Software Specifications
Intro to Computability

Cain Susko

Queen's University
School of Computing

March 28, 2022

Computability

Computability is an algorithmic problem with:

- infinite set of inputs
- function that associates an output to input.

The solution to this problem is an algorithm that computes the function. For example: for each input the function computes the correct output.

Un-Computability may well defined algorithms problems are unsolvable, which is to say, an algorithm **does not exist**.

Un-Computable Example

The specifications for the halting problem are as follows: given 2 parameters `func` and `arg`:

i function returns true if the file `func` contains a C function definition with one file parameter, and the function terminates if applied to the file `arg`

ii function returns false otherwise

which means that: the function **returns true** if `func` contains a function which halts given the input `arg`

Proving Un-Computability The Technique used for proving that a function is un-computable is called digitalization. For the sake of contradiction, assume we can define:

```
halts(FILE *func, FILE *arg) {...}
```

The implementation of the `halts` function is as follows:

```
int test(FILE *f){
        FILE *a = tmpfile();
        copy(f, a);
        if (halts(f, a)) for (j j) {...}
        return 0;
}
int main(void){
        FILE *f = tmpfile();
        fprintf(f, ...)
        /* write to file f the definition
        of test() */
        return test(f);
}
```

In the main program, the `test()` function is called with a parameter that contains the definition of `test()` such that:

i if inside `test()`, halts returns true then `test()` with it's own description which goes into an infinite loop, which causes `halt` to return an incorrect answer.

ii if halt returns false then `test()` on its own description terminates normally, again `halt` has returned an incorrect answer.

Thus, in all cases `halts` returns an incorrect answer and is a contradiction. Therefore, `halts` cannot be implemented.

Dependence on Language

From a computability point of view, all general purpose languages are equivalent; which is to say they can implement the same class of functions. This has been proven by using the fact that many languages compile to the same low-level language. Furthermore, there exists simple theoretical models that are equivalent (in terms of computability) to general purpose programming languages. For example: Turing Machines.
Strictly speaking, we have only proved that `halts` cannot be implemented in *C*. However, the argument used above will owrk for any programming language (for `halts`).