

More Computability Problems

Universality and limits in programmability – Part II

I.

Computability theory makes a big deal about the difference between

- a *total* function $f: \mathbb{N} \rightarrow \mathbb{N}$, which is the usual notion of a function: $f(n)$ is defined for any n in \mathbb{N} ; and
- a *partial* function from \mathbb{N} to \mathbb{N} , which may be undefined for certain numbers.

But what's the big deal about partial functions? They are not fundamentally different from total functions, are they? If someone gives you a partial function f from \mathbb{N} to \mathbb{N} , you can express it as a total function this way: Introduce a special symbol \perp to denote “undefined.” Then adjust the definition of f so that $f(n) = \perp$ if $f(n)$ was originally undefined. You can also define $f(\perp) = \perp$. Now we've made f into a total function, by making a completely trivial adjustment.

So how could partial functions be considered *fundamentally* different from total functions in the theory of computation?

II.

Is there an infinite loop in the following program? What can you infer?

```
bool entersInfiniteLoop( int gn, int k );
// returns true if and only if the C function with Godel number gn goes into an infinite loop on input k

bool g( void (*f)(int) );
// returns the Godel number of C function f

void diag( int gn )
{
    if ( ! entersInfiniteLoop( gn, gn ) )
        while(1);
}

void main()
{
    diag( g(diag) );
}
```