Formalizing the Construction of Exponentials in an Elementary Topos

Jason Reed

<jcreed+@andrew.cmu.edu>

May 2, 2001

1 Introduction

Commonly (for instance, in [Jac99]) an elementary topos is defined as a finitely complete cartesian closed category $\mathcal{E}$ with a subobject classifier. This is certainly a very compact definition compared to, say, the definition of a Grothendieck topos, and using the notion of cartesian closed category has a certain appeal because of the well-known fact that CCCs correspond to the simply-typed lambda-calculus. However, we need not require all exponentials to exist to arrive at the same concept; An equivalent definition of a topos used by MacLane and Moerdijk [MM92] is as follows:

Definition A topos is a category $\mathcal{E}$ with

i) A pullback for every diagram $X \to B \leftarrow Y$;

ii) A terminal object, 1;

iii) An object Ω and an arrow $\mathsf{true} : 1 \to \Omega$ such that for every monic arrow $m : S \to B$ there exists a unique arrow $\mathsf{char}\, m : B \to \Omega$ such that

$$\begin{tikzcd} S \arrow[r, "!"] \arrow[d, "m"'] \arrow[dr, phantom, "(\!|\Omega|\!)"] & 1 \arrow[d, "\mathsf{true}"] \\ B \arrow[r, "\mathsf{char}\, m"'] & \Omega \end{tikzcd}$$

is a pullback diagram[1].

[1] In general we use $(\!| X |\!)$ to indicate the diagram defining the concept X.

iv) To each object B an object PB (the "powerobject of B") and an arrow $\in_B : B \times PB \to \Omega$ such that for every arrow $f : B \times A \to \Omega$ there is a unique arrow $\bar{f} : A \to PB$ such that

$$\begin{tikzcd} B \times A \arrow[r, "f"] \arrow[d, "B \times \bar{f}"'] \arrow[dr, phantom, "(\!| P |\!)"] & \Omega \arrow[d, equal] \\ B \times PB \arrow[r, "\in_B"'] & \Omega \end{tikzcd}$$

commutes. Here $B \times \bar{f}$ is a synonym of $\mathrm{id}_B \times \bar{f}$, which is in turn an abbreviation (as described in Section 2.4) of $\langle \mathrm{id}_B \circ \mathsf{fst}, \bar{f} \circ \mathsf{snd} \rangle$.

This definition sacrifices the brevity of the former one, but the immediate demands it makes of the category $\mathcal{E}$ are more conservative. Only certain finite limits are required (from which we can construct the rest) and only certain function spaces are required, those whose codomain is the subobject classifier, Ω—the definition of PB reflects all the properties that Ω^B would have, had we required exponentials to exist. The construction of all finite limits from pullbacks and a terminal object is not terribly surprising or difficult, but the proof of the fact that all functions can be recovered from just those into Ω is somewhat less trivial, using the intuition that topoi are 'generalized universes of set theory', and imitating the usual set-theoretic definition of functions as single-valued relations.

The proof given in [MM92], however, still leaves many of the details to be checked by the reader. Here we fill in the gaps by providing a full formal proof of the existence of exponentials in an elementary topos as defined there, formalized in Twelf, using heavily its support for dependent types.

2 Formalization of a Topos

Before we can even consider formally constructing exponentials, we must first formally establish our assumptions; the axioms characterizing an elementary topos must be somehow represented in Twelf. First, we have to have that $\mathcal{E}$ is a category, then make sure that it has finite limits, and finally encode the existence of a subobject classifier and powerobjects.

2.1 Formalization of a Category

For the most part, we follow Andrzej Filinksi's [Fil92] encoding of the definition of a category in Twelf. We have type families

```
obj : type.
mor : obj -> obj -> type.
```

to represent the objects and morphisms of a category, and a family

```
== : mor A B -> mor A B -> type.
%infix none 5 ==.
```

to capture equality between morphisms at the type $A \to B$. The other ingredients of a category are the identities and composition:

```
e_id : {A : obj} mor A A.
id = e_id A.
@ : mor B C -> mor A B -> mor A C.
%infix none 10 @.
```

Some care is taken here (and this is a departure from [Fil92]) to retain a constructor `e_id` for the identity morphisms which has an *explicit* dependence on the object in question, and from it we define a term `id` which makes the dependence implicit. It turns out that both of these forms have their uses; most of the time we will use `id` for brevity, but in the process of defining the exponential object `=> : obj -> obj -> obj`, the explicit form is required. In fact, the same tactic will be used with several other defined terms, (most of them steps along the way to defining `=>`) with the convention that an '`e_`' prefix indicates the more explicit version. This permits us to provide different amounts of typing information depending on what the context requires.

Now that we have all the data of a category, we need them to satisfy certain conditions. Equality, of course, has to be an equivalence relation, so we specify

```
refl : F == F.
then : F == F' -> F' == F'' -> F == F''.
%infix right 4 then.
sym : F == F' -> F' == F.
```

and composition must be associative, and the identity morphisms must be neutral with respect to composition:

```
id_l : id @ F == F.
id_r : F @ id == F.
ass : H @ (G @ F) == (H @ G) @ F.
```

Finally, we need to have that composition is a congruence with respect to `==`: [2]

```
=@= : F == F' -> G == G' -> F @ G == F' @ G'.
```

[2] Another way of viewing these definitions is in terms of 2-categories: instead of thinking of `F == G` as morphism *equality*, we can interpret it as the type of 2-cells between the morphisms f and g. In this view, the constructor `refl` gives the identity 2-cells, `then` is vertical composition of 2-cells, and `=@=` is horizontal composition. The meaning of `sym` is less obvious, but it seems reasonable to believe that it corresponds to what [Kel82] calls a *duality involution*. This perspective seems to have some value, because the commutativity conditions for a 2-category with a duality involution give rise to a natural-sounding set of equations between proofs using `sym`, `refl`, `then`, and `=@=`, (for instance, the interchange law asserts that `(=@=` P_1 P_3`) then (=@=` P_2 P_4`)` is equal to `=@= (`P_1 `then` P_2`) (`P_3 `then` P_4`)`) but it is not pursued further here.

2.2 Terminal Object

A terminal object of a category is, by definition, an object 1 such that for every object A there is a unique arrow $!_A : A \to 1$. We again follow [Fil92] in the easy formalization of this concept, with the exception of making an explicit version of `drop`:

```
1 : obj.
e_drop : {A : obj} mor A 1.
drop = e_drop A.
term_u : drop == F.
```

Uniqueness is expressed by the fact that `drop`, (or, equivalently, `e_drop A`) the canonical arrow $A \to 1$, is equal to any other arrow $A \to 1$. Of course we do not impose any unwanted morphism equalities (say, that `drop` is equal to every other morphism) because the (dependent) type of `==` captures the fact that morphisms can only be equal if they have the same domain and codomain.

2.3 Pullbacks

The standard way of defining a pullback is as the limiting cone of a diagram of the following shape:

$$\begin{tikzcd} & A \arrow[d, "f"] \\ B \arrow[r, "g"'] & C \end{tikzcd}$$

More concretely, a category *has all pullbacks* if for every such diagram there exists an object P and arrows $\pi_1 : P \to A, \pi_2 : P \to B$ such that the diagram

$$\begin{tikzcd} P \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] & A \arrow[d, "f"] \\ B \arrow[r, "g"'] & C \end{tikzcd}$$

commutes, (in which case we call $A \xleftarrow{\pi_1} P \xrightarrow{\pi_2} B$ a cone over the diagram $A \xrightarrow{f} C \xleftarrow{g} B$) which is 'universal' among all such cones. Universality means that, for any object Z and any arrows $m : Z \to A$ and $n : Z \to B$ such that $f \circ m = g \circ n$, there exists a unique arrow $u : Z \to P$ such that $\pi_1 \circ u = m$ and $\pi_2 \circ u = n$.

We first establish a simple result that makes formalization more straightforward:

Proposition 2.1 *A category has all pullbacks iff for all arrows $f : A \to C$, $g : B \to C$*

1. *There exists an object $f \# g$,*
2. *There exist arrows $\mathsf{fst}^{\#}(f,g) : (f \# g) \to A$ and $\mathsf{snd}^{\#}(f,g) : (f \# g) \to B$.*
3. *For any two arrows $m : Z \to A$, $n : Z \to B$ such that $f \circ m = g \circ n$ there exists an arrow $\langle\!\langle m, n\rangle\!\rangle : Z \to f \# n$.*
4. *$f \circ \mathsf{fst}^{\#}(f,g) = g \circ \mathsf{snd}^{\#}(f,g)$.*
5. *$\mathsf{fst}^{\#}(f,g) \circ \langle\!\langle m, n\rangle\!\rangle = m$ and $\mathsf{snd}^{\#}(f,g) \circ \langle\!\langle m, n\rangle\!\rangle = n$.*
6. *For any $h : Z \to (f \# g)$, we have $\langle\!\langle \mathsf{fst}^{\#}(f,g) \circ h, \mathsf{snd}^{\#}(f,g) \circ h\rangle\!\rangle = h$.*

Proof If we have all pullbacks, then for every appropriate f, g we can choose a pullback

$$\begin{tikzcd} P \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ B \arrow[r, "g"'] & C \end{tikzcd}$$

and define $f \# g := P$, $\mathsf{fst}^{\#}(f,g) := \pi_1$, and $\mathsf{snd}^{\#}(f,g) := \pi_2$. Conditions 1, 2, are immediately satisfied, and 4 follows from the fact that the pullback has to be a cone over $A \xrightarrow{f} C \xleftarrow{g} B$. The universal property of the pullback provides the existence of the arrow in 3, which satisfies 5. To see that 6 holds, let $m = \mathsf{fst}^{\#}(f,g) \circ h$ and $n := \mathsf{snd}^{\#}(f,g) \circ h$ and observe that

$$\pi_1 \circ \langle\!\langle \mathsf{fst}^{\#}(f,g) \circ h, \mathsf{snd}^{\#}(f,g) \circ h\rangle\!\rangle = m$$

and

$$\pi_2 \circ \langle\!\langle \mathsf{snd}^{\#}(f,g) \circ h, \mathsf{snd}^{\#}(f,g) \circ h\rangle\!\rangle = n$$

so by uniqueness, we conclude $\langle\!\langle \mathsf{snd}^{\#}(f,g) \circ h, \mathsf{snd}^{\#}(f,g) \circ h\rangle\!\rangle = h$.

In the other direction, suppose that 1-6 hold. Conditions 1, 2, and 4 imply the existence of a cone over $A \xrightarrow{f} C \xleftarrow{g} B$. Namely, we can set $P := f \# g$, $\pi_1 = \mathsf{fst}^{\#}(f,g)$, and $\pi_2 = \mathsf{snd}^{\#}(f,g)$. For any $m : Z \to A$ and $n : Z \to B$ such that $f \circ m = g \circ n$, 3 and 5 provide the existence of a mediating map $u : Z \to P$ such that $\pi_1 \circ u = m$ and $\pi_2 \circ u = n$. Now suppose that we had a v such that also $\pi_1 \circ v = m$ and $\pi_2 \circ v = n$. Then, using property 6, we have $v = \langle\!\langle \pi_1 \circ v, \pi_2 \circ v\rangle\!\rangle = \langle\!\langle m, n\rangle\!\rangle = \langle\!\langle \pi_1 \circ u, \pi_2 \circ u\rangle\!\rangle$ so uniqueness of u holds. ∎

The formalization is now clear; We need a constructor

```
# : mor A C -> mor B C -> obj.
%infix none 10 #.
```

providing the object of the pullback cone, (satisfying condition 1 of the proposition) and constructors

```
#fst : {F : mor A C} {G : mor B C} mor (F # G) A.
#snd : {F : mor A C} {G : mor B C} mor (F # G) B.
```

for the first and second projections, satisfying 2. Condition 3 is expressed as the pairing function,

```
#pair : {M : mor Z A} {N : mor Z B}
       F @ M == G @ N -> mor Z (F # G).
```

and 4-6 are the following families of equations:

```
#com : F @ #fst F G == G @ #snd F G.
pull_l : (#fst F G) @ (#pair M N P) == M.
pull_r : (#snd F G) @ (#pair M N P) == N.
pull_u : #pair (#fst F G @ H) (#snd F G @ H) P == H.
```

Finally, we want `#pair` to be a congruence with respect to `==`:

```
=#pair= : M == M' -> N == N' -> #pair M N P == #pair M' N' P'.
```

Note the absence of an argument to `=#pair=` of the form `P == P'`—for one thing, `P`, `P'` are not morphisms, so `P == P'` is not even well-typed. Fortunately, we have no need to introduce a notion of proof equality, since the intent of `=#pair=` is to realize the equality $\langle\!\langle m, n\rangle\!\rangle = \langle\!\langle m', n'\rangle\!\rangle$ when $m = m'$ and $n = n'$ *regardless of how we proved these pairings exist.* Ideally, we would like *irrelevance* for the proofs that guarantee pairings. In the notation of [Pfe01], the type of `#pair` ought to be (omitting implicit Π-quantifiers)

$$\Pi m{:}\,\mathsf{mor}\ Z\ A.\Pi n{:}\,\mathsf{mor}\ Z\ B.\Pi p \div (F \mathbin{@} m == G \mathbin{@} n).\mathsf{mor} Z(F \mathbin{\#} G)$$

so that any two proofs of $F \mathbin{@} m == G \mathbin{@} n$ are always equal from the point of view of `#pair`. That is, `#pair M N P` and `#pair M N P'` are *definitionally equal* as long as `P` and `P'` have the right type.

2.4 Products

At this point we could again look to [Fil92] and axiomatize binary products by

```
* : obj -> obj -> obj. %infix none 10 *.

fst : mor (A * B) A.
snd : mor (A * B) B.
pair : mor A B -> mor A C -> mor A (B * C).
```

```
=pair= : F == F' -> G == G' -> pair F G == pair F' G'.

prod_l : fst @ (pair F G) == F.
prod_r : snd @ (pair F G) == G.
prod_u : pair (fst @ H) (snd @ H) == H.
```

However, we instead make use of the fact that any category with a terminal object and pullbacks has all finite limits, and derive products from the constructors we have already defined.[3] The product of A and B is recovered by taking the pullback of the (unique) arrows from A and B into 1, respectively:

$$
\begin{tikzcd}
A \times B \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] \arrow[dr, phantom, very near start, "\lrcorner"] & A \arrow[d, "!"] \\
B \arrow[r, "!"'] & 1
\end{tikzcd}
$$

This works because the arrows into 1 make the commutativity requirement for pullback pairing trivially hold, no matter what arrows our arrows with (respectively) codomain A and B are. Therefore we recover the universal property of the product.

The formalization of this argument is straightforward. We can use the explicit `e_drop` constructor to describe the object part of the product.

```
* : obj -> obj -> obj = [A] [B] e_drop A # e_drop B.
%infix none 10 *.
```

The remainder of the proof is implemented by building terms of the same type as the constructors of [Fil92].

```
fst : mor (A * B) A = #fst drop drop.
snd : mor (A * B) B = #snd drop drop.

pair : mor A B -> mor A C -> mor A (B * C)
    = [M : mor A B] [N : mor A C]
      #pair M N (sym term_u then term_u).

=pair= : F == F' -> G == G' -> pair F G == pair F' G'
        = [P] [Q] =#pair= P Q.

prod_l : fst @ (pair F G) == F = pull_l.
prod_r : snd @ (pair F G) == G = pull_r.
prod_u : pair (fst @ H) (snd @ H) == H = pull_u.
```

[3] Although the construction of other finite limits (in particular equalizers) is not difficult, we do not need them in what follows, so we only discuss binary products.

A useful notion definable from pairing arrows is 'multiplication' of arrows, $(f, g) \mapsto f \times g := \langle f \circ \mathsf{fst}, g \circ \mathsf{snd} \rangle$, which is formalized as

```
** : mor A B -> mor C D -> mor (A * C) (B * D)
     = [F] [G] pair (F @ fst) (G @ snd).
%infix none 10 **.
```

We prove here several easy lemmas that will be useful later in reasoning about `@`, `**`, and `pair`:

```
distp : (pair M N) @ F == pair (M @ F) (N @ F)
       = (sym prod_u) then =pair=
         (ass then (=@= prod_l refl))
         (ass then (=@= prod_r refl)).

**@ : (F ** G) @ (H ** J) == (F @ H) ** (G @ J)
      = distp then
        (sym (=pair= ass ass)) then
        (=pair= (=@= refl prod_l) (=@= refl prod_r)) then
        (=pair= ass ass).

=**= : F == F' -> G == G' -> F ** G == F' ** G'
       = [P] [Q] =pair= (=@= P refl) (=@= Q refl).

id**@ : id ** (F @ G) == (id ** F) @ (id ** G)
      = =**= (sym id_l) refl then
         sym **@.
```

2.5 Subobject Classifier

Since the definition of the subobject classifier is in terms of monic arrows[4], we first formalize the notion of monic. We declare the type of monomorphisms $A \rightarrowtail B$, and an inclusion into the type of morphisms.

```
subobj : obj -> obj -> type.
smor : subobj A B -> mor A B.
```

Any object of this type has the property characterizing monomorphisms:

```
subobj_def : (smor H) @ F == (smor H) @ G -> F == G.
```

and any element of `mor A B` together with a proof that it has this property can be lifted through the inclusion `smor` into `subobj A B`:

```
mors : {H : mor A B}
      ({Z : obj} {f : mor Z A} {g : mor Z A}
```

[4] Which, in this case, amounts to the same thing as subobjects, hence the names of the Twelf terms defined below.

```
        H @ f == H @ g -> f == g)
      -> subobj A B.
mor==s : smor (mors H P) == H.
```

Now the basic ingredients of the subobject classifier can be translated:

```
o : obj.
true : mor 1 o.
char : subobj A B -> mor B o.
```

However, encoding the fact that the diagram $(\!|\,\Omega\,|\!)$ from the definition of topos is a pullback requires more work. The problem is that we have no formal notion yet of what *a* pullback is, just how to find the canonical pullback for any pair of arrows into the same object. The solution is to provide enough information to guarantee an *isomorphism of cones* between

$$\begin{tikzcd} S \arrow[r, "!"] \arrow[d, "m"'] & 1 \\ A & \end{tikzcd}$$

and

$$\begin{tikzcd} \mathsf{true} \# \mathsf{char}\, m \arrow[r, "\mathsf{fst}^{\#}"] \arrow[d, "\mathsf{snd}^{\#}"'] & 1 \\ A & \end{tikzcd}$$

by which we mean a pair of maps

$$\begin{tikzcd} \mathsf{true} \# \mathsf{char}\, m \arrow[r, shift left, "\mu"] & S \arrow[l, shift left, "\upsilon"] \end{tikzcd}$$

such that

1. $\mathsf{fst}^{\#}\upsilon = \,!$ and $\mathsf{snd}^{\#}\upsilon = m$;
2. $!\mu = \mathsf{fst}^{\#}$ and $m\mu = \mathsf{snd}^{\#}$;
3. $\mu\upsilon = \mathrm{id}$ and $\upsilon\mu = \mathrm{id}$.

It is a basic fact about limits that the limiting cone is unique up to such an isomorphism, hence positing the existence of μ, υ with these properties exactly captures being a pullback. Moreover, in this case it suffices to require μ, υ such that merely

1. $m\mu = \mathsf{snd}^\#$;

2. $\mu\upsilon = \mathrm{id}$.

hold. We trivially have $\mathsf{fst}^\#\upsilon = !$ and $!\mu = \mathsf{fst}^\#$ because of uniqueness of arrows into 1, and $\mathsf{snd}^\#\upsilon = m$ follows from multiplying both sides of $m\mu = \mathsf{snd}^\#$ by υ on the right, and applying $\mu\upsilon = \mathrm{id}$. To see that $\upsilon\mu = \mathrm{id}$, compute $\mathsf{fst}^\#\upsilon\mu = !\mu = ! = \mathsf{fst}^\#$ and $\mathsf{snd}^\#\upsilon\mu = m\mu = \mathsf{snd}^\#$, and apply uniqueness of pullback pairing to conclude that $\upsilon\mu = \langle\langle \mathsf{fst}^\#\upsilon\mu, \mathsf{snd}^\#\upsilon\mu\rangle\rangle = \langle\langle \mathsf{fst}^\#, \mathsf{snd}^\#\rangle\rangle = \mathrm{id}$.

We formalize this as follows:

```
e_map : {M : subobj A B} mor (true # char M) A.
map = e_map M.

e_unmap : {M : subobj A B} mor A (true # char M).
unmap = e_unmap M.

map_c : smor M @ map == #snd true (char M).
map_u : map @ unmap == id.
```

Note that nothing in the above specifies that $\mathsf{char}\, m$ is the *unique* map making $(\!|\,\Omega\,|\!)$ a pullback. Although it is possible to express this condition, it is not necessary for the construction, so we omit it.

Remark The technique of keeping track of cone isomorphisms works satisfactorily for the pullback involved in the subobject classifier, (especially because everything on the side of the square involving the terminal object falls out as trivial) but it becomes cumbersome very quickly for even relatively easy proofs about pullbacks in general. For instance, the so-called "two-pullback lemma" (which is used heavily in [MM92] to verify Beck-Chevalley conditions for another important elementary topos construction, that of arbitrary finite colimits) would involve three canonical pullbacks, three noncanonical pullback squares, six arrows going both directions between them, and 18 commutativity conditions. It seems likely that a module system for Twelf [HP98] would make these proofs more manageable, by, if nothing else, providing a more structured namespace.

For the sake of convenience, we prove a few lemmas which more closely correspond to how the pullback properties of the subobject classifier are used in practice:

```
socp1 : char M @ smor M == true @ drop
       = sym id_r then
         =@= refl (sym map_u) then
         sym ass then
         =@= refl (ass then =@= map_c refl) then
         ass then
         =@= (sym #com) refl then
         sym ass then
```

```
         =@= refl (sym term_u).

cover : {F : subobj A B} {M : mor Z B} char F @ M == true @ drop
       -> mor Z A
         = [F : subobj A B] [M : mor Z B]
           [P : char F @ M == true @ drop]
            map @ #pair drop M (sym P).

=cover= : M == M' -> cover F M P == cover F M' P'
                  =
                    [Q : M == M']
                    =@= refl (=#pair= refl Q).

cover_r : smor F @ cover F M P == M
         = ass then =@= map_c refl then pull_r.

cover_u : cover F (smor F @ Y) P == Y
         = subobj_def cover_r.
```

2.6 Powerobject Function

It is again clear how to formalize the basic constructors for the powerobject function:

```
p : obj -> obj.
e_in : {A : obj} mor (A * p A) o.
in = e_in A.
transp: mor (B * A) o -> mor A (p B).
```

To capture the commutativity and uniqueness conditions, we first abbreviate

```
untransp = [F] in @ (id ** F).
```

so that the commutativity of the diagram $(\!| P |\!)$ is encoded as

```
pofp1 : untransp(transp F) == F.
```

To have uniqueness in a more convenient form we prove a result similar in spirit to Proposition 2.1.

Proposition 2.2 *Suppose that $\bar{\square}$ is a function from morphisms $B \times A \to \Omega$ to morphisms $A \to PB$ such that $\bar{f}$ has the property that $\in_B \circ (B \times \bar{f}) = f$. Then $\bar{f}$ is unique with this property iff for all $g : A \to PB$ we have $\overline{\in_B \circ (B \times g)} = g$*

Proof Suppose uniqueness holds. Then there is only one solution z to the equation $\in_B \circ (B \times z) = \in_B \circ (B \times g)$, namely $z := \overline{\in_B \circ (B \times g)}$. But g also is a solution, so $\overline{\in_B \circ (B \times g)} = g$. Conversely, suppose $\overline{\in_B \circ (B \times g)} = g$ for all g and $\in_B \circ (B \times z) = f$. Then $z = \overline{\in_B \circ (B \times z)} = \overline{f}$. ■

This permits formally expressing uniqueness as the family

```
pofp2 : transp(untransp F) == F.
```

of equations, and again we want to have congruence with respect to ==:

```
=transp= : F == F' -> transp F == transp F'.
```

3 The Exponential

3.1 Outline

Since [Fil92] is concerned with CCCs, its presentation of exponentials is axiomatic; there are constructors

```
=> : obj -> obj -> obj. %infix right 7 =>.

app : mor ((B => C) * B) C.
cur : mor (A * B) C -> mor A (B => C).

=cur= : F == F' -> cur F == (cur F').

exp_e : app @ (pair ((cur F) @ fst) snd) == F.
exp_u : cur (app @ (pair (G @ fst) snd)) == G.
```

As with products, we intend to instead build terms with these types out of previously defined constructors. Since we follow [MM92], however, it so happens that the products show up the other way around: our `app` has type `mor (B * (B => C)) C` and our `cur` will have type `mor (B * A) C -> mor A (B => C)`. This is not a significant problem, however, since products are commutative up to isomorphism. In fact, in Section 3.6 we show how the canonical isomorphism $\mathsf{tw}_{AB} : A \times B \cong B \times A$ can be used to recover exactly the types of [Fil92].

The construction of the exponential object is analogous to how functions are classically defined in set theory. A function $B \to C$ is a binary relation on B, C (that is, a subset of $B \times C$) which is single-valued; a relation R is single-valued iff $v(b, R)$ is always a singleton, where v is the 'relation evaluation map' $v : B \times \mathcal{P}(B \times C) \to \mathcal{P}C$ which takes an element $b \in B$ and a relation on B, C to the set of $c \in C$ such that bRc. Another way of stating this is by defining a function $u : \mathcal{P}(B \times C) \to \mathcal{P}B$ such that $u(R)$ is the set of elements b of B such that $v(b, R)$ is a singleton, and noting that R is single-valued iff $u(R) = B$.

We proceed by going back up this tower of definitions, beginning with the definition of singleton set and ending with the object of functions, replacing at each stage set-theoretic notions of elements and powersets with the corresponding topos-theoretic ideas. The exponential *object* is not the end of our work, however. To see that we have arrived at the 'right' definition of exponential, and not merely some arbitrary subobject of $P(B \times C)$, we need an evaluation arrow app and a function cur on arrows which satisfy the appropriate equations, namely those arising from an adjunction $\Box \times B \dashv \Box^B$. Specifically, we need

$$\mathsf{app} \circ (B \times \mathsf{cur}(f)) = f \qquad (\beta)$$

for any $f : B \times A \to C$ and

$$\mathsf{cur}(\mathsf{app} \circ (B \times g)) = g \qquad (\eta)$$

for any $g : A \to C^B$.

3.2 Constructing the Exponential Object

To find the topos-theoretic analogue of the notion of singleton set, we begin by defining the 'diagonal' map $\Delta_A := (1_A, 1_A)$. Formally, we have

```
e_diag = [A] pair (e_id A) (e_id A).
diag = e_diag A.
```

It is easy to show that Δ_A is always monic; in fact, whenever m is monic, we have that $\langle m, n\rangle$ is monic, and the identity is easily monic. We code these proofs as

```
pairmon : pair (smor M) N @ F == pair (smor M) N @ G
         -> F == G
           = [P : (pair (smor M) N) @ F == (pair (smor M) N) @ G]
             subobj_def (sym prod_l
                 then (=@= refl ((sym distp) then P then distp))
                 then prod_l).
```

and

```
sid = mors id
       [Z] [F] [G] [P: id @ F == id @ G]
       sym id_l then P then id_l.
```

so we can lift diag to the type of monics by

```
presdiag : pair (smor sid) id @ F == diag @ F
         = =@= (=pair= mor==s refl) refl.
e_sdiag = [A] mors (e_diag A)
       [Z] [F] [G] [P : diag @ F == diag @ G]
       pairmon (presdiag then P then sym presdiag).
sdiag = e_sdiag A.
```

Since Δ_A is monic, it has a characteristic arrow δ_A. Diagrammatically, we have

$$\begin{tikzcd}
A \arrow[r, "!"] \arrow[d, "\Delta_A"'] \arrow[dr, phantom, "(\!|\,\delta\,|\!)"] & 1 \arrow[d, "\mathsf{true}"] \\
A \times A \arrow[r, "\delta_A := \mathsf{char}\,\Delta_A"'] & \Omega
\end{tikzcd}$$

which is formalized as

```
e_kron = [A] char (e_sdiag A).
kron = e_kron A.
```

The name chosen is `kron` because δ_A is suggestive of the *Kronecker delta*

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{otherwise.} \end{cases}$$

Though in a general topos Ω may not be anything like $2 = \{0, 1\}$, (as it is in **Sets**) and objects are not necessarily sets, the subobject Δ_A of $A \times A$ is intuitively "the elements $(a_1, a_2) \in A \times A$ such that $a_1 = a_2$" and the function δ_A "takes an element $(a_1, a_2) \in A \times A$ to $\mathsf{true} \in \Omega$ iff $a_1 = a_2$".

Since δ_A is an arrow into Ω, we can take its transpose, which we call $\{\cdot\}_A := \overline{\delta_A}$. The relevant diagram is

$$\begin{tikzcd} A \times A \arrow[r, "\delta_A"] \arrow[d, "A \times \{\cdot\}_A"'] \arrow[dr, phantom, "(\!|\{\cdot\}|\!)"] & \Omega \arrow[d, equal] \\ A \times PA \arrow[r, "\in_A"'] & \Omega \end{tikzcd}$$

and the Twelf code is

```
e_sing = [A] transp (e_kron A).
sing = e_sing A.
```

In **Sets**, the morphism $\{\cdot\}_A : A \to PA$ is the function which takes an element $a \in A$ to $\{a' \mid a = a'\} = \{a\} \in PA$, and so in a general topos we think of it as taking a generalized element $x : X \to A$ to the subset $\{\cdot\}_A \circ x : X \to PA$ which 'contains exactly the element x'.

The next thing we need to express is the predicate 'is a singleton' on PA. To do this, we prove

Proposition 3.1 $\{\cdot\}_A$ *is a monomorphism.*

Proof Let $f, f' : X \to A$, be given, and suppose $\{\cdot\}_A f \overset{(\flat)}{=} \{\cdot\}_A f'$. Then the diagram

$$
\begin{tikzcd}
 & A \arrow[r, "!"] \arrow[d, "\Delta_A"'] \arrow[dr, phantom, "\llparenthesis \delta \rrparenthesis"] & 1 \arrow[d, "\mathsf{true}"] \\
A \times X \arrow[r, "A \times f"'] \arrow[dd, equal] & A \times A \arrow[r, "\delta_A"'] \arrow[d, "A \times \{\cdot\}_A \llparenthesis \{\cdot\} \rrparenthesis"] & \Omega \arrow[d, equal] \\
 \arrow[r, phantom, "A \times (\flat)"] & A \times PA \arrow[r, "\in_A"] & \Omega \arrow[d, equal] \\
A \times X \arrow[r, "A \times f'"] & A \times A \arrow[u, "A \times \{\cdot\}_A \llparenthesis \{\cdot\} \rrparenthesis"] \arrow[r, "\delta_A"] & \Omega \\
X \arrow[u, "(f', 1)"] \arrow[r, "f'"'] \arrow[ur, phantom, "(\star)"] & B \arrow[u, "\Delta_A"] \arrow[r, "!"'] \arrow[ur, phantom, "\llparenthesis \delta \rrparenthesis"] & 1 \arrow[u, "\mathsf{true}"']
\end{tikzcd}
$$

commutes. Chasing the diagram shows that $!f'$ and $\langle f', \mathrm{id}_X \rangle (A \times f)$ satisfy the conditions for the top right pullback, giving a unique mediating map h which has the property that $\Delta_A h = (f', 1)(A \times f)$. It is easy to check that $\langle f', \mathrm{id}_X \rangle (A \times f) = \langle f', f \rangle$, and $\Delta_A h = \langle h, h \rangle$, so we conclude $\langle h, h \rangle = \langle f', f \rangle$, and so $f = h = f'$. ■

To implement this proof, we begin by implementing the commutativity of the square $(\star)$:

```
obv_com : diag @ F’ == (id ** F’) @ (pair F’ id)
        =
          distp then
          =pair= id_l id_l then
          (sym (=pair=
                ((=@= refl prod_l) then id_l)
                ((=@= refl prod_r) then id_r)
                )) then
          =pair= ass ass then
          sym distp.
```

The diagram-chasing which sets up the first pullback can be implemented as

```
sm_helper = (=@= (sym pofp1) refl)  then
```

```
        sym ass then
        (=@= refl ass) then
        (=@= refl (=@= (**@) refl)).

sm_setup : sing @ F' == sing @ F ->
          true @ drop == kron @ ((id ** F) @ (pair F' id))
          =
            [P : sing @ F' == sing @ F]
            (=@= refl term_u) then
            ass then
            (sym (ass then =@= socp1 refl)) then
            (=@= refl (=@= mor==s refl)) then
            (=@= refl obv_com) then
            sm_helper then
            (=@= refl (=@= (=**= refl P) refl)) then
            sym sm_helper.
```

Then we have $(A \times f)\langle f', \mathrm{id}\rangle = \langle f', f\rangle$, as witnessed by the proof

```
sm_lemma**@ : (id ** F) @ (pair F' id) == pair F' F
         = distp then
            sym (=pair= ass ass) then
            =pair=
            (=@= refl prod_l then id_l)
            (=@= refl prod_r then id_r).
```

so we actually know that

```
sm_setup' : sing @ F' == sing @ F
             -> true @ drop == kron @ pair F' F
                 = [P] sm_setup P then
                   =@= refl sm_lemma**@.
```

The mediating map is obtained by

```
sm_map1 = [P : sing @ F' == sing @ F]
    cover sdiag (pair F' F) (sym(sm_setup' P)).
```

We can easily obtain

```
sm_penult : {P : sing @ F' == sing @ F}
            diag @ (sm_map1 P) == pair F' F
        = [P : sing @ F' == sing @ F]
 (=@= (sym mor==s) refl) then cover_r.
```

but we can also prove as a lemma that if $\Delta \circ f = \langle b_1, b_2\rangle$, then $b_1 = b_2$:

```
sm_lemma_obv1 : diag @ F == pair B1 B2
             -> F == B1
```

```
                  =
                     [P : diag @ F == pair B1 B2]
                     sym id_l then
                     sym prod_l then
                     =@= refl (sym distp then P) then
                     prod_l.

sm_lemma_obv2 : diag @ F == pair B1 B2
             -> F == B2
                  =
                     [P : diag @ F == pair B1 B2]
                     sym id_l then
                     sym prod_r then
                     =@= refl (sym distp then P) then
                     prod_r.

sm_lemma_obv : diag @ F == pair B1 B2
            -> B1 == B2
                  =
                    [P : diag @ F == pair B1 B2]
                    sym (sm_lemma_obv1 P) then sm_lemma_obv2 P.
```

which allows us to conclude that $f = f'$, so $\{\cdot\}_A$ is monic.

```
e_ssing = [A] mors (e_sing A)
       [Z] [F'] [F] [P : sing @ F' == sing @ F]
         sm_lemma_obv(sm_penult P).

ssing = e_ssing A.
```

Since $\{\cdot\}_A$ is monic, we can construct its characteristic function, which we call σ_A.

$$\begin{tikzcd}
A \arrow[r, "!"] \arrow[d, tail, "\{\cdot\}_A"'] \arrow[dr, phantom, "(\!|\sigma|\!)"] & 1 \arrow[d, "\mathsf{true}"] \\
PA \arrow[r, "\sigma_A"'] & \Omega
\end{tikzcd}$$

```
e_sigma = [A] char (e_ssing A).
sigma = e_sigma A.
```

In **Sets**, the morphism σ_A is indeed the predicate which is true when its argument is a singleton subset of A, i.e., in the range of $\{\cdot\}_A$.

Now we want to define the 'relation evaluation map' $v : B \times P(B \times C) \to PC$. Since $\in_{C\times B}$ is an arrow $C \times B \times P(C \times B) \to \Omega$, it seems sensible to define v

as $\overline{\in_{C\times B}} : B \times P(C \times B) \to PC$.

$$\begin{tikzcd}
C \times B \times P(C \times B) \arrow[r, "\in_{C\times B}"] \arrow[d, "C \times v"'] \arrow[dr, phantom, "(\!| v |\!)"] & \Omega \arrow[d, equal] \\
C \times PC \arrow[r, "\in_C"'] & \Omega
\end{tikzcd}$$

However, if we try to encode this in the obvious way,

```
e_vm = [B] [C] transp (e_in (C * B)).
vm = e_vm B C.
```

we get the result

```
readDecl
e_vm = [B] [C] transp (e_in (C * B)).%.
e_vm : {B:obj} {C:obj} mor (p (C * B)) (p (C * B))
   = [B:obj] [C:obj] transp (e_in (C * B)).
%% OK %%
readDecl
vm = e_vm B C.%.
vm : {C:obj} {B:obj} mor (p (C * B)) (p (C * B))
= [C:obj] [B:obj] e_vm B C.
%% OK %%
```

Here v has the type $P(C\times B) \to P(C\times B)$, not $B\times P(B\times C) \to PC$, as we had expected. What happened here is that the type of $\in_{B\times C}$ is $(B\times C)\times P(B\times C) \to \Omega$, and so transposing it takes all of $B \times C$ over to the codomain side of the arrow. We wanted just the B to move, and so the intended type of $\in_{B\times C}$ was the binary product $B \times (C \times P(B \times C))$. Since products are associative up to isomorphism, it is common to conceptually identify $X\times(Y\times Z)$ and $(X\times Y)\times Z$. However, the framework makes no such identification! In fact, tracking exactly how products are associated determines how formal transposition behaves, as this example demonstrates.

In order to get the correct transposition, we define the canonical 'associator' arrows

```
assr_l : mor (A * (B * C)) ((A * B) * C)
      = pair (pair fst (fst @ snd)) (snd @ snd).
assr_r : mor ((A * B) * C) (A * (B * C))
      = pair (fst @ fst) (pair (snd @ fst) snd).
```

Verifying that `assr_l` and `assr_r` are inverses of each other is a simple[5] calculation using `distp`:

[5] Although, for reasons not entirely clear, slow for twelf to verify.

```
assr_rl==id : assr_r @ assr_l == id
            = distp then
             =pair= (sym ass)
                        (distp then =pair= (sym ass) refl) then
             =pair=
             ((=@= refl prod_l) then prod_l)
             (=pair= ((=@= refl prod_l) then prod_r) prod_r)
             then =pair= refl prod_u then
             sym (=pair= id_r id_r) then
             prod_u.

assr_lr==id : assr_l @ assr_r == id
            =  distp then
             =pair= (distp then =pair= refl (sym ass))
                        (sym ass) then
             =pair=
             (=pair= prod_l ((=@= refl prod_r) then prod_l))
             ((=@= refl prod_r) then prod_r) then
              =pair= prod_u refl then
             sym (=pair= id_r id_r) then
             prod_u.
```

We then define

```
e_vm = [B] [C] transp ((e_in(C * B)) @ assr_l).
vm = e_vm B C.
```

and verify that the typing

```
readDecl
e_vm = [B] [C] transp ((e_in(C * B)) @ assr_l).%.
e_vm : {B:obj} {C:obj} mor (B * p (C * B)) (p C)
   = [B:obj] [C:obj] transp (e_in (C * B) @ assr_l).
%% OK %%
readDecl
vm = e_vm B C.%.
vm : {B:obj} {C:obj} mor (B * p (C * B)) (p C)
= [B:obj] [C:obj] e_vm B C.
%% OK %%
```

is as desired.

Finally, we use the arrows σ_A and v together to obtain the morphism $u := \overline{\sigma_C v} : P(C \times B) \to PB$, whose intended meaning is the function which takes a relation R to the subset of the domain of R where R single-valued.

$$\begin{tikzcd} B \times P(C \times B) \arrow[r, "v"] \arrow[d, "B \times u"'] \arrow[dr, phantom, "(\!| u |\!)"] & PC \arrow[d, "\sigma_C"] \\ B \times PB \arrow[r, "\in_B"'] & \Omega \end{tikzcd}$$

```
e_um = [B] [C] transp ((e_sigma C) @ (e_vm B C)).
um = e_um B C.
```

Functions are precisely the relations which are *always* single-valued, so we want to find the inverse image under u of the *maximal* subobject of B. Explicitly, we first take the maximal subobject $B \overset{\mathrm{id}_B}{\rightarrowtail} B$ of B and find its characteristic morphism $B \xrightarrow{\mathsf{char}\,\mathrm{id}_B} \Omega$. Then by $(\!| \Omega |\!)$, we have that $\mathsf{char}\,\mathrm{id}_B = (\mathsf{char}\,\mathrm{id}_B) \circ \mathrm{id}_B = \mathsf{true} \circ !$. Define $\mathsf{true}_B := \mathsf{char}\,\mathrm{id}_B = \mathsf{true} \circ !$.

$$\begin{tikzcd} B \arrow[r, equal] \arrow[d, "!"'] \arrow[dr, phantom, "(\!| \mathsf{true}_B |\!)"] & B \arrow[d, "\mathsf{true}_B"] \\ 1 \arrow[r, "\mathsf{true}"'] & \Omega \end{tikzcd}$$

Notice that for any arrow $f : B \to \Omega$ we can use the canonical isomorphism $\pi_1 : B \times 1 \cong B$ to take the transpose of $f\pi_1 : B \times 1 \to \Omega$ and obtain $\overline{f\pi_1} : 1 \to PB$. This global element of PB is called the 'name of f', written $\ulcorner f \urcorner$. We implement this as

```
nameof : mor B o -> mor 1 (p B)
              = [F] transp (F @ fst).
```

$$\begin{tikzcd} B \times 1 \arrow[r, "B \times \ulcorner f \urcorner"] \arrow[d, "\pi_1"'] \arrow[dr, phantom, "(\!| \ulcorner f \urcorner |\!)"] & B \times PB \arrow[d, "\in_B"] \\ B \arrow[r, "f"'] & \Omega \end{tikzcd}$$

Specifically, we care about the name $\ulcorner \mathsf{true}_B \urcorner$ of the morphism true_B. It is easy to show that any arrow whose domain is the terminal object is monic, and so $\ulcorner \mathsf{true}_B \urcorner : 1 \to PB$ is a subobject of the powerobject PB. In **Sets**, it is a function whose range is the singleton set $\{B\} \subseteq PB$, containing only the maximal subset of B, namely B itself.

The category-theoretic analogue of inverse image is pullback, so we pull $\ulcorner \mathsf{true}_B \urcorner$ back across u, defining C^B and π as the pullback object and second projection, respectively:

$$
\begin{tikzcd}
C^B \arrow[r, "!"] \arrow[d, "\pi"'] \arrow[dr, phantom, "(\!|\mathsf{exp}|\!)"] & 1 \arrow[d, "\ulcorner \mathsf{true}_B \urcorner"] \\
P(C \times B) \arrow[r, "u"'] & PB
\end{tikzcd}
$$

To formalize the exponential object, we again take advantage of the arrows we defined with explicit dependence on objects:

```
=> = [B] [C] (nameof(true @ drop) # (e_um B C)).
%infix right 7 =>.
```

π is then coded in a straightforward manner, with an ascribed type that uses the type family `=>` we have just defined:

```
pi : mor (B => C) (p (C * B)) = #snd (nameof(true @ drop)) um.
```

3.3 The Evaluation Map

The evaluation map is obtained by chasing the following diagram:

$$
\begin{tikzcd}
B \times C^B \arrow[rrr, dotted, bend left, "\mathsf{app}"] \arrow[rrr, phantom, bend left=20, "(\!|\mathsf{app}|\!)"] \arrow[r, "B \times \pi"] \arrow[d, "B \times !"'] \arrow[dr, phantom, "B \times (\!|\mathsf{exp}|\!)"] & B \times P(C \times B) \arrow[r, "v"] \arrow[d, "B \times u"] \arrow[dr, phantom, "(\!|u|\!)"] & PC \arrow[d, "\sigma_C"] \arrow[dr, phantom, "(\!|\sigma|\!)"] & C \arrow[l, "\{\cdot\}_C"'] \arrow[d, "!"] \\
B \times 1 \arrow[r, "B \times \ulcorner \mathsf{true}_B \urcorner"'] \arrow[d, equal] \arrow[drr, phantom, "(\!|\ulcorner \mathsf{true}_B \urcorner|\!)"] & B \times PB \arrow[r, "\in_B"'] & \Omega & 1 \arrow[l, "\mathsf{true}"] \arrow[dl, phantom, "(\!|\mathsf{true}_B|\!)"] \\
B \times 1 \arrow[rr, "\pi_1"] & & B \arrow[u, "\mathsf{true}_B"] \arrow[r, equal] & B \arrow[u, "!"']
\end{tikzcd}
$$

Since $(\!|\sigma|\!)$ is a pullback and $\mathsf{true}!\pi_1(B\times!) = \sigma_C v(B \times \pi)$, there exists a unique map $\mathsf{app} : B \times C^B \to C$ such that $(\!|\mathsf{app}|\!)$ commutes. The diagram-chasing is represented by several lemmas

```
app_lemma1 : true @ ((drop @ fst) @ (id ** drop))
          == in @ ((id @ id) ** (nameof(true @ drop) @ drop))
          =   ass then
            (=@= (sym pofp1) refl) then
            (sym ass) then
            (=@= refl **@) then
            (=@= refl (=**= refl (=@= (=transp= ass) refl))).

app_lemma2 : um @ pi == nameof (true @ drop) @ drop
          =  (sym #com) then
            =@= refl (sym term_u).

app_lemma3 : true @ ((drop @ fst) @ (id ** drop))
          == sigma @ (vm @ (id ** pi))
          = app_lemma1 then
            (=@= refl (=**= refl (sym app_lemma2))) then
            (=@= refl (sym **@)) then
            ass then
            (=@= pofp1 refl) then
            sym ass.

app_lemma4 : sigma @ (vm @ (id ** pi)) == true @ drop
          = sym app_lemma3 then
            =@= refl (sym term_u).
```

which we then plug into `cover` to get the unique mediating map app:

```
app : mor (B * (B => C)) C
      = cover ssing (vm @ (id ** pi)) app_lemma4.
```

3.4 The Currying Function

The morphism app allows us to take an arrow $g : A \to C^B$ and obtain an arrow $\mathsf{app} \circ (B \times g) : B \times A \to C$. The other half of the construction requires producing a function cur which takes a morphism $f : B \times A \to C$ and returns a morphism $\mathsf{cur}(f) : A \to C^B$. To find $\mathsf{cur}(f)$, recall from Section 3.1 that we want to have

$$\mathsf{app} \circ (B \times \mathsf{cur}(f)) = f \qquad (\beta)$$

If we suppose that this equation holds, then we have that the diagram

$$\begin{tikzcd}
B \times A \arrow[r, "f"] \arrow[dr, "B \times \mathsf{cur}(f)"'] & C \arrow[r, "\{\cdot\}_C"] & PC \\
 & B \times C^B \arrow[u, "\mathsf{app}"] \arrow[r, "B \times \pi"'] \arrow[ur, phantom, "(\!|\mathsf{app}|\!)"] & B \times P(C \times B) \arrow[u, "v"']
\end{tikzcd}$$

commutes. We can hit this with $C \times$ — and get

$$
\begin{tikzcd}
 & & & & \Omega \\
C \times B \times A \arrow[r, "C \times f"] \arrow[dr, "C \times B \times \mathsf{cur}(f)"'] & C \times C \arrow[rrru, bend left, "\delta_C"] \arrow[rr, "C \times \{\cdot\}_C"] & \arrow[u, phantom, "(\!|\{\cdot\}|\!)"] & C \times PC \arrow[ur, "\in_C"] & \arrow[d, phantom, "(\!|v|\!)"] \\
 & C \times B \times C^B \arrow[u, "C \times \mathsf{app}"] \arrow[rr, "C \times B \times \pi"'] & \arrow[u, phantom, "C \times (\!|\mathsf{app}|\!)"] & C \times B \times P(C \times B) \arrow[u, "C \times v"] \arrow[ruu, "\in_{C \times B}"'] &
\end{tikzcd}
$$

Now we see that $\pi\mathsf{cur}(f)$ is the transpose of $\delta_C(C \times f)$. We know π is monic because it is the pullback of an arrow out of 1, so $\mathsf{cur}(f)$ (if it exists) is uniquely determined as the factorization of $\overline{\delta_C(C \times f)}$ through π. Taking this hint, we set $h := \overline{\delta}_C(C \times f)$. Formally, we have (thinking of h as depending on f)

```
h : mor (B * A) C -> mor A (p (C * B))
    = [F : mor (B * A) C] transp ((kron @ (id ** F)) @ assr_r).
```

An `assr_r` appears here for the same reason that `assr_l` appeared in the formal definition of v, namely to make sure that the 'right' transposition occurs. We also prove that h is a congruence:

```
=h= : (F == F' -> h F == h F')
      = [P] =transp= (=@= (=@= refl (=**= refl P)) refl).
```

We have that the diagram

$$
\begin{tikzcd}
C \times (B \times A) \arrow[d, "C \times f"'] & (C \times B) \times A \arrow[l, "\sim"'] \arrow[r, "(C \times B) \times h"] \arrow[d, phantom, "(\!|h|\!)"] & (C \times B) \times P(C \times B) \arrow[d, "\in_{C \times B}"] \\
C \times C \arrow[rr, "\delta_C"'] & & \Omega
\end{tikzcd}
$$

commutes, by the definition of transposition, so we also have that

$$
\begin{tikzcd}
C \times (B \times A) \arrow[r, "\sim"] \arrow[d, "C \times f"'] & (C \times B) \times A \arrow[r, "(C \times B) \times h"] & (C \times B) \times P(C \times B) \arrow[d, "\in_{C \times B}"] \\
C \times C \arrow[rr, "\delta_C"'] & & \Omega
\end{tikzcd}
$$

commutes since the associator arrows are isomorphisms. We encode this fact as

```
cur_lemma0 : in @ ((id ** h F) @ assr_l) == kron @ (id ** F)
           = ass then
             =@= pofp1 refl then
             sym ass then
             =@= refl assr_rl==id then
             id_r.
```

We want to transpose (with respect to C this time, instead of $C \times B$) both sides of the equation

$$\in_{C\times B} ((C \times B) \times h) = \delta_C(C \times f)$$

that the diagram represents and conclude that

$$v(B \times h) = \{\cdot\}_C f$$

In general, the way to see that $h = \bar{k}$ for some arrows $h : Y \to PX$, $k : X \times Y \to \Omega$ is to show that $\in_X \circ(X \times h) = k$, and apply uniqueness of the transpose. This can be formalized as a lemma:

```
transp_trick: untransp(H) == K -> H == transp(K)
              = [P] sym pofp2 then =transp= P.
```

We use this first to show that

$$\overline{\delta_C(C \times f)} = \{\cdot\}_C f$$

```
cur_lemma1 : in @ (id ** (sing @ F)) == kron @ (id ** F)
           = (=@= refl id**@) then
             ass then
             (=@= pofp1 refl).

cur_lemma2 : sing @ F == transp (kron @ (id ** F))
           = transp_trick cur_lemma1.
```

and then we want to show that

$$\overline{\in_{C\times B} ((C \times B) \times h)} = v(B \times h)$$

If we proceed in the obvious way,

```
cur_lemma10 : in @ (id ** (vm @  Y)) == in @ (assr_l @ (id ** Y))
    = (=@= refl id**@) then
      ass then
      (=@= pofp1 refl) then
      (sym ass).

cur_lemma11 : vm @ Y == transp(in @ (assr_l @ (id ** Y)))
            = transp_trick cur_lemma10.
```

the result looks very much like the goal, except it is 'associated the wrong way'. Therefore we work towards proving `assr_l @ (id ** (id ** Z)) == (id ** Z) @ assr_l`. First we expand out the right side `(id ** Z) @ assr_l` using `distp`:

```
cur_lemma20 : (id ** Z) @ assr_l
                == pair (pair fst (fst @ snd)) ((Z @ snd) @ snd)
            = distp then
                =pair= (=@= id_l refl) (sym ass) then
                =pair= prod_l (=@= refl prod_r then ass).
```

Then we expand out the left side `assr_l @ (id ** (id ** Z))` in the same way:

```
cur_lemma30 :  (assr_l @ X) ==
            (pair (pair (fst @ X) ((fst @ snd) @ X))
                   (snd @ (snd @ X)))
            = distp then (=pair= distp (sym ass)).

cur_lemma31 :  (assr_l @ (id ** Y)) ==
            (pair (pair fst (fst @ (Y @ snd)))
                  (snd @ (Y @ snd)))
            = cur_lemma30 then
              (=pair= (=pair= (prod_l then id_l)
                      (sym ass then (=@= refl prod_r)))
              (=@= refl prod_r)).

cur_lemma32 : (assr_l @ (id ** (id ** Z))) ==
            (pair (pair fst (fst @ snd)) ((Z @ snd) @ snd))
            = cur_lemma31 then
              (=pair= (=pair=
                        refl
                        (ass then =@= (prod_l then id_l) refl))
              (ass then =@= prod_r refl)).
```

Now we simply realize that these two results are the same, and use `then` to obtain the desired

```
cur_lemma40 : assr_l @ (id ** (id ** Z)) == (id ** Z) @ assr_l
            = cur_lemma32 then (sym cur_lemma20).

cur_lemma41 : vm @ (id ** Z) == transp(in @ ((id ** Z) @ assr_l))
            = cur_lemma11 then =transp= (=@= refl cur_lemma40).
```

Since `transp` is a congruence, we conclude at last

```
cur_lemma50 : vm @ (id ** h F) == sing @ F
            =  cur_lemma41  then
              =transp= cur_lemma0 then
              sym cur_lemma2.
```

So we have $v(B \times h) = \{\cdot\}_C f$; left-compose both sides of this equation with σ_C and observe the commutativity of the diagram

$$
\begin{tikzcd}
B \times A \arrow[r, "f"] \arrow[d, "B \times h"'] & C \arrow[r, "!"] \arrow[d, "\{\cdot\}_C"] \arrow[dr, phantom, "(\!|\sigma|\!)"] & 1 \arrow[d, "\mathsf{true}"] \\
B \times P(B \times C) \arrow[r, "v"'] & PC \arrow[r, "\sigma_C"'] & \Omega
\end{tikzcd}
$$

to obtain $\sigma_C v(B \times h) = \mathsf{true}_C f$:

```
cur_lemma51 : sigma @ (vm @ (id ** h F)) == (true @ drop) @ F
            = =@= refl cur_lemma50 then
              ass then
              =@= (=@= refl (sym mor==s)) refl then
              =@= socp1 refl.
```

Finally, we want to transpose both sides of the equation again. First we have that $\overline{\sigma_C v(B \times h)} = uh$ by using $(\!| u |\!)$, which is coded as

```
cur_lemma60 : in @ (id ** (um @ h F))
              == sigma @ (vm @ (id ** h F))
            = =@= refl id**@ then
              ass then
              =@= pofp1 refl then
              (sym ass).
cur_lemma61 : um @ h F == transp (sigma @ (vm @ (id ** h F)))
            = transp_trick cur_lemma60.
```

The other side is just an exercise in using the uniqueness of arrows into the terminal object to show that $\overline{\mathsf{true}_C f} = \ulcorner \mathsf{true}_B \urcorner \circ !$.

```
cur_lemma70 : in @ (id ** (nameof (true @ drop) @ drop)) ==
            (true @ drop) @ F
            = =@= refl id**@ then
              ass then
              =@= (pofp1 then sym ass then =@= refl (sym term_u))
                   refl then
              sym ass then
              =@= refl (sym term_u then term_u) then
              ass.

cur_lemma71 : nameof (true @ drop) @ drop
              == transp ((true @ drop) @ F)
            = transp_trick cur_lemma70.
```

We conclude, then, that $uh = \ulcorner \mathsf{true}_B \urcorner \circ !$, but this is the pullback condition on $(\!|\,\mathsf{exp}\,|\!)$; The outer square of

$$
\begin{tikzcd}
A \arrow[rr, "!"] \arrow[dr, dashed, "\mathsf{cur}(f)"] \arrow[ddr, swap, "h"] & & \\
 & C^B \arrow[r, "!"] \arrow[d, "\pi"] \arrow[dr, phantom, "\lrcorner", very near start] \arrow[dl, phantom, "(\!|\,\mathsf{cur}\,|\!)"] \arrow[dr, phantom, "(\!|\,\mathsf{exp}\,|\!)"] & 1 \arrow[d, "\ulcorner \mathsf{true}_B \urcorner"] \\
 & P(C \times B) \arrow[r, swap, "u"] & PB
\end{tikzcd}
$$

commutes, so we get a map $\mathsf{cur}(f)$ factoring h through π as desired. This is coded as

```
cur_lemma80 = cur_lemma61 then
              =transp= cur_lemma51 then
              sym cur_lemma71.

cur : mor (B * A) C -> mor A (B => C)
        = [F] #pair drop (h F) (sym cur_lemma80).
```

and we can easily show that `cur` is a congruence using `=#pair=` and the already established fact that h is a congruence:

```
=cur= : (F == F' -> cur F == cur F')
        = [P] =#pair= refl (=h= P).
```

3.5 Verifying Beta- and Eta-Conversion

First make the abbreviation

```
uncur : mor A (B => C) -> mor (B * A) C
        = [F] app @ (id ** F).
```

Much like with `transp` and `untransp` we want to have `uncur(cur F) == F`, equation (β) from Section 3.1, and `F == cur(uncur F)`, equation (η).

To show the first equation holds, we show that for either of $\mathsf{uncur}(\mathsf{cur}(f))$ or f, if we left-compose with $\{\cdot\}_C$ and then 'untranspose', we get $\delta_C(C \times f)$. This is sufficient, because of the uniqueness of the transpose, and the fact that $\{\cdot\}_C$

is monic. This holds for $\mathsf{uncur}(\mathsf{cur}(f))$ because the diagram

$$
\begin{tikzcd}
C \times B \times A \arrow[r, equal] \arrow[d, "C \times B \times \mathsf{cur}(f)"'] \arrow[dr, phantom, "C \times B \times \llparenthesis \mathsf{cur} \rrparenthesis"] & C \times B \times A \arrow[r, "C \times f"] \arrow[d, "C \times B \times h"] \arrow[dr, phantom, "\llparenthesis h \rrparenthesis"] & C \times C \arrow[d, "\delta_C"] \\
C \times B \times C^B \arrow[r, "C \times B \times \pi"'] \arrow[d, "C \times \mathsf{app}"'] \arrow[dr, phantom, "C \times \llparenthesis \mathsf{app} \rrparenthesis"] & C \times B \times P(C \times B) \arrow[r, "\in_{C \times B}"'] \arrow[d, "C \times v"] \arrow[dr, phantom, "\llparenthesis v \rrparenthesis"] & \Omega \arrow[d, equal] \\
C \times C \arrow[r, "C \times \{\cdot\}_C"'] & C \times PC \arrow[r, "\in_C"'] & \Omega
\end{tikzcd}
$$

commutes, and it holds for f by $\llparenthesis \{\cdot\} \rrparenthesis$. This reasoning is formalized as

```
exp_lemma0 : smor ssing @ (app @ (id ** cur F))
           == vm @ (id ** h F)
           = ass then
             =@= cover_r refl then
             sym ass then
             =@= refl (sym id**@ then =**= refl pull_r).

exp_lemma1 : in @ ((id ** transp (F @ G)) @ H) == F @ (G @ H) =
           ass then
           =@= pofp1 refl then
           sym ass.

exp_lemma_i: untransp((smor ssing) @ uncur(cur F))
             == kron @ (id ** F)
           = =@= refl (=**= refl exp_lemma0 then id**@) then
             exp_lemma1 then
             =@= refl cur_lemma40 then
             exp_lemma1 then
             =@= refl assr_rl==id then
             id_r.

exp_lemma_ii: untransp((smor ssing) @ F) == kron @ (id ** F)
            = =@= refl id**@ then
              ass then
              =@= (=@= refl (=**= refl mor==s) then pofp1)
                   refl.
```

Here we apply uniqueness of the transpose to conclude that $\mathsf{uncur}(\mathsf{cur}(f))$ and f are equal after composition with $\{\cdot\}_C$

```
exp_beta' : (smor ssing) @ uncur(cur F)
```

```
            ==  (smor ssing) @ F
            = sym pofp2 then
              =transp= (exp_lemma_i then sym exp_lemma_ii) then
              pofp2.
```

and we can use the fact that $\{\cdot\}_C$ is monic to get the desired result,

```
exp_beta : uncur(cur F) == F
         = subobj_def exp_beta'.
```

It remains to show that η-expansion works. We show $\mathsf{cur}(\mathsf{uncur}(f)) = f$ by checking that $\pi \circ \mathsf{cur}(\mathsf{uncur}(f)) = \pi f$ and using the fact that π is monic.

By the definition of cur, we have that

$$\pi \circ \mathsf{cur}(\mathsf{uncur}(f)) = \overline{\delta_C(C \times \mathsf{uncur}(f))}$$

so it suffices to show that $\pi f = \overline{\delta_C(C \times \mathsf{uncur}(f))}$, and by uniqueness of the transpose it is enough to have $\in_{C\times B} (C \times B \times \pi f) = \delta_C(C \times \mathsf{uncur}(f))$. But this is just the commutativity of the top two squares in the above diagram; the diagram chase can be coded as

```
exp_cell0 : sing @ app == vm @ (id ** pi)
          = =@= (sym mor==s) refl then cover_r.

exp_lemma2 : kron @ (id ** uncur F) ==
           ((in @ assr_l) @ (id ** (id ** pi)))
                            @ (id ** (id ** F))
           = =@= (sym pofp1) id**@ then
             sym ass then
             =@= refl (ass then
                      =@= (sym id**@ then
                           =**= refl exp_cell0 then
                           id**@) refl) then
             ass then
             =@= ass refl then
             =@= (=@= pofp1 refl) refl.

exp_lemma3 : kron @ (id ** uncur F) ==
           (in @ ((id ** pi) @ (id ** F))) @ assr_l
           = exp_lemma2 then
             =@= (sym ass then =@= refl cur_lemma40) refl then
             sym ass then
             =@= refl
              (sym ass then =@= refl cur_lemma40 then ass) then
             ass.

exp_lemma4 : (kron @ (id ** uncur F)) @ assr_r ==
           in @ (id ** (pi @ F))
```

```
          = =@= exp_lemma3 refl then
            sym ass then
            =@= refl assr_lr==id then
            id_r then
            =@= refl (sym id**@).

exp_lemma5 : pi @ cur(uncur F) == pi @ F
          = pull_r then
            =transp= exp_lemma4 then pofp2.
```

We still need to formally arrive at the fact that π is monic. A proof of the fact that `pi @ F == pi @ G -> F == G` appears to be almost immediate; the hypothesis is the fact, by definition of `pi`, that `#snd (nameof (true @ drop)) um @ F == #snd (nameof (true @ drop)) um @ G`, and we can easily show

```
exp_lemma6 : #fst (nameof (true @ drop)) um @ F
          == #fst (nameof (true @ drop)) um @ G
           = =@= (sym term_u) refl then
             sym term_u then
             term_u then
             =@= term_u refl.
```

It seems plausible that application of uniqueness of pullback pairing should complete the proof. However, if we try to run the code

```
exp_lemma7 : (pi @ F == pi @ G -> F == G)
          = [P : pi @ F == pi @ G]
            sym pull_u then
            =#pair= exp_lemma6 P then pull_u.
```

we get

```
readDecl
exp_lemma7 : (pi @ F == pi @ G -> F == G)
          = [P : pi @ F == pi @ G]
            sym pull_u then
            =#pair= exp_lemma6 P then pull_u.%.
stdIn:2.14-4.46 Error:
No strict occurrence of implicit variable, use %abbrev
%% ABORT %%
```

because the proof variable involved in `#pair` does not satisfy the requirements of twelf's type reconstruction algorithm. We must provide more typing information; in this case, specializing `=#pair=` to

```
#com' : F @ (#fst F G @ H) == G @ (#snd F G @ H)
         =
          ass then
          =@= #com refl
```

```
           then (sym ass).

=#pair=' : #fst F G @ H == #fst F G @ J
              -> #snd F G @ H == #snd F G @ J
            -> #pair (#fst F G @ H) (#snd F G @ H) #com'
             == #pair (#fst F G @ J) (#snd F G @ J) #com'
              = =#pair=.
```

allows the code

```
exp_lemma6 : #fst (nameof (true @ drop)) um @ F
           == #fst (nameof (true @ drop)) um @ G
            = =@= (sym term_u) refl then
              sym term_u then
              term_u then
              =@= term_u refl.

exp_lemma7 : (pi @ F == pi @ G -> F == G)
           = [P : pi @ F == pi @ G]
             sym pull_u then
             =#pair=' exp_lemma6 P then pull_u.
```

to successfully typecheck. Finally, we have

```
exp_eta : cur (uncur F) == F
   = exp_lemma7 exp_lemma5.
```

which completes the proof.

3.6 Commuting the Products

For the sake of completeness, we can, as promised, also construct terms which have types `mor ((B => C) * B) C` and `mor (A * B) C -> mor A (B => C)` to conform with the original formulation of [Fil92]. It is not necessary to redo the entire construction to accomplish this; instead we can take advantage of the properties canonical isomorphism (sometimes called the 'twist' morphism) $\mathsf{tw}_{AB} : A \times B \to B \times A$ to derive the new exponential from the old with relatively little effort. tw is defined as $\langle \mathsf{snd}, \mathsf{fst} \rangle$, and is directly encoded as

```
tw : mor (A * B) (B * A)
= pair snd fst.
```

We will need to use the fact that tw is idempotent, so we code

```
tw_idem : tw @ tw == id
   = distp then
  =pair= prod_r prod_l then
  sym (=pair= id_r id_r) then
  prod_u.
```

The other relevant property is $\mathsf{tw} \circ (f \times g) = (g \times f) \circ \mathsf{tw}$, formalized as

```
tw**_lemma1 : tw @ (F ** G) == pair (G @ snd) (F @ fst)
   = distp then
  =pair= prod_r prod_l.

tw**_lemma2 : (F ** G) @ tw == pair (F @ snd) (G @ fst)
   = distp then
  sym (=pair= ass ass) then
  =pair= (=@= refl prod_l) (=@= refl prod_r).

tw** : tw @ (F ** G) == (G ** F) @ tw
= tw**_lemma1 then sym tw**_lemma2.
```

With the definition of tw at hand, defining the new versions of app and cur is a simple matter of inserting tw where appropriate:

```
app2 : mor ((B => C) * B) C
      = app @ tw.

cur2 : mor (A * B) C
      -> mor A (B => C)
         = [F] cur (F @ tw).
```

and proving that the same properties hold is an exercise in applying the lemmas we have just proven:

```
uncur2 : mor A (B => C) -> mor (A * B) C
       = [F] app2 @ (F ** id).

=cur2= : (F == F' -> cur2 F == cur2 F')
        = [P] =cur= (=@= P refl).

exp_beta2 :  app2 @ (pair ((cur2 F) @ fst) snd) == F
          = =@= refl (=pair= refl (sym id_l)) then
            sym ass then
            =@= refl tw** then
            ass then
            =@= exp_beta refl then
            sym ass then
            =@= refl tw_idem then
            id_r.

exp_eta2_lemma : ((app @ tw) @ (F ** id)) @ tw == app @ (id ** F)
               = sym ass then
                 =@= refl (sym tw**) then
                 ass then
                 =@= (sym ass then =@= refl tw_idem then id_r)
                      refl.
```

```
exp_eta2 : cur2 (app2 @ (pair (G @ fst) snd)) == G
         = =cur2= (=@= refl (=pair= refl (sym id_l))) then
            refl then
            =cur= exp_eta2_lemma then
            exp_eta.
```

References

[Fil92] Andrzej Filinksi. CCCs and typed λ-calculi. Unpublished manuscript, 1992.

[HP98] Robert Harper and Frank Pfenning. A module system for a programming language based on the LF logical framework. *J. Logic Computat.*, 8(1):5–31, 1998.

[Jac99] B. Jacobs. *Categorical Logic and Type Theory.* Elsevier, 1999.

[Kel82] G. M. Kelley. *Basic Concepts of Enriched Category Theory*, pages 30–31. Cambridge University Press, 1982.

[MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic.* Springer-Verlag, 1992.

[Pfe01] Frank Pfenning. Intensionality, extensionality and proof irrelevance in modal type theory. To appear, 2001.

The LaTeX source of this paper and the code described in it are available electronically at `http://functor.org/math/cd01.tar.gz`.
