

# Turing Machines & Computability

Lecture 19

# Course Trajectory



Seen lots of algorithms, what **can** be done.

But what **cannot** be done?

Need a more precise definition of what a computer / computation is

# Most General Computer?

Not all functions are computable, but *which are?*

Is there a *most general* model of computer, such that any “physically realizable” model of computation is subsumed by it?



Herbrand



Gödel



Church



Turing

General Recursive Functions

Lambda Calculus

Turing Machine

All these models turned out to be equivalent!



# Our Model So Far



Not been very precise about our computational model so far

Assumed that any integer and any array index fits into a “word”, and we can carry out arithmetic operations and load/store operations with them

But that requires our CPU to be infinitely large!

Does the model get too powerful?

We do want to allow access to arbitrarily large amounts of memory (input maybe arbitrarily large), but without using arbitrarily long addresses...

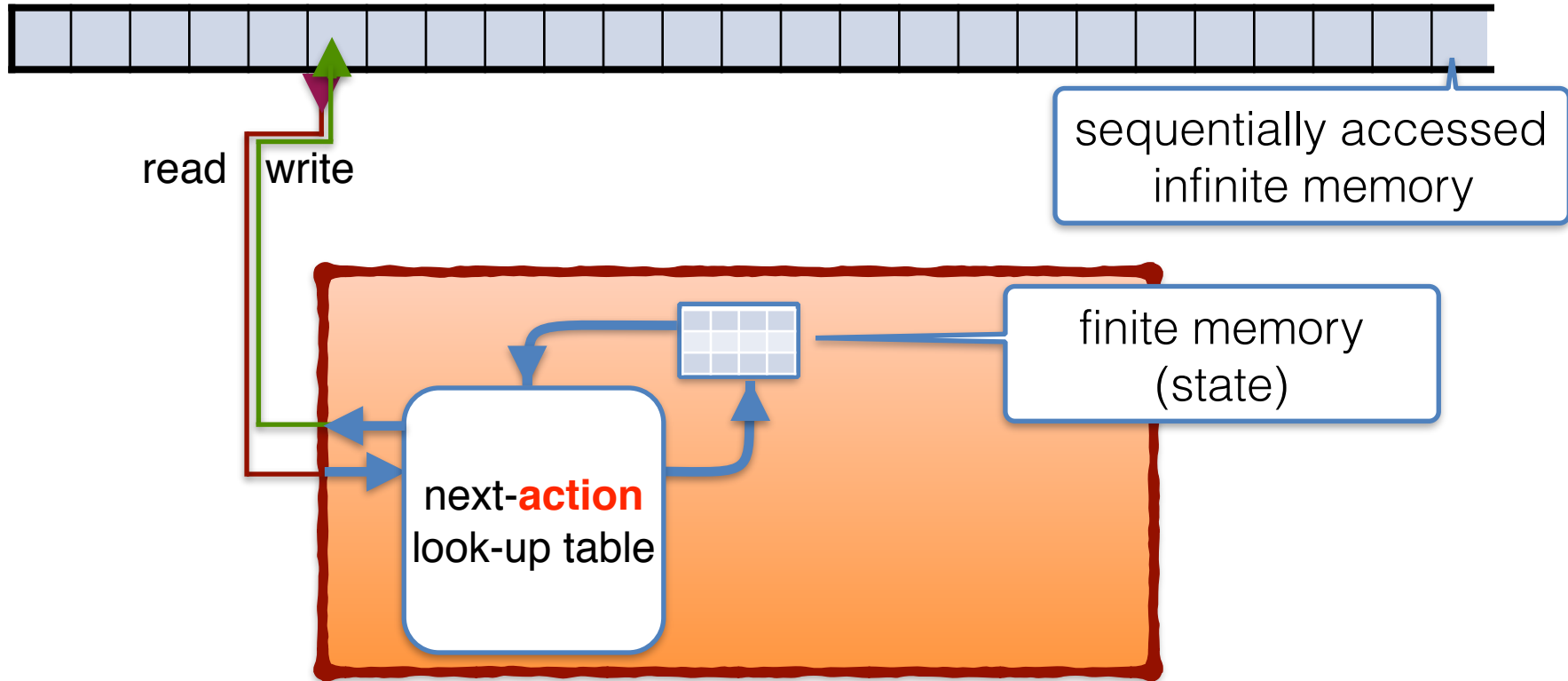
# Solution



Sequential access

# Turing Machine

move the *head*  
left or right  
by one cell



sequentially accessed  
infinite memory

finite memory  
(state)

next-action  
look-up table

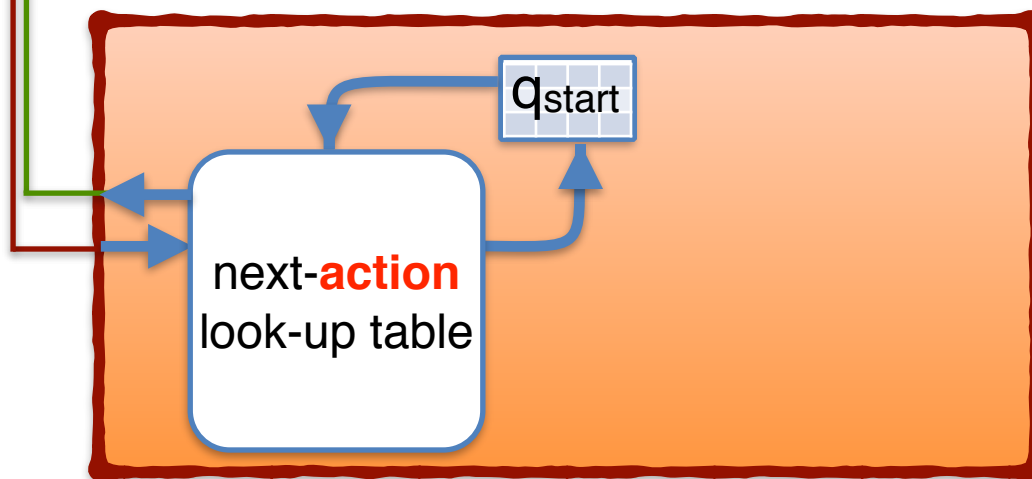
read write

# Turing Machine



input

blanks



next action:

e.g.,  
 $\delta(q, a) = (q', b, R)$   
change state to  $q'$ ,  
write  $b$  in the cell  
scanned by head,  
move head Right

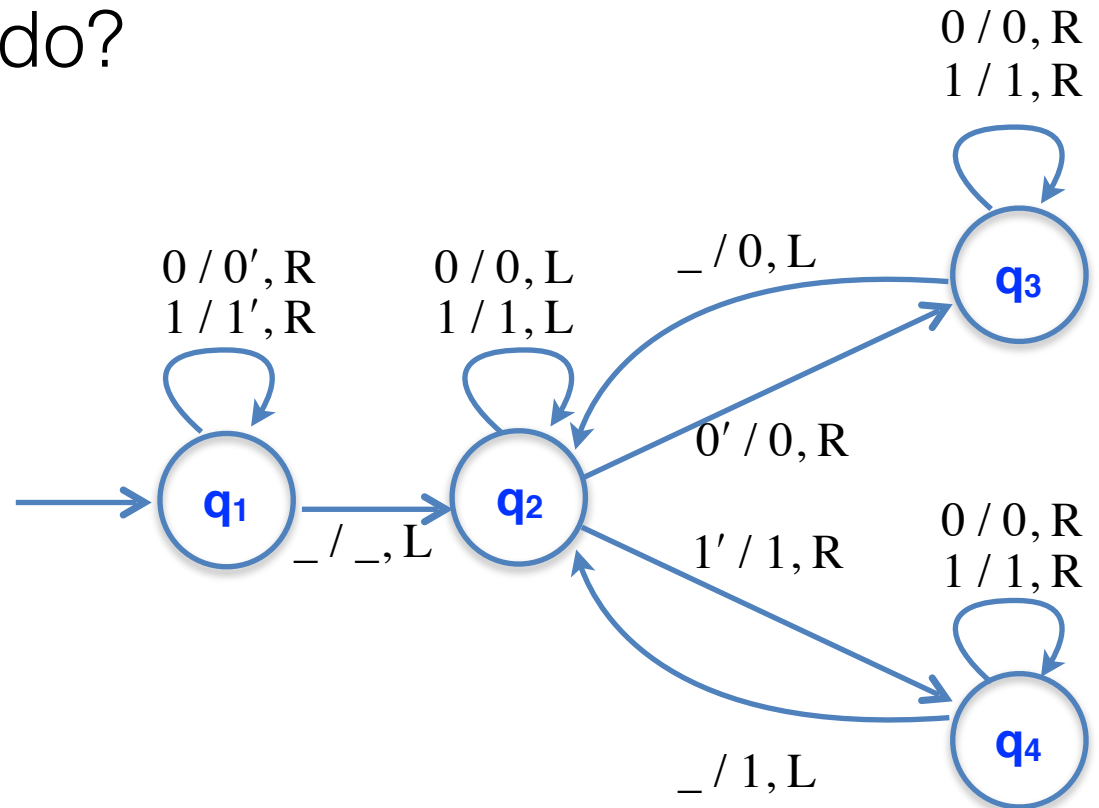
Initial configuration

# 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	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---

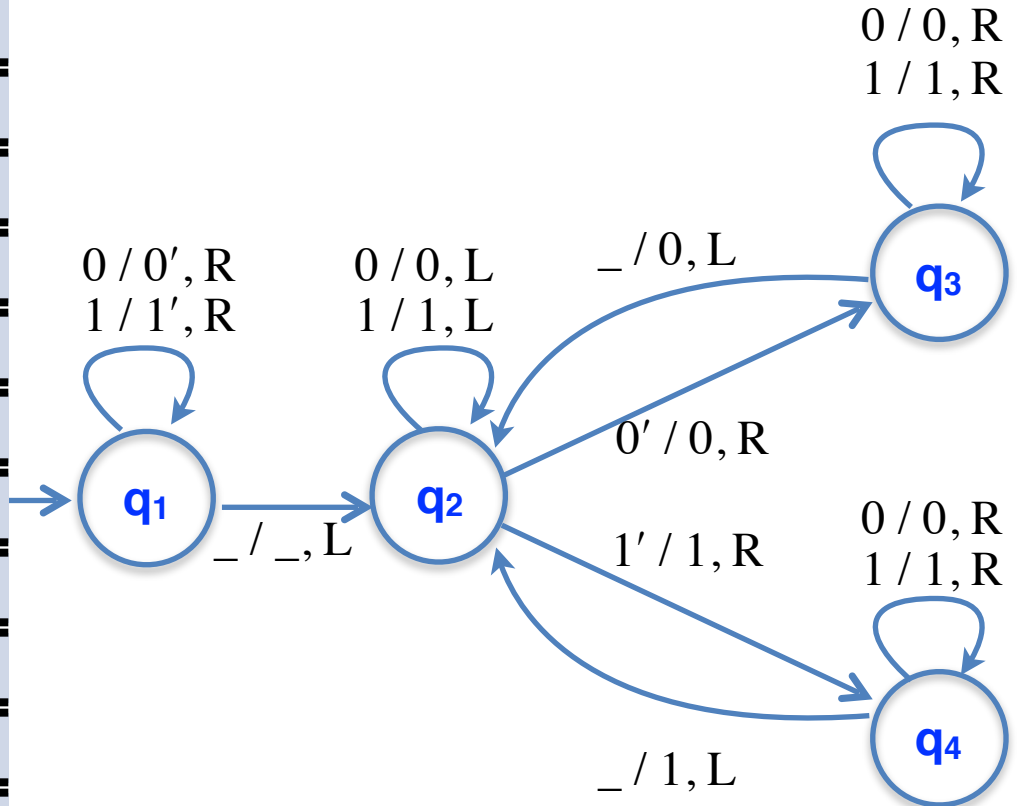
0'	0	1	1	0	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---

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

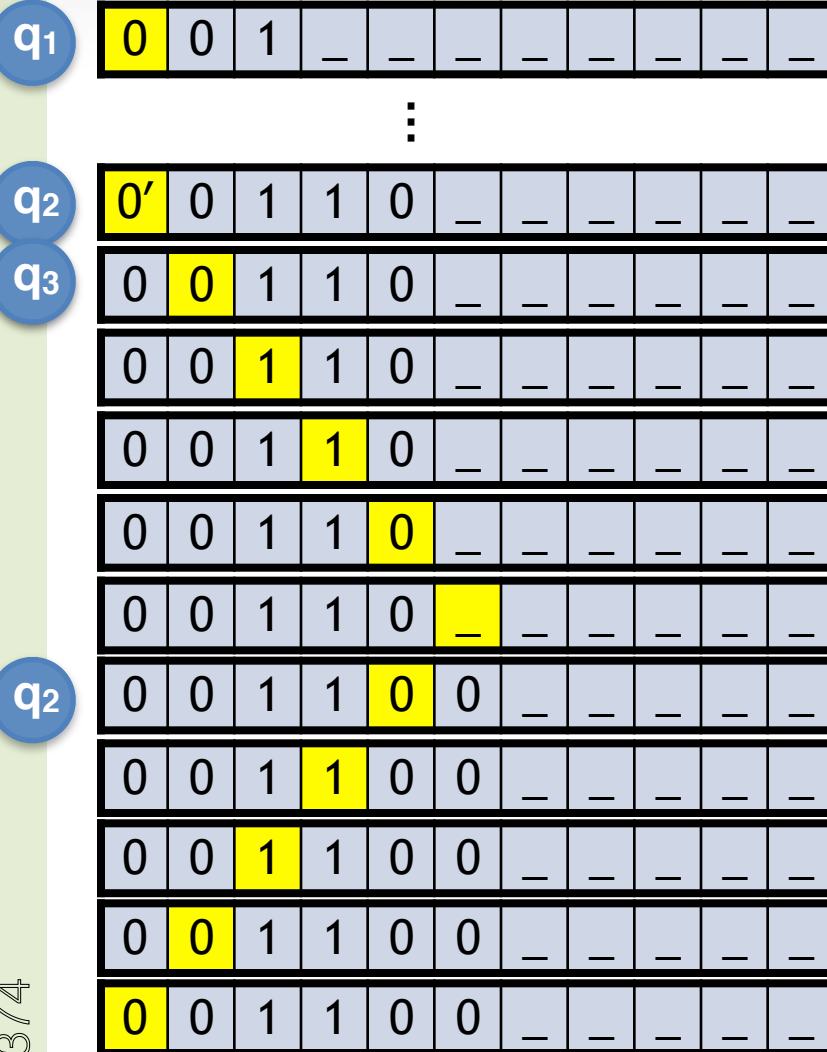
0'	0	1	1	0	_	_	_	_	_
----	---	---	---	---	---	---	---	---	---

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

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



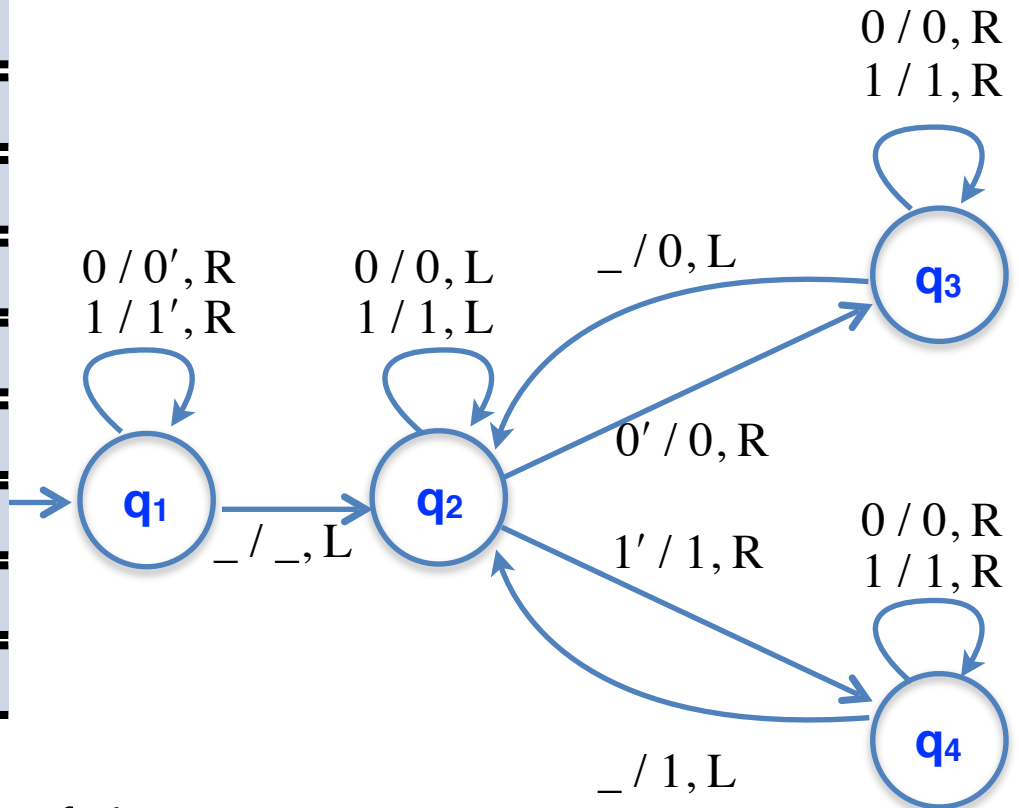
# Example



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

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

Maps  $w$  to  $ww^R$



Next?

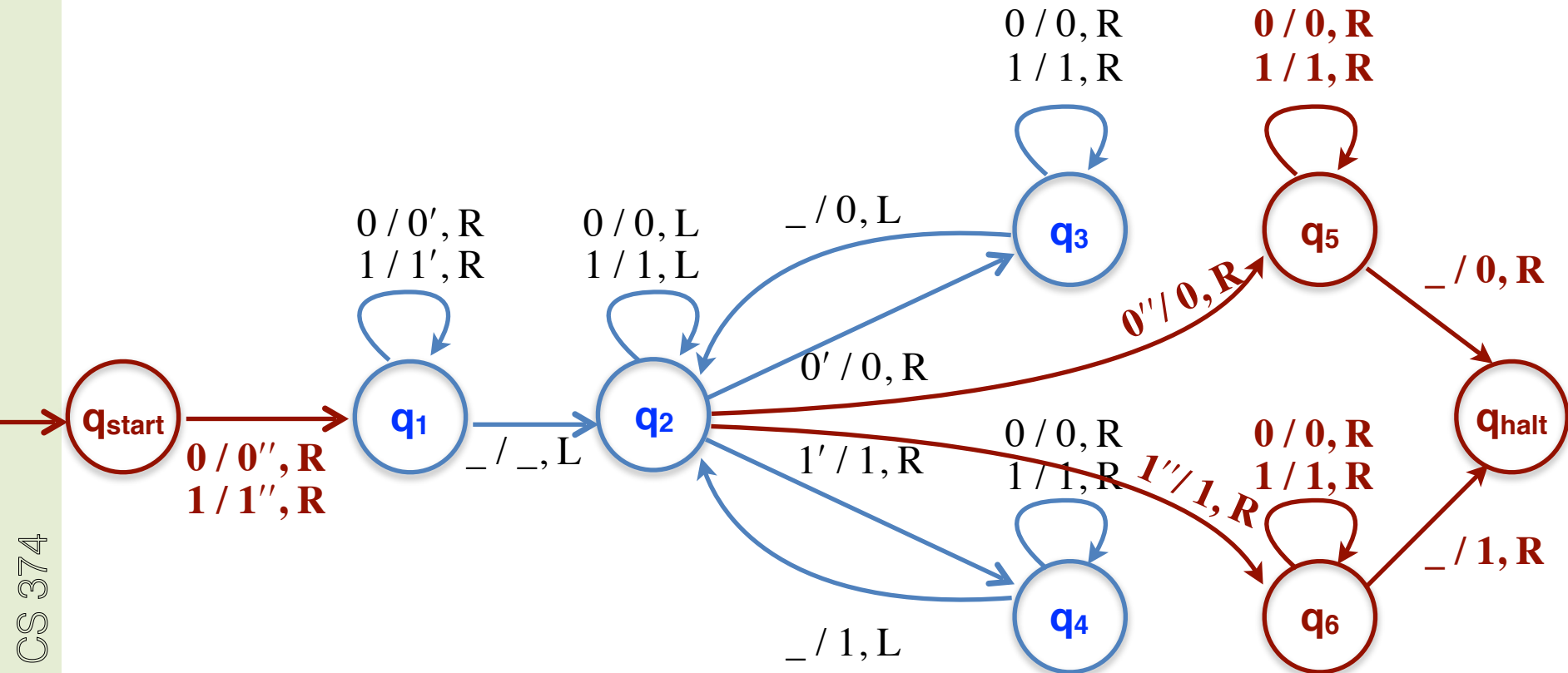
Crashes! Head moves out of the tape.

# Example

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

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

Maps  $w$  to  $ww^R$



# TM for Decision Problems



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

$Q$  is a finite set of states

$\Sigma$  is a finite input alphabet

$\Gamma$  is a finite tape alphabet. ( $\Sigma \subseteq \Gamma$ )

$B \in \Gamma - \Sigma$  is the blank symbol

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

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

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

A TM could write its output on the tape and enter  $q_{\text{halt}}$

But for decision problems (when the output is required to be a bit, yes/no), for convenience, we use *two halt states*,  $q_{\text{accept}}$  and  $q_{\text{reject}}$

# TM for Decision Problems



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

$Q$  is a finite set of states

$\Sigma$  is a finite input alphabet

$\Gamma$  is a finite tape alphabet. ( $\Sigma \subseteq \Gamma$ )

$B \in \Gamma - \Sigma$  is the blank symbol

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

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

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

No transition out of  $q_{\text{accept}}$  and  $q_{\text{reject}}$

$\delta : Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma$   
 $\rightarrow Q \times \Gamma \times \{L, R\}$

Convention: for  $(q, a) \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma$ , if  $\delta(q, a)$  is not specified, then define  $\delta(q, a)$  to be  $(q_{\text{reject}}, B, R)$

# ID: Instantaneous Description



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

state, tape-contents & head-location

Easy-to-read notation:  $xqy$

$x \in \Gamma^*$  : 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:  $q_{\text{start}} \langle \text{input} \rangle$

# 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

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

$\Rightarrow^*$  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



# Definition of Acceptance

$M$  accepts  $w$  iff  $q_{\text{start}} w \Rightarrow^* \alpha_1 q_{\text{accept}} \alpha_2$   
for some  $\alpha_1, \alpha_2 \in \Gamma^*$

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

# Deciding/Recognizing a Language

**Claim:** If  $L$  and  $\bar{L}$  are both recursively enumerable, then they are both decidable too

Say  $L = L(M_1)$  and  $\bar{L} = L(M_2)$ .

Can define a TM  $M$  which “simulates”  $M_1$  and  $M_2$  *in parallel* on the same input.

On any input, one of the simulated machines will eventually accept (and halt).

$M$  accepts or rejects accordingly.

How?  $M$  has only one tape...



# What a TM can do



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

Coming up:

A few programming tools for TMs and some equivalent models which are easier to program in

# Example

TM to accept palindromes in  $\{0,1\}^*$

Idea: Read first symbol into finite memory.  
Then move the tape head to the first symbol from right, and compare with symbol in memory.

If mismatch, reject.

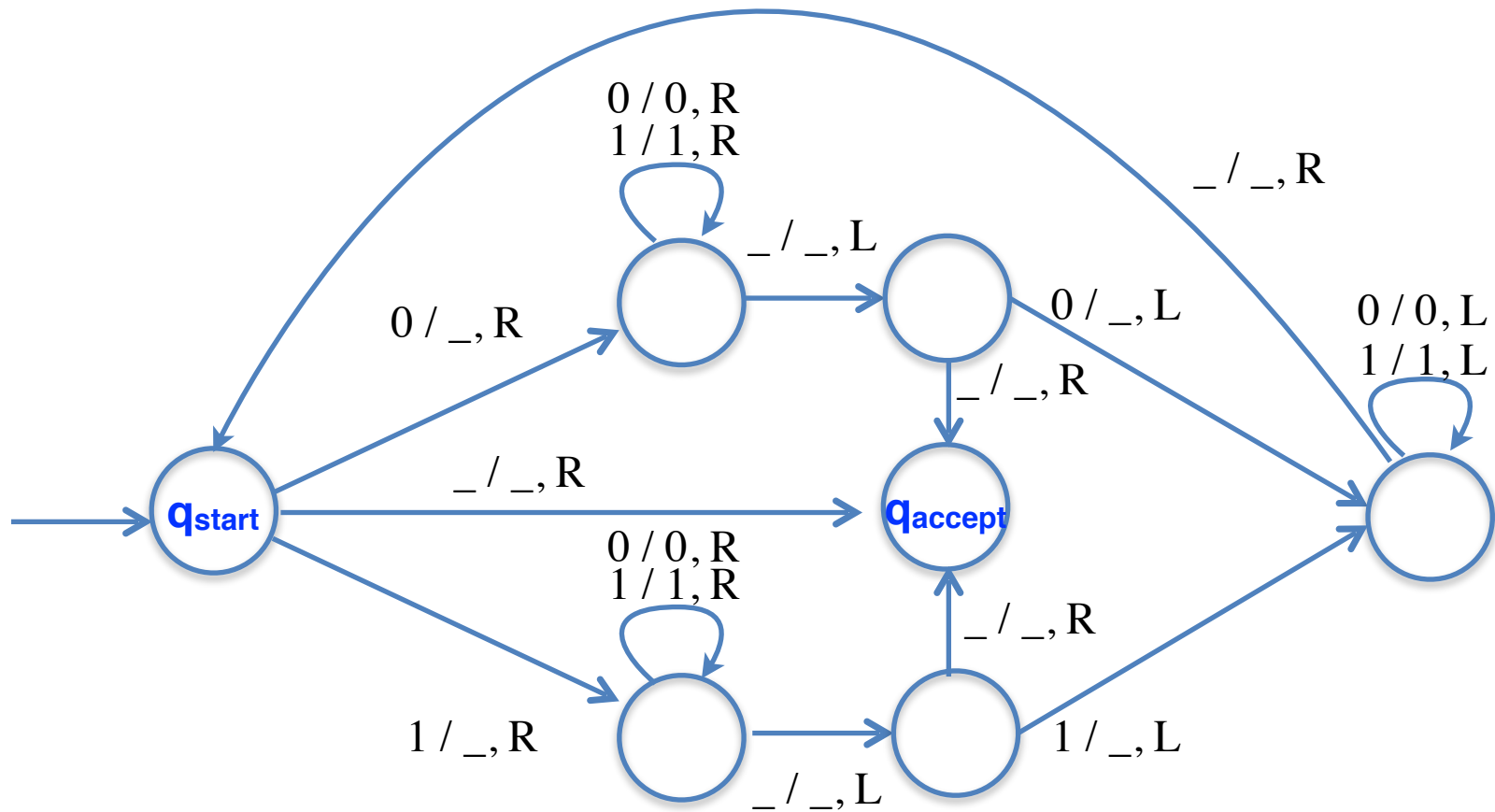
Else, return to the *next* symbol from left, read it into memory and compare with *next* symbol from right, etc.

How does the TM know which symbol is the “next”?  
Can't use finite memory to keep track of how many positions already processed.  
Need to use the tape itself.



# Example

TM to accept palindromes in  $\{0,1\}^*$



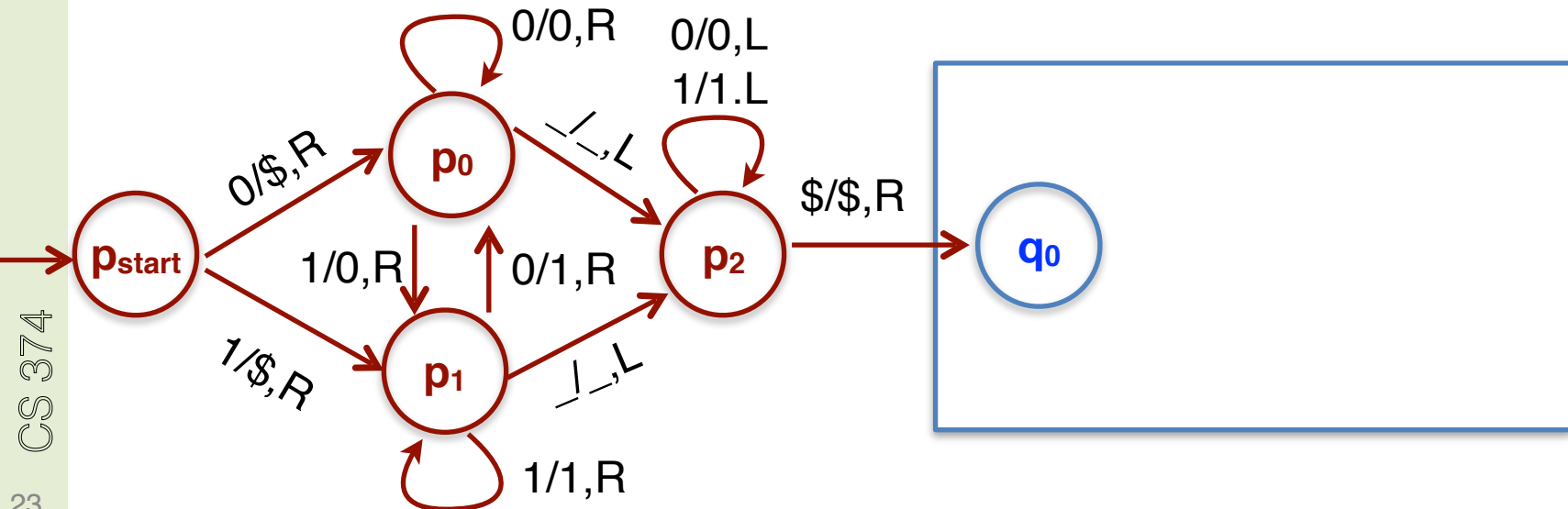
Fact: If TM not allowed to write on the tape, then can recognize only regular languages



# Avoiding Crashing

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

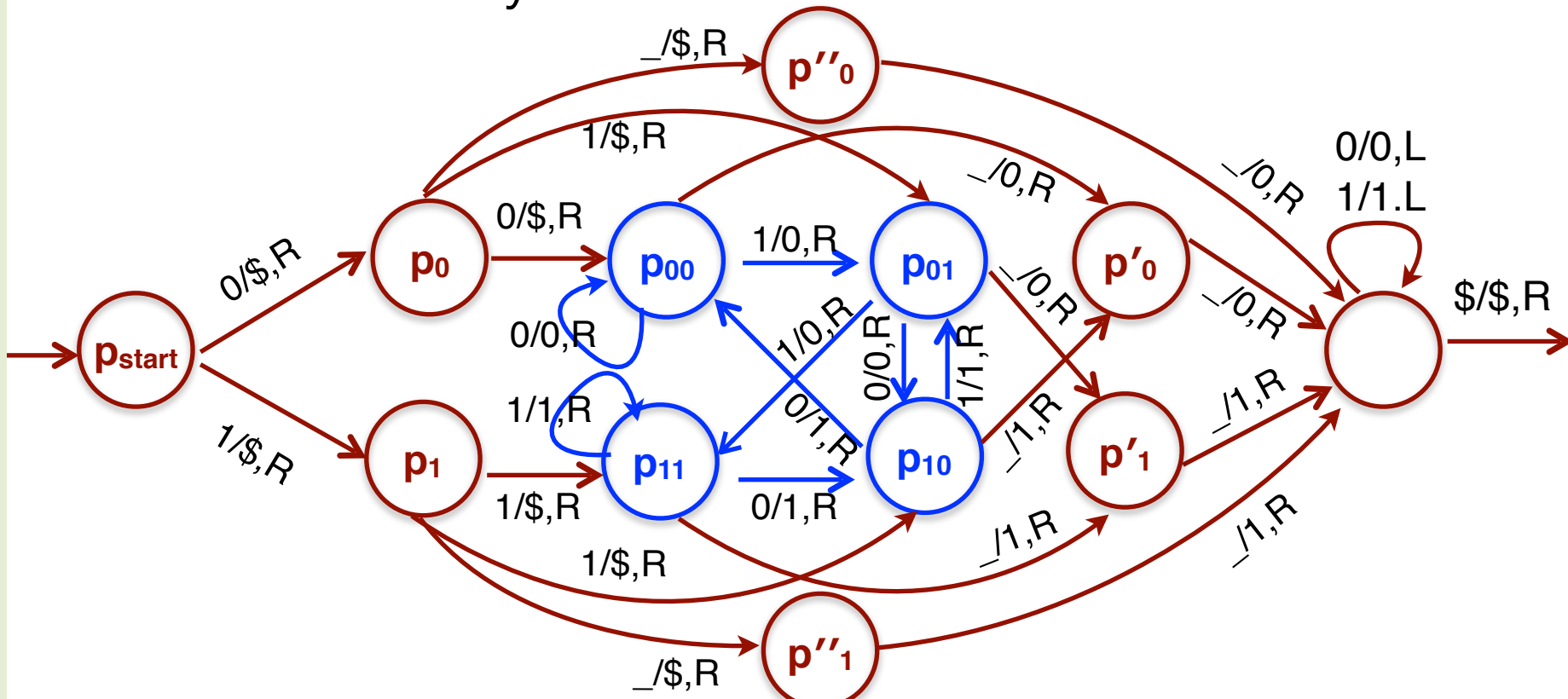
Idea: Rewrite input  $w$  to be  $\$w$ , place the head on the first symbol of  $w$  and run  $M$ . If head ever scans \$, move to  $q_{\text{reject}}$  (and move the head right)



# 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.

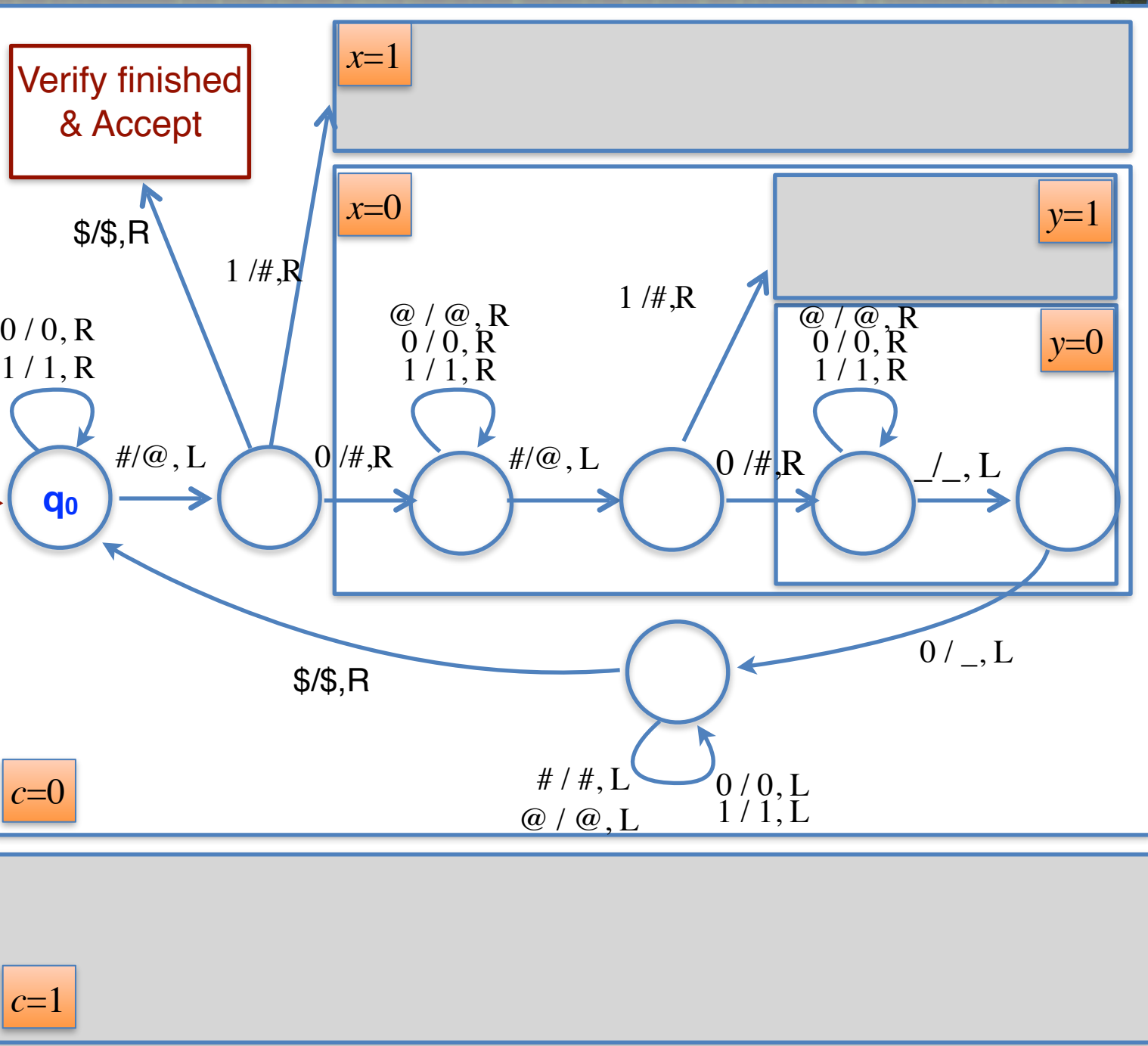
copy the digit to the left of second  $\#$  into finite state, and move  $\#$  left (replace old  $\#$  by  $@$ ). If no digit there, reject.

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



$c=0$

$c=1$

$x=1$

$x=0$

$y=1$

$y=0$

# Variants/Extensions



Adding more capabilities to TMs make them easier to program

But doesn't change what TMs can do:  
whatever the new variant can do, can be  
simulated in the original variant  
(with a lot more steps, sometimes)

# Extension: 2-Way Infinite Tape

.	.	-5	-4	-3	-2	-1	0	1	2	3	4	5	.	.
---	---	----	----	----	----	----	---	---	---	---	---	---	---	---

Simulating it in the original TM variant:

0	1	-1	2	-2	3	-3	4	-4	5	-5	6	.	.	.
---	---	----	---	----	---	----	---	----	---	----	---	---	---	---

Modify transitions:

Remember in control if +ve or -ve side of tape  
(contents of 0 cell will be marked).

If positive ( $> 1$ ):  $R \rightarrow RR$  &  $L \rightarrow LL$

If negative:  $R \rightarrow LL$  &  $L \rightarrow RR$

At 0:  $R \rightarrow R$  &  $L \rightarrow RR$



# Extension: Allowing Head to Stay

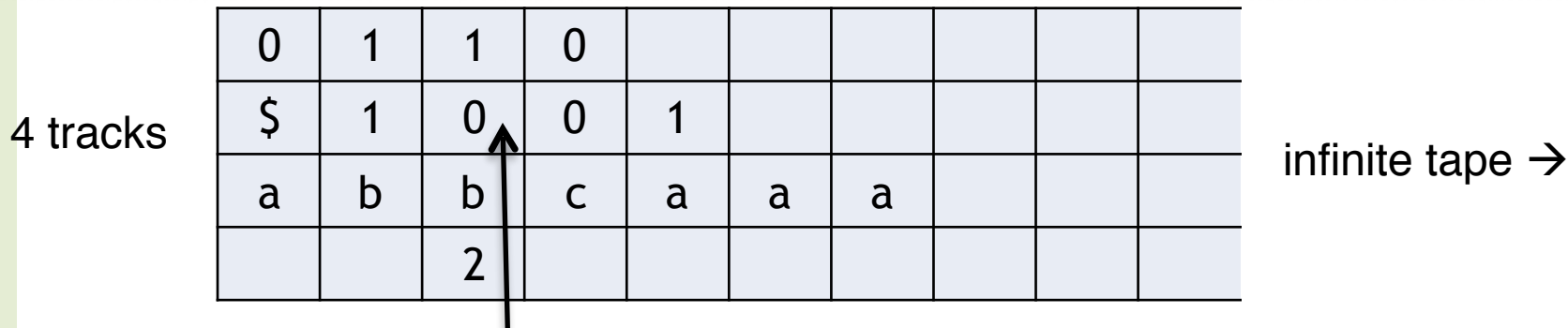
Suppose we allow the head movement to be in  $\{L,R,S\}$  (S for Stay)

Can simulate “staying” in the original TM model with two moves (and double the number of states)

If  $\delta(q,a) = (p,b,S)$ , then  
let  $\delta'(q,a) = (p',b,R)$  and  $\delta'(p',x) = (p,x,L)$  (for all  $x \in \Gamma$ )

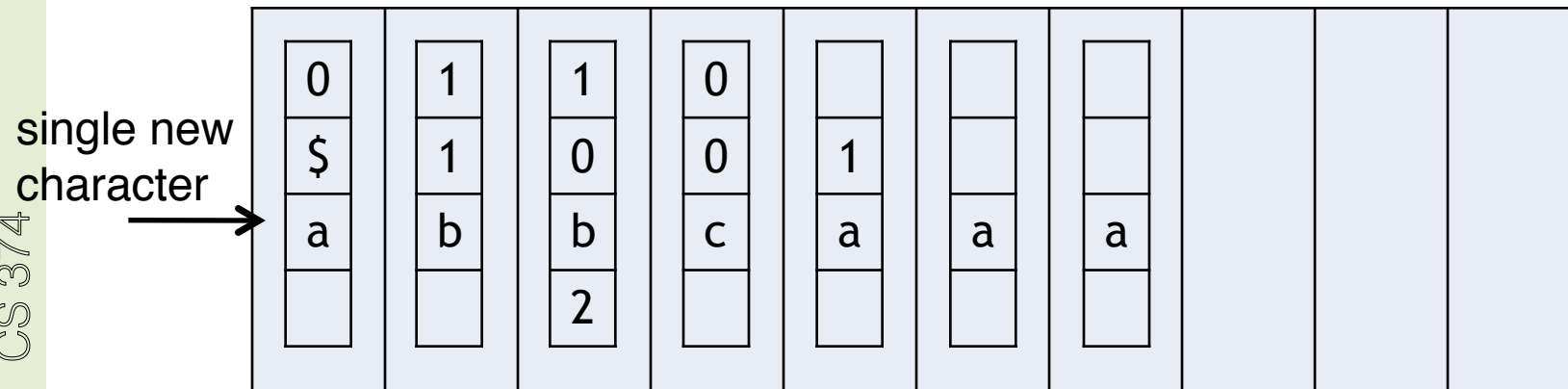


# Extension: multiple tracks



*M* can address any particular track in the cell it is scanning

Can simulate multiple tracks with a single track machine, using extra “stacked” characters:



# Extension: multiple tracks



4 tracks

0	1	1	0						
\$	1	0	0	1					
a	b	b	c	a	a	a			
		2							

infinite tape →

$$M: \delta(q, -, 0, -, -) = (p, -, -, -, 1, R)$$

“If in state  $q$  reading 0 on second track, then go to state  $p$ , write 1 on fourth track, and move right”

$$\text{Then in } M' \quad \delta\left(q, \begin{array}{|c|} \hline x \\ \hline 0 \\ \hline y \\ \hline z \\ \hline \end{array} \right) = \left(p, \begin{array}{|c|} \hline x \\ \hline 0 \\ \hline y \\ \hline 1 \\ \hline \end{array}, R\right) \quad \text{for every } x, y, z \in \Gamma$$



# Extension: multiple tapes

## $k$ -tape TM

$k$  different (2-way infinite) tapes

$k$  different independently controllable heads

input initially on tape 1; tapes 2, 3, ...,  $k$ , blank.

Single move:

read symbols under all heads

print (possibly different) symbols under heads

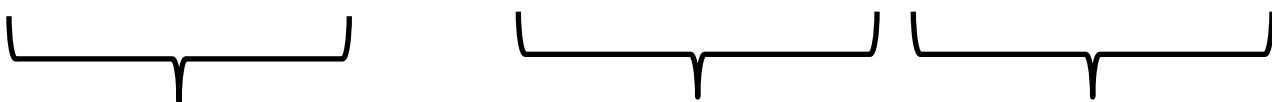
move all heads (possibly in different directions)

go to new state



# k-tape TM transition function

$$\delta(q, a_1, a_2, \dots, a_k) = (p, b_1, b_2, \dots, b_k, D_1, D_2, \dots, D_k)$$


  
 Symbols scanned on the  $k$  different tapes      Symbols to be written on the  $k$  different tapes      Directions to be moved ( $D_i$  is one of L, R, S)

Utility of multiple tapes:  
makes programming a whole lot easier

Example:  $L = \{ w\#w^R \mid w \in \{0,1\}^* \}$

With single tape, need  $\Omega(n^2)$  steps

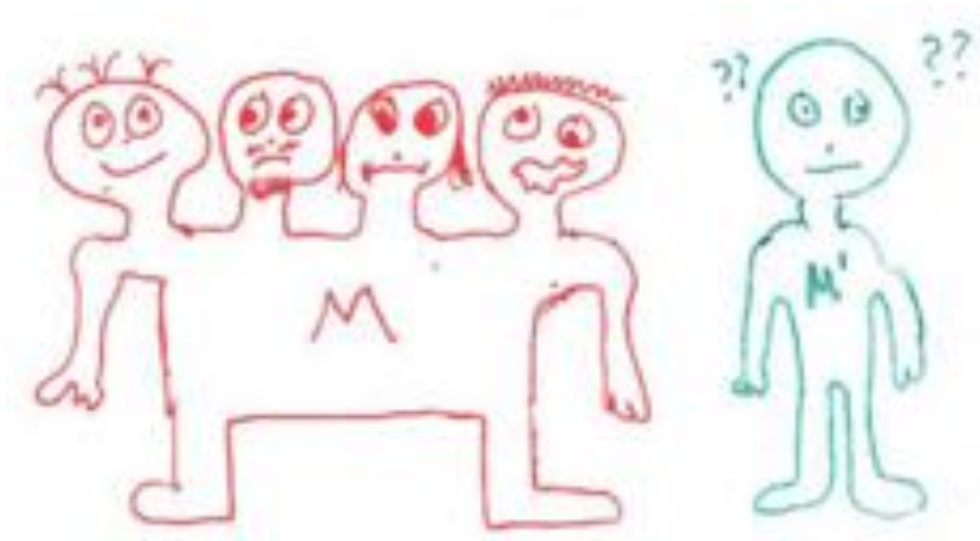
1	1	0	0	1	0	#	0	1	0	0	1	1		
1'	1	0	0	1	0	#								

With 2 tapes,  
 $n+1$  steps:  
copy till # to  
2<sup>nd</sup> tape.  
Scan it  
backwards  
after that

# Can't compute more with $k$ tapes

Theorem: If  $L$  is accepted by a  $k$ -tape TM  $M$ , then  $L$  is accepted by some 1-tape TM  $M'$ .

Idea:  $M'$  uses  $k$  tracks to simulate tapes of  $M$



BUT....  
M has  $k$  heads!

How can  $M'$  be in  
 $k$  places at once?

$M'$  will use  $2k$  tracks to simulate tapes+heads of  $M$