软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(11):2486−2498 [doi: 10.13328/j.cnki.jos.004596]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于指向更新的优先权指针分析算法*

刘 鹏, 赵荣彩, 庞建民, 姚 远

(解放军信息工程大学,河南 郑州 450002)

通讯作者: 刘鹏, E-mail: liu_peng_2011@126.com

摘 要: 指针分析是数据流分析中的关键性技术,其分析结果是编译优化和程序变换的基础.在基于包含的指针分析算法研究的基础上,对 Narse 优先权约束评估算法中存在的冗余约束评估和优先权评估模型计算开销较大的问题进行分析,以指针的指向集更新信息确定约束评估的候选集,提出了基于指向更新的约束评估算法.采用约束语句间的解,引用依赖和标量依赖构建约束依赖图,通过依赖关系确定约束评估的优先权,提出了基于约束依赖图的优先权算法,简化了既有算法中复杂的优先权评估模型,进一步给出了优化后算法的整体框架.在基准测试集 SPEC 2000/SPEC 2006 上进行实验,其结果表明,该算法与 Narse 优先权算法相比,在时间开销和存储开销上都有明显的性能提升.

关键词: 指针分析;数据流分析;指向集;流不敏感

中图法分类号: TP314

中文引用格式: 刘鹏,赵荣彩,庞建民,姚远.基于指向更新的优先权指针分析算法.软件学报,2014,25(11):2486−2498. http://www.jos.org.cn/1000-9825/4596.htm

英文引用格式: Liu P, Zhao RC, Pang JM, Yao Y. Prioritizing pointer analysis algorithm based on points-to updating. Ruan Jian Xue Bao/Journal of Software, 2014,25(11):2486−2498 (in Chinese). http://www.jos.org.cn/1000-9825/4596.htm

Prioritizing Pointer Analysis Algorithm Based on Points-to Updating

LIU Peng, ZHAO Rong-Cai, PANG Jian-Min, YAO Yuan

(PLA Information Engineering University, Zhengzhou 450002, China)

Corresponding author: LIU Peng, E-mail: liu_peng_2011@126.com

Abstract: Pointer analysis is a key technology in data flow analysis, and the result of pointer analysis is the basis of compiler optimization and program transformation. Based on the inclusion-based pointer analysis algorithm research, this paper analyzes the problems of redundant constraints evaluation and computational overhead of priority evaluation model in Narse priority constraints evaluation algorithm. Candidate set of constraint evaluation is determined by points-to set updating information of pointers, and the prioritizing pointer analysis algorithm based on points-to updating is presented. Constraint dependency graph is built by pointer dereference dependence and pointer scalar dependence in constraint statements, and priority of constraint evaluation is determined by the dependencies. Prioritizing algorithm based on constraint dependency graph is presented to simplify the complex priority evaluation model in Narse algorithm, and the overall framework of the optimized algorithm is provided. The experimental results on SPEC 2000/SPEC 2006 benchmark show that the algorithm has a significant performance boost on the time overhead and storage overhead compared with Narse priority algorithm.

Key words: pointer analysis; dataflow analysis; points-to set; flow-insensitive

指针是编程语言中常见的数据结构,其广泛存在于 C,C++等编程语言中.指针的使用给程序员带来了灵活的编程习惯,以及便捷、高效的编程方式.然而在广泛使用指针的程序中,会出现两个或多个变量访问重叠的存

* 基金项目: 国家高技术研究发展计划(863)(2009AA012201); “核高基”国家科技重大专项(2009ZX01036-001-001-2)
收稿时间: 2013-03-06; 修改时间: 2014-01-21; 定稿时间: 2014-03-27

储位置,由此引起的指针别名会使得编译器难以理解程序的行为,并阻碍编译器进行一些潜在的优化.指针分析这一研究课题即由此引起,它是一项基本的数据流分析技术,能够计算指针变量可能指向的内存对象的集合,能够分析两个指针变量是否为指针别名,其分析结果往往是其他程序分析和变换的基础,包括自动向量化[1]、安全分析[2]、bug 检测[3]、硬件合成[4]和多线程程序分析[5]等等.这些分析的效果和效率严重依赖于所提供的指针分析的精度.

然而,精确的指针分析是 NP-hard 问题[6],任何指针分析的结果都是运行时正确指向集的近似.因此,许多研究工作围绕指针分析算法的精度改进展开,主要包括考虑控制流依赖的流敏感分析[7−9]、考虑函数调用语义的上下文敏感分析[10−12]和考虑聚类数据类型成员指向关系的域敏感分析[13,14]等等.上述研究工作在不同程度上改进了算法的精度,然而随着精度的提升,对算法分析的可伸缩性带来了严重的挑战.尤其是近年来,“大数据”引起了工业界和学术界的广泛关注.在编译领域,随着测试集规模的逐步增大,可伸缩性成为指针分析算法设计的一个关键要素.

因此,除了精度方面的算法改进以外,可伸缩性是指针分析算法的另一研究热点.虽然流敏感的分析[8,15]已经能够处理百万行规模的测试集,却耗费了大量的内存空间,1 946 K 规模的 tshark 测试集需要在 64 位机上 100 GB 的环境下才能够运行.与之对应,流不敏感的指针分析在占有较少资源的同时具有较高的性能,且广泛存在于一些主流编译器中.两类传统的指针分析算法是 Anderson 提出的基于包含的指针分析算法[16]和 Steensgaard 提出的基于合并的指针分析算法[17].前者具有最坏情况下 $O(n^3)$的时间复杂度,其中,n 为指针变量的规模,主要应用于 GCC 和 LLVM 等编译器中;后者具有近似 $O(n)$的时间复杂度,主要应用于 Open64 等编译器中.之后的诸多工作都在这两种算法的基础上进行改进,基于合并的指针分析算法的优势是具有较高的时间复杂度,但以损失精度为代价.这种精度的损失往往无法让人接受,基于 Steensgaard 算法的改进工作也逐渐淡化.

本文主要在基于包含的指针分析领域展开研究.第 1 节给出基于包含的指针分析的算法分类及本文的研究动机.第 2 节对 Narse 提出的优先权算法中存在的问题进行分析.第 3 节在约束集评估顺序的研究范畴内提出一种指向更新的约束评估算法.第 4 节通过引入约束依赖图,提出一种基于约束依赖图的算法优化.第 5 节给出整个算法的框架并分析算法的优越性.第 6 节对我们的方法和 Narse 算法进行实验和比较.第 7 节给出结论并进行展望.

1 相关研究

本节重点分析基于包含的指针分析算法改进中各个维度的相关工作,并指出可分析的空间及本文的研究动机.

基于包含的指针分析将所有的指针语句分为如下 4 种形式:地址约束(p=&q)、拷贝约束(p=q)、LOAD 约束(p=*q)和 STORE 约束(*p=q),其中,LOAD 约束和 STORE 约束也称为复杂约束.如图 1 所示为基于包含的指针分析的一般算法.在简述算法前,首先给出约束图的定义.

```
procedure Inclusion_based_Pointer_Analysis(constraint graph G = NULL, constraint stmt set C)
(1)  (G1 = (V, null, PT)) = Process_Address_Constraints(C, G = NULL);
(2)  (G' = (V', W', PT)) = Process_Copy_Constraints(G = (V', W, PT'));
(3)  while (fixpoint)
(4)      Process_Load_Constraints(G');
(5)      Process_Store_Constraints(G');
(6)      Propagate_Points_to_Information(G');
(7)  end while
end.
```

Fig.1 Inclusion-Based points-to analysis algorithm

图 1 基于包含的指向分析算法

定义 1. 约束图是一个三元有向图 G=(V,E,PT),其中,V 称为顶点集;E 称为边集,E 的元素(x,y)是一条有向边,

其中,$x,y\in V$;PT 称为指向集,PT 的元素为 $x\rightarrow pt_x$,pt_x 为变量 x 的指向集.

图 1 中的算法根据约束语句逐步生成带有指向关系的约束图,L1 首先处理地址约束得到初始的指向信息;而后,L2 处理拷贝约束在约束图中添加初始的指向边,L1 和 L2 的工作称为约束生成;L3~L7 为对复杂约束的评估和指向信息的传播工作,到达定点后算法结束,该工作称为约束求解.

基于包含的指针分析算法的大量工作都是针对该算法原型的改进,近年来,在该领域较为典型的是德克萨斯大学的 Hardekopf[18,19]、印度科学院的 Nasre[20,21]和 UCLA 的 Pereira[22]等人的研究工作.中国科学院的陈聪明等人于 2011 年对近 20 年的研究工作进行了系统全面的综述[23],通过进一步研究,针对优化时机、优化阶段、评估方式等因素的不同,可将算法做如图 2 所示的分类.

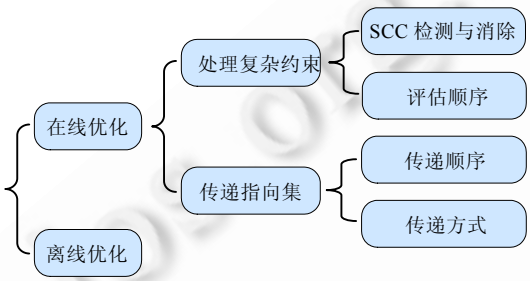

Fig.2 Classification of inclusion-based points-to analysis algorithms

图 2 基于包含的指向分析算法分类

根据优化的时机,可分为在线优化和离线优化.在约束生成之后、约束求解之前(L2 和 L3 之间)进行的优化为离线优化,在约束求解过程中进行的优化(L3~L7)为在线优化.

在处理复杂约束(L4,L5)时,会在约束图中添加新的边.Fahndrich 等人[24]指出:在新边加入约束图时,可检测并消除强连通分量(strongly connected component,简称 SCC)以提高算法效率.Pearce 等人提出的 PKH1 算法[25]通过在约束图上维持拓扑关系,在拓扑关系变化时检测及消除 SCC,适用于规模较大的程序分析.Harderkopf 等人提出的 LCD 算法[19]弱化了 SCC 的检测要求,假设两相邻节点的指向集相同则可能处于同一个 SCC,以减少计算冗余.Pearce 等人提出的 PKH2 算法[13]和 Pereira 等人提出的 WP 算法[22]在迭代中仅对 SCC 进行 1 次检测,适用于 SCC 较少的程序分析.在评估顺序的研究中,Narse[21]引入了约束优先级的概念,通过构建约束评估模型,将输入的复杂约束进行动态排序,有效地改进了分析时间.

在指向集传播中(L6),可针对指向信息的传递顺序进行优化[19,22].在维持动态拓扑顺序的同时,优先处理最近最少访问的节点.在传递方式的研究中,Pearce 等人[25]提出的差异传播技术仅传递更新后的指向信息,减小了传递指向集的开销.

在线优化的 4 类算法中:SCC 检测与消除的研究较为成熟;传递顺序和传递方式的研究较少,一般作为通用的算法在指向分析中采用;而评估顺序方面由 Narse 提出,并在主流测试集中有 20%的性能提升.本文的研究工作在评估顺序的研究范畴内展开,在 Narse 算法的基础上给出一种更为简洁、高效的约束评估算法.

2 Narse 算法问题分析

Narse 在研究中提出了两点疑问:一是一条约束语句会在约束图中添加几条边?二是一条约束语句会在约束图中的什么位置添加边?这两点因素严重影响了算法到达定点的效率.Narse 在此基础上提出了基于优先权评估的指针分析算法[21],约束评估的每次迭代中对所有复杂约束依次进行评估,并计算每条约束语句新添加的指向关系的数目 i,依据 i 值计算下次迭代中约束评估的优先级 p,以此确定下次迭代的评估顺序.

该算法针对两点疑问建立了优先权评估模型,在计算过程中,约束优先级动态更新,较高优先级的语句优先进行评估,有利于每次评估在约束图中添加更多的边以及更有效的边.

虽然该算法有一定的性能提升,然而算法中存在一定的计算冗余和繁琐的数据结构,对该算法的改进基于

两点观察:

(1) 迭代过程中存在大量无用的复杂约束评估;

(2) 优先权评估相关结构的动态维护开销较大.

对于观察(1),若某指针变量在第 i 次迭代中无指向信息的更新,则在第 i+1 次迭代中对涉及该指针变量解引用的复杂约束进行评估时不会添加新的边,这些冗余评估在文献[21]示例中所占的比重为 58.3%,耗费了一定的时间开销.因此,评估过程中仅需将能够引入新的指向信息的约束提取到候选集合中,每次的约束评估仅需从候选集中选取约束,并在传播指针信息后对候选集合进行更新.

对于观察(2),算法对每次迭代引入了评估函数集 $F=\{f_1,f_2,\ldots,f_i,\ldots\}$,由 $f_i(c)=p$ 确定每条约束语句的优先级.在迭代中需要动态维护每条复杂约束的优先权,计算开销包含优先权计算、优先权排序等,计算量与迭代次数相关.即使通过倾斜评估[21]进行优化,仍不能从数量级上减小总的评估次数,无法显著减小计算开销.从约束图上直观进行分析,指向关系会沿着约束图的拓扑序进行传播,首先评估排在拓扑序前面的节点涉及的复杂约束,就能够传播更多的指向信息,而约束语句之间的依赖关系隐含决定了指针变量在拓扑图上的先后顺序.无论对于解引用指针之间的依赖还是指针标量之间的依赖,首先评估依赖的源点约束能够传播更多的指向信息.那么,采用约束语句的依赖关系进行评估取代 Narse 的优先权评估模型,将是确定复杂约束评估顺序的一个改进方向.

3 基于指向更新的约束评估算法

下面,我们主要针对以上存在的问题,通过实例分析给出指向更新的约束评估分析的优势,并进一步提出基于约束依赖图的优先权算法优化.

该算法的关键是在相邻的迭代间保存每个指针变量的指向信息,依据更新的指向信息,计算下次迭代评估的候选集.若指针变量 a 在第 i+1 次迭代较第 i 次迭代存在指向更新,即 $PT_{i+1}(a)-PT_i(a)\neq\varnothing$,则在第 i+2 次迭代评估中,候选集中将包含形如(…=*a),(*a=…)的复杂约束.

下面采用文献[21]中的示例对该思想进行阐述.如图 3 所示为基于指向更新的约束评估过程,第 0 次迭代中给出了初始的指向信息,拷贝约束为{$e=d$,$b=a$},复杂约束为{*$e=c$,c=*a,*$a=p$}.图中斜体字表示更新的指向信息,细线表示当前迭代前引入的边,粗线表示当前迭代中新引入的边.

最初的两步与传统的 Anderson 算法[16]相同,由地址约束初始化指向信息,而后使用拷贝约束在约束图中添加边.

在第 1 次迭代之前,拷贝约束 $e=d$ 和 $b=a$ 分别添加(d,e)及(a,b)两条有向边.在约束图中传播指向信息后,当前的更新指向信息为:$PT_0(a)=\{a,q,r,s,t\}$,$PT_0(b)=\{a,q,r,s,t\}$和 $PT_0(p)=\{b,c,d\}$.由于下次迭代的约束评估中仅需考虑解引用指针指向信息更新的约束语句,因此,下次迭代的候选集为{c=*a,*$a=p$}.

在第 1 次迭代中,约束语句(c=*a)添加了(a,c)及($qrst$,c)两条有向边,*$a=p$ 语句添加了(p,a)及(p,$qrst$)两条有向边.进行指向信息传播后,得到图中所示一次迭代后的约束图,由指向更新信息计算 $PT_1(a)=\{b,c,d\}$,可得下次迭代的候选集为{c=*a,*$a=p$}.

分析过程中依次进行迭代,得到迭代 2、迭代 3 和迭代 4 所示的约束图,约束评估在满足以下两种条件之一时达到聚合:

(1) 指向传播后未生成新的指向信息;

(2) 即使生成了新的指向信息,但依据该指向信息计算得到的候选集为空集.

约束评估在第 4 次迭代后得到的候选集为空集,满足条件 2,分析过程达到聚合.

该算法与 Narse 相比,算法达到聚合均进行了 4 次迭代.然而,该算法每次迭代中仅对候选集进行约束评估,从输入规模减小了算法的时间复杂度,每次迭代后,仅需依据指向信息的变化评估新的候选集,生成新的候选集的时间复杂度为 $O(n)$,n 为复杂约束的规模.下面我们以一种更为简洁的方法替代 Narse 的优先权模型对该算法进行优化,并在此基础上给出优化后算法的整体框架.

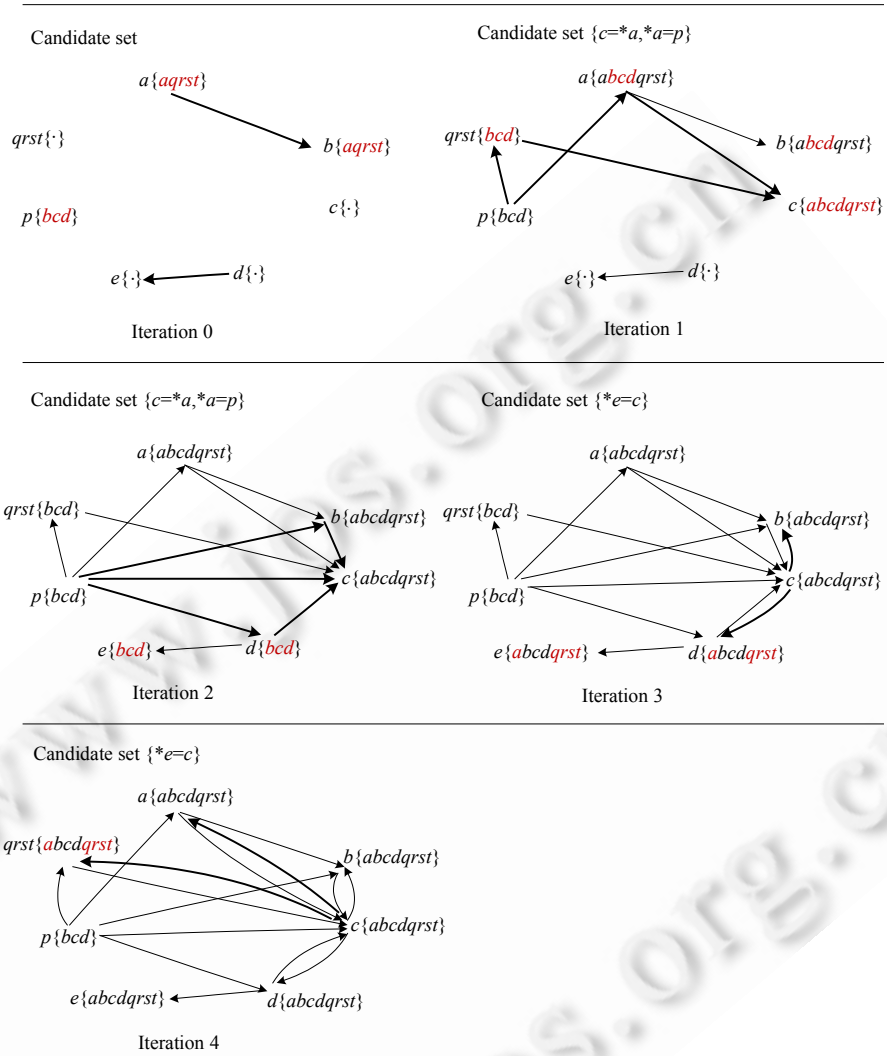

Fig.3 Constraint evaluation process based on points-to updating

图 3 基于指向更新的约束评估过程

4 基于约束依赖图的优先权算法

上一节给出的基于指向更新的约束评估通过候选集分析,避免了大量无用的复杂约束评估.在观察(2)的基础上进行分析,在优先权评估模型上仍具有一定的改进空间.

4.1 约束依赖图的构造

直观上认为,对观察(2)的改进可通过发掘约束语句间的依赖关系进行约束评估.这样既能改善按照程序顺序进行评估的固有缺陷,又极大地简化了复杂的优先权评估模型.

图 4 为指针解引用依赖对约束评估的影响,下面对该例进行分析,阐述按照依赖关系确定评估顺序的有效性.图中语句 S1 和 S2 分别为*b=a 和 c=*b,从 S1 到 S2 具有指针 b 的解引用依赖,指针 b 的指向集为 $PT(b)$.对约束语句进行评估后,指向信息将沿着箭头方向传播,若优先评估依赖的源点语句 S1,评估次序为 S1,S2,则指向信息的传播路径为 e1,e2,需要两步;而若优先评估依赖的汇点语句 S2,评估次序为 S2,S1,则指向信息的传播路径为

e2,e1,e2,需要 3 步.

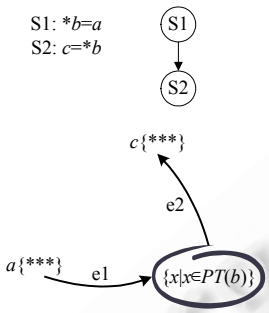

Fig.4 Impact of pointer dereference dependence on constraint evaluation

图 4 指针解引用依赖对约束评估的影响

在约束评估会改变解引用变量指向集的情况下,即包含前提 $b \in PT(b)$,那么优先评估依赖源点语句的评估顺序为 S1,S1,S2,需要 3 步;而优先评估依赖汇点语句的评估顺序为 S2,S1,S1,S2,需要 4 步.因此,按照依赖关系确定约束评估顺序既能减少指向传播的次数,又能确保算法更快地达到定点.

在这里,本文引入约束依赖图的概念,图中的节点表示约束语句,边表示语句之间的依赖关系.依赖关系包括标量依赖和解引用依赖,以图 3 中的输入语句为例,约束依赖图的构造过程如图 5 所示:首先,S2 到 S1 存在标量指针 c 的依赖,构造图中左上角所示的标量依赖图(scalar dependence graph,简称 SDG);同时,S5 到 S2 存在解引用指针 a 的解引用依赖,构造图中左下角所示的解引用依赖图(dereference dependence graph,简称 DDG);而后,将 SDG 和 DDG 合成为右侧所示的约束依赖图(constraint dependence graph,简称 CDG).对于在约束依赖图上出现依赖环的复杂的情况,则可把依赖环上的所有节点看成一个节点,消除强连通分量.

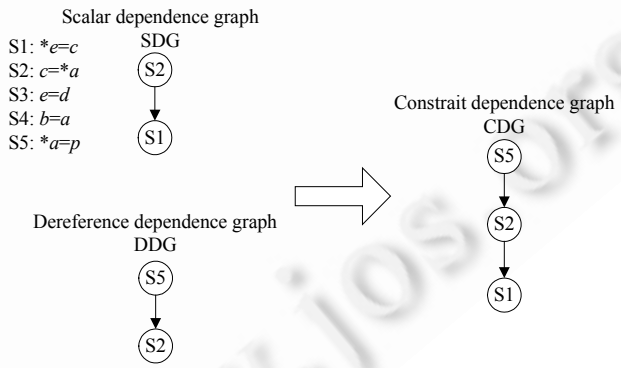

Fig.5 Build of constraint dependence graph

图 5 约束依赖图的构造

4.2 优先权算法优化

引入约束依赖图后,约束评估将按照依赖关系优先评估依赖的源点.考虑约束评估中的指向更新,若该源点语句更新了语句中解引用指针的指向信息,则再次评估依赖源点语句能够在约束图添加新的边,并传播更多的指向信息.因而,每次从工作集列表中选取一条语句进行评估,替代每次迭代评估所有约束语句,可能会减少总的约束评估及指向信息的传播次数,提高算法的效率.基于以上分析,与指向更新的约束评估相结合,优先权算法可进行以下两点改进:

(1) 依照约束依赖图确定约束评估的顺序;

(2) 每次迭代中仅评估候选集中的一条最优语句.

优化后算法的约束评估过程如图 6 所示.第 1 次迭代前,指针变量 a 具有更新的指向信息,则输入候选集为{*a=p,c=*a}.在两条约束语句中,*a=p 处于依赖的源点,具有更高的优先权,因此在第 1 次迭代中对约束语句*a=p 进行评估.

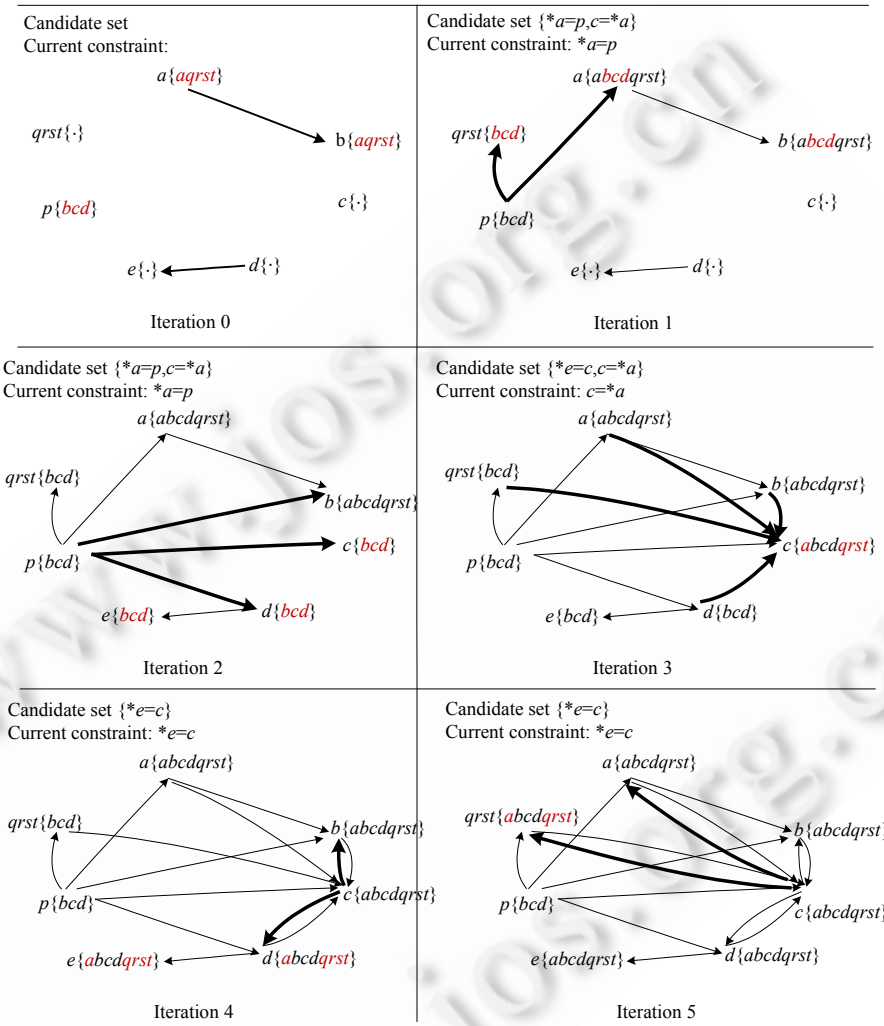

Fig.6 Constraint evaluation process of optimized algorithm

图 6 优化后算法的约束评估过程

在第 3 次迭代前,输入候选集为{*e=c,c=*a},c=*a 在约束依赖图中具有较高的优先权,本次迭代对 c=*a 进行评估.

第 5 次迭代后,由更新的指向信息计算得到的候选集为空集,算法达到聚合.

与优化前指向更新的约束评估相比,之前进行 4 次迭代算法达到聚合,共进行 6 次约束评估;而优化后仅进行 5 次约束评估即达到算法聚合,优化了对约束语句 c=*a 的一次冗余评估.

表 1 列出了几种典型约束评估算法的计算开销对比,评估算法包括 Anderson 算法、Nasre 算法、指向更新算法和优先权优化算法,评估参数包括约束评估次数、冗余的约束评估次数和指向信息传播次数,其中,冗余的约束评估次数指评估中不会在约束图中添加新边的无效约束评估.通过对比,指向更新算法和优先权优化算法不存在冗余的约束评估,优先权优化算法在约束评估次数及指向信息传播次数上也要优于其他算法.

Table 1 Four algorithms computational overhead comparison

表 1 4 种算法的计算开销对比

评估参数	Anderson 算法	Nasre 算法	指向更新算法	优先权优化算法
约束评估次数	15	12	6	5
冗余的约束评估次数	9	7	0	0
指向信息传播次数	59	53	59	52

5 算法框架及分析

5.1 算法的整体框架

在上述分析的基础上,将指向更新的约束评估和基于约束依赖图的优先权算法相结合,给出基于指向更新的优先权约束评估的形式化描述:

基于指向更新的优先级约束评估框架是一个七元组:$R=(C,Ca,P,G,\delta,f,\leqslant)$,其中,

1. C 是输入约束集合.
2. $Ca\subseteq C$,是输入候选集合.
3. P 是优先权集合.
4. $G=(V,E,PT)$是约束图,V 为顶点集合,E 为边的集合,PT 为指向信息的集合.
5. δ是指向更新转移函数,$G\times C\times\delta=G\times Ca$.
6. f 是优先权评估函数,$f(c_i)=p_i$,其中 $c_i\in C,p_i\in P$.
7. $\leqslant$是约束语句优先权的偏序关系,若 $p_i\leqslant p_j$,则 p_i 与 p_j 相比具有较高的优先权.

在约束评估中,需要计算每条复杂约束的优先权,通过优先权评估函数 $f(c_i)=p_i$ 在第 1 次迭代前计算得到,在后续计算过程中无需动态维护.约束语句的优先权计算方法如图 7 所示,图中依据依赖的深度进行层次划分,依赖源点的约束语句具有更高的优先权.对于有依赖环的特殊情况,依赖环上的节点具有同等的优先权,需打破依赖环,将依赖环上的节点合成一个节点,并将合成前各节点的出边入边添加到合成后的新节点上.

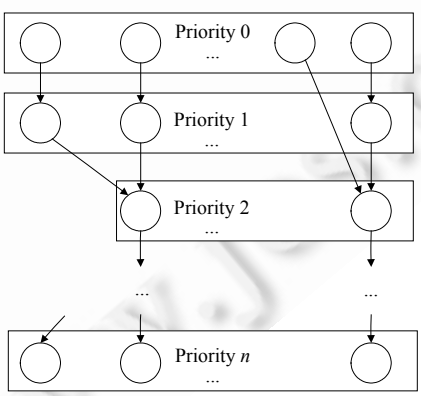

Fig.7 Priority calculation method of the constraint dependence graph

图 7 约束依赖图的优先权计算方法

图 8 给出了基于指向更新的优先权指针分析算法的伪代码.首先,由输入约束语句集合 C 构建约束依赖图;对集合 C 中的每条约束语句 c_i 依照图 7 的方法计算相应的优先级 p_i.

L5~L7 为约束图的 G 初始化部分,通过函数 *Process_Address_of_Constraints* 处理地址约束得到初始的指向信息,并通过函数 *Process_Copy_Constraints* 处理拷贝约束在约束图 G 中添加新的边,函数 *Propagation_PT_Info* 在上述处理的基础上传播初始的指向信息.L8 中的函数 *Compute_Candidate_Set* 针对初始化后的约束图和输入

约束语句的集合 C 计算得到当前候选集 Ca.

procedure *Prioritized_Points_to_Update_Analysis*(***constraint graph*** $G=(V,E,PT)$, ***constraint stmt set*** C)
(1) $cdg = Build\_CDG(C)$;
(2) **for** each $p_i \in priority\ P$ and $c_i \in C$ **do**
(3) $p_i = Compute\_Priority\_Level(c_i, cdg)$;
(4) **endfor**
(5) $Process\_Address\_of\_Constraints(C, G)$;
(6) $Process\_Copy\_Constraints(C, G)$;
(7) $Propagation\_PT\_Info(G)$;
(8) $Ca = Compute\_Candidate\_Set(C, (V, E', PT''))$;
(9) **while** $(Ca \neq \varnothing)$
(10) select c_i if $c_i \in Ca$ $c_j \in Ca (i \neq j)$ and $p_i \geqslant p_j$
(11) $(V, E', PT) = Constraint\_Evaluate(c_i, (V, E, PT))$;
(12) $(V, E', PT') = Propagation\_PT\_Info((V, E', PT))$;
(13) **if** $(PT' - PT == \varnothing)$ **break**;
(14) $Cu = Compute\_Update\_Candidate(C, PT' - PT)$;
(15) **if** $c_i \notin Cu$ **then**
(16) $Ca = Ca - \{c_i\}$;
(17) **endif**
(18) $Ca = Ca \cup Cu$;
(19) **endwhile**
end.

Fig.8 Prioritizing pointer analysis algorithm based on points-to updating
图 8 基于指向更新的优先权指针分析算法

L9~L19 的循环为算法实现的主体部分,通过选取候选集中最高优先级的语句参与约束评估 *Constraints_Evaluate*,调用函数 *Propagation_PT_Info* 传播指向信息后,指向信息由 PT 更新为 PT'.L14~L18 为通过转移函数 δ 得到下次迭代候选集 Ca 的过程,计算细节为通过 *Compute_Update_Candidate* 函数依据更新的指向信息 $PT'-PT$ 计算更新的约束集合 Cu,并利用 Cu 更新当前候选集 Ca,进行下一次迭代.

当候选集为空(L9)或指向更新为空(L13)时,跳出循环,意味着算法到达聚合.

5.2 算法的优越性

本文所提出的算法与 Narse 算法相比的优越性重点体现在以下两个方面:

(1) 减小了评估的规模

Narse 算法在每次迭代中对所有复杂约束进行了评估,本文算法引入了候选集,有效减小了评估的规模.设评估过程的迭代次数为 l,复杂约束的数目为 n,若从第 i 次迭代到第 i+1 次迭代存在指向更新的节点数为 k_i,即对所有的 $x \in V$(V 为约束图中的顶点集),$PT_{i+1}(x)-PT_i(x) \neq \varnothing$的数目,若 k_i 个存在指向更新的节点涉及的复杂约束为 $m_i (m_i \leqslant n)$个,则约束评估的规模将由 $l \times n$ 次减小到 $m_1+m_2+\ldots+m_l$ 次,冗余评估所占比重: $p=1-\frac{m_1+m_2+\ldots+m_l}{l \times n}$. 随着程序规模的增大,$p$ 值将越来越大,更能体现改进后算法的优越性.

(2) 改进了优先权评估模型

Narse 算法中的优先权需要动态维护,采用评估函数集 $F=\{f_1, f_2, \ldots, f_i, \ldots\}$,在每次迭代中,由 $f(c_i)=p$ 计算优先级.本文引入约束评估图,有效改进了优先权评估函数,仅需在第 1 次迭代前由 $f(c)=p$ 进行一次计算.同时,优先评估依赖的源点能够减少指向信息的传播次数,有序传播更多的指向信息,避免冗余传播,确保算法更快地到达定点.

这种优越性具体到算法复杂度上,可做如下分析:

若指针变量的个数为 n,约束语句的数量为 m,则 L1 构建约束依赖图的时间复杂度为 $O(2m^2)$;L2~L4 针对每条约束语句初始化优先级,复杂度为 $O(m^2)$;L8 计算当前候选集需要对所有指针变量进行遍历,查找是否存在更新的指向信息,其时间复杂度为 $O(mn)$;L9~L19 的主体循环部分依次从候选集中选取约束语句进行评估,最坏情况下迭代的次数为 $O(2m)$次.在主体循环的每次迭代中,L11 进行约束评估的时间复杂度为 $O(n)$,最坏情况下传播指向信息的复杂度为 $O(n^2)$,计算更新候选集的时间复杂度为 $O(m)$.

因此,该算法的时间复杂度为 $O(2m^2+mn+2m(n+n^2+m))\approx O(m^2+mn^2)$.理论上,若指针变量的个数为 n,则最坏情况下,约束语句的数量为 $O(2^n)$.因此,该算法优于 Narse 算法的分析 $O(m^2n+l^2mn)$.

6 实验与分析

6.1 实验环境及测试集

该算法基于开源编译器 CIL 1.5.1 实现.CIL 是 INRIA 公司采用函式编程语言 O'Caml 研发的针对 C 语言的源源变换编译器,ptranal.ml 为约束生成阶段,olf.ml 和 golf.ml 为约束求解阶段.在 CIL 指针分析框架中,我们实现了最具可伸缩性的深度传播算法,基本的优化策略包括 Pearce 的差异传播技术和离线优化中的离线变量替换,在评估顺序上实现了 Narse 提出的优先权算法和基于指向更新的优先权指针分析算法.由于 O'Caml 具有高效、简洁的函式编程特性,两种算法实现均不足 3 000 行代码.

实验环境为 IBM 3850 服务器,处理器频率 2.0GHz,内存 4GB,L1 数据缓存为 32KB,L2 缓存为 256KB,基本页面为 8KB.操作系统内核为 Linux 2.6.18,版本为 Redhat Enterprise AS 5.0.测试集采用 SPEC 2000/SPEC 2006 中的部分 C 程序集,测试集特征见表 2.由于指针分析的对象为 CIL 转化后的高度结构化的 C 中间语言,且为消除多维指针引入了临时的解引用变量,因而 ptranal.ml 生成的约束数要大于源程序中的约束数.

Table 2 SPEC 2000/SPEC 2006 benchmark characteristics

表 2 SPEC 2000/SPEC 2006 测试集特征

测试集	函数个数	规模(LOC)	CIL 约束数
164.gzip	89	8616	6 050
175.vpr	307	17 729	24 828
179.art	26	1 270	1 454
183.equake	27	1 513	2 893
186.crafty	62	21 151	33 437
188.ammp	176	13 483	17 610
197.parser	324	11 391	12 058
300.twolf	191	20 459	24 680
401.bzip2	100	8 293	9 764
429.mcf	24	2 685	1 824
433.milc	237	15 045	23 818
456.hmmer	536	35 992	42 831
458.sjeng	144	13 847	16 191
462.libquantum	95	4 357	5 189
464.h264ref	590	51 578	60 539
482.sphinx3	367	25 090	20 462

6.2 实验结果与分析

6.2.1 时间开销

实验数据采用 O'Caml 工具中的 stats.ml 进行统计,时间开销不计算前端分析时间和转化为 CIL 的时间,仅统计指针分析中的约束生成和约束求解两阶段,约束求解对 Narse 提出的优先权算法(记为 N-Prior)和本文提出的算法(记为 U-Prior)进行评估.测试结果见表 3.

由于 U-Prior 算法仅对涉及指向更新的约束进行评估,有效减少了约束评估次数和指向信息传播次数,避免了大量冗余的约束评估,因此在分析时间上要整体优于 N-Prior 算法.由实验结果可以看出:

(1) 与 N-Prior 算法相比,U-Prior 算法能够减少 3%~38%的分析时间,平均减少 20.47%;

(2) 当约束数较多时,U-Prior 算法与 N-Prior 算法相比性能提升相对较小;

(3) 在 N-Prior 和 U-Prior 算法中,分析时间的增加幅度要明显高于 CIL 约束数的增加幅度.

Table 3 SPEC 2000/SPEC 2006 test results of time overhead

表 3 SPEC 2000/SPEC 2006 时间开销测试结果

测试集	N-Prior time (s)	U-Prior time (s)	Time reduction (%)
164.gzip	15.236	13.673	10.26
175.vpr	105.788	83.664	20.91
179.art	5.926	4.893	17.43
183.equake	47.893	33.86	29.3
186.crafty	151.309	127.845	25.51
188.ammp	245.328	210.091	14.36
197.parser	72.862	53.121	27.09
300.twolf	67.175	61.38	8.63
401.bzip2	247.132	216.354	12.45
429.mcf	3.56	2.839	20.25
433.milc	89.049	57.193	35.77
456.hmmer	186.587	164.937	11.6
458.sjeng	76.522	62.592	18.2
462.libquantum	35.556	26.144	26.47
464.h264ref	136.209	131.182	3.69
482.sphinx3	83.391	63.326	24.06

6.2.2 存储开销

内存开销采用 O'Caml 工具中的垃圾收集机制进行统计,对指针分析阶段的每次收集的存储大小进行累加.测试结果见表 4.

Table 4 SPEC 2000/SPEC 2006 test results of storage overhead

表 4 SPEC 2000/SPEC 2006 存储开销测试结果

测试集	N-Prior (MB)	U-Prior (MB)	Storage reduction (%)
164.gzip	35 216	8 616	75.5
175.vpr	147 295	17 729	88.0
179.art	6 099	1 270	79.2
183.equake	67 753	1 513	97.8
186.crafty	120 045	21 151	82.4
188.ammp	412 273	13 483	96.7
197.parser	30 090	11 391	62.1
300.twolf	134 617	20 459	84.8
401.bzip2	447 938	82 935	81.5
429.mcf	1 224	265	78.3
433.milc	95 825	15 045	84.3
456.hmmer	256 428	35 992	86.0
458.sjeng	37 248	13 847	62.8
462.libquantum	87 205	4 357	95.0
464.h264ref	223 775	51 578	77.0
482.sphinx3	89 348	25 090	71.9

U-Prior 算法以简洁的约束依赖图替代 N-Prior 算法中较为复杂的优先权评估函数,在保证计算效率的同时,有效节省了约束评估中的存储空间,因此在空间开销上要整体优于 N-Prior 算法.实验结果可以看出:

(1) 与 N-Prior 算法相比,U-Prior 算法能够减少 62.1%~97.8%的空间开销,平均减少 81.5%;

(2) 当约束数较小时,U-Prior 算法与 N-Prior 算法相比能够节省相对较多的空间开销;

(3) 在 N-Prior 和 U-Prior 算法中,空间开销的增加幅度要明显高于 CIL 约束数的增加幅度.

7 结论与展望

指针分析是程序分析的基础,其分析的结果直接影响着编译优化的精度和程序变换的效率.本文在基于包含的指针分析算法中展开研究,从避免冗余约束评估和改进优先权评估模型两个方面入手,提出了一种基于指向更新的优先权指针分析算法.与最近的 Narse 指针分析算法相比,该算法做了如下方面的改进:

(1) 以指针的指向集更新信息确定约束评估的候选集,有效简化了迭代中约束评估的规模,避免了冗余

的约束评估;

(2) 通过构建约束依赖图,以依赖关系确定约束评估的优先权,极大地简化了复杂的优先权评估模型;

(3) 给出基于指向更新的优先权约束评估的形式化描述,从算法复杂度上分析了本文算法的有效性;

(4) 在保证分析精度的同时,基准测试集中的实验结果表明,该算法在时间开销和存储开销上具有明显的性能提升.

在下一步的工作中,我们计划对算法中输入候选集和评估优先权的思想进行扩展,应用到流敏感的指针分析中,进一步提升指针分析的精度和效率;与实际应用相结合,在开源高性能编译器 Open64 中进行实现,通过精度的提升改进并行化后程序的性能.

致谢 在此,我们向对本文研究工作提供基金支持的单位和评阅本文的审稿专家表示衷心的感谢,向为本文研究工作提供基础和研究平台的前辈致敬.

References:

[1] Pryanishnikov I, Krall A, Horspool N. Pointer alignment analysis for processors with SIMD instructions. In: Proc. of the 5th Workshop on Media and Streaming Processors. 2003. 50−57.

[2] Fink SJ, Yahav E, Dor N, Ramalingam G, Geay E. Effective typestate verification in the presence of aliasing. ACM Trans. on Software Engineering and Methodology, 2008,17(2):1−34. [doi: 10.1145/1348250.1348255]

[3] Li L, Cifuentes C, Keynes N. Practical and effective symbolic analysis for buffer overflow detection. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE 2010). ACM Press, 2010. 317−326. [doi: 10.1145/1882291.1882338]

[4] Zhu JW. Towards scalable flow and context sensitive pointer analysis. In: Proc. of the 42nd Annual Design Automation Conf. (DAC 2005). ACM Press, 2005. 831−836. [doi: 10.1145/1065579.1065798]

[5] Salcianu A, Rinard M. Pointer and escape analysis for multithreaded programs. In: Proc. of the 8th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP 2001). ACM Press, 2001. 12−23. [doi: 10.1145/379539.379553]

[6] Horwitz S. Precise flow-insensitive may-alias analysis is NP-hard. ACM Trans. on Programming Languages and Systems, 1997,19(1):1−6. [doi: 10.1145/239912.239913]

[7] Thiessen R. Expression data flow graph: Precise flow-sensitive pointer analysis for C programs [Ph.D. Thesis]. University of Alberta, 2011.

[8] Yu HT, Xue JL, Huo W, Feng XB, Zhang ZQ. Level by level making flow- and context-sensitive pointer analysis scalable for millions of lines of code. In: Proc. of the 8th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2010). ACM Press, 2010. 218−229. [doi: 10.1145/1772954.1772985]

[9] Hardekopf B, Lin C. Semi-Sparse flow-sensitive pointer analysis. In: Proc. of the 36th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2009). ACM Press, 2009. 226−238. [doi: 10.1145/1480881.1480911]

[10] Yan DC, Xu GQ, Rountev A. Demand-Driven context-sensitive alias analysis for Java. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis (ISSTA 2011). ACM Press, 2011. 155−165. [doi: 10.1145/2001420.2001440]

[11] Xin L, Ogawa M. An ahead-of-time yet context-sensitive points-to analysis for Java. Electronic Notes in Theoretical Computer Science, 2009,253(5):31−46. [doi: 10.1016/j.entcs.2009.11.013]

[12] Lattner C, Lenharth A, Adve V. Making context-sensitive points-to analysis with heap cloning practical for the real world. ACM SIGPLAN Notices, 2007,42(6):278−289. [doi: 10.1145/1273442.1250766]

[13] Pearce DJ, Kelly PHJ, Hankin C. Efficient field-sensitive pointer analysis of C. ACM Trans. on Programming Languages and Systems, 2007,30(1):106−148. [doi: 10.1145/1290520.1290524]

[14] Yu HT, Zhang ZQ. An aggressively field-sensitive unification-based pointer analysis. Chinese Journal of Computers, 2009,32(9):1722−1735 (in Chinese with English abstract).

[15] Hardekopf B, Lin C. Flow-Sensitive pointer analysis for millions of lines of code. In: Proc. of the 2011 9th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). IEEE, 2011. 289−298. [doi: 10.1109/CGO.2011.5764696]

[16] Andersen LO. Program analysis and specialization for the C programming Language [Ph.D. Thesis]. University of Cophenhagen, 1994.

[17] Steensgaard B. Points-To analysis in almost linear time. In: Proc. of the 23rd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 1996. 32–41. [doi: 10.1145/237721.237727]

[18] Hardekopf B, Lin C. Exploiting pointer and location equivalence to optimize pointer analysis. In: Proc. of the Int'l Conf. on Static Analysis. 2007. 265–280. [doi: 10.1007/978-3-540-74061-2_17]

[19] Hardekopf B, Lin C. The ant and the grasshopper: fast and accurate pointer analysis for millions of lines of code. ACM SIGPLAN Notices, 2007,42(6):290–299. [doi: 10.1145/1250734.1250767]

[20] Nasre R, Govindarajan R. Points-To analysis as a system of linear equations. In: Proc. of the 17th Int'l Conf. on Static Analysis (SAS 2010). 2011. 422–438.

[21] Nasre R, Govindarajan R. Prioritizing constraint evaluation for efficient points-to analysis. In: Proc. of the 9th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2011). 2011. 267–276. [doi: 10.1109/CGO.2011.5764694]

[22] Pereira FMQ, Berlin D. Wave propagation and deep propagation for pointer analysis. In: Proc. of the 7th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2009). 2009. 126–135. [doi: 10.1109/CGO.2009.9]

[23] Chen CM, Huo W, Yu HT, Feng XB. A survey of optimization technology of inclusion-based pointer analysis. Chinese Journal of Computers, 2011,34(7):1224–1238 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01224]

[24] Fähndrich M, Foster JS, Su ZD, Aiken A. Partial online cycle elimination in inclusion constraint graphs. In: Proc. of the ACM SIGPLAN'98 Conf. on Programming Language Design and Implementation (PLDI'98). ACM Press, 1998. 85–96. [doi: 10.1145/277652.277667]

[25] Pearce DJ, Kelly PHJ, Hankin C. Online cycle detection and difference propagation: Applications to pointer analysis. Software Quality Control, 2004,12(4):311–337. [doi: 10.1023/B:SQJO.0000039791.93071.a2]

附中文参考文献:

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

[23] 陈聪明,霍玮,于洪涛,冯晓兵.基于包含的指针分析优化技术综述.计算机学报,2011,34(7):1224–1238. [doi: 10.3724/SP.J.1016.2011.01224]

刘鹏(1981－),男,河南开封人,博士生,主要研究领域为先进编译,并行处理.
E-mail: liu_peng_2011@126.com

庞建民(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,信息安全.
E-mail: jianmin_pang@hotmail.com

赵荣彩(1957－),男,博士,教授,博士生导师,主要研究领域为先进编译,并行处理,信息安全.
E-mail: rczhao126@126.com

姚远(1972－),男,博士,副教授,主要研究领域为先进编译,并行处理.
E-mail: yaoyuan3070@126.com