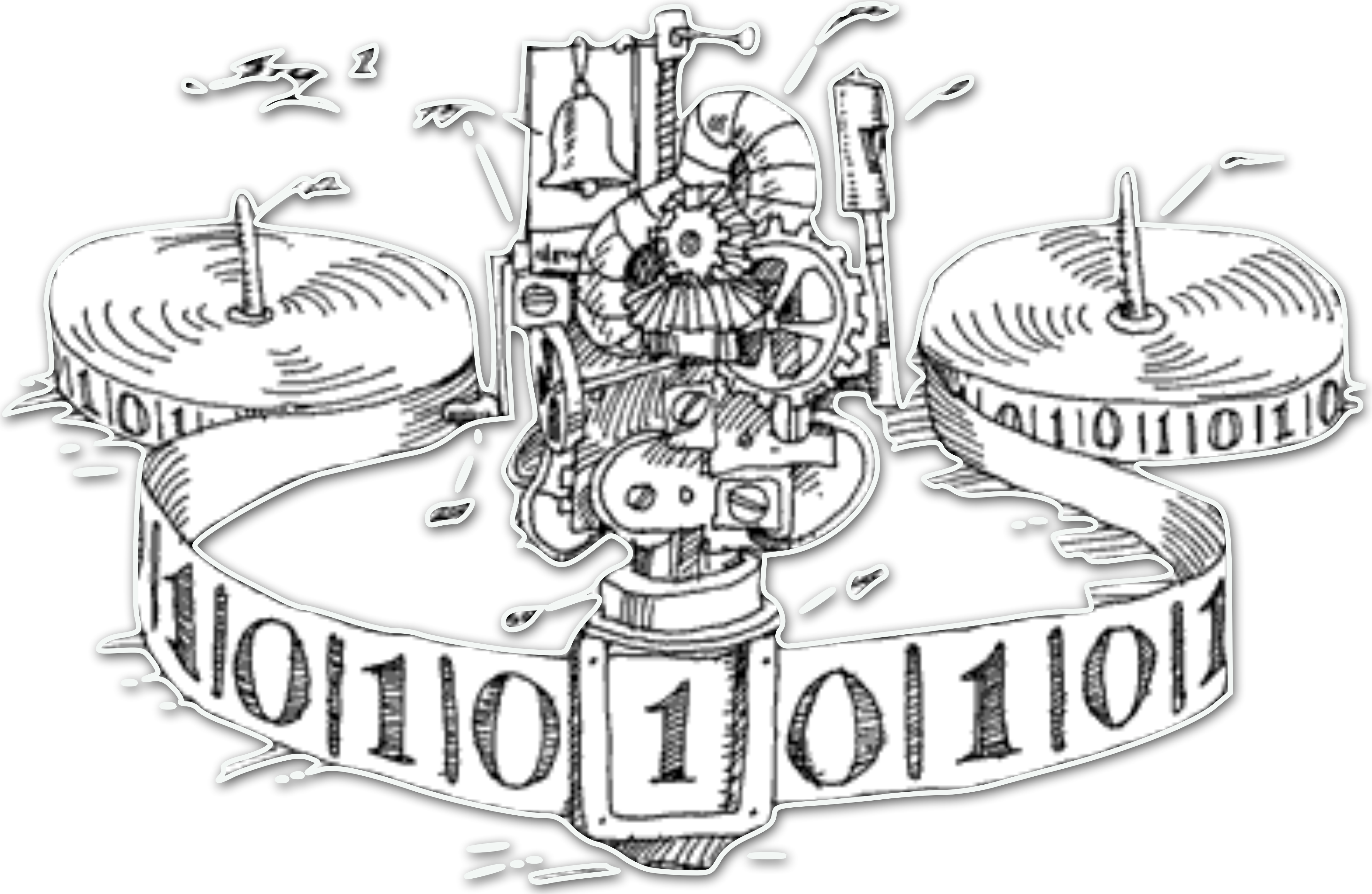
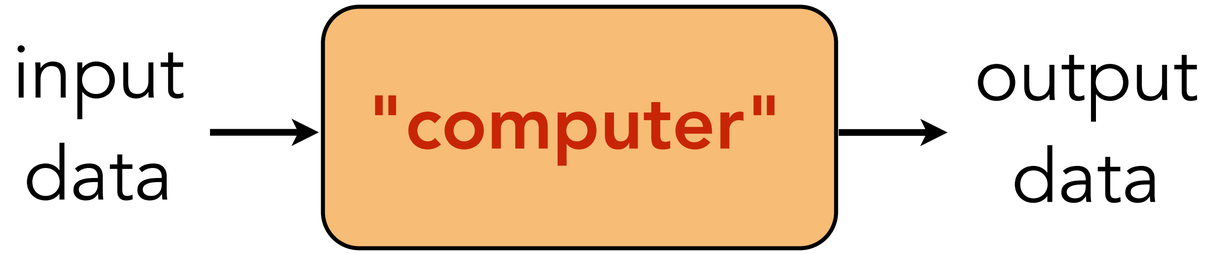
CS251

Great Ideas in *Theoretical* Computer Science



Turing Machines

This Chapter



What is **computation**?

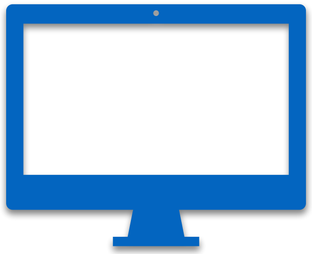
What is an **algorithm**?

How can we mathematically define them?

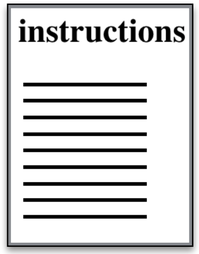
Terminology Reminder:

Computational Model — Allowed rules for information processing.

Machine = Computer = Program = Algorithm — An instantiation of the computational model.
(a specific sequence of information processing rules)



physical realization



mathematical representation

Terminology Reminder:

Computational Model — Allowed rules for information processing.

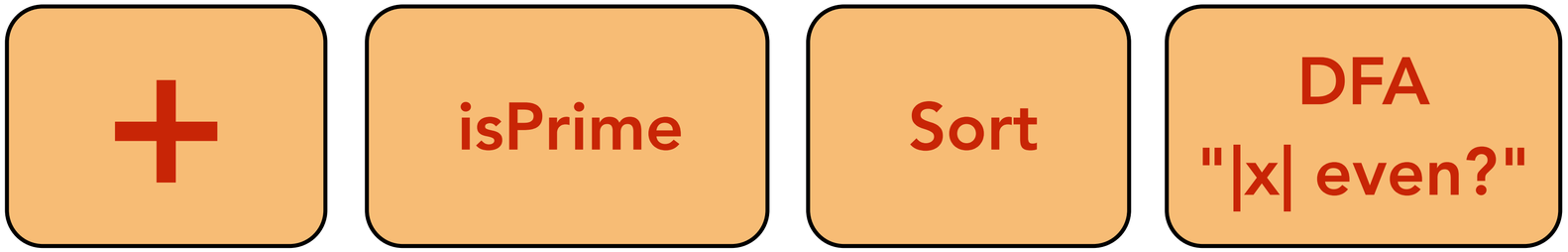
The Turing machine computational model

Machine = Computer
= Program = Algorithm — An instantiation of the computational model. (a specific sequence of information processing rules)

A Turing machine

2 Assumptions:

1. No "universal machines".



2. We only care about decision problems.

Goal of this lecture:

Define Turing machines.

Understand how they work.

Goal of next lecture:

Explore physical, philosophical, historical questions surrounding Turing machines.

What was Turing's motivation?

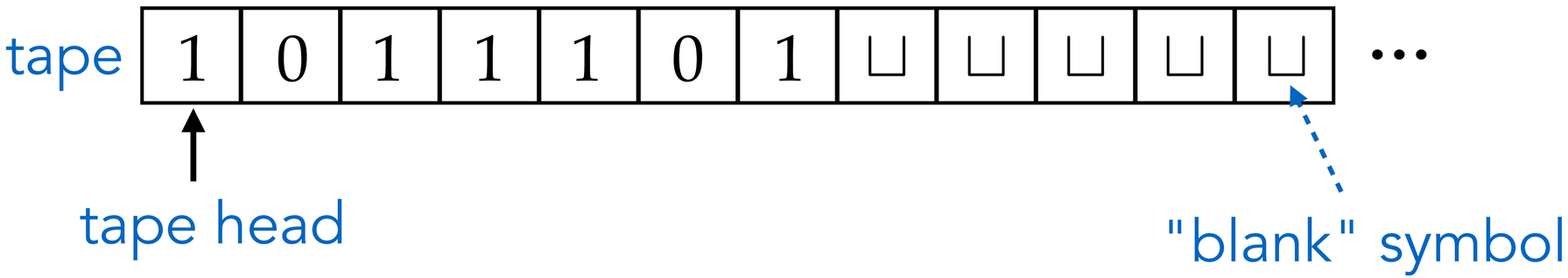
How did he come up with his definition?

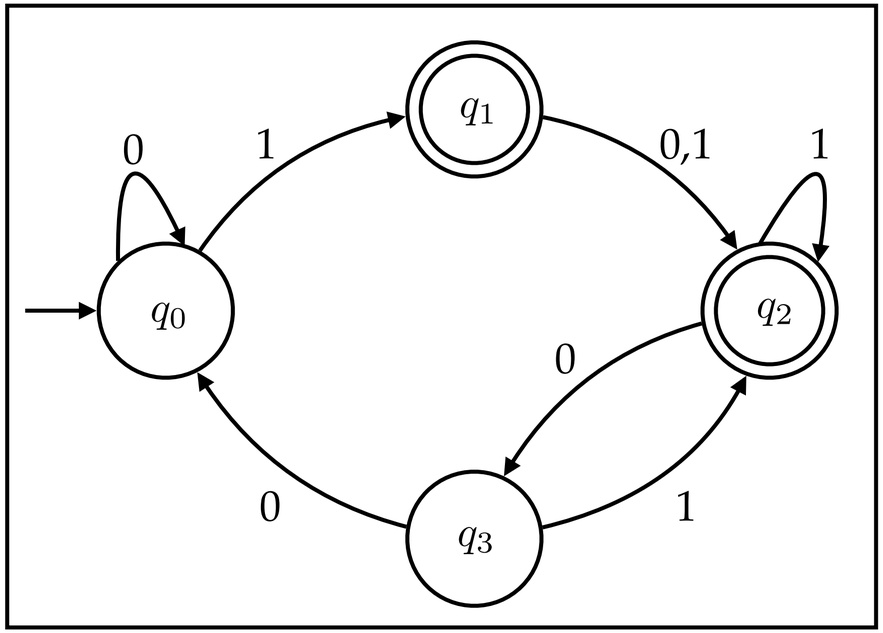
What are the (amazing) implications?

How did our understanding evolve over time?

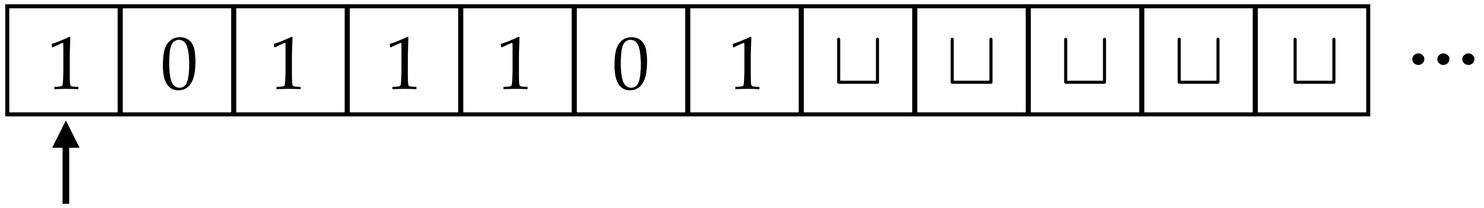
Quick Reminder on DFAs

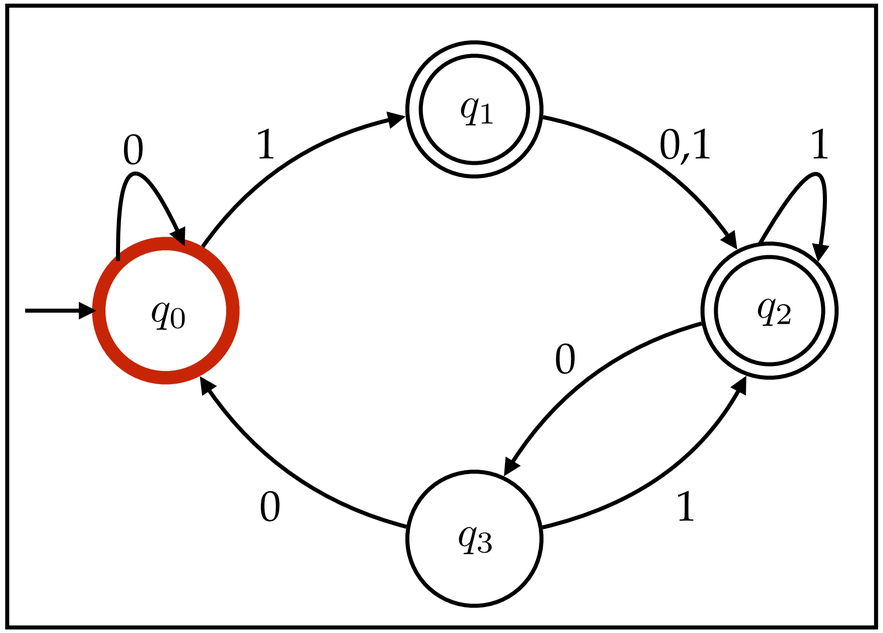
DFA: state diagram + input tape



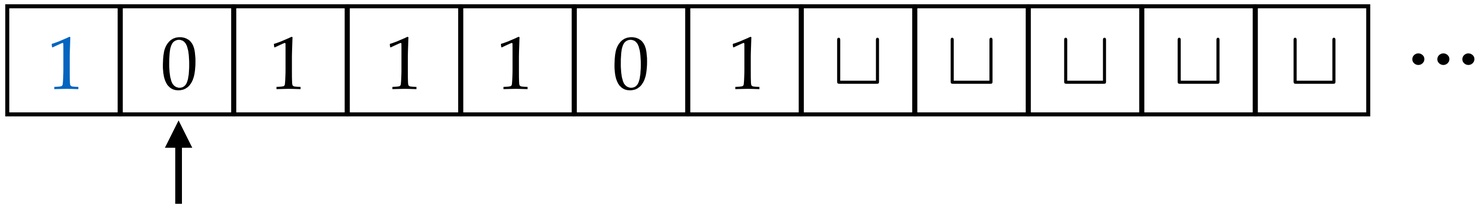


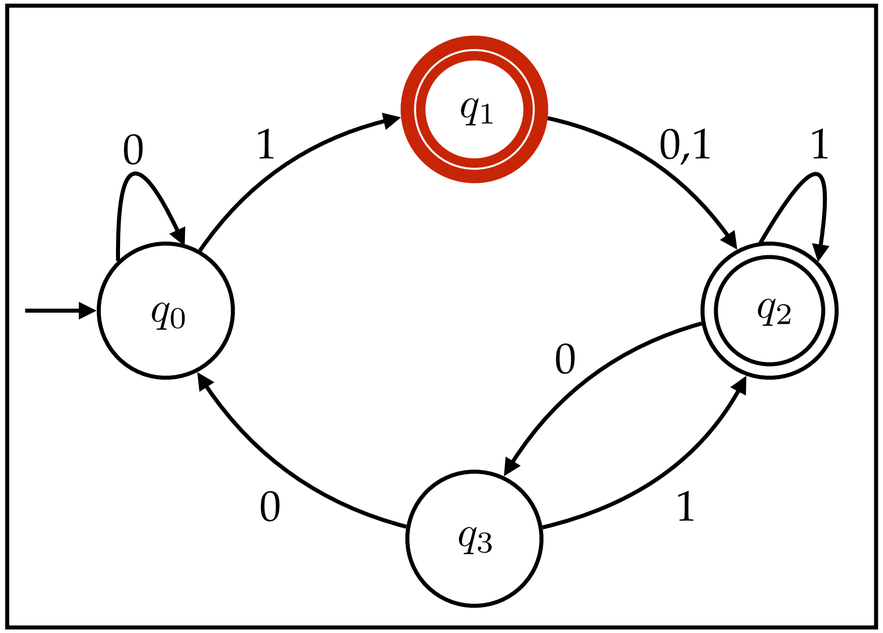
DFA: state diagram + input tape



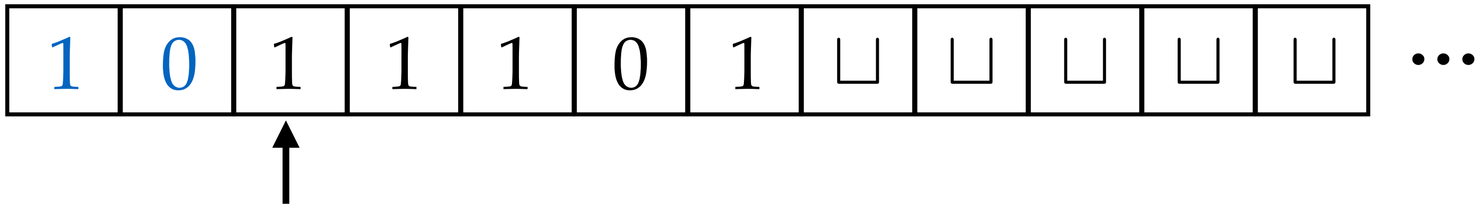


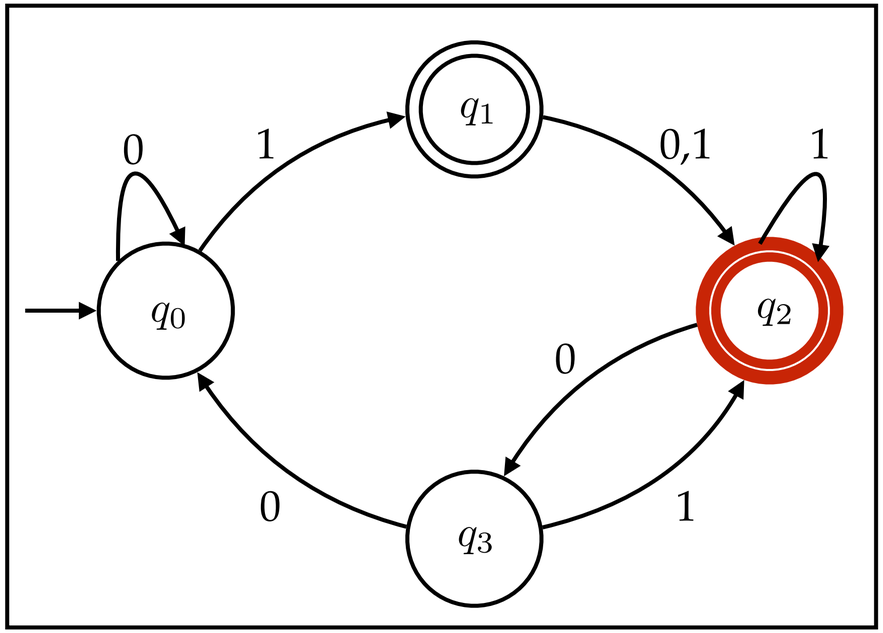
DFA: state diagram + input tape



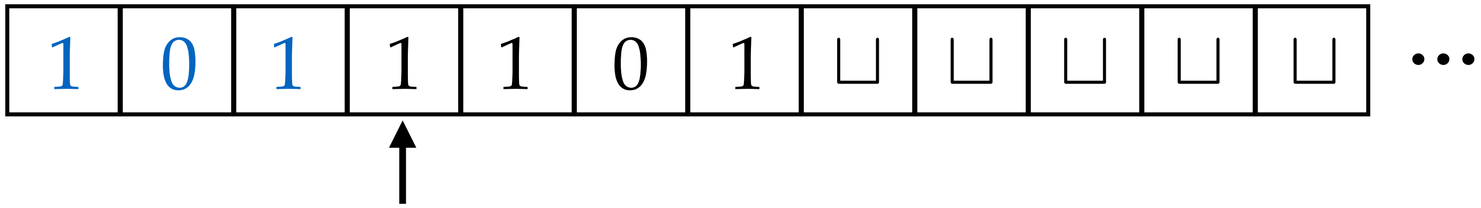


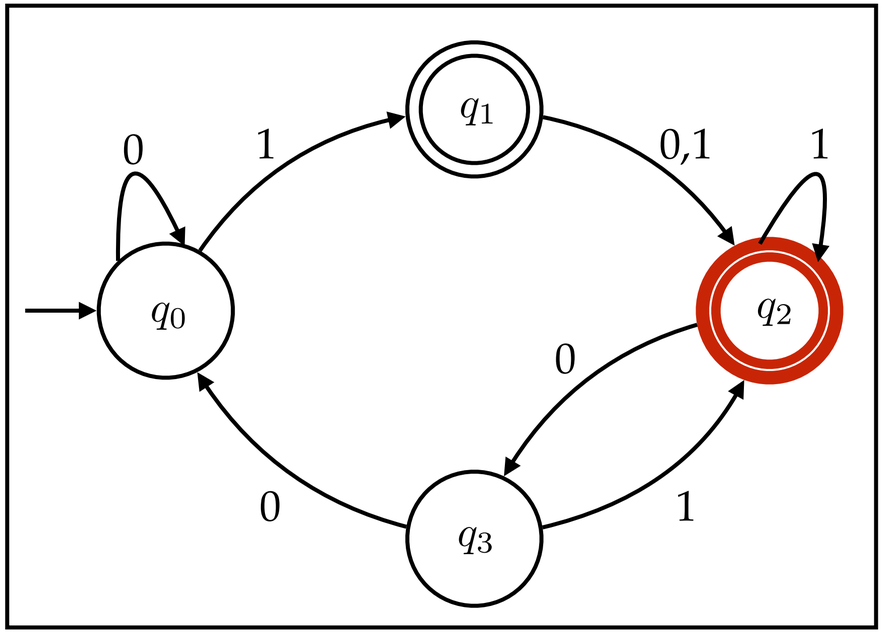
DFA: state diagram + input tape



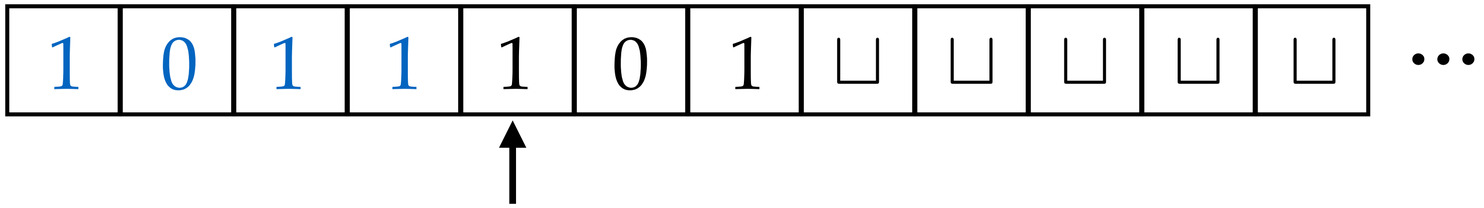


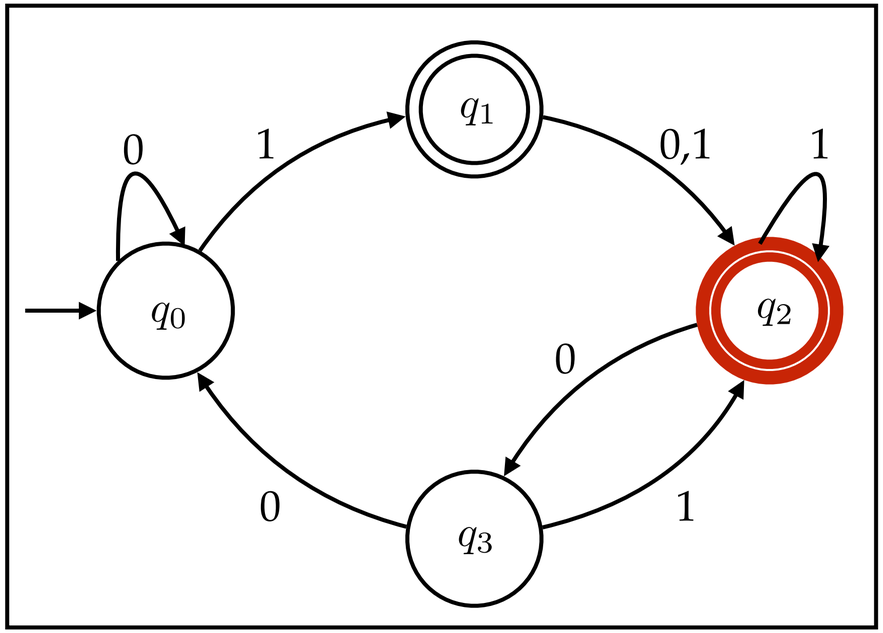
DFA: state diagram + input tape



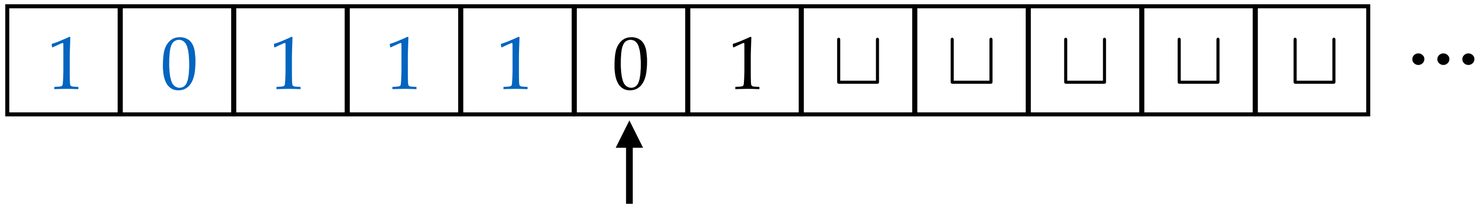


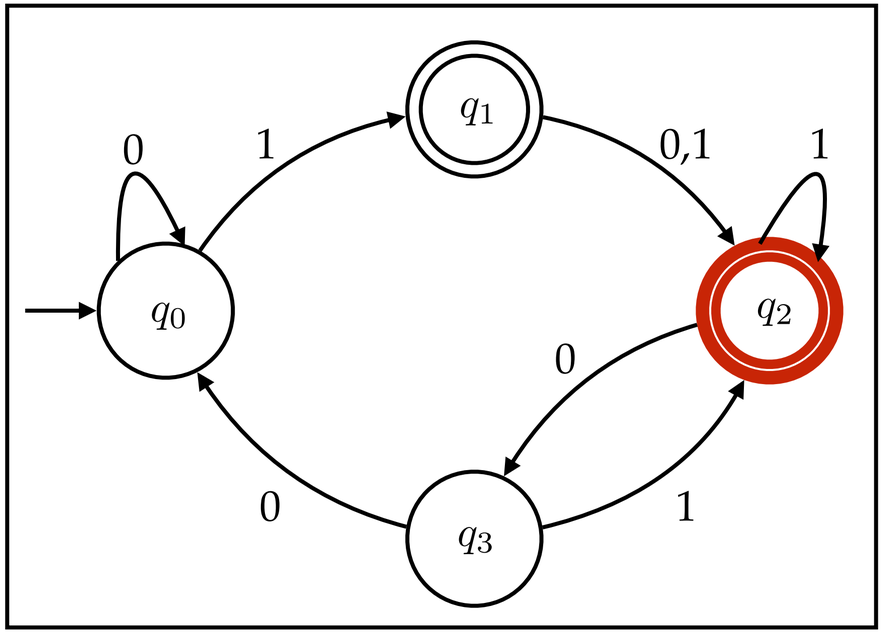
DFA: state diagram + input tape



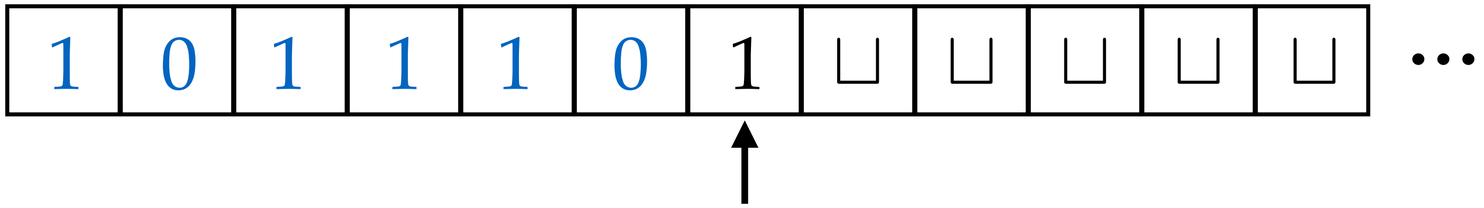


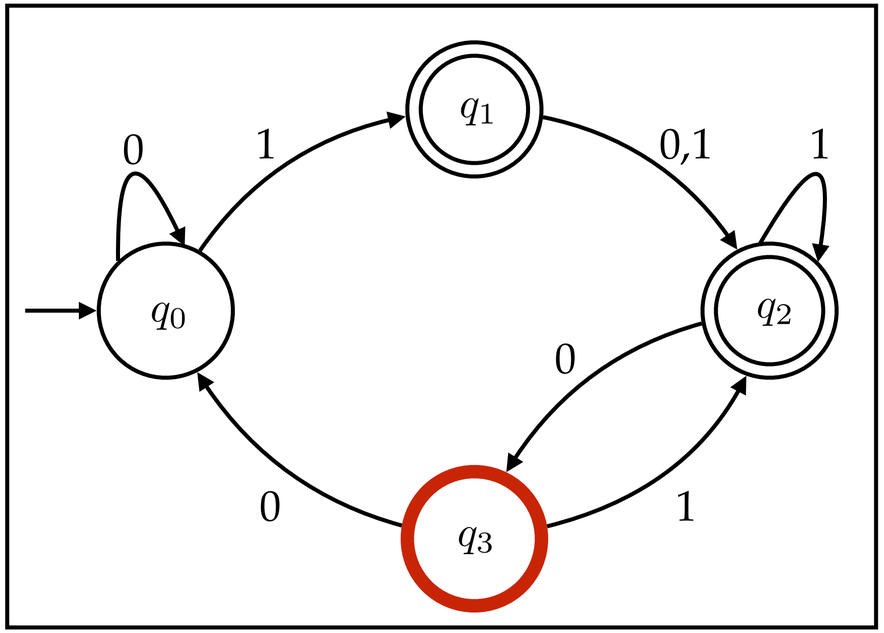
DFA: state diagram + input tape



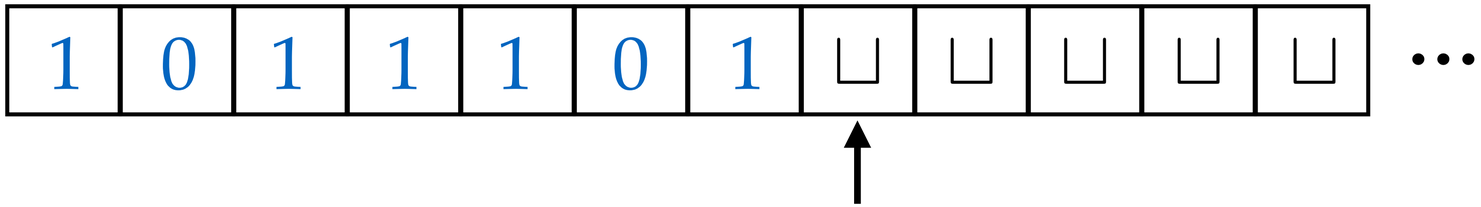


DFA: state diagram + input tape

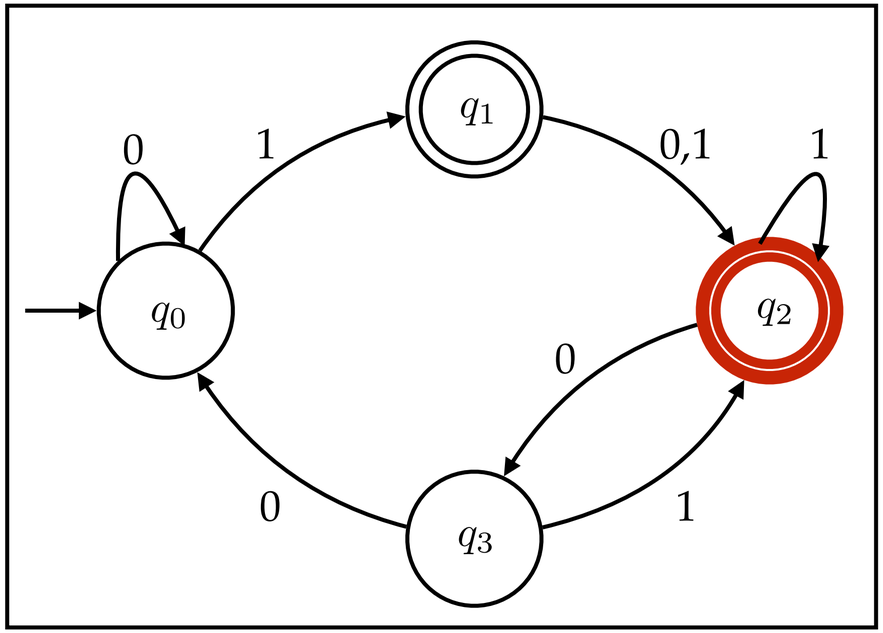




DFA: state diagram + input tape



Decision: Accept



DFA as a programming language

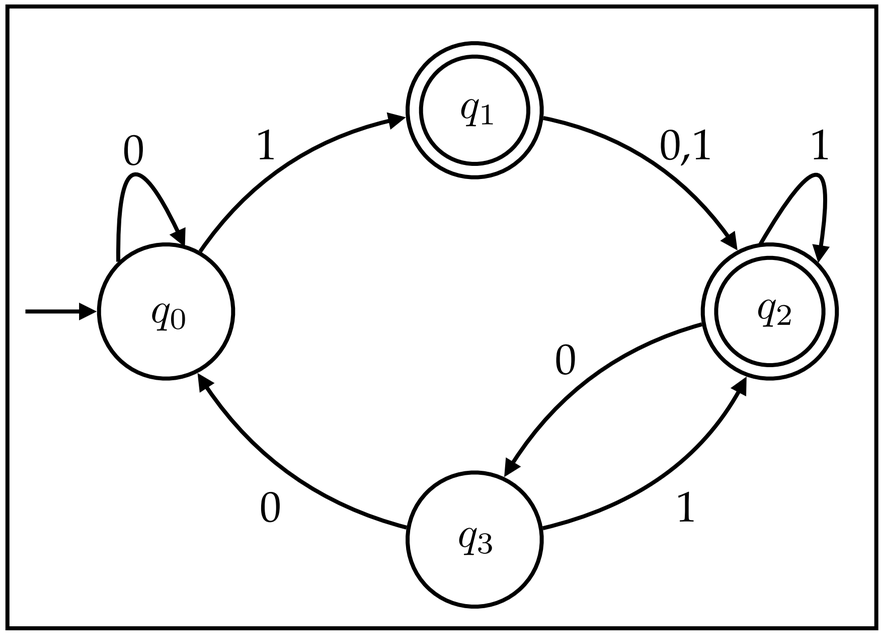
```
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case ‘0’:  go to STATE 0;
            case ‘1’:  go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case ‘0’:  go to STATE 2;
            case ‘1’:  go to STATE 2;

    ...
```

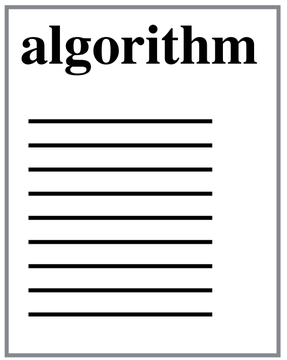
input =

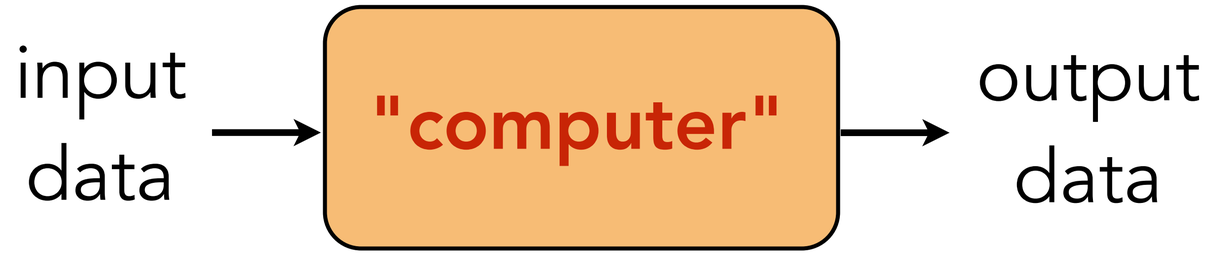
0	1	1	1	1



Machine ≈ Algorithm

|||





What is **computation**?

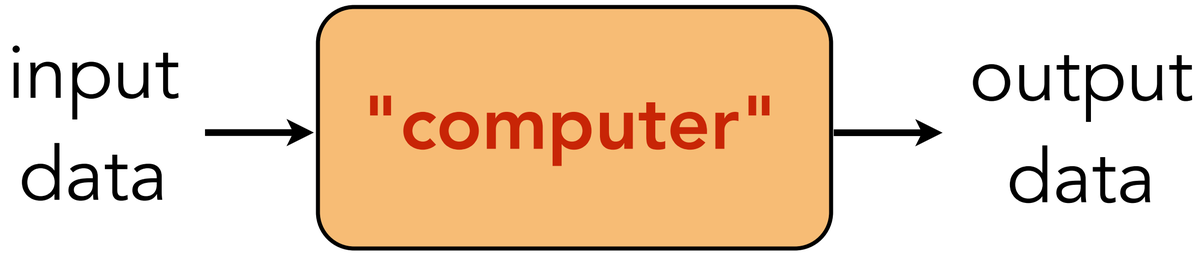
What is an **algorithm**?

How can we mathematically define them?

The properties we want from the definition:

Simplicity!

Generality!

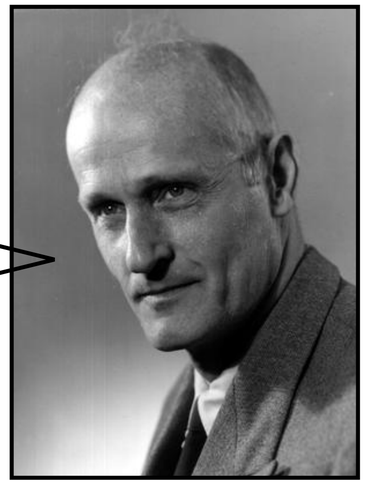


2 important observations:

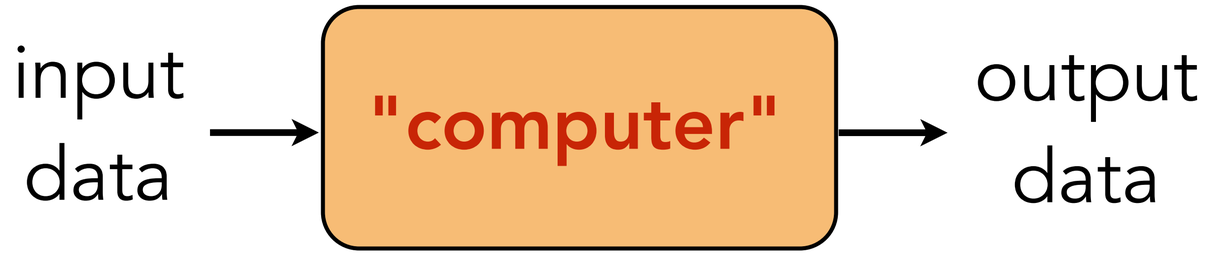
1. The device should be a "finite object".

An algorithm should be a "finite object".

An algorithm is a finite answer
to infinite number of questions.



Stephen Kleene



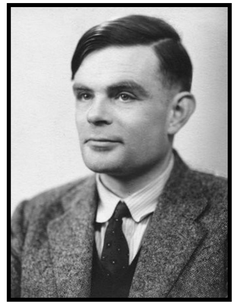
2 important observations:

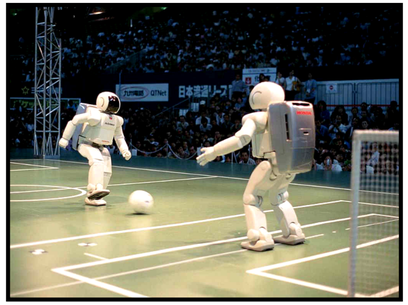
1. The device should be a "finite object".

An algorithm should be a "finite object".

2. It should have "unbounded memory".

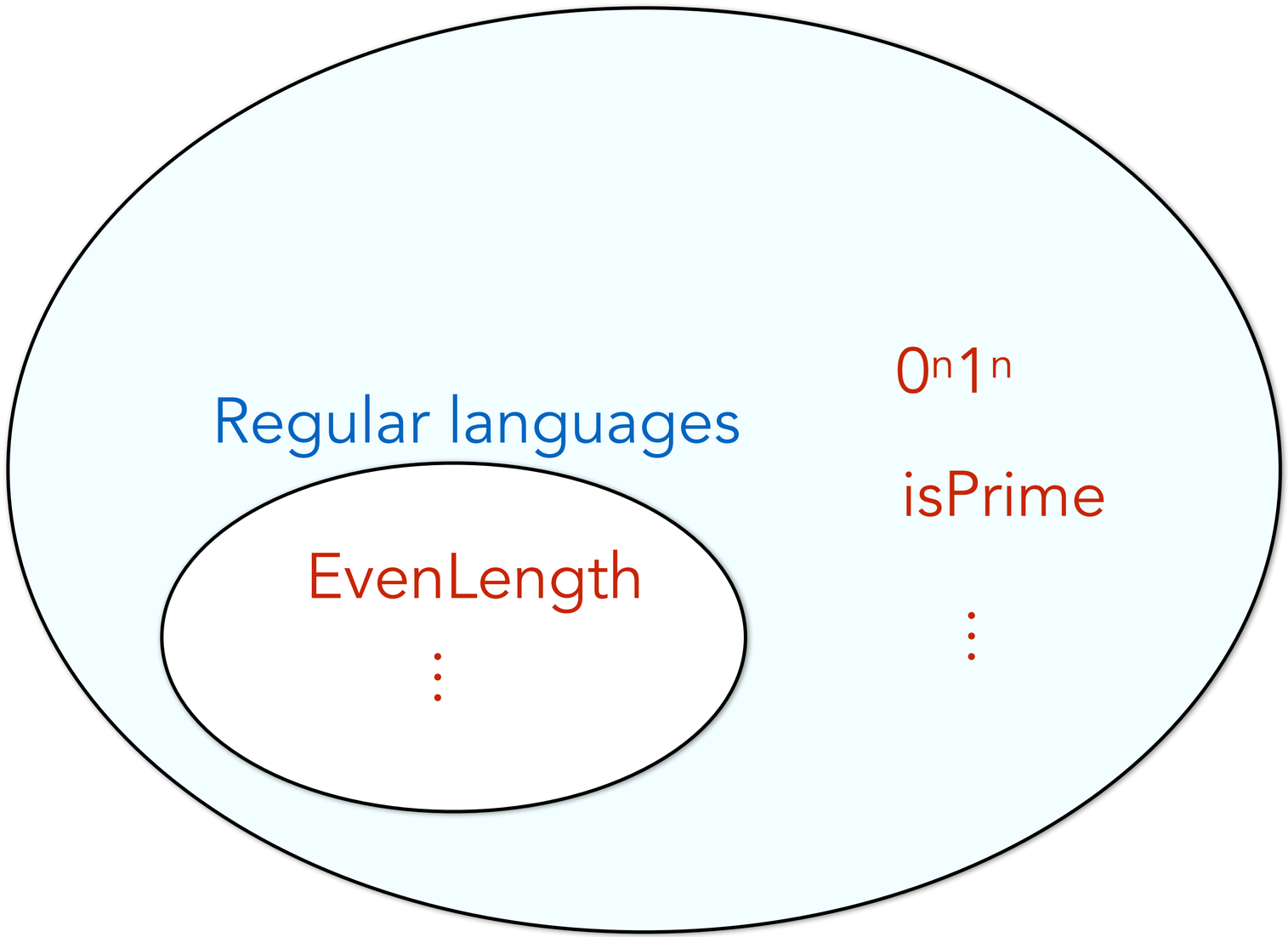
(there is always more space to work on, if needed)





1900 1936 2023

Goal is to reach the definition of a Turing machine.

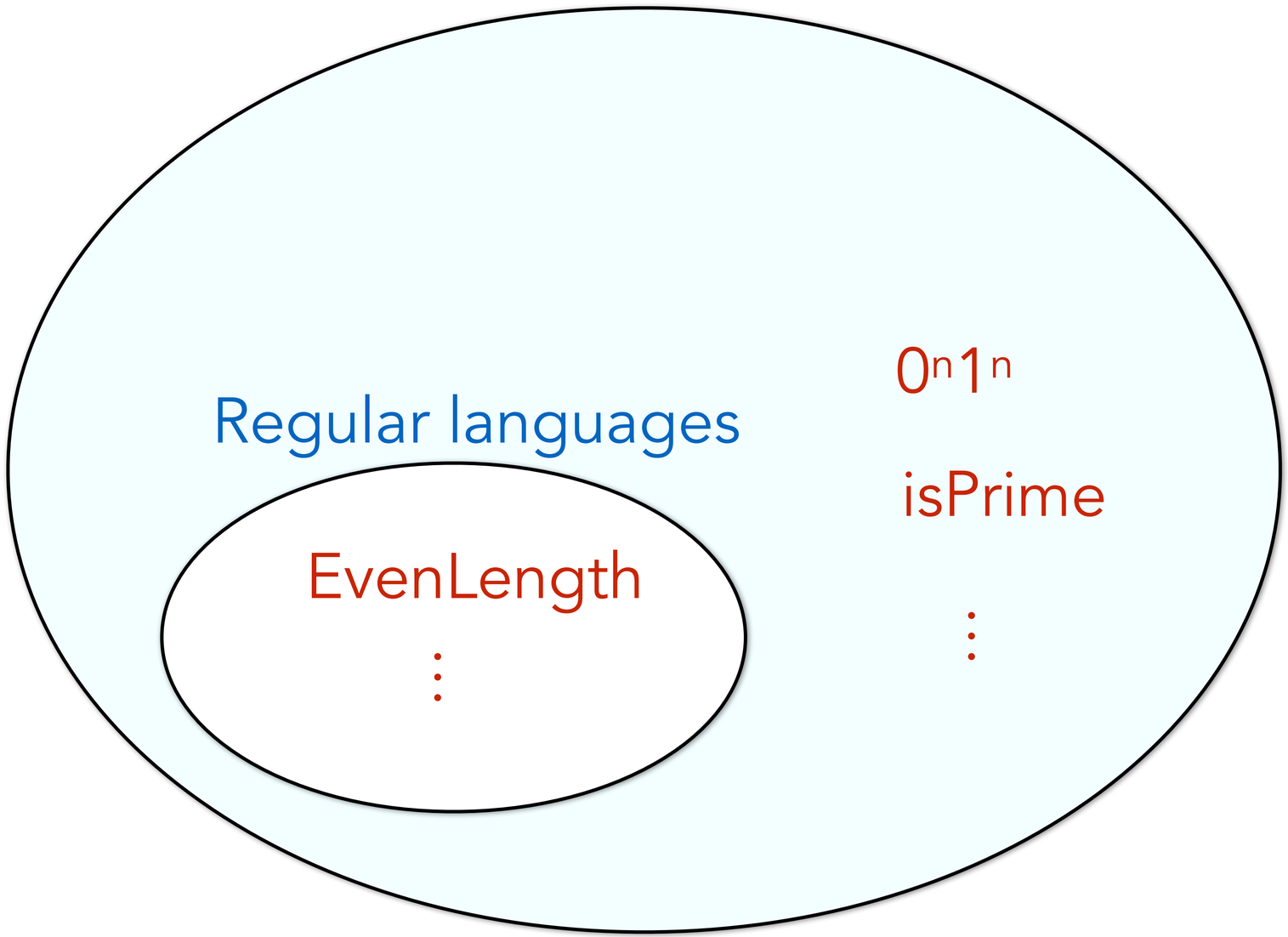

Solvable with any computing device
Regular languages
EvenLength
0^n1^n
isPrime

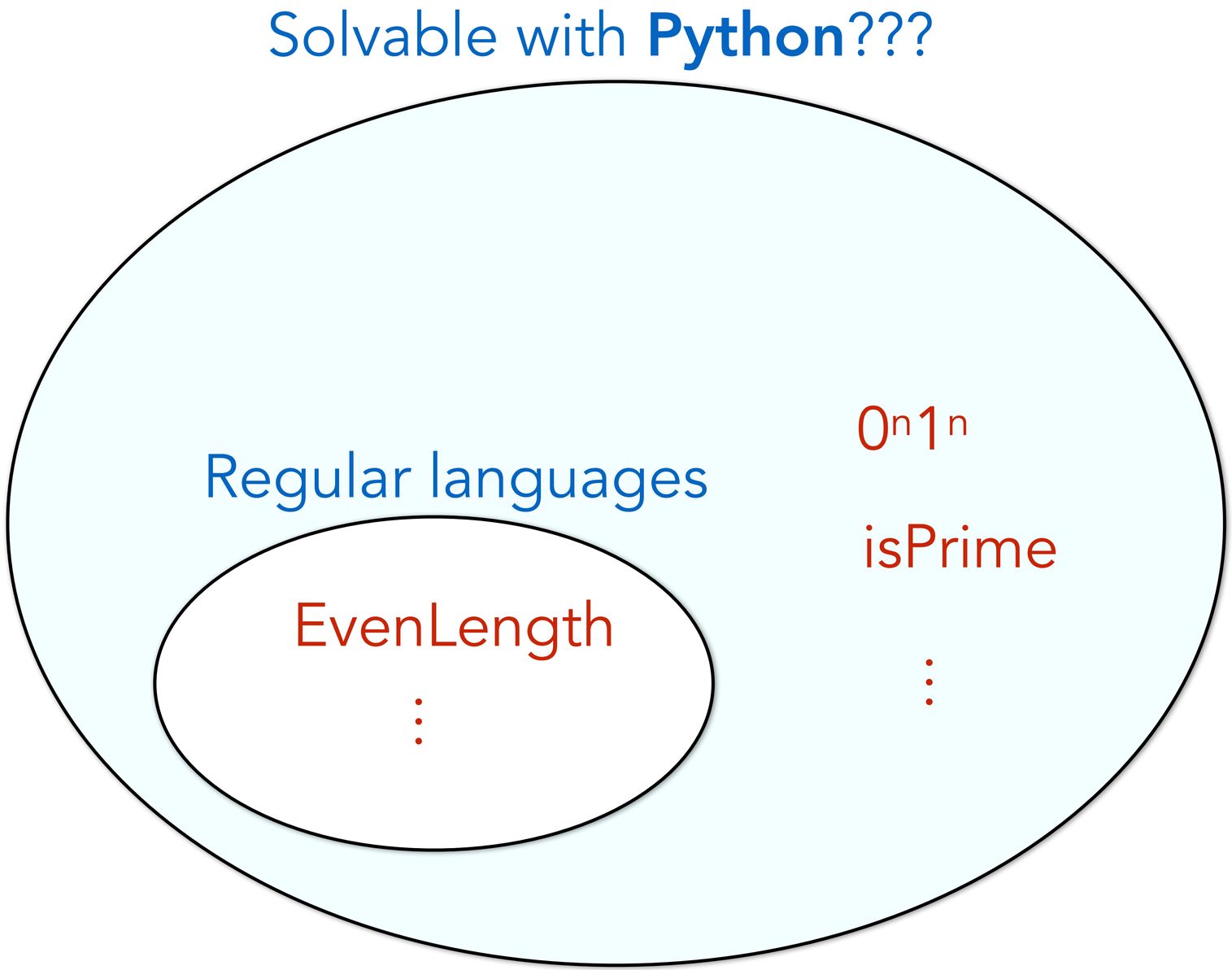
Solving 0^n1^n in Python

```
def foo(input):
  i = 0
  j = len(input) - 1
  while(j >= i):
    if(input[i] != '0' or input[j] != '1'):
      return False
    i = i + 1
    j = j - 1
  return True
```

Solving 0^n1^n in C

```
int foo(char input[]) {
  int i = 0, j;
  while (input[j] != NULL)
    j++;
  j--;
  while (j >= i) {
    if (input[i] != '0' || input[j] != '1')
      return 0;  /* Reject */
    i++;
    j--;
  }
  return 1;  /* Accept */
}
```


Solvable with any computing device
Regular languages
EvenLength
0^n1^n
isPrime


Solvable with **Python**???
Regular languages
EvenLength
0^n1^n
isPrime

Should we define **computable** to mean
what is computable by a Python program?

Potential Issues?

- Why Python, why not C, Java, SML,… ?

- Is it general enough?

- Is it simple enough?

Should we define **computable** to mean
what is computable by a Python program?

Potential Issues?

- Why Python, why not C, Java, SML,… ?

- Is it general enough?

- **Is it simple enough?**

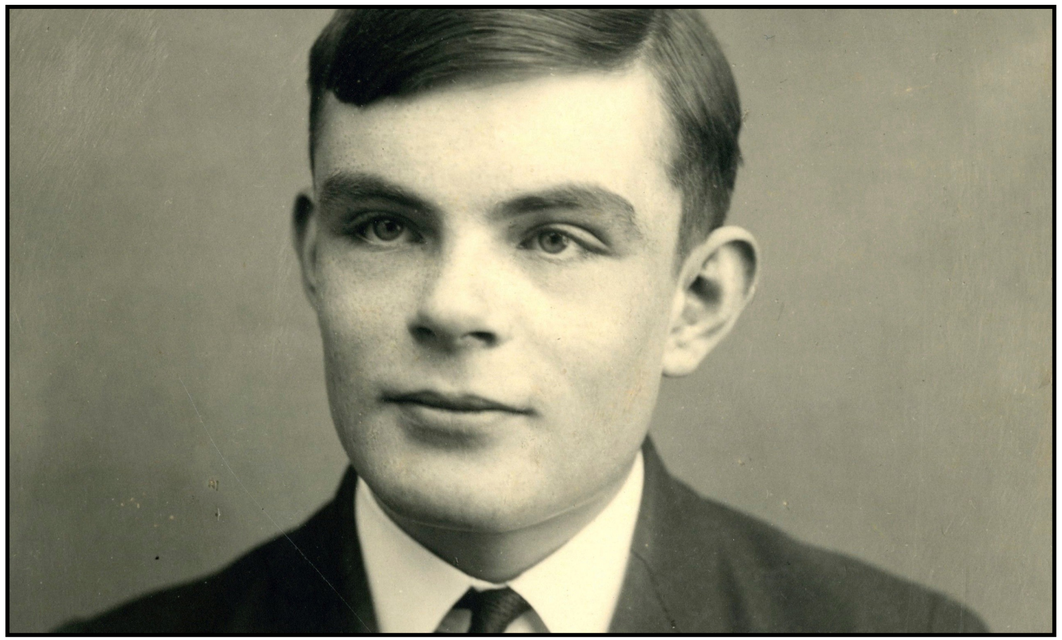
So what we want is:

A totally minimal (TM) programming language such that:

- it can simulate simple bytecode;
  (and therefore Python, C, Java, SML, etc…)

- it is simple to define and reason about mathematically rigorously.

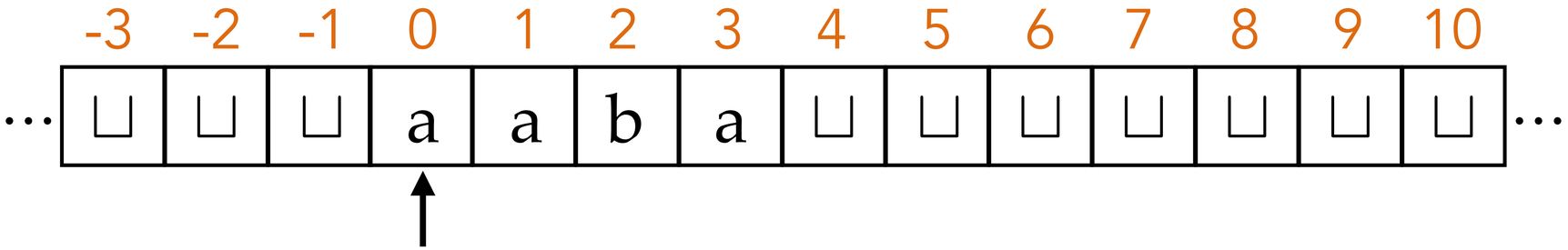
Actually TM™ stands for Turing machine.



Defined by Alan Turing in 1936
while he was a PhD student.

Turing machine description

TM ≈ DFA + infinite tape



Input written on the tape starting at index 0.

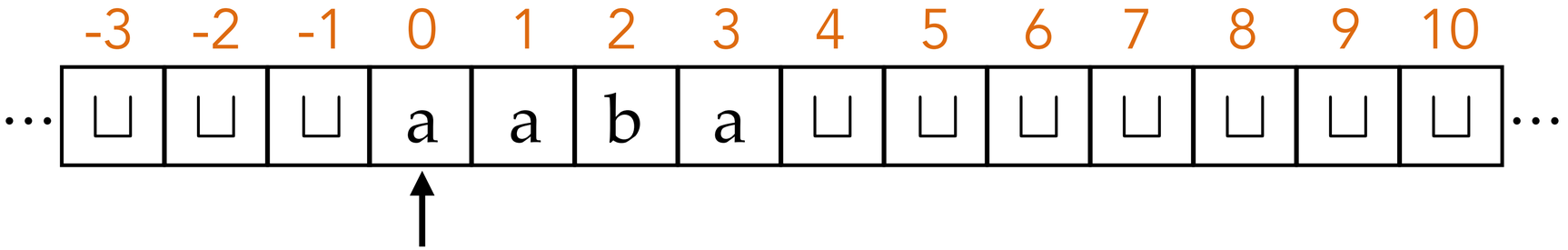
All other cells contain the ***blank*** symbol ⊔.

There is a tape ***pointer/head*** (initially at index 0),
can move left or right.

Can read & **write** to the tape cell pointed to.

Turing machine description

TM ≈ DFA + infinite tape



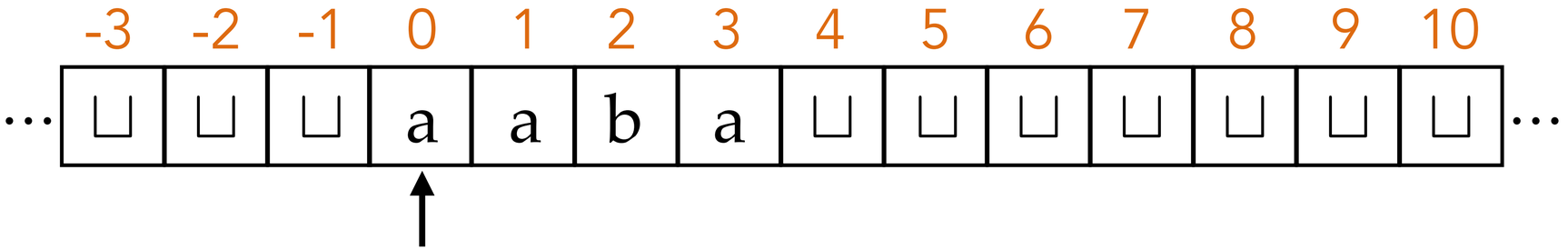
TM could be defined as a sequence of instructions, where the allowed instructions are:

> Move the head left
> Move the head right
> Write symbol a (from the alphabet)
> If head is reading symbol a, GOTO step/line j
> Halt and accept
> Halt and reject

But, we want to keep the definition as simple as possible.

Turing machine description

TM ≈ DFA + infinite tape



So a TM is a sequence of steps (states), each looking like:

STATE 0:
    switch (letter under the head):
        case ‘a’: **write** ‘b’; **move** Left; **go to** STATE 2;
        case ‘b’: **write** ‘␣’; **move** Right; **go to** STATE 0;
        case ‘␣’: **write** ‘b’; **move** Left; **go to** STATE 1;

Turing machine description

STATE 0:
    switch (letter under the head):
        case ‘a’: **write** ‘b’; **move** Left; **go to** STATE 2;
        case ‘b’: **write** ‘␣’; **move** Right; **go to** STATE 0;
        case ‘␣’: **write** ‘b’; **move** Left; **go to** STATE 1;

At each step, you have to:

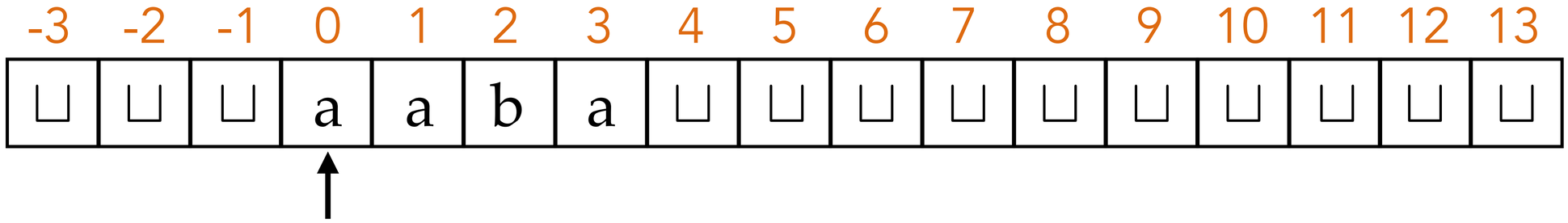
- **write** a *new* symbol to the cell under the tape head
- **move** tape head Left or Right
- **go to** a *new* state

Don't want to change the symbol: **write** same symbol.

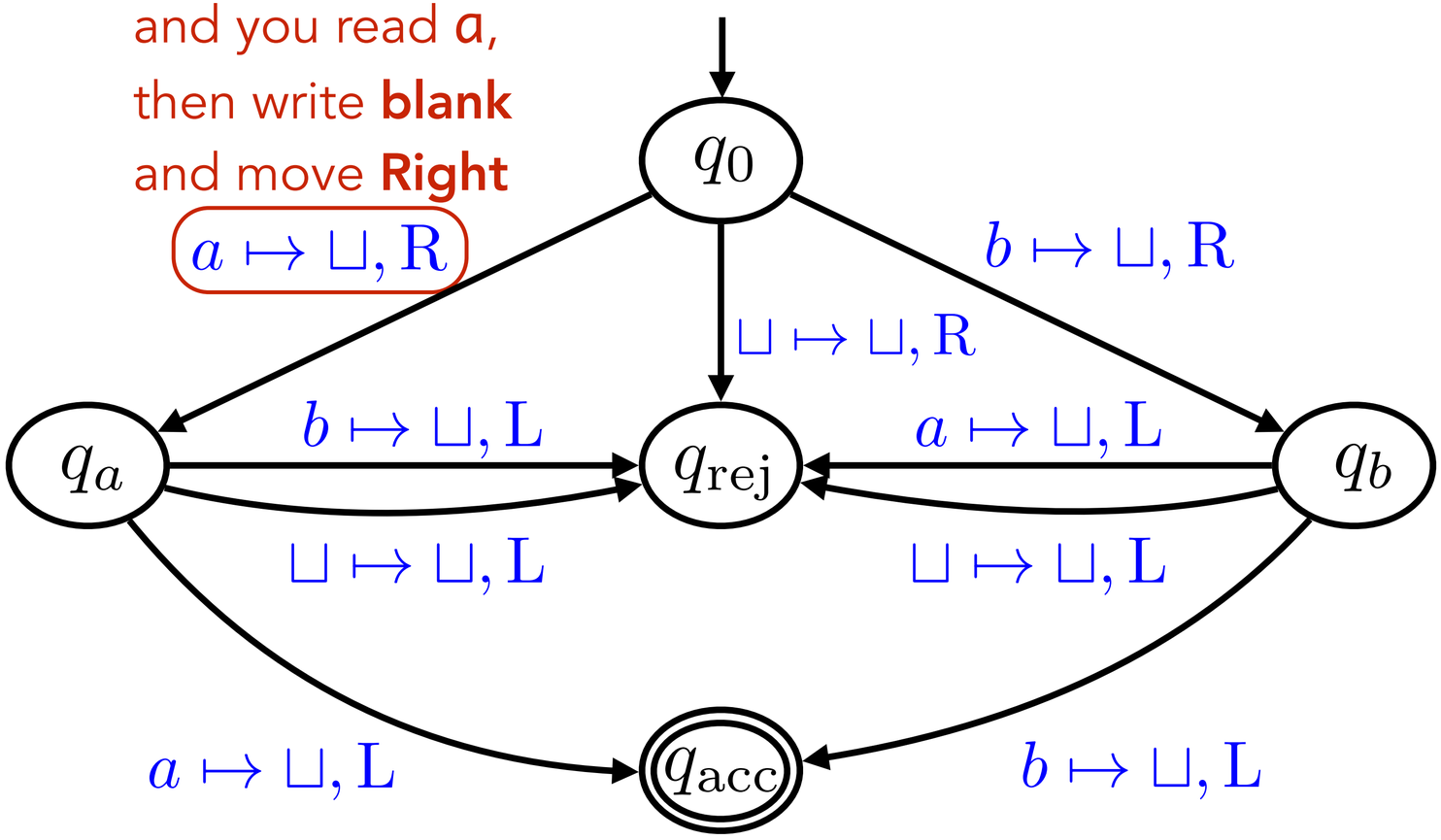
Want to stay put: **move** Left then Right.

Don't want to change state: **go to** same state.

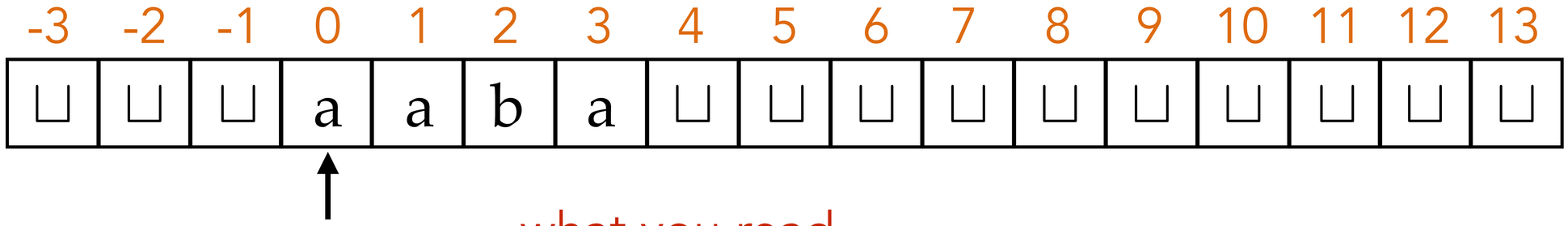
Turing machine state diagram

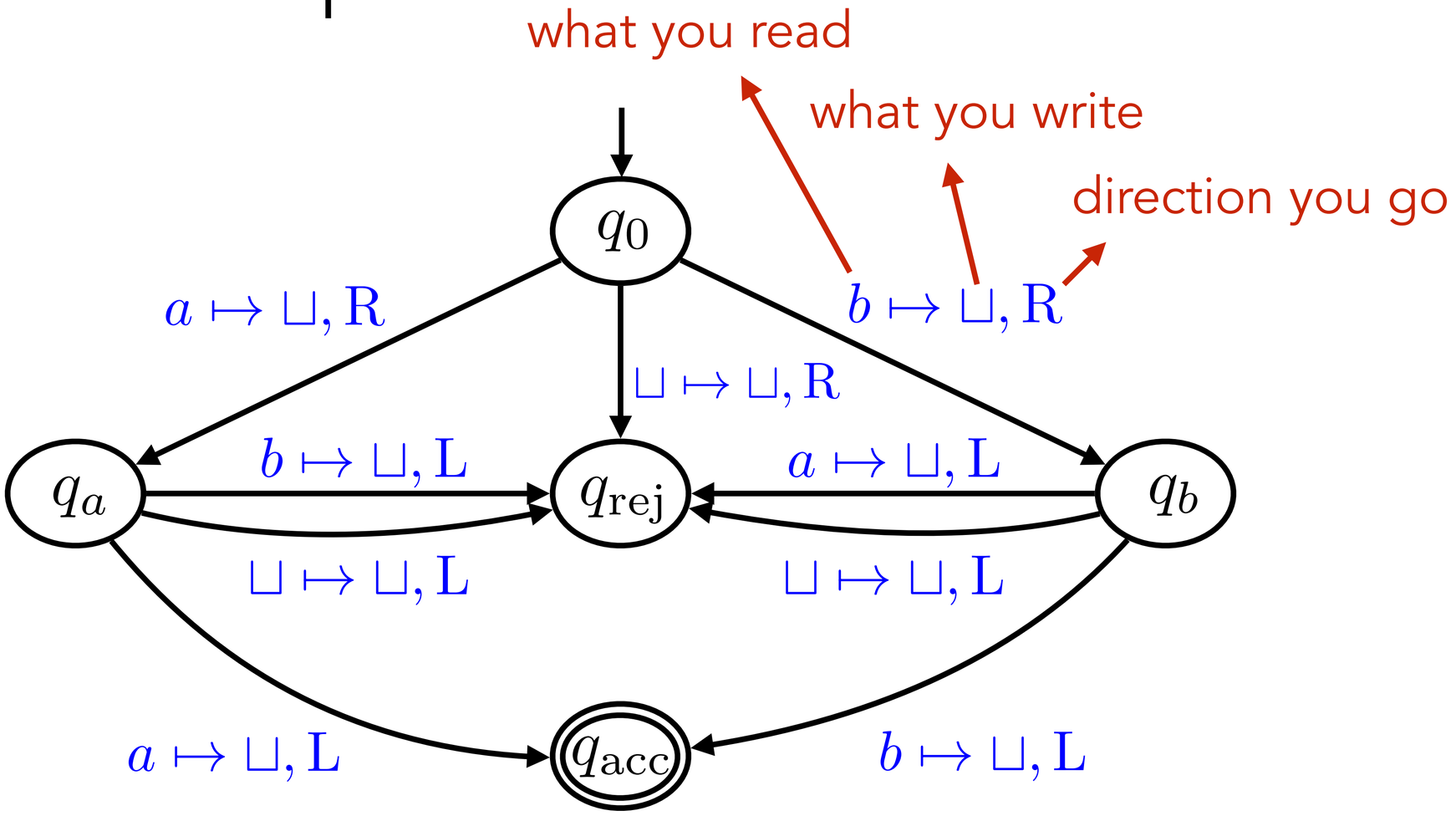


if you are in state **q_0**
and you read *a*,
then write **blank**
and move **Right**

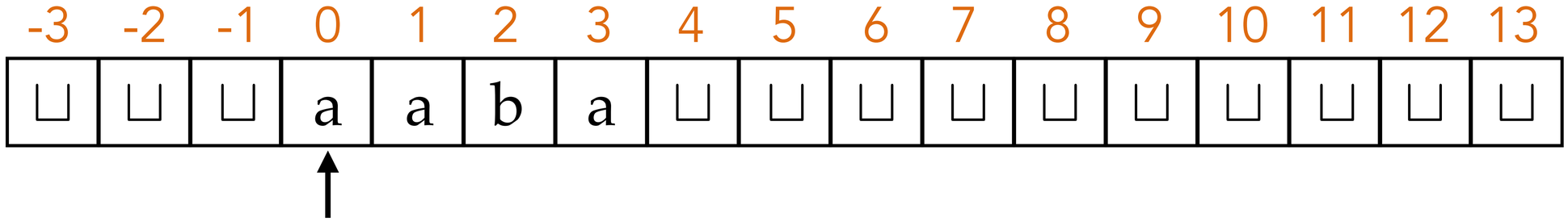


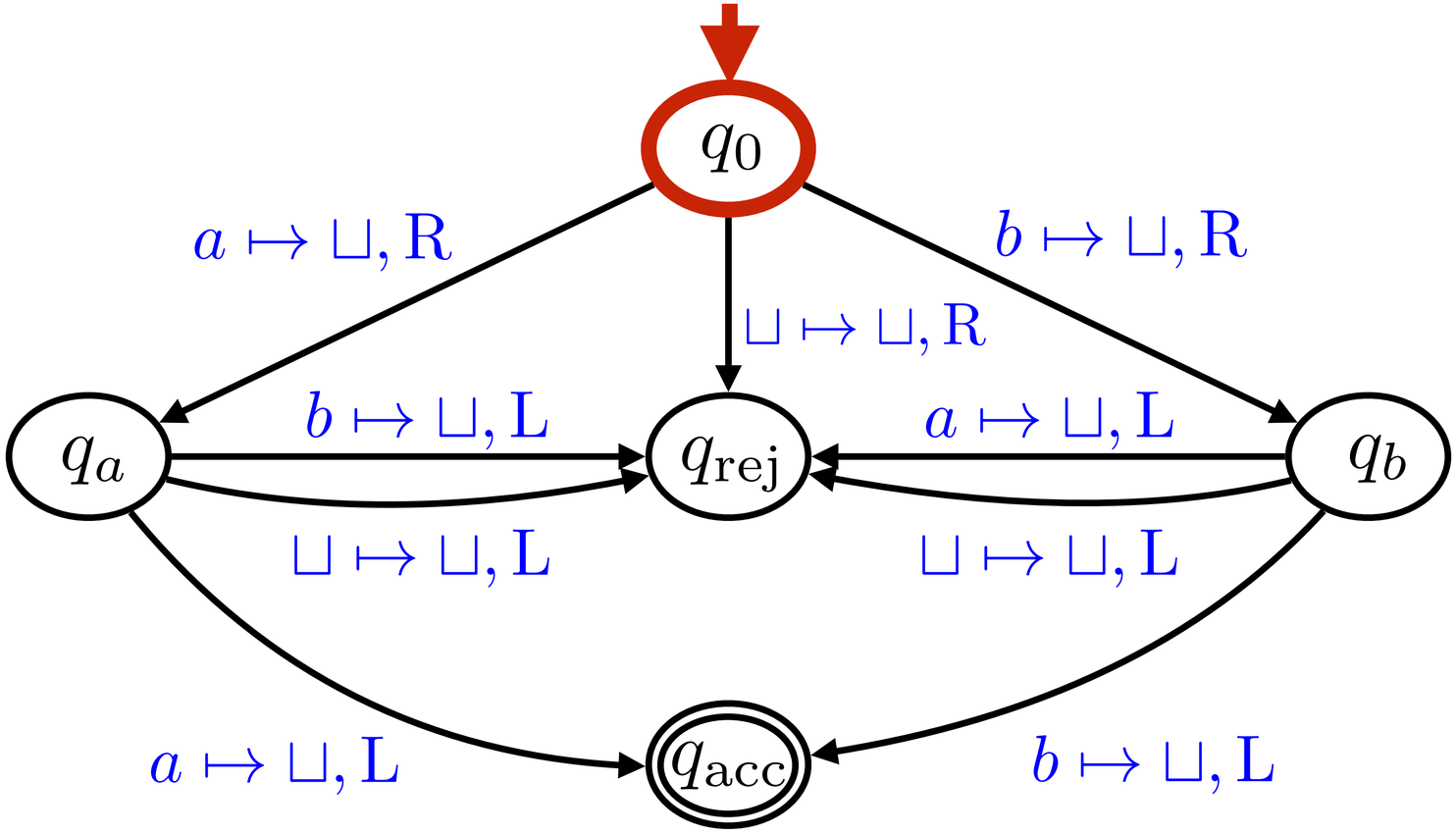
Turing machine state diagram



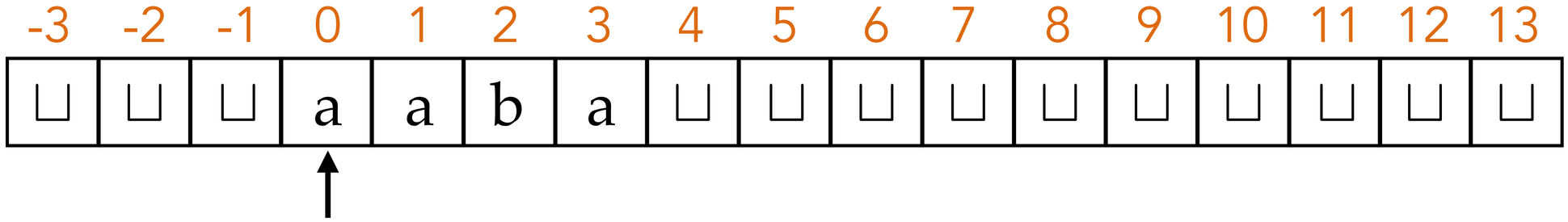


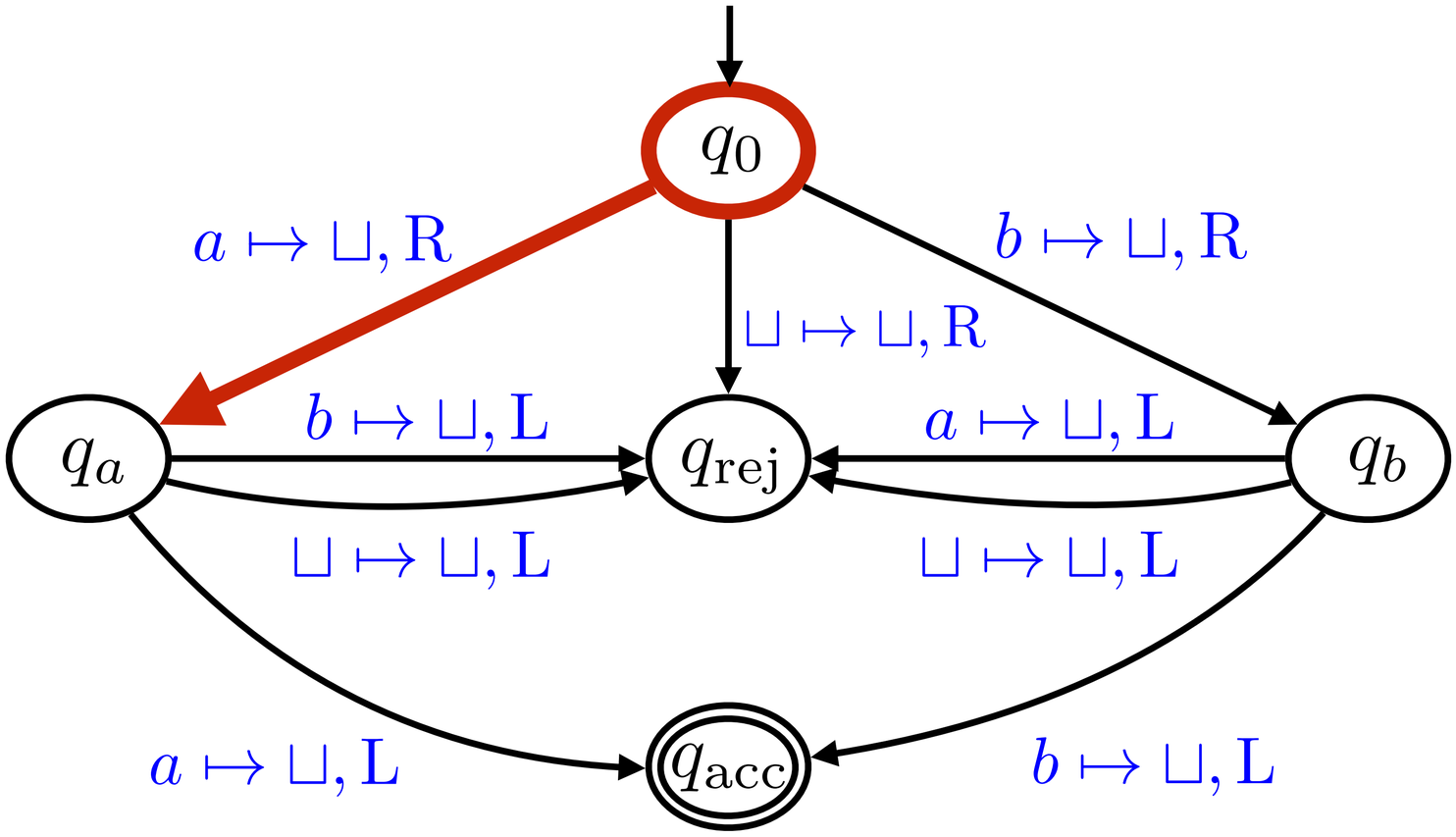
Turing machine simulation example



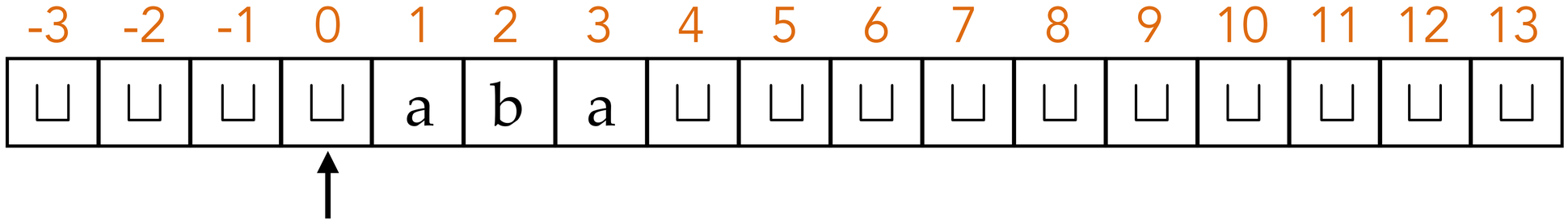


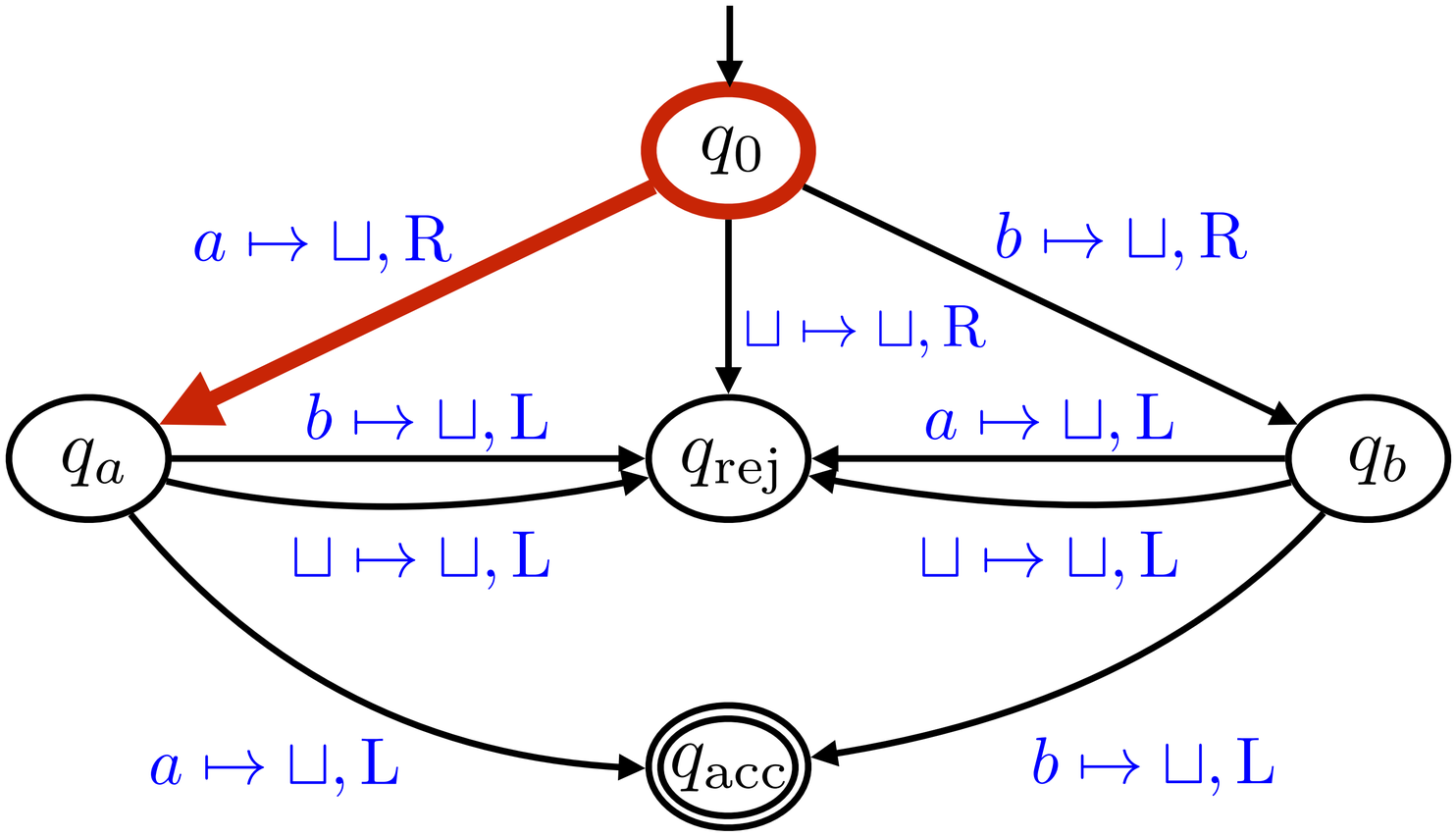
Turing machine simulation example



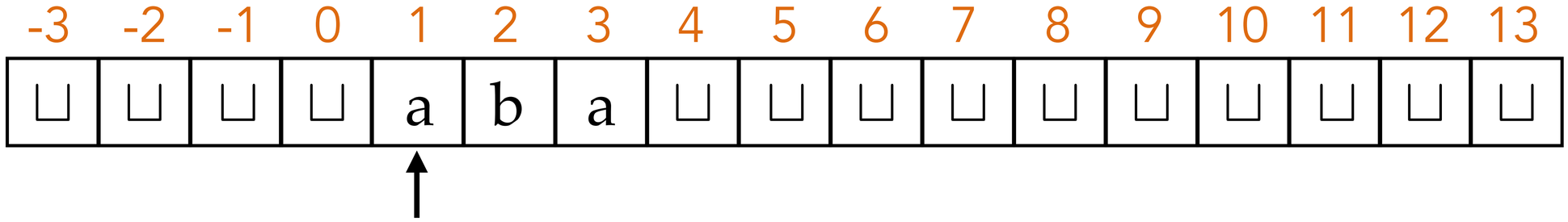


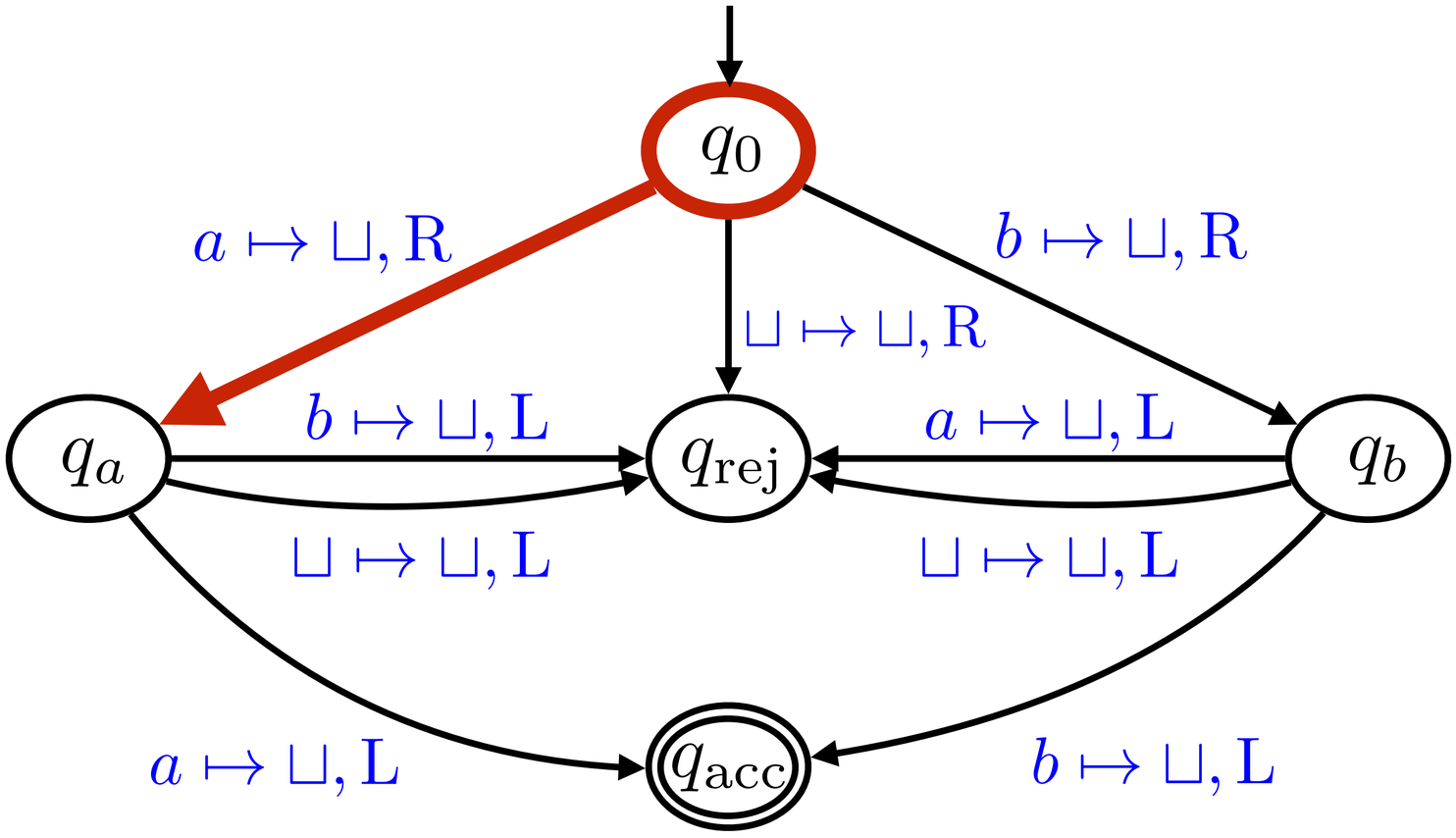
Turing machine simulation example



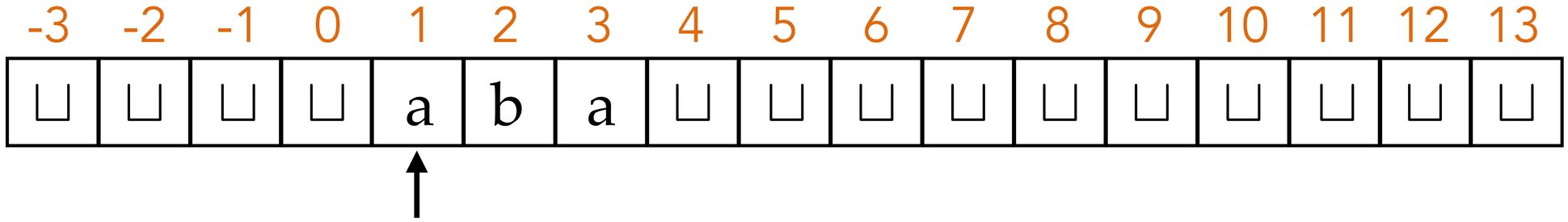


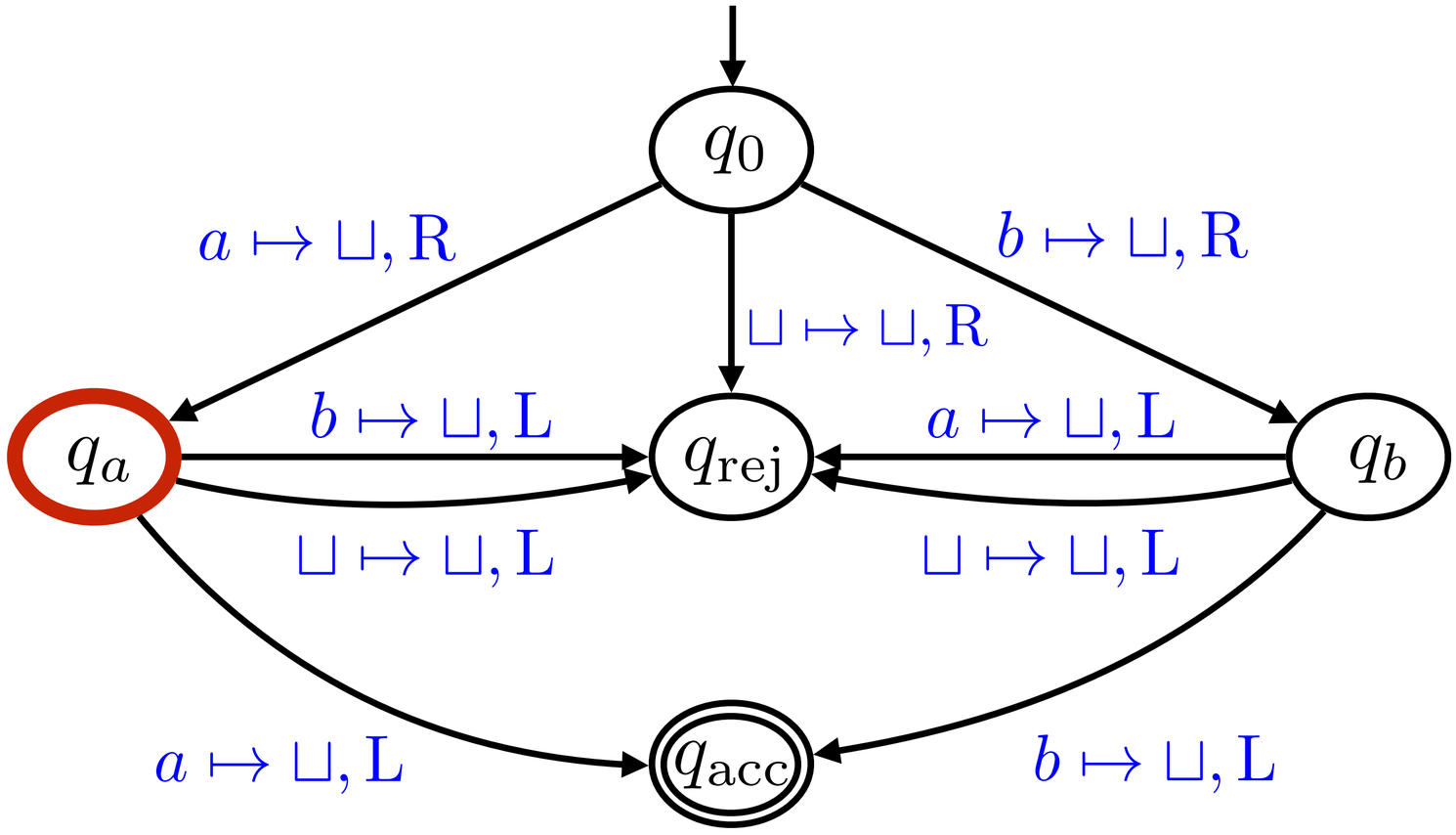
Turing machine simulation example



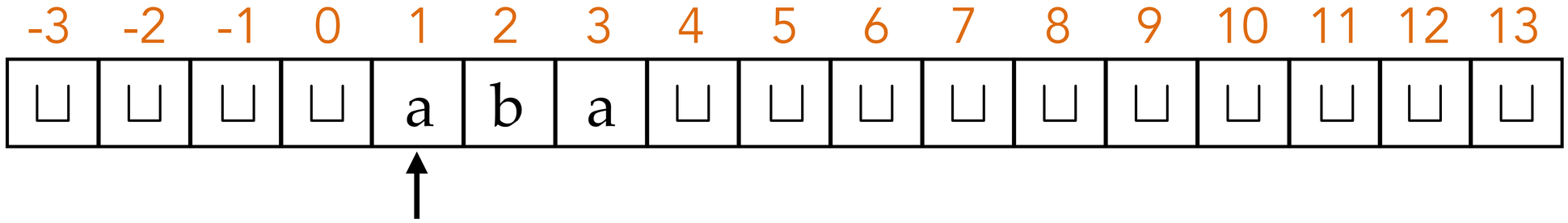


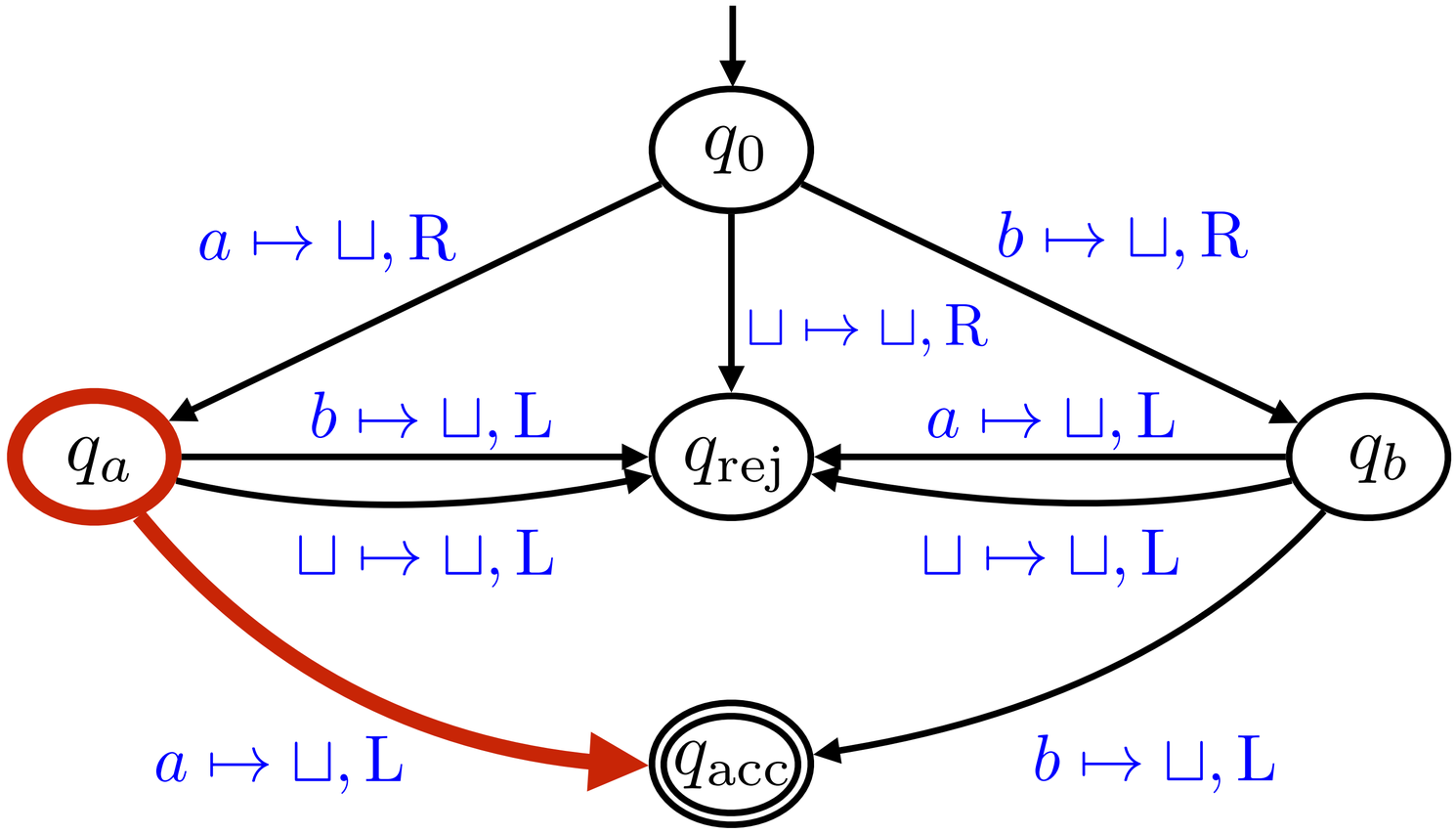
Turing machine simulation example



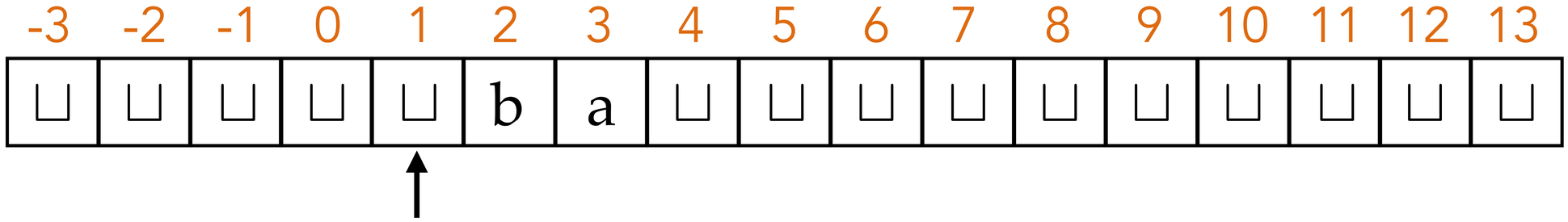


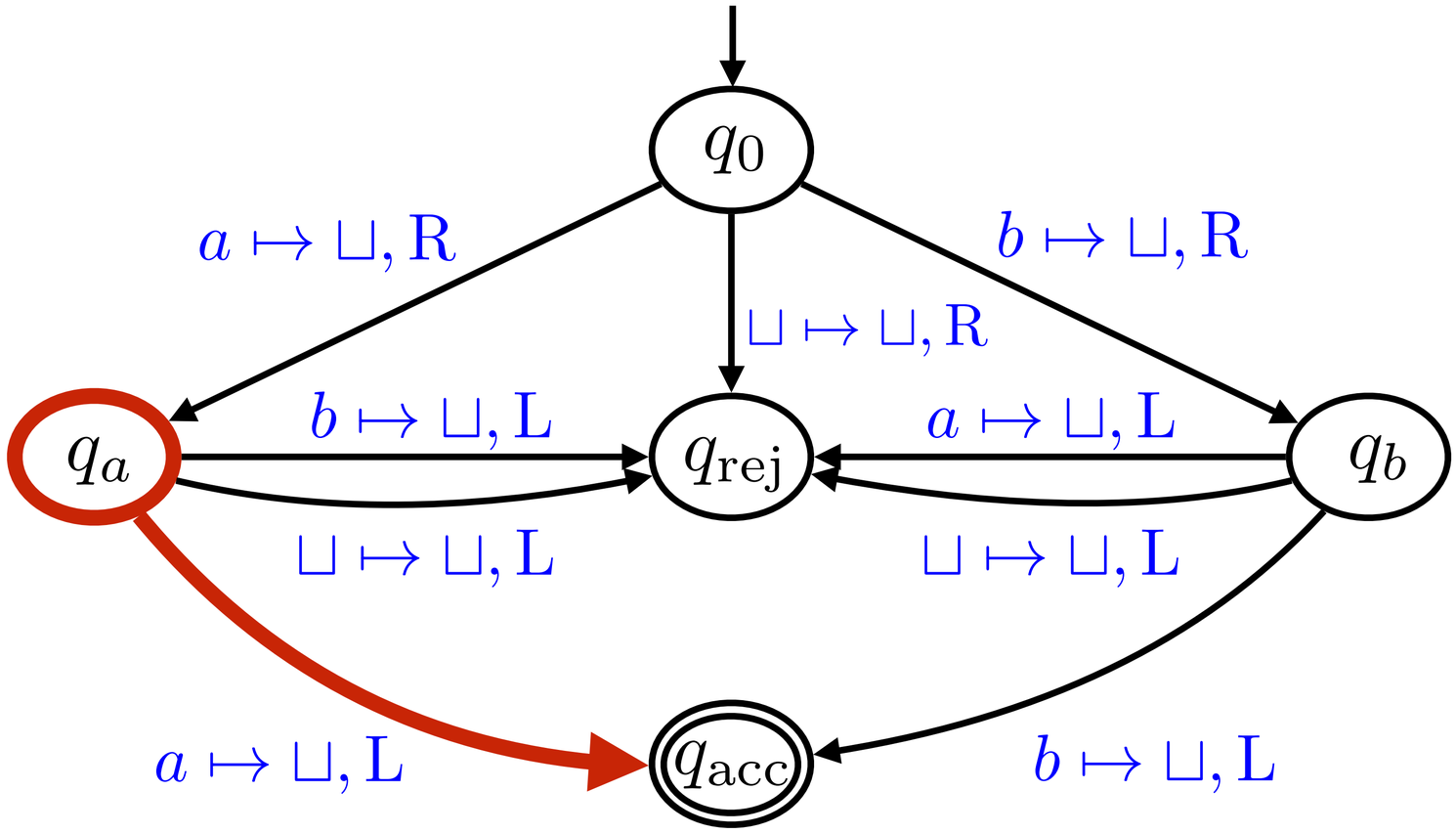
Turing machine simulation example



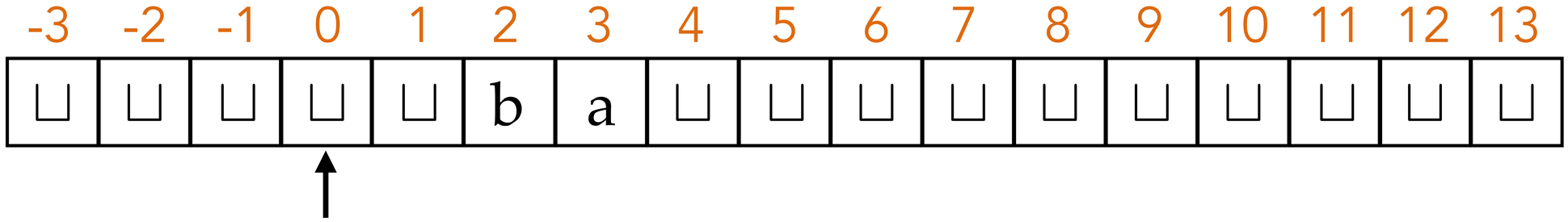


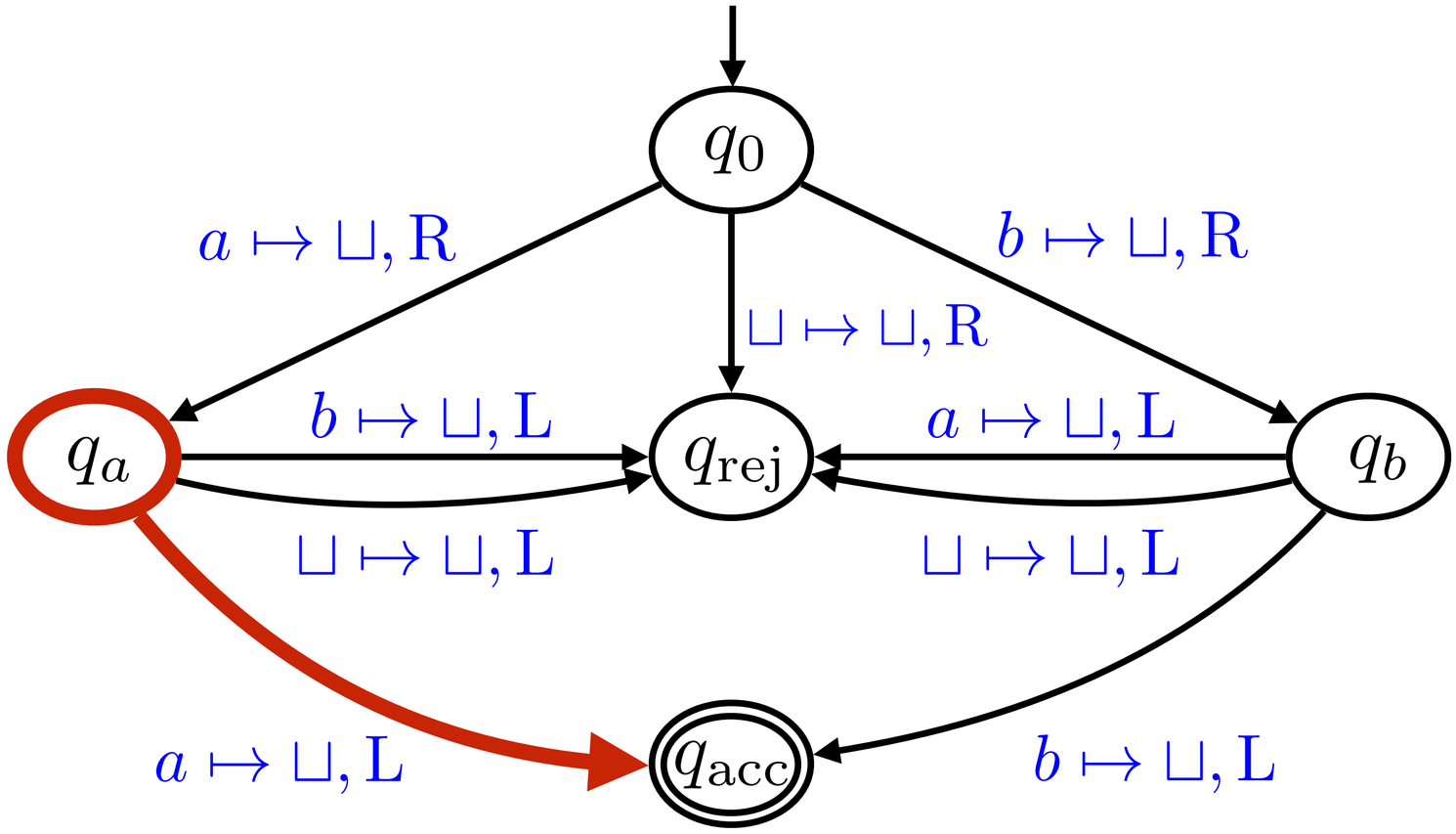
Turing machine simulation example



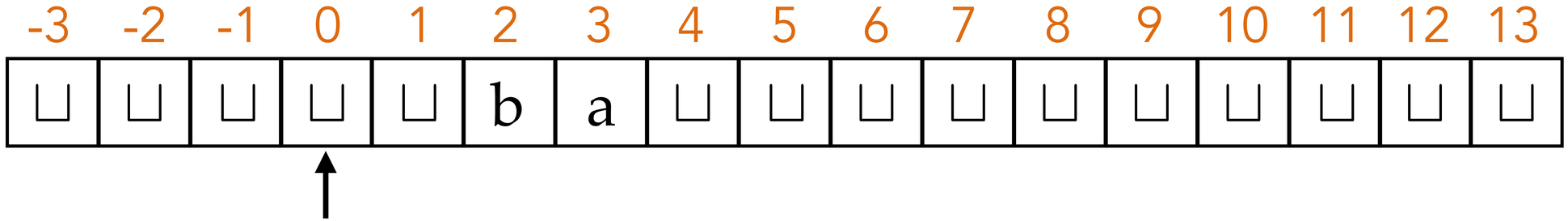


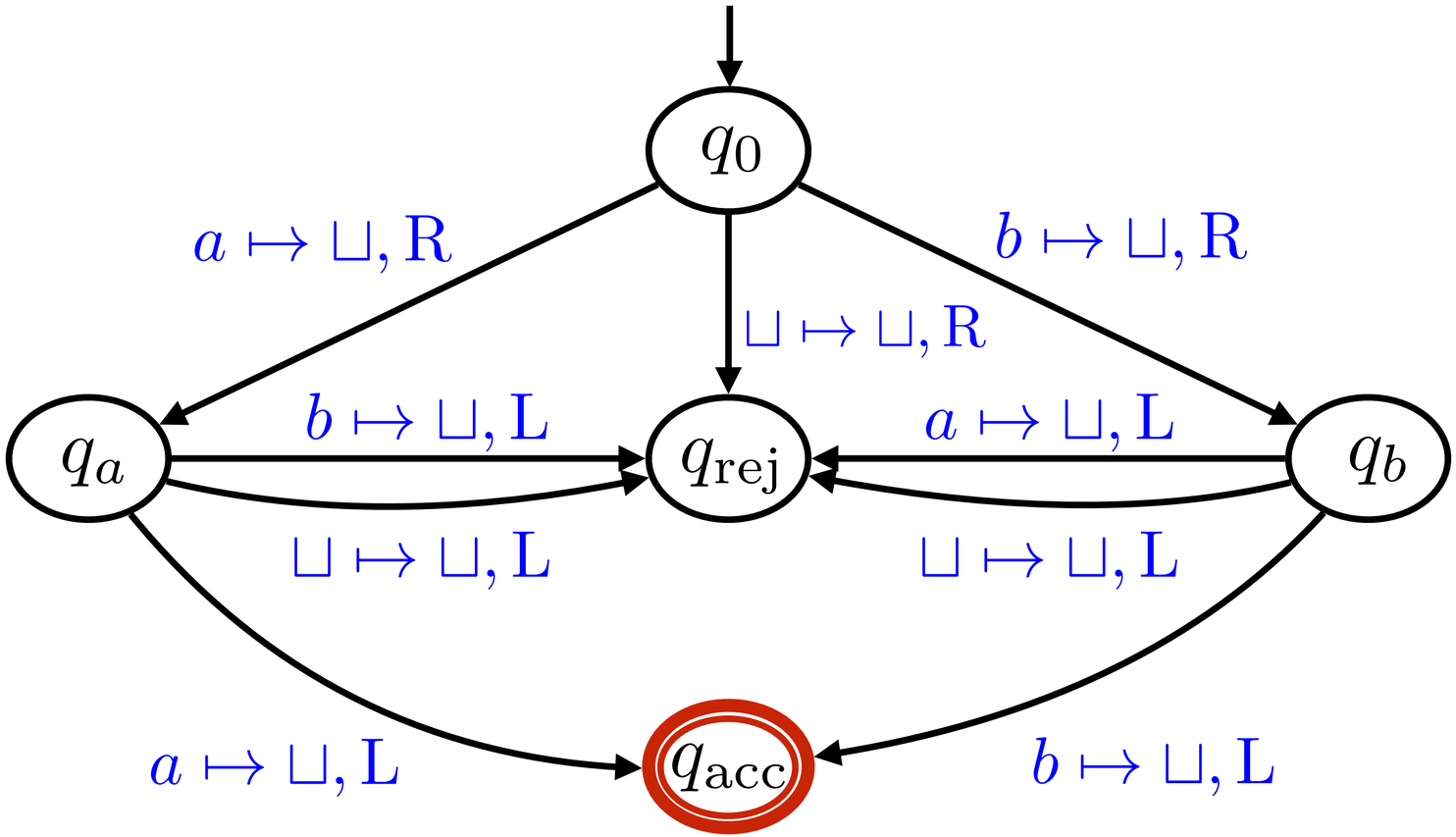
Turing machine simulation example



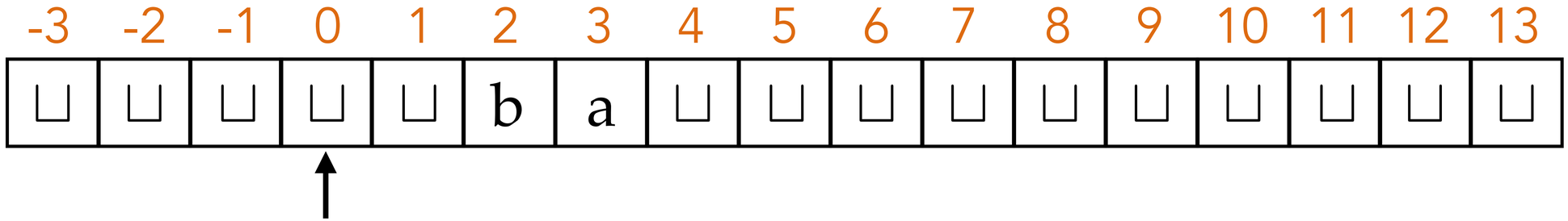


Turing machine simulation example

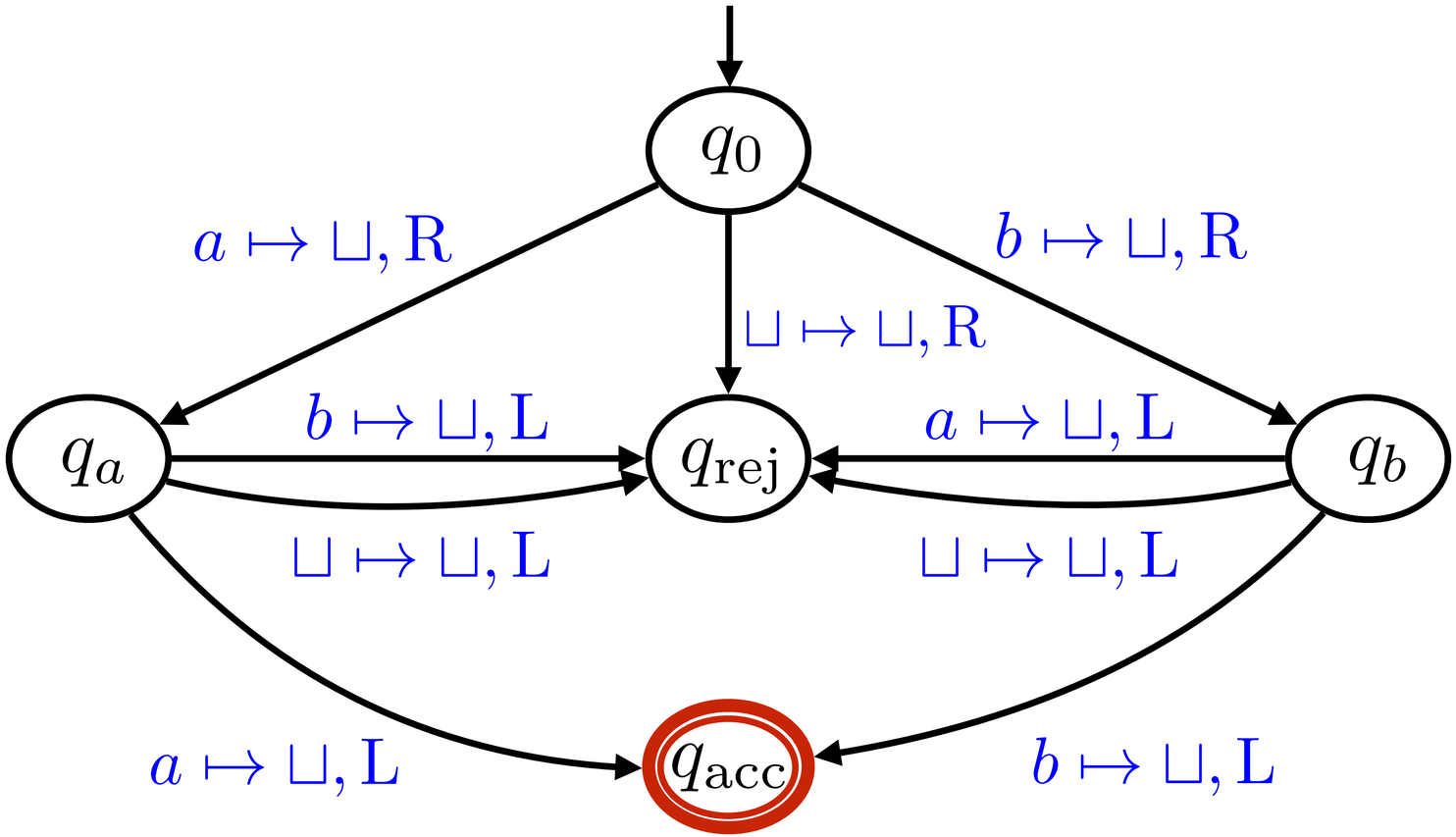




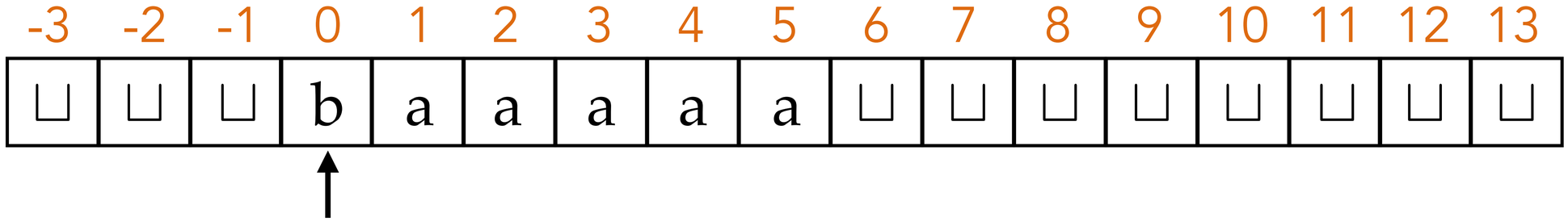
Turing machine simulation example

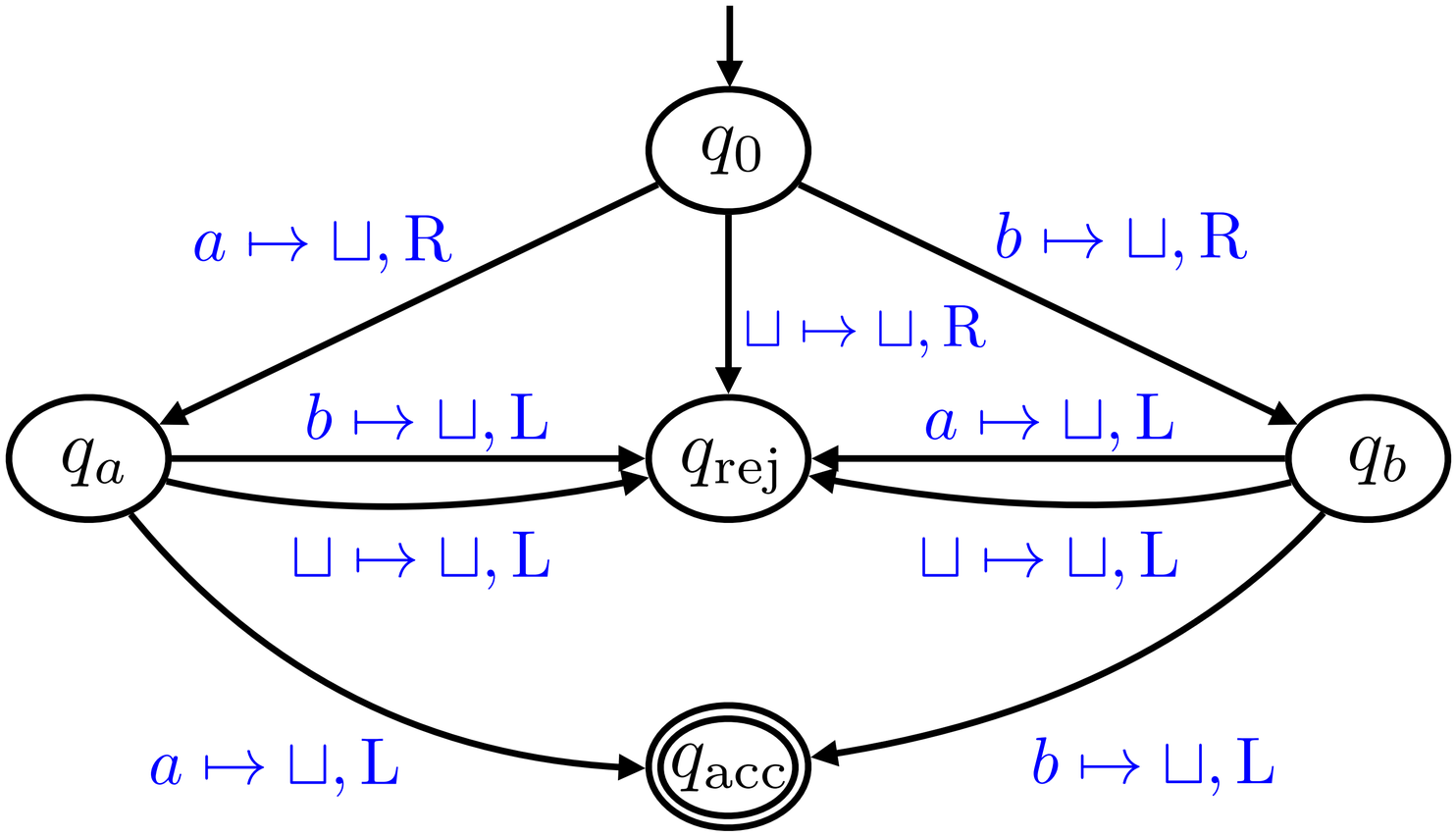


Decision: Accept

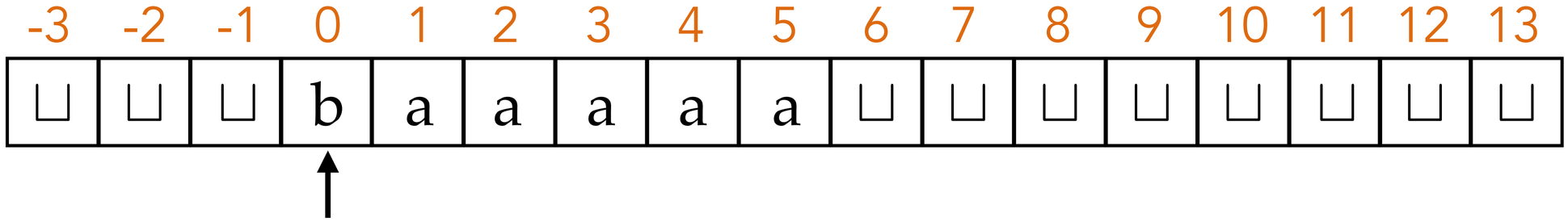


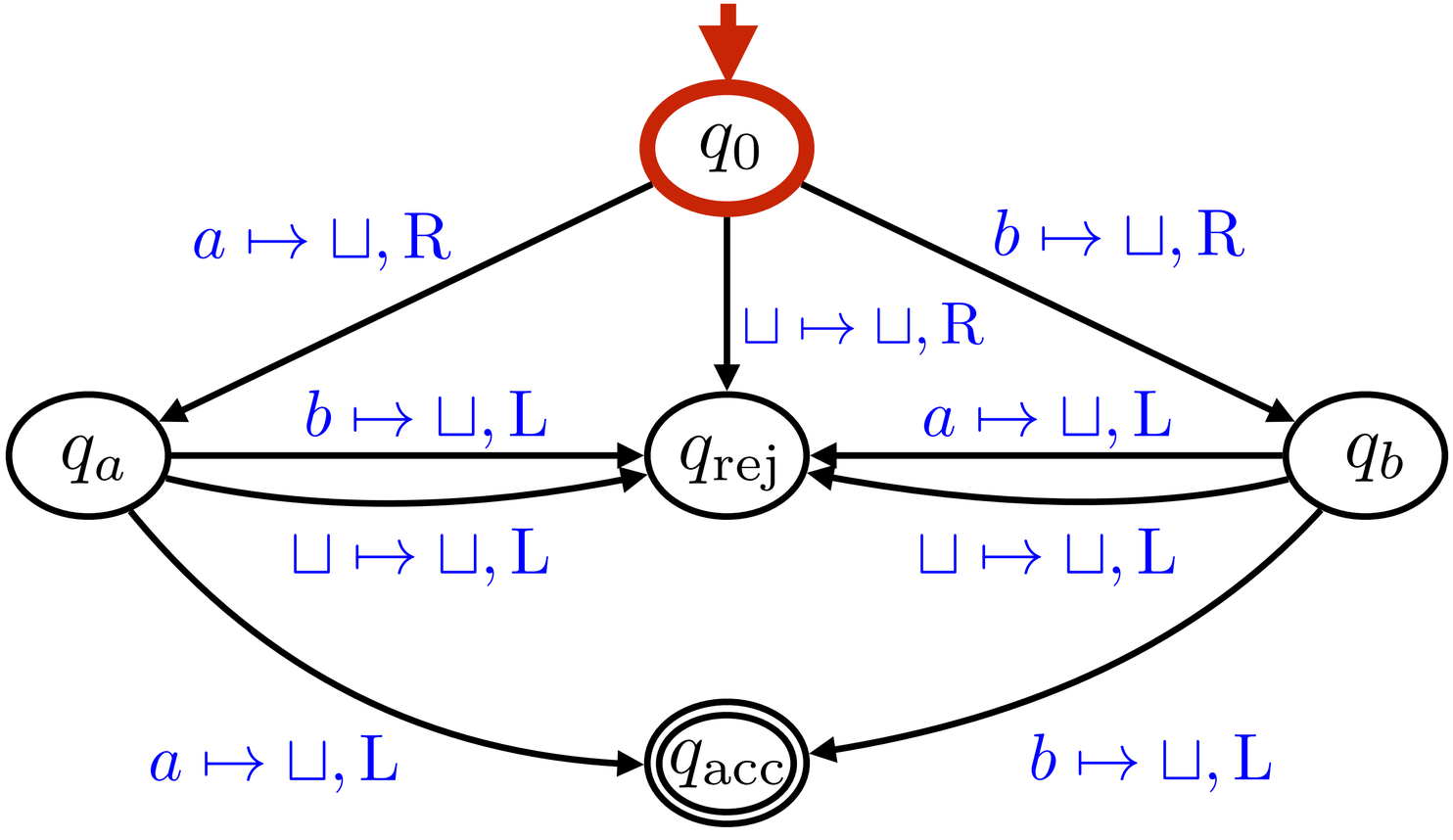
Turing machine simulation example



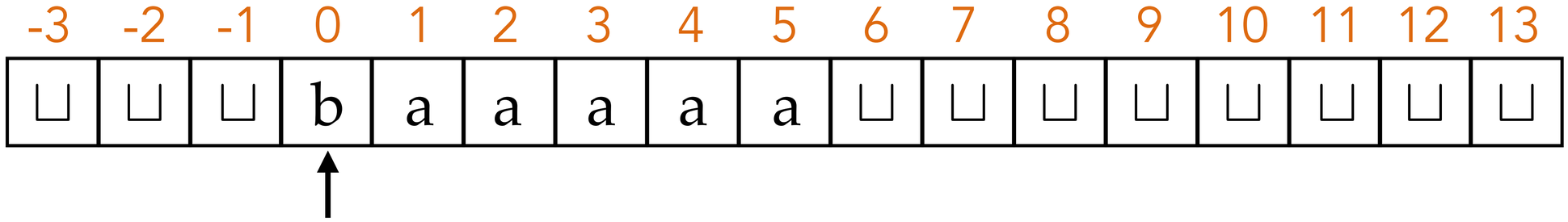


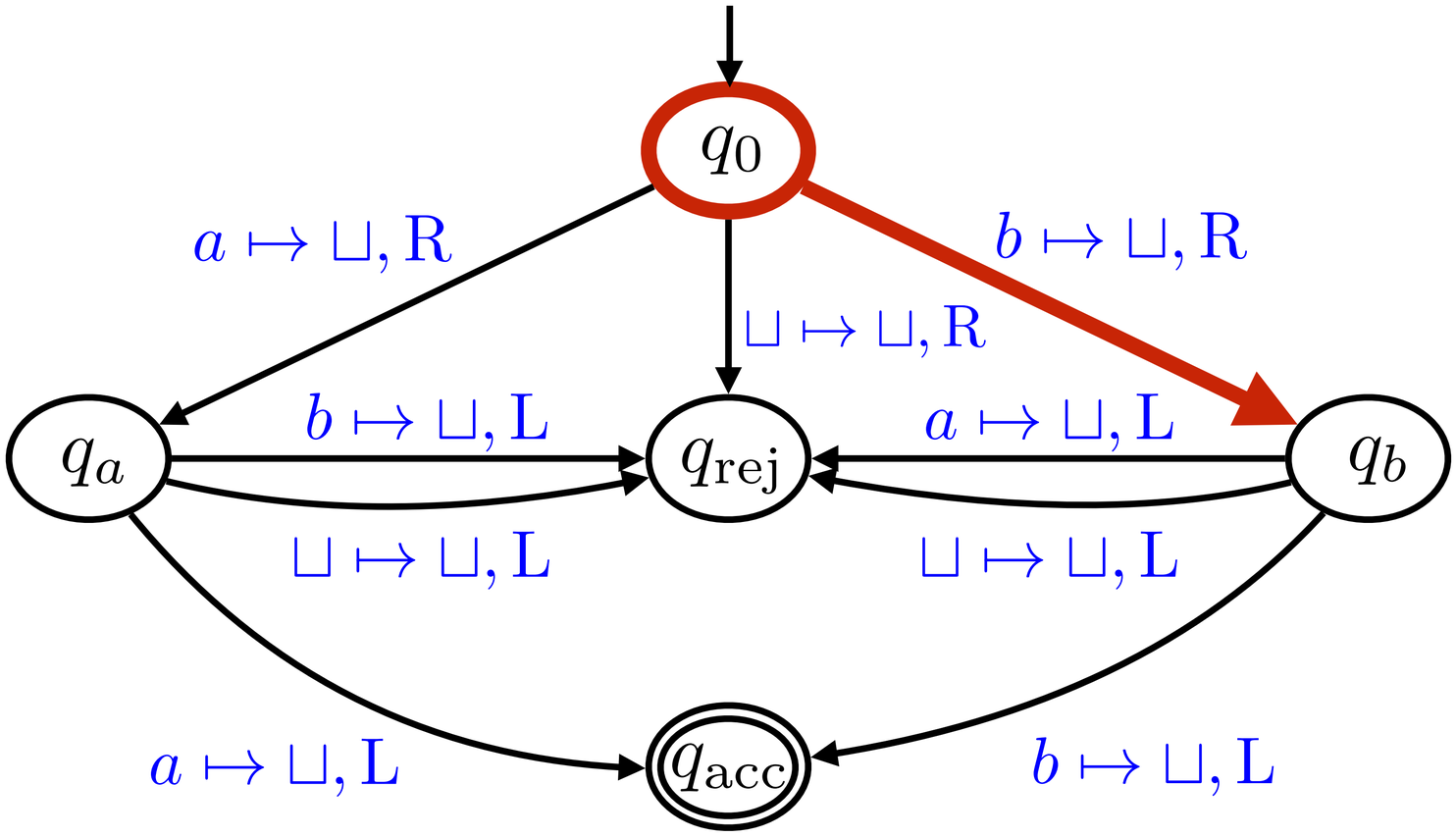
Turing machine simulation example



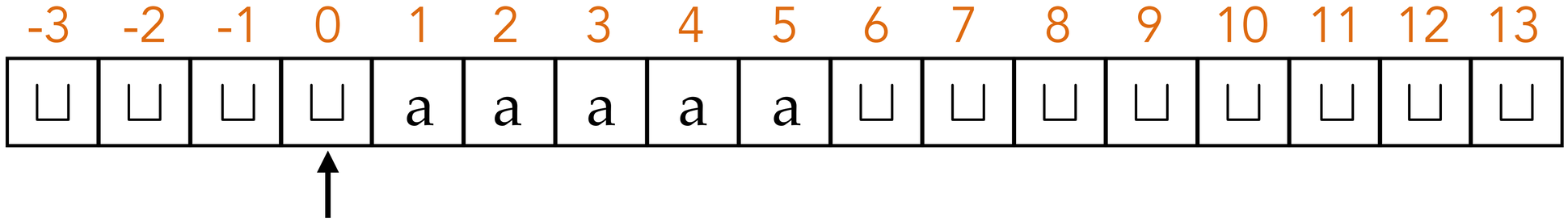


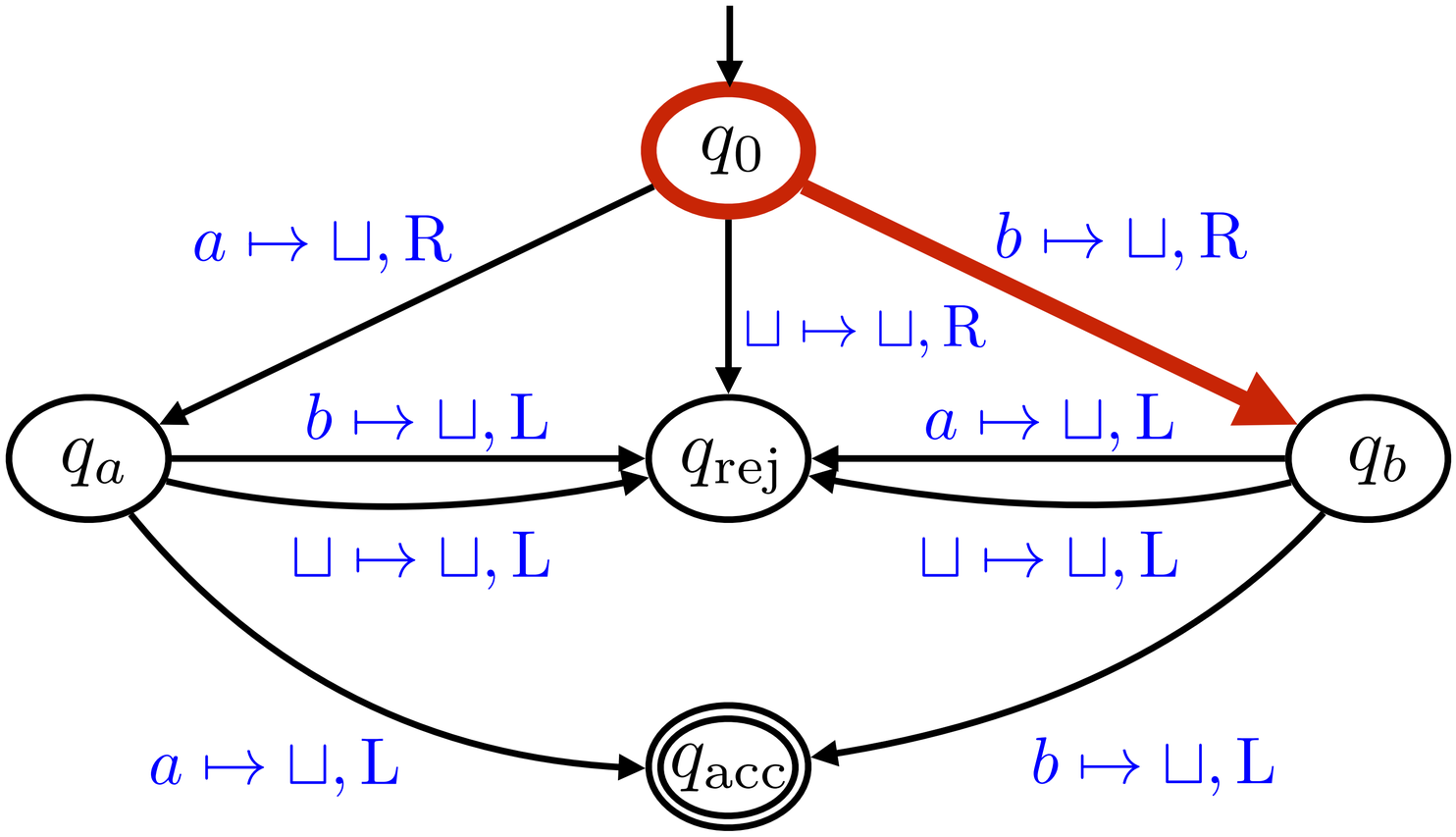
Turing machine simulation example



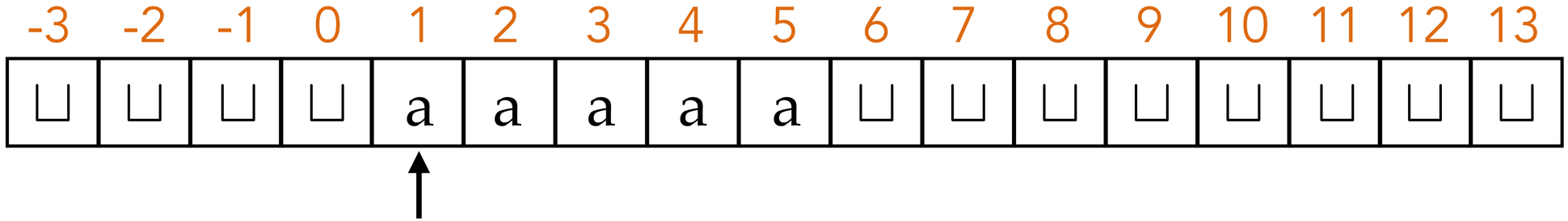


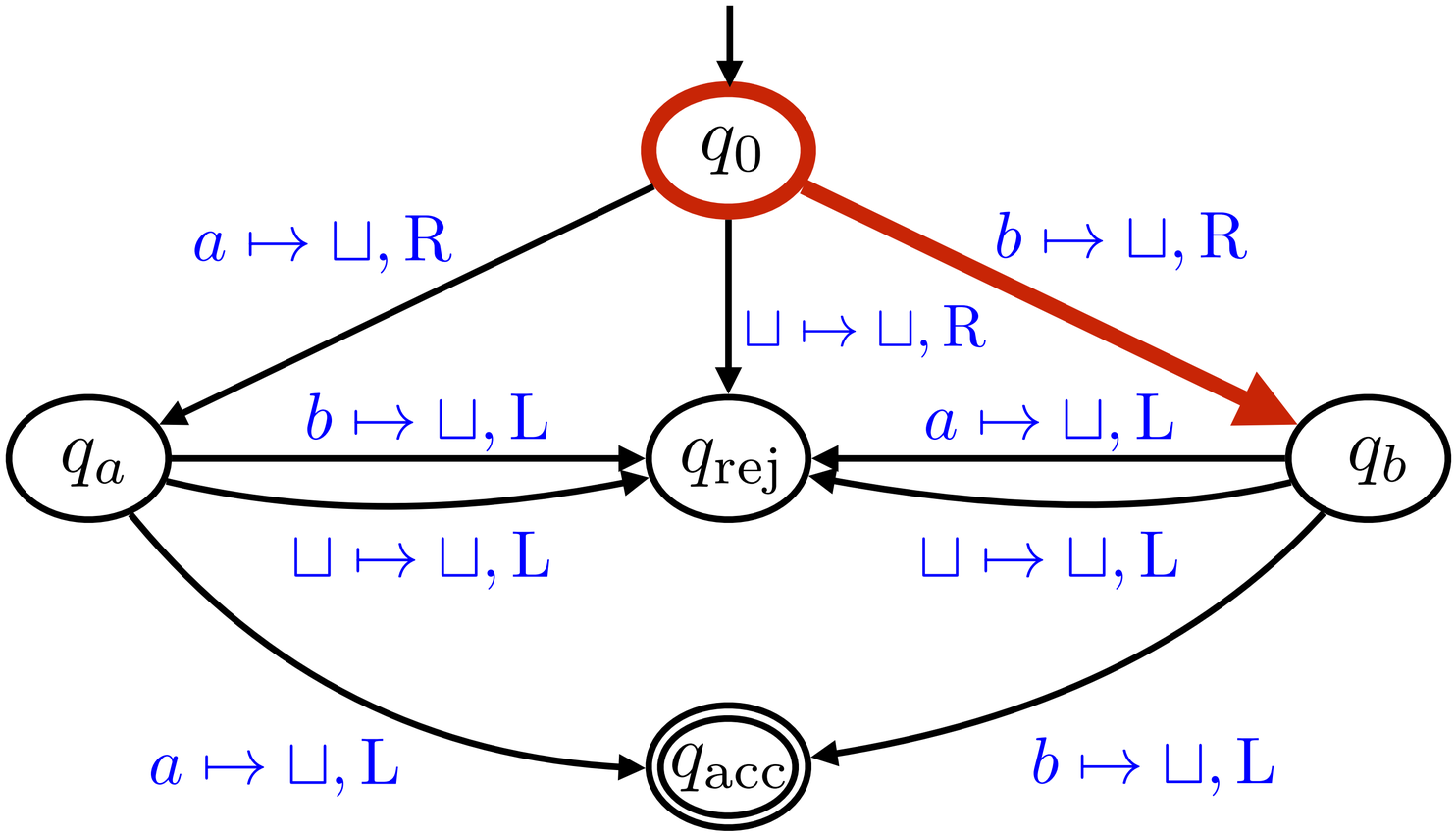
Turing machine simulation example



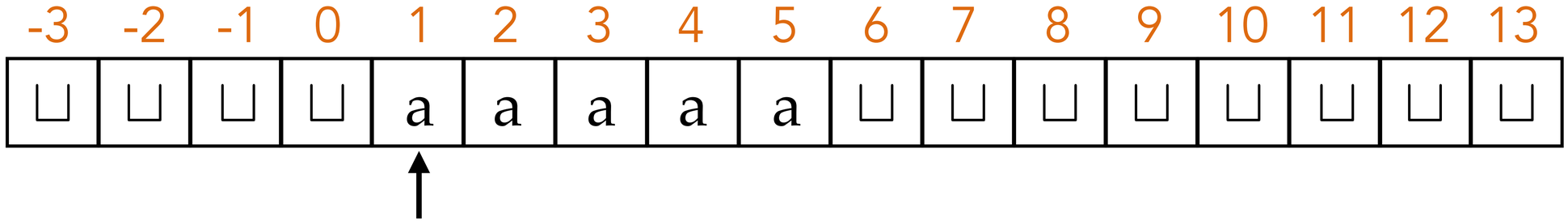


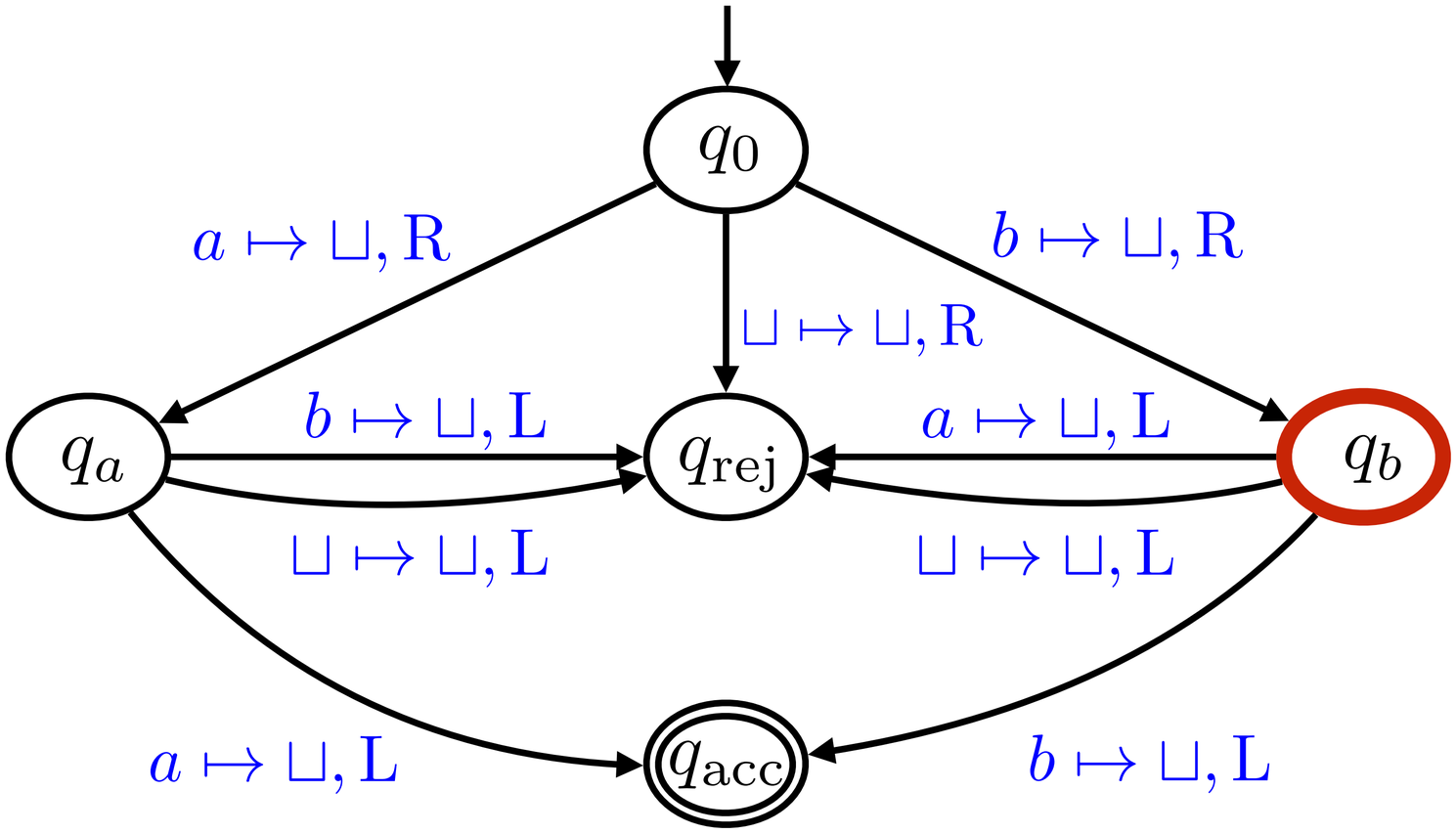
Turing machine simulation example



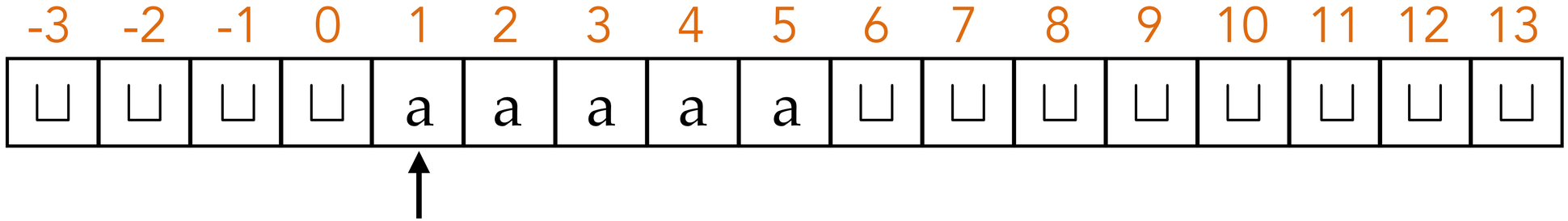


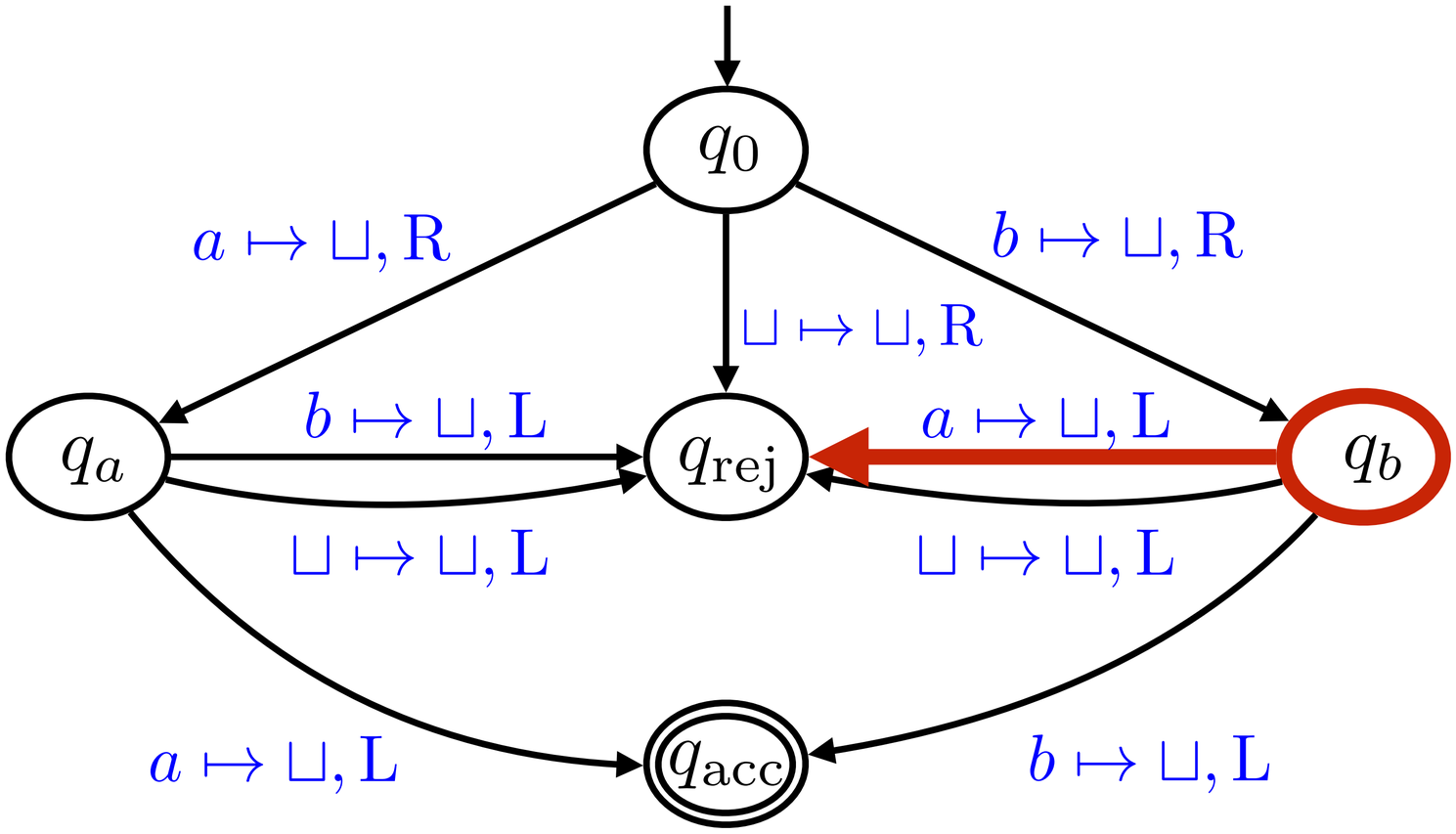
Turing machine simulation example



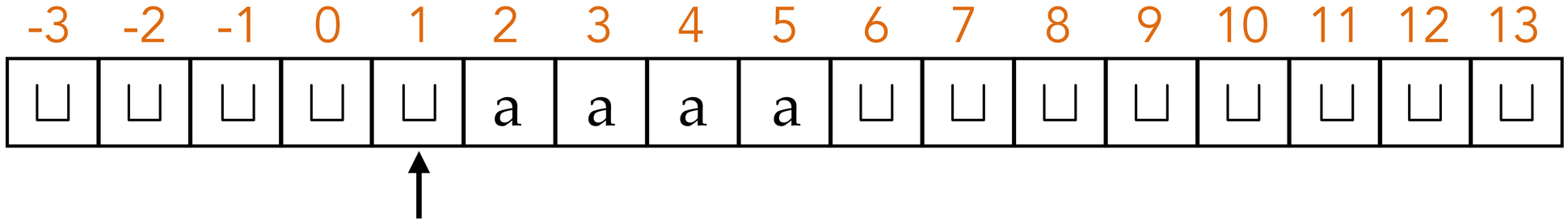


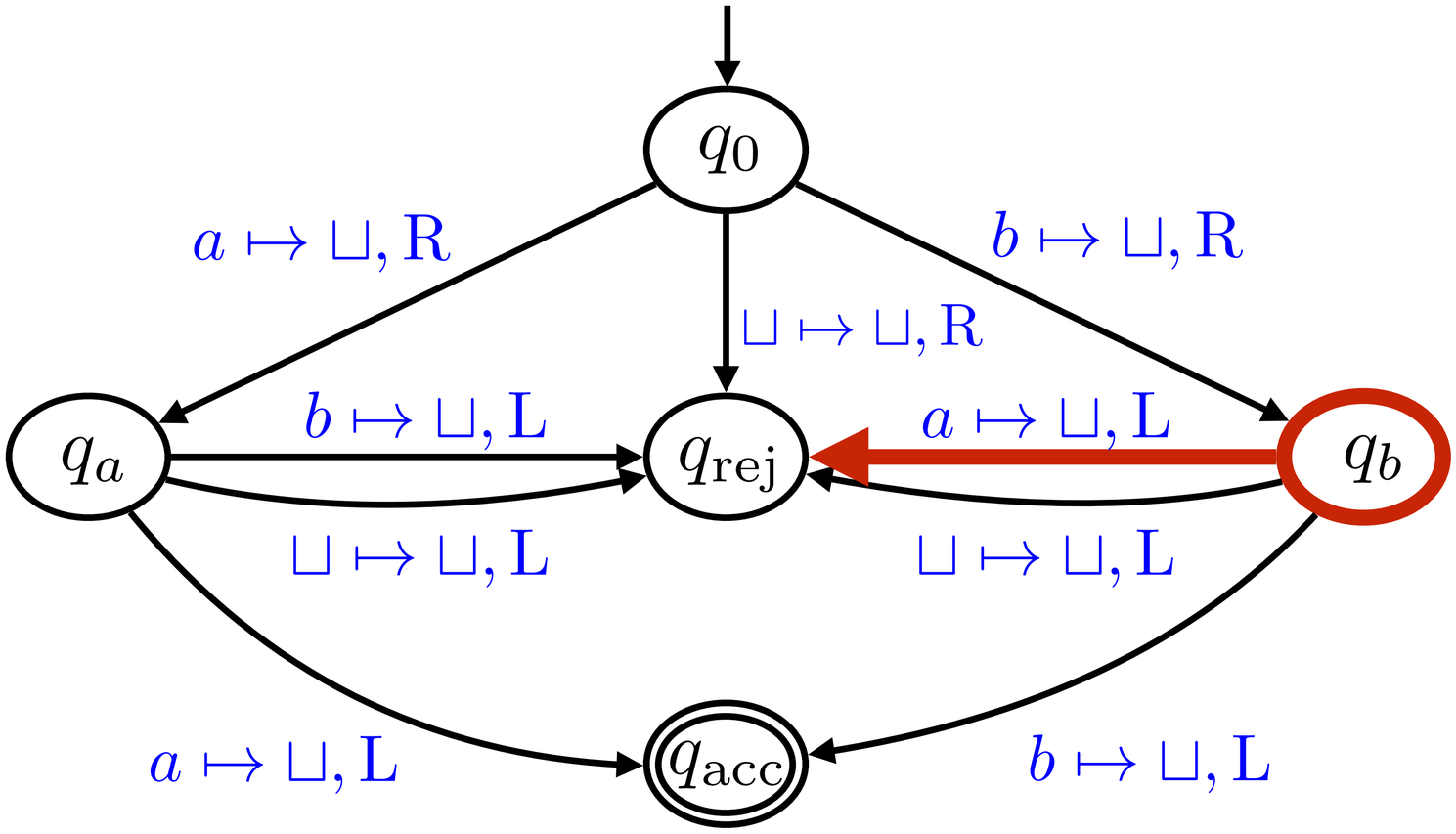
Turing machine simulation example



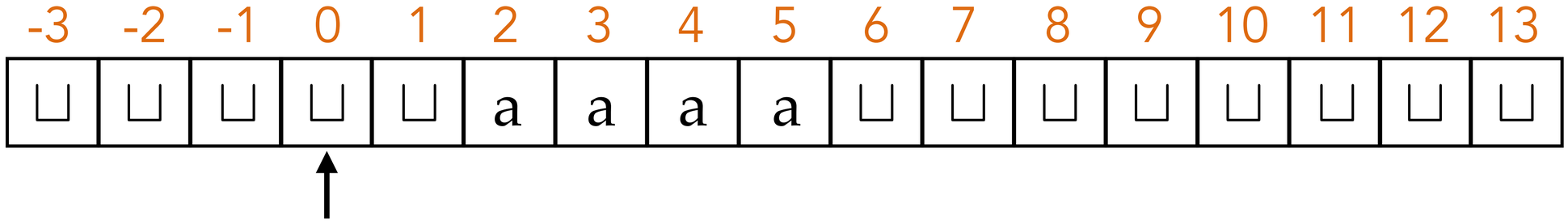


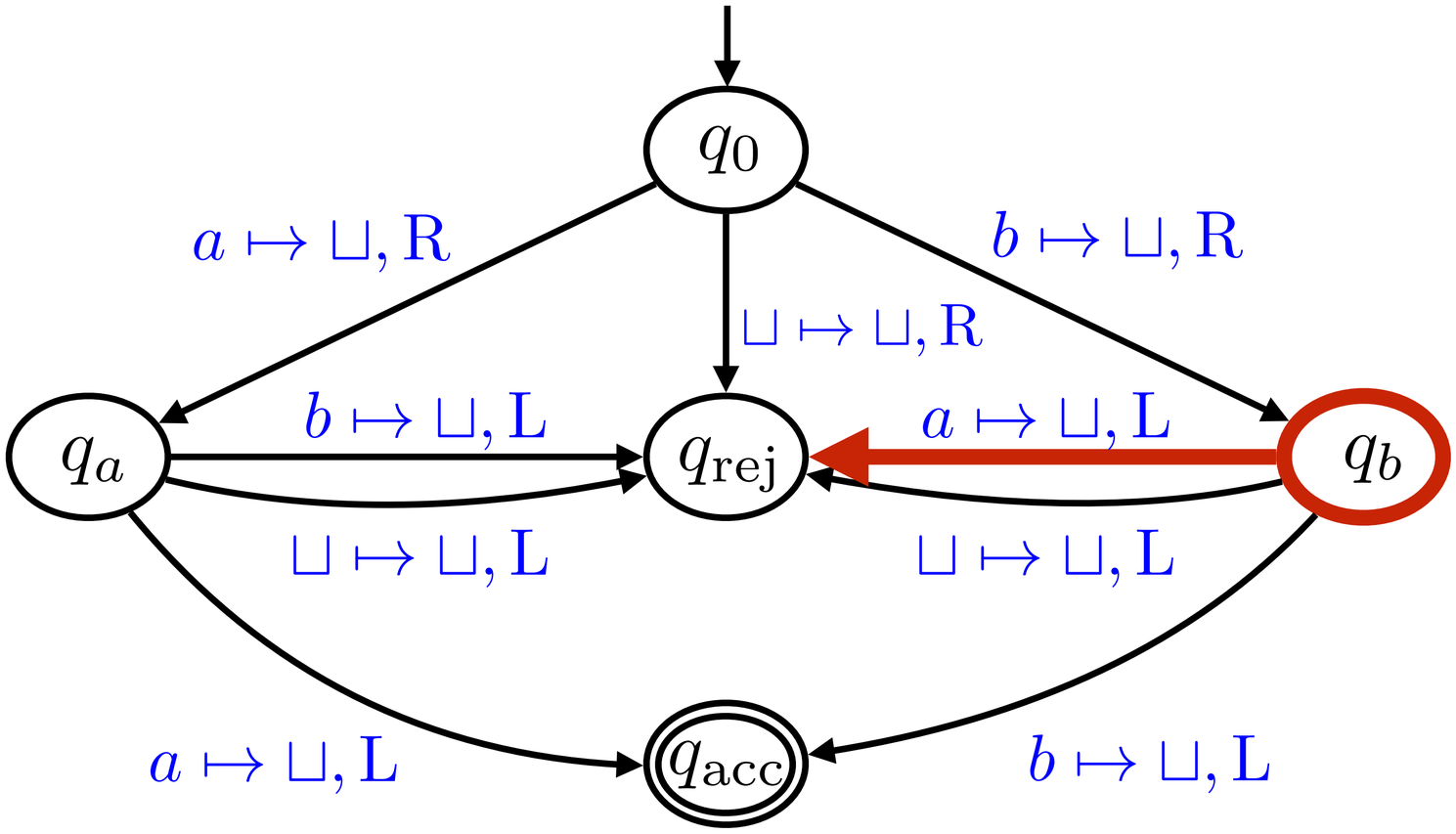
Turing machine simulation example



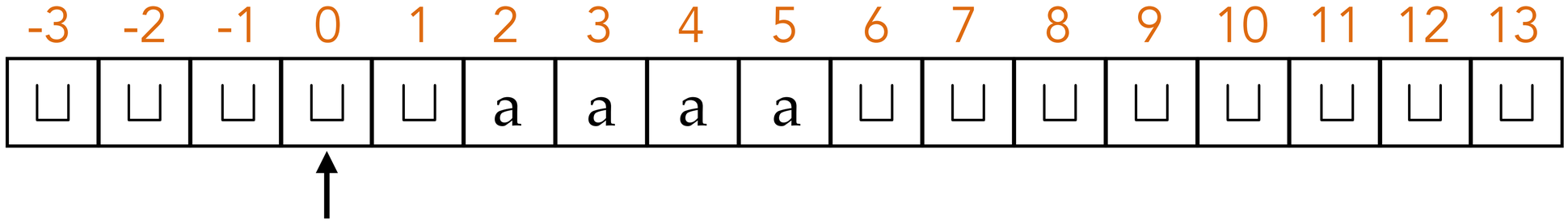


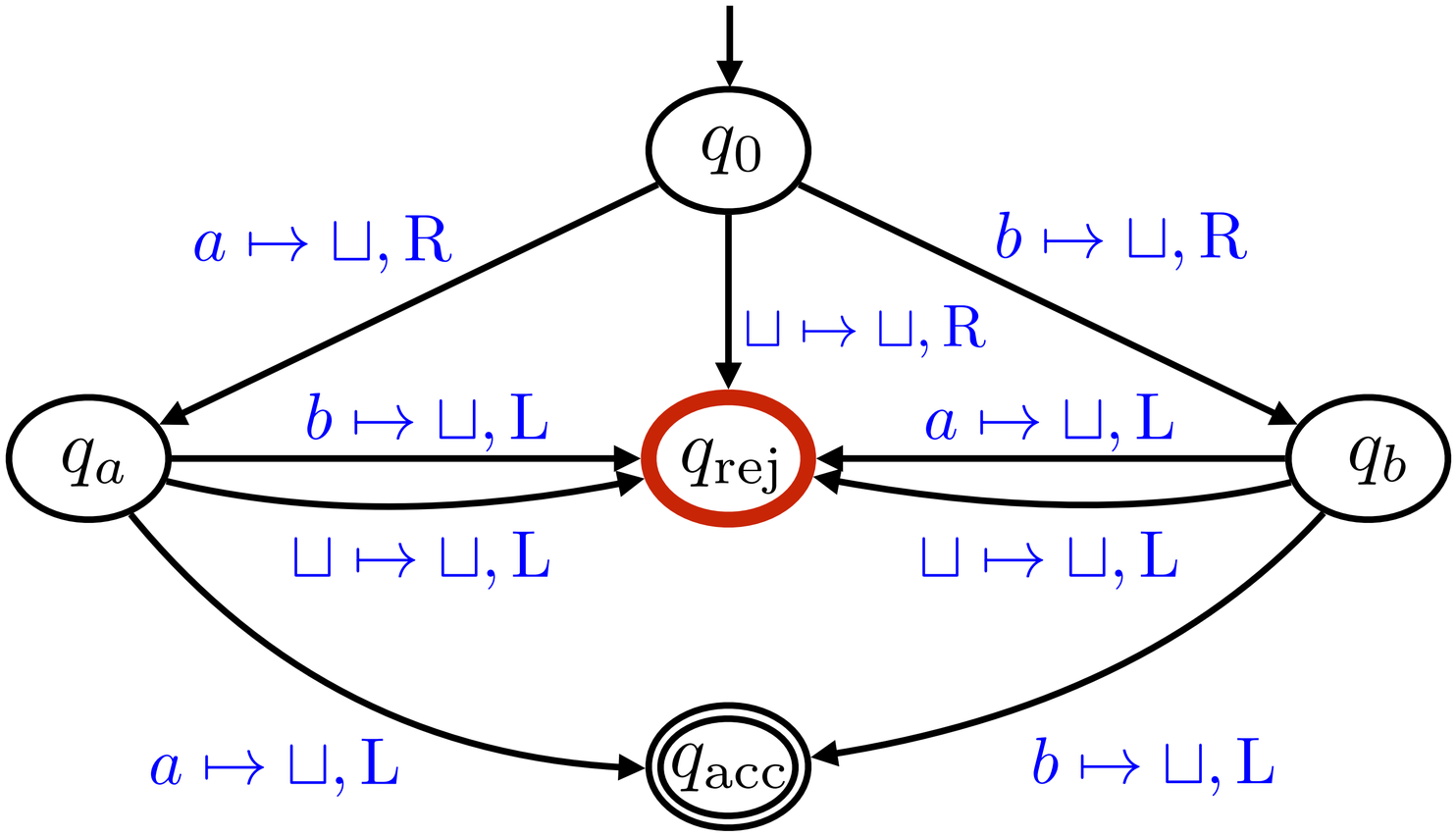
Turing machine simulation example



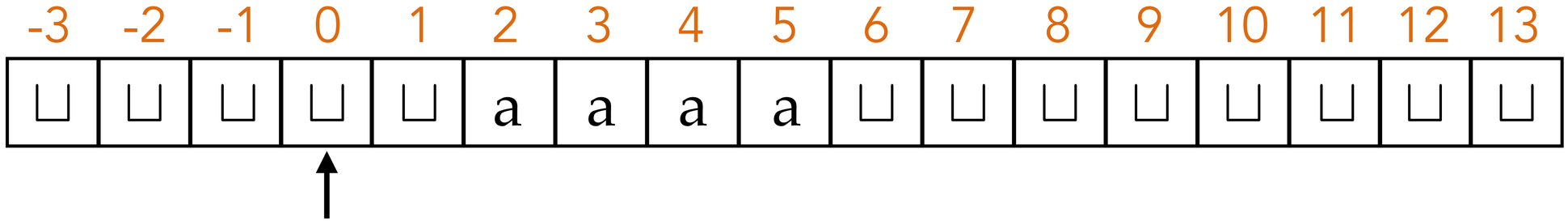


Turing machine simulation example

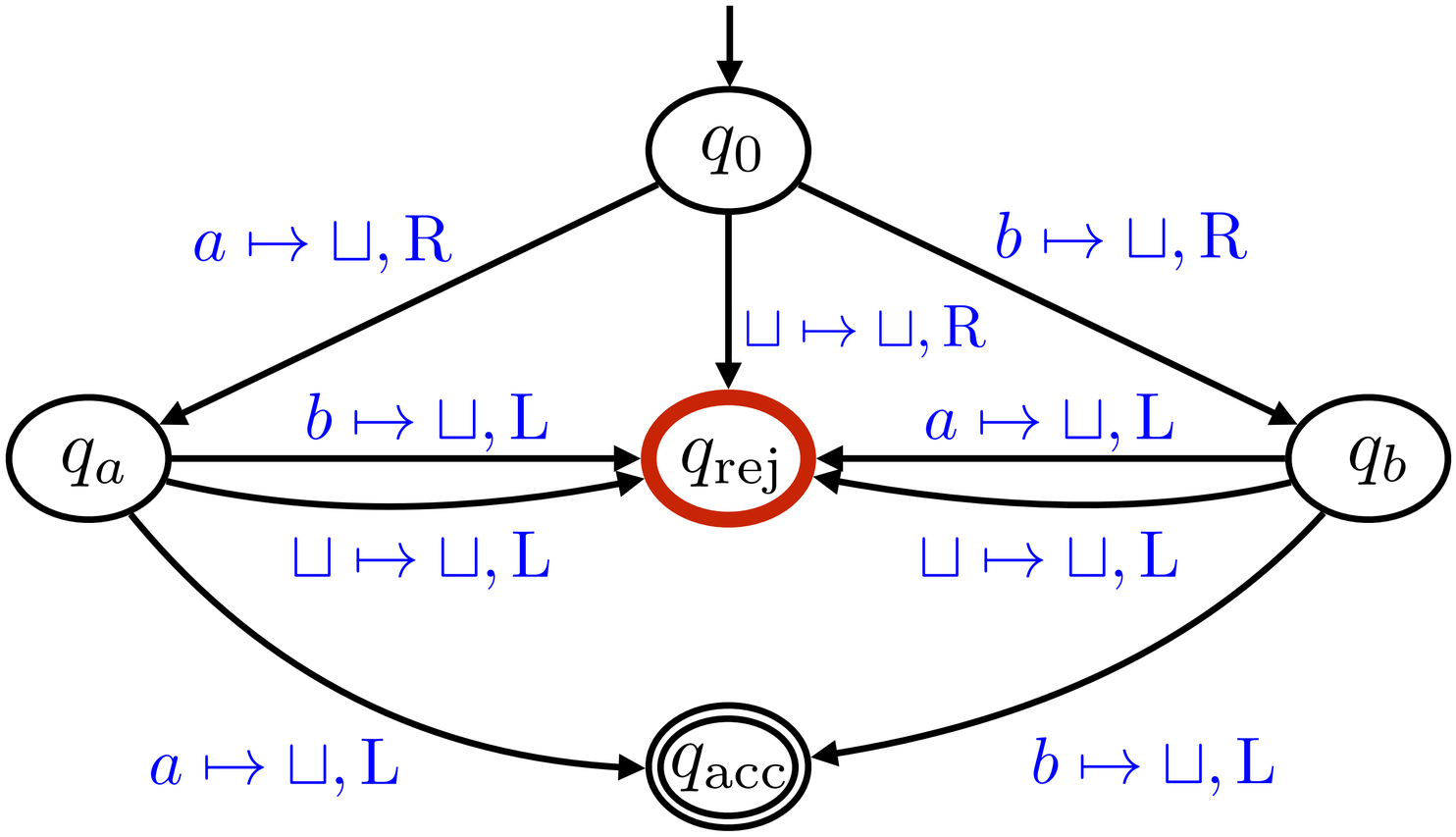




Turing machine simulation example



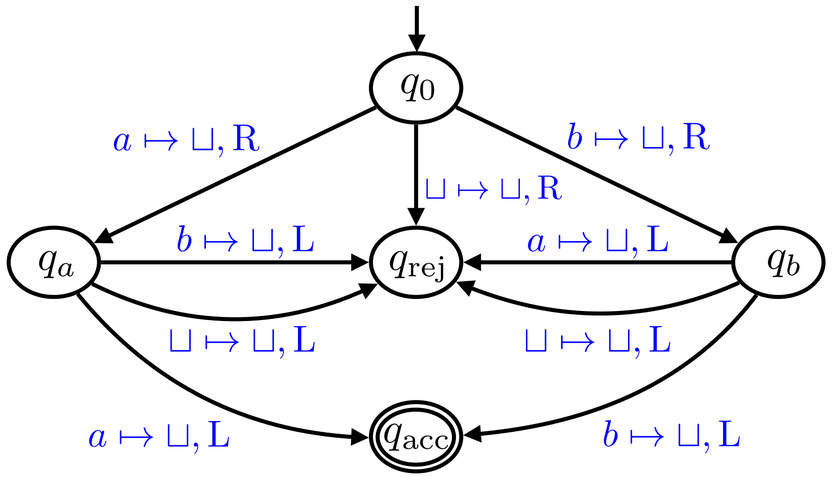
Decision: Reject



TM as a programming language

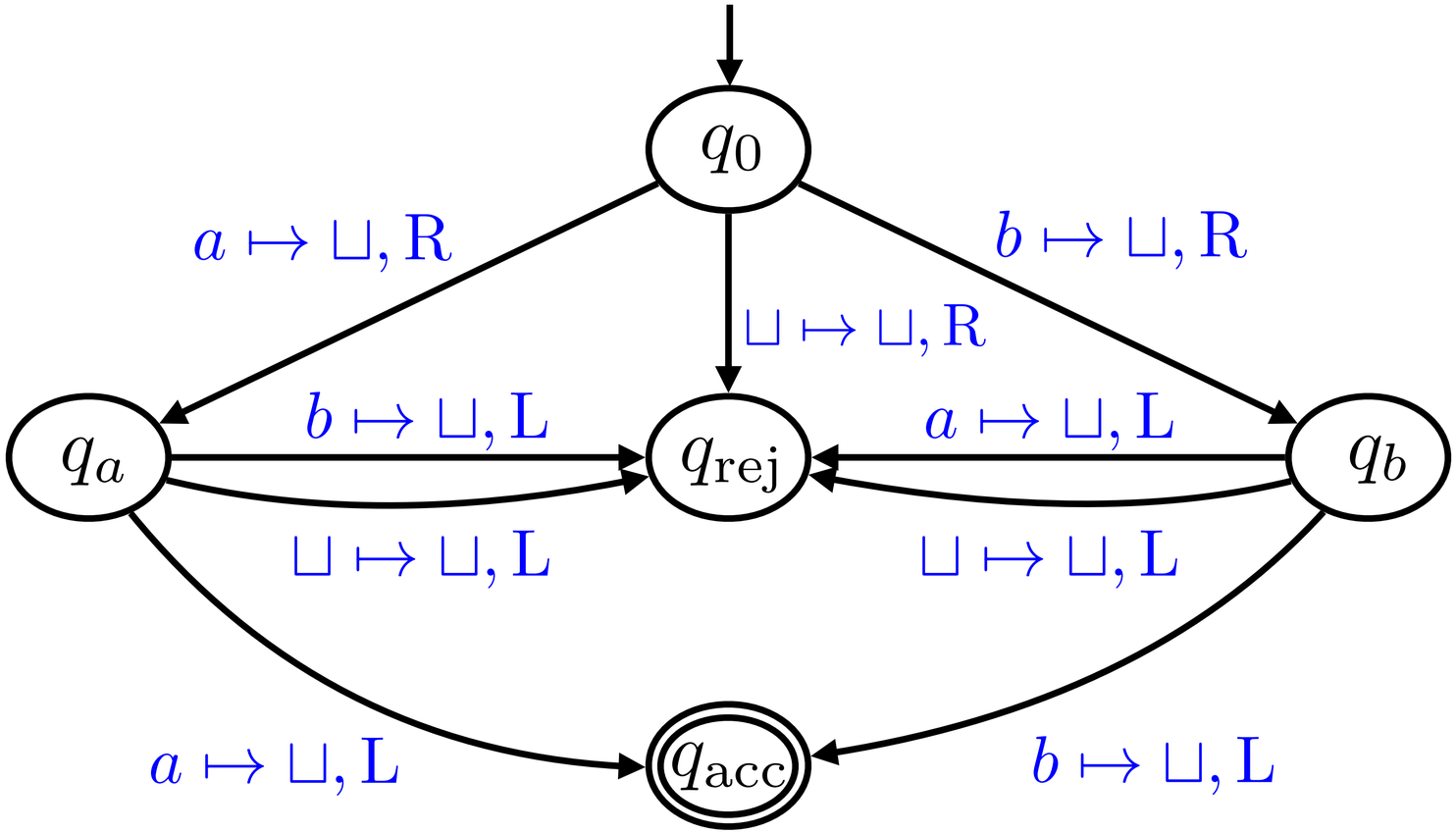
```
def M(input):
    i = 0
    STATE 0:
        letter = input[i];
        switch(letter):
            case ‘a’:  input[i] = ‘ ’; i++; go to STATE a;
            case ‘b’:  input[i] = ‘ ’; i++; go to STATE b;
            case ‘  ’:  input[i] = ‘ ’; i++; go to STATE rej;
    STATE a:
        letter = input[i];
        switch(letter):
            case ‘a’:  input[i] = ‘ ’; i--;  go to STATE acc;
            case ‘b’:  input[i] = ‘ ’; i--;  go to STATE rej;
            case ‘  ’:  input[i] = ‘ ’; i--;  go to STATE rej;

    ⋮
```



Poll

The machine accepts a string x if and only if: ???



Exercise

Let $\Sigma = \{0,1\}$.

Draw the state diagram of a TM that accepts a string iff it starts and ends with the same bit.

Formal definition: Turing machine

Definition: A ***Turing machine (TM)*** is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ where

- Q is a finite, non-empty set (called the ***set of states***);
- Σ is a finite, non-empty set with $\sqcup \notin \Sigma$ (called the ***input alphabet***);
- Γ is a finite set with $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$ (called the ***tape alphabet***);
- δ is a function of the form $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$; (called the ***transition function***);
- q_0 is an element of Q (called the ***start state***);
- q_{acc} is an element of Q (called the ***accepting state***);
- q_{rej} is an element of Q, with $q_{\text{rej}} \neq q_{\text{acc}}$ (called the ***rejecting state***).

Formal definition: TM accepting a string

See textbook.

DFAs vs TMs

- A DFA does not have access to tape cells that don't contain the input.
  (doesn't have access to unbounded memory)

- A DFA's tape head can only move right.

- A DFA can't write to the tape.

- A DFA can have more than one accepting state.

- A DFA halts once the input symbols are read.
  A TM might loop forever!!

DFAs vs TMs

- A DFA does not have access to tape cells that don’t contain the input.
  (doesn’t have access to unbounded memory)

- A DFA's tape head can only move right.

- A DFA can't write to the tape.

- A DFA can have more than one accepting state.

- A DFA halts once the input symbols are read.
A TM might loop forever!!

Definition: Decidable languages

Definition: Let M be a TM and $L \subseteq \Sigma^*$ a language.

We say that M ***solves***/***decides*** L if the following holds:

- if $w \in L$, M accepts w;
- if $w \notin L$, M rejects w.

Definition: A TM is a ***decider*** if it halts on all inputs.

Definition: A language $L \subseteq \Sigma^*$ is a called a ***decidable language*** if there is a TM that decides it.

The Big Questions

regular languages $\stackrel{?}{=}$ decidable languages

Is TM the "right" computational model?

Are all languages decidable?

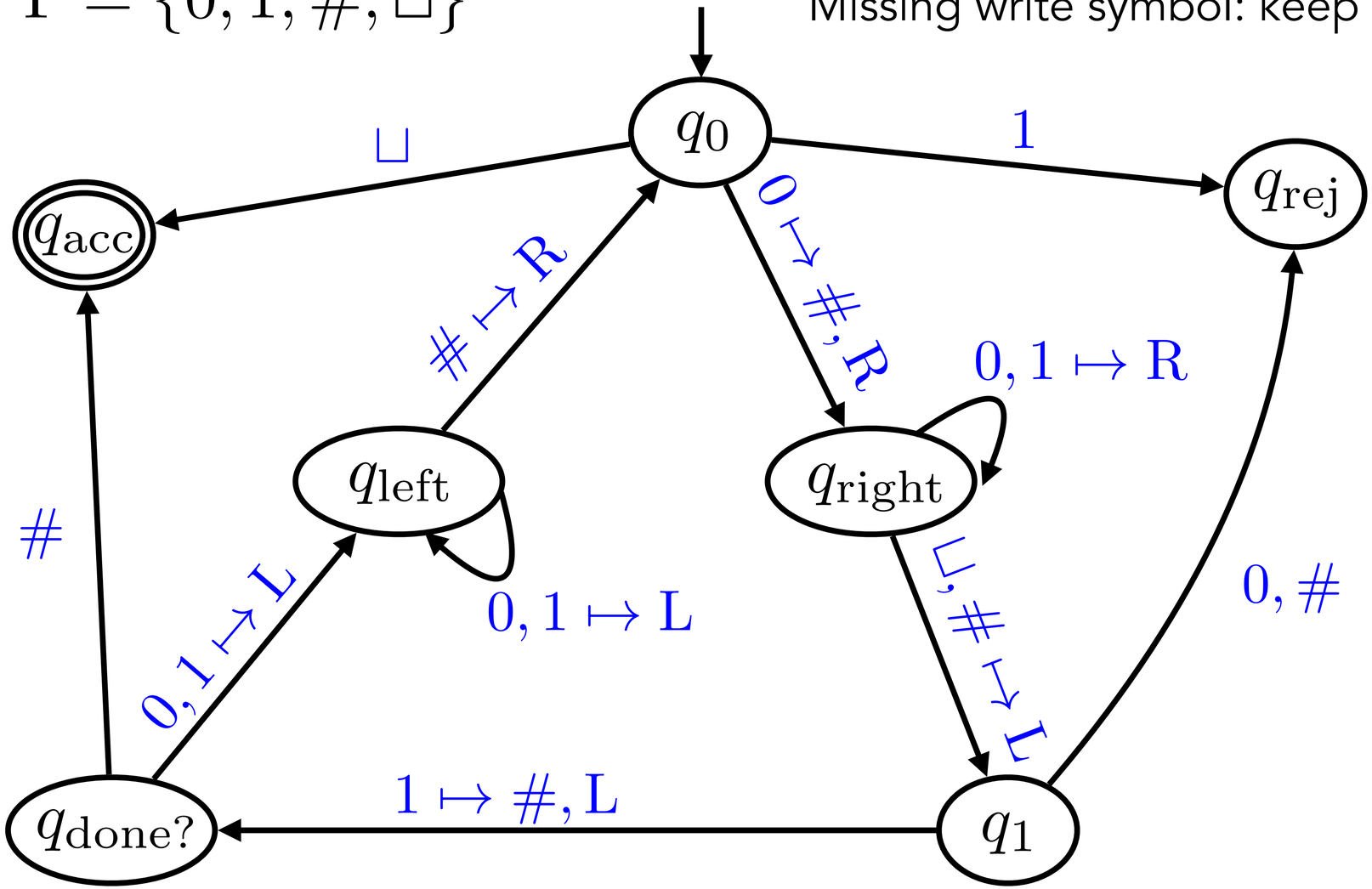
regular languages $\stackrel{?}{=}$ decidable languages

Turing machine that decides 0^n1^n

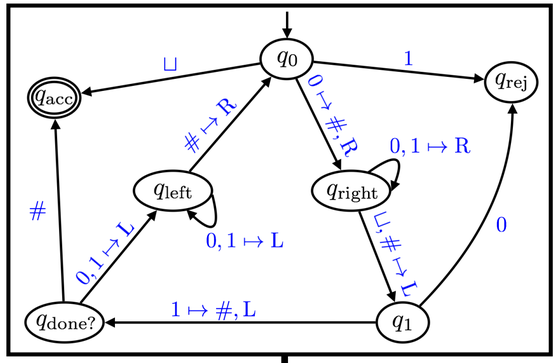
$\Sigma = \{0, 1\}$

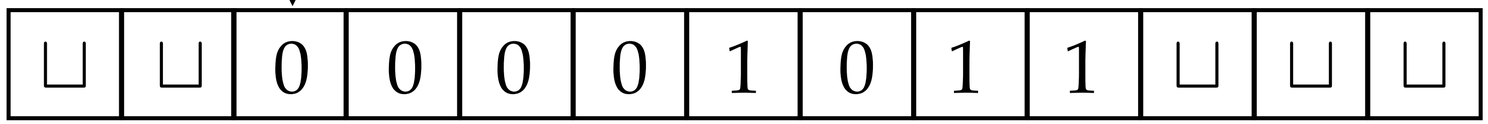
$\Gamma = \{0, 1, \#, \sqcup\}$

Missing transition: go to reject state.
Missing direction: doesn't matter.
Missing write symbol: keep the same.



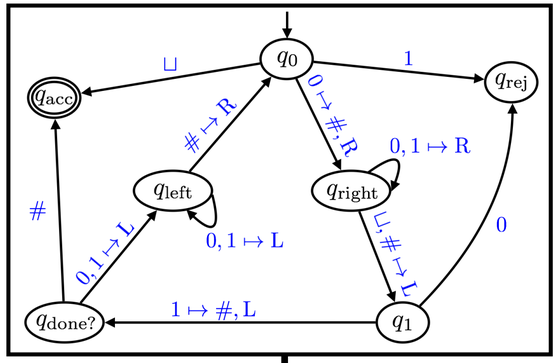
Turing machine that decides 0^n1^n





Input: 00001011

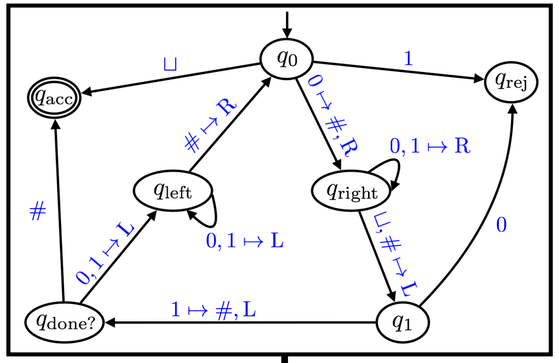
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

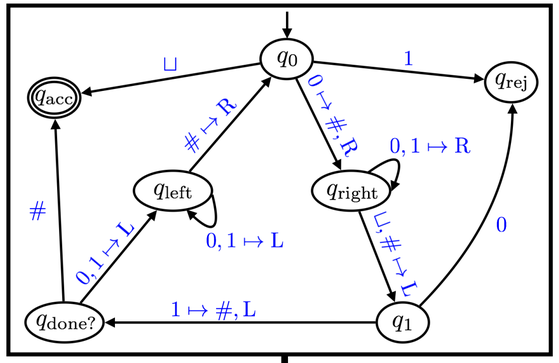
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

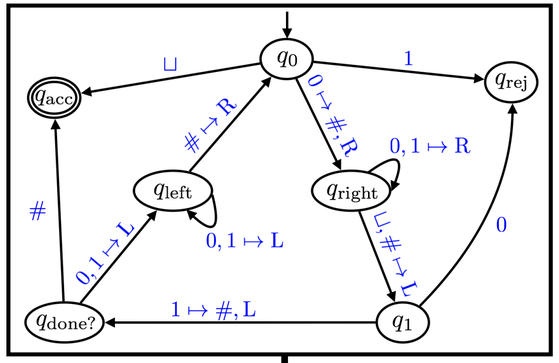
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

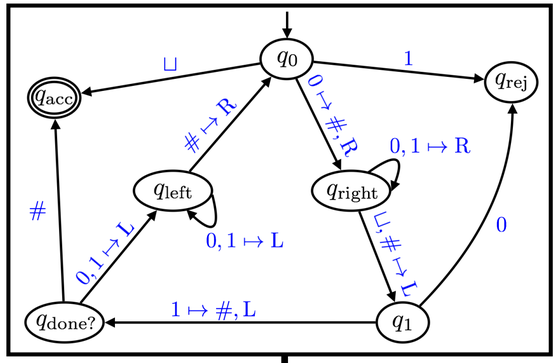
Turing machine that decides 0^n1^n

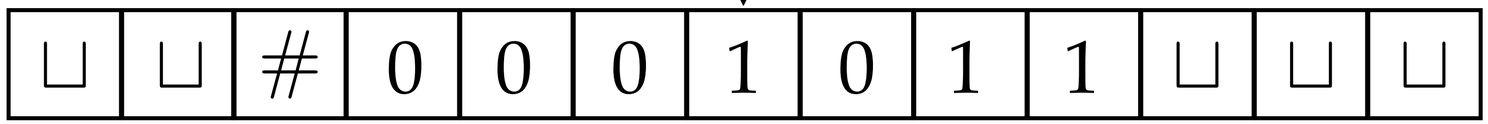


$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

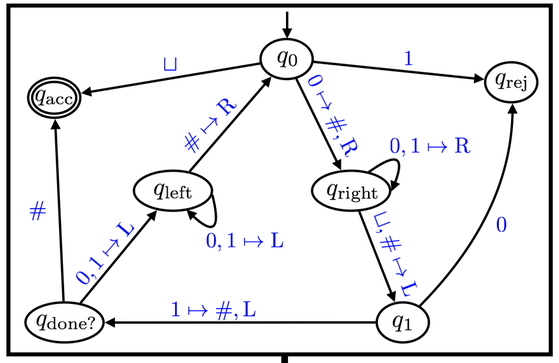
Turing machine that decides 0^n1^n

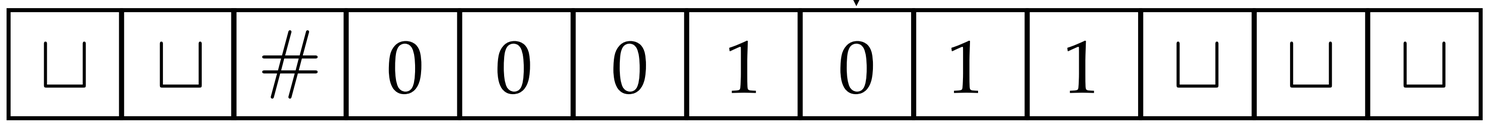




Input: 00001011

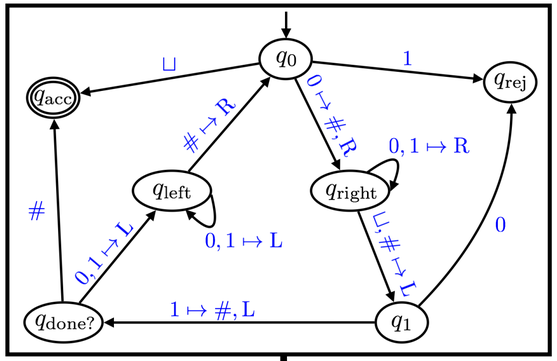
Turing machine that decides 0^n1^n





Input: 00001011

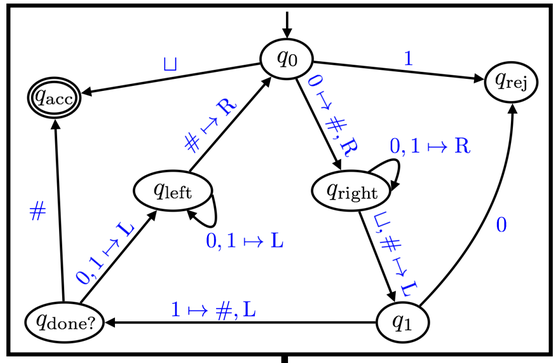
Turing machine that decides 0^n1^n

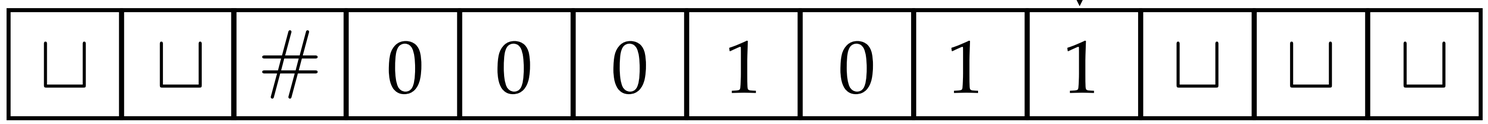


$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

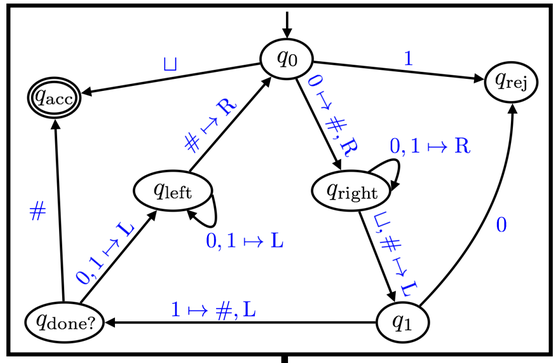
Turing machine that decides 0^n1^n





Input: 00001011

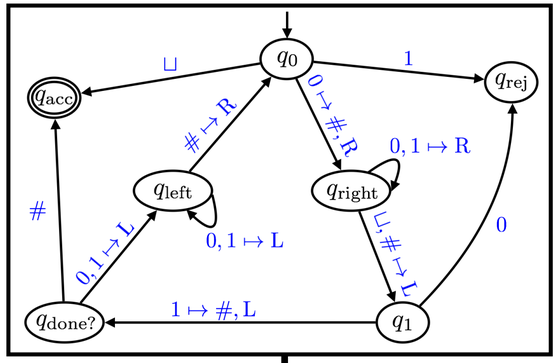
Turing machine that decides 0^n1^n

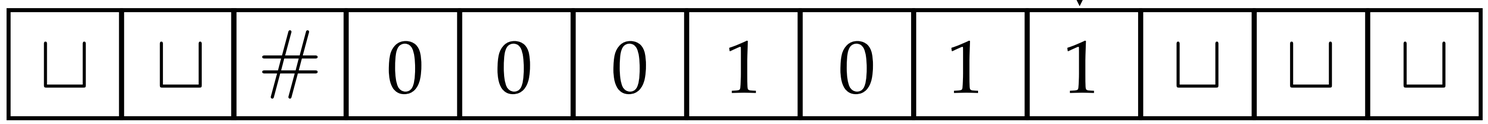


$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	1	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

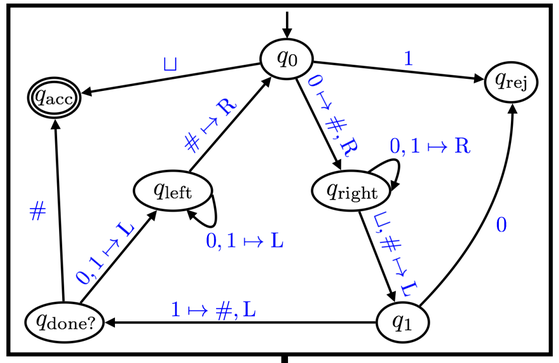
Turing machine that decides 0^n1^n





Input: 00001011

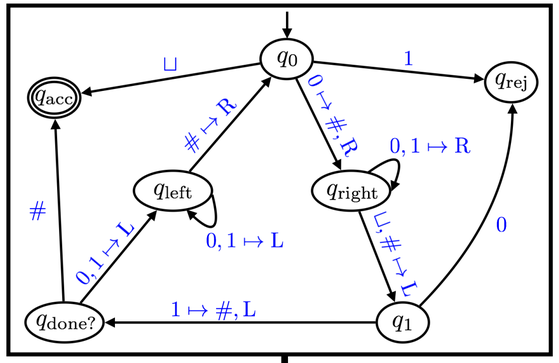
Turing machine that decides 0^n1^n

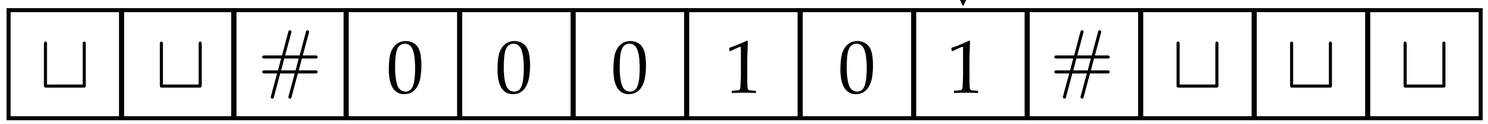


$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

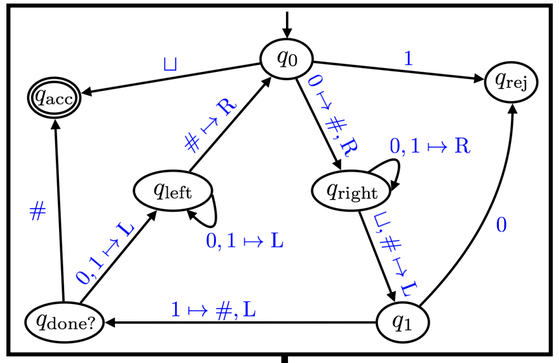
Turing machine that decides 0^n1^n

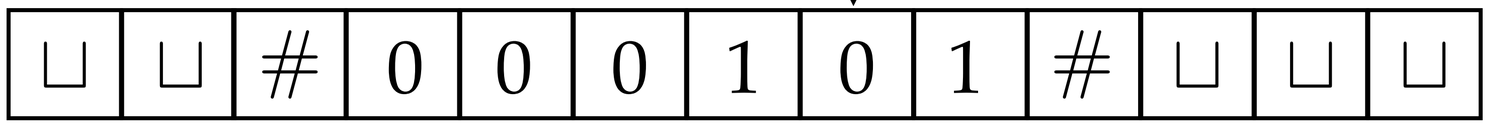




Input: 00001011

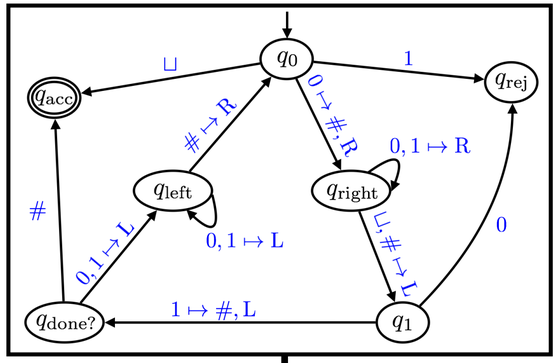
Turing machine that decides 0^n1^n

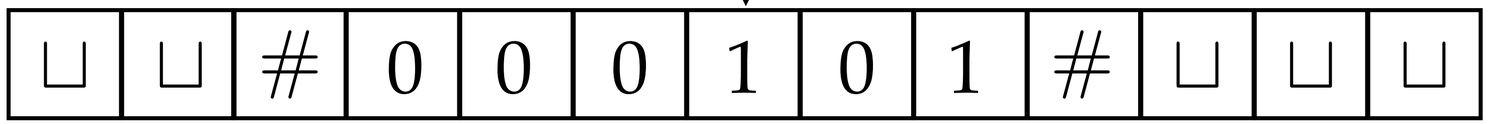




Input: 00001011

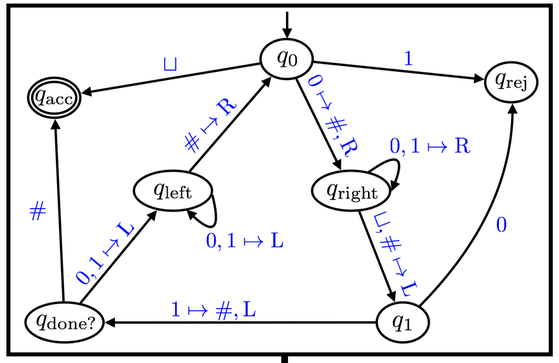
Turing machine that decides 0^n1^n





Input: 00001011

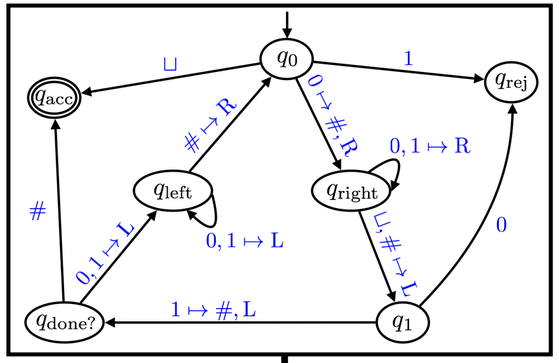
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

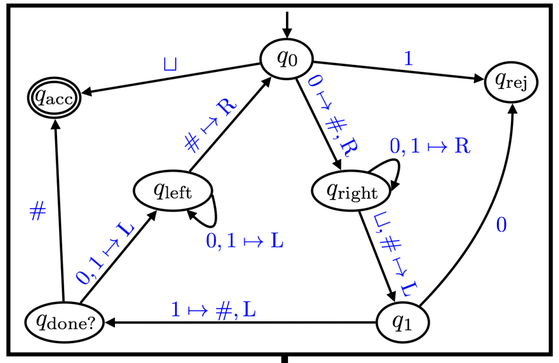
Turing machine that decides 0^n1^n

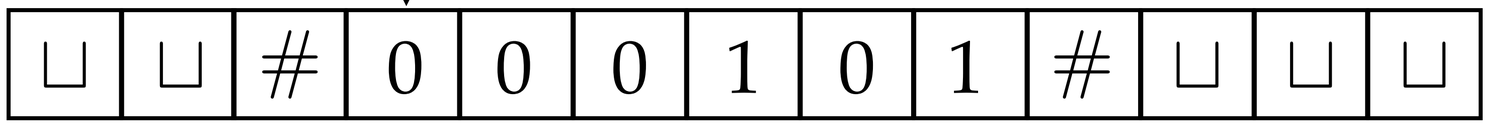


$\sqcup$	$\sqcup$	$\#$	0	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

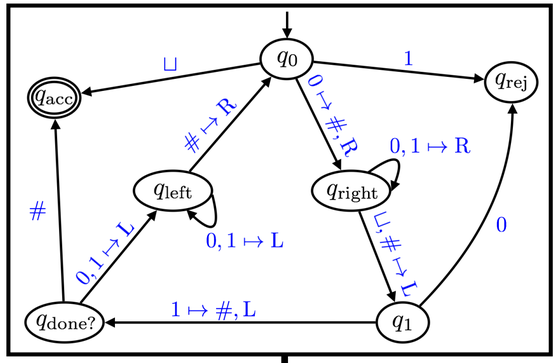
Turing machine that decides 0^n1^n

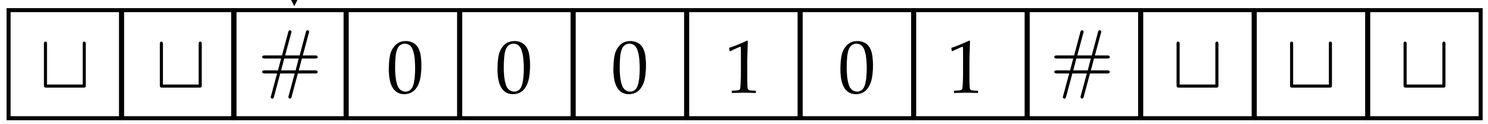




Input: 00001011

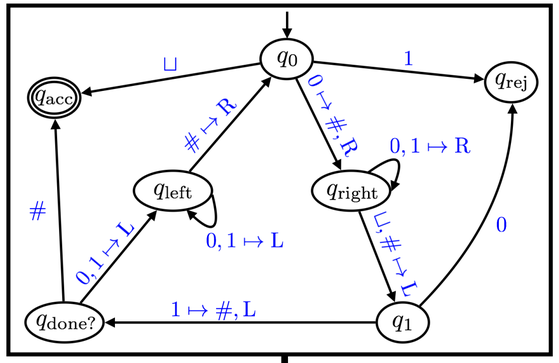
Turing machine that decides 0^n1^n

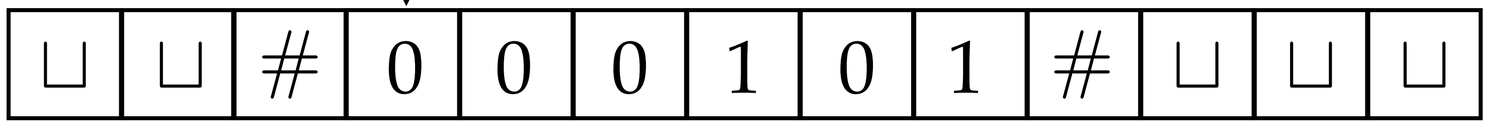




Input: 00001011

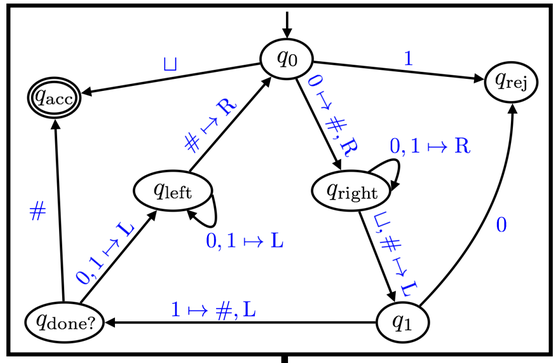
Turing machine that decides 0^n1^n





Input: 00001011

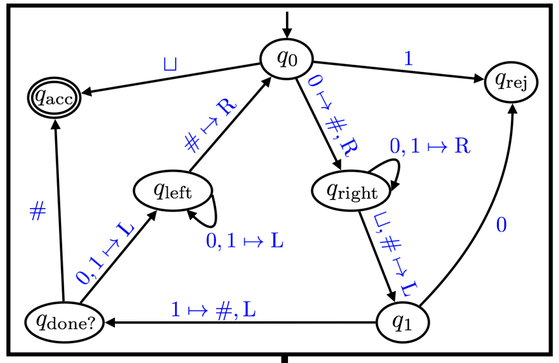
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	#	#	0	0	1	0	1	#	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

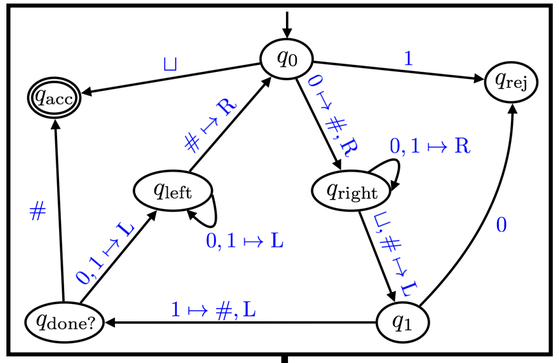
Turing machine that decides 0^n1^n

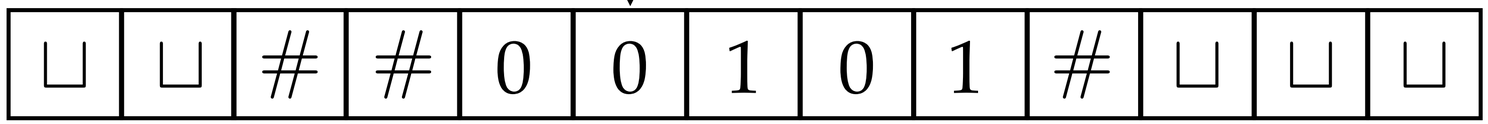


$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

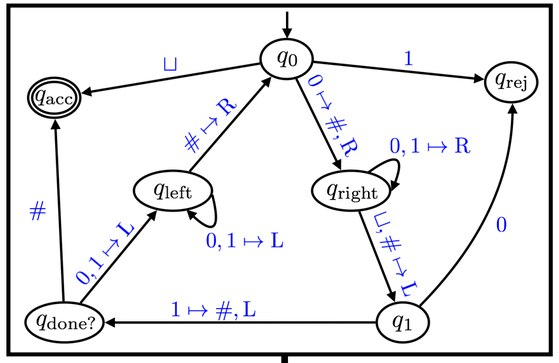
Turing machine that decides 0^n1^n

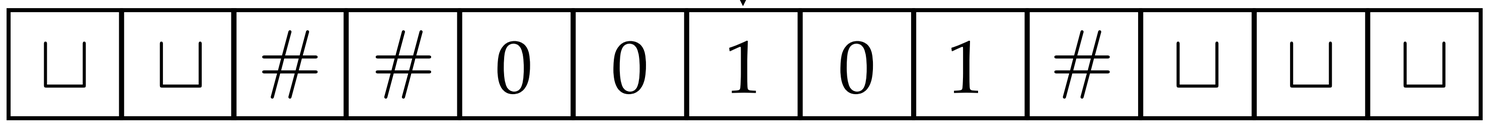




Input: 00001011

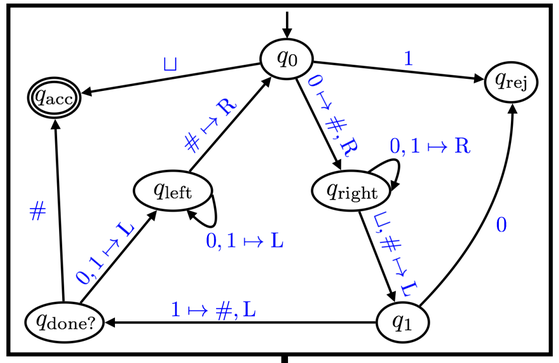
Turing machine that decides 0^n1^n

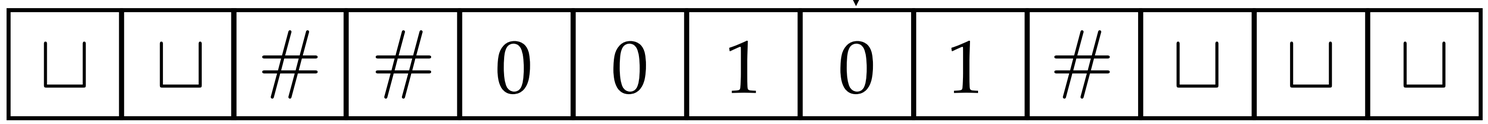




Input: 00001011

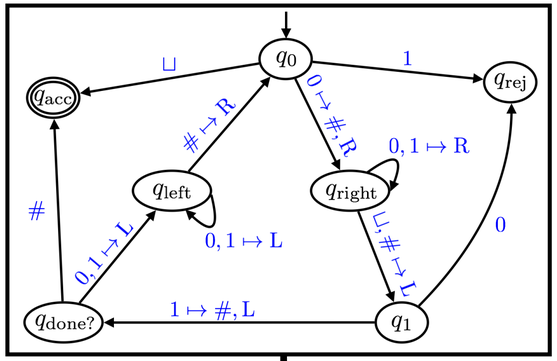
Turing machine that decides 0^n1^n





Input: 00001011

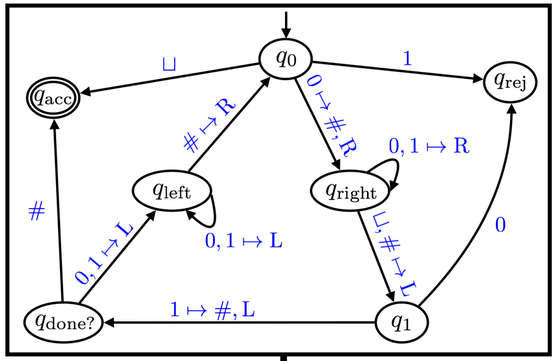
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

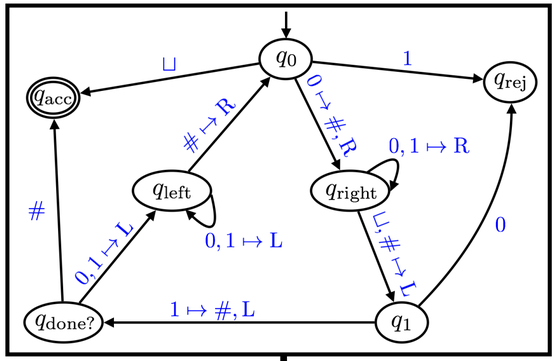
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

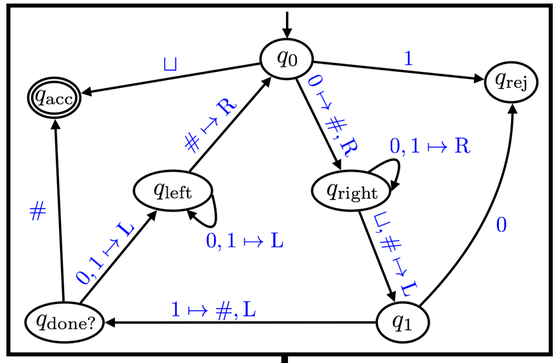
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	1	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

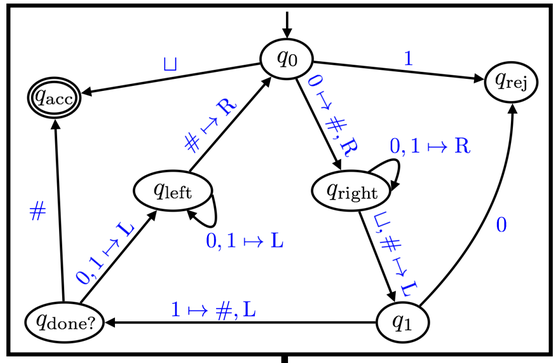
Turing machine that decides 0^n1^n

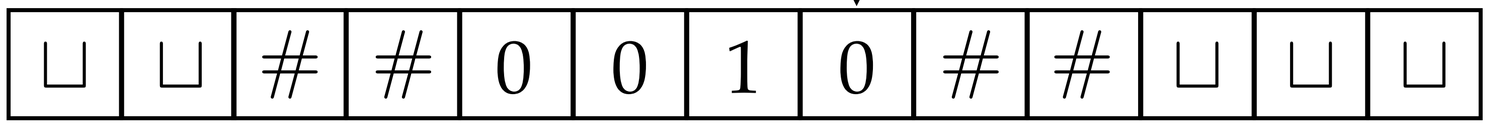


$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

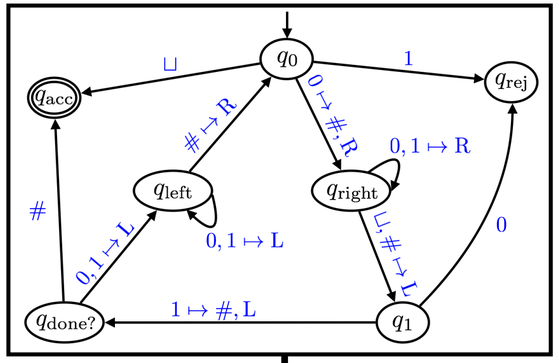
Turing machine that decides 0^n1^n

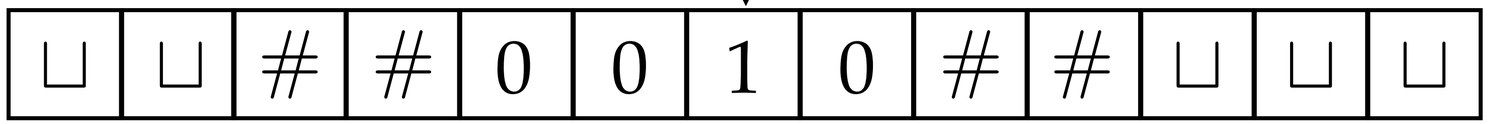




Input: 00001011

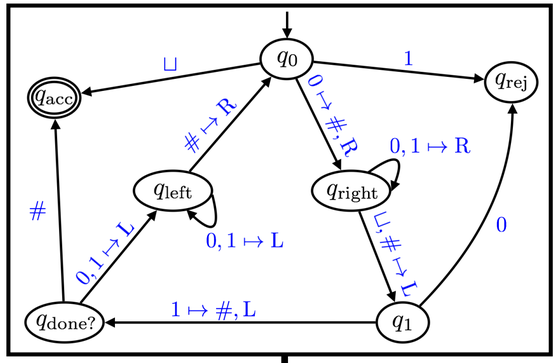
Turing machine that decides 0^n1^n

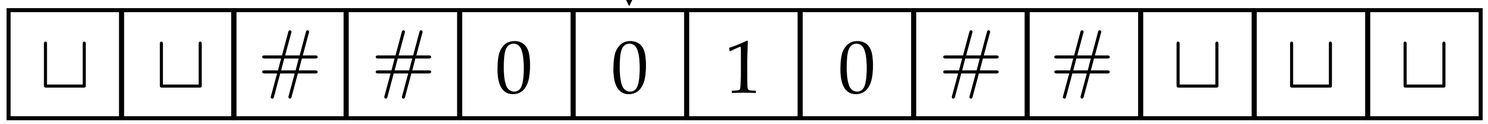




Input: 00001011

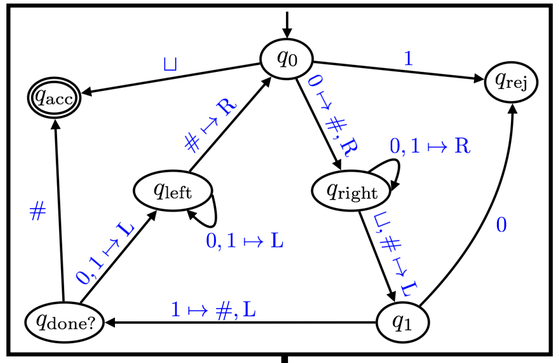
Turing machine that decides 0^n1^n





Input: 00001011

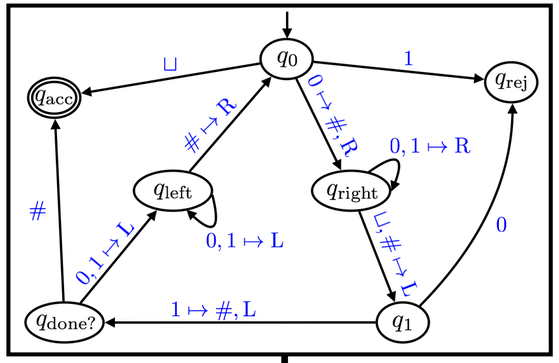
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

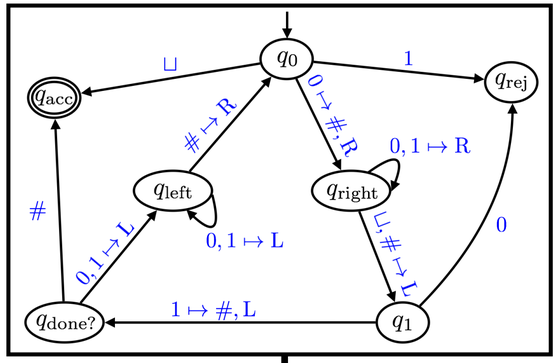
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

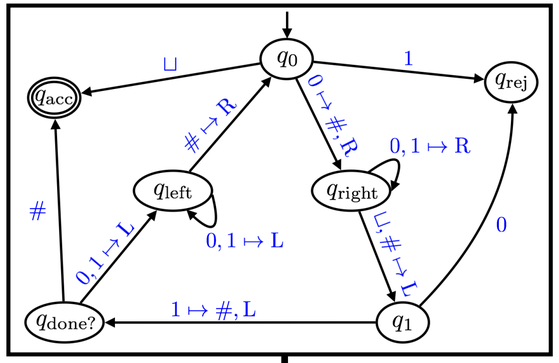
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	0	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

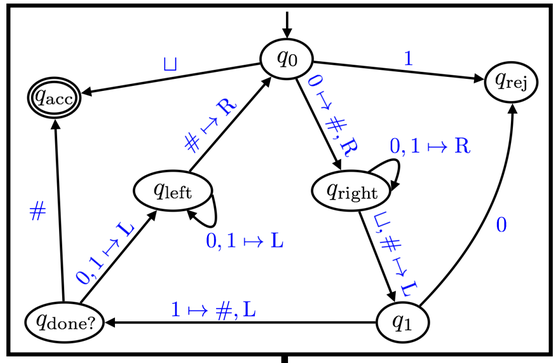
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	$\#$	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

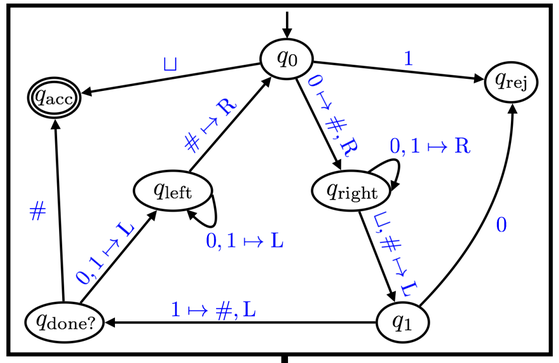
Turing machine that decides 0^n1^n

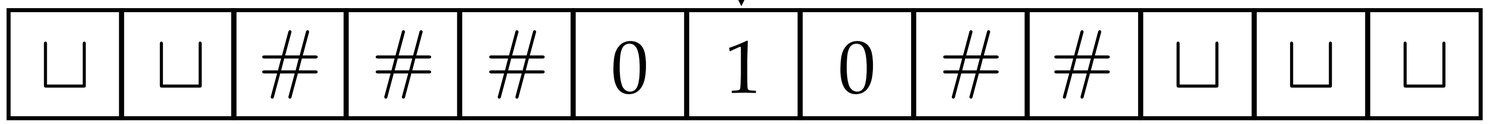


$\sqcup$	$\sqcup$	$\#$	$\#$	$\#$	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

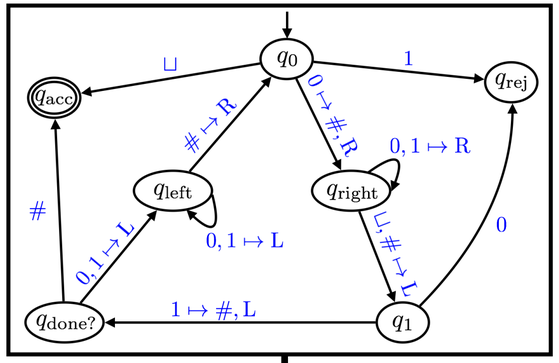
Turing machine that decides 0^n1^n





Input: 00001011

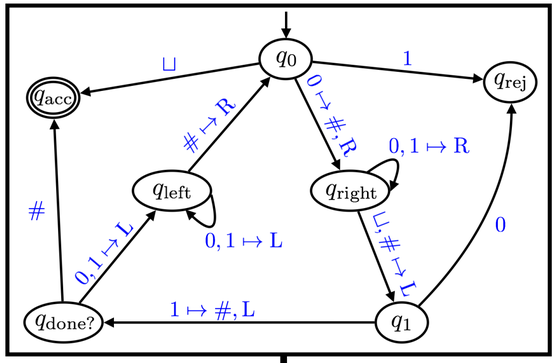
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	$\#$	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

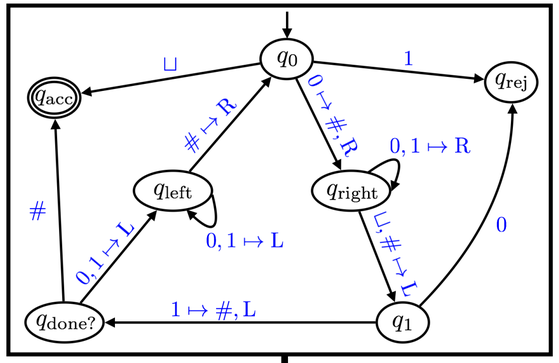
Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	$\#$	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011

Turing machine that decides 0^n1^n



$\sqcup$	$\sqcup$	$\#$	$\#$	$\#$	0	1	0	$\#$	$\#$	$\sqcup$	$\sqcup$	$\sqcup$

Input: 00001011 **Decision**: Reject

regular languages $\stackrel{?}{=}$ decidable languages

Is TM the "right" computational model?

Are all languages decidable?