使用伴随在泛型编程中

Ralf Hinze

牛津大学计算机科学系
Wolfson Building, Parks Road, Oxford, OX1 3QD, 英国
ralf.hinze@cs.ox.ac.uk
http://www.cs.ox.ac.uk/ralf.hinze/

摘要. 伴随是数学中最重要的构造之一。. 这些讲义展示了它们在泛型编程中的高度相关性。 首先，每个基本数据类型——和、积、函数类型、递归类型——都是由一个伴随构造出来的。 伴随的定义特性导致了编程代数的著名定律。 其次，伴随在统一和推广递归方案中起到了重要作用。 我们讨论了多种基本的伴随并展示了它们与编程和程序推理的直接相关性。

1 引言

Haskell程序员已经接受了函子[1]、自然变换[2]、单子[3]、单子函子[4]，也许在较小程度上接受了初始代数[5]和终结余代数[6]。 现在是时候让他们关注伴随了。

伴随的概念由丹尼尔・坎在1958年引入[7]。 简单来说，如果类型为 $\mathsf{L}\,A \to B$ 的箭头与类型为 $A \to \mathsf{R}\,B$的箭头一一对应，并且这种双射在 A和 B上还是自然的，那么函子 L和 R就是伴随的。 伴随在范畴论中被证明是最重要的思想之一，主要是因为它们的普遍性。 许多数学构造都是形成伴随和伴随函子的伴随，正如Mac Lane [8, p.vii]所说：“伴随函子无处不在。”这些讲义的目的是展示伴随的概念对编程也非常相关，特别是对于泛型编程。 这个概念至少有两种不同但相关的方式是相关的。

首先，每个基本数据类型——和、积、函数类型、递归类型——都是由伴随产生的。 伴随的范畴学要素对应于引入和消除规则；伴随的定义特性对应于 β-规则、 η-规则和融合定律，这些规则和定律体现了基本的优化原则。

其次，伴随在统一和泛化递归方案中起着重要作用。 从历史上看，编程代数[9]基于初始代数的理论：程序以折叠的形式表达，并且程序计算基于折叠的通用性质。 简而言之，通用性质

形式化地表明折叠是其定义方程的唯一解。 它暗示了计算规律和优化规律，如融合。 通过对偶原理进一步增强了推理的经济性：初始代数对偶为最终余代数，相应地折叠对偶为展开。 一份价格，两个理论。

然而，闪闪发光的并不都是黄金。 大多数，如果不是全部，程序需要一些调整才能以折叠或展开的形式给出，从而使它们适合进行形式化操作。 有点讽刺的是，这在函数式编程的“Hello, world!”程序中尤其明显：阶乘、斐波那契函数和追加。 例如，追加不具有折叠的形式，因为它需要一个稍后在基本情况中使用的第二个参数。

为了应对这个缺点，在过去的二十年里引入了大量不同的递归方案。 使用伴随的概念，许多这些方案可以统一和泛化。 得到的方案被称为伴随折叠。 标准折叠坚持一个观点，即函数的控制结构始终遵循其输入数据的结构。 伴随折叠放松了这种紧密耦合 - 控制结构通过伴随隐式给出。

从技术上讲，核心思想是通过允许折叠的参数或展开的结果被包装在一个函子应用中来获得灵活性。
在append的情况下，函子本质上是配对。 并非每个函子都是可接受的：为了保持折叠和展开的显著属性，我们要求函子具有右伴随，对偶地，展开需要左伴随。 与折叠类似，伴随折叠然后是其定义方程的唯一解，而且如预期的那样，这对偶到展开。

这些讲义分为两个主要部分。 第一部分（第2节）研究了使用伴随来定义'数据结构'。 它部分基于在春季学校分发的“范畴论入门”[10]。 本节包括一些范畴论的背景材料，旨在使讲义对没有专业知识的读者易于理解。

第二部分（第3节）说明了使用伴随来给出'算法'的精确语义。 它在即将发表的文章“伴随折叠和展开-扩展研究”[11]的基础上进行了大部分编写。 一些材料已被省略，一些新材料已被添加（第3.3.3节和第3.4.2节）；此外，所有的例子都已经重新制作。

这两部分可以相对独立地阅读。 实际上，在第一次阅读时，我建议直接跳到第二部分，尽管它依赖于第一部分的结果。第3节的发展伴随着一系列在Haskell中的例子，这些例子可能有助于激发和理解不同的构造。 然后，第一部分有助于更深入地理解材料。

这些笔记还配有一系列练习，可以用来检查进展。 其中一些练习形成独立的主题，介绍了更高级的材料。例如，伴随与

Haskell程序员最喜欢的工具，单子：每个伴随都引出一个单子和一个余单子；反过来，每个（余）单子都可以通过一个伴随来定义。
这些高级练习标有一个‘$*$’。
享受阅读！

2 数据结构的伴随

这些讲义的第一部分结构如下。 第2.1节和2.2节提供了一些范畴论的背景知识，为这些讲义的其余部分做准备。 第2.3节和2.4节展示了如何以范畴论的方式建模非递归数据类型、有限积和和。 我们强调构造的计算性质，仔细研究了伴随的核心概念，然后在第2.5节中引入了它。这一节讨论了伴随的基本性质，并通过更多的例子来说明这个概念。 特别地，它引入了指数，用于建模高阶函数类型。 然后，第2.6节展示了如何捕捉递归数据类型，引入了初始代数和终结余代数。 这两个构造都源于一个伴随，与自由代数和余自由余代数相关，这些构造在相当深入地研究。 最后，第2.7节介绍了一个重要的范畴论工具，Yoneda引理，在讲义的第二部分中反复使用。

2.1 类别，函子和自然变换

本节介绍了范畴论的三位一体：类别，函子和自然变换。 如果你已经熟悉这个主题，那么你可以跳过这一节和下一节，除非是符号。 如果这是未知领域，试着吸收定义，研究例子，最重要的是，花时间让材料沉淀。

2.1.1 类别。一个类别由对象和对象之间的箭头组成。
我们让 $\mathscr{C}$, $\mathscr{D}$ 等等代表范畴。 我们写 $A : \mathscr{C}$ 来表示 A是 $\mathscr{C}$的一个对象。 我们让 A, B 等等代表对象。 对于每一对对象A, $B : \mathscr{C}$，存在从 A到 B的箭头类，记作 $\mathscr{C}(A, B)$。 如果从上下文中明显可以得知 $\mathscr{C}$，我们可以用 $f : A \to B$或 $f : B \leftarrow A$ 来缩写 $f : \mathscr{C}(\ A, B)$。 我们也可以宽松地说 $A \to B$是 f的类型。 我们让 f, g 等等代表箭头。

对于每个对象 $A : \mathscr{C}$ 存在一个箭头 $id_A : A \to A$，称为恒等箭头。
如果两个箭头的类型匹配，它们可以组合：如果 $f : A \to B$和 $g : B \to C$，那么 $g \cdot f : A \to C$。 我们要求组合是关联的，恒等箭头是其中性元素。

一个范畴通常被认为是其对象的类。 例如，我们说 **Set**是集合的范畴。 然而，同样重要的是范畴的箭头。 因此， **Set**实际上是集合和全函数的范畴。

（还有 **Rel**，即集合和关系的范畴。）对于 **Set**，恒等箭头是恒等函数，组合是函数组合。

如果对象具有附加结构（幺半群、群等），那么箭头通常是保持结构的映射。
如果对象具有附加结构（幺半群、群等），那么箭头通常是保持结构的映射。

练习*1*。定义范畴 **Mon**，其对象为幺半群，箭头为幺半群同态。

□

然而，范畴的对象不一定是集合，箭头也不一定是函数：

练习*2.*一个前序 $\lesssim$是一个范畴的极端例子：当且仅当$A \lesssim B$时， $\mathscr{C}(\ A, B)$是非空的。 因此，每个 $\mathscr{C}(A, B)$最多只有一个元素。详细说明一下。

□

练习*3.*幺半群是范畴的另一个极端例子：只有一个对象。详细说明一下。

□

一个范畴 $\mathscr{C}$的子范畴 $\mathscr{S}$是 $\mathscr{C}$的一些对象和一些箭头的集合，使得保留恒等和组合以确保 $\mathscr{S}$构成一个范畴。 在一个全子范畴中，对于所有的对象$\mathrm{A}, B : \mathscr{S}$， $\mathscr{S}(A, B) = \mathscr{C}(A, B)$。

如果存在一个箭头 $f : A \to B$，使得存在一个箭头 $g : A \leftarrow B$满足 $g \cdot f = id_A$和 $f \cdot g = id_B$，那么箭头 f是可逆的。 如果存在逆箭头，它是唯一的，并且用 f°表示。 如果存在一个可逆箭头$\mathrm{f} : A \to B$，那么对象 A和 B是同构的，记作 $A \cong B$。我们也用 $f : A \cong B : f^\circ$表示箭头 $f : A \to B$和箭头 $f^\circ : A \leftarrow B$见证了同构 $A \cong B$。

练习*4*。证明箭头的逆是唯一的。 □

2.1.2 函子。每个数学结构都带有保持结构的映射；同样，范畴也是如此，其中这些映射被称为函子。 （事实上，范畴论可以看作是对保持结构的映射的研究。Ma**c** La**n**e[8, p.30]写道：“范畴论对于每种类型的数学对象都提出了一个问题：‘它们的态射是什么？’”）由于范畴由两部分组成，即对象和箭头，一个函子 $\mathsf{F} : \mathscr{C} \to \mathscr{D}$由一个对象的映射和一个箭头的映射组成。 通常用相同的符号表示这两个映射。

我们也会宽松地将 F的箭头部分称为“映射”。 箭头的作用必须尊重类型：如果 $f : \mathscr{C}(A, B)$，那么 $\mathsf{F}\ f : \mathscr{D}(\mathsf{F}\,A, \mathsf{F}\,B)$。 此外， F必须保持恒等和组合：

$$\mathsf{F}\ id_A = id_{\mathsf{F}\,A}\ , \tag{1}$$

$$\mathsf{F}\,(g \cdot f) = \mathsf{F}\,g \cdot \mathsf{F}\,f\text{。} \tag{2}$$

函子性的力量在于对箭头的作用以及对组合的保持。 我们让 F， G等等遍历函子。 函子 $\mathsf{F} : \mathscr{C} \to \mathscr{C}$在范畴 $\mathscr{C}$上被称为自函子。

练习5。证明函子保持同构。

$$\mathsf{F}f : \mathsf{F}A \cong \mathsf{F}B : \mathsf{F}f^\circ \quad \Leftarrow\!= \quad f : A \cong B : f^\circ \square$$

*练习6。范畴 **Mon**比 **Set**具有更多的结构。 定义一个函子$\mathsf{U} : \mathbf{Mon} \to \mathbf{Set}$，它忽略了额外的结构。 （函子 U被称为遗忘或底层函子。）

□

存在一个恒等函子， $\mathsf{Id}_{\mathscr{C}} : \mathscr{C} \to_{\mathscr{C}}$，并且函子可以组合：$(\mathsf{G} \circ \mathsf{F})\ A = \mathsf{G}\,(\mathsf{F}\ A)$和 $(\mathsf{G} \circ \mathsf{F})\ f = \mathsf{G}\,(\mathsf{F}\ f)$。 这些数据将小范畴[1]和函子转化为一个称为 **Cat**的范畴。

练习7。证明 $\mathsf{Id}_{\mathscr{C}}$和 G∘F确实是函子。 □

2.1.3 自然变换。设 F， $\mathsf{G} : \mathscr{C} \to \mathscr{D}$是两个平行函子。一个变换 $\alpha : \mathsf{F} \to \mathsf{G}$是一组箭头，对于每个对象$\mathrm{A} : \mathscr{C}$存在一个箭头 $\alpha A : \mathscr{D}(\mathsf{F}\ A, \mathsf{G}\ A)$。 换句话说，一个变换是从对象到箭头的映射。 一个变换是自然的，如果 $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$，则

$$\mathsf{G}\,h \cdot \alpha\,\hat{A} = \alpha\,\check{A} \cdot \mathsf{F}\,h \ , \tag{3}$$

对于所有对象 $\hat{A}$ 和 $\check{A}$ 以及对于所有箭头 $h : \mathscr{C}(\hat{A}, \check{A})$。 请注意 α在两个不同的实例中使用： $\mathscr{D}(\mathsf{F}\,\hat{A}, \mathsf{G}\,\hat{A})$和 $\mathscr{D}(\mathsf{F}\,\check{A}, \mathsf{G}\,\check{A})$—我们将习惯性地使用一个帽子（ˆ）和一个倒置的帽子（ˇ）来装饰实例。
现在，给定 α和 h，基本上有两种方法可以将 $\mathsf{F}\,\hat{A}$的事物转化为 $\mathsf{G}\,\check{A}$的事物。 相容性条件（3）要求它们相等。该条件通过一个交换图来可视化：从相同源到相同目标的所有路径通过组合得到相同的结果。

$$\begin{array}{ccc} \mathsf{F}\,\hat{A} & \xrightarrow{\mathsf{F}\,h} & \mathsf{F}\,\check{A} \\ {\scriptstyle \alpha\,\hat{A}}\downarrow & & \downarrow{\scriptstyle \alpha\,\check{A}} \\ \mathsf{G}\,\hat{A} & \xrightarrow[\mathsf{G}\,h]{} & \mathsf{G}\,\check{A} \end{array}$$

如果 α是一个自然同构，我们写作 $\mathsf{F} \cong \mathsf{G}$。 举个例子，恒等映射是一个类型为 $\mathsf{F} \cong \mathsf{F}$的自然同构。 我们用 α， β等来表示自然变换的范围。

2.2 对偶、积和函子范畴

在前一节中，我们遇到了一些范畴的例子。 接下来，我们将展示如何从旧范畴创建新的范畴。

为了避免悖论，我们要求 **Cat**的对象是小的，即一个范畴被称为小范畴，当其对象类和所有箭头类都是集合时。 根据这个标准， **Set**和 **Cat**本身都不是小范畴。

2.2.1 对偶范畴。设 $\mathscr{C}$是一个范畴。 对偶范畴 $\mathscr{C}^{\mathsf{op}}$具有与 $\mathscr{C}$相同的对象； $\mathscr{C}^{\mathsf{op}}$ 中的箭头与 $\mathscr{C}$中的箭头一一对应，即 $f^{\mathsf{op}} : \mathscr{C}^{\mathsf{op}}(A, B)$当且仅当 $f : \mathscr{C}(B, A)$。 恒等映射和复合的定义是相反的：

$$id = id^{\mathsf{op}} \qquad \text{和} \qquad f^{\mathsf{op}} \cdot g^{\mathsf{op}} = (g \cdot f)^{\mathsf{op}} \ .$$

类型为 $\mathscr{C}^{\mathsf{op}} \to \mathscr{D}$ or $\mathscr{C} \to \mathscr{D}^{\mathsf{op}}$的函子有时被称为反变函子 从 $\mathscr{C}$到 $\mathscr{D}$的函子，通常的类型被称为协变。 操作 $(-)^{\mathsf{op}}$ 本身可以扩展为一个协变函子 $(-)^{\mathsf{op}} : \mathbf{Cat} \to \mathbf{Cat}$，其箭头 部分定义为 $\mathsf{F}^{\mathsf{op}}\, A = \mathsf{F}\, A$和 $\mathsf{F}^{\mathsf{op}}\, f^{\mathsf{op}} = (\mathsf{F}\, f)^{\mathsf{op}}$。 我们同意 $(f^{\mathsf{op}})^{\mathsf{op}} = f$， 因此该操作是一个对合。 （在后面的章节中，我们经常会粗心地 省略箭头上的双射 $(-)^{\mathsf{op}}$ 。）

一个有点近亲的反变函子的例子是预合成 $\mathscr{C}(-, B) : \mathscr{C}^{\mathsf{op}} \to \mathbf{Set}$，其对箭头的作用由 $\mathscr{C}(h^{\mathsf{op}}, B)\, f = f \cdot h$给出。 （映射和运算符的部分应用使用“范畴 虚拟符号”来表示，其中 $-$表示第一个参数， $=$表示第二个参数（如果有）。）该函子 $\mathscr{C}(-, B)$将一个对象 A映射到从A到固定的 B的箭头集合中， 并且它将一个箭头 $\mathscr{C}^{\mathsf{op}} : \mathscr{C}^{\mathsf{op}}(\hat{A}, \check{A})$映射到一个函数 $\mathscr{C}(h^{\mathsf{op}}, B) : \mathscr{C}(\hat{A}, B) \to \mathscr{C}(\check{A}, B)$。 对偶地，后合成 $\mathscr{C}(A, -) : \mathscr{C} \to \mathbf{Set}$ 是一个协变函子， 定义为 $\mathscr{C}(A, k)\, f = k \cdot f$。

练习*8.*证明 $\mathscr{C}(A, -)$和 $\mathscr{C}(-, B)$是函子。 □

2.2.2 乘积范畴。设 $\mathscr{C}_1$和 $\mathscr{C}_2$是范畴。 乘积范畴 $\mathscr{C}_1 \times \mathscr{C}_2$的对象是一对对象 $\langle A_1, A_2 \rangle$，其中 $A_1 : \mathscr{C}_1$和 $A_2 : \mathscr{C}_2$；箭头 $(\mathscr{C}_1 \times \mathscr{C}_2)(\langle A_1, A_2 \rangle, \langle B_1, B_2 \rangle)$是一对箭头 $\langle f_1, f_2 \rangle$，其中 $f_1 : \mathscr{C}_1(A_1, B_1)$和 $f_2 : \mathscr{C}_2(A_2, B_2)$。 恒等和复合定义为逐分量定义 ：

$$id = \langle id, id \rangle \qquad \text{和} \qquad \langle g_1, g_2 \rangle \cdot \langle f_1, f_2 \rangle = \langle g_1 \cdot f_1, g_2 \cdot f_2 \rangle \ .$$

投影函子 $\mathsf{Outl} : \mathscr{C}_1 \times \mathscr{C}_2 \to \mathscr{C}_1$和 $\mathsf{Outr} : \mathscr{C}_1 \times \mathscr{C}_2 \to \mathscr{C}_2$由 $\mathsf{Outl}\, \langle A_1, A_2 \rangle = A_1$, $\mathsf{Outl}\, \langle f_1, f_2 \rangle = f_1$和 $\mathsf{Outr}\, \langle A_1, A_2 \rangle = A_2$, $\mathsf{Outr}\, \langle f_1, f_2 \rangle = f_2$给出。 产品类别避免了多个参数的函子的需要。 来自产品类别的函子，如 Outl和 Outr 有时被称为双函子。 对角函子 $\Delta : \mathscr{C} \to \mathscr{C} \times \mathscr{C}$是一个进入产品类别的函子的例子：它复制了它的参 数 $\Delta A = \langle A, A \rangle$和 $\Delta f = \langle f, f \rangle$。

如果我们固定一个双函子的参数，我们得到一个函子。 相反的情况并不成 立：每个参数分别的函子性并不能暗示两者的函子性。 相反，我们有以下情况 ：$(- \otimes =) : \mathscr{C} \times \mathscr{D} \to \mathscr{E}$是一个双函子，如果对于所有的$A : \mathscr{C}$，部分应用 $(A \otimes -) : \mathscr{D} \to \mathscr{E}$是一个函子，对于所有的 $B : \mathscr{D}$，部分应用 $(- \otimes B) : \mathscr{C} \to \mathscr{E}$是一个函子 ，并且如果两个一元函子的集合满足交换律。

$$(\check{A} \otimes g) \cdot (f \otimes \hat{B}) = (f \otimes \check{B}) \cdot (\hat{A} \otimes g) \ , \tag{4}$$

对于所有的对象 $\hat{A}$, $\check{A}$, $\hat{B}$和 $\check{B}$以及所有的箭头 $f : \mathscr{C}(\hat{A}, \check{A})$和 $g : \mathscr{D}(\hat{B}, \check{B})$。给定 f和 g，有两种将$\hat{A} \otimes \hat{B}$ 的事物转化为$\check{A} \otimes \check{B}$ 的方式：

$$\begin{array}{ccc} \hat{A} \otimes \hat{B} & \xrightarrow{\hat{A} \otimes g} & \hat{A} \otimes \check{B} \\ {\scriptstyle f \otimes \hat{B}}\downarrow & \searrow^{f \oplus g} & \downarrow{\scriptstyle f \otimes \check{B}} \\ \check{A} \otimes \hat{B} & \xrightarrow[\check{A} \otimes g]{} & \check{A} \otimes \check{B} \,. \end{array}$$

相容性条件（4）要求它们相等。双函子的箭头部分，即对角线，由（4）的任一侧给出。交换律也可以看作是两个自然性条件，即 $f \otimes -$和$- \otimes g$是自然变换！

练习*9*。证明双函子的特征描述。 □

相应的“双自然”变换的概念更直观。设 $\mathsf{F}, \mathsf{G} : \mathscr{C} \times \mathscr{D} \to \mathscr{E}$是两个平行函子。变换 $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$在两个参数上都是自然的，当且仅当它在每个参数上都是分别自然的。

练习*10*。详细说明并证明该命题。 □

我们已经注意到，预合成 $\mathscr{C}(A, -)$和后合成 $\mathscr{C}(-, B)$是函子。预合成与后合成可交换：

$$\mathscr{C}(\check{A}, g) \cdot \mathscr{C}(f^{\mathsf{op}}, \hat{B}) = \mathscr{C}(f^{\mathsf{op}}, \check{B}) \cdot \mathscr{C}(\hat{A}, g) \ , \tag{5}$$

这是交换律（4）的一个实例，因此可以得出所谓的同态函子 $\mathscr{C}(-,=) : \mathscr{C}^{\mathsf{op}} \times \mathscr{C} \to \mathbf{Set}$是一个双函子。它将一对对象映射到它们之间的箭头集合，即所谓的同态集合；它对箭头的作用由以下给出

$$\mathscr{C}(f^{\mathsf{op}}, g)\, h = g \cdot h \cdot f \ . \tag{6}$$

2.2.3 函子范畴。存在一个恒等自然变换 $id_{\mathsf{F}} : \mathsf{F} \dot{\to} \mathsf{F}$定义为 $id_{\mathsf{F}}\, A = id_{\mathsf{F}\, A}$**.** 自然变换可以组合：如果 $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$和 $\beta : \mathsf{G} \dot{\to} \mathsf{H}$, 那么 $\beta \cdot \alpha : \mathsf{F} \dot{\to} \mathsf{H}$被定义为 $(\beta \cdot \alpha)\, A = \beta\, A \cdot \alpha\, A$**.** 因此，类型为 $\mathscr{C} \to \mathscr{D}$的函子和它们之间的自然变换构成一个范畴，即函子范畴 $\mathscr{D}^{\mathscr{C}}$**.** (函子范畴是 **Cat**中的指数，因此有这个符号。下一段是证明这个事实的第一步。)将一个函子应用于一个对象本身也是函子性的。具体来说，它是一个类型为 $(-=) : \mathscr{D}^{\mathscr{C}} \times \mathscr{C} \to \mathscr{D}$的双函子。使用双函子的特征，我们需要证明对于每个$\mathsf{F} : \mathscr{D}\mathscr{C}$，$(\mathsf{F}\, -) : \mathscr{C} \to \mathscr{D}$是一个函子，对于每个 $A : \mathscr{C}$，$(-\ A) : \mathscr{D}^{\mathscr{C}} \to \mathscr{D}$是一个函子，并且这两个集合满足交换律（4）。前者是显然的，因为 $(\mathsf{F}\, -)$就是 F**.** 后者的窄部分是 $(-\ A)\, \alpha = \alpha\, A$**.** 这个操作保持恒等性

而组合是 $\mathscr{D}^{\mathscr{C}}$的定义的结果。 最后，双函子的一致性条件（4）只是自然性条件（3）。 （实际上，可以反过来说：将函子应用转化为高阶函子的愿望决定了自然变换的概念，并且进而决定了 $\mathscr{D}^{\mathscr{C}}$的定义。）为了参考，我们记录下函子应用是一个双函子，其对箭头的作用是定义的。

$$\alpha\, f \;=\; \check{\mathsf{F}}\, f \cdot \alpha\hat{A} \;=\; \alpha\check{A} \cdot \hat{\mathsf{F}}\, f \text{。} \tag{7}$$

设 $\mathsf{F} : \mathscr{C} \to {}^{\mathscr{D}}$是一个函子。 预合成 $-\circ\mathsf{F}$本身是一个函子，一个在函子范畴之间的函子 $-\circ\mathsf{F} : \mathscr{E}^{\mathscr{D}} \to \mathscr{E}^{\mathscr{C}}$。 对于箭头，也就是自然变换，其作用被定义为 $(\alpha\circ\mathsf{F})\, A = \alpha\,(\mathsf{F}\, A)$。 对偶地，后合成$\mathsf{F}\circ-$是一个类型为 $\mathsf{F}\circ- : {}^{\mathscr{C}\,\mathscr{E}} \to \mathscr{D}^{\mathscr{E}}$ 的函子，其定义为 $(\mathsf{F}\circ\alpha)\, A = \mathsf{F}\,(\alpha\, A)$。

练习*11.*证明 $\alpha\circ\mathsf{F}$和 $\mathsf{F}\circ\alpha$是自然变换。 证明
$-\circ\mathsf{F}$和 $\mathsf{F}\circ-$保持恒等和组合。 □

前合成与后合成可交换：

$$(\check{\mathsf{F}}\circ\beta)\cdot(\alpha\circ\hat{\mathsf{G}}) = (\alpha\circ\check{\mathsf{G}})\cdot(\hat{\mathsf{F}}\circ\beta) \;\;,$$

再次，可以得出函子合成
$(-\circ=) : \mathscr{E}^{\mathscr{D}} \times \mathscr{D}^{\mathscr{C}} \to \mathscr{E}^{\mathscr{C}}$ 是一个双函子。

2.3 乘积和余积

范畴论中的定义通常采用普遍构造的形式，这是我们在本节中探讨的一个概念。 这种方法的典型例子是乘积的定义-实际上，这也是历史上的第一个例子。

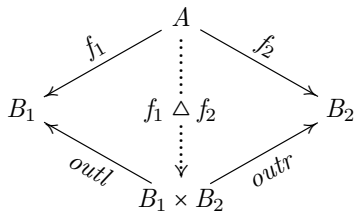
2.3.1 乘积。两个对象 B_1和 B_2的乘积由一个对象 $B_1 \times B_2$和一对箭头 $outl : B_1 \times B_2 \to B_1$和 $outr : B_1 \times B_2 \to B_2$组成。
这三个事物必须满足以下普适性质：对于每个对象 A和每对箭头 $f_1 : A \to B_1$和 $f_2 : A \to B_2$，存在一个唯一的箭头 $g : A \to B_1 \times B_2$，使得 $f_1 = outl \cdot g$和 $f_2 = outr \cdot g$。 （这个唯一的箭头也被称为中介箭头）。

如果我们用一个斯科伦函数[2]替换存在量词变量 g，那么普适性质可以更有吸引力地表述：对于每个对象 A和每对箭头 $f_1 : A \to B_1$和 $f_2 : A \to B_2$，存在一个箭头$f_1 \vartriangle f_2 : A \to B_1 \times B_2$（读作“$f_1$分裂 f_2”），使得

$$f_1 = outl \cdot g \;\;\wedge\;\; f_2 = outr \cdot g \;\;\Longleftrightarrow\;\; f_1 \vartriangle f_2 = g \;\;, \tag{8}$$

[2]存在量词变量 g在全称量词的范围内，因此需要一个斯科勒姆函数。

等价性捕捉到存在一个满足左侧属性的箭头，并且进一步说明 $f_1 \vartriangle f_2$ 是唯一的这样的箭头。 下面的图表总结了类型信息。



虚线箭头表示 $f_1 \vartriangle f_2$是从 A到 $B_1 \times B_2$的唯一箭头，使得图表成立。 任意两个 B_1 和 B_2的积都是同构的，这就是为什么我们通常称之为 theproduct（参见练习12）。 上述定义决定积只是同构的特性，而不是错误。一个好的范畴定义作为规范。 将其视为一个接口，可以有许多不同的实现方式。

像（8）这样的普适性质有两个直接的结果值得关注。如果我们将右边的值代入左边，我们得到计算规则（也称为β规则）：

$$f_1 = outl \cdot (f_1 \vartriangle f_2) \ , \tag{9}$$
$$f_2 = outr \cdot (f_1 \vartriangle f_2) \ . \tag{10}$$

它们可以被看作是定义箭头 $f \vartriangle g$的方程。

将（8）中的g实例化为恒等箭头$id_{B_1 \times B_2}$，我们得到反射规则（也称为简单η规则或η规则‘轻’）：

$$outl \vartriangle outr = id_{B_1 \times B_2} \ . \tag{11}$$

（11）这个规则表达了一个外延性质：将一个乘积拆开然后重新组装会得到原来的结果。

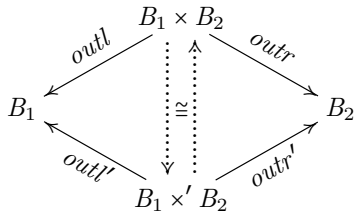
普遍性质还有两个进一步的结果，我们稍后将其识别为自然性质。 第一个结果是融合定律，它允许我们将一个分裂与一个箭头融合成另一个分裂：

$$(f_1 \vartriangle f_2) \cdot h = f_1 \cdot h \vartriangle f_2 \cdot h \ , \tag{12}$$

对于所有的 $h : \hat{A} \to \check{A}$。 该定律说明 $\vartriangle$在 A中是自然的。为了证明，我们推理

$$\begin{aligned}
& f_1 \cdot h \vartriangle f_2 \cdot h = (f_1 \vartriangle f_2) \cdot h \\
\Longleftrightarrow \quad & \{ \text{普遍性质 } (8) \ \} \\
& f_1 \cdot h = outl \cdot (f_1 \vartriangle f_2) \cdot h \ \ \wedge \ \ f_2 \cdot h = outr \cdot (f_1 \vartriangle f_2) \cdot h \\
\Longleftrightarrow \quad & \{ \text{ 计算 } (9)\text{–}(10) \qquad \} \\
& f_1 \cdot h = f_1 \cdot h \ \ \wedge \ \ f_2 \cdot h = f_2 \cdot h \ .
\end{aligned}$$

练习$_{12}$.使用计算、反射和融合来证明任意两个B_1和B_2的乘积是同构的。 更准确地说，B_1和B_2的乘积是唯一的，只要存在一个使图表



相符的同构。 （并不是说存在一个唯一的同构。例如，$B\times B$和$B\times B$之间存在两个同构：恒等同构$\mathrm{id}_{B\times B}$=outl△outr和outr△outl。）

□

练习*13*.展示

$$f_1 \vartriangle f_2 = g_1 \vartriangle g_2 \Longleftrightarrow f_1{=}g_1 \wedge f_2{=}g_2\text{，}\ f{=}g \Longleftrightarrow outl{\cdot}f{=}outl{\cdot}g \wedge outr{\cdot}f{=}outr{\cdot}g\text{。}$$

再次，尝试使用上述所有定律。 □

现在让我们假设乘积 $B_1 \times B_2$对于每个组合of B_1和 B_2都存在。 在这种情况下，乘积的定义在 B_1和 B_2中也是函子性的——上述描述中的两个对象都是完全被动的。 我们通过将 $\times$转化为类型 $\mathscr{C} \times \mathscr{C} \to \mathscr{C}$的函子来捕捉这个属性。 事实上，有一种唯一的方式可以将 $\times$转化为函子，使得投影箭头，$outl$和$outr$，在 B_1和 B_2中都是自然的：

$$k_1 \cdot outl = outl \cdot (k_1 \times k_2)\ , \tag{13}$$
$$k_2 \cdot outr = outr \cdot (k_1 \times k_2)\ . \tag{14}$$

我们诉诸于通用性质

$$\begin{aligned} & k_1 \cdot outl = outl \cdot (k_1 \times k_2)\ \wedge\ k_2 \cdot outr = outr \cdot (k_1 \times k_2) \\ & \qquad \Longleftrightarrow\ \{\,\text{通用性质 } (8)\,\} \\ & k_1 \cdot outl \vartriangle k_2 \cdot outr = k_1 \times k_2\ , \end{aligned}$$

这表明 $\times$的箭头部分被定义

$$f_1 \times f_2 = f_1 \cdot outl \vartriangle f_2 \cdot outr\ . \tag{15}$$

我们推迟证明 $\times$保留恒等和组合性质。

函数子融合定律表明我们可以将一个映射与一个分裂合并以形成另一个分裂：

$$(k_1 \times k_2) \cdot (f_1 \vartriangle f_2) = k_1 \cdot f_1 \vartriangle k_2 \cdot f_2\ , \tag{16}$$

该定律形式化了 $\triangle$在 B_1
和 B_2中的自然性。 (16)的证明基于融合和计算：

$$\begin{aligned}
&(k_1 \times k_2) \cdot (f_1 \vartriangle f_2) \\
=&\{\ \times\text{的定义 } (15)\ \} \\
&(k_1 \cdot outl \vartriangle k_2 \cdot outr) \cdot (f_1 \vartriangle f_2) \\
=&\quad \{\ \text{融合 } (12)\quad \} \\
&k_1 \cdot outl \cdot (f_1 \vartriangle f_2) \vartriangle k_2 \cdot outr \cdot (f_1 \vartriangle f_2) \\
=&\quad \{\ \text{计算 } (9)\text{–}(10)\qquad\qquad \} \\
&k_1 \cdot f_1 \vartriangle k_2 \cdot f_2\ .
\end{aligned}$$

在满足这些先决条件的情况下，很容易证明 $\times$保留了恒等性

$$\begin{aligned}
&id_A \times id_B \\
=&\{\ \times\text{的定义 } (15)\ \} \\
&id_A \cdot outl \vartriangle id_B \cdot outr \\
=&\quad \{\text{恒等和反射 } (11)\ \} \\
&id_{A \times B}
\end{aligned}$$

和组合

$$\begin{aligned}
&(g_1 \times g_2) \cdot (f_1 \times f_2) \\
=&\{\ \times\text{的定义 } (15)\ \} \\
&(g_1 \times g_2) \cdot (f_1 \cdot outl \vartriangle f_2 \cdot outr) \\
=&\quad \{\ \text{函子融合 } (16)\qquad \} \\
&g_1 \cdot f_1 \cdot outl \vartriangle g_2 \cdot f_2 \cdot outr \\
=&\{\ \times\text{的定义 } (15)\ \} \\
&g_1 \cdot f_1 \times g_2 \cdot f_2\ .
\end{aligned}$$

$\triangle$的自然性可以通过产品范畴和
同态函子精确地捕捉到（我们使用 $\forall X\ .\ \mathsf{F}\ X \to \mathsf{G}\ X$作为 $\mathsf{F} \dot{\to} \mathsf{G}$的简写）。

$$(\vartriangle) : \forall A, B\ .\ (\mathscr{C} \times \mathscr{C})(\Delta A, B) \to \mathscr{C}(A, \times B)$$

Split函数接受一对箭头作为参数，并返回一个箭头到一个乘积。
对象 B存在于一个乘积范畴中，所以 $\times B$是应用于 B的乘积函子；另一方面，
对象 A存在于 $\mathscr{C}$中，对角函子将其发送到 $\mathscr{C} \times \mathscr{C}$中的一个对象。 不要将对角函子 Δ（希腊字母Delta）与中介箭头 $\triangle$（一个朝上的三角形）混淆。 融合定律（12）捕捉了 A中的自然性，$\mathscr{C}$中的自然性（h，$\times B$）

$$\cdot (\vartriangle) = (\vartriangle) \cdot (\mathscr{C} \times \mathscr{C})(\Delta h,\ B)\text{，以}$$

及函子融合定律（16）捕捉了 B中的自然性，$\mathscr{C}$中的自然性

$$(A,\ \times k) \cdot (\vartriangle) = (\vartriangle) \cdot (\mathscr{C} \times \mathscr{C})(\Delta A, k)\ .$$

可以使用对角函子来捕捉 $outl$和 $outr$的自然性：

$$\langle outl,\ outr\rangle : \forall B\,.\,(\mathscr{C}\times\mathscr{C})(\Delta(\times B),\ B)。$$

自然性条件（13）和（14）等于

$$k\cdot\langle outl,\ outr\rangle = \langle outl,\ outr\rangle\cdot\Delta(\times k)\ .$$

所有这些的重要性在于 $\times$是对角函子 Δ的右伴随。
我们以后还需要更多地讨论伴随情况（第2.5节）。

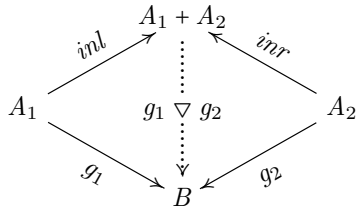
练习*14*。证明 $A\times B\cong B\times A$和 $A\times(B\times C)\cong(A\times B)\times C$。 □

练习*15*。$\langle A,\ B\rangle$和 $A\times B$之间的区别是什么？ □

2.3. 余积。构造乘积很好地对偶到余积，它们是对偶范畴中的乘积。 两个对象 A_1 和 A_2 的余积由一个写作 A_1+A_2 的对象和一对箭头$inl : A_1\to A_1+A_2$和 $inr : A_2\to A_1+A_2$组成。 这三个事物必须满足以下的普遍性质：对于每个对象 B和每对箭头$g_1 : A_1\to B$和 $g_2 : A_2\to B$，存在一个箭头 $g_1\triangledown g_2 : A_1+A_2\to B$（读作“$g_1$ join g_2”），使得

$$f = g_1\triangledown g_2 \iff f\cdot inl = g_1\ \wedge\ f\cdot inr = g_2\ , \tag{17}$$

对于所有 $f : A_1+A_2\to B$.



与乘积一样，普遍性质意味着计算、反射、融合和函子融合定律。计算定律:

$$(g_1\triangledown g_2)\cdot inl = g_1\ , \tag{18}$$
$$(g_1\triangledown g_2)\cdot inr = g_2\ . \tag{19}$$

反射定律:

$$id_{A+B} = inl\triangledown inr\ . \tag{20}$$

融合法则:

$$k\cdot(g_1\triangledown g_2) = k\cdot g_1\triangledown k\cdot g_2\ . \tag{21}$$

有一种唯一的方法可以将 $+$转化为一个函子，使得注入箭头在 A_1和 A_2中是自然的:

$$(h_1+h_2)\cdot inl = inl\cdot h_1\ , \tag{22}$$
$$(h_1+h_2)\cdot inr = inr\cdot h_2\ . \tag{23}$$

余积函子的箭头部分由以下给出

$$g_1 + g_2 = inl \cdot g_1 \triangledown inr \cdot g_2 \ . \tag{24}$$

函子融合定律:

$$(g_1 \triangledown g_2) \cdot (h_1 + h_2) = g_1 \cdot h_1 \triangledown g_2 \cdot h_2 \ . \tag{25}$$

这两个融合定律将 $\triangledown$识别为一个自然变换:

$$(\triangledown) : \forall\, A\ B\ .\ (\mathscr{C} \times \mathscr{C})(A, \Delta B) \to \mathscr{C}(+A, B) \ .$$

以下是 inl和 inr的自然性。

$$(\mathscr{C} \times \mathscr{C})(A, \Delta(+A))。$$

这一切的重要性在于 $+$是对角函子 Δ的左伴随。

2.4 初始对象和终结对象

如果对于每个对象 $B : \mathscr{C}$，对象 A被称为初始对象，那么从 A到 B的箭头只有一个。 任意两个初始对象是同构的，这就是为什么我们通常称之为初始对象的原因。 它被表示为0，从0到 B的唯一箭头被写作0 -> B或 B <-0。

$$0 \overset{0 \to B}{\dashrightarrow} B$$

这个唯一性也可以表达为一个普遍性属性：$f = 0 \to B \iff tr$

$$\text{ue}。 \tag{26}$$

对于所有 $f : 0 \to B$. 将 f实例化为恒等映射 id_0，我们得到反射定律：$id_0 = 0 \to 0$. 在唯一箭头之后的箭头可以融合成一个单一的唯一箭头。

$$k \cdot (\hat{B} \leftarrow 0) = (\check{B} \leftarrow 0) \ ,$$

对于所有 $k : \check{B} \leftarrow \hat{B}$· 融合定律表达了$0 \to B$在 B中的自然性。练习*16*。证明任意两个初始对象是同构的。 更准确地说，初始对象在唯一同构上是唯一的。 □

双对偶地，如果对于每个对象 $A : \mathscr{C}$ 存在唯一的箭头从 A到 1，写作 $A \to 1$ 或者 $1 \leftarrow A$，则 1 是一个终结对象。

$$A \overset{A \to 1}{\dashrightarrow} 1$$

我们采用算术符号来表示余积和积，初等和终结对象。 这个选择是有理由的，因为这些构造满足高中代数的许多定律（见练习14）。 下面的练习要求你进一步探索这个类比。

练习*17.*展示 $A + 0 \cong A$和对偶 $A \times 1 \cong A$. □

⋆练习*18.*那么 $A \times 0 \cong 0$和 $A \times (B + C) \cong A \times B + A \times C$怎么样? □

练习17表明0可以被看作是一个零元余积，而1则是一个零元积。 一般来说，如果一个范畴有一个终对象和二元积，则称其具有有限积。

2.5 伴随

我们在第2.3节中已经注意到积和余积是伴随的一部分。 在本节中，我们深入探讨了伴随的概念。

设 $\mathscr{C}$和 $\mathscr{D}$为范畴。 函子 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$和 $\mathsf{R} : \mathscr{C} \rightarrow \mathscr{D}$是伴随的，记作 $\mathsf{L} \dashv \mathsf{R}$。

$$\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D}$$

当且仅当在同态集合之间存在双射时

$$\lfloor - \rfloor : \mathscr{C}(\mathsf{L}\, A, B) \cong \mathscr{D}(A, \mathsf{R}\, B) : \lceil - \rceil \ ,$$

这在 A和 B之间都是自然的。 函子 L被称为 R的左伴随，而 R是 L的右伴随。同构 $\lfloor - \rfloor$被称为左伴随，而 $\lceil - \rceil$被称为右伴随。 选择 $\lfloor - \rfloor$作为左伴随的符号是因为开括号类似于'L'。 同样地——尽管这可能有点费力——$\lceil - \rceil$的开括号可以看作是一个角'r'。 左伴随的另一个名称是伴随转置，这就是为什么 $\lfloor f \rfloor$通常被称为 f的转置（通常命名为 f'）。

使用等价关系可以捕捉到 $\lfloor - \rfloor : \mathscr{C}(\mathsf{L}\, A, B) \rightarrow \mathscr{D}(A, \mathsf{R}\, B)$和 $\lceil - \rceil : \mathscr{C}(\mathsf{L}\, A, B) \leftarrow \mathscr{D}(A, \mathsf{R}\, B)$互为逆的情况。

$$f = \lceil g \rceil \quad \Longleftrightarrow \quad \lfloor f \rfloor = g \tag{27}$$

左边的方程存在于 $\mathscr{C}$，而右边的方程存在于 $\mathscr{D}$。作为一个简单的例子，恒

等函子是自伴随的： $\mathsf{Id} \dashv \mathsf{Id}$。 更一般地说，如果函子 F是可逆的，那么 F同时是左伴随和右伴随的： $\mathsf{F} \dashv \mathsf{F}^{\circ} \dashv \mathsf{F}$。 （注意，一般情况下 $\mathsf{F} \dashv \mathsf{G} \dashv \mathsf{H}$并不意味着 $\mathsf{F} \dashv \mathsf{H}$。）

2.5.1 产品和余积的再讨论。等式（27）类似于积的普遍性。 如果我们将（8）重新表述为涉及的范畴（不要混淆 Δ和 $\triangle$），那么后者确实定义了一个伴随。

$$f = \langle outl,\, outr \rangle \cdot \Delta g \quad \Longleftrightarrow \quad \triangle f = g \tag{28}$$

下图的右侧解释了这些类别。

$$\mathscr{C} \underset{\Delta}{\overset{+}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \times \mathscr{C} \underset{\times}{\overset{\Delta}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}$$

实际上，我们有一个双伴随，其中 $+$是 Δ的左伴随。 用乘积类别的术语重写后，余积的普遍性质（17）变为

$$f = \triangledown g \quad \Longleftrightarrow \quad \Delta f \cdot \langle inl,\, inr \rangle = g \text{ 。} \tag{29}$$

2.5.2 初始对象和终结对象的再讨论。初始对象和终结对象也定义了一个伴随，尽管是一个相当平凡的伴随。

$$\mathscr{C} \underset{\Delta}{\overset{0}{\underset{\longrightarrow}{\longleftarrow}}} \mathbf{1} \underset{1}{\overset{\Delta}{\underset{\longrightarrow}{\longleftarrow}}} \mathscr{C} \qquad (\bot)$$

类别 **1**由一个单一对象 $*$和一个单一箭头 id_*组成。 对角函子现在被定义为 $\Delta A = *$和 $\Delta f = id_*$。 对象0和1被视为从 **1**到常量函子的函子。 （一个对象 $A : \mathscr{C}$ 被视为一个函子 $A : \mathbf{1} \to \mathscr{C}$将 $*$映射到 A和 id_*映射到 id_A。 ）

$$f = (0 \dashrightarrow B) \cdot 0\, g \quad \Longleftrightarrow \quad \Delta f \cdot id_* = g \tag{30}$$

$$f = id_* \cdot \Delta g \quad \Longleftrightarrow \quad 1\, f \cdot (1 \dashrightarrow A) = g \tag{31}$$

通用性质在(30)的右侧和(31)的左侧都是空真值，因此它们有些退化。 此外，$0\, g$和$1\, f$都是恒等元素，因此(30)简化为(26)，(31)简化为等价关系$\text{true} \Longleftrightarrow A \dashrightarrow 1 = g$。

2.5.3 余单位和单位。伴随可以以多种方式定义。
请回忆一下，伴随 $\lfloor - \rfloor$和 $\lceil - \rceil$在 A和 B中都是自然的。

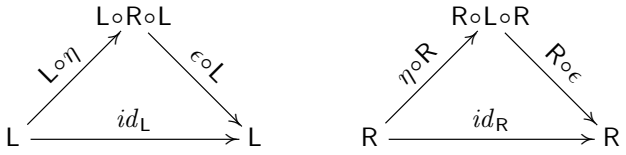
$$\lceil g \rceil \cdot \mathsf{L}\, h = \lceil g \cdot h \rceil$$
$$\mathsf{R}\, k \cdot \lfloor f \rfloor = \lfloor k \cdot f \rfloor$$

这意味着 $\lceil id \rceil \cdot \mathsf{L}\, h = \lceil h \rceil$和 $\mathsf{R}\, k \cdot \lfloor id \rfloor = \lfloor k \rfloor$。 因此，伴随通过它们对单位元素的映射来唯一确定： $\epsilon = \lceil id \rceil$和 $\eta = \lfloor id \rfloor$。 伴随的另一种定义是基于这两个自然变换，它们被称为伴随的余单位 $\epsilon : \mathsf{L} \circ \mathsf{R} \dot{\to} \mathsf{Id}$和单位 $\eta : \mathsf{Id} \dot{\to} \mathsf{R} \circ \mathsf{L}$。

这些单位必须满足所谓的三角恒等式

$$(\epsilon \circ \mathsf{L}) \cdot (\mathsf{L} \circ \eta) = id_{\mathsf{L}} \qquad \text{和} \qquad (\mathsf{R} \circ \epsilon) \cdot (\eta \circ \mathsf{R}) = id_{\mathsf{R}} \ . \tag{32}$$

图示解释了三角恒等式的名称。



总的来说，伴随包括六个实体：两个函子，两个伴随，和两个单位。 所有这些都可以根据其他实体来定义：

$$\begin{array}{lll} \lceil g \rceil = \epsilon\, B \cdot \mathsf{L}\, g & \epsilon = \lceil id \rceil & \mathsf{L}\, h = \lceil \eta\, B \cdot h \rceil \\ \lfloor f \rfloor = \mathsf{R} f \cdot \eta\, A & \eta = \lfloor id \rfloor & \mathsf{R}\, k = \lfloor k \cdot \epsilon\, A \rfloor \ , \end{array} \tag{33}$$

对于所有的 $f : \mathscr{C}(\mathsf{L}\, A, B)$, $g : \mathscr{D}(A, \mathsf{R}\, B)$, $h : \mathscr{D}(A, B)$和 $k : \mathscr{C}(A, B)$。

检查 (28) 我们注意到伴随 $\Delta \dashv \times$的余单位是投影箭头对 $\langle outl, outr\rangle$of。 单位是所谓的对角箭头 $\delta = id \vartriangle id$。 类似地，方程 (29) 表明 $+ \dashv \Delta$的单位是注入箭头对 $\langle inl, inr\rangle$ of。 余单位是所谓的余对角箭头 $id \triangledown id$。

练习*19.*展示两种定义伴随的等价方式:

1. 假设伴随是通过满足(27)的伴随函数来定义的。
   证明定义的单位 $\epsilon = \lceil id \rceil$和 $\eta = \lfloor id \rfloor$是自然的，并且满足三角恒等式(32)。

2. 反过来，假设伴随是通过满足三角恒等式(32)的单位来定义的。 证明定义的伴随函数 $\lceil g \rceil = \epsilon\, B \cdot \mathsf{L}\, g$和$\lfloor f \rfloor = \mathsf{R} f \cdot \eta\, A$是自然的，并且满足等价关系(27)。

□

2.5.4 伴随和编程语言。在编程语言的概念中，伴随对应于引入和消除规则：**split** $\vartriangle$引入一对，**join** $\triangledown$消除一个标记值。 单位可以被看作是这些规则的简单变体：**counit** $\langle outl, outr\rangle$消除对和**unit** $\langle inl, inr\rangle$引入标记值。 当我们讨论积时，我们从通用性质中推导出了各种法则。 表1使用新的术语重新表述了这些法则。 通过识别法则出现的单元格并从斜杠左侧或右侧读取标签，可以找到法则的名称。 例如，从右伴随的角度来看，恒等式 $f = \lceil \lfloor f \rfloor \rceil$对应于一个计算法则或 β-规则，从左边来看则是一个 η-规则。[3]伴随通常涉及一个简单或原始的函子。 在我们的运行示例中，这是对角函子 Δ，其伴随函数是有趣的新概念。 是新概念决定了视角。

因此，我们从右边观察等价关系（28）及其结果，并从左边观察其对偶（29）。 这个表格值得仔细研究。

2.5.5 通用箭头再探。我们在第**2.3**节中讨论了通用构造。现在让我们来研究一下它是如何推广的。由于伴随的组成部分是相互定义的，因此可以通过提供部分数据来指定伴随。 令人惊讶的是，只需要提供乘积的函子 $\mathsf{L} = \Delta$和通用箭头 $\epsilon = \langle outl, outr\rangle$，其他成分可以从这些中派生出来。 在本节的其余部分，我们将在伴随的更抽象设置中重演推导过程。

令 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$为一个函子， $\mathsf{R} : \mathscr{C} \to \mathscr{D}$为一个对象映射， $\epsilon : \mathscr{C}(\mathsf{L}\,(\mathsf{R}\,B), B)$为一个通用箭头。 普遍性意味着对于每个$f : \mathscr{C}(\mathsf{L}\,A, B)$，都存在一个唯一的箭头 $g : \mathscr{D}(A, \mathsf{R}\,B)$，使得 $f = \epsilon \cdot \mathsf{L}\, g$。 如同第2.3.1节中一样，我们用一个技巧性的写法 $\lfloor - \rfloor$来替换存在量化的变量 g。 然后该语句变为：对于每个 $f : \mathscr{C}(\mathsf{L}\,A, B)$

[3]同一个希腊字母既用于外延性（η规则）也用于伴随的单位。

表1.伴随和定律（从左/右视角）。

$\lceil - \rceil$ 引入 / 消除 $\lceil - \rceil : \mathscr{D}(A, \mathsf{R}\, B) \to \mathscr{C}(\mathsf{L}\, A, B)$	$\lfloor - \rfloor$ 消除 / 引入 $\lfloor - \rfloor : \mathscr{C}(\mathsf{L}\, A, B) \to \mathscr{D}(A, \mathsf{R}\, B)$
$f : \mathscr{C}(\mathsf{L}\, A, B)$ 普适性质 $g : \mathscr{D}(A, \mathsf{R}\, B)$ $f = \lceil g \rceil \iff \lfloor f \rfloor = g$	
$\epsilon : \mathscr{C}(\mathsf{L}\,(\mathsf{R}\, B), B)$ $\epsilon = \lceil id \rceil$	$\eta : \mathscr{D}(A, \mathsf{R}\,(\mathsf{L}\, A))$ $\lfloor id \rfloor = \eta$
— /计算规则 η规则 / β规则 $f = \lceil \lfloor f \rfloor \rceil$	计算规则/ — β规则 / η规则 $\lfloor \lceil g \rceil \rfloor = g$
反射定律/ — 简单 η规则 / 简单 β规则 $id = \lceil \eta \rceil$	— /反射定律 简单 β规则 / 简单 η规则 $\lfloor \epsilon \rfloor = id$
函子融合定律/ — $\lceil - \rceil$对 A是自然的 $\lceil g \rceil \cdot \mathsf{L}\, h = \lceil g \cdot h \rceil$	— /融合定律 $\lfloor - \rfloor$对 A是自然的 $\lfloor f \rfloor \cdot h = \lfloor f \cdot \mathsf{L}\, h \rfloor$
融合定律/ — $\lceil - \rceil$对 B是自然的 $k \cdot \lceil g \rceil = \lceil \mathsf{R}\, k \cdot g \rceil$	— /函子融合定律 $\lfloor - \rfloor$对 B是自然的 $\mathsf{R}\, k \cdot \lfloor f \rfloor = \lfloor k \cdot f \rfloor$
ϵ对 B是自然的 $k \cdot \epsilon = \epsilon \cdot \mathsf{L}\,(\mathsf{R}\, k)$	η对 A是自然的 $\mathsf{R}\,(\mathsf{L}\, h) \cdot \eta = \eta \cdot h$

存在一个箭头 $\lfloor f \rfloor : \mathscr{D}(A, \mathsf{R}\, B)$ 使得

$$f = \epsilon \cdot \mathsf{L}\, g \iff \lfloor f \rfloor = g \ , \tag{34}$$

该公式表明 $\epsilon \cdot \mathsf{L}\ g = \lceil\, g \rceil$.计算定律：
将右侧代入左侧，我们得到

$$f = \epsilon \cdot \mathsf{L} \lfloor f \rfloor \ . \tag{35}$$

反射定律：设置$f =$ □和$g = id$，得到

$$\lfloor \epsilon \rfloor = id \ . \tag{36}$$

融合定律：建立

$$\lfloor f \cdot \mathsf{L}\, h \rfloor = \lfloor f \rfloor \cdot h \ , \tag{37}$$

我们引用通用性质：

$$f \cdot \mathsf{L}\, h = \epsilon \cdot \mathsf{L}\,(\lfloor f \rfloor \cdot h) \iff \lfloor f \cdot \mathsf{L}\, h \rfloor = \lfloor f \rfloor \cdot h \ .$$

为了证明左边，我们计算

$$\begin{array}{ll} & \epsilon \cdot \mathsf{L}\,(\lfloor f \rfloor \cdot h) \\ = & \{\mathsf{L}\text{函子}\ (2)\} \\ & \epsilon \cdot \mathsf{L}\lfloor f \rfloor \cdot \mathsf{L}\,h \\ = & \{\text{计算}\ (35)\} \\ & f \cdot \mathsf{L}\,h\ . \end{array}$$

有一种独特的方式将对象映射R转化为函子，使得余单位有一B中是自然的：

$$k \cdot \epsilon = \epsilon \cdot \mathsf{L}\,(\mathsf{R}\,k)\ .$$

我们只需引用通用性质 (34)

$$k \cdot \epsilon = \epsilon \cdot \mathsf{L}\,(\mathsf{R}\,k) \quad \Longleftrightarrow \quad \lfloor k \cdot \epsilon \rfloor = \mathsf{R}\,k\ ,$$

这暗示着定义

$$\mathsf{R}\,f = \lfloor f \cdot \epsilon \rfloor\ . \tag{38}$$

函子融合定律：

$$\mathsf{R}\,k \cdot \lfloor f \rfloor = \lfloor k \cdot f \rfloor\ . \tag{39}$$

为了证明，我们推理

$$\begin{array}{ll} & \mathsf{R}\,k \cdot \lfloor f \rfloor \\ = & \{\mathsf{R}\text{定义}\ (38)\} \\ & \lfloor k \cdot \epsilon \rfloor \cdot \lfloor f \rfloor \\ = & \{\text{融合}\ (37)\} \\ & \lfloor k \cdot \epsilon \cdot \mathsf{L}\lfloor f \rfloor \rfloor \\ = & \{\ \text{计算}\ (35)\ \} \\ & \lfloor k \cdot f \rfloor\ . \end{array}$$

函子性：　R 保持恒等性

$$\begin{array}{ll} & \mathsf{R}\,id \\ = & \{\ \mathsf{R}\text{的定义}\ (38)\ \} \\ & \lfloor id \cdot \epsilon \rfloor \\ = & \{\text{恒等性和反射}\ (36)\ \} \\ & id \end{array}$$

和组合

$$\begin{aligned}
&\mathsf{R}\,g \cdot \mathsf{R}\,f \\
=&\ \{\ \mathsf{R}\text{的定义 } (38)\ \} \\
&\mathsf{R}\,g \cdot \lfloor f \cdot \epsilon \rfloor \\
=&\ \ \{\ \text{函子融合 } (39)\qquad \} \\
&\lfloor g \cdot f \cdot \epsilon \rfloor \\
=&\ \{\ \mathsf{R}\text{的定义 } (38)\ \} \\
&\mathsf{R}\,(g \cdot f)\ .
\end{aligned}$$

融合和函子融合表明 $\lfloor - \rfloor$在 A和 B中都是自然的。

相反地，一个函子 R和一个通用箭头 $\eta : \mathscr{C}\,(A, \mathsf{R}\,(\mathsf{L}\,A))$足以形成一个伴随关系。

$$f = \lceil g \rceil \quad \Longleftrightarrow \quad \mathsf{R}\,f \cdot \eta = g\ .$$

定义 $\lfloor f \rfloor = \mathsf{R}\,f \cdot \eta$ 和 $\mathsf{L}\,g = \lceil \eta \cdot g \rceil$.

练习*20*。明确自然性和融合之间的关系。 □

2.5.6 指数。让我们将伴随的抽象概念实例化为另一个具体的例子。 在 **Set**中，具有两个参数的函数 $A \times X \to B$可以被视为第一个参数 $A \to B^X$的函数，其值是函数的第二个参数。 一般来说，对象 B^X被称为 X和 B的指数。 使用应用 $apply : \mathscr{C}(\ B^X \times X, B)$可以消除 $B\,X$的元素。

应用是一个通用箭头的例子：对于每个 $f\ : \mathscr{C}(A \times X, B)$存在一个箭头 $\Lambda\ f : \mathscr{C}(A, B^X)$（发音为“curry f”），使得

$$f = apply \cdot (g \times id_X) \quad \Longleftrightarrow \quad \Lambda f = g\ , \tag{40}$$

函数 Λ将一个两个参数的函数转换为柯里化函数，因此得名。 我们认识到一个伴随情况，$- \times X \dashv (-)^X$.

$$\mathscr{C} \underset{(-)^X}{\overset{- \times X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \qquad\qquad \Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X) : \Lambda^\circ$$

左伴随是与 X配对，右伴随是从 X的指数。

转向定律，由于指数是右伴随，我们必须从右边查看表1。计算定律：

$$f = apply \cdot (\Lambda f \times id)\ . \tag{41}$$

反射定律：

$$\Lambda\text{应用} = \text{身份}\quad . \tag{42}$$

融合定律：

$$\Lambda f \cdot h = \Lambda\,(f \cdot (h \times id))\ . \tag{43}$$

有一种独特的方式可以将 $(-)^X$转化为一个函子，使得应用在B中是自然的
:

$$k \cdot \text{应用} = \text{应用} \cdot (k^X \times id) \ . \tag{44}$$

指数函子的箭头部分由以下给出

$$f^X = \Lambda\,(f \cdot \text{应用}) \ . \tag{45}$$

函子融合定律:

$$k^X \cdot \Lambda f = \Lambda\,(k \cdot f) \ . \tag{46}$$

指数具有一些额外的结构：如果所有必要的指数存在，则我们不仅有一个单一的伴随，而是一个伴随的族，$- \times X \dashv (-)\,X$，对于每个^{X}的选择都有一个。 指数在参数 X中是函子的，而伴随在该参数中是自然的。以下是详细信息：

我们已经知道 $\times$是一个双函子。 有一种唯一的方法将指数转化为一个双函子，必然是对第一个参数逆变的，使得双射 $\Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X)$: Λ°在 X上也是自然的：$\mathscr{C}(A \times \hat{X}, B) \xrightarrow{\Lambda} \mathscr{C}(A, B^{\hat{X}})$

$$\begin{array}{ccc} \mathscr{C}(A \times p, B) \downarrow & & \downarrow \mathscr{C}(A, B^p) \\ \mathscr{C}(A \times \check{X}, B) & \xrightarrow[\Lambda]{} & \mathscr{C}(A, B^{\check{X}}) \end{array}$$

我们将高层次的证明推迟到第2.7节。现在我们通过部分应用手动构造双函子。 固定一个对象 B。 $B^{(-)} : \mathscr{C}^{\mathsf{op}} \to \mathscr{C}$的箭头部分给出

$$B^p = \Lambda\,(apply \cdot (id \times p)) \ . \tag{47}$$

练习*21.*证明 $B^{(-)}$保持恒等和组合。 □

预合成 $B^{(-)}$与后合成 $(-)^A$交换：

$$g^{\check{A}} \cdot \hat{B}^f = \check{B}^f \cdot g^{\hat{A}} \ , \tag{48}$$

因此，所谓的内部同态函子 $(=)^{(-)} : \mathscr{C}^{\mathsf{op}} \times \mathscr{C} \to \mathscr{C}$是一个双函子。 它将一对对象映射到它们的指数；它对箭头的作用由 $g\ f = \Lambda(\ g \cdot apply \cdot (id \times f))$给出.

由于 Λ也是对 X自然的，我们还有另一个融合定律，即参数
融合定律:

$$B^p \cdot \Lambda f = \Lambda\,(f \cdot (A \times p))) \ . \tag{49}$$

我们还可以将这三个融合定律合并为一个定律:

$$k^p \cdot \Lambda f \cdot h = \Lambda\,(k \cdot f \cdot (h \times p))) \ .$$

然而， $apply$在 X中并不是自然的——它的源类型 $B^X \times X$ 甚至在X中也不是函子的。 (相反， $apply$是一个二自然变换的例子[12]，也可以参考[8，习题I X.4.1]。)

顺便说一下，具有有限积（$\Delta \dashv 1$ 和 $\Delta \dashv \times$存在）和指数（对于每个选择的 X，存在$-\times^X \dashv (-)^X$）的范畴被称为笛卡尔闭。

练习22。 证明逆变函子 $B^{(-)} : \mathscr{C}^{\mathrm{op}} \to \mathscr{C}$是自伴的:
$(B^{(-)})^{\mathrm{op}} \dashv B^{(-)}$. □

2.5.7 幂和余幂. 通过嵌套二进制乘积可以形成有限乘积: $A_1 \times (A_2 \times (\cdots \times A_n))$. (根据定义， n元乘积是 n =0时的最终对象1.) 或者，我们可以将乘积和余乘积推广到 n个分量（或者，事实上，推广到无限多个分量）。

双伴随 $+ \dashv \Delta \dashv \times$的核心是一个乘积范畴的概念。 乘积范畴 $\mathscr{C} \times \mathscr{C}$可以看作是一个简单的函子范畴: $\mathscr{C}^2$，其中2是某个两元素集合。为了能够处理任意数量的分量，我们将2推广为任意索引集合。一个集合形成所谓的离散范畴:对象是集合的元素，唯一的箭头是恒等箭头。 因此，从离散范畴到函子的映射是通

过其对对象的作用唯一确定的。 索引对象和箭头的范畴 $\mathscr{C}^I$，其中 I是某个任意索引集合，是一个从离散范畴到函子范畴的函子范畴: $A : \mathscr{C}^I$当且仅当 $\forall i \in I$.对角函子 $\Delta : \mathscr{C} \to \mathscr{C}^I$现在将每个索引映射到相同的对象: $(\Delta A)_i = A$. 对角函子的左伴随和右伴随推广了二进制构造。 对角函子的左伴随是一个简单形式的依赖和（也称为依赖乘积）。

$$\mathscr{C}(\sum i \in I \,.\, A_i, B) \cong \mathscr{C}^I(A, \Delta B)$$

它的右伴随是一个依赖积(也称为依赖函数空间)。

$$\mathscr{C}^I(\Delta A, B) \cong \mathscr{C}(A, \prod i \in I \,.\, B_i)$$

下面的图表总结了类型信息。

$$\mathscr{C} \underset{\Delta}{\overset{\sum i \in I \,.\, (-)_i}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}^I \underset{\prod i \in I \,.\, (-)_i}{\overset{\Delta}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}$$

让我们详细说明底层的通用性质。 箭头族 $\iota_k : A_k \to (\Sigma\, i \in I \,.\, A_i)$推广了二进制注入 inl和 inr。 对于每个箭头族 $\forall i \in I \,.\, g_i : A_i \to B$，存在一个箭头 $(\bigtriangledown\ i \in I \,.\, g_i) : (\Sigma\, i \in I \,.\, A_i) \to B$使得

$$f = (\bigtriangledown\ i \in I \,.\, g_i) \quad \Longleftrightarrow \quad (\forall i \in I \,.\, f \cdot \iota_i = g_i) \tag{50}$$

对于所有 $f : (\Sigma\, i \in I \,.\, A_i) \to B$.

相应地，家族 $\pi_k : (\Pi\ i \in I\ .\ B_i) \to B_k$推广了二元投影 $outl$ and $outr$. 对于每个箭头家族 $\forall i \in I\ .f_i : A \to B_i$，存在一个箭头 $(\bigwedge\ i \in I\ .\ f_i) : A \to (\Pi\ i \in I\ .\ B_i)$ 使得

$$(\forall i \in I\ .\ f_i = \pi_i \cdot g) \quad \Longleftrightarrow \quad (\bigwedge\ i \in I\ .\ f_i) = g \tag{51}$$

对于所有 $g : A \to (\Pi\ i \in I\ .\ B_i)$.

值得特别指出的是，我们将在以后需要的构造的一个特殊情况。首先，注意到 $\mathscr{C}^I(\Delta X, \Delta Y) \cong (\mathscr{C}(X, Y))^I \cong^I \to \mathscr{C}(X, Y)$.因此，如果求和的项和乘积的因子相同， $A_i = X$和 $B_i = Y$，我们得到另一个伴随情况：

$$\mathscr{C}(\sum I\ .\ X, Y) \cong I \to \mathscr{C}(X, Y)$$

退化的求和 $(X \sum I)$.也被称为 $copower$，有时写作 $I \bullet A$。

退化的乘积 $\prod I\ .A$也被称为 $power$，有时写作 A^I。

（因此， $\Sigma I \dashv \Pi I$本质上是柯里化的变体）。

2.5.8 伴随的性质。伴随满足许多性质。 一个值得记住的性质是，函子的左伴随和右伴随都是唯一的，可以通过自然同构来确定。 为了证明，假设函子 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$有两个右伴随：$\lfloor - \rfloor : \mathscr{C}(\mathsf{L}\ A, B) \cong \mathscr{D}(A, \mathsf{R}\ B)$ ，$\lfloor - \rfloor' : \mathscr{C}(\mathsf{L}\ A\ , B) \cong \mathscr{D}(A, \mathsf{R}'\ B\ )$。

自然同构由给出

$$\lfloor \epsilon \rfloor' : \mathsf{R} \cong \mathsf{R}' : \lfloor \epsilon' \rfloor\ .$$

我们展示同构的一半，另一半的证明完全类似（这解决了练习12，尽管是在抽象中）。

$$\begin{array}{ll} & \lfloor \epsilon \rfloor' \cdot \lfloor \epsilon' \rfloor \\ = & \{\ \text{融合：} \lfloor - \rfloor' \text{在 } A \text{中是自然的（表1）}\ \} \\ & \lfloor \epsilon \cdot \mathsf{L} \lfloor \epsilon' \rfloor \rfloor' \\ = & \{\ \text{计算（35）}\ \} \\ & \lfloor \epsilon' \rfloor' \\ = & \{\ \text{反射（表1）}\ \} \\ & id \end{array}$$

我们将在第2.6.5节中给出一个应用程序（在该应用程序中，我们展示了 $\mathsf{F}^*\ A \cong \mu \mathsf{F}_A$）。这里还有一个值得记忆的属性：左伴随保持初始对象和余积，而右伴随保持终结对象和积。 （一般来说，左伴随保持所谓的余极限，而右伴随保持所谓的极限。）在接下来的内容中，让 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$和 $\mathsf{R} : \mathscr{C} \to \mathscr{D}$成为一个伴随对的函子。

如果一个函子 $\mathsf{F} : \mathscr{C} \to \mathscr{D}$保持初始对象，那么它将 $\mathscr{C}$中的初始对象映射到 $\mathscr{D}$ 中的初始对象。 为了证明左伴随 L保持初始对象，我们需要证明对于每个对象 $B : \mathscr{C}$，存在一条从 L0到 B的唯一箭头。 所需的箭头只是从 R B到唯一箭头的转置。

$$
\begin{array}{ll}
 & f = \lceil 0 \dashrightarrow \mathsf{R}\, B \rceil \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad f = \lceil g \rceil \Longleftrightarrow \lfloor f \rfloor = g\ (27)\ \} \\
 & \lfloor f \rfloor = 0 \dashrightarrow \mathsf{R}\, B \\
\Longleftrightarrow & \{0\text{是初始对象：唯一性质}(26)\ \} \\
 & \text{真}
\end{array}
$$

由于初始对象在唯一同构下是唯一的（参见练习16），我们得出结论

$$\mathsf{L}\, 0 \cong 0 \ .$$

如果一个函子 $\mathsf{F} : \mathscr{C} \to \mathscr{D}$保持积 $B_1 \times B_2$，那么 $\mathsf{F}\,(B_1 \times B_2)$与$\mathsf{F}\, outl : \mathsf{F}\,(B_1 \times B_2) \to \mathsf{F}\, B_1$和 $\mathsf{F}\, outr : \mathsf{F}\,(B_1 \times B_2) \to \mathsf{F}\, B_2$是 $\mathsf{F}\, B_1$和 $\mathsf{F}\, B_2$的积。 为了证明右伴随 R保持 $B_1 \times B_2$，我们建立了积的普遍性质：

$$
\begin{array}{ll}
 & f_1 = \mathsf{R}\, outl \cdot g \ \wedge\ f_2 = \mathsf{R}\, outr \cdot g \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad f = \lceil g \rceil \Longleftrightarrow \lfloor f \rfloor = g\ (27)\ \} \\
 & \lceil f_1 \rceil = \lceil \mathsf{R}\, outl \cdot g \rceil \ \wedge\ \lceil f_2 \rceil = \lceil \mathsf{R}\, outr \cdot g \rceil \\
\Longleftrightarrow & \{\text{融合：}\ \lceil - \rceil\text{在 } B\text{中是自然的}(\text{表}1)\ \} \\
 & \lceil f_1 \rceil = outl \cdot \lceil g \rceil \ \wedge\ \lceil f_2 \rceil = outr \cdot \lceil g \rceil \\
\Longleftrightarrow & \{\ B_1 \times B_2\text{是一个乘积：通用性质（}8\text{）}\ \} \\
 & \lceil f_1 \rceil \vartriangle \lceil f_2 \rceil = \lceil g \rceil \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad f = \lceil g \rceil \Longleftrightarrow \lfloor f \rfloor = g\ (27)\ \} \\
 & \lfloor \lceil f_1 \rceil \vartriangle \lceil f_2 \rceil \rfloor = g \ .
\end{array}
$$

计算表明 $\lfloor \lceil f_1 \rceil \vartriangle \lceil f_2 \rceil \rfloor$是所需的中介箭头，分割 f_1和 f_2。 由于乘积在关于投影的唯一同构下是唯一的（参见练习12），我们有

$$\tau = \mathsf{R}\, outl \vartriangle \mathsf{R}\, outr : \mathsf{R}\,(B_1 \times B_2) \cong \mathsf{R}\, B_1 \times \mathsf{R}\, B_2 \ ,$$

因此

$$\mathsf{R}\, outl = outl \cdot \tau \ \wedge\ \mathsf{R}\, outr = outr \cdot \tau \ . \tag{53}$$

为了说明“保持性质”，让我们将 L $\dashv$ R实例化为“curry”伴随$- \times X \dashv (-)^X$。 对于左伴随，我们得到了熟悉的形式的定律：

$$
\begin{aligned}
0 \times X &\cong 0 \ , \\
(A_1 + A_2) \times X &\cong A_1 \times X + A_2 \times X \ .
\end{aligned}
$$

这些定律是一个分配范畴的要求（也可以参见练习17），这表明具有有限余积的笛卡尔闭范畴自动是分配的。 对于右伴随，我们得到了指数的两个定律：

$$1^X \cong 1 \ ,$$
$$(B_1 \times B_2)^X \cong B_1^X \times B_2^X \ .$$

另一个有趣的例子是由伴随 $(B^{(-)})^{\mathsf{op}} \dashv B^{(-)}$ 的练习22提供的。 由于自伴随函子 $B^{(-)}$是逆变的，它将初始对象映射到最终对象，余积映射到积：

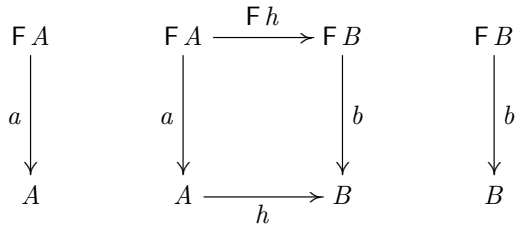
$$X^0 \cong 1 \ ,$$
$$X^{A_1+A_2} \cong X^{A_1} \times X^{A_2} \ .$$

我们得到了指数的另外两个定律。

2.6 初始代数和终极余代数

产品模型对应乘积类型，余产品模型和指数模型高阶函数类型。 在本节中，我们研究初始代数和最终余代数，它们给递归定义的类型赋予了意义。 我们将在这些笔记的第二部分（第3节）中更详细地讨论递归类型和递归类型上的函数。

2.6.1 初始代数。设 $\mathsf{F} : \mathscr{C} \to \mathscr{C}$是一个自函子。 一个 F-代数是一个对 $\langle A, a \rangle$，其中 $A : \mathscr{C}$（代数的载体）和一个箭头$a : \mathscr{C}$（$\mathsf{F}\, A, A$）（代数的作用）。 在代数 $\langle A, a\rangle$和 $\langle B, b\rangle$之间的 F-代数同态是一个箭头 $h : \mathscr{C}$（A, B），使得 $h \cdot a = b \cdot \mathsf{F}\, h$。 下面的图示说明了 F-代数及其同态。



有两种方法可以将 $\mathsf{F}\, A$转换为 B; F-代数同态的一致性要求它们相等。

身份是一个 F-代数同态，同态可以组合。 因此，数据定义了一个称为 F-**Alg**$(\mathscr{C})$或者只是 F-**Alg**的范畴，如果上下文中的基本范畴是明显的。 如果这个范畴中存在的话，初始对象被称为所谓的初始 F-代数 $\langle \mu\mathsf{F}, in\rangle$。 初始性的重要性在于

从 $\langle \mu \mathsf{F},\, in \rangle$到任何 F-代数 $\langle B,\, b \rangle$都存在唯一的箭头。 这个唯一的箭头被写作 (b)并且被称为折叠或范畴论。[4]用基本范畴的术语来表达，它满足以下唯一性属性。

$$f = (b) \quad \Longleftrightarrow \quad f \cdot in = b \cdot \mathsf{F}\, f \quad (\Longleftrightarrow \quad f : \langle \mu \mathsf{F},\, in \rangle \to \langle B,\, b \rangle) \tag{54}$$

类似于产品，唯一性属性有两个直接的结果。
将左边代入右边得到计算规律：

$$(b) \cdot in = b \cdot \mathsf{F}\,(b) \quad (\Longleftrightarrow \quad (b) : \langle \mu \mathsf{F},\, in \rangle \to \langle B,\, b \rangle) \ . \tag{55}$$

设 $f = id$和 $b = in$，我们得到反射规律：

$$id = (in) \text{ 。} \tag{56}$$

由于初始代数是一个初始对象，我们还有一个融合规律，用于将箭头与折叠结合形成另一个折叠。

$$k \cdot (\hat{b}) = (\check{b}) \quad \Longleftarrow\!= \quad k \cdot \hat{b} = \check{b} \cdot \mathsf{F}\, k \quad (\Longleftrightarrow \quad k : \langle \hat{B},\, \hat{b} \rangle \to \langle \check{B},\, \check{b} \rangle) \tag{57}$$

如果以范畴 F-$\mathbf{Alg}(\mathscr{C})$的术语来表述，证明是平凡的。 然而，我们也可以在底层范畴 $\mathscr{C}$中执行证明。

$$\begin{array}{ll}
 & k \cdot (\hat{b}) = (\check{b}) \\
\Longleftrightarrow & \{\ \text{唯一性属性 } (54) \qquad\qquad \} \\
 & k \cdot (\hat{b}) \cdot in = \check{b} \cdot \mathsf{F}\,(k \cdot (\hat{b})) \\
\Longleftrightarrow & \{\ \text{计算规律 } (55) \qquad \} \\
 & k \cdot \hat{b} \cdot \mathsf{F}\,(\hat{b}) = \check{b} \cdot \mathsf{F}\,(k \cdot (\hat{b})) \\
\Longleftrightarrow & \{\ \mathsf{F} \text{ 函子 } (2) \qquad \} \\
 & k \cdot \hat{b} \cdot \mathsf{F}\,(\hat{b}) = \check{b} \cdot \mathsf{F}\, k \cdot \mathsf{F}\,(\hat{b}) \\
\Longleftarrow\!= & \{\text{在两边都取消} - \cdot \mathsf{F}\,(\hat{b}))\ \} \\
 & k \cdot \hat{b} = \check{b} \cdot \mathsf{F}\, k \ .
\end{array}$$

融合定律表明 $(-))$在 $\langle B,\, b \rangle$中是自然的，也就是说，在 F-$\mathbf{Alg}(\mathscr{C})$中是一个箭头。这并不意味着在底层类别 $\mathscr{C}$中是自然的。（作为 $\mathscr{C}$中的一个箭头，折叠 $(-))$是一个强二自然变换。）使用这些定律，我们可以证明$L\mathrm{am}be\mathit{k}$的引理[13]，它表明 $\mu\mathsf{F}$是该函子的一个不动点： $\mathsf{F}\,(\mu\mathsf{F}) \cong \mu\mathsf{F}$。这个同构由$in : \mathscr{C}(\mathsf{F}\,(\mu\mathsf{F}), \mu\mathsf{F})$:$(\mathsf{F}\, in))$见证。我们计算$in \cdot (\mathsf{F}\, in)) = id \Longleftrightarrow \{$反射 $(56)\ \} in \cdot (\mathsf{F}\, in)) = (in)) \Longleftarrow\!= \{$融合 $(57)\ \} in \cdot \mathsf{F}\, in = in \cdot \mathsf{F}\mathrm{in}$.

[4] Catamorphism这个术语是由Meertens创造的，符号 $(-))$是由Malcolm提出的，而香蕉括号的名称归功于Van der Woude。

对于反向的方向，我们进行推理

$$\begin{aligned}
&(\mathsf{F}\,in)\cdot in\\
=\;&\{\ \text{计算}\ (55)\qquad\qquad\}\\
&\mathsf{F}\,in\cdot\mathsf{F}\,(\mathsf{F}\,in)\\
=\;&\{\ \mathsf{F}\ \text{函子}\ (2)\quad\}\\
&\mathsf{F}\,(in\cdot(\mathsf{F}\,in))\\
=\;&\{\ \text{参见上面的证明}\ \}\\
&\mathsf{F}\,id\\
=\;&\{\ \mathsf{F}\ \text{函子}\ (1)\quad\}\\
&id\ .
\end{aligned}$$

作为一个例子， $Bush = \mu\mathsf{B}$其中 $\mathsf{B}\,A = \mathbb{N}+(A\times A)$定义了二叉叶树的类型：一棵树要么是一个带有自然数标签的叶子，要么是由两个子树组成的节点。 二叉叶树可以用来表示非空的自然数序列。 为了定义一个计算这样一个序列和的函数，我们需要提供一个类型为 $\mathsf{B}\,\mathbb{N}\to\mathbb{N}$的代数。 箭头$id\ \triangledown\ plus$其中 $plus$是加法，非常合适。 因此，计算总和的函数为 $(id\ \triangledown\ plus))$。

练习*23.*探索类别 Id-$\mathbf{Alg}(\mathscr{C})$其中 Id是恒等函子。
确定初始 Id-代数。 □

∗练习*24.*包含函子 $\mathsf{Incl}:\mathscr{C}\to\mathsf{Id}\text{-}\mathbf{Alg}(\mathscr{C})$，定义 $\mathsf{Incl}\,A = \langle A,\ id\rangle$
和 $\mathsf{Incl}\,f\ = f$，将底层类别嵌入到 Id-代数类别中。
Incl有左伴随还是右伴随? □

练习*25.*探索类别 K-$\mathbf{Alg}(\mathscr{C})$其中 $\mathsf{K}\,A\ =\ C$是常数
函子。 确定初始 K-代数。 □

练习*26.*是否存在最终 F-代数? □

如果所有必要的初始代数存在，我们可以将 μ转化为高阶类型的函子 $\mathscr{C}^{\mathscr{C}}\to\mathscr{C}$。 这个函子的对象部分将一个函子映射到它的初始代数；箭头部分将一个自然变换 $\alpha:\mathsf{F}\dot{\to}\ \mathsf{G}$映射到一个箭头 $\mu\alpha:\mathscr{C}(\mu\mathsf{F},\mu\mathsf{G})$。 有一种唯一的方式来定义这个箭头，使得箭头 $in:\mathsf{F}\,(\mu\mathsf{F})\to\mu\mathsf{F}$在 F中是自然的：

$$\mu\alpha\cdot in = in\cdot\alpha\,(\mu\alpha)\ . \tag{58}$$

请注意，高阶函子 $\lambda\mathsf{F}\ .\,\mathsf{F}\,(\mu\,\mathsf{F})$，其对箭头的作用是$\lambda\,\alpha\,.\,\alpha\,(\mu\alpha) = \lambda\,\alpha\,..\,\alpha\,(\mu\mathsf{G})\cdot\mathsf{F}\,(\mu\alpha)$，涉及到‘应用函子’(7) 。 要推导出 $\mu\alpha$，我们只需引用通用性质 (54) ：

$$\mu\alpha\cdot in = in\cdot\alpha\,(\mu\mathsf{G})\cdot\mathsf{F}\,(\mu\alpha)\quad\Longleftrightarrow\quad\mu\alpha = (in\cdot\alpha\,(\mu\mathsf{G}))\ .$$

为了减少混乱，我们通常会在右侧省略 α的类型参数并定义

$$\mu\alpha = (in \cdot \alpha) \quad (59)$$

与乘积一样，我们推迟证明 μ 保留了恒等性和组合性。

折叠函数还享有第二个融合定律，我们称之为基础函子融合或者只是基础融合定律。 它表明我们可以将一个映射后的折叠函数融合成另一个折叠函数：

$$(b \cdot \alpha) = (b) \cdot \mu\alpha \ , \tag{60}$$

对于所有的 $\alpha : \hat{\mathsf{F}} \dot{\to} \check{\mathsf{F}}$，要建立基础融合，我们推理如下。 为了建立基础融合，我们推理

$$\begin{array}{ll}
& (b) \cdot \mu\alpha = (b \cdot \alpha) \\
\Leftarrow\Rightarrow & \{\ \mu\text{的定义}(59)\ \} \\
& (b) \cdot (in \cdot \alpha) = (b \cdot \alpha) \\
\Leftarrow= & \{\ \text{融合}\ (57)\ \} \\
& (b) \cdot in \cdot \alpha = b \cdot \alpha \cdot \hat{\mathsf{F}}\,(b) \\
\Leftarrow\Rightarrow & \{\ \text{计算}\ (55)\ \} \\
& b \cdot \check{\mathsf{F}}\,(b) \cdot \alpha = b \cdot \alpha \cdot \hat{\mathsf{F}}\,(b) \\
\Leftarrow\Rightarrow & \{\ \alpha\ \text{是自然的:}\ \check{\mathsf{F}}\,h \cdot \alpha = \alpha \cdot \hat{\mathsf{F}}\,h\ \} \\
& b \cdot \alpha \cdot \hat{\mathsf{F}}\,(b) = b \cdot \alpha \cdot \hat{\mathsf{F}}\,(b) \ .
\end{array}$$

在满足这些先决条件的情况下，很容易证明 μ保持恒等

$$\begin{array}{ll}
& \mu id \\
= & \{\ \mu\text{的定义}\ \ (59)\ \} \\
& (in \cdot id) \\
= & \{\ \text{恒等和反射}\ (56)\ \} \\
& id
\end{array}$$

和组合

$$\begin{array}{ll}
& \mu\beta \cdot \mu\alpha \\
= & \{\ \mu\text{的定义}\ (59)\ \} \\
& (in \cdot \beta) \cdot \mu\alpha \\
= & \{\text{基础融合}(60)\ \} \\
& (in \cdot \beta \cdot \alpha) \\
= & \{\ \mu\text{的定义}\ (59)\ \} \\
& \mu(\beta \cdot \alpha) \ .
\end{array}$$

总结一下，基础融合表达了 $(-))$在 F:

$$(-)) : \forall \mathsf{F} \ .\ \mathscr{C}(\mathsf{F}\,B, B) \to \mathscr{C}(\mu\mathsf{F}, B)$$ 中的自然性。

注意 $\mathscr{C}(-B,B)$和 $\mathscr{C}(\mu-,B)$是逆变的、高阶函子，类型为 $\mathscr{C}^{\mathscr{C}} \to \mathbf{Set}^{\mathsf{op}}$。

举个例子， $\alpha = succ \triangledown id$是一个类型为 B ⇀ B的自然变换。 窄 $\mu\alpha$增加了二叉叶树中包含的标签。

2.6.2 最终余代数。开发很好地对偶到 F-余代数和展开。 一个 F-余代数是一个由对象$\mathscr{C}$:$\mathscr{C}$和一个箭头 $c : C(\ C, \mathsf{F}\ \mathscr{C})$组成的对 $\langle\ \mathscr{C},\ c\rangle$。 在余代数 $\langle C,\ c\rangle$和 $\langle D,\ d\rangle$之间的 F-余代数同态是一个箭头 $h : C(C,D)$使得 $\mathsf{F}\, h \cdot c = d \cdot h$。 恒等映射是一个 F-余代数同态，而同态可以组合。 因此，数据定义了一个称为 F-**Coalg**($\mathscr{C}$)或者简称 F-**Coalg**的范畴。 如果存在的话，这个范畴中的最终对象被称为所谓的最终 F-余代数 $\langle \nu\mathsf{F},\ out\rangle$。 终态的重要性在于从任何F-余代数 $\langle C,\ c\rangle$到 $\langle \nu\,\mathsf{F}\,,\ out\rangle$存在唯一的箭头。

这个独特的箭头被写作 $[\!(c]\!)$并被称为展开或解构。用基本类别的术语来表达，它满足以下唯一性属性。

$$(g : \langle C,\ c\rangle \to \langle \nu\mathsf{F},\ out\rangle \quad \Longleftrightarrow) \quad \mathsf{F}\, g \cdot c = out \cdot g \quad \Longleftrightarrow \quad [\!(c]\!) = g \tag{61}$$

与初始代数一样，唯一性属性意味着计算、反射、融合和基础融合定律。计算定律：

$$([\!(c]\!) \ : \langle C,\ c\rangle \to \langle \nu\mathsf{F},\ out\rangle \quad \Longleftrightarrow) \quad \mathsf{F}\,[\!(c]\!) \ \cdot c = out \cdot [\!(c]\!) \tag{62}$$

反射定律：

$$[\!(out]\!) = id \ . \tag{63}$$

融合法则：

$$[\!(\hat{c}]\!) = [\!(\check{c}]\!) \cdot h \quad \Longleftarrow\!= \quad \mathsf{F}\, h \cdot \hat{c} = \check{c} \cdot h \quad (\Longleftrightarrow \quad h : \langle \hat{C},\ \hat{c}\rangle \to \langle \check{C},\ \check{c}\rangle) \ . \tag{64}$$

有一种唯一的方式可以将 ν转化为一个函子，使得 out在 F中是自然的：

$$\alpha\,(\nu\alpha) \cdot out = out \cdot \nu\alpha \ .$$

函子 ν的箭头部分由以下给出

$$\nu\alpha = [\!(\alpha \cdot out]\!) \ . \tag{65}$$

基本融合法则：

$$\nu\alpha \cdot [\!(c]\!) = [\!(\alpha \cdot c]\!) \ . \tag{66}$$

以 $Tree = \nu\mathsf{T}$为例，

其中 $\mathsf{T}\, A = A \times \mathbb{N} \times A$定义了分叉的类型，即无限二叉树的自然数。 (分叉是将状态或动作分为两个分支的过程。) 展开 $generate = [\!(shift0 \vartriangle id \vartriangle shift1]\!)$, 其中$shift0\ n = 2 * n + 0$ 和$shift1\ n = 2 * n + 1$, 生成了一个无限树： $generate\, 1$包含所有正整数。

练习*27.*探索类别 Id-**Coalg**($\mathscr{C}$)其中 Id是恒等函子。
确定最终 Id-余代数。

⊔⊓练习*28.*探索类别 K-**Coalg**($\mathscr{C}$)其中 $\mathsf{K}\, A = C$是常量函子。 确定最终 K-余代数。 □

练习*29.*是否存在初始 F-余代数? □

2.6.3 自由代数和余自由代数。我们已经用伴随的方式解释了余积、积和指数。我们能否对初始代数和最终余代数做同样的事情？ 嗯，初始代数是一个初始对象，因此它是 F-**Alg**和 **1**之间的一个平凡伴随。 同样地，最终余代数是一个最终对象，产生了 **1**和 F-**Coalg**之间的一个伴随。以下提供了一个更令人满意的答案：

类别 F-$\mathbf{Alg}(\mathscr{C})$比 $\mathscr{C}$有更多的结构。 遗忘或底层函子 U ： F-$\mathbf{Alg}(\mathscr{C}) \to \mathscr{C}$忽略了额外的结构： U$\langle A,\, a\rangle = A$和U $h = h$。 类似的函子可以为 F-$\mathbf{Coalg}(\ \mathscr{C})$定义。尽管遗忘函子的定义看起来很简单，但它们通过两个伴随引出了两个有趣的概念。

$$\text{F-}\mathbf{Alg}(\mathscr{C}) \underset{\mathsf{U}}{\overset{\text{自由}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \underset{\text{余自由}}{\overset{\mathsf{U}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \text{F-}\mathbf{Coalg}(\mathscr{C})$$

函子 Free将对象 A映射到所谓的自由 F-代数上 A。 对偶地，Cofree将对象 A映射到余自由 F-余代数上 A。

2.6.4 自由代数。 让我们更深入地探讨自由代数的概念。 首先，自由 A是一个 F-代数。 我们将其动作命名为 com，原因一会儿就清楚了。 在 **Set**中， U(Free A)的元素是由 F确定的构造器构建的术语，并且从 A中提取变量。 将 F函子视为描述语言语法的语法规则。 动作 $com : \mathscr{C}(\mathsf{F}\,(\mathsf{U}\,(\text{Free}\,A)), \mathsf{U}\,(\text{Free}\,A))$从子术语的F-结构构造一个复合术语。 还有一个操作 $var : \mathscr{C}(A, \mathsf{U}\,(\text{Free}\,A))$用于将变量嵌入术语中。 这个操作是一个进一步的通用箭头的例子：对于每个 $g : \mathscr{C}(A, \mathsf{U}\,B)$，都存在一个 F-代数同态映射$eval\, g$: F-$\mathbf{Alg}(\text{Free}\,A, B)$（发音为“用 g评估”），使得

$$f = eval\ g \quad \Longleftrightarrow \quad \mathsf{U} f \cdot var = g \ , \tag{67}$$

换句话说，一个术语的含义是由变量的含义唯一确定的。 事实上，$eval\, g$是一个同态映射，这意味着含义函数是组合的：一个复合术语的含义是根据其组成部分的含义来定义的。

普遍性质意味着通常的法则。 尽管U对箭头的作用是无操作， U$h = h$，但我们不会省略对 U的应用，因为它提供了有价值的“类型信息”：它明确了 h是一个F-代数同态，而不仅仅是 $\mathscr{C}$中的一个箭头。计算法则：

$$\mathsf{U}\,(eval\ g) \cdot var = g \text{。} \tag{68}$$

反射定律：

$$id = eval\ var \ . \tag{69}$$

融合定律：

$$k \cdot eval\ g = eval\,(\mathsf{U}\,k \cdot g) \ . \tag{70}$$

请注意，计算定律适用于基础类别 $\mathscr{C}$，而反射定律和融合定律适用于 F-$\mathbf{Alg}(\mathscr{C})$。

通常情况下，有一种独特的方法将 Free转化为一个函子，使得单位变量在 A 中是自然的：

$$\mathsf{U}\,(\mathsf{Free}\,h)\cdot var = var\cdot h \quad . \tag{71}$$

函子 Free的箭头部分如下所示：

$$\mathsf{Free}\,g = eval\,(var\cdot g) \quad . \tag{72}$$

函子融合定律：

$$eval\;g\cdot\mathsf{Free}\,h = eval\,(g\cdot h) \quad . \tag{73}$$

代数 $com : \mathscr{C}(\mathsf{F}\,(\mathsf{U}\,(\mathsf{Free}\,A)), \mathsf{U}\,(\mathsf{Free}\,A))$在 A中也是自然的。

$$\mathsf{U}\,(\mathsf{Free}\,h)\cdot com = com\cdot\mathsf{F}\,(\mathsf{U}\,(\mathsf{Free}\,h)) \text{ 。} \tag{74}$$

这只是一个事实的重新表述 $\mathsf{Free}\,h : \mathsf{Free}\,A \to \mathsf{Free}\,B$ 是一个 F-代数同态。

作为一个例子，平方函子的自由代数 $\mathsf{Sq}\,A = A\times A$ gen-eralises 从自然数的类型中抽象出来的二叉叶树的类型： $v$ar 创建一个叶子和 com一个内部节点（参见例子21和练习42）。

⋆练习*30*。每个伴随 $\mathsf{L}\dashv\mathsf{R}$都会产生一个单子 R∘L。 这个练习要求你探索 M = U∘Free，即所谓的自由单子 F。 单子的单位是 $var : \mathsf{Id}\dot{\to}\ \mathsf{M}$，它将一个变量嵌入到一个项中。 单子的乘法， $join : \mathsf{M}\circ\mathsf{M}\dot{\to}\ \mathsf{M}$，实现了替换。 使用 $eval$定义$join$并证明单子定律（∘比 ·的优先级更高）：

$$\begin{aligned} join\cdot var\circ\mathsf{M} &= id_{\mathsf{M}} \quad , \\ join\cdot\mathsf{M}\circ var &= id_{\mathsf{M}} \quad , \\ join\cdot join\circ\mathsf{M} &= join\cdot\mathsf{M}\circ join \quad . \end{aligned}$$

这些定律捕捉了替换的基本属性。请解释。 □

练习*31*.什么是函子 Id的自由单子？ 常量函子 $\mathsf{K}\,A = C$的自由单子有用吗？

□

⋆练习*32*.练习6要求你定义忘却函子 $\mathsf{U} : \mathbf{Mon}\to\mathbf{Set}$，它忽略了 **Mon**的附加结构。 证明函子$\mathsf{Free} : \mathbf{Set}\to\mathbf{Mon}$，它将集合 A映射到 A上的自由幺半群，是 U的左伴随。 这个伴随的单位是熟悉的列表处理函数。哪些函数？

□

自由代数具有一些额外的结构。 与乘积一样，我们没有一个单一的伴随，而是一族伴随，每个选择 F对应一个伴随。 自由代数的构造在基础的基础函子 F上是函子性的，并且操作在该函子中是自然的。 与乘积相比，情况更加复杂

因为每个选择的 F都会产生不同的代数范畴。 我们需要一些基础设施来快速切换这些不同的范畴：

令人惊讶的是，构造 $(-)$-**Alg**可以转化为一个逆变函子，类型为 $\mathscr{C}^{\mathscr{C}} \to \mathbf{Cat}^{\mathrm{op}}$：它将函子 F发送到F-代数的范畴，将自然变换 $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$发送到函子 α-**Alg** : G-**Alg** $\to$ F-**Alg**，定义为 α-**Alg** $\langle A,\, a\rangle = \langle A,\, a \cdot \alpha\, A\rangle$和 α-**Alg** $h = h$。 （作为一个例子， $\mu\alpha$是一个类型为 $\langle \mu\ \mathsf{F},\, in\rangle \to \alpha$-**Alg** $\langle \mu\mathsf{G},\, in\rangle$的F-代数同态。）出现了一些证明义务。 我们必须证明 G-代数同态 $h : A \to B$也是一个类型为 α-**Alg** $A \to \alpha$-**Alg** B的 F-代数同态。

$$
\begin{aligned}
& h \cdot a \cdot \alpha\, A \\
= \ & \{\text{假设：}h\text{是一个G-代数同态：}\ h{\cdot}a{=}b{\cdot}\mathsf{G}h\} \\
& b \cdot \mathsf{G}\, h \cdot \alpha\, A \\
= \ & \{\ \alpha\text{是自然变换：}\ \mathsf{G}h{\cdot}\alpha A\hat{}{=}\alpha A\check{}{\cdot}\mathsf{F}h\} \\
& b \cdot \alpha\, B \cdot \mathsf{F}\, h
\end{aligned}
$$

此外，$(-)$-**Alg**必须保持恒等性和组合性。

$$(id_{\mathsf{F}})\text{-}\mathbf{Alg} = \mathsf{Id}_{\mathsf{F}\text{-}\mathbf{Alg}} \tag{75}$$

$$(\alpha \cdot \beta)\text{-}\mathbf{Alg} = \beta\text{-}\mathbf{Alg} \circ \alpha\text{-}\mathbf{Alg} \tag{76}$$

证明很简单，因为函子$(-)$-**Alg**不改变代数的载体。 这是一个值得特别关注的重要性质：$\mathsf{U}_{\mathsf{F}} \circ \alpha\text{-}\mathbf{Alg} = \mathsf{U}_{\mathsf{G}}$。

$$\begin{array}{ccc} \mathsf{G}\text{-}\mathbf{Alg}(\mathscr{C}) & \xrightarrow{\ \alpha\text{-}\mathbf{Alg}\ } & \mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C}) \\ & {\scriptstyle \mathsf{U}_{\mathsf{G}}} \searrow \quad \swarrow {\scriptstyle \mathsf{U}_{\mathsf{F}}} & \\ & \mathscr{C} & \end{array} \tag{77}$$

由于涉及到多个基础函子，我们已经用相应的函子对构造进行了索引。

借助这个新的机制，我们现在可以将融合定律（70）推广到不同类型的同态。 假设 $\alpha : \mathsf{F} \dot{\to} \mathsf{G}$，我们有

$$\mathsf{U}_{\mathsf{G}}\, k \cdot \mathsf{U}_{\mathsf{F}}\, (eval_{\mathsf{F}}\ g) = \mathsf{U}_{\mathsf{F}}\, (eval_{\mathsf{F}}\, (\mathsf{U}_{\mathsf{G}}\, k \cdot g))\ 。 \tag{78}$$

原始的融合定律存在于 F-**Alg**$(\mathscr{C})$，而这个定律存在于底层范畴 $\mathscr{C}$。 证明中必须要用到性质（77）。

$$
\begin{aligned}
& \mathsf{U}_{\mathsf{G}}\, k \cdot \mathsf{U}_{\mathsf{F}}\, (eval_{\mathsf{F}}\ g) \\
= \ & \{\ \mathsf{U}_{\mathsf{G}} = \mathsf{U}_{\mathsf{F}} \circ \alpha\text{-}\mathbf{Alg}\ (77)\ \} \\
& \mathsf{U}_{\mathsf{F}}\, (\alpha\text{-}\mathbf{Alg}\, k) \cdot \mathsf{U}_{\mathsf{F}}\, (eval_{\mathsf{F}}\ g) \\
= \ & \{\ \mathsf{U}_{\mathsf{F}}\ \text{函子（2）}\ \} \\
& \mathsf{U}_{\mathsf{F}}\, (\alpha\text{-}\mathbf{Alg}\, k \cdot eval_{\mathsf{F}}\ g)
\end{aligned}
$$

$$
\begin{aligned}
&= \quad \{\ \text{融合}\ (70)\quad \} \\
&\quad\ \mathsf{U_F}\,(\text{评估}_\mathsf{F}\,(\mathsf{U_F}\,(\alpha\text{-}\mathbf{Alg}\;k)\cdot g)) \\
&= \quad \{\ \mathsf{U_G} = \mathsf{U_F}\circ\alpha\text{-}\mathbf{Alg}\ (77)\ \} \\
&\quad\ \mathsf{U_F}\,(\text{评估}_\mathsf{F}\,(\mathsf{U_G}\,k\cdot g))
\end{aligned}
$$

α-**Alg**的应用可以看作是一个适配器。 在计算中，代数是不可见的-它们可以通过使用提供的类型信息来显式表示。

现在让我们转向问题的核心。 引入一个快捷方式来表示自由代数的载体是方便的：

$$\mathsf{F}^* = \mathsf{U_F}\circ\mathsf{Free_F}\ 。 \tag{79}$$

这定义了一个函子，其箭头部分为 $\mathsf{F}^*\ g = \mathsf{U_F}\,(\text{评估}_\mathsf{F}\,(\text{变量}_\mathsf{F}\cdot g))$。 使用 F^* 我们可以为构造函数分配更简洁的类型：$\text{变量}_\mathsf{F} : \mathscr{C}\,(A, \mathsf{F}^*\ A)$和 $com_\mathsf{F} : \mathscr{C}\,(\mathsf{F}\,(\mathsf{F}^*\ A), \mathsf{F}^*\ A)$。

我们声称 $(-)\quad{}^*$ 是一个高阶函子类型 $\mathscr{C}^{\mathscr{C}} \to \mathscr{C}\,\mathscr{C}$将基本函子 F映射到所谓的自由单子函子 F的。 通常情况下，我们希望推导出箭头部分的定义，它将自然变换 $\alpha : \mathsf{F} \dot{\to}\ \mathsf{G}$映射到自然变换 $\alpha^* : \mathsf{F}^* \dot{\to}\ \mathsf{G}^*$。 人们希望构造函数 $var_\mathsf{F} : {}^{\mathscr{C}}(A, \mathsf{F}^*\ A)$和 $com_\mathsf{F} : \mathscr{C}\,(\mathsf{F}\,(\mathsf{F}^*\ A), \mathsf{F}^*\ A)$在 F中是自然的：

$$
\begin{aligned}
\alpha^*\ A\cdot var_\mathsf{F} &= var_\mathsf{G}\ , \\
\alpha^*\ A\cdot com_\mathsf{F} &= com_\mathsf{G}\cdot\alpha\,(\alpha^*\ A)\ 。
\end{aligned}
$$

请注意，函子 $\lambda\mathsf{F}\ ._\mathsf{F}(\ \mathsf{F}^*\ A)$，其对箭头的作用是$\lambda\,\alpha\,.\,\alpha\,(\alpha^*\ A) =\lambda\,\alpha\,.\,\alpha\,(\mu\mathsf{G})\cdot\mathsf{F}\,(\ \alpha^*\ A)$，涉及到'应用函子'（7）。 因此，第二个条件表达了箭头 $\alpha^*\ A$是类型为F ree F $A \to \alpha$-**Alg** $(\mathsf{Free_G}\ A)$的 F-同态的事实：

$$
\begin{aligned}
&\alpha^*\ A\cdot com_\mathsf{F} = com_\mathsf{G}\cdot\alpha\,(\mu\mathsf{G})\cdot\mathsf{F}\,\alpha^*\ A \\
&\quad\Longleftrightarrow\quad \alpha^*\ A : \langle\mathsf{F}^*\ A,\ com_\mathsf{F}\rangle \to \langle\mathsf{G}^*\ A,\ com_\mathsf{G}\cdot\alpha\,(\mu\mathsf{G})\rangle\ .
\end{aligned}
$$

为了推导 $\alpha^*\ A$的箭头部分，我们推理

$$
\begin{aligned}
&\quad\ \alpha^*\ A\cdot var_\mathsf{F} = var_\mathsf{G} \\
&\Longleftrightarrow\quad \{\ \alpha^*\ A\text{是一个 F-同态，见上文}\ \} \\
&\quad\ \mathsf{U_F}\,(\alpha^*\ A)\cdot var_\mathsf{F} = var_\mathsf{G} \\
&\Longleftrightarrow\quad \{\ \text{通用性质（67）}\qquad\qquad\ \} \\
&\quad\ \alpha^*\ A = eval_\mathsf{F}\ var_\mathsf{G} \\
&\Longleftrightarrow\quad \{\ \ eval_\mathsf{F}\ var_\mathsf{G}\text{是一个 F-同态}\ \} \\
&\quad\ \alpha^*\ A = \mathsf{U_F}\,(eval_\mathsf{F}\ var_\mathsf{G})\ .
\end{aligned}
$$

有两点需要注意。 首先，在第二步中，通用性质适用于 $var_{\mathsf{G}} : A \to \mathsf{U}_{\mathsf{G}}\,(\mathsf{Free}_{\mathsf{G}}\,A) = A \to \mathsf{U}_{\mathsf{F}}\,(\alpha\text{-}\mathbf{Alg}\,(\mathsf{Free}_{\mathsf{G}}\,A))$。 其次，这些方程式存在于不同的类别中：最后一个方程式存在于 $\mathscr{C}$中，而倒数第二个方程式存在于 $\mathsf{F}\text{-}\mathbf{Alg}(\mathscr{C})$中。 总结一下，高阶函子 α^*的箭头部分被定义为

$$\alpha^*\ A = \mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ var_{\mathsf{G}})\ . \tag{80}$$

转向证明，我们首先要证明 α^*确实是自然的。假设 $h : \mathscr{C}(\hat{A},\ \check{A})$，那么

$$\begin{aligned}
&\mathsf{G}^*\ h \cdot \alpha^*\ \hat{A} \\
=\ &\{\ \mathsf{G}^*(79)\text{和 }\alpha^*(80)\text{的定义}\ \} \\
&\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\,(var_{\mathsf{G}} \cdot h)) \cdot \mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ var_{\mathsf{G}}) \\
=\ &\{\text{广义融合 }(78)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\,(\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\,(var_{\mathsf{G}} \cdot h)) \cdot var_{\mathsf{G}})) \\
=\ &\{\ \text{计算 }(68)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\,(var_{\mathsf{G}} \cdot h)) \\
=\ &\{\ \text{函子融合 }(73)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ var_{\mathsf{G}} \cdot \mathsf{Free}_{\mathsf{F}}\ h) \\
=\ &\{\ \mathsf{U}_{\mathsf{F}}\text{ 函子 }(2)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ var_{\mathsf{G}}) \cdot \mathsf{U}_{\mathsf{F}}\,(\mathsf{Free}_{\mathsf{F}}\ h) \\
=\ &\{\ \alpha^*(80)\text{和 }\mathsf{F}^*(79)\text{的定义}\ \} \\
&\alpha^*\ \check{A} \cdot \mathsf{F}^*\ h\ .
\end{aligned}$$

像往常一样，我们推迟证明 $(-)^*$保留恒等性和组合性。

基本函子融合定律表明

$$\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\ g) \cdot \alpha^*\ A = \mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ g)\ 。 \tag{81}$$

我们推理

$$\begin{aligned}
&\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\ g) \cdot \alpha^*\ A \\
=\ &\{\ (-)^*\text{的定义 }(80)\ \} \\
&\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\ g) \cdot \mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ var_{\mathsf{G}}) \\
=\ &\{\text{广义融合 }(78)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\,(\mathsf{U}_{\mathsf{G}}\,(eval_{\mathsf{G}}\ g) \cdot var_{\mathsf{G}})) \\
=\ &\{\ \text{计算 }(68)\ \} \\
&\mathsf{U}_{\mathsf{F}}\,(eval_{\mathsf{F}}\ g)\ 。
\end{aligned}$$

在满足这些先决条件的情况下，很容易证明 $(-)^*$保留恒等性 $(id : \mathsf{F} \dot{\to} \mathsf{F})$

$$
\begin{aligned}
& id^*\ A \\
= & \{\ (-)^*(80)\text{和}(75)\text{的定义}\ \} \\
& \mathsf{U_F}\,(eval_{\mathsf{F}}\ var_{\mathsf{F}}) \\
= & \quad \{\ \text{反射}(69) \qquad \} \\
& \mathsf{U_F}\ id \\
= & \quad \{\ \mathsf{U_F}\ \text{函子}(1) \qquad \} \\
& id\ \ ,
\end{aligned}
$$

和组合$(\beta : \mathsf{G} \dot{\to} \mathsf{H}$和 $\alpha : \mathsf{F} \dot{\to} \mathsf{G})$

$$
\begin{aligned}
& \beta^* \cdot \alpha^* \\
= & \{\ (-)^*\ (80)\text{的定义}\ \} \\
& \mathsf{U_G}\,(eval_{\mathsf{G}}\ var_{\mathsf{H}}) \cdot \alpha^* \\
= & \quad \{\text{基函子融合}(81)\ \} \\
& \mathsf{U_F}\,(eval_{\mathsf{F}}\ var_{\mathsf{H}}) \\
= & \{\ (-)^*(80)\text{和}(76)\text{的定义}\ \} \\
& (\beta \cdot \alpha)^*\ \ .
\end{aligned}
$$

基函子融合表达了 $\mathsf{U_F}\,(eval_{\mathsf{F}}\ -) : \mathscr{C}(A, \mathsf{U}\,B) \to \mathscr{C}(\mathsf{F}^*\,A, \mathsf{U}\,B)$对 F的自然性——注意 F出现在逆变位置。

2.6.5 关于 $\mathbf{F}^*$ 和 $\boldsymbol{\mu}\mathbf{F}$的关系. 由于左伴随保留初始对象，我们有 $\mathsf{Free}\,0 \cong \langle \mu\mathsf{F},\ in \rangle$并且 $\mathsf{F^*}\,0 = \mathsf{U}\,(\mathsf{Free}\,0) \cong \mathsf{U}\,\langle \mu\mathsf{F},\ in \rangle = \mu\mathsf{F}$—这一步使用了函子（这里是 U）保留同构的事实（参见练习**5**）。换句话说， μF的元素是闭合项，即没有变量的项。相反，自由代数可以用初始代数来表示：$\langle \mathsf{F}^*\ A,\ com \rangle \cong \langle \mu\mathsf{F}_A,\ in \cdot inr \rangle$其中 $\mathsf{F}_A\ X = A + \mathsf{F}\ X$. 函子 F_A形式化了一个项是变量还是复合项。对于这个表示，$in \cdot$ **inl** 扮演了var 的角色， $in \cdot$ **in**r 扮演了 com的角色。为了证明这个同构，我们展示了上述数据确定了一个以 U为右伴随的伴随关系。由于左伴随在自然同构下是唯一的，结果成立。（代数之间的同构 $\langle \mathsf{F}^*\ A,\ com \rangle \cong \langle \mu\mathsf{F}_A,\ in \cdot inr \rangle$甚至在 A上是自然的。）

转向证明，我们展示对于每个 $g : \mathscr{C}(A, \mathsf{U}\,B)$存在一个F-代数同态 $\mathsf{F}\text{-}\mathbf{Alg}(\langle \mu\mathsf{F}_A,\ in \cdot inr \rangle, \langle B,\ b \rangle)$满足通用性质 (67)。 我们声称 $(g \triangledown b))$是所需的同态。

以下计算显示 $(g \triangledown b))$确实是一个 F-代数同态。

$$
\begin{aligned}
& (g \triangledown b)\mathbf{)} \cdot in \cdot inr \\
= & \quad \{ \text{ 计算 } (55) \qquad \} \\
& (g \triangledown b) \cdot \mathsf{F}_A\,(g \triangledown b)\mathbf{)} \cdot inr \\
= & \quad \{ \ \mathsf{F}_A\,X = A + \mathsf{F}\,X \text{ 且 } \quad inr \text{ 是自然的 } (23) \ \} \\
& (g \triangledown b) \cdot inr \cdot \mathsf{F}\,(g \triangledown b)\mathbf{)} \\
= & \quad \{ \text{ 计算 } (19) \qquad \} \\
& b \cdot \mathsf{F}\,(g \triangledown b)\mathbf{)}
\end{aligned}
$$

为了建立通用性质（67），我们推理

$$
\begin{aligned}
& f = (g \triangledown b)\mathbf{)} \\
\Longleftrightarrow & \quad \{ \text{ 唯一性性质（54） } \qquad \} \\
& f \cdot in = (g \triangledown b) \cdot \mathsf{F}_A\,f \\
\Longleftrightarrow & \quad \{ \ \mathsf{F}_A\,X = A + \mathsf{F}\,X \text{ 和函子融合（25） } \qquad \} \\
& f \cdot in = g \triangledown b \cdot \mathsf{F}\,f \\
\Longleftrightarrow & \quad \{ \text{ 通用性质（17） } \qquad \} \\
& f \cdot in \cdot inl = g \ \ \wedge \ \ f \cdot in \cdot inr = b \cdot \mathsf{F}\,f \\
\Longleftrightarrow & \quad \{ \ f : \langle \mu\mathsf{F}_A,\, in \cdot inr \rangle \to \langle B,\, b \rangle \ \} \\
& \mathsf{U}\,f \cdot in \cdot inl = g \ \ .
\end{aligned}
$$

最后一步利用了 f在 F-代数同态上的范围。

2.6.6 香蕉分裂。伴随 $\mathsf{Free} \dashv \mathsf{U}$告诉我们很多关于代数范畴结构的信息。 我们已经利用了左伴随保持初始对象的事实： $\mathsf{Free}\,0 \cong 0$。 由于右伴随保持终结对象和积，我们还知道

$$\mathsf{U}\,1 \cong 1 \ , \tag{82}$$

$$\mathsf{U}\,(B_1 \times B_2) \cong \mathsf{U}\,B_1 \times \mathsf{U}\,B_2 \ . \tag{83}$$

由于 U是一个遗忘函子，我们可以利用这些性质来推导代数范畴中终结对象和积的定义。 性质（82）表明最终代数由（这解决了练习26）

$$1 = \langle 1,\, \mathsf{F}\,1 \mathrel{-\!\!\!\rightarrow} 1 \rangle \ .$$

最终代数的作用是确定的，因为从F1到1的箭头只有一个。 从任何代数 $\langle A,\, a \rangle$到最终代数的唯一同态是简单的 $A \mathrel{-\!\!\!\rightarrow} 1$。 同态条件， $(1 \mathrel{\leftarrow\!\!\!-} A) \cdot a = (1 \mathrel{\leftarrow\!\!\!-} \mathsf{F}\,1) \cdot \mathsf{F}\,(1 \mathrel{\leftarrow\!\!\!-} A)$，来自于融合。

同样，属性（83）确定了乘积代数的载体。 为了确定它的作用，我们进行如下推理。 产品的保持也意味着

$\mathsf{U}\,outl = outl$和 $\mathsf{U}\,outr = outr$—这些是（53）的实例，假设在（83）中是相等而不是同构。 换句话说，outl 和outr 必须是 F-代数同态： $outl : \langle B_1 \times B_2,\, x\rangle \to \langle B_1,\, b_1\rangle$ 和 $outr : \langle B_1 \times B_2,\, x\rangle \to \langle B_2,\, b_2\rangle$其中 x是待确定的作用。让我们计算一下。

$$
\begin{aligned}
& outl : \langle B_1 \times B_2,\, x\rangle \to \langle B_1,\, b_1\rangle \;\;\wedge\;\; outr : \langle B_1 \times B_2,\, x\rangle \to \langle B_2,\, b_2\rangle \\
\Longleftrightarrow\;\; & \{\,\text{同态条件}\,\} \\
& outl \cdot x = b_1 \cdot \mathsf{F}\, outl \;\;\wedge\;\; outr \cdot x = b_2 \cdot \mathsf{F}\, outr \\
\Longleftrightarrow\;\; & \{\,\text{通用性质（8）}\,\} \\
& x = b_1 \cdot \mathsf{F}\, outl \vartriangle b_2 \cdot \mathsf{F}\, outr
\end{aligned}
$$

因此，代数的乘积被定义为

$$\langle B_1,\, b_1\rangle \times \langle B_2,\, b_2\rangle = \langle B_1 \times B_2,\, b_1 \cdot \mathsf{F}\, outl \vartriangle b_2 \cdot \mathsf{F}\, outr\rangle\ .$$

还有一个最终的证明义务：我们必须证明中介箭头 $\vartriangle$ 将同态映射到同态映射。

$$
\begin{aligned}
& f_1 \vartriangle f_2 : \langle A,\, a\rangle \to \langle B_1,\, b_1\rangle \times \langle B_2,\, b_2\rangle \\
& \quad \Longleftarrow\;\; f_1 : \langle A,\, a\rangle \to \langle B_1,\, b_1\rangle \;\;\wedge\;\; f_2 : \langle A,\, a\rangle \to \langle B_2,\, b_2\rangle
\end{aligned}
$$

我们推理

$$
\begin{aligned}
& (f_1 \vartriangle f_2) \cdot a \\
= \;\; & \{\,\text{fusion (12)}\,\} \\
& f_1 \cdot a \vartriangle f_2 \cdot a \\
= \;\; & \{\,\text{假设：}\quad f_1 : \langle A,\, a\rangle \to \langle B_1,\, b_1\rangle \wedge f_2 : \langle A,\, a\rangle \to \langle B_2,\, b_2\rangle\,\} \\
& b_1 \cdot \mathsf{F}\, f_1 \vartriangle b_2 \cdot \mathsf{F}\, f_2 \\
= \;\; & \{\,\text{计算 (9)–(10)}\,\} \\
& b_1 \cdot \mathsf{F}\,(outl \cdot (f_1 \vartriangle f_2)) \vartriangle b_2 \cdot \mathsf{F}\,(outr \cdot (f_1 \vartriangle f_2)) \\
= \;\; & \{\,\mathsf{F}\ \text{函子 (2)}\,\} \\
& b_1 \cdot \mathsf{F}\, outl \cdot \mathsf{F}\,(f_1 \vartriangle f_2) \vartriangle b_2 \cdot \mathsf{F}\, outr \cdot \mathsf{F}\,(f_1 \vartriangle f_2) \\
= \;\; & \{\,\text{融合 (12)}\,\} \\
& (b_1 \cdot \mathsf{F}\, outl \vartriangle b_2 \cdot \mathsf{F}\, outr) \cdot \mathsf{F}\,(f_1 \vartriangle f_2)\ .
\end{aligned}
$$

使用乘积代数我们可以证明香蕉分裂定律[9]，这是一个重要的程序优化，它将双重树遍历替换为单一遍历。

$$(\!|b_1|\!) \vartriangle (\!|b_2|\!) = (\!|\, b_1 \cdot \mathsf{F}\, outl \vartriangle b_2 \cdot \mathsf{F}\, outr \,|\!) : \langle \mu\mathsf{F},\, in\rangle \to \langle B_1,\, b_1\rangle \times \langle B_2,\, b_2\rangle$$

左边的双重遍历转换为右边的单一遍历。 （该定律被称为“香蕉分裂”，因为折叠括号就像香蕉一样，而 $\vartriangle$的发音是“分裂”。）该定律现在可以通过两种不同的方式进行证明：因为 $(\!|b_1|\!) \vartriangle (\!|b_2|\!))$是唯一的同态映射到乘积代数，以及因为 $(\!|b_1 \cdot \mathsf{F}\, outl \vartriangle b_2 \cdot \mathsf{F}\, outr|\!))$是从初始代数到唯一的F-代数同态映射。

练习33。形式化香蕉分裂定律的对偶（涉及展开和余积）。

□

2.6.7 余自由余代数。自由代数的对偶是余自由余代数。
在集合中， $\mathsf{U}\,(\mathsf{Cofree}\,A)$的元素是由 F确定的具有无限分支结构的树，其标签来自 A。 余代数的作用，子树 $:\mathscr{C}(\mathsf{U}\,(\mathsf{Cofree}\,A),\mathsf{F}\,(\mathsf{U}\,(\mathsf{Cofree}\,A)))$，将一棵树映射为一个 F-结构的子树。 将函子 F视为系统所有可能行为的静态描述。 此外，还有一个操作$\mathrm{label}\ :\mathscr{C}(\mathsf{U}\,(\mathsf{Cofree}\,A),A)$用于提取（根节点的）标签。 这个操作是通用的：对于每个 $f:\mathscr{C}(\mathsf{U}\,A,B)$，都存在一个 F-余代数同态$trace\,f:\mathsf{F}\text{-}\mathbf{Coalg}(A,\mathsf{Cofree}\,B)$，使得

$$f = label\cdot \mathsf{U}\,g \quad\Longleftrightarrow\quad trace\,f = g\ , \tag{84}$$

对于所有的 $g:\mathsf{F}\text{-}\mathbf{Coalg}(A,\mathsf{Cofree}\,B)$. 将$\mathrm{coalgebra}\ A$视为一种状态类型，其动作是从状态到后继状态的映射。 通用性质表达了给定起始状态的无限行为树是由状态的标记函数唯一确定的。 顺便提一下，$\mathsf{Cofree}\,A$的载体，记作 $\mathsf{F}^{\infty}\,A$，也被称为$generalised\ rose\ tree$。

举个例子，平方函子 $\mathsf{Sq}\,A = A\times A$的$\mathrm{cofree\ coalgebra}$泛化了分叉类型，将自然数类型抽象出来（参见$\mathrm{Exercise\ 43}$）。

⋆练习34.组合 U∘Cofree是函子 F的$cofree$ comonad。
探索 $\mathsf{F} = \mathsf{Id}$和 $\mathsf{F} = \mathsf{K}$的结构，其中 K是常量函子。 ⊔⊓

由于右伴随保留最终对象，我们有 $\mathsf{Cofree}\,1\cong\langle\nu\mathsf{F},\ out\rangle$并且因此 $\mathsf{F}^{\infty}\,1 = \mathsf{U}\,(\mathsf{Cofree}\,1)\cong\mathsf{U}\,\langle\nu\mathsf{F},\ out\rangle = \nu\mathsf{F}$. 换句话说，$\nu$F的元素是带有平凡标签的无限树。 相反，自由余代数可以用最终余代数表示： $\langle\mathsf{F}^{\infty}\,A,\ subtrees\rangle\cong\langle\nu\mathsf{F}_A,\ outr\cdot out\rangle$ 其中$\mathsf{F}_A\,X = A\times\mathsf{F}\,X$.

表2总结了本节讨论的伴随关系。

2.7 余内达引理

本节介绍了一个重要的范畴工具：余内达引理。它与续延传递风格相关，并引出了一个重要的证明技巧，间接证明原则[14]。

回顾一下，反变协变同态函子 $\mathscr{C}(-,X):\mathscr{C}^{\mathsf{op}}\to\mathbf{Set}$将一个箭头 $f:\mathscr{C}(B,A)$映射为一个函数 $\mathscr{C}(f,X):\mathscr{C}(A,X)\to\mathscr{C}(B,X)$。 这个函数在 X上是自然的——这是恒等式(5)的重要性。 此外，每一个类型为 $\mathscr{C}(A,-)\dot{\to}\ \mathscr{C}(B,-)$的自然变换都可以表示为 $\mathscr{C}(f,-)$的像，其中f是某个箭头。 因此，我们有以下箭头和自然变换之间的同构关系。

$$\mathscr{C}(B,A)\cong\mathscr{C}(A,-)\dot{\to}\mathscr{C}(B,-)$$

表**2.** 伴随的例子。

伴随	初始对象	终态对象	余积	积	一般余积 幂	通用产品 幂	指数到	自由代数	自由余代数
L	0	Δ	$+$	Δ	$\Sigma\, i \in I \,.\, (-)_i$	Δ	$- \times X$	自由	U
R	Δ	1	Δ	$\times$	Δ	$\Pi\, i \in I \,.\, (-)_i$	$(-)^X$	U	自由
$\lceil - \rceil$			$\triangledown$		$\bigtriangledown\ i \in I \,.\, (-)_i$		Λ°	求值	
$\lfloor - \rfloor$				$\triangle$		$\bigtriangleup\ i \in I \,.\, (-)_i$	Λ		追踪
ϵ	¡		$id \triangledown id$	$\langle outl,\, outr \rangle$			应用		标签
η		!	$\langle inl,\, inr \rangle$	$\delta = id \triangle id$	$\iota_{(-)}$	$\pi_{(-)}$		变量	

这个同构是更一般的结果的一个实例，被称为 $Yoneda$引理[8]。 设 $\mathsf{H} : \mathscr{C} \to \mathbf{Set}$ 是一个集合值函子，那么

$$\mathsf{H}\, A \cong \mathscr{C}(A, -) \dot{\to} \mathsf{H} \ . \tag{85}$$

(这个同构在 H和 A中是自然的。) 下面的箭头是Yoneda同构的证明:

$$\mathsf{y}\, s\ X = \lambda f : \mathscr{C}(A, X) \,.\, \mathsf{H}\, f\, s \qquad \text{和} \qquad \mathsf{y}^\circ\, \alpha = \alpha\, A\ id_A \ . \tag{86}$$

观察到 y只是 H的两个参数交换。 很容易看出 y°是 y的左逆。

$$\begin{aligned}
&\mathsf{y}^\circ\, (\mathsf{y}\, s) \\
&= \{\text{定义}\mathsf{y}^\circ\ (86)\text{的等式}\} \\
&\quad \mathsf{y}\text{是}A\text{的恒等函数}_A \\
&= \{\text{定义}\mathsf{y}\ (86)\text{的等式}\} \\
&\quad \mathsf{H}\text{是A的恒等函数}_A\text{的伴随函子} \\
&= \quad \{\ \mathsf{H}\ \text{函子}\ (2) \quad \} \\
&\quad s
\end{aligned}$$

对于反方向，我们利用 α的自然性，即 $\mathsf{H}\, h \cdot \alpha\, \hat{X} = \alpha\, \check{X} \cdot \mathscr{C}(A, h)$，或者以逐点的方式写为： $\mathsf{H}\, h\, (\alpha \hat{X}\ g\ \ ) = \alpha\, \check{X}\ (h \cdot g)$，其中

$h : \mathscr{C}(\hat{X}, \check{X})$和 $g : \mathscr{C}(A, \hat{X})$。

$$
\begin{aligned}
& \mathsf{y}\,(\mathsf{y}^{\circ}\,\alpha)\,X \\
= & \{\text{定义}\mathsf{y}\ (86)\text{的等式}\} \\
& \lambda f\,.\,\mathsf{H} f\,(\mathsf{y}^{\circ}\,\alpha) \\
= & \{\text{定义}\mathsf{y}^{\circ}\ (86)\text{的等式}\} \\
& \lambda f\,.\,\mathsf{H} f\,(\alpha\,A\ id_A) \\
= & \quad \{\ \alpha\text{是自然变换: } \mathsf{H}\,h\,(\alpha\hat{X}\ g\quad) = \alpha\,\check{X}\ (h \cdot g)\ \} \\
& \lambda f\,.\,\alpha\,X\,(f \cdot id_A) \\
= & \quad \{\ \text{identity}\ \} \\
& \lambda f\,.\,\alpha\,X\,f \\
= & \quad \{\ \text{extensionality—}\alpha\,X\ \text{是一个函数}\quad\} \\
& \alpha\,X
\end{aligned}
$$

此外，同构简化为 $\mathsf{y}\,g = \mathscr{C}(g, -)$作为一个快速计算显示。

$$\mathscr{C}(g, X)\,f = \mathscr{C}(g, X)$$

此外，同构简化为 $\mathsf{y}\,g = \mathscr{C}(-, g)$。

这些特殊情况引出了间接证明的原则。

$$f = g \iff \mathscr{C}(f, -) = \mathscr{C}(g, -) \tag{87}$$
$$f = g \iff \mathscr{C}(-, f) = \mathscr{C}(-, g) \tag{88}$$

我们不直接证明 f和 g的相等性，而是展示它们的Yoneda图像 $\mathsf{y}\,f$和 $\mathsf{y}\,g$的相等性。

当我们讨论指数（见第2.5节）时，我们注意到有一种将指数 B^X转化为双函子的唯一方法，从而得到双射。

$$\Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X) : \Lambda^{\circ} \tag{89}$$

也是自然的 X. 这个事实的证明基本上使用了Yoneda引理。回忆一下，如果在每个参数上都是自然的，那么在n元函子之间的变换也是自然的。 设 $p : \mathscr{C}(\check{X}, \hat{X})$, 那么

自然性条件 (89) 意味着

$$
\begin{array}{ll}
& \mathscr{C}(-, B^p) \cdot \Lambda = \Lambda \cdot \mathscr{C}(- \times p, B) \\
\Longleftrightarrow & \{\ \text{伴随性:} \quad \Lambda \cdot \Lambda^\circ = id \ \text{和} \quad \Lambda^\circ \cdot \Lambda = id \ (27)\ \} \\
& \mathscr{C}(-, B^p) = \Lambda \cdot \mathscr{C}(- \times p, B) \cdot \Lambda^\circ \\
\Longleftrightarrow & \{\ \text{Yoneda引理:} \quad \mathscr{C}(B^{\hat{X}}, B^{\check{X}}) \cong \mathscr{C}(-, B^{\hat{X}}) \dot{\to} \mathscr{C}(-, B^{\check{X}})\ (85)\ \} \\
& \mathsf{y}^\circ\,(\mathscr{C}(-, B^p)) = \mathsf{y}^\circ\,(\Lambda \cdot \mathscr{C}(- \times p, B) \cdot \Lambda^\circ) \\
\Longleftrightarrow & \{\ \mathsf{y}^\circ\text{的定义}\} \\
& \mathscr{C}(-, B^p)\, B^{\hat{X}}\, id = (\Lambda \cdot \mathscr{C}(- \times p, B) \cdot \Lambda^\circ)\, B^{\hat{X}}\, id \\
\Longleftrightarrow & \{\text{自然变换的组合}\ \} \\
& \mathscr{C}(-, B^p)\, B^{\hat{X}}\, id = (\Lambda \cdot \mathscr{C}(B^{\hat{X}} \times p, B) \cdot \Lambda^\circ)\, id \\
\Longleftrightarrow & \{\text{同态函子的定义}\ (6)\ \} \\
& B^p = \Lambda\,(\Lambda^\circ\, id \cdot (B^{\hat{X}} \times p)) \\
\Longleftrightarrow & \{\ \text{应用} \ = \Lambda^\circ\, id\ (\text{表}1) \qquad \} \\
& B^p = \Lambda\,(\text{应用} \cdot (B^{\hat{X}} \times p))\ .
\end{array}
$$

反射定律（42）意味着 B（$^-$）保留了恒等性。由于自然性条件（89）唯一确定了 B（$^-$）对箭头的作用，它还保留了组合：

$$
\begin{array}{ll}
& \mathscr{C}(-, B^{p \cdot q}) \\
= & \{\ \text{自然性条件（89）} \qquad \} \\
& \Lambda \cdot \mathscr{C}(- \times (p \cdot q), B) \cdot \Lambda^\circ \\
= & \{\ A\times\ -\text{协变函子和}\ \mathscr{C}\ (-,\ B)\ \text{逆变函子}\ \} \\
& \Lambda \cdot \mathscr{C}(- \times q, B) \cdot \mathscr{C}(- \times p, B) \cdot \Lambda^\circ \\
= & \{\ \text{自然性条件（89）} \qquad \} \\
& \Lambda \cdot \mathscr{C}(- \times q, B) \cdot \Lambda^\circ \cdot \mathscr{C}(-, B^p) \\
= & \{\text{自然性条件（89）和伴随}\ \Lambda \cdot \Lambda^\circ = id\ (\ 27)\ \} \\
& \mathscr{C}(-, B^q) \cdot \mathscr{C}(-, B^p) \\
= & \{\ \mathscr{C}(A, -)\ \text{协变函子} \qquad \} \\
& \mathscr{C}(-, B^q \cdot B^p)\ .
\end{array}
$$

应用间接证明原理（88），我们得出 $B^{p \cdot q} = B^q \cdot B^p$的结论。

练习35。将上述论证推广到具有参数的任意伴随。设 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D} \times_X$为一个双函子，为了清晰起见，写作 $\mathsf{L}\ \mathscr{X}\ A$，使得部分应用 $\mathsf{L}_X : \mathscr{C} \leftarrow \mathscr{D}$对于每个选择的 $_X :_X$，都有一个右伴随 $\mathsf{R}_X : \mathscr{C} \to \mathscr{D}$：

$$\lfloor - \rfloor : \mathscr{C}(\mathsf{L}_X\ A, B) \cong \mathscr{D}(A, \mathsf{R}_X\ B)\ 。 \tag{90}$$

1. 证明存在一种唯一的方式将 R转化为一个类型为 $\mathscr{X}^{\mathsf{op}} \times \mathscr{C} \to \mathscr{D}$的双函子，使得双射（90）在所有三个变量 A, B和 X上都是自然的。

$$\begin{array}{ccc} \mathscr{C}(\mathsf{L}_{\hat{X}}\ A, B) & \xrightarrow{\lfloor - \rfloor} & \mathscr{D}(A, \mathsf{R}_{\hat{X}}\ B) \\ \Big\downarrow & & \Big\downarrow \\ \mathscr{C}(\mathsf{L}_{p}\ A, B) & & \mathscr{D}(A, \mathsf{R}_{p}\ B) \\ \Big\downarrow & & \Big\downarrow \\ \mathscr{C}(\mathsf{L}_{\check{X}}\ A, B) & \xrightarrow[\lfloor - \rfloor]{} & \mathscr{D}(A, \mathsf{R}_{\check{X}}\ B) \end{array}$$

解释为什么 R_X在参数 X上必然是逆变的。2. 设 $\eta_X : \mathsf{Id} \dot{\to} \mathsf{R}_X \circ \mathsf{L}_X$ 为带有参数的伴随的单位。 单位 η的哪个性质对应于双射（90）在 X上的自然性？ （单位在 X上不是自然的，因为 $\mathsf{R}_X \circ \mathsf{L}_X$在 X上不是函子性的。为什么？）

□

3个算法的伴随

在这些讲义的第一部分中，我们已经看到每个基本的'数据结构'都是由一个伴随产生的。 在第二部分中，我们将注意力转向'算法'。 我们的目标是为一大类递归方程提供精确的语义-这些方程可能在Haskell程序中出现。

第二部分的组织如下。 第3.1节回顾了在第2.6节中介绍的传统折叠和展开。 我们采用了一种稍微非标准的方法，并将它们重新引入为所谓的Mendler风格方程的解决方案。
第3.2节将这些方程推广为伴随方程，并演示了许多Haskell函数都属于这个范畴。 第3.3节将伴随方程专门化为各种基本的伴随关系，并探索了由此产生的递归方案。 第3.4节发展了伴随折叠的计算属性。 与它们的普通对应物一样，它们享有反射、计算和融合定律。

对函数式编程语言Haskell [15]的一些了解是有用的，因为正式的开发与一系列的编程示例相伴而行。

3.1 不动点方程

在本节中，我们稍微介绍了数据类型的语义，但有一点不同。
以下两个Haskell程序作为运行示例。

示例*1*。数据类型 $Bush$模拟了二叉叶树，这是一种表示非空自然数序列的方式。

$$\text{数据类型}\ Bush = Leaf\ Nat \mid Fork\ (Bush, Bush)$$

类型 (A, B)是Haskell语法中的笛卡尔积 $A \times B$。

函数 $total$计算了一棵自然数的总和。

$$\begin{array}{lll} total : Bush & \to Nat \\ total\ \ (Leaf\, n) & = n \\ total\ \ (Fork(l, r)) & = total\ l + total\ r \end{array}$$

这是一个典型的 $fold$的例子，它是一个消耗数据的函数。 □

例子2.类型树捕捉分叉，无限自然数二叉树。
（分叉是将状态或动作分为两个分支的过程。）

数据树 =分支（树，自然数，树）

调用生成1 构建一个无限树，标记为从1开始的自然数。

生成 :自然数 →树
$generate\ \ n$ = 分支 (生成 $(2 * n + 0), n,$生成 $(2 * n + 1))$

这是一个典型的展开的例子，一个产生数据的函数。 □

类型 $Bush$和 $Tree$以及函数 $total$和 $generate$都是通过递归方程给出的。 一开始，这些方程是否有解以及解是否唯一都不清楚。 习惯上，将解递归方程的问题重新表述为一个不动点问题：形式为 $x = \Psi\, x$的递归方程隐式地定义了一个函数 Ψ，即递归方程的基函数。 基函数的不动点就是递归方程的解，反之亦然。

考虑定义 $Bush$的类型方程 它的基本函数或者说它的
基本函子由以下给出

数据 $\mathfrak{Bush}\ bush = \mathfrak{Leaf}\ Nat \mid \mathfrak{Fork}\,(bush, bush)$
实例 $Functor\,\mathfrak{Bush}$ **where**
$fmap\ f\ (\mathfrak{Leaf}\, n)$ =叶子节点
$fmap\ f$ (分叉 (左,右)) =分叉 (左,右) .

我们采用的约定是将基本函子命名为底层类型的名称，使用此字体表示前者，使用此字体表示后者。 $\mathfrak{B}$ush的类型参数标记了递归组件。 在Haskell中，functor的对象部分由data声明定义；箭头部分由Functor实例给出。

使用算术符号布什被写作布什 $B = Nat + B \times B$。

所有基于一阶数据类型声明（和积的和，无函数类型）的函子都有两个极端不动点：初始F-代数$\langle \mu$F，tin$\rangle$和最终F-余代数$\langle \nu$F，tout$\rangle$，其中F:$\mathscr{C} \to \mathscr{C}$是所讨论的函子（第2.6节）。一些编程语言，如$\mathscr{C}$harity [16]或$\mathscr{C}$oq [17]，允许用户在初始解和最终解之间进行选择-数据类型声明被标记为归纳或余归纳。 Haskell不是其中之一。 由于Haskell的基础范畴是S$\mathscr{C}$po，即完全偏序范畴和严格连续函数的范畴

并且初始代数和最终余代数实际上是相同的[18, 19] - 进一步的背景知识在本节末尾提供。 相比之下，在集合**S**et中，归纳类型的元素是有限的，而余归纳类型的元素可能是无限的。 从操作上看，归纳类型的元素可以在有限的步骤序列中构造，而余归纳类型的元素允许任意有限的观察序列。

转向我们的运行示例，我们将布什视为一个初始代数 - 尽管归纳和余归纳树都同样有用。 对于分叉，只有余归纳的解读是有用的，因为在集合中，树的基本函子的初始代数是空集。

定义*3.*在Haskell中，初始代数和最终余代数可以定义如下。

$$\begin{array}{ll} \text{新类型} & \mu f = In \quad \{ in^\circ : f\,(\mu f)\} \\ \text{新类型} & \nu f = Out^\circ\,\{ out : f\,(\nu f)\} \end{array}$$

这些定义使用Haskell的记录语法引入了解构器 in°和out 除了构造器 In 和 Out° 之外。 新类型声明确保 $\mu\ f$和$f\,(\mu\ f)$在运行时共享相同的表示，对于 $\nu\,f$和$f\,(\nu\ f)$也是如此。 换句话说，构造器和解构器是无操作的。 当然，由于在Haskell中初始代数和最终余代数重合，它们可以通过单个 **newtype**定义来定义。 然而，由于我们希望将Haskell用作集合的元语言，我们将它们保持分开。

□

为了实现对总体的语义，让我们首先将其定义调整为新的“两级类型” μ布什。 这个术语归功于[20]；一个级别描述数据的结构，另一个级别将递归节点联系在一起。

$$\begin{array}{lll} \text{总体} : \mu\text{布什} & \to \text{自然数总体}\ (\text{内部}\ (\text{叶子}\ n)) \\ & = n\text{总体}\ (\text{内部}\ (\text{分支}\ (\text{左,右}))) = \text{总体}\ l + \\ \text{总体}\ r \end{array}$$

现在，如果我们抽象出递归调用，我们得到一个非递归基本函数，类型为 (μ布什 →自然数)→(μ布什 →自然数)。 与函子一样，我们采用的约定是基本函数以底层函数命名，前者使用此字体，后者使用此字体。

$$\begin{array}{lll} \text{总体} : (\mu\text{布什} \to \text{自然数}) \to (\mu\text{布什} & \to \text{自然数})\text{总 总} \\ (\text{在}\ (\text{叶子节点}\ n)) & = n\text{总 总} \\ (\text{在}\ (\text{分支节点}\ (\text{左,右}))) & = \text{总 左} + \text{总 右} \end{array}$$

这种类型的函数可能有很多固定点-考虑一个极端的例子，即恒等基函数，它有无限多个固定点。
有趣的是，如果我们额外删除构造函数在，歧义的问题就会消失。

$$\begin{array}{lll} \text{总}: \forall\,\text{变量}x\ .\ (\text{变量}x \to \text{自然数}) \to (\text{树状结构}x & \to \text{自然数}) \\ \text{总} \quad \text{总} & (\text{叶子节点}\ n) & = n \\ \text{总} \quad \text{总} & (\text{分支节点}\ (\text{左,右})) & = \text{总 左} + \text{总 右} \end{array}$$

基函数的类型在递归调用的参数中变成了多态的。 我们将在下一节中展示，这种类型保证了总

$$\text{总} : \mu\text{树状结构} \to \text{自然数总} \quad (\text{在 } \mathfrak{s}) = \text{总总 } \mathfrak{s}$$

在方程中，它在某种意义上是良定义的，因为方程只有一个解。
对类型 $Tree$ 和函数
$generate$应用类似的转换，我们得到

$$\begin{aligned}
&\text{数据 } \mathfrak{Tree}\ tree = \mathfrak{Branch}\ (tree, Nat, tree)\\
&\mathfrak{generate} : \forall x\ .\ (Nat \to x) \to (Nat \to \mathfrak{Tree}\ x)\\
&\mathfrak{generate}\qquad generate\qquad n\\
&\quad = \mathfrak{Branch}\ (generate\ (2 * n + 0), n, generate\ (2 * n + 1))\\
&generate : Nat \to \nu\mathfrak{Tree}\\
&generate\quad n\quad = Out^{\circ}\ (\mathfrak{generate}\ generate\ n)\ .
\end{aligned}$$

再次，基本函数具有多态类型，保证了递归函数的良定义性。

抽象出语法的具体细节，这些例子表明了对形式为固定点方程的考虑。

$$x \cdot in = \Psi\, x, \qquad \text{并且对偶地} \qquad out \cdot x = \Psi\, x\ , \tag{91}$$

其中未知变量 x的类型在左边是 $\mathscr{C}(\mu\mathsf{F}, A)$，在右边是 $\mathscr{C}(A, \nu\mathsf{G})$。
上述的Haskell定义是这些方程的逐点版本： $x\ (In\ a) = \Psi x\ a$和$x\ a = Out^{\circ}\ (\Psi x\ a)$。
通过这种形式定义的箭头被称为Mendler风格的折叠和展开，因为它们最初是在类型理论的背景下由Mendler [21]引入的。 通常我们会省略限定词，简称解决方案为折叠和展开。 实际上，语言的滥用是合理的，因为每个Mendler风格的方程都等价于标准（不）折叠的定义方程。 接下来我们将首先考虑折叠。

3.1.1 初始不动点方程。让 $\mathscr{C}$是一些基本类别，让
$\mathsf{F} : \mathscr{C} \to \mathscr{C}$是一些自函子。 一个未知数的初始不动点方程在
$x : \mathscr{C}(\mu\mathsf{F}, A)$ 具有以下句法形式

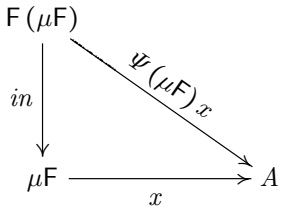
$$x \cdot in = \Psi\, x\ , \tag{92}$$

其中基本函数 Ψ的类型为

$$\mathscr{C}(X, A) \to \mathscr{C}(\mathsf{F}X, A)\ .$$

在不动点方程 (92) 中，自然变换 Ψ被实例化为
初始代数： $x \cdot in = \Psi\ (\mu\mathsf{F})\ x$. 为了可读性，我们通常省略

自然变换的‘类型参数’。 下面的图显示了所涉及的类型。



自然性条件可以看作是受保护的析构条件 [22] 的语义对应物。 如果我们将等式 $in : \mathsf{F}\,(\mu\mathsf{F}) \cong \mu\mathsf{F}$中的等价关系移动到右侧，则可以看到这一点： $x = \Psi\, x \cdot in^{\circ}$. 这里 in°是保护递归调用的析构函数。 该方程具有直观的操作解读。 对 x的参数进行析构，得到一个类型为$\mathsf{F}\,(\mu\mathsf{F})$的元素。 然后，基本函数 Ψ对 F-结构进行操作，可能将其第一个参数，即 x的递归调用，应用于类型为 $\mu\mathsf{F}$的元素。 这些元素是原始参数的适当子项—请记住，F的类型参数标记了递归组件。 Ψ的自然性确保仅这些子项可以传递给递归调用。

这是否意味着 x是终止的？ 终止是一个操作性概念；如何将这个概念转化为一个指示性设置取决于底层范畴。 我们的主要目标是证明方程（92）有一个唯一解。
在 **Set**中工作时，这个结果意味着方程有一个解，而且是一个总函数。 此外， $x = \Psi\, x \cdot in^{\circ}$的操作性解读表明， x是终止的，因为归纳类型的元素只能被有限次地析构。 （根据求值策略，这个论断也受到 F-结构本身是有限的条件的限制。）另一方面，如果底层范畴是 **SCpo**，那么解是一个连续函数，它不一定对所有输入都终止，因为 **SCpo**中的初始代数可能包含无限元素。

尽管总的定义很好地适应了上述框架，但以下程序却没有。

例子*4*。自然性条件是充分但不必要的，二进制增量的例子证明了这一点。

$$
\begin{aligned}
&\text{数据自然数} = N \mid O\ \text{自然数} \mid I\ \text{自然数}\\
&succ : \text{自然数} \to \text{自然数}\\
&succ\ \ (N) = I\ N\\
&succ\ \ (O\ b) = I\ b\\
&succ\ \ (I\ \ b) = O\,(succ\ b)
\end{aligned}
$$

与总的情况一样，我们将数据类型分为两个层次。

$$
\begin{aligned}
&\text{类型自然数} = \mu\text{自然数}\\
&\text{数据自然数}\ nat = \mathfrak{N} \mid \mathfrak{O}\ nat \mid \mathfrak{I}\ nat
\end{aligned}
$$

$$
\begin{array}{l}
\text{实例函子自然数 } \mathbf{where} \\
\quad \mathit{fmap}\ f\ (\mathfrak{N}) \quad = \mathfrak{N} \\
\quad \mathit{fmap}\ f\ (\mathfrak{O}\ b) = \mathfrak{O}\ (f\ b) \\
\quad \mathit{fmap}\ f\ (\mathfrak{I}\ \ b) = \mathfrak{I}\ \ (f\ b)
\end{array}
$$

在集合中，继承函数的实现明显是终止的。 然而，相关的基本函数

$$
\begin{array}{lll}
\mathfrak{succ} : (\text{自然数} \to \text{自然数}) \to (\text{自然数自然数} \to \text{自然数}) & & \\
\mathfrak{succ}\ \ \mathit{succ} & (\mathfrak{N}) & = \text{在}\ (\mathfrak{I}\ (\text{在}\ \mathfrak{N})) \\
\mathfrak{succ}\ \ \mathit{succ} & (\mathfrak{O}\ b) & = \text{在}\ (\mathfrak{I}\ b) \\
\mathfrak{succ}\ \ \mathit{succ} & (\mathfrak{I}\ \ b) & = \text{在}\ (\mathfrak{O}\ (\mathit{succ}\ b))
\end{array}
$$

缺乏自然性。 从某种意义上说，它的类型太具体了，因为它揭示了递归调用传递了一个二进制数。 对手可以利用这些信息，将终止的程序变成非终止的程序：

$$
\begin{array}{lll}
\mathfrak{bogus} : (\text{自然数} \to \text{自然数}) \to (\text{自然数自然数} \to \text{自然数}) & & \\
\mathfrak{bogus}\ \ \mathit{succ} & (\mathfrak{N}) & = \text{在}\ (\mathfrak{I}\ (\text{在}\ \mathfrak{N})) \\
\mathfrak{bogus}\ \ \mathit{succ} & (\mathfrak{O}\ b) & = \text{在}\ (\mathfrak{I}\ b) \\
\mathfrak{bogus}\ \ \mathit{succ} & (\mathfrak{I}\ \ b) & = \text{后继}\ (\text{在}\ (\mathfrak{I}\ b))\ .
\end{array}
$$

我们将在第3.3.2节（例子19）中回到这个例子。 □

转向唯一性的证明，让我们详细说明基本函数 Ψ的自然性质。 使用同态函子的定义（6），这展开为

$$\Psi\, f \cdot \mathsf{F}\, h = \Psi\, (f \cdot h)\ , \tag{93}$$

对于所有箭头 $f : \mathscr{C}(X_2, A)$. 这个性质特别意味着 Ψ完全由其对id 的映射确定，即 $\Psi\, h = \Psi\ \mathit{id} \cdot \mathsf{F}\, h$. 现在，为了证明方程 $x \cdot \mathit{in} = \Psi\, x$(92)有一个唯一的解，我们证明如果且仅如果 x是一个标准折叠，那么 x是一个解。

$$
\begin{array}{ll}
 & x \cdot \text{在} = \Psi\ x \\
\Longleftrightarrow & \{\ \Psi\ \text{是自然的}\ (93)\ \} \\
 & x \cdot \text{在} = \Psi\ \mathit{id} \cdot \mathsf{F}\, x \\
\Longleftrightarrow & \{\ \text{标准折叠的唯一性属性}\ (54)\ \} \\
 & x = (\!|\Psi\ \mathit{id}|\!)
\end{array}
$$

通过重载香蕉括号，固定点方程的唯一解
$x \cdot \text{在} = \Psi\ x$(92) 被写作 $(\!|\Psi|\!)$.

让我们更深入地探索标准折叠和Mendler风格折叠之间的关系。 上面的证明依赖于 Ψ的类型与 $\mathscr{C}(\mathsf{F}\, A, A)$，即 F-代数的类型同构。

$$\mathscr{C}(\mathsf{F}\, A, A) \cong (\forall X : \mathscr{C}\ .\ \mathscr{C}(X, A) \to \mathscr{C}(\mathsf{F}\, X, A))\ .$$

箭头和自然变换之间的这种双射是Yoneda引理(第2.7节)的一个实例，其中反变函子 $\mathsf{H} : \mathscr{C}^{\mathsf{op}} \to \mathbf{Set}$由 $\mathsf{H} = \mathscr{C}(\mathsf{F}\,-, A)$给出。 因此，Mendler风格的折叠和标准折叠通过 $(\Psi)) = (\mathsf{y}^{\circ}\,\Psi)) = (\Psi\ id))$和 $(\lambda x\ .\ a \cdot \mathsf{F}\ x)) = (\mathsf{y}\ a)) = (a))$相关。示例*1*。计算自然数bush的总和的标准折叠是 $(id \triangledown plus))$，参见第2.6.*1*节。以逐点

方式编写的代数$id \triangledown plus$为

$$\begin{array}{ll} \mathsf{total} : \mathfrak{Bush}\ Nat & \to Nat \\ \mathsf{total}\ \ (\mathfrak{Leaf}\ n) & = n \\ \text{总和}\ \ (\text{分叉}\,(\text{左},\text{右})) & = \text{左} + \text{右}\ . \end{array}$$

代数和基本函数之间的关系是总和 =总和 id和总和总和 =总和 $\cdot fmap$ 总和，这意味着 (总和)) =(总和)) . □

3.1.2 最终不动点方程。前一节的发展对偶到最终余代数。 为了完整起见，让我们详细说明细节。

未知数 $x : \mathscr{C}(\mathrm{A},\ \nu\mathsf{G})$的最终不动点方程具有以下形式

$$out \cdot x = \Psi\ x\ \ , \tag{94}$$

其中基本函数 Ψ的类型为

$$\mathscr{C}(A,X) \to \mathscr{C}(A, \mathsf{G}\ X)\ \ .$$

通过重载镜头括号，(94)的唯一解记为 $[(\Psi])$。在 **Set**中，自然性条件捕

捉到了由构造函数保护的条件 [22]，确保了生产力。如果我们将同构移动到右侧，这一点可以更清楚地看到： $x = out^{\circ} \cdot \Psi\ x$。这里 out°是保护递归调用的构造函数。基本函数 Ψ必须生成一个$\mathsf{G}(\ \nu\ \mathsf{G})$结构。为了创建类型为$\nu$G的递归组件，基本函数 Ψ可以使用它的第一个参数，即 x的递归调用。然而， Ψ的自然性保证这些调用只能在受保护的位置进行。

Ψ的类型与$\mathscr{C}$（A，$\mathsf{G}\ A$）的同构，即G-余代数的类型。

$$\mathscr{C}\ (A,\ \mathsf{G}\ A) \cong (\forall X{:}\mathscr{C}.\ \mathscr{C}\ (A,\ X) \to \mathscr{C}\ (A,\ \mathsf{G}\ X))\ \text{。}$$

再次，这是Yoneda引理的一个实例：现在$\mathsf{H} = \mathscr{C}$（A，G-）是一个协变函子$\mathsf{H}{:}\mathscr{D} \to \mathbf{S}\mathrm{et}$。

例子*2*。构造一个无限自然数树的标准展开是$[(\mathrm{shift}\,0 \vartriangle \mathrm{id} \vartriangle \mathrm{shift}\,1)]$，参见第*2*.6.*2*节。以逐点方式写的余代数$\mathrm{shift}\,0 \vartriangle \mathrm{id} \vartriangle \mathrm{shift}\,1$读作$generate{:}Nat \to \mathrm{Tree}\ Nat$。$generate\ n$

$$= \text{分支}\quad (2 * n + 0,\ n,\ 2 * n + 1)\ \ .$$

共代数和基础函数通过生成 =生成 id 和
生成 $gen = fmap\ gen \cdot$生成 相关，这意味着 [(生成)] = [(生成)]. □

在接下来的章节中，我们将展示定点方程是非常通用的。
比起一开始可能想的更多函数适用于这个范围。

3.1.3 互相类型递归。在Haskell中，数据类型可以通过互相递归来定义。

例子5。想象一个简单的命令式编程语言。 表达式和语句的抽象语法通常通过互相类型递归来定义（这是一个非常简化的例子）。

$$\begin{aligned}&\text{数据表达式 } = \text{变量 变量 } | \text{块 (语句,表达式)}\\&\text{数据语句 } = \text{赋值 (变量,表达式) } | \text{序列 (语句,语句)}\end{aligned}$$

由于函数遵循形式，消耗抽象语法树的函数通常通过相互值递归来定义。

$$\begin{aligned}&\text{类型变量 } = \text{集合变量}\\&\text{变量表达式 : 表达式} \quad \to \text{变量}\\&\text{变量表达式 } (\text{变量 } x) \quad = \{x\}\\&\text{变量表达式 } (\text{块 } (s, e)) \; = \text{语句变量 } s \cup \text{变量表达式 } e\\&\text{语句变量 : 语句} \quad \to \text{变量}\\&\text{语句变量 } (\text{赋值 } (x, e)) \; = \; \{x\} \cup \text{变量表达式 } e\\&\text{语句变量 } (\text{序列 } (s_1, s_2)) \; = \text{语句变量 } s_1 \cup \text{语句变量 } s_2\end{aligned}$$

这些函数确定表达式或语句的变量，假设存在适当的集合类型集合，具有操作 $\{-\}$和 $\cup$。 □

我们能否将上述定义适应前一节的框架中？ 是的，我们只需要选择一个合适的基础范畴：在这种情况下，是一个乘积范畴（第2.2.2节）。表达式和语句的基础函子是一个自函子，作用于乘积范畴上：

$$\text{语法} \qquad \langle A,\, B\rangle = \langle\, Var + B \times A,\; Var \times A + B \times B\rangle \;\; .$$

Haskell类型Expr 和 $Stat$然后成为定点的组成部分:μGrammar $= \langle\; Expr,\, Stat\rangle$.
函数 $v$ars$E$xpr 和 $v$ars$S$tat 相应地处理：我们将它们捆绑成一个单一的箭头

$$vars = \langle varsExpr,\, varsStat\rangle : (\mathscr{C} \times \mathscr{C})(\mu\mathsf{Grammar}, \langle\, Vars,\; Vars\rangle) \;\; .$$

下面的计算明确说明了初始的定点方程
在 $\mathscr{C} \times \mathscr{D}$中对应于两个方程，一个在 $\mathscr{C}$ 中，一个在 $\mathscr{D}$中。

$$\begin{aligned}&x \cdot in = \Psi\, x : (\mathscr{C} \times \mathscr{D})(\mathsf{F}\,(\mu\mathsf{F}), \langle A_1,\, A_2\rangle)\\\Longleftrightarrow\;\; &\{\text{ surjective pairing: } f = \langle \mathsf{Outl}\, f,\, \mathsf{Outr}\, f\rangle \;\}\\&\langle \mathsf{Outl}\, x,\, \mathsf{Outr}\, x\rangle \cdot \langle \mathsf{Outl}\, in,\, \mathsf{Outr}\, in\rangle = \Psi\, \langle \mathsf{Outl}\, x,\, \mathsf{Outr}\, x\rangle\\\Longleftrightarrow\;\; &\{\text{ set } x_1 = \mathsf{Outl}\, x,\; x_2 = \mathsf{Outr}\, x \text{ and } in_1 = \mathsf{Outl}\, in,\; in_2 = \mathsf{Outr}\, in \;\}\\&\langle x_1,\, x_2\rangle \cdot \langle in_1,\, in_2\rangle = \Psi\, \langle x_1,\, x_2\rangle\\\Longleftrightarrow\;\; &\{\text{ definition of composition in } \mathscr{C} \times \mathscr{D} \;\}\\&\langle x_1 \cdot in_1,\, x_2 \cdot in_2\rangle = \Psi\, \langle x_1,\, x_2\rangle\\\Longleftrightarrow\;\; &\{\text{ surjective pairing: } f = \langle \mathsf{Outl}\, f,\, \mathsf{Outr}\, f\rangle \;\}\\&\langle x_1 \cdot in_1,\, x_2 \cdot in_2\rangle = \langle \mathsf{Outl}\, (\Psi\, \langle x_1,\, x_2\rangle),\, \mathsf{Outr}\, (\Psi\, \langle x_1,\, x_2\rangle)\rangle\end{aligned}$$

$$
\begin{aligned}
&\Longleftrightarrow \quad \{\text{集合 } \Psi = \mathsf{Outl} \circ \Psi \text{ 和 } \quad \Psi_2 = \mathsf{Outr} \circ \Psi \ \} \\
&\qquad \langle x_1 \cdot \text{在}_1,\ x_2 \cdot \text{在}_2 \rangle = \langle \Psi_1 \langle x_1,\ x_2 \rangle,\ \Psi_2 \langle x_1,\ x_2 \rangle \rangle \\
&\Longleftrightarrow \quad \{\text{在 } \mathscr{C} \times \mathscr{D} \text{中的箭头相等 } \} \\
&\qquad x_1 \cdot \text{在}_1 = \Psi_1 \langle x_1,\ x_2 \rangle : \mathscr{C}(\mathsf{Outl}\,(\mathsf{F}\,(\mu \mathsf{F})), A_1) \text{ 和} \\
&\qquad\quad x_2 \cdot in_2 = \Psi_2 \langle x_1,\ x_2 \rangle : \mathscr{D}(\mathsf{Outr}\,(\mathsf{F}\,(\mu \mathsf{F})), A_2)
\end{aligned}
$$

基本函数 Ψ_1和 Ψ_2都是参数化的，参数分别为 x_1和 x_2。 除此之外，语法形式与标准的固定点方程相同。

将例子5的方程式转化为这种形式是一个简单的练习：

定义6。在Haskell中，互递归类型可以如下建模。

$$
\begin{aligned}
&\text{新类型} \quad \mu_1\, f_1\, f_2 = In_1\, \{\, in_1^\circ : f_1\, (\mu_1\, f_1\, f_2)\ (\mu_2\, f_1\, f_2) \}\\
&\text{新类型} \quad \mu_2\, f_1\, f_2 = In_2\, \{\, in_2^\circ : f_2\, (\mu_1\, f_1\, f_2)\ (\mu_2\, f_1\, f_2) \}
\end{aligned}
$$

由于Haskell在类型层面上没有对偶的概念，也就是说，没有乘积种类，我们必须对类型构造函数进行柯里化： $\mu_1\, f_1\, f_2 = \mathsf{Outl}\,(\mu\langle f_1,\, f_2\rangle)$ 和 $\mu_2\, f_1\, f_2 = \mathsf{Outr}\,(\mu\langle f_1,\, f_2\rangle)$。 □

例子7。表达式和语句的基本函子是

$$
\begin{aligned}
&\text{数据表达式表达式语句 } = \text{变量变量 } | \text{块}\,(\text{语句},\text{表达式})\\
&\text{数据语句表达式语句 } = \text{赋值}\,(\text{变量},\text{表达式}) \ | \text{序列}\,(\text{语句},\text{语句}) \quad .
\end{aligned}
$$

由于所有的Haskell函数都属于同一个范畴，我们必须用范畴 $\mathscr{C}$中的箭头对来表示 $\mathscr{C} \times \mathscr{C}$中的箭头对。

$$
\begin{aligned}
&\mathfrak{varsExpr} : \forall\, x_1\ x_2\ .\\
&\qquad (x_1 \to Vars, x_2 \to Vars) \to (\mathfrak{Expr}\, x_1\ x_2 \quad \to Vars)\\
&\mathfrak{varsExpr}\ \ (varsExpr,\ varsStat) \quad (\mathfrak{Var}\, x) \quad = \{x\}\\
&\mathfrak{varsExpr}\ \ (varsExpr,\ varsStat) \quad (\mathfrak{Block}\,(s, e)) = varsStat\ s \cup varsExpr\ e\\
&\mathfrak{varsStat} : \forall\, x_1\ x_2\ .\\
&\qquad (x_1 \to Vars, x_2 \to Vars) \to (\mathfrak{Stat}\, x_1\ x_2 \quad \to Vars)\\
&\mathfrak{varsStat}\ \ (varsExpr,\ varsStat) \quad (\mathfrak{Assign}\,(x, e)) = \{x\} \cup varsExpr\ e\\
&\mathfrak{varsStat}\ \ (varsExpr,\ varsStat) \quad (\mathfrak{Seq}\,(s_1, s_2)) \quad = varsStat\ s_1 \cup varsStat\ s_2
\end{aligned}
$$

变量表达式和变量语句的定义与上述方案完全匹配。

$$
\begin{aligned}
&\text{变量表达式:}\mu_1\text{表达式语句}\to\text{变量}\\
&\text{变量表达式}\ (In_1\ e) \quad =\text{变量表达式}\ (\text{变量表达式}, \text{变量语句})\ e\\
&\text{变量语句:}\mu_2\text{表达式语句}\to\text{变量}\\
&\text{变量语句}\ (In_2\ s) \quad =\text{变量语句}\ (\text{变量表达式}, \text{变量语句})\ s
\end{aligned}
$$

由于这两个方程等价于一个在 $\mathscr{C}\times\mathscr{C}$中的初始不动点方程，它们确实有唯一解。

□

处理相互递归的数据类型和相互递归函数不需要新的理论。 通过对偶性，对于最终余代数也是如此。
对于最终不动点方程，我们有以下对应关系。

$$out \cdot x = \Psi\ x \quad \Longleftrightarrow \quad out_1 \cdot x_1 = \Psi_1 \langle x_1, x_2 \rangle \text{ 和 } \quad out_2 \cdot x_2 = \Psi_2 \langle x_1, x_2 \rangle$$

3.1.4 类型函子。在Haskell中，数据类型可以通过类型进行参数化。

例子*8*。随机访问列表的类型[23]由以下给出

$$\begin{array}{l}
\text{数据数组}\ a = \text{空} \mid \text{零}\ (\text{数组}\ (a, a)) \mid \text{一}\ (a, \text{数组}\ (a, a)) \\
\text{实例} \qquad \text{函子} \qquad \text{数组}\ \textbf{where} \\
\quad fmap\ f\ (\text{空}\quad) \qquad = \text{空} \\
\quad fmap\ f\ (\text{零}\ s\quad) \qquad = \text{零}\quad (fmap\ (f \times f)\ s) \\
\quad fmap\ f\ (\text{一}\quad (a, s)) = \text{一}\quad (f\ a, fmap\ (f \times f)\ s) \\
(\times) : (\hat{a} \to \check{a}) \to (\hat{b} \to \check{b}) \to ((\hat{a}, \hat{b}) \to (\check{a}, \check{b})) \\
(f \times g)\ (a, b) = (f\ a, g\ b)\ .
\end{array}$$

类型 Array是所谓的嵌套数据类型[24]，因为类型参数在递归调用中从 a变为 (a, a)。 随机访问列表是一种数值表示，一种模拟数字系统的容器类型，这里是二进制数。

$$\begin{array}{lll}
size : \forall\, a\ .\ \mathsf{Array}\ a & \to Nat \\
size \quad (Null) & = 0 \\
size \quad (Zero\ s) & = 2 * size\ s + 0 \\
size \quad (One\ (a, s)) & = 2 * size\ s + 1
\end{array}$$

函数 $size$计算随机访问列表的大小，说明了随机访问列表和二进制数之间的对应关系。 定义需要多态递归[25]，因为递归调用的类型为 $\mathsf{Array}\ (a, a) \to Nat$，这是声明类型的一个替代实例。

□

我们能将上述定义放入3.1.1节的框架中吗？ 再次，答案是肯定的。我们只需要选择一个合适的基础范畴：这次是一个函子范畴（2.2.3节）。Array的基础函子是一个函子范畴中的自函子：

$$\mathfrak{Array}\ \mathsf{F}\ A = 1 + \mathsf{F}\ (A \times A) + A \times \mathsf{F}\ (A \times A)\ .$$

二阶函子$\mathfrak{Array}$将一个函子映射到另一个函子。 由于它的不动点$\mathfrak{Array} = \mu\ \mathfrak{Array}$存在于一个函子范畴中，对于随机访问列表的折叠必然是自然变换。 函数size是一个自然变换，因为我们可以给它赋予以下类型：

$$size : \mu\mathfrak{Array} \dot{\to} \mathsf{K}\ Nat\ ,$$

其中 $\mathsf{K} : \mathscr{D} \to \mathscr{D}^{\mathscr{C}}$ 是一个常量函子，定义为 $\mathsf{K}A\ B = A$. 同样，我们可以在Haskell中重复这个发展过程。

定义*9.*第二阶初代代数和最终余代数的定义与定义3相同，只是多了一个类型参数。

$$\begin{array}{ll} \text{新类型} & \mu f\ a = In \quad \{ in^\circ : f\ (\mu f)\ a \} \\ \text{新类型} & \nu f\ a = Out^\circ\ \{ out : f\ (\nu f)\ a \} \end{array}$$

为了捕捉到 μf 和 νf 都是函子的事实，只要 f 是一个二阶函子，我们就需要对Haskell 2010类系统进行扩展。

$$\begin{array}{l} \text{实例} \quad (\forall x\ .\ (Functor\ x) \Rightarrow Functor\ (f\ x)) \Rightarrow Functor\ (\mu f)\ \mathbf{where} \\ \quad fmap\ f\ (In \quad s) = In \quad (fmap\ f\ s) \\ \text{实例} \quad (\forall x\ .\ (\text{函子}\ x \quad) \Rightarrow \text{函子} \quad (f\ x)) \Rightarrow \text{函子} \quad (\nu f)\ \text{其中} \\ \quad fmap\ f\ (Out^\circ\ s) = Out^\circ\ (fmap\ f\ s) \end{array}$$

这些声明使用所谓的多态谓词[26]，精确地捕捉了 *f*将函子映射到函子的要求。不幸的是，这个扩展还没有被实现。它可以在Haskell 2010中模拟[27]，但是生成的代码有些笨拙。 或者，可以使用“递归字典”

$$\begin{array}{l} \text{实例} \quad \text{函子} \quad (f\ (\mu f)) \Rightarrow \text{函子} \quad (\mu f)\ \text{其中} \\ \quad fmap\ f\ (In \quad s) = In \quad (fmap\ f\ s) \\ \text{实例} \quad \text{函子} \quad (f\ (\nu f)) \Rightarrow \text{函子} \quad (\nu f)\ \text{其中} \\ \quad fmap\ f\ (Out^\circ\ s) = Out^\circ\ (fmap\ f\ s) \end{array}$$

并依赖于编译器来解决递归结构 [28]。 □

让我们将定点方程特化为函子范畴。

$$\begin{array}{ll} & x \cdot in = \Psi\ x \\ \Longleftrightarrow & \{\ \mathscr{D}^{\mathscr{C}}\text{中箭头的等式}\} \\ & \forall A : \mathscr{C}\ .\ (x \cdot in)\ A = \Psi\ x\ A \\ \Longleftrightarrow & \{\ \mathscr{D}^{\mathscr{C}}\text{中组合的定义}\} \\ & \forall A : \mathscr{C}\ .\ x\ A \cdot in\ A = \Psi\ x\ A \end{array}$$

在Haskell中，类型应用是不可见的，因此在函子范畴中的定点方程无法与基本范畴中的方程区分开来。

例子*10*。继续例子8， Array的基本函子将函子映射到函子：它的类型为 $(\star \to \star) \to (\star \to \star)$。

$$\begin{array}{l} \mathbf{data}\ \mathfrak{Array}\ array\ a = \mathfrak{Null} \mid \mathfrak{Zero}\ (array\ (a, a)) \mid \mathfrak{One}\ (a, array\ (a, a)) \\ \mathbf{instance}\ (Functor\ array) \Rightarrow Functor\ (\mathfrak{Array}\ array)\ \mathbf{where} \\ \quad fmap\ f\ (\mathfrak{Null}) \quad = \text{空} \\ \quad fmap\ f\ (\text{零} \quad s) \quad = \text{零} \quad (fmap\ (f \times f)\ s) \\ \quad fmap\ f\ (\text{一} \quad (a, s)) = \text{一} \quad (f\ a, fmap\ (f \times f)\ s) \end{array}$$

表格 **3.**不同类别中的初始代数和最终余代数。

类别	初始不动点方程 $x \cdot in = \Psi\, x$	最终不动点方程 $out \cdot x = \Psi\, x$
集合	归纳类型 标准折叠	余归纳类型 标准展开
Cpo	—	连续余代数（域） 连续展开 (F在 **SCpo**中局部连续)
SCpo	连续代数（域） 严格连续折叠 (F在 **SCpo**中局部连续, $\mu\mathsf{F} \cong \nu\mathsf{F}$)	连续余代数（域） 严格连续展开
$\mathscr{C} \times \mathscr{D}$	相互递归归纳类型相互递归余归纳类型 相互递归折叠 $x_1 \cdot in_1 = \Psi_1\, \langle x_1, x_2 \rangle$ $x_2 \cdot in_2 = \Psi_2\, \langle x_1, x_2 \rangle$	相互递归展开 $out_1 \cdot x_1 = \Psi_1\, \langle x_1, x_2 \rangle$ $out_2 \cdot x_2 = \Psi_2\, \langle x_1, x_2 \rangle$
$\mathscr{D}^{\mathscr{C}}$	归纳类型函子 高阶折叠 $x\ A \cdot in\ A = \Psi\, x\ A$	余归纳类型函子 高阶展开 $out\ A \cdot x\ A = \Psi\, x\ A$

它对箭头的作用，如上所示，将自然变换映射到自然变换。 因此， $size$的基本函数是一个二阶自然变换，它将自然变换映射到自然变换。

$$
\begin{array}{lll}
\mathfrak{size} : \forall x\,.\,(\forall a\,.\,x\ a \to Nat) \to (\forall a\,.\,\mathfrak{Array}\ x\ a & \to \text{自然数}) \\
\mathfrak{size} \quad size & (\mathfrak{Null}) & = 0 \\
\mathfrak{size} \quad size & (\text{零}\quad s) & = 2 * \text{大小}\ s + 0 \\
\mathfrak{size} \quad size & (\text{一个}\,(a, s)) & = 2 * \text{大小}\ s + 1 \\
\text{大小}\colon \forall a.\ \mu\text{数组}\ a \to \text{自然数} & & \\
\text{大小} \quad (\text{在}\ s) & = \text{大小大小}\ s &
\end{array}
$$

得到的方程符合初始不动点方程的模式（在Haskell中类型应用是不可见的）。 因此，它有唯一的解。 □

表3总结了我们迄今为止的发现。为了提供一些背景，**C**po是完全偏序和连续函数的范畴；**SC**po是严格函数的全子范畴。 如果一个函子F:**SC**po→**SC**po在箭头**SC**po(A,B)→**SC**po(F(A),F(B))上的作用是连续的，那么它被称为局部连续的。对于任意的对象A和B。 连续代数只是一个其载体是完全偏序且作用是连续函数的代数。 在**SC**po中，每个局部连续的函子都有一个初始代数，并且初始代数与最终余代数相一致。 这就是为什么**SC**po通常被认为是Haskell的环境范畴的原因。 起初，懒惰程序被严格函数建模可能看起来有些奇怪。 然而，非严格函数与从提升域到严格函数的一一对应关系：**SC**po($A_\perp$,B) ≅ **C**po(A,B)。（换句话说，我们有一个伴随

$(-)_\perp \dashv$ Incl在提升和包含函子之间 Incl : **SCpo** → **Cpo**之间。提升的语义概念是添加一个新的最小元素，它模拟了操作概念中的thunk（也称为closure、laze或recipe）.

3.2 伴随的不动点方程

我们在前一节中已经看到，初始和最终的不动点方程是非常通用的。然而，显然有很多定义不符合这个模式。 我们在引言中提到了列表的连接和其他内容。

例子*11.*数据类型 $Stack$表示自然数的堆栈。

$$\mathbf{data}\, Stack = Empty \mid Push\,(Nat, Stack)$$

函数 cat 将两个堆栈连接起来。

$$\begin{aligned} &cat : (Stack, && Stack) \to Stack \\ &cat\ \ (Empty, && ns) \ \ = ns \\ &cat\ \ (Push\,(m, ms), ns) && = Push\,(m, cat\,(ms, ns)) \end{aligned}$$

这个定义不符合初始不动点方程的模式，因为它接受两个参数，并且只在第一个参数上进行递归。 □

例子*12.*函数$left$ 和 $right$ 生成带有零和一标签的无限树。

$$\begin{aligned} &left : () \to Tree \\ &left\ \ () = Branch\,(left\,(), 0, right\,()) \\ &right : () \to Tree \\ &right\ \ () = Branch\,(left\,(), 1, right\,()) \end{aligned}$$

这两个定义不是最终不动点方程的实例，因为虽然函数是相互递归的，但数据类型不是。 □

在例子11中，初始代数的元素被嵌入到一个上下文中。
以点无关的风格编写， cat的定义形式为 $x \cdot (in \times id) = \Psi\, x$。这些讲义的核心思想是通过一个函子来建模这个上下文，将固定点方程推广到

$$x \cdot \mathsf{L}\,\text{在}\, = \Psi\, x, \qquad \text{并且对偶地} \qquad \mathsf{R}\, out \cdot x = \Psi\, x \ \ , \tag{95}$$

在左边，未知的x的类型是$\mathscr{C}(\mathsf{L}(\mu\mathsf{F}), A)$，在右边是$\mathscr{C}(A, \mathsf{R}(\nu\mathsf{G}))$。函子L模拟了$\mu$F的上下文。在cat的情况下，函子是$\mathsf{L} = - \times S\mathrm{tack}$。对偶地，R允许$x$返回嵌入在上下文中的$\nu$G的元素。第3.3.2节讨论了Example 12中R的合适选择。

当然，函子 L和 R不能是任意的。 例如，对于 $\mathsf{L} = \mathsf{K}\, A$其中 $\mathsf{K} : \mathscr{C} \to \mathscr{C}^{\mathscr{D}}$是常量函子， $\Psi = id$，方程 $x \cdot \mathsf{L}\, in = \Psi\, x$简化为 $x = x$，这是适当类型的每个箭头都满足的。 确保唯一性的一种方法是要求 L和 R是伴随的： $\mathsf{L} \dashv \mathsf{R}$（第2.5节）。伴随转置允许我们在箭头的源中交换 L以获得目标中的 R，这是证明广义固定点方程（95）具有唯一解的关键。这就是我们接下来要做的。

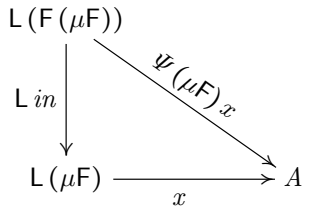
3.2.1 伴随初始固定点方程。让 $\mathscr{C}$和 $\mathscr{D}$是范畴，
让 $\mathsf{L} \dashv \mathsf{R}$是一对伴随函子 $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$ 和 $\mathsf{R} : \mathscr{C} \to \mathscr{D}$，让
$\mathsf{F} : \mathscr{D} \to \mathscr{D}$是一些自函子。 一个未知的伴随初始固定点方程in the unknown $x : \mathscr{C}(\mathsf{L}\,(\mu\mathsf{F}), A)$ has the syntactic form

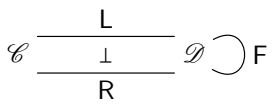
$$x \cdot \mathsf{L}\, in = \Psi\, x \ , \tag{96}$$

其中基本函数 Ψ的类型为

$$\mathscr{C}(\mathsf{L}X, A) \to \mathscr{C}(\mathsf{L}\,(\mathsf{F}\,X), A) \ .$$

（96）的唯一解称为伴随折叠，表示为 $(\Psi))_{\mathsf{L}}$。下面的图表总结了类型信息。





唯一性的证明必须基于左伴随在 A中的自然性。

$$\begin{array}{ll}
& x \cdot \mathsf{L}\, in = \Psi\, x \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad \lceil \lfloor f \rfloor \rceil = f \ \ (27)\ \} \\
& \lfloor x \cdot \mathsf{L}\, in \rfloor = \lfloor \Psi\, x \rfloor \\
\Longleftrightarrow & \{\text{融合:}\ \ \lfloor - \rfloor \text{在}\ A\text{中是自然的}\ (37)\ \} \\
& \lfloor x \rfloor \cdot in = \lfloor \Psi\, x \rfloor \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad \lceil \lfloor f \rfloor \rceil = f\ \} \\
& \lfloor x \rfloor \cdot in = \lfloor \Psi \lceil \lfloor x \rfloor \rceil \rfloor \\
\Longleftrightarrow & \{\ \text{第3.1.1节} \qquad \} \\
& \lfloor x \rfloor = (\lambda x \,.\, \lfloor \Psi \lceil x \rceil \rfloor) \\
\Longleftrightarrow & \{\ \text{伴随:} \qquad f = \lceil g \rceil \Longleftrightarrow \lfloor f \rfloor = g \ \ (27)\ \} \\
& x = \lceil (\lambda x \,.\, \lfloor \Psi \lceil x \rceil \rfloor) \rceil
\end{array}$$

通过三个简单的步骤，我们将伴随折叠 $\lfloor\ x\rfloor : \mathscr{C}(\mathsf{L}\,(\mu\mathsf{F}), A)$转化为标准折叠 $\lfloor x\rfloor : \mathscr{D}(\mu\mathsf{F}, \mathsf{R}\,A)$以及伴随基础函数$\Psi : \forall X \ .\mathscr{C}(\mathsf{L}X, A) \to \ \mathscr{C}(\mathsf{L}\,(\mathsf{F}\,X), A)$ 转化为标准基础函数$\lambda x \ .\lfloor \ \Psi \lceil x \rceil \rfloor : \forall \ X$.我们在第3.1.1节中已经证明了该方程有唯一解。总结一下，

$$(\Psi))_{\mathsf{L}} = \lceil (\lambda x \,.\, \lfloor \Psi \lceil x \rceil \rfloor)) \rceil \text{\或者，等价地，} \ \lfloor (\Psi))_{\mathsf{L}} \rfloor = (\lambda x \,.\, \lfloor \Psi \lceil x \rceil \rfloor)$$

3.2.2 伴随最终不动点方程。对偶地，一个未知数为 $x : \mathscr{D}(A, \mathsf{R}\,(\nu \mathsf{G}))$的伴随最终不动点方程具有以下句法形式

$$\mathsf{R}\; out \cdot x = \Psi\; x \quad , \tag{97}$$

其中基本函数 Ψ的类型为

$$\mathscr{D}(A, \mathsf{R}X) \to \mathscr{D}(A, \mathsf{R}\,(\mathsf{G}\;X)) \quad .$$

（97）的唯一解称为伴随展开，表示为 $[(\Psi])_{\mathsf{R}^\circ}$。

3.3 探索伴随

最简单的伴随示例是 $\mathsf{Id} \dashv \mathsf{Id}$，它证明了伴随
固定点方程（95）包含了固定点方程（91）。

$$\mathscr{C} \underset{\mathsf{Id}}{\overset{\mathsf{Id}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C}$$

在接下来的章节中，我们将探索更有趣的例子。每个章节的结构如下：我们介绍一个伴随，将方程（95）特化为伴随函子，并提供一些符合模式的Haskell示例。

3.3.1 柯里化。Haskell程序员最喜欢的伴随可能是柯里化：$- \times X \dashv (-)^X$（第2.5.6节）。让我们将伴随方程特化为 $\mathsf{L} = - \times X$和 $\mathsf{R} = (-)\; X$在 **Set**中。

$$\begin{aligned} & x \text{（在}a\text{中，在}c\text{中）} = \Psi\; x \text{（在}a\text{中，在}c\text{中）} \\ & \mathsf{R}\; out \cdot x = \Psi\; x \iff \forall\, a,\ c \circ\ out\ (x\ a\ c) = \Psi x\ a\ c \end{aligned}$$

伴随折叠函数接受两个参数，一个是初始代数的元素，另一个是第二个参数（通常是一个累加器，参见示例14），两者都在右侧可用。转置折叠函数（未显示）是一个高阶函数，产生一个函数。相反地，柯里化的展开函数被转换为非柯里化的展开函数。

示例*13*。为了将 *cat*的定义（参见示例*11*）转化为伴随方程的形式，我们按照第*3.1*节的步骤进行。首先，我们将 *Stack*转换为一个两层类型。

$$\begin{aligned} & \text{类型}\ \mathit{Stack} = \mu \mathfrak{Stack} \\ & \text{数据}\ \mathfrak{Stack}\ \mathit{stack} = \mathfrak{Empty} \mid \mathfrak{Push}\ (\mathit{Nat}, \mathit{stack}) \\ & \text{实例} \qquad \mathit{Functor}\ \mathfrak{Stack}\ \mathbf{where} \\ & \quad \mathit{fmap}\ f\ (\mathfrak{Empty}) \qquad = \mathfrak{Empty} \\ & \quad \mathit{fmap}\ f\ (\mathfrak{Push}\ (n, s)) = \mathfrak{Push}\ (n, f\ s) \end{aligned}$$

其次，我们确定从递归调用中抽象出来的基本函数，此外还要移除 in，并且我们将递归节点绑定起来（$\mathsf{L}\,(=-\times Stack)$）。

$$\begin{array}{lll}
(\mathsf{L}x \to Stack) \to (\mathsf{L}\,(\mathfrak{Stack}\,x) & & \to Stack) \\
\text{猫} \quad \text{猫} & (\mathfrak{Empty}, & ns) = ns \\
\text{猫} \quad \text{猫} & (\mathfrak{Push}\,(m, ms), & ns) = Push\,(m, cat\,(ms, ns))
\end{array}$$

$$\begin{array}{ll}
cat : \mathsf{L}\,Stack & \to Stack \\
cat\ \ (In\ ms, ns) & = \mathfrak{cat}\,cat\,(ms, ns)
\end{array}$$

定义方程符合伴随初始不动点方程的模式，
$x \cdot (in \times id) = \Psi\ x$。由于 $\mathsf{L}\,(=-\times Stack)$有一个右伴随，$cat$是唯一定义的。
转置折叠，猫′ = ⌊猫⌋,

$$\begin{array}{ll}
\text{猫}' : \text{堆栈} & \to \mathsf{R}\,\text{堆栈} \\
\text{猫}'\,(\text{在空中}) & = \lambda ns \to ns \\
\text{猫}'\,(\text{在}\,(\text{推}\,(\text{米,毫秒}))) & = \lambda ns \to \text{推}\,(\text{米}, (\text{猫}'\text{毫秒})\ ns)
\end{array}$$

只是猫的柯里化变体。 □

例子*14*。函数 $shunt$将第一个堆栈的元素推到第二个堆栈上。

$$\begin{array}{lll}
shunt : (\mu\text{堆栈}\ , & \text{堆栈}\ ) \to \text{堆栈} \\
shunt\ \ (\text{在空}, & ns) & = ns \\
shunt\ \ (\text{在}\,(\text{推}\,(\text{米,毫秒})), & ns) & = shunt\,(\text{毫秒,推}\,(\text{米}, ns))
\end{array}$$

与猫不同，$shunt$的参数在递归调用中发生了变化——它充当累加器。尽管如此，$shunt$适应了这个框架，因为它的基本函数

$$\begin{array}{lll}
\mathfrak{shunt} : \forall x\ .\ (\mathsf{L}\,x \to Stack) \to (\mathsf{L}\,(\mathfrak{Stack}\,x) & & \to Stack) \\
\mathfrak{shunt} \quad shunt & (\mathfrak{Empty}, & ns) = ns \\
\mathfrak{shunt} \quad shunt & (\mathfrak{Push}\,(m, ms), & ns) = shunt\,(ms, Push\,(m, ns))
\end{array}$$

具有所需的自然性质。修订后的 $shunt$定义

$$\begin{array}{ll}
shunt : \mathsf{L}\,(\mu\mathfrak{Stack}) & \to Stack \\
shunt\ \ (In\ ms, ns) & = \mathfrak{shunt}\,shunt\,(ms, ns)
\end{array}$$

完全符合伴随初始不动点方程的方案。 □

练习*36*。下面的尾递归变体是否是 $total$的（参见示例1）

$$\begin{array}{ll}
totalPlus : (Bush, Nat) & \to Nat \\
totalPlus\ \ (Leaf\,n, s) & = n + s \\
totalPlus\ \ (Fork\,(l, r), s) & = totalPlus\,(l, totalPlus\,(r, s))
\end{array}$$

一个伴随折叠吗? □

在Haskell中，列表是参数化的。我们能否将上述推理应用于参数化类型和多态函数?

例子*15*。列表的类型被给定为一个高阶基础函子的初始代数，其种类为 $(\star\to\star)\to(\star\to\star)$。

$$
\begin{aligned}
&\text{类型 List} = \mu\mathfrak{List}\\
&\text{数据 } \mathfrak{List}\,list\ a = \mathfrak{Nil} \mid \mathfrak{Cons}\,(a, list\ a)\\
&\text{实例} \qquad (Functor\ list) \Rightarrow Functor\,(\mathfrak{List}\,list)\ \mathbf{where}\\
&\quad fmap\ f\ \mathfrak{Nil} \qquad\qquad\quad = \mathfrak{Nil}\\
&\quad fmap\ f\ (\mathfrak{Cons}\,(a, as)) = \mathfrak{Cons}\,(f\ a, fmap\ f\ as)
\end{aligned}
$$

（再次强调，我们不需要函子对箭头的作用，它将自然变换映射为自然变换。）列表将栈、自然数序列推广到任意元素类型。同样，函数

$$
\begin{aligned}
&\text{附加}\colon \forall a\ .\ (\mu\text{列表}\ a, \qquad\qquad \text{列表}\ a) \to \text{列表}\ a\\
&\text{附加} \qquad\quad (\text{在空}, \qquad\qquad bs) \quad = bs\\
&\text{附加} \qquad\quad (\text{在}\ (\text{构造}\ (a, as)), bs) \quad = \text{在}\ (\text{构造}\ (a, \text{附加}\ (as, bs)))
\end{aligned}
$$

将cat(示例 11) 推广到任意元素类型的序列。

⊔⊓如果我们将乘积逐点提升到函子，$(\mathsf{F} \dot\times \mathsf{G})\,A = \mathsf{F}\,A \times \mathsf{G}\,A$，我们可以将附加视为类型的自然变换

$$\text{附加} : \text{列表} \dot\times \text{列表} \dot\to \text{列表}。$$

剩下的就是找到提升乘积 $-\dot\times \mathsf{H}$的右伴随。有人可能会认为 $\mathsf{F} \dot\times \mathsf{H} \dot\to \mathsf{G} \cong \mathsf{F} \dot\to (\mathsf{H} \dot\to \mathsf{G})$，但这种方法不起作用，因为 $\mathsf{H} \dot\to \mathsf{G}$在任何明智的方式下都不是一个函子（回想一下 $\mathsf{H} \dot\to \mathsf{G}$是从 H到 G的自然变换的集合）。此外，逐点提升指数 $\mathsf{G}^{\mathsf{H}}\,A = (\mathsf{G}\,A)^{\mathsf{H}\,A}$不起作用，因为数据再次不能定义一个函子，因为指数在其第一个参数中是逆变的。为了取得进展，让我们假设函子范畴是 $\mathbf{Set}^{\mathscr{C}}$，使得 $\mathsf{G}^{\mathsf{H}} : \mathscr{C} \to \mathbf{Set}$。（反变的、集合值的函子和自然变换的范畴被称为预层的范畴。）我们的推理如下：

$$
\begin{aligned}
&\mathsf{G}^{\mathsf{H}}\,A\\
\cong\ &\{\ \text{Yoneda引理}\ (85)\ \ \}\\
&\mathscr{C}(A, -) \dot\to \mathsf{G}^{\mathsf{H}}\\
\cong\ &\{\ \text{要求:} \qquad -\dot\times \mathsf{H} \dashv (-)^{\mathsf{H}}\ \}\\
&\mathscr{C}(A, -) \dot\times \mathsf{H} \dot\to \mathsf{G}。
\end{aligned}
$$

推导表明，函子 H和 G的指数由 $\mathscr{C}(A, -)\dot\times \mathsf{H} \dot\to \mathsf{G}$给出。然而，计算并没有证明所定义的函子实际上是 $-\dot\times \mathsf{H}$的右伴随，因为它的存在在第二步中被假设。我们将证明留给读者作为一个（费力的）练习 - 一个更一般的结果，从 **Set**抽象出来可以在[11]中找到。

练习*37*。证明 $-\dot\times \mathsf{H}$是 $(-)^{\mathsf{H}}$的左伴随。

1. 证明 $\mathsf{G}^{\mathsf{H}}\ A = \mathscr{C}(A,-)\dot{\times}\ \mathsf{H} \dot{\to}\ \mathsf{G}$在 A中是函子的。 （函子 G^{H}将一个对象映射到一组自然变换，将一个箭头映射到一个将自然变换映射到自然变换的函数。）2. 伴随 $-\dot{\times}\ \mathsf{H} \dashv (-)^{\mathsf{H}}$的伴随定义

$$\begin{aligned}&\sigma\ X\,(\mathsf{F}k\,s,t)\ ,\\ &\lceil\tau\rceil = \lambda A\ .\ \lambda\,(s,t)\ .\ \tau\ A\ s\ A\,(id,t)\ .\end{aligned}$$

证明它们在 F和 G中是自然的，并且互为逆。 □

$*$练习*38.* 你能理解函子 $\mathsf{Id}^{\mathsf{Id}}$和 $\mathsf{Id}^{\mathsf{Sq}}$的含义吗? □

定义*16.*指数的定义超出了Haskell 2010的范围，因为它需要二阶类型（数据构造器 *Exp*具有二阶类型）。

$$\begin{aligned}&\text{新类型}\quad \mathsf{Exp}\,h\ g\ a = Exp\,\{\,exp^{\circ} : \forall x\ .\ (a \to x, h\ x) \to g\ x\,\}\\ &\text{实例}\quad\ \ Functor\,(\mathsf{Exp}\,h\ g)\ \mathbf{where}\\ &\quad fmap\ f\ (Exp\ h) = Exp\,(\lambda(k,t) \to h\,(k\cdot f,t))\end{aligned}$$

从道义上讲， h和 g也是函子。 然而，它们的映射函数不需要用来定义 $Functor$的 Exp$h\ g$实例。伴随是被定义的

$$\begin{aligned}&\lfloor-\rfloor_{\mathsf{Exp}} : (\text{函子}\ f\quad) \Rightarrow (\forall x\ .\ (f\,x, h\ x) \to g\ x) \to (\forall x\ .\ f\ x \to \mathsf{Exp}\,h\ g\ x)\\ &\lfloor\sigma\rfloor_{\mathsf{Exp}} = \lambda s \to Exp\,(\lambda(k,t) \to \sigma\,(fmap\ k\ s,t))\\ &\lceil-\rceil_{\mathsf{Exp}} : (\forall x\ .\ f\ x \to \mathsf{Exp}\,h\ g\ x) \to (\forall x\ .\ (f\,x, h\ x) \to g\ x)\\ &\lceil\tau\rceil_{\mathsf{Exp}} = \lambda(s,t) \to exp^{\circ}\,(\tau\ s)\ (id,t)\ .\end{aligned}$$

类型变量 f, g和 h隐式地普遍量化。 再次，大多数函子实例是不需要的。 □

例子*17*。继续例子*15*，我们可以得出结论，定义方程 $append$has有唯一的解。它的转置类型 $\mathsf{List} \dot{\to}\ \mathsf{List}^{\mathsf{List}}$很有趣，因为它结合了 $append$和 $fmap$：

$$\begin{aligned}&(a \to x) \to (\ \mathsf{List}\ x \to \mathsf{List}\ x)\\ &append' \quad \text{作为} \quad = \quad \lambda f \quad \to \lambda bs \ \to append\,(fmap\ f\ as, bs)\ 。\end{aligned}$$

为了清晰起见，我们已经内联了Exp List List的定义。 □

3.3.2 相互值递归。函数**left** 和**right** 在例子**12**中通过相互递归定义。 该程序类似于例子5，它定义了 $varsExpr$和 $varsStat$，但有一个显著的区别，即只涉及一个数据类型，而不是一对相互递归的数据类型。

尽管如此，这个对应关系建议将左和右视为一个单一的箭头在一个乘积范畴中。

$$\text{树}\quad : \langle 1, 1\rangle \to \Delta(\nu\text{树}\quad)$$

箭头树是一个伴随展开，因为对角函子 $\Delta : \mathscr{C} \to \mathscr{C}\times\mathscr{C}$有一个左伴随，即余积（第2.3.2节和2.5.1节）。使用类似的推理

如同第3.1.3节所述，我们可以展开特化到对角函子的伴随最终不动点方程：

$$\Delta\, out \cdot x = \Psi\ x \quad \Leftarrow\!\Rightarrow \quad out \cdot x_1 = \Psi_1 \langle\, x_1, x_2 \rangle \text{ 和 } \quad out \cdot x_2 = \Psi_2 \langle\, x_1, x_2 \rangle \quad ,$$

得到的方程与第$3._1.3$节中的方程类似，只是现在析构函数out在两个方程中是相同的。

例子*18.*继续例子*12*，左和右的基本函数分别为

$$\begin{array}{l} ((\,)\to x,\,(\,)\to x)\to((\,)\to \text{树}\, x)\ \mathfrak{left} \\ \qquad\qquad (left, \quad right) \qquad (\,) = \mathfrak{Branch}(left(\,), 0, \mathfrak{right}(\,))\ \forall \\ x.((\,)\to x,\,(\,)\to x)\to((\,)\to \text{树}\, x)\ \mathfrak{right} \\ \qquad\qquad (left, \quad right) \qquad (\,) \ = \text{分支} \quad (\text{左}\ \ (\,), 1, \text{右} \quad (\,))\ . \end{array}$$

递归方程

$$\begin{array}{l} \text{左} \ : (\,) \to \nu \text{树} \\ \text{左} \ \ (\,) \ = \text{输出}^\circ\, (\text{左}\, (\text{左}, \text{右})\ (\,)) \\ \text{右} \quad : (\,) \to \nu \text{树} \\ \text{右} \ \ (\,) \ = \text{输出}^\circ\, (\text{右}\, (\text{左}, \text{右})\ (\,)) \end{array}$$

完全匹配上述模式（如果我们将输出°移到左边）。因此，两个函数都是唯一定义的。它们的转置，$[\langle \text{左}, \text{右} \rangle]$，使用余积将这两个函数合并为一个函数。

$$\begin{array}{l} \text{树} \quad : \text{要么}\ \ (\,)\ (\,) \to \nu \text{树} \\ \text{树} \ \ (\text{左} \ \ (\,)) \ \ = \text{输出}^\circ\, (\text{分支}\, (\text{树}\, (\text{左}\ (\,)), 0, \text{树}\ (\text{右}\ (\,)))) \\ \text{树} \ \ (\text{右}\ (\,)) \ \ = \text{输出}^\circ\, (\text{分支}\, (\text{树}\, (\text{左}\ (\,)), 1, \text{树}\ (\text{右}\ (\,)))) \end{array}$$

预定义的数据类型 Either 由 $\mathbf{data}$ Either$a\ b \ = Left\ a \ \ | Right\ b$ 是Haskell的余积。 □

让我们转向对偶情况。为了处理由相互递归定义的折叠，我们需要对角函子的右伴随，即 $product$(Sections 2.3.1and 2.5.1)。特化伴随初始不动点方程得到

$$\langle x_1,\, x_2 \rangle \cdot \Delta in = \Psi \langle x_1,\, x_2 \rangle \quad \Leftarrow\!\Rightarrow \quad x_1 \cdot in = \Psi_1 \langle x_1,\, x_2 \rangle \text{ and } \ x_2 \cdot in = \Psi_2 \langle x_1,\, x_2 \rangle \ .$$

例子 *19.*我们可以使用相互值递归来适应二进制增量（例子 4）的定义到框架中。succ 的定义具有参数形式的参数形式[29]，因为驱动递归的参数不仅仅在递归调用中使用。这个想法是通过恒等函数来'保护'另一个出现，并假装这两个函数都是通过相互递归定义的。

$$
\begin{array}{lll}
succ : \mu\mathfrak{Nat} & \to Nat & \\
succ\ \ (In\,(\mathfrak{N})) & = In\,(\mathfrak{I}\,(In\,\mathfrak{N})) & \\
succ\ \ (In\,(\mathfrak{O}\ b)) & = In\,(\mathfrak{I}\,(id\ b)) & \\
succ\ \ (In\,(\mathfrak{I}\ \ b)) & = In\,(\mathfrak{O}\,(succ\ b)) &
\end{array}
\qquad
\begin{array}{lll}
id : \mu\mathfrak{Nat} & \to Nat & \\
id\ \ (In\,(\mathfrak{N})) & = In\,(\mathfrak{N}) & \\
id\ \ (In\,(\mathfrak{O}\ b)) & = In\,(\mathfrak{O}\,(id\ b)) & \\
id\ \ (In\,(\mathfrak{I}\ \ b)) & = In\,(\mathfrak{I}\,(id\ b)) &
\end{array}
$$

如果我们忽略递归调用，我们会发现这两个基本函数确实具有所需的多态类型。

$$
\begin{array}{l}
(x \to Nat, x \to Nat) \to (\mathfrak{Nat}\,x \to Nat) \\
\mathfrak{succ} \quad (succ,\ \ id) \quad (\mathfrak{N}) \quad = In\,(\mathfrak{I}\,(In\,\mathfrak{N})) \\
\mathfrak{succ} \quad (succ,\ \ id) \quad (\mathfrak{O}\ b) \quad = In\,(\mathfrak{I}\,(id\ b)) \\
\mathfrak{succ} \quad (succ,\ \ id) \quad (x \to Nat, x \to Nat) \to (\mathfrak{N}\mathfrak{at}\,x \to Nat) \\
\mathfrak{id} \quad (succ,\ \ id) \quad (\mathfrak{N}) \quad = In\,(\mathfrak{N}) \\
\mathfrak{id} \quad (succ,\ \ id) \quad (\mathfrak{O}\ b) \quad = In\,(\mathfrak{O}\,(id\ b)) \\
\mathfrak{id} \quad (succ,\ \ id) \quad (\mathfrak{I}\ \ b) \quad = In\,(\mathfrak{I}\,(id\ b))
\end{array}
$$

转置折叠的类型为 $\mu\mathfrak{Nat} \to Nat \times Nat$，并对应于通常的参数形式编码（使用元组）。这个技巧对于'基本函数' bogus不起作用，因为结果函数仍然缺乏自然性。 □

练习*39.*证明阶乘函数

$$
\begin{array}{l}
\text{数据}\ Peano = Z \mid S\ Peano \\
fac : Peano \to Peano \\
fac\ \ (Z) \quad = 1 \\
fac\ \ (S\ n) \quad = S\ n * fac\ n
\end{array}
$$

是一个伴随折叠。 □

练习*40.*你能否也适应斐波那契函数

$$
\begin{array}{lll}
fib : Peano & \to Peano & \\
fib\ \ (Z) & = Z & \\
fib\ \ (S\ Z) & = S\ Z & \\
fib\ \ (S\,(S\ n)) & = fib\ n + fib\,(S\ n) &
\end{array}
$$

进入伴随折叠的框架中？提示：引入第二个函数 $fib'\ n = fib\ (S\ n)$并将上述嵌套递归转化为相互递归。 □

3.3.3 单值递归。我们已经讨论了相互递归函数和由单递归定义的数据类型上的相互递归函数。但是对于一个在由相互递归定义的数据类型上递归的单个函数呢？

例子 *20.*下面的数据类型（参见例子 5）模拟了一个简单函数式编程语言的抽象语法（通常情况下，这是一个非常简化的例子）。

剥离的例子。

$$\begin{array}{l}\text{数据表达式 =变量 变量} \mid Let\text{（声明，表达式}\\ \text{）数据声明 =定义（变量，表达式）} \mid And\text{（声明}\\ \text{，声明）} bound: \qquad \to \text{变量}\\ \text{绑定（定义（变量}x\text{,表达式 }e\text{））} = \{\text{变量}x\}\\ \text{绑定（与（数据}_1\text{,数据}_2\text{））=绑定 数据}_1 \cup \text{绑定 数据}\end{array}$$

函数绑定确定由声明定义的变量。

□

函数绑定通过结构递归进行，但它不是一个简单的折叠因为声明不是一个初始代数：声明 =外部 (μ语法)。 我们可以将绑定视为伴随折叠，前提是外部是伴随情况的一部分。 事实证明，如果基本类别具有初始和终结对象，则投影函子外部和外部都有左和右伴随。 我们证明外部有一个右伴随，其他证明完全类似。

$$\begin{array}{ll} & \mathscr{C}(\mathsf{Outl}\, A, B)\\ \cong & \{\ S \times 1 \cong S\ \}\\ & \mathscr{C}(\mathsf{Outl}\, A, B) \times 1\\ \cong & \{\ \text{假设:} \qquad \mathscr{D}\ \text{有一个最终对象}\ \ \}\\ & \mathscr{C}(\mathsf{Outl}\, A, B) \times \mathscr{D}(\mathsf{Outr}\, A, 1)\\ \cong & \{\ \mathscr{C} \times \mathscr{D}\text{的定义}\}\\ & (\mathscr{C} \times \mathscr{D})(A, \langle B, 1\rangle)\end{array}$$

同构在 A和 B中是自然的，因为每一步都是。下面的图表总结了伴随情况。

$$\mathscr{C} \underset{\langle -,\, 1\rangle}{\overset{\mathsf{Outl}}{\underset{\longrightarrow}{\longleftarrow}}} \mathscr{C} \times \mathscr{D} \underset{\mathsf{Outl}}{\overset{\langle -,\, 0\rangle}{\underset{\longrightarrow}{\longleftarrow}}} \mathscr{C}$$

将伴随不动点方程特化为 Outl得到

$$x \cdot in_1 = \Psi\, x, \qquad \text{和} \qquad out_1 \cdot x = \Psi\, x\ \ ,$$

其中 $in_1 = \mathsf{Outl}\, in$ 和 $out_1 = \mathsf{Outl}\, out$.

练习*41.*定义Ha*sk*ell函数

$$\begin{array}{l} freeExpr : Expr \to Vars\\ freeDecl : Decl \to Vars\end{array}$$

它们分别确定表达式和声明的自由变量。
尝试将它们捕获为伴随折叠。 （这比你可能最初想的要复杂，因为free*Ex*pr很可能还依赖于*Ex*ample 20中的*b*ound函数）。

□

给$bound$赋予语义的另一种方法是利用乘积范畴上的函子的不动点可以用一元函子的不动点来表示的事实[30]：

$$\mu\,\mathsf{F} \cong \langle \mu\,X\,.\,\mathsf{F}_1\,\langle X,\,\mu\,Y\,.\,\mathsf{F}_2\,\langle X,\,Y\rangle\rangle,\,\mu\,Y\,.\,\mathsf{F}_2\,\langle \mu\,X\,.\,\mathsf{F}_1\,\langle X,\,Y\rangle,\,Y\rangle\rangle\ ,$$

其中 $\mathsf{F}_1 = \mathsf{Outl} \circ \mathsf{F}$ 和 $\mathsf{F}_2 = \mathsf{Outr} \circ \mathsf{F}$.

3.3.4 类型应用。高阶初始代数的折叠必然是自然变换，因为它们存在于一个函子范畴中。 然而，许多Haskell函数在参数化数据类型上递归时实际上是单态的。

例子*21*。类型 Sequ泛化了二叉叶树的类型，从类型 Nat中抽象出来。

$$\begin{array}{l}
\text{类型 } \mathsf{Sequ} = \mu\mathfrak{Sequ} \\
\text{数据 } \mathfrak{Sequ}\,sequ\ a = \mathfrak{Single}\ a \mid \mathfrak{Cat}\,(sequ\ a, sequ\ a) \\
\textbf{instance}\,(Functor\ sequ) \Rightarrow Functor\,(\mathfrak{Sequ}\ sequ)\ \textbf{where} \\
\quad fmap\ f\ (\mathfrak{Single}\ a) = \mathfrak{Single}\,(f\ a) \\
\quad fmap\ f\ (\mathfrak{Cat}\,(l, r)) = \mathfrak{Cat}\,(fmap\ f\ l, fmap\ f\ r)
\end{array}$$

函数 $sums$ 定义

$$\begin{array}{lll}
sums : \mu\mathfrak{Sequ}\ Nat & \to Nat \\
sums\ \ (In\,(\mathfrak{Single}\ n)) & = n \\
sums\ \ (In\,(\mathfrak{Cat}\,(l, r))) & = sums\ l + sums\ r
\end{array}$$

sums 是一个非空自然数序列的求和函数。 它是对参数化叶子树的总和（示例1）的适应。 □

$sums$ 的定义看起来很像一个折叠函数，但它不是，因为它的类型不对。 对于随机访问列表的相应函数甚至不像折叠函数。

例子*22*。函数 $suma$ 对随机访问列表求和。

$$\begin{array}{lll}
suma : \mu\mathfrak{Array}\ Nat & \to Nat \\
suma\ \ (In\,(\mathfrak{Null})) & = 0 \\
suma\ \ (In\,(\mathfrak{Zero}\ s)) & = suma\,(fmap\ plus\ s) \\
suma\ \ (In\,(\mathfrak{One}\,(a, s))) & = a + suma\,(fmap\ plus\ s) \\
plus : \ (Nat, Nat) & \to Nat \\
plus\ \ \ (a, \quad b) & = a + b
\end{array}$$

请注意， $suma$的递归调用不适用于输入的子项。
事实上，它们不能，因为参数 s的类型是 $\mathsf{Array}\,(Nat, Nat)$，而不是 $\mathsf{Array}\ Nat$。
顺便说一下，这个定义需要 μ的函子实例（定义9）。 ⊔⊓

也许令人惊讶的是，上述定义符合伴随固定点方程的框架。 我们已经知道类型应用是一个函子（第2.2.3节）。使用这个高阶函子，我们可以为suma 分配类型
$(-Nat)\,(\mu\mathfrak{Array})\to Nat.$ 剩下的就是检查 $-X$ 是否是伴随的一部分。 事实证明，在一些温和的条件下（存在共能和幂）， $-X$ 有一个左伴随和一个右伴随。 我们选择推导出左伴随。

$$\mathscr{C}(A,\mathsf{B}\,X)\cong\{\ Yoneda引理\ (85)\ \}\forall\,Y:\mathscr{D}\,.$$
$$\mathscr{C}(\ \mathsf{Lsh}XA\ Y,\mathsf{B}\ Y)\cong\{自然变换\ \}\mathsf{Lsh}\ XA\ \dot{\to}\ \mathsf{B}$$

由于每一步在 A和 B中都是自然的，所以复合同构也是在 A和 B中自然的。 我们称 Lsh_X为 X的左移，因为没有更好的名称。
类似地，右伴随是 $\mathsf{Rsh}_X\ B=\lambda Y\ :\mathscr{D}\ .\ \prod\mathscr{D}(\ Y,X)$。$B$是 X的右移。 下面的图表总结了类型信息。

$$\mathscr{C}^{\mathscr{D}}\ \underset{-X}{\overset{\mathsf{Lsh}_X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\ \mathscr{C}\ \underset{\mathsf{Rsh}_X}{\overset{-X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\ \mathscr{C}^{\mathscr{D}}$$

回想一下，在 **Set**中，copower $\sum I$的定义如下。A是笛卡尔积 $I\times A$和幂 $\prod I$ 的集合。A是函数 $I\to A$的集合。 这个对应关系暗示了下面的Haskell实现。 然而，重要的是要记住 I是一个集合，而不是环境范畴中的对象（如 A）。

定义23。可以如下定义函子 Lsh和 Rsh。

$$\begin{aligned}&\mathbf{newtype}\,\mathsf{Lsh}_x\ a\ y=Lsh\,(x\to y,a)\\&\mathbf{instance}\,Functor\,(\mathsf{Lsh}_x\ a)\,\mathbf{where}\\&\quad fmap\ f\ (Lsh\,(k,a))=Lsh\,(f\cdot k,a)\\&\mathbf{newtype}\,\mathsf{Rsh}_x\ b\ y=Rsh\,\{\,rsh^{\circ}:(y\to x)\to b\,\}\\&\mathbf{instance}\,Functor\,(\mathsf{Rsh}_x\ b)\,\mathbf{where}\\&\quad fmap\ f\ (Rsh\ g)\ \ =Rsh\,(\lambda k\to g\,(k\cdot f))\end{aligned}$$

类型 $\mathsf{Lsh}_x a\ y$可以被看作是一个抽象数据类型： a是内部状态
而 $x\to y$是观察函数—通常情况下，但不总是，类型 a和 x
是相同的（$\mathsf{Lsh}_x\,x$ 是一个共函子，类似于共态共函子）。 相反地，
$\mathsf{Rsh}_x b\ y$实现了一个延续类型—同样，类型 x和 b很可能是

保持源代码和数学符号不变 ($\mathsf{Rsh}_x\ x$是延续单子，参见练习44)。伴随被定义为

$$\begin{aligned}
&\lfloor - \rfloor_{\mathsf{Lsh}} : \forall\, x\ a\ b\ .\ (\forall\, y\ .\ \mathsf{Lsh}_x\ a\ y \to b\ y) \to (a \to b\ x)\\
&\lfloor \alpha \rfloor_{\mathsf{Lsh}} = \lambda s \to \alpha\,(Lsh\,(id, s))\\
&\lceil - \rceil_{\mathsf{Lsh}} : \forall\, x\ a\ b\ .\ (Functor\ b) \Rightarrow (a \to b\ x) \to (\forall\, y\ .\ \mathsf{Lsh}_x\ a\ y \to b\ y)\\
&\lceil g \rceil_{\mathsf{Lsh}} = \lambda(Lsh\,(k, s)) \to fmap\ k\ (g\ s)\\
&\lfloor - \rfloor_{\mathsf{Rsh}} : \forall\, x\ a\ b\ .\ (Functor\ a) \Rightarrow (a\ x \to b) \to (\forall\, y\ .\ a\ y \to \mathsf{Rsh}_x\ b\ y)\\
&\lfloor f \rfloor_{\mathsf{Rsh}} = \lambda s \to Rsh\,(\lambda k \to f\,(fmap\ k\ s))\\
&\lceil - \rceil_{\mathsf{Rsh}} : \forall\, x\ a\ b\ .\ (\forall\, y\ .\ a\ y \to \mathsf{Rsh}_x\ b\ y) \to (a\ x \to b)\\
&\lceil \beta \rceil_{\mathsf{Rsh}} = \lambda s \to rsh^{\circ}\,(\beta\ s)\ id\ \ .
\end{aligned}$$

注意伴随也是自然的在 x中，伴随的参数。
(练习45要求你探索这个事实。) □

像往常一样，让我们专门研究伴随方程（在 **Set** 中）。

$$x\,(in\ X\ s) = \Psi x\ s$$
$$(-\ X)\ out \cdot x = \Psi\ x \quad \Longleftrightarrow \quad \forall\, a\ .\ out\ X\ (x\ a) = \Psi\ x\ a$$

由于类型抽象和类型应用在Haskell中是不可见的，事实上，伴随方程与标准的不动点方程无法区分。

例子 *24*.继续例子 *22*， $suma$的基本函数是

$$\begin{aligned}
&\mathfrak{suma}\colon \forall\, x\ .(\text{函子}\ x) \Rightarrow\\
&\qquad (x\ \text{自然数} \to \text{自然数}) \to (\text{数组}\ x\ \text{自然数} \to \text{自然数})\\
&\mathfrak{suma}\ \ suma\ \ (\mathfrak{Null}) = 0\\
&\mathfrak{suma}\ \ suma\ \ (\text{零}\ \ s) = suma\,(\text{映射加法}\ s)\\
&\mathfrak{suma}\ \ suma\ \ (\text{一个}\,(a, s)) = a + suma\,(\text{映射加法}\ s)\ \ .
\end{aligned}$$

该定义需要数组 x函子实例，进而引出了函子 x上下文。 $suma$的转置是一个返回高阶函数的折叠： $suma'$:数组 x →右移自然数 自然数.

$$\begin{aligned}
&suma' : \forall\, x\ .\ \text{数组}\ x \to (x \to \text{自然数}) \to \text{自然数}\\
&suma'\ \ (Null) = \lambda k \to 0\\
&suma'\ \ (Zero\ s) = \lambda k \to suma'\ s\ (plus \cdot (k \times k))\\
&suma'\ \ (One\,(a, s)) = \lambda k \to k\ a + suma'\ s\ (plus \cdot (k \times k))\ \ .
\end{aligned}$$

有趣的是，这个转换将一个广义折叠在[31]的意义下转化为一个高效的广义折叠在[32]的意义下。 两个版本都具有线性运行时间，但 $suma'$避免了重复调用映射函数($fmap\ plus$)。

⊔⊓

⋆练习*42*。非空序列的类型（参见示例*21*）数据 $\mathsf{Sequ}\ a = Single\ a \mid C$

$$at\,(\mathsf{Sequ}\ a, \mathsf{Sequ}\ a)$$

也可以将其视为平方函子的自由单子 $\mathsf{Sq}\ A = A \times A$，
参见第2.6.4节。用 $eval$表示 $sums$。 □

∗练习43.这个类型

$$\text{数据树}\ a\ =\text{分支}\ (\text{树}\ a, a, \text{树}\ a)$$

泛化了分叉的类型（示例2），将标签的类型抽象出来。 将捕获生成 :自然数 → 树自然数作为一个伴随展开。 树的类型也可以看作是平方函子 $\mathsf{Sq}\ A = A \times A$的无限共自由余函子（参见练习34）。 用追踪的术语来表达生成。

□

∗练习44.这个练习的目的是展示对于每个选择的$_X : \mathscr{C}$，函子 $\mathsf{M} = \mathsf{Rsh}\ X\ X$是一个单子。 至少有两种方法：通过规范或通过实现。

1. 如果我们将 Rsh_X的规范特化，伴随 $-\ X\ \dashv\ \mathsf{Rsh}\ X$，设置为
（设定 $B := X$）

$$\lfloor - \rfloor : \mathscr{C}(\mathsf{A}\ X, X) \cong \mathscr{C}^{\mathscr{C}}(\mathsf{A}, \mathsf{M}) : \lceil - \rceil\ ,$$

我们得到一个与函子 A自然同构的同构。 这表明我们可以通过定义单元和幺半群的乘法来

$$\begin{aligned} \text{返回}\ &= \lfloor id \rfloor\ , \\ \text{连接}\ &= \lfloor e \cdot \mathsf{M}\ e \rfloor \text{其中}\ e = \lceil id \rceil\ . \end{aligned}$$

请注意箭头 $e : \mathscr{C}(\mathsf{M}\ X, X)$表示一个计算。 证明 $return$ 和 $join$满足单子律（见练习30）。

2. 如果单子由 $\mathsf{M}\ A = \prod \mathscr{C}(A, X)$实现。$X$，则我们可以使用第2.5.7节的组合子来定义

$$\begin{aligned} \text{返回}\ &= \bigtriangleup\ k\ .\ k\ , \\ \text{连接}\ &= \bigtriangleup\ k\ .\ \pi_{\pi_k}\ . \end{aligned}$$

证明这样定义的 $return$和 $join$满足单子律。 尝试将这些定义与Haskell标准库中的延续单子的实现联系起来。

□

∗练习45.像咖喱伴随一样，$-_X\ \dashv\ \mathsf{Rsh}_X$是一个带有参数的伴随。 应用练习35来证明有一种唯一的方式将 Rsh转化为一个双函子，使得双射 $\mathscr{C}(\mathsf{A}\ X, B) \cong \mathscr{C}^{\mathscr{D}}(\mathsf{A}, \mathsf{Rsh}_X\ B)$在 X上也是自然的：

$$\begin{array}{ccc} \mathscr{C}(\mathsf{A}\ \hat{X}, B) & \xrightarrow{\lfloor - \rfloor} & \mathscr{C}^{\mathscr{D}}(\mathsf{A}, \mathsf{Rsh}_{\hat{X}}\ B) \\ \Big\downarrow{\scriptstyle \mathscr{C}(\mathsf{A}\ p, B)} & & \Big\downarrow{\scriptstyle \mathscr{C}^{\mathscr{D}}(\mathsf{A}, \mathsf{Rsh}_p\ B)} \\ \mathscr{C}(\mathsf{A}\ \check{X}, B) & \xrightarrow[\lfloor - \rfloor]{} & \mathscr{C}^{\mathscr{D}}(\mathsf{A}, \mathsf{Rsh}_{\check{X}}\ B)\ , \end{array}$$

其中 $p : \mathscr{D}(\hat{X}, \check{X})$。探索一下。 □

3.3.5 类型组合。 延续上一节的主题，对于参数化类型的函数，考虑以下示例
。

例子25。定义的函数 $join$

$$\begin{array}{lll} join : \forall\, a\ .\ \mu\mathfrak{Sequ}\,(\mathsf{Sequ}\ a) \to \mathsf{Sequ}\ a \\ join & (In\,(\mathfrak{Single}\ s)) & = s \\ join & (In\,(\mathfrak{Cat}\,(l, r))) & = In\,(\mathfrak{Cat}\,(join\ l, join\ r)) \end{array}$$

将非空序列的非空序列展平。 □

定义具有普通折叠的结构，但类型不完全正确：我们需要一个类型为 $\mu\mathfrak{Sequ} \dot{\to}$ G的自然变换，但 $join$的类型为 $\mu\mathfrak{Sequ} \circ \mathsf{Sequ} \dot{\to}\ \mathsf{Sequ}$。 我们能否将定义适应伴随方程的框架中？ 答案是肯定的，“是的，我们可以！”类似于上一节的发展，第一步是确定一个左伴随。 我们已经知道预合成是一个函子（第2.2.3节）。使用这个高阶函子，我们可以给 $join$分配类型 $(-\circ\mathsf{Sequ})\,(\mu\mathfrak{Sequ}) \dot{\to}\ \mathsf{Sequ}$。 （我们将Sequ∘Sequ解释为 $(-\circ\mathsf{Sequ})\,\mathsf{Sequ}$而不是 $(\mathsf{Sequ}\circ -)\,\mathsf{Sequ}$，因为外部列表，用 $\mu\mathfrak{Sequ}$表示，驱动递归。）

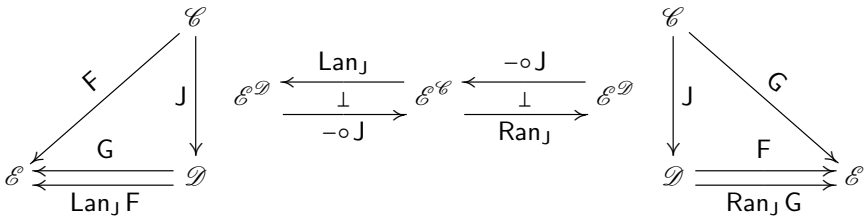
作为第二步，我们必须构建高阶
函子的右伴随。 事实证明，这是范畴论中一个经过深入研究的问题。
类似于前一节的情况，在某些条件下 $-\circ\mathsf{J}$ has
既有左伴随又有右伴随。 为了多样性，我们推导出后者。

$$\begin{array}{ll} & \mathsf{F}\circ\mathsf{J} \dot{\to} \mathsf{G} \\ \cong & \{\,\text{自然变换作为一个末尾 [8, p.223]}\,\} \\ & \forall\, Y : \mathscr{C}\ .\ \mathscr{E}(\mathsf{F}\,(\mathsf{J}\ Y), \mathsf{G}\ Y) \\ \cong & \{\ \text{Yoneda引理 (85)}\quad\ \} \\ & \forall\, Y :\ \mathscr{D}(X, \mathsf{J}\ Y) \to \mathscr{E}(\mathsf{F}X, \mathsf{G}\ Y) \\ \cong & \{\,\text{幂函数}: I \to \mathscr{C}(\ Y, B\,) \cong \mathscr{C}(\ Y, \textstyle\prod I\ .\ B\,)\ (52)\,\} \\ & \forall\, Y :\ \mathscr{E}(\mathsf{F}X, \textstyle\prod \mathscr{D}(\ X, \mathsf{J}\ Y)\ .\ \mathsf{G}\ Y) \\ \cong & \{\,\text{量词的互换 [8, p.231f]}\,\} \\ & \forall X :\ \mathscr{E}(\mathsf{F}\ X, \textstyle\prod \mathscr{D}(X, \mathsf{J}\,Y)\ .\ \mathsf{G}\ Y) \\ \cong & \{\ \text{同态函子}\qquad\quad \mathscr{E}(A, -)\ \text{保持端 [8, p.225]}\qquad\quad\ \} \\ & \forall X : \mathscr{D}\ .\ \mathscr{E}(\mathsf{F}\ X, \forall\, Y : \mathscr{C}\ .\ \textstyle\prod \mathscr{D}(X, \mathsf{J}\ Y)\ .\ \mathsf{G}\ Y) \\ \cong & \{\ \text{定义}\quad \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G} = \lambda\, X : \mathscr{D}\ .\ \forall\, Y : \mathscr{C}\ .\ \textstyle\prod \mathscr{D}(X, \mathsf{J}\ Y)\ .\ \mathsf{G}\ Y\ \} \\ & \forall X : \mathscr{D}\ .\ \mathscr{E}(\mathsf{F}\ X, \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G}\ X) \\ \cong & \{\,\text{自然变换作为一个端 [8, p.223]}\,\} \\ & \mathsf{F} \dot{\to} \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G} \end{array}$$

由于每一步在 F和 G中都是自然的，所以复合同构也是自然的
在 F和 G中。 函子 $\mathsf{Ran}_{\mathsf{J}}\,\mathsf{G}$被称为沿着 J的$\mathrm{right}\ K\ \mathrm{an}$扩展的 G。
（如果我们将 $\mathsf{J} : \mathscr{C} \to \mathscr{D}$视为一个包含函子，则 $\mathsf{Ran}_{\mathsf{J}}\,\mathsf{G} : \mathscr{D} \to \mathscr{E}$ extends

$\mathsf{G} : \mathscr{C} \to \mathscr{E}$扩展到整个 $\mathscr{D}$。）在 Ran_J的定义中，普遍量化的对象是所谓的 end，它对应于Haskell中的多态类型。一个end通常用一个积分符号表示；我更喜欢使用普遍量词，特别是因为它与自然变换的符号表示相融合。而且，自然变换是end的一个例子： $\mathscr{D}^{\mathscr{C}}(\mathsf{F},\mathsf{G}) = \forall\, X \,:\mathscr{C}\,.\,\mathscr{D}(\mathsf{F}X,\mathsf{G}\,X)$。我们将有兴趣的读者参考[8]以获取更多细节。

对偶地，$-\circ\mathsf{J}$的左伴随被称为左Kan扩展，并且被定义为$\mathsf{Lan}_\mathsf{J}\,\mathsf{F} = \lambda X \,:\mathscr{D}\,.\,\exists\, Y \,:\mathscr{C}\,.\,\sum \mathscr{D}(\mathsf{J}\,Y, X)\,.\,\mathsf{F}\,Y$。存在量化的对象是一个共端，它对应于Haskell中的存在类型（因此有这个符号）。下面的图表总结了类型信息。



定义*26*。与Exp一样，右K$a n$扩展的定义需要r$a n$k-*2*类型（数据构造函数Ran具有r$a n$k-*2*类型）。

$$\begin{aligned}&\mathbf{newtype}\,\mathsf{Ran}_j\;g\;x = Ran\,\{\,ran^\circ : \forall\, a\,.\,(x \to j\;a) \to g\;a\,\}\\&\mathbf{instance}\,Functor\,(\mathsf{Ran}_j\;g)\,\mathbf{where}\\&\quad fmap\;f\;(Ran\;h) = Ran\,(\lambda k \to h\,(k \cdot f))\end{aligned}$$

类型 $\mathsf{Ran}_j\;g$可以被看作是一个广义的延续类型—通常情况下，类型构造器$_j$和 g是相同的（Ran_J J被称为余密度单子，参见练习46）。从道义上讲，j和 g是函子。然而，它们的映射函数不需要定义 $\mathsf{Ran}\;j\;g$ 的 $Functor$实例。因此，我们省略了（$Functor\,j$，$Functor\,g$）上下文。伴随定义如下

$$\begin{aligned}&\lfloor - \rfloor_{\mathsf{Ran}} : \forall\, j\;f\;g\;.\;(Functor\;f) \Rightarrow (\forall x\;.\;f\;(j\;x) \to g\;x) \to (\forall x\;.\;f\;x \to \mathsf{Ran}_j\;g\;x)\\&\lfloor \alpha \rfloor_{\mathsf{Ran}} = \lambda s \to Ran\,(\lambda k \to \alpha\,(fmap\;k\;s))\\&\lceil - \rceil_{\mathsf{Ran}} : \forall\, j\;f\;g\;.\;(\forall x\;.\;f\;x \to \mathsf{Ran}_j\;g\;x) \to (\forall x\;.\;f\;(j\;x) \to g\;x)\\&\lceil \beta \rceil_{\mathsf{Ran}} = \lambda s \to ran^\circ\;(\beta\;s)\;id\;\;.\end{aligned}$$

请注意，伴随也是 j的自然变换，伴随的参数。
（练习47要求你探索这个事实。）

转向左Kan扩展的定义，我们需要Haskell 2010类型系统的另一个扩展：存在类型。

$$\begin{aligned}&\mathsf{Lan}\;\;(j\;a \to x, f\;a)\\&\mathbf{instance}\,Functor\,(\mathsf{Lan}_j\;f)\,\mathbf{where}\\&\quad fmap\;f\;(Lan\,(k, s)) = Lan\,(f \cdot k, s)\;\;.\end{aligned}$$

存在量词写作普遍量词在数据构造函数 Lan之前。理想情况下，Lan_j应该由一个 **newtype**声明给出，

但是 **newtype**构造函数不能有存在上下文（在GHC中）。出于类似的原因，我们不能使用析构函数，也就是选择函数 lan°。 类型 $\mathsf{Lan}_j\, f$可以看作是一个广义抽象数据类型：$f\, a$是内部状态，$j\, a \to x$是观察函数 - 同样，类型构造函数 j和 f很可能是相同的（$\mathsf{Lan}_\mathsf{J}\,\mathsf{J}$被称为密度余子）。 伴随由以下给出

$$\begin{aligned}
&\lfloor - \rfloor_{\mathsf{Lan}} : \forall j\, f\, g\, .\, (\forall x\, .\, \mathsf{Lan}_j\, f\, x \to g\, x) \to (\forall x\, .\, f\, x \to g\, (j\, x)) \\
&\lfloor \alpha \rfloor_{\mathsf{Lan}} = \lambda s \to \alpha\, (Lan\, (id, s)) \\
&\lceil - \rceil_{\text{兰}} : \forall j\, f\, g\, .\, (\text{函子}\ g\ ) \Rightarrow (\forall x\, .\, f\, x \to g\, (j\, x)) \to (\forall x\, .\, \text{兰}\ _j f\, x \to g\, x) \\
&\lceil \beta \rceil \text{兰} = \lambda(\text{兰}\ (\kappa, \sigma)) \to fmap\ \kappa\ (\beta \sigma)\ .
\end{aligned}$$

在Haskell中，这种构造的对偶性有些模糊。 □

像往常一样，让我们专门研究伴随方程（在 **Set** 中）。

$$out\, (\mathsf{J} A)\ (x\ A\ a) = \Psi x\ A\ a$$

当阅读方程作为Haskell定义时，通常的警告适用：由于类型应用是不可见的，派生方程与原始方程无法区分。

例子27。继续例子25， $join$的基本函数很直观，除了类型可能有些复杂。

$$\begin{aligned}
&\mathsf{join} : \forall x\, .\, (\forall a\, .\, x\, (\mathsf{Sequ}\ a) \to \mathsf{Sequ}\ a) \to \\
&\qquad\qquad (\forall a\, .\, \mathfrak{Sequ}\, x\, (\mathsf{Sequ}\ a) \to \mathsf{Sequ}\ a) \\
&\mathsf{join} \quad join \quad (\mathfrak{Single}\ s) \quad = s \\
&\mathsf{join} \quad join \quad (\mathfrak{Cat}\, (l, r)) \quad = In\, (\mathfrak{Cat}\, (join\ l, join\ r))
\end{aligned}$$

基本函数 join是一个二阶自然变换。 join的转置非常有启示性。 首先，它的类型是

$$join' : \mathsf{Sequ} \dot{\to} \mathsf{Ran}_{\mathsf{Sequ}}\, \mathsf{Sequ} \cong \forall a\, .\, \mathsf{Sequ}\ a \to \forall b\, .\, (a \to \mathsf{Sequ}\ b) \to \mathsf{Sequ}\ b\ .$$

类型表明 $join'$是单子 Sequ(练习42)的绑定，
这确实是这种情况!

$$\begin{aligned}
&join' : \forall a\ b\, .\, \mu\mathfrak{Sequ}\ a \to (a \to \mathsf{Sequ}\ b) \to \mathsf{Sequ}\ b \\
&join' \qquad \text{作为} \qquad = \lambda k \qquad\qquad \to join\, (fmap\ k\ as)
\end{aligned}$$

为了清晰起见，我们已经内联了 $\mathsf{Ran}_{\mathsf{Sequ}}\, \mathsf{Sequ}$。 □

Kan扩展推广了前一节的构造：如果范畴 $\mathscr{C}$非空 ($\mathscr{C} \neq \mathbf{0}$)，那么我们有 $\mathsf{Lsh}_A\, B \cong \mathsf{Lan}_{(\mathsf{K}\, A)}\, (\mathsf{K}\, B)$和$\mathsf{Rsh}_A\, B \cong \mathsf{Ran}_{(\mathsf{K}\, A)}\, (\mathsf{K}\, B)$，其中 K是常量函子。这是证明

右伴随：

$$\begin{aligned}
&\mathsf{F}\,A \to B \\
\cong\ &\{\text{箭头作为自然变换：}\ \ A \to B \cong \mathsf{K}\,A \dot{\to} \mathsf{K}\,B\text{如果}\ \mathscr{C} \neq \mathbf{0}\ \} \\
&\mathsf{K}\,(\mathsf{F}\,A) \dot{\to} \mathsf{K}\,B \\
=\ &\{\ \mathsf{K}\,(\mathsf{F}\,A) = \mathsf{F}\circ\mathsf{K}\,A\ \} \\
&\mathsf{F}\circ\mathsf{K}\,A \dot{\to} \mathsf{K}\,B \\
\cong\ &\{\ (-\circ\mathsf{J}) \dashv \mathsf{Ran}_{\mathsf{J}}\ \} \\
&\mathsf{F} \dot{\to} \mathsf{Ran}_{\mathsf{K}\,A}\,(\mathsf{K}\,B)\ .
\end{aligned}$$

由于伴随是同构唯一的（第2.5.8节），我们得出结论
$\mathsf{Ran}_{\mathsf{K}\,A}\circ\mathsf{K} \cong \mathsf{Rsh}_A$。

⋆练习*46*。Kan扩展广义移位。 同样，密度单子广义化了延续单子。 这个练习是由Mac Lane [8, *Exer-cise* X.7.3]提出的；在即将发表的论文[33]中解决。

1。 将练习44的论证推广，以证明 $\mathsf{M} = \mathsf{Ran}_{\mathsf{J}}\,\mathsf{J}$是一个单子对于每个选择的 $\mathsf{J} : \mathscr{C} \to \mathscr{D}$。 函子 M被称为 J的密度单子。 （具体来说，如果我们将伴随（$-\circ\mathsf{J}$）$\dashv \mathsf{Ran}_{\mathsf{J}}$特化为（$\mathsf{G} := \mathsf{J}$））

$$\lfloor - \rfloor : \mathscr{D}^{\mathscr{C}}(\mathsf{F}\circ\mathsf{J}, \mathsf{J}) \cong \mathscr{D}^{\mathscr{D}}(\mathsf{F}, \mathsf{M}) : \lceil - \rceil\ ,$$

我们得到一个对于函子 $\mathsf{F} : \mathscr{D} \to \mathscr{D}$的自然双射。 密度单子的单位和乘法由以下给出

$$\begin{aligned}
\text{返回}\quad &= \lfloor id \rfloor\ , \\
join &= \lfloor e \cdot \mathsf{M}\circ e \rfloor \text{其中}\quad e = \lceil id \rceil\ ,
\end{aligned}$$

其中自然变换 $e : \mathscr{D}^{\mathscr{C}}(\mathsf{M}\circ\mathsf{J}, \mathsf{J})$运行一个计算。）

2. 证明如果 $\mathsf{R} : \mathscr{C} \to \mathscr{D}$有一个左伴随 $\mathsf{L} \dashv \mathsf{R}$，则R的密度单子是由伴随（ $\mathsf{R}\circ\mathsf{L}, \eta, \mathsf{R}\circ\epsilon\circ\mathsf{L}$)引发的单子。 ⊔⊓⋆

练习*47*.伴随 $(-\circ\mathsf{J}) \dashv \mathsf{Ran}_{\mathsf{J}}$是带有参数的另一个伴随的例子。 应用练习35证明有一种唯一的方法将 Ran转化为高阶双函子，使得双射 $\mathscr{E}^{\mathscr{C}}(\mathsf{F}\circ\mathsf{J}, \mathsf{G}) \cong \mathscr{E}^{\mathscr{D}}(\mathsf{F}, \mathsf{Ran}_{\mathsf{J}}\,\mathsf{G})$在 J中也是自然的：

$$\begin{array}{ccc}
\mathscr{E}^{\mathscr{C}}(\mathsf{F}\circ\hat{\mathsf{J}}, \mathsf{G}) & \xrightarrow{\lfloor - \rfloor} & \mathscr{E}^{\mathscr{D}}(\mathsf{F}, \mathsf{Ran}_{\hat{\mathsf{J}}}\,\mathsf{G}) \\
\Big\downarrow{\scriptstyle \mathscr{E}^{\mathscr{C}}(\mathsf{F}\circ\alpha, \mathsf{G})} & & \Big\downarrow{\scriptstyle \mathscr{E}^{\mathscr{D}}(\mathsf{F}, \mathsf{Ran}_{\alpha}\,\mathsf{G})} \\
\mathscr{E}^{\mathscr{C}}(\mathsf{F}\circ\check{\mathsf{J}}, \mathsf{G}) & \xrightarrow[\lfloor - \rfloor]{} & \mathscr{E}^{\mathscr{D}}(\mathsf{F}, \mathsf{Ran}_{\check{\mathsf{J}}}\,\mathsf{G})\ ,
\end{array}$$

请记住 $-\circ=$是一个双函子（第2.2.3节）。因此， $-\circ\ \alpha$是一个高阶自然变换。探索一下。 □

3.3.6 交换参数。到目前为止，我们只考虑了归纳和共归纳类型的孤立情况。下面的例子介绍了两个将归纳类型与共归纳类型结合的函数。

例子*28*。二进制数上的分叉和函数是一一对应的。 函数 *tabulate*和 *lookup*证明了同构。

$$
\begin{array}{l}
\mathit{tabulate} : (\mathit{Nat} \to \mathit{Nat}) \to \mathit{Tree} \\
\mathit{tabulate}\ \ f \qquad\qquad\quad = \text{分支}\ (\text{制表}\ (\text{函数}\ \cdot\ O), \text{函数}\ N, \text{制表}\ (\text{函数}\ \cdot\ I)) \\
\text{查找}\ \ : \text{树}\ \ \to \qquad\quad (\text{自然数}\ \ \to \text{自然数}) \\
\text{查找}\ \ (\text{分支}\ (\text{左}, \text{值}, \text{右}))\ (N)\ \ = \text{值} \\
\text{查找}\ \ (\text{分支}\ (\text{左}, \text{值}, \text{右}))\ (O\ b)\ \ = \text{查找 左}\ b \\
\text{查找}\ \ (\text{分支}\ (\text{左}, \text{值}, \text{右}))\ (I\ \ b)\ \ = \text{查找 右}\ b
\end{array}
$$

第一个同构制表给定的函数，生成一个无限树
它的值。 它的逆向在给定位置查找二进制数。 □

制表是一个标准展开，但是查找呢？ 它的类型涉及指数：查找:$\mathscr{C}$(树,自然数$^{\mu}$自然数)。 然而，curry伴随$-\times X \dashv (-)^X$在这里不适用，因为右伴随固定源对象。我们需要它的对应物，函子$X^{(-)}$:$\mathscr{C}$^op→$\mathscr{C}$，它固定目标对象（练习22）。 由于这个函子是逆变的，查找的类型实际上是$\mathscr{C}$^op((自然数(−))^op($^{\mu}$自然数),树)，这暗示着箭头是一个伴随折叠！如果我们将伴随方程特化为$\mathscr{C}$=Set和L=$X^{(-)}$，我们得到

$$x \cdot \mathsf{L}\ \mathit{in} = \Psi\ x \quad \Longleftrightarrow \quad \forall s\ .\ \forall a\ .\ x\ a\ (\mathit{in}\ s) = \Psi\ x\ a\ s \ \ .$$

所以 x只是一个对第二个参数进行递归的柯里化函数。

到目前为止，我们还没有提到展开。原因可能令人惊讶。 在这种特殊情况下，伴随展开与伴随折叠是相同的！考虑伴随展开的类型： $\mathscr{C}(A, \mathsf{R}\,(\nu \mathsf{F}))$。 由于 $\mathsf{R} = X^{(-)}$是逆变的，在 $\mathscr{C}^{\mathrm{op}}$中的最终余代数是 $\mathscr{C}$中的初始代数。 此外，由于 $X^{(-)}$是自伴的，我们得到了伴随折叠的类型： $\mathscr{C}(A, \mathsf{L}\,(\mu \mathsf{F})) = \mathscr{C}^{\mathrm{op}}(\mathsf{L}\,(\mu \mathsf{F}), A)$。

表4总结了本节中考虑的伴随关系。

练习*48*。练习32要求你探索伴随关系 $\mathsf{Free} \dashv \mathsf{U}$，其中
$\mathsf{U} : \mathbf{Mon} \to \mathbf{Set}$是忘记了 **Mon**的附加结构的函子。
探索类型为 $\mathsf{Free}\,(\mu \mathsf{F}) \to A$的伴随折叠。 □

3.4程序验证

在本节中，我们开发了伴随折叠的计算性质——读者被邀请将结果对偶化为伴随展开。 3.4.1节关注支持结构化无点理论的定律。 然后，第3.4.2节将重点从无点理论转移到点点理论。 它提供了一种统一的证明方法，可以轻松地适应支持有效的点点计算。

表4。伴随关系和递归类型。

伴随	初始不动点方程	最终不动点方程
$\mathsf{L} \dashv \mathsf{R}$	$x \cdot \mathsf{L}\ in = \Psi\ x$ $x' \cdot in = \lfloor \Psi \lceil x' \rceil \rfloor$	输出 $\cdot x = \Psi\ x$ 输出 $\cdot$变量′ $= \lceil \Psi \lfloor$ 变量′ $\rfloor \rceil$
$\mathsf{Id} \dashv \mathsf{Id}$	标准折叠 标准折叠	标准展开 标准展开
$- \times X \dashv (-)^X$	参数化折叠 $x \cdot (in \times X) = \Psi\ x$ 折叠成指数	柯里化展开 输出$^X \cdot x = \Psi\ x$ 从乘积展开
$(X^{(-)})^{\mathsf{op}} \dashv X^{(-)}$	交换的柯里化折叠 变量在$\cdot$变量 $= \Psi$变量 折叠成指数	
$(+) \dashv \Delta$	递归从一个余积的 相互递归类型 在上的相互值递归 相互递归类型	相互值递归 输出 $\cdot$变量$_1 = \Psi_1 \langle$变量$_1$,变量$_2 \rangle$ 输出 $\cdot$变量$_2 = \Psi_2 \langle$变量$_1$,变量$_2 \rangle$ 从一个单一递归的 余积域
$\Delta \dashv (\times)$	相互值递归 $x_1 \cdot in = \Psi_1 \langle x_1,\ x_2 \rangle$ $x_2 \cdot in = \Psi_2 \langle x_1,\ x_2 \rangle$ 单一递归到一个 乘积域	递归到一个乘积的 相互递归类型 在上的相互值递归 相互递归类型
$\langle -,\ 0 \rangle \dashv \mathsf{Outl}$	—	在上的单一值递归 相互递归类型 输出$_1 \cdot x = \Psi\ x$ 在上的'相互'值递归 相互递归类型
$\mathsf{Outl} \dashv \langle -,\ 1 \rangle$	单值递归 相互递归类型 $x \cdot in_1 = \Psi\ x$ 在上的'相互'值递归 相互递归类型	—
$\mathsf{Lsh}_X \dashv (-\ X)$	—	单态展开 输出 $X \cdot x = \Psi\ x$ 从左移展开
$(-\ X) \dashv \mathsf{Rsh}_X$	单态折叠 $x \cdot in\ X = \Psi\ x$ 向右移折叠	—
$\mathsf{Lan}_\mathsf{J} \dashv (- \circ \mathsf{J})$	—	多态展开 输出$\circ$J$\cdot\ x = \Psi\ x$ 从左侧 Kan 扩展展开
$(- \circ \mathsf{J}) \dashv \mathsf{Ran}_\mathsf{J}$	多态折叠 $x \cdot in \circ \mathsf{J} = \Psi\ x$ 向右侧 Kan 扩展折叠	—

3.4.1 唯一性属性。伴随折叠享有通常的多样性属性。 伴随初始不动点方程的唯一解可以通过以下等价性来捕捉，即唯一性属性。

$$x = (\Psi)_{\mathsf{L}} \quad \Longleftrightarrow \quad x \cdot \mathsf{L}\, in = \Psi\, x \tag{98}$$

唯一性属性有两个简单的结果。 首先，将左侧代入右侧得到计算规律。

$$(\Psi)_{\mathsf{L}} \cdot \mathsf{L}\, in = \Psi\, (\Psi)_{\mathsf{L}} \tag{99}$$

该规律具有直观的操作性解读：伴随折叠的应用被折叠的主体替代。

其次，将 x 实例化为 id，我们得到反射规律。

$$(\Psi)_{\mathsf{L}} = id \quad \Longleftrightarrow \quad \Psi\, id = \mathsf{L}\, in \tag{100}$$

作为这些等式的一个应用，让我们推广香蕉分裂规律[9]。
在第2.6.6节中，我们已经以标准折叠的形式陈述了该定律。 然而，它可以很容易地转换为伴随折叠。 首先，我们介绍了两个代数的乘积的对应物（也参见练习49）：

$$(\Phi \otimes \Psi)\, x = \Phi\, (outl \cdot x) \vartriangle \Psi\, (outr \cdot x)\ 。 \tag{101}$$

值得指出的是， $\otimes$的定义既不涉及基函数 F也不涉及伴随函子 L—从某种意义上说，基函数隐藏了不必要的细节。

练习*49*。我们在第3.1.1节中已经看到，类型为 $\mathscr{C}(-, A) \to \mathscr{C}(\mathsf{F}\, -, A)$和 F-代数的基函数是一一对应的。 证明 $\otimes$对应于代数的乘积（对于 $\mathsf{L} = \mathsf{Id}$）。

□

广义的香蕉分裂定律则说明

$$(\Phi)_{\mathsf{L}} \vartriangle (\Psi)_{\mathsf{L}} = (\Phi \otimes \Psi)_{\mathsf{L}}\ . \tag{102}$$

为了证明，我们引用了唯一性属性（98）；义务如下所述被解除。

$$\begin{aligned}
& ((\Phi)_{\mathsf{L}} \vartriangle (\Psi)_{\mathsf{L}}) \cdot \mathsf{L}\, in \\
= \quad & \{\ \text{融合}\ (12)\ \ \} \\
& (\Phi)_{\mathsf{L}} \cdot \mathsf{L}\, in \vartriangle (\Psi)_{\mathsf{L}} \cdot \mathsf{L}\, in \\
= \quad & \{\ \text{计算}\ (99)\ \ \ \ \ \ \ \ \} \\
& \Phi\, (\Phi)_{\mathsf{L}} \vartriangle \Psi\, (\Psi)_{\mathsf{L}} \\
= \quad & \{\ \text{计算}\ (9){-}(10)\ \ \ \ \ \ \ \ \} \\
& \Phi\, (outl \cdot ((\Phi)_{\mathsf{L}} \vartriangle (\Psi)_{\mathsf{L}})) \vartriangle \Psi\, (outr \cdot ((\Phi)_{\mathsf{L}} \vartriangle (\Psi)_{\mathsf{L}})) \\
= \quad & \{\ \text{定义 of}\ \ \ \ \ \ \otimes\ (101)\ \} \\
& (\Phi \otimes \Psi)\ ((\Phi)_{\mathsf{L}} \vartriangle (\Psi)_{\mathsf{L}})
\end{aligned}$$

练习*50.*考虑以下同构的字符串。

$$\begin{aligned}
&\mathsf{L}\ A \to B_1 \times B_2 \\
&\cong (\mathsf{L}\ A \to B_1) \times (\mathsf{L}\ A \to B_2) \\
&\cong (A \to \mathsf{R}\ B_1) \times (A \to \mathsf{R}\ B_2) \\
&\cong A \to \mathsf{R}\ B_1 \times \mathsf{R}\ B_2 \\
&\cong A \to \mathsf{R}\ (B_1 \times B_2) \\
&\cong \mathsf{L}\ A \to B_1 \times B_2
\end{aligned}$$

证明每一步。 为什么广义香蕉分裂定律在某种意义上是不令人惊讶的?

□

融合定律规定了将箭头$f : \mathscr{C}(A, B)$与伴随折叠 $(\Phi)\,)_{\mathsf{L}} : \mathscr{C}(\mathsf{L}\,(\mu\mathsf{F}), A)$融合形成另一个伴随折叠 $(\Psi)\,)_{\mathsf{L}} : \mathscr{C}(\mathsf{L}\,(\mu\mathsf{F}), B)$的条件。这个条件可以很容易地计算出来。

$$\begin{aligned}
& h \cdot (\Phi)_{\mathsf{L}} = (\Psi)_{\mathsf{L}} \\
\Longleftrightarrow\ & \{\ \text{唯一性性质}\ (98) \qquad\qquad \} \\
& h \cdot (\Phi)_{\mathsf{L}} \cdot \mathsf{L}\ in = \Psi\,(h \cdot (\Phi)_{\mathsf{L}}) \\
\Longleftrightarrow\ & \{\ \text{计算}\ (99) \qquad\qquad \} \\
& h \cdot \Phi\,(\Phi)_{\mathsf{L}} = \Psi\,(h \cdot (\Phi)_{\mathsf{L}}) \\
\Longleftarrow\ & \{\text{从}\ (\Phi)\,)_{\mathsf{L}} \text{中抽象出来}\ \} \\
& \forall f\ .\ h \cdot \Phi f = \Psi\,(h \cdot f)
\end{aligned}$$

因此,

$$h \cdot (\Phi)_{\mathsf{L}} = (\Psi)_{\mathsf{L}} \quad \Longleftarrow \quad \forall f\ .\ h \cdot \Phi f = \Psi\,(h \cdot f)\ . \tag{103}$$

至于广义香蕉分裂，融合条件 $h \cdot \Phi f = \Psi\,(h \cdot f)$不涉及基本函子 F或伴随函子 L，这使得定律易于使用。

练习*51.*设 a和 b是与基本函数 Φ和 Ψ（对于 $\mathsf{L} = \mathsf{Id}$）相对应的代数。证明

$$h \cdot a = b \cdot \mathsf{F}\ h \quad \Longleftrightarrow \quad \forall f\ .\ h \cdot \Phi f = \Psi\,(h \cdot f)\ .$$

换句话说，融合条件要求 h 是一个 F-代数同态。

□

例子*29.*函数 $height$确定堆栈的高度。

$$\begin{aligned}
&height : Stack \qquad\qquad \to \text{自然数} \\
&\text{空的高度} \qquad\qquad\quad\ = 0 \\
&\text{高度}\quad (\text{推}\quad (n, s)) = 1 + \text{高度}\ s
\end{aligned}$$

让我们证明高度是从堆栈幺半群到自然数加法幺半群的幺半群同态，即 $:(\text{堆栈}, \text{空}, \diamond) \to (\text{自然数}, 0, +)$:

$$\text{空的高度} \quad = 0 \ , \tag{104}$$

$$\text{高度} \ \ (x \diamond y) = \text{高度}\ x \ \ + \text{高度}\ y \quad , \tag{105}$$

或者，以无点风格写成，

$$\text{高度} \ \ \cdot \text{空} \quad = \text{零} \quad , \tag{106}$$

$$\text{高度} \cdot \text{连接} = \text{加} \cdot (\text{高度} \times \text{高度}) \ \ . \tag{107}$$

这里，零是产生0的常量箭头，空是产生空的常量箭头，最后，连接和加是 $\diamond$和 $+$的前缀写法。 第一个条件 (106) 是高度定义的直接结果。 关于第二个条件 (107)，没有明显的攻击区域，因为左边和右边都不是伴随折叠。 因此，我们分两步进行：首先我们证明左边可以融合成一个伴随折叠，然后我们证明右边满足这个折叠的伴随不动点方程。

对于第一步，我们正在寻找一个基本函数 $\mathfrak{height2}$，使得

$$height \cdot (\mathfrak{cat})_{\mathsf{L}} = (\mathfrak{height2})_{\mathsf{L}} \ ,$$

其中 $\mathsf{L} = - \times Stack$. 基本函数 $\mathfrak{cat}$在示例13中定义。 融合（103）立即给我们

$$\forall\, cat \,.\, height \cdot \mathfrak{cat}\, cat = \mathfrak{height2}\, (height \cdot cat) \ , \tag{10}$$

8）从中我们可以轻松合成 $\mathfrak{height2}$的定义：
情况 $\mathfrak{Empty}$：

$$\begin{aligned}
& \mathfrak{height2}\, (height \cdot cat)\ (\mathfrak{Empty}, y) \\
= \ & \{ \text{ specification of } \mathfrak{height2}\ (108) \ \} \\
& height\, (\mathfrak{cat}\, cat\, (\mathfrak{Empty}, y)) \\
= \ & \{ \ \mathfrak{cat}\text{的定义（示例13）} \ \} \\
& height\ y \ \ .
\end{aligned}$$

情况 $\mathfrak{Push}\, (a, x)$：

$$\begin{aligned}
& \mathfrak{height2}\, (height \cdot cat)\ (\mathfrak{Push}\, (a, x), y) \\
= \ & \{ \ \mathfrak{height2}\ (108)\text{的规范} \} \\
& height\, (\mathfrak{cat}\, cat\, (\mathfrak{Push}\, (a, x), y)) \\
= \ & \{ \ \mathfrak{cat}\text{的定义（例子13）和 } In \cdot \mathfrak{Push} = Push \ \} \\
& height\, (Push\, (a, cat\, (x, y))) \\
= \ & \{ \ height\text{的定义} \} \\
& 1 + (height \cdot cat)\ (x, y) \ \ .
\end{aligned}$$

从 $height \cdot cat$中抽象出来，我们得到

$$\begin{array}{lll} \mathfrak{height2} : \forall\, x\ .\ (\mathsf{L}\, x \to Nat) \to (\mathsf{L}\, (\mathfrak{Stack}\, x) & & \to \text{自然数}) \\ \mathfrak{height2} & height2 & (\mathfrak{Empty}, \quad y) = height\ y \\ \mathfrak{height2} & height2 & (\mathfrak{Push}\, (a, x), y) = 1 + height2\, (x, y)\ . \end{array}$$

对于第二步，我们必须展示

$$\text{加} \quad \cdot (height \times height) = (\mathfrak{height2})_{\mathsf{L}}\ .$$

通过唯一性 (98)，我们还需要证明

$$\text{加} \quad \cdot (height \times height) \cdot \mathsf{L}\, in = \mathfrak{height2}\, (plus \cdot (height \times height))\ ,$$

这是很容易解决的。 □

3.4.2 唯一不动点原理。 假设你想证明两个箭头的相等。 幸运的是，如果其中一个箭头采用伴随折叠的形式，我们可以要么通过唯一性属性，要么更好地调用融合定律来证明。不幸的是，更多的情况是，两个箭头都没有明确给出作为伴随折叠，这种情况下，这些定律都不直接适用。 属性 (107) 说明了这一观察结果：方程的两边都涉及伴随折叠，但它们本身并不是伴随折叠。

下面的证明方法，唯一的不动点原理，提供了一种解决这个困境的方法。 这个想法是证明 $f \cdot \mathsf{L}\, in = \Theta\, f$和 $\Theta\, g = g \cdot \mathsf{L}\, in$。
如果方程 $x \cdot \mathsf{L}\, in = \Theta\, x$有唯一解，那么我们可以得出结论$f = g$。 重要的是我们在计算过程中动态发现基本函数 Θ。 这种风格的证明可以按照以下方式进行。

$$\begin{array}{ll} & f \cdot \mathsf{L}\, in \\ = & \{ \text{为什么？} \} \\ & \Theta\, f \\ \propto & \{\ x \cdot \mathsf{L}\, in = \Theta\, x \text{有唯一解}\ \} \\ & \Theta\, g \\ = & \{ \text{为什么？} \} \\ & g \cdot \mathsf{L}\, in \end{array}$$

符号 $\propto$意味着连接上部和下部的链接。 总的来说，这个证明证实了 $f = g$。 类似的方法可以用来证明两个伴随展开的相等性。

例子 30.让我们再次展示 $height\, (x \diamond y) = height\, x + height\, y (105)$，这次使用唯一的不动点原理。 为了简洁起见，将 cat和 $height$的定义压缩成单一方程（我们缩写为

$\mathfrak{Empty}$和 $\mathfrak{Push}$用 $\mathfrak{E}$和 $\mathfrak{P}$表示）。

$$
\begin{aligned}
&cat : (\mu\mathfrak{Stack}, \mu\mathfrak{Stack}) \to \mu\mathfrak{Stack} \\
&cat\ \ (In\ s,\quad u) \qquad = \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to u; \mathfrak{P}\,(a,t) \to In\,(\mathfrak{P}\,(a, cat\,(t,u)))\} \\
&height : \mu\mathfrak{Stack} \to Nat \\
&height\ \ (In\ s) \ = \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to 0; \mathfrak{P}\,(a,t) \to 1 + height\ t\}
\end{aligned}
$$

以逐点方式表达计算会得到一个更有吸引力的证明，
其过程如下：

$$
\begin{aligned}
& height\,(cat\,(In\ s, u)) \\
=\ & \{\ cat\text{的定义}\} \\
& height\,(\mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to u; \mathfrak{P}\,(a,t) \to In\,(\mathfrak{P}\,(a, cat\,(t,u)))\}) \\
=\ & \ \{\ \mathbf{case}\text{-fusion}\ \} \\
& \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to height\ u; \mathfrak{P}\,(a,t) \to height\,(In\,(\mathfrak{P}\,(a, cat\,(t,u))))\} \\
=\ & \{\ height\text{的定义}\} \\
& \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to height\ u; \mathfrak{P}\,(a,t) \to 1 + height\,(cat\,(t,u))\} \\
\propto\ & \ \{\ x\,(In\ s, u) = \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to height\ u; \mathfrak{P}\,(a,t) \to 1 + x\,(t,u)\}\ \} \\
& \mathbf{case}\ s\ \mathbf{of}\ \{\mathfrak{E} \to height\ u; \mathfrak{P}\,(a,t) \to 1 + (height\ t + height\ u)\} \\
=\ & \{\ (Nat, 0, +)\text{是一个幺半群}\ \} \\
& \text{情况 } s \text{到 } \{\mathfrak{E} \to 0 + \text{高度} u; \mathfrak{P}\,(a,t) \to (1 + \text{高度} t) + \text{高度} u\} \\
=\ & \ \{\text{情况-融合}\ \} \\
& (\text{情况 } s \text{到 } \{\mathfrak{E} \to 0; \mathfrak{P}\,(a,t) \to 1 + \text{高度} t\}) + \text{高度} u \\
=\ & \ \{\text{高度的定义}\ \} \\
& \text{高度}\ \ (\text{在} s) + \text{高度} u \quad .
\end{aligned}
$$

注意 (连接 (在s, u))=高度 (在s)+高度u是逐点版本的 ·连接 · L in =加 ·(高度 × 高度) · L in. 同样地,情况-融合是join-fusion (21)的逐点变体, $k \cdot (g_1 \triangledown g_2) = k \cdot g_1 \triangledown k \cdot g_2$.

证明很简短而且简洁，每一步都是或多或少被迫的。 此外，中心步骤，即应用单子律，非常明显。 沿途我们重新发现了函数 $height2$。 它还作为原始证明中的链接，证明了 $height \cdot cat = height2 = plus \cdot (height \times height)$需要两步。 新的证明格式将两个独立的证明合并为一个。 ⊔⊓

从表面上看，上面的证明非常接近传统的归纳证明，其中 in标记归纳论证， $\propto$ 标记归纳假设的应用。 事实上，在 **Set**的情况下，唯一的不动点证明可以很容易地转化为归纳证明。 （反之亦然，如果归纳证明证明了两个函数的相等性： $\forall x\ .f\,x\ = g\,x$。 ）然而，唯一的不动点原理对于底层范畴是不可知的，而且对于共归纳类型同样适用。

练习*52.*证明 (栈,空,$\diamond$)是一个幺半群:

$$\begin{aligned} &空 \quad \diamond s = s = s \diamond 空 \quad , \\ &(s \diamond t) \diamond u = s \diamond (t \diamond u) \ , \end{aligned}$$

或者，用无点风格写成

$$\begin{aligned} &猫 \cdot (空 \vartriangle 身份) = 身份 = 猫 \cdot (身份 \vartriangle 空) \ , \\ &猫 \cdot (猫 \times 身份) = 猫 \cdot (身份 \times 猫) \cdot assocr \ , \end{aligned}$$

其中 $assocr : (A \times B) \times C \cong A \times (B \times C)$是嵌套乘积之间的标准同构。

□

4 进一步阅读

本节提供了一些关于我们主题的背景知识，包括进一步阅读的参考资料。 更深入地了解相关工作可以在文章“Adjoint Folds and Unfolds—An Extended Study ” [11]中找到。

范畴论。 范畴论的三位一体——范畴、函子和自然变换——是由艾伦伯格和麦克莱恩发现的。第一篇独立研究范畴的论文出现在1945年[34]，该论文探讨了自然同构的概念，非常值得一读。. 范畴论的权威参考书是麦克莱恩的杰作[8]。范畴论的入门教材包括Awodey [35]、Barr and Wells [36]和Pierce [37]。

伴随的概念是由丹尼尔・坎在1958年引入的[7]。 Ry-deheard [38]以自由构造一个幺半群作为运行示例，解决了一些练习题，阐述了这一概念。 （在20世纪80年代和90年代，曾举办过一系列关于“范畴论和计算机科学”的会议；这篇论文是第一次会议的教程贡献，当时被称为“范畴论和计算机编程”。） Spivey [39]探索了Bird关于列表理论[40, 41]的范畴背景，后来被称为Bird-Meertens形式化。 我们对伴随的计算处理受到了Fokkinga和Meertens的论文[42]的启发。

递归方案。 关于递归方案或者‘态射’有大量的研究成果。. 利用范畴论中的函子和自然变换的概念，Malcolm [43]将Bird-Meertens形式主义推广到任意的数据类型。 他的工作假设底层范畴为**S**et，并且被Meijer等人[5]改编为范畴**C**po。后一篇论文还普及了现在著名的术语catamorphism和anamorphism（用于折叠和展开），以及香蕉和镜头括号（(–)和[(–)]）。 Fokkinga [44]通过mutumorphisms捕捉了相互递归的函数。Pardo [45]提出了解决‘append问题’的另一种方法：他引入了带参数的折叠，并使用它们来实现通用累积。在Hagino [46]、Malcolm [43]和许多其他人的工作基础上，Bird和de Moor在他们的经典教材[9]中给出了“编程代数”的全面介绍。

of Hagino [46], Malcolm [43] and many others, Bird and de Moor gave a comprehensive account of the "Algebra of Programming" in their seminal textbook [9].
教科书特别强调了一种关系方法来进行程序构建——从范畴的概念到更丰富的寓言结构。

嵌套数据类型及其表达能力的发现[24, 47, 48]引发了一系列的研究。 嵌套数据类型上的标准折叠，通过构造是自然变换，被认为表达能力不够。

Bird和Paterson在论文"嵌套数据类型的广义折叠"[31]中通过向折叠添加额外参数来解决了这个问题，从而引出了广义折叠的概念。 这些讲义的第二部分实际上是基于他们的工作。 为了证明广义折叠是唯一定义的，他们讨论了确保更一般的方程 $x \cdot \mathsf{L}\, in = \Psi\, x$，即我们的伴随初始不动点方程，唯一定义 x的条件。 对于这个问题提供了两个解决方案，其中第二个解决方案需要 L有一个右伴随。 他们还证明了右Kan扩张是前合成的右伴随。

Mendler [21]提出了一种替代的类型论方法来处理（共）归纳类型。 他的归纳组合子 R^{μ}和 S^{ν}将基本函数映射到其唯一的不动点。 强正规化由基本函数的多态类型保证。 De Bruin [49]给出了Mendler风格递归的第一个范畴论证明。

其他递归方案。我们已经证明许多递归方案都属于伴随（非）折叠的范畴。 然而，我们不能合理地期望伴随（非）折叠包含所有现有的态射类型。 例如，标准折叠的一个大部分正交扩展是来自余态射的递归方案[50，51]。 简而言之，给定一个余态射 N和一个分配律 $\alpha : \mathsf{F} \circ \mathsf{N} \dot{\to} \mathsf{N} \circ \mathsf{F}$，我们可以定义一个箭头 $f = (\mathsf{N}\, in \cdot \alpha)) : \mu\mathsf{F} \to \mathsf{N}\,(\mu\mathsf{F})$，它将一个数据结构扩展开。然后在未知数 $x : \mu\mathsf{F} \to A$ 的方程中，

$$x \cdot in = a \cdot \mathsf{F}\,(\mathsf{N}\, x \cdot f)\ ,$$

对于每个代数 $a : \mathsf{F}\,(\mathsf{N}\, A) \to A$，它有唯一的解。 这个方案包括所谓的histomorphisms作为一个特例（斐波那契函数是一个histomorphism的例子）。

我们已经注意到初始代数和最终余代数是不同的实体。
事实上，μF和νF在一般情况下不兼容，这不幸地导致我们不能自由地组合折叠（消费者）和展开（生产者）。
摆脱这个困境的方法是使用基于递归余代数的hylomorphisms作为结构化递归方案[52]。 简而言之，如果对于每个代数$\langle A\,,\, a\rangle$，方程式在未知数$x : C(\,\mathscr{C}, A)$中有解，则一个余代数$\langle C,\, c\rangle$被称为递归的。

$$x = a \cdot \mathsf{G}\, x \cdot c\ , \tag{109}$$

有一个唯一的解决方案。 这个方程捕捉了分而治之计算模式的特点：问题被分成子问题（c），子问题被递归地解决（$\mathsf{G}\, x$），最后将子解决方案组合成一个

单一解决方案（a）。 唯一定义的箭头 x被称为$hylomorphism$。 Hy-lomorphisms比adjoint folds更具表现力。 然而，这种增加的表现力是有代价的。 然而，Hylomorphisms有时会遇到实际问题，即找到一个合适的控制函子（G上面）很困难，详见[11]进行更深入的比较。

类型融合。 初始代数方法是数据类型语义的基础，起源于Lambek [13]在范畴中的不动点工作。. Lambek建议格理论为范畴论中的结果提供了一个富有成果的灵感来源。 这个观点被Backh$ouse$ [53]接受，他将一些格理论的不动点规则推广到范畴论中。 一个重要的定律是类型融合，它允许我们将一个函子的应用与一个初始代数融合，形成另一个初始代数。

$$\mathsf{L}\,(\mu\mathsf{F}) \cong \mu\mathsf{G} \quad \Longleftarrow \quad \mathsf{L}\circ\mathsf{F} \cong \mathsf{G}\circ\mathsf{L} \ .$$

同构的证人 $\mathsf{L}\,(\mu\mathsf{F}) \cong \mu\mathsf{G}$可以被定义为
(伴随) 不动点方程的解。 使用类型融合，例如，可以展示

$$\mu\text{列表自然数} \cong \mu\text{栈} \ ,$$

这使我们能够将函数 $total$和 $sums$联系起来。 论文[11]中包含了更多的例子，并展示了伴随（非）折叠和类型融合之间的密切联系。

5 结论

伴随因其普遍性而在范畴论中被证明是最重要的思想之一。 许多数学构造都被证明是形成伴随的伴随函子，麦克莱恩[8, p.vii]曾经说过，“伴随函子无处不在。”

计算机科学可能也是如此。 每个基本类型或类型构造器——初始对象、终结对象、和、积、指数、自由代数、自由余代数——都源自于一个伴随。 伴随在表达中具有惊人的简洁性，将引入、消除、β和η规则结合在一个语句中。 事实上，可以通过使用伴随来定义适当的话语范畴，例如，如果存在以下伴随：$\Delta \dashv 1$，$\Delta \dashv \times$，和 $- \times X \dashv (\text{-})X$ 对于每个选择的 X，则该范畴被称为笛卡尔闭范畴。

伴随折叠和展开在表达能力和使用便捷性之间取得了良好的平衡。我们已经证明许多 Haskell 函数适用于这个范畴。
机制很简单：给定一个（共）递归函数，我们将其抽象出递归调用，并额外删除保护这些调用的 in 和 out 出现。 在集合中，终止和生产力由结果基本函数的自然性条件保证。 伴随的范畴概念在这个发展中起着核心作用。 从某种意义上说，每个伴随都捕捉了不同的递归方案——累积参数、相互

递归，多态递归嵌套数据类型等等，并允许方案被视为伴随（不）折叠的一个实例。

致谢。 我感谢Na*te* F*oste*r，J*ere*my Gibb*ons*，J*os´e* P*edro* Magalh˜aes和Nicolas Wu为校对这些讲义的几个草稿提供的帮助。 他们发现了许多错误，并提出了许多关于风格和表达的改进意见。

参考文献

1. Jones, M.P.：构造器类的系统：重载和隐式高阶多态。 函数式编程杂志 **5**（1）（1995年1月）1-352. Wadler, P.：自由的定理！在第四届国际函数式编程语言和计算机体系结构会议（FPCA'89）上，伦敦，英国，Addison-Wesley Publishing Company（1989年9月）347-3593. Wadler, P.：理解单子。 在1990年ACM会议上的LISP和函数式编程，尼斯，ACM Press（1990年6月）61-784. McBride, C.，Paterson, R.：功能珍珠：应用编程与效果。

函数式编程期刊 **18**(1) (2008) 1–13
5. Meijer, E., Fokkinga, M., Paterson, R.: 香蕉、镜头、信封和铁丝的函数式编程。 在Hughes, J., 编辑：第5届ACM函数式编程语言和计算机体系结构会议，FPCA'91，美国马萨诸塞州剑桥。计算机科学讲座笔记的第523卷，Springer Berlin / Heidelberg（1991）124-144

6. Gibbons, J., Jones, G.: 被低估的展开。 在Felleisen, M., Hudak, P., Queinnec, C., 编辑：第三届ACM SIGPLAN国际函数式编程会议，ACM Press（1998）273-2797. Kan, D.M.: 伴随函子。 美国数学学会交易**87**(2) (1958) 294-329

8. Mac Lane, S.: 工作数学家的范畴. 第二版. 数学研究文献. 施普林格出版社, 柏林 (1998)
9. Bird, R., De Moor, O.: 程序代数. 欧洲普林斯顿大学出版社, 伦敦 (1997)

10. Hinze, R.: 一个范畴论入门 (2010) `www.cs.ox.ac.uk/ralf.hinze/SSGIP10/Notes.pdf`.
11. Hinze, R.: 伴随折叠和展开-一个扩展研究. 计算机科学的科学编程 (2011) 即将出版.
12. Mulry, P.: 强单子, 代数和不动点. 在Fourman, M., Johnstone, P., Pitts, A., 编辑: 范畴在计算机科学中的应用. 剑桥大学出版社 (1992)
13. Lambek, J.: 完全范畴的不动点定理. 数学杂志. **103** (1968) 151–161
14. Hinze, R., James, D.W.H.: 理由同构！在Oliveira, B.C., Zalewski, M., eds.：第6届ACM SIGPLAN泛型编程研讨会论文集 (WGP '10)，纽约，美国，ACM（2010年9月）85–96
15. Peyton Jones, S.：Haskell 98语言和库。剑桥大学出版社 (2003)
16. Cockett, R., Fukushima, T.：关于Charity。黄色系列报告 92/480/18，卡尔加里大学计算机科学系（1992年6月）

17. Coq开发团队：Coq证明助手参考手册（2010年）
http://coq.inria.fr.
18. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.：初始代数语义和连续代数。 ACM期刊 **24**(1)（1977年1月）68–9519. Smyth, M.B., Plotkin, G.D.：递归域方程的范畴论解决方案。 SIAM计算期刊 **11**(4)（1982年）761–783

20. Sheard, T., Pasalic, T.: 两级类型和参数化模块。 函数式编程杂志 **14**(5) (2004年9月) 547–587
21. Mendler, N.P.: 二阶λ演算中的归纳类型和类型约束。 纯粹与应用逻辑学杂志 **51**(1–2) (1991) 159–17222. Gim´enez, E.: 使用递归方案编码受保护的定义。 在Dybjer, P.,Nordstr¨om, B., Smith, J.M., eds.: 证明和程序的类型，国际Workshop TYPES'94, B ˚astad, Sweden, June 6-10, 1994, Selected Papers. Volume996 of Lecture Notes in Computer Science., Springer-Verlag (1995) 39–5923. Okasaki, C.: 纯函数数据结构。剑桥大学出版社(1998)

24. Bird, R., Meertens, L.: 嵌套数据类型。 在Jeuring, J.主编：第四届国际程序构造数学会议，MPC'98，瑞典马尔斯特兰德。计算机科学讲座笔记的第1422卷，Springer Berlin /Heidelberg（1998年6月）52-67

25. Mycroft, A.：多态类型方案和递归定义。 在Paul, M.,Robinet, B.主编：国际编程研讨会论文集，第6届学术讨论会，法国图卢兹。计算机科学讲座笔记的第167卷。（1984年）217-228

26. Hinze, R., Peyton Jones, S.：可导出的类型类。 在Hutton, G.主编：2000年ACM SIGPLAN Haskell研讨会论文集。电子理论计算机科学笔记的第41（1）卷，Elsevier Science（2001年8月）5-35。初步会议记录出现在诺丁汉大学技术报告中。

27. Trifonov, V.: 模拟量化类约束。 在：Haskell '03：2003年ACM SIGPLAN Haskell研讨会论文集，美国纽约，ACM（2003）98-102

28. L¨ammel, R., Peyton Jones, S.: 使用类扩展泛型函数。 在Pierce, B.（编）：2005年国际函数编程会议论文集，爱沙尼亚塔林，2005年9月26日-28日。（2005年9月）

29. Meertens, L.: Paramorphisms. 计算的形式方面 **4**（1992）413-424
30. Backhouse, R., Bijsterveld, M., Van Geldrop, R., Van der Woude, J.: 类别作为一致构造格理论（2003）工作文档，可从http://www.cs.nott.ac.uk/˜rcb/MPC/CatTheory.ps.gz获得。
31. Bird, R., Paterson, R.: 通用嵌套数据类型的折叠。 计算的形式方面 **11**(2) (1999) 200–222
32. Hinze, R.: 高效的通用折叠。 在Jeuring, J., ed.: 第二届通用编程研讨会论文集。 (2000) 1–16 该论文集作为乌特勒支大学的技术报告UU-CS-2000-19出版。

33. Hinze, R.: 程序优化的Kan扩展—或者：art和dan解释一个古老的技巧。 在Gibbons, J., Nogueira, P., eds.: 第11届程序构造数学国际会议(MPC '12)。 Lecture Notes in Computer Science的第7342卷., Springer Berlin / Heidelberg (2012) 324–36234. Eilenberg, S., MacLane, S.: 自然等价的通用理论。 美国数学学会交易 **58**(2) (1945年9月) 231–294

35. Awodey, S.: 范畴论. 第二版. 牛津大学出版社 (2010)
36. Barr, M., Wells, C.: 计算机科学的范畴论. 第三版. Les Publications CRM, Montr´eal (1999) 这本书可以从Centre de recherches math´ematiqueshttp://crm.umontreal.ca/获取.
37. Pierce, B.C.: 计算机科学家的基础范畴论. 麻省理工学院出版社 (1991)
38. Rydeheard, D.: 伴随. 在Pitt, D., Abramsky, S., Poign´e, A., Rydeheard, D., eds.: 范畴论和计算机科学. Lecture Notes in Computer Science. 第240卷., Springer Berlin / Heidelberg (1986) 51–57
39. Spivey, M.: 列表理论的范畴论方法. 在Van de Snepscheut,J., ed.: 程序构造的数学. L ecture Notes inComputer Science. 第375卷., Springer Berlin / Heidelberg (1989) 399 –40840. Bird, R.: 列表理论导论. 在Broy, M., ed.: 逻辑编程和离散设计的NAT
O高级研究所论文集, 德国Marktoberdorf, Springer Berlin / Heidelberg (1987) 5–4241. B ird, R.: 程序推导的函数演算. Technical Report PRG-64,Programming Research Gr oup, 牛津大学计算机实验室 (1987)42. Fokkinga, M.M., Meertens, L.: 伴随. T
echnical Report Memoranda Inf94-31, University of Twente, Enschede, 荷兰 (1994)43. Malcolm, G.: 数据结构和程序转换. 计算机科学的科学 **14**(2–3) (1990) 255–280

44. Fokkinga, M.M.：算法中的法律和秩序。 University of Twente博士论文，(1992年2月)
45. Pardo, A.：通用累积。 在Gibbons, J., Jeuring, J., eds.： Proceedings of the IFIP TC2 Working Conference on Generic Programming, Schloss Dagstuhl. Volume 243., Kluwer Academic Publishers (2002年7月) 49–78
46. Hagino, T.：具有范畴类型构造的类型化λ演算。 在Pitt, D.,Poigne, A., Rydeheard, D., eds.： Category Theory and Computer Science. Lecture Notes in Computer Scie nce的第283卷。 (1987年)
47. Connelly, R.H., Morris, F.L.：trie数据结构的一般化。 计算机科学中的数学结构 **5**(3) (1995年9月) 381–41848. Okasaki, C.：可连接的双端队列。 在：1997年ACM
SIGPLAN国际函数式编程会议论文集，阿姆斯特丹，荷兰 (1997年6月) 66–74*ACM SIG PLAN Notices,*32(8), 1997年8月.

49. De Bruin, P.J.: 在构造性语言中的归纳类型。 博士论文，格罗宁根大学（1995年）

50. Uustalu, T., Vene, V., Pardo, A.: 从余单子到递归方案。Nordic J. of Computing **8** (2001年9月) 366–390
51. Bartels, F.: 广义共归纳。Math. Struct. in Comp. Science **13** (2003) 321—-348
52. Capretta, V., Uustalu, T., Vene, V.: 从余单子到递归共代数。 Information and Computation **204**(4) (2006) 437–468
53. Backhouse, R., Bijsterveld, M., Van Geldrop, R., Van der Woude, J.: 范畴固定点演算。 In Pitt, D., Rydeheard, D.E., Johnstone, P., eds.: Proceedingsof the 6th Intern ational Conference on Category Theory and Computer Science(CTCS '95), Cambri dge, UK. Volume 953 of Lecture Notes in Computer Science.,Springer-Verlag (Aug ust 1995) 159–179