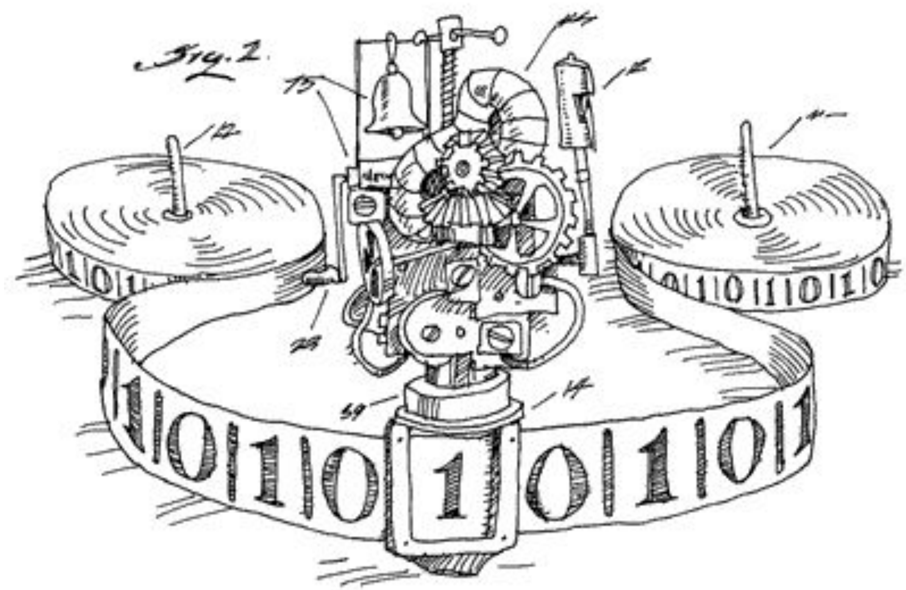
EECS 376: Foundations of Computer Science

Turing Machines



Review: Representing problems

A **decision problem** can be thought of as

$$f : \Sigma^* \rightarrow \{\text{No, Yes}\}$$

or equivalently as a **language**

$$L \subseteq \Sigma^*$$

$$L = \{x \in \Sigma^* : f(x) = \text{Yes}\} \qquad f(x) = \begin{cases} \text{Yes if } x \in L \\ \text{No if } x \notin L \end{cases}$$

E.g.: $L_{\text{PALINDROME}} = \{x \in \Sigma^* : x \text{ is a palindrome}\}$

Review: Representing problems

Let M be a DFA, using alphabet Σ.

M accepts some strings in Σ^* and rejects the rest.

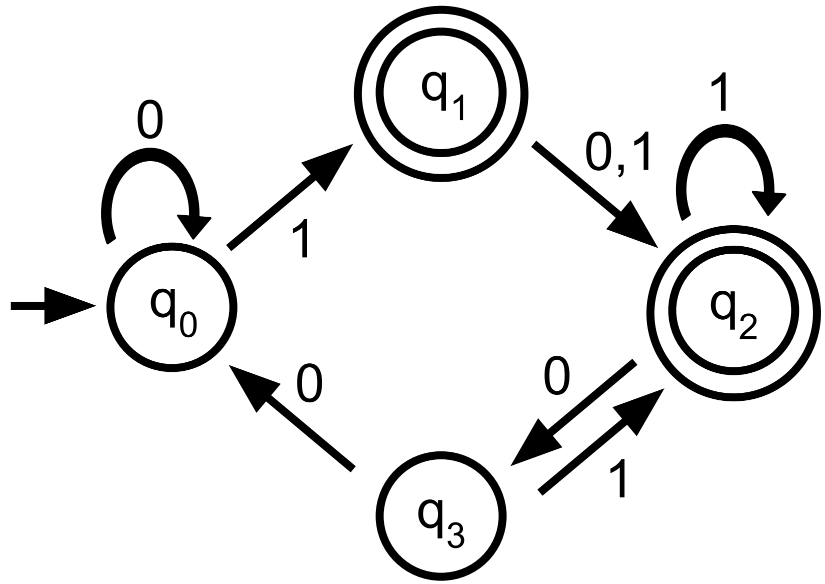
Definition: $L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$

Called the "language decided by M".

If L is a language,

we say M decides L if $L(M) = L$.

Review: DFAs



Recall: There are simple languages like L = $\{x^k y^k : k \geq 0\}$ that no DFA can decide!

Last time I told you that your laptop is more powerful than a DFA.

That was technically a lie… because your laptop has finite memory. Your laptop also cannot decide the language $L = \{x^k y^k : k \geq 0\}$.

A DFA models a computer with a fixed amount of memory (think of the states as "memory")

In our definition of a "computer", we don't want to artificially constrain ourselves to a fixed amount of memory.

Instead, we want our definition of a "computer" to model the situation where you can always buy more memory if you need it.

That's where Turing Machines come in.

But first, some history…

Some History: Hilbert's 10^{th} Problem (1900)

Consider a multivariate polynomial with integer coefficients set equal to 0

E.g. $4x^2y^3 - 2x^4z^5 + x^8 = 0$.

"*Devise a process according to which it can be determined in a finite number of operations* whether it has an integer solution."

This is asking for an **algorithm**.

Entscheidungsproblem (1928)

Given a statement in first-order logic.

Devise an "*effectively calculable procedure*" that determines if it is valid.

Again, this is asking for an **algorithm**.

Mathematicians began to think hard about a formal definition for "**algorithm**".

Gödel (1934):

Discusses some ideas for mathematical definitions of computation procedures, but isn't confident what's a good definition.

Church (1936):

Invents lambda calculus, essentially claims it should be the definition of "algorithm".

Gödel, Post (1936):

Arguments that Church's claim is in not justified.

Meanwhile… in New Jersey… a certain British grad student, unaware of all these debates…

Alan Turing (1936, age 22):

Describes a new model of computation, now known as the **Turing Machine.** TM

Gödel, Kleene, and even Church:

"Um, he nailed it. Game over, 'algorithm' defined."

1937: Turing proves TM's ≡ lambda calculus

Church–Turing Thesis

“Any natural notion of being ‘algorithmically computable’ is captured by Turing Machines.”

This is not a theorem.

Is it… …*a hypothesis*?

…*a definition*?

…*a philosophical statement*?

Well, in any case, everyone seems to believe it.

Anything that can be computed by Python, C++, a quantum computer, LaTex, pseudocode, etc. can be computed by a Turing Machine.

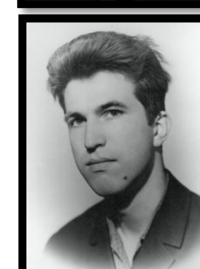
Entscheidungsproblem: Devise an **algorithm** (Turing Machine) that, given a statement in first-order logic, determines if it's valid.

There is no such algorithm! (1936)

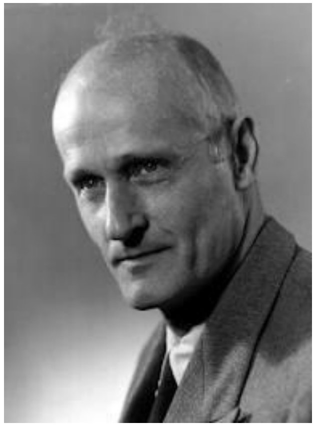
Hilbert's 10th Problem: Devise an **algorithm** (Turing Machine) that, given a multivariate polynomial with integer coefficients, determines if it has an integer root.

There is no such algorithm! (1970)



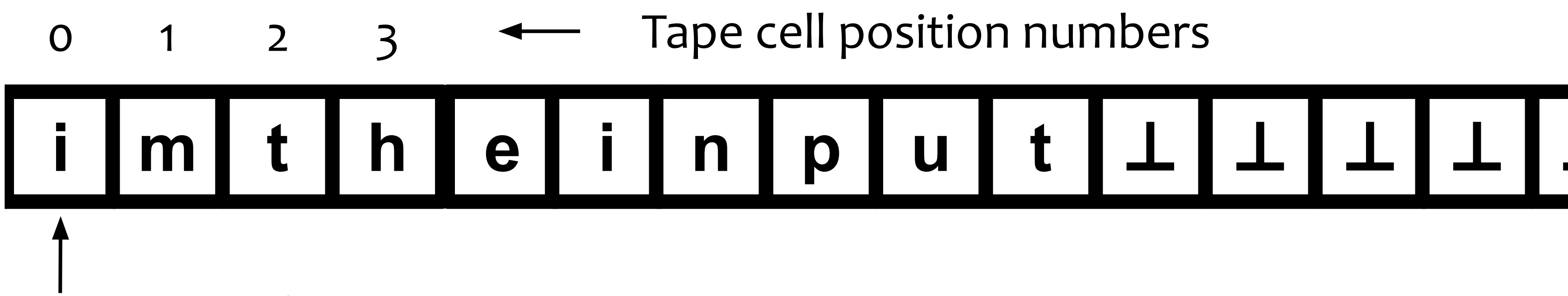



"An algorithm is a finite answer to an infinite number of questions"



What is a Turing Machine?

Memory of a Turing Machine: an infinite "tape" of cells:



The input from Σ^* is written starting at cell 0.

All other cells contain $\perp$ (blank).

In general, cells contain symbols from a tape alphabet Γ,
which must contain Σ, $\perp$, and may have other symbols.

There's a tape pointer ("head"), initially at position 0.

What is a Turing Machine?

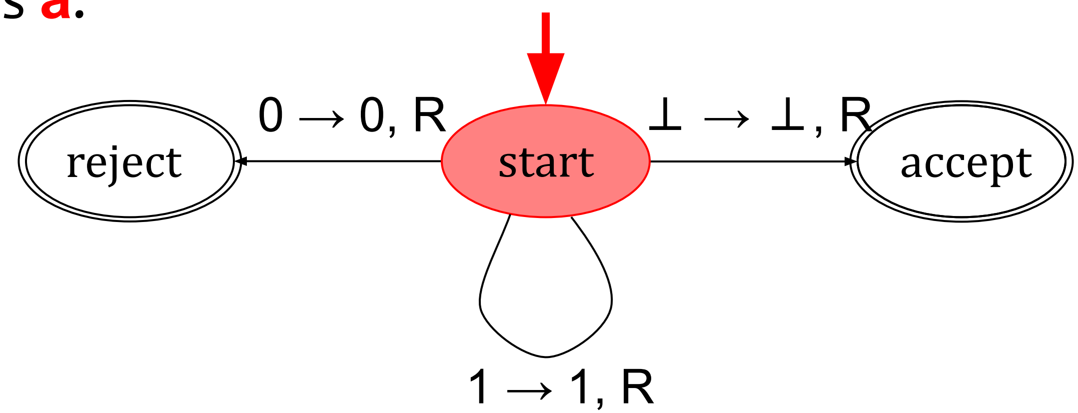
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

Specifically: **"a → b, R"** means:

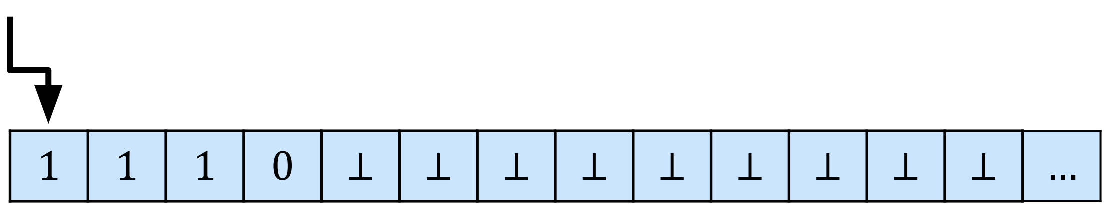
if the current tape cell is **a**:

1. write **b**
2. move **right**



There are <u>*two*</u> special "termination" states: ***accept*** and ***reject***.

INFINITE TAPE



What is a Turing Machine?

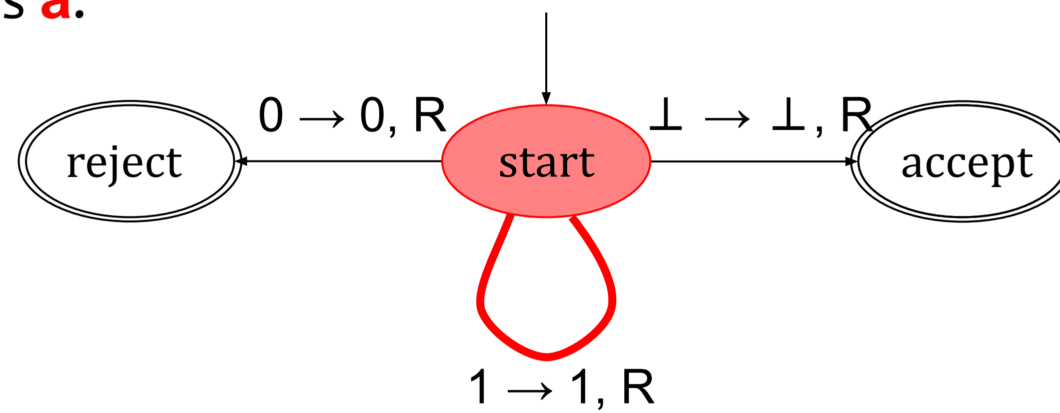
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

Specifically: **"a → b, R"** means:

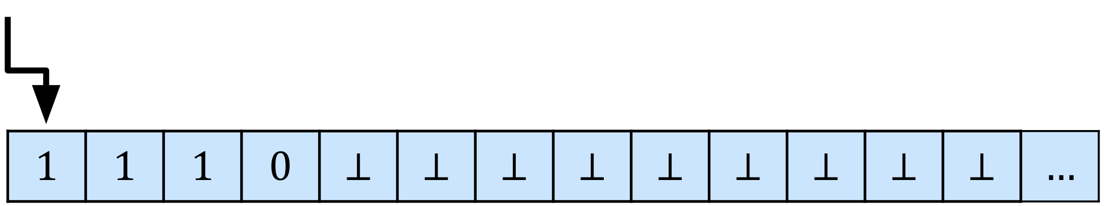
if the current tape cell is **a**:

1. write **b**
2. move **right**



There are <u>*two*</u> special "termination" states: ***accept*** and ***reject***.

INFINITE TAPE



What is a Turing Machine?

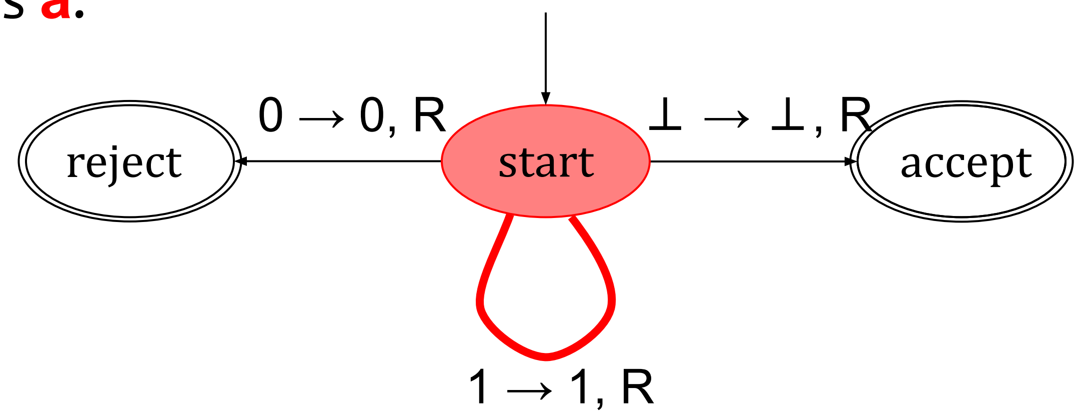
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

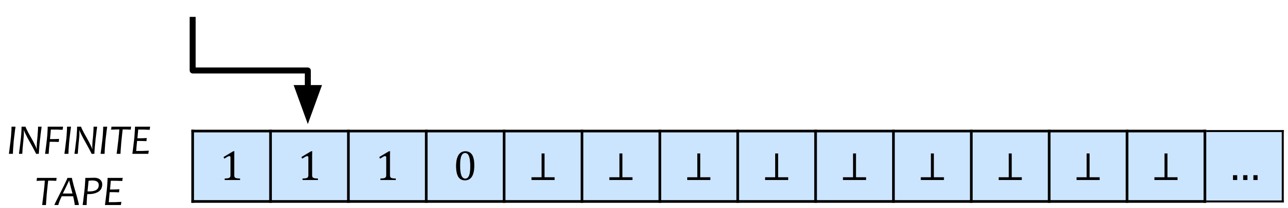
Specifically: **“a → b, R”** means:

if the current tape cell is **a**:

1. write **b**
2. move **right**



There are <u>*two*</u> special “termination” states: ***accept*** and ***reject***.



What is a Turing Machine?

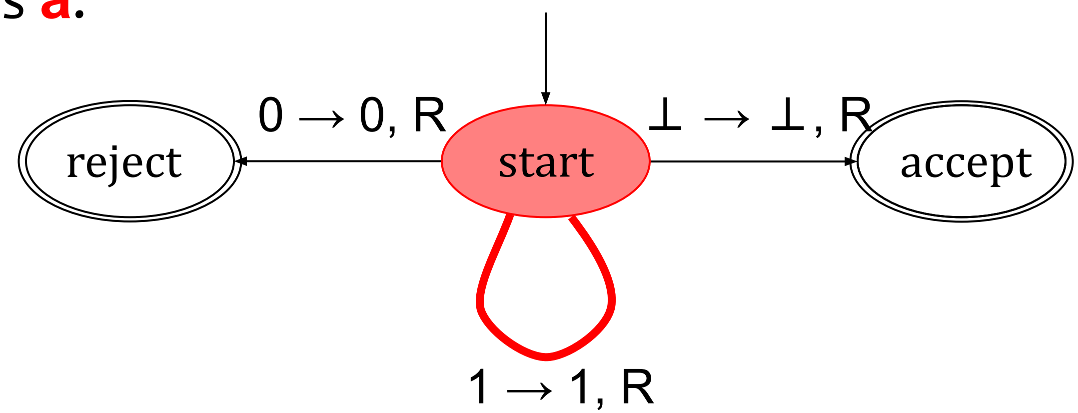
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

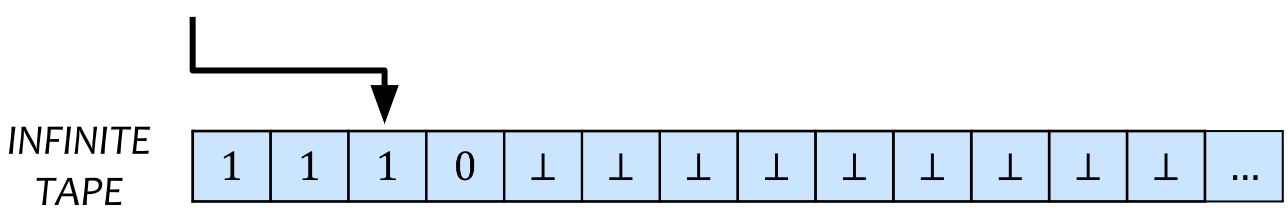
Specifically: **"a → b, R"** means:

if the current tape cell is **a**:

1. write **b**
2. move **right**



There are <u>*two*</u> special "termination" states: ***accept*** and ***reject***.



What is a Turing Machine?

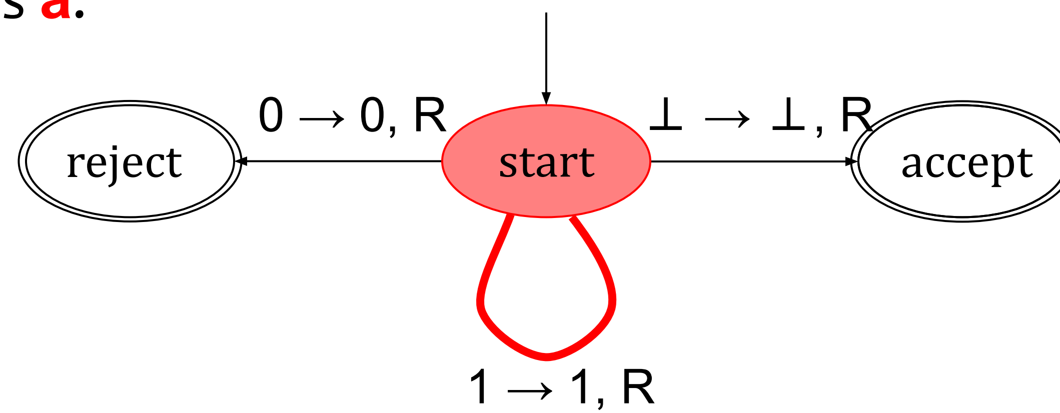
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

Specifically: **"a → b, R"** means:

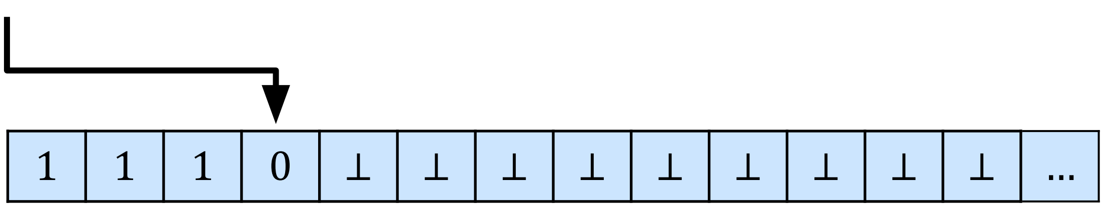
if the current tape cell is **a**:

1. write **b**
2. move **right**



There are <u>*two*</u> special "termination" states: ***accept*** and ***reject***.

INFINITE TAPE



What is a Turing Machine?

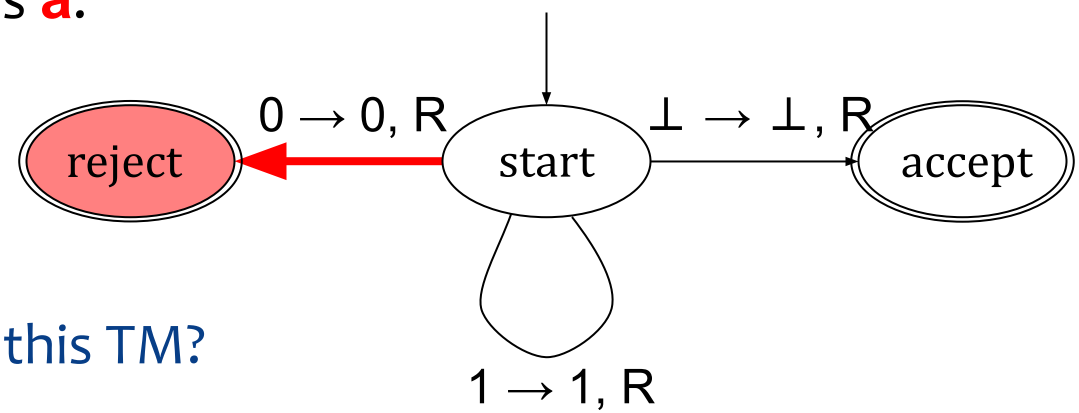
Like a DFA, except additionally specifies:

- what we *write*
- whether the head moves *left* (L) or *right* (R)

Specifically: **"a → b, R"** means:

if the current tape cell is **a**:

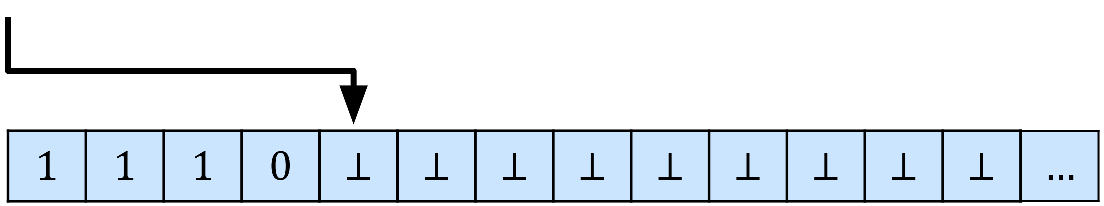
1. write **b**
2. move **right**



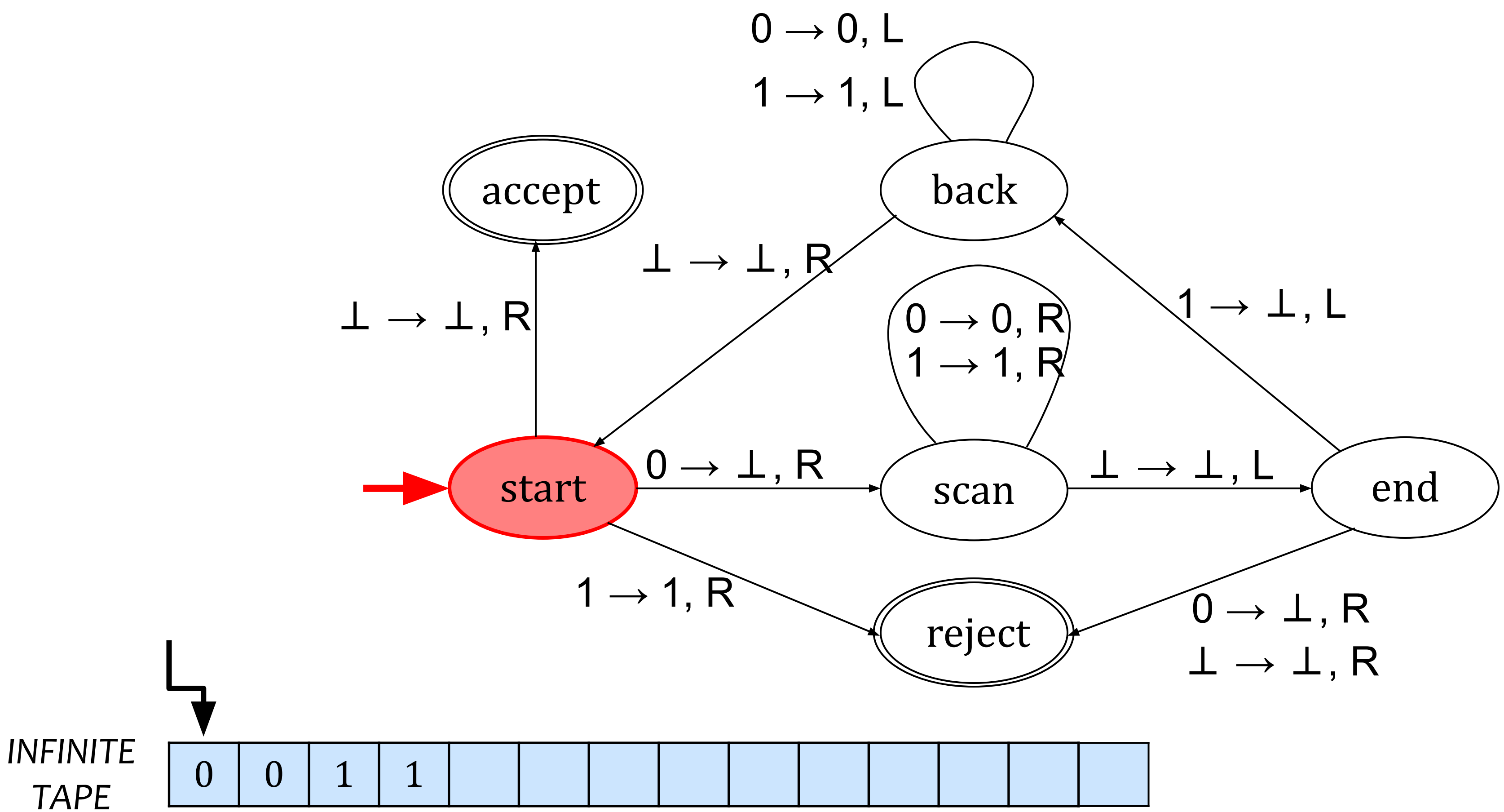
Q: Set of inputs accepted by this TM?

There are <u>*two*</u> special "termination" states: ***accept*** and ***reject***.

INFINITE TAPE

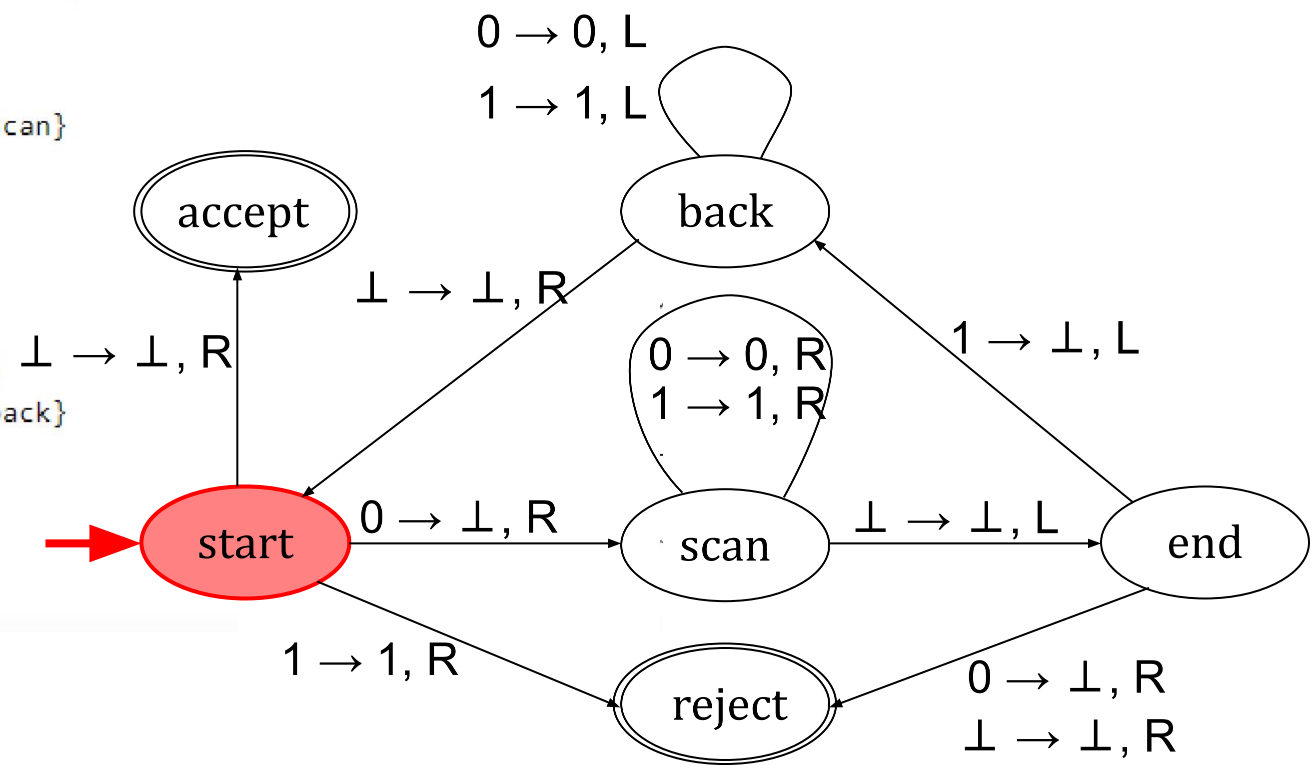


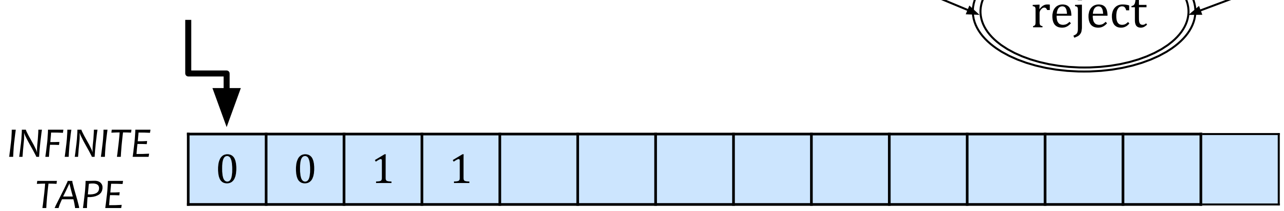
What is a Turing Machine?



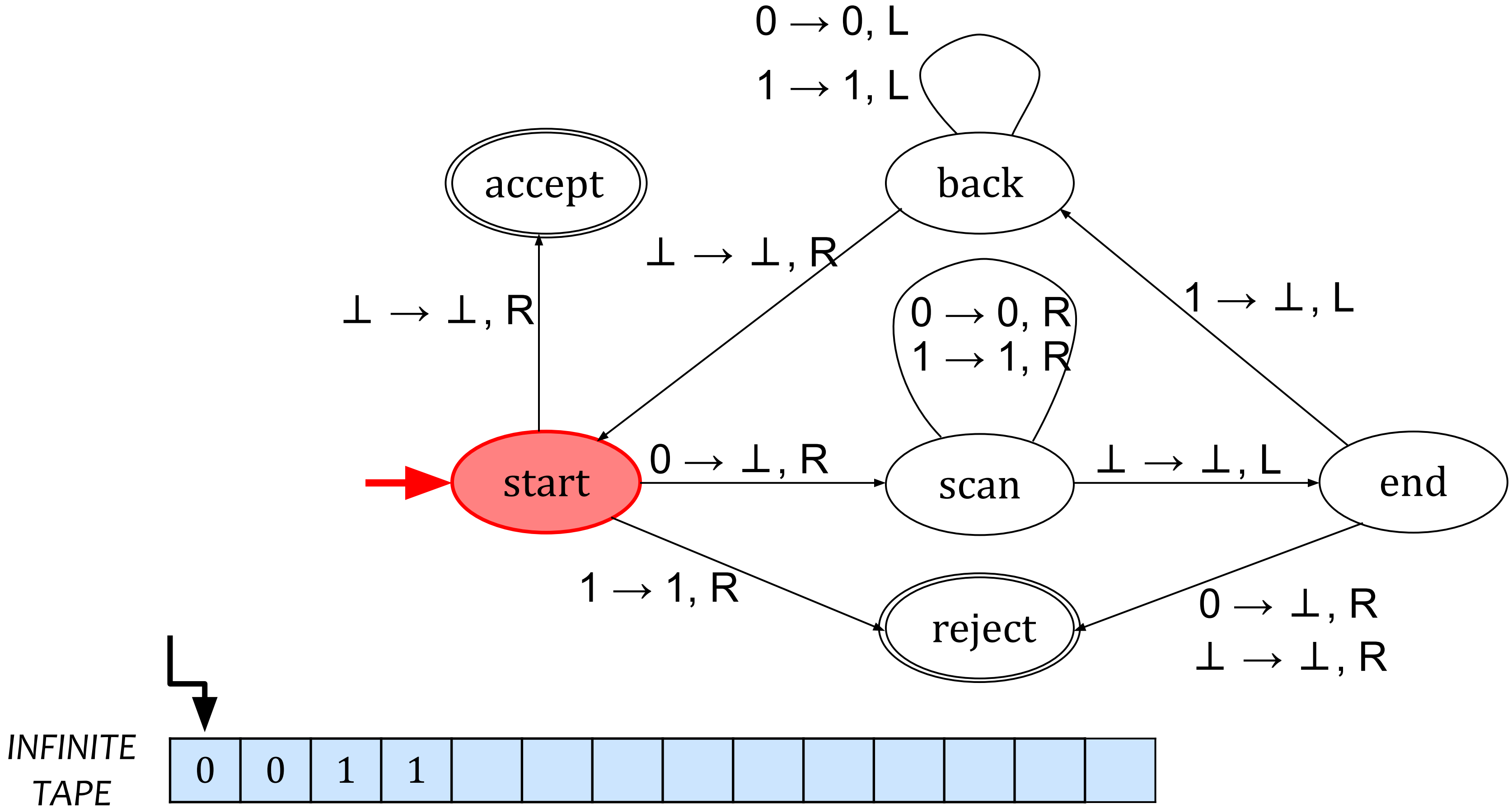
A useful tool: turingmachine.io

```
# Decides the language { 0^n1^n | n ≥ 0 }, that is,
# accepts 0's followed by 1's of the same length.
input: '0011' # try '', '0', '011', '010', '11', '0101'
blank: ' '
# Blank out the first 0 and last 1 on each pass.
# When there are only blanks, all 0's have matched to all 1's.
start state: start
table:
  start:
    0: {write: ' ', R: scan}
    1: {R: reject}
    ' ': {R: accept}
  scan:
    [0,1]: R
    ' ': {L: end}
  end:
    [0,' ']: {R: reject}
    1: {write: ' ', L: back}
  back:
    [0,1]: L
    ' ': {R: start}
  reject:
  accept:
```

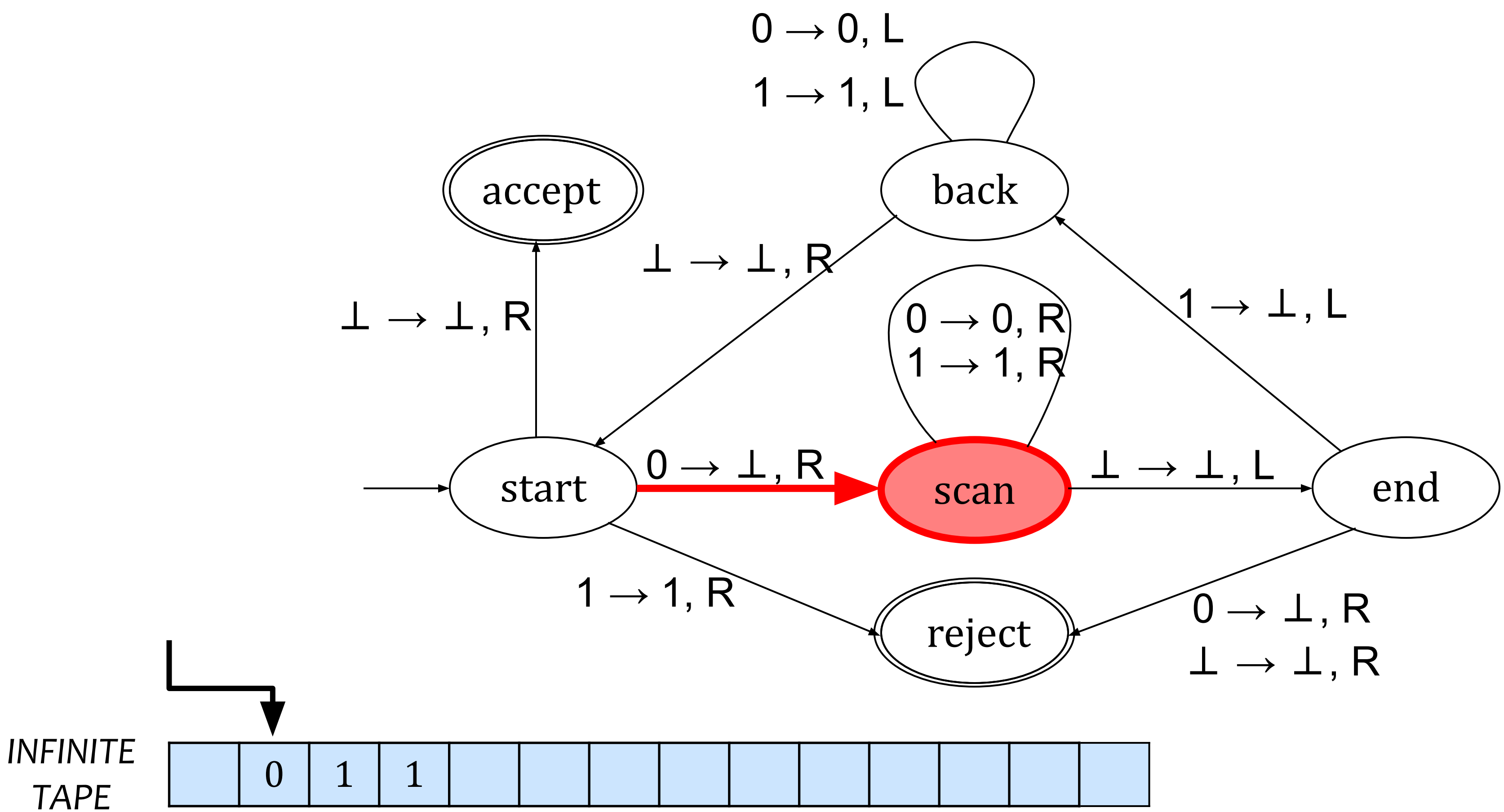




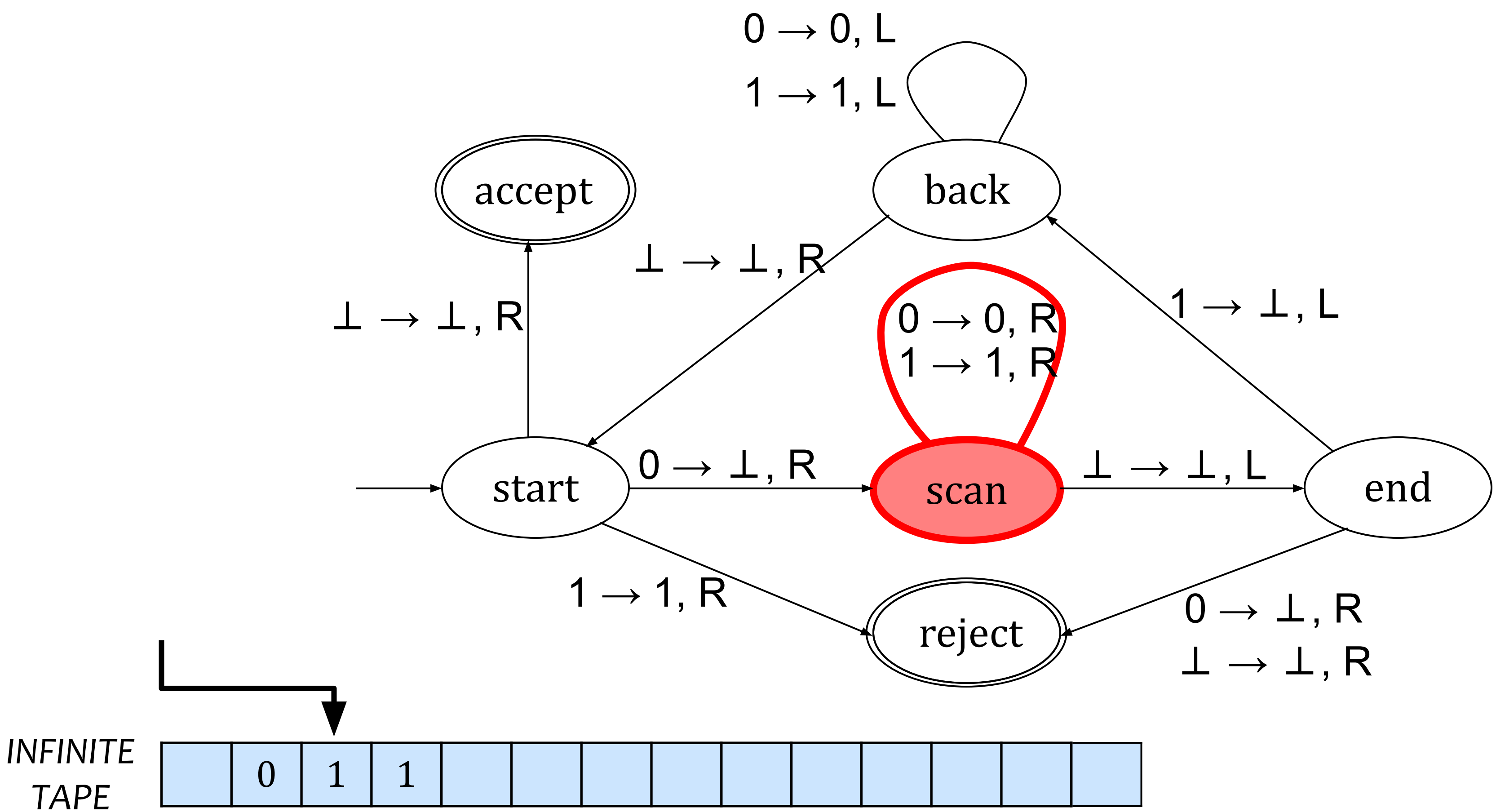
What is a Turing Machine?



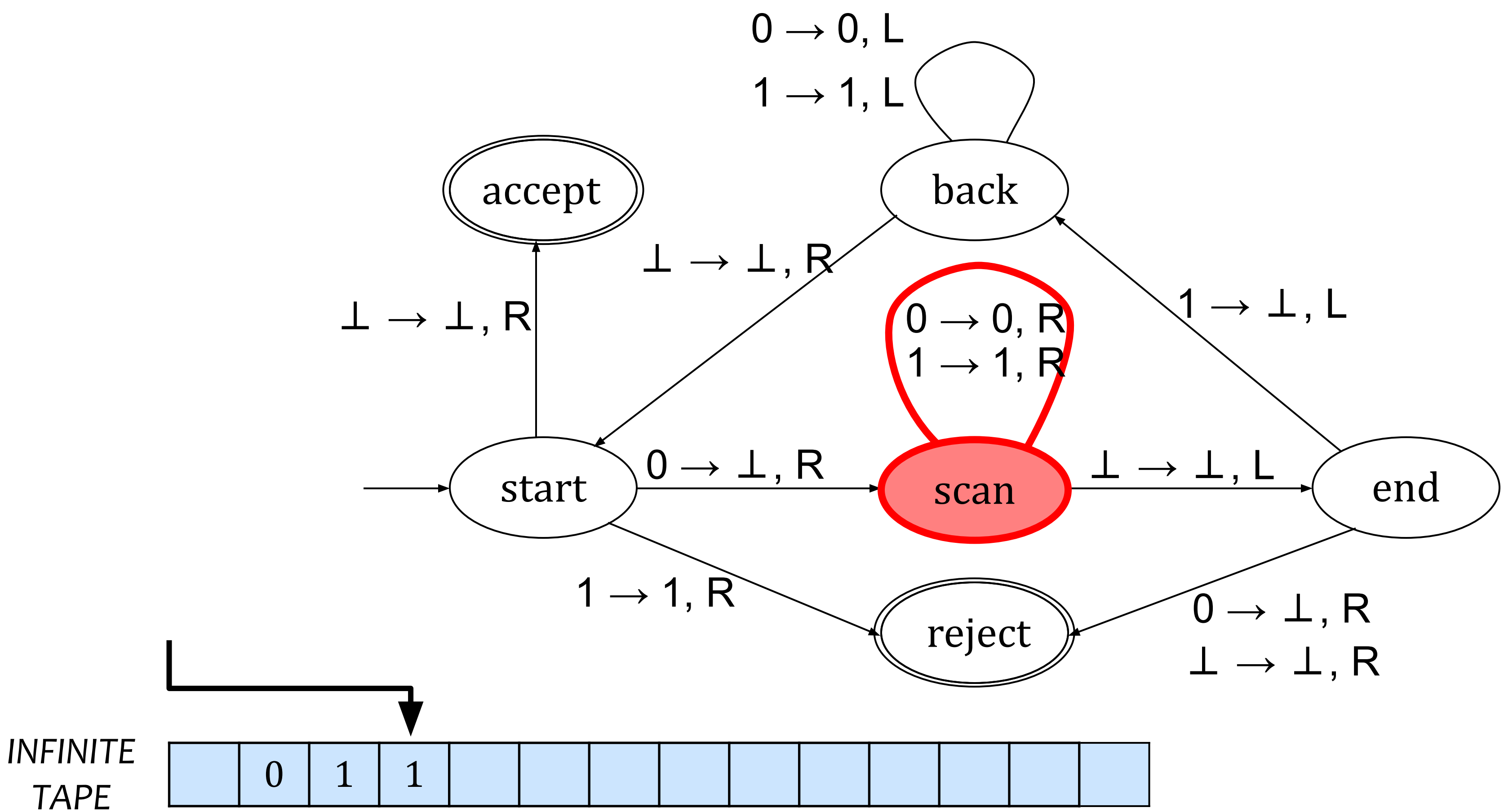
What is a Turing Machine?



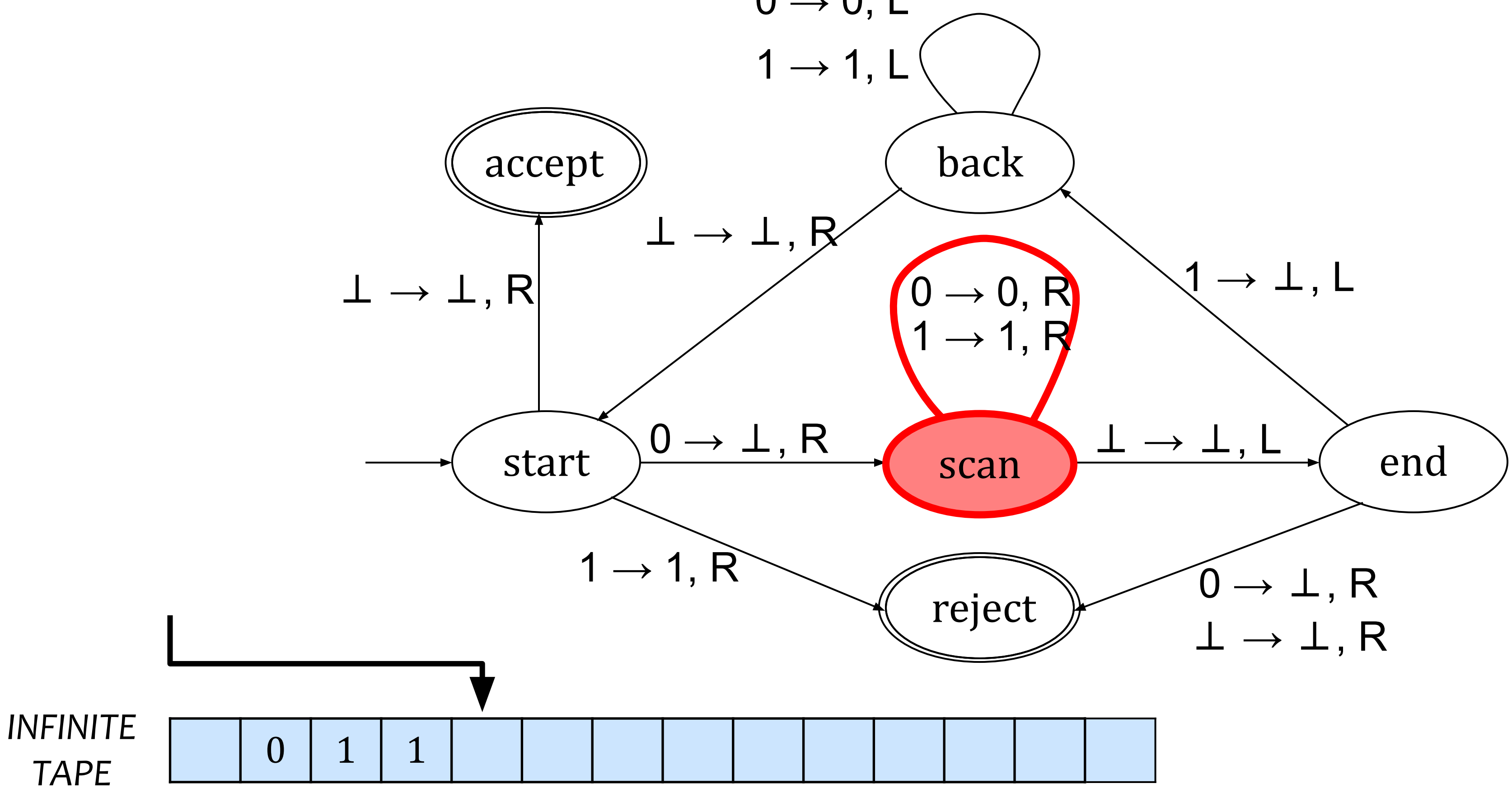
What is a Turing Machine?



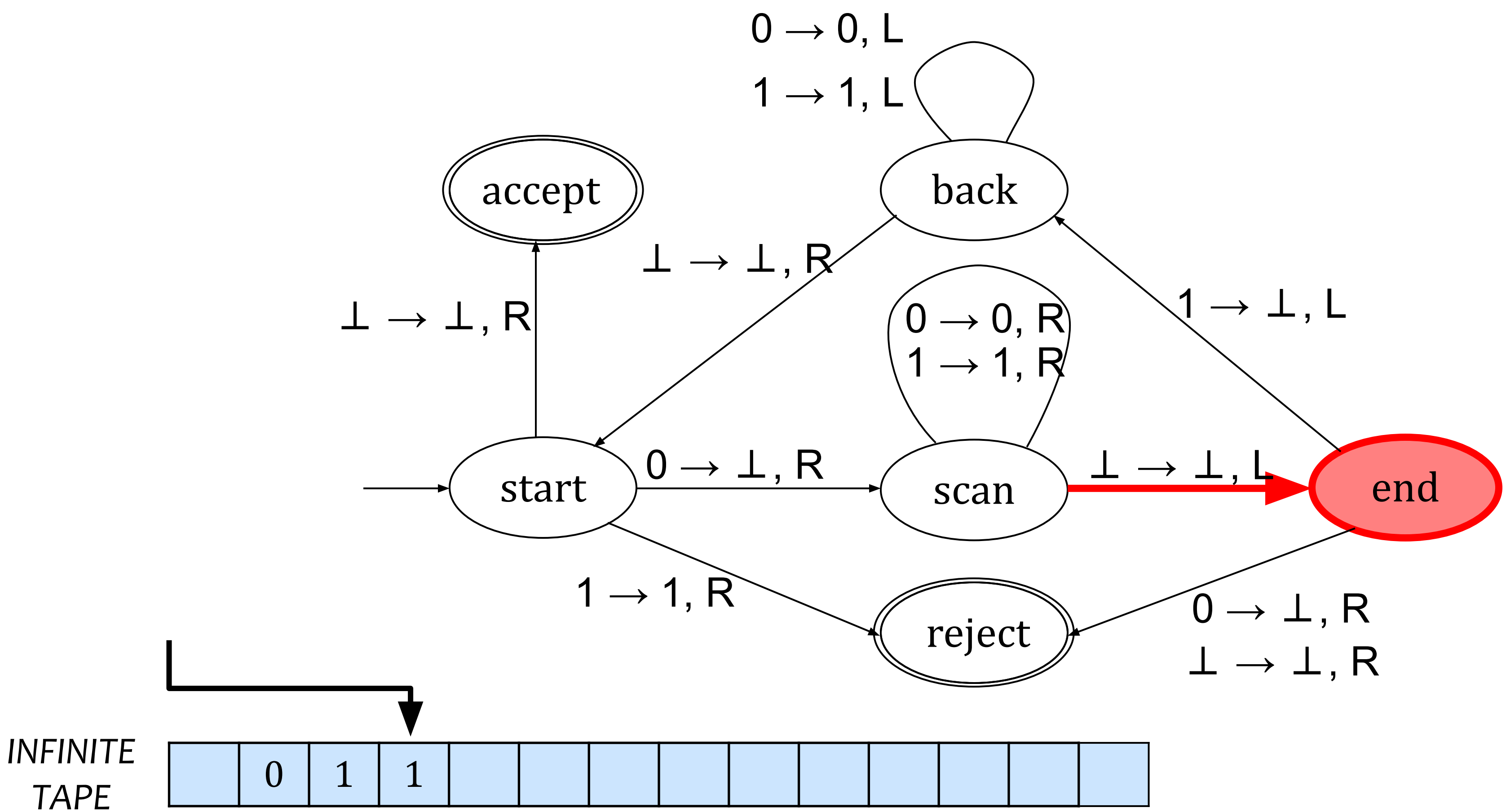
What is a Turing Machine?



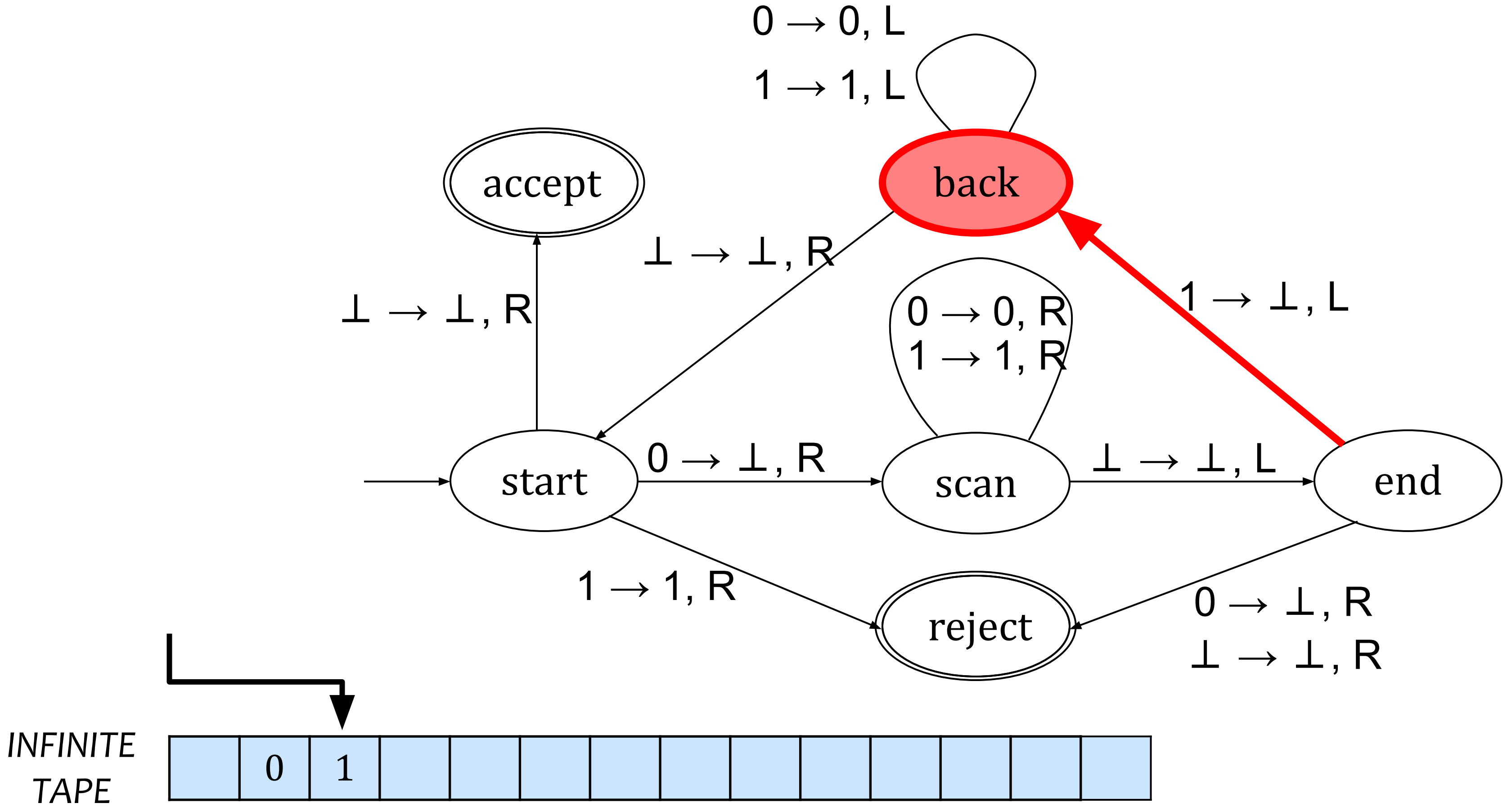
What is a Turing Machine?



What is a Turing Machine?

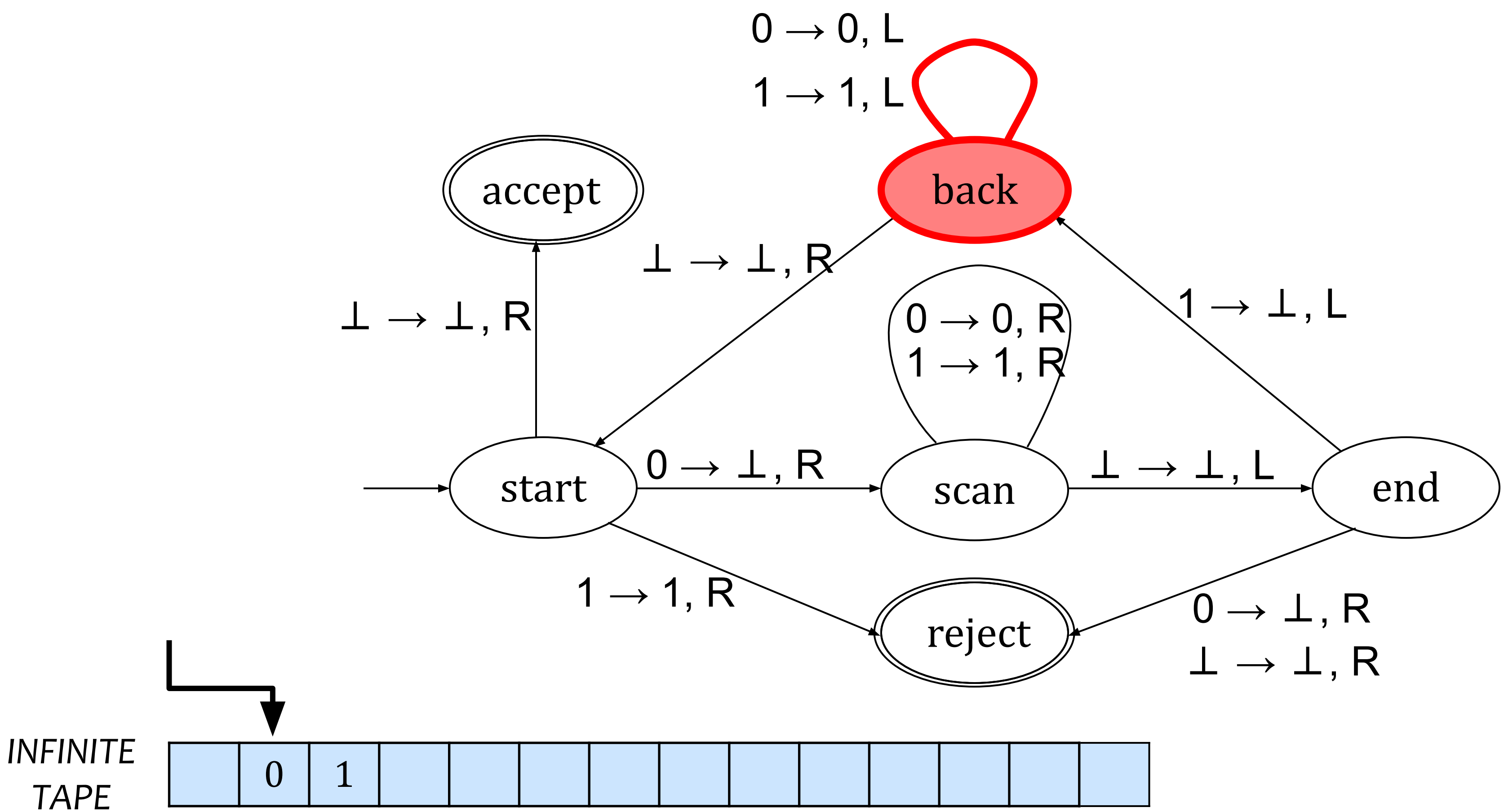


What is a Turing Machine?



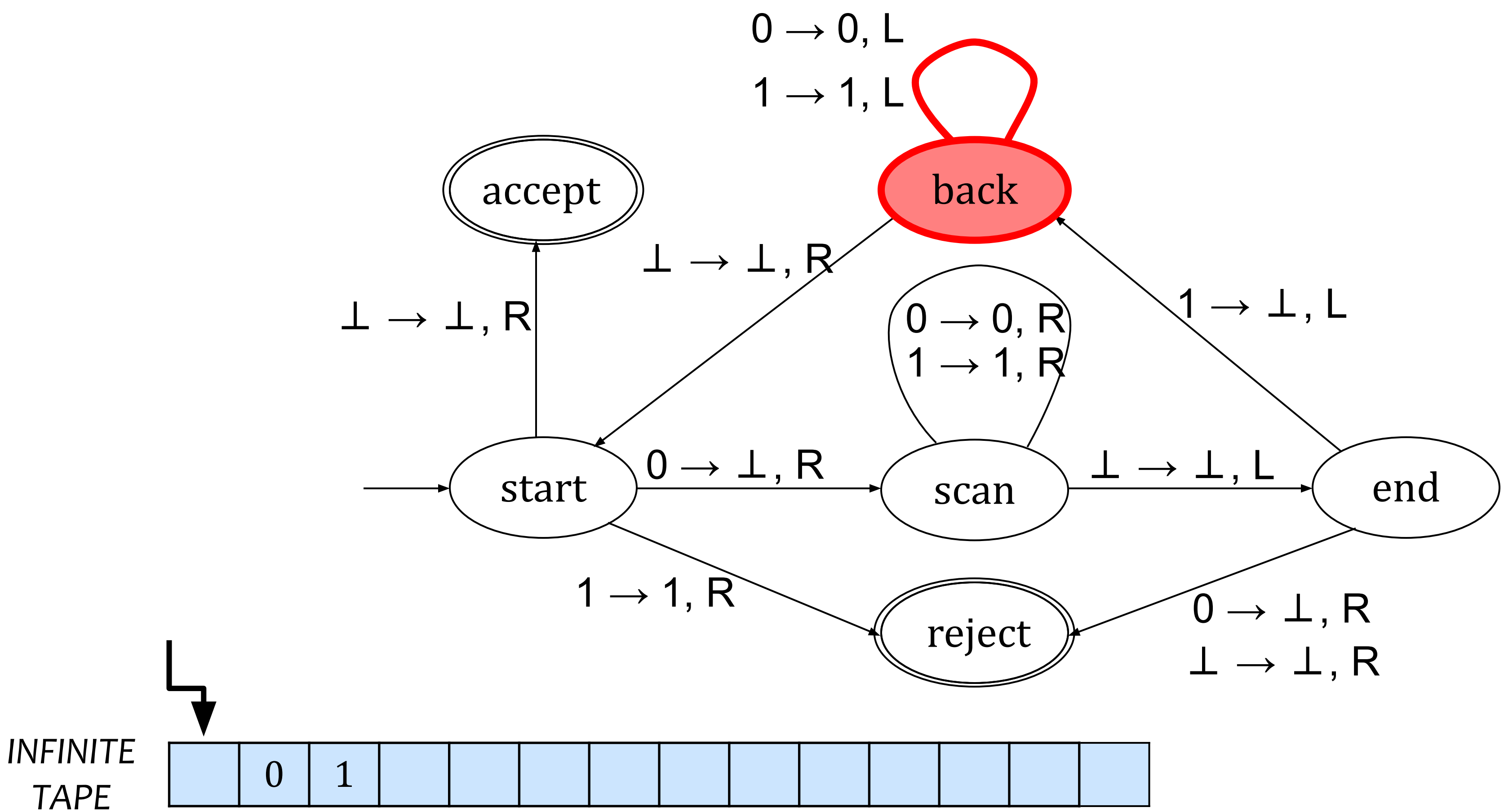
What is a Turing Machine?

TM Example

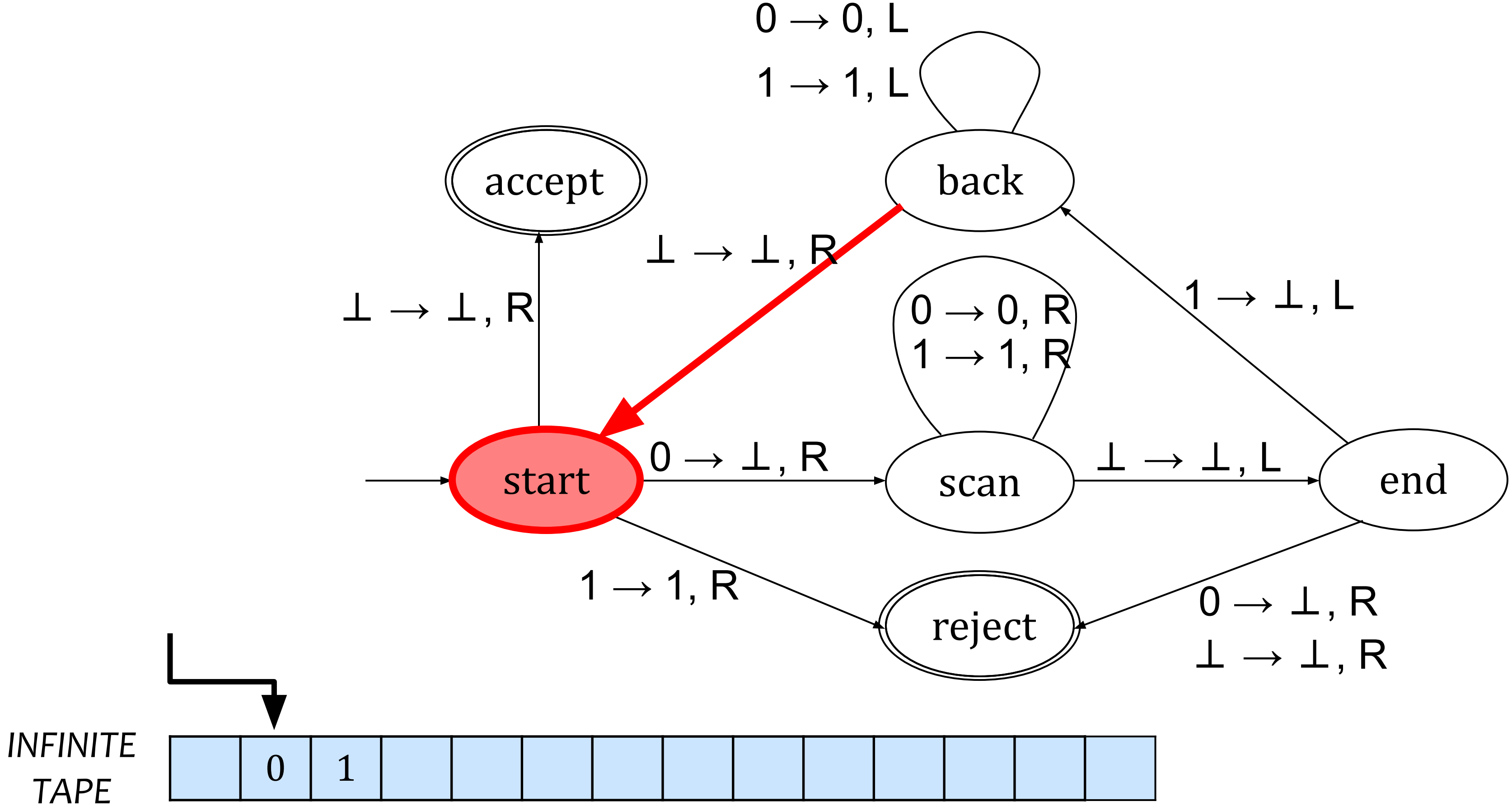


What is a Turing Machine?

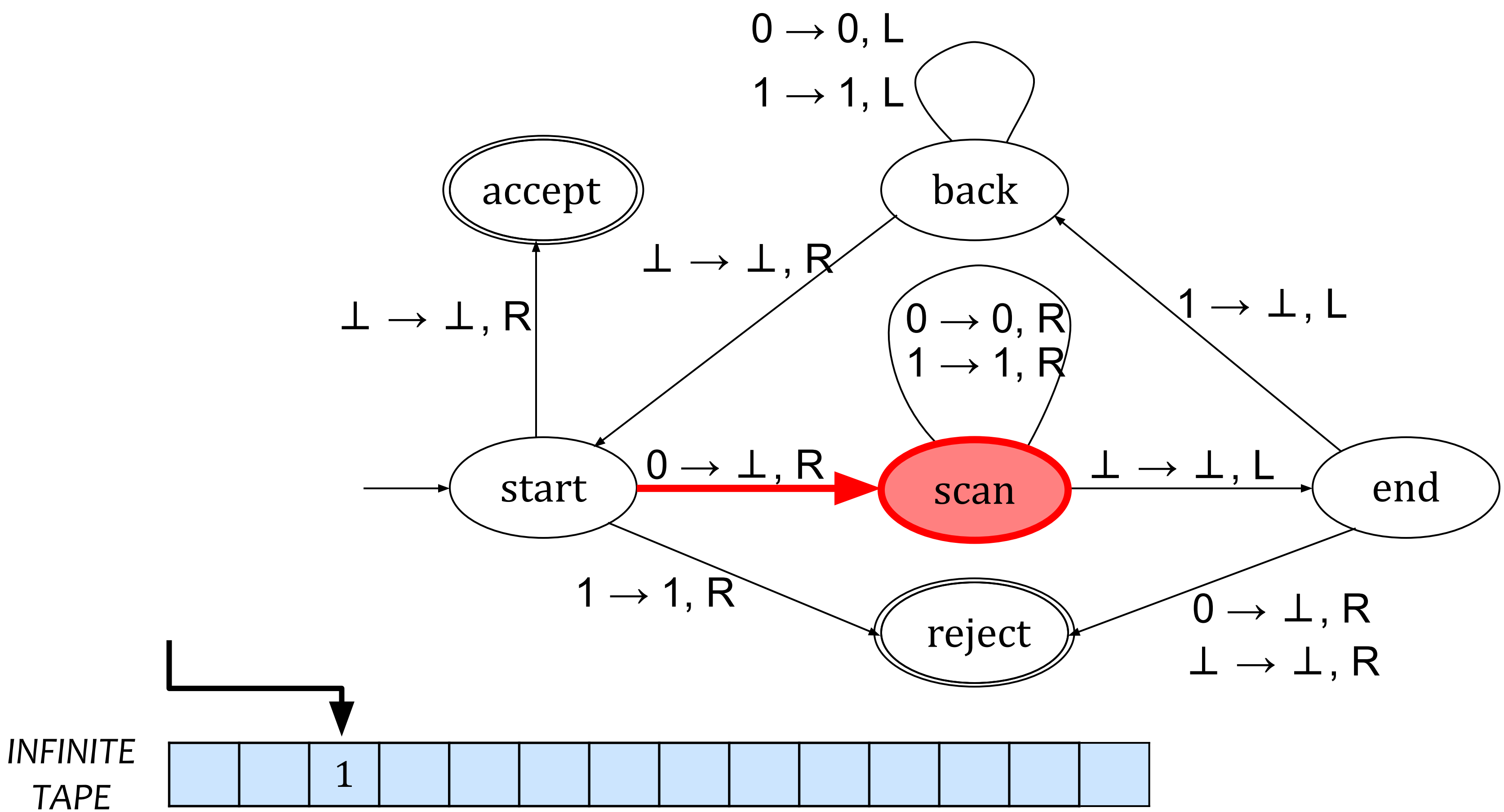
TM Example



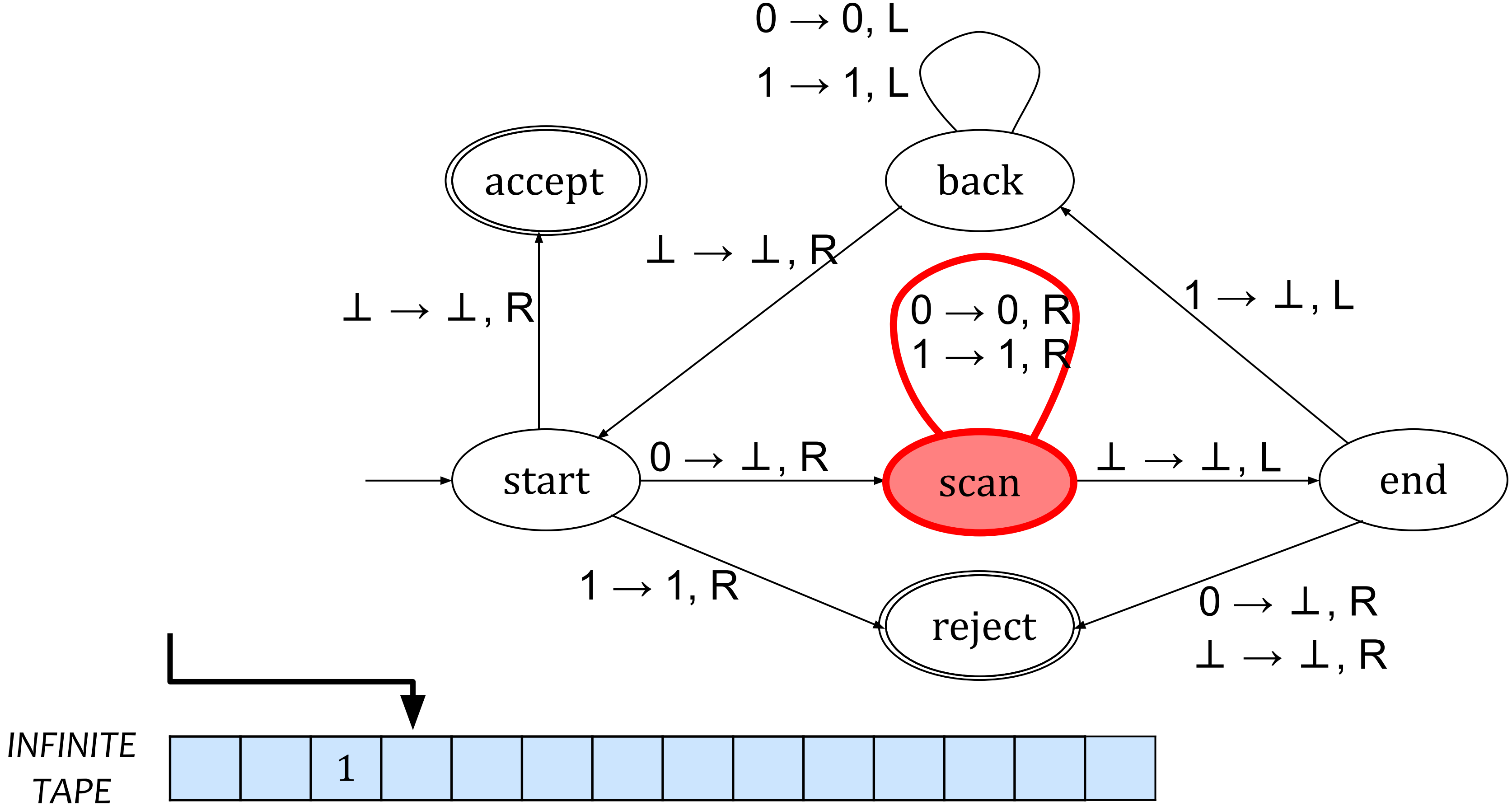
What is a Turing Machine?



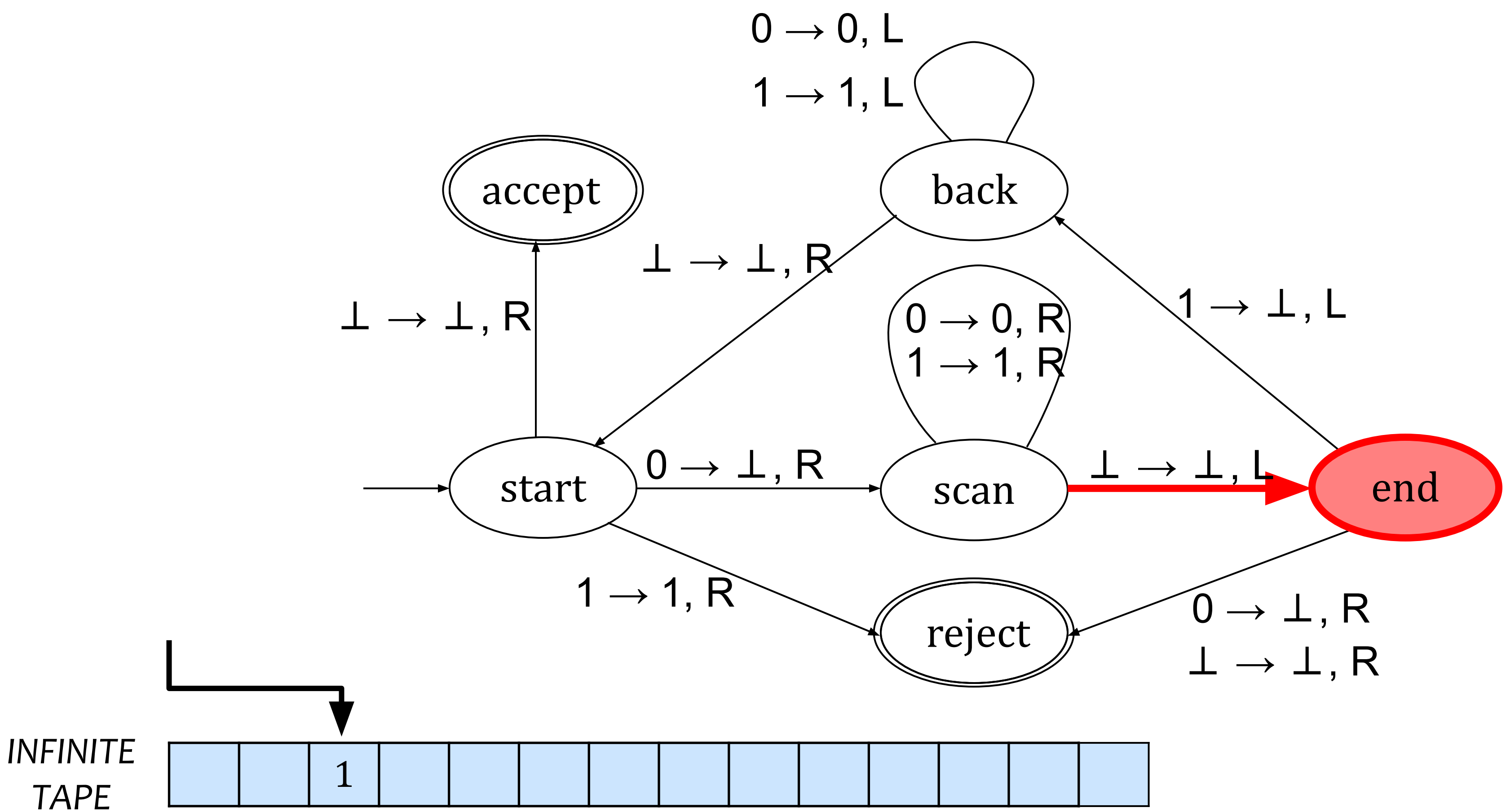
What is a Turing Machine?



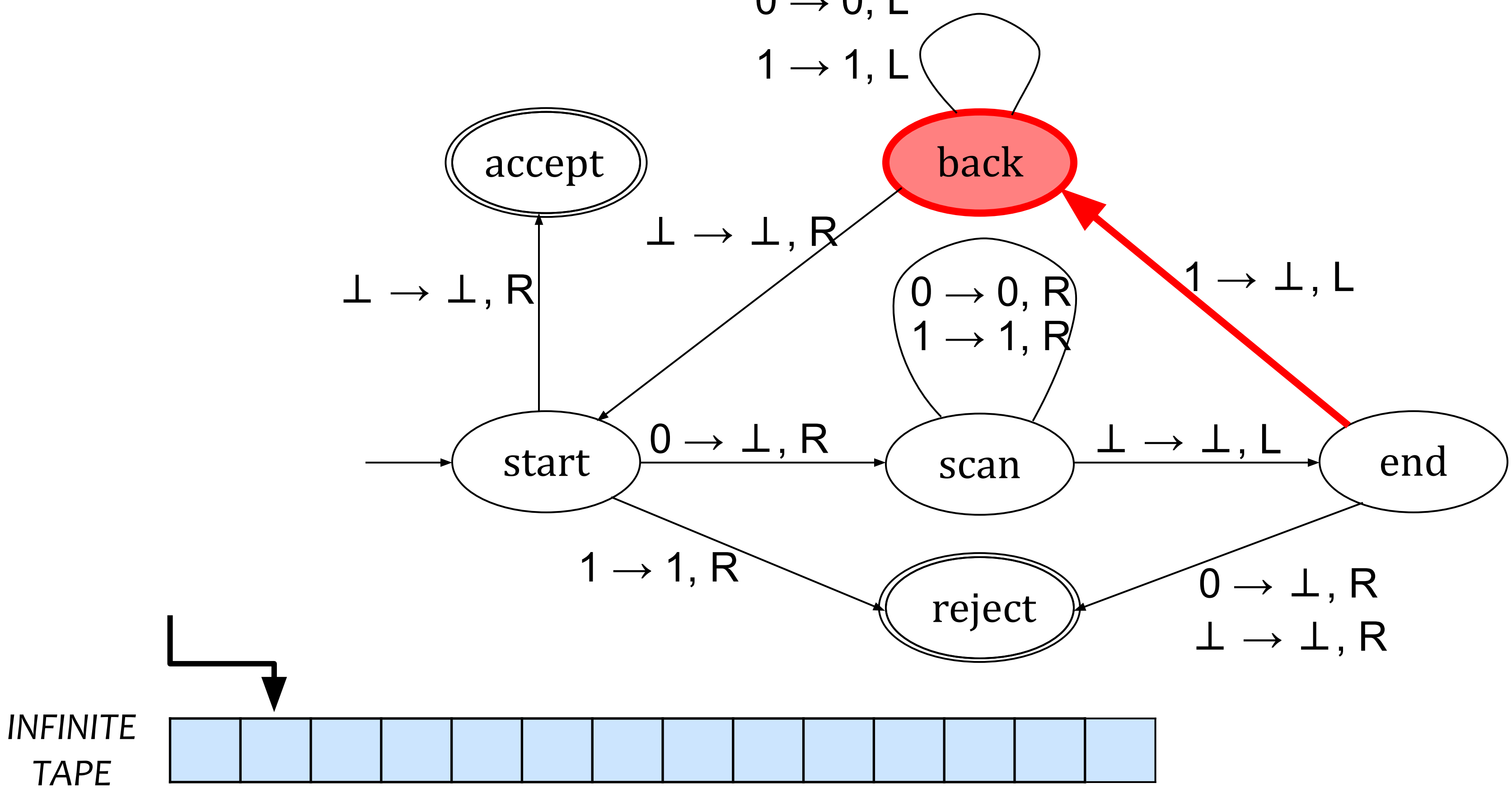
What is a Turing Machine?



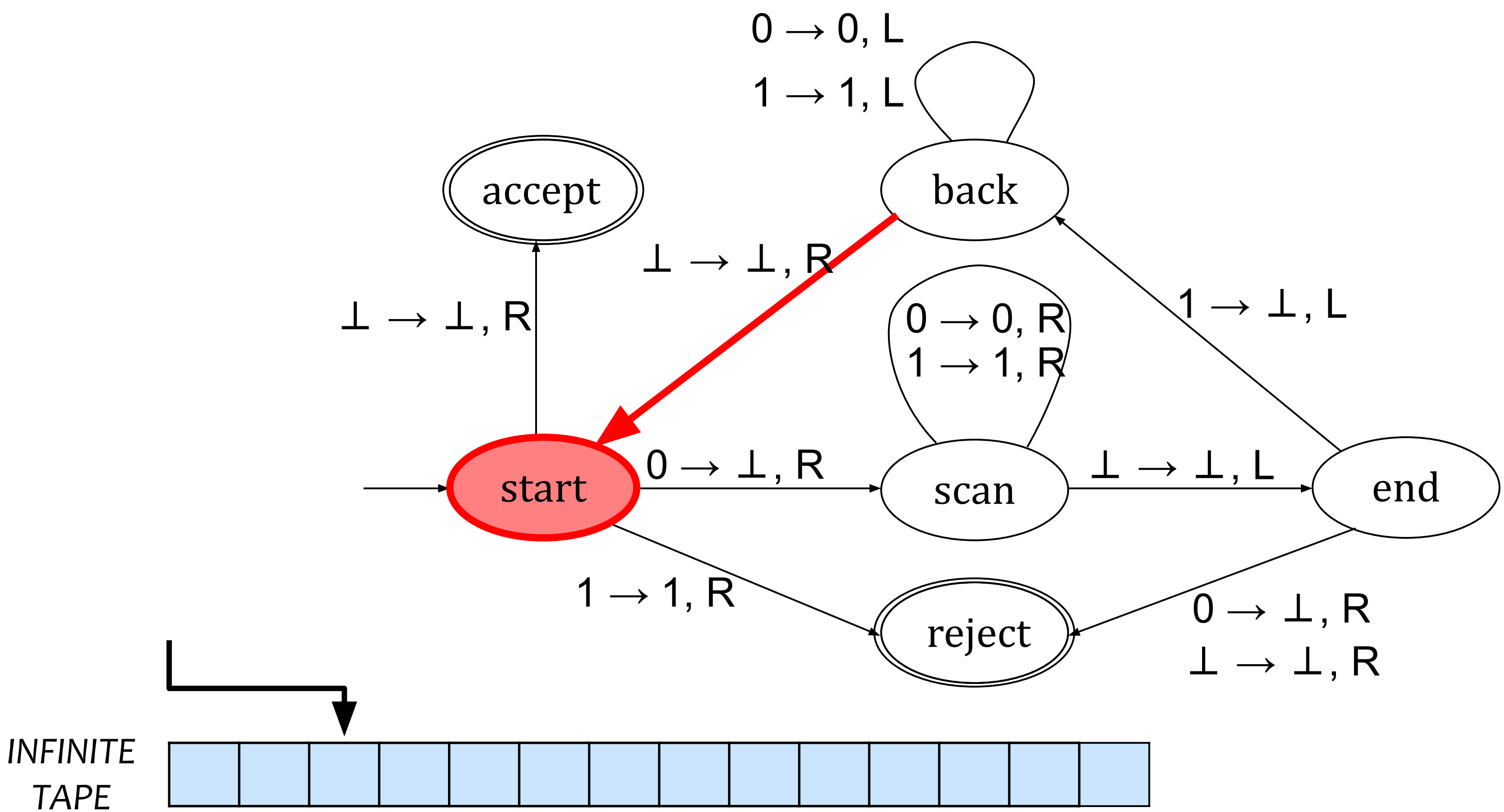
What is a Turing Machine?



What is a Turing Machine?

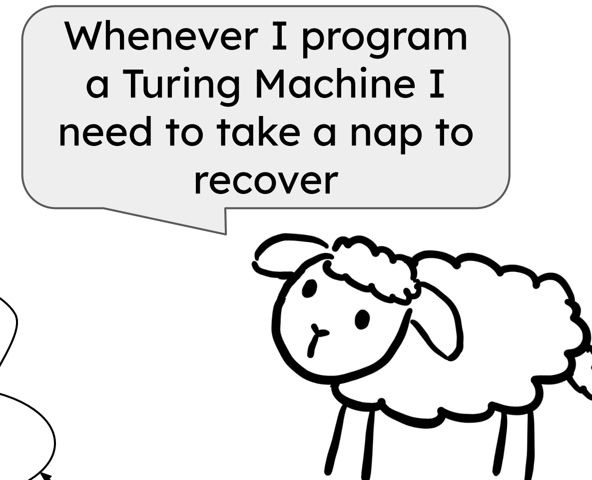


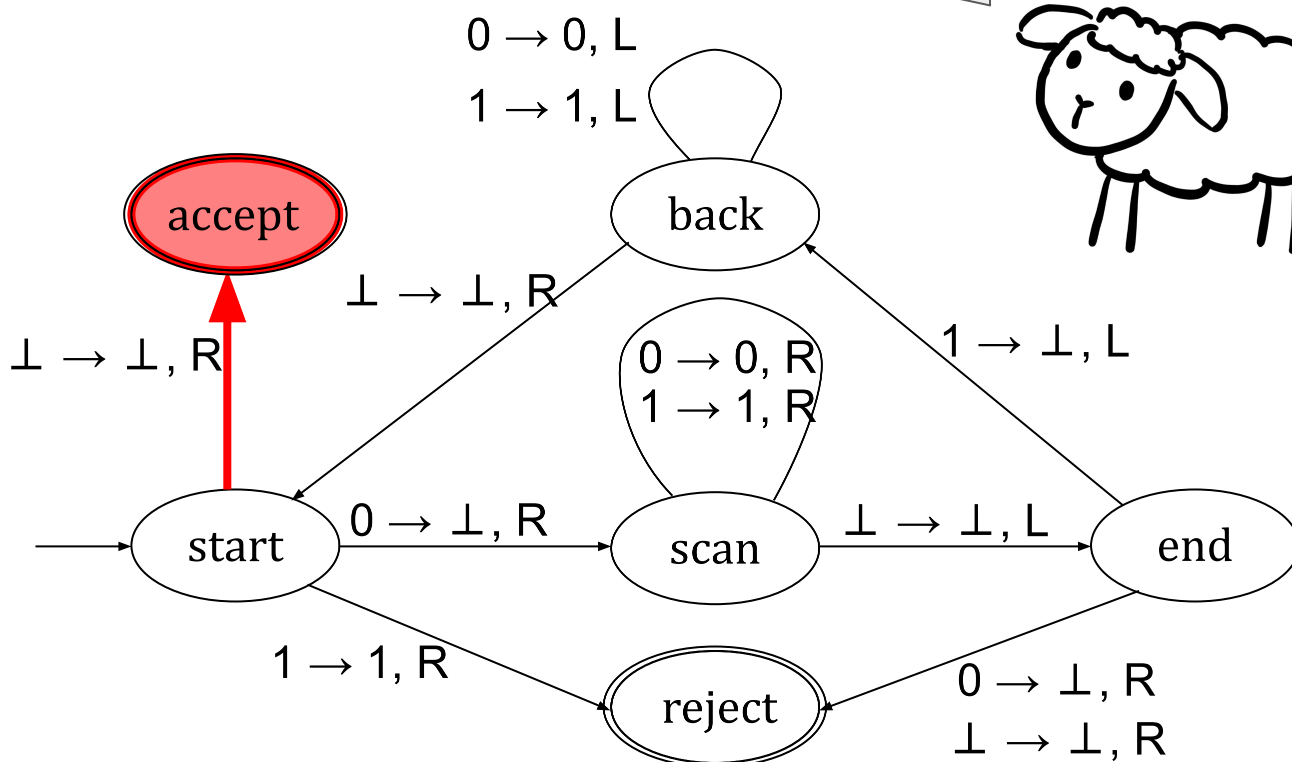
What is a Turing Machine?

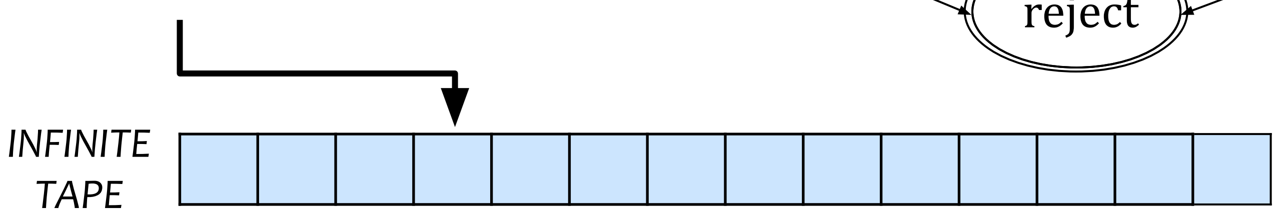


What is a Turing Machine?

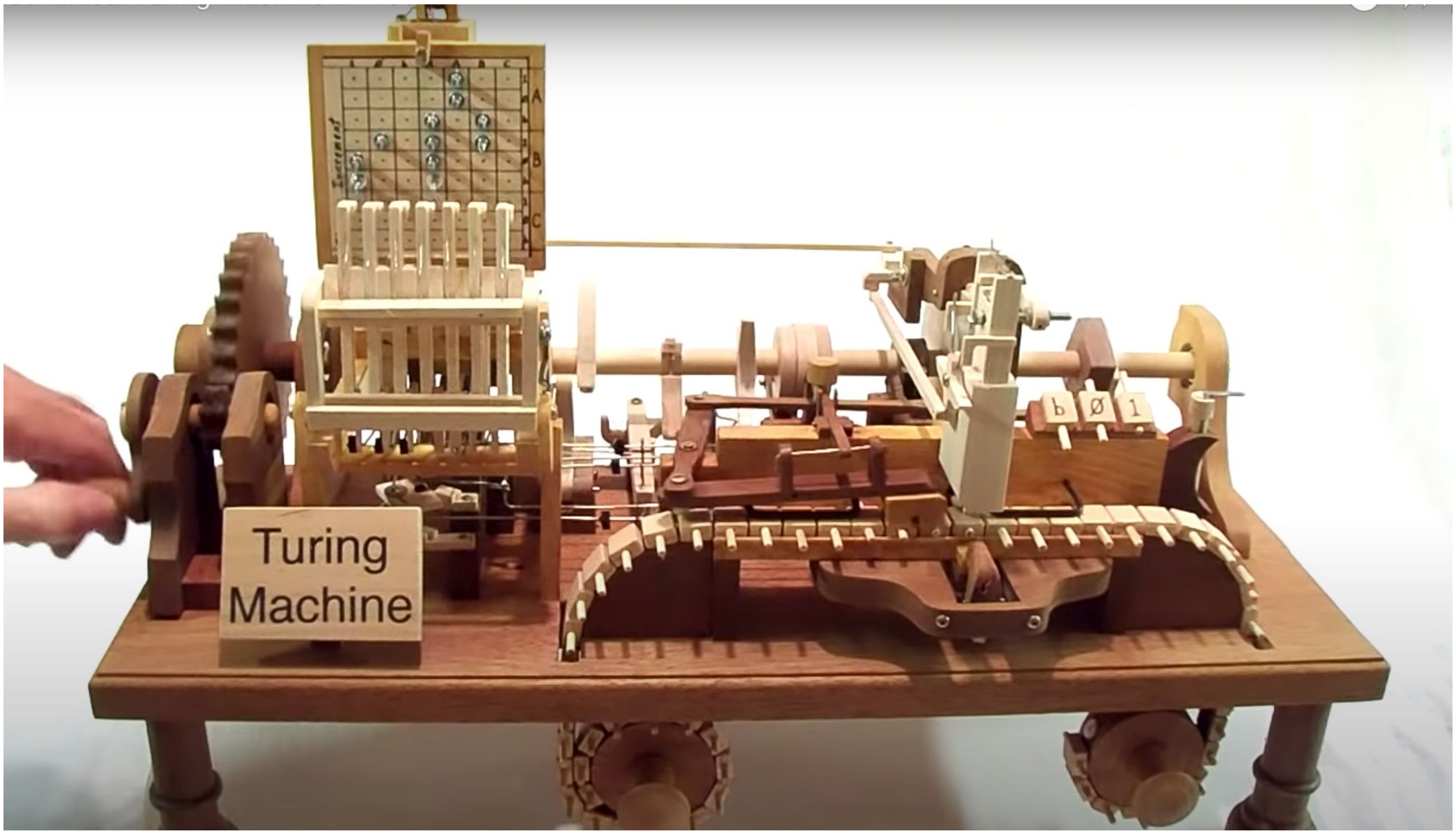
Q: Set of inputs accepted by this TM?







A Mechanical Turing Machine



Formal Definition of a Turing Machine

* A Turing machine is a 7-tuple:

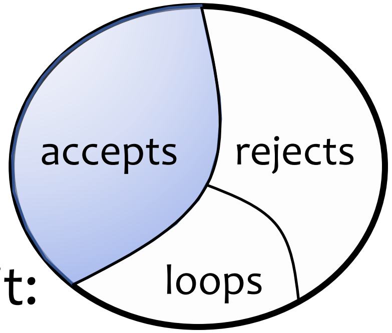
  $$M = \langle Q, \Gamma, \Sigma, \delta, q_{start}, q_{accept}, q_{reject} \rangle$$

  * Q = set of **states**
  * Σ = the **input** alphabet (typically {0,1} but not always)
  * $\perp$ = the **blank symbol**
  * Γ = the **tape alphabet** where generally $\Gamma = \Sigma \cup \{\perp\}$
  * $q_{start} \in Q,$ = the **initial state**
  * $F = \{q_{accept}, q_{reject}\} \subseteq Q,$ = the set of **final states** (one accepting state and one rejecting state)
  * $\delta: (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ = the **transition function**

The Language of a TM

Question: What are the possible outcomes of a TM **M**?

Answer: **M** either (i) accepts, (ii) rejects, *or* (iii) it *"loops" (forever)*



The language of a TM is the set of strings it accepts:

$$L(M) = \{x : M \text{ accepts } x\}$$

Definition: A Turing Machine **M** ***recognizes*** a language L if it:

1. *accepts* every string in L, and
2. *rejects* OR loops for every string not in L.

Definition: A Turing Machine **M** ***decides*** a language L if it:

1. *accepts* every string in L, and
2. *rejects* every string not in L (and never loops forever)

A *language* L is ***decidable*** if there is a TM that decides L.

Otherwise L is ***undecidable***.

Fact: If a language **L** is decidable, then the **complement of L** is also decidable. Why?

Fact: If languages $\mathbf{L_1}$ and $\mathbf{L_2}$ are both decidable, then $\mathbf{L_1 \cap L_2}$ and $\mathbf{L_1 \cup L_2}$ are both decidable. Why?

What if we added more tapes to a TM?

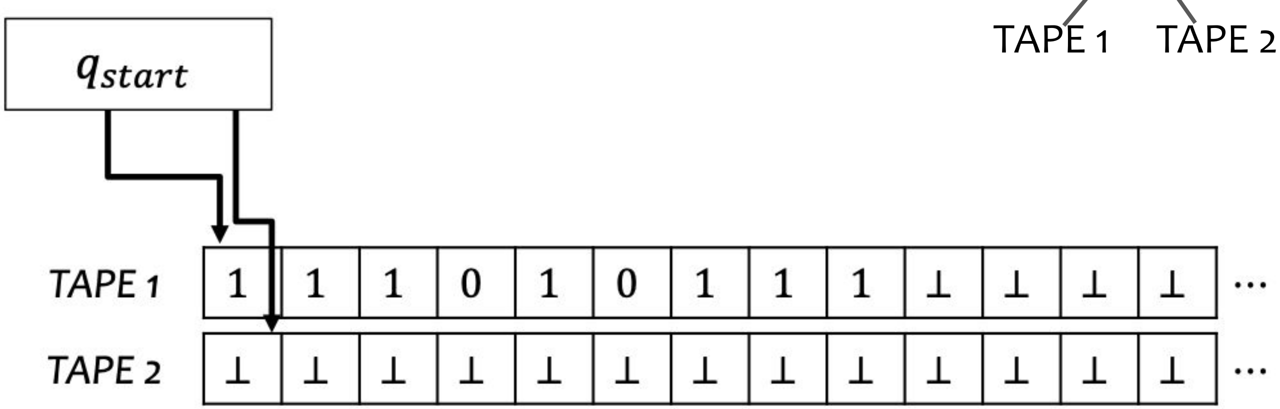
Well, the Church-Turing Thesis says there's nothing more powerful than a 1-tape TM.

So a 2-tape TM better not be more powerful. Let's prove it!

TAPE 1

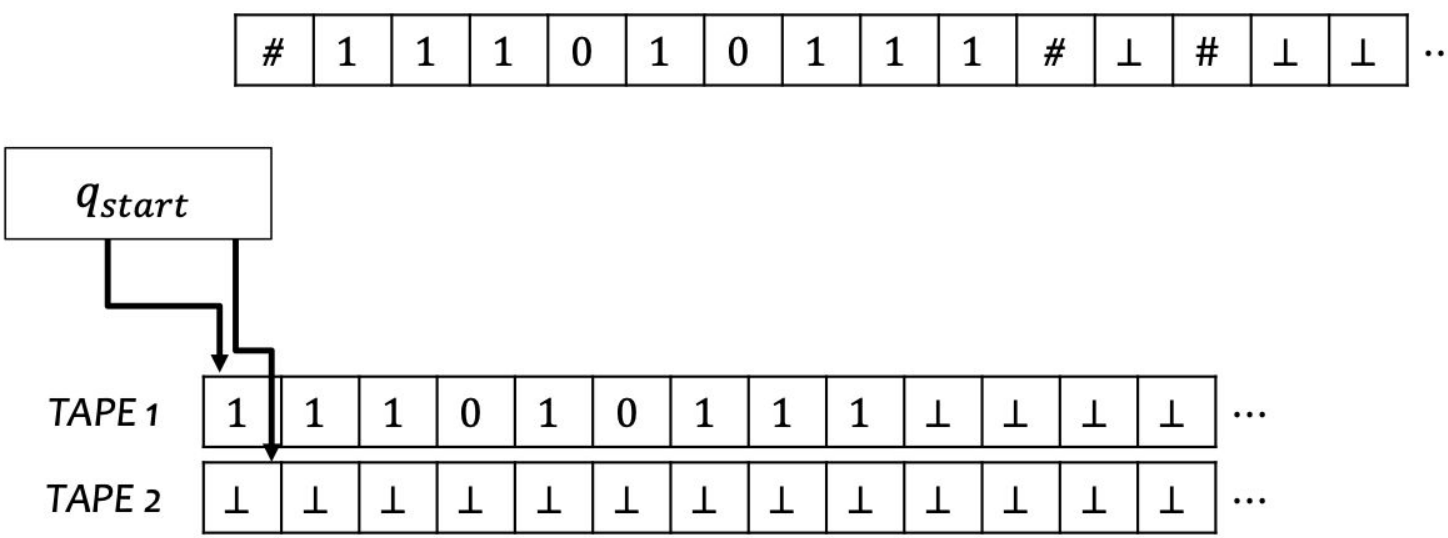
Example transition rule for 2-tape TM: $\delta(q_1, \underbrace{1}_{\text{TAPE 1}}, \underbrace{\perp}_{\text{TAPE 2}}) = (q_2, 0, 1, R, L)$

TAPE 2



Equivalence of 1-tape and 2-tape TMs

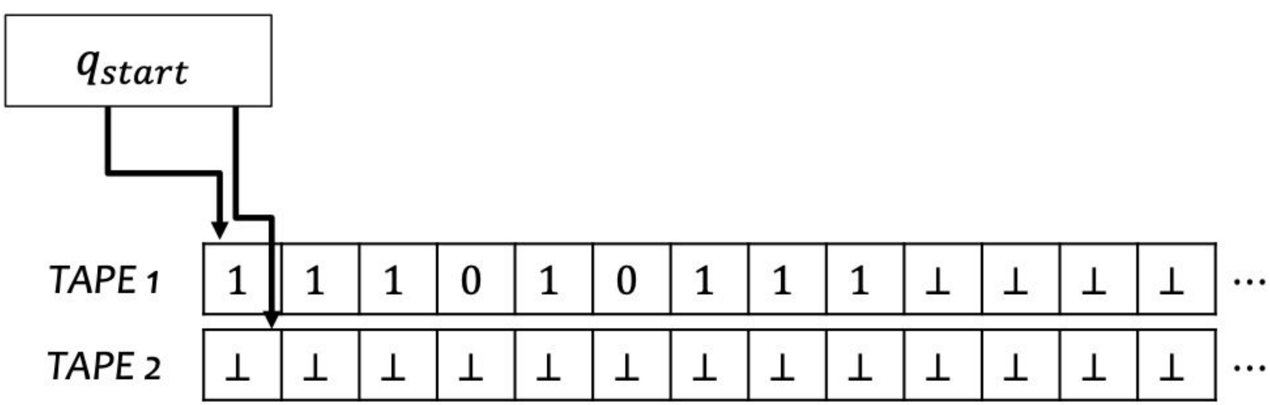
Want to show: Given a 2-tape TM, we can simulate its execution using a 1-tape TM. **How?**



Equivalence of 1-tape and 2-tape TMs

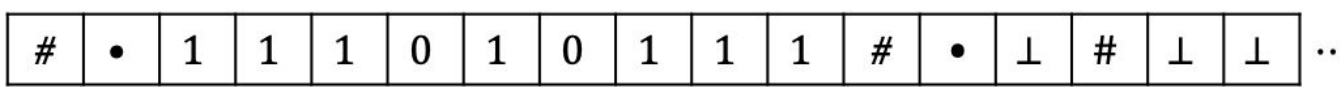
* **Idea:** Need to demarcate location of each head – use a • symbol to the left of the cell where the head is.

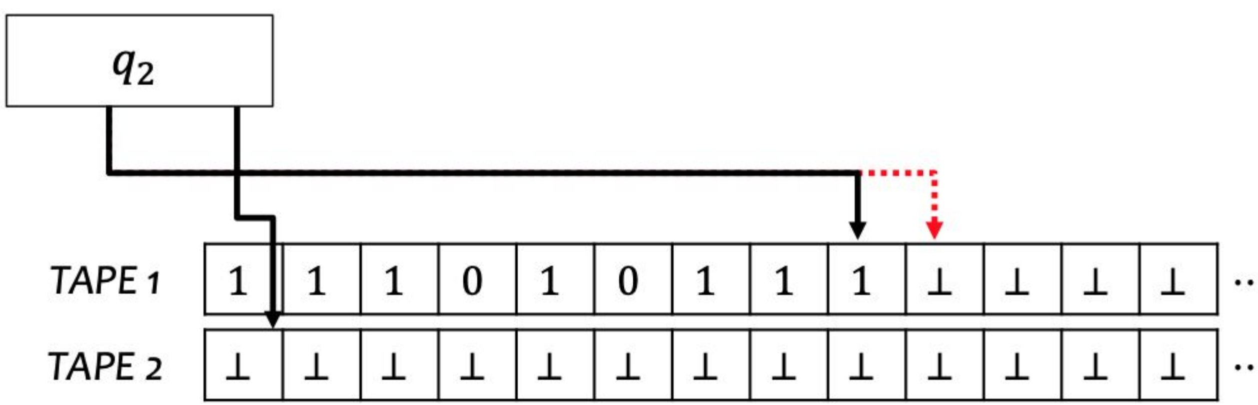
#	•	1	1	1	0	1	0	1	1	1	#	•	⊥	#	⊥	⊥	⋯



Equivalence of 1-tape and 2-tape TMs

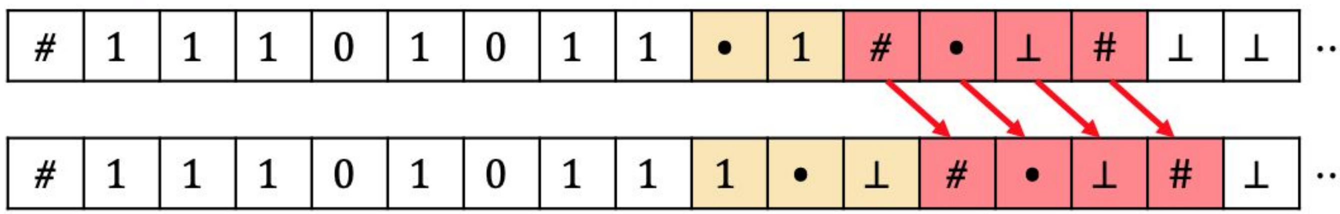
Issue 1: What if we need to insert a symbol at the end of TAPE 1 and there's no room?

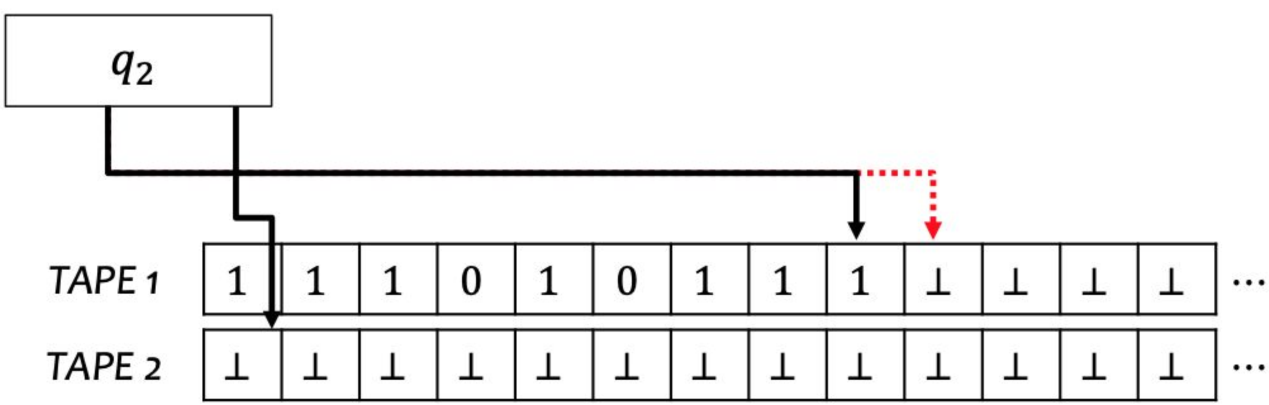




Equivalence of 1-tape and 2-tape TMs

Issue 1: What if we need to insert a symbol at the end of TAPE 1 and there's no room?





Equivalence of 1-tape and 2-tape TMs

Issue 2: To know which transition rule to use, I need to know the current symbols of **both** tapes of the 2-tape TM, but I only have 1 head on my 1-tape TM.

#	•	1	1	1	0	1	0	1	1	1	#	•	$\perp$	#	$\perp$	$\perp$	⋯

$$\delta(q_1, 1, \perp) = (q_2, 0, 1, R, L):$$

#	•	1	1	1	0	1	0	1	1	1	#	1	•	$\perp$	#	$\perp$	⋯
#	0	•	1	1	0	1	0	1	1	1	#	•	1	1	#	$\perp$	⋯

Equivalence of 1-tape and 2-tape TMs

Issue 2: To know which transition rule to use, I need to know the current symbols of **both** tapes of the 2-tape TM, but I only have 1 head on my 1-tape TM.

#	•	1	1	1	0	1	0	1	1	1	#	•	⊥	#	⊥	⊥	⋯

$$\delta(q_1, 1, \perp) = (q_2, 0, 1, R, L):$$

#	•	1	1	1	0	1	0	1	1	1	#	1	•	⊥	#	⊥	⋯

#	0	•	1	1	0	1	0	1	1	1	#	•	1	1	#	⊥	⋯