文章编号: 1000-5862(2014)04-0378-05

一类单元赋值语句型循环不变式的开发方法研究

杨黄磊,薛锦云*
(江西师范大学江西省高性能计算技术重点实验室,江西 南昌　330022)

摘要: 依据现有循环不变式的定义和开发策略,阐述了一类单元赋值语句型循环不变式开发方法,同时使用 Dijkstra 最弱前置谓词方法确认了循环不变式的正确性. 最后通过典型实例来说明该方法的应用.

关键词: 单元赋值语句; 循环不变式; 开发策略; 最弱前置谓词方法
中图分类号: TP 391　　**文献标志码**: A

0　引言

循环不变式揭示了循环程序的本质特征,循环不变式对于循环程序的理解、软件验证、形式化程序设计和自动程序设计至关重要,甚至部分人认为不知道循环不变式就不可能理解循环[1].

循环不变式的开发分为手工开发和自动开发. 早期采取手工的方式,随着理论和技术发展,自动方式被广泛应用于循环不变式的开发. 而自动方式又被分为静态方法和动态方法. 静态方法仅分析程序的源代码本身,其中基于抽象解释的方法和基于约束的方法是应用最广的系统性方法. 抽象解释是在抽象域上的符号程序执行,其抽象域综合描述了循环迭代的语义. P. Cousot 和 R. Cousot[2] 将抽象解释更新并扩展到处理现代程序语义. 基于抽象解释的不变式生成技术[3-4] 可以构造多种类型的线性和多项式循环不变式. 基于约束的技术依赖于非简单的数学域(例如多项式或者凸面多面体)上的复杂决策过程来精确地表达带有一定模板特性的循环语义. 基于约束技术同样可以构造出线性和多项式循环不变式[7]. 近 10 年动态方法开始应用于循环不变式的开发,其通过选取充足多样的输入,然后在执行程序的过程中探测程序的不变式特性. M. D. Ernst 等[8] 的 Daikon 技术表明动态方法在开发循环不变式具有实用性,并且带动了许多后续工作[9-13]. B. Meyer等[1] 采用的 domain theory,用各个应用领域内的具体特征函数和谓词高度抽象地表达包括后置断言和循环不变式在内的断言,系统地分析确认和分类计算机科学的许多领域的典型算法的循环不变式,指出循环不变式呈现与领域相关的模式特征,以期利用这些模式特征来开发循环不变式.

静态方法是稳健的并且对于其能开发出的一类不变式是完备的. 利用不变式代表数学域的可判定性、可保证稳健性和完备性,因而这些技术处理新的不变式特性时会受到不可判定性的制约. 动态方法把候选不变式特性中不违背任何一次程序执行的不变式特性被当做可能的不变式,因此这是不稳健的,只是一种启发式猜测. 动态不变式开发方法在正确实现时能较好地发挥作用,但对于许多循环不变式的开发仍存在如何获取充足多样的测试集的问题. 当前动态方法较少应用于开发循环不变式,但是其主要被应用于推导前后置断言或者中间断言中.

薛锦云[14] 提出的按循环变量的元数分类的思路来研究循环不变式,是一种静态方法. 单元赋值语句指赋值符号右边仅出现 1 个与赋值符号左边相同的循环变量且不出现其他的变量(循环控制变量除外)的赋值语句. 从而单元赋值语句型循环程序就指循环体里仅出现 1 条单元赋值语句且不出现其他多元赋值语句的循环程序.

1　循环不变式的新定义和新开发策略

这里的新定义和新的开发策略[14-15] 是由薛锦云提出,定义在循环程序段中,其值随着程序的执行不断改变的变量为循环变量. 循环不变式支配着这些循环变量的变化,反应了它们的变化规律. 因此,

收稿日期: 2014-04-09
基金项目: 国家自然科学基金重大国际合作项目(61020106009)和国家自然科学基金(61272075)资助项目.
通信作者: 薛锦云(1947-),男,江苏海门人,教授,博士生导师,主要从事形式化方法的研究.

薛锦云给出如下定义.

定义 1　给定循环语句 DO 和它的所有循环变量的集合 A,1 个反应 A 中每 1 个循环变量的变化规律且在循环体 S 执行前后均为真的谓词称为循环语句 DO 的循环不变式.

本文要研究的单元赋值语句型循环程序,按定义 A 集合里仅含 1 个循环变量,就是赋值语句符号左边出现的变量. 然后需要寻找反应该变量的变化规律,且在循环体 S 执行前后均为真的谓词,如此就能得出循环不变式.

设问题 P 的解由解序列由 $P_1, P_2, \cdots, P_n$ 构成,其中每个 $P_i(1 \leqslant i \leqslant n)$ 为其右边某个 P_k 子问题的解,P_n 为 P 的解. 把 P_i 和它前面的 1 个或者多个 P_j $(1 \leqslant j \leqslant i)$,关联起来的等式叫做问题求解序列的递推关系,简称为递推关系,用 $P_i = F(P_j)$ 表示,其中 P_j 为 P_i 子问题的序列. 寻找递推关系即把问题 P 的解表示成它的子问题的解的函数. 从而有了开发循环不变式的新策略.

策略 1　以循环程序正确性验证条件为基准,考察循环初始条件及循环结束所得的信息,分析程序所解问题的实际背景、数学性质和程序特征,通过归纳推理找出所有循环变量的变化规律,即为所求的循环不变式.

按照以上策略,分析单元赋值语句型问题的实际背景,数学性质和程序特征可得递推关系由单元赋值语句,结合初始条件和循环结束的信息,由归纳推理可以找到循环不变式.

2　2 种单元赋值语句型循环不变式开发方法

按照循环不变式新的开发策略,通过对大量单元赋值语句型循环程序的研究,发现单元赋值语句型循环程序各种各样,这里通过 2 个典型的形式来说明单元赋值语句型循环程序的开发规律. 下面分别叙述这 2 种形式的循环不变式的开发步骤及利用 Dijkstra 最弱前置谓词[16] 方法进行正确性证明,为了表述和证明的方便,程序按照 Dijkstra 卫士命令语言书写.

2.1　第 1 种循环程序$\{Q: i_n \geqslant i_0\}$

var S: real; var i: integer;

$S:=S(0)$; $i:=i_0$;

do $i<i_n \rightarrow S:=p^* S^r+q$; $i:=i+1$ od

$\{R: S=S(i_n-i_0)\}$

结合 $S=S(0)$ 和$(\forall i: i_0 \leqslant i \leqslant i_n-1: S(i-i_0+1)=p^*(S(i-i_0))^r+q)$,当 $i=i_n-1$ 时得出后置断言 $R: S=S(i_n-i_0))$,Q 为前置断言,S 为循环变量,i 为循环控制变量,$S(0)$、i_0、i_n 表示实数,单元赋值语句 $S:=p^* S^r+q$ 中 p、q、r 表示实数且 $p \neq 0$、$r \neq 0$. 循环不变式开发步骤为:

(i) 进入循环前,分析循环初始条件,得到循环变量 S 的初始值 $S(0)$ 及循环控制变量初始值 i_0;

(ii) 进入循环,首先找到循环控制条件 $i<i_n$. 结合第(i)步,可得循环不变式合取项 $S=S(i-i_0) \wedge i_0 \leqslant i \leqslant i_n$;

(iii) 然后找到 $S:=p^* S^r+q$,得到问题求解序列的递推关系 $S(i)=p^*(S(i-1))^r+q$;

(iv) 将第(ii)步和第(iii)步得到的结果合取即可得到循环不变式 I:

$$S=S(i-i_0) \wedge i_0 \leqslant i \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q].$$

循环不变式确认及程序正确性证明:

(i) $Q \Rightarrow \mathrm{WP}("S:=S(0), i:=i_0", I)$,循环开始前,$I$ 成立.

$$i_n \geqslant i_0 \Rightarrow ((S=S(i-i_0) \wedge i_0 \leqslant i \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q])_{i_0}^{i})_{S(0)}^{S} \equiv \{[S(i)=p^*(S(i-1))^r+q] \text{ 总为真且不参与文字替换}\},$$

$$i_n \geqslant i_0 \Rightarrow S(0)=S(0) \wedge i_0 \leqslant i_0 \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q] \equiv \text{true}.$$

(ii) $I \wedge \mathrm{Guard} \Rightarrow \mathrm{WP}("S=p^* S^r+q; i=i+1", I)$,Guard 表示"$i<i_n$"要保证循环每次执行后不变式成立,进行如下证明:

$$I \wedge \mathrm{Guard} \equiv S=S(i-i_0) \wedge i_0 \leqslant i \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q] \wedge i<i_n \equiv$$
$$S=S(i-i_0) \wedge i_0 \leqslant i<i_n \wedge [S(i)=p^*(S(i-1))^r+q].$$

$$\mathrm{WP}("S=p^* S^r+q; i=i+1", I) \equiv ((S=S(i-i_0) \wedge i_0 \leqslant i \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q])_{i+1}^{i})_{p^* S^r+q}^{S} \equiv \{\text{文字替换}\}.$$

$$p^* S^r+q=S(i+1-i_0) \wedge i_0 \leqslant i+1 \leqslant i_n \wedge [S(i)=p^*(S(i-1))^r+q] \equiv \{\text{利用 } [S(i)=p^*(S(i-1))^r+q] \text{ 对 } S(i+1-i_0) \text{ 等量替换}\}.$$

$$p^* S^r+q=p^*(S(i-i_0))^r+q \wedge i_0-1 \leqslant i<i_n \wedge [S(i)=p^*(S(i-1))^r+q] \equiv$$
$$S=S(i-i_0) \wedge i_0-1 \leqslant i<i_n \wedge [S(i)=p^*(S(i-1))^r+q].$$

$$I \wedge \mathrm{Guard} \Rightarrow \mathrm{WP}("S=p^* S^r+q; i=i+1", I) \equiv \{i_0 \leqslant i<i_n \Rightarrow i_0-1 \leqslant i<i_n\} \equiv \text{true}.$$

(iii) $I\wedge\neg$ Guard$\Rightarrow R$ 循环不变式和 Guard 的否定能得到后置断言成立.

$I\wedge\neg$ Guard$\equiv S=S(i_n-i_0)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i)=p^*(S(i-1))^r+q]\wedge i\geqslant i_n\equiv S=S(i_n-i_0)\wedge i=i_n\wedge [S(i)=p^*(S(i-1))^r+q]\equiv S=S(i_n-i_0)\wedge [S(i)=p^*(S(i-1))^r+q]\Rightarrow S=S(i_n-i_0)$.

因此,$(I\wedge\neg$ Guard$\Rightarrow R)\equiv$true.

(iv) 循环的终止性显然成立.

2.2 第 2 种循环程序

{Q: $i_n\geqslant i_0$}

var S: real; var i: integer;

$S:=S(0)$; $i:=i_n$;

do $i>i_0\rightarrow i:=i-1$; $S:=p^*\ S^r+a$ [i] od

{R: $S=S(i_n-i_0)$}.

结合 $S=S(0)$ 和 $(\forall i: i_0+1\leqslant i\leqslant i_n: S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]),当 $i=i_0+1$ 时得出后置断言 R: $S=S(i_n-i_0)$,开发步骤如下:

(i) 进入循环前,分析循环初始条件,得到循环变量 S 的初值 $S(0)$ 及循环控制变量的初值 i_n;

(ii) 进入循环,首先得到循环控制条件 $i>i_0$,结合第(i)步,可得循环不变式的合取项

$$S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n;$$

(iii) 然后得到 $S:=p^*\ S^r+a$ [i],可得问题求解序列的递推关系 $S(i_n-(i-1))=p^*(S(i_n-i))^r+a$ [i−1];

(iv) 将第(ii)步和第(iii)步得到的结果合取即可得循环不变式 I:

$S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i_n-(i-1))=p^*(S(i_n-i))^r+a$ [i−1]].

循环不变式确认及程序正确性证明:

(i) $Q\Rightarrow$WP("$S:=S(0)$, $i:=i_n$", I),循环开始前,I 成立.

$i_n\geqslant i_0\Rightarrow((S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]]$)^{i}_{i_n})^{S}_{S(0)}\equiv$ {$[S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] 总为真且不参与文字替换}.

$i_n\geqslant i_0\Rightarrow S(0)=S(0)\wedge i_0\leqslant i_n\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] ≡true.

(ii) $I\wedge$ Guard$\Rightarrow$WP("$i=i-1$; $S=p^*\ S^r+a$ [i]", I), Guard 表示"$i>i_0$"要保证循环每次执行后不变式成立,进行以下证明:

$I\wedge$Guard$\equiv S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] $\wedge i>i_0\equiv S=S(i_n-i)\wedge i_0<i\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]].

WP("$i=i-1$; $S=p^*\ S^r+a$ [i]", I) $\equiv$ (($S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]])$^{S}_{p^*\ S^r+a\ [i]}$)$^{i}_{i-1}\equiv$ {文字替换}.

$p^*\ S^r+a$ [i−1] $=S(i_n-i+1)\wedge i_0+1\leqslant i\leqslant i_n+1\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] ≡ {利用 $[S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] 对 $S(i_n-i+1)$ 等量替换}.

$p^*\ S^r+a$ [i−1] $=p^*(S(i_n-i))^r+a$ [i−1] $\wedge i_0+1\leqslant i\leqslant i_n+1\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] $\equiv S=S(i_n-i)\wedge i_0+1\leqslant i\leqslant i_n+1\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]].

$I\wedge$Guard$\Rightarrow$WP("$i=i-1$; $S=p^*\ S^r+a$ [i]", I) ≡ {$i_0<i\leqslant i_n\Rightarrow i_0+1\leqslant i\leqslant i_n+1$} ≡true.

(iii) $I\wedge\neg$ Guard$\Rightarrow R$ 循环不变式和 Guard 的否定能得到后置断言成立.

$I\wedge\neg$ Guard$\equiv S=S(i_n-i)\wedge i_0\leqslant i\leqslant i_n\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] $\wedge i\leqslant i_0\equiv S=S(i_n-i)\wedge i=i_0\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] $\equiv S=S(i_n-i_0)\wedge [S(i_n-i+1)=p^*(S(i_n-i))^r+a$ [i−1]] $\Rightarrow S=S(i_n-i_0)$.

因此,$I\wedge\neg$ Guard$\Rightarrow R\equiv$true.

(iv) 循环的终止性显然成立.

3 应用举例

本文开发的循环不变式具有一定的普遍性,关键就是对单元赋值语句进行了抽象,当参数 p、q、r 取不同的具体值时,就可以得到许多该类型的不同循环程序的循环不变式. 现在列举 3 例典型程序说明方法的应用.

例 1 猴子吃桃问题: 有一只猴子第 1 天摘下若干个桃子,当即吃掉了一半,又多吃了 1 个; 第 2 天将剩下的桃子吃掉一半,有多吃了 1 个; 按照这样的吃法,每天都吃前一天剩下的桃子的一半,又多吃 1 个. 到了第 10 天,就只剩下 1 个桃子,编程实现这只猴子第 1 天共摘下了多少个桃子?

{Q: $10\geqslant 1$}

var i: integer; var s: integer;

$s:=1$; $i:=1$;

do $i<10\rightarrow s:=2^*\ s+2$; $i:=i+1$ od

{R: $s=s(9)$}.

通过观察,它是第 1 种情形的具体化,这里 $p=2, q=2, r=1$.

(i) s 和 i 的初值都为 1;

(ii) $i<10$,循环不变式的合取项为

$$s=s(i-1) \wedge 1\leqslant i\leqslant 10;$$

(iii) $s:=2*s+2$,问题求解序列的递推关系: $s(i)=2*s(i-1)+2$;

(iv) 将(ii)和(iii)的结果合取得到循环不变式:

$s=s(i-1) \wedge 1\leqslant i\leqslant 10 \wedge [s(i)=2*s(i-1)+2]$.

例 2　b 进制数转化为十进制数. 给定任意 b 进制及其指数位置数值数组,将其转化为十进制数. 例如,5 进制及其指数位置数值数组 $\langle 3,2,0,1\rangle$,表示十进制数 $138=3*5^0+2*5^1+0*5^2+1*5^3$. 下面的算法程序简单直接,很容易想到,其本质上可以用秦九韶算法求解,它将 $n-1$ 次多项式的值转化为求 $n-1$ 个 1 次多项式的值,执行效率可显著提高.

$\{Q: n\geqslant 0\}$

var i: integer; var sum: integer;

var a: array(0: $n-1$, integer);

$sum:=0; i:=0;$

do $i<n\rightarrow sum:=sum+a[i]*b^i; i:=i+1$

od

$\{R: s=s(n)\}$.

这里单元赋值语句形式类似于第 1 种情形,开发步骤相似,其中 n、b 表示整数:

(i) sum 和 i 的初值都为 0;

(ii) $i<n$,循环不变式的合取项为

$$sum=sum(i) \wedge 0\leqslant i\leqslant n;$$

(iii) $sum:=sum+a[i]*b^i$,问题求解序列的递推关系 $sum(i)=sum(i-1)+a[i-1]*b^{i-1}$;

(iv) 将(ii)和(iii)的结果合取得循环不变式:

$sum=sum(i) \wedge 0\leqslant i\leqslant n \wedge [sum(i)=sum(i-1)+a[i-1]*b^{i-1}]$.

例 3　秦九韶算法,求 $a[n-1]*x^{n-1}+a[n-2]*x^{n-2}+\cdots+a[1]*x+a[0]$ 的值.

$\{Q: n\geqslant 0\}$

var i: integer; var s: integer;

var a: array(0: $n-1$, integer);

$s:=0; i:=n;$

do $i>0\rightarrow i:=i-1; s:=s*x+a[i]$ od

$\{R: s=s(n)\}$.

通过观察,它是第 2 种情形的具体化,这里 $p=x, r=1$, n 为整数.

(i) s 和 i 的初值分别为 0 和 n;

(ii) $i>0$,循环不变式的合取项为 $s=s(n-i) \wedge 0\leqslant i\leqslant n$;

(iii) $s=s*x+a[i]$,问题求解序列的递推关系为 $s(n-(i-1))=s(n-i)*x+a[i-1])$;

(iv) 将(ii)和(iii)的结果组装得循环不变式:

$s=s(n-i) \wedge 0\leqslant i\leqslant n \wedge [s(n-(i-1))=s(n-i)*x+a[i-1])]$.

4　总结和展望

从上面的介绍可看出,通过抽象概括出单元赋值语句和单元赋值语句型循环程序的概念,抓住循环不变式的本质特征,在新的开发策略的指导下,使得一类单元赋值语句型循环不变式的开发变得简单、有效. 相比其他的各种开发方法,可以避免盲目性,确保结果的可靠性. 新的开发策略提到的分析所解问题的实际背景、程序特征、数学性质和 Meyer 等的 domain theory 表示断言(尤其是后置断言和循环不变式),与利用的具体应用领域的特征函数和谓词思维角度不谋而合. 新的开发策略结合从循环不变式的本质特征和正确性验证条件得到的不变式稳健且有用,而 Meyer 等的用 domain theory 表达的循环不变式利用程序证明器保证正确理论上并不完全严格正确.

在实践上,大量开发稳健而有用的循环不变式仍是巨大的挑战,仅单元赋值语句型循环程序的循环不变式的构造就比较复杂,本文解决的是部分情形,进一步解决需要在数据组织结构和不变式的表示理论等方面努力. 还可进一步扩展到多元赋值语句型循环程序,这有待相关理论和技术的进步.

5　参考文献

[1] Furia C A, Meyer B, Velder S. Loop invariants: analysis, classification, andexamples [EB/OL]. [2012-10-16]. http://arxiv.org/abs/1211.4470.

[2] Patrick Cousot, Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints [C]. New York: ACM press, 1997: 238-252.

[3] Mine´ A. The octagon abstract domain higher-order and symbolic computation [J]. High-Order and Symbolic Computation, 2006, 19(1): 31-100.

[4] Patrick Cousot, Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program [EB/OL]. [2012-10-17]. http://www.citeulike.org/user/pganty/

article/120467.

[5] EnricRodr´1guez-Carbonell, Deepak Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation [J]. Sci Comput Program, 2007,64(1):54-75.

[6] EnricRodr´1guez-Carbonell, Deepak Kapur. An abstract interpretation approach for automatic generation of polynomial invariants [J]. SAS,2004,3148:280-295.

[7] Michael Colón, Sriram Sankaranarayanan, Henny Sipma. Linear invariant generationusing non-linear constraint solving [J]. CAV,2003,2725:420-433.

[8] Michael M D, Cockrell J, Griswold W G, et al. Dynamically discovering likely program invariants to support program evolution [J]. IEEE Transactions of Software Engineering, 2001,27(2):99-123.

[9] Jeff H Perkings, Michael D Ernst. Efficient incremental algorithms for dynamic detection of likely invariants [EB/OL]. [2013-05-16]. http://homes.cs.washington.edu/~mernst/pubs/invariants-incremental-fse 2004.pdf.

[10] Christoph Csallner, Nikolai Tillman, Yannis Smaragdakis. DySy: dynamicsymbolic execution for invariant inference [EB/OL]. [2013-07-12]. http://research.microsoft.com/apps/pubs/default.aspx?id=70511.

[11] Nadia Polikarpova, Ilinca Ciupa, Bertrand Meyer. A comparative study of programmer-written and automatically inferred contracts [EB/OL]. [2013-07-19]. http://se.inf.ethz.ch/people/polikarpova/publications/issta09.pdf.

[12] Yi Wei, Furia C A, Kazmin N, et al. Inferring better contracts [EB/OL]. [2013-08-16]. http://doi.ieee computersociety.org/10.1145/1985793.1985820.

[13] Thanhvu Nguyen, Deepak Kapur, Westley Weimer, et al. Using dynamic analysis to discover polynomial and array invariants [EB/OL]. [2013-06-17]. http://en.wikipedia.org/wiki/International_conference_on_Software_Engineering.

[14] Xue Jinyun. New concept of Loop Invariant and its application [EB/OL]. [2013-03-19]. http://link.springer.com/chapter/10.1007%2F11808107_1.

[15] Xue Jinyun. Two new strategies for developing loop invariants and their applications [J]. Journal of Computer Science and Technology,1993,8(2):147-154.

[16] Xue Jinyun. A unified approach for developing efficient algorithmic programs [J]. Journal of Computer Science and Technology,1997,12(4):314-329.

[17] Gries D. The science of programming [M]. New York: Springer Verlag,1981.

The Research on Methods of Developing a Class of Loop Invariants of Single-Variable-Assignment Type

YANG Huang-lei, XUE Jin-yun*

(Jiangxi Provincial Key Laboratory for High Performance Computing Technology, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: According to definition of loop invariants and strategy for developing loop invatiants proposed, methods of developing a class of loop invariants of single-variable-assignment type and confirmthe correct of loop invariants by using Dijkstra's weakest pre-condition method hans been elaborated. Finally, some typical examples to illustrate the application of the methods has been listed.

Key words: single-variable assignment; loop invariants; strategy for developing; the weakest pre-condition method

(责任编辑: 冉小晓)