面向软件可信性的可信指针分析技术综述 *

姚宇峰
(常熟理工学院 计算机科学与工程学院, 江苏 常熟 215500)

摘 要: 对可信指针分析技术的定义和描述、指针分析对软件可信性的保障、可信指针分析属性以及该领域主要研究成果等方面进行了综述。通过对现有可信指针分析技术的分析和比较,详细讨论了面向软件可信性的可信指针分析的关键技术;此外,重点介绍了流敏感指针分析及上下文敏感指针分析的方法和理论;最后对进一步研究工作的方向进行了展望。

关键词: 软件可信; 程序分析技术; 可信指针分析; 流敏感分析; 上下文敏感分析
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2012)02-0427-05
doi:10.3969/j.issn.1001-3695.2012.02.005

Trustworthy pointer analysis for software reliability

YAO Yu-feng
(*School of Computer Science & Engineering, Changshu Institute of Technology, Changshu Jiangsu* 215500, *China*)

Abstract: This paper reviewed the definition and description of trustworthy pointer analysis technology, the protection of pointer analysis on software reliability, the analysis attributes of trustworthy pointer and the main research on this field. Through the analysis and comparison of the existing trustworthy pointer analysis technologies, the paper specifically discussed the key technology of trustworthy pointer analysis for software reliability. Additionally, it focused on the methods and theories of flow sensitive pointer analysis and context sensitive pointer analysis. At last, it discussed the outlook of the direction of further research.

Key words: software trustworthiness; program analysis technology; trustworthy pointer analysis; flow sensitive pointer analysis; context sensitive pointer analysis

0 引言

程序分析技术伴随着程序语言的发展而发展,并不断地为程序语言提供必要的支持。首先,程序分析的结果可以用来对程序进行优化,如对于机器无关的标量优化[1]、常数传播[2]、冗余代码删除[3,4]、失效代码删除[5]等,从而提高编译器生成的可执行代码的运行速度;其次,程序分析可以指导程序的自动并行化[6,7],如程序被分割成可以独立运行的模块,然后将这些模块分配到可以并行执行的计算节点。近些年,程序分析也逐渐向保障程序可信性方面入手,成为检测软硬件错误的重要手段,例如程序分析的结果可以用来检测缓冲区越界及字符串违规操作[8],也可以检测非法指针引用及内存泄露[5,9,10]等问题,对于多线程的程序,程序分析技术还可以检测数据竞争和死锁问题[11,12]。

由于程序分析技术,特别是静态分析技术能够在程序执行之前静态地估计所有执行行为,在软件质量和安全性领域具有重要的作用。软件的质量和安全性一直以来都是软件可信性的重要表现,随着大规模软件的不断发展,系统软件如操作系统、中间件、应用软件 Web service、桌面游戏软件等都或多或少地存在潜在的安全和可靠性威胁。传统的软件测试技术对这些潜在的程序错误不能进行彻底的解决,通常只能覆盖到已知测试集中的错误情况。但对许多测试集外的错误很难检测出来,给程序的可信性遗留了很多安全和质量上的隐患。其中一些小概率错误的检测更为困难,而往往这些小概率的错误也是对系统最大的威胁。这些没有被测试集覆盖的错误类型,就需要由静态程序分析来进行可信性检测。程序中绝大多数潜在的错误威胁,如 C 语言中内存泄露、数据竞争、死锁、空指针、缓冲区越界问题都与程序指针问题有关,所以指针分析的精确和高效与否直接决定程序检测的性能和准确度。在过去的二十年里,指针分析问题也已经成为程序分析技术研究的热点问题,而且至今也是非常活跃的研究方向。

1 可信指针分析技术研究现状

指针分析技术最早应用于程序优化及代码生成阶段。随着软件不断发展和日益增长的应用需求,软件漏洞也一直成为人们关注的重点,如何利用指针对程序分析的利器进行可信的程序分析及检测,成为国内外研究的热点。可信指针分析是利用指针分析的结果对程序的正确性、可靠性、安全性等诸多非功能属性维度进行可信保障的一项技术手段。

精确的指针分析可以优化编译器生成的代码、发掘更多的指令级并行的机会[13]。性能良好的程序错误检错器也同样依赖于指针分析的精度,精确的指针分析可以减少错误检测器的误报和漏报,提高错误定位的精度[9,14]。其他一些应用,如数据竞争、死锁同样需要精确高效的指针分析作为基础。近些年来指针分析也已经成为阻碍硬件行为综合发展

收稿日期: 2011-08-25; **修回日期**: 2011-09-30 **基金项目**: 高校青年教师科研启动基金资助项目
作者简介: 姚宇峰(1981-),男,江苏常熟人,讲师,硕士,主要研究方向为软件工程(yaoyufengls@163.com).

的瓶颈[1517]。硬件的综合行为是将 C/C ++ 语言设计的算法描述自动转换成寄存器传输级描述,从而代替硬件设计师手工工作以自动生成实现电路布局。可以简单地认为,指针分析越精确,能够综合生成的硬件电路规模就越大。目前已有的指针分析要么分析效率高但精度不高,如流不敏感、上下文不敏感指针分析;要么精度高但分析效率并不理想,如流敏感、上下文敏感指针分析。这两种瓶颈都给可信程序分析与检测带来了巨大的挑战。

指针分析在保障程序可信性方面扮演着重要的角色,优秀的漏洞检测工具需要精确的指针分析作为支持,如文献[18, 19]的研究表明,精确的指针分析对提高程序切片的精度非常有帮助。指针错误使用也是引起内存非法操作的一类错误的源泉,因此指针分析是程序漏洞检测的必要步骤。例如文献[20]使用指针分析来检测缓冲区越界及字符串违规使用问题;文献[21]利用指针分析检测非法指针引用;文献[10, 22]使用指针分析解决内存泄露问题;文献[11, 12, 23, 24]利用指针分析进行数据竞争和死锁的检测;文献[25, 26]探讨了检测不同程序错误对指针分析的要求。

总之,指针分析的精度越高,对后续的应用帮助越大,但是完全精确的指针分析是不可确定问题[27, 28]。因此,不能一味地追求指针分析的精度而忽略其效率,在精度与效率之间的取舍和权衡是指针分析设计必须考虑的问题。近些年来,随着软件规模的不断扩大,高效率高精度的以指针分析为基础的保障程序可信性的技术已经成为国内外关注且活跃的研究方向[2932]。

2　指针分析对软件可信性保障

指针分析在传统的编译和程序分析领域常常用做程序的优化和代码生成,而随着近些年软件规模、程序复杂性的不断发展,人们对软件的可靠、安全、实时等非功能属性的要求不断增加,指针分析正发展成为保障软件可信性的有力工具和重要研究方向。

指针分析可以用来检测程序中的内存泄露问题,如下面列出的代码段。指针 p 在 $L1$ 点指向新分配的一块内存,在程序的 $L2$ 点指针 q 指向了指针 p,并且在 $L3$ 点将 p 的值间接修改为指向另外新分配的一块内存。虽然在程序结束时对 p 新指向的内存进行了释放,但是在 $L1$ 点分配的内存却未能释放,从而导致了内存泄露。所以指针分析的结果能够指导程序的分析,从而将潜在威胁以警告的形式在代码编译阶段告诉用户,保证了程序的可信性。

```
int example(){
    int *p, **q;
L1:p = malloc(sizeof(int));
L2:q = &p;
L3:*q = malloc(sizeof(int));
    free(p)}
```

多线程中的数据竞争也可以受益于指针分析,如下面的代码段所示,程序创建了两个线程,并且两个线程都调用 run 方法。此时在方法 run 中对共享变量 share 的操作是进行了加锁的保护,以保证同一时间不会产生两个线程同时对一个共享变量进行写操作。但是由于指针 q 指向了共享变量,所以也同样需要对 $*q$ 进行加锁和解锁的保护,确保两个线程不会在同一时间被两个线程同时进行写操作,导致数据不一致的情况。而 run 方法中并没有将 $*q$ 表达式进行加锁保护,则会造成潜在的程序威胁。所以有效的指针分析能够检测出用户程序中潜在的危险漏洞,从而避免软件灾难的产生。

```
int share;
int main(){
pthread_create(id1,run);
pthread_create(id2,run);
}
void run(){
int *q = &share;
int money;
pthread_mutex_lock(&id);
share = money;
pthread_mutex_unlock(&id);
    *q = money;
}
```

指针分析在保证程序可信性方面尚处于起步和发展阶段,指针分析是程序分析的技术和核心。指针分析的精确与否、效率高低直接决定了程序分析性能以及能力可信保障的水平。指针分析由于其本身的复杂性,精确高效的指针分析研究还处于发展中,远没有达到保证软件能力可信性的要求。所以精确高效的指针分析设计及算法面临着众多的机遇和挑战。

3　可信指针分析技术

可信指针分析具有几个主要的精度衡量属性,同时也是可信指针分析技术研究的重要方向,如流敏感性[13,33,34]、上下文敏感性[13,3436]、域敏感性[37]、路径敏感性[25,38]等。而流敏感性和上下文敏感性也是近些年指针分析技术研究的重点。如表 1 所示,显示了顶级会议 PLDI、POPL、CGO 等近十余年来对指针分析的研究。可以看出,指针分析在近些年来一直成为研究学者关注的重点。指针分析作为一项基础性研究,也指导着其他相关领域的研究发展。

表 1　流不敏感与流敏感的指针分析研究的发展

		flow insensitive		flow sensitive	
		unification-based	inclusion-based	iteration-based	SSA-based
context insensitive		Steensgaard PLDI'96 1 MLOC Manuvir Das PLDI'00 2 MLOC	Anderson 1994 5KLOC Fahndrich PLDI'98 60KLOC Heintze PLDI'01 1MLOC Berndl PLDI'03 500KLOC	Landi PLDI'92 3KLOC Choi POPL'93 30KLOC	Hardekopf POPL'09 400KLOC Lhotak POPL'11 521KLOC
context sensitive	cloning	Lattner PLDI'07 350KLOC	Whaley PLDI'04 600KLOC	Emami PLDI'94 2KLOC Zhu DAC'05 200KLOC	
	summary	Fahndrich PLDI'00 205KLOC	Cheng PLDI'00 205KLOC	Wilson PLDI'95 30KLOC Chatterjee POPL'99 6KLOC Kahlon PLDI'08 128KLOC	Hongtao CGO'10 1MLOC

3.1　流敏感分析

流敏感分析是指考虑程序的控制流程,在程序分析过程中能够根据程序执行顺序注销的指针值信息,排除虚假的指针

指向集合,因此它在每个程序点都维护了指针的指向集合。流不敏感的指针分析忽略了程序的流程控制语句,从而对程序的分析是没有强更新的。下面以简单程序例子对流敏感和流不敏感的指针分析进行了区别:

```
/* 流不敏感 */
int main(){
    int x,y,*p
    P=&x;
L1:/*p→{x,y} */
    P=&y;
L2:/*p→{x,y} */
}
/* 流敏感 */
int mian(){
    int x,y,*p
    P=&x;
L1:/*p→{x} */
    P=&y;
L2:/*p→{y} */
}
```

从以上程序可以看出,使用流不敏感的分析之后,在程序 *L*1 点指针 p 有可能指向 x,也有可能指向 y。而使用流敏感的算法对程序分析之后,在 *L*1 点 p 的指向仅为 p 指向 x。从而可以看出,流敏感的分析能够使程序分析得更加精确,反映出每个程序点的具体信息,不过需要对程序进行迭代的流程分析。而流不敏感分析没有程序点的概念,如果有指针指向就将指向加入到指针集中,而不会根据程序流程对指针集进行更新。所以对于流不敏感的指针分析,分析的开销小、速度快,但不精确。

3.1.1　流不敏感分析

当前的流不敏感分析主要分为基于合并的指针分析[39]和基于包含的指针分析[40]两种。

Steensgaard[39]第一个提出了基于合并(equivalence-based)的指针分析,又被称为是 Steensggard 风格的指针分析。基于合并的指针分析主旨是将指针之间的赋值关系看做一种等价关系,只要两个指针之间产生了赋值关系,那么两个指针将指向相同的对象,或者说两个指针指向的对象是别名的。别名关系是一种等价关系,具有自反性、对称性和传递性。互相别名的对象存在于一个等价类中。Steensgaard 指向图上的节点来表示每个等价类,从而也就代表了此类中的所有对象。图 1 展示了一个简单的程序示例片段,并给出了基于合并的指针指向图。图 2 介绍了具体的分析步骤。

图1　基于合并的流不敏感指针分析

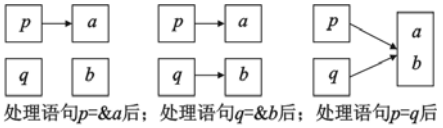

图2　基于合并的指针分析处理步骤

Andersen[40]提出了基于包含(inclusion-based)的指针分析,也称为 Andersen 风格的指针分析或者基于集合约束的指针分析。基于合并的指针分析将指针赋值关系看做一种集合的包含关系,例如若两个指针间的赋值 $p=q$ 说明了指针 q 的指向对象是 p 的指向对象的子集。Andersen 算法分为两个步骤:约束生成和约束求解。针对 C 语言规范所有的赋值语句都可以转换为如下的约束表示:

$$(p=q)\Rightarrow p\supseteq q$$
$$(p=\&q)\Rightarrow p\supseteq\{q\}$$
$$(p=*q)\Rightarrow p\supseteq q$$
$$(*p=q)\Rightarrow *p\supseteq q$$

约束求解按照如下推导规则进行:

$$\frac{a_1\supseteq\{a_2\}\quad a_1\supseteq a_3}{a_3\supseteq\{a_2\}}$$
$$\frac{a_1\supseteq *a_2\quad a_3\supseteq\{a_1\}}{a_3\supseteq\{a_2\}}$$
$$\frac{*a_1\supseteq a_2\quad a_1\supseteq\{a_3\}}{a_3\supseteq a_2}$$

图 3 展示了基于包含关系的指针分析及其指针指向图。从图中可以看出,基于包含关系的指针分析比基于合并的指针分析准确些,但是仍然不能解决指针指向强更新的问题,也就是说,使用包含关系指针分析对程序进行分析之后仍然存在指针 p 指向 a 的虚假指向未被注销掉。

图3　基于包含关系的流不敏感指针分析

3.1.2　流敏感分析

当前的流敏感指针分析主要分为两种:基于迭代数据流的分析[15,35,41]和基于静态单一赋值形式(static single assignment, SSA)的分析[33,34]。基于迭代数据流分析的方法是对每个程序点都进行指针集的保存,这样就可以精确模拟程序动态运行时的行为状态。具体例子如图 4 所示。

```
int main(){
    int a,b,*p,*q
L1: p=&a;
L2: q=&b;
L3: p=q;
}
```

在程序 *L*1 点:
$p\rightarrow a$
在程序 *L*2 点:
$q\rightarrow b$
在程序 *L*3 点:
$q\rightarrow b, p\rightarrow b$

图 4　基于迭代的流感指针分析

基于静态单一赋值形式分析的特点是将重复赋值定义的变量使用不同的变量名来表示,这样针对每一个变量就只有一次定义,从而每个变量本身就能显式地体现出数据流的信息。具体示例如下所示:

源程序

```
int main(){
    int a,b,*p,*q
L1: p=&a;
L2: q=&b;
L3: p=q;
}
```

程序的 SSA 形式

```
int main(){
    int a,b,*p,*q
L1: p1=&a;
L2: q1=&b;
L3: p2=q1;
}
```

在 SSA 形式表示的过程中,变量的使用可能用到一个特定值的产生,当且仅当在该过程的 SSA 形式中此变量的定值

和使用具有完全相同的名字。这在传统的编译优化中,简化了若干种优化转换,包括常数传播、值编号、不变代码外提和删除、部分冗余删除等,从而提高了代码优化的效率。可信程序分析也可以从这种形式中获益,如 SSA 形式实际上是流敏感的另一种变现方式,每一个被重新命名的变量都是流信息的体现,所以可信指针分析完全可以利用这种形式进行高效率的程序分析,提高软件能力可信性保障的性能。

3.2　上下文敏感分析

上下文敏感性不只是指针分析的特性,也是整个程序函数之间分析的一个共同特性。上下文敏感性是指在分析某个函数时,是否会区分不同调用上下文对此函数输入带来的不同影响。上下文不敏感分析会将所有调用上下文合并,得到一个统一的输入给该函数进行分析,然后再返回一个统一的输出结果给所有的调用点。以下例子说明了上下文敏感与上下文不敏感的区别。在两个不同的函数调用点上下文敏感分析能够区分不同的输入参数和返回结果,而上下文不敏感分析则对不同的输入参数和返回值进行统一处理。所以上下文敏感分析准确,分析的开销较大,特别是随着程序数目的增长,上下文敏感分析的效率就会随之下降。具体如图 5 所示。

```
/* 上下文不敏感 */
int main(){
    int *p, *q
    foo(&p);
    /* p→{obj}
    q→{obj} */
    foo(&q);
    /* p→{obj}
    q→{obj} */
}
```

```
/* 上下文敏感 */
int main(){
    int *p, *q
    foo(&p);
    /* p→{obj} */
    foo(&q);
    /* q→{obj} */
}
```

```
in obj;
void foo(int *x){
    *x = &obj;
}
```

图 5　上下文不敏感与上下文敏感的指针分析

上下文敏感分析的方法主要分为两种:基于克隆的分析方法[36,41]和基于摘要的分析方法[13,34,35]。基于克隆的分析方法可以理解为将函数进行内联操作,也就是说将被调用函数的函数体拷贝到调用函数的调用点,从而达到简化分析的效果。如图 6 所示,采用将 foo 函数体进行内联克隆之后,可以达到分析 main 函数同时分析 foo 函数的目的。

```
int *obj;
int main(){
    int *p, *q;
    foo(&p);
}
void foo(int *x){
    *x = &obj;
}
```

```
int main(){
    int *p, *q;
/*  foo(&p); */
    *p = &obj;
}
```

图 6　基于克隆的上下文敏感的指针分析

另一种基于摘要的上下文敏感分析是利用对函数进行总结的方式进行的,例如将被调用函数的输入与输出之间建立转移方程来描述调用点对变量的副作用。当分析到函数的调用点时,可以根据对被调用函数的摘要总结来进行分析,而不用每次都将整个函数体进行内联克隆,具体例子如图 7 所示。在调用点 *L*1 处将建立转移函数 TF(p) = 〈obj〉,用来描述此次调用对指针 p 的副作用影响,p 经过此调用将指向 obj。

```
int *obj;
int main(){
    int *p, *q;
L1: foo(&p);
}
void foo(int *x){
    *x = &obj;
}
```

```
int *obj;
int main(){
    int *p, *q;
L1: foo(&p);
/* TF(p) = 〈obj〉 */
}
void foo(int *x){
    *x = &obj;
}
```

图 7　基于摘要的上下文敏感的指针分析

4　结束语

指针分析是目前保障程序可信性的一个研究热点问题。本文对指针分析技术研究现状进行了分析,并在此基础上描述了指针分析对软件可信性保障的作用。面对软件规模、程序复杂性的不断增长,人们对软件的可靠、安全、实时等非功能属性的要求不断增加,指针分析的精确与否、效率高低直接决定了程序分析性能及能力可信保障的水平。

指针分析在保证程序可信性方面尚处于起步和发展阶段,精确高效的指针分析研究还处于发展中,远没有达到保证软件能力可信性的要求。所以精确高效的指针分析设计及算法面临着众多的机遇和挑战。

参考文献:

[1] Da SILVA J, STEFFAN J G. A probabilistic pointer analysis for speculative optimizations[C]//Proc of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press,2006:416-425.

[2] WEGMAN M N, ZADECK F K. Constant propagation with conditional branches[J]. ACM Trans on Programming Languages and Systems,1991,13(2):181-210.

[3] CHOW F, CHAN S, KENNEDY R, *et al*. A new algorithm for partial redundancy elimination based on SSA form[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,1997:273-286.

[4] KENNEDY R, CHAN S, LIU Shin-ming, *et al*. Partial redundancy elimination in SSA form[J]. ACM Trans on Programming Languages and Systems,1999,21(3):627-676.

[5] KNOOP J, RUTHING O, STEFFEN B. Partial dead code elimination [C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,1994:147-158.

[6] XIE Yi-chen, AIKEN A. Context-and path-sensitive memory leak detection[C]//Proc of the 10th European Joint Meeting of Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York: ACM Press,2005:115-125.

[7] MEDEZ-LOJO M, MATHEW A, PINGALI K. Parallel inclusion-based points-to analysis[C]//Proc of ACM International Conference on Object Oriented Programming Systems Languages and Applications. New York: ACM Press,2010:428-443.

[8] LIVSHITS V B, LAM M S. Tracking pointers with path and context sensitivity for bug detection in C programs[C]//Proc of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press,2003:317-326.

[9] XIE Yi-chen, AIKEN A. Scalable error detection using Boolean satisfiability[C]//Proc of the 32nd ACM SIGPLAN-SIGACT Symposium

on Principles of Programming Languages. New York: ACM Press, 2005:351-363.

[10] CHEREM S, PRINCEHOUSE L, RUGINA R. Practical memory leak detection using guarded value-flow analysis[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM,2007:480-491.

[11] NAIK M, AIKEN A, WHALEY J. Effective static race detection for Java[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2006: 308-319.

[12] KAHLON V, SINHA N, KRUUS E, *et al*. Static data race detection for concurrent programs with asynchronous calls[C]//Proc of the 7th European Joint Meeting of Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering. New York: ACM Press,2009:13-22.

[13] KAHLON V. Bootstrapping: a technique for scalable flow and context-sensitive pointer alias analysis[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,2008:249-259.

[14] XIE Yi-chen, AIKEN A. Saturn: a SAT-based tool for bug detection [C]//Proc of the 17th International Conference on Computer Aided Verification. Berlin: Springer,2005:139-143.

[15] ZHU Jian-wen, CALMAN S. Context sensitive symbolic pointer analysis[J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2005,24(4):516-531.

[16] SEMERIA L, De MICHELI G. Resolution, optimization, and encoding of pointer variables for the behavioral synthesis[J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems,2002,20(2):213-233.

[17] SEMERIA L, De MICHELI G. SpC: synthesis of pointers in C, application of pointer analysis to the behavioral synthesis from C[C]// Proc of the 1998 IEEE/ACM International Conference on Computer-Aided Design. New York: ACM Press,1998:340-346.

[18] SRIDHARAN M, FINK S J, BODIK R. Thin slicing[C]//Proc of the 16th ACM SIGSOFT International Symposium on Programming Language Design and Implementation. New York: ACM Press,2007: 112-122.

[19] AGRAWAL H, HORGAN J R. Dynamic program slicing[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,1990:246-256.

[20] LE Wei, SOFFA M L. Marple: a demand-driven path-sensitive buffer overflow detector[C]//Proc of the 16th European Joint Meeting of Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering. New York: ACM Press,2008: 272-282.

[21] HOVEMEYER D, SPACCO J, PUGH W. Evaluating and tuning a static analysis to find null pointer bugs[C]//Proc of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software tools and Engineering. New York: ACM Press,2006:13-19.

[22] HEINE D L, LAM M S. A practical flow-sensitive and context-sensitive C and C++ memory leak detector[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,2003:168-181.

[23] PRATIKAKIS P, FOSTER J S, HICKS M. LOCKSMITH: context-sensitive correlation analysis for race detection[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,2006:320-331.

[24] KAHLON V, YANG Yu, SANKARANA R S, *et al*. Fast and accurate static data-race detection for concurrent programs[C]//Proc of the 19th International Conference on Computer Aided Verification. London: Springer-Verlag,2007:226-239.

[25] HACKETT B, AIKEN A. How is aliasing used in systems software? [C]//Proc of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press,2006: 69-80.

[26] HIND M, PIOLI A. Which pointer analysis should I use? [C]// Proc of ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press,2000:113-123.

[27] LANDI W. Undecidability of static analysis[J]. ACM Letters on Programming Languages and Systems,1992,1(4): 323-337.

[28] RAMALINGAM G. The undecidability of aliasing[J]. ACM Trans on Programming Languages and Systems,1994,16(5):1467-1471.

[29] GHIYA R, HENDREN L J. Putting pointer analysis to work[C]// Proc of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM Press,1998:121-133.

[30] HIND M. Pointer analysis: haven't we solved this problem yet? [C]//Proc of ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York: ACM Press, 2001:54-61.

[31] 于洪涛, 张兆庆. 激进域敏感基于合并的指针分析[J]. 计算机学报,2009,32(9):1722-1735.

[32] 于洪涛. 精确高效的C语言指针分析技术研究[D]. 北京:中国科学院计算技术研究所,2010.

[33] HARDEKOPF B, LIN C. Semi-sparse flow-sensitive pointer analysis [C]//Proc of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM Press, 2009:226-238.

[34] YU Hong-two, XUE Jing-ling, HUO Wei, *et al*. Level by level: making flow-and context-sensitive pointer analysis scalable for millions of lines of code[C]//Proc of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization. New York: ACM Press,2010:218-229.

[35] WILSON R P, LAM M S. Efficient context-sensitive pointer analysis for C programs[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 1995:1-12.

[36] WHALEY J, LAM M S. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,2004:131-144.

[37] PEARCE D J, KELLY P H J, HANKIN C. Efficient field-sensitive pointer analysis of C[J]. ACM Trans on Programming Languages and Systems, 2007,30(1):37-42.

[38] GUTZMANN T, LUNDBERG J, LOWE W. Towards path-sensitive points-to analysis source[C]//Proc of the 7th IEEE International Working Conference on Code Analysis and Manipulation. Washington DC: IEEE Computer Society,2007:59-68.

[39] STEENSGAARD B. Points-to analysis in almost linear time[C]// Proc of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM Press,1996:32-41.

[40] ANDERSEN L O. Program analysis and specialization for the C programming language[D]. Copenhagen: University of Copenhagen, 1994.

[41] EMAMI M, GHIYA R, HENDREN L J. Context-sensitive interprocedural points-to analysis in the presence of function pointers[C]// Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press,1994:242-256.