程序切片技术综述

西安电子科技大学 王伟 陈平 (西安 710071)

摘要 程序切片技术在软件维护、程序调试、测试、代码理解及逆向工程等方面有许多应用。文章介绍了目前已有的程序切片技术及其应用领域，提出面向对象程序应用技术时需要考虑的问题及相应的解决方法。

关键词 程序切片技术,逆向工程

1.引言

程序切片技术 (program slicing)在软件维护、程序调试、测试、代码理解以及逆向工程等方面有许多应用。比如在程序调试中,当错误特征出现时,我们希望能够通过一种手段，找到可能产生该错误的源代码部分。我们或许不能对这样的源代码准确定位，但我们可以做到通过剔除程序中不可能产生该错误的部分，而把错误限定在一个较小的范围中。程序切片就是一种有效实现上述目的的技术。

2.程序切片技术的发展史

程序切片 (program slice)这一概念最早是由 W 在 1979 年提出,此后出现了许多略有不同的定义以及用于计算切片的算法。大体上说,程序切片技术的发展经历了从静态到动态、从前向到后向、从单一过程到多个过程、从非分布程序到分布式程序等几个方面。S Horwitz 等人给出的程序切片定义是：

“一个程序切片是由程序中的一些语句和判定表达式组成的集合。这些语句和判定表达式可能会影响在程序的某个位置 (常用行号标识)p 上所定义的或所使用的变量 v 的值。 (p,v)称为切片准则 (slicing criterion)。”

2.1 静态切片技术 (static slicing)

静态切片技术是指在计算程序切片时使用的是静态的数据流和控制流分析方法。该技术对程序的输入不做任何假设,所做的分析完全以程序的静态信息为依据。使用该技术的工作量较大，因为要分析程序所有可能的执行轨迹，所以相对于动态切片技术，静态切片技术一般用于程序理解与软件维护方面。W 最初提出的程序切片概念就属于静态切片范畴。

2.2 动态切片技术 (dynamic slicing)

动态切片技术使用的是动态的数据流和控制流分析方法，因此切片的计算过程依赖于程序的具体输入。采用这一技术,每一次的计算工作量较小，但每一次的计算都不尽相同，因此动态切片技术多用于程序调试、测试方面。

2.3 分解切片技术 (decomposition slicing)

在软件维护中，常常会遇到这样的要求：把涉及一个变量所有相关计算都封装在一个模块中。对一个小型、简单的程序，

```
scanf("%d",&x);
y=2*x;
if(y>x){
  x=x+1;
  y=y*y;}
eles{
  x=x*2;
  y=y-x;}
printf("%d",x);
```

```
scan("%d",&x);
y=2*x;
if(y>x)
  x=x+1
eles
  x=x*2;
```

```
scan("%d",&x);
y=2*x;
if(y>x)
  x=x+1
```

源程序 | (a)⟨p,v⟩=⟨8,x⟩ | (b)正向条件 x>0

图 1 一个例子及其切片)

尚可满足这种要求：但对一个较大且比较复杂的程序就不是件容易的事了。

1991 年,K B Gallagher 等人提出了分解切片技术，一种以把程序分解成不同模块为目的程序切片技术。分解切片构成的集合 (其本身仍是程序切片),可以捕获程序中对某一个变量的所有计算。不同于传统的程序切片，分解切片不依赖于语句在程序中的位置 (常用的是行号)。那些构成分解切片的程序切片并非任意排列,而是按照一定的规则排列成网格 (lattice)。通过使用这个网格，可以实现对程序的分解，显然这种分解是以程序切片技术为基础的。

使用分解切片技术，维护人员可以直观地判断出一个模块中哪些语句和变量可以被安全地修改，也即这种修改不会扩散

到其他模块中，哪些语句和变量不能随意被修改。K B Gallagher 等人还给出了一些用于禁止那些会影响其他模块的修改的原则。遵照这些原则所做的修改就可以在线性时间内回馈到原程序中。带来更大的好处是：维护者可以在一个模块中测试对语句或变量的修改，而不必担心这种修改是否会影响到其他模块。由此就可以省略传统软件维护过程模型中的回归测试。

2.4 条件切片技术（conditioned slicing）

目前对程序的理解主要有两种方法：静态分析或者是动态分析。但这两种方法都有些过于极端，前者对程序的理解只局限于对程序源代码的分析，获得的是静态信息，过于片面；后者对程序的理解则以程序多次运行的测试结果为基础，这些测试结果依触发程序运转的外界输入的不同而不同，又过于复杂。Canfora Cimitile 和 Lucia 于 1998 年提出的条件切片技术避免了上述两个极端。

Canfora 等人提出的条件切片技术通过增加一个条件扩展了传统的静态切片准则。在进行切片算法时，只有满足该切片条件的那些输入才会被分析。这个条件对应着程序的某个或某些初始状态。可以看出，条件切片是一种施加了一定条件的动态切片。如果存在从满足切片条件的任一个初始状态出发都不可能被执行的语句，那么把这些语句去除掉后，就得到了在这个切片条件下的程序切片，即条件切片。例见图 1 中的（b）。

Canfora 等人提出的条件是一种正向的条件，而 2001 年 Chris Fox 等人提出的条件则是一种逆向的条件。在生成逆向条件切片时，去除的是那些其执行不可能导致一个满足切片条件的程序状态的语句。Chris Fox 等人结合正向和逆向条件这两个概念，进一步提出了前提 / 后继条件切片技术（pre/post conditional slicing）。该技术在去除语句时采用如下的标准：当程序从一个满足前提条件的初始状态出发，遇到一些语句，这些语句的执行不可能导致后继条件的否命题为真，那么这些语句应该被去除。如果一个程序正确的满足切片的前提和后继条件，那么使用该技术就能去除影响程序行为的语句，而留下那些错误或者是无关的语句，后者往往是使用传统切片技术无法检测到的。

```
class Base{                          void main(){
  in a, b;                             int x;
  virtual void vm(){a = a + b; }       Base *pb;
public:                                scanf("%d", &i);
  Base(){a = b = 0; }                  if(x <0)
  void m1(){if(b> 0)vm(); }              pb = newBase;
};                                     else
class Derive: public Base{               pb = newDerive
  int c;                               pb. m1();
  virtual void vm(){c = a * b}       }
public:
  Derive(): Base(){c = 0; }
};
```

图 2

2.5 无定型切片技术（amorphous slicing）

如果在程序调试领域应用程序切片技术，为了使得到的程序切片包含那些可能引起错误的语句，在去除语句时就必须施加一定的约束。显然，这些约束是必要的，因为它们剔除了那些其执行不影响 p 处所切变量 v 的值的语句，从而使调试者都能够在一个较小的范围内有效地发现并且修改错误。现在的问题介是：这种约束在程序理解领域是否过于苛刻？

M Harman 和 S Danicic 发现的程序理解领域，这些约束可以被适当放宽。他们于 1997 年提出了无定型切片的概念。这种切片技术继承了传统切片技术在简化原程序的过程中保留原程序语义的功能，唯一不同之处在于，它充分利用了任何能保留语义映射的简化技术，尽可能地简化程序。

无定型切片的特点使得它更适合程序理解领域，而传统的切片技术则更多地应用在调试领域。

2.6 对象切片技术（object slicing）

由于面向对象程序的复杂性，简单套用以往对过程型程序进行切片的方法是不合适的。为了获得更准确、更有效的程序切片，D Liang 和 M J Harrold 在文献［2］中提出了对象切片技术这一概念。他们的工作是通过扩展系统依赖图（SDG，见下一部队分介绍）实现的。

3 对面向对象程序应用切片技术

自 Weiser 提出程序切片概念后，人们提出了许多用过程程序的切片方法，但这些方法并不适用于 OO 程序。因为 OO 编程语言提出了一些新的概念与特性，比如类、对象、动态绑定、封装、继承、消息及传递及多态。这些新的内容极大地增

强了 OO 编程语言解决实际问题的能力，但同时也给程序分析带来许多困难。

由于 OO 程序的复杂性，目前对它的研究更多地侧重于静态切片这一部分。这部分主要讨论一些对 OO 程序应用程序切片技术时需要他细考虑的方面。在这之前先简单地介绍一下 SDG （System Dependence Graph,8）的概念。目前的程序切片大都是以 SDG 为基础，在其上利用图的可达性算法获得的。

SDG 包含许多过程依赖图 （Procedure Dependence Graph，PDG），每一个 PDG 对应程序中的一个过程。PDG 由点和边组成，其中点代表语句或判定表达式，边又分为数据依赖边和控制依赖边。前者代表语句或表达式能否运行所依赖的控制条件（例见图 2）。有关 SDG 及基于图的可达性算法的切片算法的介绍可以参考文献。

问题一：在构造 OO 程序的 SDG 时，简单地为类的成员变量构造一个参数 （actual or formal）节点是否合适？

问题二：如果被调用的方法 （或函数）接收的参数不是简单数据类型，而是一个对象，那么在 SDG 中该如何表示这些参数对象？

问题三：如果程序中存在多态对象，为每一种可能的消息处理方法构造一个调用点 （call vertex）的方法是否足够？

问题四：当程序中存在继承时，对子类采用 “重用父类的表示，只对子类中定义的部分重新构造 ” 的策略是否周到？

显然，问题一中的做法明显不妥，它没有考虑类的成员变量在各个对象中均有一份独立拷贝的事实，因此会在一个类的两个不同对象间产生错误的数据依赖。当调用参数是一个对象时，用一棵根代表对象、孩子节点代表成员变量的树来表示这个参数更合适。这里要注意嵌套，即对象的成员变量又是另一个对象的情况。问题三中采用的方法实际上相当于用一些类型不同的对象 （类型不同、名字相同）来表示多态现象，由于构造 SDG 的算法不能区别类型不同但名字相同的情况，问题三就又变成问题一了。因此正确的做法是把多态对象也表示为一棵树：树的根表示多态对象自身，其孩子节点表示所有可能的类型。问题四中的处理方案忽略了一种情况：一个方法在父类中声明，但方法体中至少存在一条语句，该语句直接或间接调用一个在子类中被重定义的虚拟函数。

4.结束语

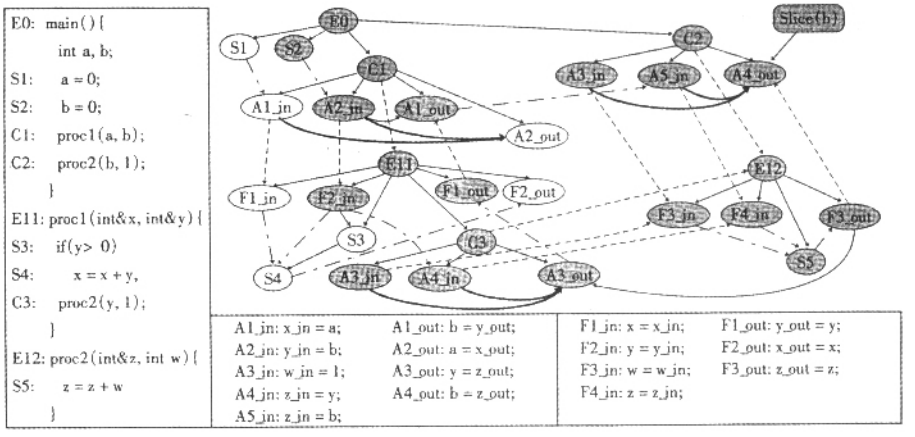
目前对程序切片技术的研究主要集中在获得过程型程序的静态切片方面。如何解决面向对象技术引入的新问题，如何更准确、更快速、更有效地计算出切片，是目前该技术研究的重点。“C3I 系统应用软件逆向工程开发工具研究 ” 项目主要侧重于程序理解领域，其研究目标是研究制一套逆向工程工具，以提供符合 UML 标准的动态模型的逆向生成、符合 UML 标准的静态模型的逆向生成与分层抽象等方面的能力。目前对程序静态结构和其动态运行信息的了解是通过开放编译器技术实现的，虽未直接用到程序切片技术，但对其的研究在了解进行逆向工程的常用手段方向还是十分有益的。

参考文献

（1） S Horwitz, T Reps and Binkley. Interprocedural Slicing Using Dependence Graphs. ACMTrans on Programming Languages and System, 1990,121:26-60.

（2） D Liang and M J Harrold. Slicing Objects Using System Dependence Graphs. Proceedings of the 1998 International Conference on Software Maintenance, November 1998, 358-367.

图 3 一个存在虚拟函数调用的例子



一个简单程序的 SDG。图中椭圆表示语句和参数 ;实线表示控制依赖 ;短边虚线表示数据依赖 ;点虚线表示调用和传递 ;粗实线表示摘要边 （summary edge）。图中阴影节点是以 （2,b）为切片标准计算出的切片。