

计算概论A—实验班

函数式程序设计

Functional Programming

胡振江, 张伟

北京大学 信息科学技术学院 计算机科学技术系

2021年09~2021年12月

初见函数式思维

两句很有哲理的话

A. To a man with a hammer, everything looks like a nail

B. 工欲善其事，必先利其器

- 1. 思维方式是一种工具**
- 2. 不能被思维方式束缚**

函数式思维 是一种什么样的思维方式

- 使用数学中的函数作为求解信息处理问题的基本成分
- “使用方式”包括：
 - 从零开始定义一些基本函数
 - 把已有的函数组装起来，形成新的函数

简要回顾：数学中的函数

定义 1.1.1 (函数) 给定定义域 X 、值域 Y ，称两者之间的一个关系 $f \subseteq X \times Y$ 是一个函数，当且仅当如下条件成立：

$$\forall x \in X : |\{(x, y) \mid (x, y) \in f\}| = 1 \quad (1.1)$$

也就是说：给定函数 f ，对于定义域中的任何一个元素 x ，在值域中都存在且仅存在一个元素 y 与之对应。

简要回顾：函数相关的表示符号

- ▶ $X \times Y$: 一个集合，其定义为 $\{(x, y) \mid x \in X, y \in Y\}$.
- ▶ $X \rightarrow Y$: 一个由函数构成的集合，包含且仅包含所有以 X 为定义域、以 Y 为值域的函数.
- ▶ $f : X \rightarrow Y$: 等价于 $f \in X \rightarrow Y$ ，表示 f 是一个函数，其定义域为 X ，值域为 Y ；也称 $X \rightarrow Y$ 为函数 f 的类型. 在一般意义上，如果 $x \in X$ ，我们也称 x 的类型为 X ，记为 $x : X$ ；同时，也称 X 为类型 X .
- ▶ $f(x)$: 函数 f 中与定义域中的元素 x 相对应的那个值². 显然，这个值是函数 f 值域中的一个元素.

简要回顾：常用的集合符号

▶ \mathbb{N} : 自然数集合

• $\mathbb{N}^+ \doteq \{e \mid e \in \mathbb{N}, e \neq 0\}$: 非 0 自然数集合

▶ \mathbb{Z} : 整数集合

• $\mathbb{Z}^+ \doteq \{e \mid e \in \mathbb{Z}, e > 0\}$: 显然可知 $\mathbb{Z}^+ = \mathbb{N}^+$

• $\mathbb{Z}^- \doteq \{e \mid e \in \mathbb{Z}, e < 0\}$

▶ \mathbb{Q} : 有理数集合

▶ \mathbb{R} : 实数集合

▶ $\mathbb{B} \doteq \{T, F\}$: 布尔值集合. 其中, T 表示真值, F 表示假值

▶ \mathbb{C} : 字符集合

简要回顾：数学中的函数

定义 1.1.2 (函数的像) 给定函数 $f : X \rightarrow Y$, 它的像 (image), 记为 $\text{img}(f)$, 是一个集合, 其定义如下:

$$\text{img}(f) \doteq \{f(x) \mid x \in X\} \quad (1.2)$$

显然可知, 对任意函数 $f : X \rightarrow Y$, 满足 $\text{img}(f) \subseteq Y$.

简要回顾：数学中的函数

定义 1.1.3 (函数的组合) 给定两个函数 $f : Y \rightarrow Z$ 和 $g : X \rightarrow Y$, 满足 $Y^- \subseteq Y$, 则这两个函数的组合 $f \cdot g$ 是一个函数, 其定义如下:

$$\begin{aligned} f \cdot g & : X \rightarrow Z & (1.3) \\ (f \cdot g)(x) & \doteq f(g(x)) \end{aligned}$$

给定函数 $f : X \rightarrow Y$, 如果 $X = X_1 \times X_2 \times \cdots \times X_n$, 则称 f 为一个 n 元函数.

为什么在函数的基础上，可以形成一种思维方式

A. 函数可以建模 **变换** 和 **因果关系**

B. 信息处理问题，本质上是一种信息的变换问题

C. 在面向特定领域问题的软件应用中，大量涉及对物理世界中因果关系的仿真

几个简单的函数

逻辑非 函数

$$\mathit{not} : \mathbb{B} \rightarrow \mathbb{B}$$

$$\mathit{not} \doteq \{T \mapsto F, F \mapsto T\}$$

$$\mathit{not} : \mathbb{B} \rightarrow \mathbb{B}$$

$$\mathit{not}(x) \doteq \begin{cases} T & \text{if } x = F \\ F & \text{if } x = T \end{cases}$$

$$\mathit{not} : \mathbb{B} \rightarrow \mathbb{B}$$

$$\mathit{not}(T) \doteq F$$

$$\mathit{not}(F) \doteq T$$

逻辑与 函数

$$\mathit{and} : \mathbb{B} \rightarrow (\mathbb{B} \rightarrow \mathbb{B})$$

$$\mathit{and}(\mathsf{T})(\mathsf{T}) \doteq \mathsf{T}$$

$$\mathit{and}(\mathsf{T})(\mathsf{F}) \doteq \mathsf{F}$$

$$\mathit{and}(\mathsf{F})(\mathsf{T}) \doteq \mathsf{F}$$

$$\mathit{and}(\mathsf{F})(\mathsf{F}) \doteq \mathsf{F}$$

$$\mathit{and}' : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$$

$$\mathit{and}'(\mathsf{T}, \mathsf{T}) \doteq \mathsf{T}$$

$$\mathit{and}'(\mathsf{T}, \mathsf{F}) \doteq \mathsf{F}$$

$$\mathit{and}'(\mathsf{F}, \mathsf{T}) \doteq \mathsf{F}$$

$$\mathit{and}'(\mathsf{F}, \mathsf{F}) \doteq \mathsf{F}$$

$$\mathit{and} : \mathbb{B} \rightarrow (\mathbb{B} \rightarrow \mathbb{B})$$

$$\mathit{and}(x)(y) \doteq \begin{cases} \mathsf{T} & \text{if } x = y = \mathsf{T} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

$$\mathit{and}' : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$$

$$\mathit{and}'(x, y) \doteq \begin{cases} \mathsf{T} & \text{if } x = y = \mathsf{T} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

后继函数

$$\mathit{succ} : \mathbb{N} \rightarrow \mathbb{N}$$

$$\mathit{succ}(x) \doteq x + 1$$

- 上面这个定义不是特别好，它利用了一个更复杂的操作符
- 更本源的定义方式如下：

$$\mathit{succ} \doteq \{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3, \dots\}$$

后继函数

$$n \doteq \underbrace{(succ \cdot succ \cdot succ \cdots succ)}_{n \text{ succ}}(0)$$

- 对于任何一个自然数n
 - 它只不过是“n个succ函数组合后作用到0上”的一种符号表示而已

加函数

$$\mathit{plus} \quad : \quad \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{plus}(n)(0) \quad \doteq \quad n$$

$$\mathit{plus}(n)(\mathit{succ}(m)) \quad \doteq \quad \mathit{succ}(\mathit{plus}(n)(m))$$

加函数：示例

$$\mathit{plus} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{plus}(n)(0) \doteq n$$

$$\mathit{plus}(n)(\mathit{succ}(m)) \doteq \mathit{succ}(\mathit{plus}(n)(m))$$

$$\begin{aligned} &= \mathit{plus}(3)(4) \\ &= \mathit{plus}(3)(\mathit{succ}(3)) \\ &= \mathit{succ}(\mathit{plus}(3)(3)) \\ &= \mathit{succ}(\mathit{plus}(3)(\mathit{succ}(2))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{plus}(3)(2))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{plus}(3)(\mathit{succ}(1)))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{plus}(3)(1)))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{plus}(3)(\mathit{succ}(0))))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{plus}(3)(0))))) \\ &= \mathit{succ}(\mathit{succ}(\mathit{succ}(\mathit{succ}(3)))) \\ &= (\mathit{succ} \cdot \mathit{succ} \cdot \mathit{succ} \cdot \mathit{succ})(3) \end{aligned}$$

加 函数

$$plus : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$plus(n)(0) \doteq n$$

$$plus(n)(succ(m)) \doteq succ(plus(n)(m))$$

- 不要被上面这种看似复杂的定义所困扰
- 它只不过用递归定义的方式表达了一件很简单的事情

$$plus(m)(n) = \underbrace{(succ \cdot succ \cdots succ)}_{n \text{ succ functions}}(m)$$

乘函数

$$\mathit{mult} \quad : \quad \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{mult}(n)(0) \quad \doteq \quad 0$$

$$\mathit{mult}(n)(\mathit{succ}(m)) \quad \doteq \quad \mathit{plus}(n)(\mathit{mult}(n)(m))$$

乘函数：示例

$$\mathit{mult} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{mult}(n)(0) \doteq 0$$

$$\mathit{mult}(n)(\mathit{succ}(m)) \doteq \mathit{plus}(n)(\mathit{mult}(n)(m))$$

$$\begin{aligned} & \mathit{mult}(3)(4) \\ = & \mathit{mult}(3)(\mathit{succ}(3)) \\ = & \mathit{plus}(3)(\mathit{mult}(3)(3)) \\ = & \mathit{plus}(3)(\mathit{mult}(3)(\mathit{succ}(2))) \\ = & \mathit{plus}(3)\mathit{plus}(3)(\mathit{mult}(3)(2)) \\ = & \mathit{plus}(3)\mathit{plus}(3)(\mathit{mult}(3)(\mathit{succ}(1))) \\ = & \mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)(\mathit{mult}(3)(1)) \\ = & \mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)(\mathit{mult}(3)(\mathit{succ}(0))) \\ = & \mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)(\mathit{mult}(3)(0)) \\ = & \mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)\mathit{plus}(3)(0) \\ = & (\mathit{plus}(3) \cdot \mathit{plus}(3) \cdot \mathit{plus}(3) \cdot \mathit{plus}(3))(0) \end{aligned}$$

乘函数

$$\mathit{mult} \quad : \quad \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{mult}(n)(0) \doteq 0$$

$$\mathit{mult}(n)(\mathit{succ}(m)) \doteq \mathit{plus}(n)(\mathit{mult}(n)(m))$$

- 不要被上面这种看似复杂的定义所困扰
- 它只不过用递归定义的方式表达了一件很简单的事情

$$\mathit{mult}(m)(n) = \underbrace{(\mathit{plus}(m) \cdot \mathit{plus}(m) \cdots \mathit{plus}(m))}_{n \text{ } \mathit{plus}(m) \text{ functions}}(0)$$

指数函数

$$\mathit{expn} \quad : \quad \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\mathit{expn}(m)(0) \quad \doteq \quad 1$$

$$\mathit{expn}(m)(\mathit{succ}(n)) \quad \doteq \quad \mathit{mult}(m)(\mathit{expn}(m)(n))$$

指数函数：示例

$$\text{expn} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\text{expn}(m)(0) \doteq 1$$

$$\text{expn}(m)(\text{succ}(n)) \doteq \text{mult}(m)(\text{expn}(m)(n))$$

$$\text{expn}(2)(3)$$

$$= \text{expn}(2)(\text{succ}(2))$$

$$= \text{mult}(2)(\text{expn}(2)(2))$$

$$= \text{mult}(2)(\text{expn}(2)(\text{succ}(1)))$$

$$= \text{mult}(2)(\text{mult}(2)(\text{expn}(2)(1)))$$

$$= \text{mult}(2)(\text{mult}(2)(\text{expn}(2)(\text{succ}(0))))$$

$$= \text{mult}(2)(\text{mult}(2)(\text{mult}(2)(\text{expn}(2)(0))))$$

$$= \text{mult}(2)(\text{mult}(2)(\text{mult}(2)(1)))$$

$$= (\text{mult}(2) \cdot \text{mult}(2) \cdot \text{mult}(2))(1)$$

指数函数

$$\text{expn} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$\text{expn}(m)(0) \doteq 1$$

$$\text{expn}(m)(\text{succ}(n)) \doteq \text{mult}(m)(\text{expn}(m)(n))$$

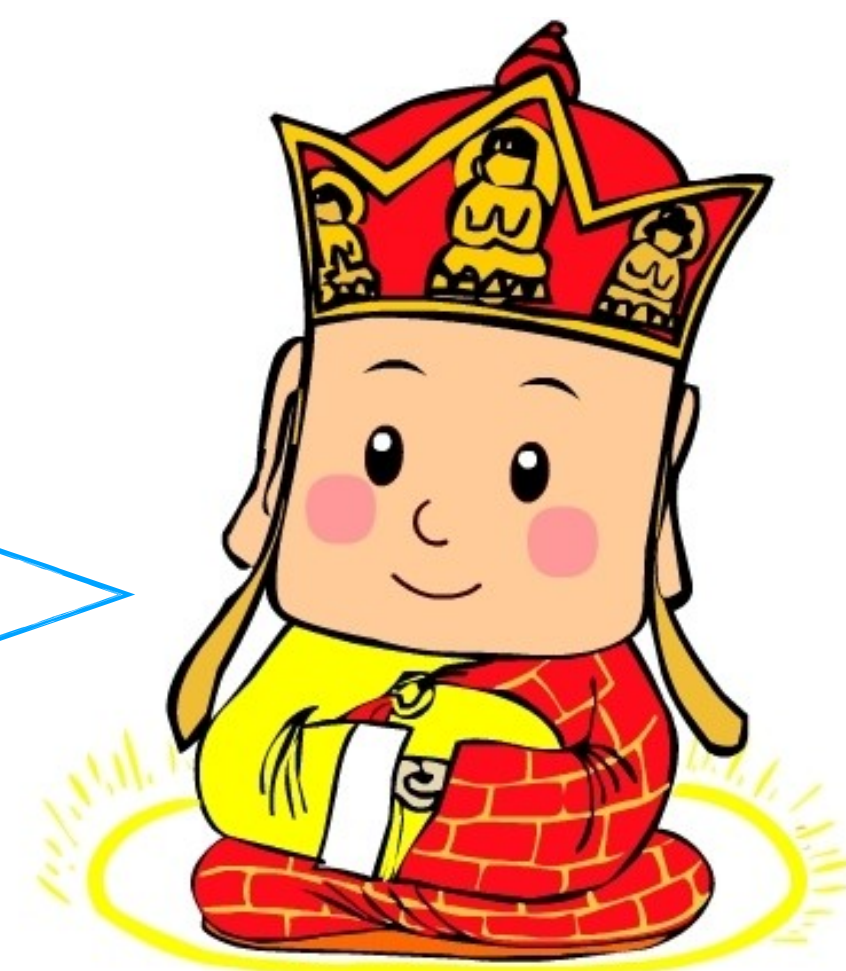
- 不要被上面这种看似复杂的定义所困扰
- 它只不过用递归定义的方式表达了一件很简单的事情

$$\text{expn}(m)(n) = \underbrace{(\text{mult}(m) \cdot \text{mult}(m) \cdots \text{mult}(m))}_{n \text{ mult}(m) \text{ functions}}(1)$$

总是 n 个相同函数的组合
能不能有些新东西呢



何必让自己这么累
这样划水不挺好嘛



阶乘函数

$$fact : \mathbf{N} \rightarrow \mathbf{N}$$

$$fact(0) \doteq 1$$

$$fact(succ(n)) \doteq mult(succ(n))(fact(n))$$

阶乘函数：示例

$$fact : \mathbb{N} \rightarrow \mathbb{N}$$

$$fact(0) \doteq 1$$

$$fact(succ(n)) \doteq mult(succ(n))(fact(n))$$

$$fact(3)$$

$$= fact(succ(2))$$

$$= mult(3)(fact(2))$$

$$= mult(3)(fact(succ(1)))$$

$$= mult(3)(mult(2)(fact(1)))$$

$$= mult(3)(mult(2)(fact(succ(0))))$$

$$= mult(3)(mult(2)(mult(1)(fact(0))))$$

$$= mult(3)(mult(2)(mult(1)(1)))$$

$$= (mult(3) \cdot mult(2) \cdot mult(1))(1)$$

阶乘 函数

$$fact : \mathbb{N} \rightarrow \mathbb{N}$$

$$fact(0) \doteq 1$$

$$fact(succ(n)) \doteq mult(succ(n))(fact(n))$$

- 不要被上面这种看似复杂的定义所困扰
- 它只不过用递归定义的方式表达了一件很简单的事情

$$fact(n) = \underbrace{(mult(n) \cdot mult(n-1) \cdot mult(n-2) \cdots mult(1))}_{n \text{ } mult(_) \text{ functions}}(1)$$

看，是不是有那么一点点新东西了 😊

斐波那契 函数

$$fib : \mathbf{N} \rightarrow \mathbf{N}$$

$$fib(0) \doteq 0$$

$$fib(1) \doteq 1$$

$$fib(succ(succ(n))) \doteq plus(fib(n))(fib(succ(n)))$$

斐波那契 函数： 示例

$fib(5)$

$= plus(fib(4))(fib(3))$

$= plus(plus(fib(3))(fib(2)))(plus(fib(2))(fib(1)))$

$= plus(plus(fib(3))(fib(2)))(plus(fib(2))(1))$

$= plus(plus(plus(fib(2))(fib(1)))(plus(fib(1))(fib(0))))(plus(plus(fib(1))(fib(0)))(1))$

$= plus(plus(plus(fib(2))(1))(plus(1)(0)))(plus(plus(1)(0))(1))$

$= plus(plus(plus(plus(fib(1))(fib(0)))(1))(plus(1)(0)))(plus(plus(1)(0))(1))$

$= plus(plus(plus(plus(1)(0))(1))(plus(1)(0)))(plus(plus(1)(0))(1))$



这下好了，没有规律了，看你怎么圆过来

嘿嘿嘿嘿



自然数上的 fold 函数

plus、mult、expn 这三个函数之间存在共性
这种共性可以被封装在一个函数中

foldn 函数

A在前面没有定义过，在这里表示一个集合变量 (set variable)
即：任何集合都可以用来替换A

$$\text{foldn} \quad : \quad (A \rightarrow A) \rightarrow (A \rightarrow (\mathbb{N} \rightarrow A))$$

$$\text{foldn}(h)(c)(0) \quad \doteq \quad c$$

$$\text{foldn}(h)(c)(\text{succ}(n)) \quad \doteq \quad h(\text{foldn}(h)(c)(n))$$

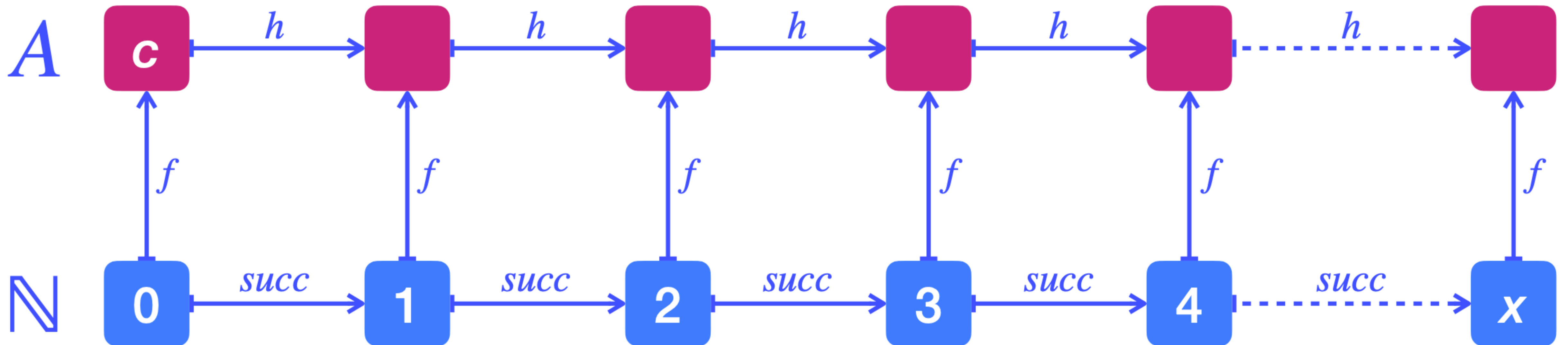
- 可知 $h : A \rightarrow A$
 $c : A$

foldn 函数

▶ 给定函数 $h : A \rightarrow A$ 、值 $c : A$ ，令 $f = \text{fold}(h)(c)$ ，由上述定义可知：

- $f(0) = c$.
- $f(\text{succ}(n)) = h(f(n))$.

$$\begin{aligned} \text{foldn} & : (A \rightarrow A) \rightarrow (A \rightarrow (\mathbb{N} \rightarrow A)) \\ \text{foldn}(h)(c)(0) & \doteq c \\ \text{foldn}(h)(c)(\text{succ}(n)) & \doteq h(\text{foldn}(h)(c)(n)) \end{aligned}$$



如果你没有体会到这个图蕴含的“美”，我们再换一个角度去理解 $foldn$ 函数

- ▶ 给定一个自然数 n ，可知：

$$n = \underbrace{(succ \cdot succ \cdots succ)}_{n \text{ succ functions}}(0)$$

- ▶ 已知 $f = foldn(h)(c)$ ，则可知：

$$f(n) = \underbrace{(h \cdot h \cdots h)}_{n \text{ h functions}}(c)$$

利用 $foldn$ 函数，可以对上面引入的三个函数 $plus, mult, expn$ 进行更简洁的定义

$$plus : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$plus(n) \doteq foldn(succ)(n)$$

$$mult : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$mult(n) \doteq foldn(plus(n))(0)$$

$$expn : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$$

$$expn(n) \doteq foldn(mult(n))(1)$$

$$m = \underbrace{(succ \cdot succ \cdots succ)}_{m \text{ succ functions}}(0)$$

$$plus(n)(m) = \underbrace{(succ \cdot succ \cdots succ)}_{m \text{ succ functions}}(n)$$

$$mult(n)(m) = \underbrace{(plus(n) \cdot plus(n) \cdots plus(n))}_{m \text{ plus}(n) \text{ functions}}(0)$$

$$expn(n)(m) = \underbrace{(mult(n) \cdot mult(n) \cdots mult(n))}_{m \text{ mult}(n) \text{ functions}}(1)$$

为了使用 *foldn* 函数定义 *fact* 和 *fib* 函数，首先引入两个辅助函数

$$\mathit{outl} \quad : \quad A \times B \rightarrow A$$

$$\mathit{outl}(a, b) \quad \doteq \quad a$$

$$\mathit{outr} \quad : \quad A \times B \rightarrow A$$

$$\mathit{outr}(a, b) \quad \doteq \quad b$$

fact 函数

$$f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$$

$$f(m, n) \doteq (m + 1, (m + 1) \times n)$$

$$fact : \mathbb{N} \rightarrow \mathbb{N}$$

$$fact \doteq outr \cdot (foldn(f)(0, 1))$$

$$m = \underbrace{(succ \cdot succ \cdots succ)}_{m \text{ succ functions}}(0)$$

$$fact(m) = outr(\underbrace{(f \cdot f \cdots f)}_{m \text{ f functions}}(0, 1))$$

fib 函数

$$g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$$

$$g(m, n) \doteq (n, m + n)$$

$$fib : \mathbb{N} \rightarrow \mathbb{N}$$

$$fib \doteq outl \cdot (foldn(g)(0, 1))$$

$$m = \underbrace{(succ \cdot succ \cdots succ)}_{m \text{ succ functions}}(0)$$

$$fib(m) = outl(\underbrace{(g \cdot g \cdots g)}_{m \text{ g functions}}(0, 1))$$

序列 (List) 以及 序列上的 fold 函数

在信息处理问题中，经常涉及一组按照某种顺序排列的数据。我们将这类数据称为序列或序列类型的数据。例如，对于排序问题而言，待排序的数据通常是采用序列的方式进行组织的，排序的结果自然也是以序列的方式返回的。

List 的表示符号

给定一个集合 A ，我们使用符号 $[A]$ 表示一个新的集合，其包含且仅包含所有由 0 到多个 A 中的元素形成的序列。例如， $[\mathbb{N}]$ 这个集合中包含且仅包含所有由 0 到多个自然数形成的序列。下面，我们以自然数序列为例，给出本章对序列的一些表示方式。

- ▶ $[\]$ 表示由零个自然数形成的序列，也即：空序列。
- ▶ $[1]$ 或 $1 \succ [\]$ 表示由一个自然数 1 形成的序列。
- ▶ $[1, 2, 2, 3, 3, 3]$ 或 $1 \succ 2 \succ 2 \succ 3 \succ 3 \succ 3 \succ [\]$ 表示由六个相应的自然数形成的序列。

List 相关的函数

$$\mathit{cons} : A \rightarrow ([A] \rightarrow [A])$$

$$\mathit{cons}(n)(ns) \doteq n \succ ns$$

$$\mathit{len} : [A] \rightarrow \mathbb{N}$$

$$\mathit{len}([]) \doteq 0$$

$$\mathit{len}(n \succ ns) \doteq 1 + \mathit{len}(ns)$$

List 相关的函数

$$\mathit{rev} : [A] \rightarrow [A]$$

$$\mathit{revm} : [A] \rightarrow ([A] \rightarrow [A])$$

$$\mathit{rev} \doteq \mathit{revm}([])$$

$$\mathit{revm}(xs)([]) \doteq xs$$

$$\mathit{revm}(xs)(y \succ ys) \doteq \mathit{revm}(y \succ xs)(ys)$$

List 相关的函数

$concat : [A] \rightarrow ([A] \rightarrow [A])$

$concat([])(ns) \doteq ns$

$concat(m \succ ms)(ns) \doteq m \succ (concat(ms)(ns))$

List 相关的函数

$$\mathit{filter} : (A \rightarrow \mathbb{B}) \rightarrow ([A] \rightarrow [A])$$

$$\mathit{filter}(p)([]) \doteq []$$

$$\mathit{filter}(p)(n \succ ns) \doteq \mathit{concat}(\mathit{sel}(p(n))([n])([]))(\mathit{filter}(p)(ns))$$

$$\mathit{sel} : \mathbb{B} \rightarrow (A \rightarrow (A \rightarrow A))$$

$$\mathit{sel}(b)(a_1)(a_2) \doteq \begin{cases} a_1 & \text{if } b = \text{T} \\ a_2 & \text{if } b = \text{F} \end{cases}$$

对于自然数集合 \mathbb{N} ，存在一个对应的 **fold** 函数；在前文中我们将其命名为 *foldn*。很多与自然数相关的函数都可以在 *foldn* 函数的基础上进行定义。那么，对于序列集合 $[A]$ ，是否也存在一个对应的 **fold** 函数呢？

时间不早了，我们直接公布答案。是的，在序列集合 $[A]$ 上，确实存在这样的 **fold** 函数。而且，更有趣的是，存在两个这样的 **fold** 函数。不妨将这两个函数分别命名为 *foldlr* 和 *foldll*。这两个名称中分别包含三个词根。前两个词根是相同的：第一个词根“**fold**”的含义很明显；第二个词根“**l**”表示“**list**”，含义也很明显。第三个词根分别为“**l**”和“**r**”，表示“**left**”和“**right**”。

foldl 函数

$$\mathit{foldl} \quad : \quad (A \rightarrow (B \rightarrow B)) \rightarrow (B \rightarrow ([A] \rightarrow B))$$

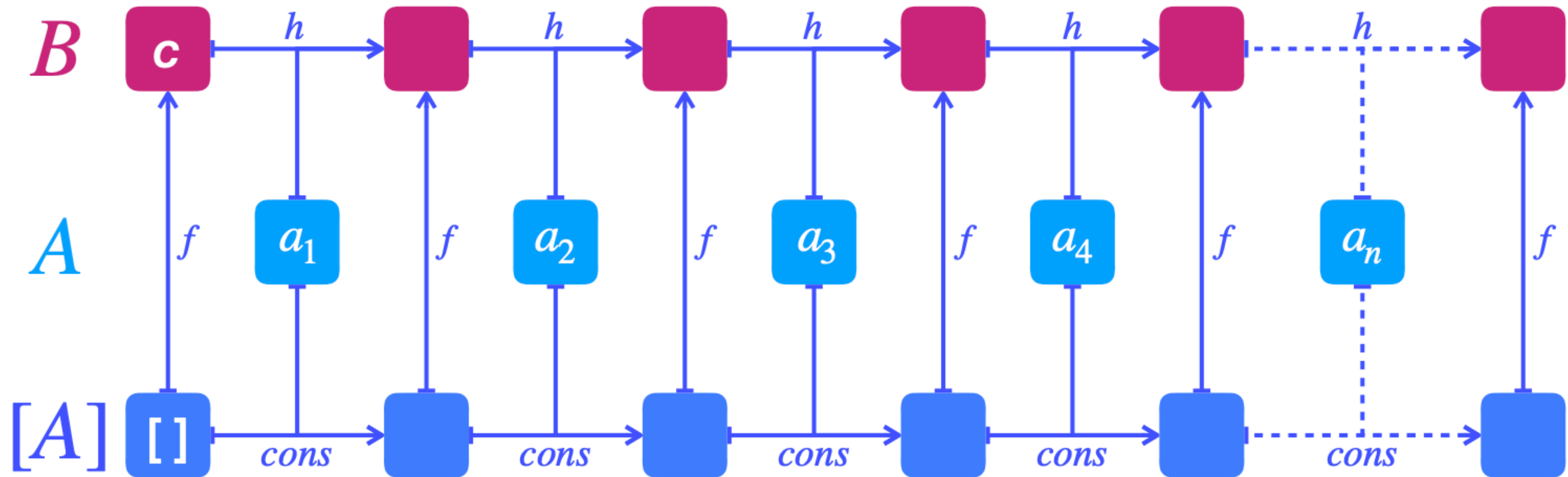
$$\mathit{foldl}(h)(c)([]) \quad \doteq \quad c$$

$$\mathit{foldl}(h)(c)(x \succ xs) \quad \doteq \quad h(x)(\mathit{foldl}(h)(c)(xs))$$

foldlr 函数

$$\begin{aligned} \text{foldlr} & : (A \rightarrow (B \rightarrow B)) \rightarrow (B \rightarrow ([A] \rightarrow B)) \\ \text{foldlr}(h)(c)([]) & \doteq c \\ \text{foldlr}(h)(c)(x \succ xs) & \doteq h(x)(\text{foldlr}(h)(c)(xs)) \end{aligned}$$

给定 $h : A \rightarrow (B \rightarrow B), c : B$, 令 $f = \text{foldlr}(h)(c)$, 图 1.2 对 foldlr 函数的效果进行了更直观的解释.



foldl 函数

如果你没有体会到上图蕴含的“美”，那么，我们再换一个角度去理解 *foldl* 函数。

- ▶ 给定 $xs : [A]$ ，不失一般性，令 $xs = a_n \succ a_{n-1} \succ \dots \succ a_1 \succ []$ ，可知：

$$xs = \underbrace{(cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1))}_{n \text{ cons}(_) \text{ functions}}([])$$

- ▶ 已知 $f = foldl(h)(c)$ ，则可知：

$$f(xs) = \underbrace{(h(a_n) \cdot h(a_{n-1}) \cdot h(a_{n-2}) \cdots h(a_1))}_{n \text{ h}(_) \text{ functions}}(c)$$

foldll 函数

$$\mathit{foldll} \quad : \quad (A \rightarrow (B \rightarrow B)) \rightarrow (B \rightarrow ([A] \rightarrow B))$$

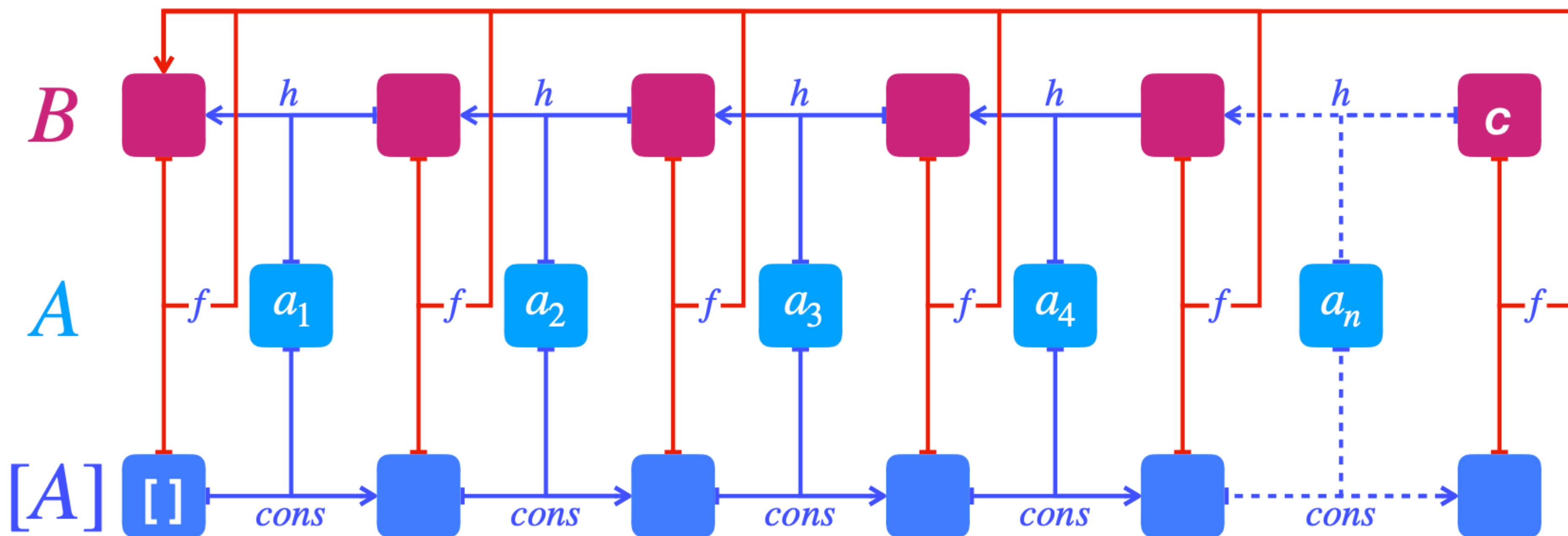
$$\mathit{foldll}(h)(c)([]) \quad \doteq \quad c$$

$$\mathit{foldll}(h)(c)(x \succ xs) \quad \doteq \quad \mathit{foldll}(h)(h(x)(c))(xs)$$

foldll 函数

$$\begin{aligned} \text{foldll} & : (A \rightarrow (B \rightarrow B)) \rightarrow (B \rightarrow ([A] \rightarrow B)) \\ \text{foldll}(h)(c)([]) & \doteq c \\ \text{foldll}(h)(c)(x \succ xs) & \doteq \text{foldll}(h)(h(x)(c))(xs) \end{aligned}$$

给定 $h : A \rightarrow (B \rightarrow B), c : B$, 令 $f = \text{foldll}(h)$, 图 1.3 对 foldll 函数的效果进行了更直观的解释.



foldll 函数

如果你没有体会到上图蕴含的“美”，那么，我们再换一个角度去理解 *foldll* 函数。

- ▶ 给定 $xs : [A]$ ，不失一般性，令 $xs = a_n \succ a_{n-1} \succ \dots \succ a_1 \succ []$ ，可知：

$$xs = \underbrace{(cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1))}_{n \text{ cons}(_) \text{ functions}}([])$$

- ▶ 则可知：

$$foldll(h)(c)(xs) = \underbrace{(h(a_1) \cdot h(a_2) \cdot h(a_3) \cdots h(a_n))}_{n \text{ h}(_) \text{ functions}}(c)$$

List 相关函数的重定义

len 函数

$$\mathit{len} : [A] \rightarrow \mathbb{N}$$

$$\mathit{len} \doteq \mathit{foldlr}(h)(0)$$

$$h : A \rightarrow \mathbb{N} \rightarrow \mathbb{N}$$

$$h(a)(n) \doteq n + 1$$

$$\begin{aligned} \mathit{len} &: [A] \rightarrow \mathbb{N} \\ \mathit{len}([]) &\doteq 0 \\ \mathit{len}(n \succ ns) &\doteq 1 + \mathit{len}(ns) \end{aligned}$$

$$xs = \underbrace{(\mathit{cons}(a_n) \cdot \mathit{cons}(a_{n-1}) \cdot \mathit{cons}(a_{n-2}) \cdots \mathit{cons}(a_1))}_{n \text{ cons}(_) \text{ functions}}([])$$

$$\mathit{len}(xs) = \underbrace{(h(a_n) \cdot h(a_{n-1}) \cdot h(a_{n-2}) \cdots h(a_1))}_{n \text{ h}(_) \text{ functions}}(0)$$

rev 函数

$rev : [A] \rightarrow [A]$

$rev \doteq foldll(cons)([])$

$rev : [A] \rightarrow [A]$

$revm : [A] \rightarrow ([A] \rightarrow [A])$

$rev \doteq revm([])$

$revm(xs)([]) \doteq xs$

$revm(xs)(y \succ ys) \doteq revm(y \succ xs)(ys)$

$xs = \underbrace{(cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1))}_{n \text{ cons}(_) \text{ functions}}([])$

$rev(xs) = \underbrace{(cons(a_1) \cdot cons(a_2) \cdot cons(a_3) \cdots cons(a_n))}_{n \text{ cons}(_) \text{ functions}}([])$

concat 函数

$concat : [A] \rightarrow ([A] \rightarrow [A])$

$concat(xs)(ys) \doteq foldlr(cons)(ys)(xs)$

$concat : [A] \rightarrow ([A] \rightarrow [A])$

$concat([])(ns) \doteq ns$

$concat(m \succ ms)(ns) \doteq m \succ (concat(ms)(ns))$

$xs = \underbrace{(cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1))}_{n \text{ cons}(_) \text{ functions}}([])$

$concat(xs)(ys) = \underbrace{(cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1))}_{n \text{ cons}(_) \text{ functions}}(ys)$

filter 函数

$$\begin{aligned} \mathit{filter} & : (A \rightarrow \mathbb{B}) \rightarrow ([A] \rightarrow [A]) \\ \mathit{filter}(p) & \doteq \mathit{foldlr}(f(p))([]) \\ f & : (A \rightarrow \mathbb{B}) \rightarrow (A \rightarrow ([A] \rightarrow [A])) \\ f(p)(a) & \doteq \mathit{sel}(p(a))(\mathit{cons}(a))(\mathit{id}) \\ \mathit{id} & : A \rightarrow A \\ \mathit{id}(a) & \doteq a \end{aligned}$$

$$\begin{aligned} \mathit{filter} & : (A \rightarrow \mathbb{B}) \rightarrow ([A] \rightarrow [A]) \\ \mathit{filter}(p)([]) & \doteq [] \\ \mathit{filter}(p)(n \succ ns) & \doteq \mathit{concat}(\mathit{sel}(p(n))([n])([]))(\mathit{filter}(p)(ns)) \\ \mathit{sel} & : \mathbb{B} \rightarrow (A \rightarrow (A \rightarrow A)) \\ \mathit{sel}(b)(a_1)(a_2) & \doteq \begin{cases} a_1 & \text{if } b = \top \\ a_2 & \text{if } b = \text{F} \end{cases} \end{aligned}$$

$$\begin{aligned} xs & = \underbrace{(\mathit{cons}(a_n) \cdot \mathit{cons}(a_{n-1}) \cdot \mathit{cons}(a_{n-2}) \cdots \mathit{cons}(a_1))}_{n \text{ cons}(_) \text{ functions}}([]) \\ \mathit{filter}(p)(xs) & = \underbrace{(f(p)(a_n) \cdot f(p)(a_{n-1}) \cdot f(p)(a_{n-2}) \cdots f(p)(a_1))}_{n \text{ f}(_)(_) \text{ functions}}([]) \end{aligned}$$

一种排序算法

$$qsort : [\mathbf{N}] \rightarrow [\mathbf{N}] \quad (1.47)$$

$$qsort([]) \doteq []$$

$$qsort(n \succ ns) \doteq \underline{\text{concat}(\text{concat}(qsort(\text{filter}(lt(n))(ns)))([n]))(qsort(\text{filter}(ge(n))(ns)))}$$

$$lt : \mathbf{N} \rightarrow (\mathbf{N} \rightarrow \mathbf{B}) \quad (1.48)$$

$$lt(n)(m) \doteq \begin{cases} \mathbf{T} & \text{if } m < n \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$ge : \mathbf{N} \rightarrow (\mathbf{N} \rightarrow \mathbf{B}) \quad (1.49)$$

$$ge(n)(m) \doteq \text{not}(lt(n)(m))$$

如果这就是用FP书写的算法

此生绝不学FP



好孩子，如果给你
三生三世的财富，学否？



我信你个鬼
你这个糟老头坏得很

内容 VS 形式

让我们再深入思考一下. 这个问题在本质上是一个关于“内容”与“形式”两者之间关系的问题. 在这里, “内容”指的是“对自然数序列进行排序的一种方法”; “形式”指的是“表现这种排序方法的形式”. 进一步而言, 这个问题可以表述为: “形式”小于“内容”. 或者可以更直接的阐述为: 内容是很好的, 但形式实在是太糟糕了. 如果你能体会到这一点, 那么, 你会发现, 这个问题的严重程度并不像表面上看起来的那样. 为什么这么说呢? 因为, 本质 (内容) 毕竟还是很好的.

重走长征路

在某种意义上，我们正在“重走长征路”。在很多年以前，很多聪明的科研工作者就已经意识到了这个问题。然后，在这个问题的驱使下，科研工作者们设计了各种各样的函数式程序设计语言。我们即将介绍的 **Haskell** 语言，就是这些函数式程序设计语言的集大成者。

初见函数式思维

暂且先到这里吧