Using Postulated Colimits in Coq

Work in Progress

March 4, 2018

Abstract

In this talk we define and construct finite colimits in the Coq proof assistant in a context that is similar to the category of sets. First we review without proof the key mathematical ideas involved in the theory of postulated colimits as described in a note of Anders Kock. This theory gives us a way to prove results about colimits in an arbitrary sheaf topos. Then we give an inductive definition in Coq of the fundamental notion of zigzag in this theory. We finish by proving the result analogous to the (mathematically easy) result that in the category of sets pushouts of monomorphisms are monomorphisms.

Contents

1 Postulated Coequalisers

We will use the traditional definition to define a coequaliser.

Definition 1.1. The *coequaliser q of a parallel pair $a, b : R \rightrightarrows X$* is an arrow

$$\begin{tikzcd}
R \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & X \arrow[r, two heads, "q"] \arrow[dr, "\forall z"'] & Q \arrow[d, dashed, "\exists!\psi"] \\
& & Z
\end{tikzcd}$$

such that

- the equality $sq = tq$ holds
- for all $z : X \to Z$ such that $sz = tz$ there exists a unique $\psi : Q \to Z$.

We will use the idea of a postulated colimit to give an explicit presentation of colimits in the category of sets. First we modify the definition of zigzag.

Definition 1.2. An *s-t-zigzag from $x \in X$ to $y \in X$* is a sequence $r_1, r_2, ..., r_n \in R$ such that

$$\begin{tikzcd}
 & r_1 \arrow[dl, maps to, "\phi_1"'] \arrow[dr, maps to, "\phi_2"] & & r_2 \arrow[dl, maps to, "\phi_3"'] \arrow[dr, maps to, "\phi_4"] & & r_3 \arrow[dl, maps to, "\phi_5"'] \arrow[dr, maps to, "\phi_6"] & & \ldots & & r_n \arrow[dl, maps to, "\phi_7"'] \arrow[dr, maps to, "\phi_8"] & \\
x & & x_1 & & x_2 & & {} & \ldots & {} & & y
\end{tikzcd}$$

where the $\phi_i \in \{s, t\}$.

Definition 1.3. A cofork

$$\begin{tikzcd}
R \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & X \arrow[r, "q"] & Q
\end{tikzcd}$$

is *a postulated coequaliser* iff

- the arrow q is an epimorphism
- for all $x, y \in X$ $(x)q = (y)q$ iff there exists an s-t-zigzag from x to y.

We can use postulated coequalisers to give a concrete presentation of coequalisers due to the following result of Kock.

Proposition 1.4. *In any Grothendieck toposes $\mathcal{E}$ every postulated coequaliser is a coequaliser.*

2 Objoids and Mapoids

In our formulation we replace a sets with objoids. An objoid is a type equipped with an equivalence relation on the terms of the type.

```
Record objoid: Type :=
  {carrier:>Type;
   eq:carrier->carrier->Prop;
   refl:reflexive carrier eq;
   sym:symmetric carrier eq;
   trans:transitive carrier eq}.
Infix "~" := eq (at level 95).
```

We replace functions with *mapoids* which are the appropriate structure preserving maps for objoids.

```
Structure mapoid (A B:objoid) :=
  {map:>carrier A -> carrier B;
   pres (a1 a2:carrier A) (H:a1~a2):map a1~map a2}.
```

Sometimes it is easier to use an infix for function application.

```
Definition application (a1:carrier A) (f1:mapoid A B) := map f1 a1.
Infix "|>" := application (at level 11, left associativity).
```

We use double pipe for composition.

```
Definition comp:=
  {|map:=fun a:carrier A=>g(f(a));
    pres:=comp_pres|}.
Infix "||>" := comp (at level 10, right associativity).
```

Finally we assert extensionality for functions.

```
Axiom mapoid_ext:
  (forall a:carrier A, f(a)~f2(a)) -> f=f2.
Axiom mapoid_app:
  f=f2 -> (forall a:carrier A, f(a)~f2(a)).
```

3 An Inductive Definition of Zigzag

It turns out that we can give an inductive definition of zigzag. Note that in order formulate this in Coq we need to add a base case and a way to cons equivalent elements onto a zigzag.

```
Inductive zigzag(x1 x2:X):Prop:=
  |xid (H:x1~x2):zigzag x1 x2
  |xcons {x3:X}: (x1~x3)-> (zigzag x3 x2) -> zigzag x1 x2
  |stcons{r:R}{x3:X} (H1:x1~(s(r))) (H2:x3~(t(r)))
         (z1:zigzag x3 x2): zigzag x1 x2
  |tscons{r:R}{x3:X} (H1:x1~(t(r))) (H2:x3~(s(r)))
         (z1:zigzag x3 x2): zigzag x1 x2.
```

Lemma 3.1. *The zigzag relation is reflexive, symmetric and transitive.*

This means that we can define the coequaliser as a postulated coequaliser.

```
Definition Q:objoid:=
  {|carrier:=X;
    eq:=zigzag;
    refl:=zigzag_refl;
    sym:=zigzag_sym;
    trans:=zigzag_trans|}.
Definition q:mapoid X Q:=
  {|map:=fun x:X => x:Q;
    pres:=id_pres|}.
```

Proposition 3.2. *The postulated coequaliser is a coequaliser.*

Proof. The factorisation is given by the identity

```
Definition fact_arrow(Z:objoid) (z:mapoid X Z):Q->Z:=
  fun x:Q => z(x).
Definition factorisation
  (Z:objoid) (z:mapoid X Z) (H:s||>z = t||>z):mapoid Q Z:=
  {|map:=fact_arrow Z z;
    pres:=(fact_arrow_pres Z z H)|}.
```

and the main work is to prove that $\sim$ is preserved. For this we refer to the proof in Coq. $\square$

4 Disjoint Unions

We have already defined and given a concrete presentation of coequalisers. In order to get all finite limits we only need to add finite coproducts. The natural translation of disjoint union into type theory is:

```
Inductive du:Type :=
  |b:B -> du
  |c:C -> du.
Inductive du_eq:relation du:=
  |beq(b1 b2:B) (H:b1~b2):du_eq (b b1) (b b2)
  |ceq(c1 c2:C) (H:c1~c2):du_eq (c c1) (c c2).
```

Proposition 4.1. *The disjoint unions defined above satisfy the universal property of coproducts.*

5 Pushouts from Coequalisers and Coproducts

In any category with finite colimits if the square

$$\begin{tikzcd}
A \arrow[r, "g"] \arrow[d, "f"] & B \arrow[d, "i_1"] \\
C \arrow[r, "i_0"] & Q
\end{tikzcd}$$

is a pushout then we can compute Q using a disjoint union and a coequaliser

$$\begin{tikzcd}
 & B \arrow[d, "u_0"] & \\
A \arrow[ur, "f"] \arrow[dr, "g"'] \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & B \amalg C \arrow[r, two heads, "q"] & Q \\
 & C \arrow[u, "u_1"] &
\end{tikzcd}$$

and where u_0 and u_1 are the coproduct inclusions $i0 = u0q$ and $i1 = u1q$.

References