Lab2 - Relational Numerical Analysis (Zone)

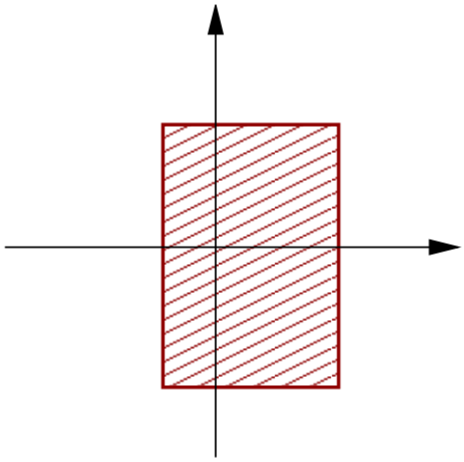
语言特性与项目编译见 Lab1 的文档。

Motivation

大家很熟悉的区间抽象域记录了在每个程序点，每个变量的取值范围在 $[a, b]$ 内，即描述了

$$\bigwedge_i (V_i \in [a_i, b_i])$$

如果程序仅包含两个变量，那它可以看成是二维平面上的一个矩形



但区间抽象域的表达能力较弱，比如以下例子

```
1 i = input();
2 j = i;
3 while(i > 0) {
4     i = i - 1;
5     j = j - 1;
6 }
7 check_interval(j, 0, 0); // fail when using interval domain
```

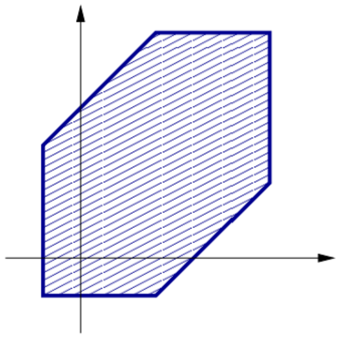
普通的区间抽象域无法证明第七行一定是对的，本质原因在于使用区间刻画程序状态忽略了变量间的关联性，程序无法得知每次执行循环体后始终有 $x = y$。为了能够从程序中获得多个变量之间的不变量，需要引入关系型抽象域。

Zone 是最简单的关系型抽象域之一，在每个程序点，它记录了变量之间差的范围，即描述了

$$\bigwedge_{ij} (V_j - V_i \leq c_{ij})$$

特殊地，我们令 V_0 为一个恒为 0 的常量，以表示 $V_i \in [-c_{i0}, c_{0i}]$ 这样的约束。

对比区间抽象域，如果程序仅包含两个变量，那它可以看成是二维平面上的这样一个区域



上述例子在采用 Zone 抽象域后就能不产生误报

```
// j = [0, 0]
// i = [0, 0]
i = input();
// j = [0, 0]
// i = [0, 255]
// j - i <= 255
// i - j <= 0
j = i;
// j = [0, 255]
// i = [0, 255]
// j - i <= 0
// i - j <= 0
while(i > 0) {
    // j = [1, 255]
    // i = [1, 255]
    // j - i <= 0
    // i - j <= 0
    i = i - 1;
    // j = [1, 255]
    // i = [0, 254]
    // j - i <= -1
    // i - j <= 1
    j = j - 1;
    // j = [0, 254]
    // i = [0, 254]
    // j - i <= 0
    // i - j <= 0
}
// j = [0, 0]
// i = [0, 0]
// j - i <= 0
// i - j <= 0
check_interval(j, 0, 0); // ok when using zone domain
```

Zone & Normal Form

Zone 通常用 Difference Bound Matrices(DBMs) 来描述，假设程序中有 n 个变量，那它就是一个 $(n+1)\times(n+1)$ 大小的矩阵 m，其中：

- $m_{ij} \neq +\infty$ 表示 $V_j - V_i$ 的上界；
- $m_{ij} = +\infty$ 表示 $V_j - V_i$ 是无界的，即没有约束；
- m_{i0}, m_{0j} 用来描述一元的约束：$-V_i \leq m_{i0}, V_j \leq m_{0j}$。

你可以把 m 看成这 $n+1$ 个点构成的有向带权图的邻接矩阵，其中每条边是 $V_i \xrightarrow{m_{ij}} V_j$。

不一样的 DBM 有可能描述的区域是一样的，比如

$$\begin{cases} V_1 - V_2 \leq 3 \\ V_2 - V_3 \leq -1 \\ V_1 - V_3 \leq 4 \end{cases}$$

等价于

$$\begin{cases} V_1 - V_2 \leq 3 \\ V_2 - V_3 \leq -1 \\ V_1 - V_3 \leq 2 \end{cases}$$

所以我们需要定义一个标准型，~~不难发现 DBM 其实就是一个差分约束系统~~，把 m 看成这 $n+1$ 个点构成的有向带权图的邻接矩阵，其中每条边是 $V_i \xrightarrow{m_{ij}} V_j$，对这个图求个多源最短路，其最短路矩阵就是标准型了（即不那么紧的限制都被求最短路时的松弛操作干掉了）。

我们默认所有操作执行完后都要进行转化为标准型。

Operations

Zone 抽象域需要支持以下操作：

Normalization. 把 Zone 转换成标准型。用 Floyd 求个多源最短路即可，不会的同学可以参考 wikipedia。

Emptiness Testing. 判断 Zone 表示的区域是否为空。我们可以通过此操作来判断当前程序状态是否可达。

区域为空其实就是 DBM 对应的图里有负环（即约束自相矛盾了），可以通过 Floyd 或者 Bellman-Ford 判负环得到，最简单的方式是转换成标准型然后判断是否存在 $v_{ii} < 0$。

Equality and Inclusion Testing. 判断两个 Zone 是否相等或一者为另一者的上界。

对于两个 DBM m 和 n，定义

$$\begin{aligned} m = n &\iff \forall i,j.\, m_{ij} = n_{ij} \\ m \sqsubseteq n &\iff \forall i,j.\, m_{ij} \leq n_{ij} \end{aligned}$$

Projection. 从 Zone 中获得某变量对应的区间抽象域。定义 DBM m 在变量 v_k 上的投影为所有满足约束 m 的取值中变量 v_k 可能的取值构成的集合，记作 $\pi_{|v_k}(m)$。

不难发现有

$$\pi_{|v_k}(m) = [-m_{k0}, m_{0k}]$$

Least Upper Bound. 当分支汇聚时，为了保证分析是上近似的，需要对不同分支对应的 Zone 求一个最小上界，即求一个最小的 Zone 使得其包裹现有的所有 Zone，DBM m 和 n 的最小上界可以简单定义为

$$(m \vee n)_{ij} \triangleq \max(m_{ij}, n_{ij})$$

Forget. 给定一个 DBM m 与一个变量 v_k ，丢弃 m 中所有有关 v_k 的信息，记作 $m_{\backslash v_k}$ ，你可以把它看作 $\pi_{|v_k}(m)$ 的逆运算，定义为

$$(m_{\backslash v_k})_{ij} \triangleq \begin{cases} \min(m_{ij}, m_{ik} + m_{kj}) & i \neq k \wedge j \neq k, \\ 0 & i = j = k, \\ +\infty & elsewhere. \end{cases}$$

Filter. 也称为 Guard，当遇到分支语句 g 时，需要计算一个给定的 DBM m 在添加了 g 这个约束后的新 DBM，记作 $m_{(g)}$。

如果 $g = (v_{j_0} - v_{i_0} \leq c)$ ，其中 $j_0 \neq i_0$ ，定义

$$(m_{(v_{j_0} - v_{i_0} \leq c)})_{ij} \triangleq \begin{cases} \min(m_{ij}, c) & i = i_0 \wedge j = j_0, \\ m_{ij} & elsewhere. \end{cases}$$

特殊地，我们的语言中不会出现以上比较 general 的形式。对于我们语言中会出现的 $g = (v_{j_0} \leq c)$ 可以视作 $i_0 = 0$；对于 $g = (v_{i_0} \geq c)$ 可以视作 $j_0 = 0$ 并转换成 $g = (-v_{i_0} \leq -c)$ 的形式。

对于 $g = (v_{j_0} = c)$ ，可以看作有两个约束 $g_1 = (v_{j_0} \leq c) \wedge g_2 = (-v_{j_0} \leq -c)$ ，所以可以定义

$$m_{(v_{j_0} - v_{i_0} \leq c)} \triangleq (m_{(v_{j_0} \leq c)})_{(-v_{j_0} \leq -c)}$$

对于 $g = (v_{j_0} < c)$ 可以转换为 $g = (v_{j_0} \leq c - 1)$，大于运算符同理。

Assignment. 当遇到赋值语句 $v_{i_0} \leftarrow e$ 时，需要计算一个给定的 DBM m 在赋值后生成的新 DBM，记作 $m_{(v_{i_0} \leftarrow e)}$。

当 $e = v_{i_0} + c$ 时，定义

$$(m_{(v_{i_0} \leftarrow v_{i_0} + c)})_{ij} \triangleq \begin{cases} m_{ij} - c & i = i_0 \wedge j \neq i_0, \\ m_{ij} + c & i \neq i_0 \wedge j = i_0, \\ m_{ij} & elsewhere. \end{cases}$$

当 $e = v_{j_0} + c(i_0 \neq j_0)$ 时，可以使用 Forget 运算与 Filter 运算定义

$$(m_{(v_{i_0} \leftarrow v_{j_0} + c)}) \triangleq ((m_{\backslash v_{i_0}})_{(v_{i_0} - v_{j_0} \leq c)})_{(v_{j_0} - v_{i_0} \leq -c)}$$

特殊的，当 $e = c$ 时也可以看作 $j_0 = 0$ 统一处理。

对于其他情况比如 $e = v_{j_0} + v_{k_0}$ 等两个变量加的，或是 $e = c - v_{j_0}$ 这种减去变量的，为了简便可以直接使用 Projection 运算与区间抽象域算出 $e \in [e^-, e^+]$ ，然后定义

$$(m_{(v_{i_0} \leftarrow e)})_{ij} \triangleq \begin{cases} e^+ & i = 0 \wedge j = i_0, \\ -e^- & j = 0 \wedge i = i_0, \\ (m_{\backslash v_{i_0}})_{ij} & elsewhere. \end{cases}$$

Implementation & Your Task

分析本体的实现在 `analysis/relationalNumericalAnalysis.h` 与 `analysis/relationalNumericalAnalysis.cpp` 中，其实现与 Lab1 类似，不需要你自己实现。

Zone 抽象域的实现在 `analysis/zoneDomain.h` 与 `analysis/zoneDomain.cpp` 中，其中 **Equality and Inclusion Testing**（`leq` 与 `eq` 函数）、**Projection**（`projection` 函数）、**Least Upper Bound**（`lub` 函数）、**Forget**（`forget` 函数）、**Filter** 的部分（`filterInst` 函数）以及 **Assignment** 的部分（`assignInst`、`assign_case1` 函数）已经实现。

你的任务是仔细阅读上述操作并补全 `zoneDomain.cpp` 中的部分函数，源代码中用 `todo` 标注的地方：

- **Normalization**（`Normalization` 函数）
- **Emptiness Testing**（`isEmpty` 函数）
- **Filter**（`filter` 函数，仅需要处理形如 $x - y \le c$ 的条件，其中 x 或 y 必有一个为 0）
- **Assignment** 剩余两种情况的处理（`assign_case2`、`assign_case3` 函数，分别对应 $x \leftarrow y + c$ 与 $x \leftarrow [l, r]$）。

完成后你可以通过运行 `build/test/zoneAnalysisTest` 来检查正确率与召回率，使用方法同 Lab1。

也可以通过以下命令来单独分析某文件

```
fdlang -zone-analysis xxx.fdlang
```

比起 Lab1 新增了四个测试用例 `rel1.fdlang` ~ `rel4.fdlang`，可以通过对比 Lab1 的区间分析在这四个测试用例上的正确率来检验是否实现正确。

正确实现 Zone 抽象域**并不能**帮助你获得 100% 的正确率，你也不需要 100% 的正确率，只要能够正常跑通且正确率 $> 80\%$ 即可。

你需要提交：

- 源代码，仅包含 `analysis/zoneDomain.cpp` （你不应该也不需要修改其他文件）
- 简单的报告，包括方法、分析结果，可以对比 Lab1 的效果

Reference & Further Reading

本 Lab 不要求实现 Intersection、Widening 与 Narrowing，感兴趣的可以参考

[1] Weakly Relational Numerical Abstract Domains - PhD. Defense - Antoine Miné

[2] A. Miné, “A New Numerical Abstract Domain Based on Difference-Bound Matrices,” in *Programs as Data Objects*, O. Danvy and A. Filinski, Eds., in PADO ’01. Berlin, Heidelberg: Springer, 2001, pp. 155–172. doi: 10.1007/3-540-44978-7_10.