

# Turing Machines

## Lecture 8

# Course Trajectory



We will see algorithms, what **can** be done.

But what **cannot** be done?

# Computation

## Problem:

To compute a function  $F$  that maps each input (a string) to an output bit

## Program:

A finitely described process taking a string as input, and outputting a bit (or not halting)

$P$  computes  $F$  if for every  $x$ ,  $P(x)$  outputs  $F(x)$  and halts

- A program is a finite bit string
- Programs can be *enumerated* — listed sequentially — (say, lexicographically) so that every program appears somewhere in the list

The set of all programs is **countable**.

1	$\epsilon$
2	0
3	1
4	00
5	01
6	10
7	11
8	000
9	001
10	010
11	011
12	100

# Computation

## Problem:

To compute a function  $F$  that maps each input (a string) to an output bit

## Program:

A finitely described process taking a string as input, and outputting a bit (or not halting)

$P$  computes  $F$  if for every  $x$ ,  $P(x)$  outputs  $F(x)$  and halts

- A function assigns a bit to each finite string
- Corresponds to an infinite bit string
- The set of all functions is **uncountable!**
  - As numerous as, say, real numbers in  $[0, 1]$

1	$\epsilon$	0
2	0	0
3	1	1
4	00	0
5	01	1
6	10	1
7	11	0
8	000	0
9	001	1
10	010	1
11	011	0
12	100	1



# Computation

## Problem:

To compute a function  $F$  that maps each input (a string) to an output bit

## Program:

A finitely described process taking a string as input, and outputting a bit (or not halting)

$P$  computes  $F$  if for every  $x$ ,  $P(x)$  outputs  $F(x)$  and halts

There are uncountably many functions!

But only countably many programs

Almost every function is uncomputable!

(non constructive proof)

# Course Trajectory

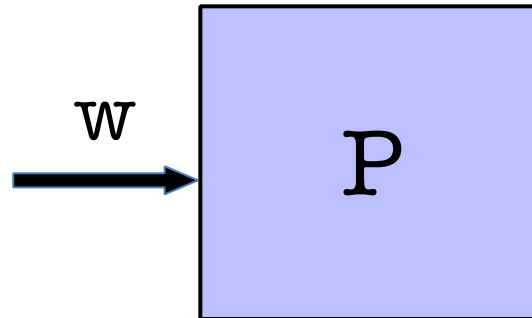


We will be looking at what can be computed at all?

What cannot be decided (undecidability)

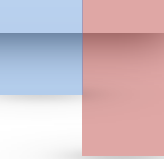
# What about a particular problems?

- Given program  $P$ , input  $w$ :



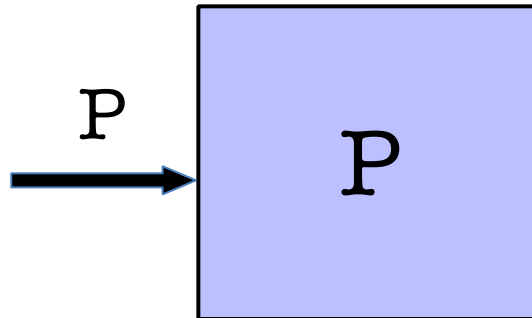
**Will halt or run into an infinite loop?**

**Halting problem!**



# Halting Problem Undecidable

- Given program  $P$ :



- *Write a program that decides if a program halts.*
- *Dual view of program as program and as data.*

# Alas!

There is no program that solves the Halting Problem!

Way to view code as data allows diagonalization proof.

# Computing

What does it mean to compute something?

“There is no algorithm for the halting problem”

What does it mean for something to be an algorithm formally?

Then I can say “For all algorithms...”

# A Brief History of Computing

- Leibnitz (circa 1600): Believed in universal language for encoding any problem (math, philosophy, religion).
- Thought if you properly encode any problem in binary form, there is a way to calculate an answer (run the algorithm)



calculemus!

# A Brief History of Computing

- Babbage (circa 1860): built Difference and Analytical Engine (Add, multiply, etc).
- It marks the transition from mechanised arithmetic to fully-fledged general purpose computation.
- Hypothesized that any mathematical question can be answered by the analytical engine, suitably encoded.



# A Brief History of Computing

- Hilbert (circa 1900): What can we prove in mathematical world?
- Proving and computing almost identical, proof= trace the program.
- The Entscheidungsproblem (decision problem): is there an **algorithm** that takes as input a statement of a **first-order logic** and answers "Yes" or "No" according to whether the statement is *universally valid*, i.e., valid in every structure satisfying the axioms.

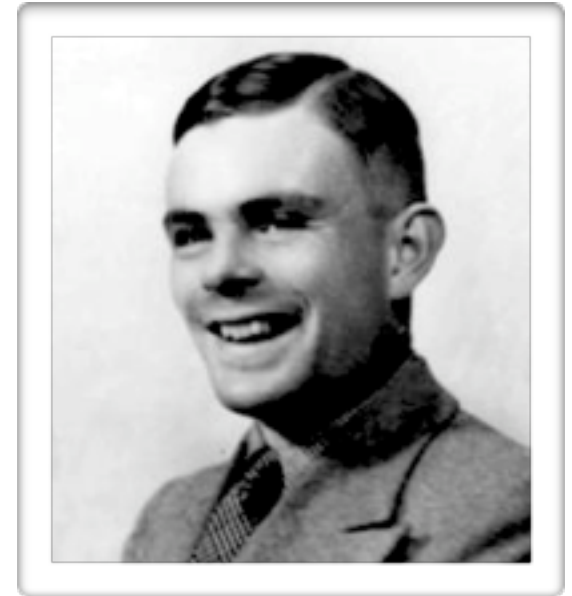
# A Brief History of Computing

- Gödel: no! there is no such algorithm
- Church: better way to prove it with functional programming



# Alan Turing

- British mathematician
  - cryptanalysis during WWII
  - arguably, father of AI, CS Theory
  - several books, movies
- Mathematically defined computation
  - Invented Turing Machines at 23 (1936). Turing machines can compute everything that is computable. He proved that **The Halting Problem** has no general algorithm (it is not possible to decide whether a turing machine will ever halt)



# Computation

- Computers were people at that time!
- The way people do math is write-erase-throw away.
- Turing proposed to abstract this process.



# Turing Machine



# Turing Machine



*Finite alphabet*

*Read*

*Write*

*Move +1 or -1*

*Halt condition*

# Turing Machine



*Finite alphabet*

*Read*

*Write*

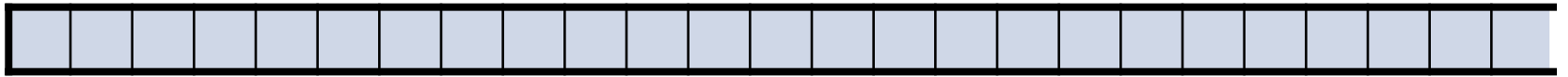
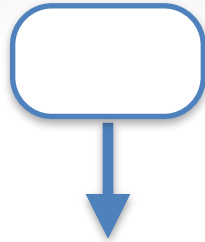
*Move +1 or -1*

*Halt condition*

*Internal state (finite number)*

Was designed as a model of human computation but it models computers as we know them

# Why do we care?



- *I need a model of computation that is simple enough to convince you that a TM can take as input a description of a TM and simulate it.*
- *Python interpreter in Python? Harder to explain Python semantics*
- *Will be able to define “what can we compute?”*





# TM for Decision Problems

$M = (Q, \Sigma, \Gamma, B, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ :

$\Gamma$  is a finite tape alphabet.

- $B$  or  $\square$  is the blank symbol (special symbol)
- $\Sigma$  is a finite input alphabet  $\Sigma \subseteq \Gamma \setminus B$

$Q$  is a finite set of states

$q_{\text{start}} \in Q$  is the initial state

$q_{\text{accept}}, q_{\text{reject}} \in Q$  accept/reject states

Or maybe run forever



# TM for Decision Problems

$M = (Q, \Sigma, \Gamma, B, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ :

$\Gamma$  is a finite tape alphabet.

- $B$  or  $\square$  is the blank symbol (special symbol)
- $\Sigma$  is a finite input alphabet  $\Sigma \subseteq \Gamma \setminus B$

$Q$  is a finite set of states

$q_{\text{start}} \in Q$  is the initial state

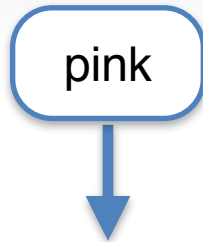
$q_{\text{accept}}, q_{\text{reject}} \in Q$  accept/reject states

Or maybe run forever

Transition function:  $\delta : Q \times \Gamma \text{ (read)} \rightarrow Q \times \Gamma \text{ (write)} \times \{L, R\}$



# Turing Machine



**tape** = string in  $\Gamma^*$  followed by infinite stream of  $\square$

we will treat  $011111 = 011111 \square \square \square \square \dots$

**configuration** = state, string (content of tape), and integer (position of tape).  $(Q, x, i) \in Q \times \Gamma^* \times \mathbb{N}$

e.g. (pink, 01011, 5)



# Configuration = ID (Instantaneous Description)

Contains all necessary information to capture the “current configuration of the computation”

state, tape-contents & head-location

a: symbol TM is about to read

Easy-to-read notation:  $(q, xay, i)$

$x \in \Gamma^*$ : tape contents left of the head

$q \in Q$ : state

$y \in \Gamma^*$ : tape contents at & right of the head  
(till last non-blank)

Initial ID:  $(q_{\text{start}}, \langle \text{input} \rangle, 0)$

# Relations $\Rightarrow$ $\Rightarrow$ on IDs

$ID_1 \Rightarrow ID_2$  iff  $ID_1$  evolves into  $ID_2$  in one step.

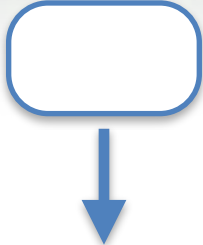
e.g., if  $\delta(q, a_i) = (q', b, L)$ , then

$$\underbrace{(q, a_1 a_2 \dots a_{i-1} a_i a_{i+1} \dots a_n, i)}_{\text{current ID}} \Rightarrow \underbrace{(q', a_1 a_2 \dots a_{i-2} a_{i-1} b a_{i+1} \dots a_n, i-1)}_{\text{next ID}}$$

$\Rightarrow^*$  is the reflexive & transitive closure of  $\Rightarrow$

Thus,  $ID_1 \Rightarrow^* ID_2$  iff  $M$ , when run from  $ID_1$ , reaches  $ID_2$  after some finite number (0 or more) of moves

# Accept/Reject



In DFAs it was clear when to accept an input string or reject it.

TM can:

- go back and forth over the input
- overwrite the input
- write on the tape way past the end of the input
- need an explicit state



# Definition of Acceptance

$M$  accepts  $w$  iff  $(q_{\text{start}}, w, 0) \Rightarrow^* (q_{\text{accept}}, x, i)$   
for some  $x \in \Gamma^*$

Note that  $M$  is allowed to accept  $w$  without scanning all of  $w$

$$L(M) = \{w \mid M \text{ accepts } w\}$$

$M$  does not accept  $w$  if starting from the ID  $q_{\text{start}} w$  :

1.  $M$  halts in  $q_{\text{reject}}$ , or
2.  $M$  crashes (head moves off the tape), or
3.  $M$  never stops



# Deciding/Recognizing a Language

$$L(M) = \{w \mid M \text{ accepts } w\}$$

is called the language *recognized* by  $M$

$M$  *decides*  $L(M)$  if on input  $w \notin L$ ,  $M$  halts in  $q_{\text{reject}}$

If a TM decides the language it recognizes,  
then, on *every* input, it halts in  $q_{\text{accept}}$  Or  $q_{\text{reject}}$

Easy to change “crashes” to rejects

But turns out the we can't avoid infinite  
executions! (can't tell if it is going to be infinite)





# Deciding/Recognizing a Language

$$L(M) = \{w \mid M \text{ accepts } w\}$$

is called the language *recognized* by  $M$

$M$  *decides*  $L(M)$  if on input  $w \notin L$ ,  $M$  halts in  $q_{\text{reject}}$

Fundamental questions of computability:

Which languages are recognizable?

Recursively  
Enumerable  
Language

Which languages are decidable?

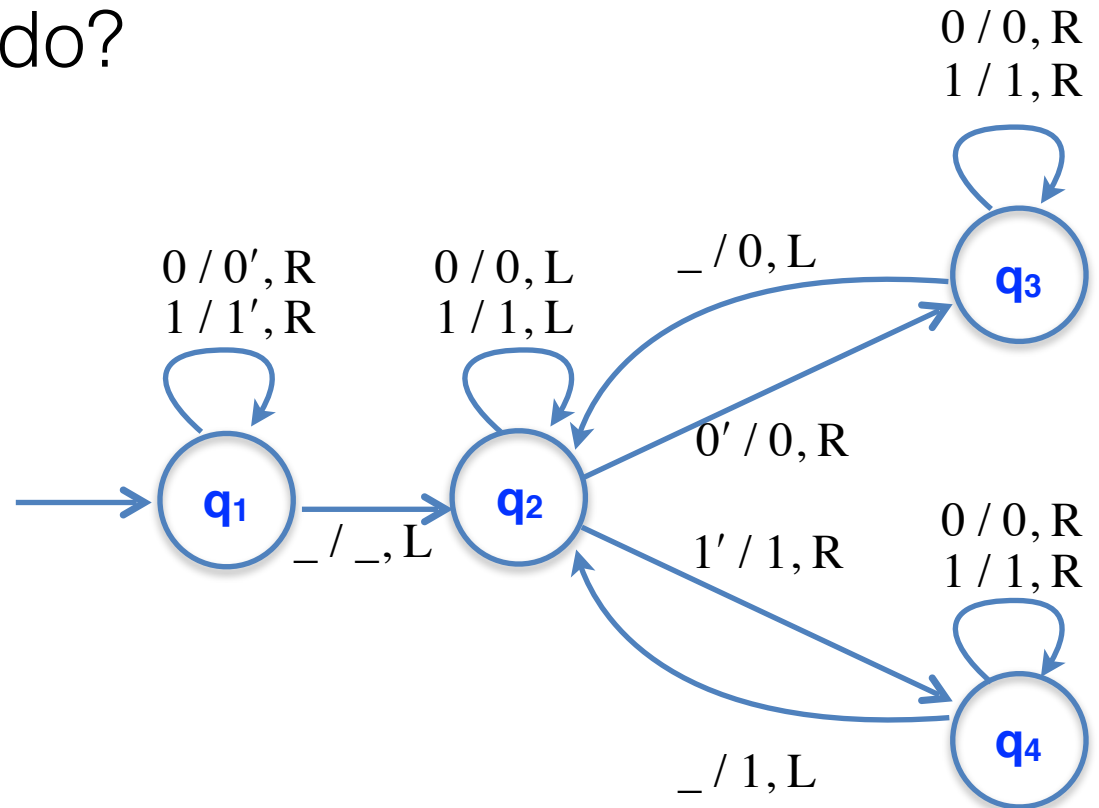
Recursive  
Language

# Example

Input alphabet :  $\Sigma = \{0,1\}$

Tape alphabet :  $\Gamma = \{0,1,0', 1', \_ \}$

What does this TM do?



# Example

q<sub>1</sub>

0	0	1	_	_	_	_	_	_	_	_
---	---	---	---	---	---	---	---	---	---	---

0'	0	1	_	_	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

0'	0'	1	_	_	_	_	_	_	_	_
----	----	---	---	---	---	---	---	---	---	---

0'	0'	1'	_	_	_	_	_	_	_	_
----	----	----	---	---	---	---	---	---	---	---

q<sub>2</sub>

0'	0'	1'	_	_	_	_	_	_	_	_
----	----	----	---	---	---	---	---	---	---	---

q<sub>4</sub>

0'	0'	1	_	_	_	_	_	_	_	_
----	----	---	---	---	---	---	---	---	---	---

q<sub>2</sub>

0'	0'	1	1	_	_	_	_	_	_	_
----	----	---	---	---	---	---	---	---	---	---

q<sub>3</sub>

0'	0'	1	1	_	_	_	_	_	_	_
----	----	---	---	---	---	---	---	---	---	---

0'	0	1	1	_	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

0'	0	1	1	_	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

0'	0	1	1	_	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

q<sub>2</sub>

0'	0	1	1	0	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

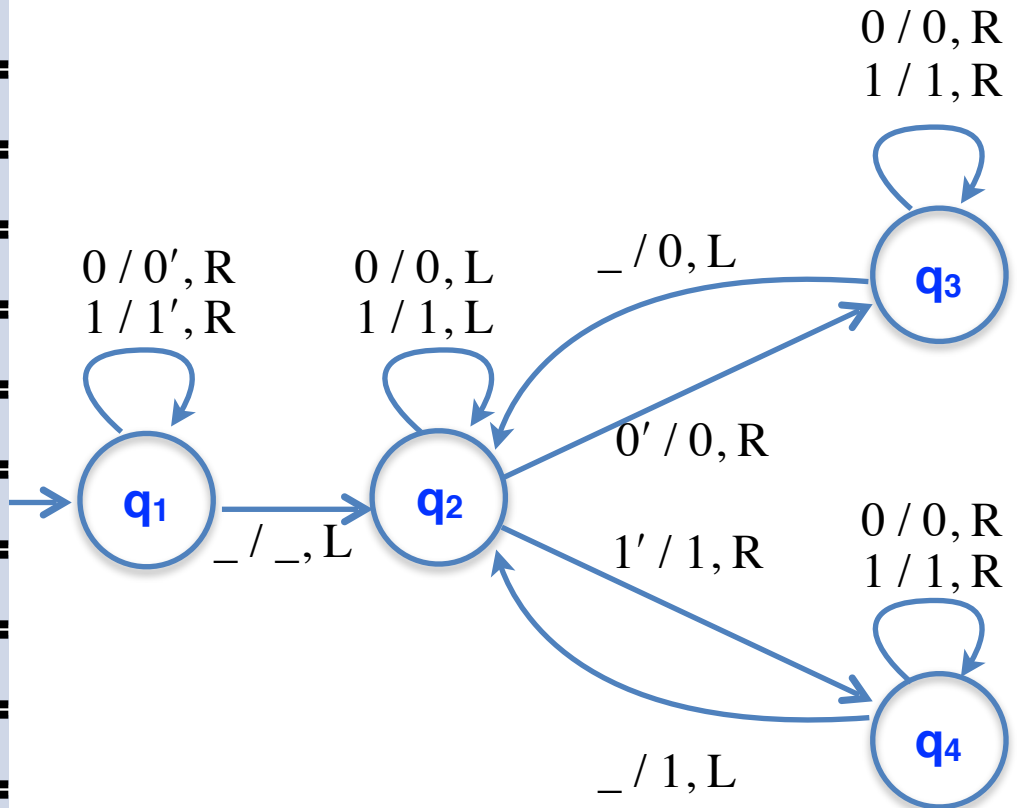
0'	0	1	1	0	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

0'	0	1	1	0	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

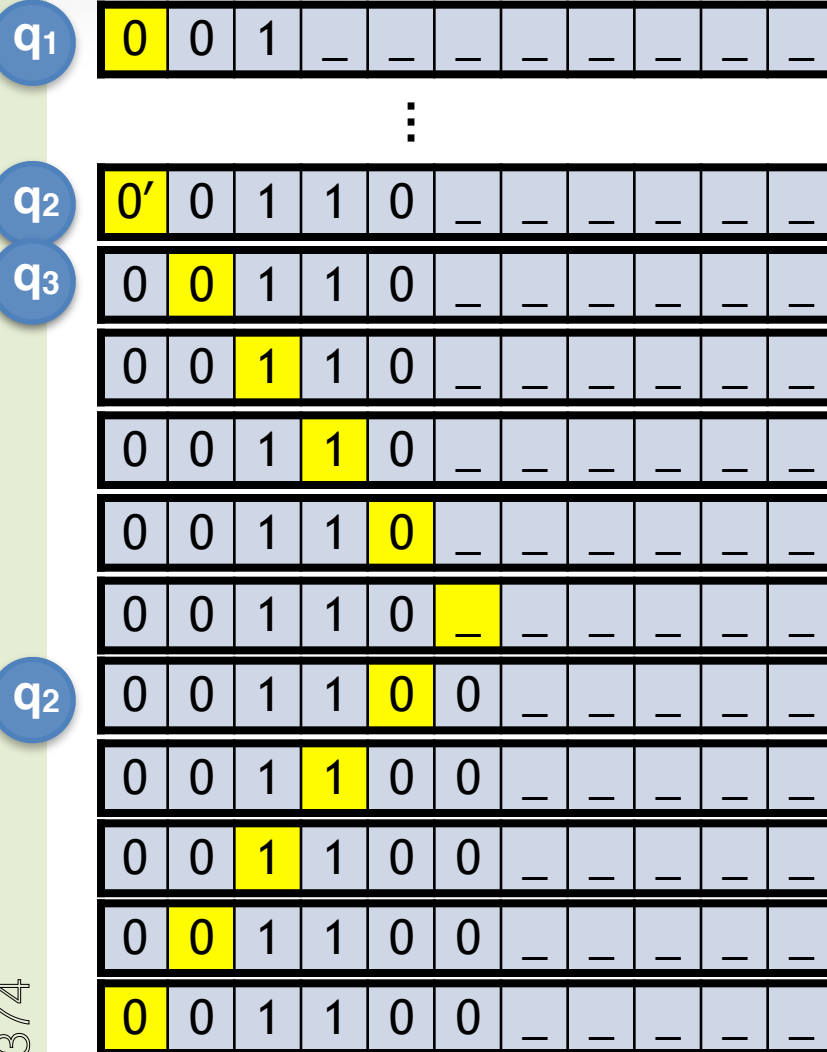
0'	0	1	1	0	_	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

Input alphabet :  $\Sigma = \{0,1\}$

Tape alphabet :  $\Gamma = \{0,1,0', 1', _\}$



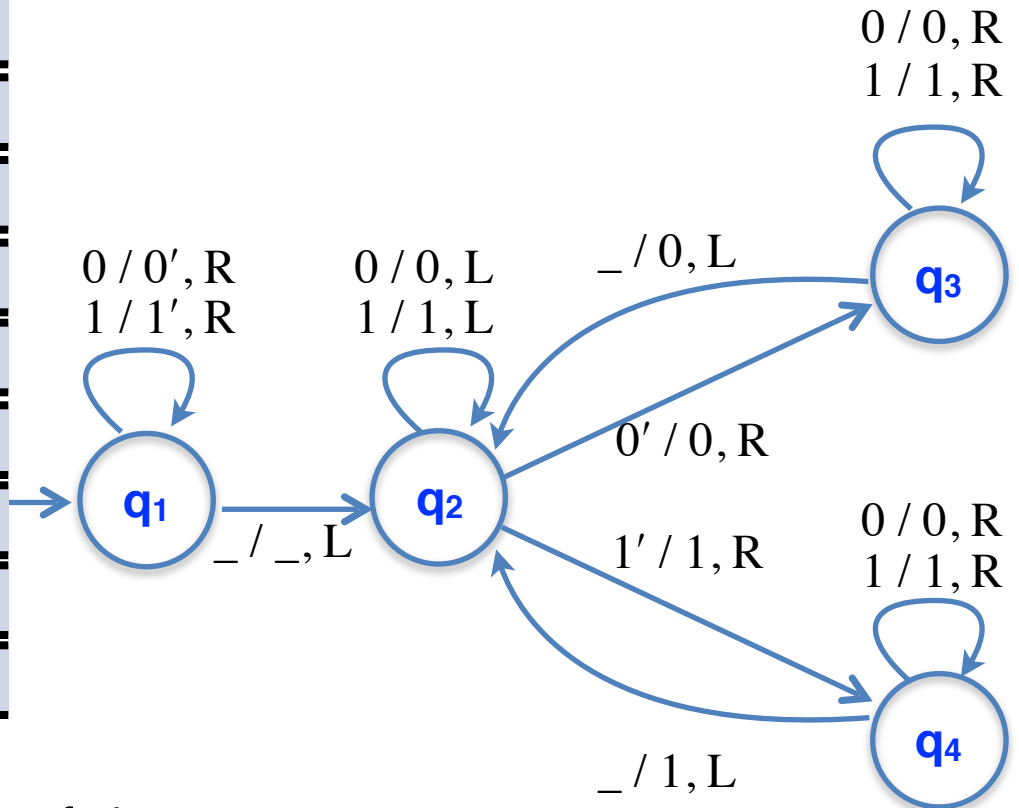
# Example



Input alphabet :  $\Sigma = \{0,1\}$

Tape alphabet :  $\Gamma = \{0,1,0', 1', \_ \}$

Maps  $w$  to  $ww^R$



Next?

Crashes! Head moves out of the tape.

# What does this TM do?

$$\delta(p, a) = (q, b, \Delta)$$

$$\delta(0, 0) = (0, 0, +1)$$

$$\delta(0, 1) = (1, 0, +1)$$

$$\delta(0, \square) = (\text{halt}, 0, +1)$$

---

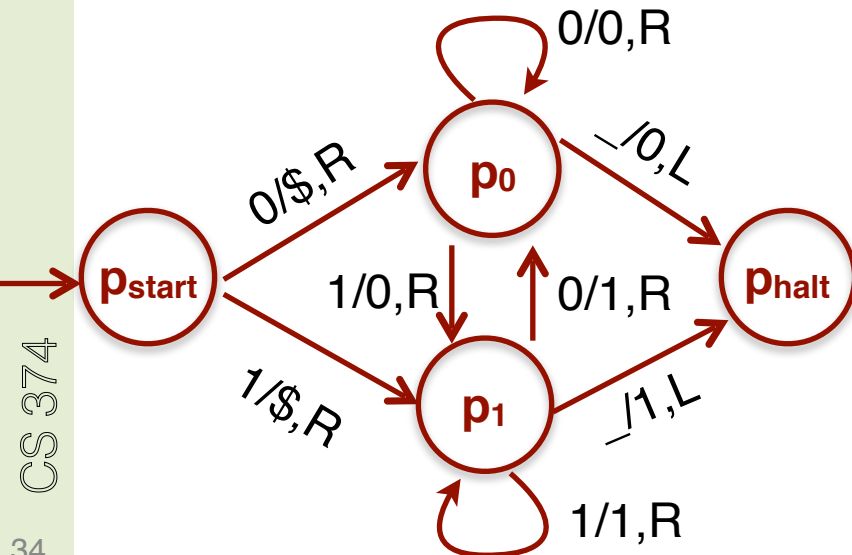
$$\delta(1, 0) = (0, 1, +1)$$

$$\delta(1, 1) = (1, 1, +1)$$

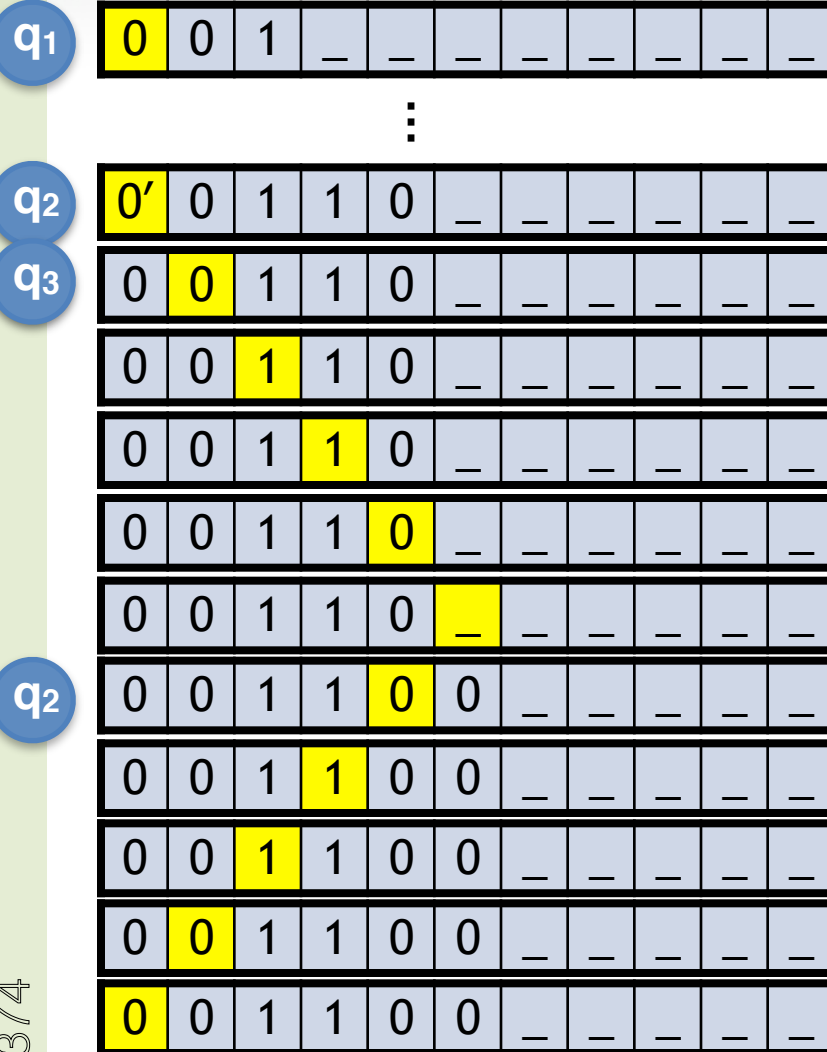
$$\delta(1, \square) = (\text{halt}, 1, +1)$$



# What does this TM do?



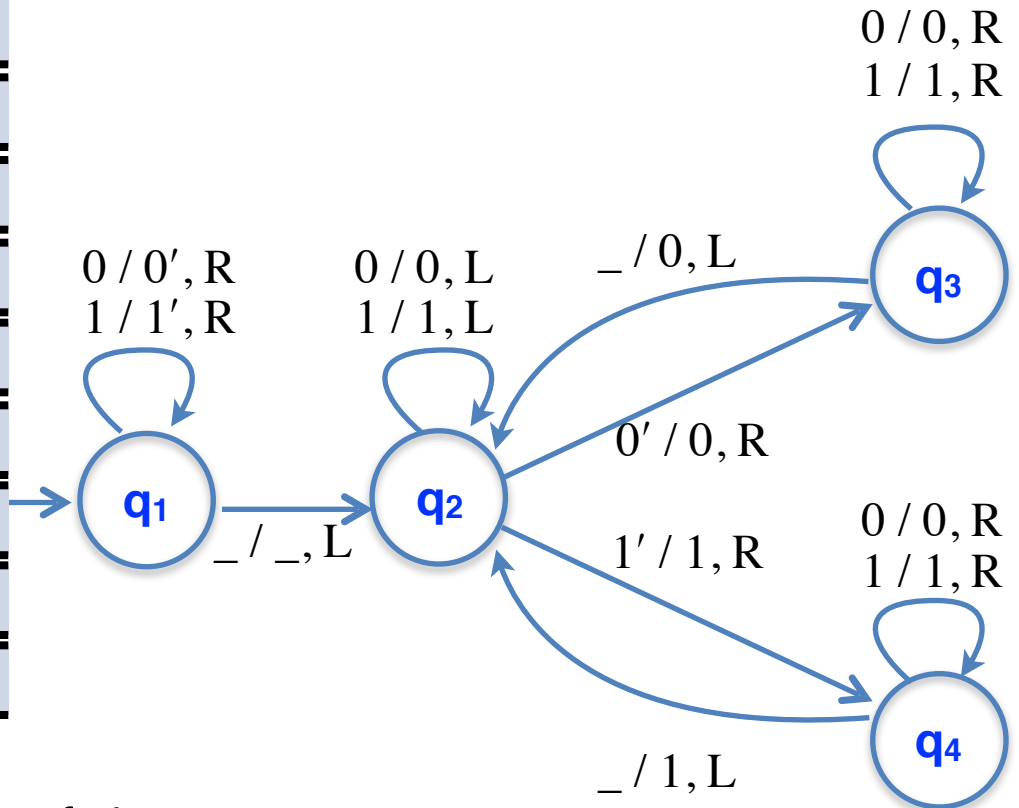
# Example



Input alphabet :  $\Sigma = \{0,1\}$

Tape alphabet :  $\Gamma = \{0,1,0', 1', \_ \}$

Maps  $w$  to  $ww^R$



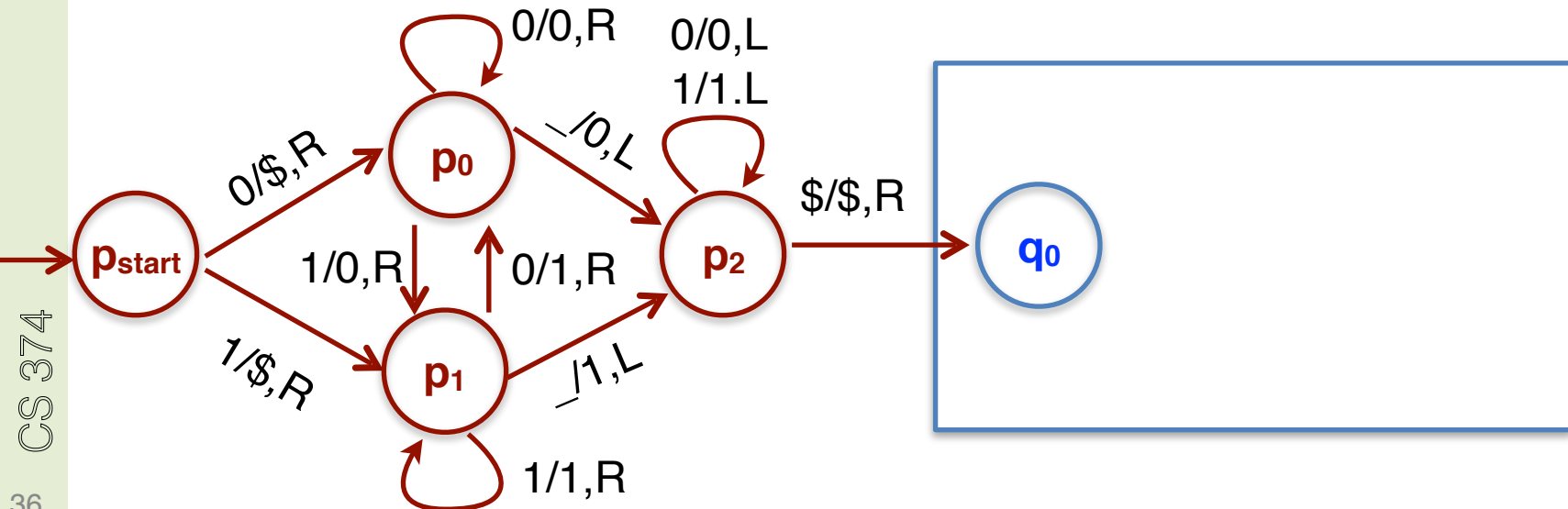
Next?

Crashes! Head moves out of the tape.

# Avoiding Crashing

Given  $M$  (that may crash), an “equivalent”  $M'$  which goes to  $q_{\text{reject}}$  instead of crashing

Idea: Rewrite input  $w$  to be  $\$w$ , place the head on the first symbol of  $w$  and run  $M$ . If head ever scans \$, move to  $q_{\text{reject}}$  (and move the head right)

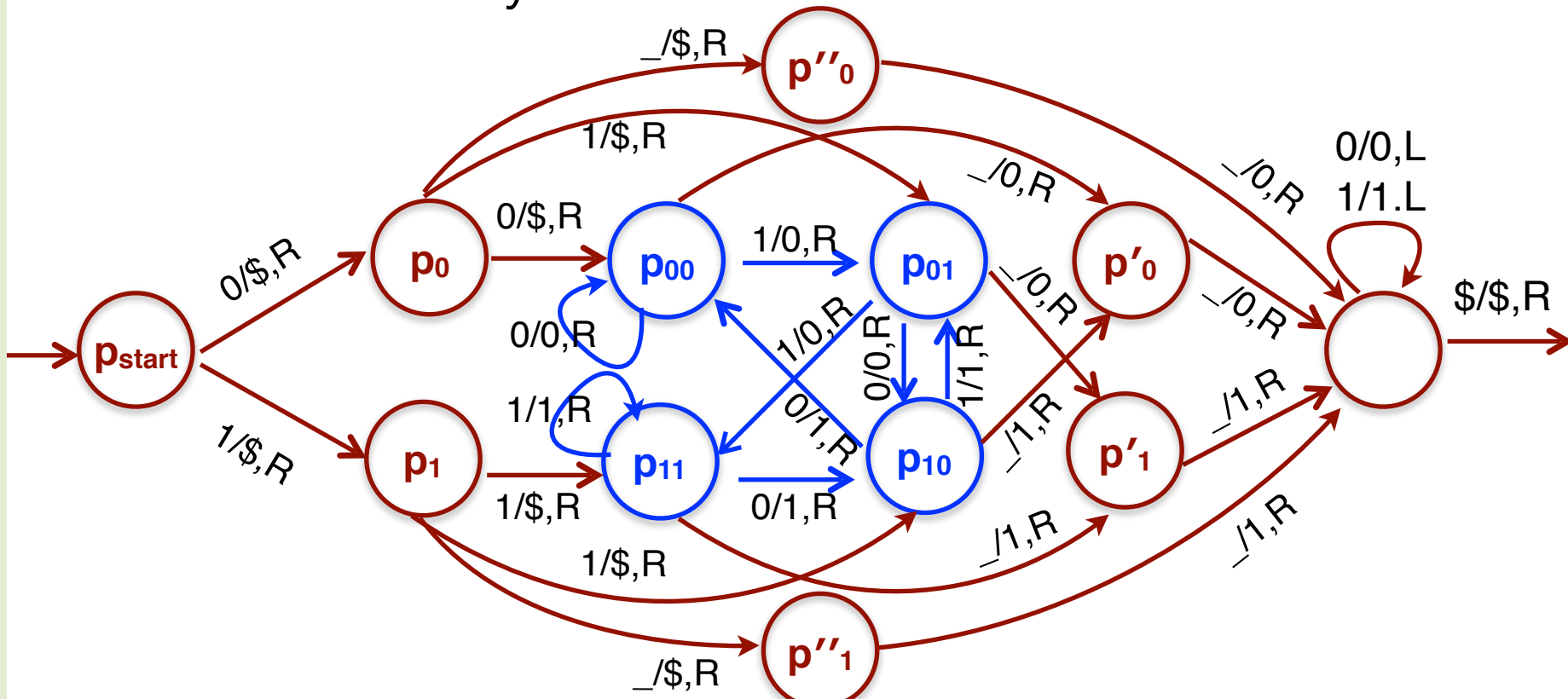




# Shifting by $k$ Positions

Can do “shift-by-1”  $k$  times. But  $k$  scans of tape.

To shift by  $k$  positions to the right in a single scan:  
Remember last  $k$  symbols. Overwrite current cell  
with symbol from  $k$  cells behind



# Binary Addition

$$L = \{ x\#y\#z \mid x, y, z \in \{0,1\}^*, |x|=|y|=|z|, x+y=z \text{ in binary} \}$$

Plan:

0	1	#	0	1	#	1	0	_	_	_
---	---	---	---	---	---	---	---	---	---	---

shift

:

\$	0	1	#	0	1	#	1	0	_	_
----	---	---	---	---	---	---	---	---	---	---

carry  $c=0$

check LSB



:

\$	0	#	@	0	#	@	1	_	_	_
----	---	---	---	---	---	---	---	---	---	---

carry  $c=1$

:

check MSB



:

\$	#	@	@	#	@	@	_	_	_	_
----	---	---	---	---	---	---	---	---	---	---

carry  $c=0$

check finished



# Binary Addition

$L = \{ x\#y\#z \mid x, y, z \in \{0,1\}^*, |x|=|y|=|z|, x+y=z \text{ in binary} \}$

Shift input  $w$  to make it  $\$w$ .

Scan the tape to ensure  $w$  matches  $(0+1)^*\#(0+1)^*\#(0+1)^*$

Return head to the left end (right of  $\$$ )

(In finite memory, carry-bit  $c$  initialized to 0)

Repeat

copy the digit to the left of first  $\#$  into finite state, and overwrite it with  $\#$  (replace old  $\#$  by  $@$ ). If no digit there, accept if carry is 0 & no digits left; else reject.

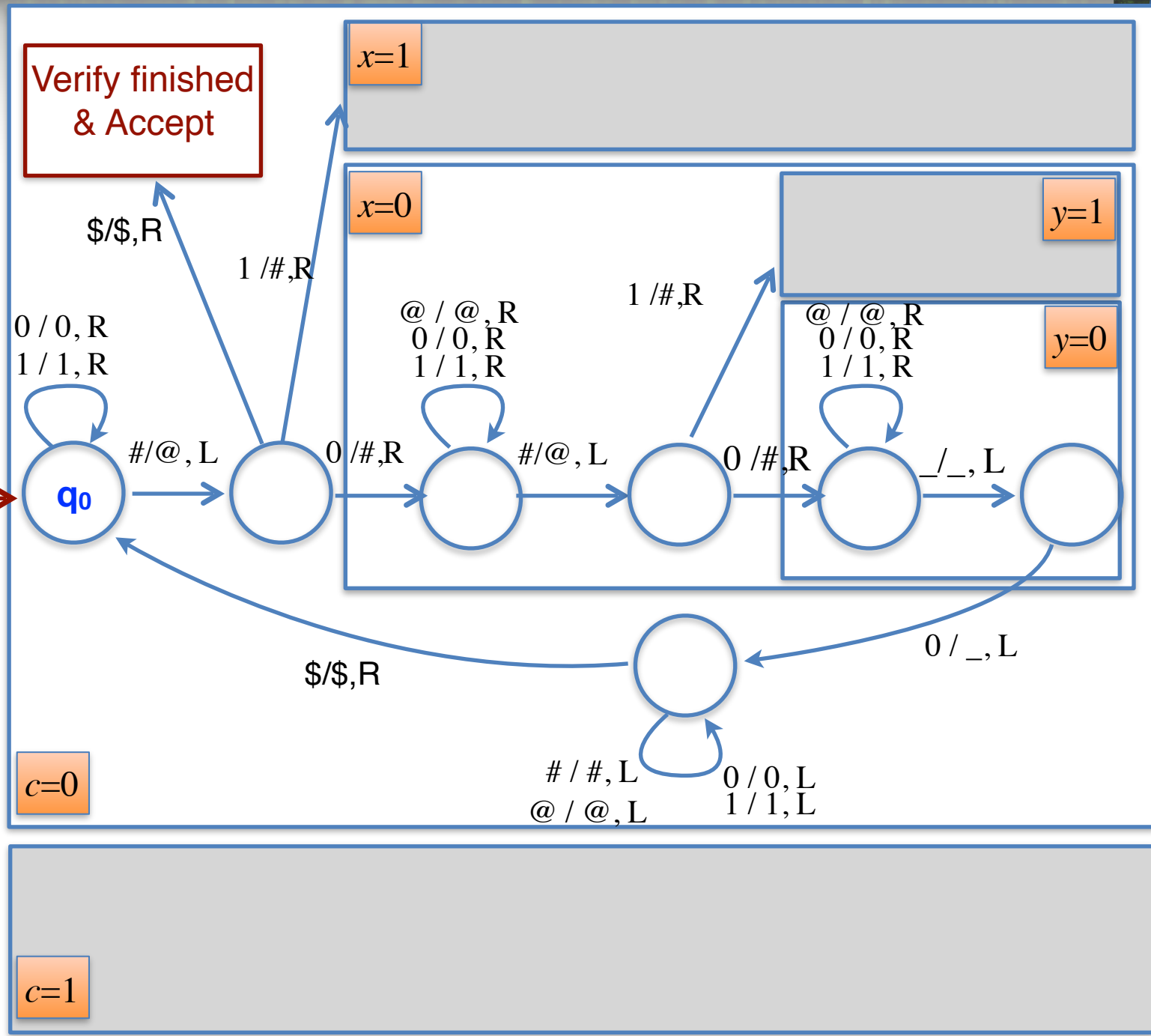
copy the digit to the left of second  $\#$  into finite state, and move  $\#$  left (replace old  $\#$  by  $@$ ). If no digit there, reject.

check if the right most digit is “correct”. Reject if no digit or if it is not correct; else erase digit and update carry.

Move head to the left end (right of  $\$$ )



Shift & Format Check



# What can a TM do?



Can shift by any number  $i$ , by keeping  $2^i$  states to remember what were in the first  $i$  places of the tape.

Can do simple arithmetic operations, addition, multiplication etc.

# What can a TM do?

$L = \{ 0^i 1^j 0^k \mid i=j=k \}$  is not a CFL!

Can be decided by TM (see notes)



# What can a TM do?



A TM can do everything that can be done in a standard programming language  
(and vice versa)