对象抽取方法研究与进展*)

Researches on Object Extracting

徐宝文[1,2]　周毓明[2]
(北京航天航空大学计算机科学与工程系　北京100083)[1]
(东南大学计算机科学与工程系　南京210096)[2]

Abstract　Transforming legacy systems written in conventional procedural languages into equivalent object-oriented systems makes software more maintainable, reliable and understandable. However, there is no concept directly corresponding to objects in non object-oriented systems. A lot of object-extraction approaches for extracting objects from legacy systems are hence proposed, which are the basis of reengineering non object-oriented systems into object-oriented systems. These object-extraction methods proposed recently are discussed, analyzed and classified in detail in this paper.

Keywords　Reverse engineering, Object, Object-extracting, Cohesion, Coupling, Concept analysis

1　引言

可理解性、可重用性和可靠性是衡量程序质量的主要标准。传统的面向过程的程序设计方法在历史上为软件危机的缓解做出了一定的贡献，但这种方法设计出来的模块独立性差、模块之间的耦合度往往比较高，从而造成系统后期维护上存在较大的困难。而现代的面向对象程序设计方法直接支持数据抽象、封装和继承，从而增强了软件的可理解性和可维护性。

在面向对象的方法出现之前，许多系统都采用面向过程的方法设计，目前有许多这样的系统正在运行，它们的维护和理解存在着较大的困难，当系统出现错误需要修改或用户需求发生变化需要增添新的功能时，维护人员不得不阅读所有的源程序代码和保留下来的设计文档，弄清系统的设计思想后，才能保证修改后的系统的正确运行。为提高这类系统的可维护性，一种方法是废弃这些系统而重新用面向对象的技术进行开发；另一种方法是通过逆向工程的方法，分析原系统的程序结构，从中抽取出对象，然后在保持系统语义的基础上将之转换成用面向对象方法设计的系统。与前一种方法相比，它不需要从设计文档开始对系统进行重新设计，能节省时间和财力。该方法的核心在于对象的识别与抽取，即怎样将相关的数据和子程序封装成问题域中的对象。除应用于非面向对象程序到面向对象程序的转换之外，对象的识别与抽取技术还有助于：

·理解原系统的设计思想。通过封装数据及相关的操作，可尽可能地还原出系统设计人员头脑中的"对象"，从而可帮助维护人员在抽象层次上理解程序，这在系统代码庞大而文档缺乏的情况下很有用。

·测试和调试，从而提高原系统的可靠性。

·提高可重用构件的生产率。抽取出候选对象后，我们理解其语义与功能、形成构件的规格描述并进行正确性测试后可将其放进可重用构件库。与那些特地开发可重用构件库的方法相比，这种方法产生可重用构件的开发时间短，生产率高。

目前，人们对非面向对象程序中对象的识别与抽取已进行了比较深入的研究，并提出了各种各样的对象抽取方法[1~13]。本文在指出对象抽取方法须解决问题的基础上，对现有对象抽取方法进行分类比较研究，并通过实例分析其中存在的问题，最后总结出将来的研究方向。

2　基本问题

在面向对象程序中，对象由数据(状态)和相关操作(方法)抽象封装而成，对象的行为通过操作展示，它不允许外界直接访问其内部状态，对象的每个方法实

*)本文得到国家自然科学基金(60073013)、江苏省自然科学基金、教育部高等学校骨干教师资助项目、江苏省"三三三"人才基金与江苏省青蓝工程跨世纪人才基金的资助。徐宝文　教授、博士生导师，主要从事程序设计语言、软件工程、并行程序设计等方向的教学与科研工作。周毓明　博士研究生。

现对象的某一特定操作。非面向对象程序中没有与对象直接对应的概念，但可将子程序与相关的类型和(或)数据组成的集合定义为候选对象。其中，子程序对应为候选对象的方法，类型对应为候选对象的属性，而数据则对应为候选对象类的实例[4]。即有如下定义：

定义1 设P是一个非面向对象程序，F为P中所有子程序组成的集合，T为P中所有类型组成的集合，D为P中所有数据项的集合，则P中的一个候选对象定义为：

$C_M^P=(\Phi,\tau,\delta)$

其中 $\Phi\subset F,\tau\subset T,\delta\subset D$，M是所采用的对象抽取方法。

下文用 $O_M^P=\{O_i|O_i=(\Phi_i,\tau_i,\delta_i),i\in n\}$ 表示对象抽取方法M从程序P中抽取出的候选对象所组成的集合。在抽取对象时，非面向对象程序中的一个子程序最多只能对应为一个候选对象的方法，即对任意两个不同的候选对象 O_i 和 O_j，应有 $\Phi_i\cap\Phi_j=\varnothing$。理想情况下，一个子程序实现某一特定的功能，因而对应于一个候选对象的方法。但实际上许多子程序不能直接对应为候选对象的方法。Myers等人将模块中的处理单元对之间的关系分为巧合内聚(Coincidental Cohesion)、逻辑内聚(Logical Cohesion)、时序内聚(Temporal Cohesion)、过程内聚(Procedural Cohesion)、通信内聚(Communicational Cohesion)、顺序内聚(Sequential Cohesion)和功能内聚(Functional Cohesion)七种类型[17,18]。参照其分类方法，我们将非面向对象程序中子程序分为如下六类：

·巧合型子程序，即一个子程序实现多个逻辑上不相关的功能。

·逻辑型子程序，即一个子程序实现一组逻辑上相关的功能，运行时由调用模块选择某一特定功能。

·过程型子程序，即一个子程序在其一个分支语句体或循环体中实现多种不同的功能。

·通信型子程序，即一个子程序顺序实现多种功能，这些功能有共同的输入或都是其他功能的输入。

·顺序型子程序，即一个子程序实现多种功能，各功能之间存在数据关系，前一个功能的结果是后一个功能的输入。

·功能型子程序，即一个子程序只实现某一特定的功能。

在非面向对象程序中抽取对象时，除功能型子程序可直接对应为候选对象的方法外，其它几种类型的子程序需处理后才能对应为候选对象的方法。因而，为了从非面向对象程序中抽取出对象，一个对象抽取方法应解决如下三个问题：①如何找出相关的子程序、类型和/数据；②如何识别与处理巧合型子程序、逻辑型子程序、过程型子程序、通信型子程序和顺序型子程序；③如何抽取出候选对象之间的继承关系。

3 现有方法评析

当前的对象抽取方法大体可分为三类：第一类方法通过程序分析，将子程序和相关的参数类型、返回值类型及其访问的全局变量封装成对象[1~6]；第二类方法则利用模块的性质来抽取对象，这类方法首先对模块的内聚度和耦合度进行定性或定量分析，然后在此基础上抽取出独立性程度较高的对象[7~11]；第三类方法运用概念分析来分析非面向对象程序中子程序和类型或数据之间的关系，将之表示为一个格，然后对格进行概念划分，一个概念对应一个待抽取的对象[12,13]。

3.1 程序分析方法

这类方法通过找出子程序和全局变量或形参/实参之间的关系来抽取对象，如S. Liu的基于全局变量和基于类型的对象抽取方法[1]、Livadas的基于接收器的对象抽取方法[4]及G. Canfora的抽象数据类型识别方法[2,6]等。在讨论这些方法之前，我们先引入如下定义[4]：

定义2 设变量x的类型是t，$f\in F$，则当x在f中定义或使用时称x在f中可见。

显然，如果x在f中可见，则x是一个全局变量、f中的局部变量或者是一个形参。

Liu的基于全局变量的对象识别方法抽取出的候选对象具有形式$(\Phi,\varnothing,x)$，其中$\Phi\subset F$，x是一个对Φ中每个子程序都可见的全局变量。基于全局变量的识别方法分三步进行对象抽取。第一步对程序中每个全局变量，找出直接使用它的子程序集合。第二步将每个全局变量涉及的子程序集合作为结点，构造一个图。如果两个结点表示的子程序集合的交集不为空，则这两个结点间有一条边相连。第三步是将图中的每一个强连通子图封装成一个对象。这种方法没有考虑Pascal这类允许嵌套过程的语言中非局部变量的存在，也没有考虑间接访问全局变量的过程，例如，全局变量x对子程序f1可见，而f1以参数x调用子程序f2，此时应认为x对f2也可见。在极端情况下，如果程序中所有子程序都访问某个相同的全局变量，则该方法会将所有子程序封装进一个对象，也即从非面向对象程序中只抽取出一个对象，这对程序的理解或转换可能没有益处。

基于类型的识别方法抽取出的对象具有形式$(\Phi,\tau,\varnothing)$，即它利用子程序的形参类型和返回值类型来分

类子程序，然后将具有相同形参和返回值类型的子程序和这些类型封装成对象。这种方法实际上是先识别某个候选对象的属性，然后找出和这些属性相关的子程序作为候选对象的方法，它有可能造成实际上不是候选对象属性的类型来分类子程序。例如，对于判断一个栈是否为空的子程序：

function Empty_Stack(S:Stack_Type):boolean

基于类型的对象识别方法将用类型 Stack_type 和 boolean 来分类子程序，实际上 boolean 并不是栈对象的属性。

Ogando 等提出了一个改进的对象抽取方法[3]，他们用类型相对复杂度来度量非面向对象程序中类型的相对复杂程度，然后利用子程序的形参类型和返回值类型中最复杂的类型来分类子程序。Livadas 等则在此基础上提出了一种新的对象抽取方法[4]，即基于接收器类型的对象识别方法，该方法注意到一个子程序可能有多个参数，但实际用来分类子程序的通常是其中的某些而不是全部参数，例如对实现入栈功能的子程序：

procedure Push(S: Stack_type; elem: Element_Type)

应该用类型 STACK_TYPE 来分类。在基于接收器类型的对象识别方法中，与一个子程序相关的接收器类型是由至少在该子程序的一条执行路径中进行写访问的参数的类型组成的集合，这些接收器类型用来分类子程序。基于接收器类型的对象识别方法与 Ogando 等定义的类型复杂度结合起来，能抽取出粒度较好的对象。

上述这些方法都是用一个二分图来模拟非面向对象程序中子程序和全局变量或类型之间的关系，在该图上每一个强连通子图对应于一个候选对象。这些方法没有区别功能型子程序和非功能型子程序。为此，Canfora 提出一个改进的对象抽取方法，该方法在识别强连通子图时利用统计函数来处理图上存在的假连接和巧合型连接，从而使能抽取出的候选对象更接近问题域中实际的对象[2,6]。在这里，巧合型连接对应于巧合型子程序，这类子程序因实现多个功能而存取多个数据结构，每个功能逻辑上属于一个对象。这类子程序因而可分割成多个子程序，每个子程序实现一种功能，逻辑上属于一个对象。而假连接对应于通信型子程序，该子程序因实现特定的功能而存取不同的数据结构，如一个子程序将栈中的元素复制到队列中，这时将其分割没有意义，所以在处理中将该子程序移去。

这类方法主要解决了对象抽取所面临的第一个问题，部分解决了第二个问题，但没有涉及到候选对象之间继承性的抽取问题。

3.2 利用模块性质抽取对象的方法

Myers 将模块独立性表述为最大化模块内部各成分之间关联关系和最小化模块之间关联关系的折中[17]。Myers、Yourdon 和 Constantine 等人认为，模块内部各成分之间的关联关系可用模块内聚度来描述，模块之间的关联关系可用模块耦合度来描述[1,7-16]。现代的面向对象程序设计方法同样强调对象的独立性，因而可从模块的性质出发，抽取出具有较高内聚度的模块，这些模块之间具有较低的耦合度，然后将之封装成对象。目前，这方面也进行了一些研究，如我们利用模块内聚度的对象抽取方法[7,8]和基于数据挖掘的对象抽取方法[9]、Pidaparthi 的资源矩阵方法[10]和 Cimitile 利用模块耦合度的对象抽取方法[11]等。

这类方法的关键在于如何将定性的模块内聚度和耦合度定量化。Cimitile 研究发现，程序分析方法不适合在由 COBOL 或 RPG 等语言编写的对数据敏感的商业软件中抽取对象，因为这类语言不象 C、Pascal 和 Ada83等语言，它们没有变量作用域的概念，子程序之间也不能通过参数传递的方式交换数据。为此，他提出了一种利用模块耦合度的对象抽取方法，该方法首先识别非面象对象程序中待抽取对象的数据结构（即对象的属性），然后将和这些数据结构相关的子程序或程序单元经过处理后对应为待抽取对象的方法。

在识别待抽取对象的方法时，Cimitile 先用一个二分图表示子程序或程序单元和数据结构之间的访问关系，每条边上的权值为子程序访问相关的数据结构的次数。然后，用一个行代表子程序、列代表数据结构的矩阵 X 表示子程序或程序单元和对象方法之间的对应关系，当一个子程序或程序单元 P 对应于一个将数据结构 D 作为属性的对象的方法时相应的矩阵入口为1，否则为0。矩阵入口为1的子程序或程序单元和数据结构在二分图上一定存在访问关系，反之，二分图上不存在访问关系的子程序和数据结构对应的入口一定为0。为使抽取出与应用域中真实对象相对应的候选对象，Cimitile 在最小化抽取出的候选对象之间的耦合度的基础上将子程序对应为对象的方法。单个候选对象与其它候选对象之间的耦合度定义为非该候选对象的方法访问其属性的强度之和，所有候选对象对应的耦合度的累加和就构成了候选对象之间的耦合度。当某个程序单元同时访问了多个对象的属性而造成不能直接对应为候选对象的方法时，Cimitile 结合耦合度和子程序调用必经结点树分解将该程序单元分解为多个对象的方法，某些子程序可能对应于候选对象的一个方法。当某个子程序访问多个对象的数据结构而不能对应为候选对象的方法时，可通过切片的方法将之分解为若干个完成特定功能的子程序，每个子程序对应为

一个候选对象的方法。

我们结合程序分析的方法，提出了一种新的利用模块内聚度的对象抽取方法[7,8]。该方法从一个边表示访问关系、边上权值表示类型复杂度的子程序-类型关系图出发，通过一系列子程序切割、图的合并和划分，最后得到若干独立的强连通子图，每个子图封装为一个候选对象，其中的子程序结点对应为候选对象的方法，类型结点对应为候选对象的属性。该方法利用模块的紧密度和重叠度来定量地衡量一个模块的内聚度，模块紧密度定义为模块内所有子程序共同涉及的类型的相对复杂度之和与模块涉及的所有类型的相对复杂度之和的比值，模块重叠度表示平均一个子程序涉及的类型之模块中所有子程序涉及的公共类型所占的复杂度比值，也即一个子程序中对公共部分处理的平均程度。一个模块的紧密度和重叠度越高，其内聚程度显然越高。当一个子程序是非功能性子程序时，我们或将之切割为实现多个独立功能的子程序，或从子程序-类型关系图上删去对应的结点和相连的边，或根据内聚度判断其应该划归给哪个模块。然后，针对特定语言Ada83，我们还初步分析了候选对象之间继承关系的抽取。

Pidaparthi 利用资源使用矩阵来表示子程序和数据结构之间的访问关系，提出了一个完全不同的对象抽取方法[10]。该方法从面向对象的观点出发，将整个非面向对象程序视为由一个内聚度较低的"God"类组成的面向对象程序。Pidaparthi 用资源使用矩阵将"God"类划分成若干个内聚度较高的类，然后在此基础上对类各种划分，以抽取出对象之间的聚集关系、关联关系和继承关系。与程序分析方法相比，该方法能抽取出粒度较好的对象。

这类方法主要解决了对象抽取所面临的前两个问题，部分地解决了第三个问题，但没有提供一个完善的继承性抽取方案。

3.3 概念分析方法

概念分析方法可用来分类实体，每一组实体具有共同的属性，表示一个概念。Birkhoff 奠定了概念分析方法的数学基础，他证明了可用一个格直观地表示实体和属性之间存在的二分关系[14]。随后，概念分析方法得到了广泛的应用，如分析软件结构、消除 C++ 程序中的死数据和重构面向对象程序的类层次等等[15,16]。概念分析方法目前被用来从非面向对象程序中抽取可能的对象[12,17]。与程序分析和利用模块性质抽取对象方法相比，概念分析抽取对象时不仅利用程序中的"正面"信息，而且还利用"反面"信息，例如某个函数 f 访问了栈 Stack 中的某个域但没有访问队列 Queue 中的域，此外，给定某个特定的非面向对象程序，程序分析和利用模块性质的对象抽取方法只能得到一个候选对象集合，而概念分析方法可抽取出多种可能的候选对象集合，软件重构时可根据需要选择某个具体候选对象集合。

Siff 的概念分析抽取对象方法中，实体对应子程序，属性对应子程序的特性，抽取步骤分三步。首先，根据非面向对象程序建立一上下文，即建立一个矩阵，其行表示子程序，其列表示子程序的特性，如使用一个队列或不返回栈等"正面"或"反面"信息。接着根据上下文建立一个格。最后在格上进行概念划分。按 Siff 的方法，一个格可能得到多个概念划分，每一个概念划分对应一个候选对象集合，用户可根据需要选择某个概念划分。

概念分析方法的特点同时也是其缺点，抽取对象时需仔细的定义实体即子程序的特性。如果非面向对象程序规模较大时，得到的概念划分数目相应地增多，给用户选择造成困难。

4 实例研究

为直观地研究现有的三类对象抽取方法之间的差别，我们用基于接收器类型的对象识别方法、利用模块内聚度的对象抽取方法和概念分析方法抽取附录(略)所示的 Pascal 程序中的对象，该程序用两个栈来模拟一个队列。

按基于接收器类型的对象识别方法，应该用 Pascal 子程序变参和子程序体中访问的全局变量的类型类型分类子程序，抽取出的对象如下：

$O^{w}_{Receiver}$ = {O_1=({Init},{Stcque,Stack}),

O_2=({Push_S,Pop_S},{Stack}),

O_3=({Insert_Q,Delete_Q,Copy},{Stcque})}

抽取出的三个对象中，O_1 没有实际意义，子程序 Empty_Q 没有对应为队列 Stcque 的方法，子程序 Empty_S 没有对应为栈 S 的方法。

利用模块内聚度的对象抽取方法首先计算各类型的相对复杂度，然后将模块内聚度大于阈值的子图加入 OverStep 列表，最后对列表中子程序结点集交集不为空的子图进行划分处理。设基本类型(即标量类型)的相对复杂度为1，则附录 A 所示程序中各类型的相对复杂度为：

$$C_{integer,0}=1$$

$$C_{boolean,0}=1$$

$$C_{Stack,0}=C_{integer,1}+C_{El\_Type,1}=2+2*C_{integer,2}=8$$

$$C_{Stcque,0}=C_{Stack,1}+C_{Stack,1}=2*(C_{integer,2}+C_{El\_Type,2})=22$$

于是，与该程序对应的子程序-类型图如图1所示。

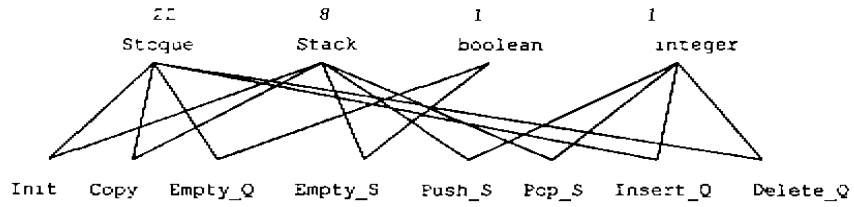

图1 子程序-类型关系图

文[7]和[8]中没有具体定义模块内聚度和模块紧密度与重叠度之间的关系，这里我们取内聚度为模块紧密度和重叠度的最小值，于是可在上面所示子程序-类型关系图上计算出各子图的内聚度：

Cohesion(M_{Stcque}) = min{Tightness(M_{Stcque}), Overlap(M_{Stcque})} = min{0.8527, 0.8672} = 0.8527

Cohesion(M_{Stack}) = min{0.4598, 0.6400} = 0.4598

Cohesion($M_{boolean}$) = min{0.0625, 0.0773} = 0.0625

Cohesion($M_{integer}$) = min{0.0625, 0.0773} = 0.0625

Canfora 通过大量的统计分析表明，模块内聚度阈值 Step 为0.3 就可抽取出粒度较好的对象，因此 OverStep = {M_{Stcque}, M_{Stack}}。子图 M_{Stcque} 和 M_{Stack} 子程序结点集的交集为{Init, Copy}，通过程序分析知，Init 是巧合型子程序，Copy 为通信型子程序，于是应将 Init 分割成两个子程序 Init_Q 和 Init_S，它们分别初始化队列和栈，而从这两个子图中去掉结点 Copy 和相应的边。最后抽取出的对象为：

$O^{F}_{Cohesion}$ = {O_1 = ({Init_S, Empty_S, Push_S, Pop_S}, {Stack}), O_2 = ({Init_Q, Empty_Q, Insert_Q, Delete_Q}, {Stcque})}

该方法抽取出一个栈对象和一个队列对象，显然比基于接收器类型的对象识别方法得到的结果合理。

一个概念是具有一组属性的最大对象集合，用二元式(对象集合，属性集合)表示。Siff 的概念分析方法首先要选择用于概念分析的属性，这里使用属性"有队列类型的形参"、"有栈类型的形参"，"访问队列的域"和"访问栈的域"。当直接应用概念分析方法抽取对象时，得到的所有概念的外延中都包含 Init，因而所有概念的外延交集都不为空，导致概念划分失败。为能进行原子划分，我们先将子程序 Init 分割成两个子程序 Init_Q 和 Init_S，分别实现队列和栈的初始化，然后增添一反面属性"没有栈类型的形参"，处理后得到的上下文如表1所示。然后按格的构造方法，容易构造出图2所示的概念格，其中各概念的外延和内延由表2所示。最后，应用概念划分方法得到的概念划分如表3所示，相应的三组对象为：

$O^{F}_{Concept}(1)$ = {O_1 = ({Init_S, Empty_S, Push_S, Pop_S}, {Stack}), O_2 = ({Init_Q, Empty_Q, Insert_Q, Delete_Q}, {Stcque}), O_3 = ({Copy}, {Stack, Stcque})}

$O^{F}_{Concept}(2)$ = {O_1 = ({Init_S, Copy, Empty_S, Push_S, Pop_S}, {Stack}), O_2 = ({Init_Q, Empty_Q, Insert_Q, Delete_Q}, {Stcque})}

$O^{F}_{Concept}(3)$ = {O_2 = ({Init_S, Empty_S, Push_S, Pop_S}, {Stack}), O_3 = ({Init_Q, Copy, Empty_Q, Insert_Q, Delete_Q}, {Stcque})}

表1 上下文

	有队列类型的形参	有栈类型的形参	访问队列的域	访问栈的域	没有栈类型的形参
Init_S		√		√	
Init_Q	√		√		√
Copy	√	√	√		
Empty_Q	√		√		√
Empty_S		√		√	
Push_S		√		√	
Pop_S		√		√	
Insert_Q	√		√		√
Delete_Q	√		√		√

表2 概念

top	({Init_S, Init_Q, Copy, Empty_Q, Empty_S, Push_S, Pop_S, Insert_Q, Delete_Q}, ∅)
C_4	({Init_Q, Copy, Empty_Q, Insert_Q, Delete_Q}, {访问队列的域})
C_3	({Init_S, Copy, Empty_S, Push_S, Pop_S}, {有栈类型的形参})
C_2	({Copy}, {有队列类型的形参，有栈类型的参数，访问队列的域})
C_1	({Init_Q, Empty_Q, Insert_Q, Delete_Q}, {有队列类型的形参，访问队列的域，没有栈类型形参})
C_0	({Init_S, Empty_S, Push_S, Pop_S}, {有栈类型的形参，访问栈的域})
Bot	(∅, {有队列类型的形参，有栈类型的形参，访问队列的域，访问栈的域})

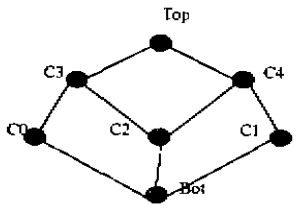

图2 概念格

表3 概念划分

P0	{C0,C1,C2}
P1	{C1,C3}
P2	{C0,C4}

概念分析方法得到的第一组对象中，子程序 Copy 不划归给栈和队列，而作为第三个对象的方法，第二组和第三组对象分别将 copy 划归给栈和队列。

由此可以看出，基于接收器类型的对象抽取方法没有对非功能型子程序 Init 和 Copy 进行处理，且子程序 Empty_S 和 Empty_Q 没有分别对应为候选对象栈和队列的方法，得到的候选对象与应用域中真正的对象相距甚远。利用模块内聚度的对象抽取方法将非功能型子程序或分割或抛弃，最后得到有实际意义的候选对象栈和队列。不同于前两类方法，概念分析方法在一个非面向对象程序中抽取多组候选对象，给用户提供了多种选择的可能性。但这种方法要求对某些非功能型子程序进行预先处理和仔细地选择属性来构造上下文，否则将导致概念分析方法失败。例如上例中如果不先分割巧合型子程序 Init 或不增加属性"没有栈类型"，都将导致该方法失败。

结论与展望 上述各类方法抽取出的候选对象与应用域中真实的对象还有一定的差距，如何缩小这种差距、如何处理非功能型子程序及如何抽取候选对象之间的继承性等问题都有待进一步研究。目前，我们正进行 Ada83实现的非面向对象程序到 Ada95实现的面向对象程序转换的研究，并准备实现一个原型系统，其核心由类内聚度度量准则[15]、对象抽取算法、服务性任务到保护对象的转换[20]和继承性抽取等几部分组成。当前的各种对象抽取方法都缺乏坚实的理论基础，我们期望通过该项目的研究能在对象抽取方面尤其在继承性抽取的理论研究上取得一些进展。

参 考 文 献

1 Liu S, Wilde N. Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery. In: Proc. of the IEEE Conf. on Software Maintenance, San Diego, CA: IEEE Computer Society Press, 1990. 266～271

2 Canfora G, Cimutile A, Munro M. A Reverse Engineering Method for Identifying Reusable Abstract Data Types. In: Proc. of the First IEEE Working Conf. on Reverse Engineering, Baltimore, Maryland. IEEE Computer Soc. Press, 1993. 73～82

3 Ogando R, Yau S, Liu S, Wilde N. An Object Finder for Program Structure Understanding in Software Maintenance. Software Maintenance: Research and Practice, 1994, 6: 261～283

4 Livadas P, Johnson T. A New Approach to Finding Objects in Programs. Journal of Software Maintenance: Research and Practice, 1994, 6: 249～260

5 Purtilo J, Swiss T, White E. Extracting Program Structure for Packaging in a Component-based Environment. Computer Language, 1995, 21(1): 39～48

6 Canfora G, Cimutile A, Munro M. An Improved Algorithm for Identifying Reusable Objects in Code. Software Practice and Experience, 26(1): 24～48

7 Zhou Y, Xu B. Extracting Objects of Ada Programs Using Module Features. In: Proc. of IEEE Conf. on Software Maintenance, IEEE Society Press, Oxford, 1999, 23～29

8 周毓明，徐宝文. 一种利用模块内聚性的对象抽取方法. 软件学报, 2000, 11(4): 557～562

9 Li S, Zhou Y, Xu B. An Object Extraction Model Using Association Rules. In: Proc. of Intl. Symposium on Software Engineering (ISES'01), Wuhan, China, 2001, also appear in: Journal of Wuhan University, 2001

10 Pidaparthi S, Zedan H, Luker Z. Resource Usage Matrix in Object Identification and Design Transformation of Legacy Procedural Software. In: 14th Automatic Software Engineering Conference

11 Cimutile A, Lucia A, Lucca G, Fasolin A. Identifying Objects in Legacy Systems Using Design Metrics. The Journal of Systems and Software, 1999, 44(3): 199～211

12 Siff M, Reps T. Identifying Modules via Concept Analysis. In: Proc. of IEEE Conf. on Software Maintenance, Bari, Italy, 1997. 170～179

13 Snelting G. Reengineering of Configurations Based on Mathematical Concept Analysis. ACM Transactions on Software Engineering and Methodology, 1996, 5(2): 146～189

14 Birkhoff O. Lattice Theory, American Mathematical Society, 1940

15 Godin R, Mili H. Building and Maintaining Analysis-level Class Hierarchies Using Galois Lattices. ACM SIGPLAN Notices. 28(10): 394～410

16 Snelting G, Tip F. Reengineering Class Hierarchies Using Concept Analysis. Software Engineering Notes, 1998, 23(6): 99～110

17 Myers G J. Composite/Structured Design, Van Nostrand Reinhold Co, New York, 1978

18 Yourdon E, Constantine L L. Structured Design Fundamentals of a Discipline of Computer Program and Systems Design (2nd Edition), Yourdon Press, New York, 1979

19 Zhou Y, Xu B. Comments on Chae's cohesion measure for classes. Software Practice & Experience, 2001 (to appear)

20 Li B, Xu B, Yu Huiming. Transforming Ada Serving Tasks into Protected Objects. In: Proc. of SIGAda'98, 1998, 240～245