

# ALGORITHMS - COMPLEXITY

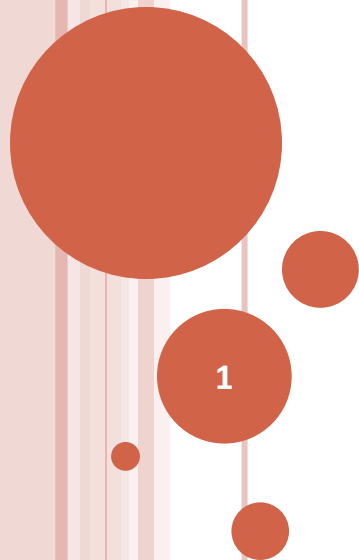
## Computability

- Decidability

- Non-Computable functions:

  - Halting Problem

  - Number of Non-Computable functions.



# EXISTENCE OF NON-COMPUTABLE FUNCTIONS

- Are there problems that are “not computable”?
  - Yes. Based on our definitions of “computability”:
    - Decision problems computable (by TM ) are referred to as decidable problems.
    - There exist proven undecidable problems.
- Theorem: *Halting problem is undecidable*
  - i.e. *There is no program  $H$  such that  $H(P,x)$  will decide whether or not program  $P$  will halt on input  $x$ .*

# HALTING PROBLEM

## ○ Undecidability of Halting problem: [*Proof by Contradiction*]

- Assume that there is a program H such that
  - $H(P,x) = 1$  if P(x) terminates
  - $H(P,x) = 0$  if P(x) does not terminate
- Define D as the procedure:
  - $D(P) \{ \text{while}(H(P,P)) ; \}$
- What is  $H(D,D)$ ?
  - Assume  $H(D,D) == 1$ 
    - Requires  $H(D,D)$  to be 0 i.e. Contradiction.
  - Assume  $H(D,D) == 0$ 
    - Requires  $H(D,D)$  to be 1 i.e. Contradiction.

# HALTING PROBLEM - DIAGONALIZATION

- This proof (with the construction) is an instance of a (proof) technique referred to as **Cantor's diagonalization** :
  - If you can imagine  $H$  as a matrix – rows for programs  $P$  and columns for inputs  $x$ ,
  - then  $H(P,P)$  for any  $P$  represents a diagonal element.

## PROOF OF UNDECIDABILITY – BY REDUCTION

- Implication of  $\pi_1 \preceq \pi_2$  :
  - If  $\pi_1$  cannot be solved, then there can be no algorithm to solve  $\pi_2$ .
- Therefore, one can prove a problem  $\pi_2$  to be undecidable by
  - reducing a known undecidable problem  $\pi_1$  (say Halting Problem) to  $\pi_2$

# HOW MANY NON-COMPUTABLE FUNCTIONS EXIST?

## ○ Theorem NonC:

- *There are more problems that are not computable than there are problems that are computable.*

## ○ If “programs” in a general purpose programming language , say C, solve “computable problems”,

- then the size of the “class of computable problems” is at most the size of the “class of programs”.

## ○ Theorem NonC (rewritten):

- *There are more problems than there are C programs.*

# HOW MANY NON-COMPUTABLE FUNCTIONS EXIST?[2]

## ○ Proof of Theorem NonC [ by Cardinality comparison ]:

- Number of programs in  $\Sigma^*$  – say –  $C$  is equal to  $|\mathbb{N}|$ 
  - By Lemma 1
- Let  $S$  be  $\{ f \mid f \text{ is a function from } \mathbb{N} \text{ to } \{0,1\} \}$ .
- $|S| = 2^{|\mathbb{N}|}$ 
  - By Lemma 2.
- $|S| < |\mathcal{P}(S)|$  for any non-empty set  $S$ , where  $\mathcal{P}(S)$  is the power-set of  $S$ .
  - By Cantor's Power-Set Theorem.

Note:  $2^{|\mathbb{N}|}$  denotes the size of the power set of  $\mathbb{N}$ .

End of Note.

# HOW MANY NON-COMPUTABLE FUNCTIONS EXIST? [2]

## ○ Lemma 1:

- The number of programs in a given programming language, say  $C$ , is equal to  $|\mathbb{N}|$  where  $\mathbb{N}$  is the set of all natural numbers.

## ○ Proof:

- Define a 1-1 onto function from the *set of all strings* in a finite alphabet to  $\mathbb{N}$ :

1. For each  $j$ , number – increasingly – each string of size  $j$ , in lexicographic order (i.e. dictionary order).
2. Repeat step 1 for  $j = 0, 1, \dots$  and continue the numbering from one step to the next.



# HOW MANY NON-COMPUTABLE FUNCTIONS EXIST?[2]

## ○ Lemma 2:

- Consider the set  $S = \{ f \mid f \text{ is a function from } \mathbb{N} \text{ to } \{0,1\} \}$
- $|S| = 2^{|\mathbb{N}|}$ ,  $2^{|\mathbb{N}|}$  denotes the number of subsets of  $\mathbb{N}$

## ○ Proof:

- Map each function  $f$  in  $S$  to a specific subset  $T_f$  of  $\mathbb{N}$  :
  - $f(x)=1$  iff  $x$  is in the subset  $T_f$
- This is a 1-to-1 onto mapping [Why?]