

程序分析Lab2实验报告

22210240026 梁超毅

实现方法

本次Lab文档由浅入深、清晰易懂。本次实验根据Lab文档的要求补全了 `normalize` , `isEmpty` , `filter` , `assign_case2` 和 `assign_case3` 五个函数, 实现方法如下:

normalize

对Zone的标准化需要找出每个变量之间的最紧约束, 等价于把DBM视为一个图并求任意两个节点间的最短路径。因此参考文档提示, 补全了 `normalize` 函数中的Floyd算法, 如下:

```
for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            ret._dbm[i][j] = std::min(ret._dbm[i][j], ret._dbm[i][k] +
ret._dbm[k][j]);
```

Floyd算法的大致思路是用图中的每个节点对图中的任意两个节点间的距离进行松弛。上述代码实现了这个逻辑。

isEmpty

参考文档提示, `isEmpty` 等价于Floyd算法执行后存在节点 i 使得 $v_{ii} < 0$, 因此补全代码如下:

```
ZoneDomain res = this->normalize();
for (int i = 0; i < n; i++)
    if (res._dbm[i][i] < 0)
        return true;
```

第一行对DBM做了标准化, 二至四行判断标准化后的DBM中是否存在节点 i 使得 $v_{ii} < 0$, 如果存在则立即返回 `true` 。如果不存在, 则继续执行后续代码行 (返回 `false`) 。

filter

该函数为DBM添加约束 $x - y \leq c$, 即 $v_j - v_i \leq c$, 对应的元素为 m_{ij} 。可以看到, i 为变量 y 的下标, j 为变量 x 的下标, 因此补全代码如下:

```
size_t j = getID(x);
size_t i = getID(y);
ret._dbm[i][j] = std::min(this->_dbm[i][j], c);
```

可以不对 `x` 和 `y` 做空字符串判断的原因是空字符串的下标被设置为了0，因此只需要正常调用 `getID` 即可。

assign_case2

参考文档提示，case2形式赋值等价于对DBM进行一次forget操作和两次filter操作，因此补全代码如下：

```
ret = this->forget(x).filter(x, y, c).filter(y, x, -c);
```

使用正确的参数调用forget和filter函数即可。

assign_case3

case3形式的赋值需要设置 `x` 的范围为 $[l, r]$ ，等价于 $-x \leq -l$ 且 $x \leq r$ 。参考文档提示，补全代码如下：

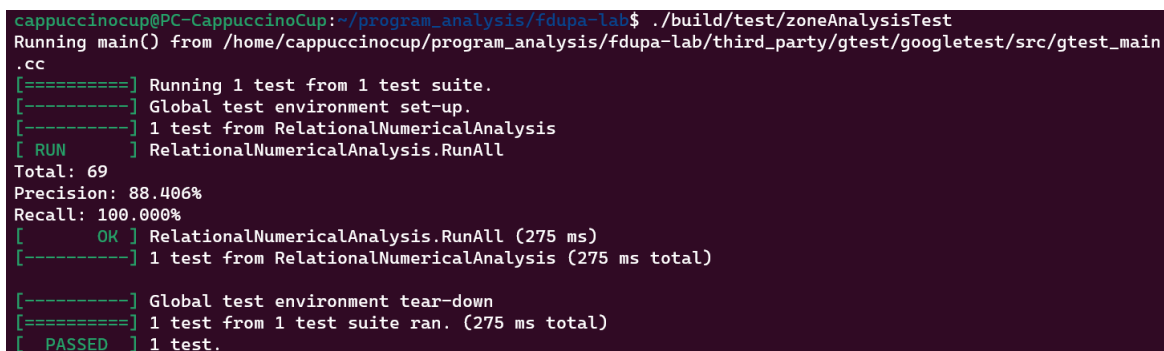
```
ZoneDomain ret = this->forget(x);
size_t i0 = getID(x);
ret._dbm[i0][0] = -l;
ret._dbm[0][i0] = r;
```

注意这里必须先调用forget再设置 m_{i0} 和 m_{0i} 的值。

上述5个函数总计补充了16行代码，与todo标注的预估代码量一致。

分析结果

运行测试的截图：



```
cappuccinocup@PC-CappuccinoCup:~/program_analysis/fdupa-lab$ ./build/test/zoneAnalysisTest
Running main() from /home/cappuccinocup/program_analysis/fdupa-lab/third_party/gtest/googletest/src/gtest_main.cc
[=====] Running 1 test from 1 test suite.
[-----] Global test environment set-up.
[-----] 1 test from RelationalNumericalAnalysis
[ RUN     ] RelationalNumericalAnalysis.RunAll
Total: 69
Precision: 88.406%
Recall: 100.000%
[ OK      ] RelationalNumericalAnalysis.RunAll (275 ms)
[-----] 1 test from RelationalNumericalAnalysis (275 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test suite ran. (275 ms total)
[ PASSED ] 1 test.
```

实验的召回率为100%，准确率为88.406%，符合实验要求。

所有测试用例中存在8个错误，其分布为：

- branch2.fclang: 错误2个
- loop1.fclang: 错误1个
- loop3.fclang: 错误2个
- loop4.fclang: 错误1个
- loop5.fclang: 错误1个
- rel4.fclang: 错误1个

上述错误原因将在下一节中与Lab1的对比一起进行深入分析。

与Lab1对比

Lab1和Lab2中所有出错的测试用例结果如下表所示：

测试用例	Lab1错误数	Lab2错误数
branch2	2	2
loop1	1	1
loop3	4	2
loop4	0	1
loop5	0	1
rel1	1	0
rel2	1	0
rel3	1	0
rel4	3	1
总计	13	8
准确率	81.159%	88.406%
召回率	100.000%	100.000%

Lab1原有的测试用例在Lab1实验报告中已经做过分析，不在此赘述。

而Lab2在branch2、loop1以及loop3中出现问题的原因与Lab1相似（具体参考Lab1 Report中的分析），即：内循环**无法将某些初始范围过滤**，导致得到的结果的范围相比正确范围多出了一些值（比如一个0）。虽然Lab2加入了变量之间的关联性，但依然没有解决这个问题。

另一方面，在新增的四个测试用例的9个check上，Lab1出现了6个错误，原因与之前的几个例子类似，即：无法准确表征变量之间的关联性。而这个问题在Lab2中得到了解决。可以看到，在这9个check上，Lab2只出现了1个错误，远比Lab1准确。从这里可以看出关系型抽象域相比区间抽象域的优势。

但即便如此，在原有的测试用例的60个check中，Lab1和Lab2的准确率却是**相同的**（均为88.333%）。这从另一方面说明了关系型抽象域依然有其局限性。