Automata and Languages

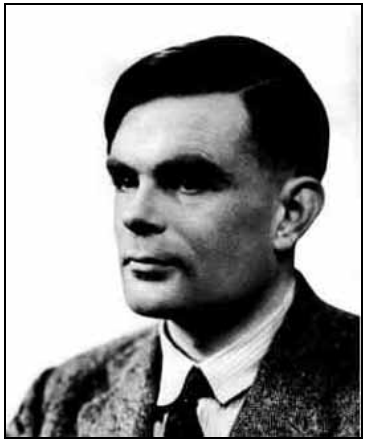
Prof. Mohamed Hamada

Software Engineering Lab.
The University of Aizu
Japan

TURING MACHINES

Turing Machines (TM)

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



In 1936, Turing introduced his abstract model for computation.

The Turing machine is the ultimate model of 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)

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!

1	1	1	1	0	1				

Turing Machines (TM)

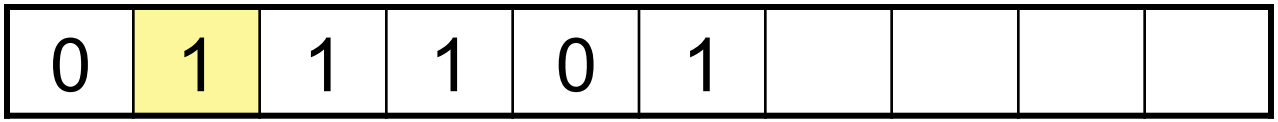
A Thinking Machine

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)

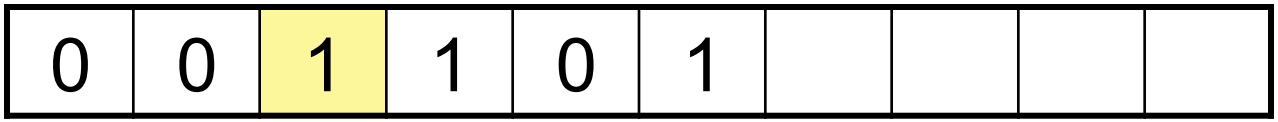
A Thinking Machine

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!

0	0	0	0	0	1				

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!

0	0	0	0	1	1				

Turing Machines (TM)

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!

1	1	1	1	1	1	1			

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!

0	1	1	1	1	1	1			

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!

0	0	1	1	1	1	1			

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!

0	0	0	1	1	1	1			

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!

0	0	0	0	1	1	1			

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!

0	0	0	0	0	1	1			

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!

0	0	0	0	0	0	1			

Turing Machines (TM)

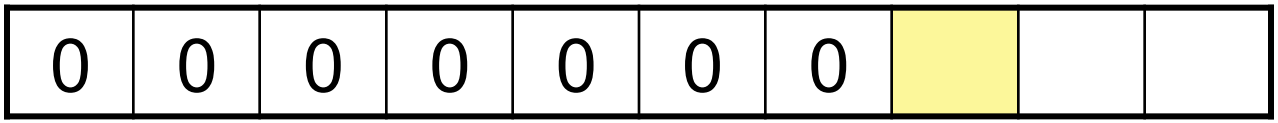
A Thinking Machine

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!

0	0	0	0	0	0	0	1		

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

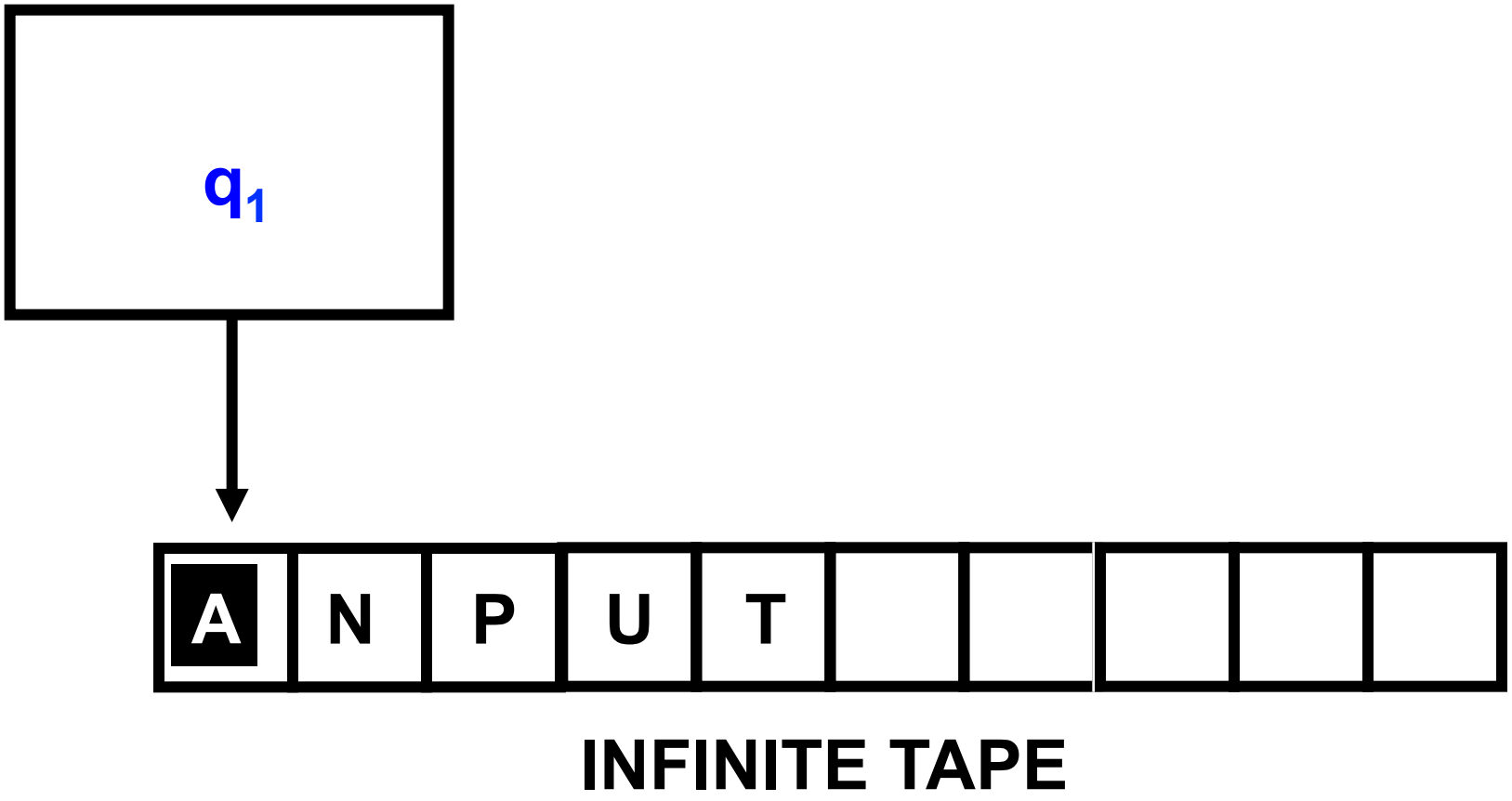
Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM			

Turing Machines (TM)

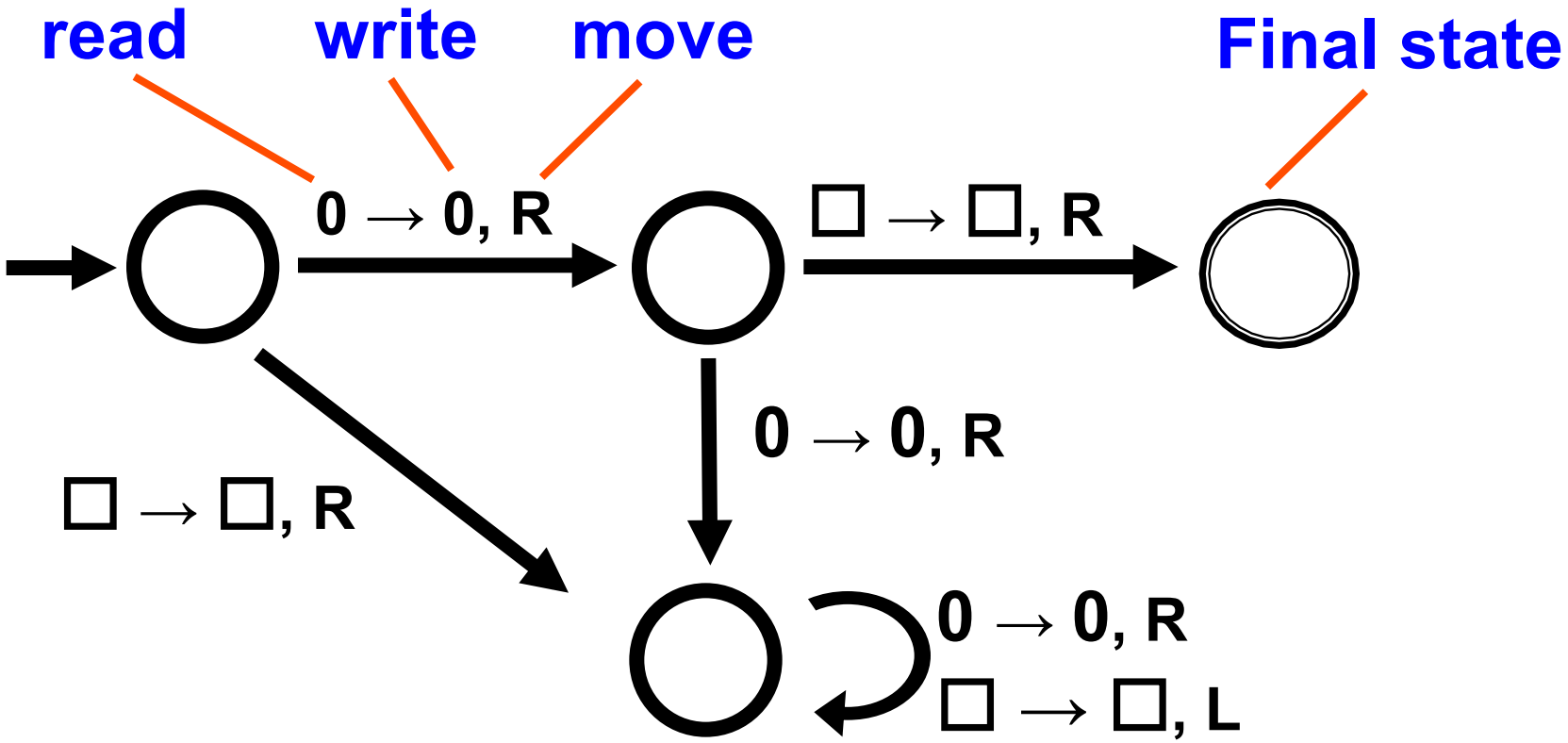
A Comparison with FA and PDA

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)

Definition

A Turing Machine is a 7-tuple
$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where:

Q is a finite set of states

Σ is the input alphabet, where $\square \notin \Sigma$

Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$

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

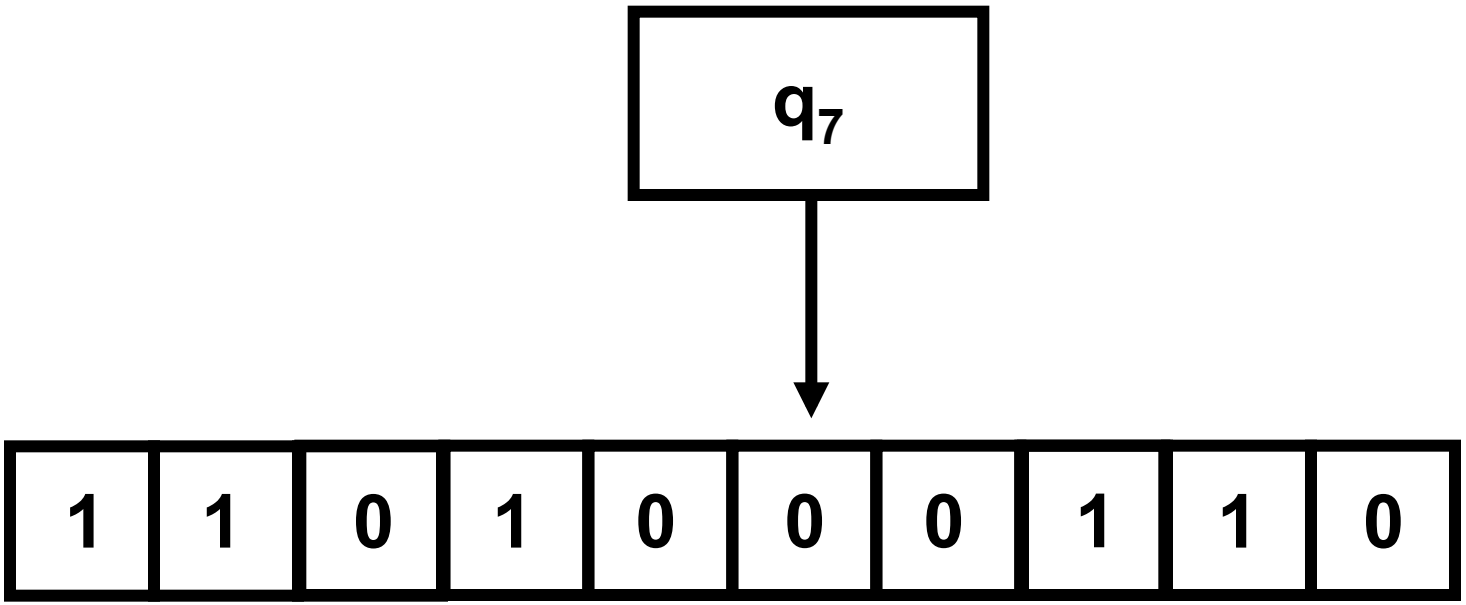
$q_0 \in Q$ is the start state

B is the empty input square $\square$

$F \subseteq Q$ is the set of final states÷

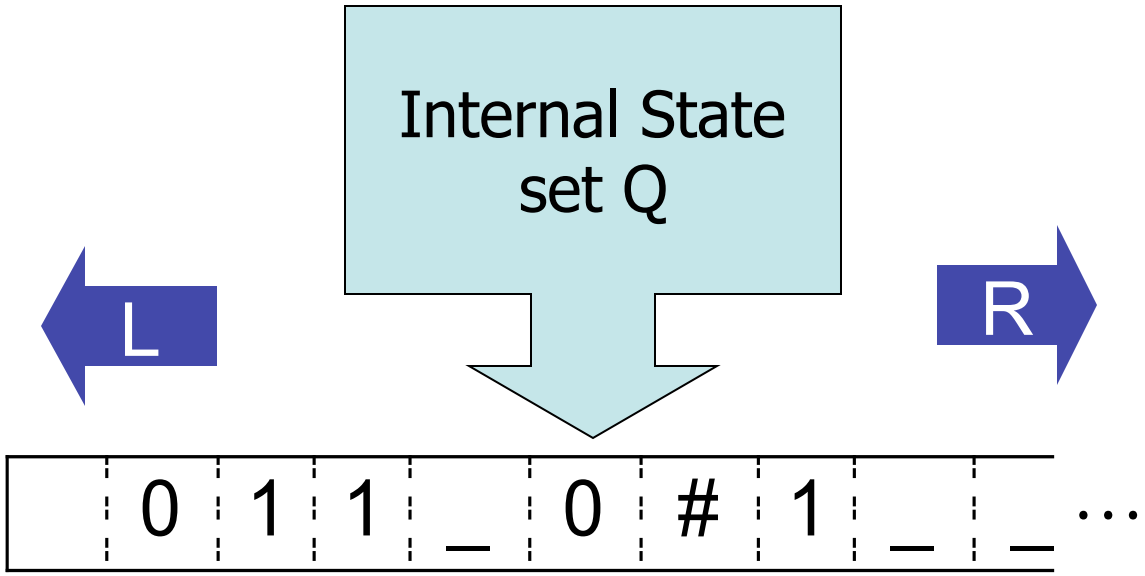
CONFIGURATIONS

$11010q_700110$



Turing Machines (TM)

Informal Description



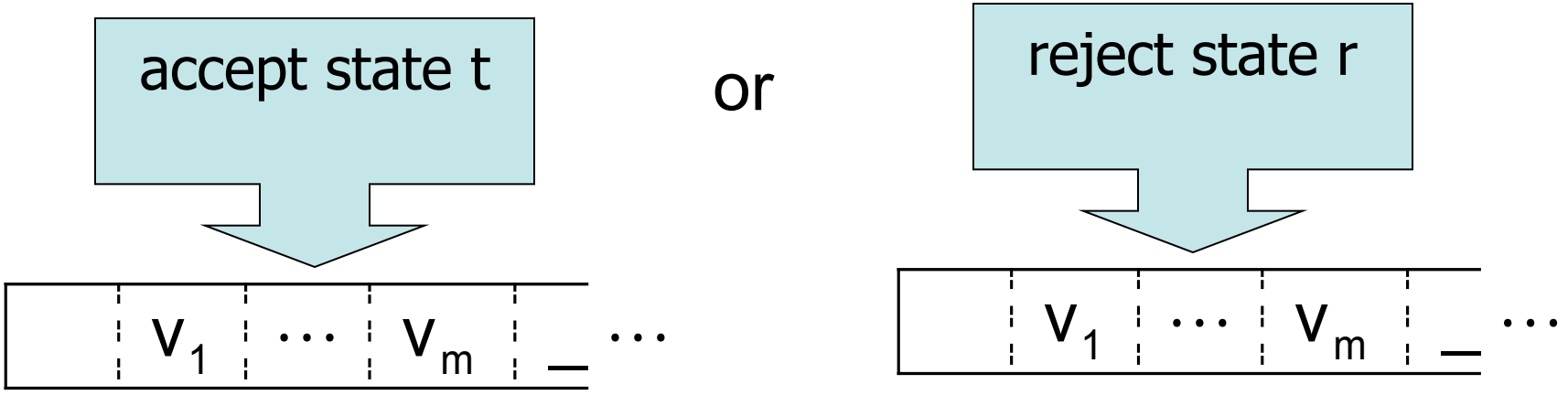
At every step, the head of the TM M reads a letter x from the tape of size ω.

Depending on x and q the, the transition function value δ(q,x) = (r,y,d) tells the TM to replace the letter x by y, 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:



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

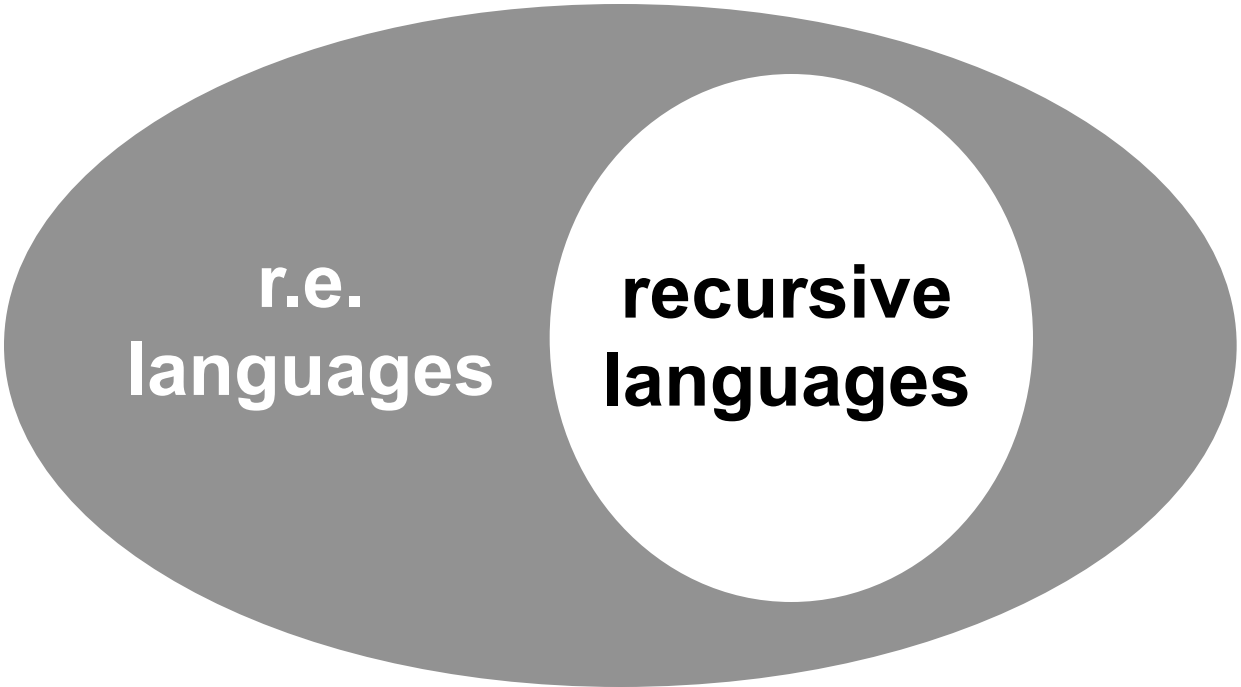
A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

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

Turing Machines (TM)

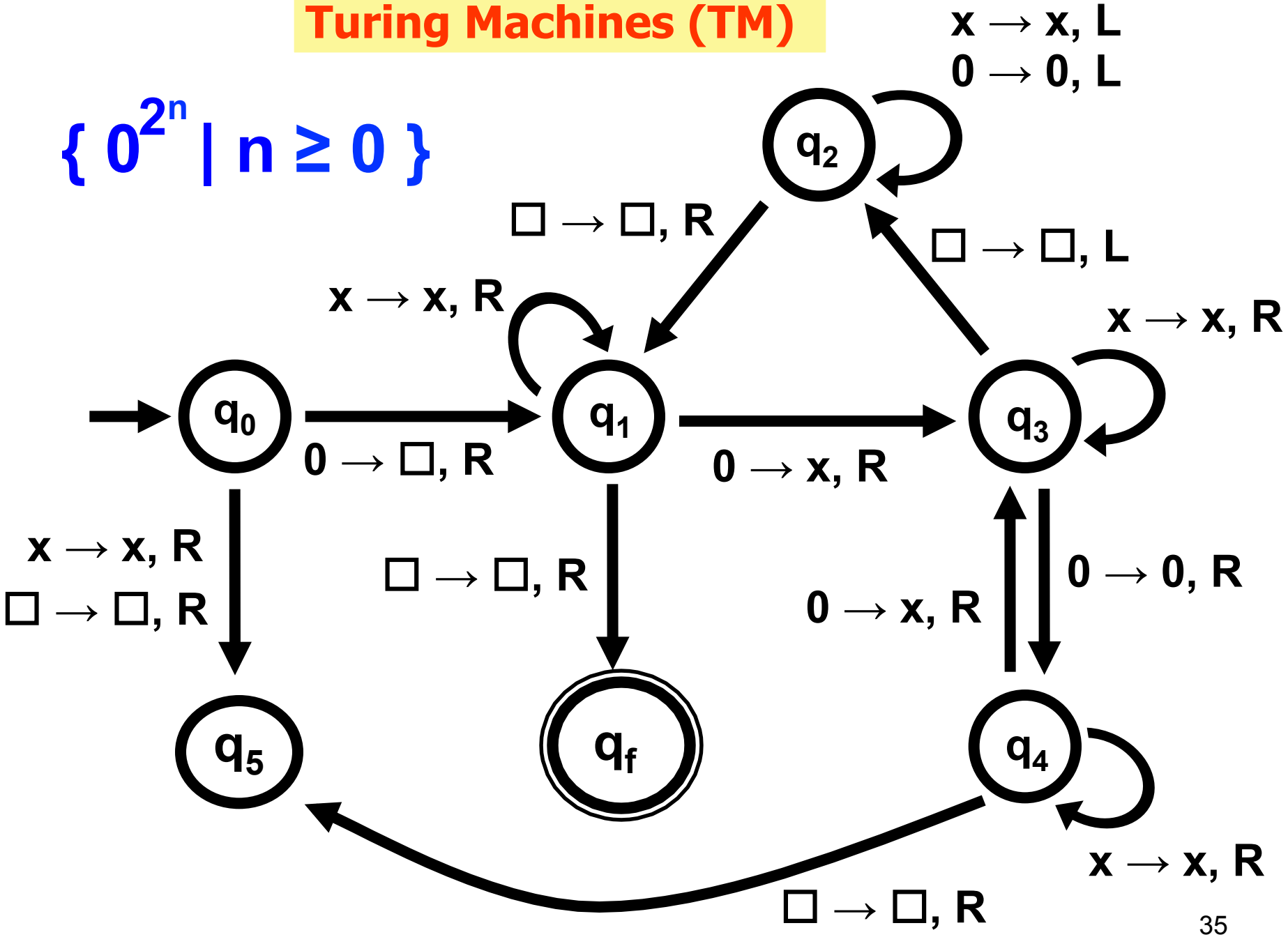
A language is called Turing-recognizable or recursively enumerable if some TM recognizes it

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



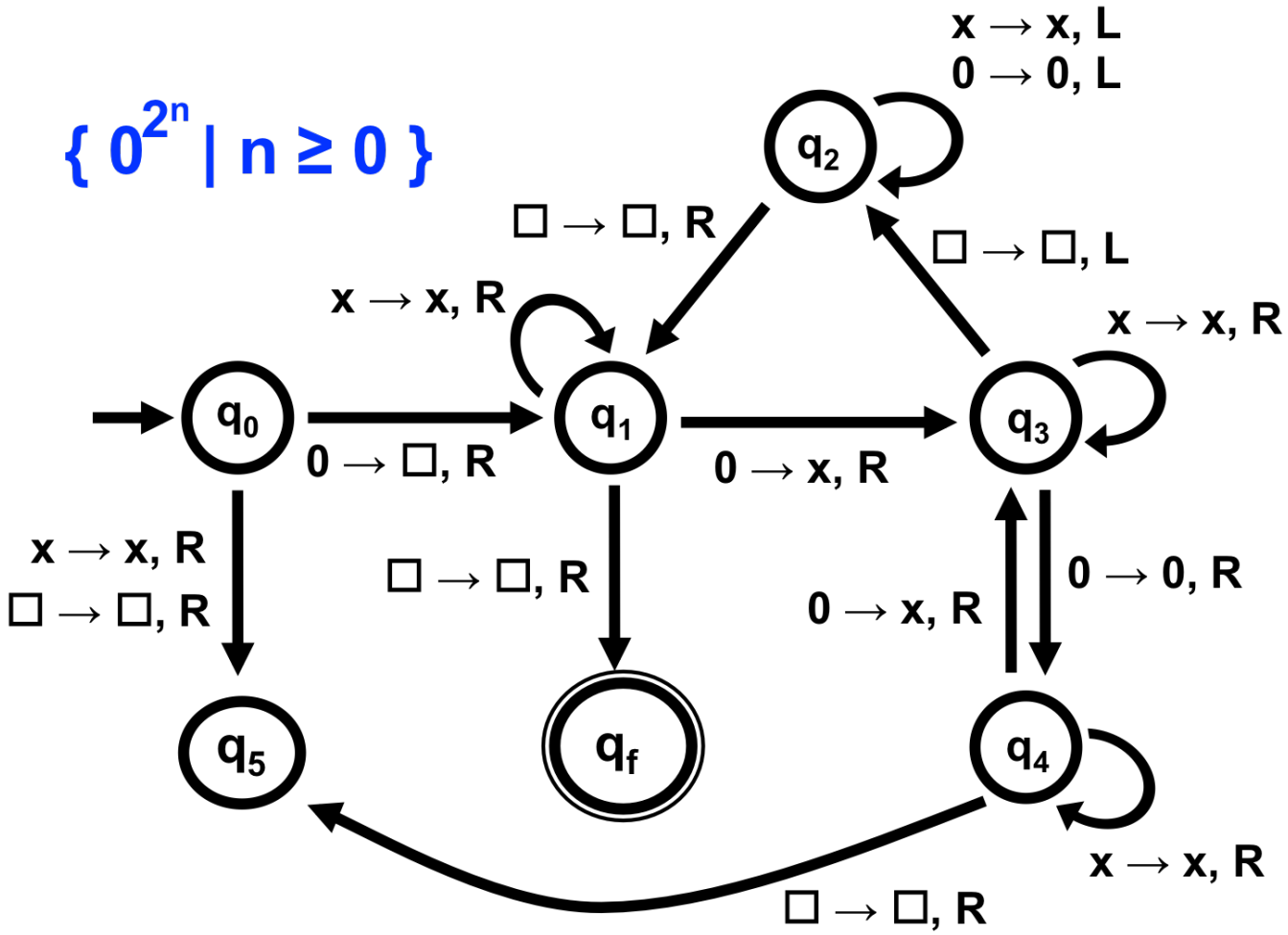
Turing Machines (TM)

$\{ 0^{2^n} \mid n \geq 0 \}$



Turing Machines (TM)

$\{ 0^{2^n} \mid n \geq 0 \}$



$q_0 0000$

$\square q_1 000$

$\square x q_3 00$

$\square x 0 q_4 0$

$\square x 0 x q_3$

$\square x 0 q_2 x$

$\square x q_2 0 x$

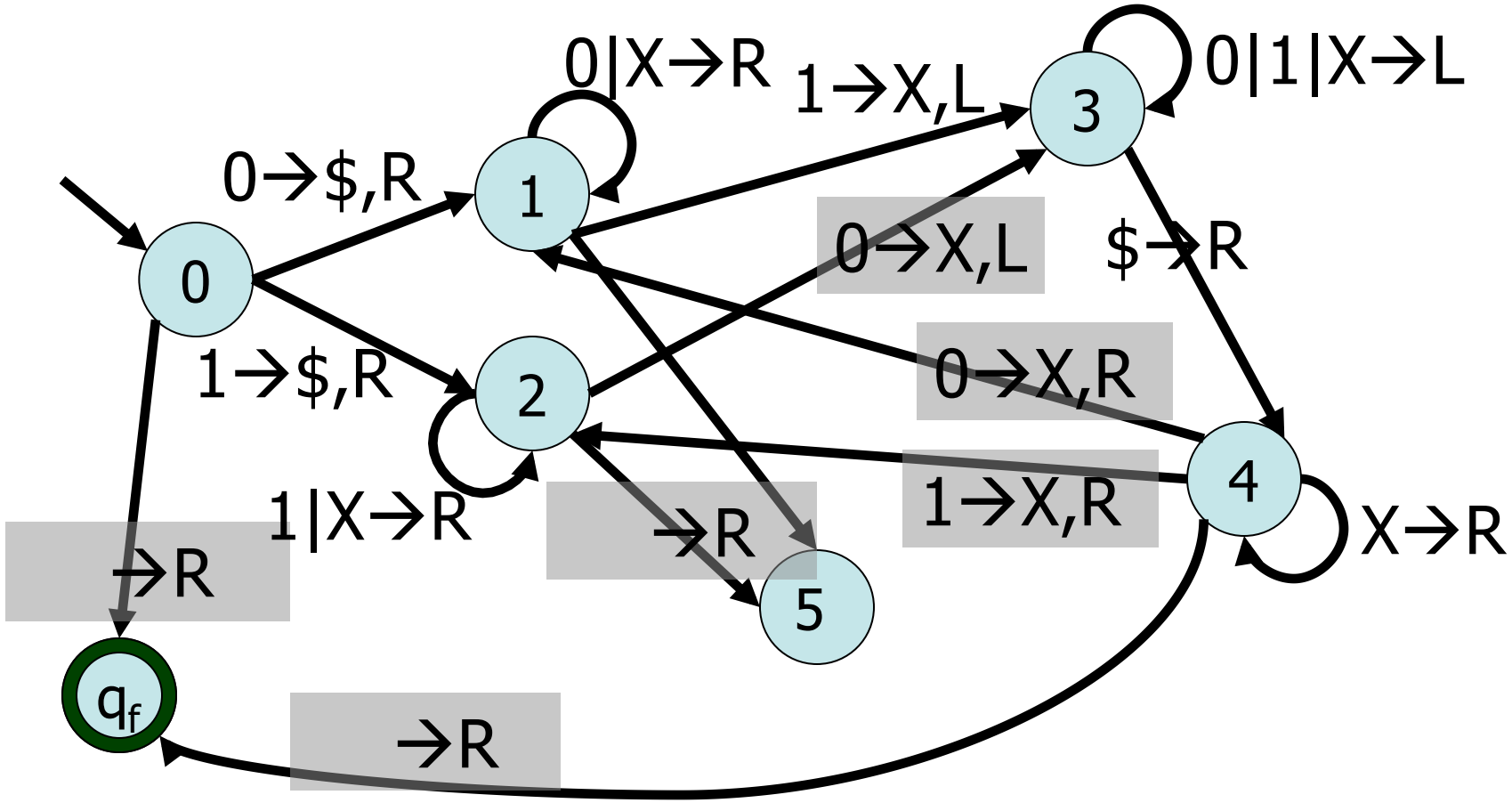
$\square q_2 x 0 x$

$q_2 \square x 0 x$

Turing Machines (TM)

Example

L={w∈**{0,1}*** **:** #(0)=#(1)}?



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:

$$uapxv \Rightarrow uayqv$$

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

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

$$uapxv \Rightarrow uqayv$$

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)

"⇒" is read as "***yields***"

Turing Machines (TM)

Definition

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

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

"$\Rightarrow^*$" 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:

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

Turing Machines (TM)

TM Acceptor and Deciders

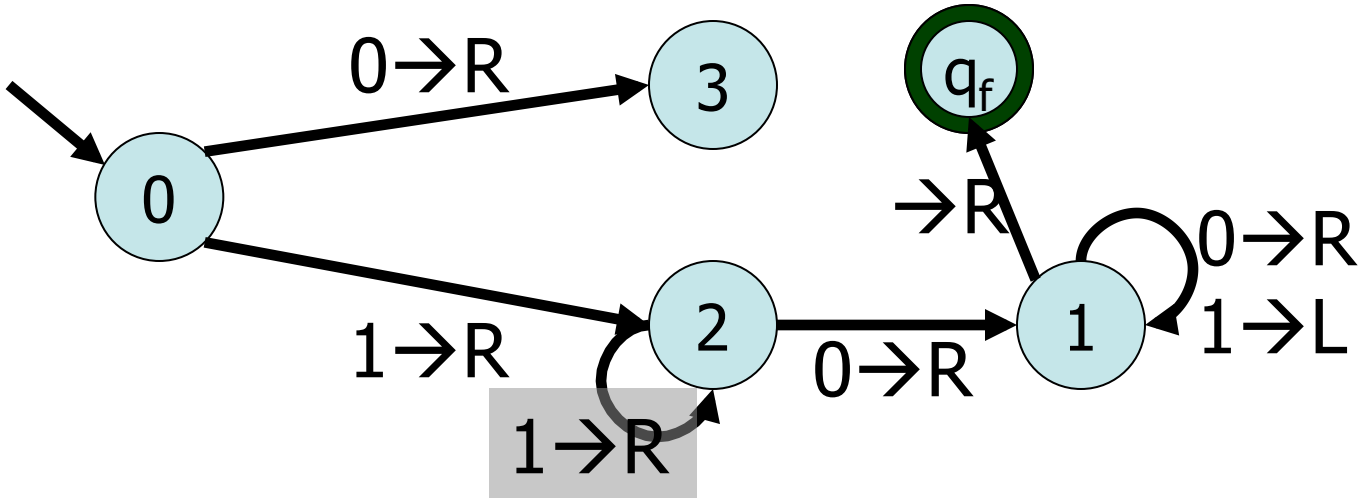
Three possibilities occur on a given input w :

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*).



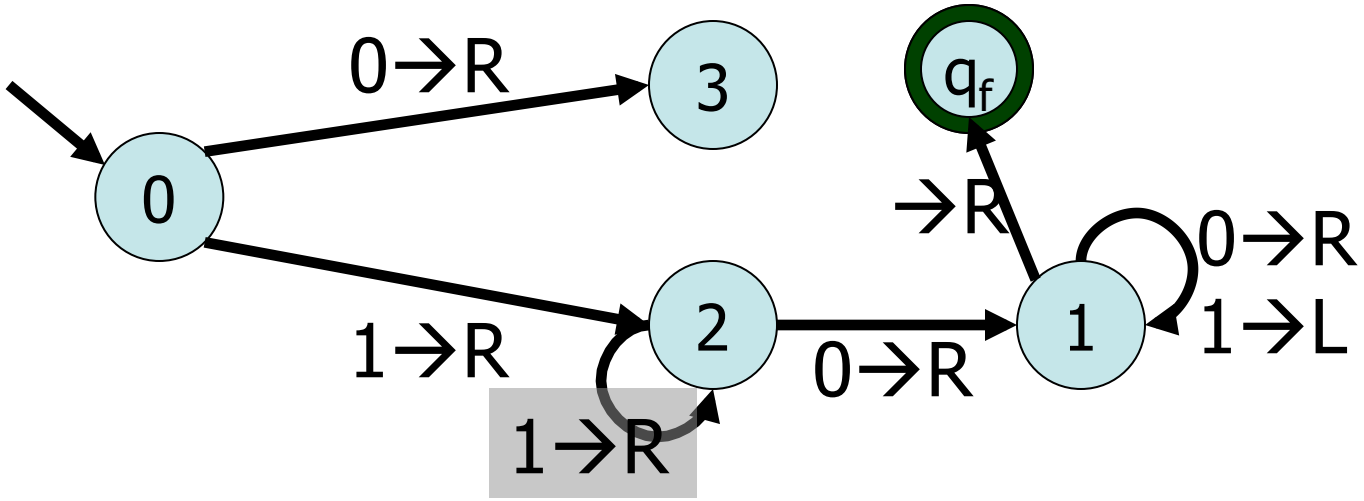
Q: Is the above *M* an recognizer? A decider? What is *L*(*M*)?

Turing Machines (TM)

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*** ?

Turing Machines (TM)

TM Acceptor and Deciders

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

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_f 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:

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

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

Content

- Decidability
- The Halting Problem
- Complexity
- The class P
- The class NP
- The class P vs. NP
- The class NP-Complete

Decidability

- What are the limits of algorithmic solvability?
- How can we tell if two Regular Expressions define the same language?
  - or, can we?
- A language is **decidable** if some TM decides it (always either accepts or rejects, always halts)

Remember:

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

Example 1

Decidability

- Acceptance Problem (DFA): Does a given DFA, *B*, accept a given string *w*?
- In terms of languages (because we have defined computation as accept/reject a language):
  - A_{DFA} = { <B, w> | B is a DFA that accepts w }
  - For ALL input pairs <B, w> can a single TM be constructed that will decide <B,w> $\in A_{DFA}$
    - can we build one TM that will work for all DFAs?
    - is there an *algorithmic* way to solve this problem?

Example 1

Theorem

- A_{DFA} is decidable
  - given <B, w> we can decide if <B, w> $\in A_{DFA}$ or <B, w> $\notin A_{DFA}$

- Proof Idea:
  - Use a TM, M, to simulate B with input w
  - Keep track of current state and current position on the input string
  - Update according to the DFA's δ

Note:

Similarly: A_{NFA} and $A_{\text{Regular Expression}}$ are also decidable

Example 2

CFGs

- A_{CFG} = {<G, w> | G is a CFG that generates w}
- A_{CFG} is decidable
- Could enumerate all strings produced by G: could be infinite, though
- Proof Idea
  - Put G in CNF
  - How deep is the parse tree for string w?
  - Check only derivations of this length
    - possibly large, but definitely finite

Example 3

Equivalence of CFGs

- EQ_{CFG} = {<G, H> | G and H are CFG and L(G) = L(H)}
  - not decidable
  - CFGs are not closed under complement or ∩

The Halting Problem

The Halting Problem

**In computability theory, the halting problem is the problem
of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.**

Alan Turing proved in 1936 that a general algorithm to solve the halting problem for *all* possible program-input pairs cannot exist.
A key part of the proof was a mathematical definition of a computer and program, which became known as a Turing machine;
the halting problem is undecidable over Turing machines.
It is one of the first examples of a decision problem.

Will it ever stop?

- A_{TM} = { <M, w> | M is a TM and M accepts w }
  - undecidable
  - remember, decidable means that the TM will eventually reach an accept or reject state; it will halt
  - U is a Universal TM
  - TM U recognizes A_{TM}:
    - 1. Simulate M on input w with U
    - 2. If M accepts then U accepts; if M rejects then U rejects; *if M never halts then U never halts*
    - If we could get U to halt, then we could get M to halt

Complexity

COMPLEXITY THEORY

Studies what can and can't be computed under limited resources such as time, space, etc

Time complexity

Time complexity

Definition: Let M be a TM that halts on all inputs. The running time or time-complexity of M is the function $f : N \rightarrow N$, where f(n) is the maximum number of steps that M uses on any input of length n.

Notation:

- M is a "f(n) time TM"
  TM is our model for computation, so
  TM ≈ algorithm

Example

- $f(n) = 5n^3 + 2n^2 + 22n + 6$
- $O(f(n)) = n^3$
- let $c = 6$ and $n_0 = 10$
- $5n^3 + 2n^2 + 22n + 6 <= 6n^3$
  - for every $n >= n_0$
- $O(f(n)) = n^4$ as well, but we want the tightest upper bound

Time Complexity classes

We define the following two time complexity classes:

Definition: TIME(t(n)) = { L | L is a language decided by a O(t(n)) time using a deterministic Turing Machine }

$$A = \{ 0^k 1^k \mid k \geq 0 \} \in TIME(n^2)$$

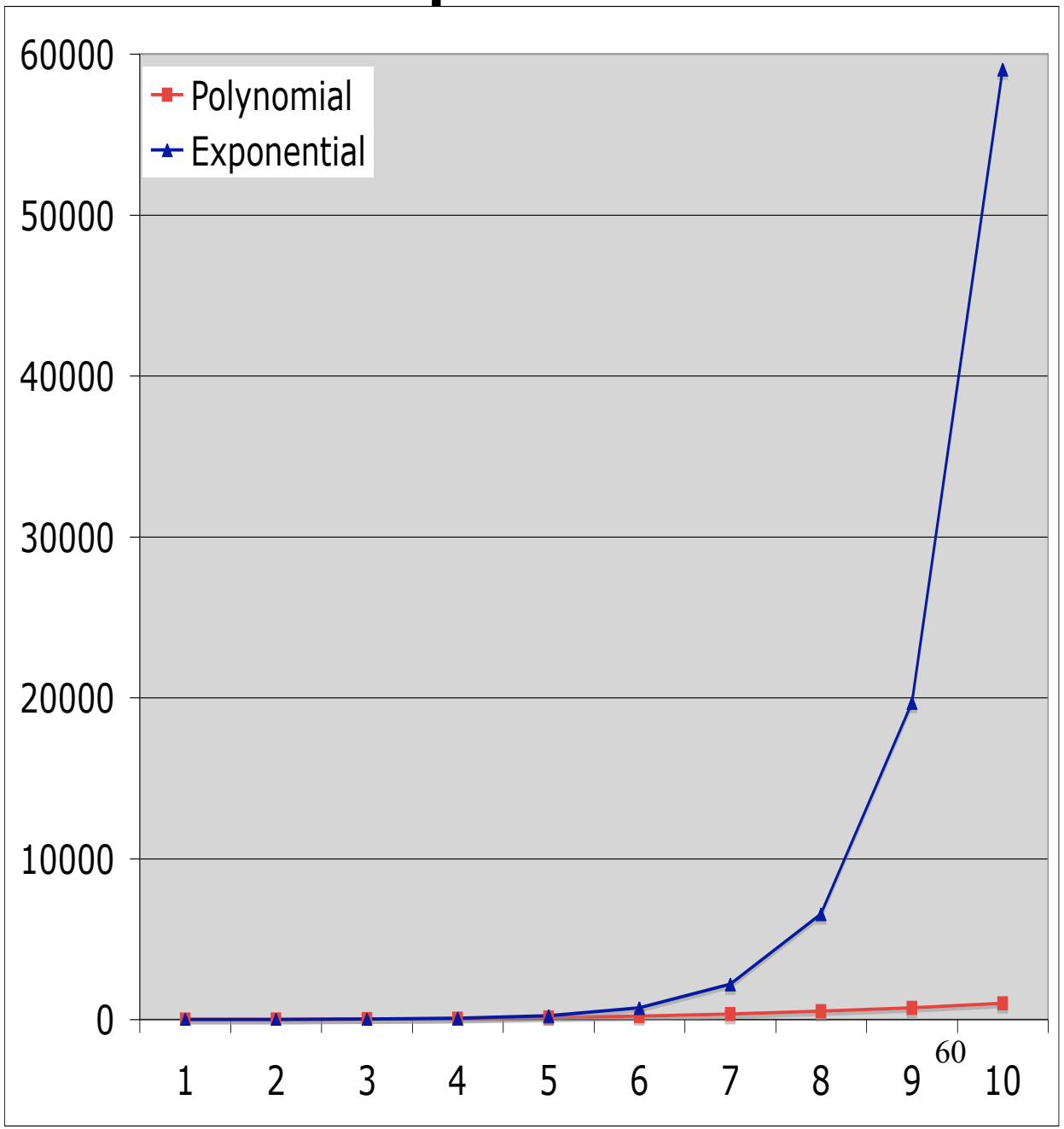
Definition: NTIME(t(n)) = { L | L is decided by a O(t(n))-time non-deterministic Turing machine }

$$TIME(t(n)) \subseteq NTIME(t(n))$$

P and NP

Polynomial vs Exponential

- Polynomial: n^3
- Exponential: 3^n
- n=1 to 10
- What if
  n = 1000?



The class P

$$\mathbf{P} = \bigcup_{k \in \mathbf{N}} \mathbf{TIME}(n^k)$$

- P is the class of languages that are decidable in polynomial time on a deterministic Turing machine

A language *L* is in P if and only if there exists a deterministic Turing machine *M*, such that:

- *M* runs for polynomial time on all inputs
- For all *x* in *L*, *M* outputs YES (1)
- For all *x* not in *L*, *M* outputs NO (0)

The class P

Examples:

1. The class P is known to contain many natural problems, including:

   - Calculating the greatest common divisor
   - Determining if a number is prime

2. Every context-free language is in P

Real Life

- Problems in class P are usually manageable on a real computer
  - n^K
  - though k=100 may introduce some practical problems

The class NP

$$\mathbf{NP} = \bigcup_{k \in \mathbf{N}} \mathbf{NTIME}(n^k)$$

- NP is the class of languages that are decidable in polynomial time on a nondeterministic Turing machine

The class NP

Examples:

1. The decision problem version of the integer factorization problem:
   - given integers n and k, is there a factor f with $1 < f < k$ and f dividing n?
2. The graph isomorphism problem:
   - It is the problem of determining whether two graphs can be drawn identically
3. A variant of the traveling salesman problem:
   - where we want to know if there is a route of some length that goes through all the nodes in a certain network

The class NP-complete

Reducibility

A language A is polynomial time *reducible* to language B, written $A \leq_P B$, if there is a polynomial time computable function f : $\Sigma^* \rightarrow \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a polynomial time reduction of A to B

The class NP-complete

Definition: A language L is NP-complete if:

1. L ∈ NP

2. Every language in NP is reducible to L (in this case L is called: NP-hard)

In other words:

- A problem *p* in NP is NP-complete if every other problem in NP can be transformed (or reduced) into *p* in polynomial time.

NP-Complete Problems

Here is a list of some **NP**-complete problems:

1. Punch-Card Puzzle
2. Discrete Linear Algebra
3. Traveling Salesperson
4. Hamiltonian Path

An example of a **NOT NP**-complete problems:

A_{NTM} is **NOT NP**-complete

HAMILTONIAN PATHS