

C 编译器优化的 Bug

一个朋友向我指出一个最近他们发现的 GCC 编译器优化过程（加上 -O3 选项）里的 bug，导致他们的产品出现诡异的行为。这使我想起以前见过的一个 GCC 的 bug。当时很多编译器专家认为那种做法是正确的，跟他们说不清楚。简言之，这种有问题的优化，喜欢利用 C 语言的“未定义行为”（undefined behavior）进行推断，最后得到奇怪的优化结果。

这类优化过程的推理方式都很类似，他们使用一种看似严密而巧妙的推理，例如：“现在有一个整数 x ，不知道是多少。但 x 出现在一个条件语句里面，如果 $x > 1$ ，那么程序会进入未定义行为，所以我们可以断定 x 的值必然小于或者等于 1，所以现在我们可以利用 $x \leq 1$ 这个事实来对相关代码进行优化……”

看似合理，却是不正确的。举一个具体的例子吧。这篇 Chris Lattner 写于 2011 年的[文章](#)里面就含有这样一个优化。文中指出，编译器利用“未定义行为”进行优化，是合理的，对于性能是很重要的，并且举出这样一个例子：

```
void contains_null_check(int *P) {
    int dead = *P;
    if (P == 0)
        return;
    *P = 4;
}
```

这例子跟我之前看到的 GCC bug 不大一样，但大致是类似的推理方式：这个函数依次经过这样两个优化步骤（RNCE 和 DCE），之后得出“等价”的代码：

```
void contains_null_check_after_RNCE(int *P) {
    int dead = *P;
    if (false) // P在上一行被访问，所以这里P不可能是null
        return;
    *P = 4;
}

void contains_null_check_after_RNCE_and_DCE(int *P) {
    // int dead = *P; // 死代码
    // if (false) // 死代码
    //     return; // 死代码
    *P = 4;
}
```

他的推理方式是这样：

1. 首先，因为在 `int dead = *P` 里面，指针 `P` 的地址被访问，如果程序顺利通过了这一行而没有出现未定义行为（比如当掉），那么之后 `P` 就不可能是 `null`，所以我们可以把 `P == 0` 优化为 `false`。
2. 因为条件是 `false`，所以整个 `if` 语句都是死代码，被删掉。
3. `dead` 变量赋值之后，没有被任何其它代码使用，所以对 `dead` 的赋值是死代码，可以消去。

最后函数就只剩下一行代码 `*P = 4`。然而经我分析，发现这个转换是错误的，而不只是像他说的“存在安全隐患”。现在我来考考你，你知道这为什么是错的吗？庆幸的是，现在如果你把这代码输入到 Clang，就算加上 -O3 选项，它也不会给你进行这个优化。这也许说明他们也许已经意识到了这个错误。

我写这篇文章的目的其实是想告诉你，不要盲目相信编译器的优化变换都是正确的。无论它看起来多么的合理，只要打开优化之后你的程序出现不合理的行为，你就不能排除编译器进行了错误优化的可能性。Lattner 指出这样的优化完全符合 C 语言的标准，但就算你符合国际标准，也有可能是错的。有时候你得相信自己的直觉……