CS 70 Discrete Mathematics and Probability Theory
Spring 2025 Rao

DIS 7B

1 Computability Intro

Note 12

Computability: The main focus is on the Halting problem, and programs that provably cannot exist.

The *Halting problem* is the problem of determining whether a program P run on input x ever halts, or whether it loops forever. It turns out that there does not exist any program that solves this problem.

Using this information, we can prove that other problems also cannot be solved by a computer program, through the use of *reductions*. The main idea is to show that if a given problem can be solved by a computer program `TestX`, then the Halting problem can also be solved by a computer program `TestHalt` that uses `TestX` as a subroutine.

The primary template we'll use for this course is as follows. Suppose we want to show that a program `TestX` does not exist, where `TestX(Q, y)` tries to determine whether a program Q on input y does some task $\mathscr{X}$ (i.e. it outputs "True" if $Q(y)$ does the task $\mathscr{X}$, and it outputs "False" if $Q(y)$ does not do the task $\mathscr{X}$). We can define `TestHalt` as follows (in pseudocode):

```
def TestHalt(P, x):
    def Q(y):
        run P(x)
        do 𝒳
    return TestX(Q, y)  # for some given y
```

Note that this template will be sufficient for our purposes in CS70, but more complex reductions will require more sophisticated programs—you'll learn more about this in classes like CS170 and CS172.

(a) Consider the reduction template given above. Let's break down what it's doing.

We follow an argument by contradiction—we assume that there is a program `TestX(Q, y)` that is able to determine whether another program Q on input y does some task $\mathscr{X}$.

There are two cases: either $P(x)$ halts, or it loops forever. We'd like to show that `TestHalt` as defined above returns the correct answer in both of these cases.

(i) Suppose $P(x)$ halts. What does `TestHalt` return, and why?

(ii) Suppose $P(x)$ loops forever. What does `TestHalt` return, and why?

(iii) What does this tell us about the existence of `TestX`? Briefly justify your answer.

2 Hello World!

Note 12

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program. Throughout this problem, you are allowed to execute programs while surpressing their print statements.

(a) You want to determine whether a program P on input x prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program P prints "Hello World!" while or before running the kth line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program P prints "Hello World!" in the first k steps of its execution. Is there a computer program that can perform this task? Justify your answer.

3 Undecided?

Let us think of a computer as a machine which can be in any of n states $\{s_1, \ldots, s_n\}$. The state of a 10 bit computer might for instance be specified by a bit string of length 10, making for a total of 2^{10} states that this computer could be in at any given point in time. An algorithm $\mathscr{A}$ then is a list of k instructions $(i_0, i_1, \ldots, i_{k-1})$, where each i_ℓ is a function of a state c that returns another state u and a number j describing the next instruction to be run. Executing $\mathscr{A}(x)$ means computing

$$(c_1, j_1) = i_0(x), \qquad (c_2, j_2) = i_{j_1}(c_1), \qquad (c_3, j_3) = i_{j_2}(c_2), \qquad \ldots$$

until $j_\ell \geq k$ for some ℓ, at which point the algorithm halts and returns $s_{\ell-1}$.

(a) How many iterations can an algorithm of k instructions perform on an n-state machine (at most) without repeating any computation?

(b) Show that if the algorithm is still running after $nk+1$ iterations, it will loop forever.

(c) Give an algorithm that decides whether an algorithm $\mathscr{A}$ halts on input x or not. Does your contruction contradict the undecidability of the halting problem?

4 Code Reachability

Note 12

Consider triplets (M, x, L) where

- `M` is a Java program
- `x` is some input
- `L` is an integer

and the question of: if we execute $M(x)$, do we ever hit line L?

Prove this problem is undecidable.