一种基于共代数的面向对象形式语义

余珊珊[1] 李师贤[1] 苏锦钿[2]
(中山大学信息科学与技术学院 广州 510275)[1] (华南理工大学计算机科学与工程学院 广州 510640)[2]

摘 要 针对面向对象方法的数学理论基础相对薄弱的问题,利用共代数方法从范畴论及观察的角度研究面向对象的形式语义及行为关系。首先,给出类和对象的共代数描述,其中抽象类定义成一个类规范,类定义为满足类规范的共代数,类的各个对象则看成共代数状态空间上的元素,并分别利用强 Monads 理论和断言给出方法的行为的参数化描述和语义约束;接着,利用共代数互模拟探讨了不同对象在强 Monads 下的行为等价关系;最后用实例说明如何通过 PVS 工具证明类规范的一致性及对象的行为关系。

关键词 面向对象方法,形式语义,共代数方法,强 Monads
中图法分类号 TP301.2 **文献标识码** A

Formal Semantics of Object Oriented Methods Based on Coalgebras

YU Shan-shan[1] LI Shi-xian[1] SU Jin-dian[2]
(School of Information Science and Technology, Sun Yet-Sen University, Guangzhou 510275, China)[1]
(College of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China)[2]

Abstract According to the problems of relative weak mathematical theory foundations of object oriented methods, the coalgebraic methods were used to analyze the formal semantics of object oriented methods from the perspectives of category theory and observation. Firstly, we presented the coalgebraic descriptions of classes and objects, among which abstract class was defined as a class specification and classes satisfying the class specification were described as coalgebras. Each object belonging to a class was viewed as an element of the state space of the class, as coalgebras. We also used strong Monads theory and assertions respectively to give the parametric descriptions and semantic restrictions of objects' behaviors. Secondly, we further used coalgebraic bisimulation to discuss the behavioral equivalence relationships of objects with the considerations of strong Monads. Finally, we took an example to demonstrate how to use PVS tool to prove the consistence of class specification and objects' behavioral relationships.

Keywords Object oriented methods, Formal semantics, Coalgebraic methods, Strong monads.

1 引言

面向对象方法是目前软件设计及开发中的一种主要技术,被公认为是解决软件危机的一种有效途径。但面向对象方法的数学理论基础研究仍十分薄弱,缺乏一个成熟的语义模型[1,2]。针对该问题,许多学者利用各种形式化方法从不同的角度研究面向对象的形式语义。较早开展这方面研究的学者有 M. Abadi 和 L. Cardelli 等人[3-5],他们主要针对对象、类型和语义等一系列的问题进行了研究,并提出一种基于函数式语言的对象演算以及关于面向对象的逻辑——Abadi-Leino 逻辑。联合国大学国际软件技术研究所的何积丰院士及刘志明团队也对面向对象的形式化进行了深入研究,并利用关系模型提出了一种基于统一理论(Unifying Theories of Programming)的面向对象的语义和精化模型[6]。R. Burstall 等从代数的角度利用带隐藏类别(Hidden Sorts)的基调描述面向对象方法,并研究对象的行为等价关系[7]。其他许多学者也分别利用 B 方法[17]和 Petri 网[8]等给出面向对象方法的形式化描述,并探讨类和对象的各种语义。这些方法偏向于从过程化或规则化的角度描述对象模型,而对封装、继承、多态、组合等关键的面向对象特征难以给出合适的描述。

作为近年来计算机科学基础理论的一个新兴研究方向,共代数方法(Coalgebraic Methods)已经逐渐成为形式化方法研究领域的一个热点,成功地应用于状态转换系统、自动机、进程代数、并发程序语义和面向对象形式语义等领域,并被证明可以构成状态转换系统、进程代数和自动机等形式化方法的理论基础[9]。文献[1]较详细地介绍了目前共代数的研究方向及其应用。

由于对象本身就是一种基于状态的系统,因此以共代数作为其理论基础是非常合适的。这方面的研究主要始于 H. Reichel,他最早在文献[2]中指出了终结共代数(Final Coal-

到稿日期:2010-08-21 返修日期:2010-12-28 本文受国家自然科学基金资助项目(60673122),广东省自然科学基金资助项目(81510 30007000002)资助。

余珊珊(1980—),女,博士生,CCF 会员,主要研究领域为形式语义、软件工程等,E-mail:yushsh@mail.sysu.edu.cn;李师贤(1944—),男,教授,博士生导师,主要研究领域为软件工程、分布计算。

gebras)可以作为面向对象语言的形式化语义基础。随后，B. Jacobs 及其学生在这一领域做出许多贡献。例如，他们利用共代数给出了抽象类、类和对象的共代数描述[10]，同时探讨了继承关系的共自由共代数解释[11]和类规范间的精化关系[12]。这些都构成了本文的研究基础。

在上述研究的基础上，本文将代数理论及函数式程序语言中的强 Monads 理论与共代数相结合，给出抽象类、类和对象的共代数描述，提高了类规范中对行为描述的抽象性。在此基础上，利用共代数互模拟进一步探讨了对象在强 Monads 下的行为等价关系。

本文第 2 节首先给出抽象类和类的共代数描述；第 3 节利用共代数互模拟探讨了对象在强 Monads 下的行为等价关系；第 4 节通过实例说明如何对类规范的一致性和对象的行为关系进行验证；最后是总结并给出下一步的工作。

2 抽象类和类的共代数描述

本文假设读者具有一些共代数和范畴论的知识，关于共代数的详细介绍可参考文献[9,13]。

在面向对象方法中，抽象类描述了所有子类的共同性质，而子类可以根据抽象类给出各自具体的实现。每一个类描述了一组具有共同属性(数据元素)和行为(方法)的对象，并且只允许外界通过有限的接口对其内部状态进行观察和处理。因此，在共代数的视角下，抽象类可视为一个类规范，对应该抽象类的各个子类就可以描述成满足类规范的一个共代数。

定义 1(一个类规范，Class Specification) 定义为一个四元组 $\mathscr{A}=\{X,F,A,C\}$，其中：

(1)X 是由类规范中所有的公有变量及私有变量的类型所构成的状态空间。假定相同名称的变量具有相同的类型。

(2)F 为集合范畴上的自函子 $F:Sets\to Sets$，是由类规范中所有的属性方法和操作方法所构成的有限一元多项式函子。

(3)A 是一个有限公理(或称为断言)集合，用于描述 F 中属性或操作方法的行为约束关系，即给出方法在执行过程中必须满足的行为语义。

(4)C 是一个有限公理集合，描述了类在实例化成对象时需要满足的初始条件。

函子 F 包括两种类型的方法：属性方法 $at:X\times I'\to O'$ 和操作方法 $op:X\times I\to X\times O$。$at$ 给出了对 X 的一个观察值，其执行不会改变 X，即对 $\forall i'\in I'$ 和 $\forall x\in X$ 有 $x.at(i')=at(x)(i')\in O$。操作方法 op 的执行将改变 X 的内部状态，并可能输出结果，即对 $\forall i\in I$ 有 $x.op(i)=op(x,i)\in X\times O$。$F$ 中可能包含多个属性方法，因此可以把 at 进一步表示为 $X\to \coprod_{1\leqslant i\leqslant n} O_i^{I_i'}$ 或 $\coprod_{1\leqslant i\leqslant n}(X\times I_i')\to \coprod_{1\leqslant i\leqslant n} O_i'$。若 at 和 op 为有多个参数 $X\times I_1\times\cdots\times I_n$ 或 $X\times O_1\times\cdots\times O_n$ 的形式，可将 I'，I，O 和 O' 都看成是积 $I_1\times\cdots\times I_n$ 和 $O_1\times\cdots\times O_n$ 的形式。同样地，若 F 中包含多个操作方法，可把 op 表示为 $X\to \coprod_{1\leqslant j\leqslant m}(X\times O_j)^{I_j}$ 或 $\coprod_{1\leqslant j\leqslant m}(X\times I_i)\to \coprod_{1\leqslant j\leqslant m} O_i$。利用 Currying 操作和积可将 at 和 op 方法进一步合并为 $\bar{\alpha}=\langle\overline{at},\overline{op}\rangle:X\to O'^{I'}\times(X\times O)^I$。函数 $l_j:X_j\to \coprod_{i\in I} X_i(1\leqslant j\leqslant I)$ 表示各元素分量到其共积的入射(Injection)，$\pi_j:\coprod_{i\in I} X_i\to X_j(1\leqslant j\leqslant I)$ 表示积到各元素分量的投影射(Projection)。

一个类规范实际上相当于一个抽象类或接口。类规范给出了方法的声明及其行为语义约束，一般不涉及到具体的实现。

给定类规范 $\mathscr{A}=\{X,F,A,C\}$，若 F 在状态 x ($x\in X$)下满足 $\mathscr{A}$ 中的公理集合 A，则记为 $F,x\vdash A$；若对所有的 $x\in X$，都满足 $F,x\vdash A$，则称 F 满足 A，记为 $F\vdash A$。若 F 具有初始状态 $x_0\in X$，且满足 $F\vdash A$ 和 $F,x_0\vdash C$，则称 $(X,\bar{\alpha},x_0)$ 是满足 $\mathscr{A}$ 的一个共代数模型，记为 $(X,\bar{\alpha},x_0)\vdash\mathscr{A}$。若 $\mathscr{A}$ 至少存在一个共代数模型 $(X,\bar{\alpha},x_0)\vdash\mathscr{A}$，则称 $\mathscr{A}$ 为一致的(Consistent)。

根据函数可见性(Visibility)的不同可将函子 F 进一步分为 $F=F_{Pub}\times F_{Pri}$，其中 F_{Pub} 和 F_{Pri} 分别表示可见性为 Public 和 Private 的属性方法及操作方法的集合(为了简单起见，本文不讨论 Protected 方法)。

由类规范 $\mathscr{A}=\{X,F,A,C\}$ 可确定一个抽象类或接口 $(X,\bar{\alpha}:X\to F(X))$，其中基调 $\bar{\alpha}$ 给出了函子 F 的具体实现，并且 $(X,\bar{\alpha})$ 满足 A 中的公理。若 $(X,\bar{\alpha})$ 具有初始状态 x_0，则 $(X,\bar{\alpha},x_0)$ 同时必须满足 C 中的公理。

为了统一和抽象地描述操作方法的各种不同行为(例如确定性、部分或不确定性行为等)，可将代数理论和函数式程序语言(如 Haskell)中的强 Monads 理论[14]与共代数相结合，进一步将 $F(X)=O'^{I'}\times(X\times O)^I$ 表示为 $F(X)=O'^{I'}\times B(X\times O)^I$。这里的 B 是从各种具体的行为中抽象出来的强 monads 结构 (B,η,μ,lt,rt)，其中 (B,η,μ) 表示一个 Monad 结构，强自然转换关系 $lt_{X,Y}:X\times B(Y)\Rightarrow B(X\times Y)$ 和 $rt_{X,Y}:B(X)\times Y\Rightarrow B(X\times Y)$ 分别称为左强度和右强度，用于提供环境相关的信息。对 B 进行实例化就可以得到各种不同的行为描述(见表 1)。

表 1 强 Monads B 与行为间的对应关系

Monads 结构	行为模式
B=Id	表示确定性行为
B=Id+1	表示部分行为，可能存在死锁
B=Id+E	表示结果可能为异常行为 E
B=P	表示非确定性行为
$B=P_\omega$	表示有限非确定性行为
B=Id*	表示有序非确定性行为

定义 2 两个类规范 $\mathscr{A}_1=\{X_1,F_1,A_1,C_1\}$ 和 $\mathscr{A}_2=\{X_2,F_2,A_2,C_2\}$ 间的态射 $\varphi:\mathscr{A}_1\to\mathscr{A}_2$ 是一个共代数态射 $\varphi:(X_1,\alpha_1:X_1\to F_1(X_1))\to(X_2,\alpha_2:X_2\to F_2(X_2))$，并且 φ 保持 A_1 中的公理和 C_1 中的初始化条件。

即对 A_1 中的每一个断言 a 和 C_1 中每一个初始化条件 o，A_2 和 C_2 中都分别存在一个 $\varphi(a)$ 和 $\varphi(o)$ 与之对应。类规范之间的态射给出了两个抽象类或接口间在语法上的一种关系。

定义 3 给定两个类规范 $\mathscr{A}_1=\{X_1,F_1,A_1,C_1\}$ 和 $\mathscr{A}_2=\{X_2,F_2,A_2,C_2\}$，若满足条件 $X_2\subseteq X_1$，$F_2\subseteq F_1$，$A_2\subseteq A_1$ 和 $C_2\subseteq C_1$，则称 $\mathscr{A}_2$ 是 $\mathscr{A}_1$ 的一个子规范，记为 $\mathscr{A}_2\subseteq\mathscr{A}_1$。

若 $\mathscr{A}_2$ 是 $\mathscr{A}_1$ 的一个子规范，意味着 F_2 是 F_1 的一个子函子(Subfunctor)，而且 A_2 和 C_2 为 A_1 和 C_1 中的一个子集。

类规范可以进行合并，构成新的类规范。

定义 4 两个类规范 $\mathscr{A}_1$ 和 $\mathscr{A}_2$ 之间的合并关系表示为 $\mathscr{A}=\mathscr{A}_1||_{\mathscr{A}_0}\mathscr{A}_2$，其中 $\mathscr{A}_0$ 为 $\mathscr{A}_1$ 和 $\mathscr{A}_2$ 的公共子规范，表示它们之间的共同部分。$\mathscr{A}=\{X,F,A,C\}$ 定义为 $X=_{df}X_1\cup X_2$，$F=_{df}F_1\cup F_2$，$A=_{df}A_1\cup A_2$，$C=_{df}C_1\cup C_2$。

若 $\mathscr{A}_0$ 为空规范，可记为 $\mathscr{A}=\mathscr{A}_1||\mathscr{A}_2$。根据集合并关系的性质容易证明以下定理 1 以及引理 1 成立。

定理 1 子规范及契约合并满足：

(1) $\mathscr{A}\subseteq\mathscr{A},\mathscr{A}||_{\mathscr{A}}\mathscr{A}=\mathscr{A}$(自反性)；

(2) $\mathscr{A}_1||_{\mathscr{A}_0}\mathscr{A}_2\equiv\mathscr{A}_2||_{\mathscr{A}_0}\mathscr{A}_1$(对称性)；

(3) $(\mathscr{A}_1||_{\mathscr{A}_0}\mathscr{A}_2)||_{\mathscr{A}_0}\mathscr{A}_3\equiv\mathscr{A}_1||_{\mathscr{A}_0}(\mathscr{A}_2||_{\mathscr{A}_0}\mathscr{A}_3)$(结合性)；

(4) $\mathscr{A}_2\subseteq\mathscr{A}_1\wedge\mathscr{A}_3\subseteq\mathscr{A}_2\Rightarrow\mathscr{A}_3\subseteq\mathscr{A}_1$(传递性)；

(5) $(\mathscr{A}_1\subseteq\mathscr{A}_2)||_{\mathscr{A}_0}\mathscr{A}_3\Rightarrow(\mathscr{A}_1||_{\mathscr{A}_0}\mathscr{A}_3)\subseteq(\mathscr{A}_2||_{\mathscr{A}_0}\mathscr{A}_3)$(||对$\subseteq$的分布性)。

引理 1 类规范的合并满足：

(1) $\mathscr{A}||\mathscr{A}=\mathscr{A}$(自反性)；

(2) $\mathscr{A}_1||\mathscr{A}_2\equiv\mathscr{A}_2||\mathscr{A}_1$(对称性)；

(3) $(\mathscr{A}_1||\mathscr{A}_2)||\mathscr{A}_3\equiv\mathscr{A}_1||(\mathscr{A}_2||\mathscr{A}_3)$(结合性)。

定义 5 给定两个类规范 $\mathscr{A}_1=\{X_1,F_1,A_1,C_1\}$ 和 $\mathscr{A}_2=\{X_2,F_2,A_2,C_2\}$，若满足条件 $X_1\subseteq X_2$，$F_1\subseteq F_2$，$A_2\vdash A_1$ 和 $C_2\vdash C_1$，并且对所有的 $(U,\bar{\alpha},u_0)\models\mathscr{A}_2$，都有 $(U,\rho\circ\bar{\alpha},u_0{}')\in P(u_0))\models\mathscr{A}_1$，则称 $\mathscr{A}_2$ 是 $\mathscr{A}_1$ 的一个规范精化，记为 $\mathscr{A}_2\sqsubseteq\mathscr{A}_1$。

其中，$u_0{}'$ 是 U 中的某一个状态，$P(u_0)$ 为一个包含 u_0 的最小不变量(Least Invariant)，表示只包含在 $\bar{\alpha}$ 下所有从 u_0 可达的状态，即 $u_0{}'\in P(u_0)$ 意味着 $u_0{}'$ 是从初始状态 u_0 可达的。$A_2\vdash A_1$(或 $C_2\vdash C_1$)表示由 A_2(或 C_2)能够推导出 A_1(或 C_1)中的所有断言。

由于类规范中可能包含私有方法，并且断言是定义在私有方法上，而我们在探讨对象的行为关系及语义的时候往往只关注对象通过公有方法所展示出来的外部可观察行为，而忽略对象中的私有方法及内部行为，因此可定义类规范的一个公有方法视图。

定义 6 若类规范 $\mathscr{A}$ 中的函子 $F=F_{Pub}\times F_{Pri}$ 包含私有方法(即 $F_{Pri}\neq\emptyset$)，那么可定义 $\mathscr{A}$ 的一个公有方法视图 $\mathscr{A}'=\{X,\rho:F\Rightarrow F_{Pub},A',C'\}$，其中 ρ 是函子 F 到 $F_{Pub}F_{Pub}$ 的一个自然转换，A' 和 C' 分别表示只包含定义在 F_{Pub} 中的方法上的断言集合。

自然转换 ρ 可以看成是函子 F 到 F_{Pub} 的一个投影射，从而将 F_{Pri} 中的方法丢弃。若 $F_{Pri}=\emptyset$，显然 ρ 就是 F 上的标识函子，这时候 $\mathscr{A}$ 与 $\mathscr{A}$ 相同。

针对类规范所提供的具体实现就构成了一个类，即相当于抽象类的一个子类。

定义 7 类是满足某个类规范 $\mathscr{A}=\{X,F,A,C\}$ 的一个具体实现，表示为一个共代数模型 $(U,\bar{\alpha}:U\to F(U),u_0\in U)$，其中：

(1)载体集 U 为有限的状态空间，给出了 $\mathscr{A}$ 中 X 的具体解释；

(2)基调 $\bar{\alpha}$ 给出了类规范中函子 F 在 U 上的具体实现，并且满足 $\mathscr{A}$ 中的公理集合 A；

(3)初始状态 $u_0\in U$ 给出了类的构造行为的解释，且 u_0 满足 $\mathscr{A}$ 中的初始化条件 C，记为 $C(u_0)$。

满足类规范 $\mathscr{A}$ 的类就描述了 $\mathscr{A}$ 的实现语义信息，可简单记为 $(U,\bar{\alpha},u_0)\models\mathscr{A}$。从面向对象的角度来看，$(U,\bar{\alpha},u_0)$ 是由类规范 $\mathscr{A}$ 所确定的抽象类的一个子类。

定理 2 满足类规范 $\mathscr{A}$ 的所有类可构成一个类范畴 $Class(\mathscr{A})$，其中：

(1) $Class(\mathscr{A})$ 中的对象为类 $(U,\bar{\alpha},u_0)|=\mathscr{A}$；

(2) $Class(\mathscr{A})$ 中的射 $f:(U_1,\overline{\alpha_1},u_0^1)\to(U_2,\overline{\alpha_2},u_0^2)$ 为状态空间上的函数 $f:U_1\to U_2$，且满足以下条件：

a) f 保持状态间的互相似关系，即对于 $u,v\in U_1$，有 $u\leftrightarrow v\Rightarrow f(u)\leftrightarrow f(v)$；

b) $\overline{at_2}\circ f=\overline{at_1}:U_1\to O^{I'}$；

c) $\overline{op_2}\circ f=B(f\times O)^I\circ\overline{op_1}$；

d) $f(u_0)\leftrightarrow v_0$。

显然，$Class(\mathscr{A})$ 中的射实际上就是保持初始状态的共代数态射，且满足复合性。因此容易证明 $Class(\mathscr{A})$ 确实是一个范畴。

例 1 假设有一个银行帐号的抽象类，用共代数类规范语言 CCSL(Coalgebraic Class Specification Language)[15] 描述如下：

```
Begin Account:ClassSpec
  Public Method
    balance:Self -> Int; %帐户余额
    deposit:[Self,Int]-> Self;%存款
    debit:[Self,Int]-> Self; %借款
  Assertion
    SelfVar s:Self
    Var i:Int
    dep_bal:s. deposit(i). balance=s. balance+i;
    deb_bal:if s. balance>i then s. debit(i). balance=s. balance-i else
            s. balance=0;
    dep_deb:s. deposit(i). debit(i) ~ s;
      Creation
    SelfVar s:Self
init_account:s. balance=0
End Account
```

由上述的共代数描述可以确定一个 Account 类规范 $\mathscr{A}=\{X,F,A,C\}$，其中 X 表示 Self，函子 $F=\langle balance:X\to Int,deposit:X\to X^{Int},debit:X\to X^{Int}\rangle$，有限公理集 $A=$ {dep_bal, deb_bal,dep_deb}，初始化条件 $C=$ {init_account}。

由 Account 类规范所确定的抽象类可得到各个不同的子类，每个子类给出各自具体的实现语义信息。例如，$C_1=\{Int,balance:Int\to Int,deposit:Int\times Int\to Int,debit:Int\times Int\to Int\}$ 就是满足条件的一个类，其中 X 用整数类型 Int 表示，基调 $\overline{\alpha_1}$ 中的各个方法定义为 $\forall s,i\in Int,balance(s)=s,deposit(s,i)=s+i,debit(s,i)=$ if $s\geqslant i$ then $(s-i)$ else 0。类 C_1 相当于以下的 Java 类。

```
Class C1{
    Private int s;//当前金额
    Public void deposit(int i) {s=s+i};
    Public int balance() { return s };
    Public void debit(int i) {
```

```
    if s >= i then s=(s -i) else s=0 }
  Public C1() { s=0}
}
```

类似地，可以给出另一个类 C_2。C_2 以数据结构 $List[Int]$ 作为帐户存借款记录，即 C_2 定义为 $C_2=\{List[Int], balance:List[Int]\to int, deposit:List[Int]\times Int\to List[int], debit:List[Int]\times Int\to List[Int]\}$，其中状态空间为包含整数的数组 $List[Int]=[i_0,i_1,i_2,\cdots,i_n]$，$i_0=0$ 表示初始状态。基调 $\overline{\alpha_2}$ 中的各个方法定义为

$$balance([i_0,i_1,i_2,\cdots,i_n])=i_n$$

$$deposit([i_0,i_1,i_2,\cdots,i_n],i)=[i_0,i_1,i_2,\cdots,i_n,i_{n+1}=i_n+i]$$

$$debit([i_0,i_1,i_2,\cdots,i_n],i)=\begin{cases}[i_0,i_1,i_2,\cdots,i_n,i_n-i] & (i_n-i\geqslant 0)\\ [i_0,i_1,i_2,\cdots,i_n,0] & (i_n-i<0)\end{cases}$$

由于类规范的各个子类均表示为同一函子下的不同共代数，因此可利用共代数态射来比较它们之间在实现上的某种关系，例如一个实现是否比另一个实现的效率更高。这也给面向对象程序的自动变换和求精提供了一个可行的途径。

3 对象及其行为关系

类的每一个对象可以看作是共代数状态空间中的某个元素。例如，$(3,\overline{\alpha_1})$ 和 $(1,\overline{\alpha_1})$ 是 C_1 的两个不同对象，而 $([0,2,1,3],\overline{\alpha_2})$ 和 $([0,2,1,3,1],\overline{\alpha_2})$ 是 C_2 的两个不同对象。

由于用户无法直接了解对象的内部状态，只能通过对象中的公有方法来观察其外部行为，因此可以利用共代数互模拟研究两个对象在其公有方法下的行为等价关系。

定义 8 给定一个类规范 $\mathscr{A}=\{X,F,A,C\}$ 及其公有方法视图 $\mathscr{A}'=\{X,\rho:F\Rightarrow F_{Pub},A',C'\}$，$C_1=(U_1,\overline{\alpha_1},u_0^1)$ 和 $C_2=(U_2,\overline{\alpha_2},u_0^2)$ 为满足 $\mathscr{A}$ 的两个类，$u_1\in U_1$ 和 $u_2\in U_2$ 分别为 C_1 和 C_2 中的两个对象。称对象 $u_1\in U_1$ 和 $u_2\in U_2$ 是互相似的（记为 $u_1\leftrightarrow u_2$）当且仅当存在 U_1 和 U_2 上的一个二元关系 $R\subseteq U_1\times U_2$，使得 R 为 $C_1'=(U_1,\rho\circ\overline{\alpha_1})$ 和 $C_2'=(U_2,\rho\circ\overline{\alpha_2})$ 间的一个共代数互模拟 $(R,\overline{\alpha_R'}:R\to F_{Pub}(R))$，并且 $(u_1,u_2)\in R$。若 $(u_0^1,u_0^2)\in R$，则称 C_1 和 C_2 在公有方法下是行为等价的，记为 $C_1\approx C_2$。

直观地讲，两个对象是行为等价的当且仅当对它们的观察在公有方法下是不可区分的。更具体地，对于上述的各个基调 $\overline{\alpha_1}=\langle\overline{at_1},\overline{op_1}\rangle$，$\overline{\alpha_2}=\langle\overline{at_2},\overline{op_2}\rangle$，$\overline{\alpha_R'}=\langle\overline{at_R'},\overline{op'_R}\rangle$ 和投影射 $\rho:F\Rightarrow F_{Pub}$，有 $\rho\circ\overline{\alpha_1}=\langle\overline{at_1'},\overline{op_1'}\rangle=\overline{\alpha_1'}$，$\rho\circ\overline{\alpha_2}=\langle\overline{at_2'},\overline{op_2'}\rangle=\overline{\alpha_2'}$，并且

$$(1)\rho\circ\overline{\alpha_1}\circ\pi_1=\rho\circ\langle O'^{I'}\times B(\pi_1\times O)^I\rangle\circ\langle\overline{at'_R},\overline{op_R'}\rangle$$
$$=\langle\overline{at'_R},\rho\circ B(\pi_1\times O)^I\circ\overline{op'_R}\rangle$$
$$=\langle\overline{at'_R},\overline{\rho\circ B(\pi_1\times O)\circ op'_R}\rangle$$

$$(2)\rho\circ\overline{\alpha_2}\circ\pi_2=\langle\overline{at'_R},\overline{\rho\circ B(\pi_2\times O)\circ op_R'}\rangle$$

如图 1 所示。

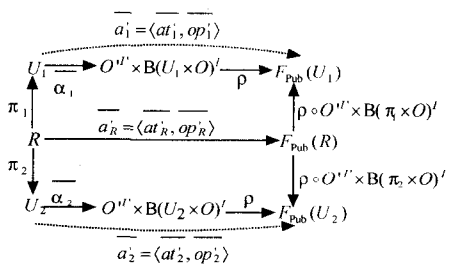

图 1 对象间的互模拟关系

即意味着对任意的 $(u_1,u_2)\in U_1\times U_2$，满足：

$(1)\overline{\alpha'_R}(u_1,u_2)=\overline{\alpha_1'}(u_1)=\overline{\alpha_2'}(u_2)$；

$(2)\overline{\rho\circ B(\pi_1\times O)\circ op'_R}(u_1,u_2)=\rho\circ\overline{op_1}(u_1)=\overline{op_1'}(u_1)$；

$(3)\overline{\rho\circ B(\pi_2\times O)\circ op'_R}(u_1,u_2)=\rho\circ\overline{op_2}(u_2)=\overline{op_2'}(u_2)$。

由于对象中方法的行为通过强 Monads 进行参数化描述，因此在定义具体的互模拟关系时必须考虑强 Monads B 的结构。例如，对 $B=Id$，$B=Id+1$，$B=P_\omega$ 和 $B=Id^*$ 的互模拟定义分别为

$B=Id:(u_1,u_2)\in R\Leftrightarrow\forall i'\in I'.\ at_1'(u_1,i')=at_2'(u_2,i')\wedge\forall i\in I(\forall(u_1',o)=op_1'(u_1,i).\ \exists(u_2',o)=op_2'(u_2,i).(u_1',u_2')\in R\wedge\forall(u_2',o)=op_2'(u_2,i).\ \exists(u_1',o)=op_1'(u_1,i).(u_1',u_2')\in R)$

$B=Id+1:(u_1,u_2)\in R\Leftrightarrow\forall i'\in I'.\ at_1'(u_1,i')=at_2'(u_2,i')\wedge\forall i\in I((\forall(u_1',o)=op_1'(u_1,i).\ \exists(u_2',o)=op_2'(u_2,i).(u_1',u_2')\in R\wedge\forall(u_2',o)=op_2'(u_1,i).\ \exists(u_1',o)=op_1'(u_1,i).(u_1',u_2')\in R)\vee(op_1'(u_1,i)=op_2'(u_2,i)=*))$

$B=P_{\omega}:(u_1,u_2)\in R\Leftrightarrow\forall i'\in I'.\ at_1'(u_1,i')=at_1'(u_2,i')\wedge\forall i\in I(\forall(u_1',o)=op_1'(u_1,i).\ \exists(u_2',o)=op_2'(u_2,i).(u_1',u_2')\in R\wedge\forall(u_2',o)=op_2'(u_2,i).\ \exists(u_1',o)=op_1'(u_1,i).(u_1',u_2')\in R)$

$B=Id^*:(u_1,u_2)\in R\Leftrightarrow\forall i'\in I'.\ at_1'(u_1,i')=at_2'(u_2,i')\wedge\forall i\in I(\forall op_1'(u_1,i)=([u_1^1,\cdots,u_n^1],o).\ \exists op_2'(u_2,i)=([u_1^2,\cdots,u_n^2],o).(u_j^1,u_j^2)\in R\wedge\forall i\in I(\forall op_2'(u_2,i)=([u_1^2,\cdots,u_n^2],o).\ \exists op_1'(u_1,i)=([u_1^1,\cdots,u_n^1],o).(u_j^1,u_j^2)\in R$

例如，对于前面 Account 类规范，$c_1=(3,\overline{\alpha_1})$ 和 $c_2=([2,1,3],\overline{\alpha_2})$ 分别为 C_1 和 C_2 中的对象，容易证明存在一个共代数互模拟 $R\subseteq Int\times List[Int]$，使得 $(3,[2,1,3])\in R$，即 $3\leftrightarrow[2,1,3]$，其中互模拟关系 R 定义为 $R=\{(s,[i_0,i_1,i_2,\cdots,i_n])\in Int\times List[Int]\mid s=i_n\}$。

4 实例与验证

由于一般情况下很难直接从形式化的角度证明类规范本身的正确性，因此可以通过在 PVS 中建立相应的模型来验证规范的一致性（即不存在逻辑错误或无法实现的操作等问题）、目标行为的正确性和其他的性质（如不同对象之间的行为等价关系或精化关系）。利用 CCSL 给出类规范的描述后，可以通过 CCSL Compiler 将类规范转换成满足 PVS 格式的高阶逻辑，然后在 PVS 理论证明工具中建立各种模型进行验证和理论证明。CCSL 的具体工作环境如图 2 所示。

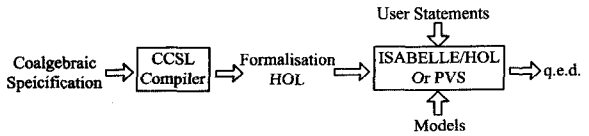

图 2 CCSL 工作环境

以本文的 Account 类规范为例，利用 CCSL Compiler 对共代数规范文件 Account. beh 进行编译后可得到相应的 PVS 文件 Account_basic. pvs。该文件包含了一系列用于描述抽象类 Account 的行为、接口、互相似性和不变量的理论，而 Account 中的各个断言也被转换成 PVS 中相应的谓词。Account_basic. pvs 中所包含的理论可以建立各种模型来验证规范的一致性以及模型之间的行为等价或精化关系。例如，可以建立以下模型 C1 来证明 Account 类规范的一致性：

```
C1:Theory
  Begin
    AState:Type={x:int |x>=0}
    PosInt:Type={x:int |x>=0}
    IMPORTING AccountBasic[AState]
    int_Account:AccountSignature[AState]=
    (#
    balance=Lambda(x:AState):x,
            deposit=Lambda(x:AState,i:PosInt):x+i,
            debit=Lambda(x:AState,i:PosInt):
                  If x>=i Then x-i Else 0 EndIf
    #)
    int_new:AccountConstructors[AState] =
            (# new_Account:=0 #)
    x:Var AState
    int_assert:Lemma AccountAssert? (int_Acc);
    int_create:Lemma AccountCreate? (int_Acc);
    int_model:Proposition AccountModel? (int_Acc,int_new)
END C1
```

要证明 C1 是 Account 类规范的一个实现，只需要证明上述 Theory 中的命题 AccountModel? 成立即可，即分别证明所定义的类满足类规范中的各个断言。同理，可以建立以 List 为状态空间的模型 C2，并可证明 C1 和 C2 均为满足条件的子类。

除了证明规范的一致性外，在 PVS 中还可以利用态射、不变量、共代数互模拟、终结共代数等证明类和对象的性质及其行为语义。

结束语 共代数方法非常适合作为面向对象方法的数学理论基础，能够从函数形式的角度给出抽象类、类和对象等关键概念的参数化描述，并且可借助范畴论从抽象和统一的理论角度探讨面向对象方法的形式语义和行为关系，而强 Monads 理论可以进一步提高对类规范和对象行为描述的抽象性。

在下一步的工作中，我们将继续对强 Monads 和终结共代数下的对象行为语义进行深入的分析，并结合共归纳证明原则探讨对象间的行为等价关系。

参 考 文 献

[1] 周晓聪，舒忠梅. 计算机科学中的共代数方法的研究综述[J]. 软件学报，2003，14(10)：1661-1671

[2] Reichel H. An Approach to Object Semantics Based on Terminal Coalgebras [J]. Mathematical Structures in Computer Science, 1995,5(2):129-152

[3] Abadi M, Cardelli L. A Semantics of Object Types [C]// Ninth Annual IEEE Symposium on Logic in Computer Science (LICS). IEEE Computer Society, 1994:332-341

[4] Abadi M, Cardelli L. A Theory of Primitive Objects: Untyped and First-order Systems [J]. Information and Computation, 1996,125(2):78-102

[5] Abadi M, Cardelli L, Viswanathan R. An Interpretation of Objects and Object Types [C]// 23rd Annual ACM Symposium on Principles of Programming Languages (POPL). ACM, 1996: 396-409

[6] Jifeng H, Zhiming L, Xiaoshan L. rCOS: A Refinement Calculus of Object Systems [J]. Theor. Comput. Sci., 2006, 365(1/2): 109-142. UNU-IIST TR 322. http://rcos.iist.unu.edu/publications/TCSprepreint.pdf

[7] Burstall R, Diaconescu R. Hiding and Behaviour: an Institutional Approach [C]// Roscoe A W, ed. A Classical Mind. Essays in honour of C. A. R. Hoare, Prentice Hall, 1994:75-92

[8] 杨武，李晓瑜，曹泽瀚. 一种面向对象 Petri 网模型的语义和行为分析[J]. 计算机科学，2005，32(10)：219-221

[9] Rutten J. Universal Coalgebra: a Theory of Systems [J]. Theoretical Computer Science, 2000:3-80

[10] Jacobs B. Objects and Classes, Co-algebraically[C]// Freitag B, Jones C B, Lengauer C, eds. Object-orientation with Parallelism and Persistence. Kluwer Acad. Publ, 1996:83-103

[11] Jacobs B, Rutten J. Inheritance and Cofree Construction [C]// Cointe P, ed. European Conference on Object- oriented Programming. Lecture Notes in Computer Science, Vol. 1098. Berlin: Springer, 1996:210-231

[12] Jacobs B. Invariants, Bisimulations and the Correctness of Coalgebraic Refinements [C]// Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology. Lecture Notes in Computer Science, Vol. 1349. 1997:276-291

[13] Gumm H P. Elements of the General Theory of Coalgebras [EB/OL]. http://www.mathematik.uni-marburg.de/~gumm/papers/ publ.html, 1999

[14] Moggi E. Notions of Computation and Monads [J]. Inf. & Comp., 1991, 93(1):55-92

[15] Tews H. The Coalgebraic Class Specification Language CCSL-Syntax and Semantics [R]. TUD-FI02-08-August. TU Dresden, 2002

[16] Hasuo I, Jacobs B. Traces for Coalgebraic Components [J]. Mathematical Structures in Computer Science, 2010

[17] Tafat A, Boulmé S, Marché C. A Refinement Methodology for Object-oriented Programs [C] // International Conference on Formal Verification of Object-oriented Software. 2010:143-159