基于指向与数值抽象的带指针算术程序的分析方法

尹帮虎 陈立前 王 戟
(国防科学技术大学计算机学院并行与分布处理国防科技重点实验室 长沙 410073)

摘 要 带指针算术的程序往往包含数组越界、缓冲区溢出等运行时错误。单纯的指针分析技术和数值分析技术都无法有效处理指针算术。为了将指针分析与数值分析相结合,首先提出一种新的指针内存模型,然后基于该模型设计了一个刻画指针指向关系和指针偏移量的抽象域。最后在抽象解释框架下,设计并实现了一个面向带指针算术C程序的静态分析工具原型PAA。实验结果表明,PAA能够有效地分析指针程序的指向关系和数值性质,并能够在效率和精度间取得合理的权衡。

关键词 静态分析,抽象解释,指向分析,数值抽象域
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.7.008

Analysis of Programs with Pointer Arithmetic by Combining Points-to and Numerical Abstractions

YIN Bang-hu CHEN Li-qian WANG Ji
(National Laboratory for Parallel and Distributed Processing, School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract Programs with pointer arithmetic often involve runtime errors such as array out of bound, buffer overflow, etc. Pure pointer analysis and pure numerical analysis cannot deal with pointer arithmetic. To combine pointer analysis and numerical analysis, we proposed a new pointer memory model. On this basis, we presented an abstract domain to capture points-to and offset information of pointers. Finally, under the framework of abstract interpretation, we implemented a static analyzer prototype named PAA for analyzing C programs with pointer arithmetic. Experimental results show that PAA can analyze points-to and numerical properties of programs with pointer arithmetic effectively. Moreover, PAA can achieve a reasonable trade-off between efficiency and accuracy.

Keywords Static analysis, Interpretation, Points-to analysis, Numerical abstract domain

1 引言

指针作为程序语言的重要数据类型,在使用时具有较高的灵活性,这种灵活性为程序的编写带来了极大的便利,但不当的指针使用也成为引发非法内存访问的源泉之一。图1中的程序Prog 1试图通过指针 p 操作数组 a,却在语句3处因指针使用不当导致未知内存的非法写入,带来不可预期的结果。而程序中大多数潜在的错误,如内存泄露、空指针引用、缓冲区越界等都与指针使用相关,因此对指针进行有效分析并发现指针程序中存在的问题是提高程序可信性的重要手段[5]。

```
void main(void){
  int a[3], * p;
  p=a;
  *(p+5)=2; //语句3
}
```

图1 简单示例程序Prog 1

现有的指针分析技术基本上都只维护了指针的指向关系,而没有维护指针变量的数值偏移关系。如经典的Steensgaard算法[2]和Anderson算法[3]在分析图1中程序Prog 1时,只知道指针 p 是指向数组 a 的,却不能发现语句3使用指针偏移算术后导致未知内存的非法写入错误。而传统的数值程序分析只针对数值型变量,未考虑指针信息,因而无法直接对含指针的程序进行分析。数值抽象解释[4]作为一种重要的程序分析技术可以对程序的数值性质进行有效的分析。基于抽象解释的值范围分析技术在检查除零错、数组越界、整数溢出等数值性质相关的运行时错误上取得了显著的成效,同时又能通过配置不同的数值抽象域在分析效率和精度之间进行合理权衡[5]。

本文针对带指针算术的程序中几类常见的运行时错误,如数组越界、缓冲区溢出、除零错、空指针引用等,将指向关系分析和数值性质分析相结合,提出了一种基于抽象解释的指针程序分析方法。本文首先提出一种新的指针内存模型,对程序中的指针进行建模,并基于该模型设计并实现一个刻画指针指向关系和指针偏移量的抽象域,最后在抽象解释框架下,结合指向抽象域和数值抽象域,设计并实现了一个面向指针C程序的静态分析工具原型PAA。实验结果表明,本文设计的指针模型具有较好的分析精度和效率,能够有效地分析

到稿日期:2014-06-10 返修日期:2014-09-15 本文受国家自然科学基金(61202120,91118007),教育部博士点基金(20124307120034)资助。
尹帮虎(1989-),男,硕士生,主要研究方向为抽象解释,E-mail:yinbanghu@163.com;陈立前(1982-),男,博士,助理研究员,主要研究方向为程序分析与验证、抽象解释;王 戟(1969-),男,博士,教授,博士生导师,主要研究方向为高可信软件技术。

指针程序的性质。

本文第2节简要回顾抽象解释的基本理论;第3节介绍了指针分析技术和现有内存模型等相关工作;第4节给出了本文提出的指针内存模型;第5节基于该模型给出了指向抽象域的实现;第6节讨论实验中工具原型的实现,并展开了一系列实验;第7节对本文进行总结,并展望下一步工作。

2 预备知识

抽象解释理论是由 P. Cousot 和 R. Cousot[4] 于 1977 年提出的程序静态分析时构造和逼近程序不动点语义的理论,该理论为自动利用静态分析方法推导程序运行时性质提供了一个通用框架。程序的抽象解释是指使用另一个抽象对象域上的计算抽象逼近程序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的部分信息。其中,抽象对象域和具体对象域的关系可以用一个 Galois 连接来刻画。

定义 1(伽罗瓦(Galois)连接) 设$\langle D,\sqsubseteq\rangle$和$\langle D^{\#},\sqsubseteq^{\#}\rangle$是两个偏序集合,$\alpha:D\to D^{\#}$和$\gamma:D^{\#}\to D$是两个映射,序偶$\langle\alpha,\gamma\rangle$称为$D$与$D^{\#}$之间的 Galois 连接当且仅当:

$\forall x\in D,y\in D^{\#},\alpha(x)\sqsubseteq y$当且仅当$x\leqslant\gamma(y)$

并记作

$$(D,\sqsubseteq)\underset{\gamma}{\overset{\alpha}{\rightleftarrows}}(D^{\#},\sqsubseteq^{\#})$$

其中,$(D,\sqsubseteq)$称为具体域,$(D^{\#},\sqsubseteq^{\#})$称为抽象域,$\alpha$称为抽象化函数,$\gamma$称为具体化函数。抽象域包括域表示和域操作两方面,是抽象解释理论框架下的核心要素,该框架下的所有计算都是在抽象域上进行的。

抽象解释本质上是在计算效率和计算精度之间取得均衡、以损失计算精度求得计算可行性、再通过迭代计算增强计算精度的一种抽象逼近方法。抽象解释的理论的一个最常见的应用就是数值程序的值范围分析。基于抽象解释的值范围分析试图发现程序中数值变量在每个程序点的近似取值范围。近年来,基于抽象解释的静态分析工具(如 Astree[6]、Fluctuat[7]、CGS[8]等)在工业界大规模软件尤其是航空航天控制软件的数值性质的分析与验证中取得了成功应用。

3 相关工作

3.1 指针分析技术

指针分析技术是程序分析与验证技术的重要组成部分,其主要目的是静态地获取程序运行时刻的指针指向信息,用于分析程序变量之间的指向关系。指针分析技术已经在检测内存泄露、非法指针引用、缓冲区溢出、字符串违规操作、数据竞争、死锁、编译代码优化以及促成指令级并行等领域得到应用。

指针分析技术根据流敏感性分为流不敏感和流敏感方法。流不敏感分析主要包括 Steensgaard[2] 提出的基于合并(Equivalence-based)的指针分析算法和 Anderson[3] 提出的基于包含(Inclusion-based)的指针分析算法。流敏感的分析主要包括基于迭代数据流的分析[9]和基于静态单赋值(Static Single Assignment,SSA)[10]的分析。其根据上下文敏感性分为上下文敏感分析和上下文不敏感分析,上下文敏感分析的方法又分为基于克隆的分析方法[11]和基于摘要的分析方法[10]。本文采用基于迭代数据流分析(流敏感)和基于克隆(上下文敏感)的分析技术相结合的静态分析方法。此外根据域敏感性、路径敏感性还可以进一步细化。

3.2 现有内存模型

对指针程序进行分析,首先需要解决的是变量在计算机中的存储问题,即刻画变量的内存模型。最初人们提出的内存刻画模型是"名字-值"模型[12],它是最基本的也是目前数值抽象域中应用最广泛的模型,其优点是非常简单直观,但缺陷是对于指针、数组、结构体等复杂数据结构不能直接进行有效刻画。如图1中程序 Prog 1 对应的程序语义 p 是一个指向 a 的指针,它的具体值是变量 a 的地址,而在"名字-值"模型中,并没有地址的概念,从而无法有效刻画 p 的信息。

为了刻画数组、指针、结构体等复杂数据结构,Hampapuram等人[13]使用了一种基于 region 的内存模型,Xu 等人[14]在此基础上引入了 region 层级结构。目前广泛使用的是 CGS[8]和 Frama-c[15]采用的字节级的"基址-偏移-长度"的内存模型。相比于"名字-值"模型,该模型在数组、指针、结构体等数据结构的刻画能力上有了显著的提高。

4 指针内存模型

指针记录了程序执行过程中内存中的某个地址,同时在指针变量所指向的地址上存储了某种类型的数据。从源代码层面来看,指针变量可能被赋值为某个绝对地址、某个简单类型变量的地址、某个数组元素的地址等。在程序分析过程中,分析算法需要为每个指针变量维护其指向地址相关的信息。本文假设待分析程序中没有动态内存分配,即没有 malloc() 语句。

对于程序中的每一个指针变量 p,在某个程序点处,本文采用如下二元组来刻画其信息:

$\langle PBaseAddr,Offset\rangle$

其中,$PBaseAddr$ 是一个地址表达式,表示指针所指向数据类型的基地址(或称起始地址),可以是绝对地址、取地址表达式、数组变量基址或复合数据类型成员域的访问路径表达式;$Offset$ 是一个整数值,表示指针变量 p 所指向地址与 $PBaseAddr$ 间的偏移量(按指针数据类型大小而非字节大小来计算)。

例1 对于图2所示的代码片段:

```
struct ss{
  float a;
  int p[3];
  int b;
  }s1;
void main(void) {
  int *q=s1.p;
  int *r=q+2;
  float *s=&s1.a;
  int *t=(int *) (0x10ffffff);
}
```

图2 指针内存模型示例程序 Prog 2

指针变量 q 的内存模型描述为$\langle s1.p,0\rangle$,指针变量 r 的内存模型描述为$\langle s1.p,2\rangle$,指针变量 s 的内存模型描述为$\langle\&s1.a,0\rangle$,指针变量 t 的内存模型描述为$\langle 0x10ffffff,0\rangle$。

不难看出,指针变量 p 的内存模型主要涉及其指向信息和相对基地址的偏移信息。指针变量的指向信息一般可以通过指向分析得到,而相对基地址的偏移量是整数值,可以通过数值抽象解释技术分析得到。下面将介绍如何在抽象解释框

架下，采用流敏感的算法来自动分析程序中每个指针变量在每个程序点(如代码行的起始位置)处的内存模型信息。

在流敏感的分析算法下，某个程序点处某指针变量 p 的指向信息可能存在多种情况。因此，指针变量多种情况下的内存模型将构成一个集合。本文将采用一个指向抽象域 PointstoDomain 来自动分析每个指针变量 p 的可能基地址集合；为每个指针变量 p 引入一个整型变量 $offset\_p$，并将 $offset\_p$ 加入到数值抽象解释抽象环境的数值变量集合中。

上述抽象把指向抽象域和数值抽象域模块化分离，从而可以容易集成已有数值抽象域的成果。两个抽象域之间通过指针变量的名字作为纽带进行通信。

例 2　对于图 3 所示的代码片段：

```
int A[10],B[20];
void main(void) {
  int *p;
  int i=0,k;
  if(k>0) {
    p=A+1;
    i=i+1;
  }
  else {
    p=B+5;
    i=i+5;
  }
}
```

图 3　指针内存模型示例程序 Prog 3

在 main()函数结束时，指针变量 p 的内存模型存在两种情况：$\langle A,1\rangle$或$\langle B,5\rangle$。文中的指向抽象域 PointstoDomain 在 main()函数结束处分析得到指针变量 p 的基地址集合为$\{A,B\}$。另一方面，本文算法将为指针变量 p 引入一个整型变量 $offset\_p$，并将其加入数值变量所构成的抽象环境中，从而数值抽象解释的抽象环境中将包括程序变量 i 和 $offset\_p$。在数值抽象解释框架下，采用多面体、八边形等数值抽象域，并在 main()函数结束处分析得到：$1\leqslant offset\_p\leqslant 5\wedge offset\_p==i$。

本文直接调用已有数值抽象域来对数值程序变量及指针偏移变量开展分析。数值抽象域的设计与实现的相关细节请参考文献[4,16]。本文接下来将重点介绍指向抽象域的设计与实现。

5　指向抽象域的设计与实现

指向抽象域主要维护指针变量可能指向的基地址的集合。该抽象域的域将维护一个映射 $\mathscr{B}$，把每一个指针变量 p 映射为某基地址表达式集合 $\mathscr{B}_p$。本文中，基地址表达式可以是一个绝对地址、变量求地址表达式、数组起始地址等。如果指针没有指向任何地址(即空指针)，则 $\mathscr{B}_p=\emptyset$，即作为指向抽象域中的 *bottom* 元素 $\perp$。指针变量 p 映射为某基地址表达式集合 $\mathscr{B}_p$ 构成的一个完全格$(P(\mathscr{B}_p),\subseteq,\cap,\cup,\perp,\top)$，其中$\top$表示程序中所有基地址表达式构成的集合。

下面给出指向抽象域上的常用域操作。指向抽象域中的包含测试操作(Inclusion test)、交操作(Meet)、接合操作(Join)分别对应集合上的子集包含操作$\subseteq$、集合交操作$\cap$、集合并操作$\cup$。另外，当源程序不含动态内存分配(malloc)时，程序中所有基地址表达式的个数是有穷的，所以基地址表达式集合 $\mathscr{B}_p$ 构成的一个完全格是有穷高度的。因此，即使存在循环和递归函数，也无需在抽象解释框架下设计指向抽象域上的加宽操作(Widening)，并依然能保证 Kleene 不动点迭代的终止性。

下面重点介绍指向抽象域上迁移函数的设计。

5.1　赋值迁移函数(Assignment Transfer Function)

本文考虑如下几类对指针变量 p 的赋值语句。

- $p:=q+iexpr$;//对应指针算术操作，其中 q 是一个指针变量，$iexpr$ 是一个整型表达式
- $p:=\&x$;//对应取地址操作，其中 x 是一个整型、浮点型、结构体或联合体变量
- $p:=A+iexpr$;//对应数组操作，其中 A 是一个数组变量，$iexpr$ 是一个整型表达式
- $p:=\&A[iexpr]$;//对应数组操作，其中 A 是一个数组变量，$iexpr$ 是一个整型表达式
- $p:=(void*)c$;//对应绝对地址操作，其中 c 是一个非负整数常数，表示绝对地址

其他对指针变量 p 的复杂赋值语句，都可以采用把程序转换为静态单赋值(Static Single Assignment，SSA)[10]形式的思想，通过引入临时变量把复杂赋值语句转换为上述形式的赋值语句。本文暂不考虑多重指针，因为多重指针涉及指针所指向内容的分析。上述指针赋值语句的赋值迁移函数定义如下：

$$[\![p:==q+iexpr]\!]^{P^\#}(\mathscr{B}^\#,O^\#)\stackrel{\triangle}{=}(\mathscr{B}_p^\#\mapsto\mathscr{B}_q^\#)\wedge[\![O_p^\#:=O_q^\#+iexper]\!]^{N^\#}(O^\#)$$

$$[\![p:=\&x]\!]^{P^\#}(\mathscr{B}^\#,O^\#)\stackrel{\triangle}{=}(\mathscr{B}_p^\#\mapsto\{\&x\})\wedge[\![O_p^\#:=0]\!]^{N^\#}(O^\#)$$

$$[\![p:=A+iexpr]\!]^{P^\#}(\mathscr{B}^\#,O^\#)\stackrel{\triangle}{=}(\mathscr{B}_p^\#\mapsto\{A\})\wedge[\![O_p^\#:=iexpr]\!]^{N^\#}(O^\#)$$

$$[\![p:=\&A[iexpr]]\!]^{P^\#}(\mathscr{B}^\#,O^\#)\stackrel{\triangle}{=}(\mathscr{B}_p^\#\mapsto\{A\})\wedge[\![O_p^\#:=iexpr]\!]^{N^\#}(O^\#)$$

$$[\![p:=(void)c]\!]^{P^\#}(\mathscr{B}^\#,O^\#)\stackrel{\triangle}{=}(\mathscr{B}_p^\#\mapsto\{c\})\wedge[\![O_p^\#:=0]\!]^{N^\#}(O^\#)$$

其中，函数的输入$(\mathscr{B}^\#,O^\#)$表示在执行赋值语句之前的抽象环境(包括基地址抽象域中抽象环境 $\mathscr{B}^\#$、数值抽象域中偏移抽象环境 $O^\#$)；$[\![assignment]\!]^{N^\#}$ 表示数值抽象域上的赋值迁移函数。具体而言，在针对指针变量 p 的赋值迁移函数中，本文除了更新 p 的基地址信息外，还把对应偏移辅助变量 O_p 的运算映射到数值抽象域上相应的赋值操作。

例 3　对于图 4 所示的代码片段：

```
int A[10];
int B[20];
void
main(void) {
  int *p,*q,*r,*s,*t;
  int a,i=3;
  p=&a;
  q=A+(i-2);
  r=q+2*i;
  s=&B[i*i];
  t=(int *)(0x0fffffff);
}
```

图 4　赋值迁移函数示例程序 Prog 4

在 main()函数结束处，一方面指向抽象域 PointstoDomain 将分析得到指针变量 p,q,r,s,t 的基址集合，分别为：$\mathcal{B}_p=\{\&a\}$，$\mathcal{B}_q=\{A\}$，$\mathcal{B}_r=\{A\}$，$\mathcal{B}_s=\{B\}$，$\mathcal{B}_t=\{0x0fffffff\}$；另一方面，数值抽象域还能得到偏移抽象环境，采用区间抽象域表示，即为：$O_p=0$，$O_q=1$，$O_r=7$，$O_s=9$，$O_t=0$，若采用多面体、八边形等精度更高的数值抽象域，还能进一步得到：$O_q=i-2$、$O_r=O_q+2*i$ 等约束关系。

5.2 测试迁移函数

测试迁移函数(Test Transfer Function)$[\![expr_1 \bowtie expr_2]\!]^{P^\#}$(其中 $\bowtie \in \{\neq,=,<,\leqslant,>,\geqslant\}$)的目的是过滤当前不满足布尔表达式 $expr_1 \bowtie expr_2$ 的环境。本文假设只有当 $expr_1$ 和 $expr_2$ 有共同基址时，测试迁移函数才在抽象语义上进行判定，否则直接返回 bottom。任何形式的测试条件都可以最终抽象成一个或多个形如 $p\leqslant expr$ 或 $p\geqslant expr$ 的约束[5,16]，$expr$ 是一个地址表达式，可以是绝对地址、取地址表达式、数组变量基址或复合数据类型成员域的访问路径表达式。正如 5.1 节所述，依据静态单赋值(SSA)转换思想，通过引入临时变量，$expr$ 总能表示成以下某种形式。

- $expr \triangleq q+iexpr$；//对应指针算术操作，其中 q 是一个指针变量，$iexpr$ 是一个整型表达式
- $expr \triangleq \&x$；//对应取地址操作，其中 x 是一个整型、浮点型、结构体或联合体变量
- $expr \triangleq A+iexpr$；//对应数组操作，其中 A 是一个数组变量，$iexpr$ 是一个整型表达式
- $expr \triangleq \&A[iexpr]$；//对应数组操作，其中 A 是一个数组变量，$iexpr$ 是一个整型表达式
- $expr \triangleq (void\ *)\ c$；//对应绝对地址操作，其中 c 是一个非负整数常数，表示绝对地址

因而本文针对形如 $p\leqslant expr$ 的约束，定义了测试迁移函数如下所示：

$$[\![p\leqslant q+iexpr]\!]^{P^\#}(\mathcal{B}^\#,O^\#)\triangleq(\mathcal{B}_p^\#,\mathcal{B}_q^\#\mapsto\mathcal{B}_q^\#\cap\mathcal{B}_p^\#)\wedge[\![O_p^\#\leqslant O_q^\#+iexper]\!]^{N^\#}(O^\#)$$

$$[\![p\leqslant \&x]\!]^{P^\#}(\mathcal{B}^\#,O^\#)\triangleq(\mathcal{B}_p^\#\mapsto\mathcal{B}_p^\#\cap\{x\})\wedge[\![O_p^\#\leqslant 0]\!]^{N^\#}(O^\#)$$

$$[\![p\leqslant A+iexpr]\!]^{P^\#}(\mathcal{B}^\#,O^\#)\triangleq(\mathcal{B}_p^\#\mapsto\mathcal{B}_p^\#\{A\})\wedge[\![O_p^\#\leqslant iexpr]\!]^{N^\#}(O^\#)$$

$$[\![p\leqslant \&A[iexpr]]\!]^{P^\#}(\mathcal{B}^\#,O^\#)\triangleq(\mathcal{B}_p^\#\mapsto\mathcal{B}_p^\#\{A\})\wedge[\![O_p^\#\leqslant iexpr]\!]^{N^\#}(O^\#)$$

$$[\![p\leqslant (void)c]\!]^{P^\#}(\mathcal{B}^\#,O^\#)\triangleq(\mathcal{B}_p^\#\mapsto\mathcal{B}_p^\#\{c\})\wedge[\![O_p^\#\leqslant 0]\!]^{N^\#}(O^\#)$$

其中，函数的输入$(\mathcal{B}^\#,O^\#)$表示在执行赋值语句之前的抽象环境；$[\![conditional\ testing]\!]^{N^\#}$ 表示数值抽象域上的测试迁移函数。对于形如 $p\geqslant expr$ 的约束，可以定义类似的测试迁移函数。

例 4　对于图 5 所示的代码片段：

```
int A[10];
int B[20];
int *p, *q,a,k;
void init(void){
  if(k > 0)
  {
    p=&a;
    q=&a;
  }
  else
  {
    p=A+9;
    q=B+15;
}
}
void main(void) {
  int i=2,j=0;
  init();
if (p≤q+i+5)
    j=1;
  if(p≤&a)
    j=2;
  if(p≤A+2*i)
    j=3;
  if(p≤&A[2*i])
    j=4;
  if(p≤(int*)0x0fffffff)
    j=5;
}
```

图 5　测试迁移函数示例程序 Prog 5

在 main()函数分析结束时，语句 init()后对应的指向抽象域环境和偏移抽象环境将分别为：$\mathcal{B}_p=\{\&a,A\}$，$\mathcal{B}_q=\{\&a,B\}$，$O_p=[0,9]$，$O_q=[0,15]$；语句 $j=1$ 后的指向抽象域环境和偏移抽象环境将分别为：$\mathcal{B}_p=\{\&a\}$，$\mathcal{B}_q=\{\&a\}$，$O_p=[0,7]$，$O_q=[2,15]$；语句 $j=2$ 后的指向抽象域环境和偏移抽象环境将分别为：$\mathcal{B}_p=\{\&a\}$，$\mathcal{B}_q=\{\&a,B\}$，$O_p=0$，$O_q=[0,15]$；语句 $j=3$ 和 $j=4$ 后的指向抽象域环境和偏移抽象环境将分别为：$\mathcal{B}_p=\{A\}$，$\mathcal{B}_q=\{\&a,B\}$，$O_p=[0,4]$，$O_q=[0,15]$；由于 if(p≤(int *)0x0fffffff)语句判定后 $\mathcal{B}_p=\emptyset$，即不存在满足条件的指针指向的地址，因此 $j=5$ 不会被执行，对应的指向抽象域环境和偏移抽象环境都将为 bottom 元素 $\perp$。

6 实验

基于开源编译器 CIL[17]、数值抽象域库 Apron[5]和本文实现的指向抽象域 PointstoDomain，开发了一个面向 C 语言的指针程序分析工具 PAA。CIL 是一个开源的编译前端工具，能够根据输入的 C 程序生成相应控制流图。Apron 为各种抽象域提供了一组必需的公共接口，基于 Apron 实现的众多数值抽象域包括区间、八边形、多面体等，用户可根据自己的精度要求进行选择[18]。

PAA 是基于经典的抽象解释框架设计并实现的[19]，分析流程大致包括以下 3 个阶段：首先通过 CIL 对源文件进行词法和语法解析，得到程序的控制流图；然后基于控制流图使用数值抽象域和指向抽象域 PointstoDomain 提供的数据类型建立语义方程；最后调用不动点迭代器对语义方程进行迭代，生成源程序对应的指向不变式和数值不变式。

本文采用的实验平台是 Ubuntu 13.04 Linux 操作系统，4GB 物理内存，Intel(R) Core(TM) i5-3330 3.0GHz 四核 CPU 处理器。为了展示 PAA 分析带指针算术程序的不变式结果和报警信息，图 6 给出了一个示例程序及对应标注的不变式。该例子是对 Frama-c Value Analysis 使用手册[15]中的程序经过简单修改得到的。

源程序	PAA 分析结果不变式标注
	values={a[0..5]=0}
	values={a[0..5]=0,b[0..7]=0,offset(p)=0};
void main(void){	points={Ptr(p)={}}
int a[6]={0},i;	values={a[0..5]=0,b[0..7]=0,i−1≥0,offset(p)=0};points={Ptr(p)={}}
int * p,b[8]={0};	values={a[0..5]=0,b[0..7]=0,i−≥0,offset(p)=2};points={Ptr(p)={a}}
if(i > 0)	values={a[0..5]=0,b[0..7]=0,−i≥0,offset(p)=0};points={Ptr(p)={}}
p=&a[2];	values={a[0..5]=0,b[0..7]=0,−i≥0,offset(p)=6};points={Ptr(p)={b}}
else	values={a[0..5]=0,b[0..7]=0,i=3,offset(p)=[2,6]};points={Ptr(p)={a,b}
p=&b[6];	values={a[0..5]=0,b[0..7]=0,i=3,offset(p)=[5,6]};points={Ptr(p)={a}}
i=3;	values={a[0,1,5]=0,a[2..4]=[0,3],b[0..7]=0,i=3,offset(p)=[2,4]};points={Ptr(p)={a}}
while(p≤&a[4])	values={a[0,1,5]=0,a[2..4]=[0,3],b[0..7]=0,i=3,offset(p)=[3,5]};points={Ptr(p)={a}}
{ * p=i;	values={a[0,1,5]=0,a[2..4]=[0,3],b[0..7]=0,i=3,offset(p)=[5,6]};points={Ptr(p)={a,b}}
p++;	values={a[0,1,5]=0,a[2..4]=[0,3],b[0..5]=0,b[6,7]=5,i=3,offset(p)=[5,6]};points={Ptr(p)={a,b}} &.& Array Out of Bound for b[a[3]+6]
}	
b[a[3]+6]=5;	
}	

图 6 示例程序及其不变式标注

对于语句 $b[a[3]+6]=5$，执行之前 $a[3]+6$ 的取值范围是[6,9]，而数组的长度只有 8，所以会产生数组越界，只需给数组 b 对应下标为[6,7]的变量赋值为 5 即可。

表 1 展示了区间域和八边形域分别与指向域结合时 PAA 分析部分指针程序的结果，其中加宽算子延迟参数(加宽算子应用之前的迭代次数)都设置为 4。Prog 1 和 Prog 5 分别对应图 1 和图 5 中的程序；程序 ff1k.c 来自 SNU 实时测试集[20]，含有大量的结构体指针和结构体数组；其余 5 个程序来自 WCET Challenge 基准测试程序[21]，其中 admpcm.c 包含大量的数组、结构体、指针以及它们的组合等复杂数据结构。

表 1 部分程序分析结果与真实结果精度对比

程序名	代码行数	变量数	以毫米为单位的计算时间(警报数目)		精度比较
			区间域+指向域	八边形域+指向域	
Prog1	5	2	<0.001(1)	<0.001(1)	=
Prog5	29	7	6.2(0)	35(0)	<
Admpcm	843	146	620(16)	>1h(−)	<
Compress	521	50	148(15)	11884(12)	<
Prime	45	6	24(0)	352(0)	<
St	182	25	36(1)	7248(1)	<
Whet	230	26	60(0)	7332(0)	<
ff1k	155	21	72(0)	7644(0)	<

表 1 给出了 PAA 分析指针程序的时间和产生的警报数量(表中只统计了空指针引用、除零错和数组越界的数量)，“精度比较”栏用以比较基于指向抽象域的分别与两种不同数值抽象域相结合分析指针程序得到的不变式的精度。其中“<”表示基于八边形抽象域比区间抽象域得到了更强的值范围信息，“=”表示两者得到的值范围信息一样。所有测试程序中除了 Prog1 中存在一个真实的错误外，其余程序都不存在错误。从表 1 中可以看出，(1)原来单纯的数值分析和指针分析方法都不能分析得到有效的带指针的程序结果，而 PAA 都可以进行分析，而且能得到可靠的程序不变式和报警结果；(2)PAA 分析数值抽象域时选择八边形域比选择区间域分析得到的不变式精度更高，但时间开销也更大。

PAA 结合了数值域和指向域的分析方法，具有较好的可扩展性，它的时间和空间复杂性与相应的数值抽象域的复杂性是一致的，例如区间域和指向域结合后的时间和空间复杂度与区间域的复杂度相同，即都为 $O(n)$；八边形结合指向域的复杂度和八边形域的相同，即时间复杂度都为 $O(n^3)$、空间复杂度都为 $O(n^2)$。用户可以根据需求选择不同的数值抽象域获得时间和效率的有效折衷。

结束语 本文首先设计了一个基址表达式与指针偏移量相结合的内存模型，以克服现有内存模型存在的不足；然后以抽象解释框架为基础，基于本文设计的内存模型，实现了支持指针算术的指向抽象域，并与现有数值抽象域相结合来分析指针变量基于基地址的偏移信息，实现了分析精度和时空效率的合理折衷。实验结果表明：本文开发的基于数值域和指针域的抽象解释工具 PAA 既能刻画变量的数值性质，又能刻画变量的指向关系。PAA 产生的警报是不存在漏报的(即包含了所有可能的错误)，所以其分析结果对于查找带指针算术程序中的运行时错误具有一定的指导意义。下一步工作将着重实现动态内存分配的有效分析，提高 PAA 的分析能力，降低分析结果的误报率，并与已有的相关分析工具进行实验对比。

参 考 文 献

[1] Dong W, Chen L Q. Recent advances on trusted computing in China[J]. Chinese Science Bulletin, 2012, 57(35): 4529-4532

[2] Steensgaard B. Points-to analysis in almost linear time [C]// Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1996: 32-41

[3] Andersen L O. Program analysis and specialization for the C programming language[D]. University of Cophenhagen, 1994

[4] Cousot P, Cousot R. Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]//Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM, 1977: 238-252

[5] 陈立前. 基于区间线性抽象域的可靠浮点及非凸静态分析[D]. 长沙：国防科学技术大学，2010
Chen Li-qian. Sound floating-point and non-convex static analysis using interval linear abstract domains[D]. Changsha: School of National University of Defense Technology, 2010

[6] Cousot P, Cousot R, Feret J, et al. The ASTRéE analyzer[M]// Programming Languages and Systems. Springer Berlin Heidelberg, 2005: 21-30

[7] Delmas D, Goubault E, Putot S, et al. Towards an industrial use of FLUCTUAT on safety-critical avionics software[M]// Formal Methods for Industrial Critical Systems. Springer Berlin Heidelberg, 2009: 53-69

[8] Venet A, Brat G. Precise and efficient static array bound checking for large embedded C programs[J]. ACM SIGPLAN Notices, 2004, 39(6): 231-242

[9] Hasti R, Horwitz S. Using static single assignment form to im-

prove flow-insensitive pointer analysis[J]. ACM SIGPLAN Notices. ACM, 1998, 33(5): 97-105

[10] Yu H, Xue J, Huo W, et al. Level by level: making flow-and context-sensitive pointer analysis scalable for millions of lines of code[C]// Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization. ACM, 2010: 218-229

[11] Whaley J, Lam M S. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams[J]. ACM SIGPLAN Notices. ACM, 2004, 39(6): 131-144

[12] King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394

[13] Hampapuram H, Yang Y, Das M. Symbolic path simulation in path-sensitive dataflow analysis[J]. ACM SIGSOFT Software Engineering Notes. ACM, 2005, 31(1): 52-58

[14] Xu Z, Kremenek T, Zhang J. A memory model for static analysis of C programs[M]// Leveraging Applications of Formal Methods, Verification, and Validation. Springer Berlin Heidelberg, 2010: 535-548

[15] Canet G, Cuoq P, Monate B. A Value Analysis for C Programs [C]// 2013 IEEE 13th International Working Conference on Source CodeAnalysis and Manipulation (SCAM). IEEE, 2009: 123-124

[16] Miné A. Weakly relational numerical abstract domains[D]. Paris: École Normale Supérreure, 2004

[17] Necula G C, McPeak S, Rahul S P, et al. CIL: Intermediate language and tools for analysis and transformation of C programs [C]// Compiler Construction. Springer Berlin Heidelberg, 2002: 213-228

[18] Chen L, Li R, Wu X, et al. Static analysis of list-manipulating programs via bit-vectors and numerical abstractions[C]// Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013: 1204-1210

[19] 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
Li Meng-jun, Li Zhou-jun, Chen Huo-wang. Program verification techniques based on the abstract interpretation theory[J]. Journal of Software, 2008, 19(1): 17-26

[20] SNUReal-Time Benchmark Suite[OL]. http://archi.snu.ac.kr/realtime/benchmark

[21] von Hanxleden R, Holsti N, Lisper B, et al. WCET tool challenge 2011: report[OL]. http://hdl.handle.net/22909/10354

（上接第 31 页）

[3] Yokoo M, Durfee E H, Ishida T, et al. Distributed constraint satisfaction for formalizing distributed problem solving[C]// Proceedings of the 12th IEEE International Conference on Distributed Computing System. Yokohama, Japan: IEEE Computer Society Press, 1992: 614-621

[4] Bessiere C, Regin J C, Yap R H C, et al. An optimal coarse-grained Arc Consistency algorithm[J]. Artificial Intelligence, 2005, 165(2): 165-185

[5] Mackworth A K. Consistency in networks of relations[J]. Artificial Intelligence, 1977, 8(1): 99-118

[6] Bessiere C, Regin J C. Refining the basic constraint propagation algorithm[C]// Proceedings of the 17th International Joint Conference on Artificial Intelligence. Seattle, USA: Morgan Kaufmann, 2001: 309-315

[7] Deburyne R, Bessiere C. Some practical filtering techniques for the constraint satisfaction problem[C]// Proceedings of the 15th International Joint Conference on Artificial Intelligence. Nagoya, Japan: Morgan Kaufmann, 1997: 412-417

[8] Bartak R, Erben R. A new algorithm for singleton arc consistency[C]// Proceedings of FLAIRS Conference. Florida, USA: AAAI Press, 2004: 257-262

[9] Bessiere C, Debruyne R. Optimal and suboptimal singleton arc consistency algorithms[C]// Proceedings of the 19th International Joint Conference on Artificial Intelligence. Edinburgh, UK: Professional Book Center, 2005: 54-59

[10] Lecoutre C, Cardon S. A greedy approach to establish singleton arc consistency[C]// Proceedings of the 19th International Joint Conference on Artificial Intelligence. Edinburgh, UK: Professional Book Center, 2005: 199-204

[11] Bessiere C, Bessiere R. Theoretical analysis of singleton arc consistency and its extensions[J]. Artificial Intelligence, 2008: 172 (1): 29-41

[12] 孙吉贵，朱兴军，张永刚，等. 最先失败原则的约束传播算法[J]. 小型微型计算机系统，2008，29(4)：678-681
Sun J G, Zhu X J, Zhang Y G. Constraint Propagation on Fail First Principle[J]. Mini-Micro Systems, 2008, 29(4): 678-681

[13] 孙吉贵，朱兴军，张永刚，等. 一种基于预处理技术的约束满足问题求解算法[J]. 计算机学报，2008，31(6)：919-926
Sun J G, Zhu X J, Zhang Y G. An Approach of Solving Constraint Satisfaction Problem Based on Preprocessing[J]. Chinese Journal of Computers, 2008, 31(6): 919-926

[14] 朱兴军，孙吉贵，张永刚，等. 一种新的基于完全独立相容性的预处理技术[J]. 自动化学报，2009，35(1)：71-76
Zhu X J, Sun J G, Zhang Y G. A new preprocessing technique based on entirety singleton consistency[J]. Acta Automatica Sinica, 2009, 35(1): 71-76

[15] 朱兴军，张永刚，李莹，等. 多值传播的相容性技术[J]. 自动化学报，2009，35(10)：1296-1301
Zhu X J, Zhang Y G, Li Y, et al. Consistency Technique of Multi-Value Propagation[J]. Acta Automatica Sinica, 2009, 35 (10): 1296-1301

[16] 刘春晖，朱兴军，孙吉贵. 一种改进的双向 singleton 弧相容算法[J]. 吉林大学学报(工学版)，2008，3(28)：666-670
Liu C H, Zhu X J, Sun J G. Improved bidirectional singleton arc consistency algorithm[J]. Journal of Jilin University Engineering and Technology Edition, 2008, 3(28): 666-670

[17] Tsang E P K. Foundations of Constraint Satisfaction[M]. Academic Press, London and San Diego, 1993

[18] Gent I P, Macintyre E, Prosser P, et al. Random constraint satisfaction: Flaws and structure[J]. Journal of Constraints, 2001, 6 (4): 345-372