Software Specifications
Computable Specifications - Reducibility

Cain Susko

Queen's University
School of Computing

March 30, 2022

The Church-Turing Thesis

The thesis claims that any algorithmic problem that has an informed algorithm can be solved by a Turing machine. This thesis is *generally* believed to be true, however it is not possible to prove the CR thesis as 'informed algorithm' does not have a precise definition.

Reducibility

The process described in the previous note was long and tedious, and not something we want to do every time we want to show something is incomputable. Thus, we can use **reduction** in order to show that a function is incomputable by taking a function that has already been proven to be incomputable, and reduce the proven function to the unproven one.

Algorithm function A reduces to function B **if** an algorithm for B can be used to construct an algorithm for A
When A is reducible to B, it can be possible that:

- A is computable then B is incomputable
- A and B are both computable
- both A and B are incomputable

Example

given:

```
bool haltsOnEmpty(FILE *func)
```

which returns **true** if func contains a definition of an int function with one file parameter and that function halts when applied to the empty file. Returns **false** otherwise.
From an initial observation, one may be able to note that `haltsOnEmpty` reduces to `halts` as it is a special case of `halts`. Thus, in order to show `haltsOnEmpty` is incomputable, we must reduce it to `halts`.
Note: The `halts` problem is the only known incomputable function.

In order to reduce the function, we must implement a file transformation such that:

`merge(func, arg, funcArg`

where if `func` contains a function `int f(FILE *a){C}`, then merge writes to FUncArg following definition:

```
inf f(FILE *b)
{
        FILE *a = tmpfile();

        fprintf(a, "%s", "arg");

        C # code C refers to file a
}
```

Now that we have the reduced code, we must show you can create `halts` with `haltsOnEmpty`:

```
bool halts (FILE *func, FILE *arg){
        FILE *funcArg = tmpfile();
        merge(func, arg, funcArg);

        return haltsOnEmpty(funcArg);
```

This works because the function written to `funcArg` operates like `func` on input `arg`.
Assuming that `haltsOnEmpty` satisfies it's specification then the above program implements `halts`
Therefore, `haltsOnEmpty` is incomputable.