一种面向对象程序的分层切片方法*

李必信，刘小东，郑 滔，李宣东，郑国梁

(南京大学 计算机软件新技术国家重点实验室，江苏 南京 210093)；
(中国科学技术大学 计算机科学技术系，安徽 合肥 230056)
E-mail：zhenggl@nju.edu.cn；lbxin@mail.hf.ah.cn
http://www.nju.edu.cn

摘要：程序切片是一种程序分析技术，广泛应用于程序的调试、测试、理解和维护等软件开发的各个阶段.在研究静态类型面向对象的程序切片时发现，利用系统依赖图计算程序切片虽然是一种有效的方法，但构造OO程序的系统依赖图是一件非常复杂的工作，而且构造过程中容易出错，这会导致切片的结果不正确，造成前功尽弃.为此，从程序逻辑分层的角度提出一种OO程序的层次模型，然后在OO程序的层次模型基础上采用逐步求精算法来分层计算OO程序的切片.

关键词：层次模型；分层切片；逐步求精；类层次结构切片
中图法分类号：TP311 **文献标识码**：A

面向对象的程序切片是调试、测试、分析、理解和维护面向对象软件的一种重要手段.很多学者投身到这一领域的研究之中.例如，Zhao Jian Jun利用一种动态面向对象的依赖图(dynamic object-oriented dependence graph，简称DODG)和系统依赖图网(system dependence net，简称SDN)，分别解决面向对象程序的动态切片问题和并发面向对象的程序切片问题[1,2]；A. Krishnaswamy利用一种面向对象的程序依赖图(object-oriented program dependence graph，简称OPDG)来计算面向对象程序的语句切片[3]；L. D. Larsen和M. J. Harrold利用系统依赖图计算OO(object-oriented)程序的对象切片和整程序切片[4].另外，Frank Tip，Christoph Steindl，R. C. H. Law以及东南大学的徐宝文等人在面向对象的程序切片领域都作出了不可忽视的贡献[5~9].但这些切片方法都是基于依赖图的，而构造OO程序的依赖图是一件非常复杂的工作，而且构造过程中容易出错，这会导致切片的结果不正确，造成前功尽弃.为此，我们从程序逻辑分层的角度提出一种OO程序的层次模型，然后在层次模型的基础上采用逐步求精算法来分层计算OO程序的切片.

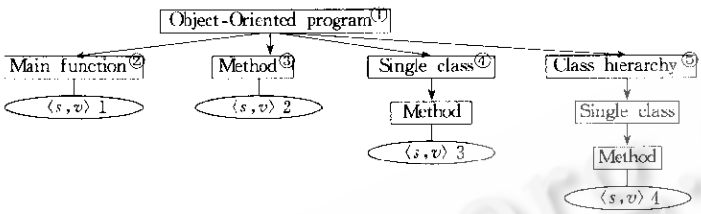
本文第1节建立了OO程序的层次结构模型.第2节讨论了面向对象切片的准则、分层切片方法的思想和算法以及类层次切片的概念和实现.第3节详细阐述了分层切片的应用.第4节给出结论并提出将来的研究工作.

1 OO程序的层次模型

现在，我们来建立一种适合分层切片思想的层次模型.该模型把面向对象的软件按逻辑结构分

* 收稿日期：2000-04-12；修改日期：2000-07-12
基金项目：国家863青年基金资助项目(863-306-QN2000-2)
作者简介：李必信(1969－)，男，安徽庐江人，博士，主要研究领域为面向对象支撑技术及其环境和工具，程序理解；刘小东(1975－)，男，安徽蚌埠人，硕士，主要研究领域为程序理解和程序切片；郑滔(1966－)，男，浙江桐乡人，副教授，主要研究领域为软件工程，形式化技术；李宣东(1963－)，男，湖南邵东人，博士，教授，主要研究领域为面向对象技术，形式化技术；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.

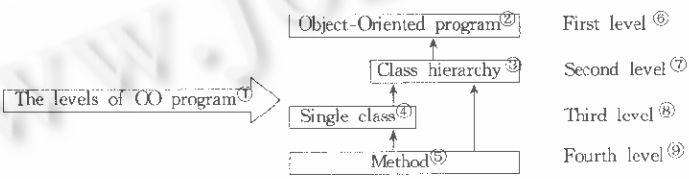
为4个层次:OO程序层表示一个完整的程序,它包含main函数、自由标准过程(这里都称为方法)、单个类以及类层次几个方面;类层次结构表示通过继承关系形成的一组相互作用的类;单个类层只含有单个的类;方法层表示自由标准过程或类中定义的成员函数(如图1和图2所示).其中图1是OO程序的结构模型,图2是由此映射的层次模型.



①OO程序,②Main函数,③方法,④单个类,⑤类层次.

Fig. 1 The structure model of OO program

图1 OO程序的结构模型



①面向对象程序的层次,②OO程序,③类层次,④单个类,⑤方法,⑥第1层,⑦第2层,⑧第3层,⑨第4层.

Fig. 2 The hierarchy model of OO program

图2 OO程序的层次模型

2 分层切片方法

在面向对象的静态程序切片过程中,切片准则⟨s,v⟩有两种不同的解释:

第1种解释是,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,此时把⟨s,v⟩记为⟨s,v⟩**(图1中切片准则⟨s,v⟩1,⟨s,v⟩2属于这种解释);

第2种解释是,v和s分别是某个类的某个方法中的一个变量和一条语句,此时把⟨s,v⟩记为⟨s,v⟩*(图1中的切片准则⟨s,v⟩3,⟨s,v⟩4属于这种解释).

第1种情况下的变量v既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.第2种情况下的变量v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.

在计算面向对象切片时,为了在处理方法上取得共性,我们把主程序和程序中所有其他自由标准过程都看成是在一个虚类中的方法来处理.这样,在系统中的所有过程体都是某个类的方法,它们具有相等的地位.在一个面向对象的程序中,可以假设没有全局变量(如果遵循了好的设计风范).即使存在全局变量,也可以把它们当作参数,以同样的方法来处理,因此,没有必要专门对它们进行讨论.

基于层次模型和切片准则,得到如下分层切片方法的主要思想,并在此基础上进一步实现面向对象的分层切片算法.由于这里的分层切片方法是采用逐步求精的算法思想来逐步计算程序切片的,故又称逐步求精算法.

2.1 分层切片方法思想

考虑切片准则⟨s,v⟩**,其中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.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程序的切片方法.这里,我们使用分层切片方法来讨论如何计算关于切片准则$\langle s,v\rangle^{**}$的程序切片.分层切片方法的主要思想是:

(1) OO 程序层.在计算与切片准则$\langle s,v\rangle^{**}$有关的程序切片时,首先确定所有与包含v和s的类有关(由继承、消息传递、数据依赖或控制依赖等形成的直接或间接关系)的类、方法、main 函数中的语句等,删除那些与包含v和s的类无任何关系的类、自由标准过程、main 函数中的语句以及控制谓词等.然后把全局变量、自由标准过程、单个类以及类层次等都作为同等的单个实体来考虑.也就是说,如果一个类层次中某个类的某个方法中的某条语句影响切片准则或受切片准则影响,则把该类包含到切片集合中.否则,在进行下一步切片时不予考虑.通过这种方法得到关于切片准则$\langle s,v\rangle^{**}$的程序切片的一个粗略版本$S'(P)$.

(2) 类层次结构层.对$S'(P)$中的类层次进行逐步求精,进一步计算该类层次中影响切片准则的类,并把不影响切片准则或不受切片准则影响的类从类层次结构中删除,从而得到 OO 程序切片的一个精化版本$S''(P)$,并且满足$S'(P)\supseteq S''(P)$.

(3) 单个类层.在上层基础上进一步计算单个类中影响切片准则或受切片准则影响的变量、语句和方法,删除那些与切片准则无关的变量、语句和方法等,然后得到$S'''(P)$,且有$S'(P)\supseteq S''(P)\supset S'''(P)$.

(4) 方法层.在前面工作的基础上进一步计算过程或方法中与切片准则有关的变量和语句,删除那些与切片准则无关的变量、语句和控制谓词等,最后得到$S(P)$,同时满足$S'(P)\supseteq S''(P)\supset S'(P)\supseteq S(P)$.至此,我们得到一个面向对象程序关于切片准则$\langle s,v\rangle^{**}$的程序切片.

考虑切片准则$\langle s,v\rangle^{*}$,其中v和s分别是某个类的某个方法中的一个变量和一条语句.切片的方法基本上与上面讨论的类似.只是如果切片准则是在一个类层次结构的某个类的某个方法中,这就要先计算类层次结构切片,再在类层次切片基础上逐步扩展,从而得到整个程序的切片.

2.2 逐步求精算法

一个程序P的静态切片是由程序P中的一些语句和谓词组成,这些语句和谓词影响在兴趣点s定义或使用的变量v的值(又称静态后向切片),或者一个程序的静态切片是由那些受s点变量v的值影响的所有语句和谓词的集合(又称静态前向切片).我们在计算程序P的切片时,把 main 函数、自由标准过程/方法、单个类以及类层次作为一个整体来处理.这样,在系统中的所有过程体、单个类、类层次等都具有相等的地位.

计算程序P的关于某个切片准则$\langle s,v\rangle$的程序切片,实际上是计算一个程序$S(P)$,使得$S(P)$和P关于准则$\langle s,v\rangle$具有同样的执行行为.如果令程序$P=(\text{Main},M,O,CH)$,其中 Main 表示 main 函数,M表示P中所有自由标准过程或类中的方法的集合,O表示P中所有单个的对象或类的集合,CH表示P中所有类层次结构的集合.显然,计算程序P的切片就是计算$S(P)$.在这里,计算$S(P)$的过程实际上是一个对P进行逐步求精的过程,也就是分别对程序中的类层次、对象或类、过程或方法分别进行求精的过程.

因为程序本身就是该程序关于任何一个准则的切片,故可令$P=S^0(P)=S^0((\text{Main},M,O,CH))$,即$S^0(P)=S^0(\text{Main})\cup S^0(M)\cup S^0(O)\cup S^0(CH)$.根据分层切片的一般描述,有下面的性质:

性质 1. 分层切片过程是一个逐步求精过程.求精过程满足下列规则:

(1) $P=S^0(P)\supseteq S^1(P)\supseteq S^2(P)\supseteq\ldots\supseteq S^n(P)=S(P)$;

(2) $\text{Main}=S^0(\text{Main})\supseteq S^1(\text{Main})\supseteq S^2(\text{Main})\supseteq\ldots\supseteq S^n(\text{Main})=S(\text{Main})$;

(3) $M=S^0(M)\supseteq S^1(M)\supseteq S^2(M)\supseteq\ldots\supseteq S^n(M)=S(M)$;

(4) $O=S^0(O)\supseteq S^1(O)\supseteq S^2(O)\supseteq\ldots\supseteq S^n(O)=S(O)$;

(5) $CH=S^0(CH)\supseteq S^1(CH)\supseteq S^2(CH)\supseteq\ldots\supseteq S^n(CH)=S(CH)$.

因为 $S(P)=S((\text{Main},M,O,CH))$,如果把对 main 函数的求精结果记为 $S(\text{Main})$,把对自由标准过程或方法集合 M 的求精结果记为 $S(M)$,把对单个类集合 O 的求精结果记为 $S(O)$,把对类层次集合 CH 的求精结果记为 $S(CH)$,则有以下公式成立:

$$S(P)=S(\text{Main})\cup S(M)\cup S(O)\cup S(CH).$$

可见,可以把计算程序 P 的切片过程化分为几个部分,即计算 main 函数的切片 $S(\text{Main})$,计算自由标准过程/方法 M 的切片 $S(M)$、计算单个类集合 O 的切片 $S(O)$以及计算类层次 CH 的切片 $S(CH)$. 计算 main 函数的切片和计算自由标准过程/方法 M 的切片较简单,实际上可以采用过程内切片的方法来计算. 如果类层次切片解决了,单个类的切片问题也就迎刃而解了. 所以,在利用分层切片方法计算面向对象程序的切片时,主要就是解决类层次结构切片问题. 为此,下面,我们专门讨论类层次切片.

2.3 类层次切片

因为一个面向对象程序被看成是一个四元组 $P=(\text{Main},M,O,CH)$,所以前期关于切片的工作可用来计算面向对象程序的切片. 也就是说,给定一个程序 $P=(\text{Main},M,O,CH)$和一个切片准则,可以计算一个语句切片 $S(P)=S((\text{Main}),M,O,CH))$,它和源程序在考虑到切片准则时具有同样的执行行为.

本节工作的目标就是计算类层次切片:给定一个程序 $P=(\text{Main},M,O,CH)$(它可能是某个语句切片算法的输出,但不一定必须),我们来计算一个与 $P=(\text{Main},M,O,CH)$具有同样执行行为的切片 $P'=(\text{Main},M,O,CH')$,这里,CH'是 CH 的子层次(定义见后). 在特别情况下,CH 仅由 CH'的那些必须在 P'的执行过程中为保证 $\Sigma(CH')$(CH'中包含的所有子对象的集合,定义见后)包含可以被实例化的所有子对象的部分构成. 然而,我们也必须保证用 CH'替代 CH 不改变 P'的执行行为. 程序执行行为中依赖于类层次的主要组件(primary component)是成员查询操作(member lookup operation). 因此,类层次切片应该包含任何在 P'的执行过程中完成成员查询的结果. 为了进行类层次切片,先简单介绍 Rossie-Friedman 的类层次框架[5,10].

令 C 是程序 P 中所有类名的集合,M 是程序 P 中所有方法名的集合,nI 是所有非虚继承关系的集合,vI 表示所有虚继承关系的集合,v 表示函数的虚实性. 则有下面的定义.

定义 1(类层次). 一个类层次 CH 是一个四元组 $CH=\langle\hat{C},\hat{M},nI,vI\rangle$,其中

(1) $\hat{C}\subseteq C$,

(2) $\hat{M}\subseteq\hat{C}\times M\times v$,

(3) $vI\subseteq\hat{C}\times\hat{C}$,

(4) $nI\subseteq\hat{C}\times\hat{C}$,

使得以下各式成立:

(i) vI 和 nI 是非自反的,

(ii) $vI\cap nI=\varnothing$,

(iii) $vI\cup nI$ 的传递闭包是反对称的,

(iv) 如果$\langle X,m,f_1\rangle\in\hat{M}$且$\langle X,m,f_2\rangle\in\hat{M}$,则$f_1=f_2$,

(v) 如果$\langle X,m,\text{“}virtual\text{”}\rangle\in\hat{M},\langle Y,m,f\rangle\in\hat{M}$且$\langle Y,X\rangle\in(vI\cup nI)$,则$f=\text{“}virtual\text{”}$.

这里,$\hat{C}$表示在该类层次中使用的类名的子集.$\hat{M}$是一个函数,它把$\hat{C}$中的每个类映射到方法名集合的一个了集,且定义类成员是虚的或不是虚的.vI和nI分别表示C中类之间的非虚继承和虚继承关系.vI和nI的自反、传递闭包记为$I=(nI\cup vI)$.约束条件(i)~(v)解释了通常的C++语言关于继承关系的约束.即(1) 一个类不可能是它自己的基类,(2) 在同一个时间,一个类不可能既是虚基类又是非虚基类,(3) 在继承图中不允许出现环,(4) 一个成员在同一个时间不可能既是虚的又是非虚的,(5) 一个虚成员不可能被另一个非虚成员用相同的名字来改写.由于实际原因,假设类层次是有限的.

在Rossie-Friedman的类层次框架的基础上,我们可以进一步定义子对象,并根据文献[10]采用的子对象识别算法来识别每个类结点的子对象同时利用方法查询算法来查询每个子对象中包含的方法,再利用其结果来达到计算类层次切片的目的.

定义2(子对象). 对一个类层次CH和一个类$A\in\hat{C}$,A的子对象集合记为$\Sigma(CH,A)$可以直观地定义如下:

(1) $[A,A]\in\Sigma(CH,A)$;

(2) $[A,B]\in\Sigma(CH,A)$如果存在一个X使得$\langle A,X\rangle\in nI(CH)$和$\langle X,B\rangle\in v\hat{I}(CH)$;

(3) $[A,\alpha\cdot X\cdot Y]\in\Sigma(CH,A)$如果$[A,\alpha\cdot X]\in\Sigma(CH,A)$且$\langle X,Y\rangle\in n\hat{I}(CH)$.

集合$\Sigma(CH)$表示在CH中的所有类的子对象的集合.如果$[A,\alpha\cdot X]\in\Sigma(CH)$,则$mdc([A,\alpha\cdot X])\triangleq A$表示类层次中某条路径的开始类结点,$ldc([A,\alpha\cdot X])\triangleq X$表示类层次中某条路径的结束类结点.

定义3(<). 令CH是一个类层次,$A\in\hat{C}(CH)$,则

$$[A,\alpha]<_{nI,CH}[A,\alpha\cdot X],$$

$$[A,\alpha\cdot X]<_{vI,CH}[A,Y],\text{如果}\langle X,Y\rangle\in v\hat{I}(CH).$$

进一步地,令$<_{CH}=<_{nI,CH}\cup<_{vI<CH}$且令$\leqslant^*_{CH}$是$<_{CH}$的传递和自反闭包.

更进一步地,对任何$\Sigma\subseteq\Sigma(CH,A)$,$\min\limits_{CH}(\Sigma)$表示$\Sigma$中与$\leqslant^*_{CH}$有关的最小元素,如果从上下文能够很明显地得到类层次CH,就可以省掉$\min\limits_{CH}(\Sigma)$中的下标CH.

定义4(子对象识别). 令CH是一个类层次,$A\in\hat{C}(CH)$,$\sigma\in\Sigma(CH,A)$,且令m是M中的一个成员名,则具有在σ可见的方法m的类A的所有子对象集合是

$$VisibleDefs(CH,\sigma,m)\triangleq\begin{cases}\{\sigma\} & \text{如果 } m\in\hat{M}(ldc(\sigma))\\ \bigcup\limits_{\sigma<\sigma'}VisibleDefs(CH,\sigma',m) & \text{如果 } m\notin\hat{M}(ldc(\sigma))\end{cases}.$$

定义5. 令CH是一个类层次,$A\in\hat{C}(CH)$,$\sigma\in\Sigma(CH,C)$,且令m是M中的一个成员名,则

$$\text{static-lookup}(CH,\sigma,m)\triangleq\min(VisibleDefs(CH,\sigma,m)),$$

$$\text{dynamic-lookup}(CH,\sigma,m)\triangleq\min(VisibleDefs(CH,[mdc(\sigma),mdc(\sigma)],m)).$$

定义6(方法查询). 令CH是一个类层次,$A\in\hat{C}(CH)\sigma\in\Sigma(CH,A)$,且令$m$是$M$中的一个成员名又令$\sigma'=\text{static-lookup}(CH,\sigma,m)$,且令$\langle ldc(\sigma'),m,v\rangle\in\hat{M}(CH)$,则

$$\text{lookup}(CH,\sigma,m)\triangleq\begin{cases}\sigma' & \text{如果 } v=\text{“}non\text{-}virtual\text{”}\\ \text{dynamic-lookup}(CH,\sigma,m) & \text{如果 } v=\text{“}virtual\text{”}\end{cases}.$$

在程序执行过程中,当指针指向或引用σ的一个实例时,我们称实例化了一个子对象σ.

定义 7(子层次). 令 CH 和 CH' 是类层次,则 CH' 是 CH 的一个子层次(记为 $CH'\subseteq CH$)的充分必要条件是

(1) $\hat{C}(CH')\subseteq\hat{C}(CH)$;

(2) $\hat{M}(CH')\subseteq\hat{M}(CH)$;

(3) $n\hat{I}(CH')\subseteq n\hat{I}(CH)$;

(4) $v\hat{I}(CH')\subseteq v\hat{I}(CH)$.

定义 8(类层次切片). 令 $P-$(Main,M,O,CH)是一个程序,如果满足以下两个条件,则称 CH 的一个子层次 CH' 是 $P'=$(Main,M,O,CH')的一个类层次切片.

(1) $\Sigma(CH')$包含每个在 P' 的执行过程中可以被实例化的了对象 σ;

(2) CH'保留了在任何子对象 σ 中任何成员 m 的查询(lookup). 这些查询可以在 P' 的执行过程中完成,即 lookup(CH',σ,m)=lookup(CH,σ,m).

3 分层切片方法的应用

M. J. Horrold 等人成功地对图 3 中的 C++程序进行了面向对象的静态切片[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〈$S39$,current_floor〉的程序切片. 所以说,利用 SDG 切片面向对象的程序是一种可行的方法. 下面,我们给出利用分层切片算法来计算图 3 中 C++程序切片的分层结果.

令 $S^0(P)=${Main,〈$CE1$,$CE23$〉},其中〈$CE1$,$CE23$〉是用一个有序组来表示一个类层次,后面的类继承前面的类. 根据分层切片算法可以按以下步骤来计算图 3 中 C++程序关于切片标准〈$S39$,*current_floor*〉的程序切片:

(1) 从 $S^0(P)=${Main,M,O,CH}中删除那些与切片准则〈$S39$,current_floor〉无关的自由标准过程/方法、单个的类、类层次等后得到 $S^1(P)$. 本例中,$M=\varnothing$,$O=\varnothing$,$CH=$〈$CE1$,$CE23$〉(用一个元组表示类层次,元组的后面组元继承前面组元). 故

$$S^1(P)=\{\text{Main},\langle CE1,CE23\rangle\}.$$

(2) 因为类层次 $CH=$〈$CE1$,$CE23$〉中的两个类 $CE1$,$CE23$ 均与切片准则〈$S39$,current_floor〉有关,$CH=$〈$CE1$,$CE23$〉的子层次就是它本身,故

$$S^2(P)=\{\text{Main},\langle CE1,CE23\rangle\}.$$

(3) 因为 $CE1$ 中与切片准则〈$S39$,current_floor〉有关的方法主要有{$E2$,$E11$,$E15$,$E21$},$CE23$ 中与切片准则〈$S39$,current_floor〉有关的方法有{$E24$,$E31$},则删除 $S^2(P)$中与切片准则无关的方法得到 $S^3(P)$,显然,

$$S^3(P)=\{\text{Main},E2,E11,E15,E21,E24,E23\}.$$

(4) 对 $S^3(P)$中的每个方法(把 main 函数作为一般的方法来看待)进行分析,删除每个方法中与切片准则〈$S39$,current_floor〉无关的语句可控制谓词后得到:

$$S(P)=\{E2,S3,S4,S5,E11,S12,E15,S16,S17,C18,S19,C20,E21,S22,E24,C25,S26,E31,S32,C33,E34,S35,S36,S37,C38,S38\}.$$

$S(P)$就是图 3 中 C++程序通过分层切片算法计算得到的关于切片准则〈$S39$,current_floor〉的切片,如图 3 所示.

```
CE1  classElevator{
       public:
E2       Elevator(int)_top_floor)
S3         {current_floor=1;
S4          current_direction=UP;
S5          top_floor=1 to p floor;}
E6       virtual~Elevator(){}
E7       void up()
S8         {current_direction=UP;}
E8       void down()
S10        {current direction=DOWN;}
E11      int which_floor()
S12        {returncurrent_floor;}
E13      Direction direction()
S14        {return current_direction;}
E15      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:
E21      add(int &a,const int&b)
S22        {a=a-b;}
       protected:
         int curent_floor;
         Direction current_direction;
         int top_floor;
       };
CE23 class AlarmElevator:publicElevator{
       public:
E24      AlarmElevator(inttop_floor):
C25        Elevator(top_floor)
S26          {alarm_on=0;}
E27        void set_alarm()
S28          {alarm_on=1;}
E29        void reset_alarm()
S30          {alarm_on=0;}
E31        void go(int floor)
S32          {if(alarm_on)
C33          Elevatr::go(floor)
             }
       protected:
         int alarm_on;
       };
E34  main(int argc,char argv){
       Elevator * e_ptr;
S35    if (argv[1])
S36      e_ptr=new larmElevator(10);
       else
S37      e_ptr=new Elevator(10);
C38    e_ptr->go(5);
S39    cout<<"\nCurrently on floor:"
         <<e ptr->which floor ()<<"\n";
       }
```

Fig. 3 A C++ program for elevator class, and its slice w.r.t. slicing criterion ⟨S39,current_floor⟩

图 3 一个关于电梯类的 C++ 程序及其关于切片标准⟨S39,current_floor⟩的切片

4 结 论

本文从程序逻辑分层的角度提出一种 OO 程序的层次模型,然后在层次模型的基础上采用逐步求精算法来分层计算 OO 程序的切片.当然,分层切片算法是在其他切片算法基础上建立起来的,其中对方法/过程的切片需要用到过程内切片和过程间切片,对类层次的切片实际上是借助文献[5]中提出的切片类层次的方法.文中主要讨论的切片算法以及例子基本上都是围绕面向对象切片准则的第 1 种解释展开的.对于第 2 种解释,即切片准则$\langle s,v\rangle^*$,其中 v 和 s 分别是某个类的某个方法中的一个变量和一条语句,文中的逐步求精算法需要稍作改动,我们准备另文加以阐述.另外,本文提出的分层切片方法还只是一种有较大限制的切片静态类型面向对象程序的方法,下一步工作是逐步对其进行扩展,使其能够计算面向对象程序的动态切片,甚至可以切片动态类型面向对象程序.

References:

[1] Zhao, J. J. Dynamic slicing of object-oriented program. Technical Report, SE-98-119, Information Processing Society of Japan, 17~23. 1998.

[2] Zhao, J. J., Cheng, J. D., 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.

[3] Krishnaswamy, A. Program slicing: an application of object-oriented program dependency graphs. Technical Report,

TR94-108, Department of Computer Science, Clemson University, 1994.

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

[5] Tip, F., Choi, J. D., Field, J., *et al*. Slicing class hierarchy in C++. In: Proceedings of the 11th Annual Conference on Object-Oriented Programming, System, Language, and Application. ACM Press, 1996. 179~197.

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

[7] Christoph, Steindl. Intermodular slicing of object-oriented programs. In: Koskimies, K., ed. Compiler Construction. Lecture Notes in Computer Science 1383, Springer, 1998.

[8] Chen, J. L., Wang, F. J., Chen, Y. L. Slicing object-oriented programs. In: Proceedings of the APSEC'97. Hong Kong, IEEE Computer Society Press, 1997. 395~404.

[9] Law, R. C. H. Object-Oriented program slicing [Ph. D. Thesis]. Regina, Canada: University of Regina, 1994.

[10] Li, Bi-xin, Liang, Jia, Zhang, Yong-xiang, *et al*. An approach for analyzing object-oriented program based on class hierarchy graph. Journal of Software, 2000,11(5): 694~700 (in Chinese).

[11] Yang, Hong, Xu, Bao-wen. Design and Implementation of a PSS/Ada program slicing system. Computer Research and Development, 1997,34(3):217~222 (in Chinese).

附中文参考文献:

[10] 李必信,梁佳,张勇翔,等.一种基于类层次图的分析面向对象程序的框架.软件学报,2000,11(5):694~700.

[11] 杨洪,徐宝文.PSS/Ada程序切片系统的设计和实现.计算机研究与发展,1997,34(3):217~222.

An Approach for Hierarchy Slicing Object-Oriented Program*

LI Bi-xin, LIU Xiao-dong, ZHENG Tao, LI Xuan-dong, ZHENG Guo-liang

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093, *China*);
(*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230056, *China*)
E-mail: zhenggl@nju.edu.cn; lbxin@mail.hf.ah.cn
http://www.nju.edu.cn

Abstract: Program slicing is an important method for debugging, testing, analyzing, understanding and maintaining object-oriented software. In this paper, a new hierarchy slicing approach is presented so as to replace old slicing methods based on SDG (system dependence graph). The main idea of hierarchy slice and algorithm are discussed in this paper, and the application of hierarchy slicing algorithm is also presented at the end of this paper.

Key words: hierarchy model; hierarchy slice; stepwise refinement; class hierarchy slice

* Received April 12, 2000; accepted July 12, 2000
Supported by the Youth Foundation of the National High Technology Development 863 Program of China under Grant No. 863-306-QN2000-2