Agda 中的证明，从二到三

2017, Nov 8 by Tesla Ice Zhang

这篇文章我们来说说模式匹配的一种特殊情况。 到目前为止，很可能部分字符无法在一些字体下正常显示，这时候使用 LaTeX 的优越性就体现出来了。

本文主要讲如何证明一个命题是假命题，即证伪。

前置知识

- 上一篇文章

以及，由于 Agda 语言的特殊性，本文将继续使用 LaTeX 和代码块来共同展示代码。 代码块唯一的作用在于便于复制，主要的呈现途径为 LaTeX 。

请注意，本文将使用极骚的函数命名。 请非 Agda 或 Lisp 用户不要在家模仿。

假命题 Bottom Type

想必写过 Scala 或者 Kotlin 的同学都很熟悉 Bottom Type 的概念，在前面两个语言当中有一个特殊的类型 `Nothing` ，它是所有类型的子类型。

这些类型有一个共同的特征，就是他们**没有实例**，只有那些在求值时会导致程序中止或者死循环的表达式才会属于这个类型。

这就是为什么他们可以放在任何地方的原因，因为执行到他们这里就会异常退出或者永远无法继续，因此它的返回值根本就不存在，不需要处理。 根据它这样的特征，在支持 subtyping 的语言中，这个类型就成了所有类型的子类型。

在 Rust 中，这个类型叫 `!` 。

在 Haskell 中（其实这个并不是严格意义上的，只是功能类似，类型上并不是，要是把它真当成 Haskell 的 Bottom Type 会被喷的），这个类型叫 `Void` ，在 `Data.Void` 里。

这类类型的统称就是 Bottom Type 。 在证明中，它对应的命题是假命题。 在集合论中，它对应的集合是空集。

对应的 GADT

标准库的定义在 `Data.Empty` 中。 我们来看看它对应的 GADT：

$$\mathrm{data}\ \bot : \mathrm{Set}\ \mathrm{where}$$

```
data ⊥ : Set where
```

只有一行，我们声明了这个类型，然后不给它定义构造器。 也就是说，这个东西没有 instance ，符合刚才的定义。 这意味着我们需要在证明的时候，既要保证对输入 total ，也要保证对输出的所有情况都不存在。

这个东西有什么用呢？

加法与乘法

比如前文所讲的 $\wedge$ 类型，如果它的两个类型参数其中一个是 $\bot$ ，那么你就无法构造这个类型了，它也变成了 Bottom Type 。

这正好和数学上的一种运算非常相似——乘法。 任何数和 0 进行 $\times$ （乘法）运算都得到 0 ，而任何类型和 $\bot$ 进行 $\wedge$ 运算都得到 $\bot$ 。

而前文所讲的 $\vee$ 类型，如果它的两个类型参数其中一个是 $\bot$ ，你依旧可以从另一个类型使用对应的 $\vee-\mathrm{intro}_x$ 构造器生成这个类型。 而这时，你只能通过另一个类型对应的 $\vee-\mathrm{intro}_x$ 构造器生成这个类型，因此这个 $\vee$ 后的类型和原来的非 $\bot$ 类型等价。

这也正好和数学上的一种运算非常相似——加法。 任何数和 0 进行 $+$ （加法）运算都得到它自己，而任何类型和 $\bot$ 进行 $\vee$ 运算也都得到它自己。

是不是很奇妙呢？

当然，它的应用远不止这些。 我们将在接下来的内容中看到它是如何表示假命题的。

Absurd pattern

在 Haskell 的 `Data.Void` 中，有一个函数叫 `absurd` ，类型是 `Void -> a` 。 之所以它能写出这么骚的类型签名，是因为它不 total 。

同样，我们也可以在 Kotlin 中写出类似的东西，在有 subtyping 的情况下， `absurd` 函数直接返回参数即可：

```
fun absurd(a: Nothing): Any = a
```

完全等价的 Scala 版本：

```
def absurd(a: Nothing): Any = a
```

在 Agda 中，假命题可以推出任何命题，比如假命题可以推出假命题，因为假命题都成立了，那还有什么不能成立的（而这种就是一个吃饱了撑的证明）。 在程序的角度来看，一个函数以一个没有值的类型作为参数，那么对它的调用就是不可能发生的，因此随便返回啥都行。

因此，我们一般证明的是其他类型在经过各种组合下可以被推为假命题的情况。 也就是说，在一般情况下，它作为证明的结论。

所以我们一般在函数的返回值里看到它（而不是参数）。

在定义一个 Agda 的函数时，我们需要对参数进行模式匹配。 如果有多种情况，我们需要一一把它们写出来（参照前文的与 $\vee$ 有关的函数）。

但是如果只有零种情况，或者说，没有情况呢？

比如，我们要写一个上面说的那个吃饱了撑的的证明，应该怎么写呢？

零种情况

我们先来写下类型签名：

$$\text{absurd} : \bot \to \bot$$

```
absurd : ⊥ → ⊥
```

然后我们在模式匹配的时候，使用 `()` 来代表零种可能的匹配的情况：

$$\text{absurd}\ ()$$

```
absurd ()
```

好， it checks 。

这个 `()` 就是 absurd pattern 了。 在匹配的模式中，如果使用了 absurd pattern ，那么就不需要，也不能写对应的函数体了。它的名字也可以说是借鉴了 Haskell 中的这个函数名。

其他例子

同理，我们可以写一些类似的代码（关于 $\wedge$ 和 $\vee$ 的定义请参考前文），练习一下对 absurd pattern 的使用：

$$\begin{aligned}&\mathrm{proof}_0 : \forall\{A\} \to \bot \wedge A \to \bot\\&\mathrm{proof}_0\ (\wedge\text{-}\mathrm{intro}\ ()\ \_)\end{aligned}$$

$$\begin{aligned}&\mathrm{proof}_1 : \forall\{A\} \to \bot \vee A \to A\\&\mathrm{proof}_1\ (\vee\text{-}\mathrm{intro}_0\ ())\\&\mathrm{proof}_1\ (\vee\text{-}\mathrm{intro}_1\ x) = x\end{aligned}$$

```
proof₀ : ∀ {A} → ⊥ ∧ A → ⊥
proof₀ (∧-intro () _)

proof₁ : ∀ {A} → ⊥ ∨ A → A
proof₁ (∨-intro₀ ())
proof₁ (∨-intro₁ x) = x
```

而真正地表达那种正正经经的假命题，可以看这个例子（ $\equiv$ 的定义参照前文）：

$$\begin{aligned}&\mathrm{proof}_2 : \{a : \mathbb{N}\} \to a \equiv \mathrm{suc}\, a \to \bot\\&\mathrm{proof}_2\ ()\end{aligned}$$

```
proof₂ : {a : ℕ} → a ≡ suc a → ⊥
proof₂ ()
```

由于 a 不可能等于 a 的继数，因此直接使用 absurd pattern 即可证伪。

好，我们现在不仅知道知道如何表达假命题，而且知道如果要证明一个命题是假命题， 只需要写一个函数，以该命题为参数，返回一个 $\bot$ 类型就好了。

一个简单的证明

我们来定义一个 GADT ，表达两个自然数为两倍关系：

$$\begin{aligned}&\mathrm{data}\ \_\mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\_ : \mathbb{N} \to \mathbb{N} \to \mathrm{Set}\ \mathrm{where}\\&\quad \mathrm{intro} : \forall\{n\ m\} \to n + n \equiv m \to m\ \mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\ n\end{aligned}$$

```
data _is-double-of_ : ℕ → ℕ → Set where
  intro : ∀ {n m} → n + n ≡ m → m is-double-of n
```

然后我们可以简单地证明 $8\ \mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\ 4$ 成立：

$$\begin{aligned}&8{=}4{\times}2 : 8\ \mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\ 4\\&8{=}4{\times}2 = \mathrm{intro}\ \mathrm{refl}\end{aligned}$$

```
8=4×2 : 8 is-double-of 4
8=4×2 = intro refl
```

而我们可以很容易地证伪 $9\ \mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\ 4$：

$$\begin{aligned}&9{\neq}4{\times}2 : 9\ \mathrm{is}{-}\mathrm{double}{-}\mathrm{of}\ 4\\&9{\neq}4{\times}2\ (\mathrm{intro}\ ())\end{aligned}$$

```
9≠4×2 : 9 is-double-of 4 → ⊥

9≠4×2 (intro ())
```

是不是很简单呢。

结束

是的，我说完了。

Tweet this

Top

创建一个 issue 以申请评论

Create an issue to apply for commentary

协议/License