计算机科学2000Vol. 27№. 1

程序切片技术的研究与应用*)

The Research and Application of Program Slice Techniques

张勇翔 李必信 郑国梁

(南京大学计算机软件新技术国家重点实验室 计算机科学与技术系 南京210093)

Abstract This paper presents the instance of research and development in slice field until now. We introduce an algorithm to get a slice, which is based on PDG or SDG. We also introduce more complex slicing: dynamic slice, conditioned slice, OOP slice and the application of them in the field of software development.

Keywords Program slicing, PDG, SDG, Dynamic slicing, OOP slicing

软件逆向工程和维护通常是一种繁复的任务,它需要对程序的大量阅读,透切理解,耗费大量的人力物力,正是基于这种情况,产生了大量有关程序理解的技术,而程序切片技术是其中比较突出的一种。

程序切片技术(Program slicing)最早由 Mark Weiser 提出,他论述了通过遍历程序依赖图(Program Dependence Graph, PDG)得到过程内切片(intraprocedural slice)的技术。此后,Horwitz 通过将 PDG 扩展为系统依赖图(System Dependence Graph, SDG)从而得到过程间切片(interprocedural slice),解决了切片技术中的过程调用问题;而 Korel 和 Laski 引入了动态切片的概念。现在,程序切片技术已经发展成为一门成熟的技术,出现了许多切片标准及其算法和相应的实现工具。

一般而言,程序切片可定义为:影响变量 v 在程序 P 中某一点状态的所有语句和断言的集合,程序切片实际上是得到了程序 P 的一个有效子集,而省略了其他不相关代码。

本文将比较全面地介绍程序切片技术的定义标准,生成算法和具体应用。

一、基本概念与技术

1. 静态切片定义

定义1 静态切片:对于程序 P 中某一点 s,变量 v 的相对于静态切片标准(s,v)的切片包括程序 P 中与 v 在 s 状态相关的所有语句。

n	Statement
1	B=1
2	C=2
3	D=3
4	A=D
5	If (A=D)then
6	D=B+D
7	C=B+D
8	Else
9	B=B+1
10	D=B+1
11	Endif
12	A=B+C
13	PrintA

(程序1)

在程序1中,考虑相对于静态切片标准(13,A)的切片,应包括12,11,9,8,7,6,5,4,3,2,1。切片为:

n	Statement
1	B=1
2	C=2
3	D=3
4	A=D
5	If(A=D)then
6	D=B+D
7	C=B+D
8	Else
9	B=B+1
11	Endif
12	A=B+C

2. PDG 图

为了构造程序切片,通常的算法均是采用基于程序依赖图的方法。

定义2 程序依赖图(PDG):程序 P 的程序依赖图表示为 G(N,E),其中 N={n∈N|n 代表程序 P 中的某一语句},E={(m,n)|m,n∈N,m 是否执行或执

*)本文工作得到江苏省应用基础(编号 BJ97036)资助。张勇翔 硕士生,研究方向为面向对象技术、程序理解;李必信 博士生,研究方向为面向对象技术、程序理解;郑国梁 教授、博士生导师,研究方向为软件工程、面向对象技术。

行次数取决于 n 中的断言(称为控制依赖)或者存在变量 v,在 m 中被定义,在 n 中被使用(称为数据依赖)}。

从定义可以看出,在程序依赖图中结点之间存在两种有向边:控制依赖与数据依赖,前者描述了程序中条件语句、循环语句等对嵌入其中的语句的控制关系,后者描述了在赋语句中左值对右值的数据依赖关系。通常,我们定义一个 Entry 结点,作为程序的入口结点,它与程序中语句的关系为控制依赖,程序1的 PDG 图如图1。

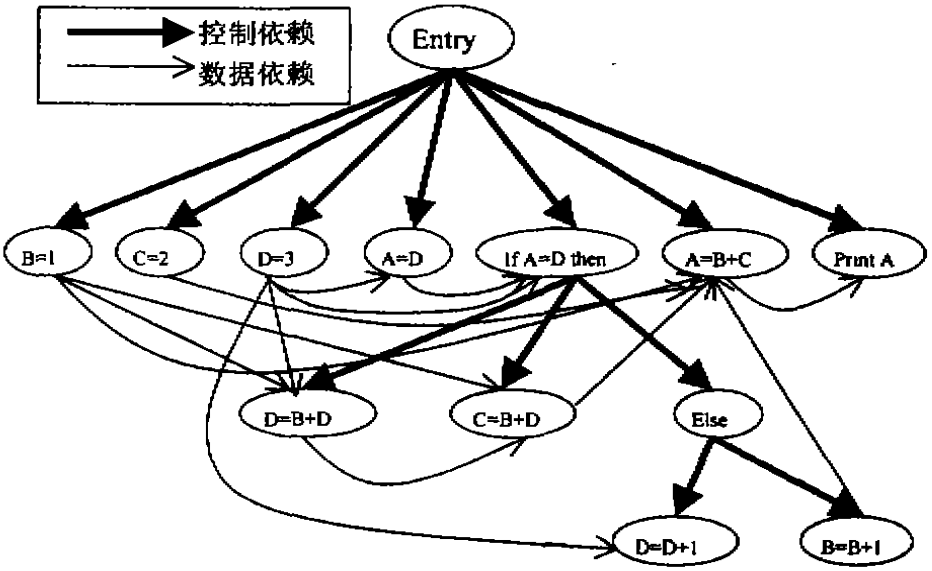

图1 程序1的 PDG 图

3. SDG 图

在 PDG 图的讨论中我们仅考虑了过程体内的情况,而没有考虑在有过程调用的情况如何获得程序切片。在过程调用中,我们要考虑过程外语句对调用语句,过程内语句对实参等的依赖关系。为此,将 PDG 图扩展为系统依赖图(SDG):

(1)对每个被调用的过程,增加相应的过程 Entry 结点。

(2)对调用语句的每个实参增加一个 Actual-in 和 Actual-out 结点,它们与调用语句的关系为控制依赖。

(3)对过程的每个形参增加一个 Formal-in 结点,对调用过程中状态发生变化的形参增加一个 Formal-out 结点,它们与过程 Entry 结点的关系为控制依赖。

(4)在调用语句结点与过程 Entry 结点之间增加一条属性为 Call 的边。

(5)在对应实参的 Actual-in 与形参的 Formal-in 结点之间增加一条属性为 parameter-in 的边。

(6)在对应实参的 Actual-out 与形参的 Formal-out 结点之间增加一条属性为 parameter-out 的边。

(7)如果 Actual-in 与 Actual-out 结点之间存在依赖关系,在二者之间增加一条属性为 summary 的边。

(8)对全局变量,可看作类似参数,用增加 Actual-in、Actual-out、Formal-in 和 Formal-out 结点的方法处理。

(程序2和图2是 SDG 的示例)

```
A=1                    | procedure AVG(I,J)
B=2                    | {
AVG(A,B)               |     if I<>J then
Print(A)               |         I=(I+J)/2
                       | }
```

(程序2)

4. 两阶段遍历切片生成算法

对于一个 SDG 图 G,计算变量 v 在结点 n 的静态程序切片可由以下算法得到:

阶段1:从结点 n 沿控制依赖,数据依赖,Call,Parameter-in 和 Summary 等边逆向遍历,得到结点集合 U。

阶段2:对每个 u∈U,从 u 沿控制依赖,数据依赖,Parameter-out 和 Summary 等边逆向遍历,得到结点集合 V。

所求切片(n,v)=U+V

二、程序切片标准的发展

1. 动态切片

Weiser 最早提出的切片概念包括了"可以影响程序中某一点变量状态"的所有语句，被称为"静态切片"，而 Korel 和 Laski 提出的"动态切片"概念，着重于寻找在程序 P 某一次特定的执行中(即程序 P 对应于某个特定的输入)，影响变量在程序中某一点状态的语句。动态切片比静态切片更加短小，应用也更加广泛。

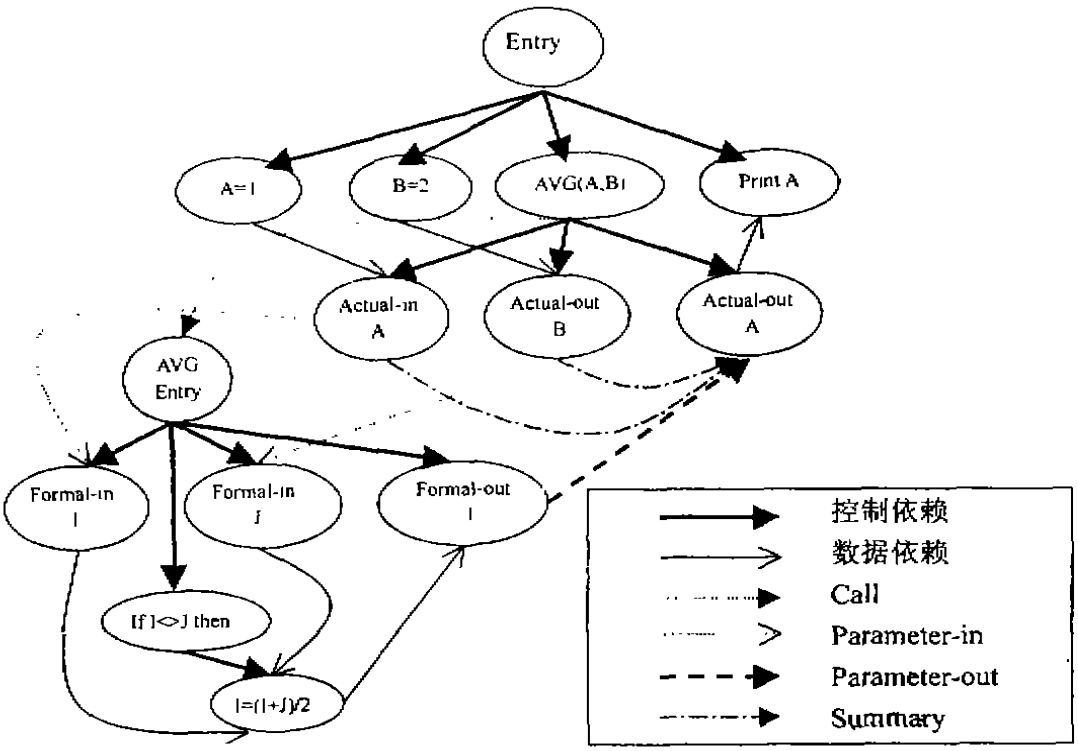

图2　程序2的 SDG 图

定义3　动态切片：对于变量 v，程序中某一点 n，输入序列 x，相对于动态切片标准(v,n,x)切片定义为在输入为 x 时，影响变量 v 在 n 状态的语句与断言。

定义4　动态循环切片：对于变量 v，程序中某一点 n，输入序列 x，整数 i，相对于动态循环切片标准(v,n,x,i)的切片定义为在输入为 x 时，影响变量 v 在 n 执行第 i 次状态的语句与断言。

构造动态切片可以通过两个途径：(1)执行记录。将程序的执行记录为由 n(I)(n 代表语句，I 代表执行次数)组成的记录串，通过分析记录串中元素的依赖关系，对记录串逐步削减，从而得到动态切片。(2)依赖图。通过在执行中记录程序依赖图中的结点或边的执行次数的方法，将记录数为零的结点或边去除，从而得到动态切片。

2. 条件切片

有一些切片标准在动态切片和静态切片之间采取了折衷的态度，比较有代表性的是准静态切片和条件切片。

由 Venkatesh 提出准静态切片标准，设计了一个输入序列前缀，来表示可能的输入，任何实际的输入均与此前缀进行比较，静态切片和动态切片均可看成准静态切片的特例：静态切片输入为空集，比较结果差异为全部，动态切片比较结果差异为零。而 Canfora 等在此基础上提出了条件切片的概念：

定义5　条件切片：对于变量 v，程序中一点 n，输入序列 x，关于输入序列的断言 W，相对于条件切片标准(v,n,x,w)的切片包括当 x 使 W 为真时，影响 v 在 n 状态的语句与断言。

通过对 W 的改变，条件切片的定义可以包容静态切片、动态切片的内容，条件切片关注的不是某个具体的输入，而是关心程序的初始化状态，可以看作是一种更广泛的准静态切片标准。

3. 前向切片

在前面所讨论的切片标准中，构造切片时都是采用逆向遍历的方法，考虑的是前面程序的影响。而 Horwitz 等提出的前向切片标准，考虑的是对后面程

序的影响。

定义8 前向切片：对于变量 v，程序中某一点 n，相对于前向切片标准(v,n)的切片表示为可以被 v 在 n 的状态影响的语句和断言集合。

构造前向切片的方法与一般切片相反：一般的切片标准是从程序中削减不相关的语句，而前向切片是向一个初始值为空的集合不断添加相关的语句。

4. 面向对象切片

在面向对象技术广泛应用的今天，如果讨论程序切片技术，我们就不得不研究面向对象语言的程序切片构造方法。

为了构造面向对象切片，我们必须将 SDG 图进一步扩展为 SDN 图(System Dependence Net)。SDN 图依据面向对象语言的特点比 SDG 图增加了三种依赖关系：选择依赖、同步依赖和通信依赖，其中，选择依赖表示不能直接决定结果的选择语句的控制关系；同步依赖表示在两个不同语句的执行开始和执行结束之间的同步关系；通信依赖表示通过过程间通信在两个变量间所建立的依赖关系。同时还要注意以下几个问题：

(1)在程序中用 VAR=NEW CLASS 语句等方法实例化一个类时，相当于对此类的构造函数进行了一次调用，在 SDN 图中，要相应添加此结点到构造函数的 Call，Parameter-in 和 Parameter-out 等边。

(2)在程序中用 DELETE VAR 语句等方法删除一个类时，相当于对此类的析构函数进行了一次调用，在 SDN 图中，要相应添加此结点到析构函数的 Call，Parameter-in 和 Parameter-out 等边。

(3)对类中所定义的变量，在每个函数中当作全局变量处理，增加相应的 Formal-in 和 Formal-out 结点。

依据 SDN 图，可利用两阶段遍历切片生成算法，构造相应的面向对象程序的静态切片。

三、程序切片技术的应用

程序切片标准的定义是与切片技术的应用密不可分的，不同的应用要求不同的切片定义标准。下面我们将论述程序切片技术应用于软件开发中的调试、测试和维护等阶段中。

1. 程序调试

在程序调试中，最常见的工作莫过于发现一个错误并找出所有与错误有关的语句，程序切片工具将帮助程序员很容易地做到这一点。例如使用动态切片，可以根据目标得到比源程序少得多的切片程序集，使错误定位比原来容易得多。

在大型软件项目的调试中，经常会遇到对于某些输入会产生正确结果，而对另一些输入则会产生错误结果。对于这种情况，如果采用多次运行程序，逐条语句跟踪的方法，将会耗费大量的时间。而采用程序切片工具，我们可以构造输入语句的前向切片，和输出语句的动态切片，并取两者的交集。这样可以极大地缩小考察的程序范围。

2. 程序测试

对一个软件进行大规模的测试后，对软件进行了某种小小的修改，是否要对整个程序进行重新测试，是一个相当重要的问题。

从程序切片的角度去考虑这个问题，首先我们应该找到程序新旧版本的不同之处。比较二者的切片和依赖图，对那些具有相同切片的结点不用考虑(很明显，这些结点在新旧版本中的作用相同)；将那些在新版本依赖图中出现，而在旧版本依赖图中不出现的结点，标记出来，称为"影响点"。(新旧版本程序的不同行为正是这些"影响点"决定的。)

对于这些"影响点"，我们做它们的静态切片与正向切片，并取两者的交集，这样，如果我们要对程序进行重新测试的话，只需要对这个交集进行测试，以减少工作量的耗费。

3. 软件维护

对于软件维护人员来说，其任务不仅是阅读理解现存的程序，更重要的是，在对程序整体不发生影响的前提下，对程序进行恰当的修改。具体而言，例如改变对一个变量的赋值，会不会对其他变量产生影响？

为解决这一问题，先构造每一变量在程序终止状态的静态切片，对于变量 v，程序中的语句可如下分类：

(1)独立部分：存在于 v 的静态切片中，而不在其他变量的静态切片中出现的语句。

(2)依赖部分：存在于 v 的静态切片中，并且在其他变量的静态切片中出现的语句。

(3)无关部分：不存在于 v 的静态切片中，而在其他变量的静态切片中出现的语句。

对于在软件维护中是否可对变量 v 的赋值进行改变可依据以下分类：

(1)可变型：对变量 v 所有的赋值操作均在它的独立部分。

(2)不可变型：至少有一条赋值语句存在于变量 v 的依赖部分，此时改变 v 的赋值可导致对其他变量的影响。

采用这种方法，可最大限度地减少所做修改对系统稳定度的影响，保障软件维护工作的正常进行。

4. 逆向工程

逆向工程关心的问题是如何对现存的软件系统进行理解，包括对程序源代码的抽象，对程序基本原理和基本算法的理解等。程序切片工具可以对此项工作进

行有力的支持:通过构造程序切片,或者对不同程序版本的切片的比较,可以对原程序进行精简,除去界面管理,输入输出处理等我们所不关心的部分,将原程序中分散的关键部分集中进行分析。

类似地,在对软件系统进行重用方面,也可以用程序切片工具进行处理,使用程序切片工具提取原系统中的可用部分,可有效地避免对同一过程的重复提取,避免提取出冗余代码。

结束语 由于篇幅所限,我们无法对程序切片技术的每项应用给出完整的论述和证明,但是毋庸置疑的是,各种切片标准和切片工具已经实际应用到软件系统的开发中,程序切片的思想给予其他软件技术的研究以极大的启发与帮助,同时,程序切片技术也是一门不断发展的技术,不断有新的切片标准,新的算法被提出,以使其更加完善,更加贴近实际应用。相信在以后的研究,将更加证实程序切片技术的价值与前景。

参考文献

1 Binkley D W, Gallagher K B. Program Slicing. In: Advances in Computers, 1996

2 Livadas P E, Croll S. Computer and Information Sciences Department University of Florida. Program Slicing: [Technical Report SERC-TR-61-F]. Software Engineering Research Centre, October 1992

3 Harman M, Simpson D. Department of Computing, University of Brighton. The next 700 slicing criteria Formal Aspects of Computer Science, 1995

4 M Sloane A, Holdsworth J. Beyond Trational Program Slicing. Department of Computer Science, James Cook University

5 Zhao J J, et al. Static Slicing of Concurrent Object-Oriented Programs. In: Proc. of the 20th IEEE Annual Intl. Computer Software and Applications Conference (COMPSAC'96). Seoul, Korea, 1996. 312～320

(上接第27页)

划用 Java 改写旧的应用系统时易犯的错误。

在网络计算环境中利用 Java 的最好途径是由 EJB 提供服务器端的构件,而由 JavaBeans 提供客户端的构件,两者结合在一起,将向"网络就是计算机"之路迈出一大步。

总之,EJB 技术将使得 Java 在分布式计算中的地位得到加强,为基于 Java 的应用系统提供了一个框架,和目前的许多系统和模型相比,EJB 具有许多优越性,种种迹象表明,EJB 有可能成为分布式应用系统的服务器端构件模型的首要选择。

结论 将来的软件开发必定包含三个技术:面向对象、网络及数据库,Java 的分布式计算技术代表的是一种前沿技术,使得复杂的网络应用系统开发变得容易,Java 的设计目标之一是支持开发代码少而又易于理解的复杂系统,对于这个目标来说,Java 所取得的成功领域之一是在分布式计算方面,传统上,分布式应用系统的开发以其复杂性著称,一个简单的分布式计算任务可能需要数页的代码,对于 Java 来说,这些任务中的许多任务已被减少到一条或几条语句,显然由于 Java 技术对分布式计算的强有力的支持,分布式应用系统的开发将变得容易。

参考文献

1 Gupta A, et al. Implementing Java Computing: Sun on architecture and applications deployment. IEEE Internet Computing, 1998, March-April: 60～64

2 Sun Microsystems. Java RMI. Available at: http://java.sun.com/products/jdk/rmi/index.html

3 Sun Microsystems. Java IDL. Available at: http://java.sun.com/products/jdk/1.2/docs/guide/idl/index.html

4 Object Management Group. CORBA/IIOP Version 2.2 Specification. Available at: http://www.omg.org/corba/corbaiiop.html

5 Anne Thomas. Enterprise JavaBeans—— Server Component Model for Java. Available at: http://java.sun.com/products/ejb/white_paper.pdf

6 Sun Microsystems Inc. Enterprise JavaBeans Specification. Available at: http://java.sun.com/products/ejb

7 Rawn Shah. Enterprise JavaBeans: Industrial-strength Java. Available at: http://www.nc worldmag.com/ncworld/ncw-01-1998/ncw-01-ejbeans.html

8 Sun Microsystems Inc. JavaBeans. Available at: http://java.sun.com/beans.

9 Kozaczynski W, Booch G. Component-based software engineering, IEEE software, 1998, 15(5): 34～36