CS 70 Discussion 7B

October 18, 2024

Halting Problem

Problem: Given any program `P` and any input `x`, can you create a program `TestHalt(P,x)` that checks if `P(x)` halts?
Solution: No. Consider designing `P`:

```
function P(x):
    if TestHalt(P,x) returns true:
        loop forever
    else:
        return nothing
```

`TestHalt` can never work on our `P` (in fact, `P` is a non-executable program), so a correct implementation of `TestHalt` can never exist (**self-reference** proof!).

Computability

Problem: How do we prove that a problem X can be solved?
Solution: Just write a program `P` that can solve your problem!

Reductions

Problem: How do we prove a problem X can't be solved?
Solution: Prove that being able to solve X implies that I can solve an unsolvable problem. In this class, we often establish the implication:

$$X \text{ can be solved} \implies \text{Halting Problem can be solved}$$

Since we know (ground-truth) that the Halting Problem has no solution, if the above implication is true, then we have no choice but to conclude X can't be solved. Therefore, the Halting Problem **reduces** to problem X (i.e. solving the Halting Problem is at most as hard as solving problem X).