基于简化系统依赖图的静态粗粒度切片方法*

李必信[1,2], 王云峰[1], 张勇翔[1], 郑国梁[1]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093);

[2](中国科学技术大学 计算机科学技术系,安徽 合肥 230027)

E-mail: libx@seg.nju.edu.cn; zhenggl@nju.edu.cn

http://www.seg.nju.edu.cn

摘要: 基于系统依赖图是计算面向对象程序切片的一个有效方法.但是,系统依赖图的缺点是太复杂,而且在建立系统依赖图的过程中容易出错,一旦出现错误就可能导致切片结果的不准确.通过对系统依赖图进行简化,得到了简化的系统依赖图.它省略了那些表示输入参数和输出参数的结点和概括边.同时,还定义了一种面向对象程序的粗粒度切片概念,讨论了它的性质,分析了它与细粒度切片的关系,并基于简化的系统依赖图计算面向对象程序的粗粒度切片.最后还讨论了切片技术的简单实现.

关键词: 系统依赖图;简化系统依赖图;粗粒度切片;静态切片;面向对象

中图法分类号: TP311 **文献标识码**: A

程序切片是一种程序分析技术,它的概念最早是1979年Mark Weiser在他的博士论文[1]中提出来的.Weiser定义的切片由一个程序点 p 和一个程序变量的子集 V 构成,是一个可执行的程序,通过从源程序中移去零个或多个语句来构造.他提出的切片算法通过在控制流图(control flow graph,简称CFG)上使用数据流分析来计算过程内和过程间的切片.K. J. Ottenstein和L. M. Ottenstein 1984年在文献[2]中提出的切片标准由一个程序点 p 和一个在 p 定义的或使用的变量 v 构成.他们在一个程序依赖图(program dependence graph,简称PDG)上使用图形可达性算法来计算切片.根据他们对切片含义的理解,切片是由程序中所有可能影响 v 在 p 点的值的语句和谓词构成.1990年,S. Horwitz, T. Reps和D. Binkley[3]使用依赖图来计算切片,他们开发了一种过程间的程序表示——系统依赖图(system dependence graph,简称SDG),并实现了一个基于系统依赖图的两阶段图形可达性切片算法.由于在使用调用位置的可传递的依赖流信息中包含被调用过程的上下文环境,所以,该算法的计算精度要比以前的算法高.

程序切片有静态和动态两种.对于程序的某个变量,静态切片计算出来的该变量的值与原来的程序在任意输入下执行时计算出的该变量的值相同.静态切片考虑了程序所有可能的执行路径,通过分析程序的源代码来获得程序的有关信息,是一种不包括人机交互作用的在编译时确定的方法.动态切片计算出来的变量的值和原来的程序计算出的该变量的值在某个输入下是相同的,动态程序切片记录多次程序运行的测试结果,并使用用户的实际输入产生精确的数据流信息(即在程序的某个特定执行过程中产生的依赖).程序切片还存在粒度问题,不同粒度的程序切片的作用是不同

* 收稿日期: 1999-04-21; 修改日期: 1999-12-03

基金项目: 国家863青年基金资助项目(863-306-QN2000-2);江苏省自然科学基金资助项目(BK99038)

作者简介: 李必信(1969-),男,安徽庐江人,博士生,讲师,主要研究领域为面向对象技术,程序分析,程序理解;王云峰(1964-),男,湖北宜兴人,博士生,讲师,主要研究领域为面向对象技术,形式化技术;张勇翔(1975-),男,江苏连云港人,硕士生,主要研究领域为面向对象程序理解;郑国梁(1937-),男,浙江桐乡人,教授,博士生导师,主要研究领域为为软件工程,面向对象技术,形式化技术.

的.例如,细粒度切片可用于程序调试、程序测试、度量分析等;粗粒度切片有助于提取程序结构、软件的构件和构架等,从而对逆向工程、软件理解和维护等起到一定的辅助作用.本文讨论面向对象程序的静态粗粒度切片问题.面向对象的程序切片技术起步比较晚,目前,这方面的主要工作有:M. J. Horrold 等人利用 SDG 切片面向对象的程序,提出面向对象的语句切片和对象切片的概念,他们采用的计算切片的方法是把文献[3]中的两阶段图形可达性算法运用到 SDG 上;J. J. Zhao 提出一种动态面向对象的依赖图(dynamic object-oriented dependence graph,简称 DODG),并在 DODG 上实现了计算面向对象程序的动态切片算法[5];A. Krishnaswamy 提出一种面向对象的程序依赖图(object-oriented program dependence graph,简称 OPDG),并在其上实现计算语句切片的算法[6].OPDG 通过对传统的程序依赖图进行面向对象的扩充而获得,但 OPDG 不能表示虚继承、动态对象等问题,因而 OPDG 是不完全的面向对象程序表示.Frank Tip 总结了程序切片的各种技术和方法,可参阅文献[7].本文从一个简化系统依赖图(simplified system dependence graph,简称 SSDG)出发,提出面向对象程序的粗粒度切片概念和算法,成功地降低了利用 SDG 计算切片的复杂度,提高了分析大型面向对象软件的效率.利用 SSDG 同样可以计算对象切片、类层次切片以及动态切片.

1 例子分析和动机

```
CE1    class Elevator{
        public:
ME2      Elevator(intl_top_flcor)
S3         {current_floor=1;
S4       current_direction=UP;
S5       top_floor=1_to p_floor;}
ME6    virtual~Elevator(){}
ME7    void up()
S8       {current_direction=UP;}
ME9    void down()
S10      {current_direction=DOWN;}
ME11   int which_floor()
S12      {returncurrent_floor;}
ME13   Direction direction()
S14      {return current_direction;}
ME15   virtual void go(int floor)
S16      {if(current_direction==UP)
S17        {while((current_floor! =floor)&&
             (current_floor<=top_floor))
C18          add(current_floor,1);}
         else
S19        {while((current_floor! =floor)&&
             (current_floor>0))
C20          add(current_floor, -1);}}
       private:
ME21     add(int&a,const int&b)
S22      {a=a+b;}
       protected:
        int current_floor;
        Direction current_direction;
        int top_floor;
       };
CE23   class AlarmEvevator:public Elevator{
        public:
ME24     AlarmElevator(inttop_floor);
C25      Elevator(top_floor)
S26        {alarm_on=0;}
ME27     void set_alarm()
S28        {alarm_on=1;}
ME29     void reset_alarm()
S30        {alarm_on=0;}
ME31     void go(int floor)
S32      {if(alarm_on)
C33      Elevator::go(floor)
         }
        protected:
         int alarm_on;
        }
ME34   main(int argc,char)
        Elevator *e_ptr;
S35     if(argv[1])
S36      e_ptr=new AlarmElevator(10);
       else
S37      e_ptr=new Elevator(10);
C38    e_ptr->go(5);
S39    cout<<"\nCurrently on floor:"
         <<e_ptr->which_floor()<<"\n";
       }
```

Fig. 1 A C++ program, the underlined parts represent the slice with respect to slicing criterion (S39, current_floor)

图 1 一个 C++程序,下划线部分表示其关于切片标准(S39,current_floor)的切片

M. J. Horrold 等人成功地对图 1 中的 C++程序进行了面向对象的静态细粒度程序切片的计

算[4]:先利用系统依赖图详细地给出了该程序的SDG描述,然后在SDG上利用两步图形可达性算法计算关于切片标准〈C20,current_floor〉和〈S39,current_floor〉的程序切片.所以说,利用SDG切片描述面向对象的程序是一种可行的方法.然而,利用SDG描述程序,特别是描述面向对象的程序既复杂又易出错.为了实现面向对象程序的动态切片,J. J. Zhao在对SDG进行简化的基础上提出一种动态的面向对象的依赖图(DODG),并在DODG上实现了计算面向对象程序的动态切片算法.细粒度程序切片在程序理解过程中起到一定的作用,但有时却不是必需的.为此,我们定义一种粗粒度程序切片,即切片集合实际上是由Main函数及自由标准过程中的一些语句和类中的一个个方法所构成.在计算这种切片时,可以不必考虑方法中的每条语句,只考虑方法接口就可以了.实际上,为了达到软件理解、逆向工程的目的,从切片到方法就足以使我们理解面向对象程序的结构、软件构件和构架以及一般的依赖关系了.为此,我们对SDG进行简化,然后利用这种简化的系统依赖图来实现面向对象程序的粗粒度切片算法.

2 SDG及其简化方法

2.1 SDG概述

系统依赖图最早是由S. Horwitz, T. Reps和D. Binkley在文献[3]中提出来的.当时,人们都致力于用程序依赖图来描述各种程序.程序依赖图(program dependence graph,简称PDG)是程序的一种图形表示,它把控制依赖和数据依赖包含在单个的结构中.如果给定程序中的语句X和Y,则X和Y可以通过控制流或者数据流来彼此关联.如果从Y至少可以引出两条路径,其中一条总会导致X的执行,而另一条可能导致X不执行,则称语句X控制依赖于语句Y;如果有一条从Y到X的路径,同时存在一个在Y点定义在X点使用的变量,同时该变量在沿从Y到X的路径上的其他任何地方都没有被重新定义,则称语句X数据依赖于语句Y.PDG的形式定义由一个控制依赖子图(control dependence subgraph,简称CDS)、一个控制流图(CFG)和一个数据依赖子图(data dependence subgraph,简称DDS)组成[1].其中CDS包含了程序中的控制依赖;CFG描述了一个程序中的控制流,它类似于正常情况下的流图;DDS是一个程序中语句之间所有数据依赖的集合.CDS包含几种类型的结点,即语句结点表示程序中的语句;域结点概括了域中语句间的控制依赖;谓词结点(由此可以引出两条边)表示程序中的策略或分支条件.CDS中的域结点可以利用相同的控制依赖来集合语句.DDS包含语句间的数据依赖.可通过在CDS的结点之间插入数据依赖边来构造DDS.程序依赖图通常是为单个过程的程序而定义的.但是,现实世界的程序一般是由多个过程组成的,所以,必须考虑使用一种与现实世界程序匹配的表示方法.在这种情况下,S. Horwitz等人提出了能够与现实世界匹配的方法来描述由多个过程相互作用而构成的复杂程序,这种方法就是系统依赖图.SDG是一棵经过装饰的表示程序的语法分析树.形式地说,它是由一个程序依赖图和一组过程依赖图(procedure dependence graph,简称PrDG)构成的有向、带标记的多重图.PDG模型化软件系统中的主程序,PrDG模型化软件系统中的多个过程体.SDG可以用来处理过程间的数据流和控制流,并能表示参数传递.SDG允许过程间分析.但是,传统的SDG必须经过面向对象的扩充以后才能够描述面向对象的程序.

2.2 简化SDG的原则

为了实现面向对象程序的粗粒度程序切片,我们根据下面的简化原则对面向对象程序的系统依赖图或用来不完全描述面向对象程序的各种系统依赖图进行了严格的简化.

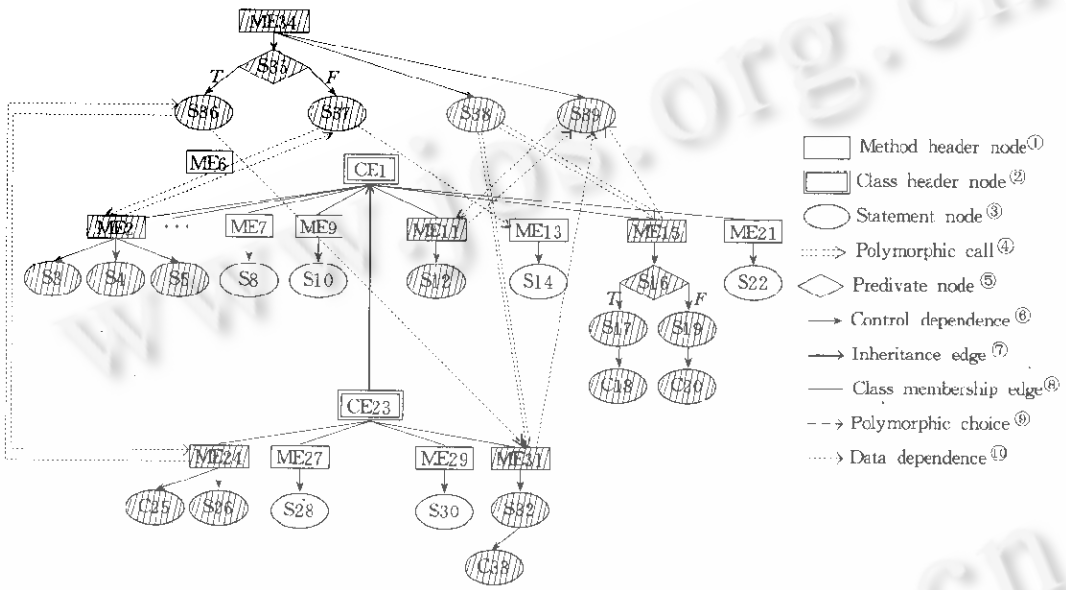
(1) 使简化以后的 SDG 省略了描述许多细节问题的结点和边,扩大了每个方法入口结点的含义(隐含参数结点及参数之间的数据传递);

(2) 两个方法的参数结点之间的数据依赖关系用方法之间的数据依赖来表示;

(3) 每个过程依赖图只需用过程入口结点表示即可,而不必再展开表示;

(4) 必要情况下,可以利用带参数的方法入口结点来表示是通过哪个参数发生的依赖关系.

图 2 是图 1 所给出的例子程序的 SSDG 表示(其 SDG 表示参见文献[4]),其中阴影部分表示该程序关于切片标准〈S39,current_floor〉的粗粒度程序切片.



①方法首部结点,②类首部结点,③语句结点,④多态性调用边,⑤谓词结点,⑥控制依赖,⑦继承边,⑧类成员边,⑨多态性选择边,⑩数据依赖.

Fig. 2 Representing C++ program using simlified SDG, shade part showing program slcing with respect to slicing criterion 〈S39,current_floor〉

图2 利用简化的SDG来表示图1中的C++程序,阴影部分表示关于切片标准〈S39,current_floor〉的程序切片

2.3 SDG 和 SSDG 大小的比较

表 1 列出了影响 SDG 大小的各种因素.我们给参数顶点 ParamVertices 一个约束,并使用这种约束来计算一个方法或过程大小的上界.

Table 1 Parameters affecting the size of an SDG

表 1 影响 SDG 大小的因素

Vertices①	Greatest number of predicates and assignments in a single method or procedure②
Edges③	Greatest number of edges in a single method or procedure④
Params⑤	Greatest number of formal predicates in any method or procedure⑥
Globals⑦	Number of global variables in the system⑧
InstanceVars⑨	Greatest number of instance variables in a class, including those in all instantiated classes⑩
CallSites⑪	Greatest number of call sites in any method or procedure⑫
TreeDepth⑬	Depth of inheritance tree determining number of possible indirect call destinations⑭
Methods⑮	Number of methods or procedures in the system⑯

①结点数,②一个方法或过程中谓词和赋值语句的最大数,③边数,④一个方法或过程中边的最大数,⑤参数个数,⑥一个方法或过程中形式参数的最大数,⑦全局变量数,⑧系统中的全局变量数,⑨实例变量数,⑩一个类的实例变量的最大数(包括所有实例化类中的实例变量,⑪调用位置数,⑫一个方法或过程中调用位置的最大数,⑬继承树深度,⑭确定可能的间接调用目标的继承树深度,⑮方法,⑯系统中方法或过程的数目.

在SDG中,对方法m来说,有以下各式成立:

(1) $ParamVertices(m)=params+Globals+InstanceVars$;

(2) $Size_{SDG}(m)=O(Vertices+Callsites*(1+TreeDepth*(2*ParamVertices(m)))-2*ParamVertices(m))$;

在整个系统中,类的数量是给定的,一个类中方法的数量也是给定的,则一个SDG中结点数量的上界是$Size(SDG)=O(Size_{SDG}(m)*Methods)$.在SSDG中,由于省略了参数结点,则对一个给定的方法m,表示它的大小的上界是$Size_{SSDG}(m)=O(Vertices+Callsites*(1+TreeDepth))$.故SSDG的大小上界是$Size(SSDG)=O(Size_{SSDG}(m)*Methods)$.显然,因为$Size_{SSDG}(m)$,故$Size(SDG)\geqslant Size(SSDG)$.

3 粗粒度程序切片

3.1 定义和性质

面向对象程序的切片标准$\langle s,v\rangle$有两种解释:(1) v和s分别是某个类的一个方法中的一个变量和一条语句.(2) v和s分别是主程序或任何一个自由标准过程中的一个变量和一条语句.第1种情况下的变量可能是该方法的局部变量,也可能是全局变量或者是类的数据成员.第2种情况下的变量s可能是局部变量;也可能是全局变量.在没有类和对象存在的情况下,这种程序切片实际上等同于过程性程序的切片,否则,就必须考虑使用面向对象的程序切片方法.显然,上面的例子中的切片标准〈C20,current floor〉属于第1种解释,切片标准〈S39,current floor〉属于第2种解释.

定义(粗粒度程序切片).如果一种切片U的构造满足下列条件,则它是粗粒度程序切片:(1) U只包含主程序中的语句和一个个完整的方法;(2) U是一个完整的面向对象程序;(3) 如果某个方法M中的一条语句属于U,则M也属于U.

可见,粗粒度程序切片是指在计算面向对象程序的切片时,对每个类中方法的语句不再考虑(如果考虑,则称为细粒度程序切片),而把每个方法作为一个整体和类中的另外的一般语句同等对待.这样,我们得到的切片集合是由程序中影响某个切片标准的语句、谓词和方法组成的.也就是说,如果某个方法中的一条语句属于切片的集合,则整个方法都属于切片的集合.与通常意义下的细粒度程序切片相比,粗粒度切片具有下面的性质.

定理1.如果U和V分别表示关于同一个切片标准$\langle s,v\rangle$的粗粒度程序切片和细粒度程序切片,则$U\supseteq V$.

证明:对于任意语句$S\in V$,即S是影响v在s点值(状态)的一条语句,存在两种情况:如果S是主程序中的一条语句,则$S\in U$;如果S是某个方法M中的一条语句,因为$M\in U$,故$S\in U$.可见$U\supseteq V$. □

定理2.如果U和V分别表示关于同一个切片标准$\langle s,v\rangle$的粗粒度程序切片和细粒度程序切片,W表示所有方法或标准过程的关于切片标准$\langle s,v\rangle$的过程内切片的集合,X表示所有方法或标准过程中语句和谓词的集合,则$U=V+(X-W)$.

证明:对粗粒度切片集合中的每个方法或过程再进行过程内切片就得到细粒度切片. □

3.2 与细粒度切片的比较

显然,粗粒度切片和细粒度切片各有优缺点.粗粒度分析可以不考虑每个方法的细节,这使得计算切片的算法比较简单,为计算切片而需要建立的SSDG相对来说比一般的SDG要简单得多.

在不需要对程序进行详细分析或对面向对象的大型软件系统进行分析的情况下,使用粗粒度切片方法能够更快地理解程序的结构和各种依赖关系,从而提高了分析程序、理解程序的效率.粗粒度切片一般不太适合对较小程序的准确分析.由定理1可知,粗粒度切片的结果包含基于SDG的细粒度切片的结果,所以只会导致不精确,而不会导致错误.细粒度程序切片考虑面向对象程序里每个类中每个方法的细节,甚至要作参数分析,所以对一个较小的程序来说,分析的结果比较准确.如果是一个比较大的面向对象软件系统,由于SDG相当复杂,人工构造SDG和分析各种依赖关系(特别是数据依赖)几乎是不可能的.实际上,根据定理2,粗粒度切片可以通过对切片集中的每个方法再执行过程内切片(或方法切片)的办法转化成细粒度切片.例如,在用我们的算法处理每个方法时,加入处理过程内切片的算法就可以得到程序细粒度切片.

3.3 切片算法和实现

3.3.1 算 法

在SSDG上可以对文献[3]所提供的两阶段图形可达性算法进行简化,然后再计算关于上述两种意义的切片标准下的程序切片.

算法的第1阶段:从要考虑的程序点出发,沿着数据依赖边、控制依赖边、多态性调用边、调用边和多态性选择边反向遍历,标记遍历过程经过的结点.

算法的第2阶段:算法从第1阶段被标记的结点出发,沿着调用边正向下降到被调用的方法(或过程).

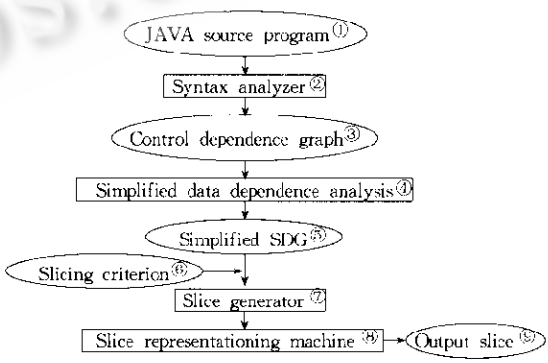
例如,利用两阶段图形可达性算法,我们可以计算切片标准为〈C39,current_floor〉的一个向后程序切片(向后切片,方法调用which_floor()),该切片包括所有可能影响which_floor的语句.在切片算法的第1阶段,从S39开始的遍历过程是沿着连接到S39的数据依赖边向后推进,找出影响S39值的那些语句、谓词和方法,并作标记.在算法的第2阶段,从所有在第1阶段标记过的结点出发,沿着调用边正向遍历,把所有遍历到的语句和方法作上标记.两阶段标记过的所有的语句、谓词和方法的集合就是所求的程序切片,如图2中阴影部分所示.这个程序中有一个多态性调用.切片算法将把来源于它的所有可能的目标的语句、谓词和方法都包括在切片中.

3.3.2 切片工具Javaslice简介

Javaslice的设计原则是:(1)有利于获得精确的切片结果;(2)有利于系统自身的研制;(3)界面管理直观,便于用户操作;(4)运算速度尽可能快;

在上述设计原则的指导下,本系统的设计思想可概括为两点:(1)基于语法分析的简化系统依赖图;(2)基于简化系统依赖图的切片生成算法.进一步来讲,根据JAVA语法规则,经过语法分析生成系统依赖图,再根据此结构生成相应的程序切片.

Javaslice的总体结构:Javaslice由4部分组成,即语法分析器、数据依赖分析器、切片生成器和切片表示器,如图3所示.



①JAVA源程序,②语法分析器,③控制依赖图,④简化数据依赖分析,⑤简化的系统依赖图,⑥切片准则,⑦切片生成器,⑧切片表示器,⑨输出切片.

Fig. 3 Javaslice architecture

图3 Javaslice体系结构

语法分析器(syntax analyzer):对JAVA源程序进行语法分析,找出所有的类、变量、函数的定义,分析IF,WHILE DO,SWITCH等控制语句,生成控制依赖图.

简化数据依赖分析(simplified data dependence analysis):进行部分语义分析,对赋值语句的左值/右值、控制语句的谓词、函数调用语句对实参和全局变量的影响以及类继承时的多态进行分析.

切片生成器(slice generator):根据给定的切片标准,依据简化系统依赖图,生成程序切片的中间表示.

切片表示器(slice representing machine):根据切片生成器产生的程序切片中间表示,用直观、简洁的方式将程序切片展现给用户.

Javaslice是一种基于简化系统依赖图(SSDG)的程序切片生成工具.它具有以下优点:(1) 切片生成准确,速度快,存储空间需求量小;(2) 界面简洁,切片表示清晰;(3) 符合面向对象的特点,支持多继承、多态等面向对象特色.

但是,本系统还处于实验阶段,仅能作为切片研究的辅助工具使用,尚不能达到作为软件工程中的通用工具的程度.另外,它在功能上还存在一些缺陷,例如,尚不支持动态切片标准等.所有这些,都将在我们今后的研究中得到改进.

4 结论与将来工作

程序切片是一种重要的程序分析技术,可以帮助程序员分析和理解程序.因为程序的一个切片实际上是该程序的一个初始版本,是对程序的一种特定的分析.程序切片移去程序中那些与分析无关的语法成分,使程序员或维护人员避免了不必要的影响.我们从面向对象程序的一个简化的系统依赖图(SSDG)出发,提出面向对象程序的粗粒度切片的概念和算法,成功地降低了利用SDG计算切片的复杂度,提高了分析大型面向对象软件的效率.同时,利用SSDG还可以计算对象切片、类层次切片以及动态切片等.今后,我们将进一步优化算法,完善SSDG的功能,并实现切片面向对象程序的一种环境:OOPSS——面向对象程序(C++/Java程序)切片子系统.

References:

[1] Weiser, M. Program slicing: formal, psychological and practical investigations of an automatic program abstraction method [Ph.D. Thesis]. Ann Arbor, Michigan: University of Michigan, 1979.

[2] Ottenstein, K.J., Ottenstein, L.M. The program dependence graph in a software development environment. ACM Software Engineering Notes, 1984,9(3):177~184.

[3] Horwitz, S., Reps, T., Binkley, D. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and System, 1990,2(1):26~60.

[4] Larsen, L.D., Harrold, M.J., Slicing object-oriented software. In: Douglas R.S. eds. Proceedings of the 18th International Conference on Software Engineering. New York: IEEE Computer Society Press, 1996, 495~505.

[5] Zhao, J.J. Dynamic slicing of object-oriented programs. Technical Report, SE-98-119, Information Processing Society of Japan, 1998, 17~23. http://www.fit.ac.jp/~zhao.

[6] Krishnaswamy, A. Program slicing: an application of object-oriented program dependency graphs. Technical Report, TR94-108, Department of Computer Science, Clemson University, 1994. http://www.clemson.edu.

[7] Tip, F. A survey of program slicing techniques. Journal of Programming Languages, 1995,3(3):121~189.

An Approach of Static Coarse-Grained Slice Based on Simplified System Dependence Graph*

LI Bi-xin[1,2], WANG Yun-feng[1], ZHANG Yong-xiang[1], ZHENG Guo-liang[1]

[1](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China*);

[2](*Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China*)

E-mail: lbx@seg.nuj.edu.cn; zhenggl@nju.edu.cn

http://www.seg.nju.edu.cn

Abstract: It is an efficient way to use SDG (system dependence graph) in slicing object-oriented program. But SDG is too complicated, so it may produce mistakes during constructing SDG, which will lead to inaccurate result. In this paper, the SSDG (simplified system dependence graph) is presented, which ignores nodes and edges representing parameter-in or parameter-out and summary edges. Meanwhile, the concept of coarse-grained slice for object-oriented program is defined, its properties are discussed, the relationships between coarse-grained slice and fine-grained slice are analyzed, the object-oriented coarse-grained program slice is computed based on simplified system dependence graph, and the implementation is also discussed.

Key words: system dependence graph; simplified system dependence graph; coarse-grained slice; static slice; object-orientation

* Received April 21, 1999; accepted December 3, 2000

Supported by the Youth Foundation of the National High Technology Development Program of China under Grant No. 863-306-QN2000-2; the Natural Science Foundation of Jiangsu Province of China under Grant No. BK99028