软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1544−1565 [doi: 10.13328/j.cnki.jos.005463]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

自动合成数组不变式*

李 彬, 翟 娟, 汤震浩, 汤恩义, 赵建华

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 赵建华, E-mail: zhaojh@nju.edu.cn

摘 要: 提出了基于抽象解释框架自动合成数组程序不变式的方法,它能够分析按照特定顺序访问一维或者多维数组的程序,然后合成不变式.该方法将性质(包括区间全称量词性质和原子性质)集合作为抽象域,通过前向迭代数据流分析合成数组性质.证明了该方法的正确性和收敛性,并通过一些实例展示了该方法的灵活性.开发了一种原型工具,该工具在各种数组程序(包括 competition on software verification 中的 array-examples benchmark)上的实验展示了方法的可行性和有效性.

关键词: 不变式合成;抽象解释;数组程序

中图法分类号: TP301

中文引用格式: 李彬,翟娟,汤震浩,汤恩义,赵建华.自动合成数组不变式.软件学报,2018,29(6):1544−1565. http://www.jos.org.cn/1000-9825/5463.htm

英文引用格式: Li B, Zhai J, Tang ZH, Tang EY, Zhao JH. Automatic invariant synthesis for array programs. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1544−1565 (in Chinese). http://www.jos.org.cn/1000-9825/5463.htm

Automatic Invariant Synthesis for Array Programs

LI Bin, ZHAI Juan, TANG Zhen-Hao, TANG En-Yi, ZHAO Jian-Hua

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

Abstract: This paper proposes a method of using abstract interpretation for discovering properties about array contents in programs which manipulate one-dimensional or multi-dimensional arrays by sequential traversal. The method directly treats invariant properties (including interval universally quantified formulas and atomic formulas) set as abstract domains. It synthesizes invariants by “iterating forward” analysis. This method is sound and converges in finite time. The paper demonstrates the flexibility of the method by some examples. The method has been implemented in a prototype tool. The experiments applying the tool on a variety of array programs (including array-examples benchmark of competition on software verification) demonstrate the feasibility and effectiveness of the approach.

Key words: invariant synthesis; abstract interpretation; arrays programs

数组作为一种数据结构具有简单高效的特点,因此常常在程序中被使用.在验证数组程序正确性时,需要分析和使用数组(及其中的元素)具有的性质.然而,静态分析数组程序的行为是一个很有挑战的问题:首先,数组索引会导致复杂的语义,可能引起别名问题;其次,数组大小可以非常大甚至未知,这意味着非常大数量或者未知数量的变量.因此,分析数组程序经常需要使用和处理全称量词.

* 基金项目: 国家自然科学基金(61632015, 61561146394); 国家重点研发计划(2016YFB1000802)

Foundation item: National Natural Science Foundation of China (61632015, 61561146394); National Key Research Program (2016YFB1000802)

本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.

收稿时间: 2017-06-30; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28

CNKI 网络优先出版: 2017-12-29 13:19:10, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.006.html

合成数组程序的不变式问题已经吸引了很多研究者的关注.谓词抽象(predicate abstraction)方法[1,2]使用一些语法启发式方法推导用于抽象的谓词,然后使用这些谓词推理数组程序的不变式.反例制导的精化(counter-example guided refinement)方法[3]和 Craig 插值(craig interpolants)方法[4]提升了谓词抽象方法的效果.文献[5]的方法将数组所有元素看做一个变量处理,虽然提高了处理效率,但是却会导致分析结果不精确.数组划分(array partitioning)方法[6−8]将索引域([1...*n*])划分为若干符号区间,并且给每一个子数组关联一个摘要辅助变量(summary auxiliary variable),但是该方法难以处理多维数组,并且它基于语法信息来划分索引区间,程序写法上的变化可能导致程序无法被处理.模板(templates)方法[9]极其强大但是代价昂贵,并且需要用户提供数组性质模板.

针对那些通过循环控制变量的简单表达式来访问数组的程序,本文提出了基于抽象解释框架[10]的方法来发现一类程序的全称量词性质.这些程序被称为归纳循环程序(induction loop program).这类程序中的循环控制变量在循环中每次增加或者减少一个固定常量.需要特别指出,循环控制变量可以在一次迭代中增加/减少多次,如图 1(d)所示的 *i* 和 *j*.在现实中,很多程序或者程序片段都是归纳循环程序.图 1 展示了 4 个归纳循环程序的例子.图 1(a)和图 1(b)表示两个版本的 *arrayPartCopy* 程序.图 1(a)先进行数组操作再更新循环控制变量,图 1(b)先更新循环控制变量再进行数组操作.图 1(c)是二维数组的例子.图 1(d)所示为 QuickSort 中根据阈值分划数组的程序片段,其中的循环控制变量 *i* 和 *j* 分别在外循环的一次迭代中增加/减少多次.

```
unsigned size
int A[size],B[size];
i=0;
while(i<size) {
      A[i]=B[i];
      i=i+2;
}
```

(a) arrayPartCopy

```
unsigned size
int A[size],B[size];
i=0;
while(i<size) {
      i=i+2;
      A[i-2]=B[i-2];
}
```

(b) another arrayPartCopy

```
unsigned col,row;
int A[row][col];
for(int i=0;i<row;i++)
   for(int j=0;j<col;j++)
      if(A[i][j]==0)
         return true;
return false
```

(c) 2d- ArrayCheck

```
unsigned size
int A[size];
int x=A[0];
int i=1;
int j=size- 1;
while(i<=j) {
   if(A[i] <x) {
      A[ i-1 ] =A[i];
      i = i +1;
   }
   else{
      while(j>=i&&A[j]>=x) {
         j = J-1;
      }
      if( j >i ) {
         A[ i-1 ] =A[j];
         A[j ] =A[i];
         i = i + 1;
         j = J-1;
      }
   }
}
A[ i-1 ] = x;
```

(d) Find: partition phase of QuickSort

Fig.1 Induction loop programs

图 1 归纳循环程序

本文的方法能够自动发现数组元素和标量以及数组元素之间的<,≤=,≠性质.例如:在两个版本的 *arrayPartCopy* 程序(如图 1(a)和图 1(b)所示)结尾处,本文的方法可以发现数组 *A* 的所有偶数下标的元素等于对应数组 *B* 下标的元素;在 2d-ArrayCheck 程序(如图 1(c)所示)结尾处,本文的方法可以发现二维数组中的所有遍历过的元素都不等于 0;在 *Find* 程序(如图 1(d)所示)结尾处,本文的方法可以发现根据阈值 *x*,数组 *A* 被划分成小于 *x* 的部分和大于等于 *x* 的部分.这些性质正是这些程序期望的性质.

本文的主要贡献在于:基于抽象解释框架提出了一种完全自动化地发现数组性质的方法:首先,通过预分析找出程序中的循环控制变量;然后进行数组性质分析发现程序中的全称量词性质.数组性质分析会在分析过程

中自动生成全称量词性质,并根据循环控制变量的变化相应地维护和更新全称量词性质的区间.本文给出了方法的正确性证明和收敛性证明;提出了直接以原子性质和区间量词性质集合作为抽象域的数组性质分析方法.这种抽象域具有较强的灵活性和表达能力,能够描述一个数组区间上的元素所具有的性质;同时能够允许区间量词的嵌套,从而使得数组性质分析可以表示和处理多维数组的性质.

本文第 1 节使用一个例子直观地展示本文的方法.第 2 节定义归纳循环程序、数组性质以及基本知识(性质的内存域(memory scope)和迭代数据流框架).第 3 节描述预分析,它给出循环控制变量的判断方法.第 4 节给出数组性质分析方法.第 5 节是工具实现和实验评估.第 6 节是相关工作介绍.最后总结全文,并对未来值得关注的研究方向进行初步探讨.

1 一个例子

本节将使用图 1(a)所示的 *arrayPartCopy* 程序简单展示本文所述方法.本文的方法关注区间形式的全称量词性质$\forall k(k\in[e_1,c,e_2)\Rightarrow p(k))$,其中,区间$[e_1,c,e_2)$表示一个集合.

$$[e_1,c,e_2)=\begin{cases}\{e_1+n\times c\mid n\geqslant 0\wedge e_1+n\times c<e_2\}, & \text{如果}c>0\\ \{e_1+n\times c\mid n\geqslant 0\wedge e_1+n\times c>e_2\}, & \text{如果}c<0\end{cases}.$$

例如,在 *arrayPartCopy* 程序结尾处,方法能够自动发现区间全称量词性质$\forall k(k\in[0,2,size)\Rightarrow A[k]=B[k])$,即数组 A 的所有偶数下标的元素等于对应数组 B 下标的元素.这正是程序实现的功能.

本文的方法首先通过预分析找出程序中的循环控制变量、循环控制变量初始化语句和循环控制变量更新语句,识别出循环控制变量初始值和每次改变的步长.在 *arrayPartCopy* 程序中,i 是循环控制变量,语句 $i=0$ 是循环控制变量初始化语句,语句 $i=i+2$ 是循环控制变量更新语句.i 的初始值是 0,步长是 2.

然后,本文的方法进行数组性质分析来发现程序中全称量词性质.数组性质分析会在分析过程中自动生成全称量词性质,并根据循环控制变量的变化相应地维护和更新全称量词性质的区间.它的基本步骤如下.

1. 在循环控制变量初始化语句 $i=init$ 之后,数组性质分析会生成一个特殊的全称量词性质$\forall k(k\in[init,c,i)\Rightarrow\text{false})$,其中,$c$ 是循环控制变量 i 的步长.因为循环控制变量初始化语句 $i=init$ 后,$[init,c,i)=\varnothing$成立,因此$\forall k(k\in[init,c,i)\Rightarrow\text{false})$成立.该性质用于生成其他全称量词性质;
2. 如果$\forall k(k\in[init,c,i)\Rightarrow\text{false})$和性质 $p(i)$在一个程序点处成立,那么数组性质分析会在该程序点上生成全称量词性质$\forall k(k\in[init,c,i+c)\Rightarrow p(k))$.因为$\forall k(k\in[init,c,i)\Rightarrow\text{false})$成立表明$[init,c,i)$是$\varnothing$,又因为循环控制变量 i 的步长是 c,所以$[init,c,i+c)$是就是集合$\{i\}$,所以 $p(i)$可推导出$\forall k(k\in[init,c,i+c)\Rightarrow p(k))$;
3. 如果$\forall k(k\in[init,c,i)\Rightarrow p(k))$和性质 $p(i)$在一个程序点处成立,那么数组性质分析会在该程序点上生成全称量词性质$\forall k(k\in[init,c,i+c)\Rightarrow p(k))$.因为循环控制变量 i 的步长是 c,所以$[init,c,i+c)$是就是集合$[init,c,i+c)\cup\{i\}$,所以$\forall k(k\in[init,c,i)\Rightarrow p(k))$和 $p(i)$可推导出$\forall k(k\in[init,c,i+c)\Rightarrow p(k))$;
4. 在循环控制变量更新语句 $i=i+c$ 之后,数组性质分析会更新全称量词性质的区间.经过循环控制变量更新语句,$\forall k(k\in[init,c,i+c)\Rightarrow p(k))$会更新成$\forall k(k\in[init,c,i)\Rightarrow p(k))$,$\forall k(k\in[init,c,i)\Rightarrow p(k))$会更新成$\forall k(k\in[init,c,i-c)\Rightarrow p(k))$.

下面将使用 *arrayPartCopy* 程序展示具体的分析过程.

假设初始时,开始语句之前的数据流值是空集(即⊥),其他语句之前和之后的数据流值是所有可能的性质的集合(即⊤).*arrayPartCopy* 程序的数组性质分析过程如下.

- 第 1 次迭代
  - 第 3 行后:$\forall k(k\in[0,2,i)\Rightarrow\text{false})$性质成立.因为循环控制变量初始化语句 $i=0$ 后,$[0,2,i)=\varnothing$成立,因此$\forall k(k\in[0,2,i)\Rightarrow\text{false})$成立;
  - 第 4 行(while 语句的汇点):$\forall k(k\in[0,2,i)\Rightarrow\text{false})$性质成立.第 4 行的性质由第 3 行之后和第 6 行之后的性质交汇得到.因为初始时第 6 行后的数据流值是所有可能性质的集合,所以第 4 行的性质就是第 3 行之后的性质;

- 第 5 行后:$\forall k(k\in[0,2,i)\Rightarrow \text{false}),i<size,A[i]=B[i],\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$性质成立.性质$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$由$\forall k(k\in[0,2,i)\Rightarrow \text{false})$和 $A[i]=B[i]$推导得到.因为$\forall k(k\in[0,2,i)\Rightarrow \text{false})$表明$[0,2,i)$是$\varnothing$,因此$[0,2,i+2)$就是集合$\{i\}$,所以 $A[i]=B[i]$可推导出$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$;
- 第 6 行后:$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$和$\forall k(k\in[0,2,i-2)\Rightarrow \text{false})$性质成立.因为循环控制变量更新语句 $i=i+2$ 之后,区间$[0,2,i+2)$和$[0,2,i)$分别变成$[0,2,i)$和$[0,2,i-2)$.所以可以由第 5 行之后的性质$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$得到$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$在第 6 行之后成立.同理,可由第 5 行之后的性质$\forall k(k\in[0,2,i)\Rightarrow \text{false})$得到$\forall k(k\in[0,2,i-2)\Rightarrow \text{false})$在第 6 行之后成立.

- 第 2 次迭代
  - 在第 4 行(while 语句的汇点):$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$性质成立.第 4 行的性质由第 3 行之后和第 6 行之后的性质交汇得到.因为第 3 行后的性质$\forall k(k\in[0,2,i)\Rightarrow \text{false})$和第 6 行后的性质$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$交运算结果是$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$,所以$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$在第 4 行成立;
  - 第 5 行后:$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k]),i<size,A[i]=B[i],\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$性质成立.性质$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$由$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$和$A[i]=B[i]$推导得到.因为 i 的步长是 2,所以$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$成立;
  - 第 6 行后:$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k]),\forall k(k\in[0,2,i-2)\Rightarrow A[k]=B[k])$成立.迭代收敛.

因此,在程序结束时有:$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k]),size\leqslant i,\forall k(k\in[0,2,size)\Rightarrow A[k]=B[k])$性质成立.因为 $size\leqslant i$,所以$\forall k(k\in[0,2,size)\Rightarrow A[k]=B[k])$成立.

2 语言和预备知识

2.1 归纳循环程序(induction loop program)

循环控制变量指在循环中每次增加或者减少一个固定常量的变量,循环控制变量可以在一次迭代中增加(或者减少)多次.归纳循环程序指通过循环控制变量的简单表达式(循环控制变量与常量的加减运算)访问数组的程序.图 2 给出了归纳循环程序中语句的语法.

$$
\begin{aligned}
statement\ ::=\ & lhs\ =\ expr\\
&\mid lcv = init\\
&\mid lcv = lcv\,'+'\,|\,'-'\,constant\\
&\mid while(cond)\ statement\\
&\mid if\ (cond)\ statement\ else\ statement\\
lhs\ ::=\ & Cvars\ \mid Array;\\
lcv\ ::=\ & Indices;\\
init\ ::=\ & expr;\\
constant\ ::=\ & INT;\\
Cvars\ ::=\ & 'x'\ \mid\ 'y'\ \cdots;\\
Array\ ::=\ & (\,'A'\ \mid\ 'B'\ \ldots)([Iexp])+;\\
Iexp\ ::=\ & constant \mid lcv \mid lcv\ ('+'\ |'-')\ INT;\\
Indices\ ::=\ & 'i'\ \mid\ 'j'\ \ldots\ ;\ (Indices\cap Cvars\ =\varnothing)\\
cond\ ::=\ & expr\ Op\ expr\\
&\mid cond\ '||'\ cond\\
&\mid cond\ '\&\&'\ cond;\\
Op\ :=\ & '==\,'|'<\,'|'\leq\,'|'\neq\,'|'>\,'|'\geq\,';
\end{aligned}
$$

Fig.2 Syntax of statements

图 2 语句语法

其中,赋值语句分为 3 类:赋值到一般的标量、赋值到数组元素、赋值到循环控制变量.赋值到循环控制变

量又分为两类:循环控制变量初始化语句 *lcv*=*init* 和循环控制变量更新语句 *lcv*=*lcv*(+|–) *constant*,其中,*lcv* 不出现在 *init* 中,循环控制变量更新语句的第 1 个 *lcv* 和第 2 个 *lcv* 相同.表达式 *expr* 是没有副作用的关于常量,标量和数组元素的+,–,*,/,%表达式(图 2 没有给出它的具体语法,本文的方法只需要 *expr* 没有副作用).数组下标表达式 *Iexp* 是常量,循环控制变量或者循环控制变量和常量的加减运算.*Iexp* 是 *expr* 的子集,因为本文的很多论述都涉及到对下标表达式的处理,因此将下标表达式作为一个独立的语法概念.条件表达式 *cond* 是析取或者合取表达式.

2.2 数组性质

本文所分析的性质语法如图 3 所示.

$$
\begin{aligned}
&prop\ ::=\ atomProp \mid forAllProp\ ;\\
&atomProp\ ::=\ expr\ op\ expr;\\
&forAllProp\ ::= \forall ID\ '('\ ID \in interval\ '\Rightarrow'\ (prop \mid false)\ ')';\\
&op\ ::=\ \ = \mid \neq \mid < \mid \leq\\
&interval\ ::= ('(' \mid '[')\ expr_1\ ','\ constant\ ','\ expr_2\ (')' \mid ']')
\end{aligned}
$$

Fig.3 Syntax of properties

图 3 性质语法

性质可以是原子性质 *atomProp* 或者全称量词性质 *forAllProp*.原子性质 *atomProp* 是表达式之间的<,≤=,≠关系.∀*ID*(*ID*∈*interval*⇒false)表示一个特别的性质,主要用于在分析过程中和其他性质一起生成新的全称量词性质.

区间 *interval* 表示一个集合,定义如下:

$$
(expr_1, constant, \exp r_2) = \begin{cases} \{expr_1 + n \times constant \mid n > 0 \wedge expr_1 + n \times constant < expr_2\}, & \text{如果} constant > 0 \\ \{expr_1 + n \times constant \mid n > 0 \wedge expr_1 + n \times constant > expr_2\}, & \text{如果} constant < 0 \end{cases},
$$

$$
[expr_1, constant, expr_2) = \begin{cases} \{expr_1 + n \times constant \mid n \geqslant 0 \wedge expr_1 + n \times constant < expr_2\}, & \text{如果} constant > 0 \\ \{expr_1 + n \times constant \mid n \geqslant 0 \wedge expr_1 + n \times constant > expr_2\}, & \text{如果} constant < 0 \end{cases},
$$

$$
(expr_1, constant, expr_2] = \begin{cases} \{expr_1 + n \times constant \mid n > 0 \wedge expr_1 + n \times constant \leqslant expr_2\}, & \text{如果} constant > 0 \\ \{expr_1 + n \times constant \mid n > 0 \wedge expr_1 + n \times constant \geqslant expr_2\}, & \text{如果} constant < 0 \end{cases},
$$

$$
[expr_1, constant, expr_2] = \begin{cases} \{expr_1 + n \times constant \mid n \geqslant 0 \wedge expr_1 + n \times constant \leqslant expr_2\}, & \text{如果} constant > 0 \\ \{expr_1 + n \times constant \mid n \geqslant 0 \wedge expr_1 + n \times constant \geqslant expr_2\}, & \text{如果} constant < 0 \end{cases}.
$$

2.3 性质的内存域(memory scopes of properties)

内存域(memory scopes)的概念来自文献[11].对任何性质 *P*,它的取值只依赖于存储在有穷多个内存单元中的值.这些内存单元称为性质 *P* 的内存域 $\boldsymbol{M}(P)$.内存域可以被用于判断一个性质是否可以被传播到赋值语句之后:对任何赋值语句 e_1:=e_2,如果性质 *P* 在该语句之前成立,并且&e_1(即 e_1 的地址)不属于 $\boldsymbol{M}(P)$,那么性质 *P* 在该语句之后仍然成立.

对任何表达式 *e*,如果 *e* 满足 $f(e_1,\ldots,e_n)$的形式,那么内存域 $\boldsymbol{M}(e)$是 $\boldsymbol{M}(e_1)\cup\boldsymbol{M}(e_2)\cup\ldots\cup\boldsymbol{M}(e_n)$,其中,*f* 是代数运算符(e.g.+,–)或布尔运算符.表 1 给出了基本形式表达式的内存域.表 2 给出了不同性质的内存域.

Table 1 Memory scope for basic forms of terms

表 1 基本表达式项的内存域

项	内存域
常量 *c*	∅
变量 *v*	{&*v*}
$e_1[e_2]$	$\{\&e_1[e_2]\}\cup\boldsymbol{M}(e_1)\cup\boldsymbol{M}(e_2)$

Table 2 Memory scope for properties

表 2 性质的内存域

性质	内存域
e_1 bop e_2	$\boldsymbol{M}(e_1)\cup\boldsymbol{M}(e_2)$
$\forall x(x\in[e_1,c,e_2)\Rightarrow\text{false})$	$\boldsymbol{M}(e_1)\cup\boldsymbol{M}(e_2)$
$\forall x(x\in[e_1,c,e_2)\Rightarrow p)$	$\boldsymbol{M}(e_1)\cup\boldsymbol{M}(e_2)\cup\{\mathbf{M}(p)\mid x\in[e_1,c,e_2)\}$

2.4 迭代数据流框架

迭代数据流框架[10]用于求解数据流问题.迭代数据流框架可以刻画为 4 元组$(D,L_G,\sqcap_G,F_G)$,其中,G 表示控制流图(CFG).

- D 是分析方向:前向或后向;
- L_G 是一个交半格(meet semi-lattice)描述符,它表示待求解问题的数据流值.L_G 高度必须有穷;
- $\sqcap_G$ 是交半格 L_G 的交操作(meet operator);
- F_G 是从 L_G 到 L_G 的流函数(flow functions)集合.流函数必须是单调的.

对任何一个数据流分析问题,只要给出上面 4 个元素,就可以通过迭代数据流分析算法进行求解.如果流函数单调,并且格的高度有穷,那么分析过程就必然终止[10].

本文使用了两个前向数据流分析方法.它们的分析方程都是如下形式:

$$In_n=\begin{cases}BI, & n \text{ is } Start\\ \bigsqcap\limits_{p\in pred(n)} F_p(In_p), & \text{otherwise}\end{cases},$$

其中,n 和 p 是语句,In_n 是语句 n 之前的数据流值.F_n:$L_G\to L_G$ 是语句 n 的流函数.BI 表示程序开始语句 $Start$ 之前的数据流值.F_G 由一系列 F_p 构成.$\sqcap$是交操作.

3 预分析

对一个程序或者程序片段,数组性质分析中需要确定哪些变量是循环控制变量、循环控制变量的初始值是什么、步长是多少.因此,在给出数组性质分析之前,需要进行一次预分析,用于获取上面所说的循环控制变量的信息.

定义 1(循环控制变量(loop control variable)). 对于标量 x,从程序开始到某个程序点 u 的任何路径中,如果每条路径中对 x 的赋值语句要么是将一个初始值赋值到 x,要么是将 x 增加(或者减少)一个固定常量,那么 x 在程序点 u 处是循环控制变量(loop control variable).

本文设计了一种前向迭代数据流分析方法来发现循环控制变量.一个循环控制变量是一个三元组(*variable*,*initial_value*,*fixed_constant_step*).在这个分析中,数据流值域 L_G 是 $2^{(Svar,Expr,Con)}$,其中,$Svar$ 是待分析的 CFG G 中所有标量的集合,$Expr$ 是 G 中所有表达式的集合,Con 是 G 中所有整数常量集合并上$\{\bot,\top\}$.程序开始语句之前的数据流值 BI 是空集.初始时其他语句之前之后的数据流值是$\{(v,init,\top)\mid v\in Svar,init\in Expr\}$.$F_G$ 是$\{F_n\mid n \text{ is a } statement\}$.$F_n$ 是语句 n 的流函数,给定语句 n 之前的数据流值 v,$F_n(v)$表示 n 之后的数据流值.对语句 n 之前的任意数据流值 x,$F_n(x)$的定义如下:

$$F_n(x) = (x-Kill_n(x))\cup Gen_n(x),$$

$$Gen_n(x)=\begin{cases}\{(v,init,\top)\}, & n\text{形如}v:=init\\ \{(v,init,c)\}, & n\text{形如}v:=v+c,\text{并且}(v,init,\top)\in x\text{或者}(v,init,c)\in x,\\ \varnothing, & \text{否则}\end{cases}$$

$$Kill_n(x)=\begin{cases}\{(v,init,*)\}, & n\text{形如}v_1:=e,\text{并且}v_1\in\{v\}\cup\{init\text{的操作数}\},\text{并且}*\text{是任何常量或}\bot,\top\\ \varnothing, & \text{否则}\end{cases}.$$

对任何 $L_1,L_2\in L_G,L_1\sqcap_G L_2$ 定义如下:

$$L_1\sqcap_G L_2=\{(x,init,c)\mid 存在(x,init,c_1)\in L_1,(x,init,c_2)\in L_2使得c=c_1\bar{\sqcap}c_2并且c\neq\bot\}.$$

$c_1\bar{\sqcap}c_2$ 定义如下:

$$c_1\bar{\sqcap}c_2=\begin{cases}c_1, & c_1=c_2\\ \bot, & c_1\neq c_2\end{cases},$$

$$c_1\bar{\sqcap}\top=c_1,\ c_1\bar{\sqcap}\bot=\bot.$$

4 数组性质分析

本节给出一种前向迭代数据流分析方法来获取数组性质.

4.1 数据流值

令 *Expr* 表示待分析的 CFG *G* 中所有表达式的集合(所有的变量也看作表达式).在给出数据流值域 L_G 之前,首先给出两个辅助定义:*ProperP* 和 $\preceq$.*ProperP* 定义了方法在 CFG *G* 中能够合成的所有性质集合,它是一个有穷的集合;$\preceq$ 定义了任何两个性质之间的偏序关系.

ProperP 是满足下面条件的性质集合.

(1) 如果 e_1 和 e_2 在 *Expr* 中,那么 $e_1\ op\ e_2$ 在 *ProperP* 中,其中,*op* 可以是<,≤=,≠;

(2) 如果下面的条件都满足,那么 $\forall x(x\in I\Rightarrow p_1)$ 在 *ProperP* 中,其中,*I* 是 (e_1,c,e_2),$[e_1,c,e_2)$,$(e_1,c,e_2]$ 或 $[e_1,c,e_2]$:

a) e_1 和 e_2 在 *Expr* 中,*c* 是 *G* 中一个常量;

b) *G* 中存在一个循环控制变量 *lcv* 使得 $p_1[lcv/x]$ 在 *ProperP* 中,并且 $p_1\neq$false;

c) p_1 中量词嵌套的深度小于 *G* 中循环控制变量的数量;

(3) 如果 e_1 和 e_2 在 *Expr* 中,并且 *c* 是 *G* 中的一个常量,那么 $\forall x(x\in[e_1,c,e_2)\Rightarrow$false)在 *ProperP* 中.

图 4 定义了运算符 *Op* 的格.*Op* 的格图定义了运算符<,≤=,≠的偏序关系 $\sqsubseteq_{op}$ 和交运算 $\sqcap_{op}$.如果 $op_2\sqsubseteq_{op}op_1$,那么对于任意表达式 e_1 和 e_2 满足 $e_1\ op_1\ e_2\Rightarrow e_1\ op_2\ e_2$.运算符的偏序关系将在下面的定义中使用.

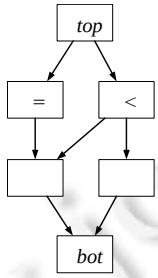

Fig.4 Lattice on *Op*, $(Op,\sqsubseteq_{op},\sqcap_{op},top,bot)$

图 4 运算符 *Op* 上的格 $(Op,\sqsubseteq_{op},\sqcap_{op},top,bot)$

下面给出 $\preceq$ 的定义.对任何性质 p_1 和 p_2,如果它们满足下面的条件之一,那么,$p_1\preceq p_2$.

(1) p_2 是原子性质 $(e_3\ op_2\ e_4)$,p_1 是原子性质 $(e_1\ op_1\ e_2)$ 并且 $e_1=e_3\wedge e_2=e_4\wedge op_1\sqsubseteq_{op}op_2$.$\sqsubseteq_{op}$ 在图 4 中定义;

(2) p_1 和 p_2 都是 $\forall x(x\in[e_3,c,e_4)\Rightarrow$false);

(3) p_2 是 $\forall x(x\in I_1\Rightarrow p_4)$,$p_1$ 是 $\forall x(x\in I_2\Rightarrow p_3)$ 并且下面条件之一成立.

a) $I_1=I_2\wedge p_4=$false;

b) $I_1=I_2\wedge p_3\preceq p_4$.

根据上面的定义可知,$p_1 \preceq p_2$ 意味着逻辑公式 $p_2 \Rightarrow p_1$ 成立.

定义 2(数组性质分析的数据流值域 L_G). L_G 的定义如下:

$$L_G=\{S|S\subseteq ProperP\wedge\forall p_1,p_2\in S,p_1\preceq p_2\Rightarrow p_1=p_2\}.$$

直观地讲,L_G 的元素是性质的集合,并且对于 L_G 中的任意元素 L,L 中的两个不同性质 p_1,p_2 之间不存在$\preceq$关系.如果 L 中有两个不同性质 p_1 和 p_2 满足 $p_1\preceq p_2$,那么意味着 L 中 p_1 是冗余的.因为 $p_2\Rightarrow p_1$,所以$\wedge L$ 逻辑等价于 $\wedge(L\backslash\{p_1\})$.根据上面的定义,空集$\varnothing$也是 L_G 中的数据流值.

L_G 上的偏序关系$\sqsubseteq$定义如下:对任何 $L_1,L_2\in L_G$,如果对于 L_1 中的任何性质 p_1,L_2 中都存在一个性质 p_2 使得 $p_1\preceq p_2$,那么 $L_1\sqsubseteq L_2$.

在数组性质分析中,程序开始语句之前的数据流值,即 BI 是$\varnothing$.初始时,其他语句之前和之后的数据流值是$\top$(所有可能性质的集合).

4.2 交操作(meet operation)

定义 L_G 上的交操作$\sqcap_G$之前,首先需要定义 $ProperP$ 上的$\overline{\sqcap}$操作.$\overline{\sqcap}$操作定义如下.

(1) $(e_1\ op_1\ e_2)\overline{\sqcap}(e_3\ op_2\ e_4)=\begin{cases}(e_1\ op_r\ e_2), & e_1=e_3\wedge e_2=e_4\wedge op_r=op_1\sqcap_{op}op_2\wedge op_r\neq bot\\ \perp, & \text{否则}\end{cases}$;

其中,$\sqcap_{op}$定义在图 4 中.

(2) 如果 p_1 和 p_2 都不是 false,那么:

$$\forall x(x\in[e_1,c_1,e_2)\Rightarrow p_1)\overline{\sqcap}\forall x(x\in[e_3,c_2,e_4)\Rightarrow p_2)=\begin{cases}\forall x(x\in[e_1,c_1,e_2)\Rightarrow p_1\overline{\sqcap}p_2), & [e_1,c_1,e_2)=[e_3,c_2,e_4)\wedge p_1\overline{\sqcap}p_2\neq\perp\\ \perp, & \text{否则}\end{cases};$$

(3) $\forall x(x\in[e_1,c_1,e_2)\Rightarrow \text{false})\overline{\sqcap}\forall x(x\in[e_3,c_2,e_4)\Rightarrow p_2)$,其中,$p_2$ 可以是 false:

$$\forall x(x\in[e_1,c_1,e_2)\Rightarrow false)\overline{\sqcap}\forall x(x\in[e_3,c_2,e_4)\Rightarrow p_2)=\begin{cases}\forall x(x\in[e_1,c_1,e_2)\Rightarrow p_2), & [e_1,c_1,e_2)=[e_3,c_2,e_4)\\ \perp, & \text{否则}\end{cases};$$

(4) 其他, $p_1\overline{\sqcap}p_2=\perp$.

有上面的定义可知,如果 $p_1\overline{\sqcap}p_2$ 不等于$\perp$,那么 $p_1\vee p_2$ 逻辑蕴含 $p_1\overline{\sqcap}p_2$.

$\sqcap_G$ 定义如下:

$$L_1\sqcap_G L_2=Reduce(\{p_1\overline{\sqcap}p_2\mid p_1\in L_1\wedge p_2\in L_2\wedge p_1\overline{\sqcap}p_2\neq\perp\}).$$

对任何 $S\in 2^{ProperP}$,$Reduce$ 函数定义如下:

$$Reduce(S)=S\backslash\{p|p\in S\wedge\exists p_1\in S(p_1\neq p\Rightarrow p\preceq p_1)\}.$$

直观地讲,对任何 $S\in 2^{ProperP}$,$Reduce$ 函数消除了 S 中具有$\preceq$关系的元素对中的较小的元素,使得 $Reduce(S)$ 属于 L_G.

定理 1. $(L_G,\sqcap_G,\sqsubseteq)$是关于 CFG G 的一个交半格(meet semi-lattice),并且高度有穷.

根据$\sqcap_G$的定义,$\sqcap_G$满足交换律、幂等律和结合律,两个元素的$\sqcap_G$运算结果是这两个元素关于$\sqsubseteq$的最大下界.因此,$(L_G,\sqcap_G,\sqsubseteq)$是一个交半格.$L_G=\{S|S\subseteq ProperP\wedge\forall p_1,p_2\in S,p_1\preceq p_2\Rightarrow p_1=p_2\}$,因此 $L_G\subseteq 2^{ProperP}$.因为 $ProperP$ 是有穷的,因此$(L_G,\sqcap_G)$的高度有穷.详细的证明过程见附录 8.1.

4.3 流函数(flow function)

语句 n 的流函数 F_n 由一系列的子函数定义.令 a 是语句 n 之前的数据流值,$F_n(a)$表示语句 n 之后的相应数据流值.在计算 $F_n(a)$时,首先计算语句 n 的语义,并且通过传播规则产生尽可能多的性质,该功能由 $Produce$ 函数定义.如果 n 是数组元素赋值语句、循环控制变量初始化语句或循环控制变量更新语句,流函数 $F_n(a)$分别进行

额外的处理:

(1) 如果 n 是数组元素赋值语句 $e_1[e_2]:=e_3$,流函数 $F_n(a)$会细粒度地检查全称量词性质是否会被赋值语句杀死,如果没有被杀死,那么该性质会传递到语句之后.该功能由 *Transfer* 函数定义;

(2) 如果 n 是循环控制变量初始化语句,流函数 $F_n(a)$产生特殊的全称量词性质.该功能由 *GenSpecial* 函数定义;

(3) 如果 n 是循环控制变量更新语句,流函数 $F_n(a)$会对全称量词的区间进行操作以产生更多的全称量词性质,该功能由 *HandleInterval* 函数定义.

最终,流函数 $F_n(a)$根据一组规则产生更多全称量词性质,该功能由 *GenAQ* 函数定义,并使得最终的性质集合属于 L_G,由 *Reduce* 函数定义.

流函数 F_n 由上面的子函数定义,*Reduce* 的定义在第 4.2 节,其他所有子函数的定义分别在后面的小节中详细描述.F_n 的正式定义会在本节的最后描述.

4.3.1 *Produce* 函数

对任何语句 n,*Produce* 函数计算语句 n 的语义,并且通过传播规则产生尽可能多的性质.*Produce* 函数定义如下:

$$Produce(n,a)=Propagated(a\cup Semantics(n,a)).$$

令 a 是语句 n 之前的数据流值.函数 $Semantics(n,a)$计算变量在语句之前和之后的关系.*Semantics* 定义如下:

$$Semantics(n,a)=\begin{cases}\{lh'=e\}\cup\{lh=lh_i\mid \&lh\neq \&lh_i\wedge lh_i\in LH\}, & \text{如果}n\text{是}lh:=e\\ \{cond\}\cup\{lh'=lh_i\mid lh_i\in LH\}, & \text{如果}n\text{是}cond\text{并且在true分支处},\\ \{\neg cond\}\cup\{lh'=lh_i\mid lh_i\in LH\}, & \text{如果}n\text{是}cond\text{并且在false分支处}\end{cases}$$

其中,lh_i 表示具有左值的表达式,LH 是出现在程序中的所有具有左值的表达式的集合.无撇号表达式表示表达式在语句之前的状态,撇号表达式表示表达式在语句之后的状态.*cond* 表示 if 和 while 语句的条件表达式.

Semantics 函数之后,*propagated* 函数会生成尽可能多的性质.*Propagated* 定义如下:

$$Propagated(a\cup Semantics(n,a))=Filter(ApplyRules(a\cup Semantics(n,a))).$$

Propagated 函数具有下面两个功能.

(1) 应用传播规则(propagation rules)来产生尽可能多的性质(*ApplyRules* 函数);

(2) 过滤掉不需要的性质.因为 *propagated* 函数的输入是 $Semantics(n,a)\cup a$.在 $Semantics(n,a)$中包含前置状态表达式(无撇号表达式).因此在应用传播规则之后,需要过滤掉包含前置状态表达式(无撇号表达式)的性质,最终留下只包含后置状态表达式(撇号表达式)的性质,然后去掉所有的变量的撇号(*Filter* 函数).

图 5 给出了部分传播规则.

(ASSIGN-1) $e_1=e_2\wedge e_1=e_3 \Rightarrow e_2=e_3$ **(ASSIGN-2)** $e_1=e_2\wedge e_1<e_3 \Rightarrow e_2<e_3$

(ASSIGN-3) $e_1=e_2\wedge \forall x(x\in[e_3,c,e_4)\Rightarrow e_5<e_1) \Rightarrow \forall x(x\in[e_3,c,e_4)\Rightarrow e_5<e_2)$

Note: x does not occur in e_1

(LT-1) $e_1<e_2\wedge e_2<e_3\Rightarrow e_1<e_3$

(LT-2) $e_1<e_2\wedge 0<c\wedge \forall x(x\in[e_3,c,e_2)\Rightarrow p) \Rightarrow \forall x(x\in[e_3,c,e_1)\Rightarrow p)$

(LT-3) $e_1<e_2\wedge 0<c\wedge \forall x(x\in[e_2,c,e_3)\Rightarrow p) \Rightarrow \forall x(x\in[e_1,c,e_3)\Rightarrow p)$

(LT-4) $e_1<e_2\wedge 0<c\wedge \forall x(x\in[e_3,c,e_2)\Rightarrow false) \Rightarrow \forall x(x\in[e_3,c,e_1)\Rightarrow false)$

(LT-5) $e_1<e_2\wedge 0<c\wedge \forall x(x\in[e_2,c,e_3)\Rightarrow false) \Rightarrow \forall x(x\in[e_1,c,e_3)\Rightarrow false)$

(LE-1) $e_1\le e_2\wedge e_2<e_3\Rightarrow e_1<e_3$

These are part of propagated rules; other rules are similar to these.

Fig.5 Propagation rules

图 5 传播规则

注意,所有的传播规则都需要满足下面给出的传播规则约束(propagation rules constraint).

传播规则约束. 令 ins 表示一个规则的输入性质集合,out 表示输出性质.一个传播规则可以描述为

$$\wedge ins \Rightarrow out.$$

传播规则需要满足如下约束:

(1) 如果 $ins \subseteq ProperP$,那么 $out \in ProperP$;

(2) 如果 $p \in ins$ 并且 p 是 $e_1 \leqslant e_2$,那么传播规则中必须包含另外两个规则:$\wedge ins[e_1<e_2/e_1 \leqslant e_2] \Rightarrow out_1$ 和 $\wedge ins[e_1=e_2/e_1 \leqslant e_2] \Rightarrow out_2$,其中,$out \preceq out_1$,$out \preceq out_2$.例如:$e_1 \leqslant e_2$ 在 LE-1 中,因此,传播规则提供了另外两个规则:LT-1 和 ASSIGN-2.LT-1 和 ASSIGN-2 被称为 LE-1 的对应规则;

(3) 如果 $p \in ins$ 并且 p 是 $e_1 \neq e_2$,那么传播规则中必须包含另外一个规则:$\wedge ins[e_1<e_2/e_1 \neq e_2] \Rightarrow out_1$,其中,$out \preceq out_1$;

(4) 如果 $p \in ins$ 并且 p 是 $\forall x(x \in [e_1,c,e_2) \Rightarrow p_1)$,那么传播规则中必须包含另外一个规则:

$$\wedge ins[\forall x(x \in [e_1,c,e_2) \Rightarrow \text{false})/\forall x(x \in [e_1,c,e_2) \Rightarrow p_1)] \Rightarrow out_1,\text{其中},out \preceq out_1.$$

传播规则约束保证了 *propagated* 函数满足单调性(证明见附录中的引理 6).

例 1:考虑图 1(a)所示 *arrayPartCopy* 程序.令语句 n 是 $A[i]=B[i]$、语句 n 之前的数据流值 $a=\{\forall k(k \in [0,2,i) \Rightarrow \text{false}),i<size\}$(第 1 次迭代时的性质):

$$Semantics(n,a)=\{A[i]'=B[i],B[i]'=B[i],i'=i,size'=size,B'=B\}.$$

应用传播规则并且过滤掉前置状态上的表达式(即不带′的表达式)之后,得到性质:$\{A[i]'=B[i]',(\forall k,k \in [0,2,i') \Rightarrow \text{false}),i'<size'\}$.去掉撇号,最终:

$$Propagated(a \cup Semantics(n,a))=\{A[i]=B[i],\forall k(k \in [0,2,i) \Rightarrow \text{false}),i<size\}.$$

4.3.2 *Transfer* 函数

如果 n 是 $e_1[e_2]:=e_3$,而 p 是数据流值 a 中的全称量词且 e_1 出现在 p 中,根据前面对内存域的讨论,若 $\&e_1[e_2] \notin \boldsymbol{M}(p)$,$p$ 在 n 之后仍然成立.但是 $Produce(n,a)$ 可能不包含 p.针对这种情况,*Transfer* 函数在 $\&e_1[e_2] \notin \boldsymbol{M}(p)$ 成立时将 p 传递到语句 n 之后.$Transfer(e_1[e_2]:=e_3,a)$ 定义如下:

$$Transfer(e_1[e_2]:=e_3,a)=\{p|p \in a \text{ 并且 } p \text{ 是一个全称量词性质并且} \&e_1[e_2] \notin \boldsymbol{M}(p)\}.$$

例 2:令语句 n 是 $A[i]=B[i]$、语句 n 之前的数据流值 $a=\{\forall k(k \in [0,2,i) \Rightarrow A[k]=B[k]),i<size\}$(“*arrayPartCopy*”程序第 2 次迭代时的性质):

$$Produce(n,a)=\{i<size\},$$
$$Transfer(n,a)=\{\forall k(k \in [0,2,i) \Rightarrow A[k]=B[k])\}.$$

因为 $i \notin [0,2,i)$,因此 $\&A[i] \notin \boldsymbol{M}(\forall k(k \in [0,2,i) \Rightarrow A[k]=B[k]))$,所以全称量词 $\forall k(k \in [0,2,i) \Rightarrow A[k]=B[k])$ 能够传递到语句 n 之后.

4.3.3 *GenSpecial* 函数

对任何循环控制变量初始化语句 $i:=init$,*GenSpecial* 定义如下:

$$GenSpecial(i:=init)=\{\forall k(k \in [init,c,i) \Rightarrow \text{false})\}.$$

c 是循环控制变量 i 的步长(通过预分析获取).在循环控制变量初始化语句 $i:=init$ 之后,因为 $[init,c,i)$ 是 $\varnothing$,所以 $\forall k(k \in [init,c,i) \Rightarrow \text{false})$ 成立.

4.3.4 *HandleInterval* 函数

对任何循环控制变量更新语句 $i:=i+c$,*HandleInterval* 定义如下:

$$HandleInterval(i:=i+c,a)=\{\forall x(x \in [init_i,c,i) \Rightarrow p) \mid \text{如果} \forall x(x \in [init_i,c,i+c) \Rightarrow p) \text{在} a \text{中}\} \cup$$
$$\{\forall x(x \in [init_i,c,i-c) \Rightarrow p) \mid \text{如果} \forall x(x \in [init_i,c,i) \Rightarrow p) \text{在} a \text{中}\},$$

其中,p 不包含 i,p 可以是 false.在 $i:=i+c$ 语句之后的 $[init_i,c,i)$ 和 $[init_i,c,i-c)$ 分别等价于语句之前的 $[init_i,c,i+c)$ 和 $[init_i,c,i)$.因此当 $\forall x(x \in [init_i,c,i) \Rightarrow p)$ 在语句 $i:=i+c$ 之前成立时,$\forall x(x \in [init_i,c,i-c) \Rightarrow p)$ 在语句之后成立.同理,当 $\forall x$

$(x\in[init_i,c,i+c)\Rightarrow p)$在语句之前成立时,$\forall x(x\in[init_i,c,i)\Rightarrow p)$在语句之后成立.

例 3:令语句 n 是 $i=i+2$、语句 n 之前的数据流值 $a=\{\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k]),i<size,A[i]=B[i],\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])\}$(*arrayPartCopy* 程序第 2 次迭代时的性质):

$$Produce(n,a)=\varnothing,$$

$$HandleInterval(n,a)=\{(\forall k(k\in[0,2,i-2)\Rightarrow A[k]=B[k]),(\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])\}.$$

4.3.5 *GenAQ* 函数

GenAQ 函数根据现有性质产生更多的全称量词性质.对任何 $S\in 2^{ProperP}$,*GenAQ* 定义如下:

$$GenAQ(S)=S\cup\{\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(...,e_1[f_1(x)],...))\mid 如果(1)或(2)成立\}\cup$$
$$\{\forall x(x\in[init_i,c,i)\Rightarrow\psi(...,e_1[f_1(x)],...))\mid 如果(3)或(4)成立\},$$

其中,ψ表示一个数组元素的性质,f_1 表示一个索引函数.需要特别注意,$\psi(\ldots,e_1[f_1(x)],\ldots)$不包含 i.

$$\forall x(x\in[init_i,c,i)\Rightarrow \text{false})在\ S\ 中,并且\ \psi(\ldots,e_1[f_1(x)],\ldots)在\ S\ 中 \tag{1}$$

$$\left.\begin{array}{l}\forall x(x\in[init_i,c,i)\Rightarrow\psi_1(...,e_1[f_1(x)],...))在S中,并且\psi_2(...,e_1[f_1(i)],...)在S中,\\ 并且\psi(...,e_1[f_1(x)],...)=\psi_1(...,e_1[f_1(x)],...)\sqcap\psi_2(...,e_1[f_1(x)],...)\end{array}\right\} \tag{2}$$

$$\forall x(x\in[init_i,c,i-c)\Rightarrow \text{false})在\ S\ 中,并且\ \psi(\ldots,e_1[f_1(i-c)],\ldots)在\ S\ 中 \tag{3}$$

$$\left.\begin{array}{l}\forall x(x\in[init_i,c,i-c),\psi_1(...,e_1[f_1(x)],...))在S中,并且\psi_2(...,e_1[f_1(i-c)],...)在S中,\\ 并且\psi(...,e_1[f_1(x)],...)=\psi_1(...,e_1[f_1(x)],...)\sqcap\psi_2(...,e_1[f_1(x)],...)\end{array}\right\} \tag{4}$$

- 如果条件(1)成立,$\forall x(x\in[init_i,c,i)\Rightarrow \text{false})$表示$[init_i,c,i)$是$\varnothing$.因此,$[init_i,c,i+c)$就是$\{i\}$,$\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots)$就是 $\psi(\ldots,e_1[f_1(i)],\ldots)$.所以$\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots)$成立;
- 如果条件(2)成立,因为 $\psi(\ldots,e_1[f_1(x)],\ldots)=\psi_1(\ldots,e_1[f_1(x)],\ldots)\sqcap\psi_2(\ldots,e_1[f_1(x)],\ldots)$,那么:$\forall x(x\in[init_i,c,i)\Rightarrow\psi_1(\ldots,e_1[f_1(x)],\ldots))$推出$\forall x(x\in[init_i,c,i)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots))$成立,$\psi_2(\ldots,e_1[f_1(i)],\ldots)$推出 $\psi(\ldots,e_1[f_1(i)],\ldots)$成立.因此,$\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots))$成立;
- 如果条件(3)成立,$\forall x(x\in[init_i,c,i-c)\Rightarrow \text{false})$表示$[init_i,c,i-c)$是$\varnothing$成立.因此,$[init_i,c,i)$就是$\{i-c\}$,$\forall x(x\in[init_i,c,i)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots)$就是 $\psi(\ldots,e_1[f_1(i-c)],\ldots)$.所以,$\forall x(x\in[init_i,c,i)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots))$成立;
- 若条件(4)成立,因为 $\psi(\ldots,e_1[f_1(x)],\ldots)=\psi_1(\ldots,e_1[f_1(x)],\ldots)\sqcap\psi_2(\ldots,e_1[f_1(x)],\ldots)$,那么:$\forall x(x\in[init_i,c,i-c)\Rightarrow\psi_1(\ldots,e_1[f_1(x)],\ldots))$ 推出 $\forall x(x\in[init_i,c,i-c)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots))$ 成立,$\psi_2(\ldots,e_1[f_1(i-c)],\ldots)$ 推出 $\psi(\ldots,e_1[f_1(i-c)],\ldots)$成立.因此,$\forall x(x\in[init_i,c,i)\Rightarrow\psi(\ldots,e_1[f_1(x)],\ldots))$成立.

综上所述,*GenAQ* 新生成的性质都能够由 S 中的规则经过推导得到.

例 4:考虑下面两个例子.

(1) 令 $S=\{A[i]=B[i],\forall k(k\in[0,2,i)\Rightarrow \text{false}),i<size\}$:

$$GenAQ(S)=S\cup\{\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])\}.$$

因为 $A[i]=B[i],\forall k(k\in[0,2,i)\Rightarrow \text{false})$在 S 中,满足条件(1),所以$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$成立.$\forall k(k\in[0,2,i)\Rightarrow \text{false})$表示区间$[0,2,i)=\varnothing$,又因为循环控制变量 i 的步长为 2,因此$[0,2,i+2)$就是$\{i\}$,所以$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$就是 $A[i]=B[i]$.

(2) 令 $S=\{A[i]=B[i],\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k]),i<size\}$:

$$GenAQ(S)=S\cup\{\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])\}.$$

因为 $A[i]=B[i]$和$\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$在 S 中,满足条件(2);又因为 $(A[k]=B[k])\sqcap(A[k]=B[k])=(A[k]=B[k])$,因此$\forall k(k\in[0,2,i+2)\Rightarrow A[k]=B[k])$成立.

4.3.6 流函数 F_n 的定义

令 a 是语句 n 之前的数据流值.流函数 $F_n(a)$定义如下:

$$F_n(a)=\begin{cases}Reduce(GenAQ(Transfer(n,a)\cup Produce(n,a))), & \text{如果}n\text{是}e_1[e_2]:=e_3\\ Reduce(GenAQ(GenSpecial(n)\cup Produce(n,a))), & \text{如果}n\text{是}lcv\text{初始化语句}\\ Reduce(GenAQ(HandleInterval(n,a)\cup Produce(n,a))), & \text{如果}n\text{是}lcv\text{更新语句}\\ Reduce(GenAQ(Produce(n,a))), & \text{否则}\end{cases}.$$

本文已经给出了数组性质分析的交半格(包括偏序关系和交运算)和流函数.根据文献[10]给出的迭代数据流框架,使用该迭代数据流框架中给出的迭代数据流分析算法就可以发现程序中的数组性质.

4.4 终止性(termination)

定理 2. 令 n 表示 CFG G 的语句.F_n 是单调的.

F_n 是单调的当且仅当对于 L_G 中的任意两个元素 x 和 y,$x\sqsubseteq y$ 可以推导出 $F_n(x)\sqsubseteq F_n(y)$.根据 F_n 的定义,只需要证明 F_n 中出现的函数都是单调的.详细的证明过程见附录.

定理 3. 数组性质分析必然会终止.

证明:因为流函数是单调的(定理 2),并且格的高度有穷(定理 1),因此数组分析必然会终止. □

4.5 正确性(soundness)

令 *States* 表示程序状态集合.一个程序状态表示为二元组(ρ_v,ρ_a),其中,ρ_v 表示标量到值的映射,ρ_a 表示数组到值的映射.图 6 给出了归纳循环程序语句的语义,这里使用 *States* 到 *States* 的函数$[\![\,]\!]$描述语句的语义,使用 $i:=e$ 表示赋值到标量(包括赋值到循环控制变量)语句,使用 $A[e_1]:=e_2$ 表示赋值到数组元素.这里简化了证明,只考虑一维数组.$\boldsymbol{Z}$ 表示整数,$\boldsymbol{B}$ 表示 boolean.

$$\begin{aligned}
&[\![\cdot]\!]:\quad statement\to States\to States\\
&\qquad e\to States\to \boldsymbol{Z}\\
&\qquad cond\to States\to \boldsymbol{B}\\
&\\
&[\![i:=e]\!](\rho_v,\rho_a)=(\rho_v[[\![e]\!](\rho_v,\rho_a)/i],\rho_a)\\
&[\![A[e_1]:=e_2]\!](\rho_v,\rho_a)=(\rho_v,\rho_a[F/A])\\
&\qquad where\ F=\lambda z\begin{cases}\rho_a(A)(z) & if\ \ z\neq[\![e_1]\!](\rho_v,\rho_a)\\ [\![e_2]\!](\rho_v,\rho_a) & otherwise\end{cases}\\
&[\![while(cond)\ stmt]\!](\rho_v,\rho_a)=\\
&\qquad\begin{cases}(\rho_v,\rho_a) & if\ \ [\![cond]\!](\rho_v,\rho_a)=false\\ [\![stmt;while(cond)stmt]\!](\rho_v,\rho_a) & otherwise\end{cases}\\
&[\![if\,(cond)\ stmt_1 else\ stmt_2]\!](\rho_v,\rho_a)=\\
&\qquad\begin{cases}[\![stmt_1]\!](\rho_v,\rho_a) & if\ [\![cond]\!](\rho_v,\rho_a)=false\\ [\![stmt_2]\!](\rho_v,\rho_a) & otherwise\end{cases}
\end{aligned}$$

Fig.6 Semantics of induction loop programs

图 6 归纳循环程序的语义

接下来的段落中,使用 c 标记程序状态(ρ_v,ρ_a).CFG G 的路径(trace) T 是程序状态序列$(c_0,c_1,...)$,其中,c_0 是初始状态,并且对所有 $i\geqslant 0$,有 $G\vdash c_i\rightsquigarrow c_{i+1}$.$\rightsquigarrow$表示状态之间的到达关系.

对于抽象值 a,$[\![a_i]\!](c_i)$成立当且仅当对于 a 中所有的性质 p,状态 c_i 使得性质 p 为真.

定理 4. 令 c_i 表示语句 n 之前的状态、a_i 表示语句 n 之前的数据流值.如果 $G\vdash c_i\rightsquigarrow c_{i+1}$ 并且$[\![a_i]\!](c_i)$成立,那么$[\![F_n(a_i)]\!](c_{i+1})$成立.

根据 F_n 的定义,只需要证明不同的语句 n,$[\![F_n(a_i)]\!](c_{i+1})$成立.详细的证明过程见附录.

定理 5. 对程序的任何初始值 c_0,$[\![BI]\!](c_0)$(BI 是程序开始语句之前的数据流值).

证明:在性质分析中,BI 是$\varnothing$.因此对任何初始值 c_0,$[\![BI]\!](c_0)$成立. □

根据定理 4 和定理 5 可知,分析结果是正确的.

4.6 方法的灵活性

在本文方法的基础上,我们可以通过扩展抽象域 L_G 和传播规则来发现更多的性质.这体现了我们方法的灵活性.

如果表达式 a 和 b 都出现在被分析的控制流图 G 中,但 $a+b$ 不出现在 G 中,根据第 4.1 节定义的 L_G,数组性质分析不能获得任何关于 $a+b$ 的性质.但是通过扩展抽象域(例如,将原来的表达式集合 *Expr* 定义为 $\{e|e$ 出现在 G 中$\}\cup\{e_1+e_2|e_1,e_2$ 出现在 G 中$\}$),数组性质分析可以获得更多性质.

类似于抽象域,方法也可以扩展传播规则.例如,对 *Summation starts* 程序(如图 7 所示),因为数组性质分析算法无法根据第 4.3.1 节给出的传播规则来获得关于 $A[i]$的任何性质,所以无法获得任何全称量词性质.但是只要增加如下两个规则,并根据传播规则约束增加相应的附属规则.

- $e_1=e_2+e_3\wedge 0\leqslant e_2\wedge 0\leqslant e_3\Rightarrow 0\leqslant e_1\wedge e_2\leqslant e_1\wedge e_3\leqslant e_1$;
- $e_1=e_2+e_3\wedge e_2\leqslant 0\wedge e_3\leqslant 0\Rightarrow e_1\leqslant 0\wedge e_1\leqslant e_2\wedge e_1\leqslant e_3$.

那么本文中的分析算法就可以在 *Summation starts* 的结尾处获得如下性质.

- $\forall k(k\in[0,1,MAXSIZE)\Rightarrow A[k]\leqslant positive_sum)$;
- $\forall k(k\in[0,1,MAXSIZE)\Rightarrow negative_sum\leqslant A[k])$.

```
negative_sum=0;
positive_sum=0;
for (i=0; i<MAXSIZE; i++) {
    if (array[i]<0) {
        negative_sum=negative_sum+array[i];
    }
    else {
        positive_sum=positive_sum+array[i];
    }
}
```

Fig.7 Summation starts

图 7 Summation starts

5 工具实现和实验

我们已经基于 clang[12]和 Z3[13]实现了一个原型工具.Clang 被用于在源代码层次进行数据流分析;Z3 被用于分析过程中的逻辑推导,所有的逻辑推导都只用到了 Z3 的线性约束求解功能.我们应用该工具在一些数组程序上发现全称量词性质,包括图 1 所示的小程序、一些常见程序(表 3)和 Competition on Software Verification (SV-COMP)[14]中的 array-examples benchmark.

5.1 小程序的分析结果

下面给出图 1 中程序的分析结果.

- *arrayPartCopy* 程序分析结果

本文给了两个版本的 *arrayPartCopy* 程序,其中,图 1(a)先进行数组操作再更新循环控制变量,图 1(b)先更新循环控制变量再进行数组操作.在两个程序的结尾,工具都可以发现下面的性质.

(1) $\forall k(k\in[0,2,i)\Rightarrow A[k]=B[k])$;

(2) $\forall k(k\in[0,2,size)\Rightarrow A[k]=B[k])$.

这些性质表明:在程序结束点上,数组 A 中偶数位索引的元素和数组 B 中偶数位索引的元素对应相等.这正是程序所实现的功能.

- *Find* 程序分析结果

Find 程序[15]使用数组的第 1 个元素分割数组,它是 QuickSort 中根据阈值分划数组的程序片段(如图 1(d)

所示).在程序结尾,工具可以发现下面的性质.

(1) $A[I-1]=x$;

(2) $\forall k(k\in[0,1,i)\Rightarrow A[k-1]<x)$;

(3) $\forall k(k\in[i,1,size-1]\Rightarrow x\leqslant A[k])$.

这些性质表明:在程序结束点上,根据阈值 x,数组 A 被划分成小于 x 的部分和大于等于 x 的部分:

- 2d-ArrayCheck 程序分析结果

2d-ArrayCheck 程序(图 1(c)所示)用于检查 2-d 数组中是否包含 0 元素.在程序的结尾,工具可以发现下面的性质:

$$\forall k_1(k_1\in[0,1,i)\Rightarrow\forall k_2(k_2\in[0,1,col)\Rightarrow A[k_1][k_2]\neq 0)).$$

这正是期望的一个性质:当运行到程序结尾处时,二维数组中的所有遍历过的元素都不等于 0.

除了上面的程序之外,我们的工具在一些常见的程序(片段)中合成了有用的不变式,包括:partition 程序(拷贝一个数组中小于等于和大于 x 的元素到两个数组中);insertion sort 的内循环;arrayMax 程序;一个版本的 firstNotNull 程序;1-d,2-d 和 3-d 数组的 arrayCopy 程序;1-d,2-d 和 3-d 数组的 arrayCheck 程序.关于更多的分析结果细节,可以访问工具和例子网址:https://github.com/libin049/InvariantSynthesisForArray_C.

表 3 给出了工具在处理上面例子时的性能结果.实验运行在 2.4GHz Intel 处理器、4GB RAM 电脑上.

Table 3 Performances results

表 3 性能结果

Procedure	Time (s)
partition	4.52
insertionSort(inner loop)	3.37
arrayMax	1.18
firstNotNull	2.28
Find	9.85
arrayPartCopy	0.51
1-dim arrayCopy	0.43
2-dim arrayCopy	5.65
3-dim arrayCopy	40.24
1-dim arrayCheck	0.56
2-dim arrayCheck	4.00
3-dim arrayCheck	52.30

表 3 说明:对于 1-d 数组程序,工具分析时间不超过 1 秒;对于多维数组,需要花费更多的时间去分析.这是因为对于多维数组程序,工具需要较多的时间分析全称量词是否会被语句杀死—这个过程需要使用 Z3 进行逻辑推导,相对而已比较耗时.

5.2 SV-COMP数组benchmark分析结果

Competition on Software Verification(SV-COMP)[14]提供了一个数组程序的 benchmark:array-examples[16]. array-examples 中的程序全部使用 GNU C 或者 ANSI C 描述.它有 88 个文件、2 299 行代码.绝大多数文件只有一个函数,并且所有的数组都是一维数组.SV-COMP 中的 array-examples benchmark 用来检查(不)可达性((un)reachability).在 88 个文件中,有 28 个文件具有错误的规约,其余的 60 个文件中有正确的规约.这些文件中,所有的规约都是不包含量词的原子断言.60 个包含正确规约的文件中,有 56 个关于数组元素的原子断言位于循环语句内部,这些原子断言等价于被遍历数组的全称量词性质.

表 4 展示了工具在 array-examples 上的执行时间(time)、文件数量(#file)、每个文件平均分析时间(time/#file)、array-examples 中循环的数量(#loop)、一共在多少循环入口发现了全称量词性质(#loop(U-inv))以及一共在多少文件中发现了全称量词性质(#file(U-inv)).表 4 指出每个文件平均分析时间是 10s,以及工具在 304 个循环入口、74 个文件中发现了全称量词.

Table 4 Analysis result of array-examples

表 4 Array-Examples 分析结果

Time (s)	#file	time/#file (s)	#loop	#loop (U-inv)	#file (U-inv)
981	88	11.2	375	304	74

我们比较了工具生成的全称量词性质和 60 个正确规约文件中在循环内部的原子断言.在 array-examples benchmark 总共 56 个这样的原子断言.我们的工具合成了 44 个等价的目标全称量词性质.我们分析了其余 12 个没有合成目标性质的原因.其中,有 2 个需要处理函数调用语句,有 4 个需要将形如$\forall k(k\in[e_1,1,e_2)\Rightarrow p(k))$的全称量词性质分解为针对两个子区间的全称量词性质,才能够分析得到目标全称量词.另外 4 个目标性质不在本文方法的数据流抽象域.例如,在 standard_copyInit_true-unreach-call_ground 程序(如图 8 所示)中,表达式 42+*incr* 只出现在原子断言中,没有出现在程序中,因此,$\forall x(x\in[0,1,N)\Rightarrow b[x]==42+incr)$不属于本文方法的抽象域.我们将在后续工作研究处理上面这几种情况的技术.

```
1   …
2   for (i=0; i<N; i++){
3      b[i]=b[i]+incr;
4   }
5            negative_sum=negative_sum+array[i];
6   int x;
7   for (x=0; x<N; x++){
8      _VERIFIER_assert(b[x]==42+incr);
9   }
10 }
```

Fig.8 standard_copyInit_true-unreach-call_ground

图 8 standard_copyInit_true-unreach-call_ground

需要指出:我们的方法虽然没有合成其余的 12 个目标性质,但是合成了可以用于推导出目标性质的全称量词性质.例如,在 standard_copyInit_true-unreach-call_ground 程序中,工具在第 2 行的 for 语句之前合成性质:$\{\forall x(x\in[0,1,N)\Rightarrow A[x]=B[x]),\forall x(x\in[0,1,N)\Rightarrow A[x]=42),\forall x(x\in[0,1,N)\Rightarrow B[x]=42)\}$.这些性质可以用于推导出最终的目标性质$\forall x(x\in[0,1,N)\Rightarrow B[x]=42+incr)$.

5.3 与其他工具比较

我们的工具与其他不变式合成工具进行了比较,这类的工作虽然有很多,比如文献[7,8,17−19],但是只找了一个可用的工具 CppInv[19].我们比较了它们合成 array-examples benchmark 中 56 个等价于全称量词性质的原子断言的个数与时间.除此之外,本文工具也与 BOOSTER[20],ESBMC[21],SMACK+Corral[22]进行了比较.其中,SMACK+Corral 和 ESBMC 是 SV-COMP2017 中关于 array-examples benchmark 得分最高的前两个工具,BOOSTER 能够直接验证量词性质.需要指出:BOOSTER, ESBMC 和 SMACK+Corral 是程序验证工具,它们需要给出待验证的性质.虽然 BOOSTER,ESBMC 和 SMACK+Corral 不是性质合成工具,但是本文工具与这些工具关于合成或验证 array-examples benchmark 中目标断言的个数和时间的比较结果可以间接反映本文方法的有效性和可行性.

对于所有的工具,我们比较这些工具是否可以验证或合成 array-examples benchmark 中 56 个等价于全称量词性质的原子断言的个数(第 2 列)、验证或合成的时间(第 3 列)、工具是否需要用户提供待验证的断言(第 4 列)、是否支持多维数组、是否能够保证分析/验证的性质一定是正确的(sound,第 5 列)以及是否直接支持合成或验证量词性质(第 6 列).因为 BOOSTER 支持量词性质验证,我们将 56 个断言转换成了等价的全称量词性质.每个工具在单个文件上的验证或合成时间最长是 15 分钟.

表 5 给出了比较结果.比较结果表明,我们的工具在合成性质的数量和时间上都优于其他工具.在这些工具中,因为 ESBMC 和 SMACK+Corral 基于有界模型检验技术,它们的正确性依赖于循环展开的次数;booster 也用到了有界模型检验技术,所以它们不能够保证验证结果一定是正确的.只有我们的工具和 CppInv 能够保证合成

的性质一定是正确的,我们的工具在合成性质的数量和时间上都明显优于 CppInv,并且能够处理多维数组.

Table 5 Synthesis or verification results of CppInv, BOOSTER, ESBMC, SMACK+Corral and our tool

表 5 CppInv,BOOSTER,ESBMC,SMACK+Corral 及我们的工具合成或验证结果

工具	合成或验证的数量	时间(s)	是否需要提供断言	是否能处理多维数组	是否 sound	是否直接支持量词性质
CppInv	27	15 163	否	否	是	是
BOOSTER	36	1 855	是	否	否	是
ESBMC	39	10 989	是	是	否	否
SMACK+Corral	44	10 310	是	是	否	否
我们的工具	44	703	否	是	是	是

6 相关工作

合成数组程序不变式的相关工作可分为如下几类:数组展开(array expansion)方法、Array smashing 方法、数组划分方法、谓词抽象方法、基于模板的方法、基于定理证明的方法.

- 数组展开方法[23].这个方法展开循环和数组单元来处理程序.数组展开方法优点是精确,缺点是只能处理小尺寸的数组,不能处理无界数组;
- Array smashing 方法[5,24].这个方法将整个数组 A 看作一个变量 a.初始时,a 被赋予一个整个数组单元都满足的已知最强的性质.$A[i]:=e$ 按照弱赋值 $a\sqcup:=e$ 来处理.Array smashing 方法的缺点是弱赋值(weak assignment$\sqcup$:=)会丢失信息,并且无法处理关于数组单元的条件测试语句.因此,这个方法获得的结果常常不精确;
- 数组划分方法[7,8,17].数组划分方法将索引域([1...n])划分为若干符号区间(e.g.,I_1=[1...i−1],I_2=[i,i],I_3=[i+1...n]),并且给每个子数组 $A[I_k]$关联一个辅助摘要变量(summary auxiliary variable)a_k.该方法基于语法启发式[7,8]或者预分析[17]的方法对索引域进行划分.这类方法与本文的方法类似.区别在于:本文的方法不是给每个子数组关联一个摘要变量,而是直接将一个数组性质表示为一个或多个全称量词性质.相对于数组划分方法,本文的方法具有更高的表达能力和灵活性.因此,本文的方法可以非常容易地处理多维数组性质,但是数组划分方法难以处理多维数组.此外,数组划分方法无法处理在循环中首先进行索引更新的程序(例如图 1(b)所示的 another arrayPartCopy 程序),然而在循环中首先进行索引更新的写法在实际编码中经常使用(例如 $A[++i]$).本文的方法可以处理这类程序;
- 谓词抽象方法[1,25,26].谓词抽象方法会使用一些语法启发式策略推导用于抽象的谓词,或者由用户提供谓词.它将程序抽象到谓词程序,然后基于谓词程序合成数组性质.反例制导的精化[3]和 Craig 插值[4]提供了获取谓词的其他方法,提高了谓词抽象方法的效果.谓词抽象方法中,因为缺少有效的选择算法,因此一些原子谓词常常必须手动提供.本文的方法不需要用户提供任何输入.此外,谓词抽象方法需要使用约束求解器检查猜出来的量词性质是否真的成立,而目前约束求解器对量词性质的支持有限,因此难以处理较复杂的量词性质.本文的方法只使用到了约束求解器的线性约束求解功能,而约束求解器(如 Z3)对线性求解功能支持比较好.因此相对于这类方法,本文的方法能够自动地分析得到更复杂的量词性质;
- 基于模板的方法[9,27].基于模板的方法非常有用但是代价昂贵,其基本思想是:用户提供所需要不变式的模板,然后分析过程去搜索实例化模板参数.这个方法的过程需要用户的参与,本文的方法是完全自动化的;
- 基于定理证明的方法[18,28].这类方法使用定理证明器去生成循环不变式.文献[18]方法的基本思想是:将数组第 i 次迭代中的修改编码为一个量化事实(quantified fact),然后运用定理证明器去推导一个封闭形式的公式.这类方法的缺点是受限于使用的定理证明器,同时可证明的性质比较有局限性.

7 总结和未来工作

本文提出了一个使用抽象解释框架自动合成数组性质的方法.该方法的特点在于抽象域是性质集合(性质包括全称量词性质和原子性质),这种抽象域灵活并且具有较高的表达能力.该方法通过前向迭代数据流分析合成数组性质.本文证明了该方法是收敛的,并且获得的不变式是正确的.本文也展示了方法具有一定的灵活性,可以通过增加抽象域和传播规则来增加分析能力.本文所述分析方法的工具原型基于 clang 和 Z3 实现.该工具被用于分析一些常见程序以及 SV-COMP 的 array-examples benchmark.array-examples benchmark 共有 88 个文件中,我们的工具在 74 个文件中发现了全称量词性质.

在未来工作中,我们计划处理全称量词性质分裂和合成的情况(一个全称量词性质分为若干全称量词性质,或者将若干全称量词性质合成一个全称量词性质).一个更长远的目标是处理数组和链表的全称量词和存在量词性质.

References:

[1] Flanagan C, Qadeer S. Predicate abstraction for software verification. ACM SIGPLAN Notices, 2002,37(1):191−202. [doi: 10.1145/565816.503291]

[2] Lahiri SK, Bryant RE. Indexed predicate discovery for unbounded system verification. In: Alur R, Peled DA, eds. Proc. of the Computer Aided Verification. Berlin: SpringerVerlag, 2004. 135−147. [doi: 10.1007/978-3-540-27813-9_11]

[3] Beyer D, Henzinger TA, Majumdar R, Rybalchenko A. Path invariants. ACM SIGPLAN Notices, 2007,42(6):300−309. [doi: 10.1145/1273442.1250769]

[4] Jhala R, McMillan KL. Array abstractions from proofs. In: Damm W, Hermanns H, eds. Proc. of the Computer Aided Verification. Berlin: Springer-Verlag, 2007. 193−206. [doi: 10.1007/978-3-540-73368-3_23]

[5] Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. A static analyzer for large safety-critical software. ACM SIGPLAN Notices, 2003,38(5):196−207. [doi: 10.1145/780822.781153]

[6] Gopan D. Numeric program analysis techniques with applications to array analysis and library summarization [Ph.D. Thesis]. Madison: University of Wisconsin, 2007.

[7] Gopan D, Reps T, Sagiv M. A framework for numeric analysis of array operations. ACM SIGPLAN Notices, 2005,40(1):338−350. [doi: 10.1145/1047659.1040333]

[8] Halbwachs N, Péron M. Discovering properties about arrays in simple programs. ACM SIGPLAN Notices, 2008,43(6):339−348. [doi: 10.1145/1379022.1375623]

[9] Gulwani S, McCloskey B, Tiwari A. Lifting abstract interpreters to quantified logical domains. ACM SIGPLAN Notices, 2008, 43(1):235−246. [doi: 10.1145/1328897.1328468]

[10] Kam JB, Ullman JD. Monotone data flow analysis frameworks. Acta Informatica, 1977,7(3):305−317. [doi: 10.1007/BF00290339]

[11] Zhao JH, Li XD. Scope logic: An extension to hoare logic for pointers and recursive data structures. In: Liu ZM, ed. Proc. of the Theoretical Aspects of Computing (CICTAC 2013). Shanghai: Springer-Verlag, 2013. 409−426. [doi: 10.1007/978-3-642-39718-9_24]

[12] http://clang.llvm.org/

[13] De Moura L, Bjrner N. Z3: An efficient smt solver. In: Ramakrishnan CR, Rehof J, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2008. 337−340. [doi: 10.1007/978-3-540-78800-3_24]

[14] http://sv-comp.sosy-lab.org/2017/index.php

[15] Hoare CA. Proof of a program: Find. Communications of the ACM, 1971,14(1):39−45. [doi: 10.1145/362452.362489]

[16] https://github.com/sosy-lab/sv-benchmarks/releases/tag/svcomp17

[17] Cousot P, Cousot R, Logozzo F. A parametric segmentation functor for fully automatic and scalable array content analysis. ACM SIGPLAN Notices, 2011,46(1):105−118. [doi: 10.1145/1925844.1926399]

[18] Kovács L, Voronkov A. Finding loop invariants for programs over arrays using a theorem prover. In: Chechik M, Wirsing M, eds. Proc. of the Fundamental Approaches to Software Engineering. Berlin: Springer-Verlag, 2009. 470−485. [doi: 10.1007/978-3-642-00593-0_33]

[19] Daniel L, Rodríguez-Carbonell E, Rubio A. SMT-Based array invariant generation. In: Giacobazzi R, Berdine J, Mastroeni I, eds. Proc. of the Verification, Model Checking, and Abstract Interpretation. Berlin: Springer-Verlag, 2013. 169−188. [doi: 10.1007/978-3-642-35873-9_12]

[20] Francesco A, Ghilardi S, Sharygina N. Decision procedures for flat array properties. In: Ábrahám E, Havelund K, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-averlag, 2014. 15−30. [doi: 10.1007/978-3-642-54862-8_2]

[21] Cordeiro L, Fischer B, Marques-Silva J. SMT-Based bounded model checking for embedded ANSI-C software. IEEE Trans. on Software Engineering, 2012,38(4):957−974. [doi: 10.1109/TSE.2011.59]

[22] Haran A, Carter M, Emmi M, Lal A, Qadeer S, Rakamarić Z. Smack+ Corral: A modular verifier. In: Baier C, Tinelli C, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2015. 451−454. [doi: 10.1007/978-3-662-46681-0_42]

[23] Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. Design and implementation of aspecial-purpose static program analyzer for safety-critical real-time embedded software. In: Mogensen TÆ, Schmidt DA, Sudborough IH, eds. Proc. of the Essence of Computation. Berlin: Springer-Verlag, 2002. 85−108. [doi: 10.1007/3-540-36377-7_5]

[24] Gopan D, DiMaio F, Dor N, Reps T, Sagiv M. Numeric domains with summarized dimensions. In: Jensen K, Podelski A, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2004. 512−529. [doi: 10.1007/978-3-540-24730-2_38]

[25] Lahiri SK, Bryant RE. Constructing quantified invariants via predicate abstraction. In: Steffen B, Levi G, eds. Proc. of the Verification, Model Checking, and Abstract Interpretation. Berlin: Springer-Verlag, 2004. 267−281. [doi: 10.1007/978-3-540-24622-0_22]

[26] Lahiri SK, Bryant RE, Cook B. A symbolic approach to predicate abstraction. In: Hunt WA, Somenzi F, eds. Proc. of the Computer Aided Verification. Berlin: Springer-Verlag, 2003. 141−153. [doi: 10.1007/978-3-540-45069-6_15]

[27] Srivastava S, Gulwani S. Program verification using templates over predicate abstraction. ACM SIGPLAN Notices, 2009,44(6): 223−234. [doi: 10.1145/1543135.1542501]

[28] McMillan KL. Quantified invariant generation using an interpolating saturation prover. In: Ramakrishnan CR, Rehof J, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2008. 413−427. [doi: 10.1007/978-3-540-78800-3_31]

附录:证明

定理1的证明

证明文中的定理 1 之前,首先给出下面的推论及引理.

推论 1. *ProperP* 的势(大小)是有穷的.

推论 2. 对任何性质 p_1,p_2 和 p_3:

(1) 如果 $p_1 \preceq p_2$ 并且 $p_2 \preceq p_3$,那么 $p_1 \preceq p_3$;

(2) 如果 $p_1 \preceq p_2$ 并且 $p_2 \preceq p_1$,那么 $p_1 = p_2$;

(3) 如果 $p_1 \preceq p_2$,那么 $M(p_1) \subseteq M(p_2)$(注意,M(false)是$\varnothing$).

推论 3. 对于任何性质 p_1,p_2 和 p_3:

(1) 如果 $p_1 \bar{\sqcap} p_2 \neq \bot$:

$$\begin{array}{l} p_1 \bar{\sqcap} p_2 \preceq p_1 \wedge p_1 \bar{\sqcap} p_2 \preceq p_2 \\ p_1 \vee p_2 \Rightarrow p_1 \bar{\sqcap} p_2 \end{array};$$

(2) $p_1 \preceq p_2 \Leftrightarrow p_1 = p_1 \bar{\sqcap} p_2$;

(3) $p_1 \preceq p_2 \wedge p_1 \preceq p_3 \Rightarrow p_1 \preceq p_2 \bar{\sqcap} p_3$.

推论 4. 对任何性质 p_1,p_2 和 p_3,$\bar{\sqcap}$满足下面的性质:

(1) $p_1 \overline{\sqcap} p_1 = p_1$;

(2) $p_1 \overline{\sqcap} p_2 = p_2 \overline{\sqcap} p_1$;

(3) $(p_1 \overline{\sqcap} p_2) \overline{\sqcap} p_3 = p_1 \overline{\sqcap} (p_2 \overline{\sqcap} p_3)$.

推论 5. 对任何 $S_1,S_2 \in 2^{ProperP}$:

(1) $Reduce(S_1) \in L_G$;

(2) $S_1 \sqsubseteq Reduce(S_1) \wedge Reduce(S_1) \sqsubseteq S_1$;

(3) $S_1 \sqsubseteq S_2 \Rightarrow Reduce(S_1) \sqsubseteq Reduce(S_2)$.

引理 1. 对任何 $L_1,L_2 \in L_G, L_1 \sqcap_G L_2 \in L_G$.

证明:结论直接来自$\sqcap_G$的定义. □

引理 2. 对任何 $S_1,S_2,S_3 \in 2^{ProperP}$:

(1) $S_1 \sqcap_G S_2 \sqsubseteq S_1 \wedge S_1 \sqcap_G S_2 \sqsubseteq S_2$;

(2) $S_1 \sqsubseteq S_2 \wedge S_1 \sqsubseteq S_3 \Rightarrow S_1 \sqsubseteq S_2 \sqcap_G S_3$.

证明:根据推论 3,对于任何性质 $p_1 \in S_1, p_2 \in S_2, p_1 \overline{\sqcap} p_2 \in S_2 \sqcap_G S_2$,并且 $p_1 \overline{\sqcap} p_2 \preceq p_1 \wedge p_1 \overline{\sqcap} p_2 \preceq p_2$.因此,$S_1 \sqcap_G S_2 \sqsubseteq S_1 \wedge S_1 \sqcap_G S_2 \sqsubseteq S_2$ 成立.$S_1 \sqsubseteq S_2 \wedge S_1 \sqsubseteq S_3$ 推出$\forall p_1 \in S_1, \exists p_2 \in S_2, \exists p_3 \in S_3 \Rightarrow p_1 \preceq p_2 \wedge p_1 \preceq p_3$.根据推论 3,$S_1 \sqsubseteq S_2 \sqcap_G S_3$ 成立. □

引理 3. 对任何 $L_1,L_2 \in L_G, L_1 \sqsubseteq L_2 \wedge L_2 \sqsubseteq L_1 \Leftrightarrow L_1 = L_2$.

证明:$L_1 \sqsubseteq L_2 \wedge L_2 \sqsubseteq L_1$ 推出$\forall p_1 \in L_1, \exists p_2 \in L_2, \exists p_3 \in L_1 \Rightarrow p_1 \preceq p_2 \wedge p_2 \preceq p_3$.$p_1 \preceq p_2 \wedge p_2 \preceq p_3 \wedge L_1 \in L_G \Rightarrow p_1 = p_3 \Rightarrow p_1 \preceq p_2 \wedge p_2 \preceq p_1 \Rightarrow p_1 = p_2$.因此$\forall p_1 \in L_1, \exists p_2 \in L_2 \Rightarrow p_1 = p_2 \Rightarrow p_1 \in L_2$.同理可证,$\forall p_2 \in L_2, \exists p_1 \in L_1 \Rightarrow p_1 = p_2 \Rightarrow p_2 \in L_1$.因此,$L_1 = L_2$. □

引理 4. 对任何 $L_1,L_2 \in L_G, L_1 \sqsubseteq L_2 \Leftrightarrow L_1 = L_1 \sqcap_G L_2$.

证明:要证明 $L_1 \sqsubseteq L_2 \Leftrightarrow L_1 = L_1 \sqcap_G L_2$,只需证明:

1) $L_1 \sqsubseteq L_2 \Rightarrow L_1 = L_1 \sqcap_G L_2$;

2) $L_1 = L_1 \sqcap_G L_2 \Rightarrow L_1 \sqsubseteq L_2$.

根据引理 2,结论 1)可证.令 $S(L_1,L_2)=\{p_1 \overline{\sqcap} p_2 | p_1 \in L_1 \wedge p_2 \in L_2 \wedge p_1 \overline{\sqcap} p_2 \neq \bot\}$.根据推论 5,$L_1 \sqsubseteq L_2 \Rightarrow L_1 \sqsubseteq S(L_1,L_2) \Rightarrow L_1 \sqsubseteq Reduce(S(L_1,L_2)) \Rightarrow L_1 \sqsubseteq L_1 \sqcap_G L_2$.根据引理 2,$L_1 \sqcap_G L_2 \sqsubseteq L_1$.根据引理 3,$L_1 \sqsubseteq L_1 \sqcap_G L_2 \wedge L_1 \sqcap_G L_2 \sqsubseteq L_1 \wedge L_1 \in L_G \wedge L_1 \sqcap_G L_2 \in L_G$ 推出 $L_1 = L_1 \sqcap_G L_2$.结论 2)成立. □

定理 1. $(L_G, \sqcap_G)$是关于 CFG G 的一个交半格(meet semi-lattice),并且格的高度有穷.

证明:$(L_G, \sqcap_G)$是交半格当且仅当,对所有 $L_1,L_2,L_3 \in L_G$:

1) $L_1 \sqcap_G L_1 = L_1$;

2) $L_1 \sqcap_G L_2 = L_2 \sqcap_G L_1$;

3) $(L_1 \sqcap_G L_2) \sqcap_G L_3 = L_1 \sqcap_G (L_2 \sqcap_G L_3)$;

4) $L_1 \sqsubseteq L_2 \Leftrightarrow L_1 = L_1 \sqcap_G L_2$.

结论 1)、结论 2)和结论 4)通过引理 4 和$\sqcap_G$定义可证.令 $S(L_1,L_2)=\{p_1 \overline{\sqcap} p_2 | p_1 \in L_1 \wedge p_2 \in L_2 \wedge p_1 \overline{\sqcap} p_2 \neq \bot\}$.要证明结论 3),只需要证明:

a) $(L_1 \sqcap_G L_2) \sqcap_G L_3 = Reduce(S(S(L_1,L_2),L_3))$;

b) $Reduce(S(S(L_1,L_2),L_3)) = Reduce(S(L_1,S(L_2,L_3)))$;

c) $Reduce(S(L_1,S(L_2,L_3))) = L_1 \sqcap_G (L_2 \sqcap_G L_3)$.

$(L_1\sqcap_G L_2)\sqcap_G L_3\sqsubseteq L_1\sqcap_G L_2\sqsubseteq S(L_1,L_2)$.根据引理 2,$(L_1\sqcap_G L_2)\sqcap_G L_3\sqsubseteq S(L_1,L_2)\sqcap_G L_3$.同理可以证明 $S(L_1,L_2)\sqcap_G L_3\sqsubseteq(L_1\sqcap_G L_2)\sqcap_G L_3$.因此,$(L_1\sqcap_G L_2)\sqcap_G L_3=S(L_1,L_2)\sqcap_G L_3=Reduce(S(S(L_1,L_2),L_3))$.同理可以证明 $Reduce(S(L_1,S(L_2,L_3)))=L_1\sqcap_G(L_2\sqcap_G L_3)$.根据函数 S 的定义,$S(S(L_1,L_2),L_3)=S(L_1,S(L_2,L_3))$可证.因此,结论 3)可证.$L_G=\{S|S\subseteq ProperP\wedge\forall p_1,p_2\in S,p_1\preceq p_2\Rightarrow p_1=p_2\}$.因此 $L_G\subseteq 2^{ProperP}$.根据推论 1,$ProperP$ 是有穷的,因此$(L_G,\sqcap_G)$的高度有穷. □

定理2的证明

证明文中的定理 2 之前,首先证明下面的引理:

引理 5. 对任何 $L_1,L_2\in L_G$,如果 $L_1\sqsubseteq L_2$,那么 $Semantics(n,L_1)\sqsubseteq Semantics(n,L_2)$.

证明:根据 *Semantics* 的定义:

(1) 如果 n 是 $cond$,那么结论显然成立;

(2) 如果 n 是 $lh:=e$,对任何 $lh_i\in LH$,如果$\wedge L_1\Rightarrow\&lh\neq\&lh_i$,那么$\wedge L_2\Rightarrow\&lh\neq\&lh_i$.因此:

$$Semantics(n,L_1)\sqsubseteq Semantics(n,L_2).$$

综上,$Semantics(n,L_1)\sqsubseteq Semantics(n,L_2)$. □

引理 6. 对任何性质集合 S_1 和 S_2,如果 $S_1\sqsubseteq S_2$,那么 $Propagated(S_1)\sqsubseteq Propagated(S_2)$.

证明:对任何 $ins_1\subseteq S_1$,如果规则 r 可以应用 ins_1,并且产生输出性质 out_1,根据 $S_1\sqsubseteq S_2$ 和传播规则约束,一定存在 $ins_2\subseteq S_2$ 和一个对应的规则 r',使得:(a) $ins_1\sqsubseteq ins_2$;(b) r'可以应用到 ins_2;(c) r'和 ins_2 产生性质 out_2 并且 $out_1\preceq out_2$.因此,$Propagated(S_1)\sqsubseteq Propagated(S_2)$. □

引理 7. 对任何 $x,y\in L_G$,如果 $x\sqsubseteq y$,$Transfer(e_1[e_2]:=e_3,x)\sqsubseteq Transfer(e_1[e_2]:=e_3,y)$.

证明:令 $TS(x)=\{p|p\in x\wedge p$ 是全称量词$\wedge\&e_1[e_2]\notin M(p)\}$.根据 *Transfer* 函数的定义,只需要证明 $TS(x)\sqsubseteq TS(y)$.对任何 $p_1\in TS(x)$,肯定存在 $p_2\in y$ 并且 $p_1\preceq p_2$.根据推论 2,$M(p_1)\subseteq M(p_2)$.$\wedge x\Rightarrow\&e_1[e_2]\notin M(p_1)$推出$\wedge y\Rightarrow\&e_1[e_2]\notin \boldsymbol{M}(p_2)$,因此,$p_2\in TS(y)$.综上,$TS(x)\sqsubseteq TS(y)$. □

引理 8. 对任何 $x,y\in L_G$,如果 n 是 $i:=i+c$,$x\sqsubseteq y$,那么 $HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$.

证明:下面分情况讨论:

(1) 如果$\forall k(k\in[init_i,c,i+c)\Rightarrow p)$在 x 中,那么肯定存在$\forall k(k\in[init_i,c,i+c)\Rightarrow p')$在 y 中,因为$\{\forall k(k\in[init_i,c,i)\Rightarrow p)\}\sqsubseteq\{\forall k(k\in[init_i,c,i)\Rightarrow p')\}$,因此 $HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$;

(2) 如果$\forall k(k\in[init_i,c,i)\Rightarrow p)$在 x 中,那么肯定存在$\forall k(k\in[init_i,c,i)\Rightarrow p')$在 y 中,因为$\{\forall k(k\in[init_i,c,i-c)\Rightarrow p)\}\sqsubseteq\{\forall k(k\in[init_i,c,i-c)\Rightarrow p')\}$,因此 $HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$;

(3) 如果 $i=init_i$ 在 x 中,那么肯定存在 $i=init_i$ 在 y 中,因此 $HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$;

(4) 其他情况,$HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$.

综上,$HandleInterval(n,x)\sqsubseteq HandleInterval(n,y)$. □

引理 9. 对任何 $S_1,S_2\in 2^{ProperP}$,如果 $S_1\sqsubseteq S_2$,那么 $GenAQ(S_1)\sqsubseteq GenAQ(S_2)$.

证明:令 $\psi(k)$表示 $\psi(\ldots,e_1[f_1(k)],\ldots)$的缩写:

(1) 如果在 x 中条件(1)成立,$GenAQ(S_1)=S_1\cup\{\forall k(k\in[init_i,c,i+c)\Rightarrow\psi(k))\}$.$S_1\sqsubseteq S_2$ 推出 $i=init_i\in S_2$ 并且$\forall k(k\in[init_i,c,i)\Rightarrow\text{false})$在 S_2 中并且 $\psi'(i)\in S_2$,其中,$\psi(i)\preceq\psi'(i)$.$GenAQ(S_2)=S_2\cup\{\forall k(k\in[init_i,c,i+c)\Rightarrow\psi'(k))\}$.因此,$GenAQ(S_1)\sqsubseteq GenAQ(S_2)$;

(2) 如果在 x 中条件(2)成立,$GenAQ(S_1)=S_1\cup\{\forall k(k\in[init_i,c,i+c),\psi(k)\}$.$S_1\sqsubseteq S_2$ 推出$\forall k(k\in[init_i,c,i)\Rightarrow\psi_1'(k))$在

S_2 中并且 $\psi'_2(i)\in S_2$,其中,$\psi_1(k)\preceq\psi'_1(k)\wedge\psi_2(i)\preceq\psi'_2(i)$.$\psi(k)=\psi_1(k)\sqcap\psi_2(k)\Rightarrow\psi(k)\preceq\psi_1(k)\wedge\psi(k)\preceq\psi_2(k)\Rightarrow\psi(k)\preceq\psi'_1(k)\wedge\psi(k)\preceq\psi'_2(k)\Rightarrow\psi(k)\preceq\psi'_1(k)\sqcap\psi'_2(k)$.令 $\psi'(k)=\psi'_1(k)\sqcap\psi'_2(k)$.$GenAQ(S_2)=S_2\cup\{\forall k(k\in[init_i,c,i+c)\Rightarrow\psi'(k))\}$.因此,$GenAQ(S_1)\sqsubseteq GenAQ(S_2)$;

(3) 如果在 x 中条件(3)和条件(4)成立,证明过程类似条件(1)和条件(2);

(4) 其他情况,$GenAQ(S_1)\sqsubseteq GenAQ(S_2)$显然成立.

综上,$GenAQ(S_1)\sqsubseteq GenAQ(S_2)$. □

推论 6. 如果 n 是循环控制变量初始化语句,$GenSpecial(n)\sqsubseteq GenSpecial(n)$.

推论 7. 令 G 表示 CFG.令 a 表示语句 n 之前的数据流值.如果 $a\in L_G$,那么 $F_n(a)\in L_G$.

定理 2. 令 n 表示 CFG G 的语句.F_n 是单调的.

证明:F_n 是单调的当且仅当$\forall x,y\in L_G:x\sqsubseteq y\Rightarrow F_n(x)\sqsubseteq F_n(y)$.根据 F_n 的定义,只需要证明 F_n 中出现的函数都是单调的.根据引理 5~引理 9 和推论 6、推论 7,F_n 是单调的. □

定理4的证明

引理 10. 令 c_i 表示语句 n 之前的状态,令 a_i 表示语句 n 之前的数据流值.如果 $G\vdash c_i\rightsquigarrow c_{i+1}$ 并且$[\![a_i]\!](c_i)$成立,那么下面的条件成立:

1) $[\![Semantics(n,a_i)]\!](c_{i+1})$成立;

2) 如果 n 是 $e_1[e_2]:=e_3$,那么$[\![Transfer(n,a_i)]\!](c_{i+1})$成立;

3) 如果 n 是循环控制变量初始化语句 $i:=init$,那么$[\![GenSpecial(n)]\!](c_{i+1})$成立;

4) 如果 n 是循环控制变量初始化语句 $i:=i+c$,那么$[\![HandleInterval(n,a_i)]\!](c_{i+1})$成立;

5) 对任何 $S\in 2^{ProperP}$,如果$[\![S]\!](c_{i+1})$成立,那么$[\![GenAQ(S)]\!](c_{i+1})$成立;

6) 对任何 $S\in 2^{ProperP}$,如果$[\![S]\!](c_{i+1})$成立,那么$[\![Recude(S)]\!](c_{i+1})$.

证明:

(1) 要证明结论 1),需要证明:

a) 如果 n 是 $lh:=e$,那么 $[\![lh]\!](c_{i+1})=[\![e]\!](c_i)\wedge\bigwedge_{[\![\&lh\neq\&lh_i]\!](c_i)\wedge lh_i\in LH}[\![lh_i]\!](c_{i+1})=[\![lh_i]\!](c_i)$ 成立;

b) 如果 n 是 $cond$,并且:

i c_{i+1} 是 true 分支语句之前的状态,那么$[\![cond]\!](c_{i+1})\wedge\bigwedge_{lh_i\in LH}[\![lh_i]\!](c_{i+1})=[\![lh_i]\!](c_i)$;

ii c_{i+1} 是 false 分支语句之前的状态,那么$[\![\neg cond]\!](c_{i+1})\wedge\bigwedge_{lh_i\in LH}[\![lh_i]\!](c_{i+1})=[\![lh_i]\!](c_i)$.

根据语句语义的定义,结论 a)、结论 b)成立.

(2) 要证明结论 2),需要证明:

$$[\![\{p|p\in a\wedge p\text{ 是全称量词性质}\wedge\&e_1[e_2]\notin M(p)\}]\!](c_{i+1}).$$

$[\![a_i]\!](c_i)\wedge p\in a_i\Rightarrow[\![p]\!](c_i)$.$(\wedge a_i\Rightarrow\&e_1[e_2]\notin M(p))\wedge[\![a_i]\!](c_i)$推出$[\![\&e_1[e_2]\notin M(p)]\!](c_i)$.因为$[\![\&e_1[e_2]\notin M(p)]\!](c_i)$,$p$ 中所有内存单元的值保持不变,因此$[\![p]\!](c_{i+1})$成立.

(3) 为了证明结论 3),需要证明$[\![\{\forall k(k\in[init,c,i)\Rightarrow\text{false}\}]\!](c_{i+1})$.因为 n 是 $i:=init$ 并且 i 不出现在 $init$ 中,所以$[\![i=init]\!](c_{i+1})$成立.$[\![i=init]\!](c_{i+1})\Rightarrow[\![[init,step,i)=\varnothing]\!](c_{i+1})$,因此$[\![\forall k(k\in[init,c,i)\Rightarrow\text{false}]\!](c_{i+1})$成立.

(4) 为了证明结论 4):

a) 如果$\forall x(x\in[init_i,c,i+c)\Rightarrow p)$在 a_i 中,那么只需要证明$[\![\forall x(x\in[init_i,c,i)\Rightarrow p)]\!](c_{i+1})$.$(\forall x(x\in[init_i,c,i+c)\Rightarrow p)$在 a_i 中并且$[\![a_i]\!](c_i)$推出$[\![\forall x(x\in[init_i,c,i+c)\Rightarrow p]\!](c_i)$.因为 n 是 $i:=i+c$,p 不包含 i,$init_i$ 不包含 i,并且$[\![i]\!](c_{i+1})=[\![i+c]\!](c_i)$,所以$[\![\forall x(x\in[init_i,c,i)\Rightarrow p]\!](c_{i+1})$;

b) 如果$(\forall x(x\in[init_i,c,i)\Rightarrow p)$在 a_i 中,那么只需要证明$\forall x(x\in[init_i,c,i-c)\Rightarrow p)(c_{i+1})$.$\forall x(x\in[init_i,c,i)\Rightarrow p)$在 a_i 中

$\wedge[\![a_i]\!](c_i)$推出$[\![\forall x(x\in[init_i,c,i)\Rightarrow p]\!](c_i)$.因为 n 是 $i:=i+c,p$ 不包含 $i,init_i$ 不包含 i,并且$[\![i-c]\!](c_{i+1})=[\![i]\!](c_i)$,所以$[\![\forall x(x\in[init_i,c,i-c)\Rightarrow p)]\!](c_{i+1})$;

c) 如果$(i=init_i)$在 a_i 中,那么只需要证明$[\![i-c=init_i]\!](c_{i+1})$.$(i=init_i)\in a_i\wedge[\![a_i]\!](c_i)$推出$[\![i=init_i]\!](c_i)$.因为 n 是 $i:=i+c,init_i$ 不包含 i,并且$[\![i-c]\!](c_{i+1})=[\![i]\!](c_i)$,所以$[\![i-c=init_i]\!](c_{i+1})$;

d) 其他情况,结论 4)显然成立.

综上,结论 4)成立.

(5) 为了证明结论 5):

a) 如果第 4.3.5 节中的条件(1)或者条件(2)成立,那么只需要证明$[\![\{\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(...,e_1[f_1(x)],...)\}]\!](c_{i+1})$成立,其中,$(...,e_1[f_1(x)],...)$不包含 i.令 $\psi(k)$表示 $\psi(...,e_1[f_1(x)],...)$的缩写:

1) 如果条件(1)成立,那么$(\wedge S\Rightarrow(1))\wedge[\![S]\!](c_{i+1})$推出$[\![(1)]\!](c_{i+1})$.$[\![(1)]\!](c_{i+1})\Rightarrow[\![i=init_i\wedge\psi(i)]\!](c_{i+1})$.因为$[\![i=init_i]\!](c_{i+1})$成立,那么$[\![[init_i,c,i+c)]\!](c_{i+1})=[\![\{i\}]\!](c_{i+1})$.因此,$[\![\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(x)]\!](c_{i+1})$成立;

2) 如果条件(2)成立,那么$(\wedge S\Rightarrow(2))\wedge[\![S]\!](c_{i+1})$推出$[\![(2)]\!](c_{i+1})$.因为$[\![(2)]\!](c_{i+1})$,因此$[\![\forall x(x\in[init_i,c,i)\Rightarrow\psi_1(x)\wedge\psi_2(i)]\!](c_{i+1})$成立.因为 $\psi(x)=\psi_1(x)\sqcap\psi_2(x)$,所以 $\psi_2(x)\Rightarrow\psi(x)\wedge\psi_1(x)\Rightarrow\psi(x)$成立.因此,$\forall x(x\in[init_i,c,i)\Rightarrow\psi_1(x))$推出$\forall x(x\in[init_i,c,i)\Rightarrow\psi(x)$.$[\![\forall x(x\in[init_i,c,i)\Rightarrow\psi_1(x)\wedge\psi_2(i)]\!](c_{i+1})$推出$[\![\forall x(x\in[init_i,c,i)\Rightarrow\psi(x)\wedge\psi(i)]\!](c_{i+1})$.因为$[\![[init_i,c,i+c)]\!](c_{i+1})=[\![[init_i,c,i)\cup\{i\}]\!](c_{i+1})$,故$[\![\forall x(x\in[init_i,c,i+c)\Rightarrow\psi(x)]\!](c_{i+1})$成立;

b) 如果第 3.3.6 节中的条件(3)或者条件(4)成立,那么只需要证明$[\![\{\forall x(x\in[init_i,c,i)\Rightarrow\psi(...,e_1[f_1(x)],...)\}]\!](c_{i+1})$成立,其中,$(...,e_1[f_1(x)],...)$不包含 i.它的证明过程类似证明条件(1)和条件(2).

综上,结论 5)成立.

(6) $[\![S]\!](c_{i+1})\Rightarrow[\![Reduce(S)]\!](c_{i+1})$,因此结论 6)成立. □

定理 4. 令 c_i 表示语句 n 之前的状态,a_i 表示语句 n 之前的数据流值.如果 $G\vdash c_i\rightsquigarrow c_{i+1}$ 并且$[\![a_i]\!](c_i)$成立,那么$[\![F_n(a_i)]\!](c_{i+1})$成立.

证明:传播规则正确,所以$[\![Semantics(n,a_i)]\!](c_i\cup c_{i+1})$成立,因此$[\![Propagated(a_i\cup Semantics(n,a_i)]\!](c_{i+1})$成立.根据引理 10 以及 F_n 的定义,$[\![F_n(a_i)]\!](c_{i+1})$成立. □

李彬(1988－),男,河北邯郸人,博士生,主要研究领域为软件工程,程序分析,程序验证.

翟娟(1988－),女,博士,CCF 专业会员,主要研究领域为软件工程,程序分析,程序验证,程序合成.

汤震浩(1989－),男,博士生,主要研究领域为软件工程,程序分析,程序验证.

汤恩义(1982－),男,博士,助理研究员,CCF 专业会员,主要研究领域为软件工程,新型软件测试方法与程序分析方法.

赵建华(1971－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为形式化方法,软件工程,程序设计语言.