程序分析Lab1实验报告

22210240026 梁超毅

算法流程

本次实验的算法在Lab文档提供的参考资料的基础上进行了扩展：

- 扩展了集合的构造和运算（交、并、补），以支持范围操作
- 扩展了条件语句的处理，以适配实验要求中对while的处理

算法的大致流程如下：

1. 对 IR 进行初步扫描：
    - 为每条 Goto 或 If 类型的 IR 记录一个数据结构，包含：
        - 发生跳转的条件，包括变量 `x` 及其范围 `Cond`
        - 上次跳转时每个变量的范围，记为 R'
    - 为每条 ChechInterval 类型的 IR 记录一个数据结构，包含：
        - 这条 IR 检查的变量 `x` 的范围，记为 $CheckR$
        - 这条 IR 执行时变量 `x` 的范围，记为 $RealR$
2. 从第一条 IR 开始按类型依次进行处理：
    - （假设从前序 IR 传入的变量的范围为 R）
    - CheckInterval：更新 $RealR$ 为 R
    - Goto：更新 R' 为 R
    - If：
        1. 如果 $R(x) \cap Cond$ 为 $\emptyset$，设置 R' 为 $\emptyset$。如果 $R(x) - R(x) \cap Cond$ 为 $\emptyset$，设置 R 为 $\emptyset$ （$Cond$为跳转条件，x 为 $Cond$ 判断的变量）
        2. 如果1不符合：如果 $R'(x)$ 不等于 $R(x) \cap Cond$，更新 $R'(x)$ 为 $R(x) \cap Cond$，$R'(others)$ 为 $R(others)$；否则不改变 R' （$others$ 为除 x 外的其它所有变量）
        3. 如果1不符合：更新 R 为 $R - R \cap Cond$
    - Label：找到以这条 IR 为跳转目标的指令的 R'，然后将 R 设置为 $R \cup R'$
    - 其它：根据这条 IR 的语义更新 R
    - （之后，将 R 往后继 IR 传递）
3. 重复执行步骤2，直到一轮迭代结束后所有 R' 都没有发生变化
4. 将每条类型为 CheckInterval 的 IR 检查的范围 $CheckR$ 和这条 IR 的记录的 $RealR$ 对比：
    - 如果 $RealR = \emptyset$，表示程序没有执行到这条 IR，标记结果类型为 `UNREACHABLE`
    - 如果 $RealR \subseteq CheckR$，标记结果类型为 `YES`
    - 如果不是上述两种情况，标记结果类型为 `NO`

算法实现

在提交的`intervalAnalysis.h`中实现了四个类，分别为：

- `Range`，实现了单个集合的构建和各类运算，用于表示一个变量的范围
- `VarRange`，基于`Range`实现了多个变量及其范围的集合的构建和运算
- `BranchInfo`，对应算法流程中 Goto 或 If 类型的 IR 记录的信息
- `CheckInfo`，对应算法流程中 CheckInterval 类型的 IR 记录的信息

在提交的`intervalAnalysis.cpp`中实现了三个函数，分别为：

- `init()`，对应算法流程中的步骤1
- `iter()`，对应算法流程中的步骤2和3
- `done()`，对应算法流程中的步骤4

分析结果

运行测试的截图：

```
cappuccinocup@PC-CappuccinoCup:~/program_analysis/fdupa-lab$ ./build/test/intervalAnalysisTest
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 IntervalAnalysis
[ RUN      ] IntervalAnalysis.RunAll
Total: 60
Precision: 88.333%
Recall: 100.000%
[       OK ] IntervalAnalysis.RunAll (80 ms)
[----------] 1 test from IntervalAnalysis (80 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (80 ms total)
[  PASSED  ] 1 test.
```

实验的召回率为100%，准确率为88.333%，符合实验要求。

所有测试用例中存在7个错误，其分布为：

- branch2.fdlang：错误2个
- loop1.fdlang：错误1个
- loop3.fdlang：错误4个

其中，loop3.fdlang中有3个错误源于分析出的内循环变量范围多出了一个“0“。出现这个问题的原因将在下一节作深入分析。

自编测试用例

这里构造了一个简单的双层嵌套循环来说明算法在loop3中遇到的问题：

```
x = 0;
y = 0;

while (x < 1) {
    while (y < 2) {
        y = y + 1;
    }
    x = x + 1;
}

check_interval(x, 1, 1);
check_interval(y, 2, 2);
```

其对应的 IR 为：

```
L0 :  x = 0;
L1 :  y = 0;
L2 :
L3 :  if x < 1 then goto L5;
L4 :  goto L15;
L5 :
L6 :
L7 :  if y < 2 then goto L9;
L8 :  goto L12;
L9 :
L10:  y = y + 1;
L11:  goto L6;
L12:
L13:  x = x + 1;
L14:  goto L2;
L15:
L16:  check_interval(x, 1, 1);
L17:  check_interval(y, 2, 2);
```

当内循环中的变量范围稳定时，`y`的范围是2，此时并不存在问题。

然而，当内循环结束时，L14处的`goto L2`会传递一个范围`x: [1, 1], y: 2`到L2处。合并后，`y`的范围在L2处就会变成`0 ∪ 2`，然后通过L4处的`goto L15`传递给L17（比`y`的实际范围多出一个0）。

之所以内循环会出现这个问题，而最外层循环不会，是因为最外层循环的判断条件（`x < 1`）会将`x`的范围过滤，**但并不会将`y`的范围过滤。**这导致`y`的初始值0绕过了L7的判断，被传递到了程序末尾。要解决这个问题，需要将`x`和`y`之间的范围关联起来，而且类似的方法也能解决branch2中的两个错误。

但这是一种启发式的方法。如果要在原理层面上进行改进，可能需要对算法进行较大的修改，因此并不是本次实验的重点。