ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.11, November 2010, pp.2725−2737
doi: 10.3724/SP.J.1001.2010.03680

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于形态分析识别设计模式中的集中管理式聚集*

周晓宇[1], 钱 巨[2,3], 陈 林[4,5], 徐宝文[1,4,5+]

[1](东南大学 计算机科学与工程学院,江苏 南京 211189)

[2](上海市计算机软件评测重点实验室,上海 201112)

[3](南京航空航天大学 信息科学与技术学院,江苏 南京 210016)

[4](南京大学 计算机科学与技术系,江苏 南京 210093)

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

Identification of Centrally Managed Aggregations in Design Patterns Using Shape Analysis

ZHOU Xiao-Yu[1], QIAN Ju[2,3], CHEN Lin[4,5], XU Bao-Wen[1,4,5+]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

[2](Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China)

[3](College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[4](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

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

+ Corresponding author: E-mail: bwxu@nju.edu.cn, http://cs.nju.edu.cn/teacher/show.aspx?id=183

Zhou XY, Qian J, Chen L, Xu BW. Identification of centrally managed aggregations in design patterns using shape analysis. *Journal of Software*, 2010,21(11):2725−2737. http://www.jos.org.cn/1000-9825/3680.htm

Abstract: In this paper, a shape-analysis based approach is proposed to automatically identify aggregations that are implemented using the commonly used implementation mechanism, pointers or references. First, this paper augments predicates of Sagiv's three-valued logical structure to describe the semantic constraints for the central aggregation management operations on linked lists. Then, this paper presents a method to identify the aggregation management behavior by analyzing the changes of shape structures for linked lists along control flow paths. Finally, the effectiveness of the proposed 1-*n* aggregation identification approach is proposed using a case study from the open-source software JEdit.

Key words: aggregation; design pattern; shape analysis; centrally managed; identification

摘 要: 针对常见的利用指针或引用的聚集实现方式,提出一种基于形态分析的一对多聚集关系的自动识别方法.首先,扩充 Sagiv 的三值逻辑结构中的谓词以描述链表上聚集管理操作的语义特征.然后,给出基于控制流上链表

* Supported by the National Natural Science Foundation of China under Grant Nos.90818027, 60633010, 60873050, 60873049, 60803008, 60903026 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2009AA01Z147 (国家高技术研究发展计划(863)); the Jiangsu Provincial Natural Science Foundation of China under Grant Nos.BK2006094, BK2008292 (江苏省自然科学基金); the Opening Project of Shanghai Key Laboratory of Computer Software Evaluating and Testing of China under Grant No.09DZ2272600 (上海市计算机软件评测重点实验室开放项目)

Received 2009-03-09; Accepted 2009-07-07

形态结构的变化识别聚集管理行为的方法.最后,通过开源软件 JEdit 中的实例分析展示了该方法的有效性.

关键词: 聚集;设计模式;形态分析;集中式管理;识别

中图法分类号: TP311 **文献标识码:** A

设计模式是一种广泛使用的程序框架技术.从源程序中自动识别设计模式,不仅可以提高设计的可追踪性,避免在维护过程中破坏已有模式,而且有助于充分利用它们的复用和扩展潜力.在备忘录、组合和观察者模式的权威描述中,均存在类间的一对多聚集关系[1].当从源代码中识别这些设计模式时,首先需要正确地识别出类间的聚集关系.

然而,从源代码中识别类间的一对多聚集关系并不容易,其主要原因在于聚集不存在直接对应的语法结构,不同的程序可能采取不同的实现方式.在实践中,有 3 种常见的聚集实现方式:

- 在第 1 种方式中,“一”方的类包含一个类型为数组的成员变量,该数组元素的类型为“多”方的类;
- 在第 2 种方式中,“一”方的类包含一个类型为预定义容器的成员变量,该容器用来存储“多”方类的对象;
- 在第 3 种方式中,“一”方的类包含一个类型为指针或引用的成员变量,该指针或者引用指向一个元素类型为“多”方类的链表.

对于前两种代码,先前的设计模式识别工作一致地将它们识别为聚集关系[2−5];对于后一种代码,先前工作要么一律将其界定为关联,要么一律将其界定为聚集[3,5,6].总体上,先前的工作大多在语法结构的基础上识别聚集,而忽略了聚集关系的语义.实际上,尽管聚集关系可以通过这些数据结构来实现,但并非所有这样的数据结构都形成聚集关系.

为了提高识别结果的准确性,应该结合语义而不应仅从语法结构上进行一对多聚集关系的识别.我们通过分析发现,在备忘录模式、组合模式和观察者模式中,一对多聚集关系都是由“一”方的对象对一组“多”方对象的管理行为而确立的.在下文中,这样的聚集称为集中管理式的聚集.对实现这种聚集的程序而言,不管采用上述哪种实现方法,“一”方的类中都包含对“多”方的一组对象进行管理的操作.从源代码中识别这样的聚集关系,关键是要能够识别出相应的聚集管理操作.

针对上述第 3 种实现方式,本文提出了一种基于形态分析的集中管理式聚集识别方法.为简化描述,本文以识别用指针或引用方式实现的备忘录模式中的一对多聚集为例,描述具体的识别方法.形态分析是研究链表结构的一种重要的语义分析技术,其基本思想是,将语义特性相同的对象合并成同一个抽象个体,以通过有限的个体描述数量任意增长的链表结构.特别地,可以用不同的谓词来表达不同的语义特性.在本文中,我们在形态分析技术的基础上通过分析链表结构的变化来识别链表上是否具有特定的管理操作.为此,本文首先对现有的三值逻辑结构的谓词进行扩充,以描述链表上聚集管理操作的语义特征.然后,给出根据不同程序点上链表形态结构的变化识别链表管理操作的方法.最后,通过开源软件 JEdit 中的实例分析展示该识别方法的有效性.

1 设计模式中的集中管理式聚集

在指针或引用实现方式下,一对多的聚集关系通常由类 *A* 通过成员变量引用类 *B* 的链表实现.实际上,许多设计模式的典型实现方式中都包含这种链表结构,但是并非所有这种链表结构都形成聚集关系.在设计模式概念的提出者 Gamma 等人的描述中,备忘录模式中原发器(originator)或备忘录管理者(caretaker)引用备忘录对象链表及观察者模式中目标引用观察者链表均形成一对多的聚集关系[1].然而在职责链模式中,客户引用处理者链表并不构成聚集关系[1].在组合模式中,对象被组合成树型数据结构[1],其中同时存在形成聚集关系的链表和不形成聚集关系的链表.

在备忘录模式中,原发器不断地生成备忘录对象以记录自身的历史信息,在需要时根据备忘录恢复自身的状态.为此,需要将新的备忘录对象添加到备忘录链表中,在需要时按时间顺序遍历链表获取需要的备忘录对象.这些行为是实现模式语义所必需的,我们称其为模式行为.在观察者模式中,一个目标通常有多个观察者.观察者向目标登记观察兴趣的行为就表现为向该目标所引用的观察者链表添加成员的操作,目标在状态变化后

对每个观察者的通知就需要在遍历链表的过程中实现.可见,在这两个模式中,链表的形成和使用都有统一的管理者,链表的添加和遍历等集中管理操作均由相应的模式行为规定,是实现模式语义必不可少的.我们把这种链表的管理者和被管理者之间形成的聚集关系称为集中管理式的聚集关系.

职责链模式的典型实现方式是客户引用处理者链,但二者并不构成聚集关系.该模式的主要语义特征是客户(client)的要求可以由处理者(handler)自行解决或由处理者自行链接后续者并将处理责任传递给后续者.这样,链表由 Handler 对象自发形成,而不是由 Client 通过集中管理建立.在该模式中,消息的传递由链表成员采用接力方式自主完成,而不是由Client在集中遍历各Handler的过程完成.总之,职责链模式不要求聚集管理操作的存在.组合模式将对象组合成树型数据结构,树型结构由兄弟关系和父子关系两种链表组合而成.其中,组合对象负责管理它所引用的兄弟关系链表,二者形成聚集关系.而父子关系链表属于自发形成,不构成聚集关系.

通过以上两方面的分析可以发现,正是组合、备忘录和观察者模式中对链表集中管理的模式行为确定了链表管理者与链表成员间的聚集关系.识别这样的聚集关系一方面要识别链表及其引用者,另一方面需要识别引用者是否承担链表管理操作.链表的管理操作可能多种多样,其中添加和遍历操作是建立和使用链表的两种必需操作,是本文的主要分析对象;其他管理操作(如顺序调整、删除等)并非必备操作,通常不作为是否形成聚集的评判依据[7],但本文介绍的方法同样适用于描述和识别这些操作.在不同的上下文环境下,添加和遍历操作上的约束也可能不同.在组合、备忘录、观察者这 3 个模式中,备忘录聚集管理操作需要满足更为严格的约束.本文以备忘录模式为例,讨论通过形态分析描述与识别聚集管理操作的语义特征的方法.

2 通过形态分析描述与识别聚集管理操作的语义特征

本节以备忘录模式为例,讨论聚集管理操作语义特征的描述与识别.在第 2.1 节中,用自然语言描述备忘录聚集管理操作的时序特征和识别流程.聚集管理操作的语义特征需要通过不同程序点上链表形态结构的变化来识别.为此,我们在第 2.2 节介绍了基于三值逻辑结构的形态分析技术,设计了结构间谓词以表达同一对象在不同结构中的可追踪性和可达关系.备忘录链表的典型实现方式是双向链表,双向链表可以看作相反方向的两个单向链表的组合.为简化表述,第 2.3 节~第 2.5 节首先通过形态分析给出了单向链表管理操作语义特征的描述和识别方法;然后,在第 2.6 节中设计了描述双向链表的谓词.这样,只需将单向链表的谓词更换为双向链表的对应谓词.第 2.3 节~第 2.5 节中的描述和识别方法就可以适用于双向链表.

2.1 备忘录聚集管理操作的时序特征及识别过程

备忘录链表的管理由原发器(originator)或专门的备忘录管理者(caretaker)负责[1].在数据结构上,为按照时间先后存、取备忘录,备忘录对象必须按时间顺序排列,并提供双向遍历途径.双向链表为最常见的实现方式,其管理操作必须满足如下约束:

(1) 新生成的备忘录对象只能添加于链表正向的末端.备忘录聚集不应存在向该方向首部或中间插入成员的操作——这意味着篡改历史,只能存在两种添加新成员的操作:
   ① 尾端添加操作,即在链表正向的末端加挂新记录;
   ② 尾端子链表替换操作,用新的记录替换某一位置之后的所有成员,对应着原发器在若干步撤销之后的一个新的正常操作.

(2) 除了上述对链表结构的调整以外,不存在对链表成员变量的写操作,这样的操作意味着篡改历史记录.

(3) 原发器的撤销和重复操作分别按相反方向进行局部遍历,通常由游标实现.

设计模式中的集中管理式聚集具有一个共同的特点,即“多”方成员均具有共同的基类[1].在链表方式下,“多”方类包含对自身或祖先类的引用.为方便表述,下文借用 Ada 语言的术语将类 O 及其派生类的集合称为 O 类域[8].这样,要识别聚集关系,首先就要通过语法分析找出这种能够形成同类域对象链表的类.

链表的管理者是链表的引用者,但并非所有对链表的引用者都是链表管理者.例如,游标变量通常就不是链表管理者.链表管理者通常是链表根节点的稳定的引用者,由此可以到达链表的所有节点.因此,在形态分析的

过程中需要识别链表根节点及其稳定的引用者.

备忘录聚集管理操作的识别流程如下:

(1) 根据语法分析识别能够形成同类域对象链表的类,要求该类中包含两个以上引用自身或祖先类的成员变量,这样的类才可能形成双向链表.

(2) 对满足以上条件的所有类 O,通过语法分析,寻找通过成员变量引用类 O 的类.

(3) 设类 C 引用类 O.对类 C 的所有方法进行形态分析,并在此过程中验证 C 所引用的 O 类域链表是否形成双向链表,是否通过特定成员变量稳定地引用根对象,是否存在向链表中间或首端插入节点的操作,是否存在对链表成员变量的写操作(意味着篡改历史记录——更改链表结构的除外),是否符合尾部添加操作、尾部子链表替换操作或遍历操作的特征.

对各方法的形态分析要求给出方法入口处的形态结构.针对这个问题,Rinetzky 等人提出了一种基于对象稳定状态和规范使用的模块化形态分析方法[9].

2.2 三值逻辑结构与结构间谓词

Sagiv 等人采用三值逻辑结构$\langle U,\iota\rangle$表达链表成员间的引用结构.其中,U 为个体(individual)组成的论域,个体分为确定个体和不确定个体两类.每个确定的个体对应一个运行时对象,每个不确定的个体对应于 1 个以上的运行时对象(下文中,在不造成歧义的情况下,我们对个体和对象不作严格区分);谓词表 P 中的每个谓词 p 描述论域中个体之间的引用关系,ι为 P 中的谓词表达式应用于各个体后的取值.该结构采用 Cleane 的三值逻辑,即用 1 和 0 表达确定的真值,用 1/2 表达不确定的真值[10,11].通常用符号σ代表具体的三值逻辑结构(下文简称形态结构).形态图(shape graph)是形态结构的直观表达,每个节点对应一个个体,实线表示的普通节点和虚线表示的缩略节点(summary node)分别对应确定和不确定个体,节点之间的实线和虚线有向边分别对应确定和不确定的引用关系,若个体 u 被变量 x 直接引用,则存在一条由 x 指向 u 的边[10,11].图 1 给出了 4 个形态结构,在形态结构$\sigma(s_1)$中,u_1 和 u_3 为确定个体,分别对应 1 个对象;u_2 为不确定个体,对应 1 个以上的对象.

三值逻辑结构的谓词分为核心谓词、辅助谓词和宏谓词 3 类.其中,核心谓词(core predicate)表达程序设计语言在对象引用方面的语义.例如,在如图 1 所示的形态结构$\sigma(s_1)$中,有 $x(u_1)=1,p(u_1,u_2)=1/2,p(u_2,u_2)=1/2$,它们表示 x 确定地引用 u_1 代表的对象,$u_1.p$ 对 u_2 所代表的对象的引用不一定存在,u_2 所代表的对象之间可能通过 p 相互引用.辅助谓词(instrumentation predicate)可由核心谓词定义,表达特定分析所必需的语义信息.宏谓词(macro predicate)由核心谓词和辅助谓词定义,主要是为表述方便而设置,并非必需.另一方面,谓词也可以根据其自由变量的数量分为一元谓词和多元谓词.一元谓词用来描述个体的属性,所有一元谓词(包括核心谓词和辅助谓词)取值相同的对象被合并为一个缩略个体;而多元谓词描述不同个体间的引用关系[10,11].

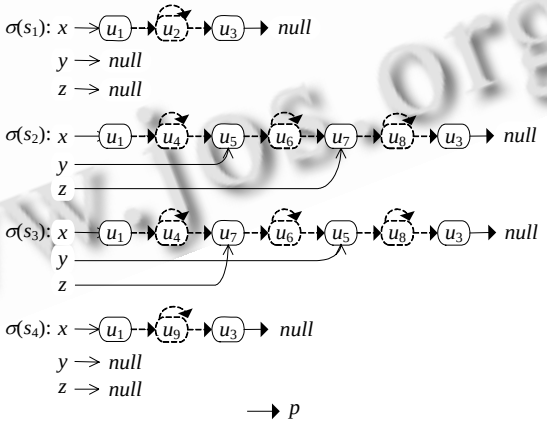

Fig.1 Operation to exchange positions of two objects

图 1 互换两个对象位置的操作

当程序中的语句执行时,链表的结构可能会发生变化.因此,在形态分析中需要针对基本的引用赋值操作给出各谓词的变迁函数,然后通过遍历程序的控制流图,根据程序的初始状态和变迁函数,逐步得到各程序点上的形态结构.在结构变迁中,不确定个体可能会分化为多个个体,多个个体也可能会由于一元谓词取值相同而被融合为一个新的个体.为表达同一对象在不同结构中的可追踪性,我们设计了分化谓词 $FocusTo(s_1,u_1,s_2,u_2)$、合并谓词 $MergeTo(s_1,u_1,s_2,u_2)$、流入谓词 $InFlow(s_1,u_1,s_2,u_2)$.为表达不同结构中对象间的可达关系,我们设计了谓词 $R[p](s_1,u_1,s_2,u_2)$.这 4 个谓词统称为结构间谓词.其中,分化谓词和合并谓词为辅助谓词,流入谓词和结构间可达谓词为宏谓词.上述谓词变元中的 s_1 和 s_2 是程序控制流图中的节点,对应程序中的语句执行前后的状态.节点 s 对应的形态结构记为 $\sigma(s)$,采用 Sagiv 等人的方法将谓词表达式 φ 在状态 $\sigma(s)$ 下的取值记作 $\varphi@\sigma(s)$.

下面首先给出表达对象可追踪性的谓词:

(1) 如果有控制流图上的两个点 $s_1,s_2,s_1\in prev^+(s_2)$(表示在控制流图中,$s_1$ 是 s_2 的直接或间接前驱),$\sigma(s_1)$中的个体 u 被分化为 $\sigma(s_2)$中的个体$\{u_1,u_2,\ldots,u_n\}$,则 $FocusTo(s_1,u,s_2,u_i)=1$,其中,$1\leqslant i\leqslant n$.

(2) 如果有控制流图上的两个点 $s_1,s_2,s_1\in prev^+(s_2)$,$\sigma(s_1)$中的个体$\{u_1,u_2,\ldots,u_n\}$在 $\sigma(s_2)$中被合并为个体 u,则 $MergeTo(s_1,u_i,s_2,u)=1$,其中,$1\leqslant i\leqslant n$.

图 1 给出了一个对象交换位置的过程.s_1~s_4 是控制流图中顺序执行的 4 个程序点,$\sigma(s_1)$~$\sigma(s_4)$是相应程序点上的形态结构.$\sigma(s_1)$中的 u_2 在 $\sigma(s_2)$中被分化成 u_4~u_8 这 5 个个体.在 $\sigma(s_3)$中,u_5 和 u_7 交换位置.在 $\sigma(s_4)$中,u_4~u_8 这 5 个个体由于在 $x(u)$,$y(u)$和 $z(u)$这 3 个一元谓词上的取值均为 0,因此被合并为不确定个体 u_9.在此过程中,有

$$FocusTo(s_1,u_2,s_2,u_i)=1,MergeTo(s_3,u_i,s_4,u_9)=1,$$

其中,$4\leqslant i\leqslant 8$.

FocusTo 和 *MergeTo* 关系由形态结构变迁中的分化操作和个体合并操作给出.Sagiv 等人给出了求取形态结构变迁的基本方法,其中,分化操作出现于谓词表达式确定化的过程中,合并操作出现于变迁后的清理过程中[10,12].*FocusTo* 和 *MergeTo* 显然满足传递性,即

$$FocusTo(s_1,u_1,s_2,u_2)\wedge FocusTo(s_2,u_2,s_3,u_3)\Rightarrow FocusTo(s_1,u_1,s_3,u_3),$$

$$MergeTo(s_1,u_1,s_2,u_2)\wedge MergeTo(s_2,u_2,s_3,u_3)\Rightarrow MergeTo(s_1,u_1,s_3,u_3).$$

为了综合表达个体分化及合并过程中的可追踪性,我们给出以下定义:

$$InFlow(s_1,u_1,s_2,u_2)\Leftrightarrow FocusTo(s_1,u_1,s_2,u_2)\vee MergeTo(s_1,u_1,s_2,u_2).$$

显然,*InFlow* 关系也具备传递性.需要说明的是,此处的 *InFlow* 关系不同于 Manevich 等人在文献[11]中定义的 *FlowTo* 关系,后者描述的是两个栈变量所关联的链表之间存在交集的现象.

若存在 s_1 到 s_2 之间的可执行路径,则我们将结构间可达谓词 $R[p](s_1,u_1,s_2,u_2)$定义为

$$R[p](s_1,u_1,s_2,u_2)\Leftrightarrow RoA[p](u_1,v_1)@\sigma(s_1)\wedge R[p](v_2,u_2)@\sigma(s_2)\wedge InFlow(s_1,v_1,s_2,v_2),$$

其中,*R* 和 *RoA* 是表达同结构内可达关系的谓词[10,11];$R[p](v_2,u_2)$表示由 v_2 出发,通过 1 次以上的 p 引用可以可达 u_2;$RoA[p](u_1,v_1)$定义为 $Aliased(u_1,v_1)\vee R[p](u_1,v_1)$,即 u_1 和 v_1 之间存在别名关系或 v_1 由 u_1 通过 p 引用可达.此定义中,v_1 和 v_2 是 u_1 和 u_2 之间可达关系的中介.该定义考虑到了 u_1 所代表的对象在 $\sigma(s_2)$中脱离链表的情况.例如,在图 2 中,s_1~s_3 是控制流上的 3 个节点,有 $s_1\in prev(s_2)$,$s_2\in prev(s_3)$,$\sigma(s_1)$中的个体 u_2 在 $\sigma(s_2)$中被删除.同时,u_5 被添加到链表尾端,u_3 和 u_4 在 $\sigma(s_3)$中被融合为 u_6.在图 2 中,有

$$RoA[p](u_2,u_3)@\sigma(s_1)\wedge R[p](u_6,u_5)@\sigma(s_3)\wedge InFow(s_1,u_3,s_3,u_6)=1,$$

因此,有 $R[p](s_1,u_2,s_3,u_5)=1$.

显然,结构间可达关系具有传递性,即,若 s_1,s_2 和 s_3 是控制流上的 3 个节点,$s_1\in prev(s_2)$,$s_2\in prev(s_3)$,则

$$R[p](s_1,u_1,s_2,u_2)\wedge R[p](s_2,u_2,s_3,u_3)\Rightarrow R[p](s_1,u_1,s_3,u_3).$$

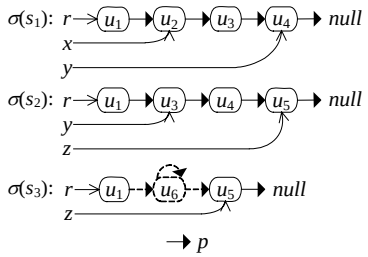

Fig.2 Reachability between individuals belonging to different logical structures

图 2 属于不同逻辑结构的个体间的可达性

2.3 尾端添加操作语义特征的描述与识别

一个链表可能有多个引用者,在某个方向上可达对象最多的引用称为该方向的根引用,被引用的对象称为根对象,用根谓词 $IsRoot[p](u)$表示[11].链表的成员即为由根对象可达的对象.作为一种特殊情况,单节点链表中的节点也可以看作是根对象.为识别尾端添加操作,我们引入谓词 $NextNull[p](u)$,表示个体 u 是 p 引用形成的链表的尾节点[11].设 s 和 s'是控制流图中的两个节点,$s\in prev^+(s')$.若满足以下 3 项条件,我们认为 s 和 s'之间的语句把对象 w 挂入了以 x 为根引用的链表尾端:

(1) 有唯一的对象 w,在$\sigma(s)$中由 x 不可达,但在$\sigma(s')$中由 x 可达,并且处于链表的尾端.

(2) 所有在$\sigma(s)$中属于该链表的对象在$\sigma(s')$中仍然属于该链表.

(3) 在 s 与 s'之间的路径上,链表成员间的顺序保持不变.

条件(1)可以表达为

$$\exists w:((IsRoot[p](x)\wedge\neg R[p](x,w))@\sigma(s)\wedge(IsRoot[p](x)\wedge R[p](x,w)\wedge NextNull[p](w))@\sigma(s'))=1 \text{ 且 } w \text{ 唯一}.$$

条件(2)可以表达为

$$\exists v:((IsRoot[p](x)\wedge RoA[p](x,v))@\sigma(s)\Rightarrow(IsRoot[p](x)\wedge RoA[p](x,v'))@\sigma(s)\wedge InFlow(s,v,s',v'))=1.$$

就条件(3)而言,在形态分析中仅凭$\sigma(s)$和$\sigma(s')$的比较可能无法识别其间发生的对象顺序的变化,因为个体合并可能湮没一些对象间的顺序信息.例如在图 1 中,仅比较$\sigma(s_1)$和$\sigma(s_4)$就无从识别其间的对象位置交换操作.因此,必须在 s 到 s'的控制流路径上逐点比较,考察是否存在对象位置交换操作.我们将条件(3)表达为:对于由 s 到 s'的路径上的任意一点 s_1,若

$$\exists u_1,u_2:(IsRoot[p](x)\wedge RoA[p](x,u_1)\wedge R[p](x,u_2)\wedge R[p](u_1,u_2))@\sigma(s_1)=1,$$

则不存在由 s_1 到 s'的路径上的点 s_2,使得

$$(IsRoot[p](x)\wedge RoA[p](x,u_2)\wedge R[p](x,u_1)\wedge R[p](u_2,u_1))@\sigma(s_2)=1.$$

2.4 尾端子链表替换操作语义特征的描述与识别

尾端子链表替换操作(下文简称尾端替换操作)首先需要将链表解裂,然后将新对象挂入解裂点尾端.因此,识别尾端替换的重点在于识别解裂操作及解裂点.图 3 给出了一个典型的尾端子链表替换操作.在$\sigma(s_2)$中,$\sigma(s_1)$中的 u_2 被分化为 u_4,u_5 和 u_6;$\sigma(s_3)$中,链表从 u_5 处解裂;在$\sigma(s_4)$中,对象 w 被挂入链表尾部;在$\sigma(s_5)$中,u_4,u_5 合并为 u_7.尾端子链表替换操作的识别特征如下:

若存在控制流上的两个点 s 和 s',$s\in prev^+(s')$,当满足以下 4 个条件时,我们认为从 s 到 s'的语句用对象 w 替换了以 x 为根引用的链表的尾部片段:

(1) 有唯一的对象 w,在$\sigma(s)$中由 x 不可达,在$\sigma(s')$中处于由 x 引用的链表的尾端.

(2) 在$\sigma(s)$中有对象 u,自链表根节点至 u 之间(含 u)的所有对象在$\sigma(s')$中仍然属于该链表,u 之后至尾端的所有对象在$\sigma(s')$中均不属于该链表.

(3) 在$\sigma(s)$和链表解裂操作之前的路径上没有对象从链表中删除,对象的顺序保持不变.

(4) 在解裂操作之后至 s'之间的路径上,自链表根节点至 u 之间的对象未被删除,且顺序保持不变.

若要识别在控制流上的点 s 与 s'之间的语句是否实现了尾端子链表替换操作,需要首先识别对象 w,然后在控制流中寻找解裂操作和尾端添加操作对应的语句,并分析整个操作中链表成员和顺序的稳定性.其过程如下:

(1) 根据尾端添加操作的第(1)项条件识别被挂入的对象 w.

(2) 自 s'开始逆着控制流检查各节点的形态结构,找到 w 被挂入链表前后的控制流节点 s_{b_attach} 和 s_{a_attach}, $s_{b\_attach}\in prev(s_{a\_attach})$,满足:

$$(RoA[p](x,w)\wedge p(u,w))@\sigma(s_{a\_attach})\wedge(\neg RoA[p](x,w)\wedge RoA[p](x,u)\wedge NextNull[p](u))@\sigma(s_{b\_attach})=1,$$

则 s_{b_attach} 和 s_{a_attach} 之间的语句是将 w 添加到尾端的操作,u 是添加 w 前的尾端.

(3) 自 s_{b_attach} 开始,逆控制流找到 s_{b_detach},使得

$$NextNull[p](u)@\sigma(s_{b\_attach})\wedge\neg NextNull[p](u)@\sigma(s_{b\_detach}))=1.$$

同时,不存在 s_{b_detach} 和 s_{b_attach} 之间的节点 s''使得$\neg NextNull[p](u)@\sigma(s'')=1$,则紧跟 s_{b_detach} 的语句执行了解裂操作,解裂点为 u.

(4) 识别解裂操作和解裂点以后,关于链表中对象及其顺序是否得到保留的判断参照第 2.3 节中的讨论,限于篇幅,这里不再赘述.

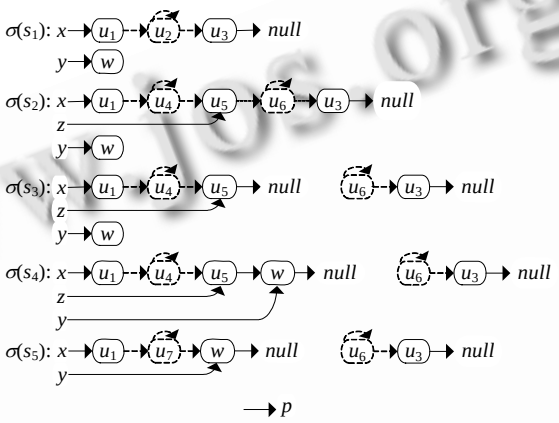

Fig.3 An operation to replace sublist at the end

图 3 一个尾端子链表替换操作

2.5 遍历操作语义特征的描述与识别

遍历操作的特点是:执行遍历操作前后,某变量所引用的对象在链表的一个固定方向上可达.这种不同程序点上的可达性由本文提出的结构间可达谓词表达.遍历操作通常通过两种方式实现:一种是循环语句,另一种是遍历方法.对前者而言,若要识别的是对 O 类域链表的遍历操作,则首先搜索存在对 O 类域对象赋值的循环语句,将循环体展开为连续的两次执行.若原循环体控制流上的某点 s 对应于展开后的两点 s'和 s'',对循环体中的同一变量 x,有 $R[p](s',x,s'',x)=1$,则表明该循环语句执行遍历操作,x 是游标,遍历方向为 p.

对遍历方法而言,方法的返回值或方法所在对象的成员变量都可能是游标.若方法 f 有参数 out,f 的连续两次执行的出口记作 $exit$ 和 $exit'$,若 $R[p](exit,out,exit',out)=1$,则该方法执行了遍历操作,$out$ 是游标.若该方法入口和出口分别为 $entry$ 和 $exit$,r 是该方法所属的对象的成员变量,若 $R[p](entry,r,exit,r)=1$,则亦可判定该方法实现了遍历操作,游标为 r.

2.6 双向链表管理操作语义特征的描述与识别

为分析双向链表,我们需要定义双向链表上的有关谓词.首先引入谓词 $C[p,q](u)$和 $C[p,q](u,v)$[10].$C[p,q](u)$ 定义为$\exists v:p[u,v]\wedge q[v,u]$,表示由 u 出发先后经过一次 p 引用和 q 引用可回到 u,意味着 u 位于 p 和 q 形成的双向

链表内.互引用谓词 $C[p,q](u,v)$定义为 $p(u,v)\wedge q(v,u)\vee q(u,v)\wedge q(v,u)$,表示 u 和 v 是 p 和 q 形成的双向链表上的相邻节点.我们设计宏谓词 $InC[p,q](u)$描述 p 和 q 形成的双向链表中的所有节点,定义为 $C[p,q](u)\vee C[q,p](u)$.

双向链表的可达性定义比单向链表复杂。我们设置谓词 $R[p,q](u,v)$表示 v 在 p 和 q 形成的双向链表上由 u 可达,具体定义为

$$R[p,q](u,v)\Leftrightarrow C[p,q](u,v)\vee(C[p,q](u)\wedge C[q,p](v)\wedge R[p](u,v)\wedge(\forall w:(R[p](u,w)\wedge R[p](w,v)\Rightarrow C[p,q](w)\wedge C[q,p](w))).$$

该定义的重点在于要求从 u 到 v 的路径上的每个节点都处于 p 和 q 形成的双向链表的内部,与单向链表类似.其中,$C[p,q](w)\wedge C[q,p](w)$表示 w 位于 p 和 q 形成的双向链表中,且不是根节点或尾节点.我们设置一元辅助谓词 $R[x,p,q](u)\Leftrightarrow R[p,q](x,u)$区分由不同变量可达的个体,设置宏谓词 $RoA[p,q](u,v)$描述双向链表下的“可达或别名”关系,定义为 $Aliased(u,v)\wedge InC[p,q](u)\wedge InC[p,q](v)\vee R[p,q](u,v)$.

我们将双向链表根对象谓词 $IsRoot[p,q](u)$定义为“$\exists x:(x(u)\wedge C[p,q](u)\wedge\neg\exists y,v:y(v)\wedge R[p,q](v,u))$”.其含义是,在双向链表中存在某变量引用 u,但不存在被变量直接引用的个体 v,使得 v 在 p 方向可到达 u.我们将 p 和 q 形成的双向链表 p 方向的尾节点定义为 $Tail[p,q](u)\Leftrightarrow C[q,p](u)\wedge\neg C[p,q](u)$.双向链表首尾对象的判断与单向链表的区别表现在两方面:首先,单个独立对象可以看作是自身的首端或尾端,而两个以上的对象才能构成双向链表;其次,双向链表在 q 方向的尾节点 u 无须满足 $NextNull[q](u)=1$.

有了以上谓词,在绝大部分情况下,只需将单向链表管理操作识别条件中的谓词换成适应双向链表的谓词即可.但是,对于尾端子链表替换操作需要进行一个细小的调整.将对象挂入单项链表的操作只由 1 条引用赋值语句即可完成;而对于双向链表,该操作至少需要两个引用赋值语句共同完成.因此,需要将第 2.4 节中步骤 2 的“$s_{b\_attach}\in prev(s_{a\_attach})$”替换为“不存在 $s''\in prev^{+}(s_{a\_attach})\wedge s_{b\_attach}\in prev^{+}(s'')$,使得

$$(RoA[p,q](x,w)\wedge C[p,q](u,w)\vee\neg RoA[p,q](x,w)\wedge RoA[p,q](x,u)\wedge Tail[p,q](u))@\sigma(s'')=1”.$$

这意味着 s_{b_attach} 和 s_{a_attach} 共同界定了将 w 挂入链表的语句所在的范围.

双向链表谓词的变迁函数可以手工计算或采用 Sagiv 等人给出的统一计算方法计算[10,12,13],限于篇幅,这里略去其计算过程和结果.

3 实例分析

本节给出了从开源程序 JEdit 中识别备忘录聚集管理操作的过程.分析步骤如下:

(1) 首先由语法分析查找通过成员变量引用自身或祖先类的类,得到类 *Edit*,它包含两个 *Edit* 类引用类型的成员变量 *prev* 和 *next*,具备形成 *Edit* 类域对象链表的可能性.

(2) 通过语法分析寻找所有通过成员变量引用 *Edit* 类的类,发现 *UndoManager* 具有 4 个 *Edit* 类引用类型的成员变量 *undosFirst*,*undosLast*,*redosFirst* 和 *redosLast*(为简化描述,此处忽略了源程序中 *UodoManager* 对 *CompundEdit* 的引用).

(3) 对程序进行形态分析,考察 *prev* 和 *next* 是否形成双向链表,*UndoManager* 是否稳定地引用 *Edit* 链表的根节点.由形态分析可以逐步得到 *UndoManager* 的所有稳定状态(指初始化后,公有方法执行前后的状态)发现,在 *UndoManager* 的所有稳定状态中,*undosFirst* 均引用由 *prev* 和 *next* 形成的双向链表的根对象;其余 3 个 *Edit* 类引用类型的成员变量当不为 *null* 时均引用同一个链表中的节点.图 4(b)中显示了 *UndoManager* 的稳定状态之一,*UndoManager* 在备忘录链表的长度超过一定范围时会放弃最早的若干备忘录对象.$\sigma(b)$中的 u_1 和 u_2 代表被放弃的备忘录对象,它们仍然属于该链表,却不能由任意变量顺 *next* 方向到达.这样,*undosFirst* 仍然是 *next* 方向上的根引用.此例说明了根对象与链表首端对象的区别.

(4) 识别 *UndoManager* 的方法中是否存在尾端添加或尾端子链表替换操作.此处给出按照第 2.3 节和第 2.4 节的步骤识别尾端子链表替换操作的过程.*addEdit* 是 *UndoManager* 的方法,图 5(a)显示了该方法中一条路径上的语句.首先按照第 2.3 节的步骤(1)考察该方法是否实施了对象添加操作.图 4(b)中的$\sigma(b)$是 *addEdit* 入口处的状态之一,其中,输入参数 *edit* 引用一个独立对象 u_9.经形态分析可以得到经过该路径后在 *addEdit* 出口处的状态如$\sigma(e)$所示,说明 u_9 被添加到双向链表中,且位于新链表的末端.

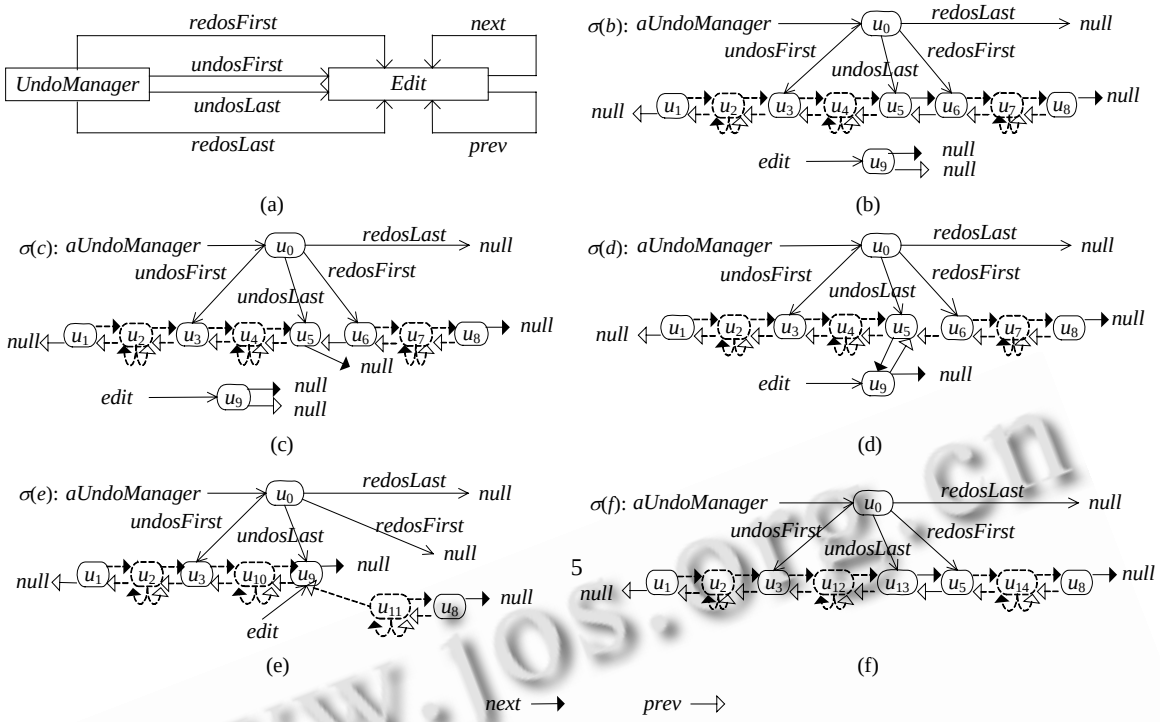

Fig.4 Process to identify aggregation management operations in JEdit by shape analysis

图 4 一个通过形态分析识别 JEdit 中聚集管理操作的过程

```
     private void addEdit(Edit edit) {
10     if (undosFirst==null)
20       …
       else {
25       undosLast.next=null;
30       undosLast.next=edit;
40       edit.prev=undosLast;
50       undosLast=edit;}
60     redosFirst=redosLast=null;
70     …}
```

(a)

```
     public int undo() {
110    if (undosLast==null)
         return −1;
       else {
120      redosFirst=undosLast;
130      undosLast=undosLast.prev;
140      if (undosLast==null)
150        undosFirst=null;   }}
```

(b)

Fig.5 Operations to manage memento aggregation in JEit

图 5 JEdit 中的备忘录聚集管理操作

按照第 2.4 节的步骤(2),从 *addEdit* 的出口逆着控制流对比各点上的形态图发现:在状态 $\sigma(d)$ 中,u_9 属于链表;在 $\sigma(c)$ 中,u_9 不属于链表.这说明尾端添加操作由 $\sigma(c)$ 和 $\sigma(d)$ 之间的语句实现,且 u_5 是可能的解裂点.而 $\sigma(c)$ 和 $\sigma(d)$ 分别是语句 25 和语句 40 执行之后的状态,由此可知,尾端添加操作由语句 30 和语句 40 实现.需要说明的是,语句 25 是在规范化预处理的过程中添加进去的.规范化预处理在所有“*x*:=*y*”这样的语句之前添加语句“*x*:=*null*;”,使得引用赋值语句的语义更为明晰[10].

现在,执行第 2.4 节的步骤(3),继续沿控制流上溯发现,u_5 在 $\sigma(c)$ 中位于 *undosFirst* 所引用的链表的尾端,但在此前的 $\sigma(b)$ 中不位于尾端,表明 $\sigma(b)$ 和 $\sigma(c)$ 之间的语句实施了解裂操作,解裂点是 u_5.因为 $\sigma(b)$ 是方法入口处的状态,由此可知语句 25 实施了解裂操作.

执行第 2.4 节的步骤(4).由于 $\sigma(b)$ 就是方法开始状态,显然,在解裂操作之前,链表中的所有元素均无变动.现在考察解裂点 u_5 之前的对象是否均保留于链表之中.可以发现,在 $\sigma(b)$ 到 $\sigma(d)$ 的各状态中,解裂点之前的对象均

无变化,$\sigma(d)$中的 u_4 和 u_5 在$\sigma(e)$中被融合进 u_{10},仍然属于该链表,即

$$(IsRoot[next,prev](undosFirst)\wedge RoA[next,prev](undosFirst,u_i)\wedge RoA[next,prev](u_i,u_5))@\sigma(b)\Rightarrow$$
$$(IsRoot[next,prev](undosFirst)\wedge RoA[next,prev](undosFirst,u_{10}))@\sigma(e)\wedge InFlow(b,u_i,e,u_{10})=1,$$

其中,i=4,5.

执行第 2.4 节的步骤(5),检验是否在解裂操作后解裂点之后的所有对象均不在链表中.在$\sigma(b)$中,解裂点 u_5 之后的个体为 u_6~u_8;在解裂后的状态$\sigma(c)$和$\sigma(d)$中,个体 u_6~u_8 仍然存在,但不在以 *undosFirst* 为根引用的链表中;在其后的状态$\sigma(e)$中,u_8 不属于该链表;u_6 和 u_7 融合为 u_{11},u_{11} 也不属于该链表.可见,解裂点之前的对象仍然保存在链表中且顺序不变,解裂点之后的对象在解裂操作后均不属于链表.至此可以断言,图 5(a)所示的路径执行了尾部替换操作.*addEdit* 中还包括其他执行路径,对于同一个输入状态,这些执行路径会产生更多的输出状态.但只要有 1 条路径实现了尾端子链表替换操作,我们就可以认定该方法实现了尾端子链表替换操作.

(5) 识别 *UndoManager* 的方法中是否存在遍历操作.图 5(b)显示了 *UndoManager* 的 *undo*()方法中与 *Edit* 引用赋值相关的代码,*undo*()操作没有参数.设该方法入口处方法所属的对象的状态为图 4 中$\sigma(b)$所示的状态,根据$\sigma(b)$中 u_4 所代表的对象数量的不同,在方法出口处可能产生不同的形态结构,图 4 中的$\sigma(f)$为其中之一,有

$$R[prev,next](b,undosLast,f,undosLast)=1,R[prev,next](b,redosFirst,f,redosFirst)=1,$$

即$\sigma(f)$中的 *undosLast* 由$\sigma(b)$中的 *undosLast* 在 *prev* 方向可达,$\sigma(f)$中的 *redosFirst* 由$\sigma(b)$中的 *redosFirst* 在 *prev* 方向可达.因此,该方法是遍历方法,其游标为 *undosLast* 和 *redosFirst*,方向为 *prev*.

(6) 至此可以确定,*UndoManager* 是 *Edit* 链表的管理者.图 4(a)中,*UndoManager* 的 4 个 *Edit* 类引用类型的成员变量可以由普通关联合并为聚集关系.除了 *addEdit* 以外,*UndoManager* 中未发现其他向 *Edit* 链表首部或中间添加对象的操作,未发现修改链表成员状态的操作.除了 *undo* 以外,可以分析出 *redo* 也是遍历操作,其游标移动方向与 *undo* 相反.因此,*UndoManager* 符合备忘录聚集管理者的特征.

通过形态分析,我们还识别了 *Edit* 类体系中的 *CompoundEdit*,*Edit*,*Remove* 和 *Insert* 这 4 个类形成的组合模式以及其中的管理操作.Java 开源程序中的观察者聚集基本上都采用预定义的容器类 *List* 实现,也符合集中管理操作的特征.

4 相关工作与比较

在现有的设计模式识别研究中,对聚集关系的描述和识别基本从语法特征的角度进行.针对常见的实现方式,若类 *A* 的方法中包含 *B* 类参数或返回值,通常二者间的关系被识别为普通关联[6].若类 *A* 包含类 *B* 的实例,Prechelt,Gueheneuc,Zhang 等人将其识别为一对一的聚集关系[3,5,6],Seemann 等人将其识别为组合关系(composite)[14],但我们认为这种数据结构不满足 UML 对组合关系的定义.若类 *A* 包含类 *B* 的指针或引用,Prechelt,Gueheneuc 等人将其识别为普通关联[3,5],Zhang 等人将其识别为聚集[6]. Seemann 等人认为:若类 *A* 同时引用多个类 *B* 的对象,则二者构成一对多的普通关联;若类 *A* 负责创建被其引用的类 *B* 对象,则两者间也可以看作是聚集关系[14].若类 *A* 包含类 *B* 的实例数组或以类 *B* 对象为元素的预定义容器对象,相关研究比较一致地将其识别为一对多的聚集[2−5].这些研究大多直接给出对实现方式的界定标准,很少给出具体理由,基本上未结合设计模式的具体语境进行深入的讨论.我们认为,数组和容器类这两种方式之所以得到研究者们在直觉上的一致认可,就是因为它们均符合集中管理的特征:数组的拥有者显然要承担数组的管理,而 C++,Java 等语言的预定义容器类均对容器中的成员实行集中管理.

Gueheneuc 等人对关联、聚集和组合关系的实现方式进行了比较深入的讨论,但是回避了类 *A* 包含类 *B* 的指针或引用的情况[5].Yeh 等人以操作传递为聚集关系的识别标准[15].对于 GoF 设计模式,该标准过于宽泛.有一类研究以识别对象间的占有关系(owner)为目的[16],这样的工作在理论上可能对识别少数具有独占关系的设计模式有益,但目前尚未有将其与设计模式识别相结合的研究.Kang 等人给出了一种识别变量间一对多关联的方法[7].该方法以聚集管理操作的识别为前提.本文的工作提供了这样的前提.

关于设计模式识别的研究,需要从语法和语义两方面进行.现有研究大多数仅考虑了语法特征或以语法特

征为主[14,17].近来,一些研究者提出了根据模式行为的顺序和调用关系进行识别的思想[18],但其前提是能够识别单个模式行为.单个的模式行为基本上不具备明显的语法特征,需要通过语义进行识别.有些研究通过参数类型声明、对成员变量的引用、对其他方法的调用等语法信息识别模式行为[19],这些信息都比较表面化,不能说明该方法履行了设计模式所要求的语义.还有一些研究考虑到方法或类的命名包含语义信息,采用命名匹配的方式识别模式行为或模式角色[19,20].然而,程序中命名的规律性是相对的,采用命名匹配方法并不可靠.这种方法没有从程序的内在性质上进行识别,缺乏理论意义.Shi 等人讨论了控制流分析对于设计模式识别的必要性,但讨论非常简单,没有引入任何具体的数据流分析方法[21].

对备忘录模式识别的研究很少.Shi 等人认为,备忘录模式难以识别,因为很难给出其可识别特征[21].相关研究仅涉及该模式中 3 种角色之间的可见、使用和创建关系,丝毫未涉及备忘录的时间特性[17,18,22].对组合模式的研究较多,大多是在假设聚集关系已经得到识别的前提下进行的[17,23].Prechelt,Niere 和 Huang 等人考虑到管理操作识别的必要性,但未就识别方法进行具体讨论[3,4,19].Zhang,Huang 等人识别出的组合模式均由数组或预定义容器类实现[2,19].现有的对观察者模式的研究基本上都意识到聚集关系以及模式行为识别的重要性,但由于缺乏语义描述和识别能力,均未给出有效的方法[17,24,25].

形态分析传统上主要用于分析链表结构的性质,不涉及链表管理操作的识别.现有研究对无环单向链表给出了比较详细的分析,对无环双向链表的整体结构的变化和成员间的可达性未有专门的分析[26].在现有研究中,所有谓词均用来表达同一形态结构中个体间的关系,未提供描述不同形态结构间关系的方法[10−13].

5 结 语

现有的聚集关系识别研究没有与具体的语境相结合,对以指针或引用方式实现的聚集未提供有效的处理方法.本文以设计模式识别为语境,给出了备忘录、组合和观察者等模式中一对多聚集关系的共同特征:聚集中的“一”方对“多”方实施集中管理.针对常用的聚集实现方式即一维无环链表,以 Sagiv 等人的形态分析方法为基础,扩展了结构间谓词以表达控制流上不同形态结构间的可追踪性和属于不同形态结构的对象间的可达性,设计了用于分析双向链表结构的若干谓词,提供了根据形态结构的变化识别备忘录链表管理操作的方法.需要指出的是,尽管具体的识别细节可能有所差异,本文的方法也适用于组合和观察者等模式中的集中管理式聚集的识别.我们的分析还揭示了现有研究非常一致地将数组和容器类界定为一对多聚集关系的原因:它们均具备集中管理的特征.现有设计模式识别研究主要关注语法特征,对语义特征的描述和识别还几乎处于空白状态.本文通过引入形态分析技术,从静态语义分析这一新的角度提高了描述和识别设计模式的能力,使得识别结果更为细致、准确.

我们通过对开源代码的分析验证了形态分析在聚集管理操作及相关模式行为识别上的有效性,今后还需要进行更大规模的实验,考察本方法的适用面.从理论上来讲,由链表方式实现的集中管理式聚集还可以有一维无环链表以外的其他实现方式,我们还需要研究针对这些实现方式的特征描述和识别方法.聚集的特征因具体语境而变,在其他设计模式中也存在聚集关系.组合、备忘录、观察者模式中聚集成员的数量是动态变化的、不确定的.在其他设计模式中还存在一些成员数量相对稳定的聚集关系,其实现方式可能更为多样,如何识别这样的聚集关系也是我们今后研究的内容.设计模式中还存在着丰富的语义特征有待描述和识别,目前这方面的工作还非常缺乏,是一个值得大力探索的领域.

References:

[1] Gamma E, Richard H, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison Wesley Longman, 1995.

[2] Zhang Y. Automatic design pattern recovery [MS. Thesis]. Montreal: Concordia University, 2003.

[3] Prechelt L, Krämer C. Functionality versus practicality: Employing existing tools for recovering structural design patterns. Journal of Universal Computer Science, 1998,4(12):866−882.

[4] Niere J, Schafer W, Wadsack JP, Wendehals L, Welsh J. Towards pattern-based design recovery. In: Proc. of the Int'l Conf. on Software Engineering. New York: ACM Press, 2002. 338−348.

[5] Gueheneuc YG, Antoniol G. A multi-layered approach for design pattern identification. IEEE Trans. on Software Engineering, 2008,34(5):667−684. [doi: 10.1109/TSE.2008.48]

[6] Zhang ZX, Li QH, Ben KR. A new method for design pattern mining. In: Proc. of the 3rd Int'l Conf. on Machine Learning and Cybemetics. Washington: IEEE Computer Society Press, 2004. 1755−1759.

[7] Kang Y, Park C, Wu C. Reverse-Engineering 1-*n* associations from Java bytecode using alias analysis. Information and Software Technology, 2007,49(2):81−98. [doi: 10.1016/j.infsof.2006.02.004]

[8] Xu BW. On object oriented programming in Ada95. Journal of Computer Research and Development, 1997,34(1):58−65(in Chinese with English abstract).

[9] Rinetzky N, Poetzsch-Heffter A, Ramalingam G, Sagiv M, Yahav E. Modular shape analysis for dynamically encapsulated programs. In: Proc. of the 16th European Symp. on Programming. LNCS 4421, Heidelberg: Springer-Verlag, 2007. 220−236.

[10] Sagiv M, Reps T, Wilhelm R. Parametric shape analysis via 3-valued logic. ACM Trans. on Programming Languages and Systems, 2002,24(3):217−298. [doi: 10.1145/514188.514190]

[11] Manevich R, Lahiri SK, Sagiv M. Lightweight analysis of acyclic unshared lists. Technical Report, TR-2005-12-1297820, Tel Aviv: Tel-Aviv University, 2005.

[12] Reps TW, Sagiv S, Loginov A. Finite differencing of logical formulas for static analysis. In: Proc. of the 12th European Symp. on Programming. LNCS 2618, Heidelberg: Springer-Verlag, 2003. 380−398.

[13] Jeannet B, Loginov A, Reps TW, Sagiv M. A relational approach to interprocedural shape analysis. In: Proc. of the 11th Static Analysis Symp. LNCS 3148, Heidelberg: Springer-Verlag, 2004. 246−264.

[14] Seemann J, von Gudenberg JW. Pattern-Based design recovery of Java software. ACM SIGSOFT Software Engineering Notes, 1998,23(6):10−16. [doi: 10.1145/291252.288207]

[15] Yeh D, Sun PC, Chu W, Lin CL, Yang H. An empirical study of a reverse engineering method for the aggregation relationship based on operation propagation. Empirical Software Engineering, 2007,12(6):575−592. [doi: 10.1007/s10664-007-9043-7]

[16] Milanova A. Precise identification of composition relationships for UML class diagrams. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2005. 76−85.

[17] Philippow I, Streitferdt D, Riebisch M, Naumann S. An approach for reverse engineering of design patterns. Software and Systems Modeling, 2004,4(1):55−70.

[18] Ng JKY, Guéhéneuc YG. Identification of behavioral and creational design patterns through dynamic analysis. In: Zaidman A, Hamou-Lhadj A, Greevy O, eds. Proc. of the 3rd Int'l Workshop on Program Comprehension through Dynamic Analysis. Technical Report, TUD-SERG-2007-022, Delft: Delft University of Technology, 2007. 34−42.

[19] Huang H, Zhang S, Cao J, Duan Y. A practical pattern recovery approach based on both structural and behavioral analysis. The Journal of Systems and Software, 2005,75(1-2):69−87. [doi: 10.1016/j.jss.2003.11.018]

[20] Dong J, Lad DS, Zhao Y. DP-Miner: Design pattern discovery using matrix. In: Proc. of the 14th Annual IEEE Int'l Conf. on Engineering of Computer Based Systems. Washington: IEEE Computer Society Press, 2007. 371−380.

[21] Shi N, Olsson RA. Reverse engineering of design patterns from Java source code. In: Proc. of the 21st IEEE Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society Press, 2006. 123−134.

[22] Gruijs D. A framework of concepts for representing object-oriented design & design patterns [MS. Thesis]. Utrecht: Utrecht University, 1998.

[23] Costagliola G, De Lucia A, Deufemia V, Gravino C, Risi M. Design pattern recovery by visual language parsing. In: Proc. of the 9th European Conf. on Software Maintenance and Reengineering. Washington: IEEE Computer Society Press, 2005. 102−111.

[24] Heuzeroth D, Mandel S, Löwe W. Generating design pattern detectors from pattern specifications. In: Proc. of the 18th IEEE Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society Press, 2003. 245−248.

[25] France RB, Kim DK, Ghosh S, Song E. A UML-based pattern specification technique. IEEE Trans. on Software Engineering, 2004, 30(3):193−206. [doi: 10.1109/TSE.2004.1271174]

[26] Cherem S, Rugina R. Maintaining doubly-linked list invariants in shape analysis with local reasoning. In: Proc. of the 8th Int'l Conf. on Verification, Model Checking and Abstract Interpretation. LNCS 4349, Heidelberg: Springer-Verlag, 2007. 234–250.

附中文参考文献:

[8] 徐宝文.Ada95 与面向对象的程序设计.计算机研究与发展,1997,34(1):58–65.

周晓宇(1972－),男,江苏淮安人,博士生,副教授,主要研究领域为软件分析与理解,逆向工程与再工程.

陈林(1979－),男,博士,讲师,主要研究领域为软件分析与维护.

钱巨(1981－),男,博士,讲师,主要研究领域为软件分析.

徐宝文(1961－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为程序设计语言,软件工程,并行与网络软件.