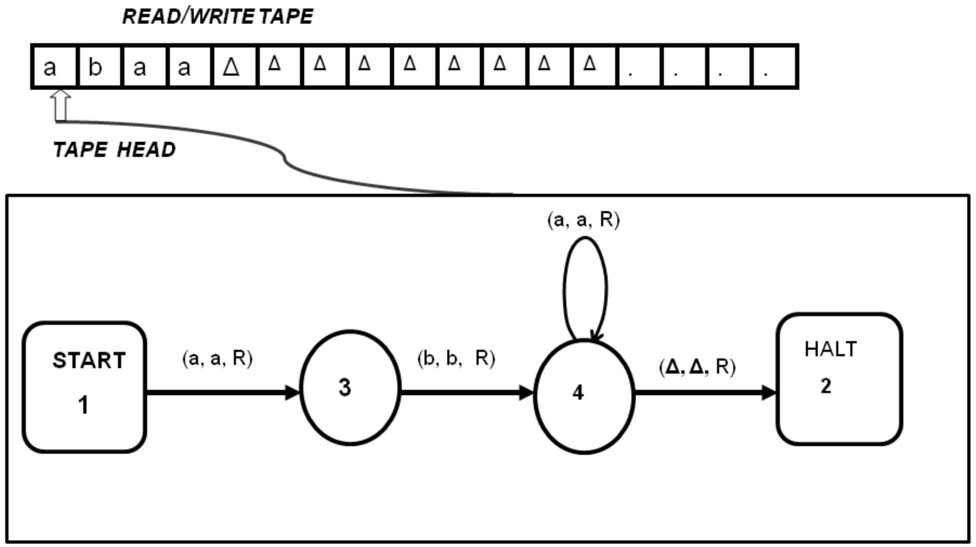
Turing Machines

AUTOMATA – Automata and Theory and Formal Languages

What is a Turing Machine?

A **Turing Machine (TM)** is a **theoretical computing model** that helps us understand what can be computed and how efficiently. It was introduced by **Alan Turing** in 1936 and is a fundamental concept in computer science and the theory of computation.

What is a Turing Machine?



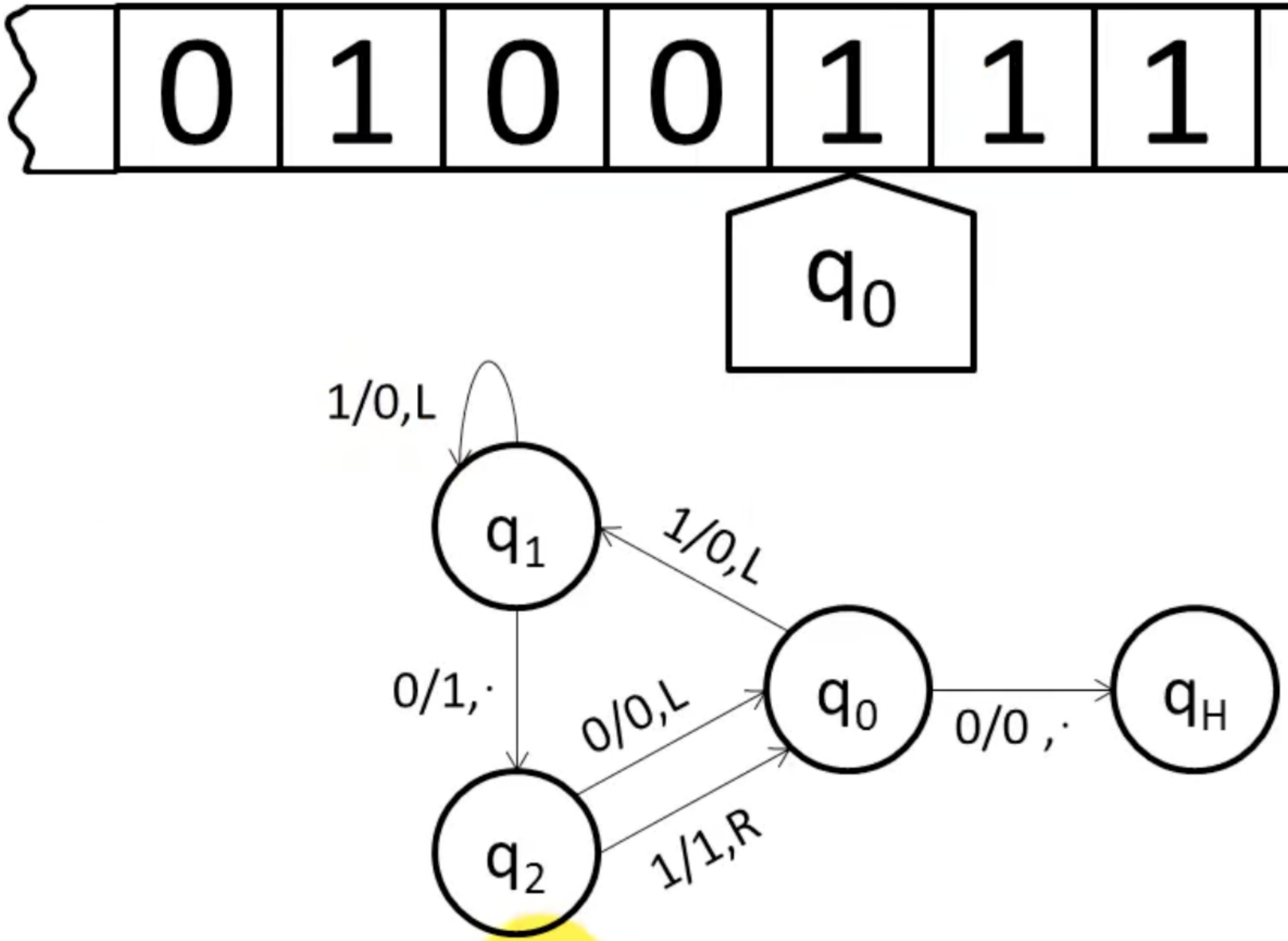
A Turing Machine for aba*

- Think of it as an **idealized computer** that can read and write symbols on an infinitely long tape. Even though it's not a physical machine, it serves as a model for understanding how algorithms work.

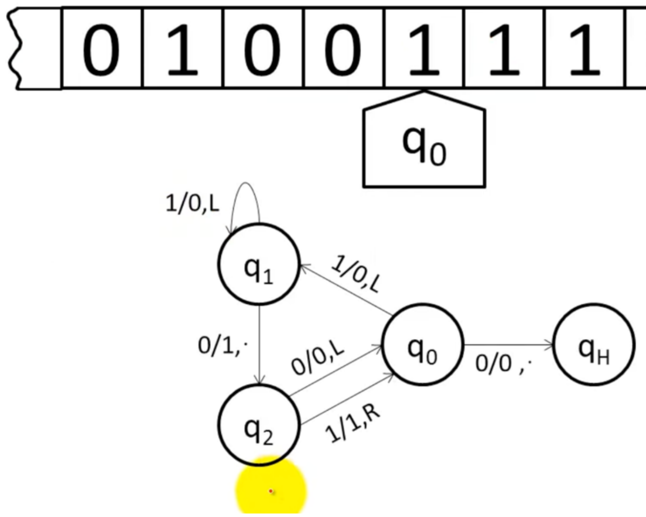
How Does a Turing Machine Work?

A Turing Machine consists of:

1. **An infinite tape**: This tape acts like a memory and is divided into cells, each holding a symbol (like 0, 1, or a blank space).
2. **A tape head**: This moves left or right on the tape, reading and writing symbols.
3. **A set of states**: The machine has a finite set of states, one of which is the starting state.
4. **A transition function**: Based on the current state and the symbol under the tape head, the machine:
   1. Writes a new symbol on the tape
   2. Moves left or right
   3. Changes to a new state
5. **A halting condition**: The machine stops when it reaches a special "halt" state.



0 1 0 0 1 1 1 0 1
q_0
1/0,L
q_1
1/0,L
0/1,·
0/0,L
q_0
0/0 ,·
q_H
q_2
1/1,R



Current State	Read Symbol	Write Symbol	Move	Next State
Q_0	1	0	Left	Q_1
Q_1	0	1	Same	Q_2
Q_2	1	1	Right	Q_0
Q_0	0	0	Halt	Q_h

Table of Instructions

**halting stage is important, as the program will continuously loop and will never reach end state.*

Why Is It Important?

The Turing Machine is important because:

- **It defines computability** – If a problem can be solved by a Turing Machine, it is **computable**.
- **It's the foundation of modern computers** – While real computers are more complex, they follow the same basic principles.
- **It helps distinguish between solvable and unsolvable problems** – Some problems (like the **Halting Problem**) cannot be solved by any Turing Machine.

A Simple Turing Machine

Imagine a Turing Machine that takes a string of 1s (like "111") and replaces all of them with 0s ("000"), then stops.

1. **Initial state:** The tape has 111 and the head starts at the first 1.
2. **Step 1:** Reads 1, writes 0, moves right.
3. **Step 2:** Reads 1, writes 0, moves right.
4. **Step 3:** Reads 1, writes 0, moves right.
5. **Halts:** No more 1s left, so the machine stops.

- The final result on the tape is 000, meaning the machine successfully replaced all 1s with 0s.

Turing Machines and Real Computers

While a Turing Machine is a simple, abstract model, modern computers work on the same principles. However, real computers:

- Have **finite memory**, unlike a Turing Machine's infinite tape.
- Use **complex instructions** instead of simple state transitions.
- Are **much faster and more practical** than a theoretical Turing Machine.
- Despite these differences, Turing Machines help computer scientists **prove what is possible to compute** and what isn't.

Turing Machine Representation

$$(Q, \Sigma, \Gamma, \delta, q_0, F)$$

where:

- Q = Set of **states**
- Σ = Input **alphabet** (symbols the machine reads)
- Γ = Tape **alphabet** (includes input symbols + blank `□`)
- δ = **Transition function** (rules for moving, writing, and changing states)
- q_0 = **Initial state**
- F = **Final (halting) state**

```
... □ □ 1 1 1 □ □ ...
        ↑
      (q0)
```

Example: A Turing Machine That Converts All 1s to 0s

Let's define a Turing Machine that **replaces all 1s with 0s** and then stops.

Tape Representation

- The tape starts with 111 and infinite blank spaces (□):

State Transition Table

Current State	Read Symbol	Write Symbol	Move	Next State
q_0	1	0	R	q_0
q_0	1	0	R	q_0
q_0	1	0	R	q_0
q_0	□	□	L	q_f

- If the machine reads `1`, it **writes** `0`, moves **right**, and stays in q_0.
- If it reads a blank (`□`), it **stops** at the final state q_f.

```
... □ □ 1 1 1 □ □ ...
     ↑
    (q0)
```

Step 1: Initial State (q0)

```
... □ □ 0 1 1 □ □ ...
         ↑
        (q0)
```

Step 2: Reads 1, writes 0, moves right.

```
... □ □ 0 0 1 □ □ ...
          ↑
         (q0)
```

Step 3: Reads 1, writes 0, moves right.

```
... □ □ 0 0 0 □ □ ...
             ↑
            (q0)
```

Step 4: Reads □ (blank), moves left, and enters final state q_f

```
(q0) --[1 → 0, R]--> (q0) --[1 → 0, R]--> (q0) --[1 → 0, R]--> (q0)
     \_______________________________________________/
                    |
                    ↓
          (q0) --[□ → □, L]--> (qf) [HALT]
```

State Diagram