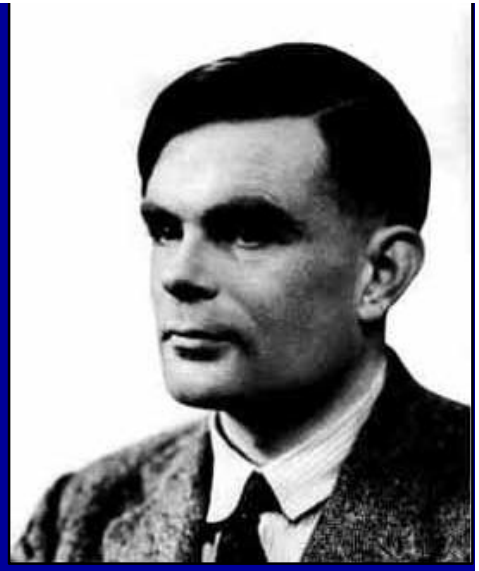
TURING MACHINES

Turing Machines (TM)

- The Turing machine is the ultimate model of computation.

Alan Turing (1912–1954), British mathematician/engineer and one of the most influential scientists of the last century.

Turing Machines (TM)



In 1936, Turing introduced his abstract model for computation.

The Turing machine model has become the standard in theoretical computer science. Think of a Turing Machine as a DPDA that can move freely through its stack (= tape).

Turing Machines (TM)

Alan Turing was one of the founding fathers of CS.

- His computer model –the Turing Machine– was inspiration/premonition of the electronic computer that came two decades later
- Was instrumental in cracking the Nazi Enigma cryptosystem in WWII
- Invented the “Turing Test” used in AI
- Legacy: The Turing Award. Pre-eminent award in Theoretical CS

Turing Machines (TM)

A Thinking Machine

First Goal of Turing's Machine: A model that can compute anything that a human can compute. Before invention of electronic computers the term "computer" actually referred to a *person* who's line of work is to calculate numerical quantities!

As this is a philosophical endeavor, it can't really be proved.

Turing's Thesis: Any "algorithm" can be carried out by one of his machines

Turing Machines (TM)

A Thinking Machine

- Second Goal of Turing's Machine: A model that's so simple, that can actually prove interesting epistemological results. Eyed Hilbert's 10th problem, as well as a computational analog of Gödel's Incompleteness Theorem in Logic.
- Philosophy notwithstanding, Turing's programs for cracking the Enigma cryptosystem prove that he really was a true hacker! Turing's machine is actually easily programmable, if you really get into it. Not practically useful, though…

Turing Machines (TM)

A Thinking Machine

Imagine a super-organized, obsessive-compulsive human computer. The computer wants to avoid mistakes so everything written down is completely specified one letter/number at a time. The computer follows a finite set of rules which are referred to every time another symbol is written down. Rules are such that at any given time, only one rule is active so no ambiguity can arise. Each rule activates another rule depending on what letter/number is currently read.

Turing Machines (TM)

A Thinking Machine: Example: Successor Program

Sample Rules:

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

Let's see how they are carried out on a piece of paper that contains the *reverse* binary representation of 47: 111101

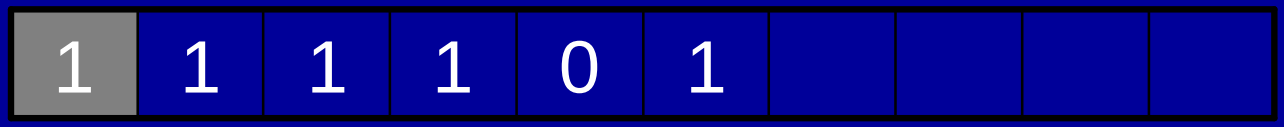
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



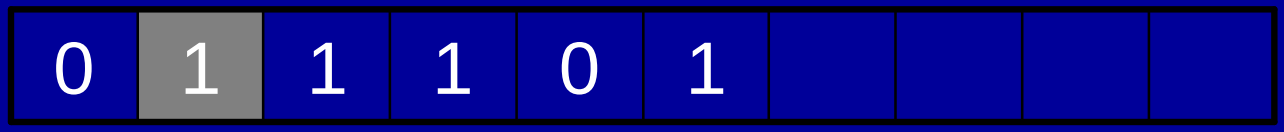
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



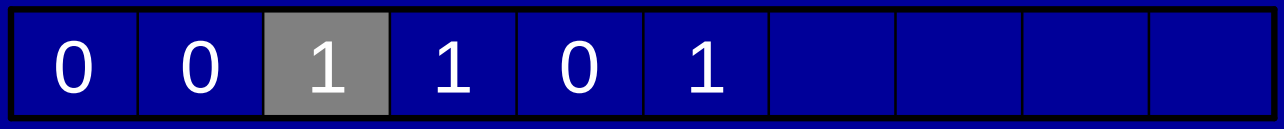
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



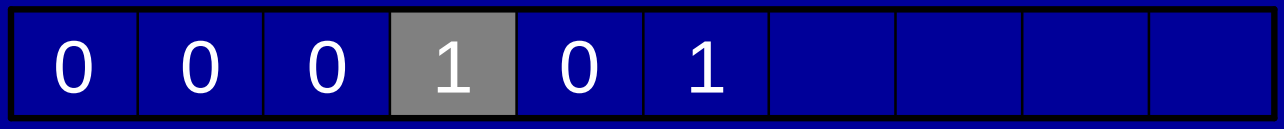
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



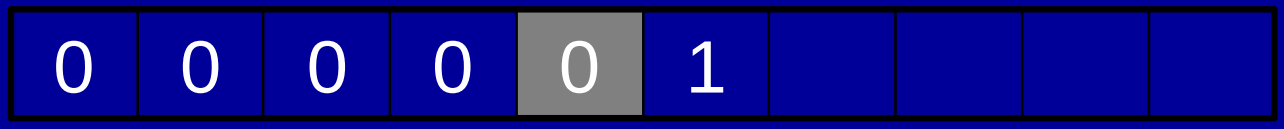
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

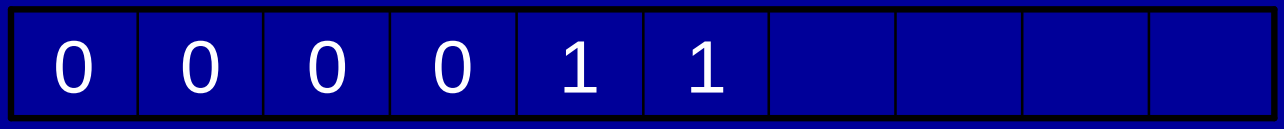


A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



A Thinking Machine: Example: Successor Program

So the successor's output on 111101 was 000011 which is the reverse binary representation of 48.

Similarly, the successor of 127 should be 128:

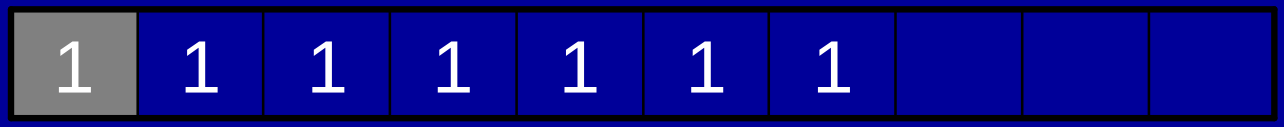
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



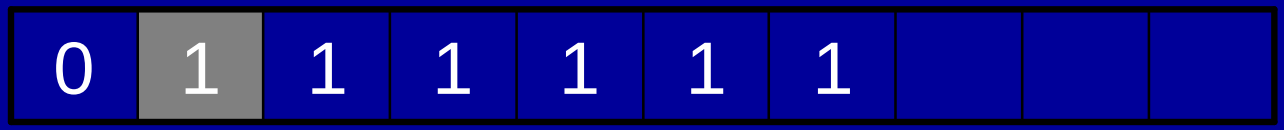
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



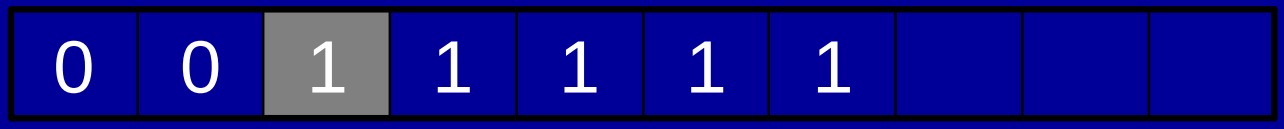
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



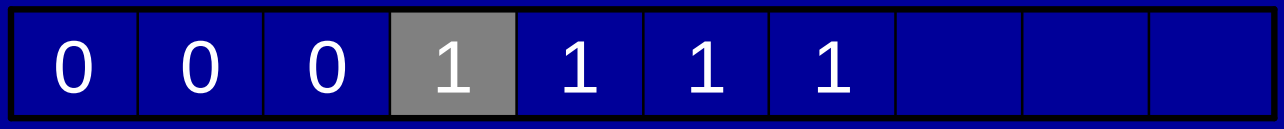
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



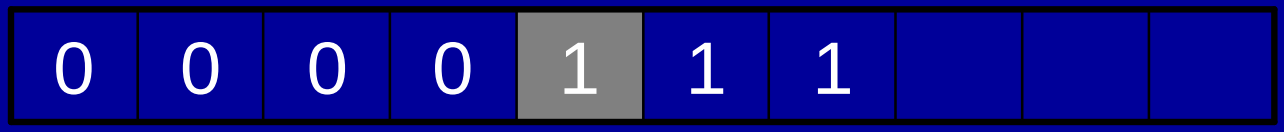
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



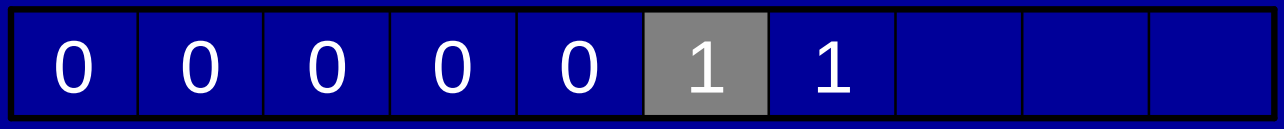
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;write 1, HALT!



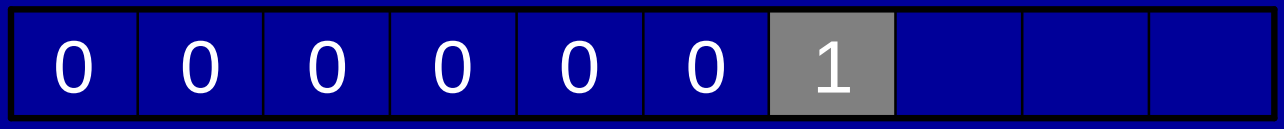
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



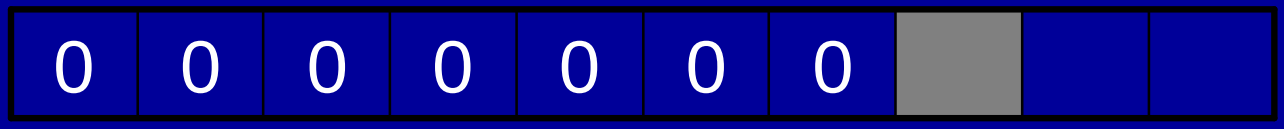
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



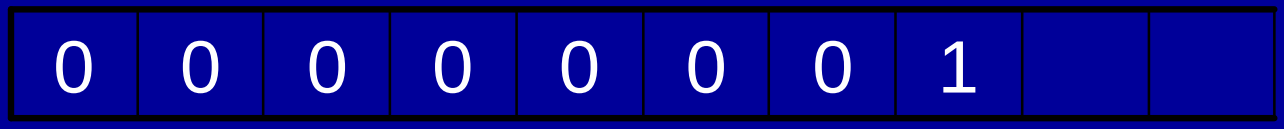
Turing Machines (TM)

A Thinking Machine: Example: Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ;, write 1, HALT!



Turing Machines (TM)

A Thinking Machine: Example: Successor Program

It was hard for the ancients to believe that *any* algorithm could be carried out on such a device. For us, it's much easier to believe, especially if you've programmed in assembly!

However, ancients did finally believe Turing when Church's lambda-calculus paradigm (on which lisp programming is based) proved equivalent!

Turing Machines (TM)

A Turing Machine (**TM**) is a device with a finite amount of *read-only* "*hard*" memory (states), and an unbounded amount of read/write tape-memory. There is no separate input. Rather, the input is assumed to reside on the tape at the time when the TM starts running.

Just as with Automata, TM's can either be input/output machines (compare with Finite State Transducers), or yes/no decision machines.

Turing Machines (TM)

Church Turing thesis

Church Turing Thesis: all reasonable models of computation can be simulated by a (single tape) Turing machine.

If we want to investigate what we can and can not do using computers, it is sufficient to study the Turing machine model.

Turing Machines (TM)

A Comparison with FA

TM can both write to and read from the tape

The head can move left and right

The string doesn’t have to be read entirely

Accept and Reject take immediate effect

Turing Machines (TM)

A Comparison with FA and PDA

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA			
PDA			
TM			

Turing Machines (TM)

A Comparison with FA and PDA

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA			
TM			

Turing Machines (TM)

A Comparison with FA and PDA

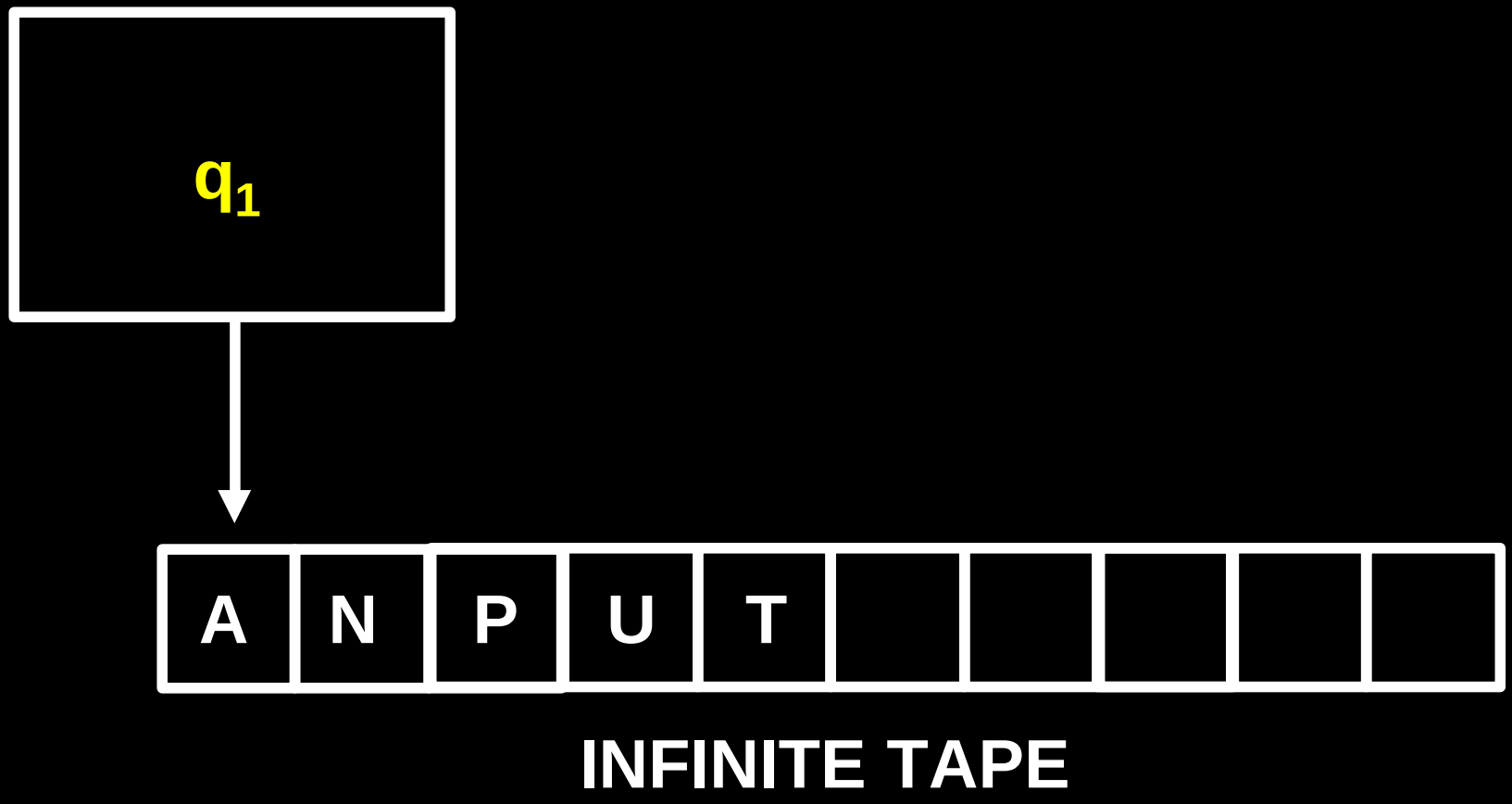
Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM			

Turing Machines (TM)

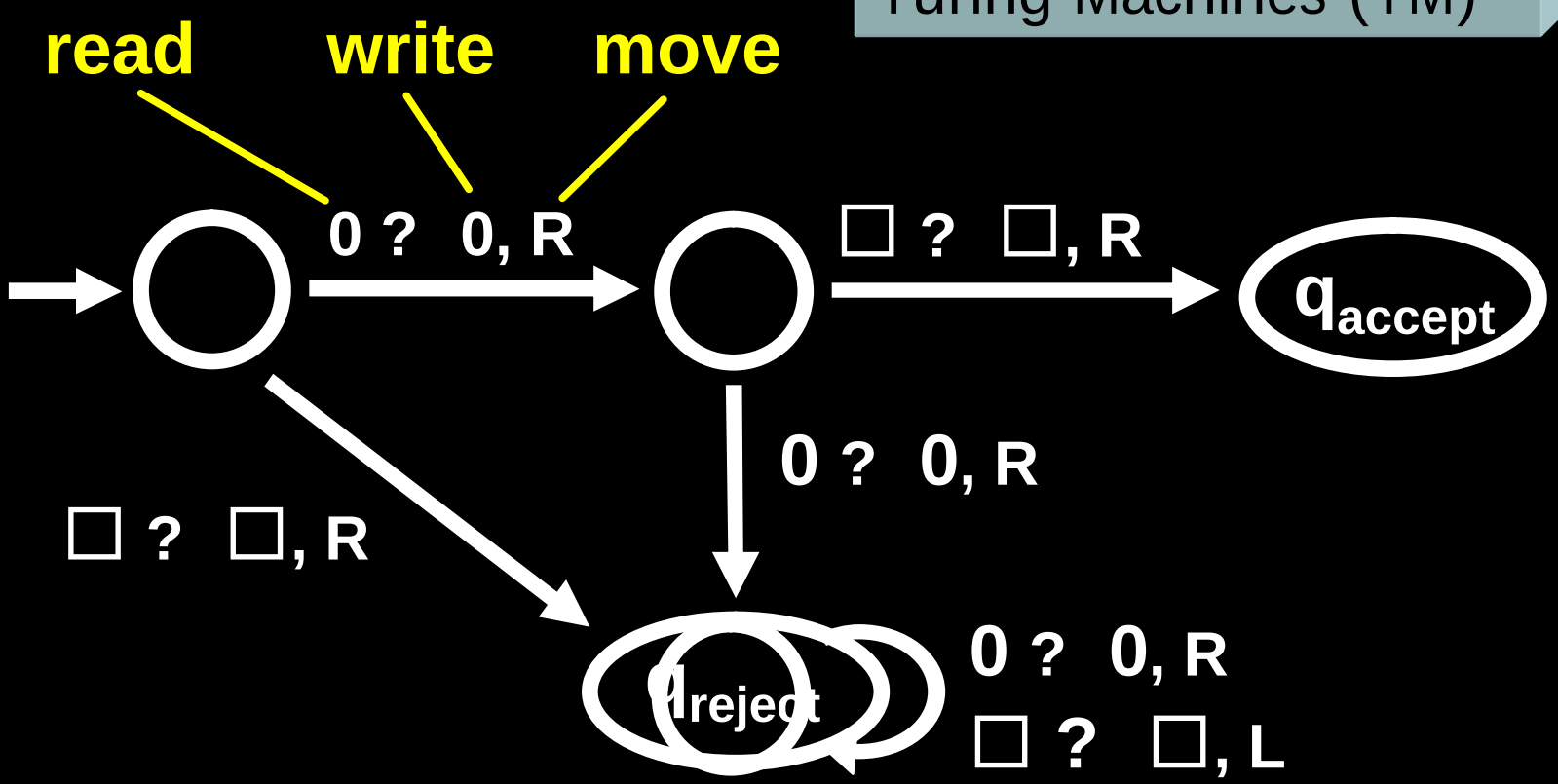
A Comparison with FA and PDA

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM	No	1-way infinite tape. 1 cell access per step.	Yes (but will also allow crashes)

Turing Machines (TM)



Turing Machines (TM)



Turing Machines (TM)

Notations

An edge from the state *p* to the state *q* labeled by …

- *a*→*b*,D means if in state *p* and tape head reading *a*, replace *a* by *b* and move in the direction D, and into state *q*
- *a*→D means if in state *p* and tape head reading *a*, don't change *a* and move in the direction D, and into state *q*
- *a*|*b*|…|*z* → … means that given that the tape head is reading any of the pipe separated symbols, take same action on any of the symbols

Turing Machines (TM)

Notations

A TM's next action is completely determined by current state and symbol read, so can predict all of future actions if know:

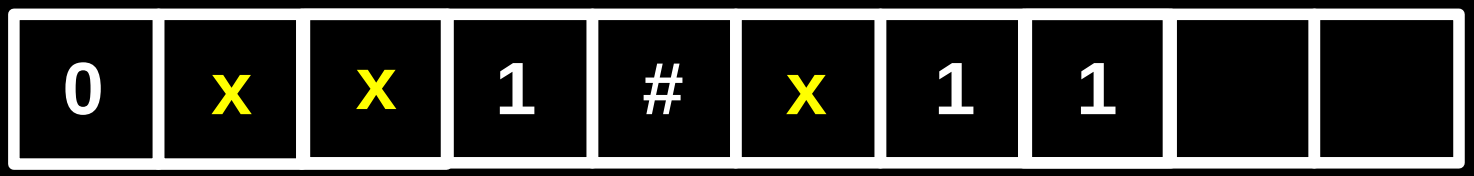
1. current state
2. current tape contents
3. current position of TM's reading "head"

Turing Machines (TM)

Example:

STATE

$q_{0,\ F}\ q_{1,\ \text{FIND}\ \neq}\ q_{\#,\ F}\ q_{0,\ F}\ q_{1,\ \text{FIND}\ \square}\ q_{\text{GO LEFT}}$



Testing membership in B = { w#w | w Î {0,1}* }

Turing Machines (TM)

Definition

A Turing Machine is a 7-tuple
$T = (Q, S, G, \delta, q_0, q_{accept}, q_{reject})$, where:

Q is a finite set of states

S is the input alphabet, where $\square \notin S$

G is the tape alphabet, where $\square \in G$ and $S \subseteq G$

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

$q_0 \in Q$ is the start state

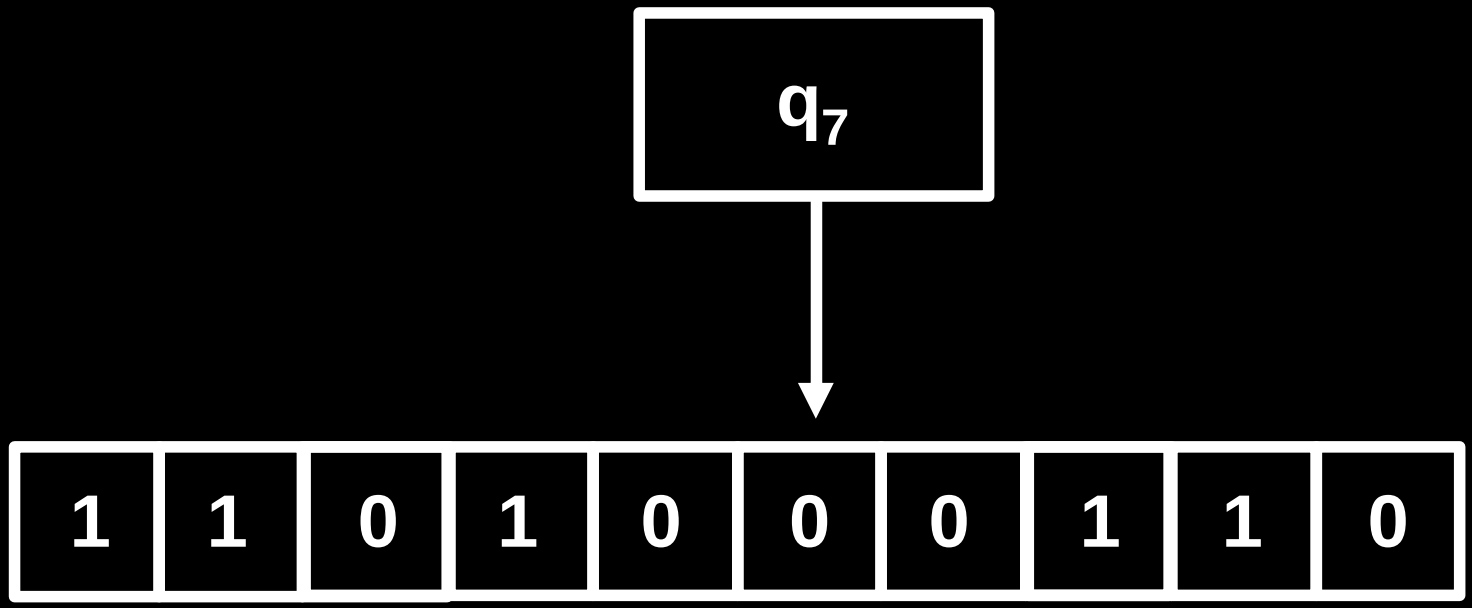
$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

Turing Machines (TM)

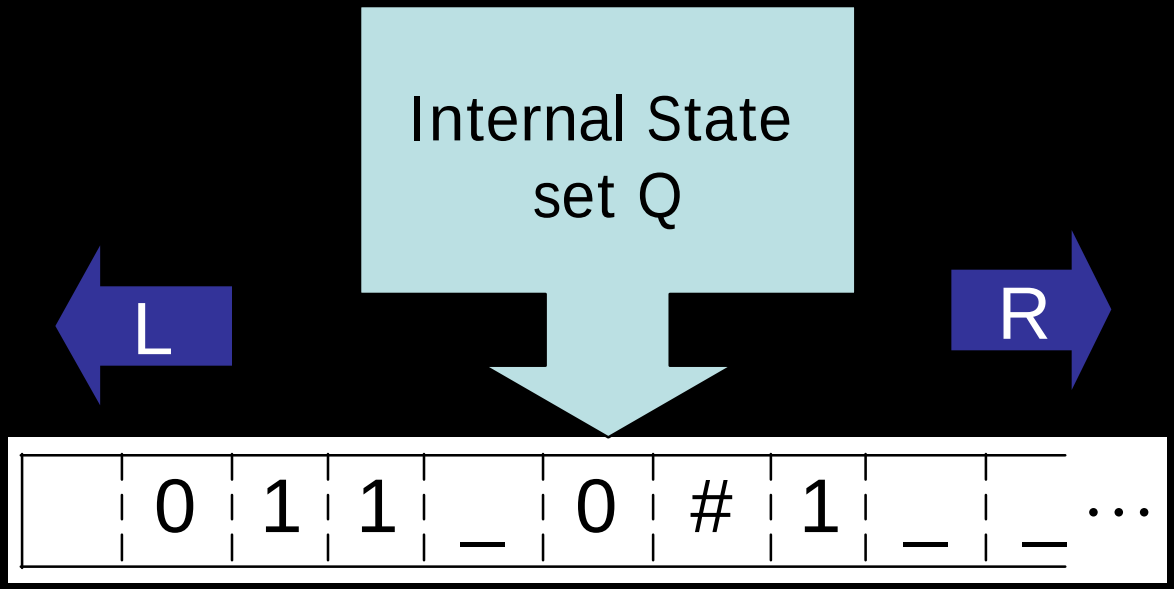
CONFIGURATIONS

$11010q_700110$



Turing Machines (TM)

Informal Description

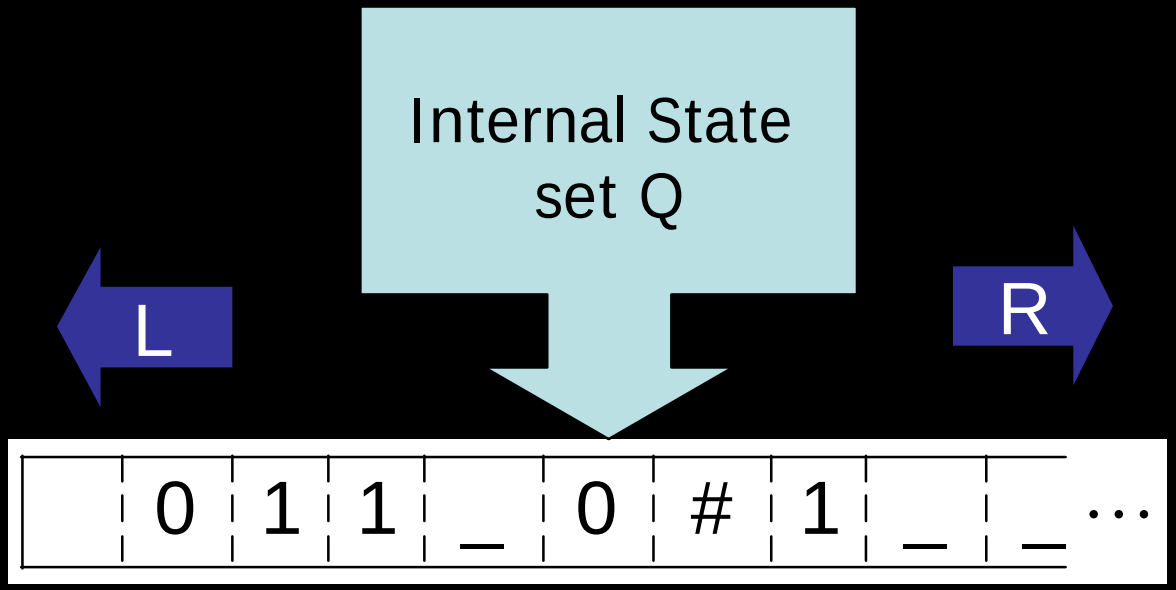


At every step, the head of the TM M reads a letter x_i from the one-way infinite tape.

Depending on its state and the letter x_i, the TM
- writes down a letter,
- moves its read/write head Left or Right, and
- jumps to a new state in Q.

Turing Machines (TM)

Informal Description



At every step, the head of the TM M reads a letter x from the tape of size ? .

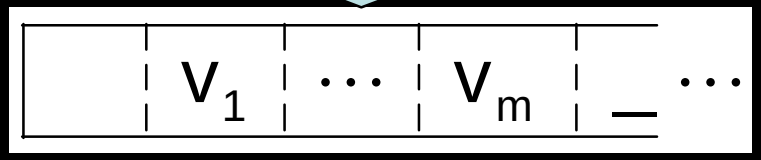
Depending on x and q the, the transition function value d(q,x) = (r,y,d) tells the TM to replace the letter a by b, move its head in direction d∈{L,R}, and change its internal state to r.

Turing Machines (TM)

Output Convention

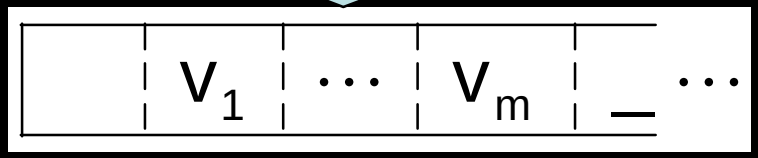
The computation can proceed indefinitely, or the machines reaches one of the two halting states:

accept state t



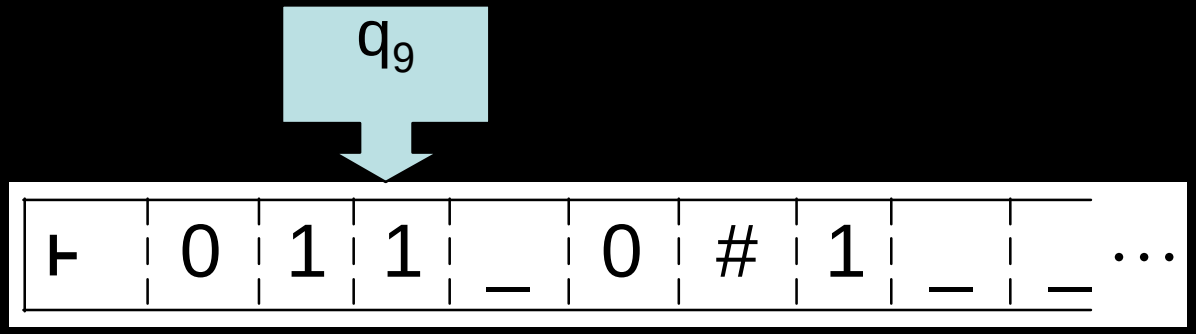
or

reject state r

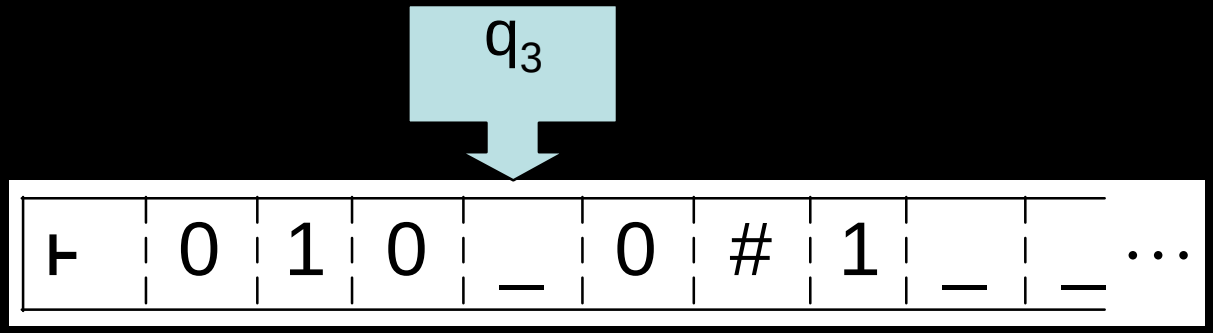


Turing Machines (TM)

Transitions in Action



$d(q_9,1) = (q_3,0,R)$ gives…



Notation: $(q_9,\vdash 011\_0\#1\ \_^{?}, 3)$? $(q_3,\vdash 010\_0\#1\ \_^{?}, 4)$

Turing Machines (TM)

A TM recognizes a language if it accepts all and only those strings in the language

A language is called Turing-recognizable or recursively enumerable if some TM recognizes it

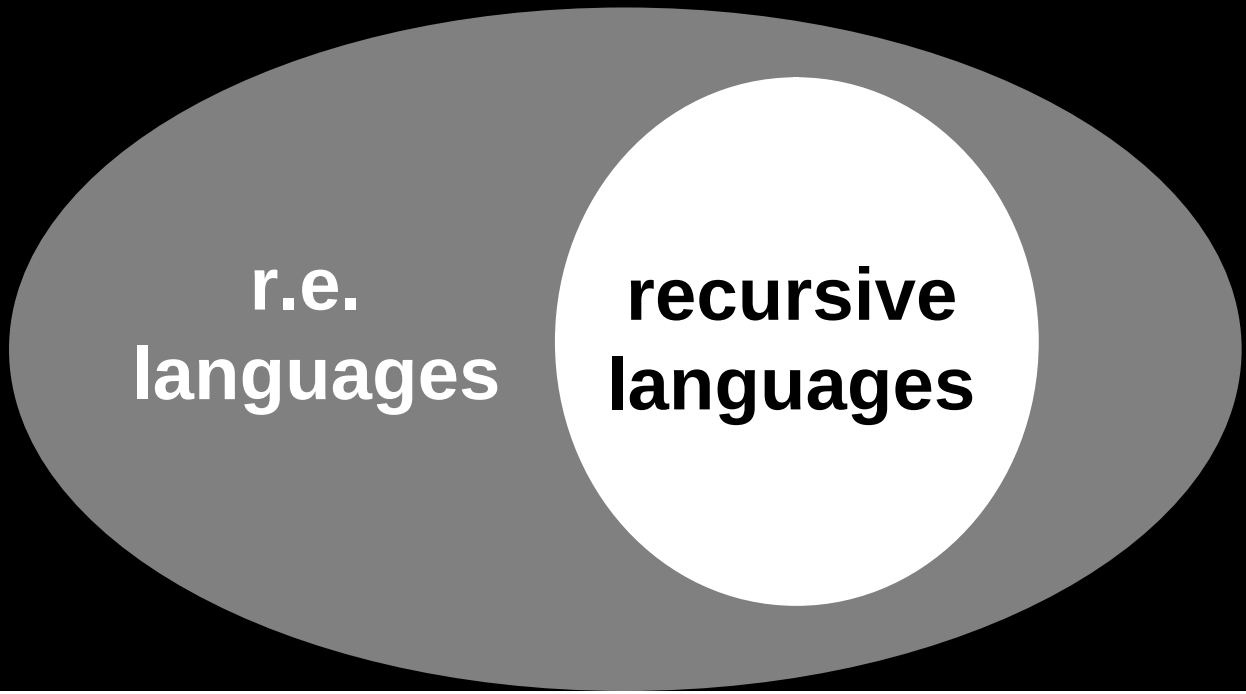
A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

A language is called decidable or recursive if some TM decides it

Turing Machines (TM)

A language is called Turing-recognizable or recursively enumerable if some TM recognizes it

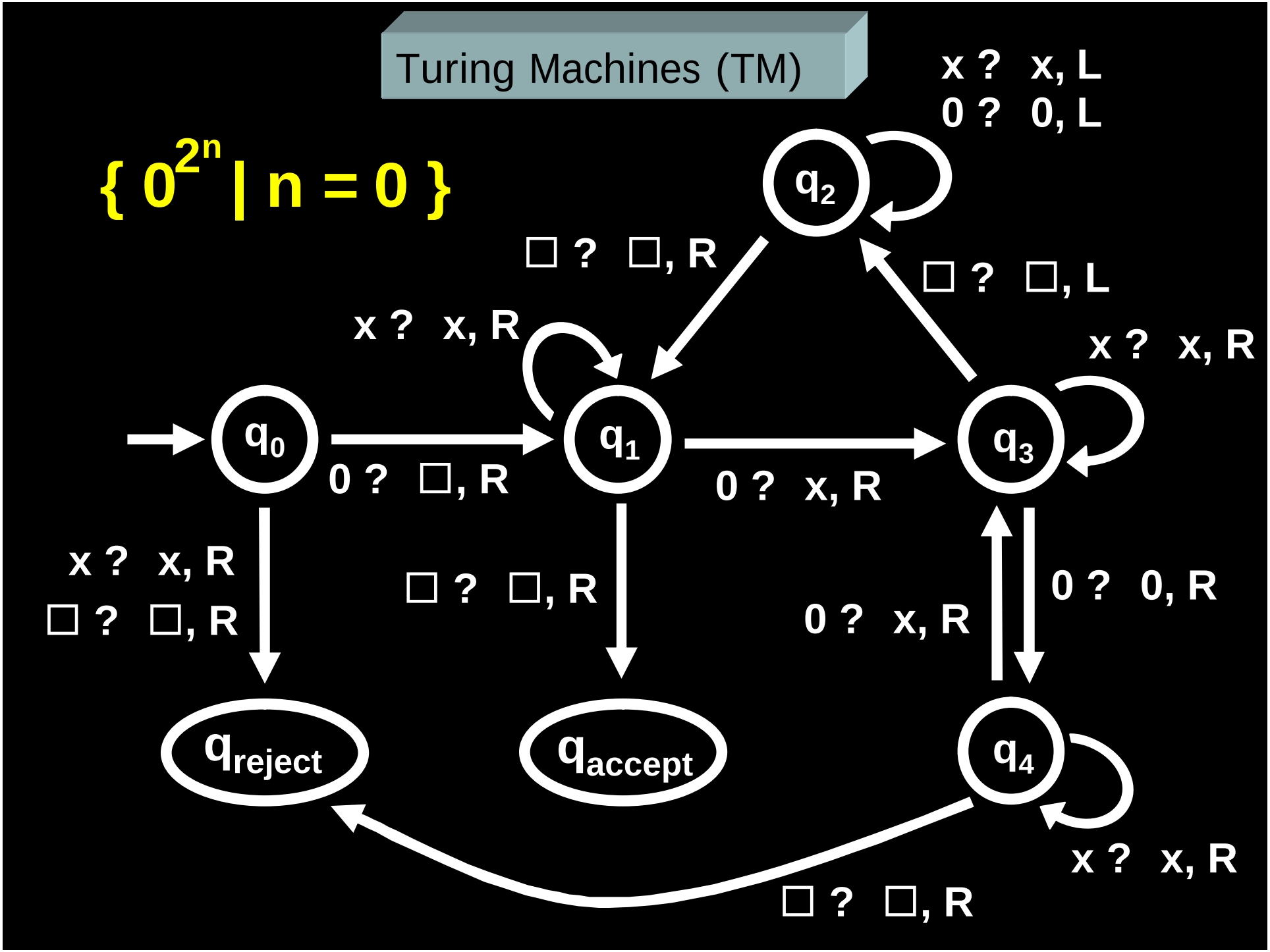
A language is called decidable or recursive if some TM decides it

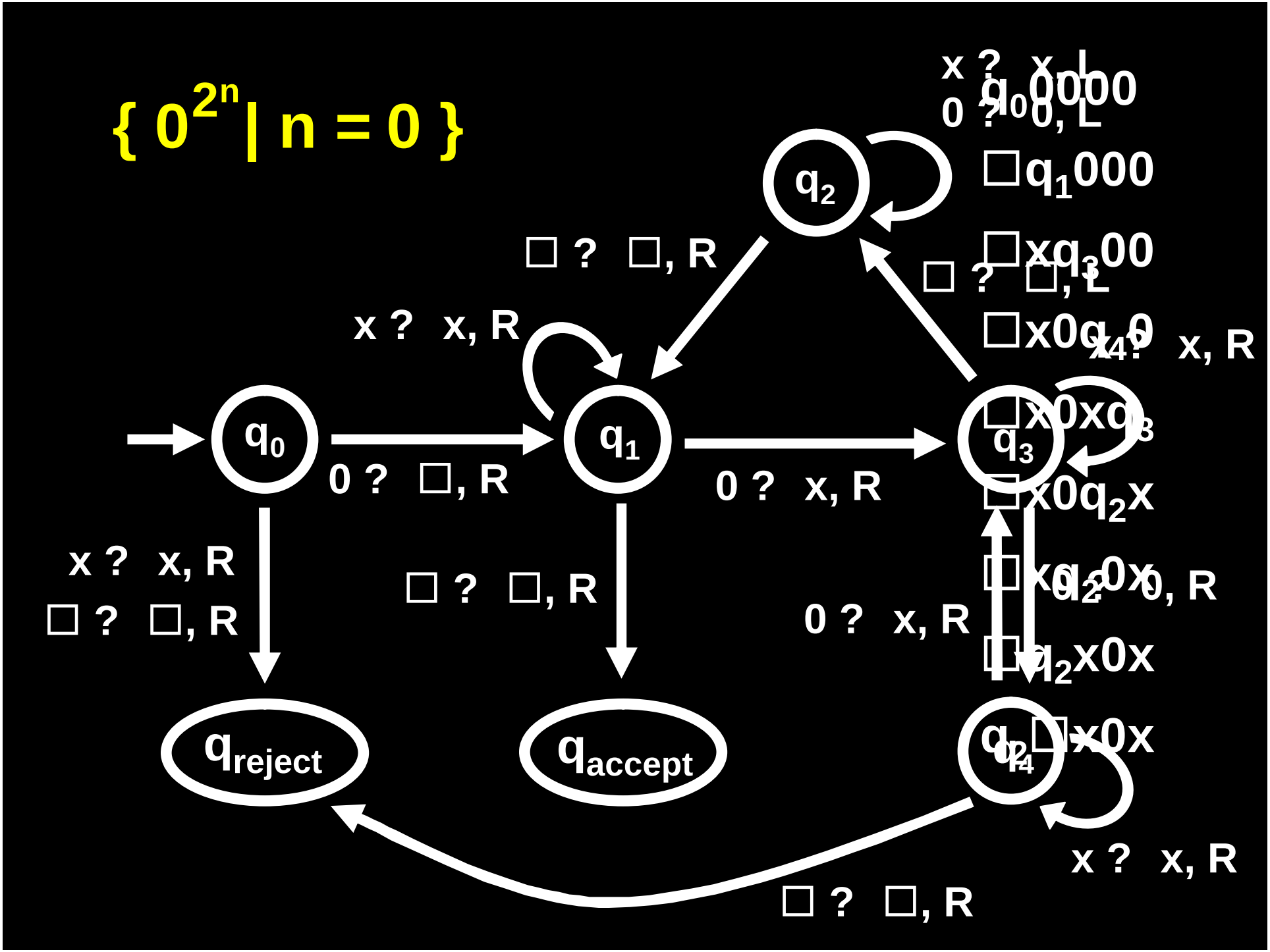


Turing Machines (TM)

Theorem: **If A and ∅A are r.e. then A is recursive**

Given TM that recognizes A and TM that recognizes ∅A, we can build a new machine that decides A


Turing Machines (TM)
{ 0^2^n | n = 0 }
x ? x, L
0 ? 0, L
q2
□ ? □, R
□ ? □, L
x ? x, R
x ? x, R
q0
q1
q3
0 ? □, R
0 ? x, R
x ? x, R
□ ? □, R
□ ? □, R
0 ? x, R
0 ? 0, R
qreject
qaccept
q4
x ? x, R
□ ? □, R


{ 0^2^n | n = 0 }
q0
q1
q2
q3
q4
qreject
qaccept
x ? x, L
0 ? 0, L
□ ? □, R
x ? x, R
0 ? □, R
0 ? x, R
x ? x, R
□ ? □, R
□ ? □, R
□ ? □, L
x ? x, R
□ ? □, R
x ? x, R
□ ? □, R
0 ? x, R
x ? x, R

Turing Machines (TM)

$C = \{a^i b^j c^k \mid ij = k \text{ and } i, j, k = 1\}$

aabbbccccccc

xabbbccccccc

xayyyzzzccc

xabbbzzzccc

xxyyyzzzzzz

Turing Machines (TM)

Example

Write a TM for the language L={w Î **{0,1}* :** #(0)=#(1)}?

Pseudocode:

```
while (there is a 0 and a 1)
  cross these out
if (everything crossed out)
  accept
else
  reject
```

Turing Machines (TM)

Example

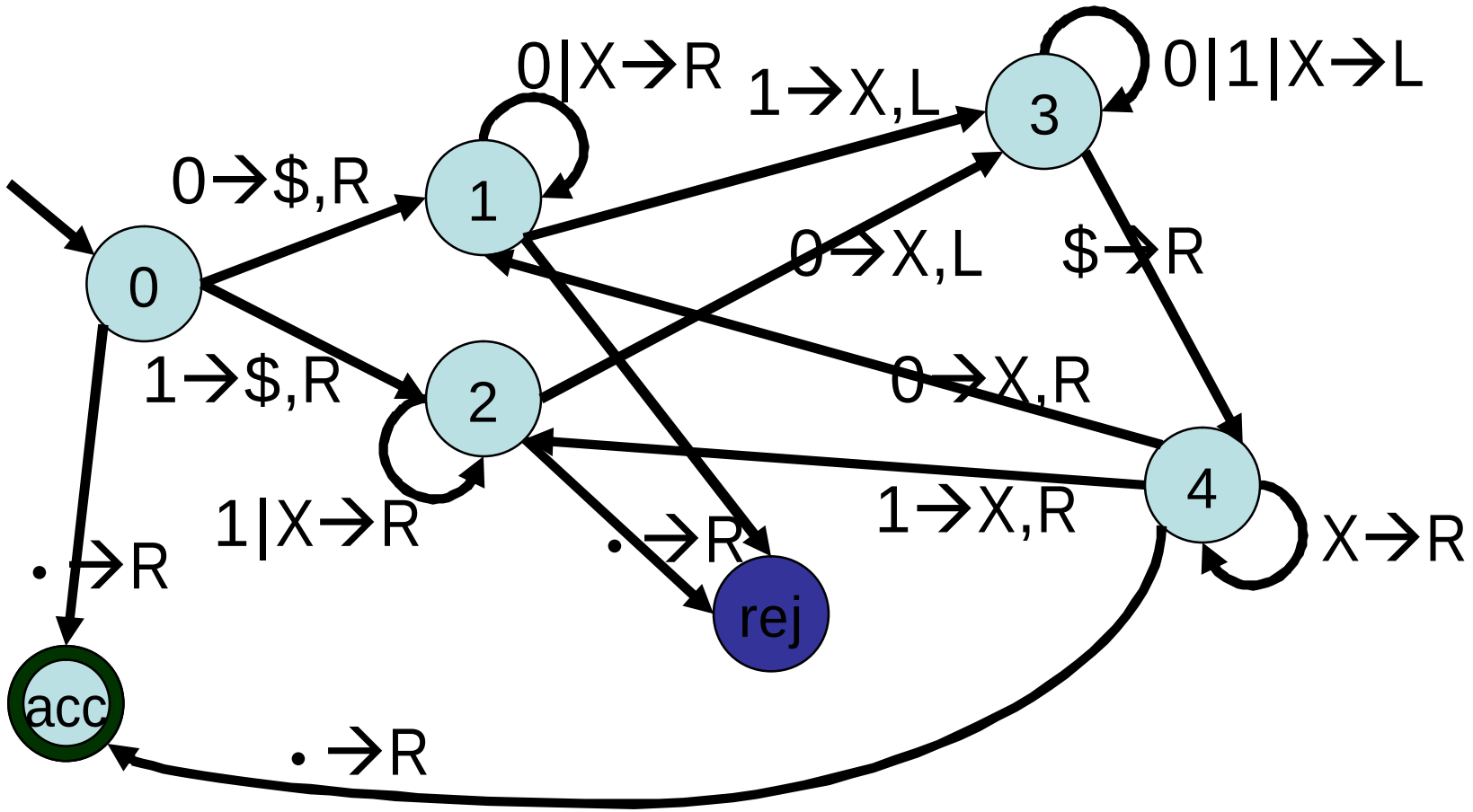
$L = \{w \in \{0,1\}^* : \#(0) = \#(1)\}$?

0. if read ␣ go right (*dummy move*), ACCEPT
   if read 0, write $, go right, goto 1 // $ detects start of tape
   if read 1, write $, go right, goto 2
1. if read ␣ go right, REJECT
   if read 0 or X, go right, repeat (= *goto* 1) // look for a 1
   if read 1, write X, go left, goto 3
2. if read ␣ go right, REJECT
   if read 1 or X, go right, repeat // look for a 0
   if read 0, write X, go left, goto 3
3. if read $, go right, goto 4 // look for start of tape
   else, go left, repeat
4. if read 0, write X, go right, goto 1 // similar to step 0
   if read 1, write X, go right, goto 2
   if read X, go right, repeat
   if read ␣ go right, ACCEPT

Turing Machines (TM)

Example

$L=\{w \in \{0,1\}^* : \#(0)=\#(1)\}$?



Turing Machines (TM)

Definition

A string *x* is ***accepted*** by a TM *M* if after being put on the tape with the Turing machine head set to the left-most position, and letting *M* run, *M* eventually enters the accept state. In this case *w* is an element of *L*(*M*) –the language accepted by *M.*

We can formalize this notion as follows:

Turing Machines (TM)

Definition

Suppose TM's configuration at time *t* is given by *uapxv* where *p* is the current state, *ua* is what's to the left of the head, *x* is what's being read, and *v* is what's to the right of the head.

If $\delta(p,x) = (q,y,\mathrm{R})$ then write:

$$uapxv \vdash uayqv$$

With resulting configuration *uayqv* at time *t*+1.

If, $\delta(p,x) = (q,y,\mathrm{L})$ instead, then write:

$$uapxv \vdash uqayv$$

There are also two special cases:

- head is forging new ground –pad with the blank symbol •
- head is stuck at left end –by def. head stays put (only case)

" $\vdash$ " is read as "***yields***"

Turing Machines (TM)

Definition

As with context free grammars, one can consider the reflexive, transitive closure "$\vdash^*$" of "$\vdash$". I.e. this is the relation between strings recursively defined by:

- if $u = v$ then $u \vdash^* v$
- if $u \vdash v$ then $u \vdash^* v$
- if $u \vdash^* v$ and $v \vdash^* w$, then $u \vdash^* w$

"$\vdash^*$" is read as "***computes to***"

A string x is said to be ***accepted*** by M if the *start configuration* $q_0 x$ computes to some *accepting configuration* y –i.e., a configuration containing q_{acc}.

The ***language accepted by*** M is the set of all accepted strings. I.e:

$$L(M) = \{ x \in \Sigma^* \mid \exists \text{ accepting config. } y,\ q_0 x \vdash^* y \}$$

TM Acceptor and Deciders

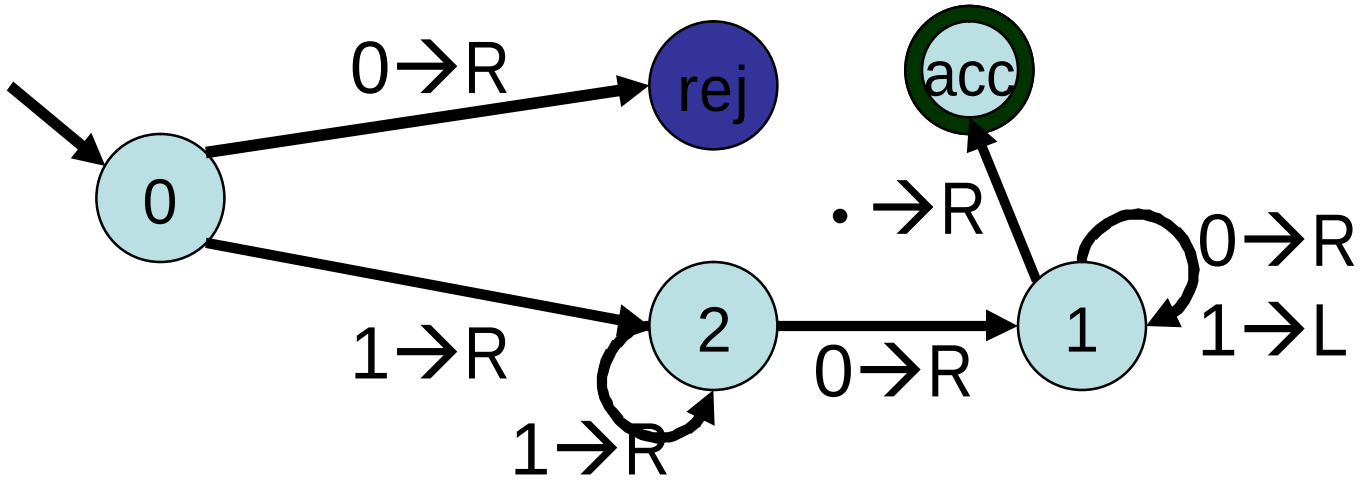
Three possibilities occur on a given input w :

1. The TM M eventually enters q_{acc} and therefore halts and accepts. ($w \in L(M)$)
2. The TM M eventually enters q_{rej} *or* crashes somewhere. M ***rejects*** w . ($w \notin L(M)$)
3. Neither occurs! I.e., M never halts its computation and is caught up in an ***infinite loop***, never reaching q_{acc} or q_{rej}. In this case w is neither accepted nor rejected. However, any string not explicitly accepted is considered to be outside the accepted language. ($w \notin L(M)$)

Turing Machines (TM)

TM Acceptor and Deciders

Any Turing Machines is said to be a ***recognizer*** and ***recognizes*** *L*(*M*); if in addition, *M never* enters an infinite loop, *M* is called a ***decider*** and is said to ***decide*** *L*(*M*).

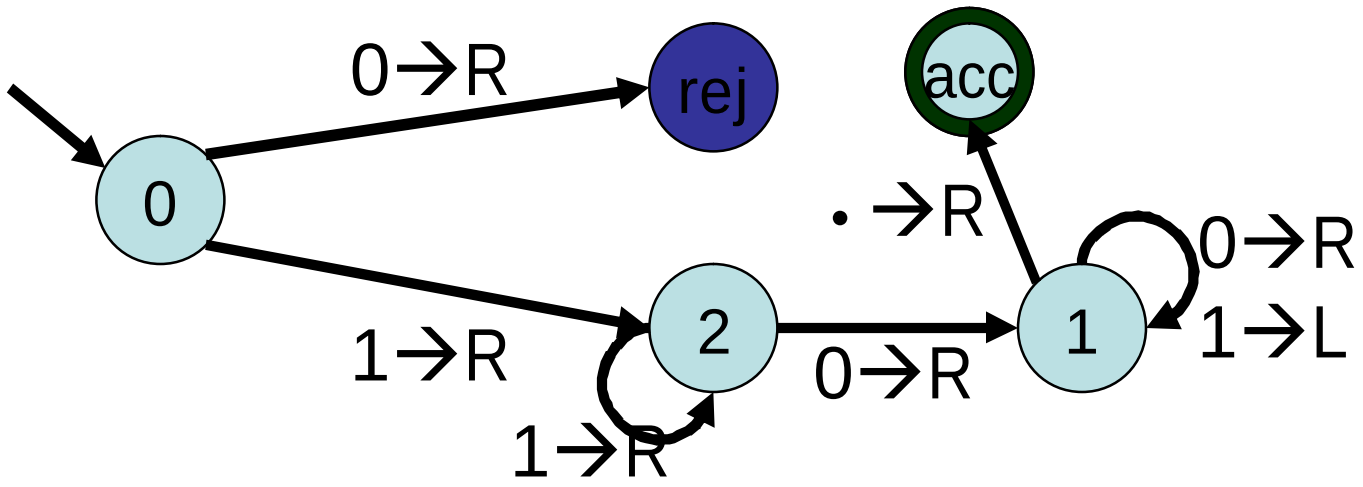


Q: Is the above *M* an recognizer? A decider? What is *L*(*M*)?

TM Acceptor and Deciders

A: *M* is an recognizer but not a decider because 101 causes an infinite loop.

$L(M) = 1^+ 0^+$



Q: Is *L*(*M*) decid***able*** ?

TM Acceptor and Deciders

A: Yes. All regular languages are decidable because can always convert a DFA into a TM without infinite loops.

Input-Output Turing Machines

Input/output (or ***IO*** or ***transducing***) Turing Machines, differ from TM recognizers in that they have a neutral halt state q_{halt} instead of the accept and reject halt states. The TM is then viewed as a string-function which takes initial tape contents u to whatever the non blank portion of the tape is when reaching q_{halt}. If v is the tape content upon halting, the notation $f_M(u) = v$ is used.

If M crashes during the computation, or enters into an infinite loop, M is said to be ***undefined*** on u.

Input-Output Turing Machines

When f_M crashes or goes into an infinite loop for some contents, f_M is a ***partial function***

If *M* always halts properly for any possible input, its function *f* is ***total*** (i.e. always defined).

Input-Output Turing Machines: Computing a function

A Turing Machine *M* can be used to compute functions. $f(n)=2n$.

A function arguments *n* will be represented as a sequence of *n* 1's on the TM tape.

For example if $n=3$ the input string 111 will be written on the TM tape.

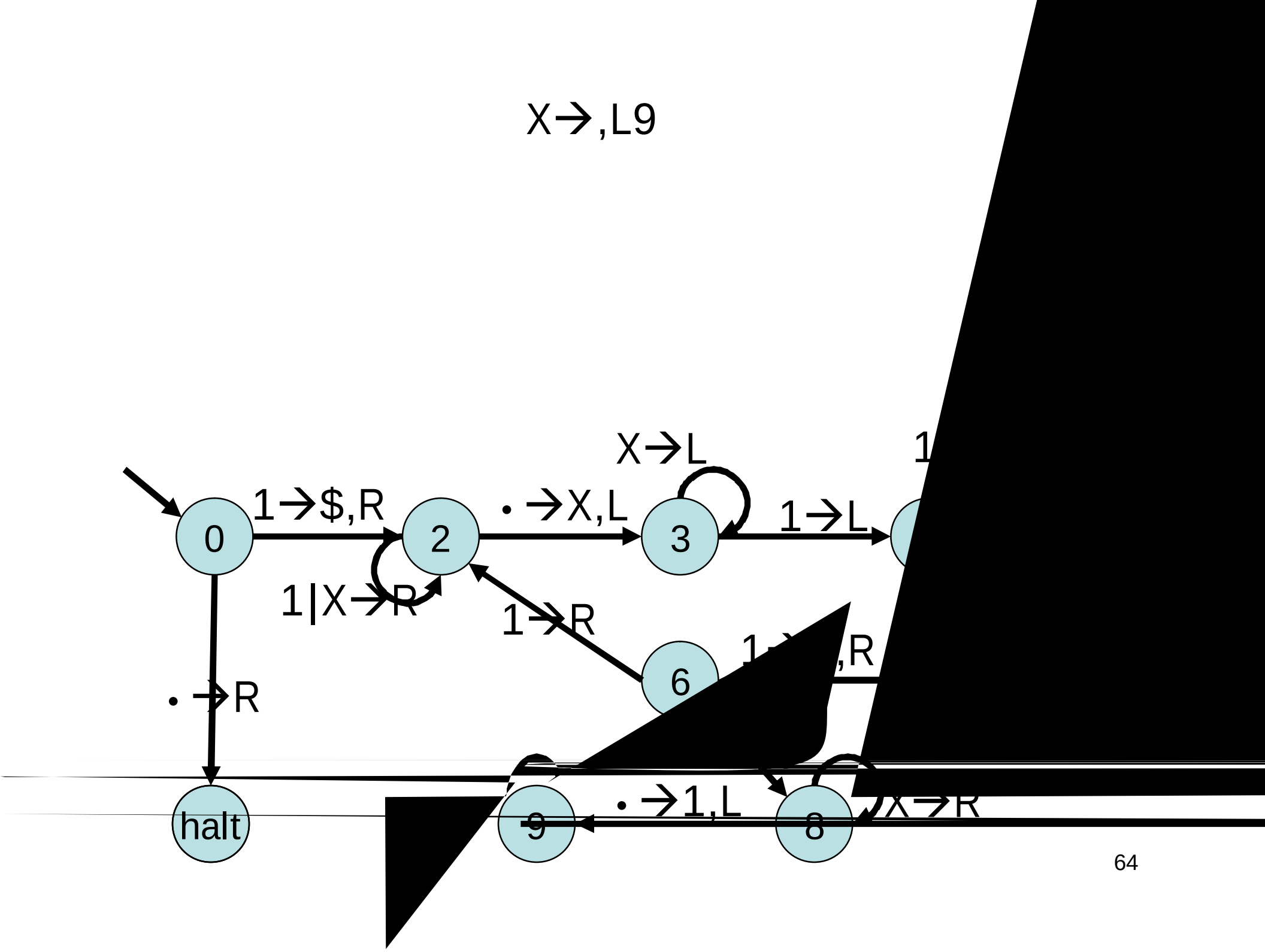
If the function has several arguments then everyone is represented as a sequence of 1's and they are separated by the * symbol.

Input-Output Turing Machines: Example 1

Let's for example describe a Turing Machine *M* which computes the function $f(n)=2n$.

The argument *n* will be represented as a sequence of *n* 1's on the TM tape.

X→,L9



Non-Deterministic Turing Machines (NTM)

A non-Deterministic Turing Machine *N* allows more than one possible action per given state-tape symbol pair.

A string *w* is ***accepted*** by *N* if after being put on the tape and letting *N* run, *N* eventually enters q_{acc} on *some computation branch*.

If, on the other hand, given any branch, *N* eventually enters q_{rej} **or** crashes **or** enters an infinite loop on, *w* is not accepted.

Symbolically as before:

$$L(N) = \{ x \in \Sigma^* \mid \exists \text{ accepting config. } y,\ q_0 x \vdash^* y \}$$

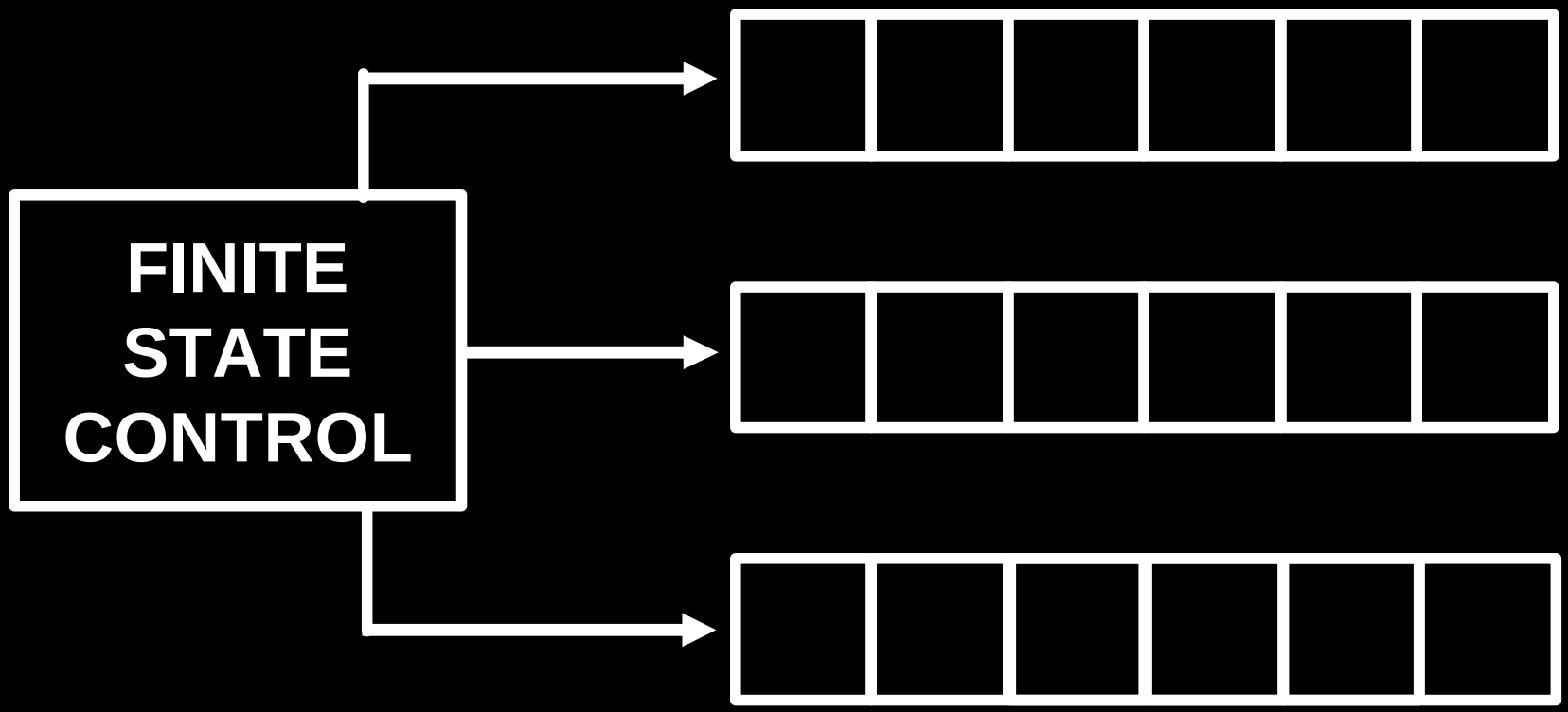
(No change needed as $\vdash$ need not be function)

Non-Deterministic Turing Machines (NTM)

NTM Acceptor and Deciders

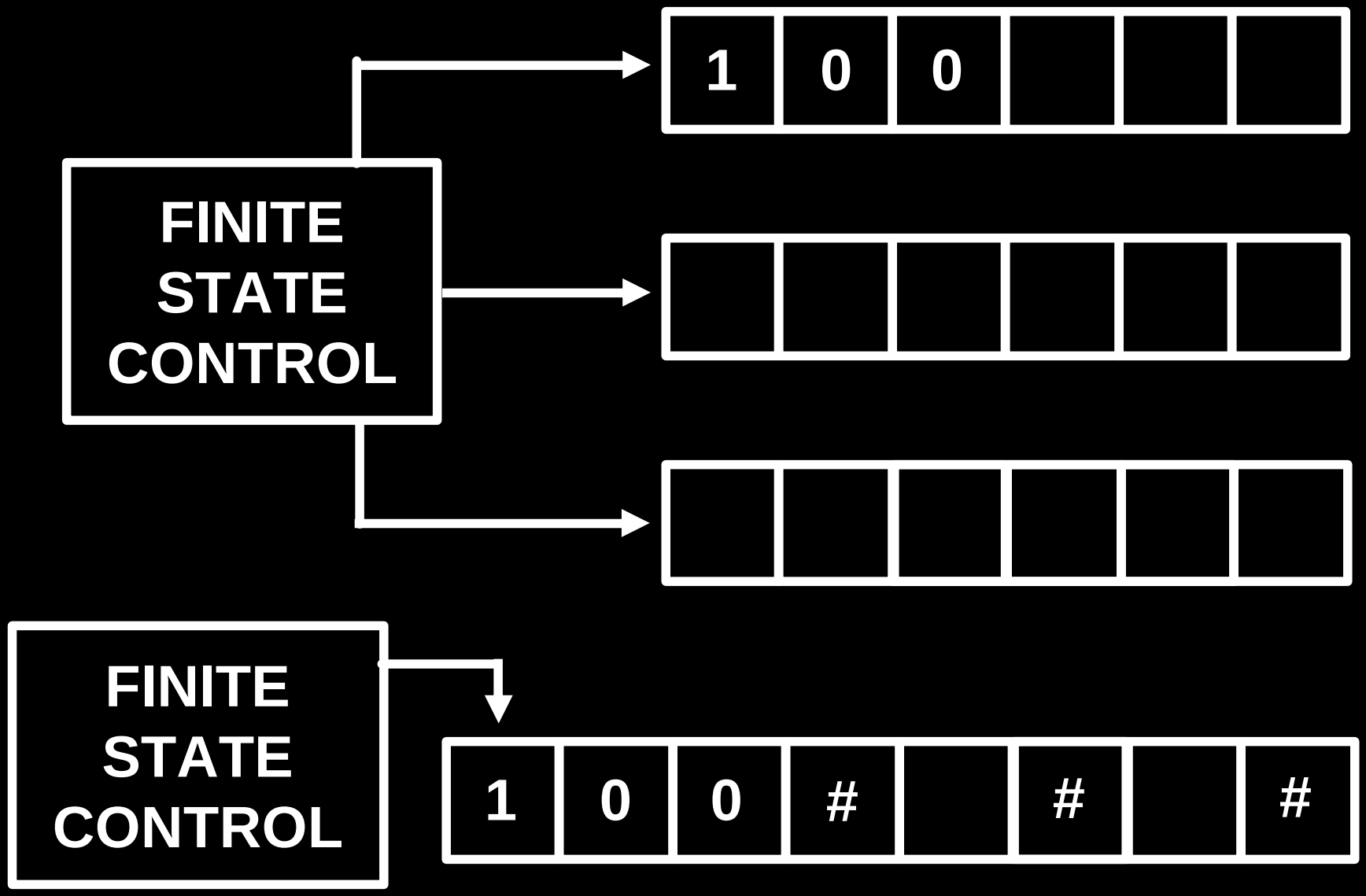
N is always called a ***non-deterministic recognizer*** and is said to *recognize* *L*(*N*); furthermore, if in addition for all inputs and all computation branches, *N* always halts, then *N* is called a ***non-deterministic decider*** and is said to decide *L*(*N*).

Multitape Turing Machines



$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

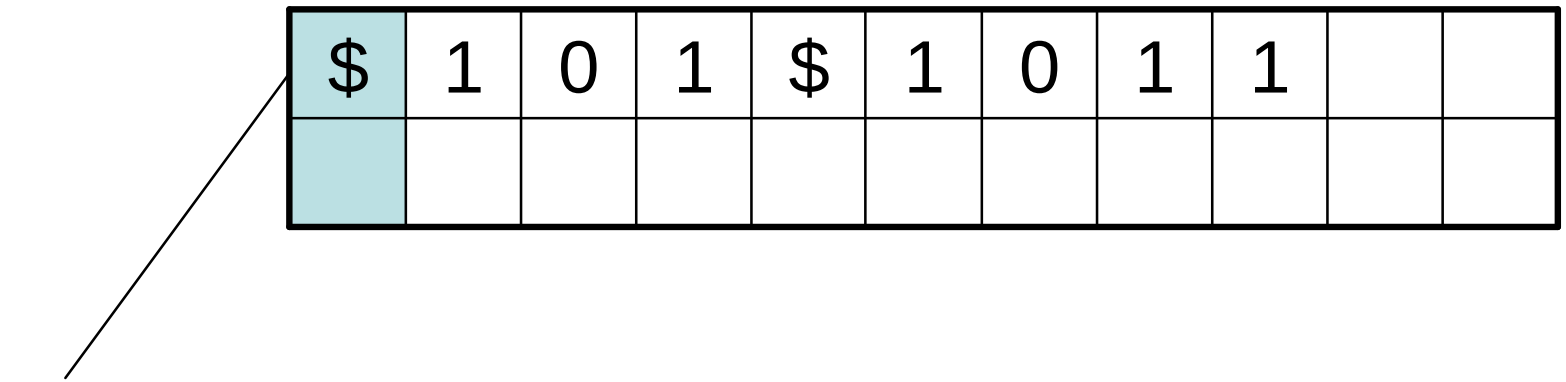


Multitape Turing Machines

Often it's useful to have several tapes when carrying out a computations. For example, consider a two tape I/O TM for adding numbers (we show only how it acts on a typical input)

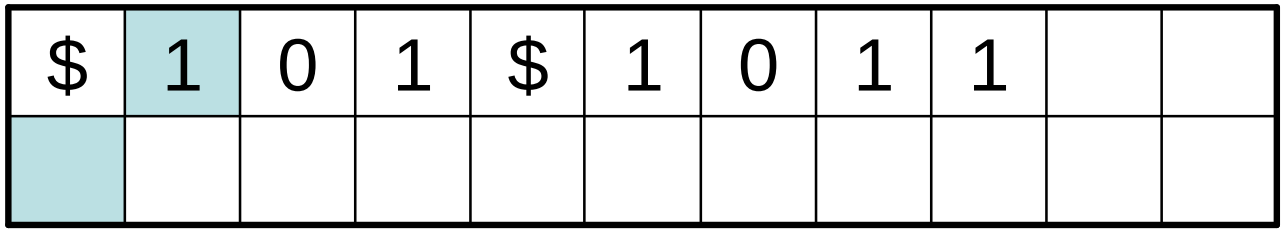
Multitape Turing Machines

Example: Addition



Multitape Turing Machines

Example: Addition



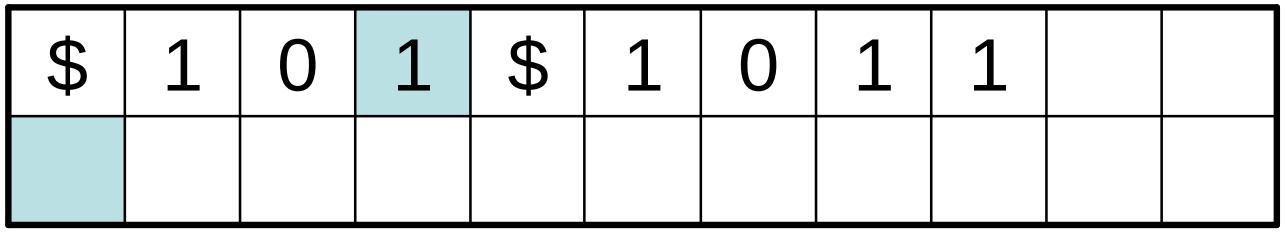
Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		

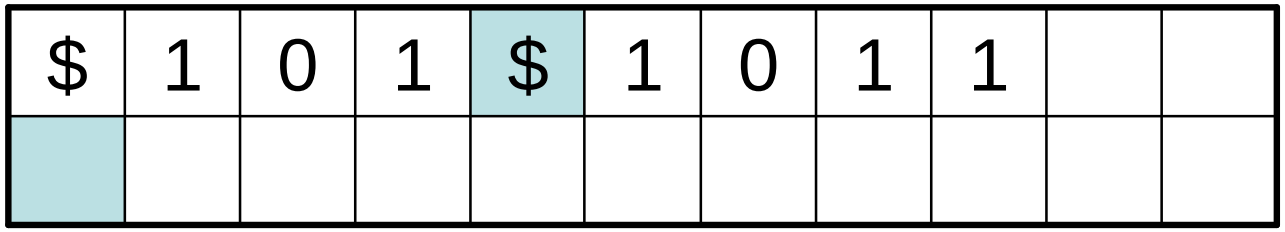
Multitape Turing Machines

Example: Addition



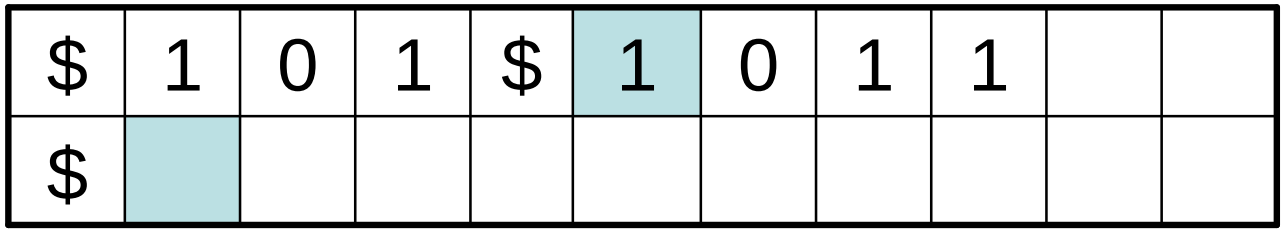
Multitape Turing Machines

Example: Addition



Multitape Turing Machines

Example: Addition



Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		
$	1									

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		
$	1	0								

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		
$	1	0	1							

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1	1		
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0	1			
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1	0				
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1	$	1					
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1	$						
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	1							
$	1	0	1	1						

Multitape Turing Machines

Example: Addition

$	1	0	0							
$	1	0	1	0						

Multitape Turing Machines

Example: Addition

$	1	0	0							
$	1	0	0	0						

Multitape Turing Machines

Example: Addition

$	0	0	0							
$	1	0	0	0						

Multitape Turing Machines

Example: Addition

0	0	0	0							
$	0	0	0	0						

Multitape Turing Machines

Example: Addition

1	0	0	0							
1	0	0	0	0						

Multitape Turing Machines

Example: Addition

1	0	0	0							
1	0	0	0	0						

Multitape Turing Machines

Example: Addition

1	0	0	0							
1	0	0	0	0						

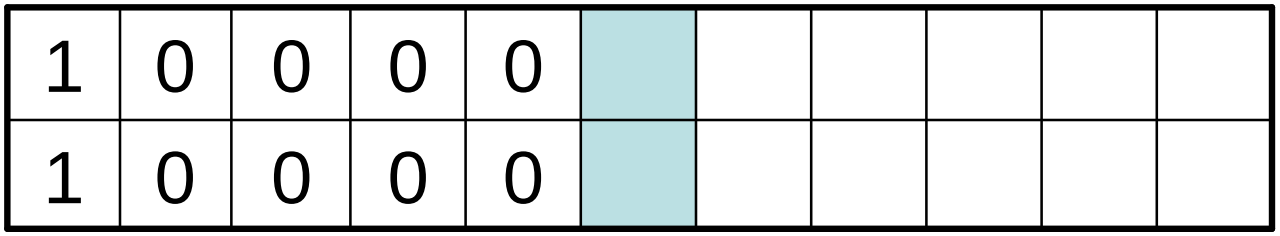
Multitape Turing Machines

Example: Addition

1	0	0	0							
1	0	0	0	0						

Multitape Turing Machines

Example: Addition



HALT!

Output string

Multitape Turing Machines

Conventions

- Input always put on the first tape
- If I/O machine, output also on first tape
- Can consider machines as “string-vector” generators. E.g., a 4 tape machine could be considered as outputting in $(\Sigma^*)^4$

The Big Deal

The notions of **recursively enumerable / TM recognizable** and **recursive / TM decidable** are the greatest contributions of computer science.

For the moment note the following:

- They are extremely robust notions (“Church
- They are different notions (“computability theory”)

Exercise

Write a TM for the Language $\{ 0^j \mid j=2^n \}$

Exercise: Answer hint

Write a TM for the Language { 0j | j=2n }

Approach: If j=0 then “reject”; If j=1 then “accept”; if j is even then divide by two; if j is odd and >1 then “reject”. Repeat if necessary.

1. Check if j=0 or j=1, reject/accept accordingly
2. Check, by going left to right if the string has even or odd number of zeros
3. If odd then “reject”
4. If even then go back left, erasing half the zeros
5. goto 1