Coq formalization of graph transformation

Samuel Arsac, supervised by Russ Harmer and Damien Pous
Équipe PLUME, LIP, ENS Lyon

March - July 2022

Contents

1 Introduction

The general context

Graph rewriting is a mathematical theory allowing for the definition of rewriting rules for graphs. This includes adding, deleting, merging and duplicating nodes and edges, but also modifiying associated data. A handbook of the general theory was published in 1997 [9].

It has applications in various domains, such as software engineering, databases, but also organic chemistry and molecular biology.

The subject of this internship is to formalize some useful categorical results for graph rewriting. There are already some Coq libraries for category theory, such as John Wiegley's *categories in Coq* [10] or the more general *UniMath* [2] and *HoTT* [8], both of which are based on the univalence axiom. There are also some libraries in other proof assistants, such as *agda-categories* [7], in Agda.

The research problem

Proofs of advanced results in graph rewriting can easily become convoluted due to the amount of morphisms they involve and thus be error-prone. Formalizing in Coq the categorical notions involved is a way to avoid this issue.

None of the previously mentioned libraries have all the required categorical constructions in order to formalize graph rewriting.

I focused on an article by Russ Harmer and Nicolas Behr about concurrency theorems in two rewriting frameworks [3].

Your contribution

Using John Wiegley's library [10] for some basic definitions, I defined some the constructions used in graph rewriting in Coq.

This includes some usual constructions such as pullbacks and other less usual notions such as final pullback complements and quasi-topoi.

I used these definitions to prove some auxiliary results in the article about concurrency theorems, laying a foundation for proving the main theorems.

I did not reach the proofs of these concurrency theorems, as I have not formalized all the required constructions.

The Coq code can be found in this repository, along with some documentation.

Arguments supporting its validity

When necessary, definitions written in Coq are proven equivalent to their usual mathematical definition. This ensures that they correspond to the expected notion, but also that they can be used in an intuitive way in a proof, while still allowing for some interesting properties in Coq.

I have defined several constructions in a way that allows linking them together, so as to make proving some usual results more efficient.

These definitions do not depend on any specific category, hence they can be used in a completely different context.

Summary and future work

Since only auxiliary results are defined and proven for now, an obvious future work is to define the few remaining constructions and prove the main concurrency theorems of the article [3].

Implementing graph categories in Coq could also be an interesting step, and then possibly to extract algorithms for graph transormation.

Another future work is a wiki for graph transformation with an underlying Coq formalization of the results. Reserchers behind this project include Nicolas Behr (IRIF), Russ Harmer and Damien Pous.

More generally, these foundations can be used for other results in graph rewriting, and in any other field making use of the same constructions, since they are purely categorical.

Notes and Acknowledgements

This internship report is written in English as it could possibly be of use in the previously mentionned wiki project.

I wish to thank Russ Harmer and Damien Pous for their helpful supervision during my internship. I would also like to thank Nicolas Behr for his insights on some parts of the article.

2 Background

The formalization aims at proving results on Sesqui-Pushout and Double-Pushout graph rewriting, as presented in [3]. For the sake of simplicity, I will present only Sesqui-Pushout rewriting here, but Double-Pushout rewriting is based on the same notions. Comments on the differences between the two approaches can be found in [4], section 1.2.

2.1 Quasi-topoi

The categories that will be used here for graph rewriting are called quasi-topoi. In order to define them, we need several auxiliary definitions.

We first define a class of monomorphisms with some stability properties (these monomorphisms will be noted $\rightarrowtail$ in the rest of the report, while $\twoheadrightarrow$ will be used for generic epimorphisms).

Definition 1 (Stable system of monics ([3], Definition 1))**.** *A stable system of monics $\mathcal{M}$ in a category* **C** *is a class of monomorphisms satisfying the following conditions:*

- *all isomorphisms are in $\mathcal{M}$,*
- *$\mathcal{M}$ is stable under composition,*
- *$\mathcal{M}$ is stable under pullbacks: if $C \xleftarrow{c} A \xrightarrow{a} B$ is a pullback of $B \xrightarrow{b} D \xleftarrow{d} C$ and b is in $\mathcal{M}$ then c is in $\mathcal{M}$.*

We then define subobject classifiers, which have an object Ω of "truth values", and which given a monomorphism $A \xrightarrow{m} B$ in $\mathcal{M}$, gives a characteristic morphism which is meant to identify the parts of B that are in the image of m and those that are not.

Definition 2 (Regular subobject classifier ([11], 14.1))**.** *In a category* **C** *with finite limits, a regular subobject classifier of $\mathcal{M}$ is a monomorphism $T \xrightarrow{T} \Omega$ where T is terminal, such that for any $A \xrightarrow{m} B$ in $\mathcal{M}$, there is a unique pullback square $A \xrightarrow{m} B \rightarrow \Omega \xleftarrow{\top} T \leftarrow A$.*

$$\begin{array}{ccc} A & \dashrightarrow & T \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \top} \\ B & \dashrightarrow & \Omega \end{array}$$

We can finaly define quasi-topoi:

Definition 3 (Quasi-topos ([3], Definition 3))**.** *A category is a quasi-topos if:*

- *it has finite limits and colimits,*
- *it is locally cartesian closed,*
- *it has a regular subobject classifier.*

2.2 Basic structures

2.2.1 Partial map classifiers

An $\mathcal{M}$-partial map is a span $A \xleftarrow{m} X \xrightarrow{f} B$, with m in $\mathcal{M}$. The idea is that we have a partial map from A to B, where X represents the elements on which the map is defined and m embeds X in A.

A classifier is a way to obtain a total map from the partial map by modifying the codomain. For **Set**, this can be done by adding a $\perp$ element to the codomain to which all elements that have no image by the partial map will be mapped.

Definition 4 ($\mathcal{M}$-partial map classifier ([3], Definition 2))**.** *Given a category* $\mathbf{C}$ *with a stable system of monics* $\mathcal{M}$*, an* $\mathcal{M}$*-partial map classifier* (T, η) *with* T *an endofunctor on* $\mathbf{C}$ *and* η *a natural transformation from* $ID_{\mathbf{C}}$ *to* T *such that:*

- *for any* X*,* $X \xrightarrow{\eta_X} T(X)$ *is in* $\mathcal{M}$*,*
- *for any* $\mathcal{M}$*-partial map* $A \xleftarrow{m} X \xrightarrow{f} B$*, there exists a unique morphism* $A \xrightarrow{\phi(m,f)} T(B)$ *such that* $X \xrightarrow{f} B \xrightarrow{\eta_B} T(B) \xleftarrow{\phi(m,f)} A \xleftarrow{m} X$ *is a pullback.*

$$\begin{array}{ccc} X & \xrightarrow{f} & B \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \eta_B} \\ A & \xrightarrow[\phi(m,f)]{} & T(B) \end{array}$$

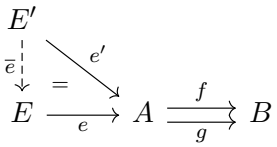
Lemma 1 ([3], Coroallary 1)**.** *A quasi-topos always has an* $\mathcal{M}$*-partial map classifier.*

2.2.2 Regular monomorphisms

We need to define first what an equalizer is.

Definition 5 (Equalizer ([1], 7.51))**.** *Given a pair of morphisms $A \xrightarrow{f} B$ and $A \xrightarrow{g} B$, a morphism $E \xrightarrow{e} A$ is an equalizer of f and g if:*

- $f \circ e = g \circ e$,
- *for any $E \xrightarrow{e'} A$ such that $f \circ e' = g \circ e'$, there exists a unique morphism $E' \xrightarrow{\overline{e}} E$ such that $e' = e \circ \overline{e}$.*



Definition 6 (Regular monomorphism ([1], 7.56))**.** *A regular morphism is a morphism that is the equalizer of a pair of morphisms.*

Regular monomorphisms are indeed monomorphisms, because of the universal property in the definition of equalizers (see [1] 7.57, or the proof in coq).

The idea behind ruglar monomorphisms is to represent embeddings, which is not always the case for generic monomorphisms (see [1] page 109 and 7.58).

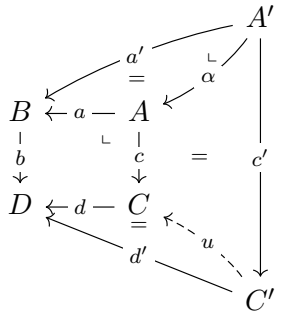
Lemma 2 ([3], Coroallary 1)**.** *A quasi topos always has a stable system of monics, which is the class of regular monos.*

2.2.3 Final Pullback Complements

We define here the notion of Final Pullback Complement (FPC), taking the definition from [5]. The idea is that given two composable morphisms $A \xrightarrow{a} B \xrightarrow{b} D$, a pullback complement is another pair $A \xrightarrow{c} C \xrightarrow{d} D$ such that the square they form is a pullback. It is final if there is a unique morphism from any "bigger" pullback to the complement.

Definition 7 (Final Pullback Complement (FPC) ([5], Definition 1))**.** *Given two composable morphisms $A \xrightarrow{a} B \xrightarrow{b} D$, an FPC of a and b is another pair $A \xrightarrow{c} C \xrightarrow{d} D$ such that*

- $A \xrightarrow{a} B \xrightarrow{b} D \xleftarrow{d} C \xleftarrow{c} A$ *is a pullback*
- *for any pullback* $A' \xrightarrow{a'} B \xrightarrow{b} D \xleftarrow{d'} C' \xleftarrow{c'} A$ *and any* $A' \xrightarrow{\alpha} A$ *such that* $a \circ \alpha = a'$*, there is a unique* $C' \xrightarrow{u} C$ *such that* $d \circ u = d'$ *and* $c \circ \alpha = u \circ c'$*.*



2.2.4 Multi-Pushout Complements

Pushout complements are ways to complete a square such that it is a pushout, similarly to pullback complements. However, contrary to FPCs, we take the set of all possible complements with an added condition related to $\mathcal{M}$.

Definition 8 ($\mathcal{M}$-Multi-Pushout Complement ([3], Definition 6))**.** *Given a category* $\mathbf{C}$ *with a stable system of monics* $\mathcal{M}$*, the* $\mathcal{M}$*-multi-pushout complement of two morphisms* $A \xrightarrow{a} B \xrightarrow{b} D$ *with* b *in* $\mathcal{M}$*, noted* $\mathcal{P}(a, b)$ *is the set of all pairs of morphisms* $A \xrightarrow{c} C \xrightarrow{d} D$ *such that:*

- c *is in* $\mathcal{M}$*,*
- $A \xrightarrow{a} B \xrightarrow{b} D \xleftarrow{d} C \xleftarrow{c} A$ *is a pushout.*

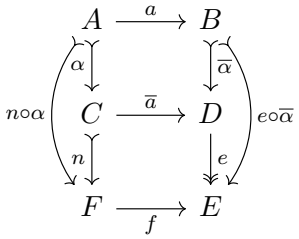
2.2.5 FPC augmentations

An $\mathcal{M}$-FPC augmentation (FPA) of a pushout square is a way to extend this square into an FPC.

Definition 9 (FPA ([3], Definition 7))**.** *Given a quasi-topos* $\mathbf{C}$ *with* $\mathcal{M} = rm(\mathbf{C})$ *and a pushout* $A \xrightarrow{a} B \xrightarrow{\overline{\alpha}} D \xleftarrow{\overline{a}} C \xleftarrow{\alpha} A$ *where* α *and* $\overline{\alpha}$ *are in* $\mathcal{M}$*, an FPA is triple* $(D \xrightarrow{e} E, C \xrightarrow{n} F, F \xrightarrow{f} E)$ *such that*

- *e is an epimorphism,*
- *$e \circ \overline{\alpha}$ and n are in $\mathcal{M}$,*
- *$A \xrightarrow{n \circ \alpha} D \xrightarrow{f} E$ is an FPC of $A \xrightarrow{a} B \xrightarrow{e \circ \overline{\alpha}} E$*

We note $FPA(\alpha, a)$ the set of all such triples.



2.2.6 Multi-sums

Multi-sums are an extension of the notion of coproduct to a family of objects.

Definition 10 ($\mathcal{M}$-Multi-sum)**.** *If $\mathbf{C}$ is a quasi-topos, the multi-sum of two object A and B, noted $\Sigma_{\mathcal{M}}(A, B)$, is a family of cospans $(A \xrightarrow{a_i} C_i \xleftarrow{b_i} B)_{i \in I}$ with (a_i) and (b_i) regular monomorphisms, such that for any cospan $A \xrightarrow{f} X \xleftarrow{g} B$, there is an i in I and a morphism $C_i \xrightarrow{x} X$ such that $f = x \circ a_i$ and $g = x \circ b_i$.*

2.3 Sesqui-pushout rewriting

Given a quasi-topos $\mathcal{C}$, we can define Sesqui-pushout rewriting (SqPO-rewriting). This makes sense when applied to categories of graphs (see [3], section 2.1)

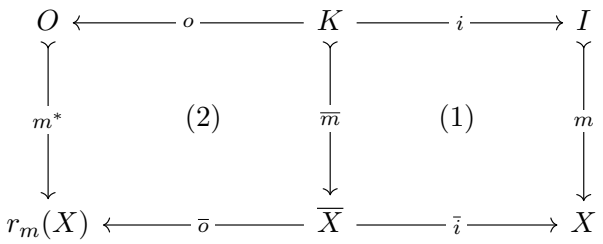
A rewriting rule is a span $O \xleftarrow{o} K \xrightarrow{i} I$. The idea is that I is the input and O the output.

All the following definitions come from [3], Definition 10.

Definition 11 (SqPO-admissible match)**.** *Given a rule $r = O \xleftarrow{o} K \xrightarrow{i} I$, its admissible matches into an object X are the regular monomorphisms $I \xrightarrow{m} X$. We note $M_r^{SqPO}(X)$ the class of such morphisms.*

Definition 12 (SqPO direct derivation)**.** *Given a rule $r = O \xleftarrow{o} K \xrightarrow{i} I$, an object X and an element m of $M_r^{SqPO}(X)$, a derivation of X with r along*

m is a diagram of the following shape, (1) being obtained as an FPC and (2) as a pushout.



It is also possible to compose rules. Admissible matches of rules are the different ways in which rules can be composed.

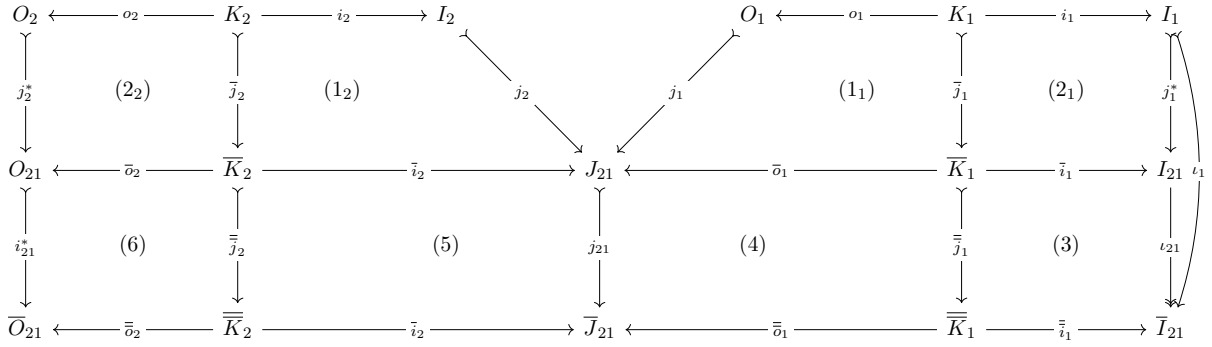
Definition 13 (SqPO-type admissible matches)**.** *Given two rules* $r_1 = O_1 \xleftarrow{o_1} K_1 \xrightarrow{i_1} I_1$ *and* $r_2 = O_2 \xleftarrow{o_2} K_2 \xrightarrow{i_2} I_2$*, we define the set of admissible matches:*

$$\begin{aligned}\mathcal{M}_{r_2}^{SqPO}(r_1) := \{(j_2, j_1, \overline{j}_1, \overline{o}_1, \overline{\overline{j}}_1; \overline{\overline{i}}_1, \iota_{21}) | (j_2, j_1) \in \Sigma_{\mathcal{M}}(I_2, O_1) \\ \wedge (\overline{j}_1, \overline{o}_1) \in \mathcal{P}(o_1, j_1) \\ \wedge (\overline{\overline{j}}_1, \overline{\overline{i}}_1, \iota_2 1) \in FPA(\overline{j}_1, i_1)\} / \sim\end{aligned}$$

See the diagram below for a clearer view of what the various morphisms are.

The set is quotiented by the relation $\sim$, which eliminates equivalent elements coming from multi-sums, multi-POCs and FPAs.

Definition 14 (SqPO-type rule composition)**.** *Given two rules* $r_1 = O_1 \xleftarrow{o_1} K_1 \xrightarrow{i_1} I_1$ *and* $r_2 = O_2 \xleftarrow{o_2} K_2 \xrightarrow{i_2} I_2$ *and an admissible match* $\mu \in \mathcal{M}_{r_2}^{SqPO}(r_1)$*, the composition is a diagram like the one below, where* (2_2)*, (6) and (4) are pushouts,* (1_1) *is a multi-POC,* (2_1) *and (3) are an FPA and* (1_2) *and* (5) *are FPCs.*



The new rule is obtained by span composition:

$$r_2 \overset{\mu}{\triangleleft} r_1 := (\overline{O}_{21} \xleftarrow{\bar{\bar{o}}_2} \overline{\overline{K}}_2 \xrightarrow{\bar{\bar{i}}_2} \overline{J}_{21}) \circ (\overline{J}_{21} \xleftarrow{\bar{\bar{o}}_1} \overline{\overline{K}}_1 \xrightarrow{\bar{\bar{i}}_1} \overline{I}_{21})$$

We thus take a pullback $\overline{K}_{21}$ of $\overline{\overline{K}}_2 \xrightarrow{\bar{\bar{i}}_2} \overline{J}_{21} \xleftarrow{\bar{\bar{o}}_1} \overline{\overline{K}}_1$ to obtain a new rule of the shape $\overline{O}_{21} \leftarrow \overline{K}_{21} \rightarrow \overline{I}_{21}$.

3 Formalization

The last diagram shows how complex things can become when studiying graph transformations, as this is only for the composition of two rules. This is why formalizing the theory in a proof assistant can be helpful.

3.1 Quasi-topoi

The definition of quasi-topoi presented earlier involves locally cartesian closed categories, which means it requires using slice categories, which means adding a layer of definitions.

However we can find an equivalent definition in [11]:

Lemma 3 ([11])**.** *A category is a quasi-topos iff:*

- *it has finite limits and colimits,*
- *it is cartesian closed,*
- *it has a partial map classifier.*

So we can just require a cartseian closed cateogry, but we need to have a partial map classifier, which is not much more complicated than a subobject classifier.

We first define stable systems of monics with a class having a type to represent monics, and a way to recover the morphisms from this type. We can then translate the requirements directly.

```
Class Stable_monics {C: Category} := {
    Stb_Monics (A B: C): Type;
    Stb_Monics_morph: forall [A B], Stb_Monics A B -> A ~> B;
    Is_ℳ {A B: C} (f: A ~> B) := ∃ f': Stb_Monics A B,
      Stb_Monics_morph f' = f;

    (** Elements of ℳ are monomorphisms *)
    M_monics: forall [A B: C] (m: A ~> B), Is_ℳ m -> Monic m;

    (** Direct translation of the definition *)
    M_isos: forall [A B: C] m, (∃ iso: A ≅ B, to iso = m)
      -> Is_ℳ m;
    M_comp: forall [A B C: C] (f: A ~> B) (g: B ~> C),
      Is_ℳ f -> Is_ℳ g -> Is_ℳ (g ∘ f);
    M_pb: forall [A B C D: C] [a: A ~> B] [b: B ~> D]
      [c: A ~> C] [d: C ~> D] (pb: Pullback a b c d),
      Is_ℳ b -> Is_ℳ c
  }.
```

We also need to define regular monomorphisms as the instance of stable system of monics to be used for quasi-topoi.

Equalizers are straightforward to define:

```
 Class Equalizer (f g: A ~> B) (e: E ~> A) := {
     Eq_comp: f ∘ e ≈ g ∘ e;

     Eq_prop: forall (E': C) (e': E' ~> A), f ∘ e' ≈ g ∘ e' ->
      ∃!~> ε: E' ~> E,
       e' ≈ e ∘ ε
   }.
```

And then regular monomorphisms, following the definition:

```
 Class Reg_mono {C: Category} [A B] (m: A ~> B) := {
    reg_prop: ∃ C (f g: B ~> C), Equalizer f g m
  }.
```

We can then define partial map classifiers:

```
Class Part_map_classifier {C: Category} {𝓜: Stable_monics}:= {
   class_T: C → C;
   class_η: Id ⇒ class_T;

   class_𝓜: forall X: C, (class_η X) ∈ 𝓜;
   class_pb: forall [X A B] (m: X ~> A) (f: X ~> B),
     m ∈ 𝓜 -> ∃!~> ϕ: A ~> class_T B,
       Pullback m ϕ f (class_η B)
 }.
```

And we can finally define quasi-topoi using the library [10] for limits, colimits and cartesian closeness:

```
Class Quasi_topos (C: Category) := {

  (** Limits and colimits *)
  Qt_limits: forall (J: Category) (F: J → C), Limit F;
  Qt_colimits: forall (J: Category) (F: J → C), Colimit F;

  (** C is cartesian *)
  Qt_cart: Cartesian;
  (** closed *)
  Qt_cart_closed: Closed;

  (** and it has a partial map classifier *)
  Qt_part_map_cl: @Part_map_classifier _ (Reg_mono_𝓜)
}.
```

Where `Reg_mono_𝓜` is the function that gives the stable system of monics made of regular monomorphisms.

3.2 Basic structures

3.2.1 Pullbacks

In John Wigley's [10], pullbacks are defined in the following way:

```
Record Pullback {C: Category} {B C D: C}
        (b: B ~> D) (d: C ~> D) := {
```

```
    Pull: C;
    pullback_fst: Pull ~> B;
    pullback_snd: Pull ~> C;

    pullback_commutes: b ∘ pullback_fst
                       ≈ d ∘ pullback_snd;
    ump_pullbacks : ∀ A' (a' : A' ~> B) (c' : A' ~> C),
        b ∘ a' ≈ d ∘ c'
        -> ∃! u : A' ~> Pull, pullback_fst ∘ u ≈ a'
                              ∧ pullback_snd ∘ u ≈ c'
}.
```

The `pullback_commutes` and `ump_pullbacks` fields are a direct translation of the commutativity and of the universal property of pullbacks, respectively. It also makes the choice to provide the pullback object and the two morphisms. While this can be useful in some cases, we also need a way to express that a given square is a pullback. However doing so with the previous definition requires some typecasting with equalities, which is very unpractical.

We can solve this problem by directly defining pulbacks as a property of a square, without changing much of the definition:

```
Record Pullback {C: Category} [A B C D: C]
        (a: A ~> B) (b: B ~> D) (c: A ~> C) (d: C ~> D) := {
    pullback_commutes: b ∘ a ≈ d ∘ c;
    ump_pullbacks : ∀ A' (a' : A' ~> B) (c' : A' ~> C),
        b ∘ a' ≈ d ∘ c'
        -> ∃! u : A' ~> A, a ∘ u ≈ a' ∧ c ∘ u ≈ c'
}.
```

Defining a class giving a pullback when given a cospan is very simple using existential quantifiers.

Once this definition is established, it remains to prove some key results about pullbacks, such as their unicity up to a unique isomorphism. This can be directly proven in Coq without much difficulty. However from a theoretical point of view, this can be proven much more elegantly with the following lemma.

Lemma 4. *Given a category* $\mathbf{C}$*, three objects* B*,* C *and* D *and a cospan* $B \xrightarrow{b} D \xleftarrow{d} C$*, a pullback of* b *along* d *is a terminal objects in the category*

of spans $B \xleftarrow{f} X \xrightarrow{g} C$ *making the square commute. The morphisms in this category being the morphisms in* $\mathbf{C}$ *between the middle objects of the spans such that the obtained diagram commutes.*

Proof. This correspond to the definition of pullbacks as the limit of a diagram.

Since we now that terminal objects are unique up to a unique isomorphism, it is almost immediate to show the unicity of pullbacks. It would be interesting to add more modularity in the formalisation, so as to be able to prove the unicity of terminal objects once, and then use it to prove the unicity of pullbacks, and of any other structure that is a terminal object in some category.

We can define a category of spans `Span_cat`, given two objects. We then define, given a cospan, a subcategory `Square_cat` containing only the spans making the square commute. This definition can be done interactively for the most part in Coq.

We then define what a terminal object is, in the same way as in John Wigley's library [10], but changing it from giving a terminal object to being a property of objects. We can then define pullbacks in a third way:

```
Class Pullback {C: Category} [A B C D: C]
        (a: A ~> B) (b: B ~> D) (c: A ~> C) (d: C ~> D):= {
  pb_comm: b ∘ a ≈ d ∘ c;
  pb_term: @Terminal (Square_cat B C
    ({|sp_l := op b; sp_r := op d|}))
    ({|sp_obj := A; sp_l := a; sp_r := c |}; pb_comm)
}.
```

I have proven the equivalence of this definition with the previous one. The interest of doing this is twofold: first, to check the correctness of the definition, second, because this definition is generally not the best to use in a proof, this proof makes it possible to build a pullback using the usual definition, and to decompose them in the same way.

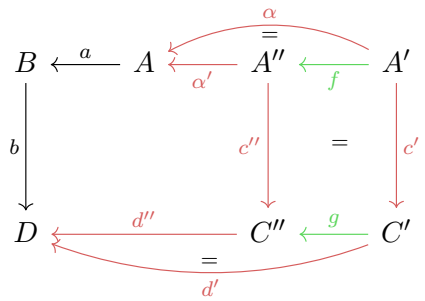
Once the unicity of terminal objects is proven, proving the unicity of pullbacks is simply a matter of unfolding definitions. It still takes a number of lines, but is quite straigthforward.

Pushouts are defined as pullbacks in the opposite category, which allows for the transfer of results from one to the other.

3.2.2 Final Pullback Complements

The definition could easily be directly translated into Coq in the same way as the original definition of pullbacks. However we can prove that FPCs are specific terminal object in a particular category, to get the same benefits as for pullbacks.

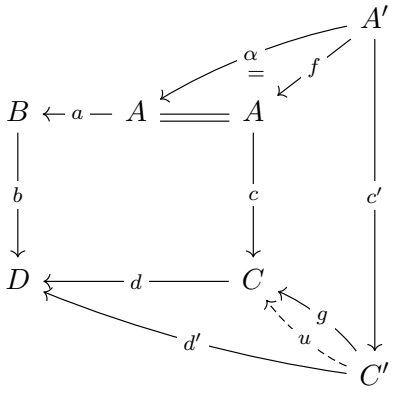
Definition 15. *Given three objects A, B, and D in a category $\mathbf{C}$ and a pair of composable morphisms $a : A \to B$ and $b : B \to D$, we take the following category of "candidates FPC" $cFPC$:*
Objects: quintuplets made of two objects A' and C' in $\mathbf{C}$ and three morphisms $\alpha : A' \to A$, $c' : A' \to C'$ and $d' : C' \to D$, such that the square $A' \xrightarrow{a \circ \alpha} B \xrightarrow{b} D \xleftarrow{d'} C' \xleftarrow{c'} A'$ is a pullback.
Morphisms: given two objects of $cFPC$, $A \xleftarrow{\alpha} A' \xrightarrow{c'} C' \xrightarrow{d'} D$ and $A \xleftarrow{\alpha'} A'' \xrightarrow{c''} C'' \xrightarrow{d''} D$, morphisms between the two are pairs of morphisms in $\mathbf{C}$, $A' \xrightarrow{f} A''$ and $C' \xrightarrow{g} C''$ such that $\alpha' \circ f = \alpha$, $c'' \circ f = g \circ c'$ and $d'' \circ g = d'$.
This is illustrated in the following diagram.



Theorem 1. *The FPCs of a and b are the terminal objects in $cFPC$ such that α is an identity morphism.*

Proof. The details of the proof are formalized in Coq. I present here the basic ideas.
First, given an FPC $A \xrightarrow{c} C \xrightarrow{d} D$ of a and b, we take the object $X := A \xrightarrow{id_A} A \xrightarrow{c} C \xrightarrow{d} D$ of $cFPC$ (it forms a pullback by definition of an FPC). We now prove that it is a terminal object: given another object $Y := A \xleftarrow{\alpha} A' \xrightarrow{c'} C' \xrightarrow{d'} D$, we have a unique morphism $C' \xrightarrow{u} C$ by the property of FPCs, and we already have $A' \xrightarrow{\alpha} A$, which gives us a morphism (α, u) between Y and X. Such a morphism is unique, if we have another morphism (f, g) from Y to X then $g = u$ because of the unicity of u, and

we have $id_A \circ f = \alpha$, so $f = \alpha$.



We now suppose we have a terminal object $X := A \overset{id_A}{=} A \overset{c}{\to} C \overset{d}{\to} D$. It is a pullback by definition. Given a pullback $A' \overset{a'}{\to} B \overset{b}{\to} D \overset{d'}{\leftarrow} C' \overset{c'}{\leftarrow} A'$ with a morphism $A' \overset{\alpha}{\to} A$ such that $a' = a \circ \alpha$, we define the object $A \overset{\alpha}{\leftarrow} A' \overset{c'}{\to} C' \overset{d'}{\to} D$. Since X is a terminal object, we have a unique pair of morphisms $A' \overset{u}{\to} A$ and $C' \overset{u'}{\to} C$ such that the square and triangles commute. We then directly have that $u = \alpha$ and thus that u' satisfies the commutation requirements of the definition of FPCs, which gives us that $A \overset{c}{\to} C \overset{d}{\to} D$ is an FPC of $A \overset{a}{\to} B \overset{b}{\to} D$. □

Once again, proving this equivalence in Coq is useful to check the proof, but also to make the usual definition of FPCs easily available.

We can remark that instead of asking for α to be the identity, askiing for it to be an isomorphism is equivalent.

Remark. *Asking for α to be an identity in the terminal object is equivalent to asking for α to be an isormorphism in a terminal object.*

Proof. If we have a terminal object $A \overset{\alpha}{\leftarrow} A' \overset{c'}{\to} C' \overset{d'}{\to} D$ where α is an isomorphism, then $A \overset{id_A}{=} A \xrightarrow{c' \circ \alpha^{-1}} C' \overset{d'}{\to} D$ is also an object ($A \overset{a}{\to} B \overset{b}{\to} D \overset{d'}{\leftarrow} C' \xleftarrow{c' \circ \alpha^{-1}} A$ is a pullback because $A' \xrightarrow{a \circ \alpha} B \overset{b}{\to} D \overset{d'}{\leftarrow} C' \overset{c'}{\leftarrow} A''$ is, this is a simple result on pullbacks which is formalized in the Coq file for pullback lemmas). We have a morphism from the first object to the second: $(\alpha, id_{C'})$, and since the first object is terminal, this means the second also is, by unicity of terminal objects up to isomorphism. □

The implementation of FPCs using this result is in the file `FPC`. Like for pullbacks, I have defined the category interactively using the `Program` library. FPCs are then defined in a straightforward way:

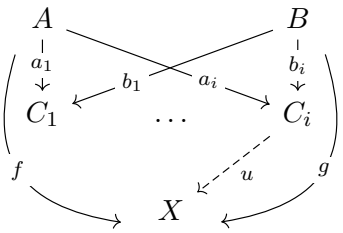
```
Class FPC {C: Category} [A B D: C] (a: A ~> B) (b: B ~> D)
   [C: C] (c: A ~> C) (d: C ~> D) := {
   FPC_pb: Pullback a b c d;
   FPC_term: Terminal (fpc_cat_id_obj C c d FPC_pb);
 }.
```

With `fpc_cat_id_obj` being the function returning an object of the category with an identity as α. This is once again a property of a pair of morphisms to be an FPC, for the same reason as for pullbacks.

3.2.3 Multi-sums

A problem with the previous definition of multi-sums is that a quotient has to be taken to remove equivalent objects from the family. One way to avoid doing that is to require the unicity of the object and the morphism, like in this definition from [6].

Definition 16 (Multi-sum [6])**.** *Given two objects A and B in a category $\mathbf{C}$, a multi-sum of A and B is a family of cospans $(A \xrightarrow{a_i} C_i \xleftarrow{b_i} B)_{i\in I}$ such that for any cospan $A \xrightarrow{f} X \xleftarrow{g} B$, there is a unique pair (i, u) with $i \in I$ and $C_i \xrightarrow{u} X$ such that $u \circ a_i = f$ and $u \circ b_i = g$.*



This can easily be linked to multi-inital families, which are a similar extension of initial objects:

Definition 17 (Multi-initial family [6])**.** *A multi-initial family in a category $\mathbf{C}$ is a family of objects $(X_i)_{i\in I}$ such that for any object Y there exists a unique pair (i, u) with $i \in I$ and $X_i \xrightarrow{u} Y$*

A multi-sum of A and B in $\mathbf{C}$ is then a multi-initial family in the category of cospans with domain A and B, equipped with the morphisms of $\mathbf{C}$ making the diagrams commute. In the case of graph rewriting [3], the definition involves regular monomorphisms, however it is exactly the same as the one presented here, on a restricted setting.

Multi-initial objects and multi-sums are the straightforward to define:

```
Definition Multi_initial {C: Category} I (i: I -> C) :=
 forall A: C, ∃! j: I, i j !~> A
   ∧ forall j': I, i j' ~> A -> j' = j.

Definition Multi_sum {C: Category} (A B: C) :=
   @Multi_initial (Cospan_cat A B).
```

Where `Cospan_cat` is the previously defined category of spans, on the opposite category.

3.2.4 Multi-POCs and FPAs

I have not defined Multi-POCs nor FPAs yet. Like FPCs and multi-sums, the idea would be to find way to define them in a way that allows for the inheritance of results, and that would also avoid the need for a quotient when defining admissibility of matches of rules For example, Multi-POCs could probably be defined as multi-initial objects in categories of pushout complements.

3.3 Formalized results

In addition to these definitions, I have formalized a number of auxiliary lemmas, mostly from Annex A of [3].

I have unfortunately not been able to reach the concurrency theorems (section 4 of [3]) since I have yet to formalize the semantics of sesqui-pushout and double-pushout graph rewriting.

4 Conclusion

Most of the conclusions and future work has been discussed in the first section.

However, in more details, implementing multi-POCs and FPAs are an obvious next step, since they are the two missing constructions for the rewriting semantics.

All definitions have also not been given as much attention as FPCs and pullbacks, there could be some improvements for the definition of quasi-topoi for instance, or in the underlying definitions such as limits and cartesian closeness.

References

[1] Jiri Adamek, Horst Herrlich, and George Strecker. "Abstract and Concrete Categories - The Joy of Cats". In: *Reprints in Theory and Applications of Categories* 17 (2006), pp. 1–507.

[2] Benedikt Ahrens, Chris Kapulkin, and Michael Shulman. "Univalent Categories and the Rezk Completion". In: *Mathematical Structures in Computer Science* 25.5 (June 2015), pp. 1010–1039. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129514000486. arXiv: 1303.0584 [math]. URL: http://arxiv.org/abs/1303.0584.

[3] Nicolas Behr, Russ Harmer, and Jean Krivine. "Concurrency Theorems for Non-linear Rewriting Theories". In: *Graph Transformation*. Ed. by Fabio Gadducci and Timo Kehrer. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 3–21. ISBN: 978-3-030-78946-6. DOI: 10.1007/978-3-030-78946-6_1.

[4] Nicolas Behr, Russ Harmer, and Jean Krivine. *Fundamentals of Compositional Rewriting Theory*. Apr. 14, 2022. DOI: 10.48550/arXiv.2204.07175. arXiv: 2204.07175 [cs]. URL: http://arxiv.org/abs/2204.07175.

[5] Andrea Corradini et al. "Sesqui-Pushout Rewriting". In: *Graph Transformations*. Ed. by Andrea Corradini et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 30–45. ISBN: 978-3-540-38872-2. DOI: 10.1007/11841883_4.

[6] Yves Diers. *Familles Universelles de Morphismes*. Publications Internes de l'U.E.R. de Mathématiques Pures et Appliquées. Lille: Université des Sciences et Techniques de Lille I, 1978. 19 p.

[7] *Formalizing Category Theory in Agda | Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. URL: https://dl.acm.org/doi/abs/10.1145/3437992.3439922.

[8] *HoTT*. Homotopy Type Theory. URL: https://github.com/HoTT/HoTT (visited on 08/20/2022).

[9] Grzegorz Rozenberg. “Handbook of Graph Grammars and Computing by Graph Transformation”. In: (Jan. 1, 1997).

[10] John Wiegley. *Category Theory in Coq*. URL: `https://github.com/jwiegley/category-theory` (visited on 08/20/2022).

[11] Oswald Wyler. “Chapter 1: Basic Properties”. In: *Lecture Notes on Topoi and Quasitopoi*. World Scientific. ISBN: 978-981-02-0153-1.

The Coq code can be found in this repository: `https://gitlab.com/SamuelArsac/m2-internship`, along with some documentation: `https://samuelarsac.gitlab.io/m2-internship`.