并发面向对象中的继承反常现象*

王生原[1], 杨良怀[2], 袁崇义[3], 杨 萍[4]

[1](清华大学 计算机科学与技术系,北京 100084);
[2](新加坡国立大学 计算学院 计算机科学系,新加坡);
[3](北京大学 计算机科学技术系,北京 100871);
[4](兰州大学 信息科学与工程学院,甘肃 兰州 730000)
E-mail: wwssyy@tsinghua.edu.cn; yanglh@comp.nus.edu.sg
http://www.cs.tsinghua.edu.cn

摘要: 如果不考虑继承性,并发性与对象技术的结合是很自然的.继承反常(又称继承异常)现象是继承性和并发性不相容的主要原因之一.现阶段人们对继承反常现象的认识有许多模糊之处,出发点不尽相同,形式化的工作也很少.对不同的 subtyping 关系考虑其特有的渐增式继承方法有利于把握继承反常现象的实质,也丰富了"在并发面向对象语言中应将 inheritance 层次和 subtyping 层次区别对待"这一认识的内涵.在阐述基本观点之后,采用范畴论的术语对相关的概念和定义做了形式化工作.一些观点和结论适用于区分和解释相关工作的出发点和贡献,并对并发面向对象技术中继承性的建模问题有所启示.

关 键 词: 并发性;面向对象;继承反常;渐增式继承;范畴论
中图法分类号: TP301 文献标识码: A

Inheritance(继承)是类(class)之间的一种层次关系.在一般的面向对象语言中,类层次自动对应了一种类型(type)层次[1],这样,superclass 和 subclass 之间的关系自然是一种 supertype 和 subtype 之间的关系.这种对应关系难免使人们在 inheritance 和 subtyping 之间产生模糊的认识.对于大多数顺序的面向对象语言来说,inheritance 和 subtyping 不加以区分几乎不会造成什么影响.但随着把并发性(concurrency)引入面向对象语言中[2~4],人们发现 inheritance 和 subtyping 之间并不容易协调,会出现所谓的继承反常(inheritance anomaly)现象[5].

在并发面向对象语言中,对象是很自然的并发单元.对象之间具有并发性,对象内部也具有并发性.无论对象之间还是对象内部的并发性,都和同步代码(synchronization code)有关.同步代码的主要作用是对对象内部及其与外界的接口的并发活动进行一种约束,即同步约束(synchronization constrains).在 inheritance 层次中,方法(及属性)的增改很可能会破坏这种约束.换句话说,subclass 要兼容其 superclass 的这种同步约束,很可能必须对继承下来的 superclass 代码进行实质性修改,而难以重用它们.这种现象在文献[5]中首次被称为继承反常.在此之前也有许多这方面问题的讨论[6,7],目前仍然很受关注[8,9].

对于继承反常现象,目前的解决方案主要有两类:(1) 设计更强的同步机制[5];(2) 去掉同步代码[8,10].但是,这些方法只能有限地减小继承反常的危害,并不能从根本上解决问题.面对所遇到的困难,有人在考虑是不是存在概念上的误区[11],目前这仍然是一个开放的话题.实际上,除了同步约束,继承反常现象也可能是由其他情况

* 收稿日期: 2000-10-08; 修改日期: 2001-12-05
基金项目: 国家自然科学基金资助项目(69973003);国家重点基础研究发展规划973资助项目(G1999032706);甘肃省自然科学基金资助项目(ZS991-A25-014-G)

作者简介: 王生原(1964－),男,山西应县人,博士,副教授,主要研究领域为分布对象计算,Petri 网应用,嵌入式软件环境;杨良怀(1967－),男,浙江新昌人,博士,讲师,主要研究领域为数据库系统实现技术,数据仓库,半结构化数据(XML),Web 数据集成,数据挖掘;袁崇义(1941－),男,山东邹平人,教授,博士生导师,主要研究领域为并行计算与 Petri 网,Petri 网理论及应用;杨萍(1964－),女,陕西西安人,副教授,主要研究领域为人工智能,基于 Agent 的系统.

引起的[12],这说明继承反常应属于 inheritance 本身的特征.

不同背景和意义下的“继承反常”足以使读者产生困惑.比如,有些工作声称可以避开继承反常现象[8,10],而另一些工作则认为继承反常是不可避免的[9].究其原因主要是它们的出发点不尽相同,没有统一规范的认识,并且这方面的形式化工作很少,也不够全面.本文对于继承反常现象给出了一种更加灵活、普适的定义,并采用范畴论的术语对其进行了形式化的工作.

本文第 1 节通过经典实例使读者初步认识继承反常现象.第 2 节分析继承反常现象的实质,并阐述本文理解这种现象的基本观点.第 3 节用范畴论的术语形式地定义继承反常现象.第 4 节利用本文的定义对文献中出现的有关继承反常现象的典型情形进行解释,并结合本文的观点,对并发面向对象技术中继承性的建模提出一些有益的建议.

1 认识继承反常现象

考虑图 1,类 Buffer 实现了一个有界缓冲区类型,可以并发地接受消息 put 和 get.同步机制采用了“method guards”方式,即为每一个 method 附加一个 guard 谓词.如果对类 Buffer 增加新方法 gget(这里 gget 的行为与 get 相同,但不可以紧跟在 put 之后执行)来构造子类 GBuffer,就不得不修改同步代码,因而方法 put 和 get 的代码须重新定义.同样,构造 Buffer 的另一个子类 LockableBuffer 也会引发同样的问题.这种“为获得有效继承而必须对父类代码进行实质性修改的现象”就是所谓的继承反常(inheritance anomaly)[5].

```
class   Buffer {
      Int   in=out=0,   buf[SIZE];
      method   put()   when   (in < out + SIZE) {in++;   //rest code of put()};
      method   get()   when   (in >= out + 1) {out++;   //rest code of get()};
}
class   Buffer2: Buffer {
      method   get2()   when   (in >= out + 2) { out + = 2;   //rest code of get2()};
}
class   GBuffer: Buffer {
      bool   after_put = false;
      method   gget()   when   (!after_put && (in >= out + 1)) {out++;   after_put = false;   //rest code of gget()};
      method   put()   when   (in < out + SIZE) {in++;   after_put = true;     //rest code of put()};
      method   get()   when   (in >= out + 1) {out++;   after_put = false;   //rest code of get()};
}
class   LockableBuffer: Buffer {
      bool   locked = false;
      method   lock()   when   (!locked) {locked = true};
      method   unlock()   when   (locked) {locked = false};
      method   put()   when   (!locked && (in < out + SIZE)) {in++;   //rest code of put()};
      method   get()   when   (!locked && (in >= out + 1)) {out++;   //rest code of get()};
}
```

Fig.1 The bounded buffer class Buffer and its subclass Buffer2, Gbuffer and LockableBuffer

图 1 有界缓冲区类 Buffer 及其子类 Buffer2,GBuffer 和 LockableBuffer

继承反常与实现同步代码的机制有关.图 1 中采用“method guards”机制,Buffer2 可以不加修改地继承 Buffer 的方法 put 和 get(新方法 get*2* 每次取出两个元素).而 ACT++(一种基于 Actor 的语言)采用 Behavior Abstractions 机制(参见文献[7]),Buffer2 在继承 Buffer 时不可避免地要重新定义方法 put 和 get.

2 理解继承反常现象

2.1 区别Inheritance和Subtyping

综合各种观点,要理解和解释继承反常现象,首先必须正确区别 inheritance 层次和 subtyping 层次,并深刻领会二者的联系.对于这个问题,文献[1,13]的观点是比较权威的.在此基础上,我们的理解是:Inheritance 是在代码层次上作修改,而 subtyping 是在语义层次上作修改.前者是代码共享的一种重要途径,但不能保证 subclass 能够继承 superclass 的行为;后者要求 subtype 保持 supertype 的某种外部可观察行为(或语义行为),在规范一级共享,

同代码没有关系.Inheritance 层次关系可以理解为“is_similar_to”(或“like”)的关系,而将“is_a”关系更适合用在理解 subtyping 层次关系上.

2.2 渐增式继承

Subtyping 要求 subtype 保持 supertype 的某种行为(可看作是一种不变量,比如同步约束).Subclass 在增加新的属性或方法时,为了避免破坏这种不变量,难免要对继承的代码进行扩展或修改.这种扩展或修改很可能是重大的或实质性的,结果使得代码共享失去意义.这便是继承反常的直观含义.

那么,什么样的扩展或修改算是重大的或实质性的呢?这是比较含糊的说法,以致于多种关于继承反常的说法使读者产生误解.文献[9]在 inheritance 必须是渐增式的假设下,给继承反常下了一个形式化的定义.按此定义,先前人们声称对继承反常问题的解决都是部分的,即没有任何一种解决方案是彻底的.这是一个极端的定义,它所规定的渐增式继承不允许对代码作任何形式的修改(重载).为了对人们在解决这个问题上的努力有所区别,同时兼顾到不同的语言特征,有必要对“渐增式继承”采取较为灵活的解释.比如,对基于 Petri 网的并发面向对象语言[11,14],将子网(subnets)关系看作渐增式继承是恰当的,没有必要苛求子类网中新增的部分与从父类网继承的部分不相交.

然而,这种子网关系的继承只强调了网结构(代码)的重用,而未顾及行为的重用.对于并发面向对象的程序设计,代码的重用固然是 inheritance 的一个重要目的,但行为的重用更突显其重要地位,也成为问题的焦点.行为的重用在并发面向对象语言中是由 subtyping 关系体现的.从前面的讨论可知,代码重用和行为重用之间的不兼容是产生继承反常的根源.因此,在并发面向对象语言中显式地定义 subtyping 层次成为一种趋势,如 CO-OPN/2[15].

至于要支持什么样的 subtyping 层次关系,应根据应用的目标来确定,这方面人们已有许多工作[16].作为一个例子,文献[17]中基于封装(encapsulation)和抽象(abstraction)的思想利用分支双模拟等价(branching-bisimulation equivalance)关系定义了 Petri 网间的两个 subtyping 关系(原文称为动态行为继承),分别记为$\leq_{pt}$和$\leq_{pj}$,表示两种不同的行为继承(前者具有 blocking 语义,后者具有 hiding 语义).由$\leq_{pt}$和$\leq_{pj}$的析取又得到另一个 subtyping 关系$\leq_{lc}$(又是一种不同的行为继承方法).

面对不同的 subtyping 关系,继承反常的多少是不相同的.文献[9]的定义中也注意到了这一点.进一步,我们还认为,对不同的 subtyping 关系,“渐增式继承”的含义也应有所区分,这既有利于更客观地比较和评价人们提出的各种解决继承反常的方法,也利于在实际开发中更方便地实现相应的 subtyping 关系.例如,在文献[17]中,针对 subtyping 关系$\leq_{pt}$,$\leq_{pj}$和$\leq_{lc}$分别给出了相应的继承保持变换规则,它们都可被包括在相应 subtyping 关系的“渐增式继承”集合之中.

2.3 理解继承反常

本文基于以下观点来理解继承反常现象(可结合第 3 节的形式定义):

(1) 不同的语言机制有不同的继承反常现象,因此继承反常的定义是针对语言的(可参考第 2 节).

(2) 语言机制确定之后,其 class 和 type 系统就确定了,有什么样的 inheritance 规则和什么样的 subtyping 关系也就确定了.每一个 class 都有一个惟一的 type 与之对应(可以是多对一),但反之则不一定.

(3) 把一个 inheritance 规则理解为所有 class 集合上的一种先序(preorder)关系,一组 inheritance 规则定义了一种 inheritance 层次关系,也是先序关系(参照定义 4 中范畴 C_R).某个语言中由全部 inheritance 规则定义的 inheritance 层次关系称为该语言的完整 inheritance 层次关系(参照定义 4 中范畴 C_L).同样,把一个 subtyping 关系理解为所有 type 集合上的一种先序关系(参照定义 5 中范畴 T_p).

(4) 一种 inheritance 层次关系实现某种 subtyping 关系,如果具有这种 inheritance 层次关系的任何两个 class 对应的 type 之间都具有这种 subtyping 关系.本文中,继承层次关系 P 实现 Subtyping 关系 p 是指存在从关系 P 到关系 p 的保序映射,不必要是双射(参照定义 6 中的实现函子 **F**).

(5) 设 P 和 Q 是可以实现某种 subtyping 关系 p 的两个 inheritance 层次关系,且 P 包含 Q.A 是任何一个 class.若在关系 P 下,A 的任何 subclass B,都存在 class C(对于 Subtyping 关系 p,class B 和 C“具有兼容的 type”),使得

在关系 Q 下 C 是 A 的 subclass,则称在 subtyping 关系 p 下 inheritance 层次关系 P"可继承归约"到 inheritance 层次关系 Q.其实际含义为:对于由 p 所定义的行为继承关系(即 Subtyping 关系)而言,继承层次关系 Q 的实现能力足以替代继承层次关系 P 的实现能力(参见定义 7).

对于某种 Subtyping 关系 p,两个 class B 和 C"具有兼容的 type"意味着对于 p,$type(B)$是 $type(C)$的子类型,同时,$type(C)$也是 $type(B)$的子类型(注意,p 是先序,而不是偏序).

(6) 一种语言机制中可以存在多种 subtyping 关系.每一种 subtyping 关系都规定一种 inheritance 层次关系为其相应的"渐增式继承"关系.这个"渐增式继承"关系一定是可以实现该 subtyping 关系的."渐增式继承"是一个相对的概念,意指"理想"的继承方式.

(7) 对某一种 subtyping 关系 p,若所有可以实现 p 的 inheritance 层次关系都可继承规约到相应于 p 的"渐增式继承"关系,则相对于 subtyping 关系 p,该语言没有继承反常(anomaly-free)(参见定义 8 中的(2)).

(8) 若对于某一语言的所有 subtyping 关系,该语言都没有继承反常,则这种语言不存在继承反常,否则就存在继承反常(参见定义 9).

(9) 对某一种 subtyping 关系,"渐增式继承"关系的限制条件越放宽(即"渐增式继承"关系所包含 inheritance 规则越多),继承反常越少,因为一个 inheritance 层次关系可继承规约到"渐增式继承"关系的可能性增大了(参见命题 3);对某一种 subtyping 关系,若加强此关系的条件,而"渐增式继承"关系不变,则继承反常会减少,因为可实现这种 subtyping 关系的 inheritance 层次关系变少了(参见命题 4).

3 定义继承反常现象

本节是对上述观点的形式化描述,可与第 2.3 节对照阅读.范畴论[18,19]的观点层次较高,易于抽象出问题的本质.

定义 **1**. 一个范畴 C 包含:(1) 对象(objects)集 $ob\ C$;(2) 对每个对象有序对(A,B),相应有一个射(morphism)的集合 $hom_C(A,B)$(若可从上下文得知 C,可简记为 $hom(A,B)$),其中的射具有 domain A 和 codomain B;(3) 对任何对象三元组(A,B,C),有一个从 $hom(A,B)\times hom(B,C)$到 $hom(A,C)$的映射:$(f,g)\rightarrow g\,f$.C 中的对象和射满足以下条件:(C1) 若 $(A,B)\neq(C,D)$,则 $hom(A,B)$和 $hom(C,D)$不相交;(C2) 若 $f\in hom(A,B),g\in hom(B,C),h\in hom(C,D)$,则 $(hg)f=h(g\,f)$;(C3) 对应每个对象 A 都有一个恒等射 $\mathbf{1}_A\in hom(A,A)$,使得对任何 $f\in hom(A,B)$,有 $f\,\mathbf{1}_A=f$,以及对任何 $g\in hom(B,A)$,有 $\mathbf{1}_A\,g=g$.

定义 **2**. 范畴 D 是范畴 C 的一个子范畴,当且仅当(1) $ob\ D\subseteq ob\ C$;(2) 对所有 $A,B\in ob\ D$,有 $hom_D(A,B)\subseteq hom_C(A,B)$;(3) 对 $A\in ob\ D$,和 $\mathbf{1}_A\in hom_C(A,A)$有 $\mathbf{1}_A\in hom_D(A,A)$;(4) 对 $f\in hom_D(A,B),g\in hom_D(B,C)$及 $g\,f\in hom_C(A,C)$,有 $g\,f\in hom_D(A,C)$.若 D 是范畴 C 的子范畴,同时 $ob\ D=ob\ C$,则 D 称为范畴 C 的宽子范畴(wide subcategory).

定义 **3**. 设 C,D 为范畴,从 C 到 D 的一个函子 $\mathbf{F}$ 包含:(1) 一个从 $ob\ C$ 到 $ob\ D$ 的映射 $A\rightarrow\mathbf{F}A$;(2) 对任何 $A,B\in ob\ C$,从 $hom_C(A,B)$到 $hom_D(A,B)$有一个映射 $f\rightarrow\mathbf{F}(f)$.$\mathbf{F}$ 应满足以下条件:(C1) 如果 $g\,f$ 定义在 C 中,则 $\mathbf{F}(g\,f)=\mathbf{F}(g)\ \mathbf{F}(f)$;(C2) $\mathbf{F}(\mathbf{1}_A)=\mathbf{1}_{\mathbf{F}A}$.

设(并发)面向对象语言 L 中,所有 $class$ 的集合为 $Class_L$,所有 type 的集合为 $Type_L$.我们约定:对任何 $A\in Class_L$,都存在 $type(A)\in Type_L$.

用继承规则集 $RL=\{r_1,r_2,\ldots,r_n\}$表示 L 的继承机制,其中每一 $r_i\subseteq Class_L\times Class_L$ 为一先序(preorder)关系,即满足自反性和传递性.L 的所有 subtyping 关系表示为集合 $P_L=\{p_1,p_2,\ldots,p_m\}$,其中每一 $p_i\subseteq Type_L\times Type_L$,也是先序关系.对任何 $p\in PL$,相应于 p 的渐增式继承用继承规则集 $R_p\subseteq R_L$ 来表示.

定义 **4**. 设 $R\subseteq R_L$,范畴 C_R 定义为:(1) $ob\ C_R=\mathbf{Class}_L$;(2) 对所有 $A,B\in ob\ C_R$,$hom(A,B)$是由下列算法确定的最小集:(i) 若存在 $r\in R$,使 $(A,B)\in r$,则 $r\in hom(A,B)$;(ii) 若对 $C\in ob\ C_R$ 存在 $r_1\in hom(A,C),r_2\in hom(C,B)$,则 $r_2,r_1\in hom(A,B)$;(iii) 反复使用(i),(ii).易验证 C_R 符合定义 1.称 C_R 为一个基于 L 的 Class 范畴(在不致混淆的情形下,简称为 Class 范畴).若 $R=R_L$,则称 C_R 为基于 L 的完全 Class 范畴(对应语言 L 的完整 inheritance 层次关系),记为 C_L.

命题 **1**. 设 $R\subseteq R_L,R'\subseteq R,R$ 定义的 Class 范畴为 C_R,R'定义的 Class 范畴为 $C_{R'}$,则 $C_{R'}$是 C_R 的子范畴,并称之

为 C_R 的子 Class 范畴.

证明:由定义 4,对任何 $A,B\in ob\ C_R=ob\ C_{R'}$,$hom_{C_{R'}}(A,B)\subseteq hom_{C_R}(A,B)$. □

推论 **1**. 任何 $R\subseteq R_L$ 定义的 Class 范畴 C_R 都是 C_L 的子 Class 范畴.

定义 **5**. 设 $p\in P_L$,定义范畴 T_p 为:(1) $ob\ T_p=Type_L$;(2) 对任何 $A,B\in ob\ T_p$,$(A,B)\in hom_{T_p}(A,B)$当且仅当$(A,B)\in p$. 称 T_p 为一个 Type 范畴,它实际上也是一个先序集范畴.

定义 **6**. 设 C_R 为一个 Class 范畴,T_p 为一个 Type 范畴,**F** 为从 $ob\ C_R$ 到 $ob\ T_p$ 的映射,满足:对任何 $A\in ob\ C_R$,$\mathbf{F}A=type(A)$.若 F 是从 C_R 到 T_p 的函子,则称 C_R 实现 T_p,**F** 为实现函子.

命题 **2**. 设 C_R 是 $C_{R'}$的子 Class 范畴,$C_{R'}$实现 T_p,则 C_R 也实现 T_p.

证明:C_R 实现 T_p 可使用与 $C_{R'}$实现 T_p 相同的实现函子. □

定义 **7**. 设 C_R 是 $C_{R'}$的子 Class 范畴,T_p 为一个 Type 范畴,$C_{R'}$实现 T_p(由命题 2,C_R 也实现 T_p),F 为实现函子. 称对于 Type 范畴 T_p,$C_{R'}$可继承归约到 C_R,记为 $C_{R'}\Rightarrow_p C_R$,当且仅当对任何 $A,B'\in ob\ C_R=ob\ C_{R'}$,若存在 $f'\in hom_{C_{R'}}(A,B')$,则存在 $f\in hom_{C_R}(A,B)$,使得$(FB,FB')\in hom_{T_p}(FB,FB')\wedge(FB',FB)\in hom_{T_p}(FB',FB)$(参见第 2.3 节的(5)).

注:$(FB,FB')\in hom_{T_p}(FB,FB')$意味着 $type(B')$是 $type(B)$(在 p 下)的子类型,而$(FB',FB)\in hom_{T_p}(FB',FB)$意味着 $type(B)$是 $type(B')$(在 p 下)的子类型.$(FB,FB')\in hom_{T_p}(FB,FB')\wedge(FB',FB)\in hom_{T_p}(FB',FB)$说明对于 Subtyping 关系 p,B 和 B'"具有兼容的 type".

定义 **8**. (1) 设 $R\subseteq R_L$ 且 C_R 实现 T_p.若对于任何 $C_{R'}$:$C_{R'}$实现 T_p,都满足 $C_{R'}\Rightarrow_p C_R$,则称 L 无(R,p)-继承反常.(2) 若 L 无(R_p,p)-继承反常,则简称 L 无 p-继承反常.

命题 **3**. 设 $R\subseteq R'\subseteq R_L$,对于由 $p\in P_L$ 确定的 subtyping 关系,$C_{R'}$实现 T_p.若 L 无(R,p)-继承反常,则 L 也无(R',p)-继承反常.

证明:对任何 $R''\supseteq R'$,且 $C_{R''}$实现 T_p($C_{R'}$,C_R 也实现 T_p),易从 $C_{R''}\Rightarrow_p C_R$ 证得 $C_{R''}\Rightarrow_p C_{R'}$. □

命题 **4**. 设 $p,p'\in P_L$:$p\subseteq p'$,以及 $R\subseteq R_L$.若 L 无(R,p')-继承反常,则 L 也无(R,p)-继承反常.

证明:对任何 R',且 $C_{R'}$实现 T_p($C_{R'}$也必然实现 $T_{p'}$),易从 $C_{R'}\Rightarrow_{p'} C_R$ 证得 $C_{R'}\Rightarrow_p C_R$. □

定义 **9**. 语言 L 无继承反常,当且仅当对任何 $p\in P_L$ 都有:L 无 p-继承反常.若不满足该条件,则语言 L 存在继承反常.

4 解 释

本节通过几个典型的例子配合本文定义的理解,并对并发面向对象中继承性的建模提出一些建议.

4.1 解释继承反常现象

例 1:在谈到顺序面向对象语言时,一般不涉及继承反常现象.这是因为,这类语言 L 都默认一个特殊的 subtyping 关系 p(由 R_L 定义的完整 inheritance 层次关系被默认为相应于 p 的渐增式继承关系),它对应的 Type 范畴记为 T_L,满足:C_L 实现 T_L,实现函子为 **F**:$\forall A(A\in ob\ C_L\rightarrow \mathbf{F}A=type(A))$.由定义 7 和定义 8,$L$ 无 p-继承反常.因为顺序面向对象语言不提供定义 subtyping 关系的机制,所以这种默认的 subtyping 关系是语言中惟一的 subtyping 关系,由定义 9,这些语言 L 无继承反常.

例 2:假设渐增式继承是最严格的一种,即子类中对于被继承的父类代码不可作任何形式的修改.在图 1 中,由类 Buffer 构造子类 GBuffer 和子类 LockableBuffer 时会产生 History-Sensive 式的继承反常现象[5].按照本文的观点,这种继承反常所针对的 subtyping 关系可认为是关系 $P1$(在下面定义).但如果面对的 subtyping 关系是 $P2$(在下面定义),而不是 $P1$,则上述继承反常就被"绕过",因为从类 Buffer 到子类 Gbuffer 或子类 LockableBuffer 的 Class 层次不能实现 subtyping 关系 $P2$.$P1$ 和 $P2$ 分别具有 blocking 和 hiding 的语义(即 encapsulation 和 abstraction 的语义[17]).

$P1$:Subtype 实例接受消息串的能力不亚于 supertype 实例,即后者可接受什么样的消息串,前者也可接受;同时,一个前者可接受的消息串,除非包含后者未定制的消息,否则后者也可接受该消息串.

$P2$:(1) 同 $P1$ 中的(1);(2) 对任何后者可接受的消息串 α,及前者可接受的消息串 β(α与屏蔽掉前者未定制

的消息后的β相等),设前者在接受消息串β后下一个可接受的消息的集合为 R_β,而后者在接受消息串α后下一个可接受的消息的集合为 R_α,则有在 R_β中将前者未定制的消息去掉后与 R_α相等.

例 3:若将图 1 中每个 method 的 guard 部分与其功能部分的代码分离开,则 LockableBuffer 中 get 和 put 的功能部分可直接从 Buffer 继承下来,不必作任何修改.一般来说,将同步代码分离可减少继承反常.进一步,根据本文的定义,若将“可对分离出来的同步代码进行修改”作为一条继承规则添入“渐增式继承”规则集中,则某些继承反常可以“消失”.比如对图 1,若在“渐增式继承”规则集中加入“可对 guard 部分进行修改”,则 LockableBuffer 对 Buffer 的继承就没有反常现象.

4.2 继承性的建模

本文的观点对并发面向对象语言(包括规范语言)的设计及继承性建模的问题有一定的启示.

首先,在“并发面向对象语言中应将 inheritance 层次和 subtyping 层次区别对待”的共识下,强调了继承反常这一概念也应有针对性,对不同的 subtyping 关系应考虑其特有的渐增式继承方法.Subtyping 关系愈强,其行为规范的能力就愈强,但对渐增式继承方法的限制愈强,即渐增式继承的规则愈少,可应用性也愈小.从继承反常的角度来看,对于较强的 subtyping 关系,继承反常较少(命题 4),但较少的渐增式继承规则却会带来较多的继承反常(命题 3).因此,subtyping 关系的设计应根据实际情形权衡利弊.

其次,在并发面向对象的软件开发中,应针对每一种 subtyping 关系分别研制其“渐增式继承”的设计方法及工具,增强它们的可应用性,减小开发人员的负担.例如,对于文献[17]中的 subtyping 关系$\leq_{pt}$,$\leq_{pj}$ 及$\leq_{lc}$,文献[17]设计出了一些便于应用的 Inheritance-Preserving-Transformation 规则(以下简称 IPT 规则).若将这些 IPT 规则与本文所述的“渐增式继承”规则相对应,可以得出结论:IPT 规则开发得越多,应用者避开继承反常的机会就越多,相应 subtyping 关系的可应用性也越好.

5 结 语

本文形式地给出了“继承反常现象”的一种一般性定义.定义对每一种 Subtyping 关系都有其相对应的“渐增式继承”,使其更具有普适性.文中“渐增式继承”是一个相对的概念,这有助于对人们“使继承反常现象得到缓解”的努力得以分类和评价.一些相关的结论对于并发面向对象语言(包括规范语言)设计及软件开发中继承性的建模有一定的启示.

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是北京大学计算机科学技术系袁崇义教授、屈婉玲教授领导的讨论班上的同学和老师表示感谢.

References:

[1] Wegner, P., Zdonik, S.B. Inheritance as an incremental modification mechanism or what like is and isn't like. In: Gjessing, S., Nygaard, K., eds. Proceedings of the ECOOP'88. LNCS 322, Heidelberg: Springer-Verlag, 1988. 55~77.

[2] Agha, G. Actors: a Model of Concurrent Computation in Distributed Systems. Cambridge, MA: MIT Press, 1986.

[3] America, P. A parallel object-oriented language with Inheritance and subtyping. In: Proceedings of the Conference on Object-Oriented Programming: Systems, Languages and Applications (ECOOP/OOPSLA'90). SIGPLAN Notices Vol 25, ACM Press, 1990. 161~168.

[4] Yonezawa, Akinori. ABCL: an Object-Oriented Concurrent System. Cambridge, MA: MIT Press, 1990.

[5] Matsuoka, Satoshi, Yonezawa, Akinori. Analysis of inheritance anomaly in object-oriented concurrent programming languages. In: Agha, G., Wegner, P., Yonezawa, Akinori, eds. Research Directions in Concurrent Object-Oriented Programming. Cambridge, MA: MIT Press, 1993. 107~ 150.

[6] America, P. Inheritance and subtyping in a parallel object-oriented language. In: Proceedings of the ECOOP'97. LNCS 276, Heidelberg: Springer-Verlag, 1987. 234~242.

[7] Kafura, D.G., Lee, Keung Hae. Inheritance in actor based concurrent object-oriented languages. In: Proceedings of the ECOOP'89. Cambridge: Cambridge University Press, 1989. 131~145.

[8] Lechner, U., Lengauer, C., Nickl, F., *et al*. (Objects+Concurrency)&Reuability——a Proposal to Circumvent the Inheitance Anomaly. In: Cointe, P., ed. Proceedings of the ECOOP'96——OOP. LNCS 1098, Heidelberg: Springer-Verlag, 1996. 222~247.

[9] Crnogorac, Lobel, Rao, A.S., Ramamohanarao, Kotagiri. Classifying inheritance mechanisms in concurrent object-oriented programming. In: Proceedings of the ECOOP'98——Object-Oriented Programming. LNCS 1445, Heidelberg: Springer-Verlag, 1998. 572~600.

[10] Meseguer, José. Solving the inheritance anomaly in concurrent object-oriented programming. In: Proceedings of the ECOOP'93——Object-Oriented Programming. LNCS 707, Heidelberg: Springer-Verlag, 1993. 220~246.

[11] Batiston, E., Chizzoni, A, De Cindo, Fiorella. Inheritance and concurrency in CLOWN. In: Proceedings of the Application and Theory of Petri Net 1995 workshop on object-oriented programs and models concurrency. 1995. http://wrcm.dsi.unimi.it/PetriLab/ws95/abstract/proc9504.html.

[12] Akis, M., Bosch, J., W. Vander Sterren, Bergmans, L. Real-Time specification inheritamce anomalies and real-time filters. In: Jorovo, M., Pareschi, R., eds. Proceedings of the ECOOP'94. LNCS 821, Heidelberg: Springer-Verlag, 1994.

[13] America, P. Designing an object-oriented programming language with behavioural subtyping. In: Proceedings of the REX School/Workshop on Foundations of Object-Oriented Languages (REX/FOOL). LNCS 489, Heidelberg: Springer-Verlag, 1991. 60~90.

[14] Lakos, C. From coloured Petri nets to object Petri nets. In: Proceedings of the 16th International Conference on the Application and Theory of Petri Nets. LNCS 935, Heidelberg: Springer-Verlag, 1995.

[15] Biberstein, Oliver. CO-OPN/2: an object-oriented formalism for the specification of concurrent systems [Ph.D. Thesis]. University of Geneva, 1997. http://www.fcul.research.ec.org/deva/trs/../papers/1.2I.ps.

[16] Liskov, B., Wing, J.M. A behavioural notion of subtyping. ACM Transactions on Programming Languages and Systems, 1994, 16(6): 1811~1841.

[17] Vander Aalst, W.M.P., Basten, J. Life-Cycle inheritance: a Petri-net-based approach. In: Proceedings of the 18th International Conference on the Application and Theory of Petri Nets. LNCS 1248, Heidelberg: Springer-Verlag, 1997. 62~81.

[18] Peirce, B.C. Basic Category Theory for Computer Scientists. Cambridge, MA: MIT Press, 1991.

[19] Mac Lane, S. Categories for the Working Mathematician. Heidelberg: Springer-Verlag, 1971.

Inheritance Anomaly in Concurrent Object Orientation*

WANG Sheng-yuan[1], YANG Liang-huai[2], YUAN Chong-yi[3], YANG Ping[4]

[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*);

[2](*Department of Computer Science, School of Computing, National University of Singapore, Singapore*);

[3](*Department of Computer Science and Technology, Beijing University, Beijing* 100871, *China*);

[4](*School of Information Science and Technology, Lanzhou University, Lanzhou* 730000, *China*)

E-mail: wwssyy@tsinghua.edu.cn; yanglh@comp.nus.edu.sg

http://www.cs.tsinghua.edu.cn

Abstract: The combination of concurrency and object orientation is definitely natural except for inheritance. One of the interferences between inheritance and concurrency is inheritance anomaly. Although having been researched extensively, inheritance anomalies are still only vaguely defined and often misunderstood, and no much formal work has been done. A new viewpoint is set forth for understanding inheritance anomalies, in which each subtyping relation has its specific incremental inheritance. Related concepts and definitions are formalized through the language of Category. Some issues are well adapted to distinguish and explain different standpoints about inheritance anomalies, and can serve as guidelines in the modeling of inheritance.

Key words: concurrency; object orientation; inheritance anomaly; incremental inheritance; category theory

* Received October 8, 2000; accepted December 5, 2001

Supported by the National Natural Science Foundation of China under Grant No.69973003; the National Grand Fundamental Research 973 Program of China under Grant No.G1999032706; the Natural Science Foundation of Gansu Province of China under Grant No.ZS991-A25-014-G