ISSN1000-9825　Journal of Software　软件学报　2000,11(5):694～700

一种基于类层次图的分析面向对象程序的框架*

李必信　梁佳　张勇翔　樊晓聪　郑国梁

(南京大学计算机软件新技术国家重点实验室　南京　210093)
(南京大学计算机科学与技术系　南京　210093)
E-mail: zhenggl@nju.edu.cn

摘要　从类层次图CHG(class hierarchy graph)出发,提出一个基于CHG的分析面向对象程序的框架OOAF(object-oriented analyzing framework),讨论了OOAF的功能、算法和设计思想,给出了子对象识别以及可见方法、主导方法的确定算法,建立了可见方法类层次图;并且通过计算方法的继承集、改写集以及对方法改写边界的确定,生成程序的虚函数调用图,从而为理解面向对象程序中的虚函数调用问题提供了一种可行的解决方案.

关键词　类层次图,面向对象,分析框架,子对象识别,方法确定,虚函数调用图.
中图法分类号　TP311

与分析结构化程序相比,分析面向对象的程序要显得困难得多.究其原因,我们认为主要有以下几点:(1)面向对象程序设计鼓励代码分解和细化程序设计,结果是不仅过程体越来越小,而且过程调用越来越频繁;(2)由于使用了动态定连机制,被调用的过程依赖接收方的动态类型,所以直到运行时才能确定该调用哪一个过程,从而导致分析调用问题特别困难;(3)面向对象程序设计允许继承机制,一个对象可能属于由许多类构成的一串类,这就使得对象的行为难以控制,特别是多继承机制的使用使得类层次结构以及类之间的关系更加复杂;(4)由于子类型、重载(多态性)机制的使用,导致在某个程序点,我们很难确定一个指针指向的对象在程序运行过程中的类型[1~4].本文提出一个基于类层次图CHG(class hierarchy graph)的分析面向对象程序的框架——OOAF(object-oriented analyzing framework).OOAF目前具有以下几个功能:(1)利用路径等价(类)技术识别子对象;(2)利用子对象(路径)进行可见方法和主导方法的确定;(3)结合确定的可见方法和类层次图建立可见方法类层次图VM-CHG(visible method class hierarchy graph);(4)在VM-CHG的基础上计算继承集和改写集,确定某个方法的改写边界;(5)构造虚函数调用图VFCG(virtual function call graph),以解决虚函数调用的模型化问题.本文第1节介绍文中将要用到的两个基本概念.第2节详细介绍OOAF功能的实现原理和实现算法.第3节概述OOAF框架的结构思想.第4节给出与相关工作的比较,并给出文中工作的总结和今后的研究内容.

1　基本概念和术语

我们可以用一个直接无环图来模型化一个C++/Java程序的类层次,这个直接无环图就是类层次图(CHG),类层次图的结点表示类,边表示继承关系.类层次图的严格定义如下:

定义1(类层次图). 令N表示C++程序中类的集合,Ev表示类Y与它的直接虚基类X构成的有序对(X,

* 本文研究得到江苏省应用基础基金(No.BJ97036)资助.作者李必信,1969年生,博士生,主要研究领域为面向对象技术及其支持环境,程序理解.梁佳,女,1974年生,硕士生,主要研究领域为软件工程,面向对象技术.张勇翔,1975年生,硕士生,主要研究领域为软件工程,面向对象技术.樊晓聪,1972年生,博士,主要研究领域为软件Agent理论,人工智能.郑国梁,1937年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:李必信,南京210093,南京大学计算机科学与技术系

本文1999-01-19收到原稿,1999-06-16收到修改稿

© 中国科学院软件研究所　http://www.jos.org.cn

Y)的集合，$\boldsymbol{Env}$ 表示类 Y 与它的直接非虚基类 X 构成的有序对(X,Y)的集合，我们把 $\boldsymbol{Ev}$、$\boldsymbol{Env}$ 中的元素分别称为虚边和非虚边. 令 $\boldsymbol{E}=\boldsymbol{Ev}\cup\boldsymbol{Env}$，我们使用记号 $X\rightarrow Y$ 表示 $\boldsymbol{E}$ 中的元素(X,Y)，则类层次图(记为 CHG)就是一个二元组 $G=(\boldsymbol{N},\boldsymbol{E})$. 如图 1(b)所示. CHG 中每个结点旁边标的是在该结点中定义的方法，并且我们把在某个结点 σ 中所定义的方法集合记为 $M[\sigma]$. 例如，$M[A]=\{foo\}$.

$A_0,A_1,\ldots,A_m\in \boldsymbol{N}; e_1,e_2,\ldots,e_l\in \boldsymbol{E}$，其中 $e_i=(A_{i-1},A_i)$是关联 A_{i-1},A_i 的边.

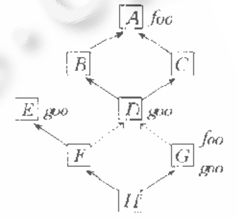
定义 2(路径). 给定类层次图 $G=(N,E)$，则交替序列 $A_je_{j+1}A_{j+1}e_{j+2}\ldots A_{k-1}e_kA_k$ 就称为连接 A_j 到 A_k 的一条路径，记为 $A_jA_{j+1}\ldots A_k$. 其中 A_j 和 A_k 分别称作路径的起点和终点，边的数目 pl 称为路径的长度.

例如，图 1(a)中 C++程序的 CHG 如图 1(b)所示，CHG 图中从结点 A 到结点 H 的路径有 $ABDFH$，$ABDGH$，$ACDFH$，$ACDGH$.

利用一条路径中的结点序列表示 CHG 中的一条路径，结点序列的方向与路径的方向一致. 例如，$ABDFH$ 表示从结点 A 到结点 H 的一条路径. 用 $\alpha\cdot\beta$ 表示由路径 α 和 β 连接而成的路径. 例如，$(ABD)\cdot(DFH)$ 就是 $ABDFH$. 称 α 是 $\alpha\cdot\beta$ 的前缀，β 是 $\alpha\cdot\beta$ 的后缀(一条路径既是它自己的前缀又是它自己的后缀)，则由定义 2 可得：类 X 是类 Y 的一个基类当且仅当在 CHG 中存在一条从 X 到 Y 的非空路径；类 X 是类 Y 的一个虚基类当且仅当存在一条从 X 到 Y 的路径且它的第 1 条边是虚边.

```
Class A {foo();}
Class B:A {}
Class C:A {}
Class D:B,C {goo();}
Class E {goo();}
Class F:E,virtual D {}
Class G:virtual D {
foo();
goo();}
ClassH:F,G {}
main()
{
  H h;
  h.foo;
  h.goo
}
```

(a) A C++ program
(a) 一个C++程序



(b) Class hierarchy graph
(b) 类层次图

Fig.1
图1

2 OOAF 的功能描述和算法设计

2.1 路径等价(类)和子对象识别

对任意一条路径 α，用 Startnode(α)表示路径的起点，用 Endnode(α)表示路径的终点. 令 Prelongest(α)表示一条不包含任何虚边的路径的最长前缀.

定义 3(路径等价). 给定 CHG 中的两条路径 α 和 β，则 α 与 β 等价(记为 $\alpha\Leftrightarrow_{path}\beta$)当且仅当 Prelongest($\alpha$)=Prelongest($\beta$)且 Endnode($\alpha$)=Endnode($\beta$).

定义 4(子对象). 由定义 1 和定义 2 可知，在类层次图中，边 $B\rightarrow D$ 表示继承边，它的本质含义是每个 D 对象包含一个 B 对象，则称此 B 对象为 D 对象的子对象. 特别地，D 对象继承了 B 对象的所有方法. 如果 A_n 是 A_1 的一个基类，则存在一条路径 $A_nA_{n-1}\ldots A_2A_1$. 这条路径暗示了每个 A_1 对象包含一个 A_2 子对象，A_2 子对象又包含一个 A_3 子对象，…，依此类推，直到 A_{n-1} 包含一个 A_n 子对象.

显然，等价关系 $\Leftrightarrow_{path}$ 可用来确定从两条不同的混合路径(既有虚继承又有非虚继承的路径)何时可以识别同一个子对象，即 $\alpha\Leftrightarrow_{path}\beta$ 当且仅当 α 和 β 都可识别同一个子对象. 对任意路径 α，令$[\alpha]$表示 α 的等价类. 根据前面的解释，我们可以利用等价类来识别或命名子对象. 因为条件 Prelongest$[\alpha]$=Prelongest$[\beta]$暗示了 Startnode$[\alpha]$=Startnode$[\beta]$，等价关系 $\Leftrightarrow_{path}$ 只有在具有相同端点的路径之间才能成立，故有 Endnode($[\alpha]$)=Endnode(α)；Startnode($[\beta]$)=Startnode(β).

我们用 $PEC(G)$表示类层次图 G 中路径的所有等价类的集合，用 Σ 表示 CHG 中所有类的对象所包含的所有子对象的集合.

定义 5. $\varphi: PEC(G)\rightarrow\Sigma$ 是 $PEC(G)$到 Σ 的单值映射，即一个路径等价类只映射到某个对象的一个子对象.

显然，映射 φ 是 $PEC(G)$到 Σ 的一一映射，故一个等价类只能识别同一个子对象，不同的等价类可以识别不同的子对象. 例如，考虑图 1 中的类层次图，从类 A 到类 H 之间有 4 条路径，这些路径的 Prelongest 部分如下：Prelongest($ABDFH$)=ABD，Prelongest($ACDGH$)=ACD，Prelongest($ACDFH$)=ACD，Prelongest($ABDGH$)=ABD. 因此有下面的等价关系：$ABDFH\Leftrightarrow_{path}ABDGH$，$ACDFH\Leftrightarrow_{path}ACDGH$. 相应地，$ABDFH$ 和 $ABDGH$

© 中国科学院软件研究所 http://www.jos.org.cn

指明 H 对象中的同一个子对象,同时,$ACDGH$ 和 $ACDFH$ 指明 H 对象中同一个子对象. 然而,$ABDFH \not\Leftrightarrow ACDFH$. 这样就有类 A 的两个不同版本的子对象在类 H 的一个实例中.

算法 1. 子对象识别算法

输入:类层次图(CHG),

输出:各类结点对应实例的子对象集合.

步骤 1. 在单继承的情况下,衍类只含有基类的一个子对象版本;

步骤 2. 在多继承存在的情况下,因为在给定的一对对象 B 和 D 之间存在着多条路径,根据有无虚继承分成 3 种情况进行讨论:① 在没有非虚继承存在的情况下,在同一对类之间的所有路径可识别同一个子对象;② 在没有虚继承存在的情况下,不同的路径总是识别不同的子对象;③ 在既有虚继承又有非虚继承的情况下,我们可以采用下面的路径等价或路径等价类的方法来识别子对象,具体做法如下:识别子对象就是寻找路径等价类,一个路径等价类可识别一个子对象,不同的等价类可识别不同的子对象.

2.2 可见方法和主导方法确定

现在,我们准备定义类实例的组成部分. 组成类 X 的一个实例的子对象的集合是:$\{\sigma \in PEC(G) \mid \text{Endnode}(\sigma)=X\}$(因为一个路径等价类对应一个子对象). 一个子对象 σ 本身就是由一些方法构成的集合,且这些方法是由 $M[\text{Startnode}(\sigma)]$ 定义的.

定义 6(路径控制). 在 CHG 中,如果 α 是 β 的一个后缀,则称 α 隐含了 β. 在 CHG 中,一条路径 α 控制另一条路径 β 当且仅当 α 隐含的某条路径 β',满足 $\beta' \Leftrightarrow_{path} \beta'$. α 控制 β 记为 $\alpha \Rightarrow_{\text{do min ates}} \beta$.

定理 1. 令 $\alpha \Leftrightarrow_{\text{path}} \alpha'$, $\beta \Leftrightarrow_{\text{path}} \beta'$, 则 $\alpha \Rightarrow_{\text{do min ates}} \beta$ 当且仅当 $\alpha' \Rightarrow_{\text{do min ates}} \beta'$.

推论 1. $[\alpha] \Rightarrow_{\text{do min ates}} [\beta]$ 当且仅当 $\alpha \Rightarrow_{\text{do min ates}} \beta$.

定理 2. $(PEC(G), \Rightarrow_{\text{do min ates}})$ 是一个偏序关系.

令 C 是 CHG 中的一个类. 在类 C 的一个对象中,包含方法 m 的所有子对象的集合定义如下:

定义 7. $Def(C,m)=\{\sigma \in PEC(G) \mid \text{Endnode}(\sigma)=C \,\&\, m \in M[\text{Startnode}(\sigma)]\}$.

例. 考虑如图 1(b)所示的 CHG. 计算各个对象中包含方法 m 的子对象集合. $Def(H,foo)=\{\{ABDFH, ABDGH\},\{ACDFH,ACDGH\},\{GH\}\}$; $Def(G,foo)=\{\{ABDG\},\{ACDG\},\{G\}\}$; $Def(F,foo)=\{\{ABDF\},\{ACDF\}\}$; $Def(E,foo)=\{\perp\}$; $Def(D,foo)=\{\{ABD\},\{ACD\}\}$; $Def(C,foo)=\{\{AC\}\}$; $Def(B,foo)=\{\{AB\}\}$; $Def(A,foo)=\{\{A\}\}$. $Def(C,foo)$中的每个元素都是路径的一个等价类,且每个等价类都分别识别包含 foo 的 H 对象的互不相同的一个子对象. 类似地,我们有 $Def(H,goo)=\{\{EFH\},\{DFH,DGH\},\{GH\}\}$. $Def(G,goo)=\{\{DG\},\{G\}\}$; $Def(F,goo)=\{\{DF\},\{EF\}\}$; $Def(E,goo)=\{\{E\}\}$; $Def(D,goo)=\{\{D\}\}$; $Def(C,goo)=\{\perp\}$; $Def(B,goo)=\{\perp\}$; $Def(A,goo)=\{\perp\}$.

定义 8. 设 $PEC(C)$是终点为结点 C 的所有路径等价类的集合,$PEC(C)=\{[P1],[P2],\ldots,[Pk]\}$,其中 $[Pi]$$(1<i<k)$是路径 Pi 的等价类.

(1) 如果存在一条路径 $\sigma \in [Pi]$,使得对某个路径 $\sigma' \in [Pj]$,满足 $\sigma \Rightarrow_{\text{do min ates}} \sigma'$,则称等价类$[Pi]$控制等价类$[Pj]$,记为$[Pi] \Rightarrow_{ec} [Pj]$.

(2) 令 $Paths(C)$是终点为 C 的所有路径的集合,如果存在一条路径 $\sigma \in Paths(C)$,使得对任意路径,满足 $\sigma \Rightarrow_{\text{do min ates}} \sigma'$,则定义路径 $\sigma \in Paths(C)$为 $PATHS(C)$的主导路径(记为 most-dominant($Paths(C)$));如果不存在这样的 σ,则我们定义 most-dominant($Paths(C)$)为$\perp$.

根据定义 8 中的(1),我们可以优化集合 $Def(C,m)$,优化原则如下:

优化原则. ① 利用等价类之间的路径控制减少 $Def(C,m)$中等价类的数目,即如果等价类$[Pi] \Rightarrow_{ec} [Pj]$,则删除$[Pj]$;② 在剩下的等价类中各取一条路径作为优化后的 $Def(C,m)$中的元素,优化后的集合记为 $Def_{\text{optimal}}(C,m)$.

定义 9(可见方法). 我们称集合 $Def_{\text{optimal}}(C,m)$为结点 C 处的可见方法集合. 可写成 visible-method($Def(C,m)$)$=Def_{\text{optimal}}(C,m)$. 实际上,一个类结点中包含的通过继承、改写或定义的方法的集合称为该结点的可见方法集.

© 中国科学院软件研究所 http://www.jos.org.cn

下面定义可见方法和主导方法的确定操作 VM-Determination 和 MDM-Determination.

定义 10. VM-Determination(C,m) = visible-method$(Def(C,m))$；MDM-Determination(C,m) = most-dominant$(Def(C,m))$.

例. 考虑例 1 中的 CHG，因为 $\{GH\}$ 控制 $Def(H,foo)$ 中的每个元素，MDM-Determination(H,goo) = $\{GH\}$. 因为 $Def(H,goo)$ 没有一个 most-dominant 元素，MDM-Determination$(H,goo)=\perp$，VM-Determination$(H,goo)=\{EFH,GH\}$，VM-Determination$(H,foo)=\{GH\}$.

定义 11. $DefPath(C,m)=\{\alpha\in Paths(G)\,|\,\text{Endnode}(\alpha)=C,\ \&\ m\in M[\text{Startnode}(\alpha)]\}$. 由此，可把定义 9 和定义 10 扩展到路径上.

定理 3. 路径 $\gamma\cdot(X\rightarrow Y)\Rightarrow_{dominates}\delta\cdot(X\rightarrow Y)$ 的充要条件是 $\gamma\Rightarrow_{dominates}\delta$.

推论 2. 如果 γ 控制 δ，且 $\gamma\neq\delta$，则对任意路径 $\gamma\cdot\omega$ 和任意包含路径 $\gamma\cdot\omega$ 的集合 S，S 有一个 most-dominant 元素当且仅当 $S-\{\delta\cdot\omega\}$ 有一个 most-dominant 元素，在这种情况下，两个集合的 most-dominant 元素是 $\Leftrightarrow_{path}$-等价的.

下面的算法将返回 $DefPath(C,m)$ 的 most-dominant 元素，对类 C 的方法 m 来说，VM-Determination 和 MDM-Determination 返回的不是一个路径的等价类，而是等价类的一个随机元素. 这里，我们给出算法的简单描述.

算法 2. 可见方法和主导方法确定算法.

输入：CHG 中的每个类结点，

输出：可见方法和主导方法.

第 1 阶段. 计算 VM-Determination(C,m).

步骤 1. 为每个类 C 和方法 m 计算集合 $DefPath(C,m)$.

步骤 2. 如果对某个类 C 来说，方法 m 是 $DefPath(C,m)$ 的元素，可以考虑用一条路径 α 来作为方法 m 的一个定义.

步骤 3. 如果 α 至少由一条边构成，我们就把定义 $\alpha::m$ 作为一个继承定义；否则，就称为是产生定义. m 的所有产生定义的集合比较容易计算：它就是简单集合 $\{C::m\,|\,C\in N$ 且 $m\in M[C]\}$.

步骤 4. 从所有产生定义的集合出发，我们能够利用通过 CHG 传播方法定义(包括产生的和继承的)的重复过程识别所有的继承方法定义，更精确地讲，一个定义 $\alpha::m$ 可以沿着 Endnode(α) 的所有输出边来传播；一个定义 $\alpha::m$ 通过一条边 $X\rightarrow Y$ 的传播可以识别一个新的(继承的)定义，命名为 $\alpha\cdot(X\rightarrow Y)::m$. 当不再有什么新定义需要传播时，这个重复过程就停止了.

第 2 阶段. 本阶段(确定阶段)简单地为每个类确定到达 C 的所有方法 m 的定义的集合是否有一个 most-dominant 定义 α. 如果有，则 MDM-Determination(C,m) 的结果是 α；如果没有，则 MDM-Determination(C,m) 无定义.

2.3 VM-CHG 及其构造方法

我们用 Bases(C) 表示类 C 的所有基类的集合，即 Bases$(C)=\{B\in \boldsymbol{C}:\langle B,C\rangle\in \boldsymbol{E}\}$；用 Derived$(C)$ 表示类 C 的所有衍类的集合，即 Derived$(C)=\{D\in \boldsymbol{C}:\langle C,D\rangle\in \boldsymbol{E}\}$. 再令 Derived$(C)$ = Derived$(C)\cup\{C\}$，Bases(C) = Bases$(C)\cup\{C\}$. 这样就使得图 VM-CHG 中存在一条从 C 到 D 的路径或存在一条从 B 到 C 的路径.

对类结点 $C\in \boldsymbol{C}$ 来说，它的可见方法集合 V 是由在该结点定义的方法和该结点从其基类继承或通过对其基类方法改写得到的方法构成的. 我们把可见方法表示成类结点中的一个子结点，即一个可见方法子结点 $v\in \boldsymbol{V}$ 是一个三元组 $\langle C,m,D\rangle$，其中，① $C\in \boldsymbol{C}$ 是 v 在其中表示可见方法的类；② $m\in M$ 是由类 C 定义的或继承的方法；③ $D\in C$ 是 m 的定义类，如果方法 m 是由类定义而非继承的，则 $C=D$，如果方法 m 是从一个基类继承来的，则 $C\in$ Derived(D).

定义 12. 一个 VM-CHG 是一个三元组 $\langle \boldsymbol{N},\boldsymbol{E},\boldsymbol{V}\rangle$，这里：$\boldsymbol{N}$ 是类结点的集合；$\boldsymbol{E}$ 是表示继承关系的边(又称为导出边)的集合，$\boldsymbol{E}$ 中的导出边是一个有序对 (B,D)，其中 $B\in \boldsymbol{N}$ 是基类，$D\in \boldsymbol{N}$ 是衍类；$\boldsymbol{V}$ 是可见方法子结点的集合.

© 中国科学院软件研究所 http://www.jos.org.cn

我们把由 VM-Determination(C,m)操作得到的每个结点的可见方法分别对应地与图 1(b)的类层次图中的结点相结合就得到如图 2 所示的 VM-CHG.图中每个结点中左边的子结点表示在该结点定义的方法,右边的子结点表示继承的方法.如 D 结点中的 $D::foo$ 是在 D 中定义的方法,$ABD::foo$ 和 $ACD::foo$ 表示 D 分别从 B 和 C 继承的方法.

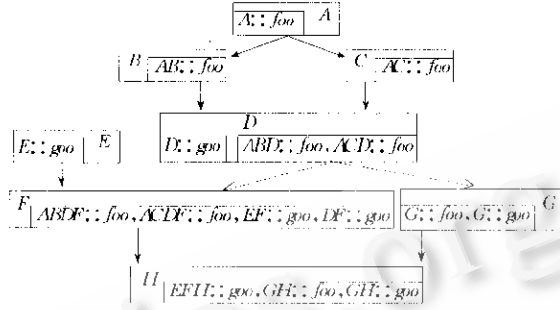


Fig.2 VM-CHG of program in Fig.1
图2 图1程序的可见方法类层次图

2.4 计算继承集和改写集

定义 13. 改写某个特定类中方法 m 的类的集合称为该方法的改写集,记为 $Override(m)$;继承某个特定类中方法 m 的类的集合,称为该方法的继承集,记为 $Inherit(m)$.对一个特定的可见方法 m,继承集和改写集在 VM-CHG 中为可见方法 m 确定了一个边界.这样,一个边界把 VM-CHG 中的继承方法 m 的结点和继承方法 m' 或定义方法 m'($m'\neq m$ 但 $Sig(m)=Sig(m')$)的结点分离开来.我们称这种边界为方法的改写边界.

例.对类结点 D 中的可见方法$\langle D,D::goo,D\rangle$来说,改写集是$\{F\}$,继承集是$\{D,G\}$.类结点 D 和 F 在边界的一边都继承方法 $D::goo$,在边界的另一边的结点 G 定义或继承了同一个标记符 $D::goo$ 的方法,但已不是原来的 $D::goo$.

计算改写集和继承集的次序是按逆拓扑次序进行的,也就是说,在访问 VM-CHG 中的基类结点之前先访问衍类结点.下面描述用来计算 VM-CHG 中所有可见方法子结点的继承集和改写集算法.

算法 3. 计算继承集和改写集的算法.

输入:VM-CHG,

输出:$Inherit(m)$,$Override(m)$.

步骤 1. 从叶子结点开始计算,因为叶子结点没有子孙,所以不能改写任何在叶子结点 K 的可见方法.同样地,K 也没有子孙能够继承它的任何可见方法.因此,对叶子结点 K 的所有可见方法来说,继承集是$\{K\}$,改写集是$\varnothing$;

步骤 2. 初始化所有可见方法的继承集和改写集;

步骤 3. 利用循环语句按照从叶子到根的逆拓扑次序处理类结点.对在循环中处理的每个结点 C,我们访问它的基类,并且更新它们的继承集和改写集.在某个基类结点 B,可见方法 m 的 $Inherit(m)$和 $Override(m)$是根据它的衍类结点 C 的相关集合计算出来的;

步骤 4. 利用检查语句来测试一下 v 中方法 m 的定义类是不是 C,即是不是 C 改写了该方法.如果测试成功,就向 m 的改写集中加入衍类结点,否则,就向 $Override(m)$中加入衍类结点的改写集;

步骤 5. 再利用检查语句来测试在 v 中与衍类结点标记符匹配的可见方法是否具有相同的定义类.如果测试成功,就把衍类结点的继承集加到 m 的继承集中以达到更新的目的.

2.5 VFCG 及其构造算法

定义 14(虚函数调用图 VFCG). $VFCG$ 是一个四元组,即 $VFCG=\langle F,S,I,R\rangle$,其中 F 是函数结点的集合(F 包括方法和非方法,因此 $M_P\subseteq F$);S 是调用位置子结点的集合;I 是调用实例边的集合;$R\subseteq F$ 是调用图的根的集合.其中,一个调用实例边是一个四元组$\langle s,f,g,P\rangle\in I$.这里,① $s\in S$ 是调用位置;② $f\in F$ 是正在进行调

© 中国科学院软件研究所 http://www.jos.org.cn

用的函数；③ $g\in F$ 是调用的目标函数；④ P 有两种情况，如果 s 不是虚调用，$P=\perp$；如果 s 是一个虚调用位置，则 P 是可能目标类的集合，且在其上能够激发虚方法调用($P\subseteq C$，C 是所有类的集合). 我们用 Σ 表示调用位置信息. Σ 中的调用位置信息包括直接调用位置(Σ_D)信息和虚调用位置(Σ_V)信息. 所有调用位置都是三元组，其中，① $s\in S$ 是调用位置标识符；② $f\in F$ 是正在进行调用活动的函数；③ 一个直接调用位置 $k\in\Sigma_D$ 是一个三元组 $\langle s,f,g\rangle$，这里，$g\in F$ 是调用的目标函数；④ 一个虚调用位置 $k\in\Sigma_V$ 是一个三元组 $\langle s,f,v\rangle$，这里，$v\in V$ 是相应调用的静态类型的可见函数. 集合 S_D 和 S_V 是 S 的子集，$S=S_D\cup S_V$，其中 S_D，S_V 可分别形式地表示成

$$S_D=\{s\in S\mid f\in F:\langle s,f,t\rangle\in\Sigma_D\},\ S_V=\{s\in S\mid f\in f,v\in V:\langle s,f,v\rangle\in\Sigma_V\}.$$

算法 4. 构造 VFCG 的算法.

输入：VM-CHG，*Inherit*(*m*)，*Override*(*m*)，

输出：VFCG.

步骤 1. 把空集赋给调用实例边的集合 I；

步骤 2. loop1. 对每个直接调用位置 $\langle s,f,g\rangle\in\Sigma_D$，把 $I\cup\{\langle s,f,g,\perp\rangle\}$ 赋给 I endloop1；

步骤 3. loop2. 对每个虚调用位置 $\langle s,f,v\rangle\in\Sigma_V$，调用函数 addVI$(s,f,v)$endloop2；

步骤 4. 函数 addVI$(s\in S_V,f\in F,v\in V)$

令 $\langle C,m,D\rangle:=v$； /* 先给 $\langle C,m,D\rangle$ 赋上一个可见方法子结点 v */

步骤 5. if：如果 $\langle s,f,m,Inherits(v)\rangle\in I$，返回；

else：把 $I\cup\{\langle s,f,m,Inherits(v)\rangle\}$ 赋给 I；

endif

步骤 6. loop3. 对每个 $w\in Override(v)$，调用 addVI(s,f,w) endloop3.

3 OOAF 结构设计概述

目前，OOAF 具有以下功能：① 利用路径等价(类)技术识别子对象；② 利用路径(子对象)进行可见方法和主导方法确定；③ 结合确定的可见方法和 CHG 建立可见方法类层次图(VM-CHG)；④ 在 VM-CHG 的基础上计算继承集和改写集，确定改写边界；⑤ 构造虚函数调用图(VFCG). 为了实现这些功能，我们设计出 OOAF 的结构，如图 3 所示. OOAF 结构由以下几部分组成：(1) 词法和语法分析器(lexical and syntax analyser)，用来分

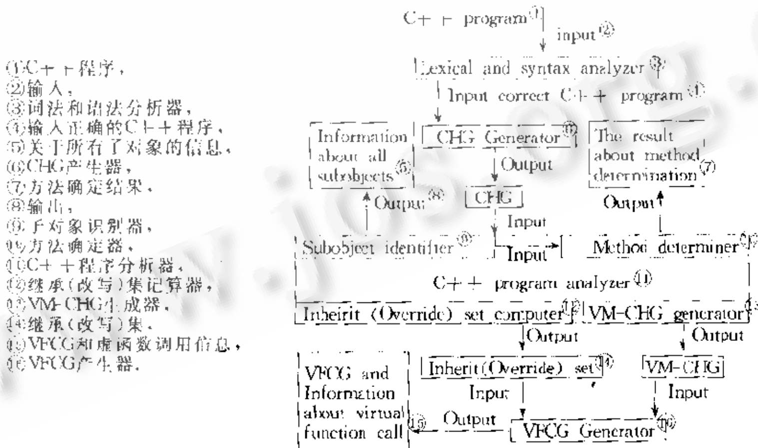


Fig. 3 The designed structure of OOAF
图3 OOAF的设计结构

析输入的 C++/Java 源程序的词法和语法正确性. (2) CHG 产生器(CHG generator). 对词法和语法正确的 C++/Java 源程序产生 CHG 图，并负责把 CHG 输入 C++/Java 程序分析器中. (3) C++/Java 程序分析器(C++/Java program analyser). 该分析器又由 4 部分组成：① 子对象识别器(subobject identifier)，用来识别各个类结点的对象包含的子对象的情况；② 方法确定处理器(method determiner)，用来确定各个对象(类)结点可见方法和主导方法；③ 继承(或改写)集计算器(inherit/override set computer)，用来计算每个方法的继承集和改

© 中国科学院软件研究所 http://www.jos.org.cn

写集,并确定该方法的改写边界;(3) VM-CHG 产生器(VM-CHG genereator),用来产生 VM-CHG,并把 VM-CHG 输入到 VFCG 产生器.(4) VFCG 产生器(VFCG generator).它利用继承(改写)集计算器和 VM-CHG 产生器输入的继承集、改写集、VM-CHG 等构造 VFCG,并报告虚函数调用的有关信息.

4 结论与相关工作

G. Ramalingam 等人在文献[5]中主要提供了一种成员查询算法,该算法也是基于类层次图的,但在虚继承和多继承存在的情况下,该算法变得相当复杂. Bacon D. F. 在他的博士论文[6]中提出一种基于类层次图的快速类型分析算法,该算法为虚函数调用的优化问题提供了一种可行的解决方案. F. Tip 等人在文献[7]中首次利用程序切片等技术来分析和理解面向对象的程序.他们的主要贡献是提出一种基于 CHG 的分析框架以及利用路径等价(类)进行了对象识别的算法、构造 VM-VHG 的方法、利用路径控制进行可见方法和主导方法确定的算法、计算继承集和改写集的算法、产生虚函数调用图(VFCG)的算法等.由此,为面向对象语言中复杂的虚函数调用问题的模型化提供了一个可行的解决方案. OOAF 适合于分析用 C++/Java 书写的面向对象程序.当然, OOAF 目前还只是一个功能较弱的分析静态类型面向对象程序的框架,在以后的研究中,我们将逐步加入对象的动态类型分析、程序切片、概念分析及可视化等技术,以进一步完善 OOAF 功能.

参考文献

1 Bacon D F, Sweeney P F. Fast static analysis of C++ virtual function calls. In: Berman A M ed. Proceedings of the 1996 ACM SIGPLAN Conference on Object Oriented Programming Systems, Language & Application (OOPSLA'96). New York: ACM Press, 1996. 324~341

2 Calder B, Grunwald D. Reducing indirect function call overhead in C++ programs. In: McKenna J ed. Conference Record of the 21th ACM Symposium on Principles of Programming Language (POPL). New York: ACM Press, 1994. 397~408

3 Driesen K, Holzle U. The direct cost of virtual function calls in C++. In: Berman A M ed. Proceedings of the 1996 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Language & Application (OOPSLA'96). New York: ACM Press, 1996. 306~323

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

5 Ramalingam G, Srinivasan H. A member lookup algorithm for C++. In: Berman A M ed. Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Language & Application (OOPSLA'97). New York: ACM Press, 1997. 18~30

6 David F B. Fast and effective optimization of statically typed object oriented Language [Ph. D Thesis]. University of California at Berkley, 1997. 17~38

7 Tip F, Choi J D, Field J *et al*. Slicing class hierarchy in C++. In: Berman A M ed. Proceedings of the 1996 ACM SIGPLAN Conference on Object Oriented Programming Systems, Language & Application (OOPSLA'96). New York: ACM Press, 1996. 179~197

A Framework for Analyzing Object-Oriented Programs Based on Class Hierarchy Graph

LI Bi-xin LIANG Jia ZHANG Yong-xiang FAN Xiao-cong ZHENG Guo-liang

(*Department of Computer Science and Technology Nanjing University Nanjing* 210093)
(*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

Abstract Based on CHG, the authors present a framework for analyzing object-oriented programs OOAF, and discuss the functions, the algorithms and the design ideas of OOAF in this paper. The algorithms for identifying subobject and determining visible methods and the most-dominant-method are also given to create visible-method-class hierarchy-graph (VM-CHG). The virtual-function-call-graph (VFCG) of the program is obtained by computing inheritance-set, override-set, and override frontier, which provides a feasible way for understanding virtual function call in object-oriented programs.

Key words Class hierarchy graph (CHG), object-orientation, analyzing framework, subobject identification, method determination, virtual function call graph (VFCG).

© 中国科学院软件研究所 http://www.jos.org.cn