doi:10.3772/j.issn.1002-0470.2019.10.003

基于抽象解释的单变量值范围分析①

李 静② 侯春燕③ 王劲松

(天津理工大学计算机科学与工程学院 天津 300384)

摘 要 针对传统基于抽象解释的变量值范围分析方法存在覆盖代码程度不高的问题,提出了一种新的描述单变量值区间方法。该方法采用找到数值型子程序的矛盾结点,全面考虑了不同的选择结构和循环结构的嵌套,并考虑了分析精度与效率的问题。以抽象解释理论作为分析框架,研究程序中各个变量的上下文状态,达到验证程序变量是否满足规范的目的。利用公开已有的变量值范围数据对变量值范围分析方法进行的验证表明,与现有的值范围分析方法相比,该方法由于更全面地考虑了不同的嵌套结构,因此取得了更好的结果。

关键词 抽象解释;区间抽象;单变量;上下文状态;静态分析

0 引 言

程序分析是对程序进行自动分析的过程,主要有动态分析与静态分析方法[1]。这两种方法都能检测软件代码,进而保证软件的整体质量。动态分析需要执行程序代码来获取程序运行时信息,但是考虑各种变量随时间而变化对程序的影响,所以难度大。但是静态分析不需要实际运行程序,只分析程序的语法语义,验证程序的正确性与可靠性[2]。在实际的程序分析过程中,静态分析利用形式化方法描述程序的行为,提供程序所有可能执行情况的信息。

其中使用的形式化方法包括模型检测、定理证明与抽象解释。模型检测是穷尽所有空间状态进行验证,存在状态空间爆炸问题。定理证明是分析逻辑理论,验证系统的安全性质。抽象解释可有效用于静态分析程序变量的取值范围。程序变量的数值性质在软件中占据重要位置,程序的数值计算须精确,即使是一个小数点的错误,都会造成不可估量的后果[3]。

1977 年,由 Cousot 等人[4]共同提出程序静态分析时构造与程序不动点语义的逼近理论,故抽象解释理论诞生。目前,国内抽象解释的研究多数集中在非凸抽象域,先后提出区间[5]、多边形[6]等抽象域来达到不同的精度。其中区间抽象域有计算效率高和简单易用的特点。相对于多面体抽象域[7],区间抽象域只能表示单个变量的性质,忽略变量之间的关系,所以分析精度与表达能力较差。在区间抽象上,文献[8]提出基于抽象解释的变量值范围分析方法,引入拓宽算子策略,并实现了 while 语句的应用,检验程序的值范围的正确性。此方法融合各种验证工具的近似理论,用来处理计算机科学中一些不可判定的问题[9]。虽然程序值范围分析已经有很多人研究,但是覆盖代码的程度不高,并且程序中各变量的缺陷也是导致错误频发的关键所在。

因此,本文选择基于抽象解释的值范围分析方法,分析程序中变量的上下文状态,检测出矛盾结点来保证程序的正确性。文章将研究重点放在基于 C 语言的简单子程序上,侧重判断程序中各个变量,注

① 天津市自然科学基金(18JCZDJC30700)和赛尔网络下一代互联网技术创新(NGII20160121)资助项目。
② 女,1994 年生,硕士生;研究方向:软件测试;E-mail: 2012036140@qq.com
③ 通信作者,E-mail: chunyanhou@tjut.edu.cn
(收稿日期:2018-12-20)

重考虑不同的选择结构和 while 循环结构的嵌套，通过找到不可达路径来验证程序的规范性。

1 理论模型

基于抽象解释理论的数值分析技术可得到程序中各种程序点变量的近似且可信的范围。为了提高变量值范围的精度，此技术结合了范围传播与范围分析[10]。王雅文等人[11]扩展经典区间抽象，将使用 if 嵌套结构实现在此分析方法中，有效地压缩变量取值空间并检测出程序中的不可达路径。

在已有基于抽象解释的值范围分析方法基础上，本文将变换不同的循环结构，对程序中变量的数值性质进行分析，验证程序中变量值范围是否满足需求。其方法中用到区间抽象的拓宽算子操作，从而分析各变量的上下文状态。算法中拓宽函数可由下面的式(1)得出[12]。

$$[c_1, d_1]\nabla[c_2, d_2] = \begin{cases} [-\infty, d_1] & \text{当 } c_2 < c_1, d_1 \geqslant d_2 \\ [c_1, +\infty] & \text{当 } c_2 \geqslant c_1, d_1 < d_2 \\ [-\infty, +\infty] & \text{当 } c_2 < c_1, d_1 < d_2 \\ [c_1, d_1] & \text{当 } c_2 \geqslant c_1, d_1 \geqslant d_2 \end{cases} \tag{1}$$

2 算法实现

2.1 算法流程

本文在抽象解释理论的基础上，给出了对程序变量的数值性质进行自动分析和验证的有效途径，主要工作如下：

(1) 构建程序的抽象环境，对源程序进行词法和语法分析，得到程序的结构信息，包括语句内容及语句之间的关系，即对程序的具体语义转化到抽象域中。首先，由源程序得到相对应的控制流程图。然后根据程序需求取得待分析的程序变量，简化控制流程图，只保留与待分析程序变量相关的结点。接着，由已知区间抽象域的规则，选取适合的数值抽象域。最后遍历程序控制流程图中有向边的集合，调用区间抽象域中一些域操作，得到每个程序点的抽象值存储到相应的控制流程图结点中。

(2) 给出在抽象环境中对程序变量的数值性质进行推导和验证的具体方法。首先，利用抽象解释理论的迭代策略对循环结构进行迭代计算，得到程序的不动点抽象值，然后将不动点抽象值转化为具体的约束关系，并写到对应的控制流图结点中，最后根据系统的需求对程序变量数值性质进行分析验证，保证了程序的安全性和可靠性[13]。

通常程序分析要求一定的准确性和可靠性，在数值抽象域的基础上，设计如下基于抽象区间域的程序分析软件，其流程如图 1 所示。

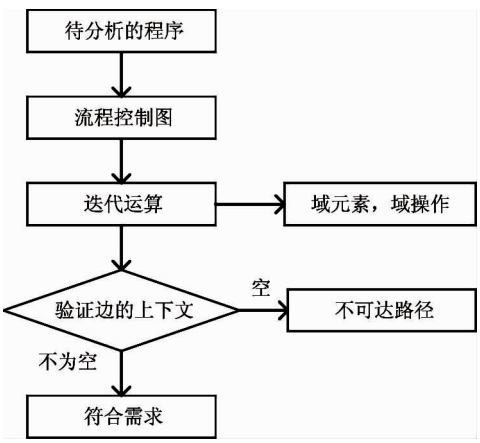

图 1 算法流程

2.2 程序的预处理阶段

以 C 语言的源程序作为输入，预处理阶段是将源程序的头部信息、注释与主函数等去掉，转化为统一格式的中间表示形式。在中间表示形式基础上，此阶段为子程序的每一条语句都标记序号。

控制流图是反映控制结构的有向图，源程序经过预处理阶段转为对应程序的控制流图。

首先将源程序作为文件流的形式输入到预处理程序中，将源程序头部信息截取，根据大括号的数量只取出主函数中间的内容，接着将此内容删除/ * 或 // 等注释信息，并整理格式。

此阶段将程序中的每一个语句作为一个结点，组成一个结点集合。输入语句即程序入口处的数据流信息，结束语句即程序出口处的数据流信息。其中结点结构包括结点类型、赋值语句内容、逻辑语句内容、结点序号、入边集合、出边集合。边结构包括前结点、后结点、边标记、边的上下文。

预处理程序使抽象语法树转化为控制流图,将复杂的程序结构分解成若干个简单的结构,其中每个分支结构的控制流图可按照其生成规则构建。

2.3　算法结构

算法的输入参数为流程控制图。如图 2 所示,由子程序的输入节点的出边集开始,首先为流程图中的边赋予初始值,初始值均设为空值。接着将结点入边放入待检测的栈中,同时循环遍历语句的结点集合。当结点集合不为空时,根据每一个结点的结构信息、结点的不同类型进行相应的计算。

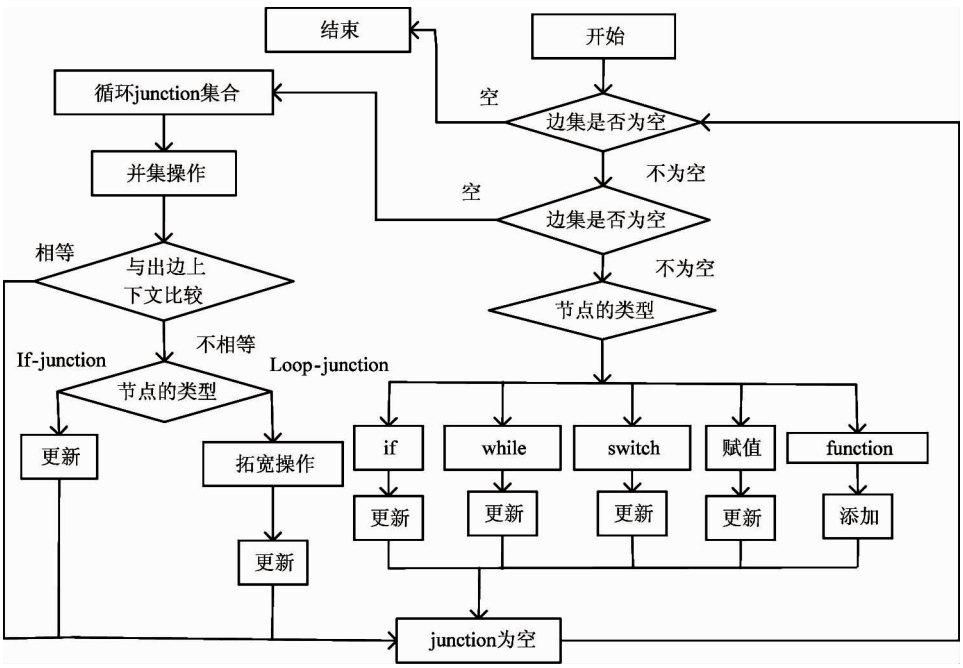

图 2　算法结构

(1) 结点类型为赋值结点,则根据求取变量名与求取表达式等函数取出结点语句中的变量和表达式的值,然后将变量的值更新为表达式的值。

(2) 结点类型为 if 判断,则根据语句内容的真假来得到变量的取值,再根据真假分支来更新对应的值。

(3) 结点类型为 while,则根据 while 中条件表达式来得到变量取值,同时也更新值。

(4) 结点类型为 switch,则根据 switch 语句中表达式以及 case 后的语句获取变量取值,遇到 break 则跳出 switch 结构。

(5) 若为 if-junction 和 loop-function 的两个结点则加入到汇合结点集合中。

(6) 若为 if-junction 结点,则计算真分支与假分支的上下文状态的并集,若得到的并集不等于结点的出边上下文,则更新值并添加此结点的出边。

(7) 若为 loop-function 结点,结点出边的上下文则是由结点的两个入边通过拓宽运算计算出。

算法中涉及到的函数:

(1) 并集操作:如图 3 所示,输入参数为两条入边的上下文状态,循环遍历两个上下文的变量,若变量相同,则其对应的值区间做并集运算。若不相同,则获取其值区间,最后合并变量与相应的值区间,也就是出边的上下文。

(2) if 判断函数:如图 4 所示,if 真分支与假分支函数根据输入的参数而定,函数的输入参数为结点和 true 或 false。当为 true 时,则代表是真分支,根据结点的结构中信息,得到此程序点的内容,再获取变量与值,接着根据 if 表达式的符号来确定真分支的上下文状态。

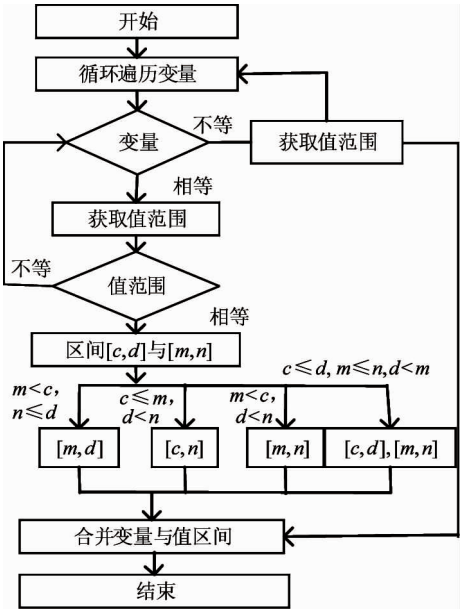

图 3　并集操作

（3）拓宽操作：如图 5 所示，整体结构与并集操作类似。在进行抽象解释分析时，由于区间运算的拓宽算子可保证分析过程的可终止性，算法选择程序循环的汇合结点为加宽结点，相当于 if 循环或 while 循环则在 if 语句的后面和 while 语句的前面分别加上。

（4）更新上下文：如图 6 所示，输入参数为边与对应边的上下文 $C2$，获取这条边的上下文 $C1$，再判断这条边的上下文与之前计算的上下文 $C2$ 是否相等。若相等则不操作，否则将两个上下文做并集操作。又因为每次计算汇合结点集合的过程中，边的上下文状态会一直改变，而这些被改变的边则会重新加入放入栈中进行计算。

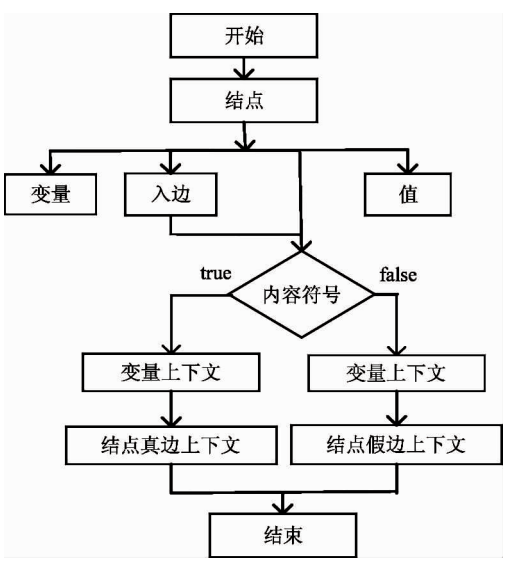

图 4　if 判断函数

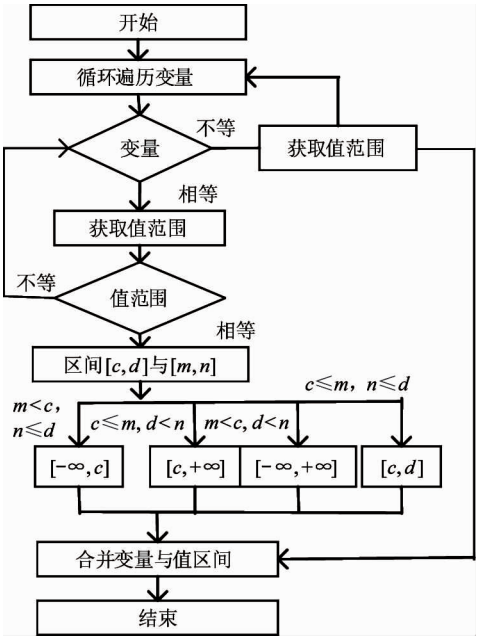

图 5　拓宽函数

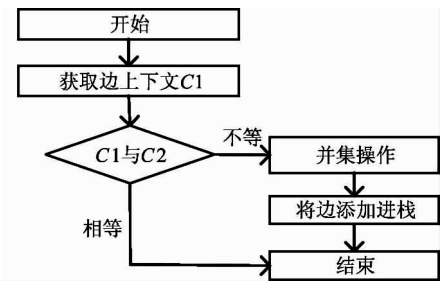

图 6　更新操作

3　验证变量值范围

源程序经过预处理阶段，转化为控制流程图，找到变量中的相关参数，再由控制流程图通过相应的迭代算法，计算出不动点的抽象值。当其中某一变量的抽象值为空时，则表示相应结点为矛盾结点，不满足约束关系。

从上面算法得到所有边的上下文状态即每条语句中相应变量的取值范围，若某条边的上下文状态为空值，则代表此结点为矛盾结点，这条边为不可达路径。

3.1　包含嵌套的 switch 语句

对于图 7 与图 8 中源程序及流程图,算法的每一步分析过程如表 1 所示,edges 栈表示待计算的边集合,inEdge 表示结点的输出边, n 表示结点,localContext 表示每经过结点的变量 i 与 j 形成变量的实际值范围区间,即边的上下文状态。junction 表示循环汇合结点集合。

```
void Fun (int i){
    int j=1;
    if(i>0){
        j++;
        switch (j>0){
            case true :
                j-=5;
                break;
            case flase :
                j+=5;
                break;
                }
        }
    else
        j--;
}
```

图 7　待分析的程序

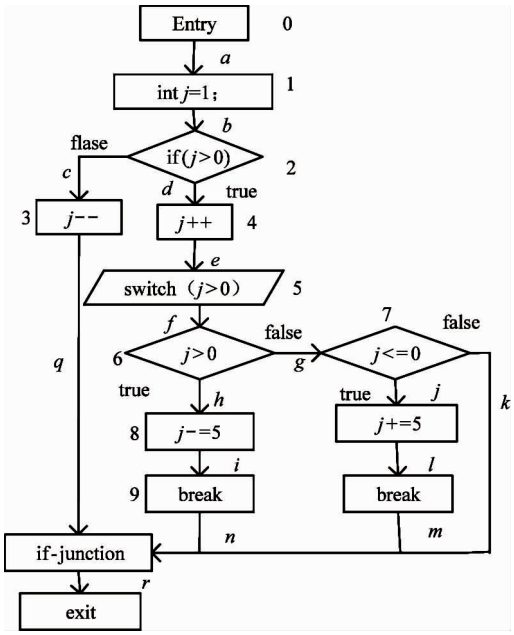

图 8　控制流程图

通过分析每一步边的上下文状态,判断此结点是否为矛盾结点,上面的子程序经过算法得到有一条不可达路径,即 g 边, g 边的 j 变量的状态为空。算法最终得到{(i,{[−∞ , + ∞ ,]}),(j,{[−3, −3],[0,0]})}说明对变量 i 和变量 j 的静态分析是准确的。

表 1　Fun 函数的值范围分析过程

初始化:参数为{i},entry = #0, exit = #9, Assign = {#1,#4,#6,#7}, while = {#3}, if = {#5}, junctions = {#2,#8}					
step	edges	inEdge	Node	localContext	junction
1	a	a	#1	b:{(i, X_N),(j,{[1,1]})}	Ø
2	b	b	#2	c:{(i,{[−∞ ,0]}),(j,{[1,1]})} d:{(i,{[1, + ∞]}),(j,{[1,1]})}	Ø
3	{d,c}	d	#4	e:{(i,{[1, + ∞]}),(j,{[2,2]})}	Ø
4	{c,e}	c	#3	q:{(i,{[−∞ ,0]}),(j,{[0,0]})}	Ø
5	{e,q}	e	#5	f:{(i,{[1, + ∞]}),(j,{[2,2]})}	Ø
6	{q,f}	q	#12		{#12}
7	f	f	#6	h:{(i,{[1, + ∞]}),(j,{[2,2]})} g:{(i,{[1, + ∞]}),(j, Ø)}	Ø
8	{h,g}	h	#8	i:{(i,{[1, + ∞]}),(j,{[−3, −3]})}	Ø
9	{g,i}	g	#7	k:{(i,{[1, + ∞]}),(j, Ø)} j:{(i,{[1, + ∞]}),(j, Ø)}	Ø
10	{i}	i	#9	n:{(i,{[1, + ∞]}),(j,{[−3, −3]})}	Ø
11	n	n	#12	r:{(i,{[−∞ , + ∞ ,]}),(j,{[−3, −3],[0,0]})}	Ø

此函数加入 switch 语句与 break 语句,并找到子程序中的不可达路径。

3.2　包含多重嵌套的 if 选择结构

类似上一个实例分析过程,对于图 9 与图 10 中

源程序及流程图,算法的每一步分析过程如表 2 所示,多重嵌套 if 选择结构也能判断出不可达路径与各个变量 i, j 和 k 的取值范围。

由上面的两个子程序分析验证所有变量,最终可以确定单变量值程序静态分析方法的正确性和可靠性。

```
void Fun 2(int i){
    int j=3;
    int k=1;
    while(i>0) {
            i--=2;
             if(j<0){
                i++;
                j++;
                }
             else{
                j++;
                if(k>0)
                        k--;
                else
                        k++;
                 }
             }
}
```

图 9　待分析的程序

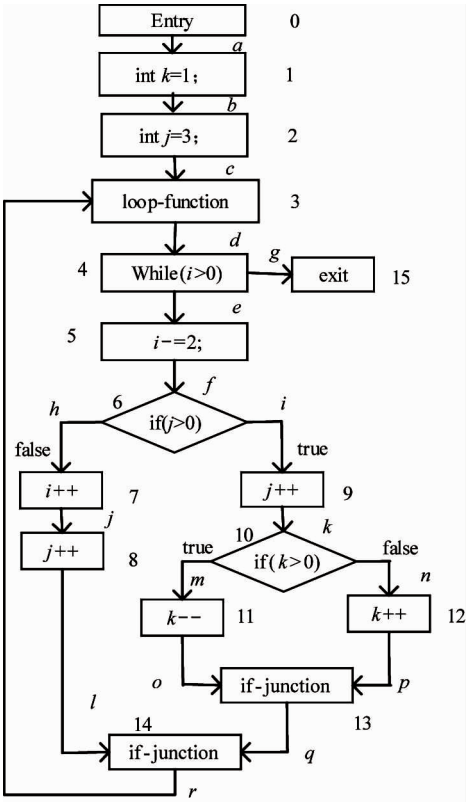

图 10　控制流程图

表 2　Fun2 函数的值范围分析过程

初始化:参数为{i},entry = #0, exit = #15, Assign = {#1, #2, #5, #7, #8, #9, #11, #12}, while = {#3}, if = {#6, #10}, junctions = {#3, #13, #14}

step	edges	inEdge	Node	localContext	junction
1	a	a	#1	b:{(i, X_N),(j,{[3,3]})}	Ø
2	b	b	#2	c:{(i, X_N),(j,{[3,3]}),(k,{[1,1]})}	Ø
3	c	c	#3		{#3}
4	Ø		#3	d: {(i, X_N),(j,{[3,3]}),(k,{[1,1]})}	{#3}
5	d	d	#4	e: {(i, {[1, +∞]}),(j,{[3,3]}),(k,{[1,1]})} g: {(i, {[−∞,0]}),(j,{[3,3]}),(k,{[1,1]})}	Ø
6	{e,g}	e	#5	f: {(i, {[−1, +∞]}),(j,{[3,3]}),(k,{[1,1]})}	Ø
7	{g,f}	g	#15		Ø
8	f	f	#6	h: {(i, {[−1, +∞]}),(j,Ø),(k,{[1,1]})} i: {(i, {[−1, +∞]}),(j,{[3,3]}),(k,{[1,1]})}	Ø
9	{h,i}	h	#7	j: {(i, {[0, +∞]}),(j,Ø),(k,{[1,1]})}	Ø
10	{i, j}	i	#9	k: {(i, {[−1, +∞]}),(j,{[4,4]}),(k,{[1,1]})}	Ø
11	{j,k}	j	#8	l: {(i, {[0, +∞]}),(j,Ø),(k,{[1,1]})}	Ø
12	{k,l}	k	#10	m: {(i, {[−1, +∞]}),(j,{[4,4]}),(k,Ø)} n: {(i, {[−1, +∞]}),(j,{[4,4]}),(k,{[1,1]})}	Ø
13	{l,m,n}	l	#14		{#14}
14	{m,n}	m	#11	o: {(i, {[−1, +∞]}),(j, {[4,4]}),(k,Ø)}	Ø

表 2 续

15	$\{n,o\}$	n	#12	p: $\{(i,\{[-1,+\infty]\}),(j,\{[4,4]\}),(k,\{[2,2]\})\}$	$\{\#13\}$
16	o	o	#13		$\{\#13\}$
17	$\varnothing$		#13	q: $\{(i,\{[-1,+\infty]\}),(j,\{[4,4]\}),(k,\{[2,2]\})\}$	$\{\#14\}$
18	q	q	#14		$\{\#14\}$
19	$\varnothing$		#14	r: $\{(i,\{[-1,+\infty]\}),(j,\{[4,4]\}),(k,\{[1,1],[2,2]\})\}$	$\varnothing$
20	r	r	#3		$\{\#3\}$
21	$\varnothing$			d: $\{(i,\{[-\infty,+\infty]\}),(j,\{[3,+\infty]\}),(k,\{[-\infty,2]\})\}$	$\{\#3\}$
…	…	…	…	…	…
51	p	p	#13		$\{\#13\}$
52	$\varnothing$		#13	q: $\{(i,\{[-1,+\infty]\}),(j,\{[4,+\infty]\}),(k,\{[-\infty,1]\})\}$	$\varnothing$
53	q	q	#14	r: $\{(i,\{[-1,+\infty]\}),(j,\{[4,+\infty]\}),(k,\{[-\infty,1]\})\}$	$\varnothing$
54	r	r	#3	d: $\{(i,\{[-\infty,+\infty]\}),(j,\{[3,+\infty]\}),(k,\{[1,1]\})\}$	$\varnothing$
55	d	d	#4	g: $\{(i,\{[-\infty,0]\}),(j,\{[3,+\infty]\}),(k,\{[1,1]\})\}$	$\varnothing$

3.3　程序代码的覆盖程度比较

本文在实现原方法所包含内容的基础上,变换不同循环结构,对 switch 选择结构和包含多重嵌套的 if 选择结构也进行测试,具体比较如表 3 所示。

表 3　程序代码覆盖程度的比较

	赋值语句	if 判断	while 循环	嵌套中包含 while 循环	嵌套中包含 if 选择	switch 语句	break 语句	多重嵌套 if 选择
RABA 算法	√	√	√	√	×	×	×	×
本实验	√	√	√	√	√	√	√	√

4　结 论

本文在统一的基于抽象解释的变量值范围的分析算法上应用不同的循环结构,研究语句中单变量的上下文状态,可对数值程序变量进行区间解释和检查验证,从而判断程序中的代码是否满足规范性与安全性。

然而,对于大型软件和硬件系统的自动分析与验证还存在一定的问题,需要进一步的研究。考虑到大型程序的数据结构复杂性更高,数值变量的调用会更错综复杂,在本文的基础上,对复杂的结构体和对象等计算可延伸到其他区间域,使之能够组合成不同的抽象方法,对抽象进行局部精化,最后形成适合数值程序变量静态检测的自动化验证工具。

参考文献

[1] 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

[2] Luo H, Liu X, Chen X, et al. Software reliability analysis using wakest reconditions in linear assignment programs[J]. *IEEE Transactions on Software Engineering*, 2016, 42(9): 866-885

[3] Mine A. Symbolic methods to enhance the precision of numerical abstract domains[J]. *Verification Model Checking & Abstract Interpretation*, 2006,22: 348-363

[4] Cousot P, Cousot R. Systematic design of program analysis frameworks[C]// ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, USA, 1979:269-282

[5] Shan C, Sun S, Xue J, et al. A dtecting method of aray bunds dfects bsed on smbolic execution[C]// School of Software Beijing Institute of Technology, Beijing, China, 2017: 373-385

[6] Min A. The octagon abstract domain[J]. *Higher-Order and Symbolic Computation*, 2006, 19(1):31-100

[7] 田楠, 陈立前, 王戟. Integer Realization Method of Pol-

yhedral Abstract Domain and Its Application in Program Analysis[EB/OL]. http://www.paper.edu.cn:IEEE, 2016

[8] Li M J, Li Z J, Chen H W. Program verification techniques based on the abstract interpretation theory[J]. *Journal of Software*, 2008, 19(1):17-26

[9] Cousot P, Cousot R. Basic concepts of abstract interpretation[J]. *LFIP Advances in Information & Communication Technology*, 2004, 156: 359-366

[10] 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

[11] 王雅文,宫云战,肖庆,等. 基于抽象解释的变量值范围分析及应用[J]. 电子学报,2011,39(2):296-303

[12] Graf S, Saidi H. Construction of abstract state graphs with PVS[J]. *Lecture Notes in Computer Science*, 1997, 1254: 72-83

[13] Neider D, Garq P, Madhusudan P, et al. Invariant synthesis for incomplete verification engines[J]. *Tools and Algorithms for the Construction and Analysis of Systems*, 2018,10805: 232-250

Value range analysis of single variable based on abstract interpretation

Li Jing, Hou Chunyan, Wang Jinsong

(School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384)

Abstract

Aiming at the problem of low code coverage in traditional variable value range analysis method based on abstract interpretation, a new method for describing single variable value range is proposed. In this method, the contradictory nodes of numerical subroutines are found, the nesting of different selection structures and cyclic structures is considered comprehensively, and the accuracy and efficiency of the analysis are also considered. The abstract interpretation theory is used as the analysis framework to study the contextual state of each variable in the program, so as to verify whether the program variables satisfy the specifications. The validation of the variable range analysis method by using published data shows that compared with the existing value range analysis method, this method has better results because it considers different nesting structures more comprehensively.

Key words: abstract interpretation, interval abstraction, single variable, context state, static analysis