面向对象程序设计语言的绑定时间分析技术*

廖湖声+, 童兆丰, 王 众

(北京工业大学 计算机学院,北京 100022)

A Technology of Binding Time Analysis for Object-Oriented Programming Languages

LIAO Hu-Sheng+, TONG Zhao-Feng, WANG Zhong

(College of Computer Science, Beijing Polytechnic University, Beijing 100022, China)

+ Corresponding author: Phn: 86-10-67392987; 86-10-67391745, Fax: 86-10-67392871, E-mail: liaohs@bjpu.edu.cn

http://www.bjpu.edu.cn

Received 2002-01-21; Accepted 2002-05-17

Liao HS, Tong ZF, Wang Z. A technology of binding time analysis for object-oriented programming languages. *Journal of Software*, 2003,14(3):415~421.

Abstract: A technology of binding time analysis for implementing partial evaluation of object-oriented programming languages is proposed in this paper. By tracing context-sensitivity of reference variables and pointer variables, the new approach can deal with elements of partly static data structure, such as attributes of each object and elements of each array. The new approach uses a two-level BTA environment to hold BTA states for static variables and local variables. The objects created at different program points are represented by a kind of specific handle. The BTA state of a reference variable is represented by a set of such handles. The algorithm of a forward analysis and backward analysis are presented. They are used to annotate source program, with BTA environment to trace binding time of various kind of variables including identifier of array, object and reference. The binding time analysis has been implemented for Java. It is able to analyse most single thread Java programs and support partial evaluation for Java with higher performance.

Key words: binding time analysis; partial evaluation; object-oriented programming language; Java language

摘 要: 为了实现面向对象语言的部分求值,提出了一种绑定时间分析技术.该技术通过针对引用类型变量和指针变量的上下文敏感分析,能够比较精确地分析面向对象语言中诸如对象元素、数组元素等复杂数据结构元素的绑定时间,进而扩大了部分求值的作用范围.这种方法采用两层 BTA 环境来保存静态变量和局部变量的 BTA 状态,设置一种专用句柄来表示不同程序点创建的对象,进而采用这种句柄的集合表示引用类型变量的 BTA 状态.在为面向对象语言程序标注绑定时间信息的过程中,采用一个正向分析和一个反向分析过程,借助于

* Supported by the National Natural Science Foundation of China under Grant No.60173013 (国家自然科学基金); the Natural Science Foundation of Beijing City of China under Grant No.4982002 (北京市自然科学基金); the Assisting Project of Ministry of Education of China for Backbone Teachers of University and College (国家教育部高等学校骨干教师资助计划)

第一作者简介: **廖湖声**(1954－),男,广东大埔人,教授,博士生导师,主要研究领域为软件自动化方法,编译技术,空间数据库技术.

BTA 环境来跟踪和设定各种变量、对象和引用变量的绑定时间.该技术已经用于实现 Java 程序的绑定时间分析,能够有效地分析大多数单线程的 Java 程序,为实现高性能 Java 程序部分求值提供了必要的手段.

关键词: 绑定时间分析;部分求值;面向对象程序设计语言;Java 语言

中图法分类号: TP311 **文献标识码**: A

绑定时间分析(binding time analysis)作为一种程序静态分析技术,经常用于程序设计语言部分求值(partial evaluation)的有效实现.在函数式语言和过程型语言的部分求值技术中,精确的绑定时间分析已经成为提高程序性能优化的主要手段[1~3].近年来,人们已经开始将有效地应用于C语言部分求值的BTA分析技术移植到Java语言的部分求值系统中,试图为提高 Java 程序的性能提供一种新的手段.

为了更精确地进行绑定时间分析,本文提出一种适用于面向对象语言的绑定时间分析技术.与传统的过程型语言绑定时间分析相比,它不仅能够分析一般变量绑定时间的上下文敏感性(context sensitivity),而且能够分析数组变量、对象引用变量以及对象成员变量、数组元素等各种程序变量,从而将部分求值处理深入到复杂数据结构的内部,有效地扩大了部分求值的作用范围.这种技术已经应用于 Java 语言的一个子集,实现了除多线程、异常处理功能之外的所有 Java 语言结构的绑定时间分析.

1 绑定时间分析与上下文敏感性

在程序设计语言的部分求值实现技术中,绑定时间分析是一种程序预处理技术,根据部分求值的要求,经过程序静态分析,确认并标明程序中哪些计算在部分求值阶段完成(静态部分),哪些计算(动态部分)保留在作为部分求值结果的滞留程序(residual program)中.例如,表 1 中如果部分求值时 x 为已知参数,y 为未知参数,则表中左边的程序经过绑定时间分析,将获得表中右边所示的具有标注的程序.

Table 1 Case on binding time analysis

表 1 绑定时间分析的案例

1.	void fun (int *x*, int *y*)	void fun (int x^S,int y^D)
2.	{	{
3.	MyObj *o*;	MyObj o^{SD};
4.	*o*=new MyObj( );	$o^{SD}=^{SD}$new^{SD} MyObj( );
5.	*o.a*=*x*;	$o.a^{SD}=^{SD}x^S$;
6.	while (*x*>10) {	while^S ($x^S>^S10^S$) {
7.	*o.a*=*o.a*+*x*;	$o.a^D=^Do.a^D+^Dx^S$;
8.	*o.a*=*o.a***y*;	$o.a^D=^Do.a^D*^Dy^D$;
9.	*x*=*x*−2;	$x^S=^Sx^S-^S2^S$;
10.	}	}
11.	}	}

程序中标注为 S 的部分在部分求值中完成,标注为 D 的部分代表剩余的计算保留在滞留程序中,而标注为 SD 的部分对于两个阶段的计算都是必要的.因此,经过部分求值,程序中的循环语句将变换成一组赋值语句.

在 BTA 分析中,对于在程序不同位置出现的变量,需要参照程序执行中的变量引用和赋值顺序,分别进行标注.例如,变量 *a* 在第 4~6 行,分别被标注为 SD,S 和 D.因此,BTA 分析中必须跟踪变量的上下文敏感性,才有可能精确地划定部分求值的范围.

对于 C 语言等过程型语言,指针和结构等复杂数据结构的存在为 BTA 分析带来困难,传统的处理方法[4,5]放弃了结构分量的上下文敏感性分析,对于属于相同结构的所有变量的每个结构分量,在 BTA 分析中设置惟一的 BTA 标注.目前,人们也将同样的方法应用于面向对象语言,对于某个类的所有对象的每个成员变量,在 BTA 分析中也设置惟一的 BTA 标注.本文为解决这个问题提出了新的方法.

2 面向对象语言的上下文敏感性 BTA 分析

为了实现对象成员变量的上下文敏感性 BTA 分析,本文根据面向对象语言中对象的使用方法,结合 BTA 分析的要求,作出如下分析:

(1) 为了实现精确的BTA分析,对于相同类的不同对象变量应该分别设置BTA标注.为此,需要对程序中不同位置出现的所有对象变量进行上下文敏感性的跟踪.这里包括了静态说明的所有对象变量和动态生成的所有对象;后者可以被看做是出现在程序不同位置的全局变量.

(2) 鉴于面向对象语言通常提供多种对象引用方法,如引用变量和指针变量等,BTA 分析在实现引用变量和指针变量的上下文敏感性分析的同时,必须能够通过引用变量或指针变量访问到相应的对象.由于这些变量在程序的不同分支可能指向不同的对象,BTA 分析中需要在不同程序位置跟踪引用变量或指针变量可能代表的所有对象变量,为对象成员变量提供访问路径.

(3) 对于程序中的数组变量(包括对象数组),可以看做是对象变量的特殊情况.BTA分析中可以按照对象变量的处理方法,跟踪数组变量的上下文敏感性和引用关系.同时,鉴于数组元素的访问往往在数组下标未知的条件下出现,BTA 分析中不仅需要跟踪指定下标的所有数组元素,而且需要跟踪数组元素在下标未知时可能参与的各种运算及其涉及的上下文敏感性问题.

在上述分析的基础上,对于面向对象语言的上下文敏感性BTA分析,本文的解决方案如下:

(1) 为了跟踪各种变量绑定时间的上下文敏感性,为每个变量设置一个BTA状态;对于一般变量,当BTA分析过程中处理到当前位置时,变量的BTA标志表示了它的BTA状态.对于引用变量(包括指针变量、数组变量),变量可能指向的所有对象变量的集合表示了其BTA状态.

(2) 在BTA分析中设置一个两层的BTA环境,下层环境保存程序中所有静态变量(包括类静态成员)的BTA状态以及所有动态生成对象(包括数组)的 BTA 状态;后者采用生成表达式的位置信息来表示.上层环境中保存当前处理的成员函数中所有参数、所有局部变量以及当前类对象的BTA状态.

(3) 将 BTA 分析过程分为两遍扫描的过程.第 1 遍扫描的目的是根据成员函数中所有输入参数的 BTA 状态,确认所有运算符和控制语句的 BTA 状态,以及被引用的所有变量的 BTA 状态.具体步骤是按照程序文本的排列顺序,从前到后逐步分析每条语句,标注每个变量、运算符和控制语句.在遇到赋值表达式时,更新 BTA 环境中变量的 BTA 状态.

(4) 第 2 遍扫描的目的是确认成员函数中所有变量说明、所有被赋值的变量和赋值符号以及对象动态生成表达式的 BTA 标注.鉴于这些 BTA 标注取决于该变量将来的引用,因此分析过程是从后向前进行,逐步地分析语句和表达式,在 BTA 环境中记录变量引用的 BTA 状态,并根据被赋值变量的引用信息,设置被赋值变量和赋值号的 BTA 标注以及变量说明的 BTA 标注.同时,将引用类型变量的 BTA 状态变为 BTA 标注.

表 2 说明了 BTA 分析两遍扫描的结果.表的左边是第 1 遍扫描的结果,控制语句和表达式右侧的变量被标注为 BTA 状态,说明循环计算将在部分求值阶段完成;表的右边是第 2 遍扫描的结果,将 BTA 状态改为 BTA 标注,并补充标明了滞留程序中需要保留的变量赋值与变量说明.例如,在第 1 遍扫描结果中,对象引用变量 *o* 的上标{4}是该变量的 BTA 状态,表示该变量指向第 4 行生成的对象.在第 2 遍扫描之后,该标注被改为标注 *SD*.

Table 2 Results of two parse phases in binding time analysis

表 2 BTA 分析两遍扫描的结果

1.	void fun (int x^S,int y^D)	void fun (int x^S,int y^D)
2.	{	{
3.	MyObj o;	MyObj o^{SD};
4.	$o^{\{4\}}=^{\{4\}}$new MyObj( );	$o^{SD}=^{SD}$new^{SD} MyObj( );
5.	$o.a=^S x^S$;	$o.a^{SD}=^{SD}x^S$;
6.	while^S ($x^S>^S 3^S$) {	while^S ($x^S>^S 10^S$) {
7.	$o.a=^D o.a^D+^D x^S$;	$o.a^D=^D o.a^D+^D x^S$;
8.	$o.a=^D o.a^D*^D y^D$;	$o.a^D=^D o.a^D*^D y^D$;
9.	$x=^S x^S-^S 2^S$;	$x^S=^S x^S-^S 2^S$;
10.	}	}
11.	}	}

这种方法带来的一个新问题是如何计算BTA状态的最小上界.在第1遍扫描中,对于一般变量的BTA状态等价于它们的 BTA 标注,计算方法相同,$S \cup D \rightarrow D$;对于引用类型变量或指针变量,可以通过合并它们的上标集合来完成最小上界的计算.在最小上界的计算中,必须保留每个对象BTA状态的位置信息,从而保证对象成员变量被赋值时的BTA信息传递到所有相关的对象.

在 BTA 分析的第 2 遍扫描中,则需要将已经确定的 BTA 状态改写为 BTA 标注.对于在计算的两个阶段都需要的变量和计算,应标注为 *SD* 状态.对于引用变量或指针变量,如果其指向的所有对象及其成员变量的 BTA 状态不都是静态或都是动态,也应该标注为 *SD*;这说明对象中某些部分将用于部分求值,而其他部分将出现在滞留程序中,是一种部分静态结构.

3 实现算法

本节以表 3 所示的 Java 语言子集为例,介绍绑定时间分析的主要实现算法.该子集包括对象调用、对象生成、一维数组、运算表示式、条件语句、循环语句和说明语句等面向对象语言的基本元素.

BTA 分析算法由 4 个部分组成,包括表达式分析算法、对象方法的分析算法、语句正向分析算法(第 1 遍扫描)和语句反向分析算法(第 2 遍扫描).表 4 说明了表达式的分析算法;在给定的 BTA 环境 v 下,计算每个表达式的 BTA 状态,用于表达式中每个变量、常数和运算符的标注.BTA 状态 **State** 的取值有两种情况:用于一般变量的 BTA 状态 Bta 以及用于引用类型变量的句柄集合.由于 Java 程序中对象均采用动态生成方法,采用了程序位置信息作为这种对象的 BTA 信息句柄,从而建立了引用变量与对象之间的关联.

Table 3 Subset of Java language

表 3 Java 语言的子集

Syntax domain:	
stmt∈**Stmt** statement	
expr∈**Expr** expression	
Id∈**Iden** identifier	
const∈**Const** constant	
Abstract syntax:	
stmt➔Id=expr;	
stmt➔Id_1=New Id_2[expr];	array (including object array)
stmt➔Id_1=New Id_2($expr_1$,…,$expr_n$);	object construction
stmt➔$Id_1.Id_2$=expr;	object's member
stmt➔Id[$expr_1$]=$expr_2$;	
stmt➔Id($expr_1$,…,$expr_n$);	method invocation for this class
stmt➔$Id_1.Id_2$($expr_1$,…,$expr_n$);	method invocation
stmt➔If (expr) $stmt_1$ Else $stmt_2$	condition statement
stmt➔While (expr) stmt	loop statement
stmt➔{$stmt_1$…$stmt_n$}	compound statement
stmt➔Type Id;	declaration statement
expr➔const	constant
expr➔Id	
expr➔$Id_1.Id_2$	object's member
expr➔Id [expr]	element of array
expr➔UnOp expr	unary operation (including cast)
expr➔$expr_1$ BinOp $expr_2$	binary operation
expr➔(expr)	

在进行 BTA 分析之前,需要确定每个方法的求值模式;也就是确定每个方法的输入参数和输出参数的 BTA 状态.对于面向对象语言,每个方法的输入参数包括方法参数变量以及方法中引用的类成员和静态成员变量,输出参数包括被赋值的类成员和静态成员变量.

在这种对象方法的 BTA 分析中,将模拟程序执行的过程,跟踪每个方法调用的求值模式,记录下每个方法可能采用的各种部分求值模式,最终为每个方法选定惟一的求值模式.同时,为每个对象和数组生成 BTA 信息句柄及其元素的 BTA 状态.这种对象方法的 BTA 分析算法可以利用函数式语言的 BTA 分析技术,限于篇幅,本文不再给予详细介绍.

表 5 说明了第 1 遍扫描中的语句正向分析算法.基本过程是按照语句排列顺序分析方法定义,维护 BTA 环境中变量的 BTA 状态,标注控制语句的 BTA 状态,并且为表达式的 BTA 标注提供 BTA 环境.

Table 4 Annotation for an expression in binding time analysis

表 4 BTA 分析中表达式的 BTA 标注求值

Representation for value:		
State=Bta+Iden*		BTA state
bt∈**Bta**		BTA annotation
bt::=S|D|SD		
Key➔Iden+Iden.Iden+Iden[Expr]		
v∈**Env=Key➔State**		BTA environment
BTA evaluation rules for expression:		
EB [**Expr**] **Env➔Bta**		
EB [const] v	=S	
EB [Id] v	=v [Id]	variable
EB [$Id_1.Id_2$] v	=v [$Id_1.Id_2$]	object's member
EB [Id [expr]] v	=v [Id[expr]]	element of array
EB [UnOp expr] v	=EB [expr] v	
EB [$expr_1$ BinOp $expr_2$] v	=EB [$expr_1$] v∪EB[$expr_2$] v	
EB [(expr)] v	=EB [expr] v	
Comment:		
Initialization:		
The initial value of v is a two-level BTA environment for the current method		
Function:		
$bt_1 \cup bt_2$ evaluation of the least-upper boundary of the two BTA states		

Table 5 Forward analysis algorithm for a statement in binding time analysis

表 5 BTA 分析的语句正向分析算法

Representation of value:	
in∈**Env**	Premise BTA environment for analyzing the current statement
out∈**Env**	Post BTA environment for analyzing the current statement
Forward analysis rules for statement:	
SFB [**Stmt**] **Env➔Env**	
SFB [Id=expr;] in=in++⟨id➔bt⟩	
	where bt=EB[expr]in
SFB [Id_1=New Id_2 [expr];] in=in++⟨id_1➔bt⟩	
	where bt=in[New.where]
SFB [Id_1=New Id_2 (e_1,…,e_n);] in＝out.getOutEnv(Id_2,Id_2)	
	where out=in++⟨id_1➔in[New.where]⟩
SFB [$Id_1.Id_2$=expr;] in＝in++⟨$Id_1.Id_2$➔bt⟩	
	where bt=EB [expr] in
SFB [Id [$expr_1$]=$expr_2$;] in＝in++⟨Id[expr]➔bt⟩	
	where bt=EB[$expr_1$] in∪EB [$expr_2$] in
SFB [Id(e_1,…,e_n);] in＝in.getOutEnv(this,Id)	
SFB [$Id_1.Id_2$(e_1,…,e_n);] in＝in.getOutEnv(Id_1,Id_2)	
SFB [If (expr) $stmt_1$ Else $stmt_2$] in＝	
	if bt==S then out else out.setDyn($stmt_1$, $stmt_2$)
	where bt=EB [expr] in
	out=SFB[$stmt_1$] in∪SFB[$stmt_2$] in
SFB [While (expr) stmt] in＝	
	if bt==S then out_n else out_n setDyn(stmt)
	where bt=EB[expr] out_n;
	in_1=in
	out_i=SFB [stmt] in_i for i=1,…,n and in_n=out_n
SFB [{s_1…s_n}] in=out_n	
	where out_i=SFB [s_i] out_{i-1} for i=1,…,n
	out_0=in
SFB [Type Id;] in＝in++⟨Id➔null⟩	
Comment	
Initialization: the initial value of in is a two-level BTA environment for the current method with each parameter's BTA state, which is set in BTA analysis for method.	
Auxiliary function: in++⟨id➔bt⟩ to update BTA environment by assigning id's BTA state with bt. env.getOutEnv(class,method) to get the post BTA environment for the method class.method.env.SetDyn (stmt) to assign dynamic BTA states of the variables those are assigned in the statement stmt.$bt_1 \cup bt_2$ evaluation of the least-upper boundary of the two BTA states.$in_1 \cup in_2$ union of the two BTA states.	
Attribute: New.where to representation the position of the statement in program's text.	

在语句正向分析算法中,各种赋值语句的处理首先要求计算赋值号右侧表达式的 BTA 状态,然后更新 BTA 环境中赋值号左侧的变量、数组元素或对象成员的 BTA 状态;对于对象动态生成语句,在对象方法 BTA 分析阶段,在 BTA 底层环境中已经设置了全局变量句柄,保存了相应的对象 BTA 状态以及对象成员和数组元素的 BTA 状态.在右侧表达式的分析中,可以从 BTA 环境中直接获得它们的 BTA 状态.

对象方法调用语句的处理已经在方法 BTA 分析中完成,这里需要利用方法输出参数的 BTA 状态来更新当前 BTA 环境.条件语句的处理首先判断条件表达式的 BTA 状态,然后在分析两个分支语句之后,进行两个 BTA 环境的合并.在 BTA 环境的合并过程中,需要针对每个变量的 BTA 状态,计算它们的最小上界.如果条件表达式的 BTA 状态为动态,则需要将分支语句中被赋值的所有变量的 BTA 状态设置为动态.在循环语句的处理过程中,初次分析得到的变量 BTA 状态,在循环体的反复执行中可能发生变化.例如,表 1 中第 7 行的对象成员变量 *o.a* 在初次分析中被标为静态,但该变量在第 8 行被标为动态,在反复循环时被再次引用,因此应被标为动态.这种现象要求 BTA 分析反复处理循环语句,直至 BTA 环境中这些变量的 BTA 状态不再发生变化为止.

在语句反向分析中,需要按照与程序执行相反的顺序,逐步分析复合语句中各条语句;记录每个被引用变量的 BTA 状态,并根据赋值语句的标注和被赋值变量的 BTA 状态来更新赋值语句的 BTA 状态.同时确定每个说明语句的 BTA 状态.限于篇幅,本文不再介绍该算法的详细步骤.

BTA 分析中语句正向分析和语句反向分析分别对源程序进行一遍扫描,计算量主要消耗在:(1) 语句语法树的扫描;(2) 控制语句处理中 BTA 环境的合并运算.前者与语法树结点的个数 n 成正比,后者主要取决于变量的个数 m 和控制语句的个数 k.在循环语句的处理中可能需要反复分析循环体模块,在多数情况下反复分析的次数小于 3 次.因此,可以认为,在最极端的情况下,语句正向分析的计算复杂度为 $O(n+m*k)$.然而,整个 BTA 分析的效率主要取决于对象方法 BTA 分析.对于具有复杂递归方法调用的程序,方法 BTA 分析中需要反复扫描整个源程序文本,才能够确定所有类方法的求值模式.语句正向分析的计算量和没有递归方法调用的对象方法 BTA 分析基本相同.

4 Java 语言的 BTA 系统

我们已经实现了一个 Java 语言的绑定时间分析系统.该系统设计考虑了多线程、异常处理功能以外的所有 Java 程序结构.如图 1 所示,系统首先通过一个程序变换程序将 Java 程序变换为一个 Java 子集描述的程序,该子集的主要部分见表 3.根据用户提供的部分求值要求,系统采用对象方法 BTA 分析程序,对源程序进行了分析,为每个类方法构造了一个求值模式,确定了输入参数和输出参数的 BTA 状态.进而采用语句正向分析程序,分析各个方法的定义,利用 BTA 环境跟踪变量的 BTA 状态,计算出整个程序各个部分的 BTA 状态.随后,通过语句反向分析再次分析各个方法的定义,完成所有变量和运算的标注,生成了具有 BTA 标注的 Java 程序.

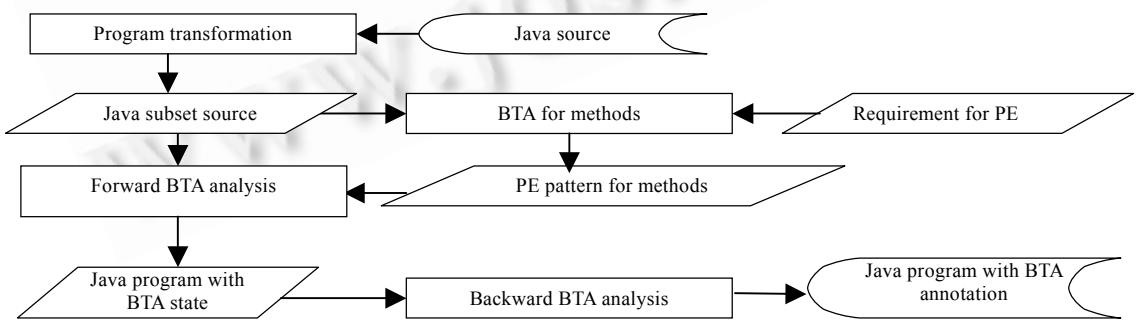

Fig.1 Binding time analysis system for Java

图 1 Java 语言的 BTA 系统

5 结束语

与传统的过程型语言的 BTA 分析技术相比,本文介绍的 BTA 分析方法能够更精确地分析对象引用变量、指针变量、数组变量的上下文敏感性,能够分别标注属于相同类、出现程序中不同位置的对象成员变量和数组元素,从而有效地扩大了部分求值的作用范围,进而提高了这种软件自动化技术的应用价值.

References:

[1] Anderson LO. Program analysis and specialization for the C programming language [Ph.D. Thesis]. Copenhagen: Department of Computer Science, University of Copenhagen, 1994.

[2] Consel C, Noel F. A general approach for run-time specialization and its application to C. In: Proceedings of the Conference Record of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. St. Petersburg Beach, FL: ACM Press, 1996. 145~156.

[3] Grant B, Mock M, Philipose M, Chambers C, Eggers SJ. Annotation-Directed run-time specialization in C. In: ACM SIGPLAN Symposium on Partial Evaluation and Semantice-Based Program Manipulation. Amsterdam: ACM Press, 1997. 163~178.

[4] Hornof L, Noye J, Consel C. Effective specialization of realistic programs via use sensitivity. Lecture Notes in Computer Science, 1997,1302:293~314.

[5] Hornof L, Noye J. Accurate binding time analysis for imperative languages: flow, context and return sensitivity. In: ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'97). Amsterdam: ACM Press, 1997. 63~73.