L108: Category theory and logic
Exercise sheet 7

Jonas Frey
jlf46@cl.cam.ac.uk

Revised: December 2, 2013, 18:44.

Limits in functor categories

1. Let $\mathbb{C}$ and $\mathbb{D}$ be categories, where $\mathbb{D}$ has finite products. Show that the functor category $\mathbb{D}^{\mathbb{C}}$ also has finite products, which are computed 'pointwise' – the object part of the binary product $F \times G$ of functors $F, G : \mathbb{C} \to \mathbb{D}$, for example, is given by the formula

$$(F \times G)(C) = F(C) \times G(C)$$

where the product on the right is formed in $\mathbb{D}$.

The pointwise construction of finite products in functor categories generalizes directly to *finite limits*, which were introduced in exercise sheet 4, and by dualization to *finite colimits*.

Exponentials in presheaf categories

Unlike limits and colimits, exponentials can not be computed pointwise, and even if $\mathbb{D}$ is cartesian closed, $\mathbb{D}^{\mathbb{C}}$ need not be so.

However, presheaf categories $\widehat{\mathbb{C}} = \mathbf{Set}^{\mathbb{C}^{\mathsf{op}}}$ for small categories $\mathbb{C}$ are always cartesian closed, which shows that **Set** has a special role among categories.

As sketched in the lecture, the formula for exponentials in $\widehat{\mathbb{C}}$ can be derived (and memorized) using the following sequence of bijections (for $F, G : \mathbb{C}^{\mathsf{op}} \to \mathbf{Set}$):

$$(G^F)(I) \cong \widehat{\mathbb{C}}(Y(I), G^F) \cong \widehat{\mathbb{C}}(Y(I) \times F, G),$$

where the first bijection is given by the Yoneda lemma, and the second one is given by uncurrying (compare exercise 10 on sheet 6). This suggests defining $G^F := \widehat{\mathbb{C}}(Y(-) \times F, G)$, or more formally

$$\begin{array}{rcccl} G^F & : & \mathbb{C}^{\mathsf{op}} & \to & \mathbf{Set} \\ & & I & \mapsto & \widehat{\mathbb{C}}(Y(I) \times F, G) \\ & & f & \mapsto & \widehat{\mathbb{C}}(Y(f) \times \mathrm{id}_F, \mathrm{id}_G) \end{array}$$

2. Show that the above definition does indeed give an exponential of F and G in $\widehat{\mathbb{C}}$. To this end

   (a) define the evaluation transformation

$$\varepsilon_G : G^F \times F \to G,$$

   (b) define the 'currying' operation $\Lambda : \widehat{\mathbb{C}}(H \times F, G) \to \widehat{\mathbb{C}}(H, G^F)$,

   (c) show that for $\eta : H \times F \to G$, $\Lambda(\eta)$ is the unique transformation making the appropriate diagram commute.

The state monad

Let us recall the definition of the state monad and of its corresponding adjunction from exercise sheet 6.

Let $\mathbb{C}$ be a cartesian closed category, and $S \in \mathrm{obj}(\mathbb{C})$. The functor

$$\begin{array}{rccc} P_S & : & \mathbb{C} & \to & \mathbb{C} \\ & & A & \mapsto & A \times S \\ & & f & \mapsto & f \times S = f \times \mathrm{id}_S \end{array}$$

has a right adjoint

$$\begin{array}{rccc} E_S & : & \mathbb{C} & \to & \mathbb{C} \\ & & A & \mapsto & A^S \\ & & f & \mapsto & f^S = \Lambda(f \circ \varepsilon). \end{array}$$

The unit $\eta : \mathrm{id}_{\mathbb{C}} \to E_S \circ P_S$ is given by

$$\eta_A = \Lambda(\mathrm{id}_{A \times S}) : A \to (A \times S)^S,$$

the components of the counit $\varepsilon : P_S \circ E_S \to \mathrm{id}_{\mathbb{C}}$ are the evaluation maps

$$\varepsilon_A : A^S \times S \to A.$$

The state monad $T : \mathbb{C} \to \mathbb{C}$ is the monad induced by this adjunction; thus it is given by $T = E_S \circ P_S$, with the same unit η as the adjunction and the multiplication $\mu : T \circ T \to T$ given by

$$\mu_A = (\varepsilon_{A \times S})^S : ((A \times X)^S \times S)^S \to (A \times S)^S.$$

3. In this exercise, we want to understand the Kleisli category $\mathbb{C}_T$ of the state monad. The idea is that a morphism $f : A \to_T B$ in $\mathbb{C}_T$ (which is by definition a morphism $A \to TB = (B \times S)^S$ in $\mathbb{C}$) models a program with input A and output B, which can access and modify some form of global state of the computer (e.g. RAM or disk, or more abstractly a global variable). The object S represents the set of states.

   To see the intuition behind the definition of T, note that morphisms $f : A \to (B \times S)^S$ are in bijection with 'uncurried' morphisms

   $$\check{f} = \varepsilon_{A \times S} \circ (f \times \mathrm{id}_S) : A \times S \to B \times S,$$

   thus morphisms in the Kleisli category can be viewed as programs taking a value of type A and a state as input, and returning a value of type B and a new state.

   Given morphisms $f : A \to (B \times S)^S$ and $g : B \to (C \times S)^S$, the uncurried morphisms $\check{f} : A \times S \to B \times S$ and $\check{g} : B \times S \to C \times S$ can be composed directly. Show that

   $$(g \circ_T f)\check{} = \check{g} \circ \check{f}.$$

   This formula can be viewed as explanation of the meaning of the composition in the Kleisli category.

Dependent types

In the lecture I stated that slice categories can be used to model 'dependent types'. The following exercises aim to convey some of the basic ideas behind this.

Syntactically, a dependent type is a type depending on a variable of another type. The standard example is the type

$$n{:}\mathbf{nat} \vdash \mathbf{list}(n)$$

of lists (of say integers) of length n, refining the type $\mathbf{nat}^*$ of all lists of integers.

Given a dependent type $x{:}A \vdash B(x)$, the *dependent sum type*

$$\Sigma x{:}A\,.\,B(x)$$

should be viewed as the union of all the $B(x)$, where x ranges over A. For example, the sum $\Sigma n{:}\mathbf{nat}\,.\,\mathbf{list}(n)$ can be identified with the type $\mathbf{nat}^*$ of all lists.

More generally, we can have types depending on several variables, and sum only over one of them, e.g. one can imagine a type $x{:}A, y{:}B(x) \vdash C(x,y)$ allowing to form the sum type $x{:}A \vdash \Sigma y{:}B(x)\,.\,C(x,y)$.

Dependent product types $x{:}A \vdash \Pi y{:}B(x)\,.\,C(x,y)$ can be formed analogously, but in this exercise we focus on the sum types.

The following exercise aims to clarify the intuition behind the association of dependent types and slice categories.

We require the following definition:

Definition 1. *A* equivalence of categories $\mathbb{C}$ *and* $\mathbb{D}$ *is a pair* $H : \mathbb{C} \to \mathbb{D}$ *and* $K : \mathbb{D} \to \mathbb{C}$ *of functors, such that* $K \circ H \cong \mathrm{id}_{\mathbb{C}}$ *and* $H \circ K \cong \mathrm{id}_D$*. If an equivalence of categories exists between* $\mathbb{C}$ *and* $\mathbb{D}$*, we call* $\mathbb{C}$ *and* $\mathbb{D}$ equivalent*.*

4. Let I be a set and $\mathbb{C}$ a category. The category $\mathbb{C}^I$ is defined as follows

   - *objects* are families $(C_i \mid i \in I)$ of objects of $\mathbb{C}$
   - a *morphism* from $(C_i \mid i \in I)$ to $(D_i \mid i \in I)$ is a family $(f_i : C_i \to D_i \mid i \in I)$ of morphisms in $\mathbb{C}$
   - composition is defined by $(g_i \mid i \in I) \circ (f_i \mid i \in I) = (g_i \circ f_i \mid i \in I)$

   Show that $\mathbf{Set}^I$ is equivalent to $\mathbf{Set}/I$ (i.e. construct an equivalence of categories between them).

5. Let $\mathbb{C}$ be a category in which all pullbacks exist. Given $f : J \to I$ in $\mathbb{C}$, we can define a functor $f^* : \mathbb{C}/I \to \mathbb{C}/J$ whose object part is given by 'pullback along f'. More precisely, for $a : A \to I$, $f^*(a)$ is the left side of the pullback square

$$\begin{array}{ccc} J \times_I A & \longrightarrow & A \\ {\scriptstyle f^*(a)}\downarrow & \text{pb.} & \downarrow{\scriptstyle a} \\ J & \xrightarrow[f]{} & I \end{array}$$

   Define the morphism part of f^*, and verify the functor axioms.

6. Postcomposition by f gives a functor

$$f_! : \mathbb{C}/J \to \mathbb{C}/I, \qquad b \mapsto f \circ b$$

Show that $f_!$ is left adjoint to f^*.

7. Using the equivalence of exercise 4, explain why in the case $\mathbb{C} = \mathbf{Set}$, $f_!$ can be viewed as a disjoint union operation. For this, you may assume that $I = 1$.