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 计算机学院

2023年09～12月

第2.1章：初见函数式思维

两句很有哲理的话

A. 工欲善其事，必先利其器

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

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：

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

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 $\{(x, y) \mid x \in X, y \in Y\}$.
- $X \to Y$：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.
- $f : X \to Y$：等价于 $f \in X \to Y$，表示 f 是一个函数，其定义域为 X，值域为 Y；也称 $X \to Y$ 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/类型
- $\mathbb{R}$：实数集合/类型
- $\mathbb{B} \doteq \{T, F\}$：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 $f: X \to Y$，它的像（image），记为 $img(f)$，是一个集合，其定义如下：

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

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

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 $f: Y \to Z$ 和 $g: X \to Y^{-}$，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：

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

给定函数 $f: X \to Y$，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

$$
\begin{aligned}
\mathit{not} &: \mathbb{B} \to \mathbb{B} \\
\mathit{not} &\doteq \{\mathrm{T} \mapsto \mathrm{F}, \mathrm{F} \mapsto \mathrm{T}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{not} &: \mathbb{B} \to \mathbb{B} \\
\mathit{not}(x) &\doteq \begin{cases} \mathrm{T} & \text{if } x = \mathrm{F} \\ \mathrm{F} & \text{if } x = \mathrm{T} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{not} &: \mathbb{B} \to \mathbb{B} \\
\mathit{not}(\mathrm{T}) &\doteq \mathrm{F} \\
\mathit{not}(\mathrm{F}) &\doteq \mathrm{T}
\end{aligned}
$$

逻辑与 函数

$$\begin{aligned} \mathit{and} &: \mathbb{B} \to (\mathbb{B} \to \mathbb{B}) \\ \mathit{and}(\mathrm{T})(\mathrm{T}) &\doteq \mathrm{T} \\ \mathit{and}(\mathrm{T})(\mathrm{F}) &\doteq \mathrm{F} \\ \mathit{and}(\mathrm{F})(\mathrm{T}) &\doteq \mathrm{F} \\ \mathit{and}(\mathrm{F})(\mathrm{F}) &\doteq \mathrm{F} \end{aligned}$$

$$\begin{aligned} \mathit{and}' &: \mathbb{B} \times \mathbb{B} \to \mathbb{B} \\ \mathit{and}'(\mathrm{T}, \mathrm{T}) &\doteq \mathrm{T} \\ \mathit{and}'(\mathrm{T}, \mathrm{F}) &\doteq \mathrm{F} \\ \mathit{and}'(\mathrm{F}, \mathrm{T}) &\doteq \mathrm{F} \\ \mathit{and}'(\mathrm{F}, \mathrm{F}) &\doteq \mathrm{F} \end{aligned}$$

$$\begin{aligned} \mathit{and} &: \mathbb{B} \to (\mathbb{B} \to \mathbb{B}) \\ \mathit{and}(x)(y) &\doteq \begin{cases} \mathrm{T} & \text{if } x = y = \mathrm{T} \\ \mathrm{F} & \text{otherwise} \end{cases} \end{aligned}$$

$$\begin{aligned} \mathit{and}' &: \mathbb{B} \times \mathbb{B} \to \mathbb{B} \\ \mathit{and}'(x, y) &\doteq \begin{cases} \mathrm{T} & \text{if } x = y = \mathrm{T} \\ \mathrm{F} & \text{otherwise} \end{cases} \end{aligned}$$

后继 函数

$$
\begin{aligned}
succ &: \mathbb{N} \to \mathbb{N} \\
succ(x) &\doteq x + 1
\end{aligned}
$$

- 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：

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

后继 函数

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

- 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

$$
\begin{aligned}
plus &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
plus(n)(0) &\doteq n \\
plus(n)(succ(m)) &\doteq succ(plus(n)(m))
\end{aligned}
$$

加 函数： 示例

$$
\begin{aligned}
plus &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
plus(n)(0) &\doteq n \\
plus(n)(succ(m)) &\doteq succ(plus(n)(m))
\end{aligned}
$$

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

加 函数

$$\begin{aligned} plus &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\ plus(n)(0) &\doteq n \\ plus(n)(succ(m)) &\doteq succ(plus(n)(m)) \end{aligned}$$

- 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

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

乘 函数

$$
\begin{aligned}
mult &: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}) \\
mult(n)(0) &\doteq 0 \\
mult(n)(succ(m)) &\doteq plus(n)(mult(n)(m))
\end{aligned}
$$

乘 函数：示例

$$
\begin{aligned}
mult &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
mult(n)(0) &\doteq 0 \\
mult(n)(succ(m)) &\doteq plus(n)(mult(n)(m))
\end{aligned}
$$

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

乘 函数

$$\begin{aligned} mult &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\ mult(n)(0) &\doteq 0 \\ mult(n)(succ(m)) &\doteq plus(n)(mult(n)(m)) \end{aligned}$$

- 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

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

指数 函数

$$\begin{aligned} expn &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\ expn(m)(0) &\doteq 1 \\ expn(m)(succ(n)) &\doteq mult(m)(expn(m)(n)) \end{aligned}$$

指数 函数： 示例

$$
\begin{aligned}
expn &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
expn(m)(0) &\doteq 1 \\
expn(m)(succ(n)) &\doteq mult(m)(expn(m)(n))
\end{aligned}
$$

$$
\begin{aligned}
& expn(2)(3) \\
= \; & expn(2)(succ(2)) \\
= \; & mult(2)(expn(2)(2)) \\
= \; & mult(2)(expn(2)(succ(1))) \\
= \; & mult(2)(mult(2)(expn(2)(1))) \\
= \; & mult(2)(mult(2)(expn(2)(succ(0)))) \\
= \; & mult(2)(mult(2)(mult(2)(expn(2)(0)))) \\
= \; & mult(2)(mult(2)(mult(2)(1))) \\
= \; & (mult(2) \cdot mult(2) \cdot mult(2))(1)
\end{aligned}
$$

指数 函数

$$
\begin{aligned}
expn &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
expn(m)(0) &\doteq 1 \\
expn(m)(succ(n)) &\doteq mult(m)(expn(m)(n))
\end{aligned}
$$

- 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

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

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

$$
\begin{aligned}
fact &: \mathbb{N} \to \mathbb{N} \\
fact(0) &\doteq 1 \\
fact(succ(n)) &\doteq mult(succ(n))(fact(n))
\end{aligned}
$$

阶乘 函数：示例

$$
\begin{aligned}
fact &: \mathbb{N} \to \mathbb{N} \\
fact(0) &\doteq 1 \\
fact(succ(n)) &\doteq mult(succ(n))(fact(n))
\end{aligned}
$$

$$
\begin{aligned}
& fact(3) \\
= \;& fact(succ(2)) \\
= \;& mult(3)(fact(2)) \\
= \;& mult(3)(fact(succ(1))) \\
= \;& mult(3)(mult(2)(fact(1))) \\
= \;& mult(3)(mult(2)(fact(succ(0)))) \\
= \;& mult(3)(mult(2)(mult(1)(fact(0)))) \\
= \;& mult(3)(mult(2)(mult(1)(1))) \\
= \;& (mult(3) \cdot mult(2) \cdot mult(1))(1)
\end{aligned}
$$

阶乘 函数

$$
\begin{aligned}
fact &\ : \ \mathbb{N} \to \mathbb{N} \\
fact(0) &\ \doteq \ 1 \\
fact(succ(n)) &\ \doteq \ mult(succ(n))(fact(n))
\end{aligned}
$$

- 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

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

看，是不是有那么一点点新东西了😊

斐波那契 函数

$$
\begin{aligned}
fib &: \mathbb{N} \to \mathbb{N} \\
fib(0) &\doteq 0 \\
fib(1) &\doteq 1 \\
fib(succ(succ(n))) &\doteq plus(fib(n))(fib(succ(n)))
\end{aligned}
$$

斐波那契 函数：示例

$$
\begin{aligned}
& fib(5) \\
= \quad & plus(fib(4))(fib(3)) \\
= \quad & plus(plus(fib(3))(fib(2)))(plus(fib(2))(fib(1))) \\
= \quad & plus(plus(fib(3))(fib(2)))(plus(fib(2))(1)) \\
= \quad & plus(plus(plus(fib(2))(fib(1)))(plus(fib(1))(fib(0))))(plus(plus(fib(1))(fib(0)))(1)) \\
= \quad & plus(plus(plus(fib(2))(1))(plus(1)(0)))(plus(plus(1)(0))(1)) \\
= \quad & plus(plus(plus(plus(fib(1))(fib(0)))(1))(plus(1)(0)))(plus(plus(1)(0))(1)) \\
= \quad & plus(plus(plus(plus(1)(0))(1))(plus(1)(0)))(plus(plus(1)(0))(1))
\end{aligned}
$$

这下好了，没有规律了，看你怎么圆过来

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

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

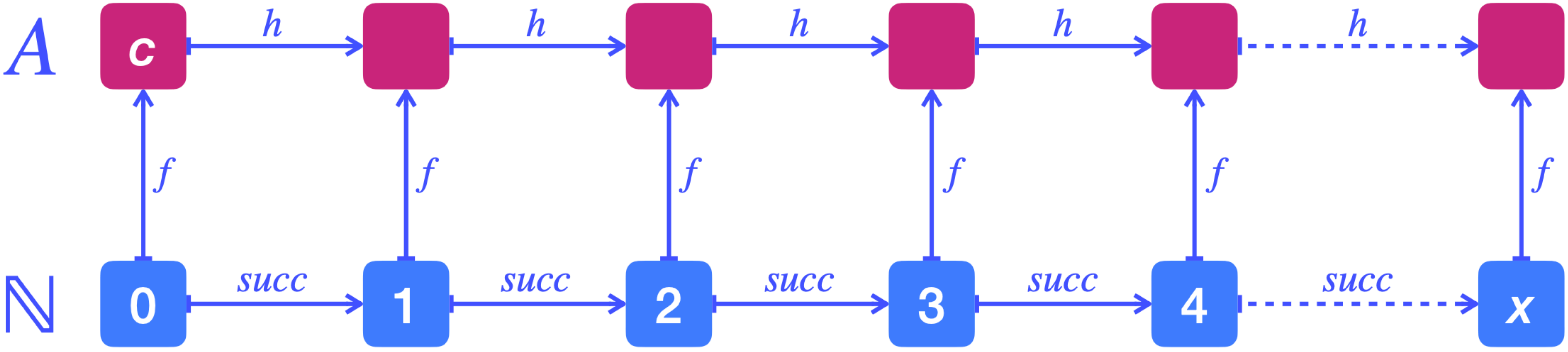
- 可知 $h : A \to A$

  $c : A$

foldn 函数

- 给定函数 $h : A \to A$、值 $c : A$，令 $f = fold(h)(c)$，由上述定义可知：
  - $f(0) = c$.
  - $f(succ(n)) = h(f(n))$.

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



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：

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

- 已知 $f = foldn(h)(c)$，则可知：

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

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

$$
\begin{aligned}
plus &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
plus(n) &\doteq foldn(succ)(n)
\end{aligned}
$$

$$
\begin{aligned}
mult &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
mult(n) &\doteq foldn(plus(n))(0)
\end{aligned}
$$

$$
\begin{aligned}
expn &: \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
expn(n) &\doteq foldn(mult(n))(1)
\end{aligned}
$$

$$
\begin{aligned}
m &= (\underbrace{succ \cdot succ \cdots succ}_{\text{m } succ \text{ functions}})(0) \\
plus(n)(m) &= (\underbrace{succ \cdot succ \cdots succ}_{\text{m } succ \text{ functions}})(n) \\
mult(n)(m) &= (\underbrace{plus(n) \cdot plus(n) \cdots plus(n)}_{\text{m } plus(n) \text{ functions}})(0) \\
expn(n)(m) &= (\underbrace{mult(n) \cdot mult(n) \cdots mult(n)}_{\text{m } mult(n) \text{ functions}})(1)
\end{aligned}
$$

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

$$
\begin{aligned}
outl &: A \times B \to A \\
outl(a, b) &\doteq a
\end{aligned}
$$

$$
\begin{aligned}
outr &: A \times B \to A \\
outr(a, b) &\doteq b
\end{aligned}
$$

$fact$ 函数

$$
\begin{aligned}
f &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N} \\
f(m, n) &\doteq (m + 1, (m + 1) \times n) \\
fact &: \mathbb{N} \to \mathbb{N} \\
fact &\doteq outr \cdot (foldn(f)(0, 1))
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
m &= (\underbrace{succ \cdot succ \cdots succ}_{\text{m } succ \text{ functions}})(0) \\
fact(m) &= outr((\underbrace{f \cdot f \cdots f}_{\text{m } f \text{ functions}})(0, 1))
\end{aligned}
}
$$

fib 函数

$$
\begin{aligned}
g &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N} \\
g(m, n) &\doteq (n, m + n) \\
fib &: \mathbb{N} \to \mathbb{N} \\
fib &\doteq outl \cdot (foldn(g)(0, 1))
\end{aligned}
$$

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

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

$$
\begin{aligned}
cons &\quad : \quad A \to ([A] \to [A]) \\
cons(n)(ns) &\quad \doteq \quad n \succ ns
\end{aligned}
$$

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

List 相关的函数

$$
\begin{aligned}
rev &: [A] \to [A] \\
revm &: [A] \to ([A] \to [A]) \\
rev &\doteq revm([]) \\
revm(xs)([]) &\doteq xs \\
revm(xs)(y \succ ys) &\doteq revm(y \succ xs)(ys)
\end{aligned}
$$

List 相关的函数

$$
\begin{aligned}
concat &: [A] \to ([A] \to [A]) \\
concat([\,])(ns) &\doteq ns \\
concat(m \succ ms)(ns) &\doteq m \succ (concat(ms)(ns))
\end{aligned}
$$

List 相关的函数

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

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”．

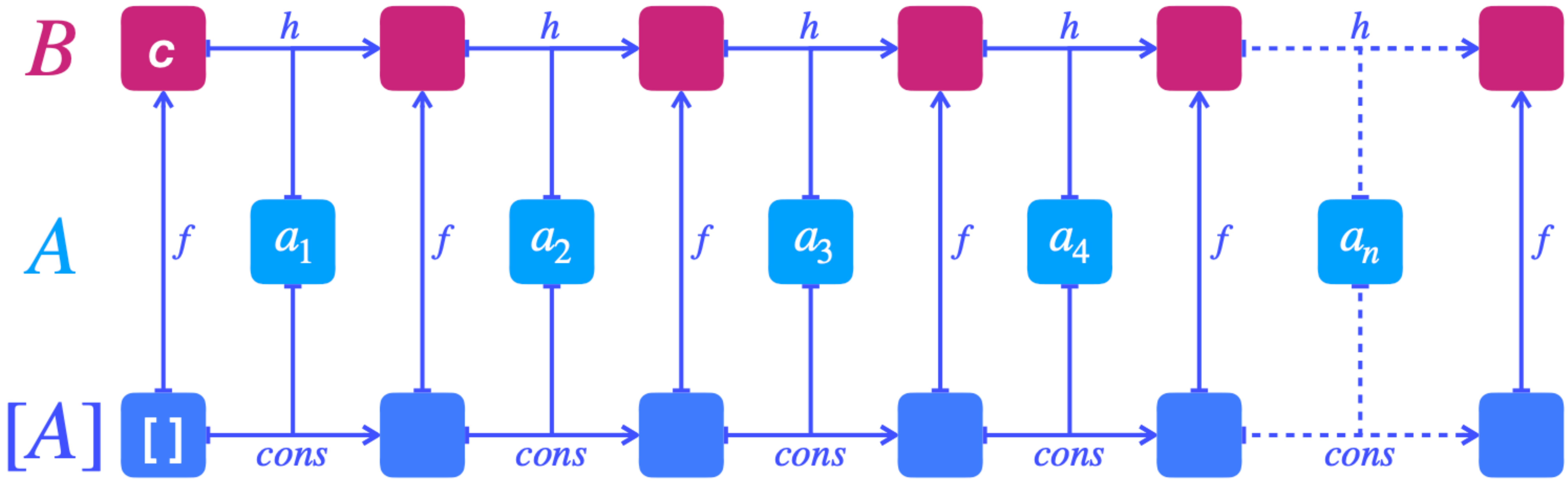
foldlr 函数

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

foldlr 函数

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

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



foldlr 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 $foldlr$ 函数.

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

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

- 已知 $f = foldlr(h)(c)$，则可知：

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

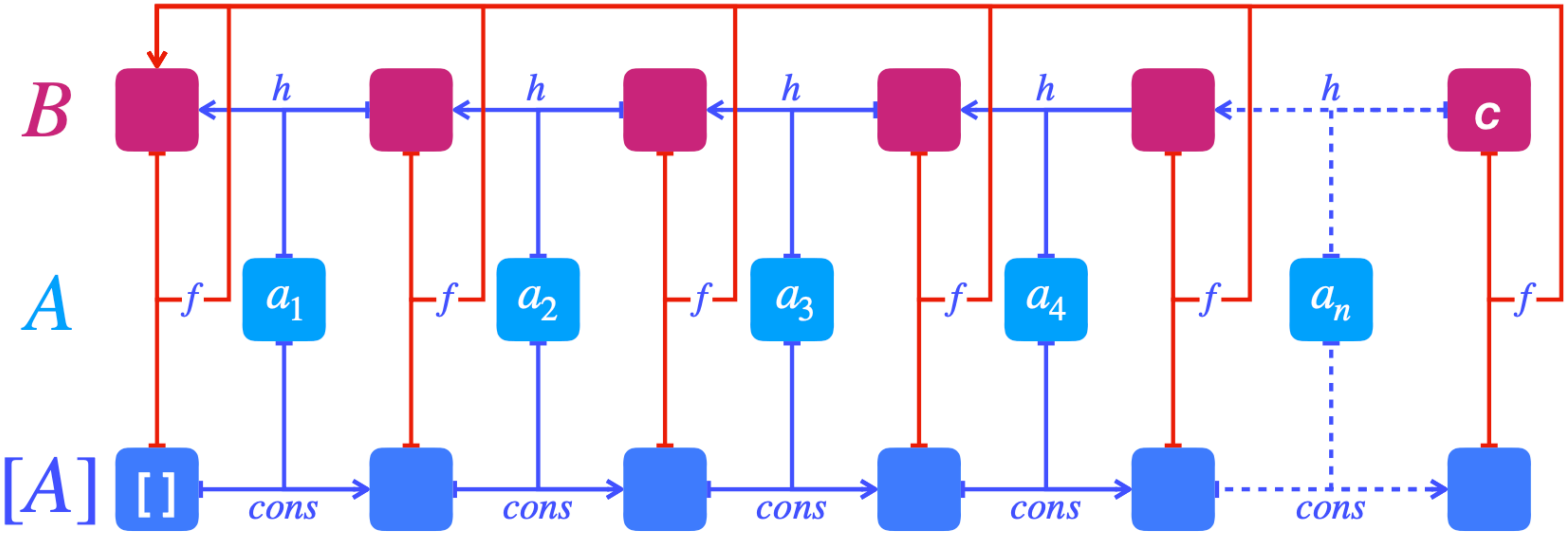
foldll 函数

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

foldll 函数

$$\begin{array}{rcl} foldll & : & (A \to (B \to B)) \to (B \to ([A] \to B)) \\ foldll(h)(c)([\,]) & \doteq & c \\ foldll(h)(c)(x \succ xs) & \doteq & foldll(h)(h(x)(c))(xs) \end{array}$$

给定 $h : A \to (B \to B), c : B$，令 $f = foldll(h)$，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 $xs = a_n \succ a_{n-1} \succ \cdots \succ a_1 \succ []$，可知：

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

- 则可知：

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

List 相关函数的重定义

len 函数

$$
\begin{aligned}
len &: [A] \to \mathbb{N} \\
len &\doteq foldlr(h)(0) \\
h &: A \to \mathbb{N} \to \mathbb{N} \\
h(a)(n) &\doteq n + 1
\end{aligned}
$$

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

$$
\begin{aligned}
xs &= (\underbrace{cons(a_n) \cdot cons(a_{n-1}) \cdot cons(a_{n-2}) \cdots cons(a_1)}_{\text{n } cons(\_) \text{ functions}})([]) \\
len(xs) &= (\underbrace{h(a_n) \cdot h(a_{n-1}) \cdot h(a_{n-2}) \cdots h(a_1)}_{\text{n } h(\_) \text{ functions}})(0)
\end{aligned}
$$

rev 函数

$$rev : [A] \to [A]$$

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

$$\begin{aligned} rev &: [A] \to [A] \\ revm &: [A] \to ([A] \to [A]) \\ rev &\doteq revm([]) \\ revm(xs)([]) &\doteq xs \\ revm(xs)(y \succ ys) &\doteq revm(y \succ xs)(ys) \end{aligned}$$

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

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

concat 函数

$$
\begin{aligned}
concat &: [A] \to ([A] \to [A]) \\
concat(xs)(ys) &\doteq foldlr(cons)(ys)(xs)
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
concat &: [A] \to ([A] \to [A]) \\
concat([])(ns) &\doteq ns \\
concat(m \succ ms)(ns) &\doteq m \succ (concat(ms)(ns))
\end{aligned}
}
$$

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

filter 函数

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

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

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

一种排序算法

$$
\begin{aligned}
qsort &: [\mathbb{N}] \to [\mathbb{N}] \\
qsort([]) &\doteq [] \\
qsort(n \succ ns) &\doteq concat\underline{(concat\overline{(qsort(filter(lt(n))(ns)))([n]))}(qsort(filter(ge(n))(ns))))}
\end{aligned} \tag{1.47}
$$

$$
\begin{aligned}
lt &: \mathbb{N} \to (\mathbb{N} \to \mathbb{B}) \\
lt(n)(m) &\doteq \begin{cases} \mathrm{T} & \text{if } m < n \\ \mathrm{F} & \text{otherwise} \end{cases}
\end{aligned} \tag{1.48}
$$

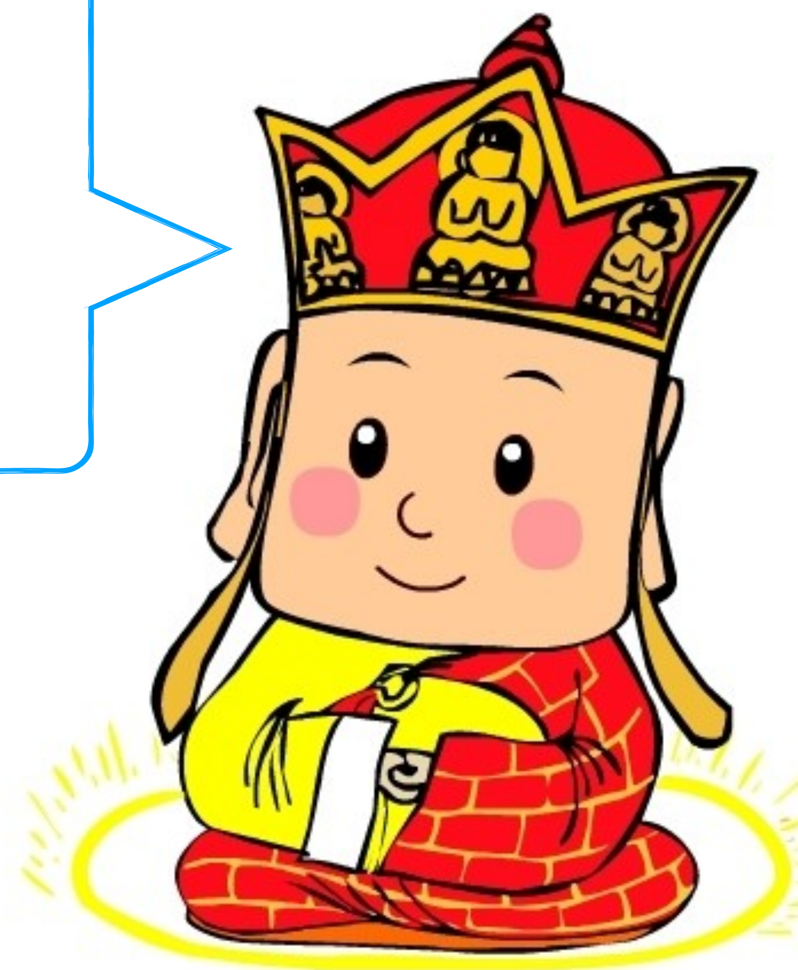
$$
\begin{aligned}
ge &: \mathbb{N} \to (\mathbb{N} \to \mathbb{B}) \\
ge(n)(m) &\doteq not(lt(n)(m))
\end{aligned} \tag{1.49}
$$

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？



内容 VS 形式

- ❖一个关于“内容”与“形式”两者之间关系的问题
  - 内容：对自然数序列进行排序的一种方法
  - 形式：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”；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，科研工作者们就已经意识到了这个问题
- 在这个问题的驱使下，他/她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

第2.1章：初见函数式思维

暂且先到这里吧