CS F364

Design & Analysis of Algorithms

ALGORITHMS - COMPLEXITY

Computability

- Computation Models: Turing Machine Model

- Church-Turing Hypothesis

- Equivalence of Models : RAM Model

Computability – Machine Models

- Church-Turing Hypothesis:
  - *Anything computable can be computed by a Turing machine.*
- Turing machine:
  - A *Turing machine* is an abstract machine characterized by:
    - states – in which the machine can be
    - (state) transitions – specifying on which input, will the machine go from one state to another and write some output
    - a (semi-infinite) tape – used for input / output

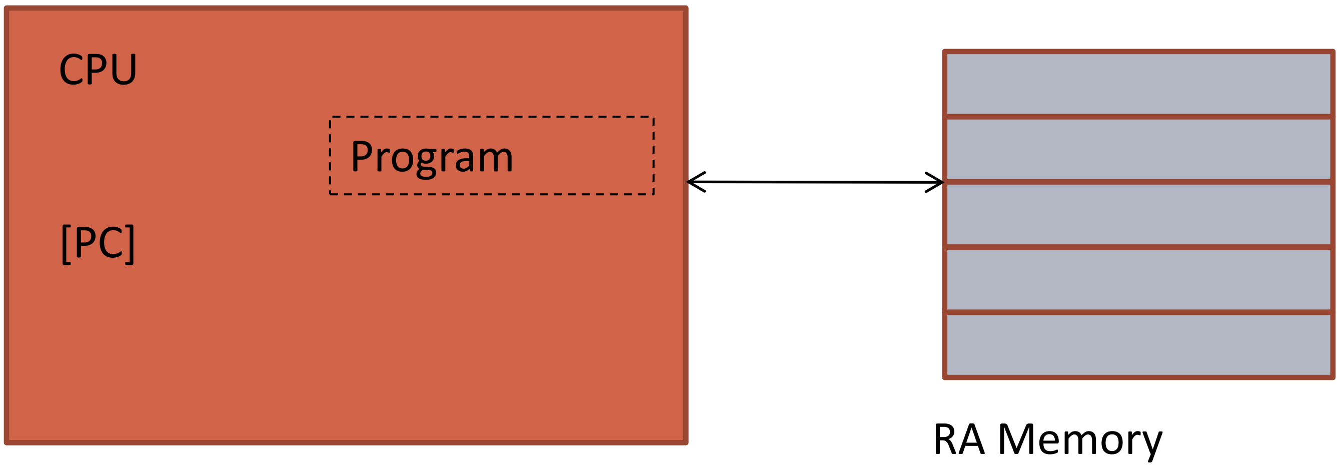
COMPUTABILITY – MACHINE MODELS [2]

- Church-Turing Hypothesis
  - Can't be proven – because "what is computable" is not a grounded notion.
  - In practice, if we mean "computable" to be
    - *computable in finite number of steps,*
    - *each of which can be executed*
      - *in a finite amount of time and*
      - *using a finite amount of resources*

    then the hypothesis is reasonable.
  - It can be disproved – if someone comes up with a "superior" machine model.

COMPUTABILITY – MACHINE MODELS [4]

- Random Access Machine (RAM) Model



Equivalent to Turing machine model.
Capability abstraction of common computers / processors

Exercise: *Prove that the RAM model is equivalent is to the TM model.*

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) { 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 $\pi_1 \precsim \pi_2$:
  - If π_1 cannot be solved, then there can be no algorithm to solve π_2.
- Therefore, one can prove a problem π_2 to be undecidable by
  - reducing a known undecidable problem π_1 (say Halting Problem) to π_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 – say – C is equal to $|\mathbb{N}|$
    - By Lemma 1
  - Let S be { f | f is a function from $\mathbb{N}$ to {0,1}}.
  - $|S| = 2^{|\mathbb{N}|}$
    - By Lemma 2.
  - $|S| < |\mathbb{P}(S)|$ for any non-empty set S, where $\mathbb{P}(S)$ is the power-set of S.
    - By Cantor's Power-Set Theorem.

Note: $2^{|\mathbb{N}|}$ denotes the size of the power set of 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, … and continue the numbering from one step to the next.

How Many Non-Computable Functions Exist?[2]

- Lemma 2:
  - Consider the set S = { f | f is a function from $\mathbb{N}$ 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?]