基于抽象解释的变量值范围分析及应用

王雅文[1],宫云战[1],肖 庆[1,2],杨朝红[2]
(1.北京邮电大学网络与交换技术国家重点实验室,北京 100876; 2.装甲兵工程学院信息工程系,北京 100072)

摘 要: 精确的变量值范围分析对于编译器优化、静态分析和软件测试至关重要.在介绍抽象解释理论的基础上,扩展了经典的区间抽象,首次提出区间集的概念并定义了新的数值型区间集代数、布尔型和引用型区间代数,给出了统一的基于抽象解释的变量值范围分析方法 RABAI,引入拓宽算子计算循环体变量范围,对过程参数定义了特殊的未定义取值(*undefined*),使用函数摘要来计算过程调用对程序上下文状态的影响.该方法能有效压缩变量取值空间,检测出程序中的矛盾语句节点和不可达路径,实验表明基于 RABAI 方法的缺陷检测工具 DTS 能有效降低误报率.

关键词: 软件测试; 静态分析; 抽象解释; 区间抽象; 范围分析; 不可达路径
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2011) 02-0296-08

A Method of Variable Range Analysis Based on Abstract Interpretation and Its Applications

WANG Ya-wen[1], GONG Yun-zhan[1], XIAO Qing[1,2], YANG Zhao-hong[2]
(1. *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876, *China*;
2. *Department of Information Engineering, Academy of Armored Force Engineering, Beijing* 100072, *China*)

Abstract: Variable range analysis is essential for compiler optimization, static analysis and software testing. This paper extends the classical interval abstraction, and defines the new numeric interval-set algebra, Boolean interval algebra and reference interval algebra. Then it presents a unified method of variable range analysis based on abstract interpretation (RABAI). This method uses widening operator to compute variable range in loop, undefined value to describe procedure parameters, and applies function summary as a stand-in for the function. RABAI compresses variable value ranges, and can detect infeasible paths in program. Finally RABAI is applied in Java code analysis tool DTS, and DTS can reduce the false positive rate of bug checking result.

Key words: software testing; static analysis; abstract interpretation; interval abstraction; range analysis; infeasible path

1 引言

在软件静态分析中,变量取值范围的精确度和计算效率对缺陷的检测结果有很大的影响,值范围越接近实际运行时的情况,检测结果的准确率越高.精确地获取变量的值范围往往是不可能的[1],必须在精度和速度之间进行折中.

抽象解释是一种针对计算机系统语义模型的近似理论,可以有效地用于静态分析程序变量的取值范围.基于抽象解释的值范围分析方法把范围传播和范围分析结合起来,因此有助于提高变量值范围的精度.Cousot[2]和 Nielson[1]等人完整地叙述了抽象解释的理论框架,部分涉及到基于抽象解释的值范围分析;文献[3]综述了基于抽象解释的 3 种典型应用:程序变换、程序安全性验证和活性性质验证;文献[4]提出了基于抽象解释和通用单调数据流框架的值范围分析框架,包括精确的定义、分析和完整的正确性证明.已有基于抽象解释的值范围分析方法[1~5]一般仅针对数值型变量,值范围采用单一区间进行描述,并且存在条件判断分支中变量值区间计算效率比较低等问题.区间抽象域(Interval Abstraction Domain)[6,7]是抽象解释理论中一种典型的、比较精确的抽象域.区间抽象将程序中的数值型变量值范围抽象表示为区间,定义了区间上的拓宽(widening)算子和收窄(narrowing)算子,并给出程序不同类型语句节点处的值范围区间计算方法.本文对经典区间抽象进行了扩展,将经典的数值区间代数扩展为数值型区间集代数,定义了布尔型和引用型变量的区间代数,在此基础上给出了一个统一的变量值范围分析方法,最后对值范围分析的若干典型应用进行了研究.

收稿日期:2009-11-22;修回日期:2010-05-28
基金项目:国家 863 高技术研究发展计划重点项目(No.2007AA010302);国家 863 高技术研究发展计划(No.2009AA012404)

2 抽象解释理论基础

抽象解释[5]是由 P Cousot 和 R Cousot 在 1977 年提出的一种针对计算机系统语义模型的近似理论.抽象解释的主要思想是对给定程序设计语言赋予具体和抽象两种语义,然后建立二者之间的正确性关系,通过对抽象语义的求解来达到保守地计算具体语义的目的,使得程序抽象执行的结果能够反映出程序真实运行的部分信息.

在计算机科学中,抽象解释是基于有序集合特别是格上的单调函数,是计算机程序语义的可靠逼近理论,其本质是在计算效率和计算精度之间取得均衡,以损失计算精度求得计算可行性,再通过迭代计算增强计算精度的一种抽象逼近方法.抽象解释在软件领域的主要具体应用是形式静态分析以及程序可能执行信息的自动提取.在满足一定约束的情况下,抽象解释能够保证分析的安全性、正确性以及可计算性.对象域抽象、伽罗瓦连接(Galois connection)以及完备格上的拓宽(widening)算子是抽象解释理论中的基本概念[3].

抽象域是程序具体语义的一种抽象,由具体语义中的各计算状态组成,包括抽象属性和抽象操作.对语义进行抽象的侧重点不同,产生的抽象域也不同,比如数值抽象、关联/非关联抽象以及符号抽象等[7].抽象域是一个完备格,具有有穷的元素或是不存在无穷递增链,从而保证抽象解释的终止.

3 基本数据类型区间代数

对程序中的变量取值范围进行区间抽象,得到的抽象域是区间抽象域(Interval Domain),也称区间代数(Interval algebra).区间抽象是一种非关联的抽象,即认为程序变量是互相独立的,忽略变量之间存在的关系.本文将区间代数的变量类型从已有的整数型扩展到数值型、引用型以及布尔型.

3.1 数值型区间代数

文献[8]详细地介绍了区间、区间基本运算以及区间函数等理论.在已有的研究文献[1~5]中,数值型变量取值范围往往采用单个区间进行描述,这种方法的计算简单,但是精度不够高.

数值型变量的单区间表示对于实际程序来说,很多情况下描述能力有限.例如,对于 $x!=3$,采用单一区间表示应该为[MIN,MAX],它虽然是变量实际取值的保守表示,但已经丢失了该表达式实际语义的关键信息.再如,整数类型变量 i 的取值是[2,4]或[9,10],用区间表示应该为[2,10],而这样会包含冗余取值区间[5,8].为了更加准确地描述一个变量的取值范围,我们引入区间集的概念.

区间集是由若干两两互不相交的区间构成的集合,区间集 IS 的表示形式如下:

$$\begin{aligned} IS &= \{X_1, X_2, \cdots, X_n\} \\ &= \{[\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}], \cdots, [\underline{x_n}, \overline{x_n}]\}, \\ & X_1 < X_2 < \cdots < X_n. \end{aligned}$$

上述变量 i 的取值范围用区间集可表示为{[2,4],[9,10]}.

定义 1　数值型区间集代数系统为$\langle L_N, \subseteq, \top_N, \bot_N, X_N\rangle$,其中 L_N 是区间集的全集,$\top_N$是最大值{[$-\infty$,$+\infty$]};$\bot_N$ 区间集的最小值,表示区间集为空;X_N 表示未定义取值(undefined)[9,10].

对于 $\forall\, l \in L_N$, $X_N \cup l = X_N \cap l = l$; $X_N \oplus l = l \oplus X_N = X_N$,其中,$\oplus \in \{+, -, *, \div\}$.

X_N 一般用于过程参数及结构体的成员变量值范围分析中.对于此类变量,在只有其类型而没有任何值范围信息的情况下,如简单处理取默认的最大值,那么在缺陷检测的时候会带来大量的误报.采用该特殊取值,可有效避免此类误报.类似的,下文采用 X_B 和 X_R 分别表示布尔型和引用型的未定义(Undefined)取值.

(1)区间集的集合运算

首先定义区间集与区间的集合运算.

设区间集

$$\begin{aligned} IS &= \{X_1, \cdots, X_j, \cdots, X_k, \cdots, X_n\} \\ &= \{[\underline{x_1}, \overline{x_1}], \cdots, [\underline{x_j}, \overline{x_j}], \cdots, [\underline{x_k}, \overline{x_k}], \cdots, [\underline{x_n}, \overline{x_n}]\} \end{aligned}$$

区间 $M=[\underline{m}, \overline{m}]$.假设 M 与 IS 中的相邻区间 $X_j, \cdots, X_k$ ($j \leqslant k$)相交,则 IS 与 M 的"并"运算定义为:

$$\begin{aligned} IS \cup M = \{&[\underline{x_1}, \overline{x_1}], \cdots, [\min(\underline{x_j}, \underline{m}), \max(\overline{x_k}, \overline{m})], \\ &\cdots, [\underline{x_n}, \overline{x_n}]\} \end{aligned}$$

IS 与 M 的"交"运算定义为:

$$IS \cap M = \{[\max(\underline{x_j}, \underline{m}), \overline{x_j}], \cdots, [\underline{x_k}, \min(\overline{x_k}, \overline{m})]\}$$

如果 M 不与 IS 中的任一区间有交集,则:

$$IS \cap M = \varnothing,$$

$$IS \cup M = \{X_1, \cdots, X_j, M, X_{j+1}, \cdots, X_n\}$$

(假设 $X_j < M < X_{j+1}$).

下面定义区间集与区间集的集合运算.

设区间集

$$IS_1 = \{X_1, X_2, \cdots, X_m\}, IS_2 = \{Y_1, Y_2, \cdots, Y_n\},$$

则 IS_1 与 IS_2 的"并"运算定义为:

$$IS_1 \cup IS_2 = (\cdots((IS_1 \cup Y_1) \cup Y_2) \cup \cdots) \cup Y_n;$$

IS_1 与 IS_2 的"交"运算定义为:

$$IS_1 \cap IS_2 = \bigcup_{i=1}^{n} (IS_1 \cap Y_i).$$

(2)区间集的四则运算

设区间集

$$\begin{aligned} IS_1 &= \{X_1, X_2, \cdots, X_m\} \\ &= \{[\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}], \cdots, [\underline{x_m}, \overline{x_m}]\} \end{aligned}$$

区间集

$$IS_2 = \{Y_1, Y_2, \cdots, Y_n\} = \{[\underline{y_1}, \overline{y_1}], [\underline{y_2}, \overline{y_2}], \cdots, [\underline{y_n}, \overline{y_n}]\}$$

设运算符 $\oplus \in \{+, -, *, \div\}$, $IS_1 \oplus IS_2$ 的计算结果 IS_0 可由下面的步骤计算得到:

$IS_0 = \varnothing$;

for int $i = 1$ to m {

for int $j = 1$ to n {

$IS_0 = IS_0 \cup (X_i \oplus Y_j)$;

}

}

(3)区间集的拓宽运算(∇)

设$\perp$为区间的最小值,对于任意的区间 i, $\perp \nabla i = i \nabla \perp = i$;

如果$m_2 < m_1$,并且 $n_2 \leqslant n_1$,则

$[m_1, n_1] \nabla [m_2, n_2] = [-\infty, n_1]$;

如果$m_1 \leqslant m_2$,并且 $n_1 < n_2$,则

$[m_1, n_1] \nabla [m_2, n_2] = [m_1, +\infty]$;

如果$m_2 < m_1$,并且 $n_1 < n_2$,则

$[m_1, n_1] \nabla [m_2, n_2] = [-\infty, +\infty]$;

如果$m_1 \leqslant m_2$,并且 $n_2 \leqslant n_1$,则

$[m_1, n_1] \nabla [m_2, n_2] = [m_1, n_1]$;

对于区间构成的无穷递增链$[0,1], [0,2], \cdots, [0, n], \cdots$,拓宽算子使该递增链迅速收敛到$[0, +\infty]$.

对于区间集 IS_1 和 IS_2,定义 $IS_1 \nabla IS_2 = \{[\min(IS_1), \max(IS_1)] \nabla [\min(IS_2), \max(IS_2)]\}$.

在不产生混淆的情况下,也称数值变量的区间集为区间.

(4)区间集的逻辑运算

程序中的条件表达式分为简单条件表达式和复合条件表达式,简单条件表达式包括由 >、!=、≤等关系运算符构成的表达式以及由单个布尔变量或常量构成的表达式.复合条件表达式是由简单条件表达式经过布尔逻辑操作符"与"、"或"、"非"、"异或"等连接构成的.

①对于简单条件表达式,仅考虑系数为常数的线性不等式(统计表明,Java 开源软件中非线性表达式的比例不足 1‰),例如: $i > j + 2$, $x + 2y > z - 1$.定义函数 *interval*(C, v),指当条件表达式 C 取值为真时,变量 v 的取值区间集.通过提取表达式中变量的系数及分离目标变量,可计算得到目标变量的取值区间集.

对于 C 中没有出现的无关变量 v', *interval*(C, v') 为 v' 的当前取值范围.

②对于仅由逻辑运算符"与"、"或"连接的复合条件表达式,假设表达式中的不同变量相互独立,定义函数 *interval* 如下:

$$interval(A \,||\, B, v) = interval(A, v) \cup interval(B, v);$$

$$interval(A \&\& B, v) = interval(A, v) \cap interval(B, v).$$

对含有"非"、"异或"操作的表达式,根据德摩根定律,可以将其变换为上述仅由"与"和"或"运算符连接的等价表达式后进行计算.但是该方法需要多次变换抽象语法树,计算效率较低.针对这一问题,文献[11]提出了变量取值的可能集和必然集来统一计算条件表达式中的变量值范围.

3.2 非数值型区间代数

引用类型的变量 r 是指向内存中对象的标识,静态分析中关注的是 r 是否会为空. r 若为空,对 r 的引用会导致运行时空指针引用的故障.

定义 2 引用型区间代数系统为$\langle L_R, \subseteq, \top_R, \perp_R, X_R\rangle$,其中 L_R 表示引用型区间值的全集$\{\perp_R$, NULL, NOTNULL, $\top_R$, $X_R\}$; NULL $\cap$ NOTNULL = $\perp_R$, NULL $\cup$ NOTNULL = $\top_R$; X_R 表示布尔类型的未定义取值(undefined).

$\forall r \in L_R, X_R \cup r = X_R \cap r = r$;

$\forall r_1, r_2 \in L_R, r_1 \nabla r_2 = r_2 \subseteq r_1 ?\ r_1 : \top_R$.

定义 3 布尔型区间代数系统为$\langle L_B, \subseteq, \top_B, \perp_B, X_B\rangle$,其中 L_B 代表布尔型区间值的全集$\{\perp_B$, TRUE, FALSE, $\top_B$, $X_B\}$; TRUE $\cap$ FALSE = $\perp_B$, TRUE $\cup$ FALSE = $\top_B$; X_B 表示引用类型的未定义取值(undefined).

$\forall b \in L_B, X_B \cup b = X_B \cap b = b$;

$\forall b_1, b_2 \in L_B, b_1 \nabla b_2 = b_2 \subseteq b_1 ?\ b_1 : \top_B$.

4 程序变量值范围分析

4.1 程序的控制流描述

变量的值范围分析通常通过遍历程序的控制流图得到.控制流图是反映过程控制结构的有向图,通常可表示为$\langle N, E, entry, exit\rangle$.其中 N 是语句节点的集合, E 是有向边的集合, *entry* 为过程的唯一入口节点, *exit* 为过程的唯一出口节点.

具体地,定义 $N = \{entry, exit\} \cup Declarations \cup Assignments \cup Tests \cup Junctions \cup Calls$.其中 *Assignments* 为赋值语句节点集合, *Declarations* 为声明语句(未初始化)节点集合, *Tests* 为条件判断节点集合, *Junctions* 为分支汇合点集合, *Calls* 为函数调用点集合.

汇合点可分为选择语句的简单汇合点和循环语句汇合点,即

$$Junctions = SimpleJunctions \cup LoopJunctions$$

定义 4 节点 n 的入边集为 $preEdges(n): N \to 2^E$, 出边集为 $succEdges(n): N \to 2^E$;

定义 5 有向边 e 的起始节点为 $origin(e)$,终止节

点为 $end(e)$;

定义 6　对于条件判断节点 $t \in Tests$,当条件表达式取值为“真”和“假”时,对应不同的后继边,分别为 $succEdge\_T(t)$ 和 $succEdge\_F(t)$,即 $succEdges(t) = \{succEdge\_T(t), succEdge\_F(t)\}$.

定义 7　边的上下文:过程的变量值范围状态由各条边上的上下文(*Context*)元组组成,上下文关系为 $C \in Variables \times Intervals$,由各个变量的名值对组成,例如,$C_1 = \{(i, \{[-1,3],[6,10]\}),(b, \text{TRUE})\}$.

定义 8　在上下文 C 中,变量 i 的取值区间为

$$C(i) = \begin{cases} v, & \text{if } (\exists v! = \bot) \text{且} (i,v) \in C \\ \bot, & \text{else} \end{cases}$$

对于上述的 C_1,有 $C_1(i) = \{[-1,3],[6,10]\}$, $C_1(j) = \bot$.

定义 9　对于上下文 C 和 C',当且仅当对于 $\forall v \in$ Variables, $C(v) \subseteq C'(v)$时,称 $C \subseteq C'$.

4.2　函数摘要

完整的变量值范围分析包括过程内分析和过程间分析两方面,创建函数摘要是一种灵活的过程间分析方法[12].首先由过程内分析的结果得到一个函数摘要,当分析到该函数的调用时,就将该函数摘要作为函数调用的替代进行使用.

定义 10　函数 f 的函数摘要是一个二元组,记为 $summary(f) = \langle constraints, postCond \rangle$.

constraints 是调用函数 f 时所必须满足的约束信息,它是一个集合,可表示为 $constraints = \{\langle value(v_1), C_1\rangle, \langle value(v_2), C_2\rangle, \cdots, \langle value(v_n), C_n\rangle\}$ $(v_i \in Var(f), i = 1,2,\cdots,n)$,其中 v_i 是与上下文环境相关的变量,如全局变量、类成员变量和参数,$value(v_i)$表示当条件 C_i 为 true 时,v_i 的取值范围,用扩展的区间值进行描述.

postCond 是函数 f 的后置信息,即 f 执行后对上下文的影响. $postCond = \{value(v_1), value(v_2), \cdots, value(v_n)\}$ $(v_i \in Var(f), i = 1,2,\cdots,n)$,其中 v_i 是函数 f 中能够对调用点上下文环境产生影响的因素,包括全局变量、类成员及返回值等.

算法 1 描述了通过一个工作队列来辅助进行函数摘要生成的迭代过程,算法在进行过程内分析的同时生成函数摘要.如果某个函数的摘要进行了更新,那么这个函数的所有调用点都应该进入队列重新分析.算法的终止条件是各个函数摘要都稳定下来不再发生变化,由于所考虑的函数摘要都是针对函数、变量、变量区间取值的有限元组,每次迭代时函数摘要中的元组个数只能增加,因此算法最终会收敛.在实际实现时,可以先依据函数调用关系进行拓扑排序调整各函数的入队顺序,从而加快算法收敛的速度.

算法 1　函数摘要生成算法

```
procedure globalAnalysis(callGraph, summaries) {
1       for each f in callGraph
2           add f to workList;
3       while! (workList.isEmpty()) {
4           f = workList.delete();
5           old = summaries.getSummary(f);
6           localRangeAnalysis(CFG(f));
7           new = generate new summary for f;
8           if (old! = new) {
9               for each function g in callGraph that calls f {
10                  if (g is not in workList)
11                      workList.add(g);
                }
            }
        }
}
```

4.3　基于抽象解释的值范围分析 RABAI

设过程 p 对应的控制流图为 *cfg*,其输入参数集合为 P,则 p 中各变量的取值范围分析过程如算法 2 所示(当集合 A 包含单一元素 a 时,假定 A 与 a 等价).

算法 2　RABAI 算法

```
1   in:过程 p 的控制流图 cfg
2   out:包含变量取值范围信息的控制流图
procedure rangeAnalysis(Graph cfg) {
3   for each e in E
4     localContext(e) = ∅;
5   junctions = ∅; //汇合点集
6   edges = succEdge(entry); //待计算的边集
7   for each p in P
8     localContext(edges) = localContext(edges) ∪ {(p, X)};
9   while (edges! = ∅) {
10    while (edges ! = ∅) {
11      Edge inEdge = edges.delete();
12      Context C = C' = localContext(inEdge);
13      Node n = end(inEdge);
14      if (n ∈ Junctions) junctions = junctions ∪ {n};
        //收集汇合点
15      if (n ∈ Calls) 根据函数摘要信息更新当前上下文状态;
16      case n in {
17        Declarations:
18          for each var in n
19            updateRange(succEdge(n), C ∪ {(var, ⊤)});
20        Assignment:
21          Variable v = id(n);// id(n):左端被赋值变量;
            expr(n):右端赋值表达式
22          updateRange(succEdge(n), (C - {(v, C(v))}) ∪ (v,
            interval(expr(n)));
23        Tests:
24          C = C' = localContext(inEdge);
25          for each var in n {
26            C = (C - {var, C(var)}) ∪ (v, C(var) ∩ interval
              (Bexpr(n), var));// Bexpr(n): n 包含的条件表
              达式
27            C' = (C' - {var, C'(var)}) ∪ (v, C(var) ∩ inter-
val(! Bexpr(n), var));
            }
29          updateRange(succEdge_T(n), C);//真分支
30          updateRange(succEdge_F(n), C');//假分支
31      }
32    }
```

```
33  for each n in junctions {
34    for each var in preEdge(n)
35      outContext(var) = ⋃ e∈preEdge(n) localContext(e, v);
36    if (outContext ! = localContext(succEdge(n))) {
37      case n in
38        SimpleJunctions:
39          updateRange(succEdge (n), outContext);
40        LoopJunctions:
41          for each var in n {
42            updateRange(succEdge(n), (localContext(succEdge
(n)) ▽ outContext(var));
43          }
44    }
45    junctions = ∅;
      }
    }
    procedure updateRange(Edge e, Context c) {
46    if (c ! = localContext(e) {
47      localContext(e) = c ⋃ localContext(e);
48      edges = edges ⋃ e;
      }
    }
```

算法 2 从控制流图中头节点的出边集开始,首先为各条边添加过程参数的取值信息,初始值均为"未定义"取值(用 X 表示);接着循环处理待测边集合 *edges* 中的边,根据当前边的终止节点 n 的不同语句类型进行相应的计算:如果是汇合点,将当前节点加入汇合点集合 *junctions* 中;如果是过程调用语句,则根据函数摘要更新当前的上下文状态;如果是声明语句,则设置声明变量的值为该类型的默认最大值;如果是赋值语句,将左端被赋值变量的值更新为右端表达式的值;如果是条件语句,分别计算条件判断表达式为"真"和"假"时各变量的取值情况,然后更新对应的"真"分支和"假"分支上的变量值.如此循环,直至待测节点集合为空,对应算法的 10~32 行.对于汇合点集合中的元素:如果是条件分支的汇合点,则将其各入边上的上下文进行合并;如果是循环语句的汇合点,通过拓宽算子更新出边上的上下文.在处理汇合点的过程中,有些边的上下文会发生变化,这些边重新加入队列 *edges* 中进行处理.如此循环,直到控制流图中所有边上的上下文不再发生变化,算法终止退出.

由上述算法得到的控制流图中各边的上下文状态即各个变量的取值区间范围.根据过程 rangeAnalysis()的计算结果,如果新的控制流图中某条边上存在变量取值为最小值⊥的情况,则该语句节点不可达,为矛盾节点.

算法的终止性说明:控制流图中每个循环都有一个循环汇合点,由抽象解释理论中拓宽算子的定义可知,控制流图中各循环汇合节点的出边上的上下文序列构成一个严格的上升链,该链必定是收敛的,即算法是可终止的.

对于图 1 中的过程 f(int i)及其控制流图,RABAI 算法对 f 的值范围分析过程如表 1 所示,localContext 列中粗体标记出来的上下文是最终的状态,即变量的稳定取值范围.

```
void f(int i) {
  int j = 0;
  if (i > 0) {
    while (j < 100)
      j + = 5;
  } else
    j - - ;
}
```

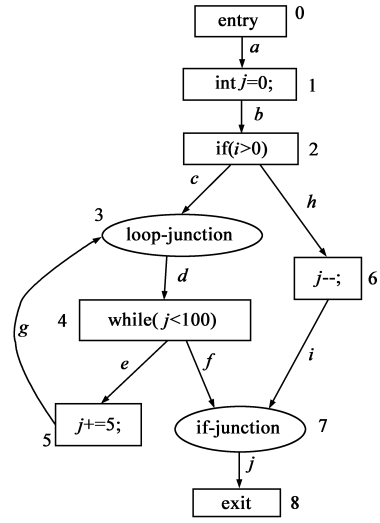


图1 过程 f 及其控制流图

表 1 过程 f 的值范围分析过程

初始化:P = {i}; *entry* = #0; *exit* = #7; *Tests* = {#2, #4}; *Assignments* = {#1, #5, #6}; *Declarations* = ∅; *SimpleJunctions* = {#7}; *LoopJunctions* = {#3}.

step	*edges*	*inEdge*	*n*	*localContext*	*junctions*
1	{#*a*}	#*a*	**#1**	**#*b***:{(***i***, X_N), (***j***, {[**0**,**0**]})}	∅
2	{#*b*}	#*b*	#2	**#*c***:{(***i***, {[**1**, +∞]}), (***j***, {[**0**,**0**]})} **#*h***:{(***i***, {[−∞, **0**]}), (***j***, {[**0**,**0**]})}	∅
3	{#c, #h}	#c	#3		{#3}
4	{#h}	#h	#6	**#*i***:{(***i***, {[−∞, **0**]}), (***j***, {[**−1**, **−1**]})}	{#3}
5	{#*i*}	#*i*			{#3, #7}
6	∅		#3	#*d*:{(*i*, {[1, +∞]}), (*j*, {[0,0]})}	{#3, #7}
7	{#*d*}		#7	#*j*:{(*i*, {[−∞, 0]}), (*j*, {[−1, −1]})}	∅
8	{#*d*, #*j*}	#*d*	#4	#*e*:{(*i*, {[1, +∞]}), (*j*, {[0,0]})} #*f*:{(*i*, {[1, +∞]}), (*j*, ⊥$_N$)}	∅ ∅
9	{#*j*, #*e*, #*f*}	#*j*			∅
10	{#*e*, #*f*}	#*e*	#5	#*g*:{(*i*, {[1, +∞]}), (*j*, {[5,5]})}	∅
11	{#*f*, #*g*}	#*f*	#7		{#7}
12	{#*g*}	#*g*	#3		{#3, #7}
13	∅		**#3**	**#*d***:{(***i***, {[**1**, +∞]}), (***j***, {[**0**, +∞]})}	{#3, #7}
14	{#*d*}		#7		∅
15	{#*d*}	**#*d***	**#4**	**#*e***:{(***i***, {[**1**, +∞]}), (***j***, {[**0**,**99**]})} **#*f***:{(***i***, {[**1**, +∞]}), (***j***, {[**100**, +∞]})}	∅
16	{**#*e*, #*f***}	**#*e***	**#5**	**#*g***:{(***i***, {[**1**, +∞]}), (***j***, {[**5**,**104**]})}	∅
17	{#*g*, #*f*}	#*g*	#3		{#3}
18	{#*f*}	#*f*	#7		{#3, #7}
19	∅		#3		{#3, #7}
20	∅		#7	**#*j***:{(***i***, {[**1**,+∞]}), (***j***, {[**−1**,**−1**], [**100**,+∞]})}	∅

5 RABAI 在静态分析中的应用

5.1 检测矛盾语句和不可达路径

由 RABAI 方法得到的控制流图上,每条边的上下文状态 localContext 中包括了各变量当前的取值范围信息.其中,localContext 中包含变量取值为⊥的节点是矛盾节点,即实际程序中的不可达语句.

以数值型变量为例,我们将程序 Test1.java 作为输入,运用 RABAI 方法进行分析,添加了区间值信息的控制流图如图 2 所示(节点中未出现的变量取值为默认区间).由 RABAI 的计算结果可知,图 2 中灰色填充的节点 stmt_3,对应源程序中的第 5 行,变量 i 的取值为空,故为矛盾节点.

源代码　Test1.java

```
1:    public class Test1 {
2:        void f(int i,int j){
3:          if (i≥3){
4:            if (i<0 && j< -5)
5:                i+ =6; //矛盾语句
6:            else
7:                j- =15;
8:          }
9:          else
10:             i- -;
11:         }
12:   }
```

作者对开源软件联盟 sourceforge* 上排名比较靠前的 10 个大型 Java 软件 进行了值范围分析.实验分别对仅数值型变量(其它类型变量取默认值,下同)、仅布尔型变量、仅引用型变量以及所有这 3 种类型变量进行了范围分析,结果报告出软件中的矛盾语句节点数如表 2 所示.可见不同类型变量的值范围分析均会检测到一定数量的矛盾节点,当所有类型变量同时进行值范围分析时检测出矛盾语句节点最多.

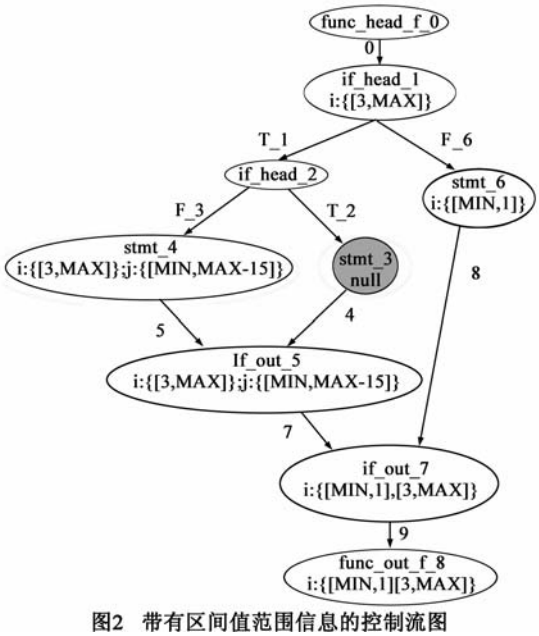


图2　带有区间值范围信息的控制流图

表 2　不同类型区间运算检测出的矛盾语句节点数

软件名称	文件数	代码行数	语句节点总数	不同变量类型的值范围分析			
				仅数值型	仅布尔型	仅引用型	全部
areca-7.1.1	426	68,090	40,320	56	56	297	635
aTunes-1.8.2	306	52,603	22,653	4	13	252	260
Azureus-3.0.5.2	2,720	575,220	206,330	2,903	1,237	3,644	7,059
cobra-0.98.1	449	70,062	33,504	345	99	1,074	1,464
freecol-0.7.3	343	110,822	45,728	536	100	1,338	1,937
freemind-0.8.1	509	102,112	51,526	40	26	170	236
jstock-1.0.4	165	38,139	16,988	78	2	210	309
megamek-0.32.2	535	212,453	118,183	1,449	242	2,213	3,647
robocode-1.6	233	53,408	24,396	128	128	298	432
SweetHome3D-1.8	154	59,943	24,086	53	21	200	281
合计	5,840	1,342,852	583,714	5,592	1,924	9,696	16,260

包含上述矛盾语句节点的所有路径均为不可达路径.此外,对单条路径上的变量值范围进行分析,如果某个节点存在变量取值为⊥的情况,则该路径不可达.

设函数 f 中的所有路径数目为 $Path_f$,图 3 给出表 2 所示 10 个大型软件中函数路径数目的分布情况,其中路径数目为 1 的函数占 60% 以上,路径数目超过 $2^{13}=8192$ 的函数所占比例不足 0.5%.

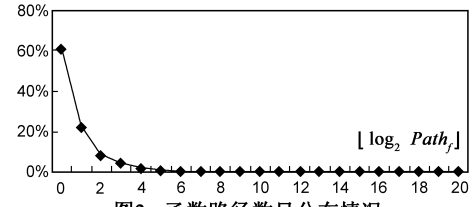


图3　函数路径数目分布情况

不可达路径的信息可以改善静态分析的准确性.程序路径中若存在变量值范围为⊥的情况则表示当前路径不可达.作者对表 2 中的 Java 程序进行了值范围分析以检测其中的不可达路径.实验分别对数值型变量、布尔型变量、引用型变量以及全部类型变量进行了范围分析,结果报告出软件中的不可达路径比例如图 4 所示.就所分析的变量类型而言,当分析程序中所有类型(数值型、引用型及布尔型)的变量值范围时,检测到的不可达路径数目最多;同时,随着函数的路径数目增多,不可达路径所占比例也相应变大,其主要原因在于结构复杂的函数包含更多的条件判断语句,不同条件分支中的变量值范围存在更多的矛盾组合情况.

* http://sourceforge.net/

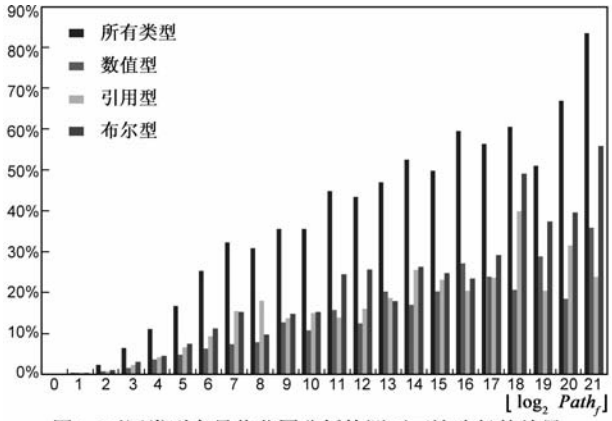


图4 不同类型变量值范围分析检测不可达路径的结果

5.2 提高代码缺陷检测的效率

RABAI 算法"压缩"了变量取值的范围空间,能有效检测出不可达路径,基于此的缺陷检测工具能对程序控制流路径进行有效的"剪枝",提高了检测效率,在一定程度上减少了误报.同时,精确的变量值范围分析有助于提高缺陷检测的准确率.

测试工具 DTS 是作者所在实验室研发的一款 Java 及 C/C++源码缺陷检测工具,该工具以 RABAI 算法为基础、为各种软件缺陷模式创建检测状态机来检测程序中各类潜在的故障、漏洞和缺陷.

下面以 Java 空指针引用(Null Pointer Dereference, NPD)为例说明不同类型的变量值范围分析对软件缺陷模式检测结果的影响情况.作者使用 DTS 对表 2 中的 Java 程序进行了 Java NPD 的缺陷检测,测试结果如表 3 所示.表中分别给出了经过人工确认后的 NPD 缺陷数目及下面两种情况下 DTS 输出的检查点数目:(1)仅对引用型变量进行值范围分析,而对数值型和布尔型变量采用默认取值范围;(2)对数值型、布尔型和引用型变量全部进行范围分析.可以看出,后者与前者相比,误报的总数由 444 减少至 381,减少了 63,减少比例约为 14.2%;平均误报率也由 34.1%降至 31.0%.

表 3 不同类型的值范围分析对 Java NPD 缺陷检测的影响

软件名称	NPD 数目	仅分析引用型所得检查点数目	分析所有类型所得检查点数目	误报减少数目
areca-7.1.1	68	123	120	3
aTunes-1.8.2	27	30	30	0
Azureus-3.0.5.2	241	420	400	20
cobra-0.98.1	10	19	17	2
freecol-0.7.3	265	294	287	7
freemind-0.8.1	117	159	155	4
jstock-1.0.4	8	12	11	1
megamek-0.32.2	176	274	250	24
robocode-1.6	49	55	53	2
SweetHome3D-1.8	5	24	24	0
合计	966	1410	1347	63

6 总结

软件的变量值范围分析是软件静态分析和动态测试的重要基础,提高值范围分析的精度和效率有着重要的意义.本文扩展了抽象解释中的区间抽象理论,分别定义了数值型、布尔型及引用型变量的表示,提出了一种统一的程序变量值范围分析计算方法.该方法对于矛盾语句和不可达路径的检测不存在误报,同时能提高程序缺陷检测的效率,降低分析结果的缺陷误报率.由于区间抽象是一种非关系型抽象,忽略了程序变量之间的关系,作者下一步的主要工作是基于关系型表示方法(如八边形抽象和凸多面体抽象等)进行变量的值范围分析,同时研究数组[13]和结构体等复杂数据类型的值范围表示及分析方法,进一步提高变量值范围的精确性.

参考文献:

[1] Nielson F, Nielson H R, Hankin C. Principles of Program Analysis[M]. Berlin: Springer-Verlag, 1999. 211 – 282.

[2] Cousot P, Cousot R. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints[A]. Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages[C]. New York, ACM Press, 1977. 238 – 252.

[3] 李梦君,李舟军,陈火旺. 基于抽象解释理论的程序验证技术[J]. 软件学报,2008,19(1):17 – 26.
Li Mengjun, Li Zhoujun, Chen Huowang. Program verification techniques based on the abstract interpretation theory[J]. Journal of Software, 2008, 19(1): 17 – 26. (in Chinese)

[4] 姬孟洛,王怀民,李梦君,等. 一种基于抽象解释和通用单调数据流框架的值范围分析方法[J]. 计算机研究与发展,2006,43(11):2020 – 2026.
Ji Mengluo, Wang Huaimin, Li Mengjun, et al. An value range analysis based on abstract interpretation and generalized monotone data flow framework[J]. Journal of Computer Research and Development, 2006, 43(11): 2020 – 2026. (in Chinese)

[5] 姬孟洛,李军,王馨,等. 一种基于抽象解释的 WCET 自动分析工具[J]. 计算机工程,2006,32(14):54 – 56.
Ji Mengluo, Li Jun, Wang Xin, et al. An automatic WCET analysis tool based on abstract interpretation[J]. Computer Engineering, 2006, 32(14): 54 – 56. (in Chinese)

[6] Cousot P, Cousot R. Static determination of dynamic properties of programs[A]. Proceedings of the 2nd International Symposium on Programming[C]. Dunod, Paris, 1976. 106 – 130.

[7] Cousot P. Abstract interpretation based formal methods and future challenges[A]. Informatics-10 Years Back, 10 Years Ahead[C]. London: Springer-Verlag, 2001. 138 – 156.

[8] 王德人,张连生,邓乃扬. 非线性方程的区间算法[M]. 上

海:上海科学技术出版社,1987.8 – 19.

[9] Hallem S, Chelf B, Xie Y, et al. A system and language for building system-specific, static analyses[A]. Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation[C]. New York: ACM Press, 2002. 69 – 82.

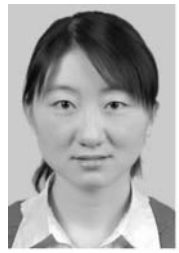
[10] Binkley David. Source Code Analysis: A Road Map[A]. 2007 Future of Software Engineering[C]. NW Washington: IEEE Computer Society, 2007. 104 – 119.

[11] Wang Yawen, Gong Yunzhan, Chen Junliang, et al. An application of interval analysis in software static analysis[A]. Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing[C]. NW Washington: IEEE Computer Society, 2008.12;367 – 372.

[12] Brian Chess, Jacob West. Secure Programming with Static Analysis[M]. Boston MA: Addison-Wesley Professional, 2007.91 – 93.

[13] 王雷,李吉,李博洋.缓冲区溢出漏洞精确检测方法研究[J].电子学报,2008,36(11):2200 – 2204.
Wang Lei, Li Ji, Li Boyang. Precisely detecting buffer overflow vulnerabilities[J]. Acta Electronica Sinica, 2008, 36(11):2200 – 2204.(in Chinese)

作者简介:



王雅文　女,1983 年生于陕西凤翔.北京邮电大学网络与交换技术国家重点实验室讲师.研究方向为软件测试、静态分析.
E-mail: wangyawen@bupt.edu.cn

宫云战　男,1962 年生于山东乳山,北京邮电大学网络与交换技术国家重点实验室教授,博士生导师.研究方向为软件测试、可信计算等.

肖　庆　男,1979 年生于湖南,北京邮电大学博士研究生,装甲兵工程学院信息工程系讲师.研究方向为软件测试、程序分析.

杨朝红　男,1976 年生于湖南,装甲兵工程学院信息工程系副教授,研究方向为软件测试.