软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(9):1973−1984 [doi: 10.3724/SP.J.1001.2011.03898]

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

基于差分方程计算循环复杂度符号化上界*

邢建英[1+], 李梦君[1], 李舟军[2]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](北京航空航天大学 计算机学院,北京 100191)

Computing Symbolic Complexity Bounds of Loops by Recurrence Solution

XING Jian-Ying[1+], LI Meng-Jun[1], LI Zhou-Jun[2]

[1](College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

+ Corresponding author: E-mail: xjy.figgo@gmail.com

Xing JY, Li MJ, Li ZJ. Computing symbolic complexity bounds of loops by recurrence solution. *Journal of Software*, 2011,22(9):1973−1984. http://www.jos.org.cn/1000-9825/3898.htm

Abstract: Computing symbolic complexity bounds of loops can prove program's termination. Based on the solving techniques of recurrence equations and optimization problems, this paper presents a practical approach for computing complexity bounds of loops called P*-solvable loops. Two algorithms are given for loops with assignments only and loops with conditional branches respectively. Compared with some other works, this approach can attain more complexity bounds of loops. The experimental results demonstrate the practicality of this approach.

Key words: recurrence equation; optimization problem; program termination; complexity bound; closed form solution

摘 要: 计算程序中循环的程序复杂度符号化上界可以验证程序的停机性.基于差分方程和最优化问题求解技术,给出了一种计算 P*-solvable 循环程序复杂度符号化上界的有效方法.分别针对含有赋值语句的循环和带条件分支的循环,提出了其程序复杂度符号化上界计算方法.与其他工作相比,该方法能够计算得到更精确的循环复杂度符号化上界,实验结果证明了该方法的有效性.

关键词: 差分方程;最优化问题;程序停机;复杂度上界;闭合形式解

中图法分类号: TP301 **文献标识码**: A

停机性证明是程序验证的重要组成部分,完全正确的程序必须停机.但是,对于像"while *b* do *S* done"这样简单的循环,证明它的停机性也不是很容易.理论上,程序的停机性证明是一个不可判定性问题,即使循环中所有循环条件和循环体中所有赋值语句都是线性形式,它的停机性问题仍然是不可判定性的[1].对于存在多路径和循环嵌套的循环程序,验证它们的停机性更加困难.

对于程序的停机性问题,学者们已经开展了大量的研究工作.强制式线性程序的停机性研究主要采用线性秩函数方法,线性秩函数是循环变量和常数的线性组合,它的值域是一个良基集,其取值随每次循环持续递

* 基金项目: 国家自然科学基金(60703075, 60973105, 90718017); 国家教育部博士点专项基金(20070006055)
收稿时间: 2009-11-16; 修改时间: 2010-04-14; 定稿时间: 2010-06-09

减.Colón 和 Sipma 在文献[2]中将线性秩函数引入到程序停机性证明工作中,在文献[3]中,Colón 和 Sipma 基于凸多面体给出了线性秩函数的一种综合方法,并将其应用到多路径循环和嵌套循环的停机性验证.Podelski 和 Rybalchenko 在文献[4,5]中给出了线性秩函数的一种完备的综合方法,并证明了对于只含线性卫式条件和赋值语句的单路径循环,它们的停机性是可判定的,并给出了一个构造性的判定过程.Cook 与 Gulwani 在文献[6]中给出了基于线性秩函数构造计算程序停机前置条件的方法.在文献[7]中,作者研究了具有有限差分树的包含多路径的多项式程序的停机性问题,根据构造出的泰勒差分树的单调性抽象验证停机性.在函数式程序的停机性验证中[8]采用了 size-change 方法,通过给出递减或不增的表达式验证停机性.文献[9−11]研究了项重写系统(term rewriting systems)的停机性问题.

文献[12]中,Gulwani 提出了一种称为控制流精化(control flow refinement)和进度不变式(progress invariants)的方法,用来计算包含多路径和嵌套循环的循环程序的复杂度精确上界.文献[13,14]采用静态分析方法,通过插桩循环计数器变量并分别构造各个分支的不变式求得计数器的上界.但该种方法主要基于线性不变式构造技术,在这种技术中使用了抽象解释理论的 widening 技术.文献[13,14]的方法能够计算出循环计数器的一个上界,但是计算得到的可能不是精确的上界.

综上所述,程序停机性验证方法主要分为对线性循环和多项式循环两大类.其中:对线性循环停机性验证的主要方法是使用秩函数,如文献[2−6]的研究;而对多项式循环的停机性验证,文献[7]给出了一种基于有穷差分树的有效验证方法,但文献[7]的方法无法给出循环的复杂度上界;文献[12−14]能够通过不变式和控制流精化给出循环的复杂度上界.

在基于秩函数的程序停机性验证方法中,如何自动构造秩函数是一个难以解决的问题;同时,有些程序本身就不存在秩函数,因而使用基于秩函数的停机性验证方法不能验证这些程序是否停机.如下面的例子:

```
assume (0<=id<maxid);
int tmp;
while (tmp<>id) do
    if (tmp<=maxid) then
        tmp:=tmp+1;
        else tmp:=0;
    endif
done
```

文献[12]指出,上述程序不存在线性秩函数,因而基于线性秩函数的程序停机性验证方法不能验证上述程序是否停机.

在强制式程序设计语言中,导致程序不停机的主要原因是程序中循环运行不停机.因而,验证程序的停机性可以采用另外一种方法.即在循环体中插桩循环计数器变量,然后计算循环计数器变量的上界.计算循环计数器变量的上界实际是给出了程序中循环体执行次数的一个上界,如果循环计数器变量存在上界,则循环的执行过程一定停机.影响强制式程序运行效率的主要因素也是程序中的循环,因而循环计数器变量的上界也可以作为程序复杂度的一种度量.程序复杂度上界通常用来度量程序中一个功能模块的性能,一些实际应用中,通常需要计算精确的程序复杂度上界.如实时系统的设计、分析和验证中,基于不精确的程序复杂度上界可能会使设计、分析和验证过程得到不正确的分析结果.与基于秩函数的程序停机性问题比较,计算循环计数器变量的上界问题,即循环的程序复杂度计算问题,是一个更复杂的问题[13].

针对 P-solvable 循环,Kovács 在文献[15−17]中基于差分方程求解和 Gröbner 基计算提出了构造多项式等式循环不变式的一种方法.针对 P*-solvable 循环(见本文定义 1.9),本文提出了程序复杂度上界的一种精确计算方法.根据差分方程,能够精确刻画程序中循环的操作语义的优点.基于差分方程和最优化问题求解,本文分别给出了只含赋值语句和含有条件分支语句的循环中循环计数器变量的符号化上界计算算法;同时,对本文的算法进行了实现,并针对典型程序进行了实验分析.实验结果说明了本文方法的有效性.

本文第 1 节给出差分方程和最优化问题的相关概念以及描述.第 2 节针对只含赋值语句的循环给出程序符号化上界计算算法.第 3 节针对包含条件分支的循环给出程序符号化上界计算算法.第 4 节对典型程序进行了实验分析.第 5 节对本文进行总结.

1 差分方程、最优化问题和 P*-solvable 循环

定义 1.1(差分方程). 差分方程是一种离散形式的微分方程,即把微分方程中的微分用相应的差分来表示.差分方程的一般形式为

$$\begin{cases} f_1[n]-f_1[n-1]=g_1[n] \\ f_2[n]-f_2[n-1]=g_2[n] \\ \cdots \\ f_n[n]-f_n[n-1]=g_n[n] \end{cases}.$$

对于差分方程 $f[n]-f[n-1]=g[n]$,不能直接计算 $f[n]$的值,只能先计算 $f[0],f[1],\ldots,f[n-1]$才能得到 $f[n]$的值.如果 n 是一个很大的值,计算过程将需要很长时间.因而,差分方程一般求解它的闭合形式解(closed form solution).有了闭合形式解,就不需要先计算 $f[0],f[1],\ldots,f[n-1]$,可以直接计算 $f[n]$.

定义 1.2. 如果差分方程的解是差分下标 n 的函数,就称这种形式的解是差分方程的闭合形式解.

差分方程的闭合形式解的计算问题是不可判定的[15].在差分方程中,一些类型的差分方程具有求解闭合形式解的算法,如 Gosper-summable 差分方程[15]、C-finite 差分方程[15]和能够通过生成函数求解的差分方程[15]等.

定义 1.3(线性差分方程). 线性差分方程是指方程中变量之间具有线性关系的差分方程.

如 $x[n]=Ax[n-1]+Bx[n-2]+Cx[n-3]$就是一个线性差分方程,其中 A,B,C 都是整常数.

程序中循环可以等价地变换为循环变量之间的差分方程,因而差分方程可以用来刻画循环的操作语义.如果确定了循环变量前一次循环迭代的取值,应用差分方程就可以求得循环变量当前的取值.因而,差分方程经常被用来计算循环变量每次循环迭代时的取值.

例 1.4:下面的程序用来计算 x 除以 y 的商 *quo* 和余数 *rem*,用 n 作为循环计数器变量:

```
quo:=0; rem:=x;
while y<=rem do
    rem:=rem–y;
    quo:=quo+1;
done;
```

循环变量 *quo* 和 *rem* 的差分方程表示如下:

$$\begin{cases} rem[n+1]=rem[n]-y \\ quo[n+1]=quo[n]+1 \end{cases},$$

其中,循环变量 *quo* 和 *rem* 的初始值分别为 $quo[0]=0;rem[0]=x$.

求解这个差分方程,得到它的闭合形式解如下:

$$\begin{cases} rem[n]=x-ny \\ quo[n]=n \end{cases}.$$

差分方程的解给出了循环体每次执行完毕时循环变量 *quo* 和 *rem* 的取值.

计算循环计数器变量符号化上界需要求解的最优化问题是一类特殊的最优化问题,可以形式化定义如下:

定义 1.5. 设 r 是循环计数器变量,X 是循环差分方程的闭合形式解,b 是循环卫式条件,E 是由 X 和 b 取反得到的一组由不等式和方程构成的约束系统,则称$(r,X,\sim b)$是一个最优化问题.如果存在变量 r 的一个满足约束系统 E 的非负整数解 s,使得对于任意满足约束系统 E 的 r 的非负整数解 t,都有 $s\leqslant t$,则称 s 是最优化问题$(r,X,\sim b)$的最优解.

例如,对于例 1.4 中的循环,n 为其循环计数器变量,循环变量差分方程的闭合形式解为$(rem=x-ny,quo=n)$,循

环卫式条件取反得到 $y>rem$,从而$(n,rem=x-ny,quo=n,y>rem)$是一个最优化问题.

定义 1.6(循环的复杂度上界). 设 L 是一个循环,r 是循环计数器变量,如果循环 L 停机,则 L 停机时 r 的最小非负整数取值称为循环 L 的复杂度上界.

定义 1.7. 设 S 为一个程序,P 和 Q 为逻辑公式,Hoare 公式$\{P\}S\{Q\}$[18]表示对于任意满足 P 的输入,如果 S 停机,则 S 停机时其输出满足 Q.其中,P 称为前置条件,Q 称为后置条件.

定义 1.8(最弱前置条件). 设$\{P\}S\{Q\}$为一个 Hoare 公式,若对于任意满足$\{R\}S\{Q\}$的逻辑公式 R,都有 $R\to P$,则称 P 为程序 S 关于 Q 的最弱前置条件,记为 $WP(S,Q)$.

定义 1.9. 对于一个只含有赋值语句的循环,如果其循环变量 $x_1,...,x_m$ 的闭合形式解可以表示为

$$\begin{cases} x_1[n]=p_{1,1}(n)\theta_1^n+...+p_{1,s}(n)\theta_s^n \\ x_2[n]=p_{2,1}(n)\theta_1^n+...+p_{2,s}(n)\theta_s^n \\ ... \\ x_m[n]=p_{m,1}(n)\theta_1^n+...+p_{m,s}(n)\theta_s^n \end{cases},$$

并且满足:

(1) $x_i[n](1\leqslant i\leqslant m)$表示 x_i 在循环第 n 次迭代时的取值;

(2) $p_{1,1},\ldots,p_{1,s},\ldots,p_{m,1},\ldots p_{m,s}\in \bar{K}[n]$,其中,$K$ 是一个零特征的数值域(例如有理数域、实数域等),并且 $\bar{K}$ 表示 K 的代数闭包, $\bar{K}[n]$ 表示 $\bar{K}$ 中关于 n 的多项式集合;

(3) $\theta_1,\ldots,\theta_s\in \bar{K}$,

该类循环称为 P*-solvable 循环.

直观地说,一个循环称为 P*-solvable 循环当且仅当其循环变量的闭合形式解是指数序列 $\theta_i^n\in\bar{K}$ 的线性组合,并且其多项式系数 $p(n)\in\bar{K}[n]$.定义 1.9 与文献[15]中 P-solvable 循环的定义不同:Kovács 的定义中不仅要求循环存在闭合形式解,而且 $\theta_1^n,...,\theta_s^n$ 之间需要存在代数依赖,这是因为循环不变式构造算法中需要通过计算闭合形式解的 Gröbner 基来消掉变量 n;而本文的 P*-solvable 循环只要求循环存在闭合形式解,不需要构造闭合形式解的 Gröbner 基,因此本文 P*-solvable 循环比 P-solvable 循环包含的循环范围更广.P-solvable 循环只是 P*-solvable 循环的一个子集,以例 1.10 进行说明.

例 1.10:下列循环是 P*-solvable 循环但不是 P-solvable 循环的循环程序:

```
x:=1; y:=0;
while x<10 do
    x:=2*x;
    y:=y+1;
done
```

该循环的差分方程存在如下的闭合形式解:

$$\begin{cases} x[n]=2^n x[0] \\ y[n]=y[0]+n \end{cases}.$$

由于 2^n 和 n 之间不存在代数依赖,因此该循环为 P*-solvable 循环但不是 P-solvable 循环.

2 只含有赋值语句的 P*-solvable 循环的复杂度符号化上界的计算

本节将给出只含有赋值语句的 P*-solvable 循环的程序复杂度符号化上界计算算法.

定理 2.1. 假设 L 是一个形如“while b do S done”的只含有赋值语句的 P*-solvable 循环,$x_1,\ldots,x_m$ 是 S 中出现的所有循环变量, $x_1^0,...,x_m^0$ 是 $x_1,\ldots,x_m$ 的初始值,n 是循环计数器, $X=\{x_1=f_1(x_1^0,...,x_m^0,n),...,x_m=f_m(x_1^0,...,x_m^0,n)\}$ 是 L 的差分方程的闭合形式解,则当最优化问题$(n,X,\sim b)$有非负整数解时 L 必然停机,其中,$\sim b$ 为循环条件的取反.

证明:如果$(n,X,\sim b)$有一个非负整数解 n_1,当循环计数器 $n=n_1$ 时,$\sim b$ 和 X 成立,由于 X 刻画了循环 L 的操作语义,而且 $n=n_1$ 时$\sim b$ 成立,即循环 L 在 $n=n_1$ 时其循环条件 b 不满足,因此循环 L 停机.所以,$(n,X,\sim b)$有非负整数解

时 L 必然停机. □

定理 2.2. 若最优化问题$(n,X,\sim b)$有非负整数解,则$(n,X,\sim b)$的最优解就是循环 L 的复杂度上界.

证明:由定理 2.1 知,若$(n,X,\sim b)$有非负整数解,则 L 停机.采用反证法,设$(n,X,\sim b)$的最优解 n_0 不是 L 的复杂度上界,则由定义 1.6 可知,存在一个非负整数 $d<n_0$,当循环计数器取 d 时,循环 L 停机,即 $n=d$ 时 X 与$\sim b$ 均成立,从而可得 d 为$(n,X,\sim b)$的一个解;又已知 $d<n_0$,故 n_0 不是$(n,X,\sim b)$的最优解,与假设矛盾.所以,$(n,X,\sim b)$的最优解是循环 L 的复杂度上界. □

算法 1. 只含赋值语句的循环的停机性证明和程序复杂度上界计算算法.

输入:只含有赋值语句的 P*-solvable 循环 L,$X=\{x_1,\ldots,x_m\}$为循环变量, $X_0=\{x_1^0,\ldots,x_m^0\}$ 为循环变量的初始值;

输出:循环 L 的复杂度上界.

步骤 1. 提取循环的差分方程;

步骤 2. 求解差分方程的闭合形式解;

步骤 3. 将循环条件 b 取反得$\sim b$,与差分方程的闭合形式解以及循环计数器 n 构成最优化问题$(n,X,\sim b)$;

步骤 4. 求解最优化问题,计算满足$(n,X,\sim b)$的 n 的最优解;若该值存在,由定理 2.2 得知,该值为循环 L 的复杂度上界,记为 $bound(n)$;

步骤 5. 若存在非负整数的 $bound(n)$,则返回 $bound(n)$.

下面通过例 2.3 具体说明算法 1 的计算过程.

例 2.3:考虑如下的程序[15]:

```
x:=10; y:=10;
while (y>0) do
    x:=2*x;
    y:=1/2*y–1
done
```

循环的复杂度上界可以运用算法 1 按照以下步骤计算:

步骤 1. $\begin{cases} x[n+1]=2*x[n] \\ y[n+1]=\dfrac{1}{2}*y[n]-1 \end{cases}$;

步骤 2. $\begin{cases} x[n]=2^n*x[0] \\ y[n]=2^{-n}*(y[0]+2)-2 \end{cases}$;

步骤 3. 考虑循环的条件 $y>0$,将其取反得 $y\leqslant 0$;y 的差分闭合形式解代入 $y[0]$的值得 $y=12*2^{-n}-2$,从而得最优化问题$(n,y=12*2^{-n}-2,y\leqslant 0)$;

步骤 4. 求解最优化问题$(n,y=12*2^{-n}-2,y\leqslant 0)$,对于目标变量 n,计算满足约束系统$(y=12*2^{-n}-2,y\leqslant 0)$的 n 的最优解,得到复杂度上界 $bound(n)=3$;

步骤 5. 返回 $bound(n)=3$.

3 含有条件分支语句的 P*-solvable 循环的复杂度符号化上界的计算

定义 3.1. 对于含有条件分支语句的 while 循环,若其经定理 3.2、引理 3.3 所述规则转换后的循环的每个内循环都是 P*-solvable 的,并且所有内循环差分方程合并后得到的外循环的差分方程也是可解的,则该含有条件分支的 while 循环称为含有条件分支的 P*-solvable 循环.

基于第 2 节的算法,本节将针对含有条件分支语句的 P*-solvable 循环给出程序复杂度上界求解算法.在文献[15,16]中,Kovács 描述了一种如何将含有条件分支的 P-solvable 循环转化成嵌套的只含赋值语句的 P-solvable 循环的方法,基于该方法,本节首先给出将带条件分支的循环变换为嵌套 P*-solvable 循环的方法.

3.1 含有两个分支的循环

定理 3.2. 考虑如下的两个循环:

循环 3.1.1:

while b do if b_1 then S_1 else S_2 endif; done;

循环 3.1.2:

while b do

while b and b_1 do S_1 done;

while b and ~b_1 do S_2 done;

done

其中,b 与 b_1 为布尔表达式,S_1 与 S_2 为赋值语句序列.则第 1 个循环停机当且仅当第 2 个循环停机.

证明:首先,当第 1 个循环停机时,循环条件 b 被破坏,即 b 不成立,所以“b and b_1”与“b and ~b_1”也都不成立,此时第 2 个循环中的内循环与外循环都会停机.

其次,如果循环 3.1.2 停机,则其外循环和两个内循环都应该停机.所以外循环条件 b 被破坏,内循环条件“b and b_1”和“b and ~b_1”也被破坏,因而得到~b,故循环 3.1.1 也会停机.

因此,循环 3.1.1 停机当且仅当循环 3.1.2 停机. □

需要指出,实际上循环 3.1.1 与循环 3.1.2 语义等价,可以证明二者有相同的执行路径[12].

引理 3.3[15]. 设 S_0~S_3 是赋值语句序列,b 是布尔表达式并且 $b_0=WP(S_0,b)$是 S_0 关于 b 的最弱前置条件,则

$$\{P\}\ S_0;\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ endif};\ S_3\ \{Q\},$$

当且仅当

$$\{P\}\ \text{if } b' \text{ then } S_0;\ S_1;\ S_3 \text{ else } S_0;\ S_2;\ S_3 \text{ endif};\ \{Q\}.$$

基于定理 3.2 和定理 3.3,可以得到如下推论:

推论 3.4. 考虑如下两个循环:

循环 3.1.3:

while b do S_0 if b_1 then S_1 else S_2 endif; S_3; done

循环 3.1.4:

while b do

L_1: while b and b_1' do S_0; S_1; S_3; done;

L_2: while b and ~b_1' do S_0; S_2; S_3; done;

done

其中,b 与 b_1 是布尔表达式,S_1 与 S_2 是赋值语句序列,并且 $b_1'=WP(S_0,b_1)$,则循环 3.1.3 停机当且仅当循环 3.1.4 停机.

证明: 可由定理 3.2 与引理 3.3 得到. □

下面具体讨论形如 3.1.4 的循环程序复杂度上界的计算问题.如前所述,循环 3.1.4 可以写成 $(L_1^*\,|\,L_2^*)^*$ 的形式,所以可得形如 3.1.4 的循环程序停机当且仅当 $(L_1^*\,|\,L_2^*)^*$ 停机.设外部循环可以迭代 k 次(k>=1),两个内循环在外循环的第 k 次迭代时分别可以迭代 i_k(i_k>=0)和 j_k(i_k>=0)次,因此循环 3.1.4 的迭代模型可以表示为序列 $(L_1^{i_k}\,|\,L_2^{j_k})^k$.另外,由于循环变量的初始值不同导致循环的执行入口可以有两个:L_1 或 L_2,因而循环的迭代模型可以表示为

$$i_1j_1i_2j_2\ldots i_kj_k \text{ 或 } j_1i_2j_2\ldots i_kj_k\ (i_1<>0,i_k>=0,j_k>=0).$$

定理 3.5. 循环 3.1.3 的程序复杂度上界为 $\sum_{k=1}^{n}(i_k+j_k)$,n 为循环 3.1.4 中外循环的迭代次数.

证明:若循环 3.1.4 的迭代序列为 $i_1j_1i_2j_2\ldots i_kj_k$,则所有内循环的迭代次数之和应为 $\sum_{k=1}^{n}(i_k+j_k)$; 如果迭代序列

为 $j_1i_2j_2\ldots i_kj_k$,所有内循环的迭代次数之和就应为 $j_1+\sum_{k=2}^{n}(i_k+j_k)$.令 i_1=0,则可重写为 $\sum_{k=1}^{n}(i_k+j_k)$.而循环 3.1.3 每次迭代每个分支只运行一次,并且循环 3.1.3 与循环 3.1.4 有相同的执行路径,所以循环 3.1.3 的迭代次数相当于所有分支的迭代次数之和,也就是循环 3.1.4 中所有内循环的迭代次数之和,即 $\sum_{k=1}^{n}(i_k+j_k)$. □

定理 3.6. 循环 3.1.3 停机当且仅当所有的 i_k 与 j_k 都是有穷非负整数并且存在一个正整数 m>1,当 k>=m 时 i_k=j_k=0.

证明:当循环 3.1.3 停机时,由推论 3.3 易知循环 3.1.4 停机,故迭代序列 $i_1j_1i_2j_2\ldots i_kj_k$ 肯定会终止.所以,i_k 与 j_k 均为有限整数并且必然存在一个正整数 m(m>1),使得 k>=m 时 i_k=j_k=0.

如果 i_k 与 j_k 为有穷整数,并且存在一个正整数 m>1,当 k>=m 时 i_k=j_k=0.此时迭代序列 $i_1j_1i_2j_2\ldots i_kj_k$ 可以化简为 $i_1j_1i_2j_2\ldots i_mj_m$,从而 $\sum_{k=1}^{m}(i_k+j_k)$ 为有限整数.故由定理 3.4 易知循环 3.1.3 的复杂度上界存在,循环 3.1.3 停机. □

推论 3.7. 循环 3.1.3 停机,当且仅当存在两个连续的 i_m=j_n=0(m=n 或 m=n+1).

证明:如果存在两个连续的 i_m=j_n=0,就是指两个内循环的入口条件连续不能得到满足,即当 k>m 时 i_k=j_k=0,从而可由定理 3.6 直接证明. □

因此,依据定理 3.2~推论 3.7,带条件分支语句的循环的程序复杂度上界可按下面的步骤来计算.

算法 2. 带条件分支语句的循环的程序复杂度上界计算算法.

输入:带条件分支语句的 P*-solvable 循环,X={x_1,…,x_m}为循环变量;

输出:循环的复杂度上界.

步骤 1. 提取每个内循环的差分方程,{$x_{1,1}$,…,$x_{m,1}$}表示{x_1,…,x_m}在内循环 L_1 中的取值,{$x_{1,2}$,…,$x_{m,2}$}表示{x_1,…,x_m}在内循环 L_2 中的取值,k 表示外循环的第 k 次迭代;

$$\text{内循环 } L_1:\begin{cases}x_{1,1}^{k}[i_k]=p_{1,1}(i_k)\theta_{1,1}^{i_k}+\ldots+p_{1,s}(i_k)\theta_{1,s}^{i_k}\\ x_{2,1}^{k}[i_k]=p_{2,1}(i_k)\theta_{1,1}^{i_k}+\ldots+p_{2,s}(i_k)\theta_{1,s}^{i_k}\\ \ldots\\ x_{2,1}^{k}[i_k]=p_{m,1}(i_k)\theta_{1,1}^{i_k}+\ldots+p_{m,s}(i_k)\theta_{1,s}^{i_k}\end{cases},i_k\in N,$$

$$\text{内循环 } L_2:\begin{cases}x_{1,2}^{k}[j_k]=p'_{1,1}(j_k)\theta_{2,1}^{j_k}+\ldots+p'_{1,s}(j_k)\theta_{2,s}^{j_k}\\ x_{2,2}^{k}[j_k]=p'_{2,1}(j_k)\theta_{2,1}^{j_k}+\ldots+p'_{2,s}(j_k)\theta_{2,s}^{j_k}\\ \ldots\\ x_{m,2}^{k}[j_k]=p'_{m,1}(j_k)\theta_{2,1}^{j_k}+\ldots+p'_{m,s}(j_k)\theta_{2,s}^{j_k}\end{cases},j_k\in N.$$

步骤 2. 合并由 $L1$ 和 $L2$ 得到的差分方程,得到{$x_1[k]$,…,$x_m[k]$}的差分方程表示.其中,{$x_1[k]$,…,$x_m[k]$}表示循环变量{x_1,…,x_m}在外层循环迭代 k 次的值;

步骤 3. 调用算法 1 分别计算内循环 L_1 和 L_2 的复杂度上界,得到 i_k 与 j_k.i_k 与 j_k 为{$x_1[k–1]$,…,$x_m[k–1]$}表示的函数;

步骤 4. 求外层循环关于{$x_1[k]$,…,$x_m[k]$}的差分方程的闭合形式解:

$$\begin{cases}x_1[k]=p_{1,1}(k)\theta_1^k+\ldots+p_{1,s}(k)\theta_s^k\\ x_2[k]=p_{2,1}(k)\theta_1^k+\ldots+p_{2,s}(k)\theta_s^k\\ \ldots\\ x_m[k]=p_{m,1}(k)\theta_1^k+\ldots+p_{m,s}(k)\theta_s^k\end{cases},n\in N.$$

步骤 5. 根据{x_1,…,x_m}差分方程的闭合形式解以及外层循环卫式条件构造最优化问题,调用算法 1 计算外层循环的复杂度上界,表示为 $bound(k)$;

步骤 6. 如果 $bound(k)$存在,利用公式 $\sum_{k=1}^{bound(k)}(i_k + j_k)$ 计算循环 3.1.3 的复杂度上界.

下面通过例 3.8 来说明算法 2 的具体步骤.

例 3.8:考虑如下程序[12]:

```
assume(n>0 && m>0);
v1:=n; v2:=0;
while (v1>0) do
   if (v2<m) then
      v2++; v1−−;
   else   v2:=0;
   endif;
done
```

```
assume(n>0 && m>0);
v1:=n; v2:=0;
while (v1>0) do
L1: while (v1>0) && (v2<m) do
      v2++; v1−−;
   done;
L2: while (v1>0) && (v2>=m) do
      v2:=0;
   done;
done
```

循环的迭代模型为 $i_1j_1i_2j_2\ldots i_kj_k$,$k$ 为第 2 个循环的外层循环的复杂度上界.

步骤 1. 提取内层循环的差分通项表达式:

内循环 L_1: $\begin{cases} v1_1^k[i_k] = v1_1^k[0] - i_k \\ v2_1^k[i_k] = v2_1^k[0] + i_k \end{cases}, i_k \in N$,

内循环 L_2: $v2_2^k[j_k] = 0, j_k \in N$;

步骤 2. 合并两个差分方程得到 $\begin{cases} v1[k] = v1_1^k[0] - i_k \\ v2[k] = 0 \end{cases}$,其中, $v1_1^k[0] = v1[k-1]$,因此有 $\begin{cases} v1[k] = v1[k-1] - i_k \\ v2[k] = 0 \end{cases}$;

步骤 3. 应用算法 1 计算内循环的程序复杂度上界,得到 $i_k = \begin{cases} v1_1^k[0], & v1_1^k[0] < m \\ m, & \text{否则} \end{cases}$, $j_k = 1.v1_1^k[0]$ 为 $v1$ 在外循环第 k 次迭代时的初始值;

步骤 4. 求得外层循环的差分闭合形式解 $v1[k]=v1[0]–mk$,代入变量初始值得到 $v1[k]=n–mk$;

步骤 5. 考虑外层循环的循环卫式条件 $v1>0$,将其取反得到 $v1\leqslant 0$,求解最优化问题$(k, v1=n–mk, v1\leqslant 0)$;对于目标变量 k,计算满足约束系统$(v1=n–mk, v1\leqslant 0)$的最优解,得到外层循环的复杂度上界为

$$bound(k) = \frac{n}{m};$$

步骤 6. $\sum_{k=1}^{n/m}(i_k + j_k) = \sum_{k=1}^{n/m}(m+1) = n + \frac{n}{m}$,因此原循环的复杂度上界为 $n + \frac{n}{m}$.

3.2 含有k个分支的循环($k\geqslant 1$)

定理 3.2 可以扩展到含有 $k\geqslant 1$ 个条件分支语句的循环,并且每个分支都含有一个赋值语句序列.在这种情况下,通过反复应用定理 3.2 中的转换规则,就可以得到一个包含 k 个内循环的嵌套循环,并且每个内循环都只含有赋值语句序列,具体转换方法如下:

定理 3.9. 考虑如下的两个循环:

循环 3.2.1:

```
while b do
   if b1 then s1
      else if b2 then s2
      else if b3 then s3
      …
```

else if b_{k-1} then s_{k-1}
else s_k
done;

循环 3.2.2:

while b do
while b and b_1 do s_1 done;
while b and ~b_1 and b_2 do
s_2 done;
…
while b and ~b_1 and … and ~b_{k-1} do
s_k done;
done

其中,$b,b_1,\ldots,b_{k-1}$ 是布尔表达式,$s_1,\ldots,s_k$ 是赋值语句序列,则循环 3.2.1 停机当且仅当循环 3.2.2 停机.

证明:证明方法类似于定理 3.2 的证明,只需将内循环的个数从 2 扩展到 k 即可. □

类似于推论 3.4,基于引理 3.3 与定理 3.9,可以得到如下推论:

推论 3.10. 考虑循环 3.2.3 和循环 3.2.4 两个循环:

循环 3.2.3:

while b do
s_0;
if b_1 then s_1
else if b_2 then s_2
else if b_3 then s_3
...
else if b_{k-1} then s_{k-1}
else s_k
s_{k+1}
done;

循环 3.2.4:

while b do
while b and b_1' do s_0; s_1;s_{k+1} done;
while b and ~b_1' and b_2' do
s_0; s_2; s_{k+1}; done;
…
while b and ~b_1' and … and ~b_k' do
s_0; s_k; s_{k+1}; done;
done;

其中,$b,b_1,\ldots,b_{k-1}$ 是布尔表达式,$s_1,\ldots,s_k$ 是赋值语句序列,并且 $b_i' = wp(s_0,b_i)$,i=1,2,…,k;则循环 3.2.3 停机当且仅当循环 3.2.4 停机.

证明:本推论可以直接由定理 3.9 与引理 3.3 得到. □

通过运用定理 3.9 和推论 3.10,含有条件分支的 P*-solvable 循环就可以转化为只含有赋值语句的嵌套循环.因此,含 k 个条件分支的循环复杂度上界计算问题可以转化为对只含赋值语句的循环问题求解.

类似于第 3.1 节中所述,假设 $i_{1m},i_{2m},\ldots,i_{km}$ 为内循环 $L_1,L_2,\ldots,L_k$ 在外循环第 m 次迭代时迭代的次数,而外循

环的复杂度上界为 n.基于定理 3.5,可以得到以下推论:

推论 3.11. 循环 3.2.3 的复杂度上界为 $\sum_{m=1}^{n}(i_{1m}+i_{2m}+...+i_{km})$.

证明:类似于定理 3.5. □

因此,形如循环 3.2.3 的循环可以通过类似于算法 2 的方法来处理得到其复杂度上界.

4 实验结果与分析

基于 Mathematica 6.0 和本文算法 1 和算法 2,我们设计并实现了一个计算 P*-solvable 循环的程序复杂度上界的原型系统,实验环境为 Intel Core2 T5500 1.66G,2G 内存.

实验借鉴了文献[15]中 Aligator 的程序语法分析方法,基于数学工具 Mathematica 6.0 求解差分方程和最优化问题.在数学工具 Mathematica 6.0 中,差分方程通过调用命令"*RSolve*[*eqn*,*a*[*n*],*n*]"来求解其闭合形式解,对于线性差分方程,*RSolve* 可以直接求解闭合形式解;对于最优化问题,通过调用命令 *Minimize*[{*r*,*cons*},{*r*,*x*,*y*,…}]来求解满足约束 *cons* 的关于 *r* 的最优化问题的最优解.

表 1 给出了一些测试程序以及它们的程序复杂度上界的计算结果,并给出了计算所需的时间;由于符号化计算的原因,表中给出的结果可能不是正整数,可采用向上取整得到实际的正整数上界.

Table 1 Experimental results of some Kovács' examples

表 1 部分 Kovács 实例程序的计算结果

Program fragments	Comlexity bounds	Time (s)

k:=0; *j*:=1; *m*:=1;
while (*m*<=*a*) do
k:=*k*+1;
j:=*j*+2;
m:=*m*+*j*;
done; (from Ref.[15,19])
	$\begin{cases}0, & a<1\\ \sqrt{a}, & a\geqslant 1\end{cases}$	0.078

x:=*a*/2; *r*:=0;
while (*x*>*r*) do
x:=*x*−*r*;
r:=*r*+1;
done; (from Ref.[15,20])
	$\begin{cases}0, & a\leqslant 0\\ -\frac{1}{2}+\frac{1}{2}\sqrt{1+4a}, & \text{otherwise}\end{cases}$	0.078

x:=*a*; *r*:=1; *s*:=13/4;
while (*x*−*s*>0) do
x:=*x*−*s*;
s:=*s*+6**r*+3;
r:=*r*+1;
done; (from Ref.[15])
	$\begin{cases}0, & a\leqslant \frac{13}{4}\\ Root[13-4a+27n+18n^2+4n^3,1], & \text{otherwise}\end{cases}$,
	Root[*f*,*k*]is the k^{th} root of *f*[*x*]=0
		0.156

为了表明本文的方法能够计算出 P*-solvable 循环更精确的复杂度上界,与 Gulwani 的实验结果[12,21]进行了比较,表 2 列出了实验结果(由于文献[12,21]未给出计算时间的结果,故算法计算时间无法作比较).对于第 1 个例子和最后一个例子,本文的计算结果明显比 Gulwani 的计算结果更加精确.对于第 1 个例子,为了计算出 $\sqrt{2m}+\max(0,-2b)+1$ 的结果,Gulwani 使用了更复杂的数值抽象域[21].

本文针对 P*-solvable 循环,基于差分方程和最优化问题求解方法给出了一种复杂度上界的精确计算方法.实验结果表明,本文的方法可以有效地对只含有赋值语句和含有条件分支语句的 P*-solvable 循环计算其符号复杂度上界;对于表 1 和表 2 中的程序,本文算法所耗费的时间基本都在 0.5s 以下.文献[13]指出,大规模程序中的循环通过程序切片方法去除与循环控制变量不相关的程序代码后,能够极大地降低循环的复杂度.因此,结合程序切片方法,本文的算法也能够应用于大规模程序中循环的复杂度上界计算.

Table 2 Results compared with Gulwani

表 2 与 Gulwani 部分实例的比较

Program fragments	Results of this paper	Results of Gulwani

x:=0; *y*:=*b*; *k*:=0;
while (*x*<*m*) do
k:=*k*+1;
y:=*y*+1;
x:=*x*+*y*;
done; (from Ref.[21])
	$\frac{1}{2}\left(\sqrt{1+4b+4b^2+8m}-1-2b\right)$
	0.094s
		$\sqrt{2m}+\max(0,-2b)+1$

assume(*n*>0 && *m*>0);
*v*1:=*n*; *v*2:=0;
while (*v*1>0) do
if (*v*2<*m*) then
*v*2++; *v*1−−;
else *v*2:=0;
endif;
done; (from Ref.[12])
	$n+\frac{n}{m}$
	0.402s
		$n+\frac{n}{m}$

assume(0<*m*<*n*);
x:=0; *y*:=0;
while (*x*<*n*) do
if (*y*<*m*) then
y++;
else
y:=0; *x*++;
endif;
done; (from Ref.[12])
	n+*nm*
	0.526s
		The result is *nm* in Ref.[12],
		but revised to *n*+*nm* in Ref.[15].

assume(0<*m*<*n*);
x:=*n*;
while (*x*>0) do
if (*x*<*m*) then
x−−;
else *x*:=*x*−*m*;
endif;
done; (from Ref.[12])
	$\frac{n}{m}+m-1$
	0.414s
		$\frac{n}{m}+m$

5 结 论

循环的停机性验证是一个不可判定的问题,但是研究发现,对于某些特定类型的循环可以给出有效的停机性验证方法.本文分别针对含有赋值语句的循环和带条件分支的 P*-solvable 循环提出了符号化复杂度上界计算方法.与其他工作相比,本文的方法能够计算得到更精确的符号化复杂度上界,实验结果表明了本文方法的有效性.

根据 P*-solvable 循环的定义,对于只含赋值语句的循环,只要循环是差分可解的,本文的方法就能够对其计算符号化复杂度上界.而对于多分支的 P*-solvable 循环,本文的方法现在还只能处理每一个内循环执行次数都能够明确计算的循环.在下一步工作中,我们将结合差分求解和文献[11]中的有穷差分树来拓宽多分支循环的停机性验证,首先建立循环的有穷差分树,然后根据有穷差分树构建循环的约束系统,通过约束系统的解构造 polyranking 函数[22]验证程序停机性.另外,我们还将结合程序切片方法应用本文的方法对大规模程序进行分析和验证.

References:

[1] Tiwari A. Termination of linear programs. In: Proc. of the CAV 2004. LNCS 3114, 2004. 70−82.

[2] Colón MA, Sipma HB. Synthesis of linear ranking functions. In: Proc. of the TACAS 2001. LNCS 2031, 2001. 67−81.

[3] Colón MA, Sipma HB. Practical methods for proving program termination. In: Proc. of the CAV 2002. SpringerLink 2404, 2002. 227−240. [doi: 10.1007/3-540-45657-0_36]

[4] Podelski A, Rybalchenko A. Transition predicate abstraction and fair termination. In: Proc. of the POPL 2005. ACM, 2005. 132−144. [doi: 10.1145/1047659.1040317]

[5] Podelski A, Rybalchenko A. A complete method for the synthesis of linear ranking functions. In: Proc. of the VMCAI 2004. SpringerLink 2937, 2005. 465−486. [doi: 10.1007/978-3-540-24622-0_20]

[6] Cook B, Gulwani S, Lev-Ami T, Rybalchenko A, Sagiv M. Proving conditional termination. In: Proc. of the CAV 2008. LNCS 5123, 2008. 328−340. [doi: 10.1007/978-3-540-70545-1_32]

[7] Bradley AR, Manna Z, Sipma HB. Termination of polynomial programs. In: Proc. of the VMCAI 2005. LNCS 3385, 2005. 113−129. [doi: 10.1007/978-3-540-30579-8_8]

[8] Lee CS, Jones ND, Ben-Amram AM. The size-change principle for program termination. In: Proc. of the POPL 2001. ACM SIGPLAN Notices, Vol.36, 2001. 81−92. [doi: 10.1145/360204.360210]

[9] Zantema H. Classifying termination of term rewriting. Technical Report, RUU-CS-91-42, Netherlands: Utrecht University, 1991.

[10] BenCherifa A, Lescanne P. Termination of rewriting systems by polynomial interpretations and its implementation. SCP, 1987,9(2): 137−159. [doi: 10.1016/0167-6423(87)90030-X]

[11] Jouannaud JP, Rubio A. The higher-order recursive path ordering. In: Proc. of the 14th IEEE Symp. on Logic in Computer Science. 1999. 402−411.

[12] Gulwani S, Jain S, Koskinen E. Control-Flow refinement and progress invariants for bound analysis. In: Proc. of the PLDI 2009. ACM, 2009.375−385. [doi: 10.1145/1542476.1542518]

[13] Gulwani S. SPEED: Symbolic complexity bound analysis. Invited Talk Paper. In: Proc. of the CAV 2009. LNCS 5643, 2009. 51−62. [doi: 10.1007/978-3-642-02658-4_7]

[14] Gulwani S, Mehra KK, Chilimbi T. SPEED: Precise and efficient static estimation of program computational complexity. In: Proc. of the POPL 2009. ACM, 2009. 127−139.

[15] Kovács L. Automated invariant generation by algebraic techniques for imperative program verification in theorema [Ph.D. Thesis]. RISC-Linz: Johannes Kepler University, 2007.

[16] Kovács L. A complete invariant generation approach for P-solvable loops. In: Proc. of the PSI 2009. SpringerLink, 2009. 184−194.

[17] Henzinger TA, Hottelier T, Kovács L. Valigator: A verification tool with bound and invariant generation. In: Proc. of the LPAR 2008. LNCS 5330, 2008. 333−342.

[18] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969,12(10):576−580. [doi: 10.1145/363235.363259]

[19] Kirchner M. Program verification with the mathematical software system theorema. Technical Report, RISC-Linz: Diplomaarbeit, 1999. 99−116.

[20] Rodríguez-Carbonell E, Kapur D. Generating all polynomial invariants in simple loops. Journal of Symbolic Computation, 2007, 42(4):443−476. [doi: 10.1016/j.jsc.2007.01.002]

[21] Gulavani BS, Gulwani S. A numerical abstract domain based on expression abstraction and max operator with application in timing analysis. In: Proc. of the CAV 2008. Springer-Verlag, 2008. 370−384. [doi: 10.1007/978-3-540-70545-1_35]

[22] Bradley AR, Manna Z, Sipma HB. The polyranking principle. In: Proc. of the Int'l Colloquium on Automata, Languages and Programming (ICALP). LNCS 3580, Springer-Verlag, 2005. 1349−1361.

邢建英(1980－),男,山东栖霞人,博士生,主要研究领域为程序验证.

李梦君(1975－),男,博士,副教授,主要研究领域为形式化方法与技术,信息安全技术.

李舟军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为安全协议的形式化分析,进程代数理论,数据挖掘.