Unsolvability and The Halting Problem

Can all function be computed?

The Church-Turing thesis stated conditions which a function must satisfy in order for it to be computable.

The Halting Problem:

Simple example of a non-computable function.
Showing that it is non-computable requires showing that a **paradox** exists.

1. Loops - instructions to a computer that are repeated over and over again.

   Repeat as long as X is not equal to 0
   Add 1 to X

2. Does this loop end? (Must check three cases)
   a. Initial value of X is 0
   b. Initial value of X is a negative integer
   c. Initial value of X is greater than 0

3. Statement of the Halting Problem:
   Given a set of instructions for a computer (a program), predict whether or not the program will terminate.
   Do this for any set of instructions.
   Can we always do this? That is, can we prove that we can always predict the outcome of a program?

   A. Running the program is not a solution. Suppose you tried to run the program of part 2c above. We could tired of waiting and abort the program. (No prediction!)

B. Write a computer program that will analyze whether or not another program will terminate, given some initial data.

We will call the method to do this Algorithm A. We will give Algorithm A the program to be tested as a compiled set of instructions. Hence, to Algorithm A, there is no difference between the representation of the program (in binary) and the data (also in binary).

The computer analysis program, called N, will have the following steps:

i. Read a program P
ii. Use Algorithm A to determine if P halts.
iii. If Algorithm A says that P halts, enter an infinite loop (like 2c above).
iv. If Algorithm A says that P never halts, write " Done" and stop.

C. Here is the paradox. Imagine giving program N itself to analyze.

1. If A says that N will terminate, then N will enter an infinite loop and never terminate
2. If A says that N will not halt, then N will write " Done" and terminate.

This paradox shows that the proposed program N can not exist! We conclude that we can not predict for all cases whether or not a program (or an algorithm) will halt. Thus, there are some problems for which we can not find the solution.

Running Time and Infeasibility

Is it possible to solve every computable function?

A function can be computable, yet be infeasible to compute in a reasonable amount of time.

B. Is a function feasible to compute?

To decide if a problem is computable in a reasonable amount of time, one must consider all possible algorithms. If one them can carry out the calculation in a reasonable amount of time (and size), the problem is feasible.

Example: Sorting a set of n elements - test 2 algorithms:

1. Method 1 - The selection sort algorithm:
   - find the largest of the n elements.
   - find the largest of the remaining n-1 elements.
   - find the largest of the remaining n-2 elements.
   ...
   - the smallest element is remaining.

   Work required is $n+ n-1+ \ldots+ 2+ 1 = n(n+ 1)/2$ units

2. Method 2 - Exhaustive listing and search:
   - list all permutations (orderings) of the elements.
   - pick the one that is sorted.

   Work required is $n(n-1)(n-2) \ldots (3)(2)(1) = n!$ units

   e.g., for n= 5:
   method 1 requires 15 units of work
   method 2 requires 120 units of work!