CILinear:一个线性不变式自动构造工具

邢建英[1] 李梦君[1] 李舟军[2]
(国防科技大学计算机学院 长沙 410073)[1] (北京航空航天大学计算学院 北京 100083)[2]

摘 要 构造不变式是程序验证的重要组成部分,而开源工具 Interproc 能对简单的程序设计语言构造线性不变式。基于 Interproc 和 C 程序编译工具 CIL,针对简化的 C 程序设计并实现了自动构造数值型程序变量线性不变式的工具 CILinear,并与 Interproc 进行了比较。实验表明 CILinear 能有效地构造线性不变式,并且比 Interproc 支持的语法更多。通过实例讨论了 CILinear 在程序验证中的实际应用。

关键词 线性不变式,程序验证,数值变量,抽象域,超图
中图法分类号 TP301 **文献标识码** A

CILinear:An Automated Generation Tool of Linear Invariant

XING Jian-ying[1] LI Meng-jun[1] LI Zhou-jun[2]
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)[1]
(School of Computer Science & Engineering, Beihang University, Beijing 100083, China)[2]

Abstract Constructing program invariants is an important part of program verification and Interproc is an open-source tool capable of constructing linear invariant of a simple language. This paper designed and implemented a tool CILinear for automatic generation of linear invariant among numeric variables of simplified C programs based on Interproc and C compiler tool, and it is showed that CILinear can construct linear invariant effectively and deal with more syntax units. The application of CILinear in program verification was also discussed by program codes.

Keywords Linear invariant, Program verification, Numeric variable, Abstract domain, Hypergraph

1 引言

C 程序设计语言对语法的限定不严格,程序设计的灵活性大、自由度大,程序设计者可以直接对程序执行所需要的资源进行控制(比如内存分配与释放)。但由于 C 程序不进行严格的类型检查和越界检查,导致 C 程序容易出现缓冲区溢出、空指针引用等运行时错误。这些运行时错误很难被检测出来,C 程序的分析验证是当前的研究热点之一。

抽象解释(abstract interpretation)理论[1]为计算机科学中的不可判定问题和复杂问题的逼近求解提供了系统性构造方法和有效算法。抽象解释(abstract interpretation)理论是 Patrick Cousot 和 Radhia Cousot 于 1977 年提出的构造和逼近(approximation)程序语义的理论[2]。程序的抽象解释就是指使用另一个抽象对象域上的计算抽象逼近程序指称的对象域(具体对象域)上的计算,使得程序抽象执行的结果能够反映出程序真实运行时的信息。

目前,运用抽象解释理论来进行程序验证的研究是一个热点。在运用抽象解释理论对 C 程序进行分析的研究工作中,Patrick Cousot 等人的静态分析工具 ASTREE 是当中的杰出代表[3-5],针对用 C 语言编写的运算密集型的嵌入式安全攸关系统,通过 ASTREE 来验证其是否存在运行时错误。

程序不变式是程序变量之间满足的并且不随程序状态迁移发生变化的关系。不变式对于程序正确性验证至关重要,不变式的自动构造直接对程序的安全性、活性等性质的验证提供有力的支持。因此,构造程序不变式是大规模软件自动验证的重要组成部分,通过构造程序不变式来计算程序的不动点语义,可得到某个程序点程序变量之间满足的关系。一般来说,线性程序不变式的计算精度低,但是它的计算效率高,非线性程序不变式的计算精度高,但是它的计算效率低。为了支持大规模软件的高效验证,在满足性质验证的前提下,应尽量构造线性程序不变式,减少构造非线性程序不变式。

目前构造不变式的比较突出的工作基本都是基于抽象解释理论的,此类方法的主要思想就是对程序执行简化的符号化过程,使其到达一个断言并且该断言随程序的执行不再发生变化,该断言就是所要得到的不变式。构造线性不变式的研究工作主要有:Patrick Cousot 等人提出的基于区间抽象域[6]、基于八边形抽象域[7]、基于凸多面体抽象域[8]等数值抽象域和抽象解释理论的方法,Sriram Sankaranarayanan 等人提出的基于不变式参数化模板和约束求解的线性不变式构造方法[9]等。

研究发现,Gaël Lalire, Mathias Argoud 与 Bertrand Jeannet 开发的开源工具 Interproc[10]是一个对自定义的简单程序

到稿日期:2010-01-15 返修日期:2010-04-03 本文受国家自然科学基金(60703075,90718017),国家教育部博士点基金(20070006055)资助。
邢建英(1980—),男,博士生,主要研究方向为程序验证,E-mail:xjy.figgo@gmail.com;李梦君(1975—),男,博士,讲师,主要研究方向为形式化方法与技术、信息安全技术;李舟军(1963—),男,博士,教授,主要研究方向为安全协议的形式化分析、进程代数理论、数据挖掘等。

设计语言(simple language)进行过程间分析的工具,通过调用 APRON[11],基于区间抽象域、八边形抽象域、凸多面体抽象域等数值抽象域,它能够计算出程序中数值变量间的线性不变式。但它只能针对自己定义的简单语言来进行分析。进一步分析发现,Interproc 不能分析包含多文件的程序,不支持 C 语言中的 switch…case、for 循环、宏定义等语法,不能够满足验证 C 语言源程序的需求。

为了对 C 程序进行验证,以获得可供使用的针对 C 程序的线性不变式生成工具,本文根据 Interproc 中针对简单程序语言计算线性不变式的方法,基于 CIL 实现了求解 C 程序线性不变式的工具 CILinear。实验表明其可以有效地对只含有数值变量的 C 程序计算线性不变式。

本文第 2 节将给出面向 C 程序的线性不变式构造方法;第 3 节结合具体实例,说明 CILinear 能够有效地生成 C 程序的线性不变式,并且比 Interproc 支持的语法更多;第 4 节给出 CILinear 在程序验证中的具体应用。

2 面向简单 C 程序的线性程序不变式的构造

C 语法的灵活性导致 C 程序难以进行处理,对 C 程序进行直接验证所面对的困难也是巨大的,如果把 C 程序转化为一种规范的中间表示,将大大方便对它们进行分析与验证。

本文中实现的线性不变式构造工具 CILinear,将首先选用合适的 C 编译工具对 C 程序进行预处理得到中间表示;然后对相对规范的中间表示进行分析,建立状态迁移图、语义方程(状态迁移图的一种方程形式的表示);最后在状态迁移图和语义方程的基础上调用数值抽象域进行求解计算,从而实现对 C 程序的数值变量的线性不变式构造。

具体各步骤之间的关系如图 1 所示。

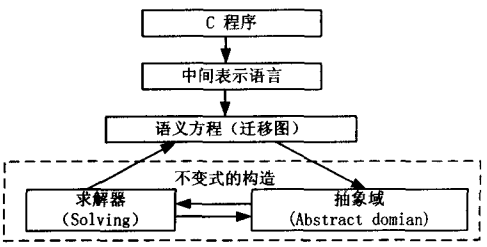

图 1 构造线性不变式的过程

2.1 中间表示方法的选择

研究人员一直致力于对 C 程序进行预处理得到中间表示的研究,并且已经有了很大的成效。但是其中一些工具(如微软的 AST[12]和贝尔的 ckit[13]),它们给出的表示过于抽象,不利于进行细节分析;而有些工具则是专门用于进行编译的,表示方式太底层,无法从转换后的程序中提取出可以识别的源程序,这类中间语言虽然不会产生二义性,但丢弃了关于类型、循环和其他高层构造的结构化信息,而且较难生成与源程序完全一致的描述;另外,如 SUIF[14]之类的中间表示虽然能部分满足需求,但不能支持 GCC 扩展,这使其不能分析带 GCC 扩展的程序(比如 Linux 的内核和驱动程序)。

G. Necula 等人开发的 CIL[15,16],其语法表示相对规范,便于进行分析和语法变换,并且语法表示接近于源代码,易于从转换后的程序中提取出可识别的源程序,CIL 简化了 C 程序的部分语法和表示形式,比如将所有的循环都处理成一种形式,所有的函数体都明确加上 return 语句,去掉像"->"之类的语法单位;将类型声明从代码中隔离出来,全部移到整个程序的开始处;所有变量的作用域都局限于当前所在的函数体内(采用变量重命名的方法)等。这些简化措施减少了操控 C 程序时的不确定情况,使得 C 程序更容易被分析和变换。虽然有很多 C 编译器在编译时也能达到类似目标,但它们无法像 CIL 这样用易理解的抽象语法来表示,以便后续的进一步分析处理。

选择 CIL 作为中间语言的另一个重要原因是,CIL 的中间表示与 Interproc 中的简单程序语言在语法和语义上比较接近,便于在建立状态迁移图以及进行不变式构造时与 Interproc 中的相应语法进行对应,在构造不变式的过程中可以大大减少工作的难度。在 3.1 节中,将具体讨论 CIL 与 Interproc 中语法的异同点。

基于上述原因,本文选用 CIL 来对 C 程序进行预处理,得出其相对规范的中间表示,然后再基于此中间表示建立状态迁移图。这里需要说明的是,本文面向的是简化的 C 程序,程序中只含有数值变量(即变量类型仅为整型、实型),不包含比如数组、字符串、指针、结构等类型,不支持强制跳转 goto。

2.2 建立程序状态迁移图

程序的状态迁移图是一种有向、有环的标记迁移图,图中的每一个结点都代表一个程序点,两个结点之间的边用迁移关系标记。

本文将状态迁移图定义为一个超图(hypergraph),超图是普通图的推广,在其中的边可以连接两个以上的顶点(类似于多元关系)。将其表示为三元组(V,E,I),其中 V 为程序的控制点集合,对应于状态迁移图的结点。E 为状态迁移图的边,即对程序状态产生影响的语句。I 为程序的相关信息,是一个四元组,用(procinfo, callret, pointenv, counter)来表示,其中 procinfo 代表程序中定义的函数信息,包括函数的起始点和返回点信息、输入输出变量以及本地变量的信息等;callret 代表函数的调用、返回点信息;pointenv 代表各程序点的变量的类型信息;counter 为一个计数器,代表图中的边的数量。

算法 1 抽象迁移图的生成算法

输入:C 程序(程序变量只为单纯的数值变量,不含有指针、数组、结构等)

输出:(V,E,I)

(1)先用 CIL 对程序进行预处理,生成程序的抽象语法,并通过 CIL 生成程序的控制流信息;

(2)对程序进行第一遍的从前到后的遍历,生成 I,这一步的重点是将程序变量与后续的数值抽象域中的变量以及变量类型相关联;

(3)对程序进行第二遍的从前到后的遍历,生成 V、E,重点是处理条件语句、循环语句以及函数调用语句,并且要将所有的表达式用数值抽象域中语法形式来表示。

2.3 构造线性不变式

在已得出语义方程的基础上,对数值抽象域的调用由 APRON[11]来实现,而对不变式的求解调用由一个开源的不动点求解模块 Fixpoint[17]来实现。

Apron 是运用抽象解释理论来对程序中的数值变量做静态分析,计算出数值变量间的不变式的开源软件包。如图 2 所示,Apron 提供了对现在所有的数值抽象域的访问接口,以便程序分析者对各种抽象域进行综合和分析比较。

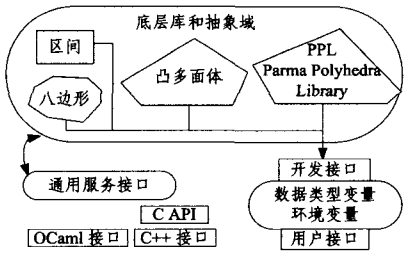

图 2　APRON 的组织结构

Fixpoint 是一个不动点求解引擎，它的接口参数都直接与抽象域相关联，并且它的计算过程也是基于抽象域的。Fixpoint 中实现了两种不动点求解方法，一种是 Kleene-Bourdoncle 的迭代技术结合 working-set 算法[18]，用来求解由超图描述的方程，并提供了一些基本的函数；另一种是 Gopan-Reps 制导的迭代技术[19]，用于求解由超图描述的方程。

下面对不变式及不变式的构造过程给出形式化的描述。

假设程序控制点 $k\in K$，表示程序语句的位置信息；环境 $\delta\in Env=Var\to\mathcal{R}$（$\mathcal{R}$ 为实数域），即程序变量的取值情况，定义成一个对一系列变量进行赋值的函数；程序的活动记录 $(k,\delta)\in K\times Env$，即程序在 k 点的环境状态；程序状态 $s\in S=(K\times Env)^+$，即一个由活动记录组成的非空栈，记录了在各程序点程序变量的取值。

定义 1（线性不变式）　程序的线性不变式 I 是一个环境 (Env) 的集合，即 $I\in\wp(Env)$。

每个控制点的数值变量间的不变式隐含了在该控制点的变量的可能的取值范围。线性不变式的构造是一个映射 P，该映射的结果是将当前程序点映射到不变式，即为：

$$P:\wp((K\times Env)^+)\to(K\to\wp(Env))$$

构造不变式的过程分为前向分析和后向分析两个过程：前向分析过程计算程序能够到达的状态集合，对于前向分析 $Reach=\{s\in S\mid s_0\in S_0\wedge s_0\to^* s\}$，构造不变式返回结果为 $P(Reach)$ 的上界逼近。

后向分析过程计算所有能够到达特定 fail 语句（一般指错误状态）的程序状态集合，对于后向分析有可达集合 $Coreach=\{s\in S\mid s\to^* s_f\wedge s_f\in F\}$，其中最终状态 $F=\{(k,\delta)\mid k\in K_{fail}\wedge\delta\in Env\}$，$K_{fail}$ 是恰在 fail 语句之前的控制点集合，从而后向分析返回 $P(Coreach)$ 的上界逼近。

3　实验结果与讨论

根据本文给出的方法，实现了工具 CILinear，用于对简化的 C 程序构造数值变量间的线性不变式。下面通过几个实验例子来说明，CILinear 不但能够与 Interproc 达到相同的目的，而且还拥有比 Interproc 更多的优点，正是这些优点，使得 CILinear 比 Interproc 的实用性更强，能够进一步应用分析于较大规模的程序。

```
    proc MC(n:int) returns(r:int) var t1:int,t2:int;
begin
  /*(L5 C5) top*/
if n>100 then
/*(L6 C17) [|n-100>=0|]*/
    r=n-10;/*(L7 C14) [|n-100>=0;-n+r+10>=0;n+
r-190>=0;r-90>=0;n-r-10>=0|]*/
  else
    /*(L8 C6) [|-n+100>=0|]*/
  t1=n+11;/*(L9 C17) [|-n+100>=0;-n+t1-11>=0;
      -n-t1+211>=0;n-t1+11>=0;-t1+111>=0|]*/
  t2=MC(t1);/*(L10 C17) [|-n+100>=0;-n+t1-11>=0;
      -n-t1+211>=0;n-t1+11>=0;-t1+111>=0;-n+
      t2-1>=0;-t1+t2+10>=0;t2-90>=0|]*/
  r=MC(t2);/*(L11 C16) [|-n+100>=0;-n+r+9>=0;r-
    90>=0;n+t1-11>=0;-n-t1+211>=0;n-t1+11>=
    0;r-t1+20>=0;t1+111>=0;-n+t2-1>=0;r+t2-
    180>=0;t1+t2+10>=0;t2-90>=0;r-t2+10>=0|]
    */
  endif;/*(L12 C8) [|-n+r+10>=0;r-90>=0|]*/
end
var a:int,b:int
begin
  /*(L17 C5) top*/
  b=MC(a);/*(L18 C12) [|-a+b+10>=0;b-90>=0|]*/
end
```

图 3　Interproc 计算的结果

```
int MC(int n)
{int t1, t2,r;
 if (n>100) /*(maccarthy91.c: L5 C111) [|-n+r+10>=0;
   r-90>=0|]*/
  r=n-10; /*(maccarthy91.c: L6 C127) [|n-100>=0; -n+
r+10>=0; n+r-190>=0; r-90>=0; n-r-10>=0|]*/
 else{
    t1=n+11;/*(maccarthy91.c:L8 C150) [|-n+100>=0;
        -n+t1-11>=0; -n-t1+211>=0;n-t1+11>=0;
        -t1+111>=0|]*/
    t2=MC(t1);/*(maccarthy91.c: L9 C168) [|-n+100>=0;
        -n+t1-11>=0; -n-t1+211>=0; n-t1+11>=0;
        -t1+111>=0; -n+t2-1>=0; -t1+t2+10>=0; t2
        -90>=0|]*/
    r=MC(t2);/*(maccarthy91.c:L10 C186)
     [|-n+100>=0;-n+r+9>=0;r-90>=0;-n+t1-
     11>=0; -n-t1+211>=0;n-t1+11>=0;r-t1+20>=
     0;-t1+111>=0;-n+t2-1>=0;r+t2-180>=0;-t1+
     t2+10>=0;t2-90>=0; r-t2+10>=0|]*/
     }
  return (r);/*(maccarthy91.c: L12 C203) [|-n+r+10>=0;
          r-90>=0|]*/
}
int main(){
int a, b;
  b=MC(a);/*(maccarthy91.c: L16 C241) [|-a+b+10>=0;
    b-90>=0|]*/
  return (0);/*(maccarthy91.c: L17 C252)[|-a+b+10>=0;
    b-90>=0|]*/
}
```

图 4　CILinear 计算的结果

实验环境为在 WinXP 用 cygwin 模拟 linux 运行环境，CPU 为 Intel Core 2 T5700，主频 1.66G，内存 1GB，硬盘 5400 转。程序编译环境为 OCaml 3.09，gcc 3.4.4，CIL 1.3.6。

为方便理解，简单说明了 Interproc 和 CILinear 生成的不

变式的结构，二者都是将不变式以注释的形式标注在程序中，Interproc 将不变式标注在分支的末尾，而 CILinear 将不变式的计算结果标注在分支程序的起始位置。例如：Interproc 中的不变式/*(L12 C8) [|-n+r+10>=0;r-90>=0|] */表示在第 12 行的第 8 列生成不变式[|-n+r+10>=0;r-90>=0|]；而 CILinear 中的/*(maccarthy91.c: L12 C203) [|-n+r+10>=0;r-90>=0|] */则表示在文件 maccarthy91.c 的第 12 列，第 203 个字节处生成不变式[|-n+r+10>=0;r-90>=0|]。

例 1　maccarthy91 程序

maccarthy91 程序的功能是对任何大于 101 的整数 n 都返回值 $n-10$，而对于任何小于等于 101 的整数，该程序都返回值 91。

本例中使用的是八边形抽象域，CILinear 运行 0.079s 计算出不变式，通过比较结果，可以看出对于存在函数递归调用的同一段程序，Interproc 与 CIL 构造的线性不变式是完全一样的，并且根据 maccarthy91 的功能，可知计算的不变式的正确性，如图 3 和图 4 所示。

例 2　一个简单的含有 for 循环的程序

Interproc 对于含有 for 循环的程序无法构造不变式，而 CILinear 却能很成功地构造相应的不变式（运行时间 0.072s，使用的是区间抽象域），如图 5 所示。

```
int main()
{
  int i,j,y,mm,nn;
  i=1; /* (test2.c: L4 C96) [|i-1=0|] */
  nn=1; /* (test2.c: L5 C101) [|i-1=0; nn-1=0|] */
  for (i=1;i<=5;i++)
       /* (test2.c: L6 C107) [|i-1>=0; nn-1>=0|] */
  {
    nn=2*i+nn; /* (test2.c: L8 C128) [|i-1>=0; -i+5>
       =0; nn-3>=0|] */
  }
  return (0); /* (test2.c: L5 C101) [|i-1>=0; nn-1>=0|]
       */
}
```

图 5　CILinear 对 for 循环的计算结果

例 3　而对于图 6 中这样包含多文件的程序，尤其是像 C 程序中经常会有宏定义的程序，Interproc 将无法对其计算不变式，而 CILinear 同样能够有效地构造不变式（运行时间为 0.086s，采用区间抽象域，如图 7 所示）。

```
#include "test1.c"
#define exp(2*i+2*j)
int main(){
  int m,y,i,j;
    ......
  y=exp;
  m=getmax(j,y);/* 函数 getmax()在文件 test1.c 中 */
    ......
  return(0);
}
```

图 6　Interproc 无法处理的含多文件程序

```
#include "test1.c"
#define exp (2*i+2*j)
int main(){
  int m,y,i,j;
    ......
  y=exp;/* (test3.c:L30 C424[|i-1=0;j-2=0;y-6=0|]) */
  m=getmax(j,y);/* (test3.c:L31 C437)[|i-1=0;j-2=0;m-
     6=0;y-6=0|] */
  ......
  return(0);
}
```

图 7　CILinear 的计算结果

由表 1 可以看出，Interproc 支持的语言的语法是本文所能处理的语法的一个子集。

表 1　CILinear 与 Interproc 处理的语法比较

语句类型	Interproc	CIL/CILinear
赋值语句	是	是
函数调用	是	是
if 条件分支语句	是	是
while 语句	是	是
do s while e	否	是
多分支选择的 switch 语句	否	是
for 循环	否	是
宏定义	否	是
多文件	否	是

表 2 中列出了典型测试实例，即对每个实例给出求解线性不变式所需要的时间、主要计算不变式算法的迭代次数以及生成不变式的数量。

表 2　实例测验结果

程序实例	所使用的抽象域	时间	迭代次数	线性不变式数量
ackerman	区间	0.140s	4	54
fact	区间	0.234s	8	71
symmetricalstairs	区间	0.047s	5	20
cousot77	区间	0.078s	4	15
fibonacci	区间	0.062s	5	25
maccarthy91	八边形	0.109s	4	40
bubblesort	八边形	0.156s	6	184

由实验结果可以看出，CILinear 可以有效地构造出 C 程序中数值变量间的线性不变式，并且 CILinear 比 Interproc 支持的语法更广泛，对多文件程序以及实际的 C 程序语法，CILinear 更加适用，由此可以看出 CILinear 的实用性比 Interproc 更强。

4　CILinear 在程序验证中的应用

程序验证的目的就是从形式化上证明程序的正确性。而程序的正确性则意味着程序的行为合乎程序说明书中所描述的内容。对程序正确性证明的方法比较有名的是 Floyd 的归纳断言法和 Hoare 的公理化方法。

归纳断言法的核心思想就是找出证明程序正确性的一些逻辑表达式，或者叫检验条件。通过证明这些检验条件为真，以证明整个程序为真。公理化方法是归纳断言法的扩展，区别就在于通过引入 1 条赋值公理和 4 条推理规则规范了归纳断言法中检验条件的抽取，这里检验条件被称为逻辑表达式 LOG。Floyd 的归纳断言法和 Hoare 的公理化方法的共同缺点是在进行断言推理时需要引入前置和后置条件的计算；而

在大部分情况下计算程序的前置和后置条件是比较困难的并且计算程序的最弱前置条件是不可判定的，这使得 Floyd 的归纳断言法和 Hoare 的公理化方法给较大规模程序的分析带来了困难。

基于不变式的方法降低了自动验证的难度，其在每个程序点都计算程序变量需要满足的不变式集合。不变式集合描述了程序变量需要满足的各种关系，只需要验证该不变式集合是否满足程序断言即可。

定理 1 设 S 为简单 C 程序，p 为 S 中一个语句，$Inv(p,S)$ 为语句 p 处的不变式集合，$assert(\Phi)$ 为语句 p 处需满足的断言，则断言 $assert(\Phi)$ 成立当且仅当 $Inv(p,S)\rightarrow\Phi$。

证明：首先证明如果 $Inv(p,S)\rightarrow\Phi$，则 $assert(\Phi)$ 成立。因为 $Inv(p,S)$ 是 p 处的不变式集合，故 $Inv(p,S)$ 中的每一个不变式都在 p 处成立，即 $Inv(p,S)$ 成立，从而由推演规则可知 Φ 为 true，即 $assert(\Phi)$ 成立。

其次证明如果 $assert(\Phi)$ 成立，则 $Inv(p,S)\rightarrow\Phi$ 成立。用反证法，假设 $Inv(p,S)\rightarrow\Phi$ 不成立。而已知 $Inv(p,S)$ 成立，故可得 Φ 不成立与假设相矛盾。从而可得 $assert(\Phi)$ 成立，则 $Inv(p,S)\rightarrow\Phi$ 成立。

因此断言 $assert(\Phi)$ 成立当且仅当 $Inv(p,S)\rightarrow\Phi$。

以以下程序为例说明如何结合不变式集合进行程序验证。

例 4 计算 x 除以 y 的商 *quo* 和余数 *rem*

```
int main(){
  quo=0;
  rem=x;
  while y≤rem{
    rem =rem−y;
    quo=quo+1;
  }/*assert(x=quo*y+rem,0≤rem≤y−1,x≥0,y>0)*/
}
```

用 CILinear 可以得出“done”语句处的线性不变式集合为 $\{-rem+y-1>=0;rem>=0;quo>=0;quo+rem-1>=0\}$。由循环不变式生成工具可得循环不变式为 $\{rem+quo*y-x=0\}$。

从而 $Inv(done,S)$ 为 $\{rem+quo*y-x=0;-rem+y-1>=0;rem>=0;quo>=0;quo+rem-1>=0\}$。

所以断言成立当且仅当 $rem+quo*y-x=0\wedge-rem+y-1>=0\wedge rem>=0\wedge quo>=0\wedge quo+rem-1>=0\rightarrow x=quo*y+rem,0\wedge rem\leqslant y-1\wedge x\geqslant 0\wedge y>0$。

结束语 基于 Interproc 对简单程序构造线性不变式的方法，结合 CIL，本文设计并实现了 C 程序数值变量线性不变式自动构造工具 CILinear，并具体给出了其在程序验证中的应用。本文实现的构造线性不变式的工具 CILinear，比 Interproc 能够支持更广泛的语法，能够支持多文件，能够支持宏定义；与 Interproc 比较，CILinear 的通用性更强，能够用于分析大规模的工程文件包。

但本文的工作仍然仅限于单纯的数值型变量，下一步的工作，将研究扩展到支持数组、指针等；并且由于线性不变式的精度问题，对于循环语句无法得出精确的非线性不变式，下一步将对循环语句考虑多项式不变式的构造方法，并结合多项式等式不变式与线性不等式不变式构造多项式不等式不变式，并通过本文实现的工具 CILinear 与数学工具 Mathematica 将多项式不等式不变式的构造方法自动化。

参 考 文 献

[1] Cousot P, Cousot R. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]// 4th POPL. Los Angeles, CA, ACM Press, 1977: 238-252

[2] 李梦君，李舟军，陈火旺. 基于抽象解释理论的程序验证技术[J]. 软件学报，2008，19(1)：17-26

[3] Blanchet B, Cousot P, Cousot R, et al. A static analyzer for large safety-critical software[C]// Proc. ACMSIGPLAN'2003 Conf. PLDI: ACM Press, 2003: 196-207

[4] Cousot P, Cousot R, Feret J. Laurent Mauborgne, Antoine Miné, David Monniaux & Xavier Rival. The ASTRéE analyser[C]// Sagiv M, ed. ESOP 2005-The European Symposium on Programming, Lecture Notes in Computer Science 3444. Edinburgh, Springer, April 2005: 21-30

[5] Mauborgne L. ASTRéE: verification of absence of run-time error [C]// Jacquard R, ed. Building the Information Society. Kluwer Academic Publishers, 2004: 385-392

[6] Cousot P, Cousot R. Static determination of dynamic properties of programs[C]// Robinet B, ed. Proc. of the 2nd Int'l Symp. on Programming. Paris, 1976: 106-130

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

[8] Cousot P, Halbwachs N. Automatic Discovery of Linear Restraints Among Variables of a Program[C]// Annual Symposiuim on POPL. 1978: 84-96

[9] Colón M, Sankaranarayanan S, Sipma H. Linear Invariant Generation Using Non-linear Constraint Solving[C]// Proc. of CAV, volume 2725 of Lecture Notes in Computer Science. 2003: 420-432

[10] Lalire G, Argoud M, Jeannet B. Interproc[OL]. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/ interproc/ index.html

[11] Jeannet B, Antoine Min_e. The APRON library for Numerical Abstract Domains[OL]. http://apron.cri.ensmp.fr/library/

[12] Corporation M. The AST Toolkit [OL]. http://research.microsoft.com/sbt/asttoolkit/ast.asp

[13] Labs B. ckit: A Front End for C in SML[OL]. http://cm.bell-labs.com/cm/cs/what/smlnj/doc/ckit/overview.html

[14] Wilson R, French R, Wilson C, et al. The SUIF compiler system: a parallelizing and optimizing research compiler[R]. CSL-TR-94-620. Stanford University, Computer Systems Laboratory, May 1994

[15] Necula G C, McPeak S, Rahul S P, et al. CIL: Infrastructure for C Program Analysis and Transformation[OL]. http://manju.cs.berkele y.edu/cil/, 2007

[16] Necula G C, McPeak S, Rahul S P, et al. CIL: Intermediate Language and Tools for Analysis and Transformation of C programs [C]// Proceedings of Conference on Compiler Construction. 2002

[17] Fixpoint[OL]. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/ fixpoint/index.html

[18] Bourdoncle F. Efficient Chaotic Iteration Strategies with Widenings[C]// Proc. of the International Conference on Formal Methods in Programming and their Applications, Lecture Notes in Computer Science 735. Springer-Verlag, 1993: 128-141

[19] Gopan D, Reps T W. Guided Static Analysis[Z]. SAS 2007: 349-365