Turing Machines

Part II

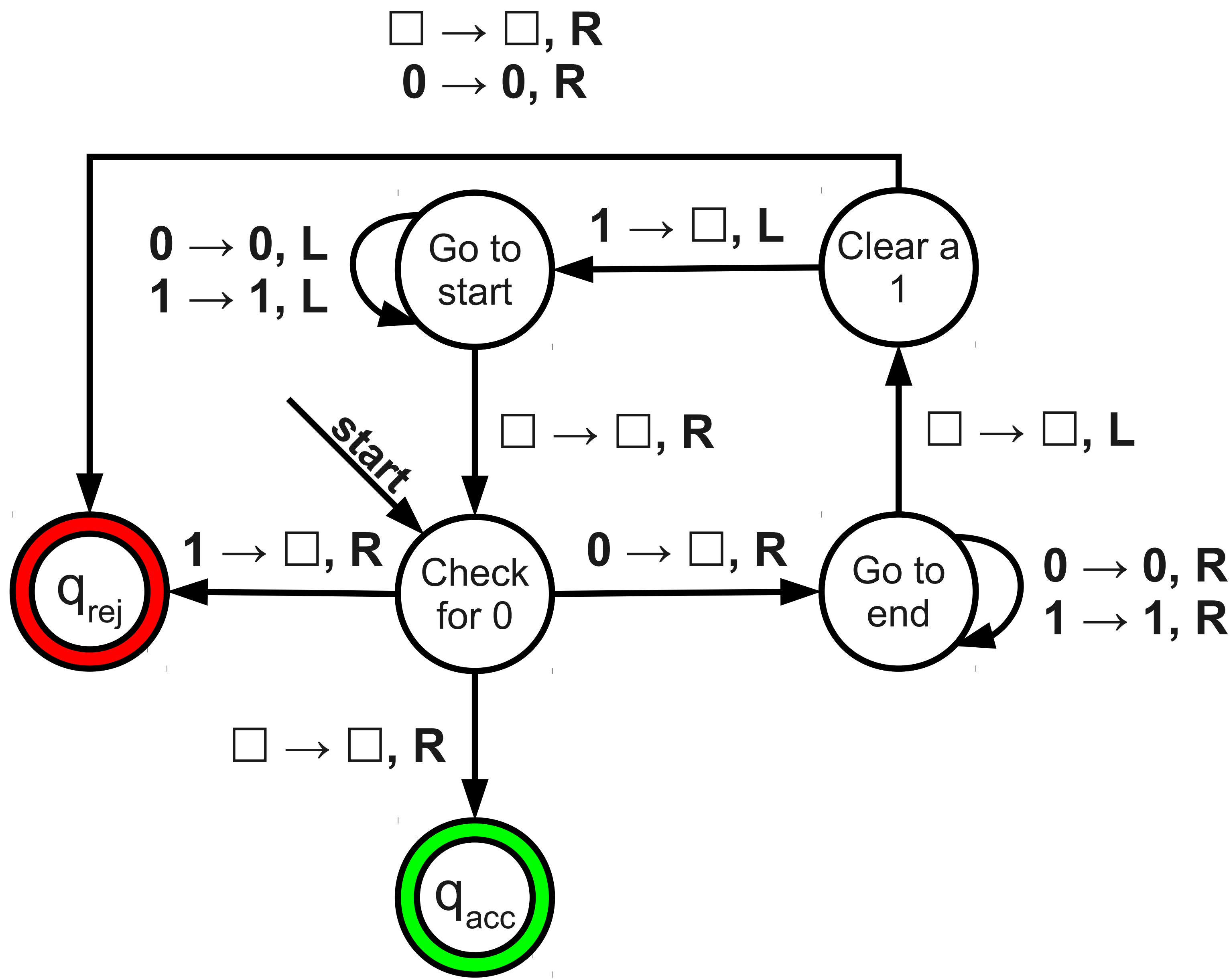
Friday Four Square!
Today at 4:15PM, Outside Gates

Announcements

- Problem Set 6 due next Monday, February 25, at 12:50PM.
    - Email the staff list with questions! (**cs103-win1213-staff@lists.stanford.edu**)

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.



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

Key Idea: Subroutines

- A **subroutine** of a Turing machine is a small set of states in the TM such that performs a small computation.
- Usually, a single entry state and a single exit state.
- Many very complicated tasks can be performed by TMs by breaking those tasks into smaller subroutines.

Turing Machine Memory

- Turing machines often contain many seemingly replicated states in order to store a finite amount of extra information.
- A Turing machine can remember one of k different constants by copying its states k times, once for each possible value, and wiring those states appropriately.

The Power of Turing Machines

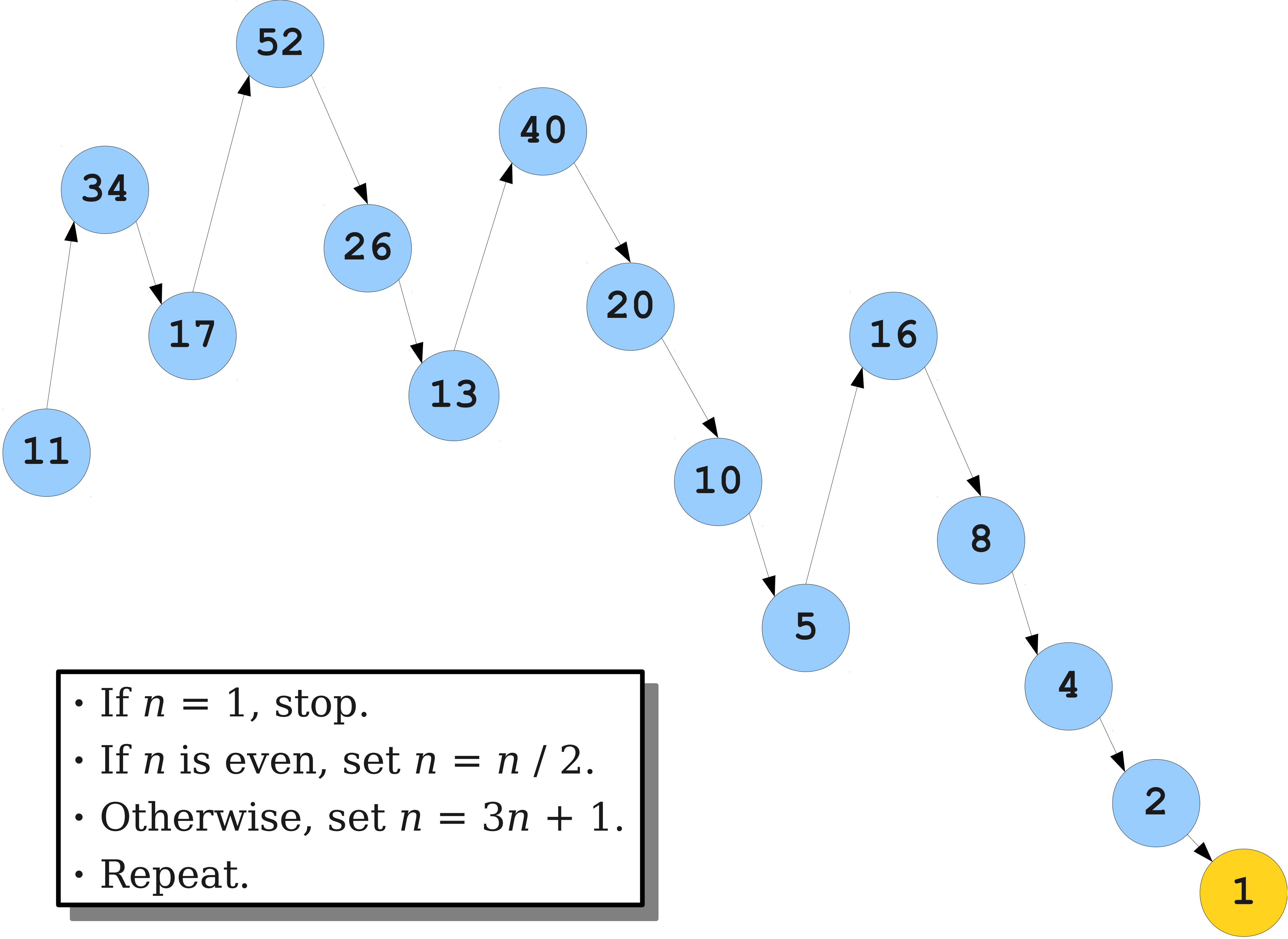
- Turing machines can
  - Perform standard arithmetic operations (addition, subtraction, multiplication, division, exponentiation, etc.)
  - Manipulate lists of elements (searching, sorting, reversing, etc.)
- What else can Turing machines do?

Outline for Today

- **Exhaustive Search**
  - A fundamentally different approach to designing Turing machines.
- **Nondeterministic Turing Machines**
  - What does a Turing machine with Magic Superpowers look like?
- **The Church-Turing Thesis**
  - Just how powerful *are* Turing machines?

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 $\Sigma = \{\mathtt{1}\}$ and consider the language

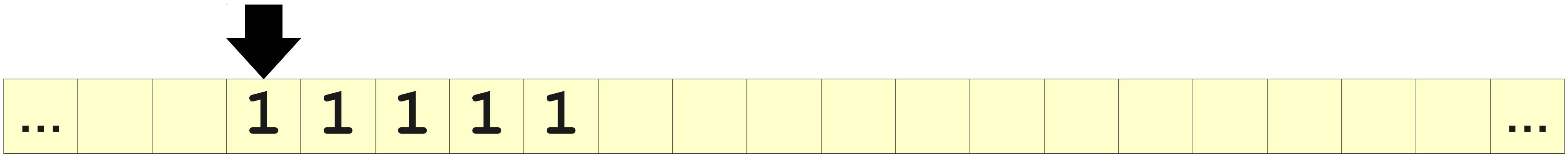
$$L = \{\ \mathtt{1}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n\ \}.$$

- 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



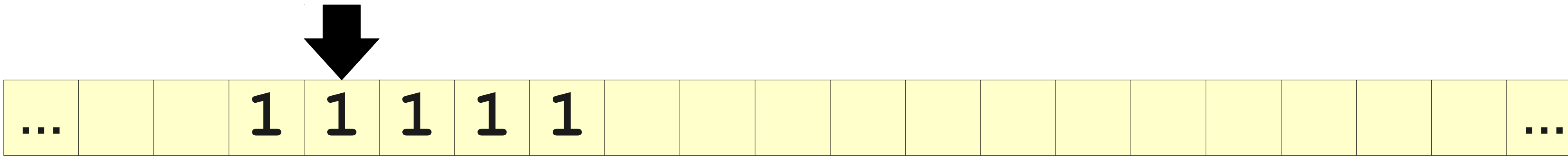
If the input is ε, reject.

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



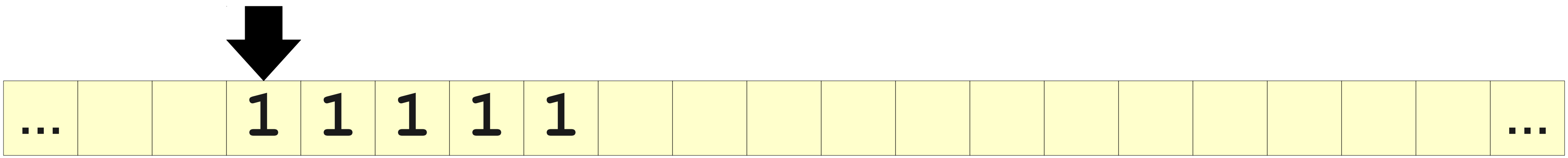
If the input is ε, reject.

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



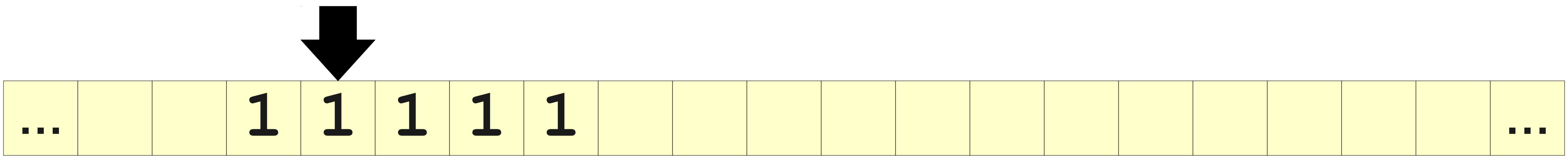
If the input is ε, reject.

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



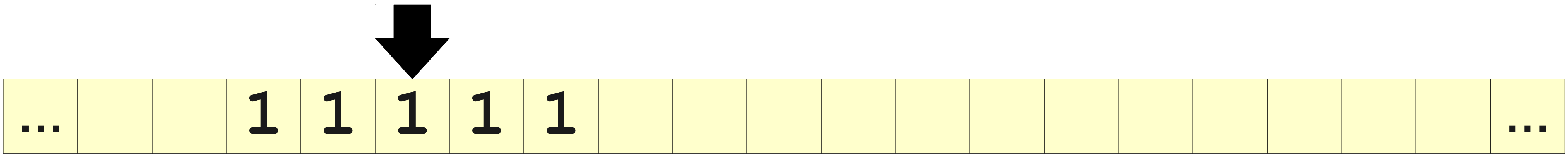
If the input is ε, reject.

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



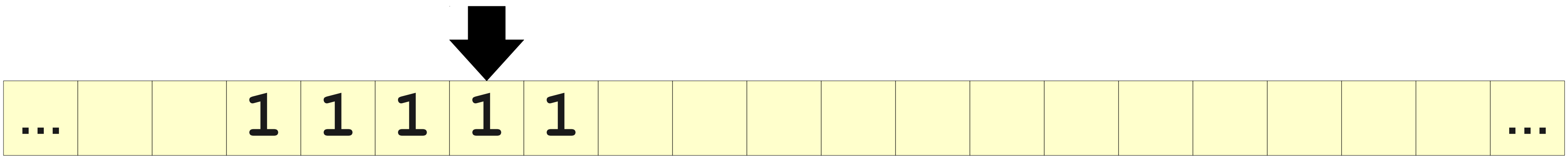
If the input is ε, reject.

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



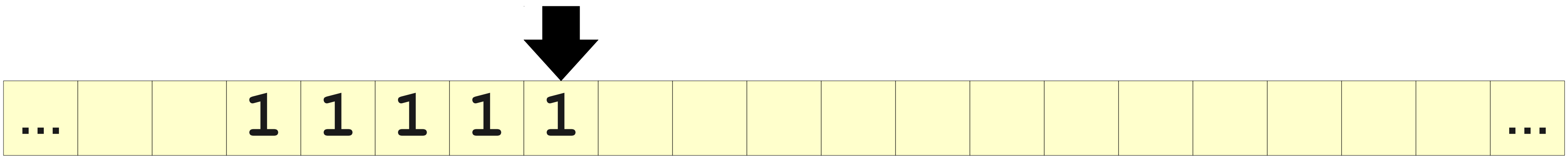
If the input is ε, reject.

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



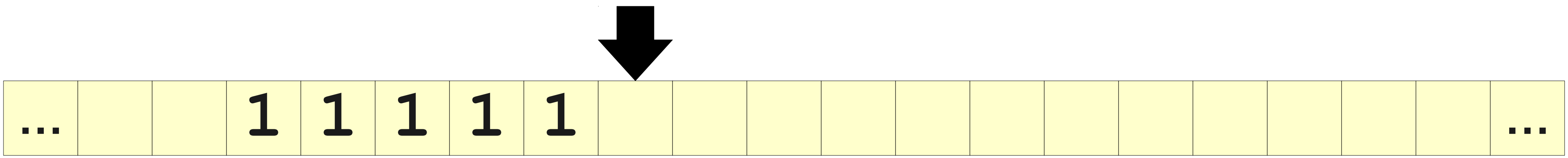
If the input is ε, reject.

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



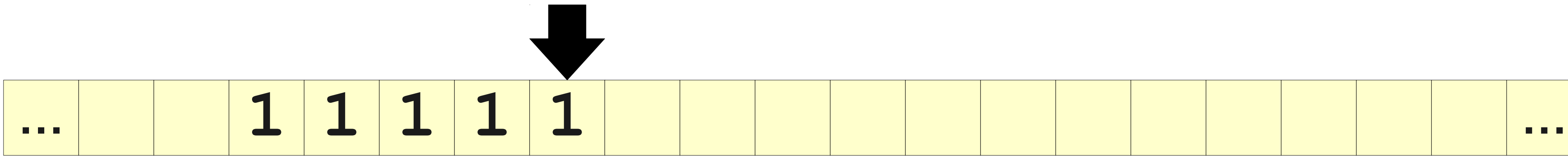
If the input is ε, reject.

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



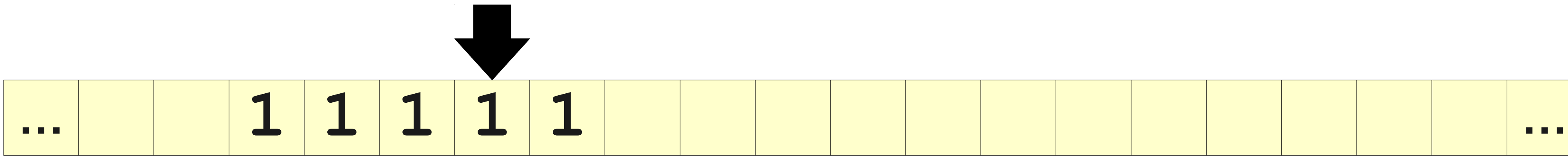
If the input is ε, reject.

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



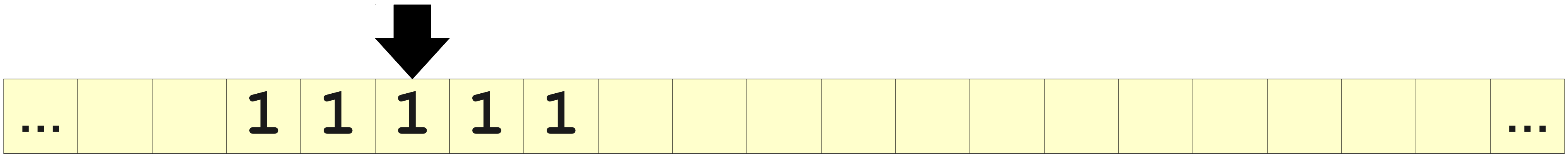
If the input is ε, reject.

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



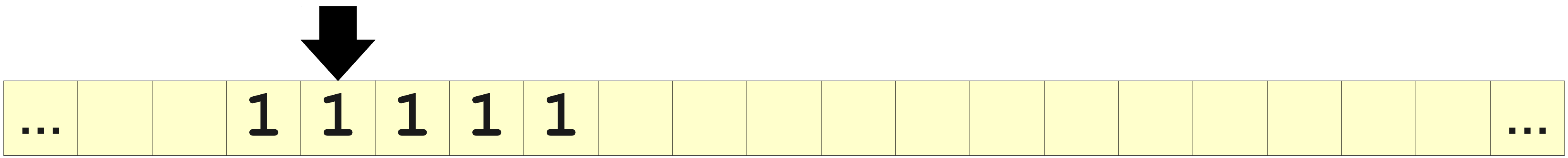
If the input is ε, reject.

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



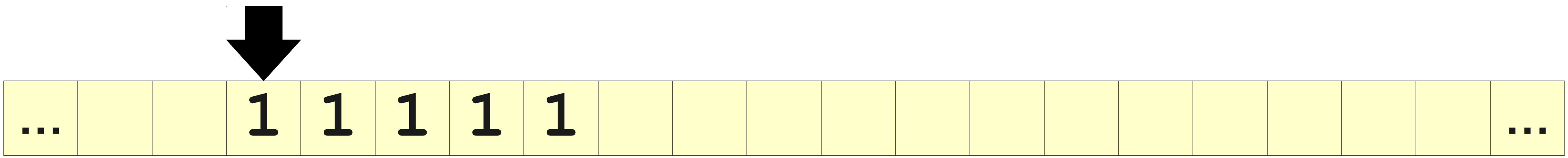
If the input is ε, reject.

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



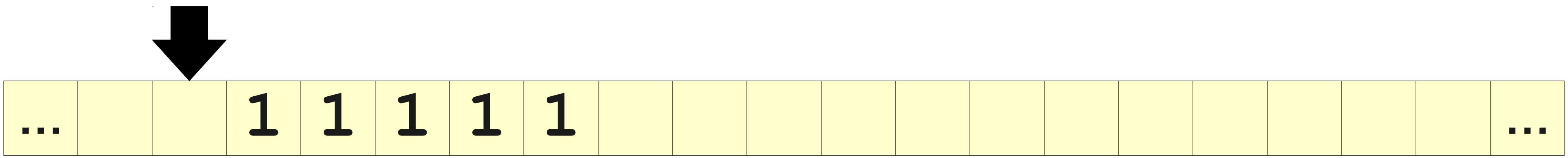
If the input is ε, reject.

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



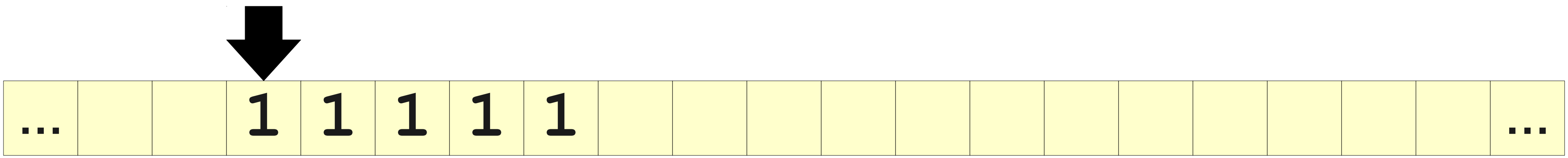
If the input is ε, reject.

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



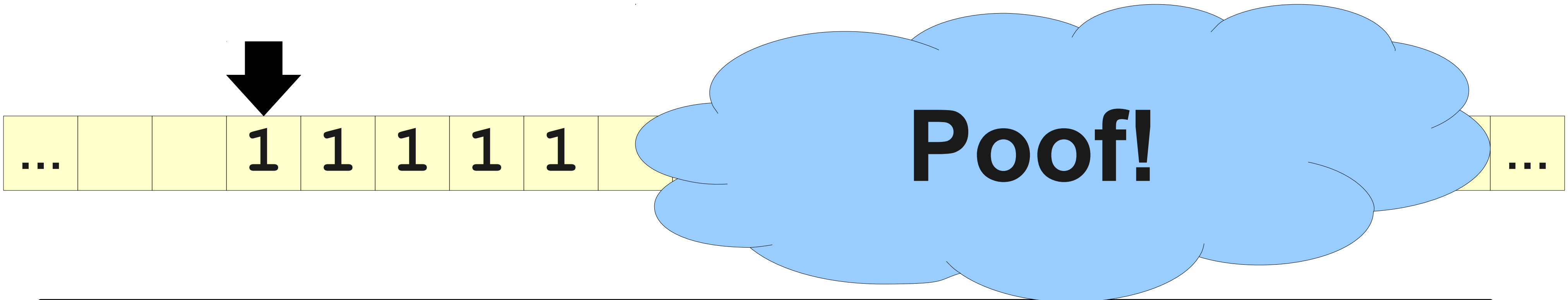
If the input is ε, reject.

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



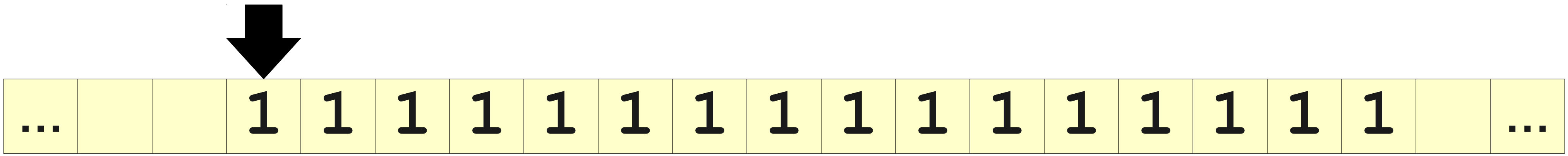
If the input is ε, reject.

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



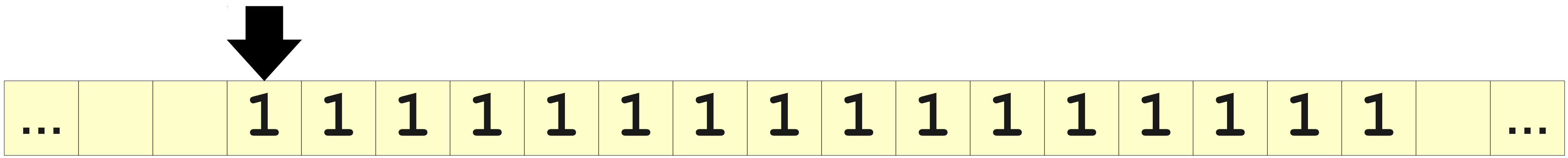
If the input is ε, reject.

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.

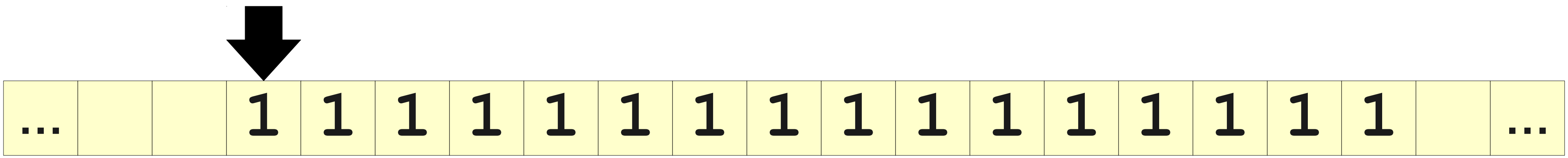
While the input is not 1:

- If the input has even len[...] of the string.
- If the input has odd leng[...] string and append a 1.

Accept.

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

The Hailstone Turing Machine



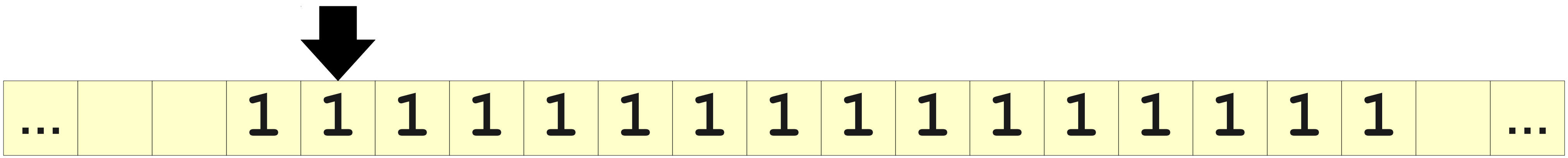
If the input is ε, reject.

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



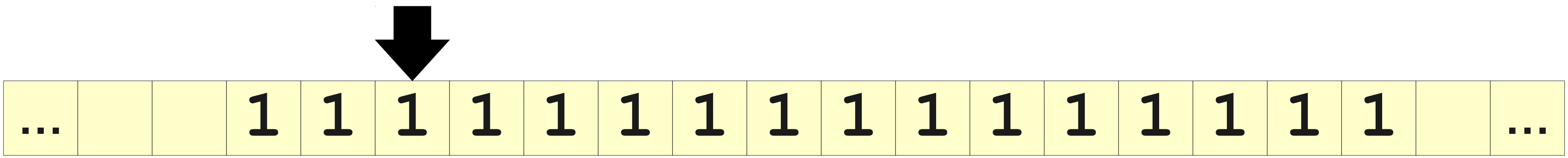
If the input is ε, reject.

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



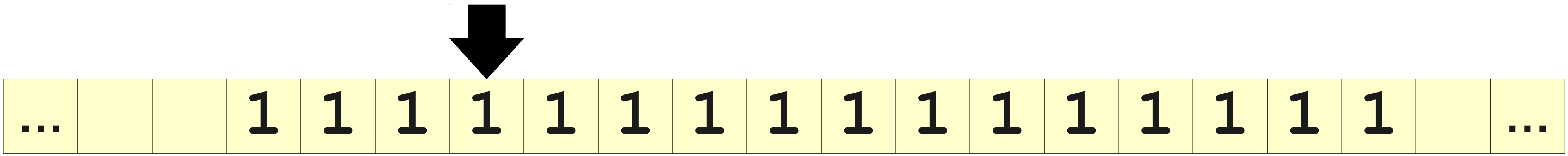
If the input is ε, reject.

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



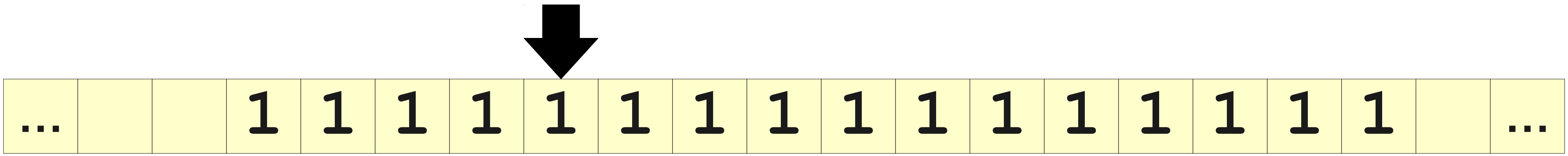
If the input is ε, reject.

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



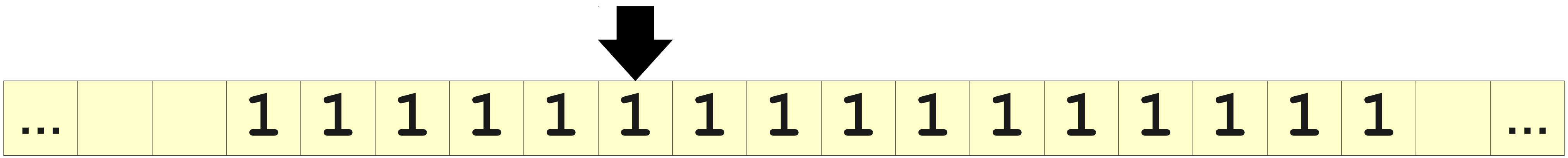
If the input is ε, reject.

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



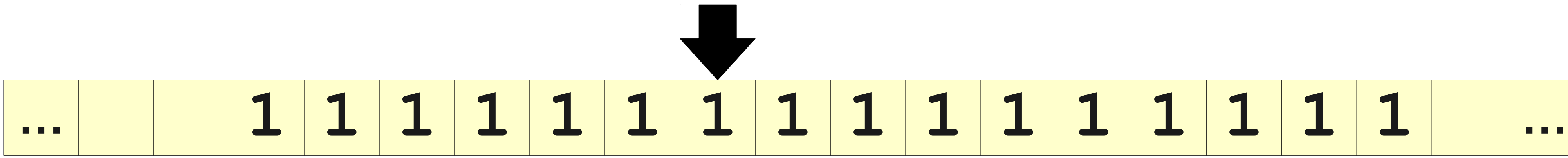
If the input is ε, reject.

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



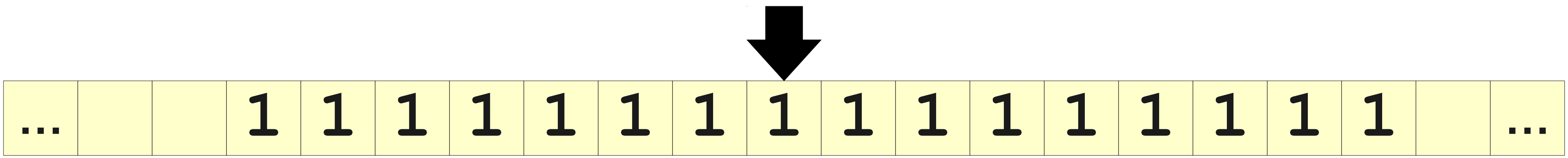
If the input is ε, reject.

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



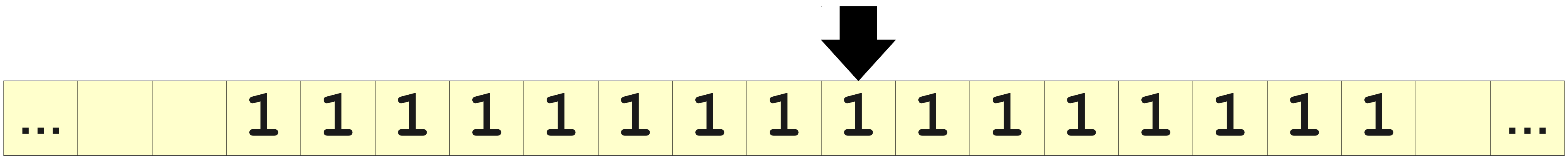
If the input is ε, reject.

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



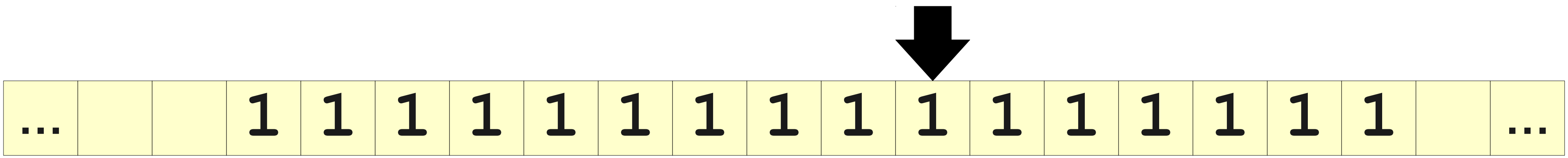
If the input is ε, reject.

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



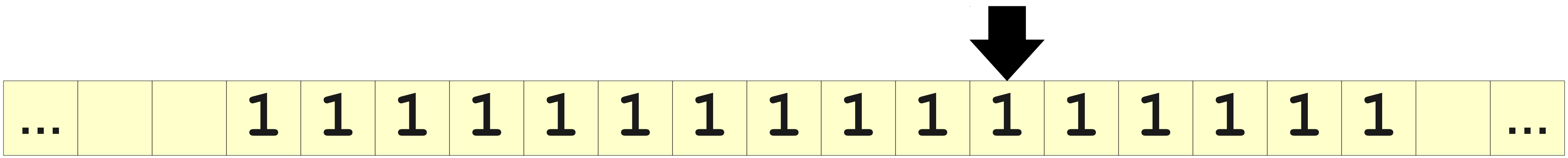
If the input is ε, reject.

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



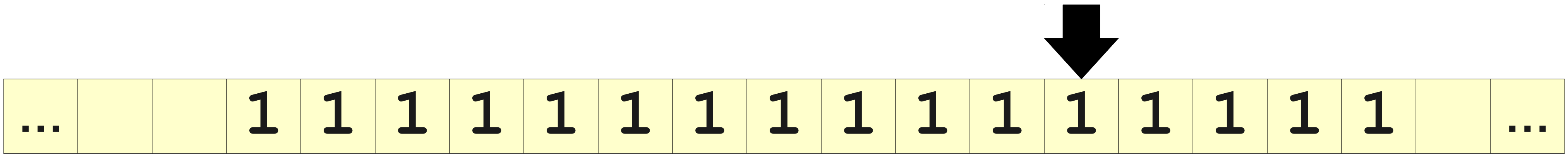
If the input is ε, reject.

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



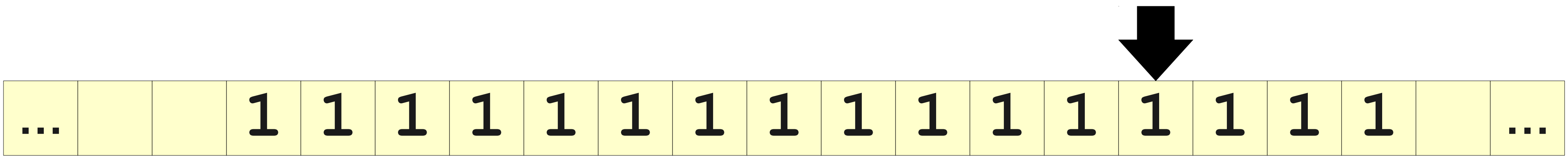
If the input is ε, reject.

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



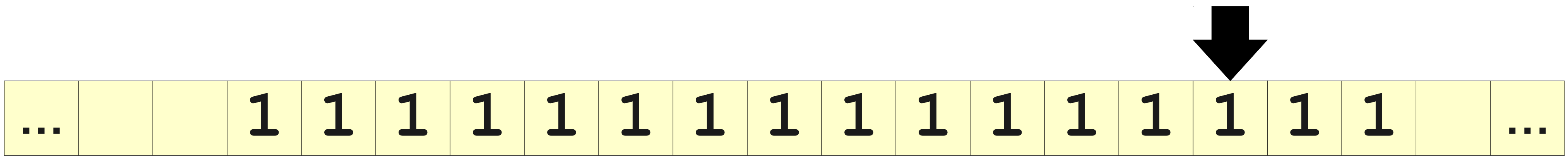
If the input is ε, reject.

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



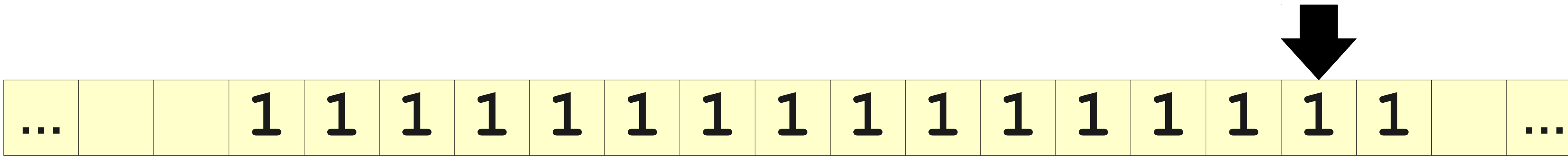
If the input is ε, reject.

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



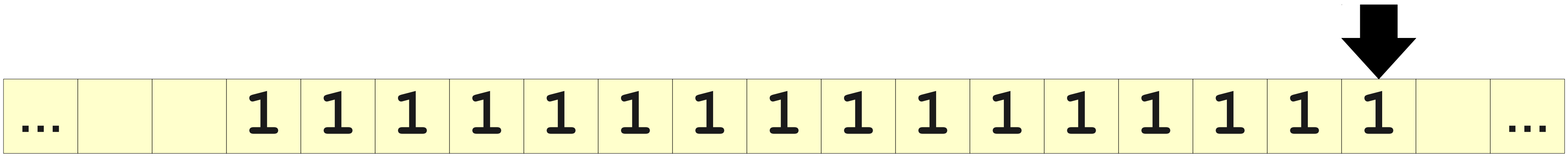
If the input is ε, reject.

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



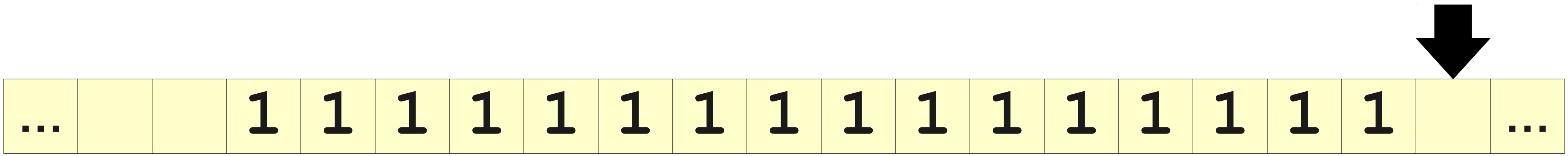
If the input is ε, reject.

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



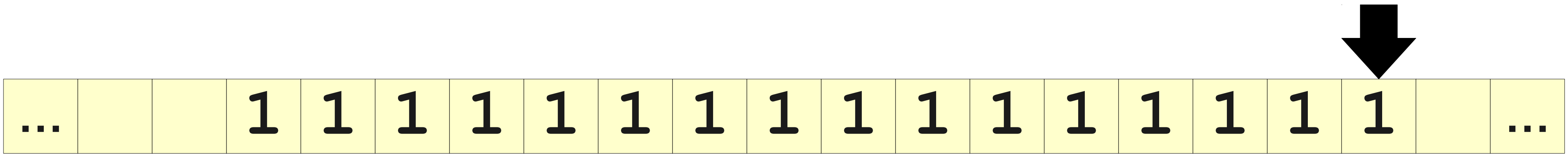
If the input is ε, reject.

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



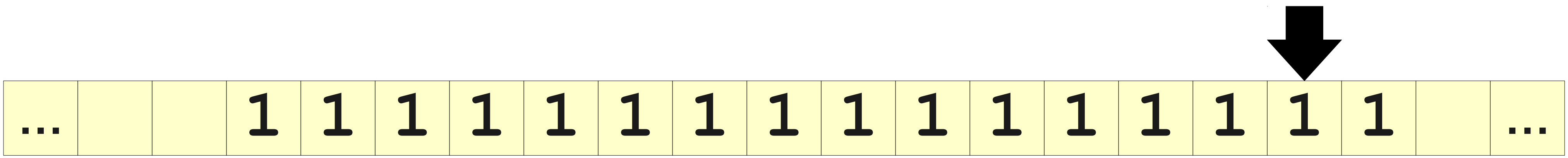
If the input is ε, reject.

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



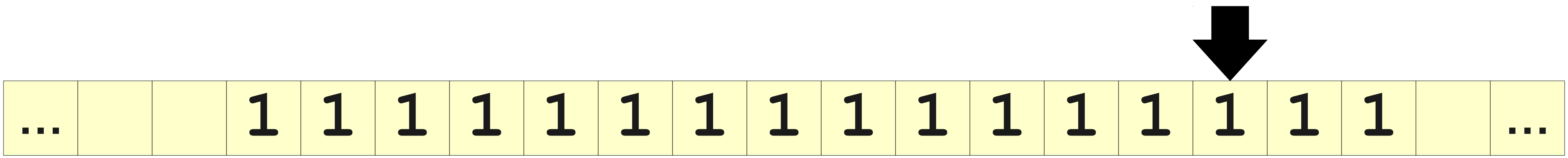
If the input is ε, reject.

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



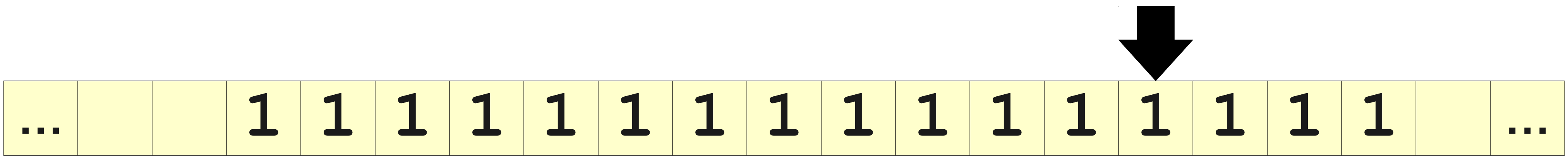
If the input is ε, reject.

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



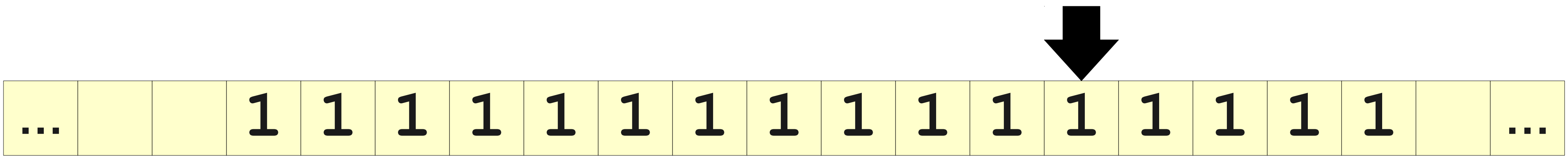
If the input is ε, reject.

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



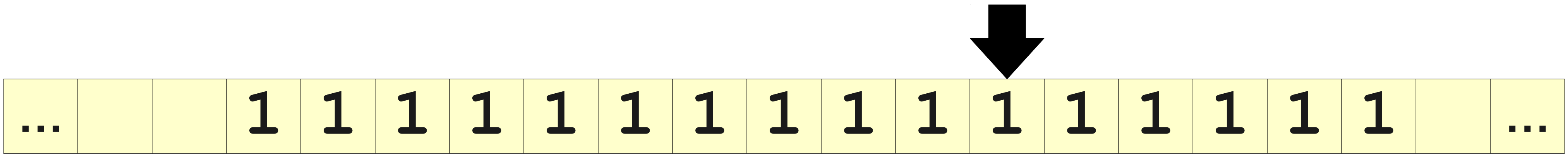
If the input is ε, reject.

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



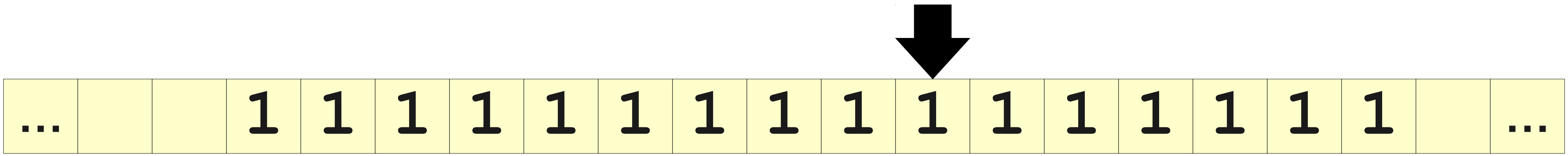
If the input is ε, reject.

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



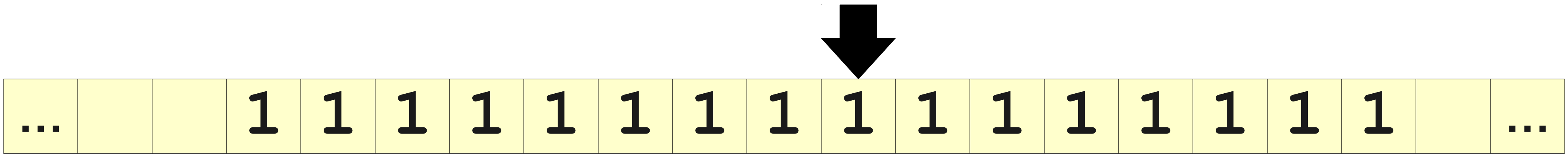
If the input is ε, reject.

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



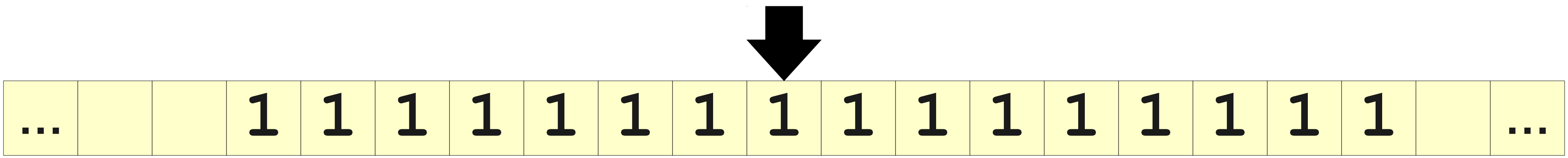
If the input is ε, reject.

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



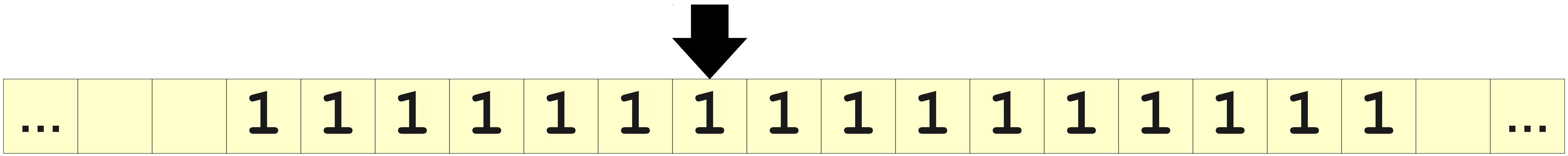
If the input is ε, reject.

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



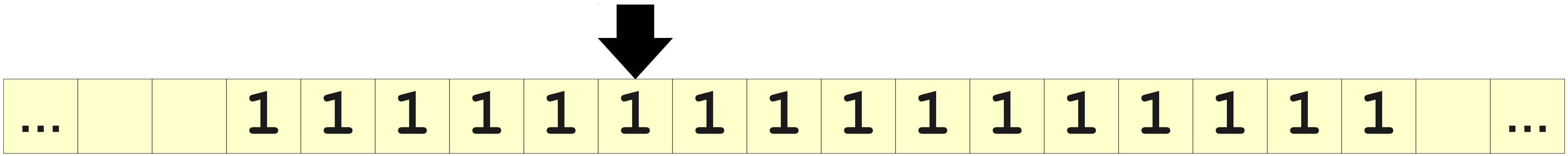
If the input is ε, reject.

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



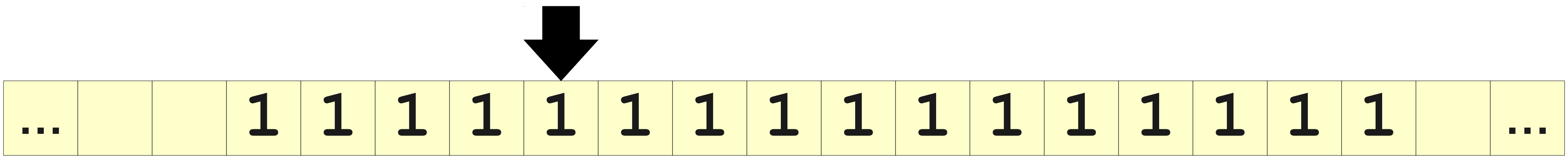
If the input is ε, reject.

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



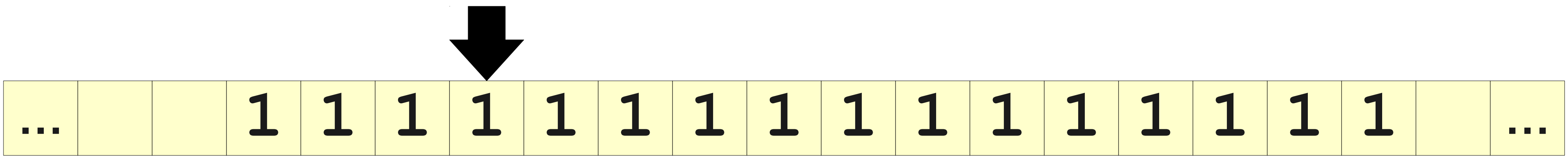
If the input is ε, reject.

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



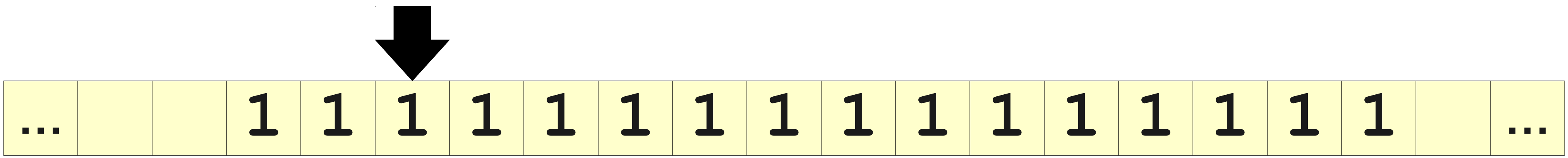
If the input is ε, reject.

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



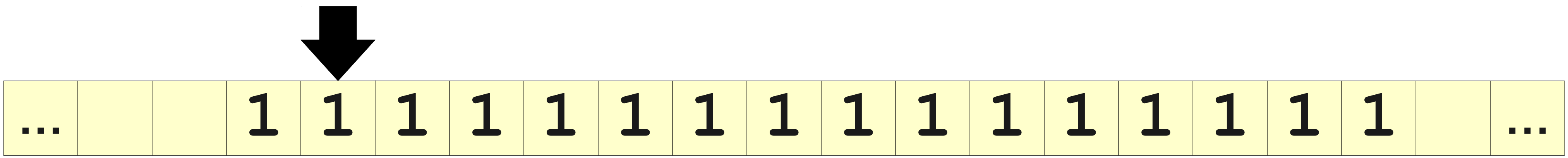
If the input is ε, reject.

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



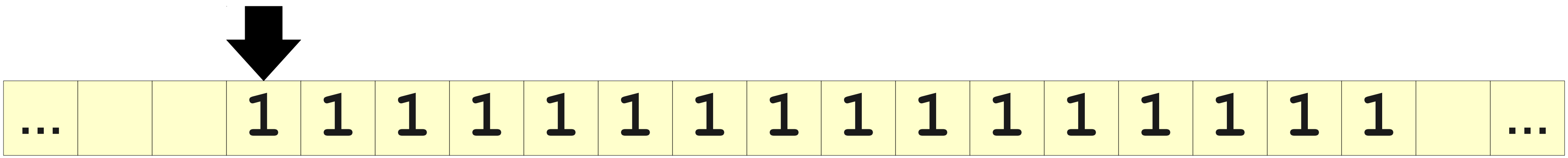
If the input is ε, reject.

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



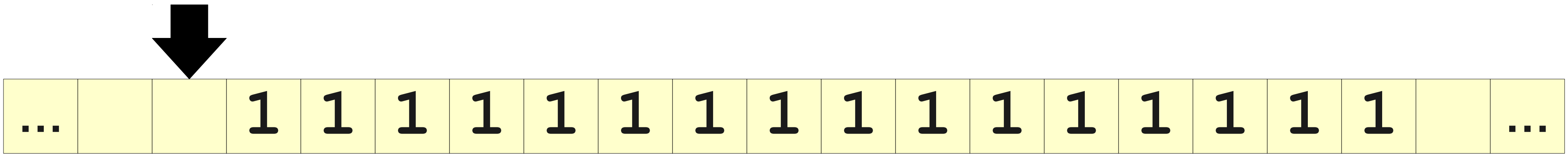
If the input is ε, reject.

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



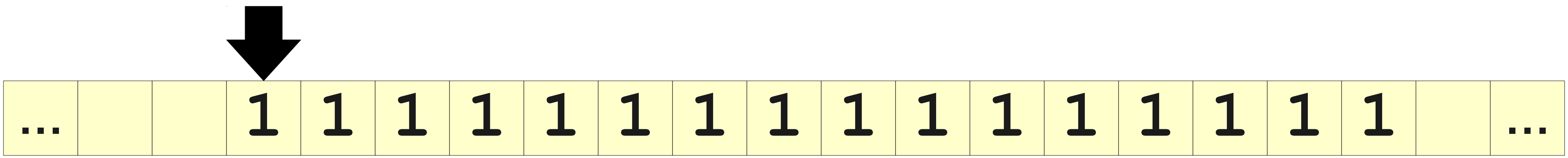
If the input is ε, reject.

While the input is not **1**:

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

Accept.

The Hailstone Turing Machine



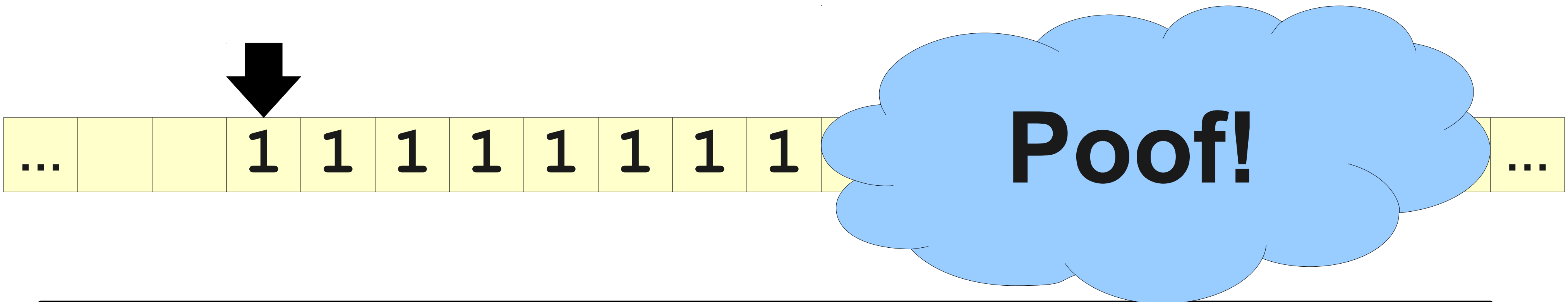
If the input is ε, reject.

While the input is not **1**:

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

Accept.

The Hailstone Turing Machine



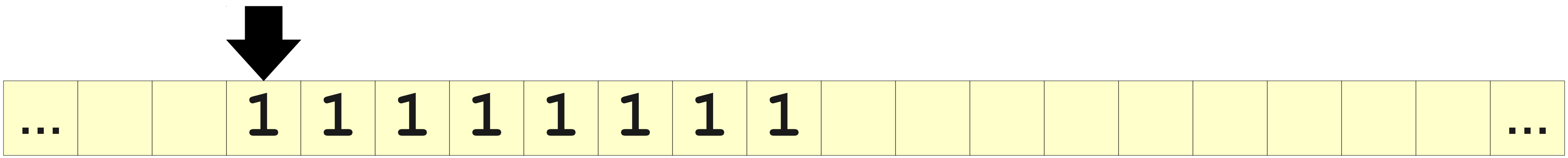
If the input is ε, reject.

While the input is not **1**:

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

Accept.

The Hailstone Turing Machine



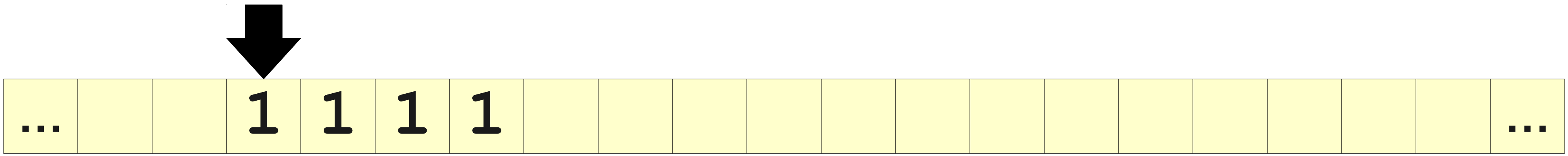
If the input is ε, reject.

While the input is not `1`:

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

Accept.

The Hailstone Turing Machine



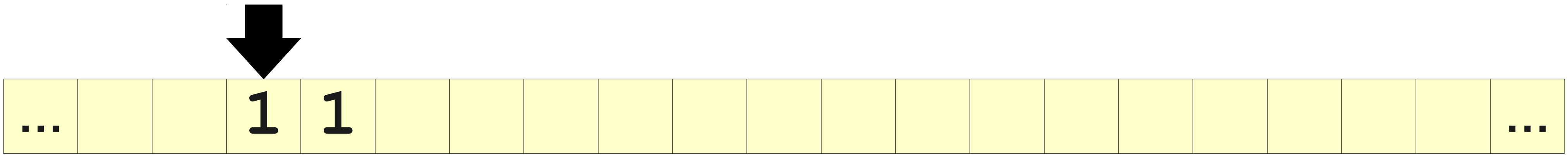
If the input is ε, reject.

While the input is not **1**:

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

Accept.

The Hailstone Turing Machine



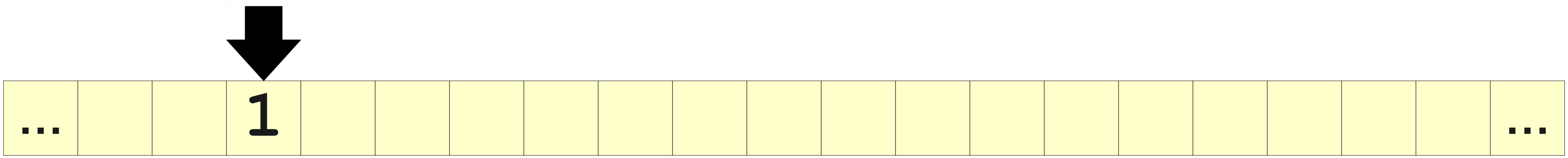
If the input is ε, reject.

While the input is not **1**:

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

Accept.

The Hailstone Turing Machine



If the input is ε, reject.

While the input is not **1**:

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

Accept.

Does this Turing machine always accept?

The Collatz Conjecture

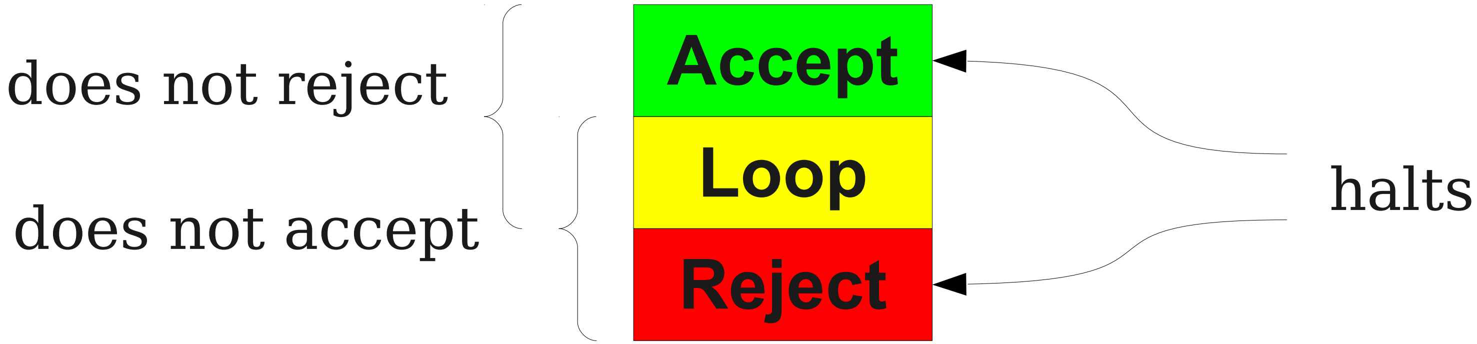
- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, ***no one knows whether the TM described in the previous slides will always stop running!***
- The conjecture (claim) that this always terminates is called the **Collatz Conjecture**.

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.

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*** 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 $\mathscr{L}(M)$, is the set of all strings that M accepts:

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

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

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

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`**.

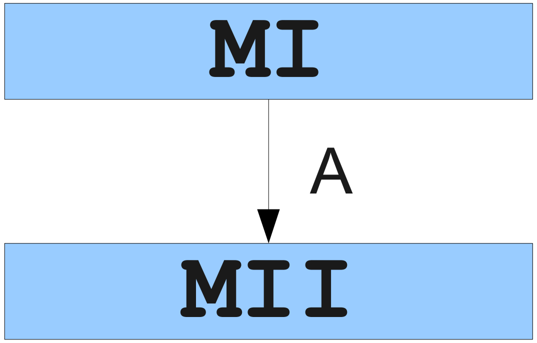
MI

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**

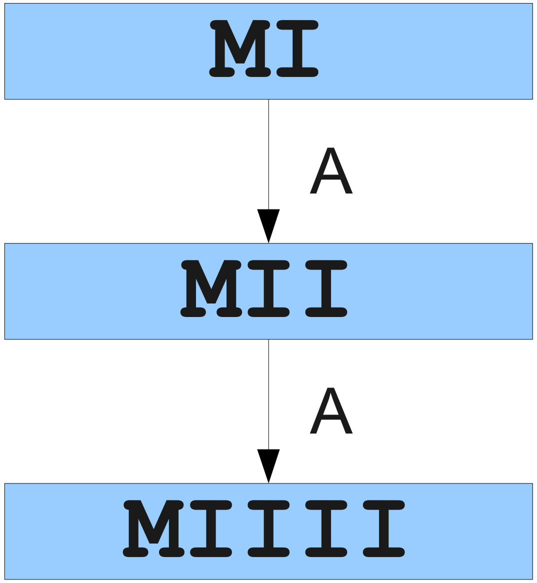
D) Append **U** if the string ends in **I**.

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`**.

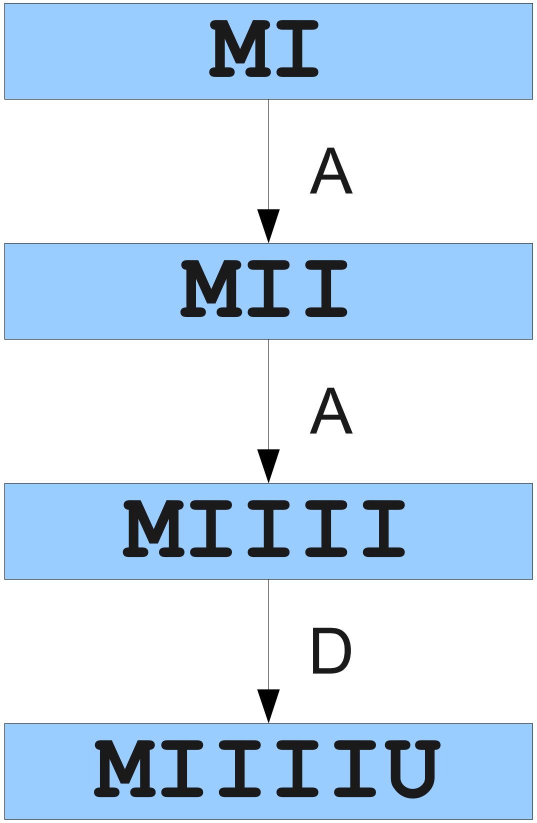


A) Double the contents of the string after **M**.

B) Replace **III** with **U**.

C) Remove **UU**

D) Append **U** if the string ends in **I**.

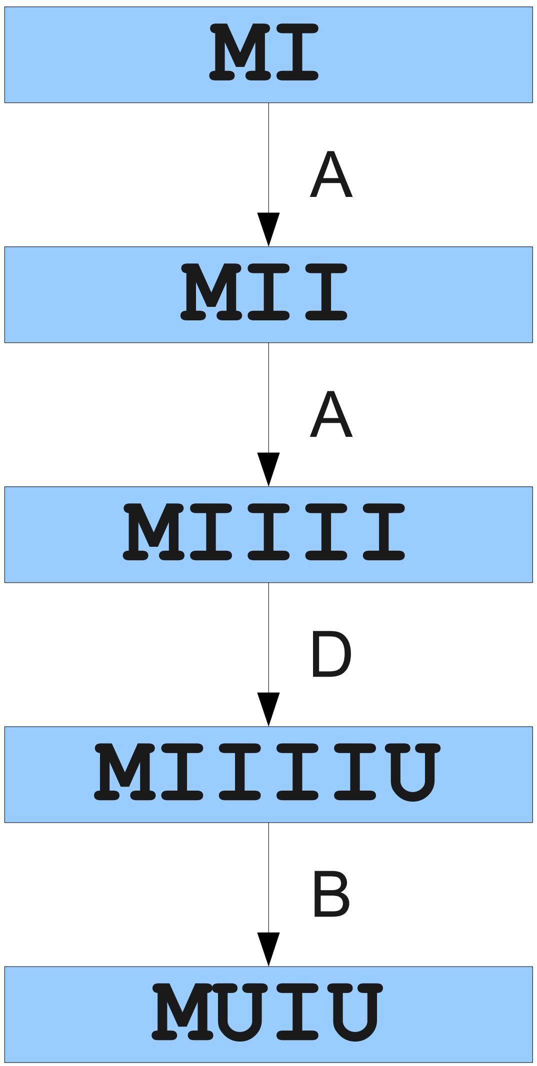


A) Double the contents of the string after **M**.

B) Replace **III** with **U**.

C) Remove **UU**

D) Append **U** if the string ends in **I**.

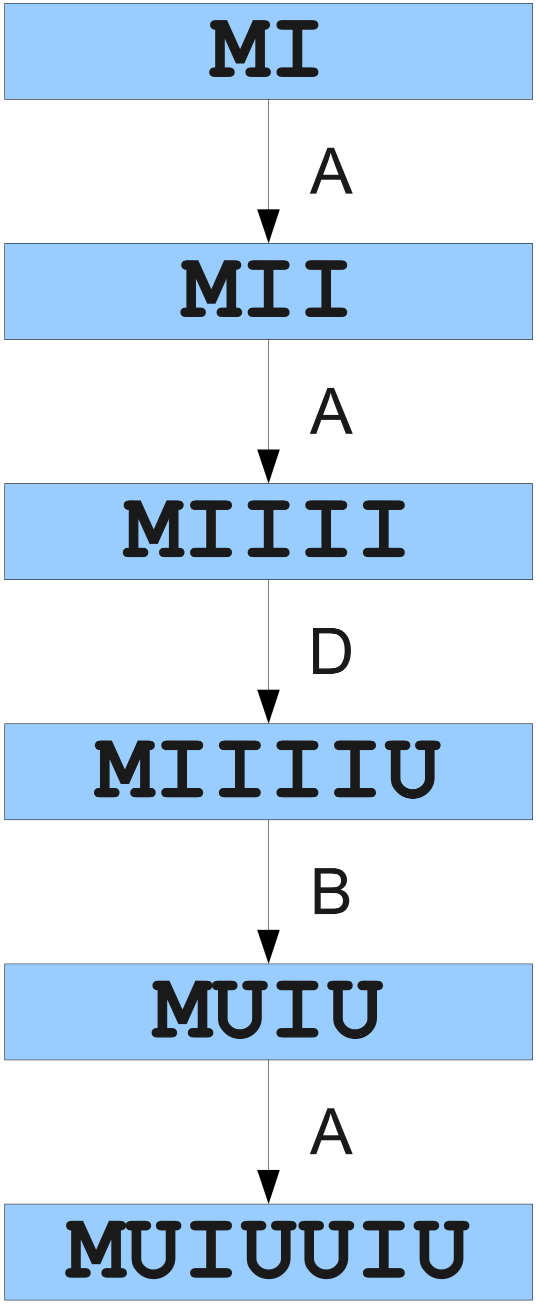


A) Double the contents of the string after **M**.

B) Replace **III** with **U**.

C) Remove **UU**

D) Append **U** if the string ends in **I**.

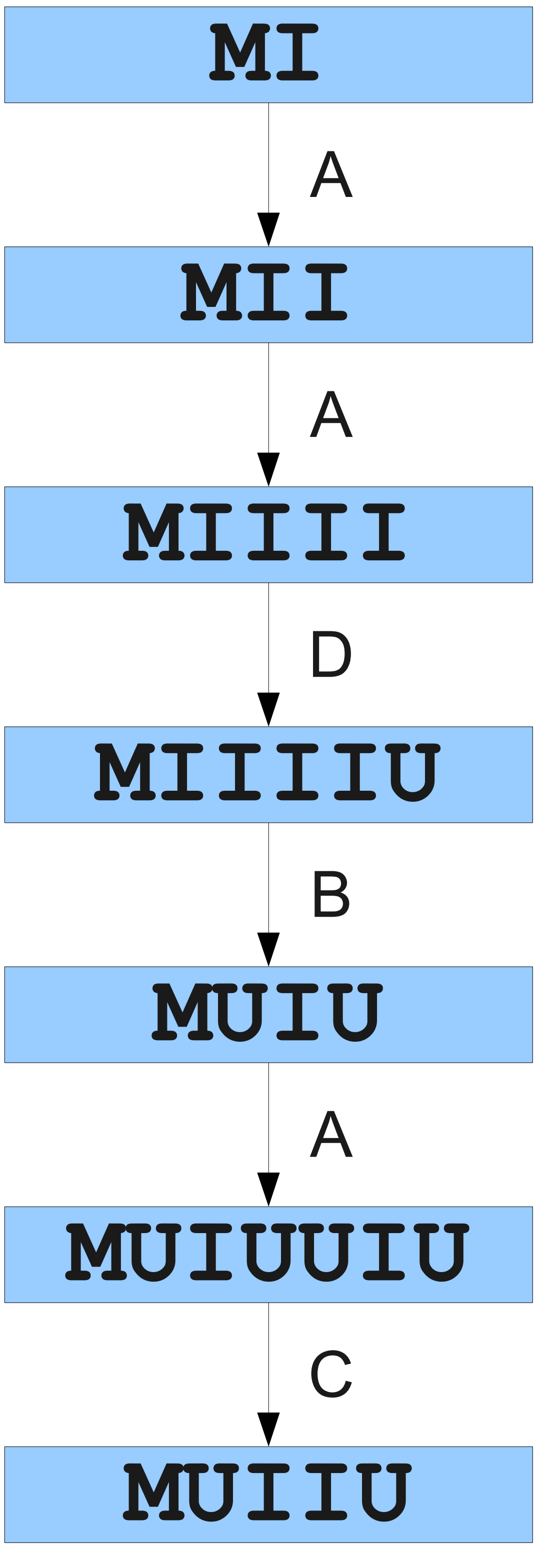


A) Double the contents of the string after **M**.

B) Replace **III** with **U**.

C) Remove **UU**

D) Append **U** if the string ends in **I**.



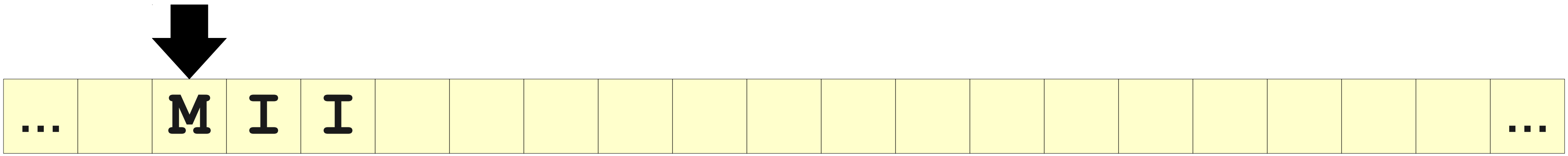
A Recognizable Language

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

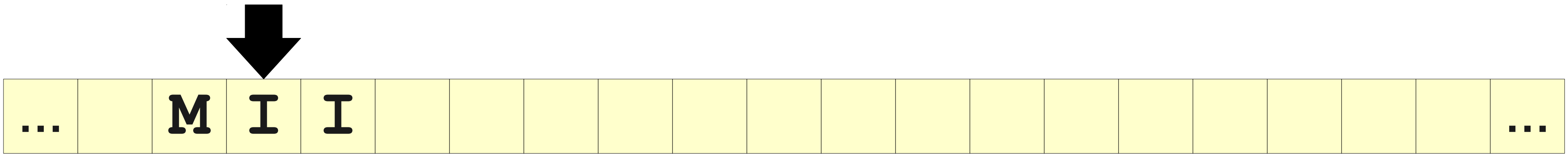
A Sketch of the Turing Machine



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.

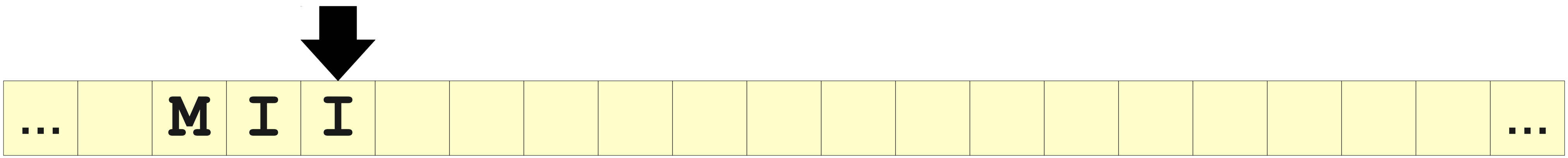
A Sketch of the Turing Machine



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.

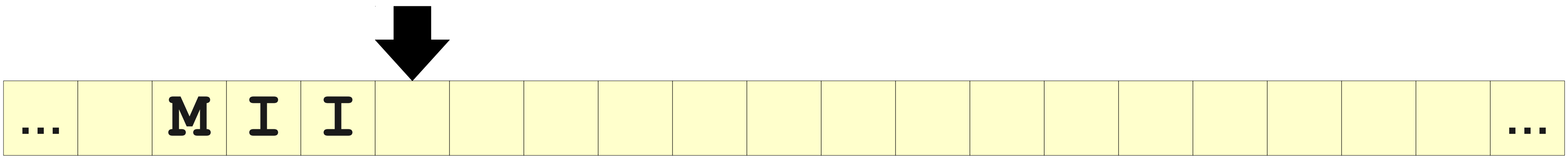
A Sketch of the Turing Machine



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.

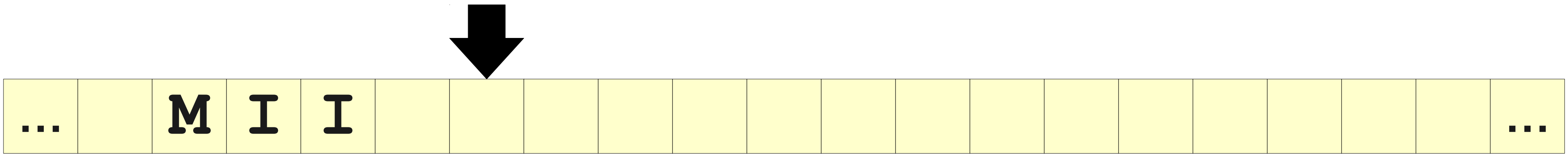
A Sketch of the Turing Machine



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.

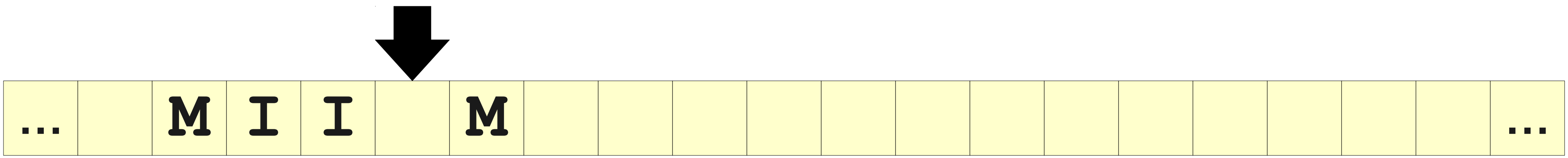
A Sketch of the Turing Machine



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.

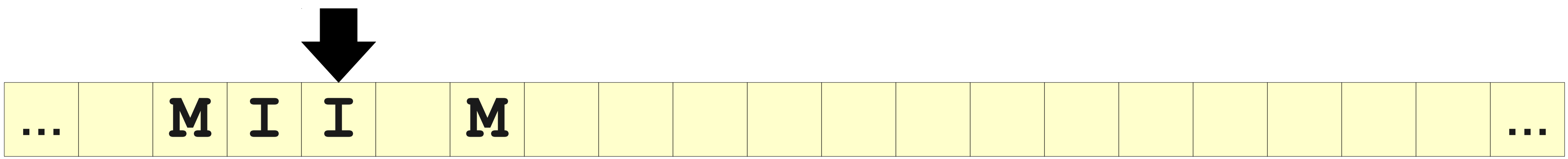
A Sketch of the Turing Machine



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.

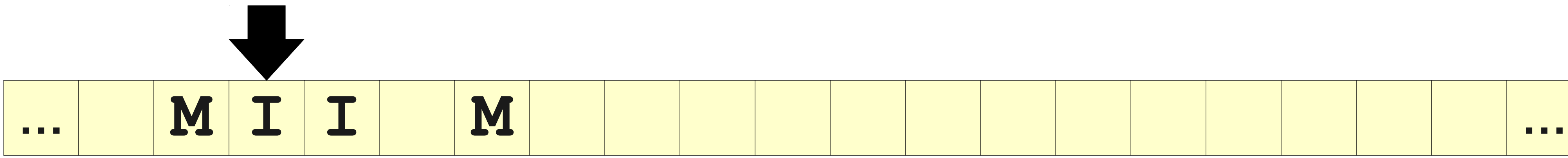
A Sketch of the Turing Machine



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.

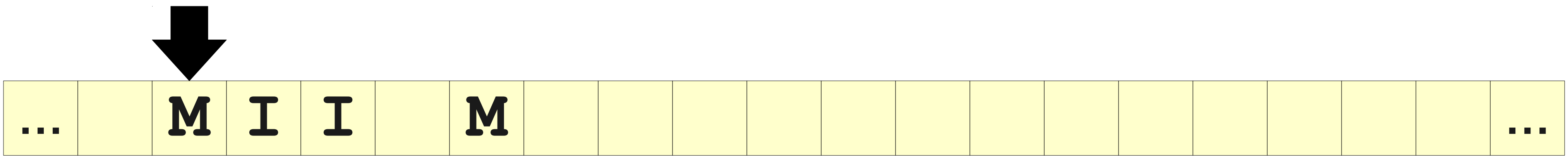
A Sketch of the Turing Machine



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.

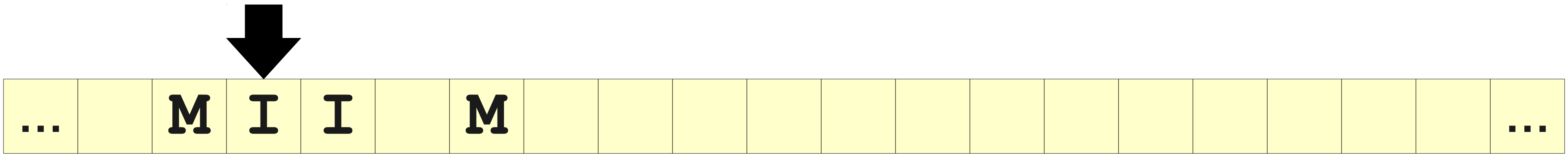
A Sketch of the Turing Machine



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.

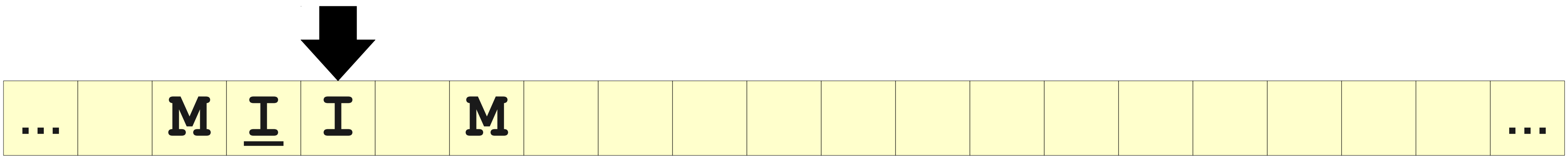
A Sketch of the Turing Machine



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.

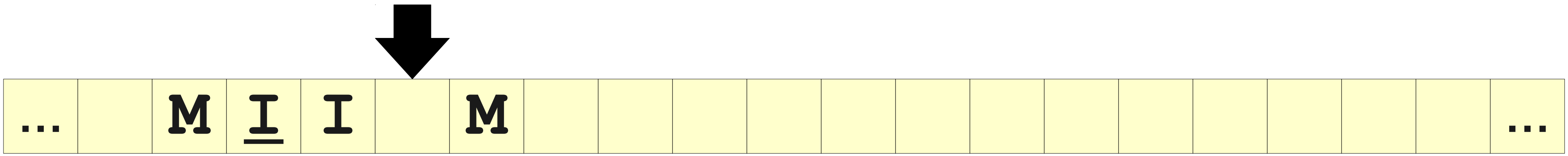
A Sketch of the Turing Machine



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.

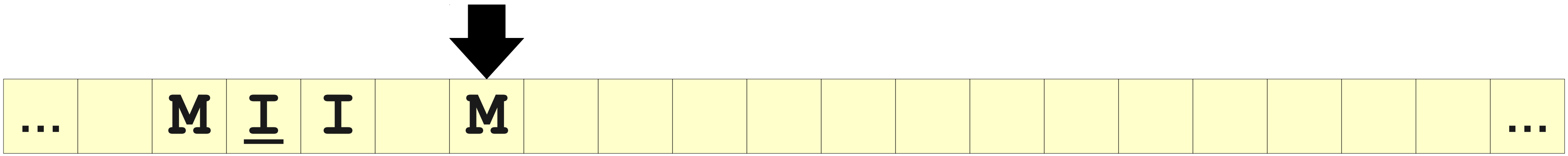
A Sketch of the Turing Machine



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.

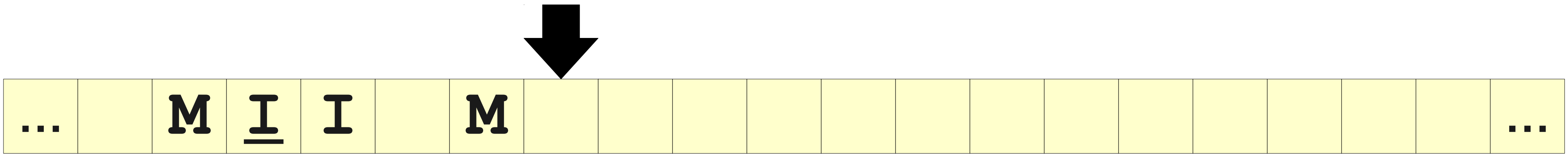
A Sketch of the Turing Machine



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.

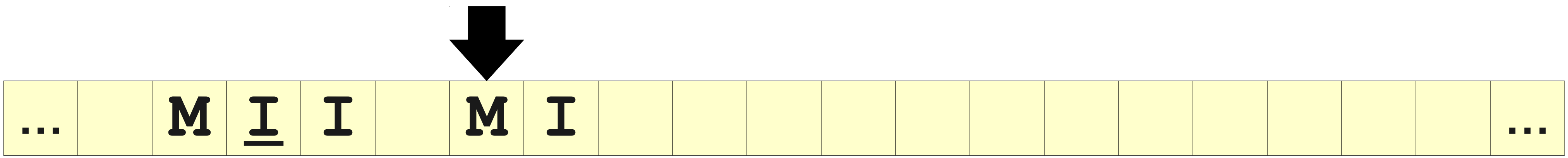
A Sketch of the Turing Machine



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.

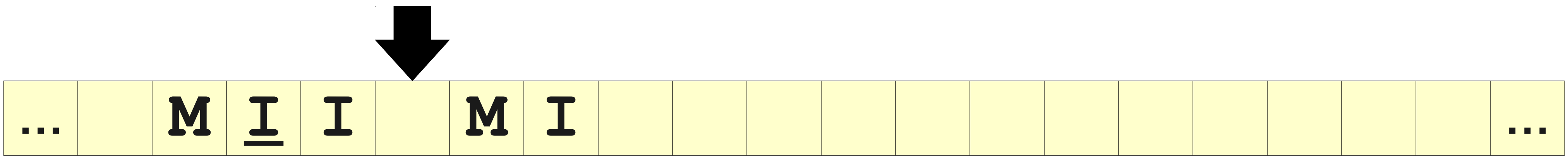
A Sketch of the Turing Machine



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.

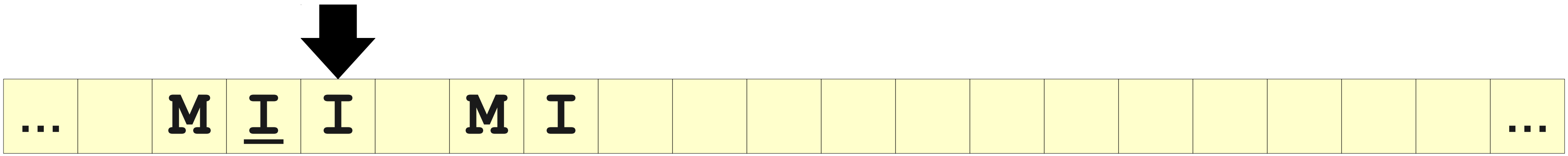
A Sketch of the Turing Machine



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.

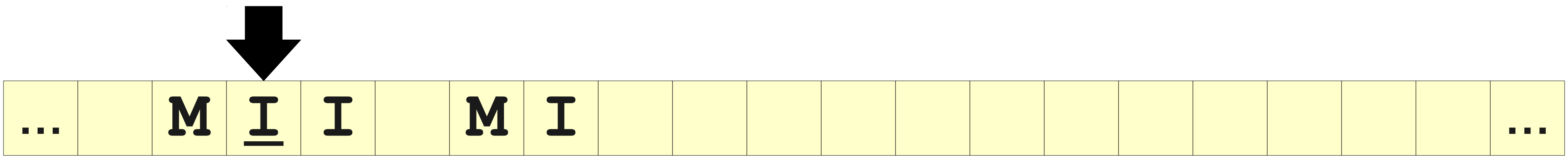
A Sketch of the Turing Machine



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.

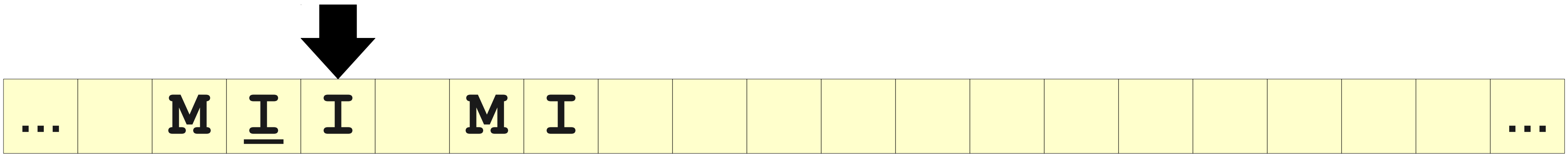
A Sketch of the Turing Machine



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.

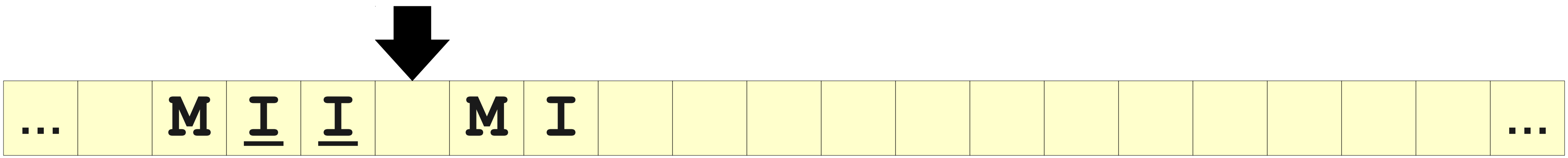
A Sketch of the Turing Machine



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.

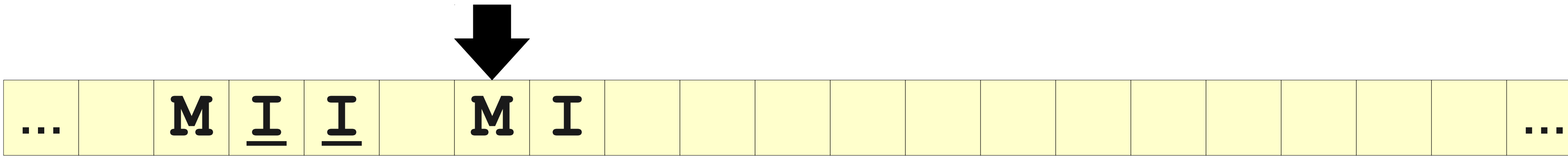
A Sketch of the Turing Machine



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.

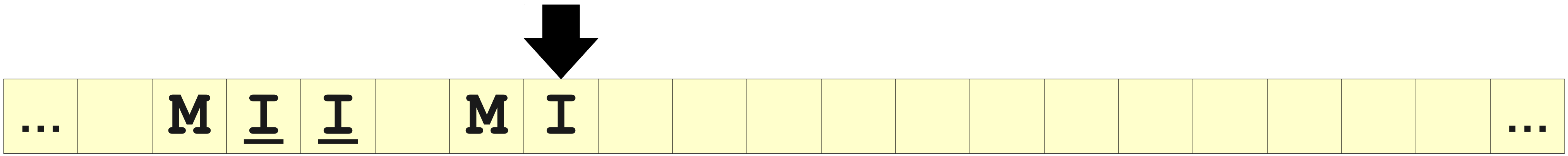
A Sketch of the Turing Machine



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.

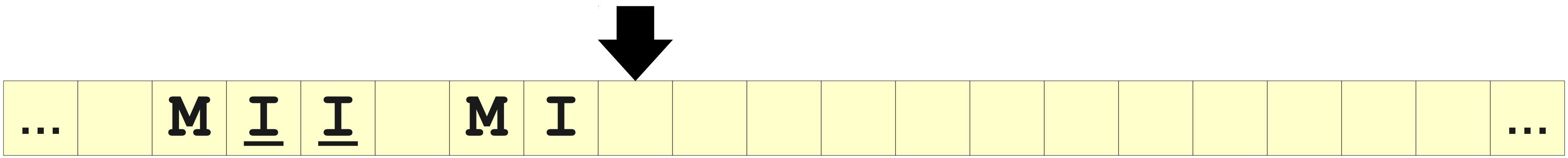
A Sketch of the Turing Machine



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.

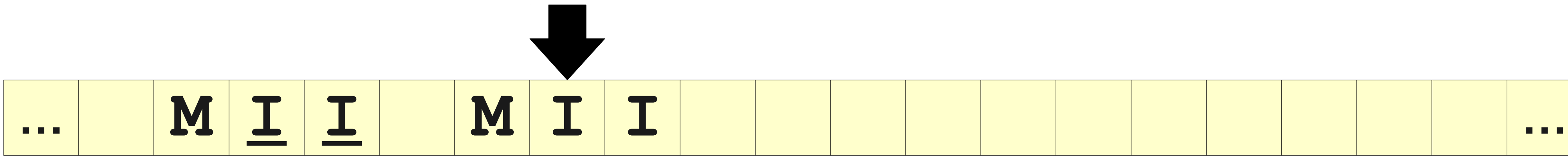
A Sketch of the Turing Machine



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.

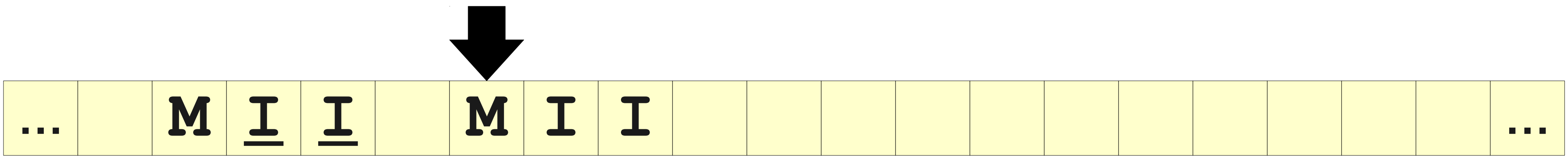
A Sketch of the Turing Machine



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.

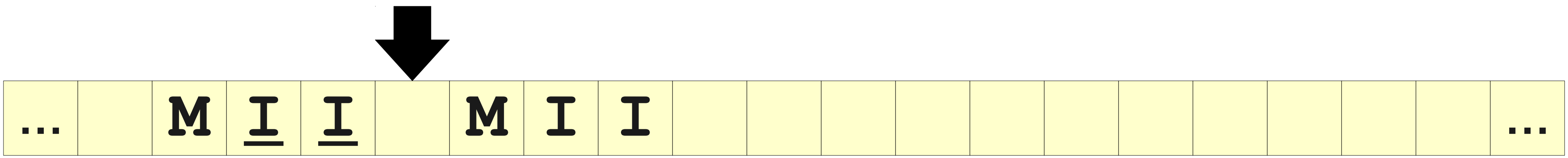
A Sketch of the Turing Machine



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.

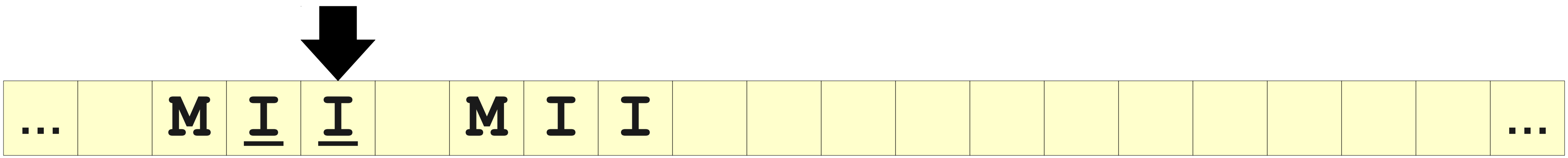
A Sketch of the Turing Machine



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.

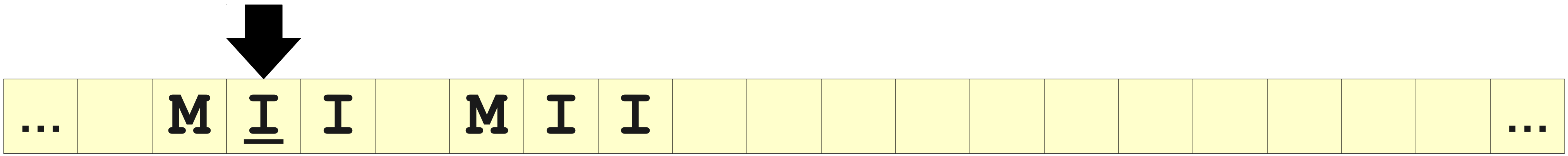
A Sketch of the Turing Machine



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.

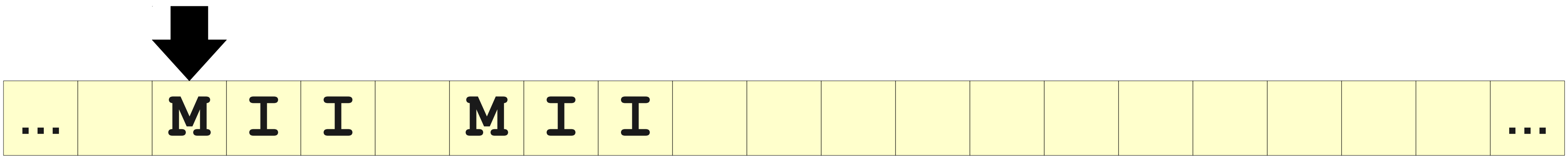
A Sketch of the Turing Machine



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.

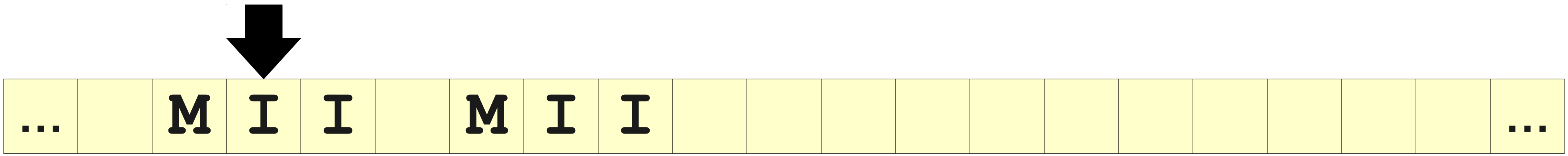
A Sketch of the Turing Machine



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.

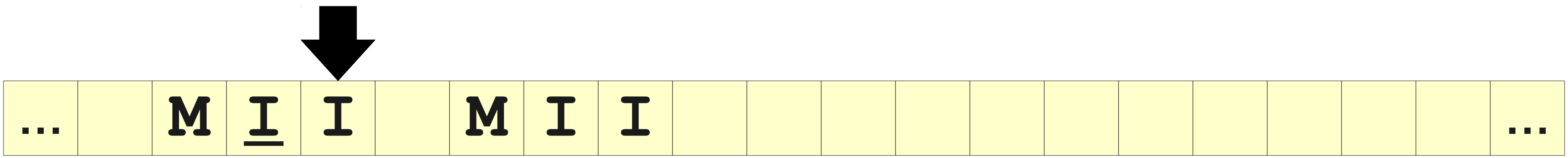
A Sketch of the Turing Machine



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.

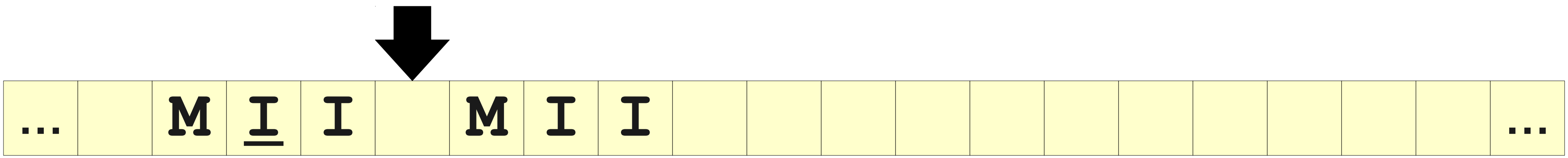
A Sketch of the Turing Machine



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.

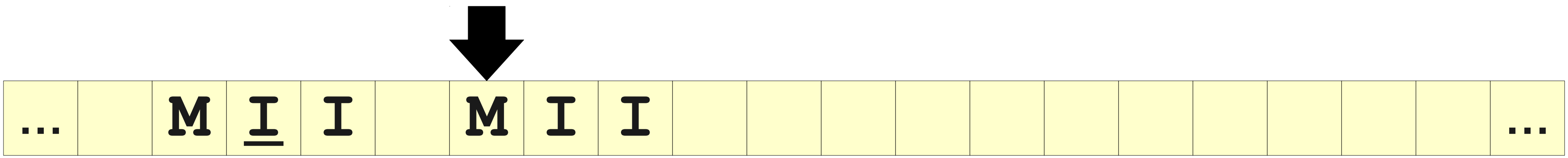
A Sketch of the Turing Machine



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.

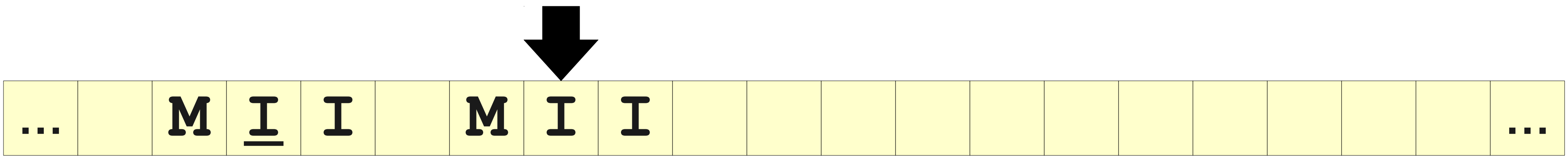
A Sketch of the Turing Machine



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.

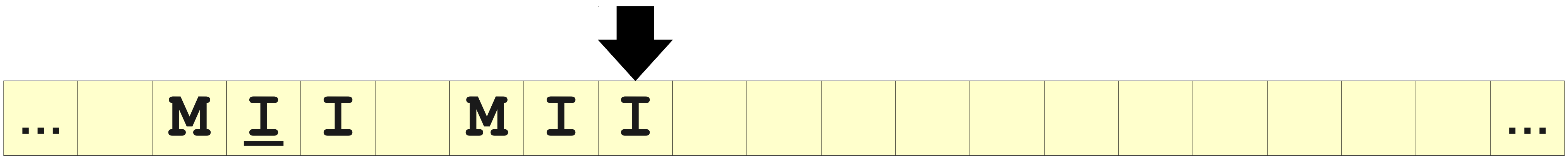
A Sketch of the Turing Machine



To recognize L, our TM M does the following:

- While M has not found the string **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.

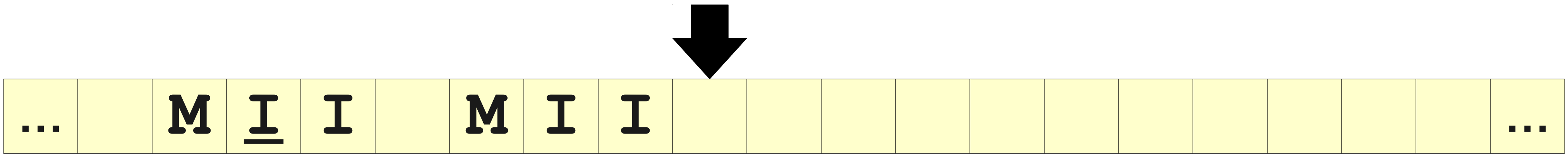
A Sketch of the Turing Machine



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.

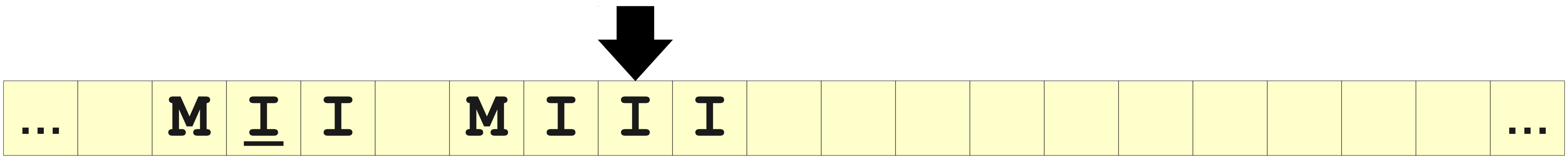
A Sketch of the Turing Machine



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.

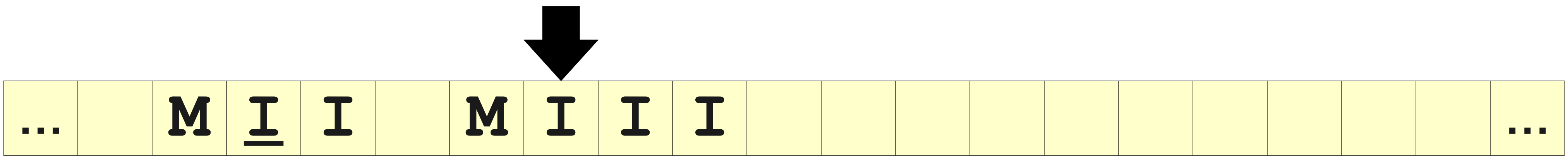
A Sketch of the Turing Machine



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.

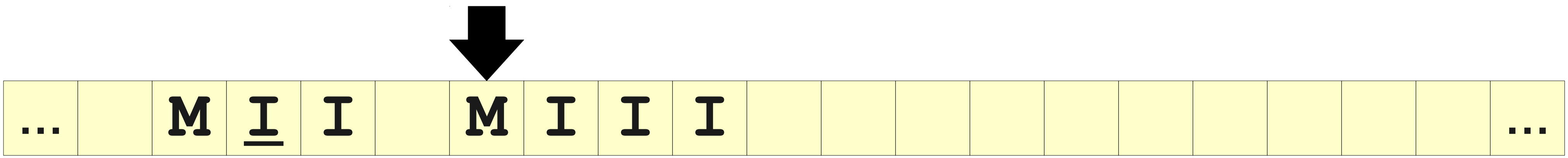
A Sketch of the Turing Machine



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.

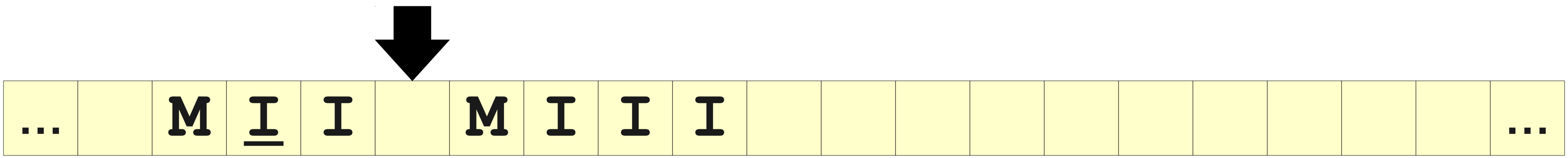
A Sketch of the Turing Machine



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.

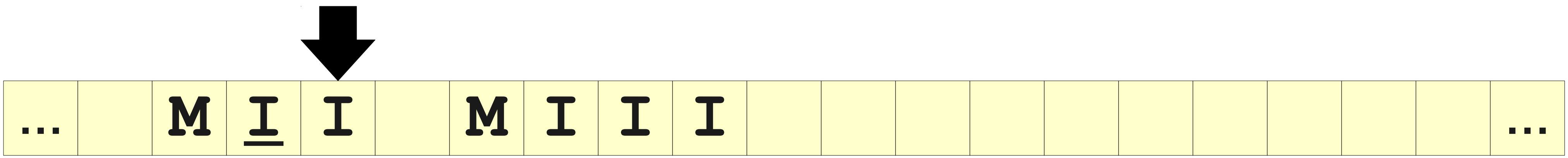
A Sketch of the Turing Machine



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.

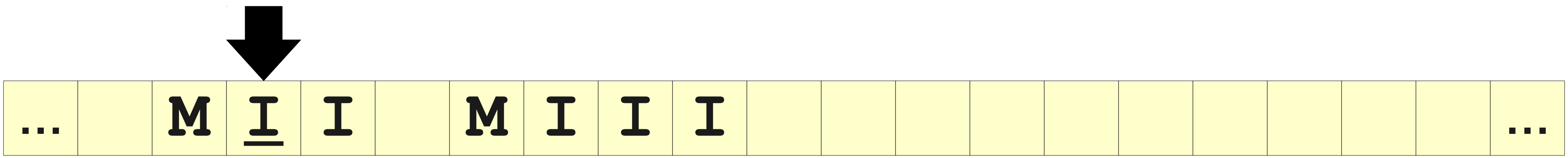
A Sketch of the Turing Machine



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.

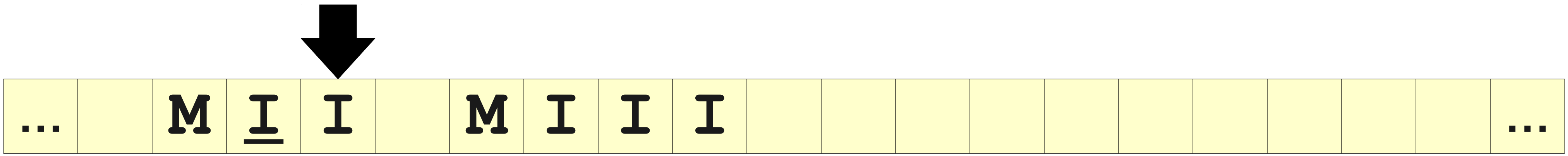
A Sketch of the Turing Machine



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.

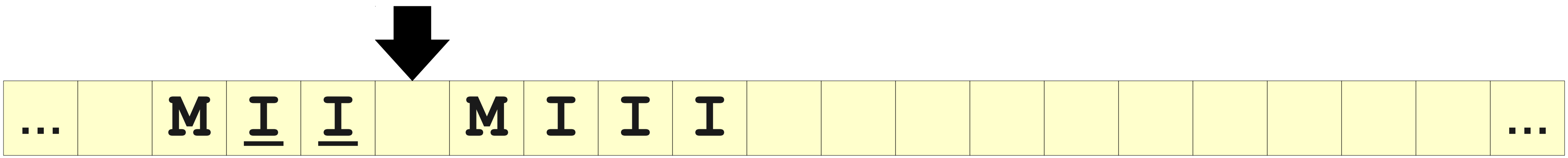
A Sketch of the Turing Machine



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.

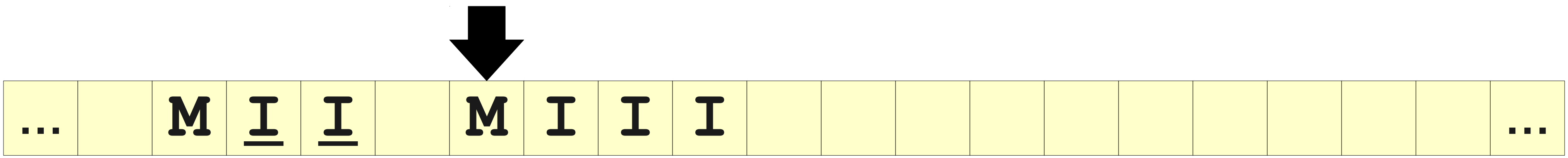
A Sketch of the Turing Machine



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.

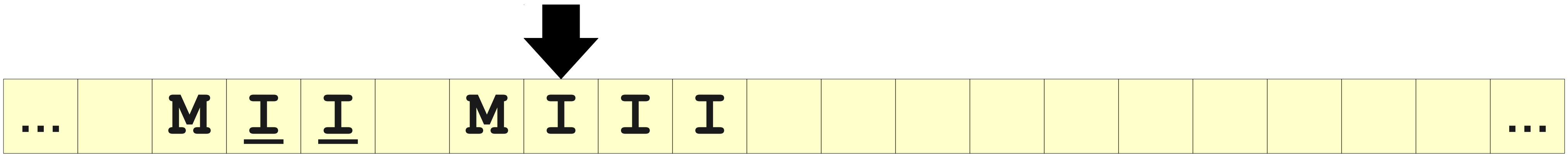
A Sketch of the Turing Machine



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.

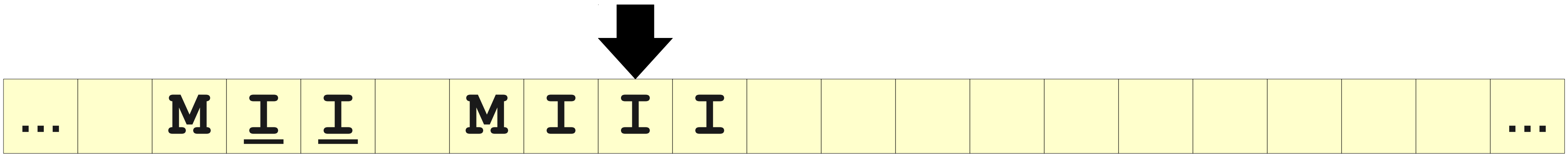
A Sketch of the Turing Machine



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.

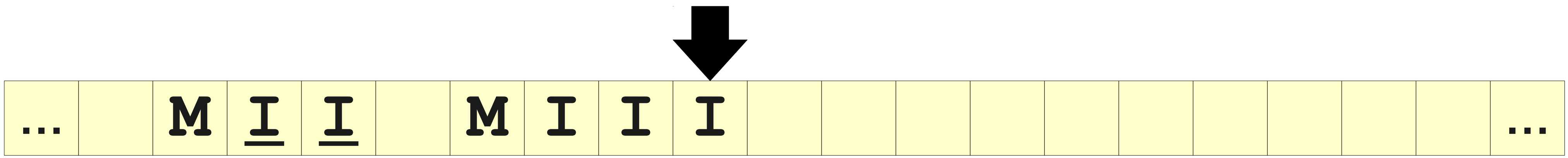
A Sketch of the Turing Machine



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.

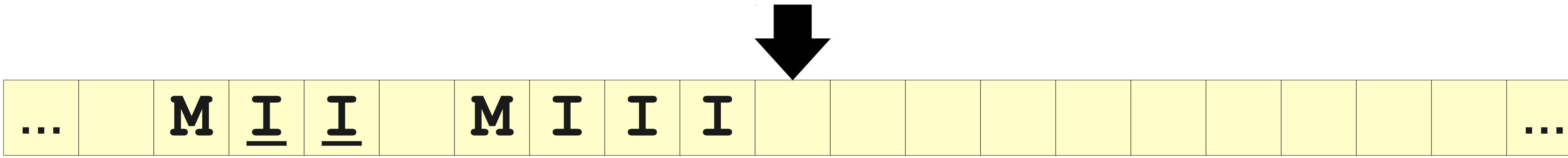
A Sketch of the Turing Machine



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.

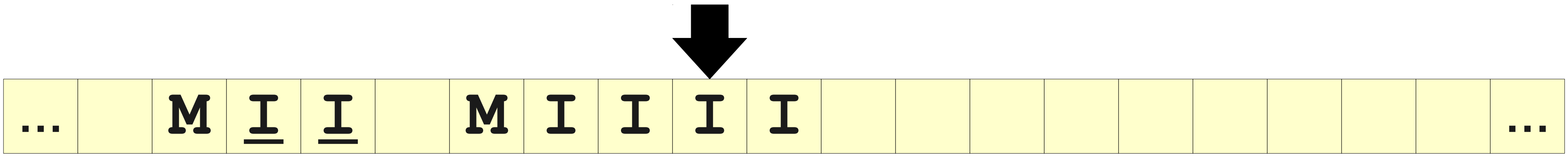
A Sketch of the Turing Machine



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.

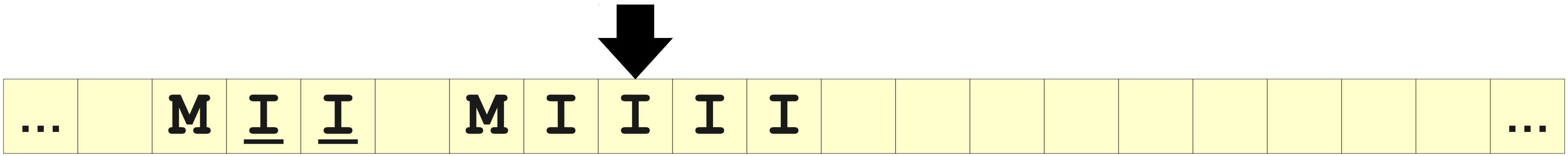
A Sketch of the Turing Machine



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.

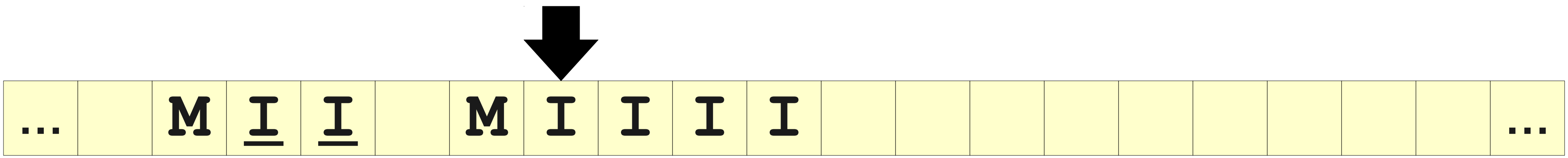
A Sketch of the Turing Machine



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.

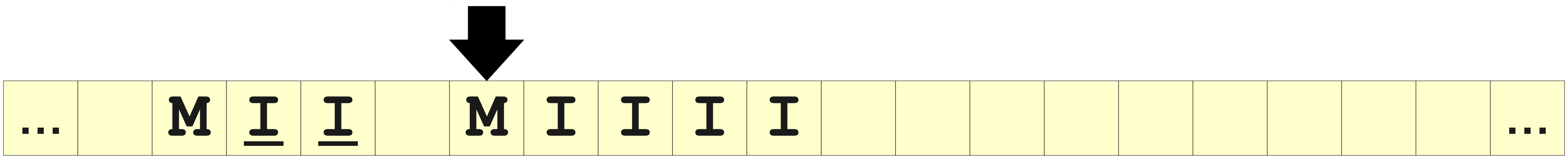
A Sketch of the Turing Machine



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.

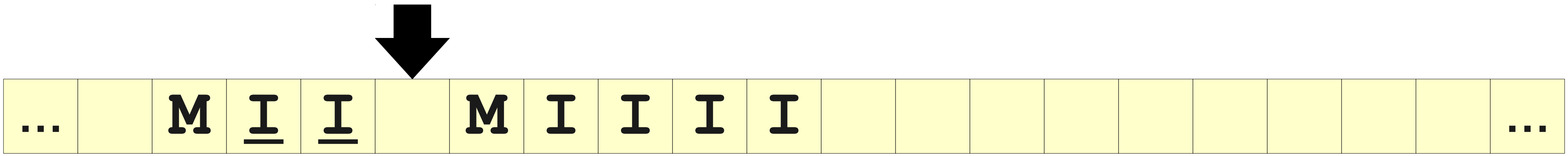
A Sketch of the Turing Machine



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.

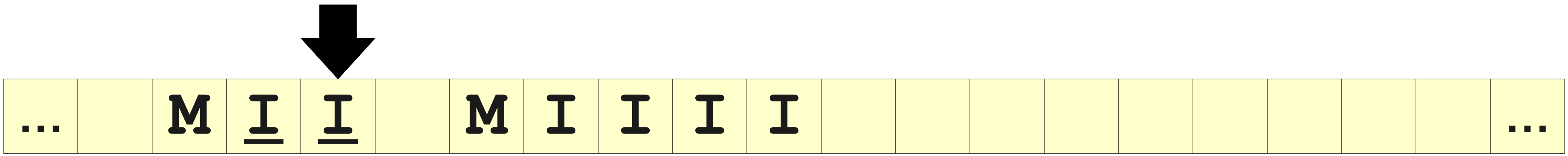
A Sketch of the Turing Machine



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.

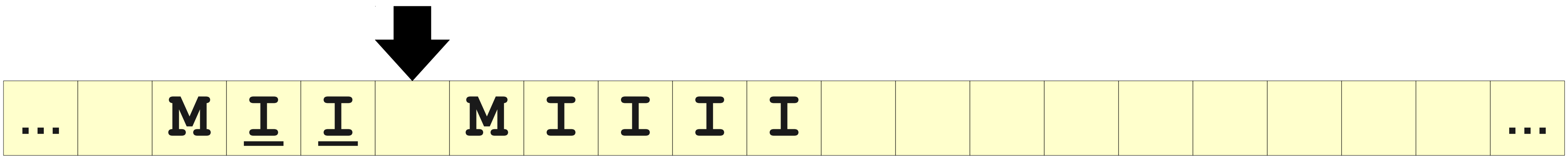
A Sketch of the Turing Machine



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.

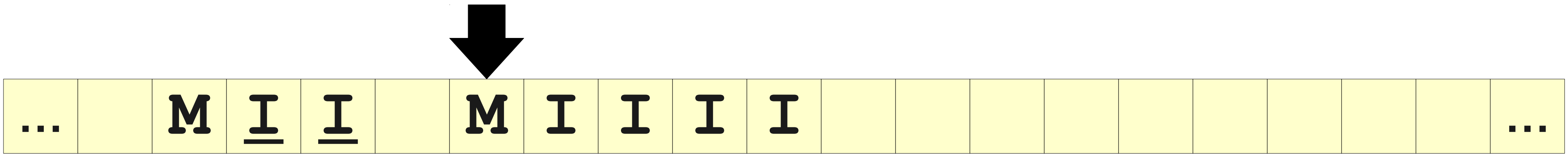
A Sketch of the Turing Machine



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.

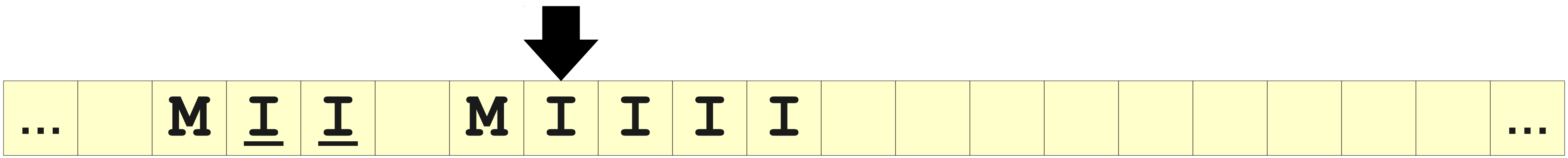
A Sketch of the Turing Machine



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.

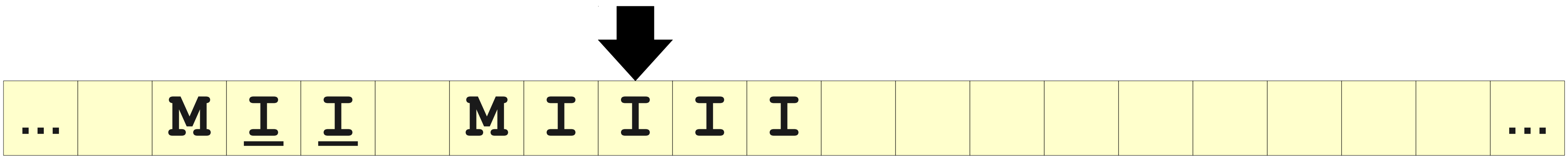
A Sketch of the Turing Machine



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.

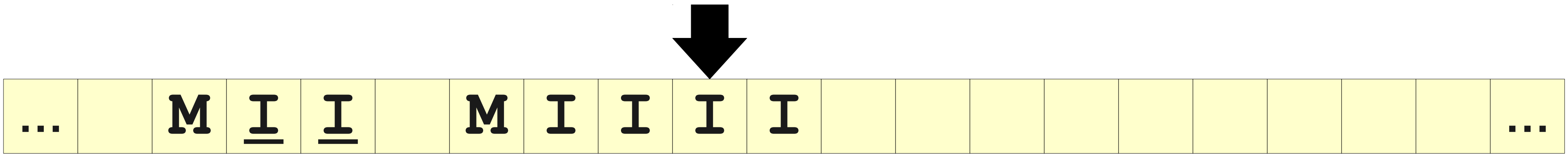
A Sketch of the Turing Machine



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.

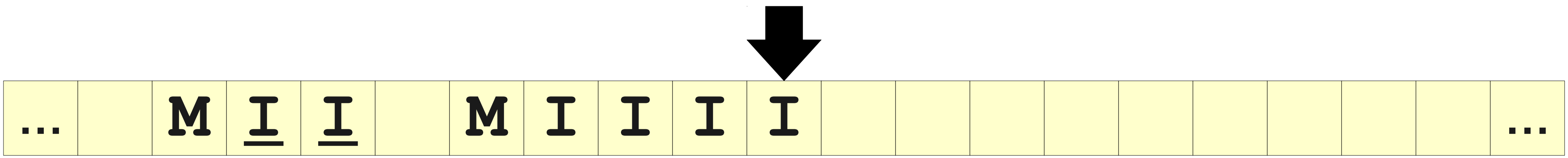
A Sketch of the Turing Machine



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.

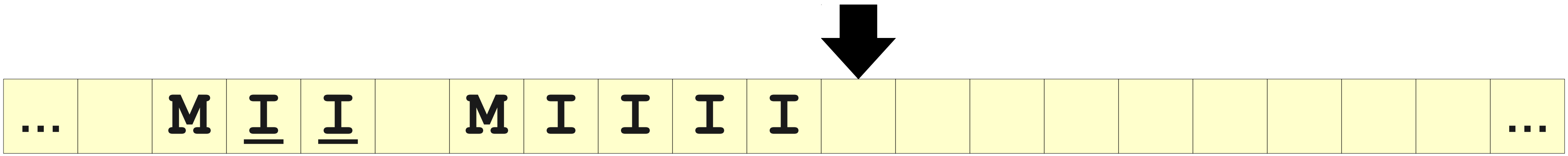
A Sketch of the Turing Machine



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.

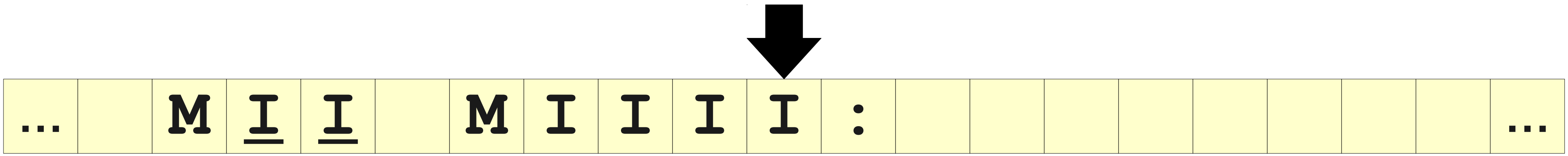
A Sketch of the Turing Machine



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.

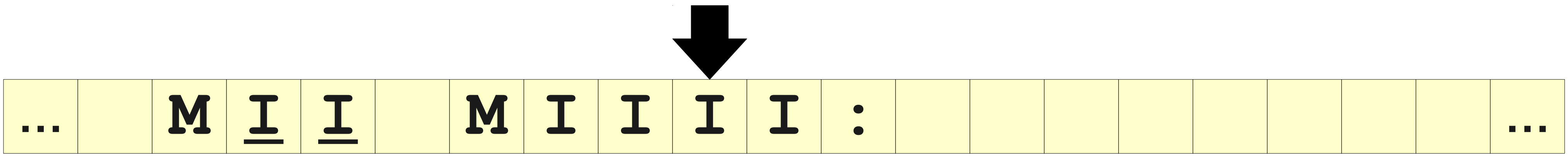
A Sketch of the Turing Machine



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.

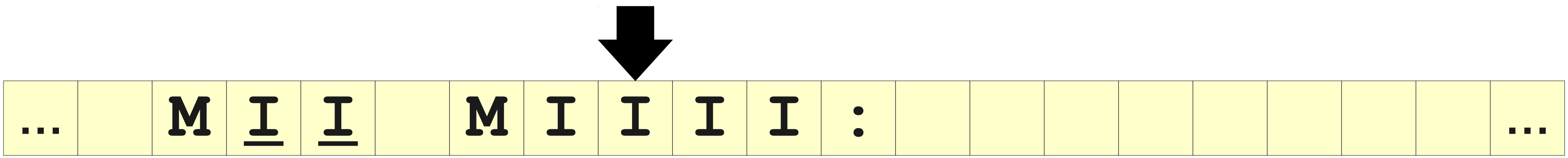
A Sketch of the Turing Machine



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.

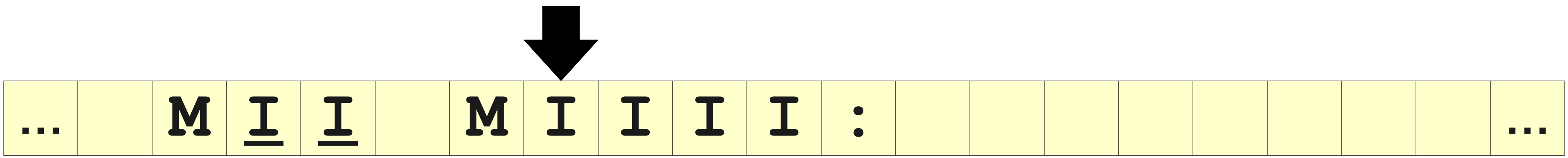
A Sketch of the Turing Machine



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.

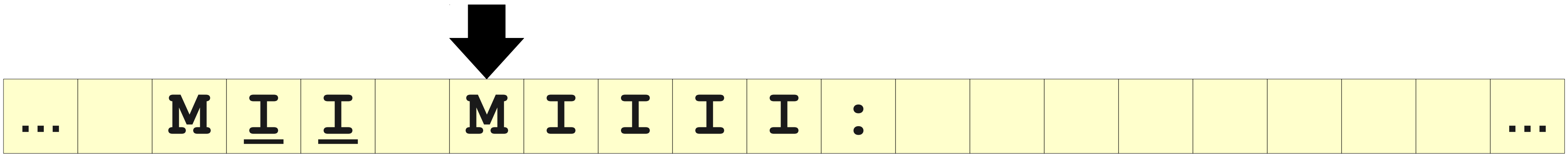
A Sketch of the Turing Machine



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.

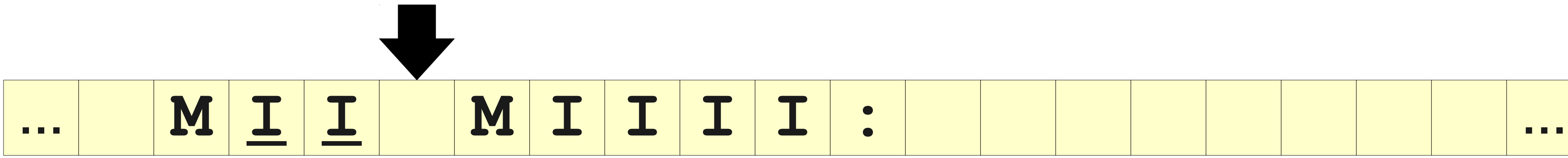
A Sketch of the Turing Machine



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.

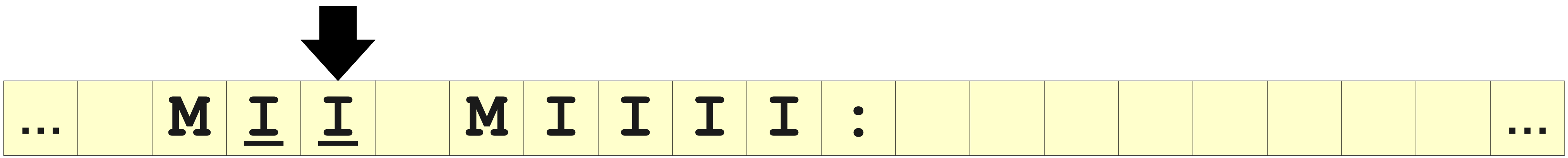
A Sketch of the Turing Machine



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.

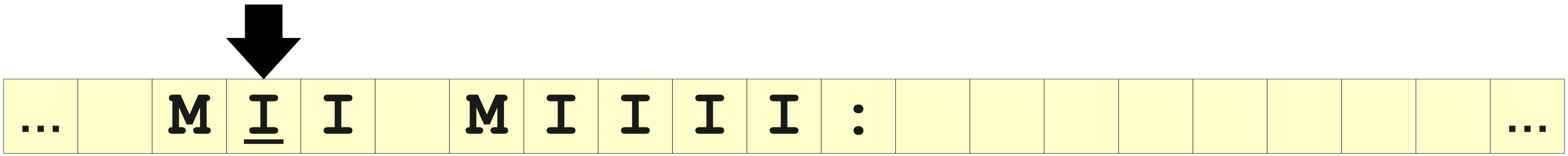
A Sketch of the Turing Machine



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.

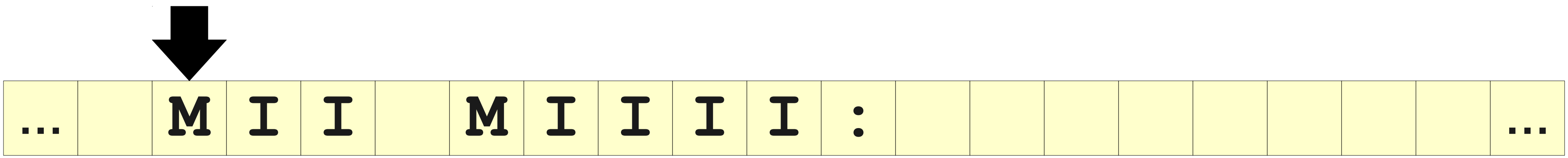
A Sketch of the Turing Machine



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.

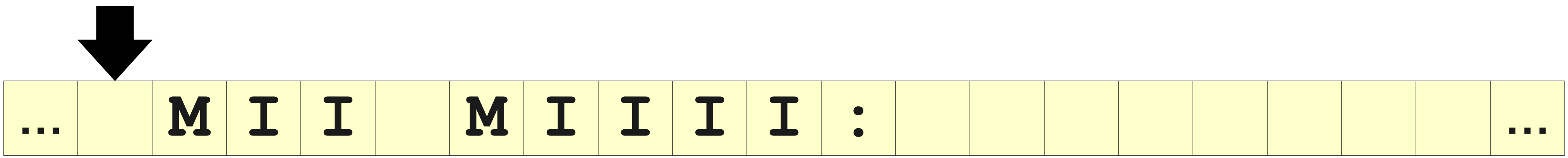
A Sketch of the Turing Machine



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.

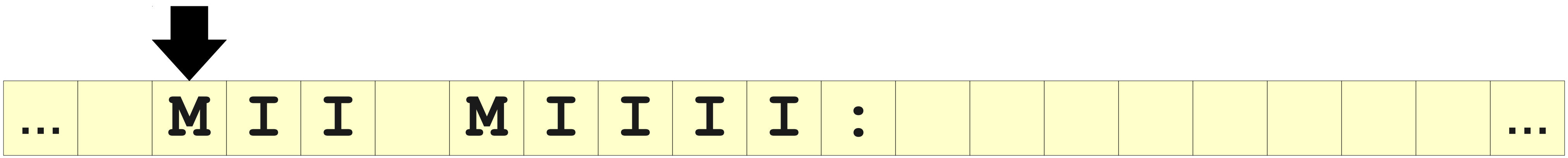
A Sketch of the Turing Machine



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.

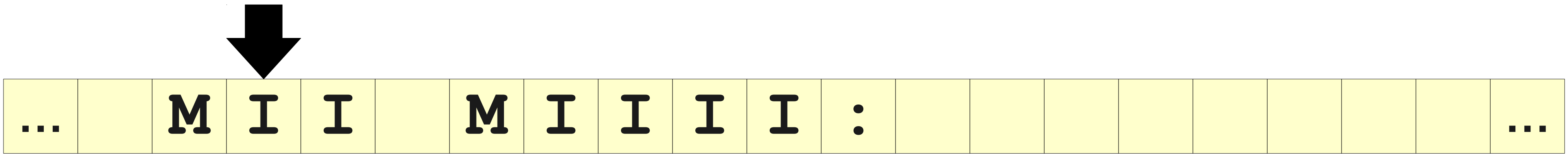
A Sketch of the Turing Machine



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.

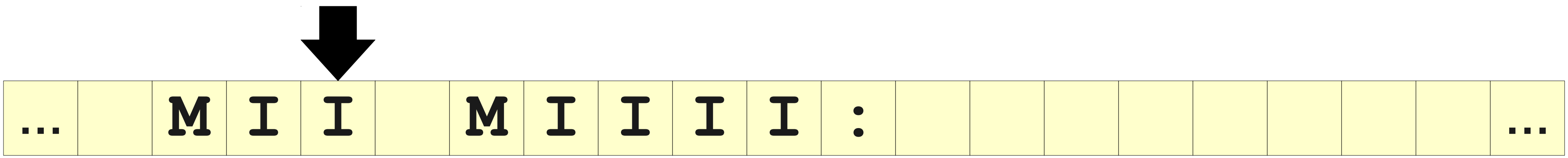
A Sketch of the Turing Machine



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.

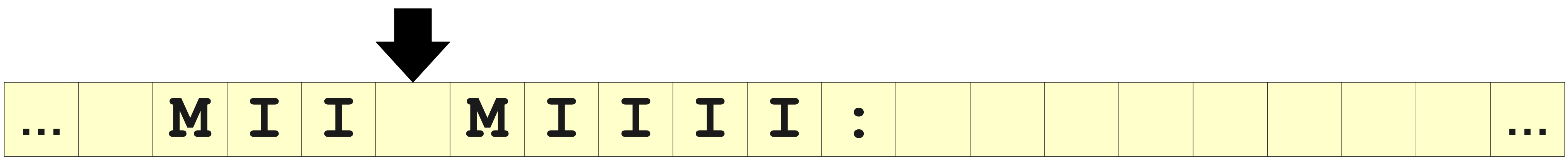
A Sketch of the Turing Machine



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.

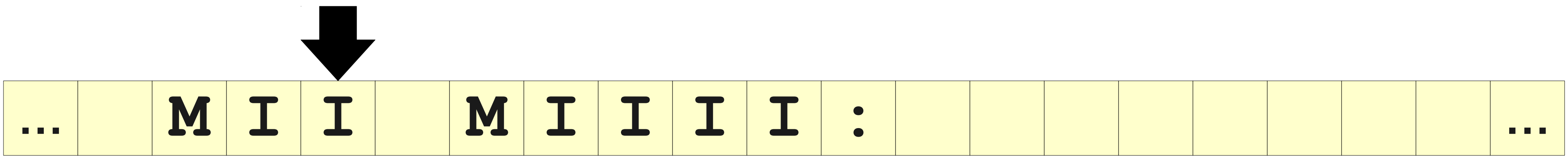
A Sketch of the Turing Machine



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.

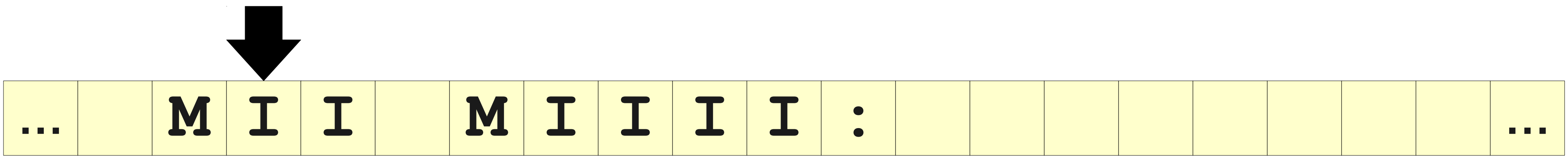
A Sketch of the Turing Machine



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.

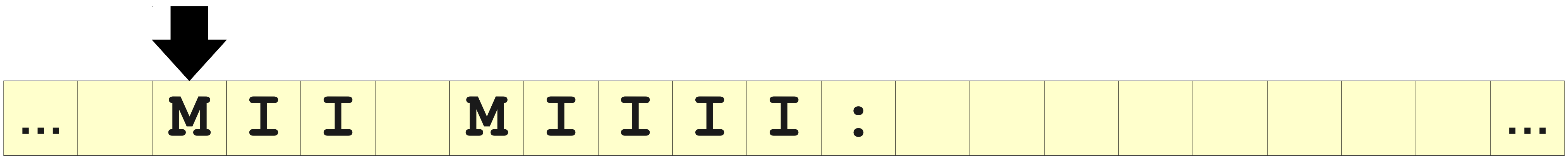
A Sketch of the Turing Machine



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.

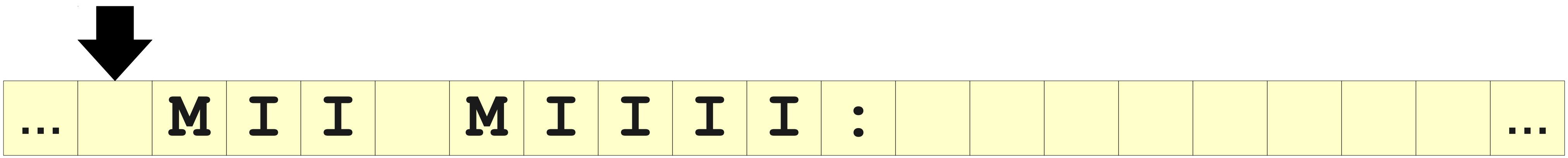
A Sketch of the Turing Machine



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.

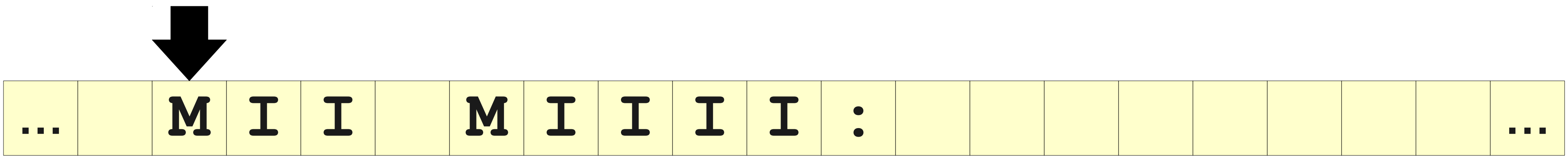
A Sketch of the Turing Machine



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.

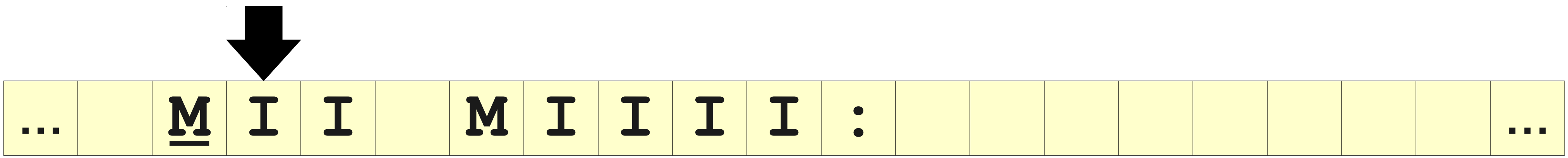
A Sketch of the Turing Machine



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.

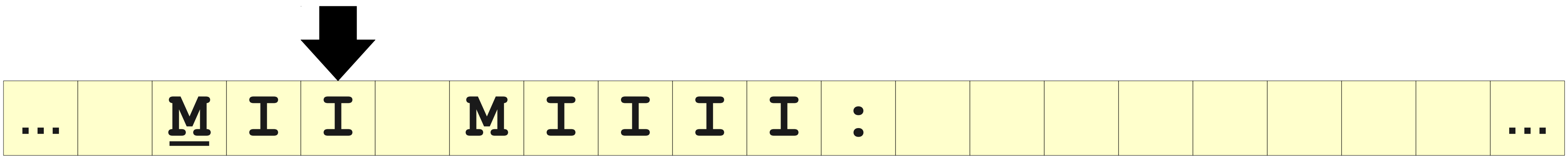
A Sketch of the Turing Machine



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.

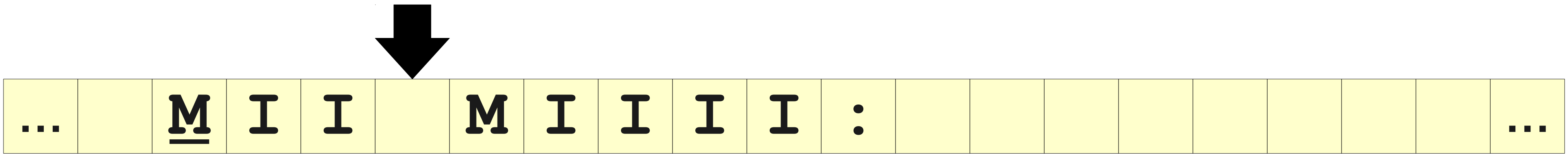
A Sketch of the Turing Machine



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.

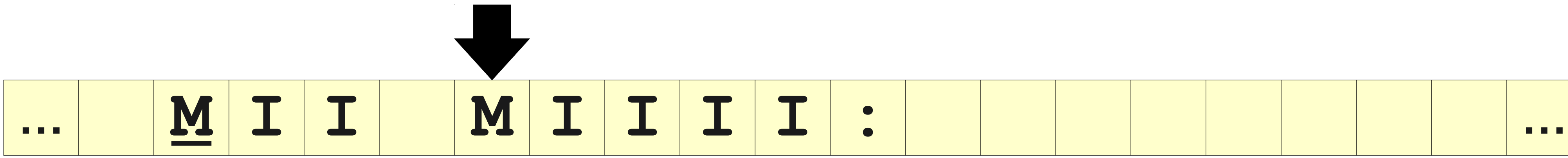
A Sketch of the Turing Machine



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.

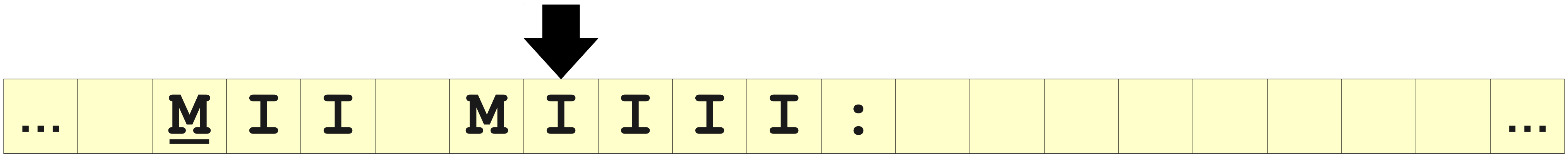
A Sketch of the Turing Machine



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.

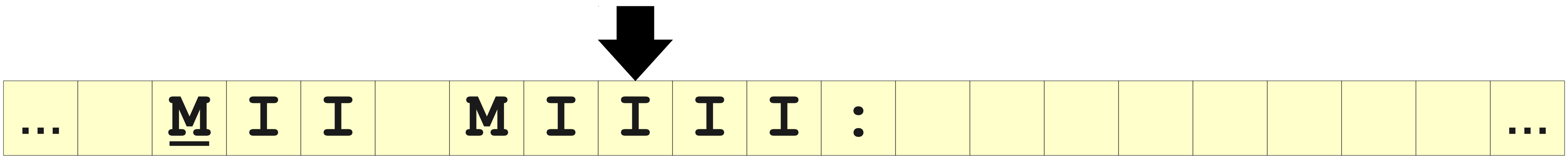
A Sketch of the Turing Machine



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.

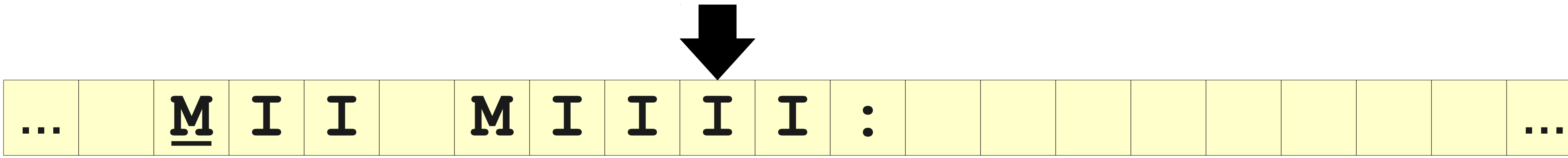
A Sketch of the Turing Machine



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.

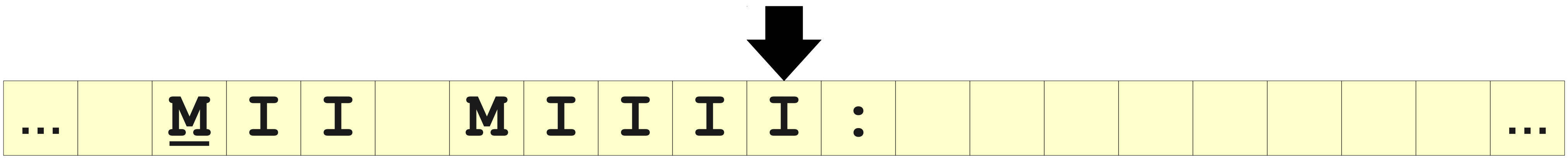
A Sketch of the Turing Machine



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.

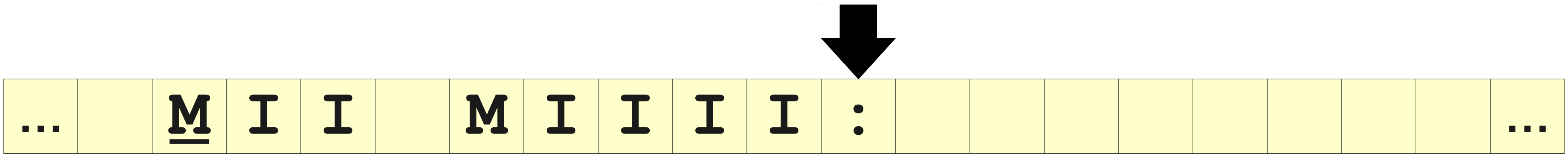
A Sketch of the Turing Machine



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.

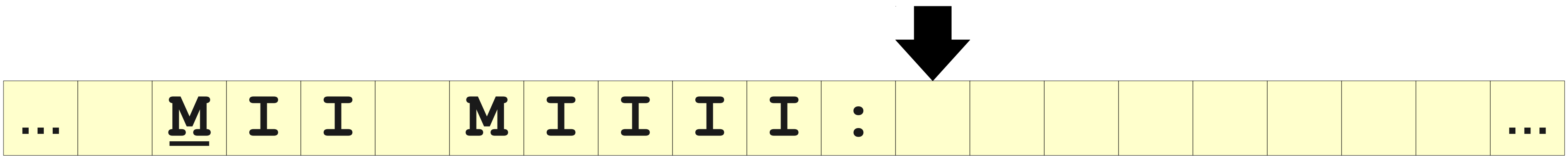
A Sketch of the Turing Machine



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.

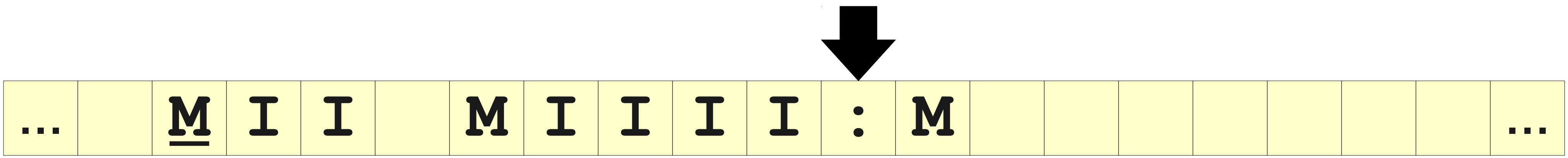
A Sketch of the Turing Machine



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.

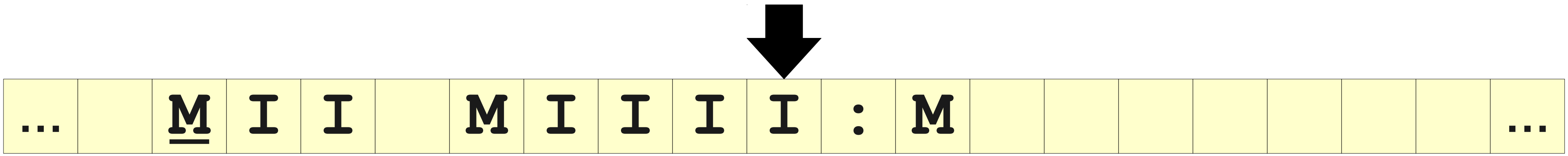
A Sketch of the Turing Machine



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.

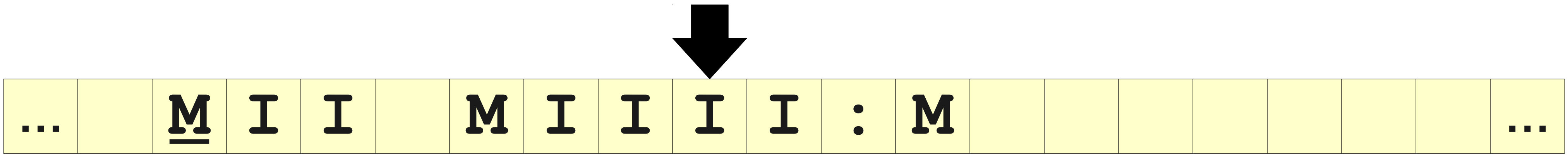
A Sketch of the Turing Machine



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.

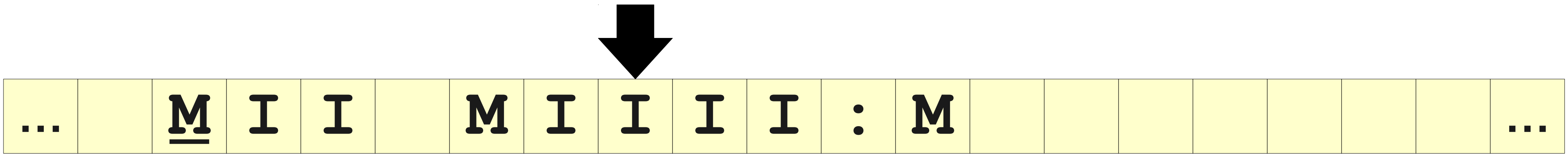
A Sketch of the Turing Machine



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.

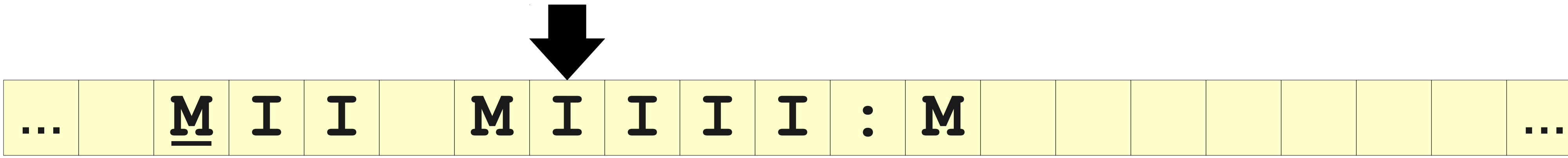
A Sketch of the Turing Machine



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.

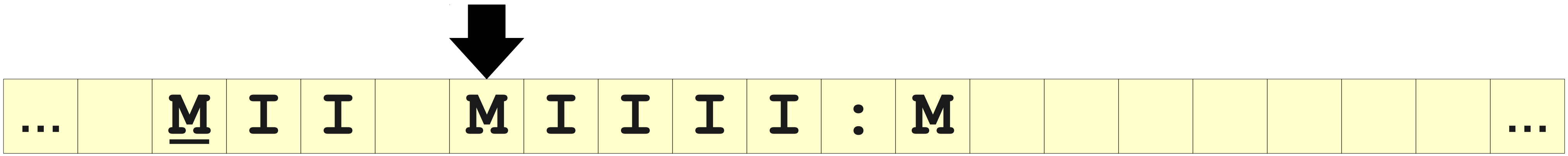
A Sketch of the Turing Machine



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.

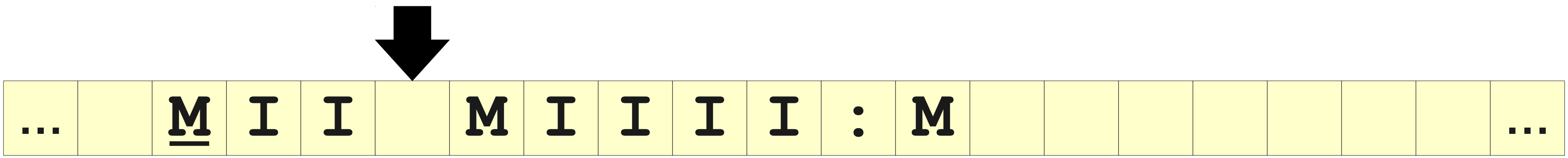
A Sketch of the Turing Machine



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.

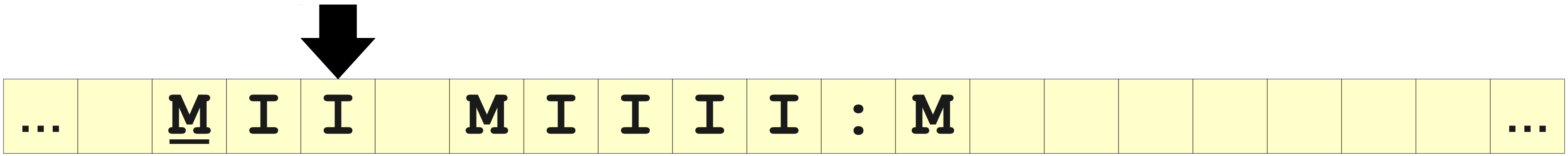
A Sketch of the Turing Machine



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.

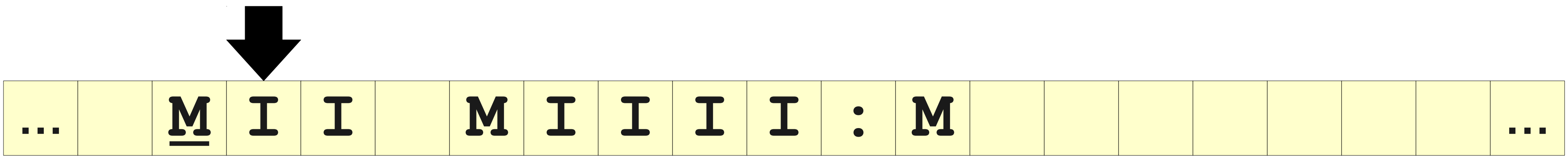
A Sketch of the Turing Machine



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.

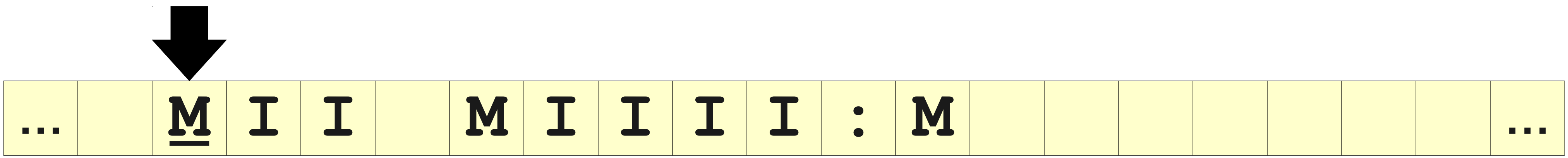
A Sketch of the Turing Machine



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.

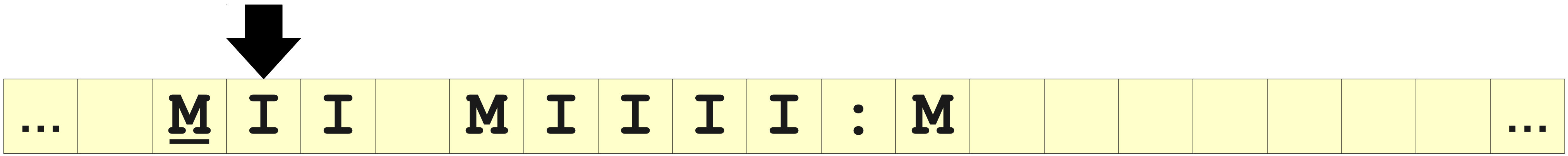
A Sketch of the Turing Machine



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.

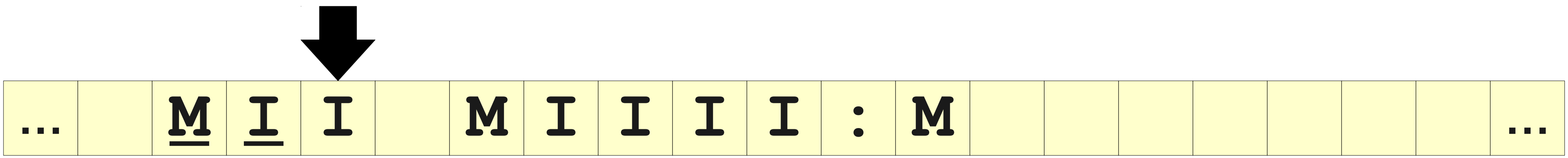
A Sketch of the Turing Machine



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.

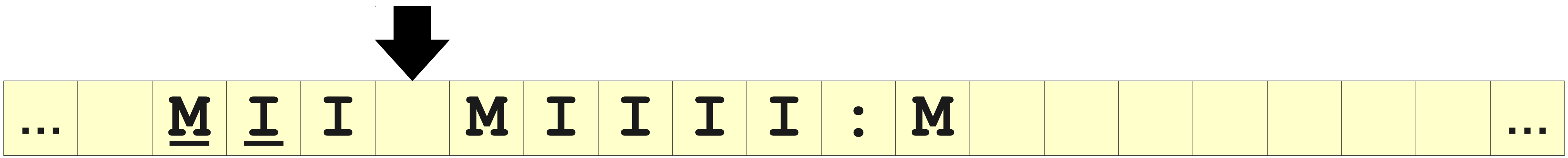
A Sketch of the Turing Machine



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.

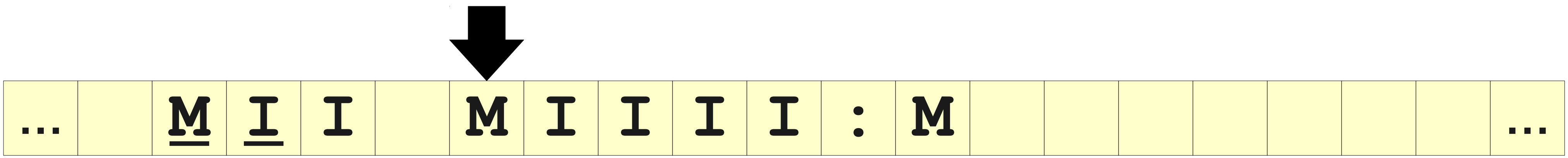
A Sketch of the Turing Machine



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.

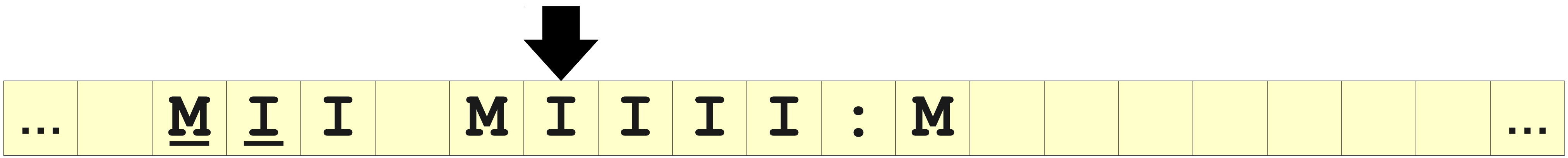
A Sketch of the Turing Machine



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.

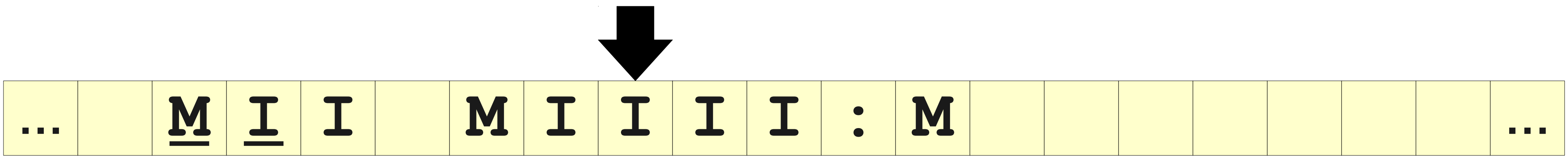
A Sketch of the Turing Machine



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.

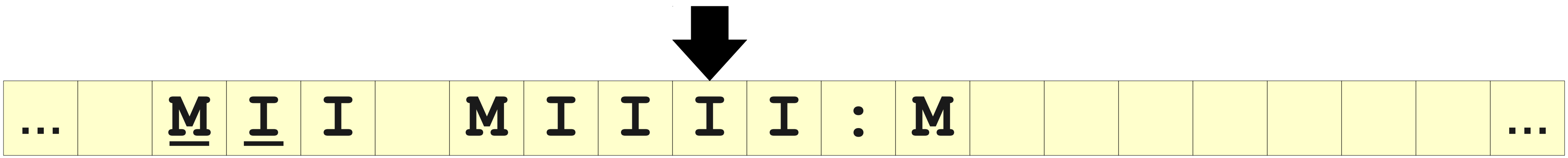
A Sketch of the Turing Machine



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.

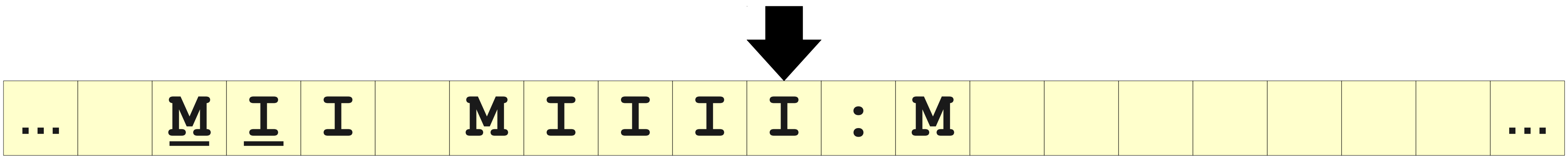
A Sketch of the Turing Machine



To recognize L, our TM M does the following:

- While M has not found the string **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.

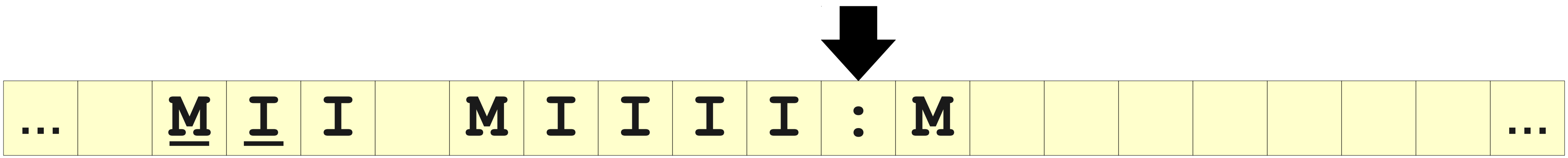
A Sketch of the Turing Machine



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.

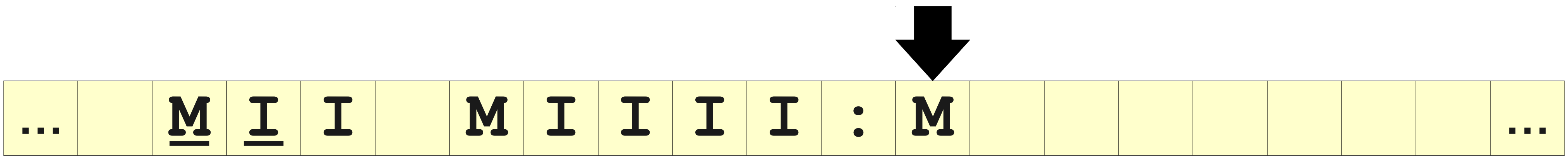
A Sketch of the Turing Machine



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.

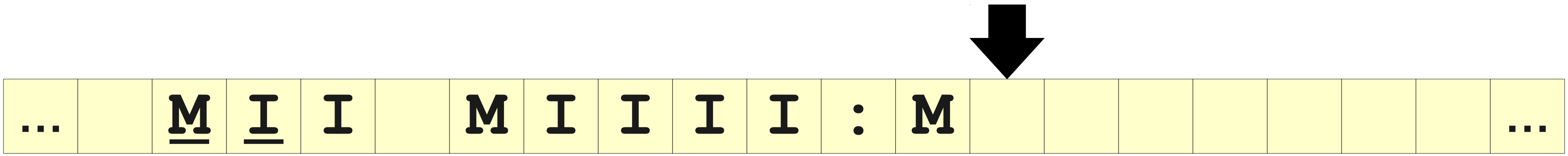
A Sketch of the Turing Machine



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.

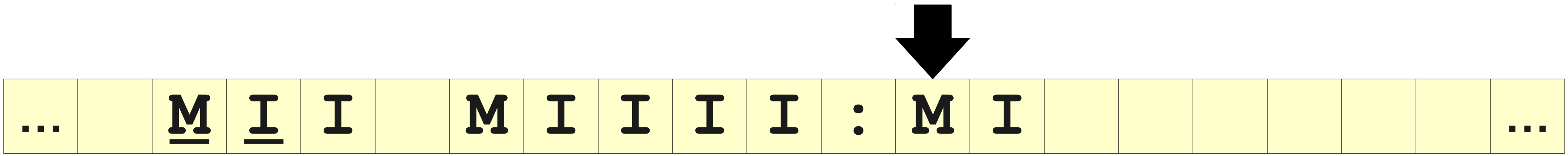
A Sketch of the Turing Machine



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.

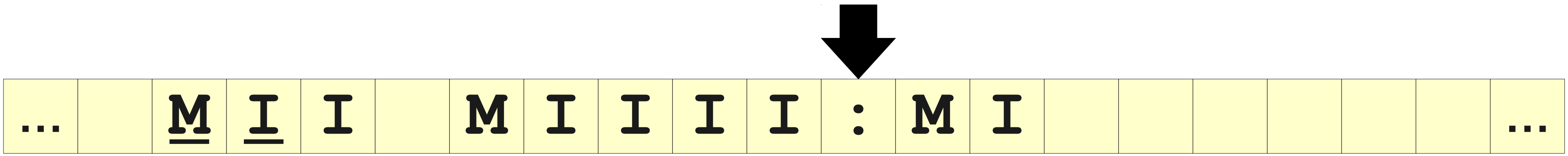
A Sketch of the Turing Machine



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.

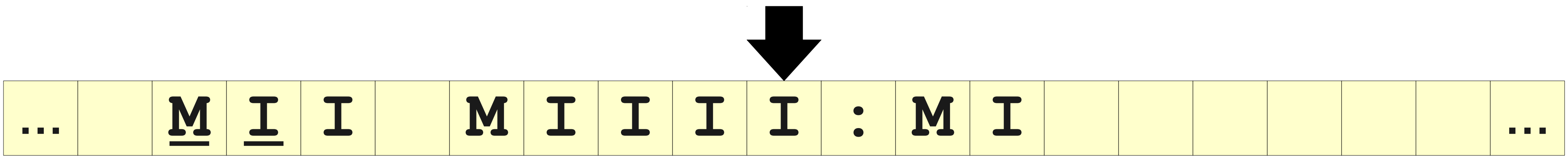
A Sketch of the Turing Machine



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.

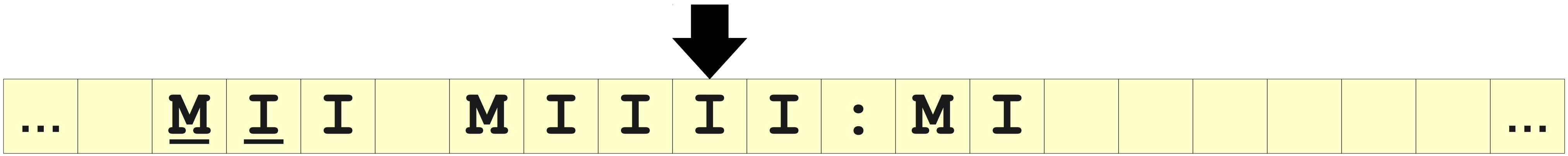
A Sketch of the Turing Machine



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.

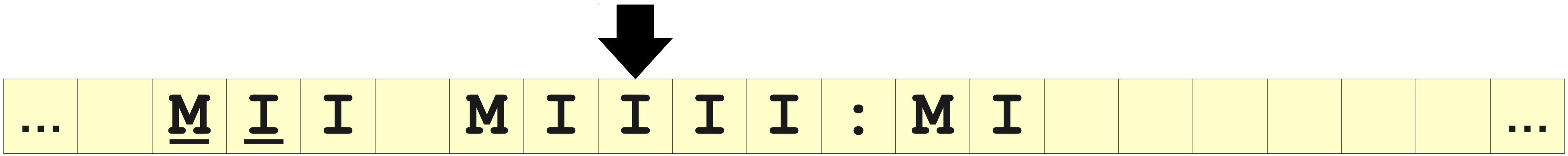
A Sketch of the Turing Machine



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.

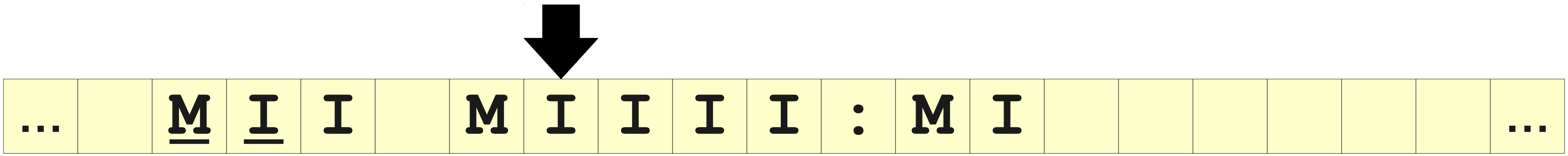
A Sketch of the Turing Machine



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.

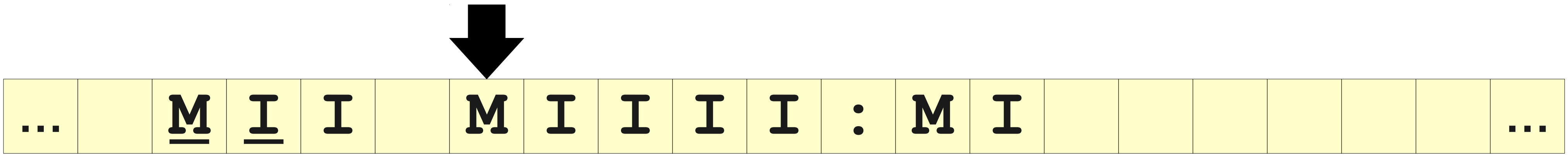
A Sketch of the Turing Machine



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.

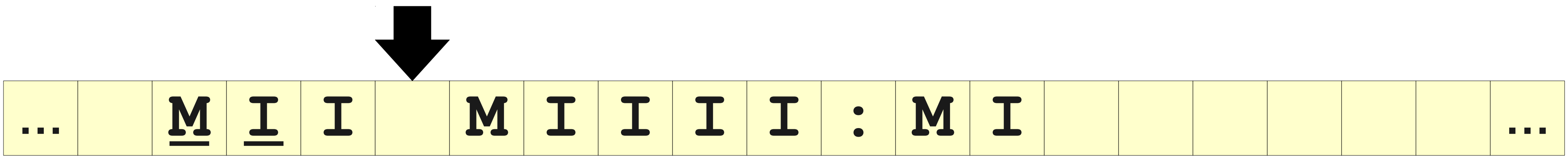
A Sketch of the Turing Machine



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.

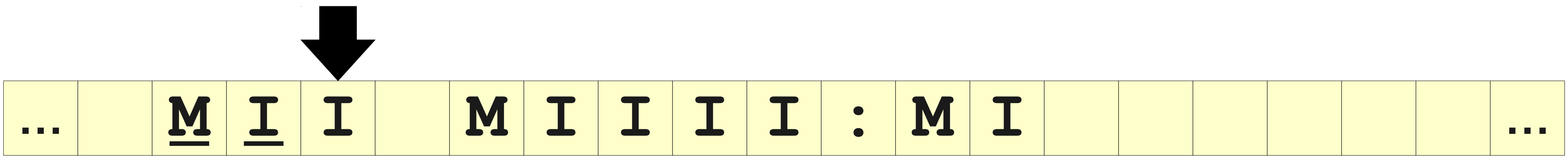
A Sketch of the Turing Machine



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.

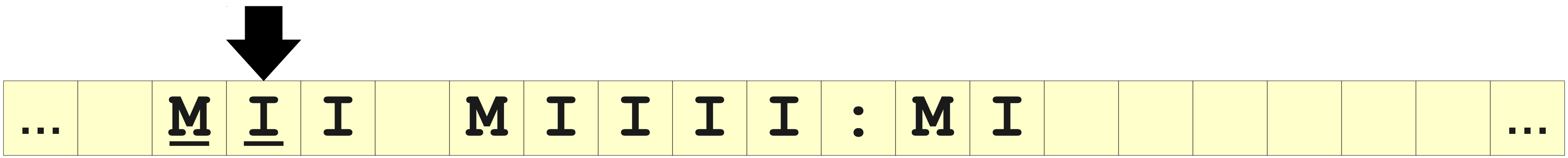
A Sketch of the Turing Machine



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.

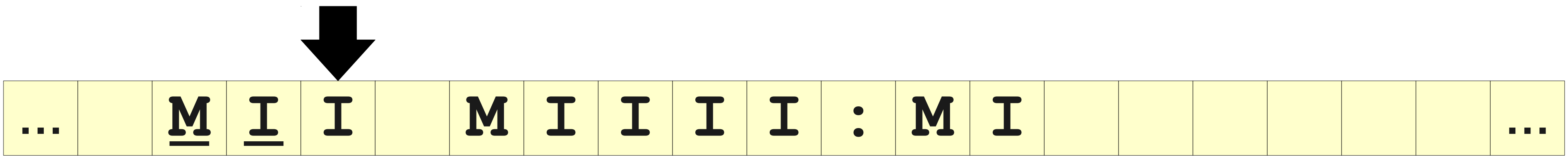
A Sketch of the Turing Machine



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.

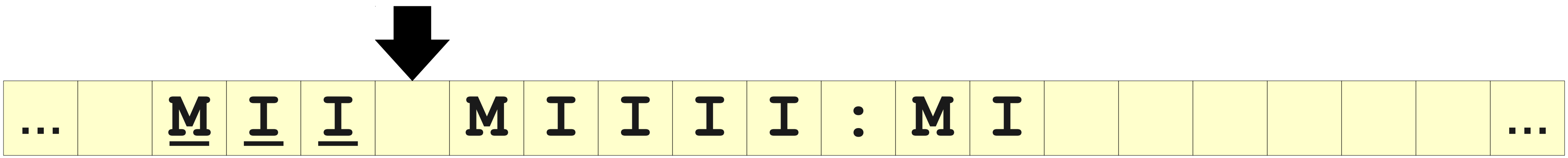
A Sketch of the Turing Machine



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.

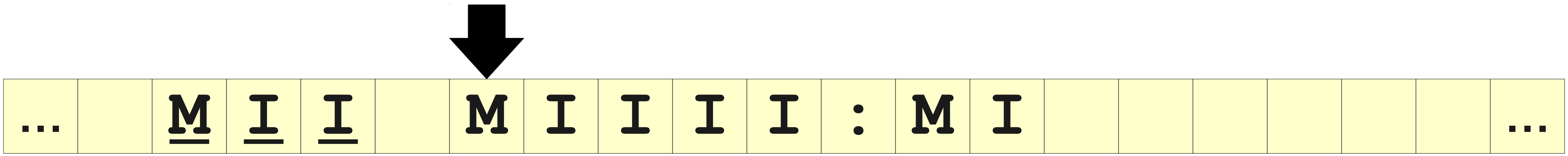
A Sketch of the Turing Machine



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.

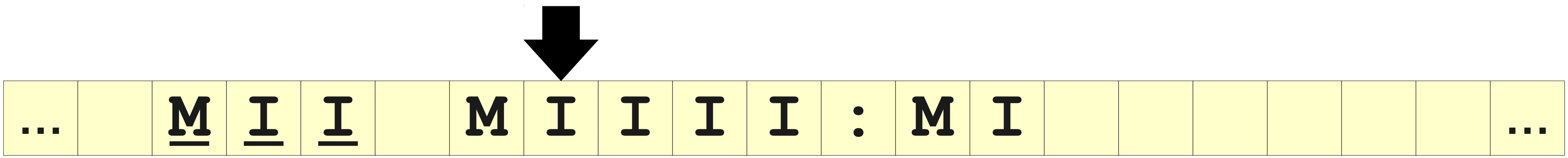
A Sketch of the Turing Machine



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.

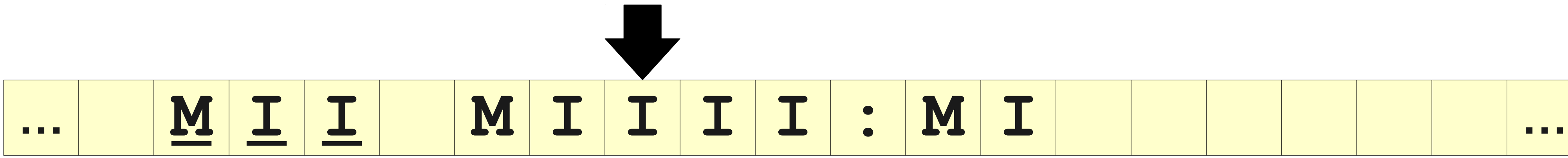
A Sketch of the Turing Machine



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.

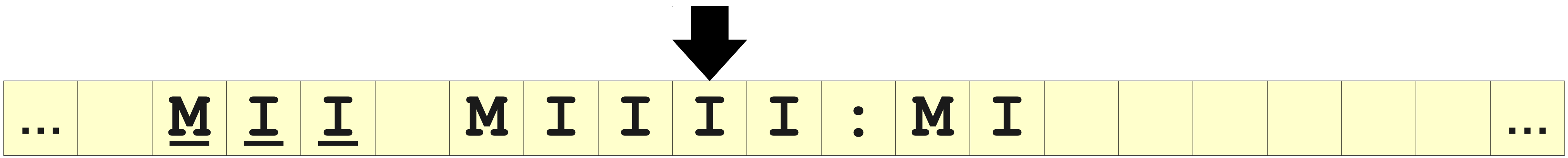
A Sketch of the Turing Machine



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.

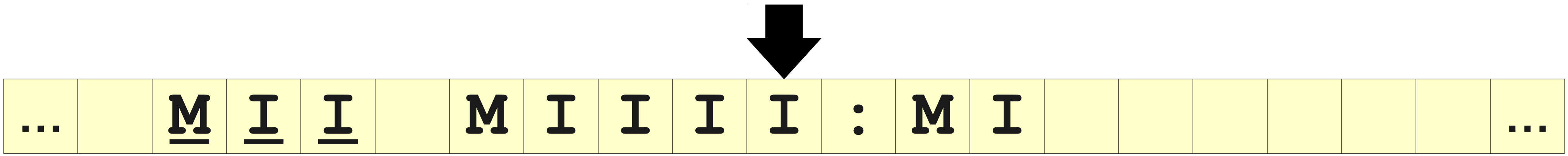
A Sketch of the Turing Machine



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.

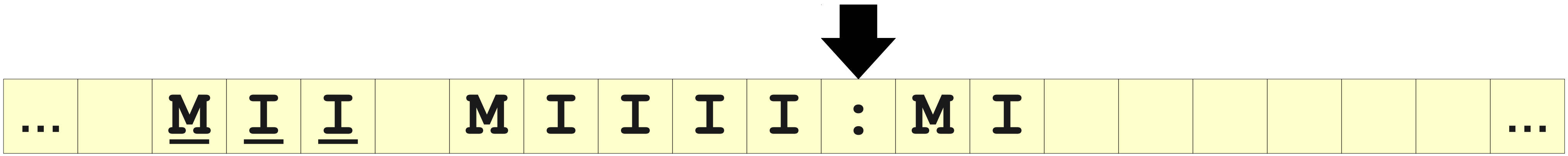
A Sketch of the Turing Machine



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.

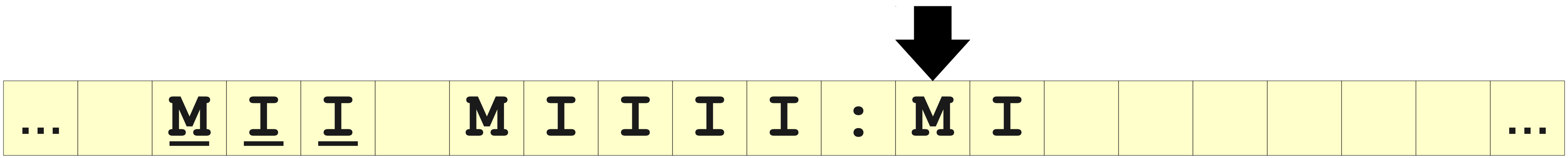
A Sketch of the Turing Machine



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.

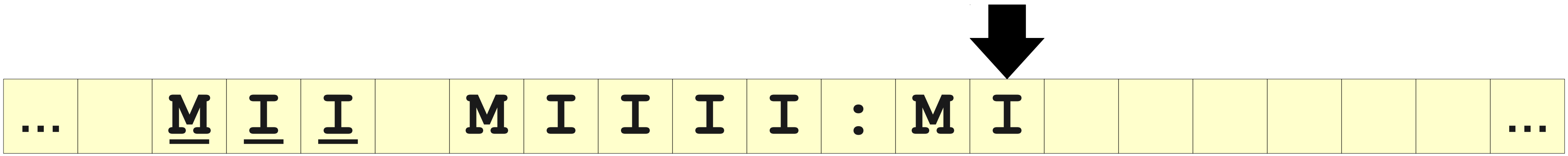
A Sketch of the Turing Machine



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.

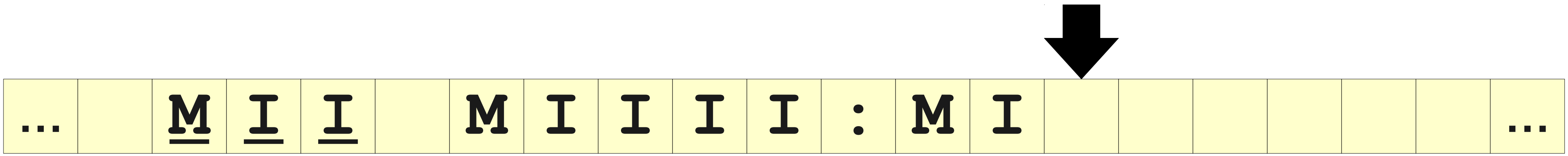
A Sketch of the Turing Machine



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.

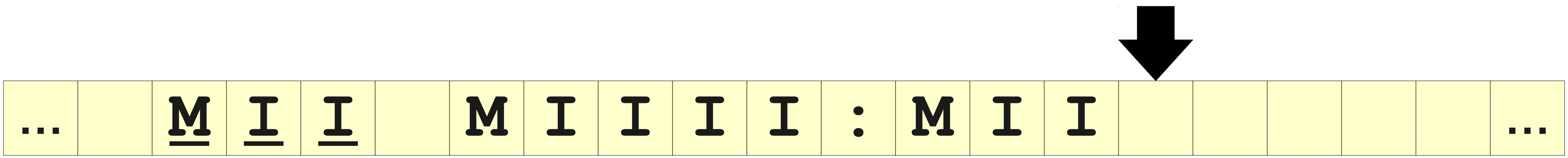
A Sketch of the Turing Machine



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.

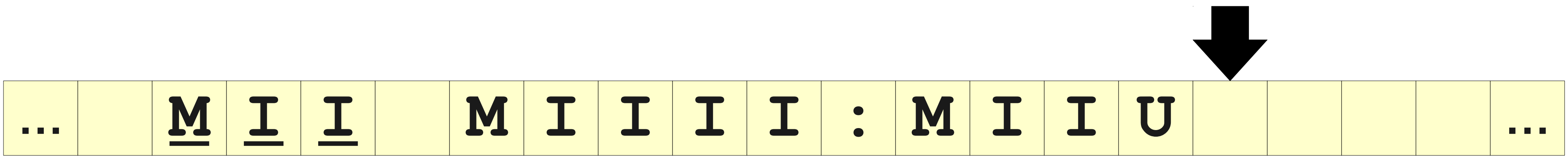
A Sketch of the Turing Machine



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.

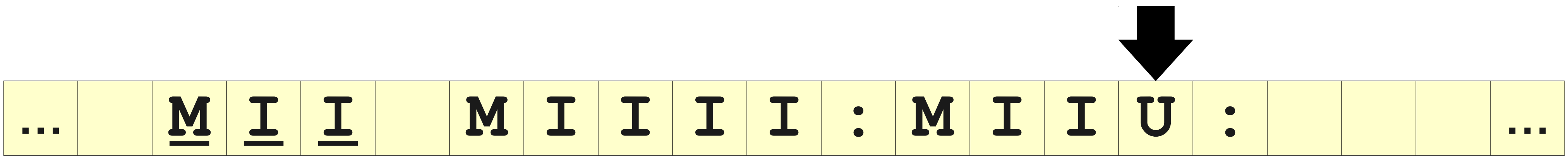
A Sketch of the Turing Machine



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.

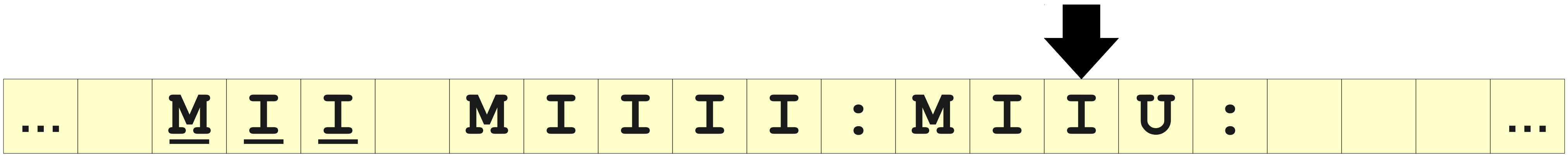
A Sketch of the Turing Machine



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.

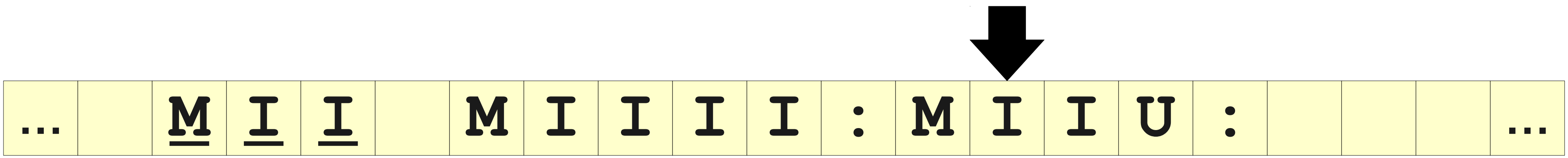
A Sketch of the Turing Machine



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.

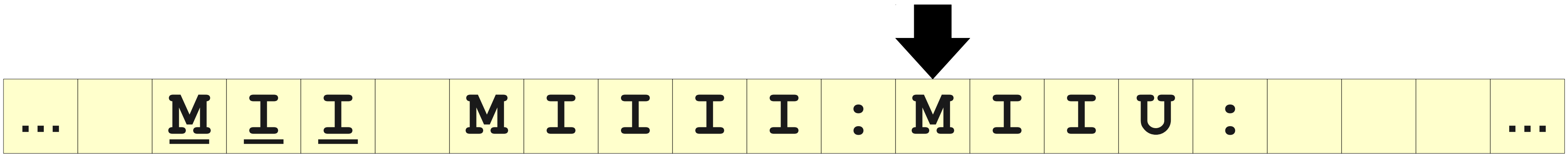
A Sketch of the Turing Machine



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.

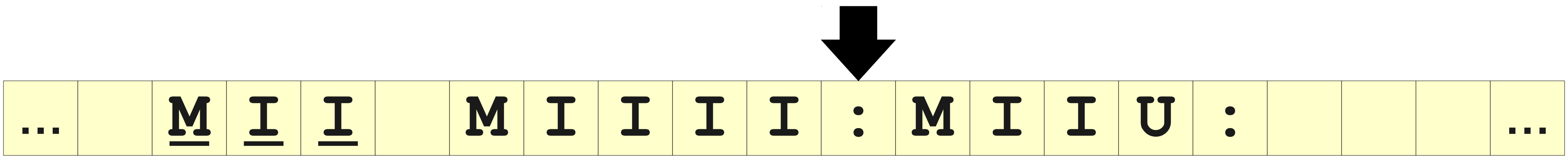
A Sketch of the Turing Machine



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.

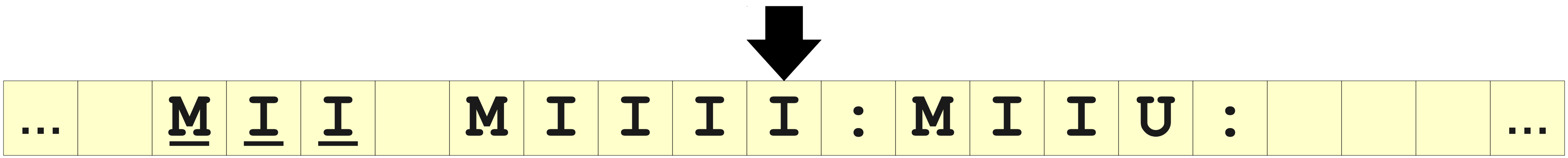
A Sketch of the Turing Machine



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.

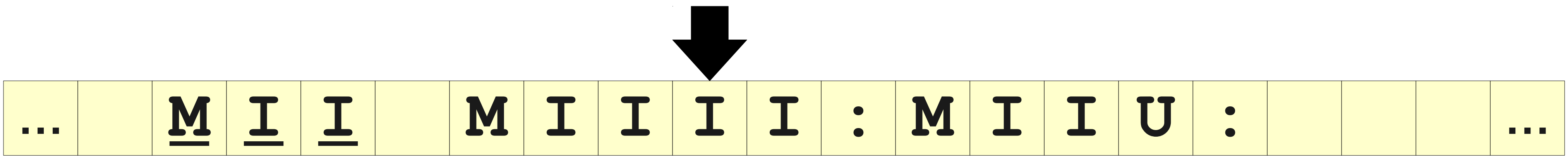
A Sketch of the Turing Machine



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.

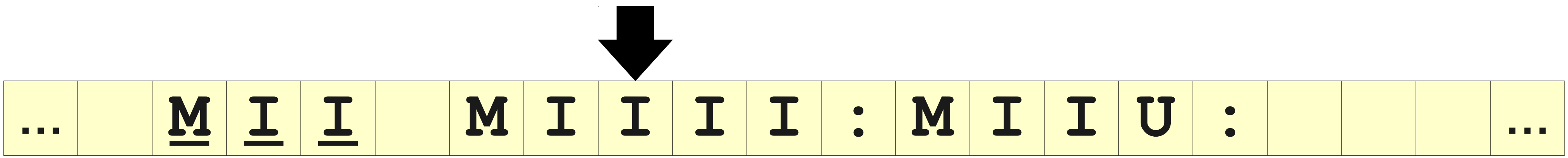
A Sketch of the Turing Machine



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.

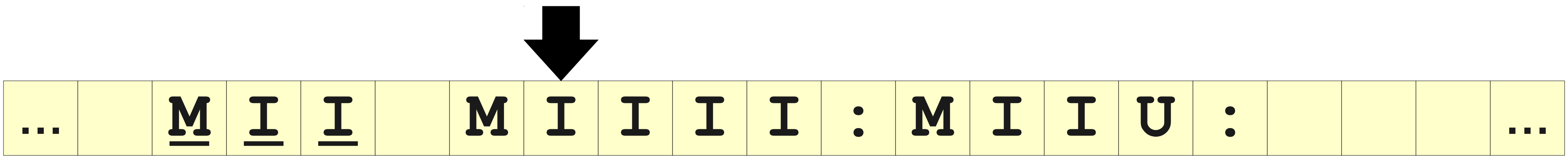
A Sketch of the Turing Machine



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.

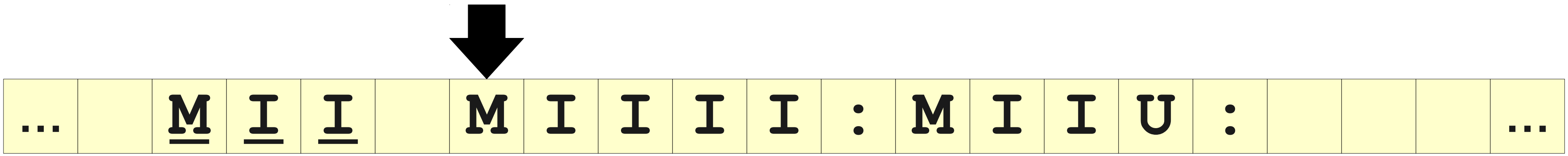
A Sketch of the Turing Machine



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.

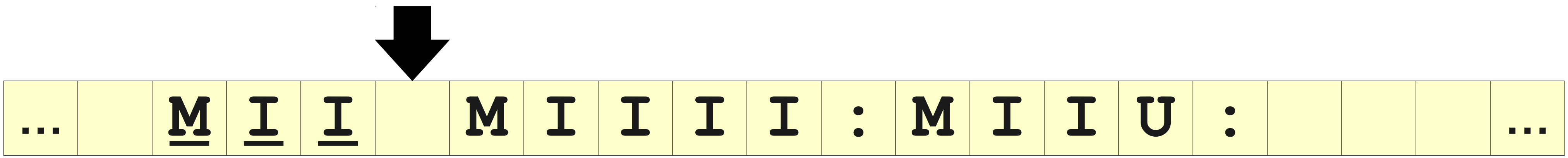
A Sketch of the Turing Machine



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.

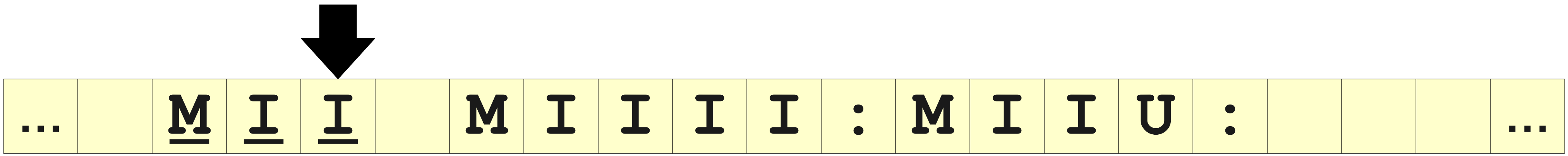
A Sketch of the Turing Machine



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.

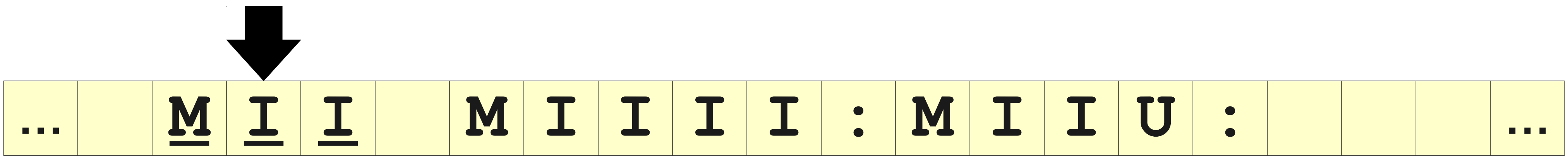
A Sketch of the Turing Machine



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.

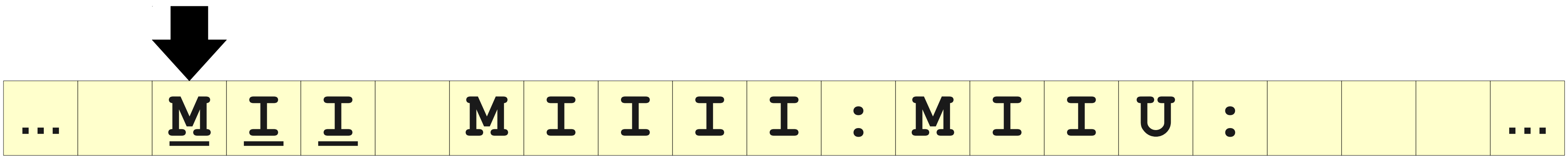
A Sketch of the Turing Machine



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.

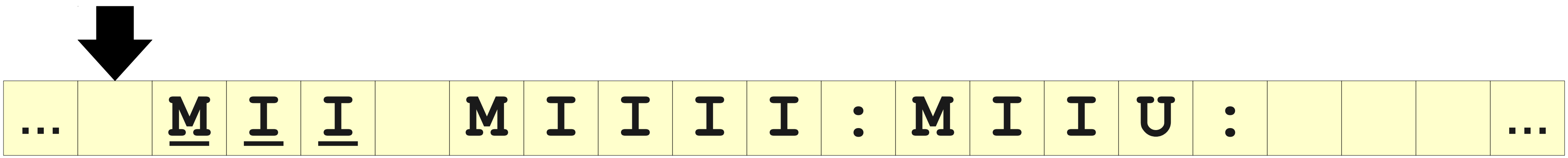
A Sketch of the Turing Machine



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.

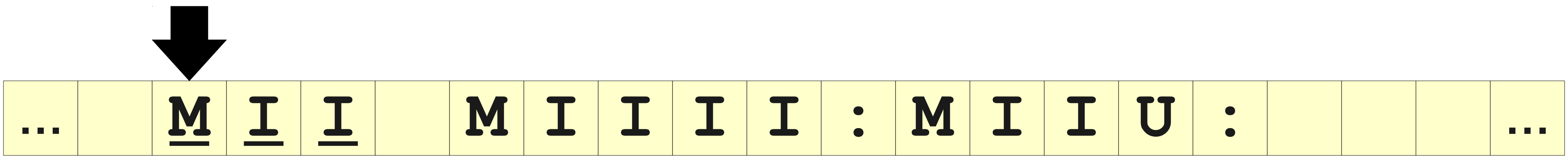
A Sketch of the Turing Machine



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.

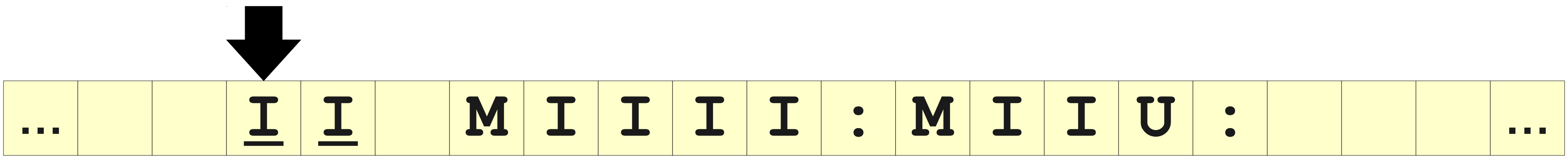
A Sketch of the Turing Machine



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.

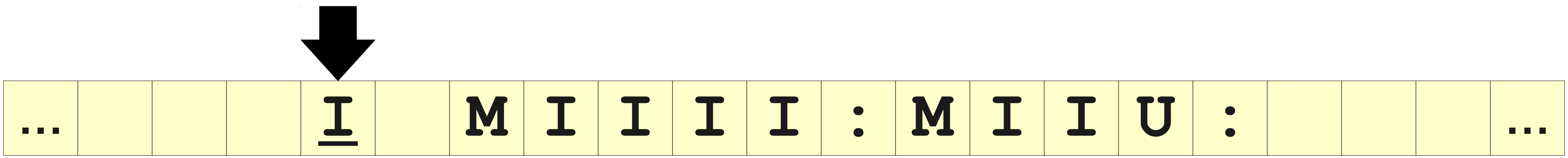
A Sketch of the Turing Machine



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.

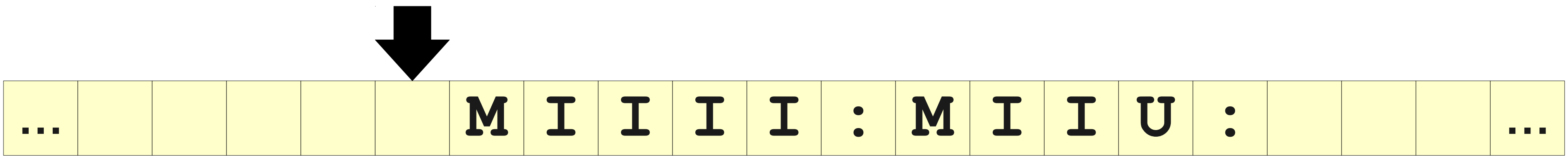
A Sketch of the Turing Machine



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.

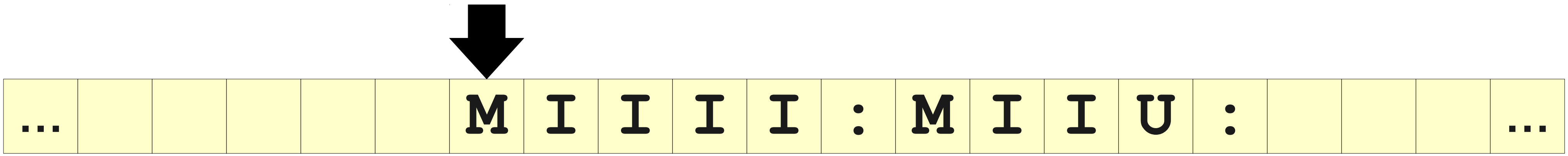
A Sketch of the Turing Machine



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.

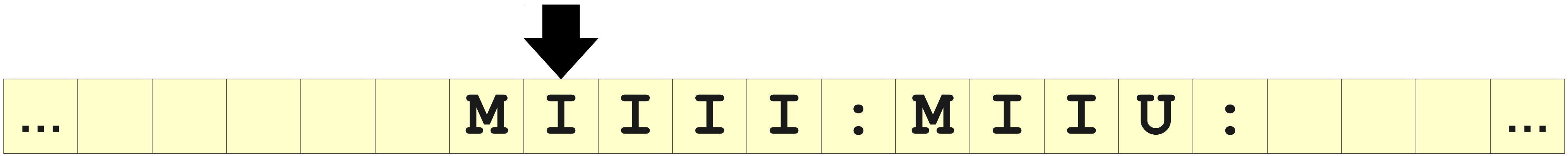
A Sketch of the Turing Machine



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.

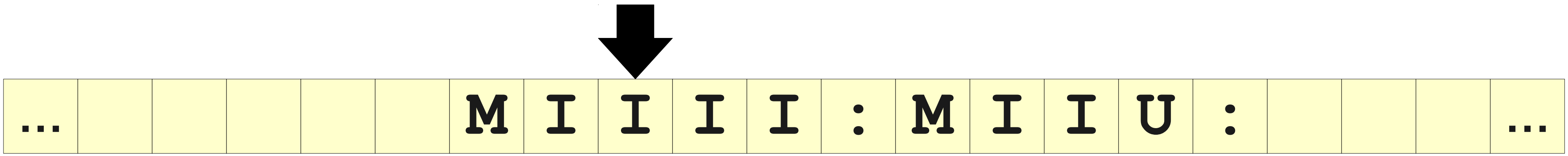
A Sketch of the Turing Machine



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.

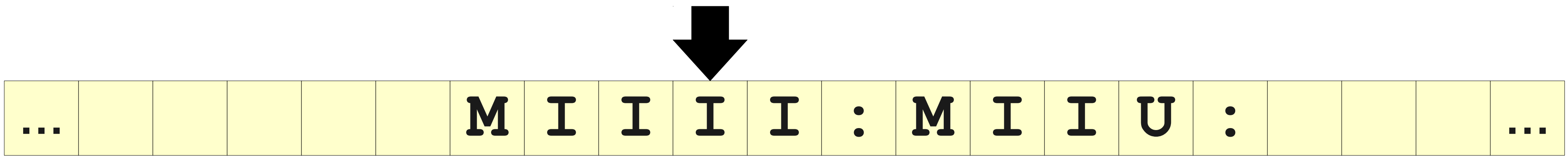
A Sketch of the Turing Machine



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.

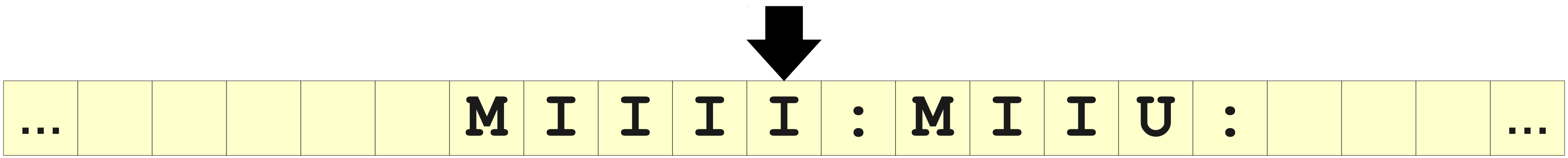
A Sketch of the Turing Machine



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.

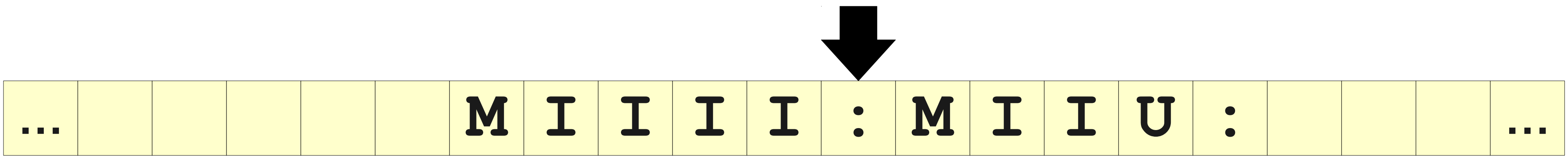
A Sketch of the Turing Machine



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.

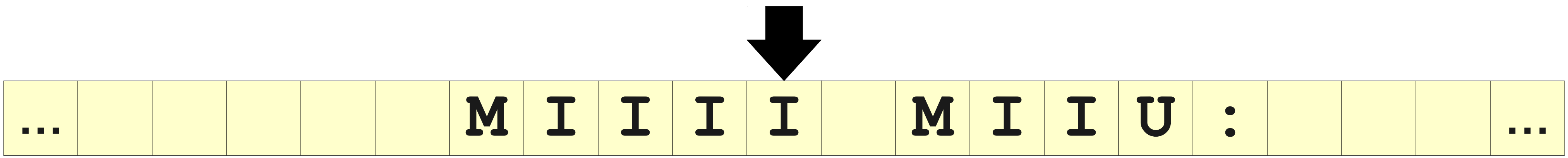
A Sketch of the Turing Machine



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.

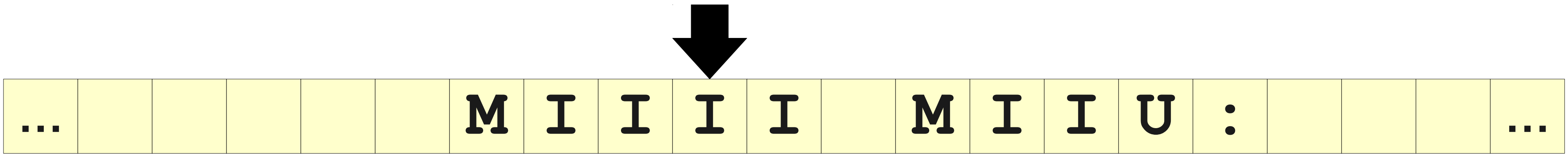
A Sketch of the Turing Machine



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.

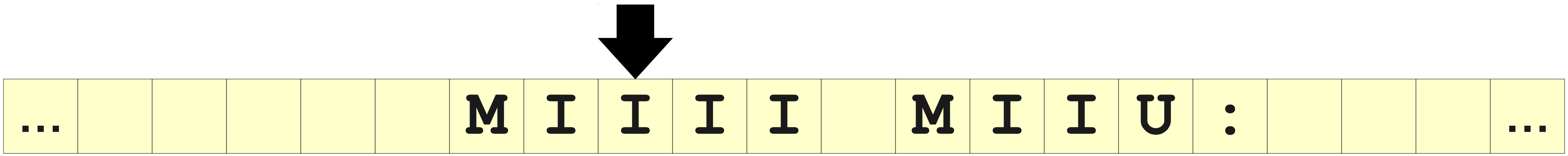
A Sketch of the Turing Machine



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.

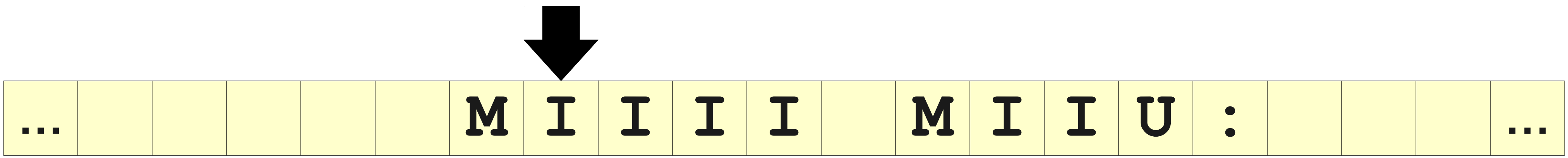
A Sketch of the Turing Machine



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.

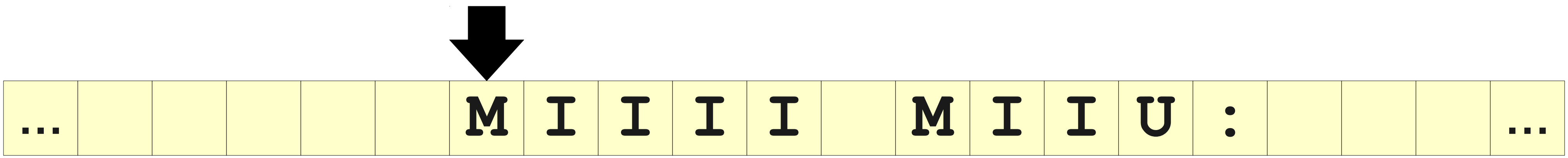
A Sketch of the Turing Machine



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.

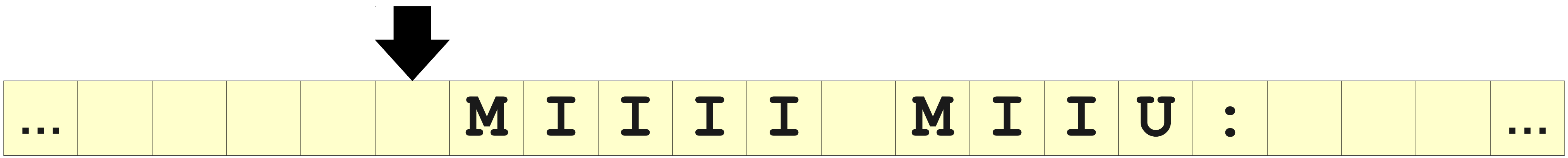
A Sketch of the Turing Machine



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.

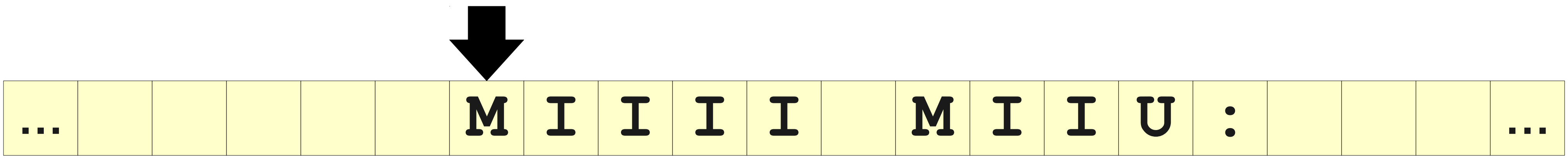
A Sketch of the Turing Machine



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.

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.

Searching and Guessing

Nondeterminism Revisited

- Recall: One intuition for nondeterminism is **perfect guessing**.
  - The machine has many options, and somehow magically knows which guess to make.
- With regular languages, we could *simulate* perfect guessing by building an enormous DFA to try out each option in parallel.
  - Only finitely many possible options.
- With context-free languages, some NPDAs do not have corresponding DPDAs.
  - Cannot simulate all possible stacks with just one stack.

Nondeterministic TMs

- A **nondeterministic Turing machine** (or **NTM**) is a variant on a Turing machine where there can be any number of transitions for a given state/tape symbol combination.
  - Notation: “Turing machine” or “TM” refers to a deterministic Turing machine unless specified otherwise. The term **DTM** specifically represents a deterministic TM.
- The NTM accepts iff there is *some possible series of choices* it can make such that it accepts.

Questions for Now

- How can we build an intuition for nondeterministic Turing machines?
- What sorts of problems can we solve with NTMs?
- What is the relative power of NTMs and DTMs?

Designing NTMs

- When designing NTMs, it is often useful to use the approach of guess and check:
  - **Nondeterministically** guess some object that can "prove" that $w \in L$.
  - **Deterministically** verify that you have guessed the right object.
- If $w \in L$, there will be some guess that causes the machine to accept.
- If $w \notin L$, then no guess will ever cause the machine to accept.

Composite Numbers

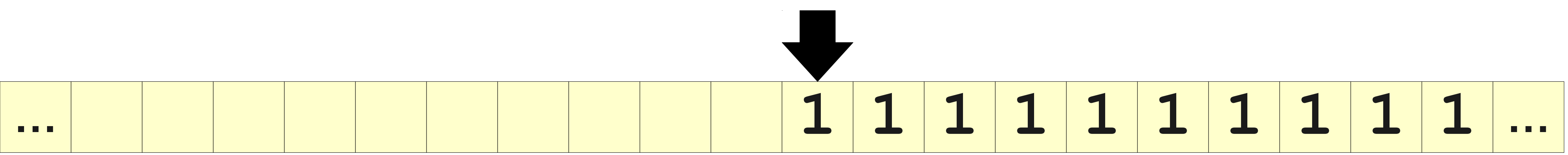
- A natural number $n \geq 2$ is called **composite** iff it has a factor other than 1 and n.
- Equivalently: there are two natural numbers $r \geq 2$ and $s \geq 2$ such that $rs = n$.
- Let $\Sigma = \{\mathtt{1}\}$ and consider the language

$$L = \{\ \mathtt{1}^n \mid n \text{ is composite }\}$$

- How might we design an NTM for L?

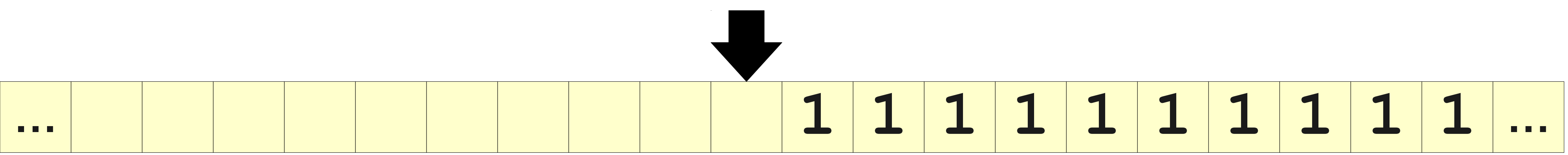
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



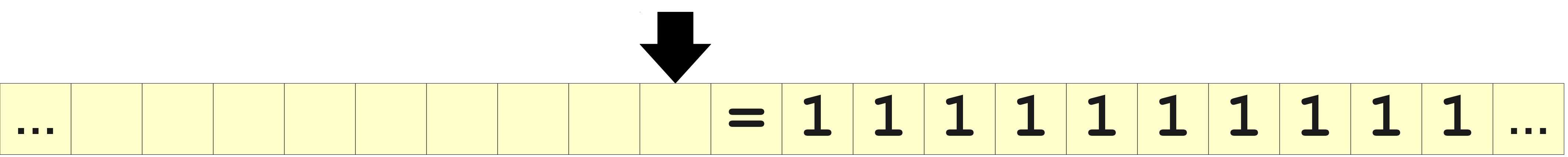
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



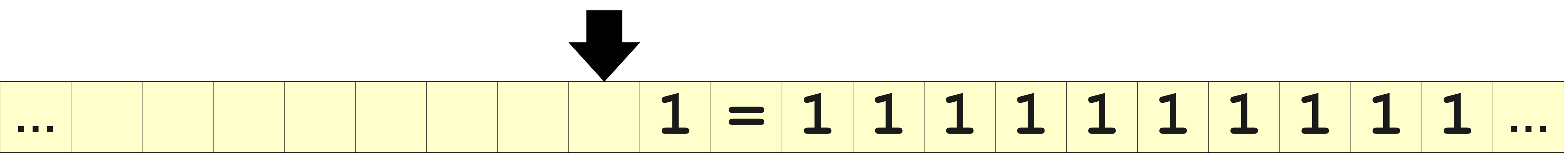
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



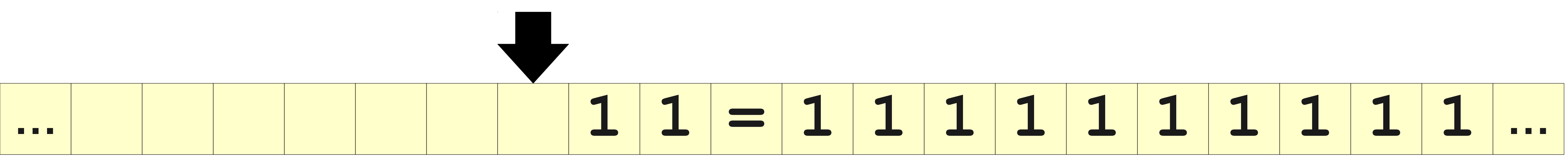
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



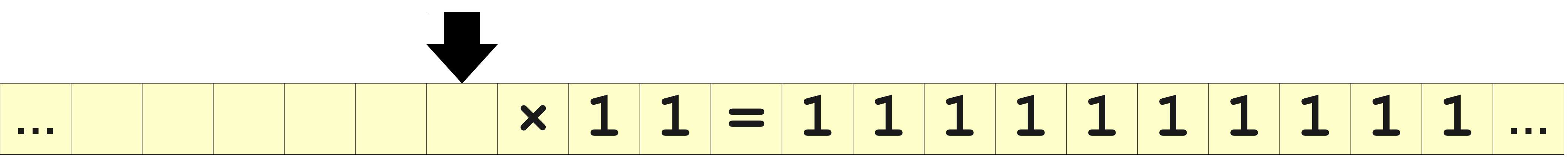
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



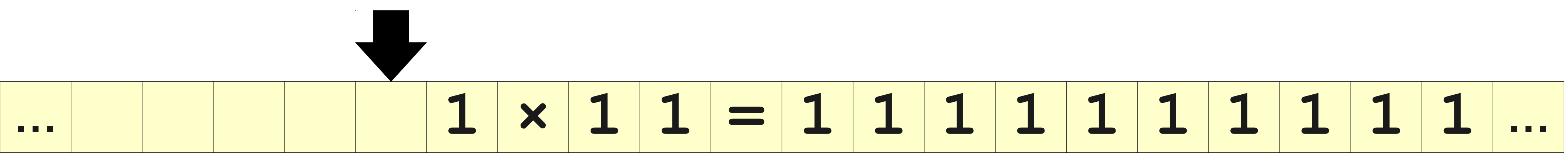
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



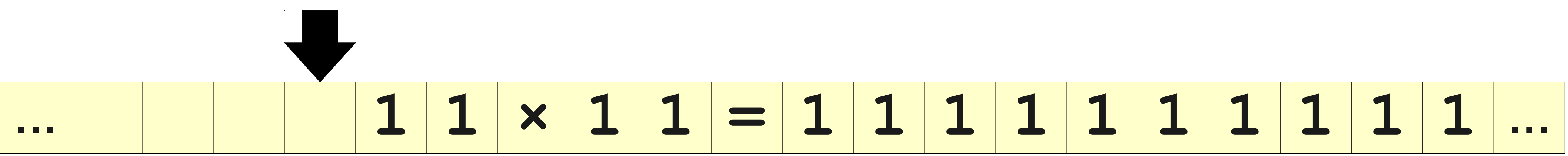
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



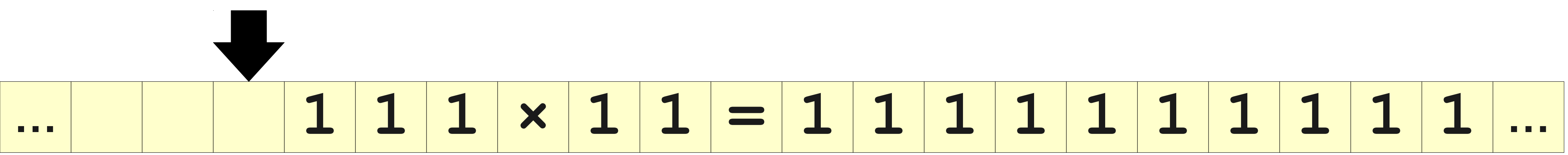
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



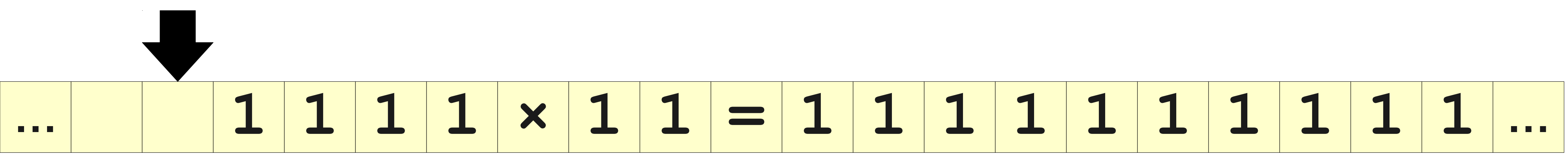
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



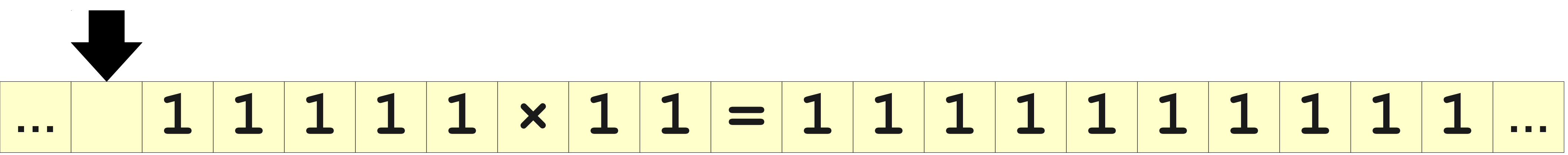
A Sketch of the NTM

- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.



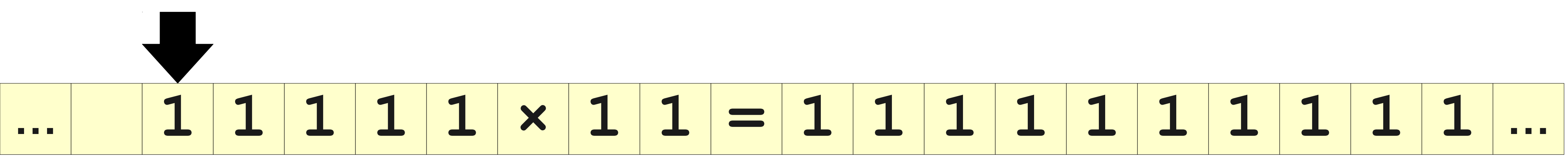
A Sketch of the NTM

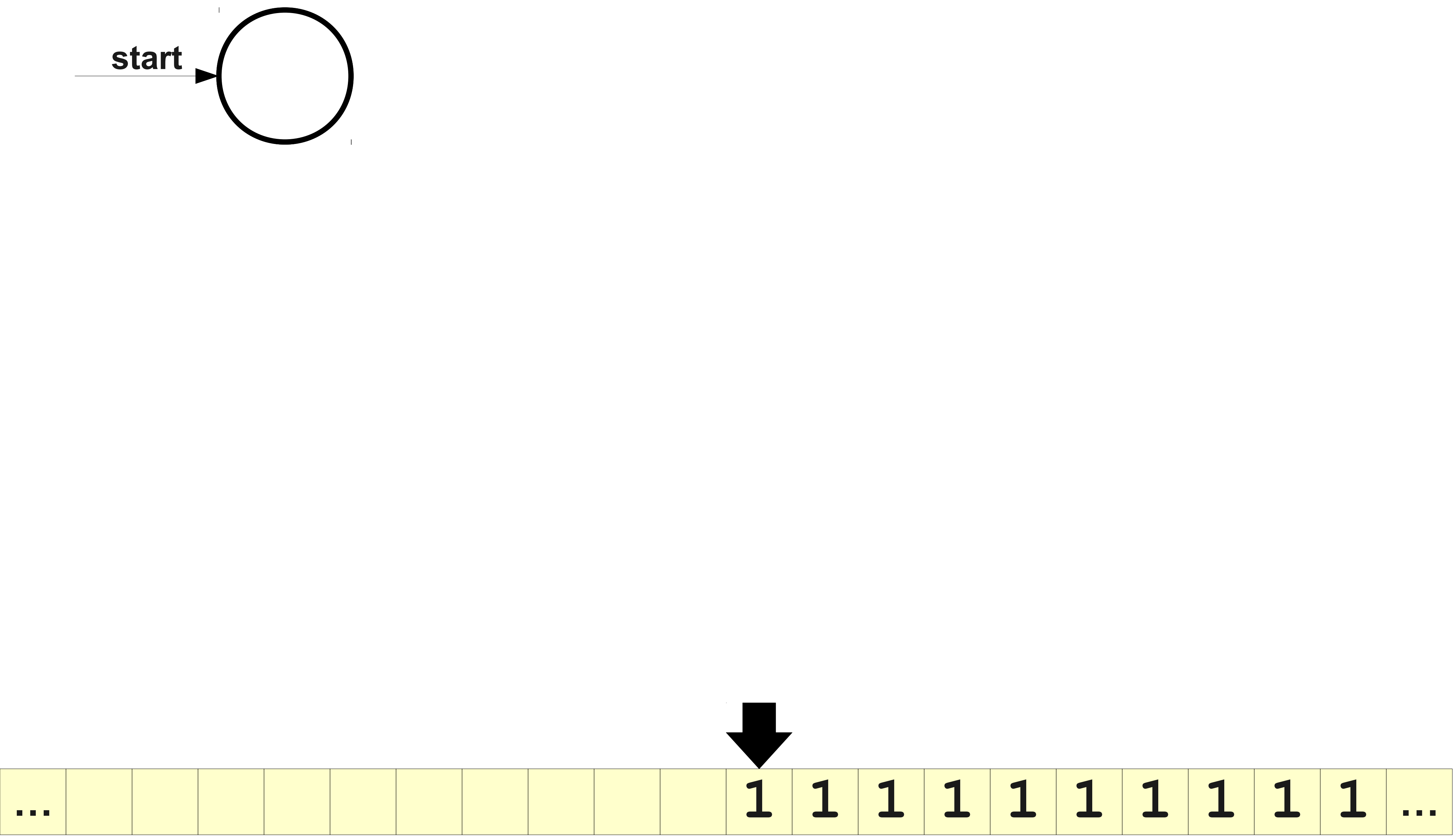
- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.

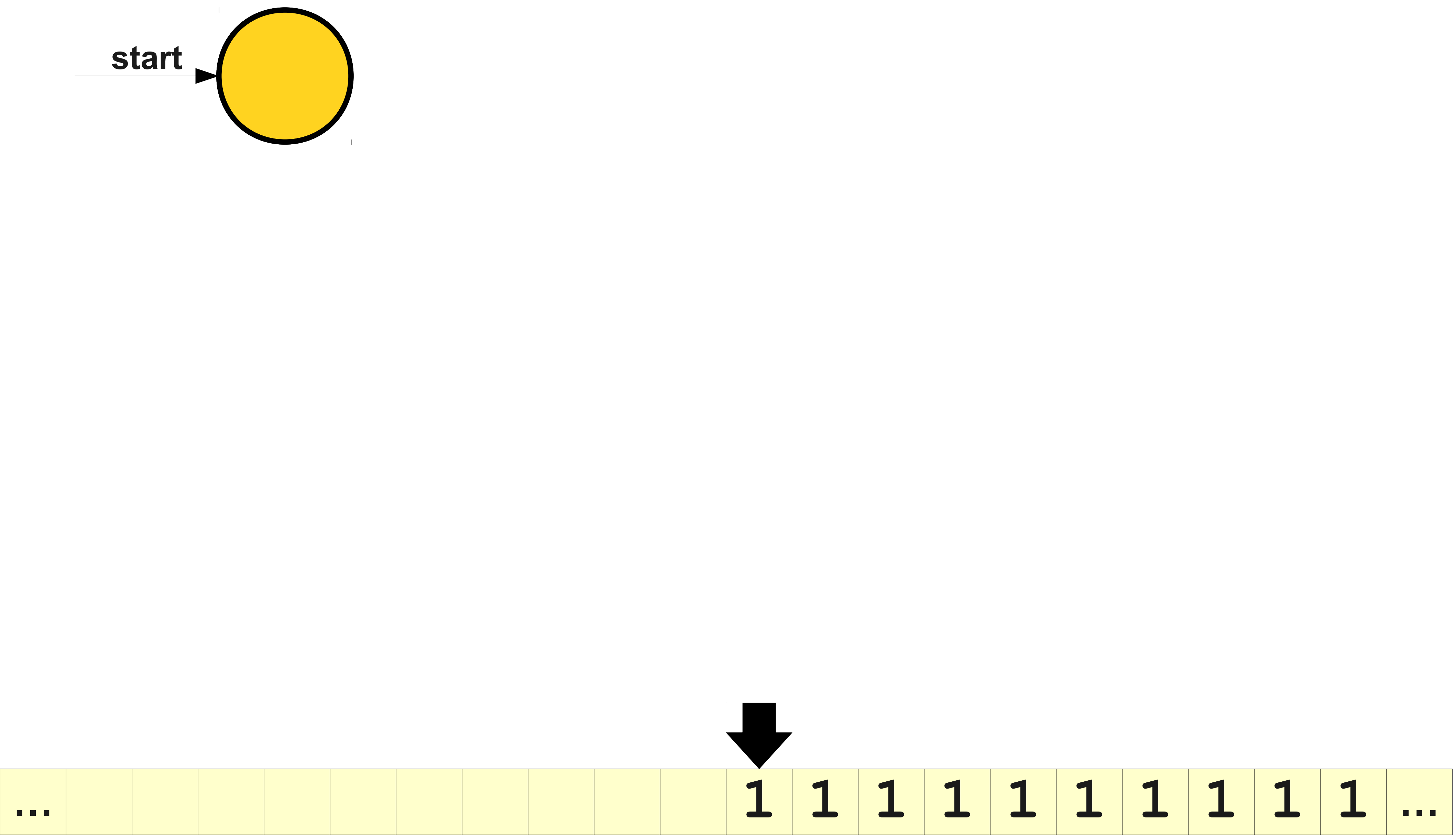


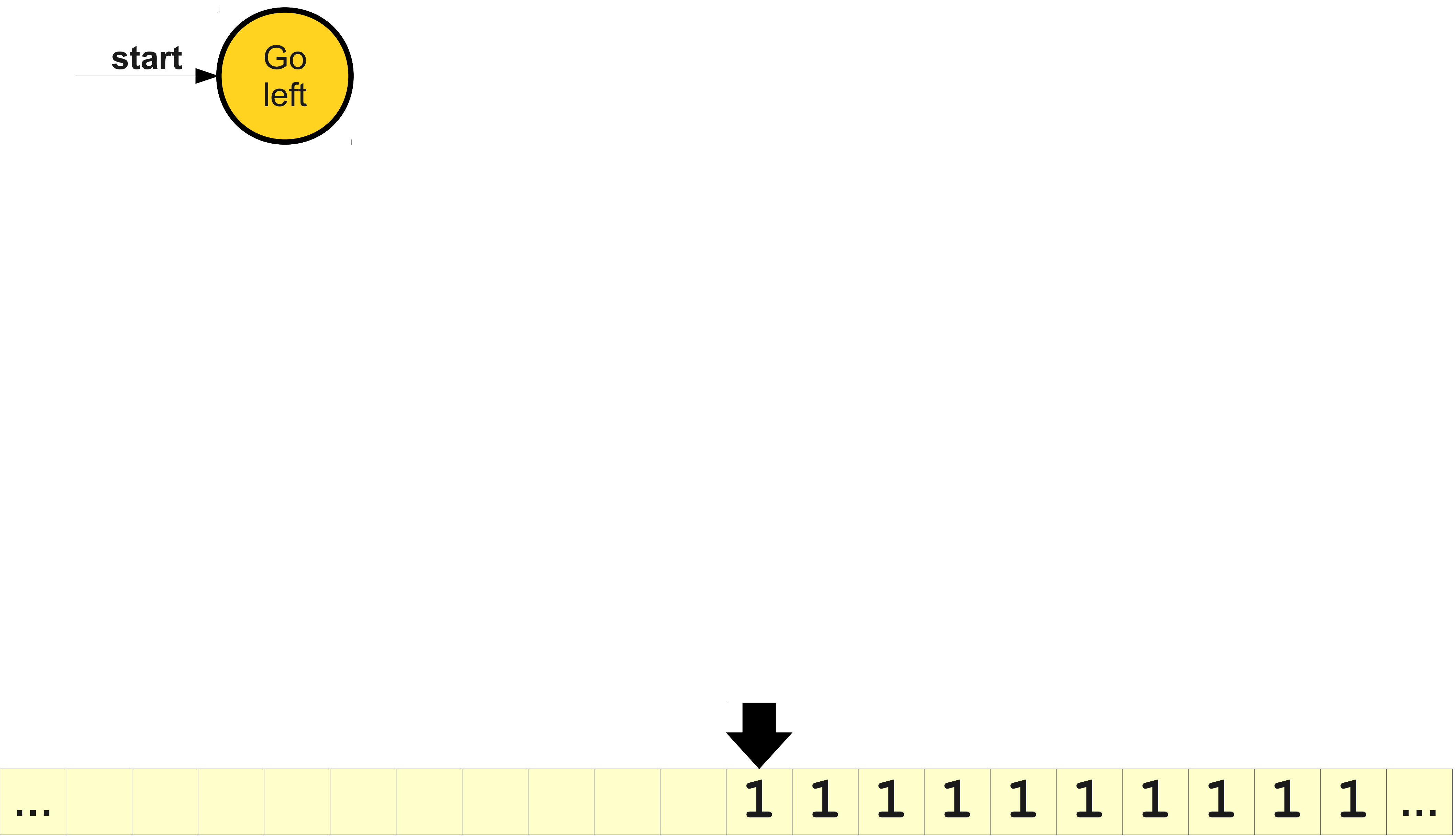
A Sketch of the NTM

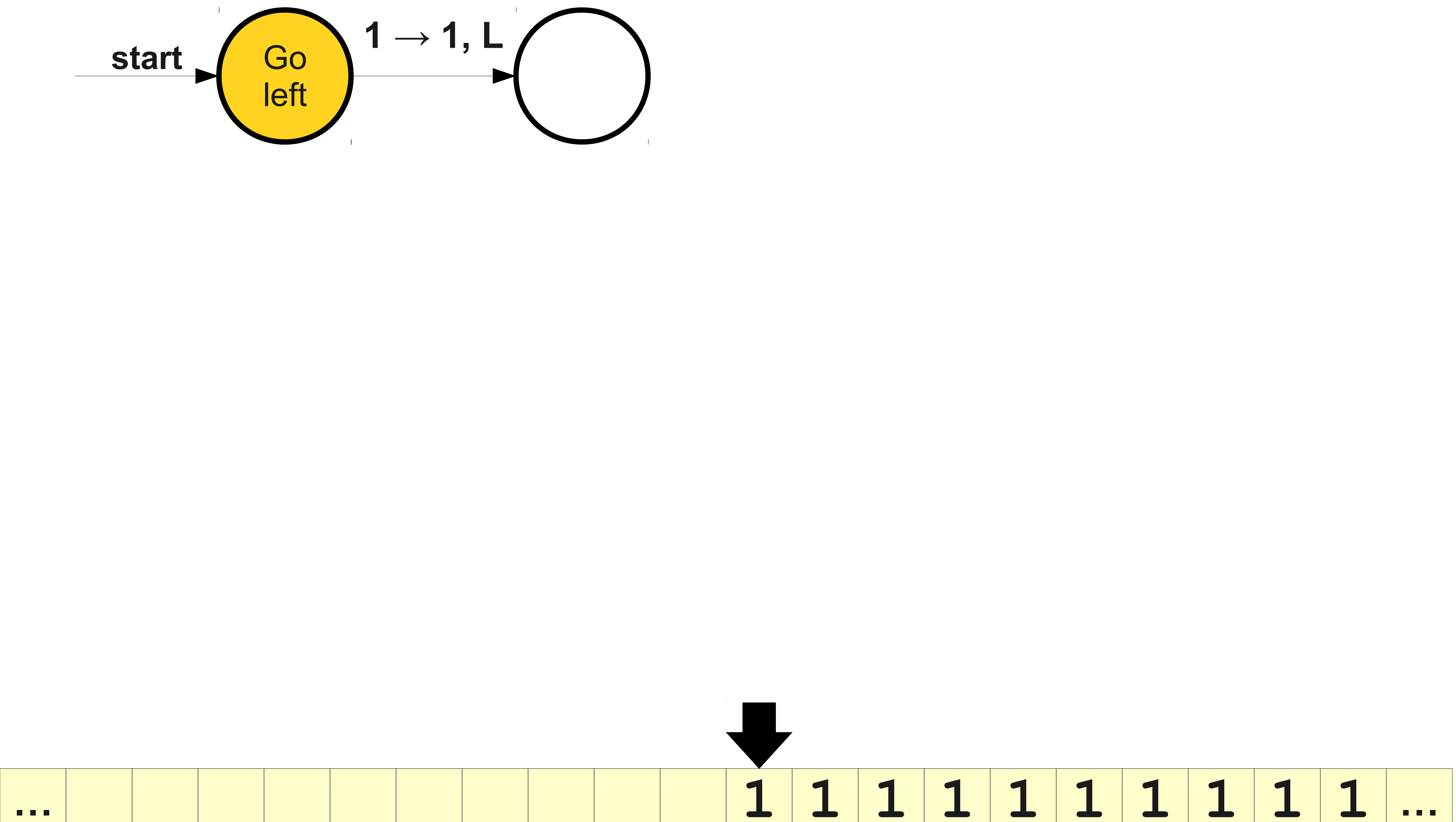
- We saw how to build a TM that checks for correct multiplication.
- Have our NTM
  - **Nondeterministically** guess two factors, then
  - **Deterministically** run the multiplication TM.

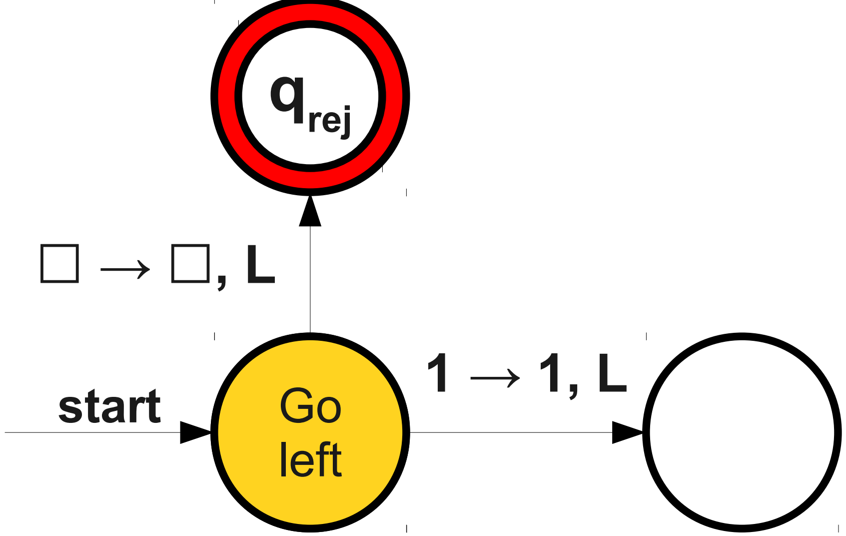



start
...
1 1 1 1 1 1 1 1 1 1 1
...

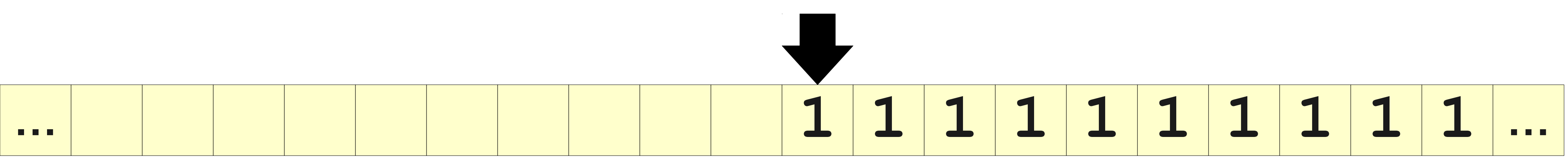

start
...
1 1 1 1 1 1 1 1 1 1
...


start
Go left
...
1 1 1 1 1 1 1 1 1 1 1
...

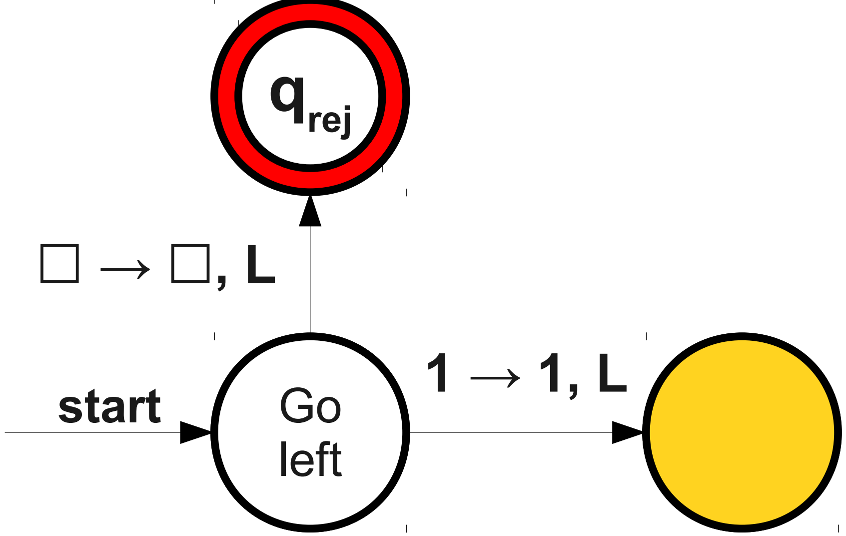

start
Go left
1 → 1, L
…
1 1 1 1 1 1 1 1 1 1
…



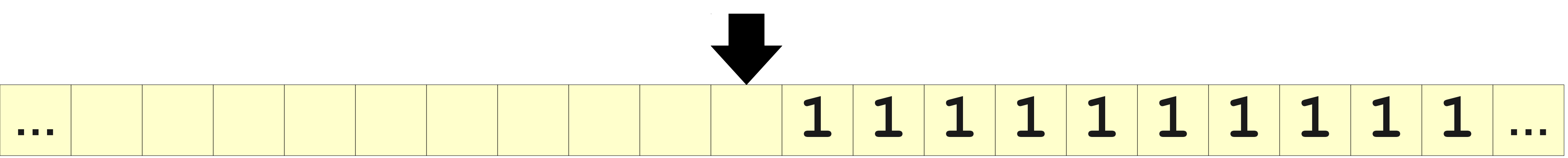
q_rej
□ → □, L
start
Go left
1 → 1, L



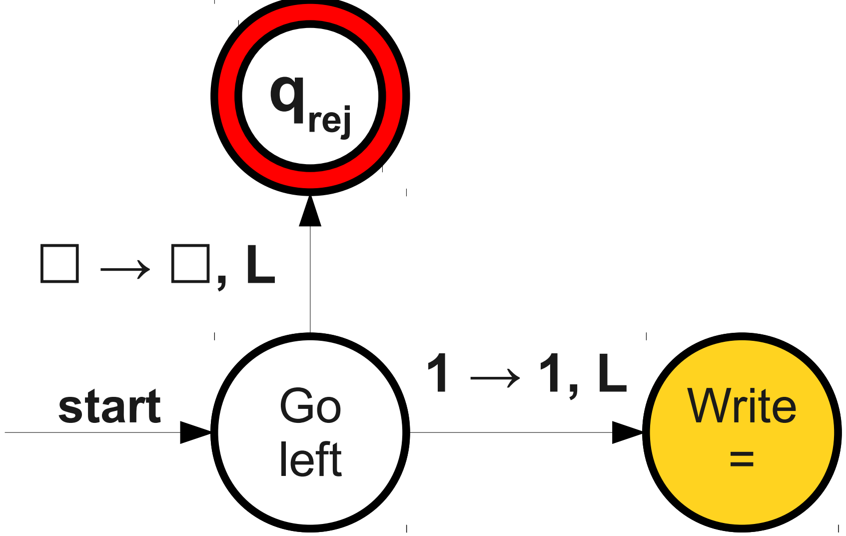
...
1 1 1 1 1 1 1 1 1 1
...

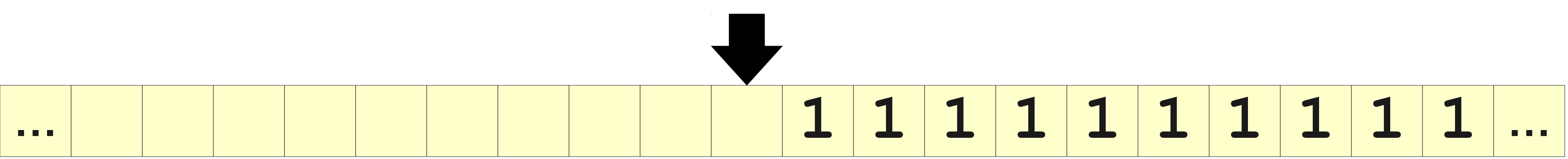


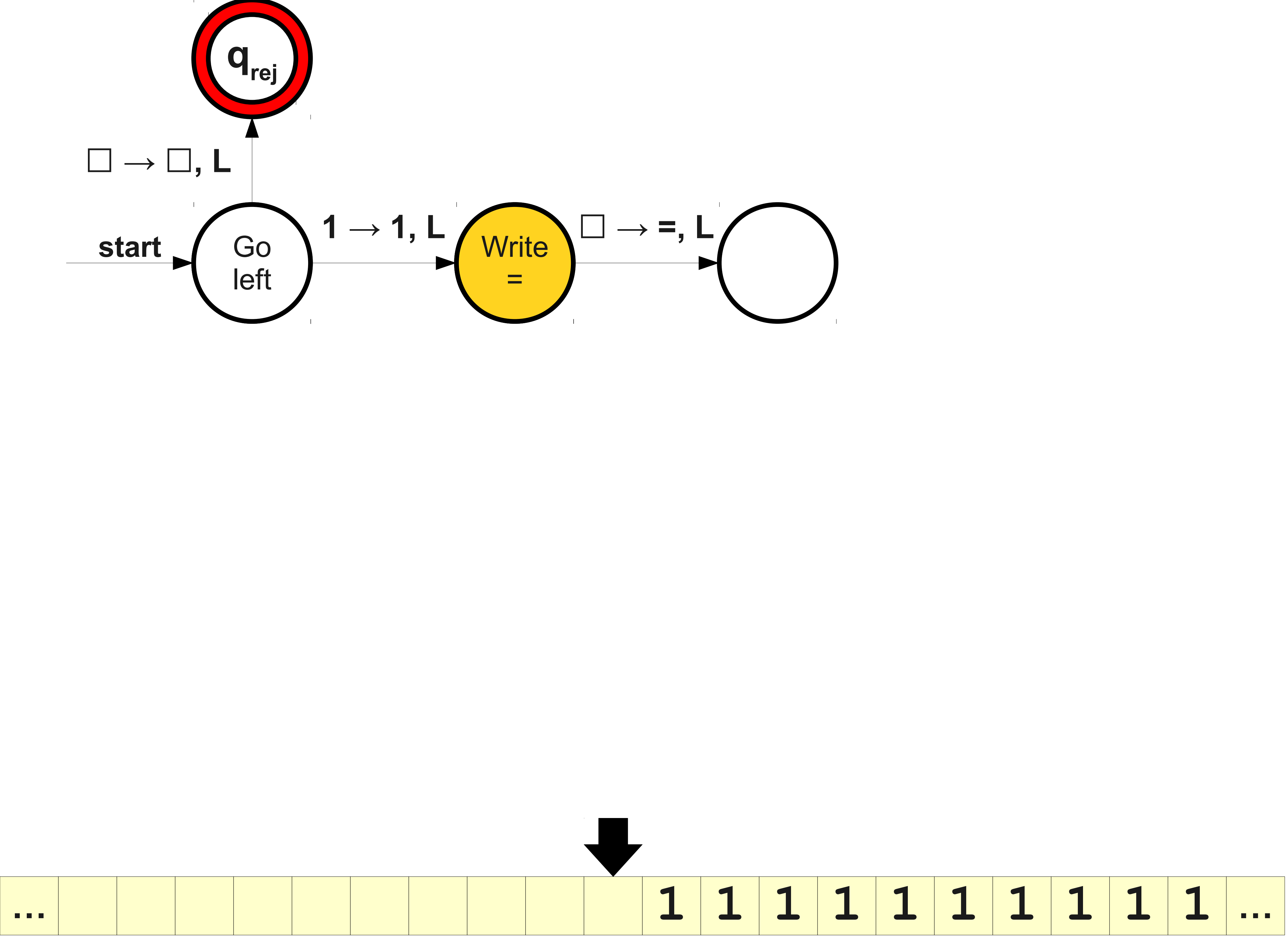
q_rej
□ → □, L
start
Go left
1 → 1, L



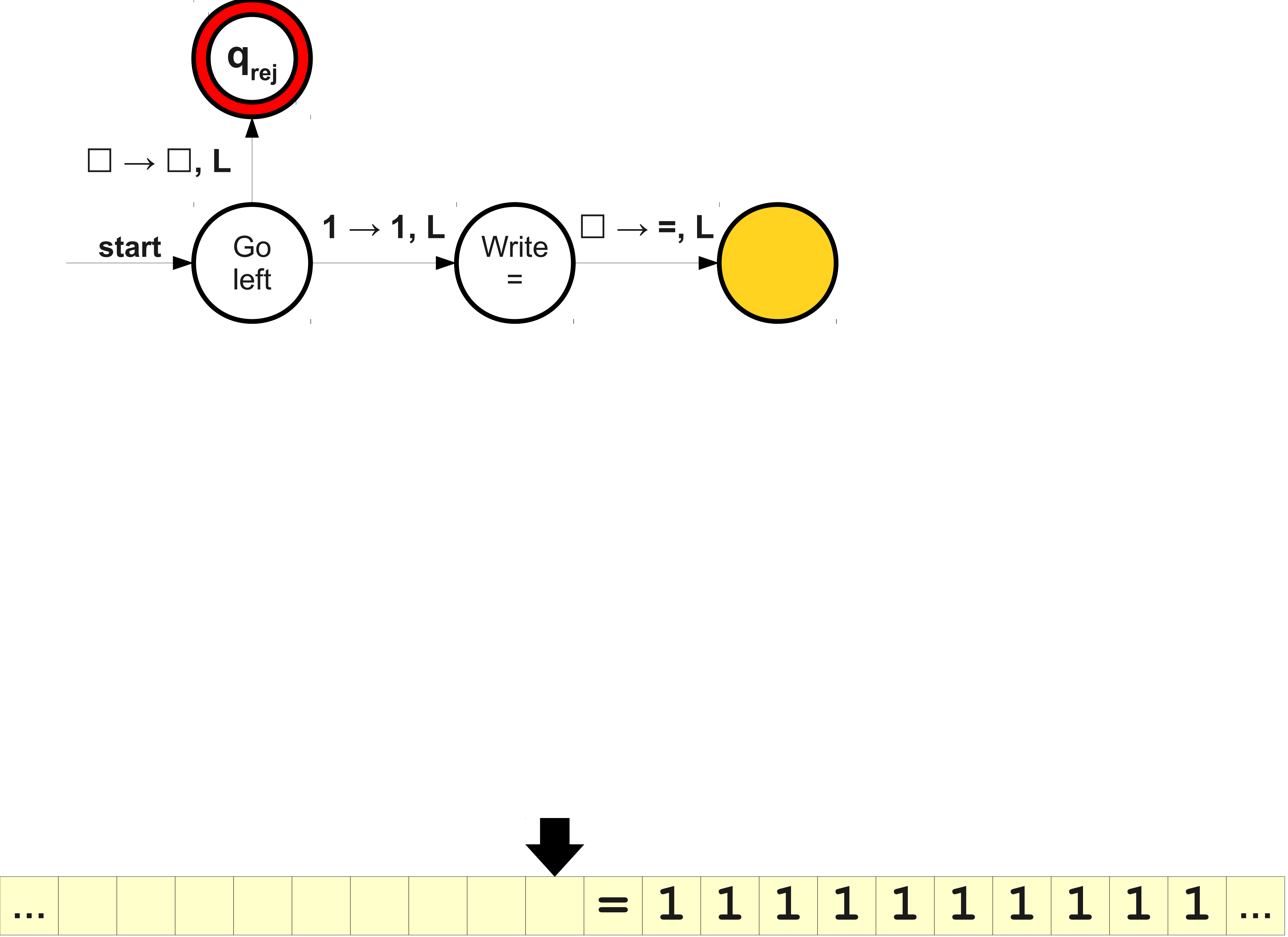
... 1 1 1 1 1 1 1 1 1 1 ...


q_{rej}
□ → □, L
start
Go left
1 → 1, L
Write =

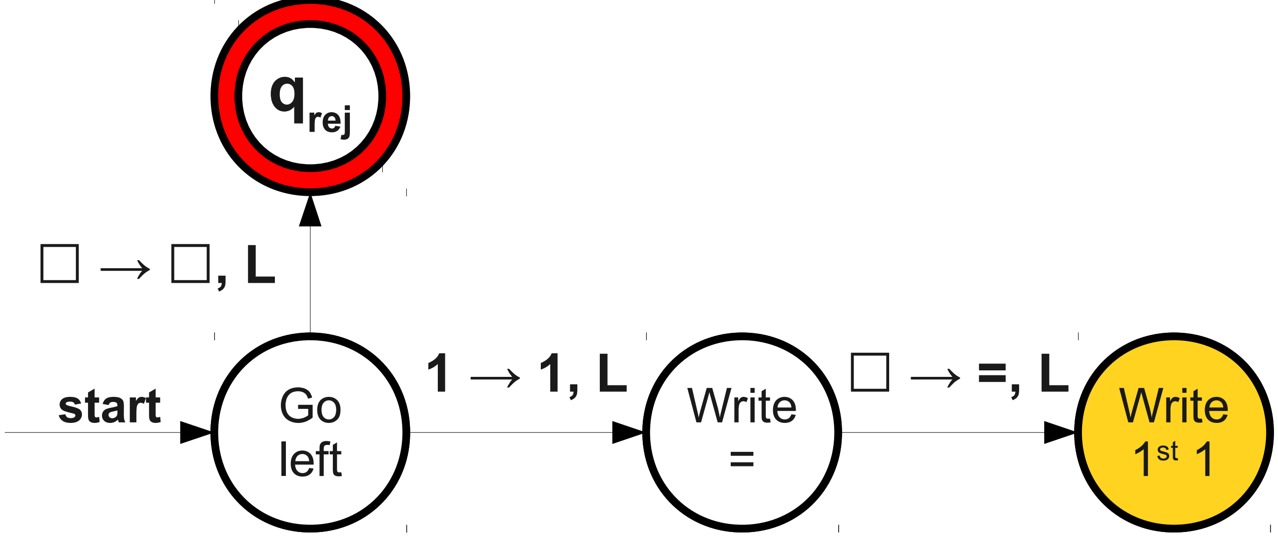

...
1 1 1 1 1 1 1 1 1 1
...



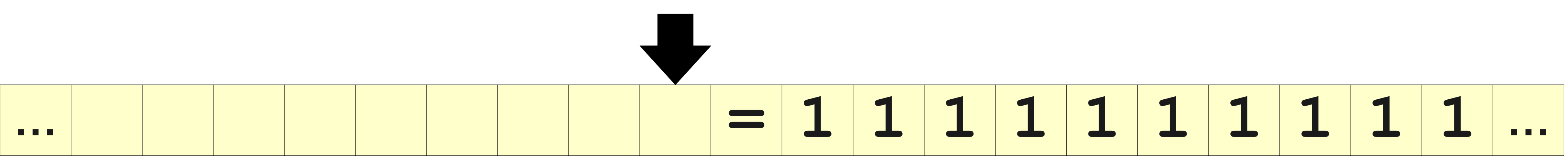
q_{rej}
$\square \to \square$, L
start
Go left
1 → 1, L
Write =
$\square \to$ =, L
... 1 1 1 1 1 1 1 1 1 1 ...



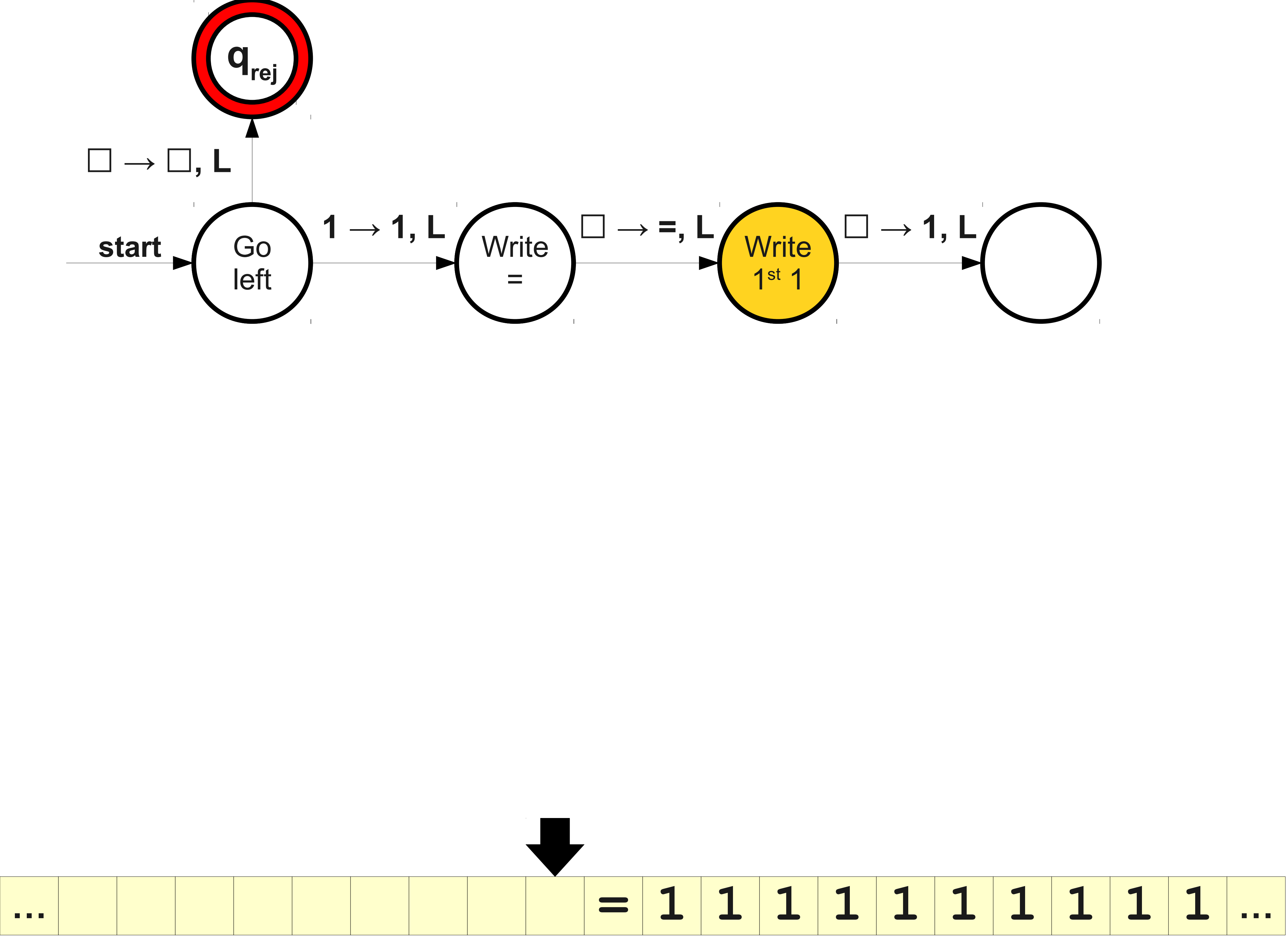
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
...
=
1
1
1
1
1
1
1
1
1
1
...



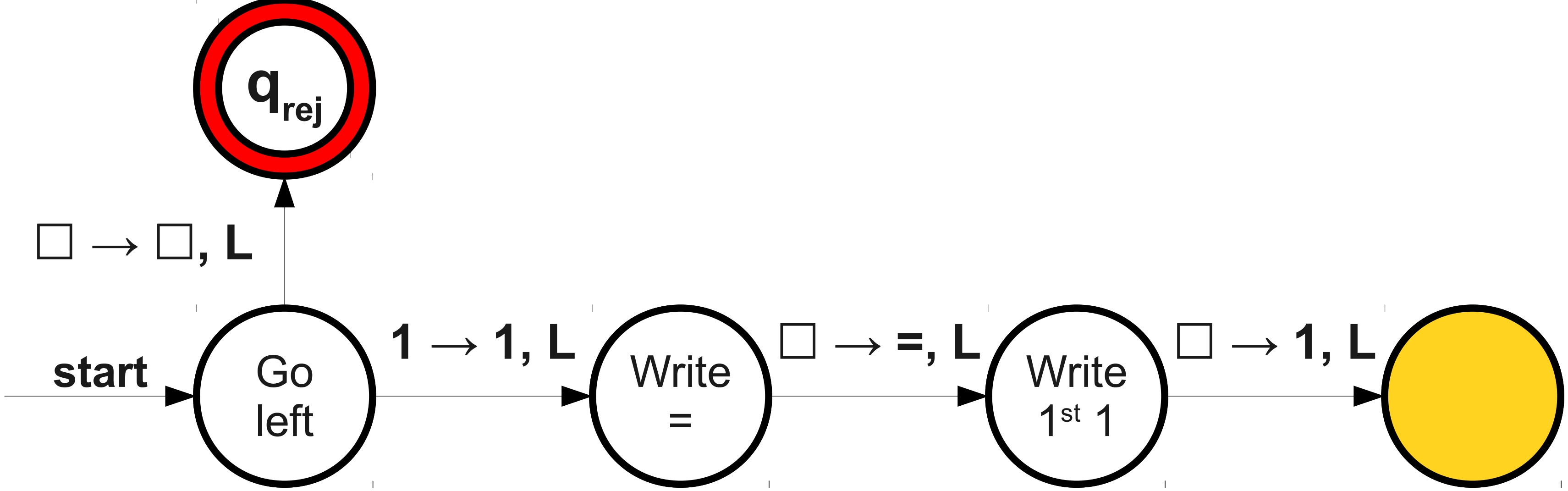
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1



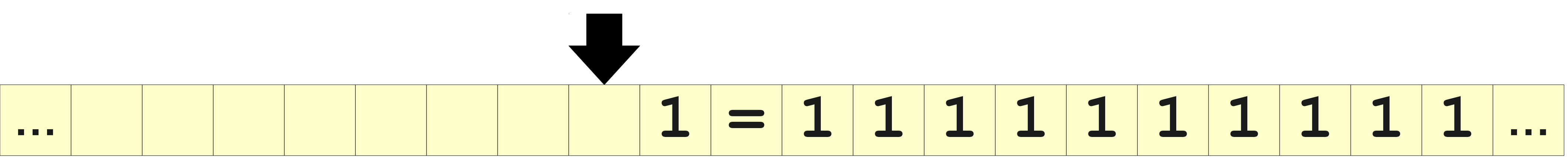
...
=
1
1
1
1
1
1
1
1
1
1
...



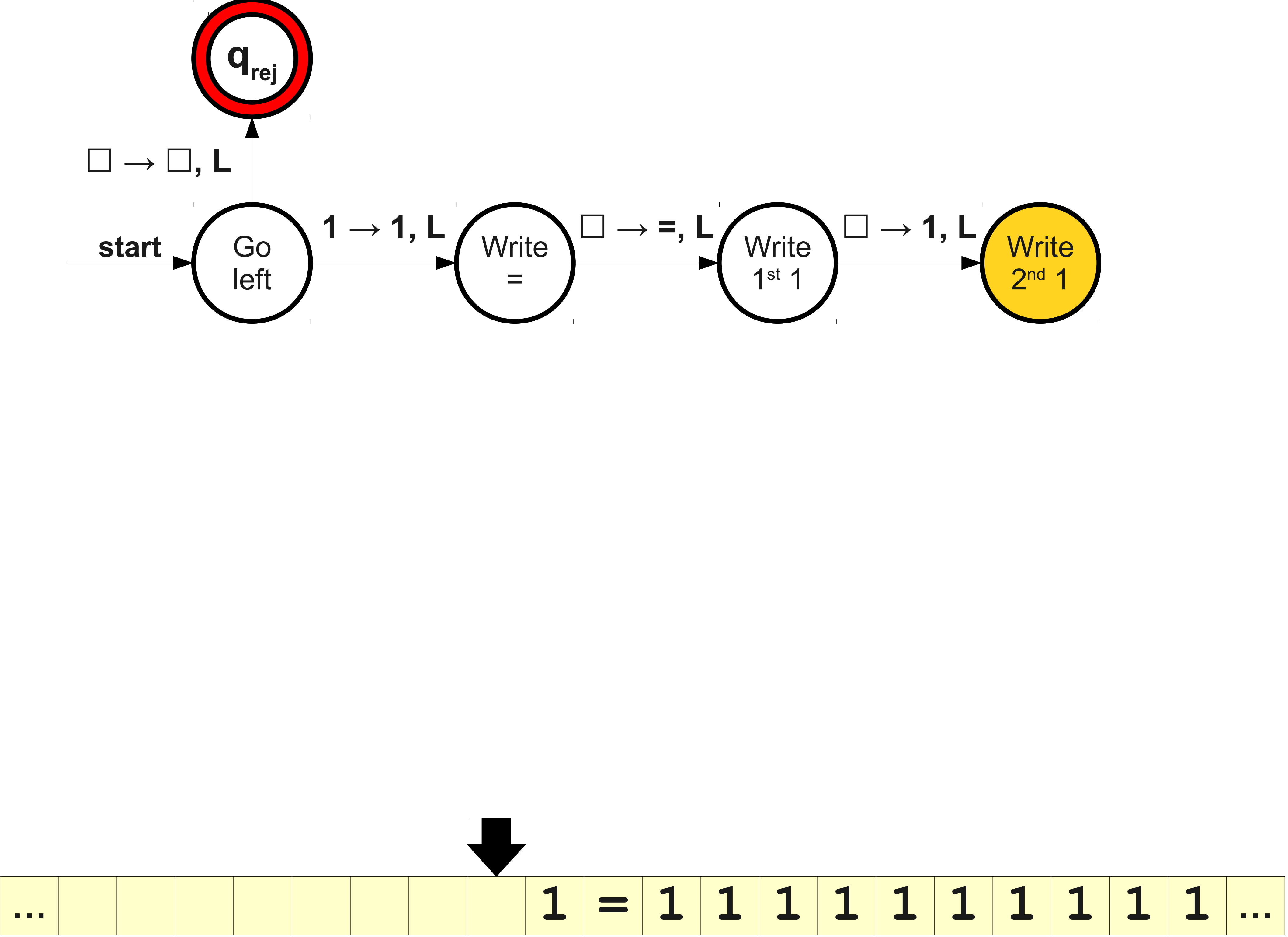
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
...
=
1 1 1 1 1 1 1 1 1 1
...



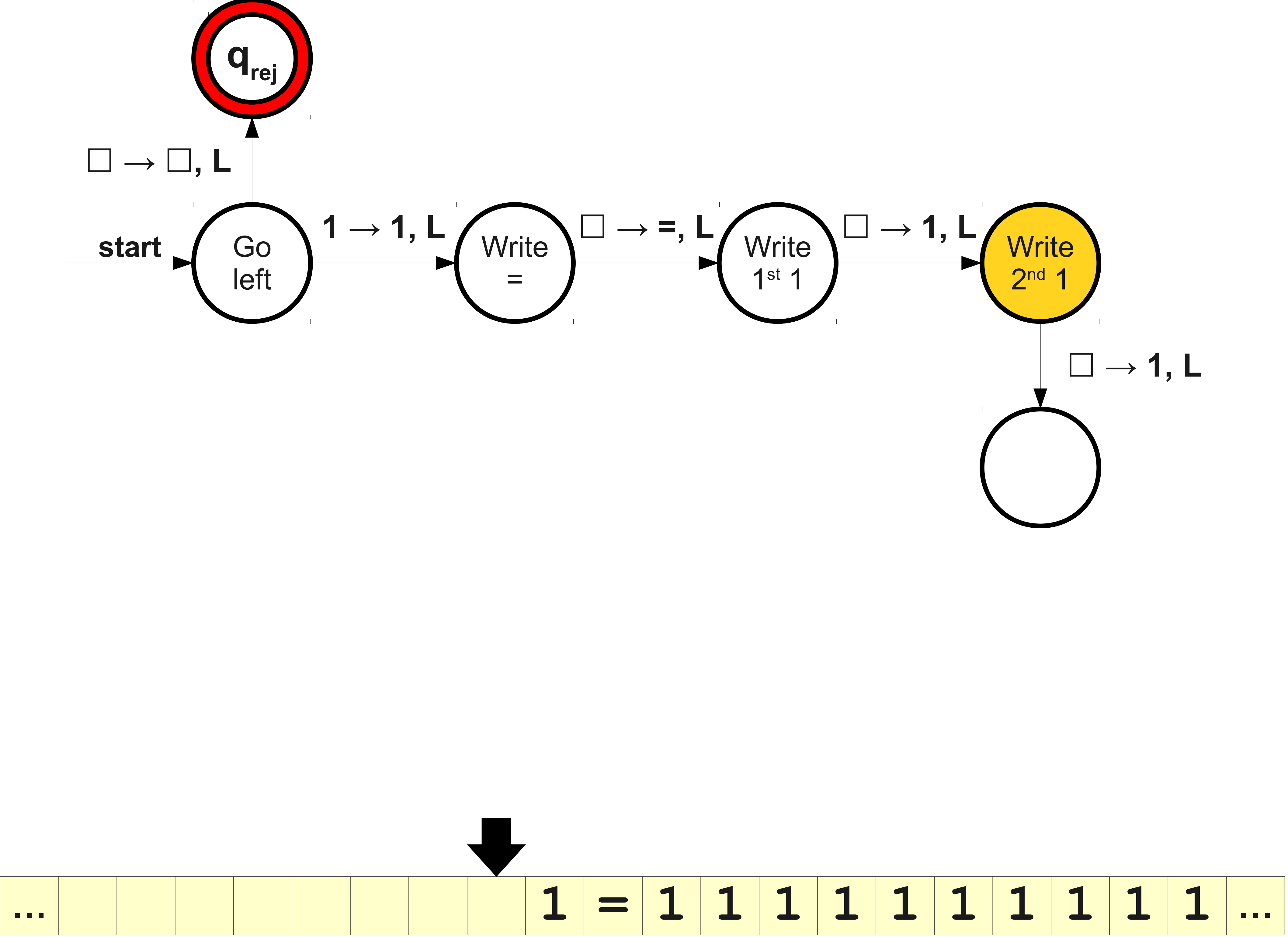
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L

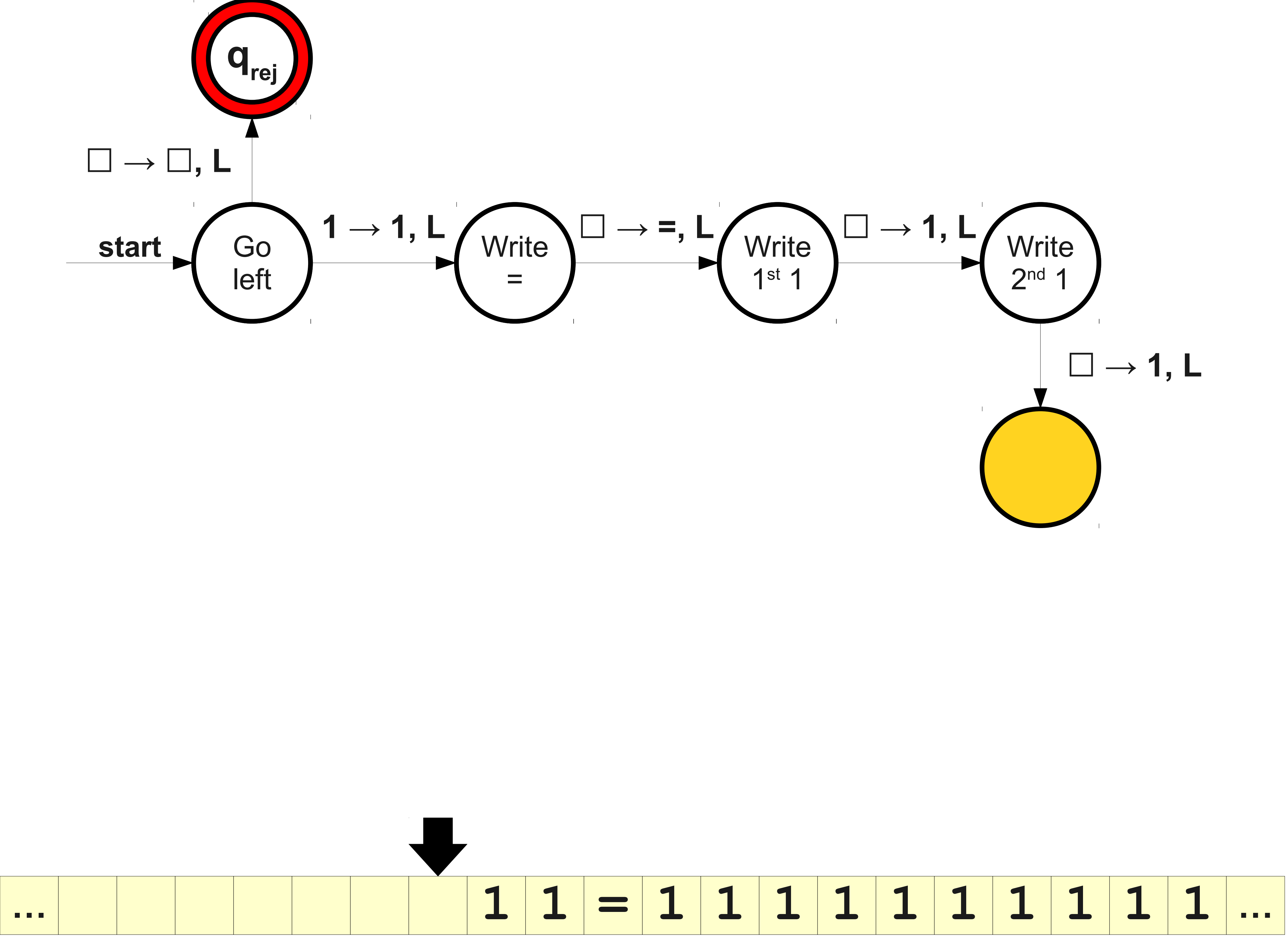


...
1
=
1
1
1
1
1
1
1
1
1
1
...

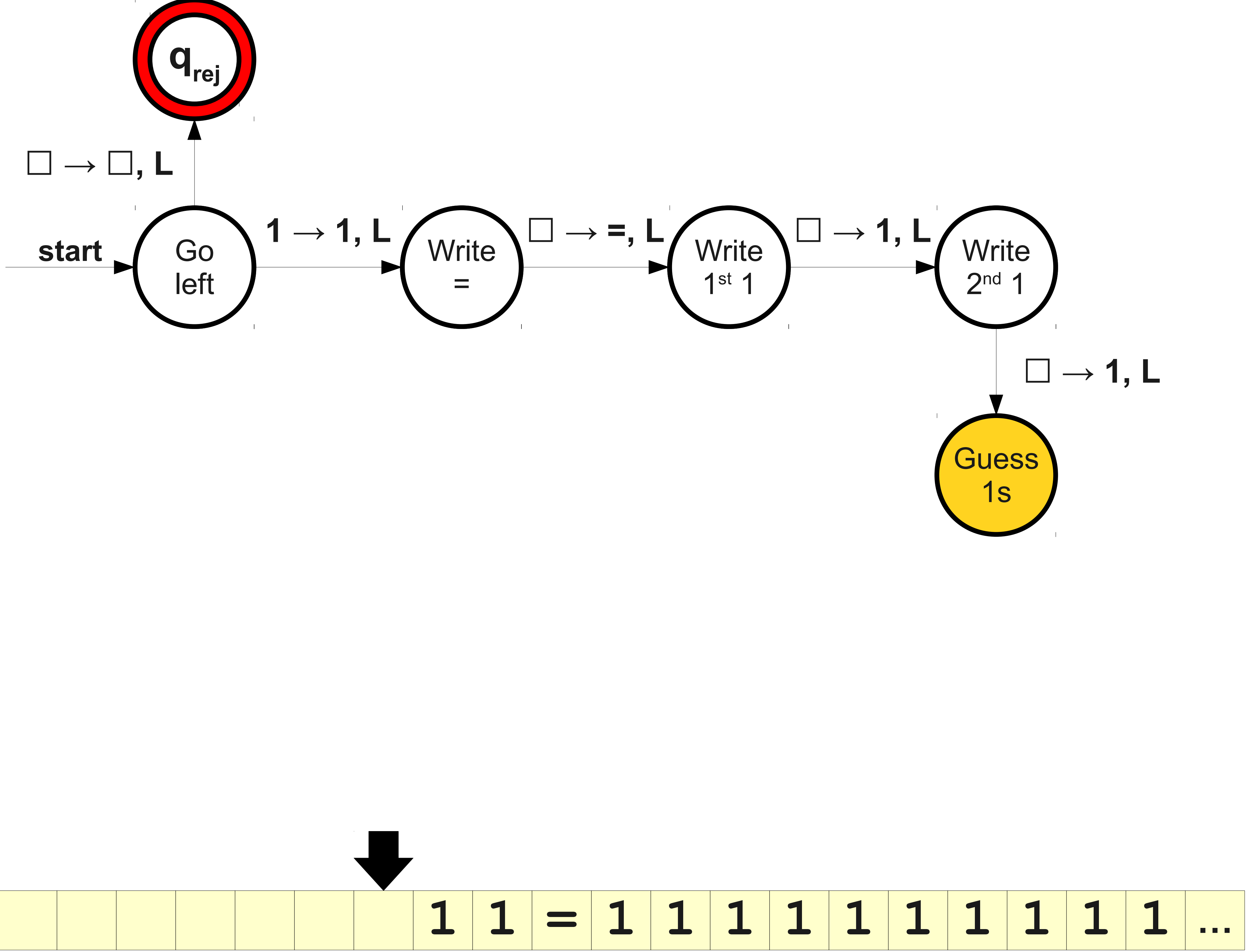


q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
... 1 = 1 1 1 1 1 1 1 1 1 1 ...

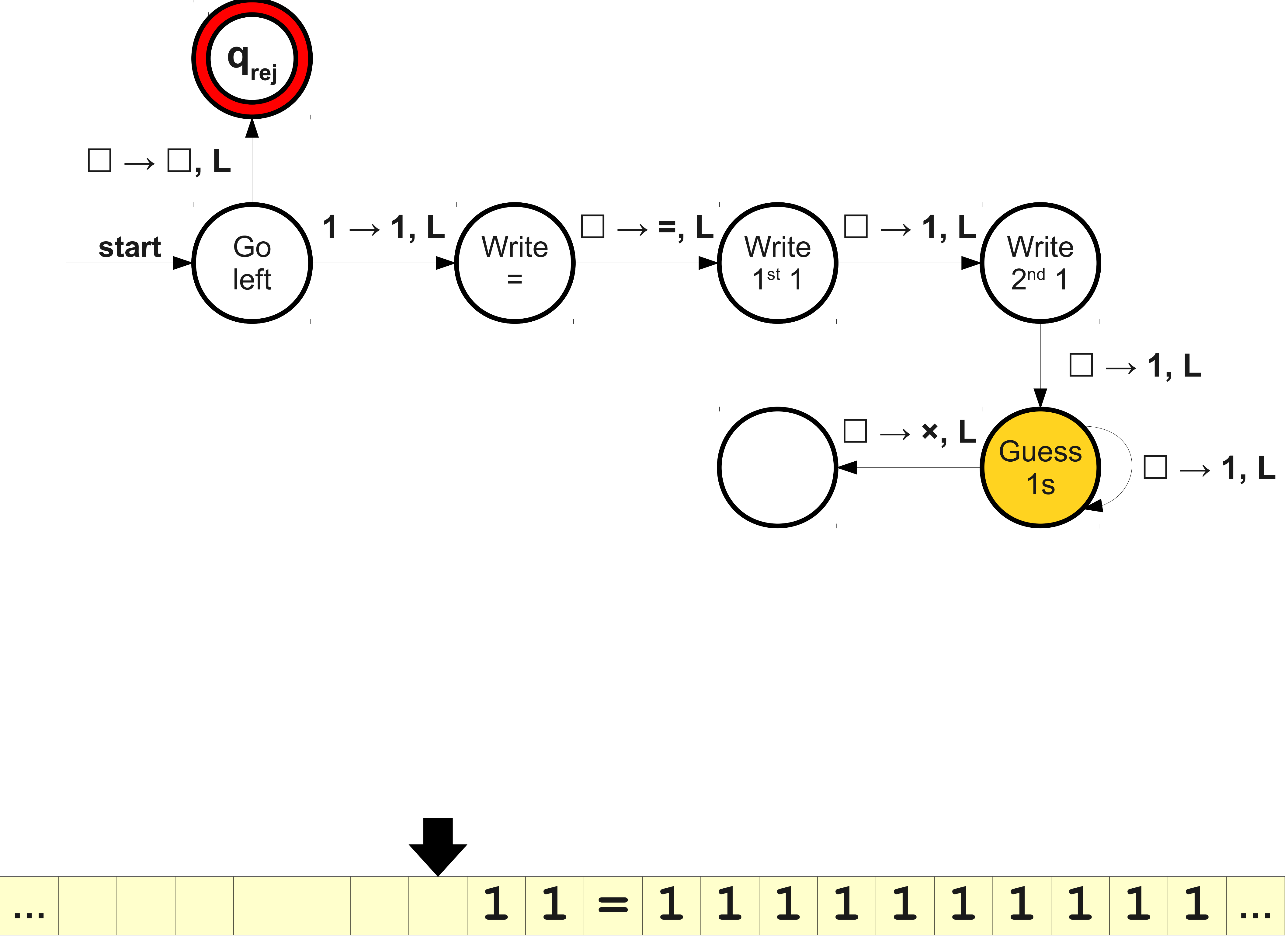

q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
...
1 = 1 1 1 1 1 1 1 1 1 1
...



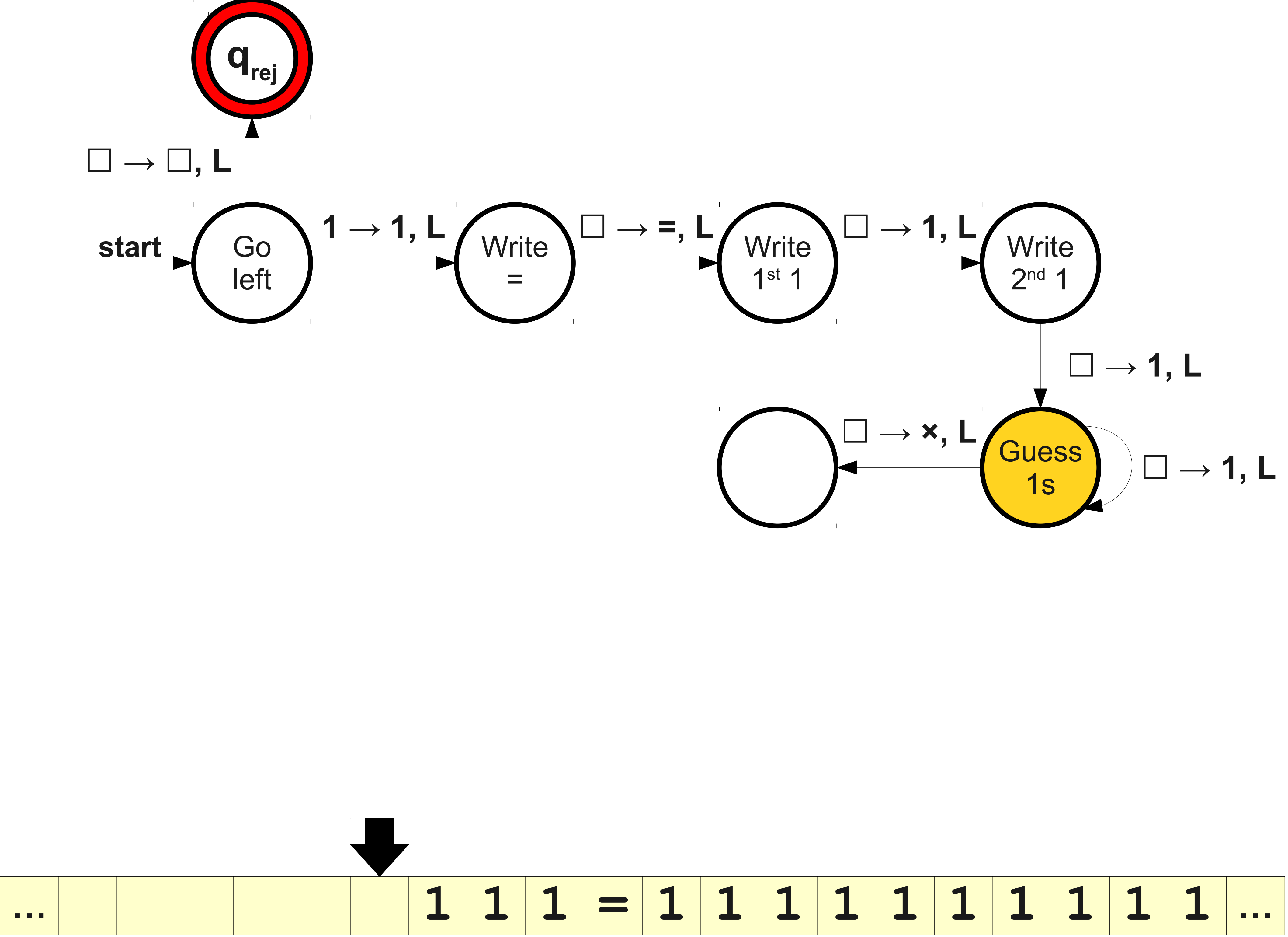
q_rej
☐ → ☐, L
start
Go left
1 → 1, L
Write =
☐ → =, L
Write 1st 1
☐ → 1, L
Write 2nd 1
☐ → 1, L
… 1 1 = 1 1 1 1 1 1 1 1 1 1 …



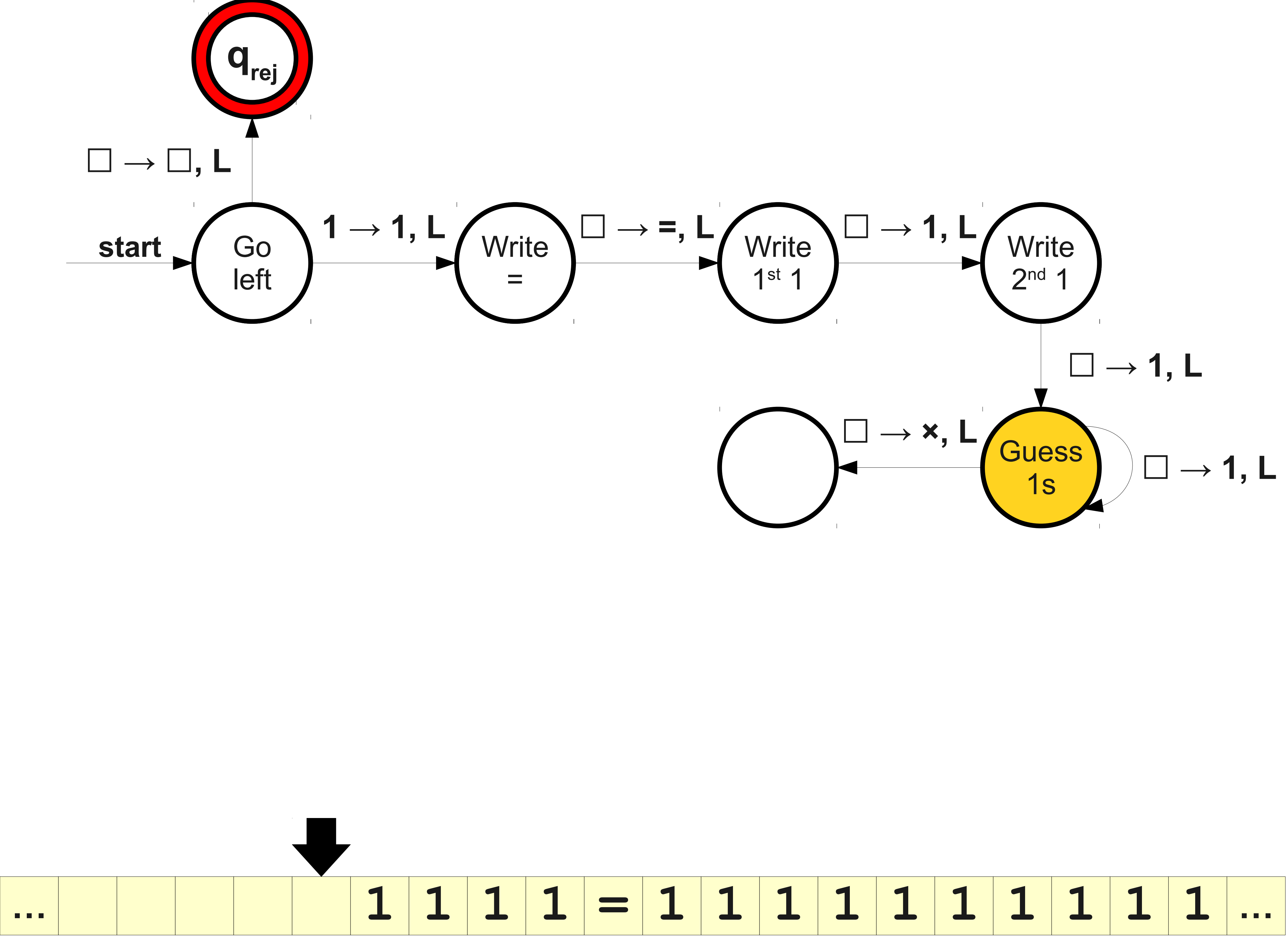
q rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
... 1 1 = 1 1 1 1 1 1 1 1 1 1 ...



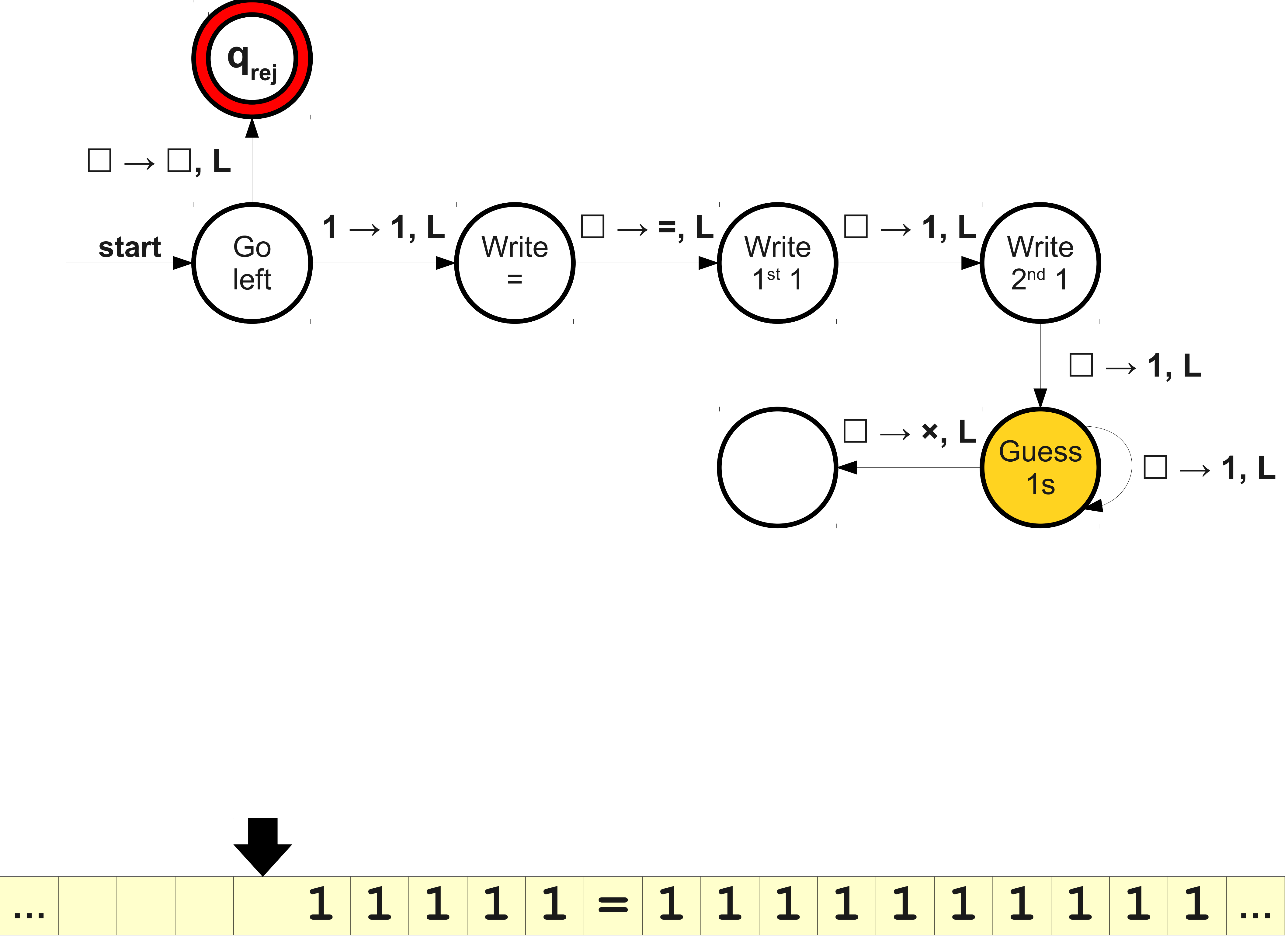
q_{rej}
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
□ → ×, L
Guess 1s
□ → 1, L
... 1 1 = 1 1 1 1 1 1 1 1 1 1 ...



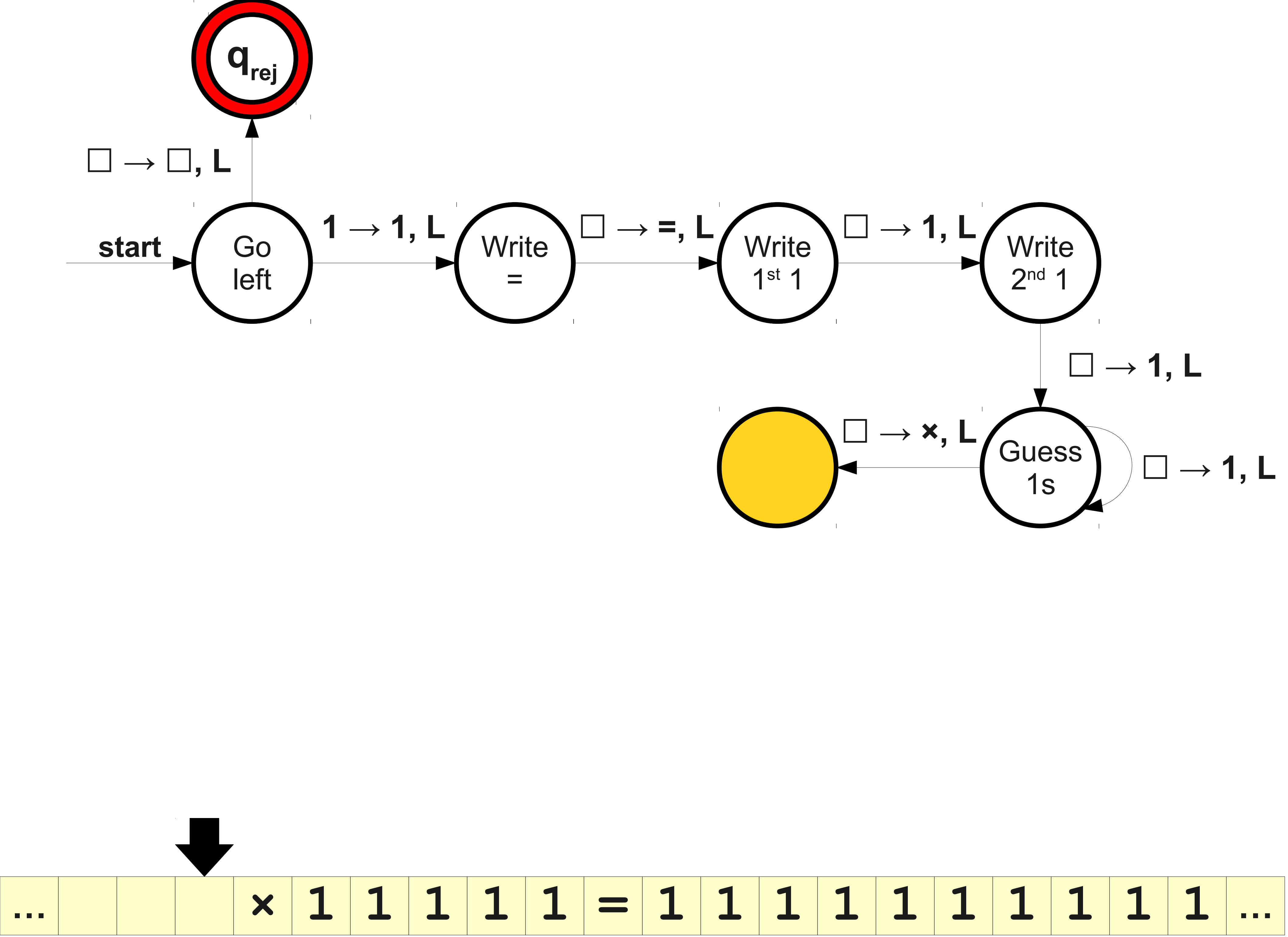
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
...
1 1 1 = 1 1 1 1 1 1 1 1 1 1
...

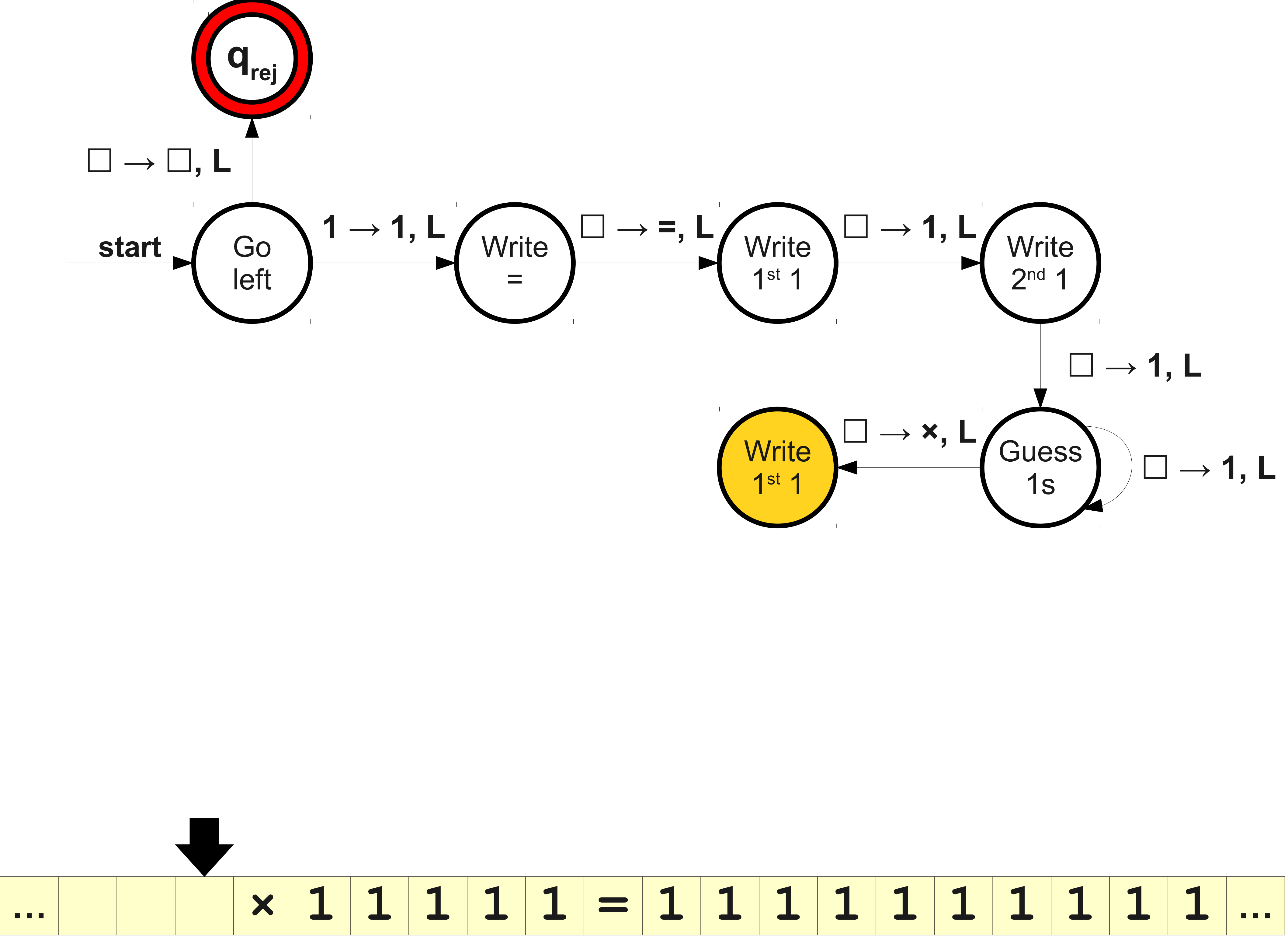


q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
□ → ×, L
Guess 1s
□ → 1, L
… 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …

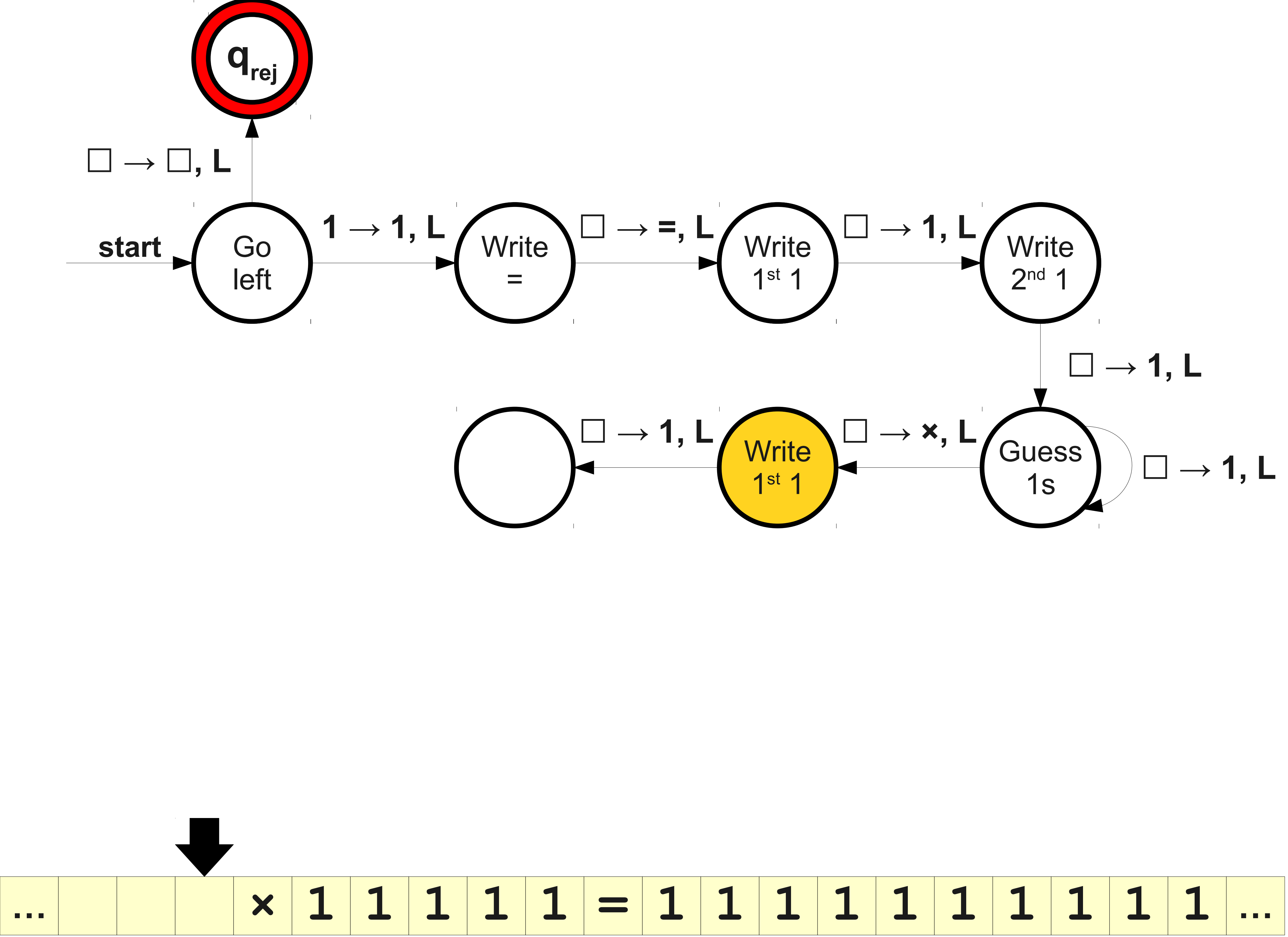


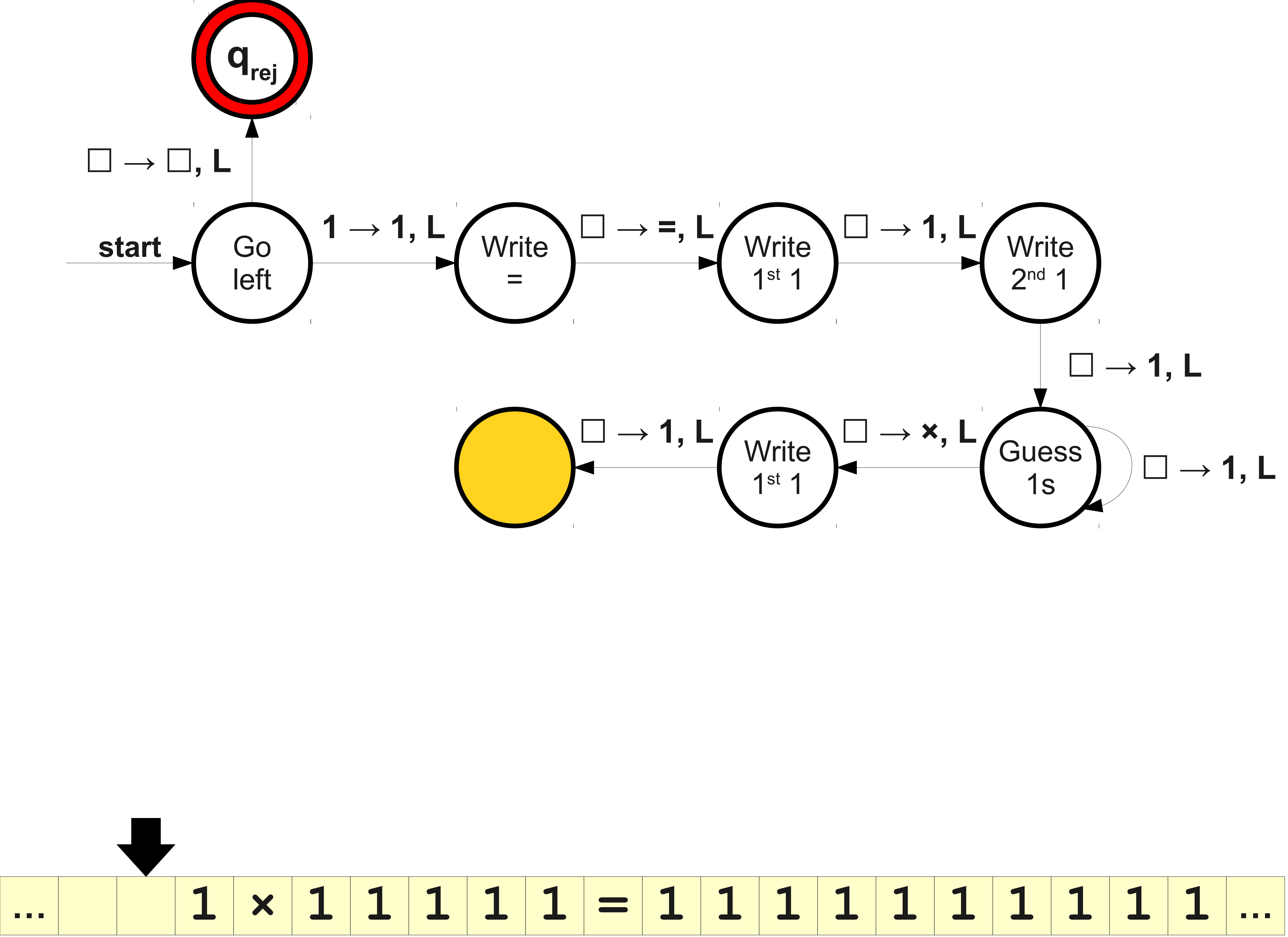
q_{rej}
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
… 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …

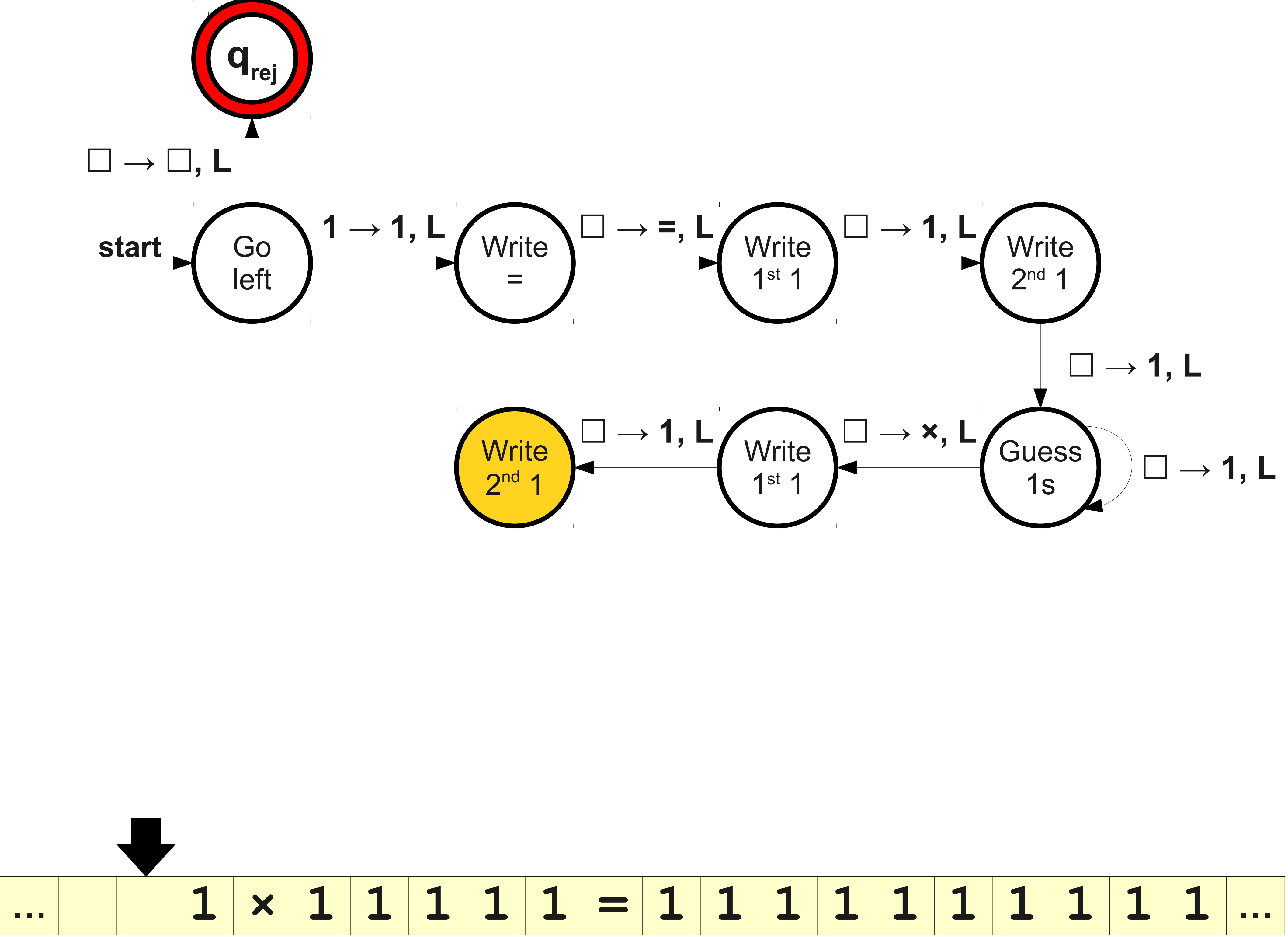

q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
... × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 ...



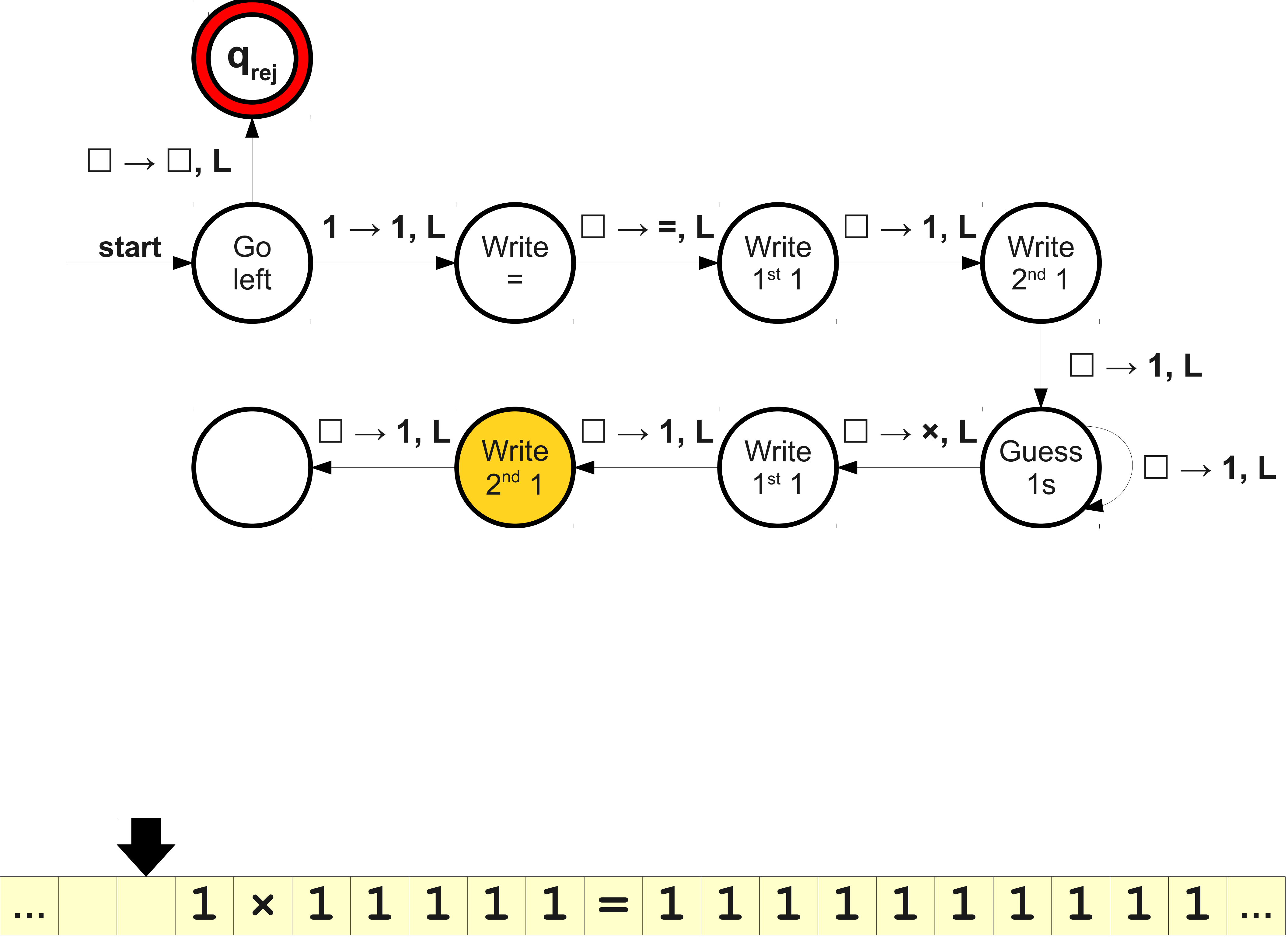
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
… × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …


q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
...
×
1
1
1
1
1
=
1
1
1
1
1
1
1
1
1
1
...

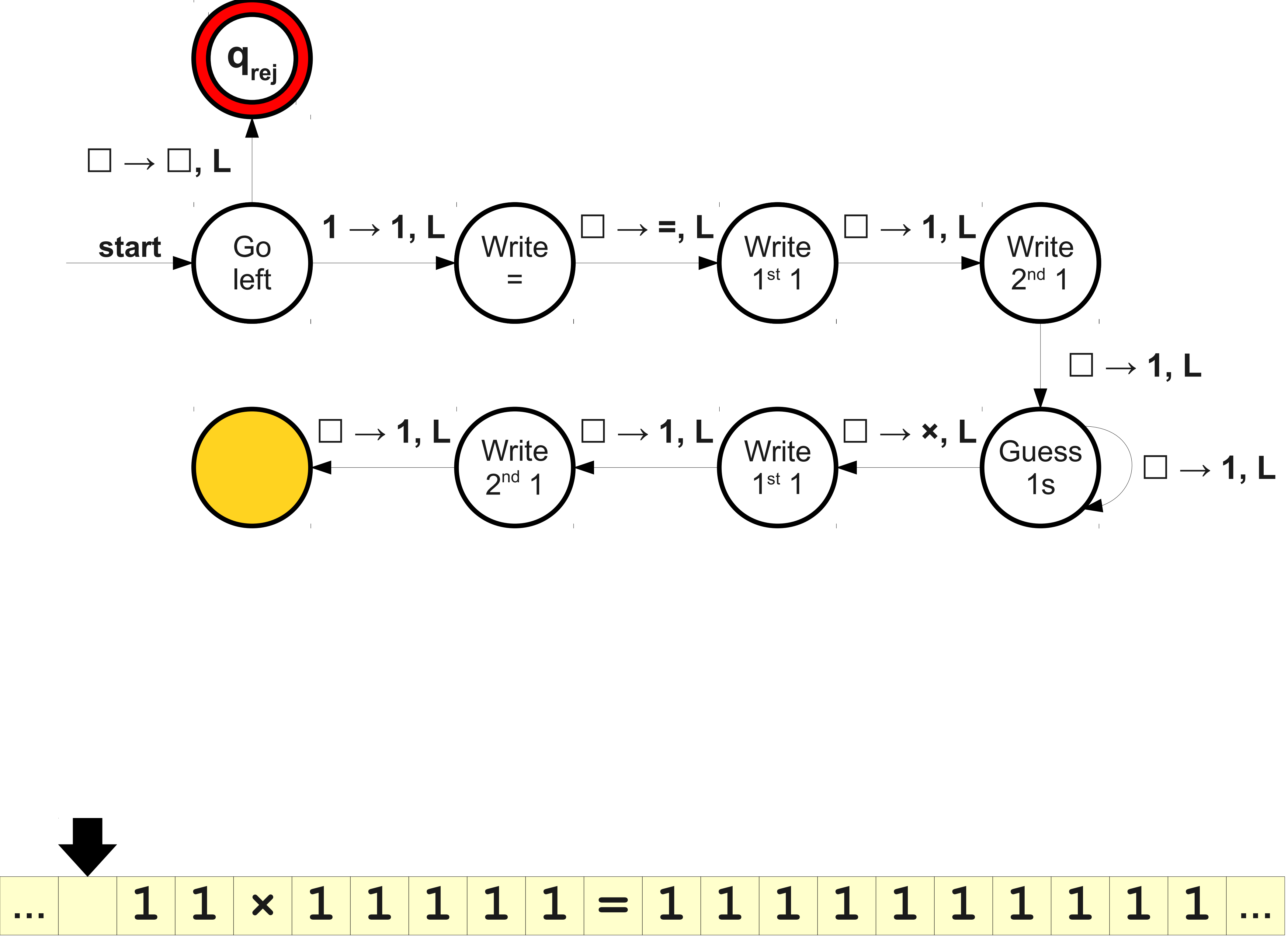

q rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
... 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 ...



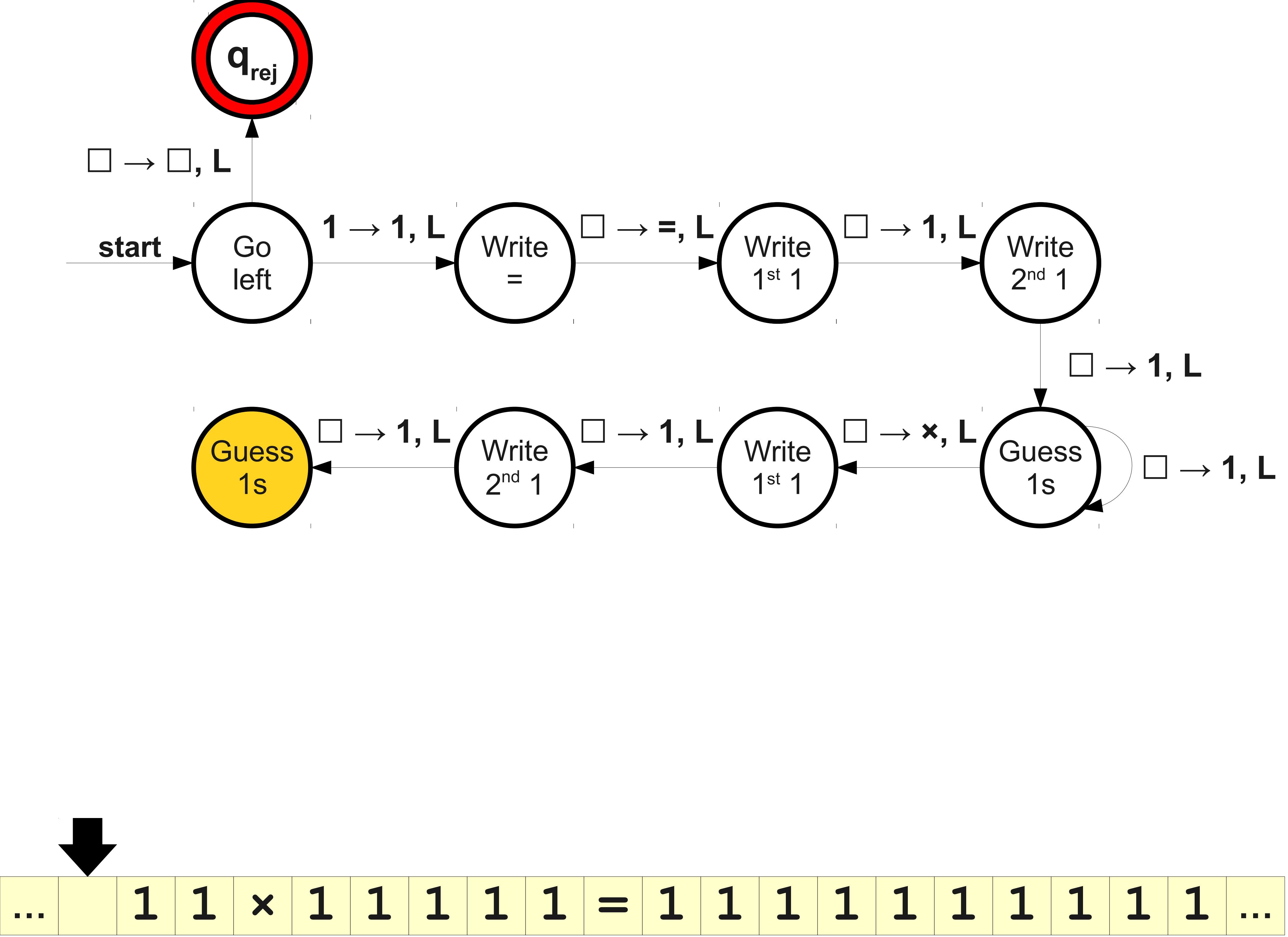
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
… 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …

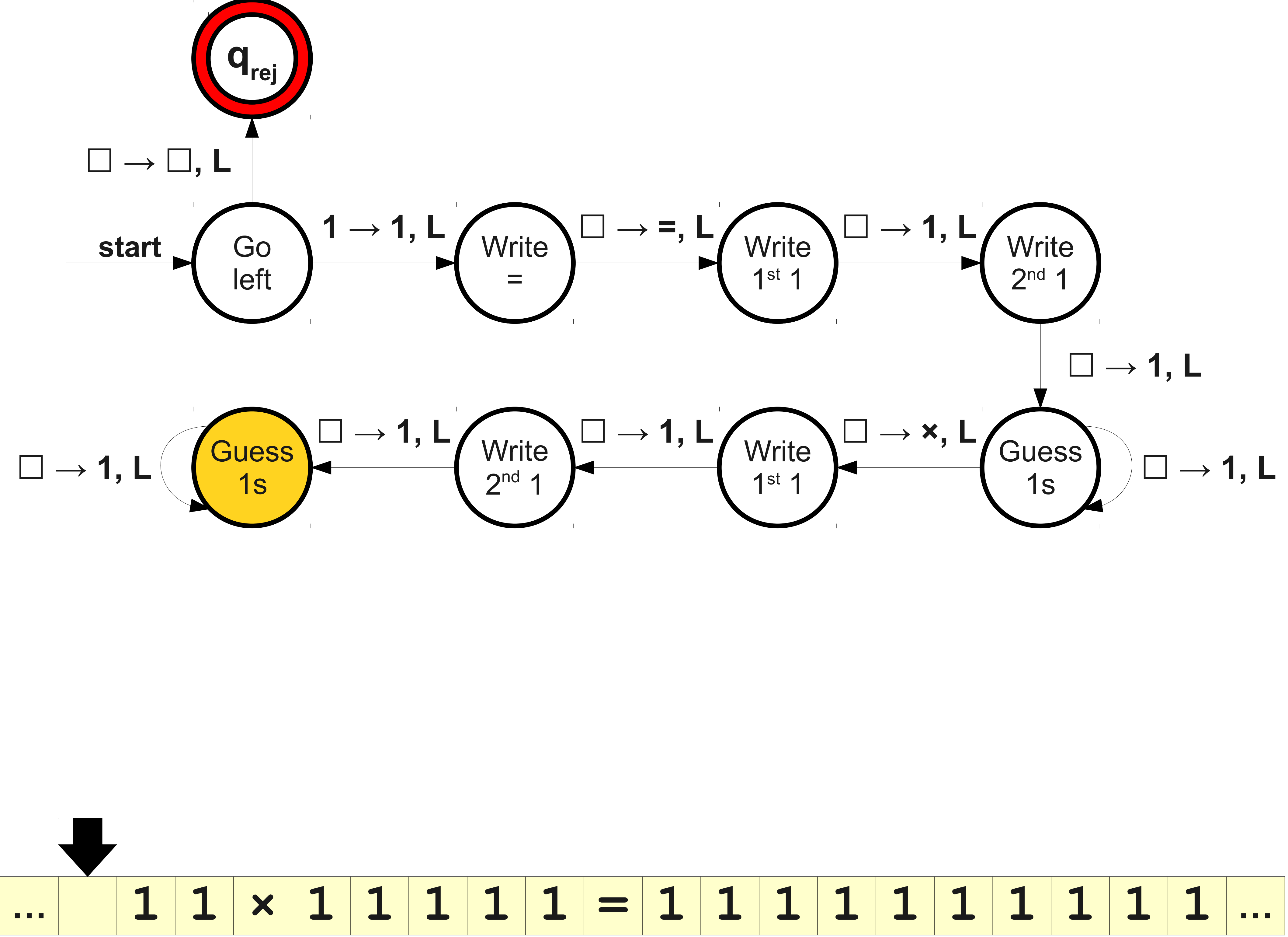


q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
… 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …

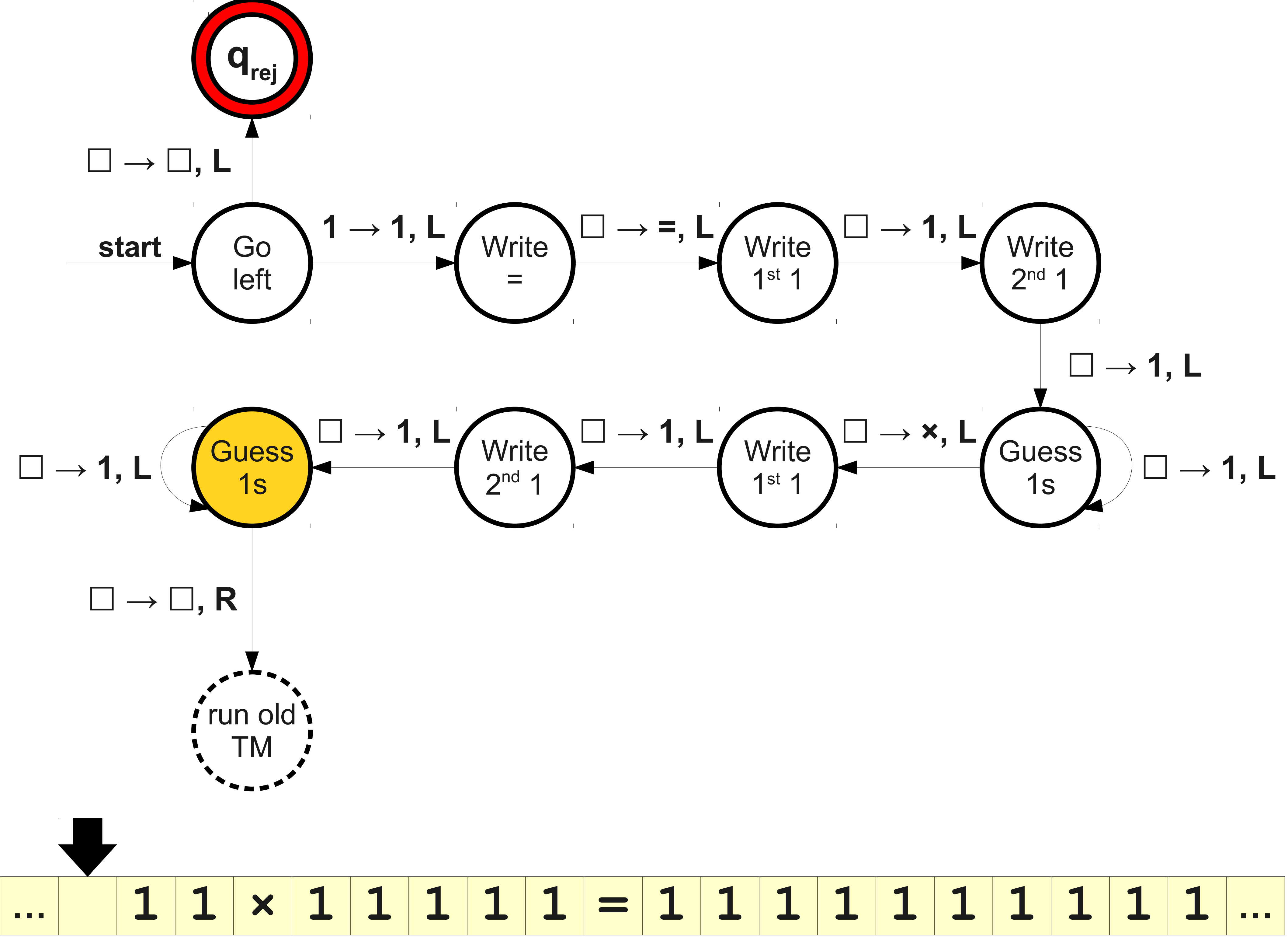


q rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
... 1 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 ...

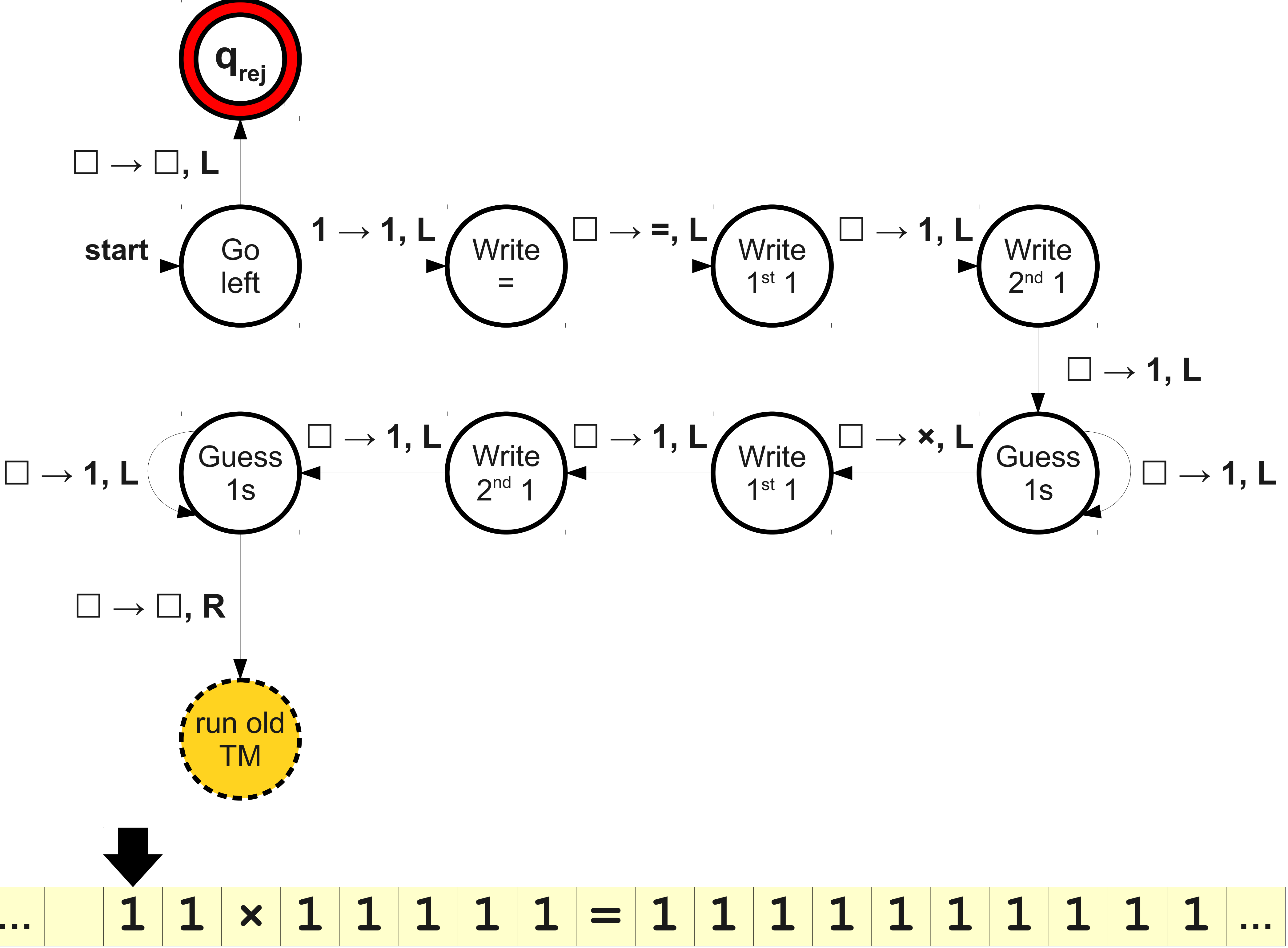

q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
... 1 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 ...



q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
… 1 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …



q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
… 1 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 …

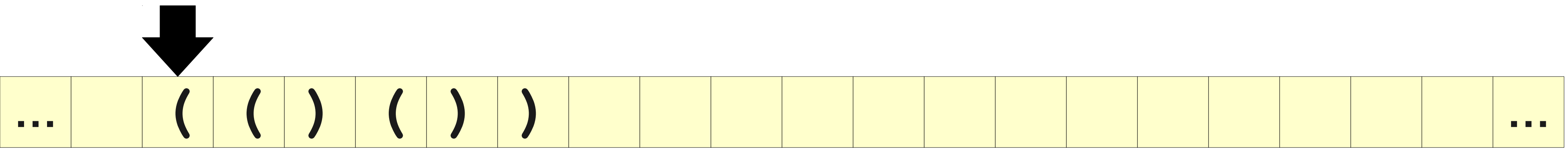


q_{rej}
$\square \rightarrow \square$, L
start
Go left
1 → 1, L
Write =
$\square \rightarrow$ =, L
Write 1^{st} 1
$\square \rightarrow$ 1, L
Write 2^{nd} 1
$\square \rightarrow$ 1, L
Guess 1s
$\square \rightarrow$ 1, L
$\square \rightarrow$ ×, L
Write 1^{st} 1
$\square \rightarrow$ 1, L
Write 2^{nd} 1
$\square \rightarrow$ 1, L
Guess 1s
$\square \rightarrow$ 1, L
$\square \rightarrow \square$, R
run old TM
... 1 1 × 1 1 1 1 1 = 1 1 1 1 1 1 1 1 1 1 ...

Designing NTMs

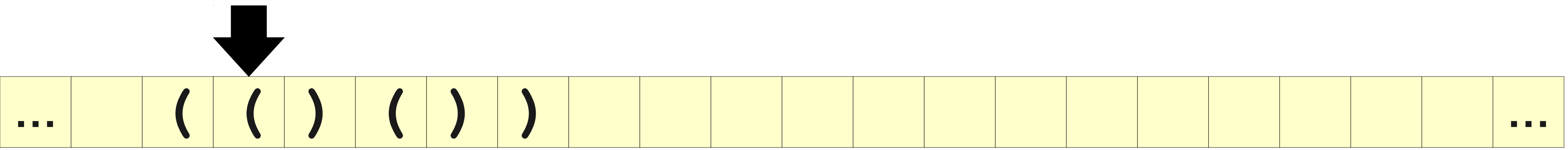
- Suppose that we have a CFG G.
- Can we build a TM M where $\mathscr{L}(M) = \mathscr{L}(G)$?
- **Idea:** Nondeterministically guess which productions ought to be applied.
  - Keep the original string on the input tape.
  - Keep guessing productions until no nonterminals remain.
  - Accept if the resulting string matches.

A Sketch of the Construction



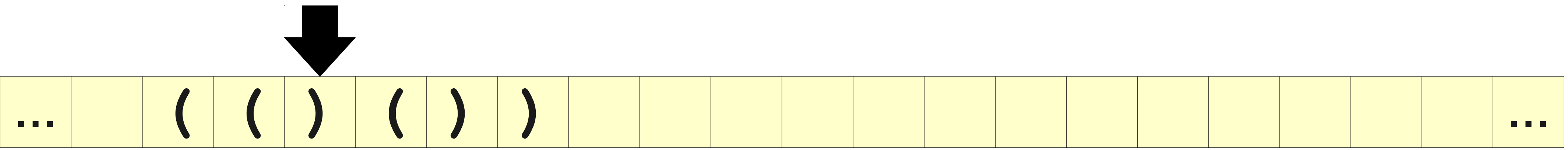
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



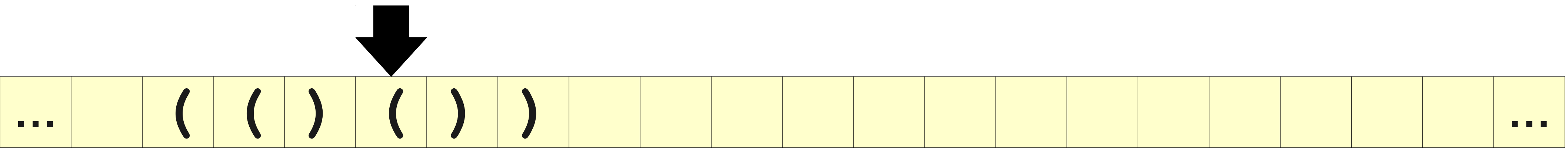
$$\mathbf{S} \to \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



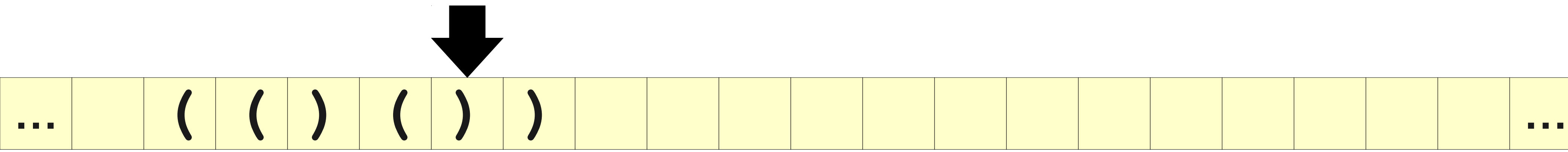
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



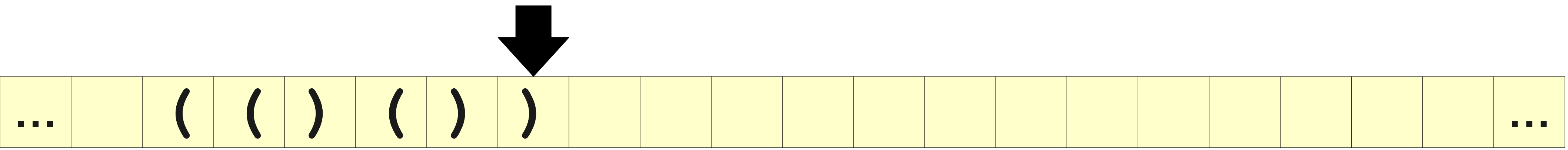
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



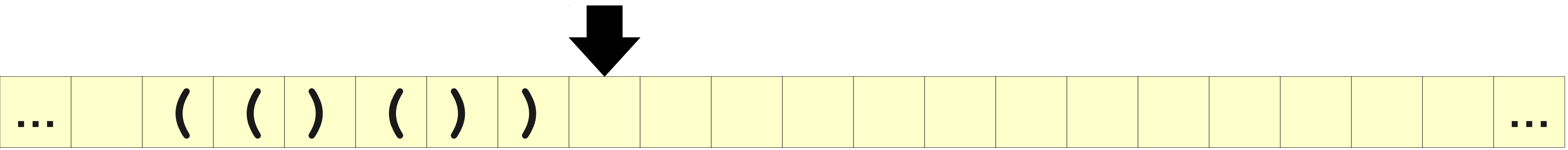
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



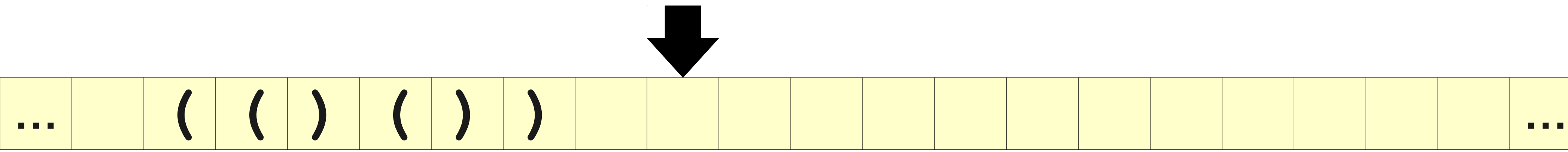
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



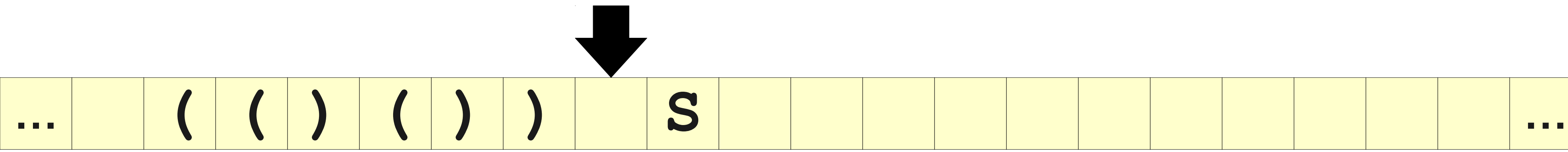
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



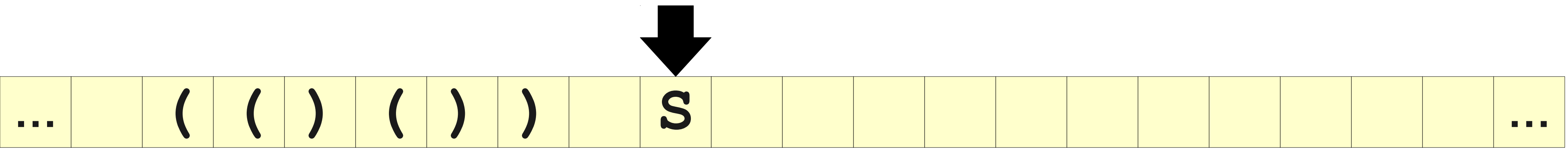
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



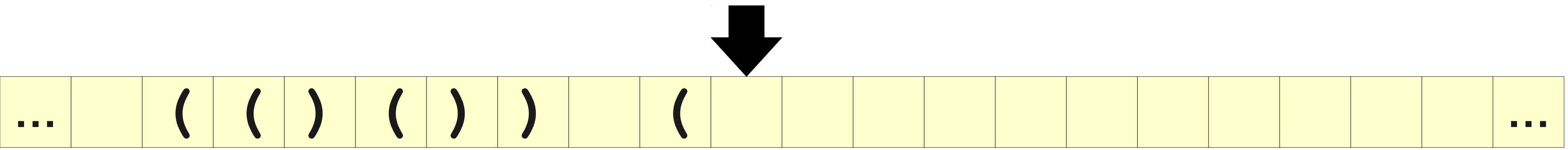
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



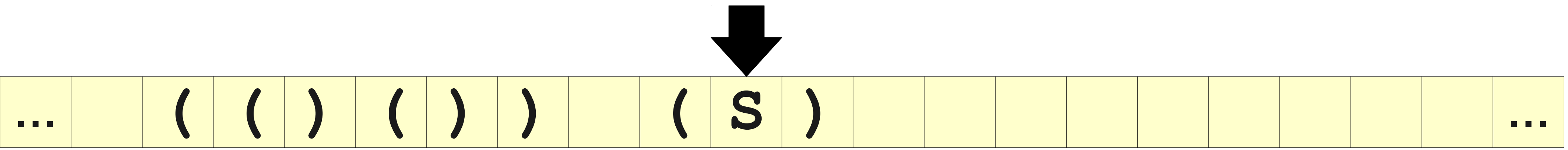
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	S											…

$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



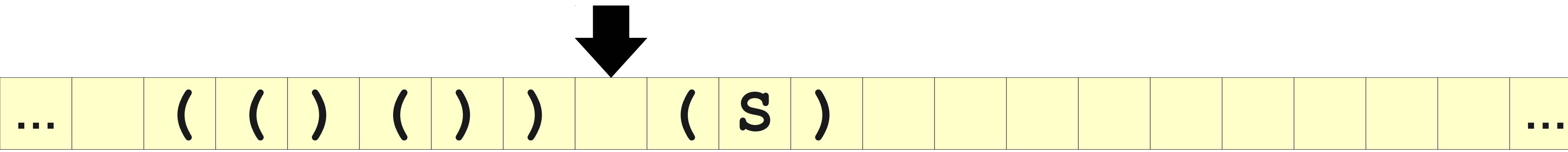
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	S	)										…

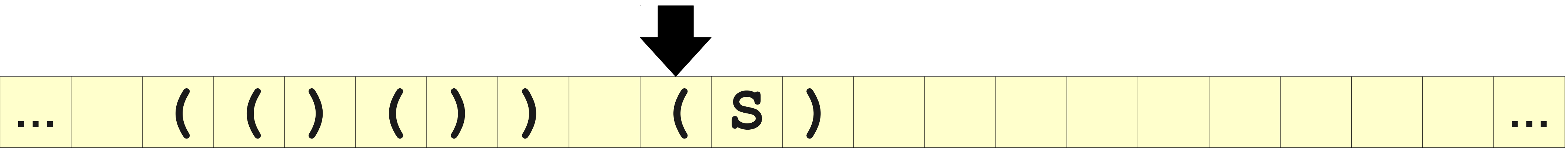
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



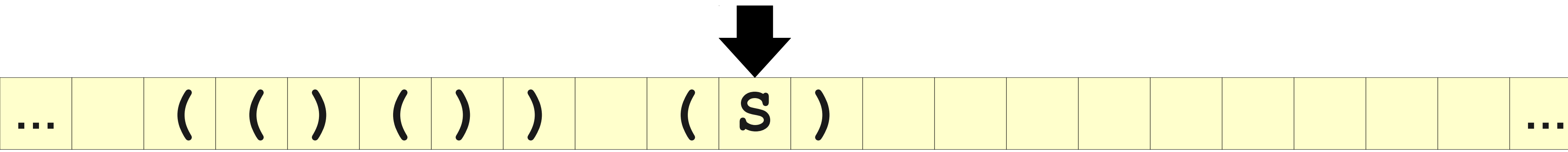
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



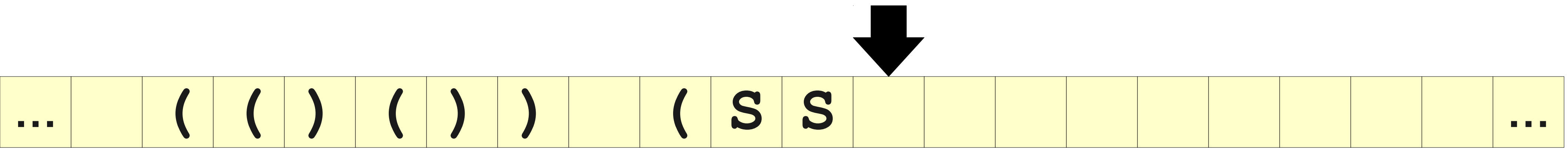
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	S	)										…

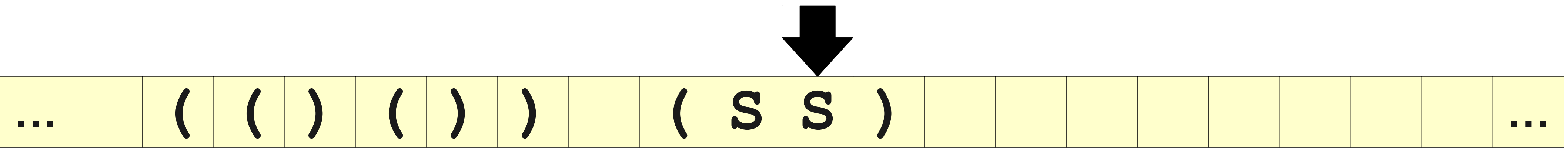
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



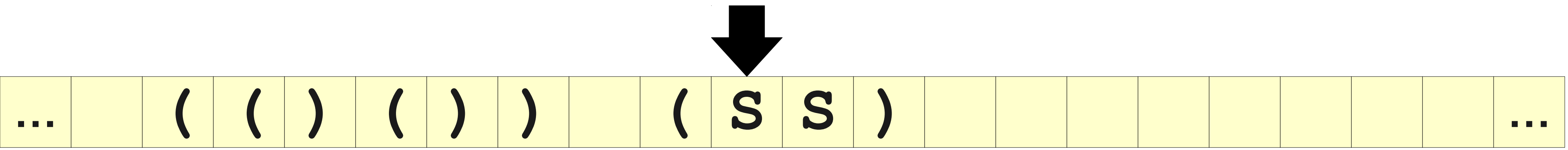
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



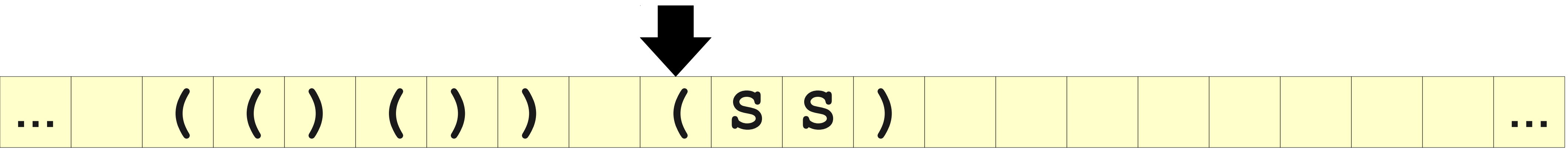
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



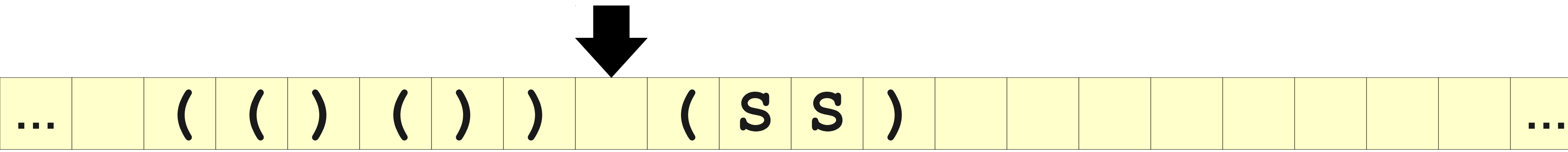
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



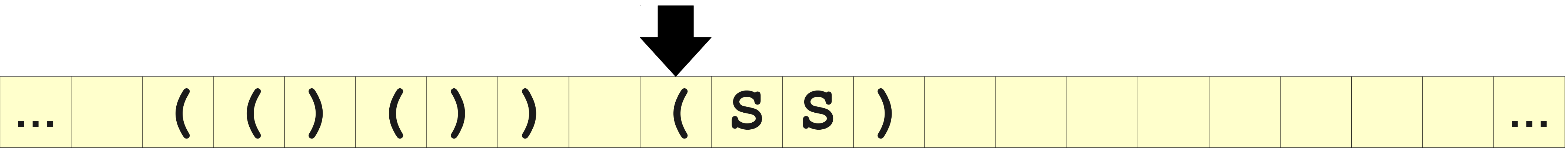
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



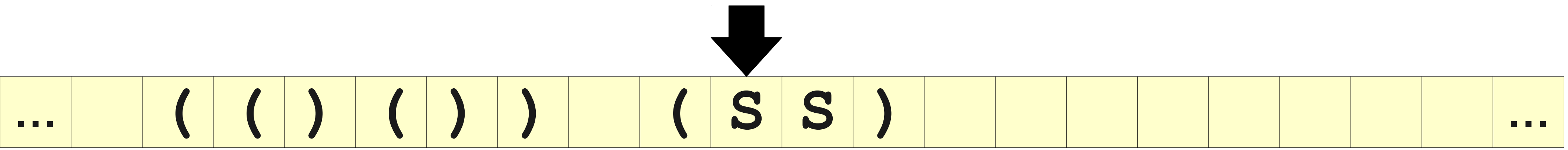
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



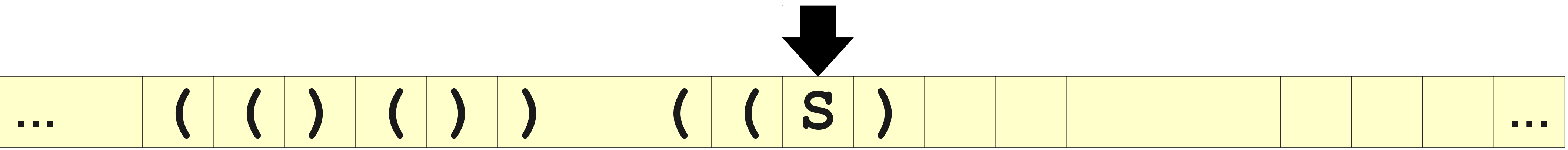
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



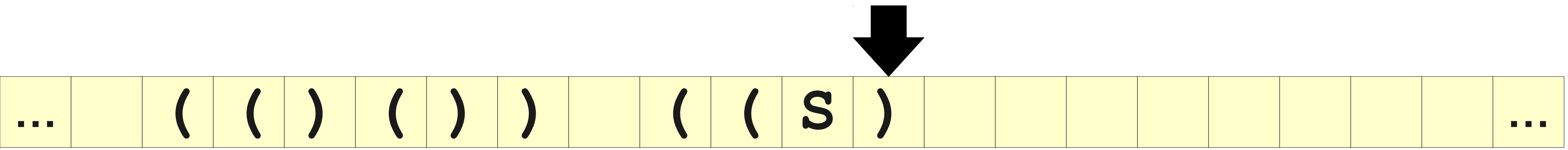
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



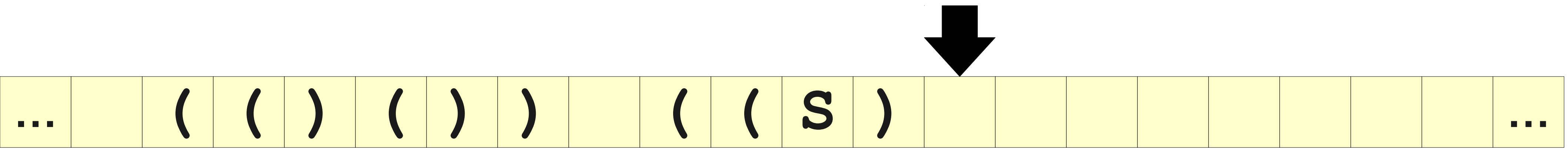
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \to \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	S	)	S								…

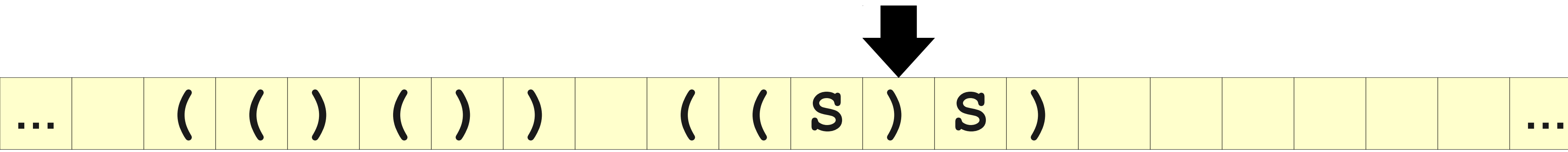
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	S	)	S	)							…

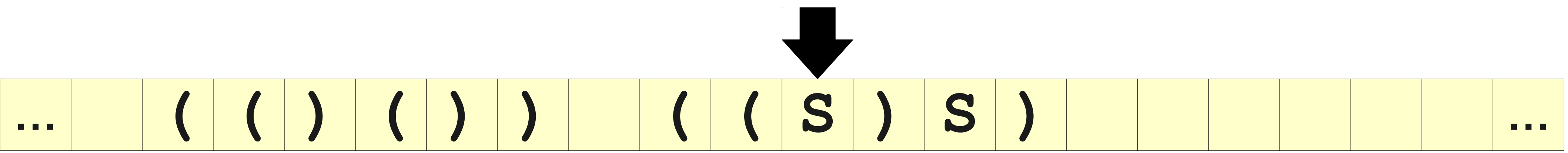
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



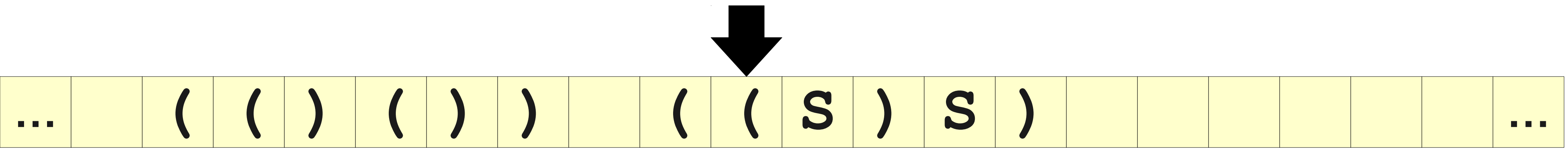
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



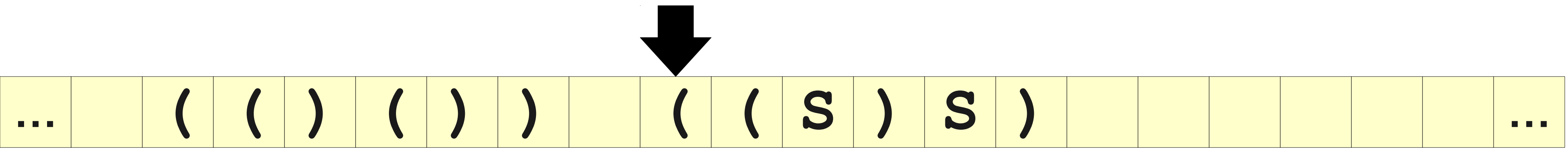
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



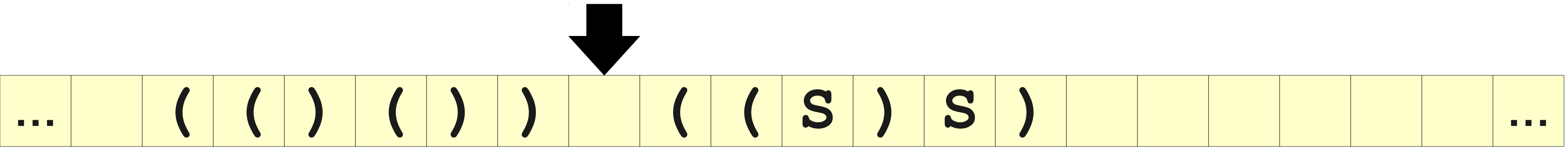
$$\mathbf{S} \to \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



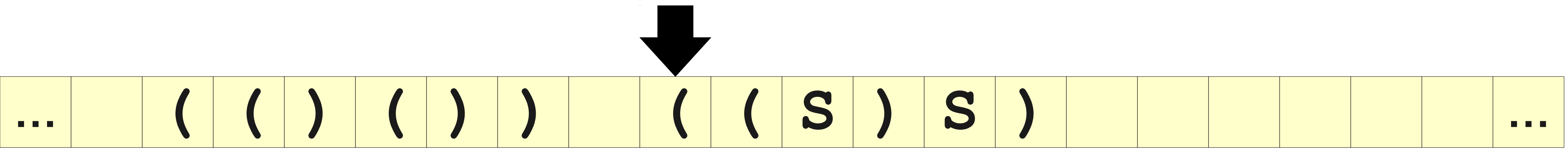
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



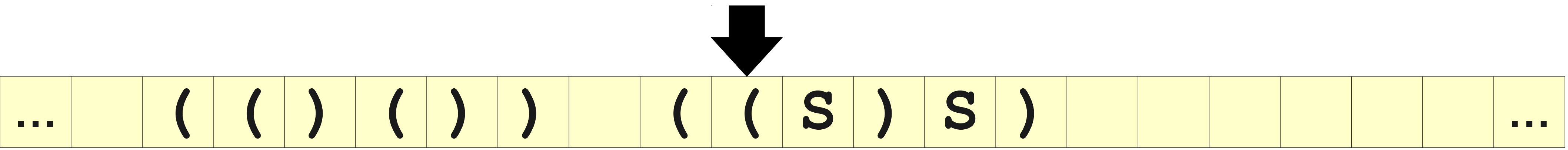
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



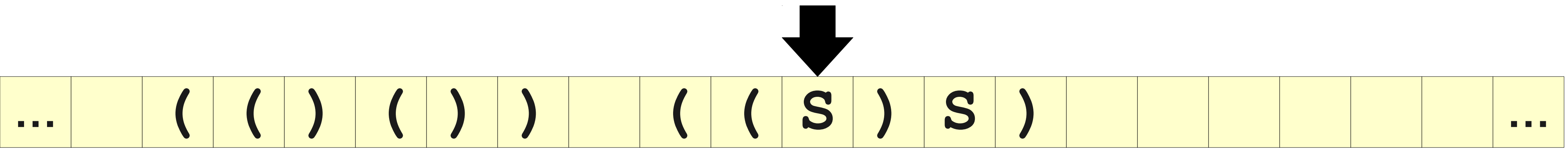
S → SS | (S) | ε

A Sketch of the Construction



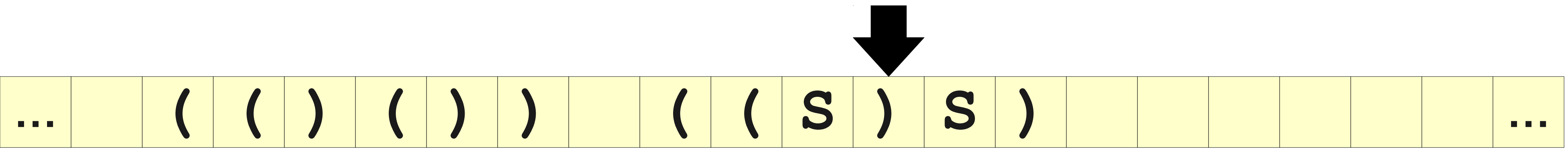
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



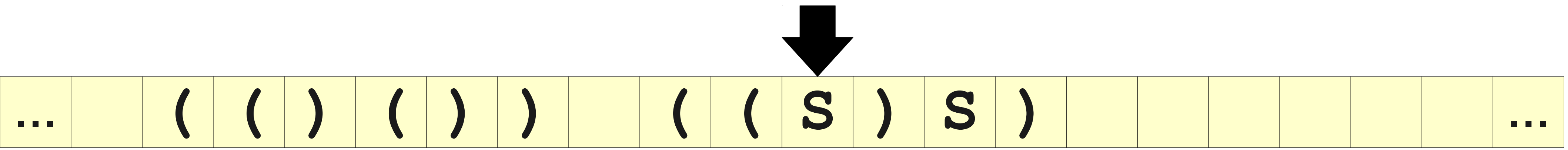
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



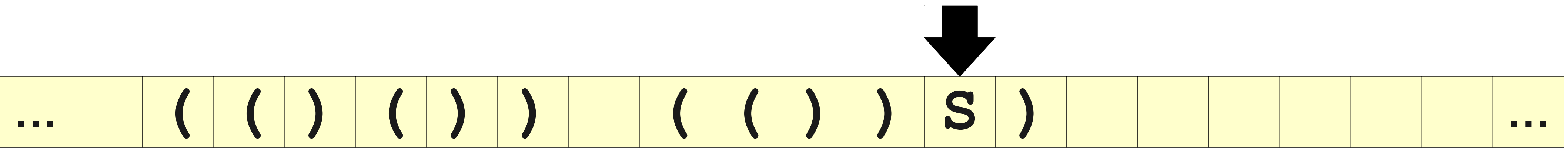
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction

...		(	(	)	(	)	)		(	(	)	)	S	)							...

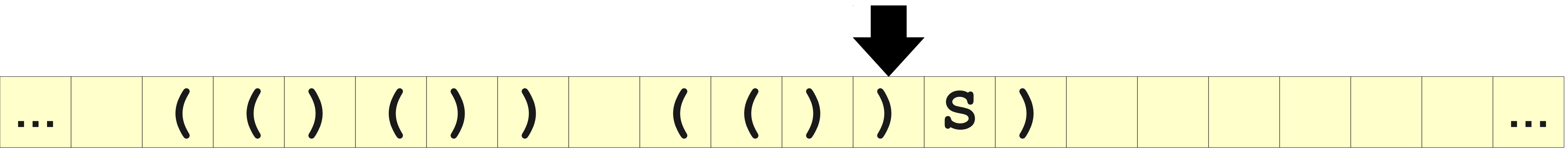
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



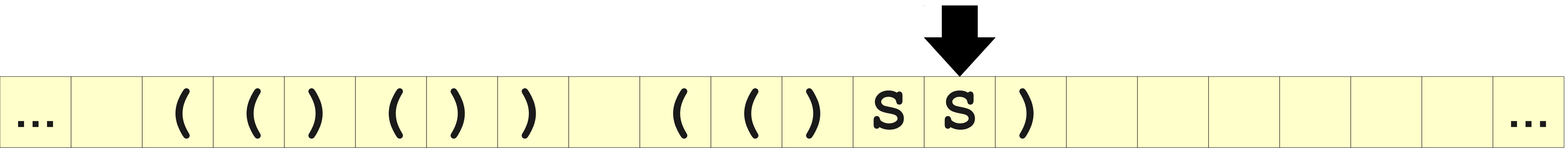
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



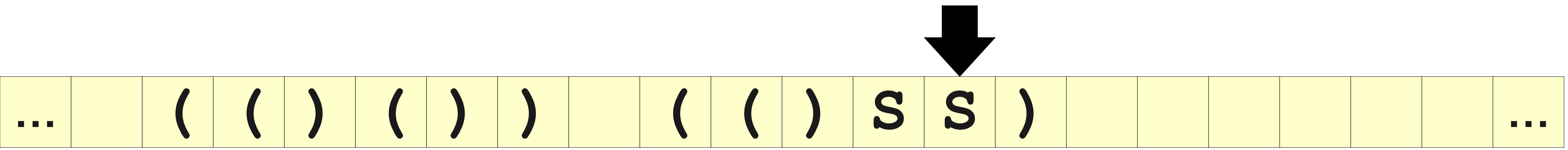
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	)	S	S	)							…

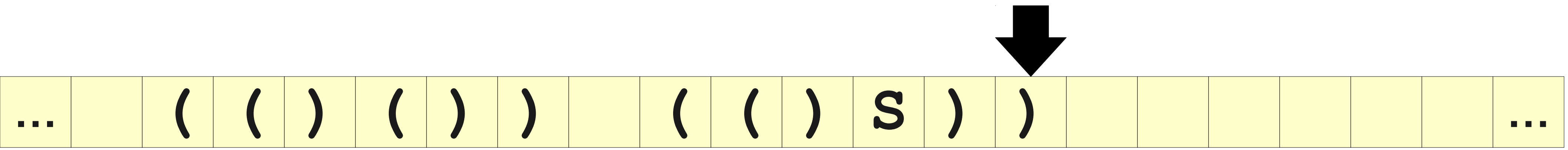
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



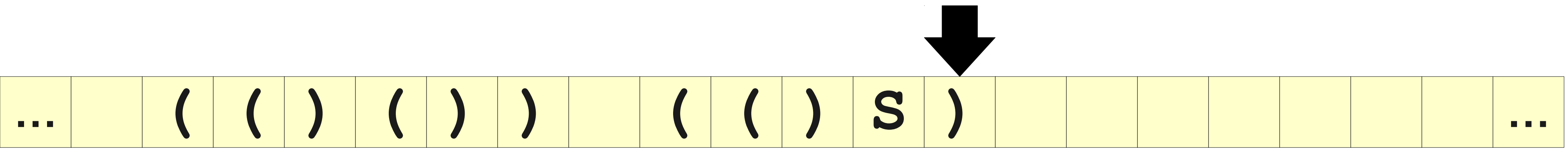
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



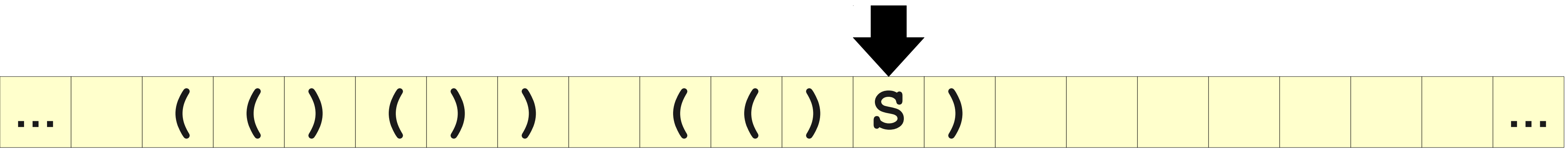
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



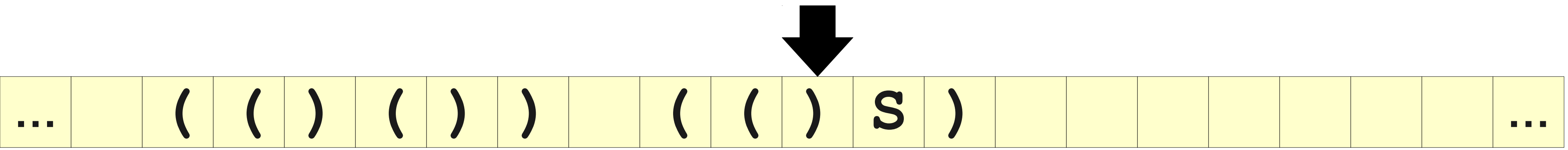
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



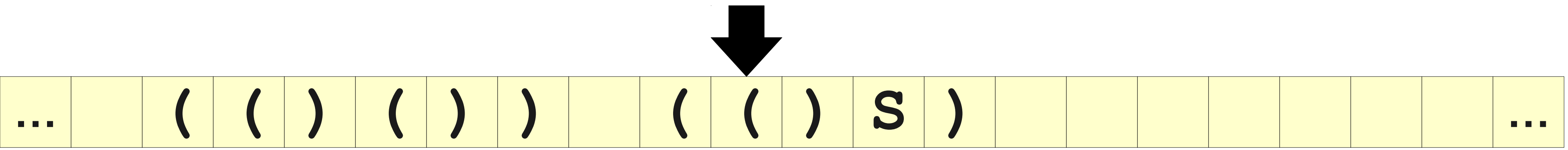
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



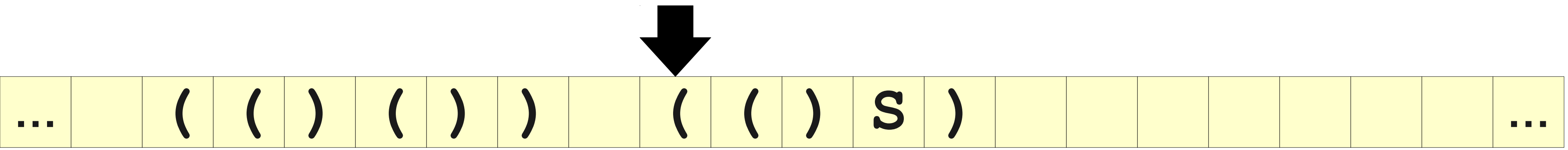
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



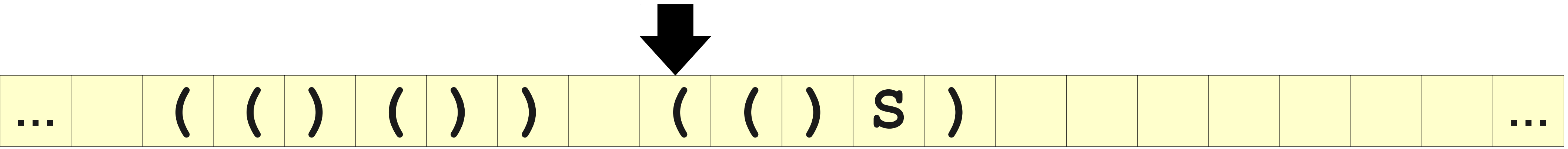
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	)	S	)								…

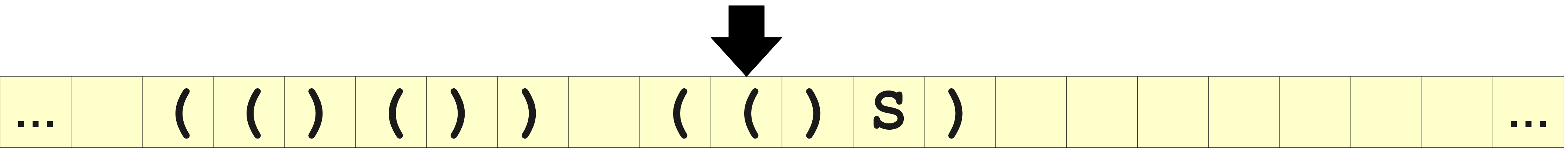
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



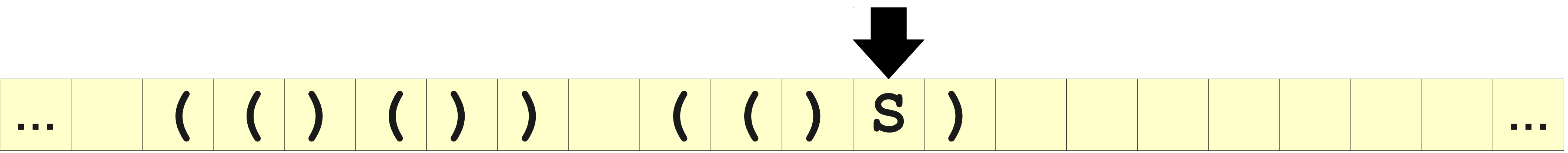
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	)	S	)								…

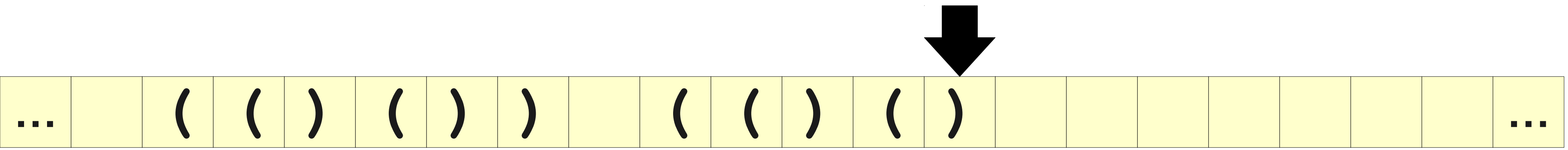
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



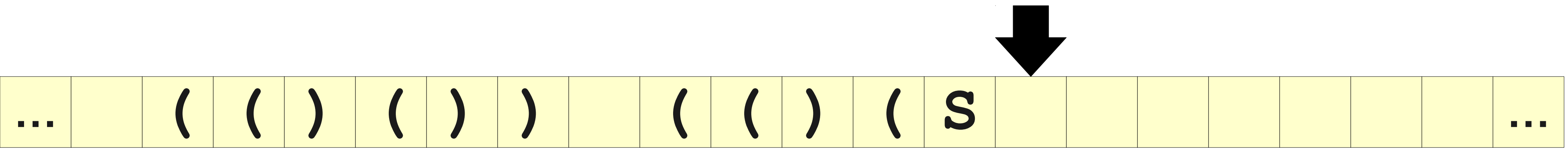
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



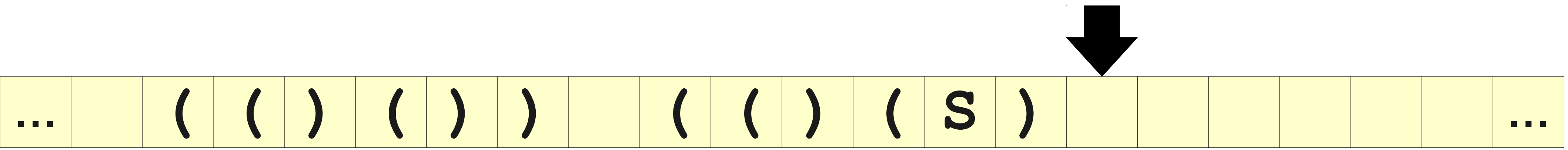
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



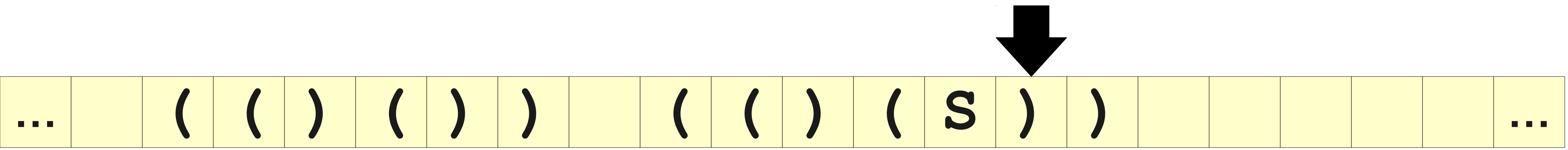
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



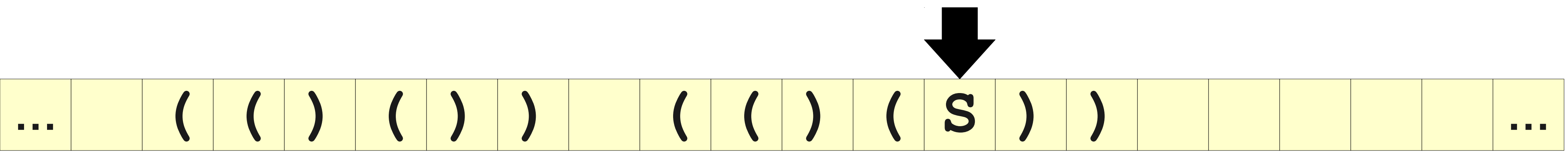
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



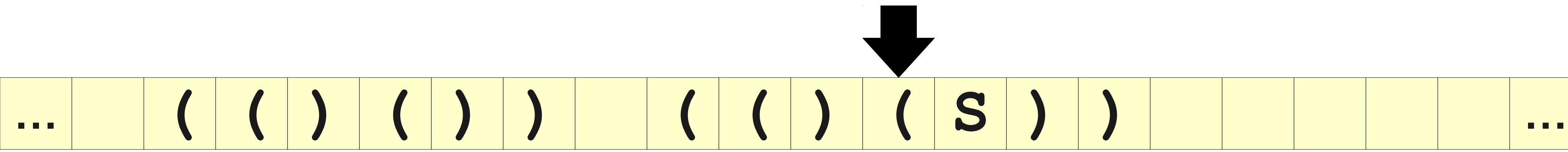
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



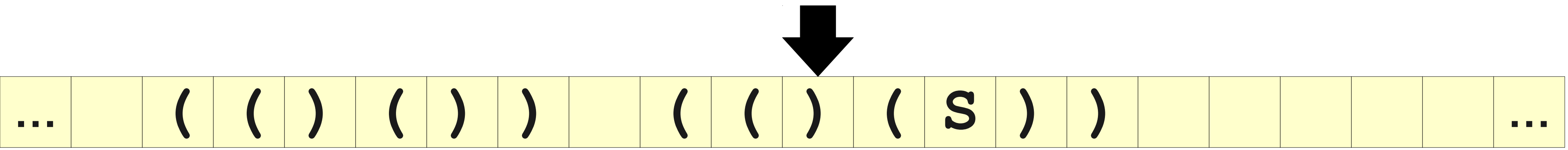
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



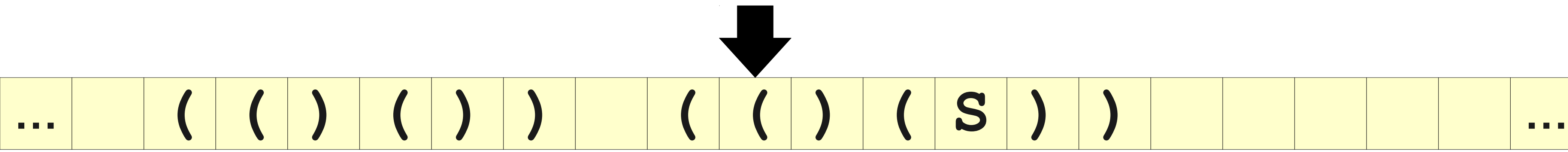
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



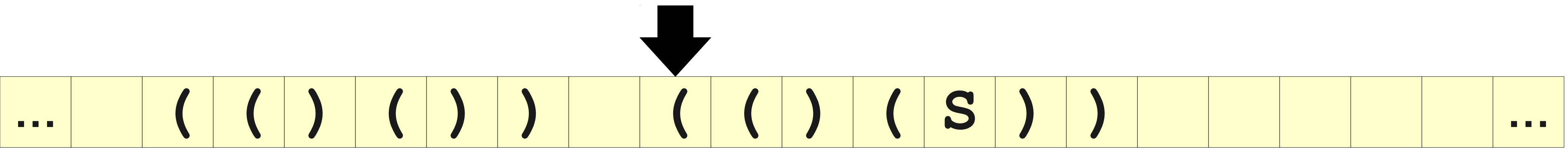
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



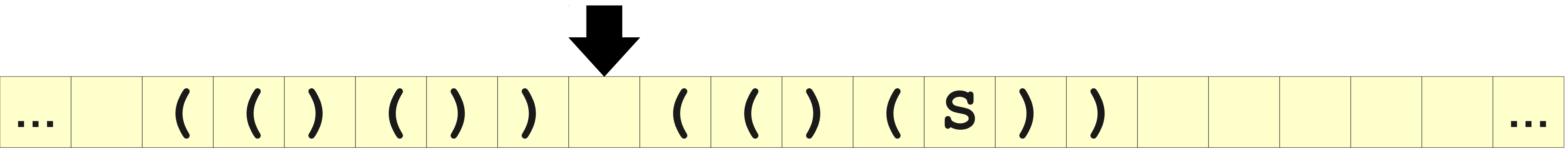
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



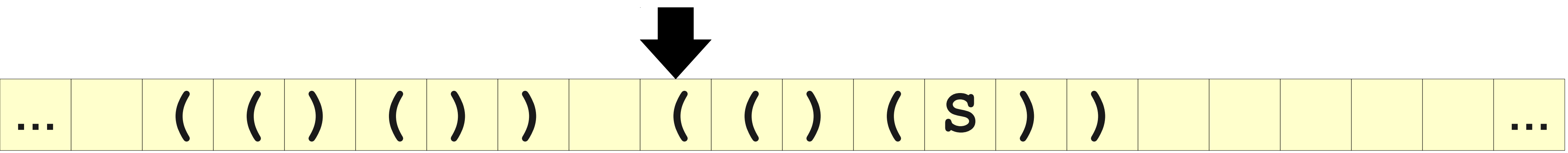
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



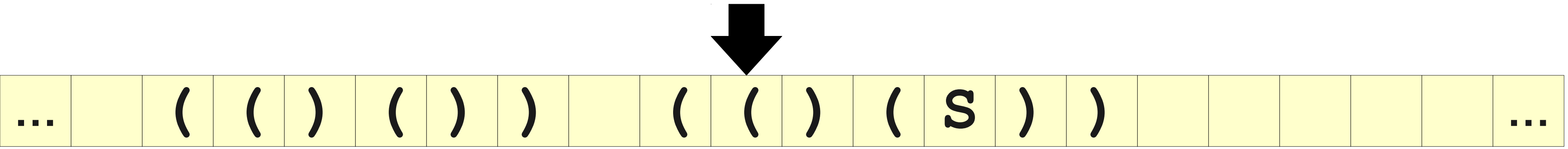
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



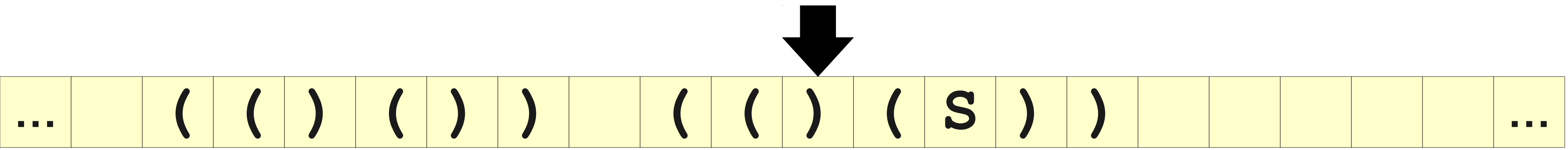
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



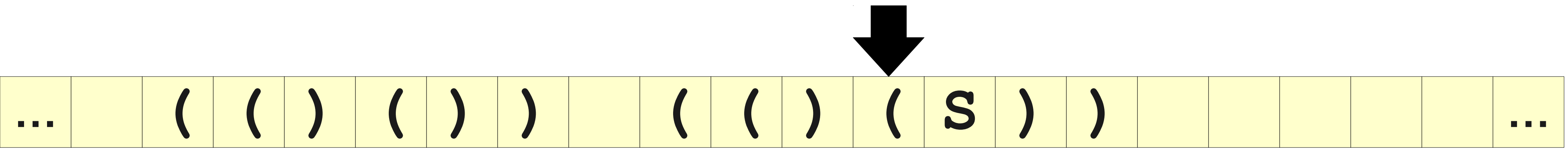
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



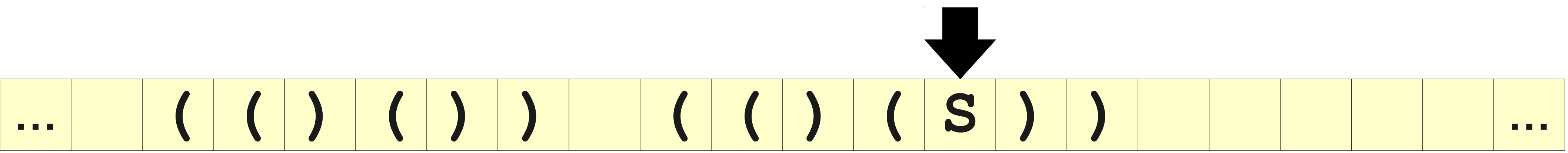
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



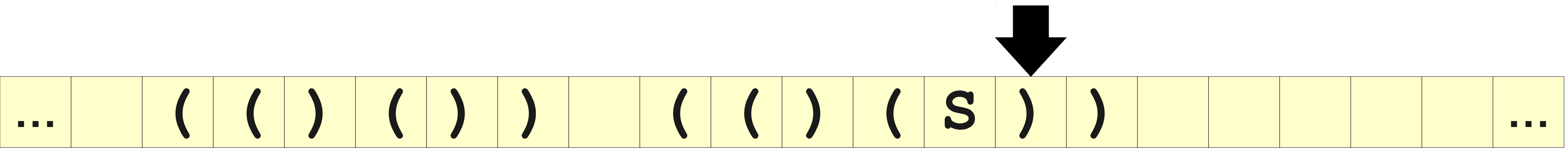
$$\mathbf{S} \to \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



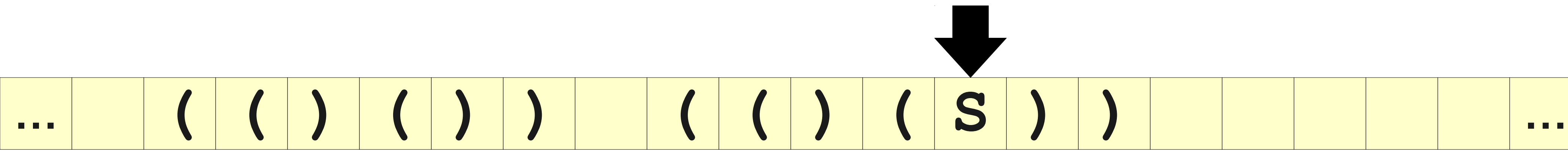
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



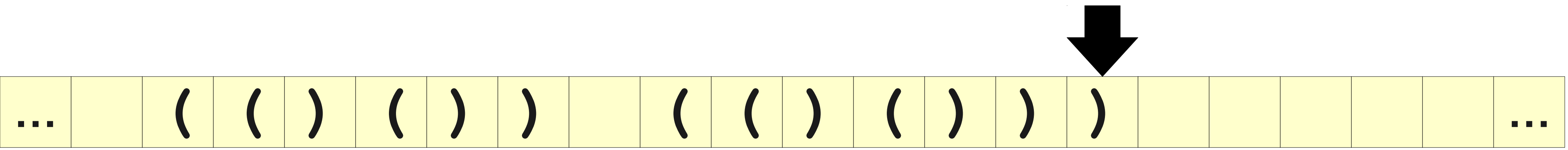
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	)	(	)	)	)						…

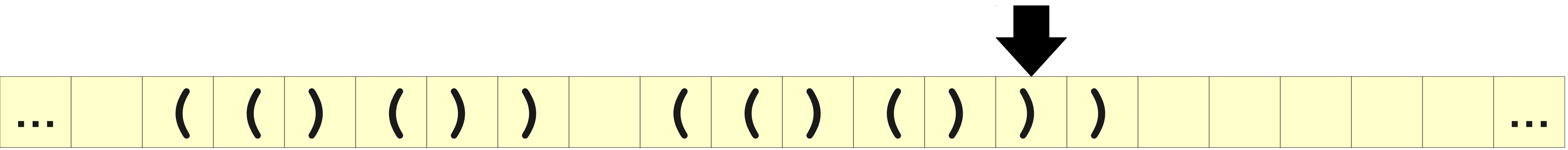
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



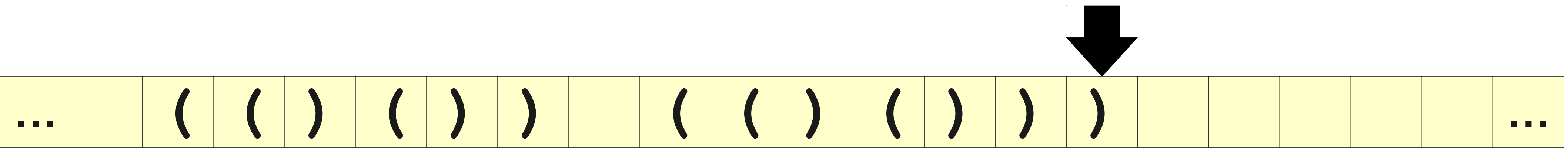
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



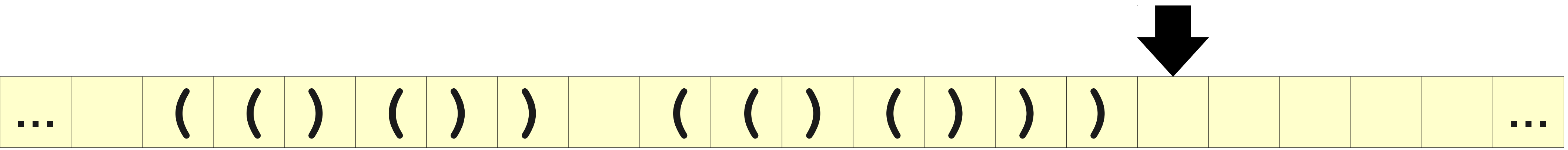
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



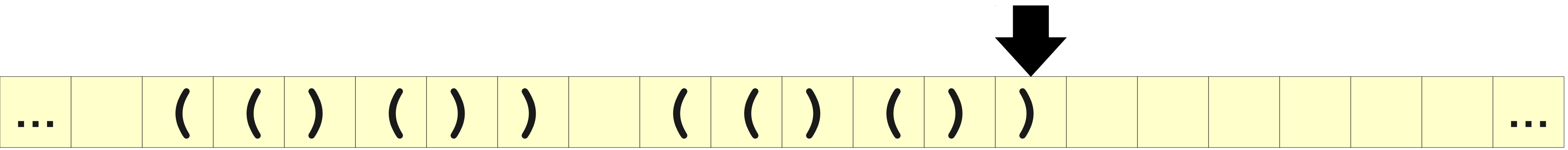
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction

...		(	(	)	(	)	)		(	(	)	(	)	)	)						...

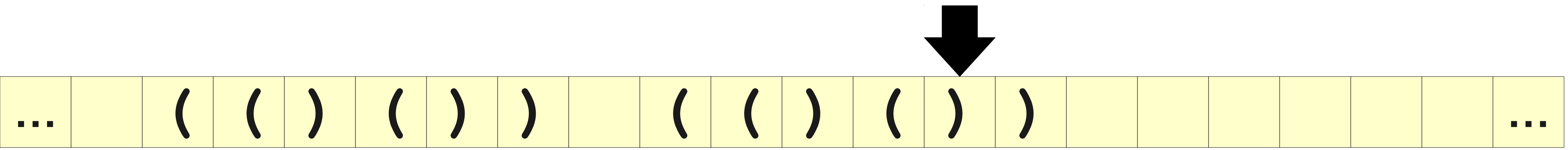
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



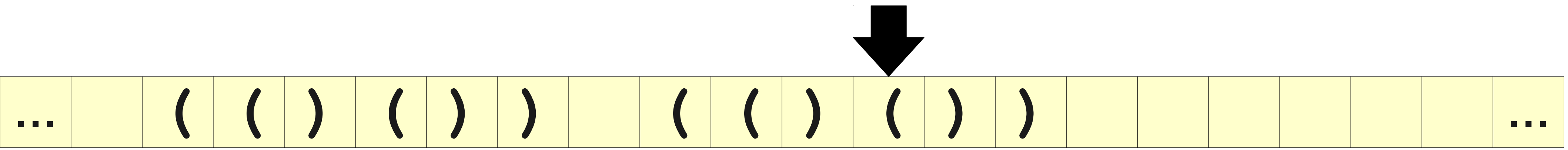
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



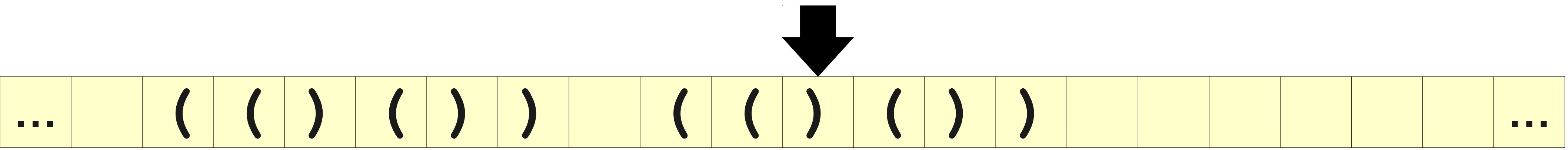
$$\mathbf{S} \to \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



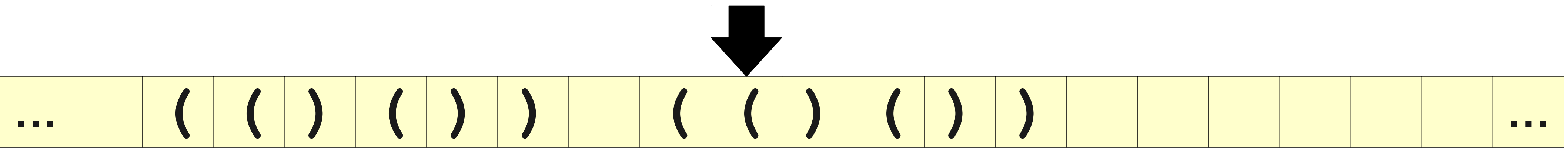
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



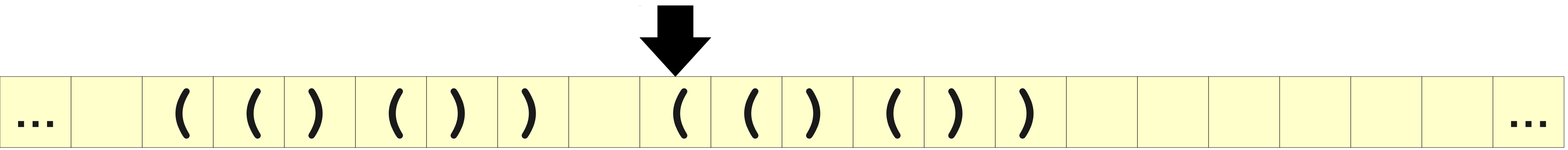
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



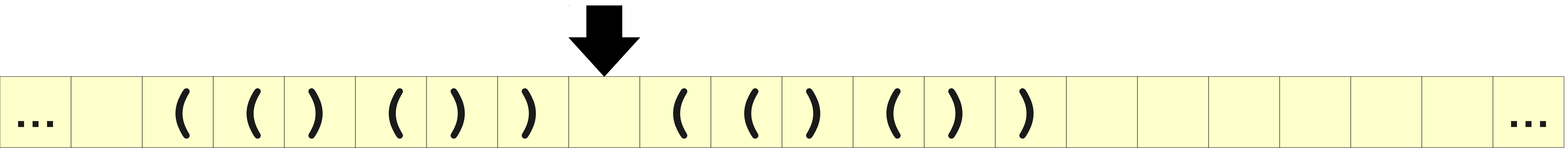
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



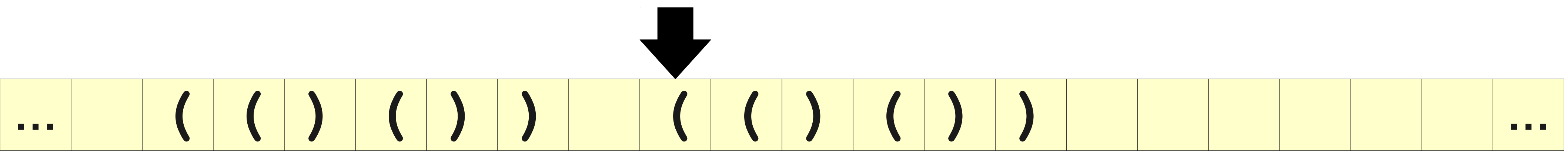
$$\mathbf{S} \to \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction



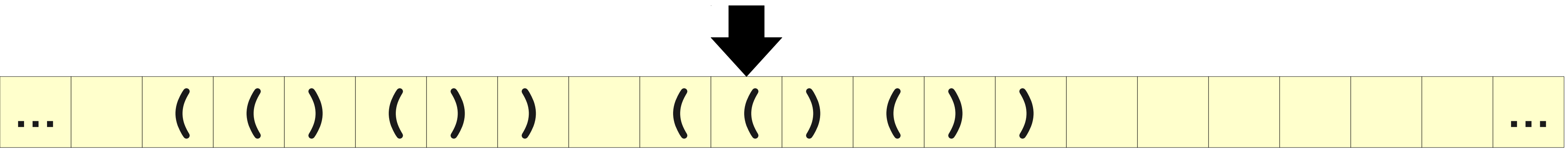
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



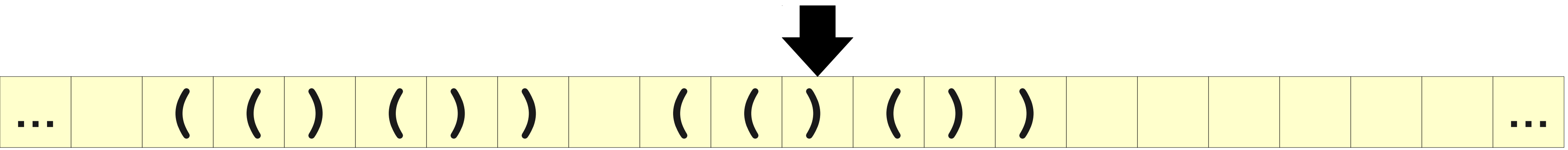
$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathtt{(}\mathbf{S}\mathtt{)} \mid \varepsilon$$

A Sketch of the Construction



$$\mathbf{S} \rightarrow \mathbf{SS} \mid \texttt{(}\mathbf{S}\texttt{)} \mid \varepsilon$$

A Sketch of the Construction



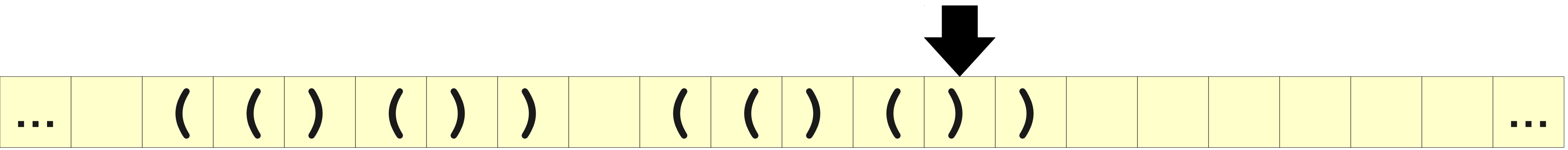
$$\mathbf{S} \rightarrow \mathbf{SS} \mid (\mathbf{S}) \mid \varepsilon$$

A Sketch of the Construction

…		(	(	)	(	)	)		(	(	)	(	)	)							…

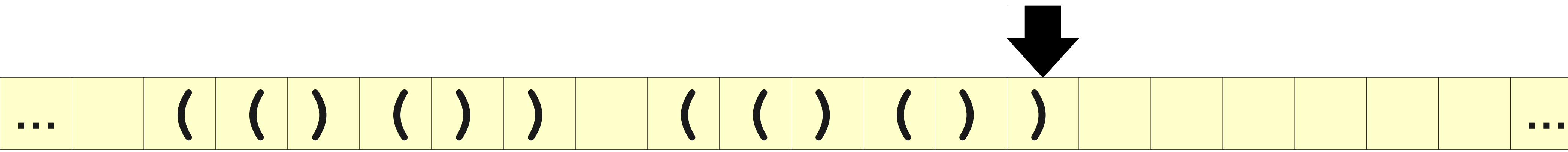
$$S \to SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



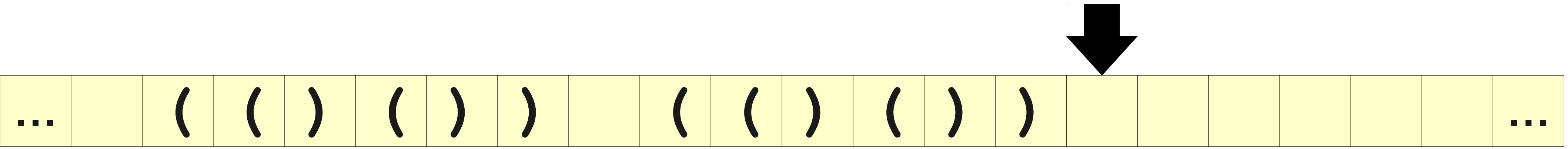
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



$$S \rightarrow SS \mid (S) \mid \varepsilon$$

A Sketch of the Construction



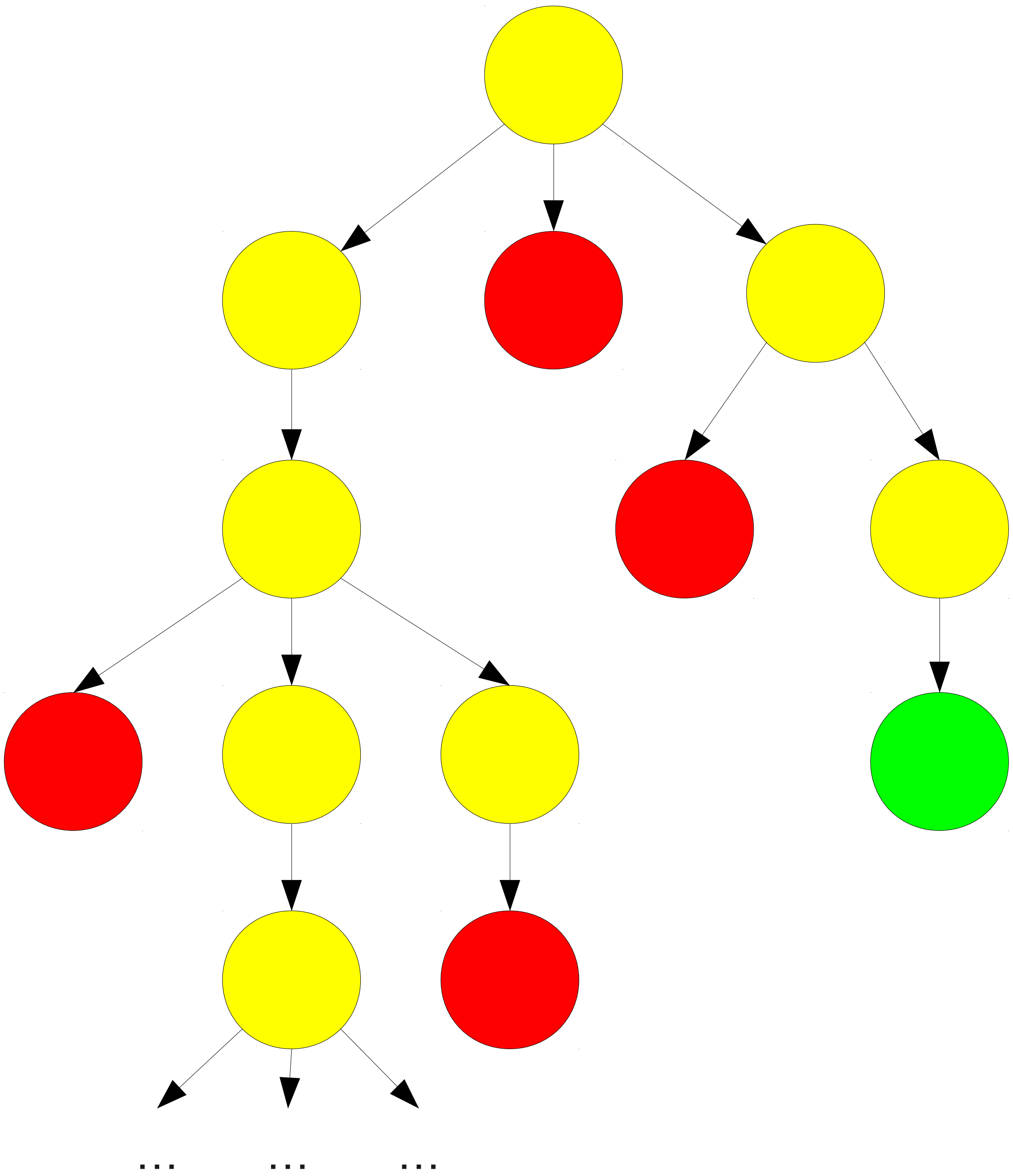
$$S \rightarrow SS \mid (S) \mid \varepsilon$$

The Story So Far

- We now have two different models of solving search problems:
  - Build a worklist and explicitly step through all options.
  - Use a nondeterministic Turing machine.
- Are these two approaches equivalent?
- That is, are NTMs and DTMs equal in power?

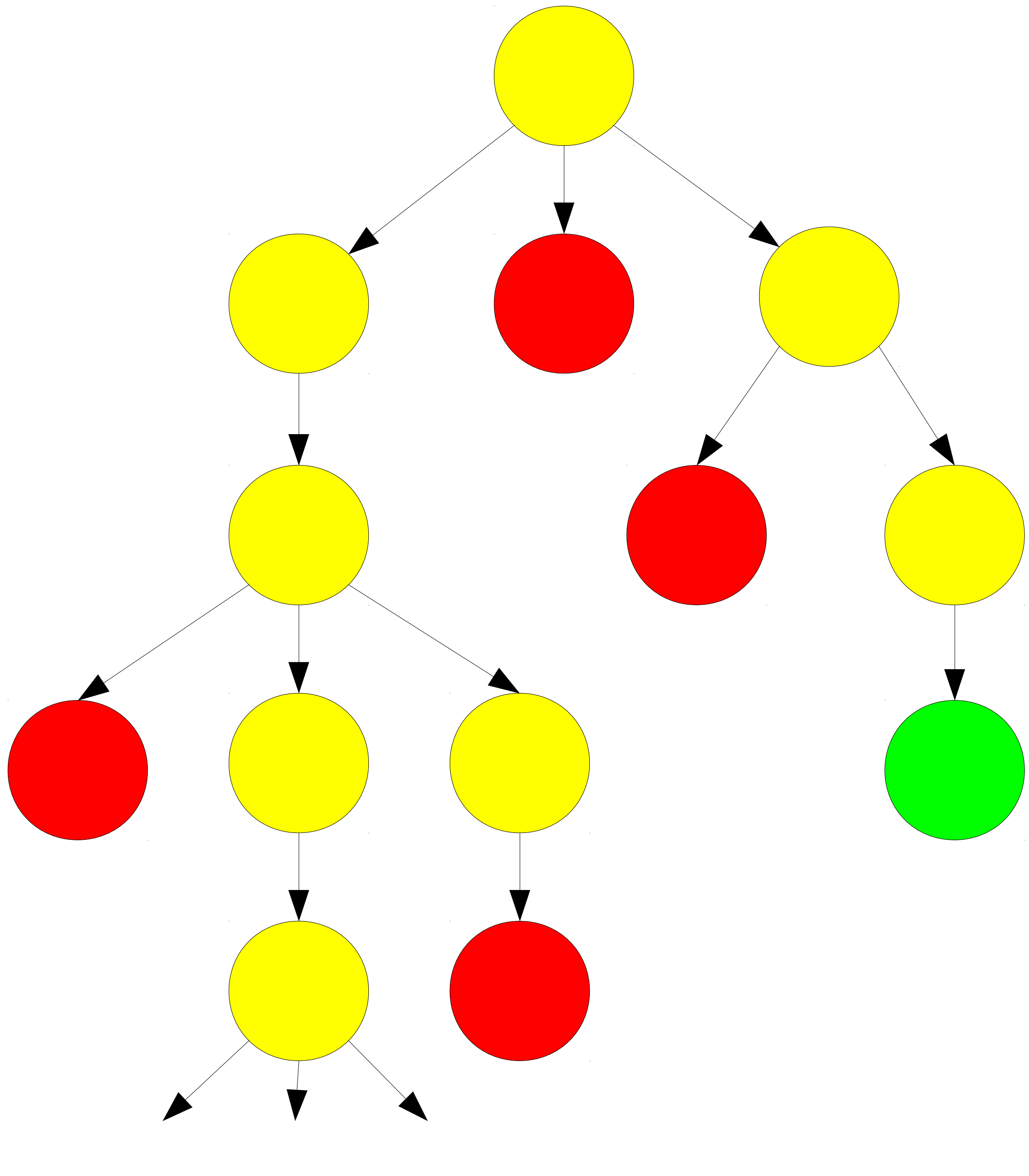
Review: Tree Computation

- One interpretation of nondeterminism is as a **tree computation**.
- Each node in the tree has children corresponding to each possible choice for the computation.
- The computation accepts if *any* node in tree enters an accepting state.



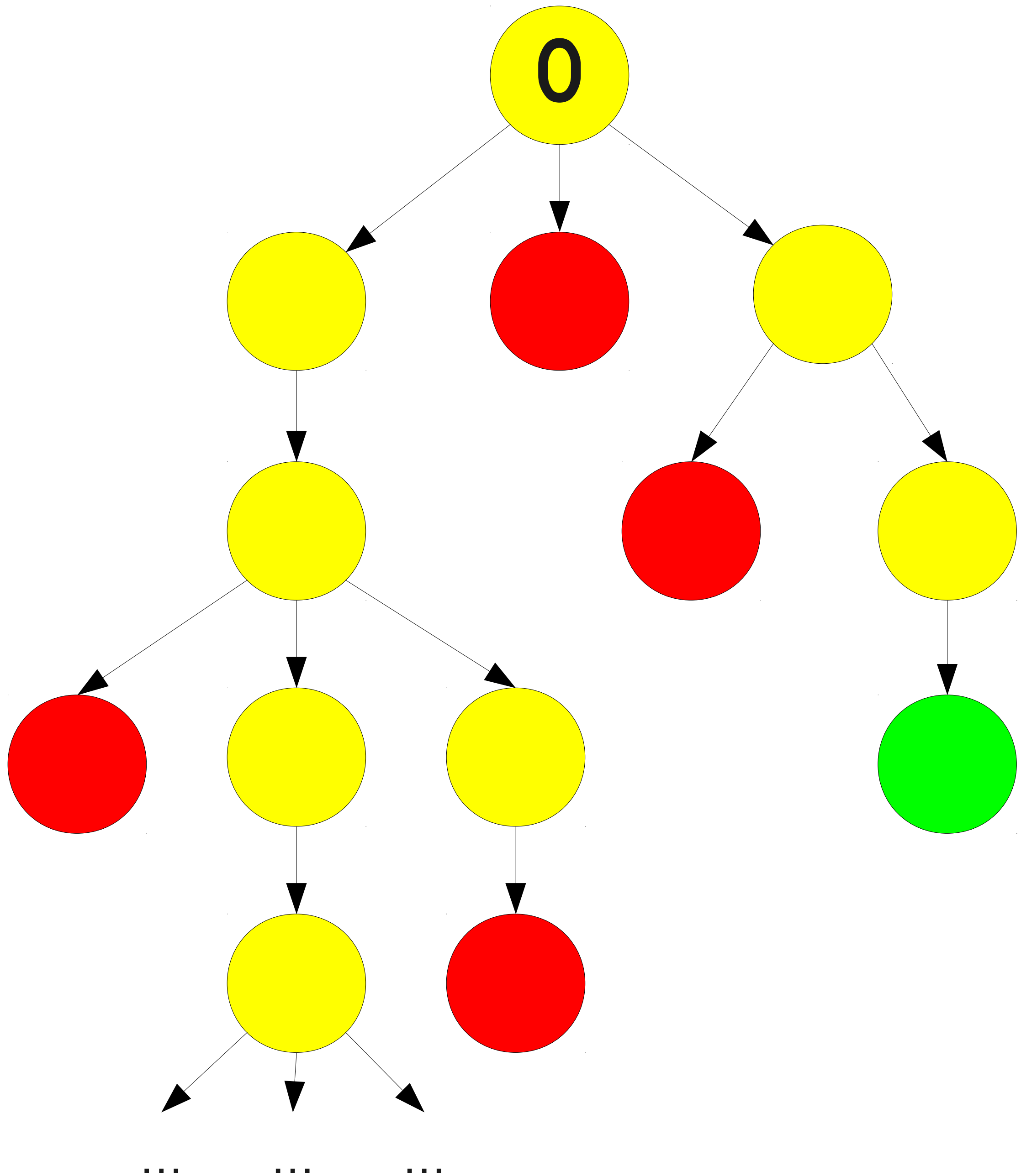
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



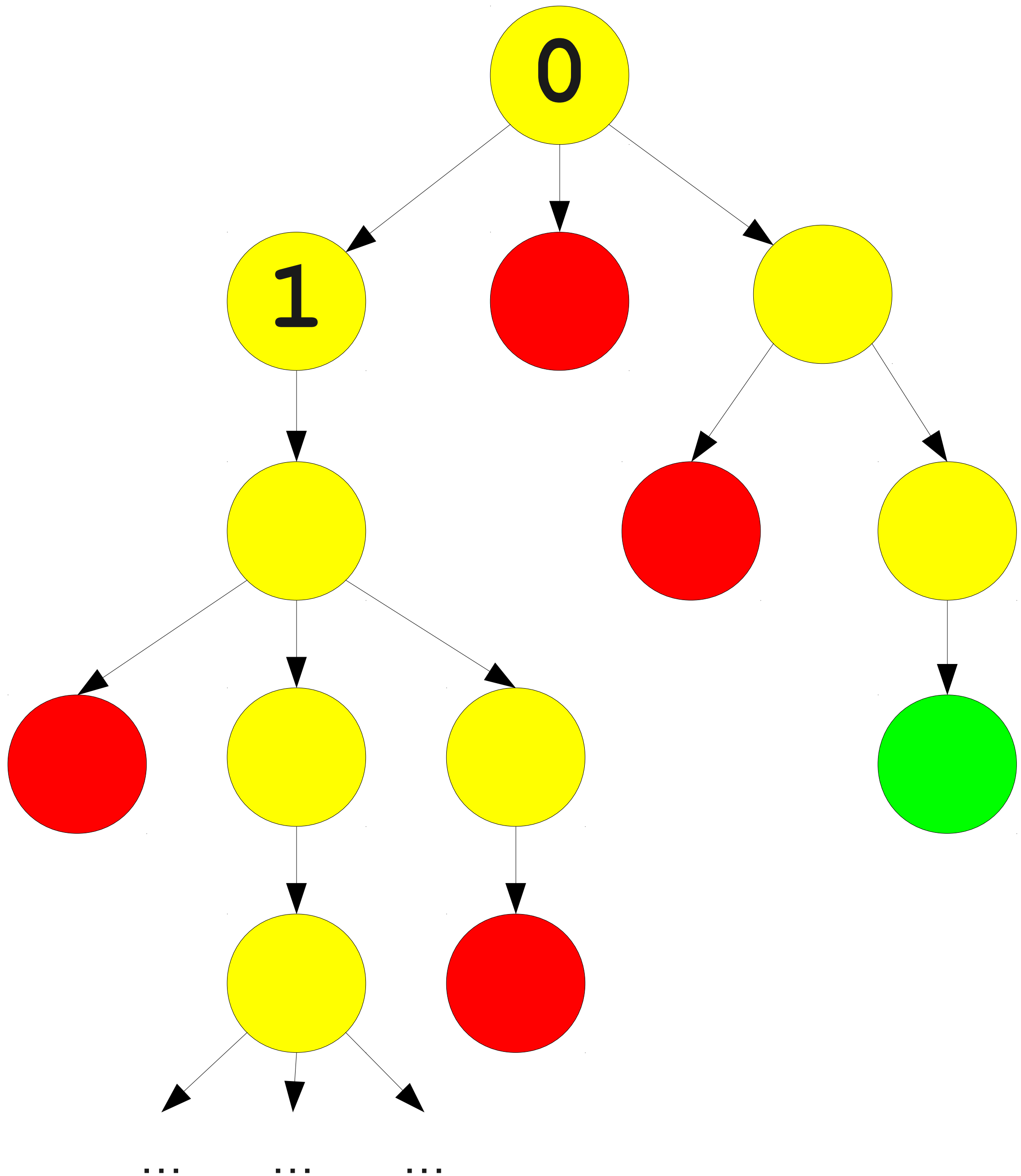
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



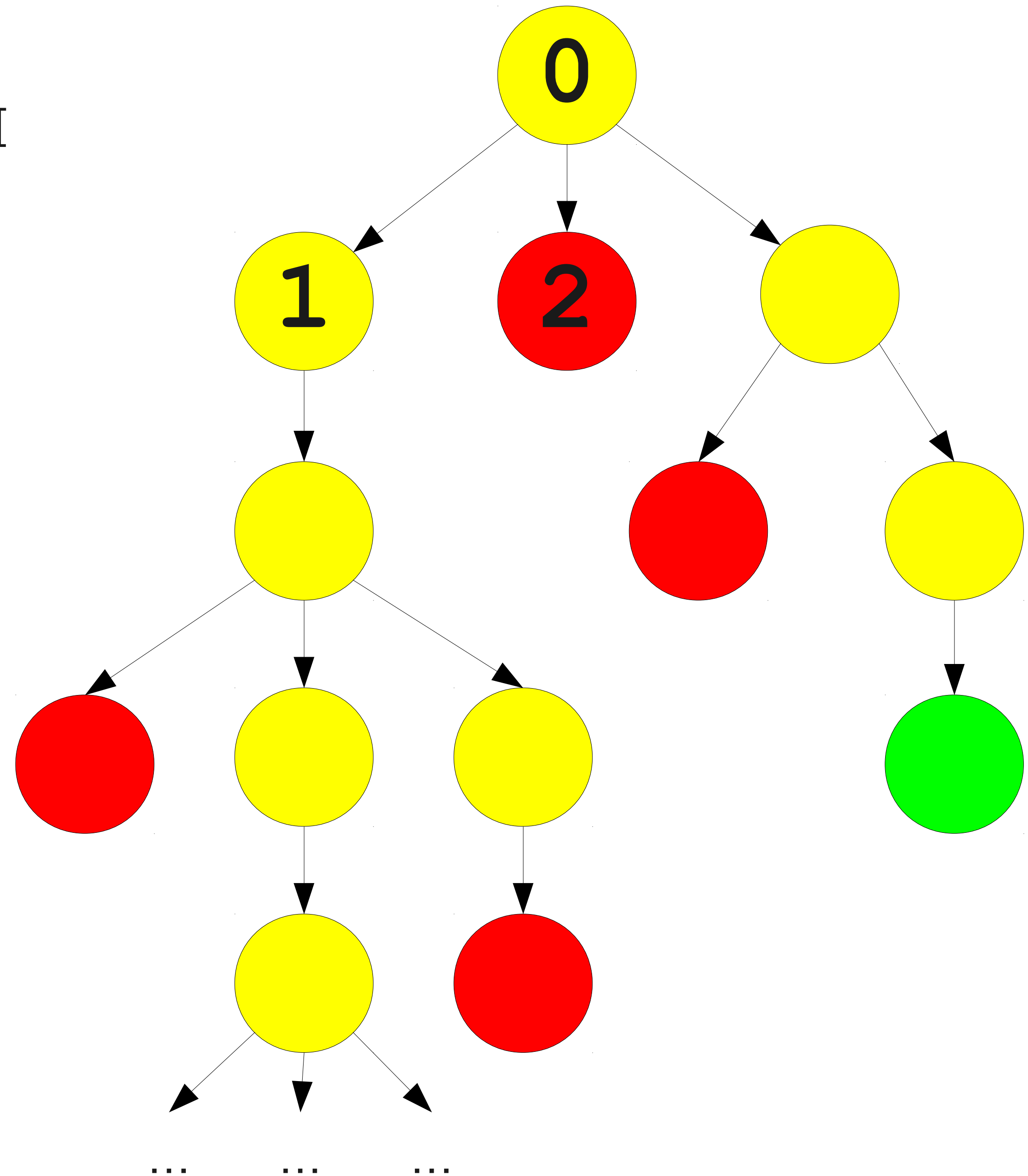
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



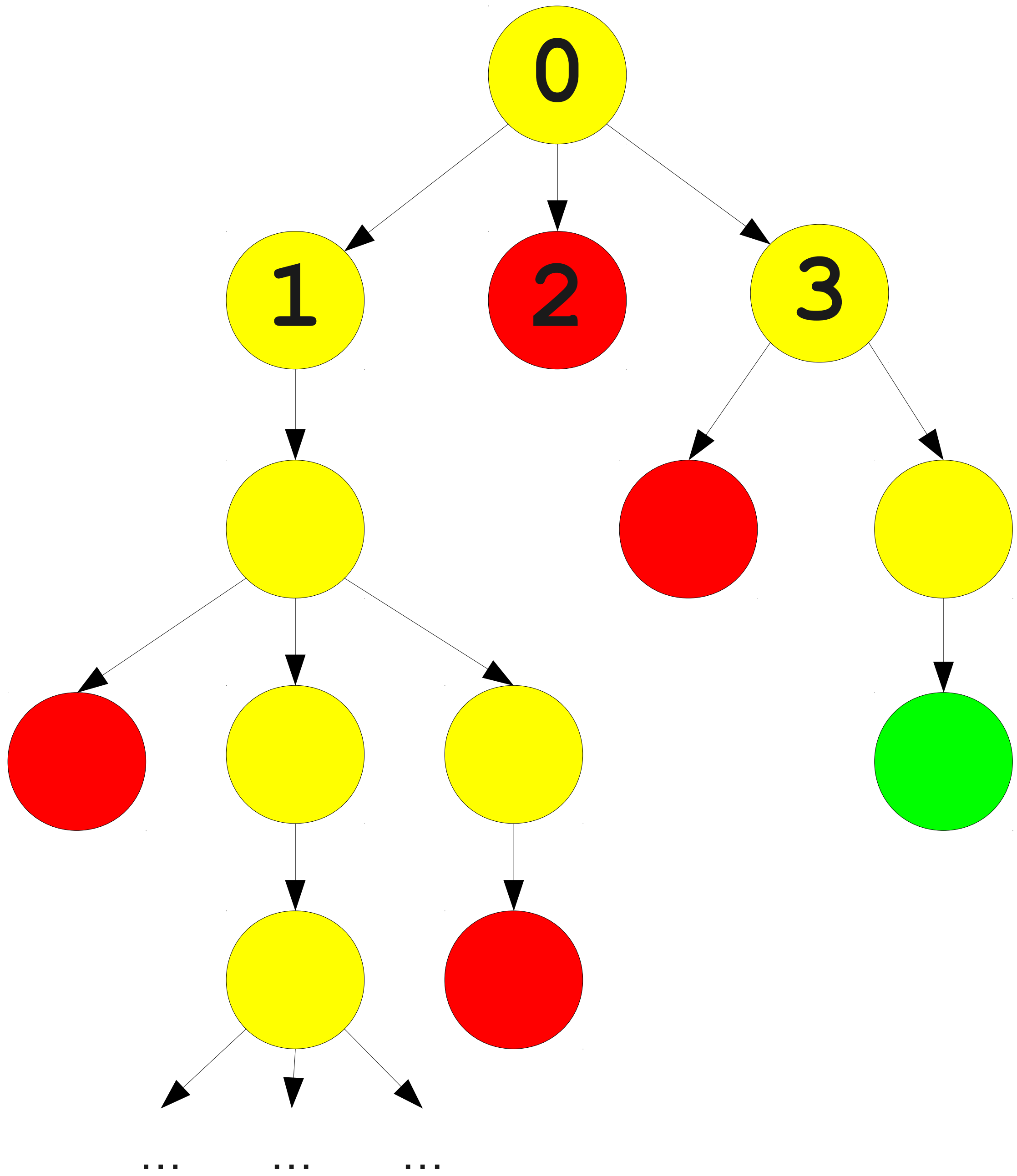
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



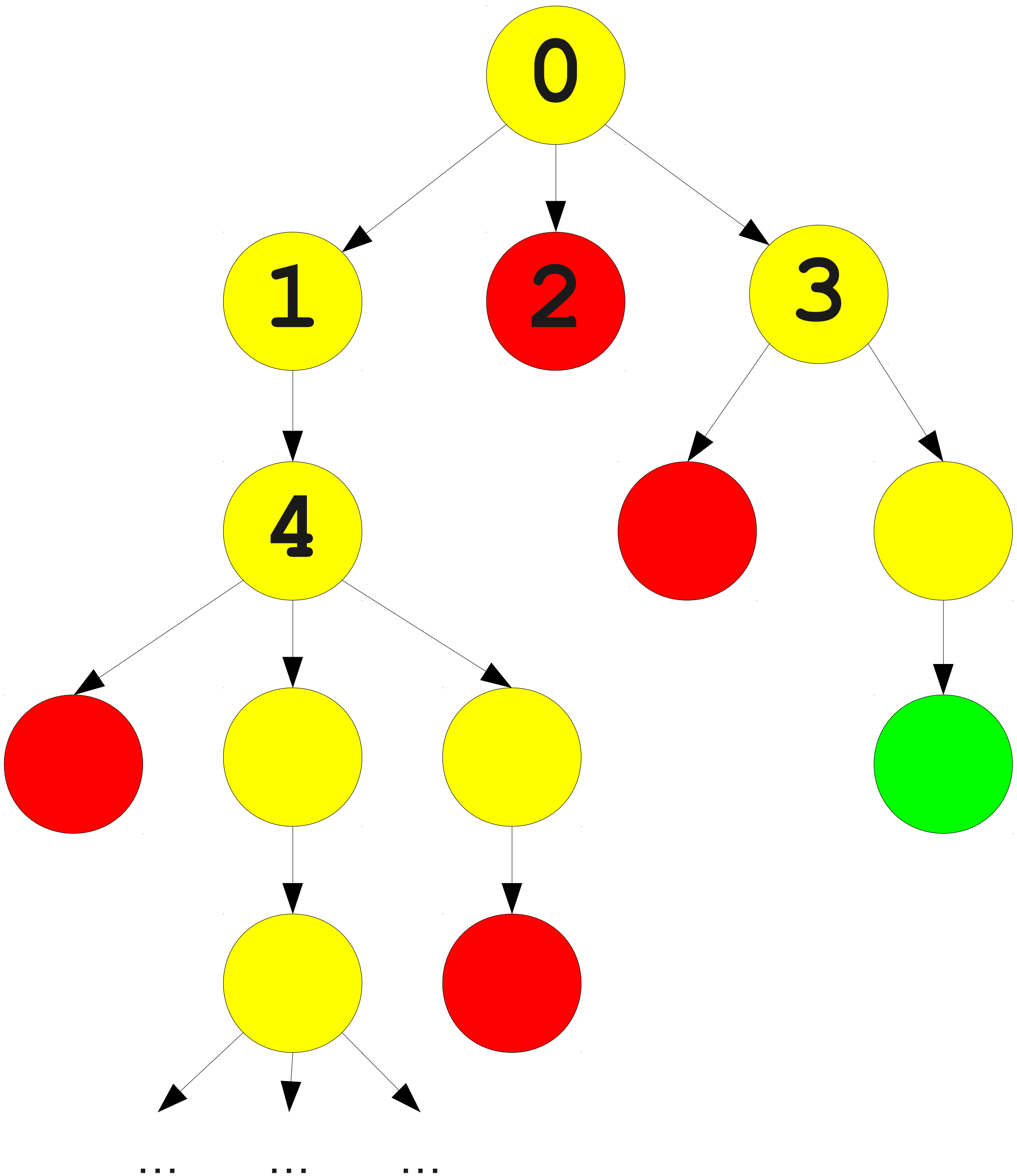
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



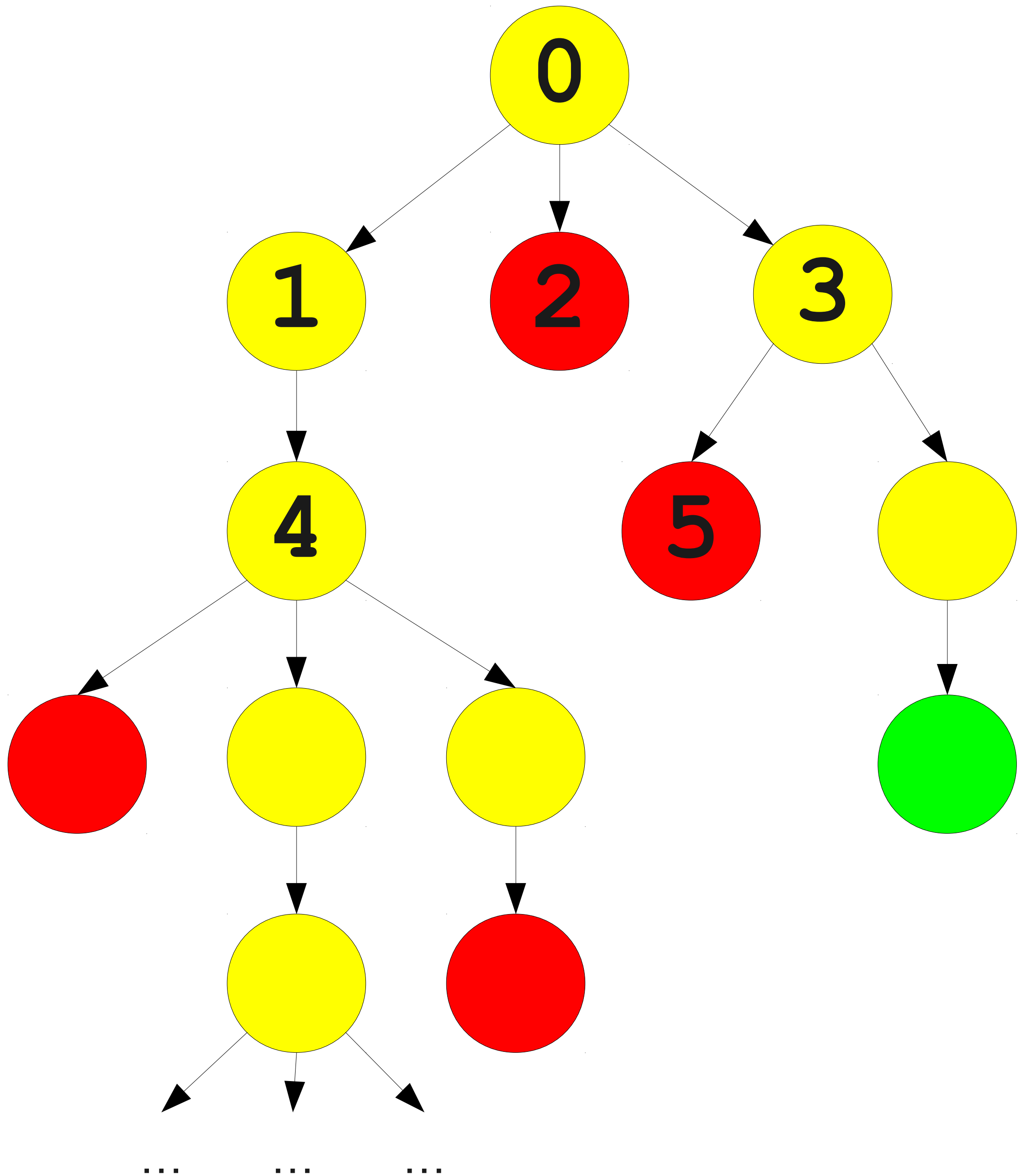
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



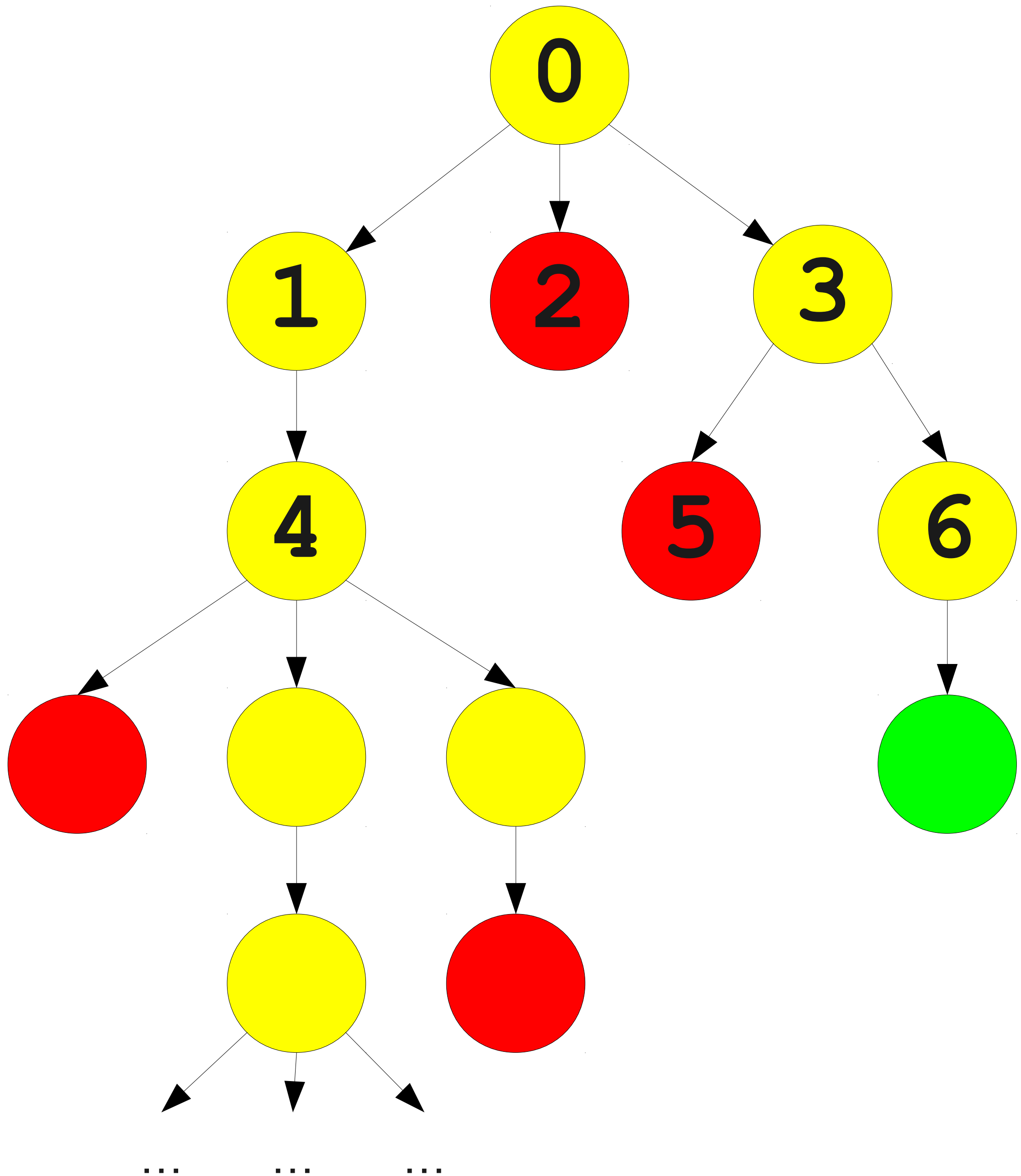
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



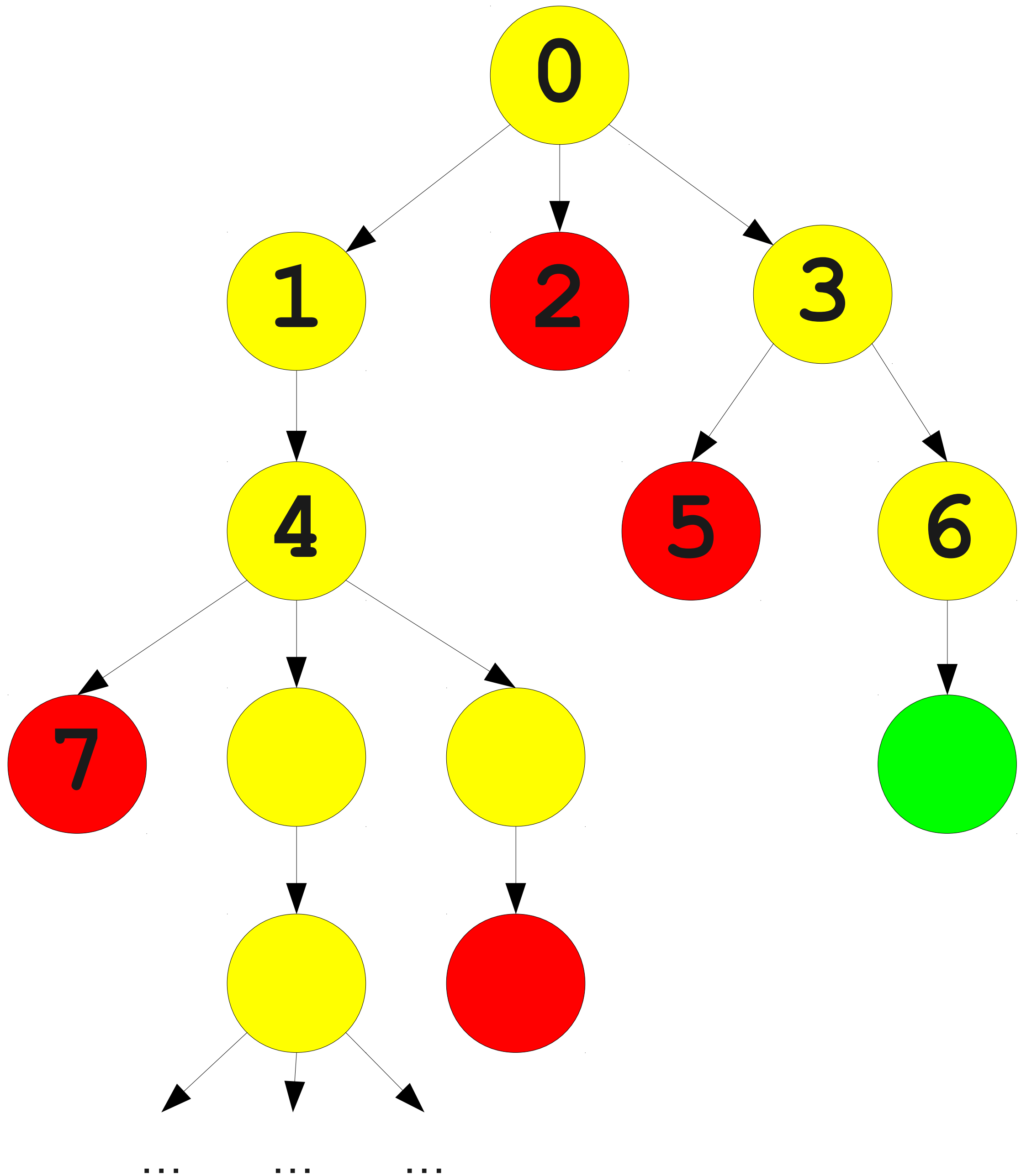
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



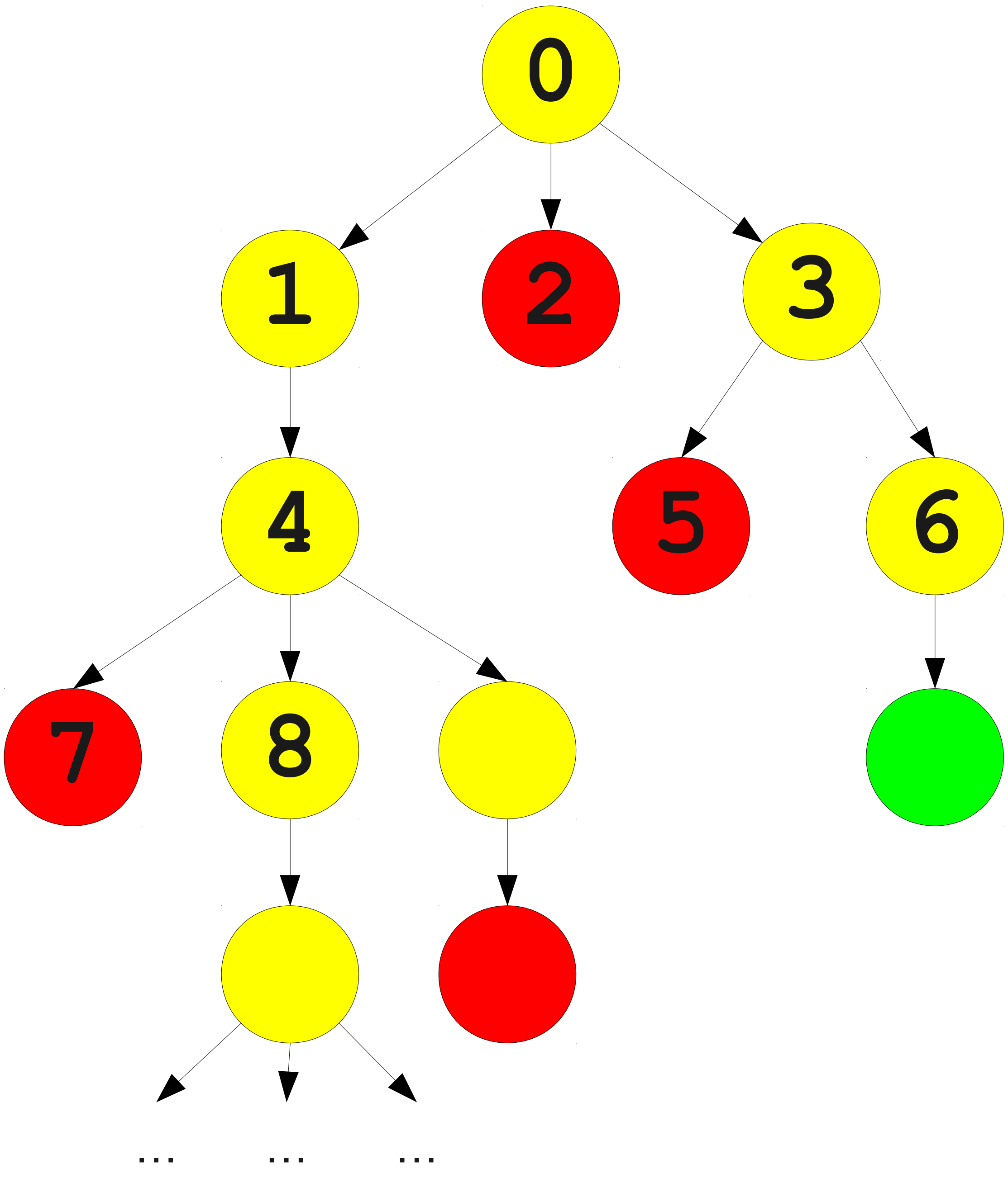
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



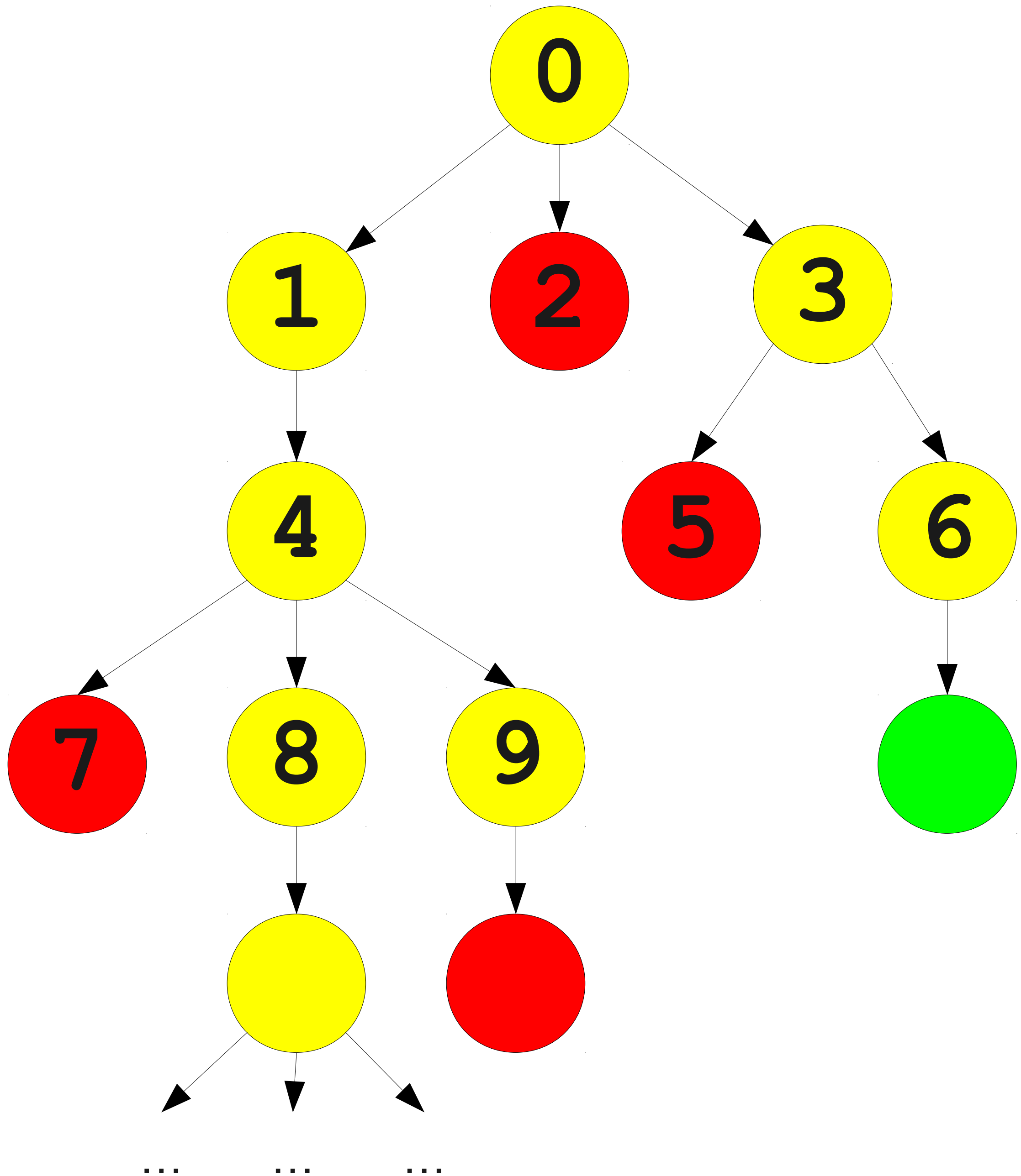
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



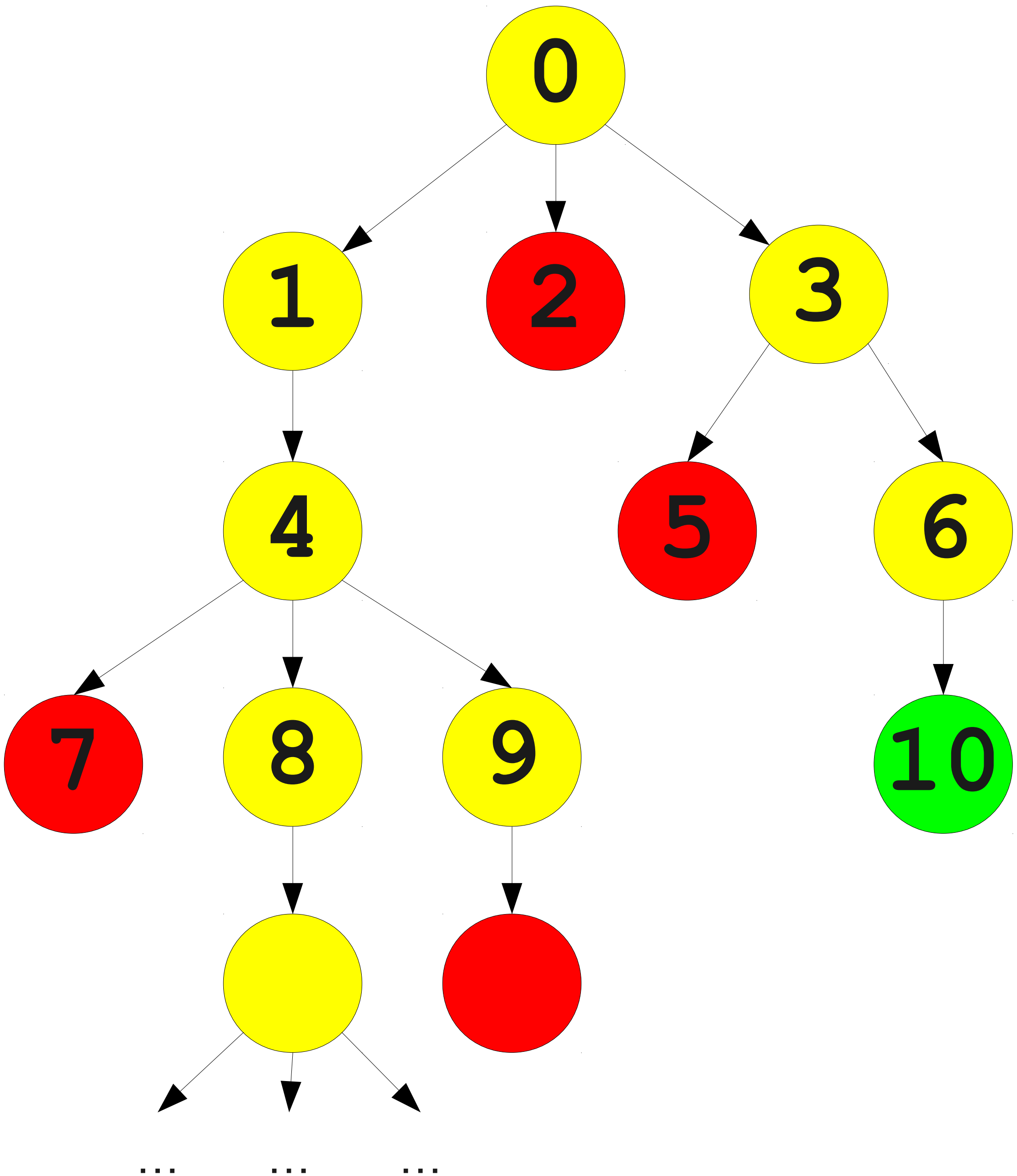
The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



The Key Idea

- **Idea:** Simulate an NTM with a DTM by exhaustively searching the computation tree!
- Start at the root node and go layer by layer.
- If an accepting path is found, accept.
- If all paths reject, reject.
- Otherwise, keep looking.



Exploring the Tree

- Each node in this tree consists of one possible configuration that the NTM might be in at any time.
- What does this consist of?
  - The contents of the tape.
  - The position of the tape head.
  - The current state.

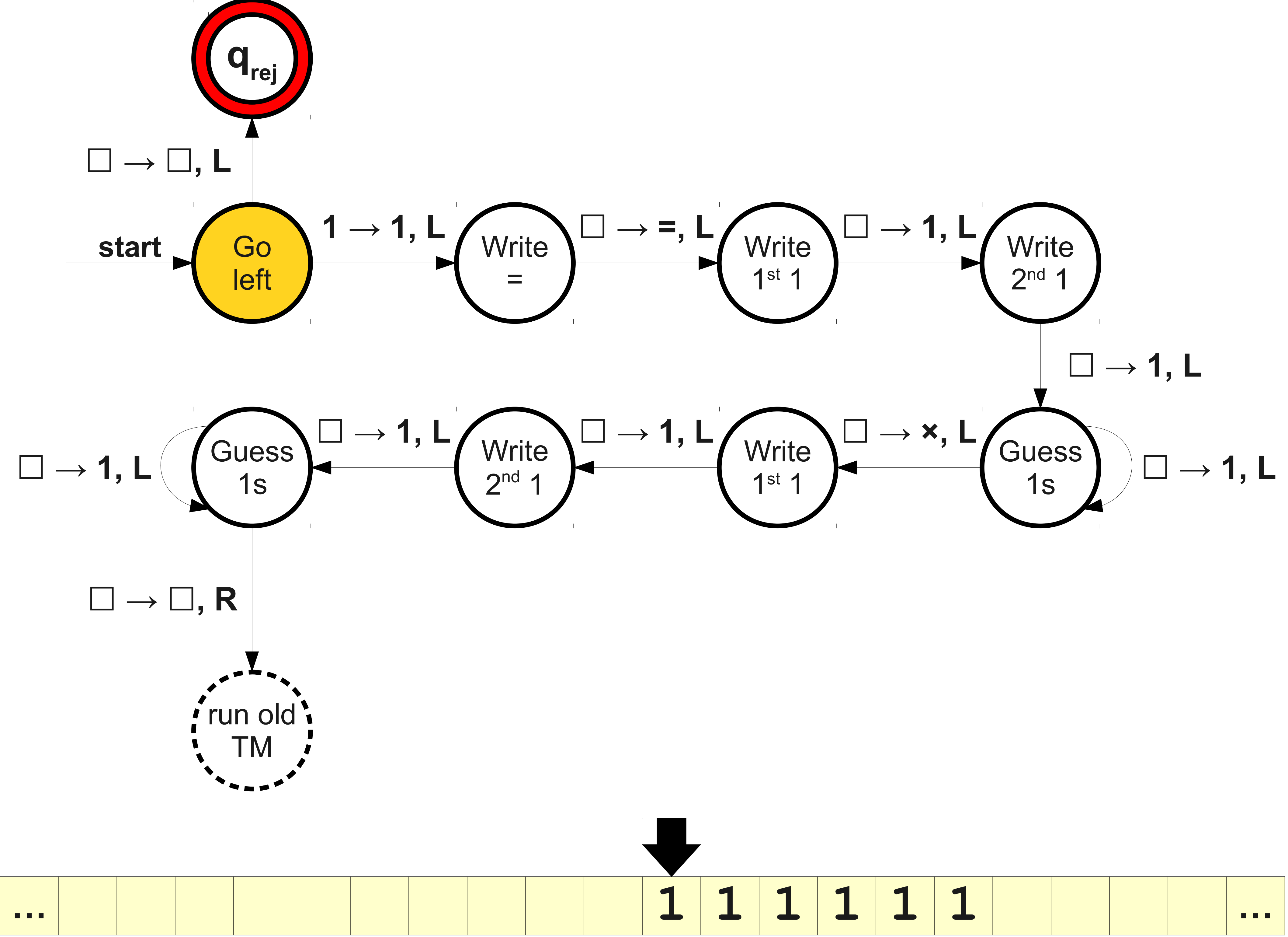
Exploring the Tree

Each node in this tree consists of one possible configuration that the NTM might be in at any time.

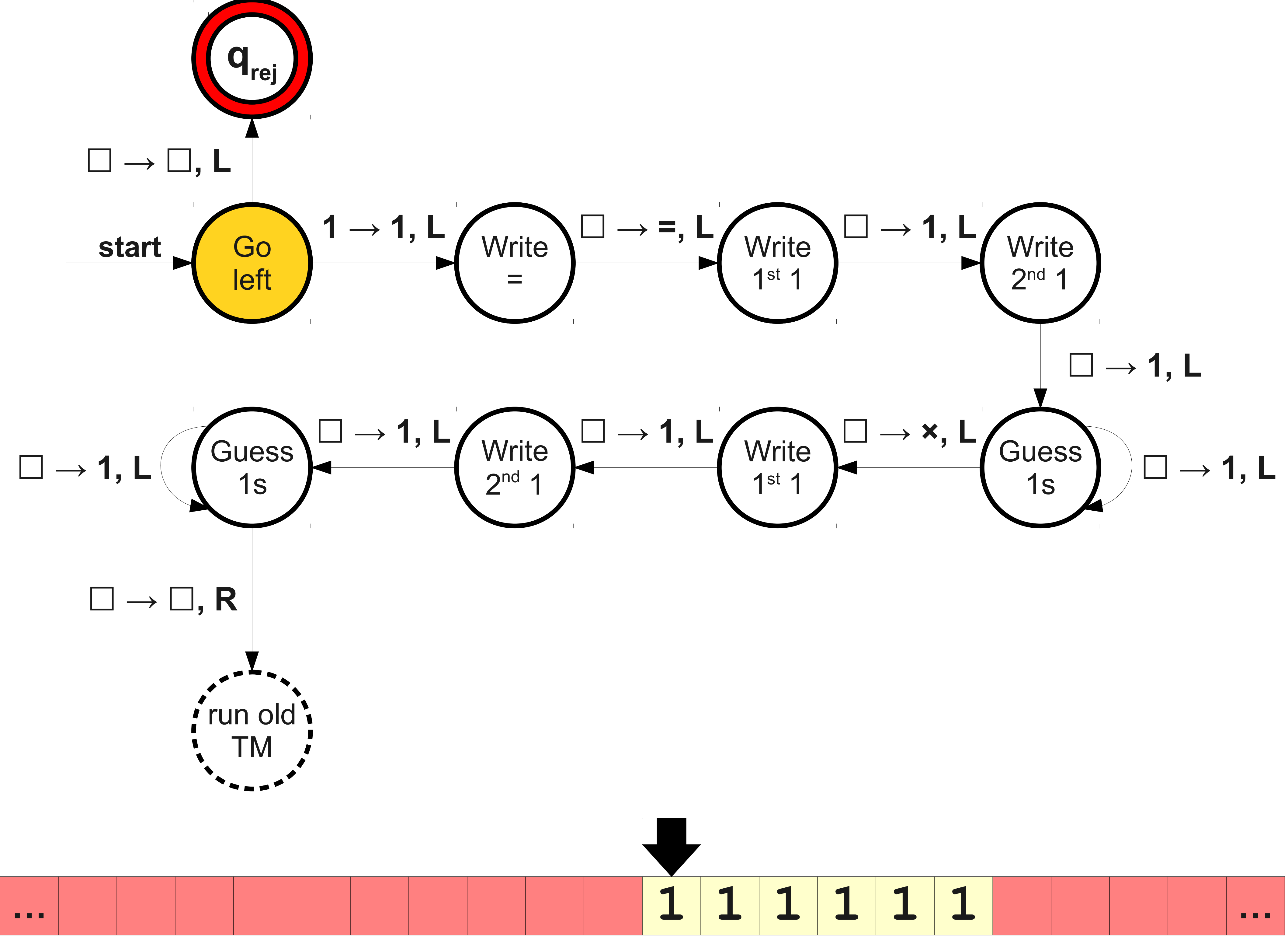
What does this consist of?

- The contents of the tape.
- The position of the tape head.
- The current state.

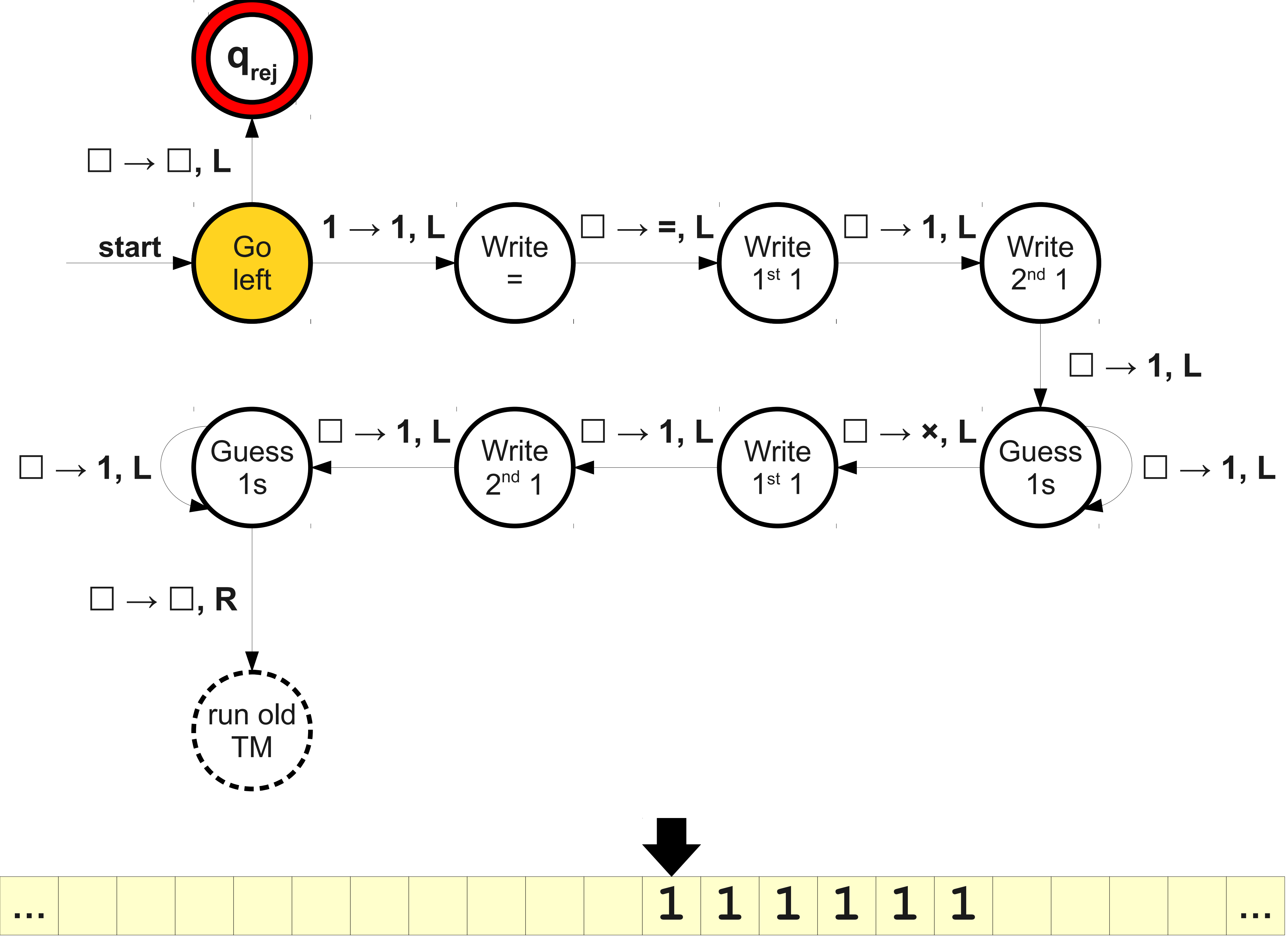
The tape is infinite!



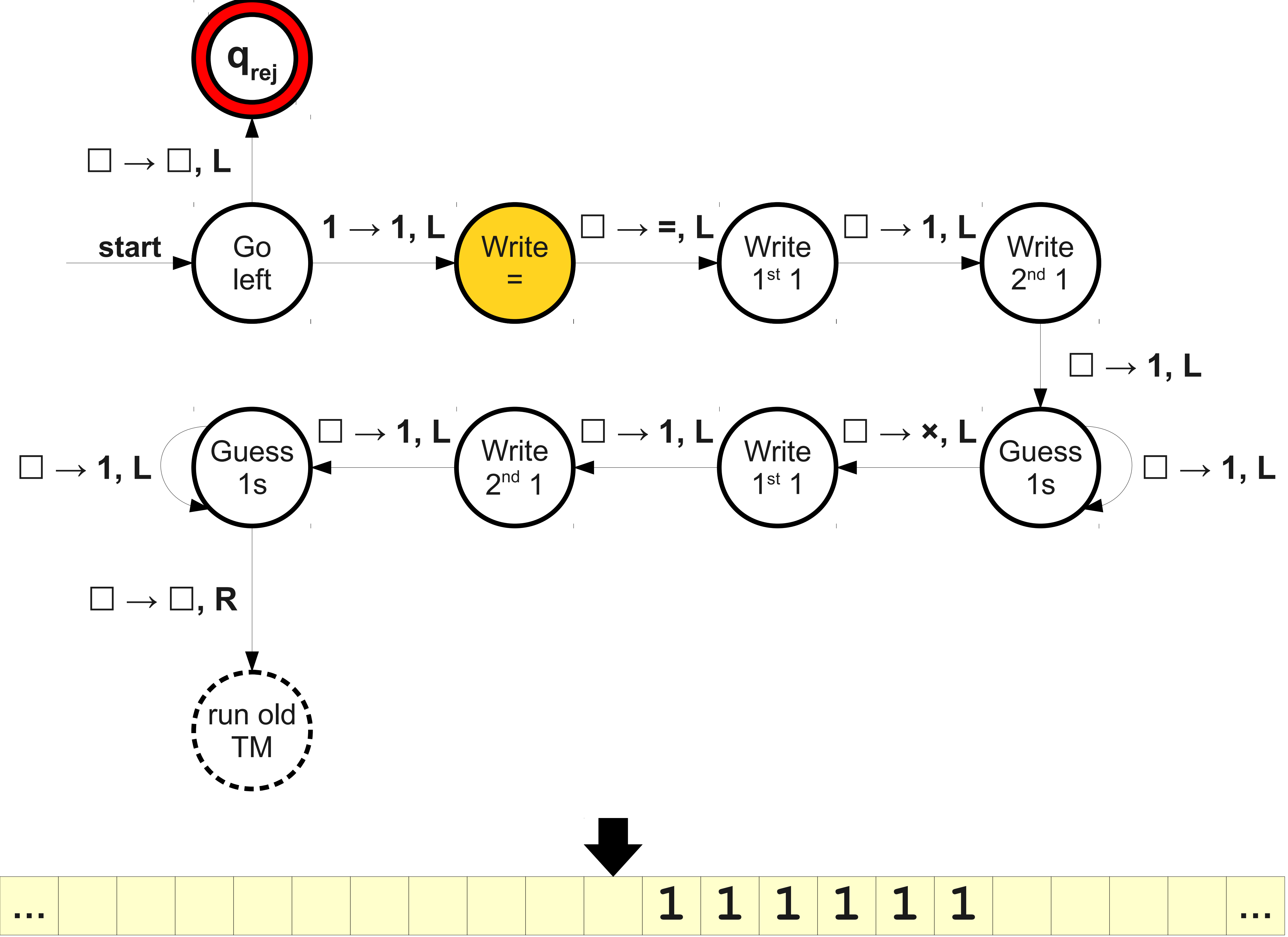
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
… 1 1 1 1 1 1 …



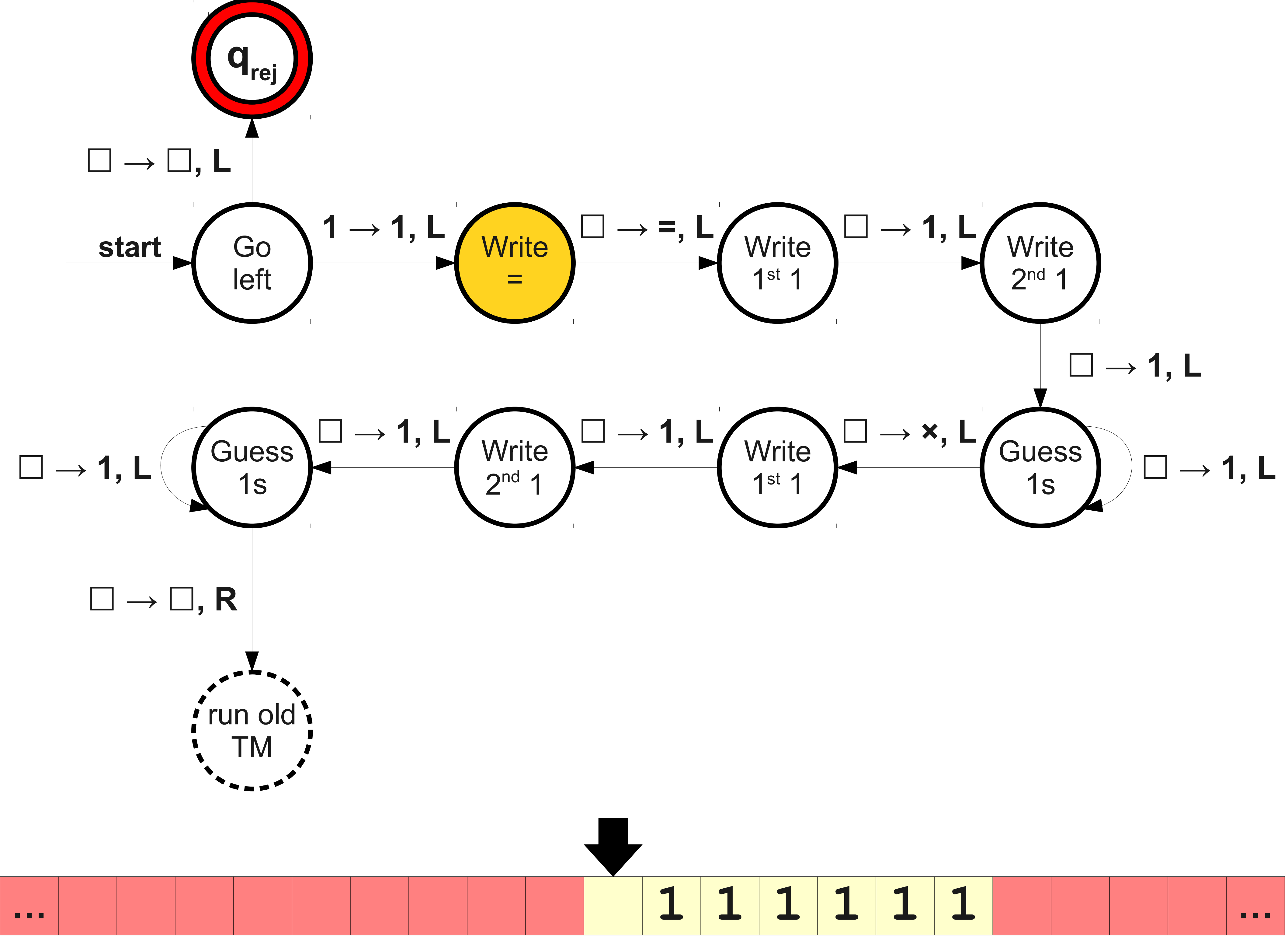
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
... 1 1 1 1 1 1 ...



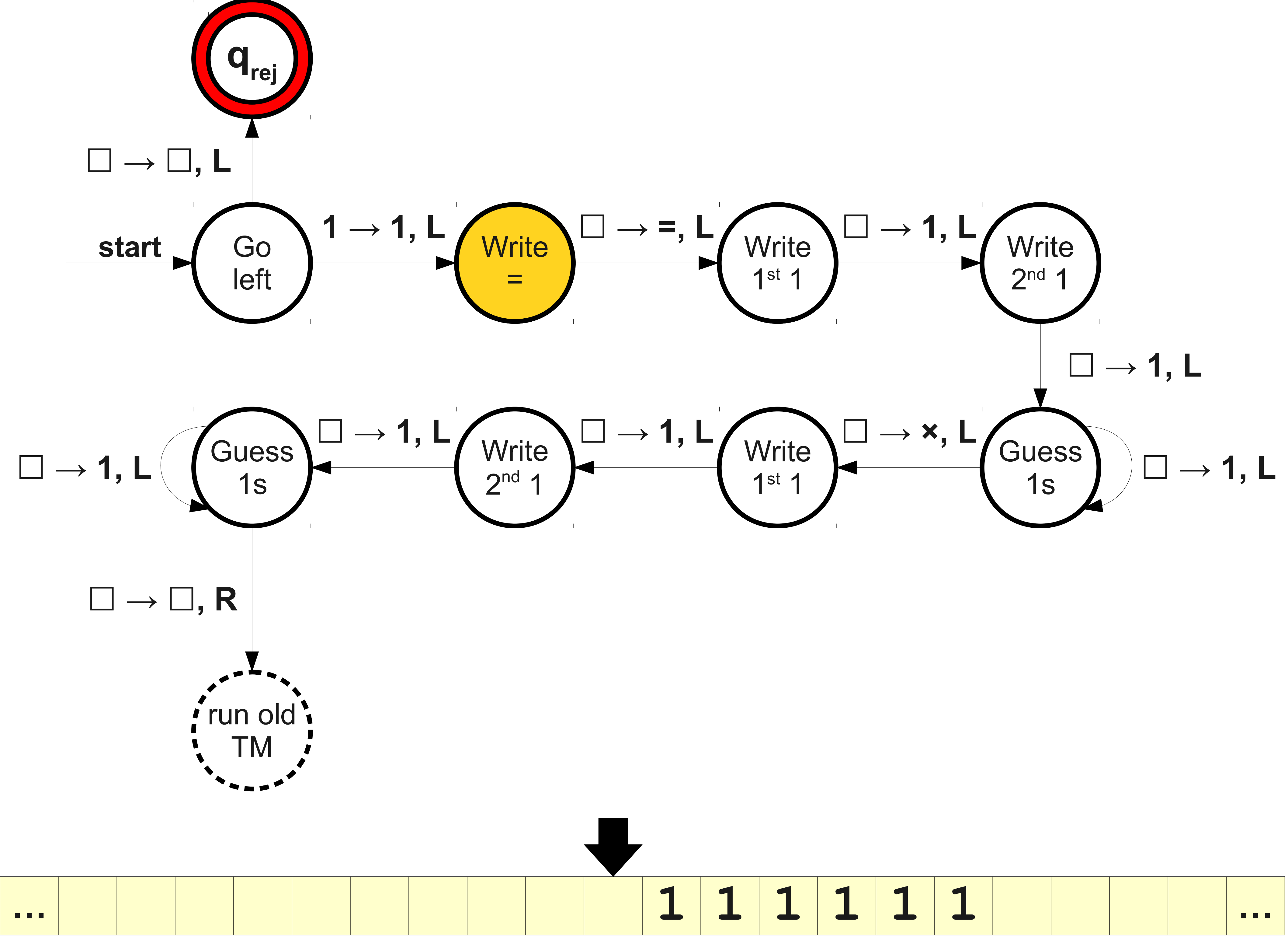
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
… 1 1 1 1 1 1 …



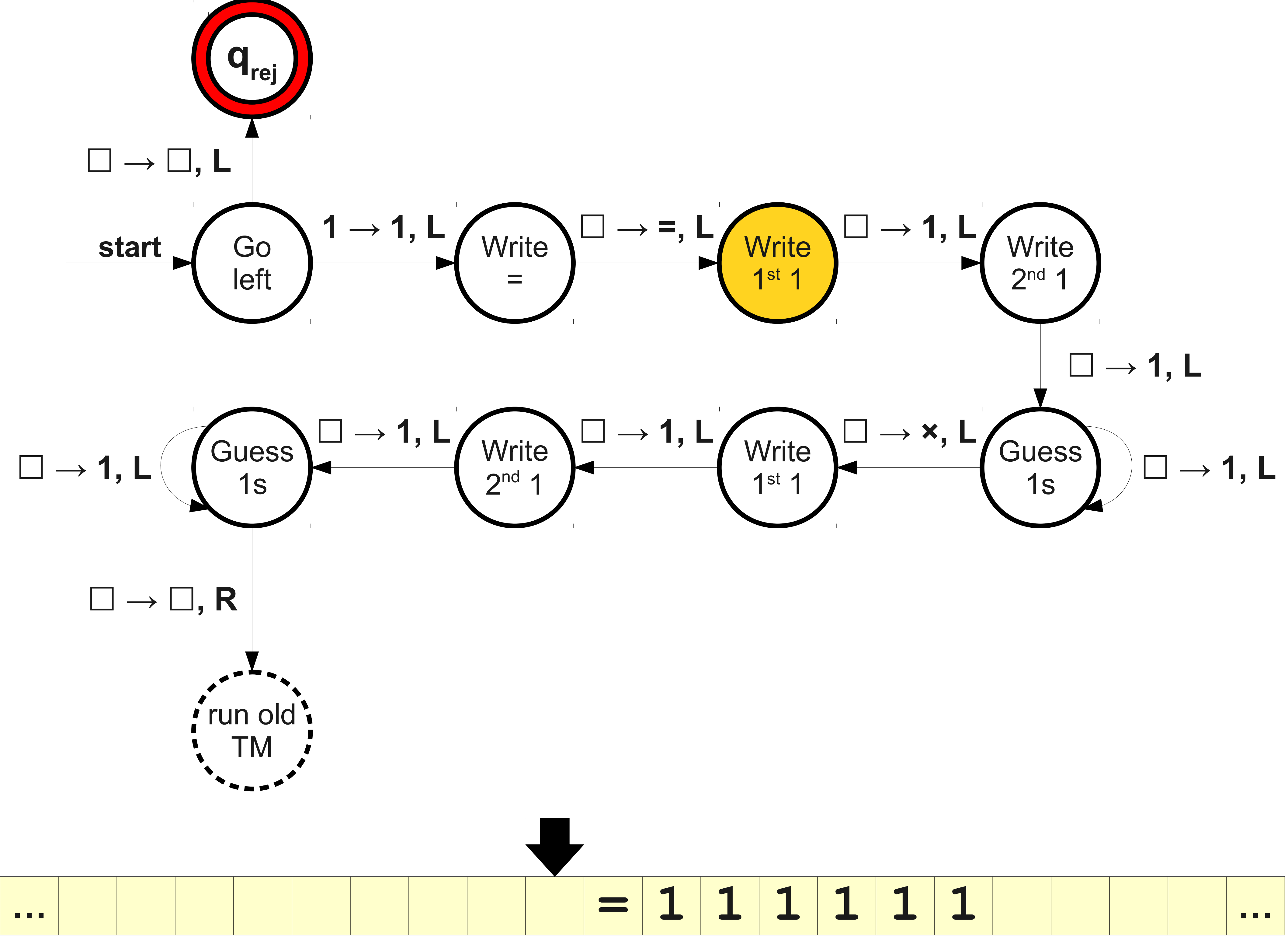
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
...
1 1 1 1 1 1
...



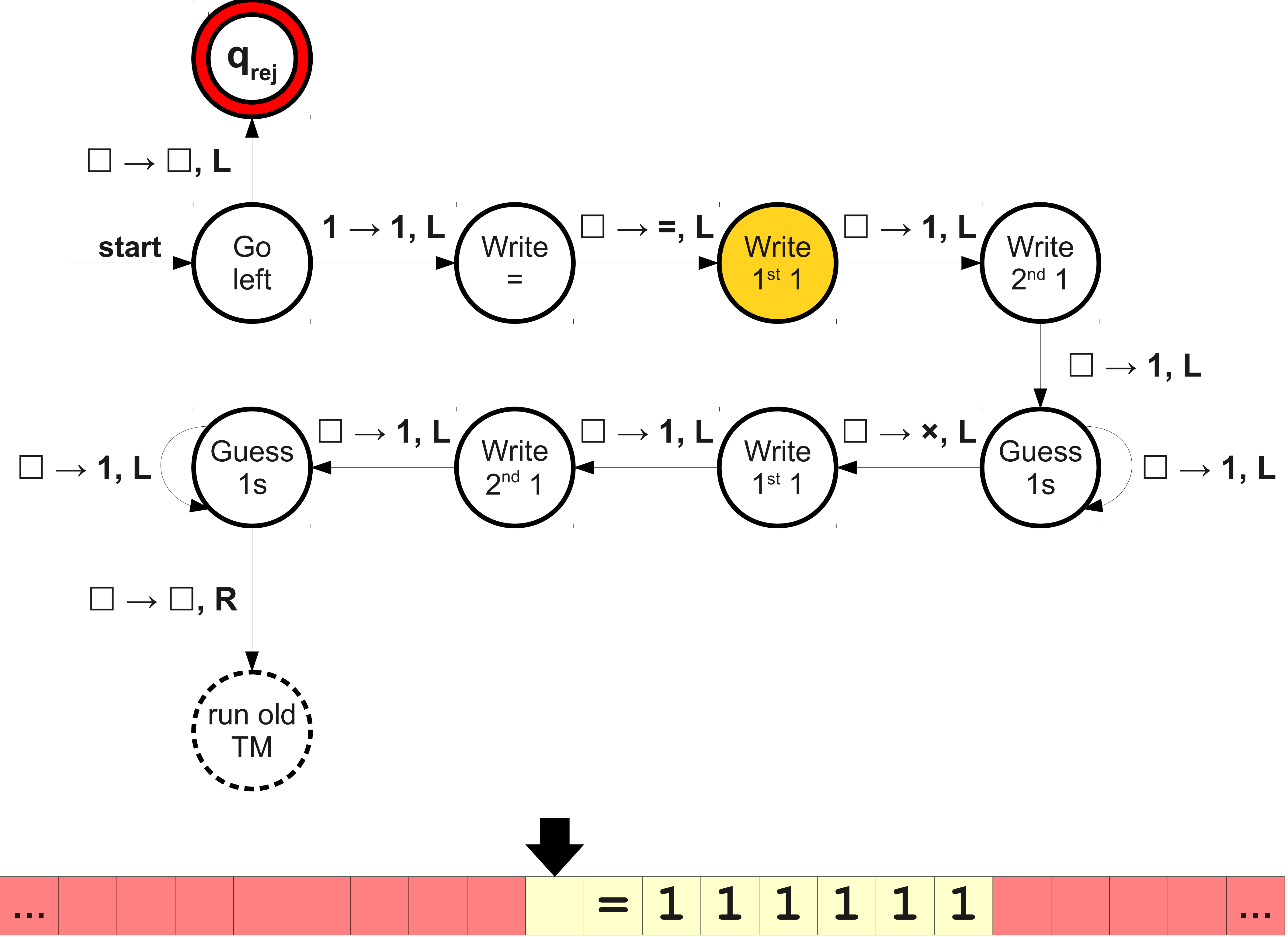
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
...
1 1 1 1 1 1
...

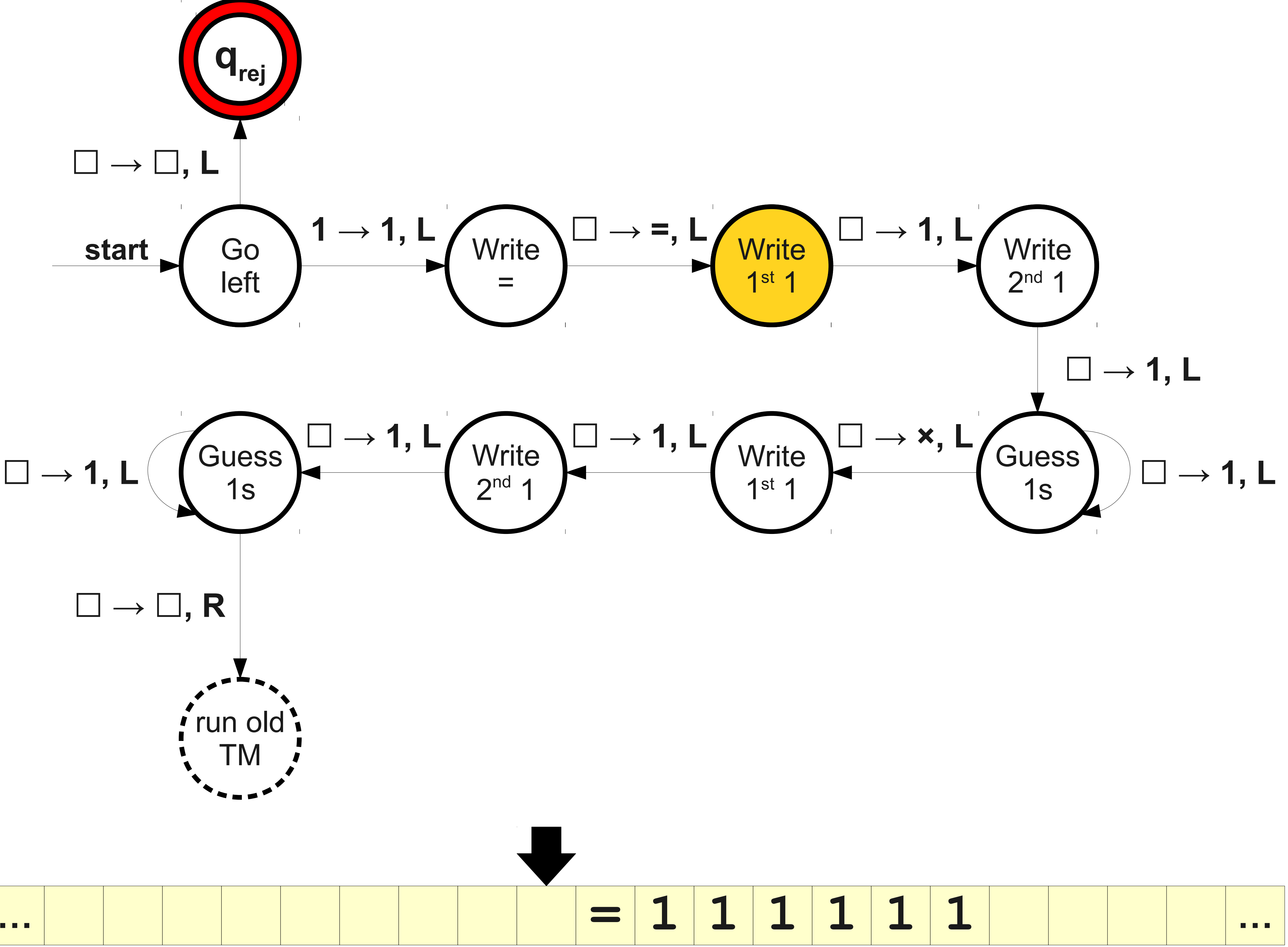


q_{rej}
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
...
1 1 1 1 1 1
...

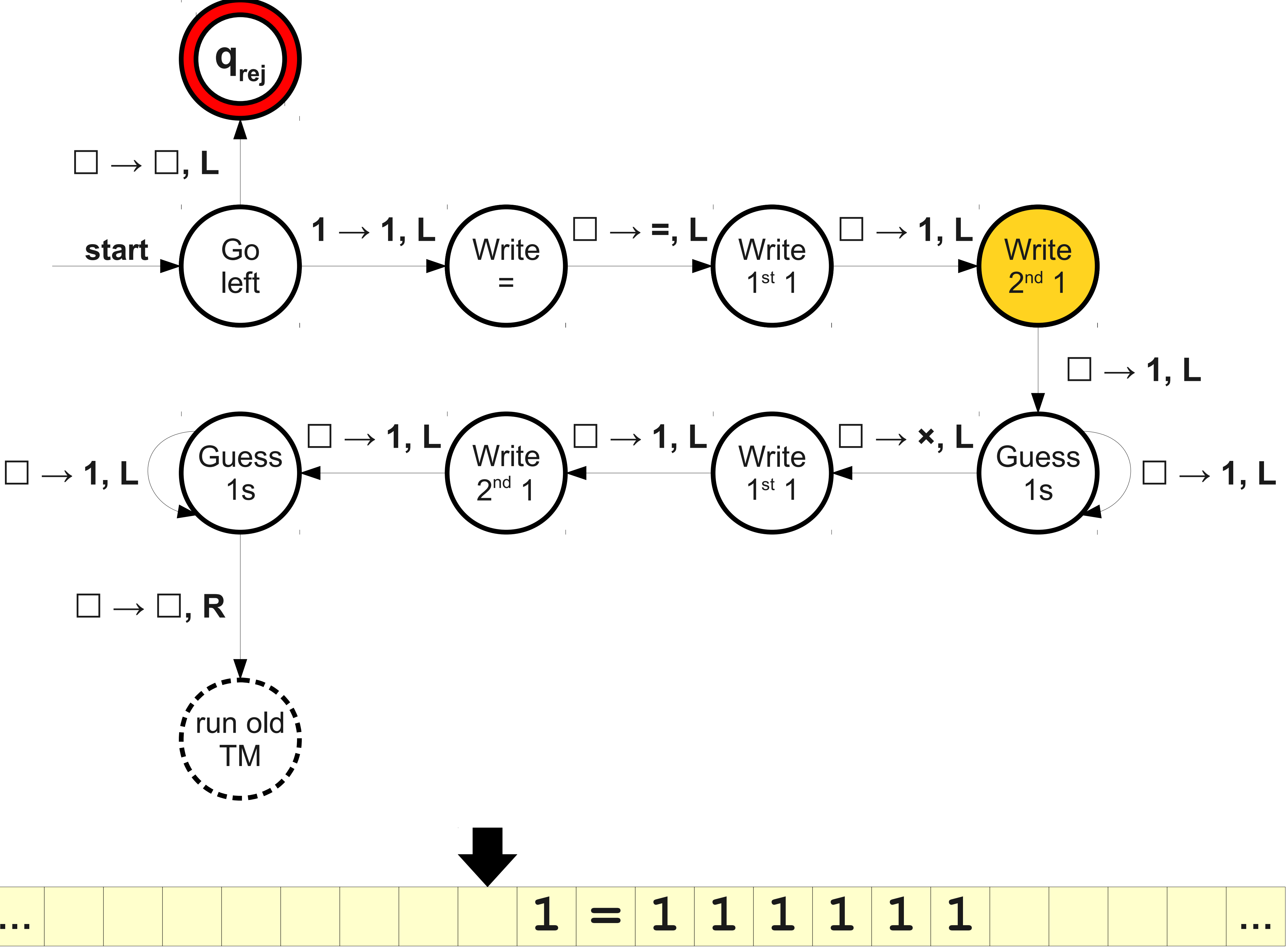


q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
... = 1 1 1 1 1 1 ...

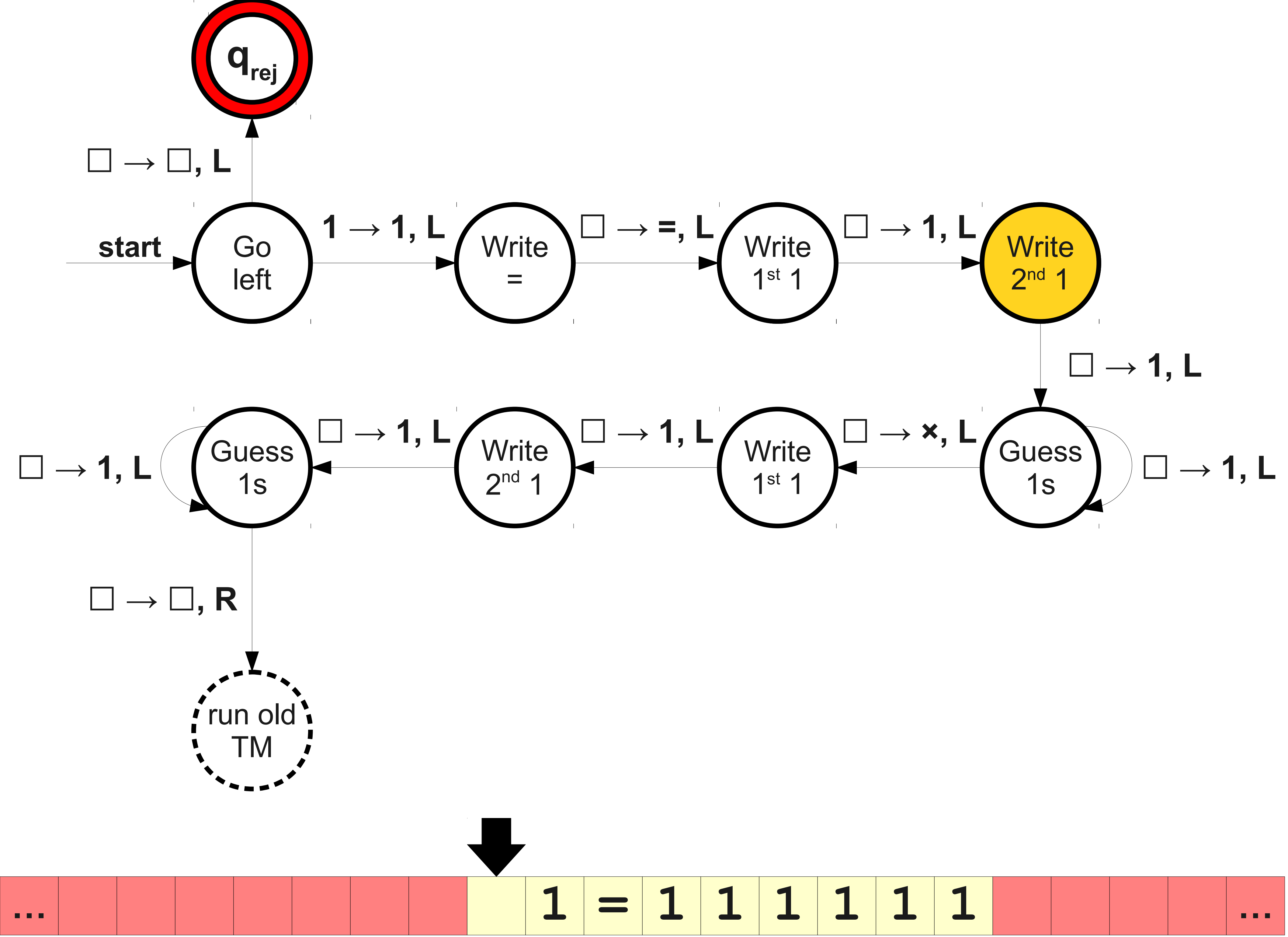

q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
…
= 1 1 1 1 1 1
…



q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
… = 1 1 1 1 1 1 …



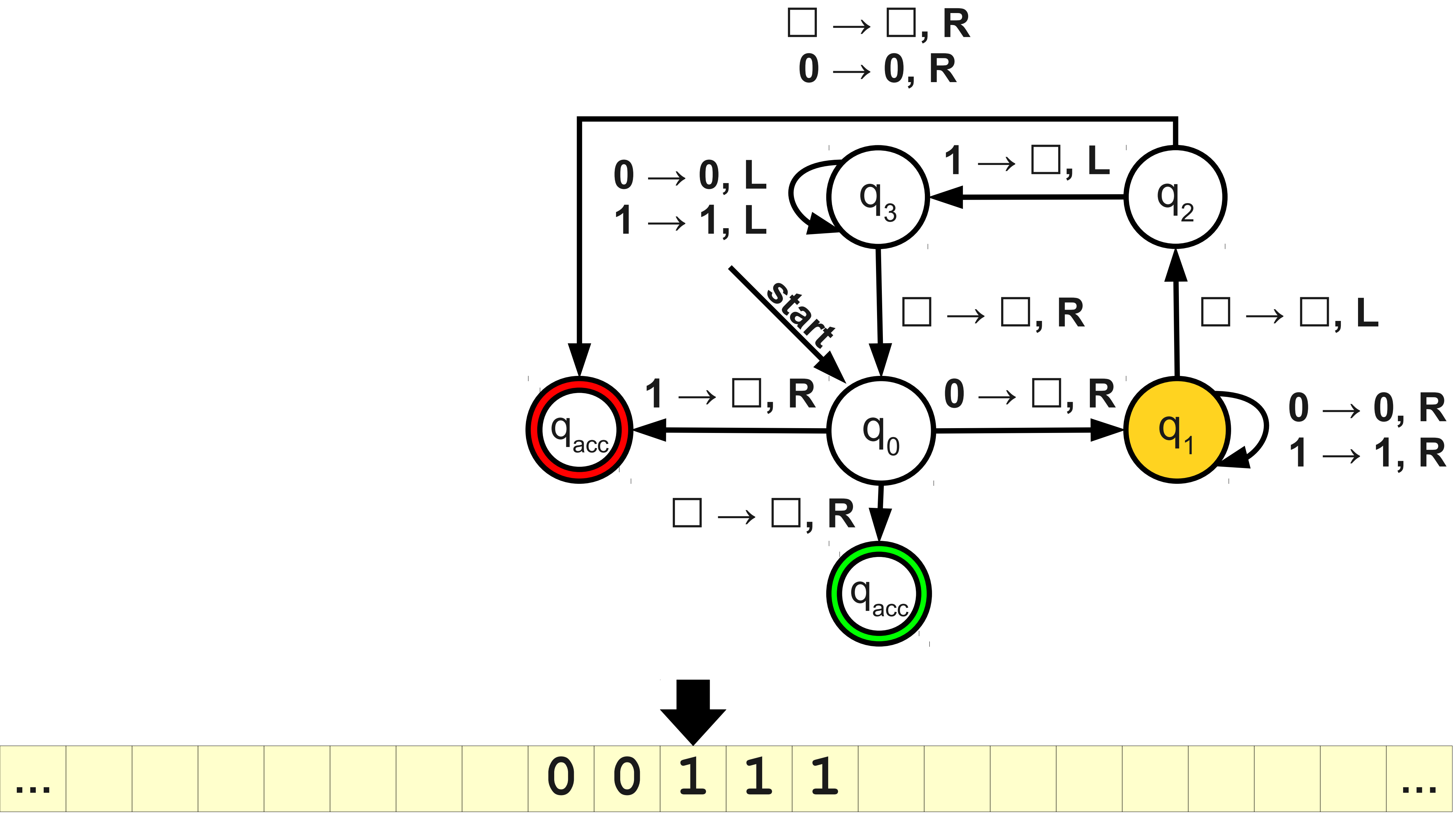
q_rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
... 1 = 1 1 1 1 1 1 ...


q rej
□ → □, L
start
Go left
1 → 1, L
Write =
□ → =, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → ×, L
Write 1st 1
□ → 1, L
Write 2nd 1
□ → 1, L
Guess 1s
□ → 1, L
□ → □, R
run old TM
... 1 = 1 1 1 1 1 1 ...

Instantaneous Descriptions

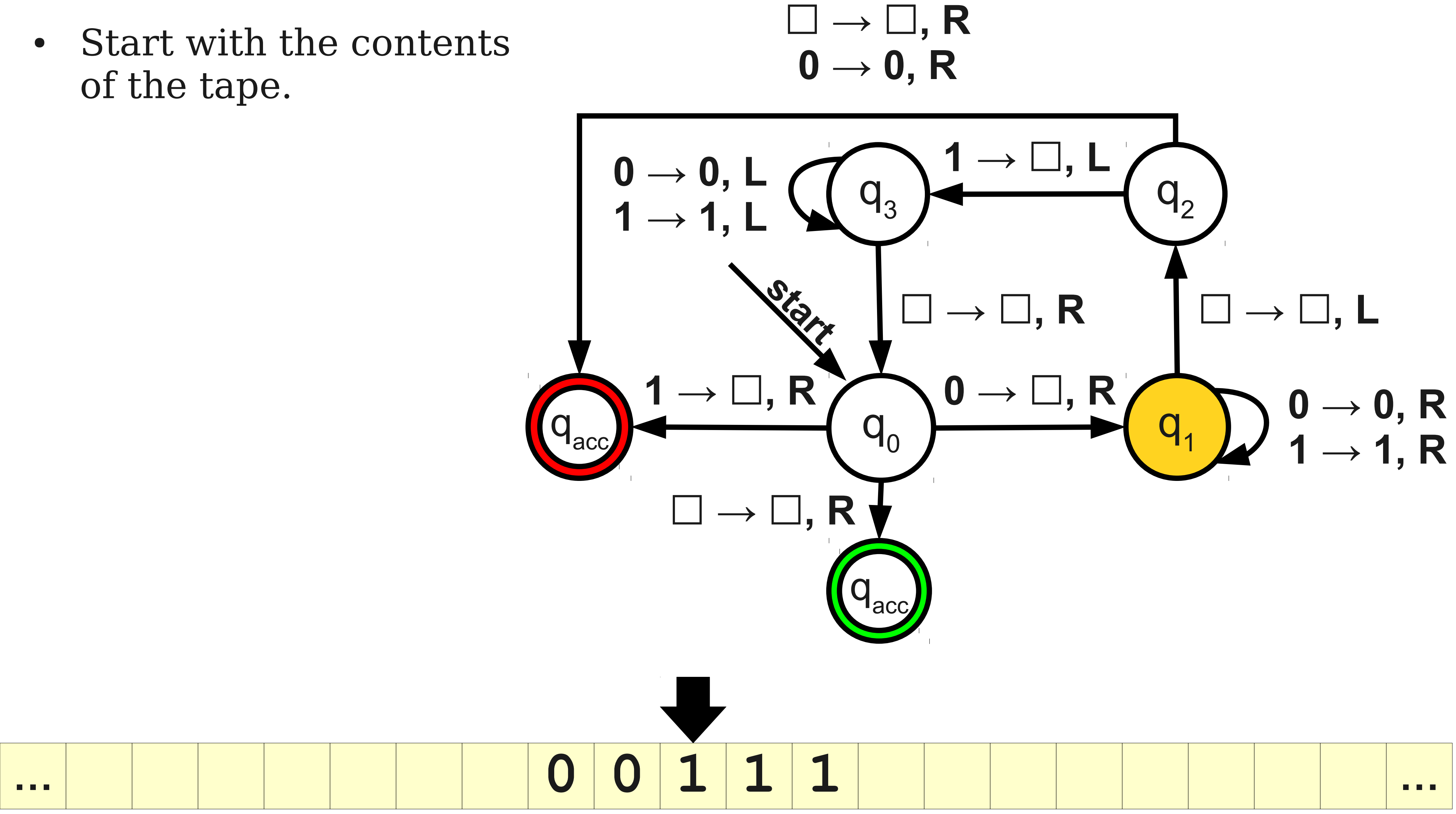
- At any instant in time, only finitely many cells on the TM's tape may be non-blank.
- An **instantaneous description** (ID) of a Turing machine is a string representation of the TM's tape, state, and tape head location.
  - Only store the “interesting” part of the tape.
- There are many ways to encode an ID; we'll see one in a second.

Building an ID



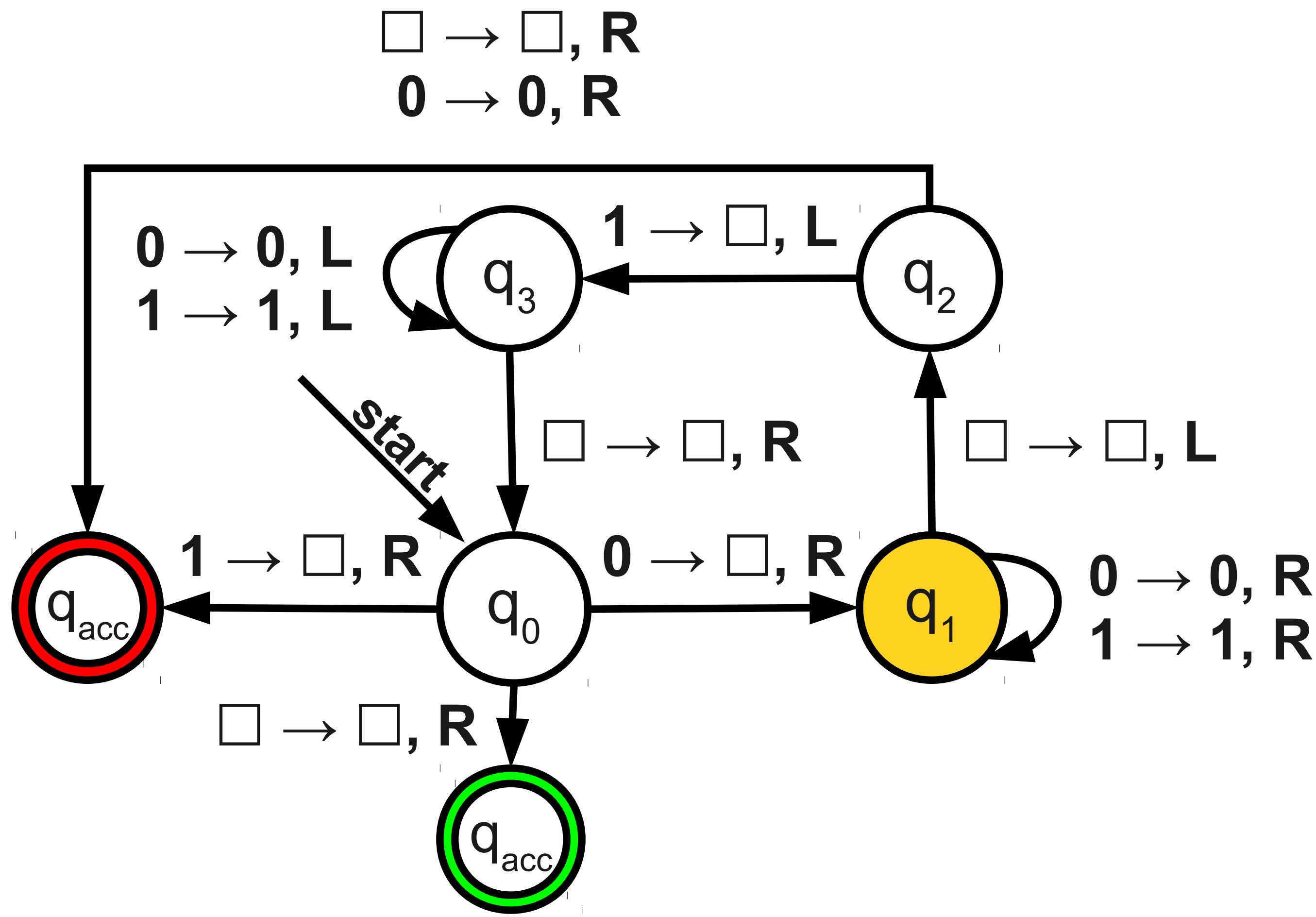
Building an ID

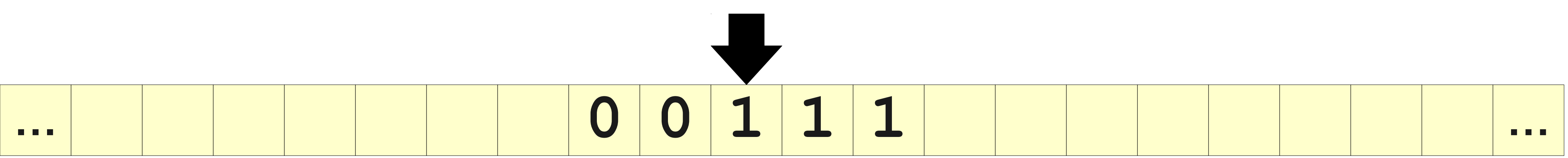
- Start with the contents of the tape.



Building an ID

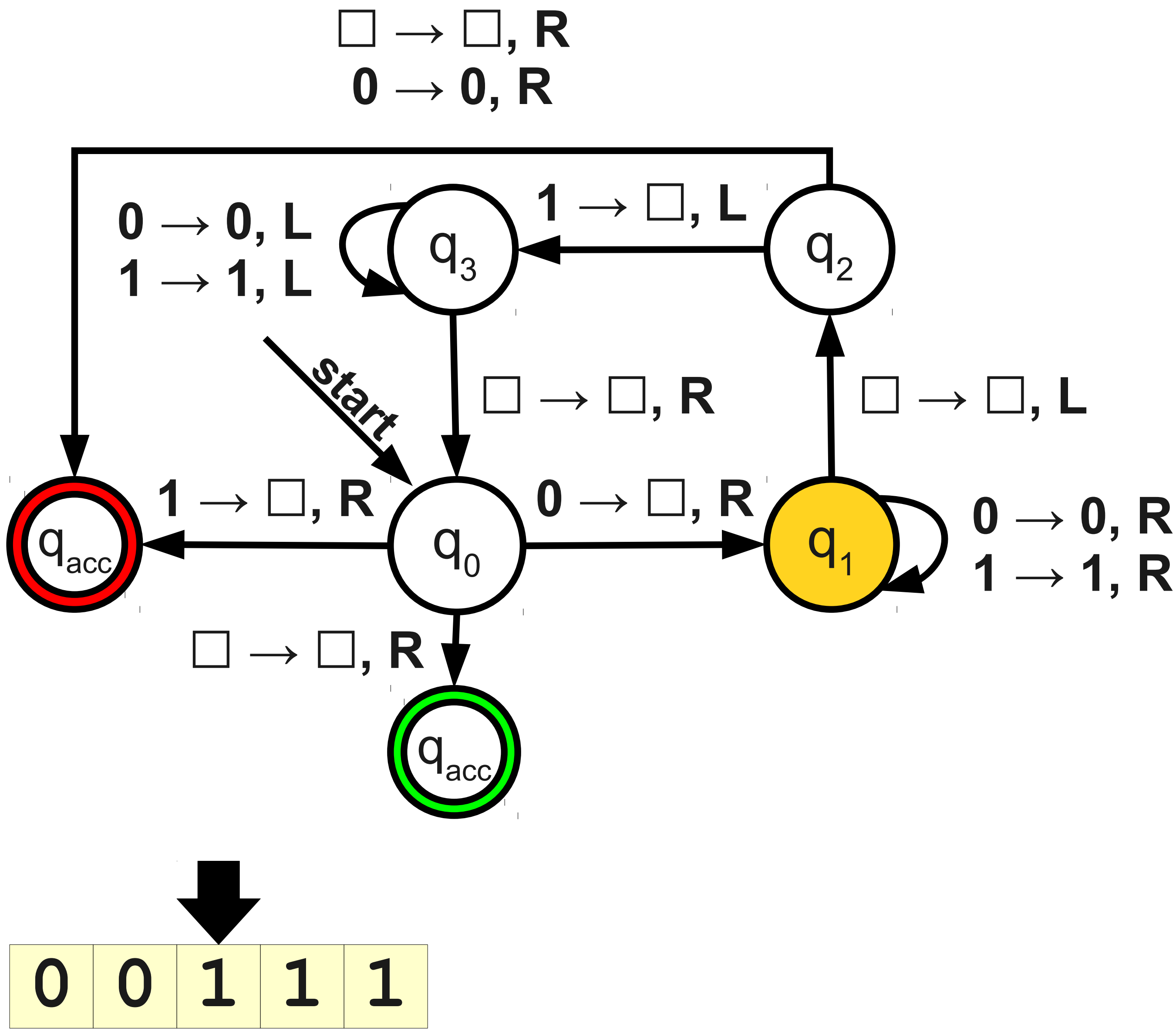
- Start with the contents of the tape.
- Trim "uninteresting" blank symbols from the ends of the tape (though remember blanks under the tape head).





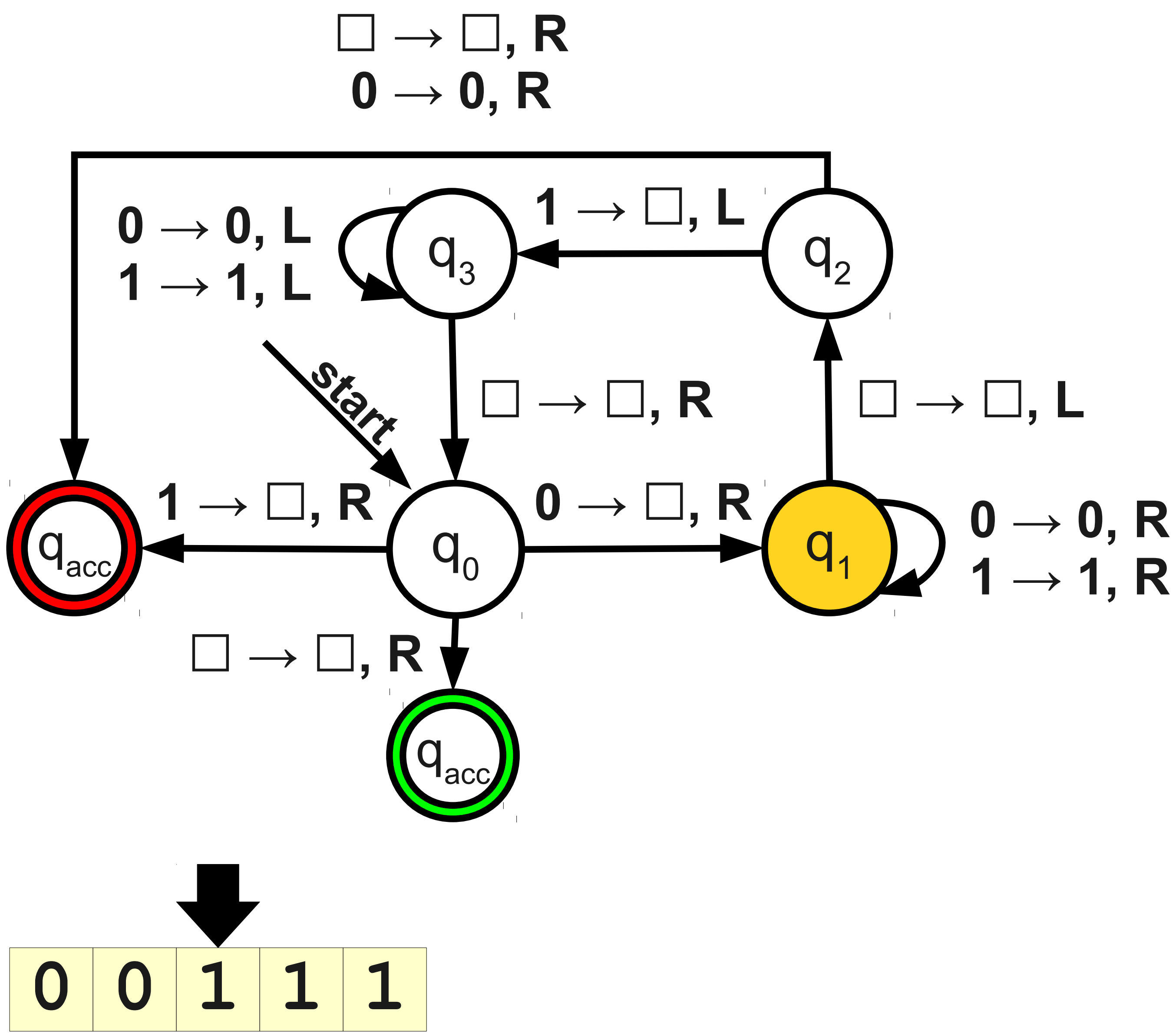
Building an ID

- Start with the contents of the tape.
- Trim "uninteresting" blank symbols from the ends of the tape (though remember blanks under the tape head).



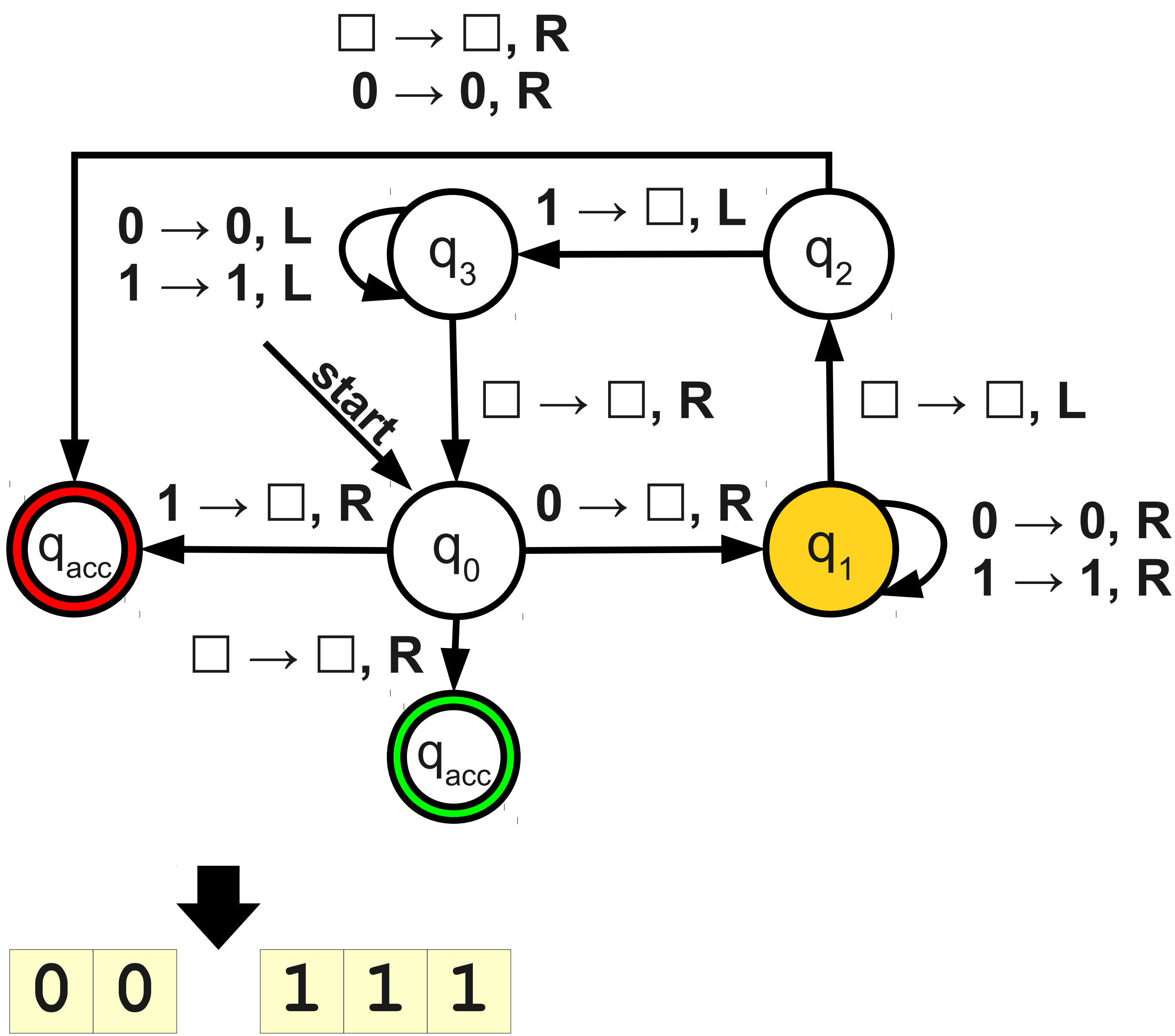
Building an ID

- Start with the contents of the tape.
- Trim “uninteresting” blank symbols from the ends of the tape (though remember blanks under the tape head).
- Insert a marker for the tape head position that encodes the current state.



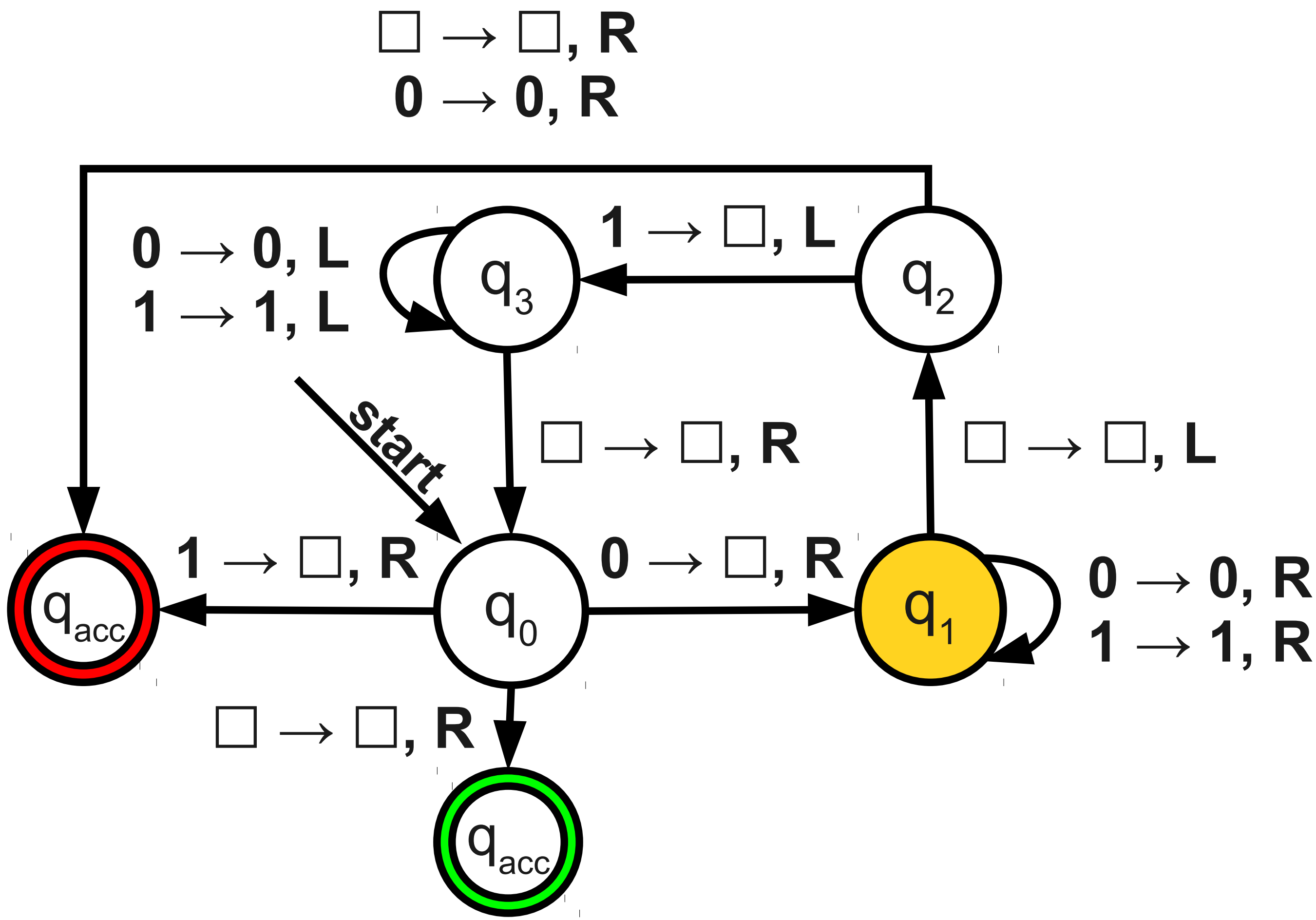
Building an ID

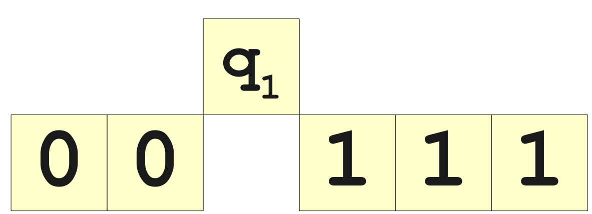
- Start with the contents of the tape.
- Trim "uninteresting" blank symbols from the ends of the tape (though remember blanks under the tape head).
- Insert a marker for the tape head position that encodes the current state.



Building an ID

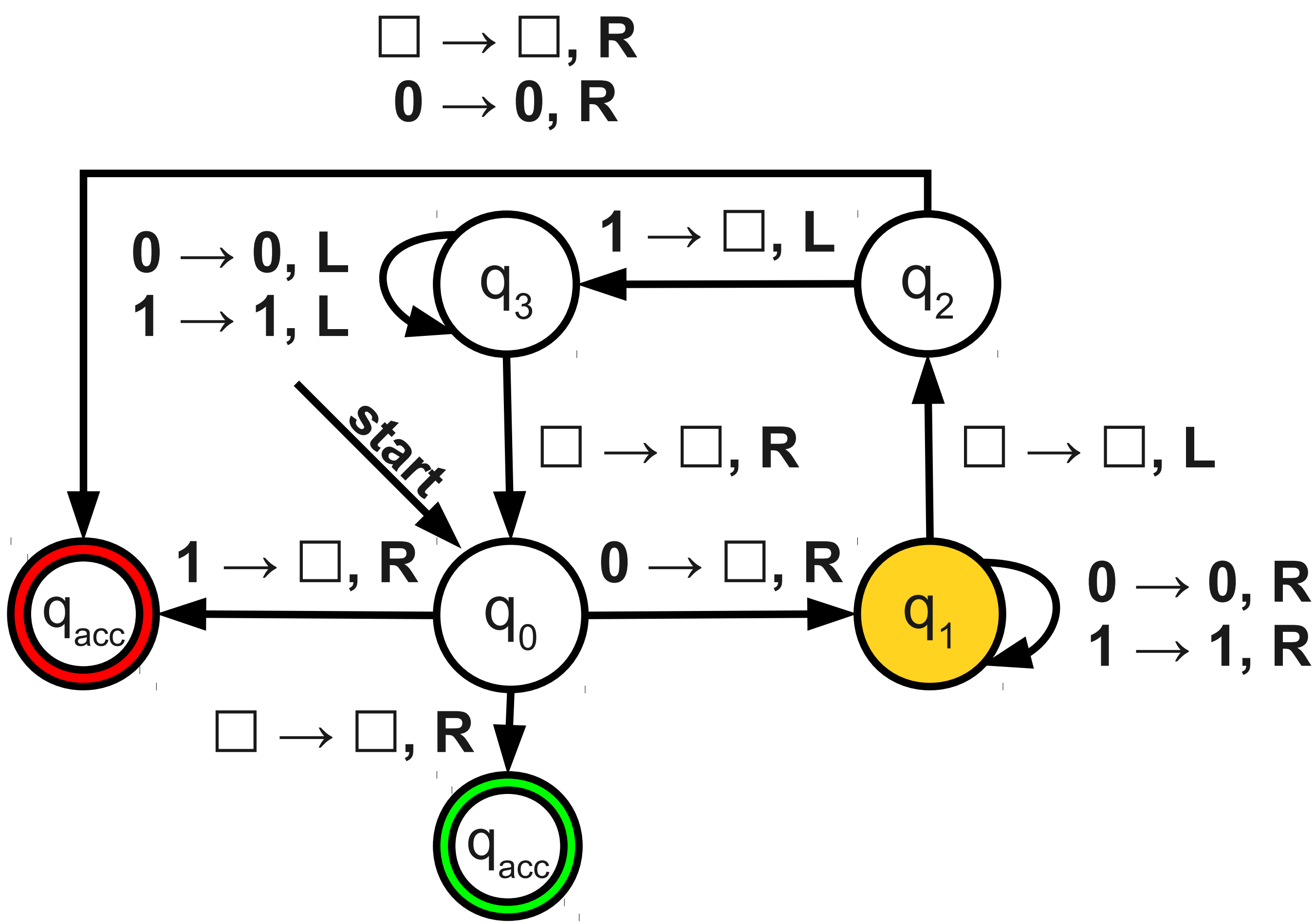
- Start with the contents of the tape.
- Trim “uninteresting” blank symbols from the ends of the tape (though remember blanks under the tape head).
- Insert a marker for the tape head position that encodes the current state.





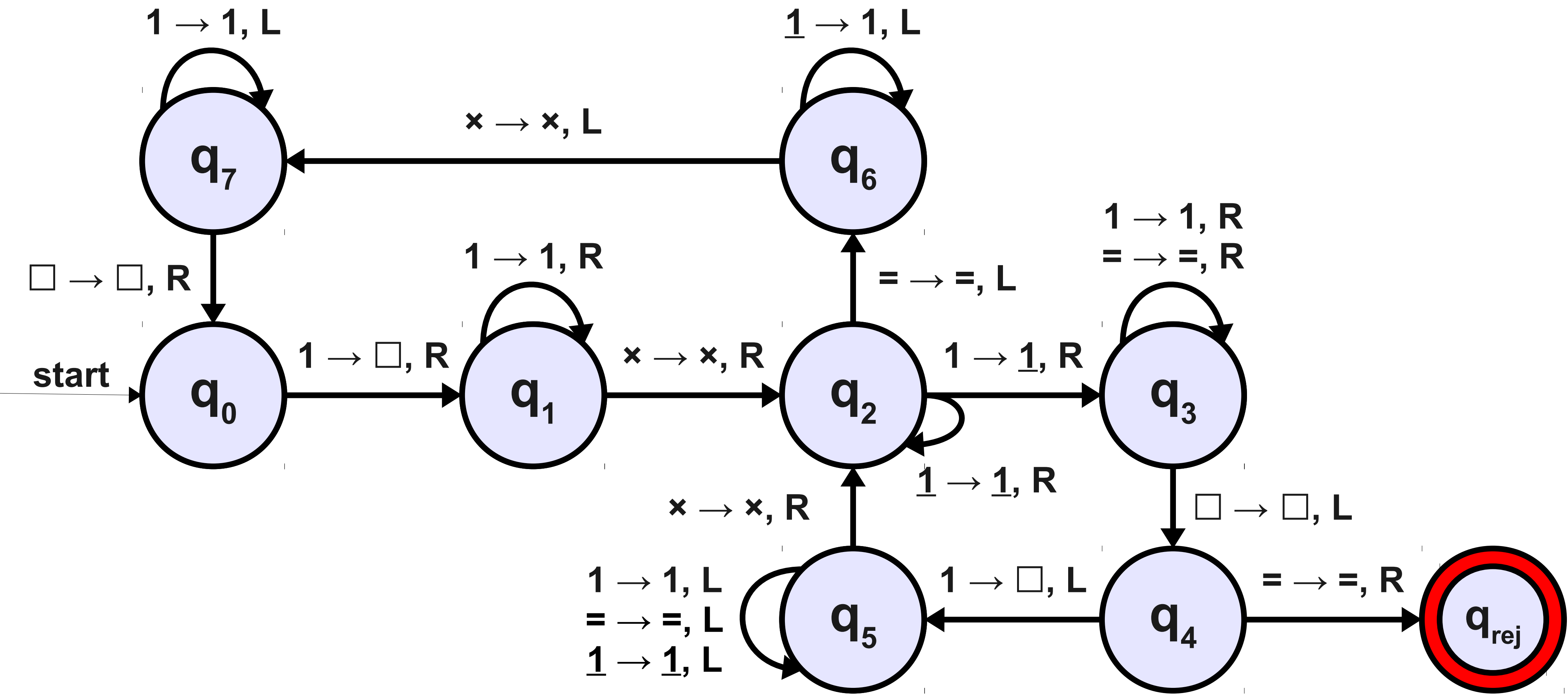
Building an ID

- Start with the contents of the tape.
- Trim "uninteresting" blank symbols from the ends of the tape (though remember blanks under the tape head).
- Insert a marker for the tape head position that encodes the current state.

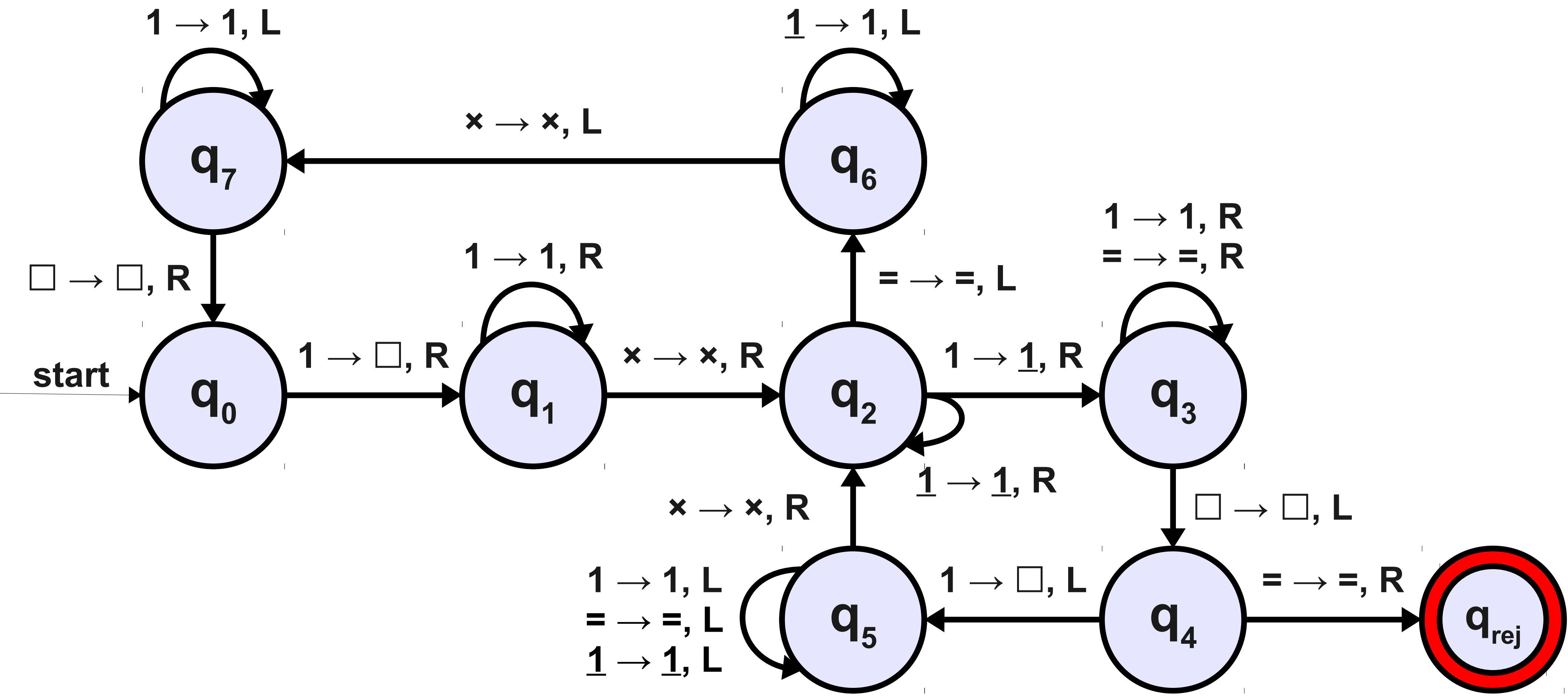


0	0	q_1	1	1	1

Rehydrating IDs

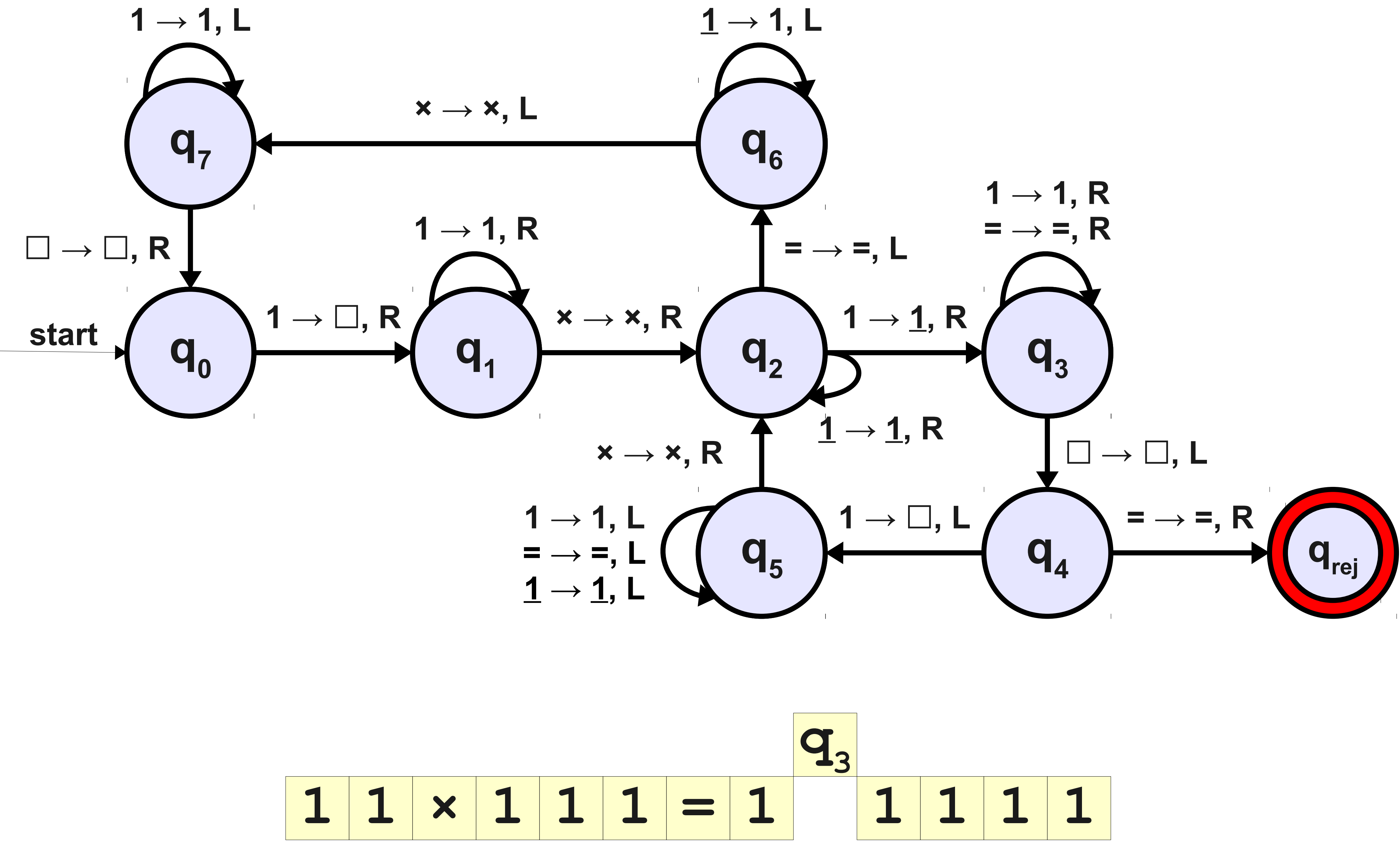


Rehydrating IDs

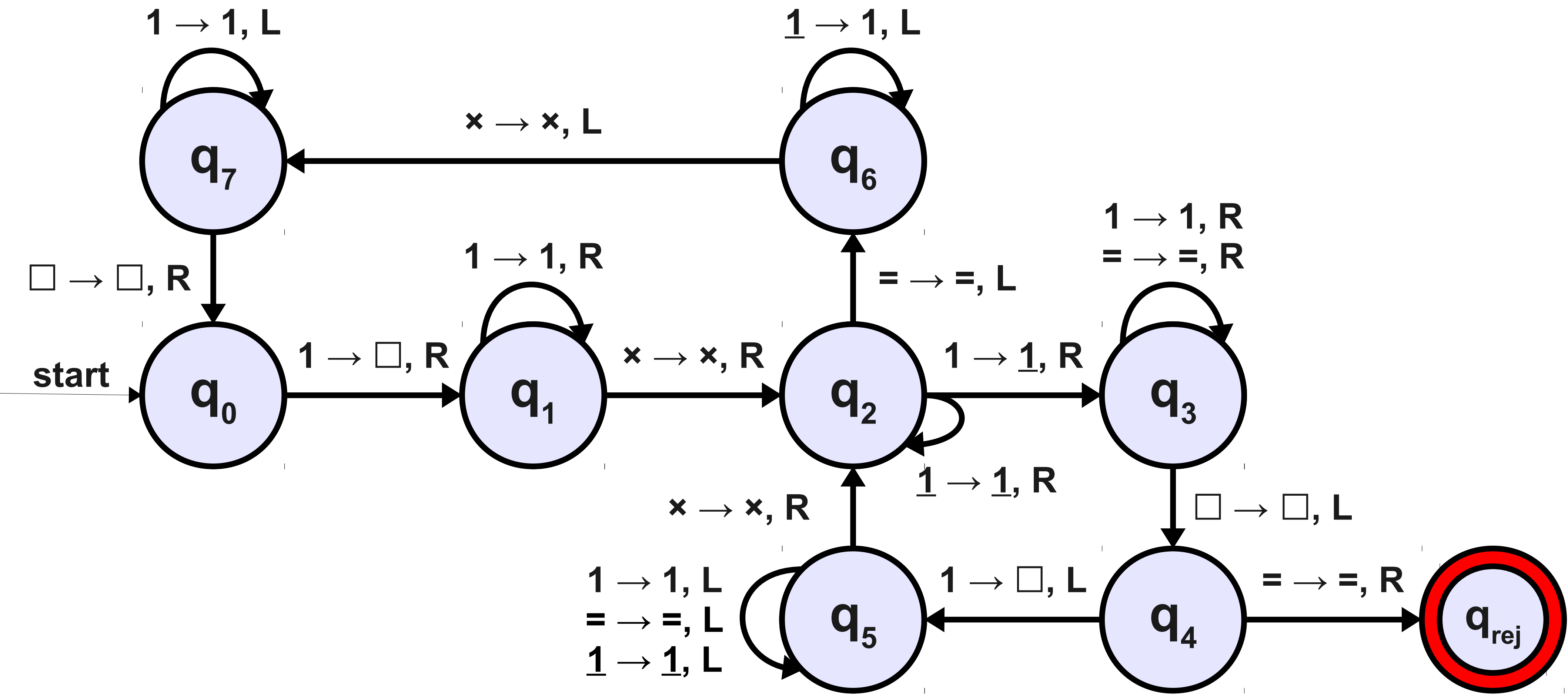


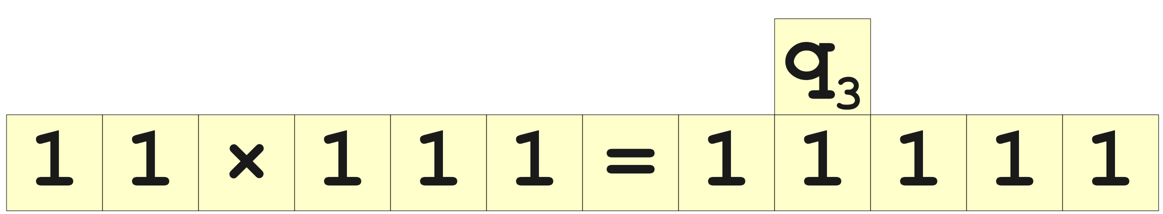
1	1	×	1	1	1	=	1	q_3	1	1	1	1

Rehydrating IDs

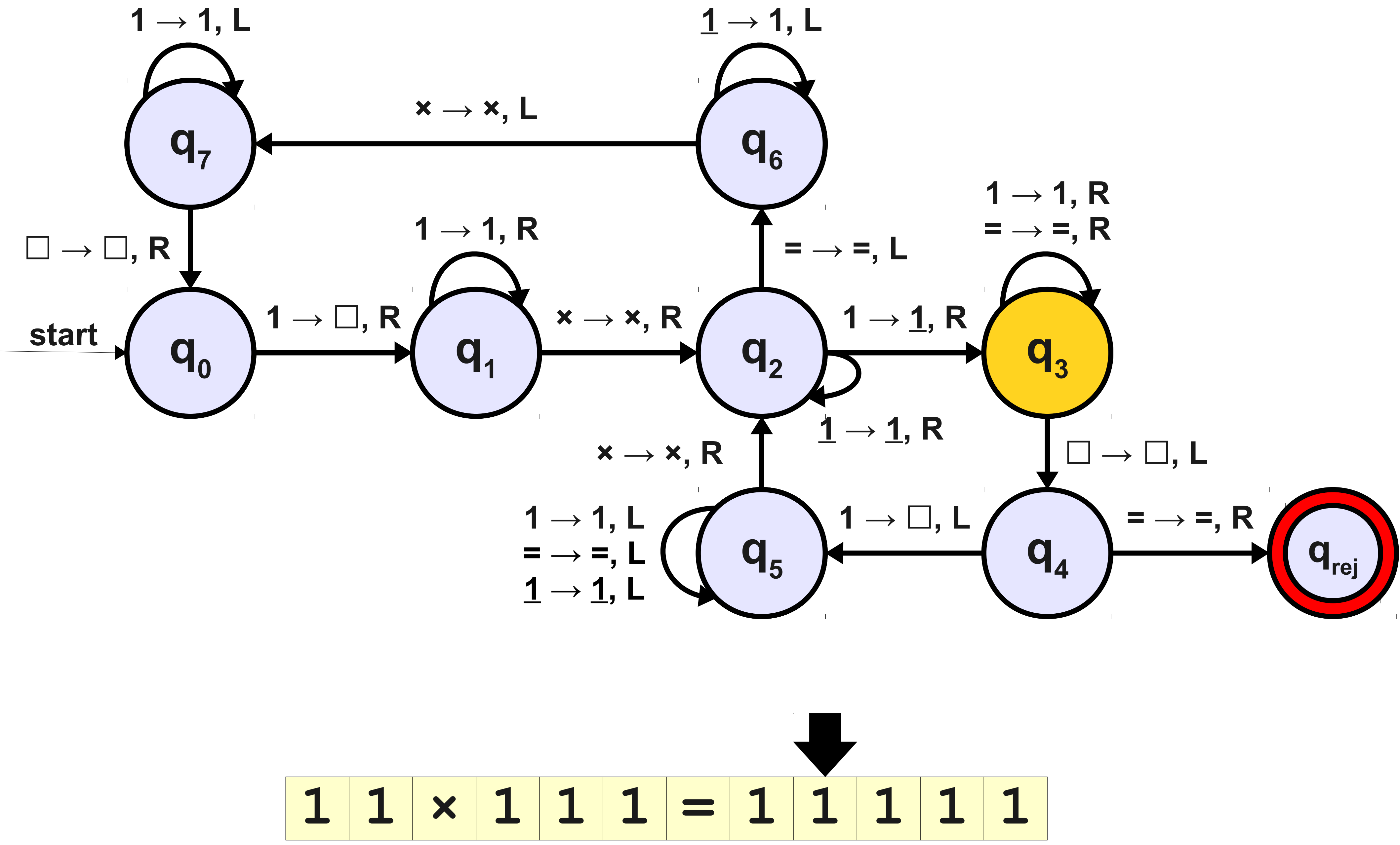


Rehydrating IDs

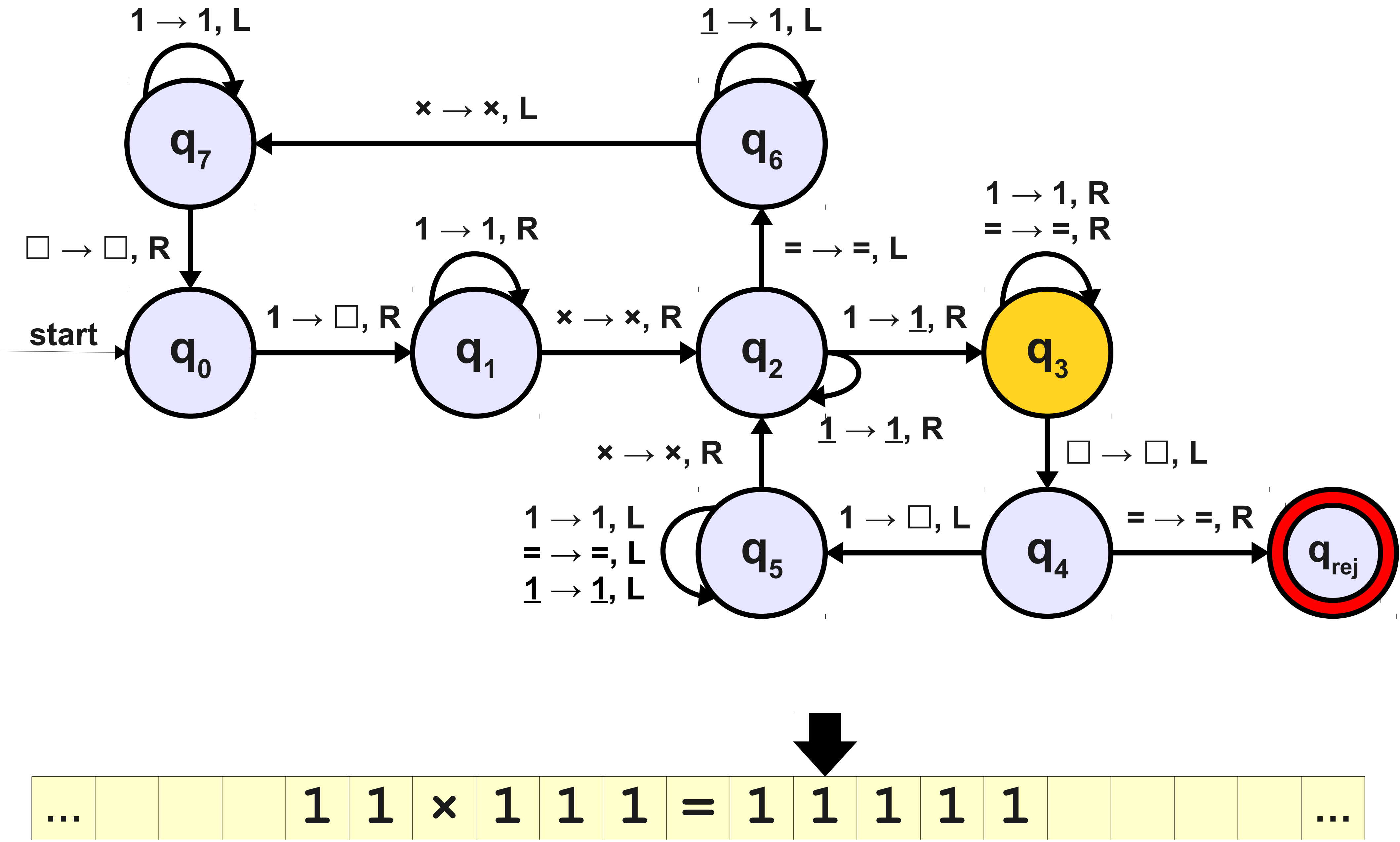




Rehydrating IDs



Rehydrating IDs



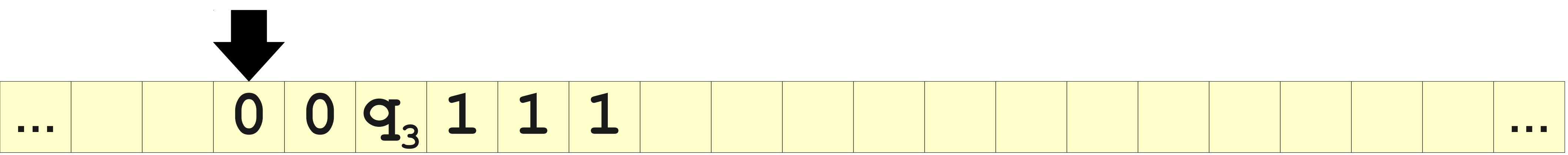
Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

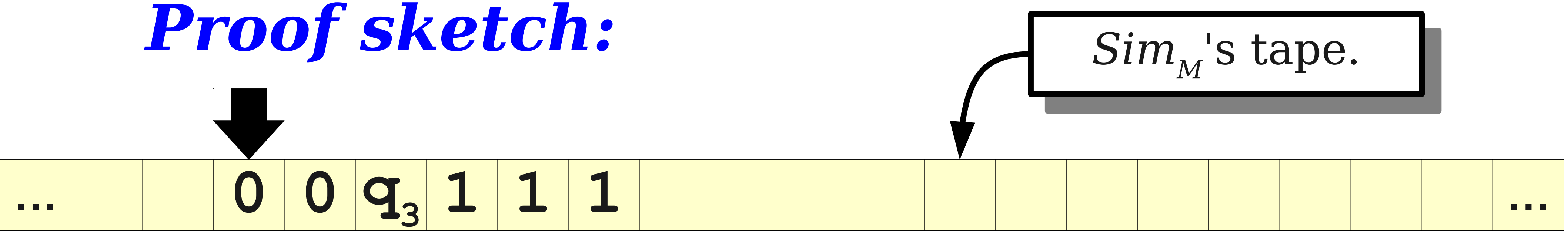
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

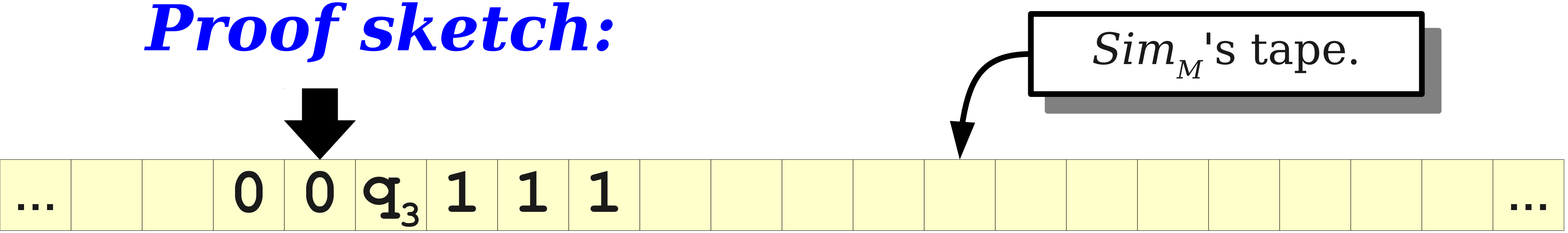
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

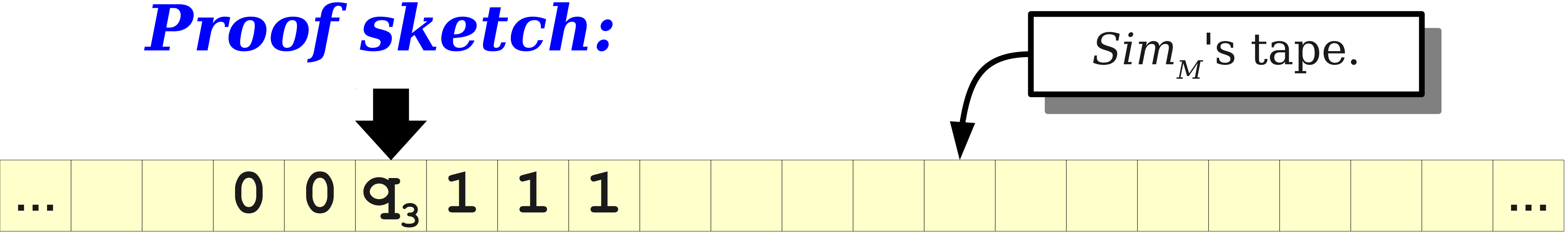
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

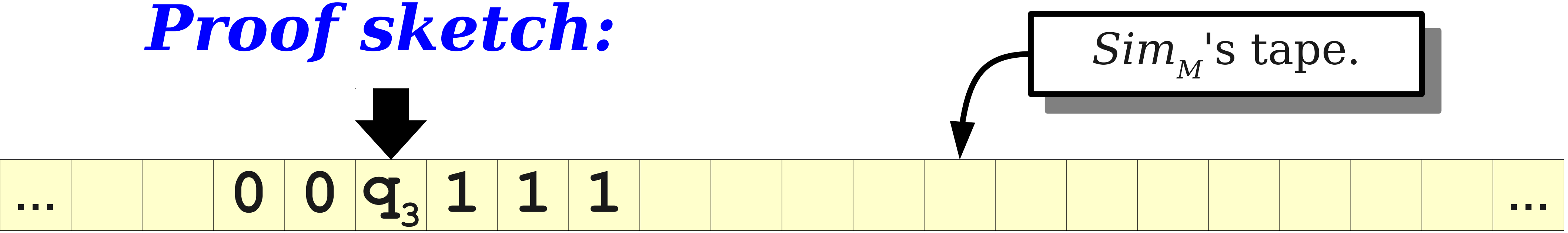
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

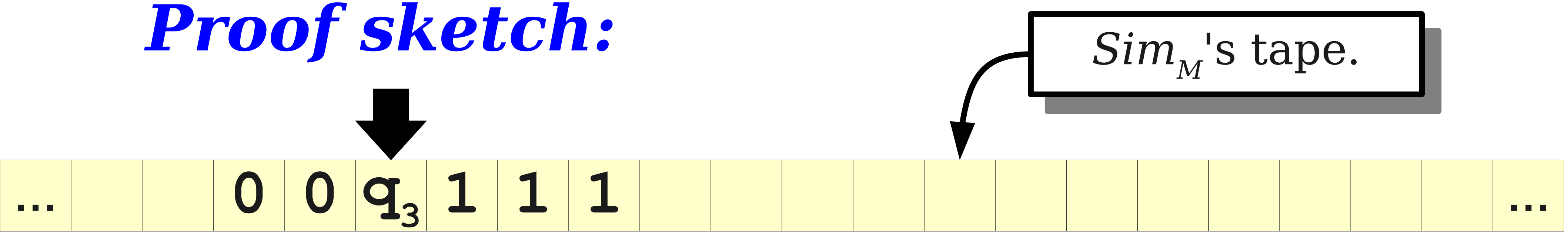


At this point, $\boldsymbol{Sim_M}$ can remember that it's seen state $\boldsymbol{q_3}$. There are only finitely many possible states.

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

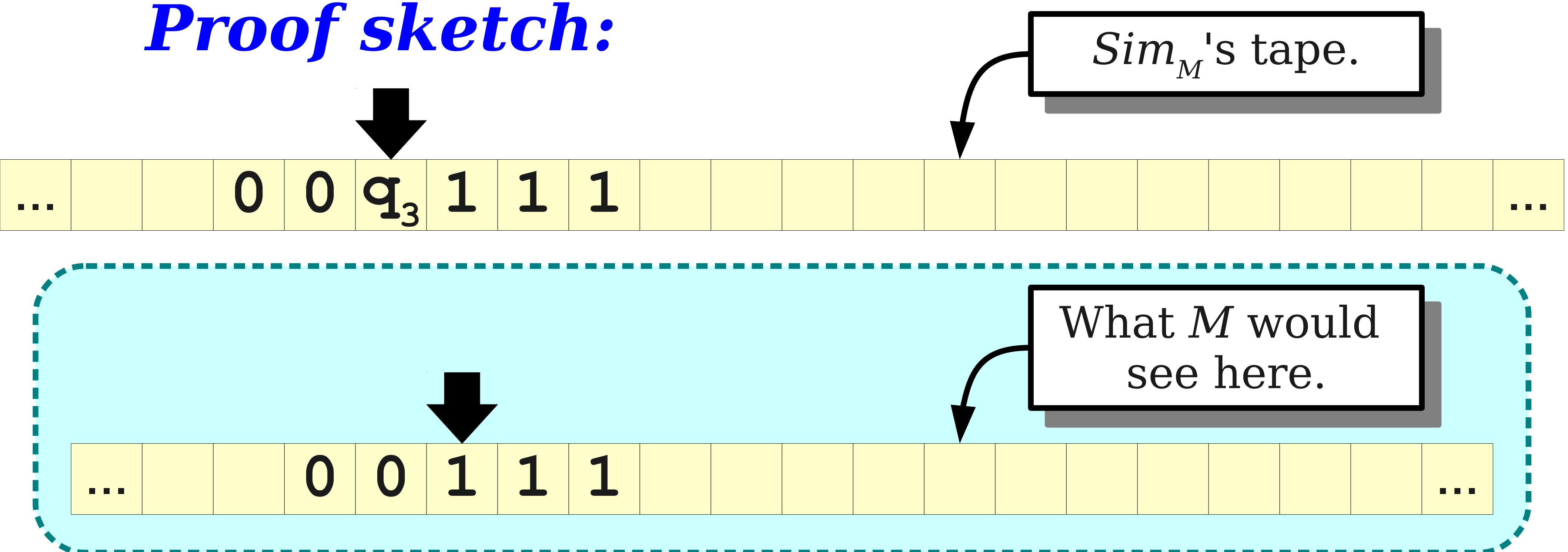
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

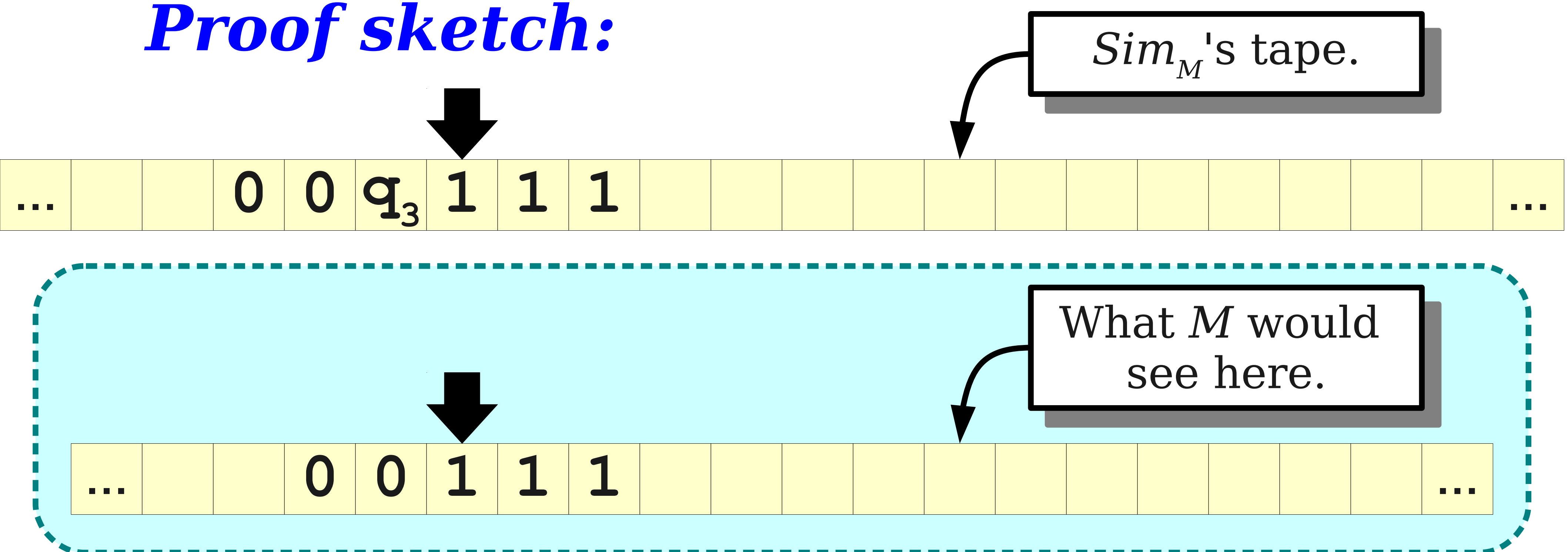
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

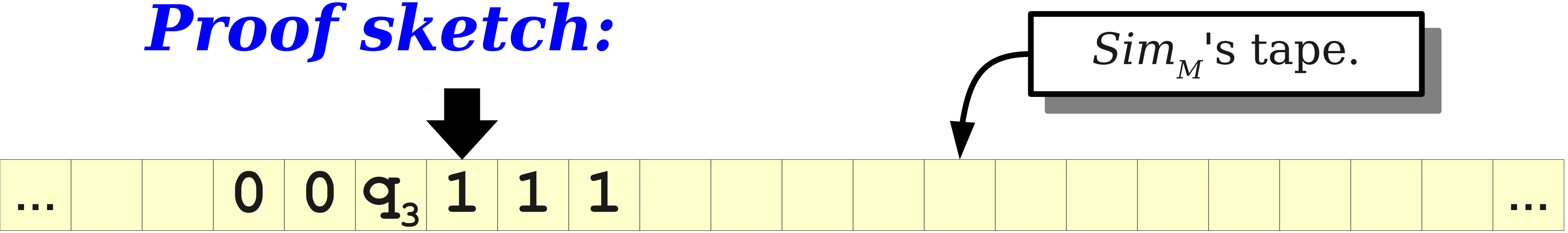
Proof sketch:



Manipulating IDs

Theorem: For any TM *M*, it is possible to build a second TM Sim_M that, given a a tape containing an ID of *M*, simulates one step of *M*'s computation.

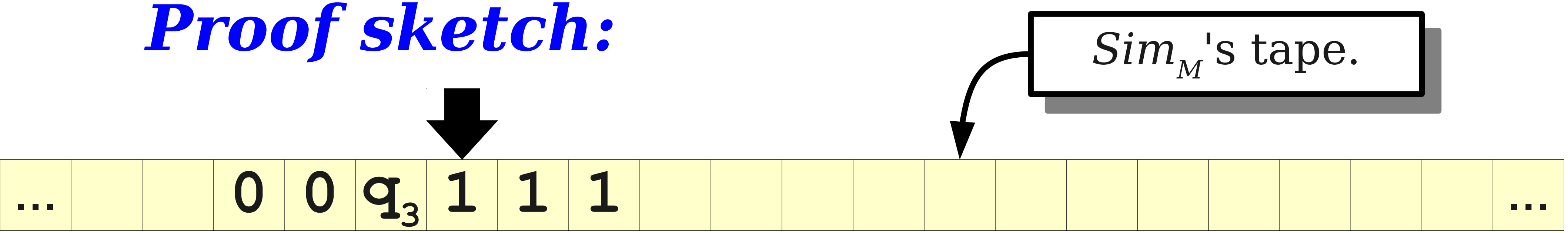
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

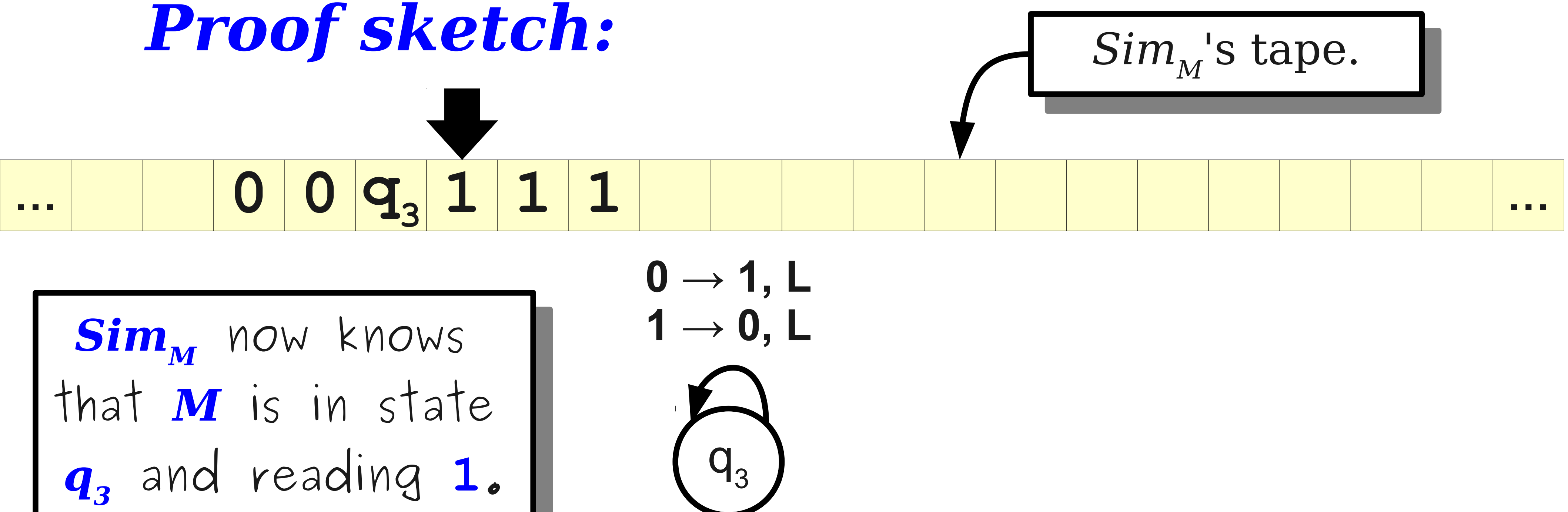


Sim_M now knows that **M** is in state **q_3** and reading **1**.

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

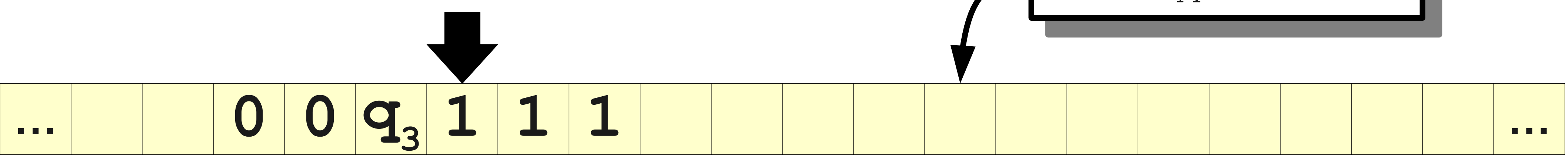


Manipulating IDs

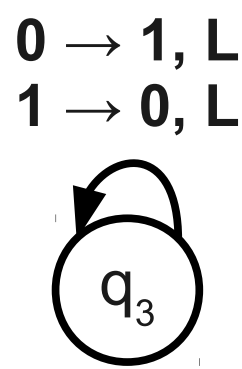
Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

Sim_M's tape.



Sim_M now knows that M is in state q_3 and reading **1**.



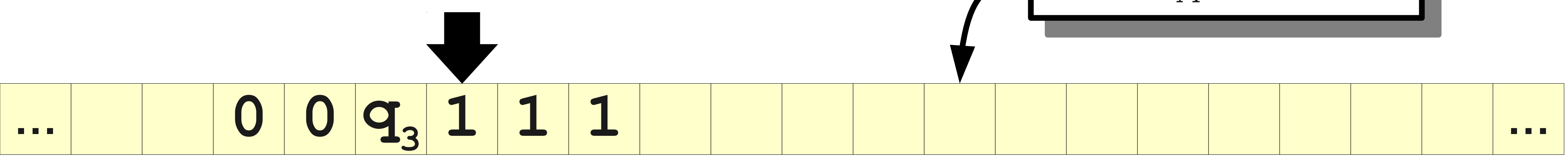
(Hypothetically, suppose that this is the state q_3 in the machine M)

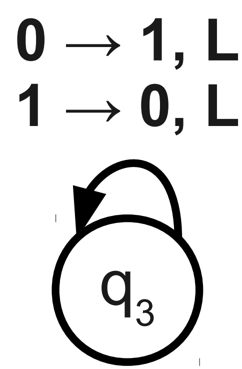
Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

Sim_M's tape.



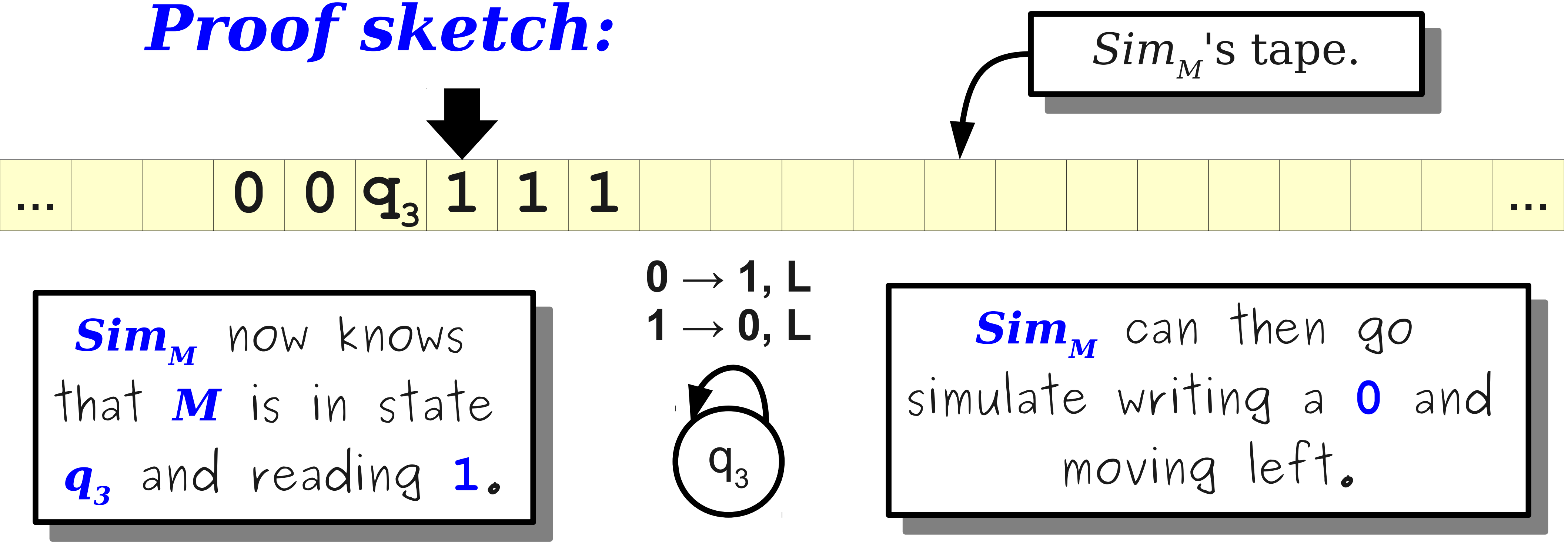


Sim_M now knows that **M** is in state **q_3** and reading **1**.

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

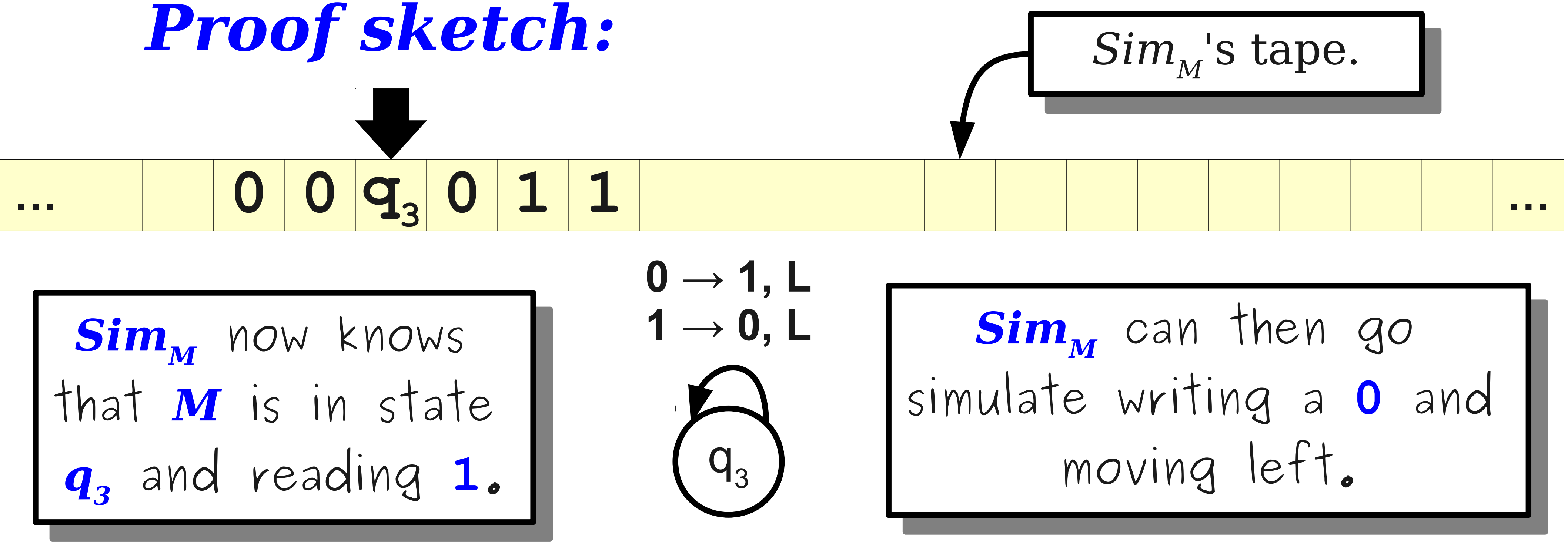
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

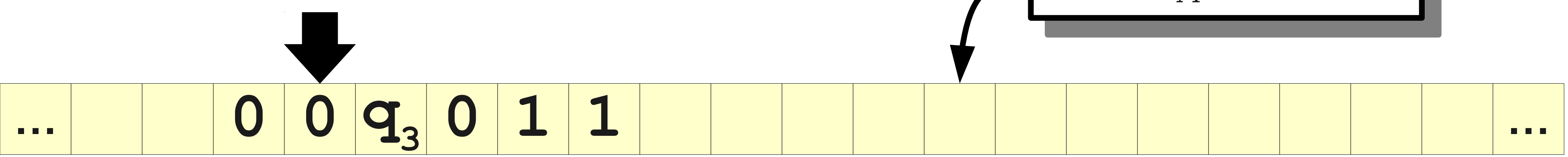


Manipulating IDs

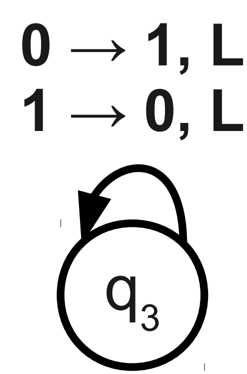
Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

Sim_M's tape.



Sim_M now knows that **M** is in state **q_3** and reading **1**.



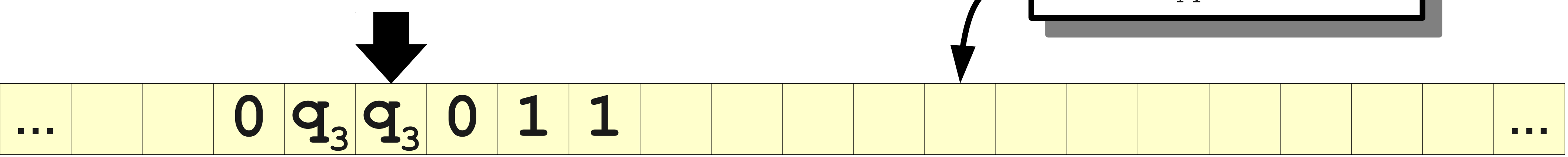
Sim_M can then go simulate writing a **0** and moving left.

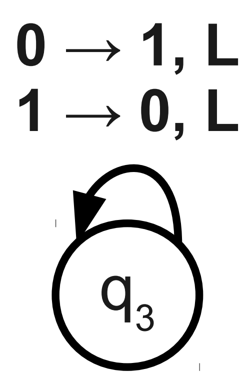
Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

Sim_M's tape.





$\boldsymbol{Sim_M}$ now knows that $\boldsymbol{M}$ is in state $\boldsymbol{q_3}$ and reading **1**.

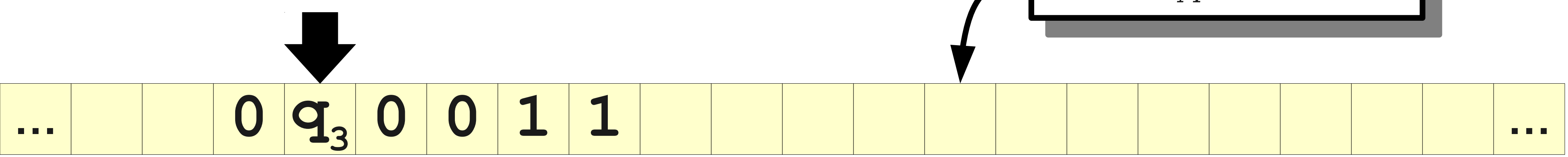
$\boldsymbol{Sim_M}$ can then go simulate writing a **0** and moving left.

Manipulating IDs

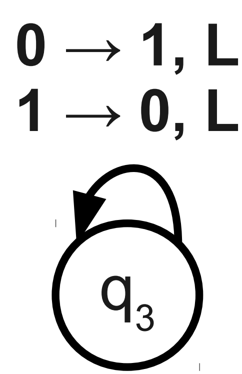
Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

Sim_M's tape.



Sim_M now knows that **M** is in state **q_3** and reading **1**.

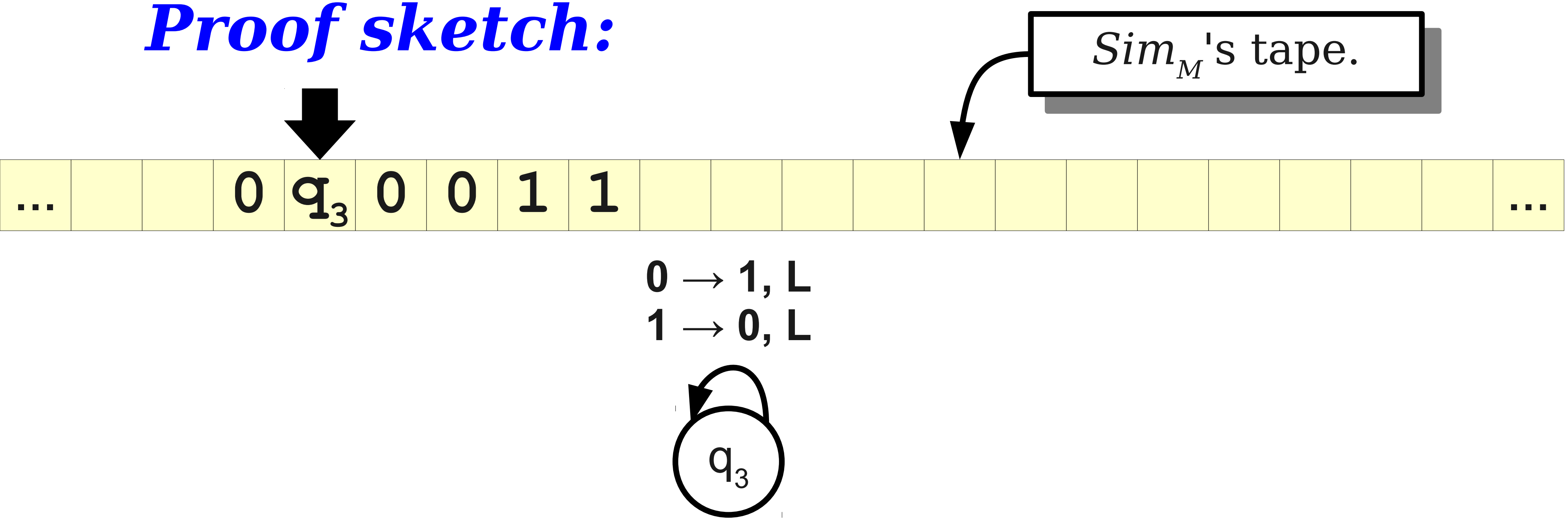


Sim_M can then go simulate writing a **0** and moving left.

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

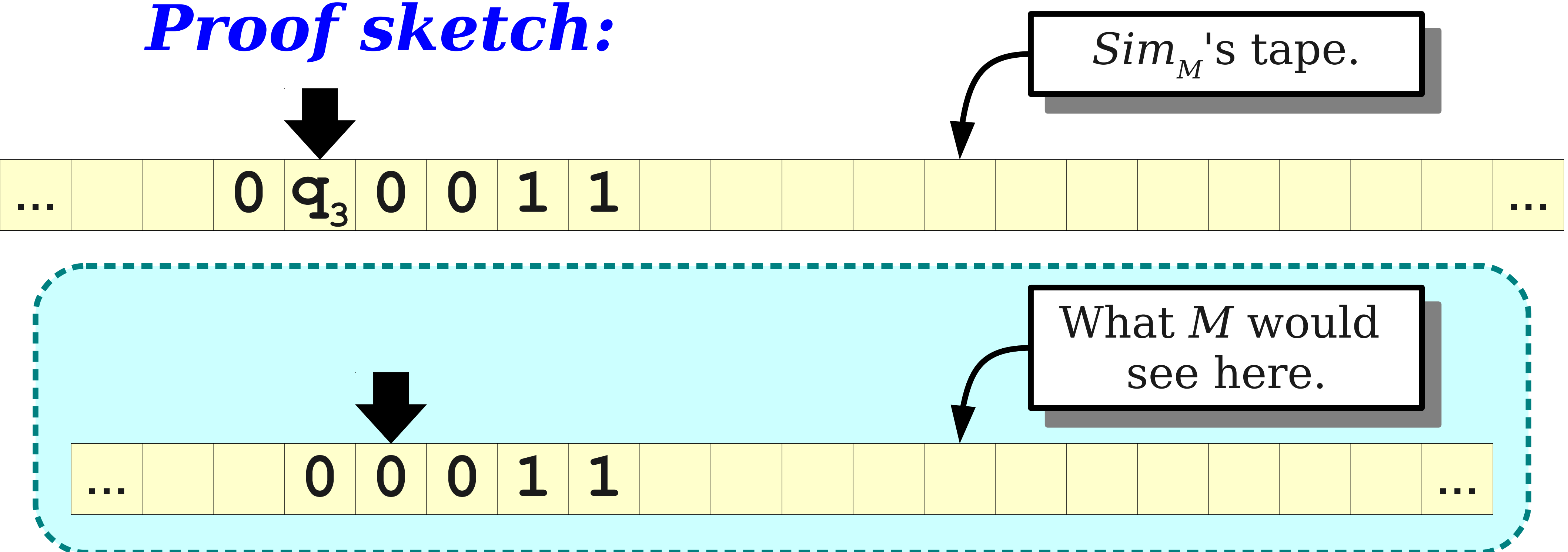
Proof sketch:



Manipulating IDs

Theorem: For any TM *M*, it is possible to build a second TM Sim_M that, given a a tape containing an ID of *M*, simulates one step of *M*'s computation.

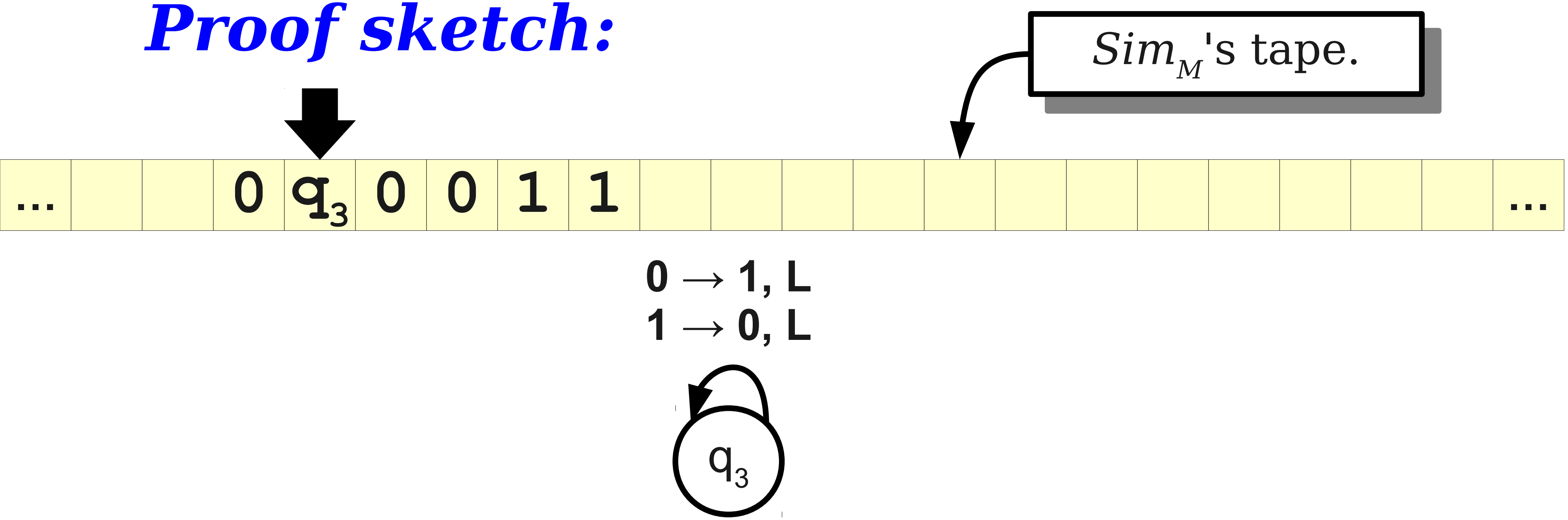
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

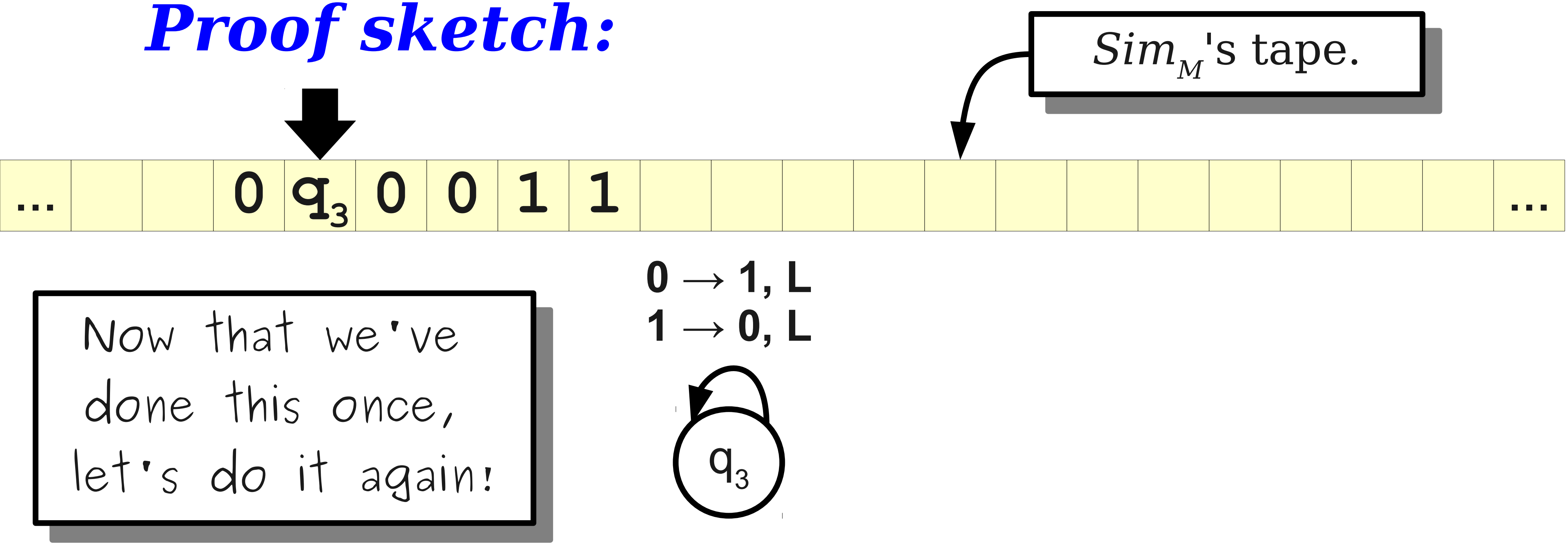
Proof sketch:



Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

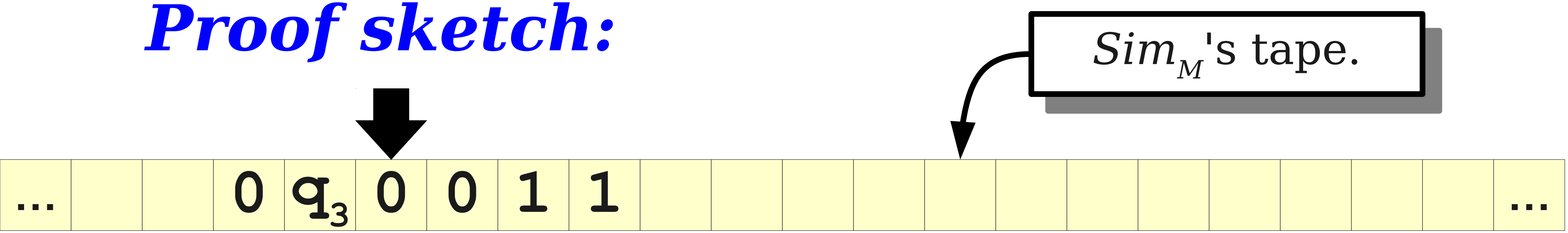
Proof sketch:

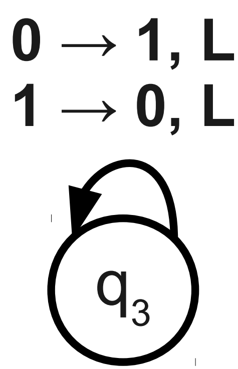


Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:



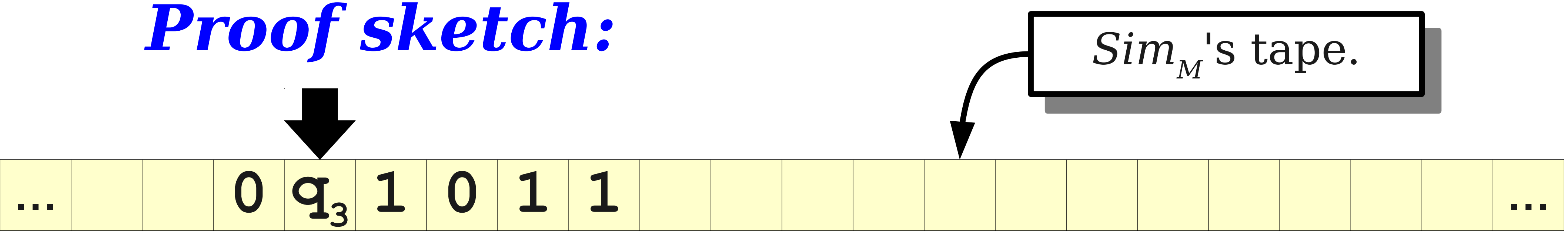


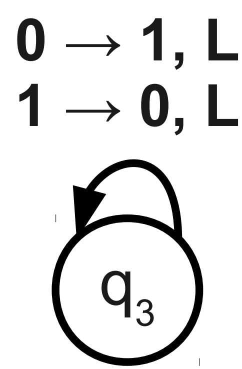
Now that we've done this once, let's do it again!

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:



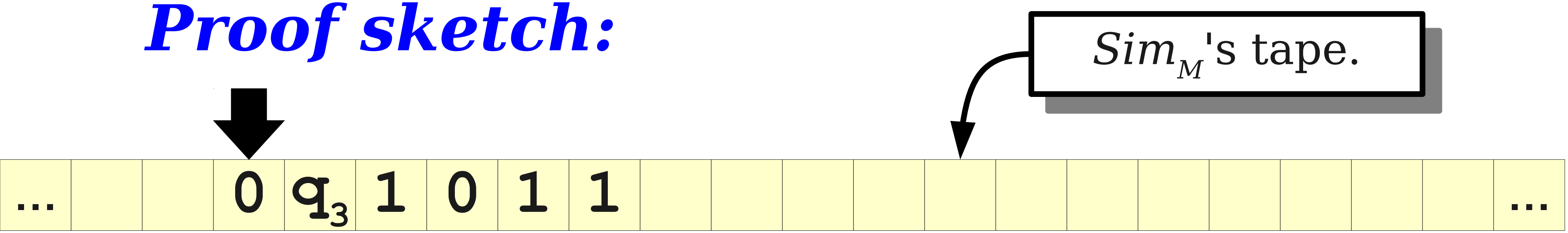


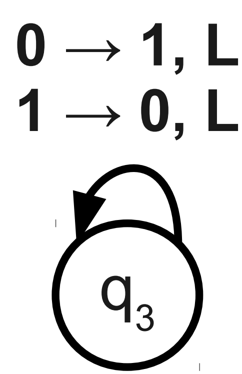
Now that we've done this once, let's do it again!

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:



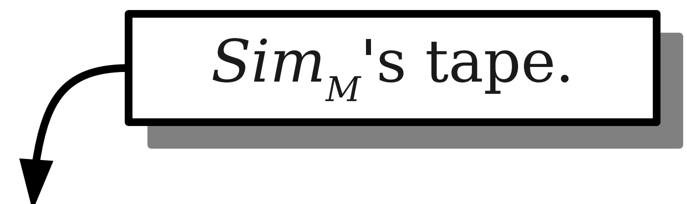


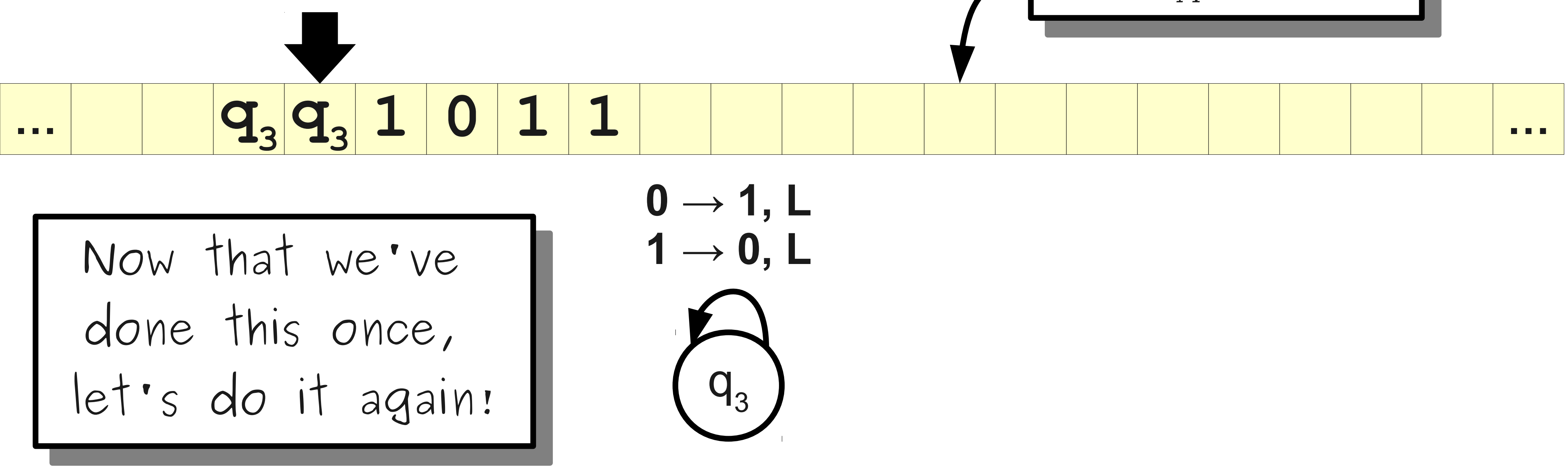
Now that we've done this once, let's do it again!

Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:

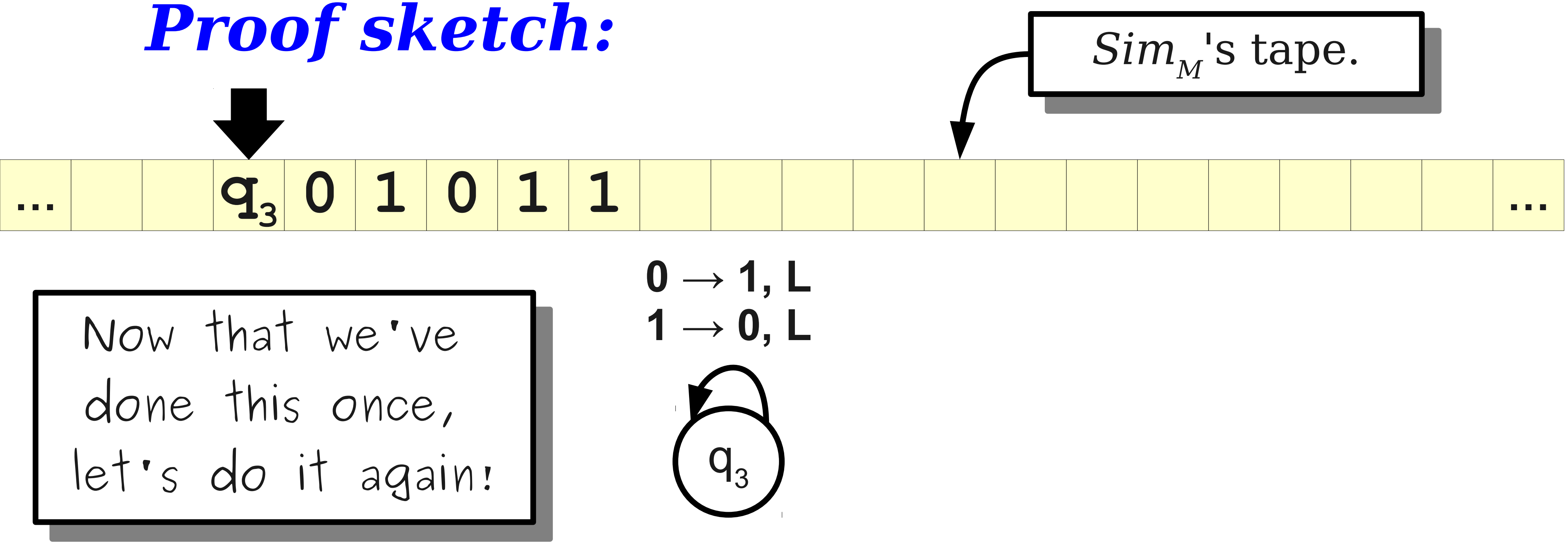




Manipulating IDs

Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

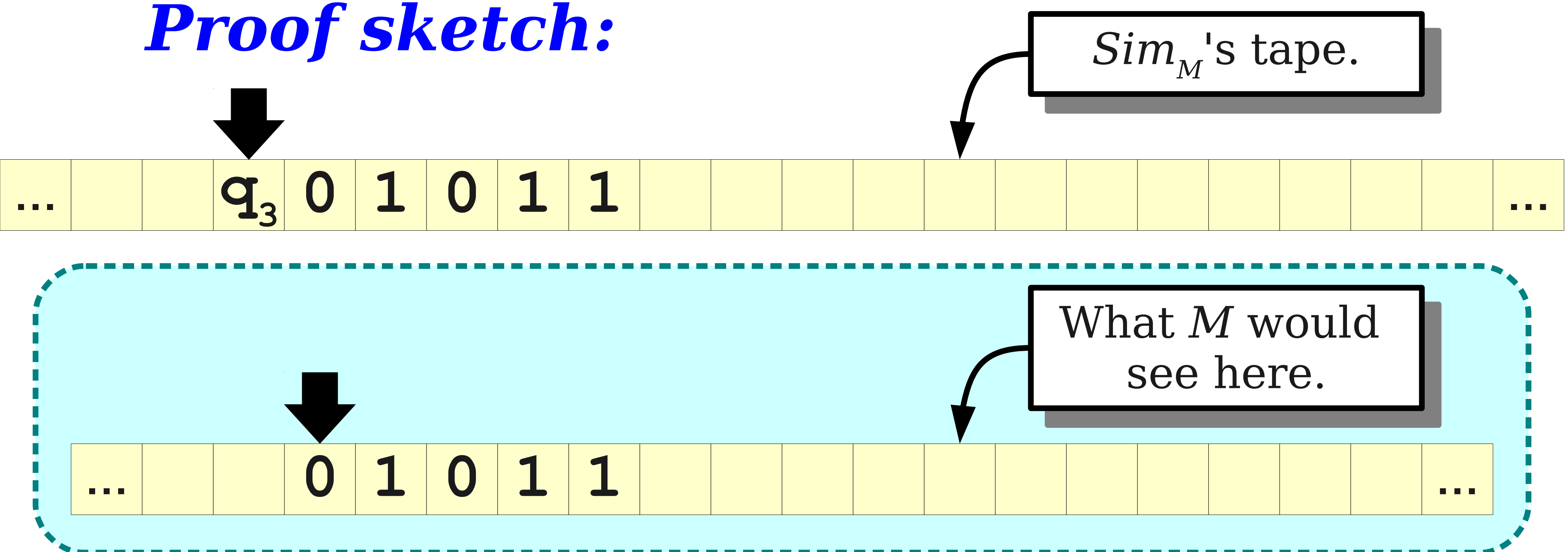
Proof sketch:



Manipulating IDs

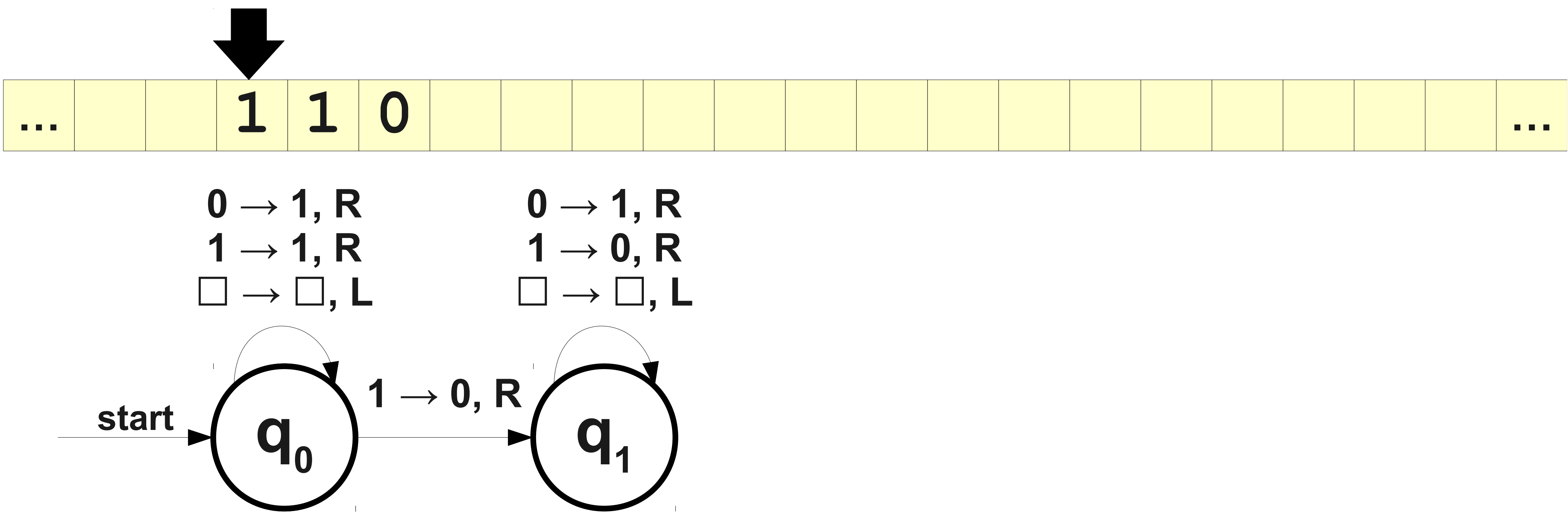
Theorem: For any TM M, it is possible to build a second TM Sim_M that, given a a tape containing an ID of M, simulates one step of M's computation.

Proof sketch:



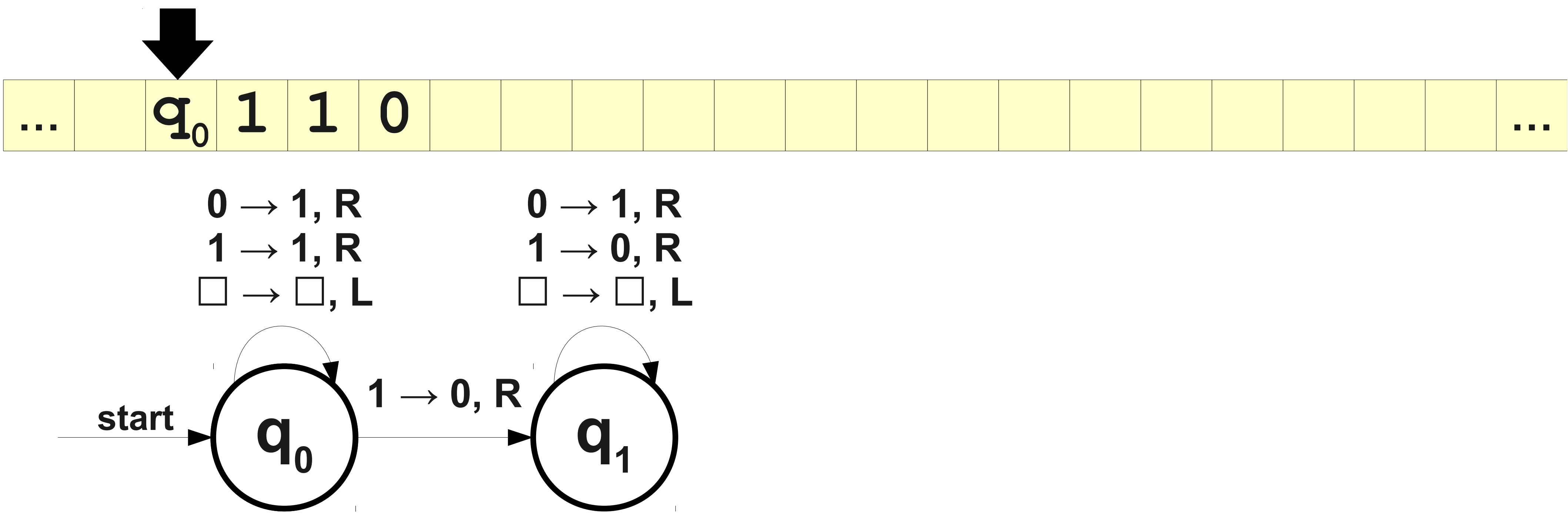
Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



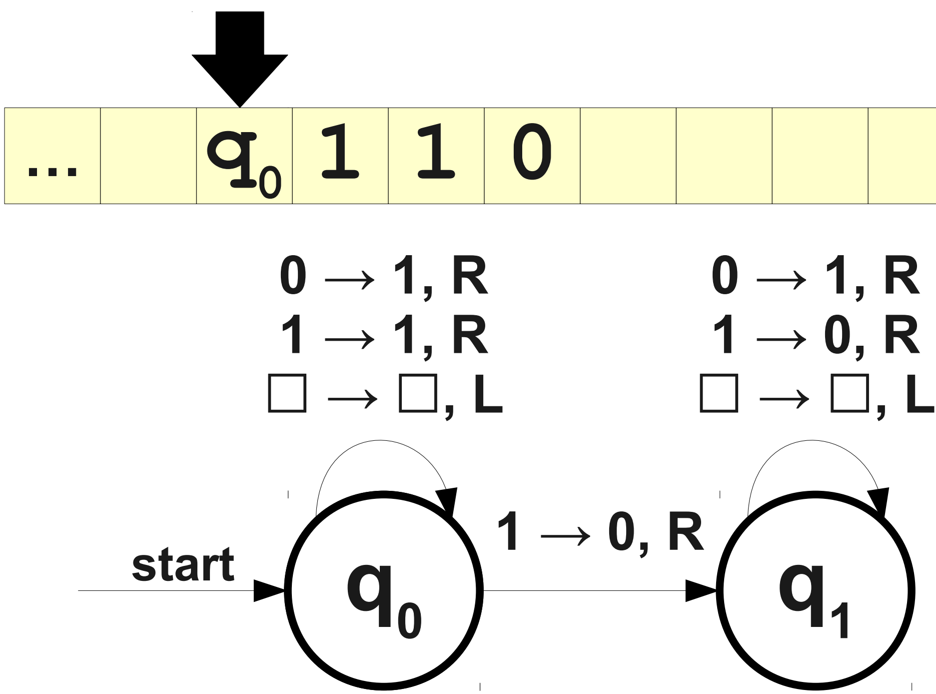
Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

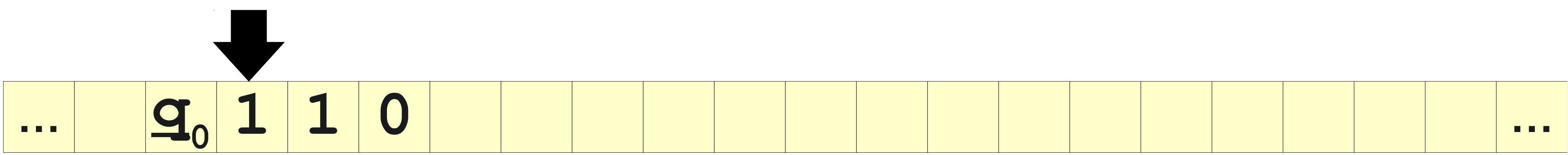
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

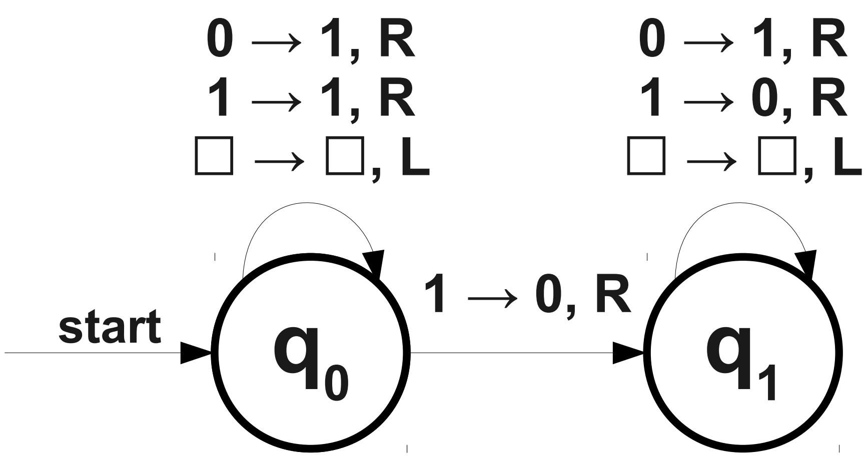


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

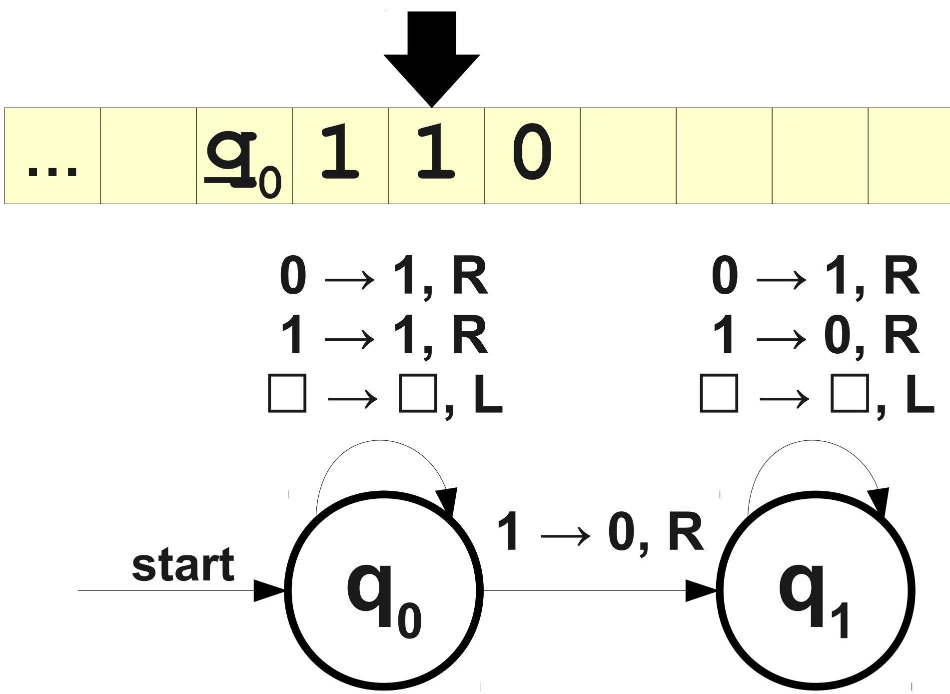




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

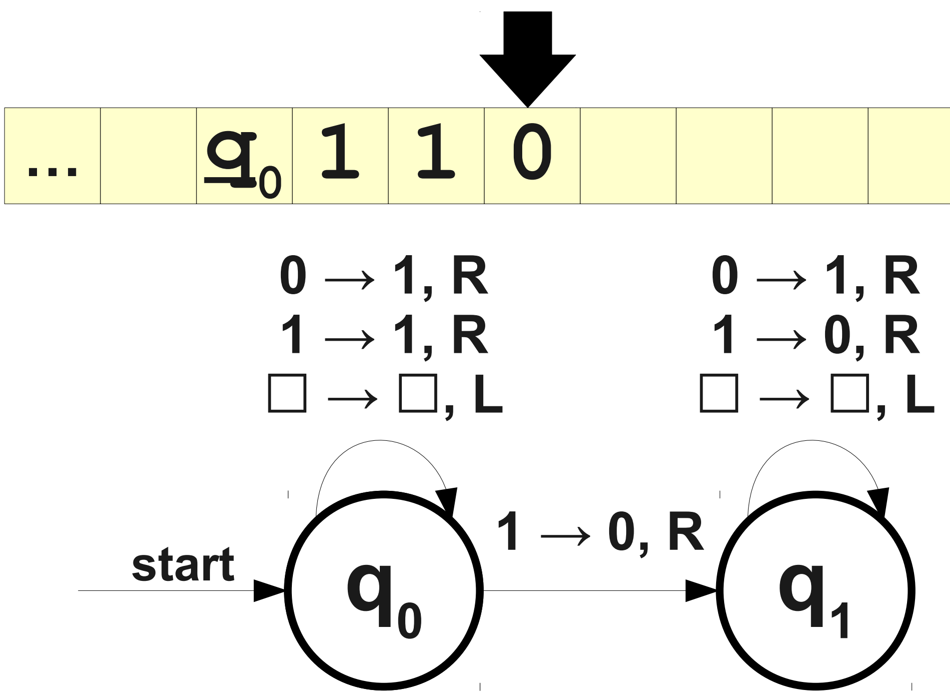
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

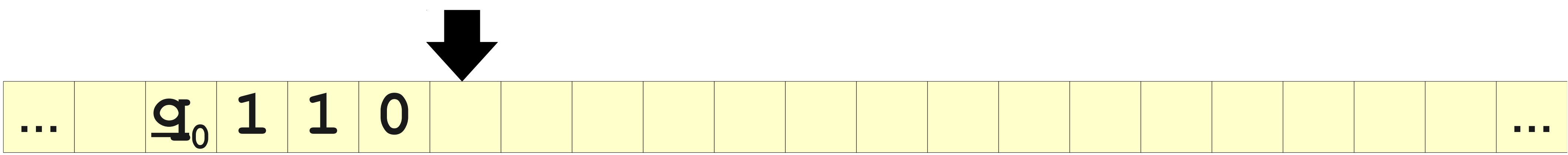
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

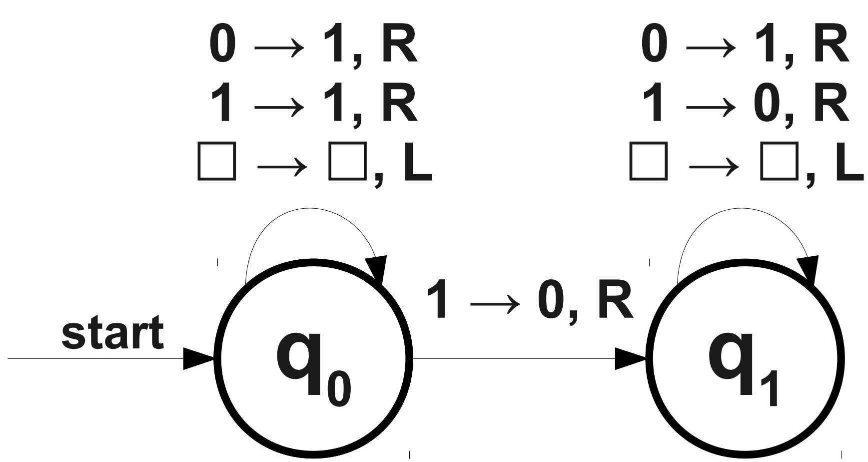


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

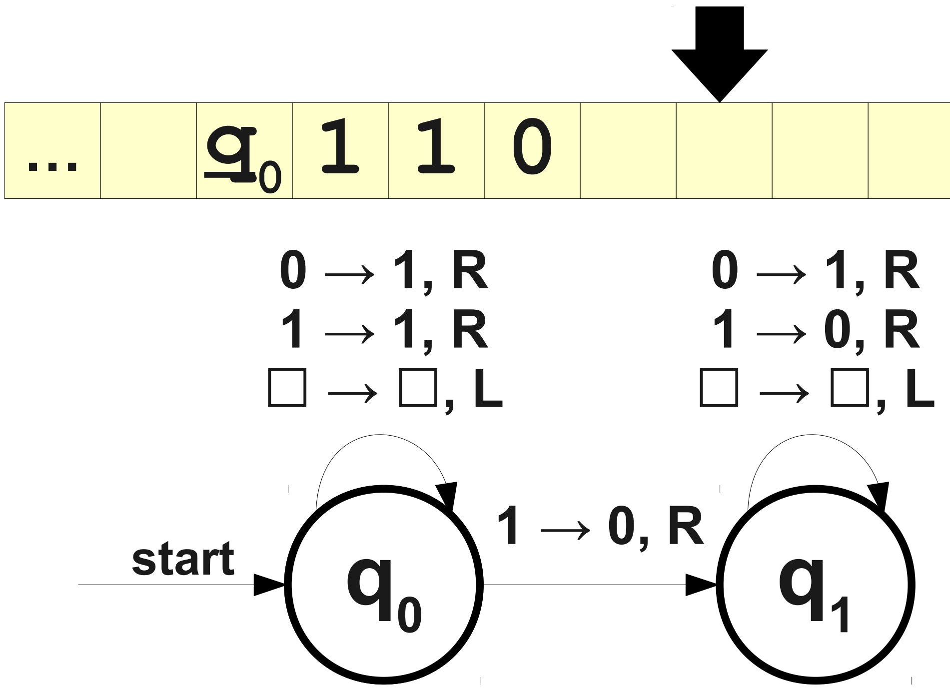




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

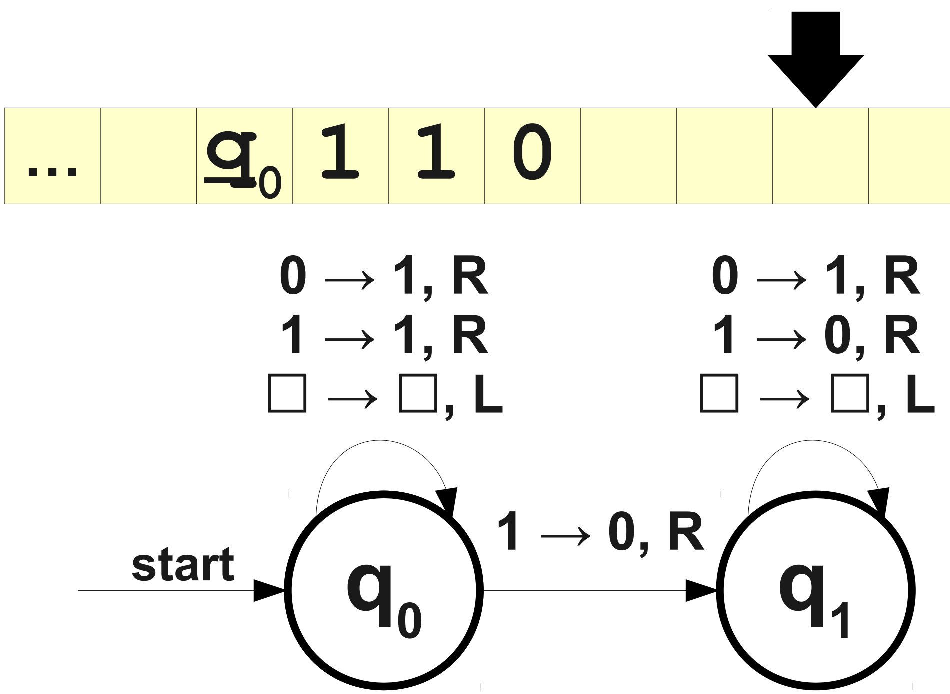
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

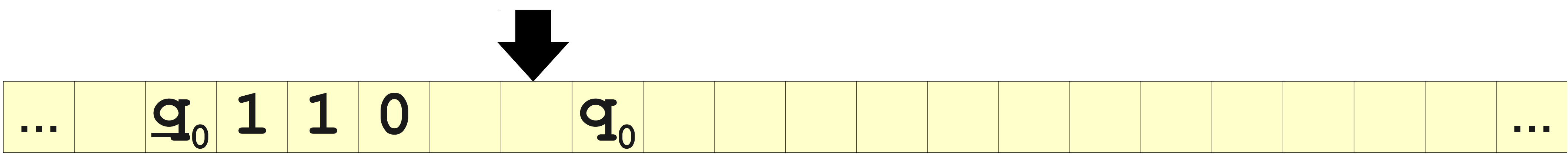
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

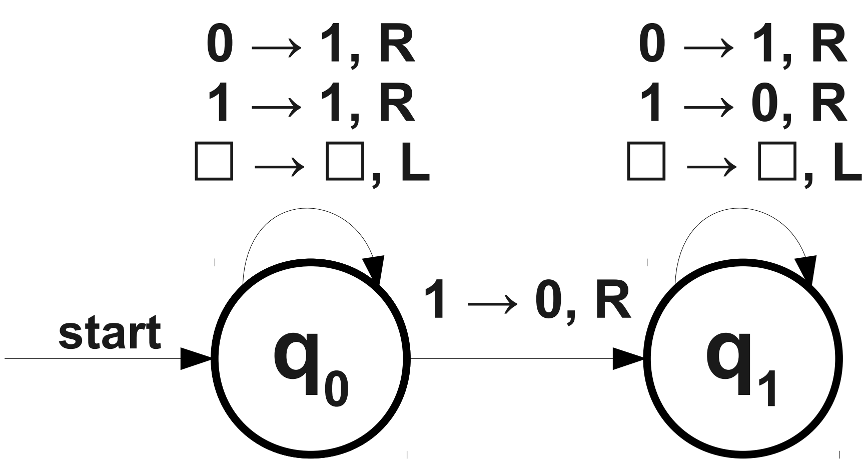


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

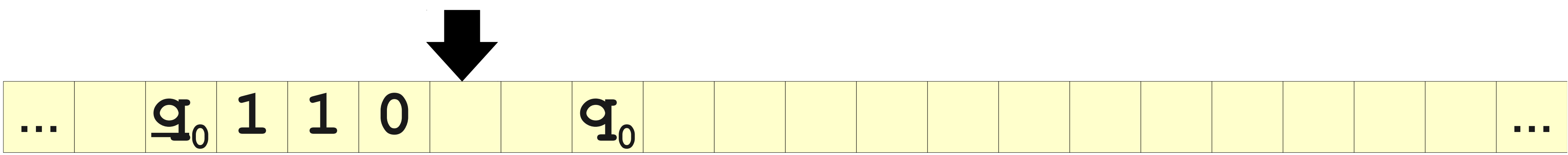


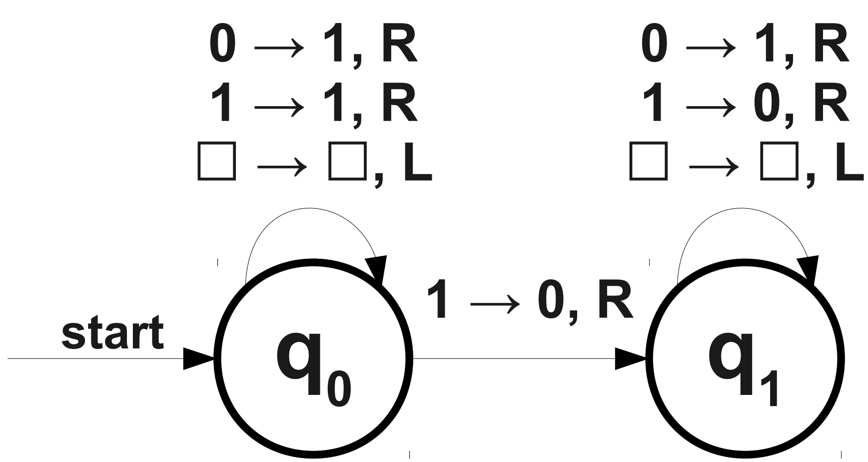


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

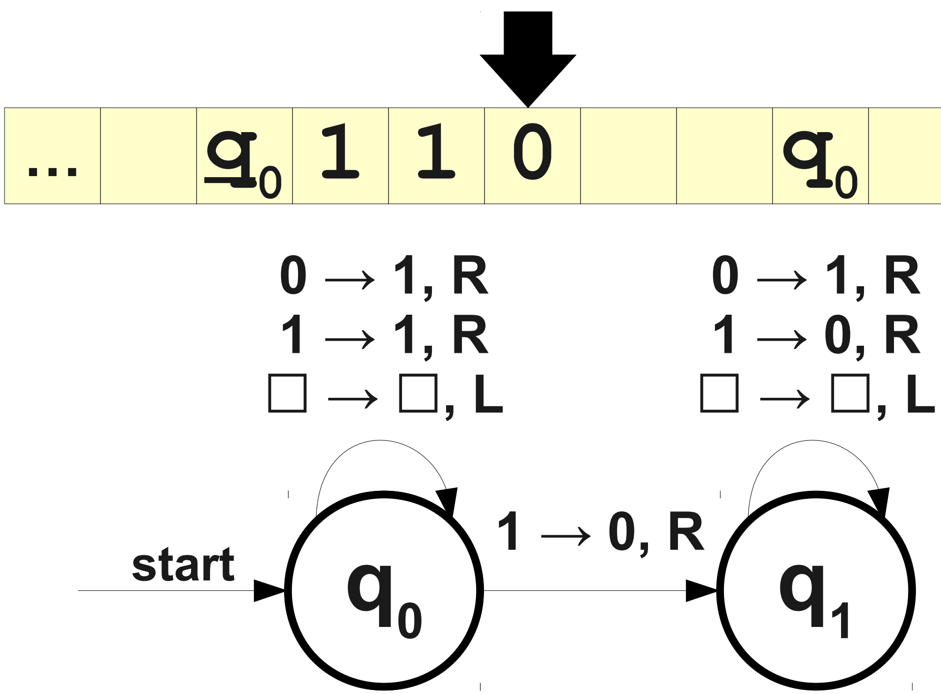




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

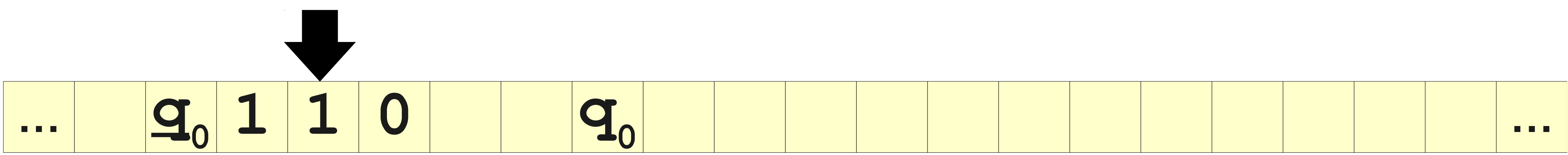
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

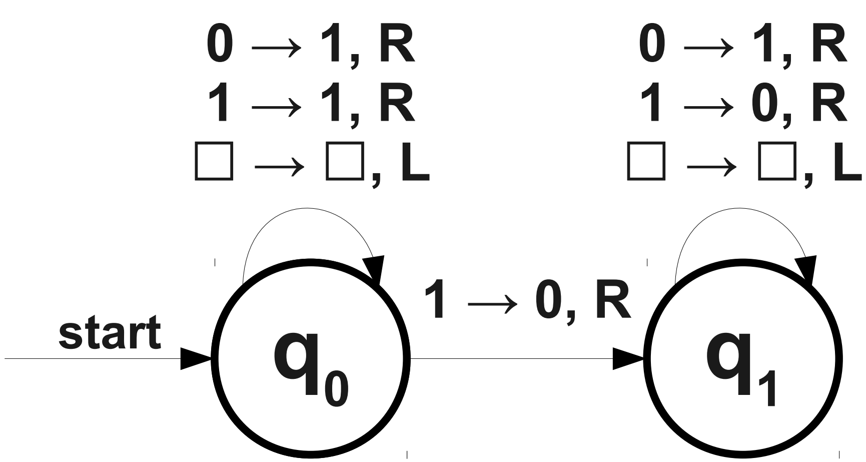


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

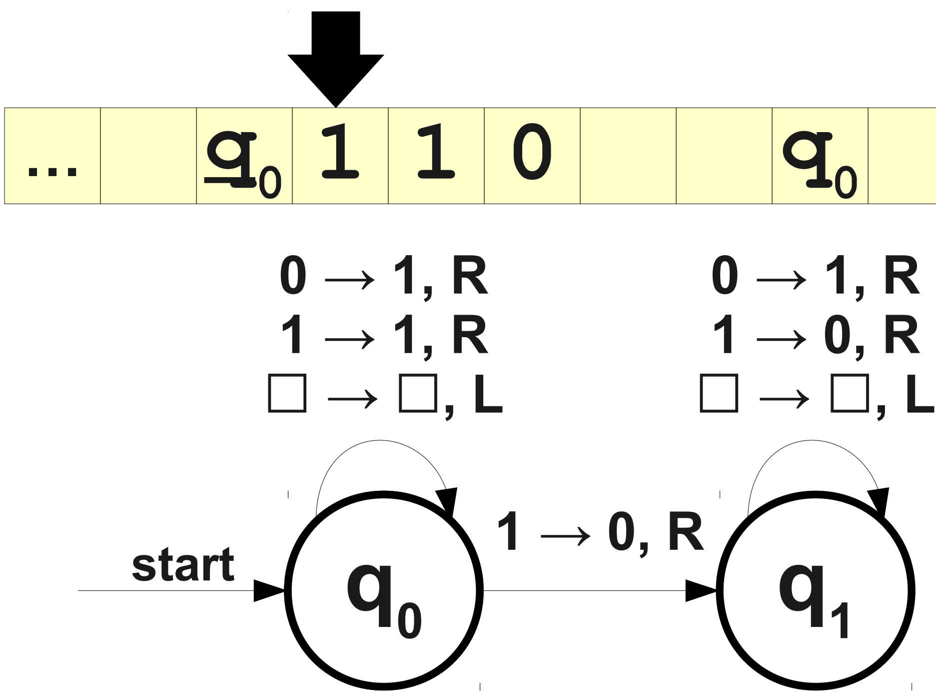




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

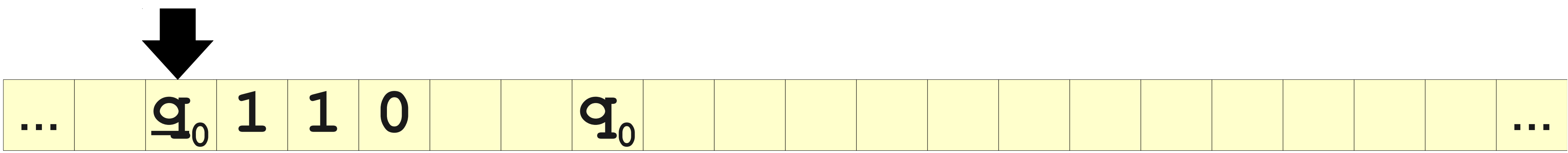
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

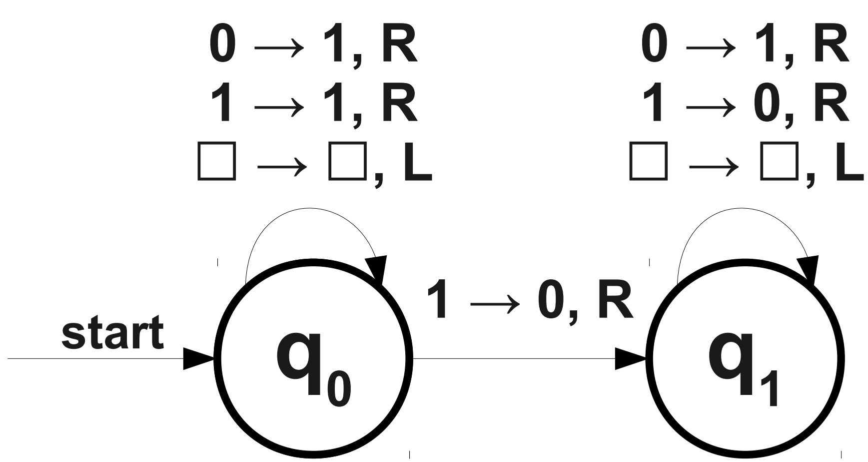


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

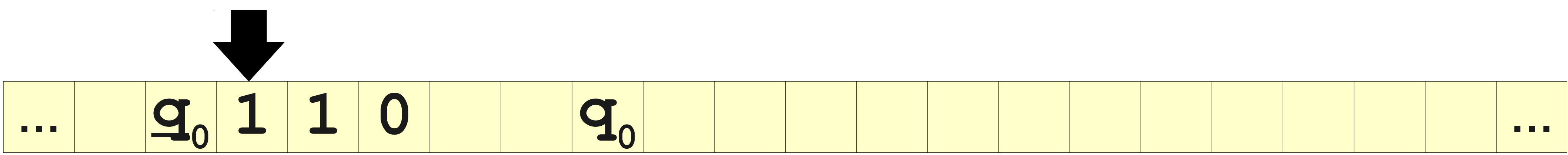


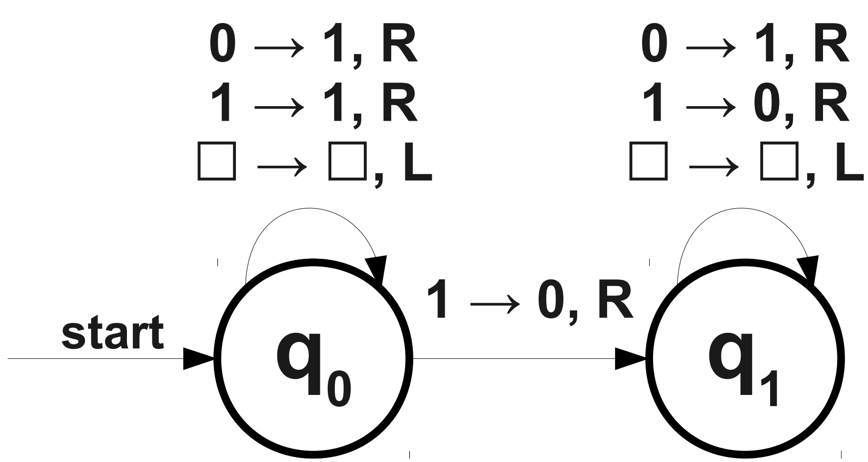


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

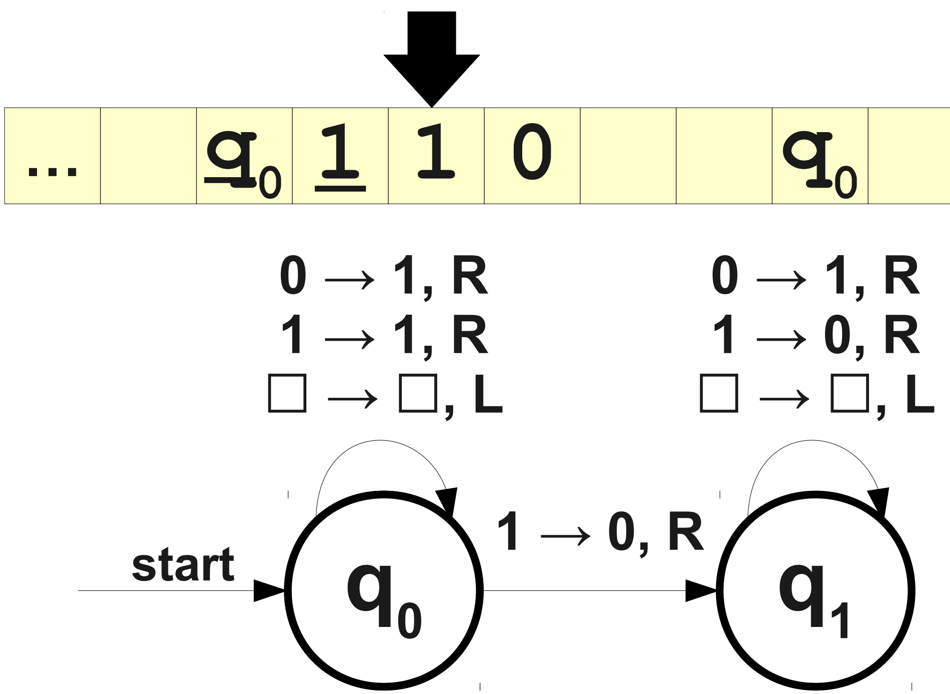




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

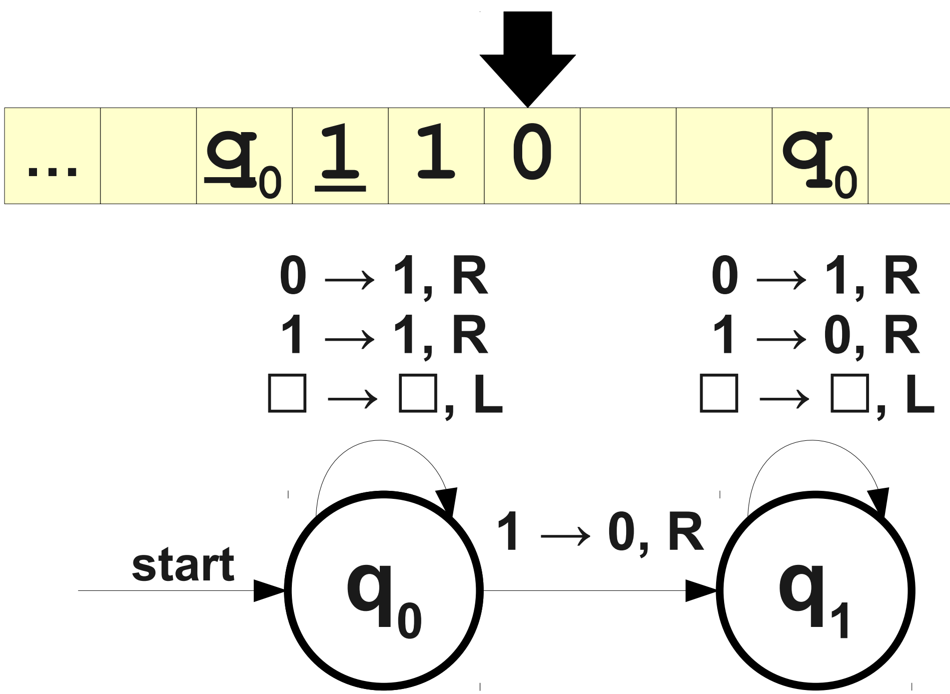
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

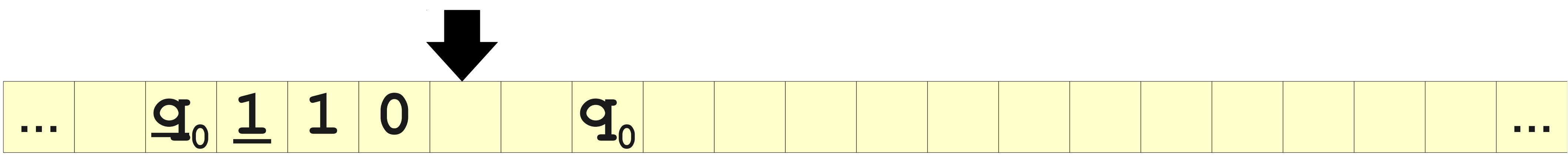
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

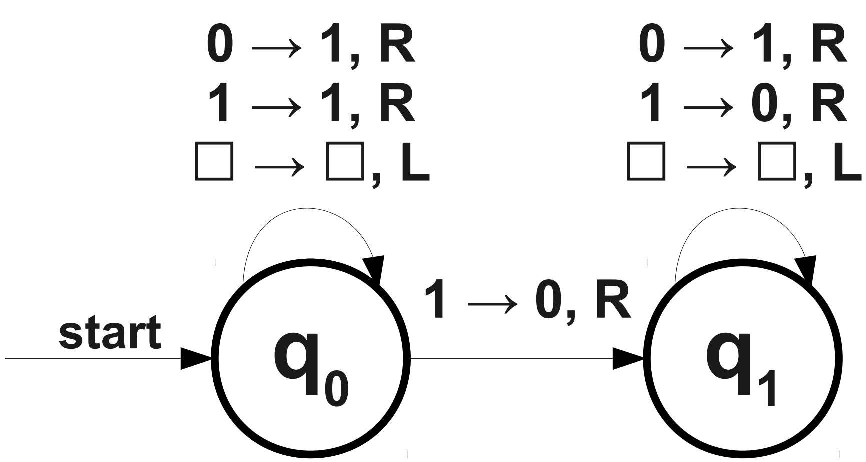


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

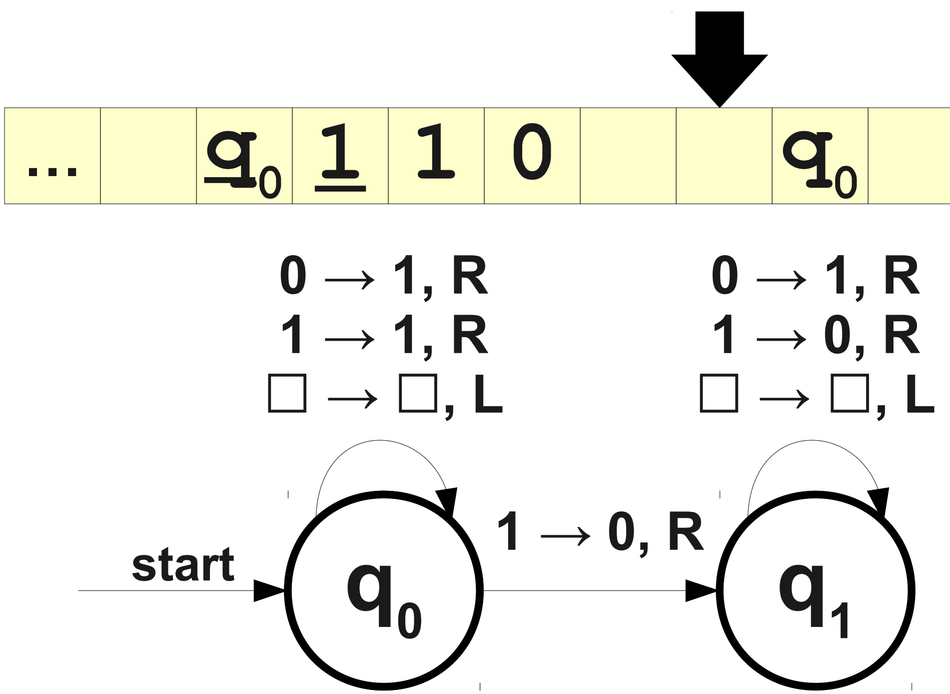




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

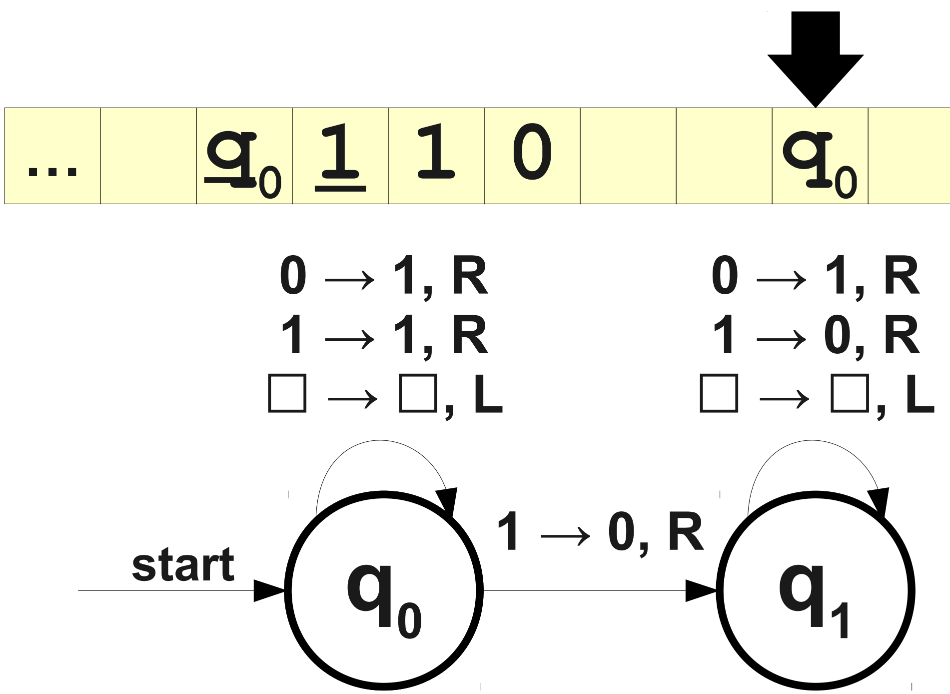
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

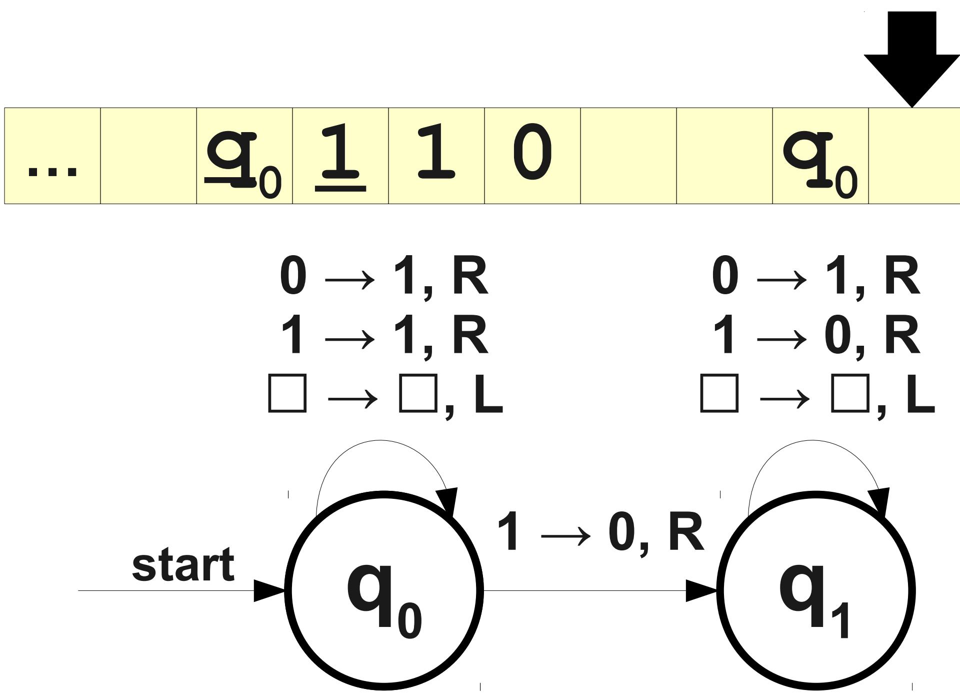
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

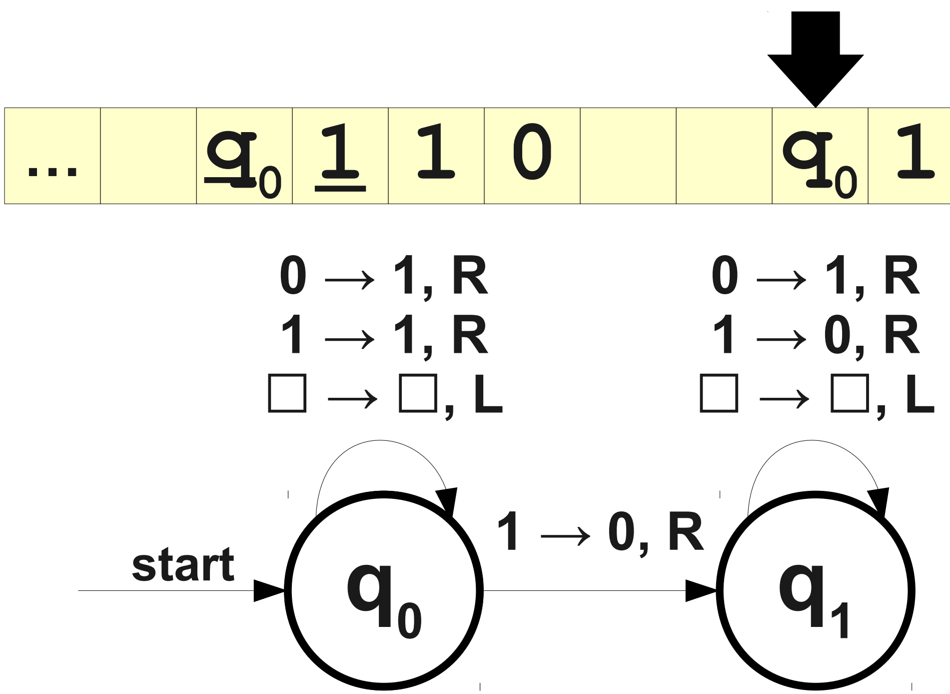
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

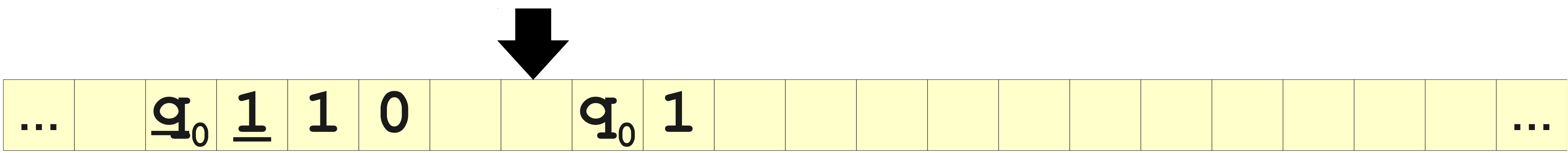
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

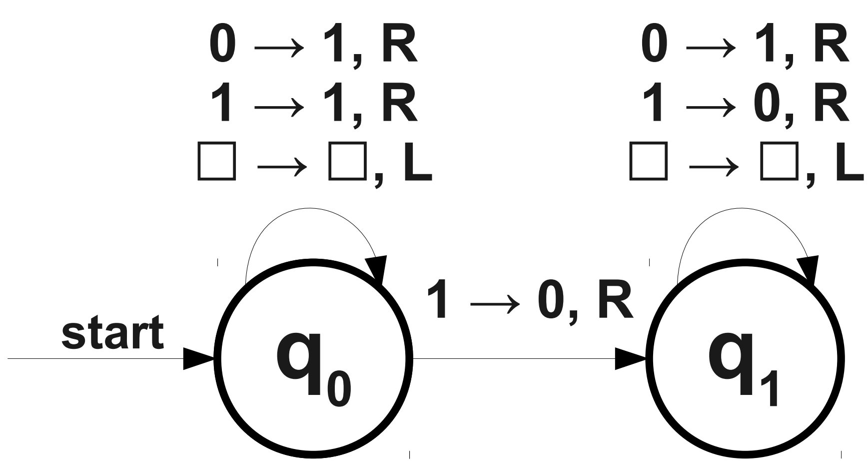


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

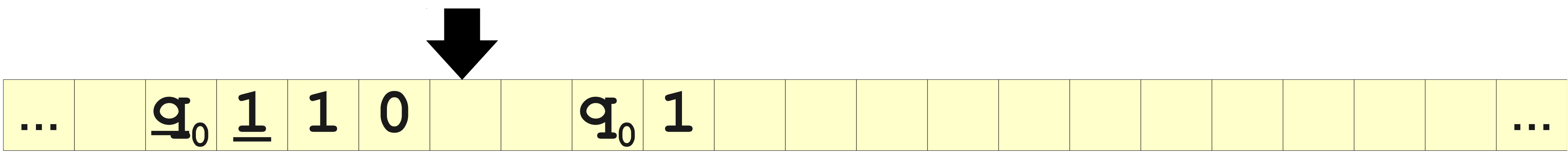


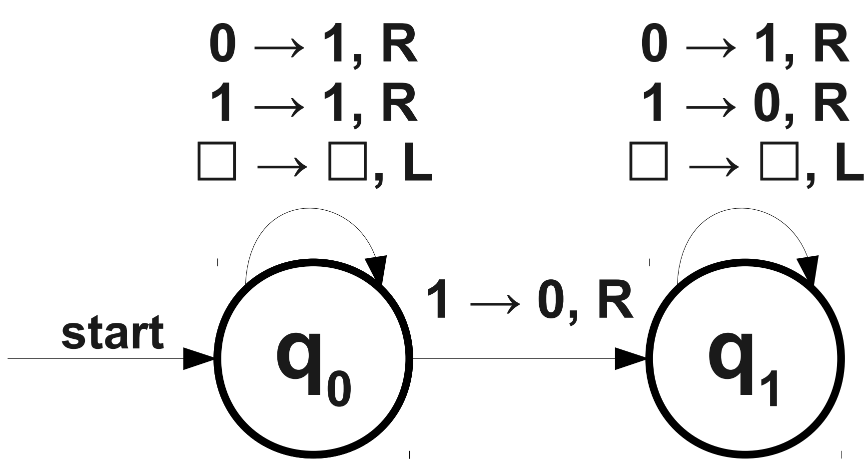


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

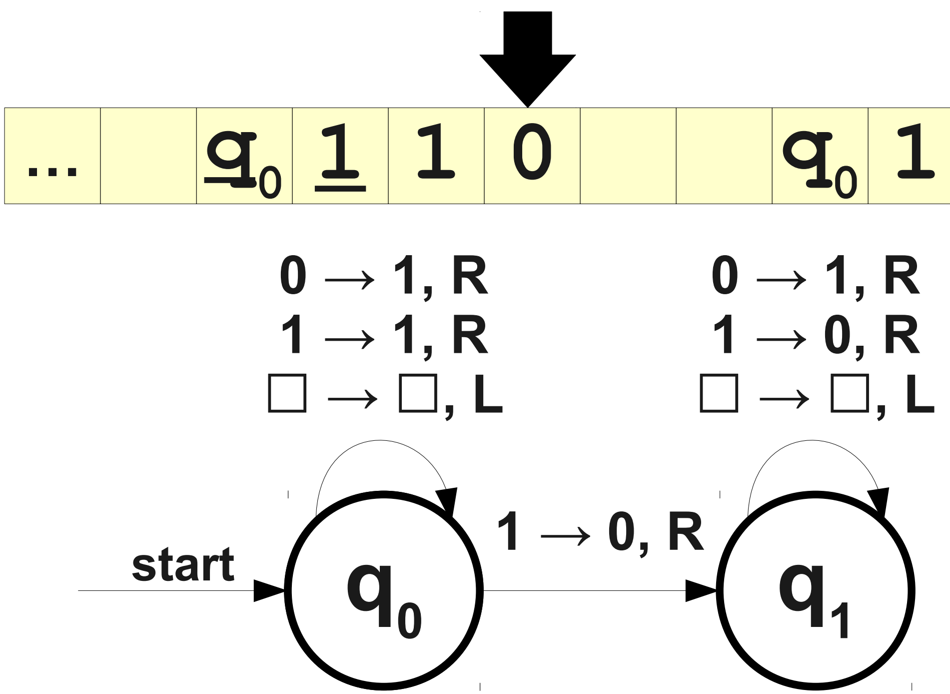




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

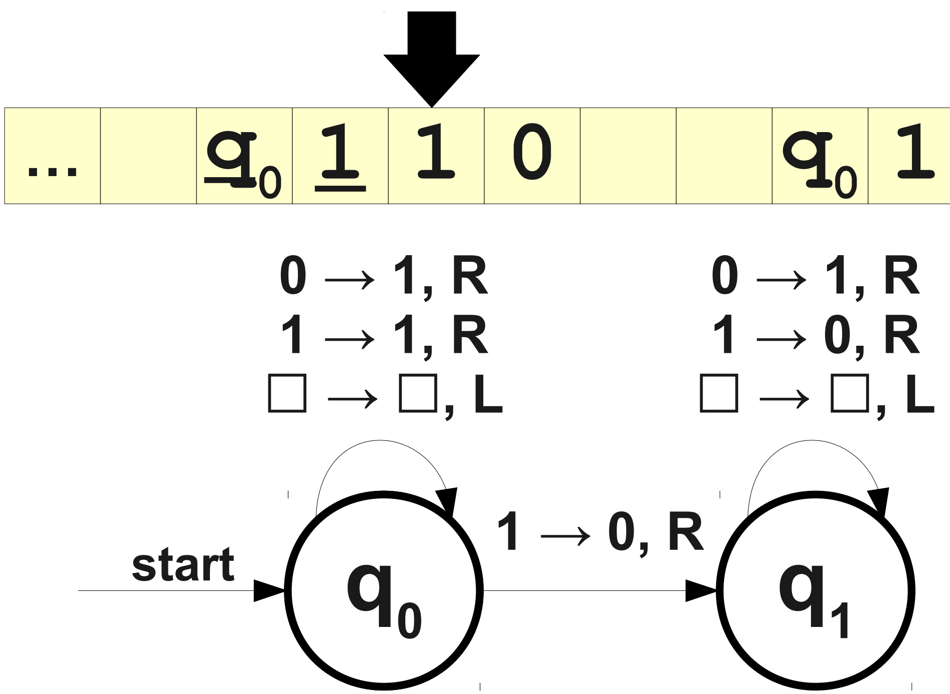
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

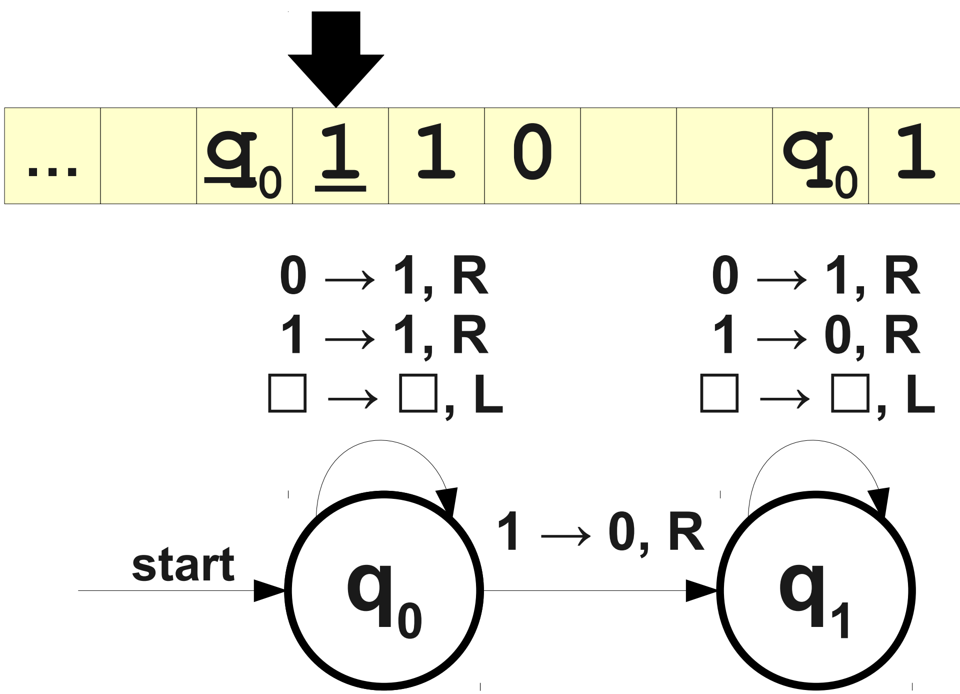
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

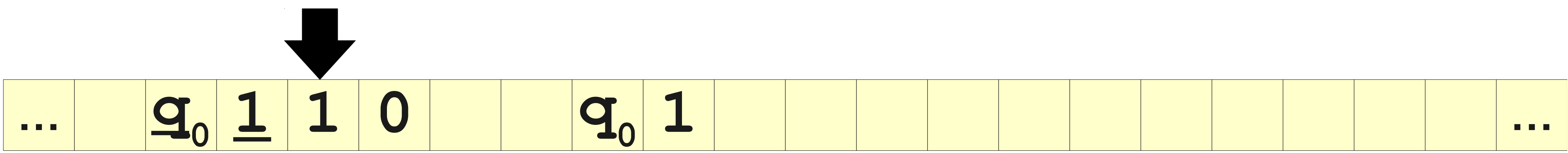
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

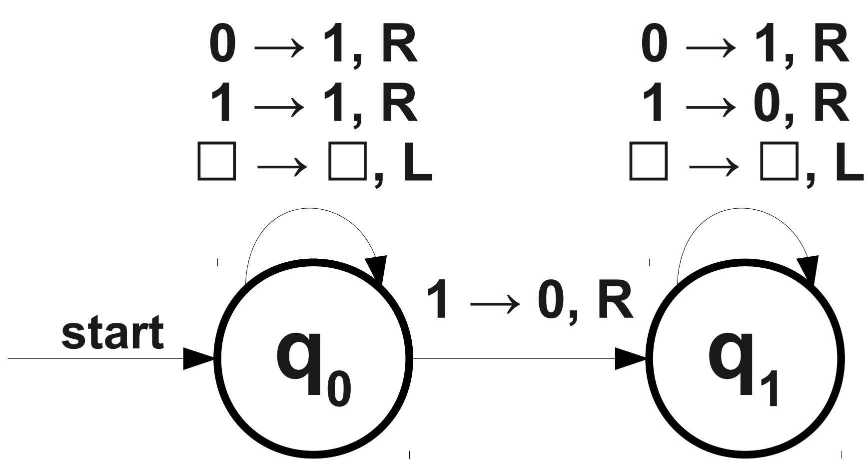


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

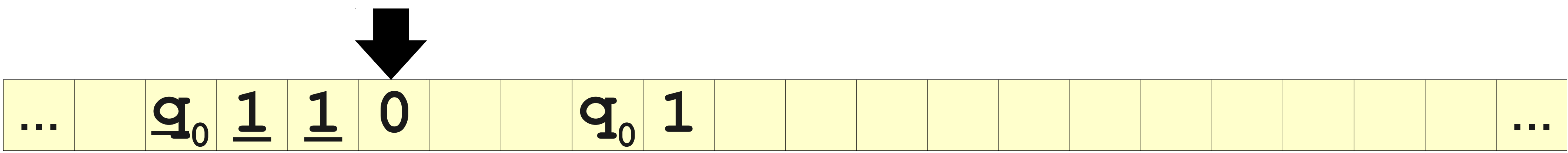


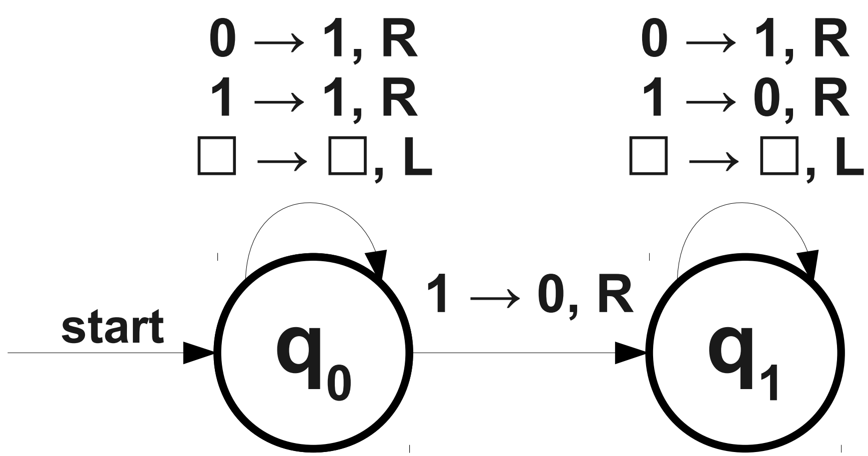


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

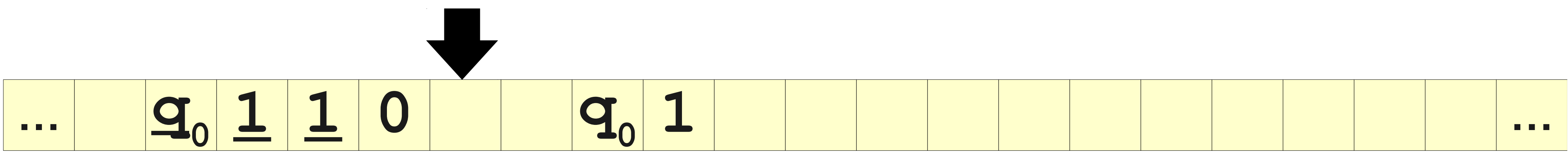


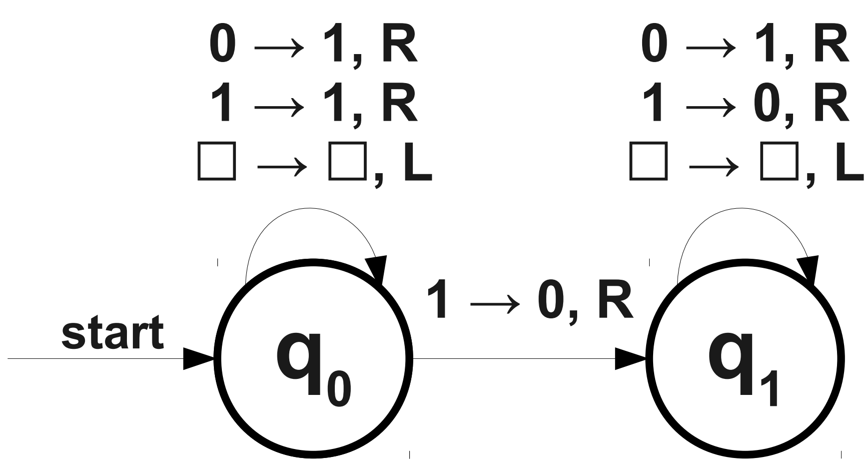


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

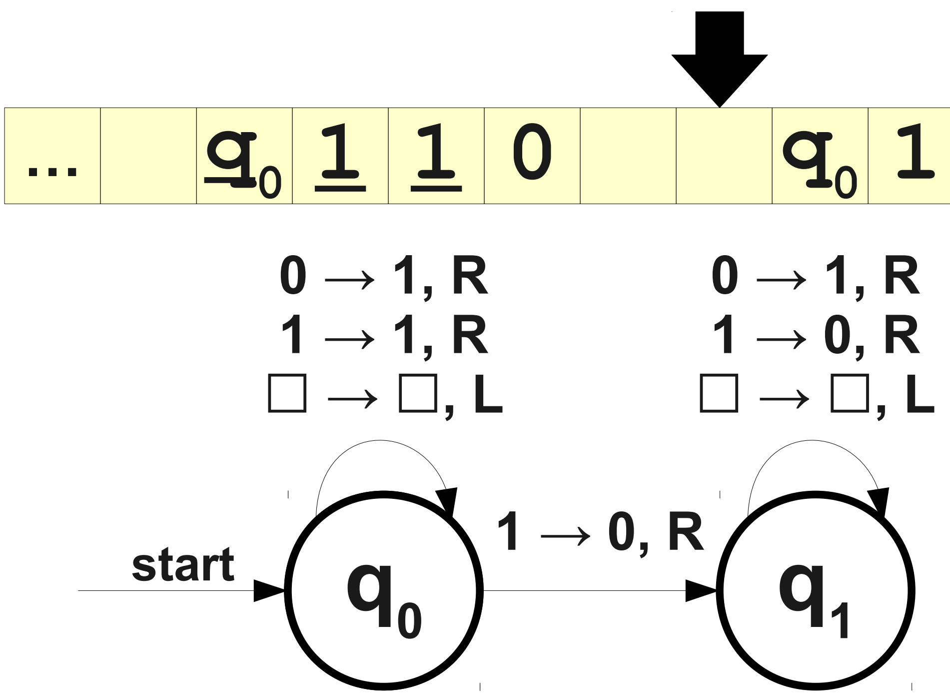




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

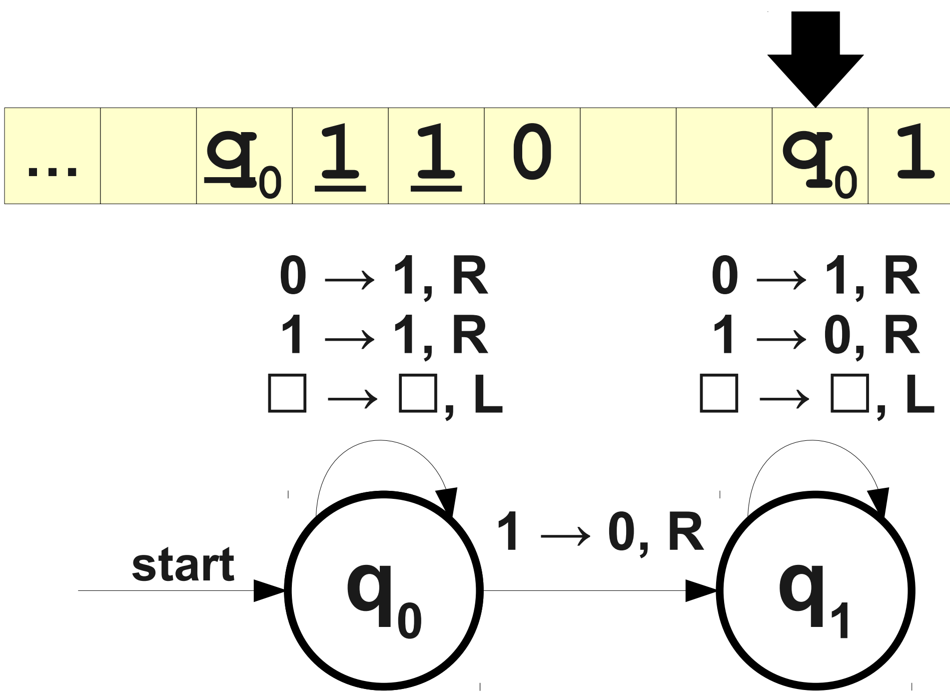
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

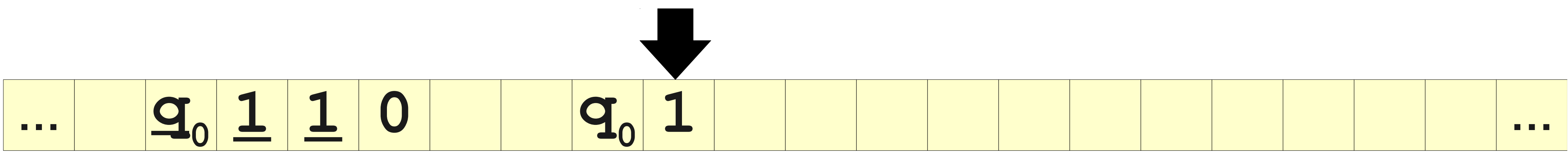
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

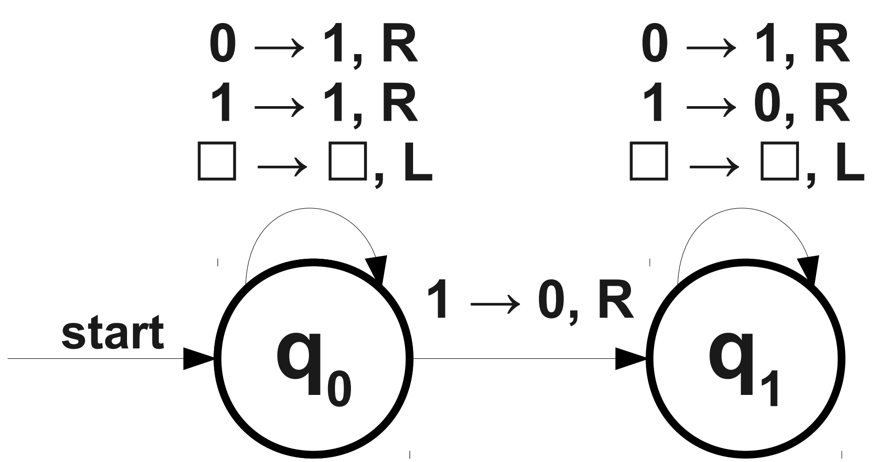


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

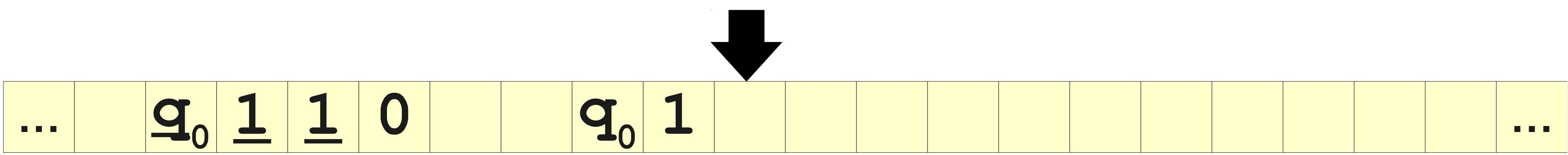


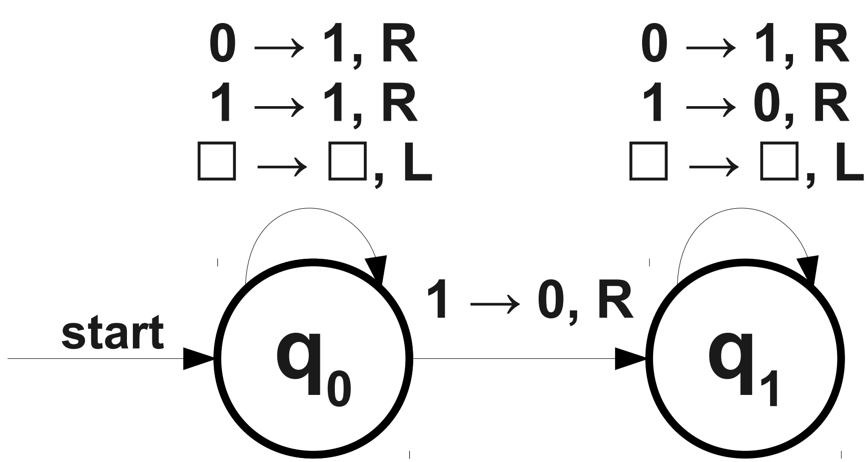


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

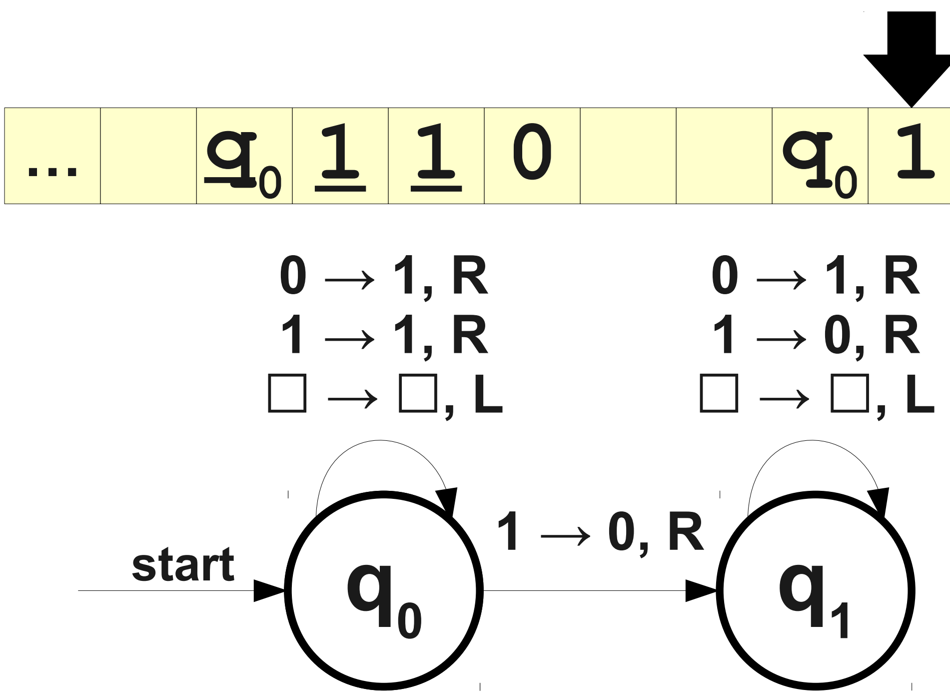




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

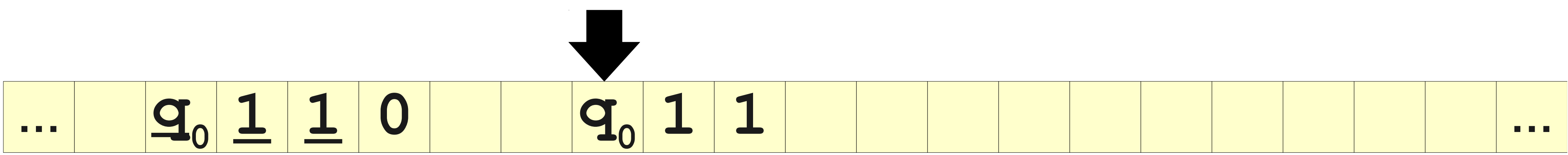
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

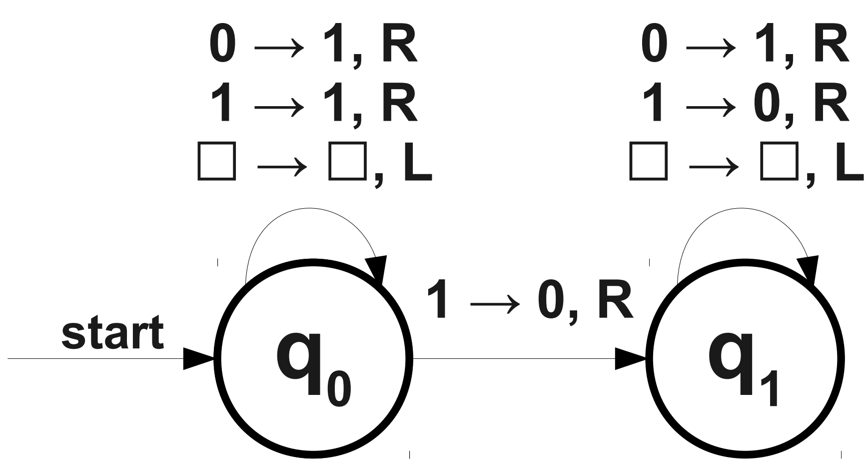


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

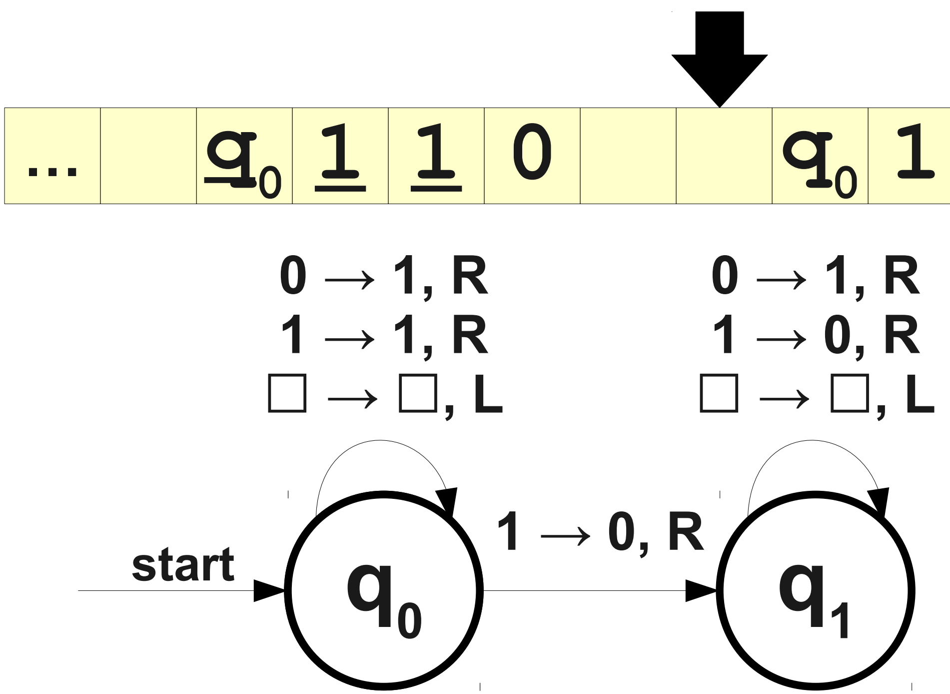




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

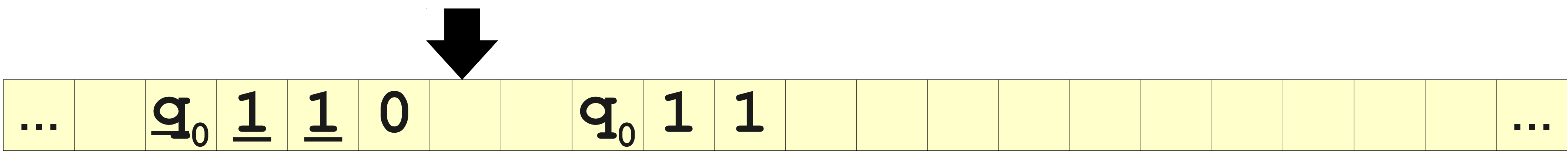
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

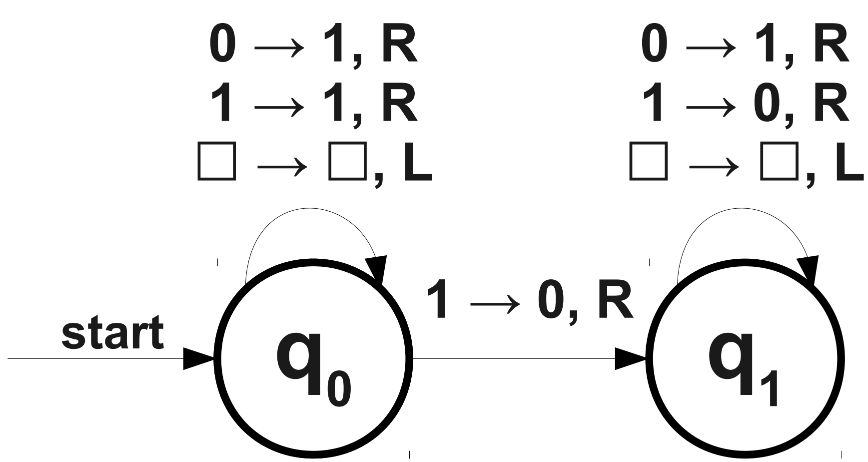


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

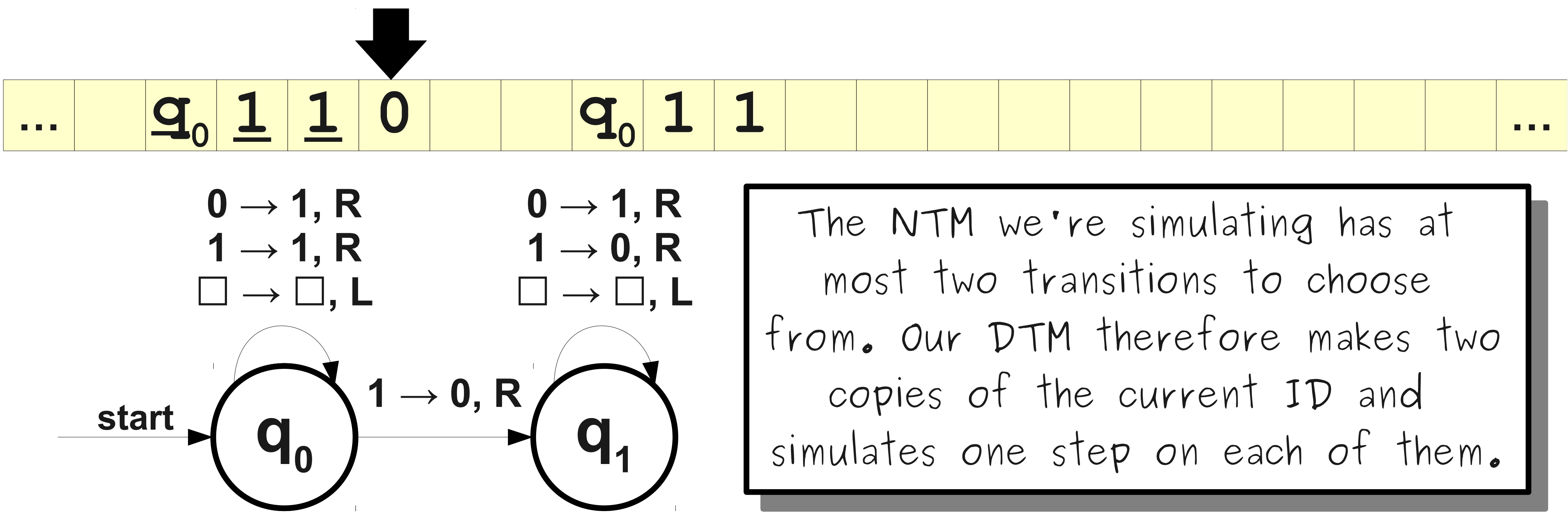




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

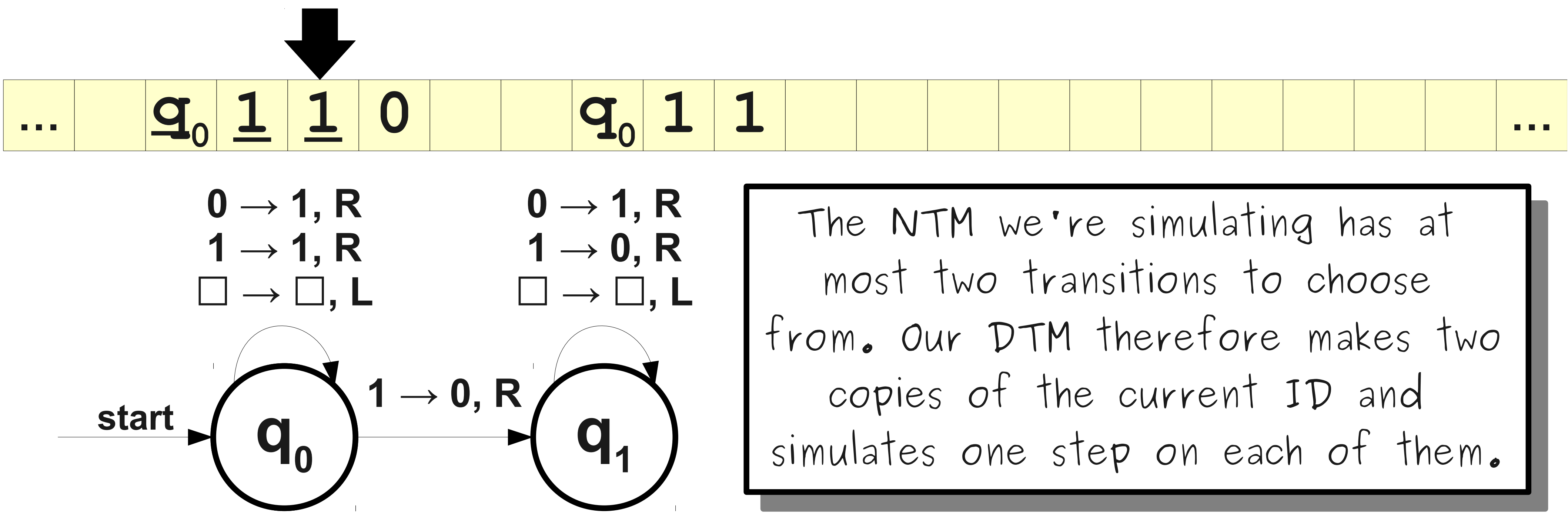
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

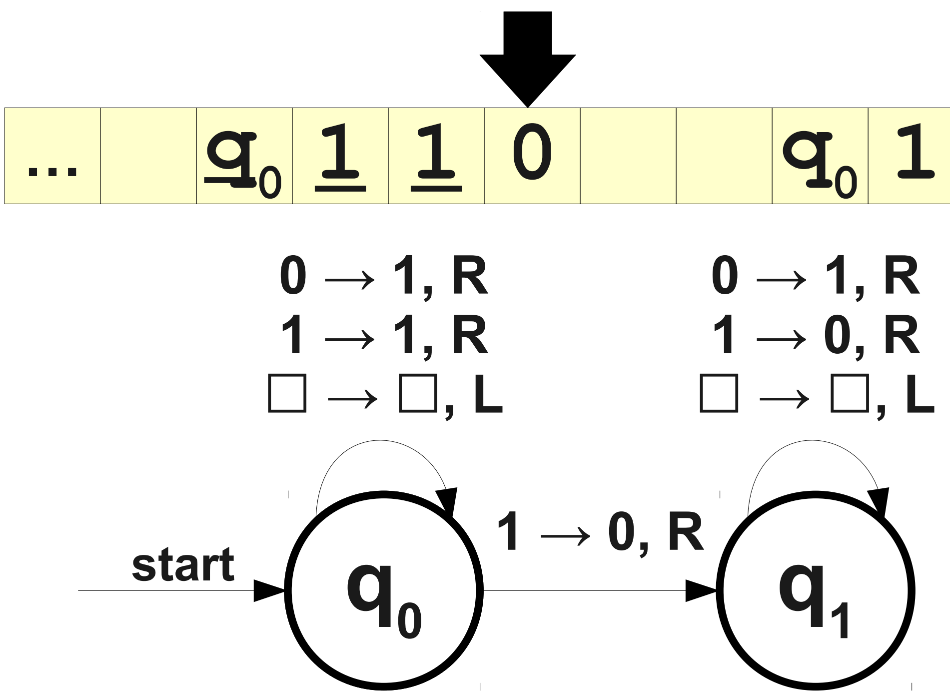
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

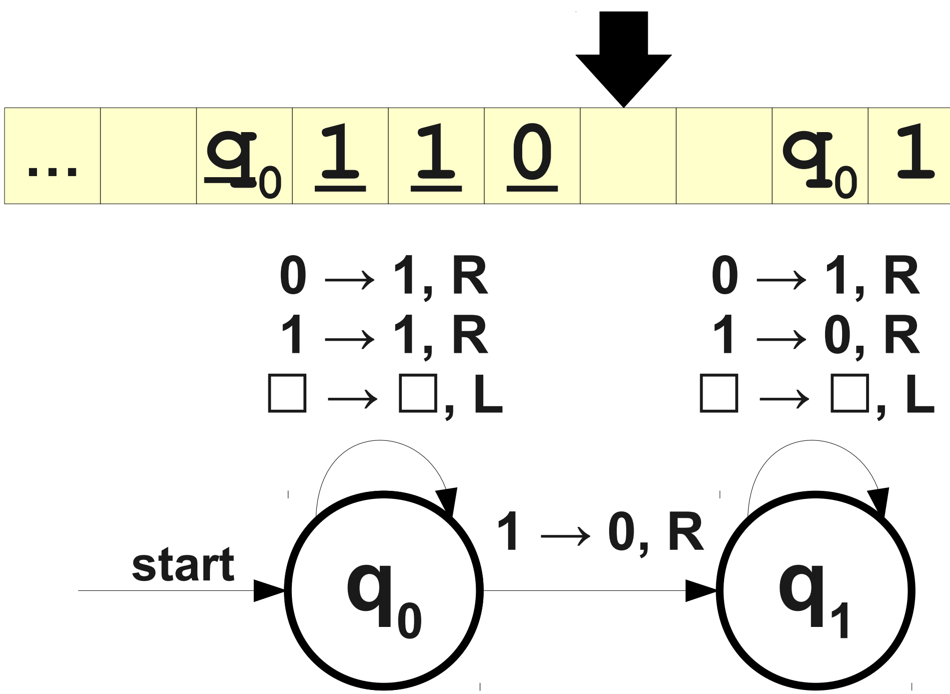
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

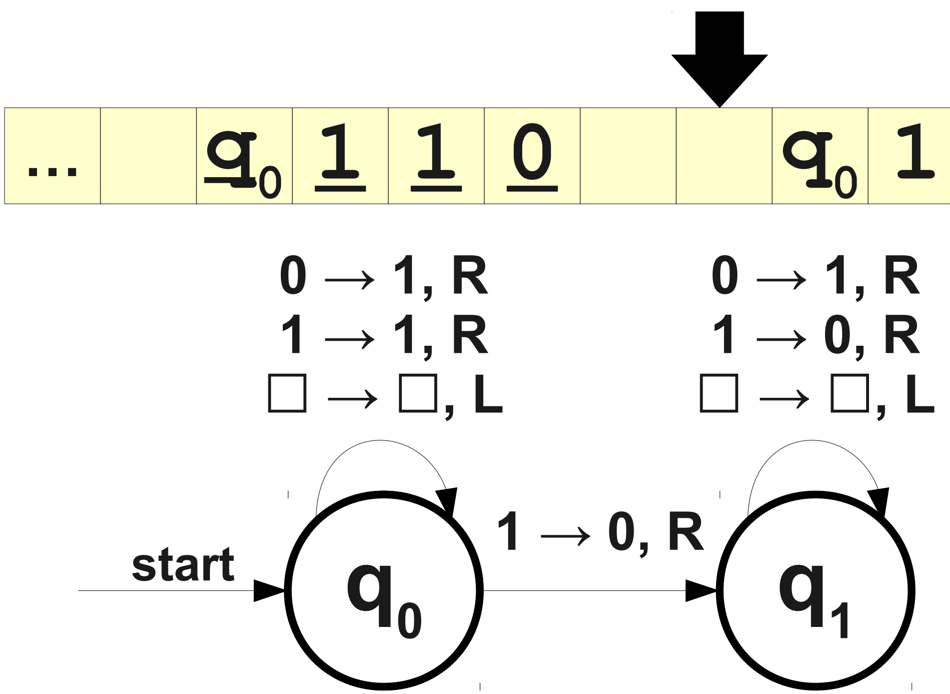
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

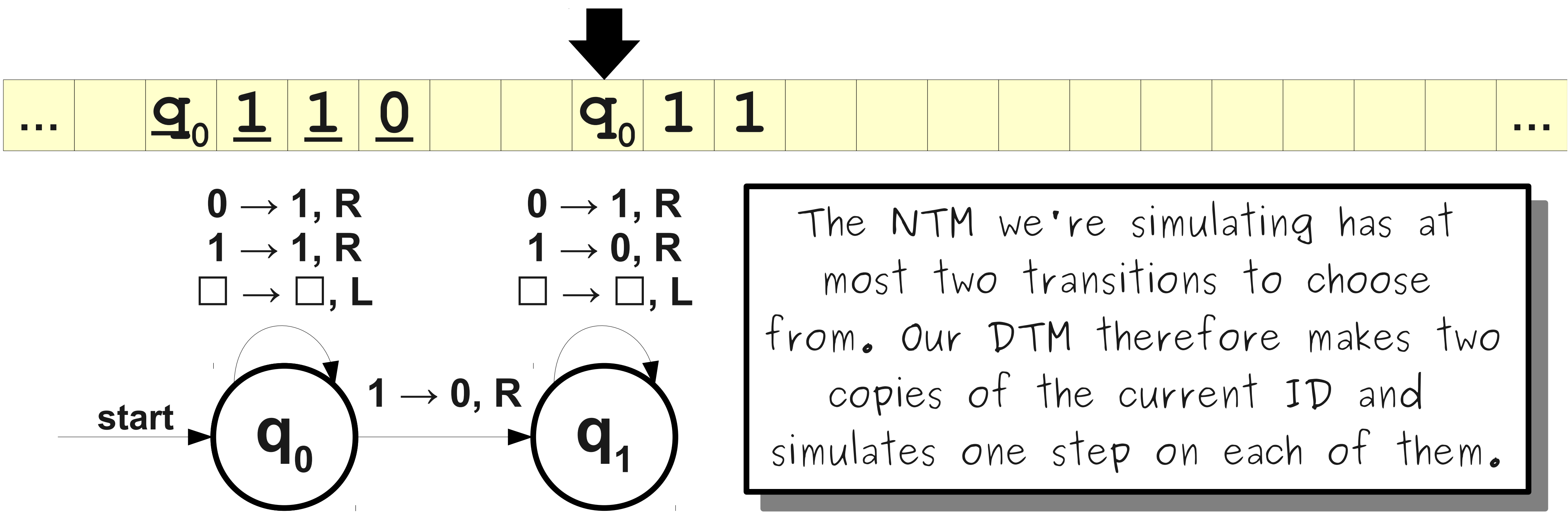
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

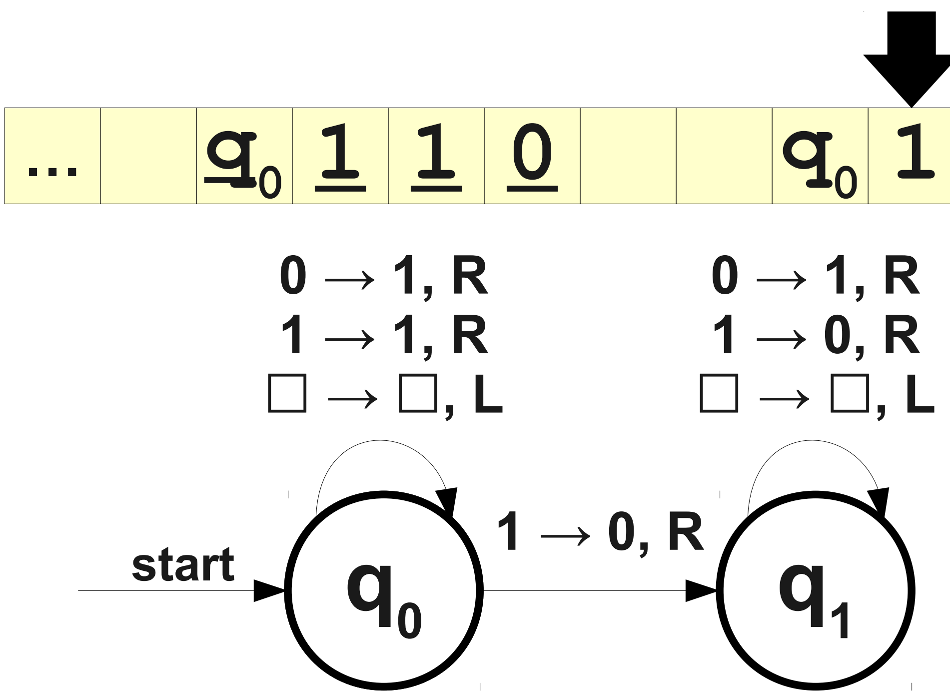
Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

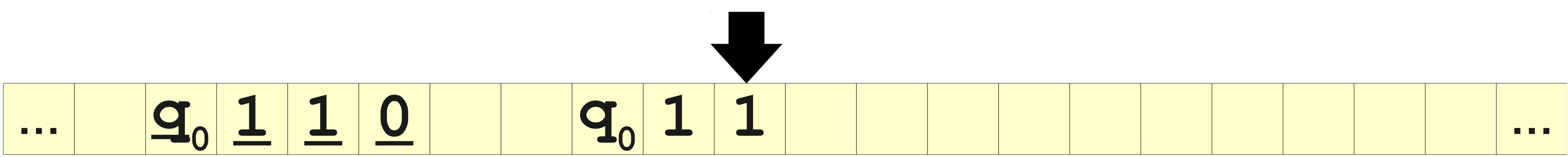
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

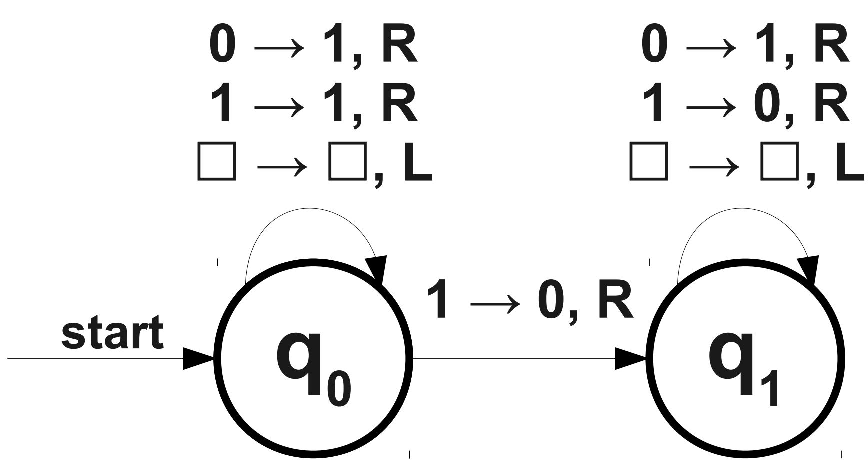


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

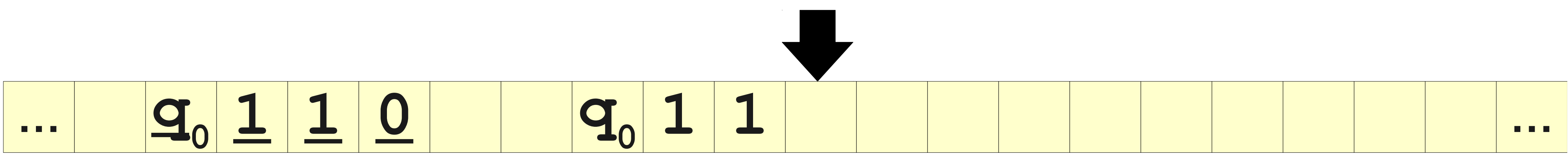


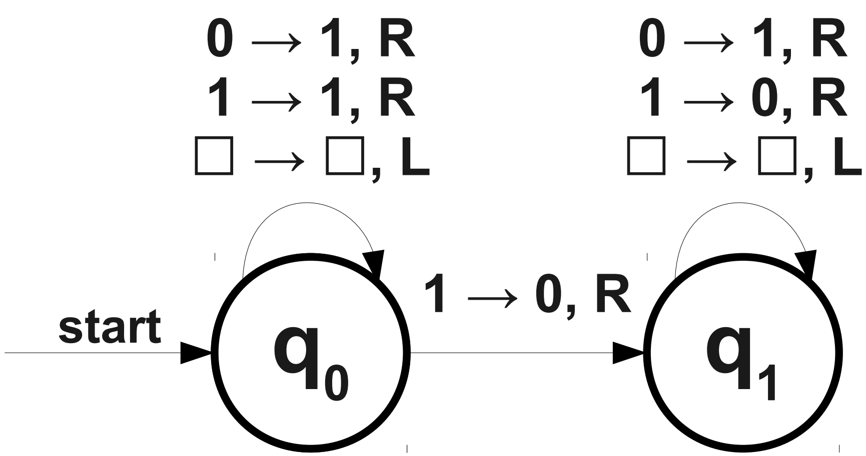


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

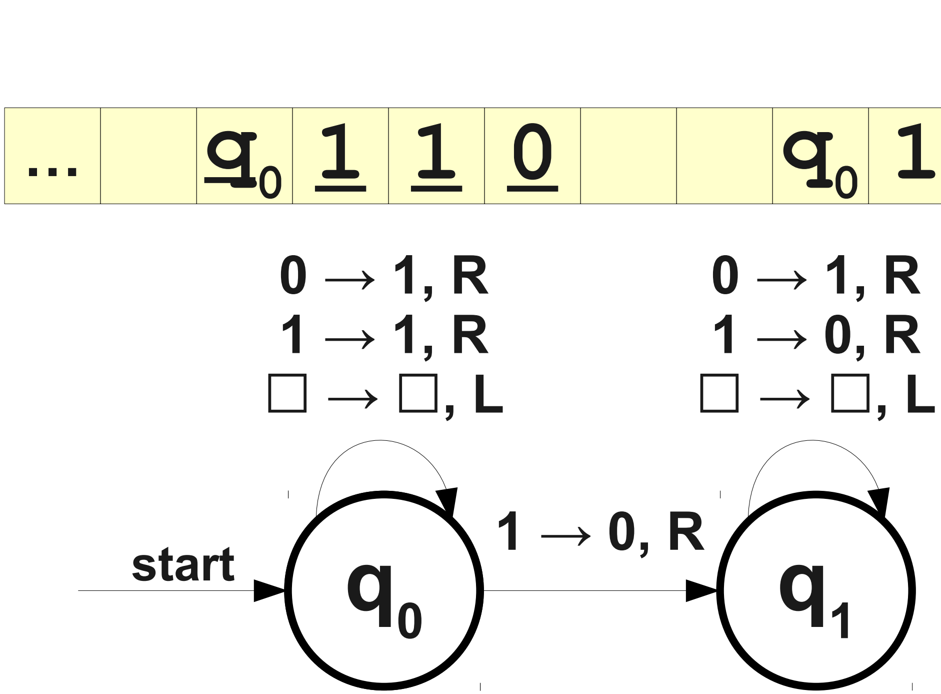




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

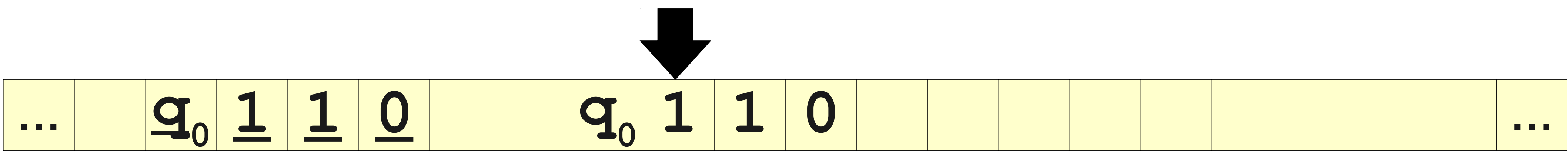
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

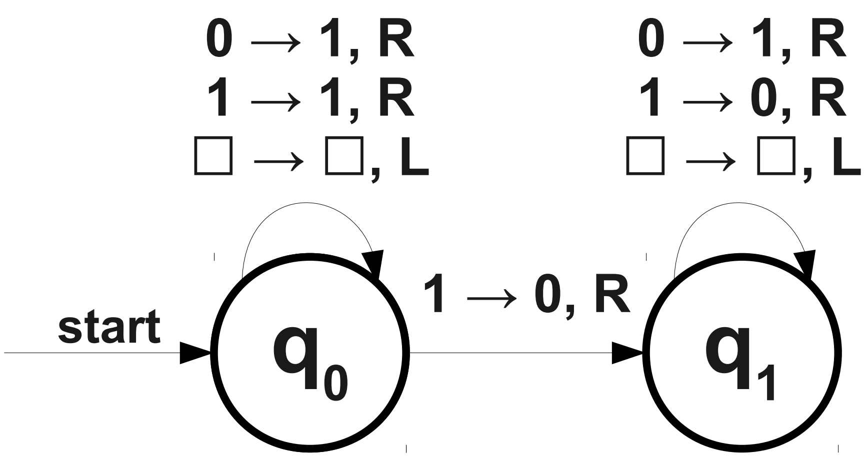


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

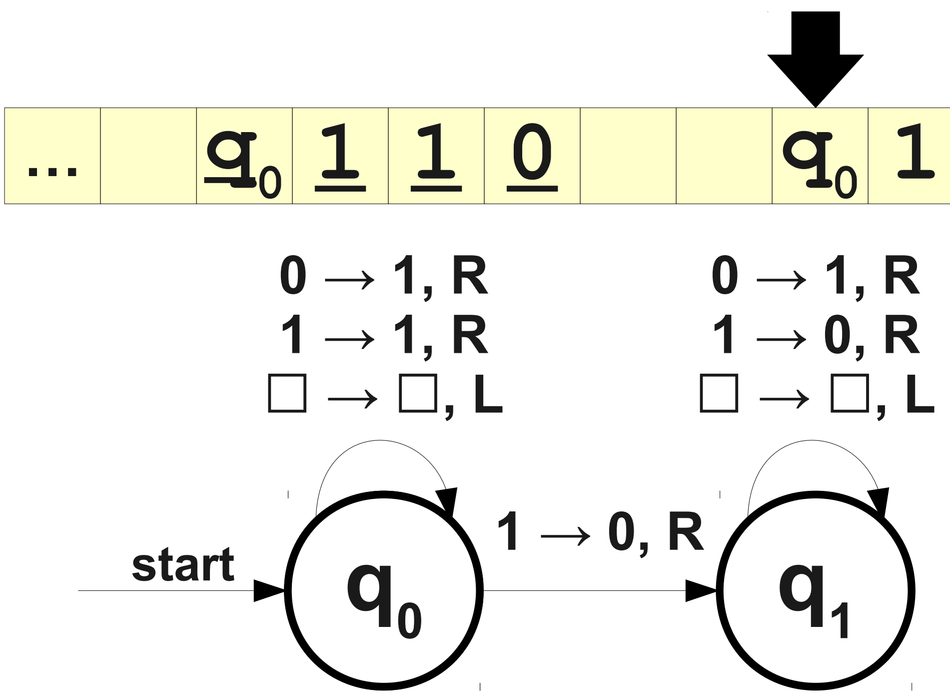




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

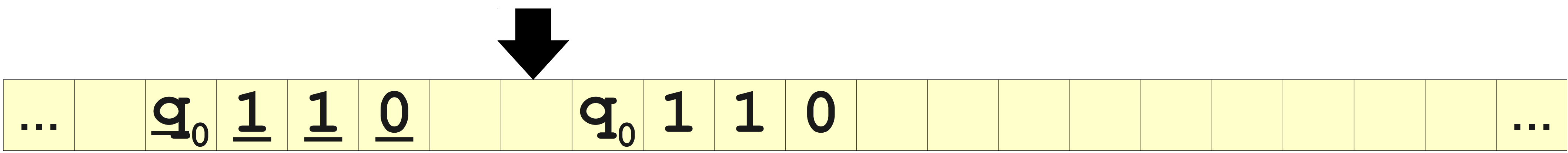
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

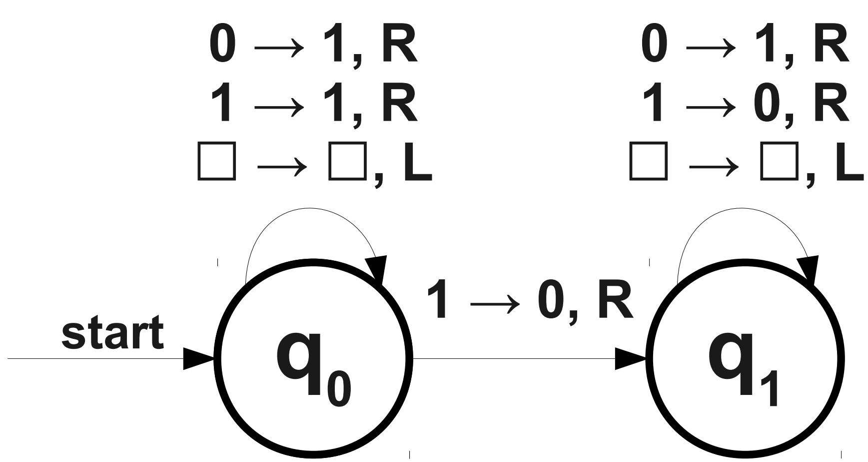


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

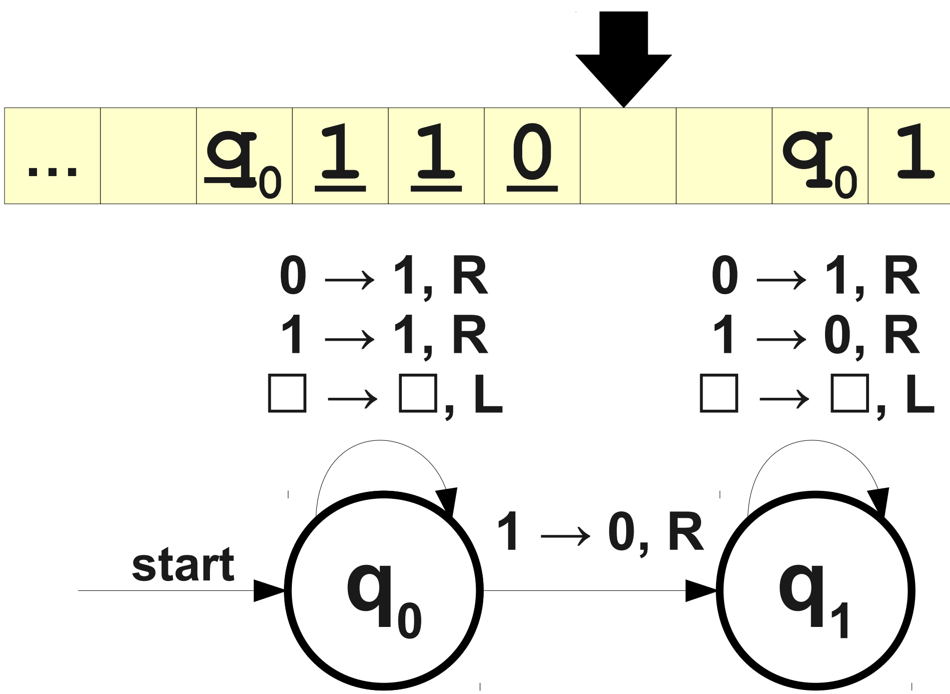




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

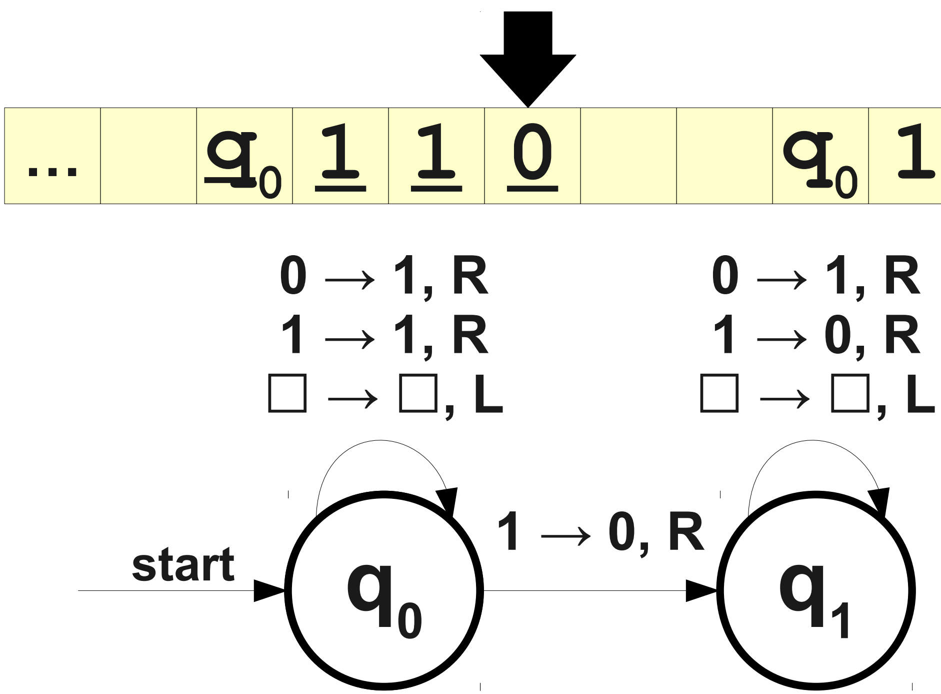
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

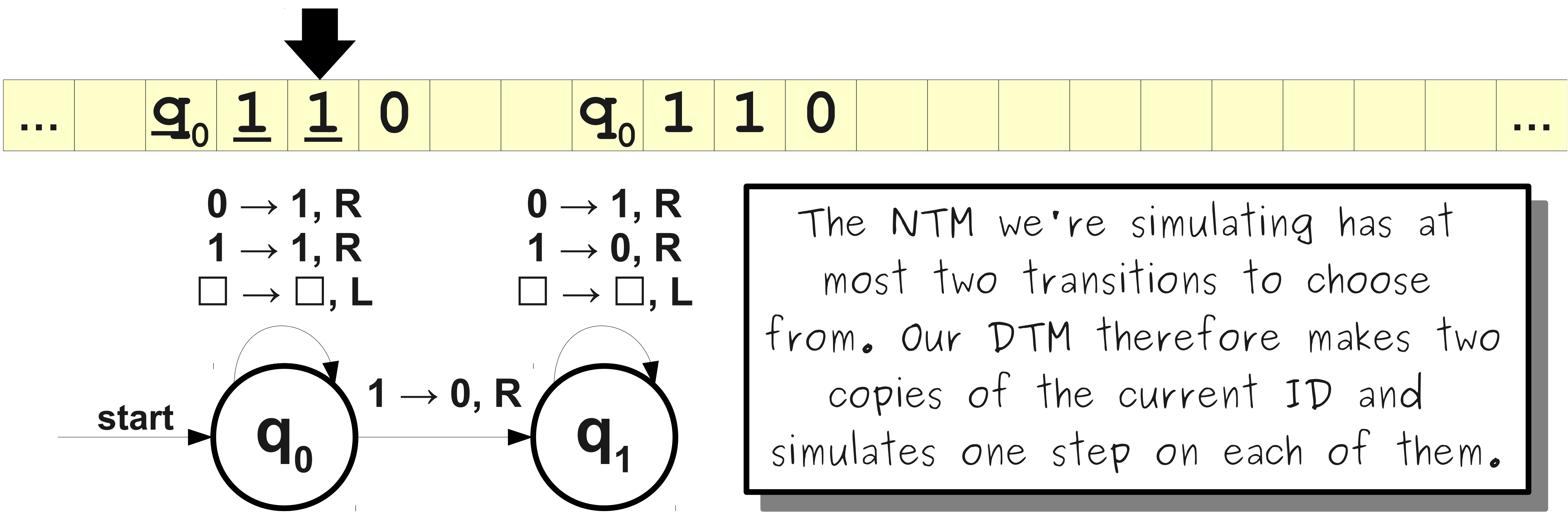
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

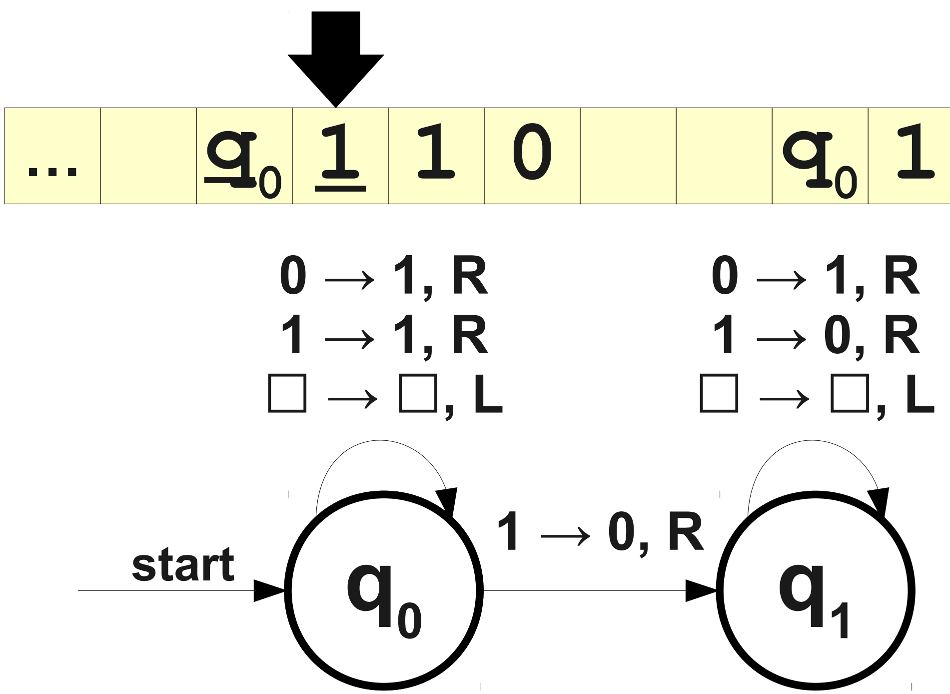
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

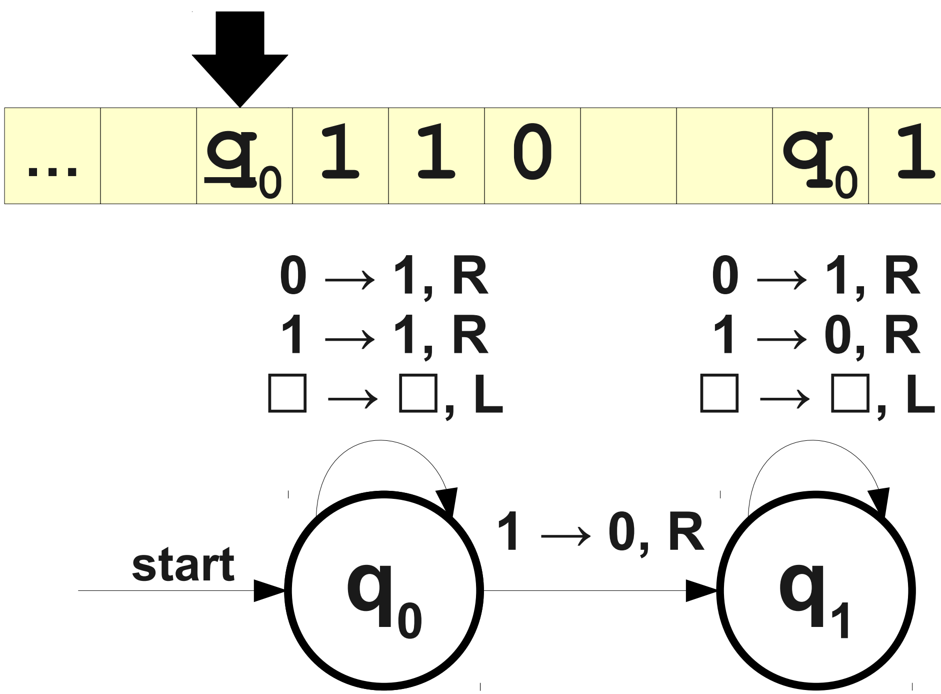
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

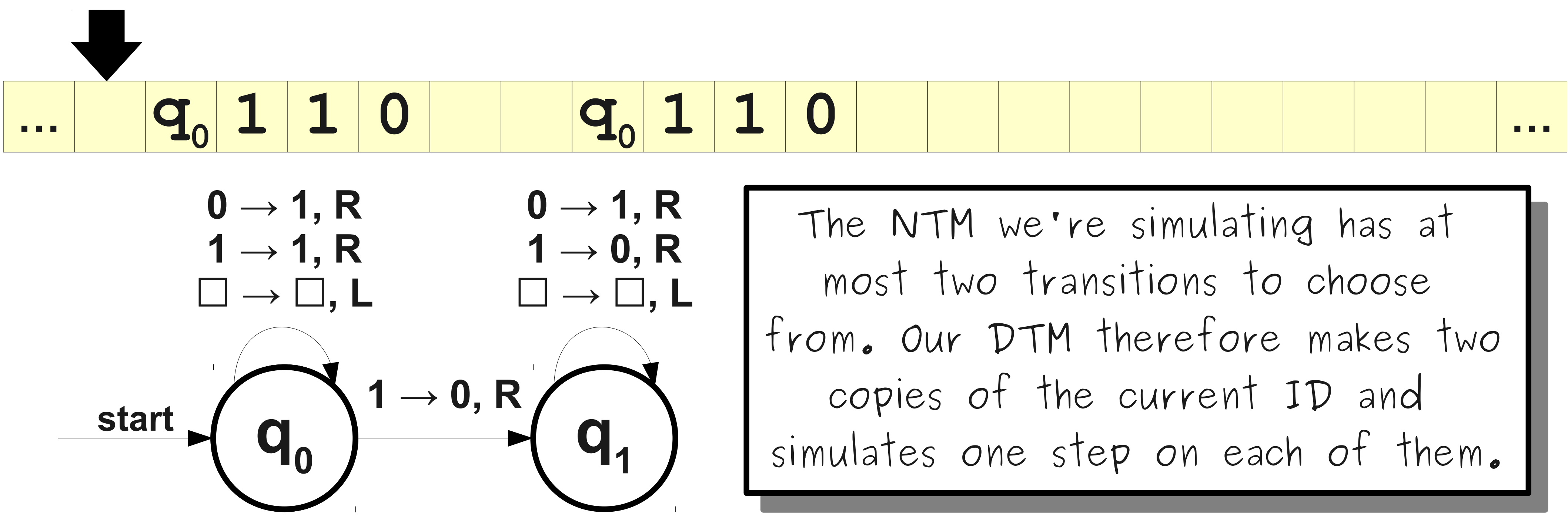
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

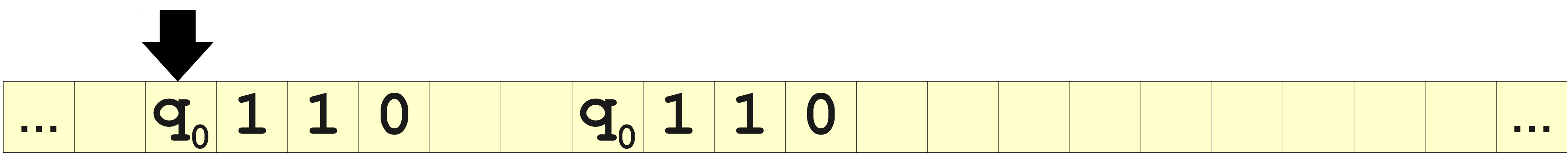
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

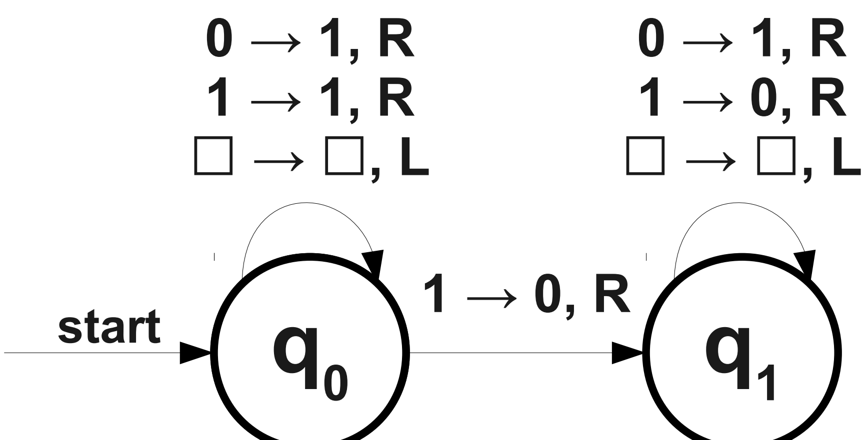


The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

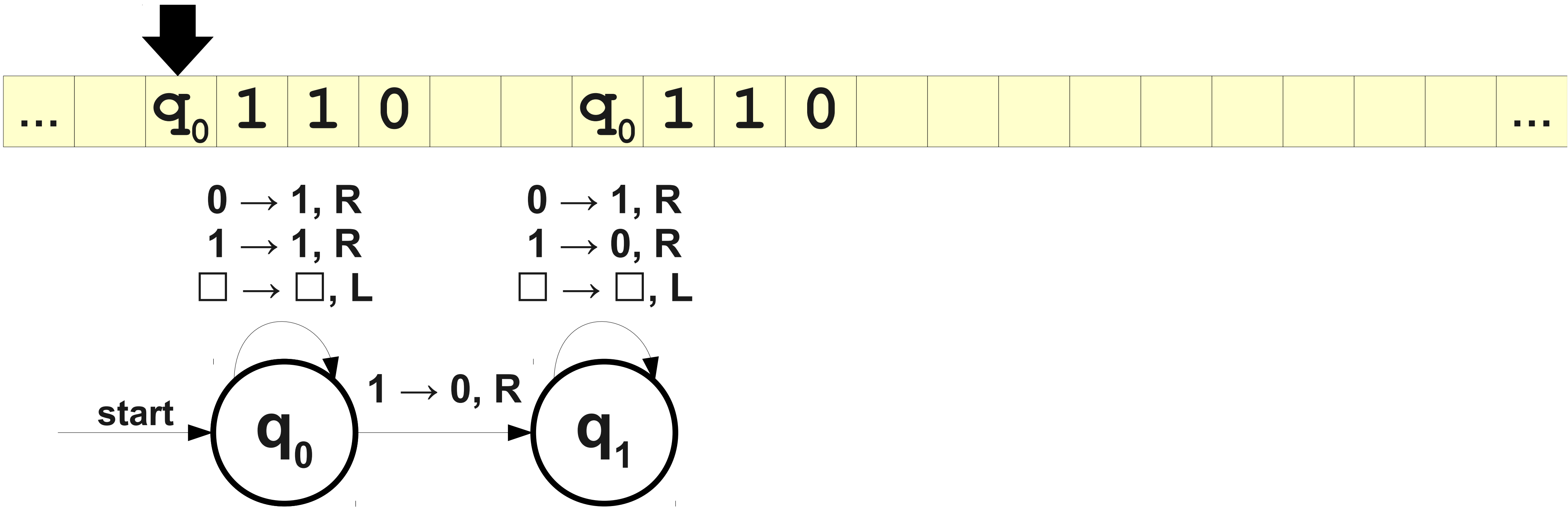




The NTM we're simulating has at most two transitions to choose from. Our DTM therefore makes two copies of the current ID and simulates one step on each of them.

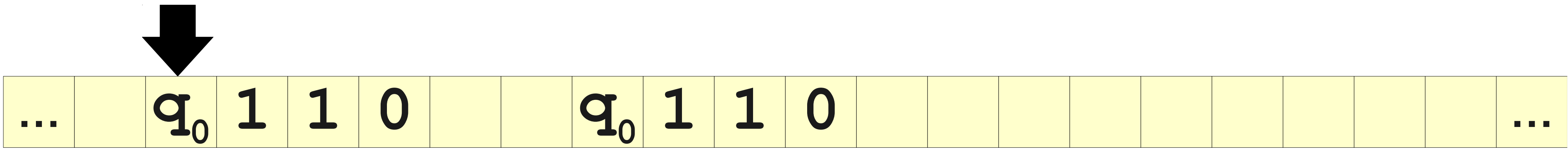
Putting Everything Together

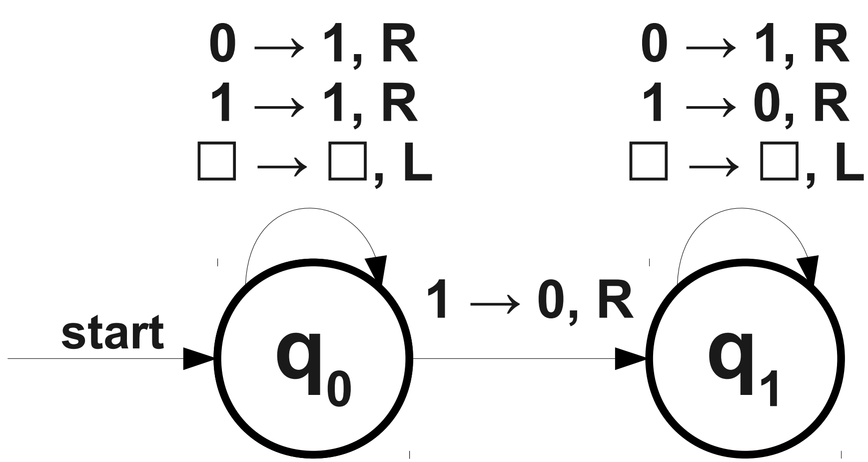
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

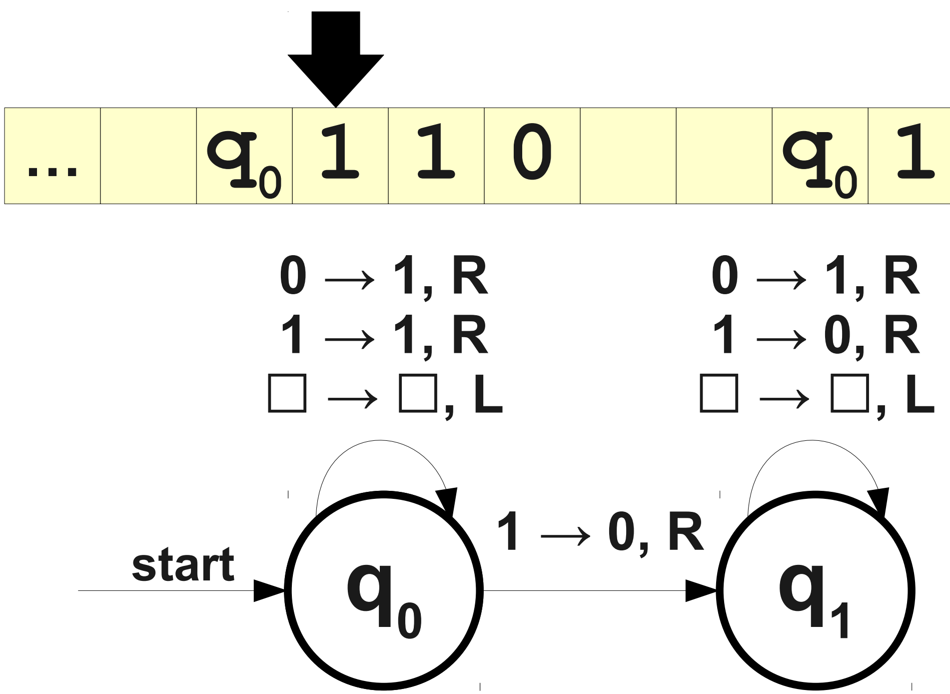




Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

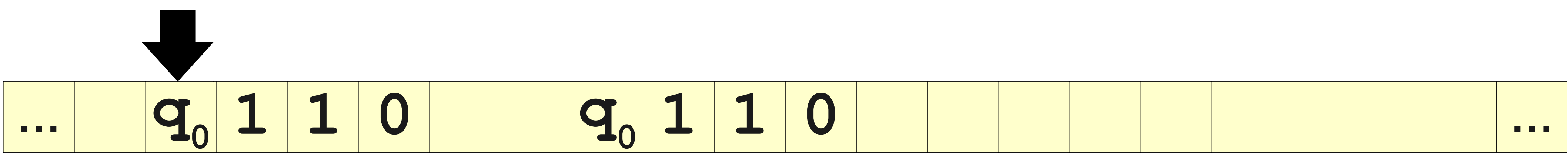
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

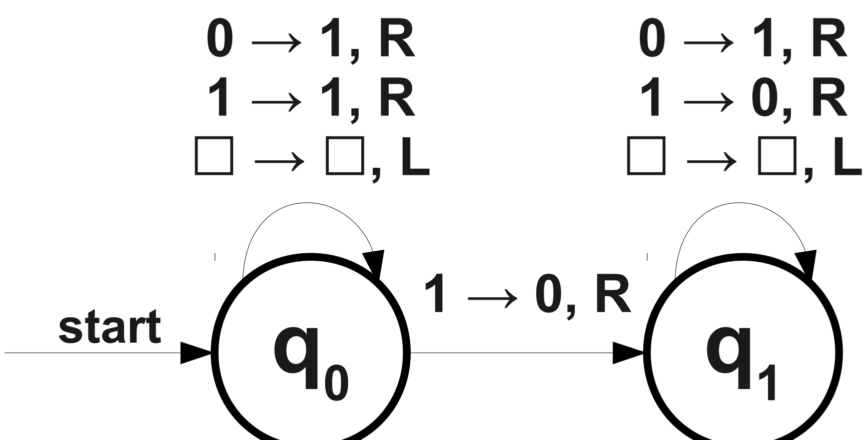


Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

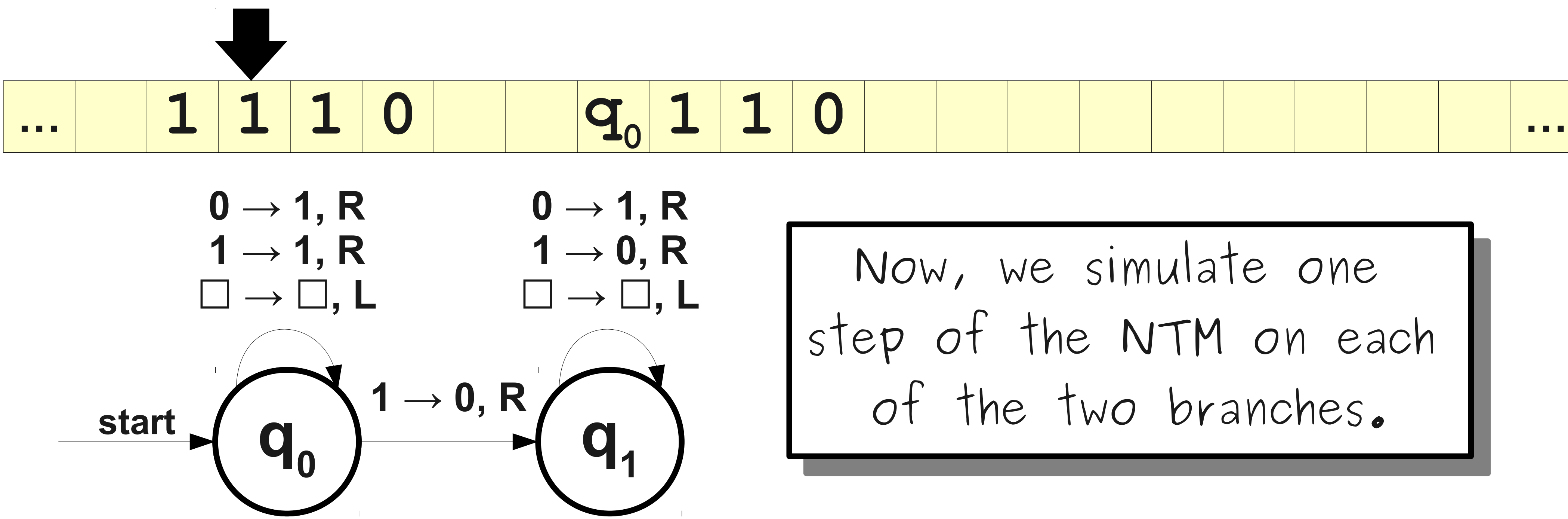




Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

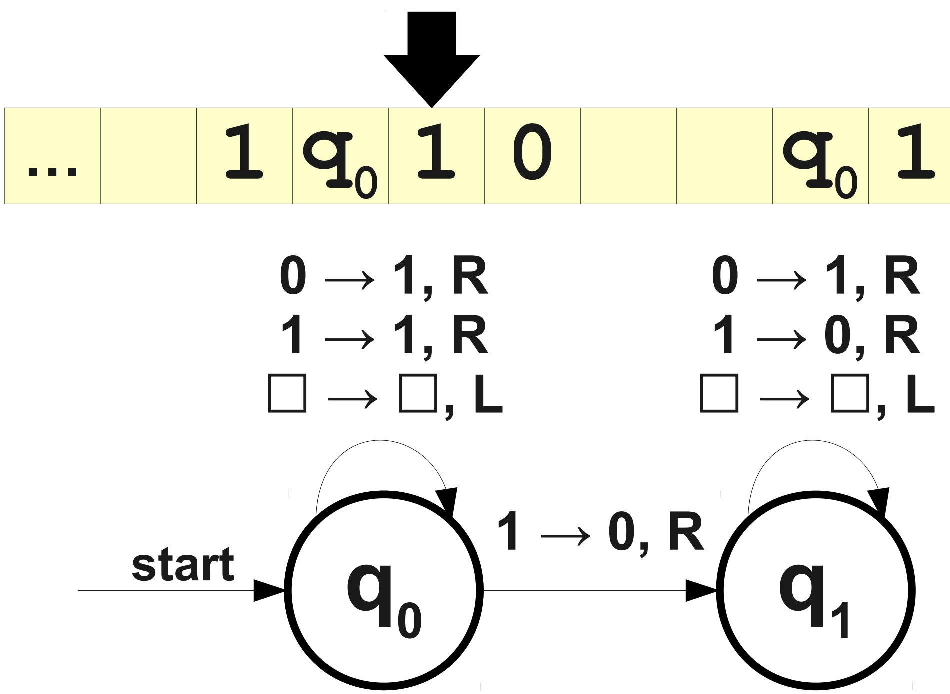
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

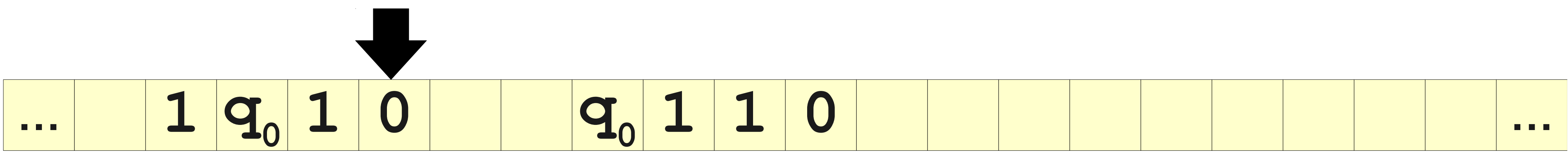
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

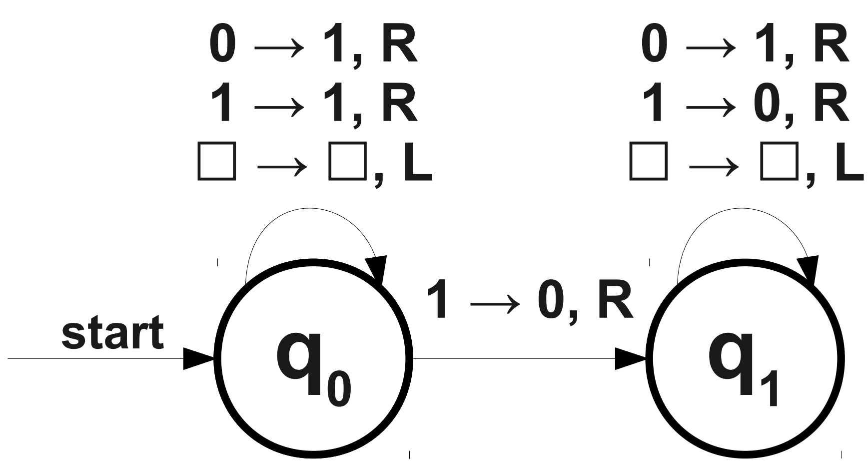


Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

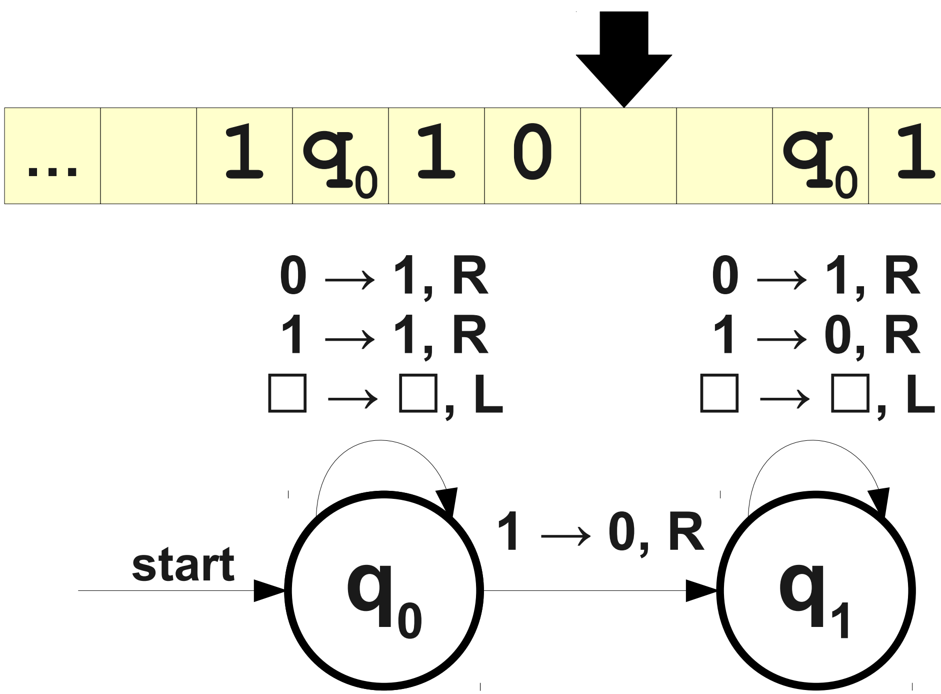




Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

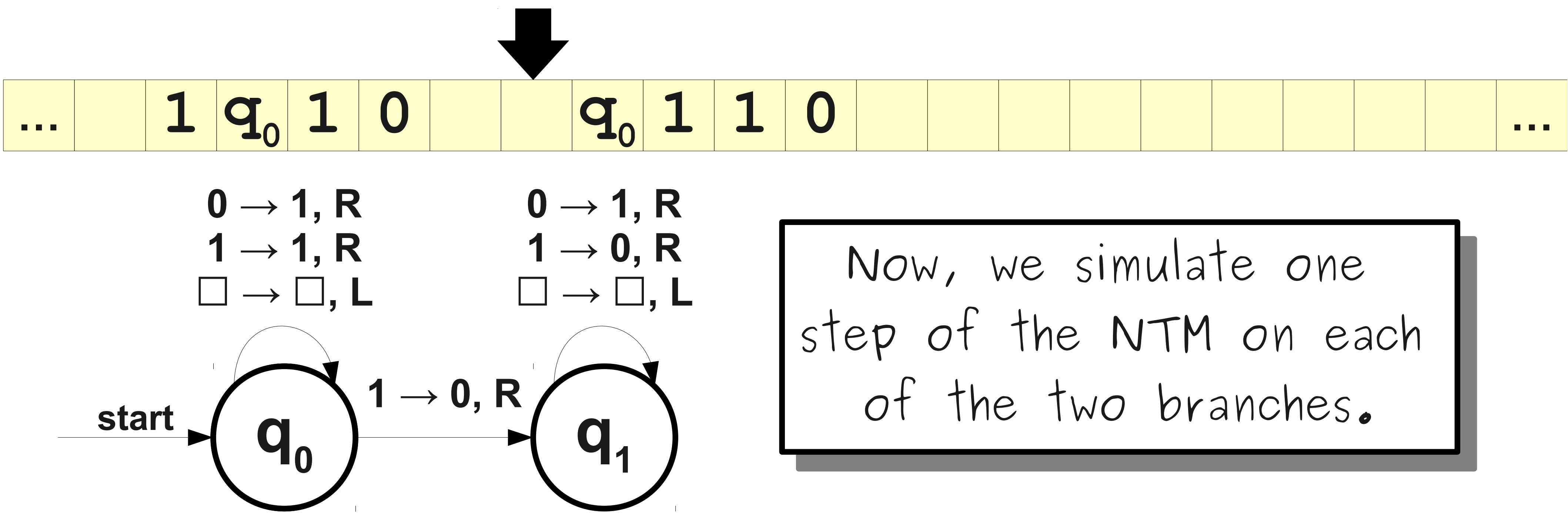
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

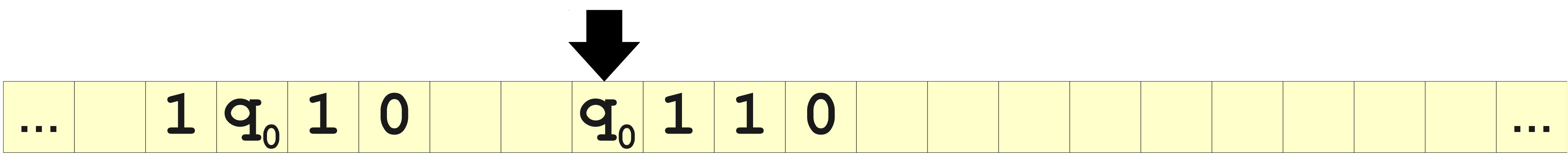
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

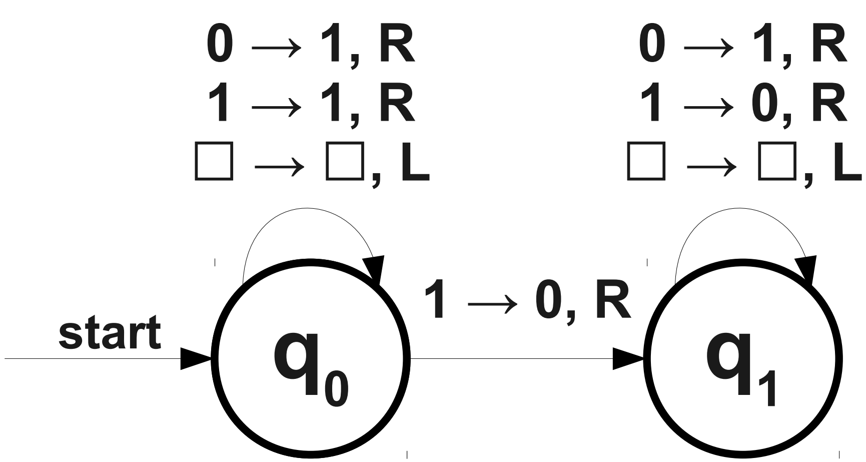


Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

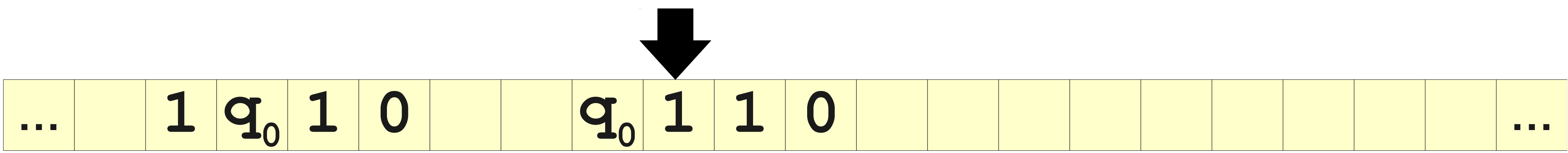


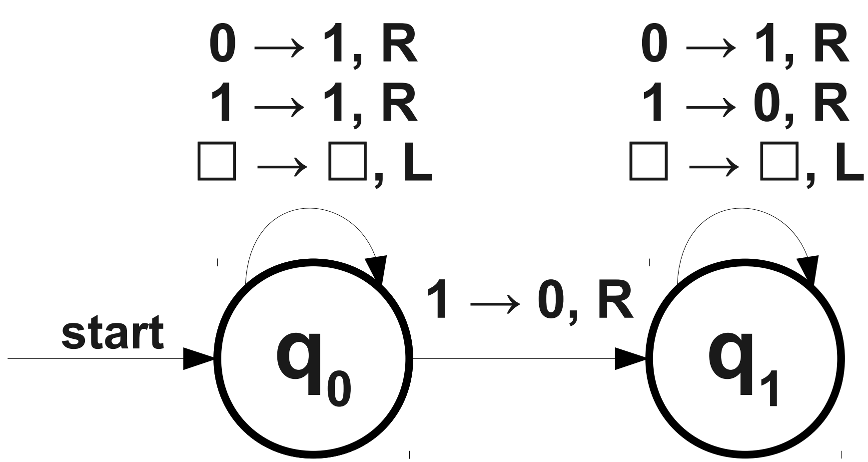


Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

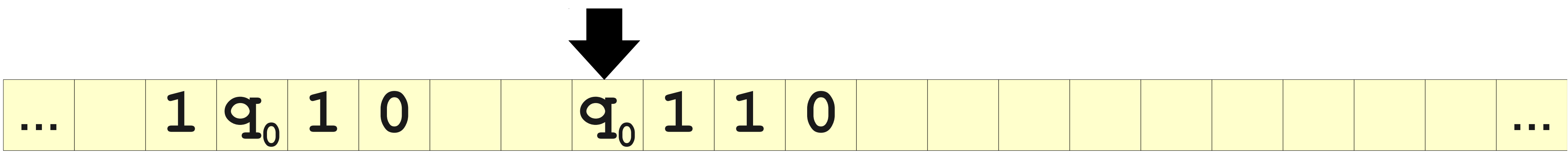


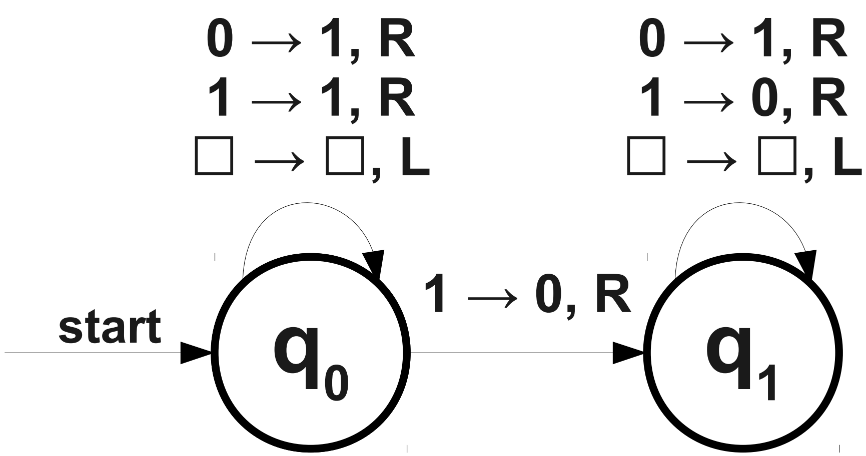


Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

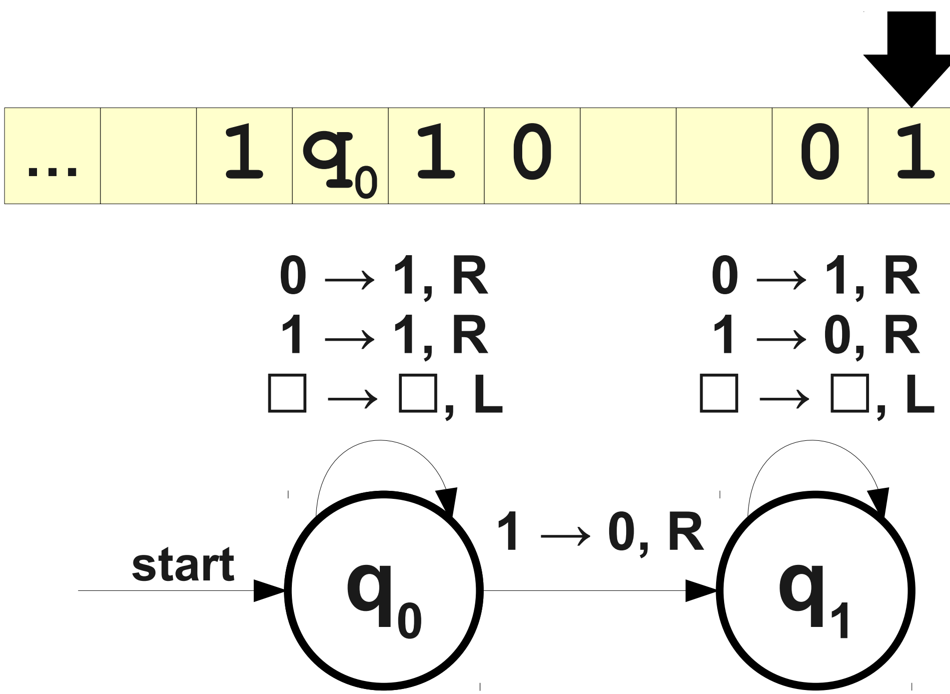




Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

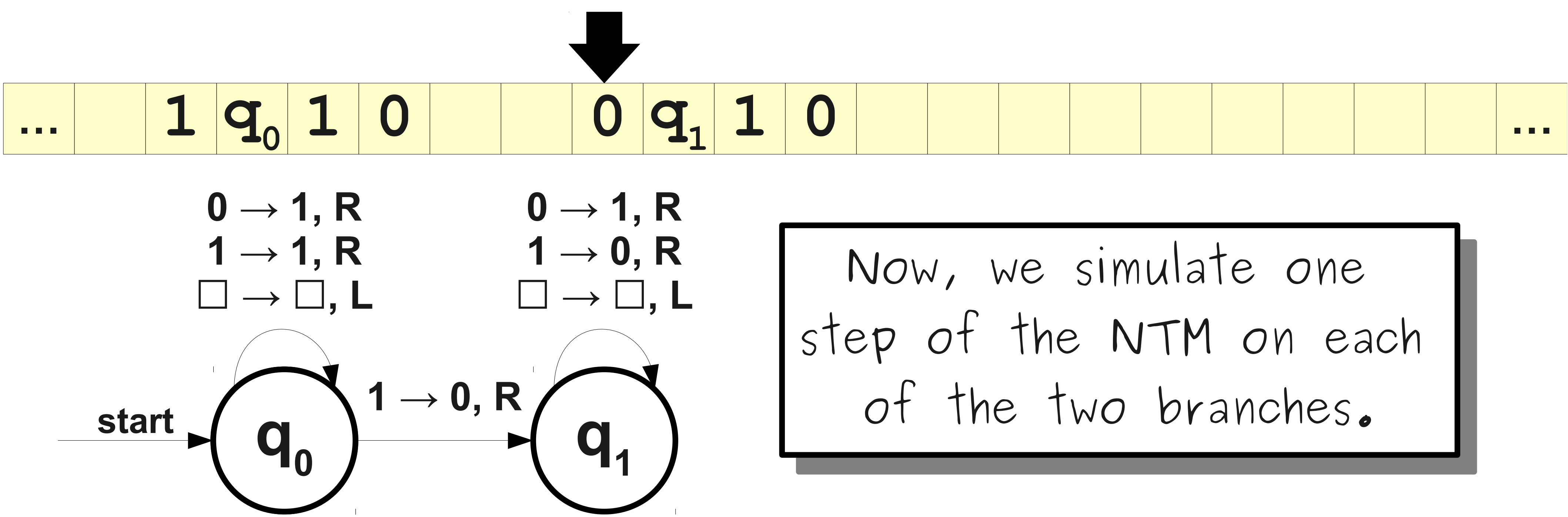
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

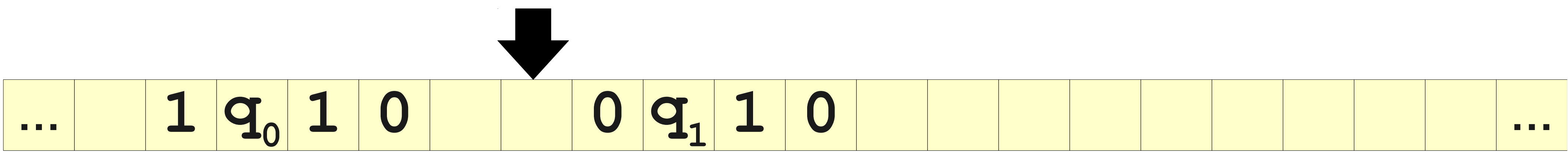
Putting Everything Together

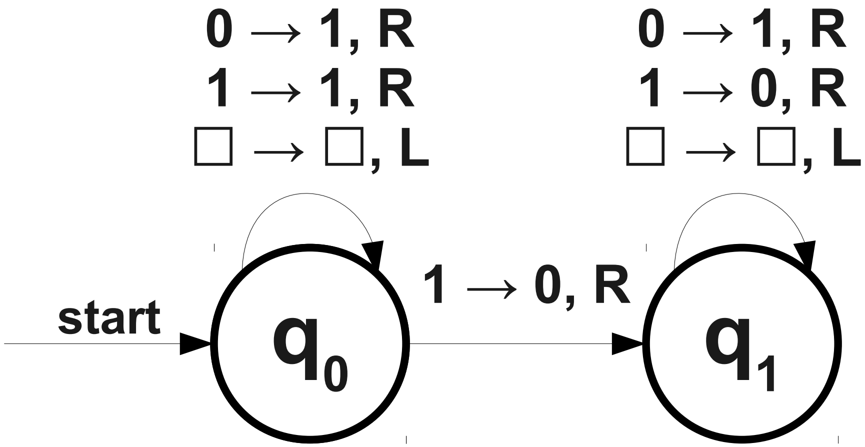
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

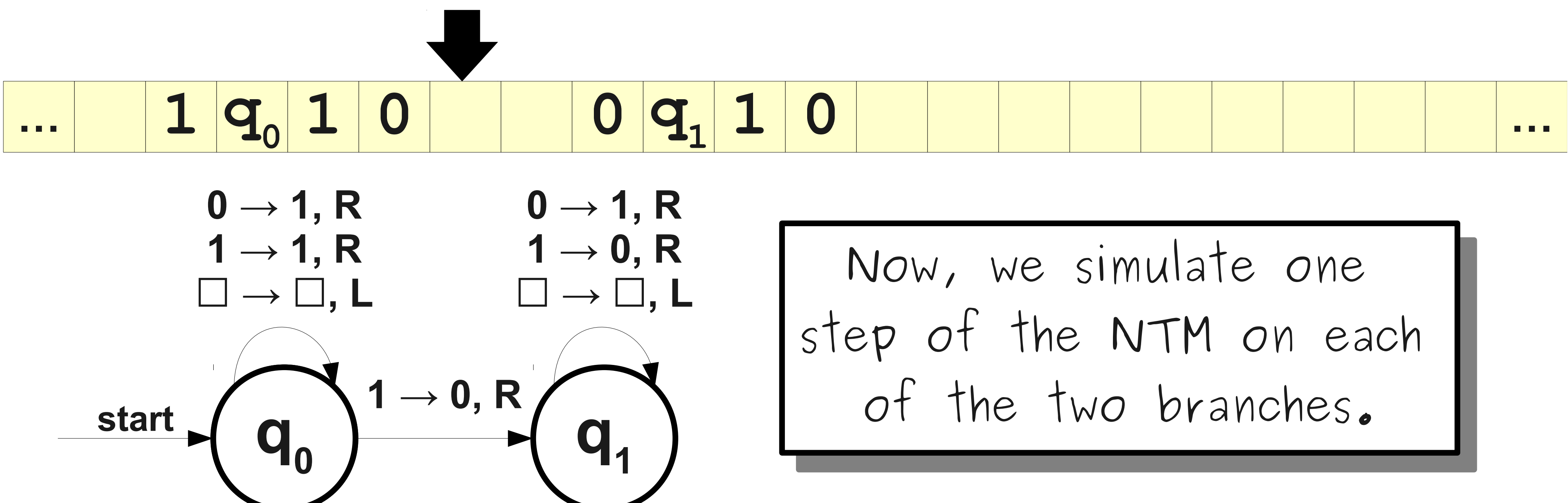




Now, we simulate one step of the NTM on each of the two branches.

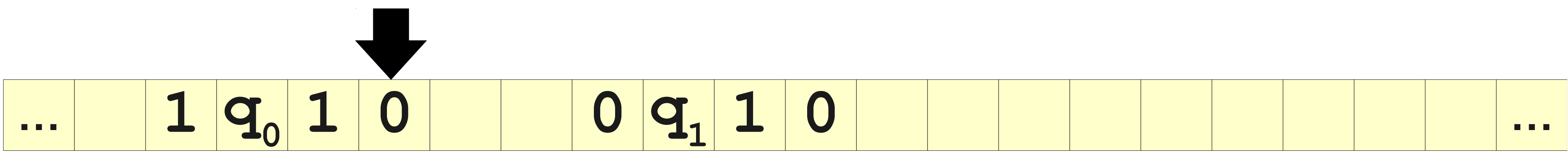
Putting Everything Together

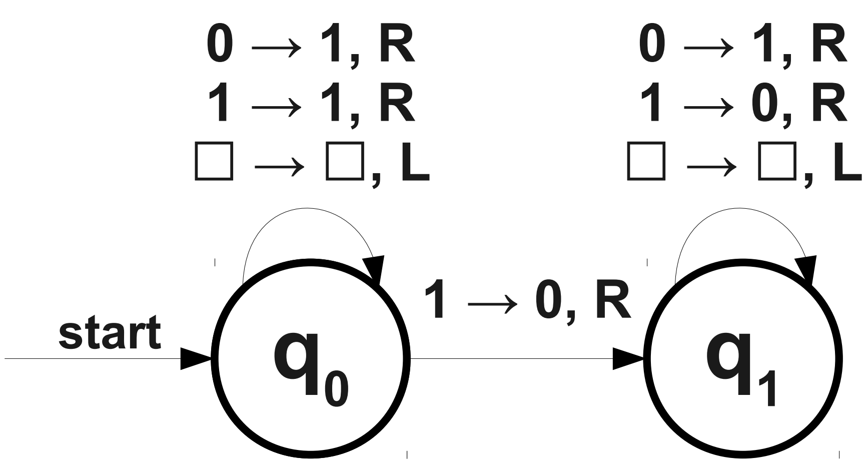
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

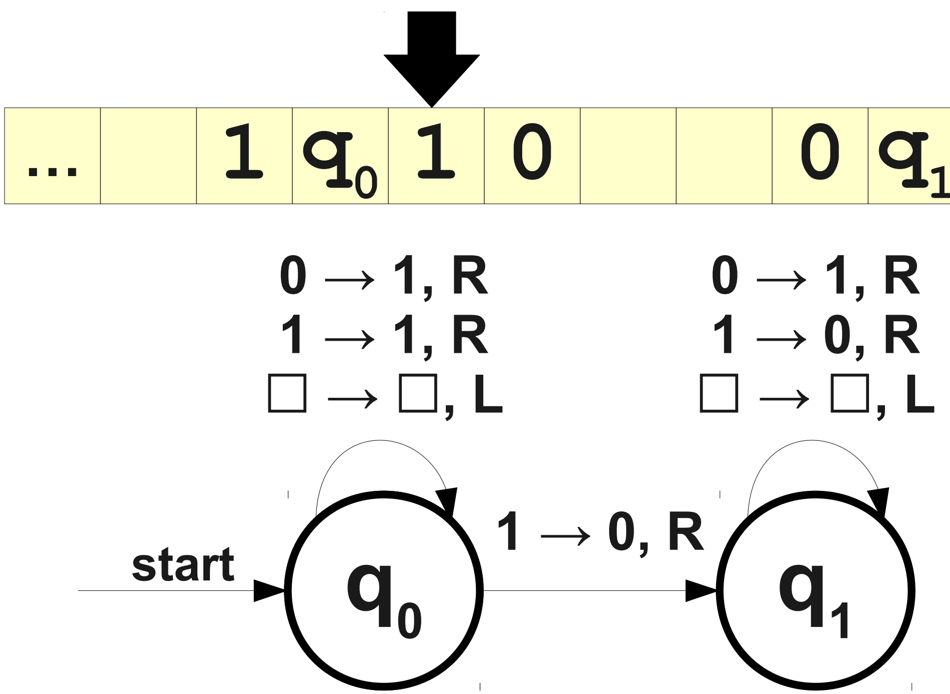




Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

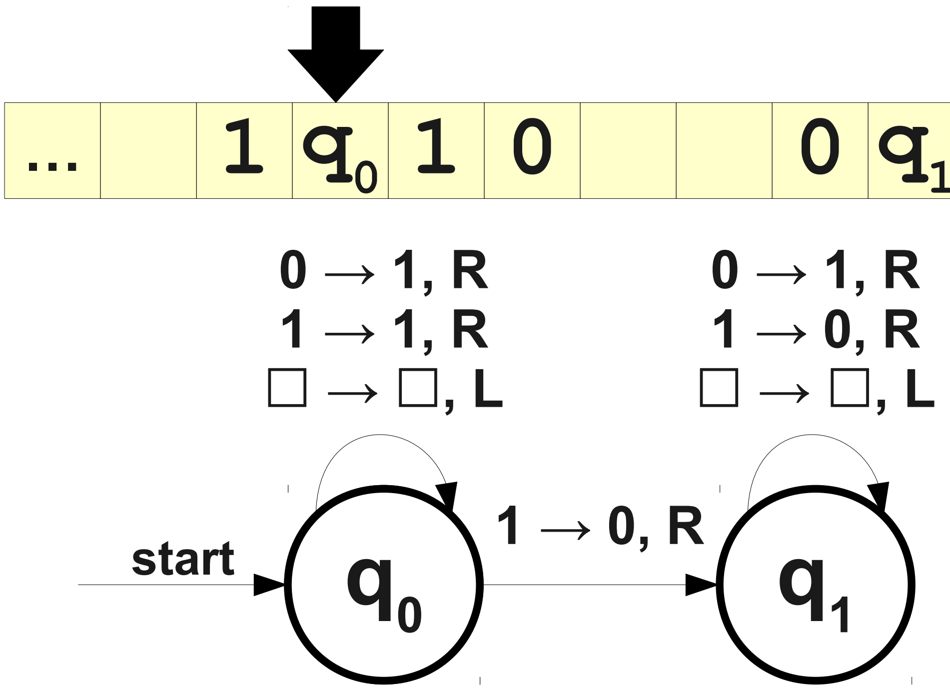
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

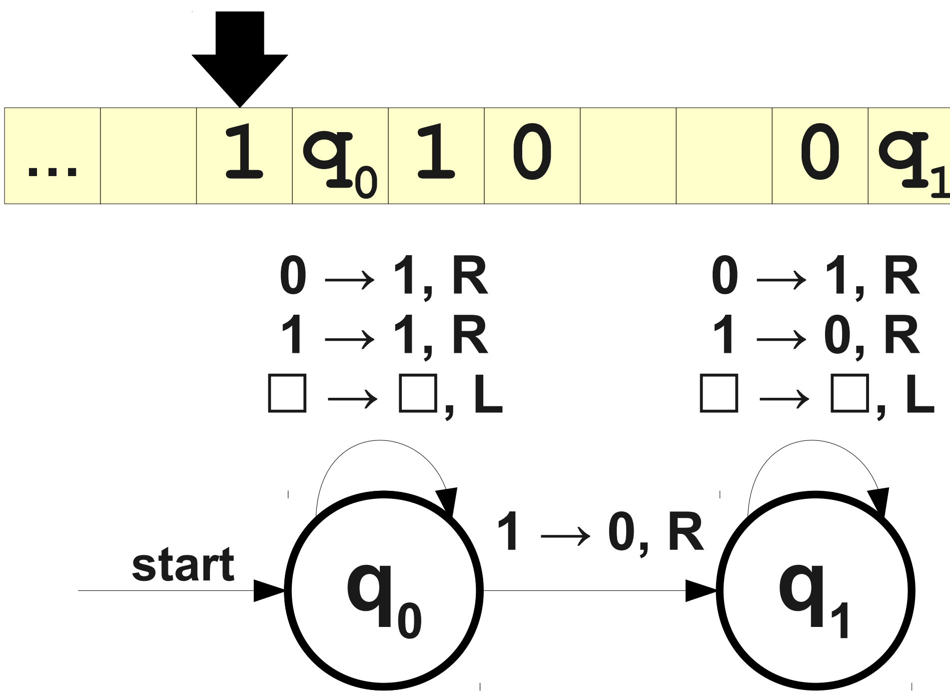
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

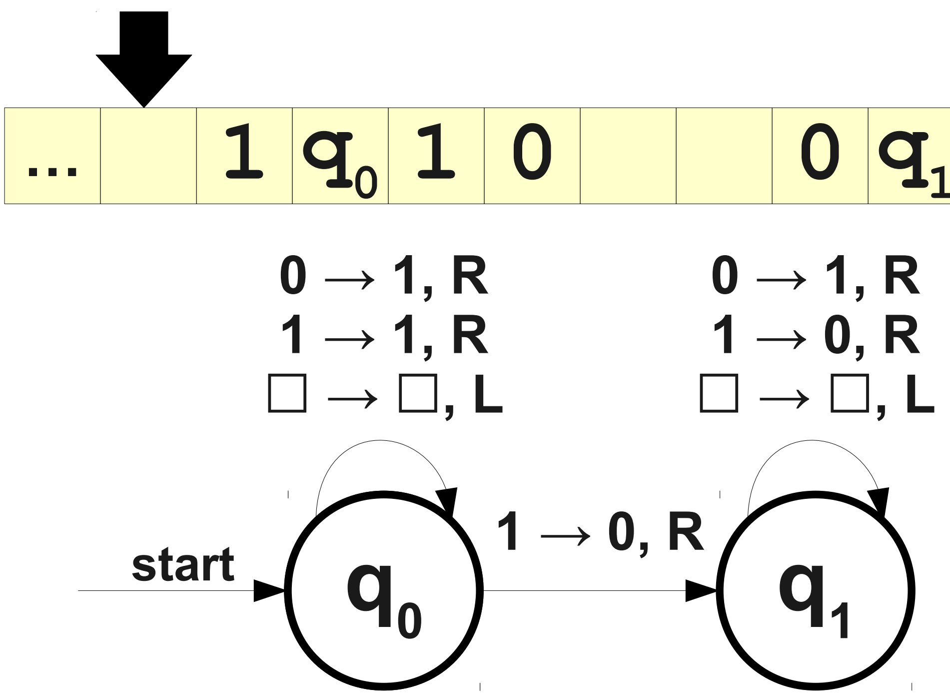
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

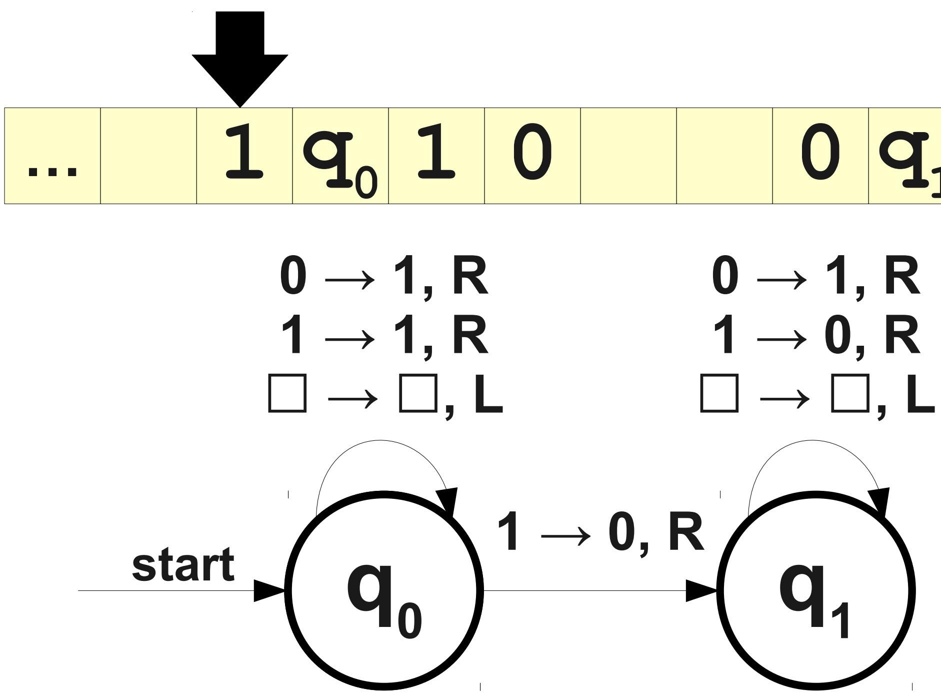
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

Putting Everything Together

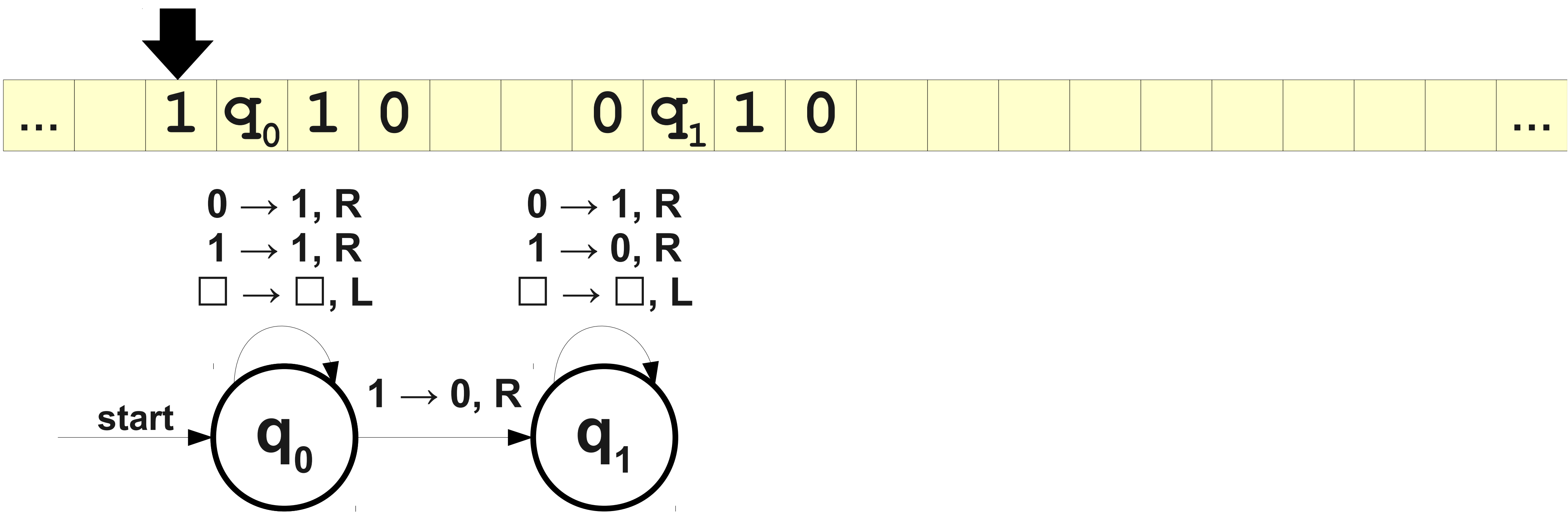
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Now, we simulate one step of the NTM on each of the two branches.

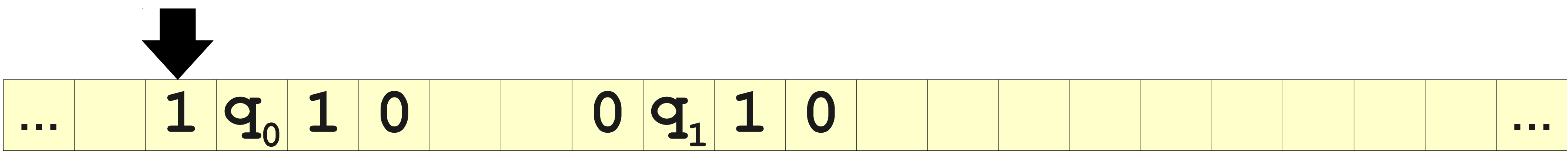
Putting Everything Together

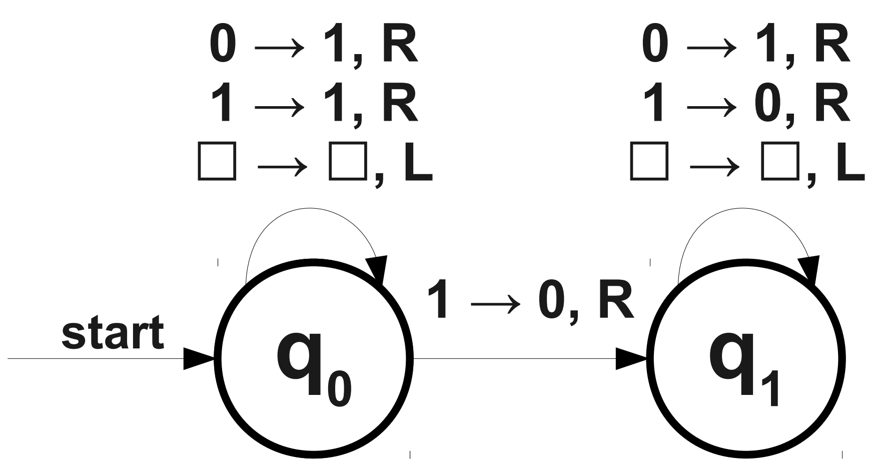
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

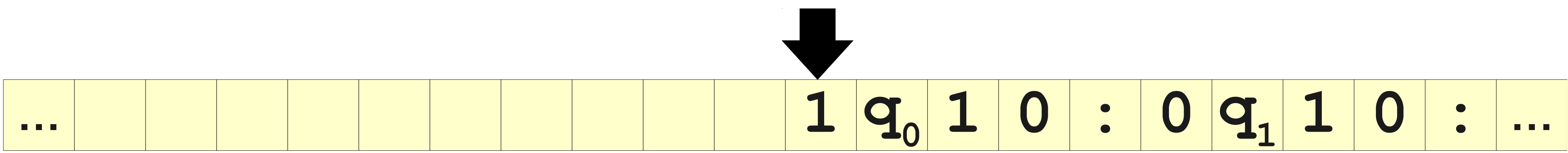


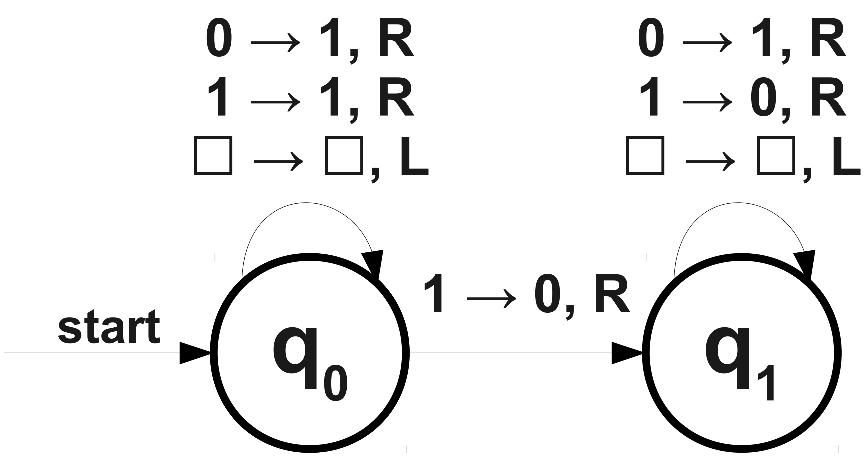


Next, the DTM copies these strings over into a worklist for future use. (I'll omit the animation.)

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

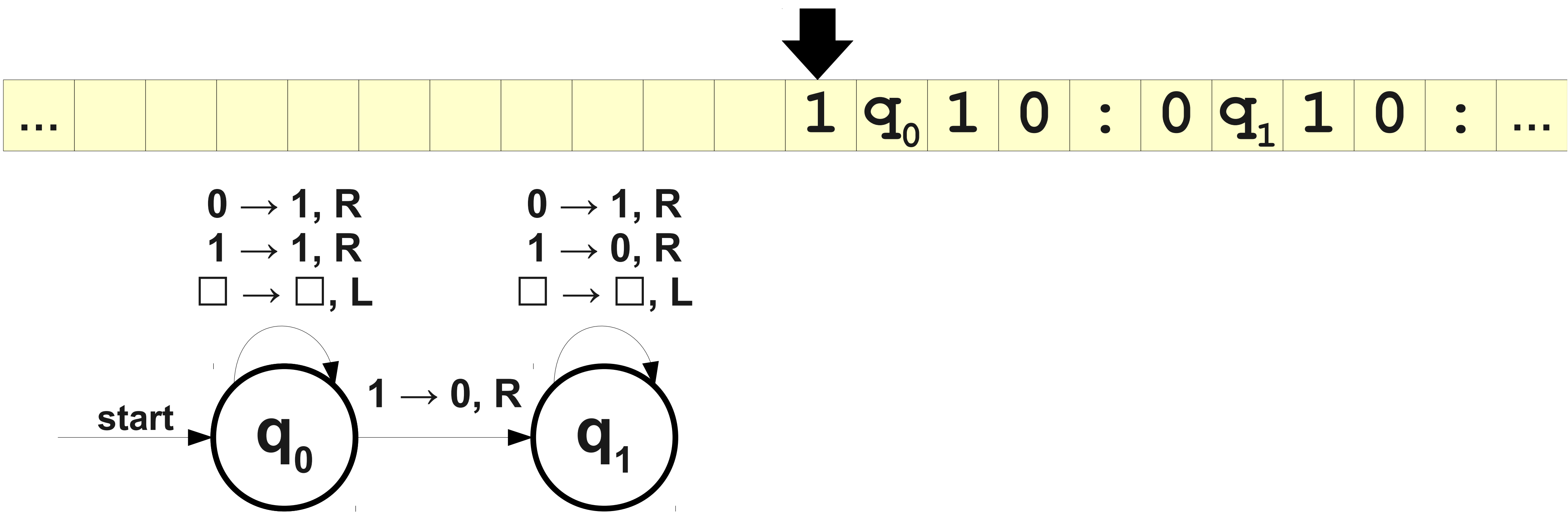




Next, the DTM copies these strings over into a worklist for future use. (I'll omit the animation.)

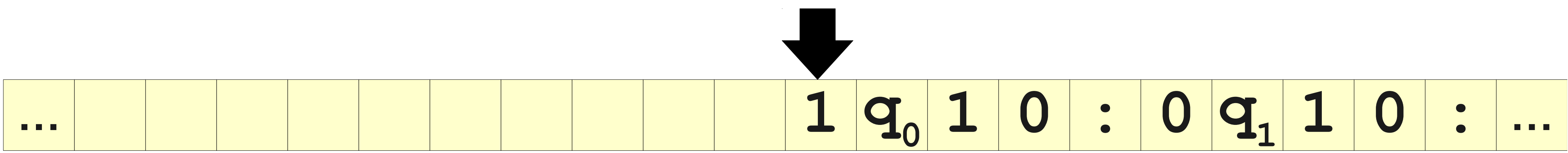
Putting Everything Together

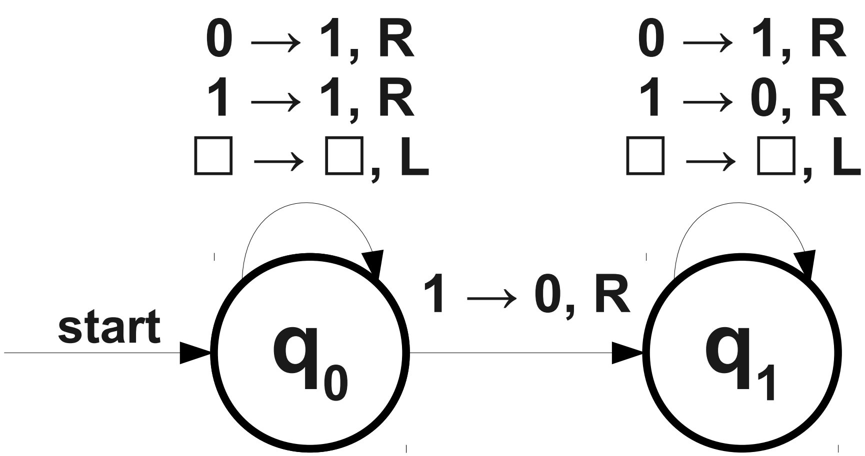
- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.



Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.

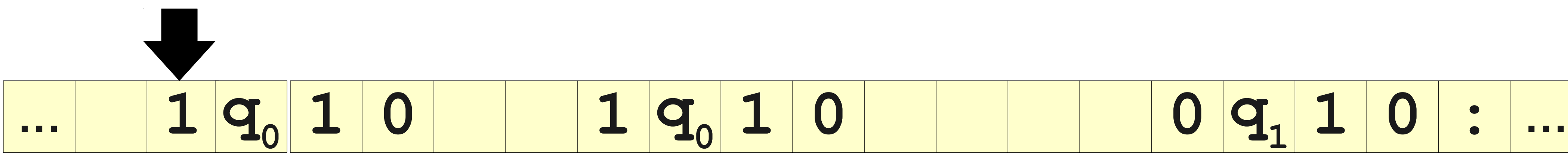


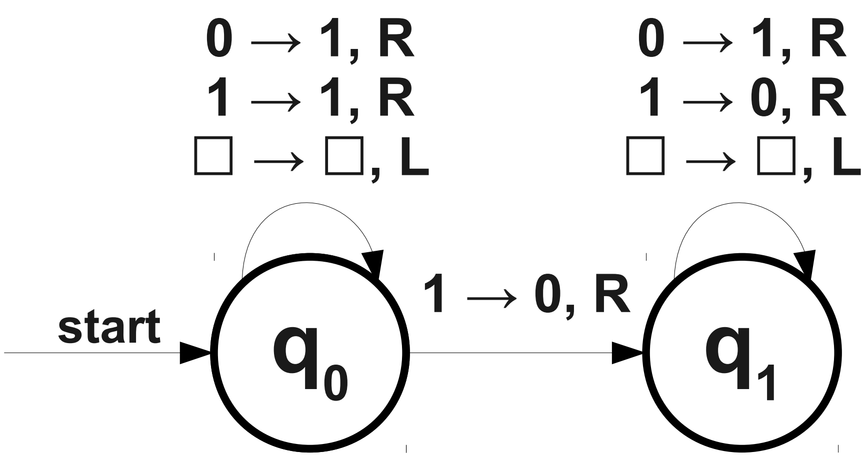


Finally, the DTM removes the next unprocessed ID from the worklist and repeats this process.

Putting Everything Together

- ***Theorem:*** For any NTM N, there exists a DTM D such that $\mathscr{L}(N) = \mathscr{L}(D)$.
- ***Proof sketch:*** D uses a worklist to exhaustively search over N's computation tree.





Finally, the DTM removes the next unprocessed ID from the worklist and repeats this process.

Schematically

Workspace	Worklist of IDs

To simulate the NTM N with a DTM D, we construct D as follows:

- On input w, D converts w into an initial ID for N starting on w.
- While D has not yet found an accepting state:
  - D finds the next ID for N from the worklist.
  - D copies this ID once for each possible transition.
  - D simulates one step of the computation for each of these IDs.
  - D copies these IDs to the back of the worklist.

Why All This Matters

- The equivalence of NTMs and DTMs should be absolutely astounding!
  - Nondeterministic TMs can magically guess a single correct option out of infinitely many options.
  - Deterministic TMs can just zip back and forth across the tape.
- This suggests that Turing machines are extremely powerful.

Just how powerful **are** Turing machines?

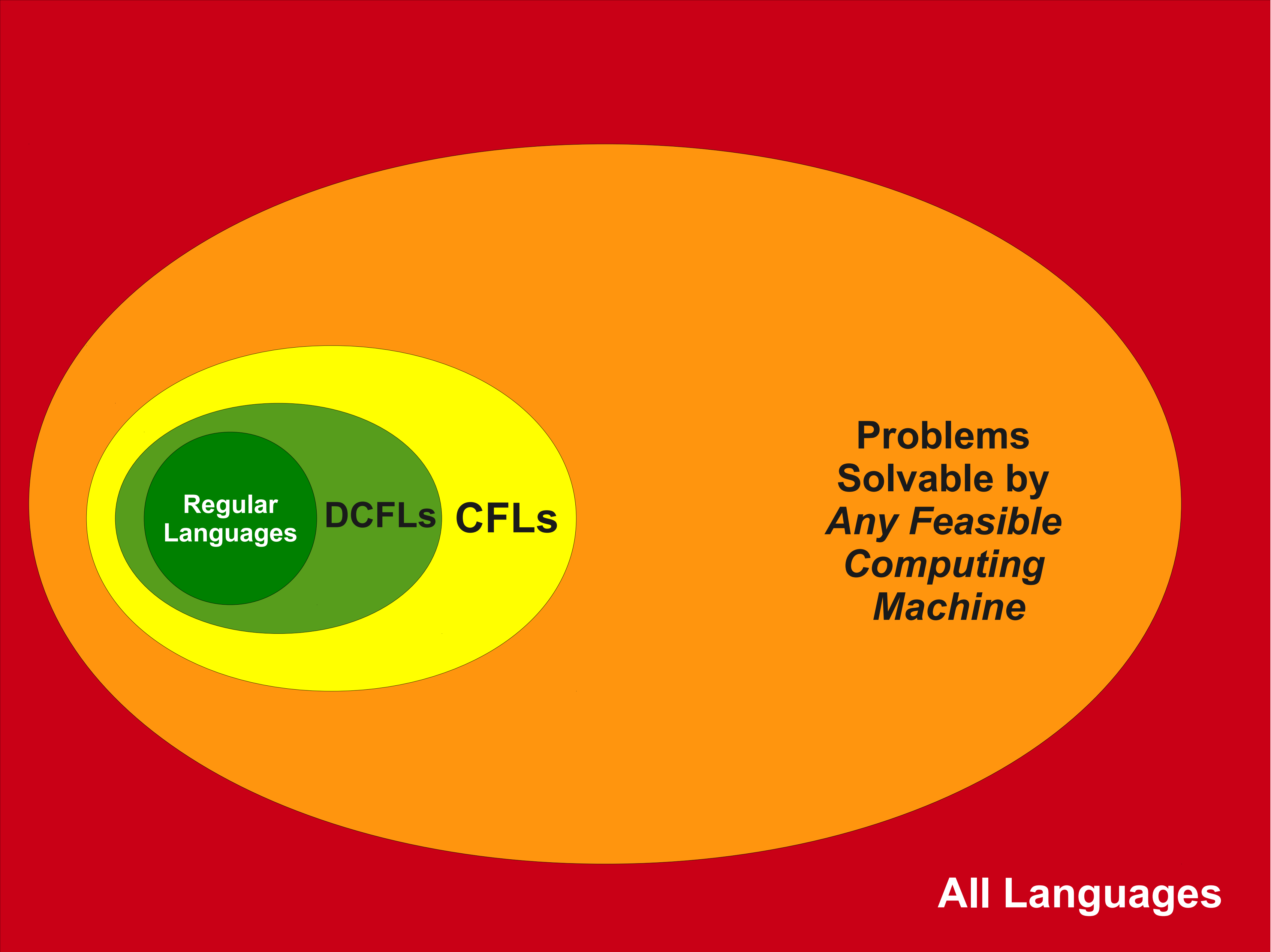
Effective Computation

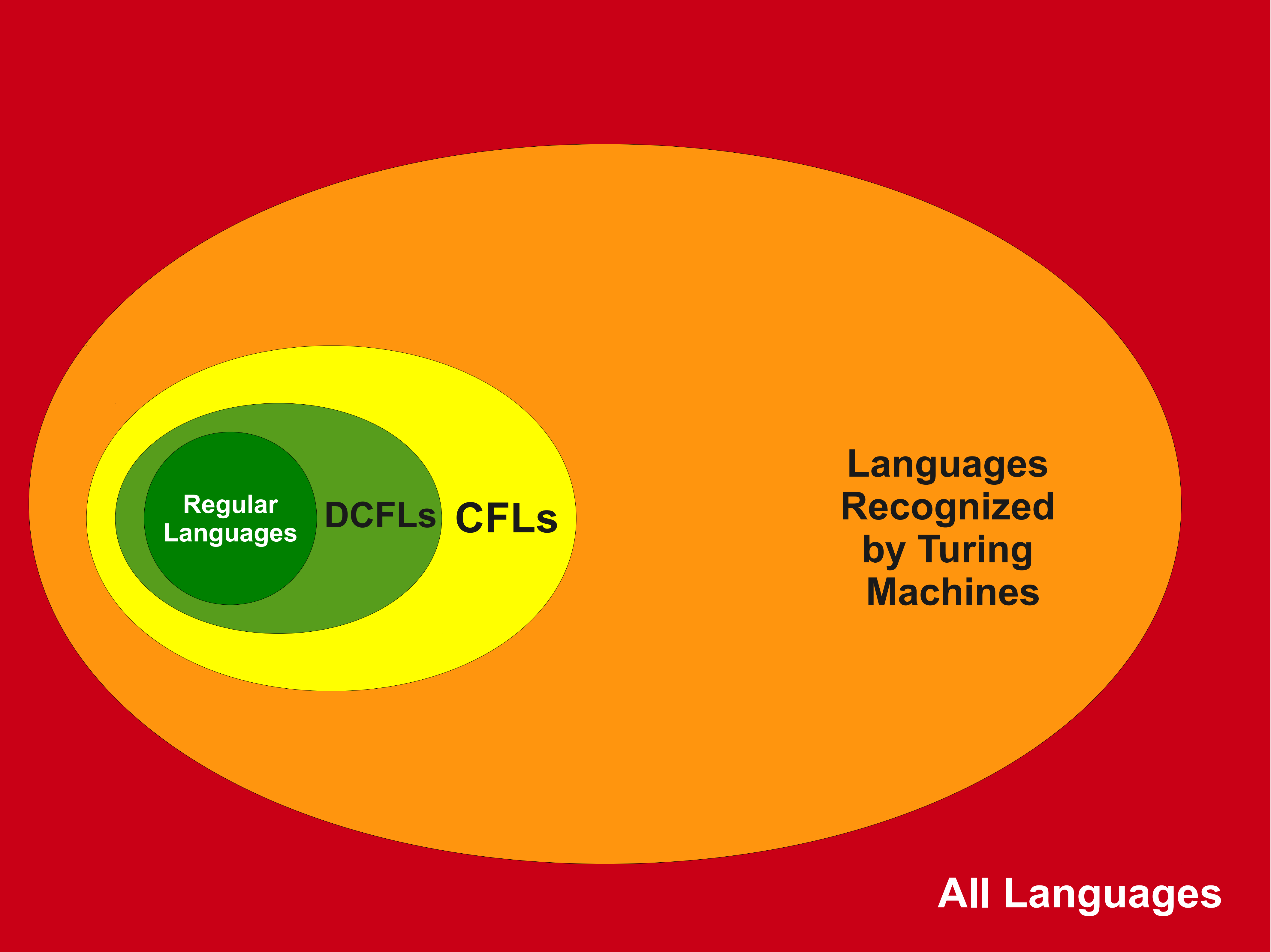
- An **effective method of computation** is a form of computation with the following properties:
  - The computation consists of a set of steps.
  - There are fixed rules governing how one step leads to the next.
  - Any computation that yields an answer does so in finitely many steps.
  - Any computation that yields an answer always yields the correct answer.

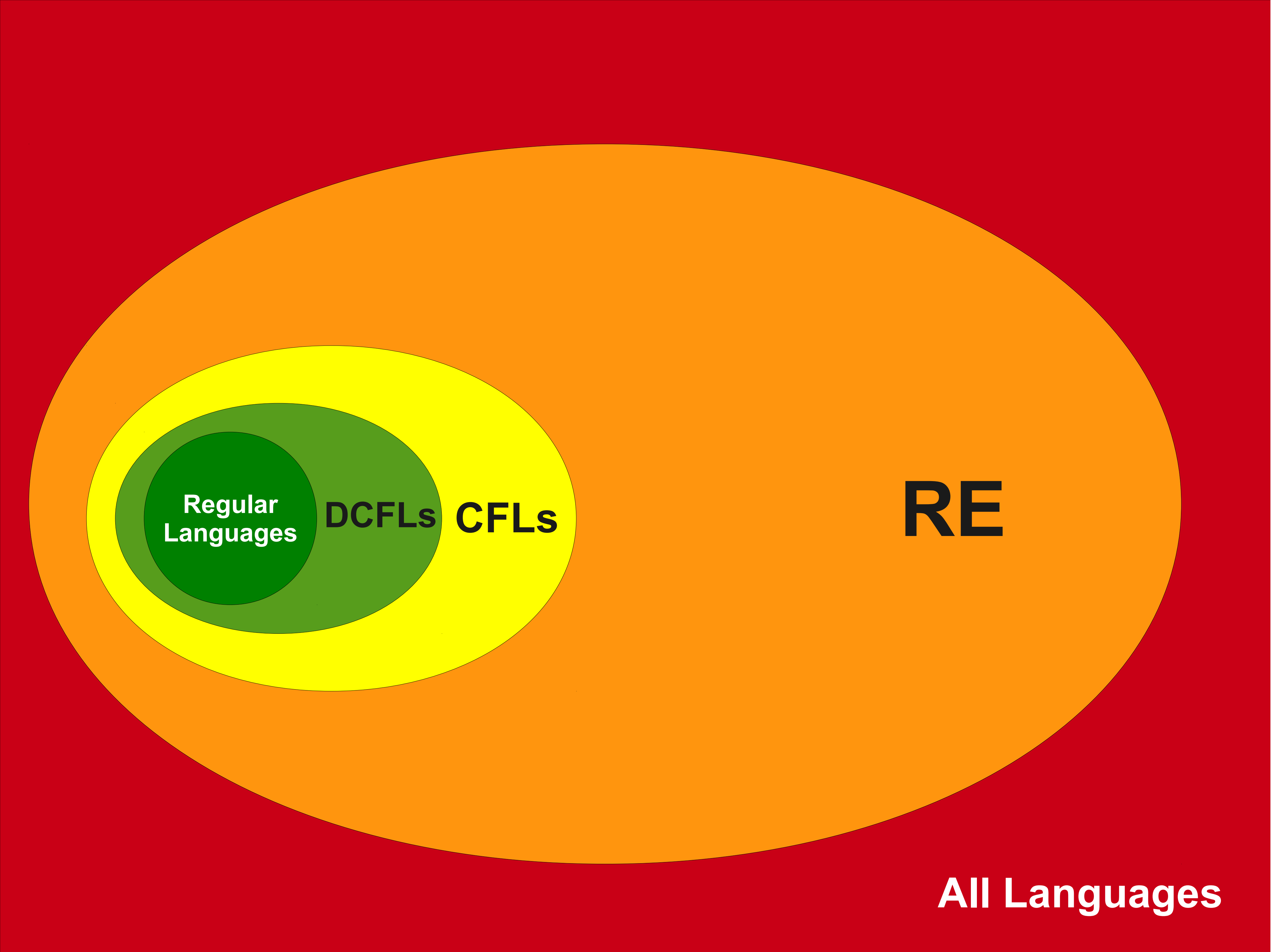
The **Church-Turing Thesis** states that

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


Regular Languages
DCFLs
CFLs
Languages Recognized by Turing Machines
All Languages


Regular Languages
DCFLs
CFLs
RE
All Languages

Next Time

- **Encodings**
  - How do we compute over arbitrary objects?
- **The Universal Turing Machine**
  - Can TMs compute over themselves?
- **The Limits of Turing Machines**
  - A language not in **RE**.