程序语言中的共归纳数据类型及其应用

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

摘 要 归纳数据类型利用代数方法从构造的角度归纳地描述数据类型的有限语法结构,但在描述动态行为方面存在一定的不足。作为归纳数据类型的范畴对偶概念,共归纳数据类型利用共代数方法从观察的角度共归纳地描述了数据类型的动态行为。首先,从范畴论和代数的角度给出程序语言中的归纳数据类型定义,并分析了相应的递归操作;接着,利用共代数给出共归纳数据类型的范畴论定义,并根据共归纳数据类型的终结性分析了相应的共递归操作;最后,指出如何利用λ-双代数及分配律将归纳与共归纳数据类型有机地融合起来,探讨数据类型的语法构造与动态行为关系。

关键词 归纳数据类型,共归纳数据类型,范畴论,代数,共代数,双代数

中图法分类号 TP301.2 **文献标识码** A

Coinductive Data Types and their Applications in Programming Languages

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

Abstract Inductive data types mainly focus on the finite syntactic structures inductively in terms of algebras from the construction perspective, but have some disadvantages in describing dynamic behaviors. As their categorical dual notions, coinductive data types aim to coinductively describe the observable behaviors of data types in terms of coalgebras from the observation perspective. We firstly gave the definitions of inductive data types in programming languages from the categorical and algebraic viewpoints. After that, we continued to present the definition of coinductive data types with coalgebras and analyze the corresponding corecursion operations according to the finality of coinductive data types. Finally, we pointed out how to use λ-bialgebras and distributive laws to combine inductive and coinductive data types and discuss the relations between syntactic constructions and dynamic behaviors of data types.

Keywords Inductive data type, Coinductive data type, Category theory, Algebras, Coalgebras, Bialgebras

1 引言

抽象数据类型(Abstract Data Type, ADT)一直以来是程序语言及类型理论的一个重要组成部分和研究重点。传统的ADT主要以归纳数据类型(Inductive Data Type)为主,侧重于描述数据类型的有限语法构造,一般包含一些称为“构造子”(Constructor)的操作。例如,有限分支树、有限数组、队列、堆栈、自然数和集合等都是典型的归纳数据类型。在范畴论的视角下,每一个归纳数据类型都可以抽象地描述为某个代数函子下的初始代数,其中代数函子对应着该数据类型的构造操作。因此,归纳数据类型也经常被称为代数数据类型(Algebraic Data Types)。由初始代数的初始性可以定义归纳数据类型上的一个递归操作,称为折叠(fold)[1]操作或 catamorphism[2],且该操作满足一系列的代数计算定律。归纳数据类型上的递归操作及其计算定律在程序的计算及转换研究中具有重要的作用[3,4]。

代数方法对归纳数据类型的研究起了非常重要的作用,例如,ADj 研究小组对 ADT 的主要研究思路是将数据结构看成是封装了数据类型和操作的代数结构。但代数方法适合于描述数据类型的静态语法构造,在刻画其动态行为方面则存在一定的不足。

共归纳数据类型(Coinductive Data Types)[5],也称为共代数数据类型[6]或共数据类型(Codata Types)[7,8],是近年来类型理论中研究数据类型的一种新思路,可看成是归纳数据类型的范畴对偶概念。例如,分支树、数组和流等都可以看成是典型的共归纳数据类型。与归纳数据类型不同,共归纳数据类型侧重于描述数据类型在执行过程中所展现出来的(无限)动态行为。在范畴论的视角下,这一类具有动态行为特征的数据类型可以进一步抽象为某个共代数函子下的终结共代数(Final Bialgebras)[9],其中共代数函子对应着该数据类型

到稿日期:2010-12-23 返修日期:2011-03-21 本文受2010年高校博士点科研基金-新教师类(20100172120043),华南理工大学中央高校基本科研业务费专项资金(2009ZM0158)资助。

苏锦钿(1980—),男,博士,讲师,主要研究方向为形式化方法及形式语义、构件技术、共代数与双代数,E-mail:SuJD@scut.edu.cn;**余珊珊**(1980—),女,博士生,CCF会员,主要研究方向为形式语义、软件工程等。

上的观察操作。由终结共代数的终结性可定义该数据类型上的一个共递归(Corecursion)操作,称为 unfold[10,11]操作或 atamorphism[2],且该操作满足一系列的共代数计算定律。共归纳数据类型上的共递归操作及其计算定律在基于行为特征的函数定义和程序推理及转换过程中具有重要的作用[5,8,12-16]。

共归纳数据类型以共代数作为其数学理论基础,因此能够将共代数中的互模拟、终结共代数、共归纳原理等理论引入到类型理论中,大大地增强了数据类型对动态行为的描述和验证能力,并与归纳数据类型形成互补,从而为研究数据类型的语法构造与动态行为之间的关系提供了一种可行的途径。

本文的主要工作是从范畴论和共代数的角度,探讨共归纳数据的性质及其在程序语言中的应用。第 2 节首先从代数的角度探讨归纳数据类型;第 3 节给出共归纳数据类型的共代数描述,分析相应的共递归操作;第 4 节探讨了如何通过 λ-双代数及分配律对归纳和共归纳数据类型进行融合;第 5 节分析了当前的一些相关研究工作;最后是总结并给出下一步的工作。

2 归纳数据类型

归纳数据类型一般包含一些构造子操作,而该数据类型的值都可通过这些构造子在有限步骤内递归地构造出来。例如,给定数据类型 A,其上的构造子 Σ 通常具有形如 $\Sigma:A_1\times\cdots\times A_n\to A$ 的形式,其中 A_i 是某些数据类型。直观地说,构造子 Σ 就是从已有的数据类型构造出类型为 A 的数据。程序语言中广泛使用的数组和自然数等都是典型的归纳数据类型。

例 1 给定一个任意的集合 A,设 $A^\infty=A^*\cup A^N$ 为所有包含 A 中元素的有限数组 A^* 和无限数组 A^N 的集合。即 A^∞ 中的每一个元素 $\delta\in A^\infty$ 或是形如 $\delta=\langle a_1,\cdots,a_n\rangle$ 的有限数组,或是形如 $\delta=\langle a_1,a_2,\cdots\rangle$ 的无限数组,其中 $\forall i\geq 1, a_i\in A$。$A^\infty$ 上的操作定义为:$emtpy:1\to A^\infty$,$cons:A\times A^\infty\to A^\infty$。

初始化操作 $empty$ 用于将数组初始化为空,插入操作 $cons$ 将一个类型为 A 的元素插入到一个现有的数组中。显然,A^∞ 就是程序语言中用于保存类型为 A 的元素的有限或无限数组,可看成是由集合 A 及其上的初始化操作 $empty$ 和插入操作 $cons$ 迭代构造而成。

例 2 自然数 N 是由类别 nat 及其上的一组基调 $zero:0\to$ nat,$succ:$ nat $\to$ nat,$add:$ nat $\times$ nat $\to$ nat 等所构成,其中初始化函数 $zero:0\to$ nat 用于将自然数初始化为 0,后续函数 $succ$ 映射 $succ^k(0)\mapsto succ^{k+1}(0)$,加法函数 add 映射 $succ^k(0)+succ^l(0)\mapsto succ^{k+l}(0)$。

由于归纳数据类型可以抽象地看成是在数据集合上给出一组满足某些性质的操作,用于从已有的集合成员构造新的集合成员,因此其本质上就是一种代数。在计算机科学中,常用代数方法来研究归纳数据类型。代数上的操作一般具有 $\alpha_X:T(X)\to X$ 的形式,其中 $T:Set\to Set$ 为集合范畴上的自函子,X 为某个数据类型集合,称为载体集。α_X 称为基调(Signature),可看成是函子 T 在 X 上的构造操作。

下面首先给出范畴论中的代数定义。

定义 1 给定集合范畴 Set 和其上的自函子 $T:Set\to Set$,函子 T 上的代数定义为一个二元组 $(X,\alpha_X:T(X)\to X)$,其中 X 是 Set 中的对象,称为该 T-代数的载体,$\alpha_X:T(X)\to X$ 是 Set 中的射,称为该 T-代数的结构射或基调。任意两个 T-代数 $(X,\alpha_X:T(X)\to X)$ 和 $(Y,\alpha_Y:T(Y)\to Y)$ 间的同态射 $f:(X,\alpha_X)\to(Y,\alpha_Y)$ 是 Set 中的射 $f:X\to Y$,且满足图 1 所示的图表交换。

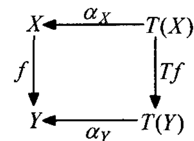

图 1 代数间的同态射

例如,上述的数组可以描述为代数 $ListA=(A^\infty,[nil,cons]:1+A\times A^\infty\to A^\infty)$,自然数 N 可以描述为代数 $N=(\text{nat},[zero,succ,add]:0+\text{nat}+\text{nat}\times\text{nat}\to\text{nat})$。

归纳数据类型不仅可以抽象地描述为代数,而且是以构造子为运算的初始代数。例如,由 $\langle\rangle$,$cons(a_1,\langle\rangle)$,$cons(a_2,cons(a_1,\langle\rangle))$,$\cdots$,$cons(\cdots(cons(a_n\cdots(cons(a_1,\langle\rangle)))))$ 所构成的数组是函子 $T(X)=1+A\times X$ 下的初始代数。自然数中由所有的闭项 0,$succ(0)$,$succ(succ(0))$,$\cdots$,$succ^k(0)$ 所构成的项代数是函子 $T(X)=1+X+X\times X$ 下的初始代数。

归纳数据类型由于可描述成初始代数,因此具有某种泛性质,即到同一函子下的其他任意代数都存在一个唯一的同态射,称为 fold 操作或 catamorphism。fold 实际上给出了归纳数据类型上的一种递归操作的定义。利用 fold 的唯一性及初始代数上的同余是等价关系的性质可以得到所谓的归纳原理(包括归纳定义和证明原则)。

例 3 给定一个集合 X,及其上的元素 $e\in X$ 和函数 $g:X\to X$,则存在从自然数集 nat 到 X 的唯一函数 $f:\text{nat}\to X$,使得对所有的 $n\in\text{nat}$ 满足:

$f(0)=e$ 和 $f(succ(n))=g(f(n))$。

显然,函数 f 在 0(0 为 nat 中序关系下的最小元素)上的值 e 就是数学归纳法中的归纳基,而 g 为归纳步骤。

例 4 要定义数组 A^∞ 上的一个求长度的操作 $len:A^\infty\to$ nat,可先定义 nat 上的两个操作 $zero:0\to$ nat 和 $succ:$ nat $\to$ nat,使得满足以下等式:

(1) $len(nil)=0$;

(2) $len(cons(a,L))=succ(len(L))$,其中 $L\in A^\infty$。

显然,len 是一个代数同态射 $len:(A^\infty,[nil,cons])\to(\text{nat},[zero,succ\circ\pi_2])$,其中 $\pi_2:A\times\text{nat}\to\text{nat}$ 为笛卡尔积上的一个投影射,$succ\circ\pi_2:A\times\text{nat}\to\text{nat}$ 使得 nat 具有与 A^∞ 相同的操作类型,即可看成是同一函子下的不同基调。

例 5 要定义函数 $double:A^\infty\to A^\infty$,它将原来数组中的每个元素变成两个相邻的元素,即:

$double(cons(a_1,cons(a_2,nil)))=cons(a_1,cons(a_1,cons(a_2,cons(a_2,nil))))$

只需要定义 A^∞ 上的操作 $(A^\infty,[nil,doubleCons]:1+A\times A^\infty\to A^\infty)$,使得 $nil:1\to A^\infty$,且对于 $\forall a\in A, L\in A^\infty$ 有 $doubleCons(a,L)=cons(a,cons(a,L))$,容易验证 $double:(A^\infty,[nil,cons])\to(A^\infty,[nil,doubleCons])$ 就是一个代数同态射,且为目标函数。

3 共归纳数据类型

共归纳数据类型可看成是归纳数据类型的对偶概念,它

关注的不是数据类型的内部语法构造，而是从观察的角度探讨数据类型的外部行为关系，即在运行过程中所展现出来的动态行为特征。

例 6　下面是一些典型的共归纳数据类型：

(1)流(Stream)：流一般包含两个操作，一个操作 *value* 用于得到流当前状态所能显示的值(假设值的类型为 A)，另一个操作 *next* 则让流进行下一个状态：$\langle value, next\rangle: X \to A\times X$。

(2)数组：对于一个由有限数组 A^* 和无限数组 A^N 所构成的数组 $A^\infty = A^* \cup A^N$，可以通过两个操作来观察数组中的元素，其中操作 $head: A^\infty \to 1+A$ 用于获取数组的头部元素，操作 $tail: A^\infty \to 1+A^\infty$ 用于返回数组中除头元素外的其他元素。若数组为空，则函数返回 $\perp \in 1$ 作为结果：

$\langle head, tail\rangle: A^\infty \to A\times A^\infty + 1$

(3)标签二叉树 btree(A)(A 为标签元素的类型)：对标签二叉树 btree(A)的观察可以描述如下：

$leaf$: btree$(A) \to A+1$

$left$: btree$(A) \to$ btree$(A)+1$

$right$: btree$(A) \to$ btree$(A)+1$

其中，操作 *leaf* 给出根节点上的标签，若二叉树为空，则均返回 $* \in 1$。*left* 和 *right* 分别返回根节点下的左右子树，若左右子树为空，则返回 $* \in 1$。

类似地，堆栈和队列等也可以看成是共归纳数据类型。

从上述的例子可看出，共归纳数据类型是由某个数据类型集合及其上的一组析构(Destructor)操作所构成，其中析构操作给出了对数据类型的观察结果。对于给定的共归纳数据类型 X，其上的析构操作 σ 通常具有 $\sigma: X \to A_1 \times A_2 \times \cdots \times A_n$ 的形式，其中 X 表示状态空间，A_i 为某个已知的数据类型，表示对应的观察结果。

作为代数的范畴对偶概念，共代数本质上也是集合及其上满足一定性质的操作。但不同的是，共代数在数据类型 X 上的操作具有 $\beta: X \to FX$ 的形式。在计算机科学中，共代数常被看成是具有内部状态的系统，X 是系统中所有可能状态的集合，基调 β 是对系统的一种观察，且其共域(Codomain)通常是结构化的。显然，共归纳数据类型可以进一步抽象为共代数。

共代数的范畴论定义与代数类似。

定义 2　给定集合范畴 *Set* 和其上的自函子 $F: Set \to Set$，函子 F 上的共代数定义为一个二元组$(X, \beta_X: X \to F(X))$，其中 X 是 *Set* 中的对象，称为该 F-共代数的载体，$\beta_X: X \to F(X)$ 是 *Set* 中的射，称为该 F-共代数的变迁射或基调。F-共代数$(X, \beta_X: X \to F(X))$和$(Y, \beta_Y: Y \to F(Y))$两者间的同态射 $f: (X, \beta_X) \to (Y, \beta_Y)$是 *Set* 中的射 $f: X \to Y$，且满足图 2 所示的图表交换。

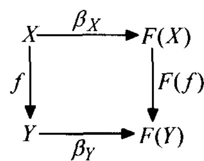

图 2　共代数同态射

例 7　上述的流、数组和二叉树等共归纳数据类型都可以表示为共代数：

(1)流可以表示为函子 $S_A(X) = A\times X$ 下的共代数(X, $\beta_X = \langle value, next'\rangle: X \to A\times X$)，$X$ 是流中所有可能状态的集合。

(2)数组可以表示为函子 $L_A(X) = A\times X+1$ 下的共代数$(X, \beta_X = \langle head, tail\rangle: X \to A\times X+1)$。

(3)二叉树可以表示为函子 $B_A(X) = A\times X\times X+1$ 下的共代数$(X, \beta_X = \langle leaf, left, right\rangle: X \to A\times X\times X+1)$。

共归纳数据类型不仅可以抽象地描述为共代数，而且是对应共代数函子上的最大固定点，即为该函子的一个终结共代数，记为$(\nu F, out_F: \nu F \to F\nu F)$。该终结共代数实际上给出了共归纳类型的定义。

例如，流可以看成是函子 $S_A(X) = A\times X$ 下的一个终结共代数$(A^N, \langle value, next\rangle: A^N \to A\times A^N)$，$A^N$ 是由 A 中元素所构成的无限集合。数组可以看成是函子 $L_A(X) = A\times X+1$ 的一个终结共代数$(A^\infty, \langle head, tail\rangle: A^\infty \to A\times A^\infty + 1)$。二叉树可以看成是函子 $B_A(X) = A\times X\times X+1$ 的一个终结共代数(btree(A), $\langle leaf, left, right\rangle$: btree$(A) \to A\times$ btree$(A)\times$ btree$(A)+1$)，其中 btree(A)是由 A 中的元素所构成的有限及无限二叉树结构。

以数组为例，当对任意的一个数组 $\delta \in A^\infty$ 进行观察时，可以分别应用 $head(\delta)$和 $tail(\delta)$得到该数组的第一个元素和剩余的其他元素。若数组的尾部不为空，那么重复地应用 $head(tail(\cdots \mathrm{tail}(\delta)))$就可以最终得到该数组中剩余的各个元素。利用 *next* 可以将 *head* 和 *tail* 操作合并在一起，即利用 *next* 操作对数组进行观察，若数组不为空，则返回该数组的头部元素 *head* 及剩余部分 *tail*；若数组为空，则返回 $\perp \in 1$。

例 8　数组$(A^\infty, next = \langle head, tail\rangle: A^\infty \to A\times A^\infty + 1)$是共代数函子 $F(X) = A\times X+1$ 下的终结共代数，使得对于任意的共代数$(X, c: X \to A\times X+1)$，均存在一个唯一的同态射 $beh_c: X \to A^\infty$，且满足以下的条件：

(1)$c(x) = 1 \Rightarrow next(beh_c(x)) = 1$；

(2)$c(x) = (a, x') \Rightarrow next(beh_c(x)) = (a, beh_c(x'))$。

即得到图 3 所示的图表交换。

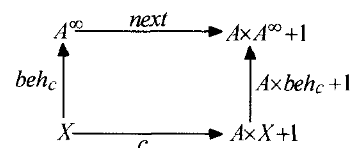

图 3　数组的终结共代数及共归纳扩展射

beh_c 称为由基调 c 所确定的共归纳扩展射，给出了集合 X 中的元素 x 到其行为语义的映射关系。在函数式程序语言中，beh_c 被称为 unfold 操作或 atamorphism，它给出了共归纳数据上的一个共递归(Corecursion)操作。

由 *beh* 的存在性和唯一性可以分别得到共归纳定义和证明原则。共归纳定义原则给出了如何定义共代数到其终结共代数的同态射。例如，要定义一个函数 $beh_c: X \to A^\infty$，只需要在 X 上构造相应的操作 c，使得(X, c)成为一个 $F(X) = A\times X+1$ 共代数即可。而共归纳证明原则说明了如何证明两个射是相等的。例如，要证明 $f, g: X \to A^\infty$ 是相等的，只需要证明它们都是同一个共代数$(X, c: X \to F(X))$到其终结共代数的同态射即可，即证明它们均等于 beh_c。

例 9　要定义函数 $odd: A^\infty \to A^\infty$，它取数组中奇数位上的元素构成一个新的数组，只需要定义 A^∞ 上的操作$\langle head$,

$next \circ next\rangle$，使得满足以下条件：

(1)$head(odd(x))=head(x)$；

(2)$next(odd(x))=odd(next(next(x)))$。

上述说明 $odd:(A^{\infty},\langle head, next \circ next\rangle)\rightarrow(A^{\infty},\langle head, next\rangle)$是一个共代数同态射，且为一个 unfold 操作。

例 10　要定义函数 $merge:A^{\infty}\times A^{\infty}\rightarrow A^{\infty}$，它依次交换将两个数组中的头元素取出，并构成一个新的数组，即 $merge(a_1:x_1,x_2)=a_1:merge(x_2,x_1)$，那么只需要定义 $A^{\infty}\times A^{\infty}$ 上的操作$\langle hd,tl\rangle$，使得满足以下条件：

(1)$head(merge(x_1,x_2))=hd(x_1)$；

(2)$tail(merge(x_1,x_2))=merge(x_2,tl(x_1))$。

显然，$merge:(A^{\infty}\times A^{\infty},\langle hd,tl\rangle)\rightarrow(A^{\infty},\langle head,tail\rangle)$，是一个共代数同态射，且为一个 unfold 操作。

对终结共代数来说，其上的转换结构为同构射。例如，数组$(A^{\infty},next:A^{\infty}\rightarrow A\times A^{\infty}+1)$中的 $next$ 为一个同构射，其逆射 $next^{-1}:A\times A^{\infty}+1\rightarrow A^{\infty}$ 将 1 映射为空数组，将元素对(a,δ)映射为数组 $a\cdot\delta$，即可看成是在数组 δ 的前面插入元素 a。

4　归纳与共归纳数据类型的融合

对数据类型来说，有时候既包含可由代数方法进行描述的静态语法结构，也包含可由共代数方法进行描述的动态行为结构。由于代数和共代数本质上都是集合及其上满足一定性质的操作，从范畴论的角度来看，两者可以构成对偶概念。代数$(X,\alpha_X:T(X)\rightarrow X)$和共代数$(X,\beta_X:X\rightarrow F(X))$都可以看作是 *Set* 上满足一定条件的自函子，在合适的条件下 T 与 F 之间可以构成组合函子，并通过称为分配律(Distributive Laws)的自然转换有机地融合在一起，这为探讨数据类型的语法构造与动态行为关系提供了一种可行的途径。

定义 3　给定 *Set* 上的自函子 T 和 F，一个$\langle T,F\rangle$-结构为一个三元组(X,α_X,β_X)，其中 X 为状态集，$\alpha_X:T(X)\rightarrow X$ 和 $\beta_X:X\rightarrow F(X)$分别为同一载体集 X 上的 T-代数结构和 F-共代数结构。任意两个$\langle T,F\rangle$-结构(X,α_X,β_X)和(Y,α_Y,β_Y)之间的同态射是一个函数 $f:X\rightarrow Y$，且满足：$f\circ\alpha_X=\alpha_Y\circ T(f)$和 $\beta_Y\circ f=F(f)\circ\beta_X$，如图 4 所示。

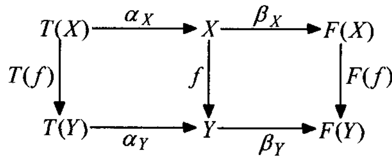

图 4　$\langle T,F\rangle$-结构及同态射

$f:X\rightarrow Y$ 可看成是同时为 T-代数和 F-共代数之间的同态射。实际上，这里的$\langle T,F\rangle$-结构就是 D. Turi 等提出的双代数(Bialgebras)[17]。

$\langle T,F\rangle$-双代数给出了同一个基集上的代数操作和共代数行为变迁结构，但没有说明两者之间的变换关系，因此无法直接用于描述抽象数据类型。下面进一步给出 λ-双代数的定义。

定义 4　设 T 和 F 分别为 *Set* 上的自函子，则 T 对 F 的一个分配律为自然转换 $\lambda:TF\Rightarrow FT$。双代数上的分配律 λ，简称为 λ-双代数，是一个$\langle T,F\rangle$-双代数(X,α_X,β_X)，且满足图 5 所示的图表交换。

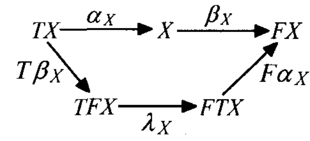

图 5　λ 分配律

根据上述的图表交换，λ-双代数通过分配律将双代数结构中的代数运算和共代数行为变迁有机地融合在一起。简单地说，一方面代数上的构造操作是具有良行为的操作，另一方面共代数上的行为变迁在这些构造操作下保持。因此，利用 λ-双代数可以很好地刻画抽象数据类型上的语法构造和动态行为关系，而且分配律可以让我们方便地进行函子化的提升，或者构造初始和终结双代数。

每一个分配律 $\lambda:TF\Rightarrow FT$ 定义了一个自函子 T_λ，将 T 提升到 F-共代数上，即将一个共代数(X,β_X)映射为$(TX,\lambda_X\circ T\beta_X:TX\rightarrow F(TX))$，如下所示：

$$\frac{X\xrightarrow{\beta_X}F(X)}{T(X)\xrightarrow{T\beta_X}TF(X)\xrightarrow{\lambda_X}FT(X)}$$

而 T_λ 可看成是对代数函子 T 的一种 λ 提升[18]：将 X 提升为 $T_\lambda X=TX$，将 β_X 提升为 $T_\lambda\beta_X=\lambda_X$。$T\beta_X\circ$项 TX 中的元素可看成是以参数 X 中的元素为变量，以 T 中的方法为构造操作所得到的代数项，且 X 具有 B-共代数结构。而分配律 λ_X 说明了在给定参数 X 的共代数行为变迁结构的前提下如何对项 TX 中的元素指派相应的共代数行为变迁结构。

在合适的条件下，若共代数函子 F 上的互模拟在代数函子 T 中是同余关系，那么 λ 就满足复合性，可用于研究数据类型的语法构造与动态行为间的关系。

例 11　上述的数组就是一个典型的 λ-双代数：

$(A^{\infty},[empty,cons]:1+A\times A^{\infty}\rightarrow A^{\infty},next:A^{\infty}\rightarrow A\times A^{\infty}+1)$

分配律 $\lambda:TF\Rightarrow FT$ 自然地描述了数组 A^{∞} 的语法构造与动态行为之间的关系。例如，图 6 给出了对数组[2,1,0]的构造和观察之间的变换关系。

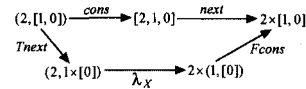

图 6　数组上的 λ 分配律例子

假设 δ 不为空，且 $\delta=a'\cdot\delta'(\delta'\in A^{\infty})$，则对于 $\delta\in A^{\infty}$，利用 λ 分配律可分别得到：

(1)$(a,\delta)\xrightarrow{cons}a\cdot\delta\xrightarrow{next}a\times\delta$；

(2)$(a,\delta)\xrightarrow{Tnext}(a,a'\times\delta')\xrightarrow{\lambda}a\times(a',\delta')\xrightarrow{Fcons}a\times\delta$。

上面的式(1)和式(2)是相等的，这说明要描述在数组 δ 中插入一个元素 a 后的观察效果，可采取以下两种方式：

(1)先利用 *cons* 将一个元素 a 插入到现有的数组 δ 中，然后再通过 *next* 对其进行观察并得到结果 $a\times\delta$；

(2)或者先在保持代数结构的前提下利用 *next* 对 δ 进行观察得到$(a,a'\times\delta')$，接着通过 λ 分配律将结果$(a,a'\times\delta')$转换成 $a\times(a',\delta')$，最后在保持共代数结构的前提下再通过 *cons* 操作得到 $a\times\delta$。

这两种方式所得到的结果都是一样的，即先构造后观察与先观察后转换再构造的结果是相同的。而这两种方式与数组中所包含的元素类型无关，即体现了 λ 的自然性特点。

5 相关研究

共归纳数据类型近十几年来才逐渐引起计算机科学工作者的广泛关注与研究。T. Hagino 最早在其文献[19]中针对范畴编程语言提出了范畴数据类型(Categorical Data Types),并利用所谓的 dialgebras 结构探讨了归纳与共归纳数据类型之间的关系。随后,J. Greiner、R. Hinze、R. Harper 等对程序语言中的共归纳数据类型进行了深入的研究[5,8,12],其主要思路是将归纳数据类型看作初始代数,将共归纳数据类型看作终结共代数,并将归纳原理和共归纳原理引入到程序语言中。E. Giménez[20] 和 J. Leitner[21] 等分别对共归纳数据类型在理论证明工具 Coq 和 Isabelle/HOL 中的应用进行了研究。E. Poll 在文献[22]利用子类型(Subtyping)和继承关系对 Hagino 的范畴论数据类型进行了扩展,并通过共代数与代数之间的对偶性研究了归纳与共归纳数据类型中的子类型和继承关系。R. B. Kieburtz 以流为例,从共代数和 Comonads 的角度对 Haskell 中的共归纳数据类型进行了研究[7]。

从现有的研究来看,在程序语言中的归纳数据类型的基础上逐渐融入共归纳数据类型已经成为当前一些程序语言的最新发展趋势。例如,L. S. Borbosa[23] 和 V. Vene[16] 分别对 CHARITY 和 Haskell 中的共归纳数据类型进行了研究。而 M. Erwig 和 P. Nogueira 等则探讨了如何利用双代数结构研究归纳与共归纳数据类型之间的融合,并分别分析了相应的计算定律[24]及其在多态编程中的应用[25],但他们没有说明如何利用 λ-双代数研究数据类型上的语法构造和动态行为。

相对于上述的研究,本文主要是在范畴论的基础上利用代数和共代数分别研究了归纳和共归纳数据类型,并初步探讨如何通过 λ-双代数和合适的分配律对归纳和共归纳数据类型进行融合,其主要目的是从范畴论的角度探讨数据类型的语法构造与行为之间的关系。

结束语 将共归纳数据类型引入到程序语言中的重要意义不仅在于它为程序语言中数据类型的动态行为提供了一种独特的描述方式和数学理论框架,更重要的是可以将互模拟、终结共代数和共归纳原理等共代数理论引入到程序语言中,从而大大增强数据类型对动态行为的描述与验证能力。

在下一步工作中,将继续对归纳和共归纳数据类型之间的对偶性及融合进行深入的探讨和分析。

参考文献

[1] Bird R. Introduction to Functional Programming using Haskell (2nd Edition)[M]. Prentice-Hall, UK, 1998

[2] Meijer E, Fokkinga M, Paterson R. Functional Program- ming with Bananas, Lenses, Envelopes and Barbed Wire[M]. Hughes J, ed. Functional Programming Languages and Compu-ter Architecture, number 523 in Lect. Notes Comp. Sci., Berlin: Springer, 1991: 215-240

[3] Bird R S, Moor O D. Algebra of Programming [M]. Prentice Hall, UK, 1997

[4] Gibbons J. Lecture Notes on Algebraic and Coalgebraic Methods for Calculating Functional Programs[M]. Summer School on Algebraic and Coalgebraic Methods in the Mathematics of Program Construction, Oxford, UK, 2000

[5] Greiner J. Programming with Inductive and Co-inductive Types [R]. CMU-CS-92-109. School of Comp. Sci., Carnegie-Mellon Univ., Pittsburgh, 1992

[6] Hensel U, Jacobs B. Coalgebraic Theories of Sequences in PVS [J]. Journal of Logic and Computation, 1999

[7] Kieburtz R B. Codata and Comonads in Haskell[Z]. unpublished manuscript, 1999

[8] Hinze R. Reasoning about Codata [J]. Lecture Notes in Computer Science, 2010, 6299: 42-93

[9] Jacobs B. Introduction to Coalgebra: Towards Mathe matics of States and Observations. Book Draft[OL]. http://www.cs.ru.nl/~bart, 2005

[10] Hutton G. Fold and Unfold for Program Semantics[C]// Proc. 3rd. ACM SIGPLAN International Conference on Functional Programming. ACM, 1998

[11] Gibbons J, Jones G. The Under-Appreciated Unfold[C]// Proc. 3rd. ACM SIGPLAN International Conference on Functional Programming. ACM, 1998

[12] Harper R. Practical Foundations fro Programming Languages [Z]. Carnegie Mellon University, Draft 2010

[13] Vos T. Program Construction and Generation Based on Recursive Types[D]. Univ. of Utrecht, 1995

[14] Vene V, Uustalu T. Functional Programming with Apomorphisms(Corecursion)[J]. Proceedings of the Estonian Academy of Science: Physics, Mathematics, 1998, 47(3): 147-161

[15] Uustalu T, Vene V. Primitive(Co)Recursion and Course of-Value(Co) Iteration, Categorically[J]. INFORMATICA, 1999, 10 (1): 5-26

[16] Vene V. Categorical Programming with Inductive and Coinductive types[D]. University of Tartu, 2000

[17] Turi D. Functorial Operational Semantics and its Denotational Dual[D]. Free University, Amsterdam, 1996

[18] Bartels F. Generalised Coinduction[J]. Electronic Notes in Theoretical Computer Science, 2001, 44(1): 67-87

[19] Hagino T. A Categorical Programming Language[D]. Univ. Edinburgh, 1987

[20] Giménez E, Castéran P. A Tutorial on [Co-]Inductive Types in Coq[OL]. http://www.labri.fr/perso/casteran/ RecTutor ial.pdf. 1998

[21] Leitner J. Coalgebraic Methods in the Verification of Optimizing Program Transformations using Theorem Provers [D]. Universität Karlsruhe, June 2005

[22] Poll E. Subtyping and Inheritance for Categorical Datatypes[C]// Proc. of Theories of Types and Proofs(TTP)-Kyoto, RIMS Lecture Notes 1023. 1998: 112-125

[23] Barbosa L S. Coalgebraic Structures in Program Construction [C]// Haeusler H, Camarão C, eds. Invited Tutorial at SBLP'02, Proc. 6th Brazilian Symposium on Programming Languages. Rio de Janeiro, 2002

[24] Erwig M. Categorical Programming with Abstract Data Types [C]// 7th Int. Conf. on Algebraic Methodology and Software Technology(AMAST'98). LNCS 1548. 1998: 406-421

[25] Nogueira P, Moreno-Navarro J J. Bialgebra Views: A Way for Polytypic Programming to Cohabit with Data Abstract[C]// WGP '08 Proceedings of the ACM SIGPLAN workshop on Generic Programming. ACM New York, NY, 2008