Software Analysis: Assignment-1

> Yuxing Xiang, 2000012959

> 停机问题的证明定义在没有输入的函数上，能否改成在带输入的函数上？注意这时Halt(p, i)函数接受两个参数，其中i是输入。

The modification is fairly straightforward. Define the `Evil` function as follows:

```
fn Evil(i) {
  if !Halt(Evil, i) {
    return;
  }
  else {
    loop {}
  }
}
```

Consider `Halt(Evil, i)` for any input `i`. If `Halt(Evil, i)` is true, then by the function definition, `Evil` must go into `loop{}`, and thus won't halt. If `Halt(Evil, i)` is false, then by definition `Evil` will return, and thus will halt. Either way gives a contradition. Hence the function `Halt(p, i)` must not exist.

(Note that the choice of input `i` is irrelevant to the argument.)

> 假设我们把符号分析的抽象域改成{自然数、负、 槑}三个值，其中自然数表示所有正数和零，请写出加法和除法的计算规则，并给出一个式子， 在该抽象域上得到的结果不如原始分析精确。

For brevity, let's denote the abstract field as `{N, Neg, Unk}`, where `N` denotes 自然数, `Neg` denotes 负, and `Unk` denotes 槑.

Here is the rule of addition and division over this field:

```
N + N := N    N + Neg := Unk    N + Unk := Unk
              Neg + Neg := Neg  Neg + Unk := Unk
                                Unk + Unk := Unk
(Addition is commutative)

N / N := Unk    N / Neg := Neg    N / Unk := Unk
Neg / N := Unk  Neg / Neg := N    Neg / Unk := Unk
Unk / N := Unk  Unk / Neg := Unk  Unk / Unk := Unk
```

Consider this expression: `a / (b/b)`, where `a` is `N` and `b` is `Neg`. The rules above can only deduce that `b/b` is `N`; hence it concludes the entire expression is `N / N = Unk`. However, it is clear that `b/b` evaluates to `1`, and thus should never be zero. Hence `a / (b/b)` is actually always `N`.