Formalizing Geometrical Proofs

Antoine Allioux

Oriel College

University of Oxford

A thesis submitted for the degree of

MSc in Computer Science

Trinity 2015

Acknowledgements

First and foremost I would like to thank my supervisor, Doctor Jamie Vicary, for the time he dedicated to me. I am grateful to him for having given me a first insight into research.
I also wish to thank the professors of the courses I took this year, which I greatly enjoyed.
I would like to thank my friends for the good times we had as well as for the enriching talks.
Finally I would like to thank my family for their support and without which I would not have been able to do this degree.

Abstract

In this thesis, we propose a diagrammatic representation of the superposition rule and we outline some approaches to implement it in a graphical proof-assistant.

The superposition rule is a consequence of the enrichment of a category over commutative monoids. This rule allows to sum morphisms from a same hom-set. This notion, while basic, has not been incorporated into the framework of string diagrams yet. We introduce a graphical depiction of this notion then we study extensively the interactions between the superposition rule, the monoidal structure and biproducts. In particular, we show how elegant a proof can be by proving in a purely diagrammatic way that in a monoidal category with biproducts and under a certain condition, the tensor product distributes over the arising superposition rule.

Finally we outline different approaches to implement the superposition rule in Globular, a graphical proof-assistant handling higer-categorical diagrams.

Contents

Chapter 1

Introduction

1.1 Background

Mathematical reasoning is a constructive human activity according to Brouwer, and as such proofs should reflect the steps leading to their conclusion. A well-suited language to describe proofs in order to convey this notion should therefore meet two criteria: it should make clear how from one step we reach the next one and it should make easier the understanding of each step.

With the increase in complexity of mathematics, contemporary mathematicians like to hide low-level details to focus on more complex concepts like a computer scientist would do using a high-level programming language instead of writing pure assembly code.

This is the basis of geometrical proofs which use the space (the plane if we only take into consideration two dimensions) to absorb in an intuitive fashion underlying mathematical details of proofs to focus on higher-level concepts.

We proceed by mapping mathematical notions to specific geometrical notations or transformations. Then, we just have to manipulate a diagram which model the system we are working on using the liberties which have been established in order to reason about a particular problem.

Geometrical proofs have already been extensively used in areas such as topology, network theory or quantum mechanics to name a few.

In this thesis we shall focus on string diagrams which have been used with success to express operations in monoidal categories. String diagrams have been developed since the 1970s by mathematicians such as André Joyal, Ross Street and Roger Penrose.

We call them string diagrams as objects are traditionally represented by strings, and morphisms between tensor products are represented by nodes whose inputs and outputs are strings. Then, we attribute a geometrical property to each additional structure on monoidal categories. The aim is to have a full and faithful mapping between the definition of our monoidal category and its geometrical properties. Therefore, when we encounter this notation in a diagram describing a complicated system, we understand at first sight what is happening without losing any formalism. Indeed, all the algebraic details are embedded in the diagram.

Another interesting property of string diagrams over the more classical equational notation is the fact they embed type information which allows the reader to quickly check that the typing of the performed operations is correct.

One could object this type information is retained in pasting diagrams, an ubiquitous notation when it comes to proofs involving category theory. But, contrary to the equational notation, pasting diagrams poorly reflect the different steps of the reasoning leading to the result.

String diagrams actually combine the best of both worlds by reflecting the different steps of the reasoning as well as the type of the operations.

However, the use of geometrical proofs is still young and informal.
We still lack a geometrical representation of some mathematical concepts which prevents us to use geometrical notations exclusively for the development of some proofs.

We believe that easing the way we think about mathematics is of prime importance, we want to make the exchange of ideas flows better. Indeed, contemporary mathematicians tend to work collaboratively and time spent understanding the work of colleagues should be minimized.

In addition, we notice contemporary mathematicians have a growing interest in formalizing their theory using computerized proof assistants which is of great interest when working collaboratively. Indeed, work can be organized in a modular way by splitting tasks and developing libraries of formalized mathematical definitions, theorems and proofs which can be reused in other projects. In fact, it is safe to say software development methods have inspired mathematicians. One of the most successful recent attempt being the formalization of homotopy type theory in Coq[1] by a team of researchers from the Institute for Advanced Studies lead by Vladimir Voevodsky.

[1] `https://github.com/HoTT/HoTT`

1.2 Outline

In this thesis we shall focus on the superposition rule which has no geometrical representation yet in spite of the fact it is a basic feature of categories enriched over commutative monoids. A category possessing this property allows one to sum maps belonging to the same hom-set. This lack of diagrammatic depiction of the superposition rule prevents us to provide a fully geometrical proof for theorems involving this notion. Therefore we shall propose a diagrammatic depiction of the superposition rule which fits elegantly in the string diagrams framework.

Once this rule formalized, we will establish interesting connections between the superposition rule and biproducts. Notably that biproducts entail the existence of a unique superposition rule and that in a presence of a superposition rule coproducts are biproducts.

Then we shall examine some conditions for a superposition rule to arise in monoidal categories. Indeed, monoidal categories are our categories of predilection as string diagrams precisely aim at depicting these categories.

Next we shall study the interactions between the superposition rule, the monoidal structure and biproducts. This will lead us to prove that the tensor product distributes over a superposition of maps in the presence of biproducts under a certain condition.

Finally we will propose an implementation of the superposition rule in Globular, a proof-assistant aiming at handling higher-dimensional categorical diagrams. To this effect we will describe two different approaches which vary in expressiveness and in difficulty of implementation.

Chapter 2

String diagrams

String diagrams are a graphical calculus allowing the graphical representation of monoidal categories. The graphical calculus is an alternative to the algebraic notation allowing to reason about mathematical formulae using a visual representation. Its graphical nature allows to reason about problems in a more intuitive fashion. Indeed, in the case of string diagrams we use the two dimensions of the plane and its corresponding topological properties. Intuitively, this is to be opposed to the only dimension provided by the algebraic notation.

In this chapter we quickly remind the reader of the most common constructions regarding string diagrams. We also present the diagrammatic depictions of the product and of the coproduct introduced by Marsden [5].

2.1 Monoidal Categories

As string diagrams are a graphical calculus targeting monoidal categories, we shall begin by introducing them before describing how to depict them.

Monoidal categories are categories equipped with a monoid structure in $\mathscr{C}at$, the category of categories, and some additional properties ensuring the coherence. Without these isomorphisms guaranteeing the coherence we would restrict ourselves to strict monoidal categories.

Definition 2.1.1 (Monoidal Category)**.** A monoidal category $\mathscr{C}$ is a category equipped with

- a unit object I
- a tensor product functor $\otimes : \mathscr{C} \times \mathscr{C} \Rightarrow \mathscr{C}$
- a natural isomorphism called the associator $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$

- a natural isomorphism called left unitor $\lambda_a : I \otimes A \to A$
- a natural isomorphism called right unitor $\rho_a : A \otimes I \to A$

such that the following diagrams commute:

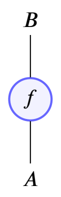
$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[rr, "\alpha_{A,I,B}"] \arrow[dr, "\rho_A \otimes \mathrm{id}_B"'] & & A \otimes (I \otimes B) \arrow[dl, "\mathrm{id}_A \otimes \lambda_B"] \\
& A \otimes B &
\end{tikzcd} \tag{2.1}$$

$$\begin{tikzcd}
(A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B\otimes C,D}"] & & A \otimes ((B \otimes C) \otimes D) \arrow[d, "\mathrm{id}_A \otimes \alpha_{B,C,D}"] \\
((A \otimes B) \otimes C) \otimes D \arrow[u, "\alpha_{A,B,C} \otimes \mathrm{id}_D"] \arrow[dr, "\alpha_{A\otimes B,C,D}"'] & & A \otimes (B \otimes (C \otimes D)) \\
& (A \otimes B) \otimes (C \otimes D) \arrow[ur, "\alpha_{A,B,C\otimes D}"'] &
\end{tikzcd} \tag{2.2}$$

We will introduce the graphical notation of each component defining a monoidal category.

Objects in monoidal categories are depicted by strings, and morphisms between objects are depicted by nodes possessing source and target strings.

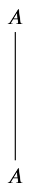
As a result, maps $f : A \to B$ will be depicted this way:



(2.3)

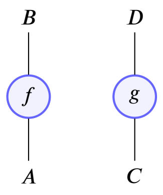
We read diagrams from the bottom to the top.

Identities are a special case, indeed we do not distinguish identities from their respective object. This lead to the following representation for id_A:



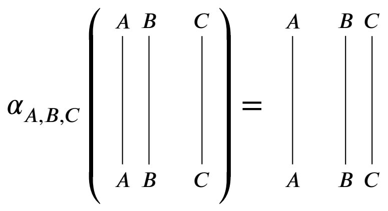
(2.4)

The tensor product of two objects or of two morphisms in monoidal categories is depicted by placing these objects or these morphisms side by side. For example, the tensor product of two morphisms $f : A \to B$ and $g : C \to D$, namely $f \otimes g : A \otimes C \to B \otimes D$ is depicted this way:



(2.5)

As for the associator, our strings behave such that the underlying bracketing does not matter. Indeed, the associator acts on tensor products the following way:

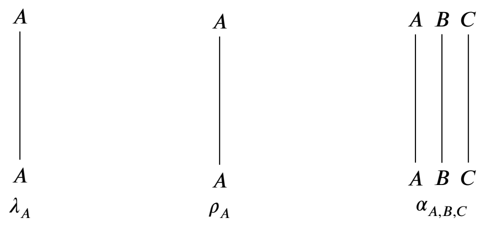


(2.6)

But we do not distinguish the source diagram from the resulting diagram.

Concerning the depiction of the unit object I, we choose to write it as an invisible string so it does not appear in a diagram. As a consequence, for every objects of $\mathscr{C}$ left and right unitors have the same depiction as the identity map.

To summarize, we depict the unitors and the associator the following way:



2.2 Dual object

As we shall see, several interesting results arise from the interaction of dual objects with the monoidal structure.

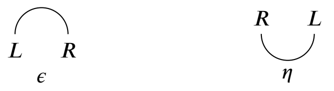
In a monoidal category, the duality of an object corresponds to adjointness. Indeed, a (left or right) dual object in a monoidal category $\mathscr{C}$ is equivalent to the (left or right) adjoint of this object seen as a 1-cell in the delooping bicategory of $\mathscr{C}$: $\mathscr{BC}$.

Definition 2.2.1 (Dual object)**.** In a monoidal category, we say that an object R has a left dual L when there exist two morphisms $\eta : I \to R \otimes L$ and $\epsilon : L \otimes R \to I$ called the evaluation map and the coevaluation map such that these two diagrams commute:

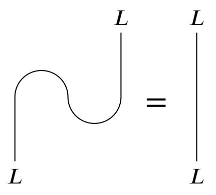
$$\begin{tikzcd}
L \arrow[r, "\rho_L^{-1}"] \arrow[d, "\mathrm{id}_L"'] & L \otimes I \arrow[r, "\mathrm{id}_L \otimes \eta"] & L \otimes (R \otimes L) \arrow[d, "\alpha_{L,R,L}^{-1}"] \\
L & I \otimes L \arrow[l, "\lambda_L"] & (L \otimes R) \otimes L \arrow[l, "\epsilon \otimes \mathrm{id}_L"]
\end{tikzcd} \tag{2.7}$$

$$\begin{tikzcd}
R \arrow[r, "\lambda_R^{-1}"] \arrow[d, "\mathrm{id}_R"'] & I \otimes R \arrow[r, "\eta \otimes \mathrm{id}_R"] & (R \otimes L) \otimes R \arrow[d, "\alpha_{R,L,R}"] \\
R & R \otimes I \arrow[l, "\rho_R"] & R \otimes (L \otimes R) \arrow[l, "\mathrm{id}_R \otimes \epsilon"]
\end{tikzcd} \tag{2.8}$$

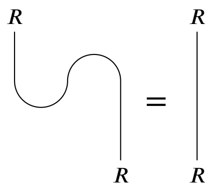
In the framework of string diagrams, we depict the evaluation map and the coevaluation map using the cap and cup notation:



Using this diagrammatic notation, saying that R has a left dual L boils down to having these two equations:



(2.9)

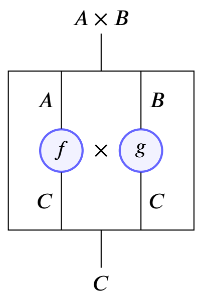
 (2.10)

2.3 Products and coproducts

In order to describe products and coproducts using string diagrams, we borrow the box notation introduced by Marsden [5] in the context of 2-categories and we adapt it to monoidal categories which are just bicategories (a particular notion of weak 2-categories) with one object.

2.3.1 Product

In a category with products, for all morphisms $f : C \to A, g : C \to B$, we represent the morphism $\langle f, g\rangle : C \to A \times B$ by the following diagram:

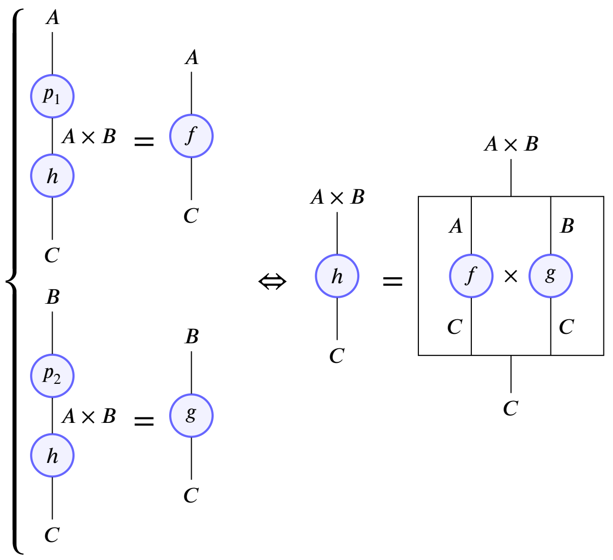


The box reveals the two diagrams corresponding to the two components of the product. The symbol × inside the box will allow us to distinguish a product box from a coproduct box.

We can now characterize the product in a purely diagrammatic way.
In a category with products, for all objects A and B there are morphisms

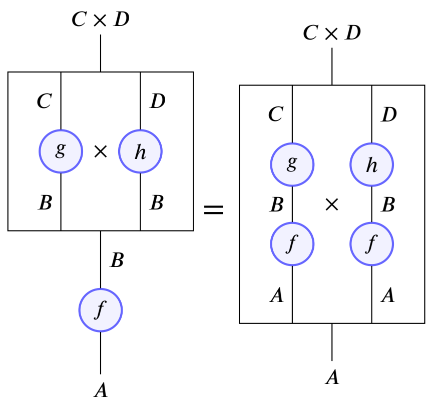
$$p_1 : A \times B \to A$$
$$p_2 : A \times B \to B$$

such that for all morphisms $f : C \to A, g : C \to B$,



(2.11)

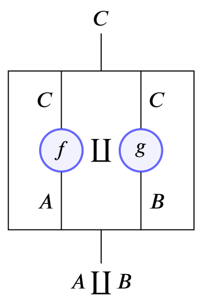
We have the following calculation rule:
$\forall f : A \to B, g : B \to C, h : B \to D,$



(2.12)

2.3.2 Coproduct

Dually, we characterize the coproduct.
In a category with coproducts, for all morphisms $f : A \to C, g : B \to C$, we represent the morphism $[f, g] : A \coprod B \to C$ by the following diagram:

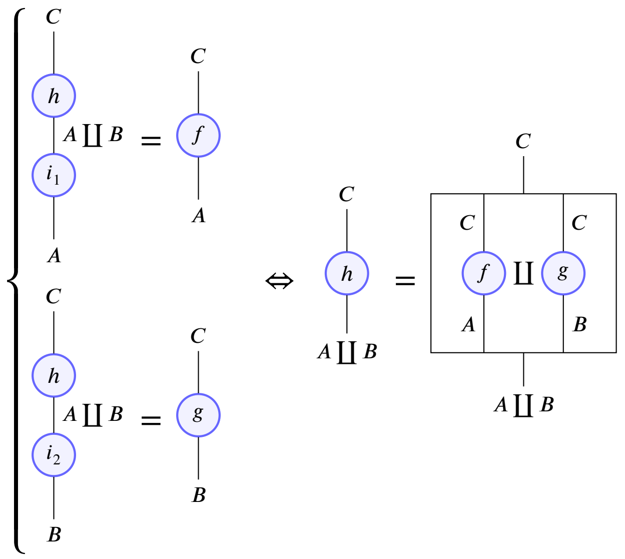


The box reveals the two diagrams corresponding to the two components of the coproduct.

This allows us to give the following diagrammatic definition of the coproduct. In a category with coproducts, for all objects A and B there are morphisms

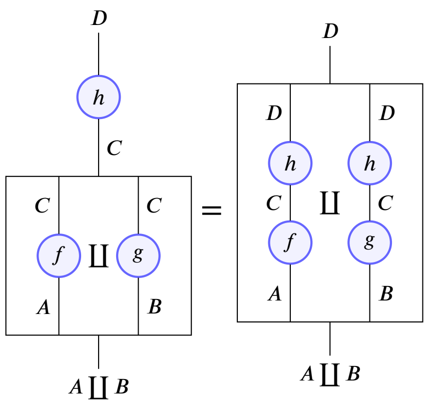
$$i_1 : A \to A \coprod B$$
$$i_2 : B \to A \coprod B$$

such that for all morphisms $f : A \to C, g : B \to C$,



(2.13)

We have the following calculation rule:
$\forall f : A \to C, g : B \to C, h : C \to D,$



(2.14)

Chapter 3

The superposition rule

The superposition rule arises in categories enriched over $\mathscr{CM}on$, the category of commutative monoids. This is a rule of importance as this allows the sum of morphisms of a same hom-set and this provides some linear properties to categories equipped with this rule.

In the first place we shall give a diagrammatic depiction of the superposition rule. This will be the main contribution of this thesis.

Once this diagrammatic depiction formalized, we shall establish interesting connections between the existence of a superposition rule and biproducts in a category. Notably that the existence of biproducts implies the existence of a unique superposition rule and that in a category with a superposition rule, coproducts are biproducts.

Then, as monoidal categories are our main subject of interest, we shall study some conditions for a monoidal category to be equipped with a superposition rule.

Finally we shall expose some interesting properties resulting from the interaction of the monoidal structure with the superposition rule. Namely that the tensor product distributes over the a superposition of maps when biproducts exist under a certain condition.

3.1 A diagrammatic depiction of the superposition rule

We shall start by giving the algebraic definition of the superposition rule as mentioned in [7].

Definition 3.1.1 (Superposition Rule)**.** We say that a category $\mathscr{C}$ is equipped with a superposition rule if for all arrows $f, g, h : A \to B$ in $\mathscr{C}$, the operation $+$:

$\mathscr{C}(A,B) \times \mathscr{C}(A,B) \to \mathscr{C}(A,B)$ is defined such that the following properties hold:

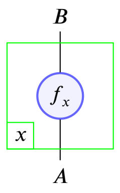
- *Commutativity*
$$f + g = g + f$$
- *Associativity*
$$(f+g)+h = f+(g+h)$$
- *Units*
There is a morphism $u_{A,B} : A \to B$ such that for all $f : A \to B$ we have $f + u_{A,B} = f$
- *Distributivity of the composition over the sum*
$\forall f, g : A \to B$, $\forall h : B \to C$, $\forall i : C \to A$
$$h \circ (f+g) = (h \circ f) + (h \circ g)$$
$$(f+g) \circ i = (f \circ i) + (g \circ i)$$
- *Compatibility of units with composition*
$$u_{B,C} \circ u_{A,B} = u_{A,C}$$

Despite being a basic feature of categories enriched over commutative monoids, no diagrammatic depiction has been proposed in order to enrich our string diagrams framework. This thesis aims at proposing a graphical representation of this rule which would incorporate consistently in the existing framework.

We shall now introduce a diagrammatic notation for the superposition rule allowing us to work on proofs involving this notion in a fully diagrammatic way.

Definition 3.1.2 (Diagrammatic depiction of the superposition rule)**.**

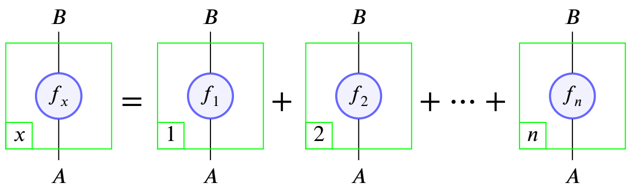
We graphically represent a superposition of maps indexed over x, $f_x : A \to B$ by the following diagram:



We can see this diagram as a diagram whose green box contains n diagrams indexed by x with $x \in [\![1, n]\!]$. The context will often make explicit the range of x. This representation is particularly well-suited to be displayed in the graphical user interface of a proof-assistant. We could imagine the user cycling through the different diagrams which are part of the superposition using his mouse.

In the present case, as the medium of this thesis is unlikely to be some sort of e-paper, we will mainly restrict ourselves to superpositions of diagrams having the same diagrammatic structure. In the case we have to explicitly expose the content of the superpositions, we will adopt the following notation:

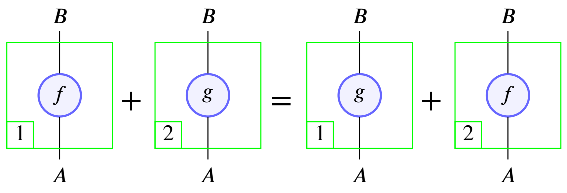
For a superposition of maps f_x indexed over $x \in [\![1, n]\!]$ we write:



The structure of the diagrams inside the boxes need not be the same but their boundary has to be as they represent a collection of morphisms from a same hom-set.

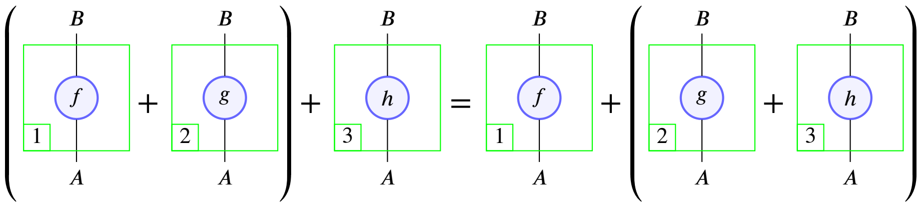
The properties of the superposition rule are the following:

- *Commutativity*
  $\forall f, g : A \to B$



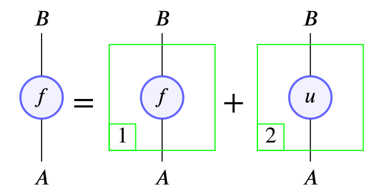
(3.1)

- *Associativity*
  $\forall f, g, h : A \to B$



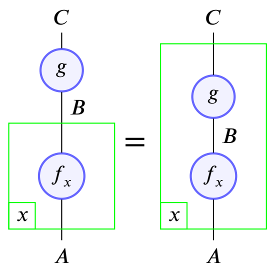
(3.2)

- *Units*
  $\forall A, B \in \mathcal{O}bj(\mathscr{C}), \exists u : A \to B, \forall f : A \to B$:

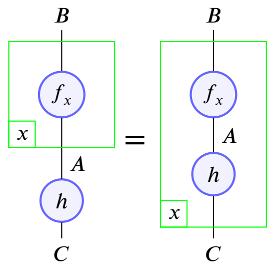


(3.3)

- *Distributivity of the composition over the superposition*
  For all families of morphisms indexed over x, $f_x : A \to B$, $\forall g : B \to C, \forall h : C \to A$,



(3.4)



(3.5)

It turns out that categories with biproducts—also called semiadditive categories—are equipped with a superposition rule.

3.2 Biproducts

It is interesting to note biproducts and superposition rules are connected. In particular we will show that in a category, the existence of biproducts implies the existence of a unique superposition rule. Conversely, in a category with a superposition rule, we shall show that coproducts and products are biproducts.

3.2.1 First definition

There are several equivalent definitions of a category with finite biproducts. We will first study a definition which does not involve the superposition rule as a prerequisite.

Then we shall show the existence of biproducts implies the existence of a unique superposition rule.

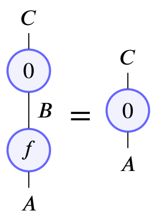
Finally we will give a simpler definition of the biproduct which is more suitable for a diagrammatic depiction using the superposition rule and we will show the two definitions are equivalent.

As our definition of biproducts involve the notion of 0-object, we shall start by introducing this notion.

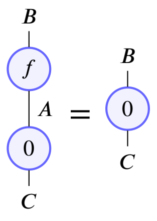
Definition 3.2.1 (0-object)**.** A 0-object is an object which is both initial and terminal.

A consequence of the existence of a 0-object in a category $\mathscr{C}$ is that each hom-set $\mathscr{C}(A, B)$ possesses a unique map factorizing through the 0-object. We write it $0_{A,B}$ and call it a 0-morphism.

In particular, the composition of any morphism with a 0-morphism factorizes through the 0-object and is therefore a 0-morphism. Therefore we have the following equalities for all $f : A \to B$:



(3.6)



(3.7)

We can now introduce the definition of biproducts. This definition appears in [4, p. 198] and is the one given on nLab.

Definition 3.2.2 (Biproduct (definition 1))**.** A category $\mathscr{C}$ with a 0-object, finite products and finite coproducts is said to have biproducts if for all objects A_1, A_2, there is a unique isomorphism $\Psi : A_1 \coprod A_2 \to A_1 \times A_2$ such that:

$$A_x \xrightarrow{i_x} A_1 \coprod A_2 \xrightarrow{\Psi} A_1 \times A_2 \xrightarrow{p_y} A_y = \begin{cases} \mathrm{id}_{A_x} & \text{if } x = y \\ 0_{A_x, A_y} & \text{otherwise} \end{cases} \tag{3.8}$$

$A_1 \coprod A_2$ and $A_1 \times A_2$ being isomorphic in a unique way, we call them the biproduct of A_1 and A_2 and refer to it using the following notation: $A_1 \oplus A_2$.

This biproduct is equipped with the following maps:

$$p_x : A_1 \oplus A_2 \to A_x$$
$$i_x : A_x \to A_1 \oplus A_2$$

such that

$$p_x \circ i_y = \begin{cases} \mathrm{id}_{A_x} & \text{if } x = y \\ 0_{A_x, A_y} & \text{otherwise} \end{cases} \tag{3.9}$$

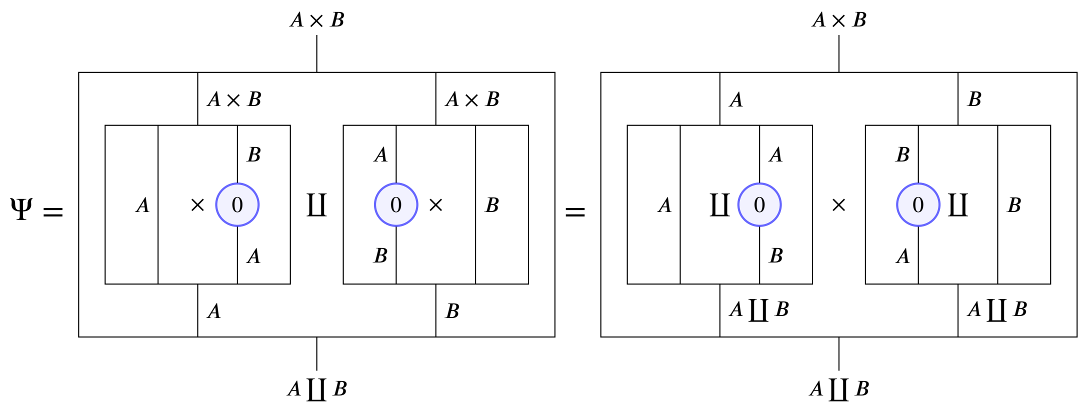
Note that the maps p_x and i_x are different from the one previously defined.
We will often refer to the canonical projection and coprojection maps by the name p_x and i_x without specifying the objects on which they act. We let the reader infer this information thanks to the context.

3.2.2 Biproducts entail the existence of a unique superposition rule

In this section we will show that a unique superposition rule arises from the existence of biproducts.

Let $\mathscr{C}$ be a category with biproducts and a 0-object. We will show that $\oplus : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$ defines a functor.
This will give us precious information on the behaviour of the biproduct.
As we already know how it acts on objects we will focus on finding how it acts on morphisms.

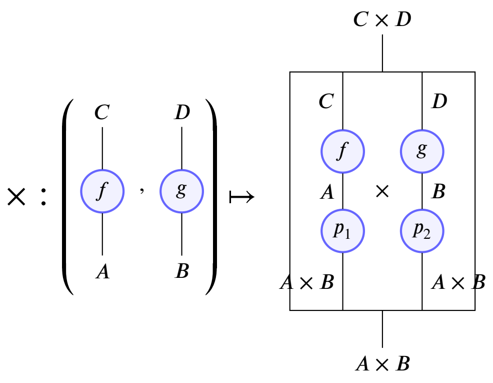
To this effect we want to have more precision on the nature of the isomorphism $\Psi : A \coprod B \to A \times B$.
In particular we notice that the following morphism is defined and satisfies 3.8:

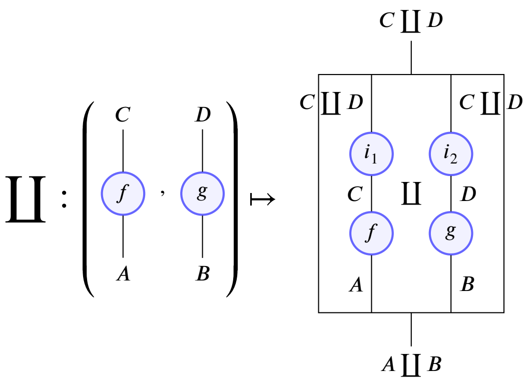


(3.10)

The equality is easily provable by composing with p_1 and p_2 and by observing the projections are the same.

As our category is equipped with products and coproducts, it is also equipped with a product functor and a coproduct functor acting on morphisms the following way:





We shall show that Ψ is a natural isomorphism from $\coprod$ to $\times$.

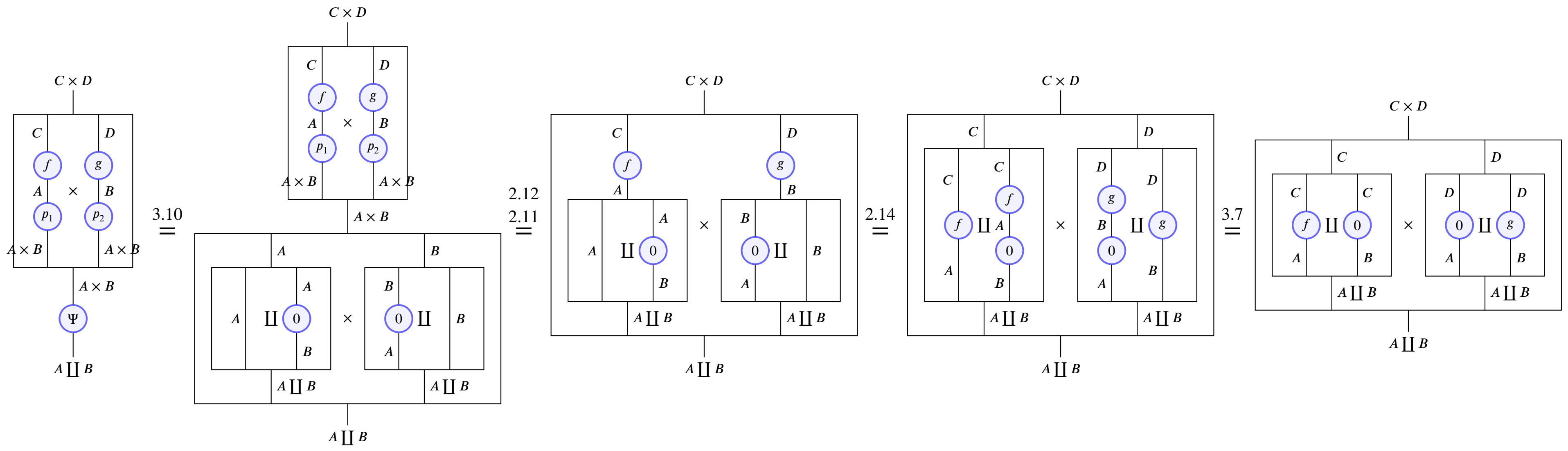
Lemma 3.2.3. Ψ *is a natural isomorphism.*

Proof. We already know that Ψ is an isomorphism from the definition of biproducts. It remains to show it is a natural transformation by showing the following diagram commutes for all f and g.

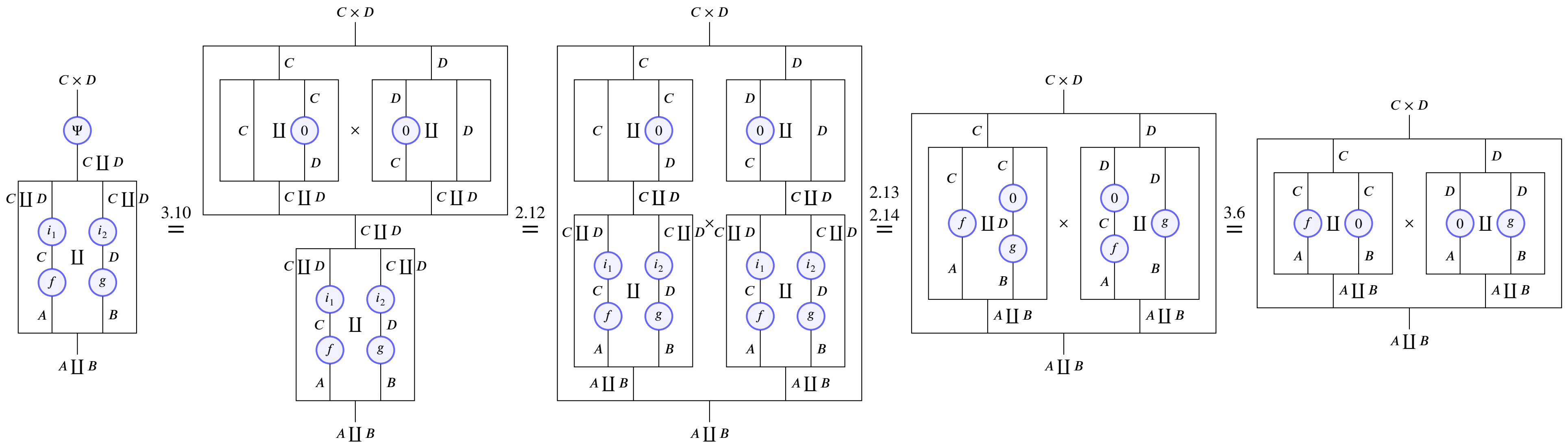
$$\begin{tikzcd}
A \coprod B \arrow[r, "\Psi_{A,B}"] \arrow[d, "f \coprod g"'] & A \times B \arrow[d, "f \times g"] \\
C \coprod D \arrow[r, "\Psi_{C,D}"] & C \times D
\end{tikzcd}$$

The evaluation of the two composites translates graphically to:

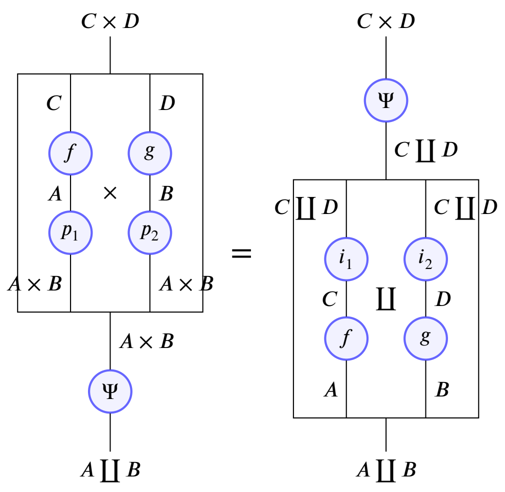
Calculation of $(f \times g) \circ \Psi_{A,B}$:



Calculation of $\Psi_{C,D} \circ (f \coprod g)$:



We have demonstrated the following equality:



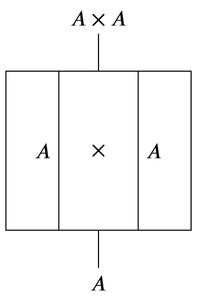
Therefore Ψ is a natural isomorphism. $\square$

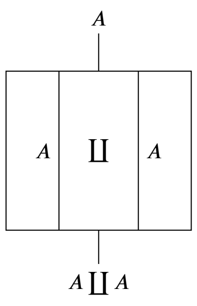
The functors $\coprod$ and $\times$ being naturally isomorphic, we define the biproduct functor $\oplus$ to be the one or the other. It is just a matter of convention.

We will now study the consequences of the existence of biproducts on hom-sets. $\forall f_1, f_2 : A \to B$, we note $f_1 + f_2$ the following composite:

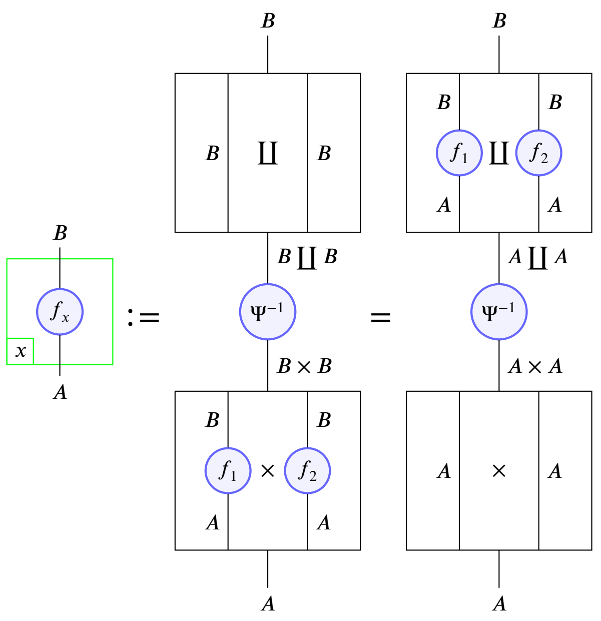
$$A \xrightarrow{\Delta} A \times A \equiv A \oplus A \xrightarrow{f_1 \oplus f_2} B \oplus B \equiv B \coprod B \xrightarrow{\nabla} B$$

With Δ and ∇ being the diagonal and the codiagonal maps defined as follows for all A:

$\Delta =$ 

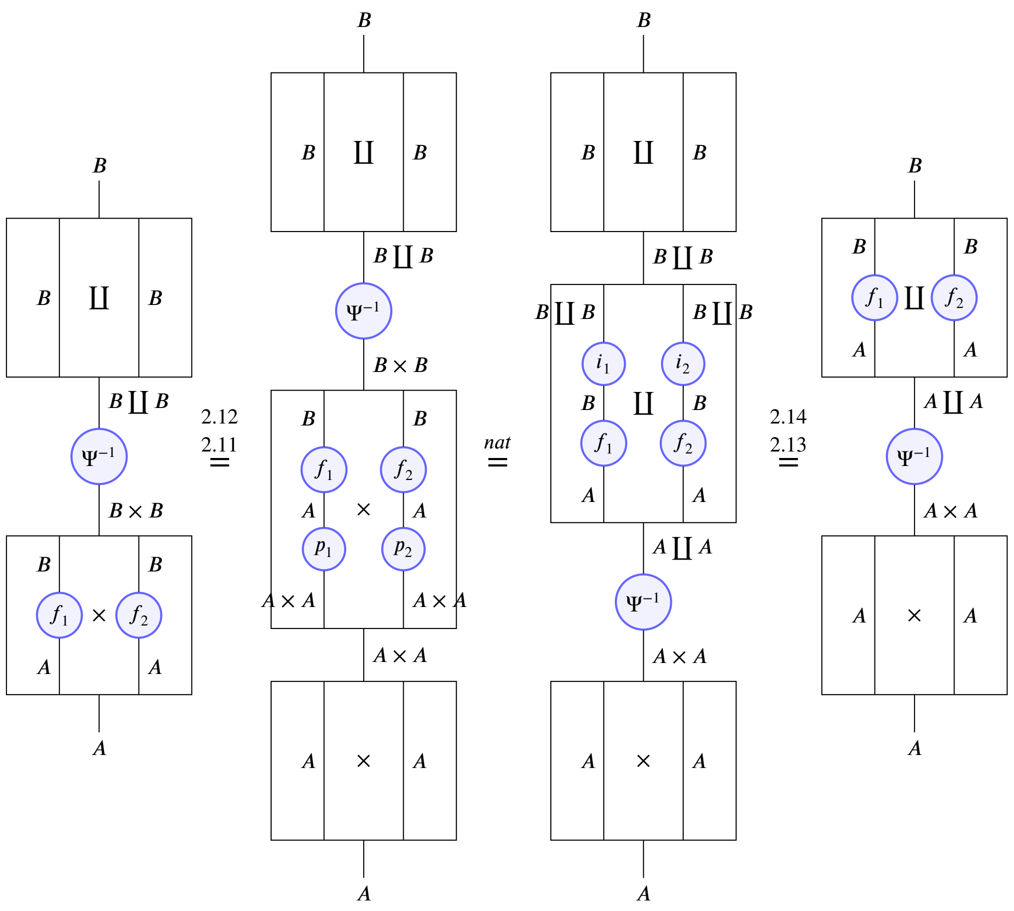
$\nabla =$ 

Graphically, $f_1 + f_2$ is defined the following way:

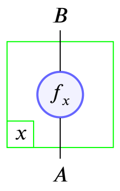


(3.11)

Proof. We prove the equality.



We obtain the second equality by naturality of $\mathbf{\Psi}^{-1}$. $\square$

Theorem 3.2.4. 

is a superposition rule with the 0*-morphism* $0_{A,B}$ *as unit.*

Proof.

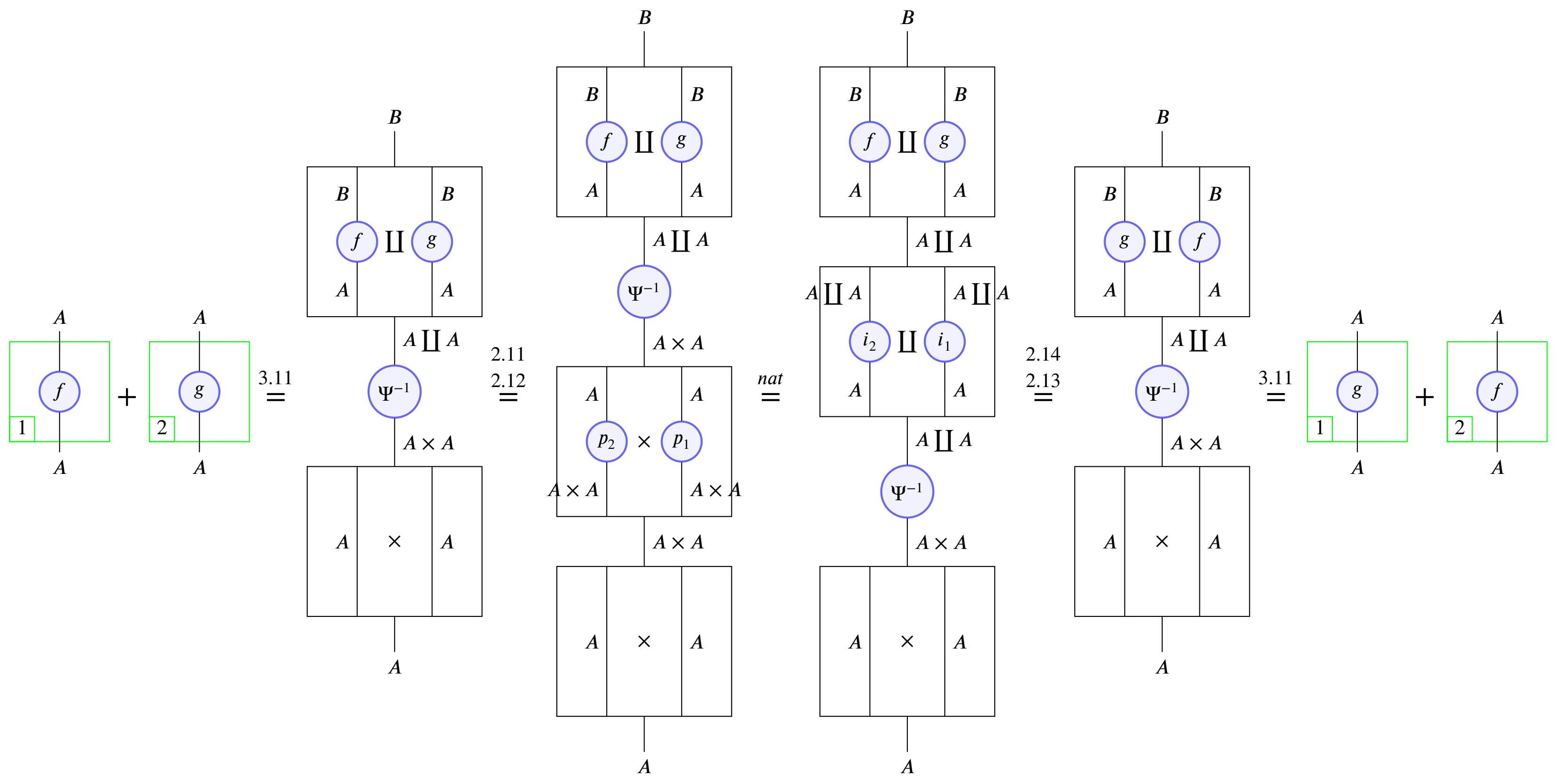
- *Commutativity*

  We show that for all $f, g : A \to B$,

  $$f + g = g + f$$

  This translates graphically to:

Calculation of $f + g$:

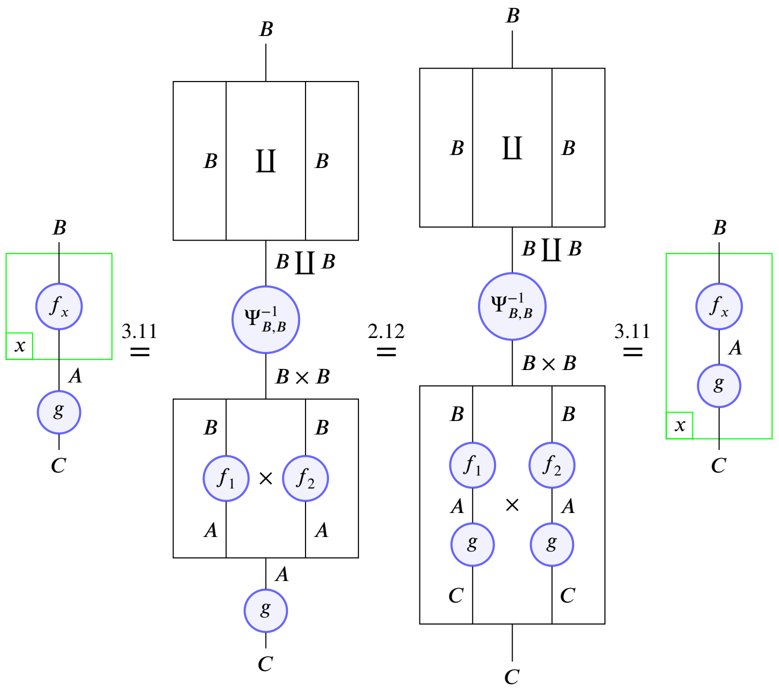


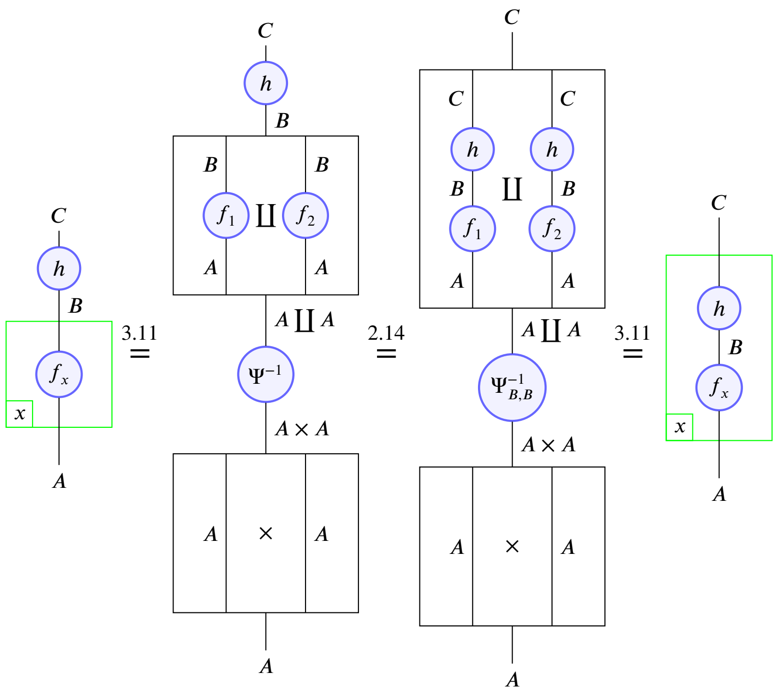
The second equality is obtained using the naturality of Ψ^{-1}.

- *Distributivity of the composition over the superposition*

  We show that for all maps $f_1, f_2 : A \to B, g : C \to A, h : B \to C$ we have:

$$(f_1 + f_2) \circ g = f_1 \circ g + f_2 \circ g$$
$$h \circ (f_1 + f_2) = h \circ f_1 + h \circ f_2$$



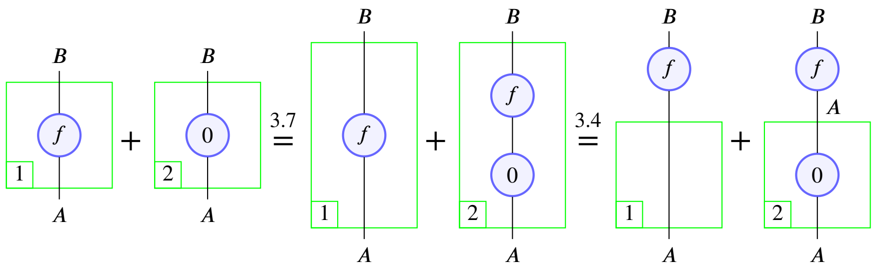


- *Units*

  We show that there exists a map $u_{A,B} : A \to B$ such that for all $f : A \to B$,

$$f + u_{A,B} = f$$

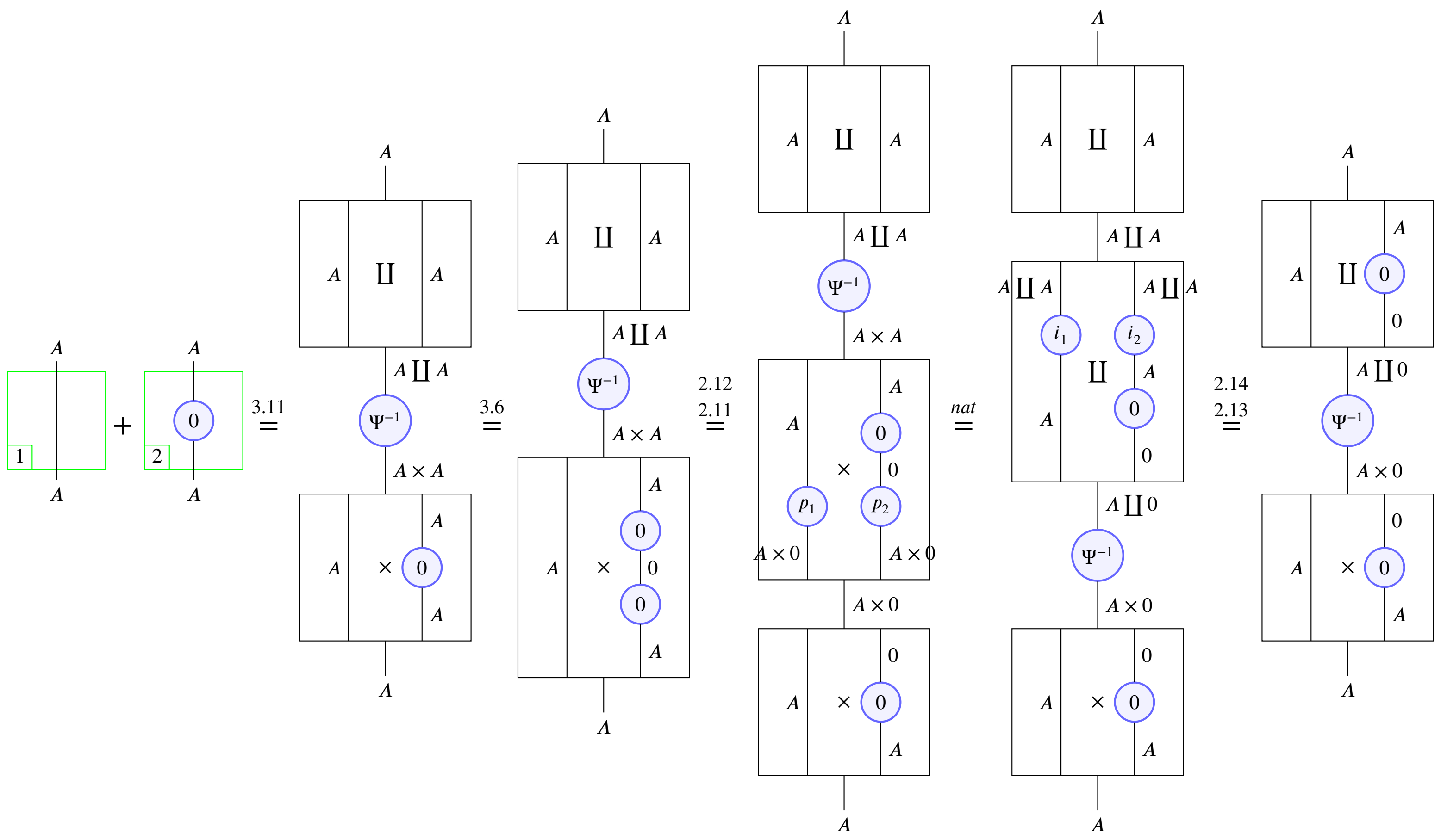
We remind that our candidates for unit morphisms are 0-morphisms.



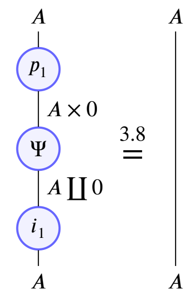
We shall clarify the notation used on the last diagram. The two maps f are outside the box, they are not part of the superposition therefore they are the same: they should not be considered as two instances of the same map f. It translates algebraically to: $f \circ (\mathrm{id}_A + 0_{A,A})$

We show that the superposition in the last diagram is equal to id_A.

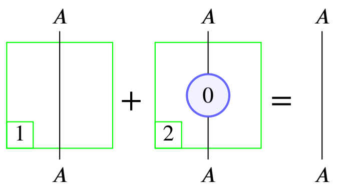
Calculation of $\mathbf{id}_A + 0_{A,A}$:



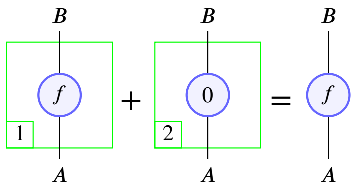
But the inverse of this morphism is precisely:



Therefore



Hence the result:

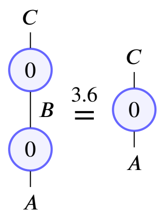


- *Units compose*

  We show that for all units $u_{A,B}$, $u_{B,C}$:

$$u_{B,C} \circ u_{A,B} = u_{A,C}$$

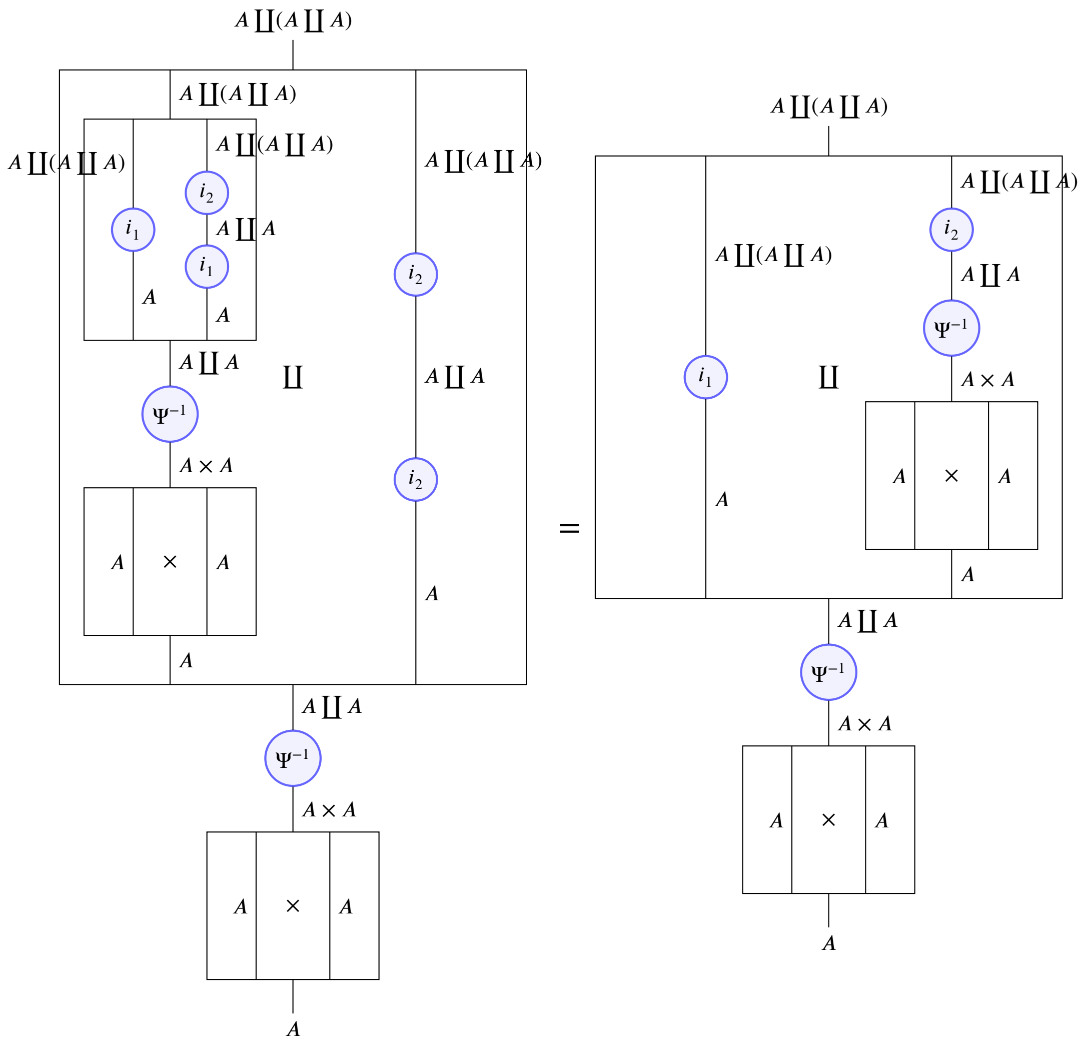
  By the properties of the 0-object we obviously have the following equality:



- *Associativity*

  We show that for all $f, g, h : A \to B$, we have $(f + g) + h = f + (g + h)$.

We start by showing the following equality:

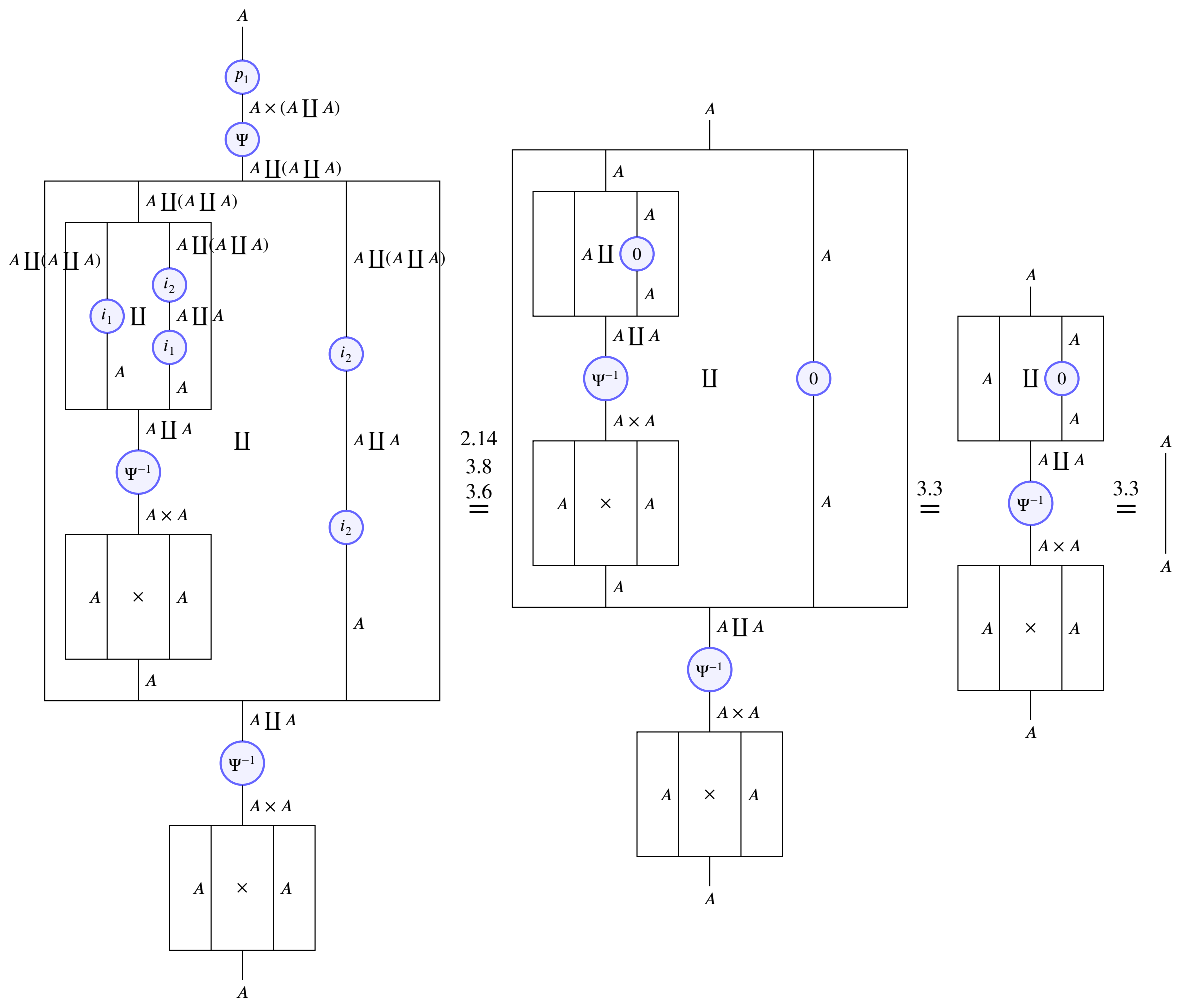


(3.12)

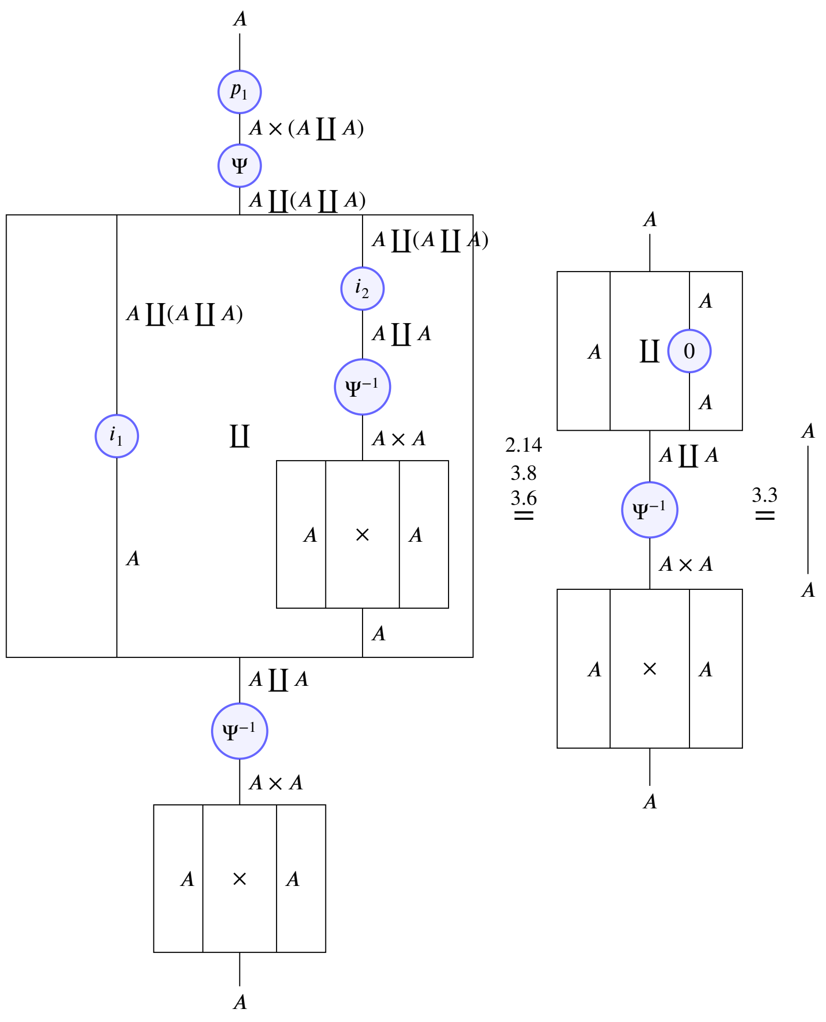
Composed with the isomorphism $\Psi_{A,A\coprod A}$ the codomain of these maps is a product. By the universal properties of the product, these maps are equal if their projections are equal.

We will call the first map f and the second map g.

Calculation of $p_1 \circ \Psi_{A, A \coprod A} \circ f$:

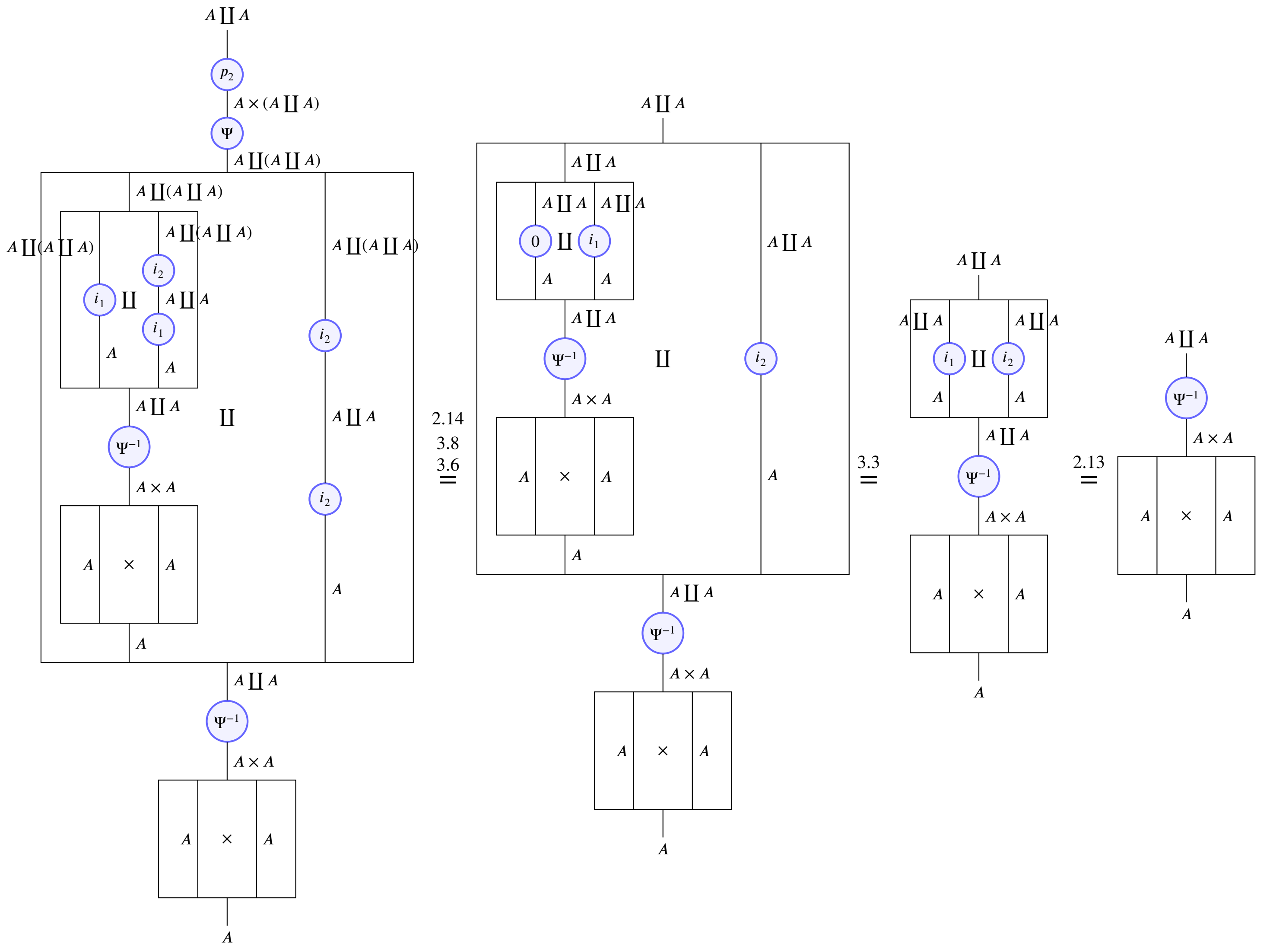


Calculation of $p_1 \circ \Psi_{A,A\coprod A} \circ g$:

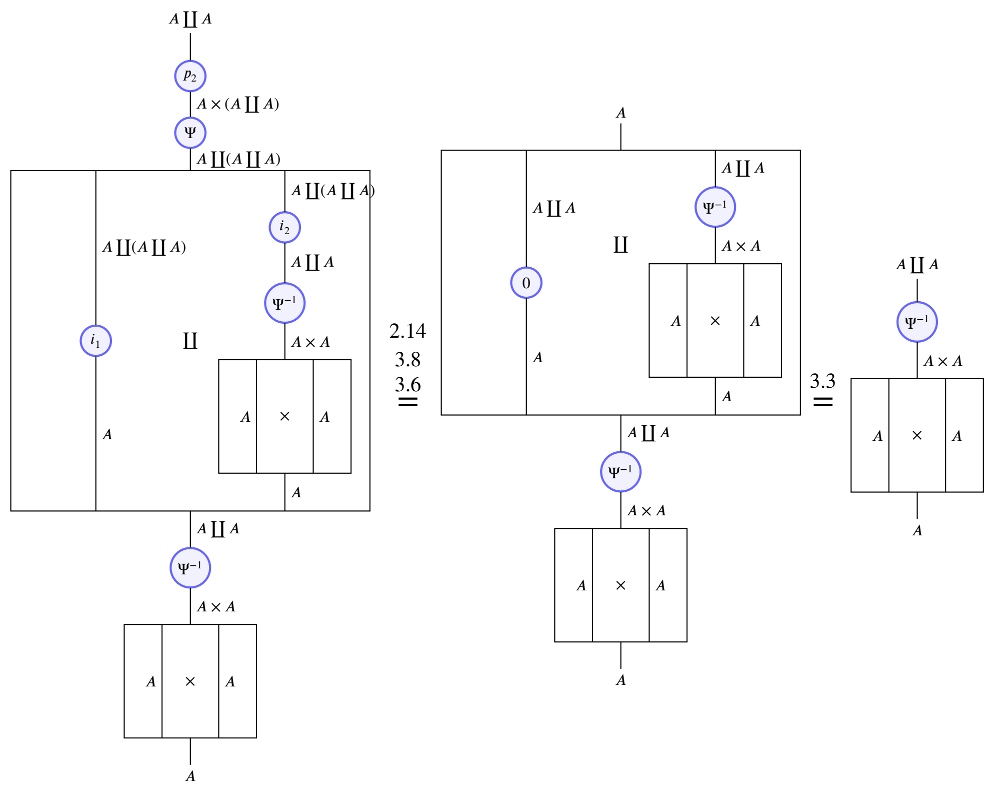


It remains to show the same applies for the second projection.

Calculation of $p_2 \circ \Psi_{A,A \coprod A} \circ f$:



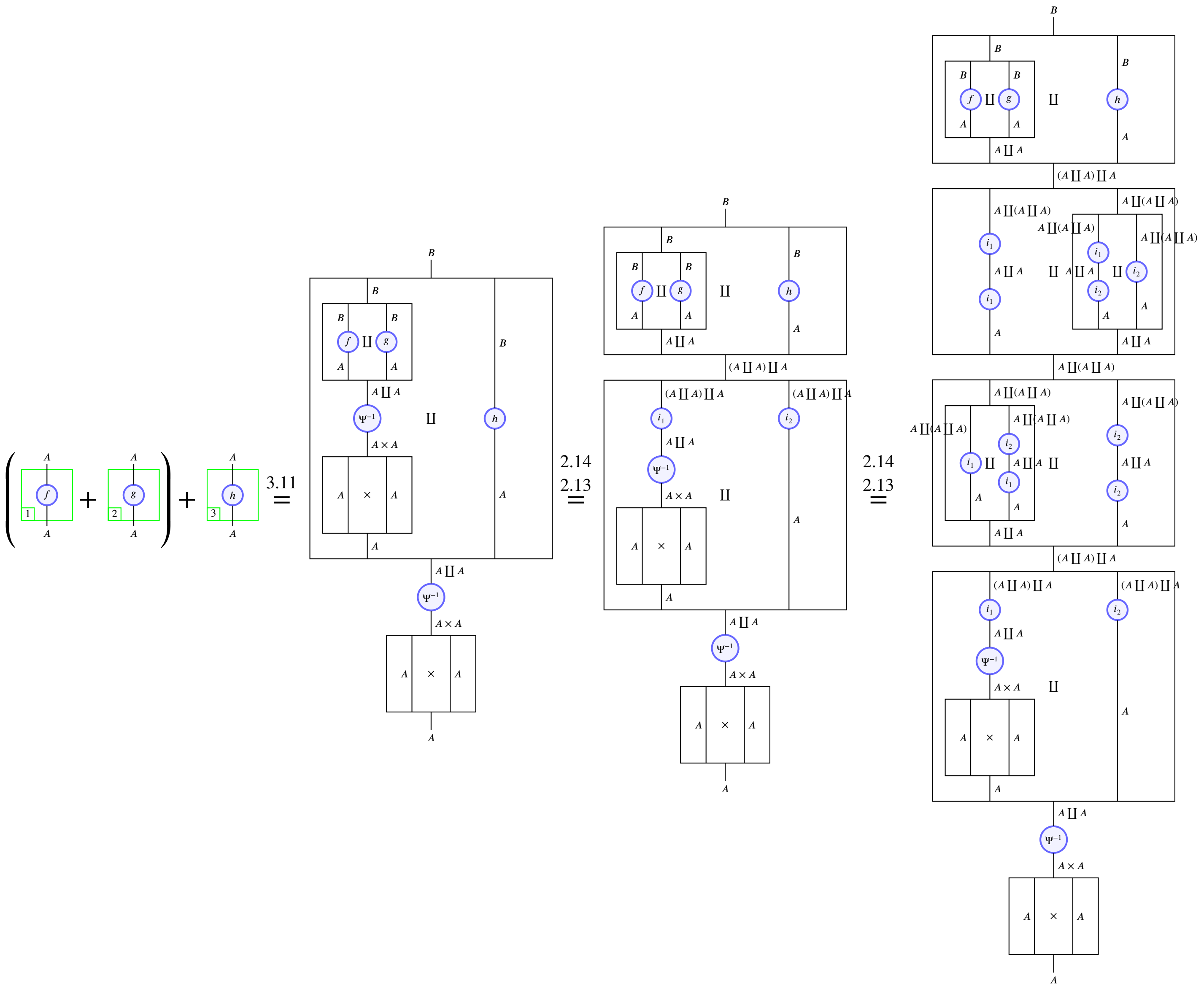
Calculation of $p_2 \circ \Psi_{A,A\coprod A} \circ g$:

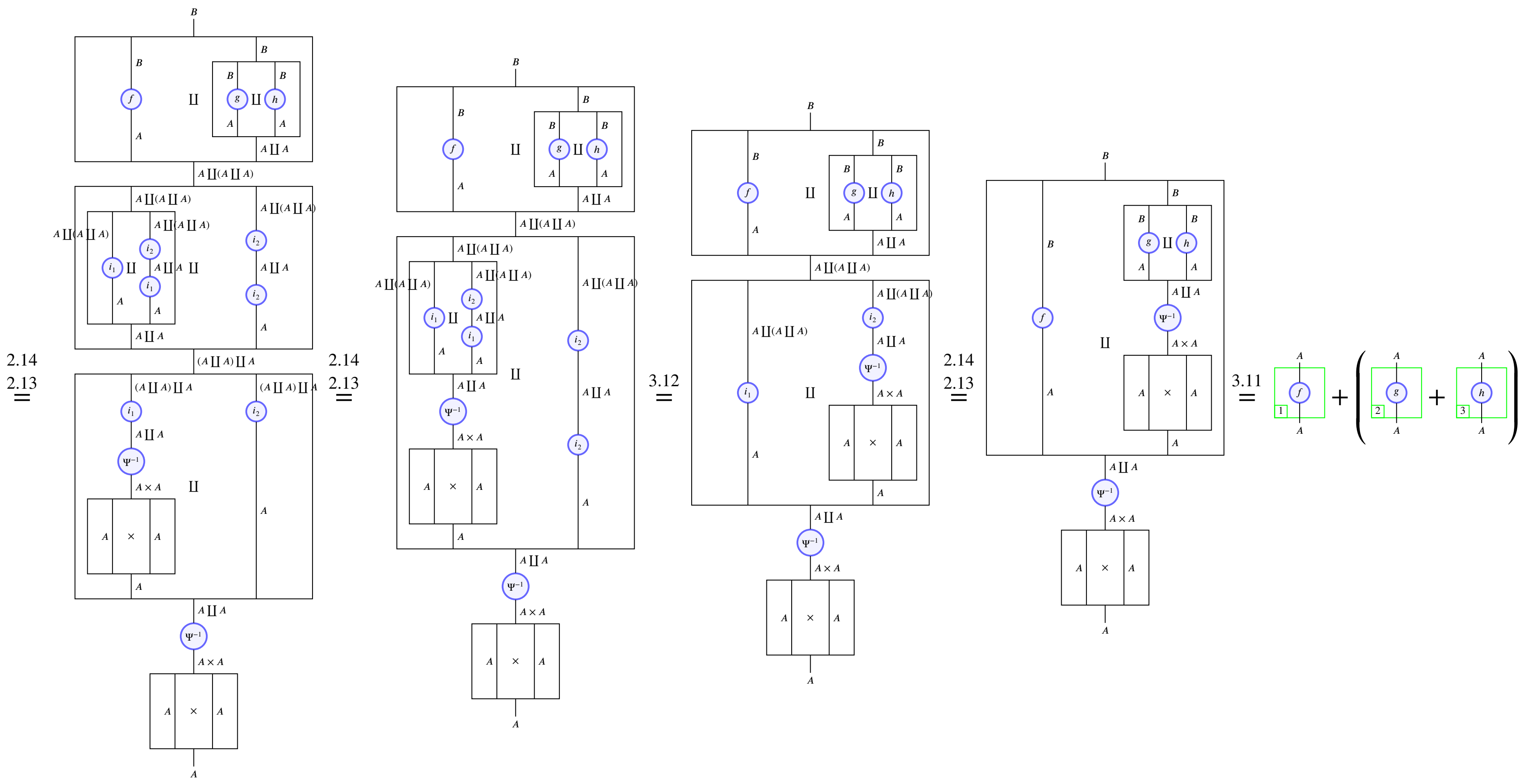


We conclude that the two morphisms are the same.

We now prove the associativity of the superposition by making explicit the associators $\alpha_{A,A,A} \ : \ (A\coprod A)\coprod A \to A\coprod(A\coprod A)$ and $\alpha^{-1}_{A,A,A} \ : \ A\coprod(A\coprod A) \to$

$(A \coprod A) \coprod A$ in the third equality:





2.14
2.13
=
2.14
2.13
=
3.12
=
2.14
2.13
=
3.11
=
+
+

We conclude the superposition is associative.

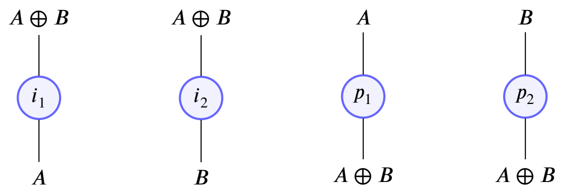
□

Therefore a category with biproducts and a 0-object has a superposition rule. We shall show later its uniqueness after having introduced a diagrammatic definition of the biproduct.

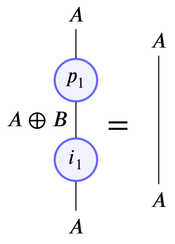
3.2.3 Diagrammatic definition of the biproduct

We now expose an alternative definition of biproducts [4, p. 196] which is equivalent to the first one.
This definition uses the superposition rule but having shown a superposition rule naturally arises from the existence of biproducts, this requirement is not too strong.

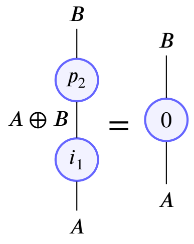
Definition 3.2.5 (Biproducts)**.** In a category with finite biproducts and a superposition rule, to each pair of objects A and B corresponds an object $A \oplus B$ equipped with the following maps:



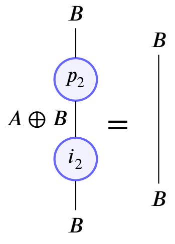
These maps are such that:



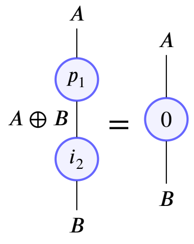
(3.13)



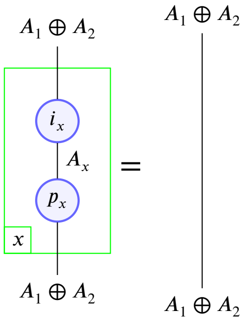
(3.14)



(3.15)



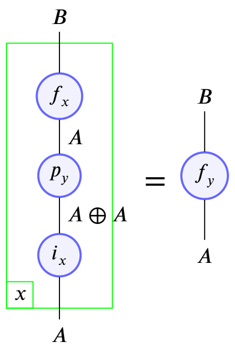
(3.16)



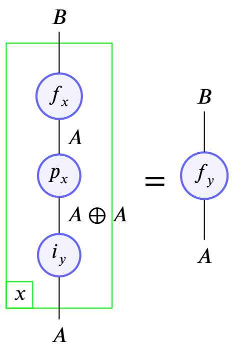
(3.17)

We shall prove this definition is equivalent to the first one.
But first we will formulate a useful lemma:

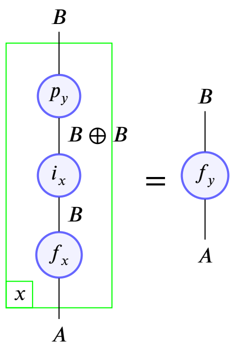
Lemma 3.2.6. *In a category with biproducts, for all families of maps* $f_x : A \to B$ *indexed over* x*, we have:*



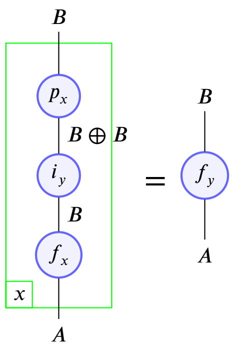
(3.18)



(3.19)

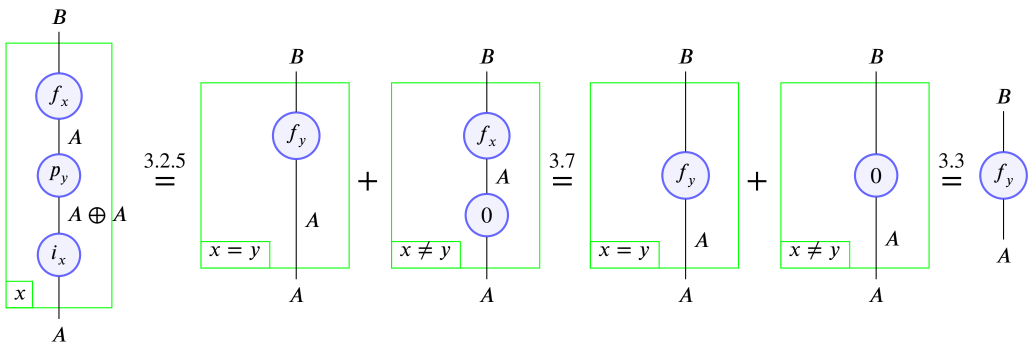


(3.20)



(3.21)

Proof. We will only prove the first equation. The three others are proved similarly.



□

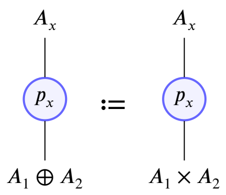
We can now prove the equivalence of the two definitions of biproducts.

Proposition 3.2.7. *The definitions 3.2.2 and 3.2.5 are equivalent.*

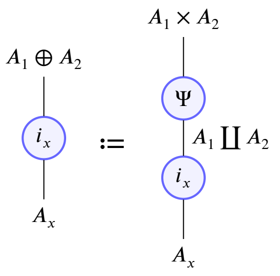
Proof. 3.2.2 ⇒ 3.2.5

Let $\mathscr{C}$ be a category with biproducts defined as in 3.2.2.
We take the biproduct functor $\oplus$ to be the product functor $\times$.

We define the following maps:

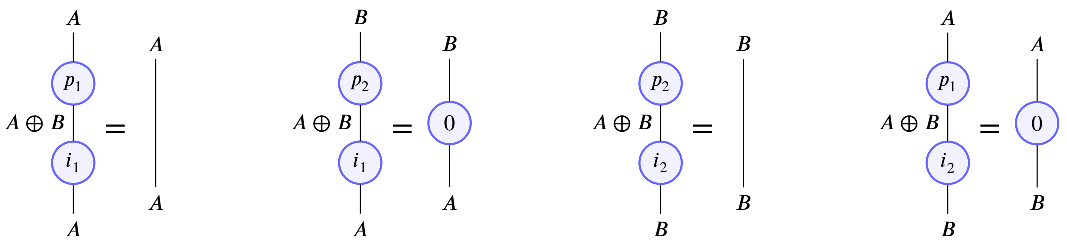


(3.22)

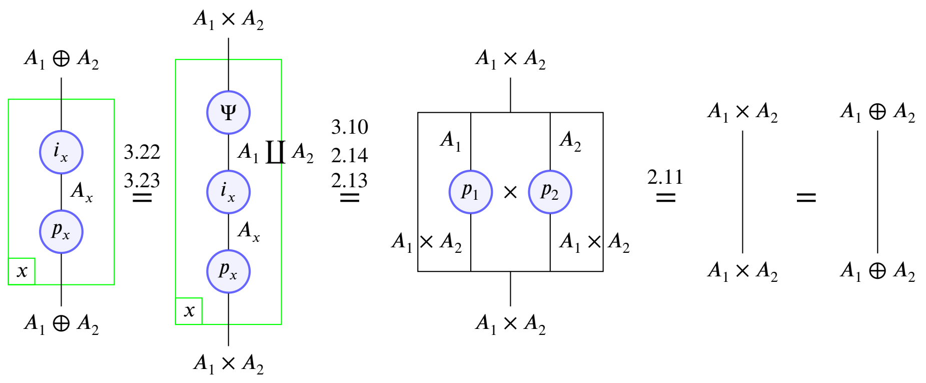


(3.23)

Note the use of types allowing us to distinguish maps having the same name.
By the definition of Ψ, we have the following equalities:



It remains to show that the fifth equality of the second definition of the biproduct holds.



The third equality is obtained by using the definition of Ψ (3.10) and the fact that the product is linear:

$$\langle f+g,h\rangle = \langle f,h\rangle + \langle g,h\rangle$$
$$\langle f,g+h\rangle = \langle f,g\rangle + \langle f,h\rangle$$

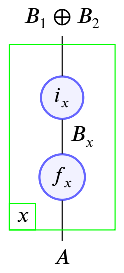
These equalities are easily proven by noticing the projections of each side are equal.

The last equality results from the fact our biproduct functor is the product functor.

We conclude the first definition of biproducts implies the second definition of biproducts.
It remains to show the converse.

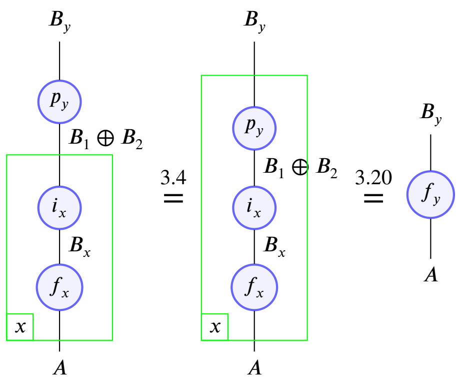
3.2.5 ⇒ 3.2.2
It suffices to show a biproduct as defined in 3.2.5 is both a product and a coproduct and that $\mathrm{id}_{A\oplus B}$ is the unique automorphism $\Psi : A \oplus B \to A \oplus B$ such that $p_x \circ \Psi \circ i_y = \delta_{x,y}$.

We show it is a product:
$\forall f_1 : A \to B_1, \forall f_2 : A \to B_2$, we consider the following map:

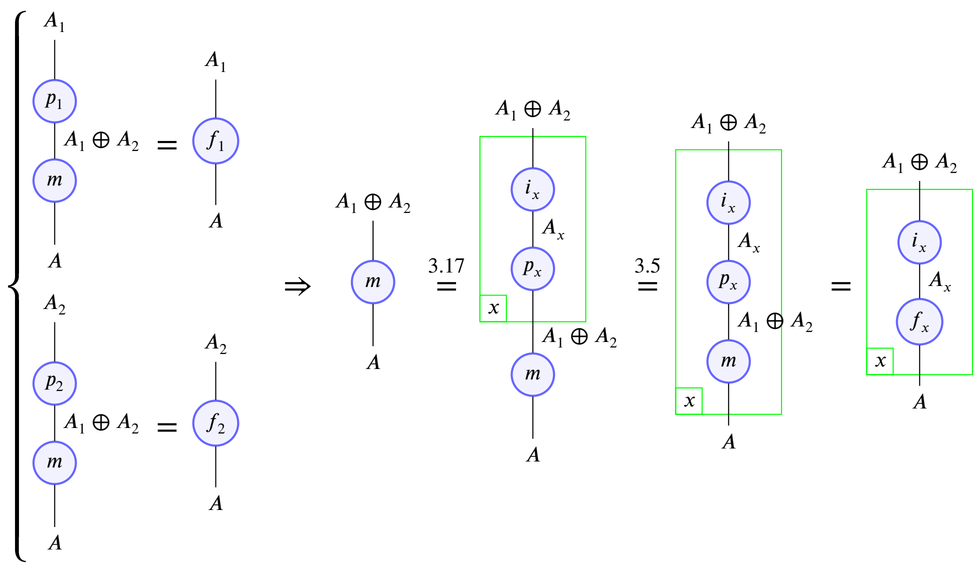


We show this is the only map m such that $p_1 \circ m = f_1$ and $p_2 \circ m = f_2$.

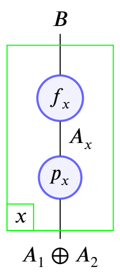
First we show our candidate satisfies the previous equations:



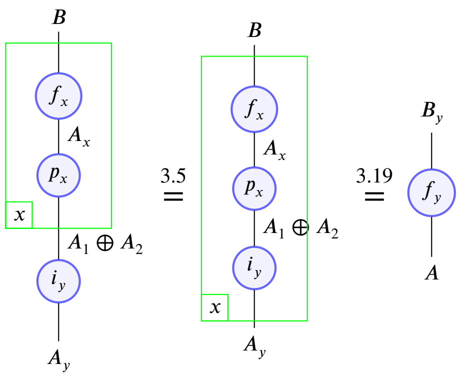
Now we show the uniqueness of this map,



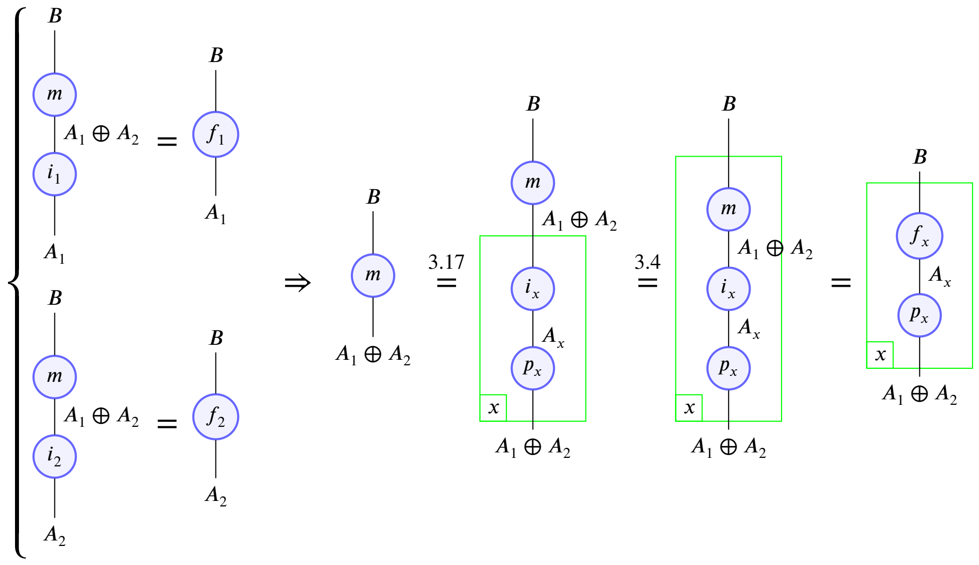
Similarly we show $B_1 \oplus B_2$ is a coproduct.
$\forall f_1 : A_1 \to B, \forall f_2 : A_2 \to B$, we consider the following map:



We show this is the only map m such that $m \circ i_1 = f_1$ and $m \circ i_2 = f_2$. First we show our candidate satisfies the previous equations:



We show the uniqueness of m,



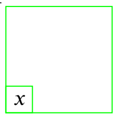
Finally, Ψ is uniquely defined such that $p_x \circ \Psi \circ i_y = \delta_{x,y}$ by the universal properties of the product and of the coproduct. In particular it must be equal to 3.10. As $\mathrm{id}_{A \oplus B}$ is a candidate for such a Ψ, they must coincide.

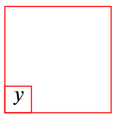
Therefore the two definitions are equivalent. $\square$

Finally, we can prove the uniqueness of the superposition rule arising from the existence of biproducts.

Theorem 3.2.8 (Uniqueness of the superposition rule)**.** *A category with biproducts and a* 0*-object has a unique superposition rule.*

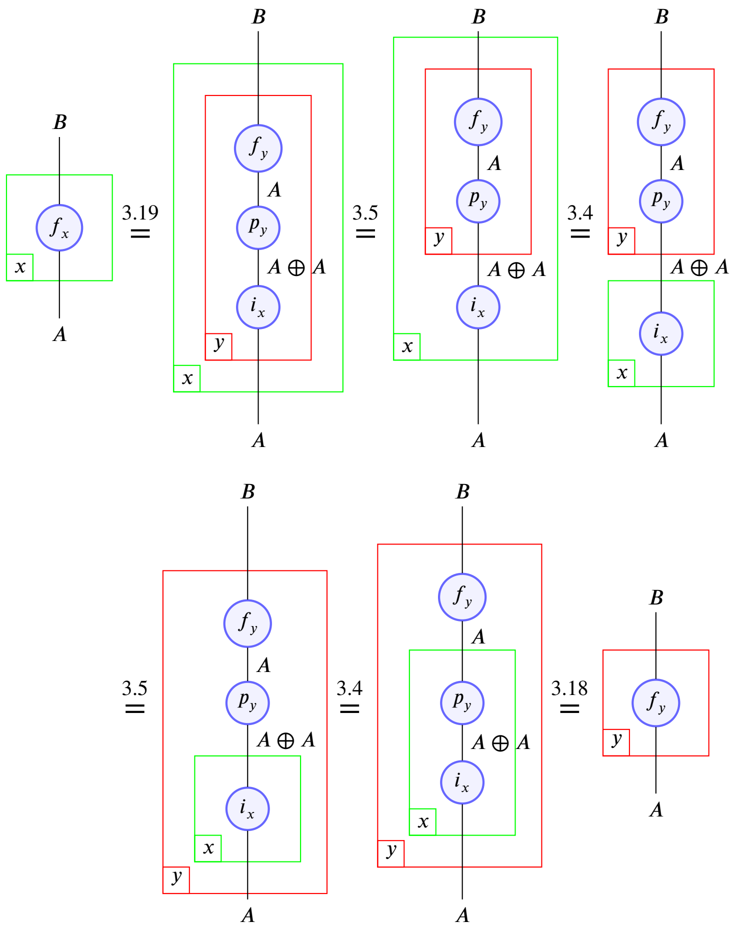
Proof. We have already proved a category with biproducts and a 0-object has a superposition rule. We shall prove it is unique.

Let 

and 

be two superposition rules.

For all families of maps $f_x : A \to B$ indexed over x,



The superposition rule is therefore unique. □

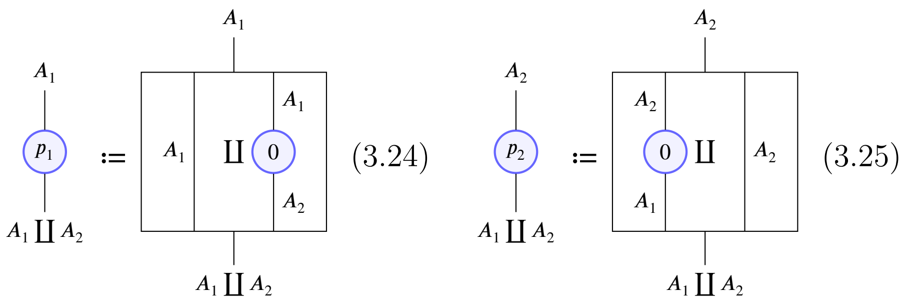
3.2.4 The superposition rule entails coproducts are biproducts

Now that we have a diagrammatic way to represent the superposition rule and the biproducts, we shall show that the existence of the superposition rule implies coproducts (and products) are biproducts.

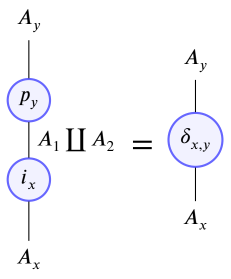
Theorem 3.2.9. *In a category equipped with a superposition rule and a* 0*-object, coproducts (and products) are biproducts.*

Proof. We will only prove the case of coproducts as we prove the case of products in a similar fashion. We will follow the argumentation of the proof given by Garner and Schäppi [2].

As our category has a 0-object, for any coproduct $A_1 \coprod A_2$ the following two maps are defined:



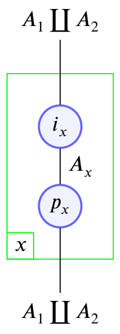
We notice we have the following equality:



with $\delta_{x,y}$ equals to $\mathbf{id}_{A_x}$ if $x = y$ and $0_{A_x,A_y}$ otherwise.

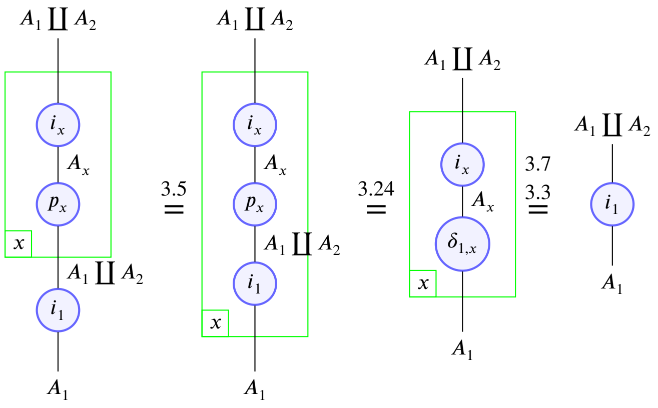
This equality looks familiar. Indeed, we have just recovered four out of the five equations these morphisms have to satisfy for $A_1 \coprod A_2$ to be a biproduct.

As our category also has a superposition rule, the following morphism is defined:

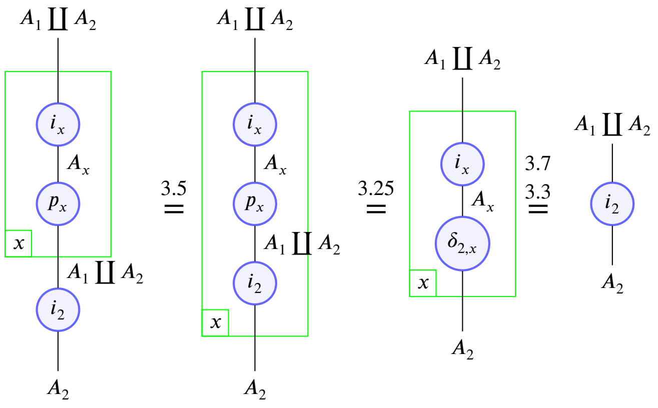


If we can show this morphism is equal to $\mathbf{id}_{A_1 \coprod A_2, A_1 \coprod A_2}$, we will have recovered all the properties of the biproduct.

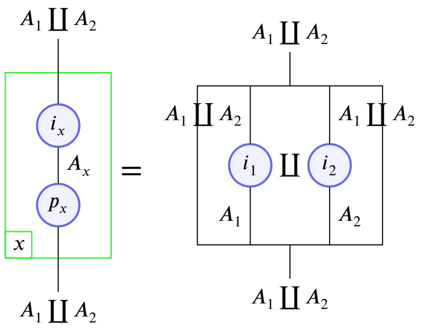
We calculate the following composite:



Likewise, we have:

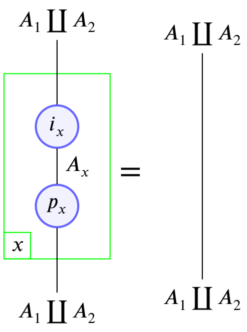


But according to the universal properties of the coproduct this signifies:



According to the universal properties of the coproduct this morphism is precisely $\mathrm{id}_{A_1 \coprod A_2, A_1 \coprod A_2}$.

Hence the result:



The coproduct $A_1 \coprod A_2$ is therefore a biproduct.

□

3.3 The superposition rule in monoidal categories

We shall specify some conditions for a monoidal category to have a superposition rule. Houston [3] already proved it sufficed for a compact closed category to have finite products (or finite coproducts) for the category to be semiadditive.

Recently, Garner and Schäppi [2] proved that in a monoidal category having nullary and binary copowers of I preserved by $A \otimes -$, the initial object 0 and $I \coprod I$ have right duals if and only if the category is semiadditive.

As we have just shown that if a category is semiadditive then it is equipped with a unique superposition rule, these results will inform us on conditions for a monoidal category to have a superposition rule.

The following result is due to Garner and Schäppi [2].

Theorem 3.3.1. *In a monoidal category $(\mathscr{C}, \otimes, I)$ with nullary and binary copowers of I preserved by $A \otimes -$, the following propositions are equivalent:*

1. *The initial object* 0 *and $I \coprod I$ have a right dual;*
2. *The identity functor has a counital comagma structure in the functor category $[\mathscr{C}, \mathscr{C}]$;*
3. *$\mathscr{C}$ is semiadditive.*

Proof. The proof is adapted from [2].

$1 \Rightarrow 2$

We shall first show 0 is a 0-object, this will be our candidate for the counit of our comagma. As 0 has a right dual, its coevaluation map is defined $\eta : I \to 0^* \otimes 0$.

As left-tensoring preserves coproducts, it preserves in particular the nullary coproduct which is the initial object 0. Therefore $\eta : I \to 0^* \otimes 0 \equiv 0$.
By left-tensoring with any identity we obtain the natural family: $\rho_A = \mathrm{id}_A \otimes \eta : A \to 0$. There is therefore a cocone $\mathrm{id}_{\mathscr{C}} \Rightarrow \Delta_0$ from the identity functor on $\mathscr{C}$ to the constant functor Δ_0 mapping objects of $\mathscr{C}$ to 0.
It suffices to show for all A, ρ_A is the unique arrow $A \to 0$ to deduce 0 is terminal.

Consider this diagram:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "\rho_A"] & 0 \arrow[d, "\rho_0"] \\ & 0 \end{tikzcd} \tag{3.26}$$

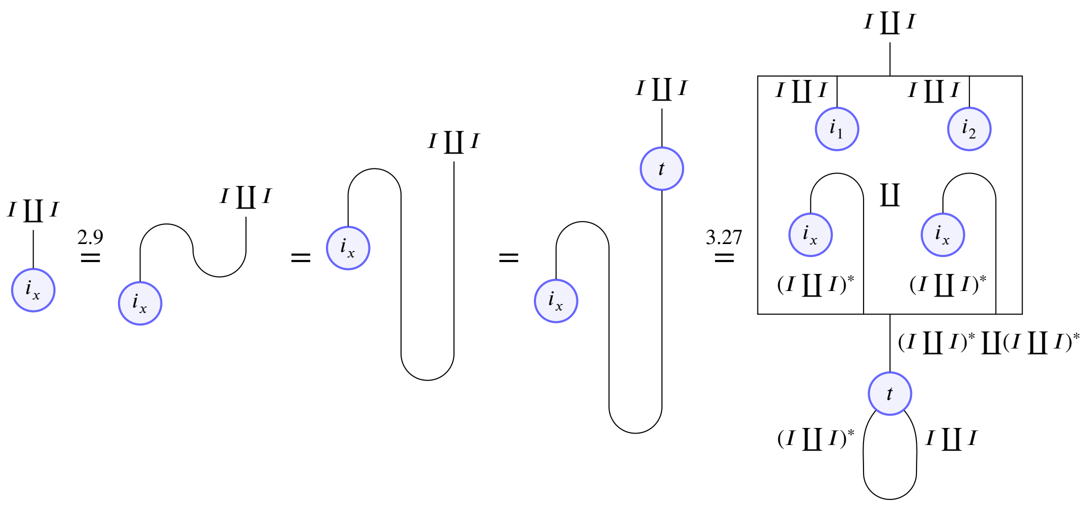
As 0 is initial ρ_0 must be equal to id_0.
ρ being a cocone $\mathrm{id}_{\mathscr{C}} \Rightarrow \Delta_0$ this triangle commutes for all f.
We deduce $f = \rho_A$.
Therefore 0 is also a terminal object and thus a 0-object.
0-morphisms therefore exist. We will use then as counits for the comagma structure later in the proof.

We will now determine the comultiplication.

Preservation of copowers of I by left tensoring is witnessed by the natural isomorphism t:

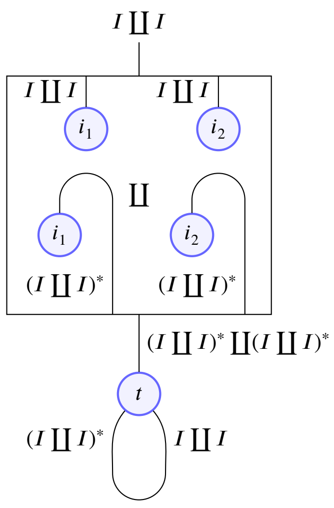
$$\begin{tikzcd} A \otimes (I \coprod I) \arrow[r, "t_A"] \arrow[d, "f \otimes \mathrm{id}_{I \coprod I}"] & A \coprod A \arrow[d, "f \coprod f"] \\ B \otimes (I \coprod I) \arrow[r, "t_B"] & B \coprod B \end{tikzcd} \tag{3.27}$$

$\mathscr{C}$ having copowers of I, it is equipped with coprojection maps $i_x : I \to I \coprod I$. Moreover $I \coprod I$ has a right dual. These properties allow us to write:



The third equality is obtained by making explicit the isomorphism $t_I : I \otimes (I \coprod I) \to I \coprod I$ which is actually equal to the left unitor $\lambda_{I \coprod I}$.

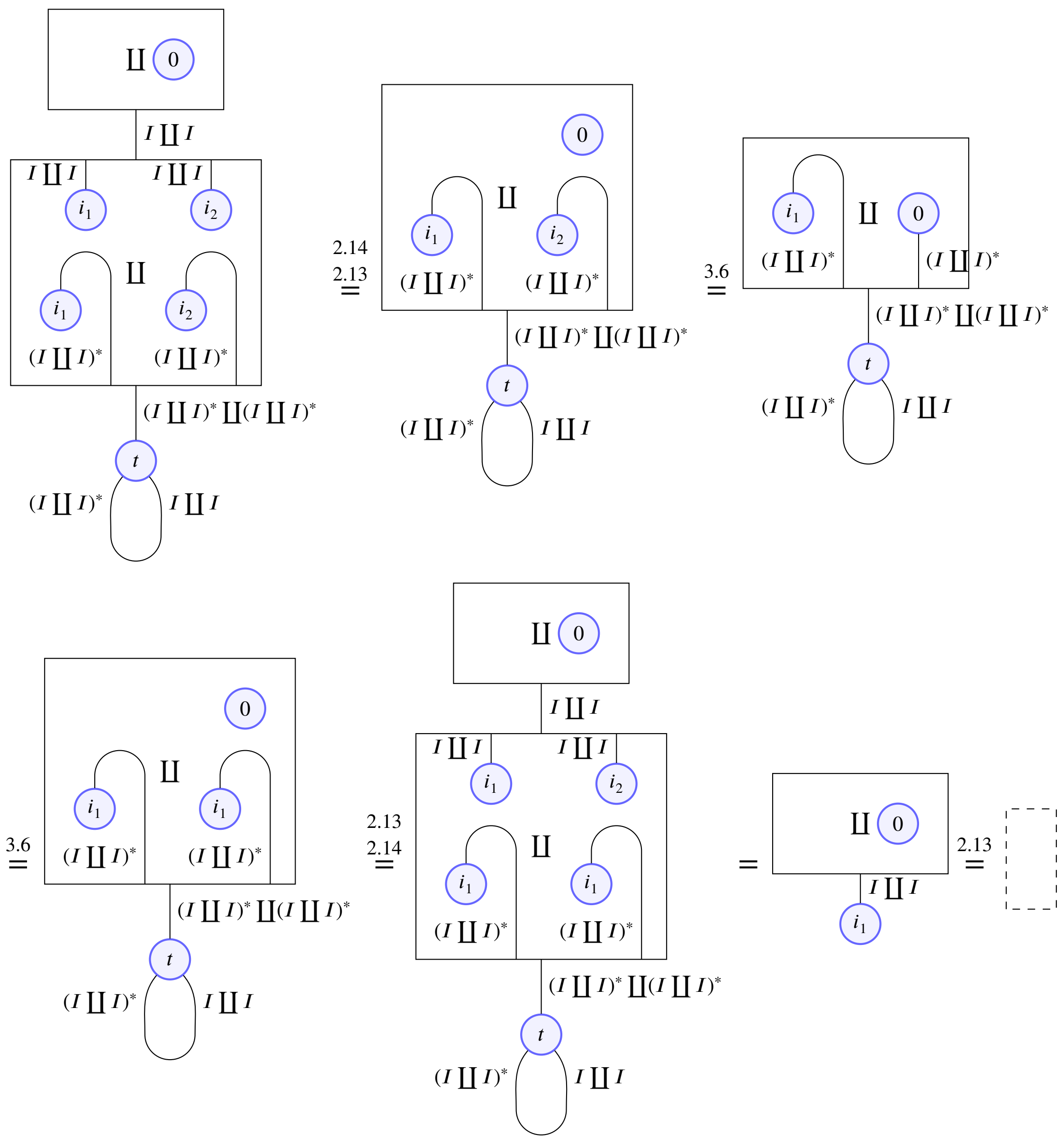
Now consider the following map:



We shall show it is the comultiplication δ of a counital comagma. To this effect we show

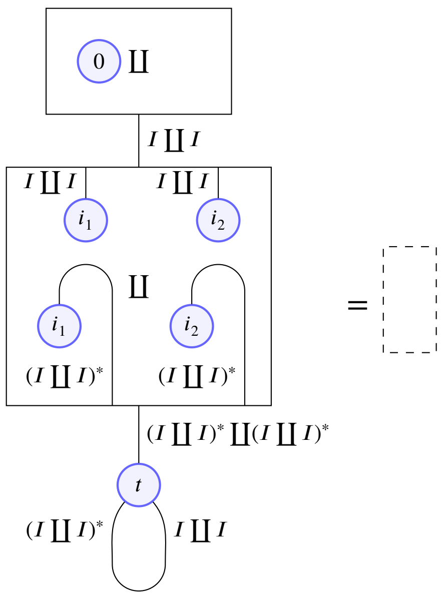
$$[\mathrm{id}_I, 0_{I,I}] \circ \delta = [0_{I,I}, \mathrm{id}_I] \circ \delta = \mathrm{id}_I$$

This translates graphically to:



The dashed rectangle represents the diagram containing the identity for I.

Similarly, we show:



As we have natural isomorphisms $A \otimes (I + I) \cong A + A$ and $A \otimes 0 \cong 0$ the identity functor in $[\mathscr{C}, \mathscr{C}]$ is equipped with a counital comagma structure.

$2 \Rightarrow 3$

We shall now prove $\mathscr{C}$ is semiadditive.

Suppose the identity functor in $[\mathscr{C}, \mathscr{C}]$ is equipped with a counital comagma structure. We therefore have the following natural maps: $\delta_A : A \to A \coprod A$ and $\epsilon_A : A \to 0$. They satisfy the following equation:

$$(\epsilon_A \coprod \mathrm{id}_A) \circ \delta = (\mathrm{id}_A \coprod \epsilon_A) \circ \delta = \mathrm{id}_A$$

Naturality of δ makes the rectangular diagram commute whereas naturality of the coprojection maps $i_x : A \to A \coprod A$ makes the triangular diagram commute.

$$\begin{tikzcd}
A \arrow[r, "\delta_A"] \arrow[d, "\delta_A"] & A \coprod A \arrow[d, "\delta_A \coprod \delta_A"] \\
A \coprod A \arrow[r, "\delta_{A \coprod A}"] & A \coprod A \coprod A \coprod A
\end{tikzcd}$$

$$\begin{tikzcd}
& A \coprod A \arrow[dl, "\delta_{A \coprod A}"'] \arrow[dr, "\delta_A \coprod \delta_A"] & \\
A \coprod A \coprod A \coprod A \arrow[rr, "\mathrm{id}_A \coprod [i_2, i_1] \coprod \mathrm{id}_A"] & & A \coprod A \coprod A \coprod A
\end{tikzcd}$$

In the case of the triangular diagram, it suffices to make explicit the isomorphisms corresponding to the bracketing, to compose the maps and to use naturality of δ to see that the diagram commutes.

Using the information of these two diagrams, we can claim the unital magma $(A, \delta_A, \epsilon_A)$ in $\mathscr{C}^{op}$ is medial. Indeed, in $\mathscr{C}^{op}$ the triangular diagram tells us for all generalized elements $a, b, c, d : * \to A$

$$(\delta_A(a,b), \delta_A(c,d)) = \delta_{A \coprod A}(a,c,b,d)$$

In turn, the rectangular diagram allows us to write:

$$\delta_A(\delta_{A \coprod A}(a,b,c,d)) = \delta_A(\delta_A(a,b), \delta_A(c,d))$$

By combining these two informations we can affirm our unital magma is medial because the following equation holds.

$$\delta_A(\delta_A(a,b), \delta_A(c,d)) = \delta_A(\delta_A(a,c), \delta_A(b,d))$$

By the Eckmann-Hilton argument, this multiplication is therefore associative and commutative.
Hence the commutative monoid structure of $(\mathscr{C}(A,B), \mathscr{C}(\delta_A, B), \mathscr{C}(\epsilon_A, B)$.

$\mathscr{C}$ is therefore semiadditive.

$3 \Rightarrow 1$

Suppose $\mathscr{C}$ is semiadditive, we have to show 0 and $I \coprod I$ have a right dual.
We remind that in a monoidal category $\mathscr{C}$, an object has a left (corresponding right) dual if this object seen as a morphism in the delooping bicategory $\mathscr{B}\mathscr{C}$ has a left (corresponding right) adjoint.
Therefore an object X has a right dual X^* if for all objects A, B we have the natural bijection:

$$\hom(A, B \otimes X) \cong \hom(A \otimes X^*, B)$$

$\mathscr{C}$ being semiadditive, it has a superposition rule implying that copowers are biproducts (3.2.9), in particular they are powers. $\mathscr{C}$ having a nullary coproduct, it is also a nullary product. Therefore the initial object 0 is also terminal.

As left or right tensoring preserves initiality and terminality, the following equivalence holds:

$$\hom(A, B \otimes 0) \cong \hom(A \otimes 0, B)$$

Indeed, these two hom-sets have the same cardinality if we consider that $B \otimes 0$ is terminal and that $A \otimes 0$ is initial.
0 is therefore self-dual.

As for the coproduct $I \coprod I$, we obviously have the following equivalences:

$$\hom(A \otimes (I \coprod I), B) \cong \hom(A \coprod A, B) \cong \hom(A, B) \coprod \hom(A, B)$$

But for all B, $B \otimes (I \coprod I) \cong B \coprod B$ is also a product.
Therefore,

$$\hom(A, B \otimes (I \coprod I)) \cong \hom(A, B \coprod B) \cong \hom(A, B) \coprod \hom(A, B)$$

Hence the equivalence

$$\hom(A, B \otimes (I \coprod I)) \cong \hom(A \otimes (I \coprod I), B)$$

We conclude that $I \coprod I$ is also self-dual. □

3.4 Interaction between the monoidal structure and the superposition rule

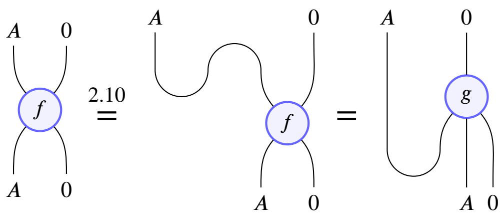
Interesting results arise from the interaction of the monoidal structure and the superposition rule. In particular one of the contributions of this thesis will be to prove in a purely diagrammatic way that the tensor product distributes over the superposition rule in the presence of biproducts under a certain condition. This result has already been proved in [7] but the proof used stronger assumptions and was more laborious.

We shall first introduce three lemmas we will use to prove our main result.

Lemma 3.4.1. *In a monoidal category with a* 0*-object, if* A *has a dual we have the following result:*

$$0 \otimes A \equiv 0 \equiv A \otimes 0$$

Proof. We follow the proof exposed in [7].
We suppose A has a left dual A^*. The proof would be the same for A having a right dual.

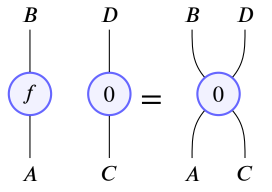


As there is only one such morphism $g : A^* \otimes A \otimes 0 \to 0$, g is unique and so must be f.
Therefore f must be the identity $\mathbf{id}_{A\otimes 0} : A \otimes 0 \to A \otimes 0$.
We also know the following composite exists $A \otimes 0 \xrightarrow{0_{A\otimes 0,0}} 0 \xrightarrow{0_{0,A\otimes 0}} A \otimes 0$ and it must be equal to $\mathbf{id}_{A\otimes 0}$.
Therefore, the morphisms $A \otimes 0 \xrightarrow{0_{A\otimes 0,0}} 0$ and $0 \xrightarrow{0_{0,A\otimes 0}} A \otimes 0$ must be inverse to each other.

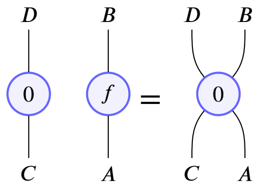
Hence the result $A \otimes 0 \equiv 0$.

□

Lemma 3.4.2. *In a monoidal category with a* 0*-object, if either* A *or* B *has a dual:*



(3.28)



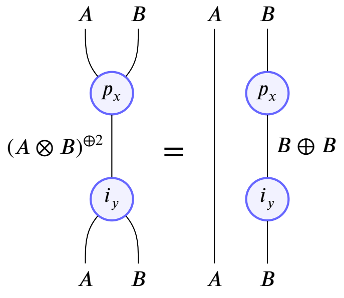
(3.29)

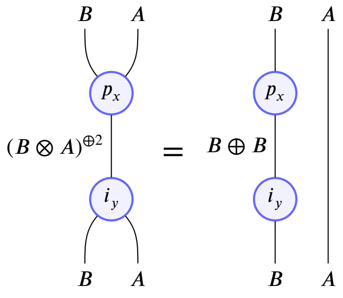
Proof. We follow the proof exposed in [7].
We will only prove the first equation as the second one can be proved similarly.
Suppose A has a dual. $f \otimes 0_{C,D}$ factorizes through $A \otimes 0$ but we know that $A \otimes 0 \equiv 0$ 3.4.3 therefore $f \otimes 0_{C,D}$ must be a 0-morphism. As it is unique, we have the result.

□

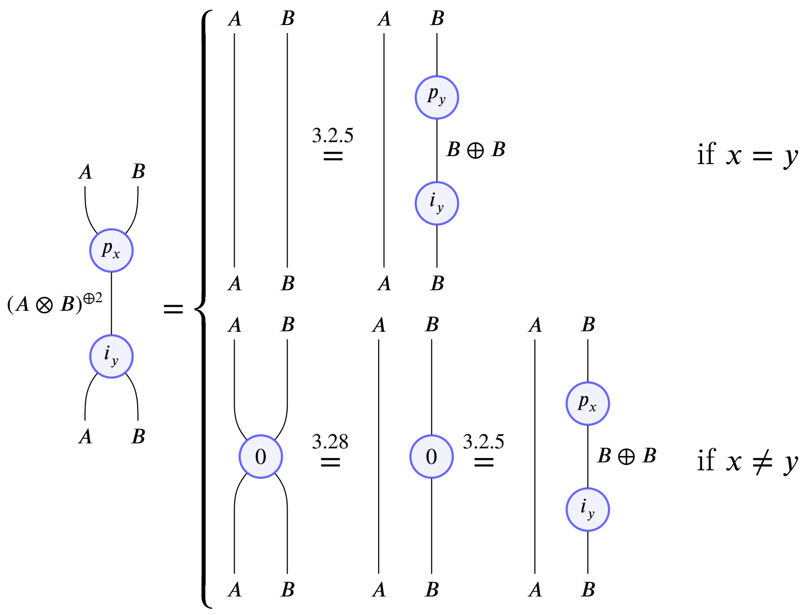
This allows us to prove the following lemma:

Lemma 3.4.3. *In a monoidal category with biproducts and a* 0*-object, we have:*



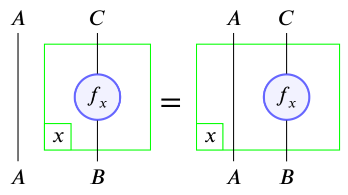


Proof. Indeed,

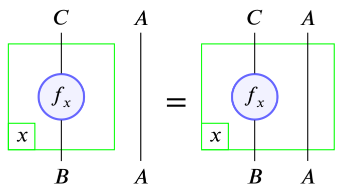


We proceed similarly to prove the second equation. ☐

Theorem 3.4.4 (Distributivity of the tensor product in the presence of biproducts)**.** *In a monoidal category with biproducts and a superposition rule, if A has either a right or a left dual then the tensor product distributes over the superposition rule.*

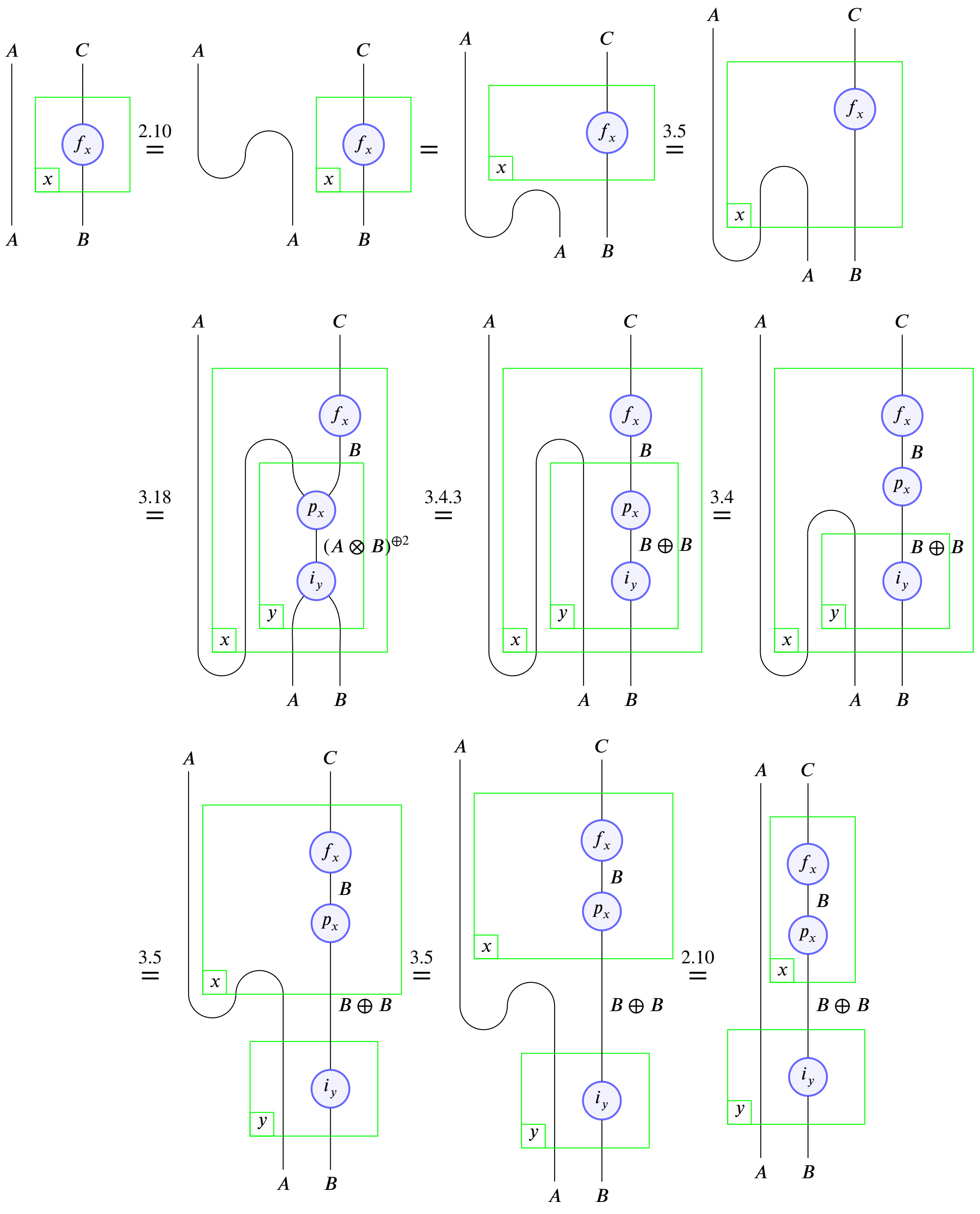


(3.30)

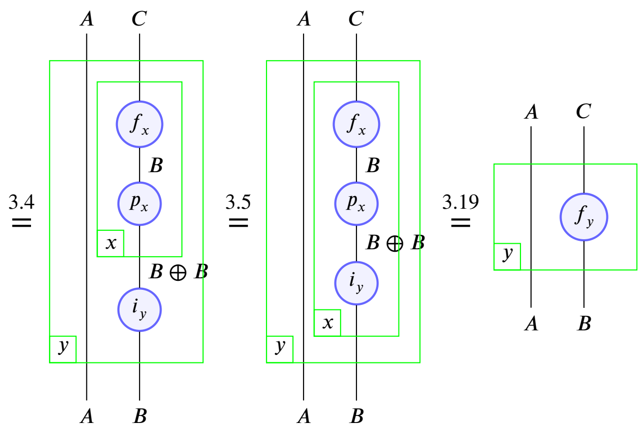


(3.31)

Proof. We will only prove 3.30 in the case where A has a right dual as the other cases are proved similarly.



A
C
f_x
x
B
2.10
3.5
3.18
3.4.3
3.4
p_x
i_y
y
$(A \otimes B)^{\oplus 2}$
$B \oplus B$



$\square$

In this proof we did not use any shortcut, every step precisely uses a result which has been introduced previously. This is in order to give a full account of how a proof-assistant would proceed to check the proof.

It is also interesting to notice we did not use the fifth axiom of biproducts (3.17) contrary to the proof in [7] so our requirement for $\mathscr{C}$ to have biproducts is in fact a bit too strong. We only need our objects A_1 and A_2 to have a corresponding object $A_1 \oplus A_2$ equipped with maps $p_x : A_1 \oplus A_2 \to A_x$ and $i_x : A_x \to A_1 \oplus A_2$ such that $p_x \circ i_y = \delta_{x,y}$.

Chapter 4

Implementing the superposition rule in a graphical proof-assistant

In this chapter we are interested in implementing the superposition rule in the graphical proof-assistant Globular. Globular is a graphical proof-assistant currently being developed by a team of researchers of the Department of Computer Science at the University of Oxford aiming at providing a tool handling higher-categorical diagrams.

In the first place, we shall describe a manner to handle monoidal categories suitable to computers. In particular we will describe the graph-based approach adopted by Globular.

Then we will describe two different approaches to implement the superposition rule in Globular. The first one will only allow the superposition of diagrams having the same structure whereas the second one, more expressive, will allow the superposition of any diagram whose boundaries match. To this effect, in each case we will describe the modified data types and algorithms of Globular and outline our contributions.

It is worth noting that while Globular can handle diagrams of semistrict n-categories, we will focus on strict 2-categories. Indeed, in our case we are only interested in modeling monoidal categories which can be modeled by bicategories, a particular notion of weak 2-categories. Moreover we do not want to distinguish between monoidal categories which are equivalent up to coherence isomorphisms hence our focus on strict 2-categories. Graphically this translates to the fact we do not want to distinguish between two string diagrams which are equivalent up to planar isotopy.

4.1 A way to model string diagrams suitable to computers

The approach adopted by Globular in order to model string diagrams is a graph-based approach. Indeed we shall see that 2-categories can be elegantly modeled by directed graphs. This is convenient as graphs are ubiquitous in computer science.

4.1.1 String diagrams as 2-categories

A monoidal category can be interpreted as a bicategory with one object. The 1-cells and the 2-cells of this category representing respectively the objects and the maps of the monoidal category. The horizontal composition of 2-morphisms being the tensor product of morphisms in monoidal categories.

According to this interpretation, a string diagram is a bicategory where regions are the object, strings are 1-cells whose sources and targets are objects and morphisms are 2-cells whose sources and targets are 1-cells.

We will relax the condition on the unique object of the bicategory in order to distinguish the different regions of a string diagram.

From now on we will only be interested in strict 2-categories for the previously mentioned reasons. Namely that we do not want to distinguish between two string diagrams which are equivalent up to planar isotopy. This translates to the fact a string diagram is uniquely determined by the way its components relate to one another. That is, a string diagram is uniquely determined by the source and target of its edges and vertices.

We should also add a restriction on morphisms. We are interested in globular k-morphisms (for $k \geq 1$) meaning the k-morphisms are morphisms between $(k-1)$-morphisms having the same domain and the same codomain. From now on, we will only refer to globular k-morphisms.

Now that we know how our string diagrams relate to strict 2-categories, we shall study how to concretely represent string diagrams.

4.1.2 A graph representation of string diagrams

We shall begin by exposing the concept of polygraph [6] to model strict n-categories. Indeed, Globular adopts a similar approach.

To understand the concept of polygraph, it suffices to notice we can consider a category as a directed graph with extra structure allowing the composition of arrows and the existence of identities. This directed graph represents the generators of the category. We can recover the category by building the free category over this graph.

The following graph represents the polygraph of an ordinary category with objects and morphisms. We define E_0 to be the set of objects, E_1 to be the set of generating morphisms and E_1^* to be the set of all morphisms built out of the generators in E_1:

$$\begin{tikzcd}
 & E_1 \arrow[dl, shift right, "s_0"'] \arrow[dl, shift left, "t_0"] \arrow[d, hook, "i_1"] \\
E_0 & E_1^* \arrow[l, shift right, "s_0^*"'] \arrow[l, shift left, "t_0^*"]
\end{tikzcd}$$

s and t are the functions mapping the morphisms to their source and target objects and i_1 is the obvious inclusion map.

We can generalize this construction to the case of n-categories and in particular to the case of 2-categories. This time we add the sets corresponding to the 2-cells and their corresponding functions.

$$\begin{tikzcd}
 & E_1 \arrow[dl, shift right, "s_0"'] \arrow[dl, shift left, "t_0"] \arrow[d, hook, "i_1"] & E_2 \arrow[dl, shift right, "s_1"'] \arrow[dl, shift left, "t_1"] \arrow[d, hook, "i_2"] \\
E_0 & E_1^* \arrow[l, shift right, "s_0^*"'] \arrow[l, shift left, "t_0^*"] & E_2^* \arrow[l, shift right, "s_1^*"'] \arrow[l, shift left, "t_1^*"]
\end{tikzcd}$$

The difference between polygraphs and the model we will use is that we are not interested in generating the entire free-category over the generators. We are only interested in modeling some composites made out of these generators. These composites will be the components of a specific string diagram.

Indeed, if we make the connection between the previous graph and string diagrams, we can see E_2^* as the set of vertices and of their composites by tensoring and by composition, E_1^* as the set of edges and of their composites by tensoring and E_0 as the set of regions.

Functions s and t map the different components of the diagram to their corresponding source and target.

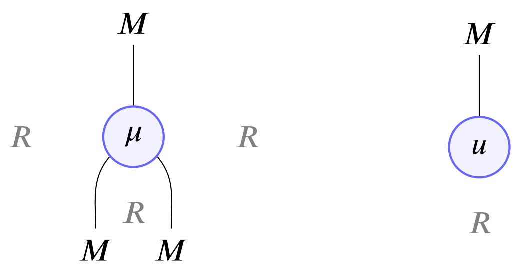
Finally the source and target functions must satisfy the globularity identities to give full account of the fact we are dealing with globular k-morphisms.

These equations are:

$$\forall i \in \mathbb{N}, \quad \begin{aligned} s_i \circ s_{i+1} &= s_i \circ t_{i+1} \\ t_i \circ s_{i+1} &= t_i \circ t_{i+1} \end{aligned} \tag{4.1}$$

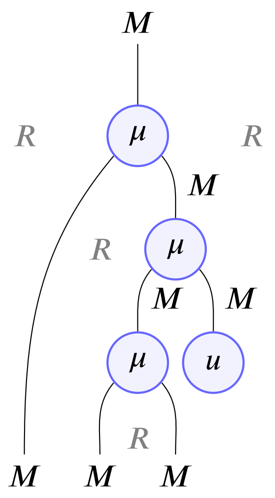
It remains to show how we can model a unique string diagram knowing its components and how they relate to one another.

Consider the following example. If an object M in a category is equipped with the structure of a monoid with a multiplication μ and a unit u, we can describe this data using the following generators:

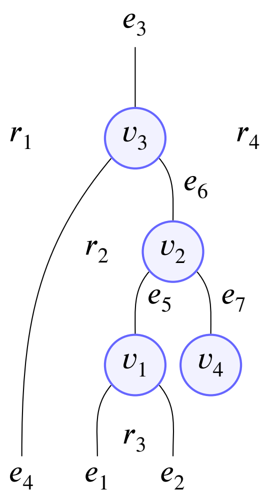


R is the name of the unique object of our 2-category. We colored the name of regions in gray not to confuse them with the type of the edges.

Then, from these generators we can build any diagram involving them such as this one:



When describing a diagram we have to distinguish its structure from its algebraic meaning. This means a description of a diagram is comprised of a description of its graphical components (vertices, edges, regions) and of a mapping attributing an algebraic meaning to these components.
For instance, the structure of the previous diagram is the following:



This structure just tells us how the different elements of the diagram relate to one another but has no algebraic meaning. By labelling each component uniquely, there is no ambiguity in how these components compose and the above diagram is the only one we can build from these components.

Then we recover the types of the original diagram by defining a function mapping the elements of the structure of the diagram to the generators they represent. In the case of our previous example, our function would be defined for each component (vertices, edges and regions) as follows:

$$
\begin{aligned}
f_v &= \{(v_1, \mu), (v_2, \mu), (v_3, \mu), (v_4, u)\} \\
f_e &= \{(e_1, \boldsymbol{M}), (e_2, \boldsymbol{M}), (e_3, \boldsymbol{M}), (e_4, \boldsymbol{M}), (e_5, \boldsymbol{M}), (e_6, \boldsymbol{M}), (e_7, \boldsymbol{M})\} \\
f_r &= \{(r_1, \boldsymbol{R}), (r_2, \boldsymbol{R}), (r_3, \boldsymbol{R}), (r_4, \boldsymbol{R})\}
\end{aligned}
$$

To summarize, to model a string diagram we need a graph of generators, a graph of the components of the diagram uniquely labelled instructing us on how these components relate to one another, and a function mapping these components to the generators they originate from.

4.2 A practical implementation

Now that we have outlined the principles underlying the handling of diagrams by Globular, we will describe its formal description and we shall expose which changes we have to bring in order to handle the superposition rule.

To this effect we will describe two different approaches, the first one being easier to implement but of a limited expressiveness and the second one requiring more changes to the existing framework but being more expressive.
The lesser expressive approach will only allow us to sum diagrams having the same diagrammatic structure. This limitation is not as serious as one could think. Indeed, we proved several theorems in the last chapter involving the sum of diagrams sharing the same diagrammatic structure.
On the other hand, the more expressive approach will allow us to sum diagrams sharing a different diagrammatic structure but having the same boundaries. By boundaries we refer to the type of the morphisms corresponding to the diagrams.

We remind that in this project we are not interested in keeping the semistrictness feature of Globular especially as it is incompatible with the second approach we will describe. Therefore we shall only describe the parts of the implementation of Globular relevant to our interests.

A last particularity of the following approach is that we only focus on the superposition of 2-cells. This restriction will not be reflected in our data structures for a question of genericity but we assume this is guaranteed by construction.

4.2.1 The weaker approach

This approach only requires minor changes to the existing framework. In this case a superposition of diagrams is a superposition of diagrams having the same diagrammatic structure. This method will not allow us to superpose different diagrams whose boundaries match but is easier to incorporate in Globular. Another advantage of this method is that we could keep the semistrictness of Globular. Nonetheless, we have chosen not to do so as it is not relevant to our interests.

4.2.1.1 The data types

The data types used to describe the diagrams are comprised of 3 structures: **Diag**, **Sig** and **Map** with **Diag** and **Sig** being mutually defined.

A diagram is described by a structure of type **Diag** combining a structure of type **Sig** describing its generators, a structure of type **Sig** describing the diagrammatic structure of the diagram and a mapping of type **Map** between the diagrammatic structure and the generators.

Sources and targets of generators of the type **Sig** are in turn described by a diagram of type **Diag**. As these types are dependent over an integer specifying the dimension of the generators and as the dimension always decreases, the recursion is well-founded.

- **Sig**
  The data type **Sig** describes the components of a diagram.
  This type depends on an integer $n \geq 0$. It indicates the dimension of the described morphisms.

  Definition 4.2.1 (**Sig**)**.** **Sig** depends on an integer $n \geq 0$.

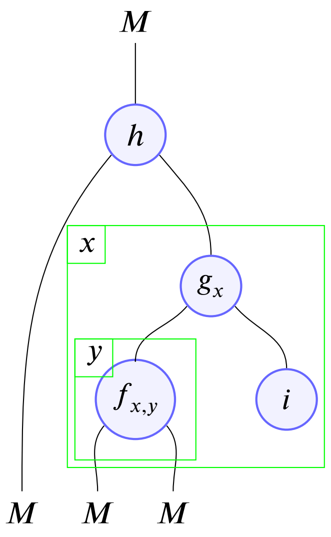
  $$\begin{aligned} \sigma &: \mathbf{Sig}(n-1) && \text{if } n > 0 \\ g &: \mathbf{Set} \\ s,t &: g \to \mathbf{Diag}(n-1,\sigma) && \text{if } n > 0 \\ h &: \mathbf{Tree}(g) \end{aligned}$$

  where **Tree** is a recursive and dependent type defined as follows:

  $$\mathbf{Tree}(g) = g \times \mathbf{Set}(\mathbf{int} \times \mathbf{Tree}(g))$$

g is a set of generators of dimension n. Generators of dimension $n > 0$ have a source and a target given by s and t and described by diagrams of dimension $n-1$. These diagrams are defined over the signature σ.
Our addition to the original structure **Sig** used in Globular is the tree structure h which is used to describe the superposition structure of the diagram. h exposes the hierarchy of superpositions and which generators they contain.

Consider the following example:



with $x \in [\![1,4]\!]$ and $y \in [\![1,3]\!]$.
In this case, if δ is the signature corresponding to the vertices of the diagram, we have:

$$\delta.h = (\{h\}, \{(4, (\{g, i\}, \{(3, (\{f\}, \{\}))\}))\})$$

Superpositions of diagrams are described by an integer corresponding to the number of diagrams forming the superposition and a node corresponding to the diagram itself.
Note that the signature does not give account of how the morphisms are indexed. It just tells us in which box are the morphisms. For example, the fact that i is in the x-indexed box but is not indexed by x will be reflected in the mapping we shall introduce soon.

- **Diag**

Diagrams such as the one described previously are built out of a set of generators we can compose according to their source and target.
Therefore the diagram type depends on a signature σ and on an integer n referring to the dimension of the described n-category.
This structure is the same as the one described in the original proposal of Globular.

Definition 4.2.2 (**Diag**)**.** The Diagram type depends on an integer n and on a signature of generators σ: **Sig**(n)

$$\begin{aligned} \delta &: \mathbf{Sig}(n) \\ s, t &: \mathbf{Diag}(n-1, \delta.\sigma) && \text{if } n > 0 \\ f &: \mathbf{Map}(n, \delta, \sigma) \end{aligned}$$

The embedded signature δ of the diagram refers to the structure of the diagram. It allows us to describe the actual layout of the diagram.

A n-diagram ($n > 0$) has a source diagram s and a target diagram t defined over the same signature but described in terms of $n-1$-diagrams. Intuitively this corresponds to the type of the morphism represented by the diagram.

The morphism f described in the next section will then allow us to map the components of δ to the generators they belong to which are described in σ.

- **Map**

As a diagram is built out of a set of generators we need a type mapping the components of the diagram to the corresponding generators. This type is defined recursively in order to map each k-morphisms with $k \in [\![0, n]\!]$ if we describe a n-category.

Definition 4.2.3 (**Map**)**. Map** is dependent on an integer $n \geq 0$, on a source signature σ and on a target signature τ.

$$\begin{aligned} f &: \sigma.g \to \mathbf{List(int)} \to \tau.g \\ m &: \mathbf{Map}(n-1, \sigma.\sigma, \tau.\sigma) \end{aligned}$$

f maps generators of $\sigma.g$ to generators of $\tau.g$.
Our addition to handle the superposition of morphisms is to use an higher-order

function taking an ordered list of integers corresponding to the indices indexing the generator we are considering.

If we consider the example we introduced earlier, $f(f, [2, 3])$ would correspond to $f_{2,3}$.

The fact that i is in the x-indexed box but is not indexed by x means that there is $y \in \tau.g$ such that for all $x \in [\![1, 4]\!]$ we have $f(i, [x]) = y$.

4.2.1.2 The algorithms

Now that we have a way to encode diagrams, we want to manipulate them. In particular, we want to be able to build new diagrams by attaching subdiagrams whose types match or be able to modify an existing diagram by rewriting parts of it. This means we want a way to pattern-match our diagrams so we can rewrite subdiagrams using our rewriting rules which is the whole purpose of a proof-assistant.

The details of these algorithms is taken from the original paper on Globular [1]. We have modified them consequently in order to handle the superposition of diagrams.

- **Match**

  One of the key operations we want to perform on diagrams is the matching operation. This operation establishes a bijection from one diagram to another if they are equivalent. This is one of the operations we need in order to apply our rewriting rules on diagrams. Indeed, we want to be able to identify parts of diagrams that match our rewriting rules so we can apply them and rewrite diagrams.

  The matching operation is defined as follows:

  $$\mathbf{Match}(\Delta_1 : \mathbf{Diag}(n, \Sigma),\ \Delta_2 : \mathbf{Diag}(n, \Sigma)) : \mathbf{Map}(n, \Delta_1, \Delta_2)$$

  This function takes two n-diagrams Δ_1 and Δ_2 defined over the same signature. If Δ_1 is equivalent to Δ_2, this operation establishes a bijection m : $\mathbf{Map}(n, \Delta_1, \Delta_2)$ from Δ_1 to Δ_2. m establishes a bijection of generators for each dimension of the described n-category.

  Its principle is similar to the one described in [1]. Every generators of $\Delta_1.\delta$ are either in the source or target of Δ_1 or in the source or target of one of its n-generators therefore **Match** first attempts to match the boundaries of the diagrams by recursive calls Match($\Delta_1.s$, $\Delta_2.s$) and Match($\Delta_1.t$, $\Delta_2.t$) then builds the unique bijection $b : \Delta_1.\delta.g \to \Delta_2.\delta.g$ between the set of generators $\Delta_1.\delta.g$ and $\Delta_2.\delta.g$. Then for all $x \in \Delta_1.\delta.g$, it calls Match($\Delta_1.\delta.s(x)$, $\Delta_2.\delta.s(b(x))$) and Match($\Delta_1.\delta.t(x)$, $\Delta_2.\delta.t(b(x))$).

In addition we have to check that the possible superpositions of diagrams are equivalent. To this effect, we check that $\Delta_1.\delta.h$ is equivalent to $\Delta_2.\delta.h$ using the bijection b we have previously constructed. This means for all $x \in \Delta_1.g$, x and $b(x)$ have to be contained in the same boxes and these boxes must be indexed the same way.

If any of these steps fails, the matching fails otherwise we return the mapping between these two diagrams.

- **Attach**

  Attach allows to build a diagram out of two other diagrams by attaching them along one of their matching boundaries. As a superposition of diagrams can only occur in the interior of a diagram, the operation of attaching two diagrams remains almost unchanged compared to the one described in [1].

  The attachment operation is defined as follows:

  $$\mathbf{Attach}(\Delta : \mathbf{Diag}(n, \sigma), \Psi : \mathbf{Diag}(n, \sigma), \mathrm{T} : \mathbf{Diag}(m, \sigma), d : \mathbb{B}) : \mathbf{Diag}(n, \sigma)$$

  The **Attach** operation returns a new diagram Δ' built out of two other diagrams Δ and Ψ. $\mathbf{T}$ is a subdiagram of Δ which corresponds to the m-boundary along which Ψ will be attached. The boolean d informs us if $\mathbf{T}$ is part of a source or a target boundary of Ψ.

  As $\mathbf{T}$ is part of a boundary of Δ and as it is isomorphic to a boundary of Ψ, we can attach the two diagrams by a double pushout of the two diagrams along $\mathbf{T}$. Then, the boundaries $\Delta'.s$ and $\Delta'.t$ are constructed by attaching the boundaries of Δ and Ψ. We proceed by recursive calls to $\mathbf{Attach}(\Delta.s, \Psi.s, \mathrm{T}, d)$ and $\mathbf{Attach}(\Delta.t, \Psi.t, \mathrm{T}, d)$.

  Finally, we merge the two structures $\Delta.\delta.h$ and $\Psi.\delta.h$. If we call p_1 and p_2 the first and second projections of the cartesian product forming h, we construct $\Delta'.\delta.h$ the following way:

  $$\Delta'.\delta.h = (p_1(\Delta.\delta.h) \cup p_1(\Psi.\delta.h), p_2(\Delta.\delta.h) \cup p_2(\Psi.\delta.h))$$

- **Rewrite**

  The rewrite operation is defined as follows:

  $$\textbf{Rewrite}(\Delta : \textbf{Diag}(n, \sigma),\ \Psi_1 : \textbf{Diag}(m, \Delta.\delta),\ \Psi_2 : \textbf{Diag}(m, \sigma)) : \textbf{Diag}(n, \sigma)$$

  The rewriting operation takes a n-diagram Δ to rewrite, a m-diagram Ψ_1 (with $m \leq n$) which must be defined over the generators of the structure of the diagram of Δ and a m-diagram Ψ_2. Then the procedure uses **Match** to construct a bijection between $\Psi_1.s$ and $\Psi_2.s$ and between $\Psi_1.t$ and $\Psi_2.t$. Then the interior of Ψ_1 is cut out from the occurrence of Ψ_1 in Δ and replaced by the interior of Ψ_2. $\Delta.f$ is updated accordingly.

  Finally, $\Delta.\delta.h$ is updated by removing the data relative to $\Psi_1.\delta.h$ and by adding the data of $\Psi_2.\delta.h$.

4.2.2 The stronger approach

In this section we will describe a stronger approach which will allow us to superpose different diagrams whose boundaries match. But we will have to make more changes to the existing structures than in the first case. We will allow diagrams to be made of other diagrams in addition to be made of generators. Indeed, we can see superpositions of diagrams as black boxes with a source and a target like ordinary generators. A mapping allows us to inspect the content of these superpositions by mapping these black boxes to a set of diagrams constituting the superposition.

We still have our three structures **Sig**, **Diag** and **Map** which serve the same purpose but this time we will have to introduce a fourth structure: **SSig**.

4.2.2.1 The data types

- **Sig**

  In this case the signature is a little simpler than previously, we do not need the tree structure anymore. In fact it is identical to the one defined in Globular.

Definition 4.2.4 (**Sig**)**.** The type $\mathbf{Sig}(n)$ depends on an integer $n \geq 0$.

$$\begin{aligned} \sigma &: \mathbf{Sig}(n-1) && \text{if } n > 0 \\ g &: \mathbf{Set} \\ s,t &: g \to \mathbf{Diag}(n-1,\sigma) && \text{if } n > 0 \end{aligned}$$

We can use this data type to describe generators which do not involve the superposition rule. This is not a disadvantage as any sum of morphisms originates from a sum of primitive morphisms. g refers to the generators of dimension n. s and t map the generators to their source and target diagrams.

- **Diag**

  The diagram type still depends on a signature σ and on an integer n referring to the dimension of the described n-category.

  Definition 4.2.5 (**Diag**)**.** The diagram type depends on a signature of generators σ: $\mathbf{Sig}(n)$

$$\begin{aligned} \delta &: \mathbf{SSig}(n,\sigma) \\ s,t &: \mathbf{Diag}(n-1,\sigma) && \text{if } n > 0 \\ f &: \mathbf{Map}(n,\delta,\sigma) \end{aligned}$$

  This time, the internal structure of a diagram is described by a different type than **Sig**. As we will see, this type is similar to **Sig** but can handle the sum of arbitrary diagrams whose boundaries match. We notice this type is dependent on the signature over which the diagram itself is dependent. Indeed, we want to allow our diagram to be made of other diagrams dependent on the same signature.

- **SSig**

  Definition 4.2.6 (**SSig**)**.** The type $\mathbf{SSig}(n,\delta)$ depends on an integer $n \geq 0$ and on a signature $\delta : \mathbf{Sig}(n)$.

$$\begin{aligned}
\sigma &: \mathbf{Sig}(n-1) && \text{if } n > 0\\
g &: \mathbf{Set}\\
s_g, t_g &: g \to \mathbf{Diag}(n-1, \sigma) && \text{if } n > 0\\
h &: \mathbf{Set}\\
s_h, t_h &: h \to \mathbf{Diag}(n-1, \sigma) && \text{if } n > 0\\
m &: h \to \mathbf{Set}(\mathbf{Diag}(n, \delta))
\end{aligned}$$

This structure has all the components of **Sig** allowing it to describe a regular diagram. In addition, we define a set of superposed diagrams h. These have a source and a target we can access through s_h and t_h. Finally, m allows us to access superpositions of diagrams corresponding to the elements of h.
These superpositions are defined as a set of diagrams. As we know, the superposition rule is associative therefore our set need not be ordered.
The motivation behind the dependence on δ is that superpositions of diagrams have to be defined over the same signature **Sig** as the diagram referring to the signature **SSig**. This is also why we had to create a new type, **SSig**. Indeed, without this new type, **Sig** would have to be dependent on itself hence giving birth to a case of Hofstadter's strange loop.

- **Map**

  In this case we use the unmodified structure from Globular. Indeed, each diagram (the main one or the ones part of superpositions) embed their own mapping.

  Definition 4.2.7 (Map)**.**

$$\begin{aligned}
f &: \sigma.g \to \tau.g\\
m &: \mathbf{Map}(n-1, \sigma.\sigma, \tau.\sigma)
\end{aligned}$$

4.2.2.2 The algorithms

As in the last section, we give a succinct description of the algorithms needed for this approach.

- **Match**

  The matching operation is defined as follows:

  $$\mathbf{Match}(\Delta_1 : \mathbf{Diag}(n, \Sigma),\ \Delta_2 : \mathbf{Diag}(n, \Sigma)) : \mathbf{Map}(n, \Delta_1, \Delta_2)$$

  If Δ_1 is equivalent to Δ_2, this operation establishes a bijection $m : \mathbf{Map}(n, \Delta_1, \Delta_2)$ from Δ_1 to Δ_2. m establishes a bijection of generators for each dimension of the described n-category.

  We remind the principle of this algorithm. Every generators of $\Delta_1.\delta$ are either in the source or target of Δ_1 or in the source or target of one of its n-generators therefore **Match** first attempts to match the boundaries of the diagrams by recursive calls $\mathbf{Match}(\Delta_1.s,\ \Delta_2.s)$ and $\mathbf{Match}(\Delta_1.t,\ \Delta_2.t)$ then builds the unique bijection $b : \Delta_1.\delta.g \rightarrow \Delta_2.\delta.g$ between the set of generators $\Delta_1.\delta.g$ and $\Delta_2.\delta.g$. Then for all $x \in \Delta_1.\delta.g$, it calls $\mathbf{Match}(\Delta_1.\delta.s_g(x),\ \Delta_2.\delta.s_g(b(x)))$ and $\mathbf{Match}(\Delta_1.\delta.t_g(x),\ \Delta_2.\delta.t_g(b(x)))$.

  In addition we have to check that the possible superpositions of diagrams are equivalent. To this effect, for all $x \in \Delta_1.\delta.h$ we establish the unique bijection b_h between $\Delta_1.\delta.h$ and $\Delta_2.\delta.h$. Then, for all $x \in \Delta_1.\delta.h$, we have to recursively match the sets of diagrams $\Delta_1.\delta.m(x)$ $\Delta_2.\delta.m(b_h(x))$ as well as the source boundaries $\Delta_1.\delta.s_h(x)$ and $\Delta_2.\delta.s_h(b_h(x))$ and the target boundaries $\Delta_1.\delta.t_h(x)$ and $\Delta_2.\delta.t_h(b_h(x))$.
  Our approach will be the naive one: we try to match each diagram of the first set with every diagrams of the second set until we find a good match or we have exhausted all the possibilities. In the latter case, the matching fails.

  If any of these steps fails, the matching fails otherwise we return the mapping between these two diagrams.

- **Attach**

  The attachment operation is defined as follows:

  $$\mathbf{Attach}(\Delta : \mathbf{Diag}(n,\ \sigma),\ \Psi : \mathbf{Diag}(n,\ \sigma),\ \mathrm{T} : \mathbf{Diag}(m,\ \sigma),\ d : \mathbb{B}) : \mathbf{Diag}(n,\ \sigma)$$

  The **Attach** operation returns a new diagram Δ' built out of two other diagrams Δ and Ψ. T is a subdiagram of Δ which corresponds to the m-boundary along which Ψ will be attached. The boolean d tells us if T is part of a source or target boundary of Ψ.

As $\mathbf{T}$ is part of a boundary of $\boldsymbol{\Delta}$ and as it is isomorphic to a boundary of $\boldsymbol{\Psi}$, we can attach the two diagrams by a double pushout of the two diagrams along $\mathbf{T}$. This step need only to be slightly altered to handle our superpositions of diagrams as we can see them as ordinary components of our diagram. Then, the boundaries $\Delta'.s$ and $\Delta'.t$ are constructed by attaching the boundaries of $\boldsymbol{\Delta}$ and $\boldsymbol{\Psi}$. We proceed by recursive calls to $\mathbf{Attach}(\Delta.s, \Psi.s, \mathrm{T}, d)$ and $\mathbf{Attach}(\Delta.t, \Psi.t, \mathrm{T}, d)$.

- **Rewrite**

  The rewrite operation is defined as follows:

  $$\mathbf{Rewrite}(\Delta : \mathbf{Diag}(n, \sigma),\ \Psi_1 : \mathbf{Diag}(m, \Delta.\delta),\ \Psi_2 : \mathbf{Diag}(m, \sigma)) : \mathbf{Diag}(n, \sigma)$$

  The rewriting operation takes a n-diagram $\boldsymbol{\Delta}$ to rewrite, a m-diagram $\boldsymbol{\Psi}_1$ (with $m \leq n$) which must be defined over the generators of the structure of the diagram of $\boldsymbol{\Delta}$ and a m-diagram $\boldsymbol{\Psi}_2$. Then the procedure uses **Match** to construct a bijection between $\Psi_1.s$ and $\Psi_2.s$ and between $\Psi_1.t$ and $\Psi_2.t$. Then the interior of $\boldsymbol{\Psi}_1$ is cut out from the occurrence of $\boldsymbol{\Psi}_1$ in $\boldsymbol{\Delta}$ and replaced by the interior of $\boldsymbol{\Psi}_2$.

  Once again the logic of the original algorithm should remain unchanged to handle the superpositions of diagrams. Indeed, the superpositions are part of the interior of the diagrams.

Chapter 5

Conclusion

In this thesis we introduced a diagrammatic representation of the superposition rule before studying some conditions leading to the existence of a superposition rule in a monoidal category.

This new graphical depiction of the superposition rule, allowing the representation of superpositions of string diagrams representing morphisms from a same hom-set, is the main contribution of this thesis. We tried to propose a depiction fitting consistently in the string diagram framework. This representation is especially suitable for a use in a diagrammatic proof-assistant as the interactivity provided by the graphical user interface could allow the user to iterate over the content of the boxes, what can not be done on paper.

Despite the limitations of a printed document, this box notation proves itself to be still useful to depict superpositions of diagrams having the same structure. Indeed, in this case we only have to expose the structure of the diagram once and we use a set of indices to put in relation the maps and their corresponding superposition. A good example of this technique is the proof of the distributivity of the tensor product over the superposition rule when biproducts exist under a certain condition which is another important contribution of this thesis.

Finally we proposed two different approaches to implement the superposition rule in Globular, a proof-assistant handling higher-categorical diagrams developed by a team of researchers from the Department of Computer Science. These two approaches vary in expressiveness and difficulty of implementation. However, their description remains quite succinct and it remains to concretely implement them. This could appear to be far from trivial for complexity reasons which have not been dealt with during their conception. This could make the object of further investigations.

Bibliography

[1] Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: a proof assistant for semistrict higher rewriting.

[2] Richard Garner and Daniel Schäppi. When coproducts are biproducts. 2015.

[3] Robin Houston. Finite Products are Biproducts in a Compact Closed Category. *Journal of Pure and Applied Algebra*, 212(2):394–400, 2008.

[4] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 1998.

[5] Daniel Marsden. Category Theory Using String Diagrams.

[6] Samuel Mimram. Towards 3-dimensional rewriting theory. *Logical Methods in Computer Science*, 10(2), 2014.

[7] Jamie Vicary and Chris Heunen. Lecture notes in Categorical Quantum Mechanics. Oxford University.