区间算术在软件测试中的应用*

王志言 刘椿年

(北京工业大学计算机系 北京 100022)

摘要 程序结构测试可以分为4个阶段:静态分析、路径选择、测试数据生成和动态分析.本文应用区间算术在测试数据生成阶段对约束集求解.由于正则约束式的引入,能够处理复杂的逻辑表达式组,找到一组解以供第4阶段动态分析使用.文中提出的算法具有非常大的灵活性,可以处理非线性约束,经改进后,它甚至可以处理包含函数的表达式.

关键词 区间算术,区间削减,正则约束式,区间分裂,软件测试.

中图法分类号 TP311

软件测试从理论上可分为两大类:功能测试(Functional Testing)和结构测试(Structural Testing).结构测试可分为4个阶段:静态分析、路径选择、测试数据生成和动态执行.[1]与功能测试相对照,结构测试是把被测试程序看作一个白盒子,从分析其具体结构入手,进行测试.具体过程是:在静态分析阶段,读入被测试程序的源代码,生成控制流图.然后根据某一路径选择标准,选择出一组路径的集合(所谓路径是指从程序入口到程序出口的一次执行.它包含一些程序语句,而可能把另一些语句排除在外.例如,对于条件转移语句,1条路径只能包含它的1个分支).对于每一条路径,都有可能存在一组数据,经输入语句输入后,使程序按这条路径执行(否则,我们称之为不可执行路径).这些数据可以从控制转移语句的条件表达式得到.最后,我们利用这些数据,动态执行程序,记录程序执行结果,供测试人员分析使用.

本文所要讨论的问题就是在选定某一条路径之后,如何根据相应的条件表达式生成测试数据.注意,我们并不需要得到所有的解,只要有一组解就可以了.

本文针对软件测试问题的需要,提出了基于正则约束式的、能够求解复杂算术约束集的区间计算方法,并成功地应用到我们的软件自动测试系统AUTOTEST中.

区间算术是求解算术约束集的一个方法[5],我们已成功地应用到BPU-CLP[3]中,利用区间算术对约束集进行求解.本文将区间算术应用到软件测试中去.下面先讨论一些基本概念,然后再给出算法.

1 基本概念

1.1 区间算术

在区间算术中,任意一个数或变量都用一个区间来表示,整数5可以表示为[5,5],变量x在$-3\sim6$之间取值可以表示为$[-3,6]$.特别地,对于保存在计算机中的数,由于它总有一个上下界(设下界为$dBottom$,上界为$dTop$),所以不受任何约束的变量可以表示为$[dBottom,dTop]$.

区间算术的基本运算 $+,-,*,/$ 定义如下:

设有3个变量r,d_1,d_2分别表示为$[r.bottom,r.top]$,$[d_1.bottom,d_1.top]$,$[d_2.bottom,d_2.top]$.

若$r=d_1+d_2$ 则

$$r.bottom=d_1.bottom+d_2.bottom$$
$$r.top=d_1.top+d_2.top$$

若$r=d_1-d_2$ 则

$$r.bottom=d_1.bottom-d_2.top$$
$$r.top=d_1.top-d_2.bottom$$

---

* 本文研究得到国家自然科学基金和国家863高科技项目基金资助.
作者王志言,1972年生,硕士生,主要研究领域为人工智能.刘椿年,1944年生,博士,教授,主要研究领域为人工智能,软件工程.

本文通讯联系人:刘椿年,北京 100022,北京工业大学计算机系
本文1997-03-02收到原稿,1997-06-16收到修改稿

对于乘、除法,要分区间定义.

若 $r=d_1 * d_2$

(1) $d_1.bottom>=0, d_2.bottom>=0$
$r.bottom=d_1.bottom * d_2.bottom$
$r.top=d_1.top * d_2.top$

(2) $d_1.top<0, d_2.bottum>=0$
$r.bottom=d_1.bottom * d_2.top$
$r.top=d_1.top * d_2.bottom$

(3) $d_1.bottom>=0, d_2.top<0$
$r.bottom=d_1.top * d_2.bottom$
$r.top=d_1.bottom * d_2.top$

(4) $d_1.top<0, d_2.top<0$
$r.bottom=d_1.top * d_2.top$
$r.top=d_1.bottom * d_2.bottom$

除法以此类推.

1.2 区间削减

含有变量的算术约束(等式或不等式约束)可以削减变量的取值区间,称为区间削减(Interval Narrowing).若有变量 x 和表达式 $x=y+1$,其中 $x=[3,5]$, $y=[3,7]$,则

$$x=[3,5]$$
$$y+1=[3,7]+[1,1]=[4,8]$$

等式约束意味着 x 应该从它原来的区间被约束到它与 $y+1$ 的取值区间的交,为[4,5].所以,x 新的区间为[4,5].我们称表达式 $x=y+1$ 对 x 进行了区间削减.

因为[3,5]不是[4,8]的子集,即 x 的区间实际上是被削减了,我们称削减成功.否则认为削减失败.对于区间削减的不同结果,将在算法中作不同的处理.

相应地,还可以针对不等式约束做区间削减,以大于号为例:

若 $d_1>d_2$ 则

```
if(d1.bottom<=d2.bottom)
  d1.bottom=d2.bottom+1
```

对 d_2 亦有类似的结果(关于 $d_2.top$).

1.3 正则约束式

为使区间算术可以应用于软件测试,我们提出了使用正则约束式(Normal Form of Constraint)的结构来表示变量之间的约束关系.我们所处理的数据有两种类型:整型和布尔型.一个正则约束式形如下面的表达式

$$r=d_1 \text{ option } d_2$$

其中 option 是指+,-,*,/等算术运算或=,!=,>,>=,<,<=等逻辑运算,d_1 和 d_2 为整型(变量或常数),当option为算术运算时,r 为整型变量;option 为布尔运算时,r 为布尔型变量.

2 约束分解算法

算法可分为4步:初始化、事例(Case)生成、区间削减和区间分裂(Interval Splitting).

在这个算法中,我们使用两个表:变量表和正则约束式表.变量表保存算法中用到的所有变量,包括常量和临时生成的内部变量.在正则约束式表中,option 用于记录运算的类型,另外的3个变量只表示为指向变量表的索引,以记录正则约束式中的变量之间的约束关系.

步骤1. 初始化

第1步的目的是根据被测试的源程序填这两个表.在我们提出的约束分解法中,只需对被测试程序进行一遍扫描,就可以得到约束集.而且避免了重复计算,大大提高了算法的效率,从而克服了传统方法的缺点.

根据约束分解算法,在初始化阶段,我们只需处理3种语句:变量声明语句、赋值语句和各种条件语句中的条件表达式.方法如下:

(1) 变量声明语句:将所声明的变量填入变量表.

(2) 赋值语句:分析赋值号后面的算术表达式,填正则约束式表.把在分析过程中遇到的常量和临时变量填入变量表.而对于变量,则反向查找(我们在后面会谈到为什么反向查找)已部分生成的变量表,找到后,将其索引填入正则

约束式表.注意,由于变量都是先声明后使用,所以一定能够找到.在分析完赋值号后面的表达式后,会有一个临时变量,它的值代表了整个表达式的值,设其为 V_i.读入赋值号前面的被赋值变量名,用它代替 V_i.即被赋值变量的值代表了整个表达式的值.在这里,对于被赋值变量,我们没有查变量表,而是让它代替了临时变量 V_i,赋值语句的左部变量总是再次填入变量表.这是因为要考虑如下情况,例如:

```
    int x,y
    x1:=2
    y1:=3
*   x2:=y1+1
    if x2>4
    ... ...
```

很明显,在赋值语句 * 前后的 x 不应该是同一个 x,赋值语句为 x 赋予了全新的约束关系.条件表达式中对 x 的引用必须引用赋值语句 * 后新填入的 x.也正是由于变量在表中的多重出现,我们必须反向查找变量表,才能找到我们所需要的变量索引,把正确的约束关系填入正则约束式.

(3) 条件表达式:用上面的方法分析条件表达式,这时,应该会有临时变量 V_i,它的值代表了整个表达式的值.根据软件测试第 2 阶段所选定的路径,令临时变量 V_i 为真或假.

设函数 Initialize()完成初始化的工作.

步骤 2. 事例生成

这一步的存在有两个原因:①在区间算术中,乘除法是分正负考虑的;②在进行区间削减时,所有的布尔变量都必须是已经确定的常量.这一阶段的任务就是在进行区间削减之前,把这些待定的因素都确定下来.所以,我们必须用穷举法确定布尔变量的值,还要对参与乘除运算的变量分区间进行讨论.我们称每一种可能为一个事例 (Case).

设函数 BOOL GenCase()生成下一个事例.在实际实现中,对每一个变量,我们分大于零、等于零、小于零 3 种情况讨论.

步骤 3. 区间削减

区间削减是根据我们前面定义的区间运算,利用正则约束式对变量的区间进行削减.由于已经经过了事例生成阶段,所以,此时所有的布尔变量均为常量,参与乘除运算的整形变量均被确定为大于零、等于零或小于零,我们可以选定乘除法中的某一情况进行运算.用伪语言书写的算法如下:

```
BOOL Solving (VariableArray)
{
    SetConstraintActive ();
    while (NotEmpty (ActiveArray))
    {
        i=SelectAConstraint ();
        if (Narrowing (constraint (i).r))
        {
            if (VariableEmpty (constraint (i).r))
                return FALSE;
            else
                Refreshing (constraint (i).r);
        }
        if (Narrowing (constraint (i).d1))
        { ... ... }
        if (Narrowing (constraint (i).d2))
        { ... ... }
        MoveToInactiveArray (i);
    }
    return TRUE;
}
```

这是一个以正则约束式为顺序的对变量表进行区间削减的算法.我们为正则约束式维护两个队列,活动队列 ActiveArray 和非活动队列 InactiveArray.在算法开始时,函数 SetConstraintActive()把所有的正则约束式置入活动队列.从活动队列中选一个正则约束式,对这个约束式中的变量进行区间削减.如果削减成功,则将处在非活动队列中的所有与这一变量相关的正则约束式移入活动队列,以便对其它的变量进一步进行削减.最后,把当前的正则约束式移入非活动队列.当活动队列为空时,算法结束,函数返回真.

在上面的算法中,我们应注意,当函数返回假时,表明算法无解,但返回真时,并不能表明算法有解.而且,在变量表中,还可能有几个变量的结果是处在某一段区间,不是确定的解.因此,当算法结束时,可能已经得到确定的一组解,另外还有以下几种可能:①算法已得到解,但不唯一;②本事例有解,但没有得到;③本事例无解,但算法终止时没有返回假.对于软件测试,我们只需得到一组解.所以,上述3种情况都可以通过算法的第4步区间分裂来解决.

步骤4. 区间分裂

区间分裂是一个把结果区间对分的循环过程.算法如下:

```
BOOL SplitToGetAnswer ()
{
    while (NotEmpty (IntervalStack))
    {
        Interval tmp,tmp1,tmp2;
        tmp=PopStack (IntervalStack);
        if(! IntervalSplit (tmp,tmp1,tmp2))
            return TRUE;
        if (Solving (tmp1))
            PushStack (IntervalStack,tmp1);
        if (Solving (tmp2))
            PushStack (IntervalStack,tmp2);
    }
    return FALSE;
}
```

以上是算法的4步,总的算法如下:

```
BOOL GetAnswer ()
{
    Initialize ();
    While (GenCase ())
    {
        if (Solving())
            if (SplitToGetAnswer ())
                return TRUE;
    }
    return FALSE;
}
```

3 例　子

下面用一个例子来说明整个算法的执行过程.设有一段程序如下:

```
1. int x,y;
2. if ((x>=y+1) and (x<0))
3.     if (x*y=6)
4.         ... ...
```

若要求语句4被执行,整个算法的过程如下:

第1. 初始化

	变量表	正则约束式表
1. y	[−32 768,32 767]	3:1+2
2.	[1,1]	5:4>=3
3. v_1	[−32 768,32 767]	7:4<6
4. x	[−32 768,32 767]	8:5 and 7
5. v_2	T/F	9:4*1
6.	[0,0]	11:9=10
7. v_3	T/F	
8. v_4	T	
9. v_5	[−32 768,32 767]	
10.	[6,6]	
11. v_6	T	

注意,正则约束式中的参数是指向变量表的指针.

第2. 事例生成

在本例中,变量1和4为整型变量,要分大于零、等于零、小于零3种情况讨论.变量5和7为布尔变量,要分T和F进行讨论.所以,一共有$3^3 * 2^2 = 108$个事例.函数*GenCase*()会去掉一些明显错误的事例.正则约束式4使得变量5和7均为真,而正则约束式5使得变量1和4与变量9有约束关系.所以,函数*GenCase*()实际输出的事例只有9个,前4个列出如下

	变量表1	变量表2	变量表3	变量表4	
1	y	[1,32 767]	[−32 768,−1]	[1,32 767]	[−32 768,−1]
4	x	[1,32 767]	[−32 768,−1]	[−32 768,−1]	[1,32 767]
5	v_2	T	T	T	T
7	v_3	T	T	T	T
9	v_5	[1,32 767]	[1,32 767]	[−32 768,−1]	[−32 768,−1]

此外,还有5个变量9为[0,0]时的事例.

第3. 区间削减

第1个事例在区间削减后无解.在对第2个事例进行区间削减后,函数*Solving*()返回真.此时,变量表为:

1. y [−6,−2]
2. [1,1]
3. v_1 [−5,−1]
4. x [−3,−1]
5. v_2 T
6. [0,0]
7. v_3 T
8. v_4 T
9. v_5 [6,6]
10. [6,6]
11. v_6 T

第4. 区间分裂

将上面的变量表压栈,开始进行区间分裂.

取出栈顶元素,找到第1个可以被分裂的变量,在这里是y[−6,−2],分裂为[−6,−4]和[−3,−2].用分裂后的变量再进行区间削减,得到结果如下:

		变量表1(y[−6,−4])	变量表2(y[−3,−2])
1.	y	[−6,−6]	[−3,−3]
2.		[1,1]	[1,1]
3.	v_1	[−5,−5]	[−2,−2]
4.	x	[−1,−1]	[−2,−2]
5.	v_2	T	T
6.		[0,0]	[0,0]
7.	v_3	T	T
8.	v_4	T	T
9.	v_5	[6,6]	[6,6]
10.		[6,6]	[6,6]
11.	v_6	T	T

由于对两组变量进行区间削减,函数*Solving*()都返回真,所以,将两组变量压栈.

再次取栈顶元素,取出了上面的变量表2.在变量表2中,已经没有可以被分裂的变量,函数*SplitToGetAnswer*()返回真,得到一组解,$x=-3, y=-2$.

因为我们只要求得到一组答案,所以,至此整个算法结束.

4 关于算法性质的讨论

4.1 算法正确性与终止性

关于程序的正确性,我们要证明两点.

第1点是要证明当算法终止、返回值为真时,我们所得到的确实是一组解.这一点很容易证明,因为在SplitToGetAnswer()中包含对确定解(即各个变量均无法再分裂)进行Solving()的过程(每一个变量表只有成功才能压栈),这实际上是对解的验算过程.

第2点是要证明,当返回值为假时,变量表无解.为此,我们只需证明,算法在进行运算时,不会把有效的解削减掉.函数GenCase()可以保证遍历所有的解空间,当变量表中包含解时,区间算术会保证在进行区间削减时,Solving()不会把真正的解削减掉,SplitToGetAnswer()是一个对分穷举的过程,也不会把有效的解漏掉.这样,当返回值为假时,变量表无解.

下面我们来证明算法的终止性.

函数GenCase()和SplitToGetAnswer()是遍历和循环过程,其终止性易于保证.我们只讨论函数Solving()的终止性.函数终止的条件是活动队列为空,每一次区间削减都会导致一个正则约束式被移出活动队列,只有区间削减成功才会导致某些正则约束式移入活动队列.而各个变量的区间都是有限的,即区间削减成功的次数是有限的,因此,活动队列最终会为空,算法必定终止.

4.2 算法评价与改进

在计算机中,所有的数都是有限的和离散的,这就使得区间算术特别适用于计算机中的约束求解过程.我们对正则约束式的引入使得对复杂的逻辑表达式求解成为可能.正是由于正则约束式的引入,使得我们在初始化阶段所采用的方法与传统方法截然不同,只需对被测试程序进行一遍正向扫描就可以了.同时,正则约束式的引入还消除了重复计算,大大提高了系统的效率.

该算法的灵活性是很大的,我们甚至可以定义某些函数,例如三角函数的区间运算,使得它能够对包含函数的表达式求解.

这个算法是适用于整形变量的.虽然从理论上说,它也可以适用于任意精度的离散数据,但当精度很大时,特别是对于标准C中的浮点数,其精度可达10^{-38},计算量明显加大,效果并不好.而且,当路径相对长一些,我们得到的正则约束式比较多时,区间削减的次数也会大大增加.为此,我们需要对这个算法进行一些改进.例如,对于浮点数,我们可以限定区间的最小尺寸,假设为10^{-2};或者在进行约束求解的过程中进行过滤,把3个变量都已经约束为常量的正则约束式从约束集中去掉,等等.通过附加这些改进方法,可以大大减少区间削减的次数,使得区间算术在处理这些相对复杂的情况时,效果也会比较好.

在区间算术中,我们所处理的区间本质上是一个有序集,这就使得用区间算术处理字符变量非常方便,由于我们可以把字符集看作有序集,所以我们几乎不需要进行什么大的修改就可以使这种方法完全适应于字符变量.而对于真正的集合运算来说,就不那么简单了,主要的问题在于要找到"与"、"或"等集合运算的逆运算,这样,我们才能把这种算法有效地应用于集合运算,而这些将是我们今后的任务.

参考文献

1 Huang J C. An approach to program testing. Computing Surveys, Sep. 1975,7(3):113~128

2 Lee J H M, Van Emden M H. Interval computation as deduction in CHIP. Journal of Logic Programming, 1993,16(3~4):255~277

3 刘椿年,张秀珍,杨凯等.多重论域CLP系统及其部分演绎.软件学报,1996,7(863专刊):303~309.
(Liu Chun-nian, Zhang Xiu-zhen, Yang Kai *et al.* A multi-domain CLP system and its partial deduction. Journal of Software, 1996,7(863 special issue):303~309)

The Application of Interval Computation in Software Testing

WANG Zhi-yan LIU Chun-nian

(*Department of Computer Science Beijing Polytechnic University Beijing* 100022)

Abstract There are four phases in structural testing: static analysis, path selection, case generation and dynamic analysis. In this paper, the authors use interval computation as deduction in the third phase, case generation. Because of the introduction of normal forms of constraints, the authors can analyze complicated logic expressions now, and get the answer. This method is very flexible, with the capability of dealing with non-linear constraints. And when extended, it can even deal with functions in expressions.

Key words Interval computation, interval narrowing, normal forms of constraints, interval splitting, software testing.