Comonads理论及其在函数式程序语言Haskell中的应用

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 函数式程序语言Haskell中的Monads理论在描述上下文依赖计算等方面存在一定的不足。作为Monads的范畴论对偶概念,Comonads理论可以有效地提高Haskell对上下文依赖计算的描述能力。首先给出Comonads的范畴论定义和性质,以及Comonads在Haskell的具体实现;接着探讨Comonads的CoKleisli三元组和CoKleisli范畴,通过实例说明如何将其应用于上下文依赖计算的描述和推理中;最后进一步研究Comonads与Monads之间的分配律,指出如何通过分配律将效果计算与上下文依赖计算有机地融合起来。

关键词 Comonads,函数式程序语言,Haskell,上下文依赖计算,范畴论
中图法分类号 TP301.2 **文献标识码** A

Comonad Theory and its Applications in Functional Programming Language Haskell

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 Monad theory in functional programming language Haskell has some disadvantages in describing the context dependent computations. As the categorical dual notion of monads, comonad theory can effectively improve Haskell's description ability of context-dependent computations. Firstly, we gave the categorical definitions and properties of Comonads, as well as their implementations in Haskell. Secondly, we discussed the CoKleisli triple and CoKleisli category, and used some examples to demonstrate how to apply them into the descriptions and reasoning of context dependent computations. Finally, we also discussed the distributive laws between Comonads and Monads, and showed its uses in merging the effectful computations and context dependent computations.

Keywords Comonads, Functional programming, Haskell, Context dependent computations, Category theory

1 引言

作为函数式程序语言Haskell中的一个非常重要的基础理论及核心库,Monads理论[1-5]为effectful函数的计算效果(例如异常、确定性行为、概率计算、非确定性行为、状态转换、连续性等)提供了一种统一的数学表达和推理方式,并使得值和对值的计算能够显式地分离开来,从而提高程序的模块化、灵活性和隔离性。但Monads并不适合描述所有的impure函数,特别是涉及到上下文依赖的计算。某些计算不仅仅是产生了计算效果,而且可能消耗了值以外的因素,比如值的上下文。这一类的计算通常被称为上下文依赖计算。例如,在数据流计算中,一个流的输出值可能不仅取决于当前的输入值,还依赖于过去或未来的输入值。也就是说,流的输出值依赖于当前的输入值及其所处的上下文。

作为Monads的范畴论对偶概念,Comonads理论近年来才引起计算机科学工作者的注意,并逐渐成为共代数(Coalgebras)领域和范畴论的一个研究热点。Monads上的自函子可以视为是对值的某种计算,而Comonads上的自函子则可以看作某种上下文环境,这就为描述上下文依赖计算及与效果计算的融合提供了一种新的途径和思路。例如,由Comonads所得到的CoKleisli范畴可以直观地描述上下文依赖计算,其中CoKleisli箭头表示上下文依赖计算函数,而箭头之间的组合关系对应着函数之间的映射关系。因此,将Comonads应用于Haskell不仅能够有效地提高Haskell对各种impure计算的描述能力,并且由Monads和Comonads间的分配律可将效果计算的副作用与上下文依赖计算有机地融合在一起。另外,Comonads是以范畴论为理论基础的,这也为研究各种上下文依赖计算提供了一个统一和抽象的数学理论框架。

近年来,一些学者开始对Comonads理论进行研究,主要侧重于Comonads的范畴论性质,例如Comonads与Monads的对偶性、共独异点结构、Comonads与伴随函子、自由Comonads及其共代数等。而R. B. Kieburtz, J. R. Lewis和T. Uustalu等人则对Comonads在程序设计中的应用进行了

到稿日期:2010-08-23 返修日期:2010-12-27 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。

研究[6-9]。但总的来说，目前对 Comonads 的研究仍处于起步阶段，尚未引起足够的重视。Comonads 要在程序设计语言中推广还面临着许多理论和应用方面的困难，仍需要很多研究工作。我们认为 Comonads 将对程序设计语言中的行为和计算描述产生积极的影响，并且可以促进人们对共递归程序、终结共代数和共代数共簇(Covariety)等的研究。

本文的主要目的是从范畴论的角度探讨 Comonads 及相应的 Cokleisli 三元组和 CoKleisli 范畴性质，并给出 Comonads 在 Haskell 中的详细实现和应用。本文第 2 节给出 Comonads 的范畴论定义和性质；第 3 节给出 Comonads 的 Cokleisli 三元组及其范畴的定义，并对其性质进行分析；第 4 节探讨了 Monads 与 Comonads 之间的分配律和在融合效果计算及上下文依赖计算方面的应用；第 5 节对当前一些相关研究进行分析；最后是总结并给出下一步的工作。

2 Comonads 的范畴论定义

限于篇幅，本文不介绍 Monads 理论和范畴论的相关知识，请另行参考文献[1-4]。

Comonads 可以看成是 Monads 的范畴对偶概念，其定义如下：

定义 1 范畴$\mathbb{C}$上的 Comonad 定义为一个三元组(D,ε,υ)，其中$D:\mathbb{C}\to\mathbb{C}$为$\mathbb{C}$上的自函子，$\varepsilon:D\Rightarrow Id$和$\upsilon:D\Rightarrow D^2$为自然转换，且满足图 1 所示的图表交换。

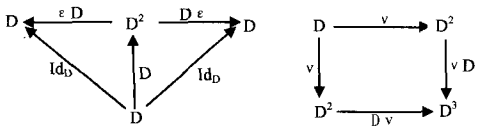


图 1 Comonads 的图表交换

上述的两个定律(1)$\varepsilon D\circ\upsilon=D\varepsilon\circ\upsilon=Id_D$和(2)$D\upsilon\circ\upsilon=\upsilon D\circ\upsilon$也称为 Comonads 的一致性性质(Coherence Properties)。ε和υ分别称为 Comonad (D,ε,υ)的共单元(Counit)和共乘(Comultiplication)，可看成是自函子 D 上的额外结构。$\varepsilon D\circ\upsilon=Id_D$和$D\varepsilon\circ\upsilon=Id_D$分别称为 Comonads 的左共单元定律和右共单元定律。

一个 Comonad 可以看成是范畴$\mathbb{C}^{\mathbb{C}}$上的一个共独异点(Comonoid)。

利用 Haskell 给出 Comonads 的具体实现如下：

```
class Comonad d where
counit::da→a
cobind::(da→b)→da→db
cmap::Comonad d=>(a→b)→da→db
cmap::f=cobind(f. counit)
```

cmap 将一般的纯函数 $f:a\to b$ 提升为 $df:da\to db$，其提升过程通过应用 counit 和 cobind 来实现：$f.\ counit=f:a\to b.\ counit:da\to a=da\to b$。

例 1 假设$\mathbb{C}$为某个 Cartesian 封闭范畴(例如 Set)，下面给出一些典型的 Comonads 例子：

(1)积 Comonad(也称为流 Comonad)

$DX=_{df}X\times E$，其中 E 为 Set 上的某个固定对象。

共单元$\varepsilon_X=_{df}X\times E\xrightarrow{\pi_1}X$

共乘$\upsilon_X=_{df}X\times E\xrightarrow{\langle Id,\pi_2\rangle}(X\times E)\times E$

显然，积 Comonad 与 Monads 中的指数 monad $TX=X^E$一一对应，即两者是等价的。

(2)指数 Comonad

$DX=_{df}S\Rightarrow X$，其中(S,e,m)为范畴$\mathbb{C}$上的一个独异点，$e:1\to S$为单位元，$m:S\times S\to S$为满足组合律二元操作符，则：

共单元$\varepsilon_X=_{df}S\Rightarrow X\xrightarrow{ur^{-1}}(S\Rightarrow X)\times 1$

$\xrightarrow{Id\times e}(S\Rightarrow X)\times S\xrightarrow{ev}X$

共乘$\upsilon_X=_{df}\Lambda(\Lambda(\upsilon_X'))$

其中，

$\upsilon_X'=_{df}(S\Rightarrow X)\times S\times S\xrightarrow{a}(S\Rightarrow X)\times(S\times S)$

$\xrightarrow{Id\times m}(S\Rightarrow X)\times S\xrightarrow{ev}X$

例 2 例如独异点$(S,e,m)=_{df}(Nat,0,+)$，则指数 Comonads 表示自然数与集合 X 之间的一个函数关系 $f:Nat\to X$。例如对于$DX=(x_0,x_1,\cdots,x_n)$，有：

$\varepsilon_X((x_0,x_1,\cdots,x_n))=x_0$

$\upsilon_X'((x_0,x_1,\cdots,x_n),i,j)=\upsilon_X'((x_0,x_1,\cdots,x_n),(i+j))=f(i+j)$，其中$i,j\in Nat$且$0\leqslant i+j\leqslant n$。

(3)共状态 Comonad

$DX=(S\Rightarrow X)\times S$，其中 S 为范畴$\mathbb{C}$上的一个对象。

$\varepsilon_X=_{df}(S\Rightarrow X)\times S\xrightarrow{ev}X$

$\upsilon_X=_{df}(S\Rightarrow X)\times S\xrightarrow{coev\times Id}(S\Rightarrow((S\Rightarrow X)\times S))\times S$

例 3 流函数$StrFun=(Nat\Rightarrow X)\times Nat$可看作共状态 Comonad 的一个实例，即 S 为自然数 Nat。对于$f:Nat\to X$和$StrX=(x_0,x_1,\cdots,x_n)$，有：

共单元$\varepsilon_X((x_0,x_1,\cdots,x_n),i)=f(i)$，即$(f,i)\mapsto f(i)$

共乘$\upsilon_X((x_0,x_1,\cdots,x_n),i)=(\lambda j.\ (f,j),i)$，即：

$(f,i)\mapsto(\lambda j.\ (f,j),i)$，其中$i,j\in Nat$且$i,j\leqslant n$。

下面分别给出上述各个 Comonad 在 Haskell 中的具体实现：

(1)积 Comonad $DX=A\times X$

```
data Stream a=a:< Stream a
instance Comonad Stream where
counit(a:< _)=a
   cobind k d@( _:< as)=kd:< cobind k as
nextS::Stream a→a
nextS::( _:<(a':< _))=a'
```

(2)指数 Comonad $DX=S\Rightarrow X$

```
data Prod s x=x:& s
instance Comonad(Prod s) where
counit(x:& _)=x
cobind k d@(_:& s)=kd:& s
askP::Prod s x → x
askP(_:& s)=s
localP::(s→s)→Prod s x→Prod s x
localP::g(x:& s)=(x:& g s)
```

(3)共状态 Comonad(设 S 为整数类型 Int)

```
data StrFun x=(Int→x):@ Int
instance Comonad StrFun where
counit(f:@i)=f i
cobind k(f:@i)=(\ i' →k(f:@i')):@i
```

由 Comonad 可定义两个 Comonads 之间的态射。

定义 2 范畴$\mathbb{C}$上的两个 Comonads $(D_1,\varepsilon_1,\upsilon_1)$和$(D_2,\varepsilon_2,\upsilon_2)$之间的射可以看成是两个函子之间的一个自然转换 $\rho:D_1\Rightarrow D_2$，并且满足图 2 所示的图表交换。

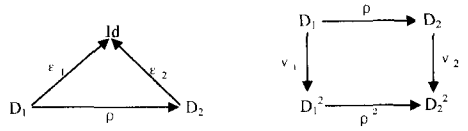


图 2 Comonads 态射的图表交换

并不是所有的 Comonads 都存在态射。直观来讲，D_1 和 D_2 之间存在态射 $\rho:D_1\to D_2$ 的前提条件是 D_1 的结构要能够嵌入到 D_2 中。若 ρ 存在，则可以利用 ρ 将具有 D_1 的计算结构转换为具有 D_2 的计算结构，同时保持原来的状态空间及计算行为。

3 CoKleisli 三元组及其范畴

下面给出 Comonads 的另一种等价定义，称为 CoKleisli 三元组。

定义 3 范畴$\mathbb{C}$上的一个 CoKleisli 三元组表示为$(D,\varepsilon,-^D)$，其中 $D:|\mathbb{C}|\to|\mathbb{C}|$，使得对于任意的对象 $X\in|\mathbb{C}|$有 $\varepsilon_X:DX\to X$，对于函数 $f:DX\to Y$ 有 $f^D:DX\to DY$，并且满足以下等式：

(1)$\varepsilon_X^D=Id_{DX}$；

(2)对于函数 $f:DX\to Y$，有等式 $f^D;\varepsilon_Y=f$，即 $DX\xrightarrow{f^D}DY\xrightarrow{\varepsilon_Y}Y=f$；

(3)对于 $f:DX\to Y$ 和 $g:DY\to Z$，有复合关系 $f^D;g^D=(f^D;g)^D$。

定理 1 CoKleisli 三元组和 Comonads 之间存在一一对应关系。

证明：给定一个 CoKleisli 三元组$(D,\varepsilon,-^D)$，则对应的 Comonad 为(D,ε,υ)，其中 D 是函数 D 到自函子的扩展，即对于 $f:X\to Y$ 有 $Df=(\varepsilon_X;f)^D$，且 $\upsilon_X=Id_{IX}^D$。相反地，给定一个 Comonad (D,ε,υ)，对应的 CoKleisli 三元组为$(D,\varepsilon,-^D)$，其中 D 是将函子限制为对象间的函数关系，并且对于 $f:DX\to Y$，有 $f^D:DX\to Y$。

函数$-^D$可以看成是一种提升，例如将 $f:DX\to X$ 提升为 $Id_{DX}=f^D:DX\to DX$，把 $f:DX\to Y$ 提升为 $f^D:DX\to DY$。对于 $X\in\mathbb{C}$，若 ε_X 为满的，则称该 CoKleisli 三元组满足满要求(Epi Requirement)。对于 $f:DX\to Y$ 和 $g:DY\to Z$ 来说，其 CoKleisli 复合定义为：

$$g\circ^D f:DX\xrightarrow{\upsilon_X}DDX\xrightarrow{Df}DY\xrightarrow{g}Z$$

因此，$Df\circ\upsilon$ 有时也记为 $f^D:DX\to DY$，称之为 f 的共扩展(Coextension)。

直观上来看，$\varepsilon_X:DX\to X$ 表示将处于某个上下文环境 DX 中的值 X 取出，而 f^D 表示将上下文依赖与值之间的计算 $f:DX\to Y$ 扩展到上下文依赖计之间的函数关系 $f^D:DX\to DY$。事实上，对 X 的计算与对 ε_X 的值的计算是一样的。

为了验证 CoKleisli 三元组中的定理，由 CoKleisli 三元组可以进一步定义一个 CoKleisli 范畴 CoKl($\mathbb{D}$)，其中的态射对应着上下文依赖计算，态射组合对应着计算间的映射关系。因此，CoKl($\mathbb{D}$)也可看成是由上下文依赖计算及它们之间的组合关系所构成的范畴。

定义 4 给定范畴$\mathbb{C}$上的一个 CoKleisli 三元组$(D,\varepsilon,-^D)$，对应的 CoKleisli 范畴 CoKl($\mathbb{D}$)定义为：

(1)CoKl($\mathbb{D}$) 中的对象就是$\mathbb{C}$的对象；

(2)CoKl($\mathbb{D}$) 中从 X 到 Y 的态射 CoKl($\mathbb{D}$)(X,Y)就是$\mathbb{C}(DX,Y)$；

(3)CoKl($\mathbb{D}$)在 X 上的标识函子为 $\varepsilon_X:DX\to X$；

(4)$f\in$CoKl($\mathbb{D}$)(X,Y)与 $g\in$CoKl($\mathbb{D}$)(Y,Z)之间的组合是 $g\circ f^D:DX\to Z$，即：

$$g\circ f^D:DX\xrightarrow{f^D}DY\xrightarrow{g}Z$$

CoKl($\mathbb{D}$)中的组合可利用 f^D 获得直观和简单的解释：

$$\frac{x:DX\vdash f(x):Y\quad y:DY\vdash g(y):Z}{x:DX\xrightarrow{f^D;g}(let\ y\leftarrow f(x)\ ing(y)):Z}$$

利用共单元 ε 可以将 pure 函数看作是上下文依赖计算，而共乘 υ 使 CoKleisli 范畴中上下文依赖计算可能组合。

在 Haskell 中，Cokleisli 箭头可以看作是 Arrow 类的一个实例，如下所示：

```
newtype CoKleisli d a b=Cokleisli(da→b)
instance Comonad d=> Arrow(CoKleisli d) where
  pure f=CoKleisli(f . counit)
  CoKleisli k>>>CoKleisli l=CoKleisli(l. cobind k)
    First(CoKleisli k)=CoKleisli(pair(k . cmap fst)(snd . counit))
```

对于由$\mathbb{C}$上的对象标识函子 Id 所构成的范畴，容易证明存在一个包含函子 J 将其映射到 CoKl($\mathbb{D}$)范畴上，且 J 是定义在映射上：对于函数 $f:X\to Y$，有：

$$J(f)=_{df}DX\xrightarrow{\varepsilon_X}X\xrightarrow{f}Y=DX\xrightarrow{Df}DY\xrightarrow{\varepsilon_Y}Y$$

则函子 J 存在一个左伴随 $U:$CoKl($\mathbb{D}$)$\to\mathbb{C}$，使得 $UX=DX$，且对于 $k:DX\to Y$，有 $Uk=DX\xrightarrow{k^D}DY$。

将 CoKleisli 范畴作为 impure 计算的范畴的直观含义是：将 $f:X\to Y$ 看成是 pure 函数的范畴，而 D 表示上下文，DX 表示将类型为 X 的值放入到一个上下文环境中，通过对 D 进行实例化就可以定义各种不同的上下文环境(例如数据流、标签树)。函数 $\varepsilon_X=DX\to X$ 表示将上下文环境 DX 中的值 X 取出，变成不依赖于上下文的标识函数。$J(f):DX\to Y$ 表示由一个一般的 pure 函数 $f:X\to Y$ 转换为上下文依赖计算的函数。共扩展函数 $k^D:DX\to DY$ 表示将由一个上下文依赖计算 $k:DX\to Y$ 扩展为输出值也包含上下文环境的计算。因此，对象为 $f:DX\to Y$ 的 CoKl($\mathbb{D}$)范畴就可以直观地表示上下文依赖计算及其组合关系。

例 4 数据流 $StrA=\mu X.A\times X$ 可以采用有序集合的形式表示为 $StrA=Nat\Rightarrow A$，那么数据流之间的函数关系 $f:StrA\to StrB$ 就可等价地表示为：

$$f:(Nat\Rightarrow A)\to(Nat\Rightarrow B)\cong(Nat\Rightarrow A)\times Nat\to B$$

容易证明 $D=(Nat\Rightarrow-)\times Nat$ 为一个 Comonad

其中：

$\varepsilon_X=_{df}(Nat\Rightarrow X)\times Nat\to A$，即$(f,n)\vdash\to fn$。

$\upsilon_X=(Nat\Rightarrow X)\times Nat\to(Nat\Rightarrow((Nat\Rightarrow X)\times Nat))\times Nat$，即$(f,n)\vdash\to(\lambda m.(f,m),n)$。

则流函数可以表示为 CoKleisli 范畴中的射。在 Haskell 中，该同构射的具体实现描述如下：

```
data Stream a=a:< Stream a
--str2fun 函数用于将流映射为自然数与集合之间的函数关系
```

str2fun::Stream a→Int→a

str2fun(a:<as)(i+1)=str2fun as i

--fun2str 函数用于将自然数与集合之间的函数关系映射为流

fun2str::(Int→a)→Stream a

fun2str f=fun2str' f 0

fun2str' f i=f I:< fun2str' f(i +1)

例 5　假设有集合$\{a_0,a_1,\cdots,a_n\}$和自然数 0..n,以及流$as=(a_0,a_1,\cdots,a_n)$,则 str2fun 函数作用于流 $as=(a_0,a_1,\cdots,a_n)$后可得到一个集合$\{a_0,a_1,\cdots,a_n\}$和自然数 0..n 间的映射关系。而 fun2str 则将集合$\{a_0,a_1,\cdots,a_n\}$和自然数 0..n 间的函数关系变成一个流$as=(a_0,a_1,\cdots,a_n)$。

例 6　一个表示确定性状态转换系统的函子 $FX=1+Act\times X$ 的行为由包含有限和无限的行为变迁序列构成:

$$d_x=x\xrightarrow{a_1}x_1\xrightarrow{a_2}x_2\xrightarrow{a_3}\cdots$$

式中,$x\in X$,动作 $a_i\in Act$,$d_x\in DX$。则 ε_X 表示取出该抽象行为序列的根元素 x,而共乘 $\upsilon_X:DX\to D^2X$ 将 d_x 映射为另一个保持相同转换结构的抽象行为,并且其中的每一个状态 x_i 都被替换成该状态所对应的全局抽象行为序列 d_{x_i}:

$$\varepsilon_X(x\xrightarrow{a_1}x_1\xrightarrow{a_2}x_2\xrightarrow{a_3}\cdots)=x,$$

$$\upsilon_X(x\xrightarrow{a_1}x_1\xrightarrow{a_2}x_2\xrightarrow{a_3}\cdots)=dx\xrightarrow{a_1}dx_1\xrightarrow{a_2}dx_2\xrightarrow{a_3}\cdots$$

4　Monads 与 Comonads 间的融合

由于 Monads 与 Comonads 都可以看作是范畴$\mathbb{C}$上满足额外结构的自函子,并且在合适的条件下 T 与 D 之间可以构成组合函子,因此可以通过 T 和 D 之间的自然转换(称为分配律)将两者有机地融合在一起。

定义 5　给定范畴$\mathbb{C}$上的一个 Monad (T,η,μ)和一个 Comonad (D,ε,υ),称自然转换 $\lambda:DT\Rightarrow TD$ 为 D 对 T 的分配律,当且仅当满足图 3 所示的图表交换。

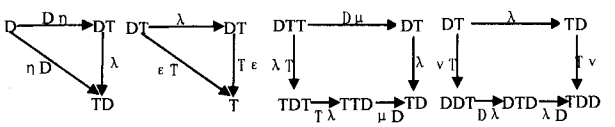


图 3　Comonads 对 Monads 的分配律

由 Comonads 对 Monads 的分配律 $\lambda:DT\Rightarrow TD$ 可以确定一个范畴$\mathbb{C}_{D,T}$,其具体定义如下:

定义 6　由 Comonad (D,ε,υ)对 Monad (T,η,μ)的分配律 $\lambda:DT\Rightarrow TD$ 可以确定一个范畴$\mathbb{C}_{D,T}$(称为 cmKleisli 范畴),其中

(1)$\mathbb{C}_{D,T}$中对象与$\mathbb{C}$中的对象相同,即$|\mathbb{C}_{D,T}|=|\mathbb{C}|$;

(2)$\mathbb{C}_{D,T}$中的对象射为$\mathbb{C}$中的 $DX\to TY$,即$\mathbb{C}_{D,T}(X,Y)=\mathbb{C}(DX,TY)$;

(3)$\mathbb{C}_{D,T}$中对象的标识函子$(Id_{D,T})_X=\eta_X\circ\varepsilon_X$,即$(Id_{D,T})_X=DX\xrightarrow{\varepsilon_X}X\xrightarrow{\eta_X}TX$;

(4)对于对象射 $k:DX\to TY$ 和 $\ell:DY\to TZ$,有 $\ell\circ_{D,T}k=\ell^*\circ\lambda_Y\circ k^D$,即:

$$\ell\circ_{D,T}k=DX\xrightarrow{k^D}DTY\xrightarrow{\lambda_Y}TDY\xrightarrow{\ell^*}TZ$$

式中,ℓ^* 是将 $\ell:DY\to TZ$ 提升为 $\ell^*:TDY\to TZ$。

在 Haskell 中,D 对 T 的分配律的具体实现如下:

```
class(Comonad d,Monad t)=> Dist d t where
  dis::d(t a)→t(d a)
newtype cmKleislid t a b=cmKleisli(da→tb)
instance Dist d t a b=cmKleisli(da→tb)
  pure f=cmKleisli(return. f. counit)
cmKleisli k >>> cmKleisli l=
cmKleisli((>>=l) . dist . cobind k)
  first(cmKleisli k)=cmKleisli( \d →
      mstrength(k(cmap fst d))(snd(counit d))
```

5　相关研究

最早对 Comonads 的研究主要是在范畴论的框架下讨论 Comonads 的各种范畴性质,并未应用于程序语言及计算描述中。后来,S. Brookes 等人提出一种计算 Comonads 用于研究指称语义的内涵语义(Intensional Semantics)[10]。虽然 S. Brookes 的研究并未直接涉及到一般的上下文依赖计算,但内涵语义实际上就是指称语义的 impure instrumentation 的一种形式,同样适合于上下文依赖计算。

在函数式程序语言的应用方面,R. B. Kieburtz 最早提出在函数式编程中利用 Comonads 作为上下文依赖计算的结构化描述工具,并给出一些有趣的例子[6]。随后,J. R. Lewis 将 Comonads 应用于环境传递计算和隐含函数(Implicit Parameters)的研究中[7]。在此基础上,T. Uustalu 进一步利用 Comonads 研究了上下文依赖计算[8],并对一些重要的内容依赖计算进行了分析,指出数据流计算[9](基于 stream 的计算)和属性评价计算[11]均可以采用 Comonads 给出统一和抽象的描述。

在上述研究的基础上,本文深入地分析了 Comonads 的范畴论性质,指出 CoKleisli 三元组和 CoKleisli 范畴在描述上下文依赖计算方面的应用,并利用 Haskell 给出具体的实现。同时探讨了 Comonads 与 Monads 之间的分配律及其在融合效果计算与上下文依赖计算中的应用。

结束语　将 Comonads 理论引入到 Haskell,其重要意义不仅在于它对上下文依赖计算提供了一种新的独特的描述方式和数学理论框架,更重要的是可以将终结共代数、共归纳原理等关键的共代数概念引入到程序设计中,这可以大大地增强程序语言对动态行为及 impure 计算的描述和验证能力。随着共代数研究的深入和进一步完善,在程序语言中融合共代数已经成为未来程序语言发展的一个趋势,而 Comonads 提供了一种将共代数和范畴论融入到程序语言的有效和可行的途径。

在下一步工作中,我们将继续研究共代数中的 Comonads 及其性质,特别是共自由 Comonads 及其共代数范畴、Comonads 与伴随函子及其伴随性质等。

参 考 文 献

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

[2] Meijer E, Fokkinga M, Paterson R. Functional Programming with Bananas [C]// Envelopes L, Wire B, Hughes J, eds. Functional Programming Languages and Computer Architecture, number 523 in Lect. Notes Comp. Sci. Berlin: Springer, 1991: 215-240

(下转第 156 页)

6.3 静态维度

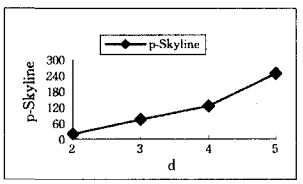
固定数据规模 $N=800$，非确定段长度 $Length=12$，移动对象遍历路径 $Path=3$。

对任意数据点 $\vec{P},\vec{Q}$，当 d 增大时，$\exists$ 两静态维 $i,j(1\leqslant i,j\leqslant k\&\&i\neq j)$ 使 $x_i^P<x_i^Q$ and $x_j^P>x_j^Q$ 成立的概率增大，导致数据点间存在静态支配的概率减少，产生 event 的概率降低，表 1 验证了上述分析，剪枝后的 event 数量与剪枝前相比大大减少，体现了剪枝规则的有效性。

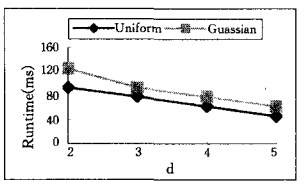
表 1 剪枝前后 event 数量比较

静态维度	event 规模	
	剪枝前	剪枝后
2	2084	39
3	1357	50
4	861	75
5	388	53

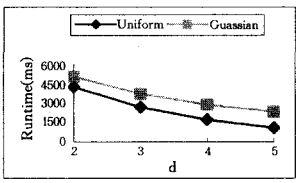
同理，对任意数据点 $\vec{P}$ 来讲，维度 d 的增大导致数据点 $\vec{Q}$ 使 $x_Q^S<x_P^S$ 成立的概率降低，则 $\vec{P}$ 成为 p-Skyline 点的概率增大，p-Skyline 的规模随 d 的增大而不断增大，图 5(a)验证了上述分析。产生 event 的概率降低，算法的平均处理时间减少，图 5(b)、图 5(c)验证了上述分析。



(a) p-Skyline 规模



(b) U-CPSCRN 算法运算时间



(c) U-SPSCRN 算法运算时间

图 5 静态维度对算法的影响

结束语 针对道路网络环境下不确定移动对象的连续概率 Skyline 计算问题进行了研究。首先选择路段模式作为路网的建模方式；然后提出了各数据点间支配概率和 Skyline 概率的表示方式，并定义了两类可能引起 p-Skyline 集合变动的 event 事件，提出了对不确定移动对象进行连续概率 Skyline 计算的动态增量算法框架 U-CPSCRN 和对比的静态算法 U-SPSCRN；最后利用大量的实验分析，从数据规模、路径数量、静态维度等几个方面论证了 U-CPSCRN 算法的有效性。

参 考 文 献

[1] Borzsonyi S, Kossmann D, Stocker K. The Skyline operator[C]// Proceedings of the Int'l. Conf. on Data Engineering. Heidelberg, Germany, 2001: 421-430

[2] Tan K, Eng P, Ooi B. Efficient progressive skyline computation [C]// Proceedings of the Int'l. Conf. on Very Large Data Bases. Roma, Italy, 2001: 301-310

[3] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries[C]// Proceedings of the Int'l. Conf. on Very Large Databases. Hong Kong, China, 2002: 275-286

[4] Papadias D, Tao Y. Progressive skyline computation in database systems[J]. ACM Transactions on Database Systems, 2005, 30(1): 41-82

[5] Huang Zhi-yong, Lu Hua, Ooi B, et al. Continuous skyline queries for moving objects[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(12): 1645-1658

[6] Lee W, Hwang S. Continuous skyline on volatile moving data [C]// Proceedings of the 25th IEEE Int'l. Conf. on Data Engineering. Shanghai, China, 2009: 1568-1575

[7] Pei Jian, Jiang Bin, Lin Xue-ming, et al. Probabilistic skylines on uncertain data[C]// Proceedings of the 33th Int'l. Conf. on Very Large Databases. Trondheim, Norway, 2005: 253-264

[8] 王亚琴. 道路交通流数据挖掘研究[D]. 上海：复旦大学，2007

[9] Huang Xue-gang, Christian S. In-Route skyline querying for location-based services[M]. Web and Wireless Geographical Information Systems, 2005: 120-135

[10] 丁晓锋. 移动计算环境下非确定数据的索引与查询方法研究[D]. 武汉：华中科技大学，2007

(上接第 147 页)

[3] Wadler P. Comprehending Monads [J]. Math. Struct. in Comp. Sci., 1992, 2: 461-493

[4] Wadler P. Monads for Functional Programming[C]// Jeuring J, Meijer E, eds. Advanecd Functional Programming. Springer Lecture Notes of Computer Science(925). 1995

[5] 袁华强，孙永强. 基于 Monad 的纯函数式语言通道系统设计[J]. 计算机科学，2004，31(3)：167-169

[6] Kieburtz R B. Codata and Comonads in Haskell [R]. Unpublished Manuscript, 1999

[7] Lewis J R, Shields M B, Meijer E, et al. Implicit Parameters: Dynamic Scoping with Static Types [C]// Proc. of 27th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (PoPL'00). ACM Press, 2000: 108-118

[8] Uustalu T, Vene V. Comonadic Notions of Computation [J]. Electronic Notes in Theoretical Computer Science, 2008, 203: 263-284

[9] Uustalu T, Vene V. Signals and Comonads [J]. Journal of Universal Computer Science, 2005, 22(7): 1310-1326

[10] Brookes S, Geva S. Computational Comonads and Intensional Semantics [C]// Fourman M P, Johnstone P T, Pitts A M, eds. Applications of Categories in Computer Science. London Math. Society Lecture Note Series 177. Cambridge Univ. Press, 1992: 1-44

[11] Uustalu T, Vene V. Comonadic Functional Attribute Evaluation [C]// van Eekelen M, ed. Trends in Functional Programming 6. Intellect, 2005: 145-162