程序标准化转换中的指针分析算法研究

王甜甜,苏小红,马培军
(哈尔滨工业大学计算机科学与技术学院,黑龙江哈尔滨 150001)

摘 要: 针对已有指针分析算法的程序中间表示不能充分表示程序的语法结构与语义,而导致不适合应用于程序标准化转换的问题,提出基于控制依赖树的流敏感和上下文敏感的过程间指针分析算法.将程序表示为控制依赖树,改进指向表示法用以表示指针别名,在此基础上定义数据流公式,对控制依赖树进行流敏感和上下文敏感的指针分析.实验结果表明,该算法的准确性高于 Emami 指针分析算法的准确性,并且应用于程序标准化时可显著提高代码多样化消除率.

关键词: 程序标准化; 指针分析; 指针别名; 控制依赖树
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2009) 05-1104-05

Research on Pointer Analysis Algorithm for Program Standardization

WANG Tian-tian, SU Xiao-hong, MA Pei-jun
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China)

Abstract: Existing pointer analysis algorithms usually adopt a lower-level intermediate representation which can not sufficiently represent the syntactical structure and the semantic of programs. This makes them difficult to apply to program standardization. To solve this problem, a flow-sensitive and context-sensitive pointer analysis algorithm based on control dependence tree is presented in this paper. The control dependence tree is used as the intermediate representation for the source program, and an improved point-to representation is proposed to represent alias information. Based on this, data flow equations are defined to compute the point-to information by traversing the control dependence tree. Test results show that its accuracy is higher than that of Emami's approach, and it can greatly improve the variation removal rate of the program standardization.

Key words: program standardization; pointer analysis; pointer alias; control dependence tree

1 引言

当程序中存在指针数据类型时,多个表达式可能代表同一内存地址,这时称这些表达式互为别名.如果指针的指向即别名信息不明确,则会影响程序的数据流分析,继而影响随后的相关处理.指针分析的目的是确定指针变量可能的指向,从而使数据流分析更加准确.因此,指针分析在编译器优化、程序理解等领域有着重要的研究意义和实用价值.

已有的指针分析算法大多应用于编译器中的代码优化或程序并行化[1~5],而应用于程序标准化中的指针分析算法还很少.程序标准化是根据一系列标准化规则对程序进行语义等价的转换的过程,目的是使语法表示不同但语义等价的程序有相同的系统表示,从而消除代码多样化,简化程序分析[6~9].程序标准化要求程序的中间表示既能够充分表示程序的语法结构,又能够充分表示程序语义.而已有的指针分析算法通常将程序逐步转换成越来越低级的表示形式,使得它们不适合直接应用于程序标准化中.

本文在分析影响指针分析效率的关键因素与程序标准化间的关系的基础上,提出基于控制依赖树的流敏感和上下文敏感的指针分析算法.实验结果表明,该算法能够有效处理指针算术表达式与数组元素,应用于程序标准化时可显著提高代码多样化消除率.

2 研究背景

2.1 影响指针分析效果的关键因素

针对不同应用,需要权衡指针分析的精度和时间效率,因此设计指针分析算法时应考虑如下因素.

(1) 程序的中间表示应该便于执行别名分析,并且充分利用指针分析结果.

(2) 别名表示方式应该全面、直观地表示指针表达式与其指向目标间的关系.

(3) 如果考虑控制流信息,即程序语句的执行顺

收稿日期:2008-05-07;修回日期:2008-09-15
基金项目: 国家自然科学基金(No. 60673035)

序,则算法流敏感. 否则,非流敏感. 前者分析精度较后者高,但时间效率较低.

(4) 如果根据同一函数不同调用点指针指向模式的不同而产生不同的分析结果,则算法上下文敏感,否则,上下文不敏感. 前者分析精度较后者高,但时间效率较低.

(5) 区分集合的各个元素比不区分的算法分析精度高,但时间效率较低.

(6) 区分必然别名与可能别名比不区分的算法分析精度高,但时间效率较低.

2.2　程序标准化与指针分析

程序标准化主要包括:拆分复合语句、表达式标准化、控制结构标准化、函数调用标准化、消除冗余代码、变量重命名、语句重排序、指针标准化(包括必然别名替换,指针与数组表示的标准化等). 其中前三种标准化称为基本标准化,在执行转换时不需要数据流与指针别名信息. 后五种标准化称为高级标准化,需要在转换之前进行指针与数据流分析,以确保程序语义不变.

3　程序标准化转换中的指针分析模型

程序标准化中的指针分析模型如图 1 所示.

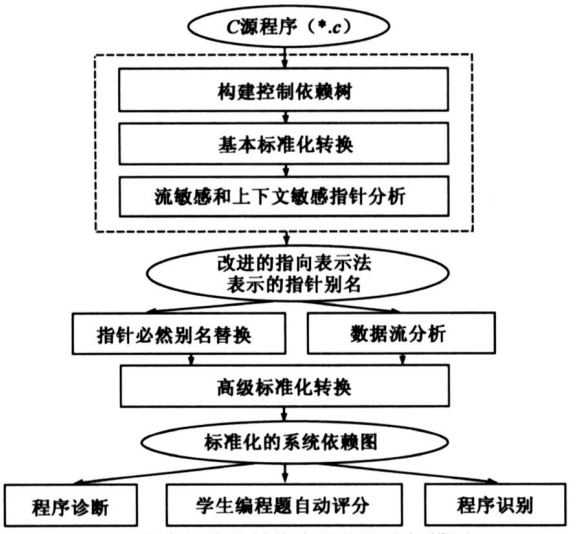

图1　程序标准化转换中的指针分析模型

首先解析源程序. 为了使程序的中间表示既便于程序标准化,又简化指针分析,本文提出控制依赖树表示,在表示语法结构的同时充分表达语义.

然后对控制依赖树进行基本标准化,以消除部分代码多样化,从而简化随后的指针分析.

接下来,遍历控制依赖树,计算各个节点的指针别名. 程序标准化需要确保程序的语义不变,对指针分析的准确性要求较高,因此本文采用了流敏感和上下文敏感的指针分析方法.

然后,就可以利用指针别名执行必然别名替换和数据流分析,进一步执行高级标准化. 最后生成的标准化系统依赖图可直接应用于程序识别等领域.

4　用控制依赖树表达程序的语义和语法结构

本文将系统依赖图[10]的控制依赖子图表示为控制依赖树. 主要改进如下:

(1) 系统依赖图将表达式表示为节点中的 token 串. 本文将其表示为抽象语法树,连接到语句节点上,使得控制依赖树不但可以表示控制流,还可以充分表示语法信息,便于执行指针分析和程序转换.

(2) 系统依赖图将数组元素和指向数组的指针表示为不同的 token. 本文用统一的抽象语法树表示它们,如图 2 所示,可以消除指针和数组表示的多样化,而且便于分析指针算术表达式的别名.

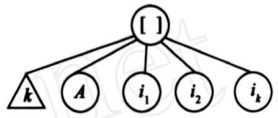

图2　$A[i_1][i_2]\ldots[i_k]$和$*(*\ldots(*(*(A+i_1)+i_2)\ldots)+i_k)$的统一语法树表示

5　指向表示法的改进

本文在 Emami 指向表示[2]的基础上提出改进的指向表示:采用三元组 (p, t, r) 表示指向信息,其中 p 表示指针变量,t 表示该指针变量所指向的表达式,r 表示是必然指向(D)还是可能指向(P). 将数组元素用数组名及下标表示,并用统一的形式表示数组元素与指向数组的指针. 改进的指向表示与 Emami[2]、黄波[3]的指向表示的对比如表 1 所示.

表1　几种指向表示法的比较

程序语句	Emami 指向表示[2]	黄波指向表示[3]	本文指向表示

int a, * p; p= &a;
int a[100], b, * q;
if(a[0] > b)) q= &b;
else q= a;
	{(p, a, D)}
	{(q, b, P),
	(q, a. head, P)}
		{(p, a, D, ϕ, ϕ)}
		{(q, b, P, ϕ, ϕ),
		(q, a, P, ϕ, TR_0)}
		其中 TR_0 = {(0,2,1)}
		{(p, q, D, PR_0, TR_0)}
		其中 PR_0 = {(0,4,3)},
		TR_0 = {(2 i,2 i+1)}
			{(p, a, D)}
			{(q, b, P),
			(q, a[0], P)}

int * p[10], q[N]; p[0] = q+ i;	{(p. head, q. head, P), (p. head, q. tail, P)}		{(p[0], q[i], D)}
int * * pq[N][N]; p= q[i] + j;	{(p, q. head, P}, (p, q. tail, P)	{(p, q, P, ϕ, TR_0)}其中 TR_0 = {((i * N + j+1) *2,2, (i * N + j) *2-1)}	{(p, q[i][j], D)}

例如,对 int * * p, q[N][N]; p = q[i] + j;分析出指向集合{(p, q[i][j], D)},执行必然别名替换时可直接用 q[i][j]替换 * p,将 * p = 5;转换为 q[i][j] = 5. 而其它两种指向表示法都不适合于直接执行该转换. 可见,本文改进的指向表示不但可以准确表示数组元素和指针算术表达式,同时,计算和表示都简单直观,程序标准化中可以直接利用指针别名分析结果.

6　基于控制依赖树的指针分析算法

6.1　指针分析的数据流公式

本文定义如表2所示的数据流公式,用于计算控制依赖树中每个指针赋值语句节点生成(GEN)、改变(CHANGE)与注销(KILL)的指向集合.

表2中符号解释如下:

(1) $\underbrace{*\cdots*}_{n}x=\underbrace{*\cdots*}_{m}y$ 是指针赋值语句,其中,x 和 y 表示指针表达式,n 和 m 表示解引用的级数,满足 $n>=m$, $m>=-1$. $m=-1$ 表示对 y 取地址.

(2) $X_n(S)$、$Y_{m+1}(S)$ 分别表示对于当前指向集合 S,对 x 进行 n 次解引用,对 y 进行 $m+1$ 次解引用,所到达的位置集合.

(3) $Definite_S(v_1, v_2)$ 为真,仅当在 S 中 v_1 到 v_2 的所有指向均为 D.

(4) $GEN_N(S)$ 由 $X_n(S)$ 与 $Y_{m+1}(S)$ 合并生成.生成的指向关系 (p,t,r), $r=D$ 仅当在 S 中 (x,p) 与 (y,t) 的指向均为 D,否则 $r=P$.

(5) $CHANGE_N(S)$ 将 S 中某些必然指向修改为可能指向. $KILL_N(S)$ 注销 S 中的某些指向关系. OUT_N 是节点 N 的最终指向集合.

表2 赋值语句节点 $\underbrace{*\cdots*}_{n}x=\underbrace{*\cdots*}_{m}y$ 的数据流公式

集合	数据流公式	说明
IN_N	$IN_N=OUT_M$ 其中,M 是 N 在控制依赖树中的前驱节点	流入节点 N 的指向集合
S	$S=IN_N$	节点 N 当前的指向集合
GEN_N	$GEN_N(S)=\{(p,t,r):p\in X_n(S)\wedge t\in Y_{m+1}(S)\wedge r=\begin{cases}D & \text{if } Definite_S(x,p)\wedge Definite_S(y,t)\\ P & \text{otherwise}\end{cases}\}$	节点 N 生成的指向集合
$CHANGE_N$	$CHANGE_N(S)=\{(p,t,r):p\in X_n(s)\wedge(p,t,r)\in S\wedge r=\begin{cases}D & \text{if } Definite_S(x,p)\\ P & \text{otherwise}\end{cases}\}$	节点 N 改变的指向集合
$KILL_N$	$KILL_N(S)=\{(p,t,r):p\in X_n(S)\wedge(p,t,r)\in S\wedge Definite_S(x,p)\}$	节点 N 注销的指向集合
OUT_N	$OUT_N=(((IN_N-\{(p,t,D):(p,t,P)\in CHANGE_N\})\cup CHANGE_N)-KILL_N)\cup GEN_N$	流出节点 N 的指向集合

6.2 指针赋值语句节点的指向分析算法

对于赋值语句节点,首先调用 lvalue 算法(见图3),后根序遍历左部表达式语法树,求 Gl, Cl, Kl 集合.然后调用 rvalue 算法(见图4),后根序遍历右部表达式语法树,求 Gr.最后利用这些集合,根据表2中的数据流公式更新该节点的指向集合 $GEN=\{(p,t,r):(x,p,r1)\in Gl,(y,t,r2)\in Gr, r=r1\wedge r2\}$.

```
算法:lvalue
输入:赋值语句节点 N 的表达式左部语法树 e, N 的当前指向集合 S
输出: Gl, Cl, Kl
  switch e
    case *el:
      (G, C, K) = lvalue(El, S)
        return(dereference(G, S, Gl), dereference(C, S, Cl), derference
(K, S, Kl))
    case v
        Gl = {(~, v, D)}
    if 至少存在一个三元组(v, x, l) ∈S then
          Cl = Kl = {(v, x, l):(v, x, l) ∈S}
    else
          Cl = Kl = {(v, ~, D)}
    endif
    return(Gl, Cl, Kl)
  endswitch
```

图3 计算指针赋值语句左部的指向集合的算法

算法中的符号解释如下:

(1) Gl、Gr、Cl 以及 Kl 分别对应于数据流公式中的 $X_n(S)$、$Y_{m+1}(S)$、$CHANGE$ 与 $KILL$ 集合.

(2) 指向关系 $r1\wedge r2$ 为 D,当且仅当 $r1$ 与 $r2$ 均为 D,否则 $r1\wedge r2$ 为 P.

(3) deference$(S1, S, S2)=\{(p,t,r1\wedge r):(p,t,r1\wedge r)\in S2\wedge(x,p,r1)\in S1\wedge(p,t,r)\in S\}$,根据当前指向集合 S,对 $S1$ 中的元素进行解引用操作,生成 $S2$ 集合.

(4) address$(S1, S2)=\{(\sim,t,r):(\sim,t,r)\in S2\wedge(p,t,r)\in S1\}$,对指向集合 $S1$ 中的元素进行取地址操作,生成 $S2$ 集合.其中,~表示任意的指针变量.

```
算法:rvalue
输入:赋值语句节点 N 的表达式右部语法树 e, N 的当前指向集合 S
输出: Gr
    switch e
      case &el
            G = rvalue(el, S)
            return(address(G, Gr))
    case *el:
            G = r value (el, S)
            return(dereference(G, S, Gr))
    case v:
          Gr = {(v, ~, D)}
          return(Gr)
endswitch
```

图4 计算指针赋值语句右部的指向集合的算法

6.3 流敏感和上下文敏感的指针分析

后根序遍历控制依赖树,首先,令流入每个节点的指向集合等于其前驱节点流出的指向集合.控制依赖树中语句节点从左到右顺序,与它们在源程序中的先后顺序一致,确保了指针分析是流敏感的.

然后,根据节点类型更新指向集合:对赋值语句调用指向分析算法;对选择结构递归分析每个分支,指向集合是各个分支的指向集合的并集;对循环结构迭代分析循环体,直到指向集合不变为止;对函数调用节点进行上下文敏感的分析. 对递归函数调用,采用迭代方式,依据实参上下文的变化,更新被调函数各节点处的别名. 对非递归函数调用,则采用图5所示的分析方法.

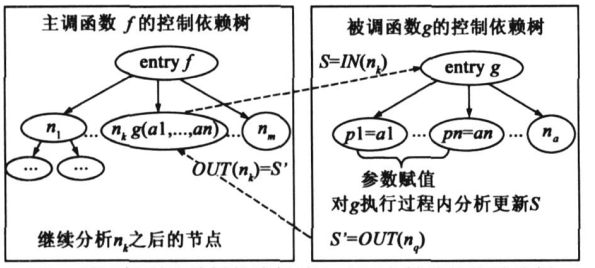

图5　基于控制依赖树的流敏感与上下文敏感的指针分析

7　实验与分析

本文的指针分析算法已经在程序标准化工具 Normalizer 中实现,并且应用于C语言编程题自动评分系统[9]和程序识别系统中.

7.1　准确性分析

我们用 Emami 使用的开源测试程序[2],并用如下的 *avg* 衡量指针分析的准确性.

$$avg = \text{指针指向的地址总数} / \text{间接引用的总数} \quad (1)$$

实验结果如表3所示. 表3中前四列第一个值表示 $*x$ 形式的解引用(普通指针解引用)的个数,第二个值表示 $x[i][j]$ 形式的间接引用(其中 x 为指向数组的指针)的个数.

由表3可得到如下结论:(1) Emami 算法与本文算法的 *avg* 平均值分别为 1.15 和 1.07,这表明本文算法具有较高的准确性. 对比前4列数据,可以看出本文算法能够更准确地分析指向数组的指针;(2) Emami 算法共有 27.9 %的间接引用的解引用指针确定指向一个地址,使用这些确定信息,20.2 %的间接引用可以被直接引用替换. 本文算法的这两个值分别为 29.4 % 和 20.4 %. 可见本文算法准确性更高,更有利于执行必然别名替换等标准化操作.

表3　指针分析准确性比较

测试程序	确定指向一个地址的间接引用个数		可能指向一个或多个地址的间接引用个数		程序中间接引用的总数		可以被直接引用替换的间接引用个数		程序中指针指向的地址总数		解引用指针指向的地址的平均数 *avg*	
	Emami 算法	本文算法	Emami 算法	本文算法	Emami 算法	本文算法	Emami 算法	本文算法	Emami 算法	本文算法	Emami 算法	本文算法
dry	2,11	2,11	45,0	45,0	58	58	9	10	66	66	1.14	1.14
clinpack	7,98	7,111	0,45	0,32	150	150	101	115	197	184	1.31	1.23
toplev	5,0	7,0	112,0	110,0	117	117	5	7	171	123	1.46	1.05
compress	0,0	0,0	40,10	40,10	50	50	0	0	50	50	1.00	1.00
stanford	6,61	6,62	74,2	74,0	143	143	51	52	145	144	1.014	1.007
sim	0,0	0,0	122,231	122,231	353	353	0	0	353	353	1.00	1.00

7.2　程序标准化中指针分析效果

我们用 Normalizer 对8个编程题目(每题选取100个含有代码多样化的程序)进行两次标准化实验,其中第一次实验不应用指针分析,第二次实验应用本文的指针分析算法. 实验结果如表4所示.

表4　指针分析在程序标准化中的效果

编程题目	平均代码行数	应用指针分析前的 VRR	应用指针分析后的 VRR	VRR 提高值
1 指针实现 strcmp	21	63.5 %	96.7 %	33.2 %
2 查找数组最大值和最小值	32	40.2 %	95.8 %	55.6 %
3 二维矩阵转置	48	52.4 %	94.2 %	41.8 %
4 字符串数组中插入字符串	75	37.8 %	94.6 %	56.8 %
5 clinpack	1386	31.8 %	77.7 %	45.9 %
6 sim	1556	30.7 %	70.9 %	40.2 %
7 toplev	2073	32.1 %	74.8 %	42.7 %
8 flex2.5.4	22720	30.2 %	75.3 %	45.1 %

我们用如下代码多样化消除率(Variation Removal Rate, VRR)[8],衡量程序标准化效果. 其中, x 和 y 分别表示执行标准化之前和之后同一题目不同的程序表示形式个数.

$$VRR = ((x-1)-(y-1))/(x-1) \times 100\% \quad (2)$$

由表4可见,应用本文的指针分析算法后, *VRR* 平均提高了 45.1 %. 这是因为指针分析后可执行更多的标准化转换,如别名替换,以及与数据流相关的高级标准化转换等.

8　结论

本文提出一种适用于程序标准化的指针分析算法. 将指针分析与程序标准化转换过程有机地结合在一起,既利用程序标准化简化指针分析,又将指针分析结果直接应用于程序标准化中,提高程序标准化的代码多样化消除率.

参考文献:

[1] David J Pearce, Paul H J Kelly, PChris Hankin. Efficient field-sensitive pointer analysis of C[J]. ACM Transactions on Pro-

gramming Languages and Systems ,2007 ,30(1) :4:1 - 4:42.

[2] Maryam Emami , et al. Context-sensitive interprocedural points-to analysis in the presence of function pointers[A]. Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation[C]. Florida: ACM Press , 1994. 242 - 256.

[3] 黄波,臧斌宇,俞一峻,朱传琪. 指针数组的过程内别名分析[J]. 软件学报,1999,10(6) :600 - 607.
Huang Bo, et al. Intraprocedural alias analysis for pointer array [J]. Journal of Software ,1999 ,10(6) :600 - 607. (in Chinese)

[4] 黄波,臧斌宇,俞一峻,朱传琪. 上下文敏感的过程间指针分析[J]. 计算机学报,2000,23(5) :477 - 485.
Huang Bo, et al. Context sensitive interprocedural pointer analysis[J]. Chinese Journal of Computers ,2000 ,23(5) :477 - 485. (in Chinese)

[5] 戚晓芳,徐宝文,周晓宇. 一种基于程序可达图的并发程序依赖性分析方法[J]. 电子学报,2007,35(2) :287 - 291.
Qi Xiao-fang, et al. An approach to analyzing dependence of concurrent programs based on program reachability graphs[J]. Acta Electronica Sinica ,2007 ,35(2) :287 - 291. (in Chinese)

[6] Metzger R, Wen Z. Automatic Algorithm Recognition: A New Approach to Program Optimization[M]. London: MIT Press , 2000. 87 - 116.

[7] S Xu, Y S Chee. Transformation-based diagnosis of student programs for programming tutoring systems[J]. IEEE Transactions on Software Engineering ,2003 ,29 (4) :360 - 384.

[8] Hattori N, Ishii N. A method to remove variations in source codes[J]. Information and Software Technology ,1996 ,38(1) : 25 - 36.

[9] Wang T. T, et al. Semantic similarity-based grading of student programs[J]. Information and Software Technology ,2007(2) , 49:99 - 107.

[10] Livadas P. E, Johnson T. An optimal algorithm for the construction of the system dependence graph[J]. Information Sciences ,2000 ,125(1 - 4) :99 - 131.

作者简介:

王甜甜 女,1980 年生,哈尔滨工业大学计算机科学与技术学院博士研究生,主要研究方向为程序分析,软件缺陷检测,软件工程. E-mail: wttsweet @shu. com

苏小红 女,1996 年生,教授,博士生导师,中国计算机学会高级会员. 主要研究方向:软件缺陷检测,图像处理与识别,信息融合,智能计算等.