软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(7):1741−1756 [doi: 10.13328/j.cnki.jos.004836]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

含有析取语义循环的不变式生成改进方法*

潘建东[1,2], 陈立前[3], 黄达明[1,2], 孙 浩[1,2], 曾庆凯[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](南京大学 计算机科学与技术系,江苏 南京 210023)

[3](并行与分布处理国防科技重点实验室(国防科学技术大学 计算机学院),湖南 长沙 410073)

通讯作者: 曾庆凯, E-mail: zqk@nju.edu.cn

摘 要: 抽象解释为程序不变式的自动化生成提供了通用的框架,但是该框架下的大多数已有数值抽象域只能表达几何上是凸的约束集.因此,对于包含(所对应的约束集是非凸的)析取语义的特殊程序结构,采用传统数值抽象域会导致分析结果不精确.针对显式和隐式含有析取语义的循环结构,提出了基于循环分解和归纳推理的不变式生成改进方法,缓解了抽象解释分析中出现的语义损失问题.实验结果表明:相比已有方法,该方法能为这种包含析取语义的循环结构生成更加精确的不变式,并且有益于一些安全性质的推理.

关键词: 抽象解释;抽象域;不变式;析取语义;循环分解

中图法分类号: TP311

中文引用格式: 潘建东,陈立前,黄达明,孙浩,曾庆凯.含有析取语义循环的不变式生成改进方法.软件学报,2016,27(7):1741−1756. http://www.jos.org.cn/1000-9825/4836.htm

英文引用格式: Pan JD, Chen LQ, Huang DM, Sun H, Zeng QK. Improving invariant generation for loops containing disjunctive semantics. Ruan Jian Xue Bao/Journal of Software, 2016,27(7):1741−1756 (in Chinese). http://www.jos.org.cn/1000-9825/4836.htm

Improving Invariant Generation for Loops Containing Disjunctive Semantics

PAN Jian-Dong[1,2], CHEN Li-Qian[3], HUANG Da-Ming[1,2], SUN Hao[1,2], ZENG Qing-Kai[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[3](National Laboratory for Parallel and Distributed Processing (School of Computer Science, National University of Defense Technology), Changsha 410073, China)

Abstract: Abstract interpretation provides a general framework to generate program invariants automatically. However, most existing numerical abstract domains under this framework can only express constraint sets that are geometrically convex. Therefore, using convex abstract domains to analyze special program structures involving disjunctive semantics (whose constraint sets are non-convex) may lead to imprecise results. This paper presents a novel approach based on loop decomposition and deduction to improve invariant generation for loop structures with explicit and implicit disjunctive semantics. This approach can alleviate the problem of great semantic loss during abstract interpretation of loop structures with disjunctive semantics. Compared with existing approaches, experimental results show that

* 基金项目: 国家自然科学基金(61170070, 61572248, 61431008, 61321491); 国家科技支撑计划(2012BAK26B01); 国家高技术研究发展计划(863)(2011AA1A202)

Foundation item: National Natural Science Foundation of China (61170070, 61572248, 61431008, 61321491); National Key Technology Research and Development Program of the Ministry of Science and Technology of China (2012BAK26B01); National High Technology Research and Development Program of China (863) (2011AA1A202)

收稿时间: 2014-12-12; 修改时间: 2015-03-02; 采用时间: 2015-03-23; jos 在线出版时间: 2015-11-12

CNKI 网络优先出版: 2015-11-11 17:00:17, http://www.cnki.net/kcms/detail/11.2560.TP.20151111.1700.001.html

the presented approach can generate more precise invariants for loop structures involving disjunctive semantics, which is also helpful for reasoning about certain security properties.

Key words: abstract interpretation; abstract domain; invariant; disjunctive semantics; loop decomposition

不变式是在某一程序点处,程序的任意执行状态都满足的断言,被广泛用于程序的正确性证明、终止性证明和安全性质的推理.因此,自动化地生成不变式一直都是程序分析与验证领域中一个重要的课题.抽象解释[1]是基于程序语义的不变式自动化生成的一般方法,其主要思想是:通过对程序具体语义进行抽象(或上近似),获得可计算的程序抽象语义;然后基于抽象语义,通过静态地分析程序的动态执行来获得不变式.具体而言,抽象解释把程序状态抽象为抽象域上的域元素,把程序动作抽象为抽象域上的域操作;然后,根据程序的控制流图,在抽象域上通过调用对应的域操作,采用基于迭代的方法来计算程序的抽象不动点,最终获得每个程序点处的不变式.不同的抽象域可以表达不同的程序性质,常用的数值抽象域有区间抽象域[2]、同余抽象域[3]、凸多面体抽象域[4]等.目前,有很多程序分析工具都是基于抽象解释来生成不变式的,包括 ASTREE[5],Interproc[6],StInG[7],InvGen[8],ExtItv[9]等.

在计算程序不动点[10]时,抽象解释方法为了保证循环的收敛,会调用加宽操作对变量的可能取值进行放大;针对控制流接合(control-flow join),会调用接合(join)操作来计算两个输入抽象域元素的最小上界[11].这两类操作均会造成生成的不变式精度上的损失,这种损失在分析含有析取语义的程序时尤为显著.

1) 显式析取语义

对于循环中存在的 if…else 语句结构,抽象解释方法在分析时会对两个分支的不变式进行接合(join),将两个分支状态的最小上界作为下一轮循环分析的输入.每一次接合操作都会对不变式进行放大,造成精度损失.这是因为 if…else 结构是析取的语义,用接合操作虽然保证了可靠性、加速了循环收敛,但却引入了不可行的变量取值空间.如图 1 中的程序 prog1 及其产生的不变式所示,无法验证程序点③处 $x+y>50$ 的程序性质(prog1 中有多个变量,因此使用可推导出变量间关系的凸多面体抽象域来分析).

```
prog1
1 int x, y;
2 x=60; y=20;                ①
3 while (x>1) do
4     if (x>50) then
5         x=x−y;
6         else
7             x=x−1;
8             y=y+1;
9     endif;
10        y=y+1;
11    done;              ②    //assert (x+y>50)      ③
```

位置	凸多面体抽象域
①	⊤ (no information)
②	$-x+1\geqslant 0$

Fig.1 Program prog1 (left) and its generated invariants (right)

图 1 程序 prog1(左)及其产生的不变式(右)

对于这种析取语义所导致的精度损失,Sharma 等人[12]提出了基于切分谓词(splitter predicate)的不变式生成改进方法.其基本思想是:借助阶段切分谓词(phase splitter predicate),可将程序行为分为多个阶段的循环,分解为串行的若干个单阶段循环.考虑形如 while $(P)\{\overline{B};B[C]\}$ 的多阶段循环,其中,P 为循环判断条件,B 为循环中条件分支语句及其之后的所有语句,$\overline{B}$ 为循环体内 B 之前的所有语句,C 为循环中条件分支语句的判断条件.候选的切分谓词由 C 经最弱前置条件 $WP(\overline{B},C)$ 计算得到,用断言 Q 表示.

当 Q 满足的条件 $\{P\wedge Q\}B[C]\{Q\vee\neg P\}$ 和 $\{\neg Q\}\overline{B}\{\neg C\}$ 在任何情况下都成立时,称 Q 为原循环的阶段切分谓词.可以分解循环为串行的单阶段循环,即:

$$\text{while } (P\wedge\neg Q)\{B[\text{false}]\};\ \text{while } (P\wedge Q)\{B[\text{true}]\}.$$

但是,选择和 C 相关的切分谓词过于粗糙,并非所有循环都能得到满足条件的阶段切分谓词.例如,对于图 1 中的程序 prog1,P 为 $x>1$,唯一的循环内条件分支语句的判断条件 C 为 $x>50$,$\overline{B}$ 为空,$B[C]$为 prog1 代码的第 4

行~第 10 行(如图 1 所示).按照文献[12]的方法,得到 $Q=WP(\overline{B},C)=x>50$, 这是唯一的候选切分谓词.计算发现,存在某些实例,例如:当 x=60,y=20 时,满足前置条件 $P\wedge Q$;经过语句 $B[C]$,却推出后置条件 $Q\vee\neg P$ 不成立.因此,$\{P\wedge Q\}B[C]\{Q\vee\neg P\}$并不永真,该候选的 Q 不成立,因此无法找到满足条件的阶段切分谓词,也就无法对这个循环进行分解.更进一步地,即便用该方法完成了循环的分解,也只表达了某一前置条件下多阶段循环的执行流程,并不能全面表达在不同前置条件下循环的不同语义行为.

2) 隐式析取语义

含有条件分支语句的循环,是显式程序析取语义的体现.然而,即便将其分解为无条件分支的简单循环,程序中还存在一些语句,仍旧隐式地含有析取语义.例如,循环中含有 $x=a\times x+b$(其中,$a\neq0,1$)表达式的程序,当 $a<0$ 时,x 的取值范围由多个不相交的区间组成,是析取语义的体现.图 2 中的程序 prog2 具有该特征,而程序 prog3 不具有该特征.对于程序 prog2 和 prog3,同余抽象域将给出相同的循环不变式,均为 $x+1=0 \bmod 3$.但是,在 prog2 中循环的实际语义中,x 的取值密度是 $6\mathbb{Z}$(除了初始的 2 次取值外),在 prog3 中循环的实际语义中,x 的取值密度是 $3\mathbb{Z}$.可见,对于 prog2 中这种含有隐式析取语义的循环,抽象解释的抽象放大程度要远高于 prog3 这种不含隐式析取语义的循环.

这种程度的放大会影响程序安全性质的推理.以图 2 中的程序 prog2′为例,prog2′是对 prog2 的变形,即去掉了 prog2 第 2 行对于变量 x 的赋初值.假设 x 由用户输入决定.位置③处的循环不变式结果太过宽泛,并没有分析出当 x=1 时会陷入死循环的不安全状态.另一方面,使用描述单变量性质的同余抽象域和区间抽象域进行分析,prog2 中位置④处的不变式是$-x+29\geqslant0\wedge x-11\geqslant0\wedge x+1=0 \bmod 3$,包含 x=20,程序推理时会产生除数为 0 的警告.可是 x 的实际语义是 x=17,对于之后的除法操作来说是安全的,因此之前的警告是误报.

```
prog2                          prog3                          prog2′
1   int x;                     1   int x;                     1   int x;          ③
2   x=2;            ①         2   x=2;            ②         2   while (x⩽10) do
3   while (x⩽10) do            3   while (x⩽10) do           3     x=-2*x+3;
4     x=-2*x+3;                4     x=x+3;                   4   done;
5   done;           ④         5   done;
6   x=6/(x-20);                6   x=6/(x-20);
```

位置	同余抽象域+区间抽象域的分析结果
①	$x+1=0 \bmod 3$; $-x+29\geqslant0$; $x+13\geqslant0$
②	$x+1=0 \bmod 3$; $-x+11\geqslant0$; $x-2\geqslant0$
③	$x=0 \bmod 1$;
④	$x+1=0 \bmod 3$; $-x+29\geqslant0$; $x-11\geqslant0$

Fig.2 Programs prog2, prog3, prog2′ (above) and the generated invariants (below)

图 2 程序 prog2,prog3,prog2′(上)及其产生的不变式(下)对比

为了解决上述问题,本文提出了针对显式和隐式含有析取语义的循环结构的不变式生成改进方法,运用基于图的循环分解和归纳推理的程序分析技术辅助抽象解释,以生成更精确的不变式.

1) 通过研究判断条件划分的程序状态在循环语句中的迁移,构造循环的状态可达迁移图.状态可达迁移图涵盖了比阶段切分谓词 Q[12]更多的程序信息,利用这些信息把含有条件分支语句的循环进行更细粒度的循环分解.将其具有的复杂结构,根据前置状态的划分,横向分解为多条路径;对于每条路径,基于它的前置状态,纵向地将其对应的语句序列分解为不含条件分支的简单循环形式.这种多维度的分析比文献[12]的单维度更全面地表达了循环语义,从而使得循环分解更加全面和精确.最后,根据循环前置条件和分解的路径分别开展抽象解释的分析,生成的析取形式的不变式比文献[12]中的方法更精确;
2) 通过研究 $x=a\times x+b$ 赋值语句在循环中的取值规律,借助归纳推理直接推导出循环不变式,从而避免了循环迭代过程中最小上界的求解,更精确地表达了分解后子循环的语义.

最后,本文将改进方法嵌入到传统数值抽象域(如区间抽象域、凸多面体抽象域等)中,以提升原抽象域的表达能力.实验结果表明,嵌入了本文改进方法的数值抽象域相比使用了文献[12]中方法的数值抽象域,能够生成更精确的不变式.

本文第 1 节给出包含条件分支语句的循环分解和不变式生成方法.第 2 节给出单变量线性赋值循环的不变式生成方法及其扩展.第 3 节讨论嵌入本文改进方法的数值抽象域的原型实现和对比性实验.第 4 节比较相关工作.第 5 节总结全文并展望未来工作.

1 包含条件分支语句的循环的不变式生成方法

对于包含条件分支语句的复杂控制流循环,本文提出基于状态可达迁移图的语义等价的循环分解方法,有条件地消去循环体中 if…else 控制结构,分析出循环的所有可能执行路径.再根据循环前置条件,对分解后循环的子路径的幂集扩展[13]作抽象解释,从而生成更精确的析取形式的不变式.

1.1 基本定义

循环体内含有条件分支语句,形如 while (*BW*){*blockBefore*;*ifBlock*;*blockAfter*}的循环称为含有条件分支语句的循环.其中,*BW* 为 while 循环的判断条件,*ifBlock* 为循环中首个出现的条件分支语句,*blockBefore* 和 *blockAfter* 分别为 *ifBlock* 前、后的语句块.设 *ifBlock* 所表示的条件分支语句形如 if (*B*) s_1 else s_2,其中,s_1,s_2 分别为判断条件 *B*,¬*B* 下的语句块(if…else 分支中可能存在嵌套的 if…else 语句,递归处理即可).

下面考虑含有条件分支语句的循环的一般形式:while (*BW*){*blockBefore*;if (*B*) then s_1 else s_2;*blockAfter*}.算法 1 将其转换为如下规范形式:while (*BW*){if (B_1) then *trans_*s_1;if (B_2) then *trans_*s_2},其中,B_1,B_2,*trans_*s_1,*trans_*s_2 均由算法 1 得到.

算法 1. 规范化算法 *norm_loop*(*LoopBody*).

输入:一般形式的循环体 *blockBefore*; if (*B*) then s_1 else s_2; *blockAfter*(不考虑嵌套循环);

输出:规范形式的循环体 if (B_1) then *trans_*s_1; if (B_2) then *trans_*s_2.

1. $s_1' \leftarrow s_1$; *norm_loop*(*blockAfter*)
2. $s_2' \leftarrow s_2$; *norm_loop*(*blockAfter*)
3. *trans_*$s_1 \leftarrow$ *blockBefore*; s_1'
4. *trans_*$s_2 \leftarrow$ *blockBefore*; s_2'
5. $B' \leftarrow WP$(*blockBefore*,*B*)
6. $\sim B' \leftarrow WP$(*blockBefore*,¬*B*)
7. $\Phi \leftarrow B' \cap B'$
8. $B_1 \leftarrow B' \setminus \Phi$
9. $B_2 \leftarrow \sim B' \setminus \Phi$

值得注意的是:算法 1 中,若直接将 *blockBefore* 后移到 *ifBlock* 内,有时会改变程序语义.因为 *blockBefore* 中如果有对 *B* 中变量的赋值修改的话,必将影响分支判断的结果.因此需要把 *B*,¬*B* 分别作为后置条件,计算关于 *blockBefore* 的最弱前置条件来保证可靠性.

特别地,*WP*(*blockBefore*,*B*)和 *WP*(*blockBefore*,¬*B*)有时并不是互补的,因为在最弱前置条件的计算过程中,为了保证可靠性,有时会对变量取值空间进行一定程度的放大.因此,存在交集:

$$\Phi = WP(blockBefore,B) \cap WP(blockBefore,\neg B).$$

当循环以 Φ 为前置状态时,无法判断循环执行时会选择哪条分支.因此,当循环以 Φ 为前置状态时不进行规范化和循环分解,仍旧用原抽象解释方法生成不变式.含有条件分支语句循环的规范化是要把循环中可判定语义的部分和不能判定的部分区分开,对可判定部分作进一步分析.因此,互斥访问的循环内条件分支的判断条件 B_1 为 $WP(blockBefore,B)\setminus\Phi$,$B_2$ 为 $WP(blockBefore,\neg B)\setminus\Phi$.这样,将循环中的语句全部变换到条件分支内,有利于循环状态的提取和状态可达迁移图的构造.

定义 1(循环路径的前置状态). 循环路径的前置状态是指循环中能够执行某条路径所对应的前置条件的全集.

根据循环路径的前置状态的定义,含有条件分支语句的循环规范形式 while (BW)\{if (B_1) then $trans\_s_1$;if (B_2) then $trans\_s_2$\}中存在 3 条循环路径,分别为 $trans\_s_1$,$trans\_s_2$ 和 skip.那么,循环路径的前置状态分别为:

1) 执行 $trans\_s_1$ 路径的前置状态为 $\boxed{BW\cap B_1}$;
2) 执行 trans_s_2 路径的前置状态为 $\boxed{BW\cap B_2}$;
3) 跳出循环的前置状态为$\boxed{\neg BW}$.

1.2 状态可达迁移图

为了分析循环迭代过程中条件分支语句的执行规律,我们根据循环路径的前置状态,利用抽象解释,静态地模拟状态的迁移,得到状态间的迁移条件,据此构造循环的状态可达迁移图,从而更精确地表达含有条件分支语句循环的语义.由于在状态 $\boxed{BW\cap\Phi}$下无法进行更细粒度的语义分析,因此循环的状态可达迁移图中并不包含该状态下的迁移过程.

算法 2. 含有条件分支语句循环的状态可达迁移图构造算法.

输入:含有条件分支语句循环的规范形式 while (BW)\{if (B_1) then $trans\_s_1$;if (B_2) then $trans\_s_2$\};

初始状态集 $\boxed{BW\cap B_1}$,$\boxed{BW\cap B_2}$,$\boxed{\neg BW}$;

输出:状态可达迁移图.

1. 建立基础状态结点 $\boxed{BW\cap B_1}$,$\boxed{BW\cap B_2}$,结束状态结点$\boxed{\neg BW}$;
2. 以 $\boxed{BW\cap B_1}$为初始状态,采用抽象解释求对应代码块 $trans\_s_1$ 的后置条件状态 $\boxed{after\_s_1}$;
   以 $\boxed{BW\cap B_2}$为初始状态,采用抽象解释求对应代码块 $trans\_s_2$ 的后置条件状态 $\boxed{after\_s_2}$;
3. 建立中间状态结点 $\boxed{after\_s_1}$,$\boxed{after\_s_2}$.添加从基础状态结点到中间状态结点的迁移边,用虚线箭头表示(注意:迁移边是有向的,代表了迁移关系;迁移函数为导致状态迁移的代码块,标记在迁移边的上方);
4. 以$\boxed{\neg BW}$为初始状态时,因为跳出循环,所以代码块为空,在图中省略.
5. 用 $\boxed{after\_s_1}$,$\boxed{after_s_2}$分别和 3 个初始状态集求交,当交集不为空时,认为可达.添加从中间状态结点到基础状态结点的可达边.用实线箭头表示(注意:可达边是有向的,代表了可达关系;计算可达条件,标记在可达边的上方).

算法 2 中,抽象解释方法的可靠性保证了状态间迁移的可靠性.下面以图 1 中的程序 prog1 为例,构造其状态可达迁移图(如图 3 所示).

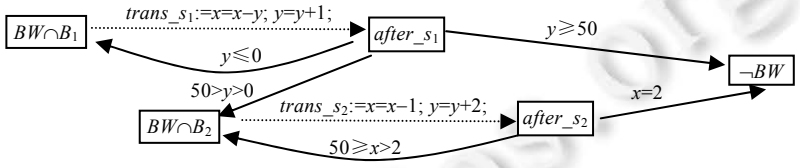


Fig.3 Loop states and the state transition graph of program prog1

图 3 程序 prog1 的循环状态和状态可达迁移图

图 3 的生成过程描述如下:

1) 提取程序 prog1 中的循环 while $(x>1)$ {if $(x>50)$ then {$x=x-y;$} else {$x=x-1;y=y+1;$}$y=y+1;$}.根据算法 1,得到 $trans\_s_1$ 为{$x=x-y;y=y+1;$},$trans\_s_2$ 为{$x=x-1;y=y+1;y=y+1;$}.因为 *blockBefore* 为空语句,所以 B_1 为 $x>50$,B_2 为 $x\leqslant 50$,Φ为∅.那么,prog1 的规范形式为

   while $(x>1)$\{if $(x>50)$ then {$x=x-y;y=y+1$};if $(x\leqslant 50)$ then {$x=x-1;y=y+1;y=y+1;$}\}.

2) 根据定义 1 和 prog1 的规范形式,得到循环初始状态为:
   - 基础状态结点:$\boxed{BW\cap B_1}=x>1\wedge x>50=x>50$,$\boxed{BW\cap B_2}=1<x\leqslant 50$;

- 结束状态结点:$\boxed{\neg BW}$=$x\leqslant 1$;

3) 根据算法 2,构建状态可达迁移图.首先,建立基础状态结点和结束状态结点;然后,计算中间状态结点 $\boxed{after_s_1}$为 $x=x_{origin}-y_{origin};y=y_{origin}+1(x_{origin}>50)$,$\boxed{after_s_2}$为 $x=x_{origin}-1;y=y_{origin}+2(x_{origin}>1\wedge x_{origin}\leqslant 50)$;添加迁移边;虚线上方标记迁移函数;最后,计算中间状态结点到基础状态结点和结束状态结点的可达条件.例如:要使 $\boxed{after_s_1}\cap\boxed{BW\cap B_1}$成立,则需使得 $x=(x_{origin}-y_{origin})>50\wedge x_{origin}>50$ 有解,解得 $y_{origin}\leqslant 0$,为中间状态 $\boxed{after_s_1}$到基础状态 $\boxed{BW\cap B_1}$的可达条件.其他同理.当有解时,添加可达边.实线上方标记可达条件.当无解时,不标记.

1.3 基于状态可达迁移图的语义等价的循环分解方法

定义 2(状态的自环). 在含有条件分支语句循环的状态可达迁移图中,一个基础状态经过迁移函数迁移到中间状态,如果该中间状态在某个条件下又对自身可达,我们称这样一种结构为状态的自环.

状态的自环对应于实际程序中的意义是:含有条件分支语句的循环在某条执行路径下会反复执行的一段语句序列.亦即经过语义分析得到的一个细粒度的子循环,循环体是迁移函数,循环控制条件是函数对应的前置状态.可见,状态可达迁移图自然地完成了对每条执行路径下循环的分解.状态的自环有可能构成不终止的子循环,形成一条不安全路径.可以通过状态可达迁移图进行判断.具体地,以状态的自环的可达条件作为前置条件,对状态的自环对应的迁移函数开展抽象解释,得到后置条件.如果前置条件能够蕴含生成的后置条件,那么意味着之后的每次迭代,后置条件都是前置条件的子集,亦即一直满足循环控制条件,则说明该状态下的自环是死循环(不终止路径).例如,程序 prog1 的状态可达迁移图中状态 $\boxed{BW\cap B_1}$的自环对应的子循环是 while (x>50) {$x=x-y;y=y+1$;},可达条件是 $y<0$,迁移函数是代码块 $trans\_s_1$:={$x=x-y;y=y+1$;}.以 $y<0$ 为前置条件,关于 $trans\_s_1$ 的后置条件是 $y<1$,不是原可达条件 $y<0$ 的子集.那么该状态的自环所对应的程序不是死循环.同理,状态 $\boxed{BW\cap B_2}$ 的自环对应的子循环也不是死循环.当修改程序 prog1,使得它的标准形式中 $trans\_s_1'$:={$x=x-y;y=y-1$;},记作程序 prog1′.那么再以 $y<0$ 为前置条件,关于代码块 $trans\_s_1'$ 的后置条件为 $y<-1$.$y<0$ 能够蕴含 $y<-1$.说明在 $y<0$ 的状态下,该子循环是死循环,这是一条不安全路径.

定理 1. 当含有条件分支语句循环的状态可达迁移图不存在环路(但可存在状态的自环)时,循环内的条件分支不存在交叉访问.

证明:假设循环内的条件分支存在交叉访问,即,基础状态间存在互相的迁移.表现为:$\exists x_1\in BW\cap B_1$,经过 $trans\_s_1$ 的变换得到 $x_1',x_1'\in BW\cap B_2$;$\exists x_2\in BW\cap B_2$,经过 $trans\_s_2$ 的变换得到 $x_2',x_2'\in BW\cap B_1$.于是,存在迁移路径,使得 $\boxed{BW\cap B_1}\rightarrow\boxed{BW\cap B_2}$可达,$\boxed{BW\cap B_2}\rightarrow\boxed{BW\cap B_1}$可达.从而,该循环的状态可达迁移图中存在环路 $\boxed{BW\cap B_1}\leftrightarrows\boxed{BW\cap B_2}$.然而,这与状态可达迁移图中不存在环路的命题相悖. □

定理 1 是用本文方法分解含有条件分支语句的循环的前提条件.当含有条件分支语句循环的状态可达迁移图存在环路时,则无法静态地获得循环状态间互相访问的次数,甚至可能存在循环内的条件分支无限次地互相访问而导致循环不终止的现象.需要结合动态方法进一步分析,本文不作讨论.当含有条件分支语句循环的状态可达迁移图不存在环路时,满足定理 1,根据不同的初始状态,循环执行中状态迁移的次数和顺序是固定的,因此,这样的循环可以进行分解.

根据循环的状态可达迁移图可知:在不同的前置条件下,循环会执行不同的子路径.为了在任意前置条件下全面分析循环,我们选择展开状态可达迁移图中所有的程序执行路径.即,从 3 个循环状态 $\boxed{BW\cap B_1}$,$\boxed{BW\cap B_2}$和 $\boxed{\neg BW}$出发的所有可达状态的序列的集合.

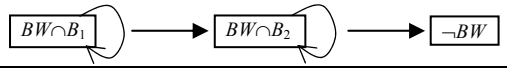
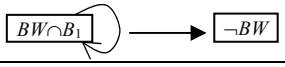
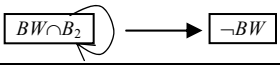
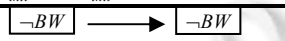
由定理 1 可知:如果状态可达迁移图中不存在环路,那么状态可达路径是有限的.可以通过枚举的方式将复杂路径分解为多条展开路径.每条展开路径中都把状态的自环看作是一个状态结点.例如,程序 prog1 中的状态可达迁移图中的路径如表 1 所示.

- 状态的自环完成了在某次执行路径中,含有条件分支语句循环的细粒度解析,是纵向的循环分解;
- 不同执行路径的展开完成了在不同循环路径的前置状态下,循环不同执行轨迹的刻画,是横向的循环分解.

这种从两个维度对循环的分解,是对循环语义更深层次的挖掘.随后,借助抽象解释技术,对分解后的循环结构计算不变式.结合不变式幂集扩展的表达形式,能够生成更精确的不变式.

Table 1 Decomposed paths of the reachable state transition graph

表 1 状态可达迁移图的分解路径

路径 1	
对应程序	While (x>50){x=x−y;y=y+1;} while (x>1 && x≤50){x=x−1;y=y+2;}
前置状态	x_{init}>50∧y_{init}<50
路径 2	
对应程序	While (x>50){x=x−y;y=y+1;}
前置状态	x_{init}>50∧y_{init}≥50
路径 3	
对应程序	While (x>1 && x≤50){x=x−1;y=y+2;}
前置状态	x_{init}≤50∧x_{init}>1
路径 4	
对应程序	skip
前置状态	x_{init}≤1

下面,我们分别从前置条件涵盖单一路径和多条路径的情况阐述不变式的生成.

1) 当循环的前置条件是确定的标量或者能够确定只在某条路径对应的前置状态全集内,则原循环就等价于某一条分路径.分路径对应的程序对原循环进行了极大的简化和分解,并且没有损失程序语义.直接对分路径作抽象解释,可以得到更精确的循环不变式和循环后置条件.

例如,根据程序 prog1 的状态可达迁移图的分路径和循环的前置条件:x_{init}∈[60,60]∧y_{init}∈[20,20],循环只会执行表 1 中的路径 1.那么,直接对路径 1 作抽象解释,得到循环不变式为

$$x+y-52\geqslant 0\wedge x+2y-54\geqslant 0;$$

后置条件不变式为

$$-x+1\geqslant 0\wedge x+y-52\geqslant 0 \tag{1}$$

由图 1 中描述的,凸多面体抽象域分析程序 prog1,得到循环不变式为

$$\top \text{ (no information)};$$

后置条件(位置②)不变式为

$$-x+1\geqslant 0 \tag{2}$$

可见:相比文献[12]中的方法,本文提出的基于状态可达迁移图的循环分解方法能够进行语义等价的循环分解;相比传统的基于凸多面体抽象域的抽象解释方法,本文方法能够分析出有意义的循环不变式.公式(1)蕴含公式(2),并且公式(1)能够验证断言 $x+y$>50,而公式(2)不能.可见,本文的改进方法生成了更精确的不变式.

2) 当循环的前置条件是某个区间值并且牵涉多条路径时,则用路径幂集扩展的方法来表达程序的析取语义.主要思想是:把与分路径的前置状态的交集作为对应分路径的前置条件,分别用抽象解释方法根据分路径所对应的程序得到循环不变式,然后把这些不变式用析取符号连接,即得到对应路径不变式的幂集扩展,也就是最终的不变式结果.这样的计算过程完全保留了程序的析取语义.

例如,修改程序 prog1 的前置条件状态得到程序 prog1″,其初始状态修改为 x_{init}∈(40,60)∧y_{init}∈[20,20],其余不变.与各个分路径的前置状态全集求交,发现存在两条可执行的路径:表 1 中的路径 1 和路径 3.把关于 x 的前置状态拆分为两个集合:x_{init}∈(40,50] 和 x_{init}∈(50,60),把它们分别作为路径 1 和路径 3 的前置条件,再作抽象解释,将生成的不变式用析取符号连接.

得到循环不变式为

$$\begin{aligned}&(x_{init}-60<0\wedge x_{init}-50>0\wedge y_{init}-20=0\wedge -x-21y+480\geqslant 0\wedge\\&\ -x+20y+459\geqslant 0\wedge y-20\geqslant 0\wedge x+y-52\geqslant 0\wedge x+20y-451\geqslant 0)\vee\\&(x_{init}-40>0\wedge x_{init}-50\leqslant 0\wedge y_{init}-20=0\wedge -2x-y+120\geqslant 0\wedge\\&\ y-20\geqslant 0\wedge x-1\geqslant 0\wedge 2x+y-102\geqslant 0)\end{aligned}\tag{3}$$

后置条件不变式为

$$\begin{aligned}&(x_{init}-60<0\wedge x_{init}-50>0\wedge y_{init}-20=0\wedge x-1=0\wedge 148y-7839\geqslant 0)\vee\\&(x_{init}-40>0\wedge x_{init}-50\leqslant 0\wedge y_{init}-20=0\wedge x-1=0\wedge -y+118\geqslant 0\wedge y-100\geqslant 0)\end{aligned}\tag{4}$$

凸多面体抽象域分析程序 prog1″,得到循环不变式为

$$y-20\geqslant 0\wedge x+10y-241\geqslant 0\wedge 300x+300y-8621\geqslant 0\wedge 2021x+9000y-251651\geqslant 0\tag{5}$$

后置条件不变式为

$$-x+1\geqslant 0\wedge 300x+300y-8621\geqslant 0\tag{6}$$

公式(3)蕴含公式(5),公式(4)蕴含公式(6).可见:本文的方法表达了多种可能执行路径的程序语义,生成了更精确的不变式.

2 单变量线性赋值循环的不变式生成方法

循环分解后,对于子路径中不含条件分支的循环,如果其中含有 $x=a\times x+b$ 结构,则仍旧隐式地含有析取语义.本文采用归纳推理的方法分析,直接生成不变式,从而完全避免了抽象解释的分析过程中循环迭代计算不动点带来的语义损失和不精确这样的问题.

2.1 单变量线性赋值循环的指数型循环不变式

定义 3(单变量线性赋值循环). 含有线性赋值语句形如 $x=a\times x+b(a,b\in\mathbb{Z})$的循环,当循环体内不再对变量 x 进行二次赋值时,我们称这样的循环为单变量线性赋值循环.

定义 4(指数型循环不变式). 形如 $x=t_1\times t_2^n+t_3$ ($n\in\mathbb{N}$)称为指数型不变式,其中,t_1,t_2,t_3 是常量,n 是任意自然数.当指数型不变式满足循环不变式的定义[14]时,我们称这样的不变式为指数型循环不变式.

在实际的程序中,单变量线性赋值循环结构较为常见.其中,当 $a=0$ 时,线性赋值退化为常量赋值;当 $a=1$ 时,x 在循环中呈线性变化.这些都是简单且易于分析的.

当 $a\neq 0$ 且 $a\neq 1$ 时,单变量线性赋值循环呈现出非线性变化.特别地,当 $a<0$ 时,x 的取值发生振荡变化,且由多个不相交的区间组成,是析取语义的体现.现有的抽象解释方法无法生成精确的不变式.本文通过研究其取值特征后发现:指数型循环不变式可以精确地表达单变量线性赋值循环的非线性语义,生成精确的不变式.定理 2 和定理 3 分别描述了当 $a>0$ 和 $a<0$ 时,单变量线性赋值循环具有的指数型循环不变式的性质,并给出证明过程.

由于指数型循环不变式和循环迭代次数相关,因此,我们在循环中引入循环计数器整型变量 n.进入循环前插装 $n=0$,循环体尾部插装语句 $n=n+1$.

定理 2. 单变量线性赋值循环 while (){…;$x=a\times x+b$;…},当 $a>0$ 且 $a\neq 1$ 时,存在指数型循环不变式:

$$x=t_1\times t_2^n+t_3,$$

其中,$t_1=x_0+b/(a-1)$,$t_2=a$,$t_3=-b/(a-1)$,x_0 为进入循环前 x 的初始值.

证明:参见附录. □

根据定理 2 中描述的指数型循环不变式的特征,变量 x 的取值规律为:

1) 在任何判断条件下,循环不终止的条件为 $x_0=b/(1-a)$.此时,$x\equiv b/(1-a)$;
2) 当 $0<a<1$ 时,x 单调递增的条件为 $x_0<b/(1-a)$;x 单调递减的条件为 $x_0>b/(1-a)$;
3) 当 $a>1$ 时,x 单调递增的条件为 $x_0>b/(1-a)$;x 单调递减的条件为 $x_0<b/(1-a)$.

对于单变量线性赋值循环 while (){…;$x=a\times x+b$;…}中 $a<0$ 的情况,当 n 无限大时,无论 x_0 为何值,x 的取值最终会呈现一正一负的振荡变化形式.如果仍然用定理 2 给出的不变式描述,则无法表达变量的变化规律,也无法

描述变量取值分布的属性.针对这一问题,本文经过分析,加上逻辑符号的辅助,采用两个单调变化的指数型循环不变式的析取形式,来刻画单变量线性赋值循环取值分布的特性.

引理 1. $\forall n_1,n_2\in\mathbb{N}$,等式$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$不成立,其中,$x_0\neq b/(1-a)$.

证明:参见附录. □

定理 3. 单变量线性赋值循环 while (){…;$x=a\times x+b$;…},当 $a<0$ 时,存在指数型循环不变式:

$$x=(t_1'\times t_2^{n_1}+t_3)\vee(t_1''\times t_2^{n_2}+t_3),$$

其中, $t_1'=x_0+b/(a-1),t_1''=a\times x_0+a\times b/(a-1)$, $t_2=a,t_3=-b/(a-1),x_0$ 为进入循环前 x 的初始值,$n_1=n/2$(其中,n mod 2=0),$n_2=(n-1)/2$(其中,n mod 2=1).

证明:参见附录. □

根据定理 3 中描述的指数型循环不变式的特征,变量 x 的取值规律为:

1) 当$-1<a<0$ 时,公式 $x=t_1'\times t_2^{n_1}+t_3$ 单调递减、公式 $x=t_1''\times t_2^{n_2}+t_3$ 单调递增的条件为 $x_0>b/(1-a)$,且两式分别从正负两个方向的初始值 x_0 和 $a\times x+b_0$ 开始,无限逼近常量 $b/(1-a)$;公式 $x=t_1''\times t_2^{n_2}+t_3$ 单调递减、公式 $x=t_1'\times t_2^{n_1}+t_3$ 单调递增的条件为 $x_0<b/(1-a)$,且两式分别从正负两个方向的初始值 $a\times x+b_0$ 和 x_0 开始,无限逼近常量 $b/(1-a)$.公式 $x=(t_1'\times t_2^{n_1}+t_3)\vee(t_1''\times t_2^{n_2}+t_3)$ 的取值是离散的且无交点.

2) 当 $a<-1$ 时,公式 $x=t_1'\times t_2^{n_1}+t_3$ 单调递减、公式 $x=t_1''\times t_2^{n_2}+t_3$ 单调递增的条件为 $x_0<b/(1-a)$,且两式分别从初始值 x_0 和 $a\times x+b_0$ 开始向$-\infty$和$+\infty$方向发散;公式 $x=t_1''\times t_2^{n_2}+t_3$ 单调递减、公式 $x=t_1'\times t_2^{n_1}+t_3$ 单调递增的条件为 $x_0>b/(1-a)$,且两式分别从初始值 $a\times x+b_0$ 和 x_0 开始向$-\infty$和$+\infty$方向发散.公式 $x=(t_1'\times t_2^{n_1}+t_3)\vee(t_1''\times t_2^{n_2}+t_3)$ 的取值是离散的且无交点.

例如,程序 prog2 中的单变量线性赋值循环,其中,$x_0=2,a=-2,b=3$.运用定理 3,生成的指数型循环不变式为 $x=4^{n_1}+1\vee(-2)\times 4^{n_2}+1$. 再根据循环判断条件 $x\leqslant 10$,可以推得 n 只能取到 0,1,2,3,4.

当循环初始值 x_0 是标量,并且循环判断条件中关于 x 的限定(如果有的话)是一个确定区间$[l,r]$时,我们可以根据一些规则直接获得循环后置条件中指数型不变式中的 n 值,从而得到精确的循环后置条件不变式,为后续程序不变式的推理带来精度上的提高.同时,规则中还列出了循环的不安全状态.本文考虑的规则如下:

规则 A. 当 $0<a<1$ 时,根据定理 2,单变量线性赋值循环的循环后置条件为:

- 当 $l\leqslant x_0<\frac{b}{1-a}$ 时,不变式单调递增,跳出循环时的后置条件为 $x=\left(x_0+\frac{b}{a-1}\right)\times a^{n_N}-\frac{b}{a-1}>r$,其中,$n_N$ 为满足条件的最小的自然数.当 $r=+\infty$时,循环不终止.当 $x_0<l$ 时,无法进入循环;
- 当 $r\geqslant x_0>\frac{b}{1-a}$ 时,不变式单调递减,跳出循环时的后置条件为 $x=\left(x_0+\frac{b}{a-1}\right)\times a^{n_M}-\frac{b}{a-1}<l$, 其中,$n_M$ 为满足条件的最大的自然数.当 $l=-\infty$时,循环不终止.当 $x_0>r$ 时,无法进入循环.

规则 B. 当 $a>1$ 时,根据定理 2,单变量线性赋值循环的循环后置条件为:

- 当 $l\leqslant x_0<\frac{b}{1-a}$ 时,不变式单调递减,跳出循环时的后置条件为 $x=\left(x_0+\frac{b}{a-1}\right)\times a^{n_M}-\frac{b}{a-1}<l$, 其中,$n_M$ 为满足条件的最大的自然数.当 $l=-\infty$时,循环不终止.当 $x_0<l$ 时,无法进入循环;
- 当 $r\geqslant x_0>\frac{b}{1-a}$ 时,不变式单调递增,跳出循环时的后置条件为 $x=\left(x_0+\frac{b}{a-1}\right)\times a^{n_N}-\frac{b}{a-1}>r$, 其中,$n_N$ 为满足条件的最小的自然数.当 $r=+\infty$时,循环不终止.当 $x_0>r$ 时,无法进入循环.

规则 C. 当 $a<0$ 时,根据定理 3,单变量线性赋值循环的循环后置条件为:

当 $x_0\neq\frac{b}{1-a}$ 且 $x_0\in[l,r]$时,不变式的取值是震荡变化的.因此,根据这个特征,分别求得公式(7)~公式(10)中满足条件的最小的 n_1,n_1',n_2,n_2'.

$$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}<l \tag{7}$$

$$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1'}-\frac{b}{a-1}>r \tag{8}$$

$$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_2+1}-\frac{b}{a-1}<l \tag{9}$$

$$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_2'+1}-\frac{b}{a-1}>r \tag{10}$$

其中, $n_1,n_1',n_2,n_2'\in\mathbb{N}$.再比较这 4 个值 $n_{1\mathrm{MIN}},n_{1\mathrm{MIN}}',n_{2\mathrm{MIN}},n_{2\mathrm{MIN}}'$ (这 4 个值并不一定都存在),选取其中最小的作为后置条件中的 n 值,代入所属的不变式,求得此时 x 的值.假设该值为 x_t,则 $x=x_t$ 为跳出循环时的后置条件.如果不存在任何满足条件的 n_1,n_1',n_2,n_2' ,则循环不终止.

例如,对于程序 prog2 中的单变量线性赋值循环,有 $x_0=2,a=-2,b=3,l=-\infty,r=10$,那么可以根据规则 C 来计算不变式.通过计算可知:公式(7)、公式(9)、公式(10)均没有满足条件的 n_1,n_2,n_2' ;满足公式(8)的 n_1' 值最小为 $n_{1\mathrm{MIN}}'=2$,代入公式(8)可得 $x=x_t=17$,则改进方法生成的循环后置条件不变式为

$$x-17=0 \tag{11}$$

对比经典抽象解释方法获得的循环后置条件不变式为

$$x+1=0 \bmod 3\wedge -x+29\geqslant 0\wedge x-11\geqslant 0 \tag{12}$$

显然,公式(11)蕴含公式(12),因此公式(11)更加精确,本文的改进方法生成了更加精确的不变式.

2.2 带区间系数的线性赋值循环

当单变量线性赋值循环 while (){…;$x=a\times x+b$;…}中 a,b 不是确定的值,而是用区间形式表示的值的范围时,同样可以利用本文方法快速生成指数型循环不变式.假设分别用$[a_1,a_2],[b_1,b_2]$表示 a,b 的值范围,x_0 为循环初始值, n 表示插装的循环计数器,则根据定理 1、定理 2 可知:

1) 当 $0,1\notin[a_1,a_2]$时,带区间系数的线性赋值循环的循环不变式为

$$x=x_0\times[a_1,a_2]^n+\frac{[b_1,b_2]\times[a_1,a_2]^n}{[a_1-1,a_1-1]}-\frac{[b_1,b_2]}{[a_1-1,a_2-1]};$$

2) 当 $0\in[a_1,a_2]$且 $1\notin[a_1,a_2]$时,则 $x=b$ 的赋值语句存在执行的可能.相当于此时 $x_0=b$,并带入之后的循环迭代中.原循环重新从初始值为 b 开始迭代计算 x 的值,因此循环不变式为

$$x=\{x_0,b\}\times[a_1,a_2]^n+\frac{[b_1,b_2]\times[a_1,a_2]^n}{[a_1-1,a_1-1]}-\frac{[b_1,b_2]}{[a_1-1,a_2-1]};$$

3) 当 $1\in[a_1,a_2]$时,由于此时会导致指数型循环不变式的除数为 0,因此不能用本文方法得到结果.这种情况下,本文将返回$(-\infty,+\infty)$.

本节方法具有较强的通用性.可通过区间线性化的技术[15]加以扩展,将更一般形式的多变量以及非线性循环程序抽象为带区间系数的线性赋值循环,从而应用本节方法.在抽象解释框架下,任意形式的表达式都可以通过区间化技术[15]抽象为带区间系数的线性表达式.因此,任意形式的非嵌套循环都可以抽象为带区间系数的线性赋值循环.对于有多个变量存在且含有非线性赋值语句的循环,形如 while (){…;$x=y\times x+z$;…}.假设我们关注的是变量 x,则先对程序用区间抽象域开展抽象解释得到循环不变式,假设得到的 y 和 z 的取值区间分别用$[y_1,y_2]$和$[z_1,z_2]$表示.得到转化后的带区间系数的线性赋值循环为 while (){…;$x=[y_1,y_2]\times x+[z_1,z_2]$;…}.另外,对于含有条件分支语句的循环,形如 while (){…;if () then $x=a_1\times x+b_1$; else $x=a_2\times x+b_2$;…},我们也采用区间组合技术[16]把该循环抽象为带区间系数的线性赋值循环.假设 $a_1\leqslant a_2,b_1\leqslant b_2$,则 x 的系数可取值 a_1 和 a_2.为了保证可靠性,放大成区间形式为$[a_1,a_2]$.同理,表达式的常量部分可取值$[b_1,b_2]$,得到转化后的带区间系数的线性赋值循环为 while (){…;$x=[a_1,a_2]\times x+[b_1,b_2]$;…}.虽然区间组合技术的上近似操作会带来精度损失,但在无法进行循环分解时,仍然是一种程序转换和分析的有效方法.

3 实现与实验结果

3.1 原型系统的实现

本文在传统抽象解释框架下实现了我们提出的改进方法.具体地,对支持 APRON[17]库的基于抽象解释的静态分析工具 Interproc[6]进行扩展,加入了含有条件分支语句循环和单变量线性赋值循环的不变式生成模块.其中,修改了Interproc的程序分析流程:识别含有条件分支语句的循环,调用循环分解函数将其分解为简单循环,根据可达的分路径的幂集扩展,生成析取形式的不变式并将其融入原流程中继续分析;识别单变量线性赋值循环,提取参数,直接生成指数型循环不变式.由于指数型循环不变式不是传统数值抽象域的域元素形式,因此在保留精确形式的同时,求得每个指数型循环不变式的最大值和最小值,替换为区间的形式,随后将其带入原流程中继续分析.由第 2 节的讨论可知,每个指数型不变式都是单调的,因此这种转换是可行且易于实现的.

此外,修改了 APRON 库中区间抽象域和凸多面体抽象域的实现,增加了支持幂集扩展的计算规则:

$$(l_1\vee l_2)\odot l_3=(l_1\odot l_3)\vee(l_2\odot l_3)=\{l_1\odot l_3,l_2\odot l_3\},$$

其中,l_1,l_2,l_3 为抽象域的域元素;$\odot$表示抽象域中原有的运算符,包括逻辑运算中的交与接合、算术操作中的加减乘除、加宽算子、变窄算子.$l_1\vee l_2$ 为由改进方法获得的析取形式的不变式,由于它是通过语义分析所得,由前面的分析可知,l_1,l_2 不存在交集.

3.2 实验评估

为了评估本文方法的有效性,我们把基于循环分解的改进方法和基于归纳推理的改进方法分别嵌入传统数值抽象域中,并与没有嵌入本文改进方法的传统数值抽象域在生成不变式的精度和性能等方面进行了比较.随后,将两种改进方法同时嵌入到传统数值抽象域中,以说明同时考虑两种方法对不变式生成的精度有更大程度的提高.本文采用的实验平台为 Ubuntu 12.04 Linux 操作系统,4G 物理内存,Intel i5 3.2GHz 四核 CPU 处理器.

3.2.1 基于循环分解的改进方法

为了说明基于状态可达迁移图的循环分解方法在处理条件分支循环上的优势,将其与同样处理该类循环的文献[12]中的阶段切分谓词方法进行对比.表 2 中给出这两种方法分别嵌入凸多面体抽象域的对比实验结果.

Table 2 Experimental results of the method based on loop decomposition

表 2 基于循环分解改进方法的对比实验结果

程序名	变量数目	凸多面体抽象域+循环分解方法		凸多面体抽象域+阶段切分谓词方法		精度比较
		计算时间(s)	安全性质(Y/N)	计算时间(s)	安全性质(Y/N)	
prog1	2	0.021	NS	0.013	NS	>
prog1′	2	0.020	Y	0.013	N	>
prog1″	2	0.024	NS	0.014	NS	>
rajamani	5	0.033	Y	0.020	N	>
phase	3	0.023	NS	0.012	NS	=
tacs08	5	0.025	NS	0.016	NS	=
tacs08′	5	0.027	NS	0.017	NS	>

程序 prog1 为图 1 的代码实例.prog1′和 prog1″为 prog1 程序稍作修改的新程序,在第 1.3 节给出了描述.并且,以上 3 个程序作为例子在第 1 节中给出了基于循环分解方法的不变式生成过程以及与凸多面体抽象域分析结果的对比(由本文开始部分可知,该程序用文献[12]中方法无法处理).rajamani 是 InvGen 工具的测试用例.phase 是 Frama-c[18]工具的测试用例.tacs08 是文献[19]的例子,tacs08′是 tacs08 的一个变种,把其标量值的初始状态放大为区间值.

表 2 中:“变量数目”栏给出了程序中的变量数目;“精度比较”栏比较了两种方法在程序结尾处生成的不变式的精度,其中,>表示本文改进方法生成的不变式的精度大于所对比的方法,=表示本文改进方法生成的不变式精度和所对比的方法一样;“安全性质”栏比较了两种方法生成的不变式能否证明程序中安全性质相关的断言,其中,Y 表示能够证明,N 表示不能证明,NS 表示程序中没有和安全性质相关的断言.

表 2 中的实验结果显示:针对条件分支循环,与文献[12]中的阶段切分谓词方法相比,基于循环分解的方法

能够优化某些文献[12]中方法所不能处理的循环,并且生成的不变式比原抽象域的分析结果更精确.当文献[12]中的方法能够改进时,本文方法生成的不变式精度不低于文献[12].具体地,由表 2 可知:

- 前 4 个测试程序均不满足文献[12]中的判断规则,因此文献[12]中的方法无法处理;而基于循环分解的改进方法能够进行优化,进而生成比原抽象域分析结果更精确的不变式;
- 后 3 个程序用本文方法和文献[12]中方法均能处理,但当程序变量初始值不存在确定值时,程序执行会存在多种路径可能,本文方法能够更细粒度地分析,生成比文献[12]中方法更精确的不变式(如程序 tacs08′);当存在确定的单一路径时,本文方法和文献[12]中方法生成的不变式相同;而且本文方法能够识别程序中可能的不终止路径,并给出警告.

3.2.2 基于归纳推理的改进方法

为了说明归纳推理方法在处理单变量线性赋值循环上的优势,将其生成的指数型不变式和未借助归纳推理方法的区间抽象域的分析结果进行对比.需要指出的是:目前实现的此类改进只是针对非关系型的单变量不变式,而区间抽象域正是针对单变量的抽象域.表 3 中给出了对比实验结果.

Table 3 Experimental results of the method based on deduction

表 3 基于归纳推理改进方法的对比实验结果

程序名	变量数目	区间抽象域+归纳推理方法		区间抽象域		精度比较
		计算时间(s)	安全性质(Y/N)	计算时间(s)	安全性质(Y/N)	
prog2	1	0.008	Y	0.006	N	>
prog2′	1	0.008	Y	0.005	N	>
PADO01	1	0.006	NS	0.004	NS	>
SAS09	2	0.010	NS	0.006	NS	>
rec_lin2	6	0.028	Y	0.019	N	>
10-Filter	8	0.031	NS	0.022	NS	>

程序 prog2 和 prog2′为图 2 的代码实例,已在第 2 节作为例子分析了利用归纳推理方法的不变式生成过程以及与区间抽象域分析结果的差别.PADO01 和 SAS09 分别是文献[20]和文献[16]中的程序片段.rec_lin2 和 10-Filter 分别是文献[21]和文献[22]中的例子.这些程序均是含有或者经过化简含有单变量线性赋值循环的程序.

表 3 中的各栏说明同表 2.表 3 中的实验结果显示:针对单变量线性赋值循环,相比于区间抽象域的分析结果,本文的基于归纳推理的改进方法嵌入区间抽象域后能够生成更精确的不变式.并且,根据指数型循环不变式的性质(规则 A~规则 C),能够分析出程序中导致不安全状态的变量取值.

3.2.3 本文改进方法的总体效果

由于凸多面体抽象域是在区间抽象域的基础上构造的,从单变量扩展为表达变量间关系的区间性质,因此,凸多面体抽象域的分析结果包含了区间抽象域的分析结果.于是,本文在验证改进方法总体效果的有效性时,将同时嵌入了两种改进方法的凸多面体抽象域的分析结果与只嵌入了其中一种的分析结果进行比较.表 4 给出了对比实验结果.

Table 4 Experimental results of the integral methods

表 4 改进方法的总体对比实验结果

程序名	变量数目	凸多面体抽象域+循环分解方法+归纳推理方法	凸多面体抽象域+循环分解方法	凸多面体抽象域+归纳推理方法	精度比较	
		计算时间(s)	计算时间(s)	计算时间(s)		
rabin	4	0.062	0.045	0.039	>	>
md5	7	0.081	0.059	0.052	>	>
ecdsa	11	0.213	0.154	0.146	>	>

据我们所知,目前的研究工作中并没有同时研究这两种程序结构的不变式生成问题,因此没有现成的测试案例.但是,现实的开源程序中不乏同时满足两个程序特征的片段,比如含有较多数值计算的加密算法.我们选取了有代表性的 3 个作为测试案例.表格中的 rabin,md5,ecdsa 分别为 rabin 加密算法、md5 加密算法、椭圆曲线签名算法的实现程序片段.由于本文改进方法实现的工具只支持 C 语言语法的子集,因此在使用之前,我们对这些现实程序进行了一些人工的修改.

表 4 中:“精度比较”分为两个子栏,分别表示同时使用了两种改进方法和只使用其中一种在生成不变式精度方面的比较;其余各栏说明同表 2.表 4 中的实验结果显示:同时使用两种改进方法应用在凸多面体抽象域上,相比于只用了一种改进方法,能够生成更精确的不变式.由于待分析程序中存在的循环同时满足含有条件分支语句和单变量线性赋值语句,说明这个程序同时含有显式析取语义和隐式析取语义.改进方法的处理过程是叠加的:条件分支循环经过分解,成为不含条件判断语句的简单循环序列后,基于归纳推理的改进方法才会去寻找单变量线性赋值循环作进一步的处理.因此不会产生冲突或相互抑制,生成的不变式精度比单用其中一种改进方法更高.由表 2~表 4 的“计算时间”栏可以看出:本文的改进方法由于需要识别特殊的程序结构,并且作细粒度的处理,会额外消耗更多的时间,但是都在可接受的范围之内(一般不会超过传统数值抽象域的两倍).

4 相关工作

不变式在程序的形式化分析和验证中发挥着重要的作用,因此,程序不变式的自动化生成是学术界一直关注的问题.Cousot[11]最先提出了抽象解释方法用于不变式的推导,该方法是在抽象域上进行符号执行,通过过近似(over-approximate)的语义分析得到不变式.近些年,Cousot 把抽象解释框架应用到终止性证明[23]等领域.Miné 等人[24]提出的八面体抽象域在实际工业界代码的分析中取得了成功应用.Chen 等人[16]提出的区间多面体抽象域对传统凸多面体抽象域进行了扩展,具有更强的表达能力.Jeannet 等人[25,26]提出了利用 Jordan 标准型的抽象加速技术来解决抽象解释对程序状态空间的放大问题.

对于含有析取语义的程序结构而言,抽象解释作为一种静态的分析方法为了加速不动点计算会得到过于保守的结果.一些近期的工作对该问题进行了改进:Sharma 等人[12]利用切分谓词对部分简单的多阶段循环进行了循环分解,但其所能解决的场景比较有限,需要满足严格的逻辑表达式;Mauborgne 等人[27]关于迹划分的工作不是完全自动化的,它需要用户输入对于划分的指导;Gulwani 等人[28]使用了控制流精化的方法获得循环迭代次数的上界,但是由于其属于更高层次的抽象表达,对于不变式的精化帮助有限.相比于以上的工作,本文引入了状态可达迁移图,能够处理的循环场景比文献[12]更多样,并且是完全自动化的,能够表达复杂程序结构的变量取值特性.

对于隐式含有析取语义的单变量线性赋值循环,现有的改进方法需要借助 Grobner 基[29]计算、一阶量词消去[30]等复杂度较高的方法来生成非线性的不变式.Mastroeni 等人[31]提出了表达代数幂性质的抽象域,主要用于随机程序的静态分析和因式分解.它能够处理本文提出的含有 $x=a\times x+b$ 的循环,但其计算效率相比本文归纳推理的方法要低很多.Yang 等人[32,33]将符号计算方法应用在程序验证领域,将不变式生成转化为半代数系统的求解,取得了较好的研究成果.该方法不仅对线性程序,而且对含有析取语义的非线性程序同样能够生成较为精确的不变式.但是该方法的计算复杂度较高,本文基于归纳推理的改进方法在处理含有 $x=a\times x+b$ 的循环时只要根据参数就可以直接生成不变式,计算复杂度更低.

5 总结和展望

本文提出了含有析取语义的循环不变式生成的改进方法.主要针对包含条件分支语句的循环和单变量线性赋值循环这两类程序,运用基于状态可达迁移图的循环分解技术和归纳推理,能够更精确地表达程序语义,从而生成更精确的不变式.实现了嵌入改进方法的传统数值抽象域的原型系统.实验结果表明,嵌入了改进方法的传统数值抽象域能够生成更精确的不变式.进一步的工作包括:结合动态方法,分析含有条件分支语句的循环和区间线性化技术的研究.

References:

[1] Cousot P, Cousot R. Abstract interpretation frameworks. Journal of Logic and Computation, 1992,2(4):511−547. [doi: 10.1093/logcom/2.4.511]

[2] Cousot P, Cousot R. Static determination of dynamic properties of programs. In: Proc. of the 2nd Int'l Symp. on Programming. Paris, 1976. 106−130. https://nyu.pure.elsevier.com/en/publications/static-determination-of-dynamic-properties-of-programs

[3] Granger P. Static analysis of arithmetical congruences. Int'l Journal of Computer Mathematics, 1989,30(3-4):165−190. [doi: 10.1080/00207168908803778]

[4] Cousot P, Halbwachs N. Automatic discovery of linear restraints among variables of a program. In: Proc. of the 5th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 1978. 84−96. [doi: 10.1145/512760.512770]

[5] Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. The ASTREE analyzer, programming languages and systems. In: Proc. of the 14th European Symp. on Programming, ESOP. Edinburgh: Springer-Verlag, 2005. 21−30. [doi: 10.1007/978-3-540-31987-0_3]

[6] Jeannet B. Interproc analyzer for recursive programs with numerical variables. INRIA. 2010. http://pop-art.inrialpes.fr/interproc/interprocweb.cgi

[7] Sankaranarayanan S, Sipma HB, Manna Z. Constraint-Based linear-relations analysis. In: Proc. of the 11th Int'l Symp. (SAS). Verona: Springer-Verlag, 2004. 53−68. [doi: 10.1007/978-3-540-27864-1_7]

[8] Gupta A, Rybalchenko A. Invgen: An efficient invariant generator. In: Proc. of the 21st Int'l Conf. (CAV). Grenoble: Springer-Verlag, 2009. 634−640. [doi: 10.1007/978-3-642-02658-4_48]

[9] Chen L, Wang J, Hou S. An abstract domain of one-variable interval linear inequalities. Chinese Journal of Computers, 2010,33(3):427−439 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=3050 [doi: 10.3724/SP.J.1016.2010.00427]

[10] Tarski A. A lattice-theoretical fixpoint theorem and its applications. Pacific journal of Mathematics, 1955,5(2):285−309. [doi: 10.2140/pjm.1955.5.285]

[11] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 1977. 238−252. [doi: 10.1145/512950.512973]

[12] Sharma R, Dillig I, Dillig T, Aiken A. Simplifying loop invariant generation using splitter predicates. In: Proc. of the 23rd Int'l Conf. (CAV). Snowbird: Springer-Verlag, 2011. 703−719. [doi: 10.1007/978-3-642-22110-1_57]

[13] Cousot P, Cousot R. Systematic design of program analysis frameworks. In: Proc. of the 6th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 1979. 269−282. [doi: 10.1145/567752.567778]

[14] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. Cambridge: MIT Press, 2001.

[15] Mine A. Weakly relational numerical abstract domains [Ph.D. Thesis]. Paris: Ecole Normale Superieure, 2004.

[16] Chen L, Miné A, Wang J, Cousot P. Interval polyhedra: An abstract domain to infer interval linear relationships. In: Proc. of the 16th Int'l Symp. (SAS). Los Angeles: Springer-Verlag, 2009. 309−325. [doi: 10.1007/978-3-642-03237-0_21]

[17] Jeannet B, Miné A. Apron: A library of numerical abstract domains for static analysis. In: Proc. of the 21st Int'l Conf. (CAV). Grenoble: Springer-Verlag, 2009. 661−667. [doi: 10.1007/978-3-642-02658-4_52]

[18] Cuoq P, Kirchner F, Kosmatov N, Prevosto V, Signoles J, Yakobowski B. Frama-c. In: Proc. of the 10th Int'l Conf. (SEFM). Greece: Springer-Verlag, 2012. 233−247. [doi: 10.1007/978-3-642-33826-7_16].

[19] Gulavani BS, Chakraborty S, Nori AV, Rajamani SK. Automatically refining abstract interpretations. In: Proc. of the 14th Int'l Conf. (TACAS). Budapest: Springer-Verlag, 2008. 443−458. [doi: 10.1007/978-3-540-78800-3_33]

[20] Mastroeni I. Numerical power analysis. In: Proc. of the 2nd Int'l Conf. (PADO). Aarhus: Springer-Verlag, 2001. 117−137. [doi: 10.1007/3-540-44978-7_8]

[21] Boldo S. Floats and ropes: A case study for formal numerical program verification. In: Proc. of the 36th Int'l Collogquium (ICALP). Rhodes: Springer-Verlag, 2009. 91−102. [doi: 10.1007/978-3-642-02930-1_8]

[22] Taylor DE, Turner JS. Classbench: A packet classification benchmark. In: Proc. of the 24th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM). IEEE, 2005. 2068−2079. [doi: 10.1109/INFCOM.2005.1498483]

[23] Cousot P, Cousot R. An abstract interpretation framework for termination. In: Proc. of the 39th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 2012. 245−258. [doi: 10.1145/2103621.2103687]

[24] Miné A. The octagon abstract domain. Higher-Order and Symbolic Computation, 2006,19(1):31−100. [doi: 10.1007/s10990-006-8609-1]

[25] Schrammel P, Jeannet B. Logico-Numerical abstract acceleration and application to the verification of data-flow programs. In: Proc. of the 18th Int'l Symp. (SAS). Venice: Springer-Verlag, 2011. 233−248. [doi: 10.1007/978-3-642-23702-7_19]

[26] Jeannet B, Schrammel P, Sankaranarayanan S. Abstract acceleration of general linear loops. In: Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 2014. 529−540. [doi: 10.1145/2535838.2535843]

[27] Mauborgne L, Rival X. Trace partitioning in abstract interpretation based static analyzers. In: Proc. of the 14th European Symp. on Programming (ESOP). Edinburgh: Springer-Verlag, 2005. 5−20. [doi: 10.1007/978-3-540-31987-0_2]

[28] Gulwani S, Jain S, Koskinen E. Control-Flow refinement and progress invariants for bound analysis. In: Proc. of the 2009 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). New York: ACM Press, 2009. 375−385. [doi: 10.1145/1542476.1542518]

[29] Sankaranarayanan S, Sipma HB, Manna Z. Non-Linear loop invariant generation using Gröbner bases. In: Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 2004. 318−329. [doi: 10.1145/964001.964028]

[30] Kovács L. Reasoning algebraically about p-solvable loops, tools and algorithms for the construction and analysis of systems. In: Proc. of the 14th Int'l Conf. (TACAS). Budapest: Springer-Verlag, 2008. 249−264. [doi: 10.1007/978-3-540-78800-3_18]

[31] Mastroeni I. Algebraic power analysis by abstract interpretation. Higher-Order and Symbolic Computation, 2004,6300:297−345. [doi: 10.1007/s10990-004-4867-y]

[32] Yang L, Zhan N, Xia B, Zhou C. Program verification by using DISCOVERER. In: Proc. of the 1st IFIP TC 2/WG 2.3 Conf. (VSTTE). Zurich: Springer-Verlag, 2005. 528−538. [doi: 10.1007/978-3-540-69149-5_58]

[33] Yang L, Zhou C, Zhan N, Xia B. Recent advances in program verification through computer algebra. Frontiers of Computer Science in China, 2010,4(1):1−16. [doi: 10.1007/s11704-009-0074-7]

附中文参考文献:

[9] 陈立前,王戟,侯苏宁.单变量区间线性不等式抽象域.计算机学报,2010,33(3):427−439. http://cjc.ict.ac.cn/qwjs/view.asp?id=3050 [doi: 10.3724/SP.J.1016.2010.00427]

附录

定理 2. 单变量线性赋值循环 while (){…;$x=a\times x+b$;…},当 $a>0$ 且 $a\neq 1$ 时,存在指数型循环不变式 $x=t_1\times t_2^n+t_3$.其中,$t_1=x_0+b/(a-1)$,$t_2=a$,$t_3=-b/(a-1)$,x_0 为进入循环前 x 的初始值.

证明:根据数学归纳法和循环不变式的定义,当 $x=x_0$,即,x 等于初始值时:

$$x=\left(x_0+\frac{b}{a-1}\right)\times a^n-\frac{b}{a-1}=x_0,$$

其中,a,b 为常量,$n\in\mathbb{N}$.

可以解得:当 $n=0$ 时,等式成立.故存在 n,使得等式成立.

假设当 $x_k=t$ 时($k\in\mathbb{N}$)等式成立,即 $x_k=\left(x_0+\frac{b}{a-1}\right)\times a^{n_k}-\frac{b}{a-1}=t$, 则当 $x_{k+1}=a\times t+b$ 时:

$$x_{k+1}=a\times x_k+b=a\times\left(\left(x_0+\frac{b}{a-1}\right)\times a^{n_k}-\frac{b}{a-1}\right)+b=\left(x_0+\frac{b}{a-1}\right)\times a^{n_k+1}-\frac{b}{a-1},$$

其中,当 $n=n_k+1$ 时,满足指数型循环不变式的形式.故存在 n,使得等式成立. □

引理 1. $\forall n_1,n_2\in\mathbb{N}$,等式 $\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$ 不成立,其中,$x_0\neq b/(1-a)$.

证明:假设存在 $n_1,n_2\in\mathbb{N}$,使得 $\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$ 成立.

那么有 $\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}=\left(x_0+\frac{b}{a-1}\right)\times a^{1+2\times n_2}$,

化简得 $2\times n_1=1+2\times n_2$,即 $n_1=0.5+n_2$,在 $n_1,n_2\in\mathbb{N}$的条件下,找不到满足条件的 n_1,n_2.

因此,$\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\wedge\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}=\varnothing(n_1,n_2\in\mathbb{N})$. □

定理 3. 单变量线性赋值循环 while (){…;$x=a\times x+b$;…},当 $a<0$ 时,存在指数型循环不变式:

$$x=(t_1'\times t_2^{n_1}+t_3)\vee(t_1''\times t_2^{n_2}+t_3),$$

其中, $t_1'=x_0+b/(a-1)$,$t_1''=a\times x_0+a\times b/(a-1)$, $t_2=a$,$t_3=-b/(a-1)$,x_0 为进入循环前 x 的初始值,$n_1=n/2$(其中,n mod

2=0),$n_2=(n-1)/2$(其中,$n \bmod 2=1$).

证明:根据数学归纳法和循环不变式的定义,当 $x=x_0$,即,x 等于初始值时:

$$x=\left(x_0+\frac{b}{a-1}\right)\times a^n-\frac{b}{a-1}=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}=x_0,$$

其中,a,b 为常量.可以解得:当 $n=0$ 时,等式成立.故存在 n,使得等式成立.

假设当 $x_k=t$ 时($k\in\mathbb{N}$),等式成立,即,公式(13)成立:

$$x_k=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1} \tag{13}$$

根据引理 1 可知,公式(13)的析取符号两边的表达式无交集,则公式(13)可化简为

$$x_k=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee x_k=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}.$$

当 $x_{k+1}=a\times t+b$ 时,要证明 $x_{k+1}=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$,即证明:

$$x_{k+1}=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee x_{k+1}=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}.$$

分以下两种情况讨论:

i. 当 $x_k=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}$ 时:

$$x_{k+1}=a\times x_k+b=a\times\left(\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\right)+b=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}.$$

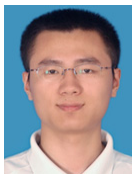
当 $n_1=n_2$ 时,得证.

ii. 当 $x_k=\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$ 时:

$$x_{k+1}=a\times x_k+b=a\times\left(\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}\right)+b=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_2+2}-\frac{b}{a-1}.$$

当 $n_1=n_2+1$ 时,得证.

综合情形 i、情形 ii,得到 $x_{k+1}=\left(x_0+\frac{b}{a-1}\right)\times a^{2\times n_1}-\frac{b}{a-1}\vee\left(a\times x_0+a\times\frac{b}{a-1}\right)\times a^{2\times n_2}-\frac{b}{a-1}$ 成立. □



潘建东(1989－),男,江苏南京人,硕士生,主要研究领域为程序分析与验证,抽象解释,信息安全.

孙浩(1987－),男,博士生,主要研究领域为信息安全,程序分析.

陈立前(1982－),男,博士,助理研究员,CCF 会员,主要研究领域为程序分析与验证,抽象解释.

曾庆凯(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,分布计算.

黄达明(1976－),男,博士生,讲师,CCF 会员,主要研究领域为形式化验证和测试,安全评估.