# INTRODUCTION

## (1.) WHAT IS COMPUTABLE AND WHAT ISN'T?

Example 1

FACTORING: GIVEN x, OUTPUT ALL PRIME FACTORS OF x

IS COMPUTABLE

Example 2

HALTING PROBLEM: GIVEN (P, w) WHERE P IS A C++ PROGRAM, DOES P HALT WHEN RUN ON INPUT w?

NOT COMPUTABLE

TO MAKE THESE QUESTIONS PRECISE WE WILL FIRST NEED TO DEFINE OUR MODEL OF COMPUTATION (TURING MACHINES) AND ARGUE THAT IT CAPTURES ALL CONCEIVABLE COMPUTATION.

USING OUR MODEL, WE CAN CLASSIFY PROBLEMS AS COMPUTABLE OR NOT.

(2) WHAT IS EFFICIENTLY COMPUTABLE AND WHAT ISN'T?

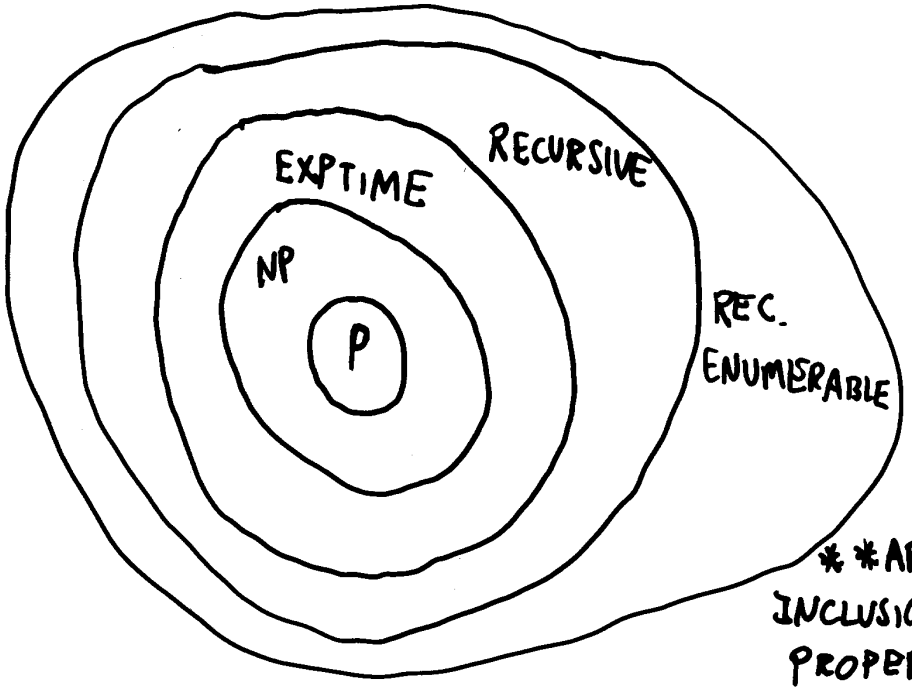
Example 1 FACTORING IS NOT KNOWN TO HAVE AN EFFICIENT SOLUTION.

Example 2 GRAPH REACHABILITY DOES HAVE AN EFFICIENT SOLUTION.

EFFICIENT ≈ POLYNOMIAL TIME

ie, an algorithm that solves the problem within $n^k$ basic operations where $k$ is a small constant and $n = |x|$ is the input length.

THIS GIVES RISE TO A RICH COMPLEXITY HIERARCHY:




**ARE THE INCLUSIONS PROPER ??

③ SOUNDS NICE...
BUT HOW CAN WE COME UP WITH EFFICIENT ALGORITHMS FOR IMPORTANT PROBLEMS?

SOMETIMES A POLYNOMIAL-TIME ALGORITHM WILL BE OBVIOUS.

BUT OTHER TIMES THE PROBLEM SEEMS VERY DIFFICULT, AND AN EFFICIENT SOLUTION REQUIRES A CLEVER / INGENIOUS IDEA.

WE WILL STUDY TWO COMMON PARADIGMS THAT ARE OFTEN USED TO OBTAIN EFFICIENT ALGORITHMS:

a) GREEDY METHOD

b) DYNAMIC PROGRAMMING

# PRELIMINARIES (CHAPTER 0)

AN <u>ALPHABET</u> IS A FINITE SET

<u>Examples</u> $\Sigma_1 = \{0,1\}$

$\Sigma_2 = \{a,b,c,\ldots,x,y,z\}$

A <u>STRING</u> OVER AN ALPHABET IS A <u>FINITE</u> SEQUENCE OF SYMBOLS FROM THE ALPHABET

<u>Example</u> 01001 is a string over $\Sigma_1$

THE <u>LENGTH OF A STRING</u> $w$, $|w|$, IS THE NUMBER OF SYMBOLS IT CONTAINS.

$\varepsilon$ DENOTES THE STRING OF LENGTH ZERO

$\Sigma^*$ DENOTES THE SET OF ALL STRINGS OVER $\Sigma$

A <u>LANGUAGE</u> $L$ OVER $\Sigma$ IS A (POSSIBLY INFINITE) SUBSET OF $\Sigma^*$.

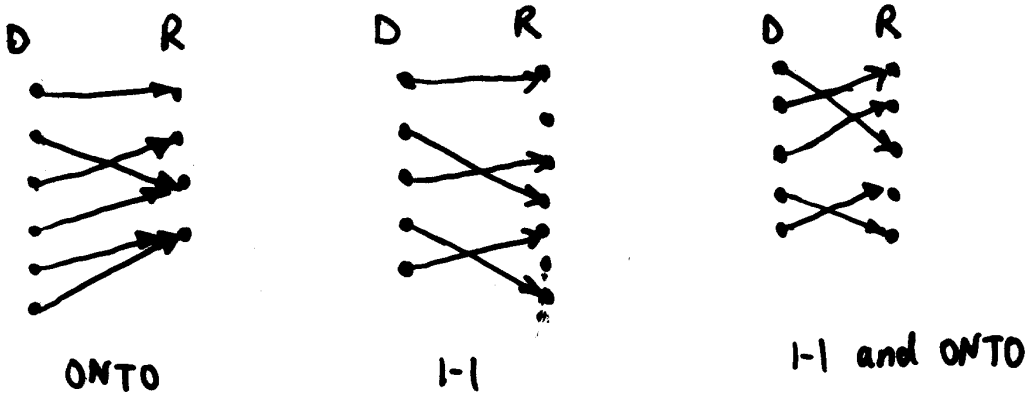
$\phi$ DENOTES THE EMPTY LANGUAGE.

A <u>FUNCTION</u> $f: D \to R$ IS A MAPPING FROM ELEMENTS IN THE DOMAIN $D$ TO ELEMENTS IN THE RANGE $R$.

WE USUALLY CONSIDER FUNCTIONS $f: \Sigma^* \to \Sigma^*$ WHERE $\Sigma$ IS A STANDARD ALPHABET

A FUNCTION $f: D \to R$ IS <u>ONTO</u> IF EVERY ELEMENT $r \in R$ IS MAPPED TO BY SOME $d \in D$.

A FUNCTION $f: D \to R$ IS <u>1-1</u> IF EVERY ELEMENT $r \in R$ IS MAPPED TO BY AT MOST ONE $d \in D$.




A <u>RELATION</u> R IS A FUNCTION WITH RANGE $R = \{TRUE, FALSE\}$

A LANGUAGE CAN BE VIEWED AS A FUNCTION FROM $\Sigma^*$ TO $\{TRUE, FALSE\}$

<u>Example</u> (of a language)

PRIMES $\subseteq \{0,1\}^*$ : $x \in$ PRIMES IFF $x$ VIEWED AS A NUMBER IN BINARY IS PRIME

THE <u>CARTESIAN PRODUCT</u> OF 2 SETS A AND B, <u>A×B</u> IS THE SET OF ALL PAIRS $(a,b)$ WHERE $a \in A$ and $b \in B$.

SIMILARLY FOR $A \times B \times C$, ETC.

# TURING MACHINE BASICS (CHAPTER 3)

OUR GOAL IS TO DEFINE A SIMPLE MATHEMATICAL MODEL THAT WILL BE POWERFUL ENOUGH TO SIMULATE ANY COMPUTATION. THIS WILL ALLOW US TO THINK ABOUT WHAT FUNCTIONS/LANGUAGES ARE COMPUTABLE WITHOUT WORRYING ABOUT LOTS OF DETAILS OF ORDINARY COMPUTERS.

WE FIRST FOCUS ON COMPUTERS AS ACCEPTORS OF LANGUAGES. LATER WE WILL SEE THAT OUR DEF'NS EASILY MODIFY TO HANDLE FUNCTIONS.

**DEF'N**

A TM (ACCEPTING A LANGUAGE) IS A 7-TUPLE

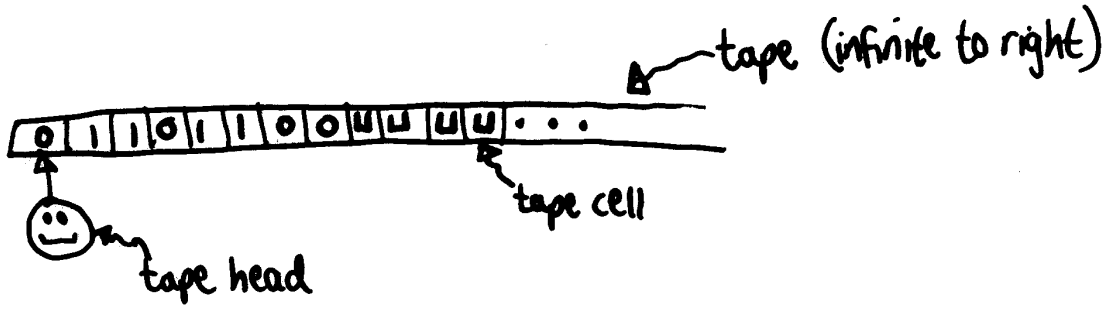
$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

- $Q$: finite set of states including $q_0$, $q_{reject}$, $q_{accept}$
- $\Sigma$: finite set of input symbols not including $\sqcup$
- $\Gamma$: finite set of tape symbols, including $\sqcup$ and $\Sigma$
- $q_0$: start state
- $q_{accept}$: halt and accept state
- $q_{reject}$: halt and reject state; not equal to $q_{accept}$

transition function

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

SEMANTICS OF A TM:




- Initially on input $x = 01101100$, we are in the above "initial" configuration.
- M takes a step according to $\delta$, changing state and printing a symbol, and then moving tape head (cursor)
- If head on leftmost cell/square reading 'a' in state $q$ and $\delta(q,a) \to (r,b,L)$ then head stays on 'a' to prevent head from falling off left end of tape.
- Computation stops whenever M on x enters either $q_{accept}$ or $q_{reject}$
- If M on x halts in $q_{accept}$, then $M(x) = yes$.
  If M on x halts in $q_{reject}$, then $M(x) = NO$.
  If M on x does not halt on x, then $M(x) = \nearrow$.
- $L(M) = \{x \mid M(x) = yes\} \subseteq \Sigma^*$
  ↖ language accepted by M

## EXAMPLES OF TM's

(1) $L = \{w \mid w \in \{0,1\}^*$ and w has an even number of 1's$\}$

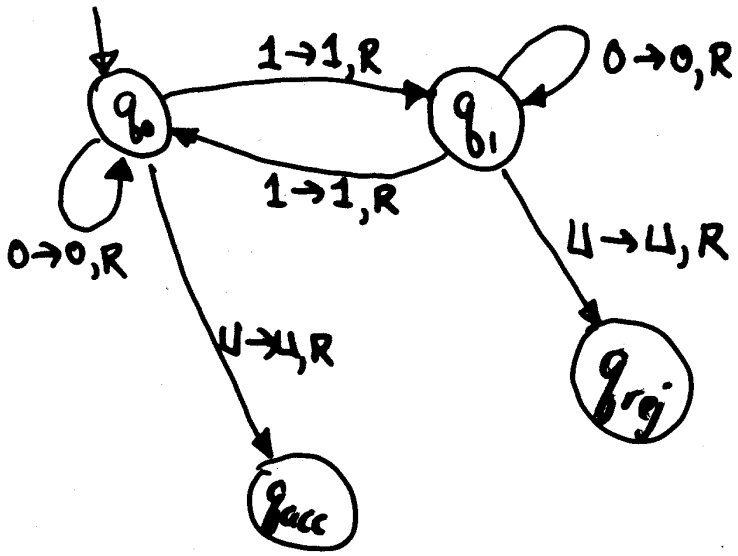
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

$Q = \{q_0, q_1, q_{acc}, q_{rej}\}$

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

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

$\delta$ is described as follows:



② $L_1 = \{x\#x \mid x \in \{0,1\}^*\}$

FIRST WE GIVE AN INFORMAL DESCRIPTION OF A TM $M_1$ SUCH THAT $L(M_1) = L_1$

$M_1$ on input $w$:

1. Scan input to be sure it contains one # symbol. If not, reject.

2. Zig-zag across tape to positions on either side of # to check if positions contain same symbol. If not, reject. Cross off symbols as they are checked.

3. When all symbols to left of # have been checked, check for any remaining symbols to right of #. If any found, reject. Otherwise accept.

# TRANSITION FUNCTION FOR $M_1$ (p. 133)