CS 70 Discrete Mathematics and Probability Theory
Spring 2026 Sinclair, Song

DIS 6A

1 Computability Intro

Note 11

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.

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

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

Then we will show that if `TestX` exists, then `TestHalt` would correctly solve the halting problem.

Note that reductions in CS70 will generally look like this template. 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(P', y)` that is able to determine whether another program P' 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.

Solution:

(a) (i) If $P(x)$ halts, then $Q(y)$ will finish executing $P(x)$, and eventually do the task $\mathscr{X}$. This means that `TestX` would return "True", since $Q(y)$ does eventually do $\mathscr{X}$.

(ii) If $P(x)$ loops forever, then $Q(y)$ will get stuck while executing $P(x)$, and will never get to doing the task $\mathscr{X}$. This means that `TestX` would return "False", since $Q(y)$ never does $\mathscr{X}$.

(iii) These answers returned by `TestHalt` exactly solve the Halting problem! However, we've already shown that the Halting problem cannot be solved by a computer program—this is a contradiction. As such, `TestX` cannot exist.

2 Hello World!

Note 11

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.

Solution:

(a) Uncomputable. We will reduce `TestHalt` to `PrintsHW`(P,x).

```
TestHalt(P, x):
  P'(y):
    run P(y) while suppressing print statements
    print("Hello World!")

  return PrintsHW(P', x)
```

If `PrintsHW` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsHW` cannot exist.

(b) Uncomputable. We will reduce `TestHalt` to `PrintsHWByK`(P,x,k).

```
TestHalt(P, x):
    P'(y):
        run P(y) while suppressing print statements
```

```
        print("Hello World!")
    return PrintsHWByK(P', x, 2)
```

Here, we notice that `P'` has only two lines (or at most `len(P) + 1` lines, depending on how this is implemented), and we print "Hello World!" by the last line of `P'` if and only if `P(x)` halts.

Alternatively, we can reduce `PrintsHW`(P,x) from part (a) to this program `PrintsHWByK`(P,x,k):

```
PrintsHW(P, x):
  for i in range(len(P)):
    if PrintsHWByK(P, x, i):
      return true
  return false
```

Note that we technically need to iterate through all the lines here, since there may be large jumps within the code of `P`; this means that we may for example jump from line 1 to line 100 and back to line 2 to print "Hello World!", but `PrintsHWByK`$(P,x,100)$ will return false, since we first reach line 100 without printing "Hello World!".

(c) Computable. You can simply run the program until k steps are executed. If P has printed "Hello World!" by then, return true. Else, return false.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.

3 Code Reachability

Note 11

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 that this problem is undecidable.

Solution: Suppose we had a procedure that could decide the above; call it `Reachable(M, x, L)`. Consider the following example of a program deciding whether $P(x)$ halts:

```
def Halt(P, x):
    def M(t):
        run P(x)   # line 1 of M
        return     # line 2 of M
    return Reachable(M, 0, 2)
```

Program M reaches line 2 if and only if $P(x)$ halted. Thus, we have implemented a solution to the halting problem — contradiction.