把类型类用 record 实现出来

前排提醒：你或许需要一个游标卡尺。

这是一个类型类教程，尝试使用我改进过的 Literate Agda 后端生成网页 （如果想使用我改进的功能，请拉取 master 分支的 Agda 编译器然后 `cabal install` 编译。 我加的功能的使用说明见这个网页）。

为了让我能用我的手机正常显示自己的文章里的字符，本文会尽量使用非 Unicode 定义。

首先我们定义模块：

```
{-# OPTIONS --no-unicode #-}
{-# OPTIONS --without-K  #-}
module Typeclassopedia where
```

为了强迫自己事先熟悉 HoTT 的 Agda 实现， 我已经开始全面使用 HoTT-Agda 作为编程基础库（和标准库基本一致）。

```
open import lib.Base
```

由于 `lib.types.Bool` 依赖的包比较多（而且 Emacs 看不懂基于 dependent product 的 `Bool`）， 就自己写一个，加上内置定义：

```
data Bool : Type0 where false true : Bool

{-# BUILTIN BOOL  Bool  #-}
{-# BUILTIN FALSE false #-}
{-# BUILTIN TRUE  true  #-}
```

然后常用的函数：

```
not : Bool -> Bool
not true  = false
not false = true
```

比较自然数大小：

```
natEq : Nat -> Nat -> Bool
natEq  O     O    = true
natEq (S _)  O    = false
natEq  O    (S _) = false
natEq (S n) (S m) = natEq n m
{-# BUILTIN NATEQUALS natEq #-}
```

不熟悉这个库的读者不必担心，本文用到的定义都属于看名字就知道的那种（基本上在标准库或者内置库里面都有一个名字一样的，另外 `idp` 是 `refl`）。

泛化一个阶变量（Girard 悖论你好），免得到处写隐式参数：

```
variable i : ULevel
```

有个函数 `of-type` 和 Idris 的函数 `the` 语义一致：先传入一个类型，再传入这个类型的实例:

```
_ : (A : Type i) (u : A) -> A
_ = of-type
```

HoTT-Agda 提供了这种便利的语法来间接使用 `of-type`：

```
_ = 233 :> Nat
_ = lzero :> ULevel
_ = unit :> Unit
_ = idp :> (O == O)
```

实例参数

```
module InstanceArgument where
```

注：我的第一册书里译作*即时*参数，因为在英语中这个名字是一语双关的（ 因为查找实例使用的算法比较快，所以 instance 此处有*即时*之意。 但这个特性又用于查找类型实例，所以 instance 又有*实例*之意）， 现在决定改为取相对来说更内涵的含义。

实例参数类似隐式参数，只不过它们使用两层大括号：

```
  postulate f : {{ x : Nat }} -> Nat
```

手动传值也是两层：

```
  _ = f {{ 233 }} :> Nat
```

隐式参数一般是通过后面的参数往前推导得到的，实例参数则使用另一种方式： **直接在上下文里查找变量。**

听起来很暴力，但其实也没有那么暴力啦，只有：

- 放在 `instance` 代码块里的定义
- 局部变量

才属于被查找的对象，我们称之为实例。 Agda wiki 说构造器也会，但目前看编译器的行为应该是不允许了。

放在 `instance` 代码块里的定义的例子：

```
-- instance 代码块定义
instance rua : Nat
         rua = 233
-- 喏，自动传入啦
_ = f :> Nat
```

局部变量：

```
_ = let jojo = O in f :> Nat
_ = f :> Nat where jojo = O
```

如果作用域内有两个满足需求的实例，就会报错。

依赖记录

```
module DependentRecord where
```

这里*依赖*是形容词，不是动词。英文原文是 dependent record ， 表示后面成员的类型可以依赖前面成员的 `record` 。

Haskell 中的类型类使用 `class` 关键字定义，比如 `Eq` 类型类：

```
class Eq t where
  (==) :: t -> t -> Bool
```

立即使用 record 表达：

```
record Eq {a} (A : Type a) : Type a where
  field _===_ : A -> A -> Bool
  infixl 32 _===_
```

由于 Agda 的 record 就是带参数的 module ，我们可以把它的成员定义 open 出来：

```
module EqAsExplicitArgument where
  open Eq
```

可以简单地看出，被 open 出来的 _===_ 的类型：

```
  _ = _===_ :> (Eq Nat -> Nat -> Nat -> Bool)
```

要使用的话，先搞个实例，然后 Haskell 里当成 constraint 用的情况改成参数就是了：

```
  natEqInstance : Eq Nat
  natEqInstance = record { _===_ = natEq }
```

我们还可以使用优美的余模式（copattern）：

```
  natEqInstance' : Eq Nat
  _===_ natEqInstance' = natEq
```

根据 `Eq` 类型类，定义不等于（就不 point-free 了，怕伤害不精通各种 arrow 操作的萌新）：

```
natNeq' : {A : Type i} -> (Eq A) -> A -> A -> B
natNeq' eq a b = not $ _===_ eq a b
```

使用一下这个不等于：

```
_ : false == natNeq' natEqInstance 0 0
_ = idp

_ : true == natNeq' natEqInstance 1 2
_ = idp
```

嗯，非常妙。

我们现在已经实现了一个非常不优美（不优美的原因是，我们要手动传入类型实例） 但比 Haskell 版本更灵活的类型类，并给它创建类型实例了！

```
module EqAsInstanceArgument where
```

在这个基础上，我们如果把那个 `(Eq A)` 做成实例参数，就可以让编译器自己去找类型实例， 实现 Haskell 中 `=>` 一样的效果了！

我们需要做两个手脚，首先 `open` 出来的 `_===_` 中的 `(Eq A)` 需要变成实例参数 `{{ Eq A }}`：

```
open Eq {{ ... }}
```

没错，这个需求有点复杂，所以 Agda 专门做了个语法来简化这个流程。 当然，不代表我们不能手动实现这个语法糖实现的功能：

```
module StupidImplementation
       {i} {A : Type i} {{ eq : Eq A }} where
  open module EqInstances = Eq eq public
```

其次我们的类型实例需要被 `instance` 修饰：

```
instance
  natEqInstance : Eq Nat
  natEqInstance = record { _===_ = natEq }
```

嗯，由于我们已经把 `Eq` 以实例参数形式 `open` 出来，就用不了余模式了 QAQ。
然后再试试用这个类型类实现不等于。先显式传入，找找感觉（实例参数必须 introduce 一个变量，不知道为什么。如果不想写，命名为下划线就是）：

```
natNeq' : {A : Type i} -> {{_ : Eq A}} -> A ->
natNeq' {{eq}} a b = not $ _===_ {{eq}} a b
```

自动传入（终于能用上运算符语法了，嘤嘤嘤）：

```
natNeq : {A : Type i} -> {{_ : Eq A}} -> A -> A
natNeq a b = not $ a === b
```

运算符重载了解一下，自定义优先级结合性了解一下：

```
infixl 32 _=/=_
_=/=_ : {A : Type i} -> {{_ : Eq A}} -> A -> A
a =/= b = not $ a === b
```

相信此时浏览器等宽字体设置为 Fira Code 的读者已经被爽到了（噫）。

简单地调用，证明一下性质：

```
_ : false == O =/= O
_ = idp

_ : true == O =/= S O
_ = idp
```

完美！我们有类型类了！

嗯，数学作业要写不完了 QAQ

创建一个 issue 以申请评论

Create an issue to apply for commentary
