代数数据类型通用编程

张淞

网易杭州研究院

2016 年 5 月 1 日

同构的定义

一点看似没用的数学

定义

同构的类型：对于类型 A 与 B，若存在函数 $\psi : A \to B$ 与 $\phi : B \to A$，$\psi \circ \phi = id_B$, $\phi \circ \psi = id_A$ 那么则说类型 A 与 B 同构。

$$\mathbf{data}\ Color = Red \mid Green \mid Blue$$
$$\mathbf{data}\ Level = Low \mid Medium \mid High$$

$$\begin{aligned} &f :: Color \to Level \\ &f\ Red = Low \\ &f\ Green = Medium \\ &f\ Blue = High \end{aligned} \qquad \begin{aligned} &g :: Level \to Color \\ &g\ Low = Red \\ &g\ Medium = Green \\ &g\ High = Blue \end{aligned}$$

同构的性质

- 自反的（reflexive）：$A \simeq A$
  - 证明：$id :: A \to A$
- 对称的（symmetric）：若 $A \simeq B$，那么 $B \simeq A$
  - 证明：由 $A \simeq B$ 得函数 $\psi : A \to B$ 与 $\phi : B \to A$，根据定义显然 $B \simeq A$。
- 传递的（transitive）：若 $A \simeq B$ 且 $B \simeq C$，那么 $A \simeq C$
  - 证明：由 $A \simeq B$ 有 $\psi : A \to B$ 与 $\phi : B \to A$ 且它们的复合是 id；由 $B \simeq C$ 有 $\gamma : B \to C$ 与 $\delta : C \to B$ 且它们的复合是 id。欲证 $A \simeq C$ 则需要构造类型为 $A \to C$ 与 $C \to A$ 的函数，显然它们是 $\gamma \circ \psi$ 与 $\phi \circ \delta$，有兴趣的可以验证一下它们的复合为 id。

代数类型及运算

- **0**，这个类型中没有任何值

  $$\textbf{data } V$$

- **1** 这个类型中只有一个值——U

  $$\textbf{data } U = U$$

- $\times$ 两种类型的笛卡尔积。例如 $Bool \times Level$ 中有 6 个值

  $$\textbf{data } (:*:)\ a\ b = a :*: b$$

- $+$ 两种类型的不相交并。例如 $Bool + Level$ 中有 5 个值，$Bool + Bool$ 中有 4 个值：$L\ True$，$L\ False$，$R\ True$，$R\ False$

  $$\textbf{data } (:+:)\ a\ b = L\ a \mid R\ b$$

显然 **0** 是 $+$ 的单位元，而 **1** 是 $\times$ 的单位元。

相等类型类

$$
\begin{aligned}
&\textbf{data}\ \mathit{Bool} = \mathit{False} \mid \mathit{True}\\
&\textbf{instance}\ \mathit{Eq}\ \mathit{Bool}\ \textbf{where}\\
&\quad \mathit{True} \equiv \mathit{True} = \mathit{True}\\
&\quad \mathit{False} \equiv \mathit{False} = \mathit{True}\\
&\quad \_ \equiv \_ = \mathit{False}\\
&\textbf{data}\ T\ a\ b = Q \mid N\ a\ b\\
&\textbf{instance}\ (\mathit{Eq}\ a, \mathit{Eq}\ b) \Rightarrow \mathit{Eq}\ (T\ a\ b)\ \textbf{where}\\
&\quad Q \equiv Q = \mathit{True}\\
&\quad (N\ \mathit{x1}\ \mathit{y1}) \equiv (N\ \mathit{x2}\ \mathit{y2}) = \mathit{x1} \equiv \mathit{x2} \wedge \mathit{y1} \equiv \mathit{y2}\\
&\quad \_ \equiv \_ = \mathit{False}
\end{aligned}
$$

利用基本代数类型

$$\textbf{type}\ Bool = U :+: U$$

$$\textbf{instance}\ Eq\ U\ \textbf{where}$$

$$\quad U \equiv U = True$$

$$\textbf{instance}\ (Eq\ a, Eq\ b) \Rightarrow Eq\ (a :+: b)\ \textbf{where}$$

$$\quad L\ a \equiv L\ b = a \equiv b$$

$$\quad R\ a \equiv R\ b = a \equiv b$$

$$\quad \_ \equiv \_ = False$$

演示

$$\textbf{instance}\ (Eq\ a, Eq\ b) \Rightarrow Eq\ (a :*: b)\ \textbf{where}$$

$$\quad (a1 :*: b1) \equiv (a2 :*: b2)$$

$$\qquad = a1 \equiv a2 \wedge b1 \equiv b2$$

$$\textbf{data}\ Bool = False \mid True\ \textbf{deriving}\ Show$$

$$\textbf{type}\ Bool = U :+: U$$

$$fromBool :: Bool \rightarrow U :+: U$$

$$fromBool\ False = L\ U$$

$$fromBool\ True = R\ U$$

$$toBool :: U :+: U \rightarrow Bool$$

$$toBool\ (L\ U) = False$$

$$toBool\ (R\ U) = True$$

$$\textbf{class}\ Generic\ a\ \textbf{where}$$
$$\quad \textbf{type}\ Rep\ a :: *$$
$$\quad from :: a \to Rep\ a$$
$$\quad to :: Rep\ a \to a$$
$$\textbf{instance}\ Generic\ Bool\ \textbf{where}$$
$$\quad \textbf{type}\ Rep\ Bool = U :+: U$$
$$\quad from = fromBool$$
$$\quad to = toBool$$

- 用基本代数类型组合出的同构类型
- 它们之间的转换函数

以上两点可以由 GHC 自动完成。

库的设计者想定义一个 GEq 类型类

$$
\begin{aligned}
&\textbf{class } GEq\ a \textbf{ where}\\
&\quad geq :: a \rightarrow a \rightarrow Bool\\
&\quad \textbf{default } geq :: (Generic\ a, GEq\ (Rep\ a)) \Rightarrow a \rightarrow a \rightarrow Bool\\
&\quad geq\ a\ b = geq\ (from\ a)\ (from\ b)\\
&\textbf{instance } (GEq\ a, GEq\ b) \Rightarrow GEq\ (a :+: b) \textbf{ where}\\
&\quad geq\ (L\ a1)\ (L\ a2) = geq\ a1\ a2\\
&\quad geq\ (R\ b1)\ (R\ b2) = geq\ b1\ b2\\
&\quad geq\ \_\ \_ = False\\
&\textbf{instance } (GEq\ a, GEq\ b) \Rightarrow GEq\ (a :*: b) \textbf{ where}\\
&\quad geq\ (a1 :*: b1)\ (a2 :*: b2) = geq\ a1\ a2 \wedge geq\ b1\ b2\\
&\textbf{instance } GEq\ U \textbf{ where}\\
&\quad geq\ U\ U = True
\end{aligned}
$$

由 GHC 生成

$$
\begin{aligned}
&\textbf{instance } \mathit{Generic\ Level}\ \textbf{where}\\
&\quad \textbf{type } \mathit{Rep\ Level} = (U :+: U) :+: U\\
&\quad \mathit{from\ Low} = L\ (L\ U)\\
&\quad \mathit{from\ Medium} = L\ (R\ U)\\
&\quad \mathit{from\ High} = R\ U\\
&\quad \mathit{to}\ (L\ (L\ U)) = \mathit{Low}\\
&\quad \mathit{to}\ (L\ (R\ U)) = \mathit{Medium}\\
&\quad \mathit{to}\ (R\ U) = \mathit{High}
\end{aligned}
$$

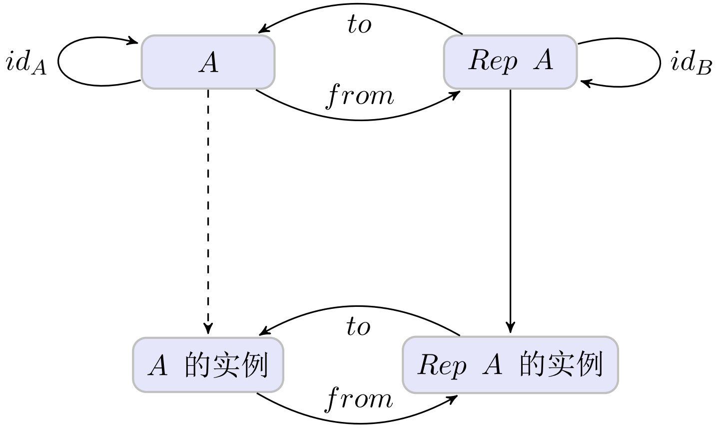
我们需要做的

$$\textbf{instance}\ GEq\ Level$$

借助 GHC7.10 以上的 $DeriveAnyClass$ 扩展：

$$\textbf{data}\ Level = Low \mid Medium \mid High\ \textbf{deriving}\ GEq$$

演示



图：代数数据类型通用编程示意图

问：之前的类型 kind 全为 `*`。对于需要 `* -> *` 的类型类如 `Functor` 怎么办？

答：为 U、$:+:$、$:*:$ 再引入一个参数。

$$\textbf{data}\ \mathit{U1}\ p = \mathit{U1}\ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})$$

$$\textbf{data}\ (:*:)\ a\ b\ p = a\ p :*: b\ p\ \textbf{deriving}\ (\mathit{Eq}, \mathit{Show})$$

$$\textbf{data}\ (:+:)\ a\ b\ p = L\ (a\ p) \mid R\ (b\ p)\ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})$$

$$\textbf{class}\ \mathit{Generic}\ (a :: *)\ \textbf{where}$$

$$\quad \textbf{type}\ \mathit{family}\ \mathit{Rep}\ a :: * \to *$$

$$\quad \mathit{from} :: a \to \mathit{Rep}\ a\ x$$

$$\quad \mathit{to} :: \mathit{Rep}\ a\ x \to a$$

$$\textbf{class}\ \mathit{Generic1}\ (f :: * \to *)\ \textbf{where}$$

$$\quad \textbf{type}\ \mathit{Rep1}\ f :: * \to *$$

$$\quad \mathit{from1} :: f\ p \to \mathit{Rep1}\ f\ p$$

$$\quad \mathit{to1} :: \mathit{Rep1}\ f\ p \to f\ p$$

$$\textbf{newtype}\ \mathit{Par}\ p = \mathit{Par}\ \{\mathit{unPar} :: p\}\ \textbf{deriving}\ \mathit{Show}$$

$$\textbf{newtype}\ \mathit{Rec}\ a\ p = \mathit{Rec}\ \{\mathit{unRec} :: a\ p\}\ \textbf{deriving}\ \mathit{Show}$$

$$\textbf{data}\ List\ a = Nil \mid Cons\ a\ (List\ a)$$

$$\textbf{data}\ Tree\ a = Leaf \mid Node\ a\ (Tree\ a)\ (Tree\ a)$$

$$\textbf{instance}\ Generic1\ List\ \textbf{where}$$

$$\quad \textbf{type}\ Rep1\ List = U1 :+: (Par :*: (Rec\ List))$$

$$\quad from1\ Nil = L\ U1$$

$$\quad from1\ (Cons\ a\ xs) = R\ (Par\ a :*: Rec\ xs)$$

$$\quad to1\ (L\ U1) = Nil$$

$$\quad to1\ (R\ (Par\ a :*: Rec\ xs)) = Cons\ a\ xs$$

$$\textbf{instance}\ Generic1\ Tree\ \textbf{where}$$

$$\quad \textbf{type}\ Rep1\ Tree = U1 :+: (Par :*: (Rec\ Tree) :*: (Rec\ Tree))$$

$$\quad from1\ Leaf = L\ U1$$

$$\quad from1\ (Node\ n\ l\ r) = R\ (Par\ n :*: Rec\ l :*: Rec\ r)$$

$$\quad to1\ (L\ U1) = Leaf$$

$$\quad to1\ (R\ (Par\ n :*: Rec\ l :*: Rec\ r)) = (Node\ n\ l\ r)$$

$$
\begin{array}{l}
\textbf{class}\ \mathit{GFunctor}\ (f :: * \to *)\ \textbf{where} \\
\quad \mathit{gfmap} :: (a \to b) \to (f\ a \to f\ b) \\
\quad \textbf{default}\ \mathit{gfmap} :: (\mathit{Generic1}\ f, \mathit{GFunctor}\ (\mathit{Rep1}\ f)) \Rightarrow \\
\qquad (a \to b) \to (f\ a \to f\ b) \\
\quad \mathit{gfmap}\ f\ x = \mathit{to1}\ (\mathit{gfmap}\ f\ (\mathit{from1}\ x)) \\
\textbf{instance}\ \mathit{GFunctor}\ \mathit{U1}\ \textbf{where} \\
\quad \mathit{gfmap}\ f\ \mathit{U1} = \mathit{U1} \\
\textbf{instance}\ (\mathit{GFunctor}\ a, \mathit{GFunctor}\ b) \Rightarrow \mathit{GFunctor}\ (a :*: b)\ \textbf{where} \\
\quad \mathit{gfmap}\ f\ (a :*: b) = \mathit{gfmap}\ f\ a :*: \mathit{gfmap}\ f\ b \\
\textbf{instance}\ (\mathit{GFunctor}\ a, \mathit{GFunctor}\ b) \Rightarrow \mathit{GFunctor}\ (a :+: b)\ \textbf{where} \\
\quad \mathit{gfmap}\ f\ (L\ a) = L\ (\mathit{gfmap}\ f\ a) \\
\quad \mathit{gfmap}\ f\ (R\ b) = R\ (\mathit{gfmap}\ f\ b) \\
\textbf{instance}\ \mathit{GFunctor}\ \mathit{Par}\ \textbf{where} \\
\quad \mathit{gfmap}\ f\ (\mathit{Par}\ p) = \mathit{Par}\ (f\ p)
\end{array}
$$

$$\textbf{instance}\ GFunctor\ List$$

$$\textbf{instance}\ GFunctor\ Tree$$

借助 GHC7.10 以上的 $DeriveAnyClass$ 扩展可以使用 deriving 关键字导出。

GHC 中的 Generic

$\textbf{newtype}\ \mathit{M1}\ i\ c\ (f :: * \rightarrow *)\ p = \mathit{M1}\ \{\mathit{unM1} :: f\ p\}$

$\textbf{data}\ D$ -- 数据类型标记

$\textbf{data}\ C$ -- 数据构造器标记

$\textbf{data}\ S$ -- 访问器标记

$\textbf{type}\ \mathit{D1} = \mathit{M1}\ D$ -- 说明标记的是数据类型的信息

$\textbf{type}\ \mathit{C1} = \mathit{M1}\ C$ -- 说明标记的是数据构造器的信息

$\textbf{type}\ \mathit{S1} = \mathit{M1}\ S$ -- 说明标记的是访问器函数的信息

$\textbf{class}\ \mathit{Datatype}\ d$ -- 得到类型

$\textbf{class}\ \mathit{Selector}\ s$ -- 得到访问器函数

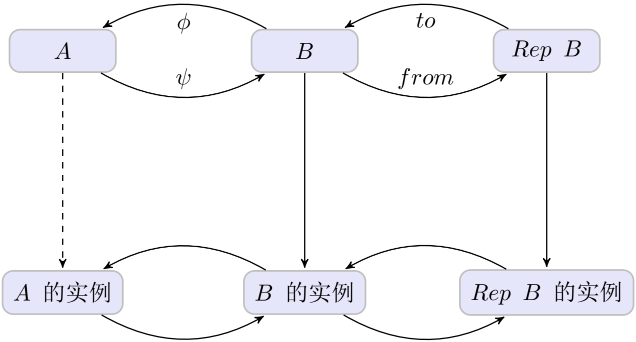
$\textbf{class}\ \mathit{Constructor}\ c$ -- 得到构造器

演示

- 表示其他类型所用到的类型如 $V1$、$U1$、$:+:$、$:*:$ 等等也实现一 Generic 类型类实例。
- 为什么呢？因为 $A \simeq A$，也就是说它们可以用来表示它们自己。

- 对于一些类型可能我们无法实现 $Generic$，比如 $ByteString$
- 其中的一些构造可能是私有的或者可能用了 C 语言的实现

$$pack :: [\,Word8] \to ByteString$$
$$unpack :: ByteString \to [\,Word8]$$



图：代数数据类型通用编程示意图 2

由于同构关系是传递的。

$$
\begin{array}{l}
\textbf{instance } \mathit{Generic}\ \mathit{ByteString}\ \textbf{where} \\
\quad \textbf{type } \mathit{Rep}\ \mathit{ByteString} = \mathit{Rep}\ [\mathit{Word8}] \\
\quad \mathit{from}\ \mathit{bs} = \mathit{from}\ (\mathit{unpack}\ \mathit{bs}) \\
\quad \mathit{to}\ w = \mathit{pack}\ (\mathit{to}\ w)
\end{array}
$$

- 浏览一下 deeqpseq 中 NFData 类型类的实现
- 演示 binary、aeson 库

参考文献

[1] José Predro Magalhães. Less is more, generic programming theory and practice, 2012. `http://www.dreixel.net/research/pdf/thesis.pdf`.

[2] José Predro Magalhães, Atze Dijkstra, Johan Jeuring, and Andres Lőh. A Generic Deriving Mechanism for Haskell. September 2010.