Formally verifying currying via the product-exponential adjunction

Ciprian Stanciu g.c.stanciu@student.tudelft.nl $- \times b \quad \dashv \quad (-)^b$ Benedikt Ahrens, Lucas Escot

i. Introduction

Functional programming languages take advantage of concepts and terms from category theory in order to structure data and computation.

How can we make it easier for computer science students to understand categorical concepts in functional programming and category theory?

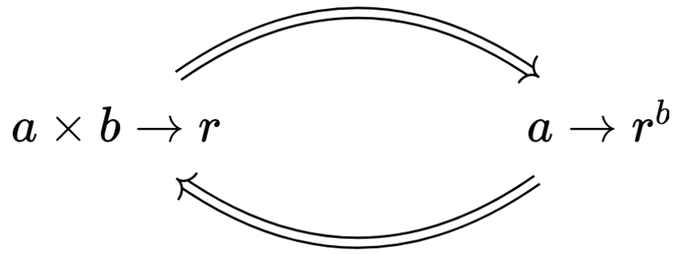
We developed a library of category theory definitions, theorems, proofs and examples in a computer proof assistant. It targets beginners, therefore it skips proof automation and limits future extensibility for closer correspondence to maths.

The main result in the library is a proof of the adjunction that generates currying as we know it in programming languages.

ii. Method

Lean's logical foundation allows us to be certain that if we can write a theorem and its proof without any errors, then it must be correct.

We followed established mathematical notation within the library, included descriptive comments where needed in proofs, and explicitly defined the lemmas involved.




```
f :: a -> (b -> r)
≡
f :: (a, b) -> r
```

Figure 1: **Currying**

Figure 2: **Maths-Lean correspondence**

$$\mathcal{D} \underset{R}{\overset{L}{\leftrightarrows}} \mathcal{C} \qquad \eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow R \circ L \qquad \epsilon : L \circ R \Rightarrow \mathrm{Id}_{\mathcal{D}}$$

$$L \xRightarrow{L\eta} LRL \xRightarrow{\epsilon L} L \;=\; L \xRightarrow{\mathrm{id}_L} L$$
$$R \xRightarrow{\eta R} RLR \xRightarrow{R\epsilon} R \;=\; R \xRightarrow{\mathrm{id}_R} R$$

```
structure adjunction_unit {C D : category} (L : functor C D) (R : functor D C) :=
(η : natural_transformation (Id C) (R∘L))
(ϵ : natural_transformation (L∘R) (Id D))
(id_L : ∀ (c : C), D.compose (ϵ.α (L c)) (L.map_hom (η.α c)) = D.id (L.map_obj c))
(id_R : ∀ (d : D), C.compose (R.map_hom (ϵ.α d)) (η.α (R d)) = C.id (R.map_obj d))
```

iii. Existing work

There are already many existing category theory libraries, but they are all targeted towards working categoricians who use them to prove more complex theorems and not to beginners.

Although they are incomparably broader in scope, our library also implements simpler examples or concepts that other implementations skip over.

iv. Design decisions

The process required taking decisions about the actual implementation of the mathematical code. Some of them were taken contrary to recommendations from literature, showing the difference between libraries targeting extensibility and ours.

- Family-of-collections-of-morphisms category definition.
- Structures instead of typeclasses.
- Bundling of parameters.
- Skolemization of existential qualifiers.
- Category universes.

v. Result and contributions

The final library proves the adjunction formed by the relation in figure 1 which is the equivalent of currying from a category theoretic perspective, showing why currying works in programming languages.

All the prerequisite concepts are built into the library (as a consequence of formal proving), however simpler examples are also included. For instance, the adjunction between product and diagonal functors, showing that we can freely create pairs.