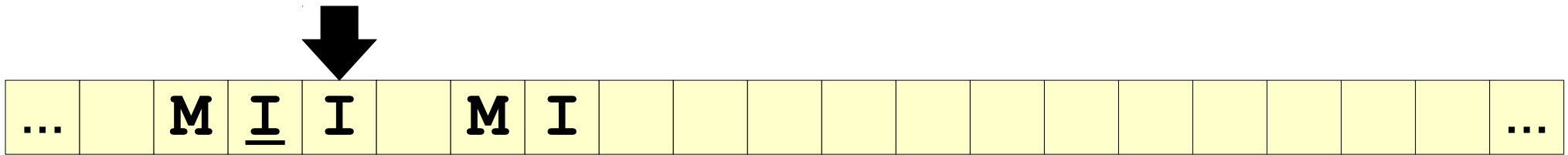
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

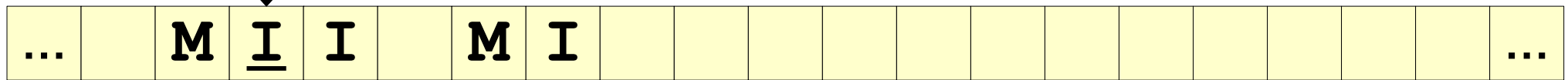
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

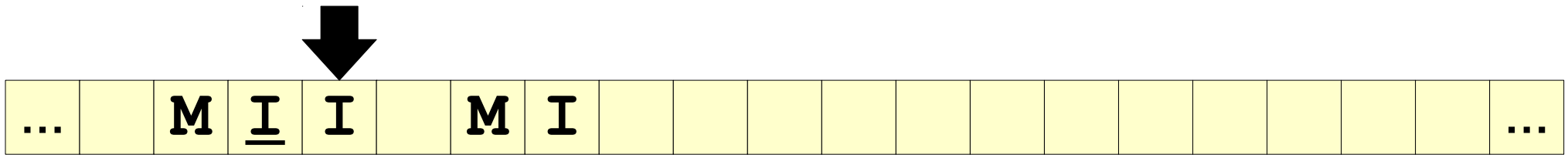
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

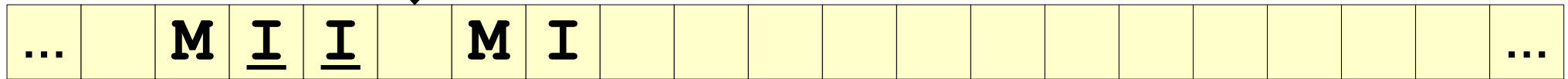
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

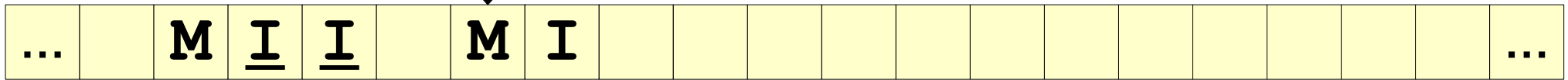
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

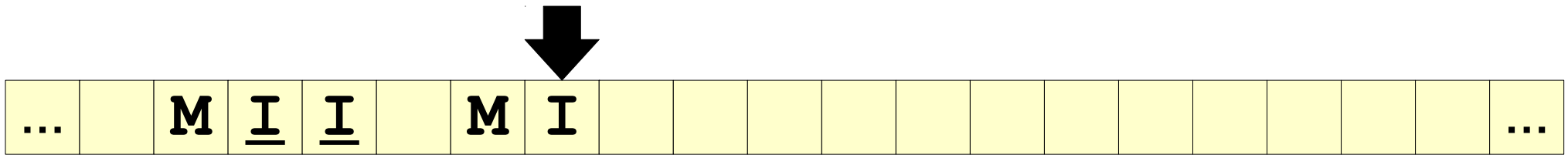
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

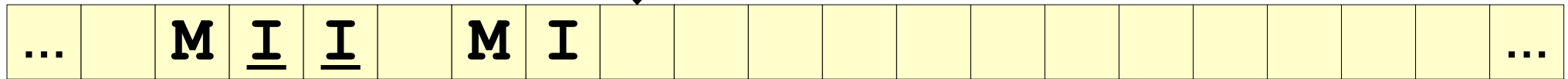
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

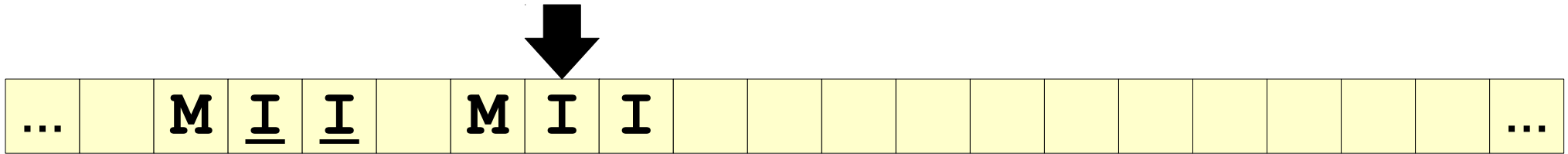
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

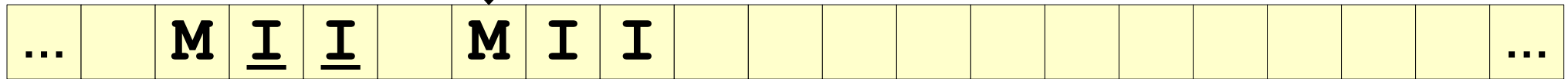
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

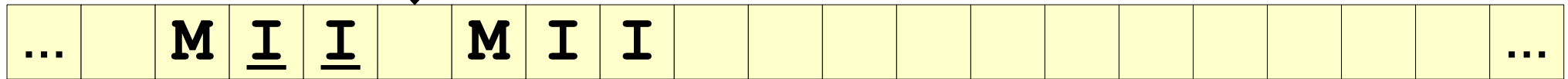
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

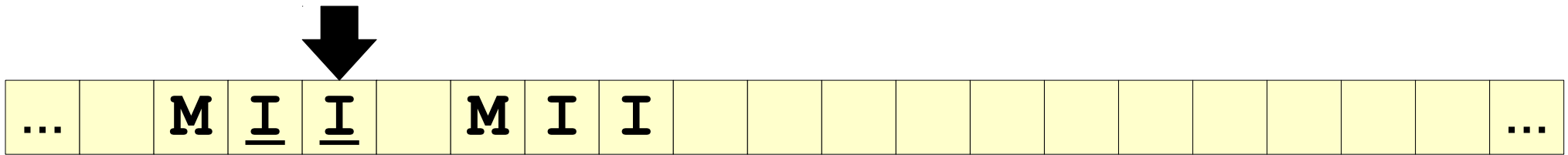
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

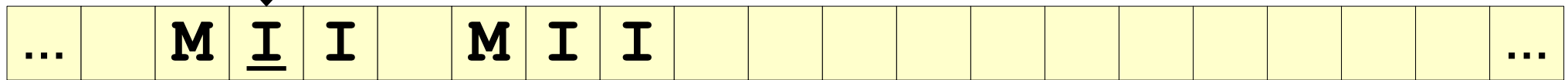
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

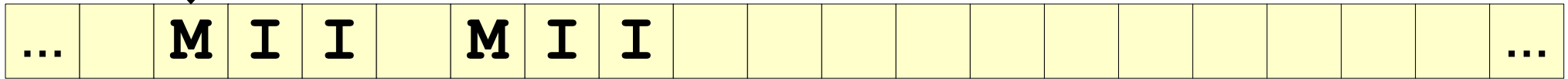
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

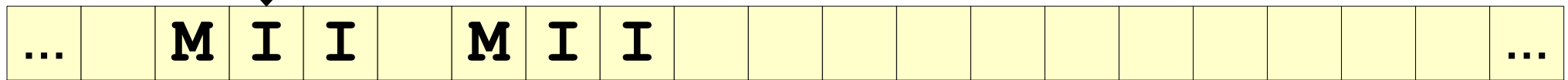
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

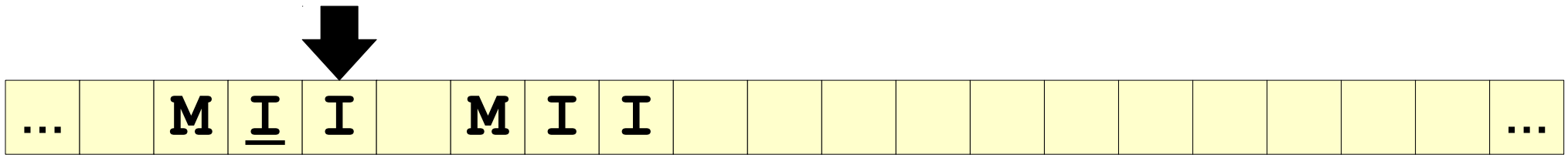
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

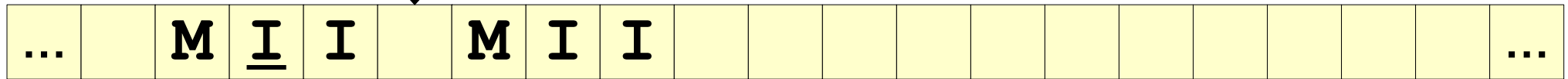
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

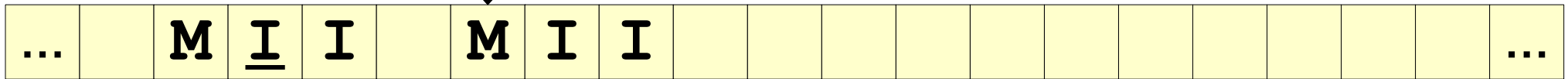
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

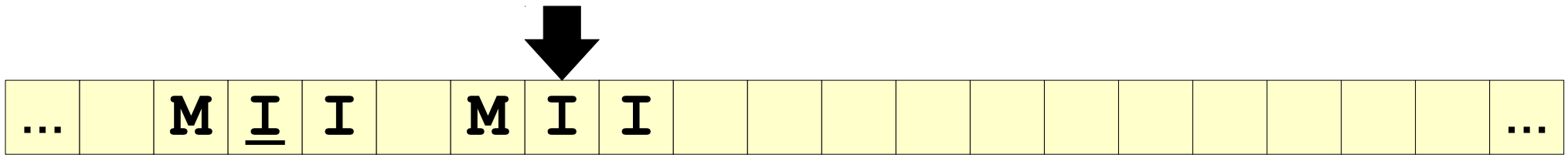
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

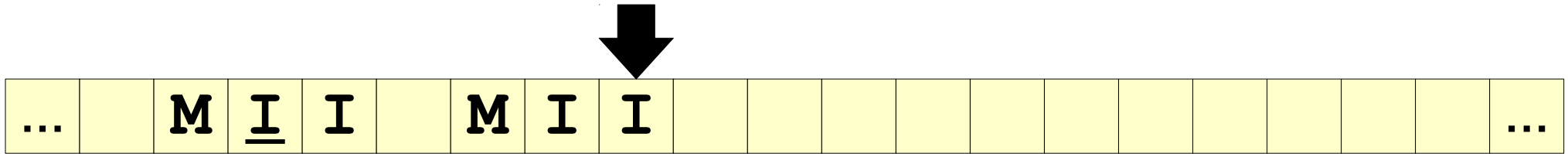
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

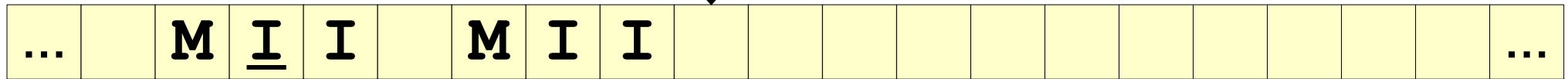
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

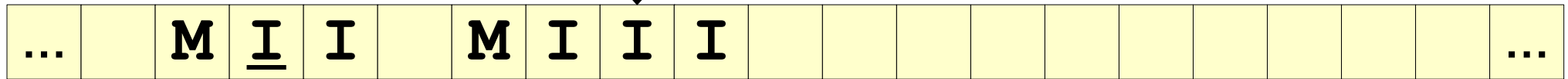
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

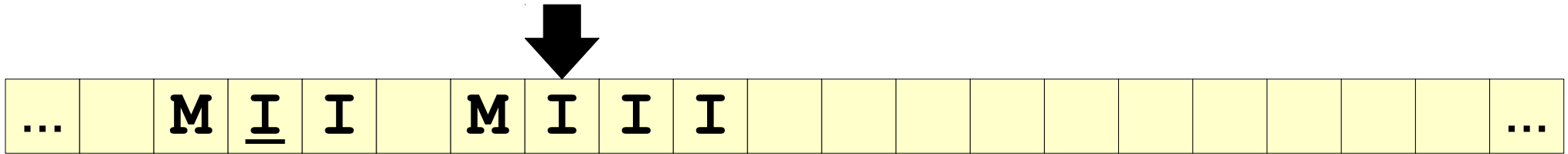
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

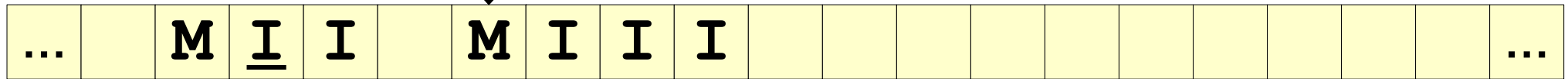
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

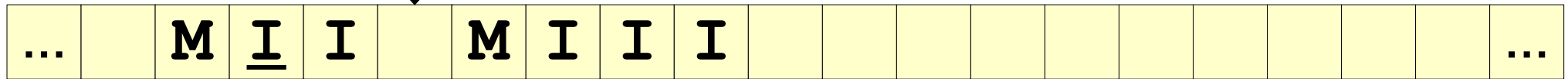
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

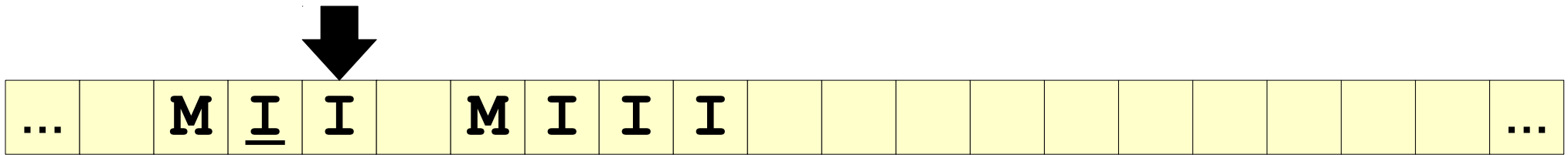
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

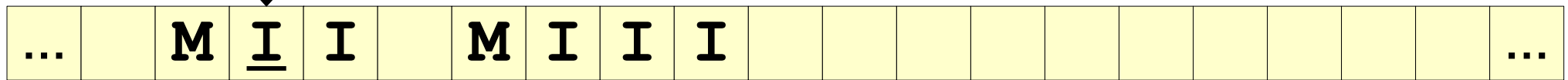
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

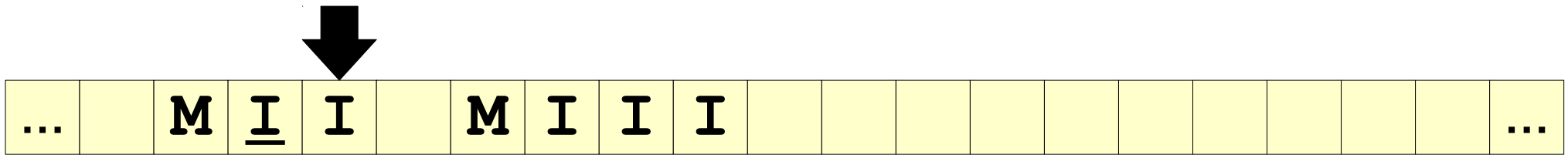
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

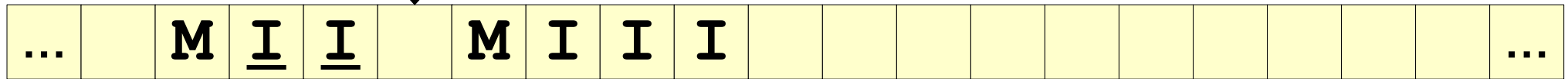
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

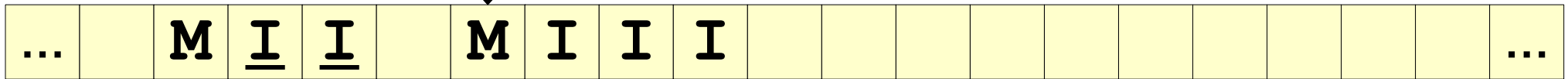
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

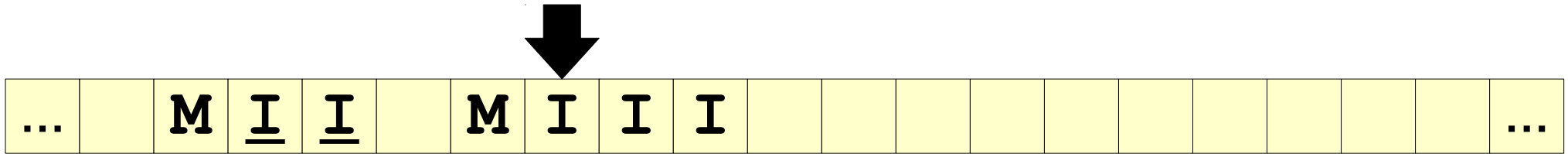
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

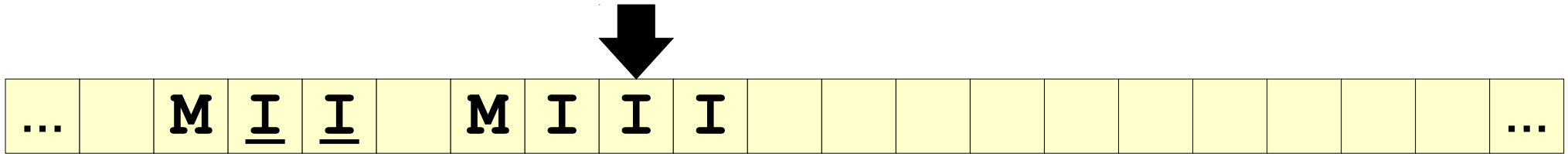
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

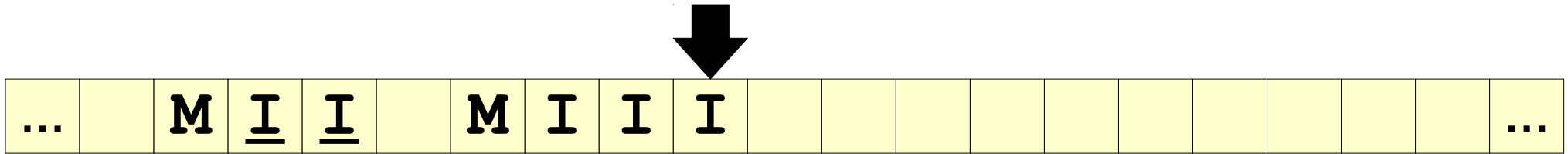
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

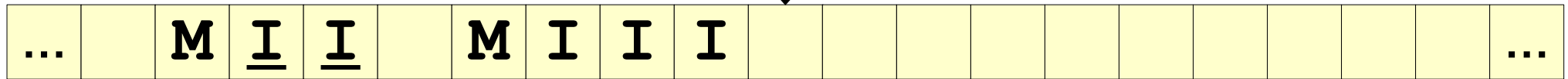
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

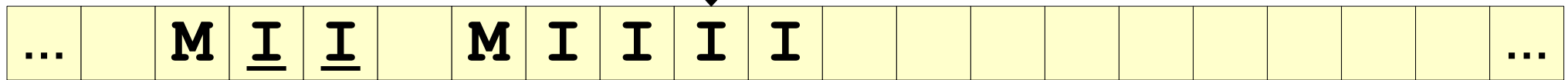
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

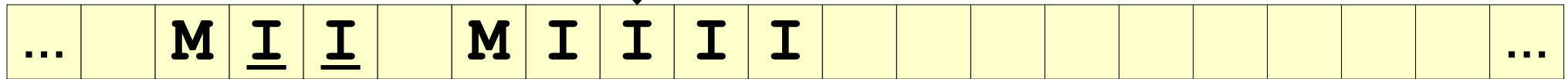
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

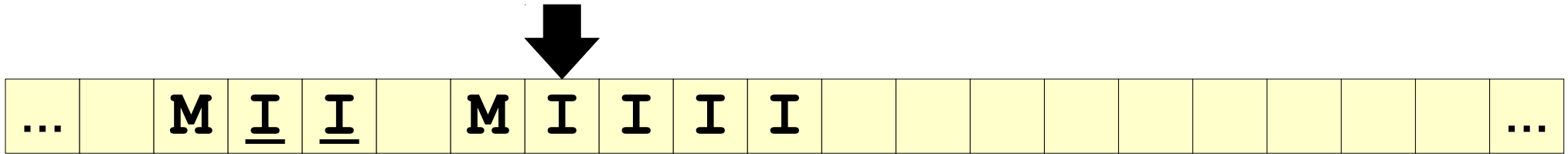
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

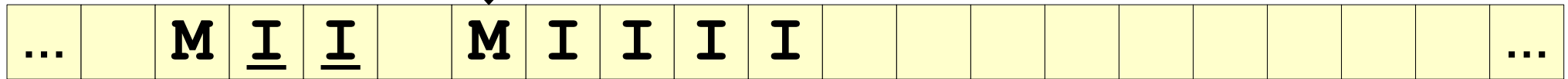
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

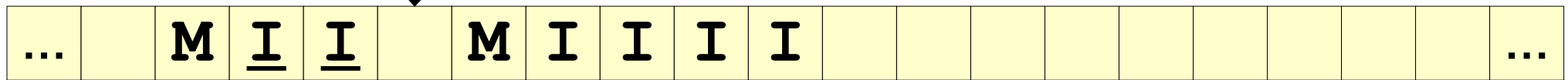
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

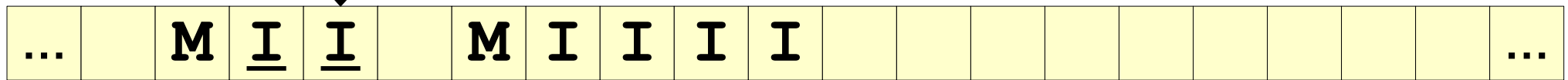
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

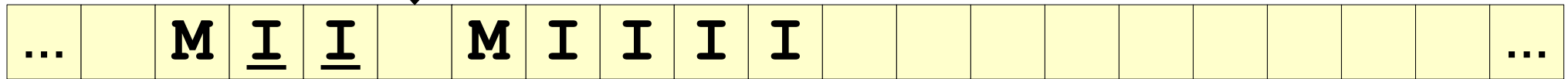
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

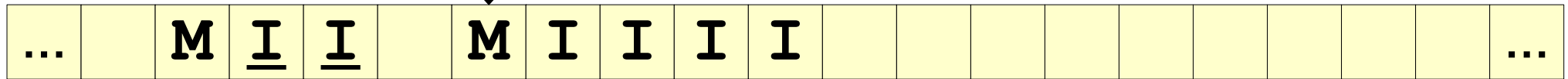
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

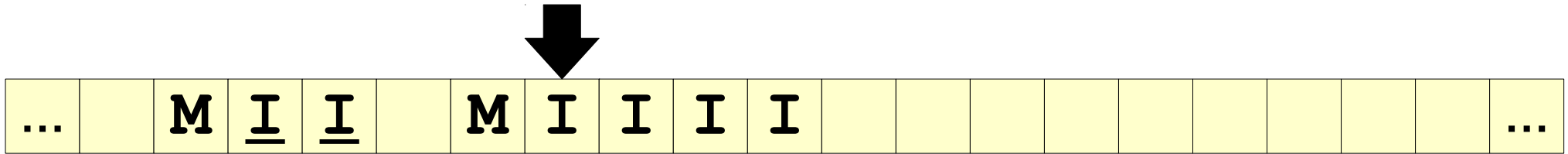
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

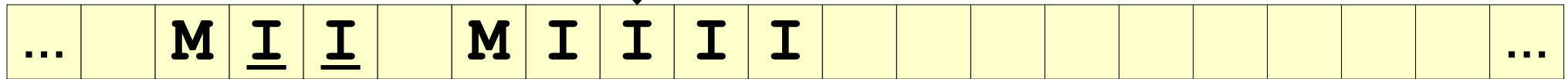
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

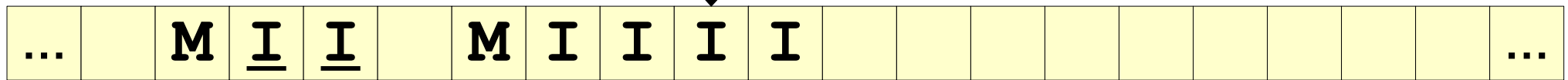
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

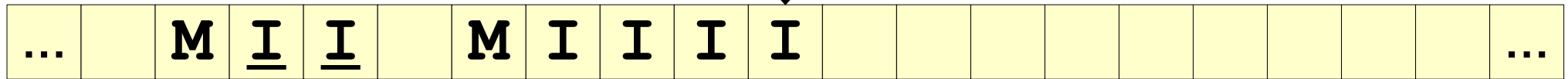
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

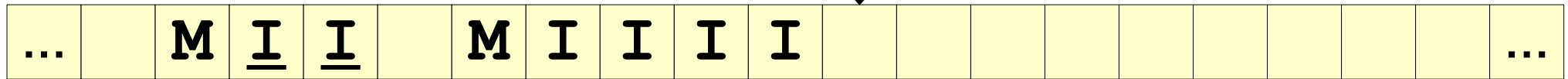
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

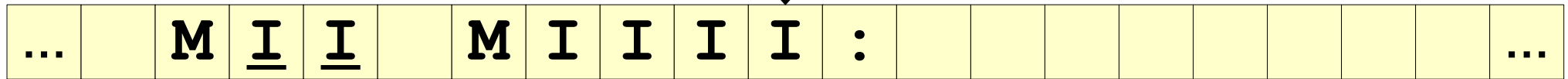
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

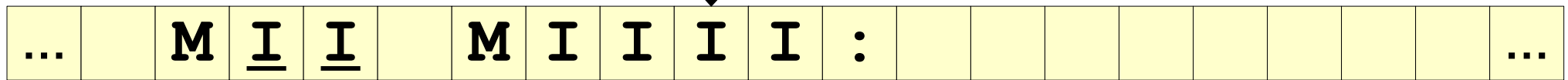
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

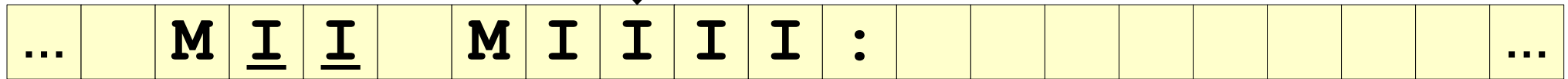
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

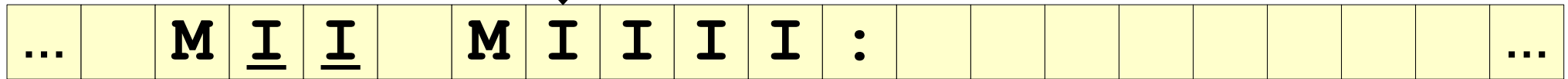
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

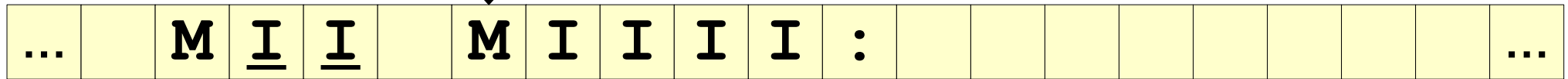
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

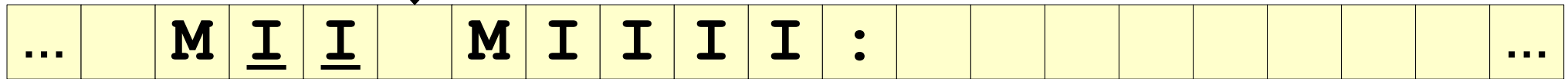
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

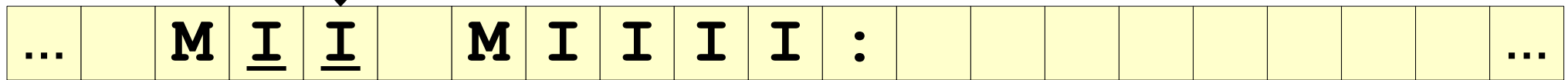
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

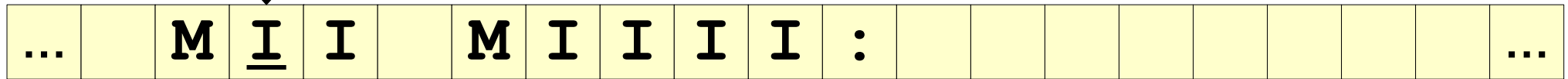
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

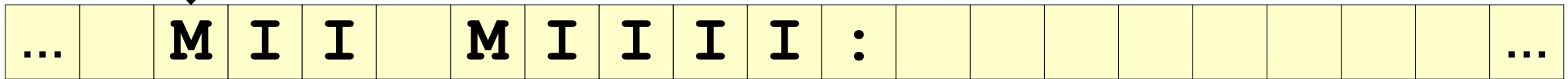
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

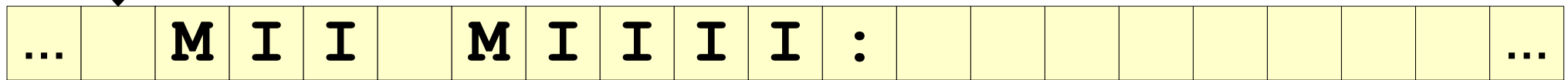
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

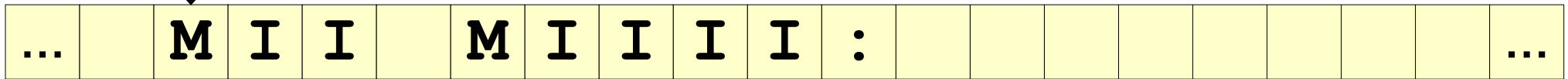
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

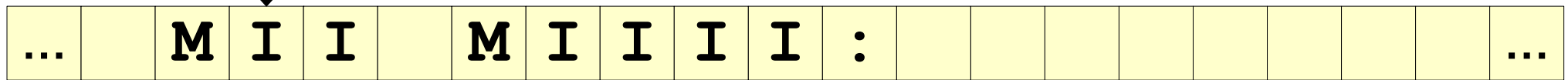
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

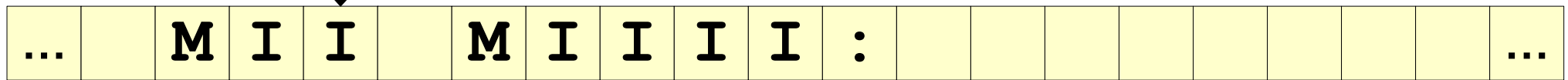
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

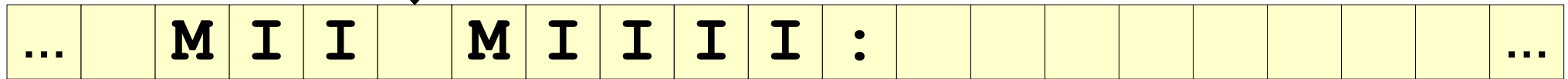
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

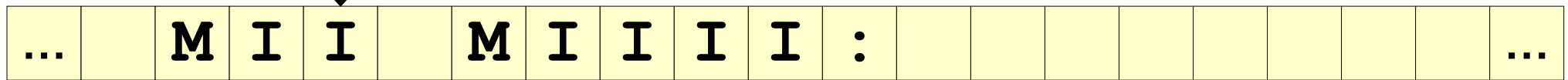
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

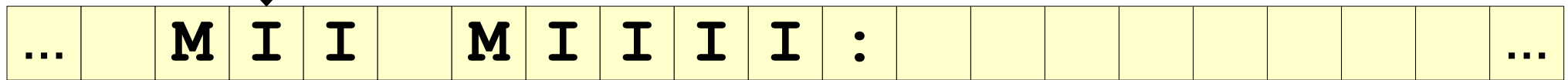
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

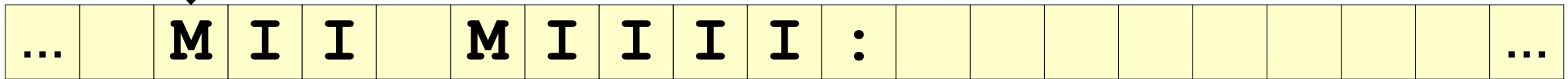
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

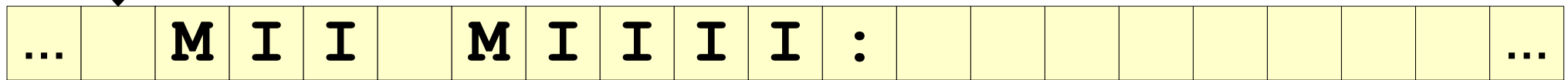
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

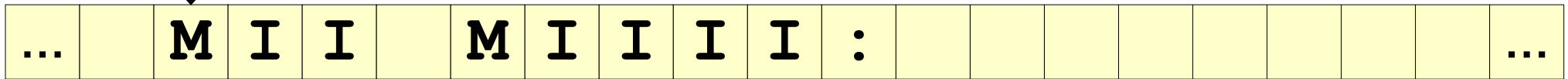
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

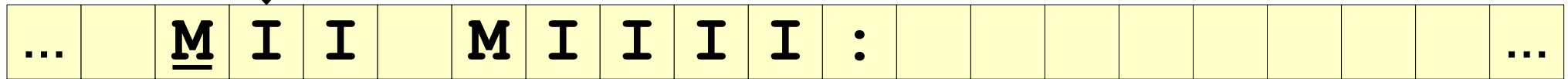
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

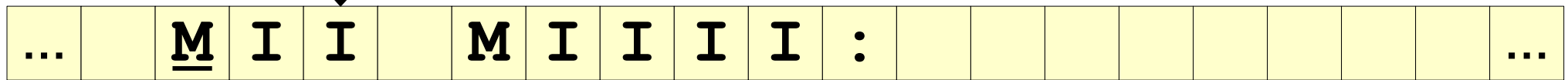
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

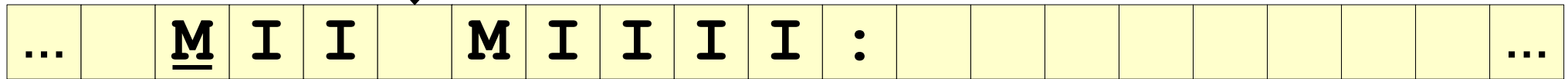
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

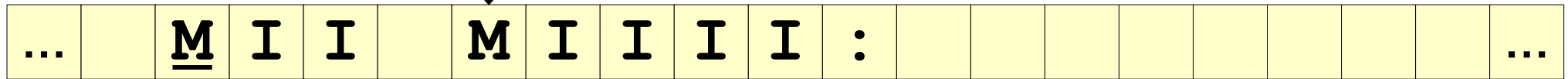
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

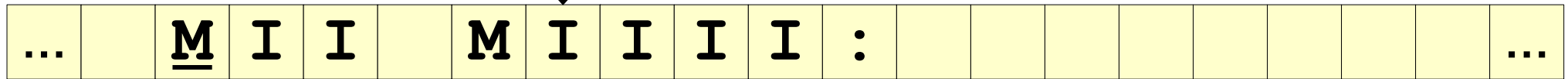
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

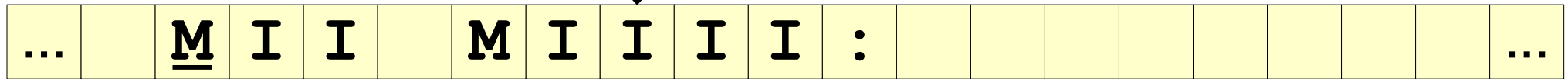
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

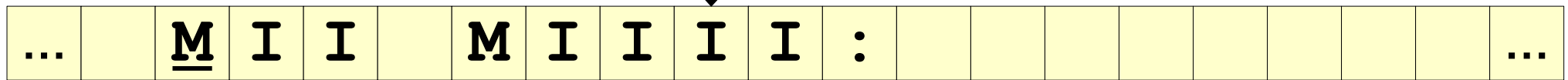
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

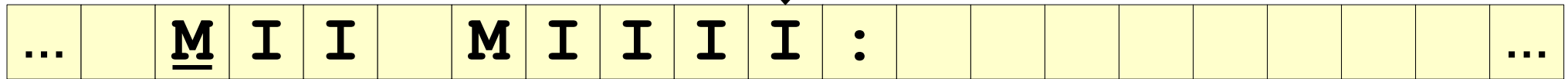
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

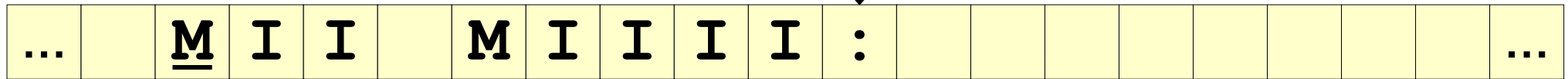
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

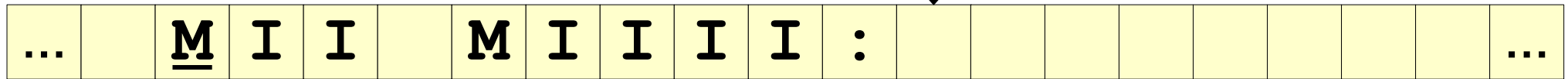
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

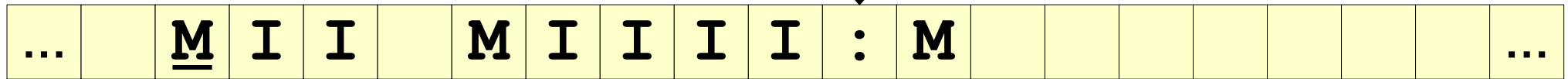
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

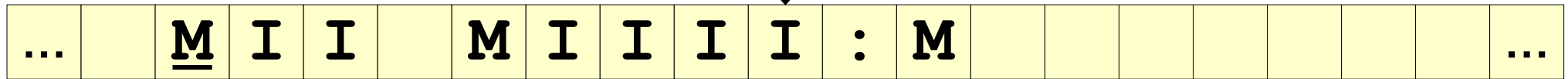
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

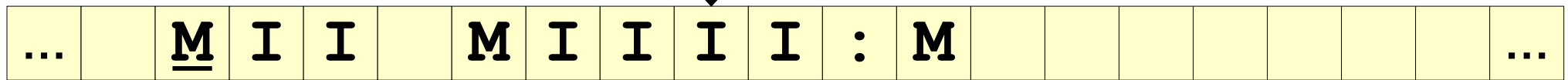
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

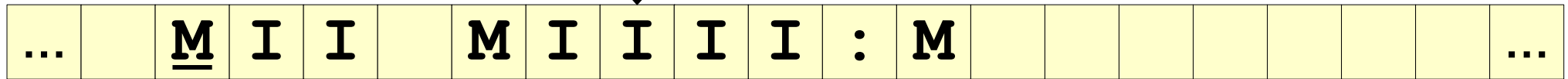
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

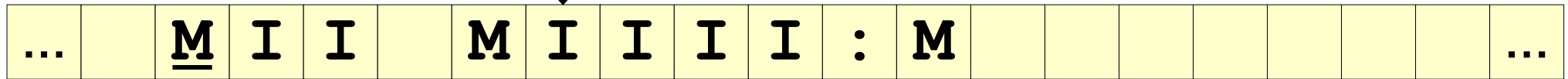
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

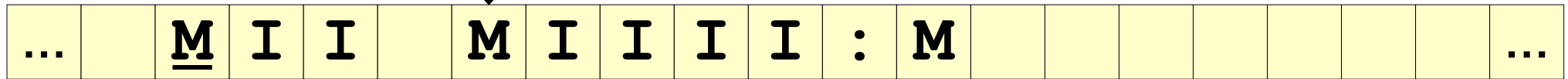
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

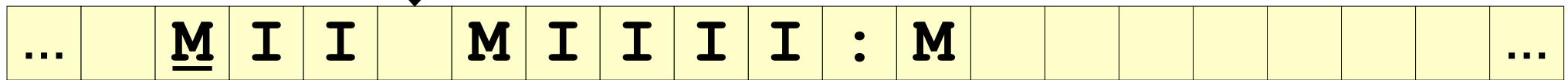
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

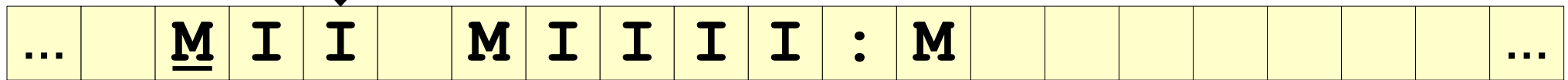
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

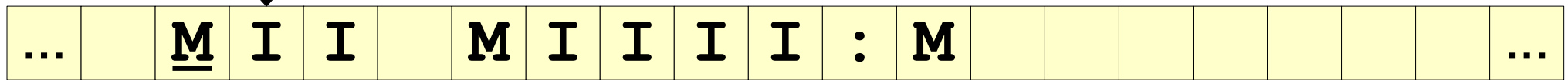
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

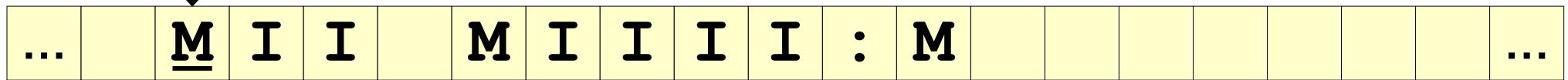
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

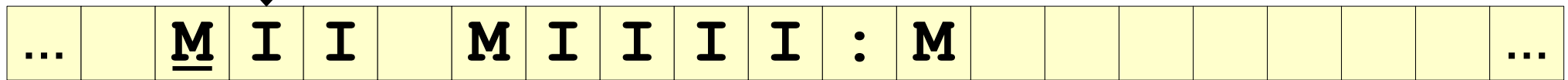
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

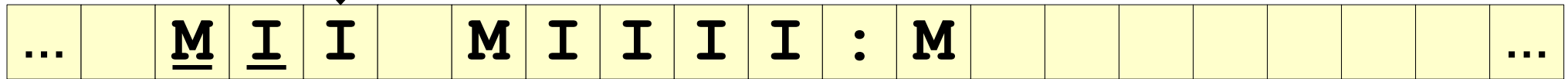
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

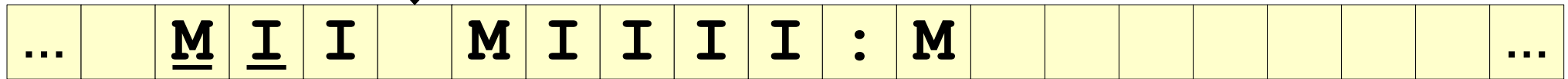
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

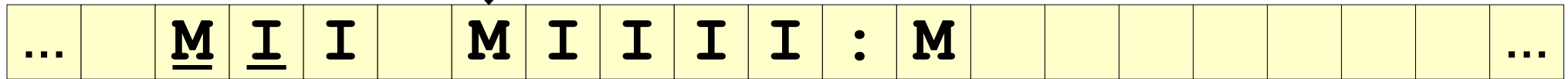
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

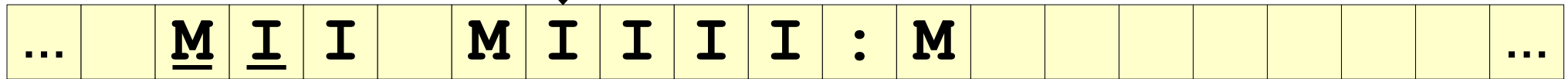
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

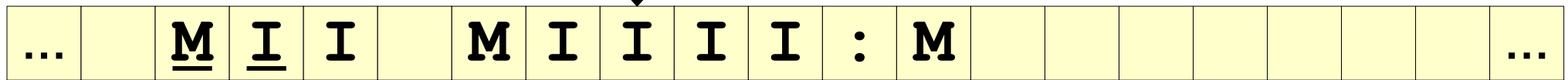
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

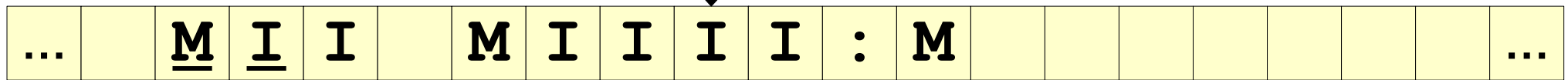
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

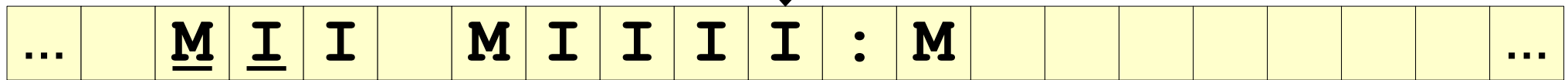
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

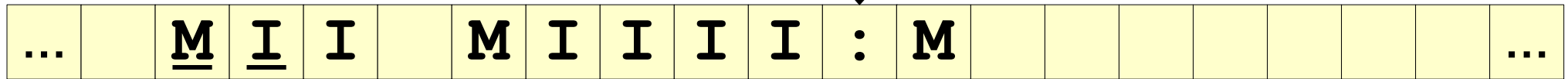
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

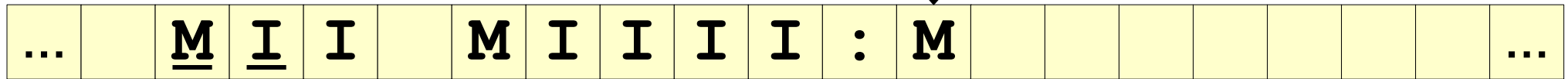
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

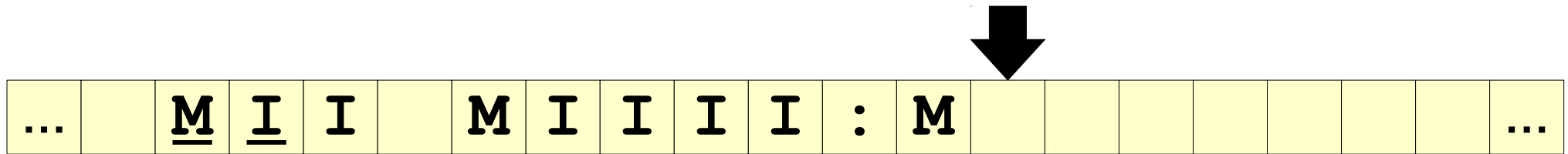
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

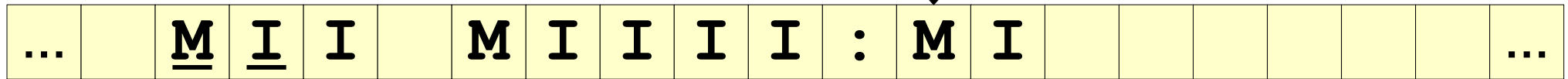
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

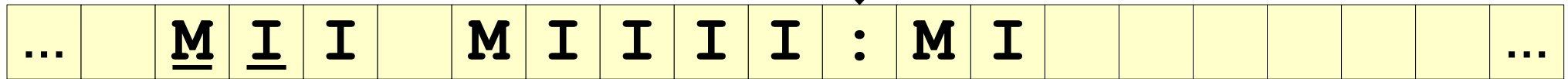
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

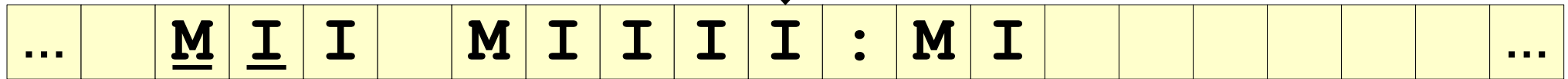
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

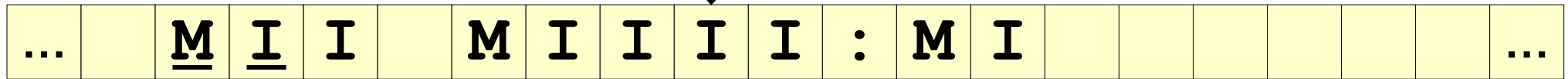
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

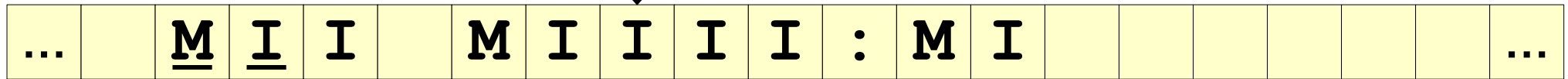
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

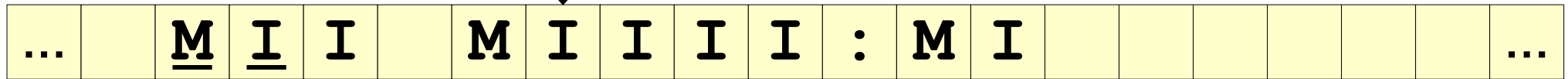
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

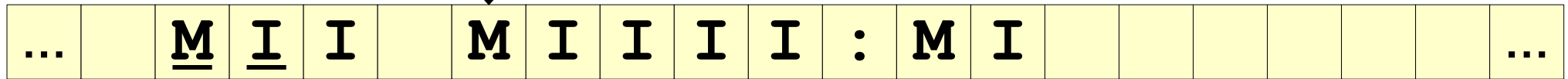
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

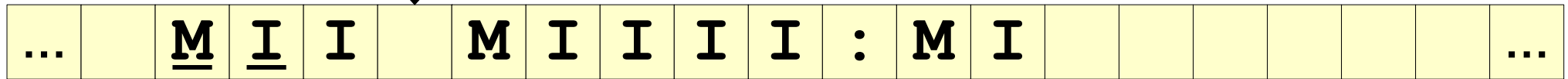
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

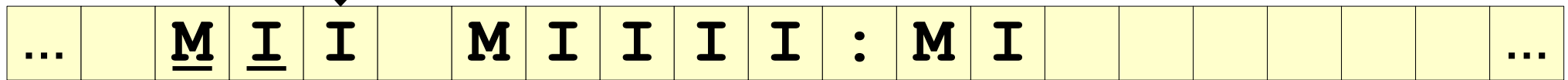
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

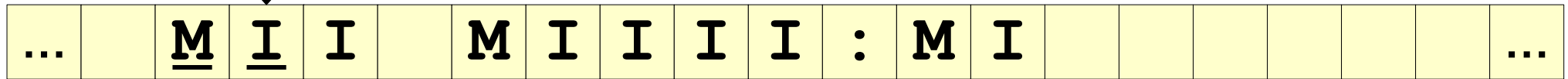
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

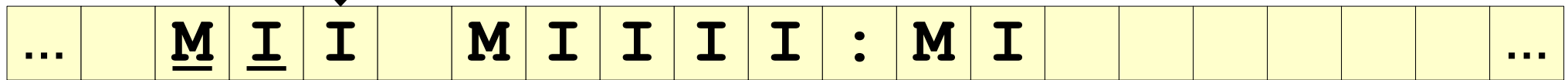
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

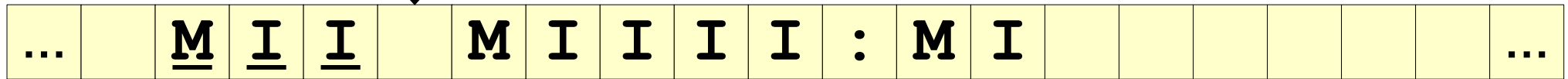
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

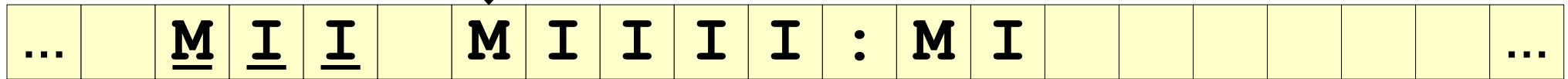
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

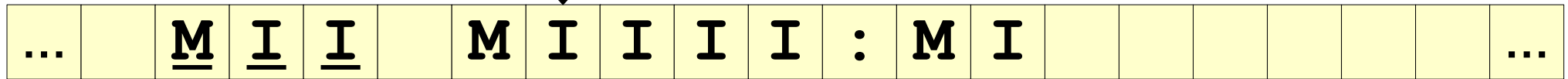
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

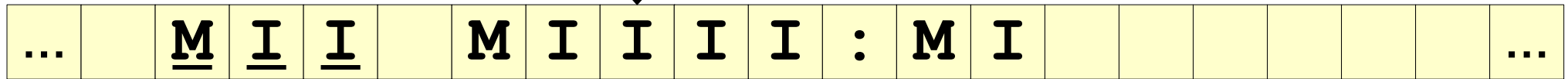
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

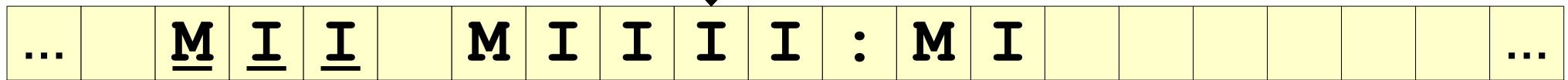
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

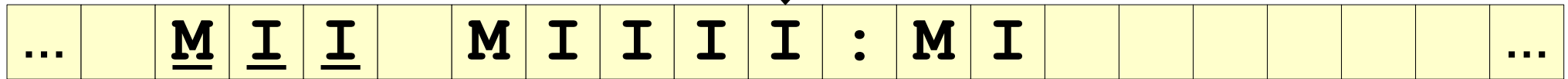
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

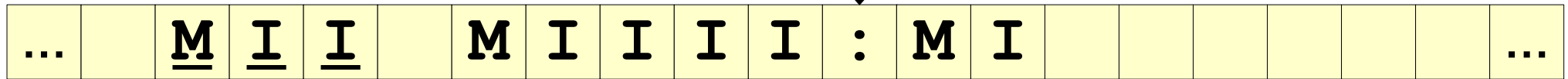
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

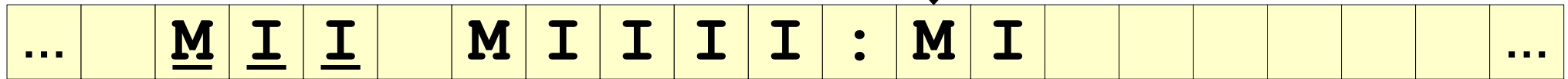
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

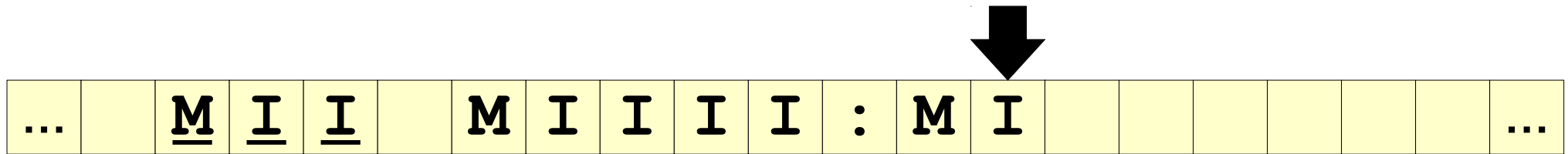
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

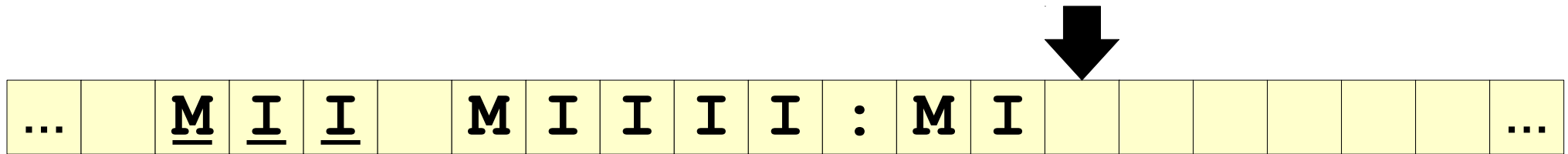
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

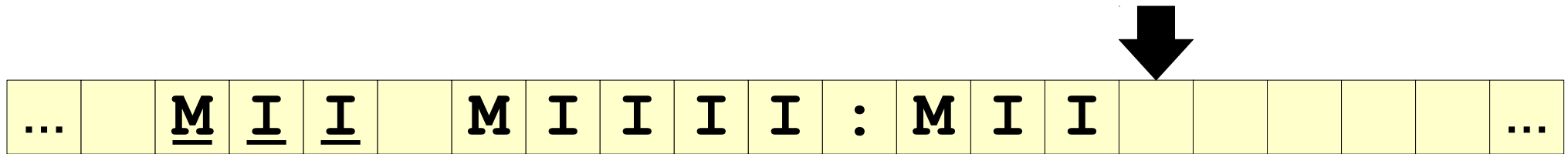
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MI**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

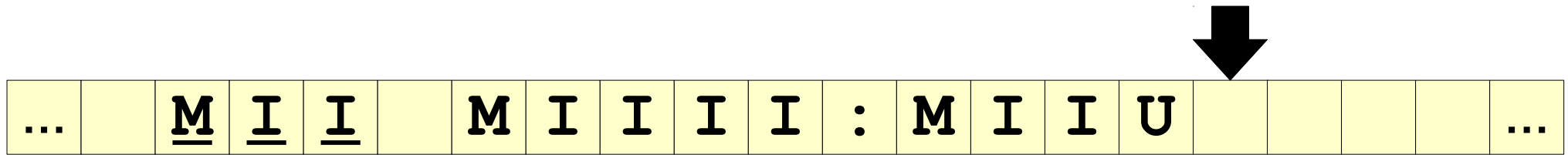
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

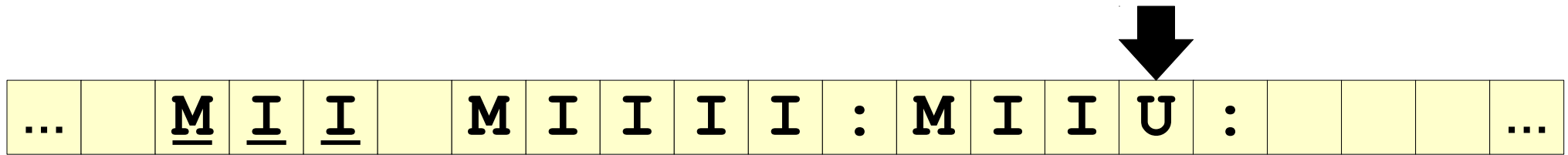
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

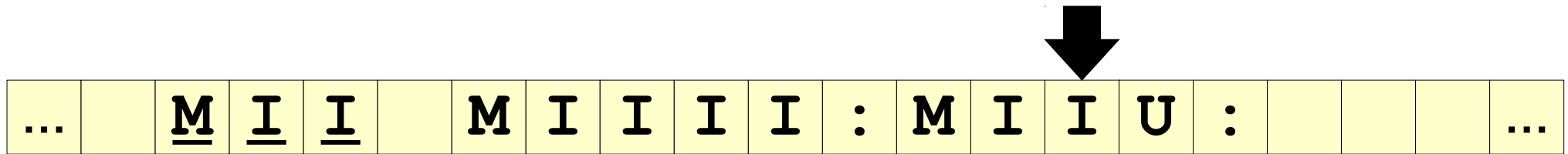
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

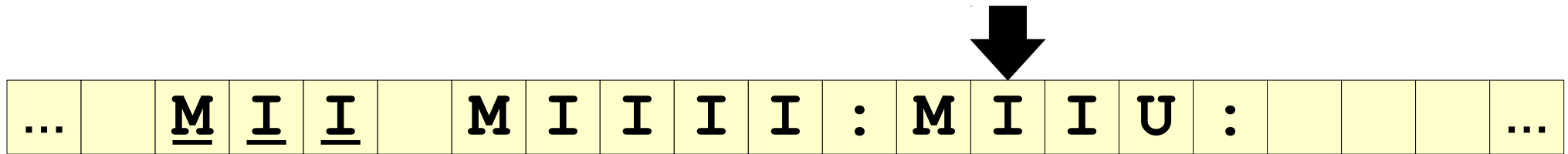
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

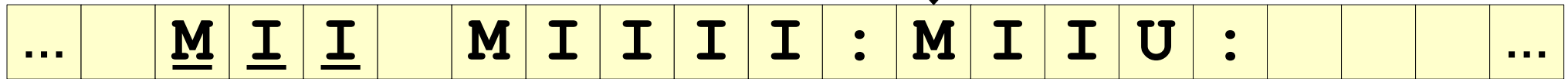
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MIU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

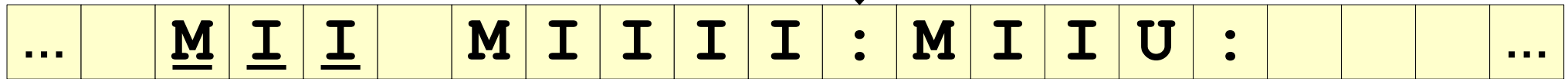
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

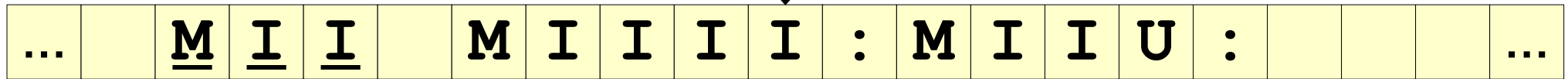
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

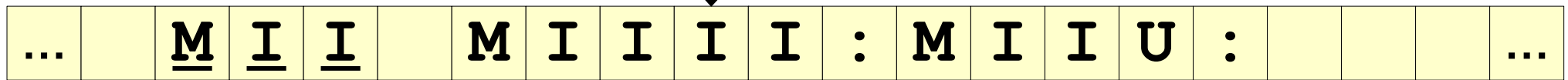
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

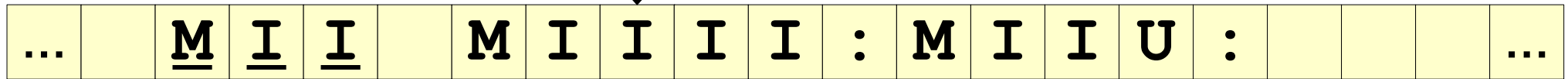
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

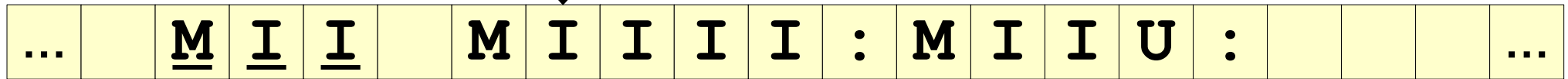
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

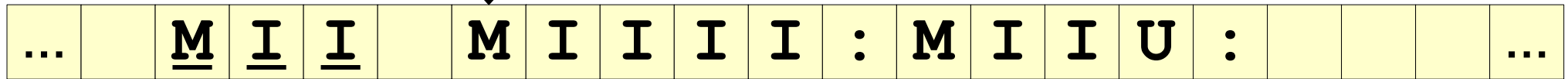
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

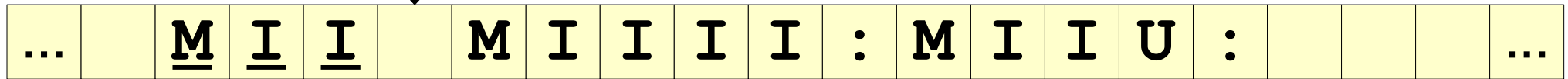
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

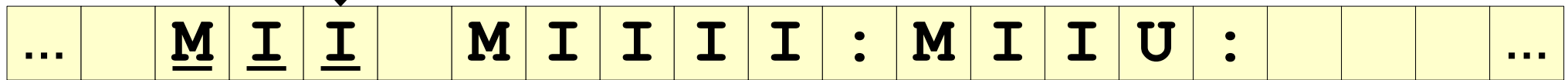
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

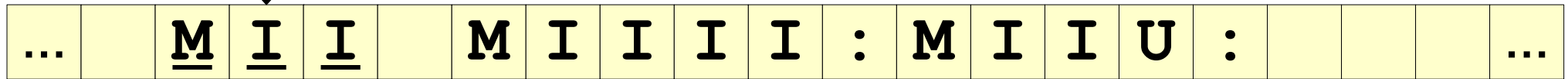
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

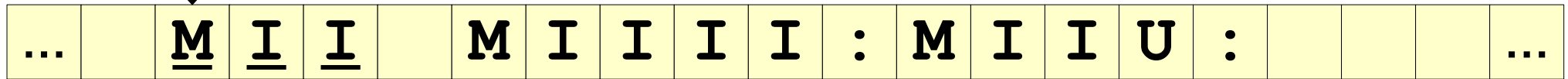
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

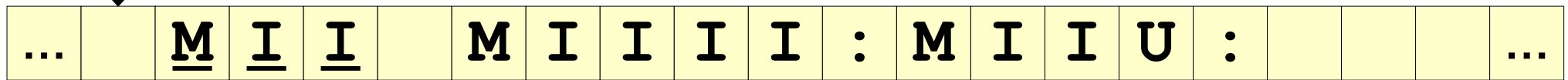
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

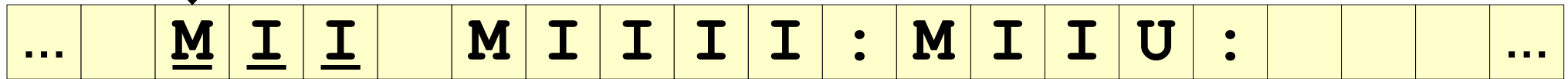
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

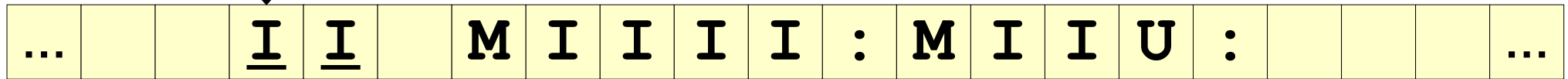
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

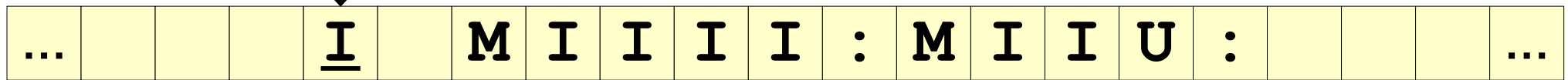
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

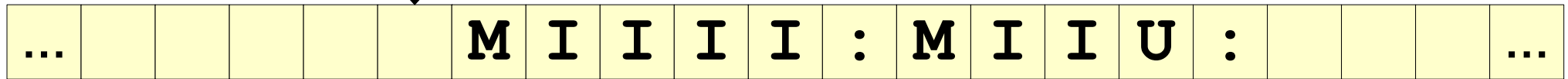
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

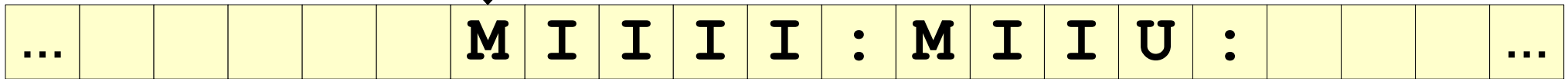
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

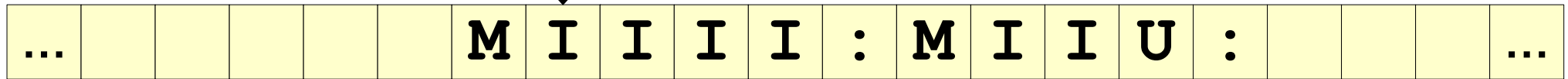
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

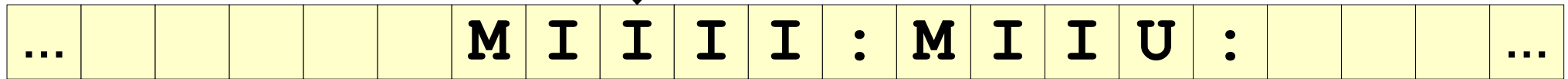
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

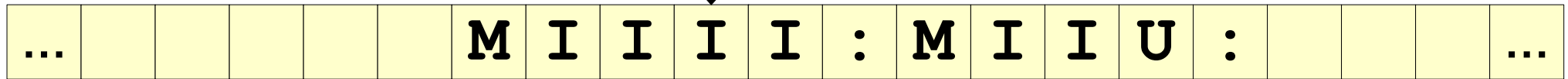
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

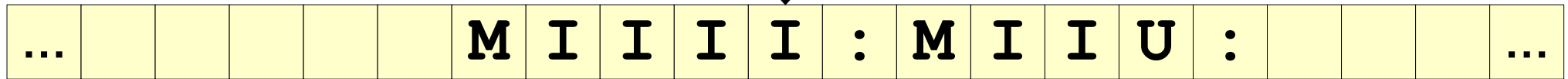
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

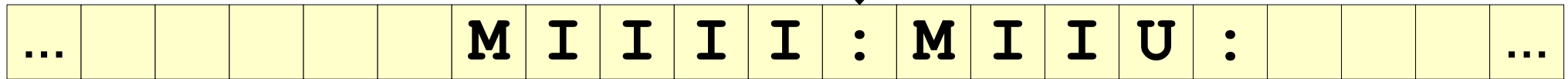
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

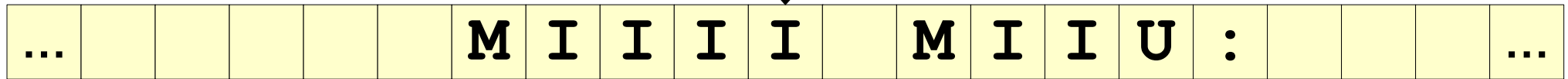
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

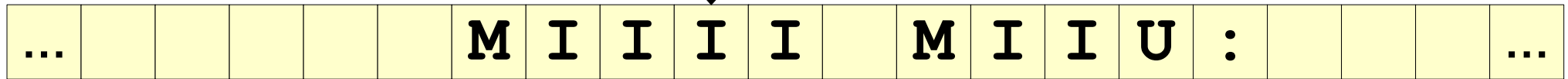
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

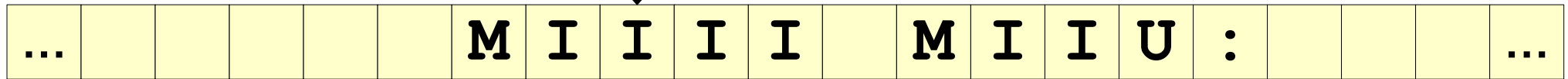
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

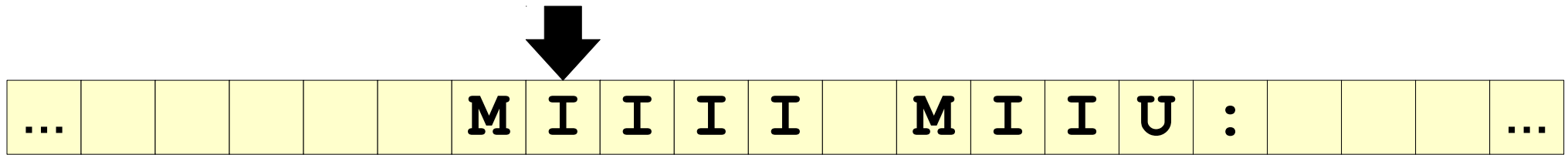
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

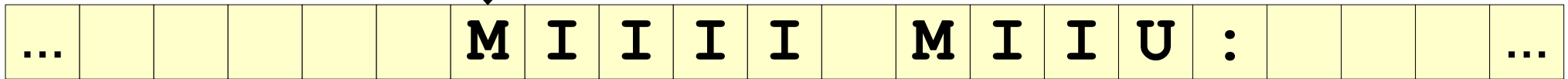
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

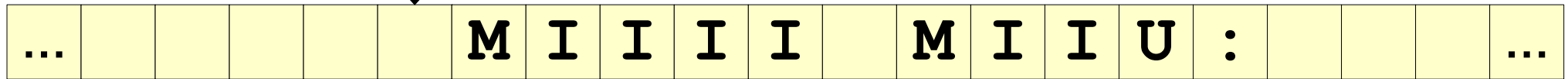
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

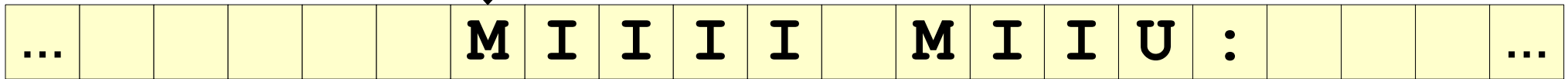
A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

A Sketch of the Turing Machine



To recognize L , our TM M does the following:

- While M has not found the string **MU**:
 - M grabs the next string from the worklist.
 - For each possible single step, M performs that step and appends the result to the worklist.

The Power of TMs

- The worklist approach makes that all of the following languages are recognizable:
 - Any context-free language: simulate all possible production rules and see if the target string can be derived.
 - Solving a maze – use the worklist to explore all paths of length 0, 1, 2, ... until a solution is found.
 - Determining whether a polynomial has an integer zeros: try 0, -1, +1, -2, +2, -3, +3, ... until a result is found.

Why “Recognizable?”

- Given TM M with language $\mathcal{L}(M)$, running M on a string w will not necessarily tell you whether $w \in \mathcal{L}(M)$.
- If the machine is running, as an observer, you can't tell whether
 - it is eventually going to halt, but just needs more time, or
 - it is never going to halt.
- However, if you know for a fact that $w \in \mathcal{L}(M)$, then the machine can confirm this (it eventually accepts).
- The machine can't *decide* whether or not $w \in \mathcal{L}(M)$, but it can *recognize* strings that are in the language.
- We sometimes call a TM for a language L a **recognizer** for L .

Is this a satisfactory definition
of “solving” a problem?