## 一、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。

> 假设存在 `Halt(p, i)` 函数，若 `p(i)` 函数停机，则返回*true*，若不停机则返回*false*。由于程序本身就是一种数据，所以不妨将程序本身设为输入。可以写出以下恶意程序：
>
> ```
> void Evil(program p) {
>   if (!Halt(p, p)) return;
>   else while (1);
> }
> ```
>
> 当 `Halt(p, p)` 返回*true*，即 p 停机时， `Evil(p)` 死循环。当 `Halt(p, p)` 返回*False*，即 p 死循环时， `Evil(p)` 停机。那么，我们将无法判断 `Halt(Evil, Evil)` 的返回值。
>
> - 当 `Halt(Evil, Evil)` 返回*true*时，说明 `Evil(Evil)` 停机。而 `Evil(Evil)` 停机意味着 `Evil` 函数执行 `return` 语句，即 `Halt(Evil, Evil)` 返回值为*false*。
> - 当 `Halt(Evil, Evil)` 返回*false*时，说明 `Evil(Evil)` 死循环。而 `Evil(Evil)` 死循环意味着 `Evil` 函数执行 `while(1)` 语句，即 `Halt(Evil, Evil)` 返回值为*true*。
>
> 所以，结果与假设存在矛盾，不存在这样的 `Halt(p, i)` 函数。

---

## 二、假设我们把符号分析的抽象域改成{自然数、负、鞣}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。

> 在上述抽象域中，加法的计算规则为：
>
| 抽象+ | 自然数 | 负 | 鞣 |
| --- | --- | --- | --- |
| 自然数 | 自然数 | 鞣 | 鞣 |
| 负 | 鞣 | 负 | 鞣 |
| 鞣 | 鞣 | 鞣 | 鞣 |
>
> 对于以下式子：
>
> $$a+b+(-b)$$
>
> 抽象分析为：不论$a, b$是自然数还是负数，式子的符号永远为“鞣”
>
> 原始分析为：不论$b$是自然数还是负数，式子的符号永远和$a$一致
>
> 显然原始分析更加精确

在上述抽象域中，除法的计算规则为：

| 抽象/ | 自然数 | 负 | 槑 |
| --- | --- | --- | --- |
| 自然数 | 槑 | 槑 | 槑 |
| 负 | 槑 | 自然数 | 槑 |
| 槑 | 槑 | 槑 | 槑 |

对于以下式子：

$$x/(-x)$$

抽象分析为：不论$x$是自然数还是负数，式子的符号永远为“槑”

原始分析为：当$x$为负时，式子符号为负，其他情况为“槑”

显然原始分析更加精确