Lab1 - Interval Analysis

Overview

你需要在一门架空的语言 `FDlang` 上实现一个简单的区间分析（一种数据流分析），并简单分析它的性能与精度。

Build from source

本 Lab 使用 C++ 语言，为了能够顺利编译源代码，你需要：

- 一个支持 C++17 标准的现代编译器如 clang++/g++/msvc++
- CMake (>=3.0.0)

推荐使用 Linux/Mac 系统完成本 Lab，以下为 Linux 的指南：

1. 克隆本项目至本地并进入根目录
2. 使用 CMake 构建项目

```
cmake -B build
```

3. 编译源代码

```
cmake --build build -j32
```

一切顺利的话可以在 `build/tools/` 里发现可执行文件 `fdlang`，运行该程序应该出现以下字样

```
Usage: fdlang [-format] [-modelchecker] [-interval-analysis] [-dumpir] path-to-src-file
e.g.: fdlang -interval-analysis src.fdlang
```

Windows 用户的 CMake 构建与编译请自行度娘，装过 VS 的话应该自带 MSVC++ 和 Ninja。

FDlang Introduction

该语言你可以认为是一个非常简单的玩具语言，首先它无类型（或者说所有类型都是无符号整型），支持加减运算，如

```
x = a + 5;
y = 6 - x;
z = 7;
t = z;
```

整数的值只能在 $[0, 255]$ 中，a + b 的结果为 $\min\{a+b, 255\}$，a - b 的结果为 $\max\{a-b, 0\}$。

运算无法嵌套，所以以下写法是**不合法**的

```
x = 1 + 3 - 2;
```

使用变量时不需要声明，每个变量的作用域为全程序，使用未初始化过的变量将自动赋初值为 0。

该语言包含两种控制语句，分别是 `if` 语句和 `while` 语句，以一段简单的程序为例

```
function case1(){
    x = input();
    y = input();

    while (y >= 1) {
        check_interval(y, 1, 10);
        if (x <= 10) {
            y = y + 1;
        } else {
            y = y - 1;
        }
    }

    check_interval(y, 0, 11);
}
```

在使用 `while` 循环语句和 `if` 分支语句时，大括号不能省略，`else` 分支不能省略。为了分析的方便，判断条件左边一定是一个变量，右边一定是整数常量，比较运算符包含 `==`、`>=`、`>`、`<=`、`<`。

本 Lab不涉及函数调用，`input()` 可以看作是一个关键字，表示用户输入一个 $[0, 255]$ 内任意的一个整数；`check_interval(y, 0, 11)` 表示检查变量 y 在该程序点的值是否在区间 $[0, 11]$ 内。

除此之外你可以使用 `nop` 语句，功能与 python 中的 `pass` 语句一样，例如

```
function case2(){
    if (x >= 5) {
        x = x - 5;
    } else {
        nop;
    }
}
```

FDlang IR Introduction

在本 Lab 中，我们的分析建立在语言的中间表示 (Intermediate Representation) 之上，其定义可以在 `lib/IR/IR.h` 中找到。

IR 帮助我们把原程序简化成了一个线性的结构，下面展示一段源代码和其对应的 IR

```
function case3(){
    x = 1;

    while (x < 10) {
        y = y + x;
        x = x + 1;
        check_interval(x, 0, 9);
    }

    check_interval(y, 45, 45);
}
```

```
L0 :  function case3(){
L1 :
L2 :  x = 1;
L3 :
L4 :  if x < 10 then goto L6;
L5 :  goto L11;
L6 :
L7 :  y = y + x;
L8 :  x = x + 1;
L9 :  check_interval(x, 0, 9);
L10:  goto L3;
L11:
L12:  check_interval(y, 45, 45);
L13:
L14:  }
L15:
```

IR 中变量和整数常量被统称为 `Value`；指令（语句）为 `Inst`，分为以下几类：

- `AddInst` 加法指令：`operand0 = operand1 + operand2;`
- `SubInst` 减法指令：`operand0 = operand1 - operand2;`
- `InputInst` 输入指令：`operand0 = input();`
- `AssignInst` 赋值指令：`operand0 = operand1;`
- `CheckIntervalInst` 检查指令：`check_interval(operand0, operand1, operand2);`
- `IfInst` 有条件跳转指令：`if operand0 cmpop operand1 then goto dest;`
- `GotoInst` 无条件跳转指令：`goto dest;`
- `LabelInst` 标签
- `CallInst` 函数调用指令：`call function ( param0,param1, ... );`

更详细的内容可以阅读源码 `lib/IR/IR.h`。

Your Task

编写 `analysis/intervalAnalysis.cpp` 与 `analysis/intervalAnalysis.h`，你无须修改除此之外的任何文件。

目标是对于源码中的每个形如 `check_interval(x, l, r)` 的语句（l 和 r 一定是常量），计算它的结果为 `YES` 或 `NO` 或 `UNREACHABLE`：

- `YES` 表示你认为存在一条执行路径能够执行到该语句，且对于任何能执行到该语句的执行路径，变量 x 在该语句处的值都在 $[l, r]$ 内
- `NO` 表示你认为存在一条执行路径能够执行到该语句，且存在一条可能的执行路径使得变量 x 的值在该语句处不在 $[l, r]$ 内
- `UNREACHABLE` 表示你认为这块代码是死代码

你只需要将计算出的结果填写到 `IntervalAnalysis` 类的 `results` 成员变量内即可，注意每句 `check_interval` 应该恰好对应一个结果，不能多也不能漏。

完成后你可以通过运行 `build/test/intervalAnalysisTest` 来检查正确率与召回率，输出形如

```
Total: 60
Precision: 90.000%
Recall: 100.000%
```

注意：因为我们必须保证分析结果是 Sound 的（即真实的程序行为是我们认为可能发生的程序行为的子集，换句话说我们只允许分析出来的数的取值范围大于等于它真实的可达到的范围），所以你的分析程序允许把 `UNREACHABLE` 判断成 `YES` 或 `NO`，或者把 `YES` 判断成 `NO`，但不允许有相反的情况。即允许正确率不为 100% 但召回率必须为 100%。实现正确在原本的测试集上可以达到 90% 左右正确率，不做过高要求。

所有需要用到的接口、函数等都在 `analysis/intervalAnalysis.h` 与 `lib/IR/IR.h` 中，无需花时间看其他的代码。

你需要提交：

- 源代码，仅包含 `analysis/intervalAnalysis.cpp` 与 `analysis/intervalAnalysis.h`
- 简单的报告，包括方法、分析结果，可以自己构造几个测试用例

除了课上讲的，可以参考的一些资料：

- https://zhuanlan.zhihu.com/p/325018769
- 北京大学公开课 软件分析技术 熊英飞 https://www.bilibili.com/video/BV1Rt4y1s7tC P2/P3
- 南京大学 软件分析 李樾、谭添 https://www.bilibili.com/video/BV1oE411K79d/ P3/P4

Tools

格式化代码（没啥用）

```
fdlang -format xxx.fdlang
```

打印 IR

```
fdlang -dumpir xxx.fdlang
```

模型检测（你可以认为它的结果是一定对的，但比较慢）

```
fdlang -modelchecker xxx.fdlang
```

区间分析（你的实现）

```
fdlang -interval-analysis xxx.fdlang
```

Bug Report

编者能力有限，发现 bug 或有疑问可以联系 23110240130@m.fudan.edu.cn

CFL

Lab 并不需要关心这些，仅供参考。

```
Value      := IDENTIFIER
            | NUMBER

Cond       := IDENTIFIER < NUMBER
```

```
            | IDENTIFIER > NUMBER
            | IDENTIFIER >= NUMBER
            | IDENTIFIER <= NUMBER
            | IDENTIFIER == NUMBER

Args := ε | IDENTIFIER | IDENTIFIER , Args

BinaryAssignStmt :=
              IDENTIFIER = Value + Value ;
            | IDENTIFIER = Value - Value ;

UnaryAssignStmt  :=
              IDENTIFIER = input ( ) ;
            | IDENTIFIER = IDENTIFIER ;
            | IDENTIFIER = Value ;

AssignStmt := BinaryAssignStmt
            | UnaryAssignStmt

IfStmt     := if ( Cond ) { Stmts } else { Stmts }

WhileStmt  := while ( Cond ) { Stmts }

CallStmt   := call IDENTIFIER ( Args ) ;

CheckStmt  := check_interval ( IDENTIFIER , NUMBER , NUMBER ) ;

NopStmt    := nop ;

Stmt       := AssignStmt
            | IfStmt
            | WhileStmt
            | CheckStmt
            | NopStmt
            | CallStmt

Stmts      := Stmt^+

FunctionNode    := function IDENTIFIER ( Args ) { Stmts }

FunctionNodes   := FunctionNode^+
```