一种系统依赖图的面向对象扩充方案*

李必信[1,2], 李宣东[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: 提出一种对传统的系统依赖图进行面向对象扩充的方案.把传统的系统依赖图和类依赖子图、类层次子图相结合,从而构成了适合描述面向对象程序的面向对象系统依赖图.详细说明了对系统依赖图进行面向对象语法、语义扩充的过程,同时给出了构造面向对象系统依赖图的一般算法以及应用分析.

关键词: 系统依赖图;面向对象系统依赖图;构造算法;程序分析;程序切片

中图法分类号: TP311 **文献标识码**: A

在分析和理解结构化程序的过程中,我们借助各种程序依赖图来帮助分析和理解程序,这些依赖图形象而直观地描述了程序的结构以及各种依赖关系,从而提高了分析和理解程序的效率和准确性.然而,到目前为止,还没有一个比较完整的图形表示方法来准确地描述面向对象的程序.其原因是多方面的,主要原因是面向对象的诸多特征(如类、对象、继承性、多态性、动态定连等)影响了一种有效的面向对象程序表示的开发.在面向对象的程序中,类定义了它的实例(对象)将要处理的一些属性.一个类的属性包括实现对象状态的实例变量和在对象状态上实现操作的方法.我们经常设计、实现和测试类,却对它的调用环境不是很清楚.这样,在一种有效的用于面向对象软件的图形表示中,不仅要包括在构造其他类时能够重用的类的表示信息,而且还要包括使用该类的应用程序信息.通过继承关系可以定义子类,且允许子类继承父类的属性,或者按某种方式扩充、限制、重定义或替代父类的属性.像继承机制可以方便代码重用一样,类的一种有效的图形表示将方便分析信息的重用.在构造一个子类的表示时,应考虑到已经为父类计算的分析信息在子类中可以重用,无须重复计算,只须计算在了类定义的那些新信息.多态性是面向对象程序的又一个重要特征,它允许程序运行时可以在某个方法调用目标的可能集合中进行选择.可见,除非能够静态地确定其类型,否则这种动态选择的静态表示都要包括所有可能的目标.为了描述这些面向对象的特性,本文提出一种对传统的系统依赖图(简称 SDG)进行面向对象扩充的方案.通过把传统的系统依赖图和类依赖子图(class dependence subgraph,简称 ClDS)、类层次子图(class hierarchy subgraph,简称 CHS)相结合,从而构成了适合描述面向对象程序的面向对象系统依赖图.文中详细说明了对 SDG (system dependence graph)进行面向对象语法语义扩充的过程,同时给出了构造 OOSDG(object-oriented system dependence graph)的一般算法和应用分析.本文第 1 节分析了传统系统依赖图的

* 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,讲师,主要研究领域为面向对象设计技术及其支撑工具,程序理解;李宣东(1963-),男,湖南长沙人,博士,教授,主要研究领域为面向对象技术,实时系统;郑国梁(1937-),男,浙江桐乡人,教授,博士生导师,主要研究领域为软件工程,面向对象技术,形式化技术.

缺陷,第2节详细叙述了OOSDG对SDG的语法语义扩充、构造OOSDG的算法以及OOSDG的应用,第3节给出结论和将来的工作.

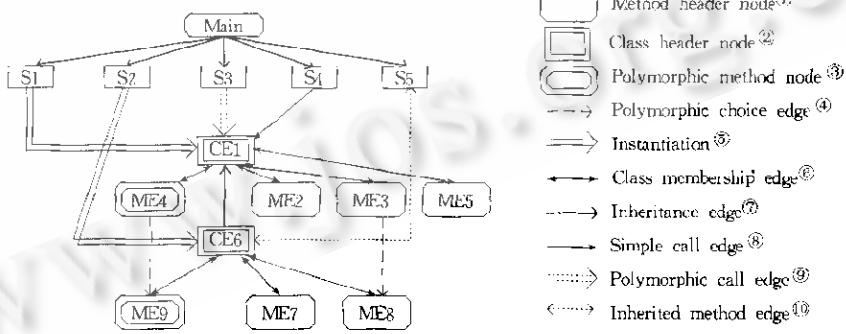
1 传统系统依赖图的缺陷分析

传统的系统依赖图[1,2]是在控制流图(control flow graph,简称CFG)、数据流图(data flow diagram,简称DFD)、控制依赖子图(control dependence subgraph,简称CDS)、数据依赖子图(data dependence subgraph,简称DDS)、过程依赖子图(procedure dependence subgraph,简称PrDS)和程序依赖图(program dependence graph,简称PDG)的基础上建立起来的一种语法分析树表示.结点表示程序构造(constructs)、输入/输出参数(in,out参数)和调用位置等.边表示与之相连的结点之间的各种依赖(例如数据依赖、控制依赖、声明等).形式地说,SDG是由一个程序依赖图和一组过程依赖图(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,但面向对象程序除了包含一些过程或方法以外,还存在继承、多态、动态定连等多种关系或特性,所以,传统的SDG所提供的机制还不足以描述这些面向对象的概念.例如,利用传统的SDG就很难表示如图1所示的C++程序.

```
CE1  class A{
      public:
ME2     A()
ME3     virtual~A();
ME4     virtual void do something();
ME5     void set_x(int)
     };

CE6  class B:public A{
      public:
ME7     B()
ME8     ~B();
ME9     void do something();
     };
```

```
main()
{
S1  A Aobj; //Instantiation
S2  B Bobj; //Instantiation
S3  Aobj.do something(); //Polymorphic call
S4  Aobj.set_x(int); //Simple call
S5  Bobj.set_x(int); //Inherited method
}
```



①方法首部结点,②类首部结点,③多态方法结点,④多态性选择边,⑤实例化,⑥类成员边,⑦继承边,⑧简单调用,⑨多态性调用边,⑩继承方法边.

Fig.1 A C++ program and its OOSDG representation
图1 C++程序及其OOSDG表示

由于这些传统的图形表示方法不能直接用来表示面向对象程序,故一般采取以下两种策略:

策略1. 引进新的适合表示面向对象程序的表示方法.如类层次子图(CHS)[3]可用来表示类之间的继承关系,可见方法类层次子图(visible-method-class-hierarchy-subgraph,简称VM-CHS)可用来静态地反映CHS中各结点的可见方法,虚函数调用图(virtual-function-call-graph,简称VFCG)可用来解决C++/Java语言中复杂的虚函数调用问题.

策略2. 改进已有的图形表示方法,扩充其功能,以使其具有表示面向对象程序的能力.如动态

面向对象程序依赖图(dynamic object-oriented dependence graph,简称 DODG)可用来表示面向对象程序的动态行为,面向对象的程序依赖图(object-oriented program dependence graph,简称 OPDG)[5]可用来表示面向对象语言中的多态性和动态定连问题,程序依赖网(program dependence net,简称 PDN)[6]可用来表示并发面向对象程序等.

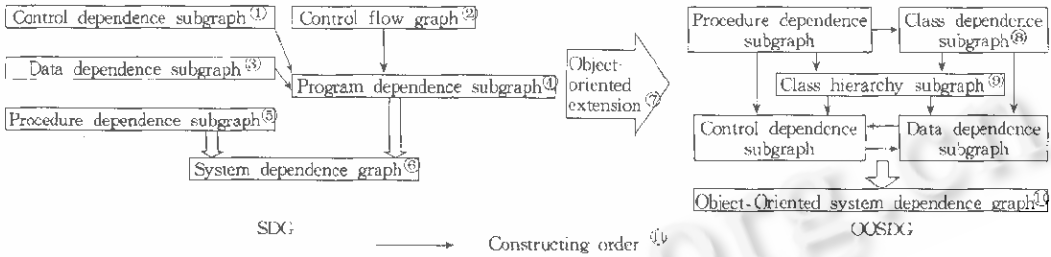
这些图形表示方法从不同的角度表示面向对象程序.它们基本上都是为某个目的而设计的,分别表示了面向对象程序的部分特性.例如,OPDG 不能用来表示面向对象的过程间行为,而 DODG 则只能表示系统的动态行为.要做到对面向对象程序的全面的分析和理解,必须有一种能全面描述面向程序静态特征的表示方法.为此,我们对 SDG 进行面向对象扩充,得到面向对象的系统依赖图(OOSDG).利用它来描述图 1 中 C++程序如图 1 下半部分所示,具体过程见后文说明.

2 面向对象系统依赖图——OOSDG

本节详细叙述了 OOSDG 对传统 SDG 的扩充过程,描述了 OOSDG 的基本组成模型、OOSDG 对 SDG 的语法语义扩充,提供了构造 OOSDG 的一般算法以及应用分析.

2.1 OOSDG 的基本组成模型

OOSDG 是过程依赖子图、类依赖子图、类层次子图、控制依赖子图和数据依赖子图这几种表示方法的并集.OOSDG 是对传统 SDG 的一种面向对象的扩充.SDG 和 OOSDG 的比较如图 2 所示.



①控制依赖子图,②控制流图,③数据依赖子图,④程序依赖子图,⑤过程依赖子图,⑥系统依赖图,⑦OO扩充,⑧类依赖子图,⑨类层次子图,⑩面向对象系统依赖图,⑪构造次序.

Fig. 2 The comparision of OOSDG and SDG

图2 OOSDG与SDG的比较

过程依赖子图:把一个过程表示成一个图,其中结点表示语句或谓词表达式,数据依赖边表示语句或表达式之间的数据流,控制依赖边表示一个语句或表达式执行时依赖的控制条件.每个过程依赖子图包含一个入口结点,表示过程的入口.为了模型化参数传递,OOSDG 为每个过程入口结点附上形参输入和形参输出结点.

类依赖子图:在不清楚调用环境的情况下,类依赖子图确定一个类中的控制依赖和数据依赖关系.在类依赖子图中,方法由过程依赖子图来表示.每种方法都有一个方法入口结点.类依赖子图也包含一个类入口结点,并通过类成员边把它和类中每个方法的入口结点连接起来.当一个类和另一个类或系统结合时,类入口结点和类成员边使我们能够很快访问方法的信息.

类层次子图:用图形来表示类间的继承关系,是方法和类定义的一种融合.类层次子图包含每个类的首部结点和在类中定义的每个方法的首部结点.类层次子图中的边把每个类的首部结点和与其具有继承关系的类的相应的首部结点连接起来.由方法首部表示的方法结点被连接到定义该方法的类的首部结点.子类中没有重新表示从超类中继承的方法,因而消除了对继承方法的重复表

示.因为类层次子图不表示方法的任何实现细节,所以不是一个程序的完全的静态表示.

控制依赖子图:每个方法的实现包含在这层表示中.因为这是一种静态表示方法,某些信息还不能够完全得到解决.例如,不能惟一地确定哪些消息的接收者将是动态定连的.这种表示会识别具有多态性的类的集合,而不是形式地定义接收者.这是一种完全的静态表示,它能够支持许多分析.

数据依赖子图:一个面向对象程序的运行环境比一个过程程序的运行环境要复杂得多.在数据依赖子图上,对象被加到表示中.这使得把消息动态地定连到对象中的特定方法得到完全解决.其他传统类型的数据依赖信息也得到表示,并为许多静态和动态分析提供支持.

2.2 OOSDG 对 SDG 的语法语义扩充

2.2.1 OOSDG 中的结点

在 OOSDG 中,需要用到许多结点.下面分别给出它们的语法解释.

• 首部结点:代表一类可表示实体的开始.这些可表示实体的语法类似于真实代码.(1) 类首部结点:一个类首部表示类中所有方法和数据属性的组合,参与表示类间的继承层次.类首部包含一列定义在该类中的方法和指向类的所有被继承方法的指针,且和所有的超类或子类连接.类首部还包含类的数据成员的一些信息.(2) 方法/过程首部结点:一个方法首部或过程首部是相应方法或过程的入口结点.方法首部通过封装信息(例如,定义该方法的类、类的数据属性以及从实参到形参的参数转换等,反之亦然)向表示中溶入另外的含义.方法首部也可构成某个类成员关系边的一个终点.(3) 虚方法首部结点:有些面向对象语言允许类的所有方法都是动态定连的.相反,C++是通过在方法声明的前面加上关键字 virtual 来规定该方法是动态定连的.我们用类似的方法来区分可以动态定连或不可以动态定连的方法.这使得理解多态性和动态定连的表示更加容易.一个虚方法是一个类的特定成员,如果对此方法的一次调用是多态的,它会得到动态的解决.所以,用一种增强的表示方法来表示虚方法首部.虚方法首部和多态性选择边在清晰地表示动态定连时起到很重要的作用.其他的与方法首部类似.

• 语句结点.(1) 基本语句结点.基本语句结点表示的语句,可以完成部分计算,且不会导致对一个方法或过程的调用,或者是从过程返回或退出.(2) 调用结点.一个调用结点表示在某个调用位置对方法或过程的一次调用.对 OOSDG 中的不同的调用边(也就是简单调用边、实例边和多态性调用边)来说,这是开始点.(3) 过程返回或退出结点.过程返回或退出结点表示在该条控制流路径中的最后一条被执行的语句.

• 其他结点.(1) 谓词结点.谓词结点表示条件语句中的谓词部分.(2) 域结点.一个域结点用相同的控制依赖来组合语句.(3) 参数结点.参数结点用来表示实参和形参.对每个输入参数(实参和形参)都有一个参数结点与它对应.如果某个参数是可修改的(又称输出参数),则为每个实输出参数或形输出参数也设置一个结点.

2.2.2 OOSDG 中的边

同样地,在 OOSDG 中,还用到多种边.下面给出所有边的语法定义.

• 控制边.(1) 流边.流边是单方向的,用来描述程序中清晰的控制流.隐含的控制流可按逆时针的记号循序来表示.(2) 依赖边.这是一些在表示中显示结点之间控制依赖关系的单向边.这里要注意的是,这些控制依赖边的起点总是域、谓词或方法/过程结点.(3) 调用边.有 3 类调用边.① 简单调用边,表示一次对方法或对自由标准过程的调用.简单调用边表示能够被静态解决的调用,表示调用位置和被调用的方法(过程)之间的控制流和数据流.② 实例化调用边.在面向对象的

程序中,实例化意味着创建了一个类的实例.这是靠调用一个类的构造函数来完成的,类的构造函数初始化对象状态并把它带进系统.用一条特定的边来表示实例调用的重要性.实例边把实例语句和类首部(它的首部实例被建立)连接起来.③ 多态调用边.如果在编译时不能解决对一个特定方法的调用,则称这种调用是动态定连的.在表示多态性和动态定连时要使用多态调用边.多态调用边是调用位置和类的类首部结点之间的一条边.(4) 多态性选择边.这种边起始于超类中的虚方法首部结点,终止于子类中具有相同规约的另一个虚方法首部结点.对一个超类虚方法的多态调用能够从该层开始,自上而下地在层次中的任何虚方法处得到解决.

· 成员边.(1) 继承边.继承层次是面向对象设计方法的骨架.继承边表示一个继承层次中类之间的关系,并按依赖的方向连接子类和它的超类.(2) 类成员边.每个定义的类中都有固定数目的方法,这样,每个方法(包括该类的构造函数和析构函数)都属于那个类,且只能通过那个类的实例才可以访问(寻地址).一条类成员边连接方法首部结点和定义该方法的类首部结点.(3) 表示继承的方法边.如果在超类中定义的一个方法被继承到子类中,则称此方法为继承的方法.(4) 概括边(summary edge),表示输入参数和输出参数、实输入参数和形输入参数、形输出参数和实输出参数之间的数据可传递关系.

2.3 基本问题的解决策略

调用边.调用边包括简单调用边、继承方法调用边和多态性调用边 3 种.调用边的终止点是类首部结点而不是方法首部结点.使用这种办法来表示动态定连,可以解决继承方法的调用问题.类首部存储了有关成员函数、继承的方法、在类层次中的位置以及数据属性等方面的信息.在 OOSDG 中包含了用于解决一个方法调用的动态定连问题的所有必要信息.

参数处理.面向对象程序中的消息包含用来交换信息的参数,因此参数处理是我们要解决的又一个问题.传统的处理参数流的技术是在调用位置和被调用过程之间附加一条边,用来连接实参和相应的形参.因为在面向对象系统中有大量的消息需要交换,这种技术由于边数过多会使得表示混乱且难以控制.使用这些附加边是为了描述实参和形参之间的信息流.我们扩充调用边的含义使其可以从两个方向上表示参数流,也就是从调用位置到方法或过程,或者反过来.这样就减少了实参和形参之间的边数,从而使表示更为清晰.我们通过在调用位置和方法或过程首部结点的参数结点来表示输入参数,利用调用位置和方法或过程首部结点最右端的参数结点来表示可以被修改的参数(输出参数),在实参位置的参数结点和相应的形参位置的参数结点之间定义一个一一映射,这样就省略了实参和形参结点之间的边,使得表示更加直接并且更易于理解.

多态性.在面向对象语言中,一个多态性调用可能涉及到多个类的实例.这样,一个多态性调用将具有静态和动态两种类型.在 OOSDG 中,使用多态性调用边、虚方法首部结点和多态性选择边来清晰地表示多态性.一条多态性调用边是一条表示方法调用的特殊的边.这样,调用在类层次中某个类及其向下的任何类中可以用同样的规约定连到一个方法的实现.在对运行环境的静态表示中,通过把一列虚方法附着到多态性调用边上来表示这些特性.一旦多态性调用在运行时得以解决,就可以使用实际的调用来访问这列方法中的某个方法,并且执行这个方法.

类.(1) 表示基类:利用 ClDS 来表示系统中的每个类.ClDS 能够在不知道调用环境的情况下捕获某个类中控制和数据依赖关系.ClDS 中每个方法由 PrDS 来表示,每种方法都有一个方法入口结点来表示进入一个方法的入口.ClDS 也包含一个类入口结点,可以通过类成员边把它和类中每个方法的入口结点连接起来.当一个类和另外一个类或系统结合时,类入口结点和类成员边使我们能够很快地访问方法信息.ClDS 扩充了每个方法入口,使其包含形参输入和输出结点.因为一个

类的实例变量对类中的每个方法来说都是可访问的,所以把它们当作类中所有方法的全局变量.同时,为所有在方法中调用的实例变量设置了参数输入和输出结点.(2) 表示衍类:通过为衍类中每个定义的方法构造表示以及重用所有从基类继承来的方法的原始表示来构造一个衍类的 ClDS.我们为每个衍类建立一个类入口结点,并且加入类成员边来连接衍类入口结点和所有在衍类定义中定义的每个方法的方法入口结点.我们也建立一种类成员边来连接类入口结点和任何在基类中定义而被衍类继承的每个方法的入口结点.

相互作用类.在面向对象的软件中,一个类可以通过声明或使用操作 new 来实例化另一个类.当类 C1 实例化类 C2 时,就会存在一个对 C2 的构造函数的潜在的调用.为了表示这种潜在的构造函数的调用,我们在构造 ClDS 时,在 C1 中实例化的位置加入一个调用结点.用一条调用边把这个调用结点和 C2 的构造函数方法连接起来.我们构造 ClDS 时,在调用位置加入实参输入结点和实参输出结点,以便与 C2 的构造函数中形参输入和形参输出结点.

完全程序:通过连接不完全系统依赖图中每个 ClDS 的方法调用的办法来构造一个完全面向对象程序的 OOSDG.它把所有的调用结点和方法入口结点、实参输入结点和形参输入结点以及形参输出结点和实参输出结点连接起来.在调用位置把分析过的类中的方法的概括边连接到实参输入和实参输出结点之间.一个面向对象系统的 OOSDG 构造最大限度地重用先前已经构造的部分表示.在应用程序范围引入的变量(例如全局变量)不会影响任何 ClDS 的表示.任何一个被类调用或修改的变量必须在类中声明为 extern.因此,在建立一个类的 ClDS 时将包含这种信息.

2.4 OOSDG 的构造算法

因为 OOSDG 由 PrDS,ClDS,CHS,CDS 和 DDS 这 5 个部分组成,所以,我们的构造 OOSDG 的算法是从构造类中方法的 PrDS 开始的,按 CHS 的拓扑结构从底到顶的顺序依次构造 ClDS,CDS 和 DDS,从而完成对 OOSDG 的构造.由于 CHS 的构造很简单——由类和继承关系可以直接获得 CHS,所以,我们的算法中不包括 CHS 的构造.下面给出构造 OOSDG 的算法,并给出简要说明.算法的输入是每个过程或方法的 CFG 和 DFG.根据类层次的拓扑结构,按照从底到顶的风格,为每个类 C 的方法建立过程依赖子图.为了计算 C 中方法的 PrDS,算法首先识别哪些方法需要重新建立 PrDS;然后为在 C 中声明的每个方法 m 构造 PrDS.对每个在基类中声明的方法 m,如果 m 直接或间接调用了一个在 C 中重定义的虚方法,算法就会在为基类中方法 m 建立的 PrDS 的基础上来构造 m 的新的 PrDS;如果 m 没有直接或间接调用在 C 中重定义的任何方法,则算法重用为基类中的 m 而建立的 PrDS.在为每个方法构造了 PrDS 以后,算法在此基础上进一步构造 ClDS,CDS 和 DDS(实际上,DDS 可以通过在 CDS 的结点之间加入数据依赖边的办法得到).然后,算法调用 Connect()函数,并使用定连边(binding edges)把 PrDS 中的每个调用位置和被调用方法的入口结点连接起来.构造 OOSDG 的算法的最后一步是计算调用位置的概括边(summary edges),连接形参输入结点和形参输出结点以及实参输入结点和实参输出结点.

构造算法

输入:程序 P 及其控制流图和数据流图

输出:程序 P 的 OOSDG

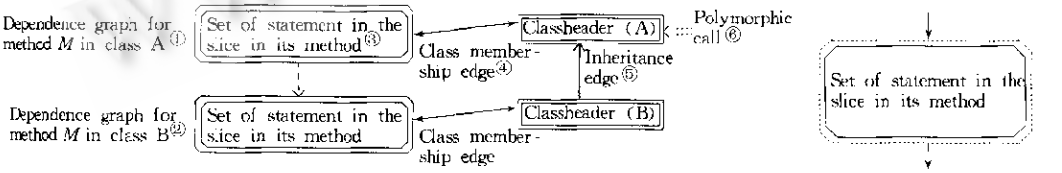
```
Start ConstructOOSDG()
/* 先建立程序 P 的类层次子图(CHS),按 CHS 的拓扑结构,以从底到顶的次序建立 PrDS,ClDS,CDS 和
   DDS.下面的算法假定已经建立了 CHS */
1  for 1 对每个类 C∈CHS
2     确认 C 中需要新表示的方法
```

```
3    for 2 对每个在 C 中声明的方法 m
7      为 m 计算控制依赖;
8      使用通常的数据流分析预处理 m;
9      用 m 扩充对象;
10     为所有对象计算数据依赖;
11     为每个方法建立过程依赖子图(PrDS);
12   endfor 2
13   for 3 对每个基类中方法 m
14       If 如果 m 是"做过标记的",则
15         复制旧的过程依赖路径;
16         调整调用位置;
17       else 否则
18         重用 m 的老的过程依赖路径;
19       endif
20   endfor 3
21  为每个类建立类依赖图(CIDS);
22 endfor 1
23 建立控制依赖子图(CDS);
24 建立数据依赖子图(DDS);
25 调用函数 Connect();
26 调用函数 buildsummaryEdges();
End ConstructOOSDG
```

2.5 OOSDG 的一个应用实例

由于多态性引用的存在,面向对象程序的切片可以分别以编译时观点和运行时观点来考虑.如果所有的引用能够在编译时得到解决,则面向对象程序的编译时切片和运行时切片是类似的.反之,同一个语句-变量对的组合就可能有多个切片与之对应.在 OOSDG 中,我们利用一种特定的边和顶点来表示多态性引用和多态性选择,这使得多态性切片的表示更加紧凑.当存在多态性切片时,多态性选择边是切片的一部分.可以通过跟踪连接某个方法的多态性选择边的办法来获得切片.

考虑如图 3 所示的例子.编译时,由于没有解决对方法 M 的多态性引用,我们得到 M 中语句的多个切片.在运行时,由于把对方法 M 的多态性引用解决为对某个特定对象的引用,从而消除了这种模棱两可的情况.切片包含了解决了引用问题的对象中的所有语句.实际上,可以利用 OOSDG 中的数据依赖子图确定惟一的切片,因为数据依赖子图可以用来识别对象实例以及其他运行时实体等.



(a) Compile time polymorphic slice
(a) 编译时多态性切片

(b) Run-Time single slice
(b) 运行时被减少为单个切片

①类A中方法 M 的依赖图,②类B中方法 M 的依赖图,③切片包含该方法的语句集合,④类成员边,⑤继承边,⑥多态性调用.

Fig. 3 Compile-Time polymorphic and run time-single slice
图3 编译时多态性切片和运行时单个切片

3 结论与将来的工作

我们构造OOSDG的主要目的是为了便于进行面向对象的程序分析、程序理解和程序切片.但是我们的OOSDG表示方法也可用在面向对象程序的优化、测试、调试、维护以及逆向工程等方面.我们下一步的工作就是在OOSDG的基础上实现一组面向对象的程序切片技术,通过这些切片技术,分别对面向对象程序中的语句、对象、对象的状态及其行为和类层次进行切片.另外,还需进一步扩大OOSDG的功能,一方面使其可以描述并发面向对象和分布式面向对象程序;另一方面,在OOSDG上实现一些面向对象程序的动态切片、条件切片以及无定型切片技术.

References:

[1] Harrod, M. J., Malloy, B., Rothermel, G. Efficient construction of program dependence graphs. ACM International Symposium on Software Testing and Alalysis, 1993,18(3):160~170.

[2] Horwitz, S., Reps, T., Binkley, D. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and System, 1990,12(1):26~60.

[3] Dean, J., Grove, D., Chamber C. Optimization of object-oriented programs using class hierarchy analysis. In: Olthoff, W. ed. Proceedings of the 9th European Conference on Object-Oriented Programming (ECOOP'95). Heidelberg, Germany: Springer-Verlag, 1995. 77~101.

[4] Zhao, 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.

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

[6] Zhao, J., Cheng, J., Ushijima, K. Static slicing of concurrent object-oriented programs. In: Proceedings of the 20th IEEE Annual International Computer Software and Applications Conference. IEEE Computer Society Press, 1996. 312~320.

A Scheme for Extending System Dependence Graph Based on Object Orientation*

LI Bi-xin[1,2], LI Xuan-dong[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: lbxin@mail.hf.ah.cn; zhenggl@nju.edu.cn

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

Abstract: In this paper, a scheme for extending traditional system dependence graph based on object orientation is presented, i.e., an object-oriented system dependence graph (OOSDG) suitable for object-oriented program is constructed by combining SDG with CIDS (class dependence subgraph) and CHS (class hierarchy subgraph). The extension of syntax and semantics and function of SDG are discussed. Meanwhile, the algorithm for constructing OOSDG is provided, and application aspect is also analyzed.

Key words: system dependence graph; object-oriented system dependence graph; constructing algorithm; program analysis; program slicing

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

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. BK99038